

ALIBABA CLOUD

阿里云

阿里云实时计算
Blink独享模式（公共云已停止新
购）

文档版本：20221124

 阿里云

法律声明

阿里云提醒您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。
<code>Courier</code> 字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
<i>斜体</i>	表示参数、变量。	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.新功能发布记录	20
1.1. Blink-3.7.10	20
1.2. Blink-3.7.9	20
1.3. Blink-3.7.7	20
1.4. Blink-3.6.8	20
1.5. Blink-3.6.5	21
1.6. Blink-3.6.2	22
1.7. Blink-3.6.0	22
1.8. Blink-3.5.0-hotfix	22
1.9. Blink-3.4.4	23
1.10. Blink-3.4.3	23
1.11. Blink-3.3.0	25
1.12. Blink-3.2.3	26
1.13. Blink-3.2.1	27
1.13.1. Blink-3.2.1新功能发布记录	27
1.13.2. Blink 3.2与Flink 1.5.1版本API兼容性报告	28
1.13.3. Blink 3.0与Blink 2.0版本SQL不兼容项汇总	30
2.产品概览	32
2.1. 概述	32
2.2. 发展历程	34
2.3. 业务流程	35
2.4. 管理控制台介绍	37
2.5. 支持的上下游存储	39
2.6. 产品安全	40
2.7. 使用限制	41
2.8. 独享模式系统架构（已停售）	42

3.产品定价	44
3.1. 计量项	44
3.2. 计费方式	44
3.3. 规格选择	47
3.4. 续费指导	48
3.4.1. 手动续费	48
3.4.2. 自动续费	49
3.5. 变配指导	50
3.6. 释放集群	52
4.准备工作	53
4.1. RAM用户授权	53
4.2. 开通服务和创建项目	54
4.3. 角色授权	58
4.3.1. 独享模式角色授权	58
4.3.2. 共享模式角色授权	59
5.Blink SQL参考	64
5.1. 概述	64
5.2. 关键字	64
5.3. 基本概念	66
5.3.1. 时区	66
5.3.2. 时间属性	68
5.3.3. Watermark	71
5.3.4. 计算列	72
5.4. 数据类型	72
5.4.1. 类型转换	73
5.4.2. 数学和逻辑运算	74
5.5. 创建数据视图	75
5.6. DDL语句	77

5.6.1. 概述	77
5.6.2. 创建数据源表	80
5.6.2.1. 数据源表概述	80
5.6.2.2. 创建Oracle数据库源表	83
5.6.2.3. 创建数据总线DataHub源表	86
5.6.2.4. 创建交互式分析Hologres源表	90
5.6.2.5. 创建日志服务SLS源表	93
5.6.2.6. 创建消息队列MQ源表	98
5.6.2.7. 创建消息队列Kafka源表	102
5.6.2.8. 创建表格存储Tablestore源表	117
5.6.2.9. 创建全量MaxCompute源表	119
5.6.2.10. 创建增量MaxCompute源表	124
5.6.3. 创建数据结果表	129
5.6.3.1. 数据结果表概述	129
5.6.3.2. 创建Oracle数据库结果表	129
5.6.3.3. 创建交互式分析Hologres结果表	132
5.6.3.4. 创建云原生数据仓库AnalyticDB MySQL版2.0结果表	136
5.6.3.5. 创建数据总线DataHub结果表	138
5.6.3.6. 创建日志服务SLS结果表	142
5.6.3.7. 创建消息队列MQ结果表	144
5.6.3.8. 创建表格存储Tablestore结果表	147
5.6.3.9. 创建云数据库RDS MySQL版结果表	149
5.6.3.10. 创建MaxCompute结果表	153
5.6.3.11. 创建云数据库HBase版结果表	159
5.6.3.12. 创建Elasticsearch结果表	163
5.6.3.13. 创建时序数据库结果表	167
5.6.3.14. 创建消息队列Kafka结果表	169
5.6.3.15. 创建云数据库HybridDB for MySQL结果表	172

5.6.3.16. 创建云数据库RDS SQL Server版结果表	174
5.6.3.17. 创建云数据库Redis版结果表	177
5.6.3.18. 创建云数据库MongoDB版结果表	181
5.6.3.19. 创建云原生数据仓库AnalyticDB MySQL版3.0结果表	182
5.6.3.20. 创建自定义结果表	184
5.6.3.21. 创建InfluxDB结果表	190
5.6.3.22. 创建Phoenix5结果表	192
5.6.3.23. 创建分析型数据库PostgreSQL版结果表	194
5.6.4. 创建数据维表	196
5.6.4.1. 概述	197
5.6.4.2. 创建交互式分析Hologres维表	198
5.6.4.3. 创建表格存储Tablestore维表	202
5.6.4.4. 创建云数据库RDS MySQL版维表	205
5.6.4.5. 创建云数据库HBase版维表	209
5.6.4.6. 创建MaxCompute维表	214
5.6.4.7. 创建云数据库Redis维表	220
5.6.4.8. 创建Elasticsearch维表	222
5.6.4.9. 创建Phoenix5维表	224
5.6.4.10. 创建云原生数据仓库AnalyticDB MySQL版3.0维表	227
5.6.4.11. 创建Oracle维表	231
5.7. DML语句	234
5.7.1. EMIT语句	234
5.7.2. INSERT INTO语句	237
5.8. QUERY语句	238
5.8.1. SELECT语句	238
5.8.2. WHERE语句	240
5.8.3. HAVING语句	241
5.8.4. GROUP BY语句	242

5.8.5. JOIN语句	243
5.8.6. 维表JOIN语句	245
5.8.7. IntervalJoin语句	247
5.8.8. UNION ALL语句	251
5.8.9. TopN语句	253
5.8.10. GROUPING SETS语句	258
5.8.11. 复杂事件处理（CEP）语句	259
5.8.12. 去重语句	265
5.9. 窗口函数	267
5.9.1. 概述	267
5.9.2. 滚动窗口	268
5.9.3. 滑动窗口	271
5.9.4. 会话窗口	275
5.9.5. OVER窗口	277
5.10. 内置函数	283
5.10.1. 字符串函数	283
5.10.1.1. CHAR_LENGTH	283
5.10.1.2. CHR	283
5.10.1.3. CONCAT	284
5.10.1.4. CONCAT_WS	285
5.10.1.5. FROM_BASE64	286
5.10.1.6. HASH_CODE	287
5.10.1.7. INITCAP	288
5.10.1.8. INSTR	288
5.10.1.9. JSON_VALUE	290
5.10.1.10. KEYVALUE	291
5.10.1.11. LOWER	292
5.10.1.12. LPAD	293

5.10.1.13. MD5	294
5.10.1.14. OVERLAY	295
5.10.1.15. PARSE_URL	296
5.10.1.16. POSITION	297
5.10.1.17. REGEXP	298
5.10.1.18. REGEXP_EXTRACT	299
5.10.1.19. REGEXP_REPLACE	300
5.10.1.20. REPEAT	302
5.10.1.21. REPLACE	303
5.10.1.22. REVERSE	304
5.10.1.23. RPAD	305
5.10.1.24. SPLIT_INDEX	306
5.10.1.25. STR_TO_MAP	307
5.10.1.26. SUBSTRING	308
5.10.1.27. TO_BASE64	309
5.10.1.28. TRIM	310
5.10.1.29. UPPER	311
5.10.2. 数学函数	312
5.10.2.1. 加	312
5.10.2.2. 减	312
5.10.2.3. 乘	313
5.10.2.4. 除	314
5.10.2.5. ABS	315
5.10.2.6. ACOS	315
5.10.2.7. BIN	316
5.10.2.8. ASIN	317
5.10.2.9. ATAN	318
5.10.2.10. BITAND	319

5.10.2.11. BITNOT	320
5.10.2.12. BITOR	320
5.10.2.13. BITXOR	321
5.10.2.14. CARDINALITY	322
5.10.2.15. CONV	323
5.10.2.16. COS	324
5.10.2.17. COT	325
5.10.2.18. EXP	325
5.10.2.19. E	326
5.10.2.20. FLOOR	327
5.10.2.21. LN	328
5.10.2.22. LOG	329
5.10.2.23. LOG10	330
5.10.2.24. LOG2	330
5.10.2.25. PI	331
5.10.2.26. POWER	332
5.10.2.27. RAND	332
5.10.2.28. SIN	333
5.10.2.29. SQRT	334
5.10.2.30. TAN	335
5.10.2.31. CEIL	336
5.10.2.32. CHARACTER_LENGTH	336
5.10.2.33. DEGREES	337
5.10.2.34. MOD	338
5.10.2.35. ROUND	339
5.10.3. 日期函数	340
5.10.3.1. LOCALTIMESTAMP	340
5.10.3.2. CURRENT_DATE	341

5.10.3.3. CURRENT_TIMESTAMP	341
5.10.3.4. DATEDIFF	342
5.10.3.5. DATE_ADD	343
5.10.3.6. DATE_FORMAT	344
5.10.3.7. DATE_SUB	345
5.10.3.8. DAYOFMONTH	346
5.10.3.9. EXTRACT	347
5.10.3.10. FROM_UNIXTIME	348
5.10.3.11. HOUR	349
5.10.3.12. LOCALTIME	350
5.10.3.13. MINUTE	351
5.10.3.14. MONTH	352
5.10.3.15. NOW	352
5.10.3.16. SECOND	353
5.10.3.17. TIMESTAMPADD	354
5.10.3.18. TO_DATE	356
5.10.3.19. TO_TIMESTAMP	357
5.10.3.20. UNIX_TIMESTAMP	358
5.10.3.21. WEEK	359
5.10.3.22. YEAR	360
5.10.4. 逻辑函数	360
5.10.4.1. =	361
5.10.4.2. >	361
5.10.4.3. >=	362
5.10.4.4. <=	363
5.10.4.5. <	364
5.10.4.6. <>	365
5.10.4.7. AND	365

5.10.4.8. BETWEEN AND	366
5.10.4.9. IS NOT FALSE	368
5.10.4.10. IS NOT NULL	369
5.10.4.11. IS NOT TRUE	369
5.10.4.12. IS NOT UNKNOWN	370
5.10.4.13. IS NULL	371
5.10.4.14. IS TRUE	372
5.10.4.15. IS UNKNOWN	373
5.10.4.16. LIKE	374
5.10.4.17. NOT	375
5.10.4.18. NOT BETWEEN AND	376
5.10.4.19. IN	378
5.10.4.20. OR	379
5.10.4.21. IS DISTINCT FROM	379
5.10.4.22. IS NOT DISTINCT FROM	380
5.10.4.23. NOT IN	381
5.10.5. 条件函数	382
5.10.5.1. CASE WHEN	382
5.10.5.2. COALESCE	384
5.10.5.3. IF	385
5.10.5.4. IS_ALPHA	386
5.10.5.5. IS_DECIMAL	387
5.10.5.6. IS_DIGIT	388
5.10.5.7. NULLIF	389
5.10.6. 表值函数	389
5.10.6.1. GENERATE_SERIES	389
5.10.6.2. JSON_TUPLE	390
5.10.6.3. STRING_SPLIT	391

5.10.6.4. MULTI_KEYVALUE	392
5.10.7. 类型转换函数	394
5.10.7.1. CAST	394
5.10.8. 聚合函数	395
5.10.8.1. AVG	395
5.10.8.2. CONCAT_AGG	396
5.10.8.3. COUNT	398
5.10.8.4. FIRST_VALUE	398
5.10.8.5. LAST_VALUE	401
5.10.8.6. MAX	404
5.10.8.7. MIN	404
5.10.8.8. SUM	405
5.10.8.9. VAR_POP	406
5.10.8.10. STDDEV_POP	407
5.10.9. 其他函数	408
5.10.9.1. UUID	408
5.10.9.2. DISTINCT	409
5.11. 自定义函数（UDX）	412
5.11.1. 概述	412
5.11.2. 自定义标量函数（UDF）	415
5.11.3. 自定义聚合函数（UDAF）	418
5.11.4. 自定义表值函数（UDTF）	423
5.11.5. 使用IntelliJ IDEA开发自定义函数	426
6.Blink SQL开发指南	431
6.1. 概述	431
6.2. 数据存储	431
6.2.1. 概述	431
6.2.2. 注册数据存储	434

6.2.2.1. 注册数据总线DataHub	434
6.2.2.2. 注册云原生数据仓库AnalyticDB MySQL版	436
6.2.2.3. 注册表格存储Tablestore	437
6.2.2.4. 注册云数据库RDS版	438
6.2.2.5. 注册日志服务SLS	439
6.2.3. VPC访问授权	440
6.2.4. 数据存储白名单配置	442
6.3. 作业开发	443
6.3.1. 开发	443
6.3.2. 上线	446
6.3.3. 启动	446
6.3.4. 暂停	447
6.3.5. 停止	448
6.4. 作业调试	448
6.4.1. 本地调试	448
6.4.2. 线上调试	452
6.5. 作业运维	454
6.5.1. 登录作业运维平台	454
6.5.2. 运行信息	454
6.5.3. 数据曲线	458
6.5.4. Timeline	463
6.5.5. Failover	463
6.5.6. Checkpoints	464
6.5.7. JobManager	465
6.5.8. TaskExecutor	466
6.5.9. 血缘关系	467
6.5.10. 属性参数	468
6.5.11. 作业诊断	469

6.6. 作业调优	470
6.6.1. 概述	470
6.6.2. 高性能Flink SQL优化技巧	472
6.6.3. AutoConf自动配置调优	480
6.6.4. AutoScale自动配置调优	488
6.6.5. 手动配置调优	493
6.6.6. 典型的反压场景及优化思路	499
6.6.7. SQL Tuning Advisor	502
6.6.7.1. Partitioned All Cache调优	502
6.6.7.2. MiniBatch/MicroBatch调优	503
6.6.7.3. Cache优化	504
6.6.7.4. Async优化	506
6.6.7.5. APPROX_COUNT_DISTINCT优化	508
6.6.7.6. Local-Global优化	510
6.6.7.7. ROW_NUMBER OVER WINDOW去重	511
6.6.7.8. Partial-Final优化	513
6.7. 监控报警	513
6.8. 自定义日志级别和下载路径	514
6.9. 管理独享集群Blink版本	517
6.10. 监控报警	519
7.Blink Datastream开发指南	520
7.1. 概述	520
7.2. 数据存储白名单配置	520
7.3. 自定义参数	522
7.4. 监控报警	523
7.5. 作业开发	524
7.6. 作业提交	526
7.7. 作业开发	528

7.8. Datastream示例	529
7.8.1. 读取DataHub数据示例	529
7.8.2. 读取Kafka数据示例	532
7.8.3. 读取DataHub数据写入阿里云HBase示例	534
7.8.4. 读取日志服务SLS示例	537
8.OpenAPI	540
8.1. 简介	540
8.2. OpenAPI概览	541
8.3. RAM资源授权	544
8.4. 调用方式	545
8.4.1. 签名机制	545
8.4.2. 请求结构	548
8.4.3. 返回结果	548
8.4.4. 公共参数	549
8.5. 实例	551
8.5.1. BatchGetInstanceRunSummary	551
8.5.2. GetInstance	553
8.5.3. GetInstanceResource	559
8.5.4. GetInstanceCheckpoint	561
8.5.5. GetInstanceConfig	571
8.5.6. GetInstanceDetail	578
8.5.7. GetInstanceExceptions	584
8.5.8. GetInstanceFinalState	586
8.5.9. GetInstanceMetric	587
8.5.10. GetInstanceRunSummary	592
8.5.11. ListInstance	595
8.5.12. ModifyInstanceState	610
8.5.13. GetInstanceHistoryAutoScalePlanContent	612

8.5.14. GetInstanceHistoryAutoScalePlanList	614
8.6. 文件夹	615
8.6.1. CreateFolder	615
8.6.2. DeleteFolder	617
8.6.3. GetFolder	618
8.6.4. ListChildFolder	619
8.6.5. MVFolder	621
8.7. 作业	622
8.7.1. CheckRawPlanJson	623
8.7.2. CommitJob	624
8.7.3. CreateJob	628
8.7.4. DeleteJob	630
8.7.5. GetJob	632
8.7.6. ListJob	635
8.7.7. OfflineJob	638
8.7.8. StartJob	640
8.7.9. UpdateJob	645
8.7.10. ValidateJob	650
8.7.11. CalcPlanJsonResource	654
8.7.12. UpdateAutoScaleConfig	656
8.7.13. GetJobLatestAutoScalePlan	658
8.7.14. GetRawPlanJson	659
8.8. Package	661
8.8.1. CreatePackage	661
8.8.2. DeletePackage	664
8.8.3. GetPackage	665
8.8.4. GetRefPackageJob	668
8.8.5. ListPackage	677

8.8.6. UpdatePackage	682
8.9. Queue	684
8.9.1. CreateQueue	684
8.9.2. DeleteQueue	685
8.9.3. BindQueue	687
8.9.4. UnbindQueue	688
8.9.5. GetClusterQueueInfo	690
8.9.6. ListProjectBindQueue	693
8.9.7. ListProjectBindQueueResource	695
8.9.8. UpdateQueue	697
8.10. Cluster	699
8.10.1. CreateCluster	699
8.10.2. CreateCellClusterOrder	701
8.10.3. DestroyCluster	703
8.10.4. ExpandCluster	704
8.10.5. ShrinkCluster	706
8.10.6. ListCluster	707
8.10.7. GetClusterDetails	711
8.10.8. GetClusterEngineVersions	715
8.10.9. GetClusterResource	717
8.10.10. ModifyMasterSpec	719
8.10.11. GetClusterMetrics	720
8.11. OpenAPI DEMO	723
8.12. Project	737
8.12.1. CreateProject	737
8.12.2. DeleteProject	739
8.12.3. GetProject	740
8.12.4. ListProject	743

8.12.5. UpdateProject	747
8.12.6. UpdateProjectConfig	749
9.常见问题	751
9.1. 计量计费	751
9.2. 基本概念	753
9.3. 如何使用	765
9.4. 分场景排错指引	778
9.5. 上下游存储	805
9.6. SQL常见问题	813
9.6.1. 语法检查报错	813
9.6.2. 调试报错	818
9.6.3. 上线报错	822
9.6.4. 启动报错	827
9.6.5. 运行Failover	832
10.最佳实践	853
10.1. 电商行业最佳实践	853
10.1.1. 电商场景实战之多类目成交总额管理	853
10.1.2. 电商场景实战之订单与销量统计	855
10.1.3. 电商场景实战之实时PV和UV曲线	863
10.1.4. 电商场景实战之营销红包	867
10.1.5. 电商场景实战之实时态势感知和订单地理分布	873
10.1.6. 电商场景实战之最新交易记录获取	876
10.2. 视频直播行业最佳实践	879
10.2.1. 视频直播解决方案之视频核心指标监控	879
10.2.2. 视频直播解决方案之直播数字化运营	884
11.相关协议	889
11.1. 实时计算服务等级协议（SLA）	889

1.新功能发布记录

1.1. Blink-3.7.10

本文为您介绍Blink 3.7.10版本的重大功能变更和主要修复缺陷。

版本重大功能变更

消息队列RocketMQ Connector升级底层客户端依赖版本，因此所有的type=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](#)。

主要缺陷修复

修复AutoScale的缺陷。

1.2. Blink-3.7.9

本文为您介绍Blink 3.7.9版本的重大功能变更和主要修复缺陷。

版本重大功能变更

增加维表Partitioned Join校验。

说明

- 对于普通Partitioned Join，如果上游是更新流，为了保证语义的正确性，上游Unique Key必须包含Shuffle Key。
- 在维表开启Partitioned Join时，可能发生热点问题。因此，引入了Bucket Partitioned Join解决该问题。但是对于Bucket Partitioned Join，上游不能是更新流。源表计算每条数据应该去的并发，然后打散到下游的某个Bucket里，维表侧会在Bucket并发上都缓存该条维表数据，达到热点数据打散的效果。

主要缺陷修复

修复由于GeminiKeyedStateHandle没有注册FileSegmentStateHandle，导致Checkpoint文件被误删的缺陷。

1.3. Blink-3.7.7

本文为您介绍Blink 3.7.7版本的重大功能变更和主要修复缺陷。

主要缺陷修复

- 修复作业Checkpoint所占存储空间过大的问题。
- 修复Checkpoint文件残留清理机制存在误删文件的问题。
- 修复Blink不能使用带Schema的Hologres物理表。

1.4. Blink-3.6.8

本文为您介绍Blink 3.6.8版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 优化时间序列数据库TSDB写入性能。
- 优化分析型数据库PostgreSQL写入性能。
- 调整表格存储Tablestore连接超时时间为20s。

主要缺陷修复

- 修复AutoScale运行期间出现作业失败无法恢复的问题。
- 修复分析型数据库MySQL版2.0结果表出现NullPointerException报错的问题。
- 修复使用RetracRank算子的作业出现ArrayIndexOutOfBoundsException报错的问题。
- 修复使用RetracRank算子的作业连续收到相同的retract消息时出现数据异常的问题。
- 修复Tablestore源表Tunnel全量转增量时出现数据丢失的问题。

1.5. Blink-3.6.5

本文为您介绍Blink 3.6.5版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 优化DataHub源表CPU使用率过高的问题。
- 优化维表的partitionJOIN功能，支持缓存策略为All的维表加载分区。此外，引入多维表异步并行优化，所有支持异步并行优化的维表可以在WITH参数中，设置partitionedJoin = 'true'，开启partitionJOIN功能。
- 日志服务SLS源表新增 startupMode参数。参数取值如下：
 - **TIMESTAMP**：每个Shard从指定时间开始消费。
 - **Earliest**：每个Shard从最早位置开始消费。
 - **Latest**：每个Shard从最新位置开始消费。
 - **Group_Offsets**：每个Shard优先从服务端保存的Checkpoint开始消费，必须指定ConsumerGroup。

 **说明** 仅当State中不存在Checkpoint时，Earliest、Latest、Group_Offsets和Timestamp的配置才生效。

- 降低大规模作业拓扑的内存消耗。
- Oracle源表的timeFieldType参数支持多种时间格式：
 - **TO_DATE**：DATE类型。
 - **TIMESTAMP**：TIMESTAMP类型。
 - **VARCHAR**：日期字符串类型。
 - **NUMBER**：数字类型。
 - 实现OracleSourceQueryCondition接口并配置类名。

主要缺陷修复

- 修复在一个Task Manager存在多个线程的情况下，开启partitionJOIN功能后，MaxCompute维表存在

错误数据的缺陷。

- 修复Sink DDL field type与实际插入数据类型不一致时出现报错的缺陷。
- 修复SLS源表不消费数据的缺陷。

1.6. Blink-3.6.2

本文为您介绍Blink 3.6.2版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 优化AutoScale功能：没有流量的作业会触发Scale Down，减少资源浪费。
- 优化资源Rescale。
- 优化SLS Connector解析数据的方法：使用FastLogGroup方法解析数据。
- 降低大规模作业拓扑的内存消耗。

主要缺陷修复

- 修复Blink无法消费表格存储Tablestore源表积压数据的缺陷。
- 修复维表Cache ALL多并发导致NullPointerException的缺陷。
- 修复MaxCompute Sink动态分区无法正常提交数据的缺陷。

1.7. Blink-3.6.0

本文为您介绍Blink 3.6.0版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 新增Connector：
 - 新增创建分析型数据库PostgreSQL版结果表。
 - 新增创建Phoenix5维表。
 - 新增创建交互式分析Hologres源表。
 - 新增创建交互式分析Hologres维表。
 - 新增创建交互式分析Hologres结果表。
- 新增功能：
 - MQ结果表CSV格式支持写入KEY，二进制格式不支持写入KEY。
 - 支持Gemini 2.0（Gemini 增强版）。

主要缺陷修复

- 修复Kafka 08版本结果表，语法检查报错：Kafka dont have sufficient arguments 。
- 修复OTS源表因为某个分区一直没有数据，线上运行源表节点导致的Failover： java.lang.NullPointerException 。
- 修复OTS维表以注册存储方式引用，Token过期导致的Failover： OTSNoPermissionAccess, [Message]: You have no permission to access the requested resource, please contact the resource owner 。

1.8. Blink-3.5.0-hotfix

本文为您介绍Blink 3.5.0版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 新增 [创建InfluxDB结果表](#)。
- 新增 [创建Phoenix5结果表](#)。
- 升级DataHub Connector SDK版本。

主要缺陷修复

- 修复使用全局ORDER BY但语法检查没有报错的缺陷。
- 修复底层VARCHAR类型隐式转换为整型，行为错误的缺陷。
- 修复因calcite bug，数据按照时间分组进行聚合，导致计算结果异常的缺陷。
- 修复语法检查DataHub源表产生 `java.lang.NoClassDefFoundError: com/aliyun/datahub/client/auth/Account` 报错的缺陷。

1.9. Blink-3.4.4

本文为您介绍Blink 3.4.4版本的重大功能变更和主要修复缺陷。

版本重大功能变更

Blink-3.4.4版本支持MaxCompute源表监听新分区。在DDL语句中设置 `subscribeNewPartition` 参数，默认是false。当参数值是true时，不指定partition参数，会动态监听新分区。详情参见 [创建全量MaxCompute源表](#)。

主要缺陷修复

- RDS连接性问题：Blink使用Druid链接池链接RDS，优化Druid链接池链接RDS的方式，避免链接长时间不用导致作业的Failover。
- DataHub结果表：修复因一条数据中某一列为NULL，造成作业报错 `Caused by: java.lang.NullPointerException` 的缺陷。
- DataHub结果表：修复将BIGINT类型数据插入至字段为TIMESTAMP类型的DataHub结果表中，产生类型转换异常的缺陷。

1.10. Blink-3.4.3

本文为您介绍Blink 3.4.3版本的重大功能变更和主要修复的缺陷。

版本重大功能简介

GeminiStateBackend是Blink开发的新一代后台，它使用自研的存储引擎GeminiDB。通过线上部分作业的测试显示，GeminiStateBackend的性能可以达到NiagaraStateBackend性能的1.5倍。

GeminiStateBackend的主要特性如下：

- 采用LSM + Hash索引的存储模型。其中LSM提高写性能，Hash索引存放在内存中以弥补LSM读放大的缺点。简单来说，GeminiDB将文件划分为更小粒度的单元（Page），以Page为粒度进行冲洗和压缩处理，Hash索引根据键快速定位其所在的单元，从而减少I/O次数，提高读性能。
- 优化Cache策略。如果新插入的数据或压缩后的数据存在热点，GeminiDB会将其缓存在内存中。而传统的LSM实现会首先将这些数据刷到磁盘上，导致新增数据需要至少经过一次读I/O才能进入缓存，因此缓

存命中率下降。

- 优化数据冲洗到磁盘的策略。GeminiDB只有在内存的缓存填满后才会将部分数据冲洗到磁盘，因此在内存充足并且压缩及时的情况下，不会产生数据文件。而传统LSM的实现中，会将数据冲洗到磁盘以便进行持久化处理。这在Blink的应用场景中是没有必要的，Blink具有原生的Checkpoint机制以保证数据的一致性，因此只需在Checkpoint时进行持久化处理。
- 支持In-memory Compaction。对于内存中数据及时进行压缩，减少写放大以及压缩的读I/O。
- 使用Java消除RocksDB/Niagara中的JNI开销。
- 支持增量Checkpoint。
- 支持Local Recovery，使作业失效后快速恢复。
- 支持存储计算分离，使作业重启或者Rescale后的快速恢复，功能还在完善中。

GeminiStateBackend针对DataStream和SQL的作业配置如下：

- **DataStream作业**

- API配置方式

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
GeminiStateBackend stateBackend = new GeminiStateBackend(checkpointDir);
// Configuration for gemini
Configuration config = new Configuration();
config.setString("state.backend.gemini.heap.size", "1024mb");
// set configuration to backend
stateBackend.setConfiguration(config);
// use GeminiStateBackend as state backend
env.setStateBackend(new GeminiStateBackend(checkpointDir));
```

- 主要配置项

配置项	类型	单位	默认值	说明
state.backend.gemini.ttl.ms	LONG	ms	-1（未开启）	可选。数据存活时间。
state.backend.gemini.heap.size	STRING	支持以下字节单位： <ul style="list-style-type: none"> ▪ 1024 ▪ 1024kb ▪ 1024mb ▪ 1024gb 	无	可选。单个GeminiDB能够使用的内存大小。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ? 说明 推荐进行配置，否则backend会根据JVM和TaskManager的配置计算默认值。 </div>

- **SQL作业**


```
# 使用Gemini。
state.backend.type=gemini
# State的TTL的大小。
state.backend.gemini.ttl.ms=129600000
# GeminiDB允许使用的内存大小，单位MB。注意：Operator配置的内存资源要包含这部分，默认为512MB。
state.backend.gemini.heap.size.mb=512
# 推荐的TaskManager的JVM参数。
blink.job.option=-yD env.java.opts.taskmanager='-XX:NewRatio=3 -XX:SurvivorRatio=3 -XX:ParallelGCThreads=8 -XX:+UnlockDiagnosticVMOptions -XX:ParGCCardsPerStrideChunk=4096 -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=75 -Djdk.nio.maxCachedBufferSize=10240'
```

主要缺陷修复

- 修复Calc的codegen代码可能会出现NPE的缺陷。
- 修复状态存储必须存储完整行的缺陷。
- 修复 code split 缺陷：JavaCodeSplitter在把 local variable 抽取成 member field 时，没有替换 foreach control 语句中的 local variable 。在 distinct with filter 场景中可能出现这种缺陷。

1.11. Blink-3.3.0

本文为您介绍实时计算公告内容，包括版本重大功能发布和产品变更介绍。

版本重大功能

- 小CU模式
 - 新增初始plan设置CU预期：生成plan时，作业的初始并发度会根据用户设置的CU预期进行缩放。
 - 新增小CU模式资源配置：当用户设置的CU预期很小时，如果每个Vertex并发都设置为最小值1时，总的CU数还是超过了用户设置的CU，则会触发小CU模式，将多个vertex的subtask放到一个slot（即线程）中运行。同时将所有Vertex调度到一个TaskManager上，减少框架资源开销。
- DataHub
 - 写入DataHub支持通过hashFields的配置，使相同列值的记录写入同一个下游Shard。
 - 修复Blink 3.2版本读取closed shard的DataHub数据Failover，且任务不能恢复的问题。

产品变更

- Auto Scale变更
 - Autoscale最大资源设置语义修改：Max CU限制从作业整体资源的上限调整为Plan的资源上线，以解决Max CU限制，导致作业无法启动的问题。Max CU限制调整后，作业实际使用的资源可能会超过Max CU。
 - Autoscale添加可选作业参数配置，可以手动关闭scale down功能，保证作业稳定性。配置项：healthmanager.resource.scale.down.enabled（资源scale down开关）和 healthmanager.parallelism.scale.down.enabled（并发度scale down开关）。
 - 新增支持Datastream作业手动设置资源，开启Autoscale。从Blink3.3.0版本开始，Datastream作业支持手动编辑资源Plan，同时允许开启Autoscale。Datastream的Autoscale目前仅处于试用阶段。
- FirstRow on Rowtime

支持您按照Rowtime字段读取FirstRow，去重后保持time字段的Rowtime属性，即您可以在去重节点后继续使用Window。

- SQL文件大小写变更

该改动将忽略SQL文件中的大小写（大小写不敏感），导致部分作业在编译时报错。该错误表明您在同一段代码中，使用大小写来区分变量或标识符。

- 示例

```

4 t.taobao_bind AS taobao_bind,
5 t.et_taobao_bind AS et_taobao_bind
6 FROM
7 view_t_partner_map_taobao_bind t,
8 lateral table (STRING_SPLIT(t.utdids, '\004')) AS T(utdid0);
9
10 INSERT INTO

```

- 报错信息

```

5 org.apache.flink.table.api.ValidationException:
6 *****
7 ERR_ID:
8     SQL-00120001
9 CAUSE:
10    SQL validation failed:
11    From line 3, column 9 to line 3, column 69: Duplicate relation name 'T' in FROM clause
12 ACTION:
13    Please see descriptions above. If it doesn't help, please contact customer support for
14    this.
15    DETAIL:
16    .....

```

1.12. Blink-3.2.3

为了给您带来更好的开发交互体验，实时计算发布了Blink-3.2.3版本，本文为您介绍版本优化点和修复的问题。

优化

- 优化Vertex Topology页面与作业资源配置界面展示不一致问题。
- 优化3.2.1版本中数据曲线页面不显示Task ID的问题。
- 优化运维界面TaskManager.log日志中乱码问题。
- 优化3.2.1版本输出的GC日志会覆盖正常的调试结果的输出的问题。
- 优化独享集群AutoScale需要增加作业参数才能开启问题。
- 优化作业运行代码为LEFT JOIN，运维界面Vertex拓扑图中显示为INNER JOIN问题。
- 优化代码编辑有误时不定位至具体的一行问题。

问题修复

- REGEXP_EXTRACT函数参数为null或正则表达式不合法时，返回null。
- 开发代码中使用反斜线的正则表达式后，使用用分号编译报错。
- 上线调试运行结果打印至taskmanager.out中，与实际不符。
- LEFT JOIN维表，JOIN时无维表标识不报错。
- Time类型的数据写入ADS结果表出现异常。
- 3.2.1版本MaxCompute维表JOIN时，ON条件的字段数据类型为TIMESTAMP，作业运行报错。
- 3.2.1和3.2.2版本使用 minibatch 作业参数，作业运行报错。
- CEP语法检测正常情况下，作业运行时仍然报错。
- 云数据库HBase结果表的 maxRetryTimes 参数未生效。

- Kafka源表读取Kafka端的数据时，未显示对应的Group ID。
- 只写入一个VARBINARY类型的数据至MQ，运行报错。

1.13. Blink-3.2.1

1.13.1. Blink-3.2.1新功能发布记录

本文为您介绍Blink-3.2.1版本的核心功能以及新版本对于Datastream和SQL的兼容性。

版本核心功能介绍

Blink-3.2.1是Blink正式开源后第1个基于开源代码的正式版本。Blink-3.2.1版本核心功能介绍如下：

- Job AutoScale

Blink-3.2.1版本中加入了AutoConfig和AutoScale相结合的自动调优功能。自动调优功能能够在作业运行过程中，根据作业运行状态以及输入流量，自动调整各个算子的并发和资源，从而保证作业的低延时。在Blink-3.2.1版本中，该功能处于公测阶段。

- 支持Datastream API

- Blink-3.2.1版本中正式加入了对Datastream API的支持。Blink-3.2.1基于开源Flink 1.5.1分支开发，Blink-3.2.1中Datastream API接口与开源Flink1.5.1版本兼容性对比请参见[Blink 3.2与Flink 1.5.1版本API兼容性报告](#)。
- Datastream Connector新增。

对接系统	Source	Sink
Kafka	兼容	兼容
HBase	不兼容	
JDBC		
RDS<MySQL>		
ES		
MongoDB		

- SQL Connector新增

Blink-3.2.1在Blink2.0版本基础上新增了如下Connector。

对接系统	Connector类型
ES	DIM
MongoDB	SINK
Redis	DIM
Redis	SINK

对接系统	Connector类型
消息队列for Apache RocketMQ (4.2.0)	SOURCE
SQL Server	SINK

兼容性

- Datastream兼容性请参见[Blink 3.2与Flink 1.5.1版本API兼容性报告](#)。
- SQL兼容性请参见[Blink 3.0与Blink 2.0版本SQL不兼容项汇总](#)。

1.13.2. Blink 3.2与Flink 1.5.1版本API兼容性报告

本文为您介绍Blink 3.2与Flink 1.5.1版本API兼容性测试的结果。

校验范围

- flink-clients
- flink-core
- flink-java
- flink-java8
- flink-optimizer
- flink-scala
- flink-scala-shell
- flink-streaming-java
- flink-streaming-scala
- flink-yarn
- flink-connectors
- flink-filesystems
- flink-formats
- flink-metrics
- flink-queryable-state
- flink-state-backends

兼容性详情

- flink-core

总计方法个数：6126。不兼容数量：1。

序号	严重程度（高、中或低）	Old API	Change	Effect
----	-------------	---------	--------	--------

序号	严重程度（高、中或低）	Old API	Change	Effect
1	中	GenericCsvInputFormat.supportsMultiPaths()	supportsMultiPaths()方法从该类中删除，返回的默认值发生变化，新的默认值不支持multiPath。	所有GenericCsvInputFormat子类涉及multiPath的功能会受到影响。

• **flink-connector-elasticsearch**

总计方法个数：14；不兼容数量：1。

序号	严重程度（高、中或低）	Old API	Change	Effect
1	中	ElasticsearchSink	父类从ElasticsearchSinkBase<T>变为了ElasticsearchSinkBase<T,org.elasticsearch.client.Client>。	Flink 1.5.1不兼容ElasticsearchSinkBase<T>父类的子类。

flink-json

总计方法个数：34；不兼容数量：1。

序号	严重程度（高、中、低）	Old API	Change	Effect
1	中	JsonSchemaConverter	类名重命名成了JsonRowSchemaConverter。	子类不兼容。

flink-streaming-java

总计方法个数：3031。不兼容数量：4。

序号	严重程度（高、中、低）	Old API	Change	Effect
1	中	TwoInputStreamOperator.processElement1 or TwoInputStreamOperator.processElement2	返回类型从void变成TwoInputSelection。TwoInputStreamOperator中新增endInput1和endInput2 abstract方法。	Blink 3.2不兼容所有的TwoInputStreamOperator实现，而Flink 1.5.1兼容所有的TwoInputStreamOperator实现。

序号	严重程度（高、中、低）	Old API	Change	Effect
2	中	OneInputStreamOperator类	添加endInput() abstract方法。	不兼容所有的OneInputStreamOperator实现。
3	中	StreamOperator类	添加requireState abstract方法。	不兼容所有的StreamOperator实现。
4	中	OneInputStreamOperator类	添加了endInput() abstract方法。	不兼容所有的OneInputStreamOperator实现。

1.13.3. Blink 3.0与Blink 2.0版本SQL不兼容项汇总

SQL语法变更

- 语法变动
 - [Over Agg] The window rank function without order by
- 行为变动
 - [Division to double] | 隐式类型转换，从2.2版本开始生效。
 - [Decimal] DDL decimal type default precision changed to (10, 0) | Decimal的默认精度变更，从2.2版本开始生效。
 - [CEP] pattern不可以贪婪匹配结束。例如，不支持pattern (a b+)。可以通过将(a b+)转换为(a b+ c)，并把c定义为not b的方式，解决不能以贪婪匹配结束的问题。
 - [CEP] within语句不支持动态窗口。

开发接口变更

- 重构和语义变更
 - [StreamTableSink#emitDataStream returns values changes from void to StreamTableSink]
- Class位置挪动
 - [Parser 继承 com.alibaba.blink.streaming.connectors.common.source.SourceCollector]
 - [Class not found] com/alibaba/blink/exceptions/NotEnoughParamsException
 - [Class not found] com/alibaba/blink/exceptions/UnsupportedTableException
 - [Class not found] org/apache/flink/table/sources/BatchExecTableSource
 - [Class not found] org/apache/flink/table/functions/aggfunctions/DoubleSumWithRetractAggFunction
 - [Class not found] org/apache/flink/table/functions/Monotonicity
 - [Class not found] Lcom/alibaba/blink/cache/Cache
 - [Class not found] org/apache/flink/table/row/GenericRow
- 实现变更

- [Method not found] com.alibaba.blink.table.api.RichTableSchema.getColumnTypes
- [Method not found] Lorg/apache/flink/table/types/DataType.of
- [Verification] java.lang.VerifyError: class com.koubei.blink.connector.sls.CustomTableFactory overrides final method setClassLoader
- [Class not found] com.aliyun/odps/OdpsException

Connectors

- [ODPS] ODPSTableSink stream mode do not support overwrite
- [ODPS] Only batch mode support overwrite

2. 产品概览

2.1. 概述

阿里云实时计算Flink版独享/共享集群（原产品线）支持共享模式和独享模式两种产品模式。独享模式是基于共享模式的补充，具备更加丰富的功能。本文为您介绍实时计算共享模式和独享模式的优势及区别。

共享模式（已停购）

不同用户共享计算集群的物理资源（网络、磁盘、CPU或内存等），通过账号管理、CGroup（Control Groups）等方式进行资源隔离和安全管理。基于账号安全、业务安全和数据安全方面的考虑，共享模式不提供自定义函数功能。

🔍 说明 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算独享模式或Flink半托管模式。

独享模式

- 独享模式优势

独享模式是指在阿里云云服务器ECS（Elastic Compute Service）上单独为用户创建的独立计算集群。单个用户独享计算集群的物理资源（网络、磁盘、CPU或内存等），与其它用户的资源完全独立。与共享模式相比，独享模式具有以下优点：

- 多种硬件均可适配

实时计算独享集群可以充分复用阿里云在CPU、MEM配比、GPU或FPGA等硬件层面的各类优化，为您解决各类硬件适配问题。

- 用户间的隔离

如果使用ECS独享集群，您能够使用专有网络VPC和独享计算资源。既能满足您对专网专用、资源独享的需求，也能够与您的开发平台连通，满足您的业务需求。

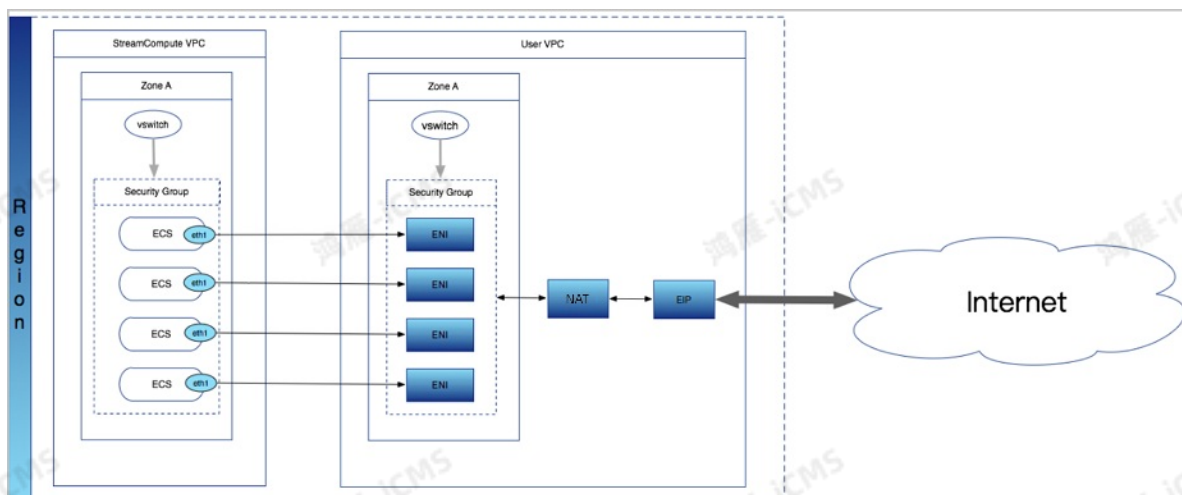
- 支持自定义函数

独享模式在网络及物理机层面与其它用户完全的隔离，支持自定义函数和底层API，满足您的业务需求。自定义函数详情，请参见[概述](#)。

- 丰富的功能

- Data Lake场景下的ETL：通过Flink SQL+UDF的方式，使ETL任务开发更加便利。
- 异构数据源计算：支持从异构数据源读取数据做分析。例如，从对象存储服务（Object Storage Service, OSS）远程读取归档日志数据，并关联云数据库HBase中的高危IP，完成网络攻击分析。
- 支持更加丰富的上下游数据存储，例如 [创建消息队列Kafka源表](#) 和 [创建消息队列Kafka结果表](#)。

- 独享模式系统网络架构



- 实时计算独享模式为全托管模式。您所购买的ECS托管在实时计算VPC下，暂不提供用户登录接口。
- 您可以在创建集群时，在您账号下申请弹性网卡，通过弹性网卡，您可以访问其VPC下所有资源。
- 如果需要访问公网，可在弹性网卡上绑定NAT网关及弹性公网IP，具体操作步骤请参见 [绑定EIP](#)。

说明 不访问公网时，弹性网卡不会产生任何额外费用。

- 如果您需访问VPC内其它安全组的服务，请配置相应安全组的规则。

独享模式与共享模式区别

类别	独享模式	共享模式
自定义功能开发	支持UDF和API等功能，作业开发更加灵活。	不支持
网络类型	支持阿里云VPC网络。 说明 独享模式集群仅能访问同一VPC、同一Region下的上下游存储资源。如果您需要访问其他VPC下的资源，请配置安全组的端口号，通过高速通道等方式打通网络。	不支持
机器类型选择	您可以在购买实时计算独享模式服务时，选择机器类型。	不支持
支持数据源类型	<ul style="list-style-type: none"> • 数据总线（DataHub） • 日志服务（Log Service） • 消息队列（MQ） • 消息队列（Kafka） 说明 更多实时计算数据源表介绍，请参见 数据源表概述 。	<ul style="list-style-type: none"> • 数据总线（DataHub） • 日志服务（Log Service） • 消息队列（MQ） 说明 更多实时计算数据源表介绍，请参见 数据源表概述 。

类别	独享模式	共享模式
支持数据输出类型	<ul style="list-style-type: none"> 分析型数据库 (AnalyticDB) 数据总线 (DataHub) 日志服务 (Log Service) 消息队列 (MQ) 表格存储 (Table Store) 云数据库 (RDS/DRDS) 高性能时间序列数据库 (TSDB) 云数据库 (HybridDB for MySQL) Kafka 云数据库 (HBase) ElasticSearch (ES) <p>? 说明 更多实时计算数据结果表介绍，请参见数据结果表概述。</p>	<ul style="list-style-type: none"> 分析型数据库 (AnalyticDB) 数据总线 (DataHub) 日志服务 (Log Service) 消息队列 (MQ) 表格存储 (Table Store) 云数据库 (RDS/DRDS) 高性能时间序列数据库 (TSDB) 云数据库 (HybridDB for MySQL) <p>? 说明 更多实时计算数据结果表介绍，请参见数据结果表概述。</p>
支持地域	<ul style="list-style-type: none"> 独享模式（按量付费）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）。 独享模式（包年包月）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）、华北3（张家口）、中国（香港）、新加坡。 	<p>华南1（深圳）</p> <p>? 说明 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算独享模式或Flink半托管模式。</p>
形态	每个用户在独立的ECS集群上，搭建独立的实时计算引擎。	大集群，共同使用公共资源池。
隔离性	隔离性强。使用ECS隔离网络、安全组等资源，与其他用户的资源相互独立。	隔离性较弱。无法隔离网络等资源。
面向客户群体	具备较强开发技术，对开发的灵活性和可控性具备需求的大数据团队。	仅需实现流式业务，对开发过程无特殊需求的团队或个人。
成本	相对共享模式，独享模式成本较高。高可用模式下每个用户需单独承担额外的3台Master机器的费用。	仅需支付计算服务费用。
集群管理和运维	需要具备集群管理、网络配置等技能。	无需管理集群。

2.2. 发展历程

本文为您介绍阿里云实时计算的发展历程。

阿里云实时计算在原有Flink系统基础上，提供一整套的开发平台和完整的流式数据处理业务流程。受益于阿里巴巴大数据多年的技术和业务沉淀，您在使用实时计算时，可以享受到阿里巴巴集团前沿的计算引擎能力，规避阿里巴巴集团多年在流式大数据业务上的试错结果，更快、更轻松地享受流式计算对大数据业务发展的助力。

- 起源：脱胎于双十一实时大屏业务

实时计算脱胎于阿里巴巴集团的双十一实时大屏业务。实时计算团队起初仅负责支持双十一大屏展现和部分实时报表业务，在历经多年的摸索和发展后，最终成长为独立稳定的云产品团队。实时计算定位将阿里巴巴集团本身沉淀多年的实时计算产品、架构、业务能够以云产品的方式对外提供服务，助力更多中小企业实时化自身大数据业务。

- 萌芽：以开源Flink作为基础

最初，阿里巴巴集团支撑双十一大屏等业务同样采用开源的Flink作为基础系统支持，并在上面开发Flink代码。这个时期的实时业务，处于萌芽阶段，规模尚小。数据开发人员使用Flink原生API开发流式作业，开发门槛高，系统调试难，存在大量重复的人力工作。

- 发展：基于Flink的API开发

阿里巴巴集团的工程师针对这类大量重复的工作，开始考虑进行业务封装和抽象。工程师们基于Flink的API，开发出大量可复用的数据统计组件。例如，实现了简单过滤、聚合、窗口等等作为基础的编程组件，并基于这类组件提供了一套XML语义的业务描述语言。基于这套设计，流式计算用户可以使用XML语言将不同的组件进行拼装描述，最终完成一整套完整的实时计算处理流程。基于XML+Flink组件的编程方式，从底层上避免了用户大量的重复开发工作，同时也降低了使用门槛。但我们的数据分析人员仍然需要熟悉整套编程组件和XML描述语法，这套编程方式离分析人员最熟悉的SQL方式仍然相差甚远。

- 成熟：Flink SQL开发完成

任何技术的发展一定遵循小众创新到大众普及的成长轨迹。而从小众到大众，从创新到普及的转折点，在于技术的功能成熟和成本降低。阿里巴巴工程师开始思考如何更大程度地降低数据分析产品的门槛，从而普及到更多的用户。得益于用户群对关系型数据库几十年的沉淀，阿里巴巴工程师最终开发了Flink SQL替换了原有的XML+组件的编程方式，使用经典的SQL模式能够完成计算和数据处理。这套系统成为今天实时计算的核心计算引擎（Flink）。当前这套系统已有单机群数千台机器的规模，日均消息处理数万亿级，流量近PB级，成为阿里巴巴集团核心的流式计算集群。

Flink SQL的优势：

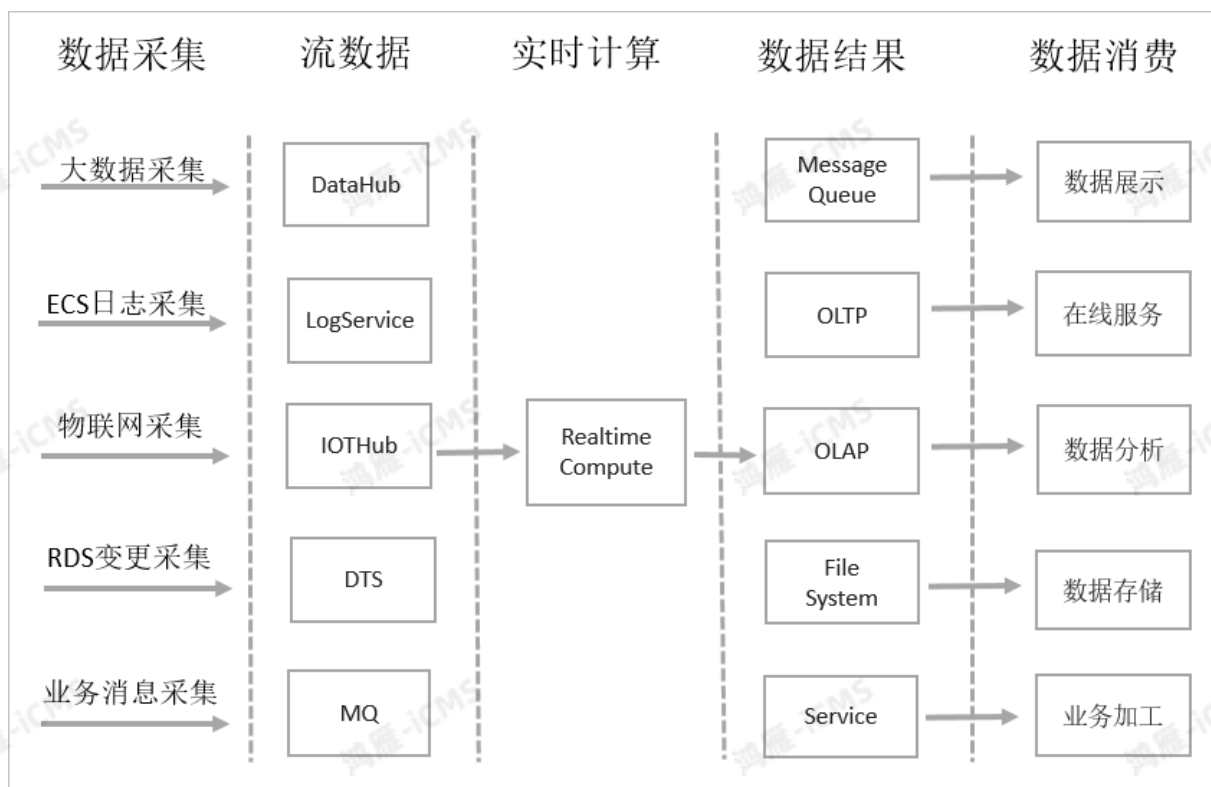
- Flink SQL对标SQL功能，可以提高您和开发人员的技术成熟度。
- 您可以在Flink SQL运用您熟悉的SQL模型，降低您使用实时计算的难度。

2.3. 业务流程

本文为您介绍阿里云实时计算业务流程的系统架构和数据链路。

业务流程简介

实时计算业务流程系统架构图如下。



1. 数据采集

广义的实时数据采集，是指使用流式数据采集工具，将数据实时地采集并传输到大数据Pub/Sub（发布订阅）系统。Pub/Sub系统将为下游实时计算提供源源不断的事件源，触发流式计算作业的运行。阿里云大数据生态提供了针对不同场景领域的流式数据Pub/Sub系统。阿里云实时计算天然集成上图中诸多的Pub/Sub系统，能够集成各类流式数据。

说明 例如，您可以直接使用实时计算对接日志服务（LogService）的LogHub系统，快速的集成并使用ECS日志。

2. 流式计算

流数据作为实时计算的触发源，驱动实时计算运行。一个实时计算作业至少使用一个流数据作为数据源。对于复杂的业务场景，实时计算支持和静态数据存储进行关联查询。

说明 例如，针对DataHub流式数据，实时计算可以根据流式数据的主键，和RDS中数据进行关联查询（即JOIN查询）。

3. 实时集成

阿里云实时计算可以将计算的结果数据直接写入目的数据存储。阿里云实时计算天然集成了OLTP（例如RDS）、NoSQL（例如OTS）、OLAP（例如ADB）、MessageQueue（例如DataHub、ONS）、MassiveStorage（例如OSS、MaxCompute）等阿里云生态系统，最大程度地降低全链路数据的时延和数据链路的复杂度，保证数据加工的实时性。

4. 数据消费

流式计算的结果数据进入各类数据存储后，您可以运用个性化的应用，操控结果数据。例如使用数据存储系统访问数据，使用消息投递系统接受信息，或使用告警系统生成异常结果数据警报。

数据链路

部分阿里云生态外部数据存储不能和实时计算系统完全匹配，需要使用其它类型流数据进行转换。

- DataHub

DataHub提供了多类数据（如日志、数据库BinLog、IoT数据流等）上传到DataHub的工具、界面，以及开源、商业软件的集成。请参见[DataHub使用文档](#)对DataHub做更详尽了解。

- LogService

日志服务（LogService）是针对日志类数据的一站式服务。LogService提供了诸多针对日志的采集、消费、投递、查询分析等功能。请参见[数据采集概述](#)，了解如何使用日志进行流式数据采集。

- IoTHub

阿里云物联网平台（IoTHub）是能够帮助开发者搭建安全的数据通道，方便终端（如传感器、执行器、嵌入式设备或智能家电等等）和云端的双向通信。使用IoTHub规则引擎，可以将IoT数据方便投递到DataHub，并利用实时计算和MaxCompute进行数据加工计算。请参见[设置数据流转规则](#)，了解如何将IoT数据推送到DataHub。

- DTS

数据传输（DTS）支持以数据库为核心的结构化存储产品之间的数据传输。DTS是一种集数据迁移、数据订阅及数据实时同步于一体的数据传输服务。使用DTS的数据传输功能，可以方便您将RDS等BinLog解析并投递到DataHub，并利用实时计算和MaxCompute进行数据加工计算。请参见[从RDS MySQL同步至DataHub](#)，了解如何使用DTS进行流式数据采集。

- MQ

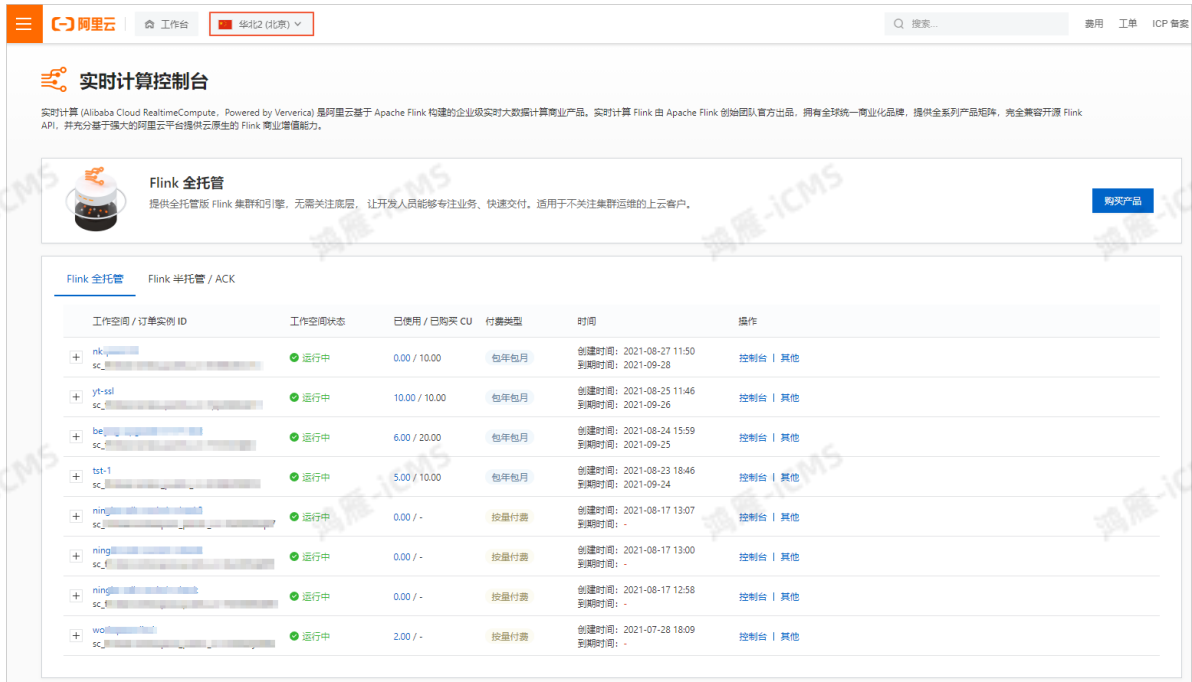
阿里云MQ服务是一套完整的消息云服务。阿里云MQ服务基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）、监控报警等功能。

2.4. 管理控制台介绍

本文为您介绍如何登录Flink全托管管理控制台，以及续费、工作空间和项目空间资源变配、修改虚拟交换机、添加项目空间和查看工作空间详情入口。

登录管理控制台

1. 登录[实时计算控制台](#)。
2. 单击管理控制台，进入Flink全托管管理控制台。
3. 在页面左上角，选择目标区域。



功能及操作入口

功能	操作入口
控制台	在Flink全托管页签，单击目标工作空间操作列下的控制台，即可进入Flink全托管开发控制台，进行业务开发和运维工作。
订单续费	在Flink全托管页签，选择目标工作空间操作列下的其他 > 订单续费，即可选择续费时长，选中服务协议，确认订单。
工作空间资源变配	在Flink全托管页签，选择目标工作空间操作列下的其他 > 工作空间资源变配，即可填写变配后的资源数量，选中服务协议，单击确认。
修改虚拟交换机	在Flink全托管页签，选择目标工作空间操作列下的其他 > 修改虚拟交换机，即可在可用交换机列表中，选中目标交换机，单击确认。 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> <p>说明 如果可用交换机列表中没有可用交换机，您可以先创建虚拟交换机，然后刷新此页面。虚拟交换机的创建步骤，请参见创建和管理交换机。</p> </div>
工作空间详情	在Flink全托管页签，选择目标工作空间操作列下的其他 > 工作空间详情，即可查看工作空间名称、工作空间ID、专有网络名和虚拟交换机等信息。
添加项目空间	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，填写项目空间名称和所需资源，单击确认。
项目空间资源分配	在Flink全托管页签，选择目标工作空间操作列下的其他 > 项目空间资源分配，即可滑动 <input type="checkbox"/> 选择变配后的资源，单击确认。

2.5. 支持的上下游存储

实时计算支持丰富的上下游生态。

- 数据源表
 - 创建Oracle数据库源表
 - 创建数据总线DataHub源表
 - 创建日志服务SLS源表
 - 创建交互式分析Hologres源表
 - 创建消息队列MQ源表
 - 创建消息队列Kafka源表
 - 创建表格存储Tablestore源表
 - 创建全量MaxCompute源表
 - 创建增量MaxCompute源表
- 数据结果表
 - 创建云原生数据仓库AnalyticDB MySQL版2.0结果表
 - 创建交互式分析Hologres结果表
 - 创建Oracle数据库结果表
 - 创建数据总线DataHub结果表
 - 创建日志服务SLS结果表
 - 创建消息队列MQ结果表
 - 创建表格存储Tablestore结果表
 - 创建云数据库RDS MySQL版结果表
 - 创建MaxCompute结果表
 - 创建云数据库HBase版结果表
 - 创建Elasticsearch结果表
 - 创建时序数据库结果表
 - 创建消息队列Kafka结果表
 - 创建云数据库HybridDB for MySQL结果表
 - 创建云数据库RDS SQL Server版结果表
 - 创建云数据库Redis版结果表
 - 创建云数据库MongoDB版结果表
 - 创建云原生数据仓库AnalyticDB MySQL版3.0结果表
 - 创建分析型数据库PostgreSQL版结果表
 - 创建自定义结果表
 - 创建InfluxDB结果表
 - 创建Phoenix5结果表
- 数据维表
 - 创建交互式分析Hologres维表

- 创建表格存储Tablestore维表
- 创建云数据库RDS MySQL版维表
- 创建云数据库HBase版维表
- 创建MaxCompute维表
- 创建云数据库Redis维表
- 创建Phoenix5维表
- 创建云原生数据仓库AnalyticDB MySQL版3.0维表
- 创建Elasticsearch维表

2.6. 产品安全

实时计算支持整体全链路实时计算的安全，包括账号安全，业务安全以及数据安全。

账号安全分为实时计算账号安全和数据存储账号安全：

- 实时计算账号安全

实时计算账号当前仅支持阿里云账号体系（包括登录用户名+密码或签名密钥）。传输链路使用HTTPS协议，保证全链路的用户账户安全。实时计算账号安全详情，请参见[RAM用户授权](#)。

- 数据存储账号安全

针对将数据存储保存到连接账号的问题，实时计算提供基于RAM或STS的两种连接方式。避免您因为账户信息丢失导致的业务信息泄露。数据存储账号安全详情，请参见[共享模式角色授权](#)和[独享模式角色授权](#)。

业务安全

实时计算的业务安全部分主要包含项目隔离安全和业务流程安全：

- 项目隔离安全

实时计算对不同的项目进行了严格的项目权限区分。不同用户或项目无法访问或操作项目内的所有子产品实体。项目级别的资源隔离能够保证您与其他用户在使用资源时不会相互干扰。

② 说明 例如用户A的作业在运行期间，由于数据量的突增，提升了该作业对CPU的使用率。实时计算底层的虚拟化技术对资源进行隔离，保证了该场景下仅用户A的CPU使用率提升，不会影响到其它用户作业的CPU使用状况。

- 业务流程安全

实时计算对于流式计算开发进行了严格的流程定义，区分了数据开发和数据运维。保证了整体业务流程的完整和安全性。

- 提供代码版本

支持代码版本回滚和对比。方便您进行代码追溯、比对、排错。

- 提供IDE单机调试容器

避免代码线下运行影响线上真实数据。您可以自行构造线下输入表、维表、输出表相关数据，不影响线上生产作业。

- 提供发布流程

发布流程避免了线下代码的改动对生产运行的影响。线下代码调试完成后，通过上线作业将作业提交到数据运维系统。已运行的实时计算作业并不会直接使用新代码，需经过您确认后，停止已运行作业并使用新代码重新运行。从流程上保证发布的严谨性。

数据安全

数据安全分为实时计算系统数据安全和业务数据安全。

- 系统数据安全

实时计算系统保证自身数据安全：

- 使用HTTPS访问方式，确保传输链路的安全。
- 数据存储使用AES高强度加密方式连接信息，避免敏感信息泄露。
- 实时计算系统通过全面且深入的攻击测试。
- 阿里云安全团队为实时计算提供安全服务。

- 业务数据安全

实时计算不负责存储用户的业务数据安全。业务数据安全交由不同的阿里云存储系统管理，具体的业务数据安全机制请通过不同数据存储的安全模型以及最佳安全实践进行了解。

2.7. 使用限制

本文介绍实时计算所支持的服务范围和相应的限制，包括CU处理能力的限制、项目创建的限制。


请您仔细评估下列限制对于您业务的影响情况。

- 实时计算共享模式暂不提供自定义函数（UDX）功能。如果需要支持UDX功能（参见 [概述](#)），请使用实时计算独享模式。
- **实时计算控制台** 仅支持Chrome浏览器访问。

支持地域

实时计算当前支持地域如下。实际支持地域以实时计算 [购买页面](#) 为准：

- Flink半托管（基于ACK）：华南1（深圳）、华北2（北京）、华东2（上海）、华北3（张家口）、华东1（杭州）。
- Blink独享/共享集群（原产品线）：
 - 独享模式（按量付费）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）。
 - 独享模式（包年包月）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）、华北3（张家口）、中国（香港）、新加坡。
 - 共享模式：华南1（深圳）

 **说明** 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算独享模式或Flink半托管模式。

CU处理能力

实时计算当前在内部压测场景下，一个CU的处理能力估算如下：

- 简单业务：例如单流过滤、字符串变换等操作，1CU每秒可以处理10000条数据。
- 复杂业务：例如JOIN、窗口、GROUP BY等操作，1CU每秒可以处理1000到5000条数据。

作业、任务数量限制

实时计算对整个项目（Project）下属的作业、Task版本、IDE打开Task页面数量均有不同限制。包括：

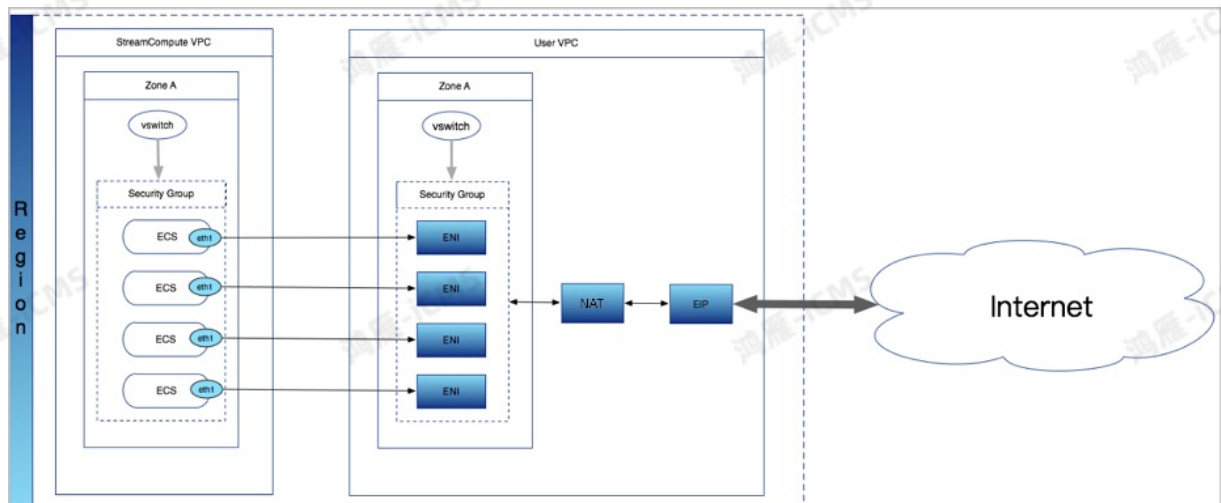
- 单个项目下允许最多创建作业的个数为100。
- 单个项目下允许最多的文件夹的个数为50，层级最大不超过5层。
- 单个项目下允许最多的UDX或JAR个数为50。
- 单个项目下允许最多注册数据存储的个数为50。
- 单个作业允许最多的历史保存版本数为20。

2.8. 独享模式系统架构（已停售）

本文为您介绍实时计算独享模式系统架构。

架构介绍

下图为独享模式系统架构。



- 实时计算独享模式为全托管管理方式。您所购买的ECS都托管在实时计算VPC下，暂不提供登录方式。
- 为了访问您的VPC内服务，会在创建集群时在您账号下申请弹性网卡。您可以通过弹性网卡访问VPC下所有资源。
- 如果您的实时计算集群需要访问公网，可以在弹性网卡上绑定NAT网关及弹性公网IP，详情请参见 [绑定ECS实例](#)。
- 实时计算为您申请的弹性网卡存放于您账号下一个单独的安全组内。如果您需要访问VPC内其它安全组的服务，请配置该安全组的规则。

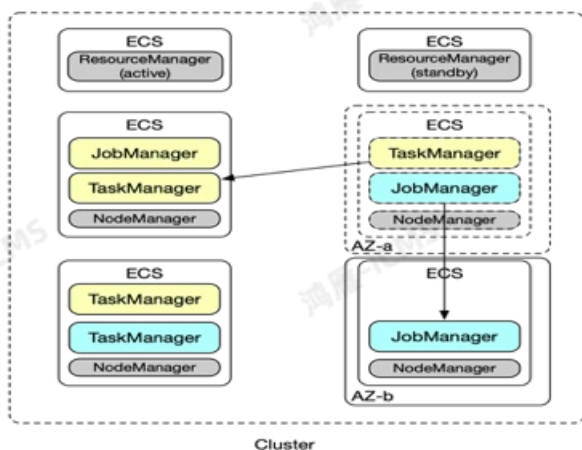
说明 在您不访问公网的情况下，弹性网卡不会产生任何额外费用。

产品特点

- 支持全链路实时计算开发

- 提供基于Flink SQL的实时数据处理能力，支持故障场景的自动恢复，保证异常场景仍然可以准确处理数据。
- 支持多种内置函数，例如字符串、日期和聚合函数。
- 支持多种窗口类型，例如滚动、滑动和会话窗口。
- 计算资源控制精确，保证作业的隔离性。
- 以下关键性能指标高于开源Flink：
 - 数据计算的延迟可以达到亚秒级。
 - 单个作业吞吐量可以达到百万条记录/秒，单集群规模达到数千台。
- 深度整合各类云数据存储。方便您直接读写数据总线DataHub、日志服务SLS、云数据库RDS版、表格存储TableStore和分析型数据库MySQL版等数据存储系统。
- 完全托管的实时计算服务
 - 采用完全托管的流式计算引擎。
 - 无需预置或管理任何基础设施，即可运行和查询流数据。
 - 支持一键启用流式数据服务。
 - 试用和迁移流式计算成本小，集成存储数据、开发数据、运维数据和监控报警等功能。
 - 为不同租户间的托管运行服务提供有效的隔离和全面防护。
- 节省人力和集群成本
 - 大幅优化SQL执行引擎，计算作业比原生Flink作业更经济高效。
 - 开发和运行成本远低于开源流式框架。
- 高可用性

当实时计算产品底层ECS异常或者作业发生Failover时，JobManager及TaskManager可以在同可用区的ECS恢复使用，实现作业高可用性；也可以在不同可用区或地域的ECS恢复使用，实现跨可用区高可用性。



3. 产品定价

3.1. 计量项

本文为您介绍实时计算计量项。

实时计算的基本计量单位为Compute Unit（CU），即计算资源，1CU=1核CPU+4 GB内存。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1CU的处理能力可以通过以下方式估算：

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

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，示例如下：
 - 如果实时计算需要从日志服务（Log Service）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允许的范围之内。
 - 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4CU。

3.2. 计费方式

本文为您介绍实时计算计费规则。

重要 当项目欠费后有停机风险，系统会提醒您，请及时续费，避免对您的服务造成影响。

重要

- 当项目欠费后有停机风险，系统会提醒您，请及时续费，避免对您的服务造成影响。
- 包年包月到期后请及时续费，如果未及时续费，到期7天后您的实例会被释放。
- 包年包月，售卖5天内，可以无条件退余款，超过5天，非产品问题不予退款。

共享模式（已停购）

说明 实时计算共享模式已于2019年12月24日正式下线，将不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。若有新购需求，推荐使用实时计算独享模式或Flink云原生模式。

根据CU数进行计费，每个CU每天的价格是6元人民币。例如，您购买10CU，使用1个月（假设当月天数为30天）的费用为1800元人民币。实时计算共享模式的最小起售单位为2CU每天。详情参见[正式商业公告](#)。

独享模式包年包月（已停购）

说明 独享模式包年包月已于2021年4月28日暂停新购，目前仅支持原有项目的扩缩容和续费操作。如果您有新购需求，推荐使用实时计算Flink全托管。

独享模式包年包月根据购买的机型及数量进行计费，各机型单价如下。


重要 实际价格请以产品购买页面为准。

机型 (Master/Slave)	价格 (单位: 元/月)
4核16GB	850
8核32GB	1450
16核64GB	2710
24核96GB	3950
32核128GB	5180
56核224GB	8780
64核256GB	9950

独享模式集群配置建议请参见[规格选择](#)，您也可以通过[实时计算控制台](#)中价格计算器的方案推荐功能选择集群配置，价格计算如下图所示。



独享模式按量付费（已停购）

 **说明** 独享模式按量付费已于2021年4月28日暂停新购，目前仅支持原有项目的扩缩容和续费操作。如果您有新购需求，推荐使用实时计算Flink全托管。

独享模式按量付费是先使用后付费的计费方式。独享模式按量付费方式下，您可以按需取用资源，随时开启或释放资源，无需提前购买大量资源。与购买主机的传统投入方式相比，按量付费方式的成本更低。独享模式按量付费以小时作为计费周期，各种机型的按量付费价格如下。

 **重要** 实际价格请以产品购买页面为准。

机型	计费周期	价格（元/（台*小时））
4核16GB	小时	2.23
8核32GB		3.84
16核64GB		7.06
24核96GB		9.23
32核128GB		13.5
56核224GB		23.16
64核256GB		23.59

● **计费时间点：**

- 计费开始：单击创建集群。
- 计费结束：集群释放完成。

请在集群使用完毕后，及时释放集群（参见 [释放集群](#)），避免产生额外费用。

● **开通条件：**开通按量付费的实时计算资源时，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。

● **计费周期：**

- 按量付费资源从单击创建集群开始计费，到释放结束计费，按小时计费，以1个整点为1个计费周期（均以UTC+8时间为准）。
- 整点结束后，进入新的计费周期。
- 如果账号可用余额小于上1周期的账单金额，您会收到提醒余额不足的短信或邮件。

计费周期示例如下（同一天内）：

- 如果01:30:34单击创建集群，01:55:20释放所有资源，则1:00:00~2:00:00为1个计费周期，实际计费时长为1小时。
- 如果01:50:03单击创建集群，02:50:02释放所有资源，则01:00:00~02:00:00为第1个计费周期，实时计算资源按1小时计费；02:00:00~03:00:00为第2个计费周期，实际计费时长共为2小时。
- **变更配置后计费：**变更配置后会产生1个新的订单按新配置的价格计费，旧订单自动失效。如果您在1个计费周期内变更了实例规格，将按照当前计费周期内的较高标准收取费用。例如，01:00:10创建集群，初始配置为每小时6元；01:30:09集群升级，升级后的配置为每小时9元，则01:00:00~02:00:00为1个计费周期，该计费周期的费用为9元。

- 欠费后资源状态变化：账号当前的可用余额（含现金、相关代金券、优惠券、预授权等）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。

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](#)。

② 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。

3.3. 规格选择

本文为您介绍配置独享模式集群时如何选择规格以及注意事项。

背景介绍

独享模式集群相当于一个主从分布式集群。整个集群由Master和Slave两部分构成：

- Master：管理整个集群的资源和Slave之间的交互，但不能用于计算。
- Slave：计算节点。

② 说明 设备间通信以及操作系统需要消耗资源，所以一台Slave的资源并不能全部用于计算。

注意事项

- Slave机型配置决定了您以后的扩容或者缩容的步骤。例如，您的Slave机型是8核32GB，则每次扩容或缩容即增加或减少n台8核32GB的机器，您的可用资源即增加或者减少n个6CU。
- 购买时Master选择三台是为了集群的稳定性，可以做到Master故障时完成主备切换。如果您选择三台Master的配置，阿里云会为您提供服务可用性（SLA）保障。
- Master机型的台数不能进行变更。

规格选择

与共享模式的CU概念（1CU=1核4GB）一样，独享模式配置也可以按照CU的方式进行换算。您可以按照以下计算逻辑，选择出各种机型搭配，实时计算Flink版价格计算器能够为您计算出价格较低的机型配比。

② 说明 如果您是新用户，您可以根据业务的QPS（每秒多少条数据）和业务逻辑的复杂度来转换为CU，再按照[计费方式](#)选择规格。具体的转换方式如下：

- 简单业务：1CU每秒可以处理10000条数据。例如，单流过滤、字符串变换等操作。
- 复杂业务：1CU每秒可以处理1000到5000条数据。例如，JOIN、WINDOW和GROUP BY等操作。

Slave型号以及Master型号和实际可用CU的换算关系经验值如下表：

- 购买Slave机型的最小台数为2，即独享模式最小的实际计算能力为6CU（3CU*2）。

Slave型号	实际可用计算CU数
4核16GB	3CU
8核32GB	6CU
16核64GB	13CU
24核96GB	21CU
32核128GB	28CU
56核224GB	52CU
64核256GB	60CU

 说明 经验值仅供参考。

- Master型号受整体集群CU大小的限制，对应集群最大CU规格经验值如下表。

Master型号	集群最大CU规格
4核16GB	80CU
8核32GB	160CU
16核64GB	800CU
24核96GB	800CU以上


 说明 经验值仅供参考。

3.4. 续费指导

3.4.1. 手动续费

本文为您介绍如何手动为实时计算实例续费。

续费是指项目到期后为项目增加时长。续费时长单位为月或年，最小续费时长为一个月。

 **重要** 实时计算管理控制台的总览页面会为您显示项目的剩余时长。当项目欠费后有停机风险，系统会提醒或通知您，请及时续费，避免对您的服务造成影响。

独享模式续费步骤

1. 登录项目管理页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 鼠标悬停至页面右上角账号位置。

- iii. 单击项目管理。
2. 单击集群管理 > 集群列表。
3. 在集群列表区域，单击目标集群操作列下的更多 > 续费。
4. 在续费页面，选择计费周期。
5. 选中服务协议。
6. 单击立即购买。
7. 单击订购。

共享模式（已停购）续费步骤

1. 登录项目管理页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 鼠标悬停至页面右上角账号位置。
 - iii. 单击项目管理。
2. 单击项目管理 > 项目列表。
3. 在项目列表区域，单击目标集群操作列下的续费。
4. 在续费页面，选择购买时长。
5. 勾选服务协议。
6. 单击立即购买。
7. 单击订购。

3.4.2. 自动续费

本文为您介绍实时计算自动续费功能。您可以为实例开通自动续费、取消自动续费以及修改自动续费时长。

开通自动续费

1. 登录[阿里云控制台](#)。
2. 在页面右上角，单击待续费。
3. 单击手动续费或到期不续费。
4. 在实例列表中，单击目标实例操作列下的开通自动续费。
5. 在开通自动续费页面，选择您期望的自动续费周期。
6. 单击开通自动续费。

取消自动续费

1. 登录[阿里云控制台](#)。
2. 在页面右上角，单击待续费。
3. 单击自动续费。
4. 在实例列表中，单击目标实例操作列下的恢复手动续费。
5. 在恢复为手动续费页面，单击确认。

修改自动续费时长

1. 登录 [阿里云控制台](#)。
2. 在页面右上角，单击待续费。
3. 单击自动续费。
4. 在实例列表中，单击目标实例操作列下的修改自动续费。
5. 在修改自动续费页面，选择您期望的自动续费周期。
6. 单击确定。

3.5. 变配指导

本文为您介绍如何为实时计算集群和项目扩容或缩容。

集群变配

说明

- 变配后的费用：
 - 扩容（master升配或slave扩容）：您需要支付升级相关的费用。
 - 缩容（master降配或slave缩容）：缩容后的差额将退还至您的账户。
- master升配或slave扩容后，需要在使用的数据存储中添加新增的白名单，具体步骤请参见 [数据存储白名单配置](#)。
- 仅独享模式集群支持进行slave扩容（一次扩容不超过20台）或缩容。原因是超过20台扩容的工作流可能会有问题。

您可以调整master型号或slave数量，实现独享集群配置的变更。

1. 登录项目管理页面。
 - i. 登录 [实时计算控制台](#)。
 - ii. 鼠标悬停至页面右上角账号位置。
 - iii. 单击项目管理。
2. 在左侧导航栏，单击 [集群管理](#) > [集群列表](#)。
3. 在集群列表中，单击目标集群操作列下的更多，选择扩容或缩容。
4. 变更资源配置：
 - 独享模式（包年包月）

以扩容为例进行介绍，扩容方式分为以下两种：

■ master升配（使用更高级的型号）

- a. 在变配页面，单击master升配。
- b. 选择您希望升级的master型号。

② 说明

- 只能变更master型号，不支持变更master数量。
- 如果您使用的是3台master的高可用集群，master升配操作会对3台master同时进行升级。

■ slave扩容（增加计算节点的台数）

- a. 在变配页面，单击slave扩容。
- b. 增大slave数量的数值。

② 说明

- 当CU不足时，建议选择slave扩容变配方式，增加slave数量。
- 如果您选择了台数并选中产品服务协议后无法支付，则需要新建一个交换机，并将新建的交换机ID填入VSwitchId文本框，校验通过后，即可付款完成扩容。创建交换机的操作步骤，请参见[创建和管理交换机](#)。
- 不支持跨可用区增加slave数量。
- 已购买独享集群无法变更slave型号，只能调整slave数量。如果您有变更slave型号的需求，则需要重新购买集群。

○ 独享模式（按量付费）

② 说明 按量付费（即后付费）集群暂不支持master升配操作。

- a. 在集群扩容窗口，输入目标机器数。
 - b. 单击确认。
5. 选中产品服务协议。
 6. 单击立即购买。


项目变配

② 说明

- 如果当期资源的计算能力不能满足您的业务需求，您可以进行项目扩容，提高系统的计算能力。
- 如果当前的计算资源的计算能力远超您的业务需求，您可以通过降配的方式，降低费用。

1. 登录项目管理页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 鼠标悬停至页面右上角账号位置。
 - iii. 单击项目管理。

2. 在左侧导航栏，单击项目管理 > 项目列表。
3. 变更资源配置：
 - 独享模式
 - a. 在项目列表中，单击目标项目 操作列下的扩/缩容。
 - b. 在项目扩容/缩容窗口，设置CU数。
 - c. 单击确定。
 - 共享模式（已停购）

 **说明** 因为资源紧张，共享模式无法扩容。如果您在共享模式下有新增作业的需求，请购买独享模式集群。

- a. 在项目列表中，单击目标项目 操作列下的扩容或缩容。
- b. 在变配页面，增加或减少计算资源数量。

3.6. 释放集群

完成集群使用后，您可以通过释放独享模式（按量付费）集群的方式，停止计费。本文为您介绍如何释放实时计算集群。

重要

- 集群释放完成后，项目会被删除。如果您再次购买实时计算服务后新建项目，可以导入原有项目的作业。
- 仅独享模式（按量付费）支持集群释放功能。

1. 登录[实时计算控制台](#)。
2. 鼠标悬停至页面右上角账号位置，单击 项目管理。
3. 在左导航栏中，单击集群列表。
4. 在集群列表页面中，单击目标集群操作列中的释放。
5. 单击确认。

4.准备工作

4.1. RAM用户授权

您可以使用阿里云账号购买和创建实时计算项目。同时，阿里云账号也可以授权RAM用户使用阿里云账号创建的实时计算项目。本文为您介绍如何为RAM用户授权。

什么是RAM用户

RAM用户是RAM的一种实体身份类型，有确定的身份ID和身份凭证，它通常与某个确定的人或应用程序一一对应。RAM用户具备以下特点：

- RAM用户由阿里云账号（主账号）或具有管理员权限的其他RAM用户、RAM角色创建，创建成功后，归属于该阿里云账号，它不是独立的阿里云账号。
- RAM用户不拥有资源，不能独立计量计费，由所属的阿里云账号统一付费。
- RAM用户必须在获得授权后，才能登录控制台或使用API访问阿里云账号下的资源。
- RAM用户拥有独立的登录密码或访问密钥。
- 一个阿里云账号下可以创建多个RAM用户，对应企业内的员工、系统或应用程序。

您可以创建RAM用户并为其授权，实现不同RAM用户拥有不同资源访问权限的目的。当您的企业存在多用户协同访问资源的场景时，使用RAM可以按需为用户分配最小权限，避免多用户共享阿里云账号密码或访问密钥，从而降低企业的安全风险。

授权操作

1. 创建RAM用户

创建RAM用户具体步骤请参见 [创建RAM用户](#)。


说明

- 第一次使用RAM需要进行初始化工作，具体步骤请参见 [设置RAM用户密码强度](#) 和 [设置RAM用户安全策略](#)。
- 为了保证账号安全，实时计算设置了账号验证的功能。如果长时间没有对作业进行操作，系统将会发送短信和邮件进行身份信息验证。

2. 创建自定义权限策略

在RAM用户中添加自定义权限策略的具体步骤请参见 [创建自定义权限策略](#)。实时计算授权策略代码如下。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "stream:*",
      "Resource": "acs:stream:*:*:*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:PassRole",
      "Resource": "acs:ram:*:*:*",
      "Effect": "Allow"
    }
  ]
}
```

 **说明** 实时计算授权策略支持授权至项目粒度，即可以为不同的RAM用户分别授权不同的项目。如果需要为RAM用户授权单个项目，请将以上代码中的Resource修改为"Resource": "acs:stream:*:*:projectname"，其中projectname为需要授权的项目名称。

3. 授权用户或用户组

将上述权限策略授权添加至指定用户或者用户组，具体步骤请参见 [为RAM用户授权](#)和[为用户组授权](#)。

4. 使用RAM用户登录实时计算控制台

在[RAM控制台](#)概览页面中，账号管理区域查看RAM用户登录地址。

4.2. 开通服务和创建项目

本文为您介绍如何开通实时计算Flink版 共享模式和独享模式服务，以及如何创建 共享模式的项目、独享模式的集群和项目。

独享模式开通流程

说明

- 独享模式已于2021年4月28日暂停新购，目前仅支持原有项目的扩缩容和续费操作。如果您有新购需求，推荐使用实时计算Flink全托管。
- 独享模式集群仅能访问相同专有网络VPC、相同Region和相同安全组下的存储资源。如果需要访问其它VPC下的资源，请通过[高速通道](#)等方式连通网络。

独享模式在购买订单后，需要先创建集群，再创建项目。

1. 开通服务

- 登录[实时计算产品首页](#)。

 **说明** 请使用阿里云账号（不能使用阿里云RAM用户）开通服务和创建项目。

- 单击[管理控制台](#)。

- 单击[Blink独享集群](#)下的购买产品，进入购买页面。

- iv. 单击实时计算独享模式（包年包月）或实时计算独享模式（按量付费）。
- v. 填写配置信息。根据需求选择地域、master型号、master数量、slave型号、slave数量和计费周期。

 说明 关于集群配置，请参见 [规格选择](#)。

- vi. 单击立即购买。
- vii. 选中服务协议。例如，实时计算独享模式（包年包月）服务协议。
- viii. 单击去支付。


2. 创建集群

o 准备工作

- 实时计算Flink版独享模式开通时，实时计算Flink版会在您的VPC内创建安全组并申请弹性网卡，详情请参见 [弹性网卡概述](#)。

 说明 请勿删除此安全组和弹性网卡，否则会造成集群无法创建成功。

- 如果已有专有网络VPC，请选择实时计算Flink版需要访问的VPC。
- 如果没有VPC，请开通阿里云VPC服务。如何开通专有网络VPC请参见 [网络规划](#)。


 说明 搭建专有网络时，请确保以下两点：

- VPC内的ECS充足。
- vSwitch内可用IP个数，要大于等于实时计算Flink版集群节点个数。详情请参见 [数据存储白名单配置](#)和[创建和管理交换机](#)。

- 为了保护您的数据安全，独享集群中您上传的UDF包，都会保存到您的OSS存储空间中，请选择已有OSS存储空间。如果没有OSS存储空间，请先创建OSS存储空间。OSS存储空间创建方法，请参见 [创建存储空间](#)。
- 完成[独享模式角色授权](#)。

o 创建集群步骤

- a. 订单支付成功后，单击管理控制台。
- b. 在集群列表页面，单击创建集群。

 说明 新购订单后，集群管理 > 集群列表 > 新建集群上，显示红色数字，表明存在未创建项目的订单。

- c. 在选择订单页面，订单号列表选择您的订单号。
- d. 在基本信息页面，填写集群名称和集群备注。
- e. 在集群配置页面填写配置信息。

 说明 您购买的上下游存储必须和实时计算Flink版所选集群存在于相同的Region、相同的VPC和相同的安全组。

■ OSS Bucket

选择保存您UDF的OSS存储空间。如果您没有OSS存储空间，请先 **创建存储空间**。在创建存储空间（Bucket）时，**存储类型** 请务必选择 **标准存储**，此外，**读写权限** 推荐选择 **私有**，不能选择 **公共读**。

新建 Bucket 创建存储空间 ×

注意： Bucket 创建成功后，您所选择的**存储类型**、**区域**不支持变更。

Bucket 名称 0/63

Bucket 命名规范：
 1. 只能包含小写字母，数字和短横线
 2. 必须以小写字母和数字开头和结尾
 3. 长度限制在 3-63 之间

区域

相同区域内的产品内网可以互通；订购后不支持更换区域，请谨慎选择

Endpoint oss-cn-beijing.aliyuncs.com

存储类型 **标准存储** 低频访问 归档存储

标准：高可靠、高可用、高性能，数据会经常被访问到。
[如何选择适合您的存储类型？](#)

读写权限 **私有** 公共读 公共读写

私有：对文件的所有访问操作需要进行身份验证。

确定 取消

■ VPC

针对需要连接的VPC，编辑自定义名称，并选择需要连接的VPC。

专有网络

创建专有网络 新建 自定义

实例ID名称	IPV4网段	状态	默认专有网络	路由表	交换机	流策略	操作
<input type="text"/>		可用	是	1	3	默认流策略	管理 删除

说明 实时计算只能识别自定义编辑后的VPC名称。

■ Zone

VPC参数正确选择后，可选用的Zone参数会自动显示。

② 说明 以下两种原因可能导致无可选Zone或者vSwitch：

- 所选Zone（可用区）内，ECS库存不足。创建ECS的操作步骤，请参见[使用向导创建实例](#)。
- 所选vSwitch内可用IP个数，小于实时计算Flink版集群节点个数。详情请参见[数据存储白名单配置](#)和[创建和管理交换机](#)。

■ Blink网段

可用的Blink网段会自动显示。

f. 在确认信息页面，单击创建。

② 说明 集群创建过程，即集群状态从启动中变化为运行中，大概需要半个小时。

3. 新建项目

- 在集群管理 > 集群列表，单击目标集群操作列下的创建项目。
- 在新建项目页面，填写项目名称和项目备注，在指定CU右侧，滑动指针，选择目标CU。
- 单击创建。

共享模式开通流程

② 说明 实时计算Flink版共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算Flink版独享模式或Flink半托管/ACK模式。

1. 开通服务

- 登录[实时计算产品首页](#)。

② 说明 请使用阿里云账号（不能使用阿里云RAM用户）开通服务和创建项目。

- 单击管理控制台。
- 单击共享模式（包年包月）下的购买。
- 根据需求选择区域、计算资源和购买时长。
- 单击立即购买。
- 选中服务协议。
- 单击去支付。

2. 创建项目

② 说明 在创建项目或第一次进入项目时，如果出现类似授权错误或提醒，请参见[共享模式角色授权](#)完成角色授权，详情请参见[共享模式角色授权](#)。

- i. 支付完成后，在支付 > 支付完成页面，单击实时计算控制台。
- ii. 在页面顶部菜单栏上，鼠标悬停在用户头像上，单击项目管理。
- iii. 在项目管理 > 项目列表页面，单击右上角新建项目

 说明 订单支付成功后，如果新建项目上显示红色数字标志，表明存在未创建项目的订单。

- iv. 在创建项目页面，选择订单号、填写项目名称和项目备注信息。
- v. 单击创建。

 说明 项目创建完成后，可在项目管理 > 项目列表查看目标项目。

4.3. 角色授权

4.3.1. 独享模式角色授权

本文为您介绍如何完成独享模式角色授权。

创建角色

在创建集群过程中需要您创建角色AliyunStreamDefaultRole，具体操作流程请参见 [自动化角色授予流程](#)。

 说明 如果AliyunStreamDefaultRole角色已存在，则不必重复创建。

添加授权策略

完成RAM角色创建后，您可以将指定的授权策略添加至RAM角色中。

1. 登录RAM控制台。
 - 云账号登录 [RAM控制台](#)。
 - RAM用户登录 [RAM控制台](#)。
2. 在左侧导航栏，单击权限管理 > 权限策略管理。
3. 单击创建权限策略。
4. 填写策略名称（本文以AliyunStreamDefaultRolePolicy为例）和备注。
5. 配置模式中，选择可视化配置或脚本配置。
 - 可视化配置：单击添加授权语句，根据界面提示，配置权限效力、操作名称和资源等。
 - 脚本配置：请参见 [语法结构](#) 编辑策略内容。
6. 在策略内容区域代码框中，输入以下代码，单击确认。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "vpc:DescribeVpcs",
        "vpc:DescribeVSwitches"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecs:CreateSecurityGroup",
        "ecs:AuthorizeSecurityGroup",
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:AttachNetworkInterface",
        "ecs:DescribeNetworkInterfacePermissions",
        "ecs:CreateNetworkInterfacePermission"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

说明 以上授权策略中，以下两个权限可以在创建集群后删除：

- ecs:CreateSecurityGroup
- ecs:AuthorizeSecurityGroup

7. 在RAM角色管理页面底部的RAM角色名称列中，单击AliyunStreamDefaultRole操作列下的添加权限。
8. 在选择权限 > 自定义策略下方搜索栏中，输入AliyunOSSFullAccess。
9. 单击权限策略名称中的AliyunOSSFullAccess。
0. 在选择权限区域，单击系统权限策略 > 自定义策略。
 1. 在选择权限 > 系统策略下方搜索栏中，输入AliyunStreamDefaultRolePolicy。
 2. 单击权限策略名称中的AliyunStreamDefaultRolePolicy。
 3. 单击确定。

4.3.2. 共享模式角色授权

本文为您介绍实时计算的角色授权操作，包括自动化角色授予流程、重新初始化流程和常见问题。

您开通实时计算服务时，需要先为实时计算的服务账号，授予系统默认角色 AliyunStreamDefaultRole。

说明

- 仅当AliyunStreamDefaultRole被正确授予后，实时计算才能正常地调用相关服务（例如RDS）。
- 实时计算共享模式已于2019年12月24日正式下线，将不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。若有新购需求，推荐使用实时计算独享模式或Flink云原生模式。

自动化角色授予流程

1. 单击前往授权，进行角色授权。



说明

- 请先完成实时计算产品购买，产品购买步骤请参见 [开通服务和创建项目](#)。
- 以上角色授权提示，仅在您没有正确授予实时计算服务账号默认角色时出现。

2. 单击AliyunStreamDefaultRole > 同意授权，完成角色授权。



说明

当完成以上授权步骤后，刷新实时计算的控制台，即可以进行业务操作。

当前角色授权信息查看

1. 登录RAM角色管理。
2. 在RAM角色管理页面底部的RAM角色名称列中单击AliyunStreamDefaultRole。
3. 在AliyunStreamDefaultRole页面中单击权限管理 > AliyunStreamRolePolicy。
4. 在策略内容页签内查看实时计算当前策略信息，如下所示。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ots:List*",
        "ots:DescribeTable",
        "ots:Get*",
        "ots:*Row"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "dhs:Create*",
        "dhs:List*",
        "dhs:Get*",
        "dhs:PutRecords",
        "dhs>DeleteTopic"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:List*",
        "log:Get*",
        "log:Post*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "mns:List*",
        "mns:Get*",
        "mns:Send*",
        "mns:Publish*",
        "mns:Subscribe"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "drds:DescribeDrdsInstance",
        "drds:ModifyDrdsIpWhiteList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
```

```
"rds:Describe*",
  "rds:ModifySecurityIps*"
],
"Resource": "*",
"Effect": "Allow"
},
{
  "Action": [
    "vpc:DescribeVpcs",
    "vpc:DescribeVSwitches"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
{
  "Action": [
    "ecs:CreateSecurityGroup",
    "ecs:AuthorizeSecurityGroup",
    "ecs:CreateNetworkInterface",
    "ecs:DescribeNetworkInterfaces",
    "ecs:AttachNetworkInterface",
    "ecs:DescribeNetworkInterfacePermissions",
    "ecs:CreateNetworkInterfacePermission"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
{
  "Action": "oss:*",
  "Resource": "*",
  "Effect": "Allow"
}
]
}
```

重新初始化

重要

- 删除AliyunStreamDefaultRole角色后，可能导致实时计算正在运行的作业瞬间无法读写上下游外部存储，直到后续重新初始化AliyunStreamDefaultRole角色后才能恢复正常。请务必确认这项操作对于线上实时计算业务造成的影响。
- RAM删除角色和流计算重新初始化操作均需要当前用户的阿里云账号或者经阿里云账号授权的RAM用户操作。

当您删除或修改AliyunStreamRolePolicy授权策略导致无法恢复时，您可以在RAM中删除AliyunStreamDefaultRole，并重新进入实时计算控制台，让实时计算重新启动 [自动化角色授予流程](#)。


重新初始化步骤：

1. 登录[RAM角色管理](#)。
2. 在RAM角色管理页面底部的RAM角色名称列中单击AliyunStreamDefaultRole。

3. 在AliyunStreamDefaultRole页面的权限管理页签中，单击AliyunStreamRolePolicy操作列下的移除权限。

 说明 如果AliyunStreamRolePolicy已经删除，可跳过此步骤。

4. 单击AliyunStreamDefaultRole页面底部的RAM角色管理，返回RAM角色管理页面。
5. 在RAM角色管理页面底部，单击AliyunStreamDefaultRole的操作列下的删除。
6. 登录[实时计算控制台](#)。
7. 单击前往授权。
8. 单击同意授权。
9. 通过手机短信等验证方式，完成角色授权。

 说明 实时计算和RAM对于授权均有不同程度的缓存，有可能在删除角色或者授权角色后，实时计算仍然提示需要授权角色。此时系统已经完成授权任务，单击授权完毕即可。

常见问题

Q: RAM用户在操作数据存储的时候如果出现错误信息 **操作错误：RAM用户没有passRole权限，请到RAM对该RAM用户赋权。错误详情为：922019。**

A: 请先完成角色授权。详情请参见 [授权操作](#)。

5. Blink SQL参考

5.1. 概述

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

本章节通过以下方面，为您介绍实时计算中Flink SQL的使用方法。

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

5.2. 关键字

本文为您介绍实时计算中已保留的关键字和使用关键字字符的方法。

关键字常用类型

常用类型	关键字
数据类型	<ul style="list-style-type: none">• VARCHAR• INT• BIGINT• DOUBLE• DATE• BOOLEAN• TINYINT• SMALLINT• FLOAT• DECIMAL• VARBINARY
DDL	<ul style="list-style-type: none">• CREATE TABLE• CREATE FUNCTION• CREATE VIEW

常用类型	关键字
DML	INSERT INTO
SELECT 子句	<ul style="list-style-type: none"> • SELECT FROM • WHERE • GROUP BY • JOIN

命名规则

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

保留关键字

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

A,ABS,ABSOLUTE,ACTION,ADA,ADD,ADMIN,AFTER,ALL,ALLOCATE,ALLOW,ALTER,ALWAYS,AND,ANY,ARE,ARRAY,AS,ASC,ASENSITIVE,ASSERTION,ASSIGNMENT,ASYMMETRIC,AT,ATOMIC,ATTRIBUTE,ATTRIBUTES,AUTHORIZATION,AVG,
BEFORE,BEGIN,BERNOULLI,BETWEEN,BIGINT,BINARY,BIT,BLOB,BOOLEAN,BOTH,BREADTH,BY,
C,CALL,CALLED,CARDINALITY,CASCADE,CASCADE,CASE,CAST,CATALOG,CATALOG_NAME,CEIL,CEILING,CENTURY,CHAIN,CHAR,CHARACTER,CHARACTERISTICS,CHARACTERS,CHARACTER_LENGTH,CHARACTER_SET_CATALOG,CHARACTER_SET_NAME,CHARACTER_SET_SCHEMA,CHAR_LENGTH,CHECK,CLASS_ORIGIN,CLOB,CLOSE,COALESCE,COBOL,COLLATE,COLLATION,COLLATION_CATALOG,COLLATION_NAME,COLLATION_SCHEMA,COLLECT,COLUMN,COLUMN_NAME,COMMAND_FUNCTION,COMMAND_FUNCTION_CODE,COMMIT,COMMITTED,CONDITION,CONDITION_NUMBER,CONNECT,CONNECTION,CONNECTION_NAME,CONSTRAINT,CONSTRAINTS,CONSTRAINT_CATALOG,CONSTRAINT_NAME,CONSTRAINT_SCHEMA,CONSTRUCTOR,CONTAINS,CONTINUE,CONVERT,CORR,CORRESPONDING,COUNT,COVAR_POP,COVAR_SAMP,CREATE,CROSS,CUBE,CUME_DIST,CURRENT,CURRENT_CATALOG,CURRENT_DATE,CURRENT_DEFAULT_TRANSFORM_GROUP,CURRENT_PATH,CURRENT_ROLE,CURRENT_SCHEMA,CURRENT_TIME,CURRENT_TIMESTAMP,CURRENT_TRANSFORM_GROUP_FOR_TYPE,CURRENT_USER,CURSOR,CURSOR_NAME,CYCLE,
DATA,DATABASE,DATE,DATETIME_INTERVAL_CODE,DATETIME_INTERVAL_PRECISION,DAY,DEALLOCATE,DECADE,DECIMAL,DECLARE,DEFAULT,DEFAULTS,DEFERRABLE,DEFERRED,DEFINED,DEFINER,DEGREE,DELETE,DENSE_RANK,DEPTH,DEREF,DERIVED,DESC,DESCRIBE,DESCRIPTION,DESCRIPTOR,DETERMINISTIC,DIAGNOSTICS,DISALLOW,DISCONNECT,DISPATCH,DISTINCT,DOMAIN,DOUBLE,DOW,DOY,DROP,DYNAMIC,DYNAMIC_FUNCTION,DYNAMIC_FUNCTION_CODE,
EACH,ELEMENT,ELSE,END,END-
EXEC,EPOCH,EQUALS,ESCAPE,EVERY,EXCEPT,EXCEPTION,EXCLUDE,EXCLUDING,EXEC,EXECUTE,EXISTS,EXPLAIN,EXTEND,EXTERNAL,EXTRACT,
FALSE,FETCH,FILTER,FINAL,FIRST,FIRST_VALUE,FLOAT,FLOOR,FOLLOWING,FOR,FOREIGN,FORTRAN,FOUND,FRAC_SECOND,FREE,FROM,FULL,FUNCTION,FUSION,
G,GENERAL,GENERATED,GET,GLOBAL,GO,GOTO,GRANT,GRANTED,GROUP,GROUPING,
HAVING,HIERARCHY,HOLD,HOUR,IDENTITY,IMMEDIATE,IMPLEMENTATION,IMPORT,
IN,INCLUDING,INCREMENT,INDICATOR,INITIALLY,INNER,INOUT,INPUT,INSENSITIVE,INSERT,INSTANCE,INSTANTIABLE,INT,INTEGER,INTERSECT,INTERSECTION,INTERVAL,INTO,INVOKER,IS,ISOLATION,
JAVA,JOIN,
K,KEY,KEY_MEMBER,KEY_TYPE,
LABEL,LANGUAGE,LARGE,LAST,LAST_VALUE,LATERAL,LEADING,LEFT,LENGTH,LEVEL,LIBRARY,LIKE,LIMIT,LN,LOCAL,LOCALTIME,LOCALTIMESTAMP,LOCATOR,LOWER,
M,MAP,MATCH,MATCHED,MAY,MAYVALUE,MEMBER,MERGE,MESSAGE_LENGTH,MESSAGE_OCTET_LENGTH,ME

```

M,MAP,MATCH,MATCHED,MAX,MAXVALUE,MEMBER,MERGE,MESSAGE_LENGTH,MESSAGE_OCTET_LENGTH,MESSAGE_TEXT,METHOD,MICROSECOND,MILLENNIUM,MIN,MINUTE,MINVALUE,MOD,MODIFIES,MODULE,MONTH,
MORE,MULTISET,MUMPS,
NAME,NAMES,NATIONAL,NATURAL,NCHAR,NCLOB,NESTING,NEW,NEXT,NO,NONE,NORMALIZE,NORMALIZED,
NOT,NULL,NULLABLE,NULLIF,NULLS,NUMBER,NUMERIC,
OBJECT,OCTETS,OCTET_LENGTH,OF,OFFSET,OLD,ON,ONLY,OPEN,OPTION,OPTIONS,OR,ORDER,ORDERING,ORDINALITY,OTHERS,OUT,OUTER,OUTPUT,OVER,OVERLAPS,OVERLAY,OVERRIDING,
PAD,PARAMETER,PARAMETER_MODE,PARAMETER_NAME,PARAMETER_ORDINAL_POSITION,PARAMETER_SPECIFIC_CATALOG,PARAMETER_SPECIFIC_NAME,PARAMETER_SPECIFIC_SCHEMA,PARTIAL,PARTITION,PASCAL,PASTHROUGH,PATH,PERCENTILE_CONT,PERCENTILE_DISC,PERCENT_RANK,PLACING,PLAN,PLI,POSITION,POWER,PRECEDING,Precision,PREPARE,PRESERVE,PRIMARY,PRIOR,PRIVILEGES,PROCEDURE,PUBLIC,
QUARTER,
RANGE,RANK,READ,READS,REAL,RECURSIVE,REF,REFERENCES,REFERENCING,REGR_AVGX,REGR_AVGY,REGR_COUNT,REGR_INTERCEPT,REGR_R2,REGR_SLOPE,REGR_SXX,REGR_SXY,REGR_SYY,RELATIVE,RELEASE,REPEATABLE,RESET,RESTART,RESTRICT,RESULT,RETURN,RETURNED_CARDINALITY,RETURNED_LENGTH,RETURNED_OCTET_LENGTH,RETURNED_SQLSTATE,RETURNS,REVOKE,RIGHT,ROLE,ROLLBACK,ROLLUP,ROUTINE,ROUTINE_CATALOG,ROUTINE_NAME,ROUTINE_SCHEMA,ROW,ROWS,ROW_COUNT,ROW_NUMBER,
SAVEPOINT,SCALE,SCHEMA,SCHEMA_NAME,SCOPE,SCOPE_CATALOGS,SCOPE_NAME,SCOPE_SCHEMA,SCROLL,SEARCH,SECOND,SECTION,SECURITY,SELECT,SELF,SENSITIVE,SEQUENCE,SERIALIZABLE,SERVER,SERVER_NAME,SESSION,SESSION_USER,SET,SETS,SIMILAR,SIMPLE,SIZE,SMALLINT,SOME,SOURCE,SPACE,SPECIFIC,SPECIFICTYPE,SPECIFIC_NAME,SQL,SQLException,SQLSTATE,SQLWARNING,SQL_TSI_DAY,SQL_TSI_FRAC_SECOND,SQL_TSI_HOUR,SQL_TSI_MICROSECOND,SQL_TSI_MINUTE,SQL_TSI_MONTH,SQL_TSI_QUARTER,SQL_TSI_SECOND,SQL_TSI_WEEK,SQL_TSI_YEAR,SQRT,START,STATE,STATEMENT,STATIC,STDDEV_POP,STDDEV_SAMP,STREAM,STRUCTURE,STYLE,SUBCLASS_ORIGIN,SUBMULTISET,SUBSTITUTE,SUBSTRING,SUM,SYMMETRIC,SYSTEM,SYSTEM_USER,
TABLE,TABLESAMPLE,TABLE_NAME,TEMPORARY,THEN,TIES,TIME,TIMESTAMP,TIMESTAMPADD,TIMESTAMPDIFF,TIMEZONE_HOUR,TIMEZONE_MINUTE,TINYINT,TO,TOP_LEVEL_COUNT,TRAILING,TRANSACTION,TRANSACTIONS_ACTIVE,TRANSACTIONS_COMMITTED,TRANSACTIONS_ROLLED_BACK,TRANSFORM,TRANSFORMS,TRANSLATE,TRANSLATION,TREAT,TRIGGER,TRIGGER_CATALOG,TRIGGER_NAME,TRIGGER_SCHEMA,TRIM,TRUE,TYPE,UESCAPE,UNBOUNDED,UNCOMMITTED,UNDER,UNION,UNIQUE,UNKNOWN,UNNAMED,UNNEST,UPDATE,UPPER,UPSERT,USAGE,USER,USER_DEFINED_TYPE_CATALOG,USER_DEFINED_TYPE_CODE,USER_DEFINED_TYPE_NAME,USER_DEFINED_TYPE_SCHEMA,USING,
VALUE,VALUES,VARBINARY,VARCHAR,VARYING,VAR_POP,VAR_SAMP,VERSION,VIEW,
WEEK,WHEN,WHENEVER,WHERE,WIDTH_BUCKET,WINDOW,WITH,WITHIN,WITHOUT,WORK,WRAPPER,WRITE,XML,
YEAR,
ZONE

```

5.3. 基本概念

5.3.1. 时区

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

配置时区参数

- 配置相同时区参数

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

- 时区列表请参见 [支持的时区列表](#)。
- 时区参数配置方法请参见 [配置作业参数](#)。

- 配置不同时区参数

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

```
CREATE TABLE mysql_source_my_table (
  -- ...
) WITH (
  timeZone='America/New_York'
  -- ...
);
```

时区参数配置示例

实时计算Flink版时区相关函数语义上为自定义的时区。下文以 `Asia/Shanghai` 自定义时区为例进行说明。

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

```
TO_TIMESTAMP('2020-08-03 10:59:45.957')
-- 输出：`2020-08-03 10:59:45.957`。
TIMESTAMP '2020-08-03 10:59:45.957'
-- 输出：`2020-08-03 10:59:45.957`。
UNIX_TIMESTAMP('2020-08-03 10:59:45.957')
-- 输出：`1596423585`。
```

- 时间类型转字符串（`DATE_FORMAT`和`FROM_UNIXTIME`）

 说明 当参数为TIMESTAMP时，输出结果取决于自定义设定的时区。

```
DATE_FORMAT(TO_TIMESTAMP(1596702949000), 'yyyy-MM-dd HH:mm:ss')
-- 输出：`2020-08-06 16:35:49`。
DATE_FORMAT('2020-08-06 16:35:49', 'yyyy-MM-dd HH:mm:ss', 'yyyy/MM/dd HH:mm:ss')
-- 输出：`2020/08/06 16:35:49`
FROM_UNIXTIME(1596702949000/1000)
-- 输出：`2020-08-06 16:35:49`
```

- 时间相关计算函数

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

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

- 当前时间函数

当前时间函数包

括 `LOCALTIMESTAMP()`、`CURRENT_TIMESTAMP()`、`NOW()` 和 `UNIX_TIMESTAMP()` 等。

```
-- 当前时间是2020-08-03 10:59:45 (Asia/Shanghai)
LOCALTIMESTAMP
-- 输出: `2020-08-03 10:59:45.957`。
CURRENT_TIMESTAMP
-- 输出: `2020-08-03 10:59:45.957`。
NOW()
-- 输出: `1596423585`。
UNIX_TIMESTAMP()
-- 输出: `1596423585`。
```

- **DATE和TIME类型**

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

支持的时区列表

实时计算Flink版支持的时区列表请参见 [TimeZone](#)。

5.3.2. 时间属性

本文为您介绍Blink SQL所支持的Event Time和Processing Time时间类型及属性。

Apache Flink支持三种与流数据处理相关的时间概念：Processing Time、Event Time和Ingestion Time。Blink SQL仅支持其中的两种时间类型Event Time和Processing Time：

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

Event Time

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

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

窗口函数基于Event Time聚合的示例如下。

```
CREATE TABLE tt_stream (  
  a VARCHAR,  
  b VARCHAR,  
  ts TIMESTAMP,  
  WATERMARK wk1 FOR ts as withOffset (ts, 1000) --Watermark计算方法。  
) WITH (  
  type = 'sls',  
  topic = '<yourTopicName>',  
  accessId = '<yourAccessId>',  
  accessKey = '<yourAccessSecret>'  
);  
CREATE TABLE rds_output (  
  id VARCHAR,  
  win_start TIMESTAMP,  
  win_end TIMESTAMP,  
  cnt BIGINT  
) WITH (  
  type = 'rds',  
  url = 'jdbc:mysql://****3306/test',  
  tableName = '<yourTableName>',  
  userName = '<yourUserName>',  
  password = '<yourPassword>'  
);  
INSERT  
  INTO rds_output  
SELECT  
  a AS id,  
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,  
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,  
  COUNT (a) AS cnt  
FROM  
  tt_stream  
GROUP  
  BY SESSION (ts, INTERVAL '1' SECOND),  
  a
```

Processing Time

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

```
fileName as PROCTIME()
```

窗口函数基于Processing Time聚合的示例如下。

```
CREATE TABLE mq_stream (  
  a VARCHAR,  
  b VARCHAR,  
  c BIGINT,  
  ts AS PROCTIME () --在数据源表的声明中明文定义一个Processing Time列。  
) WITH (  
  type = 'mq',  
  topic = '<yourTopic>',  
  accessId = '<yourAccessId>',  
  accessKey = '<yourAccessSecret>'  
);  
CREATE TABLE rds_output (  
  id VARCHAR,  
  win_start TIMESTAMP,  
  win_end TIMESTAMP,  
  cnt BIGINT  
) with (  
  type = 'rds',  
  url = '<yourDatebaseURL>',  
  tableName = '<yourDatabasTableName>',  
  userName = '<yourUserName>',  
  password = '<yourPassword>'  
);  
INSERT  
  INTO rds_output  
SELECT  
  a AS id,  
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,  
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,  
  COUNT (a) AS cnt  
FROM  
  mq_stream  
GROUP  
  BY SESSION (ts, INTERVAL '1' SECOND),  
  a
```

时间属性字段传递

时间属性字段经过如下操作后会失去时间属性特性：

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

 **说明** 双流JOIN详情请参见[JOIN语句](#)。

- **复杂事件处理（CEP）语句**中的MATCH_RECOGNIZE操作。
- **OVER窗口**中的PARTITION BY操作。
- **UNION操作**。UNION = RETRACT + UNION ALL。

如果经过以上操作后，继续使用该时间属性字段进行窗口函数运算，会出现类

似 `org.apache.flink.table.api.ValidationException: Window can only be defined over a time attribute column.` 的报错。

5.3.3. Watermark

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

定义

由于实时计算的输入数据是持续不断的，因此我们需要一个有效的进度指标，来帮助我们确定关闭时间窗口的正确时间点，保证关闭窗口后不会再有数据进入该窗口，可以安全输出这个窗口的聚合结果。而Watermark就是一种衡量Event Time进展的有效机制。随着时间的推移，最早流入实时计算的数据会被处理完成，之后流入的数据处于正在处理状态。处于正在处理部分的和已处理部分的交界的时间戳，可以被定义为Watermark，代表在此之前的事件已经被处理完成并输出。

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

语法

Watermark的定义是数据源表DDL定义的一部分，Watermark语法定义如下。

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

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

示例

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

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

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

说明

- **Event Time**和**Processing Time**只能在源表声明。
- 在使用Event Time Watermark时，rowtime必须是TIMESTAMP类型。如果您的数据源表中没有TIMESTAMP类型的列，可以使用**计算列**方法从其它类型的字段进行转换。
- 当前实时计算支持毫秒级别的、在Unix时间戳里是13位TIMESTAMP数据类型。如果您的数据源表的rowtime字段为非13位的时间戳，可以使用**计算列**方法完成转化。

5.3.4. 计算列

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

计算列概念

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

计算列的用途

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

计算列语法

```
column_name AS computed_column_expression
```

计算列示例

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

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

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

5.4. 数据类型

5.4.1. 类型转换

实时计算数据类型转换。

本文为您介绍实时计算支持的数据类型以及数据类型之间的转换方法。

实时计算支持的数据类型

数据类型	说明	值域
VARCHAR	可变长度字符串	VARCHAR最大容量为4MB。
BOOLEAN	逻辑值	取值为TRUE、FALSE或UNKNOWN。
TINYINT	微整型，1字节整数。	-128 ~ 127
SMALLINT	短整型，2字节整数。	-32768 ~ 32767
INT	整型，4字节整数。	-2147483648 ~ 2147483647
BIGINT	长整型，8字节整数。	-9223372036854775808 ~ 9223372036854775807
FLOAT	4字节浮点型	6位数字精度
DECIMAL	小数类型	示例：123.45 是 DECIMAL(5,2) 的值。
DOUBLE	浮点型，8字节浮点型。	15位十进制精度。
DATE	日期类型	示例：DATE'1969-07-20'
TIME	时间类型	示例：TIME '20: 17: 40'
TIMESTAMP	时间戳，显示日期和时间。	示例：TIMESTAMP '1969-07-20 20:17:40'
VARBINARY	二进制数据	byte[] 数组。 说明 VARBINARY没有最大容量的限制。

数据类型转换

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

类型转换示例

- 测试数据

var1 (VARCHAR)	big1 (BIGINT)
1000	323

- 测试语句

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

- 测试结果

AA (BIGINT)	BB (VARCHAR)
1000	323

5.4.2. 数学和逻辑运算

实时计算数据类型运算

本文为您介绍实时计算数据类型之间的数学运算和逻辑运算关系。

说明 同一运算语句中numeric1和numeric2的数据类型需要保持一致。

运算语句	描述	numeric1和numeric2支持的数据类型	示例
numeric1 + numeric2	返回前后两数字数学运算的和		2+4.2
numeric1 - numeric2	返回前后两数字数学运算差值		3-5.3
numeric1 * numeric2	返回前后两数字数学运算积值		2*4
numeric1 / numeric2	返回前后两数字数学运算商值		2.4/5


运算语句	描述	numeric1和numeric2支持的数据类型	示例
<code>numeric1 > numeric2</code>	返回前面数字是否大于后面数字的逻辑运算值	<ul style="list-style-type: none"> DECIMAL BIGINT 	<code>2.4>5</code>
<code>numeric1 < numeric2</code>	返回前面数字是否小于后面数字的逻辑运算值		<code>2.4<5</code>
<code>numeric1 >= numeric2</code>	返回前面数字是否大于等于后面数字的逻辑运算值		<code>2.4>=5</code>
<code>numeric1 <= numeric2</code>	返回前面数字是否小于等于后面数字的逻辑运算值		<code>2.4<=5</code>
<code>numeric1 = numeric2</code>	返回前后两数字是否相同（等）的逻辑运算值	<ul style="list-style-type: none"> INT DOUBLE DECIMAL BIGINT VARCHAR 	<code>'iphone' = 5</code>
<code>numeric1 <> numeric2</code>	返回前后两数字是否不相同（等）的逻辑运算值		<code>'iphone' <> 5</code>

5.5. 创建数据视图

您可以通过创建实时计算数据视图简化开发过程。

背景信息

通常，业务逻辑比较复杂时，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 VIEW viewName[ (columnName[ , columnName]* ) ] AS queryStatement;
```

- `viewName`: 视图名称。
- `columnName`: 字段名称。
- `queryStatement`: 嵌套语句别名。

示例1

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

示例2

- 测试数据

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

- 测试语句

```

CREATE TABLE datahub_stream (
  a VARCHAR,
  b BIGINT,
  c TIMESTAMP,
  d AS PROCTIME()
) WITH (
  TYPE='datahub',
  ...
);
CREATE TABLE rds_output (
  a VARCHAR,
  b TIMESTAMP,
  cnt BIGINT,
  PRIMARY KEY(a)
)WITH(
  TYPE = 'rds',
  ...
);
CREATE VIEW rds_view AS
SELECT a,
  CAST(
    HOP_START(d, INTERVAL '5' SECOND, INTERVAL '30' SECOND) AS TIMESTAMP
  ) AS cc,
  SUM(b) AS cnt
FROM
  datahub_stream
GROUP BY
  HOP(d, INTERVAL '5' SECOND, INTERVAL '30' SECOND),a;
INSERT INTO
  rds_output
SELECT
  a,
  cc,
  cnt
FROM
  rds_view
WHERE
  cnt=4;

```

- 测试结果

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

5.6. DDL语句

5.6.1. 概述

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

语法

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

说明

阿里云实时计算Flink版本本身不具有数据存储功能，所有涉及表创建的DDL操作，实际上均是对外部数据表、存储的引用声明。

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

以上代码不是在Flink SQL中创建了消息队列（MQ）源表的topic，而是声明了一个名称为 `mq_stream` 的表引用。下游所有对MQ `blink_mq_test` Topic的相关DML操作，均可以使用别名 `mq_stream` 代替。声明外部表时，请注意：

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

说明

- 如果上下游插件支持根据KEY取值，则不要求两者字段数量一致，但字段名称需要一致。
- 如果上下游插件不支持根据KEY取值，则需要字段数量和字段顺序一致。

字段映射

声明表的字段映射根据外部数据源是否有Schema，分为两大类：

- 顺序映射

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

以MQ的1条记录为例。


```
asavfa,sddd32,sdfds
```

按照命名规范设置MQ的字段名。

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


- 名称映射

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

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

DataHub定义的Schema如下：

字段名	类型
name	STRING
age	BIGINT
value	STRING

 **说明** DataHub中的STRING数据类型对应实时计算Flink版中的VARCHAR类型。

关于上述DataHub声明的DDL如下：

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

 **说明** 建议您将所有列进行声明引用。声明引用时可以减少字段，不能新增字段。

处理大小写敏感

SQL标准定义中，大小写是不敏感的。如下两个示例，语句的含义相同。

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

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

实时计算Flink版引用的大量外部数据源中，有时会存在要求大小写敏感的数据源。例如，表格存储（Table Store）对于大小写是敏感的。如果需要在Table Store定义大写 NAME 字段，您应该进行如下定义。

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

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

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

相关章节

请参见以下文档，了解如何创建实时计算Flink版数据源表、数据结果表和数据维表。

- [数据源表概述](#)。
- [数据结果表概述](#)。
- [概述](#)。

5.6.2. 创建数据源表

5.6.2.1. 数据源表概述

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

语法


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

示例

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

获取数据源表属性字段

- 获取数据源表属性字段语法

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

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

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

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

- 获取源表属性字段示例

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

字段名	说明
<code>__source__</code>	消息源
<code>__topic__</code>	消息主题

字段名	说明
<code>__timestamp__</code>	日志时间

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

示例如下：

○ 示例数据

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

○ 示例语句

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

○ 测试结果

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

包含窗口函数的数据源表

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

支持创建的数据源表类型

实时计算支持创建多种类型的数据源表：

- **创建数据总线DataHub源表**

- [创建日志服务SLS源表](#)
- [创建消息队列MQ源表](#)
- [创建消息队列Kafka源表](#)
- [创建表格存储Tablestore源表](#)
- [创建全量MaxCompute源表](#)

5.6.2.2. 创建Oracle数据库源表

本文为您介绍如何创建Oracle数据库源表，以及创建源表时使用的WITH参数、类型映射、代码示例和常见问题。

重要

- 本文仅适用于Blink 3.4.x及以上版本。
- 仅支持使用Oracle 11g版本创建Oracle数据库源表。
- 请不要手动修改Oracle Source节点的并发数量，默认一个Table对应一个并发。

语法示例

实时计算Flink版支持使用Oracle数据库作为源表，代码示例如下。

```
create table oracle_source (  
  EMPLOYEE_ID BIGINT,  
  START_DATE TIMESTAMP,  
  END_DATE TIMESTAMP,  
  JOB_ID VARCHAR,  
  DEPARTMENT_ID VARCHAR  
) with (  
  type = 'oracle',  
  url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',  
  userName = 'userName',  
  password = 'password',  
  dbName = 'hr',  
  tableName = 'job_history',  
  timeField = 'START_DATE',  
  startTime = '2007-1-1 00:00:00'  
);
```

WITH 参数

参数	说明	是否必选	备注
type	源表类型	是	固定值为 <i>oracle</i> 。
url	数据库连接串	是	固定句式为 <code>jdbc:oracle:thin:@//数据库IP:端口号/数据库名</code> 。例 如 <code>jdbc:oracle:thin:@//127.0.0.1:1521/XE</code> 。

参数	说明	是否必选	备注
userName	登录数据库的用户名	是	无
password	登录数据库的密码	是	无
tableName	数据库的表名。数据库的表名有以下两种表达方式： <ul style="list-style-type: none"> 表名1,表名2 数据库名.表名1,表名2 <div style="border: 1px solid #ccc; background-color: #e0f2f1; padding: 5px; margin-top: 5px;"> ? 说明 多个表名之间用英文逗号(,)隔开。 </div>	是	<ul style="list-style-type: none"> table1,table2 db1.table1,table2
timeField	更新数据库的时间	是	无
dbName	数据库名	否	如果tableName参数配置了数据库名,则dbName不需要重复配置。
startTime	读取数据的开始时间	否	2019-5-15 00:00:00
timeZone	数据库时区	否	Asia/Shanghai, "UTC"
queryTimeRangeMs	获取数据的时长,单位为毫秒。 <div style="border: 1px solid #ccc; background-color: #e0f2f1; padding: 5px; margin-top: 5px;"> ? 说明 queryTimeRangeMs参数的取值需要大于queryIntervalMs参数。 </div>	否	默认值为5000。
queryIntervalMs	查询数据库的时间间隔,单位为毫秒。	否	默认值为100。
connectionMaxActive	最大活跃连接数	否	默认值为10。
maxRetry	最大连接失败重试次数	否	默认值为3。
escapeFields	是否对数据库字段名进行转义。	否	escapeFields参数的取值如下： <ul style="list-style-type: none"> false (默认值), 不区分大小写。 true, 区分大小写。

参数	说明	是否必选	备注
lengthCheck	单行字段条数的检查策略	否	lengthCheck参数的取值如下： <ul style="list-style-type: none"> • NONE（默认值）： <ul style="list-style-type: none"> ◦ 当解析的字段个数大于定义个数时，按从左到右的顺序，取定义字段的数据使用。 ◦ 当解析的字段个数小于定义个数时，跳过当前行数据。 • SKIP：当解析的字段个数和定义个数不同时，跳过当前行数据。 • EXCEPTION：当解析的字段个数和定义个数不同时，系统提示异常。 • PAD： <ul style="list-style-type: none"> ◦ 当解析的字段个数大于定义个数时，按从左到右的顺序，取定义字段的数据使用。 ◦ 当解析的字段个数小于定义个数时，按从左到右顺序，在行尾用<code>null</code>填充缺少的字段。
columnError Debug	是否打开调试开关。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; width: fit-content;"> ? 说明 如果打开调试开关，则会将解析异常的日志打印出来。 </div>	否	默认值为 <code>false</code> 。

类型映射

Oracle字段类型	实时计算Flink版字段类型
<ul style="list-style-type: none"> • CHAR • VARCHAR • VARCHAR2 	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

代码示例

实时计算Flink版包含Oracle数据库源表的代码示例如下。

```
create table oracle_source (  
  EMPLOYEE_ID BIGINT,  
  START_DATE TIMESTAMP,  
  END_DATE TIMESTAMP,  
  JOB_ID VARCHAR,  
  DEPARTMENT_ID VARCHAR  
) with (  
  type = 'oracle',  
  url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',  
  userName = 'userName',  
  password = 'password',  
  dbName = 'hr',  
  tableName = 'job_history',  
  timeField = 'START_DATE',  
  startTime = '2007-1-1 00:00:00'  
);  
create table test_out(  
  EMPLOYEE_ID BIGINT,  
  START_DATE TIMESTAMP,  
  END_DATE TIMESTAMP,  
  JOB_ID VARCHAR,  
  DEPARTMENT_ID VARCHAR  
) with (  
  type='print'  
);  
INSERT INTO test_out  
SELECT  
  EMPLOYEE_ID,  
  START_DATE,  
  END_DATE,  
  JOB_ID,  
  DEPARTMENT_ID  
from oracle_source;
```

常见问题

Q: 查询不到数据该如何处理?

A: 查询不到数据是因为Blink运行故障。您需要查看TaskManager的 `Round start:[]`、`end:[]` 和 `Round read records` 日志，如果未查询到日志数据，则Blink运行故障。您可以参见[运行Failover](#)解决Blink运行故障。

说明

- `Round start:[]`、`end:[]` 用来显示查询数据的起始时间戳。
- `Round read records` 用来显示查询到的数据记录。

5.6.2.3. 创建数据总线DataHub源表

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

 **重要** 本文仅适用于Blink 1.4.5及以上版本。

什么是数据总线

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。实时计算Flink版使用DataHub作为流式数据存储源头或输出目的端。

语法示例

DataHub可以作为实时计算Flink版的数据输入，示例如下。

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

 **说明** 建议您使用存储注册功能，详情请参见 [注册数据总线DataHub](#)。

WITH参数

参数	说明	是否必填	备注
type	源表类型	是	固定值为 <code>datahub</code> 。
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	读取的项目	是	无
topic	Project下的具体的Topic名称	是	无  重要 目前只支持Tuple模式的Topic。
startTime	启动位点的时间	否	批处理时必须填，格式为 <code>yyyy-MM-dd hh:mm:ss</code> 。
endTime	DataHub日志结束时间	否	批处理时必须填，格式为 <code>yyyy-MM-dd hh:mm:ss</code> 。

参数	说明	是否必填	备注
maxRetryTimes	读取最大重试次数	否	不同版本中，该参数的默认值如下： <ul style="list-style-type: none"> Blink 2.2.7以下版本：3 Blink 2.2.7及以上版本：20
retryIntervalMs	重试间隔	否	不同版本中，该参数的默认值如下（单位为毫秒）： <ul style="list-style-type: none"> Blink 2.2.7以下版本：1000 Blink 2.2.7及以上版本：50
batchReadSize	单次读取条数	否	默认值为10，可设置的最大值为1000。
lengthCheck	单行字段条数检查策略	否	<ul style="list-style-type: none"> NONE（默认值）： <ul style="list-style-type: none"> 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。 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。 <div style="background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>说明</p> <ul style="list-style-type: none"> 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。 </div>
columnErrorDebug	是否开启调试功能	否	<ul style="list-style-type: none"> false（默认值）：关闭调试功能。 true：开启调试功能，将打印解析异常的日志。
isBlob	DataHub是否为BLOB类型	否	默认值为false。 <div style="background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>重要</p> <ul style="list-style-type: none"> 仅Blink 3.4.x 及以上版本支持该参数。 使用BLOB类型时，字段需要声明为VARBINARY类型，与METAQ类似。 </div>

类型映射

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

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

属性字段

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

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

Shard数据抽样

指定时间: 2019-02-15 11:11

数量限制: 10

抽样

Shard ID	System Time	name (STRING)	place (STRING)
0	2019-02-15 11:11:46	/abC[Q7	beijing
0	2019-02-15 11:11:46	@*G%T{	guangzhou
0	2019-02-15 11:11:46	XvoedT	guangzhou
0	2019-02-15 11:11:46	%:iHj+O	guangzhou

说明 获取属性字段的方法，请参见 [获取数据源表属性字段](#)。

示例代码

包含DataHub源表的实时计算Flink版作业代码示例如下。

```
create table datahub_input(  
  name VARCHAR  
) with (  
  type='datahub',  
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',  
  project='test1',  
  topic='topic1',  
  accessId='<yourAccessID>',  
  accessKey='<yourAccessSecret>',  
  startTime='2018-06-01 00:00:00'  
);  
create table test_out(  
  name VARCHAR  
) with (  
  type='print'  
);  
INSERT INTO test_out  
SELECT  
  LOWER(name)  
from datahub_input;
```

常见问题

- Q: DataHub和实时计算Flink版中的TIMESTAMP的精确级别不同，如何进行转换？

A: DataHub的TIMESTAMP是精确到微秒，在Unix时间戳里是16位，但实时计算Flink版定义的TIMESTAMP是精确到毫秒，在Unix时间戳里是13位，所以建议您使用BIGINT映射。如果您需要使用TIMESTAMP，建议使用计算列进行转换，详情请参见[计算列](#)，代码示例如下。

```
CREATE TABLE datahub_test (  
  id          VARCHAR,  
  `type`      VARCHAR,  
  `value`     DOUBLE,  
  `time`      BIGINT,  
  ts as TO_TIMESTAMP(`time` / 1000),  
  WATERMARK wk for ts as withoffset(ts,2000)  
) WITH (  
  type = 'datahub'  
);
```

- Q: 分裂或者扩容DataHub Topic后导致实时计算Flink版作业失败，如何恢复？

A: Blink 2.2.0以下版本不支持DataHub Shard扩容和扩容功能，如果分裂或者扩容了某个实时计算正在读取的Topic，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（停止->启动）来使任务恢复正常。

- Q: 可以删除正在使用的DataHub Topic吗？

A: 所有版本均不支持删除或重建正在引用的DataHub Topic。

5.6.2.4. 创建交互式分析Hologres源表

本文为您介绍创建交互式分析Hologres源表DDL定义，以及创建源表时用到的WITH参数、类型映射和代码示例。

使用限制

本文仅适用于Blink 3.6.0及以上版本，如果您的Blink为3.6.0以下的版本，建议您升级到Blink 3.7.0及以上版本。

注意事项

- 在流数据和批数据处理中都可以使用Hologres源表。
- Hologres源表支持Projection Pushdown操作，您可以只读取需要的列。
- Hologres源表使用快照语句高速读取当前数据，读取完后结束作业。如果出现读取数据失败，将重新执行读取操作。
- 并发的Blink作业都可以读取一个或多个Hologres Shard，建议您配置的Blink并发数小于等于Hologres的Shard数。
- 实时消费Hologres源表的数据需要开启Binlog。开启Binlog的方法，请参见 [订阅Hologres Binlog](#)。

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。

DDL定义

```
create table mysource(  
  name varchar,  
  age BIGINT,  
  birthday BIGINT  
) with (  
  type='hologres',  
  dbname='...',  
  tablename='...',  
  username='...',  
  password='...',  
  endpoint='...',  
  field_delimiter='...' --该参数可选。  
);
```

WITH参数

参数	说明	是否必填	备注
type	源表类型	是	固定值为 <i>hologres</i> 。
dbname	数据库名称	是	无
tablename	表名称 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;">? 说明 如果Schema不为Public时，则tableName需要填写为schema.tableName。</div>	是	无

参数	说明	是否必填	备注
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	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。  重要 不能在数据中插入分隔符。	是	默认值为“\u0002”。
bulkread	是否全量读取列存表数据。	否	取值如下： <ul style="list-style-type: none"> • true：全量读取列存表数据。 • false（默认值）：不全量读取列存表数据。

类型映射

Hologres	BLINK
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL

Hologres	BLINK
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

代码示例

包含Hologres源表的实时计算作业代码示例如下。

```
create table mysource(  
  name varchar,  
  age BIGINT,  
  birthday BIGINT  
) with (  
  type='hologres',  
  dbname='...',  
  tablename='...',  
  username='...',  
  password='...',  
  endpoint='...',  
  field_delimiter='...' --该参数可选。  
);  
create table print_output(  
  a varchar,  
  b BIGINT,  
  c BIGINT  
) with (  
  type='print'  
);  
INSERT INTO print_output  
SELECT  
  a, b, c  
from mysource;
```

5.6.2.5. 创建日志服务SLS源表

本文为您介绍如何创建日志服务SLS源表，以及创建过程涉及到的属性字段、WITH参数和类型映射。

 **重要** 本文仅适用于Blink 1.4.5及以上版本。

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，对于日志服务而言，数据格式类似JSON，示例如下。

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

日志服务本身是流数据存储，实时计算Flink版能将其作为流式数据的输入。

语法示例

实时计算Flink版日志服务源表DDL示例如下（代码中的 `sls` 代表日志服务）。

```
create table sls_stream(
  a INT,
  b INT,
  c VARCHAR
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessId>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='<yourProjectName>',
  logStore ='<yourLogStoreName>',
  consumerGroup ='<yourConsumerGroupName>'
);
```

WITH参数

参数	说明	是否必选	备注
type	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	读取的SLS项目名称	是	无
logStore	Project下的具体的LogStore名称	是	无
startTime	日志消费的开始时间	否	无
consumerGroup	消费组名	否	您可以自定义消费组名（没有固定格式）。
heartBeatIntervalMills	消费客户端心跳间隔时间	否	默认值为10000，单位为毫秒。
maxRetryTimes	读取最大尝试次数	否	默认值为5。
batchGetSize	单次读取logGroup的条数	否	默认值为100。

参数	说明	是否必选	备注
columnErrorDebug	是否打开调试开关	否	默认值为false，不打开。如果选择打开，则打印解析异常的日志。
startupMode	启动消费模式	否	<p>取值如下：</p> <ul style="list-style-type: none"> • TIMESTAMP（默认值）：每个Shard从指定时间开始消费。 • Earliest：每个Shard从最早位置开始消费。 • Latest：每个Shard从最新位置开始消费。 • Group_Offsets：每个Shard优先从服务端保存的Checkpoint开始消费，必须指定consumerGroup。消费模式有以下几种情况： <ul style="list-style-type: none"> ◦ 如果从Flink State中恢复状态成功，则从Flink状态中的Checkpoint开始消费。 ◦ 如果从Flink State中恢复状态失败： <ul style="list-style-type: none"> ▪ 如果consumerGroup中存在Checkpoint，则尝试从consumerGroup的Checkpoint开始消费。 ▪ 如果consumerGroup中不存在Checkpoint： <ul style="list-style-type: none"> ▪ 指定了startTime：从startTime开始消费。 ▪ 未指定startTime：每个shard从最早位置开始消费。 <div style="background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p> 重要</p> <ul style="list-style-type: none"> • 仅Blink 3.6.5及以上版本支持该参数。 • 仅当state中不存在Checkpoint时，上述配置才有效。 </div>

② 说明

- 实时计算Flink版1.6.0及以下版本，在指定consumerGroup的Shards数目时，可能会影响读取性能，该缺陷正在修复。
- SLS暂不支持MAP类型的数据。
- SLS对于不存在字段会置为Null。
- 字段顺序支持无序（建议字段顺序和表中定义一致）。
- 输入数据源为JSON形式时，注意定义分隔符，并且需要采用内置函数 `JSON_VALUE` 分析，否则就会解析失败，报错如下。

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

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

类型映射

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

日志服务字段类型	实时计算Flink版字段类型
STRING	VARCHAR

属性字段

目前Flink SQL默认支持3个SLS属性字段的获取，也支持其它自定义字段的写入。属性字段使用方法请参见[获取数据源表属性字段](#)。

字段名	注释说明
<code>__source__</code>	消息源
<code>__topic__</code>	消息主题
<code>__timestamp__</code>	日志时间

代码示例


```
create table sls_input(
  a int,
  b int,
  c varchar
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId =<yourAccessId>,
  accessKey =<yourAccessKey>,
  startTime = '2017-07-05 00:00:00',
  project ='ali-cloud-streamtest',
  logStore ='stream-test',
  consumerGroup ='consumerGroupTest1'
);
create table print_output(
  a int,
  b int,
  c varchar
) with (
  type='print'
);
INSERT INTO print_output
SELECT
  a, b, c
from sls_input;
```

常见问题

- Q: 为什么Job的整体延迟增加，或有窗口聚合的Job无输出？

A: 如果一个Partition没有新数据写入，会导致上述情况，只需要把并发数调整为读写的Partition数量即可。

- Q: 如何设置并发数？


A: 并发数建议等于Partition数量，否则当两个Partition读取速度差异较大时，理论上在追数据场景，存在数据被过滤掉和数据延迟的风险。

- Q: Flink Job延迟增大应该如何排查？

A: SLS源表可能会发生Shard分裂，分裂后的Shard index会不连续，导致Flink延迟增大。如果发现Flink Job延迟增大，请查看SLS源是否发生分裂。

- Q: 如何获取属性字段？

A: 属性字段获取方法，请参见[获取数据源表属性字段](#)。

 **说明** 本地调试时无法抽取到属性字段数据，建议您使用线上调试方法，在日志中进行查看，详情请参见[线上调试](#)。

相关文档

- 日志服务帮助文档，请参见[什么是日志服务](#)。
- 日志服务中实时计算消费文档，请参见[实时计算（Blink）消费](#)。

5.6.2.6. 创建消息队列MQ源表

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


 **说明** 如果您需要使用带独立命名空间的MQ，请使用Blink 3.x作业版本。

什么是消息队列MQ

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

示例


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

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

CSV格式

假设您的1条CSV格式消息记录如下。

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

 **说明** 1条MQ消息可以包括0条到多条数据记录，记录之间使用 `\n` 分隔。

在实时计算作业中，声明MQ数据源表的DDL如下。

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

二进制格式

二进制格式测试代码如下。

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

说明

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

WITH参数

参数	说明	是否必填	备注
type	源表类型	是	固定值为mq。
topic	topic名称	是	无。
endPoint	endPoint地址	是	<p>阿里云消息队列提供内网服务MQ（非公网region）和公网服务MQ（公网region）两种类型，请务必根据您购买的MQ的类型选择对应正确的接入地址（endPoint）：</p> <ul style="list-style-type: none"> • Blink 3.7.10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： <ul style="list-style-type: none"> ◦ 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。 • Blink 3.7.10（不含）以下版本的作业，使用如下接入点： <ul style="list-style-type: none"> ◦ 内网服务MQ（阿里云经典网络/VPC）接入地址： <ul style="list-style-type: none"> ▪ 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）：<code>onsaddr-internal.aliyun.com:8080</code> ▪ 亚太东南1（新加坡）：<code>ap-southeastaddr-internal.aliyun.com:8080</code>。 ▪ 中东东部1（迪拜）：<code>ons-me-east-1-internal.aliyuncs.com:8080</code>。 ▪ 亚太南部1（孟买）：<code>ons-ap-south-1-internal.aliyuncs.com:8080</code>。 ▪ 亚太东南3（吉隆坡）：<code>ons-ap-southeast-3-internal.aliyun.com:8080</code>。 ◦ 公网服务MQ接入地址：<code>onsaddr-internet.aliyun.com:80</code>。

参数	说明	是否必填	备注 说明
			<ul style="list-style-type: none"> 如果您已使用了Blink 3.7.10（不含）以下版本的RocketMQ Connector，则需要将您的实时计算作业升级至Blink 3.7.10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2021年11月1日，RocketMQ旧的接入点将不可用，您需要适配升级实时计算作业。 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。
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无
consumerGroup	订阅消费group名称	是	<ul style="list-style-type: none"> 独享集群默认不能访问公网，如果需要请配置NAT网关，详情参见独享集群如何访问公网？
pullIntervalMs	拉取时间间隔	否	<ul style="list-style-type: none"> 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。 单位为毫秒。
startTime	消息消费启动的时间点	否	无
startMessageOffset	消息开始的偏移量	否	如果填写，将优先以startMessageoffset的位点开始加载。
tag	订阅的标签	否	无
lineDelimiter	解析block时行分隔符	否	默认值为 <code>\n</code> 。
fieldDelimiter	字段分隔符	否	默认值为 <code>\u0001</code> 。表示在只读模式下以 <code>\u0001</code> （ <code>\u0001</code> 在只读模式不可见）作为分隔符，在编辑模式下以 <code>^A</code> 作为分隔符。
encoding	编码格式	否	默认值为 <code>utf-8</code> 。

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： <ul style="list-style-type: none"> 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。 <ul style="list-style-type: none"> 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。 <ul style="list-style-type: none"> 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	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。

类型映射

MQ字段类型	实时计算字段类型
STRING	VARCHAR

5.6.2.7. 创建消息队列Kafka源表

本文为您介绍如何创建实时计算Flink版消息队列Kafka源表、Kafka版本对应关系及Kafka消息解析示例。

重要

- 本文仅适用于Blink 2.0及以上版本。
- 本文仅适用于独享模式。
- Kafka源表支持读取自建Kafka集群，但需要注意版本对应关系，以及自建集群和实时计算Flink版集群的网络环境配置。
- 二进制数据不支持本地调试，语法检查没有问题请进行线上调试，详情请参见 [线上调试](#)。

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。实时计算Flink版支持将Kafka作为流式数据的数据源表或结果表。

从Kafka输出的数据为序列化后的VARBINARY（二进制）格式。对于输出的每条数据，需要您编写自定义表值函数（UDTF）将其解析为序列化前的数据结构。Kafka源表数据解析流程通常为：Kafka Source Table -> UDTF -> Realtime Compute for Apache Flink -> Sink。此外，Flink SQL中也支持通过CAST函数将VARBINARY解析为VARCHAR类型。自定义表值函数请参见[自定义表值函数（UDTF）](#)。

DDL定义

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

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

WITH参数

- 通用配置

参数	注释说明	是否必选	备注
type	Kafka对应版本	是	Kafka版本需要是Kafka08、Kafka09、Kafka010或Kafka011。版本对应关系请参见 Kafka版本对应关系 。
topic	读取的单个topic	是	无
topicPattern	读取一批topic的表达式	否	Topic用竖线 () 分割。例如， <code>topic1 topic2 topic3</code> 。详情请参见 Class Pattern 。

参数	注释说明	是否必选	备注
startupMode	启动位点	否 说明 建议手动配置该参数。	<p>启动位点取值如下：</p> <ul style="list-style-type: none"> GROUP_OFFSETS（默认值）：根据Group读取。 EARLIEST：从Kafka最早分区开始读取。 LATEST：从Kafka最新位点开始读取。 TIMESTAMP：从指定的时间点读取。 <p>您可以在WITH参数中指定 startTimeMs（时间戳）或startTime参数（按照 yyyy-MM-dd HH:mm:ss 格式）。优先使用startTimeMs，不指定则按实时计算Flink版指定的启动位点时间开始消费。</p> <div style="border: 1px solid #ccc; padding: 5px;"> <p>说明</p> <ul style="list-style-type: none"> 如果没有明文指定，则默认为GROUP_OFFSETS模式。 设置为TIMESTAMP模式时，需要在作业参数中明文指定时区。例如 blink.job.timeZone=Asia/Shanghai。 GROUP_OFFSETS模式下，GroupID的第一次消费，没有设置偏移（Offset）值，默认从Kafka分区末尾开始读取数据。 仅Kafka010和Kafka011版本支持TIMESTAMP。 </div>
partitionDiscoveryIntervalMS	定时检查是否有新分区产生	否	<ul style="list-style-type: none"> Kafka 08版本：系统默认开启该功能。 Kafka 09版本及以上版本：不支持partitionDiscoveryIntervalMS参数。您可以通过设置WITH参数，extraConfig='flink.partition-discovery.interval-millis=60000'达到相同的效果。单位为毫秒，默认值为60000，即1分钟。
extraConfig	额外的KafkaConsumer配置项目	否	不在可选配置项中，但是期望额外增加的配置。例如：`fetch.message.max.bytes=104857600`，多个配置使用分号（;）分隔。

• Kafka08配置

◦ Kafka08必选配置

参数	注释说明	是否必选
group.id	消费组ID	是
zookeeper.connect	zk链接地址	是

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

- Kafka09/Kafka010/Kafka011配置

- Kafka09/Kafka010/Kafka011必选配置

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

- Kafka09/Kafka010/Kafka011可选配置，请参见如下Kafka官方文档进行配置：

- Kafka09
- Kafka010
- Kafka011

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

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

Kafka版本对应关系

type	Kafka版本
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2及以上

Kafka消息解析示例

- 场景1：将Kafka中的数据进行计算，并将计算结果输出到RDS。

Kafka中保存了JSON格式数据，需要使用实时计算Flink版进行计算，消息格式示例如下。

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

- 方法1: Kafka SOURCE->Realtime Compute for Apache Flink->RDS SINK

Blink 2.2.7及以上版本支持将VARBINARY类型通过CAST函数转换为VARCHAR类型，再通过JSON_VALUE函数对Kafka数据进行解析，示例如下。

```
CREATE TABLE kafka_src (  
  messageKey VARBINARY,  
  `message` VARBINARY,  
  topic VARCHAR,  
  `partition` INT,  
  `offset` BIGINT  
) WITH (  
  type = 'kafka010' --请参见Kafka版本对应关系。  
)  
);  
CREATE TABLE rds_sink (  
  `name` VARCHAR,  
  age VARCHAR,  
  grade VARCHAR  
) WITH(  
  type='rds'  
);  
CREATE VIEW input_view AS  
SELECT CAST(`message` as VARCHAR) as `message`  
FROM kafka_src;  
INSERT INTO rds_sink  
SELECT  
  JSON_VALUE(`message`, '$.name'),  
  JSON_VALUE(`message`, '$.age'),  
  JSON_VALUE(`message`, '$.grade')  
FROM input_view;
```

- 方法2: Kafka Source->UDTF->Realtime Compute for Apache Flink->RDS Sink

针对不规则数据、复杂JSON数据，需要您自行编写UDTF代码进行解析，示例如下。

■ SQL

```
-- 定义解析Kafka message的UDTF。  
CREATE FUNCTION kafkparser AS 'com.alibaba.kafkaUDTF';  
-- 定义源表。注意：Kafka源表DDL字段必须与以下示例完全一致。WITH中参数可以修改。  
CREATE TABLE kafka_src (  
  messageKey VARBINARY,  
  `message` VARBINARY,  
  topic VARCHAR,  
  `partition` INT,  
  `offset` BIGINT  
) WITH (  
  type = 'kafka010', --请参见Kafka版本对应关系。  
  topic = 'test_kafka_topic',  
  `group.id` = 'test_kafka_consumer_group',  
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'  
);  
CREATE TABLE rds_sink (  
  name VARCHAR,  
  age INT,  
  grade VARCHAR,
```

```
updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourDatabasePassword>'
);
-- 使用UDTF，将二进制数据解析成格式化数据。
CREATE VIEW input_view (
  name,
  age,
  grade,
  updateTime
) AS
SELECT
  T.name,
  T.age,
  T.grade,
  T.updateTime
FROM
  kafka_src as S,
  LATERAL TABLE (kafkparser(`message`)) as T (
  name,
  age,
  grade,
  updateTime
);
-- 使用解析出的格式化数据进行计算，并将结果输出到RDS。
INSERT INTO rds_sink
SELECT
  name,
  age,
  grade,
  updateTime
FROM input_view;
```

■ UDTF

UDTF创建步骤请参见[自定义表值函数（UDTF）](#)。Blink 2.2.4版本Maven依赖，示例如下。

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

```
package com.alibaba;
import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
import java.sql.Timestamp;
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
            /* input message :
            {
                "name":"Alice",
                "age":13,
                "grade":"A",
                "updateTime":1544173862
            }
            */
            String msg = new String(message, "UTF-8");
            try {
                JSONObject data = JSON.parseObject(msg);
                if (data != null) {
                    String name = data.getString("name") == null ? "null" : data.getString("name");
                    Integer age = data.getInteger("age") == null ? 0 : data.getInteger("age");
                    String grade = data.getString("grade") == null ? "null" : data.getString("grade");
                    Timestamp updateTime = data.getTimestamp("updateTime");
                    Row row = new Row(4);
                    row.setField(0, name);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3, updateTime);
                    System.out.println("Kafka message str ==>" + row.toString());
                    collect(row);
                }
            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg + "is not json string");
            }
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
        }
    }
    @Override
    // 如果返回值是Row，重新加载UDTF这个方法，并指明系统返回的字段类型。
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.STRING, DataTypes.INT, DataTypes.STRING, DataTypes.TIMESTAMP);
    }
}
```

- 场景2：从Kafka读取数据，输入实时计算Flink版进行窗口计算。

按照实时计算Flink版目前的设计，滚动或滑动等窗口操作，必须在源表DDL上定义 **Watermark**。Kafka源表比较特殊。如果要以Kafka中message字段中的Event Time进行窗口操作，需先从message字段使用UDX解析出Event Time，才能定义Watermark。在Kafka源表场景中，需要使用**计算列**。例如Kafka中写入数据：2018-11-11 00:00:00|1|Annafemale。计算流程为：Kafka Source->UDTF->Realtime Compute for Apache Flink->RDS Sink。

- 方法1: Kafka SOURCE->Realtime Compute for Apache Flink->RDS SINK

Blink 2.2.7及以上版本支持将VARBINARY类型通过CAST函数转换为VARCHAR类型，再通过JSON_VALUE函数对Kafka数据进行解析，示例如下。

```
CREATE TABLE kafka_src (
  messageKey VARBINARY,
  `message` VARBINARY,
  topic VARCHAR,
  `partition` INT,
  `offset` BIGINT,
  ts as to_timestamp(json_value(cast(`message` as VARCHAR),'$.nodes.time')),
  WATERMARK wk FOR ts as withOffset(ts, 2000)
) WITH (type = 'kafka' --请参见Kafka版本对应关系。);

CREATE TABLE rds_sink (
  starttime TIMESTAMP,
  endtime TIMESTAMP,
  `message` BIGINT
) WITH (type = 'rds');

INSERT INTO rds_sink
SELECT
  TUMBLE_START(ts, INTERVAL '1' MINUTE),
  TUMBLE_END(ts, INTERVAL '1' MINUTE),
  count(`message`)
FROM
  kafka_src
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE);
```

- 方法2: Kafka SOURCE->UDTF->Realtime Compute for Apache Flink->RDS SINK

■ SQL

```
-- 定义解析Kafka message的UDTF。
CREATE FUNCTION kafkapaser AS 'com.alibaba.kafkaUDTF';
CREATE FUNCTION kafkaUDF AS 'com.alibaba.kafkaUDF';
-- 定义源表，注意：Kafka源表DDL字段必须与以下示例一模一样。WITH中参数可改。
create table kafka_src (
  messageKey VARBINARY,
  `message` VARBINARY,
  topic VARCHAR,
  `partition` INT,
  `offset` BIGINT,
  ctime AS TO_TIMESTAMP(kafkaUDF(`message`)), -- 定义计算列，计算列可理解为占位符，源表中并没有这一列，其中的数据可经过下游计算得出。注意:计算列的类型必须为TIMESTAMP才能创建Watermark。
  watermark for `ctime` as withoffset(`ctime`,0) -- 在计算列上定义Watermark。
) WITH (
  type = 'kafka010', -- 请参见Kafka版本对应关系。
  topic = 'test_kafka_topic'
```

```
topic = test_kafka_topic ,
`group.id` = 'test_kafka_consumer_group',
bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);
create table rds_sink (
  `name` VARCHAR,
  age INT,
  grade VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourPassword>'
);
-- 使用UDTF，将二进制数据解析成格式化数据。
CREATE VIEW input_view AS
SELECT
  S.ctime,
  T.`order`,
  T.`name`,
  T.sex
from
  kafka_src as S,
  LATERAL TABLE (kafkapaser (`message`)) as T (
    ctime,
    `order`,
    `name`,
    sex
  );
-- 对input_view中输出的数据做计算。
CREATE VIEW view2 (
  cnt,
  sex
) AS
SELECT
  COUNT(*) as cnt,
  T.sex
from
  input_view
Group BY sex, TUMBLE(ctime, INTERVAL '1' MINUTE);
-- 使用解析出的格式化数据进行计算，并将结果输出到RDS。
insert into rds_sink
SELECT
  cnt,sex
from view2;
```

■ UDF&UDTF

UDF和UDTF创建步骤请参见[自定义标量函数（UDF）](#)和[自定义表值函数（UDTF）](#)。Blink 2.2.4版本Maven依赖，示例如下。


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

- UDTF

```
package com.alibaba;
import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
/**
 以下例子解析输入Kafka中的JSON字符串，并将其格式化输出。
**/
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
            // 读入一个二进制数据，并将其转换为String格式。
            String msg = new String(message, "UTF-8");
            // 提取JSON Object中各字段。
            String ctime = Timestamp.valueOf(data.split("\\\\")[0]);
            String order = data.split("\\\\")[1];
            String name = data.split("\\\\")[2];
            String sex = data.split("\\\\")[3];
            // 将解析出的字段放到要输出的Row()对象。
            Row row = new Row(4);
            row.setField(0, ctime);
            row.setField(1, age);
            row.setField(2, grade);
            row.setField(3, updateTime);
            System.out.println("Kafka message str ==>" + row.toString());
            // 输出一行。
            collect(row);
        } catch (ClassCastException e) {
            System.out.println("Input data format error. Input data " + msg + "is not json string");
        }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
@Override
// 如果返回值是Row，重新加载UDTF这个方法，并指明系统返回的字段类型。
// 定义输出Row()对象的字段类型。
public DataType getResultType(Object[] arguments, Class[] argTypes) {
    return DataTypes.createRowType(DataTypes.TIMESTAMP,DataTypes.STRING, DataTypes.Integer, DataTypes.STRING,DataTypes.STRING);
}
}
```

■ UDF

```
package com.alibaba;
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class KafkaUDF extends ScalarFunction {
    // 可选，open方法可以不写。
    // 需要import org.apache.flink.table.functions.FunctionContext;
    public String eval(byte[] message) {
        // 读入一个二进制数据，并将其转换为String格式。
        String msg = new String(message, "UTF-8");
        return msg.split("\\\\|')[0];
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不写。
    @Override
    public void close() {
    }
}
```

自建Kafka

• 示例

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

• WITH参数

详情请参见 [WITH参数](#)。

? 说明

- bootstrap.servers参数需要填写自建的地址和端口号。
- 仅在Blink 2.2.6及以上版本支持阿里云Kafka或自建Kafka显示TPS和RPS等指标信息。

常见问题

- Q: 作业启动时产生如下报错，应该如何处理？

```

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

```

A: 时区设置错误导致以上报错。请在作业参数中增加如下参数。

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

- Q: 自建Kafka消费不到数据，应该如何排查？

A:

- 问题原因

Kafka各broker会向zookeeper汇报自身Meta信息，Kafka consumer最终会根据broker Meta信息listener_security_protocol_map中的Endpoint地址（例如IP或本机域名加端口）访问broker拉取数据。如果实时计算Flink版作业所在机器无法访问Endpoint地址，则connector中consumer将无法拉取数据，导致流程的停滞。

- 排查思路

- 查看zookeeper > broker > listener_security_protocol_map > Endpoint信息。

```

Command failed: java.lang.IllegalArgumentException: Invalid path string "/ja" caused by invalid character at
get /brokers/ids/0
[{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT":"endpoints":["PLAINTEXT/
c2xid = 0x248
ctime = Tue Aug 27 16:38:14 CST 2019
m2xid = 0x248
etime = Tue Aug 27 16:38:14 CST 2019
p2xid = 0x248
cversion = 0
dataVersion = 0
ackVersion = 0
subscribedMap = 0x16d1ca050000

```

- 通过网络探测功能检测Endpoint的IP或域名是否可访问。

- 登录机器再次确认。

- 解决方案

- 如果Endpoint采用的是IP形式，则确认Kafka服务端是否配置了**数据存储白名单配置**。如果未配置，请配置后重试。
- 如果Endpoint采用的是域名形式，由于实时计算Flink版独享集群内部无法解析域名，所以需要在白名单已配置的情况下，通过以下两种解决方案解决：

- 不能重启Kafka服务

购买**云解析PrivateZone**，配置所有Kafka Broker的域名解析，通过域名进行网络探测成功后，重启实时计算Flink版作业。

- 可以重启Kafka服务

将Kafka的advertised.listeners属性配置为相应Broker的IP和端口（通常为bootstrap.servers中IP和Port），以保证网络畅通，然后重启实时计算Flink版作业。

- Q: 消费Kafka时，为什么作业一启动后马上就结束了？

A: Blink 3.3.0以下版本的Kafka Connector，使用timestamp模式时。如果设置启动位点后，Kafka的所有partition均没数据，则Connector会判定没有partition消费而结束作业。通常对应的 TaskManager.log 日志中会存在类似 Consumer subtask {} initially has no partitions to read from. 的日志信息。建议升级至Blink 3.3.0及以上版本。

- Q: 实时计算Flink版中的COMMIT OFFSET有什么作用？

A: 实时计算Flink版默认采用 commitOffsetOnCheckpointing，用户设置的Commit Offset策略不生效。只有开启checkpoint后，在每次checkpoint成功时，才会commit当前分区消费的offset至Kafka，以便在作业失败恢复过程中，从上一次checkpoint的commit位点开始消费，保证计算的Exactly Once。如果将checkpoint间隔设置过大，Kafka端可能会查询不到当前消费 offset。

5.6.2.8. 创建表格存储Tablestore源表

本文为您介绍如何创建实时计算Flink版表格存储Tablestore源表。

 **重要** 本文仅适用于实时计算Flink版3.2.2及以上版本。

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上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

表格存储通道服务

表格存储通道服务是基于表格存储数据接口的全增量一体化服务，通过Tunnel Service API和SDK，为您提供了增量、全量和增量加全量三种类型的分布式数据实时消费通道。通过Tunnel Service数据通道，您可以使用流式计算的方式，消费表中存量或新增数据。实时计算Flink版可以将Tunnel Service数据通道作为流式数据的输入，每条数据类似一个JSON格式。Tunnel Service数据通道的示例如下。

```
{
  "OtsRecordType": "PUT", // 数据操作类型，包括PUT、UPDATE和DELETE。
  "OtsRecordTimestamp": 1506416585740836, //数据写入时间（单位为微秒），全量数据时为0。
  "PrimaryKey": [
    {
      "ColumnName": "pk_1", //第1主键列。
      "Value": 1506416585881590900
    },
    {
      "ColumnName": "pk_2", //第2主键列。
      "Value": "string_pk_value"
    }
  ],
  "Columns": [
    {
      "OtsColumnType": "PUT", // 列操作类型，包括PUT、DELETE_ONE_VERSION和DELETE_ALL_VERSION
      "ColumnName": "attr_0",
      "Value": "hello_table_store",
    },
    {
      "OtsColumnType": "DELETE_ONE_VERSION", // DELETE操作没有Value字段。
      "ColumnName": "attr_1"
    }
  ]
}
```

DDL定义

实时计算Flink版支持使用Tablestore作为源表，示例代码如下。

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

属性字段

表格存储源表属性字段的获取和使用方法，请参见 [获取数据源表属性字段](#)。

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

WITH参数

参数	说明	备注
type	connector类型	固定值为 <code>ots</code> 。
endPoint	表格存储的实例访问地址	VPC网络环境需要选择实例的VPC Endpoint。
instanceName	表格存储的实例名称	无
tableName	表格存储的数据表名	实时计算读取Tablestore源表数据时，已读取的数据不会再被读取，如果您有再次读取全量数据的需求，则需要重新创建新的数据通道。
tunnelName	表格存储数据表的数据通道名	无
accessId	表格存储读取的AccessKey ID	无
accessKey	表格存储读取的密钥AccessKey Secret	无
ignoreDelete	是否忽略DELETE操作的数据。	可选，默认值为 <code>false</code> 。

5.6.2.9. 创建全量MaxCompute源表

本文为您介绍如何创建全量MaxCompute源表，以及创建源表时使用的WITH参数和类型映射。

重要

- 本文仅适用于Blink 2.2.7及以上版本。
- 与数据总线DataHub、Kafka等数据源不同，全量MaxCompute源表通常作为有限流表使用。Blink 3.4.4版本临时支持将全量MaxCompute源表作为无限流表使用，实现不间断监听并读取新增分区的功能。Blink 3.5.0版本废弃该功能，如果您需要使用无限流MaxCompute源表，请参见[创建增量MaxCompute源表](#)。

DDL定义

实时计算Flink版支持使用全量MaxCompute作为源表输入，示例代码如下。

```
create table odps_source(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  type = 'odps',
  endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
  project = '<projectName>',
  tableName = '<tableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****' --如果您的MaxCompute源表为非分区表，不声明该参数即可。
);
```

WITH参数

参数	说明	是否必填	备注
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。
partition	分区名。	否	<ul style="list-style-type: none"> 只存在一级分区的全量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。 存在多级分区的全量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。

参数	说明	是否必填	备注
subscribeNewPartition	是否监听符合条件的新分区。	否	<p>默认值为false，不监听新产生的分区。</p> <p>说明</p> <ul style="list-style-type: none"> subscribeNewPartition参数值为 true 时，不可以指定partition的参数值，否则会造成无法读取新产生的分区的状况。 该参数只在Blink 3.4.4版本临时存在，Blink 3.5.0版本废弃该参数，请使用创建增量MaxCompute源表。
subscribeIntervalInSec	监听新分区的间隔。	否	<p>默认值为30，单位为秒。</p> <p>说明 监听间隔设置太小，会对全量MaxCompute MetaData服务造成压力，有可能导致监听服务失败。</p>
maxPartitionCount	未设置Partition参数时，读取当前分区表的分区个数。	否	<p>默认值为100。</p> <p>说明 仅Blink 3.0及以上版本支持该参数。</p>

类型映射

全量MaxCompute字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

全量MaxCompute字段类型	实时计算Flink版字段类型
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

代码示例

包含全量MaxCompute源表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE odps_source (  
  cid varchar,  
  rt DOUBLE,  
) with (  
  type = 'odps',  
  endPoint = '<yourEndpointName>',  
  project = '<yourProjectName>',  
  tableName = '<yourTableName>',  
  accessId = '<yourAccessId>',  
  accessKey = '<yourAccessPassword>',  
  partition = 'ds=20190712'  
);  
CREATE TABLE test (  
  cid varchar,  
  invoke_count BIGINT  
) with (  
  type='print'  
);  
INSERT INTO test  
SELECT  
  cid,  
  count(*) as invoke_count  
FROM odps_source GROUP BY cid;
```

常见问题


- Q: DDL中endPoint和tunnelEndpoint参数配置错误，有什么影响？
A: endPoint和tunnelEndpoint参数介绍请参见[各地域Endpoint对照表（外网连接方式）](#)。参数配置错误会导致以下问题：
 - endPoint配置错误：任务上线进度停滞在91%处。
 - tunnelEndpoint配置错误：任务运行失败。
- Q: 全量MaxCompute数据存储底层是如何读取全量MaxCompute源表的？
A: 全量MaxCompute数据存储通过Tunnel读取全量MaxCompute数据的，读取速度及带宽受全量MaxCompute Tunnel带宽限制。
- Q: 实时计算Flink版作业启动后，全量MaxCompute源表是否能读取新写入到全量MaxCompute表或者全量MaxCompute分区的数据？

A: 全量MaxCompute数据存储使用Tunnel读取表数据或者分区数据。实时计算Flink版作业启动后，读取完成Reader后，不再读取新写入全量MaxCompute表和分区的数据。

- Q: 实时计算Flink版作业启动后，全量MaxCompute源表如何能够读取到全量MaxCompute表或者全量MaxCompute分区的数据？

A: 实时计算Flink版 3.4及以上版本支持 subscribeNewPartition参数，控制是否需要监听新产生的分区。新产生的数据可以写入到新的分区，示例代码如下。

```
CREATE TABLE blink_source (
  cid varchar,
  rt DOUBLE,
) with (
  type = 'odps',
  endPoint = '<yourEndpointName>',
  project = '<yourProjectName>',
  tableName = '<yourTableName>',_table_name',
  subscribeNewPartition = 'true'
  --请注意：如果需要监听新的分区，则不能指定partition值。
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
);
CREATE TABLE test (
  cid varchar,
  invoke_count BIGINT
) with (
  type='print'
);
INSERT INTO test
SELECT
  cid,
  count(*) as invoke_count
FROM blink_source GROUP BY cid;
```

 说明 全量MaxCompute表新分区产生的数据需要写入实时计算Flink版全量MaxCompute源表的新分区。如果写入现有分区，则不生效。

- Q: 如何查看全量MaxCompute源表的分区名？

A: 您可以在数据表详情页面查看全量MaxCompute源表的分区名，步骤如下：

- 在数据地图，搜索表名称。
- 单击目标表名。
- 在数据表详情页面，右侧 明细信息 > 分区信息 > 分区名查看全量MaxCompute分区名。

- Q: WITH参数中partition值是否可以使用 max_pt() 或者 max_pt_with_done() ？

A: 不推荐使用。如果需要使用，请务必了解在没有开启监听新分区和开启监听新分区的场景下 max_pt() 在全量MaxCompute源表的作用：

- 未开启监听新分区

任务启动后，会通过全量MaxCompute MetaData服务获取当前全量MaxCompute 源表所有分区列表，从中读取 max_pt() 。读取结束后，Reader退出。不再读取该分区中的新数据或者监听新分区。

- 开启监听新分区

任务启动后，会通过全量MaxCompute MetaData服务获取当前全量MaxCompute源表所有分区列表，从中读取 `max_pt()`。读取结束后，不再读取该分区中的新数据。间隔一定周期（参见`subscribeIntervalInSec`参数）监听是否有新分区产生。如果有新分区产生，读取新产生的分区列表，从中获取 `max_pt()`，并读取。读取完成后还会等待下次监听事件。如果没有新分区产生，等待下次监听事件。

- Q: 提交任务阶段出现Akka超时报错，但是全量MaxCompute源表获取Metadata速率正常，应该如何处理？

A: 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](#)。

- Q: 引用全量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读取？

A: 数据不会被读取，而且可能导致作业失败。全量MaxCompute数据存储使用 `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` 那一瞬间数据的映射，后续的数据读取以这个映射为基础。因此在新建 `DOWNLOAD_SESSION` 后，追加到全量MaxCompute表或者分区里的数据，正常流程下是不会被读取的。分为两种情况：

- Tunnel在读取数据的过程中，写入新数据会产生报错 `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.`
- Tunnel已经关闭，写入新数据。这些数据不会被读取的，因为Tunnel已经关闭。而且一旦作业发生Failover或者暂停恢复作业后，不能保证数据正确性，例如，已经读过的数据可能会重读，新追加的数据可能读不全。

- Q: 全量MaxCompute源表作业是否支持暂停、恢复和修改源表的并发度？

A: 全量MaxCompute源表作业不支持暂停、恢复和修改源表并发度。系统根据并发度，计算每个并发应该读哪些分区的哪一段内容，并且每个并发会把自己消费的情况记录到State里。以便暂停恢复后或者Failover后，可以从上次读取的位置继续读取。这个逻辑是建立在并发度确定的前提下的。如果修改了全量MaxCompute源表的并发度后进行暂停恢复操作，对作业产生的影响是无法预估的，因为已经读取的数据可能会再次读取，没有读的数据反而被遗漏。

- Q: 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读取？

A: 设置启动位点只对消息队列类型的数据源有效（例如DataHub），对全量MaxCompute源表无效。实时计算Flink版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。

- Q: 作业运行过程中报错`ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:****'`

A: 该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见[用户认证](#)。

5.6.2.10. 创建增量MaxCompute源表

本文为您介绍如何创建增量MaxCompute源表，以及创建维表时使用的WITH参数、类型映射和常见问题。

 重要

- 本文仅适用于Blink 3.5.0 hotfix及以上版本。
- 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。

语法示例

```
create table odps_source(  
  id int,  
  user_name VARCHAR,  
  content VARCHAR  
) with (  
  type = 'continuous-odps',  
  endPoint = 'your_end_point_name',  
  project = 'your_project_name',  
  tableName = 'your_table_name',  
  accessId = 'your_access_id',  
  accessKey = 'your_access_key',  
  startPartition = 'ds=20180905'  
);
```

WITH参数

参数	说明	是否必填	备注
type	connector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	<ul style="list-style-type: none">• 是：VPC环境必填• 否：其他非VPC环境	请参见Endpoint。
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	<p>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。</p> <p>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。</p> <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>说明</p> <ul style="list-style-type: none"> 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。 </div>	是	<p>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。</p> <p>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：</p> <ul style="list-style-type: none"> ds=20191201,type=a ds=20191201,type=b ds=20191202,type=a ds=20191202,type=b ds=20191202,type=c <p>不同startPartition，满足分区的列表如下：</p> <ul style="list-style-type: none"> ds=20191202 <ul style="list-style-type: none"> ds=20191202,type=a ds=20191202,type=b ds=20191202,type=c ds=20191201,type=c <ul style="list-style-type: none"> ds=20191202,type=a ds=20191202,type=b ds=20191202,type=c

类型映射

MaxCompute字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP

MaxCompute字段类型	实时计算Flink版字段类型
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

 重要

- 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 TABLE odps_source (  
  cid VARCHAR,  
  rt DOUBLE,  
) with (  
  type = 'continuous-odps',  
  endPoint = 'your_end_point_name',  
  project = 'your_project_name',  
  tableName = 'your_table_name',  
  accessId = 'xxxx',  
  accessKey = 'xxxx',  
  startPartition = 'ds=20191201'  
);  
CREATE TABLE test (  
  cid VARCHAR,  
  rt DOUBLE,  
) with (  
  type='print'  
);  
INSERT INTO test  
SELECT  
  cid, rt FROM odps_source;
```

常见问题

- Q: DDL中endPoint和tunnelEndpoint参数配置错误的影响?

A: endPoint和tunnelEndpoint参数介绍请参见 [各地域Endpoint对照表（外网连接方式）](#)。参数配置错误会导致以下问题：

- endPoint配置错误：任务上线进度停滞在91%处。
- tunnelEndpoint配置错误：任务运行失败。

- Q: 增量MaxCompute数据存储底层是如何读取MaxCompute表的?

A: 增量MaxCompute数据存储是通过tunnel读取MaxCompute数据，读取速度及带宽受MaxCompute tunnel带宽限制。

- Q: 实时计算Flink版作业启动后，增量MaxCompute源表是否能读取新写入到已经读取过的分区的数据？

A: 增量MaxCompute数据存储使用Tunnel读取MaxCompute源表分区的数据。实时计算Flink版作业启动后，读取完成Reader就退出，不会读取新写入MaxCompute源表已读取分区的数据。

- Q: 如何查看MaxCompute分区名？

A: 您可以在数据表详情中查看MaxCompute分区名，步骤如下：

- i. 在[数据地图](#)，搜索表名称。
- ii. 在所有表区域，单击目标表名。
- iii. 在数据表详情页面，右侧**明细信息 > 分区信息 > 分区名**查看MaxCompute分区名。

- Q: 提交任务阶段出现Akka超时报错，但是增量MaxCompute源表获取Metadata速率正常，应该如何处理？

A: 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](#)。

- Q: 引用增量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读取？

A: 数据不会被读取，而且可能导致作业失败。MaxCompute数据存储使用 `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` 那一瞬间数据的映射，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。分为两种情况：

- Tunnel在读取数据的过程中，写入新数据会产生报错 `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.`。
 - Tunnel已经关闭，写入新数据。这些数据不会被读取的，因为Tunnel已经关闭。而且一旦作业发生Failover或者暂停恢复作业后，不能保证数据正确性（例如已经读过的数据可能会重读，新追加的数据可能读不全）。
- Q: 增量MaxCompute源表作业是否支持暂停、恢复和修改源表的并发度？

增量MaxCompute源表作业暂不支持暂停、恢复和修改源表并发度。系统根据并发度，计算每个并发读取指定分区的指定数据。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。

如果修改了增量MaxCompute源表的并发度后暂停恢复作业，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。

- Q: 监听到新分区的时候，如果该分区还有数据没有写完，如何处理？

A: 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，表T的写入过程分为以下两种方式：

- 不允许的写入方法：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。
 - 推荐的写入方法：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。
- Q: 作业运行过程中报错`ErrorMessage=Authorization Failed [4019], You have NO`


```
privilege'ODPS:****'
```

A: 该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见[用户认证](#)。

5.6.3. 创建数据结果表

5.6.3.1. 数据结果表概述

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

实时计算结果表有Append类型和Update类型。

- Append类型：如果输出存储是日志系统、消息系统、或未定义主键的RDS数据库，数据流的输出结果会以追加的方式写入到存储中，不会修改存储中原有的数据。
- Update类型：如果输出存储是声明了主键（primary key）的数据库（例如，RDS和HBase），数据流的输出结果有以下2种情况。
 - 如果根据主键查询的数据在数据库中不存在，则会将该数据插入数据库。
 - 如果根据主键查询的数据在数据库中不存在，则会根据主键更新数据。

结果表语法

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

结果表示例

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

5.6.3.2. 创建Oracle数据库结果表

本文为您介绍如何创建Oracle结果表，以及创建结果表时使用的WITH参数、类型映射和注意事项。

重要

- 本文仅适用于Blink 3.6.0及以上版本。
- 仅支持使用Oracle 19c版本创建Oracle数据库结果表。

注意事项

- 实时计算Flink版将每行结果数据拼接成一行SQL语句写入目标数据库。
- 根据数据库中是否存在需要的表，执行以下操作：
 - 当数据库中不存在需要的表时，Oracle结果表会新建一个用于写入结果的表。
 - 当数据库中不存在需要的表时，Oracle结果表会向该表中写入或者更新数据。
- 逻辑表和物理表的主键不一定完全相同，要求逻辑表的主键包含物理表的主键。
- 使用以下两种函数处理Oracle数据库结果表的数据：
 - 如果没有定义主键，则执行Append函数向表中插入数据。
 - 如果已经定义主键，则执行Upsert函数向表中插入数据。当主键不存在时，则插入主键；当主键存在时，则更新主键。

语法示例

实时计算Flink版支持使用Oracle数据库作为结果表，代码示例如下。

```
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT,
  PRIMARY KEY(employee_id)
) WITH (
  type = 'oracle',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>'
);
```

WITH 参数

参数	描述	是否必选	示例值
type	结果表类型	是	固定值为 <i>oracle</i> 。
url	数据库连接串	是	jdbc:oracle:thin:@192.168.171.62:1521:sit0
userName	登录数据库的用户名	是	无
password	登录数据库的密码	是	无
tableName	数据库的表名	是	无
maxRetryTimes	向结果表插入数据的最大尝试次数	否	默认值为10。

参数	描述	是否必选	示例值
batchSize	<p>单次写入数据的批次大小</p> <p>说明</p> <ul style="list-style-type: none"> 指定主键后 batchSize 参数才生效。 batchSize 或 bufferSize 参数达到阈值时都会触发数据写入。 	否	默认值为50。
bufferSize	<p>去重的缓存大小</p> <p>说明</p> <ul style="list-style-type: none"> 指定主键后 bufferSize 参数才生效。 batchSize 或 bufferSize 参数达到阈值时都会触发数据写入。 	否	默认值为500。
flushIntervalMs	<p>写超时时间，单位为毫秒。</p> <p>说明 如果在写超时时间内没有向数据库写入数据，系统会将缓存的数据再写一次。</p>	否	默认值为500。
excludeUpdateColumns	<p>更新表的某行数据时，是否不更新该行指定的列数据。</p> <p>说明 更新表的某行数据时，默认不更新主键数据。</p>	否	默认不填写。
ignoreDelete	是否忽略删除操作。	否	默认值为false。

类型映射

Oracle 字段类型	实时计算Flink版字段类型
<ul style="list-style-type: none"> CHAR VARCHAR VARCHAR2 	VARCHAR

Oracle 字段类型	实时计算Flink版字段类型
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

代码示例

实时计算Flink版包含Oracle数据库结果表的代码示例如下。

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT
) WITH (
  type = 'random'
);
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT,
  primary key(employee_id)
)with(
  type = 'oracle',
  url = 'jdbc:oracle:thin:@192.168.171.62:1521:sit0',
  userName = 'blink_test',
  password = 'blink_test',
  tableName = 'oracle_sink'
);
INSERT INTO oracle_sink
SELECT * FROM oracle_source;
```

5.6.3.3. 创建交互式分析Hologres结果表

本文为您介绍如何创建交互式分析Hologres结果表，以及创建结果表时使用的WITH参数、流式语义和类型映射。

重要

- 本文仅适用于Blink 3.6.0及以上版本，如果您的Blink为3.6.0以下的版本，您可以升级Blink版本至3.6.0及以上版本，详细请参见[管理独享集群Blink版本](#)。

下载并安装使用 [blink-connector-hologres-blink-3.6.8.jar](#)或[blink-connector-hologres-blink-3.7.jar](#)包。

- 建议您使用Hologres 0.7及以上版本。
- 由于Hologres是异步写入数据的，因此需要添加**blink.checkpoint.fail_on_checkpoint_error=true** 作业参数，作业异常时才会触发Failover。

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。

语法示例

实时计算Flink版支持使用Hologres作为结果表，代码示例如下。

```
create table Hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='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
type	结果表类型	是	固定值为 <i>hologres</i> 。
dbname	数据库名称  说明 如果Schema不为Public时，则tableName需要填写为schema.tableName。	是	无
tablename	表名称	是	无
username	用户名	是	无
password	密码	是	无
endpoint	Hologres VPC 端点信息	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。  重要 不能在数据中插入分隔符，且需与bulkload语义一同使用。	否	默认值为 <code>"\u0002"</code> 。

参数	说明	是否必填	备注
mutateType	流式写入语义，详情请参见 流式语义 。	否	默认值为 <code>insertorignore</code> 。
partitionrouter	分区表写入	否	默认值为 <code>false</code> 。 ⓘ 说明 3.6.x版本 Hologres Sink暂不支持自动创建分区表。因此，在写入分区表之前，需要您在 Hologres中手工创建对应的子表。
ignoredelete	是否忽略撤回消息。	否	默认值为 <code>false</code> 。 ⓘ 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	<ul style="list-style-type: none"> • <code>false</code>（默认值）：不会自动创建。 • <code>true</code>：自动创建。 ⓘ 说明 仅 Blink 3.7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结果表中使用时语义，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：使用后续出现的数据整行替换已有数据。
- insertorupdate：使用后续出现的数据选择性替换已有数据。

说明

- 当mutateType设置为insertorupdate或insertorreplace时，系统根据主键更新数据。
- Blink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。
 - Blink Connector不会自动创建分区表，因此，需要您提前手动创建需要导入数据的分区表，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。

类型映射

Hologres	BLINK
INT	INT
INT[]	ARRAY<INT>

Hologres	BLINK
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

5.6.3.4. 创建云原生数据仓库AnalyticDB MySQL版2.0结果表

本文为您介绍如何创建云原生数据仓库AnalyticDB MySQL版2.0结果表，以及云原生数据仓库AnalyticDB MySQL版和实时计算Flink版字段类型之间的映射关系。

重要

- 本文仅适用于Blink 1.4.5及以上版本。
- 云原生数据仓库AnalyticDB MySQL版2.0数据库支持自增主键。如果实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段。例如，ID是自增字段，实时计算Flink版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。

什么是云原生数据仓库AnalyticDB MySQL版

云原生数据仓库AnalyticDB MySQL版是阿里巴巴自主研发的海量数据实时高并发在线分析（Realtime OLAP）云计算服务，支持在毫秒级单位时间内，对千亿级数据进行实时地多维分析透视和业务探索。

DDL定义

 说明 云原生数据仓库AnalyticDB MySQL版3.0结果表DDL请参见 [创建云原生数据仓库AnalyticDB MySQL版3.0结果表](#)。

实时计算Flink版支持使用云原生数据仓库AnalyticDB MySQL版2.0作为结果输出。示例代码如下。

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

说明

- 建议使用存储注册功能，详情请参见 [注册云原生数据仓库AnalyticDB MySQL版](#)。
- 云原生数据仓库AnalyticDB MySQL版结果表声明中的主键要和云原生数据仓库AnalyticDB MySQL版里的主键保持一致，大小写敏感。如果不一致，则会导致数组索引越界的异常现象。

WITH参数

参数	说明	备注
type	结果表类型	固定值为ads。
url	JDBC连接地址	<p>云原生数据仓库AnalyticDB MySQL版数据库地址。示例：<code>url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'</code>。其中参数解释如下：</p> <ul style="list-style-type: none"> • URI地址查询步骤如下： <ol style="list-style-type: none"> 登录 AnalyticDB控制台。 单击对应的集群名称，进入 基本信息 页面。 在连接信息中查看相应的连接地址。 • 云原生数据仓库AnalyticDB MySQL版数据库名称（databaseName）即云原生数据仓库AnalyticDB MySQL版实例名称。
tableName	表名	无
username	账号	无
password	密码	无
maxRetryTimes	写入重试次数	可选，默认值为10。

参数	说明	备注
bufferSize	流入多少条数据后开始去重	可选，默认值为5000，表示输入的数据达到5000条就开始输出。
batchSize	一次批量写入的条数	可选，默认值为1000。 ? 说明 云原生数据仓库AnalyticDB MySQL版批量写入数据的总大小不能超过1 MB，否则会报错误码20015。如果遇到该错误码，您需要调小batchSize的参数值。
batchWriteTimeoutMs	写入超时时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
connectionMaxActive	单连接池最大连接数	可选，默认值为30。
ignoreDelete	是否忽略delete操作	默认值为false。

类型映射

建议使用云原生数据仓库AnalyticDB MySQL版和实时计算Flink版字段类型对应关系进行DDL声明。

云原生数据仓库AnalyticDB MySQL版字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
TINYINT	INT
SMALLINT	
INT	
BIGINT	BIGINT
DOUBEL	DOUBLE
VARCHAR	VARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

5.6.3.5. 创建数据总线DataHub结果表

本文为您介绍如何创建实时计算数据总线DataHub结果表，以及创建结果表时使用的WITH参数和类型映射。

 **重要** 本文仅适用于Blink 1.4.5及以上版本。

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。实时计算Flink版使用DataHub作为流式数据存储源头或输出目的端。

DDL定义

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

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

 **说明** 建议您使用存储注册功能，详情请参见 [注册数据总线DataHub](#)。

WITH参数

参数	参数说明	是否必填	备注
type	源表类型	是	固定值为 <code>datahub</code> 。
endPoint	endPoint地址	是	DataHub的endPoint地址详情，请参见 DataHub域名列表 。
project	DataHub项目名称	是	无
topic	DataHub中Topic名称	是	无
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无
maxRetryTimes	读取最大重试次数	否	不同版本中，该参数的默认值如下： <ul style="list-style-type: none">Blink 2.2.7以下版本：3Blink 2.2.7及以上版本：20

参数	参数说明	是否必填	备注
batchSize	一次批量写入的条数	否	不同版本中，该参数的默认值如下： <ul style="list-style-type: none"> Blink 3.3以下版本：300 Blink 3.3及以上版本：100
batchWriteTimeoutMs	缓存数据的超时时间	否	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
maxBlockMessages	每次写入的最大Block数	否	默认值为100。
reserveMillisecond	TIMESTAMP类型是否保留毫秒	否	默认值为false。 <div style="border: 1px solid #add8e6; padding: 5px; margin-top: 5px;">  重要 仅实时计算2.2.6以上版本支持该功能。 </div>
partitionBy	写入结果表前会根据该值进行Hash分类，数据会流向对应的结果表。	否	默认值为空，随机进行数据分配。 <div style="border: 1px solid #add8e6; padding: 5px; margin-top: 5px;">  说明 partitionBy决定数据流到Blink的哪个Subtask。 </div>
hashFields	指定了列名之后，相同列的值会写入到同一个Shard。	否	默认值为Null，即随机写入。可以指明多个列值，用逗号（,）分隔。例如， <code>hashFields='a,b'</code> 。 <div style="border: 1px solid #add8e6; padding: 5px; margin-top: 5px;">  重要 <ul style="list-style-type: none"> 仅Blink 3.3.0及以上版本支持该功能。 hashFields决定数据流写到DataHub的哪个Shard。 </div>

类型映射

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

DataHub字段类型	实时计算字段类型
BIGINT	BIGINT

DataHub字段类型	实时计算字段类型
TIMESTAMP	<p>BIGINT</p> <p>② 说明 DataHub的TIMESTAMP精确到微秒，在Unix时间戳中为16位，但实时计算定义的TIMESTAMP精确到毫秒，在Unix时间戳中为13位，所以建议您使用BIGINT进行映射。</p> <p>如果您需要使用TIMESTAMP，建议使用计算列进行转换，详情请参见 计算列。</p>
STRING	VARCHAR
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL

代码示例

包含DataHub结果表的实时计算作业代码示例如下。

```
create table datahub_input(
  name VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='test1',
  topic='topic1',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2018-06-01 00:00:00'
);
create table datahub_output(
  name varchar
)with(
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='test2',
  topic='topic2',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  batchSize='1000',
  batchWriteTimeoutMs='500'
);
INSERT INTO datahub_output
SELECT
  LOWER(name)
from datahub_input;
```

5.6.3.6. 创建日志服务SLS结果表

本文为您介绍如何创建实时计算Flink版日志服务SLS结果表。

重要

- 本文仅适用于Blink 1.4.5及以上版本。
- 日志服务SLS结果表仅支持VARCHAR类型的字段。


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，实时计算Flink版能将其作为流式数据的输入。

DDL定义

实时计算Flink版支持使用日志服务作为结果输出。日志服务结果表声明示例如下。

```
create table sls_stream(
  `name` VARCHAR,
  age BIGINT,
  birthday BIGINT
)with(
  type='sls',
  endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
  accessId='<yourAccessId>',
  accessKey='<yourAccessKey>',
  project='<yourProjectName>',
  logstore='<yourLogstoreName>'
);
```

 **说明** 建议使用日志服务存储注册功能，详情请参见 [注册日志服务SLS](#)。

WITH参数

参数	注释说明	是否必填	备注
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
topic	属性字段	否	默认值为空，可以将选定的字段作为属性字段 topic 填充。

参数	注释说明	是否必填	备注
timestampColumn	属性字段	否	默认值为空，可以将选定的字段作为属性字段 timestamp 填充（类型必须为INT）。如果未指定，则默认填充当前时间。
source	属性字段。日志的来源地，例如产生该日志机器的IP地址。	否	默认值为空，可以将选定的字段作为属性字段 source 填充。
partitionColumn	分区列	否	如果 mode 为 partition，则该参数必填。
flushIntervalMs	触发数据写入的周期	否	默认值为2000，单位为毫秒。
reserveMilliSecond	TIMESTAMP类型是否保留毫秒值。	否	默认值为false，不保留。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; border-radius: 5px;"> <p>? 说明 实时计算Flink版2.2.6及以上版本支持该参数。</p> </div>

类型映射

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

日志服务字段类型	实时计算Flink版字段类型
STRING	VARCHAR

代码示例

包含日志服务SLS结果表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE random_input (
  a VARCHAR,
  b VARCHAR) with (
  type = 'random'
);
create table sls_output(
  a varchar,
  b varchar
)with(
  type='sls',
  endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
  accessId='<yourAccessId>',
  accessKey='<yourAccessKey>',
  project='ali-cloud-streamtest',
  logStore='stream-test2'
);
INSERT INTO sls_output
SELECT a, b
FROM random_input;
```

常见问题

Q: 如何配置输出结果表中的 topic?

A: 需要配置topic为结果表中的一个字段。例如，本文示例代码中设置 `topic='age'`。配置完成后，结果值中 `age` 字段的值会写入日志服务，同时日志服务不会把 `age` 字段写入下游。

相关文档

- 日志服务帮助文档，请参见 [什么是日志服务](#)。
- 日志服务中实时计算消费文档，请参见 [实时计算（Blink）消费](#)。

5.6.3.7. 创建消息队列MQ结果表

本文为您介绍如何创建实时计算Flink版消息队列MQ结果表，以及创建过程涉及到的WITH参数。

重要

- 本文仅适用于Blink 1.4.5及以上版本。
- 如果您需要使用带独立命名空间的MQ，请使用Blink 3.x作业版本。

什么是消息列队MQ

消息队列MQ（Message Queue）是阿里云商用的专业消息中间件，是企业级互联网架构的核心产品。消息队列基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）和监控报警等一套完整的消息云服务。

CSV格式

实时计算Flink版可以将消息队列作为流式数据进行输出，CSV格式输出示例如下。

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


二进制格式

实时计算Flink版可以将消息队列作为流式数据进行输出，二进制格式输出示例如下。


```

CREATE TABLE source_table (
  commodity VARCHAR
)WITH(
  type='random'
);
CREATE TABLE result_table (
  mess VARBINARY
) WITH (
  type = 'mq',
  endpoint='<yourEndpoint>',
  accessID='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic='<yourTopicName>',
  producerGroup='<yourGroupName>'
);
INSERT INTO result_table
SELECT
CAST(SUBSTRING(commodity,0,5) AS VARBINARY) AS mess
FROM source_table;

```

 **说明** `cast(varchar as varbinary)` 需要在Blink 2.0及以上版本使用。如果版本低于2.0，请先完成版本升级，详情请参见 [管理独享集群Blink版本](#)。

WITH参数

参数	说明	备注
type	结果表类型	固定值为mq。
topic	Message Queue队列名称	无

参数	说明	备注
endpoint	地址	<p>阿里云消息队列提供内网服务MQ（非公网region）和公网服务MQ（公网region）两种类型，请务必根据您购买的MQ的类型选择对应正确的接入地址：</p> <ul style="list-style-type: none"> Blink 3.7.10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： <ul style="list-style-type: none"> 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。 Blink 3.7.10（不含）以下版本的作业，使用如下接入点： <ul style="list-style-type: none"> 内网服务MQ（阿里云经典网络/VPC）接入地址： <ul style="list-style-type: none"> 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）：<code>onsaddr-internal.aliyun.com:8080</code> 亚太东南1（新加坡）：<code>ap-southeastaddr-internal.aliyun.com:8080</code>。 中东东部1（迪拜）：<code>ons-me-east-1-internal.aliyuncs.com:8080</code>。 亚太南部1（孟买）：<code>ons-ap-south-1-internal.aliyuncs.com:8080</code>。 亚太东南3（吉隆坡）：<code>ons-ap-southeast-3-internal.aliyun.com:8080</code>。 公网服务MQ接入地址：<code>onsaddr-internet.aliyun.com:80</code>。 <div style="background-color: #e6f2ff; padding: 10px; margin-top: 10px;"> <p>重要</p> <ul style="list-style-type: none"> 如果您已使用了Blink 3.7.10（不含）以下版本的RocketMQ Connector，则需要将您的实时计算作业升级至Blink 3.7.10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2021年11月1日，RocketMQ旧的接入点将不可用，您需要适配升级实时计算作业。 内网服务无法跨地域访问。例如，您所购买的实时计算Flink版服务的地域为华东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。 独享集群默认不能访问公网，如果需要请配置NAT网关，详情参见独享集群如何访问公网？ 由于阿里云网络安全策略动态变化，实时计算Flink版连接公网服务MQ时可能会出现网络连接问题，推荐您使用内网服务MQ。 </div>
accessID	AccessKey ID	无
accessKey	AccessKey Secret	无

参数	说明	备注
produce rGroup	写入的群组	无
tag	写入的标签	可选，默认值为空。
fieldDeli miter	字段分割符	可选，默认值为 <code>\u0001</code> 。分隔符的使用情况如下所示： <ul style="list-style-type: none"> 只读模式：以 <code>\u0001</code> 作为分隔符，<code>\u0001</code> 在只读模式不可见。 编辑模式：以 <code>^A</code> 作为分隔符。
encodin g	编码类型	可选，默认值为 <code>utf-8</code> 。
retryTim es	写入的重试次数	可选，默认值为10。
sleepTi meMs	重试间隔时间	可选，默认值为1000（毫秒）。
instance ID	MQ实例ID	<ul style="list-style-type: none"> 如果MQ实例无独立命名空间，则不可以使用 <code>instanceID</code>参数。 如果MQ实例有独立命名空间，则 <code>instanceID</code>参数必选。

5.6.3.8. 创建表格存储Tablestore结果表

本文为您介绍如何创建实时计算Flink版表格存储结果表，以及表格存储和实时计算Flink版字段类型之间的映射关系。

 **重要** 本文仅适用于Blink 1.4.5及以上版本。

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上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

DDL定义

实时计算Flink版支持使用Tablestore作为结果输出，示例代码如下。

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

说明

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


WITH参数

参数	说明	备注
type	结果表类型	固定值为ots。
instanceName	实例名	无
tableName	表名	无
endPoint	实例访问地址	参见 服务地址 。
accessId	AccessKey ID	无
accessKey	AccessKey Secret	无
valueColumns	指定插入的字段列名	插入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	可选，默认值为5000，表示输入的数据达到5000条就开始输出。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> <p>说明 在实时计算Flink版系统，bufferSize根据Tablestore主键对结果数据进行去重后，再在bufferSize的基础上进行batchSize。</p> </div>

参数	说明	备注
batchWriteTimeoutMs	写入超时的时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
batchSize	一次批量写入的条数	可选，默认值为100。
retryIntervalMs	重试间隔时间	可选，单位毫秒，默认值为1000。
maxRetryTimes	最大重试次数	可选，默认值为100。
ignoreDelete	是否忽略DELETE操作	默认值为False。

类型映射

Tablestore字段类型	实时计算Flink版字段类型
INTEGER	BIGINT
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

 说明 Tablestore结果表必须定义有 Primary Key ，以Update方式写入结果数据到Tablestore表。Update方式说明请参见 [Update类型](#)。

5.6.3.9. 创建云数据库RDS MySQL版结果表

本文为您介绍如何创建实时计算云数据库RDS MySQL版结果表，以及创建结果表时使用的WITH参数和类型映射。

什么是云数据库RDS MySQL版

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。详情请参见[概述](#)。

使用限制

实时计算Flink版暂不支持通过数据存储功能中存储注册的方式使用RDS MySQL 8.0版本，建议您使用明文方式使用RDS MySQL 8.0版本。数据存储功能详情请参见[概述](#)。

语法示例

实时计算Flink版支持使用RDS MySQL版作为结果输出，示例代码如下。

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

说明

- 实时计算Flink版写入RDS MySQL数据库结果表原理：针对实时计算Flink版每行结果数据，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。如果实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段即可。例如，ID是自增字段，实时计算Flink版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，拆分键必须在实时计算Flink版DDL里 PRIMARY KEY () 中声明，否则拆分的表无法写入。
- 建议使用数据存储注册方式，请参见 [注册云数据库RDS版](#)。
- 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	是否必填	备注
type	结果表类型	是	固定值为 <code>rds</code> 。
url	JDBC (Java DataBase Connectivity) 连接地址	是	URL的格式为： <code>jdbc:mysql://<内网地址>/<databaseName></code> ，其中databaseName为对应的数据库名称。内网地址参见如下链接： <ul style="list-style-type: none"> RDS的内网地址 DRDS的内网地址 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>说明 如果访问通过VPC访问授权过的RDS，对应的URL配置请参见VPC访问授权。</p> </div>
tableName	表名	是	无
userName	用户名	是	无

参数	说明	是否必填	备注
password	密码	是	无
maxRetry Times	最大重试次数	否	默认值为10。
batchSize	一次批量写入的条数	否	默认值为4096。
bufferSize	流入多少条数据后开始去重	否	默认值为10000。
flushIntervalMs	清空缓存的时间间隔	否	默认值为2000，单位为毫秒。表示如果缓存中的数据在等待2秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
excludeUpdateColumns	忽略指定字段的更新	否	默认值为空（默认忽略PRIMARY KEY字段）。表示更新主键值相同的数据时，忽略指定字段的更新。
ignoreDelete	是否忽略delete操作	否	默认值为false，表示支持delete功能。
partitionBy	分区	否	默认为空。表示写入Sink节点前，会根据该值进行Hash分区，数据会流向相应的Hash节点。

类型映射

RDS字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

RDS字段类型	实时计算Flink版字段类型
VARBINARY	VARBINARY

JDBC连接参数

参数名称	说明	默认值	最低版本要求
useUnicode	是否使用Unicode字符集，如果参数characterEncoding设置为GB2312或GBK，本参数值必须设置为true。	false	1.1g
characterEncoding	当useUnicode设置为true时，指定字符编码。例如可设置为GB2312或GBK。	false	1.1g
autoReconnect	当数据库连接异常中断时，是否自动重新连接。	false	1.1
autoReconnectForPools	是否使用针对数据库连接池的重连策略。	false	3.1.3
failOverReadOnly	自动重连成功后，连接是否设置为只读。	true	3.0.12
maxReconnects	autoReconnect设置为true时，重试连接的次数。	3	1.1
initialTimeout	autoReconnect设置为true时，两次重连之间的时间间隔，单位为秒。	2	1.1
connectTimeout	和数据库服务器建立socket连接时的连接超时时长，单位为毫秒。0表示永不超时，适用于JDK 1.4及以上版本。	0	3.0.1
socketTimeout	socket操作（读写）超时，单位：毫秒。0表示永不超时。	0	3.0.1

代码示例

包含云数据库RDS版结果表的实时计算Flink版作业代码示例如下。


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

FAQ

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

A: 如果在DDL中定义了主键，会采用 `INSERT INTO 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。

- Q: 使用RDS表中的唯一索引进行GROUP BY时需要注意什么？

A:

- 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，实时计算Flink版作业中不能声明为PRIMARY KEY。

5.6.3.10. 创建MaxCompute结果表

本文为您介绍如何创建MaxCompute结果表，以及创建过程中及到的WITH参数、类型映射和常见问题。

重要

- 仅Blink 1.5.1及以上版本支持MaxCompute结果表。
- MaxCompute中的Clustered Table表不支持作为MaxCompute结果表。

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。

- 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。

语法示例

实时计算Flink版支持使用MaxCompute作为结果输出，示例代码如下。

说明 DDL中定义的字段需要与MaxCompute物理表中的字段名称、顺序以及类型保持一致，否则可能导致MaxCompute物理表中查询的数据为 /n 。

```
create table odps_output(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****'
);
```

WITH参数

参数	说明	是否必填	备注
type	结果表类型。	是	固定值为 <code>odps</code> 。
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环境下必填。
project	MaxCompute项目名称。	是	无。
tableName	表名。	是	无。
accessId	AccessKey ID。	是	无。

参数	说明	是否必填	备注
accessKey	AccessKey Secret。	是	无。
partition	分区名。	否	<p>如果存在分区表则必填：</p> <ul style="list-style-type: none"> 固定分区 例如 <code>`partition` = 'ds=20180905'</code> 表示将数据写入分区 <code>ds= 20180905</code> 。 动态分区（Blink 3.2.1及以上版本支持） 如果不明文显示分区的值，则会根据写入数据中的分区列具体的值，写入到不同的分区中。例如 <code>`partition` = 'ds'</code> 表示根据 <code>ds</code> 字段的值写入分区。 如果要创建多级动态分区，With参数中Partition的字段顺序和结果表的DDL中的分区字段顺序，必须与物理表一致，各个分区字段之间使用逗号（,）分割。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 10px; margin-top: 10px;"> <p>说明</p> <ul style="list-style-type: none"> 动态分区列需要显式写在建表语句中。 对于动态分区字段为空的情况，如果数据源中 <code>ds=null</code> 或者 <code>ds=""</code>，则输出结果为： <ul style="list-style-type: none"> 3.2.1及以前的版本会抛出空指针异常（NPE）。 3.2.2及以后的版本会创建<code>ds=NULL</code>的分区。 </div>
flushIntervalMs	<p>Odps tunnel writer 缓冲区Flush间隔，单位毫秒。</p> <p>MaxCompute Sink 写入记录时，先放入数据到MaxCompute的缓冲区中，等缓冲区溢出或者每隔一段时间（flushIntervalMs）时，再把缓冲区里的数据写到目标MaxCompute表。</p>	否	<p>默认值是30000毫秒，即30秒。</p> <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 10px; margin-top: 10px;"> <p>说明 Blink 3.6.0及以上版本支持该参数。</p> </div>

参数	说明	是否必填	备注
partitionBy	写入Sink节点前，系统会根据该值做Hash Shuffle，数据就会流向对应的Sink节点。	否	系统按照多个列进行Hash Shuffle，各个列名之间使用逗号(,)分割。默认为空。 ? 说明 Blink 3.6.0及以上版本支持该参数。
isOverwrite	写入Sink节点前，是否将结果表清空。	否	<ul style="list-style-type: none"> Blink 3.2以下版本默认参数值为 true。 Blink 3.2及以上版本默认参数值为 false。在声明MaxCompute的流式作业结果表时，isOverwrite参数必须为false，否则在编译时会抛出异常。 ? 说明 Blink 3.2及以上版本支持 isOverwrite参数值变更为true。Blink 3.2以下版本如需变更isOverwrite参数值，请升级版本。
dynamicPartitionLimit	分区数目最大值。	否	默认值是100，内存中会把出现过的分区和Tunnel/writer的映射关系维护到一个Map里，如果这个Map的大小超过了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	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	VARCHAR

MaxCompute字段类型	实时计算Flink版字段类型
DECIMAL	DECIMAL

代码示例

包含MaxCompute结果表的实时计算Flink版作业代码示例如下：

- 写入固定分区

```
CREATE TABLE source (  
  id INT,  
  len INT,  
  content VARCHAR  
) with (  
  type = 'random'  
);  
create table odps_sink (  
  id INT,  
  len INT,  
  content VARCHAR  
) with (  
  type = 'odps',  
  endPoint = '<yourEndpoint>',  
  project = '<yourProjectName>',  
  tableName = '<yourTableName>',  
  accessId = '<yourAccessId>',  
  accessKey = '<yourAccessPassword>',  
  `partition` = 'ds=20180418'  
);  
INSERT INTO odps_sink  
SELECT  
  id, len, content  
FROM source;
```

- 写入动态分区

```
CREATE TABLE source (  
  id INT,  
  len INT,  
  content VARCHAR,  
  c TIMESTAMP  
) with (  
  type = 'random'  
);  
create table odps_sink (  
  id INT,  
  len INT,  
  content VARCHAR,  
  ds VARCHAR --动态分区列需要显式写在建表语句中。  
) with (  
  type = 'odps',  
  endPoint = '<yourEndpoint>',  
  project = '<yourProjectName>',  
  tableName = '<yourTableName>',  
  accessId = '<yourAccessId>',  
  accessKey = '<yourAccessPassword>',  
  `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 source;
```

常见问题

- Q: 公共云的endPoint和tunnelEndpoint是什么？配置错误会产生什么结果？

A: 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。

- endPoint配置错误：任务上线停滞在91%的进度。
- tunnelEndpoint配置错误：任务运行失败。

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

A: Blink 3.2以下版本支持，Blink 3.2及以上版本不支持。

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

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

- Q: 作业运行过程中报错ErrorMessage=Authorization Failed [4019], You have NO

```
privilege'ODPS:****'
```

A: 该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见[用户认证](#)。

5.6.3.11. 创建云数据库HBase版结果表

本文为您介绍如何创建实时计算云数据库HBase版结果表。

重要

- 本文档仅适用于实时计算独享模式。
- Blink 3.3.0以下版本仅支持HBase标准版。
- Blink 3.3.0及以上版本同时支持HBase标准版和HBase增强版。
- Blink 3.5.0及以上版本支持HBase写入主备切换。
- 实时计算HBase结果表不支持自建的开源HBase。

DDL定义

实时计算支持使用HBase作为结果输出。

- HBase标准版示例代码如下。

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

- HBase增强版示例代码如下。

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  endPoint = '<host:port>', ----HBase增强版的Java API访问地址。
  userName = 'root', --用户名。
  password = 'root', --密码。
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

- Blink 3.5.0及以上HBase增强版示例代码如下。

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', ----HBase增强版的Java API访问地址。
  userName = 'root', --用户名。
  password = 'root', --密码。
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

- Blink 3.5.0及以上HBase写入主备切换示例代码如下。

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', ----HBase高可用访问地址。
  haClusterID = 'ha-xxx', ----HBase高可用实例ID。
  userName = 'root', --用户名。
  password = 'root', --密码。
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```


② 说明

- PRIMARY KEY支持定义多字段。多字段以 `rowkeyDelimiter`（默认为 `:`）作为分隔符进行连接。
- HBase执行撤回删除操作时，如果COLUMN定义了多版本，将清空所有版本的COLUMN值。
- HBase标准版和HBase增强版DDL的区别为连接参数不同：
 - HBase标准版使用连接参数 `zkQuorum`。
 - HBase增强版使用连接参数 `endPoint`。

WITH参数

参数	说明	是否必填	备注
<code>type</code>	结果表类型	是	固定值为cloudhbase。
<code>zkQuorum</code>	HBase集群配置的zk地址	是	可以在 <code>hbase-site.xml</code> 文件中查看 <code>hbase.zookeeper.quorum</code> 相关配置。 ② 说明 仅在HBase标准版中生效。
<code>zkNodeParent</code>	集群配置在zk上的路径	否	可以在 <code>hbase-site.xml</code> 文件中查看 <code>hbase.zookeeper.quorum</code> 相关配置。 ② 说明 仅在HBase标准版中生效。
<code>endPoint</code>	HBase地域名称	是	可在购买的HBase实例控制台中获取。 ② 说明 仅在HBase增强版中生效。
<code>userName</code>	用户名	否	仅在HBase增强版中生效。
<code>password</code>	密码	否	仅在HBase增强版中生效。
<code>tableName</code>	HBase表名	是	无
<code>columnFamily</code>	列族名	是	仅支持插入同一列族。
<code>maxRetryTimes</code>	最大尝试次数	否	默认值为10。 ② 说明 仅实时计算3.2.3及以上版本支持 <code>maxRetryTimes</code> 参数。

参数	说明	是否必填	备注
bufferSize	流入多少条数据后进行去重	否	默认值为5000。
batchSize	一次批量写入的条数	否	默认值为100。 ❓ 说明 建议batchSize参数值为200~300。过大的batchSize值可能导致任务OOM（内存不足）报错。
flushIntervalMs	周期性清理buffer的间隔，可以减少写入HBase的延迟。	否	默认值为2000，单位为毫秒。
writePkValue	是否写入主键值	否	默认值为false。
stringWriteMod	是否都按照STRING插入	否	默认值为false。
rowkeyDelimiter	rowKey的分隔符	否	默认值为冒号（:）。
isDynamicTable	是否为动态表	否	默认值为false。
haClusterID	HBase高可用实例ID	否	只有访问同城主备实例时才需要配置。

动态表

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

rowkey	cf:0	cf:1	cf:2
20170707	100	cf:1	300

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

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

❓ 说明 使用动态表时，所有数据类型需要先转换为STRING类型，再进行输入。

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

说明

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

代码示例

包含HBase结果表的实时计算作业代码示例如下。

```
create table source (  
  id TINYINT,  
  name BIGINT  
) with (  
  type = 'random'  
);  
create table sink (  
  id TINYINT,  
  name BIGINT,  
  primary key (id)  
) with (  
  type = 'cloudhbase',  
  zkQuorum = '<yourZkQuorum>',  
  columnFamily = '<yourColumnFamily>',  
  tableName = '<yourTableName>'  
);  
INSERT INTO sink  
SELECT id, name FROM source;
```

常见问题

Q: 为什么HBase结果表作业运行时会出现 cloudHbase update error, No columns to insert for #10 item ?

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

5.6.3.12. 创建Elasticsearch结果表

本文为您介绍如何创建实时计算Flink版Elasticsearch（ES）结果表以及创建结果表时使用的WITH参数。

 **重要** 本文仅适用于Blink 3.2.2及以上版本。

DDL定义

实时计算Flink版支持使用ES作为结果输出，示例代码如下。

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

说明

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

WITH参数（通用配置）

参数	说明	是否必选	默认值
type	connector类型。	是	elasticsearch
endPoint	Server地址，例 入： http://127.0.0.1:9211 。	是	无
accessId	创建ES时的登录名。  说明 如果您通过Kibana插件操作ES，请填写Kibana登录ID。	是	无

参数	说明	是否必选	默认值
accessKey	创建ES时的登录密码。 ? 说明 如果您通过Kibana插件操作ES，请填写Kibana登录密码。 	是	无
index	索引名称。	是	无
typeName	文档类型。	是	_doc
bufferSize	流入多少条数据后开始去重。	否	1000
maxRetryTimes	异常重试次数。	否	30
timeout	读超时，单位为毫秒。	否	600000
discovery	是否开启节点发现。如果开启，客户端每5分钟刷新一次Server List。	否	false
compression	是否使用GZIP压缩Request Bodies。	否	true
multiThread	是否开启JestClient多线程。	否	true
ignoreWriteError	是否忽略写入异常。	否	false
settings	创建Index的Settings配置。	否	无
updateMode	指定主键（PRIMARY KEY）后的更新模式： <ul style="list-style-type: none"> • full：全量覆盖。 • inc：增量更新。 	否	full

WITH参数（动态索引相关）

参数	说明	是否必选	备注
dynamicIndex	是否开启动态索引。	否	参数取值如下： <ul style="list-style-type: none"> • true：开启。 • false（默认值）：不开启。
indexField	抽取索引的字段名。	dynamicIndex为true时必填。	只支持TIMESTAMP（以秒为单位）、DATE和LONG3种数据类型。
indexInterval	切换索引的周期。	dynamicIndex为true时必填。	参数值如下： <ul style="list-style-type: none"> • d（默认值）：天 • m：月 • w：周

参数	说明	是否必选	备注
mapping	启用动态索引时，设置文档各字段的类型与格式。例如，设置名为sendTime字段的格式： <pre>{ "properties": { "sendTime": { "type": "date", "format": "yyyy-MM-dd HH:mm:ss" } } }</pre>	否	默认值为空。 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> <p>? 说明</p> <ul style="list-style-type: none"> Blink 3.7.0版本及以上版本支持该参数。 Elasticsearch 7.0不支持mapping参数。 </div>

? 说明

- 仅Blink 2.2.7及以上版本支持动态索引功能。
- 当开启动态索引后，基本配置中的 index 名称会作为后续创建索引的统一Alias，Alias和索引为一对多关系。
- 不同的 indexInterval 对应的真实索引名称：
 - d -> Alias "yyyyMMdd"
 - m -> Alias "yyyyMM"
 - w -> Alias "yyyyMMW"
- 对于单个的真实索引可使用Index API进行修改，但对于Alias只支持 get 功能。如果需要更新Alias，请参见Index Aliases。

代码示例

创建动态索引的代码示例如下所示。

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

5.6.3.13. 创建时序数据库结果表

本文为您介绍如何创建实时计算Flink版时序数据库（TSDB）结果表以及创建结果表时使用的WITH参数。

重要

- 本文仅适用于Blink 2.0及以上版本。
- 实时计算Flink版引用时序数据库（TSDB）结果表需要配置数据存储白名单，详情请参见 [数据存储白名单配置](#)。

什么是时间序列数据库（TSDB）

阿里云时序数据库（Time Series Database，简称TSDB）是一种集时序数据高效读写、压缩存储、实时计算能力为一体的数据库服务，可以广泛应用于物联网和互联网领域，实现对设备及业务服务的实时监控，实时预测告警。

DDL定义

实时计算Flink版支持使用TSDB作为结果输出，示例代码如下。

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

建表默认格式：

- 第0列：metric（VARCHAR）。
- 第1列：timestamp（INTEGER），单位为秒。
- 第2列：value（DOUBLE）。
- 第3~N列：TagKey，即时间序列数据库中的fieldName。

说明

- tag可以为多列。
- 必须声明metric、timestamp和value，且字段名称、字段顺序和字段数据类型必须和TSDB保持完全一致。
- 参数设置详情请参见 [TSDB建表规范](#)。

WITH参数

参数	说明	备注
type	结果表类型	固定值为 <code>hitsdb</code> 。
host	IP或VIP域名	填写注册实例的Host，详情请参见 连接实例 。
port	端口	默认值为8242。
virtualDomainSwitch	是否使用VIPServer	默认值为false，如果需要使用VIPServer，则virtualDomainSwitch设置为true。
httpConnectionPool	HTTP连接池	默认值为10。
httpCompress	使用GZIP压缩	默认值为false，即不压缩。
httpConnectTimeout	HTTP连接超时时间	默认值为0。
ioThreadCount	IO线程数	默认值为1。
batchPutBufferSize	缓冲区大小	默认值为10000。
batchPutRetryCount	写入重试次数	默认值为3。
batchPutSize	每次提交数据量	默认每次提交500个数据点。
batchPutTimeLimit	缓冲区等待时间	默认值为200，单位为毫秒。
batchPutConsumerThreadCount	序列化线程数	默认值为1。

Blink写入TSDB模型

Blink 3.2 及以上版本支持6种Blink写入TSDB的模型，分别是：

- 支持单值无tag数据点的写入，其schema格式如下所示，包括3个字段，这3个字段名称不可使用其他名称代替。

```
metric,timestamp,value
```

- 支持单值带tag数据点的写入，其schema格式如下所示，除metric、timestamp和value关键词名字必须保持一致外，tag的名称可以任意指定。

```
metric,timestamp,value,tagKey1,.....,tagKeyN
```

- 支持单值带不确定tag个数的数据点的写入，其schema格式如下所示，包括4个字段，这4个字段名称不可使用其他名称代替。

```
metric,timestamp,value,tags
```

其中，tags的内容为形如下格式的JSON字符串，便于绕过Blink table schema需要固定tag个数的限制。

```
{"tagKey1":"tagValue1","tagKey2":"tagValue2",.....,"tagKeyN":"tagValueN"}
```


- 支持多值无tag数据点的写入，其schema格式如下所示。

```
metric,timestamp,field_name1,field_name2,.....,field_nameN
```

其中metric和timestamp字段名称不可使用其他名称代替。对于多值的fields，由于需要区分tag和field，同时兼容之前的单值写入，这里约定需要给每个field加上固定前缀field_，自动识别field字段。例如，对于一个field名称 field_name1，在写入TSDB时，会自动将前缀 field_ 去掉，只保留name1，即对于上面的schema，实际写入TSDB的格式如下，name1和name2是多值field的名称。

```
metric,timestamp,name1,name2,.....,nameN
```

- 支持多值带tag数据点的写入，其schema格式如下所示，除metric和timestamp关键词名字必须保持一致外，tag的名称可以任意指定。

```
metric,timestamp, tagKey1,.....,tagKeyN, field_name1,field_name2,.....,field_nameN
```

- 支持单值带不确定tag个数的数据点的写入，其schema格式如下所示。

```
metric,timestamp,tags,field_name1,field_name2,.....,field_nameN
```

其中，tags的内容类似如下JSON字符串，便于绕过Blink table schema需要固定tag个数的限制。

```
{"tagKey1":"tagValue1","tagKey2":"tagValue2",.....,"tagKeyN":"tagValueN"}
```

常见问题

Q: 为什么Failover中会报错：LONG类型不能转换成INT类型？

A: Blink 2.2.5以下版本仅支持INT类型。Blink 2.2.5及以上版本支持BIGINT类型。

5.6.3.14. 创建消息队列Kafka结果表

本文档为您介绍如何创建实时计算Flink版消息队列Kafka结果表，以及Kafka版本对应关系。

重要

- 本文仅适用于实时计算2.0及以上版本。
- 本文仅适用于独享模式。
- Kafka结果表支持写入自建Kafka集群，但需注意版本对应关系，以及自建集群和实时计算集群的网络环境配置。

什么是Kafka结果表

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。实时计算采用Kafka作为流式数据的数据源表或结果表。

DDL定义

消息队列Kafka结果表需要定义的DDL如下。

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

说明

- 创建Kafka结果表时，必须明文指定 PRIMARY KEY (messageKey)。
- 仅Blink 2.2.6及以上版本，支持阿里云Kafka或自建Kafka的TPS和RPS等指标信息的显示。

WITH参数

• 通用配置

参数	说明	是否必选	备注
type	Kafka对应版本	是	必须是Kafka08、Kafka09、Kafka010、Kafka011中的一种，版本对应关系请参见 Kafka版本对应关系 。
topic	Topic名称	是	无

• 必选配置

○ Kafka08必选配置

参数	说明
zookeeper.connect	zk连接ID

○ Kafka09/Kafka010/Kafka011必选配置

参数	说明
bootstrap.servers	Kafka集群地址

• 可选配置参数

- consumer.id
- socket.timeout.ms
- fetch.message.max.bytes
- num.consumer.fetchers
- auto.commit.enable
- auto.commit.interval.ms
- queued.max.message.chunks

- `rebalance.max.retries`
- `fetch.min.bytes`
- `fetch.wait.max.ms`
- `rebalance.backoff.ms`
- `refresh.leader.backoff.ms`
- `auto.offset.reset`
- `consumer.timeout.ms`
- `exclude.internal.topics`
- `partition.assignment.strategy`
- `client.id`
- `zookeeper.session.timeout.ms`
- `zookeeper.connection.timeout.ms`
- `zookeeper.sync.time.ms`
- `offsets.storage`
- `offsets.channel.backoff.ms`
- `offsets.channel.socket.timeout.ms`
- `offsets.commit.max.retries`
- `dual.commit.enabled`
- `partition.assignment.strategy`
- `socket.receive.buffer.bytes`
- `fetch.min.bytes`

🔍 说明 其它可选配置项请参见以下Kafka官方文档：

- [Kafka09](#)
- [Kafka010](#)
- [Kafka011](#)

Kafka版本对应关系

type	Kafka版本
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2及以上

示例

```
create table datahub_input (  
  id VARCHAR,  
  nm VARCHAR  
) with (  
  type = 'datahub'  
);  
create table sink_kafka (  
  messageKey VARBINARY,  
  `message` VARBINARY,  
  PRIMARY KEY (messageKey)  
) with (  
  type = 'kafka010',  
  topic = '<yourTopicName>',  
  bootstrap.servers = '<yourServerAddress>'  
);  
INSERT INTO  
  sink_kafka  
SELECT  
  cast(id as VARBINARY) as messageKey,  
  cast(nm as VARBINARY) as `message`  
FROM  
  datahub_input;
```

5.6.3.15. 创建云数据库HybridDB for MySQL结果表

本文为您介绍如何创建实时计算Flink版云数据库HybridDB for MySQL结果表，以及创建结果表时使用的WITH参数。

重要

- 阿里云数据库HybridDB for MySQL产品已下线。
- 本文仅适用于Blink 1.4.5及以上版本。

什么是云数据库HybridDB for MySQL

云数据库HybridDB for MySQL（原名PetaData）是同时支持海量数据在线事务（OLTP）和在线分析（OLAP）的HTAP（Hybrid Transaction/Analytical Processing）关系型数据库。HybridDB for MySQL采用一份数据存储来进行OLTP和OLAP处理，避免把一份数据复制多次进行数据分析，极大地降低了数据存储的成本。

DDL定义

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

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

说明

- 实时计算Flink版写入PetaData数据库结果表原理：针对实时计算Flink版每行结果数据，拼接成一行SQL语句，输入至目标端数据库。
- bufferSize默认值是 1000，如果到达bufferSize阈值，则会触发写出。因此您配置batchSize的同时还需要配置bufferSize。bufferSize和batchSize大小相同即可。
- batchSize数值不建议设置过大，建议设置 `batchSize='4096'`。

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为petaData。
url	地址	是	切换网络类型。
tableName	表名	是	无
userName	用户名	是	无
password	密码	是	无
maxRetryTimes	最大重试次数	否	默认值为3。
batchSize	一次批量写入的条数	否	默认值为1000，表示每次写多少条。
bufferSize	流入多少条数据后开始去重	否	无
flushIntervalMs	清空缓存的时间间隔	否	单位为毫秒。默认值为3000，表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
ignoreDelete	是否忽略Delete操作	否	默认值为false。

5.6.3.16. 创建云数据库RDS SQL Server版结果表

本文为您介绍如何创建云数据库RDS SQL Server版结果表，以及创建结果表时使用的WITH参数、类型映射和JDBC连接参数。

重要

- 本文仅适用于Blink 3.2.0及以上版本。
- 实时计算Flink版暂不支持引用RDS SQL Server版本作为数据存储。

语法示例

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

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

说明

- 实时计算Flink版写入RDS和DRDS数据库结果表原理：针对实时计算Flink版每行结果数据，拼接成一行SQL语句，输入至目标端数据库。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 SQL Server数据库支持自增主键。如果需要让实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段。例如，ID是自增字段，实时计算Flink版DDL不写出该自增字段，则数据库在一行数据写入过程中会自动填补相关的自增字段。
- 如果DRDS有分区表，拆分键必须在实时计算Flink版DDL里 primary key () 中声明，否则拆分的表无法写入。
- DDL声明的字段必选至少存在一个非主键的字段，否则产生报错。

WITH参数

参数	参数说明	是否必填	备注
url	JDBC (Java DataBase Connectivity) 连接地址	是	地址请参见： <ul style="list-style-type: none"> • RDS的URL地址 • DRDS的URL地址

参数	参数说明	是否必填	备注
tableName	表名	是	无
username	账号	是	无
password	密码	是	无
maxRetryTimes	写入重试次数	否	默认值为10。
bufferSize	流入多少条数据后开始去重	否	默认值为10000，表示输入的数据达到10000条开始去重。
flushIntervalMs	清空缓存的时间间隔	否	单位为毫秒，默认值为2000，表示如果缓存中的数据在等待2秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
excludeUpdateColumns	忽略指定字段的更新	否	可选，默认值为空（默认忽略Primary Key字段），表示更新主键值相同的数据时，忽略指定字段的更新。
ignoreDelete	是否忽略Delete操作	否	默认值为False。

类型映射

RDS字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

JDBC连接参数

参数名称	参数说明	缺省值	最低版本要求
useUnicode	是否使用Unicode字符集。如果参数CharacterEncoding设置为GB2312或GBK，useUnicode值必须设置为True。	False	1.1g
characterEncoding	当useUnicode设置为True时，指定字符编码。例如可设置为GB2312或GBK。	False	1.1g
autoReconnect	当数据库连接异常中断时，是否自动重新连接。	False	1.1
autoReconnectForPools	是否使用针对数据库连接池的重连策略。	False	3.1.3
failOverReadOnly	自动重连成功后，连接是否设置为只读。	True	3.0.12
maxReconnects	autoReconnect设置为True时，重试连接的次数。	3	1.1
initialTimeout	autoReconnect设置为True时，两次重连之间的时间间隔，单位为秒。	2	1.1
connectTimeout	和数据库服务器建立socket连接时的超时，单位为毫秒。	0，表示永不超时，适用于JDK 1.4及以上版本。	3.0.1
socketTimeout	socket操作（读写）超时，单位为毫秒。	0，表示永不超时。	3.0.1

代码示例

包含云数据库RDS SQL Server版结果表的实时计算Flink版作业代码示例如下。


```
CREATE TABLE source (  
  id INT,  
  len INT,  
  content VARCHAR  
) with (  
  type = 'random'  
);  
CREATE TABLE rds_output(  
  id INT,  
  len INT,  
  content VARCHAR,  
  PRIMARY KEY (id,len)  
) WITH (  
  type='jdbc',  
  url='<yourDatabaseURL>',  
  tableName='<yourDatabaseTable>',  
  userName='<yourDatabaseUserName>',  
  password='<yourDatabasePassword>'  
);  
INSERT INTO rds_output  
SELECT id, len, content FROM source;
```

常见问题

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

A: 根据DDL中是否定义主键，分为以下两种方式：

- DDL中定义主键：采用 `insert into on duplicate key update` 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。
- DDL中没有定义主键：采用 `insert into` 的方式插入记录，追加数据。

- Q: 使用RDS表中的唯一索引进行GROUP BY操作需要注意什么？

A: 需要注意以下两点：

- 在作业中的主键（Primary Key）中声明该唯一索引。
- RDS中只有一个自增主键，实时计算Flink版作业中不能将该自增主键声明为主键。

5.6.3.17. 创建云数据库Redis版结果表

本文为您介绍如何创建实时计算Flink版云数据库Redis版结果表以及创建过程中涉及的WITH参数、类型映射和属性字段。

重要

- 本文仅适用于Blink 3.2.0及以上版本。
- 实时计算Flink版Redis结果表支持自建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。实时计算Flink版支持将云数据库Redis版作为流式数据的输出。

语法示例

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 resik_output (  
  a varchar,  
  b varchar,  
  primary key(a)  
) with (  
  type = 'redis',  
  mode = 'string',  
  host = '${redisHost}', -- 例如, '127.0.0.1'.  
  port = '${redisPort}', -- 例如, '6379'.  
  dbName = '${dbName}', -- 默认为0。  
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认为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 resik_output (  
  a varchar,  
  b varchar,  
  primary key(a)  
) with (  
  type = 'redis',  
  mode = 'list',  
  host = '${redisHost}', -- 例如, '127.0.0.1'.  
  port = '${redisPort}', -- 例如, '6379'.  
  dbName = '${dbName}', -- 默认为0。  
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认为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 resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'set',
  host = '${redisHost}', -- 例如, '127.0.0.1'。
  port = '${redisPort}', -- 例如, '6379'。
  dbName = '${dbName}', -- 默认值为0。
  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 resik_output (
  a varchar,
  b varchar,
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'hashmap',
  host = '${redisHost}', -- 例如, '127.0.0.1'。
  port = '${redisPort}', -- 例如, '6379'。
  dbName = '${dbName}', -- 默认值为0。
  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 `add key score value`。

```
create table resik_output (
  a varchar,
  b double, --必须为DOUBLE类型。
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'sortedset',
  host = '${redisHost}', -- 例如, '127.0.0.1'。
  port = '${redisPort}', -- 例如, '6379'。
  dbName = '${dbName}', -- 默认值为0。
  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	参数说明	是否必填	取值
type	结果表类型	是	固定值为 redis 。
mode	对应Redis的数据结构		取值如下： <ul style="list-style-type: none"> • string • list • set • hashmap • sortedset
host	Redis Server对应地址		取值示例： 127.0.0.1 。目前不支持配置多个host:port
port	Redis Server对应端口	否	默认值为6379。目前不支持配置多个host:port
dbNum	Redis Server对应数据库序号		默认值为0。
ignoreDelete	是否忽略Retraction消息		默认值为false，可取值为true或false。如果设置为false，收到Retraction时，同时删除数据对应的key及已插入的数据。
password	Redis Server对应的密码		默认值为空，不进行权限验证。
clusterMode	Redis是否为集群模式		取值如下： <ul style="list-style-type: none"> • true: Redis为集群模式。 • flalse（默认值）: Redis为单机模式。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ? 说明 仅Blink 3.6.x及以上版本支持该参数。 </div>

类型映射

Redis和实时计算Flink版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。

Redis字段类型	实时计算Flink版字段类型
STRING	VARCHAR
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

包含Redis结果表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE random_stream (  
  v VARCHAR,  
  p VARCHAR) with (  
  type = 'random'  
);  
create table resik_output (  
  a VARCHAR,  
  b VARCHAR,  
  primary key(a)  
) with (  
  type = 'redis',  
  mode = 'string',  
  host = '<yourRedisHost>',  
  password = '<yourRedisPassword>'  
);  
INSERT INTO resik_output  
SELECT v, p  
FROM random_stream;
```

5.6.3.18. 创建云数据库MongoDB版结果表

本文为您介绍如何创建实时计算Flink版云数据库MongoDB版结果表，以及创建过程中涉及到的WITH参数。

重要

- 本文仅适用于Blink 3.2.2及以上版本。
- MongoDB结果表不支持主键更新，数据输入形式为重复插入。

DDL定义

实时计算Flink版支持使用MongoDB作为数据输出的结果表，示例代码如下。

```
CREATE TABLE mongodb_sink (  
  `a` VARCHAR  
) WITH (  
  type = 'mongodb',  
  database = '<yourDatabaseName>',  
  collection = '<yourCollectionName>',  
  uri = 'mongodb://<{databaseAccount}>:<{atabasePassword}>@<{host}>:****?replicaSet=mgset-1224****',  
  keepAlive = 'true',  
  maxConnectionIdleTime = '20000',  
  batchSize = '2000'  
);
```

WITH参数

参数	说明	是否必填	备注
type	Connector类型	是	固定值为mongodb。
database	数据库名称	是	无
collection	数据集合	是	无
uri	MongoDB连接串	是	无
keepAlive	是否保持长连接	否	默认值为true。
maxConnectionIdleTime	连接超时时长	否	整型值，不能为负数，单位为毫秒。默认值为60000。0表示无连接超时限制。
batchSize	每次批量写入的条数	否	整型值，默认值为1024。系统会设定一个大小为batchSize的缓冲条数，当数据的条数达到batchSize时，触发数据的输出。 ? 说明 当Checkpoint时间达到时，即使数据未到达batchSize值，也将触发数据的输出。

5.6.3.19. 创建云原生数据仓库AnalyticDB MySQL版3.0结果表

本文为您介绍如何创建云原生数据仓库AnalyticDB MySQL版3.0结果表，以及创建过程中涉及的WITH参数。

重要

- 云原生数据仓库AnalyticDB MySQL版3.0结果表暂不支持注册存储功能。
- 本文仅适用于 Blink 3.3.0 及以上版本。
- 云原生数据仓库AnalyticDB MySQL版2.0结果表创建说明，请参见 [创建云原生数据仓库AnalyticDB MySQL版2.0结果表](#)。
- 云原生数据仓库AnalyticDB MySQL版3.0数据库支持自增主键。如果实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段。例如，ID是自增字段，实时计算Flink版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定义

实时计算Flink版支持使用云原生数据仓库AnalyticDB MySQL版3.0版本作为结果表输出。示例代码如下。

```
CREATE TABLE adb_output (
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id,len)
) WITH (
  type='ADB30',
  url='jdbc:mysql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
```

实现原理

实时计算Flink版写入云原生数据仓库AnalyticDB MySQL版3.0结果表可以分为以下两个阶段：

1. 将实时计算Flink版每行的结果数据拼接为一行SQL。
2. 在目标数据库执行拼接后的SQL。

WITH参数

参数	注释说明	是否必选	备注
type	connector类型	是	固定值为 ADB30 。
url	jdbc连接地址	是	云原生数据仓库AnalyticDB MySQL版数据库地址。示例： <code>url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'</code> 。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>? 说明</p> <ul style="list-style-type: none"> • 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。 </div>
tableName	表名	是	无
username	账号	是	无
password	密码	是	无
maxRetryTimes	写入重试次数	否	默认值为3。

参数	注释说明	是否必选	备注
bufferSize	流入多少条数据后开始去重	否	默认值为1000，表示输入的数据达到1000条则开始输出。 ? 说明 需要指定主键后才能生效。
batchSize	一次批量写入的条数	否	默认值为1000。 ? 说明 需要指定主键后才能生效。
flushIntervalMs	清空缓存的时间间隔	否	单位为毫秒，默认值为3000。表示如果缓存中的数据在等待3秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
ignoreDelete	是否忽略DELETE操作	否	默认值为false，表示支持DELETE功能。
replaceMode	是否采用replace into语法插入数据	否	取值如下： <ul style="list-style-type: none"> • true（默认）：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 replaceMode参数生效，需要满足以下两个条件： <ul style="list-style-type: none"> • Blink为3.x及以上版本。 • ADB为3.1.3.5及以上版本。
excludeUpdateColumns	在更新主键值相同的数据时，忽略指定字段的更新。	否	如果要忽略的字段为多个，则需要使用英文逗号（,）分割。例如 <code>excludeUpdateColumns=column1,column2</code> 。 ! 重要 要忽略的多个字段需要写在一行中，不能换行，否则不生效。
reserveMillisEcond	TimeStamp类型是否保留毫秒	否	默认值为false，不保留毫秒数值。

5.6.3.20. 创建自定义结果表

本文为您介绍如何创建实时计算Flink版自定义结果表，自定义结果表可以满足您各种差异化的输出需求。



重要

- 本文仅适用于Blink 1.4.5及以上版本。
- 本文仅适用于独享模式。

搭建环境

您可以通过以下两种方式搭建自定义结果表的开发环境：


- 直接使用示例中的环境。

为了便于您快速开发业务，实时计算Flink版提供如下自定义结果表示例：

- [实时计算Flink版3.0版本示例](#)。
- [实时计算Flink版2.0版本示例](#)。
- [实时计算Flink版1.0版本示例](#)。

 说明 示例中已为您配置对应版本的开发环境，您无需搭建环境。

- 下载JAR包自行搭建环境

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

- 实时计算Flink版3.0版本
 - JAR包下载
 - [blink-connector-custom-blink-3.2.1](#)
 - [blink-connector-common-blink-3.2.1](#)

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

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

- 依赖包

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

- 实时计算Flink版2.0版本

- JAR包下载

- [blink-connector-common-blink-2.2.4](#)
- [blink-connector-custom-blink-2.2.4](#)
- [blink-table-blink-2.2.4](#)
- [flink-table_2.11-blink-2.2.4](#)
- [flink-core-blink-2.2.4](#)

■ 依赖包

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

- 实时计算Flink版1.0版本
 - JAR包下载
 - [blink-connector-common-blink-1.4](#)
 - [blink-connector-custom-blink-1.4](#)
 - [blink-table-blink-1.4](#)
 - [flink-core-blink-1.4](#)
 - [flink-streaming-scala_2.11-blink-1.4](#)
 - 依赖包

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


接口说明

自定义结果表Class需要继承自定义Sink插件的基类CustomSinkBase，并使用如下方法实现。

```
protected Map<String,String> userParamsMap;// userParamsMap是自定义SQL的WITH语句中定义的键值对，所有的键均为小写。
protected Set<String> primaryKeys;// primaryKeys是自定义的主键字段名。
protected List<String> headerFields;// headerFields是标记为header的字段列表。
protected RowTypeInfo rowTypeInfo;// 字段类型和名称。
/**
 * 初始化方法。每次初始建立和Failover的时候会调用一次。
 *
 * @param taskNumber taskNumber为当前节点的编号。
 * @param numTasks numTasks为Sink节点的总数。
 * @throws IOException
 */
public abstract void open(int taskNumber,int numTasks) throws IOException;
/**
 * close方法，释放资源。
 *
 * @throws IOException
 */
public abstract void close() throws IOException;
/**
 * 处理插入单行数据。
 *
 * @param row
 * @throws IOException
 */
public abstract void writeAddRecord(Row row) throws IOException;
/**
 * 处理删除单行数据。
 *
 * @param row
 * @throws IOException
 */
public abstract void writeDeleteRecord(Row row) throws IOException;
/**
 * 如果进行批量插入，该方法需要把线程中缓存的数据全部刷入下游存储；如果不进行批量插入，可以不使用该方法。
 *
 * @throws IOException
 */
public abstract void sync() throws IOException;
/**
 * 返回类名。
 */
public String getName();
```

自定义Redis结果表示例

下载[实时计算Flink版3.0版本示例](#)，进入blink_customersink_3x目录，执行 `mvn clean package` 命令，再在实时计算Flink版开发控制台上传刚编译成功后的JAR包blink_customersink_3x/target/blink-customersink-3.x-1.0-SNAPSHOT-jar-with-dependencies.jar，引用资源之后，对于自定义的Sink插件，需要指明 `type = 'custom'`，并且指明实现接口的Class。

 **重要** 本示例仅作为自定义结果表开发参考，不适合直接作为生产使用。

```
create table in_table(
  kv varchar
)with(
  type = 'random'
);
create table out_table(
  `key` varchar,
  `value` varchar
)with(
  type = 'custom',
  class = 'com.alibaba.blink.customersink.RedisSink',
  -- 1. 可以定义更多自定义参数, 在open函数中通过userParamsMap获取。
  -- 2. with参数里key大小写不敏感。在实时计算Flink版中, 参数key的值直接处理为全小写。建议您在引用数据存储的DDL中使用小写声明key。
  host = 'r-uf****.redis.rds.aliyuncs.com',
  port = '6379',
  db = '0',
  batchsize = '10',
  password = '<yourHostPassword>'
);
insert into out_table
select
  substring(kv,0,4) as `key`,
  substring(kv,0,6) as `value`
from in_table;
```

Redis Sink插件的参数说明如下。

参数	说明	是否必填	备注
host	Redis实例内网连接地址 (host)	是	无
port	Redis实例端口号	是	无
password	Redis连接密码	是	无
db	Redis Database编号	否	默认值为0, 表示db0。
batchsize	每次批量写入的条数	否	默认值为1, 表示不批量写入。

5.6.3.21. 创建InfluxDB结果表

本文为您介绍如何创建实时计算Flink版InfluxDB结果表，以及创建结果表时使用的WITH参数和类型映射。

 重要

- InfluxDB暂不支持注册存储功能。
- 本文仅适用于Blink 3.5.0-hotfix及以上版本。

DDL定义

实时计算Flink版支持使用InfluxDB作为结果输出，示例代码如下。

```
create table stream_test_influxdb(  
  `metric` varchar,  
  `timestamp` BIGINT,  
  `tag_value1` varchar,  
  `field_fieldValue1` Double  
)with(  
  type = 'influxdb',  
  endpoint = 'http://service.cn.influxdb.aliyuncs.com:****',  
  database = '<yourDatabaseName>',  
  batchPutsSize = '1',  
  username = '<yourDatabaseUserName>',  
  password = '<yourDatabasePassword>'  
);
```

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
type	结果表类型	是	固定值为InfluxDB。
endpoint	InfluxDB的Endpoint	是	在InfluxDB中，Endpoint是VPC网络地址，例如： <code>https://localhost:3242</code> 或 <code>http://localhost:8086</code> 。 Endpoint支持HTTP和HTTPS。
database	InfluxDB的数据库名	是	例如：db-blink或者blink。
batchPutSize	批量提交的记录条数	否	默认每次提交500个数据点。
username	InfluxDB的用户名	是	需要对写入的数据库有写权限。
password	InfluxDB的密码	是	默认值为0。
retentionPolicy	保留策略	否	为空时，默认填写为每个database的默认保留策略。

类型映射

InfluxDB字段类型	实时计算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

5.6.3.22. 创建Phoenix5结果表

本文为您介绍如何创建实时计算Flink版云数据库Phoenix5结果表。

重要

- 本文仅适用于实时计算Flink版独享模式。
- 本文仅适用于Blink 3.4.0以上版本。
- Phoenix5仅支持5.x版本。
- Phoenix5是云数据库HBase实例中的一种HBase SQL服务，须在云数据库HBase实例中开启HBase SQL服务后才可以使用Phoenix5。

DDL定义

实时计算Flink版支持使用Phoenix5作为结果输出，示例代码如下。

```
create table US_POPULATION_SINK (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY) --主键必填。
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
```

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为 PHOENIX5 。
serverUrl	Phoenix5的Query Server地址： <ul style="list-style-type: none"> • 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 。 serverUrl格式为http://host:port，其中： <ul style="list-style-type: none"> • host: Phoenix5服务的域名。 • port: Phoenix5服务的端口号，固定值为8765。

参数	说明	是否必填	备注
tableName	读取Phoenix5表名。	是	Phoenix5表名格式为 SchemaName.TableName，其中： <ul style="list-style-type: none"> • SchemaName：模式名，可以为空，即不写模式名，仅写表名，表示使用数据库的默认模式。 • TableName：表名。

代码示例

包含Phoenix5结果表的实时计算Flink版作业代码示例如下。

```
create table `source` (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar
) WITH (
  type = 'random'
);
create table sink (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar,
  primary key (id)
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
INSERT INTO sink
SELECT `id`, `name`, `age`, `birthday`
FROM `source`;
```

5.6.3.23. 创建分析型数据库PostgreSQL版结果表

本文为您介绍如何创建分析型数据库PostgreSQL版结果表，以及创建过程中涉及到的WITH参数和类型映射。

 **重要** 本文仅适用于Blink 3.6.0及以上版本。

实现原理

实时计算Flink版写入分析型数据库PostgreSQL结果表可以分为以下两个阶段：

1. 将实时计算Flink版每行的结果数据拼接为一行SQL。

2. 在目标数据库执行拼接后的SQL。

DDL定义

实时计算Flink版支持使用分析型数据库PostgreSQL版作为结果输出。示例代码如下。

```
create table rds_output(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id)
) with (
  type='adbpg',
  url='jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
```

WITH参数

参数	说明	是否必填	备注
type	源表类型。	是	固定值为 adbpg 。
url	JDBC连接地址。	是	分析型数据库PostgreSQL版数据库的JDBC连接地址。格式为'jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>', 其中： <ul style="list-style-type: none"> • <i>yourNetworkAddress</i>: 内网地址。 • <i>PortId</i>: 连接端口。 • <i>yourDatabaseName</i>: 连接的数据库名称。 示例：url='jdbc:postgresql://gp-xxxxxx.gpdb.cn-chengdu.rds.aliyuncs.com:3432/postgres'
tableName	表名。	是	无。
username	账号。	是	无。
password	密码。	是	无。
maxRetryTimes	写入重试次数。	否	默认值为3。
useCopy	是否采用Copy API 写入数据。	否	参数取值如下： <ul style="list-style-type: none"> • 0（默认值）：采用INSERT方式写入数据。 • 1：采用copy API方式写入数据。
batchSize	一次批量写入的条数。	否	默认值为5000。

参数	说明	是否必填	备注
exception Mode	数据写入过程中出现异常时的处理策略。	否	支持以下两种处理策略： <ul style="list-style-type: none"> ignore（默认值）：忽略出现异常时写入的数据。 strict：数据写入异常时，Failover报错。
conflictMode	当主键冲突或唯一索引出现冲突时的处理策略。	否	支持以下三种处理策略： <ul style="list-style-type: none"> ignore（默认值）：忽略主键冲突，保留之前的数据。 strict：主键冲突时，Failover报错。 update：主键冲突时，更新新到的数据。
targetSchema	Schema名称。	否	默认值为public。
connectionMaxActive	单个task允许的最大连接数。	否	请根据实际的并发task个数，以及目标端数据库允许的最大连接数进行设置。

类型映射

分析型数据库PostgreSQL和实时计算Flink版字段类型对应关系如下。

分析型数据库PostgreSQL版字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
SMALLINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
DOUBLE PRECISION	DOUBLE
TEXT	VARCHAR
TIMESTAMP	DATETIME
DATE	DATE
REAL	FLOAT
DOUBLE PRECISION	DECIMAL
TIME	TIME
TIMESTAMP	TIMESTAMP

5.6.4. 创建数据维表

5.6.4.1. 概述

在维表DDL语法中增加1行PERIOD FOR SYSTEM_TIME的声明，定义维表的变化周期，即可使用标准的CREATE TABLE语法定义实时计算维表。

示例

```
CREATE TABLE white_list (  
  id varchar,  
  name varchar,  
  age int,  
  PRIMARY KEY (id),  
  PERIOD FOR SYSTEM_TIME --定义维表的变化周期。实时计算3.x及以上版本，维表DDL中可以不声明该句，在维表JOIN时，声明FOR SYSTEM_TIME AS OF PROCTIME () 即可。  
) with (  
  type = 'RDS',  
  ...  
);
```

说明

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

INDEX语法

说明 建议在实时计算2.2.7及以上版本使用 INDEX 语法。

实时计算2.2以下版本，维表定义要求声明 PRIMARY KEY ，这种情况下只能实现一对一连接。为支持一对多连接的需求，引入了 INDEX 语法。非 Cache All 的维表JOIN通过 INDEX LOOKUP 的方式实现一对多连接的需求。

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

UNIQUE INDEX 表示一对一连接，而 INDEX 表示一对多连接。

说明

- 实时计算2.2.7及以后版本支持 UNIQUE CONSTRAINT （ UNIQUE KEY ），实时计算2.2.7以下版本可以使用 PRIMARY KEY 的定义。
- 在生成执行计划时，引擎优先采用 UNIQUE INDEX 。即如果DDL中使用INDEX，但JOIN等连接条件中同时包含 UNIQUE 和 NON-UNIQUE INDEX 时，优先使用 UNIQUE INDEX 查找右表数据。
- 支持一对多连接的维表类型，例如RDS和MaxCompute。
- 您可以增加 maxJoinRows 参数，表示在一对多连接时，左表一条记录连接右表的最大记录数（默认值为1024）。在一对多连接的记录数过多时，可能会极大的影响流任务的性能，因此您需要增大Cache的内存（ cacheSize 限制的是左表key的个数）。
- 表格存储Tablestore和Hologres维表不支持使用INDEX进行一对多JOIN。

维表、源表和结果表的区别

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

5.6.4.2. 创建交互式分析Hologres维表

本文为您介绍如何创建交互式分析Hologres维表，以及创建维表时使用的WITH参数、CACHE参数和类型映射。

重要

- 本文仅适用于Blink 3.6.0及以上版本，如果您的Blink为3.6.0以下的版本，您可以升级Blink版本至3.6.0及以上版本，详情请参见[管理独享集群Blink版本](#)。您可以下载并安装使用 [blink-connector-hologres-blink-3.6.8.jar](#)或[blink-connector-hologres-blink-3.7.jar](#)包。
- 建议您使用Hologres 0.7及以上版本。

什么是交互式分析Hologres

交互式分析Hologres是实时交互分析产品，兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延实时时分析与处理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。

使用限制

- 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Blink Join On的字段，Blink 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。
- Hologres Blink Connector的维表功能不支持一对多的输出。
- 不支持读取Hologres分区表的数据。

语法示例

实时计算Flink版支持使用Hologres作为维表，代码示例如下。

```
CREATE TABLE hologres_dim_table(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME --定义维表的变化周期。
) WITH (
  type='hologres',
  endpoint='...',
  dbname='...',
  tablename='...',
  username='...',
  password='...'
);
```

WITH参数

参数	描述	是否必选	备注
type	数据库类型。	是	固定值为 <i>hologres</i> 。
endpoint	Hologres端点。	是	无。
tablename	表名称。 <div style="border: 1px solid #add8e6; padding: 5px; width: fit-content;"> <p>? 说明 如果 Schema不为Public 时, 则tableName需要填写为schema.tableName。</p> </div>	是	无。
dbname	数据库名称。	是	无。
username	用户名称。	是	无。
password	密码。	是	无。

CACHE参数

参数	描述	是否必选	备注
cache	缓存策略。	否	目前交互式分析Hologres维表支持以下两种缓存策略： <ul style="list-style-type: none"> • <i>None</i>（默认值）：无缓存。 • <i>LRU</i>：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（ <i>cacheSize</i> ）和缓存更新时间间隔（ <i>cacheTTLms</i> ）。
cacheSize	缓存大小。	否	当缓存策略选择 <i>LRU</i> 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	当缓存策略选择 <i>LRU</i> 时，可以设置缓存失效时间，默认不过期。
partitionedJoin	是否根据JoinKey进行分区。	否	参数的取值如下： <ul style="list-style-type: none"> • <i>false</i>（默认值）：不根据JoinKey进行分区。 • <i>true</i>：根据JoinKey进行分区，将数据分发到各JOIN节点，提高缓存命中率。
async	是否异步读取数据。	否	<i>async</i> 参数的取值如下： <ul style="list-style-type: none"> • <i>false</i>（默认值）：同步读取数据。 • <i>true</i>：异步读取数据。

类型映射

Hologres字段类型	实时计算Flink版字段类型
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

代码示例

实时计算Flink版包含Hologres维表的代码示例如下。

```
create table randomSource (a int, b VARCHAR, c VARCHAR) with (type = 'random');
create table test (
  a int,
  b VARCHAR,
  c VARCHAR,
  PRIMARY KEY (a, b), PERIOD FOR SYSTEM_TIME
) with (
  type = 'hologres',
  ...
);
create table print_sink (
  a int,
  b VARCHAR
) with (
  type = 'print',
  `ignoreWrite` = 'false'
);
insert into print_sink
select randomSource.a, test.b from randomSource
LEFT JOIN test FOR SYSTEM_TIME AS OF PROCTIME()
on randomSource.a = test.a and randomSource.b = test.b;
```

5.6.4.3. 创建表格存储Tablestore维表

本文为您介绍如何创建实时计算Flink版表格存储Tablestore维表。

 **重要** 本文仅适用于Blink 1.4.5及以上版本。

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上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

示例

实时计算Flink版支持表格存储Tablestore作为维表，示例如下。

```
CREATE TABLE ots_dim_table (
  id int,
  len int,
  content VARCHAR,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME--维表标识。
) WITH (
  type='ots',
  endPoint='<yourEndpoint>',
  instanceName='<yourInstanceName>',
  tableName='<yourTableName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessKey>'
);
```

说明

- 在声明维表时，必须要指名主键。
- 在维表JOIN时，ON条件必须包含所有主键的等值条件。
- Tablestore的主键即表的Rowkey。

WITH参数

参数	说明	备注
type	维表类型	固定值为 <code>ots</code> 。
endPoint	表格存储的实例访问地址	VPC网络环境需要选择实例的VPC Endpoint。
instanceName	表格存储的实例名称	无
tableName	表格存储的数据表名	无
accessId	表格存储读取的AccessKey ID	无
accessKey	表格存储读取的密钥AccessKey Secret	无

CACHE参数

参数	说明	备注
----	----	----

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： <ul style="list-style-type: none"> • <i>None</i>（默认值）：无缓存。 • <i>LRU</i>：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 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，默认为10000行。
cacheTTLms	缓存超时时间，单位为毫秒。	当选择LRU缓存策略后，可以设置缓存失效的超时时间。

代码示例

```

CREATE TABLE datahub_input1 (
  id      BIGINT,
  name    VARCHAR,
  age     BIGINT
) WITH (
  type='datahub'
);
CREATE TABLE phoneNumber(
  name VARCHAR,
  phoneNumber bigint,
  primary key(name),
  PERIOD FOR SYSTEM_TIME--维表标识。
)with(
  type='ots'
);
CREATE TABLE result_infor(
  id bigint,
  phoneNumber bigint,
  name VARCHAR
)with(
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;

```

维表的详细语法请参见 [维表JOIN语句](#)。

5.6.4.4. 创建云数据库RDS MySQL版维表

本文为您介绍如何创建实时计算云数据库RDS MySQL版维表，以及创建维表时使用的WITH参数、CACHE参数和类型映射。

云数据库RDS MySQL版

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。详情请参见[概述](#)。

使用限制

实时计算Flink版暂不支持通过数据存储功能中存储注册的方式使用RDS MySQL 8.0版本，建议您使用明文方式使用RDS MySQL 8.0版本。数据存储功能详情请参见[概述](#)。

语法示例

实时计算Flink版支持使用RDS MySQL版作为维表，示例代码如下。

```
CREATE TABLE rds_dim_table(  
  id INT,  
  len INT,  
  content VARCHAR,  
  PRIMARY KEY (id),  
  PERIOD FOR SYSTEM_TIME --定义维表的变化周期。  
) with (  
  type='rds',  
  url='<yourDatabaseURL>',  
  tableName='<yourDatabaseTableName>',  
  userName='<yourDatabaseUserName>',  
  password='<yourDatabasePassword>'  
);
```

 **说明** 声明维表时，必须要指名主键。维表JOIN时，ON条件必须包含所有主键的等值条件。RDS或DRDS的主键可以定义为表的主键或唯一索引列。

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为 <code>rds</code> 。

参数	说明	是否必填	备注
url	JDBC (Java DataBase Connectivity) 连接地址	是	URL的格式为： <code>jdbc:mysql://<内网地址>/<databaseName></code> ，其中databaseName为对应的数据库名称。内网地址参见如下链接： <ul style="list-style-type: none"> RDS的内网地址 DRDS的内网地址 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ? 说明 如果访问通过VPC访问授权过的RDS，对应的URL配置请参见VPC访问授权。 </div>
tableName	表名	是	无
userName	用户名	是	无
password	密码	是	无
maxRetries	最大尝试连接次数	否	默认值为10。

CACHE参数

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： <ul style="list-style-type: none"> 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）。 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。 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）。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ? 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。 </div>

参数	说明	是否必填	备注
cacheSize	缓存大小	否	当选择 LRU 缓存策略后，可以设置缓存大小，默认10000行。
cacheTTLms	缓存超时时间，单位为毫秒	否	当选择 LRU 缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择 ALL 策略，则为缓存加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	缓存策略选择 ALL 时启用。更新时间黑名单，防止在此时间内做 cache更新（例如双11场景）。	否	默认为空。自定义输入格式为 2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00 。用逗号 (,) 来分隔多个黑名单，用箭头 (->) 来分割黑名单的起始结束时间。
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。 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> <p>? 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。</p> </div>

? 说明 CACHE底层原理请参见 [维表JOIN 与异步优化](#)。

代码示例

包含云数据库RDS版维表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE datahub_input1 (
  id      BIGINT,
  name    VARCHAR,
  age     BIGINT
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);
create table phoneNumber(
  name VARCHAR,
  phoneNumber BIGINT,
  primary key(name),
  PERIOD FOR SYSTEM_TIME--定义维表的变化周期。
)WITH(
  type='rds',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)WITH(
  type='rds',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;
```

维表详细语法请参见 [维表JOIN语句](#)。

类型映射

RDS字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN

RDS字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

5.6.4.5. 创建云数据库HBase版维表

本文为您介绍如何创建实时计算云数据库HBase版维表，以及创建维表时使用的WITH参数和CACHE参数。

重要

- Blink 3.3.0以下版本仅支持HBase企业标准版。
- Blink 3.3.0及以上版本同时支持HBase企业标准版和HBase性能增强版。
- Blink 3.5.0及以上版本支持HBase写入主备切换。
- 云数据库HBase版维表的JOIN语法详情请参见 [维表JOIN语句](#)。
- 实时计算HBase维表不支持自建的开源HBase。
- HBase维表仅支持一个PK（Primary Key）。

DDL定义

- HBase企业标准版

```
CREATE TABLE hbase (  
  `key` varchar,  
  `name` varchar,  
  PRIMARY KEY (`key`), --HBase中的rowkey字段。  
  PERIOD FOR SYSTEM_TIME --维表标识。  
) with (  
  TYPE = 'cloudhbase',  
  zkQuorum = '<yourzkQuorum>',  
  columnFamily = '<yourColumnFamilyName>',  
  tableName = '<yourTableName>'  
);
```

- HBase性能增强版

```
CREATE TABLE hbase (  
  `key` varchar,  
  `name` varchar,  
  PRIMARY KEY (`key`), --HBase中的rowkey字段。  
  PERIOD FOR SYSTEM_TIME --维表标识。  
) with (  
  TYPE = 'cloudhbase',  
  endPoint = '<host:port>', --HBase增强版的Java API访问地址。  
  userName = 'root', --HBase用户名。  
  password = 'root', --HBase密码。  
  columnFamily = '<yourColumnFamilyName>',  
  tableName = '<yourTableName>'  
);
```

- Blink-3.5.0以上HBase性能增强版

```
create table liuxd_user_behavior_test_front (  
  row_key varchar,  
  from_topic varchar,  
  origin_data varchar,  
  record_create_time varchar,  
  primary key (row_key)  
) with (  
  type = 'cloudhbase',  
  zkQuorum = '<host:port>', --HBase增强版的Java API访问地址。  
  userName = 'root', --HBase用户名。  
  password = 'root', --HBase密码。  
  columnFamily = '<yourColumnFamily>',  
  tableName = '<yourTableName>',  
  batchSize = '500'  
);
```

- Blink-3.5.0以上支持HBase写入主备切换

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', --HBase高可用访问地址。
  haClusterID = 'ha-xxx', --HBase高可用实例ID。
  userName = 'root', --HBase用户名。
  password = 'root', --HBase密码。
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

说明

- 在声明维表时，必须要指名主键。
- 在维表进行JOIN时，ON的条件必须包含所有主键的等值条件。示例中HBase中的主键是row_key。
- HBase企业标准版和HBase性能增强版DDL的区别为连接参数不同：
 - HBase企业标准版：zkQuorum。
 - HBase性能增强版：endPoint。
 - Blink 3.5.0以上标准版和增强版：zkQuorum。

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为 cloudhbase 。
zkQuorum	HBase集群配置的zk地址，是以逗号(,)分隔的主机列表。	是	可以在hbase-site.xml文件中查看hbase.zookeeper.quorum相关配置。 
zkNodeParent	集群配置在zk上的路径	否	可以在hbase-site.xml文件中查看hbase.zookeeper.quorum相关配置。 

参数	说明	是否必填	备注
endPoint	HBase地域名称	是	可在购买的HBase实例控制台中获取。 说明 仅在HBase性能增强版中生效。
userName	用户名	否	说明 仅在HBase性能增强版中生效。
password	密码	否	说明 仅在HBase性能增强版中生效。
tableName	HBase表名	是	无
columnFamily	列族名	是	仅支持插入同一列族。
maxRetryTimes	最大尝试次数	否	默认值为10次。
partitionedJoin	是否使用joinKey进行分区。	否	默认值为False。在设置partitionedJoin为True时，使用joinKey进行分区，将数据分发到各JOIN节点，提高缓存命中率。
shuffleEmptyKey	是否将上游EMPTY KEY随机发送到下游节点。	否	默认值为True。参数取值如下： <ul style="list-style-type: none"> True: 如果上游有多个EMPTY KEY，会将所有EMPTY KEY随机发送到各个JOIN节点。 False: 如果上游有多个EMPTY KEY，会将所有EMPTY KEY发送至一个JOIN节点。 说明 shuffleEmptyKey在partitionedJoin生效后才能使用。

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	<p>HBase维表支持以下三种缓存策略：</p> <ul style="list-style-type: none"> • <i>None</i>（默认值）：无缓存。 • <i>LRU</i>：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。 <p>需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLms）。</p> <ul style="list-style-type: none"> • <i>ALL</i>：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。 <p>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。</p> <p>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）。</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>ⓘ 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。</p> </div>
cacheSize	缓存大小	否	当选择 <i>LRU</i> 缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLms	缓存超时时间，单位为毫秒。	否	当选择 <i>LRU</i> 缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择 <i>ALL</i> 策略，则为缓存加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	缓存策略选择 <i>ALL</i> 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	<p>默认为空。自定义输入格式如下。</p> <pre>2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00</pre> <p>使用逗号（,）分隔多个黑名单，使用箭头（->）分割黑名单的起始和结束时间。</p>
cacheScanLimit	缓存策略选择 <i>ALL</i> 时启用。读取全量HBase数据，服务端一次RPC返回给客户端的行数。	否	默认值为100条。

代码示例

包含HBase维表的实时计算作业代码示例如下。

```
create table source (  
  id TINYINT,  
  name BIGINT  
) with (  
  type = 'random'  
);  
create table dim (  
  id TINYINT,  
  score BIGINT  
  primary key(id),  
  PERIOD FOR SYSTEM_TIME  
)with(  
  type = 'cloudhbase',  
  zkQuorum = '<yourzkQuorum>',  
  columnFamily = '<yourColumnFamilyName>',  
  tableName = '<yourTableName>'  
);  
CREATE table result_infor(  
  id BIGINT,  
  score BIGINT  
)with(  
  type='rds'  
);  
INSERT INTO result_infor  
SELECT  
  t.id,  
  w.score  
FROM source as t  
JOIN dim FOR SYSTEM_TIME AS OF PROCTIME() as w  
ON t.id = w.id;
```

5.6.4.6. 创建MaxCompute维表

本文为您介绍如何创建实时计算Flink版MaxCompute维表，以及创建维表时使用的WITH参数、CACHE参数和类型映射。

重要

- Blink 2.1.1及以上版本支持MaxCompute维表。
- 维表的Query语法请参见 [维表JOIN语句](#)。
- 使用MaxCompute表作为维表，需要先赋予MaxCompute账号读权限。

DDL定义

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME --维表的标识。
) WITH (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****',
  cache = 'ALL'
);
```

说明

- 声明维表时，必须要指名主键，MaxCompute维表主键必须具有唯一性，否则会被去重。
- 在维表进行JOIN时，ON条件必须包含所有主键的等值条件。
- partition是关键字，需要使用反引号（`）注释，例如 `partition`。
- 如果是分区表，目前不支持将分区列写入到DDL定义中。

WITH参数

参数	说明	是否必填	备注
type	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。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 VPC环境下为必填。 </div>
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	<ul style="list-style-type: none"> 固定分区 <ul style="list-style-type: none"> 只存在一个分区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。 存在多个分区的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。 非固定分区 <ul style="list-style-type: none"> Blink 2.2.0及以上版本支持 `partition` = 'max_pt()' 功能，即每次加载所有分区列表中字典序最大的分区。 Blink 3.2.2及以上版本支持 `partition` = 'max_pt_with_done()' 功能，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。 <p>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。</p>
maxRowCount	可加载的最大表格数量。	否	<p>默认值为100000。</p> <p>说明 如果您的数据超过100000，需要设置 maxRowCount 参数。建议设定值比实际值大。</p>

CACHE参数

参数	说明	备注
----	----	----

参数	说明	备注
cache	缓存策略	<p>目前MaxCompute维表仅支持 ALL 策略，必须显式声明。</p> <p>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。</p> <p>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。</p> <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 10px; margin-top: 10px;"> <p>? 说明</p> <ul style="list-style-type: none"> • 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： <ul style="list-style-type: none"> ◦ Blink 3.6.0及以后版本，设置参数partitionedJoin = 'true'，即打开PartitionedJoin优化。 ◦ 改为支持LRU缓存策略的KV型维表，例如云数据库Hbase版维表。 </div>
cacheSize	缓存大小	可以设置缓存大小，MaxCompute默认缓存值为100000行。
cacheTTLms	缓存超时时间	单位为毫秒，当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）。	<p>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：</p> <ul style="list-style-type: none"> • 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。
partitionedJoin	当开启PartitionedJoin优化时，每个并发内存里只缓存维表的部分数据，即该并发上需要的缓存数据。	可选，默认值为false，表示每个并发内存里缓存全量维表数据。

代码示例

包含MaxCompute维表的实时计算Flink版作业代码示例如下。

```

CREATE TABLE datahub_input1 (
  id BIGINT,
  name VARCHAR,
  age BIGINT
) with (
  type='datahub'
);
CREATE TABLE odps_dim (
  name VARCHAR,
  phoneNumber BIGINT,
  PRIMARY KEY (name),
  PERIOD FOR SYSTEM_TIME --维表的标识。
) with (
  type = 'odps',
  endPoint = '<yourEndpointName>',
  project = '<yourProjectName>',
  tableName = '<yourTableName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessPassword>',
  `partition` = 'ds=20180905', --动态分区、固定分区请参见WITH参数说明。
  cache = 'ALL'
);
CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)with(
  type='print'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN odps_dim FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;
    
```

类型映射

MaxCompute和Flink全托管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。

MaxCompute	BLINK
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT

MaxCompute	BLINK
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

 说明 实时计算Flink版MaxCompute维表仅支持上述MaxCompute字段类型。

常见问题


- max_pt() 和 max_pt_with_done() 的区别是什么？

max_pt() 仅仅选取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。

分区列表
ds=20190101
ds=20190101.done
ds=20190102
ds=20190102.done
ds=20190103

示例中 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 。

 说明 仅Blink 3.3.2及以上版本支持 partition='max_pt_with_done()' 功能。

- Q：如何查看MaxCompute分区名？

A：查看MaxCompute分区名步骤如下：

- 在数据地图，搜索表名称。
- 单击目标表名。

iii. 在数据表详情页面，右侧 明细信息 > 分区信息 > 分区名查看MaxCompute分区名。

- Q: 任务出现了 RejectedExecutionException: Task java.util.concurrent.ScheduledThreadPoolExecutor\$ScheduledFutureTask 的Failover该怎么处理？

A: Blink 1.0版本中维表JOIN存在一定的问题，推荐升级到Blink 2.1.1及以上版本。如果需要继续使用原有版本，需要对作业进行暂停恢复操作，根据Failover History中第一条报错信息进行排查。

- Q: 公共云的endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？

A: endPoint和tunnelEndpoint参数说明参见 [Endpoint](#)。VPC环境中，如果这两个参数配置错误可能会导致任务异常：

- endPoint配置错误：任务上线停滞在91%的进度。
- tunnelEndpoint配置错误：任务运行失败。

- Q: 作业运行过程中报错ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:****'

A: 该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见[用户认证](#)。

5.6.4.7. 创建云数据库Redis维表

本文为您介绍如何创建云数据库Redis维表，以及创建过程中涉及的WITH参数、CACHE参数、类型映射和代码示例。

重要

- 本文仅适用于Blink 3.2.2及以上版本。
- 实时计算Flink版Redis维表仅支持引用Redis数据存储中STRING类型的数据。
- 实时计算Flink版Redis维表支持自建Redis服务。

语法示例

实时计算Flink版支持使用Redis作为数据存储维表。Redis维表语法示例如下。

```
CREATE TABLE white_list (  
  id VARCHAR,  
  name VARCHAR,  
  PRIMARY KEY (id), --Redis中的Row Key字段。  
  PERIOD FOR SYSTEM_TIME --维表标识。  
) WITH (  
  type = 'redis',  
  host = '<yourHostName>',  
  port = '<yourPort>',  
  password = '<yourPassword>',  
  dbName = '<yourDatabaseNumber>'  
);
```

说明

- Redis维表必须声明且只能声明一个主键。
- 维表JOIN时，ON条件必须包含所有主键的等值条件。
- Redis维表仅支持声明两个字段，且字段类型必须为VARCHAR。

WITH参数

参数	说明	是否必填	备注
type	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。
dbName	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。
hashName	Hash模式下的Hash Key名称	否	<p>默认值为空，实时计算Flink版从Redis中读取STRING类型的数据。</p> <p>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：</p> <ul style="list-style-type: none"> • 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	<p>云数据库Redis维表支持以下两种缓存策略：</p> <ul style="list-style-type: none"> • <i>None</i>（默认值）：无缓存。 • <i>LRU</i>：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 <p>需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLms）。</p>
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	否	默认缓存不超时，单位为毫秒。可选LRU缓存策略，即设置缓存失效的超时时长。

参数	说明	是否必填	备注
cacheEmpty	是否缓存空结果	否	默认值为true。

类型映射

Redis和实时计算Flink版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。

Redis字段类型	实时计算Flink版字段类型
STRING	VARCHAR

代码示例

包含Redis维表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE event (
  id VARCHAR,
  data VARCHAR) with (
  type = 'random'
);
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id), --Redis中的Row Key字段。
  PERIOD FOR SYSTEM_TIME --维表的标识。
) WITH (
  type = 'redis',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

5.6.4.8. 创建Elasticsearch维表

本文为您介绍如何创建实时计算Flink版Elasticsearch（ES）维表，以及创建维表时使用的WITH参数和CACHE参数。

 **重要** 本文仅适用于Blink 3.2.2及以上版本。

DDL定义

实时计算Flink版支持使用ES作为维表，示例代码如下。

```
CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type = 'elasticsearch',
  endPoint = '<yourEndPoint>',
  accessId = '<yourUsername>',
  accessKey = '<yourPassword>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

 **说明** ES维表支持根据ES的PRIMARY KEY进行PRIMARY KEYUPDATE，且PRIMARY KEY只能为1个字段。

WITH参数

参数	说明	默认值	是否必选
type	维表类型	elasticsearch	是
endPoint	Server地址，例 如： <i>http://127.0.0.1:9211</i> 。	无	是
accessId	创建ES时的登录名。  说明 如果您通过Kibana插件操作ES，请填写Kibana登录ID。	无	是
accessKey	创建ES时的登录密码。  说明 如果您通过Kibana插件操作ES，请填写Kibana登录密码。	无	是
index	索引名称，类似于数据库Database的名称。	无	是
typeName	Type名称，类似于数据库的Table名称。	无	是
maxRetryTimes	异常重试次数	30	否
timeout	读取超时时长，单位为毫秒。	600000	否
discovery	是否开启节点发现。如果开启，客户端每5分钟刷新一次Server List。	false	否

参数	说明	默认值	是否必选
compression	是否使用GZIP压缩Request Bodies。	true	否
multiThread	是否开启JestClient多线程。	true	否

CACHE参数

参数	说明	备注
cache	缓存策略	<ul style="list-style-type: none"> None (默认值)：无缓存。 LRU：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLms）。 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。
cacheSize	缓存大小	选择 LRU 缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLms	缓存更新时间间隔	默认缓存不超时，单位为毫秒。不同缓存策略下的功能如下： <ul style="list-style-type: none"> LRU：设置缓存失效的超时时长。 ALL：设置缓存加载的间隔时长，默认不重新加载。

5.6.4.9. 创建Phoenix5维表

本文为您介绍如何创建实时计算Flink版Phoenix5维表，以及创建维表时使用的WITH参数和CACHE参数。

 **重要** 仅Blink 3.4.0以上版本支持Phoenix5维表。

语法示例


```
create table US_POPULATION_DIM (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<YourServerUrl>',
  tableName = '<YourTableName>'
);
```

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为 PHOENIX5 。
serverUrl	Phoenix5的Query Server地址： <ul style="list-style-type: none"> 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	<p>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。</p> <p>serverUrl格式为http://host:port，其中：</p> <ul style="list-style-type: none"> host: Phoenix5服务的域名。 port: Phoenix5服务的端口号，固定值为8765。
tableName	Phoenix5表名	是	<p>Phoenix5表名格式为SchemaName.TableName，其中：</p> <ul style="list-style-type: none"> SchemaName: 模式名，可以为空，即不写模式名，仅写表名，表示使用数据库的默认模式。 TableName: 表名。

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	<p>目前Phoenix5维表支持以下三种缓存策略：</p> <ul style="list-style-type: none"> 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）。 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。 需要配置相关参数：缓存更新时间间隔（cacheTTLms），更新时间黑名单（cacheReloadTimeBlackList）。 <p>? 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。</p>
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	缓存超时时间，单位为毫秒。	否	当选择LRU缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择ALL策略，则为缓存加载的间隔时间，默认不重新加载。
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）。	否	可选，默认空，格式为 '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'。用逗号（,）来分隔多个黑名单，用箭头（->）来分割黑名单的起始或结束时间。

代码示例

```
CREATE TABLE datahub_input1 (  
  id BIGINT,  
  name VARCHAR,  
  age BIGINT  
) WITH (  
  type='datahub'  
);  
create table phoneNumber(  
  name VARCHAR,  
  phoneNumber BIGINT,  
  primary key(name),  
  PERIOD FOR SYSTEM_TIME--定义维表的变化周期。  
)with(  
  type='PHOENIX5'  
);  
CREATE table result_infor(  
  id BIGINT,  
  phoneNumber BIGINT,  
  name VARCHAR  
)with(  
  type='rds'  
);  
INSERT INTO result_infor  
SELECT  
  t.id,  
  w.phoneNumber,  
  t.name  
FROM datahub_input1 as t  
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定该声明。  
ON t.name = w.name;
```

5.6.4.10. 创建云原生数据仓库AnalyticDB MySQL版3.0维表

本文为您介绍如何创建云原生数据仓库AnalyticDB MySQL版3.0维表、以及创建维表时使用的WITH参数和CACHE参数。



重要 本文仅适用于Blink-3.5.0-hotfix及以上版本。

语法示例

```
CREATE TABLE dim_ads(
  `name` VARCHAR,
  id VARCHAR,
  PRIMARY KEY (`name`),
  PERIOD FOR SYSTEM_TIME
)with(
  type='ADB30',
  url='jdbc:mysql://<内网地址>/<databaseName>',
  tableName='xxx',
  userName='xxx',
  password='xxx'
);
```

说明

- 在声明一个维表时，必须指明主键。
- 在维表进行JOIN时，ON条件必须包含所有主键的等值条件。
- 云原生数据仓库AnalyticDB MySQL版的主键可以定义为表的主键或唯一索引列。

WITH参数

参数	说明	是否必选	备注
type	维表类型。	是	固定值为ADB30。
url	云原生数据仓库AnalyticDB MySQL版数据库地址。	是	云原生数据仓库AnalyticDB MySQL版数据库地址。示例： <code>url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'</code> 。 说明 <ul style="list-style-type: none"> • 云原生数据仓库AnalyticDB MySQL版数据库连接信息，请参见注册云原生数据仓库AnalyticDB MySQL版。 • databaseName：云原生数据仓库AnalyticDB MySQL版数据库名称。
tableName	表名。	是	无。
userName	用户名。	是	无。
password	密码。	是	无。
maxRetryTimes	写入重试次数。	否	默认值为3。

CACHE参数

参数	说明	是否必填	备注
cache	缓存策略	否	<p>目前云原生数据仓库AnalyticDB MySQL版3.0支持以下三种缓存策略：</p> <ul style="list-style-type: none"> • <i>None</i>（默认值）：无缓存。 • <i>LRU</i>：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLms）。 • <i>ALL</i>：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。 需要配置相关参数：缓存更新时间间隔（cacheTTLms），更新时间黑名单（cacheReloadTimeBlackList）。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 10px; margin-top: 10px;"> <p>说明</p> <ul style="list-style-type: none"> • 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。 • 对于数据量比较大的维表，选择CACHE ALL时，可能会出现OOM或者Full GC耗时很久的情况，针对这个问题，可以选择以下两种解决方式： <ul style="list-style-type: none"> ◦ 对于支持Cache All策略的维表，开启PartitionedJoin优化。3.6.0版本之前，每个并发默认加载维表全量数据。3.6.0版本之后，CACHE ALL策略支持PartitionedJoin优化。开启PartitionJoin优化后，每个并发只缓存自己并发所需要的数据。 ◦ 使用HBase或者RDS等Key-Value类型的维表。 </div>
cacheSize	缓存大小	否	当选择 <i>LRU</i> 缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLms	缓存更新时间间隔。系统会根据您设置的缓存更新时间间隔，重新加载一次维表中的最新数据，保证源表能JOIN到维表的最新数据。	否	单位为毫秒。默认不设置此参数，表示不重新加载维表中的新数据。

参数	说明	是否必填	备注
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	可选，默认空，格式为 '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'。其中分割符使用情况如下： <ul style="list-style-type: none"> 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。
partitionedJoin	是否开启partitionedJoin。在开启partitionedJoin优化时，主表会在关联维表前，先按照Join KEY进行Shuffle，这样做有以下优点： <ul style="list-style-type: none"> 在缓存策略为LRU时，可以提高缓存命中率。 在缓存策略为ALL时，节省内存资源，因为每个并发只缓存自己并发所需要的数据。 	否	默认情况下为false，表示不开启partitionedJoin。 ? 说明 使用partitionedJoin优化前，需要您手动设置partitionedJoin = 'true'。
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。

代码示例

```
CREATE TABLE datahub_input1 (  
  id BIGINT,  
  name VARCHAR,  
  age BIGINT  
) WITH (  
  type='datahub'  
);  
create table phoneNumber (  
  name VARCHAR,  
  phoneNumber BIGINT,  
  primary key(name),  
  PERIOD FOR SYSTEM_TIME--维表标识。  
) with (  
  type='ADB30'  
);  
CREATE table result_infor (  
  id BIGINT,  
  phoneNumber BIGINT,  
  name VARCHAR  
) with (  
  type='rds'  
);  
INSERT INTO result_infor  
SELECT  
  t.id,  
  w.phoneNumber,  
  t.name  
FROM datahub_input1 as t  
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定该声明。  
ON t.name = w.name;
```

5.6.4.11. 创建Oracle维表

本文为您介绍如何创建Oracle维表以及创建过程中涉及到的WITH参数、类型映射和属性字段等。

DDL定义

```
CREATE TABLE oracle_dim(  
  employee_id BIGINT,  
  phone_number BIGINT,  
  dollar DOUBLE,  
  PRIMARY KEY (employee_id)  
) WITH (  
  type = 'oracle_dim',  
  url = '<yourUrl>',  
  userName = '<yourUserName>',  
  password = '<yourPassword>',  
  tableName = '<yourTableName>',  
  cache = 'ALL'  
);
```

WITH 参数

参数	描述	是否必选	示例值
type	维表类型	是	固定值为oracle_dim。
url	JDBC的Oracle地址	是	jdbc:oracle:thin:@ip:port:sid
userName	数据库的用户名	是	无
password	数据库的密码	是	无
tableName	表名称	是	无
maxRetryTimes	读取维表异常重试的最大次数	否	默认值为10。

CACHE参数

参数	描述	是否必选	示例值
cache	缓存策略	否	<p>目前Oracle维表支持以下三种缓存策略：</p> <ul style="list-style-type: none"> 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)。 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，则关键健不存在，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)。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>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。</p> </div>
cacheSize	缓存大小，即缓存多少行数据。	否	当缓存策略选择LRU时，可以设置缓存大小，默认值为10000行。

参数	描述	是否必选	示例值
cacheTTLms	缓存超时时间，单位为毫秒。	否	<ul style="list-style-type: none"> 当缓存策略选择<code>LRU</code>时，可以设置缓存失效时间，默认不过期。 当缓存策略选择<code>ALL</code>时，缓存失效时间为缓存重新加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为 <code>ALL</code> 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'。分割符如下： <ul style="list-style-type: none"> 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。
maxJoinRows	一对多连接时，左表一条记录连接右表的最大记录数。	否	默认值为1024。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ? 说明 一对多连接的记录数过多时，需要调cache的内存。因为cacheSize限制的是左表key个数，单条左表记录对应的右表记录较多时，可能会极大地影响流任务的性能。 </div>

类型映射

Oracle字段类型	实时计算Flink字段类型
<ul style="list-style-type: none"> CHAR VARCHAR VARCHAR2 	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL


代码示例

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT
) WITH (
  type = 'random'
);
CREATE TABLE oracle_dim (
  employee_id BIGINT,
  phone_number BIGINT,
  dollar DOUBLE,
  PRIMARY KEY (employee_id)
) WITH (
  type = 'oracle_dim',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>',
  cache = 'ALL'
);
CREATE TEMPORARY TABLE oracle_sink (
  employee_id BIGINT,
  phone_number BIGINT,
  employee_name VARCHAR
) WITH (
  type = 'oracle',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>'
);
INSERT INTO oracle_sink
SELECT t.employee_id, w.phone_number, t.employee_name
FROM oracle_source as t JOIN oracle_dim FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.employee_id = w.employee_id;
```

5.7. DML语句

5.7.1. EMIT语句

EMIT语句可以使QUERY根据不同场景，定义不同的输出策略，从而达到控制延迟或提高数据准确性的效果。

 **重要** EMIT语句仅支持实时计算2.0以上版本。

使用限制

- EMIT策略只支持TUMBLE和HOP窗口，暂不支持SESSION窗口。
- 如果一个Job有多个输出，则多个输出的EMIT需要定义成相同策略，后续会支持不同策略。
- EMIT语法还不能用来配置minibatch的allowLateness，后续计划使用EMIT策略来声明allowLateness。

什么是EMIT策略

EMIT策略是指在Flink SQL中，QUERY根据不同场景选择不同的输出策略（例如最大延迟时长）。传统的ANSI SQL语法不支持该类输出策略。例如，1小时的时间窗口，窗口触发之前希望每分钟都能看到最新的结果，窗口触发之后希望不丢失迟到一天内的数据。如果1小时窗口内的统计结果无变化，则不更新输出结果；如果1小时窗口内的统计结果有变化，则更新输出结果。

针对这类场景，实时计算抽象出EMIT语法，并扩展到SQL语法。以下为不同场景下EMIT策略的示例：

- 窗口结束之前，按1分钟延迟输出，窗口结束之后无延迟输出。

```
EMIT
WITH DELAY '1' MINUTE BEFORE WATERMARK,
WITHOUT DELAY AFTER WATERMARK
```

- 窗口结束之前不输出，窗口结束之后无延迟输出。

```
EMIT WITHOUT DELAY AFTER WATERMARK
```

- 全局都按1分钟的延迟输出（您可以启用minibatch参数来增加延迟）。

```
EMIT WITH DELAY '1' MINUTE
```


- 窗口结束之前按1分钟延迟输出。

```
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK
```

EMIT语法的用途

EMIT语法能够实现以下两种功能：

- 控制延迟：针对窗口，设置窗口触发之前的EMIT输出频率，减少结果输出延迟。
- 提高数据精确性：不丢弃窗口触发之后的迟到的数据，修正输出结果。

 **说明** 选择EMIT策略时，请权衡业务复杂程度和资源消耗情况。越低的输出延迟、越高的数据精确性，需要提供越高的计算开销。

EMIT语法

EMIT语法定义在INSERT INTO输出语句中，定义输出结果的策略。当INSERT INTO中未配置EMIT语法时，保持原有默认行为，即只在WATERMARK触发时，输出一个窗口结果。

 **说明** EMIT语句只能置于INSERT INTO语句中的所有QUERY语句之后，不能置于VIEW语句中。

```
INSERT INTO tableName
<Query>
EMIT strategy [, strategy]*
strategy ::= {WITH DELAY timeInterval | WITHOUT DELAY}
           [BEFORE WATERMARK | AFTER WATERMARK]
timeInterval ::= 'string' timeUnit
```

参数	说明
WITH DELAY	可延迟输出，即按指定时间间隔输出。
WITHOUT DELAY	不可以延迟输出，即每来一条数据就输出。
BEFORE WATERMARK	窗口结束之前的策略配置，即WATERMARK触发之前。
AFTER WATERMARK	窗口结束之后的策略配置，即WATERMARK触发之后。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>? 说明 如果配置了AFTER WATERMARK策略，需要使用明文方式声明 <code>blink.state.ttl.ms</code>，标识最大延迟时长。</p> </div>

其中 `strategy` 的配置方式包括以下几种：

- 配置为一个BEFORE。
- 配置为一个AFTER。
- 配置为一个BEFORE和一个AFTER。

? 说明 `strategy` 不支持同时配置为多个BEFORE或者多个AFTER。

生命周期

AFTER策略允许接收迟到的数据，窗口的状态（State）允许保留一定时长，等待迟到的数据。这段保留的时长称为生命周期TTL。运用AFTER策略后，通过明文声明 `blink.state.ttl.ms` 参数，您可以设置状态允许的生命周期。例如，`blink.state.ttl.ms=3600000` 表示状态允许保留超时时长为1小时内的数据，超时时长大于1小时的数据不被录入状态。

EMIT语法示例

窗口区间为1小时的滚动窗口 `tumble_window` 的语法示例如下。

```
CREATE VIEW tumble_window AS
SELECT
  `id`,
  TUMBLE_START(rowtime, INTERVAL '1' HOUR) as start_time,
  COUNT(*) as cnt
FROM source
GROUP BY `id`, TUMBLE(rowtime, INTERVAL '1' HOUR);
```

默认 `tumble_window` 的输出需要等到1小时结束才能显示。如果您需要尽早看到窗口的结果（即使是不完整的结果），例如每分钟看到最新的窗口结果，可以添加如下语句。

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK; --窗口结束之前，每隔1分钟输出一更新结果。
```

默认 `tumble_window` 会忽略并丢弃窗口结束后到达的数据，如果您需要将窗口结束后1天到达的数据统计进入结果，并且需要每接收1条数据后立刻更新结果，可以添加如下语句。

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK,
WITHOUT DELAY AFTER WATERMARK; --窗口结束后，每收到一条数据输出一次更新结果。
```

此外，您还需要在作业参数中配置 `blink.state.ttl.ms = 86400000`（增加1天状态生命周期）。

DELAY概念

EMIT策略中的 DELAY 指的是用户可接受的数据延迟时长，该延迟是指从用户的数据进入实时计算，到看到结果数据（从实时计算系统输出）的时间（Event Time或Processing Time）。延迟的计算基于系统时间。动态表（流式数据在实时计算内部的存储）中的数据发生变化的时间和结果表（实时计算外部的存储）中显示新记录的时间的间隔，称为延迟。

假设，实时计算系统的处理耗时是0，则在流式数据积攒和Window等待窗口数据的过程可能会导致延迟。如果您指定了最多延迟30秒，则30秒可用于流式数据的积攒。如果Query是1小时的窗口，则最多延迟30秒的含义是每隔30秒更新结果数据。

- 配置 EMIT WITH DELAY '1' MINUTE

对于Group By聚合，系统会在1分钟内积攒流式数据。如果有Window且Window的Size大于1分钟，Window就每隔1分钟更新一次结果数据。如果Window的Size小于1分钟，因为窗口依靠Watermark的输出就能保证Latency SLA，所以系统就会忽略这个配置。

- 配置 EMIT WITHOUT DELAY

对于Group By聚合，不会启用minibatch参数来增加延迟，每来一条数据都会触发计算和输出。对于Window函数，也是每来一条数据都触发计算和输出。

5.7.2. INSERT INTO语句

本文为您介绍在实时计算Flink版中如何使用INSERT INTO语句及使用限制。

操作约束

表类型	使用限制
源表	只能引用（FROM），不可执行INSERT。
维表	只能引用（JOIN），不可执行INSERT。
结果表	仅支持INSERT操作。
视图	只能引用（FROM）。

语法

```
INSERT INTO tableName
[(columnName[, columnName]*)]
queryString;
```

示例

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
INSERT INTO Orders(z,v)
SELECT c,d FROM OO;
```

说明

- 单个实时计算Flink版作业支持在一个SQL作业中包含多个DML操作，同样也允许包含多个数据源、数据目标端和维表。例如，一个作业文件中包含两段业务上完全独立的SQL，分别写到不同的数据目标端。
- 实时计算Flink版不支持单独的SELECT查询，必须在CREATE VIEW或INSERT INTO内才能操作。
- INSERT INTO支持UPDATE更新。例如，向设置了主键字段的RDS结果表中插入一个KEY值。如果这个KEY值存在就更新，如果不存在就插入一条新的KEY值。

5.8. QUERY语句

5.8.1. SELECT语句

SELECT语句用于从表中选取数据。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

测试数据

t

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

简单查询

- 测试语句

```
SELECT * FROM t;
```

- 测试结果

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

重命名查询

- 测试语句

```
SELECT a, c AS d FROM t;
```

- 测试结果

a (VARCHAR)	d (DATE)
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

去重查询

- 测试语句

```
SELECT DISTINCT a FROM t;
```

- 测试结果

a (VARCHAR)
a1
b1
c1

子查询

普通的SELECT是从几张表中读数据，例如 `SELECT column_1, column_2 ... FROM table_name`，但查询的对象也可以是另外一个SELECT操作。


 **说明** 当查询的对象是另一个SELECT操作时，必须为子查询加别名。

- 测试语句

```
INSERT INTO result_table
SELECT * FROM
  (SELECT t.a,
    sum(t.b) AS sum_b
  FROM t1 t
  GROUP BY t.a
  ) t1
WHERE t1.sum_b > 100;
```

- 测试结果

a (VARCHAR)	b (INT)
a1	211
b1	120
a1	257

 **说明** 此结果为调试结果，会显示出计算过程。如果您的结果表是DataHub、消息队列Kafka或消息队列MQ等，正式上线也会显示过程数据。但如果您的结果表是云数据RDS等关系型数据库，正式上线，主键相同的记录显示为一条数据。

5.8.2. WHERE语句

WHERE语句可用于对SELECT语句中的数据进行筛选。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

下面的运算符可在WHERE语句中使用。

操作符	描述
=	等于
<>	不等于
>	大于
>=	大于等于
<	小于
<=	小于等于

示例

- 测试数据

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	shanghai

- 测试语句

```
SELECT * FROM XXXX WHERE City='Beijing';
```

- 测试结果

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

5.8.3. HAVING语句

在使用聚合函数时，您需要添加HAVING语句，以实现与WHERE语句一样的过滤效果。

语法

```
SELECT [ ALL | DISTINCT ] { * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ WHERE booleanExpression ]  
[ GROUP BY { groupItem [, groupItem ]* } ]  
[ HAVING booleanExpression ];
```

示例

- 测试数据

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000

Customer	OrderPrice
Carter	100

- 测试语句

```
SELECT Customer,SUM(OrderPrice) FROM XXX  
GROUP BY Customer  
HAVING SUM(OrderPrice)<2000;
```

- 测试结果

Customer	SUM (OrderPrice)
Carter	1700

5.8.4. GROUP BY语句

GROUP BY语句用于根据一个或多个列对结果集进行分组。

语法

```
SELECT [ DISTINCT ]  
{ * | projectItem [, projectItem ]* }  
FROM tableExpression  
[ GROUP BY { groupItem [, groupItem ]* } ];
```

示例

- 测试数据（T1）

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- 测试语句

```
SELECT Customer,SUM(OrderPrice) FROM T1  
GROUP BY Customer;
```

- 测试结果

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

5.8.5. JOIN语句

实时计算的JOIN和传统批处理JOIN的语义一致，都用于将两张表关联起来。区别为实时计算关联的是两张动态表，关联的结果也会动态更新，以保证最终结果和批处理结果一致。

语法

```
tableReference [, tableReference ]* | tableexpression  
[ LEFT ] JOIN tableexpression [ joinCondition ];
```

- tableReference: 表名称。
- tableexpression: 表达式。
- joinCondition: JOIN条件。

重要

- 只支持等值连接，不支持非等值连接。
- 只支持INNER JOIN和LEFT OUTER JOIN两种JOIN方式。

Orders JOIN Products表的数据示例

- 测试数据 Orders

rowtime	productId	orderId	units
10:17:00	30	5	4
10:17:05	10	6	1
10:18:05	20	7	2
10:18:07	30	8	20
11:02:00	10	9	6
11:04:00	10	10	1
11:09:30	40	11	12
11:24:11	10	12	4

Products

productId	name	unitPrice
30	Cheese	17
10	Beer	0.25
20	Wine	6
30	Cheese	17
10	Beer	0.25
10	Beer	0.25
40	Bread	100
10	Beer	0.25

- 测试语句

```
SELECT o.rowtime, o.productId, o.orderId, o.units, p.name, p.unitPrice
FROM Orders AS o
JOIN Products AS p
ON o.productId = p.productId;
```

- 测试结果

o.rowtime	o.productId	o.orderId	o.units	p.name	p.unitPrice
10:17:00	30	5	4	Cheese	17
10:17:05	10	6	1	Beer	0.25
10:18:05	20	7	2	Wine	6
10:18:07	30	8	20	Cheese	17
11:02:00	10	9	6	Beer	0.25
11:04:00	10	10	1	Beer	0.25
11:09:30	40	11	12	Bread	100
11:24:11	10	12	4	Beer	0.25

datahub_stream1 JOIN datahub_stream2表的数据示例

- 测试数据 datahub_stream1

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21

datahub_stream2

a (BIGINT)	b (BIGINT)	c (VARCHAR)
0	10	test11
1	10	test21
0	10	test31
1	10	test41

- 测试语句


```
SELECT s1.c,s2.c
FROM datahub_stream1 AS s1
JOIN datahub_stream2 AS s2
ON s1.a =s2.a
WHERE s1.a = 0;
```

- 测试结果

s1.c (VARCHAR)	s2.c (VARCHAR)
test11	test11
test11	test31

5.8.6. 维表JOIN语句

对于每条流式数据，可以关联一个外部维表数据源，为实时计算Flink版提供数据关联查询。

 **说明** 维表是一张不断变化的表，在维表JOIN时，需指明该条记录关联维表快照的时刻。维表JOIN仅支持对当前时刻维表快照的关联，未来会支持关联左表rowtime所对应的维表快照。关于维表的详细介绍，请参见概述。

维表JOIN语法

```
SELECT column-names
FROM table1 [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF PROCTIME() [AS <alias2>]
ON table1.column-name1 = table2.key-name1;
```

事件流JOIN白名单维表，示例如下。

```
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

说明

- 维表支持 INNER JOIN 和 LEFT JOIN ，不支持 RIGHT JOIN 或 FULL JOIN 。
- 必须加上 FOR SYSTEM_TIME AS OF PROCTIME() ，表示JOIN维表当前时刻所看到的每条数据。
- 源表后面进来的数据只会关联当时维表的最新信息，即JOIN行为只发生在处理时间（Processing Time）。如果JOIN行为发生后，维表中的数据发生了变化（新增、更新或删除），则已关联的维表数据不会被同步变化。
- ON条件中必须包含维表所有的PRIMARY KEY的等值条件（且要求与真实表定义一致）。此外，ON条件中也可以有其他等值条件。
- 如果您有一对多JOIN需求，请在维表DDL INDEX中指定关联的KEY，详情请参见 [INDEX语法](#)。
- 维表和维表不能进行JOIN。
- ON条件中维表字段不能使用CAST等类型转换函数。如果您有类型转换需求，请在源表字段进行操作。

示例

- 测试数据 datahub_input1

id (bigint)	name (varchar)	age (bigint)
1	lilei	22
2	hanmeimei	20
3	libai	28

phoneNumber

name (varchar)	phoneNumber (bigint)
dufu	13900001111
baijuyi	13900002222
libai	13900003333
lilei	13900004444

- 测试语句

```

CREATE TABLE datahub_input1 (
  id      BIGINT,
  name    VARCHAR,
  age     BIGINT
) WITH (
  type='datahub'
);
create table phoneNumber(
  name VARCHAR,
  phoneNumber bigint,
  primary key(name),
  PERIOD FOR SYSTEM_TIME
)with(
  type='rds'
);
CREATE table result_infor(
  id bigint,
  phoneNumber bigint,
  name VARCHAR
)with(
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.name = w.name;

```

- 测试结果

id (bigint)	phoneNumber (bigint)	name (varchar)
1	13900004444	lilei
3	13900003333	libai

5.8.7. IntervalJoin语句

IntervalJoin语句可以让两个流进行JOIN时，左流和右流中每条记录只关联另外一条流上同一时间段内的数据，且进行完JOIN后，仍然保留输入流上的时间列，让您继续进行基于Event Time的操作。

语法格式

```

SELECT column-names
FROM table1 [AS <alias1>]
[INNER | LEFT | RIGHT | FULL ] JOIN table2
ON table1.column-name1 = table2.key-name1 AND TIMEBOUND_EXPRESSION

```

说明

- 支持INNER JOIN、LEFT JOIN、RIGHT JOIN和FULL JOIN，如果直接使用JOIN，默认为INNER JOIN。
- 暂不支持SEMI JOIN和ANTI JOIN。
- TIMEBOUND_EXPRESSION为左右两个流时间属性列上的区间条件表达式，支持以下三种条件表达式：
 - ltime = rtime
 - ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
 - ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND

示例1（基于Event Time）

统计下单后4个小时内的物流信息。

- 测试数据
 - 订单表 (orders)

id	productName	orderTime
1	iphone	2020-04-01 10:00:00.0
2	mac	2020-04-01 10:02:00.0
3	huawei	2020-04-01 10:03:00.0
4	pad	2020-04-01 10:05:00.0

- 物流表 (shipments)

shipId	orderId	status	shiptime
0	1	shipped	2020-04-01 11:00:00.0
1	2	delivered	2020-04-01 17:00:00.0
2	3	shipped	2020-04-01 12:00:00.0
3	4	shipped	2020-04-01 11:30:00.0

- 测试语句


```
CREATE TABLE Orders(  
  id BIGINT,  
  productName VARCHAR,  
  orderTime TIMESTAMP,  
  WATERMARK wk FOR orderTime as withOffset(orderTime, 2000) --为rowtime定义Watermark。  
) WITH (  
  type='datahub',  
  endpoint='<yourEndpoint>',  
  accessId='<yourAccessID>',  
  accessKey='<yourAccessSecret>',  
  projectName='<yourProjectName>',  
  topic='<yourTopic>',  
  project='<yourProjectName>'  
);  
CREATE TABLE Shipments(  
  shipId BIGINT,  
  orderId BIGINT,  
  status VARCHAR,  
  shiptime TIMESTAMP,  
  WATERMARK wk FOR shiptime as withOffset(shiptime, 2000) --为rowtime定义Watermark。  
) WITH (  
  type='datahub',  
  endpoint='<yourEndpoint>',  
  accessId='<yourAccessID>',  
  accessKey='<yourAccessSecret>',  
  projectName='<yourProjectName>',  
  topic='<yourTopic>',  
  project='<yourProjectName>'  
);  
--使用RDS作为结果表  
CREATE TABLE rds_output(  
  id BIGINT,  
  productName VARCHAR,  
  status VARCHAR  
) WITH (  
  type='rds',  
  url='<yourDatabaseURL>',  
  tableName='<yourDatabaseTablename>',  
  userName='<yourDatabaseUserName>',  
  password='<yourDatabasePassword>'  
);  
INSERT INTO rds_output  
SELECT id, productName, status  
FROM Orders AS o  
JOIN Shipments AS s on o.id = s.orderId AND  
  o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime;
```

● 测试结果

id(bigint)	productName(varchar)	status(varchar)
1	iphone	shipped

id(bigint)	productName(varchar)	status(varchar)
3	huawei	shipped
4	pad	shipped

示例2（基于Processing Time）

- 测试数据

- datahub_stream1


k1	v1
1	val1
2	val2
3	val3

- datahub_stream2

k1	v1
1	val1
2	val2
3	val3

- 测试语句

```
CREATE TABLE datahub_stream1 (  
  k1 BIGINT,  
  v1 VARCHAR,  
  d AS PROCTIME()  
) WITH (  
  type='datahub',  
  endpoint='<yourEndpoint>',  
  accessId='<yourAccessID>',  
  accessKey='<yourAccessSecret>',  
  projectName='<yourProjectName>',  
  topic='<yourTopic>',  
  project='<yourProjectName>'  
);  
CREATE TABLE datahub_stream2 (  
  k2 BIGINT,  
  v2 VARCHAR,  
  e AS PROCTIME()  
) WITH (  
  type='datahub',  
  endpoint='<yourEndpoint>',  
  accessId='<yourAccessID>',  
  accessKey='<yourAccessSecret>',  
  projectName='<yourProjectName>',  
  topic='<yourTopic>',  
  project='<yourProjectName>'  
);  
--使用RDS作为结果表  
CREATE TABLE rds_output(  
  k1 BIGINT,  
  v1 VARCHAR,  
  v2 VARCHAR  
) WITH (  
  type='rds',  
  url='<yourDatabaseURL>',  
  tableName='<yourDatabaseTableName>',  
  userName='<yourDatabaseUserName>',  
  password='<yourDatabasePassword>'  
);  
INSERT INTO rds_output  
SELECT k1, v1, v2  
FROM datahub_stream1 AS o  
JOIN datahub_stream2 AS s on o.k1 = s.k2 AND  
  o.d BETWEEN s.e - INTERVAL '4' MINUTE AND s.e;
```

 **说明** 由于结果取决于两个流里每条数据进入系统的时间，具有不确定性，因此该示例暂不提供预期结果。

5.8.8. UNION ALL语句

UNION ALL语句将两个流式数据合并。两个流式数据的字段必须完全一致，包括字段类型和字段顺序。

语法

```
select_statement
UNION ALL
select_statement;
```

说明 实时计算Flink版同样支持 UNION 函数。UNION ALL 允许重复值，UNION 不允许重复值。在实时计算Flink版系统中，UNION 相当于 UNION ALL+Distinct，运行效率低，通常不推荐使用 UNION。

示例

- 测试数据 test_source_union1

a (varchar)	b (bigint)	c (bigint)
test1	1	10

test_source_union2

a (varchar)	b (bigint)	c (bigint)
test1	1	10
test2	2	20

test_source_union3

a (varchar)	b (bigint)	c (bigint)
test1	1	10
test2	2	20
test1	1	10

- 测试语句

```
SELECT
  a,
  sum(b) as d,
  sum(c) as e
FROM
  (SELECT * from test_source_union1
  UNION ALL
  SELECT * from test_source_union2
  UNION ALL
  SELECT * from test_source_union3
  )t
GROUP BY a;
```

• 测试结果

a (varchar)	d (bigint)	e (bigint)
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40
test1	4	40

② 说明 此结果为调试结果，会显示出计算过程。如果您的结果表是DataHub、消息队列Kafka或消息队列MQ等，正式上线也会显示过程数据。但如果您的结果表是云数据RDS等关系型数据库，正式上线，主键相同的记录显示为一条数据。

5.8.9. TopN语句

TopN语句用于对实时数据中某个指标的前N个最值的筛选。Flink SQL可以基于OVER窗口操作灵活地完成TopN的功能。

语法

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

② 说明

- ROW_NUMBER()：行号计算函数 OVER 的窗口，行号计算从1开始。
- PARTITION BY col1[, col2..]：指定分区的列，可以不指定。
- ORDER BY col1 [asc|desc][, col2 [asc|desc]...]：指定排序的列和每列的排序方向。

如上语法所示，TopN需要两层Query：

- 查询中使用 ROW_NUMBER() 窗口函数来对数据根据排序列进行排序并标上排名。
- 外层查询中对排名进行过滤，只取前N条。例如N=10即为取前10条的数据。

在执行过程中，Flink SQL会对输入的数据流根据排序键进行排序。如果某个分区的前N条记录发生了改变，则会将改变的那几条数据以更新流的形式发给下游。

 **说明** 如果需要将TopN的数据输出到外部存储，后接的结果表必须是一个带主键的表。

WHERE条件的限制

为了使Flink SQL能识别出这是一个TopN的query，外层循环中必须要指定 `rownum <= N` 的格式来指定前N条记录。此时，您不可以将 `rownum` 置于某个表达式中，例如 `rownum - 5 <= N`。WHERE条件中，可以额外带上其他条件，但是必须使用 `AND` 连接条件。

示例1

本例中，先统计查询流中每小时、每个城市和关键字被查询的次数，然后输出每小时、每个城市被查询最多的前100个关键字。在输出表中，小时、城市、排名三者可以唯一确定一条记录，所以需要在这三列声明成联合主键（在外部存储中也需要有同样的主键设置）。

```
CREATE TABLE rds_output (
  rownum BIGINT,
  start_time BIGINT,
  city VARCHAR,
  keyword VARCHAR,
  pv BIGINT,
  PRIMARY KEY (rownum, start_time, city)
) WITH (
  type = 'rds',
  ...
)
INSERT INTO rds_output
SELECT rownum, start_time, city, keyword, pv
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY start_time, city ORDER BY pv desc) AS rownum
  FROM (
    SELECT SUBSTRING(time_str,1,12) AS start_time,
      keyword,
      count(1) AS pv,
      city
    FROM tmp_search
    GROUP BY SUBSTRING(time_str,1,12), keyword, city
  ) a
) t
WHERE rownum <= 100
```

示例2

- 测试数据

ip (VARCHAR)	time (VARCHAR)
192.168.1.1	100000000
192.168.1.2	100000000
192.168.1.2	100000000

ip (VARCHAR)	time (VARCHAR)
192.168.1.3	100030000
192.168.1.3	100000000
192.168.1.3	100000000

- 测试语句

```
CREATE TABLE source_table (
  IP VARCHAR,
  `TIME` VARCHAR
)WITH(
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>'
);
CREATE TABLE result_table (
  rownum BIGINT,
  start_time VARCHAR,
  IP VARCHAR,
  cc BIGINT,
  PRIMARY KEY (start_time, IP)
) WITH (
  type = 'rds',
  url='<yourDatabaseAddress>',
  tableName='blink_rds_test',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO result_table
SELECT rownum,start_time,IP,cc
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY cc DESC) AS rownum
  FROM (
    SELECT SUBSTRING(`TIME`,1,2) AS start_time,--可以根据真实时间取相应的数值，这里取得是测试数据。
    COUNT(IP) AS cc,
    IP
    FROM source_table
    GROUP BY SUBSTRING(`TIME`,1,2), IP
  )a
  )t
WHERE rownum <= 3 --可以根据真实top值取相应的数值，这里取得是测试数据。
```

- 测试结果

rownum (BIGINT)	start_time (VARCHAR)	ip (VARCHAR)	cc (BIGINT)
1	10	192.168.1.3	3
2	10	192.168.1.2	2
3	10	192.168.1.1	1

无排名优化

- 使用无排名优化，您可以解决数据膨胀问题。

- 数据膨胀问题

根据TopN的语法，rownum 字段会作为结果表的主键字段之一写入结果表。但是这可能导致数据膨胀的问题。例如，收到一条原排名9的更新数据，更新后排名上升到1，则从1到9的数据排名都发生了变化，需要将这些数据作为更新都写入结果表。这样就产生了数据膨胀，导致结果表因为收到了太多的数据而降低更新速度。


- 无排名优化方法

结果表中不保存 rownum，最终的 rownum 由前端计算。因为TopN的数据量通常不会很大，前端排序100个数据很快。当收到一条原排名9、更新后排名上升到1的数据，也只需要发送这一条数据，而不用把排名1到9的数据全发送下去。这种优化能够提升结果表的更新速度。

- 无排名优化语法

```
SELECT col1, col2, col3
FROM (
  SELECT col1, col2, col3
  ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
  ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
FROM table_name)
WHERE rownum <= N [AND conditions]
```

语法与上文类似，只是在外层查询中将rownum字段裁剪掉即可。

 **说明** 在无rownum的场景中，对于结果表主键的定义需要特别小心。如果定义有误，会直接导致TopN结果的不正确。无rownum场景中，主键应为TopN上游GROUP BY节点的KEY列表。

- 无排名优化示例

本示例来源于自视频行业的案例。用户每个视频在分发时会产生大量流量，依据视频产生的流量可以分析出最热门的视频。本例用于统计出每分钟流量最大的Top5的视频。

○ 测试语句

```
--从SLS读取数据原始存储表。
CREATE TABLE sls_cdnlog_stream (
  vid VARCHAR, -- video id
  rowtime TIMESTAMP, -- 观看视频的时间。
  response_size BIGINT, -- 观看产生的流量。
  WATERMARK FOR rowtime as withOffset(rowtime, 0)
) WITH (
  type='sls',
  ...
);
--1分钟窗口统计vid带宽数。
CREATE VIEW cdnvid_group_view AS
SELECT vid,
  TUMBLE_START(rowtime, INTERVAL '1' MINUTE) AS start_time,
  SUM(response_size) AS rss
FROM sls_cdnlog_stream
GROUP BY vid, TUMBLE(rowtime, INTERVAL '1' MINUTE);
--存储表。
CREATE TABLE hbase_out_cdnvidtoplog (
  vid VARCHAR,
  rss BIGINT,
  start_time VARCHAR,
  -- 注意结果表中不存储rownum字段。
  -- 特别注意该主键的定义，为TopN上游GROUP BY的KEY。
  PRIMARY KEY(start_time, vid)
) WITH (
  type='RDS',
  ...
);
-- 统计每分钟Top5消耗流量的vid，并输出。
INSERT INTO hbase_out_cdnvidtoplog
-- 注意次外层查询，不选出rownum字段。
SELECT vid, rss, start_time FROM
(
  SELECT
  vid, start_time, rss,
  ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY rss DESC) as rownum
FROM
  cdnvid_group_view
)
WHERE rownum <= 5;
```

- 测试数据

vid (VARCHAR)	rowtime (Timestamp)	response_size (BIGINT)
10000	2017-12-18 15:00:10	2000
10000	2017-12-18 15:00:15	4000
10000	2017-12-18 15:00:20	3000
10001	2017-12-18 15:00:20	3000
10002	2017-12-18 15:00:20	4000
10003	2017-12-18 15:00:20	1000
10004	2017-12-18 15:00:30	1000
10005	2017-12-18 15:00:30	5000
10006	2017-12-18 15:00:40	6000
10007	2017-12-18 15:00:50	8000

- 测试结果

start_time (VARCHAR)	vid (VARCHAR)	rss (BIGINT)
2017-12-18 15:00:00	10000	9000
2017-12-18 15:00:00	10007	8000
2017-12-18 15:00:00	10006	6000
2017-12-18 15:00:00	10005	5000
2017-12-18 15:00:00	10002	4000

5.8.10. GROUPING SETS语句

如果您经常需要对数据进行多维度聚合分析（例如既需要按照a列聚合，也要按照b列聚合，同时要按照a和b两列聚合），您可以使用GROUPING SETS语句进行多维度聚合分析，避免多次使用UNION ALL影响性能。

语法格式

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
GROUP BY
[ GROUPING SETS { groupItem [, groupItem ]* } ];
```

示例

- 测试数据


username	month	day
Lily	10	1
Lucy	11	21
Lily	11	21

- 测试案例

```
SELECT
  `month`,
  `day`,
  count(distinct `username`) as uv
FROM tsmall_item
group by
grouping sets((`month`,`month`,`day`));
```

- 测试结果

month	day	uv
10	1	1
10	null	1
11	21	1
11	null	1
11	21	2
11	null	2

 **说明** 此结果为调试结果，会显示出计算过程。如果您的结果表是DataHub、消息队列Kafka或消息队列MQ等，正式上线也会显示过程数据。但如果您的结果表是云数据RDS等关系型数据库，正式上线，主键相同的记录显示为一条数据。

5.8.11. 复杂事件处理（CEP）语句

复杂事件处理（CEP）语句MATCH_RECOGNIZE，用于识别输入流中符合指定规则的事件，并按照指定方式输出。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[MATCH_RECOGNIZE (
[PARTITION BY {partitionItem [, partitionItem]*}]
[ORDER BY {orderItem [, orderItem]*}]
[MEASURES {measureItem AS col [, measureItem AS col]*}]
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]
[AFTER MATCH SKIP]
PATTERN (patternVariable[quantifier] [ patternVariable[quantifier]]*) WITHIN intervalExpression
DEFINE {patternVariable AS patternDefinationExpression [, patternVariable AS patternDefinationExpression ]* }
)];
```

参数	说明
PARTITION BY	分区的列，可选项。
ORDER BY	可以指定多列，但是必须以 <code>EVENT TIME</code> 或 <code>PROCESS TIME</code> 列作为排序的首列，可选项。
MEASURES	定义如何根据匹配成功的输入事件构造输出事件。
ONE ROW PER MATCH	对于每一次成功的匹配，只产生一个输出事件。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除了匹配成功时产生的输出外，超时也会产生输出。超时时间由 <code>PATTERN</code> 语句中的 <code>WITHIN</code> 语句定义。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>? 说明 Blink 3.6.0及以后版本，由于Calcite升级，不再支持 <code>ONE ROW PER MATCH WITH TIMEOUT ROWS</code>参数。</p> </div>
ALL ROWS PER MATCH	对于每一次成功的匹配，对应于每一个输入事件，都会产生一个输出事件。
ALL ROWS PER MATCH WITH TIMEOUT ROWS	除了匹配成功时产生输出外，超时也会产生输出。超时时间由 <code>PATTERN</code> 语句中的 <code>WITHIN</code> 语句定义。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>? 说明 Blink 3.6.0及以后版本，由于Calcite升级，不再支持 <code>ALL ROWS PER MATCH WITH TIMEOUT ROWS</code>参数。</p> </div>
[ONE ROW PER MATCH ALL ROWS PER MATCH ONE ROW PER MATCH WITH TIMEOUT ROWS ALL ROWS PER MATCH WITH TIMEOUT ROWS]	可选项，默认为 <code>ONE ROW PER MATCH</code> 。
AFTER MATCH SKIP TO NEXT ROW	匹配成功后，从匹配成功的事件序列中的第一个事件的下一个事件开始进行下一次匹配。

参数	说明
AFTER MATCH SKIP PAST LAST ROW	匹配成功后，从匹配成功的事件序列中的最后一个事件的下一个事件开始进行下一次匹配。
AFTER MATCH SKIP TO FIRST patternItem	匹配成功后，从匹配成功的事件序列中第一个对应于 patternItem的事件开始进行下一次匹配。
AFTER MATCH SKIP TO LAST patternItem	匹配成功后，从匹配成功的事件序列中最后一个对应于 patternItem的事件开始进行下一次匹配。
PATTERN	<p>定义待识别的事件序列需要满足的规则，需要定义在 <code>()</code> 中，由一系列自定义的 patternVariable 构成。</p> <p>说明</p> <ul style="list-style-type: none"> patternVariable 之间如果以空格间隔，表示符合这两种 patternVariable 的事件中间不存在其他事件。 patternVariable 之间如果以 <code>-></code> 间隔，表示符合这两种 patternVariable 的事件之间可以存在其它事件。 Blink 3.6.0 及以后版本，由于 Calcite 升级，不再支持 PATTERN 参数。

参数解释

- quantifier

quantifier 用于指定符合 patternVariable 定义事件的出现次数。

参数	说明
*	0次或多次
+	1次或多次
?	0次或1次
{n}	n次
{n,}	大于等于n次
{n, m}	大于等于n次，小于等于m次
{,m}	小于等于m次

默认为贪婪匹配。例如，对于 `pattern: A -> B+ -> C`，输入为 `a bc1 bc2 c`（其中 `bc1` 和 `bc2` 表示既匹配 `B` 也匹配 `C`），则输出为 `a bc1 bc2 c`。可以在 quantifier 符号后面加 `?` 来表示非贪婪匹配。例如：

- `*?`

- +?
- {n}?
- {n,}?
- {n, m}?
- {,m}?

说明 Blink 3.x以上版本不支持 (e1 e2+)贪婪匹配，您可以使用 e1 e2+ e3 e3 as not e2 绕行方案，但请务必使用一个e3，才能有数据输出。

此时，对于上面例子中的 PATTERN及输入，产生的输出为 a bc1 bc2,a bc1 bc2 c。

- WITHIN定义符合规则的事件序列的最大时间跨度。
- 静态窗口格式：INTERVAL 'string' timeUnit [TO timeUnit] 。例如：INTERVAL '10' SECOND, INTERVAL '4 5' DAY, INTERVAL '10:20' MINUTE TO SECOND, INTERVAL '10:20.10' MINUTE TO SECOND, INTERVAL '10:20' HOUR TO MINUTE, INTERVAL '1-5' YEAR TO MONTH 。
- 动态窗口格式：INTERVAL intervalExpression 。例如：INTERVAL A.windowTime + 10 ，其中A为PATTERN定义中第一个patternVariable。在intervalExpression的定义中，可以使用PATTERN定义中出现过的patternVariable。当前只能使用第一个patternVariable。intervalExpression中可以使用UDF，intervalExpression的结果必须为LONG，单位为millisecond，表示窗口的大小。
- DEFINE定义在PATTERN中出现的patternVariable的具体含义，如果某个patternVariable在DEFINE中没有定义，则认为对于每一个事件，该patternVariable都成立。

● MEASURES和DEFINE语句函数

函数	说明
Row Pattern Column References	形式为 patternVariable.col 。表示访问patternVariable所对应的事件的指定的列。
PREV	<p>只能用在DEFINE语句中，通常与 Row Pattern Column References 合用。用于访问指定的PATTERN所对应的事件之前，偏移指定的offset所对应的事件的指定的列。</p> <p>例如， DOWN AS DOWN.price < PREV(DOWN.price) ， PREV(A.price) 表示当前事件的前一个事件的 price 列的值。</p> <p>说明</p> <ul style="list-style-type: none"> ○ DOWN.price 等价于 PREV(DOWN.price, 0) 。 ○ PREV(DOWN.price) 等价于 PREV(DOWN.price, 1) 。
FIRST、LAST	<p>一般与 Row Pattern Column References 合用，用于访问指定的PATTERN所对应的事件序列中的指定偏移位置的事件。例如：</p> <ul style="list-style-type: none"> ○ FIRST(A.price, 3) 表示 PATTERN A 所对应的事件序列中的第4个事件。 ○ LAST(A.price, 3) 表示 PATTERN A 所对应的事件序列中的倒数第4个事件。

● 输出列

函数	输出列
ONE ROW PER MATCH	包括 PARTITION BY 中指定的列及 MEASURES 中定义的列。对于 PARTITION BY 中已经指定的列，在 MEASURES 中无需重复定义。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除匹配成功的时候产生输出外，超时的时候也会产生输出，超时时间由PATTERN语句中的WITHIN语句定义。

🔍 说明

- 当定义PATTERN时，最好也定义WITHIN，否则可能会造成STATE越来越大。
- ORDER BY中定义的首列必须为EVENT TIME列或者PROCESS TIME列。

示例

● 示例语法

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
PARTITION BY symbol
ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
LAST(DOWN.tstamp) AS bottom_tstamp,
LAST(UP.tstamp) AS end_tstamp
ONE ROW PER MATCH
AFTER MATCH SKIP TO NEXT ROW
PATTERN (STRT DOWN+ UP+) WITHIN INTERVAL '10' SECOND
DEFINE
DOWN AS DOWN.price < PREV(DOWN.price),
UP AS UP.price > PREV(UP.price)
);
```


● 测试数据

timestamp (TIMESTAMP)	card_id(VARCHAR)	location (VARCHAR)	action (VARCHAR)
2018-04-13 12:00:00	1	Beijing	Consumption
2018-04-13 12:05:00	1	Shanghai	Consumption
2018-04-13 12:10:00	1	Shenzhen	Consumption
2018-04-13 12:20:00	1	Beijing	Consumption

● 测试语法

当相同的card_id在十分钟内，在两个不同的location发生刷卡现象，就会触发报警机制，以便监测信用卡盗刷等现象。

```
CREATE TABLE datahub_stream (
  `timestamp`      TIMESTAMP,
  card_id          VARCHAR,
  location         VARCHAR,
  `action`        VARCHAR,
  WATERMARK wf FOR `timestamp` AS withOffset(`timestamp`, 1000)
) WITH (
  type = 'datahub'
  ...
);
CREATE TABLE rds_out (
  start_timestamp  TIMESTAMP,
  end_timestamp    TIMESTAMP,
  card_id          VARCHAR,
  event           VARCHAR
) WITH (
  type = 'rds'
  ...
);
--定义计算逻辑。
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
from datahub_stream
MATCH_RECOGNIZE (
  PARTITION BY card_id --按card_id分区，将相同卡号的数据分发到同一个计算节点。
  ORDER BY `timestamp` --在窗口内，对事件时间进行排序。
  MEASURES          --定义如何根据匹配成功的输入事件构造输出事件。
    e2.`action` as `event`,
    e1.`timestamp` as `start_timestamp`, --第一次的事件时间为start_timestamp。
    LAST(e2.`timestamp`) as `end_timestamp` --最新的事件时间为end_timestamp。
  ONE ROW PER MATCH --匹配成功输出一条。
  AFTER MATCH SKIP TO NEXT ROW --匹配后跳转到下一行。
  PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE --定义两个事件，e1和e2。
  DEFINE          --定义在PATTERN中出现的patternVariable的具体含义。
    e1 as e1.action = 'Consumption', --事件一的action标记为Consumption。
    e2 as e2.action = 'Consumption' and e2.location <> e1.location --事件二的action标记为Consumption，
    且事件一和事件二的location不一致。
);
```

 **说明** 如果存在满足CEP条件的数据，但没有结果输出是因为只有 Watermark > e2.ts的数据才会被处理，但由于e2后面已经没有数据了，导致Watermark一直是e2.ts -1000，所以e2的数据一直没有被处理，从而导致没有结果输出。

- 测试结果

start_timestamp (TIME STAMP)	end_timestamp (TIME STAMP)	card_id (VARCHAR)	event (VARCHAR)
2018-04-13 12:00:00. 0	2018-04-13 12:05:00. 0	1	Consumption
2018-04-13 12:05:00. 0	2018-04-13 12:10:00. 0	1	Consumption

5.8.12. 去重语句

您可以通过多种方式实现去重需求，例如FIRST_VALUE、LAST_VALUE和DISTINCT等。本文为您介绍如何使用TopN方法实现去重，以及使用过程中的注意事项。

去重的方案通常有两种：

- 保留第一条。
- 保留最后一条。

 说明 ORDER BY后的时间属性字段必须在源表中定义。

语法

由于SQL没有直接去重的语法，因此我们使用SQL的 ROW_NUMBER OVER WINDOW 功能实现去重。ROW_NUMBER OVER WINDOW 与TopN语句方法类似，可以理解为一种特殊的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，行号计算从1开始。
- PARTITION BY col1[, col2..]：指定分区的列，即去重的Key，也可以不指定分区的列。
- ORDER BY timeAttributeCol [asc|desc]：指定排序的列，必须是时间属性字段（Processing Time或Event Time）。可以指定为顺序（Keep First Row）或倒序（Keep Last Row）。
- 外层查询 rownum 必须为 =1 或者 <=1。条件必须是 AND，且不能存在Undeterministic的UDF的条件。

如上语法所示，去重需要两层Query：

- 子查询中：使用 ROW_NUMBER()，按照时间属性列对数据进行排序编号。
- 外层查询中：对排名进行过滤，只取第一条，达到去重的目的。时间列排序方向可以为：
 - 顺序：deduplicate keep first row。
 - 倒序：deduplicate keep last row。

当排序字段是Processing Time列时，Flink会按系统时间去重，其每次运行结果不确定。当排序字段是Event Time列时，Flink会按业务时间去重，其每次运行结果是确定的。

Deduplicate Keep First Row


保留首行的去重策略，即保留指定Key下第一条出现的数据，之后出现在该Key下的数据会被丢弃掉。因为其State中只存储了Key数据，因此性能较优。示例如下。

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

本例中，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在以上示例中是源表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字段简化成PROCTIME()函数进行调用，可以省略proctime字段的声明。

 说明 Blink-3.3.1版本后，FirstRow支持使用Event Time进行开窗，并且不会产生Retraction。

Deduplicate Keep Last Row

 重要 LastRow不支持使用Event Time进行开窗。

LastRow的作用也是去重，且只保留该主键下最后一条出现的数据。其性能略胜于LAST_VALUE函数，示例如下。

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

FAQ

Q: 执行 ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY now() as time DESC) 语句时，产生如下报错，应该如何处理？

```
java.lang.RuntimeException: Can not retract a non-existent record:
 38c30001,1b800000008,1c000000013,85000035343a3731,5d304013.
This should never happen.
```

A: 通常有以下两种原因，您可以按照以下方式进行处理：

- 问题原因：由代码中 now() 导致。因为TopN不支持非确定性的排序字段，now() 每次输出的值不同，所以导致Retraction会找不到之前的值。

解决方法：Event Time或源表中一个具有Processing Time属性的字段。

- 问题原因： `blink.state.ttl.ms` 或 `state.backend.niagara.ttl.ms` 参数值设置过小。

解决方法：设置过小的TTL参数使用默认配置，或调大参数值。

5.9. 窗口函数

5.9.1. 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

窗口函数Flink SQL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的网页，可以通过定义一个窗口来收集最近1分钟内的数据，并对这个窗口内的数据进行计算。

Flink SQL支持的窗口聚合主要是两种：Window聚合和Over聚合。本文档主要为您介绍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）。

时间属性

Flink SQL支持以下两种时间属性。实时计算可以基于这两种时间属性对数据进行窗口聚合。

- **Event Time**：您提供的事件时间（通常是数据的最原始的创建时间），Event Time一定是您提供在Schema里的数据。
- **Processing Time**：对事件进行处理的本地系统时间。

 说明 实时计算时间属性详情，请参见 [时间属性](#)。

级联窗口

Rowtime列在经过窗口操作后，其Event Time属性将丢失。您可以使用辅助函数

`TUMBLE_ROWTIME`、`HOP_ROWTIME` 或 `SESSION_ROWTIME`，获取窗口中的Rowtime列的最大值 `max(rowtime)` 作为时间窗口的Rowtime，其类型是具有Rowtime属性的TIMESTAMP，取值为 `window_end - 1`。例如 `[00:00, 00:15)` 的窗口，返回值为 `00:14:59.999`。

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 TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000) --为Rowtime定义Watermark。
) with (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);
CREATE VIEW one_minute_window_output as
SELECT
  // 使用TUMBLE_ROWTIME作为二级Window的聚合时间。
  TUMBLE_ROWTIME(ts, INTERVAL '1' MINUTE) as rowtime,
  username,
  COUNT(click_url) as cnt
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
INSERT INTO tumble_output
SELECT
  TUMBLE_START(rowtime, INTERVAL '1' HOUR),
  TUMBLE_END(rowtime, INTERVAL '1' HOUR),
  username,
  SUM(cnt)
FROM one_minute_window_output
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), username;
```

5.9.2. 滚动窗口

本文为您介绍如何使用实时计算Flink版滚动窗口函数。

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。

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和Watermark。

标识函数

使用标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
<code>TUMBLE START(time-attr, size-interval)</code>	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。
<code>TUMBLE END(time-attr, size-interval)</code>	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。
<code>TUMBLE ROWTIME(time-attr, size-interval)</code>	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。
<code>TUMBLE PROCTIME(time-attr, size-interval)</code>	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)	ts (TIMESTAMP)
Jark	<code>http://taobao.com/xxx</code>	2017-10-10 10:00:00.0
Jark	<code>http://taobao.com/xxx</code>	2017-10-10 10:00:10.0
Jark	<code>http://taobao.com/xxx</code>	2017-10-10 10:00:49.0
Jark	<code>http://taobao.com/xxx</code>	2017-10-10 10:01:05.0
Jark	<code>http://taobao.com/xxx</code>	2017-10-10 10:01:58.0
Timo	<code>http://taobao.com/xxx</code>	2017-10-10 10:02:10.0

- 测试语句

```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000) --为rowtime定义Watermark。
) WITH (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) WITH (
  type='RDS'
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE) 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
Jark	http://taobao.com/xxx

username (VARCHAR)	click_url (VARCHAR)
Jark	http://taobao.com/xxx
Timo	http://taobao.com/xxx

● 测试语句

```
CREATE TABLE window_test (  
  username VARCHAR,  
  click_url VARCHAR,  
  ts as PROCTIME()  
) WITH (  
  type='datahub',  
  ...  
);  
CREATE TABLE tumble_output(  
  window_start TIMESTAMP,  
  window_end TIMESTAMP,  
  username VARCHAR,  
  clicks BIGINT  
) WITH (  
  type='print'  
);  
INSERT INTO tumble_output  
SELECT  
  TUMBLE_START(ts, INTERVAL '1' MINUTE),  
  TUMBLE_END(ts, INTERVAL '1' MINUTE),  
  username,  
  COUNT(click_url)  
FROM window_test  
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

● 测试结果

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Jark	5
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Timo	1

说明 因为本地调试是瞬时的，处理时间可能小于1秒，所以使用Processing Time时间属性对数据进行窗口聚合，可能会出现本地调试没有结果的情况。

5.9.3. 滑动窗口

本文为您介绍如何使用实时计算滑动窗口函数。

说明 实时计算滑动窗口（HOP）暂不支持与LAST_VALUE、FIRST_VALUE或TopN函数共同使用。

什么是滑动窗口

滑动窗口（HOP），也被称作Sliding Window。不同于滚动窗口，滑动窗口的窗口可以重叠。

滑动窗口有两个参数：slide和size。slide为每次滑动的步长，size为窗口的大小。

- slide < size，则窗口会重叠，每个元素会被分配到多个窗口。
- slide = size，则等同于滚动窗口（TUMBLE）。
- slide > size，则为跳跃窗口，窗口之间不重叠且有间隙。

通常，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。

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和Watermark。

滑动窗口标识函数

使用滑动窗口标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
HOP START (<time-attr>, <slide-interval>, <size-interval>)	TIMESTAMP	返回窗口的起始时间（包含边界）。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。

窗口标识函数	返回类型	描述
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 TABLE user_clicks (
  username VARCHAR,
  click_url VARCHAR,
  ts TIMESTAMP,
  WATERMARK wk FOR ts AS WITHOFFSET (ts, 2000)--为rowtime定义Watermark。
) WITH ( TYPE = 'datahub',
  ...);
CREATE TABLE hop_output (
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) WITH (TYPE = 'rds',
  ...);
INSERT INTO
  hop_output
SELECT
  HOP_START (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
  HOP_END (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
  username,
  COUNT (click_url)
FROM
  user_clicks
GROUP BY
  HOP (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
  username
```

• 测试结果

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 09:59:30.0	2017-10-10 10:00:30.0	Jark	2
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	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
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

5.9.4. 会话窗口

本文为您介绍如何使用实时计算Flink版会话窗口函数。

什么是会话窗口

会话窗口 (SESSION) 通过SESSION活动来对元素进行分组。会话窗口与滚动窗口和滑动窗口相比，没有窗口重叠，没有固定窗口大小。相反，当它在一个固定的时间周期内不再收到元素，即会话断开时，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和Watermark。

会话窗口标识函数

使用标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
SESSION START (<time-attr>, <gap-interval>)	Timestamp	返回窗口的起始时间（包含边界）。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。
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>。

窗口标识函数	返回类型	描述
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，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，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 TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000) -- 为rowtime定义watermark
) WITH (
  type='datahub',
  ...
);
CREATE TABLE session_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) WITH (
  type='rds',
  ...
);
INSERT INTO session_output
SELECT
  SESSION_START(ts, INTERVAL '30' SECOND),
  SESSION_END(ts, INTERVAL '30' SECOND),
  username,
  COUNT(click_url)
FROM user_clicks
GROUP BY SESSION(ts, INTERVAL '30' SECOND), username;

```

- 测试结果

window_start (TIMESTAMP)	window_end (TIMESTAMP)	username (VARCHAR)	clicks (BIGINT)
2017-10-10 10:00:00.0	2017-10-10 10:00:40.0	Jark	2
2017-10-10 10:00:49.0	2017-10-10 10:01:35.0	Jark	2
2017-10-10 10:01:58.0	2017-10-10 10:02:28.0	Jark	1
2017-10-10 10:02:10.0	2017-10-10 10:02:40.0	Timo	1

5.9.5. 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。详情请参见[Over Aggregation](#)。

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。
- 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

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 TABLE tsmall_item(  
  itemID VARCHAR,  
  itemType VARCHAR,  
  onSellTime TIMESTAMP,  
  price DOUBLE,  
  WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)  
) WITH (  
  type = 'sls',  
  ...  
);  
SELECT  
  itemID,  
  itemType,  
  onSellTime,  
  price,  
  MAX(price) OVER (  
    PARTITION BY itemType  
    ORDER BY onSellTime  
    ROWS BETWEEN 2 preceding AND CURRENT ROW) AS maxPrice  
FROM tsmall_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。

• 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

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 TABLE tmall_item(
  itemID VARCHAR,
  itemType VARCHAR,
  onSellTime TIMESTAMP,
  price DOUBLE,
  WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
  ...
);
SELECT
  itemID,
  itemType,
  onSellTime,
  price,
  MAX(price) OVER (
    PARTITION BY itemType
    ORDER BY onSellTime
    RANGE BETWEEN INTERVAL '2' MINUTE preceding AND CURRENT ROW) AS maxPrice
FROM tmall_item;
```

- 测试结果

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	40
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

5.10. 内置函数

5.10.1. 字符串函数

5.10.1.1. CHAR_LENGTH

本文为您介绍如何使用实时计算字符串函数CHAR_LENGTH。

语法

```
CHAR_LENGTH(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回字符串中的字符的数量。

示例

- 测试数据

var1(INT)
ss
231ee

- 测试语句

```
SELECT CHAR_LENGTH(var1) as aa  
FROM T1;
```

- 测试结果

aa(INT)
2
5

5.10.1.2. CHR

本文为您介绍如何使用实时计算字符串函数CHR。

语法

```
VARCHAR CHR(INT ascii)
```

入参

参数	数据类型	说明
ascii	INT	是0到255之间的整数。如果不在此范围内，则返回NULL。

功能描述

将ASCII码转换为字符。

示例

• 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

• 测试语句

```
SELECT CHR(int1) as var1, CHR(int2) as var2, CHR(int3) as var3  
FROM T1;
```

• 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

5.10.1.3. CONCAT

本文为您介绍如何使用实时计算字符串函数CONCAT。

语法

```
VARCHAR CONCAT(VARCHAR var1, VARCHAR var2, ...)
```

入参

参数	数据类型	说明
var1	VARCHAR	普通字符串值
var2	VARCHAR	普通字符串值

功能描述

连接两个或多个字符串值从而组成一个新的字符串。如果任一参数为NULL时，则跳过该参数。

示例

- 测试数据

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

- 测试语句

```
SELECT CONCAT(var1, var2, var3) as var
FROM T1;
```

- 测试结果

var(VARCHAR)
HelloMyWorld
HelloWorld
World
N/A

5.10.1.4. CONCAT_WS

本文为您介绍如何使用实时计算字符串函数CONCAT_WS。

语法

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```

入参

参数	数据类型	说明
separator	VARCHAR	分隔符
var1	VARCHAR	-
var2	VARCHAR	-

功能描述

将每个参数值和第一个参数separator指定的分隔符依次连接到一起组成新的字符串，长度和类型取决于输入值。

说明 如果separator取值为null，则将separator视作与空串进行拼接。如果其它参数为NULL，在执行拼接过程中跳过取值为NULL的参数。

示例

测试数据

sep(VARCHAR)	str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)
	Jack	Harry	John
null	Jack	Harry	John
	null	Harry	John
	Jack	null	null

测试语句

```
SELECT CONCAT_WS(sep, str1, str2, str3) as var FROM T1;
```

测试结果

var(VARCHAR)
Jack Harry John
JackHarryJohn
Harry John
Jack

5.10.1.5. FROM_BASE64

本文为您介绍如何使用实时计算字符串函数FROM_BASE64。

语法

```
BINARY FROM_BASE64(str)
```

入参

参数	数据类型	说明
str	VARCHAR	base64编码的字符串。

功能描述

将base64编码的字符串str解析成对应的binary类型数据输出。

示例

- 测试数据

a(INT)	b(BIGINT)	c(VARCHAR)
1	1L	null

- 测试语句

```
SELECT  
from_base64(c) as var1,from_base64('SGVsbG8gd29ybGQ=') as var2  
FROM T1;
```

- 测试结果

var1(BINARY)	var2(BINARY)
null	Byte Array: [72('H'), 101('e'), 108('l'), 108('l'), 111('o'), 32(' '), 119('w'), 111('o'), 114('r'), 108('l'), 100('d')]

5.10.1.6. HASH_CODE

本文为您介绍如何使用实时计算字符串函数HASH_CODE。

语法

```
INT HASH_CODE(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

返回字符串的HASH_CODE()的绝对值。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

- 测试语句

```
SELECT HASH_CODE(str1) as var1, HASH_CODE(str2) as var2, HASH_CODE(nullstr) as var3
FROM T1;
```

- 测试结果

var1(INT)	var2(INT)	var3(INT)
1099348823	401392878	null

5.10.1.7. INITCAP

本文为您介绍如何使用实时计算字符串函数INITCAP。

语法

```
VARCHAR INITCAP(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回字符串，每个字转换器的第一个字母大写，其余为小写。

示例

- 测试数据

var1(VARCHAR)
aADvbn

- 测试语句

```
SELECT INITCAP(var1)as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
Aadvbn

5.10.1.8. INSTR

本文为您介绍如何使用实时计算Flink版字符串函数INSTR。

 **重要** 仅Blink 2.2.0及以上版本支持INSTR函数。

语法

```
INT instr( string1, string2 )  
INT instr( string1, string2 [, start_position [, nth_appearance ] ] )
```

入参

参数	数据类型	说明
string1	VARCHAR	源字符串，要在该字符串中查找string2。
string2	VARCHAR	目标字符串，string1中查找的字符串。
start_position	INT	起始位置，表示在string1中开始查找的其实位置： <ul style="list-style-type: none">该参数省略（默认）：字符串索引从1开始。该参数为正：从左到右开始检索。该参数为负：从右到左开始检索。
nth_appearance	INT	匹配序号代表要查找第几次出现的string2： <ul style="list-style-type: none">该参数省略（默认）：第1次出现。该参数为负：系统报错。

功能描述

返回目标字符串在源字符串中的位置，如果在源字符串中未找到目标字符串，则返回0。

示例

测试数据

string1(VARCHAR)
helloworld

测试语句

```
SELECT  
instr('helloworld','lo') as res1,  
instr('helloworld','l',-1,1) as res2,  
instr('helloworld','l',3,2) as res3  
FROM T1;
```

测试结果

res1(INT)	res2(INT)	res3(INT)
4	9	4

5.10.1.9. JSON_VALUE

本文为您介绍如何使用实时计算字符串函数JSON_VALUE。

语法

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

入参

- content

VARCHAR类型，需要解析的JSON对象，使用字符串表示。

- path

VARCHAR类型，解析JSON的路径表达式。目前path支持如下表达式。

符号	功能
\$	根对象
[]	数组下标
*	数组通配符
.	取子元素

功能描述

从JSON字符串中提取指定path的值，不合法的JSON和null都统一返回null。

示例

- 测试数据

id(INT)	json(VARCHAR)	path1(VARCHAR)
1	[10, 20, [30, 40]]	\$\$[2][*]
2	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"],"iii":"jjj"}}	\$.ccc.hhh[*]
3	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"],"iii":"jjj"}}	\$.ccc.hhh[1]
4	[10, 20, [30, 40]]	NULL
5	NULL	\$\$[2][*]
6	"{xx}"	"\$\$[2][*]"

- 测试语句

```
SELECT
  id,
  JSON_VALUE(json, path1) AS `value`
FROM
  T1;
```

- 测试结果

id (INT)	value (VARCHAR)
1	[30,40]
2	["h0","h1","h2"]
3	h1
4	NULL
5	NULL
6	NULL

5.10.1.10. KEYVALUE

本文为您介绍如何使用实时计算字符串函数KEYVALUE。

语法

```
VARCHAR KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
```

入参

参数	数据类型	说明
str	VARCHAR	字符串中的key-value (kv) 对。
split1	VARCHAR	kv对的分隔符。
split2	VARCHAR	kv的分隔符。
key_name	VARCHAR	键的名称

功能描述

解析str字符串中，匹配有split1（kv对的分隔符）和split2（kv的分隔符）的key-value对，根据key_name返回对应的数值。如果key_name值不存在或异常时，返回NULL。

示例

- 测试数据

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)
k1=v1;k2=v2	;	=	k2
null	;		:
k1:v1 k2:v2	null	=	:
k1:v1 k2:v2		=	null
k1:v1 k2:v2		=	:

- 测试语句

```
SELECT KEYVALUE(str, split1, split2, key1) as `result`
FROM T1;
```

- 测试结果

result(VARCHAR)
v2
null
null
null
null

5.10.1.11. LOWER

本文为您介绍如何使用实时计算字符串函数LOWER。

语法

```
VARCHAR LOWER(A)
```

入参

- A
- VARCHAR类型。

功能描述

返回转换为小写字母的字符串。

示例

- 测试数据

var1(VARCHAR)
Ss
yyT

- 测试语句

```
SELECT LOWER(var1) as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
ss
yyt

5.10.1.12. LPAD

本文为您介绍如何使用实时计算字符串函数LPAD。

语法

```
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
```

入参

参数	数据类型	说明
str	VARCHAR	启始的字符串。
len	INT	新的字符串的长度。
pad	VARCHAR	需要重复补充的字符串。

功能描述

字符串str左端填充若干个字符串pad，直到新的字符串达到指定长度len为止。

任意参数为null时返回null。

len 为负数时返回为null。

pad 为空串时，如果 len 不大于 str 长度，返回 str 裁剪后的结果。如果 len 大于 str 长度时，则返回null。

示例

- 测试数据

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	-2	空
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	空
空	2	s
asd	4	空
空	0	空

• 测试语句

```
SELECT LPAD(str, len, pad) AS result
FROM T1;
```

• 测试结果

result(VARCHAR)
null
JohnJHelloWorld
Jo
HelC
null
null
as
ss
null
空

5.10.1.13. MD5

本文为您介绍如何使用实时计算字符串函数MD5。

语法

```
VARCHAR MD5(VARCHAR str)
```

入参

- str
- VARCHAR类型

功能描述

返回字符串的MD5值。如果参数为空串（即参数为"）时，则返回空串。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)
k1=v1;k2=v2	空

- 测试语句

```
SELECT  
  MD5(str1) as var1,  
  MD5(str2) as var2  
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	空

5.10.1.14. OVERLAY

本文为您介绍如何使用实时计算字符串函数OVERLAY。

语法

```
VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position [ FOR INT length ] ) )
```

入参

参数	数据类型
x	VARCHAR
y	VARCHAR

参数	数据类型
start_position	INT
length（可选）	INT

功能描述

用y替换x的子串。从start_position开始，替换length+1个字符。

示例

测试语句

```
OVERLAY('abcdefg' PLACING 'hij' FROM 2 FOR 2) as result
FROM T1;
```

测试结果

result(VARCHAR)
ahijdefg

5.10.1.15. PARSE_URL

本文为您介绍如何使用实时计算字符串函数PARSE_URL。

语法

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key])
```

入参

参数	数据类型	说明
urlStr	VARCHAR	链接
partToExtract	VARCHAR	指定链接中解析的部分。取值如下： <ul style="list-style-type: none"> • HOST • PATH • QUERY • REF • PROTOCOL • FILE • AUTHORITY • USERINFO
key	VARCHAR	可选，指定截取部分中，具体的键。

功能描述

返回urlStr中指定的部分解析后的值。如果urlStr参数值为null时，则返回值为null。

示例

- 测试数据

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

- 测试语句

```
SELECT PARSE_URL(url1, 'QUERY', 'query') as var1,
    PARSE_URL(url1, 'QUERY') as var2,
    PARSE_URL(url1, 'HOST') as var3,
    PARSE_URL(url1, 'PATH') as var4,
    PARSE_URL(url1, 'REF') as var5,
    PARSE_URL(url1, 'PROTOCOL') as var6,
    PARSE_URL(url1, 'FILE') as var7,
    PARSE_URL(url1, 'AUTHORITY') as var8,
    PARSE_URL(nullstr, 'QUERY') as var9,
    PARSE_URL(url1, 'USERINFO') as var10,
    PARSE_URL(nullstr, 'QUERY', 'query') as var11
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)	var11(VARCHAR)
1	query=1	facebook.com	/path/p1.php	null	http	/path/p1.php?query=1	facebook.com	null	null	null

5.10.1.16. POSITION

本文为您介绍如何使用实时计算字符串函数POSITION。

语法

```
INTEGER POSITION(x IN y)
```

入参

参数	数据类型
x	VARCHAR
y	VARCHAR

功能描述

返回目标字符串x在被查询字符串y里第一次出现的位置。如果目标字符串x在被查询字符串y中不存在，返回值为0。

示例

- 测试语句

```
POSITION('in' IN 'china') as result  
FROM T1;
```

- 测试结果

result(INT)
3

5.10.1.17. REGEXP

本文为您介绍如何使用实时计算字符串函数REGEXP。

语法

```
BOOLEAN REGEXP(VARCHAR str, VARCHAR pattern)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	指定的匹配模式。

功能描述

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。如果str或者pattern为空或为NULL时，则返回false。

示例

- 测试数据

str1(VARCHAR)	pattern1(VARCHAR)
k1=v1;k2=v2	k2*
k1:v1 k2:v2	k3
null	k3
k1:v1 k2:v2	null
k1:v1 k2:v2	(

- 测试语句

```
SELECT REGEXP(str1, pattern1) AS result
FROM T1;
```

- 测试结果

result(BOOLEAN)
true
false
false
false
false

5.10.1.18. REGEXP_EXTRACT

本文为您介绍如何使用实时计算字符串函数REGEXP_EXTRACT。

语法

```
VARCHAR REGEXP_EXTRACT(VARCHAR str, VARCHAR pattern, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	匹配的字符串。
index	INT	第几个被匹配的字符串。

重要 正则常量请按照Java代码来写。CodeGen会将SQL常量字符串自动转化为Java代码。如果要描述一个数字\`d`，需要写成\`'d'`，即和Java中正则相同。

功能描述

使用正则模式Pattern匹配抽取字符串Str中的第Index个子串，Index从1开始，正则匹配提取。当参数为NULL或者正则不合法时，则返回NULL。

示例

- 测试数据

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	foo(.?)(bar)	2
100-200	(\\d+)-(\\d+)	1
null	foo(.?)(bar)	2
foothebar	null	2
foothebar	空	2
foothebar	(2

- 测试语句

```
SELECT REGEXP_EXTRACT(str1, pattern1, index1) as result
FROM T1;
```

- 测试结果

result(VARCHAR)
bar
100
null
null
null
null

5.10.1.19. REGEXP_REPLACE

本文为您介绍如何使用实时计算字符串函数REGEXP_REPLACE。

语法

VARCHAR REGEXP_REPLACE(VARCHAR str, VARCHAR pattern, VARCHAR replacement)

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	被替换的字符串。
replacement	VARCHAR	用于替换的字符串。

 **重要** 请您按照Java代码编写正则常量。Codegen会自动将SQL常量字符串转化为Java代码。描述一个数值 (\d) 的正则表达式和Java中一样，为 'd'。

功能描述

用字符串 replacement 替换字符串 str 中正则模式为 pattern 的部分，并返回新的字符串。如果参数为 NULL或者正则不合法时，则返回NULL。

如果您的业务数据中存在特殊字符（即系统使用的字符或者SQL关键字），则需要对特殊字符进行转义处理。常见的特殊字符和转义方式如下表所示。

特殊字符	字符的转义方式
'	\'
\	\\
.	\.
\$	\\\$

示例

• 测试数据

str1(VARCHAR)	pattern1(VARCHAR)	replace1(VARCHAR)
2014-03-13	-	空
NULL	-	空
2014-03-13	-	NULL
2014-03-13	空	s
2014-03-13	(s
100-200	(\d+)	num

- 测试语句

```
SELECT REGEXP_REPLACE(str1, pattern1, replace1) as result
FROM T1;
```

- 测试结果

result(VARCHAR)
20140313
null
null
2014-03-13
null
num-num

5.10.1.20. REPEAT

本文为您介绍如何使用实时计算字符串函数REPEAT。

语法

```
VARCHAR REPEAT(VARCHAR str, INT n)
```

入参

参数	数据类型	说明
str	VARCHAR	重复字符串值。
n	INT	重复次数。

功能描述

返回以字符串值为str，重复次数为N的新的字符串。如果参数为null时，则返回null。如果重复次数为0或负数，则返回空串。

示例

- 测试数据

str(VARCHAR)	n(INT)
J	9
Hello	2

str(VARCHAR)	n(INT)
Hello	-9
null	9

- 测试语句

```
SELECT REPEAT(str,n) as var1  
FROM T1;
```

- 测试结果

var1(VARCHAR)
JJJJJJJJ
HelloHello
空
null

5.10.1.21. REPLACE

本文为您介绍如何使用实时计算字符串函数REPLACE。

语法

```
VARCHAR REPLACE(str1, str2, str3)
```

入参

参数	数据类型	说明
str1	VARCHAR	原字符
str2	VARCHAR	目标字符
str3	VARCHAR	替换字符

功能描述

字符串替换函数。

示例

- 测试数据

str1(INT)	str2(INT)	str3(INT)
alibaba blink	blink	flink

- 测试语句

```
SELECT REPLACE(str1, str2, str3) as `result`  
FROM T1;
```

- 测试结果

result(VARCHAR)
alibaba flink

5.10.1.22. REVERSE

本文为您介绍如何使用实时计算字符串函数REVERSE。

语法

```
VARCHAR REVERSE(VARCHAR str)
```

入参

参数	数据类型	说明
str	VARCHAR	普通字符串值。

功能描述

反转字符串，返回字符串值的相反顺序。如果任一参数为null时，则返回null。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)	str4(VARCHAR)
iPhoneX	Alibaba	World	null

- 测试语句

```
SELECT REVERSE(str1) as var1,REVERSE(str2) as var2,  
       REVERSE(str3) as var3,REVERSE(str4) as var4  
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

5.10.1.23. RPAD

本文为您介绍如何使用实时计算字符串函数RPAD。

语法

```
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
```

入参

参数	数据类型	说明
str	VARCHAR	起始的字符串。
len	INT	新的字符串的长度。
pad	VARCHAR	需要重复补充的字符串。

功能描述

字符串str右端填充若干个字符串pad，直到新的字符串达到指定长度 len 为止。

- 如果任意参数为null时，则返回null。
- 如果 len 长度为负数时，则返回null。
- 当 pad 为空串，如果 len 不大于 str 长度时，则返回 str 裁剪后的结果。
- 如果 len 大于 str 长度，则返回null。

示例

- 测试数据

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	-2	空
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	空

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	2	s
asd	4	空
空	0	空

- 测试语句

```
SELECT RPAD(str, len, pad) as result
FROM T1;
```

- 测试结果

result(VARCHAR)
null
HelloWorldJohnj
Jo
CHel
null
null
as
ss
null
空

5.10.1.24. SPLIT_INDEX

本文为您介绍如何使用实时计算字符串函数SPLIT_INDEX。

语法

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	被分隔的字符串。

参数	数据类型	说明
sep	VARCHAR	分隔符的字符串。
index	INT	截取的字段的位。

功能描述

以 `sep` 作为分隔符，将字符串 `str` 分隔成若干段，取其中的第 `index` 段。`index` 从0开始，如果取不到字段，则返回null。如果任一参数为NULL，则返回null。

示例

测试数据

str(VARCHAR)	sep(VARCHAR)	index(INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0
null	,	0

测试语句

```
SELECT SPLIT_INDEX(str, sep, index) as var1
FROM T1;
```

测试结果

var1(VARCHAR)
Mary
null
null
null

5.10.1.25. STR_TO_MAP

本文为您介绍如何使用实时计算字符串函数STR_TO_MAP。

语法

```
MAP STR_TO_MAP(VARCHAR text)
MAP STR_TO_MAP(VARCHAR text, VARCHAR listDelimiter, VARCHAR keyValueDelimiter)
```

功能描述

使用listDelimiter将text分隔成K-V对，然后使用keyValueDelimiter分隔每个K-V对，组装成MAP返回。默认listDelimiter为(,)，keyValueDelimiter为(=)。

入参

参数	数据类型	说明
text	VARCHAR	输入文本。
listDelimiter	VARCHAR	用来将text分隔成K-V对。默认为(,)。
keyValueDelimiter	VARCHAR	用来分隔每个key和value。默认为(=)。

 **重要** 这里的Delimiter使用的是Java的正则表达式，遇到特殊字符需要转义。

测试语句

```
SELECT
  STR_TO_MAP('k1=v1,k2=v2')['k1'] as a
FROM T1;
```

测试结果

a(VARCHAR)
v1

5.10.1.26. SUBSTRING

本文为您介绍如何使用实时计算字符串函数SUBSTRING。

语法

```
VARCHAR SUBSTRING(VARCHAR a, INT start)
VARCHAR SUBSTRING(VARCHAR a, INT start, INT len)
```

入参

参数	数据类型	说明
a	VARCHAR	指定的字符串。
start	INT	在字符串a中开始截取的位置。

参数	数据类型	说明
len	INT	类截取的长度。

功能描述

获取字符串子串。截取从位置start开始，长度为len的子串。如果未指定len，则截取到字符串结尾。start从1开始，start为0当1看待，为负数时表示从字符串末尾倒序计算位置。

示例

测试数据

str(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	null

测试语句

```
SELECT SUBSTRING(" 222222222) as var1,
SUBSTRING(str, 2) as var2,
SUBSTRING(str, -2) as var3,
SUBSTRING(str, -2, 1) as var4,
SUBSTRING(str, 2, 1) as var5,
SUBSTRING(str, 22) as var6,
SUBSTRING(str, -22) as var7,
SUBSTRING(str, 1) as var8,
SUBSTRING(str, 0) as var9,
SUBSTRING(nullstr, 0) as var10
FROM T1;
```

测试结果

var1(V ARCHA R)	var2(V ARCHA R)	var3(V ARCHA R)	var4(V ARCHA R)	var5(V ARCHA R)	var6(V ARCHA R)	var7(V ARCHA R)	var8(V ARCHA R)	var9(V ARCHA R)	var10(V ARCHA R)
空	1=v1;k 2=v2	v2	v	1	空	空	k1=v1; k2=v2	k1=v1; k2=v2	null

5.10.1.27. TO_BASE64

本文为您介绍如何使用实时计算字符串函数TO_BASE64。

语法

```
VARCHAR TO_BASE64(bin)
```

入参

参数	数据类型
bin	BINARY

功能描述

将BINARY类型数据转换成对应base64编码的字符串输出。

示例

- 测试数据

c(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

- 测试语句

```
SELECT TO_BASE64(FROM_BASE64(c)) as var1  
FROM T1;
```

- 测试结果

var1(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

5.10.1.28. TRIM

本文为您介绍如何使用实时计算字符串函数TRIM。

语法

```
VARCHAR TRIM( VARCHAR x )
```

入参

参数	数据类型
x	VARCHAR

功能描述

除掉一个字串中的字头或字尾。最常见的用途是移除字首或字尾的空格。

示例

- 测试语句

```
SELECT TRIM(' Sample ') as result  
FROM T1;
```

- 测试结果

result(VARCHAR)
Sample

5.10.1.29. UPPER

本文为您介绍如何使用实时计算字符串函数UPPER。

语法

```
VARCHAR UPPER(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回转换为大写字符的字符串。

示例

- 测试数据

var1(VARCHAR)
ss
ttee

- 测试语句

```
SELECT UPPER(var1) as aa  
FROM T1;
```

- 测试结果

aa(VARCHAR)
SS
TTEE

5.10.2. 数学函数

5.10.2.1. 加

本文为您介绍如何使用实时计算数学函数加法。

语法

```
A + B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A加B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	20	30

- 测试语句

```
SELECT int1+int2+int3 as aa  
FROM T1;
```

- 测试结果

aa(int)
60

5.10.2.2. 减

本文为您介绍如何使用实时计算数学函数减法。

语法

```
A - B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A减B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	10	30

- 测试语句

```
SELECT int3 - int2 - int1 as aa  
FROM T1;
```

- 测试结果

aa(int)
10

5.10.2.3. 乘

本文为您介绍如何使用实时计算数学函数乘法。

语法

```
A * B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A乘以B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	20	3

- 测试语句

```
SELECT int1*int2*int3 as aa  
FROM T1;
```

- 测试结果

aa(int)
600

5.10.2.4. 除

本文为您介绍如何使用实时计算数学函数除法。

语法

```
A / B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A除以B的结果。

示例

- 测试数据

int1(INT)	int2(INT)
8	4

- 测试语句

```
SELECT int1 / int2 as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
2.0

5.10.2.5. ABS

本文为您介绍如何使用实时计算数学函数ABS。

语法

```
DOUBLE ABS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的绝对值。

示例

- 测试数据

in1(DOUBLE)
4.3

- 测试语句

```
SELECT ABS(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
4.3

5.10.2.6. ACOS

本文为您介绍如何使用实时计算数学函数ACOS。

语法

```
ACOS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反余弦值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ACOS(in1) as aa  
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.7707963267948966
1.1592794807274085

5.10.2.7. BIN

本文为您介绍如何使用实时计算数学函数BIN。

语法

```
VARCHAR BIN(BIGINT number)
```

入参

参数	数据类型
----	------

参数	数据类型
number	BIGINT  说明 参数的隐式转换支持INT，不支持其他类型。


功能描述

将长整型参数转换为二进制形式，返回类型为字符串。

示例

- 测试数据

id(INT)	x(BIGINT)
1	12L
2	10L
3	0L
4	1000000000L

 **说明** 测试数据x列中的字母 **L** 代表的是数据类型 **Long**，并不是做二进制转换的数值的一部分。

- 测试语句

```
SELECT id, bin(x) as var1  
FROM T1;
```

- 测试结果

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	1001010100000010111110010000000000

5.10.2.8. ASIN

本文为您介绍如何使用实时计算数学函数ASIN。

语法

DOUBLE ASIN(A)

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反正弦值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ASIN(in1) as aa  
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.8
0.41151684606748806

5.10.2.9. ATAN

本文为您介绍如何使用实时计算数学函数ATAN。

语法

DOUBLE ATAN(A)

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反正切值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ATAN(in1) as aa  
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.6222796222326533
0.3805063771123649

5.10.2.10. BITAND

本文为您介绍如何使用实时计算数学函数BITAND。

语法

```
INT BITAND(INT number1,INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

运算符按位“与”操作。输入和输出类型均为INT整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITAND(a, b) as intt
FROM T1;
```

- 测试结果

intt(INT)
2

5.10.2.11. BITNOT

本文为您介绍如何使用实时计算数学函数BITNOT。

语法

```
INT BITNOT(INT number)
```

入参

参数	数据类型
number	INT

功能描述

按位取反。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)
7

- 测试语句

```
SELECT BITNOT(a) as var1
FROM T1;
```

- 测试结果

var1(INT)
0xfff8

5.10.2.12. BITOR

本文为您介绍如何使用实时计算数学函数BITOR。

语法

```
INT BITOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

按位取“或”。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITOR(a, b) as var1  
FROM T1;
```

- 测试结果

var1(INT)
3

5.10.2.13. BITXOR

本文为您介绍如何使用实时计算数学函数BITXOR。

语法

```
INT BITXOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

按位取“异或”。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITXOR(a, b) as var1  
FROM T1;
```

- 测试结果

var1(INT)
1

5.10.2.14. CARDINALITY

本文为您介绍如何使用实时计算数学函数CARDINALITY。

语法

```
CARDINALITY(str)
```

入参

参数	数据类型
number1	数组

功能描述

返回一个集合中的元素数量。

示例

- 测试语句

```
SELECT cardinality(array[1,2,3]) AS `result`  
FROM T1;
```

- 测试结果

result(INT)
3

5.10.2.15. CONV

本文为您介绍如何使用实时计算进制转换内置函数CONV。

 说明 blink-3.2.2及以上版本支持该函数。

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
number	BIGINT、VARCHAR。
FROM_BASE	INT，非负数，取值范围[2, 36]。
TO_BASE	INT，可以为正数（无符号整数）、负数（有符号整数）、ABS(TO_BASE)，取值范围[2, 36]。

功能描述

将长整型数字或字符串形式数字从一种进制转换为另一种进制，返回类型为字符串，CONV()精度为64位。

 说明 当number是null或非法字符时，结果返回为NULL。

样例

- 测试数据

id(INT)	x(BIGINT)	y (VARCHAR)
1	12L	'12'
2	10L	'10'
3	0L	'test'
4	NULL	NULL

- 测试案例

```
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	O	NULL
4	NULL	NULL

5.10.2.16. COS

本文为您介绍如何使用实时计算数学函数COS。

语法

```
DOUBLE COS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的余弦值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT COS(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

5.10.2.17. COT

本文为您介绍如何使用实时计算数学函数COT。

语法

```
DOUBLE COT(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的余切值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT COT(in1) as aa  
FROM T1;
```

- 测试结果

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

5.10.2.18. EXP

本文为您介绍如何使用实时计算数学函数EXP。

语法

```
DOUBLE EXP(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回自然常数e的A次幂的DOUBLE类型数值。

示例

- 测试数据

in1(DOUBLE)
8.0
10.0

- 测试语句

```
SELECT EXP(in1) as aa  
FROM T1;
```

- 测试结果

aa(DOUBLE)
2980.9579870417283
22026.465794806718

5.10.2.19. E

本文为您介绍如何使用实时计算数学函数E。

语法

```
DOUBLE E()
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回自然常数e的DOUBLE类型值。

示例

- 测试数据

in1(DOUBLE)
8.0
10.0

- 测试语句

```
SELECT e() as dou1, E() as dou2  
FROM T1;
```

- 测试结果

dou1(DOUBLE)	dou2(DOUBLE)
2.718281828459045	2.718281828459045

5.10.2.20. FLOOR

本文为您介绍如何使用实时计算数学函数FLOOR。

语法

```
B FLOOR(A)
```

入参

参数	数据类型
A	INT、BIGINT、FLOAT、DOUBLE

功能描述

返回小于或等于A的最大整数数值，即舍去小数位的数值。B的数据类型与A的数据类型一致。

示例

- 测试数据

in1(DOUBLE)	in2(BIGINT)
8.123	3

- 测试语句

```
SELECT
FLOOR(in1) as out1,
FLOOR(in2) as out2
FROM T1;
```

- 测试结果

out1(DOUBLE)	out2(BIGINT)
8.0	3

5.10.2.21. LN

本文为您介绍如何使用实时计算数学函数LN。

语法

```
DOUBLE ln(DOUBLE number)
```

入参

参数	数据类型
number	DOUBLE  说明 若输入为VARCHAR类型或BIGINT类型，会隐式转换到DOUBLE类型后参与运算。

功能描述

返回number的自然对数。返回值是DOUBLE类型的对数值。

示例

- 测试数据

ID(INT)	X(DOUBLE)
1	100.0
2	8.0

- 测试语句

```
SELECT id, ln(x) as dou1, ln(e()) as dou2
FROM T1;
```

- 测试结果

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	4.605170185988092	1.0
2	2.0794415416798357	1.0

5.10.2.22. LOG

本文为您介绍如何使用实时计算数学函数LOG。

语法

```
DOUBLE LOG(DOUBLE base, DOUBLE x)
DOUBLE LOG(DOUBLE x)
```

入参

参数	数据类型
base	DOUBLE
x	DOUBLE

功能描述

返回以base为底的x的自然对数。返回值是DOUBLE类型的对数值。如果是只有一个参数的，返回以x为底的自然对数。

示例

- 测试数据

ID(INT)	BASE(DOUBLE)	X(DOUBLE)
1	10.0	100.0
2	2.0	8.0

- 测试语句

```
SELECT id, LOG(base, x) as dou1, LOG(2) as dou2
FROM T1;
```

- 测试结果

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

5.10.2.23. LOG10

本文为您介绍如何使用实时计算数学函数LOG10。

语法

```
DOUBLE LOG10(DOUBLE x)
```

入参

参数	数据类型
x	DOUBLE

功能描述

返回以10为底的自然对数。若x为NULL，则返回NULL。若x为负数会引发异常。

示例

- 测试数据

id(INT)	X(INT)
1	100
2	10

- 测试语句

```
SELECT id, log10(x) as dou1  
FROM T1;
```

- 测试结果

id(INT)	dou1(DOUBLE)
1	2.0
2	1.0

5.10.2.24. LOG2

本文为您介绍如何使用实时计算数学函数LOG2。

语法

```
DOUBLE LOG2(DOUBLE x)
```

入参

参数	数据类型
x	DOUBLE

功能描述

返回以2为底的自然对数。若x为NULL，则返回NULL。若x为负数，会引发异常。

示例

- 测试数据

id(INT)	X(INT)
1	8
2	2

- 测试语句

```
SELECT id, log2(x) as dou1  
FROM T1;
```

- 测试结果

id(INT)	dou1(DOUBLE)
1	3.0
2	1.0

5.10.2.25. PI

本文为您介绍如何使用实时计算数学函数PI。

语法

```
DOUBLE PI()
```

功能描述

获取圆周率常量PI值。

示例

- 测试数据

ID(INT)	X(INT)
1	8

- 测试语句

```
SELECT id, PI() as dou1
FROM T1;
```

- 测试结果

ID(INT)	dou1(DOUBLE)
1	3.141592653589793

5.10.2.26. POWER

本文为您介绍如何使用实时计算数学函数POWER。

语法

```
DOUBLE POWER(A, B)
```

入参

参数	数据类型
A	DOUBLE
B	DOUBLE

功能描述

返回A的B次幂。

示例

- 测试数据

in1(DOUBLE)	in2(DOUBLE)
2.0	4.0

- 测试语句

```
SELECT POWER(in1, in2) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
16.0

5.10.2.27. RAND

本文为您介绍如何使用实时计算数学函数RAND。

语法

```
DOUBLE RAND([BIGINT seed])
```

入参

参数	数据类型	说明
seed	BIGINT	Seed取值为随机数，决定随机数序列的起始值。

功能描述

返回大于等于0小于1的DOUBLE类型随机数。

示例

- 测试数据

id(INT)	X(INT)
1	8

- 测试语句

```
SELECT id, rand(1) as dou1, rand(3) as dou2  
FROM T1;
```

- 测试结果

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

5.10.2.28. SIN

本文为您介绍如何使用实时计算数学函数SIN。

语法

```
DOUBLE SIN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的正弦值。

示例

- 测试数据

in1(DOUBLE)
8.0
0.4

- 测试语句

```
SELECT SIN(in1) as aa  
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.9893582466233818
0.3894183423086505

5.10.2.29. SQRT

本文为您介绍如何使用实时计算数学函数SQRT。

语法

```
DOUBLE SQRT(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的平方根。

示例

- 测试数据

in1(DOUBLE)
8.0

- 测试语句

```
SELECT SQRT(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
2.8284271247461903

5.10.2.30. TAN

本文为您介绍如何使用实时计算数学函数TAN。

语法

```
DOUBLE TAN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

计算A的正切值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT TAN(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

5.10.2.31. CEIL

本文为您介绍如何使用实时计算数学函数CEIL。

语法

```
B CEIL(A)
```

入参

参数	数据类型
A	INT、BIGINT、FLOAT或DOUBLE
B	INT、BIGINT、FLOAT或DOUBLE

功能描述

输出B为大于或等于输入值A的最小整数数值。输出B的数据类型与输入参数A的数据类型一致。

示例

- 测试数据

in1(INT)	in2 (DOUBLE)
1	2.3

- 测试语句

```
SELECT  
CEIL (in1) as out1  
CEIL (in2) as out2  
FROM T1;
```

- 测试结果

out1(INT)	out2(DOUBLE)
1	3.0

5.10.2.32. CHARACTER_LENGTH

本文为您介绍如何使用实时计算数学函数CHARACTER_LENGTH。

语法

```
INTEGER CHARACTER_LENGTH( VARCHAR x )
```

入参

参数	数据类型
x	VARCHAR

功能描述

返回字符串x中的字符数。

示例

- 测试数据

ID(INT)	X(VARCHAR)
1	StreamCompute

- 测试语句

```
SELECT CHARACTER_LENGTH( x ) as result  
FROM T1;
```

- 测试结果

ID(INT)	result(INT)
1	13

5.10.2.33. DEGREES

本文为您介绍如何使用实时计算数学函数DEGREES。

语法

```
DOUBLE DEGREES( double x )
```

入参

参数	数据类型
x	DOUBLE

功能描述

将弧度x转换为数值。

示例

- 测试语句

```
SELECT DEGREES (PI()) as result  
FROM T1;
```

- 测试结果

result(DOUBLE)
180.0

5.10.2.34. MOD

本文为您介绍如何使用实时计算数学函数MOD。

语法

```
INTEGER MOD(INTEGER x,INTEGER y)
```

入参

参数	数据类型
x	INTEGER
y	INTEGER

功能描述

整数运算中，求整数x除以整数y的余数。当x为负值时，或者x、y均为负值时，结果为负值。

示例

- 测试数据

X (INT)	Y (INT)
29	3
-29	3
-29	-3

- 测试语句

```
SELECT MOD(x,y) as result  
FROM T1;
```

- 测试结果

ID(INT)	result(INT)
1	2
2	-2
3	-2

ID(INT)	result(INT)
---------	-------------

5.10.2.35. ROUND

本文为您介绍如何使用实时计算数学函数ROUND。

语法

```
T ROUND( T x, INT n)
```

入参

参数	数据类型
x	T参数类型，支持DECIMAL、TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE
n	INT

功能描述

把数值x字段舍入为指定的小数n位数。

示例1

- 测试数据 T1

in1(DECIMAL)
0.7173560908995228
0.4

- 测试语句

```
SELECT ROUND(in1,2) as `result`  
FROM T1;
```

- 测试结果

result(DECIMAL)
0.72
0.40

示例2

- 测试数据 T2

in2(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ROUND(in2,2) as `result`  
FROM T2;
```

- 测试结果

result(DOUBLE)
0.72
0.4

5.10.3. 日期函数

5.10.3.1. LOCALTIMESTAMP

本文为您介绍如何使用实时计算字符串函数LOCALTIMESTAMP。

语法

```
timestamp LOCALTIMESTAMP
```

入参

无

功能描述

返回当前系统的时间戳。

示例

- 测试语句

```
SELECT  
LOCALTIMESTAMP as `result`  
FROM T1;
```

- 测试结果

result (TIMESTAMP)
2018-07-27 14:04:38.998

5.10.3.2. CURRENT_DATE

本文为您介绍如何使用实时计算日期函数CURRENT_DATE。

语法

```
CURRENT_DATE
```

功能描述

返回当前系统日期。

示例

- 测试语句

```
SELECT CURRENT_DATE as res  
FROM T1;
```

- 测试结果

res(DATE)
2018-09-20

5.10.3.3. CURRENT_TIMESTAMP

本文为您介绍如何使用实时计算日期函数CURRENT_TIMESTAMP。

语法

```
TIMESTAMP CURRENT_TIMESTAMP
```

 说明 Blink 3.6.0以下版本，语法格式为TIMESTAMP CURRENT_TIMESTAMP ()。

功能描述

返回当前UTC（GMT+0）时间戳，时间戳单位为毫秒。

示例

- 测试语句

```
SELECT CURRENT_TIMESTAMP as var1  
FROM T1;
```


- 测试结果

```
var1(TIMESTAMP)
```

```
2007-04-30 13:10:02.047
```

5.10.3.4. DATEDIFF

本文为您介绍如何使用实时计算日期函数DATEDIFF。


 **说明** 建议在实时计算3.3.0及以上版本使用该函数，在3.3.0以下版本使用该函数时，数据结果可能不符合您的预期。

语法

```
INT DATEDIFF(VARCHAR enddate, VARCHAR startdate)
INT DATEDIFF(TIMESTAMP enddate, VARCHAR startdate)
INT DATEDIFF(VARCHAR enddate, TIMESTAMP startdate)
INT DATEDIFF(TIMESTAMP enddate, TIMESTAMP startdate)
```

入参

参数	数据类型
startdate	TIMESTAMP或VARCHAR
enddate	TIMESTAMP或VARCHAR

 **说明** VARCHAR日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。

功能描述

计算从enddate到startdate两个时间的天数差值，返回整数。若有参数为NULL或解析错误，返回NULL。

示例

- 测试数据

datetime1(VARCHAR)	datetime2(VARCHAR)	nullstr(VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

- 测试语句

```
SELECT 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	31	-31	null	null	null	31

5.10.3.5. DATE_ADD

本文为您介绍如何使用实时计算日期函数DATE_ADD。

语法

```
VARCHAR DATE_ADD(VARCHAR startdate, INT days)
VARCHAR DATE_ADD(TIMESTAMP time, INT days)
```

入参

参数	数据类型
startdate	TIMESTAMP或VARCHAR <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> ? 说明 VARCHAR类型日期格式：yyyy-MM-dd 或 yyyy-MM-dd HH:mm:ss。 </div>
enddate	TIMESTAMP
days	INT

功能描述

返回指定startdate日期days天数后的VARCHAR类型日期，返回string格式的日期为 yyyy-MM-dd 。如果有参数为null或解析错误，返回null。

示例

- 测试数据

datetime1(VARCHAR)	nullstr(VARCHAR)
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

5.10.3.6. DATE_FORMAT

本文为您介绍如何使用实时计算日期函数DATE_FORMAT。

语法

```
VARCHAR DATE_FORMAT(TIMESTAMP time, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR from_format, VARCHAR to_format)
```

入参

参数	数据类型
date	VARCHAR ? 说明 默认日期格式: yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
from_format	VARCHAR
to_format	VARCHAR

功能描述

将字符串类型的日期从源格式转换至目标格式。第一个参数（time 或 date）为源字符串。第二个参数from_format可选，为源字符串的格式，默认为yyyy-MM-dd hh:mm:ss。第三个参数为返回日期的格式，返回值为转换格式后的字符串类型日期。如果有参数为NULL或解析错误，则返回NULL。

示例

- 测试数据

date1(VARCHAR)	datetime1(VARCHAR)	nullstr(VARCHAR)
0915-2017	2017-09-15 00:00:00	NULL

- 测试语句

```
SELECT DATE_FORMAT(datetime1, 'yyMMdd') as var1,  
DATE_FORMAT(nullstr, 'yyMMdd') as var2,  
DATE_FORMAT(datetime1, nullstr) as var3,  
DATE_FORMAT(date1, 'MMdd-yyyy', nullstr) as var4,  
DATE_FORMAT(date1, 'MMdd-yyyy', 'yyyyMMdd') as var5,  
DATE_FORMAT(TIMESTAMP '2017-09-15 23:00:00', 'yyMMdd') as var6  
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)
170915	null	null	null	20170915	170915

5.10.3.7. DATE_SUB

本文为您介绍如何使用实时计算日期函数DATE_SUB。

语法

```
VARCHAR DATE_SUB(VARCHAR startdate, INT days)  
VARCHAR DATE_SUB(TIMESTAMP time, INT days)
```

入参

参数	数据类型
startdate	VARCHAR <p>说明 VARCHAR类型日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。</p>
time	TIMESTAMP
days	INT

功能描述

返回startdate减去days天数的日期。返回VARCHAR类型的yyyy-MM-dd日期格式。如果有参数为null或解析错误，返回null。

示例

- 测试数据

date1(VARCHAR)	nullstr(VARCHAR)
2017-10-15	null

- 测试语句

```
SELECT DATE_SUB(date1, 30) as var1,
DATE_SUB(TIMESTAMP '2017-10-15 23:00:00',30) as var2,
DATE_SUB(nullstr,30) as var3
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15	2017-09-15	null

5.10.3.8. DAYOFMONTH

本文为您介绍如何使用实时计算日期函数DAYOFMONTH。

语法

```
BIGINT DAYOFMONTH(TIMESTAMP time)
BIGINT DAYOFMONTH(DATE date)
```

入参

参数	数据类型
date	DATE
time	TIMESTAMP

功能描述

返回输入时间参数date或time中所指代的“日”。返回值范围为1~31。

示例

- 测试数据

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT DAYOFMONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
DAYOFMONTH(DATE '2017-09-22') as int2,
DAYOFMONTH(tdate) as int3,
DAYOFMONTH(ts) as int4,
DAYOFMONTH(CAST(dateStr AS DATE)) as int5,
DAYOFMONTH(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
15	22	10	15	15	15

5.10.3.9. EXTRACT

本文为您介绍如何使用实时计算日期函数EXTRACT。

语法

```
BIGINT EXTRACT(unit FROM time)
```

入参

参数	数据类型
time	任意日期表达式。

参数	数据类型
unit	可取值如下： <ul style="list-style-type: none"> • MICROSECOND • SECOND • MINUTE • HOUR • DAY • WEEK • MONTH • QUARTER • YEAR • SECOND_MICROSECOND • MINUTE_MICROSECOND • MINUTE_SECOND • HOUR_MICROSECOND • HOUR_SECOND • HOUR_MINUTE • DAY_MICROSECOND • DAY_SECOND • DAY_MINUTE • DAY_HOUR • YEAR_MONTH

功能描述

返回日期/时间的单独部分，例如年、月、日、小时、分钟、周数等。

示例

• 测试语句

```
EXTRACT(YEAR FROM CURRENT_TIMESTAMP) AS OrderYear,
EXTRACT(MONTH FROM CURRENT_TIMESTAMP) AS OrderMonth,
EXTRACT(DAY FROM CURRENT_TIMESTAMP) AS OrderDay,
EXTRACT(WEEK FROM CURRENT_TIMESTAMP) AS OrderWeek
```

• 测试结果

OrderYear(BIGINT)	OrderMonth(BIGINT)	OrderDay(BIGINT)	OrderWeek(BIGINT)
2018	10	11	41

5.10.3.10. FROM_UNIXTIME

本文为您介绍如何使用实时计算日期函数FROM_UNIXTIME。

语法

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime[, VARCHAR format])
```

入参

参数	数据类型
unixtime	BIGINT
format	VARCHAR

说明

- 参数unixtime为长整型，是以秒为单位的时间戳。
- 参数format可选，为日期格式，默认格式为yyyy-MM-dd HH:mm:ss，表示返回VARCHAR类型的符合指定格式的日期，如果有参数为null或解析错误，则返回null。

功能描述

返回值为VARCHAR类型的日期值，默认日期格式：yyyy-MM-dd HH:mm:ss，若指定日期格式按指定格式输出任一输入参数是NULL，返回NULL。

示例

测试数据

unixtime1(BIGINT)	nullstr(VARCHAR)
1505404800	null

测试语句

```
SELECT FROM_UNIXTIME(unixtime1) as var1,
FROM_UNIXTIME(unixtime1,'MMdd-yyyy') as var2,
FROM_UNIXTIME(unixtime1,nullstr) as var3
FROM T1;
```

测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15 00:00:00	0915-2017	null

5.10.3.11. HOUR

本文为您介绍如何使用实时计算日期函数HOUR。

语法

```
BIGINT HOUR(TIME time)
BIGINT HOUR(TIMESTAMP timestamp)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数time或timestamp中的24小时制的小时数，范围0~23。

示例

- 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT HOUR(TIMESTAMP '2016-09-20 23:33:33') AS int1,
HOUR(TIME '23:30:33') AS int2,
HOUR(time2) AS int3,
HOUR(timestamp1) AS int4,
HOUR(CAST(time1 AS TIME)) AS int5,
HOUR(TO_TIMESTAMP(datetime1)) AS int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
23	23	22	11	22	11

5.10.3.12. LOCALTIME

本文为您介绍如何使用实时计算日期函数LOCALTIME。

语法

```
TIME LOCALTIME
```

功能描述

以数据类型TIME的值返回会话时区中的当前时间。LOCALTIME可当变量直接使用。

示例

- 测试语句

```
SELECT LOCALTIME as `result`  
FROM T1;
```

- 测试结果

result(TIME)
19:00:47

5.10.3.13. MINUTE

本文为您介绍如何使用实时计算日期函数MINUTE。

语法

```
BIGINT MINUTE(TIME time)  
BIGINT MINUTE(TIMESTAMP timestamp)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中time或timestamp中的“分钟”部分。取值范围0~59。

示例

- 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT MINUTE(TIMESTAMP '2016-09-20 23:33:33') as int1,  
MINUTE(TIME '23:30:33') as int2,  
MINUTE(time2) as int3,  
MINUTE(timestamp1) as int4,  
MINUTE(CAST(time1 AS TIME)) as int5,  
MINUTE(CAST(datetime1 AS TIMESTAMP)) as int6  
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	30	23	12	23	12

5.10.3.14. MONTH

本文为您介绍如何使用实时计算日期函数MONTH。

语法

```
BIGINT MONTH(TIMESTAMP timestamp)
BIGINT MONTH(DATE date)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中的月，范围1~12。

示例

- 测试数据

a(TIMESTAMP)	b(DATE)
2016-09-15 00:00:00	2017-10-15

- 测试语句

```
SELECT
MONTH(cast( a as TIMESTAMP)) as int1,
MONTH(cast( b as DATE)) as int2
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)
9	10

5.10.3.15. NOW

本文为您介绍如何使用实时计算日期函数NOW。

语法

```
BIGINT NOW()  
BIGINT NOW(a)
```

入参

参数	数据类型
a	INT

功能描述

- 未指定参数时返回当前时区时间的戳，单位为秒。
- 可以在括号内输入INT类型参数作为偏移值（单位：秒），返回偏移后的时间戳。例如，`now(100)` 返回当前时间戳加100秒的时间戳。

 说明 偏移值a为NULL时，NOW(a)返回值为NULL。

示例

- 测试数据 T1

a(INT)
null

- 测试语句

```
SELECT  
  NOW() as now,  
  NOW(100) as now_100,  
  NOW(a) as now_null  
FROM T1;
```

- 测试结果

now(BIGINT)	now_100(BIGINT)	now_null(BIGINT)
1403006911	1403007011	null

5.10.3.16. SECOND

本文为您介绍如何使用实时计算日期函数SECOND。

语法

```
BIGINT SECOND(TIMESTAMP timestamp)  
BIGINT SECOND(TIME time)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中的“秒”部分，范围0~59。

示例

• 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

• 测试语句

```
SELECT SECOND(TIMESTAMP '2016-09-20 23:33:33') as int1,
SECOND(TIME '23:30:33') as int2,
SECOND(time2) as int3,
SECOND(timestamp1) as int4,
SECOND(CAST(time1 AS TIME)) as int5,
SECOND(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1;
```

• 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	33	24	13	24	13

5.10.3.17. TIMESTAMPADD

本文为您介绍如何使用实时计算日期函数TIMESTAMPADD。

语法

```
TIMESTAMP TIMESTAMPADD(interval,INT int_expr,TIMESTAMP datetime_expr)
DATE TIMESTAMPADD(interval,INT int_expr,DATE datetime_expr)
```

入参

参数	数据类型
interval	VARCHAR

参数	数据类型
int_expr	INT
datetime_expr	TIMESTAMP或DATE

interval可取值如下。

interval参数	时间间隔单位
FRAC_SECOND	毫秒
SECOND	秒
MINUTE	分钟
HOUR	小时
DAY	天
WEEK	星期
MONTH	月
QUARTER	季度
YEAR	年

功能描述

返回类型与datetime_expr类型相同。

将整型表达式int_expr添加日期或日期时间到表达式datetime_expr中，返回会话时区中的当前时间（数据类型TIME的值）。

示例

- 测试数据

a(TIMESTAMP)	b(DATE)
2018-07-09 10:23:56	1990-02-20

- 测试语句

```
SELECT  
TIMESTAMPADD(HOUR,3,a) AS `result1`  
TIMESTAMPADD(DAY,3,b) AS `result2`  
FROM T1;
```

- 测试结果

result1(TIMESTAMP)	result2(DATE)
2018-07-09 13:23:56.0	1990-02-23

5.10.3.18. TO_DATE

本文为您介绍如何使用实时计算日期函数TO_DATE。

语法

```
Date TO_DATE(INT time)
Date TO_DATE(VARCHAR date)
Date TO_DATE(VARCHAR date,VARCHAR format)
```

入参

参数	数据类型
time	INT ? 说明 表示从1970-1-1到所表示时间之间天数。
date	VARCHAR ? 说明 默认格式为yyyy-MM-dd。
format	VARCHAR

功能描述

将INT类型的日期或者VARCHAR类型的日期转换成DATE类型。

示例

- 测试数据

date1(INT)	date2(VARCHAR)	date3(VARCHAR)
100	2017-09-15	20170915

- 测试语句

```
SELECT TO_DATE(date1) as var1,
TO_DATE(date2) as var2,
TO_DATE(date3,'yyyyMMdd') as var3
FROM T1;
```

- 测试结果

var1(DATE)	var2(DATE)	var3(DATE)
1970-04-11	2017-09-15	2017-09-15

5.10.3.19. TO_TIMESTAMP

本文为您介绍如何使用实时计算Flink版日期函数TO_TIMESTAMP。

语法

```
TIMESTAMP TO_TIMESTAMP(BIGINT time)
TIMESTAMP TO_TIMESTAMP(VARCHAR date)
TIMESTAMP TO_TIMESTAMP(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
time	BIGINT <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 单位为毫秒。 </div>
date	VARCHAR <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 默认格式为 yyyy-MM-dd HH:mm:ss 。如果您的 date 为非默认格式，请使用自定义函数编写Java代码进行转化，详情请参见 自定义标量函数 (UDF) 。 </div>
format	VARCHAR

功能描述

将BIGINT类型的日期或者VARCHAR类型的日期转换成TIMESTAMP类型。

示例

- 测试数据

timestamp1(BIGINT)	timestamp2(VARCHAR)	timestamp3(VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

- 测试语句

```
SELECT TO_TIMESTAMP(timestamp1) as var1,
TO_TIMESTAMP(timestamp2) as var2,
TO_TIMESTAMP(timestamp3, 'yyyyMMddHHmmss') as var3
FROM T1;
```

- 测试结果

var1(TIMESTAMP)	var2(TIMESTAMP)	var3(TIMESTAMP)
2017-12-13 03:27:57.0	2017-09-15 00:00:00.0	2017-09-15 00:00:00.0

5.10.3.20. UNIX_TIMESTAMP

本文为您介绍如何使用实时计算日期函数UNIX_TIMESTAMP。

语法

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(VARCHAR date)
BIGINT UNIX_TIMESTAMP(TIMESTAMP timestamp)
BIGINT UNIX_TIMESTAMP(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
timestamp	TIMESTAMP
date	VARCHAR <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 默认日期格式为 yyyy-MM-dd HH:mm:ss 。 </div>
format	VARCHAR <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 默认格式为 yyyy-MM-dd hh:mm:ss 。 </div>

功能描述

返回date转换成的长整型的时间戳，单位为秒。无参数时返回当前时间的时间戳，单位为秒，与now语义相同。如果有参数为null或解析错误，返回null。

示例

- 测试数据

nullstr (VARCHAR)
null

- 测试语句

```
SELECT UNIX_TIMESTAMP() as big1,
       UNIX_TIMESTAMP(nullstr) as big2
FROM T1;
```

- 测试结果

big1 (BIGINT)	big2 (BIGINT)
1403006911	null

5.10.3.21. WEEK

本文为您介绍如何使用实时计算日期函数WEEK。

语法

```
BIGINT WEEK(DATE date)
BIGINT WEEK(TIMESTAMP timestamp)
```

入参

参数	数据类型
date	DATE
timestamp	TIMESTAMP

功能描述

计算指定日期在一年中的第几周，周数取值区间1~53。

示例

- 测试数据

dateStr(VARCHAR)	date1(DATE)	ts1(TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT WEEK(TIMESTAMP '2017-09-15 00:00:00') as int1,
       WEEK(date1) as int2,
       WEEK(ts1) as int3,
       WEEK(CAST(dateStr AS DATE)) as int4
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)
37	45	41	37

5.10.3.22. YEAR

本文为您介绍如何使用实时计算日期函数YEAR。

语法

```
BIGINT YEAR(TIMESTAMP timestamp)
BIGINT YEAR(DATE date)
```

入参

参数	数据类型
date	DATE
timestamp	TIMESTAMP

功能描述

返回输入时间的年份。

示例

- 测试数据

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT YEAR(TIMESTAMP '2016-09-15 00:00:00') as int1,
YEAR(DATE '2017-09-22') as int2,
YEAR(tdate) as int3,
YEAR(ts) as int4,
YEAR(CAST(dateStr AS DATE)) as int5,
YEAR(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
2016	2017	2017	2017	2015	2017

5.10.4. 逻辑函数

5.10.4.1. =

本文为您介绍如何使用实时计算逻辑运算函数=。

语法

```
A = B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	65

- 测试语句

```
SELECT int1 as aa  
FROM T1  
WHERE int3 = int2;
```

- 测试结果

aa(int)
97

5.10.4.2. >

本文为您介绍如何使用实时计算逻辑运算函数>。

语法

```
A > B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A大于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	100

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2;
```

- 测试结果

aa(int)
97

5.10.4.3. >=

本文为您介绍如何使用实时计算逻辑运算函数>=。

语法

```
A >= B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A大于等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	65
9	6	61

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 >= int2;
```

- 测试结果

aa(int)
97
9

5.10.4.4. <=

本文为您介绍如何使用实时计算逻辑运算函数<=。

语法

```
A <= B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A小于等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	65
9	6	5

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 <= int2;
```

- 测试结果

aa(int)
97
9

5.10.4.5. <

本文为您介绍如何使用实时计算逻辑运算函数<。

语法

```
A < B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A小于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	65
9	6	5

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2;
```

- 测试结果

aa(int)
97
9

5.10.4.6. <>

本文为您介绍如何使用实时计算逻辑运算函数<>。

语法

```
A <> B
```

入参

参数	数据类型
A	TIMESTAMP、BIGINT、INT、VARCHAR、DECIMAL
B	TIMESTAMP、BIGINT、INT、VARCHAR、DECIMAL

功能描述

如果A不等于B，则返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	6

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 <> int2;
```

- 测试结果

aa(int)
97

5.10.4.7. AND

本文为您介绍如何使用实时计算逻辑运算函数AND。

语法

```
A AND B
```

入参

参数	数据类型
A	BOOLEAN
B	BOOLEAN

功能描述

如果A和B均为TRUE，则为TRUE，否则为FALSE。

示例

• 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

• 测试语句

```
SELECT int2 as aa  
FROM T1  
WHERE int1=255 AND int3=65;
```

• 测试结果

aa(int)
97

5.10.4.8. BETWEEN AND

本文为您介绍如何使用实时计算Flink版逻辑运算函数BETWEEN AND。

语法

```
A BETWEEN B AND C
```

入参

参数	数据类型
A	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIME MP, TIME

参数	数据类型
B	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIME, TIMESTAMP, TIME
C	DOUBLE, BIGINT, INT, VARCHAR, DATE, TIME, TIMESTAMP, TIME

功能描述

BETWEEN操作符用于选取介于两个值之间的数据范围内的值。

示例1

- 测试数据

int1(INT)	int2(INT)	int3(INT)
90	80	100
11	10	7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1 BETWEEN int2 AND int3;
```

- 测试结果

aa(int)
90

示例2

- 测试数据

var1(varchar)	var2(varchar)	var3(varchar)
b	a	c

- 测试语句

```
SELECT var1 as aa
FROM T1
WHERE var1 BETWEEN var2 AND var3;
```

- 测试结果

aa(varchar)
b

示例3

- 测试数据

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

- 测试语句

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- 测试结果

aa(TIMESTAMP)
1969-07-20 20:17:30

5.10.4.9. IS NOT FALSE

本文为您介绍如何使用实时计算逻辑运算函数IS NOT FALSE。

语法

```
A IS NOT FALSE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE时，则返回TRUE。如果A是FALSE时，则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS NOT FALSE;
```


- 测试结果

aa(int)
97

5.10.4.10. IS NOT NULL

本文为您介绍如何使用实时计算逻辑运算函数IS NOT NULL。

语法

```
value IS NOT NULL
```

入参

参数	数据类型
value	任意数据类型

功能描述

如果value为 `NULL` 时，则返回 `FALSE` ，否则返回 `TRUE` 。

示例

- 测试数据

int1(INT)	int2(VARCHAR)
97	NULL
9	ww123

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NOT NULL;
```

- 测试结果

aa(int)
9

5.10.4.11. IS NOT TRUE

本文为您介绍如何使用实时计算逻辑运算函数IS NOT TRUE。

语法

A IS NOT TRUE

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE时，则返回FALSE。如果A是FALSE时，则返回TRUE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int1 as aa  
FROM T1  
WHERE int1=25 IS NOT TRUE;
```

- 测试结果

aa(int)
97

5.10.4.12. IS NOT UNKNOWN

本文为您介绍如何使用实时计算逻辑运算函数IS NOT UNKNOWN。

语法

A IS NOT UNKNOWN

入参

参数	数据类型
A	BOOLEAN

功能描述

A为逻辑比较表达式，例如：6<8。

正常情况下数值型与数值型作逻辑比较时，A值为TRUE或者FALSE。当其中一个不为数值型数据类型时，就会出现无法比较的情况。IS NOT UNKNOWN 就是判断这种情况是否存在。当两边无法进行正常的逻辑判断时，即A值既不是 TRUE 也不是 FALSE ，返回 FALSE 。可正常逻辑判断时，即A值为 TRUE 或者 FALSE ，返回 TRUE 。

示例一

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS NOT UNKNOWN;
```

- 测试结果

aa(int)
97

示例二

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1 < null IS NOT UNKNOWN;
```

- 测试结果

aa(int)
空

5.10.4.13. IS NULL

本文为您介绍如何使用实时计算逻辑运算函数IS NULL。

语法

```
value IS NULL
```

入参

参数	数据类型
value	任意数据类型

功能描述

如果value为 `NULL` 时，则返回 `TRUE`，否则返回 `FALSE`。

示例

• 测试数据

int1(INT)	int2(VARCHAR)
97	无
9	www

• 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NULL;
```

• 测试结果

aa(int)
97

5.10.4.14. IS TRUE

本文为您介绍如何使用实时计算逻辑运算函数IS TRUE。

语法

```
A IS TRUE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE时，则返回TRUE。如果A是FALSE时，则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa  
FROM T1  
WHERE int1=255 IS TRUE;
```

- 测试结果

aa(int)
97

5.10.4.15. IS UNKNOWN

本文为您介绍如何使用实时计算逻辑运算函数IS UNKNOWN。

语法

```
A IS UNKNOWN
```

入参

参数	数据类型
A	BOOLEAN

功能描述

IS UNKNOWN通过逻辑判断关系，返回结果：

- A（逻辑比较表达式）值既不是 TRUE 也不是 FALSE ，即无法进行正常的逻辑判断，返回 TRUE 。
- A（逻辑比较表达式）值为 TRUE 或者 FALSE ，即可以进行正常逻辑判断，返回 FALSE 。

正常情况下数值型与数值型进行逻辑比较时（例如 $6 < 8$ ），A值为TRUE或者FALSE。但是，当其中一个不为数值型数据类型时，就会出现无法比较的情况。IS UNKNOWN 用于判断这种情况是否存在。

示例一

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS UNKNOWN;
```

- 测试结果

aa(int)
空

示例二

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1 > null IS UNKNOWN;
```

- 测试结果

aa(int)
97

5.10.4.16. LIKE

本文为您介绍如何使用实时计算逻辑运算函数LIKE。

语法

```
A LIKE B
```

入参

参数	数据类型
A	VARCHAR
B	VARCHAR

功能描述

LIKE函数用于模糊查询。% 用于定义通配符。

示例1

- 测试数据

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR2 LIKE 'ss%';
```

- 测试结果

aa(int)
90
99

示例2

- 测试数据

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ho%';
```

- 测试结果

aa(int)
99

5.10.4.17. NOT

本文为您介绍如何使用实时计算逻辑运算函数NOT。

语法

```
NOT A
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是 `TRUE` 时，则返回 `FALSE`。如果A是 `FALSE` 时，则返回 `TRUE`。

示例

• 测试数据

int2(INT)	int3(INT)
97	65

• 测试语句

```
SELECT int2 as aa
FROM T1
WHERE NOT int3=62;
```

• 测试结果

aa(int)
97

5.10.4.18. NOT BETWEEN AND

本文为您介绍如何使用实时计算逻辑运算函数NOT BETWEEN AND。

语法

```
A NOT BETWEEN B AND C
```

入参

参数	数据类型
A	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME
B	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME
C	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME

功能描述

NOT BETWEEN AND 操作符用于选取不存在与两个值之间的数据范围内的值。

示例1

- 测试数据

int1(INT)	int2(INT)	int3(INT)
90	97	80
11	10	7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1 NOT BETWEEN int2 AND int3;
```

- 测试结果

aa(int)
11

示例2

- 测试数据

var1(varchar)	var2(varchar)	var3(varchar)
d	a	c

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE var1 NOT BETWEEN var2 AND var3;
```

- 测试结果

aa(varchar)
d

示例3

- 测试数据

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

- 测试语句

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 NOT BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- 测试结果

aa(TIMESTAMP)
1969-07-20 20:17:30

5.10.4.19. IN

本文为您介绍如何使用实时计算逻辑运算函数IN。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

入参

参数	数据类型
value1	常数
value2	常数

功能描述

用来查找在参数中的记录。

示例

- 测试数据

id(Int)	LastName(Varchar)
1	Adams
2	Bush
3	Carter

- 测试语句

```
SELECT *
FROM T1
WHERE LastName IN ('Adams','Carter');
```

- 测试结果

id(Int)	LastName(Varchar)
1	Adams
3	Carter

5.10.4.20. OR

本文为您介绍如何使用实时计算逻辑运算函数OR。

语法

```
A OR B
```

入参

参数	数据类型
A	BOOLEAN
B	BOOLEAN

功能描述

如果A和B中至少有一个为TRUE，则为TRUE，否则为FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 OR int3=65;
```

- 测试结果

aa(int)
97

5.10.4.21. IS DISTINCT FROM

本文为您介绍如何使用实时计算逻辑运算函数IS DISTINCT FROM。

语法

```
A IS DISTINCT FROM B
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

功能描述

- A和B的数据类型、值不完全相同则返回 **TRUE**。
- A和B的数据类型、值都相同返回 **FALSE**。
- 将空值视为相同。

示例

- 测试数据

A(int)	B(varchar)
97	97
null	sss
null	null

- 测试语句

```
SELECT  
A IS DISTINCT FROM B as 'result'  
FROM T1;
```

- 测试结果

result(BOOLEAN)
true
true
false

5.10.4.22. IS NOT DISTINCT FROM

本文为您介绍如何使用实时计算逻辑运算函数IS NOT DISTINCT FROM。

语法

A IS NOT DISTINCT FROM B

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

功能描述

- A和B的数据类型、值不完全相同则返回 `FALSE`。
- A和B的数据类型、值都相同返回 `TRUE`。
- 将空值视为相同。

示例

- 测试数据

A(int)	B(varchar)
97	97
null	sss
null	null

- 测试语句

```
SELECT  
A IS NOT DISTINCT FROM B as `result`  
FROM T1;
```

- 测试结果

result(BOOLEAN)
false
false
true

5.10.4.23. NOT IN

本文为您介绍如何使用实时计算逻辑运算函数NOT IN。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1,value2,...)
```

入参

参数	数据类型
value1	常数
value2	常数

功能描述

用来查找在不在参数中的记录。

示例

- 测试数据

id(Int)	LastName(Varchar)
1	Adams
2	Bush
3	Carter

- 测试语句

```
SELECT *
FROM T1
WHERE LastName NOT IN ('Adams','Carter');
```

- 测试结果

id(Int)	LastName(Varchar)
2	Bush

5.10.5. 条件函数

5.10.5.1. CASE WHEN

本文为您介绍如何使用实时计算条件函数CASE WHEN。

语法

```
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
```

功能描述

如果表达式a为TRUE，则返回b；如果表达式a为FALSE、c为TRUE，则返回d；如果表达式a和c都为FALSE，则返回e。

注意事项

CASE WHEN返回常量字符串时，会在字符串后面补全空格。例如，当满足else条件时，返回值 ios 后面会多几个空格。

```
case when device_type = 'android'  
then 'android'  
else 'ios'  
end as os
```

解决方法：

- 利用TRIM函数去除空格，该示例中，涉及 os 的字段都改为 TRIM(os)。
- 利用CAST函数去除空格，将常量字符串转为VARCHAR类型。

示例

- 测试数据

device_type(VARCHAR)
android
ios
win

- 测试语句
 - 利用TRIM函数

```
SELECT  
  trim(os), --添加trim。  
  CHAR_LENGTH(trim(os)) --添加trim。  
from(  
  SELECT  
    case when device_type = 'android'  
    then 'android'  
    else 'ios'  
  end as os  
FROM T1  
);
```

- 利用CAST函数

```
SELECT
  os,
  CHAR_LENGTH(os)
from
(SELECT
  case when device_type = 'android'
  then cast('android' as varchar) --添加cast。
  else cast('ios' as varchar) --添加cast。
  end as os
FROM T1
);
```

- 测试结果

os(VARCHAR)	length(INT)
android	7
ios	3
ios	3

5.10.5.2. COALESCE


本文为您介绍如何使用实时计算条件函数COALESCE。

语法

```
COALESCE(A,B,...)
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

 **说明** 所有这些值类型必须相同或为null，否则会引发异常。

功能描述

返回列表中第一个非null的值，返回值类型和参数类型相同。如果列表中所有的值都是null，则返回null。

 **说明** 至少要有有一个参数，否则引发异常。

示例

- 测试数据

var1(VARCHAR)	var2(VARCHAR)
null	30

- 测试语句

```
SELECT COALESCE(var1,var2) as aa  
FROM T1;
```

- 测试结果


aa(VARCHAR)
30

5.10.5.3. IF

本文为您介绍如何使用实时计算条件函数IF。

语法

```
T if (BOOLEAN testCondition, T valueTrue, T valueFalseOrNull)
```

 说明 T代表任意类型的返回值。

入参

参数	数据类型
testCondition	BOOLEAN
valueTrue	可以为任意类型，但valueTrue和valueFalseOrNull类型需保持一致。
valueFalseOrNull	可以为任意类型，但valueTrue和valueFalseOrNull类型需保持一致。

功能描述

以testCondition的布尔值为判断标准，返回对应的参数值：

- true：返回第2个参数valueTrue。
- false：返回第3个参数valueFalseOrNull。

说明

- 返回值类型为第2和第3参数的数据类型。
- 如果testCondition为null时，则判定结果为false。
- 如果其它参数为null时，则按照正常语义运行运算。

示例

测试数据

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null
1	2	null	Harry

测试语句

```
SELECT IF(int1 < int2, str1, str2) as int1  
FROM T1;
```

测试结果

int1(VARCHAR)
Jack
Jack
null

5.10.5.4. IS_ALPHA

本文为您介绍如何使用实时计算条件函数IS_ALPHA。

语法

```
BOOLEAN IS_ALPHA(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

如果str中只包含字母，则返回true，否则返回false。

示例

- 测试数据

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- 测试语句

```
SELECT IS_ALPHA(e) as boo1,IS_ALPHA(f) as boo2,IS_ALPHA(g) as boo3  
FROM T1;
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
false	true	false

5.10.5.5. IS_DECIMAL

本文为您介绍如何使用实时计算条件函数IS_DECIMAL。

语法

```
BOOLEAN IS_DECIMAL(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

str字符串如果可以转换为十进制数值，则返回TRUE，否则返回FALSE。

示例

- 测试数据

a(VARCHAR)	b(VARCHAR)	c(VARCHAR)	d(VARCHAR)	e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
1	123	2	11.4445	3	asd	null

- 测试语句

```
SELECT
IS_DECIMAL(a) as boo1,
IS_DECIMAL(b) as boo2,
IS_DECIMAL(c) as boo3,
IS_DECIMAL(d) as boo4,
IS_DECIMAL(e) as boo5,
IS_DECIMAL(f) as boo6,
IS_DECIMAL(g) as boo7
FROM T1;
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)	boo4(BOOLEAN)	boo5(BOOLEAN)	boo6(BOOLEAN)	boo7(BOOLEAN)
true	true	true	true	true	false	false

5.10.5.6. IS_DIGIT

本文为您介绍如何使用实时计算条件函数IS_DIGIT。

语法

```
BOOLEAN IS_DIGIT(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

如果str中只包含数字，则返回true，否则返回false。返回值为BOOLEAN类型。

示例

- 测试数据

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- 测试语句

```
SELECT
IS_DIGIT(e) as boo1,
IS_DIGIT(f) as boo2,
IS_DIGIT(g) as boo3
FROM T1;
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

5.10.5.7. NULLIF

本文为您介绍如何使用实时计算条件函数NULLIF。

语法

```
NULLIF(A,B)
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果两个参数的值相同，则返回null，如果值不同则返回第一个参数的值。

示例

- 测试数据

var1(INT)	var2(INT)
30	30

- 测试语句

```
SELECT NULLIF(var1,var2) as aa  
FROM T1;
```

- 测试结果

aa(INT)
null

5.10.6. 表值函数

5.10.6.1. GENERATE_SERIES

本文为您介绍如何使用实时计算表值函数GENERATE_SERIES。

语法

```
GENERATE_SERIES(INT from, INT to)
```

入参

参数	数据类型
from	INT类型，指定下界，包含下界值。
to	INT类型，指定上界，不包含上界值。

功能描述

生成一串连续的数值，分别为 from、from+1、from+2 ... to-1。

示例

- 测试数据

s(INT)	e(INT)
1	3
-2	1

- 测试语句

```
SELECT s, e, v
FROM T1, lateral table(GENERATE_SERIES(s, e))
as T(v);
```

- 测试结果

s(INT)	e(INT)	v(INT)
1	3	1
1	3	2
-2	1	-2
-2	1	-1
-2	1	-0

5.10.6.2. JSON_TUPLE

本文为您介绍如何使用实时计算Flink版表值函数JSON_TUPLE。

 **重要** 仅Blink支持JSON_TUPLE，Flink不支持JSON_TUPLE。

语法

```
JSON_TUPLE(str, path1, path2 ..., pathN)
```

入参

参数	数据类型	说明
str	VARCHAR	JSON字符串
path1 ~ pathN	VARCHAR	表示路径的字符串，前面不需要 \$。

功能描述

从JSON字符串中取出各路径字符串所表示的值。

示例

• 测试数据

d(VARCHAR)	s(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	qwe3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	qwe2

• 测试语句

```
SELECT d, v  
FROM T1, lateral table(JSON_TUPLE(d, 'qwe', s))  
AS T(v);
```

• 测试结果

d(VARCHAR)	v(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	asd
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	asd3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	asd4
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	asd5

5.10.6.3. STRING_SPLIT

本文为您介绍如何使用实时计算表值函数STRING_SPLIT。

语法

string_split(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 分隔符separator暂不支持多字符串形式，只支持单个字符串形式。

功能描述

根据指定的分隔符将目标字符串拆分为子字符串行，返回子字符串的单列的表。需要注意以下几点：

- 如果目标字符串为NULL，则STRING_SPLIT表值函数返回一个空行。
- 如果目标字符串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。
- 如果目标字符串未包含指定分隔符，则只返回目标字符串。

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

5.10.6.4. MULTI_KEYVALUE

本文为您介绍如何使用实时计算表值函数MULTI_KEYVALUE。

 说明 仅支持实时计算2.2.2及以上版本。

语法

```
MULTI_KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name1, VARCHAR key_name2, ...)
```

入参

参数	数据类型	说明
str	VARCHAR	字符串中的key-value (kv) 对。
split1	VARCHAR	kv对的分隔符。当split1为null时，表示按照whitespace作为kv对的分割符。当split1的长度>1时，split1仅表示分隔符的集合，每个字符都表示一个有效的分隔符。
split2	VARCHAR	kv的分隔符。当split2为null时，表示按照whitespace作为kv的分割符。当split2的长度>1时，split2仅表示分隔符的集合，每个字符都表示一个有效的分隔符。
key_name1, key_name2, ...	VARCHAR	需要获取value的key值列表。

功能描述

解析str字符串中的key-value对，匹配有split1和split2的key-value对，并返回参数列表里key_name1, key_name2等对应的value值列表。key_name值不存在时，对应的value值是null。

示例

• 测试数据

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)	key2(VARCHAR)
k1=v1;k2=v2	;	=	k1	k2
null	;	=	k1	k2
k1:v1;k2:v2	;	:	k1	k3
k1:v1;k2:v2	;	=	k1	k2
k1:v1;k2:v2	,	:	k1	k2
k1:v1;k2=v2	;	:	k1	k2

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)	key2(VARCHAR)
k1:v1abck2:v2	cab	:	k1	k2
k1:v1;k2=v2	;	:=	k1	k2
k1:v1 k2:v2	null	:	k1	k2
k1 v1;k2 v2	;	null	k1	k2

• 测试语句

```
SELECT c1, c2
FROM T1, lateral table(MULTI_KEYVALUE(str, split1, split2, key1, key2))
as T(c1, c2);
```

• 测试结果

c1(VARCHAR)	c2(VARCHAR)
v1	v2
null	null
v1	null
null	null
null	null
v1	null
v1	v2
v1	v2
v1	v2
v1	v2

5.10.7. 类型转换函数

5.10.7.1. CAST

本文为您介绍如何使用实时计算类型转换函数CAST。

语法

```
CAST(A AS type)
```

入参

参数	数据类型
A	请参见 类型转换 。

功能描述

将A值转换为给定类型。如果转换后的类型和目标表字段类型不匹配时，会出现类似 `Insert into: Query result and target table 'test_result' field type(s) not match.` 的报错。

示例

- 测试数据

var1(VARCHAR)	var2(INT)
1000	30

- 测试语句

```
SELECT CAST(var1 AS INT) as aa  
FROM T1;
```

- 测试结果

aa(INT)
1000

5.10.8. 聚合函数

5.10.8.1. AVG

本文为您介绍如何使用实时计算聚合函数AVG。Flink SQL中使用AVG函数返回指定表达式中所有值的平均值。

语法

```
AVG(A)
```

入参

参数	数据类型
A	TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL和DOUBLE。

功能描述

返回指定表达式中所有值的平均值。

说明 返回值默认为DOUBLE类型，如果您的结果表字段为非DOUBLE类型，您需要使用 **CAST** 进行转化。

示例

测试数据

var1(INT)	var2(INT)
4	30
6	30

测试语句

```
SELECT AVG(var1) as aa  
FROM T1;
```

测试结果

aa(INT)
5

5.10.8.2. CONCAT_AGG

本文为您介绍如何使用实时计算聚合函数CONCAT_AGG。Flink SQL中使用CONCAT_AGG函数将对应字段的所有字符串连接成新的字符串。

语法

```
CONCAT_AGG([linedelimiter,] value )
```

入参

参数	数据类型
linedelimiter	目前只支持字符串常量。可选。

功能描述

连接对应字段的字符串，默认连接符 `\n`，连接完成后新生成的字符串。返回值VARCHAR类型。

示例

测试数据

b(VARCHAR)	c(VARCHAR)
Hi	milk

b(VARCHAR)	c(VARCHAR)
Hi	milk
Hi	milk
Hi	milk
Hi	milk
Hi	milk
Hello	cola
Hello	cola
Happy	suda
Happy	suda

● 测试语句

```
SELECT
  b,
  concat_agg(c) as var1,
  concat_agg('-', c) as var2
FROM MyTable
GROUP BY b;
```

● 测试结果

b (VARCHAR)	var1(VARCHAR)	var2(VARCHAR)
Hi	milk milk milk milk milk milk	milk-milk-milk-milk-milk-milk
Hello	cola cola	cola-cola
Happy	suda suda	suda-suda

5.10.8.3. COUNT

本文为您介绍如何使用实时计算聚合函数COUNT。Flink SQL中使用COUNT函数返回输入列的数量。

语法

```
COUNT(A)
```

入参

参数	数据类型
A	<ul style="list-style-type: none">支持TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL、DOUBLE、BOOLEAN和VARCHAR类型。不支持DATE、TIME、TIMESTAMP和VARBINARY类型。

示例

- 测试数据

var1(VARCHAR)
1000
100
10
1

- 测试语句

```
SELECT COUNT(var1) as aa  
FROM T1;
```

- 测试结果

aa(BIGINT)
4

5.10.8.4. FIRST_VALUE

本文为您介绍如何使用实时计算聚合函数FIRST_VALUE。Flink SQL中使用FIRST_VALUE函数返回数据流的第一条非null数据。

语法

```
T FIRST_VALUE( T value )
T FIRST_VALUE( T value, BIGINT order )
```

入参

参数	数据类型
value	任意参数类型，但输入参数只能为同一种类型。
order	BIGINT

功能描述

获取数据流的第1条非null数据。根据order判定FIRST_VALUE所在的行，取order值最小的记录作为FIRST_VALUE。

示例1

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	"Hello"
2L	2	"Hello"
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

- 测试语句

```
SELECT c,
FIRST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(BIGINT)
"Hello"	1
"Hello"	1
"Hello"	1
"Hello"	1
"Hello"	1
"Hello"	1
"Hello World"	7
"Hello World"	7
"Hello World"	7

示例2

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	"Hello"	7
2L	2	"Hello"	7
3L	3	"Hello"	7
4L	4	"Hello"	7
5L	5	"Hello"	6
6L	6	"Hello"	4
7L	7	"Hello World"	3
8L	8	"Hello World"	2
20L	20	"Hello World"	1

- 测试语句

```
SELECT c,
FIRST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```


- 测试结果

c(VARCHAR)	var1(INT)
"Hello"	1
"Hello"	1
"Hello"	1
"Hello"	1
"Hello"	5
"Hello"	6
"Hello World"	7
"Hello World"	8
"Hello World"	20

5.10.8.5. LAST_VALUE

本文为您介绍如何使用实时计算聚合函数LAST_VALUE。Flink SQL中使用LAST_VALUE函数返回指定数据流的最后1条非NULL数据。

语法

```
T LAST_VALUE(T value)
T LAST_VALUE(T value, BIGINT order)
```

入参

参数	数据类型
value	任意参数类型 <div style="border: 1px solid #ccc; background-color: #e0f2f1; padding: 5px; margin-top: 5px;">? 说明 所有输入参数需要为相同的数据类型。</div>
order	BIGINT

功能描述

获取数据流的最后1条非NULL数据。根据ORDER判定LAST_VALUE所在的行，取ORDER值最大的记录作为LAST_VALUE。

示例1

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	"Hello"
2L	2	"Hello"
3L	3	"Hello"
4L	4	"Hello"
5L	5	"Hello"
6L	6	"Hello"
7L	7	"Hello World"
8L	8	"Hello World"
20L	20	"Hello World"

- 测试语句

```
SELECT c,
LAST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(INT)
"Hello"	1
"Hello"	2
"Hello"	3
"Hello"	4
"Hello"	5
"Hello"	6
"Hello World"	7
"Hello World"	8
"Hello World"	20

示例2

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	"Hello"	5
2L	2	"Hello"	5
3L	3	"Hello"	6
4L	4	"Hello"	5
5L	5	"Hello"	6
6L	6	"Hello"	5
7L	7	"Hello World"	4
8L	8	"Hello World"	8
20L	20	"Hello World"	9

- 测试语句

```
SELECT c,
LAST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(INT)
"Hello"	1
"Hello"	1
"Hello"	3
"Hello"	3
"Hello"	3
"Hello"	3
"Hello World"	7
"Hello World"	8
"Hello World"	20

5.10.8.6. MAX

本文为您介绍如何使用实时计算聚合函数MAX。Flink SQL中使用MAX函数返回所有输入值的最大值。

语法

```
MAX(A)
```

入参

参数	数据类型
A	<ul style="list-style-type: none">支持TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL、DOUBLE、BOOLEAN和VARCHAR类型。不支持DATE、TIME、TIMESTAMP和VARBINARY类型。

功能描述

返回所有输入值的最大值。

示例

- 测试数据

var1(INT)
4
8

- 测试语句

```
SELECT MAX(var1) as aa  
FROM T1;
```

- 测试结果

aa(INT)
8

5.10.8.7. MIN

本文为您介绍如何使用实时计算聚合函数MIN。Flink SQL中使用MIN函数返回所有输入值的最小值。

语法

```
MIN(A)
```

入参

参数	数据类型
A	<ul style="list-style-type: none">支持TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL、DOUBLE、BOOLEAN和VARCHAR类型。不支持DATE、TIME、TIMESTAMP和VARBINARY类型。

功能描述

返回所有输入值的最小值。

示例

- 测试数据

var1(INT)
4
8

- 测试语句

```
SELECT MIN(var1) as aa  
FROM T1;
```

- 测试结果

aa(INT)
4

5.10.8.8. SUM

本文为您介绍如何使用实时计算聚合函数SUM。Flink SQL中使用SUM函数来返回所有输入值的数值之和。

语法

```
SUM(A)
```

入参

参数	数据类型
A	TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL和DOUBLE。

功能描述

返回所有输入值的数值之和。

示例

- 测试数据

var1(INT)
4
4

- 测试语句

```
SELECT sum(var1) as aa  
FROM T1;
```

- 测试结果

aa(INT)
8

5.10.8.9. VAR_POP

Flink SQL中使用VAR_POP函数返回指定表达式中所有值的总体统计方差。

语法

```
T VAR_POP(T value)
```

入参

参数	数据类型
value	数值型，可以为BIGINT和DOUBLE等类型。

功能描述

返回所有输入值的方差。

示例

- 测试数据

a(BIGINT)	c(VARCHAR)
2900	Hi
2500	Hi
2600	Hi
3100	Hello

a(BIGINT)	c(VARCHAR)
11000	Hello

- 测试语句

```
SELECT
VAR_POP(a) as `result`,
c
FROM MyTable
GROUP BY c;
```

- 测试结果

result(BIGINT)	c
28889	Hi
15602500	Hello

5.10.8.10. STDDEV_POP

本文为您介绍如何使用实时计算聚合函数STDDEV_POP。Flink SQL中使用STDDEV_POP函数来返回数值的总体标准差。

语法

```
T STDDEV_POP(T value)
```

入参

参数	数据类型
value	BIGINT或DOUBLE

功能描述

返回数值的总体标准差。

示例

- 测试数据

a(DOUBLE)	c(VARCHAR)
0	Hi
1	Hi
2	Hi

a(DOUBLE)	c(VARCHAR)
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

- 测试语句

```
SELECT c, STDDEV_POP(a) as dou1
FROM MyTable
GROUP BY c;
```

- 测试结果

c(VARCHAR)	dou1(DOUBLE)
Hi	2.8722813232690143

5.10.9. 其他函数

5.10.9.1. UUID

本文为您介绍如何使用实时计算UUID。Flink SQL中使用UUID函数返回通用唯一标识字符。

语法

```
VARCHAR UUID()
```

功能描述

返回通用唯一标识字符。

示例

- 测试语句

```
SELECT uuid() as `result`
FROM T1;
```

- 测试结果


```
result(VARCHAR)
```

```
a364e414-e68b-4e5c-9166-65b3a153e257
```

5.10.9.2. DISTINCT

DISTINCT用于SELECT语句中，可以对查询结果进行去重。

DISTINCT语法

```
SELECT DISTINCT expressions
FROM tables;
```

- DISTINCT 必须放到开始位置。和其他函数一起使用时，DISTINCT 也必须放到开始位置，例如，concat_agg(DISTINCT ',device_id')。
- expressions 是一个或多个expression，可以是具体的column，也可以是function等任何合法表达式。

DISTINCT示例

- SQL语句

Flink SQL中DISTINCT的示例如下所示。

```
CREATE TABLE distinct_tab_source(
  FirstName VARCHAR,
  LastName VARCHAR
)WITH(
  type='random'
);
CREATE TABLE distinct_tab_sink(
  FirstName VARCHAR,
  LastName VARCHAR
)WITH(
  type = 'print'
);
INSERT INTO distinct_tab_sink
SELECT DISTINCT FirstName, LastName --按照FirstName和LastName两个列进行去重。
FROM distinct_tab_source;
```

- 测试数据

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- 查询结果

序号	操作	FirstName	LastName
1	Insert	SUNS	HENGRAN
2	Insert	SUN	JINCHENG
3	Insert	SUN	SHENGRAN

说明

- 测试数据有4条记录。DISTINCT FirstName, LastName 将SUN和SHENGRAN去重后，输出3条结果记录。
- SUNS,HENGRAN 和 SUN,SHENGRAN 两条记录并没有被去重，说明 DISTINCT FirstName, LastName 是对两个字段分别处理的，而不是Concat到一起再进行去重。

DISTINCT的等效写法

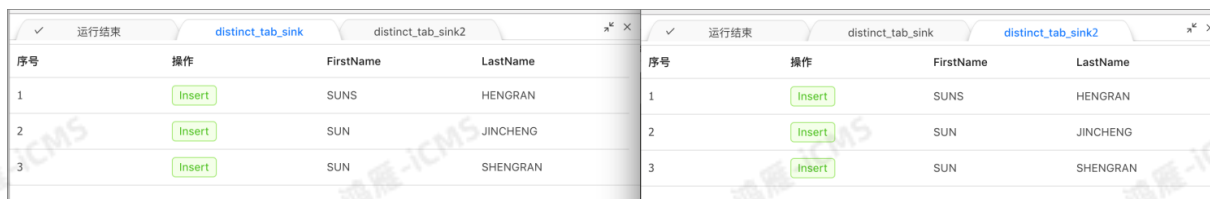
在SQL中利用 GROUP BY 语句也可以到达和 DISTINCT 类似的去重效果。GROUP BY 语法如下。

```
SELECT expressions
FROM tables
GROUP BY expressions
;
```

通过多路输出的方式编写的和DISTINCT示例等效的SQL，示例如下。

```
CREATE TABLE distinct_tab_source(
  FirstName VARCHAR,
  LastName VARCHAR
)WITH(
  type='random'
);
CREATE TABLE distinct_tab_sink(
  FirstName VARCHAR,
  LastName VARCHAR
)WITH(
  type = 'print'
);
CREATE TABLE distinct_tab_sink2(
  FirstName VARCHAR,
  LastName VARCHAR
)WITH(
  type = 'print'
);
INSERT INTO distinct_tab_sink
  SELECT DISTINCT FirstName, LastName --按照FirstName和LastName两个列进行去重。
  FROM distinct_tab_source;
INSERT INTO distinct_tab_sink2
  SELECT FirstName, LastName
  FROM distinct_tab_source
  GROUP BY FirstName, LastName; --按照FirstName和LastName两个列进行去重。
```


和DISTINCT示例使用相同的测试数据，最终的结果数据相同，说明两种方式的语义一致。



DISTINCT在聚合函数COUNT中的作用

DISTINCT 能使聚合函数 COUNT 统计去重后的计数。

COUNT(DISTINCT expression)

 **重要** expression 目前只支持一个表达式。

COUNT DISTINCT语法示例

- SQL语法

```
CREATE TABLE distinct_tab_source(  
  FirstName VARCHAR,  
  LastName VARCHAR  
)WITH(  
  type='random'  
);  
CREATE TABLE distinct_tab_sink(  
  cnt BIGINT,  
  distinct_cnt BIGINT  
)WITH(  
  type = 'print'  
);  
INSERT INTO distinct_tab_sink  
  SELECT  
    COUNT(FirstName), --不去重。  
    COUNT(DISTINCT FirstName) --按照FirstName去重。  
  FROM distinct_tab_source;
```

- 测试数据

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- 测试结果

序号	操作	cnt	distinct_cnt
1	Insert	1	1
2	Insert	2	2
3	Insert	3	2
4	Insert	4	2

5.11. 自定义函数（UDX）

5.11.1. 概述

本文为您介绍如何搭建实时计算Flink版自定义函数的环境并使用自定义函数。

重要

- 仅独享模式支持自定义函数。
- Blink在开源Flink SQL的基础上对性能进行了增强，Blink是阿里云实时计算版本的Flink。UDX函数仅适用于Blink，对开源Flink暂不适用。
- 为了避免JAR依赖冲突，您需要注意以下几点：
 - 开发页面选择的Blink版本，请和Pom依赖Blink版本保持一致。
 - Blink相关依赖，scope请使用provided，即 `<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](#)。

UDX分类

实时计算Flink版支持以下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，其输入与输出是一一对应的关系，即读入一行数据，写出一条输出值。
UDAF (User Defined Aggregation Function)	自定义聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值。可以与SQL中的GROUP BY语句一起使用。具体语法请参见 聚合函数 。
UDTF (User Defined Table-valued Function)	自定义表值函数，调用一次函数输出多行或多列数据。

UDX示例

实时计算Flink版为您提供UDX示例，便于您快速开发业务。实时计算Flink版UDX示例中包含UDF、UDAF和UDTF的实现，示例如下。

说明

- 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。
- 示例为Maven项目，您可以使用IntelliJ IDEA进行开发。开发方法请参见 [开发](#)。

- 实时计算Flink版3.0版本

[Blink_UDX_3x](#)

- 实时计算Flink版2.0版本

[Blink_UDX_2x](#)

- 实时计算Flink版1.0版本

[Blink_UDX_1x](#)

环境搭建

以上示例主要使用的依赖JAR包如下，如果您需要单独使用，可以自行下载。

- 基于实时计算Flink版3.2.1以下版本

- [flink-streaming-java_2.11](#)
- [flink-table_2.11](#)
- [flink-core-blink-2.2.4](#)

- 基于实时计算Flink版3.2.1及以上版本

请根据需求自行添加开源版本所支持的 **POM依赖包**。下载查看 [完整依赖包示例](#)。

说明 如果您需要依赖Snapshot版本，可以自行添加Snapshot版本所支持的 **POM依赖包**。

注册使用

1. 登录 [实时计算控制台](#)。
2. 在顶部菜单中，单击 [开发](#)。
3. 在左侧的导航栏中，单击 [资源引用](#)。
4. 在资源引用页签的右上角，单击 [新建资源](#)。
5. 在上传资源页面，输入资源配置信息。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 <p>说明 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。</p>
资源选择	单击选择资源，选择需要引用的资源。

参数名称	说明
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

- 在资源引用页签中，将鼠标悬停在对应作业的右侧的 **更多**上。
- 在下拉列表中，选择引用。
- 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

参数与返回值类型

实时计算Flink版支持定义Java UDX时，使用Java类型作为参数和返回值。下面为实时计算Flink版类型和Java类型的映射关系。

实时计算Flink版数据类型	Java类型
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL	java.math.BigDecimal
BOOLEAN	java.lang.Boolean
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CHAR	java.lang.Character
STRING	java.lang.String
VARBINARY	java.lang.byte[]
ARRAY	暂不支持
MAP	暂不支持

获取自定义函数参数

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，自定义配置项可以通过此对象来传递。

假设，您需要在作业中添加如下两个参数。

```
testKey1=lincoln  
test.key2=todd
```


以UDTF为例，在Open方法中通过 `context.getJobParameter` 即可获取，示例如下。

```
public void open(FunctionContext context) throws Exception {  
    String key1 = context.getJobParameter("testKey1", "empty");  
    String key2 = context.getJobParameter("test.key2", "empty");  
    System.err.println(String.format("end open: key1:%s, key2:%s", key1, key2));  
}
```

 说明 具体的作业参数请参见 [作业参数](#)。

5.11.2. 自定义标量函数（UDF）

本文为您介绍如何为实时计算Flink版自定义标量函数（UDF）搭建开发环境、编写业务代码及上线。

 重要 阿里云实时计算Flink版共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义


自定义标量函数（UDF）将0个、1个或多个标量值映射到一个新的标量值。

搭建环境

搭建开发环境，请参见 [环境搭建](#)。

业务代码

UDF需要在ScalarFunction类中实现 `eval` 方法。`open` 方法和 `close` 方法可选。

 重要 UDF默认对于相同的输入会有相同的输出。如果UDF不能保证相同的输出，例如，在UDF中调用外部服务，相同的输入值可能返回不同的结果，建议您使用 `override isDeterministic()` 方法，返回 `False`。否则在某些条件下，输出结果不符合预期。例如，UDF算子前移。

以Java为例，示例代码如下。

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
    // 可选，open方法可以不写。
    // 如果编写open方法需要声明'import org.apache.flink.table.functions.FunctionContext;'。
    @Override
    public void open(FunctionContext context) {
    }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不写。
    @Override
    public void close() {
    }
}
```

编写SQL语句

在指定的Class中编写SQL语句，自定义函数中SQL语句示例如下。


```
-- udf str.length()
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
create table sls_stream(
  a int,
  b int,
  c varchar
) with (
  type='sls',
  endPoint='<yourEndpoint>',
  accessKeyId='<yourAccessId>',
  accessKeySecret='<yourAccessSecret>',
  startTime = '2017-07-04 00:00:00',
  project='<yourProjectName>',
  logStore='<yourLogStoreName>',
  consumerGroup='consumerGroupTest1'
);
create table rds_output(
  id int,
  len bigint,
  content VARCHAR
) with (
  type='rds',
  url='yourDatabaseURL',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
insert into rds_output
select
  a,
  stringLengthUdf(c),
  c as content
from sls_stream;
```

注册使用

1. 登录[实时计算控制台](#)。
2. 在顶部菜单中，单击开发。
3. 在左侧的导航栏中，单击资源引用。
4. 在资源引用页签的右上角，单击新建资源。
5. 在上传资源页面，输入资源配置信息。

参数名称	说明
------	----

参数名称	说明
上传方式	<p>实时计算控制台上仅支持本地上传。</p> <p> 说明 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。</p>
资源选择	单击选择资源，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

- 在资源引用页签中，将鼠标悬停在对应作业的右侧的 **更多** 上。
- 在下拉列表中，选择引用。
- 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

上线和启动

在作业开发页面单击上线后，在运维页面单击启动，即可完成自定义函数的上线。


常见问题

Q: 为什么实现了一个随机数生成函数，在运行时产生的值却一样？

A: 如果自定义函数是无参的，并且没有声明是非确定性函数，编译期间可能会被优化成一个常量值。如果需要让函数变成非确定性函数，不被优化为常量，建议您使用 `override isDeterministic()` 方法，返回 `False`。

5.11.3. 自定义聚合函数（UDAF）

本文为您介绍如何为实时计算Flink版自定义聚合函数（UDAF）搭建开发环境、编写业务代码及上线。

 **重要** 阿里云实时计算Flink版共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

自定义聚合函数（UDAF）可以将多条记录聚合成1条记录。

UDAF抽象类内部方法

❓ 说明 虽然UDAF可以使用Java或Scala实现，但是建议您使用Java，因为Scala的数据类型有时会造成不必要的性能损失。

AggregateFunction的核心接口方法，如下所示：

- createAccumulator和getValue方法

```
/*
 * @param <T> UDAF的输出结果的类型。
 * @param <ACC> UDAF的accumulator的类型。accumulator是UDAF计算中用来存放计算中间结果的数据类型。
 * 您可以根据需要根据需要自行设计每个UDAF的accumulator。
 */
public abstract class AggregateFunction<T, ACC> extends UserDefinedFunction {
    /*
     * 初始化AggregateFunction的accumulator。
     * 系统在进行第一个aggregate计算之前，调用一次此方法。
     */
    public ACC createAccumulator();
    /*
     * 系统在每次aggregate计算完成后，调用此方法。
     */
    public T getValue(ACC accumulator);
}
```

❓ 说明

- createAccumulator和getValue可以定义在AggregateFunction抽象类内。
- UDAF必须包含1个accumulate方法。

- accumulate方法

```
public void accumulate(ACC accumulator, ...[用户指定的输入参数]...);
```

❓ 说明

- 您需要实现一个accumulate方法，来描述如何计算输入的数据，并更新数据到accumulator中。
- accumulate方法的第一个参数必须是使用AggregateFunction的ACC类型的accumulator。在系统运行过程中，runtime代码会把accumulator的历史状态和您指定的上游数据（支持任意数量，任意类型的数据）作为参数，一起传递给accumulate方法。

- retract和merge方法

createAccumulator、getValue和accumulate 3个方法一起使用，可以设计出一个最基本的UDAF。但是实时计算Flink版一些特殊的场景需要您提供retract和merge两个方法才能完成。

通常，计算都是对无限流的一个提前的观测值（early firing）。既然有early firing，就会有对发出的结果的修改，这个操作叫作撤回（retract）。SQL翻译优化器会帮助您自动判断哪些情况下会产生撤回的数据，哪些操作需要处理带有撤回标记的数据。但是您需要实现一个retract方法来处理撤回的数据。

```
public void retract(ACC accumulator, ...[您指定的输入参数]...);
```

说明

- retract方法是accumulate方法的逆操作。例如，实现Count功能的UDAF，在使用accumulate方法时，每来一条数据要加1；在使用retract方法时，就要减1。
- 类似于accumulate方法，retract方法的第1个参数必须使用AggregateFunction的ACC类型的accumulator。在系统运行过程中，runtime代码会把accumulator的历史状态，和您指定的上游数据（任意数量，任意类型的数据）一起发送给retract计算。

在实时计算Flink版中一些场景需要使用merge方法，例如session window。由于实时计算Flink版具有out of order的特性，后输入的数据有可能位于2个原本分开的session中间，这样就把2个session合为1个session。此时，需要使用merge方法把多个accumulator合为1个accumulator。

```
public void merge(ACC accumulator, Iterable<ACC> its);
```

说明

- merge方法的第1个参数，必须是使用AggregateFunction的ACC类型的accumulator，而且第1个accumulator是merge方法完成之后，状态所存放的地方。
- merge方法的第2个参数是1个ACC类型的accumulator遍历迭代器，里面有可能存在1个或多个accumulator。

搭建开发环境

搭建开发环境请参见 [环境搭建](#)。

编写业务逻辑代码


以Java为例，举例代码如下。

```
import org.apache.flink.table.functions.AggregateFunction;
public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    //定义存放count UDAF状态的accumulator的数据的结构。
    public static class CountAccum {
        public long total;
    }
    //初始化count UDAF的accumulator。
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }
    //getValue提供了如何通过存放状态的accumulator计算count UDAF的结果的方法。
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }
    //accumulate提供了如何根据输入的数据更新count UDAF存放状态的accumulator。
    public void accumulate(CountAccum accumulator, Object iValue) {
        accumulator.total++;
    }
    public void merge(CountAccum accumulator, Iterable<CountAccum> its) {
        for (CountAccum other : its) {
            accumulator.total += other.total;
        }
    }
}
}
```

 **说明** AggregateFunction的子类支持open和close方法作为可选方法，请参见 [自定义标量函数（UDF）](#) 或 [自定义表值函数（UDTF）](#) 的写法。

注册使用

1. 登录[实时计算控制台](#)。
2. 在顶部菜单中，单击**开发**。
3. 在左侧的导航栏中，单击**资源引用**。
4. 在资源引用页签的左上角，单击**新建资源**。
5. 在上传资源页面，输入资源配置信息。

参数名称	说明
上传方式	<p>实时计算控制台上仅支持本地上传。</p> <p> 说明 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。</p>
资源选择	单击 选择资源 ，选择需要引用的资源。

参数名称	说明
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

- 在资源列表中，将鼠标悬停在目标资源右侧的 **更多** 上。
- 在下拉列表中，单击 **引用**。
- 在作业编辑窗口，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

上线和启动


自定义聚合函数（UDAF）的上线和启动步骤，请参见 [上线和启动](#)。

示例

```
-- UDAF计算count
CREATE FUNCTION countUdaf AS 'com.hjc.test.blink.sql.udx.CountUdaf';
create table sls_stream(
  a int,
  b bigint,
  c varchar
) with (
  type='sls',
  endPoint='yourEndpoint',
  accessKeyId='yourAccessId',
  accessKeySecret='yourAccessSecret',
  startTime='2017-07-04 00:00:00',
  project='<yourPorjectName>',
  logStore='stream-test2',
  consumerGroup='consumerGroupTest3'
);
create table rds_output(
  len1 bigint,
  len2 bigint
) with (
  type='rds',
  url='yourDatabaseURL',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
insert into rds_output
select
  count(a),
  countUdaf(a)
from sls_stream;
```

5.11.4. 自定义表值函数（UDTF）

本文为您介绍如何为实时计算Flink版自定义表值函数（UDTF）搭建开发环境、编写业务代码以及上线。

 **说明** 阿里云实时计算Flink版共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

与自定义的标量函数类似，自定义的表值函数（UDTF）将0个、1个或多个标量值作为输入参数（可以是变长参数）。与标量函数不同，表值函数可以返回任意数量的行作为输出，而不仅是1个值。返回的行可以由1个或多个列组成。

搭建开发环境

参见[环境搭建](#)。

编写业务逻辑代码

UDTF需要在TableFunction类中实现eval方法。open方法和close方法可选。以Java为例，示例代码如下。

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
public class SplitUdtf extends TableFunction<String> {
    // 可选，open方法可不编写。如果编写，则需要添加声明'import org.apache.flink.table.functions.FunctionContext;'。
    @Override
    public void open(FunctionContext context) {
        // ... ..
    }
    public void eval(String str) {
        String[] split = str.split("\\\\");
        for (String s : split) {
            collect(s);
        }
    }
    // 可选，close方法可不编写。
    @Override
    public void close() {
        // ... ..
    }
}
```

多行返回

UDTF可以通过多次调用 `collect()` 实现将1行的数据转为多行返回。

多列返回

UDTF不仅可以进行1行转多行，还可以1列转多列。如果您需要UDTF返回多列，只需要将返回值声明成Tuple或Row。Tuple或Row解释如下：

- 返回值为Tuple

实时计算Flink版支持使用Tuple1到Tuple25，定义1个字段到25个字段。用Tuple3来返回3个字段的UDTF示例如下。


```
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.table.functions.TableFunction;
// 使用Tuple作为返回值，一定要显式声明Tuple的泛型类型，例如，String、Long和Integer。
public class ParseUdtf extends TableFunction<Tuple3<String, Long, Integer>> {
    public void eval(String str) {
        String[] split = str.split(",");
        // 以下代码仅作参考，实际业务需要添加更多的校验逻辑。
        String first = split[0];
        long second = Long.parseLong(split[1]);
        int third = Integer.parseInt(split[2]);
        Tuple3<String, Long, Integer> tuple3 = Tuple3.of(first, second, third);
        collect(tuple3);
    }
}
```

 说明 使用Tuple时，字段值不能为null，且最多只能存在25个字段。

- 返回值为Row

使用Row来实现返回3个字段的UDTF示例如下。

```
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
public class ParseUdtf extends TableFunction<Row> {
    public void eval(String str) {
        String[] split = str.split(",");
        String first = split[0];
        long second = Long.parseLong(split[1]);
        int third = Integer.parseInt(split[2]);
        Row row = new Row(3);
        row.setField(0, first);
        row.setField(1, second);
        row.setField(2, third);
        collect(row);
    }
    @Override
    // 如果返回值是Row，则必须重载实现getResultType方法，显式地声明返回的字段类型。
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.STRING, DataTypes.LONG, DataTypes.INT);
    }
}
```

 说明 Row的字段值可以是null，但如果需要使用Row，必须重载实现 `getResultType` 方法。

SQL语法

UDTF支持cross join和left join，在使用UDTF时需要添加 `lateral` 和 `table` 关键字。以 `ParseUdtf` 为例，需要先注册一个Function名字。

```
CREATE FUNCTION parseUdtf AS 'com.alibaba.blink.sql.udtf.ParseUdtf';
```


- cross join

左表的每一行数据都会关联上UDTF产出的每一行数据，如果UDTF不产出任何数据，则这1行不会输出。

```
select S.id, S.content, T.a, T.b, T.c
from input_stream as S,
lateral table(parseUdtf(content)) as T(a, b, c);
```

- left join

左表的每一行数据都会关联上UDTF产出的每一行数据，如果UDTF不产出任何数据，则这1行的UDTF的字段会用null值填充。

 说明 left join UDTF语句后面必须添加 `on true` 参数。

```
select S.id, S.content, T.a, T.b, T.c
from input_stream as S
left join lateral table(parseUdtf(content)) as T(a, b, c) on true;
```

注册使用

1. 登录[实时计算控制台](#)。
2. 在顶部菜单中，单击开发。
3. 在左侧的导航栏中，单击资源引用。
4. 在资源引用页签的右上角，单击新建资源。
5. 在上传资源页面，输入资源配置信息。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。  说明 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击选择资源，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

6. 在资源引用页签中，将鼠标悬停在对应作业的右侧的[更多](#)上。

- 在下拉列表中，选择引用。
- 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

上线和启动

自定义聚合函数（UDTF）的上线和启动步骤，请参见[上线和启动](#)。

UDTF示例

```
-- UDTF str.split("\\|");
create function splitUdtf as 'com.hjc.test.blink.sql.udx.SplitUdtf';
create table sls_stream(
  a INT,
  b BIGINT,
  c VARCHAR
) with (
  type='sls',
  endPoint='yourEndpoint',
  accessKeyId='yourAccessKeyId',
  accessKeySecret='yourAccessSecret',
  startTime = '2017-07-04 00:00:00',
  project='yourProjectName',
  logStore='yourLogStoreName',
  consumerGroup='consumerGroupTest2'
);
-- 将c字段传入splitUdtf，切分后得到多行1列的表T(s)。s表示字段名字。
create view v1 as
select a,b,c,s
from sls_stream,
lateral table(splitUdtf(c)) as T(s);
create table rds_output(
  id INT,
  len BIGINT,
  content VARCHAR
) with (
  type='rds',
  url='yourDatabaseURL',
  tableName='yourDatabaseTableName',
  userName='yourDatabaseUserName',
  password='yourDatabasePassword'
);
insert into rds_output
select
a,b,s
from v1;
```

5.11.5. 使用IntelliJ IDEA开发自定义函数

本文为您介绍如何使用IntelliJ IDEA开发实时计算Flink版自定义函数，包括搭建开发环境和实时计算Flink版作业中引用自定义函数。

背景信息

重要

- 仅独享模式支持自定义函数功能。
- 请使用 **IntelliJ IDEA工具** 开发自定义函数。

配置Maven

1. 下载Maven。

- i. 登录 [Maven官网下载页面](#)。
- ii. 下载 `apache-maven-3.5.3-bin.tar.gz`。
- iii. 解压下载的安装包到指定目录，例如：`/Users/<userName>/Documents/maven`。

2. 配置环境变量。

- i. 在Terminal中，执行 `vim ~/.bash_profile` 命令。
- ii. 在 `.bash_profile` 文件中添加如下命令。

```
export M2_HOME=/Users/<userName>/Documents/maven/apache-maven-3.5.3
export PATH=$PATH:$M2_HOME/bin
```

- iii. 保存并关闭 `.bash_profile` 文件。
- iv. 执行 `source ~/.bash_profile` 命令使配置生效。

3. 执行 `mvn -v` 命令查看配置是否生效。

如果打印如下信息，则配置生效。

```
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-04T03:39:06+08:00)
Maven home: /Users/<userName>/Documents/maven/apache-maven-3.5.0
Java version: 1.8.0_121, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre
Default locale: zh_CN, platform encoding: UTF-8
OS name: "mac os x", version: "10.12.6", arch: "x86_64", family: "mac"
```

搭建开发环境

1. 下载 [UDX示例](#)。
2. 在Linux环境下解压下载文件。

```
tar xzvf RealtimeCompute-udxDemo.gz
```

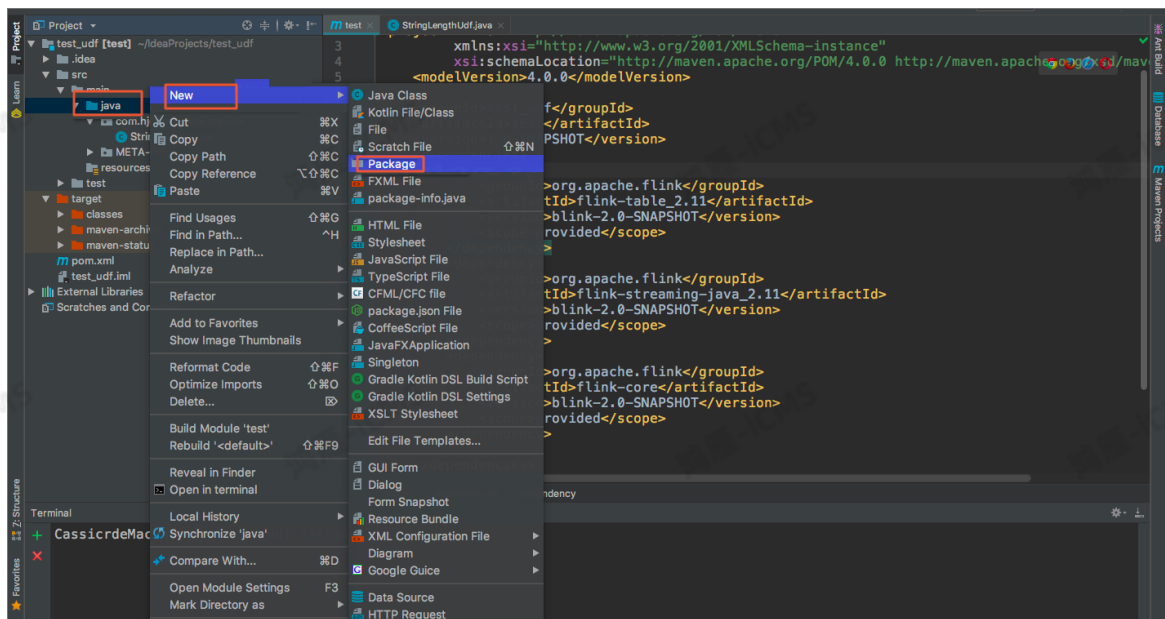
3. 打开IntelliJ IDEA，单击Open打开下载的UDX示例。



实时计算Flink版作业引用JAR包

1. 创建Package。

i. 右键单击java > New > Package。




ii. 在New Package中输入Package名称。本文以 `com.hjc.test.blink.sql.udx` 为例。

iii. 单击OK。

2. 创建Class。

i. 右键单击 `com.hjc.test.blink.sql.udx` > New > Java Class。

ii. 在Create New Class中，输入Class名称。

 说明 Kind保持默认选择Class。

iii. 单击OK。

3. 在Class中输入以下代码。

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
    // 可选，open方法可以不编写。
    // 如果编写open方法，需要声明'import org.apache.flink.table.functions.FunctionContext;'。
    @Override
    public void open(FunctionContext context) {
    }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不编写。
    @Override
    public void close() {
    }
}
```

4. 在Terminal中，执行 `mvn package` 或 `mvn assembly:assembly`，将项目写入JAR包。

 说明

- 如果需要将第三方依赖写入JAR包，请使用 `mvn assembly:assembly`。
- 编译后的JAR包为 `RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT.jar`或`RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT-jar-with-dependencies.jar`（将第三方依赖写入JAR包）。

5. 实时计算Flink版作业引用JAR包。

i. 登录实时计算控制台。

ii. 在顶部菜单中，单击开发。

iii. 在左侧的导航栏中，单击资源引用。

iv. 在资源引用页签的右上角，单击新建资源。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 ? 说明 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击选择资源，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

v. 在资源引用页签中，鼠标悬停在对应作业的右侧的更多上。

vi. 在下拉列表中，选择引用。

vii. 在作业的编辑窗口顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

6. Blink SQL开发指南

6.1. 概述

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

Flink SQL开发指南主要包含以下内容：

- 数据存储

实时计算Flink版开发平台提供了实时计算Flink版上下游存储设备（例如，云数据库RDS版、数据总线DataHub和表格存储Tablestore等）管理功能。通过存储注册方式引入的上下存储设备，可以实现数据预览、数据抽样以及DDL生成功能。数据存储详情请参见[概述](#)。

 说明

- 如果共享模式集群需访问阿里云VPC网络下的存储资源，请参见 [VPC访问授权](#)。
- 如果实时计算Flink版访问的上下游存储存在白名单机制，请参见 [数据存储白名单配置](#)。

- 作业开发

作业开发章节为您介绍Flink SQL作业的开发、上线和启动流程，详情请参见 [开发](#)、[上线](#)和[启动](#)。

- 作业调试

作业调试章节为您介绍Flink SQL作业的调试功能，包括线上调试和本地调试，详情请参见 [本地调试](#)和[线上调试](#)。

- 作业运维

作业运维章节为您介绍运行信息、数据曲线和Failover等实时计算Flink版作业运维内容，详情请参见 [运行信息](#)、[数据曲线](#)和[Failover](#)。

- 监控报警

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

- 作业调优

作业调优章节为您介绍Flink SQL作业的调优功能，包括高性能Flink SQL优化技巧、AutoConf自动配置调优、AutoScale自动配置调优和手动配置调优，详情请参见[高性能Flink SQL优化技巧](#)、[AutoConf自动配置调优](#)、[AutoScale自动配置调优](#)和[手动配置调优](#)。

- Flink SQL

Flink SQL章节为您介绍Flink SQL语法，详情请参见 [概述](#)。

6.2. 数据存储

6.2.1. 概述

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


使用限制

独享模式集群仅能访问相同专有网络VPC、相同Region和相同安全组下的存储资源。如果需要访问其它VPC下的资源，请参见[如何访问跨VPC里的存储资源？](#)

数据存储的含义

数据存储有两层含义：

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

 **说明** 使用注册存储功能前，请先完成实时计算对存储设备的访问授权，授权方法请您参见[共享模式角色授权](#)和[独享模式角色授权](#)。

实时计算产品支持2种引用上下游存储资源的方式：明文方式和存储注册方式。


明文方式

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

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

存储注册方式

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

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

- 注册数据存储

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

- i. 登录[实时计算控制台](#)。

- ii. 单击页面顶部的开发。
- iii. 在开发界面，单击左侧导航栏中的数据存储。
- iv. 在数据存储页面的右上角，单击+注册与网络。
- v. 在注册数据存储与网络探测 窗口，完成对应储存设备的参数配置。

目前实时计算产品仅支持注册如下五种存储资源，具体方法请点击以下产品链接：

- [注册数据总线DataHub](#)
- [注册云原生数据仓库AnalyticDB MySQL版](#)
- [注册表格存储Tablestore](#)
- [注册云数据库RDS版](#)
- [注册日志服务SLS](#)

● 数据预览

对于已经注册的存储资源，实时计算提供数据预览功能，功能实现步骤如下：

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

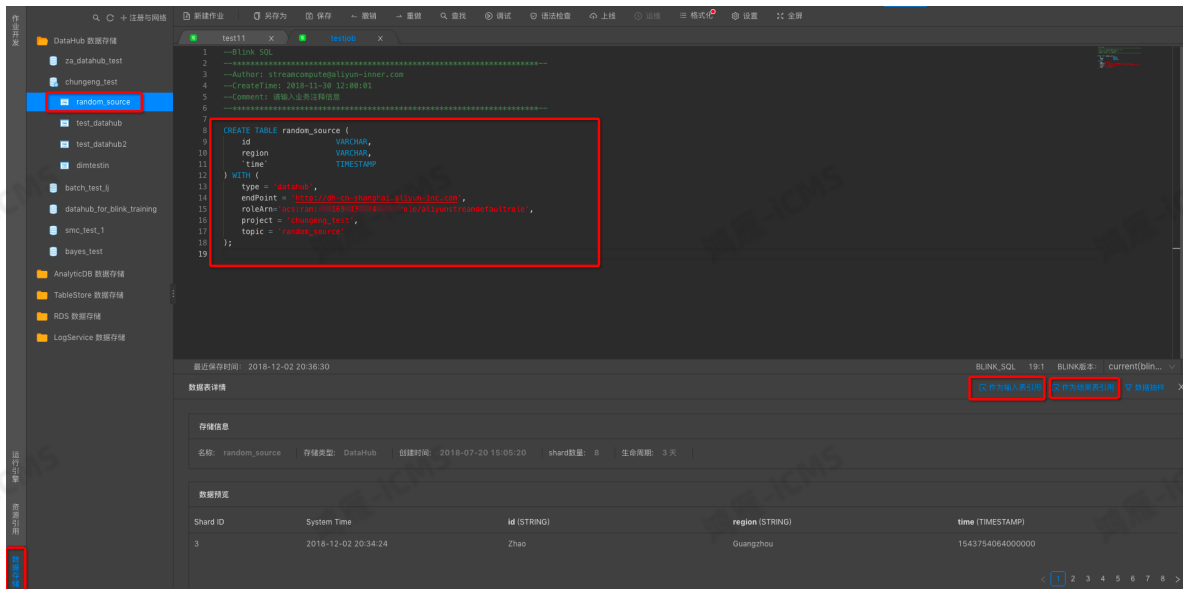


● 自动生成DDL

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

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

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

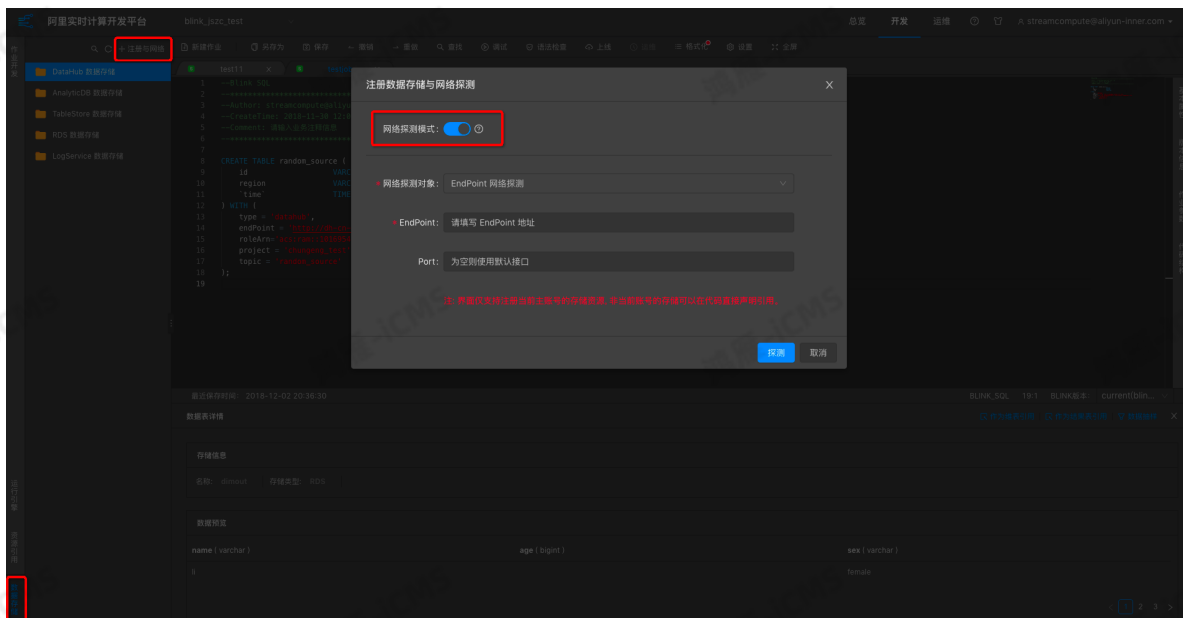


● 网络探测

说明 金融云杭州地域未安装云助手，网络探测功能暂时不支持。

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

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



6.2.2. 注册数据存储


6.2.2.1. 注册数据总线DataHub

本文为您介绍注册数据总线DataHub所需的参数信息。

什么是数据总线

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。实时计算Flink版使用DataHub作为流式数据存储源头或输出目的端。

注册存储


 **说明** 使用注册存储功能前，请先完成实时计算Flink版对存储设备的访问授权，授权方法请您参见[共享模式角色授权](#)和[独享模式角色授权](#)。

1. 登录[实时计算控制台](#)。
2. 在页面顶部，单击[开发](#)。
3. 在左侧导航栏，单击[数据存储](#)。
4. 在页面左上角，单击[注册与网络](#)。
5. 在注册数据存储与网络探测对话框，配置存储设备参数。
6. 单击注册。


参数说明

• EndPoint


填写DataHub的EndPoint。不同地域DataHub有不同的EndPoint，具体请参见[DataHub域名列表](#)。

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

- 独享模式：使用DataHub VPC EndPoint。例如，华东1（杭州）使用 `http://dh-cn-hangzhou-int-vpc.aliyuncs.com`。


 **说明** 独享模式集群所在的区域与DataHub所在区域要求一致。

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

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

• Project

填写DataHub的Project。

 **说明** 实时计算Flink版暂不支持跨属主的数据存储注册。例如，A用户拥有DataHub的ProjectA，但B用户希望在实时计算Flink版使用ProjectA。如果您需要使用跨属主的数据存储，可使用明文方式，详情请参见[明文方式](#)。

常见问题

Q: 注册数据存储失败的原因有哪些?

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

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

Q: 为什么数据抽样仅支持时间抽样?

A: DataHub定位是流数据存储，对外提供的接口仅有时间参数。因此，实时计算Flink版仅提供基于时间的抽样。

6.2.2.2. 注册云原生数据仓库AnalyticDB MySQL版

本文为您介绍注册云原生数据仓库AnalyticDB MySQL版数据存储所需的参数信息。

重要 本文档仅适用于云原生数据仓库AnalyticDB MySQL版 2.0版本。云原生数据仓库AnalyticDB MySQL版 3.0版本暂不支持结果表注册存储功能，请使用明文方式进行注册，详情请参见[创建云原生数据仓库AnalyticDB MySQL版3.0结果表](#)。

注册存储

说明 使用注册存储功能前，请先完成实时计算Flink版对存储设备的访问授权，授权方法请您参见[共享模式角色授权](#)和[独享模式角色授权](#)。

1. 登录[实时计算控制台](#)。
2. 在页面顶部，单击开发。
3. 在左侧导航栏，单击数据存储。
4. 在页面左上角，单击注册与网络。
5. 在注册数据存储与网络探测 对话框，配置存储设备参数。
6. 单击注册。

参数说明

参数名称	说明
------	----

参数名称	说明
URL	云原生数据仓库AnalyticDB MySQL版控制台链接信息。分为以下两种情况： <ul style="list-style-type: none">共享模式：请使用云原生数据仓库AnalyticDB MySQL版的经典网络URL。独享模式：请使用云原生数据仓库AnalyticDB MySQL版的VPC网络URL。 URL地址查询步骤如下： <ol style="list-style-type: none">登录AnalyticDB 控制台。单击对应的集群名称，进入基本信息页面。在网络信息中查看相应的连接地址。
Database	云原生数据仓库AnalyticDB MySQL版数据库名称，即云原生数据仓库AnalyticDB MySQL版实例名称。
AccessKey ID	阿里云账号的AccessKey ID。
AccessKey Secret	阿里云账号的AccessKey Secret。

 **说明** AccessKey ID和AccessKey Secret信息查询方法请参见 [如何查看AccessKey ID和AccessKey Secret信息?](#)


6.2.2.3. 注册表格存储Tablestore

本文为您介绍注册表格存储Tablestore所需的参数信息。

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NoSQL数据存储服务。表格存储能够提供海量结构化数据的存储服务 and 实时访问服务。Tablestore具备低延迟和运算复杂度低的特点，因此适合作为实时计算Flink版的数据存储维表或结果表。

注册存储

 **说明** 使用注册存储功能前，请先完成实时计算Flink版对存储设备的访问授权，授权方法请您参见 [共享模式角色授权](#) 和 [独享模式角色授权](#)。

1. 登录**实时计算控制台**。
2. 在页面顶部，单击**开发**。
3. 在左侧导航栏，单击**数据存储**。
4. 在页面左上角，单击**注册与网络**。
5. 在注册数据存储与网络探测对话框，配置存储设备参数。
6. 单击**注册**。

参数说明

- Endpoint

- 填写Tablestore的Endpoint。请在 [表格存储控制台](#) 查看Tablestore的Endpoint信息，填写Tablestore私网地址。具体步骤请参见 [服务地址](#)。
- Tablestore访问网络类型设置为 [允许任意网络访问](#)。操作步骤如下：
 - a. 登录 [表格存储控制台](#)。
 - b. 单击目标实例名称。
 - c. 在网络管理页签，单击更改。
 - d. 选择 [允许任意网络访问](#)。
 - e. 单击确定。

- 实例名称

填写实例名称。

6.2.2.4. 注册云数据库RDS版

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

什么是云数据库RDS版

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

② 说明

- 高频或高并发写入场景，不建议使用RDS作为实时计算Flink版作业的结果表，存在死锁风险。建议使用表格存储作为结果表（详情请参见 [创建表格存储Tablestore结果表](#)）。
- 云数据库RDS 8.0版本不支持注册存储功能，请使用 [明文方式](#)。

注册存储

② 说明 使用注册存储功能前，请先完成实时计算Flink版对存储设备的访问授权，授权方法请您参见 [共享模式角色授权](#) 和 [独享模式角色授权](#)。

1. 登录 [实时计算控制台](#)。
2. 在页面顶部，单击 [开发](#)。
3. 在左侧导航栏，单击 [数据存储](#)。
4. 在页面左上角，单击 [注册与网络](#)。
5. 在注册数据存储与网络探测 对话框，配置存储设备参数。
6. 单击 [注册](#)。

参数说明

② 说明 存储注册过程中系统会为RDS自动配置IP白名单。

参数	说明
数据存储类型	选择RDS 数据存储。
Region	选择RDS所在的地域。
Instance	填写RDS实例ID。 <p>ⓘ 说明 请填写实例ID，不是实例名称。</p>
DBName	填写RDS中Database的名称。 <p>ⓘ 说明 Database是RDS的数据库名称，不是实例名称。</p>
User Name	登录数据库的用户名。
Password	登录数据库的密码。

常见问题

Q: 存储注册过程中出现 操作错误：未在VPC中授权 报错如何处理？

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

6.2.2.5. 注册日志服务SLS

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

什么是日志服务SLS

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，实时计算Flink版能将其作为流式数据的输入。

注册存储

ⓘ 说明 使用注册存储功能前，请先完成实时计算Flink版对存储设备的访问授权，授权方法请您参见[共享模式角色授权](#)和[独享模式角色授权](#)。

1. 登录[实时计算控制台](#)。
2. 在页面顶部，单击开发。
3. 在左侧导航栏，单击数据存储。
4. 在页面左上角，单击注册与网络。
5. 在注册数据存储与网络探测 对话框，配置存储设备参数。
6. 单击注册。

参数说明

• Endpoint

填写日志服务的Endpoint。不同地域的日志服务Endpoint不同，详情请参见 [服务入口](#)。

说明

- Endpoint必须以 `http://` 开头，且不能以 `/` 结尾，例如 `http://cn-hangzhou-intranet.log.aliyuncs.com`。
- 实时计算Flink版和日志服务均处于阿里云内网，建议您填写经典网络或VPC网络Endpoint。不建议填写公网Endpoint，填写公网Endpoint可能导致性能问题和外网宽带的消耗。

• Project

填写Project名称。

说明 实时计算Flink版暂不支持跨属主的数据存储注册。例如，A用户拥有日志服务的Project A，但B用户希望在实时计算Flink版使用Project A。如果您需要使用跨属主的数据存储，可以使用明文方式，详情请参见 [明文方式](#)。

常见问题

Q: 注册数据存储失败的原因有哪些？

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

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

Q: 为什么数据抽样仅支持时间抽样？

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

6.2.3. VPC访问授权

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

背景

实时计算共享模式属于阿里云经典网络，如果需要访问阿里云VPC中的存储资源（目前仅支持VPC授权RDS），则需要完成VPC访问授权。

② 说明

- 独享模式集群处于阿里云VPC，无需进行VPC访问授权。
- 完成VPC访问授权后，实时计算访问对应的存储资源可能存在带宽受限等性能问题。不建议在实时计算共享模式下访问VPC中的存储资源。
- 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果有新购需求，推荐使用实时计算独享模式或Flink云原生模式。

VPC访问授权步骤

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

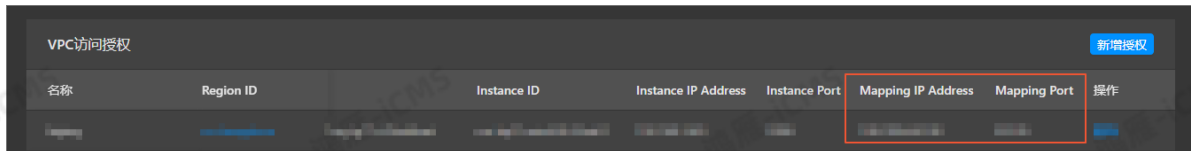
参数	说明
名称	VPC的名称。
地域	VPC中存储设备所在的地域。
VPC ID	VPC中存储设备的VPC网络ID。RDS VPC网络ID查看步骤如下： i. 登录 RDS管理控制台 。 ii. 在左侧导航栏，单击 实例列表 。 iii. 在页面左上角，选择实例所在地域。 iv. 单击目标实例ID。 v. 单击左侧导航栏中的 数据库连接 。 vi. 在数据库连接页面中，查看网络类型信息。 例如，RDS的VPC ID为 <code>vpc-bp1lysht98wrvl9n3****</code> 。
Instance ID	VPC中存储设备的实例ID。RDS中实例ID查看步骤如下： i. 登录 RDS管理控制台 。 ii. 在左侧导航栏，单击 实例列表 。 iii. 在页面左上角，选择实例所在地域。 iv. 单击目标实例ID，进入 基本信息 页面。 v. 在基本信息页面，查看RDS实例的ID。
Instance Port	VPC中存储设备的端口ID。RDS端口查询方法请参见 查看或修改内外网地址和端口 。

常见问题

Q：明文方式中，如何添加专有网络存储设备的URL参数？

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

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



6.2.4. 数据存储白名单配置

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

实时计算IP地址

实时计算分为共享模式和独享模式，2种模式的IP地址有所不同。

● 共享模式IP地址

说明 实时计算共享模式已于2019年12月24日正式下线，将不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。若有新购需求，推荐使用实时计算独享模式或Flink云原生模式。

○ 访问经典网络下的数据存储

可根据项目所在的区域，配置对应区域的IP地址。

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

- 访问VPC下的数据存储

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

- a. 登录[实时计算控制台](#)。
- b. 将鼠标悬停至页面右上角账号名称。
- c. 在下拉菜单中，单击项目管理。
- d. 在左侧导航栏中，单击VPC访问授权。
- e. 在VPC访问授权页面中，单击对应VPC访问授权的Region ID字段下的链接，进入白名单网段页面。
- f. 在白名单网段窗口查询VPC授权白名单网段。

- 独享模式IP地址

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

- i. 登录[实时计算控制台](#)。
- ii. 将鼠标悬停至页面右上角账号名称。
- iii. 在下拉菜单中，单击项目管理。
- iv. 单击左侧导航栏中的集群列表。
- v. 在集群列表页面，单击名称字段下目标集群名称。
- vi. 在集群信息窗口，查看集群的ENI信息。

配置RDS白名单

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

6.3. 作业开发

6.3.1. 开发

本文为您介绍实时计算Flink版作业开发流程以及语法检查、配置作业参数、配置项目参数、SQL辅助和SQL版本管理功能。

背景信息

① 说明

- 实时计算Flink版主要使用Flink SQL进行作业开发，Flink SQL开发手册请参见[概述](#)。
- 实时计算Flink版独享模式不支持归档保存已停止（含暂停）的作业运行日志。如果您需要查询已停止（含暂停）的作业运行日志，请将日志输出至您自定义的日志服务SLS或对象存储OSS中。详情请参见[自定义日志级别和下载路径](#)。

编写SQL代码

1. 登录[实时计算控制台](#)。
2. 在页面顶部，单击开发。

3. 在开发页面，单击页面顶部的新建作业。

4. 在新建作业界面，输入作业配置信息。

作业参数	说明
文件名称	作业的名称。  说明 作业名称在当前项目中必须保持唯一。
作业类型	<ul style="list-style-type: none"> 共享模式：仅支持FLINK_STREAM/SQL作业类型。 独享模式：支持FLINK_STREAM/DATASTREAM和FLINK_STREAM/SQL作业类型。
存储位置	在文件夹目录中，指定该作业的代码文件所属的文件夹。您还可以单击现有文件夹右侧的  图标，新建子文件夹。

5. 单击确定。

6. 在作业编辑页面，编写SQL代码。

说明

- 您可以在作业开发页面右侧的代码结构查看SQL代码结构。
- 建议您使用作业开发页面左侧的数据存储管理上下游存储，详情请参见[概述](#)。

配置作业参数

- 登录[实时计算控制台](#)。
- 在页面顶部，单击**开发**。
- 在左侧**作业开发**列表页面，单击目标作业名称。
- 在目标作业开发页面右侧，单击**作业参数**。
- 配置作业所需参数。

作业参数配置详情，请参见[作业参数调优](#)。

配置项目参数

作业参数针对单个作业生效，项目参数针对该项目下所有作业生效，开启项目参数后，会产生以下两种效果：

- 替换变量**：单击启动、调试或语法检查后，系统会替换SQL作业中的变量或Datastream作业中代码的变量。
- 参数下发**：项目级别系统参数会与作业参数、启动参数（仅Batch作业可以配置）进行Merge，参数优先级为：启动参数 > 作业参数 > 项目级别系统参数。Merge后作为最终参数下发到Blink作业。例如，作业参数配置和项目参数配置冲突，系统则以作业参数配置为准。

- 登录[实时计算控制台](#)。

2. 在页面顶部菜单栏上，鼠标悬停在用户头像后，单击 **项目管理**。
3. 在项目列表区域，单击目标项目名称。
4. 在页面顶部，单击 **开发**。
5. 在左侧作业开发列表页面，单击目标作业名称。
6. 配置项目参数生效。

该功能默认关闭（`disable.project.config=false`），您可以按照以下方式配置生效：

- SQL作业：在作业参数中配置 `enable.project.config=true`。
 - Datastream作业：在代码中配置 `enable.project.config=true`。
7. 在页面顶部，单击 **项目参数**。
 8. 配置项目所需参数。

项目参数仅支持SQL和Datastream两种作业类型，在配置项目级别系统参数时，您需要在项目级别配置参数前添加作业类型前缀，例如，`sql.name=LiLei` 或 `datastream.name=HanMeimei`。

启动语法检查

1. 登录 **实时计算控制台**。
2. 在页面顶部，单击 **开发**。
3. 在左侧作业开发列表页面，单击目标作业名称。
4. 在目标作业开发页面上方，单击 **语法检查**。

说明

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

SQL辅助

● Flink SQL语法检查

在您修改SQL后即可自动保存。保存操作可以触发SQL语法检查功能。语法校验出错后，将在作业 **开发** 页面提示出错行数、列数以及错误原因。

● Flink SQL智能提示

在您输入Flink SQL过程中，作业 **开发** 页面提供包括关键字、内置函数、表和字段智能记忆等提示功能。

● Flink SQL语法高亮显示

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

SQL版本管理

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

1. 登录 **实时计算控制台**。
2. 在页面顶部，单击 **开发**。
3. 在左侧作业开发列表页面，单击目标作业名称。

- 在目标作业开发页面右侧，单击版本信息。
- 单击操作 > 更多。
- 选择相应的版本管理功能。
 - 对比：查看最新代码和指定版本的差异。
 - 回滚：回滚到指定版本。
 - 删除：实时计算Flink版默认版本数上限为20。在版本数小于20时，您可以提交作业。如果当前的版本数为20，系统将不允许该作业的提交请求，并提示您删除部分旧版本作业。

 说明 当前版本数低于版本上限数后可以再次提交作业。

- 锁定：锁定当前作业版本。

 说明 解锁前无法提交新版本。

6.3.2. 上线

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

上线步骤

1. 资源配置

选择对应的资源配置方式。第1次启动建议使用系统默认配置。

 说明 实时计算Flink版支持手动资源配置和自动资源配置两种资源配置方式：


- 手动资源配置方法，请参见 [手动配置调优](#)。
- 自动资源配置方法根据实时计算Flink版本，分为以下两种方式：
 - 实时计算Flink版 3.0以上版本：AutoScale自动配置，详情请参见 [AutoScale自动配置调优](#)。
 - 实时计算Flink版 3.0及以下版本：AutoConf自动配置，详情请参见 [AutoConf自动配置调优](#)。

2. 数据检查

通过数据检查后，单击下一步。

3. 上线作业

单击上线。

 说明 作业上线后只是将作业提交至集群，并没有启动作业。启动步骤，详情请参见 [启动](#)。

6.3.3. 启动

完成作业开发和作业上线后，您可以在运维页面启动作业。

操作步骤

1. 登录实时计算控制台。
2. 单击页面顶部的 运维。
3. 在运维页面，单击目标作业操作列下的启动。
4. 在启动作业页面，单击指定数据读取数据时间（即指定启动位点）文本框。



5. 指定读取数据时间（启动位点），单击 确定，完成作业启动。

启动位点表示从数据源表中读取数据的时间点：

- 选择当前时间：表示从当前时间开始读取数据。
- 选择历史时间：表示从历史时间点开始读取数据，通常用于回追历史数据。

说明 作业启动完成后即可进入 [运行信息](#) 阶段。

6.3.4. 暂停

修改资源配置后，可以经过暂停和恢复的步骤使变更生效。本文为您介绍如何暂停作业。

背景信息

重要

- 只能对运行状态为 运行的作业进行暂停操作。
- 暂停操作不会清除任务状态，即如果有COUNT操作，作业 暂停 > 恢复后，COUNT会从上次成功Checkpoint的状态开始继续计算。
- 实时计算3.5.0以上版本才可以使用暂停（Checkpoint）功能，否则右上角会出现报错：发生错误系统错误：BLINK版本异常。错误原因：blink version >= blink-3.5 is required, instance blink-3.4.4。

操作步骤

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

② 说明 更多目录的暂停（Checkpoint）在进行暂停作业的同时，会主动触发一次Checkpoint，因此暂停（Checkpoint）所消耗的时间可能会比暂停的时间长。

6.3.5. 停止

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

🔔 重要

- 只能对运行状态为运行或启动中的作业进行停止操作。
- 停止操作会清除任务状态，即如果有COUNT操作，作业停止 > 启动后，COUNT从0开始计算。
- 实时计算3.5.0以上版本才可以使用停止（checkpoint）功能，否则右上角会出现报错：发生错误系统错误：BLINK版本异常。错误原因：blink version >= blink-3.5 is required, instance blink-3.4.4。

作业停止操作步骤如下：

1. 登录[实时计算控制台](#)。
2. 单击页面顶部的[运维](#)。
3. 在运维页面，单击目标作业操作列下的[停止](#)。

② 说明 更多目录的停止（checkpoint）功能与停止功能的唯一区别是：停止（checkpoint）在进行停止作业的同时，会主动触发一次Checkpoint，因此停止（checkpoint）作业所消耗的时间可能会比停止作业所消耗的时间稍长一些。但作业停止后依然会清除作业的状态，该功能在个别场景下会有其他作用，例如在上游存储为kafka时，系统触发一次Checkpoint会提交一次offset，确保提交到kafka服务端的offset和实际消费的数据量一致。

6.4. 作业调试

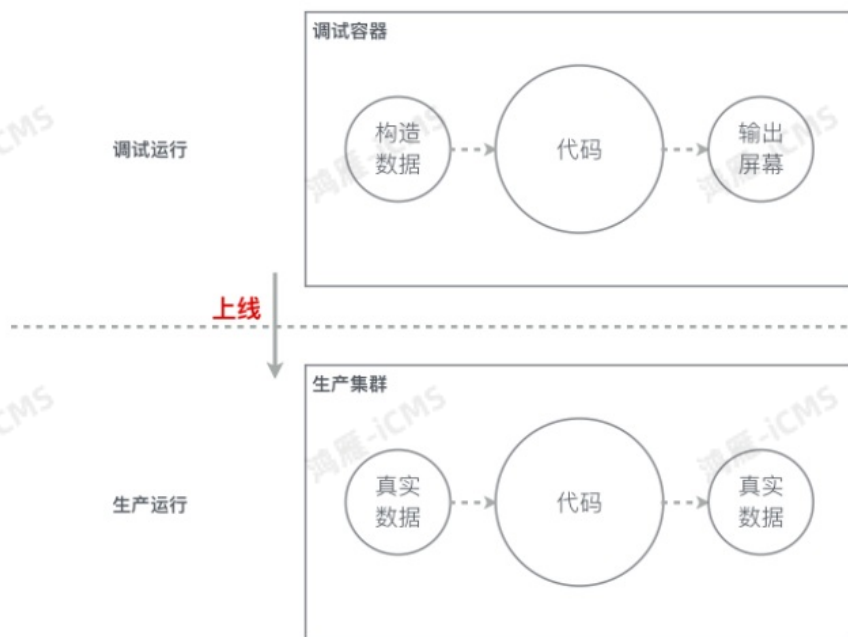
6.4.1. 本地调试

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

本地调试环境的特点

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

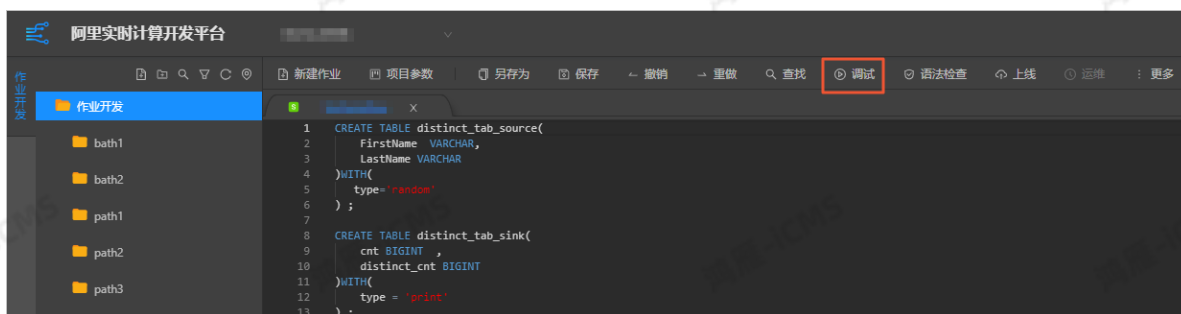
② 说明 实时计算调试模式无法检查出数据存储中的因为数据格式兼容性问题而导致的运行失败。例如，输出数据长度大于RDS建表最大值的问题。



本地调试步骤

说明 进行作业调试前，请您先完成作业的开发。作业开发流程请参见 [开发](#)。

1. 登录 [实时计算控制台](#)。
2. 在页面顶部，单击 [开发](#)。
3. 在目标作业开发编辑区域，单击页面顶部的 [调试](#)。



4. 在调试作业页面，输入测试数据。本地调试提供2种调试数据输入方式：

- 本地上传方式
 - a. 在数据预览区域，单击 [下载模板](#)。
 - b. 根据模板，编辑自定义调试数据。

说明 调试数据的默认分隔符为逗号 (,)，如果需要自定义分隔符请参见 [调试数据分隔符](#)。

- c. 在数据预览区域，单击 [上传](#)，上传自定义调试数据。
- 线上抽样方式

② 说明 使用顺序抽样线上数据功能前，请确保数据源在抽取时间段存在数据。

- 在数据预览区域，单击随机抽样线上数据或顺序抽样线上数据。
- 输入抽样配置信息。
- 单击确定。

② 说明 数据上传成功后，可在数据预览区域查看已上传数据。

- 单击确定，启动调试。
- 在作业编辑区域的底部，查看调试输出结果。

```
5 --Comment: 请输入业务注释信息
6 -----
7 insert into rds_ipplace
8 select t.`name`, w.place
9 from datahub_ipplace as t
10 join rds_dim for system_time as of proctime() as w
11 on t.place = w.place;
12
13
14
15
16
17
18
```

序号	操作	name	place
1	Insert	test01	beijing

调试数据分隔符

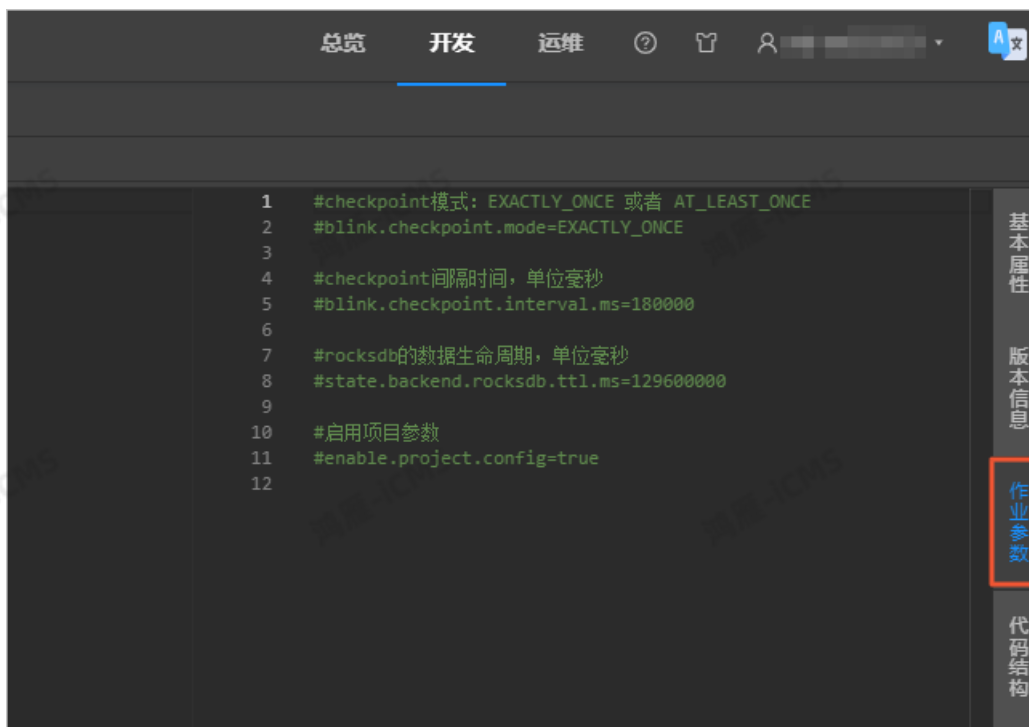
调试数据默认使用逗号(,)作为分隔符，如果输入的数据内容(例如JSON文件)中已存在逗号(,)，您需要自定义其它字符作为调试数据的分隔符，例如竖线(|)。

② 说明 实时计算仅支持使用单个英文字符(例如竖线(|))为分隔符。不支持使用字符串(例如aaa)作为分隔符。

调试数据分隔符的配置步骤如下：

② 说明 配置数据分隔符前，请您先完成作业的开发。作业开发流程请参见 [开发](#)。

- 在作业编辑区域，单击右侧的作业参数。



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

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

本地调试UDX的日志输出

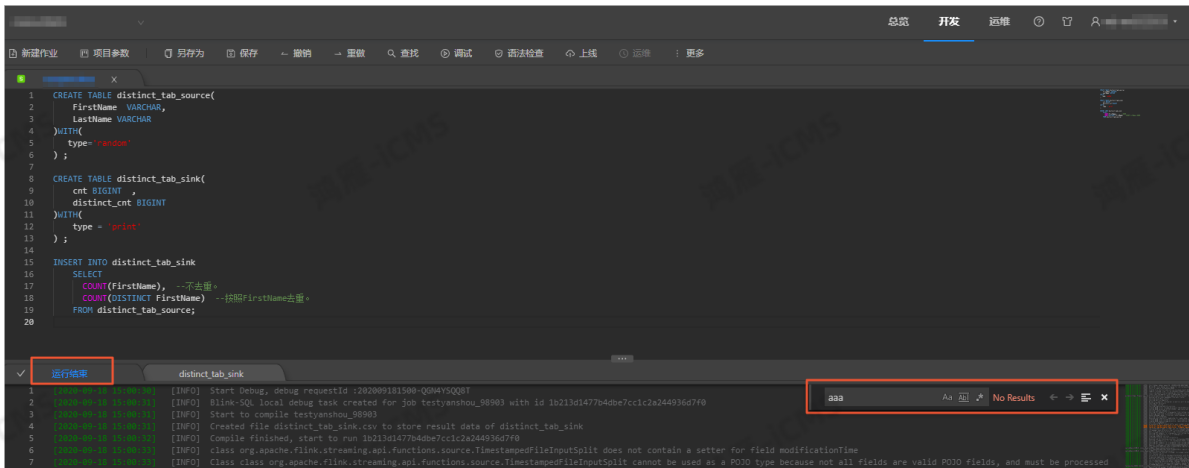
- 本地调式时打印UDX中日志

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

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

- 查看UDX的日志输出

调试结束后，在作业编辑区域底部的运行结束页面，可以查看UDX的日志输出。



说明 您可以通过快捷键 Ctrl+F 的方式搜索相应的日志信息。

6.4.2. 线上调试

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

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

线上调试步骤

1. 开发作业。作业开发详情，请参见 [开发](#)。
2. 更新数据存储DDL中的 type 参数。
 - 源表：type = 'random'
 - 结果表：type = 'print'
3. 上线作业。作业上线步骤请参见 [上线](#)。
4. 启动作业。作业启动步骤请参见 [启动](#)。

线上调试Connector

阿里实时计算平台提供以下2种线上调试功能的Connector：

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

Connector表参数

- Random表参数

参数	说明
type	必选，取值唯一且为random。

参数	说明
interval	可选，产生数据的时间间隔（单位为毫秒），默认值为500。

• Print表参数

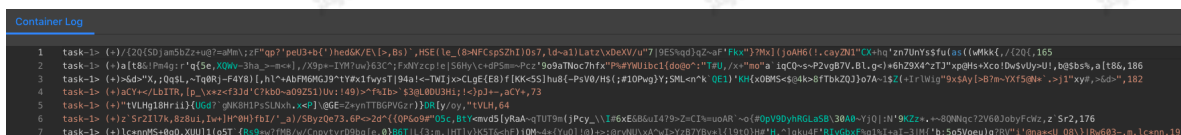
参数	说明
type	必选，取值唯一且为print。
ignoreWrite	可选，默认值为false。可选参数值如下： <ul style="list-style-type: none"> false：同时输出结果表和日志。 true：仅输出无数据的结果表，不输出日志数据。

线上调试示例

• 测试语句

```
CREATE TABLE random_source (
  instr      VARCHAR
) WITH (
  type = 'random'
);
CREATE TABLE print_sink(
  instr VARCHAR,
  substr VARCHAR,
  num INT
)with(
  type = 'print'
);
INSERT INTO print_sink
SELECT
  instr,
  SUBSTRING(instr,0,5) AS substr,
  CHAR_LENGTH(instr) AS num
FROM random_source
```

• 测试结果



线上调试结果查询步骤

② 说明 查询线上调试结果前，请先完成作业 **上线**和**启动**。

线上调试结果查询查看步骤如下：

1. 登录**实时计算控制台**。

- 单击顶部菜单栏中的 **运维**，进入运维页面。
- 单击作业名称字段下对应的作业，进入作业运维页面。
- 在Vertex拓扑区域，单击相应结果表节点。
- 单击SubTask List > 查看日志，进入日志查看窗口。
- 查看相应的日志。
 - Print结果表输出
单击taskmanager.out右侧的查看日志。
 - UDX日志输出
如果您使用了自定义函数UDX（UDX使用方法请参见 [概述](#)），可以使用以下两种方式查看日志：
 - system out/err方法
单击 taskmanager.out 或 taskmanager.err 右侧的查看日志。
 - SLF4J的Logger方法
单击 taskmanager.log 右侧的查看日志。

6.5. 作业运维

6.5.1. 登录作业运维平台

您可以在作业运维平台查看作业的各项信息，例如运行信息、数据曲线、Failover、属性参数等。本文为您介绍如何登录作业运维平台。

- 登录 [实时计算控制台](#)。
- 单击页面顶部的 **运维**。
- 在作业列表区域，单击作业名称下的目标作业名。

6.5.2. 运行信息

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

登录运行信息页面

- 登录作业运维页面。
 - 登录 [实时计算控制台](#)。
 - 单击页面顶部的 **运维**。
 - 在作业列表区域，单击作业名称下的目标作业名。
- 在作业运维页面，单击页面顶部的运行信息。

Task状态

Task状态为您显示作业各状态的数量。Task存在以下7种状态：

- 创建
- 运行

- 失败
- 完成
- 调度
- 取消中
- 已取消

作业瞬时值

Task状态下方为您展示作业的瞬时数值。

名称	作用描述
输入TPS	每秒从源端读到的Block数。对于日志服务而言，可以将多条数据打包到一个LogGroup读取。Block数反映的是从源端每秒读取LogGroup数量。
输入RPS	每秒读取数据源表的RPS数，单位为条/秒。
输出RPS	每秒写入数据结果的RPS数，单位为条/秒。
输入BPS	每秒读取数据源表的字节数，单位为Bytes/s。
消耗CU	单个Job当前的CU的消耗状况。
最近启动时间	Job的启动的时间。
运行时长	Job的启动后运行的时间。

Vertex拓扑

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

- 显示模式

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

- Task节点状态信息

视图模式Task参数信息



视图模式的节点框中为您显示了当前Task节点的信息，参数说明如下。

名称	信息描述
资源健康分	<p>通过资源健康分检查机制，反映作业的性能状况。资源健康分小于60分表明当前节点存在数据堆积，数据处理性能较差。</p> <p>? 说明 数据处理性能较差时，建议使用 AutoConf自动配置调优或手动配置调优提升性能。</p>
PARALLEL	并发量。
TPS	每秒读取上游数据的Block数。
DELAY	Task节点的业务延时。
IN_Q	Task节点的输入队列的百分比。
OUT_Q	Task节点的输出队列的百分比。

- 列表模式Task参数信息

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

名称	信息描述
Name	每个Task的详情信息。
Status	每个Task的运行的状态。
In Queue	Task节点的输入队列，单位是百分比。
Out Queue	Task节点的输出队列，单位是百分比。
Delay(ms)	Task节点的业务延时。
TPS	每秒读取上游的信息量。
Bytes Received	Task节点接收到的数据量。
Records Received	Task节点接收到的记录数。
Bytes Sent	Task节点发送的数据量。
Records Sent	Task节点发送的记录数。
Task	每个Task节点的并发的运行状态。

- Task节点线程信息

单击Task节点，在SubTasks页签，查看Task的线程列表。

Vertex信息

 说明 仅实时计算3.0.0以上版本支持以下功能。

- Vertex内部Operator信息

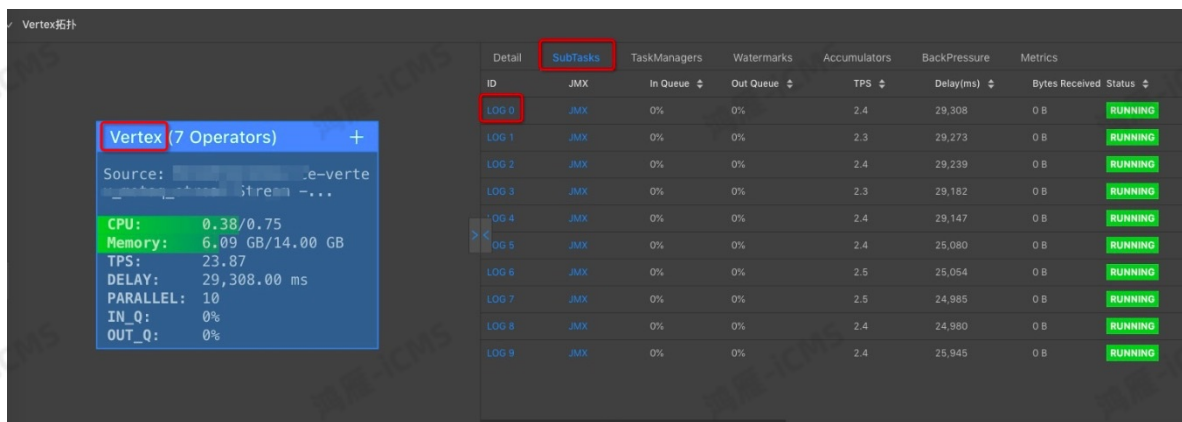
在Vertex拓扑区域，单击单个Vertex框右上角加号（+），可以显示Vertex内部Operator信息。

- 显示Vertex内部详情

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

- Vertex详情页面

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



6.5.3. 数据曲线

阿里云实时计算提供了当前作业的核心指标概览页面。您可以通过数据曲线对作业的运行情况进行一键式的诊断。

数据曲线示例如下。



说明

- 实时计算作业只有在运行状态下才显示数据曲线指标，暂停和停止状态均不显示数据曲线指标。
- 作业指标是实时计算系统异步后台采集，存在一定延迟。作业启动1分钟后，开始逐步采集各项指标，并在数据曲线显示。

登录数据曲线页面

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 单击页面顶部的[运维](#)。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。

2. 在作业运维页面，单击页面顶部的数据曲线。

Overview

• Failover

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

• 延时

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

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

② 说明

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

• 各Source的TPS数据输入

对实时计算作业所有的流式数据输入进行统计，记录每秒读取数据源表的Block的数，让您直观地了解数据存储TPS（Transactions Per Second）的情况。与TPS不同，RPS（Record Per Second）是读取数据源TPS的Block数解析后的数据，单位是条/秒。例如，如果日志服务1秒读取5个LogGroup，则TPS=5。如果每个LogGroup解析出来8个日志记录，则一共解析出40个日志记录，RPS=40。

• 各Sink的数据输出

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

• 各Source的RPS数据输入

统计实时计算作业所有的流式数据输入，让您直观地了解数据存储RPS（Record Per Second）情况。通常，在系统运维过程中，如果出现没有数据输出的情况，需要查看该值，判断数据源输入数据是否存在异常。

• 各Source的数据流量输入


统计实时计算作业所有的流式数据输入和每秒读取输入源表的流量，让您直观地了解数据流量BPS（Byte Per Second）情况。

• 各Source的脏数据

显示实时计算Source各时间段脏数据条数。

- **AutoScale的成功和失败数**

显示AutoScale成功执行和未成功执行的次数。

 **重要** 仅实时计算3.0.0以上版本支持此曲线。


- **AutoScale使用的CPU**

显示执行AutoScale时消耗的CPU数量。

 **重要** 仅实时计算3.0.0以上版本支持此曲线。

- **AutoScale使用的MEM**

显示执行AutoScale时消耗的内存量。

 **重要** 仅实时计算3.0.0以上版本支持此曲线。

Advanced View

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

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

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

数据曲线名称	说明
Checkpoint Duration	进行Checkpoint保存状态所花费的时间，单位为毫秒。
Checkpoint Size	进行Checkpoint所消耗的内存大小。
Checkpoint Alignment Time	当前节点进行Checkpoint操作，等待上游所有节点到达当前节点的时间。当Sink Operator（DAG流的终点）从其输入流接收到所有Barrier n时，它会向Checkpoint Coordinator确认Snapshot n已完成。当所有Sink都确认了这个快照，快照就被标识为完成。这个等待的时间就是CheckpointAlignmentTime。
Checkpoint Count	指定时间段内Checkpoint的数量。
Get	指定时间段内每个SubTask对ROCKSDB进行Get操作所花费的时间（最大值）。

数据曲线名称	说明
Put	指定时间段内每个SubTask对ROCKSDB进行Put操作所花费的时间（最大值）。
Seek	指定时间段内每个SubTask对ROCKSDB进行Seek操作所花费的时间（最大值）。
State Size	指定时间段内Job内部State存储大小（如果增量过快，Job是异常的）。
GMS GC Time	指定时间段内Job内部容器进行GC（Garbage Collection）花费的时间。
GMS GC Rate	指定时间段内Job内部容器进行GC（Garbage Collection）的频率。

WaterMark

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

Delay

Source SubTask 最大延迟 Top 15



表示每个Source并发的业务延时的时长。

Throughput

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

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

Queue

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

Tracing

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

Process

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

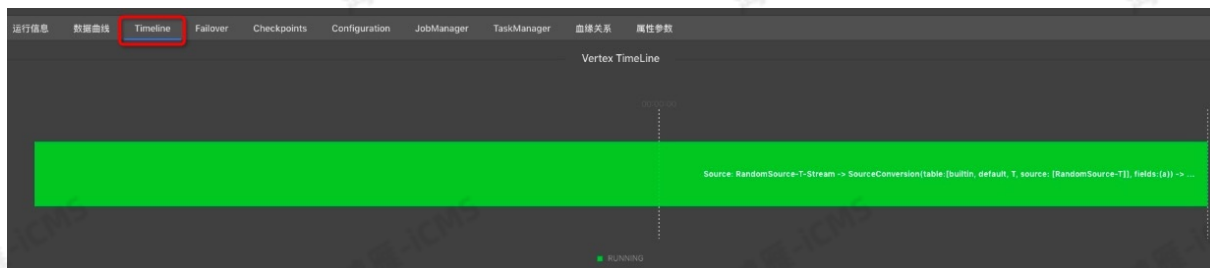
JVM

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

6.5.4. Timeline

Timeline页面为您显示各Vertex从启动位点到当前时间的运行状态。

 说明 仅实时计算3.0.0以上版本支持Timeline功能。



6.5.5. Failover

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

登录Failover页面

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 单击页面顶部的 **运维**。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的 **Failover**。

Latest FailOver

Latest FailOver为您展示作业当前的报错信息。

 说明 仅支持实时计算3.0以下版本。

FailOver History

FailOver History为您展示作业的历史报错信息。

 说明 仅支持实时计算3.0以下版本。

Root Exception

Root Exception为您展示作业当前的报错信息。

 说明 仅支持实时计算3.0及以上版本。

Exception History

Exception History为您展示作业的历史报错信息。

? 说明 仅支持实时计算3.0及以上版本。

6.5.6. Checkpoints

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

登录Checkpoints页面

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 单击页面顶部的 **运维**。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的 **Checkpoints**。

Overview

? 说明 该功能仅适用于实时计算3.0及以上版本。

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

History

? 说明 该功能仅适用于实时计算3.0及以上版本。

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

Summary

? 说明 该功能仅适用于实时计算3.0及以上版本。

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

Configuration

? 说明 该功能仅适用于实时计算3.0及以上版本。

Configuration为您展示Checkpoint的配置信息。


Completed Checkpoints

? 说明 该功能仅适用于实时计算3.0以下版本。

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

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

Task Latest Completed Checkpoint

 说明 该功能仅适用于实时计算3.0以下版本。

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

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

6.5.7. JobManager

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

登录JobManager页面

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 单击页面顶部的 **运维**。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的**JobManager**。

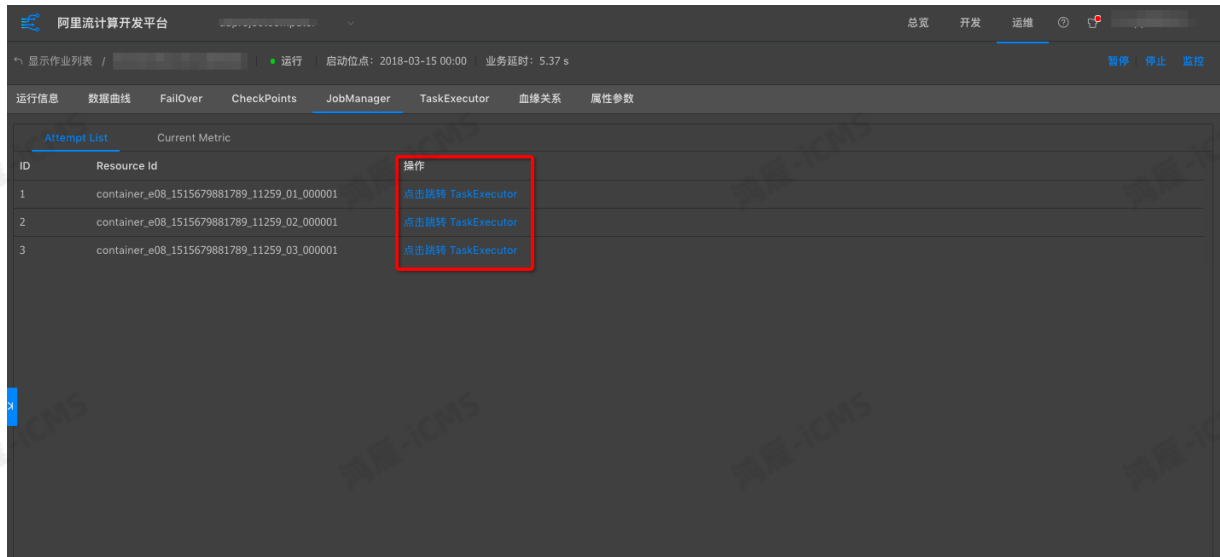
JobManager在集群启动中的作用

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

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

JobManager参数信息

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



6.5.8. TaskExecutor

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

重要 本文档仅适用于实时计算3.0以下版本。

背景

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

登录TaskExecutor界面

1. 登录[实时计算控制台](#)。
2. 单击界面顶部的 **运维**。
3. 在作业列表区域，单击作业名称下的目标作业名。
4. 在作业运维界面，单击顶部的TaskExecutor。

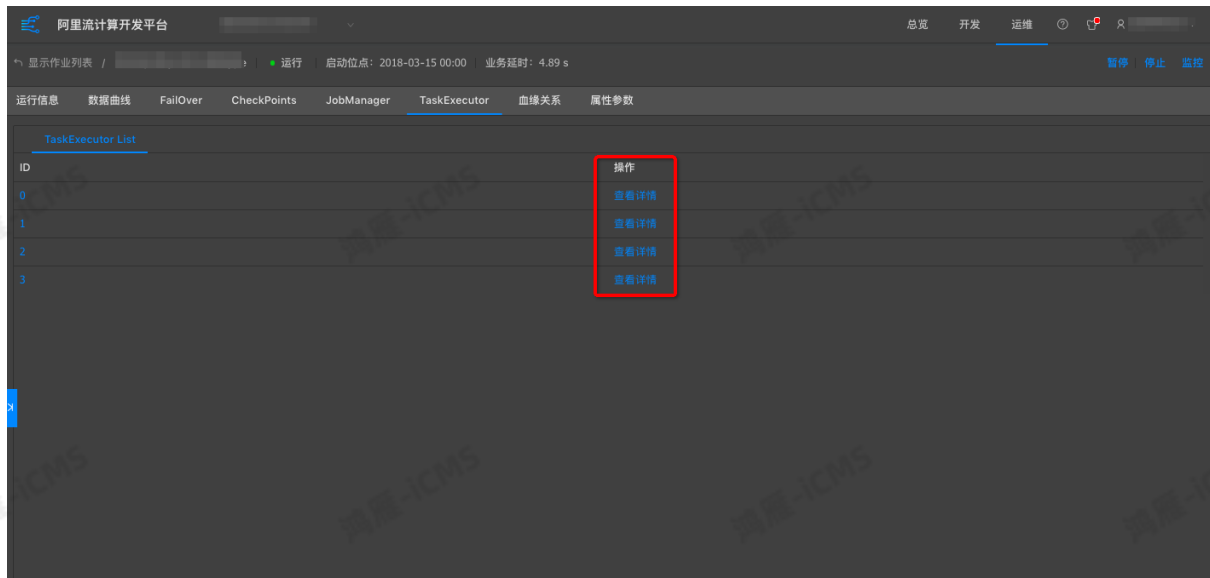
TaskExecutor在集群启动中的作用

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

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

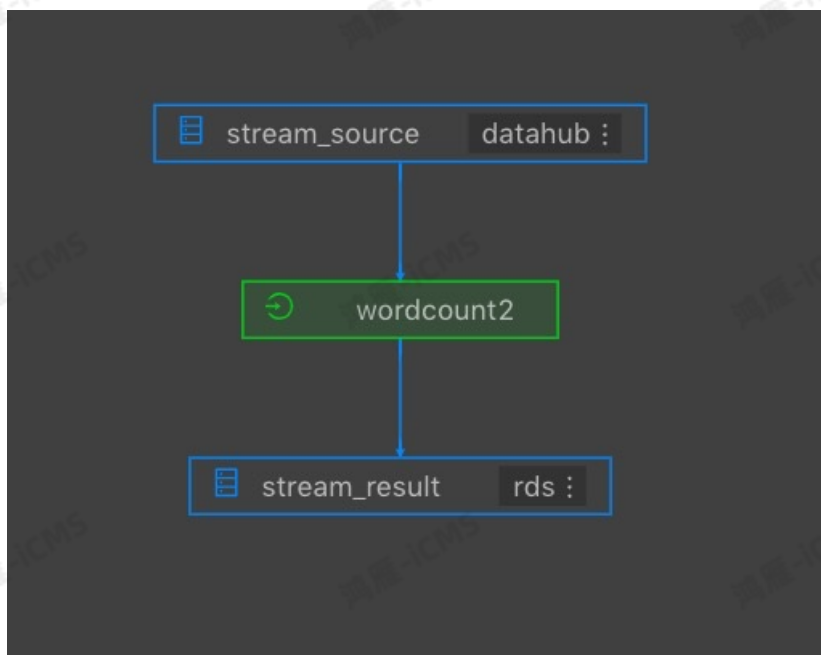
TaskExecutor界面

TaskExecutor界面为您提供Task列表以及Task详情的接口。



6.5.9. 血缘关系

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



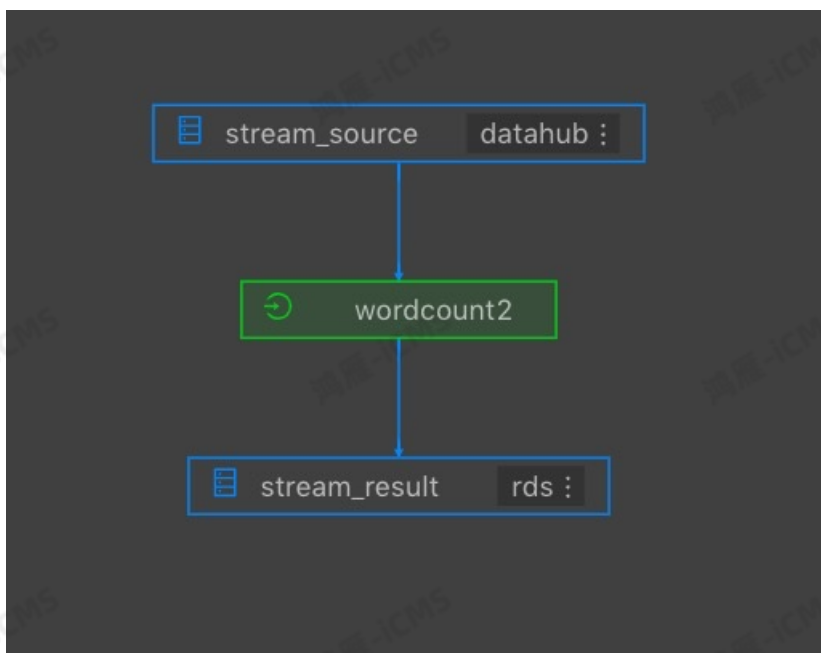
登录血缘关系页面

1. 登录作业运维页面。
 - i. 登录**实时计算控制台**。
 - ii. 单击页面顶部的**运维**。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的**血缘关系**。

数据抽样

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

1. 在作业上下游中单击表名。



2. 数据抽样页面，单击下方的抽样。

6.5.10. 属性参数

属性参数提供了当前作业的详情信息，包括当前运行信息以及历史运行记录。

登录属性参数页面

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 单击页面顶部的 **运维**。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 在作业运维页面，单击页面顶部的属性参数。

作业代码

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

资源配置

- 资源配置页面为您展示Job运行中所涉及资源的配置信息，如CPU、MEM、并发数等。
- 开启AutoScale功能后，可在资源配置页面中查询AutoScale迭代的历史详情。

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

作业属性

运行属性为您展示Job的基本信息。

运行参数

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

历史记录

历史记录为您展示作业的操作信息，包括操作人员、启动位点、终止时间等。

作业参数

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

6.5.11. 作业诊断

实时计算提供作业诊断功能方便您快速的排查作业问题。

? 说明 仅运行的作业支持诊断功能。

作业诊断步骤

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

诊断指标

- Failover
 - 作业Failover事件：检查作业在最近30分钟是否出现Failover。
 - 作业管理节点Failover事件：监控AM是否出现Failover。
- Yarn调度事件为您显示Yarn的检查结果。无异常时，返回结果如下图。



- 机器所在机器环境 为您显示机器所在机器环境的检查结果。无异常时，返回结果如下图。



- **Blink Metric:**

检查并获取作业延时时长。出现延时时，会显示反压的节点。

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

6.6. 作业调优

6.6.1. 概述

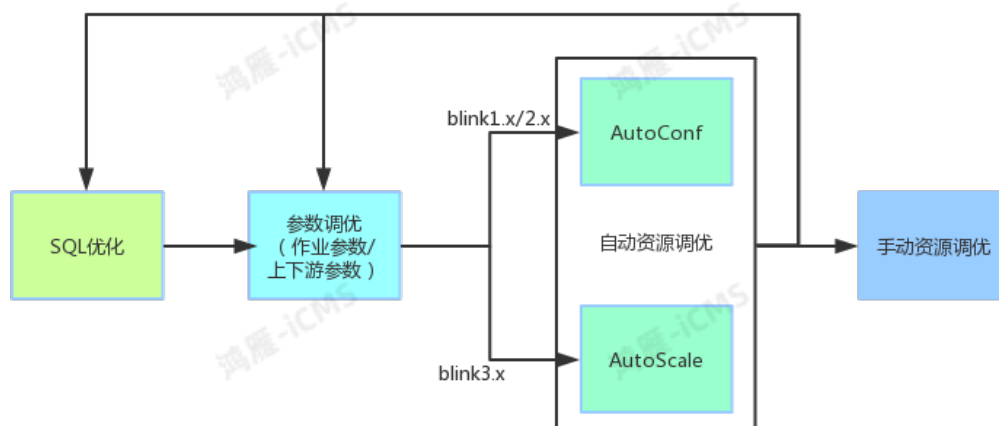
实时计算作业开发过程中，完成了业务逻辑实现、作业上线和启动运行后，为了满足实时计算作业的性能需求，还需对作业进行调优。

作业调优的目的和衡量标准

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

作业调优步骤

建议的作业调优顺序和步骤如下图。



1. SQL优化

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

2. 参数调优

○ 作业参数调优

选择底层优化策略，例如设置minibatch优化State访问策略等，具体请参见[作业参数调优](#)。

○ 上下游存储参数调优

优化上下游存储读写，例如，攒批读写提升吞吐和设置Cache策略提升维表JOIN效率等，具体请参见[上下游参数调优](#)。

3. 自动资源调优

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

○ Blink 1.x（自Blink 1.6.4开始）和Blink 2.x版本请参见[AutoConf自动配置调优](#)。

○ Blink 3.x版本请参见[AutoScale自动配置调优](#)。

4. 手动资源调优或再次进行调优

○ 手动资源调优

自动配置调优无法满足需求的情况下，您可以针对性的进行[手动配置调优](#)。

说明 Blink 3.x版本可参见问题[如何对实时计算Flink版3.0以上版本的作业进行反压检测？](#)协助您判断性能反压点。

○ 再次进行调优

如果一次调优后的结果不能满足您的业务需求，可按照步骤1~4，再次进行调优。

6.6.2. 高性能Flink SQL优化技巧

本文为您介绍提升性能的Flink SQL推荐写法、配置及函数。

Group Aggregate优化技巧

- 开启MicroBatch或MiniBatch（提升吞吐）

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

MiniBatch主要依靠在每个Task上注册的Timer线程来触发微批，需要消耗一定的线程调度性能。

MicroBatch是MiniBatch的升级版，主要基于事件消息来触发微批，事件消息会按您指定的时间间隔在源头插入。MicroBatch在元素序列化效率、反压表现、吞吐和延迟性能上都要优于MiniBatch。

- 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。通常对于聚合的场景，微批处理可以显著的提升系统性能，建议开启。

 说明 MicroBatch模式也能解决两级聚合数据抖动问题。

- 开启方式

MicroBatch和MiniBatch默认关闭，开启方式如下。

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

- 开启LocalGlobal（解决常见数据热点问题）

LocalGlobal优化将原先的Aggregate分成Local+Global两阶段聚合，即MapReduce模型中的Combine+Reduce处理模式。第一阶段在上游节点本地攒一批数据进行聚合（localAgg），并输出这次微批的增量值（Accumulator）。第二阶段再将收到的Accumulator合并（Merge），得到最终的结果（GlobalAgg）。

LocalGlobal本质上能够靠LocalAgg的聚合筛除部分倾斜数据，从而降低GlobalAgg的热点，提升性能。

- 适用场景

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

 说明 开启LocalGlobal需要UDAF实现 Merge 方法。

- 开启方式

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

- 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）

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。

之前，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）。自 2.2.0 版本开始，实时计算提供了COUNT DISTINCT自动打散，即PartialFinal优化，您无需自行改写为两层聚合。

- 适用场景

使用COUNT DISTINCT，但无法满足聚合节点性能要求。

② 说明

- 不能在包含UDAF的Flink SQL中使用PartialFinal优化方法。
- 数据量不大的情况下，不建议使用PartialFinal优化方法。PartialFinal优化会自动打散成两层聚合，引入额外的网络Shuffle，在数据量不大的情况下，浪费资源。

- 开启方式

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

- 判断是否生效

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

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

② 说明 仅实时计算2.2.2及以上版本支持AGG WITH FILTER语法。

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

建议您将AGG WITH CASE WHEN的语法都替换成AGG WITH FILTER的语法，尤其是对同一个字段上计算不同条件下的COUNT DISTINCT结果，性能提升很大。

- 原始写法

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

- 优化写法

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

TopN优化技巧

- TopN算法

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

- UpdateFastRank：最优算法。

需要具备2个条件：

- 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（仅实时计算2.2.2及以上版本支持）。

如果您要获取到优化Plan，则您需要在`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
) a
)
WHERE rownum <= 100;
```

- UnaryUpdateRank：仅次于UpdateFastRank的算法。需要具备1个条件：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，则可进行UnaryUpdateRank或UpdateFastRank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[TopN语句](#)。

- 增加TopN的Cache大小

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

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

例如，Top100配置缓存10000条，并发50，当您的PartitionBy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PartitionKey维度特别大时，可以适当加大TopN的CacheSize，相对应的也建议适当加大TopN节点的Heap Memory（请参见[手动配置调优](#)）。

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

- PartitionBy的字段中要有时间类字段

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

高效去重方案

 说明 仅Blink 3.2.1版本支持高效去重方案。

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

- 语法

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

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

参数	说明
ROW_NUMBER()	计算行号的OVER窗口函数。行号从1开始计算。
PARTITION BY col1[, col2..]	可选。指定分区的列，即去重的KEYS。
ORDER BY timeAttributeCol [asc desc]	指定排序的列，必须是一个 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）。

参数	说明
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。

说明

- 当排序字段是Proctime列时，Flink就会按照系统时间去重，其每次运行的结果是不确定的。
- 当排序字段是Rowtime列时，Flink就会按照业务时间去重，其每次运行的结果是确定的。

- ii. 对排名进行过滤，只取第一条，达到了去重的目的。

说明 排序方向可以是按照时间列的顺序，也可以是倒序：

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

● **Deduplicate Keep FirstRow**

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

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

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

● **Deduplicate Keep LastRow**

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

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

② 说明 以上示例是将T表按照b和d字段进行去重，并按照业务时间保留最后一条数据。Rowtime在这里是源表T中的一个具有Event Time属性的字段。

高效的内置函数

● 使用内置函数替换自定义函数

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

- 优化数据序列化和反序列化的耗时。
- 新增直接对字节单位进行操作的功能。

● KEY VALUE函数使用单字符的分隔符

KEY VALUE 的签名：`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 的值，而不会将整个content做切分。性能约提升30%。

● 多KEY VALUE场景使用MULTI_KEYVALUE

② 说明 仅实时计算 2.2.2 及以上版本支持MULTI_KEYVALUE。

在Query中对同一个Content进行大量KEY VALUE的操作，会对性能产生很大影响。例如Content中包含10个Key-Value对，如果您希望把10个Value的值都取出来作为字段，您就需要写10个KEY VALUE函数，则系统就会对Content进行10次解析，导致性能降低。

在这种情况下，建议您使用 **MULTI_KEYVALUE**表值函数，该函数可以对Content只进行一次Split解析，性能约能提升50%~100%。

● LIKE操作注意事项

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

● 慎用正则函数（REGEXP）

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

- **REGEXP**
- **REGEXP_EXTRACT**
- **REGEXP_REPLACE**

网络传输的优化

目前常见的Partitioner策略包括：

1. KeyGroup/Hash: 根据指定的Key分配。
2. Rebalance: 轮询分配给各个Channel。
3. Dynamic-Rebalance: 根据下游负载情况动态选择分配给负载较低的Channel。
4. Forward: 未Chain一起时，同Rebalance。Chain一起时是一对一分配。
5. Rescale: 上游与下游一对多或多对一。

- 使用Dynamic-Rebalance替代Rebalance

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

- 使用Rescale替代Rebalance

 说明 仅实时计算2.2.2及以上版本支持Rescale。

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

推荐的优化配置方案

综上所述，作业建议使用如下的推荐配置。

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

6.6.3. AutoConf自动配置调优

为了增加用户的体验度，阿里云为您提供自动配置调优（AutoConf）功能。

 说明 AutoConf自动配置调优功能支持blink 1.0和2.0版本。

背景及功能范围

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

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

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

操作

- 新作业

- i. 上线作业

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



- b. 选择资源配置方式。第一次不需要指定CU数直接使用系统默认配置。

- **智能CU配置**：指定使用CU AutoConf算法会基于系统默认配置，生成CU数，进行优化资源配置。如果是第一次运行，算法会根据经验值生成一份初始配置。建议作业运行了5~10分钟以上，确认Source RPS等Metrics稳定2~3分钟后，再使用智能配置，重复3~5次才能调优出最佳的配置。
 - **使用上次资源配置（手动资源配置）**：即使用最近一次保存的资源配置。如果上一次是智能配置的，就使用上一次智能配置的结果。如果上一次是手工配置的，就使用上次手工配置的结果。

- ii. 使用默认配置启动作业

- a. 使用默认配置启动作业，出现如下的界面。



b. 启动作业。



示例如下。第一次默认配置生成的资源配置为71个CU。

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

Task状态	创建	运行	失败	完成	调度	取消中	已取消		
计算耗时	输入TPS	输入RPS	输出RPS	输入BPS	申请CPU	使用MEM	申请MEM	启动时间	运行时长
499.94 ms	95.95 Block/s	1781.57 条/s	6.57 条/s	6322753 B/s	71 Cores	105976.5 MB	259072 MB	2018-01-13 14:38:41	3天2小时48分钟15秒

iii. 使用智能配置启动作业

a. 资源调优

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

■ CU的最小配置

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

■ CU增加数量

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

■ 可多次调优

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

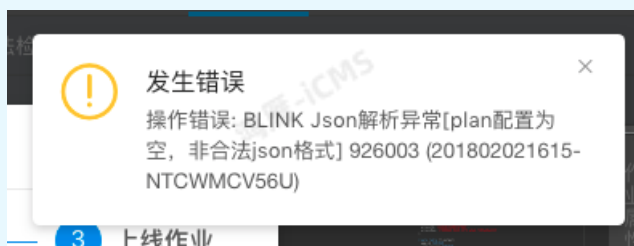
上线新版本

x

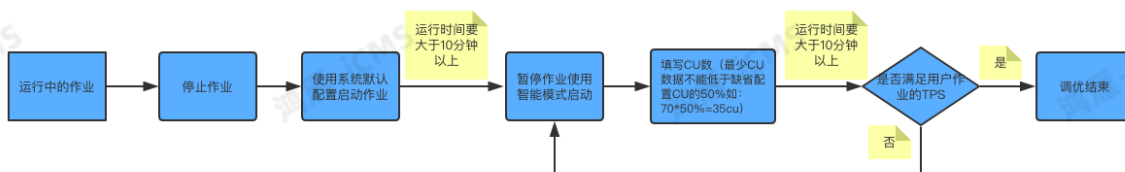
b. 调优后的结果如下图。

Task状态	创建: 0	运行: 1	失败: 0	完成: 0	调度: 0	取消中: 0	已取消: 0
输入TPS	输入RPS	输出RPS	输入BPS	消耗CU	最近启动时间	运行时长	
2 Blocks/s	-条/s	-条/s	- Bytes/s	0.69 CU	2020-08-26 20:14:30	12天 22小时 16分钟 11秒	

① 说明 如果是新任务，请不要选择使用上次资源配置，否则会报错。



- 已存在作业
 - 调优流程示意图



① 说明

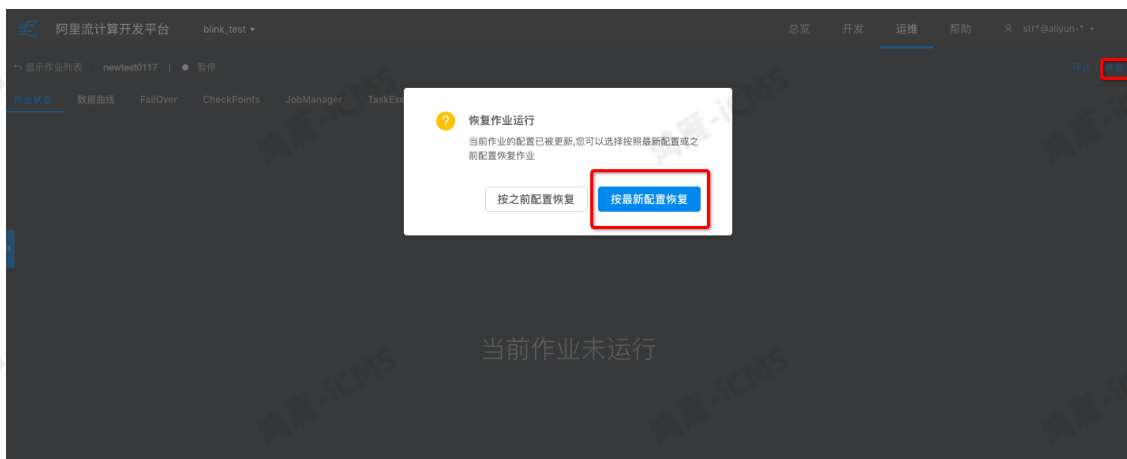
- 已存在的作业在进行调优前，您一定要检查是否有状态的计算。因为在调优的过程中可能会清除之前作业保存的状态，请您谨慎操作。
- 当您的作业有改动时（例如，您对作业的SQL有修改，或更改了实时计算版本），自动配置调优功能不保证能按照之前的运行信息进行调优。原因是这些改动会导致拓扑信息变化，造成数据曲线无法对应和状态无法复用等问题。自动配置调优功能无法根据运行历史信息作出调优判断。此时再使用自动配置调优会报异常。这时您需要将改动后的任务当做新任务，重新进行操作。

○ 调优流程

a. 暂停任务。



b. 重复新作业的调优步骤，使用最新的配置启动作业。



常见问题

以下几点可能会影响自动调优的准确性：

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

调优建议

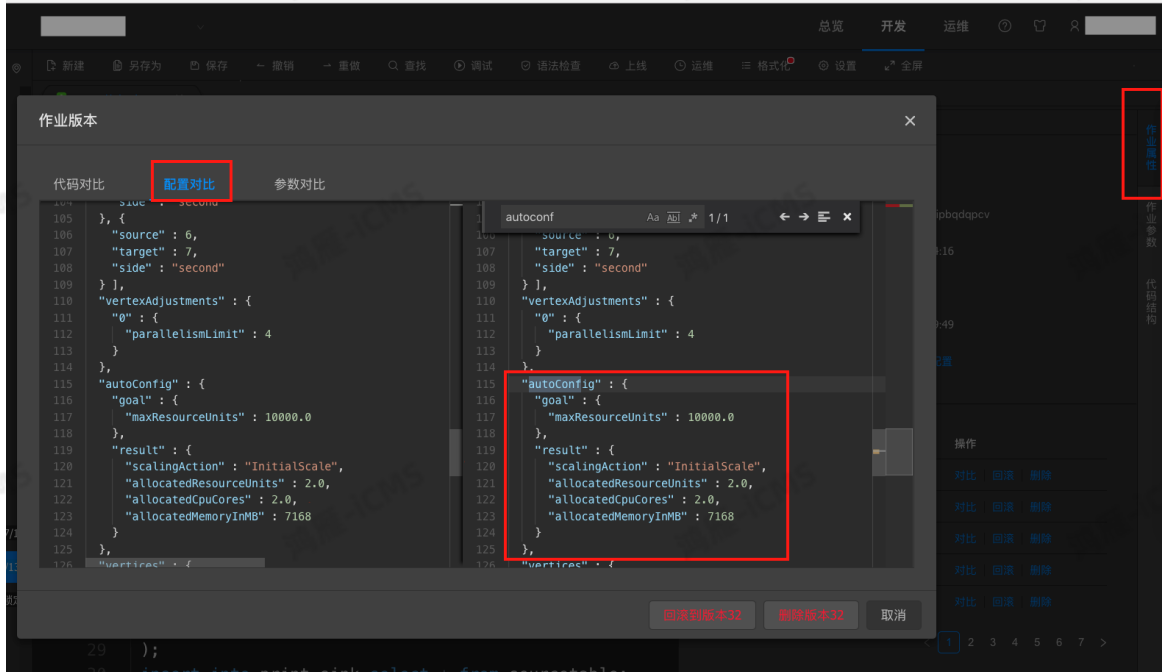
- 每次触发智能配置前任务稳定运行超过10分钟。这样有利于AutoConf准确搜集的任务运行时的指标信息。
- AutoConf可能需要3~5次迭代才能见效。
- 使用AutoConf时，您可以设置让任务回追数据甚至造成反压。这样会更有利于快速体现调优成功。

如何判断自动配置调优功能生效或出现问题？

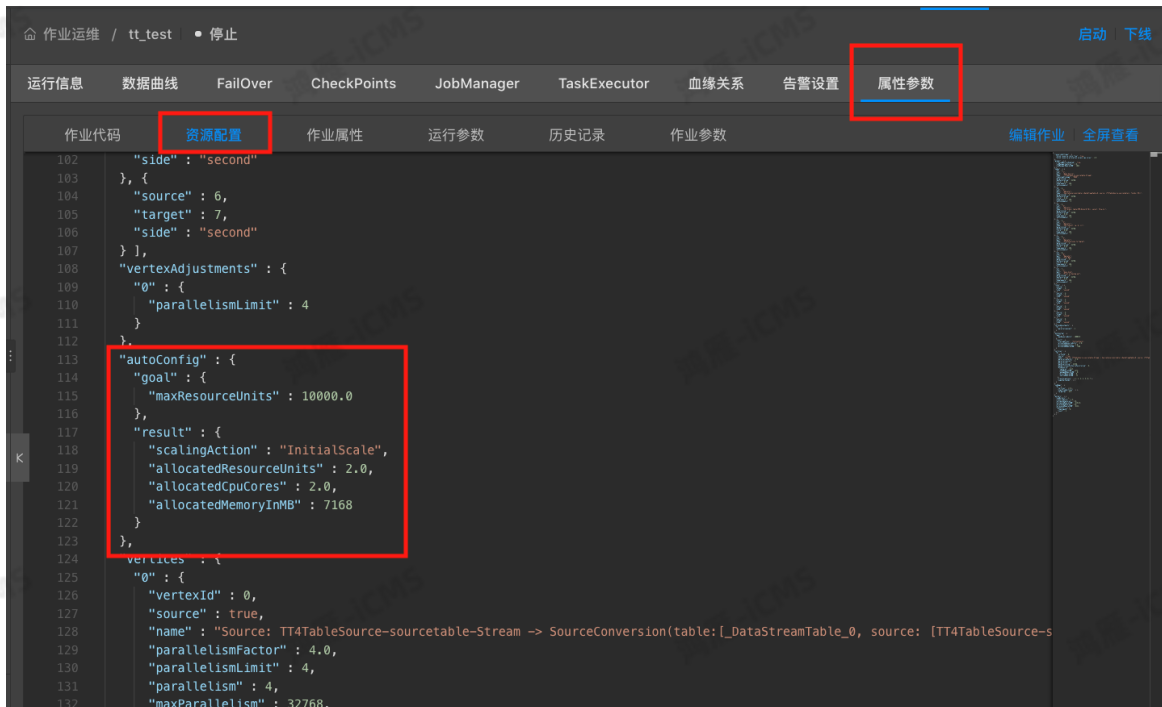
自动配置调优功能通过JSON配置文件与实时计算交互。您在调优后，可以通过查看JSON配置文件了解自动配置调优功能的运行情况。

- 查看JSON配置文件的两种方式

- i. 通过作业编辑界面，如下图。



- ii. 通过作业运维界面，如下图。



- JSON配置解释

```
"autoconfig": {
  "goal": { // AutoConf 目标
    "maxResourceUnits": 10000.0, // 单个Blink作业最大可用CU数，不能修改，查看时可忽略。
    "targetResourceUnits": 20.0 // 用户指定CU数。用户指定为20CU。
  },
  "result": { // AutoConf 结果。这里很重要
    "scalingAction": "ScaleToTargetResource", // AutoConf的运行行动 *
    "allocatedResourceUnits": 18.5, // AutoConf分配的总资源。
    "allocatedCpuCores": 18.5, // AutoConf分配的总CPU。
    "allocatedMemoryInMB": 40960 // AutoConf分配的总内存。
    "messages": "xxxx" // 很重要。 *
  }
}
```

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

异常信息问题

`IllegalStateException`异常

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

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

```
java.lang.IllegalStateException: Could not initialize keyed state backend.  
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initKeyedState(AbstractStreamOperator.java:687)  
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initializeState(AbstractStreamOperator.java:275)  
    at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeOperators(StreamTask.java:870)  
    at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeState(StreamTask.java:856)  
    at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:292)  
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)  
    at java.lang.Thread.run(Thread.java:834)  
Caused by: org.apache.flink.api.common.typeutils.SerializationException: Cannot serialize/deserialize the object.  
    at com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSecondaryState.deserializeStateEntry(AbstractRocksDBRawSecondaryState.java:167)  
    at com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRestoreOperation.restoreRawStateData(RocksDBIncrementalRestoreOperation.java:425)  
    at com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRestoreOperation.restore(RocksDBIncrementalRestoreOperation.java:119)  
    at com.alibaba.blink.contrib.streaming.state.RocksDBKeyedStateBackend.restore(RocksDBKeyedStateBackend.java:216)  
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.createKeyedStateBackend(AbstractStreamOperator.java:986)  
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initKeyedState(AbstractStreamOperator.java:675)  
    ... 6 more  
Caused by: java.io.EOFException  
    at java.io.DataInputStream.readUnsignedByte(DataInputStream.java:290)  
    at org.apache.flink.types.StringValue.readString(StringValue.java:770)  
    at org.apache.flink.api.common.typeutils.base.StringSerializer.deserialize(StringSerializer.java:69)  
    at org.apache.flink.api.common.typeutils.base.StringSerializer.deserialize(StringSerializer.java:28)  
    at org.apache.flink.api.java.typeutils.runtime.RowSerializer.deserialize(RowSerializer.java:169)  
    at org.apache.flink.api.java.typeutils.runtime.RowSerializer.deserialize(RowSerializer.java:38)  
    at com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSecondaryState.deserializeStateEntry(AbstractRocksDBRawSecondaryState.java:162)  
    ... 11 more
```

6.6.4. AutoScale自动配置调优

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

说明

- AutoScale自动配置调优功能仅支持实时计算3.0及以上版本。
- 升级实时计算3.0版本前，需要删除所有低于3.0版本的PlanJson，并重新获取配置资源。详情请参见[如何重新获取配置资源？](#)

启动AutoScale自动配置调优

作业上线的过程中即可完成AutoScale自动配置调优的开启。

1. 登录作业编辑页面。
 - i. 登录**实时计算控制台**。
 - ii. 单击页面顶部的**开发**。
 - iii. 在作业开发区域，双击目标文件夹或目标作业名，进入作业编辑页面。
2. 单击页面顶部菜单栏中的**上线**，进入**上线新版本**页面。
3. 在**初始资源**页面，选择初始资源类型，单击**下一步**。



- **上次自动调优**：使用上次AutoScale的Planjson配置启动作业。满足以下所有条件后即可选择**上次自动调优**功能。
 - 作业已开启AutoScale并完成迭代。
 - 作业为暂停状态。

- 在资源配置页面获取AutoScale配置信息。



- 系统分配：新增作业或者作业逻辑没有修改且兼容实时计算版本时，可以选择 系统分配。
 - 手动资源配置：使用手动生成的资源配置方法。在手动重新生成或者修改AutoScale时，需要选择 手动资源配置。
4. 在数据检查，通过检查后，单击下一步。
 5. 在资源配置，配置自动调优信息后，单击下一步。

参数	说明
自动调优	选择开启，开启自动调优功能。
planjson最大CU数	作业可用的资源上限，单位为CU，1CU=1 Core + 4G MEM。最大CU数需小于项目可用CU数。
调优策略	目前调优策略仅支持数据滞留时间。系统会根据选择的调优策略和期望值对作业自动调优。
期望值	数据滞留时间阈值。当数据源的数据滞留时间超过该值时，触发AutoScale，优化作业并发数。

② 说明 例如，调优策略设置数据滞留时间的期望值为5（秒）。如果此时作业的数据滞留时间大于5秒，则系统会不断进行自动调优，直至数据滞留时间小于5秒。

6. 在上线作业，单击上线。
7. 启动作业。详情请参见 启动。

关闭AutoScale

说明 在启动作业时，需要开启AutoScale自动调优功能后，才可以进行关闭AutoScale自动调优操作。

您可以在作业运行期间，动态关闭AutoScale自动调优功能。

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 单击页面顶部的 **运维**。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 单击页面右上角的 **关闭自动调优**。
3. 单击**确认**。

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

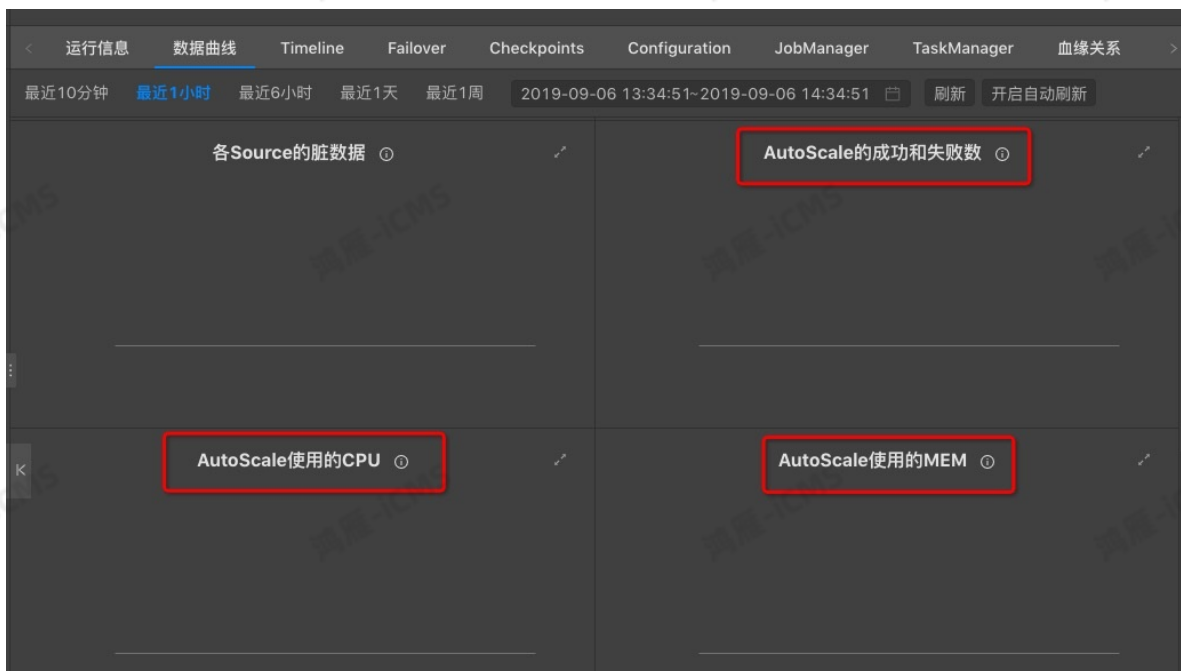
查看AutoScale效果

说明 请登录作业运维界面进行查看。作业运维页面登录步骤如下：

1. 登录[实时计算控制台](#)。
2. 单击页面顶部的 **运维**。
3. 在作业列表区域，单击作业名称下的目标作业名。

• AutoScale Metric

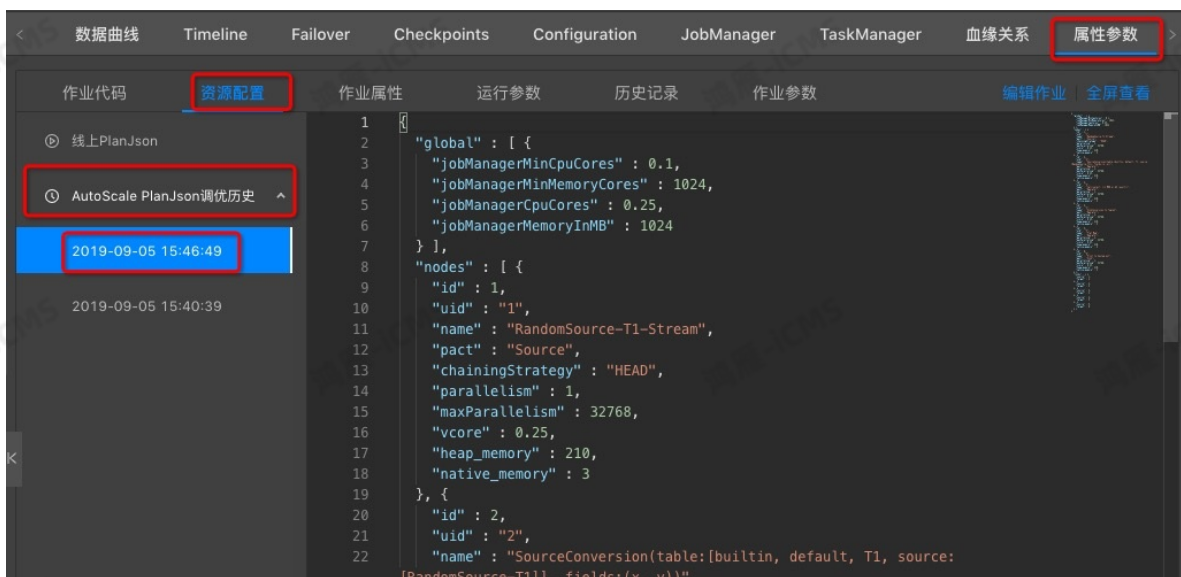
在**数据曲线 > OverView**页面，可以查看AutoScale的信息。



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

● AutoScale PlanJson信息

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



常见问题

- Q: 作业启动时产生报错： ERR_ID: CLI-00000001 。
- A: 参见报错：CLI-00000001。
- Q: 作业上线时参数校验报错： resource validate failed! please modify your planjson or max CU and try again. 。
- A: 以下2种场景可能导致该报错：
 - Planjson资源文件和实际作业不匹配
 - 报错原因：Planjson资源文件和实际作业不匹配，导致能够关联（Chain）在一起的节点没有关联在一起，使作业所需资源变大。
 - 解决方案：重新生成配置信息，操作步骤参见 报错：CLI-00000001。
 - 配置的可用资源过小
 - 报错原因：作业所需的CPU或者内存中的一个资源超过最大资源。
 - 解决方案：返回上一步，增加最大可用资源的CU数。
- Q: AutoScale没有启动。
- A: 排查方法如下：

- 检查作业是否频繁运行失败（Failover）。AutoScale开启的前提是作业本身逻辑正确并且能够稳定运行。
- 查看JobManager的相关日志，检查是否存在系统异常。
- Q: AutoScale没有效果。
 - A: 请按照以下顺序进行排查：
 - i. 检查最大资源限制，确认作业已使用资源是否达到最大资源限制。
 - ii. 检查作业数据源（Source）节点的逻辑，查看Source节点是否关联（Chain）过多的Operator。如果关联过多Operator，请手动编辑Planjson，在一些关键的位置进行关联（Chain）拆分。解决方案参见[手动配置调优](#)。
 - iii. 查看JobManager的相关日志，检查是否存在系统异常。
- Q: AutoScale可能出现的问题有哪些？
 - A:
 - 作业自动重启

AutoScale根据实际流量来动态调整并发数以及资源，因此可能会在流量上涨以及流量下降时，通过自动重启触发资源调整。
 - 短时间内出现数据传输延迟

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

在部分场景下AutoScale无法有效工作，导致作业延迟和作业频繁的进行配置调整。
 - 并发数先减少后增多

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

6.6.5. 手动配置调优

您可以手动调整实时计算Flink版的作业参数、资源参数和上下游参数，提升实时计算Flink版作业的性能。

 **说明** 建议您完成作业反压检测后，再判断是否需要配套调优。实时计算Flink版3.0以上版本作业反压检测方法，详情请参见[如何对实时计算Flink版3.0以上版本的作业进行反压检测？](#)。

手动配置调优概述

手动配置调优的内容主要有3种类型：

- 上下游参数调优：对作业中上下游存储的参数进行调优。
- 作业参数调优：对作业中miniBatch等参数进行调优。
- 资源参数调优：对作业中Operator的并发数（Parallelism）、CPU（Core）和堆内存（Heap Memory）等参数进行调优。

下面对以上3类调优进行介绍。参数调优后将生成新的配置，作业需要先停止再启动或者先暂停再恢复后才能使用新的配置，启动新配置的方法请参见[重新启用新的配置](#)。

上下游参数调优

实时计算Flink版每条数据均会触发上下游存储的读写，因此会对上下游存储形成存储压力。设置batchsize，批量读写上下游存储数据可以降低上下游存储的压力。支持batchsize参数的上下游存储如下表所示。

上下游	参数	说明	设置参数值
DataHub源表	batchReadSize	单次读取的条数	可选，默认值为10。
DataHub结果表	batchSize	单次写入的条数	可选，默认值为300。
日志服务LOG源表	batchGetSize	单次读取logGroup的条数	可选，默认值为100。
分析型数据库MySQL版2.0结果表	batchSize	一次批量写入的条数	可选，默认值为1000。
云数据库RDS版结果表	batchSize	一次批量写入的条数	可选，默认值为4096。

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

作业参数调优

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

- 新增加作业参数后，建议您停止作业，再启动作业。
- 更新作业参数值后，建议您暂停作业，再恢复作业。

```

# 3.2及以上版本开启window miniBatch方法（3.2及以上版本默认不开启window miniBatch）。
sql.exec.mini-batch.window.enabled=true
# exactly-once语义。
blink.checkpoint.mode=EXACTLY_ONCE
# checkpoint间隔时间，单位毫秒。
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# 实时计算Flink版2.0及以上版本使用niagara作为statebackend，以及设定state数据生命周期，单位毫秒。
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# 实时计算Flink版2.0及以上版本开启5秒的microbatch（窗口函数不需要设置该参数）。
blink.microBatch.allowLatencyMs=5000
# 表示整个Job允许的延迟。
blink.miniBatch.allowLatencyMs=5000
# 双流join节点优化参数。
blink.miniBatch.join.enabled=true
# 单个Batch的size。
blink.miniBatch.size=20000
# local优化，实时计算Flink版2.0及以上版本默认已经开启，1.6.4版本需要手动开启。
blink.localAgg.enabled=true
# 实时计算Flink版2.0及以上版本开启partial优化，解决count distinct效率低问题。
blink.partialAgg.enabled=true
# union all优化。
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# GC优化（源表为SLS时，不能设置该参数）。
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-verbose:gc -XX:NewRatio=3 -XX:+Print
GCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4'
# 时区设置。
blink.job.timeZone=Asia/Shanghai

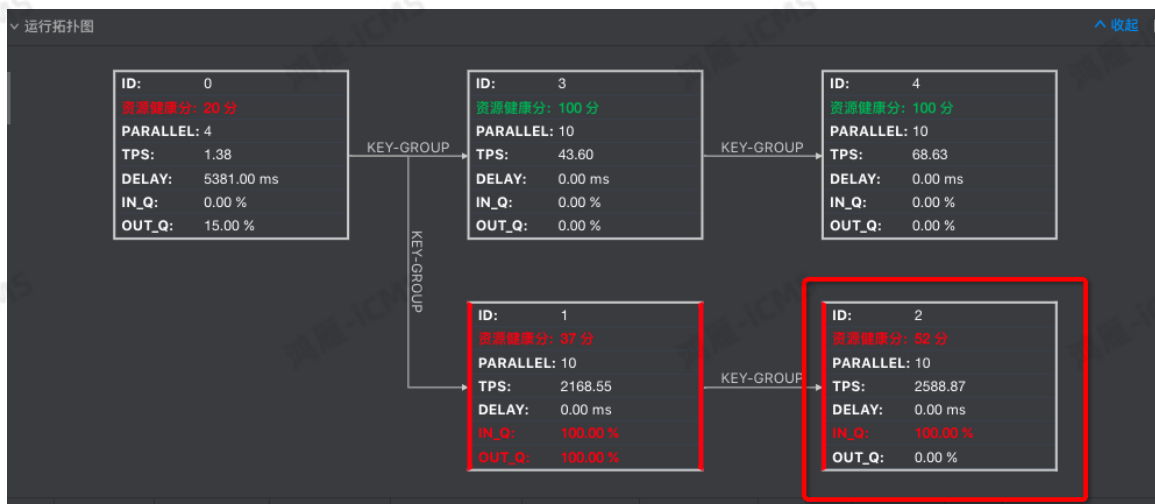
```

资源调优

下文通过示例为您介绍资源调优方法：



1. 分析过程

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



- ii. 单击目标Task节点（本示例为2号Task节点）。
- iii. 在SubTask List, 查看In Queue为 100% 的行。
- iv. 单击目标行左侧的LOGO。
- v. 在Metrics Graph页签, 查看CPU和MEM的使用情况。

2. 性能调优

- i. 单击作业编辑页面右侧的 资源配置, 进入调优窗口。
- ii. 对目标Operator或者Group进行参数修改。
 - 单个Operator参数修改
 - a. GROUP框内, 单击右上角的加号 (+) 。
 - b. 鼠标悬停至目标Operator框内。
 - c. 单击目标Operator右侧的  图标。
 - d. 在修改Operator数据页面, 修改参数。
 - GROUP批量参数修改
 - a. 鼠标悬停至GROUP框内。
 - b. 单击GROUP右侧的  图标。
 - c. 在批量修改Operator数据页面, 修改参数。
- iii. 完成配置参数修改后, 单击 资源配置 右上角的配置信息操作 > 应用当前配置。
 - 如果增加了Group的资源配置后, 作业的运行效率提升不明显, 需要按照以下顺序分析问题:
 - a. 该节点是否存在数据倾斜现象。
 - b. 复杂运算的Operator节点（例如, GROUP BY、WINDOW和JOIN等）的子节点是否存在异常。
 - Operator子节点拆解方法:
 - a. 单击目标Operator。
 - b. 修改chainingStrategy参数值为 HEAD 。

如果chainingStrategy已设置为 HEAD , 需要修改后一个Operator的chainingStrategy参数值为 HEAD 。chainingStrategy参数说明如下。

参数	说明
ALWAYS	算子会尽可能地链接在一起。为了优化性能, 通常需要让算子尽可能链接在一起, 同时增加并发度。
NEVER	算子不会和上下游的算子链接在一起。
HEAD	算子不会和上游的算子链接在一起, 但是会和下游的算子链接在一起。

3. 资源参数的配置原则和建议

- 作业的资源配置建议为: core:heap_memory=1:4 , 即1个核对应4 GB内存。例如:
 - core参数值为1CPU, heap_memory参数值为4 GB, 则是按资源分配结果的 1CPU:4G

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

② 说明

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

○ `parallelism`

■ Source节点

Source的个数和上游Partition数量有关。例如，Source的个数是16，Source的并发数可以为16、8或4等，不得超过16。

② 说明 Source的并发数不能大于Source的Shard数。

■ 中间节点

根据预估的QPS计算：

- QPS低的任务，中间处理节点数和Source并发数一样。
- QPS高的任务，中间处理节点数配置为比Source并发数大，例如，64、128或256。

■ Sink节点

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

② 说明 如果配置过大会导致输入超时或失败。例如，下游Sink节点个数是16，建议Sink的并发数最大为48。

○ `core`

默认值为0.1，根据实际CPU使用配置，建议配置为0.25。

○ `heap_memory`

堆内存，默认为值为256（单位为MB），请根据实际的内存使用状况进行配置。

○ `state_size`

存在GROUP BY、JOIN、OVER和WINDOW的Task节点中需要配置参数 `state_size`为 1，表示该Operator会使用state功能，作业会为该Operator申请额外的内存。`state_size`默认值为0。

② 说明 如果`state_size`参数值不设置为 1，则作业可能运行失败。

重新启用新的配置

完成配置后，建议您采用暂停再恢复作业的方式使新配置生效，而不是停止再启动作业的方式使新配置生效。因为停止作业后，作业状态会被清除，从而可能导致计算结果不一致。

说明

- 暂停再恢复：仅更改资源配置、调整WITH参数大小或调整作业参数大小。
- 停止再启动：需要更改SQL逻辑、更改作业版本、增加WITH参数或增加作业参数。

完成作业重启或恢复后，可以在 运维页面，运行信息页签下的Vertex拓扑中查看新的配置是否生效。

- 暂停再恢复步骤如下：
 - 上线作业。上线步骤参见上线，详情请参见 [上线](#)。配置方式选择使用上次资源配置（手动资源配置）。
 - 在运维页面，单击目标作业操作列下的 暂停。
 - 在运维页面，单击目标作业操作列下的 恢复。
 - 在恢复作业运行，单击按最新配置恢复。



- 停止再启动步骤如下：
 - 停止作业。停止作业步骤请参见 [停止](#)。
 - 启动作业。启动作业步骤参见 [启动](#)。

相关参数说明

- Global

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

- Nodes

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

参数	说明	能否修改
chainingStrategy	ChainingStrategy用来定义算子链接的策略。当一个算子和上游算子链接在一起，表示它们会运行在同一个线程，合并为一个有多个运行步骤的算子。 ChainingStrategy支持以下3种方式： <ul style="list-style-type: none">◦ ALWAYS：算子会尽可能的链接在一起。为了优化性能，通常最佳实践是让算子尽可能链接在一起，同时增加并发度。◦ NEVER：算子不会和上下游的算子链接在一起。◦ HEAD：算子不会和上游的算子链接在一起，但是会和下游的算子链接在一起。	能
parallelism	并发数，默认值为1，可以根据实际数据量增大并发数。	能
core	CPU，默认为0.1，可以根据实际CPU使用情况进行配置。建议设置为0.25。	能
heap_memory	堆内存，默认为256（单位为MB），可以根据实际内存使用情况进行配置。	能
direct_memory	JVM（Java虚拟机）堆外内存，默认0（单位为MB）。	能，但不建议您修改该参数。
native_memory	JVM（Java虚拟机）堆外内存，JNI（Java Native Interface）中使用，默认为0。如果需要，可以配置为10，主要提供给statebackend使用。	能，但不建议您修改该参数。

• Chain

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

说明

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

6.6.6. 典型的反压场景及优化思路

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

反压检测机制

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

1. 登录作业运维页面。
 - i. 登录[实时计算控制台](#)。

- ii. 单击页面顶部的 运维。
 - iii. 在作业列表区域，单击作业名称下的目标作业名。
2. 在运行信息页签，单击对应的Vertex节点边框。



3. 在右侧信息区域的 BackPressure > Status，查看反压状态。
- 红色high：表示该节点存在反压。
 - 绿色ok：表示该节点不存在反压。

反压场景及优化思路

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

- 仅存在1个Vertex，经检测无反压



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

- 存在多个Vertex，倒数第2个Vertex经检测存在反压



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

- 若只涉及写出下游存储的操作，则可能是写出速度较慢所导致。建议增加Vertex2的并发数，或者配置对应结果表（Sink）的 `batchsize` 参数，具体操作步骤请参见[上下游参数调优](#)。
 - 若涉及到除写出下游存储外的其他操作，需要将其他操作对应的Operator拆分后再进一步判断。拆分方法参见[资源调优](#)。
- 存在多个Vertex，非倒数第2个Vertex经检测存在反压



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

- GROUP BY操作：可以考虑增加并发或设置 `miniBatch` 参数优化状态（State）的操作，具体方法参见[作业参数调优](#)。
 - 维表JOIN操作：可以考虑增加并发数或者设置维表的Cache策略，具体请参见对应维表文档。
 - UDX操作：可以考虑增加并发数或者优化对应的UDX代码。
- 存在多个Vertex，所有Vertex经检测不存在反压

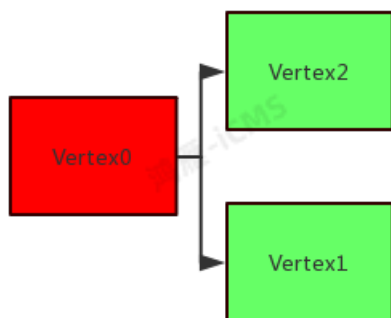


该场景表示：可能的性能瓶颈点在Vertex0，可以通过查看Vertex0中的Operator组的名称判断具体的操作。

- 若其中只存在读取上游源表（Source）的操作，则可能实时计算本身没有性能瓶颈，只是读取Source的速度较慢而导致延时较大。此时可通过增加Source节点的并发数或者设置读取Source的 batchsize 的方法进行优化，具体步骤参见 [上下游参数调优](#)。

❓ 说明 Source节点增加后的并发数不能大于上游存储的Shard数。

- 若其中除读取上游Source的操作外，还存在其他操作，建议将其他操作的Operator拆分后再进一步判断，拆分方法参见[资源调优](#)。
- 存在多个Vertex，其中1个Vertex经检测存在反压，但其后续的多个并行Vertex经检测不存在反压



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

6.6.7. SQL Tuning Advisor

6.6.7.1. Partitioned All Cache调优

您可以使用Partitioned All Cache调优方式解决超大维表使用Cache All策略在进行JOIN时，缓存无法加载维表全部数据的问题。

背景信息

维表JOIN时使用Cache All策略，默认每个进程都会加载全量的维表数据到缓存中。使用Cache All的情况下，维表JOIN节点配置的内存大小至少是维表内存大小的2倍。

当维表较大时，维表JOIN节点需要消耗大量的内存，另外作业GC会比较严重。甚至对于超大维表（超过1TB），内存里无法加载全量维表数据。为了解决这个问题，引入Partitioned All Cache优化，上游数据按照Join Key进行Shuffle，每个进程上只需要加载所需的维表数据到缓存中。

开启PartitionedJoin优化可以减少内存开销。但由于上游数据需要按照Join Key进行Shuffle，则会引入额外的网络开销和CPU开销。因此以下场景不适合开启PartitionedJoin：

- 上游数据在JOIN Key上存在严重的数据倾斜，这种场景如果开启PartitionedJoin，则会因为数据倾斜导致慢节点。
- 维表数据较小，例如小于2 GB。这种场景如果开启PartitionedJoin，节约的内存开销和额外引入的网络开销和CPU开销相比，不划算。

调优方式

在维表的DDL的WITH参数中添加 partitionedJoin = 'true' 参数。

示例代码

```
CREATE TABLE white_list (  
  id varchar,  
  name varchar,  
  age int,  
  PRIMARY KEY (id),  
  PERIOD FOR SYSTEM_TIME --维表标识。  
) with (  
  type = 'odps',  
  endPoint = 'your_end_point_name',  
  project = 'your_project_name',  
  tableName = 'your_table_name',  
  accessId = 'your_access_id',  
  accessKey = 'your_access_key',  
  `partition` = 'ds=20180905',  
  cache = 'ALL',  
  partitionedJoin = 'true' -- 开启partitionedJoin。  
);
```

6.6.7.2. MiniBatch/MicroBatch调优

您可以通过MiniBatch/MicroBatch调优方式提升吞吐。

背景信息

MiniBatch和MicroBatch都是微批处理，只是微批的触发机制略有不同。原理都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。但是二者有以下差异：

- MiniBatch：主要依靠在每个Task上注册的Timer线程来触发微批，需要消耗一定的线程调度性能。
- MicroBatch：为MiniBatch的升级版，主要基于事件消息来触发微批，事件消息会按您指定的时间间隔在源头插入。MicroBatch在元素序列化效率、反压表现、吞吐和延迟性能上都要优于MiniBatch。

微批处理可以显著的提升系统性能，建议您开启微批处理，例如在聚合场景。但以下场景不建议您开启：

- 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。
- GroupAggregate聚合度很低（Output/Input > 0.8）时，一个批次里基本聚合不到数据，不建议开启。

优化方式

在开发页面右侧作业参数中设置blink.microBatch.allowLatencyMs或blink.miniBatch.allowLatencyMs参数，二者效果相同，单位为ms。

说明 如果一个作业中同时设置了blink.microBatch.allowLatencyMs和blink.miniBatch.allowLatencyMs参数，建议二者值保持一致。如果设置的值不一致，则代码下方参数生效。

代码示例

- 设置microBatch

```
blink.microBatch.allowLatencyMs=5000
```

- 设置miniBatch

```
blink.miniBatch.size=20000
```

- 同时设置了microBatch和miniBatch

```
# 防止OOM设置每个批次最多缓存数据的条数。  
blink.miniBatch.size=20000  
blink.microBatch.allowLatencyMs=5000  
blink.miniBatch.allowLatencyMs=6000---该配置生效。
```

6.6.7.3. Cache优化

您可以在维表Join时开启Cache策略并配置相关参数，提高作业吞吐。

背景信息

实时计算Flink版支持LRU和ALL两种缓存策略，它们的调优原理如下：


- **LRU**：缓存维表里的部分数据。您可以通过 `cache='LRU'` 开启LRU缓存优化，系统会为每个JoinTable节点创建一个LRU本地缓存。源表的每条数据都会触发系统在Cache中查找数据，如果存在则直接关联输出，减少了一次I/O请求。如果不存在，再发起查询请求去维表中查找，返回的结果会先存入缓存以备下次查询。

为了防止缓存无限制增长，可以通过 `cacheSize` 调整缓存大小。为了定期更新维表数据，可以通过 `cacheTTLms` 调整缓存的失效时间。`cacheTTLms` 作用于每条缓存数据上，也就是某条缓存数据在指定时间内没有被访问，则会从缓存中移除。

- **ALL**：缓存维表里的所有数据。您可以通过 `cache='ALL'` 开启ALL缓存优化。系统会为JoinTable节点起一个异步线程去同步维表数据。在作业刚启动时，会先阻塞上游数据，直到缓存数据加载完毕，可以保证在处理上游数据时，维表数据已经被加载到缓存中。

之后所有的维表查询请求都会通过Cache查询。如果在Cache中无法找到数据，则关联键不存在。在缓存失效后，重新加载维表数据到缓存中。重新加载维表的过程不会阻塞维表关联的处理流程。重新加载的维表数据暂时存放在临时内存中，等加载完毕再和原先的维表数据进行原子替换。

因为几乎没有I/O请求，所以使用cache ALL的维表JOIN性能可以非常高。但是由于内存需要可以同时容纳两份维表数据，因此需要加大内存的配置。

 **说明** 如果您的业务场景要求每次维表查询请求必须发起一次查询，不可以使用缓存数据，请不要开启缓存策略。

调优方式

维表的DDL的WITH参数中添加 `cache='LRU'`或`cache='ALL'`。Cache相关参数如下。

参数	说明	是否必填	备注
cache	缓存策略	否	<ul style="list-style-type: none"> <i>None</i>（默认值）：无缓存。 <i>LRU</i>：需要配置相关参数：缓存大小（cacheSize）、缓存更新时间间隔（cacheTTLms）和是否开启PartitionedJoin（partitionedJoin）。 <i>ALL</i>：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）和是否开启PartitionedJoin（partitionedJoin）。
cacheSize	缓存大小	否	当缓存策略选择 <i>LRU</i> 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	<ul style="list-style-type: none"> 当缓存策略选择<i>LRU</i>时，可以设置缓存失效时间，默认不过期。 当缓存策略选择<i>ALL</i>时，缓存失效时间为缓存重新加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为 <i>ALL</i> 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	可选，默认空，格式为 '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'。 <ul style="list-style-type: none"> 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。
partitionedJoin	是否开启PartitionedJoin。	否	默认情况下为false，表示不开启partitionedJoin。在开启PartitionedJoin优化时，主表会在关联维表前，先按照Join KEY进行Shuffle，这样做有以下优点： <ul style="list-style-type: none"> 在缓存策略为<i>LRU</i>时，可以提高缓存命中率。 在缓存策略为<i>ALL</i>时，节省内存资源，因为每个并发只缓存自己并发所需要的数据。

示例代码

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id)
) with (
  type = 'odps',
  endPoint = 'your_end_point_name',
  project = 'your_project_name',
  tableName = 'your_table_name',
  accessId = 'your_access_id',
  accessKey = 'your_access_key',
  `partition` = 'ds=20180905',
  cache = 'ALL'
);
```

6.6.7.4. Async优化

您可以在维表JOIN时开启Async优化并配置相关参数，提高吞吐。

背景信息

维表JOIN默认为同步访问方式，上游进来一条数据，则系统去物理表中查询一次，等待返回后输出关联结果。因此网络等待时间极大地阻碍了吞吐和延迟。为了解决同步访问的问题，引入异步模式并发处理查询请求，从而连续请求之间不需要阻塞等待。

Flink SQL基于Flink Async I/O和异步客户端实现了维表JOIN的异步化，极大地提高了吞吐率。

调优方式

在维表的DDL的WITH参数中添加 `async='true'`，Async 相关参数如下。

参数	说明	是否必填	备注
<code>async</code>	是否开启异步请求	否	默认值为false。
<code>asyncResultOrder</code>	异步结果顺序	否	取值如下： <ul style="list-style-type: none"> <code>unordered</code>（默认值）：无序。 <code>ordered</code>：有序。
<code>asyncTimeoutMs</code>	异步请求的超时时间	否	单位毫秒，默认值为3分钟。
<code>asyncCapacity</code>	异步请求的队列容量	否	默认值为100。
<code>asyncCallbackThreads</code>	回调处理线程数	否	回调类中的 <code>onComplete</code> 和 <code>onError</code> 默认会在线程池中处理该线程池的大小，默认值为50。

参数	说明	是否必填	备注
asyncConnectionQueueMaxsize	最大请求发送数	否	当等待某个服务器返回结果的请求数量达到asyncConnectionQueueMaxsize值时，异步请求调用也会被阻塞，以防止客户端自身OOM（OutOfMemory），默认值为100。
asyncCallbackQueueMaxsize	最大回调处理队列	否	当等待回调处理的请求达到asyncCallbackQueueMaxsize值时，异步请求调用也会被阻塞，以防止客户端自身OOM，默认值为500。

示例代码

```
CREATE TABLE dim_cn_item(
  rowkey VARCHAR,
  item_id VARCHAR,
  title VARCHAR,
  cate_id VARCHAR,
  cate_name VARCHAR,
  cate_level1_id VARCHAR,
  cate_level2_id VARCHAR,
  cate_level3_id VARCHAR,
  cate_level1_name VARCHAR,
  cate_level2_name VARCHAR,
  cate_level3_name VARCHAR,
  pinlei_id VARCHAR,
  pinlei_name VARCHAR,
  bu_id VARCHAR,
  bu_name VARCHAR,
  PRIMARY KEY(rowkey)
) WITH(
  type='alihbase',
  diamondKey = 'xxx',
  diamondGroup = 'yyy',
  cacheTTLs='3600000',
  async='true',
  cache='LRU',
  columnFamily='cf',
  cacheSize='1000',
  tableName='yourTableName'
);
```


常见问题

- [报错详情](#)

```
Caused by: org.apache.flink.table.api.TableException: Output mode can not be UNORDERED if the input is an update stream.
at org.apache.flink.table.plan.util.TemporalJoinUtil.validate(TemporalJoinUtil.scala:340)
at org.apache.flink.table.plan.nodes.common.CommonTemporalTableJoin.translateToPlanInternal(CommonTemporalTableJoin.scala:144)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translateToPlanInternal(StreamExecTemporalTableJoin.scala:98)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translateToPlanInternal(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:58)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.org$apache$flink$table$plan$nodes$exec$StreamExecNode$$super$translateToPlan(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.exec.StreamExecNode$class.translateToPlan(StreamExecNode.scala:38)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translateToPlan(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translateToPlan(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlanInternal(StreamExecCalc.scala:89)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlanInternal(StreamExecCalc.scala:43)
at org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:58)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.org$apache$flink$table$plan$nodes$exec$StreamExecNode$$super$translateToPlan(StreamExecCalc.scala:43)
at org.apache.flink.table.plan.nodes.exec.StreamExecNode$class.translateToPlan(StreamExecNode.scala:38)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlan(StreamExecCalc.scala:43)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translate(StreamExecSink.scala:158)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translateToPlanInternal(StreamExecSink.scala:103)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translateToPlanInternal(StreamExecSink.scala:53)
at org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:58)
at
```

- 报错原因

上游数据为Update Stream，维表JOIN时Async模式未开启ordered。

 说明 Update Stream为TOPN或双流JOIN等。

- 解决方案

在维表WITH参数中设置 `asyncResultOrder='ordered'`。

6.6.7.5. APPROX_COUNT_DISTINCT优化

您可以通过APPROX_COUNT_DISTINCT（近似Count Distinct）优化提升作业性能。

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优化存在的性能瓶颈。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。

调优方式

SQL中直接使用APPROX_COUNT_DISTINCT(user)代替COUNT(DISTINCT user)。APPROX_COUNT_DISTINCT(user)语法格式如下。

```
APPROX_COUNT_DISTINCT(col [, accuracy])
```

其中：

- col：字段名称，任意类型。
- accuracy：指定准确率，可选，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

a (VARCHAR)	c (BIGINT)
Hi	1
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 MyTable  
GROUP BY a;
```

- 测试结果

a (VARCHAR)	b (BIGINT)	c (BIGINT)
Hi	5	5

6.6.7.6. Local-Global优化

您可以通过Local-Global优化解决Aggregate数据倾斜问题。

背景信息

Local-Global优化即将原先的Aggregate分成Local和Global两阶段聚合，即MapReduce模型中Combine+Reduce处理模式。第一阶段在上游节点本地攒一批数据进行聚合（localAgg），并输出这次微批的增量值（Accumulator）。第二阶段再将收到的Accumulator merge起来，得到最终的结果（globalAgg）。

Local-Global本质上能够靠localAgg聚合掉倾斜的数据，从而降低globalAgg热点，从而提升性能。Local-Global用于提升SUM、COUNT、MAX、MIN和AVG等普通Aggregate性能，以及解决这些场景下的数据热点问题。

调优方式

UDAF必须实现merge方法才可以触发Local-Global优化。实现merge方法请参见 [示例代码](#)。

 说明 Blink 2.0及以后版本默认开启Local-Global。如果您需要关闭Local-Global，您可以在 [作业参数](#)中，设置blink.localAgg.enabled=false。

示例代码

```
import org.apache.flink.table.functions.AggregateFunction;
public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    //定义存放count udaf的状态的accumulator数据结构
    public static class CountAccum {
        public long total;
    }
    //初始化count udaf的accumulator
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }
    //getValue提供了如何通过存放状态的accumulator计算count UDAF的结果的方法
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }
    //accumulate提供了如何根据输入的数据更新count UDAF存放状态的accumulator
    public void accumulate(CountAccum accumulator, Object iValue) {
        accumulator.total++;
    }
    public void merge(CountAccum accumulator, Iterable<CountAccum> its) {
        for (CountAccum other : its) {
            accumulator.total += other.total;
        }
    }
}
```

6.6.7.7. ROW_NUMBER OVER WINDOW去重

您可以通过ROW_NUMBER OVER WINDOW实现高效去重源数据。

背景信息

去重本质是一种特殊的TopN，实时计算Flink版支持以下两种去重策略：

- 保留首行的去重策略（Deduplicate Keep FirstRow）：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。
- 保留末行的去重策略（Deduplicate Keep LastRow）：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。

调优方式

由于SQL中没有直接的去重语法，实时计算Flink版使用SQL的ROW_NUMBER OVER WINDOW语法表示去重，语法格式如下。

```
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，必须指定一个时间属性字段（Proctime或Rowtime）。还需要指定排列顺序（Deduplicate Keep FirstRow）或者倒序（Deduplicate Keep LastRow）。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px;"> <p>说明</p> <ul style="list-style-type: none"> 如果不声明时间属性字段，默认为Proctime。 如果不声明排列顺序，默认为asc。 </div>
rownum	仅支持rownum=1或rownum<=1。

ROW_NUMBER OVER WINDOW去重时需要执行两层查询：

1. ROW_NUMBER()窗口函数根据时间属性列，对相同数据进行排序并标上排名。

说明

- 当排序字段是Proctime列时，Flink就会按照系统时间去重，其每次运行的结果是不确定的。
- 当排序字段是Rowtime列时，Flink就会按照业务时间去重，其每次运行的结果是确定的。

2. 对排名进行过滤，只取第一条或最后一条，达到去重目的。

代码示例

- Deduplicate Keep FirstRow

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。

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

- Deduplicate Keep LastRow

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。


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

6.6.7.8. Partial-Final优化

您可以通过Partial-Final优化的方式解决Count Distinct热点问题。

背景信息

通常，为了解决Count Distinct热点问题，您会通过增加按Distinct Key取模的打散层的方式，手动将Aggregate改写为两层聚合。Blink 2.2.0及以后版本支持Count Distinct自动打散，即Partial-Final优化。以下场景适用于开启Partial-Final优化：

- 使用Count Distinct后但无法实现聚合节点性能要求。
- Count Distinct所在的Aggregate不包含UDAF。

 说明 因为partial-final优化会自动将Aggregate打散成两层聚合，引入额外的网络shuffle。所以，在数据量不大的情况下，反而造成资源浪费。

调优方式

在开发页面右侧作业参数中设置blink.partialAgg.enabled=true。

开启Partial-Final优化后，您可以在最终生成的拓扑图的节点名中，观察是否包含Expand节点，或者原来一层的Aggregate变成了两层的Aggregate。

示例代码

```
blink.partialAgg.enabled=true
```

6.7. 监控报警

本文为您介绍实时计算监控报警的操作流程以及报警规则的创建步骤。

什么是云监控报警服务


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

创建报警规则

创建报警规则详情，参见[设置报警规则](#)。

实时计算Flink版监控项

监控项	单位	Metric	Dimensions	Statistics
业务延迟 (s)	s	inputDelay	userId、regionId、projectName、jobName	Average
读入RPS (RPS)	RPS	ParserTpsRate	userId、regionId、projectName、jobName	Average
写出RPS (RPS)	RPS	SinkOutTpsRate	userId、regionId、projectName、jobName	Average
FailoverRate (%)	%	TaskFailoverRate	userId、regionId、projectName、jobName	Average
处理延迟 (s)	s	FetchDelay	userId、regionId、projectName、jobName	Average

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

查看监控报警信息

1. 登录[实时计算控制台](#)。
2. 单击页面顶部的[运维](#)。
3. 在运维界面，单击目标作业名称。
4. 在页面右上角，单击[更多](#) > [监控](#)。
5. 在监控图标页面，查看作业的监控指标。

6.8. 自定义日志级别和下载路径

您可以编辑实时计算作业参数，自定义实时计算作业的日志级别和下载路径。

 **重要** 仅实时计算3.2及以上版本支持自定义日志级别和下载路径。

操作步骤

1. 登录[实时计算控制台](#)。
2. 单击页面顶部的[开发](#)。

3. 在左侧导航栏的作业开发区域，双击文件夹，查找目标作业。
4. 双击目标作业，进入作业编辑页面。
5. 在作业编辑页面右侧的作业参数页面，输入log4j的配置信息。

log4j配置信息	说明
根Logger	<p>Logger负责处理日志记录的部分操作，其语法格式为：<code>log4j.rootLogger = [level], appenderName, appenderName, ...</code>，表示输出指定级别以上的日志信息到指定的一个或者多个位置。其中：</p> <ul style="list-style-type: none"> ◦ level：日志级别，优先级从高到低分别为ERROR（严重错误）、WARN（一般警告）、INFO（提示信息）、DEBUG（调试信息）。 ◦ appenderName：指定日志信息输出到哪个地方，可以同时指定多个输出目的地。
日志信息输出目的地Appender	<p>Appender负责控制日志记录操作的输出，其语法格式如下。</p> <pre>log4j.appender.appenderName = fully.qualified.name.of.appender.class log4j.appender.appenderName.option1 = value1 ... log4j.appender.appenderName.optionN = valueN</pre> <p>Appender分为以下两种：</p> <ul style="list-style-type: none"> ◦ log4j自带的Appender <ul style="list-style-type: none"> ▪ <code>org.apache.log4j.ConsoleAppender</code>：控制台 ▪ <code>org.apache.log4j.FileAppender</code>：文件 ▪ <code>org.apache.log4j.DailyRollingFileAppender</code>：每天产生一个日志文件 ▪ <code>org.apache.log4j.RollingFileAppender</code>：文件大小到达指定大小时产生一个新文件 ▪ <code>org.apache.log4j.WriterAppender</code>：以流格式发送日志信息到任意指定地方 ◦ 自定义Appender <p>目前只支持日志服务SLS和对象存储OSS，且对应的类是固定的，如下所示：</p> <ul style="list-style-type: none"> ▪ SLS：<code>log4j.appender.loghub=com.alibaba.blink.log.loghub.BlinkLogHubAppender</code> ▪ OSS：<code>log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender</code> <p>此外，您也可以配置任何log4j语法支持的配置项。</p> <p>详细示例请参见 修改日志级别为DEBUG，并输出日志至OSS示例。</p>

 **说明** 完成编辑作业参数后，请重启作业。您可以在OSS存储空间查看新生成的日志。


6. 停止作业。作业停止步骤请参见 [停止](#)。

7. 启动作业。作业启动步骤请参见 [启动](#)。

注意事项

- 选择和独享模式集群相同的OSS存储路径。
- 日志下载包含log4j日志的输出方式，不包含 *system.out* 日志。
- 指定的SLS或OSS存储和作业所在集群可连通。
- 如果自定义日志输出方式配置错误，作业通常可以启动成功，但不能按照自定义配置打印日志。

修改日志级别为DEBUG，并输出日志至OSS示例

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

```
#将总体日志级别修改为DEBUG，并将日志输出至OSS存储空间。
log4j.rootLogger=DEBUG, file, oss
#固定写法。配置OSS的appender类。
log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender
#Endpoint地址。
log4j.appender.oss.endpoint=oss-cn-hangzhou****.aliyuncs.com
#accessId。
log4j.appender.oss.accessId=U****4ZF
#accessKey。
log4j.appender.oss.accessKey=hsf****DeLw
#bucket名称。
log4j.appender.oss.bucket=et****
#定义日志存储的子目录。
log4j.appender.oss.subdir=/luk****/test/
```

排除指定类的日志，并输出日志至自定义的SLS存储示例

```
#关闭log4j.logger.org.apache.hadoop包下面的日志输出。
log4j.logger.org.apache.hadoop = OFF
#固定写法。配置Loghub的appender类。
log4j.appender.loghub = com.alibaba.blink.log.loghub.BlinkLogHubAppender
#仅将ERROR级别日志输出至SLS。
log4j.appender.loghub.Threshold = ERROR
#SLS中的project名称。
log4j.appender.loghub.projectName = blink-errdumpsls-test
#SLS中的logstore名称。
log4j.appender.loghub.logstore = logstore-3
#SLS的Endpoint地址。
log4j.appender.loghub.endpoint = http://cn-shanghai****.sls.aliyuncs.com
#accessKeyId。
log4j.appender.loghub.accessKeyId = Tq****WR
#accessKey。
log4j.appender.loghub.accessKey = MJ****nfVx
```

修改日志级别为WARN，并关闭指定包下面的日志输出示例

```
#修改整体日志级别为WARN。
log4j.rootLogger=WARN,file
#关闭log4j.logger.org.apache.hadoop包下面的日志输出。
log4j.logger.org.apache.hadoop = OFF
```

6.9. 管理独享集群Blink版本

实时计算Flink版为独享模式集群提供版本管理功能，您可自行管理集群对应的Blink引擎版本。

安装版本

1. 登录版本管理页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 将鼠标悬停至控制台页面右上角个人账户位置，单击 **项目管理**。
 - iii. 在左侧导航栏，单击 **集群管理 > 集群列表**。
 - iv. 在集群列表区域，单击目标集群操作列下的 **更多 > 版本管理**。
2. 在可安装版本页面，单击目标版本操作列下的 **安装**。

您可以在可安装版本页签，根据业务需求和版本特性，安装适合的版本。

标签	说明
stable	当前推荐安装的稳定版本
beta	测试版本 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> <p>? 说明 在特定场景下才可安装使用，非特定场景不保证版本质量和稳定性，不建议安装。</p> </div>
无标签	历史稳定版本

? 说明

- 一次只能安装一个版本，安装过程中集群状态显示为 **版本部署中**。
- 不可安装已安装过的版本。
- 每个版本设有服役时间（默认一年），版本退役后您可以继续使用此版本，但阿里云不再负责维护。

切换版本

不同的Blink版本具备不同的特性，您可以根据实际需求，设置合适的Blink版本为当前版本。您可以通过 **版本切换** 功能，为单个作业变更Blink版本。

1. 登录[实时计算控制台](#)。
2. 单击页面的顶部 **开发**。
3. 在作业开发页面，双击目标文件夹下的目标作业，进入 **作业开发** 页面。

4. 在作业开发页面的右下角，单击版本信息。
5. 在选项列表中选择目标版本。

卸载版本

当前实时计算Flink版仅支持安装3个Blink版本，如果您已经安装3个Blink版本，且需要安装新版本，则需要先卸载版本。

1. 登录版本管理页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 将鼠标悬停至控制台页面右上角个人账户位置，单击 **项目管理**。
 - iii. 在左侧导航栏，单击**集群管理** > **集群列表**。
 - iv. 在**集群列表**区域，单击目标集群操作列下的**更多** > **版本管理**。
2. 在**已安装版本**页面，单击目标版本操作列下的**卸载**。

说明

- 不能卸载已设置为 **current** 的版本。
- 不能卸载作业已引用的版本。

设置为当前版本

您可以选择一个已安装的Blink版本，将它设置为当前集群内作业的默认版本。不同的Blink版本具备不同的特性，您可以根据实际需求，将合适的Blink版本设置为当前版本。

1. 登录版本管理页面。
 - i. 登录[实时计算控制台](#)。
 - ii. 将鼠标悬停至控制台页面右上角个人账户位置，单击 **项目管理**。
 - iii. 在左侧导航栏，单击**集群管理** > **集群列表**。
 - iv. 在**集群列表**区域，单击目标集群操作列下的**更多** > **版本管理**。
2. 在**已安装版本**页面，单击目标版本操作列下的**设置为current**。

常见问题

- Q: 安装引擎出现 `blink-<version> already installed` 报错信息。
A: 该版本已经安装，无需再次安装。
- Q: 安装引擎出现 `Flink versions exceeded max limitation:3` 报错信息。
A: 超出版本限制。若需安装新版本，需预先删除原有版本，使已安装版本数小于3。
- Q: 卸载引擎出现 `Node:<nodeName> in project:<projectName> still ref the version:<blinkVersion>` 报错信息。
A: 线上存在引用此版本引擎的作业，请根据报错信息中的作业名称和项目名称，将该作业先进行下线。
- Q: 单击**语法检查**或**上线**产生如下报错。

```
code:[30006], brief info:[blink script not exist, please check blink version], context info:[blink script:[/home/admin/blink/blink-2.2.6-hotfix0/bin/flink], blink version:[/home/admin/blink/blink-2.2.6-hotfix0/bin/flink]]
```

A: 该作业使用的引擎不存在。进入 [版本管理](#) > [已安装版本](#) 查看是否存在该版本引擎，若不存在请您切换或安装此版本引擎。

6.10. 监控报警

7. Blink Datastream开发指南

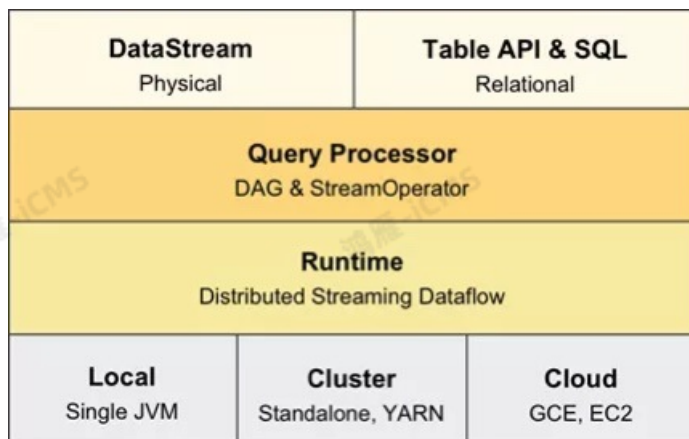
7.1. 概述

阿里云为实时计算产品Flink Datastream提供了作业提交、上线和启动功能，并提供了便利的运维管理和监控报警能力。

重要

- 仅独享模式Blink 3.2.2及以上版本，支持Flink Datastream功能。
- Flink Datastream暂不支持Flink SQL作业中的注册数据存储、作业调试和配置调优等功能。
- 如果Datastream作业访问的上下游存储提供了白名单机制，您需要进行白名单配置，配置方法请参见[数据存储白名单配置](#)。
- 目前实时计算产品支持的Datastream作业是基于开源的Flink版本的，详情请参见[开源Flink版本](#)。
- Blink Datastream API完全兼容开源Flink 1.5版本，基于Flink 1.5版本开发的Datastream作业（包括Connector）均可以在Blink上正常运行。Blink Datastream API与非1.5版本的开源Flink可能会存在不兼容的情况。

Flink Datastream提供了阿里云实时计算产品的底层API调用功能，方便您灵活地使用实时计算。



Flink Datastream开发指南主要包含如下内容：

- 作业开发
介绍在阿里实时计算开发平台上提交、上线和启动Flink Datastream作业的流程。
- 监控报警
介绍如何创建和启动报警规则，目前仅支持FailoverRate指标的监控报警。


7.2. 数据存储白名单配置

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

实时计算IP地址

实时计算分为共享模式和独享模式，2种模式的IP地址有所不同。

● 共享模式IP地址

 **说明** 实时计算共享模式已于2019年12月24日正式下线，将不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。若有新购需求，推荐使用实时计算独享模式或Flink云原生模式。

○ 访问经典网络下的数据存储

可根据项目所在的区域，配置对应区域的IP地址。

Region	白名单
华东2（上海）	11.53.0.0/16,11.50.0.0/16,10.152.0.0/16,10.154.0.0/16,11.132.0.0/16,11.178.0.0/16,11.200.210.74.11.200.215.195.11.217.0.0/16,11.219.0.0/16,11.222.0.0/16,11.223.116.79,11.223.69.0/24,11.223.70.0/24,11.223.70.173,11.223.70.48
华北2（北京）	11.223.0.0/16,11.220.0.0/16,11.204.0.0/16
华南1（深圳）	11.200.0.0/16

○ 访问VPC下的数据存储

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

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

● 独享模式IP地址

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

- 登录**实时计算控制台**。
- 将鼠标悬停至页面右上角账号名称。
- 在下拉菜单中，单击**项目管理**。
- 单击左侧导航栏中的**集群列表**。
- 在**集群列表**页面，单击**名称**字段下目标集群名称。
- 在**集群信息**窗口，查看集群的**ENI**信息。

配置RDS白名单

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

7.3. 自定义参数

在DataStream作业中，您可以根据实际需求在作业开发页面配置自定义参数后，再从Main函数中获取该自定义参数。

配置方法

在DataStream作业中配置自定义参数，自定义参数格式为 `paramName=paramValue`，其中 `paramName` 为参数名，`paramValue` 为参数值。如下为DataStream作业开发页面默认的注释信息，您可以按照注释信息提示配置自定义参数。

```
--完整主类名，必填，例如com.alibaba.realtimedatacompute.DatastreamExample。  
blink.main.class=${完整主类名}  
--包含完整主类名的JAR包资源名称，多个JAR包时必填，例如blink_datastream.jar。  
--blink.main.jar=${完整主类名jar包的资源名称}  
--默认state backend配置，当作业代码没有显式声明时生效。  
state.backend.type=niagara  
state.backend.niagara.ttl.ms=129600000  
--默认Checkpoint配置，当作业代码没有显式声明时生效。  
blink.checkpoint.interval.ms=180000  
--默认启用项目参数。  
--enable.project.config=true  
--设置自定义参数，代码中获取自定义参数的方法请参考如下链接。  
--https://help.aliyun.com/document_detail/127758.html?spm=a2c4g.11174283.6.677.61fb1e49NjowTR
```

 说明 一个DataStream作业中可定义多个自定义参数。

获取方法

在Datastream作业的Main函数中获取自定义参数。如果您已在作业开发页面配置了自定义参数，例如 `blink.job.name=jobnametest`，则可以通过如下代码将字符串 `jobnametest` 赋值于 `blink.job.name` 变量。

```
import org.apache.flink.api.java.utils.ParameterTool;
import java.io.StringReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Properties;
public class getParameterExample {
    public static void main(String[] args) throws Exception {
        final String jobName;
        final ParameterTool params = ParameterTool.fromArgs(args);
        /*此处必须写configFile，读取configFile中的参数。*/
        String configFile = params.get("configFile");
        /*创建一个Properties对象，用于保存在平台中设置的相关参数值。*/
        Properties properties = new Properties();
        /*将平台页面中设置的参数值加载到Properties对象中。*/
        properties.load(new StringReader(new String(Files.readAllBytes(Paths.get(configFile)), StandardCharsets.UTF_8)));
        /*获取参数。*/
        jobName = (String) properties.get("blink.job.name");
    }
}
```

② 说明 该示例仅适用于在Main函数中获取并使用自定义参数。如果您需要在Flink算子中使用自定义参数，则需要按照以下步骤进行：

1. 在Main代码的基础上增加如下代码，将自定义参数转化为全局作业参数（GlobalJobParameters）。

```
env.getConfig().setGlobalJobParameters(ParameterTool.fromPropertiesFile(configFilePath));
```

2. 在Flink算子获取自定义参数，代码如下。

```
getRuntimeContext().getExecutionConfig().getGlobalJobParameters().toMap()
```

7.4. 监控报警

本文为您介绍实时计算监控报警的操作流程以及如何创建报警规则。

什么是云监控报警服务

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

查看监控报警信息

1. 登录[实时计算控制台](#)。
2. 单击页面顶部的[运维](#)。
3. 在实时计算[运维](#)界面，单击目标作业名称。
4. 在目标作业[运维](#)信息页面的右上角，单击[更多 > 监控](#)。

5. 在监控页面，查看作业的监控指标。

创建报警规则

创建报警规则详情，参见 [设置报警规则](#)。

说明

- Failover Rate表示最近1分钟平均每秒Failover的次数。例如，最近1分钟Failover了1次，则 Failover Rate=1/60=0.01667。
- DataStream作业开发过程中，若引用了开源Flink提供的Connector，则在云监控中不显示 业务延迟、读入RPS和写入RPS这三项监控指标。

7.5. 作业开发

本文为您介绍Datastream作业开发POM依赖包、Datastream作业开发示例和Datastream Connector。

重要

- 仅独享模式Blink3.2.2及以上版本支持Datastream功能。
- 建议使用IntelliJ IDEA中的Maven工程开发Datastream作业。
- 实时计算Flink版独享模式不支持归档保存已停止（含暂停）的作业的运行日志。如果您需要查询已停止（含暂停）的作业运行日志，请将日志输出至您自定义的日志服务SLS或对象存储OSS中。Datastream作业的日志输出和级别修改步骤，请参见 [自定义日志级别和下载路径](#)。
- 为了避免JAR依赖冲突，您需要注意以下几点：
 - 开发页面选择的Blink版本，请和Pom依赖Blink版本保持一致。
 - Blink相关依赖，scope请使用provided，即 `<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](#)。

POM依赖包

请根据实际运行作业的Blink版本，自行添加开源版本所支持的 **POM依赖包**。Blink3.4.0版本POM文件示例如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.alibaba.blink</groupId>
  <artifactId>blink-datastreaming</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <scala.version>2.11.12</scala.version>
    <scala.binary.version>2.11</scala.binary.version>
    <blink.version>blink-3.4.0</blink.version>
    <maven.compiler.source>1.8</maven.compiler.source>
```

```
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencies>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>flink-streaming-java_${scala.binary.version}</artifactId>
    <version>${blink.version}</version>
    <scope>provided</scope>
  </dependency>
<!-- Add test framework-->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>2.11.12</version>
  </dependency>
<!-- Add logging framework-->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.7</version>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.2.0</version>
      <executions>
        <execution>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
            <transformers>
              <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                <manifestEntries>
                  <Main-Class>your main class</Main-Class>
                  <X-Compile-Source-JDK>${maven.compiler.source}</X-Compile-Source-JDK>
                  <X-Compile-Target-JDK>${maven.compiler.target}</X-Compile-Target-JDK>
                </manifestEntries>
              </transformer>
            </transformers>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

```
</manifestEntries>
</transformer>
</transformers>
<relocations combine.self="override">
  <relocation>
    <pattern>XXX</pattern>
    <shadedPattern>shaded.XXX</shadedPattern>
  </relocation>
</relocations>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

 说明 如果您需要依赖Snapshot版本，可以自行添加Snapshot版本所支持的 **POM依赖包**。

作业示例

完整示例请参见 [读取DataHub数据示例](#)。

Connector列表

Blink 3.2版本新增如下Datastream Connector:

- Kafka
- Kafka（开源版本）
- Hbase（开源版本）
- JDBC
- RDS SINK
- Elasticsearch
- MongoDB
- Redis

 说明 Datastream支持的部分Connector已完成开源，开源信息请参见 [alibaba-flink-connectors](#)。

7.6. 作业提交

本文为您介绍如何提交Datastream作业。

前提条件

已创建实时计算项目。

重要

- 仅独享模式Blink 3.2.2及以上版本支持Flink Datastream功能，推荐使用Blink 3.4.0及以上版本。
- Datastream API作业不支持资源配置调优和启动位点设置，Blink 3.4.0以下版本，上线和启动作业过程中，使用默认配置即可。

操作步骤

1. 在**实时计算控制台**，单击顶部菜单栏中的**开发**。
2. 在开发页面的顶部菜单栏中，单击**新建作业**。
3. 在**新建作业**页面，配置作业参数。

作业参数	说明
文件名称	自定义作业的名称。作业名称需在当前项目中保持唯一。
作业类型	FLINK_STREAM/DATASTREAM。 <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 Datastream API作业和Table API作业均选择 FLINK_STREAM/DATASTREAM 作业类型。 </div>
存储位置	作业存储的位置。

4. 单击左侧导航栏中的**资源引用**，进入资源引用窗口。
5. 单击**新建资源**上传已经完成开发的Datastream作业JAR包。

? 说明 在上传JAR包时，JAR包大小上限为300 MB。如果JAR包超过300 MB，请在集群绑定的OSS上传，或通过OpenAPI的方式上传JAR。

6. 单击**引用**。
7. 在作业开发界面配置参数。

```
blink.main.class=<完整主类名>
--函数完整类名，例如com.alibaba.realtimcompute.DemoTableAPI。
blink.job.name=<作业名>
--例如datastream_test。
blink.main.jar=<完整主类名JAR包的资源名称>
--完整主类名JAR包的资源名称，例如blink_datastream.jar。
```

- blink.main.class和blink.job.name为必须参数。请务必保证blink.job.name的值与步骤3中的文件名一致。如果不一致，实际作业名称将以步骤3中的文件名称为准。
- 上传多个JAR包时需要配置blink.main.jar参数。
- 您可以先自行配置其它参数，然后在程序中引用。自定义参数配置及在代码中获取参数值的方法，请参见[自定义参数](#)。
- 请不要在参数配置中使用空格。

- Blink3.2.0及以上版本无需设置Checkpoint路径，系统会自动生成Checkpoint路径。
- Blink自3.4.0开始，JAR包代码中的所有参数配置优先级会高于实时计算平台上的参数配置。例如：
 - JAR包代码和自定义参数中都设置了 statebackend，则优先使用JAR包中代码的配置。
 - JAR包代码和自定义参数中没有设置 statebackend，则优先使用实时计算平台作业模板中的默认参数niagara statebackend。

说明 请您谨慎删除模板中的默认参数，否则可能会导致作业无法Checkpoint和容错。作业名称blink.job.name是特例，代码中env.execute("jobname")设置的作业名称将会被创建作业时设置的作业名称替换，从而保持一致。此外，Metric（包括自定义Metric）名称也需要和创建作业时设置的作业名称保持一致。

8. 上线作业。

- Blink 3.4.0以下版本

a. 资源配置

选择对应的资源配置方式。第1次启动作业时，建议使用系统默认配置。

说明 实时计算支持手动资源配置，手动资源配置的方法请参见 [手动配置调优](#)。

b. 数据检查

通过数据检查后，单击下一步。

c. 上线作业

单击上线。

- Blink 3.4.0及以上版本

a. 单击作业上方上线。

b. 选择资源配置方式。

- **代码配置**：使用代码内的资源配置，与开源Flink形式一致。
- **手动配置**：使用资源配置界面中手动调整的资源配置。
 - a. 在开发页面右侧资源配置栏，单击配置信息操作 > 重新获取配置信息。
 - b. 根据需要手动修改配置信息。
 - c. 单击配置信息操作 > 应用当前配置，保存配置。

说明 手动配置时，代码显式配置的资源优先级高于平台界面上的资源配置。例如，代码中显式设置了某些算子的资源，则平台界面中对应算子的资源配置失效。实际运行时，算子的资源以您代码中显式的配置为准。代码中未显示的资源配置，以平台界面上的配置信息为准。

c. 单击下一步进行数据检查或单击 跳过数据检查。

d. 单击上线。

- 9. 在运维页面，单击目标作业操作列下的启动。

7.7. 作业开发


7.8. Datastream示例

7.8.1. 读取DataHub数据示例

本文为您介绍如何使用Datastream作业读取阿里云DataHub数据。

前提条件

- 本地安装了Java JDK 8。
- 本地安装了Maven 3.x。
- 本地安装了用于Java或Scala开发的IDE，推荐IntelliJ IDEA，且已配置完成JDK和Maven环境。
- 在DataHub上创建了Topic，并且Topic中存在 [测试数据](#)。

 **说明** 测试数据需要有4个字段，数据类型依次为STRING、STRING、DOUBLE和BIGINT。

- 已下载 [datahub-demo-master](#) 示例。


背景信息

本文以Windows和Mac操作系统为例进行演示。

 **重要** 仅Blink3.x版本支持本示例。

开发

1. 实时计算Datastream完全兼容开源Flink 1.5.2版本。下载并解压 [flink-1.5.2-compatible](#) 分支到本地。

 **说明** 下载文件中的datahub-connector中同样实现了DataHub Sink功能，具体实现请参见下载文件中的DatahubSinkFunction.java和DatahubSinkFunctionExample.java。

2. 在CMD命令窗口，进入alibaba-flink-connectors-flink-1.5.2-compatible目录后，执行如下命令。

```
mvn clean install
```

可以看到如下结果。

```
[INFO] -----
[INFO] Reactor Summary for aliyun-flink-connectors-parent 0.1-SNAPSHOT:
[INFO]
[INFO] aliyun-flink-connectors-parent ..... SUCCESS [04:54 min]
[INFO] aliyun-connectors-common ..... SUCCESS [01:55 min]
[INFO] cloudhbase-connector ..... SUCCESS [04:47 min]
[INFO] datahub-connector ..... SUCCESS [ 24.694 s]
[INFO] sls-shaded-sdk ..... SUCCESS [ 23.142 s]
[INFO] sls-connector ..... SUCCESS [ 10.359 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 12:39 min
[INFO] Finished at: 2020-03-10T18:20:03+08:00
[INFO] -----
```

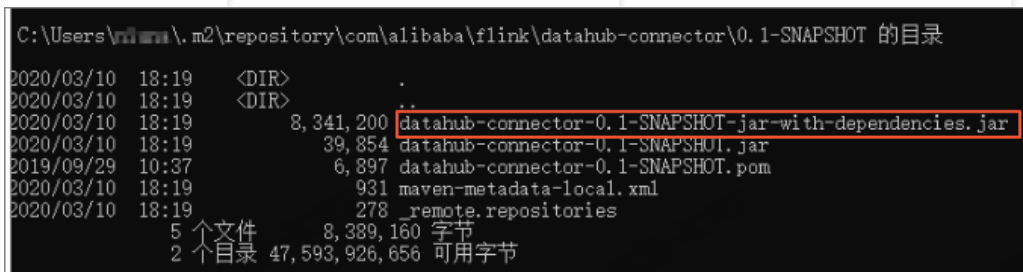
命令执行成功后，datahub-connector对应的JAR包安装到本地的Maven仓库，通常默认安装在当前登录的用户文件夹下的.m2文件夹下。

3. 执行如下命令确认是否存在 datahub-connector-0.1-SNAPSHOT-jar-with-dependencies.jar 文件（将一个JAR及其依赖的三方JAR全部打到一个包中），后续会使用该JAR。

o Windows操作系统

```
dir C:\Users\用户名\.m2\repository\com\alibaba\flink\datahub-connector\0.1-SNAPSHOT
```

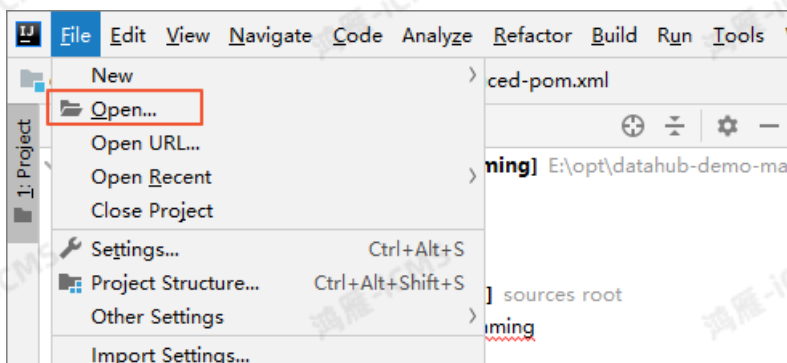
Windows操作系统执行结果



o Mac操作系统

```
ls /Users/用户名/.m2/repository/com/alibaba/flink/datahub-connector/0.1-SNAPSHOT
```

4. 在Intellij IDEA中，单击File > Open，打开刚才解压缩完成的datahub-demo-master包后，双击pom.xml查看代码。



重要

- o IDE本地调试时需要将<scope>provided</scope>注释掉。
- o 在本示例中已默认使用<classifier>jar-with-dependencies</classifier>依赖步骤3中的datahub-connector-0.1-SNAPSHOT-jar-with-dependencies.jar。

5. 修改DatahubDemo.java文件中的DataHub相关参数。

```
private static String endPoint = "inner endpoint";//内网访问。
//private static String endPoint = "public endpoint";//公网访问（填写内网Endpoint，就不用填写公网Endpoint）。
private static String projectName = "yourProject";
private static String topicSourceName = "yourTopic";
private static String accessId = "yourAK";
private static String accessKey = "yourAS";
private static Long datahubStartInMs = 0L;//设置消费的启动位点对应的时间。
```

② 说明 在实时计算产品上运行时请使用 **内网Endpoint**，VPC ECS Endpoint和经典网络ECS Endpoint都属于内网Endpoint，在使用时请注意：

- 如果您的环境是VPC，请使用VPC ECS Endpoint。
- 如果您的环境是OXS，请使用经典网络ECS Endpoint。

6. 在下载文件pom.xml所在的目录执行如下命令打包文件。

```
mvn clean package
```

根据您的项目设置的 artifactId，target目录下会出现 blink-datastreaming-1.0-SNAPSHOT.jar，即代表完成了开发工作。

上线

请参见 [上线完成作业上线](#)。

🔊 重要 作业上线前，请在开发页面右侧的资源配置页签，配置源表的并发数，源表并发数不能大于源表的Shard数，否则作业启动后JM（Job Manager）报错。

本示例对应的作业内容如下。

```
--完整主类名，必填，例如com.alibaba.realtimecompute.DatastreamExample。  
blink.main.class=com.alibaba.blink.datastreaming.DatahubDemo  
--作业名称。  
blink.job.name=datahub_demo  
--包含完整主类名的JAR包资源名称，多个JAR包时必填，例如blink_datastream.jar。  
blink.main.jar=${完整主类名jar包的资源名称}  
--默认statebackend配置，当作业代码没有显式配置时生效。  
state.backend.type=niagara  
state.backend.niagara.ttl.ms=129600000  
--默认checkpoint配置，当作业代码没有显式配置时生效。  
blink.checkpoint.interval.ms=180000
```

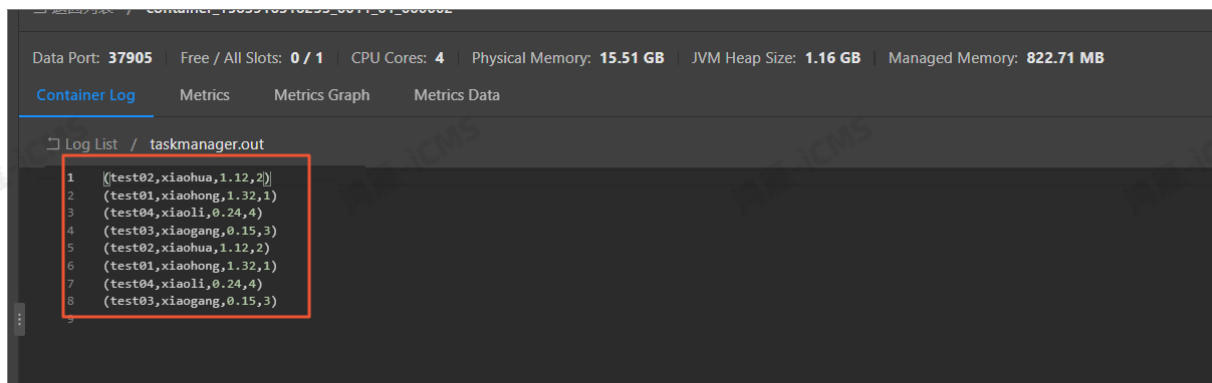
② 说明

- 注意修改blink.main.class和blink.job.name。
- 您可以设置自定义参数，详情请参见 [自定义参数](#)。

验证

在运维页面，查看Sink节点的taskmanager.out信息，本示例中使用Print作为Sink。

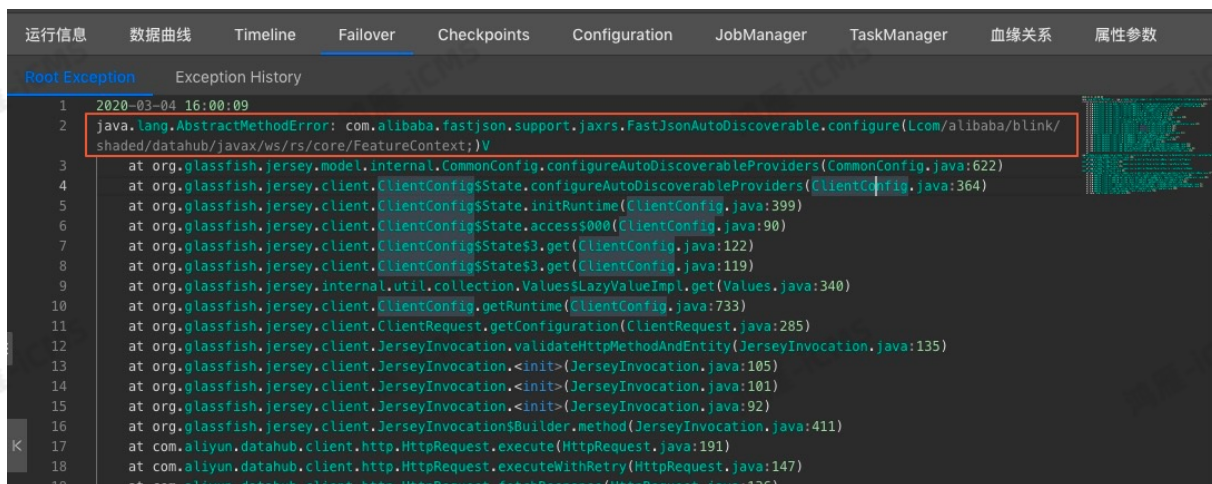
如果出现如下结果，则表示已经成功读取了阿里云DataHub中的数据。



常见问题

在作业运行时，如果界面上出现如下类似的错误，表示存在JAR包冲突。

```
java.lang.AbstractMethodError: com.alibaba.fastjson.support.jaxrs.FastjsonAutoDiscoverable.configure(Lcom/alibaba/blink/shaded/datahub/javax/ws/rs/core/FeatureContext;)
```



建议您使用 maven-shade-plugin 插件的 Relocation 功能，解决 JAR 包冲突的问题。

```
<relocations combine.self="override">
  <relocation>
    <pattern>org.glassfish.jersey</pattern>
    <shadedPattern>com.alibaba.blink.shaded.datahub.org.glassfish.jersey</shadedPattern>
  </relocation>
</relocations>
```

7.8.2. 读取Kafka数据示例

本文为您介绍如何使用Datastream作业读取阿里云Kafka数据。

前提条件

- 本地安装了Java JDK 8。
- 本地安装了Maven 3.x。

- 本地安装了用于Java或Scala开发的IDE，推荐IntelliJ IDEA，且已配置完成JDK和Maven环境。
- 创建与实时计算独享模式相同VPC的Kafka实例，并创建了Topic和Consumer Group。

背景信息

- 实时计算Datastream完全兼容开源Flink 1.5.2版本，阿里云Kafka兼容开源Kafka，因此可以直接使用Maven仓库里的Kafka Connetor来连接阿里云Kafka。
- 实时计算独享模式通过内网接入阿里云Kafka，无需进行SASL认证鉴权。如果您在本地IDE上通过公网方式接入阿里云Kafka，则需要进行SASL认证鉴权，具体配置请参见[kafka-java-demo](#)。

 **重要** 仅Blink3.x版本支持本示例。

开发

1. 下载并解压 [alikaafka-demo-master](#) 示例到本地。
2. 在IntelliJ IDEA中，单击File > Open，打开刚才解压缩完成的alikaafka-demo-master。
3. 双击打开 `\alikaafka-demo-master\src\main\resources` 目录下的kafka.properties后，修改bootstrap.servers、topic和group.id为您创建的Kafka实例对应值。

```
## 接入点，通过控制台获取。  
## 注意公网和VPC接入点的区别，在Blink独享模式下需要使用默认接入点，而非SSL接入点。  
bootstrap.servers=ip1:port,ip2:port,ip3:port  
## Topic，通过控制台创建。  
topic=your_topic  
## ConsumerGroup，通过控制台创建。  
group.id=your_groupid
```


4. 在下载文件中pom.xml所在的目录执行如下命令打包文件。

```
mvn clean package
```

根据您的项目设置的artifactId，target目录下会出现 blink-datastreaming-1.0-SNAPSHOT.jar的JAR包，即代表完成了开发工作。

上线

请参见 [上线完成作业上线](#)。

 **说明** 注意修改blink.main.class、blink.job.name和blink.main.jar。

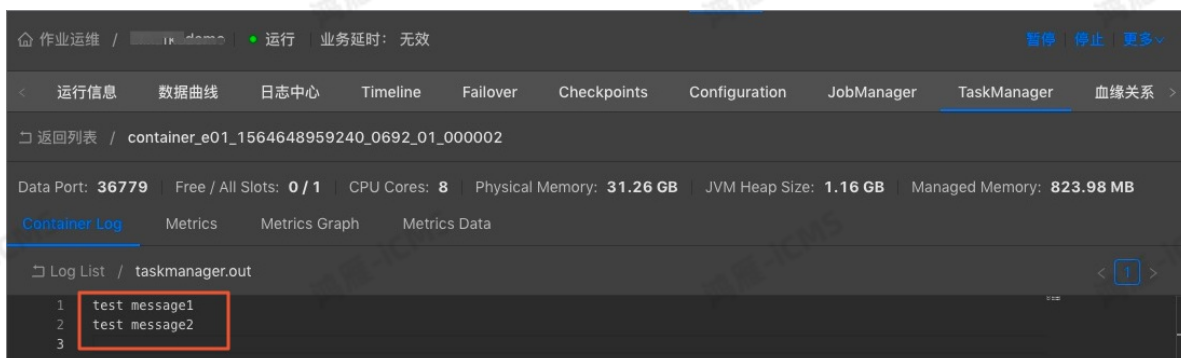
本示例对应的作业内容如下。

```
--完整主类名，必填，例如com.alibaba.realtimecompute.DatastreamExample。  
blink.main.class=com.alibaba.blink.datastreaming.AliKafkaConsumerDemo  
--作业名称。  
blink.job.name=alikafkaconsumerdemo  
--包含完整主类名的JAR包资源名称，多个JAR包时必填，例如blink_datastream.jar。  
blink.main.jar=blink-datastreaming-1.0-snapshot.jar  
--默认statebackend配置，当作业代码没有显式配置时生效。  
state.backend.type=niagara  
state.backend.niagara.ttl.ms=129600000  
--默认checkpoint配置，当作业代码没有显式配置时生效。  
blink.checkpoint.interval.ms=180000
```

说明 您可以设置自定义参数，详情请参见 [自定义参数](#)。

验证

1. 在Kafka控制台发送消息。
2. 在实时计算的运维界面，查看Sink节点的taskmanager.out输出结果，本示例中使用Print作为Sink。
出现类似如下输出（具体内容以实际发送的消息内容为准），则表示已经成功读取了阿里云Kafka中的数据。



7.8.3. 读取DataHub数据写入阿里云HBase示例

本文为您介绍如何使用Datastream作业读取DataHub数据写入HBase。

前提条件

- 本地安装了Java JDK 8。
- 本地安装了Maven 3.x。
- 本地安装了用于Java或Scala开发的IDE，推荐IntelliJ IDEA，且已配置完成JDK和Maven环境。
- 在DataHub上创建了Topic，并且Topic中存在测试数据。

说明 测试数据需要有3个字段，数据类型依次为BOOLEAN、STRING和STRING。

- 创建与实时计算独享模式同一地域下相同VPC的HBase实例，并创建表和列簇。通过Shell访问HBase集群步骤请参见[使用Shell访问](#)。

说明

- 本示例为标准版HBase。
- 实时计算集群IP需要添加至HBase白名单。

背景信息

本文以Windows系统为例进行演示。

 **重要** 仅Blink 3.x版本支持本示例。

开发

1. 下载并解压**Hbase_Demo-master**示例到本地。
2. 在Intellij IDEA中，单击File > Open，打开刚才解压缩完成的Hbase_Demo-master。
3. 双击打开 \Hbase_Demo-master\src\main\java\Hbase_Demo后，修改HbaseDemo.java文件中的DataHub与HBase相关参数。

```
//DataHub相关参数
//private static String endPoint = "public endpoint";//公网访问（填写内网Endpoint，就不用填写公网Endpoint）。
private static String endPoint = "inner endpoint";//内网访问。
private static String projectName = "yourProject";
private static String topicSourceName = "yourTopic";
private static String accessId = "yourAK";
private static String accessKey = "yourAS";
private static Long datahubStartInMs = 0L;//设置消费的启动位点对应的时间。
//Hbase相关参数
private static String zkQuorum = "yourZK";
private static String tableName = "yourTable";
private static String columnFamily = "yourcolumnFamily";
```


4. 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
```

根据您的项目设置的artifactId，target目录下会出现 Hbase_Demo-1.0-SNAPSHOT-shaded.jar的JAR包，即代表完成了开发工作。

上线

请参见[上线完成作业上线](#)。

 **说明** 作业上线前，请在开发页面右侧的资源配置页签，配置源表的并发数，源表并发数不能大于源表的Shard数，否则作业启动后JM（Job Manager）报错。

本示例对应的作业内容如下。

```
--完整主类名，必填。  
blink.main.class=Hbase_Demo.HbaseDemo  
--作业名称。  
blink.job.name=datahub_demo  
--包含完整主类名的JAR包资源名称，多个JAR包时必填。  
--blink.main.jar=Hbase_Demo-1.0-snapshot.jar  
--默认statebackend配置，当作业代码没有显式配置时生效。  
state.backend.type=niagara  
state.backend.niagara.ttl.ms=129600000  
--默认checkpoint配置，当作业代码没有显式配置时生效。  
blink.checkpoint.interval.ms=180000
```

② 说明 您可以设置自定义参数，详情请参见 [自定义参数](#)。

验证

1. 在实时计算控制台发送测试数据至DataHub。

```
CREATE TABLE kafka_src (  
  a BOOLEAN  
) WITH (  
  type = 'random'  
);  
CREATE TABLE event_logs (  
  `a` BOOLEAN,  
  b VARCHAR,  
  `c` VARCHAR  
) WITH (  
  type = 'datahub',  
  endPoint = '<yourEndpoint>',  
  project = '<yourProject>',  
  topic = '<yourTopic>',  
  accessId='<yourAccessId>',  
  accessKey='<yourAccessKey>'  
);  
INSERT INTO event_logs  
SELECT  
  a,'rowkey3' as b,'123' as c  
FROM kafka_src;
```

2. 连接HBase集群，详情请参见 [使用Shell访问](#)。
3. 执行 `scan 'hbase_sink'` 查看写入数据。

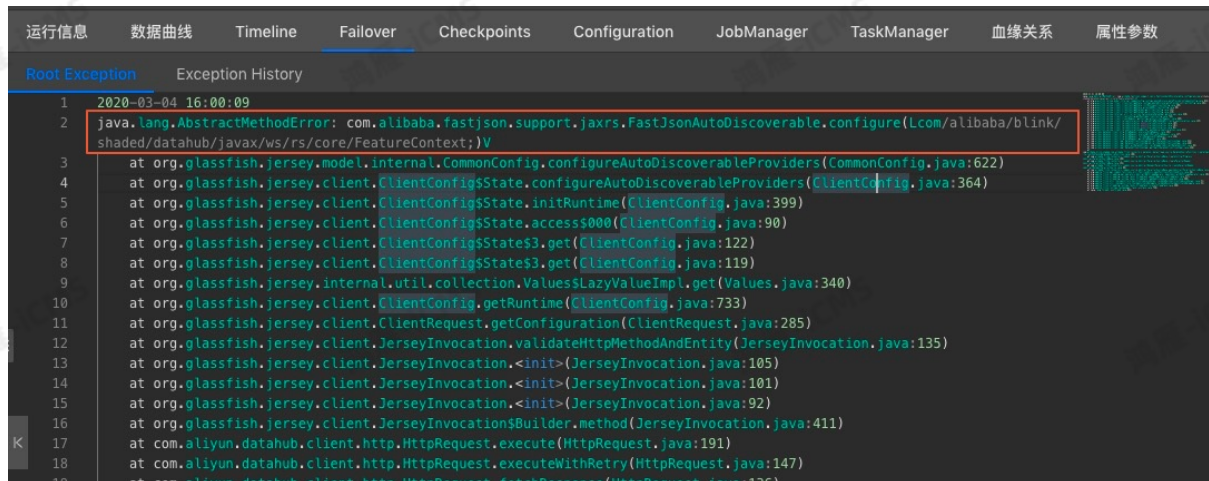
出现类似如下输出，则表示已经成功将DataHub数据写入阿里云HBase。

```
hbase(main):128:0> scan 'hbase_sink'  
ROW COLUMN+CELL  
rowkey3 column=f1:a, timestamp=1597741134871, value=[B@56bc3604  
rowkey3 column=f1:b, timestamp=1597741134871, value=[B@f5c05  
rowkey3 column=f1:c, timestamp=1597741134871, value=[B@25c03326  
1 row(s)  
Took 0.0590 seconds
```

常见问题

在作业运行时，如果界面出现如下类似的错误，表示存在JAR包冲突。

```
java.lang.AbstractMethodError: com.alibaba.fastjson.support.jaxrs.FastJsonAutoDiscoverable.configure(Lcom/alibaba/blink/shaded/datahub/javax/ws/rs/core/FeatureContext;)
```



建议您使用maven-shade-plugin插件的Relocation功能，解决JAR包冲突的问题。

```
<relocations combine.self="override">
  <relocation>
    <pattern>org.glassfish.jersey</pattern>
    <shadedPattern>com.alibaba.blink.shaded.datahub.org.glassfish.jersey</shadedPattern>
  </relocation>
</relocations>
```

7.8.4. 读取日志服务SLS示例

本文为您介绍如何使用Datastream作业读取阿里云日志服务SLS数据示例。

前提条件

- 本地安装了Java JDK 8。
- 本地安装了Maven 3.x。
- 本地安装了用于Java或Scala开发的IDE，推荐IntelliJ IDEA，且已配置完成JDK和Maven环境。
- SLS上已创建了logstore，并且logstore中存在测试数据。

背景信息

本文以Windows操作系统为例进行演示。

 **重要** 仅Blink 3.x版本支持本示例。

开发

1. 下载并解压SLS_Demo示例到本地。
2. 在IntelliJ IDEA中，单击File > Open，打开刚才解压缩完成的SLS_Demo-master。

3. 双击打开 `SLS_Demo-master\src\main\java\com\aliyun\openservices\log\flink\ConsumerSample` 后，修改 `ConsumerSample.java` 文件中的SLS的相关参数。

```
private static final String SLS_ENDPOINT = "VPC endpoint";//线上使用经典网络及VPC Endpoint
// private static final String SLS_ENDPOINT = "public endpoint";//本地测试使用 公网Endpoint
private static final String ACCESS_KEY_ID = "yourAK";
private static final String ACCESS_KEY_SECRET = "yourAS";
private static final String SLS_PROJECT = "yourProject";
private static final String SLS_LOGSTORE = "yourlogstore";
//1、启动位点秒级的时间戳读取数据;2、读取全量加增量数据Consts.LOG_BEGIN_CURSOR;
//3、读取增量数据Consts.LOG_END_CURSOR
private static final String StartInMs = Consts.LOG_END_CURSOR;
```

② 说明 IDE本地调试注意将<scope>provided</scope>注释掉。

4. 在下载文件中 `pom.xml` 所在的目录执行如下命令打包文件。

```
mvn clean package
```

根据您的项目设置的 `artifactId`，`target` 目录下会出现 `flink-log-connector-0.1.21-SNAPSHOT.jar` 的JAR包，即代表完成了开发工作。

上线

请参见 [上线完成作业上线](#)。

② 说明 作业上线前，请在开发页面右侧的资源配置页签，配置源表的并发数，源表并发数不能大于源表的Shard数，否则作业启动后JM（Job Manager）报错。

本示例对应的作业内容如下。

```
--完整主类名，必填。
--blink.main.class=com.aliyun.openservices.log.flink.ConsumerSample
--作业名称。
blink.job.name=sls
--包含完整主类名的JAR包资源名称，多个JAR包时必填。
--blink.main.jar=flink-log-connector-0.1.21-snapshot.jar
--默认statebackend配置，当作业代码没有显式配置时生效。
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
--默认checkpoint配置，当作业代码没有显式配置时生效。
blink.checkpoint.interval.ms=180000
```

② 说明 您可以设置自定义参数，详情请参见 [自定义参数](#)。

验证

在实时计算Flink版运维界面，查看Sink节点的 `taskmanager.out` 输出结果，本示例中使用Print作为Sink。

如果出现如下结果，则表示已经成功读取了阿里云SLS中的数据。



8.OpenAPI

8.1. 简介

您可以使用实时计算的OpenAPI来操作您在实时计算产品上的所有项目作业，达到和使用WebUI操作相同的效果。

使用OpenAPI时，您需要先构造一个 `DefaultAcsClient` 类的客户端，传入相关配置，通过构造的客户端来发送和接受请求相关信息

所有的OpenAPI功能均成对出现，分为 `x-request` 和 `x-response` 两类，`request` 的OpenAPI负责接受参数，通过您创建的 `client` 客户端向实时计算发送请求，请求返回的结果是对应的 `response` 类，您可以通过 `get` 方法获取该类中存储的信息。

集群地域

集群地域和OpenAPI中 `regionID` 的对照关系，如下表所示。

地域	名称 (regionID)
华东1	cn-hangzhou
华东2	cn-shanghai
华南1	cn-shenzhen
华北2	cn-beijing
华北3	cn-zhangjiakou
华东1金融云	cn-hangzhou-finance
华东2金融云	cn-shanghai-finance-1
华南1金融云	cn-shenzhen-finance-1
中国（香港）	cn-hongkong
日本（东京）	ap-northeast-1
亚太东南1（新加坡）	ap-southeast-1
亚太东南3（吉隆坡）	ap-southeast-3
美国东部1（弗吉尼亚）	us-east-1
欧洲中部1（法兰克福）	eu-central-1

机器型号

机器型号和OpenAPI中 `model` 的对应关系，如下表所示。

机器型号	名称 (model)
4核16g	Ecs_4c16g
8核32g	Ecs_8c32g
16核64g	Ecs_16c64g
24核96g	Ecs_24c96g
32核128g	Ecs_32c128g
56核224g	Ecs_56c224g
64核256g	Ecs_64c256g

8.2. OpenAPI概览

OpenAPI概览为您列举实时计算所支持的OpenAPI。

作业

OpenAPI	描述
CheckRawPlanJson	检测作业PlanJson的获取状态。
CommitJob	提交作业。
CreateJob	创建作业。
DeleteJob	删除作业。
GetJob	获取job信息。
ListJob	搜索Job。
OfflineJob	下线作业。
StartJob	启动作业。
UpdateJob	更新作业。
ValidateJob	作业语法检查。

文件夹

OpenAPI	描述
MVFolder	移动文件夹。
CreateFolder	创建文件夹。

OpenAPI	描述
CreateFolder	删除文件夹。
GetFolder	获取文件夹。
ListChildFolder	获取子目录。

实例

OpenAPI	描述
BatchGetInstanceRunSummary	批量获取实例运行信息。
GetInstanceResource	获取运行实例使用资源。
GetInstance	获取运行实例详情。
GetInstanceCheckpoint	获取运行实例的Checkpoint。
GetInstanceConfig	获取作业运行参数。
GetInstanceDetail	获取实例运行的DAG图。
GetInstanceExceptions	获取运行实例的Failover信息。
GetInstanceFinalState	获取运行实例最终状态。
GetInstanceMetric	获取运行实例的Metric信息。
GetInstanceRunSummary	获取作业运行实例的运行概要。
ListInstance	获取某个项目下所有的运行实例。
ModifyInstanceState	修改实例状态。
GetRawPlanJson	获取原始执行计划。

Package

OpenAPI	描述
CreatePackage	获取Package信息。
DeletePackage	删除Package。
GetPackage	获得Package。
ListPackage	搜索Package。
GetRefPackageJob	获取引用特定Package的作业。

OpenAPI

OpenAPI	描述
UpdatePackage	更新Package。

Queue

OpenAPI	描述
CreateQueue	创建Queue。
DeleteQueue	删除Queue。
BindQueue	绑定项目运行需要的Queue。
UnbindQueue	解绑项目运行的Queue。
GetClusterQueueInfo	获取集群中Queue的信息。
ListProjectBindQueue	查询项目绑定的队列信息。
ListProjectBindQueueResource	查询项目绑定的队列的资源信息。

Cluster

OpenAPI	描述
CreateCluster	创建集群。
ExpandCluster	后付费集群升级。
DestroyCluster	注销集群。
ShrinkCluster	集群缩容。
ListCluster	查询已存在集群信息。
ModifyMasterSpec	变更Master机型规格。
GetClusterResource	获取集群资源状态信息。
GetClusterEngineVersions	获取集群引擎版本。
GetClusterDetails	获取集群详情。

Project

OpenAPI	描述
CreateProject	创建项目。
DeleteProject	删除项目。

OpenAPI	描述
GetProject	获取项目详情。
ListProject	查询您已有的项目。

8.3. RAM资源授权

刚创建的子账号无法使用主账号的资源，您需要通过RAM授权的方式，赋予子账号操作主账号资源的权限。

默认情况下，您对自己创建的资源拥有完整的操作权限，可以使用本文中列举的OpenAPI对资源进行操作。

 **说明** 如果您不需要跨账户授权和访问实例资源，您可以跳过该章节。

授权策略（Policy）

授权策略（Policy）描述授权的具体内容。授权内容主要包含效果（Effect）、资源（Resource）、对资源所授予的操作权限（Action）以及限制条件（Condition）这几个基本元素。以下内容以OpenAPI网关为例，为您介绍授权策略（Policy）。

● 系统授权策略

OpenAPI网关已经预置了AliyunApiGatewayFullAccess和AliyunApiGatewayReadOnlyAccess两个系统权限，您可以在RAM控制台，[权限管理 > 权限策略管理](#)页面查看。

- AliyunApiGatewayFullAccess：管理员权限，拥有主账号下OpenAPI分组、OpenAPI、流控策略、应用等所有资源的管理权限。
- AliyunApiGatewayReadOnlyAccess：可以查看主账号下OpenAPI分组、OpenAPI、流控策略、应用等所有资源，但不可以操作。

● 自定义策略

您可以在RAM控制台上[权限策略管理](#)页面，自定义策略查看已经定义好的自定义授权。

您也可以根据需要自定义管理权限，支持更为精细化的授权。可以为某个操作或者某个资源授权。例如，GetUsers的编辑权限。自定义策略的查看、创建、修改和删除方法请参见[授权策略管理](#)。授权策略内容填写方法请参见[权限策略基本元素](#)、[权限策略语法和结构](#)。

● 示例

允许所有的查看操作。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "apigateway:Describe*",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```


- Action（操作名称列表）格式

"Action": "<service-name>:<action-name>"

- service-name: 阿里云产品名称，请填写 apigateway。
- action-name: OpenAPI接口名称，支持通配符 *。

"Action": "apigateway:Describe*" --表示所有的查询操作。

"Action": "apigateway:*" --表示API网关所有操作。

- Resource（操作对象列表）

- Resource通常指操作对象

OpenAPI网关中的OpenAPI分组、流控策略、应用都被称为Resource，语法格式：acs:<service-name>:<region>:<account-id>:<relative-id>。其中：

- acs: Alibaba Cloud Service的首字母缩写，表示阿里云的公共云平台。
- service-name: 阿里云产品名称，请填写 apigateway。
- region: 地区信息，可以使用通配符号 * 来表示所有区域。
- account-id: 账号ID，例如， 123456789012****，也可以使用 * 代替。
- relative-id: 与服务相关的资源描述部分，其语义由具体服务指定，该部分的格式支持树状结构（类似文件路径）。以OSS为例，表示一个OSS对象的格式为：relative-id = "mybucket/dir1/object1.jpg"。

- Resource语法

acs:apigateway:\$regionid:\$accountid:apigroup/\$groupid

- Resource示例

acs:apigateway:cn-hangzhou:195000****654:apigroup/i-xxxx

鉴权规则

以OpenAPI网关为例。

action-name	资源（Resource）
AbolishApi	acs:apigateway:\$regionid:\$accountid:apigroup/\$groupid

8.4. 调用方式

8.4.1. 签名机制

阿里云通过使用Access Key ID和Access Key Secret对称加密方法对请求发送者的身份进行验证。因此使用HTTP协议或HTTPS协议提交请求时，都需要在请求中包含签名（Signature）信息。

通过在阿里云官网申请，您可以获取由阿里云官网授予的 Access Key ID 和 Access Key Secret。

 **重要** 您的密钥信息仅用于身份验证，请勿透传给无关人员使用。

- Access Key ID：标识访问者的身份。

- **Access Key Secret**：加密签名字符串和服务器端验证签名字符串的密钥。

说明 阿里云提供了多种语言的SDK及第三方SDK，可以免去您对签名算法进行编码的工作。详情请参见[阿里云开发工具包（SDK）](#)。

签名操作

您在使用API访问阿里云时，需要按照以下步骤对请求进行签名处理。

1. 使用请求参数构造规范化的请求字符串（Canonicalized Query String）。

- i. 参数排序。按照参数名称的字典顺序对请求中所有的请求参数（包括除 **Signature** 以外的**公共请求参数**和接口的自定义参数）进行排序。

说明 当使用GET方法提交请求时的参数信息，即请求URI中的参数部分（即URI中 `?` 之后由 `&` 连接的部分）。

- ii. 参数编码。对排序之后的请求参数的名称和值分别用UTF-8字符集进行URL编码。编码的规则如下：

- 对于字符A~Z、a~z、0~9以及中划线（-）、下划线（_）、英文句号（.）、波浪号（~）不编码。
- 对于其它字符编码成 `%XY` 的格式，其中 `XY` 是字符对应ASCII码的16进制表示。例如，英文的双引号（"）对应的编码为 `%22`。
- 对于扩展的UTF-8字符，编码成 `%XY%ZA...` 的格式。
- 英文空格要编码为 `%20`，而不是加号（+）。

该编码方式和通常采用的 `application/x-www-form-urlencoded` MIME格式编码算法的实现（例如Java标准库中的 `java.net.URLEncoder`）相似，但又有所不同。实现时，可以先用标准库的方式进行编码，然后把编码后的字符串中加号（+）替换成 `%20`、星号（*）替换成 `%2A`、`%7E` 替换回波浪号（~），即可得到上述规则描述的编码字符串。该算法可以用下面的`percentEncode`方法实现。

```
private static final String ENCODING = "UTF-8";
private static String percentEncode(String value) throws UnsupportedOperationException {
    return value != null ? URLEncoder.encode(value, ENCODING).replace("+", "%20").replace("*", "%2A").replace("%7E", "~") : null;
}
```

- iii. 将编码后的参数名称和值用英文等号（=）进行连接。

- iv. 将等号连接得到的参数组合按照参数排序依次使用（&）符号连接，即得到规范化请求字符串。

2. 将上一步构造的规范化字符串按照下面的规则构造成待签名的字符串。

```
StringToSign= HTTPMethod + "&" + percentEncode( "/" ) + "&" +
    percentEncode(CanonicalizedQueryString)
```

其中：

- `HTTPMethod` 是提交请求用的HTTP方法，例如GET。
- `percentEncode("/")` 是按照参数排序中描述的URL编码规则对字符 `/` 进行编码得到的值，即 `%2F`。
- `percentEncode(CanonicalizedQueryString)` 是对步骤1中构造的规范化请求字符串按参数编码中描述的URL编码规则编码后得到的字符串。

3. 按照RFC 2104的定义，计算待签名字符串StringToSign的HMAC值。

② 说明 计算签名时使用的Key就是您持有的Access Key Secret加上一个 `&` 字符（ASCII:38），使用的哈希算法是SHA1。

- 按照Base64编码规则把上面的HMAC值编码成字符串，即得到签名值（Signature）。
- 将得到的签名值作为Signature参数添加到请求参数中，即完成对请求签名的过程。

② 说明 得到的签名值在作为最后的请求参数值提交给ECS服务器时，要和其它参数一样，按照RFC3986的规则进行URL编码。

示例

以DescribeRegions为例，假设使用的 Access Key Id 为 `testid`，Access Key Secret 为 `testsecret`，则签名前的请求URL如下所示。

```
http://ecs.aliyuncs.com/  
?TimeStamp=2016-02-23T12:46:24Z  
&Format=XML  
&AccessKeyId=testid  
&Action=DescribeRegions  
&SignatureMethod=HMAC-SHA1  
&SignatureNonce=3ee8c1b8-83d3-44af-a94f-4e0ad82fd6cf  
&Version=2014-05-26  
&SignatureVersion=1.0
```

计算得到的待签名字符串 `StringToSign` 如下所示。

```
GET  
&%2F  
&AccessKeyId%3Dtestid  
&Action%3DDescribeRegions  
&Format%3DXML&SignatureMethod%3DHMAC-SHA1  
&SignatureNonce%3D3ee8c1b8-83d3-44af-a94f-4e0ad82fd6cf  
&SignatureVersion%3D1.0  
&TimeStamp%3D2016-02-23T12%253A46%253A24Z  
&Version%3D2014-05-26
```

因为 Access Key Secret 为 `testsecret`，所以用于计算HMAC的Key为 `testsecret&`，计算得到的签名值为 `CT9X0VtwR86fNWSnsc6v8YGOjuE=`。

将签名作为Signature参数加入到URL请求中，最后得到的URL如下所示。

```
http://ecs.aliyuncs.com/  
?SignatureVersion=1.0  
&Action=DescribeRegions  
&Format=XML  
&SignatureNonce=3ee8c1b8-83d3-44af-a94f-4e0ad82fd6cf  
&Version=2014-05-26  
&AccessKeyId=testid  
&Signature=CT9X0VtwR86fNWSnsc6v8YGOjuE%3D  
&SignatureMethod=HMAC-SHA1  
&TimeStamp=2016-02-23T12%3A46%3A24Z
```

8.4.2. 请求结构

本文为您介绍OpenAPI服务的请求结构。

服务地址

API 的服务接入地址，如下所示：

地域	服务地址
杭州	foas.aliyuncs.com
cn-shanghai: 上海	foas.[RegionId].aliyuncs.com

通信协议

为了获得更高的安全性，仅支持使用HTTPS通道发送OpenAPI请求。

字符编码

请求及返回结果都使用UTF-8 字符集进行编码。


8.4.3. 返回结果

本文为您介绍OpenAPI服务的返回结果。

返回结果格式

调用OpenAPI服务后返回数据采用统一格式：

- 返回的HTTP状态码为 2xx ，代表调用成功。
- 返回的HTTP状态码为 4xx 或 5xx ，代表调用失败。

 **说明** 本文档中的返回示例为了便于用户查看，做了格式化处理，实际返回结果是没有进行换行、缩进等处理的。

返回结果成功

调用成功返回的数据格式主要有XML和JSON两种，外部系统可以在请求时传入参数来制定返回的数据格式，默认为XML格式。返回结果成功示例如下：

}

返回结果错误

调用接口出错后，将不会返回结果数据。调用方可根据每个接口对应的错误码以及下述 **公共错误码** 来定位错误原因。当调用出错时，HTTP 请求返回一个 4xx 或 5xx 的 HTTP 状态码。返回的消息体中是具体的错误代码及错误信息。另外还包含一个全局唯一的请求ID：RequestId 和一个您该次请求访问的站点ID：HostId。在调用方找不到错误原因时，可以联系阿里云客服，并提供该 HostId 和 RequestId，以便尽快为您解决问题。返回结果错误示例如下：

null

公共错误码

请参见：[公共错误码表](#)。

8.4.4. 公共参数

公共参数指的是所有接口调用都需要用到的参数，包含公共请求参数和公共返回参数两种。

公共请求参数

公共请求参数是指每个接口都需要使用到的请求参数。

名称	类型	是否必选	描述
Format	String	否	返回值的类型，支持JSON与XML。默认为XML。
Version	String	是	OpenAPI版本号。为日期形式：YYYY-MM-DD，本版本对应为2018-11-11。
AccessKeyId	String	是	阿里云颁发给用户的访问服务所用的密钥ID。
Signature	String	是	签名结果串，关于签名的计算方法，请参见 签名机制 。
SignatureMethod	String	是	签名方式，目前支持HMAC-SHA1。
Timestamp	String	是	请求的时间戳。日期格式按照ISO8601标准表示，并需要使用UTC时间。格式为：YYYY-MM-DDThh:mm:ssZ，例如，2014-05-26T12:00:00Z（为北京时间2014年5月26日20点0分0秒）。

SignatureVersion	String	是	签名算法版本，目前版本是 1.0。
SignatureNonce	String	是	唯一随机数，用于防止网络重放攻击。您在不同请求间要使用不同的随机数值
ResourceOwnerAccount	String	否	本次OpenAPI请求访问到的资源拥有者账户，即登录用户名。此参数的使用方法，详见 借助RAM实现子账号对主账号的资源访问 （只能在RAM中可对ECS资源进行授权的Action中才能使用此参数，否则访问会被拒绝）。

公共请求参数示例

```
https://foas.aliyuncs.com/
?Format=xml
&Version=2018-11-11
&Signature=Pc5WB8gokVn0xfeu%2FZV%2BiNM1dgl%3D
&SignatureMethod=HMAC-SHA1
&SignatureNonce=15215528852396
&SignatureVersion=1.0
&AccessKeyId=key-test
&Timestamp=2012-06-01T12:00:00Z
...
```

公共返回参数

您发送的每次接口调用请求，无论成功与否，系统都会返回一个唯一识别码 `RequestId`。

● XML示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!--结果的根结点-->
<接口名称+Response>
<!--返回请求标签-->
<RequestId>4C467B38-3910-447D-87BC-AC049166F216</RequestId>
<!--返回结果数据-->
</接口名称+Response>
```

● JSON示例

```
{
  "RequestId": "4C467B38-3910-447D-87BC-AC049166F216",
  /* 返回结果数据 */
}
```

8.5. 实例

8.5.1. BatchGetInstanceRunSummary

您可以通过BatchGetInstanceRunSummary接口获取指定项目下，指定类别的作业的Job的运行情况。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/runsummary HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobNames	String	Query	是	job1,job2	作业名称，多个名称时使用逗号(,)分隔。
jobType	String	Query	是	FLINK_STREAM	作业类型： <ul style="list-style-type: none"> FLINK_BATCH：批作业使用。 FLINK_STREAM：流作业使用。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	1CA9A8E9-F398-493F-83CA-A92A38501B47	请求ID。
RunSummarys	Array of RunSummary		作业运行状态详情。
RunSummary			

名称	类型	示例值	描述
ActualState	String	RUNNING	返回作业的实际运行状态： <ul style="list-style-type: none"> • WAITING：等待。 • RUNNING：正在运行。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（批作业）。
EngineJobHandler	String	application_1561366511517_113805 a10928cec91de395ff3d7593f7ff46c5	YARN中为作业分配的名称：ApplicationID JobID（JM分配的ID）。
ExpectState	String	RUNNING	作业期望状态： <ul style="list-style-type: none"> • WAITING：等待。 • RUNNING：正在运行。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（批作业）。
Id	Long	214650	运行实例ID。
InputDelay	Long	0	业务延时。
JobName	String	ss2	作业名称，多个逗号（,）分隔。
LastErrorMessage	String	error	上次错误信息。
LastErrorTime	Long	1548397575000	上次出现错误时间（时间戳，精确到毫秒）。

示例

请求示例

```
/api/v2/projects/project1/runsummary
```

正常返回示例

XML 格式


```
<RequestId>1CA9A8E9-F398-493F-83CA-A92A38501B47</RequestId>
<RunSummarys>
  <RunSummary>
    <ActualState>RUNNING</ActualState>
    <EngineJobHandler>application_1561366511517_113805|a10928cec91de395ff3d7593f7ff46c5</EngineJobHandler>
    <ExpectState>RUNNING</ExpectState>
    <JobName>ss2</JobName>
    <Id>214650</Id>
    <InputDelay>0</InputDelay>
  </RunSummary>
</RunSummarys>
```

JSON 格式

```
{
  "RequestId": "1CA9A8E9-F398-493F-83CA-A92A38501B47",
  "RunSummarys": {
    "RunSummary": [
      {
        "ActualState": "RUNNING",
        "EngineJobHandler": "application_1561366511517_113805|a10928cec91de395ff3d7593f7ff46c5",
        "ExpectState": "RUNNING",
        "JobName": "ss2",
        "Id": 214650,
        "InputDelay": 0
      }
    ]
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.2. GetInstance

您可以通过GetInstance获取运行作业的全部信息，未运行的作业调用该接口会失败。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
instanceId	Long	是	123	实例ID，可以通过ListInstance接口或者StartJob接口获取。
jobName	String	是	job1	作业名称
projectName	String	是	project1	项目名称
RegionId	String	是	cn-hangzhou-pre	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; width: fit-content;"> ? 说明 公共云用户请忽略此参数。 </div>

返回数据

名称	类型	示例值	描述
Instance	Struct		Instance详情
ActualState	String	RUNNING	Instance的实际状态： <ul style="list-style-type: none"> WAITING：等待。 RUNNING：运行中。 PAUSED：暂停。 TERMINATED：停止。 SUCCESS：成功（批作业）。 FAILED：失败（批作业）。
ApiType	String	DATASTREAM	作业类型： <ul style="list-style-type: none"> DATASTREAM：Datastream作业。 SQL：SQL作业。
AutoScaleParams	String	" {\"isOnOff\":false,\"strategyConfigure\":null,\"maxCU\":null}"	作业上线时设置的AutoScale参数。
ClusterId	String	rcmp9x37ztfb63g1x7lt****	集群ID

名称	类型	示例值	描述
Code	String	--完整主类名，必填，例如 com.alibaba.realtim ecompute.Datastr eamExample\nblin k.main.class=Hbase _Demo.HbaseDemo \n\n--包含完整主类 名的JAR包资源名称， 多个JAR包时必填，例 如 blink_datastream.jar\n-- blink.main.jar=hbas e_demo-1.0- snapshot.jar \n\n-- 默认state backend配 置，当作业代码没有 显式声明时生效 \nstate.backend.ty pe=niagara\nstate. backend.niagara.ttl .ms=129600000\n\n --默认Checkpoint配 置，当作业代码没有 显式声明时生效 \nblink.checkpoint.i nterval.ms=180000\ n\n\n--默认启用项目参 数\n-- enable.project.conf ig=true	Instance的运行代码： <ul style="list-style-type: none"> • SQL作业返回SQL代码。 • Datastream作业返回Datastream配置。
EndTime	Long	1548397575000	作业结束时间，未结束作业返回空值。
EngineJobHand ler	String	application_159765 4819348_**** 405a6 d40d7f4f2618ed532 359887****	Instance在YARN中的名称，格式为： ApplicaitonID jobID (JobManager分配的 jobid)。
EngineVersion	String	blink-3.4.4	Blink版本
ExpectState	String	RUNNING	期望状态： <ul style="list-style-type: none"> • RUNNING：运行中。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（流作业）。

名称	类型	示例值	描述
Id	Long	412536	InstanceID
InputDelay	Long	-99999999	业务延迟，单位为毫秒。
JobName	String	job1	作业名称
JobType	String	FLINK_STREAM	作业类型： <ul style="list-style-type: none"> FLINK_STREAM：流作业。 FLINK_BATCH：批作业。
LastErrorMessage	String	error	最新的报错信息
LastErrorTime	Long	1548397575000	最新的报错时间
LastOperateTime	Long	1599794334000	最新的操作时间
LastOperator	String	239960377092765296	最新的操作人
Packages	String	hbase_demo-1.0-snapshot-shaded.jar	实例引用的包名称，多个Package使用逗号（,）分隔，未引用为空。
PlanJson	String	{a:b}	执行计划
ProjectName	String	project1	项目名称

名称	类型	示例值	描述
Properties	String	#checkpoint模式： EXACTLY_ONCE 或者 AT_LEAST_ONCE\n# blink.checkpoint.m ode=EXACTLY_ONCE \n\n#checkpoint间 隔时间，单位毫秒 \n#blink.checkpoint .interval.ms=180000 \n\n#rocksdb的数据 生命周期，单位毫秒 \n#state.backend.r ocksdb.ttl.ms=1296 00000\n\n#启用项目 参数 \n#enable.project.c onfig=true\nblink.jo b.timeZone=Asia/S hanghai	作业运行参数
QueueName	String	root.project1	队列名称
StartTime	Long	1599794335000	开始时间
Starter	String	23996037709276529 6	启动人
RequestId	String	B2F5A5C5-5CCC- 45C8-B884- 7152CC345EFA	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instances/123
```

正常返回示例

XML 格式

```

<RequestId>9B7BD33D-5F3A-4C4C-BCF5-AD686CAD2370</RequestId>
<Instance>
  <ExpectState>RUNNING</ExpectState>
  <EngineVersion>blink-3.4.4</EngineVersion>
  <ProjectName>project1</ProjectName>
  <ClusterId>rcmp9x37ztfb63g1x7lt****</ClusterId>
  <PlanJson/>
  <JobName>job1</JobName>
  <StartTime>1599794335000</StartTime>
  <Properties>#checkpoint模式：EXACTLY_ONCE 或者 AT_LEAST_ONCE
#blink.checkpoint.mode=EXACTLY_ONCE
#checkpoint间隔时间，单位毫秒
#blink.checkpoint.interval.ms=180000
#rocksdb的数据生命周期，单位毫秒
#state.backend.rocksdb.ttl.ms=129600000
#启用项目参数
#enable.project.config=true
blink.job.timeZone=Asia/Shanghai</Properties>
  <Starter>239960377092765296</Starter>
  <InputDelay>-99999999</InputDelay>
  <Code>--完整主类名，必填，例如com.alibaba.realtimecompute.DatastreamExample
blink.main.class=Hbase_Demo.HbaseDemo
--包含完整主类名的JAR包资源名称，多个JAR包时必填，例如blink_datastream.jar
--blink.main.jar=hbase_demo-1.0-snapshot.jar
--默认state backend配置，当作业代码没有显式声明时生效
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
--默认Checkpoint配置，当作业代码没有显式声明时生效
blink.checkpoint.interval.ms=180000
--默认启用项目参数
--enable.project.config=true</Code>
  <ActualState>RUNNING</ActualState>
  <EngineJobHandler>application_1597654819348_0011|405a6d40d7f4f2618ed532359887d344</EngineJob
Handler>
  <LastOperator>239960377092765296</LastOperator>
  <JobType>FLINK_STREAM</JobType>
  <Packages>hbase_demo-1.0-snapshot-shaded.jar</Packages>
  <AutoScaleParams>{"isOnOff":false,"strategyConfigure":null,"maxCU":null}</AutoScaleParams>
  <ApiType>DATASTREAM</ApiType>
  <Id>412536</Id>
  <LastOperateTime>1599794334000</LastOperateTime>
  <QueueName>root.project1</QueueName>
</Instance>

```

JSON 格式

```
{
  "RequestId": "9B7BD33D-5F3A-4C4C-BCF5-AD686CAD2370",
  "Instance": {
    "ExpectState": "RUNNING",
    "EngineVersion": "blink-3.4.4",
    "ProjectName": "project1",
    "ClusterId": "rcmp9x37ztfb63g1x7lt****",
    "PlanJson": "",
    "JobName": "job1",
    "StartTime": 1599794335000,
    "Properties": "#checkpoint模式：EXACTLY_ONCE 或者 AT_LEAST_ONCE\n#blink.checkpoint.mode=EXACTLY_ONCE\n\n#checkpoint间隔时间，单位毫秒\n#blink.checkpoint.interval.ms=180000\n\n#rocksdb的数据生命周期，单位毫秒\n#state.backend.rocksdb.ttl.ms=129600000\n\n#启用项目参数\n#enable.project.config=true\n\nblink.job.timeZone=Asia/Shanghai",
    "Starter": "239960377092765296",
    "InputDelay": -99999999,
    "Code": "--完整主类名，必填，例如com.alibaba.realtimecompute.DatastreamExample\n\nblink.main.class=HbaseDemo.HbaseDemo\n\n--包含完整主类名的JAR包资源名称，多个JAR包时必填，例如blink_datastream.jar\n\n--blink.main.jar=hbase_demo-1.0-snapshot.jar\n\n--默认state backend配置，当作业代码没有显式声明时生效\n\nstate.backend.type=niagara\n\nstate.backend.niagara.ttl.ms=129600000\n\n--默认Checkpoint配置，当作业代码没有显式声明时生效\n\nblink.checkpoint.interval.ms=180000\n\n--默认启用项目参数\n\n--enable.project.config=true",
    "ActualState": "RUNNING",
    "EngineJobHandler": "application_1597654819348_0011|405a6d40d7f4f2618ed532359887d344",
    "LastOperator": "239960377092765296",
    "JobType": "FLINK_STREAM",
    "Packages": "hbase_demo-1.0-snapshot-shaded.jar",
    "AutoScaleParams": "{\"isOnOff\":false,\"strategyConfigure\":null,\"maxCU\":null}",
    "ApiType": "DATASTREAM",
    "Id": 412536,
    "LastOperateTime": 1599794334000,
    "QueueName": "root.project1"
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.3. GetInstanceResource

您可以通过GetInstanceResource OpenAPI获取运行实例信息和已使用的CPU和Memory等信息。未运行的作业调用GetInstanceResource OpenAPI会报错。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/resource HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	403936	实例ID： <ul style="list-style-type: none"> 流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。 批作业：可以通过ListInstance和StartJob等接口获得。
jobName	String	Path	是	test_0812	作业名称。
projectName	String	Path	是	bayes_team	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	1A90ED7D-6A31-4385-8372-B73A617A0D50	请求ID。
Resource	Struct		资源详情。
AllocatedMB	Long	2816	作业申请的内存。
AllocatedVirtualCores	Long	125	作业申请的Vcore。
TotalMB	Long	1024	实际运行使用的内存。
TotalVirtualCores	Long	50	实际运行使用的Vcore。

示例

请求示例


```
/api/v2/projects/bayes_team/jobs/test_0812/instances/412532/resource
```

正常返回示例

XML 格式

```
<RequestId>1A90ED7D-6A31-4385-8372-B73A617A0D50</RequestId>
<Resource>
  <AllocatedMB>2816</AllocatedMB>
  <AllocatedVirtualCores>125</AllocatedVirtualCores>
</Resource>
```

JSON 格式

```
{
  "RequestId": "1A90ED7D-6A31-4385-8372-B73A617A0D50",
  "Resource": {
    "AllocatedMB": 2816,
    "AllocatedVirtualCores": 125
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.4. GetInstanceCheckpoint

您可以通过GetInstanceCheckpoint API获取正在运行实例作业的Checkpoint，非运行状态的作业调用此API会返回错误。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/checkpoints HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
----	----	----	------	-----	----

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	403936	实例ID： <ul style="list-style-type: none"> 流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。 批作业：可以通过ListInstance、StartJob等接口获得。
jobName	String	Path	是	test_0812	作业名称。
projectName	String	Path	是	bayes_team	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
		<pre>{ "counts": { "restored": 1, "total": 61440, "in_progress": 0, "completed": 61440, "failed": 0 }, "summary": { "state_size": { "min": 4384, "max": 4384, "avg": 4384 }, "end_to_end_duration": { "min": 9, "max": 981, "avg": 15 }, "alignment_buffered": { "min": 0, "max": 0, "avg": 0 } }, "latest": { "completed": { "@class": "completed", "id": 61440, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 16105450869, "latest_ack_timestamp": 1610505450883, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 14, "alignment_b</pre>	

名称	类型	示例值	描述
		<pre> offered":0,"num_subtasks":1,"num_ _acknowledged_su btasks":1,"tasks" : {},\external_path" :"\hdfs://et2cloudp rod/blink_rest_serv er/root.bayes_team /test_0812/chk- 61440\","discarde d":false},\savepoi nt":null,\failed": null,\restored": {"id":7965,\resto re_timestamp":160 0879950869,\is_sa vepoint":false,\ex ternal_path":\hdf s://et2cloudprod/bl ink_rest_server/root .bayes_team/test_0 812/chk- 7965\}},\history": [{"@class":\comp leted","id":61440 ,\status":\COMPL ETED","is_savepoi nt":false,\trigger_ timestamp":16105 05450869,\latest_a ck_timestamp":161 0505450883,\state _size":4384,\full_s tate_size":4384,\e nd_to_end_duration \":14,\alignment_b offered":0,"num_ subtasks":1,"num _acknowledged_su btasks":1,"tasks" : {},\external_path" :"\hdfs://et2cloudp rod/blink_rest_serv er/root.bayes_team /test_0812/chk- 61440\","discarde d":false}, {"@class":\compl eted","id":61439, \status":\COMPL ETED","is_savepoi nt":false,\trigger_ timestamp":16105 05270869,\latest_a </pre>	

名称	类型	示例值	描述
		<pre> ck_timestamp\":1610505270888,\"state_size\":4384,\"full_state_size\":4384,\"end_to_end_duration\":19,\"alignment_buffed\":0,\"num_subtasks\":1,\"num_acknowledged_subtasks\":1,\"tasks\": : {,\"external_path\": \"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61439\", \"discarded\":true}, { \"@class\": \"completed\", \"id\":61438, \"status\": \"COMPLETED\", \"is_savepoint\":false, \"trigger_timestamp\":1610505090869, \"latest_ack_timestamp\":1610505090882, \"state_size\":4384, \"full_state_size\":4384, \"end_to_end_duration\":13, \"alignment_buffed\":0, \"num_subtasks\":1, \"num_acknowledged_subtasks\":1, \"tasks\": : {,\"external_path\": \"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61438\", \"discarded\":true}, { \"@class\": \"completed\", \"id\":61437, \"status\": \"COMPLETED\", \"is_savepoint\":false, \"trigger_timestamp\":1610504910869, \"latest_ack_timestamp\":1610504910882, \"state_size\":4384, \"full_state_size\":4384, \"end_to_end_duration\":13, \"alignment_b </pre>	

名称	类型	示例值	描述
Checkpoints	String	<pre> { "offered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": { "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61437", "discarded": true, "@class": "completed", "id": 61436, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504730869, "latest_ack_timestamp": 1610504730882, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 13, "alignment_b": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": { "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61436", "discarded": true, "@class": "completed", "id": 61435, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504550869, "latest_ack_timestamp": 1610504550883, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 14, "alignment_b": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": { "external_path": </pre>	Checkpoint详情。

名称	类型	示例值	描述
		<pre> :\hdfs://et2cloudp rod/blink_rest_serv er/root.bayes_team /test_0812/chk- 61435\", \"discarde d\":true}, { \"@class\": \"compl eted\", \"id\":61434, \"status\": \"COMPL ETED\", \"is_savepoi nt\":false, \"trigger_ timestamp\":16105 04370869, \"latest_a ck_timestamp\":161 0504370885, \"state _size\":4384, \"full_s tate_size\":4384, \"e nd_to_end_duration \":16, \"alignment_b uffered\":0, \"num_ subtasks\":1, \"num_ acknowledged_su btasks\":1, \"tasks\" : { }, \"external_path\" :\hdfs://et2cloudp rod/blink_rest_serv er/root.bayes_team /test_0812/chk- 61434\", \"discarde d\":true}, { \"@class\": \"compl eted\", \"id\":61433, \"status\": \"COMPL ETED\", \"is_savepoi nt\":false, \"trigger_ timestamp\":16105 04190869, \"latest_a ck_timestamp\":161 0504190882, \"state _size\":4384, \"full_s tate_size\":4384, \"e nd_to_end_duration \":13, \"alignment_b uffered\":0, \"num_ subtasks\":1, \"num_ acknowledged_su btasks\":1, \"tasks\" : { }, \"external_path\" :\hdfs://et2cloudp rod/blink_rest_serv er/root.bayes_team /test_0812/chk- 61433\", \"discarde d\":true}, </pre>	

名称	类型	示例值	描述
		<pre>{ "@class": "completed", "id": "61432", "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504010869, "latest_ack_timestamp": 1610504010896, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 27, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": [{ "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61432", "discarded": true }] }</pre>	
RequestId	String	<pre>/test_0812/chk-A0623FAD-D4D7-2DFA-B126-B730C849338ED</pre>	请求ID。

示例

请求示例

```
/api/v2/projects/bayes_team/jobs/test_0812/instances/403936/checkpoints
```

正常返回示例

XML 格式


```
<RequestId>A0623FAD-D4D7-4DFA-B126-D730C84338ED</RequestId>
<Checkpoints>{"counts":{"restored":1,"total":61440,"in_progress":0,"completed":61440,"failed":0},"summary":{"state_size":{"min":4384,"max":4384,"avg":4384},"end_to_end_duration":{"min":9,"max":981,"avg":15},"alignment_buffered":{"min":0,"max":0,"avg":0},"latest":{"@class":"completed","id":61440,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610505450869,"latest_ack_timestamp":1610505450883,"state_size":4384,"full_state_size":4384,"end_to_end_duration":14,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61440","discarded":false},"savepoint":null,"failed":null,"restored":{"id":7965,"restore_timestamp":1600879950869,"is_savepoint":false,"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-7965"},"history":[{"@class":"completed","id":61440,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610505450869,"latest_ack_timestamp":1610505450883,"state_size":4384,"full_state_size":4384,"end_to_end_duration":14,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61440","discarded":false},{@class":"completed","id":61439,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610505270869,"latest_ack_timestamp":1610505270888,"state_size":4384,"full_state_size":4384,"end_to_end_duration":19,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61439","discarded":true},{@class":"completed","id":61438,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610505090869,"latest_ack_timestamp":1610505090882,"state_size":4384,"full_state_size":4384,"end_to_end_duration":13,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61438","discarded":true},{@class":"completed","id":61437,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610504910869,"latest_ack_timestamp":1610504910882,"state_size":4384,"full_state_size":4384,"end_to_end_duration":13,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61437","discarded":true},{@class":"completed","id":61436,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610504730869,"latest_ack_timestamp":1610504730882,"state_size":4384,"full_state_size":4384,"end_to_end_duration":13,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61436","discarded":true},{@class":"completed","id":61435,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610504550869,"latest_ack_timestamp":1610504550883,"state_size":4384,"full_state_size":4384,"end_to_end_duration":14,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61435","discarded":true},{@class":"completed","id":61434,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610504370869,"latest_ack_timestamp":1610504370885,"state_size":4384,"full_state_size":4384,"end_to_end_duration":16,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61434","discarded":true},{@class":"completed","id":61433,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610504190869,"latest_ack_timestamp":1610504190882,"state_size":4384,"full_state_size":4384,"end_to_end_duration":13,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61433","discarded":true},{@class":"completed","id":61432,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610504010869,"latest_ack_timestamp":1610504010896,"state_size":4384,"full_state_size":4384,"end_to_end_duration":27,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61432","discarded":true},{@class":"completed","id":61431,"status":"COMPLETED","is_savepoint":false,"trigger_timestamp":1610503830869,"latest_ack_timestamp":1610503830890,"state_size":4384,"full_state_size":4384,"end_to_end_duration":21,"alignment_buffered":0,"num_subtasks":1,"num_acknowledged_subtasks":1,"tasks":{},"external_path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61431","discarded":true}}}</Checkpoints>
```

JSON 格式

```

{
  "RequestId": "A0623FAD-D4D7-4DFA-B126-D730C84338ED",
  "Checkpoints": [{"counts": {"restored": 1, "total": 61440, "in_progress": 0, "completed": 61440, "failed": 0}, "summary": {"state_size": {"min": 4384, "max": 4384, "avg": 4384}, "end_to_end_duration": {"min": 9, "max": 981, "avg": 15}, "alignment_buffered": {"min": 0, "max": 0, "avg": 0}}, "latest": {"completed": {"@class": "completed", "id": 61440, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610505450869, "latest_ack_timestamp": 1610505450883, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 14, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61440", "discarded": false}, "savepoint": null, "failed": null, "restored": {"id": 7965, "restore_timestamp": 1600879950869, "is_savepoint": false, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-7965"}}, {"history": [{"@class": "completed", "id": 61440, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610505450869, "latest_ack_timestamp": 1610505450883, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 14, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61440", "discarded": false}, {"@class": "completed", "id": 61439, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610505270869, "latest_ack_timestamp": 1610505270888, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 19, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61439", "discarded": true}, {"@class": "completed", "id": 61438, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610505090869, "latest_ack_timestamp": 1610505090882, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 13, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61438", "discarded": true}, {"@class": "completed", "id": 61437, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504910869, "latest_ack_timestamp": 1610504910882, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 13, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61437", "discarded": true}, {"@class": "completed", "id": 61436, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504730869, "latest_ack_timestamp": 1610504730882, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 13, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61436", "discarded": true}, {"@class": "completed", "id": 61435, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504550869, "latest_ack_timestamp": 1610504550883, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 14, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61435", "discarded": true}, {"@class": "completed", "id": 61434, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504370869, "latest_ack_timestamp": 1610504370885, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 16, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61434", "discarded": true}, {"@class": "completed", "id": 61433, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504190869, "latest_ack_timestamp": 1610504190882, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 13, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61433", "discarded": true}, {"@class": "completed", "id": 61432, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610504010869, "latest_ack_timestamp": 1610504010896, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 27, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61432", "discarded": true}, {"@class": "completed", "id": 61431, "status": "COMPLETED", "is_savepoint": false, "trigger_timestamp": 1610503830869, "latest_ack_timestamp": 1610503830890, "state_size": 4384, "full_state_size": 4384, "end_to_end_duration": 21, "alignment_buffered": 0, "num_subtasks": 1, "num_acknowledged_subtasks": 1, "tasks": {}, "external_path": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/chk-61431", "discarded": true}]}]}

```

```

    "discarded": true,
  }
}

```

错误码

访问[错误中心](#)查看更多错误码。

8.5.5. GetInstanceConfig

您可以通过GetInstanceConfig OpenAPI获取作业实例运行时的具体参数，非运行状态的作业调用此API会返回错误。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/config HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	512532	实例ID： <ul style="list-style-type: none"> 流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。 批作业：可以通过ListInstance和StartJob等接口获得。
jobName	String	Path	是	test_0812	作业名称。
projectName	String	Path	是	bayes_team	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	DBBB9DE4-97F5-4BB4-9931-16FE88603D36	请求ID
		<pre>{ "jid": "166845b9753a86282144c59d0c3d82b7", "name": "test_0812", "execution-config": { "execution-mode": "PIPELINE D", "restart-strategy": "Cluster level default restart strategy", "job-parallelism": -1, "object-reuse-mode": true, "user-config": { "env.java.opts.jobmanager": "-Xloggc:<LOG_DIR>/jobmanager-gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=2 -XX:GCLogFileSize=512M", "env.java.opts": "-verbose:gc -XX:NewRatio=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4", "metrics.reporter.kmonitor.class": "com.alibaba.blink.monitor.RichKmonitorReporter", "sharedslot-enabled.default": "false", "job.app-master-core": "0.25", "high-availability.zookeeper.client.session-timeout": "120000", "yarn.container-launcher-number": "100", </pre>	

名称	类型	资源.cpu.scale 示例值	描述
		"resource.cpu.scale-up.ratio": "1.5", "resource.cpu.scale-down.ratio": "1.5", "high-availability.storageDir": "hdfs://et2cloudprod/tmp/blink3.0/ha", "state.backend.niagara.ttl.ms": "12960000", "state.backend.niagara.checkpointpath": "hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/", "taskmanager.network.memory.buffer-per-external-blocking-channel": "16", "__inner_queueName__": "root.bayes_team", "metrics.reporters": "kmonitor,jmx", "metrics.reporter.kmonitor.interval": "10 SECONDS", "healthmonitor.high-cpu-detector.severe.threshold": "0.9", "yarn.per-job-cluster.include-user-jar": "LAST", "parallelism.scale.ratio.max": "4", "blink.job.source.startTime": "2020-09-07 10:34:05", "stsAccesskey": "v91V042KRCL7mFEYfIP02INwSv71m0Jz18RUsNbRzFM=", "healthmonitor.high-cpu-detector.threshold": "0.9", "state.backend.local-recovery": "true", "__inner_clusterName__": "et2cloudprod", "high-availability.zookeeper	

名称	类型	示例值	描述
Config	String	<pre>perQuorum\": \"zk. et2cloudprod.blink. alibaba- inc.com:12181\", \"r estart- strategy.fixed- delay.delay\": \"10 s\", \"metrics.proces s- tree.tm.enabled\": \" true\", \"metrics.pr ocess- tree.tm.update- interval\": \"10000\" , \"metrics.reporter.j mx.class\": \"org.ap ache.flink.metrics.j mx.JMXReporter\", \" slot.enable- shared- slot\": \"false\", \"ya rn.vcore- ratio\": \"100\", \"bli nk.checkpoint.exter nal.cleanup\": \"RET AIN_ON_CANCELLAT ION\", \"healthman ager.enabled\": \"tr ue\", \"metrics.repo rter.jmx.port\": \"10 25- 10000\", \"taskman ager.network.reque st- backoff.max\": \"30 0000\", \"akka.watc h.heartbeat.interval \": \"10 s\", \"parallelism.sc ale.ratio.min\": \"2\" , \"blink.job.timeZo ne\": \"Asia/Shangh ai\", \"metrics.repor ter.kmonitor.tag.val ue.length\": \"64\", \" yarn.application- attempts\": \"100\", \"_inner_zk_quoru m_\": \"zk.et2cloud prod.blink.alibaba- inc.com:12181\", \" akka.ask.timeout\": \"120 s\", \"stsAccessId\": \"9juusi0kMFGpRs8 c\", \"task.external.s</pre>	作业运行的相关参数

名称	类型	示例值	描述
		huffle.max-concurrent-requests\": \"512\", \\\\"blink.job.checkpoint.dir\": \"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/\", \\\\"blink.job.source.startTime.mills\": \"1599446045306\", \\\\"blink.checkpoint.interval.ms\": \"180000\", \\\\"taskmanager.network.memoryfloating-buffers-per-gate\": \"256\", \\\\"_inner__projectName\": \"bayes_team\", \\\\"web.timeout\": \"120000\", \\\\"jobmanager.heap.mb\": \"1024\", \\\\"action.selector.rescale.resource.first\": \"true\", \\\\"statebackend.onheap\": \"false\", \\\\"restart-strategy.fixed-delay.attempts\": \"2147483647\", \\\\"stsUid\": \"1709064687573327\", \\\\"jobmanager.execution.attempts-history-size\": \"100\", \\\\"slotmanager.taskmanager.timeout\": \"600000\", \\\\"task.external.huffle.compression.enable\": \"true\", \\\\"_inner__jobName\": \"test_0812\", \\\\"akka.framesize\": \"102400kB\", \\\\"state.checkpoints.create-subdirs\": \"false\", \\\\"high-availability\": \"zookeeper\", \\\\"blink.checkpoint.path\": \"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/\", \\\\"stsRegionId\": \"cn-	

名称	类型	示例值	描述
		<pre>shanghai\", \"blink.checkpoint.mode\": \"EXACTLY_ONCE\", \"restart-strategy\": \"fixed-delay\", \"blink.checkpoint.external.path\": \"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/external_cp\", \"env.java.opts.taskmanager\": \"-Xloggc: <LOG_DIR>/taskmanager-gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=2 -XX:GCLogFileSize=512M\"}}}</pre>	

示例

请求示例

```
/api/v2/projects/bayes_team/jobs/test_0812/instances/512532/config
```

正常返回示例

[XML](#) 格式


```
<RequestId>DBBB9DE4-97F5-4BB4-9931-16FE88603D36</RequestId>
<Config>{"jid":"166845b9753a86282144c59d0c3d82b7","name":"test_0812","execution-config":{"execution-mode":"PIPELINED","restart-strategy":"Cluster level default restart strategy","job-parallelism":-1,"object-reuse-mode":true,"user-config":{"env.java.opts.jobmanager:"-Xloggc:&lt;LOG_DIR&gt;/jobmanager-gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=2 -XX:GCLogFileSize=512M","env.java.opts:"-verbose:gc -XX:NewRatio=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4"},"metrics.reporter.kmonitor.class":"com.alibaba.blink.monitor.RichKmonitorReporter","sharedslot-enabled.default":"false","job.app-master-core":"0.25","high-availability.zookeeper.client.session-timeout":"120000","yarn.container-launcher-number":"100","resource.cpu.scale-up.ratio":"1.5","resource.cpu.scale-down.ratio":"1.5","high-availability.storageDir":"hdfs://et2cloudprod/tmp/blink3.0/ha","state.backend.niagara.ttl.ms":"12960000","state.backend.niagara.checkpointpath":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/" ,"taskmanager.network.memory.buffers-per-external-blocking-channel":"16","__inner__queueName__":"root.bayes_team","metrics.reporters":"kmonitor,jmx","metrics.reporter.kmonitor.interval":"10 SECONDS","healthmonitor.high-cpu-detector.severe.threshhold":"0.9","yarn.per-job-cluster.include-user-jar":"LAST","parallelism.scale.ratio.max":"4","blink.job.source.startTime":"2020-09-07 10:34:05","stsAccesskey":"v91V042KRCL7mFEYfIP02INwSv71m0Jz18RUsNbRzFM=","healthmonitor.high-cpu-detector.threshhold":"0.9","state.backend.local-recovery":"true","__inner__clusterName__":"et2cloudprod","high-availability.zookeeper.quorum":"zk.et2cloudprod.blink.alibaba-inc.com:12181","restart-strategy.fixed-delay.delay":"10 s","metrics.process-tree.tm.enabled":"true","metrics.process-tree.tm.update-interval":"10000","metrics.reporter.jmx.class":"org.apache.flink.metrics.jmx.JMXReporter","slot.enable-shared-slot":"false","yarn.vcore-ratio":"100","blink.checkpoint.external.cleanup":"RETAIN_ON_CANCELLATION","healthmanager.enabled":"true","metrics.reporter.jmx.port":"1025-10000","taskmanager.network.request-backoff.max":"300000","akka.watch.heartbeat.interval":"10 s","parallelism.scale.ratio.min":"2","blink.job.timeZone":"Asia/Shanghai","metrics.reporter.kmonitor.tag.value.length":"64","yarn.application-attempts":"100","__inner__zk_quorum__":"zk.et2cloudprod.blink.alibaba-inc.com:12181","akka.ask.timeout":"120 s","stsAccessId":"9juusi0kMFGpRs8c","task.external.shuffle.max-concurrent-requests":"512","blink.job.checkpoint.dir":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/" ,"blink.job.source.startTime.mills":"1599446045306","blink.checkpoint.interval.ms":"180000","taskmanager.network.memory.floating-buffers-per-gate":"256","__inner__projectName__":"bayes_team","web.timeout":"120000","jobmanager.heap.mb":"1024","action.selector.rescale.resource.first":"true","statebackend.onheap":"false","restart-strategy.fixed-delay.attempts":"2147483647","stsUid":"1709064687573327","jobmanager.execution.attempts-history-size":"100","slotmanager.taskmanager-timeout":"600000","task.external.shuffle.compression.enable":"true","__inner__jobName__":"test_0812","akka.framesize":"102400kB","state.checkpoints.create-subdirs":"false","high-availability":"zookeeper","blink.checkpoint.path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/" ,"stsRegionId":"cn-shanghai","blink.checkpoint.mode":"EXACTLY_ONCE","restart-strategy":"fixed-delay","blink.checkpoint.external.path":"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/external_cp","env.java.opts.taskmanager:"-Xloggc:&lt;LOG_DIR&gt;/taskmanager-gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=2 -XX:GCLogFileSize=512M"}}}</Config>
```

JSON 格式

```
{
  "RequestId": "DBBB9DE4-97F5-4BB4-9931-16FE88603D36",
  "Config": "{\"jid\":\"166845b9753a86282144c59d0c3d82b7\",\"name\":\"test_0812\",\"execution-config\":{\"execution-mode\":\"PIPELINED\",\"restart-strategy\":\"Cluster level default restart strategy\",\"job-parallelism\":-1,\"object-reuse-mode\":true,\"user-config\":{\"env.java.opts.jobmanager\":\"-Xloggc:<LOG_DIR>/jobmanager-gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=2 -XX:GCLogFileSize=512M\",\"env.java.opts\":\"-verbose:gc -XX:NewRatio=3 -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4\",\"metrics.reporter.kmonitor.class\":\"com.alibaba.blink.monitor.RichKmonitorReporter\",\"sharedslot-enabled.default\":\"false\",\"job.app-master-core\":\"0.25\",\"high-availability.zookeeper.client.session-timeout\":\"120000\",\"yarn.container-launcher-number\":\"100\",\"resource.cpu.scale-up.ratio\":\"1.5\",\"resource.cpu.scale-down.ratio\":\"1.5\",\"high-availability.storageDir\":\"hdfs://et2cloudprod/tmp/blink3.0/ha\",\"state.backend.niagara.ttl.ms\":\"12960000\",\"state.backend.niagara.checkpointpath\":\"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/\",\"taskmanager.network.memory.buffers-per-external-blocking-channel\":\"16\",\"__inner__queueName\":\"root.bayes_team\",\"metrics.reporters\":\"kmonitor,jmx\",\"metrics.reporter.kmonitor.interval\":\"10 SECONDS\",\"healthmonitor.high-cpu-detector.severe.threshold\":\"0.9\",\"yarn.per-job-cluster.include-user-jar\":\"LAST\",\"parallelism.scale.ratio.max\":\"4\",\"blink.job.source.startTime\":\"2020-09-07 10:34:05\",\"stsAccesskey\":\"v91V042KRCL7mFEYfIP02INwSv71m0Jz18RUsNbrZFM\",\"healthmonitor.high-cpu-detector.threshold\":\"0.9\",\"state.backend.local-recovery\":\"true\",\"__inner__clusterName\":\"et2cloudprod\",\"high-availability.zookeeper.quorum\":\"zk.et2cloudprod.blink.alibaba-inc.com:12181\",\"restart-strategy.fixed-delay.delay\":\"10 s\",\"metrics.process-tree.tm.enabled\":\"true\",\"metrics.process-tree.tm.update-interval\":\"10000\",\"metrics.reporter.jmx.class\":\"org.apache.flink.metrics.jmx.JMXReporter\",\"slot.enable-shared-slot\":\"false\",\"yarn.vcore-ratio\":\"100\",\"blink.checkpoint.external.cleanup\":\"RETAIN_ON_CANCELLATION\",\"healthmanager.enabled\":\"true\",\"metrics.reporter.jmx.port\":\"1025-10000\",\"taskmanager.network.request-backoff.max\":\"300000\",\"akka.watch.heartbeat.interval\":\"10 s\",\"parallelism.scale.ratio.min\":\"2\",\"blink.job.timeZone\":\"Asia/Shanghai\",\"metrics.reporter.kmonitor.tag.value.length\":\"64\",\"yarn.application-attempts\":\"100\",\"__inner__zk_quorum\":\"zk.et2cloudprod.blink.alibaba-inc.com:12181\",\"akka.ask.timeout\":\"120 s\",\"stsAccessId\":\"9juusi0kMFGpRs8c\",\"task.external.shuffle.max-concurrent-requests\":\"512\",\"blink.job.checkpoint.dir\":\"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/\",\"blink.job.source.startTime.mills\":\"1599446045306\",\"blink.checkpoint.interval.ms\":\"180000\",\"taskmanager.network.memory.floating-buffers-per-gate\":\"256\",\"__inner__projectName\":\"bayes_team\",\"web.timeout\":\"120000\",\"jobmanager.heap.mb\":\"1024\",\"action.selector.rescale.resource.first\":\"true\",\"statebackend.onheap\":\"false\",\"restart-strategy.fixed-delay.attempts\":\"2147483647\",\"stsUid\":\"1709064687573327\",\"jobmanager.execution.attempts-history-size\":\"100\",\"slotmanager.taskmanager-timeout\":\"600000\",\"task.external.shuffle.compression.enable\":\"true\",\"__inner__jobName\":\"test_0812\",\"akka.framesize\":\"102400kB\",\"state.checkpoints.create-subdirs\":\"false\",\"high-availability\":\"zookeeper\",\"blink.checkpoint.path\":\"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/\",\"stsRegionId\":\"cn-shanghai\",\"blink.checkpoint.mode\":\"EXACTLY_ONCE\",\"restart-strategy\":\"fixed-delay\",\"blink.checkpoint.external.path\":\"hdfs://et2cloudprod/blink_rest_server/root.bayes_team/test_0812/external_cp\",\"env.java.opts.taskmanager\":\"-Xloggc:<LOG_DIR>/taskmanager-gc.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=2 -XX:GCLogFileSize=512M\"}}}"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.6. GetInstanceDetail

您可以通过GetInstanceDetail OpenAPI获取实例运行的DAG图，以JSON的形式返回。非运行状态的作业调用此API会返回错误。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/details HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	403936	实例ID： <ul style="list-style-type: none">流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。批作业：可以通过ListInstance和StartJob等接口获得。
jobName	String	Path	是	test_0812	作业名称。
projectName	String	Path	是	bayes_team	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
		<pre>{\"jid\": \"166845b9753a86282144c59d0c3d82b7\", \"name\": \"test_0812\", \"isStoppable\": false, \"state\": \"CANCELED\", \"start-time\": 1600879950827, \"end-time\": 1610505455432, \"duration\": 9625504605, \"now\": 1610505455494, \"timestamp\":</pre>	

名称	类型	示例值	描述
		<pre> {"CREATED":1600879950827,"RUNNING":1600879950869,"RECONCILING":0,"FAILING":0,"FINISHED":0,"CANCELING":1610505454138,"RESTARTING":1600879940762,"SUSPENDING":0,"SUSPENDED":0,"FAILED":0,"CANCELED":1610505455432},"vertices":{"id":"717c7b8afebbfb7137f6f0f99beb2a94","topology":{"source":{"RandomSource-test_input-Stream->SourceConversion(table:[builtin,default],_DataStreamTable_0,source:[RandomSource-test_input]),fields:(s,rowtime)->WatermarkAssigner(rowtime:rowtime,offset:4000)->Calc(select:(sASc))->SinkConversiontoTuple2->FlatMap->Sink:PrinttoSystem.out},"parallelism":1,"status":"CANCELED"},"start-time":1600880000174,"end-time":1610505455427,"duration":9625455253,"tasks":{"CANCELING":0,"RUNNING":0,"SCHEDULED":0,"RECONCILING":0,"DEPLOYING":0,"CREATED":0,"CANCELED":1,"FINISHED":0,"FAILED":0},"met </pre>	

名称	类型	示例值	描述
Detail	String	<pre> { "read-bytes-complete": true, "write-bytes-complete": true, "read-records-complete": true, "write-records-complete": true, "buffers-in-pool-usage-max": 0.0, "buffers-out-pool-usage-max": 0.0, "tps-complete": true, "tps-delay-complete": true, "status-counts": { "CANCELING": 0, "RUNNING": 0, "SCHEDULED": 0, "RECONCILING": 0, "DEPLOYING": 0, "CREATED": 0, "CANCELED": 1, "FINISHED": 0, "FAILED": 0 }, "plan": { "jid": "166845b9753a86282144c59d0c3d82b7", "name": "test_0812", "nodes": [{ "id": "717c7b8afebbf7137f6f0f99beb2a94", "parallelism": 1, "operator": "\", "operator_strategy": "\", "description": "Source: RandomSource-test_input-Stream - </pre>	作业运行的DAG信息

名称	类型	> 示例值	描述
		SourceConversion(able:[builtin, default, _DataStreamTable_0, source:[RandomSource-test_input]], fields:(s, rowtime)) -> WatermarkAssigner(rowtime: rowtime, offset: 4000) -> Calc(select: (s AS c)) -> SinkConversion to Tuple2 -> Flat Map -> Sink: Print to System.out\,\"optimizer_properties\":	
RequestId	String	0F099DE0-9E70-4E57-9B16-5CBDB10211FB	请求ID。

示例

请求示例

```
/api/v2/projects/bayes_team/jobs/test_0812/instances/412532/details
```

正常返回示例

XML 格式

```
<RequestId>2FC99DE0-9E70-4E57-9B16-5CBDB10211FB</RequestId>
<Detail>{"jid":"166845b9753a86282144c59d0c3d82b7","name":"test_0812","isStoppable":false,"state":"CAN
CELED","start-time":1600879950827,"end-
time":1610505455432,"duration":9625504605,"now":1610505455494,"timestamps":{"CREATED":1600879950
827,"RUNNING":1600879950869,"RECONCILING":0,"FAILING":0,"FINISHED":0,"CANCELLING":1610505454138,"
RESTARTING":1600879940762,"SUSPENDING":0,"SUSPENDED":0,"FAILED":0,"CANCELED":1610505455432},"ve
rtices":[{"id":"717c7b8afebbfb7137f6f0f99beb2a94","topology-id":0,"name":"Source: RandomSource-test_in
put-Stream -&gt; SourceConversion(table:[builtin, default, _DataStreamTable_0, source: [RandomSource-te
st_input]], fields:(s, rowtime)) -&gt; WatermarkAssigner(rowtime: rowtime, offset: 4000) -&gt; Calc(select: (
s AS c)) -&gt; SinkConversion to Tuple2 -&gt; Flat Map -&gt; Sink: Print to System.out","parallelism":1,"status
":"CANCELED","start-time":1600880000174,"end-time":1610505455427,"duration":9625455253,"tasks":{"CA
NCELING":0,"RUNNING":0,"SCHEDULED":0,"RECONCILING":0,"DEPLOYING":0,"CREATED":0,"CANCELED":1,"FIN
ISHED":0,"FAILED":0},"metrics":{"read-bytes":0,"read-bytes-complete":true,"write-bytes":0,"write-bytes-co
mplete":true,"read-records":0,"read-records-complete":true,"write-records":0,"write-records-complete":tru
e,"buffers-in-pool-usage-max":0.0,"buffers-in-pool-usage-max-complete":true,"buffers-out-pool-usage-m
ax":0.0,"buffers-out-pool-usage-max-complete":true,"tps":-1.0,"tps-complete":true,"delay":-1,"delay-com
plete":true}}],"status-counts":{"CANCELING":0,"RUNNING":0,"SCHEDULED":0,"RECONCILING":0,"DEPLOYING":
0,"CREATED":0,"CANCELED":1,"FINISHED":0,"FAILED":0},"plan":{"jid":"166845b9753a86282144c59d0c3d82b7
","name":"test_0812","nodes":[{"id":"717c7b8afebbfb7137f6f0f99beb2a94","parallelism":1,"operator":"","op
erator_strategy":"","description":"Source: RandomSource-test_input-Stream -&gt; SourceConversion(t
able:[builtin, default, _DataStreamTable_0, source: [RandomSource-test_input]], fields:(s, rowtime)) -&gt;
&gt; WatermarkAssigner(rowtime: rowtime, offset: 4000) -&gt; Calc(select: (s AS c)) -&gt; SinkCon
version to Tuple2 -&gt; Flat Map -&gt; Sink: Print to System.out","optimizer_properties":{}}]}</Det
ail>
```

JSON 格式

```
{
  "RequestId": "2FC99DE0-9E70-4E57-9B16-5CBDB10211FB",
  "Detail": "{\"jid\":\"166845b9753a86282144c59d0c3d82b7\",\"name\":\"test_0812\",\"isStoppable\":false,
  \"state\":\"CANCELED\",\"start-time\":1600879950827,\"end-time\":1610505455432,\"duration\":96255046
  05,\"now\":1610505455494,\"timestamps\":{\"CREATED\":1600879950827,\"RUNNING\":1600879950869,\"
  RECONCILING\":0,\"FAILING\":0,\"FINISHED\":0,\"CANCELLING\":1610505454138,\"RESTARTING\":16008799
  40762,\"SUSPENDING\":0,\"SUSPENDED\":0,\"FAILED\":0,\"CANCELED\":1610505455432},\"vertices\":{\"id
  \":\"717c7b8afebbfb7137f6f0f99beb2a94\",\"topology-id\":0,\"name\":\"Source: RandomSource-test_inp
  ut-Stream -> SourceConversion(table:[builtin, default, _DataStreamTable_0, source: [RandomSource-test_inp
  ut]], fields:(s, rowtime)) -> WatermarkAssigner(rowtime: rowtime, offset: 4000) -> Calc(select: (s AS c)) -> Si
  nkConversion to Tuple2 -> Flat Map -> Sink: Print to System.out\",\"parallelism\":1,\"status\":\"CANCELED\"
  },\"start-time\":160088000174,\"end-time\":1610505455427,\"duration\":9625455253,\"tasks\":{\"CANCELI
  NG\":0,\"RUNNING\":0,\"SCHEDULED\":0,\"RECONCILING\":0,\"DEPLOYING\":0,\"CREATED\":0,\"CANCELED
  \":1,\"FINISHED\":0,\"FAILED\":0},\"metrics\":{\"read-bytes\":0,\"read-bytes-complete\":true,\"write-bytes\"
  :0,\"write-bytes-complete\":true,\"read-records\":0,\"read-records-complete\":true,\"write-records\":0,\"
  write-records-complete\":true,\"buffers-in-pool-usage-max\":0.0,\"buffers-in-pool-usage-max-complete\"
  :true,\"buffers-out-pool-usage-max\":0.0,\"buffers-out-pool-usage-max-complete\":true,\"tps\":-1.0,\"tp
  s-complete\":true,\"delay\":-1,\"delay-complete\":true}},\"status-counts\":{\"CANCELING\":0,\"RUNNING\"
  :0,\"SCHEDULED\":0,\"RECONCILING\":0,\"DEPLOYING\":0,\"CREATED\":0,\"CANCELED\":1,\"FINISHED\":0,
  \"FAILED\":0},\"plan\":
  {\"jid\":\"166845b9753a86282144c59d0c3d82b7\",\"name\":\"test_0812\",\"nodes\":{\"id\":\"717c7b8afe
  bbfb7137f6f0f99beb2a94\",\"parallelism\":1,\"operator\":\"\",\"operator_strategy\":\"\",\"description\":\"
  Source: RandomSource-test_input-Stream -&gt; SourceConversion(table:[builtin, default, _DataStreamTable
  _0, source: [RandomSource-test_input]], fields:(s, rowtime)) -&gt; WatermarkAssigner(rowtime: rowtime, of
  fset: 4000) -&gt; Calc(select: (s AS c)) -&gt; SinkConversion to Tuple2 -&gt; Flat Map -&gt; Sink: Print to Syste
  m.out\",\"optimizer_properties\":{}}}}
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.7. GetInstanceExceptions

您可以通过GetInstanceExceptions OpenAPI获取运行实例的Failover信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/exceptions HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	403936	实例ID： <ul style="list-style-type: none"> 流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。 批作业：可以通过ListInstance、StartJob等接口获得。
jobName	String	Path	是	test_0812	作业名称。
projectName	String	Path	是	bayes_team	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
Exceptions	String	{\"root-exception\":null,\"timestamp\":null,\"all-exceptions\": [],\"truncated\":false}	Failover的具体信息。
RequestId	String	97381B4E-EE09-497A-A4AB-9AA6E5B7D0DE	请求ID。

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instances/412532/exceptions
```

正常返回示例

XML 格式

```
<RequestId>97381B4E-EE09-497A-A4AB-9AA6E5B7D0DE</RequestId>
<Exceptions>{\"root-exception\":null,\"timestamp\":null,\"all-exceptions\": [],\"truncated\":false}</Exceptions>
```

JSON 格式

```
{
  "RequestId": "97381B4E-EE09-497A-A4AB-9AA6E5B7D0DE",
  "Exceptions": "{\"root-exception\":null,\"timestamp\":null,\"all-exceptions\":[],\"truncated\":false}"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.8. GetInstanceFinalState

您可以通过GetInstanceFinalState OpenAPI查看运行实例在YARN上的最终状态。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/finalstate HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	412532	实例ID： <ul style="list-style-type: none">流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。批作业：可以通过ListInstance或StartJob等接口获得。
jobName	String	Path	是	job1	作业名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
Finalstate	String	RUNNING	YARN上的最终状态。
RequestId	String	3F4ADBCF-B116-4680-80B3-E9A139841A0D	请求ID。

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instances/412532/finalstate
```

正常返回示例

XML 格式

```
<requestId>3F4ADBCF-B116-4680-80B3-E9A139841A0D</requestId>
```

JSON 格式


```
{
  "requestId": "3F4ADBCF-B116-4680-80B3-E9A139841A0D"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.9. GetInstanceMetric

您可以通过GetInstanceMetric获取正在运行实例的所有Metric信息。

 **说明** 使用GetInstanceMetric调用未运行的实例的Metric时，会产生报错。metricJson参数的详细请参见[opentsdb标准协议](#)。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/projects/[projectName]/jobs/[jobName]/metric HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称。
metricjson	String	Body	是	<pre>{ "start":1606291390000,"end":1606294626216, "limit":"avg:sample:50","queries": [{"downsample":"20s-avg","metric":"blink.projectName.jobName.task_failover.rate","granularity":"20s","aggregator":"max"}]}</pre>	<p>使用特定JSON来获取Metric:</p> <ul style="list-style-type: none"> • start: Metric开始时间。使用13位时间戳，精确到毫秒。 • limit: 取各条线值类型：max、avg和min。取值：bottom、above和sample。 • end: Metric结束时间。使用13位时间戳，精确到毫秒。 • downsample: 采样方式时间（秒）。取值：max、avg和min。 • metric: blink.项目名称.作业名称.指标名称。常用指标名称例如：业务延时（delay）。 • granularity: 采样时间，每隔多少秒取一个点。需要根据start和end的进行计算。采集时间点太多会造成调用超时。 • aggregator: 聚合方式。按照采样时间将底层的点聚合后输出为一个点。聚合方式有最大值max，最小值min和平均值avg。
projectName	String	Path	是	project1	项目名称。
instanceId	Long	Query	否	-1	<p>InstanceID。流作业和批作业对应的实例ID如下：</p> <ul style="list-style-type: none"> • 流作业：只有一个运行实例，此处填 -1L，指在线上运行的。 • 批作业：可以通过 ListInstance 接口或者 StartJob 接口获得。
RegionId	String	Header	否	cn-hangzhou	地域ID。

返回数据

名称	类型	示例值	描述
----	----	-----	----

名称	类型	示例值	描述
Metrics	Array of Metric		Metric信息详情。
Metric			
Dps	Map	k:v	时间点和对应的Metric值。
MetricName	String	blink.bayes_team.s s2.delay	Metric名称。
Summary	Float	494400448	聚合后的Metric值。
Tags	Map	k:v	Metric标记。
RequestId	String	4D7AB7CF-4EF2- 4635-9137- 51803CD71DAD	请求ID。

Metric名称对应表

参数名称	指标名称
业务延时 (ms)	delay
数据滞留时间 (ms)	fetches_delay
数据间隔时间 (ms)	no_data_delay
作业失败率	task_failover.rate
Source的tps数据输入	tps.rate
Sink的数据输出	sink.outTps.rate
Source的RPS数据输出	parserTps.rate
Source的数据流量输入 (byte)	inBps.rate
Source的脏数据比例	parserSkipCount

参数名称	指标名称
Checkpoint Duration (ms)	lastCheckpointDuration
Checkpoint大小 (byte)	lastCheckpointSize
checkpoint对齐时间 (ns)	checkpointAlignmentTime
checkpoint数量	checkpoints
获取state的时间 (ns)	rocksdb_get.mean
写入state的时间 (ns)	rocksdb_put.mean
seek (ns)	niagara_seek.meam / rocksdb_seek.mean
state大小 (byte)	state.state_size / state.state_newsize
GMS GC Time (ms)	Status.JVM.GarbageCollector.ConcurrentMarkSweep.Time
GMS GC Rate	Status.JVM.GarbageCollector.ConcurrentMarkSweep.Count
WaterMark Delay (ms)	watermarkLatency
数据迟到丢弃TPS	lateRecordsDroppedRate
数据迟到累计丢弃数	numLateRecordsDropped
读TT延时 (ms)	input.tt.readLatency
Task Input TPS	numRecordsInPerSecond.rate
Task Output TPS	numRecordsOutPerSecond.rate
Input Queue Usage	buffers.inPoolUsage

参数名称	指标名称
Output Queue Usage	buffers.outPoolUsage
Time Used In Processing Per Second (ns)	taskLatency.sum
Time Used In Waiting Oputput Per Second (ns)	waitOutput.sum
TaskLatency Histogram Mean (ns)	taskLatency.histogram.mean
WaitOutput Histogram Mean (ns)	waitOutput.histogram.mean
WaitInput Histogram Mean (ns)	waitInput.histogram.mean
PartitionLatency Mean (ns)	partitionLatency.mean
Process MEM Rss (kb)	Status.JVM.Memory.Process.rss
CPU Usage	Status.JVM.CPU.Usage
Memory Heap Used (byte)	Status.JVM.Memory.Heap.Used
Memory NonHeap Used (byte)	Status.JVM.Memory.NonHeap.Used
Threads Count	Status.JVM.Threads.Count
GC(CMS)	Status.JVM.GarbageCollector.ConcurrentMarkSweep.Count

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/metric
{"metricJson":{"start":1606291390000,"end":1606294626216,"limit":{"avg:sample:50},"queries":[{"downsample":{"20s-avg"},"metric":{"blink.projectName.jobName.task_failover.rate},"granularity":{"20s},"aggregator":{"max"}}]}}
```

正常返回示例

XML 格式

```
<requestId>4D7AB7CF-4EF2-4635-9137-51803CD71DAD</requestId>
<metrics>
  <metricName>blink.bayes_team.ss2.delay</metricName>
  <dps>
    <1574247000>4.929E+8</1574247000>
    <1574247600>4.9350016E+8</1574247600>
    <1574248200>494100352</1574248200>
    <1574248800>494700544</1574248800>
    <1574249400>495300736</1574249400>
    <1574250000>495900896</1574250000>
  </dps>
  <summary>494400448</summary>
  <tags/>
</metrics>
```

JSON 格式

```
{
  "requestId": "4D7AB7CF-4EF2-4635-9137-51803CD71DAD",
  "metrics": [
    {
      "metricName": "blink.bayes_team.ss2.delay",
      "dps": {
        "1574247000": "4.929E+8",
        "1574247600": "4.9350016E+8",
        "1574248200": "494100352",
        "1574248800": "494700544",
        "1574249400": "495300736",
        "1574250000": "495900896"
      },
      "summary": "494400448",
      "tags": {}
    }
  ]
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.10. GetInstanceRunSummary

您可以通过GetInstanceRunSummary OpenAPI获取作业运行实例的运行信息。该OpenAPI是GetInstance接口的轻量级版本，会返回作业运行时的部分信息，相比GetInstance接口更加轻量级。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/runsummary HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	-1	InstanceID: <ul style="list-style-type: none"> 流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。 批作业：可以通过ListInstance、StartJob等接口获得。
jobName	String	Path	是	job1	作业名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
RequestId	String	0B3BAFBB-3C0B-43B2-8C5E-3FA93173E160	请求ID。
RunSummary	Struct		运行概要信息。
ActualState	String	RUNNING	实际运行状态： <ul style="list-style-type: none"> WAITING：等待。 RUNNING：运行。 PAUSED：暂停。 TERMINATED：停止。 SUCCESS：成功（批作业）。 FAILED：失败（流作业）。
EngineJobHandler	String	application_1561366511517_113805 a10928ceec91de395ff3d7593f7ff46c5	Instance在YARN中的名称：ApplicaitonID JobID（JM分配的JobID）。

名称	类型	示例值	描述
ExpectState	String	RUNNING	期望运行状态： <ul style="list-style-type: none">• WAITING：等待。• RUNNING：运行。• PAUSED：暂停。• TERMINATED：停止。• SUCCESS：成功（批作业）。• FAILED：失败（流作业）。
Id	Long	214650	实例ID。
InputDelay	Long	0	业务延时。
JobName	String	job1	作业名称。
LastErrorMessage	String	error	最近错误信息。
LastErrorTime	Long	1548397575000	最近错误时间。

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instances/-1/runsummary
```

正常返回示例

XML 格式

```
<RequestId>0B3BAFBB-3C0B-43B2-8C5E-3FA93173E160</RequestId>
<RunSummary>
  <ActualState>RUNNING</ActualState>
  <EngineJobHandler>application_1561366511517_113805|a10928cec91de395ff3d7593f7ff46c5</EngineJob
Handler>
  <ExpectState>RUNNING</ExpectState>
  <JobName>job1</JobName>
  <Id>214650</Id>
  <InputDelay>0</InputDelay>
</RunSummary>
```

JSON 格式

```
{
  "RequestId": "0B3BAFBB-3C0B-43B2-8C5E-3FA93173E160",
  "RunSummary": {
    "ActualState": "RUNNING",
    "EngineJobHandler": "application_1561366511517_113805|a10928cec91de395ff3d7593f7ff46c5",
    "ExpectState": "RUNNING",
    "JobName": "job1",
    "Id": 214650,
    "InputDelay": 0
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.11. ListInstance

您可以通过ListInstance获取指定项目下所有的运行实例信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/instances HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobType	String	Query	是	FLINK_STREAM	作业类型： <ul style="list-style-type: none"> FLINK_STREAM（流作业）。 FLINK_BATCH（批作业）。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID。
jobName	String	Query	否	job1	作业名称。

名称	类型	位置	是否必选	示例值	描述
startBeginTs	Long	Query	否	1548397575000	启动时间范围上限，和启动时间下限配合使用，搜索启动时间在该范围内的运行实例。使用13位时间戳，单位到毫秒。
startEndTs	Long	Query	否	1548397575000	启动时间范围下限，和启动时间上限配合使用，搜索启动时间在该范围内的运行实例。使用13位时间戳，单位到毫秒。
endBeginTs	Long	Query	否	1548397575000	停止时间范围上限，和停止时间下限配合使用，搜索停止时间在该范围内的运行实例。使用13位时间戳，单位到毫秒。
endEndTs	Long	Query	否	1548397575000	停止时间范围下限，和停止时间上限配合使用，搜索停止时间在该范围内的运行实例。使用13位时间戳，单位到毫秒。
apiType	String	Query	否	SQL	AP类型：DATASTREAM或SQL。
expectState	String	Query	否	RUNNING	期望状态： <ul style="list-style-type: none"> • RUNNING：运行。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（流作业）。
actualState	String	Query	否	RUNNING	实际运行状态： <ul style="list-style-type: none"> • RUNNING：运行。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（流作业）。

名称	类型	位置	是否必选	示例值	描述
isArchived	Boolean	Query	否	true	是否搜索归档实例，默认搜索非归档实例。对于流作业，同时只存在一个实例，当作业启动或者恢复运行的时候，将生成新的实例，旧实例会被归档；对于批作业，运行结束两天的实例会被归档。
pageSize	Integer	Query	否	10	分页选项，每页的实例数。
pageIndex	Integer	Query	否	1	分页选项，第几页。

返回数据

名称	类型	示例值	描述
Instances	Array of Instance		实例详情。
Instance			
ActualState	String	RUNNING	实际运行状态： <ul style="list-style-type: none"> • RUNNING：运行。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（流作业）。
ApiType	String	SQL	API类型：DATASTREAM或SQL。
ClusterId	String	et2cloud****	集群ID。
Code	String	<pre>CREATE TABLE `test_output` (\n c1 BIGINT,\n c2 BIGINT,\n c3 VARCHAR\n) WITH (\n type = 'print'\n);\n\nINSERT INTO `test_output`\n select\n *\n from `test_input`;</pre>	实例代码。

名称	类型	示例值	描述
EndTime	Long	1548397575000	结束时间。
EngineJobHandler	String	application_1609851266735_7475 00bbc29106f9bf321048f85e5e57cd4c	Instance在YARN中的名称: ApplicaitonID JobID (JM分配的JobID)。
EngineVersion	String	blink-3.5.0-hotfix	引擎版本。
ExpectState	String	RUNNING	期望状态: <ul style="list-style-type: none"> • RUNNING: 运行。 • PAUSED: 暂停。 • TERMINATED: 停止。 • SUCCESS: 成功 (批作业)。 • FAILED: 失败 (流作业)。
Id	Long	589016	实例ID。
InputDelay	Long	0	业务延时。
JobName	String	job1	作业名称。
JobType	String	FLINK_STREAM	作业类型: <ul style="list-style-type: none"> • FLINK_STREAM (流作业)。 • FLINK_BATCH (批作业)。
LastErrorMessage	String	error	最近一次错误信息。
LastErrorTime	Long	1612336822000	最近错误时间。
LastOperateTime	Long	1612336822000	最近操作时间。
LastOperator	String	170906468757****	最近操作者。
Packages	String	package1.jar	Instance引用的Package, 多个Package用逗号 (,) 分隔, 未引用为空。

名称	类型	示例值	描述
PlanJson	String	{\n \"global\": [\n \"jobManagerMinCpuCores\": 0.1,\n \"jobManagerMinMemoryCores\": 1024,\n \"jobManagerCpuCores\": 0.25,\n \"jobManagerMemoryInMB\": 1024\n],\n \"nodes\": [\n \"id\": 1,\n \"uid\": \"1\"	作业上线执行计划。
Priority	Integer	5	123
ProjectName	String	project1	项目名称。
Properties	String	#state.backend.rocksdb.ttl.ms=12960000\n\n#启用项目参数\n\n#disable.project.config=false\n\nblink.job.timeZone=Asia/Shanghai	作业运行配置参数。
QueueName	String	root.project1	队列名称。
StartTime	Long	1612336822000	作业启动时间。
Starter	String	170906468757****	启动人。
PageIndex	Integer	1	分页属性，第几页。
PageSize	Integer	10	分页属性，每页多少实例。
RequestId	String	33DBCFDA-5D75-452C-8DB3-5964CFE7747C	请求ID。
TotalCount	Long	15	总实例数目。
TotalPage	Integer	2	总页数。

示例

请求示例

```
/api/v2/projects/project1/instances
```

正常返回示例

XML 格式

```
<Instances>
  <Instance>
    <ExpectState>RUNNING</ExpectState>
    <EngineVersion>blink-3.5.0-hotfix</EngineVersion>
    <ProjectName>project1</ProjectName>
    <ClusterId>et2cloud****</ClusterId>
    <Priority>5</Priority>
    <PlanJson>{
      "global" : [ {
        "jobManagerMinCpuCores" : 0.1,
        "jobManagerMinMemoryCores" : 1024,
        "jobManagerCpuCores" : 0.25,
        "jobManagerMemoryInMB" : 1024
      } ],
      "nodes" : [ {
        "id" : 1,
        "uid" : "1",
        "name" : "RandomSource-test_input-Stream",
        "pact" : "Source",
        "chainingStrategy" : "HEAD",
        "parallelism" : 1,
        "maxParallelism" : 32768,
        "vcore" : 0.25,
        "heap_memory" : 252,
        "native_memory" : 4
      }, {
        "id" : 2,
        "uid" : "2",
        "name" : "SourceConversion(table:[builtin, default, test_input, source: [RandomSource-test_input]], fields:
(s1, s2, s3))",
        "pact" : "Operator",
        "parallelism" : 1,
        "maxParallelism" : 32768,
        "vcore" : 0.25,
        "heap_memory" : 252,
        "native_memory" : 4
      }, {
        "id" : 3,
        "uid" : "3",
        "name" : "SinkConversion to Tuple2",
        "pact" : "Operator",
        "parallelism" : 1,
        "maxParallelism" : 32768,
        "vcore" : 0.25,

```



```

"heap_memory" : 252,
"native_memory" : 4
}, {
  "id" : 4,
  "uid" : "4",
  "name" : "Flat Map",
  "pact" : "Operator",
  "parallelism" : 1,
  "maxParallelism" : 32768,
  "vcore" : 0.25,
  "heap_memory" : 252,
  "native_memory" : 4
}, {
  "id" : 5,
  "uid" : "5",
  "name" : "Print to System.out",
  "pact" : "Sink",
  "parallelism" : 1,
  "maxParallelism" : 32768,
  "vcore" : 0.25,
  "heap_memory" : 252,
  "native_memory" : 4
}], {
  "links" : [{
    "source" : 1,
    "target" : 2
  }, {
    "source" : 2,
    "target" : 3
  }, {
    "source" : 3,
    "target" : 4
  }, {
    "source" : 4,
    "target" : 5
  }
  ]
}
</PlanJson>
  <JobName>job1</JobName>
  <StartTime>1612336822000</StartTime>
  <Properties>#checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE
#blink.checkpoint.mode=EXACTLY_ONCE
#checkpoint间隔时间, 单位毫秒
#blink.checkpoint.interval.ms=180000
#rocksdb的数据生命周期, 单位毫秒
#state.backend.rocksdb.ttl.ms=129600000
#启用项目参数
#disable.project.config=false
blink.job.timeZone=Asia/Shanghai</Properties>
  <Starter>170906468757****</Starter>
  <InputDelay>0</InputDelay>
  <Code>--SQL
--*****
--Author: streamcompute@aliyun-test.com
CreateTime: 2020-03-05 10:47:13

```

```

--CreateTime: 2020-03-05 19:47:13
--Comment: 请输入业务注释信息
--*****_
CREATE TABLE `test_input` (
  s1 BIGINT,
  s2 BIGINT,
  s3 VARCHAR
) WITH (
  `type` = 'random'
);
CREATE TABLE `test_output` (
  c1 BIGINT,
  c2 BIGINT,
  c3 VARCHAR
) WITH (
  type = 'print'
);
INSERT INTO `test_output`
select
*
from `test_input`;</Code>
<ActualState>RUNNING</ActualState>
<EngineJobHandler>application_1609851266735_7475|00bbc29106f9bf321048f85e5e57cd4c</EngineJobHandler>
<JobType>FLINK_STREAM</JobType>
<LastOperator>170906468757****</LastOperator>
<Packages/>
<ApiType>SQL</ApiType>
<Id>589016</Id>
<LastOperateTime>1612336822000</LastOperateTime>
<QueueName>root.bayes_team</QueueName>
</Instance>
<Instance>
<ExpectState>RUNNING</ExpectState>
<EngineVersion>blink-2.2.6</EngineVersion>
<ProjectName>bayes_team</ProjectName>
<ClusterId>et2cloudprod</ClusterId>
<Priority>5</Priority>
<PlanJson>{
"resourceSettings" : {
"blink.resource.unit.per.worker" : 1,
"blink.resource.unit.granularity" : 0.1,
"blink.metrics.endpoint" : "http://100.118.128.166:9000/api/query?tenant=blink&token=e8ff43ab",
"blink.resource.allocation.jm.min.cpu.cores" : 0.2,
"blink.resource.allocation.jm.min.memory.mb" : 512
},
"autoConfig" : {
"goal" : {
"maxResourceUnits" : 10000.0
},
"result" : {
"attemptId" : 1,
"attempts" : 1,
"attemptsCurrentPlan" : 1,
"statusMessage" : "New job. using default configuration."
}
}
}

```

```
        "scalingAction": "InitialScale",
        "allocatedResourceUnits": 0.9000000000000001,
        "allocatedCpuCores": 0.9000000000000001,
        "allocatedMemoryInMB": 3685
    }
},
"global": [{
    "tracingMetricsEnabled": true,
    "jobManagerCpuCores": 0.30000000000000004,
    "jobManagerMemoryInMB": 1228
}],
"nodes": [{
    "id": 1,
    "uid": "1",
    "pact": "Data Source",
    "name": "GSIsTableSource-sls_source-Stream",
    "chainingStrategy": "HEAD",
    "parallelism": 2,
    "maxParallelism": 32768,
    "vcore": 0.1,
    "heap_memory": 128
}, {
    "id": 2,
    "uid": "2",
    "pact": "Operator",
    "name": "SourceConversion(table:[_DataStreamTable_0, source: [GSIsTableSource-sls_source]], fields:
(f0))",
    "parallelism": 2,
    "maxParallelism": 32768,
    "vcore": 0.1,
    "heap_memory": 128
}, {
    "id": 3,
    "uid": "3",
    "pact": "Operator",
    "name": "correlate: table(SlsParser0($cor0.f0)), select: name",
    "parallelism": 2,
    "maxParallelism": 32768,
    "vcore": 0.1,
    "heap_memory": 128
}, {
    "id": 5,
    "uid": "5",
    "pact": "Operator",
    "name": "GroupAggregate(groupBy: (name), select: (name, COUNT(name) AS EXPR$0))",
    "chainingStrategy": "HEAD",
    "parallelism": 2,
    "maxParallelism": 32768,
    "vcore": 0.1,
    "heap_memory": 128
}, {
    "id": 6,
    "uid": "6",
    "pact": "Operator",
```

```
"name" : "Calc(select: (EXPR$0, name))",
"parallelism" : 2,
"maxParallelism" : 32768,
"vcore" : 0.1,
"heap_memory" : 128
}, {
  "id" : 7,
  "uid" : "7",
  "pact" : "Operator",
  "name" : "SinkConversion to Tuple2",
  "parallelism" : 2,
  "maxParallelism" : 32768,
  "vcore" : 0.1,
  "heap_memory" : 128
}, {
  "id" : 8,
  "uid" : "8",
  "pact" : "Operator",
  "name" : "Flat Map",
  "parallelism" : 2,
  "maxParallelism" : 32768,
  "vcore" : 0.1,
  "heap_memory" : 128
}, {
  "id" : 9,
  "uid" : "9",
  "pact" : "Data Sink",
  "name" : "OutputFormatAdapterSink:com.alibaba.blink.streaming.connector.sls.sink.SlsOutputFormat@c
cd341d",
  "parallelism" : 2,
  "maxParallelism" : 32768,
  "vcore" : 0.1,
  "heap_memory" : 128
}],
"links" : [{
  "source" : 1,
  "target" : 2,
  "side" : "second"
}, {
  "source" : 2,
  "target" : 3,
  "side" : "second"
}, {
  "source" : 3,
  "target" : 5,
  "ship_strategy" : "HASH",
  "side" : "second"
}, {
  "source" : 5,
  "target" : 6,
  "side" : "second"
}, {
  "source" : 6,
  "target" : 7,
```

```

"side" : "second"
}, {
  "source" : 7,
  "target" : 8,
  "side" : "second"
}, {
  "source" : 8,
  "target" : 9,
  "side" : "second"
}],
"statefulNodes" : [ 5 ],
"vertexAdjustments" : {
  "0" : {
    "parallelismFactor" : 2.0,
    "parallelismLimit" : 2
  },
  "1" : {
    "parallelismFactor" : 2.0
  }
},
"vertices" : {
  "0" : {
    "vertexId" : 0,
    "source" : true,
    "name" : "Source: GSlsTableSource-sls_source-Stream -&gt; SourceConversion(table:[_DataStreamTable_0, source: [GSlsTableSource-sls_source]], fields:(f0)) -&gt; correlate: table(SlsParser0($cor0.f0)), select: name",
    "parallelismFactor" : 2.0,
    "parallelismLimit" : 2,
    "parallelism" : 2,
    "maxParallelism" : 32768,
    "maxPartitionsPerSourceParallelism" : 8,
    "minResource" : {
      "cpuCores" : 0.1,
      "heapMemoryInMB" : 384,
      "managedMemoryInMB" : 0,
      "directMemoryInMB" : 0,
      "nativeMemoryInMB" : 0
    },
    "transformations" : [ 1, 2, 3 ],
    "inputVertexIds" : []
  },
  "1" : {
    "vertexId" : 1,
    "name" : "GroupAggregate(groupBy: (name), select: (name, COUNT(name) AS EXPR$0)) -&gt; Calc(select : (EXPR$0, name)) -&gt; SinkConversion to Tuple2 -&gt; Flat Map -&gt; Sink: OutputFormatAdapterSink:com.alibaba.blink.streaming.connector.sls.sink.SlsOutputFormat@ccd341d",
    "parallelismFactor" : 2.0,
    "parallelismLimit" : 32768,
    "parallelism" : 2,
    "maxParallelism" : 32768,
    "minResource" : {
      "cpuCores" : 0.1,
      "heapMemoryInMB" : 640,

```

```

    "managedMemoryInMB" : 0,
    "directMemoryInMB" : 0,
    "nativeMemoryInMB" : 0
  },
  "transformations" : [ 5, 6, 7, 8, 9 ],
  "inputVertexIds" : [ 0 ]
}
},
"subDags" : {
  "0" : {
    "vertices" : [ 0, 1 ],
    "inputFromVertices" : [ ],
    "targetTps" : 0.0
  }
},
"workers" : [ {
  "numWorkers" : 1,
  "resourceUnits" : 0.2,
  "estimatedCpuCores" : 0.2,
  "estimatedMemoryInMB" : 768.0,
  "allocatedCpuCores" : 0.2,
  "allocatedMemoryInMB" : 819,
  "taskSlots" : [ {
    "topologyID" : 0,
    "count" : 2
  } ]
} ], {
  "numWorkers" : 1,
  "resourceUnits" : 0.4,
  "estimatedCpuCores" : 0.2,
  "estimatedMemoryInMB" : 1280.0,
  "allocatedCpuCores" : 0.4,
  "allocatedMemoryInMB" : 1638,
  "taskSlots" : [ {
    "topologyID" : 1,
    "count" : 2
  } ]
} ]
}
}
</PlanJson>
  <JobName>ss2</JobName>
  <StartTime>1575353621000</StartTime>
  <Properties>#checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE
#blink.checkpoint.mode=EXACTLY_ONCE
#checkpoint间隔时间, 单位毫秒
#blink.checkpoint.interval.ms=180000
#rocksdb的数据生命周期, 单位毫秒
#state.backend.rocksdb.ttl.ms=129600000
blink.job.timeZone=Asia/Shanghai</Properties>
  <Starter>1709064687573327</Starter>
  <InputDelay>0</InputDelay>
  <Code>--Blink SQL
  __*****__
  --Author: streamcompute@aliyun-test.com
  CreateTime: 2019-12-25 14:40:51

```

```

--CreateTime: 2018-12-25 14:49:51
--Comment: 请输入业务注释信息
--*****_
CREATE TABLE sls_source (
  `name` VARCHAR,
  num VARCHAR,
  id BIGINT
) WITH (
  type='sls',
  endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
  roleArn='acs:ram::1709064687573327:role/aliyunstreamdefaultrole',
  project='bayes-test-jituan',
  logStore='sls_source'
);
CREATE TABLE sls_output (
  id BIGINT,
  `name` varchar
) WITH (
  type='SLS',
  endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
  roleArn='acs:ram::1709064687573327:role/aliyunstreamdefaultrole',
  project='bayes-test-jituan',
  logStore='sls_output'
);
INSERT INTO sls_output
SELECT
  COUNT(`name`),
  `name`
FROM sls_source GROUP BY `name`</Code>
<ActualState>RUNNING</ActualState>
<EngineJobHandler>application_1561366511517_113805|a10928cec91de395ff3d7593f7ff46c5</EngineJobHandler>
<JobType>FLINK_STREAM</JobType>
<LastOperator>1709064687573327</LastOperator>
<Packages/>
<ApiType>SQL</ApiType>
<Id>214650</Id>
<LastOperateTime>1575353621000</LastOperateTime>
<QueueName>root.bayes_team</QueueName>
</Instance>
</Instances>
<TotalCount>15</TotalCount>
<RequestId>33DBCFDA-5D75-452C-8DB3-5964CFE7747C</RequestId>
<PageSize>10</PageSize>
<TotalPage>2</TotalPage>
<PageIndex>1</PageIndex>

```

JSON 格式

```

{
  "Instances": {
    "Instance": [
      {
        "ExpectState": "RUNNING",

```

```

"EngineVersion": "blink-3.5.0-hotfix",
"ProjectName": "project1",
"ClusterId": "et2cloud****",
"Priority": 5,
"PlanJson": "{\n  \"global\": [\n    {\n      \"jobManagerMinCpuCores\": 0.1,\n      \"jobManagerMinMemoryCores\": 1024,\n      \"jobManagerCpuCores\": 0.25,\n      \"jobManagerMemoryInMB\": 1024\n    },\n    {\n      \"id\": 1,\n      \"uid\": \"1\",\n      \"name\": \"RandomSource-test_input-Stream\",\n      \"pact\": \"Source\",\n      \"chainingStrategy\": \"HEAD\",\n      \"parallelism\": 1,\n      \"maxParallelism\": 32768,\n      \"vcore\": 0.25,\n      \"heap_memory\": 252,\n      \"native_memory\": 4\n    },\n    {\n      \"id\": 2,\n      \"uid\": \"2\",\n      \"name\": \"SourceConversion(table:[builtin, default, test_input, source: [RandomSource-test_input]], fields:(s1, s2, s3))\",\n      \"pact\": \"Operator\",\n      \"parallelism\": 1,\n      \"maxParallelism\": 32768,\n      \"vcore\": 0.25,\n      \"heap_memory\": 252,\n      \"native_memory\": 4\n    },\n    {\n      \"id\": 3,\n      \"uid\": \"3\",\n      \"name\": \"SinkConversion to Tuple2\",\n      \"pact\": \"Operator\",\n      \"parallelism\": 1,\n      \"maxParallelism\": 32768,\n      \"vcore\": 0.25,\n      \"heap_memory\": 252,\n      \"native_memory\": 4\n    },\n    {\n      \"id\": 4,\n      \"uid\": \"4\",\n      \"name\": \"Flat Map\",\n      \"pact\": \"Operator\",\n      \"parallelism\": 1,\n      \"maxParallelism\": 32768,\n      \"vcore\": 0.25,\n      \"heap_memory\": 252,\n      \"native_memory\": 4\n    },\n    {\n      \"id\": 5,\n      \"uid\": \"5\",\n      \"name\": \"Print to System.out\",\n      \"pact\": \"Sink\",\n      \"parallelism\": 1,\n      \"maxParallelism\": 32768,\n      \"vcore\": 0.25,\n      \"heap_memory\": 252,\n      \"native_memory\": 4\n    }\n  ],\n  \"links\": [\n    {\n      \"source\": 1,\n      \"target\": 2\n    },\n    {\n      \"source\": 2,\n      \"target\": 3\n    },\n    {\n      \"source\": 3,\n      \"target\": 4\n    },\n    {\n      \"source\": 4,\n      \"target\": 5\n    }\n  ]\n}",
"JobName": "job1",
"StartTime": 1612336822000,
"Properties": "#checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE\n#blink.checkpoint.mode=EXACTLY_ONCE\n\n#checkpoint间隔时间, 单位毫秒\n#blink.checkpoint.interval.ms=180000\n\n#rocksdb的数据生命周期, 单位毫秒\n#state.backend.rocksdb.ttl.ms=129600000\n\n#启用项目参数\n#disable.project.config=false\n\nblink.job.timeZone=Asia/Shanghai",
"Starter": "170906468757****",
"InputDelay": 0,
"Code": "--SQL\n--*****--\n--Author: streamcompute@aliyun-test.com\n--CreateTime: 2020-03-05 19:47:13\n--Comment: 请输入业务注释信息\n--*****\n\n\nCREATE TABLE `test_input` (\n s1 BIGINT,\n s2 BIGINT,\n s3 VARCHAR\n) WITH (\n `type` = 'random'\n);\n\nCREATE TABLE `test_output` (\n c1 BIGINT,\n c2 BIGINT,\n c3 VARCHAR\n) WITH (\n `type` = 'print'\n);\n\nINSERT INTO `test_output`\n select *\n from `test_input`;",
"ActualState": "RUNNING",
"EngineJobHandler": "application_1609851266735_7475|00bbc29106f9bf321048f85e5e57cd4c",
"JobType": "FLINK_STREAM",
>LastOperator": "170906468757****",
"Packages": "",
"ApiType": "SQL",
"Id": 589016,
>LastOperateTime": 1612336822000,
"QueueName": "root.bayes_team"
},
{
"ExpectState": "RUNNING",
"EngineVersion": "blink-2.2.6",
"ProjectName": "bayes_team",
"ClusterId": "et2cloudprod",
"Priority": 5,
"PlanJson": "{\n  \"resourceSettings\": {\n    \"blink.resource.unit.per.worker\": 1,\n    \"blink.resource.unit.granularity\": 0.1,\n    \"blink.metrics.endpoint\": \"http://100.118.128.166:9000/api/query?tenant=blink&token=e8ff43ab\",\n    \"blink.resource.allocation.jm.min.cpu.cores\": 0.2,\n    \"blink.resource.allocation.jm.min.memory.mb\": 1024\n  }\n}"

```



```

on.jm.min.memory.mb\": 512\n },\n\n \"autoConfig\": {\n  \"goal\": {\n    \"maxResourceUnits\": 10000.0\n  },\n  \"result\": {\n    \"attemptId\": 1,\n    \"attempts\": 1,\n    \"attemptsCurrentPlan\": 1,\n    \"statusMessage\": \"New job, using default configuration.\",\n    \"scalingAction\": \"InitialScale\", \n    \"allocatedResourceUnits\": 0.9000000000000001,\n    \"allocatedCpuCores\": 0.9000000000000001,\n    \"allocatedMemoryInMB\": 3685\n  }\n },\n\n \"global\": [\n  {\n    \"tracingMetric sEnabled\": true,\n    \"jobManagerCpuCores\": 0.30000000000000004,\n    \"jobManagerMemoryInMB\": 1228\n  }\n ],\n\n \"nodes\": [\n  {\n    \"id\": 1,\n    \"uid\": \"1\", \n    \"pact\": \"Data Source\", \n    \"name\": \"GSIsTableSource-sls_source-Stream\", \n    \"chainingStrategy\": \"HEAD\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 2,\n    \"uid\": \"2\", \n    \"pact\": \"Operator\", \n    \"name\": \"SourceConversion(table:[_DataStreamTable_0, source: [GSIsTableSource-sls_source]], fields:(f0))\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 3,\n    \"uid\": \"3\", \n    \"pact\": \"Operator\", \n    \"name\": \"correlate: table(SlsParser0($cor0.f0)), select: name\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 5,\n    \"uid\": \"5\", \n    \"pact\": \"Operator\", \n    \"name\": \"GroupAggregate(groupBy: (name), select: (name, COUNT(name) AS EXPR$0))\", \n    \"chainingStrategy\": \"HEAD\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 6,\n    \"uid\": \"6\", \n    \"pact\": \"Operator\", \n    \"name\": \"Calc(select: (EXPR$0, name))\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 7,\n    \"uid\": \"7\", \n    \"pact\": \"Operator\", \n    \"name\": \"SinkConversion to Tuple2\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 8,\n    \"uid\": \"8\", \n    \"pact\": \"Operator\", \n    \"name\": \"Flat Map\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }, \n  {\n    \"id\": 9,\n    \"uid\": \"9\", \n    \"pact\": \"Data Sink\", \n    \"name\": \"OutputFormatAdapterSink:com.alibaba.blink.streaming.connector.sls.sink.SlsOutputFormat@ccd341d\", \n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"vc core\": 0.1,\n    \"heap_memory\": 128\n  }\n ],\n\n \"links\": [\n  {\n    \"source\": 1,\n    \"target\": 2,\n    \"side\": \"second\"\n  }, \n  {\n    \"source\": 2,\n    \"target\": 3,\n    \"side\": \"second\"\n  }, \n  {\n    \"source\": 3,\n    \"target\": 5,\n    \"side_strategy\": \"HASH\", \n    \"side\": \"second\"\n  }, \n  {\n    \"source\": 5,\n    \"target\": 6,\n    \"side\": \"second\"\n  }, \n  {\n    \"source\": 6,\n    \"target\": 7,\n    \"side\": \"second\"\n  }, \n  {\n    \"source\": 7,\n    \"target\": 8,\n    \"side\": \"second\"\n  }, \n  {\n    \"source\": 8,\n    \"target\": 9,\n    \"side\": \"second\"\n  }\n ],\n\n \"statefulNodes\": [ 5 ],\n\n \"vertexAdjustments\": {\n  \"0\": {\n    \"parallelismFactor\": 2.0,\n    \"parallelismLimit\": 2\n  }, \n  \"1\": {\n    \"parallelismFactor\": 2.0\n  }\n },\n\n \"vertices\": {\n  \"0\": {\n    \"vertexId\": 0,\n    \"source\": true,\n    \"name\": \"Source: GSIsTableSource-sls_source-Stream -> SourceConversion(table:[_DataStreamTable_0, source: [GSIsTableSource-sls_source]], fields:(f0)) -> correlate: table(SlsParser0($cor0.f0)), select: name\", \n    \"parallelismFactor\": 2.0,\n    \"parallelismLimit\": 2,\n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"maxPartitionsPerSourceParallelism\": 8,\n    \"minResource\": {\n      \"cpuCores\": 0.1,\n      \"heapMemoryInMB\": 384,\n      \"managedMemoryInMB\": 0,\n      \"directMemoryInMB\": 0,\n      \"nativeMemoryInMB\": 0\n    }, \n    \"transformations\": [ 1, 2, 3 ], \n    \"inputVertexIds\": []\n  }, \n  \"1\": {\n    \"vertexId\": 1,\n    \"name\": \"GroupAggregate(groupBy: (name), select: (name, COUNT(name) AS EXPR$0)) -> Calc(select: (EXPR$0, name)) -> SinkConversion to Tuple2 -> Flat Map -> Sink: OutputFormatAdapterSink:com.alibaba.blink.streaming.connector.sls.sink.SlsOutputFormat@ccd341d\", \n    \"parallelismFactor\": 2.0,\n    \"parallelismLimit\": 32768,\n    \"parallelism\": 2,\n    \"maxParallelism\": 32768,\n    \"minResource\": {\n      \"cpuCores\": 0.1,\n      \"heapMemoryInMB\": 640,\n      \"managedMemoryInMB\": 0,\n      \"directMemoryInMB\": 0,\n      \"nativeMemoryInMB\": 0\n    }, \n    \"transformations\": [ 5, 6, 7, 8, 9 ], \n    \"inputVertexIds\": [ 0 ]\n  }\n }, \n\n \"subDags\": {\n  \"0\": {\n    \"vertices\": [ 0, 1 ], \n    \"inputFromVertices\": [], \n    \"targetTps\": 0.0\n  }\n }, \n\n \"workers\": [\n  {\n    \"numWorkers\": 1,\n    \"resourceUnits\": 0.2,\n    \"estimatedCpuCores\": 0.2,\n    \"estimatedMemoryInMB\": 768.0,\n    \"allocatedCpuCores\": 0.2,\n    \"allocatedMemoryInMB\": 819,\n    \"taskSlots\": [\n      {\n        \"topologyID\": 0,\n        \"count\": 2\n      }\n    ], \n    \"numWorkers\": 1,\n    \"resourceUnits\": 0.4,\n    \"estimatedCpuCores\": 0.2,\n    \"estimatedMemoryInMB\": 1280.0,\n    \"allocatedCpuCores\": 0.4,\n    \"allocatedMemoryInMB\": 1638,\n    \"taskSlots\": [\n      {\n        \"topologyID\": 1,\n        \"count\": 2\n      }\n    ]\n  }\n ]\n }\n }\n\n \"JobName\": \"ss2\", \n\n \"StartTime\": 1575353621000, \n\n \"Properties\": {\"#checkpoint模式\": \"EXACTLY_ONCE\" 或者 \"AT_LEAST_ONCE\" \n\n #blink_checkpoint_mode=EXACTLY

```

```
Properties : #checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE\n#blink.checkpoint.mode=EXACTLY_ONCE\n\n#checkpoint间隔时间, 单位毫秒\n#blink.checkpoint.interval.ms=180000\n\n#rocksdb的数据生命周期, 单位毫秒\n#state.backend.rocksdb.ttl.ms=129600000\n\nblink.job.timeZone=Asia/Shanghai",
"Starter": "1709064687573327",
"InputDelay": 0,
"Code": "--Blink SQL\n--*****--\n--Author: streamcompute@aliyun-test.com\n--CreateTime: 2018-12-25 14:49:51\n--Comment: 请输入业务注释信息\n--*****--\n\nCREATE TABLE sls_source (\n  `name` VARCHAR,\n  `num` VARCHAR,\n  `id` BIGINT\n) WITH (\n  `ttype`='sls',\n  `tendPoint`='http://cn-hangzhou-corp.sls.aliyuncs.com',\n  `troleArn`='acs:ram::1709064687573327:role/aliyunstreamdefaultrole',\n  `tproject`='bayer-test-jituan',\n  `tlogStore`='sls_source'\n);\n\nCREATE TABLE sls_output (\n  `tid` BIGINT,\n  `tname` VARCHAR\n) WITH (\n  `ttype`='SLS',\n  `tendPoint`='http://cn-hangzhou-corp.sls.aliyuncs.com',\n  `troleArn`='acs:ram::1709064687573327:role/aliyunstreamdefaultrole',\n  `tproject`='bayer-test-jituan',\n  `tlogStore`='sls_output'\n);\n\nINSERT INTO sls_output\nSELECT `name`,\nCOUNT(`name`)\nFROM sls_source\nGROUP BY `name`",
"ActualState": "RUNNING",
"EngineJobHandler": "application_1561366511517_113805ja10928cec91de395ff3d7593f7ff46c5",
"JobType": "FLINK_STREAM",
"LastOperator": "1709064687573327",
"Packages": "",
"ApiType": "SQL",
"Id": 214650,
"LastOperateTime": 1575353621000,
"QueueName": "root.bayer_team"
}
]
},
"TotalCount": 15,
"RequestId": "33DBCFDA-5D75-452C-8DB3-5964CFE7747C",
"PageSize": 10,
"TotalPage": 2,
"PageIndex": 1
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.12. ModifyInstanceState

您可以通过ModifyInstanceState修改运行实例的状态，实现对实例的操作。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

PUT /api/v2/projects/[projectName]/jobs/[jobName]/instances/[instanceId]/expectstate HTTPS

请求参数

名称	类型	是否必选	示例值	描述
expectState	String	是	RUNNING	期望状态： <ul style="list-style-type: none"> • RUNNING：运行中。 • PAUSED：暂停。 • TERMINATED：停止。 • SUCCESS：成功（批作业）。 • FAILED：失败（流作业）。
instanceId	Long	是	-1	实例ID： <ul style="list-style-type: none"> • 流作业：填写实际运行的实例ID或直接填写-1（当前运行的实例）。 • 批作业：可以通过ListInstance、StartJob等接口获得。
jobName	String	是	job1	作业名称
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 5px;"> ? 说明 公共云用户请忽略此参数。 </div>
isFlush	Boolean	否	true	在恢复作业时，是否使用最新配置。默认值为true。
triggerCheckpoint	Boolean	否	true	在暂停或者停止作业时，是否进行一次Checkpoint。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 5px;"> ? 说明 Blink 3.5.0及以上版本支持该参数。 </div>

返回数据

名称	类型	示例值	描述
----	----	-----	----

名称	类型	示例值	描述
RequestId	String	EA0059FA-FD69-4A83-B216-6B36B03129A7	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instances/-1/expectstate  
{\"expectState\": \"RUNNING\"}
```

正常返回示例

XML 格式

```
<requestId>EA0059FA-FD69-4A83-B216-6B36B03129A7</requestId>
```

JSON 格式

```
{  
  \"requestId\": \"EA0059FA-FD69-4A83-B216-6B36B03129A7\"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.13. GetInstanceHistoryAutoScalePlanContent

您可以通过GetInstanceHistoryAutoScalePlanContent查看到历史调优某次Planjson的详细内容，需要和获取作业历史调优Planjson列表配合使用。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instance/[instanceId]/autoscale/plancontent HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	-1	作业实例ID。流作业填写-1，批作业不支持使用。
jobName	String	Path	是	job1	作业名称。
planName	String	Query	是	2018-09-18 10:10:20	自动调优某次PlanJson的名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	是	cn-shanghai	区域ID。

返回数据

名称	类型	示例值	描述
PlanContent	String	{k:v}	PlanJson的详细内容。
RequestId	String	6C799EEF-1DD8-4C55-961C-3B9D55A87AE6	请求ID。

示例

请求示例

```
/api/v2/projects/[projectName]/jobs/[jobName]/instance/[instanceId]/autoscale/plancontent
```

正常返回示例

XML 格式

```
<RequestId>6C799EEF-1DD8-4C55-961C-3B9D55A87AE6</RequestId>
<Message>Specified parameter instanceId is not valid.</Message>
<Recommend>https://error-center.aliyun.com/status/search?Keyword=InvalidinstanceId&source=PopGw</Recommend>
<HostId>foas.cn-shanghai.aliyuncs.com</HostId>
<Code>InvalidinstanceId</Code>
```

JSON 格式

```
{
  "RequestId": "6C799EEF-1DD8-4C55-961C-3B9D55A87AE6",
  "Message": "Specified parameter instanceId is not valid.",
  "Recommend": "https://error-center.aliyun.com/status/search?Keyword=InvalidinstanceId&source=PopGw",
  "HostId": "foas.cn-shanghai.aliyuncs.com",
  "Code": "InvalidinstanceId"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.5.14. GetInstanceHistoryAutoScalePlanList

您可以通过GetInstanceHistoryAutoScalePlanList获取作业自动调优Planjson列表。配合GetInstanceHistoryAutoScalePlanContent接口可以定位到指定的Planjson。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/instance/[instanceId]/autoscale/planlist HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
instanceId	Long	Path	是	-1	作业实例ID。流作业参数值是 -1 ，批作业暂不支持。
jobName	String	Path	是	job1	作业名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	是	cn-shanghai	区域ID

返回数据

名称	类型	示例值	描述
PlanList	List	[planA,planB]	自动调优作业历史Plan名称。
RequestId	String	9C010057-9F22-45DC-A415-4E4E53181E5E	请求ID。

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instance/-1/autoscale/planlist
```

正常返回示例

XML 格式

```
<RequestId>9C010057-9F22-45DC-A415-4E4E53181E5E</RequestId>
```

JSON 格式

```
{
  "RequestId": "9C010057-9F22-45DC-A415-4E4E53181E5E"
}
```


错误码

访问[错误中心](#)查看更多错误码。

8.6. 文件夹

8.6.1. CreateFolder

您可以通过CreateFolder OpenAPI在项目对应路径下创建文件夹，并在开发控制台上保存文件夹ID和作业路径。

 **说明** 作业开发对应根目录，在开发控制台上，作业都显示在作业开发目录下。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

POST /api/v2/projects/[projectName]/folders HTTPS

请求参数

名称	类型	是否必选	示例值	描述
path	String	是	/path	文件夹路径
projectName	String	是	project1	项目名称
RegionId	String	是	cn-hangzhou	区域ID 说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
FolderId	Long	33193	文件夹ID
RequestId	String	B43F695A-F126-46C0-B336-846D6290474D	请求ID

示例

请求示例

```
/api/v2/projects/project1/folders  
{"path":"/path"}
```

正常返回示例

XML 格式

```
<RequestId>B43F695A-F126-46C0-B336-846D6290474D</RequestId>  
<FolderId>33193</FolderId>
```

JSON 格式

```
{  
  "RequestId": "B43F695A-F126-46C0-B336-846D6290474D",  
  "FolderId": 33193  
}
```


错误码

访问[错误中心](#)查看更多错误码。

8.6.2. DeleteFolder

您可以通过DeleteFolder OpenAPI删除空文件夹，删除非空文件夹会报错。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/projects/[projectName]/folders HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
path	String	是	/path1/path2	文件夹路径
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/projects/project1/folders
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{"requestId": "C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.6.3. GetFolder

您可以通过GetFolder OpenAPI获取文件夹信息，即文件夹ID和文件夹路径。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/folders HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
path	String	是	/path1	文件夹路径
projectName	String	是	project1	项目名称
RegionId	String	是	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff; margin-top: 5px;">? 说明 公共云用户请忽略此参数。</div>

返回数据

名称	类型	示例值	描述
Folder	Struct		文件夹详情
FolderId	Long	27552	文件夹ID
Path	String	/path1	文件夹路径
RequestId	String	AFD7EF32-88FA-4539-A83C-EF91DDFA2C96	请求ID

示例

请求示例

```
/api/v2/projects/project1/folders
```

正常返回示例

XML 格式

```
<RequestId>AFD7EF32-88FA-4539-A83C-EF91DDFA2C96</RequestId>
<Folder>
  <Path>/path1</Path>
  <FolderId>27552</FolderId>
</Folder>
```

JSON 格式

```
{
  "RequestId": "AFD7EF32-88FA-4539-A83C-EF91DDFA2C96",
  "Folder": {
    "Path": "/path1",
    "FolderId": 27552
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.6.4. ListChildFolder

您可以通过ListChildFolder OpenAPI获取指定主目录下所有子目录的ID和路径。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/folders/children HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
path	String	是	/path1	主目录路径
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; display: inline-block;"> ? 说明 公共云用户请忽略此参数。 </div>

返回数据

名称	类型	示例值	描述
Folders	Array of Folder		文件目录详情
Folder			
FolderId	Long	33215	文件夹ID
Path	String	/path1/path2	文件夹目录
RequestId	String	DA0A418B-E495-4BCC-B44F-1BA3CE731103	请求ID

示例

请求示例

```
/api/v2/projects/project1/folders/children
```

正常返回示例

XML 格式

```
<RequestId>DA0A418B-E495-4BCC-B44F-1BA3CE731103</RequestId>
<Folders>
  <Folder>
    <Path>/path1/path2</Path>
    <FolderId>33215</FolderId>
  </Folder>
</Folders>
```

JSON 格式

```
{
  "RequestId": "DA0A418B-E495-4BCC-B44F-1BA3CE731103",
  "Folders": {
    "Folder": [
      {
        "Path": "/path1/path2",
        "FolderId": 33215
      }
    ]
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.6.5. MVFolder

您可以通过MVFolder OpenAPI移动指定项目下的文件夹。移动非空文件夹时，会将文件夹下的作业一起移动。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/projects/[projectName]/folders HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
destPath	String	是	/path2	目标路径
projectName	String	是	project1	项目名称
srcPath	String	是	/path1	原路径
RegionId	String	否	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 5px;">? 说明 公共云用户请忽略此参数。</div>

返回数据

名称	类型	示例值	描述
RequestId	String	D34CBA74-DED8-4B1E-80B4-DE4C58291726	请求ID

示例

请求示例

```
/api/v2/projects/project1/folders  
{\"srcPath\": \" /path1\", \"destPath\": \" /path2\"}
```

正常返回示例

XML 格式

```
<RequestId>D34CBA74-DED8-4B1E-80B4-DE4C58291726</RequestId>
```

JSON 格式

```
{  
  \"RequestId\": \"D34CBA74-DED8-4B1E-80B4-DE4C58291726\"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7. 作业

8.7.1. CheckRawPlanJson

您可以组合调用CheckRawPlanJson和GetRawPlanJson接口，来获取某个作业的PlanJson。
CheckRawPlanJson接口发出调用请求后会返回一个Session ID，GetRawPlanJson获取该session ID后可以查询到PlanJson的详情。

 **说明** 建议循环调用CheckRawPlanJson直到成功获取。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/planjson/check HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID。
sessionId	String	Query	是	j6b43mm10nnzywsixmeh8maf lgt6xq5afyeufl ow3fravqlby0u di605mi7sty2p em2w9nysqizt ag5je4esvmgq 04pdcazgdthy3 u5riy6na0fz6f mgph1k6b62b vjs7kqorn69ujs k0	当使用CheckRawPlanJson发送请求后，会返回一个SessionId，将该sessionId填在此处。

返回数据

名称	类型	示例值	描述
PlanJsonInfo	Struct		PlanJson详情。
ErrorMessage	String	eror	错误信息。
PlanJson	String	{a:b}	执行计划。
Status	String	success	GetRawPlanJson的执行状态： <ul style="list-style-type: none">• Sessionin run：执行中。• success：成功。• fail：失败。
RequestId	String	FD0FF8C0-779A-45EB-9674-FF3E127B10D2	请求ID。

示例

请求示例

```
/api/v2/projects/project1/jobs/project1/planjson/check
```

正常返回示例

XML 格式

```
<RequestId>78261AC6-B5C9-4514-B82F-39BCC45A5D63</RequestId>
```

JSON 格式

```
{  
  "RequestId": "78261AC6-B5C9-4514-B82F-39BCC45A5D63"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.2. CommitJob

您可以通过CommitJob提交作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/projects/[projectName]/jobs/[jobName]/commit HTTPS
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称
projectName	String	Path	是	project1	项目名称
RegionId	String	Header	是	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。
isOnOff	Boolean	Body	否	true	是否启用AutoScale。
maxCU	Float	Body	否	10.0	最大CU数，作业运行期间使用的资源不会超过该上限。单位为CU，1 CU=1 Core + 4 GB RAM。

名称	类型	位置	是否必选	示例值	描述
configure	String	Body	否	<code>{"fetch_delay": 1.0,"recommendOnly":true}</code>	<p>您可以配置以下几项：</p> <ul style="list-style-type: none"> 调优策略：系统根据选择的策略和期望值对作业自动调优，以达到期望值。目前只支持数据滞留时间 <code>fetch_delay</code>，单位为秒。 期望值：数据滞留时间阈值，当 <code>Source</code>的数据滞留时间超过该阈值时，触发 <code>AutoScale</code>并优化作业并发度。 建议 <code>recommendOnly: true</code>表示开启 <code>PlanJson</code>建议功能。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>? 说明</p> <ul style="list-style-type: none"> 仅支持通过 <code>AutoScale</code>后台服务提供 <code>PlanJson</code>建议。 仅在 <code>AutoScale</code>设置为 <code>true</code>时，<code>recommendOnly</code>生效。 仅 <code>Blink 3.7.8</code>以上版本支持 <code>recommendOnly</code>。 仅 <code>Blink 3.x</code>版本支持 <code>AutoScale</code>功能。 当开启 <code>AutoScale</code>时，需要传入 <code>maxCu</code>和 <code>configure</code>参数。 </div>
recommendOnly	Boolean	Body	否	false	<p>启用 <code>AutoScale</code>后，系统是否自动修改您的 <code>PlanJson</code>：</p> <ul style="list-style-type: none"> <code>true</code>：系统不会自动修改。 <code>false</code>（默认值）：系统会自动修改。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 10px;"> <p>? 说明 可以通过SDK中 <code>getInstanceHistoryAutoScalePlanListRequest</code>获取 <code>PlanJson</code>列表。</p> </div>
suspendPeriodParam.N.plan	String	Body	否	ORIGINAL	<p>指定暂停作业后使用的 <code>PlanJson</code>：</p> <ul style="list-style-type: none"> ORIGINAL：初始的 <code>PlanJson</code>。 CURRENT：当前的 <code>PlanJson</code>。

名称	类型	位置	是否必选	示例值	描述
suspendPeriodParam.N.policy	String	Body	否	EVERY_DAY_TIME	<p>指定暂停时间段：</p> <ul style="list-style-type: none"> EVERY_DAY_TIME：指定每天的暂停时间段，需要填写暂停起始时间（startTime:HH:mm）和暂停结束时间（endTime:HH:mm）。 SPECIFIED_TIME：指定具体的暂停时间段，需要填写暂停起始时间（startTime:yyyy-MM-dd HH:mm）和暂停结束时间（endTime:yyyy-MM-dd HH:mm）。
suspendPeriodParam.N.startTime	String	Body	否	18:20	<p>暂停时间段的开始时间。</p> <p> 说明 根据policy设置不同格式的时间格式。</p>
suspendPeriodParam.N.endTime	String	Body	否	18:25	<p>暂停时间段的结束时间。</p> <p> 说明 根据policy设置不同格式的时间格式。</p>
suspendPeriods	String	Body	否	<pre>[{"endTime":"18:20","startTime":"18:00","plan":"ORIGINAL","policy":"EVERY_DAY_TIME"}, {"endTime":"19:00","startTime":"18:40","plan":"CURRENT","policy":"EVERY_DAY_TIME"}, {"endTime":"20:20-07-17 20:00","startTime":"2020-07-17 19:30","plan":"CURRENT","policy":"SPECIFIED_TIME"}]</pre>	<p>开启AutoScale后，指定暂停AutoScale时间段和使用的Planjson。</p>

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/commit
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{  
  "requestId": "C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.3. CreateJob

您可以通过CreateJob OpenAPI在指定项目下创建作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/projects/[projectName]/jobs HTTPS
```

请求参数

名称	类型	位置	是否必选	示例值	描述
----	----	----	------	-----	----

名称	类型	位置	是否必选	示例值	描述
apiType	String	Body	是	Datastream	API类型： <ul style="list-style-type: none"> DATASTREAM: Datastream作业 SQL: SQL作业
clusterId	String	Body	是	rcmp9x37ztfb63g1x7lt****	集群ID  说明 您可以使用listcluster获取集群ID信息。
code	String	Body	是	blink.main.class=com.hjc.test.blink_test.ChinanTopSpeedWindowing2\r\nblink.job.name=datastream_pretest	作业代码： <ul style="list-style-type: none"> SQL作业: 填写SQL代码。 DataStream作业: 填写DataStream作业的相关参数。
engineVersion	String	Body	是	blink_2.2.4	引擎版本
jobName	String	Body	是	job1	作业名称
jobType	String	Body	是	FLINK_STREAM	作业类型： <ul style="list-style-type: none"> FLINK_STREAM: 流作业 FLINK_BATCH: 批作业
planJson	String	Body	是	{a:b}	执行计划
projectName	String	Path	是	project1	项目名称
queueName	String	Body	是	root.default	队列名称
RegionId	String	Header	是	区域ID	区域ID  说明 公共云用户请忽略此参数。

名称	类型	位置	是否必选	示例值	描述
properties	String	Body	否	{k:v}	作业运行的相关参数
packages	String	Body	否	package1,package2	JAR包名称，多个使用逗号(,)分隔。
description	String	Body	否	test	作业备注
folderId	Long	Body	否	123	文件夹ID

返回数据

名称	类型	示例值	描述
RequestId	String	B661F03F-0F9D-4EFF-A40F-4ED1519A549A	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs
{"jobName":"job1","jobType":"FLINK_STREAM","apiType":"Datastream","code":"blink.main.class=com.hjc.test.blink_test.ChinanTopSpeedWindowing2\r\nblink.job.name=datastream_prestest","planJson":{"a:b"},"engineVersion":"blink_2.2.4","clusterId":"rcmp9x37ztfb63g1x7lt****","queueName":"root.default"}
```

正常返回示例

XML 格式

```
<requestId>B661F03F-0F9D-4EFF-A40F-4ED1519A549A</requestId>
```

JSON 格式

```
{
  "requestId": "B661F03F-0F9D-4EFF-A40F-4ED1519A549A"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.4. DeleteJob

您可以通过DeleteJob OpenAPI删除已经下线的作业，删除未下线作业会报错。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/projects/[projectName]/jobs/[jobName] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
jobName	String	是	job1	作业名称
projectName	String	是	project1	项目名称
RegionId	String	是	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> ? 说明 公共云用户请忽略此参数。 </div>

返回数据

名称	类型	示例值	描述
RequestId	String	78261AC6-B5C9-4514-B82F-39BCC45A5D63	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1
```

正常返回示例

XML 格式

```
<RequestId>78261AC6-B5C9-4514-B82F-39BCC45A5D63</RequestId>
```

JSON 格式

```
{
  "RequestId": "78261AC6-B5C9-4514-B82F-39BCC45A5D63"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.5. GetJob

您可以通过GetJob获取作业详细信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
jobName	String	是	job1	作业名称
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
Job	Struct		作业详情

名称	类型	示例值	描述
ApiType	String	DATASTREAM	作业类型：DATASTREAM或SQL。
ClusterId	String	xxx	集群ID
Code	String	xxxxxx	运行代码： <ul style="list-style-type: none"> SQL作业返回SQL代码。 Datastream作业返回Datastream配置。
CreateTime	Long	1595493794000	作业创建时间
Creator	String	xxx	作业创建者
Description	String	test	作业备注信息
EngineVersion	String	blink-3.5.0-hotfix	引擎版本
FileId	String	75724	文件ID
FolderId	Long	26808	文件夹ID
IsCommitted	Boolean	true	是否为上线状态。
JobId	String	rd8r1lgccsnchh9fq wukw6e	作业ID
JobName	String	datastream_enable	作业名称
JobType	String	FLINK_STREAM	作业类型： <ul style="list-style-type: none"> FLINK_STREAM：流作业。 FLINK_BATCH：批作业。
Modifier	String	xxx	最新修改者
ModifyTime	Long	1597743765000	最新修改时间

名称	类型	示例值	描述
Packages	String	big-data-1.0-snapshot.jar	实例引用的包名称，多个包使用逗号（,）分隔，未引用为空。
PlanJson	String	{a:b}	作业上线的执行计划
ProjectName	String	bayes_team	项目名称
Properties	String	k:v	作业运行配置参数
QueueName	String	root.default	队列名称
RequestId	String	2C6E2F4C-2FA1-4159-87DA-676DB15F4826	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1
```

正常返回示例

XML 格式

```
<requestId>2C6E2F4C-2FA1-4159-87DA-676DB15F4826</requestId>
<job>
  <jobName>datastream_enable</jobName>
  <projectName>bayes_team</projectName>
  <jobType>FLINK_STREAM</jobType>
  <apiType>DATASTREAM</apiType>
  <code>xxxxx</code>
  <packages>big-data-1.0-snapshot.jar</packages>
  <isCommitted>true</isCommitted>
  <creator>xxx</creator>
  <createTime>1595493794000</createTime>
  <modifier>xxx</modifier>
  <modifyTime>1597743765000</modifyTime>
  <engineVersion>blink-3.5.0-hotfix</engineVersion>
  <clusterId>xxx</clusterId>
  <queueName>root.default</queueName>
  <folderId>26808</folderId>
  <jobId>rd8r1lgccsnchh9fqwukw6e</jobId>
  <fileId>75724</fileId>
</job>
```

JSON 格式

```
{
  "requestId": "2C6E2F4C-2FA1-4159-87DA-676DB15F4826",
  "job": {
    "jobName": "datastream_enable",
    "projectName": "bayes_team",
    "jobType": "FLINK_STREAM",
    "apiType": "DATASTREAM",
    "code": "xxxxx",
    "packages": "big-data-1.0-snapshot.jar",
    "isCommitted": true,
    "creator": "xxx",
    "createTime": 1595493794000,
    "modifier": "xxx",
    "modifyTime": 1597743765000,
    "engineVersion": "blink-3.5.0-hotfix",
    "clusterId": "xxx",
    "queueName": "root.default",
    "folderId": 26808,
    "jobId": "rd8r1lgccsnchh9fqwukw6e",
    "fileId": 75724
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.6. ListJob

您可以通过ListJob OpenAPI获取指定project下满足条件的Job信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
----	----	----	------	-----	----

名称	类型	位置	是否必选	示例值	描述
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID。
pageSize	Integer	Query	否	10	每页返回Job数。
pageIndex	Integer	Query	否	1	页码。
jobName	String	Query	否	job1	作业名称。
jobType	String	Query	否	FLINK_STREAM	作业种类： <ul style="list-style-type: none"> FLINK_STREAM：流作业。 FLINK_BATCH：批作业。
apiType	String	Query	否	SQL	API类型：DATASTREAM和SQL。
engineVersion	String	Query	否	blink_2.2.4	引擎版本。
clusterId	String	Query	否	d6wxwo5tnrmuamx2ly3m7vklz	集群ID。
queueName	String	Query	否	root.default	队列名称。
folderId	Long	Query	否	123	文件夹ID。
isShowFullField	Boolean	Query	否	true	123。

返回数据

名称	类型	示例值	描述
Jobs	Array of Job		作业详情。
Job			
ApiType	String	SQL	API类型：DATASTREAM和SQL两种。

名称	类型	示例值	描述
ClusterId	String	d6wxwo5tnrmuamx 2ly3m7vkz	集群ID。 ? 说明 您可以使用 <code>listcluster</code> 获取集群ID信息。
Code	String	code	作业代码。
CreateTime	Long	1548397575000	作业创建时间。
Creator	String	xxxx	创建者。
Description	String	test	作业备注描述。
EngineVersion	String	blink_2.2.4	引擎版本。
FolderId	Long	123	文件夹ID
IsCommitted	Boolean	true	作业是否已经提交运行。
JobId	String	123	Job ID。
JobName	String	job1	作业名称。
JobType	String	FLINK_STREAM	作业种类： <ul style="list-style-type: none"> FLINK_STREAM：流作业。 FLINK_BATCH：批作业。
Modifier	String	xxxx	最新修改者。
ModifyTime	Long	1548397575000	最新修改时间。
Packages	String	package1.jar	Package名称。
PlanJson	String	{a:b}	执行计划。

名称	类型	示例值	描述
ProjectName	String	project1	项目名称。
Properties	String	k:v	作业运行配置。
QueueName	String	root.default	队列名称。
PageIndex	Integer	1	页码。
PageSize	Integer	10	每页Job数。
RequestId	String	FD0FF8C0-779A-45EB-9674-FF3E127B10D2	请求ID。
TotalCount	Long	50	总Job数。
TotalPage	Integer	5	总页数。

示例

请求示例

```
/api/v2/projects/project1/jobs
```

正常返回示例

XML 格式

```
<RequestId>5A0AF0E2-A715-40FD-8A2B-1EDBF4213489</RequestId>
```

JSON 格式

```
{
  "RequestId": "5A0AF0E2-A715-40FD-8A2B-1EDBF4213489"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.7. OfflineJob

您可以通过OfflineJob OpenAPI下线作业。

 说明 下线作业前，需要先停止运行作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/projects/[projectName]/jobs/[jobName]/offline HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
jobName	String	是	job1	作业名称
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	5A0AF0E2-A715-40FD-8A2B-1EDBF4213489	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/offline
```

正常返回示例

XML 格式

```
<RequestId>5A0AF0E2-A715-40FD-8A2B-1EDBF4213489</RequestId>
```

JSON 格式

```
{
  "RequestId": "5A0AF0E2-A715-40FD-8A2B-1EDBF4213489"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.8. StartJob

您可以通过StartJob OpenAPI启动指定项目下的目标作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/projects/[projectName]/jobs/[jobName]/instance HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称
projectName	String	Path	是	project1	项目名称
RegionId	String	Header	是	cn-hangzhou	区域ID
parameterJson	Json	Body	否	{"blink.checkpoint.interval.ms": "180000"}	JSON格式的配置参数，详情请参见配置参数表。

配置参数

参数名称	说明	备注
------	----	----

参数名称	说明	备注
blink.checkpoint.mode	Checkpoint模式	AT_LEAST_ONCE：至少计算一次。 EXACTLY_ONCE：精确计算一次。
blink.checkpoint.interval.ms	Checkpoint间隔时间	数据类型为LONG，例如180000，单位为毫秒。
blink.checkpoint.timeout.ms	Checkpoint超时时间	数据类型为LONG，例如600000，单位为毫秒，默认值为10分钟。
blink.job.option.jmMemMB	Job Manager内存	例如1024 MB。
blink.job.option	传给Flink Run的参数，可以传递多个参数，具体请参见Flink命令说明。	例如设置Task Manager的JVM线程的堆栈大小，可以配置： blink.job.option=-yD env.java.opts='-Xss8192K'。
client.jvm.option	编译阶段的JVM	例如：-Xss2048k。
blink.miniBatch.allowLatencyMs	Batch最大允许延迟时间	添加或删除该参数Job状态会丢失，修改值大小状态不会丢失，数据类型为LONG，例如10000，单位为毫秒。
blink.miniBatch.size	Batch最大条数	添加或删除该参数Job状态会丢失，修改值大小状态不会丢失；此参数不能设置过大，如果一个Batch实际去完重的数据超过65536，在访问statebackend时可能触发JNI OOM异常；数据类型为LONG，例如1000。

参数名称	说明	备注
blink.microBatch.allowLatencyMs	Batch最大允许延迟时间，与minibatch的区别在于microbatch是基于batchmark触发的，而不是timer，具有更高的效率。	数据类型为LONG，例如10000，Blink 2.2及以上版本支持该参数，单位为毫秒。
blink.localAgg.enabled	是否开启Local-Global Agg。需要开启MiniBatch，Local-Global Agg功能才生效。	添加或删除该参数Job状态会丢失，修改值大小状态不会丢失；Blink 2.2版本默认值为true，其他版本默认值为false。
blink.partialAgg.enabled	是否开启Partial-Final Agg。Parital-Final Agg常用于解决COUNT DISTINCT热点问题。	默认值为false，Blink 2.2.0及以上版本支持该参数。
sql.exec.mini-batch.window.enabled	是否开启minibatch window	默认值为false（不开启），Blink 3.2.0及以上版本支持该参数。
sql.exec.source.idle.timeout.ms	Source空闲超时的时间，当source被标记为空闲后，下游就不会再等待该分区source发来的watermark，从而能触发window。	默认值为-1（不开启），取值大于0则表示开启。Blink 3.2.0及以上版本支持该参数。
blink.state.ttl.ms	TopN和groupBy节点的状态过期清理时间。状态在指定时间内没有被更新过则会被清除。	数据类型为LONG，例如60000，单位为毫秒，默认为不清除。
blink.topn.cache.size	TopN缓存数据的条数。例如Top100，配置缓存10000条，即缓存了100组数据。	数据类型为LONG，默认值为10000。
blink.job.submit.timeoutInSeconds	SQL编译超时时间，建议设置最大不超过250s。	数据类型为LONG，默认值为180。

参数名称	说明	备注
blink.job.timeZone	作业的时区	默认为Bayes前端时区, 例如Asia/Shanghai。
blink.object.reuse	是否开启对象重用。如果不开启, chain一起的operator之前的数据传输会发生序列化或反序列化	Blink 2.2.0及以上版本默认值为true, 其他版本默认值为false。
blink.auto.watermark.interval.ms	Watermark发送的频率。需要权衡延迟和性能。太高会导致延迟增加, 太低会导致频繁发送watermark, 影响性能。	默认值为100ms。
blink.job.timestamp.reserve.ms	对于Ads、Petadata、SLS、Elasticsearch、DataHub类型的Sink, 在写入timestamp类型时支持毫秒精度。	默认值为false, Blink 2.2.6及以上版本支持该参数。
yarn.app.blink.source.buffer-len	Connector source里面RecordReader线程作为生产者, 上层ParallelReader线程作为消费者, 中间的阻塞队列的长度, 当数据源单条数据较小, 数据量极大时, 调高此项能提升部分性能。	默认值为10。
yarn.app.blink.source.source.idle-interval	Connector source里面ParallelReader无数据可读时, sleep的时间。	默认值为10, 单位为100ms, Blink 2.2.5及以上版本支持该参数。
sql.exec.fault.tolerance.enabled	是否开启容错机制	默认值为false, Blink 3.4.0及以上版本支持该参数。
state.backend.niagara.ttl.ms	Niagara StateBackend中数据的保存时间	数据类型为LONG, 例如129600000, 默认值为一半天。

参数名称	说明	备注
state.backend.block.cache.size.mb	Niagara StateBackend 读缓存大小	默认值为256 MB。
state.backend.mem.table.size.mb	Niagara StateBackend 写缓存大小	默认值为128 MB。
state.backend.gemini.ttl.ms	Gemini StateBackend数据的保存时间	数据类型为LONG，例如129600000，默认值为一天半。
state.backend.gemini.mem.size.mb	Gemini StateBackend Native内存大小	默认值为384 MB。
state.backend.gemini.ttl.ms	Gemini StateBackend中数据的保存时间	数据类型为LONG，例如129600000，默认值为一天半。
state.backend.block.cache.size.mb	Gemini StateBackend读缓存大小	默认值为256 MB。
state.backend.mem.table.size.mb	Gemini StateBackend写缓存大小	默认值为128 MB。

返回数据

名称	类型	示例值	描述
RequestId	String	5313EF6E-6E3F-45DB-8E33-20DA279D8F3F	请求ID
instanceId	Long	415854	实例ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/instance
```

正常返回示例

XML 格式

```
<instanceId>415854</instanceId>
<RequestId>5313EF6E-6E3F-45DB-8E33-20DA279D8F3F</RequestId>
```

JSON 格式

```
{
  "instanceId": 415854,
  "RequestId": "5313EF6E-6E3F-45DB-8E33-20DA279D8F3F"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.9. UpdateJob

您可以通过UpdateJob OpenAPI更新作业属性。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/projects/[projectName]/jobs/[jobName] HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称
projectName	String	Path	是	project1	项目名称
packages	String	Body	否	package1.jar	package名称
planJson	String	Body	否	{a:b}	作业执行计划
code	String	Body	否	code	作业代码

名称	类型	位置	是否必选	示例值	描述
RegionId	String	Header	否	cn-hangzhou	区域ID
description	String	Body	否	test	作业备注
engineVersion	String	Body	否	blink_2.2.4	引擎版本
clusterId	String	Body	否	d6wxwo5tnrm uamx2ly3m7vk z	集群ID 
queueName	String	Body	否	root.default	队列名称
folderId	Long	Body	否	123	文件夹ID
properties	String	Body	否	blink.checkpoint.interval.ms: 180000	作业运行配置参数，参数详情请参见配置参数表。

配置参数

参数名称	说明	备注
blink.checkpoint.mode	Checkpoint模式	AT_LEAST_ONCE：至少计算一次。 EXACTLY_ONCE：精确计算一次。
blink.checkpoint.interval.ms	Checkpoint间隔时间	数据类型为LONG，例如180000，单位为毫秒。
blink.checkpoint.timeout.ms	Checkpoint超时时间	数据类型为LONG，例如600000，单位为毫秒，默认值为10分钟。
blink.job.option.jmMemMB	Job Manager内存	例如1024 MB。

参数名称	说明	备注
blink.job.option	传给Flink Run的参数，可以传递多个参数，具体请参见Flink命令说明。	例如设置Task Manager的JVM线程的堆栈大小，可以配置： blink.job.option=-yD env.java.opts='-Xss8192K'。
client.jvm.option	编译阶段的JVM	例如：-Xss2048k。
blink.miniBatch.allowLatencyMs	Batch最大允许延迟时间	添加或删除该参数Job状态会丢失，修改值大小状态不会丢失，数据类型为LONG，例如10000，单位为毫秒。
blink.miniBatch.size	Batch最大条数	添加或删除该参数Job状态会丢失，修改值大小状态不会丢失；此参数不能设置过大，如果一个Batch实际去完重的数据超过65536，在访问statebackend时可能触发JNI OOM异常；数据类型为LONG，例如1000。
blink.microBatch.allowLatencyMs	Batch最大允许延迟时间，与minibatch的区别在于microbatch是基于batchmark触发的，而不是timer，具有更高的效率。	数据类型为LONG，例如10000，Blink 2.2及以上版本支持该参数，单位为毫秒。
blink.localAgg.enabled	是否开启Local-Global Agg。需要开启MiniBatch，Local-Global Agg功能才生效。	添加或删除该参数Job状态会丢失，修改值大小状态不会丢失；Blink 2.2版本默认值为true，其他版本默认值为false。
blink.partialAgg.enabled	是否开启Partial-Final Agg。Parital-Final Agg常用于解决COUNT DISTINCT热点问题。	默认值为false，Blink 2.2.0及以上版本支持该参数。

参数名称	说明	备注
sql.exec.mini-batch.window.enabled	是否开启minibatch window	默认值为false（不开启），Blink 3.2.0及以上版本支持该参数。
sql.exec.source.idle.timeout.ms	Source空闲超时的时间，当source被标记为空闲后，下游就不会再等待该分区source发来的watermark，从而能触发window。	默认值为-1（不开启），取值大于0则表示开启。Blink 3.2.0及以上版本支持该参数。
blink.state.ttl.ms	TopN和groupBy节点的状态过期清理时间。状态在指定时间内没有被更新过则会被清除。	数据类型为LONG，例如60000，单位为毫秒，默认为不清除。
blink.topn.cache.size	TopN缓存数据的条数。例如Top100，配置缓存10000条，即缓存了100组数据。	数据类型为LONG，默认值为10000。
blink.job.submit.timeoutInSeconds	SQL编译超时时间，建议设置最大不超过250s。	数据类型为LONG，默认值为180。
blink.job.timeZone	作业的时区	默认为Bayes前端时区，例如Asia/Shanghai。
blink.object.reuse	是否开启对象重用。如果不开启，chain一起的operator之前的数据传输会发生序列化或反序列化	Blink 2.2.0及以上版本默认值为true，其他版本默认值为false。
blink.auto.watermark.interval.ms	Watermark发送的频率。需要权衡延迟和性能。太高会导致延迟增加，太低会导致频繁发送watermark，影响性能。	默认值为100ms。

参数名称	说明	备注
blink.job.timestamp.reserve.ms	对于Ads、Petadata、SLS、Elasticsearch、DataHub类型的Sink，在写入timestamp类型时支持毫秒精度。	默认值为false，Blink 2.2.6及以上版本支持该参数。
yarn.app.blink.source.buffer-len	Connector source里面RecordReader线程作为生产者，上层ParallelReader线程作为消费者，中间的阻塞队列的长度，当数据源单条数据较小，数据量极大时，调高此项能提升部分性能。	默认值为10。
yarn.app.blink.source.source.idle-interval	Connector source里面ParallelReader无数据可读时，sleep的时间。	默认值为10，单位为100ms，Blink 2.2.5及以上版本支持该参数。
sql.exec.fault.tolerance.enabled	是否开启容错机制	默认值为false，Blink 3.4.0及以上版本支持该参数。
state.backend.niagara.ttl.ms	Niagara StateBackend中数据的保存时间	数据类型为LONG，例如129600000，默认值为一夫半。
state.backend.block.cache.size.mb	Niagara StateBackend读缓存大小	默认值为256 MB。
state.backend.mem.table.size.mb	Niagara StateBackend写缓存大小	默认值为128 MB。
state.backend.gemini.ttl.ms	Gemini StateBackend数据的保存时间	数据类型为LONG，例如129600000，默认值为一夫半。
state.backend.gemini.mem.size.mb	Gemini StateBackend Native内存大小	默认值为384 MB。

参数名称	说明	备注
state.backend.gemini.ttl.ms	Gemini StateBackend中数据的保存时间	数据类型为LONG，例如129600000，默认值为一天半。
state.backend.block.cache.size.mb	Gemini StateBackend读缓存大小	默认值为256 MB。
state.backend.mem.table.size.mb	Gemini StateBackend写缓存大小	默认值为128 MB。

返回数据

名称	类型	示例值	描述
RequestId	String	C3FC248C-1CF4-4A64-97C1-0BD667A4D09D	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1
```

正常返回示例

XML 格式

```
<RequestId>C3FC248C-1CF4-4A64-97C1-0BD667A4D09D</RequestId>
```

JSON 格式

```
{
  "RequestId": "C3FC248C-1CF4-4A64-97C1-0BD667A4D09D"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.10. ValidateJob

您可以通过ValidateJob OpenAPI对作业进行语法检查。如果检查通过，则返回参数。如果检查不通过，则会返回解析SQL出现错误的报错信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/validate HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称
projectName	String	Path	是	project1	项目名称
RegionId	String	Header	否	cn-hangzhou	区域ID

返回数据

名称	类型	示例值	描述
JobInOut	Struct		作业输入输出
Dims	Array of Dim		作业中的维表
Dim			
Alias	String	phoneNumber	维表别名
Name	String	phoneNumber	表名
Properties	Map	k:v	配置组
Type	String	rds	表类型
Workspace	String	project1	表所属项目

名称	类型	示例值	描述
Inputs	Array of Input		输入数组详情
Input			
Alias	String	datahub_input1	源表别名
Name	String	datahub_input1	表名
Properties	Map	k:v	配置组
Type	String	datahub	表类型
Workspace	String	project1	表所属项目
Outputs	Array of Output		返回数组详情
Output			
Alias	String	result_infor	结果表别名
Name	String	result_infor	表名
Properties	Map	k:v	配置组
Type	String	rds	表类型
Workspace	String	project1	表所属项目
RequestId	String	5A3EB3CC-D899-4B0B-AF1F-1C0C0B6A0F91	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/validate
```

正常返回示例

XML 格式

```
<JobInOut>
  <Outputs>
    <Output>
      <Type>rds</Type>
      <Alias>result_infor</Alias>
      <Properties>
        <type>rds</type>
      </Properties>
    </Output>
  </Outputs>
  <Dims>
    <Dim>
      <Type>ots</Type>
      <Alias>phoneNumber</Alias>
      <Properties>
        <type>ots</type>
      </Properties>
    </Dim>
  </Dims>
  <Inputs>
    <Input>
      <Type>datahub</Type>
      <Alias>datahub_input1</Alias>
      <Properties>
        <type>datahub</type>
      </Properties>
    </Input>
  </Inputs>
</JobInOut>
<RequestId>5A3EB3CC-D899-4B0B-AF1F-1C0C0B6A0F91</RequestId>
```

JSON 格式

```
{
  "JobInOut": {
    "Outputs": {
      "Output": [
        {
          "Type": "rds",
          "Alias": "result_infor",
          "Properties": {
            "type": "rds"
          },
          "Workspace": ""
        }
      ]
    },
    "Dims": {
      "Dim": [
        {
          "Type": "ots",
          "Alias": "phoneNumber",
          "Properties": {
            "type": "ots"
          }
        }
      ]
    },
    "Inputs": {
      "Input": [
        {
          "Type": "datahub",
          "Alias": "datahub_input1",
          "Properties": {
            "type": "datahub"
          }
        }
      ]
    },
    "RequestId": "5A3EB3CC-D899-4B0B-AF1F-1C0C0B6A0F91"
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.11. CalcPlanJsonResource

您可以通过CalcPlanJsonResource获取作业执行计划所需要的CPU、内存等资源消耗信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/planjson-resource HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-shanghai	地域ID。

返回数据

名称	类型	示例值	描述
PlanJsonResource	Struct		执行计划的资源信息。
Cores	Float	5	作业所消耗的CPU内核个数。
MemoryGB	Float	12.5	作业所消耗的内存大小，单位是GB。
RequestId	String	4D3D89FC-4C0F-4FF5-82BD-BF8C00F22129	请求ID。

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/planjson-resource
```

正常返回示例

XML 格式

```
<PlanJsonResource>
  <Cores>5</Cores>
  <MemoryGB>12.5</MemoryGB>
</PlanJsonResource>
<RequestId>4D3D89FC-4C0F-4FF5-82BD-BF8C00F22129</RequestId>
```

JSON 格式

```
{
  "PlanJsonResource": {
    "Cores": 5,
    "MemoryGB": 12.5
  },
  "RequestId": "4D3D89FC-4C0F-4FF5-82BD-BF8C00F22129"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.12. UpdateAutoScaleConfig

您可以通过本API对开启AutoScale功能的作业进行AutoScale功能的关闭或者关闭后的再次开启。

UpdateAutoScaleConfig仅适用于使用AutoScale功能启动的作业，未使用AutoScale功能的作业不支持该API。

调试

您可以在[OpenAPI Explorer](#)中直接运行该接口，免去您计算签名的困扰。运行成功后，[OpenAPI Explorer](#)可以自动生成SDK代码示例。

请求头

除公共请求头，该接口还需要指定以下请求头。关于公共请求头，请参见公共请求参数文档。

名称	类型	是否必选	示例	描述
RegionId	String	否	cn-shanghai	地域ID

请求语法

```
PUT /api/v2/projects/[projectName]/jobs/[jobName]/instance/[instanceId]/autoscale/config HTTP/1.1
```

请求参数

名称	类型	是否必选	示例值	描述
configJson	Json	是	{"autoscale.enable": "true"}	设置开启或者关闭。

名称	类型	是否必选	示例值	描述
instanceId	Long	是	-1	作业实例ID。流作业填 -1，批作业暂时不支持该API。
jobName	String	是	job1	作业名称。
projectName	String	是	project1	项目名称。

返回数据

名称	类型	示例值	描述
RequestId	String	A017B343-33DC-4A97-BAC5-4B57872D5B61	请求ID

示例

请求示例

```
PUT /api/v2/projects/[projectName]/jobs/[jobName]/instance/[instanceId]/autoscale/config HTTP/1.1
RegionId:
公共请求头
```

正常返回示例

XML 格式

```
<RequestId>A017B343-33DC-4A97-BAC5-4B57872D5B61</RequestId>
<Metrics>
  <metricName>et2hunbu.yarn.resourcemanager.jvmmetrics.MemHeapUsed</metricName>
  <dps>
    <1574244620>3675.839599609375</1574244620>
    <1574244640>3817.712646484375</1574244640>
    <1574244660>3864.84130859375</1574244660>
    <1574244680>3973.963134765625</1574244680>
    <1574244700>4121.57958984375</1574244700>
    <1574244720>4170.6123046875</1574244720>
    <1574244740>4279.62109375</1574244740>
    <1574244760>4427.05615234375</1574244760>
    <1574244780>4472.11474609375</1574244780>
  </dps>
  <summary>6700.7467978583445</summary>
</Metrics>
```

JSON 格式

```
{
  "Metrics":[
    {
      "tags":{},
      "summary":6700.7467978583445,
      "dps":{
        "1574244640":3817.712646484375,
        "1574244720":4170.6123046875,
        "1574244780":4472.11474609375,
        "1574244740":4279.62109375,
        "1574244680":3973.963134765625,
        "1574244760":4427.05615234375,
        "1574244700":4121.57958984375,
        "1574244660":3864.84130859375,
        "1574244620":3675.839599609375
      },
      "metricName":"et2hunbu.yarn.resourcemanager.jvmmetrics.MemHeapUsed"
    }
  ],
  "RequestId":"A017B343-33DC-4A97-BAC5-4B57872D5B61"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.13. GetJobLatestAutoScalePlan

您可以通过GetJobLatestAutoScalePlan获取作业最新的自动调优执行计划。

 **说明** 仅开启自动调优功能的作业支持GetJobLatestAutoScalePlan。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/autoscale/latestplanjson HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名
projectName	String	Path	是	project1	项目名
RegionId	String	Header	是	cn-shanghai	地区ID。

返回数据

名称	类型	示例值	描述
PlanJson	String	{k:v}	作业执行计划
RequestId	String	C0D76C25-2056-43E4-B4EA-A7BBF774BAC1	请求ID

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/autoscale/latestplanjson
```

正常返回示例

XML 格式

```
<RequestId>C0D76C25-2056-43E4-B4EA-A7BBF774BAC1</RequestId>
```

JSON 格式

```
{  
  "RequestId": "C0D76C25-2056-43E4-B4EA-A7BBF774BAC1"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.7.14. GetRawPlanJson

您可以通过GetRawPlanJson OpenAPI获取作业初始执行计划。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/jobs/[jobName]/planjson HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
jobName	String	Path	是	job1	作业名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID。
autoconfEnable	Boolean	Query	否	true	是否开启智能调优，取值如下： <ul style="list-style-type: none"> true：开启，根据作业历史Metric，生成一份执行计划。 false：不开启，使用默认执行计划。
expectedCore	Float	Query	否	1	期望CPU数。
expectedGB	Float	Query	否	4	期望内存。
AdvisorAction	String	Query	否	advancedInfo	取值如下： <ul style="list-style-type: none"> advancedInfo：获取作业plan的同时，也获取作业的SQL建议。 advisor：仅获取作业的SQL建议。 <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 5px;"> <p> 说明 仅Blink 3.7.0以上版本支持该参数。</p> </div>

返回数据

名称	类型	示例值	描述
RequestId	String	9B8DCCAB-6C1C-4195-8847-F17C3A36007E	请求ID。
SessionId	String	6sozg4hlznp8ylrgp6dpoxpisewil57ymps4e0ues7ikwt2nz12neu7w5u28dt2066o9dn6kzzhztqshgbb16uwosbq9g930ucmnzopviyzqgefhpypwtbly5r1wg4vtsd8k8ud4ukhui39ce	会话ID，需要传入checkrowplanjson接口中使用。

示例

请求示例

```
/api/v2/projects/project1/jobs/job1/planjson
```

正常返回示例

XML 格式

```
<RequestId>9B8DCCAB-6C1C-4195-8847-F17C3A36007E</RequestId>
<SessionId>6sozg4hlznp8ylrgp6dpoxpisewil57ymps4e0ues7ikwt2nz12neu7w5u28dt2066o9dn6kzzhztqshgbb16uwosbq9g930ucmnzopviyzqgefhpypwtbly5r1wg4vtsd8k8ud4ukhui39ce</SessionId>
```

JSON 格式

```
{
  "RequestId": "9B8DCCAB-6C1C-4195-8847-F17C3A36007E",
  "SessionId":
  "6sozg4hlznp8ylrgp6dpoxpisewil57ymps4e0ues7ikwt2nz12neu7w5u28dt2066o9dn6kzzhztqshgbb16uwosbq9g930ucmnzopviyzqgefhpypwtbly5r1wg4vtsd8k8ud4ukhui39ce"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.8. Package

8.8.1. CreatePackage

您可以通过CreatePackage OpenAPI获取您在OSS中创建的Package，并将该Package传送到作业的执行节点。

说明

- Package的版本信息由您自行维护，上传时可以选填MD5值，开发控制台协助校验。
- 共享集群用户无法使用CreatePackage。
- 独享集群用户只能使用创建集群时的OSS信息，填写时只需要填写ossPath，无需填写完整信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

POST /api/v2/projects/[projectName]/packages HTTP/1.1

请求参数

名称	类型	位置	是否必选	示例值	描述
ossBucket	String	Body	是	blinktest2.oss-cn-hangzhou-internal.aliyuncs.com	OSSBucket
ossEndpoint	String	Body	是	oss-cn-hangzhou-internal.aliyuncs.com	OSS接入点
ossOwner	String	Body	是	user1	OSS所有者
ossPath	String	Body	是	path1/path2/a.jar	Package在OSS中的路径
packageName	String	Body	是	package1.jar	Package名称
projectName	String	Path	是	project1	项目名称

名称	类型	位置	是否必选	示例值	描述
type	String	Body	是	JAR	Package类型： <ul style="list-style-type: none"> JAR: JAR包 DICTIONARY: 普通文件 SCRIPT: 脚本 PYTHON: Python文件或者ZIP包
originName	String	Body	否	12222	Package原名
description	String	Body	否	test	Package的备注描述
md5	String	Body	否	3F7468C153E52 9B141C326332 DF15D05	Package的MD5值
tag	String	Body	否	{"function": "group by time"}	Package的标记
RegionId	String	Header	否	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 5px;"> ? 说明 公共云用户请忽略此参数。 </div>

返回数据

名称	类型	示例值	描述
RequestId	String	F85A9751-638B-4186-B5E5-3F66D8A1CBFE	请求ID

示例

请求示例

```
/api/v2/projects/project1/packages
{"packageName": "package1.jar", "type": "JAR", "ossEndpoint": "oss-cn-hangzhou-internal.aliyuncs.com", "ossBucket": "blinktest2.oss-cn-hangzhou-internal.aliyuncs.com", "ossOwner": "user1", "ossPath": "path1/path2/a.jar"}
```

正常返回示例

XML 格式

```
<RequestId>F85A9751-638B-4186-B5E5-3F66D8A1CBFE</RequestId>
```

JSON 格式


```
{
  "RequestId": "F85A9751-638B-4186-B5E5-3F66D8A1CBFE"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.8.2. DeletePackage

您可以通过DeletePackage OpenAPI删除开发控制台上通过CreatePackage创建的Package。删除后开发控制台无法再获取对应的Package信息。

 **说明** 使用DeletePackage删除不存在的Package时，系统会报错。如果已上线的作业已引用该Package，删除该Package时也会报错。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/projects/[projectName]/packages/[packageName] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
packageName	String	是	package.jar	Package名称
projectName	String	是	project1	项目名称
RegionId	String	是	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	A604B9FA-D181-43D7-A064-5FF971B90466	请求ID

示例

请求示例

```
/api/v2/projects/project1/packages/package.jar
```

正常返回示例

XML 格式

```
<RequestId>A604B9FA-D181-43D7-A064-5FF971B90466</RequestId>
```

JSON 格式


```
{  
  "RequestId": "A604B9FA-D181-43D7-A064-5FF971B90466"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.8.3. GetPackage

您可以通过GetPackage OpenAPI获取Package详细信息。

 **说明** GetPackage只能获取到通过Web或CreatePackage接口在开发控制台上保存的Package相关信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/packages/[packageName] HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
packageName	String	Path	是	aliyun-java-sdk-foas-2.9.0.jar	Package名称。
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	是	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; display: inline-block;"> ? 说明 公共云用户请忽略此参数。 </div>

返回数据

名称	类型	示例值	描述
Package	Struct		Package详情。
CreateTime	Long	1600156643000	创建时间（13位时间戳，精确到毫秒）。
Creator	String	1709064687573327	创建者。
Description	String	adas	Package备注。
Md5	String	e0899e291f0524c6e6572c126599f73b	Package的Md5值。
Modifier	String	1709064687573327	最新修改者。
ModifyTime	Long	1600156643000	最新修改时间。
OriginName	String	aliyun-java-sdk-foas-2.9.0.jar	Package别名。
OssBucket	String	blinktest2.oss-cn-hangzhou-internal.aliyuncs.com	OSS Bucket。
OssEndpoint	String	oss-cn-hangzhou-internal.aliyuncs.com	OSS接入点。

名称	类型	示例值	描述
OssOwner	String	owner1	OSS所有者。
OssPath	String	79zec7svps7hca0xnogbqvpu/aliyun-java-sdk-foas-2.9.0.jar/l1jsfzlp4z4pk5rhutb91k.bayes	OSS路径。
PackageName	String	package1.jar	Package名称。
ProjectName	String	project1	项目名称。
ScanErrorMessage	String	未开始进行安全检查。	JAR包安全扫描信息。
ScanExtBizNo	String	5467**	安全扫描ID。
ScanLink	String	http://xxxxx	扫描详情URI。
ScanState	String	NOT_START	JAR包安全扫描状态。参数取值如下： <ul style="list-style-type: none"> • SKIP_SCAN：无需进行安全检查。 • NOT_START：未开始进行安全检查。 • EXECUTING：正在进行安全检查。 • PASS：安全检查通过。 • FAIL：安全检查未通过。 • EXCEPTION：安全检查出错。
Tag	String	{"function":"group by time"}	Package的标记。
Type	String	JAR	Package类型： <ul style="list-style-type: none"> • JAR：JAR包 • DICTIONARY：普通文件 • SCRIPT：脚本 • PYTHON：Python文件或者ZIP包
RequestId	String	7076AAD2-EF55-479A-867E-3245B9105E1D	请求ID。

示例

请求示例

```
/api/v2/projects/project1/packages/aliyun-java-sdk-foas-2.9.0.jar
```

正常返回示例

XML 格式

```
<RequestId>7076AAD2-EF55-479A-867E-3245B9105E1D</RequestId>
<Package>
  <ModifyTime>1600156643000</ModifyTime>
  <Description>adas</Description>
  <ProjectName>project1</ProjectName>
  <PackageName>aliyun-java-sdk-foas-2.9.0.jar</PackageName>
  <CreateTime>1600156643000</CreateTime>
  <Creator>1709064687573327</Creator>
  <OriginName>aliyun-java-sdk-foas-2.9.0.jar</OriginName>
  <Type>JAR</Type>
  <OssPath>79zec7svps7hca0xnogbqpvu/aliyun-java-sdk-foas-2.9.0.jar/l1jsjzlp4z4pk5rhutb91k.bayes</OssPath>
  <Modifier>1709064687573327</Modifier>
  <Md5>e0899e291f0524c6e6572c126599f73b</Md5>
</Package>
```

JSON 格式

```
{
  "RequestId": "7076AAD2-EF55-479A-867E-3245B9105E1D",
  "Package": {
    "ModifyTime": 1600156643000,
    "Description": "adas",
    "ProjectName": "project1",
    "PackageName": "aliyun-java-sdk-foas-2.9.0.jar",
    "CreateTime": 1600156643000,
    "Creator": "1709064687573327",
    "OriginName": "aliyun-java-sdk-foas-2.9.0.jar",
    "Type": "JAR",
    "OssPath": "79zec7svps7hca0xnogbqpvu/aliyun-java-sdk-foas-2.9.0.jar/l1jsjzlp4z4pk5rhutb91k.bayes",
    "Modifier": "1709064687573327",
    "Md5": "e0899e291f0524c6e6572c126599f73b"
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.8.4. GetRefPackageJob

您可以通过GetRefPackageJob OpenAPI获取指定Package的所有作业详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/packages/[packageName]/jobs HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
packageName	String	是	package1.jar	Package名称
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。
pageSize	Integer	否	50	包大小
pageIndex	Integer	否	123	索引

返回数据

名称	类型	示例值	描述
Jobs	Array of Job		作业详情
Job			
ApiType	String	DATASTREAM	API类型：DATASTREAM或SQL
ClusterId	String	et2cloud****	集群ID

名称	类型	示例值	描述
Code	String	blink.main.class=com.hjc.test.blink_test.ChinanTopSpeedWindowing2\r\nblink.job.name=datastream_prestest	作业代码
CreateTime	Long	1551161842000	作业创建时间
Creator	String	1709064687573327	作业创建者
Description	String	test	作业备注
EngineVersion	String	blink_2.2.4	引擎版本
FolderId	Long	10148	文件夹ID
IsCommitted	Boolean	true	是否上线
JobId	String	9t9m6jfw8rb44iexhbpv3tnz	作业ID
JobName	String	job1	作业名称
JobType	String	FLINK_STREAM	作业类型： <ul style="list-style-type: none"> FLINK_STREAM：流作业 FLINK_BATCH：批作业
Modifier	String	1709064687573327	最新修改者
ModifyTime	Long	1548397575000	最新修改时间
Packages	String	blink_test_datastream.jar	Package名称，多个Package请使用逗号（,）分隔。
		{\n \"nodes\": [\n {\n \"id\": 1,\n \"type\": \"Source: Custom Source\",\n \"pact\": \"Data Source\",\n \"contents\":	

名称	类型	示例值	描述
PlanJson	String	<pre> \Source: Custom Source\", \n \parallelism\": 1\n }, \n { \n \"id\": 2, \n \"type\": \"Timestamps/Wate rmarks\", \n \"pact\": \"Operator\", \n \"contents\": \"Timestamps/Wate rmarks\", \n \parallelism\": 1, \n \"predecessors\": [\n { \n \"id\": 1, \n \"ship_strategy\": \"FORWARD\", \n \"side\": \"second\" \n } \n] \n }, \n { \n \"id\": 4, \n \"type\": \"TriggerWindow(Gl obalWindows(), ListStateDescriptor{ serializer=org.apac he.flink.api.commo n.typeutils.base.List Serializer@b2107f6 7}, DeltaTrigger(com.hj c.test.blink_test.Chi nanTopSpeedWindo wing\$1@350aac89, 50.0), TimeEvictor(10000), WindowedStream.r educe(WindowedStr eam.java:258))\", \n \"pact\": \"Operator\", \n \"contents\": \"TriggerWindow(Gl obalWindows(), ListStateDescriptor{ serializer=org.apac he.flink.api.commo n.typeutils.base.List Serializer@b2107f6 7}, DeltaTrigger(com.hj c.test.blink_test.Chi nanTopSpeedWindo wing\$1@350aac89, 50.0), TimeEvictor(10000), </pre>	作业上线的执行计划

名称	类型	示例值	描述
RequestId	String	31FDAE27-F135-4DAB-9C75-FADD6B0C4D1D	请求ID
TotalCount	Long	1	作业总数
TotalPage	Integer	1	页码总数

示例

请求示例

```
/api/v2/projects/project1/packages/package1.jar/jobs
```

正常返回示例

XML 格式

```
<TotalCount>1</TotalCount>
<TotalPage>1</TotalPage>
<PageSize>10</PageSize>
<RequestId>31FDAE27-F135-4DAB-9C75-FADD6B0C4D1D</RequestId>
<Jobs>
  <Job>
    <ModifyTime>1583722910000</ModifyTime>
    <Description/>
    <EngineVersion>blink-3.4.4</EngineVersion>
    <ProjectName>bayes_team</ProjectName>
    <ClusterId>et2cloud****</ClusterId>
    <PlanJson>{
      "nodes": [
        {
          "id": 1,
          "type": "Source: Custom Source",
          "pact": "Data Source",
          "contents": "Source: Custom Source",
          "parallelism": 1
        },
        {
          "id": 2,
          "type": "Timestamps/Watermarks",
          "pact": "Operator",
          "contents": "Timestamps/Watermarks",
          "parallelism": 1,
          "predecessors": [
            {
              "id": 1,
              "ship_strategy": "FORWARD",
              "side": "second"
            }
          ]
        }
      ]
    }
  </Job>
</Jobs>
```

```

    ]
  },
  {
    "id": 4,
    "type": "TriggerWindow(GlobalWindows(), ListStateDescriptor{serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@b2107f67}, DeltaTrigger(com.hjc.test.blink_test.ChinanTopSpeedWindowing$1@350aac89, 50.0), TimeEvictor(10000), WindowedStream.reduce(WindowedStream.java:258))",
    "pact": "Operator",
    "contents": "TriggerWindow(GlobalWindows(), ListStateDescriptor{serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@b2107f67}, DeltaTrigger(com.hjc.test.blink_test.ChinanTopSpeedWindowing$1@350aac89, 50.0), TimeEvictor(10000), WindowedStream.reduce(WindowedStream.java:258))",
    "parallelism": 1,
    "predecessors": [
      {
        "id": 2,
        "ship_strategy": "KEY-GROUP",
        "side": "second"
      }
    ]
  }
],
{
  "id": 5,
  "type": "Sink: Unnamed",
  "pact": "Data Sink",
  "contents": "Sink: Unnamed",
  "parallelism": 1,
  "predecessors": [
    {
      "id": 4,
      "ship_strategy": "FORWARD",
      "side": "second"
    }
  ]
}
]
}</PlanJson>
<CreateTime>1551161842000</CreateTime>
<JobName>job1</JobName>
<Creator>1709064687573327</Creator>
<FolderId>10148</FolderId>
<Properties>#checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE
#blink.checkpoint.mode=EXACTLY_ONCE
#checkpoint间隔时间, 单位毫秒
#blink.checkpoint.interval.ms=180000
#rocksdb的数据生命周期, 单位毫秒
#state.backend.rocksdb.ttl.ms=129600000
</Properties>
<Code>blink.main.class=com.hjc.test.blink_test.ChinanTopSpeedWindowing2
blink.job.name=datastream_prestest</Code>
<JobType>FLINK_STREAM</JobType>
<Packages>blink_test_datastream.jar</Packages>
<ApiType>DATASTREAM</ApiType>
<IsCommitted>>true</IsCommitted>
<Modifier>1709064687573327</Modifier>

```

```
<QueueName>root.bayes_team</QueueName>  
<JobId>9t9m6jfw8rb44iexbpv3tnz</JobId>  
</Job>  
</Jobs>  
<PageIndex>1</PageIndex>
```

JSON 格式

```

{
  "TotalCount": 1,
  "TotalPage": 1,
  "PageSize": 10,
  "RequestId": "31FDAE27-F135-4DAB-9C75-FADD6B0C4D1D",
  "Jobs": {
    "Job": [
      {
        "ModifyTime": 1583722910000,
        "Description": "",
        "EngineVersion": "blink-3.4.4",
        "ProjectName": "bayes_team",
        "ClusterId": "et2cloud****",
        "PlanJson": "{\n  \"nodes\": [\n    {\n      \"id\": 1,\n      \"type\": \"Source: Custom Source\",\n      \"pact\": \"Data Source\",\n      \"contents\": \"Source: Custom Source\",\n      \"parallelism\": 1\n    },\n    {\n      \"id\": 2,\n      \"type\": \"Timestamps/Watermarks\",\n      \"pact\": \"Operator\",\n      \"contents\": \"Timestamps/Watermarks\",\n      \"parallelism\": 1,\n      \"predecessors\": [\n        {\n          \"id\": 1,\n          \"ship_strategy\": \"FORWARD\",\n          \"side\": \"second\"\n        }\n      ],\n        {\n          \"id\": 4,\n          \"type\": \"TriggerWindow(GlobalWindows(), ListStateDescriptor{serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@b2107f67}, DeltaTrigger(com.hjc.test.blink_test.ChinanTopSpeedWindowing$1@350aac89, 50.0), TimeEvictor(10000), WindowedStream.reduce(WindowedStream.java:258))\",\n          \"pact\": \"Operator\",\n          \"contents\": \"TriggerWindow(GlobalWindows(), ListStateDescriptor{serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@b2107f67}, DeltaTrigger(com.hjc.test.blink_test.ChinanTopSpeedWindowing$1@350aac89, 50.0), TimeEvictor(10000), WindowedStream.reduce(WindowedStream.java:258))\",\n          \"parallelism\": 1,\n          \"predecessors\": [\n            {\n              \"id\": 2,\n              \"ship_strategy\": \"KEY-GROUP\",\n              \"side\": \"second\"\n            }\n          ],\n            {\n              \"id\": 5,\n              \"type\": \"Sink: Unnamed\",\n              \"pact\": \"Data Sink\",\n              \"contents\": \"Sink: Unnamed\",\n              \"parallelism\": 1,\n              \"predecessors\": [\n                {\n                  \"id\": 4,\n                  \"ship_strategy\": \"FORWARD\",\n                  \"side\": \"second\"\n                }\n              ]\n            }\n          ]\n        }\n      ]\n    }",
        "CreateTime": 1551161842000,
        "JobName": "job1",
        "Creator": "1709064687573327",
        "FolderId": 10148,
        "Properties": "#checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE\n#blink.checkpoint.mode=EXACTLY_ONCE\n\n#checkpoint间隔时间, 单位毫秒\n#blink.checkpoint.interval.ms=180000\n\n#rocksdb的数据生命周期, 单位毫秒\n#state.backend.rocksdb.ttl.ms=129600000",
        "Code": "blink.main.class=com.hjc.test.blink_test.ChinanTopSpeedWindowing2\r\n\nblink.job.name=datastream_prestest",
        "JobType": "FLINK_STREAM",
        "Packages": "blink_test_datastream.jar",
        "ApiType": "DATASTREAM",
        "IsCommitted": true,
        "Modifier": "1709064687573327",
        "QueueName": "root.bayes_team",
        "JobId": "9t9m6jfw8rb44iexhbpv3tnz"
      }
    ]
  },
  "PageIndex": 1
}

```

错误码

访问[错误中心](#)查看更多错误码。

8.8.5. ListPackage

您可以通过ListPackage OpenAPI获取指定Project下满足特定条件的Package信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/packages HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
projectName	String	是	project1	项目名称
packageName	String	否	package1.jar	Package名称
type	String	否	JAR	Package类型： <ul style="list-style-type: none">JAR: JAR包DICTIONARY: 普通文件SCRIPT: 脚本PYTHON: Python文件或者ZIP包
pageSize	Integer	否	10	每页包含Package数量
pageIndex	Integer	否	1	索引
tag	String	否	{"function": "group by time"}	Package的标记
RegionId	String	否	cn-hangzhou	区域ID <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px; margin-top: 5px;">? 说明 公共云用户请忽略此参数。</div>

返回数据

名称	类型	示例值	描述
Packages	Array of Package		Package详情
Package			
CreateTime	Long	1600156643000	创建时间（13位时间戳，精确到毫秒）
Creator	String	1709064687573327	创建者
Description	String	adas	Package备注
Md5	String	e0899e291f0524c6e6572c126599f73b	Package的Md5值
Modifier	String	1709064687573327	最新修改者
ModifyTime	Long	1600156643000	最新修改时间
OriginName	String	aliyun-java-sdk-foas-2.9.0.jar	Package别名
OssBucket	String	79zec7svps7hca0xnogbqpvu/aliyun-java-sdk-foas-2.9.0.jar/l1jsjfzlp4z4pk5rhutb91k.bayes	OssBucket
OssEndpoint	String	oss-cn-hangzhou-internal.aliyuncs.com	Oss接入点
OssOwner	String	owner1	Oss拥有者
OssPath	String	path1/path2/a.jar	Oss路径
PackageName	String	aliyun-java-sdk-foas-2.9.0.jar	Package名称
ProjectName	String	project1	项目名称
Tag	String	{"function": "group by time"}	Package的标记

名称	类型	示例值	描述
Type	String	JAR	Package 类型： <ul style="list-style-type: none"> JAR: JAR包 DICTIONARY: 普通文件 SCRIPT: 脚本 PYTHON: Python文件或者ZIP包
PageIndex	Integer	1	第几页
PageSize	Integer	10	每页包含Package数量
RequestId	String	F33B93C0-56E6-43BD-B30F-E1FF1D4EF1EE	请求ID
TotalCount	Long	4	Package总数
TotalPage	Integer	1	总页面数

示例

请求示例

```
/api/v2/projects/project1/packages
```

正常返回示例

XML 格式

```
<TotalCount>4</TotalCount>
<TotalPage>1</TotalPage>
<PageSize>10</PageSize>
<RequestId>F33B93C0-56E6-43BD-B30F-E1FF1D4EF1EE</RequestId>
<Packages>
  <Package>
    <ModifyTime>1600156643000</ModifyTime>
    <OriginName>aliyun-java-sdk-foas-2.9.0.jar</OriginName>
    <Type>JAR</Type>
    <Description>adas</Description>
    <ProjectName>project1</ProjectName>
    <PackageName>aliyun-java-sdk-foas-2.9.0.jar</PackageName>
    <CreateTime>1600156643000</CreateTime>
    <OssPath>79zec7svps7hca0xnogbqpvu/aliyun-java-sdk-foas-2.9.0.jar/l1jsjflps4z4pk5rhutb91k.bayes
  </OssPath>
    <Creator>1709064687573327</Creator>
    <Modifier>1709064687573327</Modifier>
    <Md5>e0899e291f0524c6e6572c126599f73b</Md5>
  </Package>
</Packages>
```

```

</Package>
<Package>
  <ModifyTime>1596436948000</ModifyTime>
  <OriginName>blink-udx-2.x-1.0-snapshot.jar</OriginName>
  <Type>JAR</Type>
  <Description>123</Description>
  <ProjectName>bayes_team</ProjectName>
  <PackageName>blink-udx-2.x-1.0-snapshot.jar</PackageName>
  <CreateTime>1596436947000</CreateTime>
  <OssPath>79zec7svps7hca0xnogbqpvu/blink-udx-2.x-1.0-snapshot.jar/dr6r901verct65f612ca4t0c.bayes</OssPath>
  <Creator>1709064687573327</Creator>
  <Modifier>1709064687573327</Modifier>
  <Md5>70bf694b7c737ee608f1aaf11b15816a</Md5>
</Package>
<Package>
  <ModifyTime>1551161104000</ModifyTime>
  <OriginName>blink_test_datastream.jar</OriginName>
  <Type>JAR</Type>
  <Description>1</Description>
  <ProjectName>bayes_team</ProjectName>
  <PackageName>blink_test_datastream.jar</PackageName>
  <CreateTime>1551161103000</CreateTime>
  <OssPath>79zec7svps7hca0xnogbqpvu/blink_test_datastream.jar/mg5174rvvsv472xrkvv23wn3.bayes</OssPath>
  <Creator>1709064687573327</Creator>
  <Modifier>1709064687573327</Modifier>
  <Md5>7fd9782e4aad6e4b6a9aed8521774d40</Md5>
</Package>
<Package>
  <ModifyTime>1546094734000</ModifyTime>
  <OriginName>sql_udx-3.0</OriginName>
  <Type>JAR</Type>
  <Description>1</Description>
  <ProjectName>bayes_team</ProjectName>
  <PackageName>sql_udx-3.0</PackageName>
  <CreateTime>1546094733000</CreateTime>
  <OssPath>79zec7svps7hca0xnogbqpvu/sql_udx-3.0/6za6dxizvsscdjpupnbp0cfq.bayes</OssPath>
  <Creator>1709064687573327</Creator>
  <Modifier>1709064687573327</Modifier>
  <Md5>d8e19321c1b33af2e52643ea169d4f7d</Md5>
</Package>
</Packages>
<PageIndex>1</PageIndex>

```

JSON 格式

```

{
  "TotalCount": 4,
  "TotalPage": 1,
  "PageSize": 10,
  "RequestId": "F33B93C0-56E6-43BD-B30F-E1FF1D4EF1EE",
  "Packages": {
    "Package": [

```



```
Package : {
  "ModifyTime": 1600156643000,
  "OriginName": "aliyun-java-sdk-foas-2.9.0.jar",
  "Type": "JAR",
  "Description": "adas",
  "ProjectName": "project1",
  "PackageName": "aliyun-java-sdk-foas-2.9.0.jar",
  "CreateTime": 1600156643000,
  "OssPath": "79zec7svps7hca0xnogbqpvu/aliyun-java-sdk-foas-2.9.0.jar/l1jsfzlp54z4pk5rhutb91k.bayes"
},
{
  "Creator": "1709064687573327",
  "Modifier": "1709064687573327",
  "Md5": "e0899e291f0524c6e6572c126599f73b"
},
{
  "ModifyTime": 1596436948000,
  "OriginName": "blink-udx-2.x-1.0-snapshot.jar",
  "Type": "JAR",
  "Description": "123",
  "ProjectName": "bayes_team",
  "PackageName": "blink-udx-2.x-1.0-snapshot.jar",
  "CreateTime": 1596436947000,
  "OssPath": "79zec7svps7hca0xnogbqpvu/blink-udx-2.x-1.0-snapshot.jar/dr6r901verct65f612ca4t0c.bayes",
  "Creator": "1709064687573327",
  "Modifier": "1709064687573327",
  "Md5": "70bf694b7c737ee608f1aaf11b15816a"
},
{
  "ModifyTime": 1551161104000,
  "OriginName": "blink_test_datastream.jar",
  "Type": "JAR",
  "Description": "1",
  "ProjectName": "bayes_team",
  "PackageName": "blink_test_datastream.jar",
  "CreateTime": 1551161103000,
  "OssPath": "79zec7svps7hca0xnogbqpvu/blink_test_datastream.jar/mg5174rvvsv472xrkvv23wn3.bayes",
  "Creator": "1709064687573327",
  "Modifier": "1709064687573327",
  "Md5": "7fd9782e4aad6e4b6a9aed8521774d40"
},
{
  "ModifyTime": 1546094734000,
  "OriginName": "sql_udx-3.0",
  "Type": "JAR",
  "Description": "1",
  "ProjectName": "bayes_team",
  "PackageName": "sql_udx-3.0",
  "CreateTime": 1546094733000,
  "OssPath": "79zec7svps7hca0xnogbqpvu/sql_udx-3.0/6za6dxizvsscdjpupnbp0cfq.bayes",
  "Creator": "1709064687573327",
  "Modifier": "1709064687573327",
  "Md5": "d8e19321c1b33af2e52643ea169d4f7d"
}
}
```

```
{
  },
  "PageIndex": 1
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.8.6. UpdatePackage

您可以通过UpdatePackage OpenAPI更新开发控制台上保存在指定项目下的Package。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/projects/[projectName]/packages/[packageName] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
packageName	String	是	package1.jar	Package名称
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。
originName	String	否	package2.jar	Package别名
description	String	否	test	Package备注描述
md5	String	否	3F7468C153E529 B141C326332DF1 5D05	Package的Md5值

名称	类型	是否必选	示例值	描述
ossEndpoint	String	否	oss-cn-hangzhou-internal.aliyuncs.com	OSS接入点
ossBucket	String	否	blinktest2.oss-cn-hangzhou-internal.aliyuncs.com	OSS Bucket
ossOwner	String	否	owner1	OSS所有者
ossPath	String	否	path1/path2/a.jar	OSS路径
tag	String	否	{"function": "group by time"}	Package的标记

返回数据

名称	类型	示例值	描述
RequestId	String	77D5CB98-5E9F-42BB-8AA9-001125B48E28	请求ID

示例

请求示例

```
/api/v2/projects/project1/packages/package1.jar
```

正常返回示例

XML 格式

```
<RequestId>77D5CB98-5E9F-42BB-8AA9-001125B48E28</RequestId>
```

JSON 格式

```
{  
  "RequestId": "77D5CB98-5E9F-42BB-8AA9-001125B48E28"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9. Queue

8.9.1. CreateQueue

您可以使用CreateQueue将集群资源分配到创建的Queue中，并绑定Queue中的项目。Queue创建成功后即开始消耗资源。

 说明 CreateQueue仅限独享模式使用。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

POST /api/v2/clusters/[clusterId]/queue HTTP/1.1

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Path	是	cmy99ugusuco66x9qc6k****	集群ID。  说明 您可以使用listcluster获取集群ID信息。
maxMemMB	Integer	Body	是	16	Queue的最大内存，单位为MB。
maxVcore	Integer	Body	是	4	Queue的最大CPU个数。  说明 100Vcore = 1CU = 4G MEM。
queueName	String	Body	是	root.default	Queue的名称。
RegionId	String	Header	否	cn-shanghai	地域ID。

名称	类型	位置	是否必选	示例值	描述
gpu	Integer	Body	否	1	GPU个数。

返回数据

名称	类型	示例值	描述
RequestId	String	9F1CAD8D-E80E-45AF-82D7-8314FE17A34A	请求ID

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/queue
```

正常返回示例

XML 格式

```
<RequestId>9F1CAD8D-E80E-45AF-82D7-8314FE17A34A</RequestId>
```

JSON 格式

```
{  
  "RequestId": "9F1CAD8D-E80E-45AF-82D7-8314FE17A34A"  
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.2. DeleteQueue

您可以通过DeleteQueue OpenAPI删除已存在的Queue。如果需要删除的Queue已被项目绑定，则需要先解绑项目，再进行删除。

 说明 DeleteQueue仅适用于独享模式。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/clusters/{clusterId}/queue HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
clusterId	String	是	d6wxwo5tnrmuamx2ly3m****	集群ID ? 说明 您可以使用listcluster获取集群ID信息。
queueName	String	是	root.default	Queue名称 ? 说明 您可以使用GetClusterQueueInfo获取Queue名称信息。
RegionId	String	是	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/clusters/d6wxwo5tnrmuamx2ly3m****/queue
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{"requestId":"C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.3. BindQueue

您可以通过BindQueue OpenAPI将Project绑定在已经存在但没有被其他Project绑定的Queue上，绑定成功后，系统才会给您的Project分配资源，您才能在该项目上创建和运行作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/projects/[projectName]/queue HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
clusterId	String	是	d6wxwo5tnrmua mx2ly3m****	集群ID  说明 您可以使用 listcluster 获取集群ID信息。
projectName	String	是	project1	项目名称
queueName	String	是	queue1	要绑定的Queue名称  说明 您可以使用 GetClusterQueueInfo 获取Queue名称信息。

名称	类型	是否必选	示例值	描述
RegionId	String	否	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/projects/project1/queue  
{\"clusterId\": \"d6wxwo5tnrmuamx2ly3m****\", \"queueName\": \"queue1\"}
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{\"requestId\": \"C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2\"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.4. UnbindQueue

您可以通过UnbindQueue OpenAPI将Queue中已经绑定的项目进行解绑。执行UnbindQueue操作前，需要将项目上的作业全部下线。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/projects/[projectName]/queue HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
clusterId	String	是	cmy99ugusuco66x9qc6k****	集群ID  说明 您可以使用 <code>listcluster</code> 获取集群ID信息。
projectName	String	是	project1	项目名称
queueName	String	是	queue1	要绑定的Queue名称  说明 您可以使用 <code>GetClusterQueueInfo</code> 获取Queue名称信息。
RegionId	String	否	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/projects/project1/queue
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{"requestId":"C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.5. GetClusterQueueInfo

您可以通过GetClusterQueueInfo查询集群上已存在的Queue的信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/clusters/[clusterId]/queueinfo HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Path	是	h6272cj4etgqe7oets4s****	集群ID。 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff; margin-top: 5px;">说明 您可以使用listcluster获取集群ID信息。</div>
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
Queues	Array of Queue		返回Queues详情。
Queue			

名称	类型	示例值	描述
ClusterId	String	h6272cj4etgqe7oets4s****	集群ID。  说明 您可以使用 <code>listcluster</code> 获取集群ID信息。
ExternalInfo	String	<code>{"minGpu\":0,\"maxGpu\":0,\"name\": \"root.cloudpay_dev\", \"minMem\":24535,\"usedVCore\":50,\"usedGpu\":0,\"usedMem\":2816,\"minVCore\":599,\"maxVCore\":600,\"maxMem\":24576}</code>	未转换为INTEGER时的原始信息。  说明 <ul style="list-style-type: none"> 仅在 aliyun-java-sdk-foas 2.7.0 及以上版本返回ExternalInfo。 Integer类型返回-9999时，表示该数值超过Integer能表示的大小，请从ExternalInfo中取值。
MaxGpu	Integer	0	最大GPU。
MaxMem	Integer	24576	内存最大值（MB）。
MaxVCore	Integer	599	最大Vcore数。
MinGpu	Integer	0	最小GPU。
MinMem	Integer	24535	最小内存数（MB）。
MinVCore	Integer	599	最小Vcore数。
QueueName	String	root.cloudpay_dev	队列名称。
UsedGpu	Integer	0	已使用GPU。
UsedMem	Integer	2816	已使用内存（MB）。
UsedVCore	Integer	50	已使用Vcore。
RequestId	String	DACD3B42-5977-4293-B5A8-197A33A954BB	请求ID。

示例

请求示例

```
/api/v2/clusters/h6272cj4etgqe7oets4s****/queueinfo
```

正常返回示例

XML 格式

```
<RequestId>DACD3B42-5977-4293-B5A8-197A33A954BB</RequestId>
<Queues>
  <Queue>
    <MaxMem>24576</MaxMem>
    <UsedGpu>0</UsedGpu>
    <ExternalInfo>{"minGpu":0,"maxGpu":0,"name":"root.cloudpay_dev","minMem":24535,"usedVCore":50,"usedGpu":0,"usedMem":2816,"minVCore":599,"maxVCore":600,"maxMem":24576}</ExternalInfo>
    <UsedMem>2816</UsedMem>
    <ClusterId>h6272cj4etgqe7oets4s****</ClusterId>
    <MinGpu>0</MinGpu>
    <MaxVCore>600</MaxVCore>
    <MinVCore>599</MinVCore>
    <MaxGpu>0</MaxGpu>
    <MinMem>24535</MinMem>
    <QueueName>root.cloudpay_dev</QueueName>
    <UsedVCore>50</UsedVCore>
  </Queue>
</Queues>
```

JSON 格式

```
{
  "RequestId": "DACD3B42-5977-4293-B5A8-197A33A954BB",
  "Queues": {
    "Queue": [
      {
        "MaxMem": 24576,
        "UsedGpu": 0,
        "ExternalInfo": "{\"minGpu\":0,\"maxGpu\":0,\"name\":\"root.cloudpay_dev\",\"minMem\":24535,\"usedVCore\":50,\"usedGpu\":0,\"usedMem\":2816,\"minVCore\":599,\"maxVCore\":600,\"maxMem\":24576}",
        "UsedMem": 2816,
        "ClusterId": "h6272cj4etgqe7oets4s****",
        "MinGpu": 0,
        "MaxVCore": 600,
        "MinVCore": 599,
        "MaxGpu": 0,
        "MinMem": 24535,
        "QueueName": "root.cloudpay_dev",
        "UsedVCore": 50
      }
    ]
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.6. ListProjectBindQueue

您可以通过ListProjectBindQueue OpenAPI查看指定项目对应的队列相关的信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/queues HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
projectName	String	是	project1	项目名称
RegionId	String	否	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。
clusterId	String	否	d6wxwo5tnrmua mx2ly3m****	集群ID ? 说明 您可以使用listcluster获取集群ID信息。
queueName	String	否	root.project1	队列名称 ? 说明 您可以使用GetClusterQueueInfo获取Queue名称信息。

返回数据

名称	类型	示例值	描述
Queues	Array of Queue		队列详情
Queue			
ClusterId	String	d6wxwo5tnrmuamx2ly3m****	集群ID
QueueName	String	root.project1	队列名称
RequestId	String	36277E3E-99BA-43C7-8F91-ADCC8FE1EC96	请求ID

示例

请求示例

```
/api/v2/projects/project1/queues
```

正常返回示例

XML 格式

```
<RequestId>36277E3E-99BA-43C7-8F91-ADCC8FE1EC96</RequestId>
<Queues>
  <Queue>
    <ClusterId>d6wxwo5tnrmuamx2ly3m****</ClusterId>
    <QueueName>root.project1</QueueName>
  </Queue>
</Queues>
```

JSON 格式

```
{
  "RequestId": "36277E3E-99BA-43C7-8F91-ADCC8FE1EC96",
  "Queues": {
    "Queue": [
      {
        "ClusterId": "d6wxwo5tnrmuamx2ly3m****",
        "QueueName": "root.project1"
      }
    ]
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.7. ListProjectBindQueueResource

您可以通过ListProjectBindQueueResource OpenAPI查询Project关联的队列详情。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects/[projectName]/queueresource HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
projectName	String	是	project1	项目名称
clusterId	String	否	d6wxwo5tnrmua mx2ly3m****	集群ID  说明 您可以使用listcluster获取集群ID信息。
queueName	String	否	root.project1	队列名称  说明 您可以使用GetClusterQueueInfo获取Queue名称信息。
RegionId	String	否	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
Queues	Array of Queue		队列详情

名称	类型	示例值	描述
Queue			
ClusterId	String	d6wxwo5tnrmuamx2ly3m****	集群ID
MaxGpu	Integer	0	GPU最大值
MaxMem	Integer	24576	内存最大值（MB）
MaxVCore	Integer	600	最大Vcore数
MinGpu	Integer	0	最小GPU数
MinMem	Integer	24535	最小内存数（MB）
MinVCore	Integer	599	最小Vcore数
QueueName	String	root.project1	队列名称
UsedGpu	Integer	0	已使用GPU
UsedMem	Integer	10496	已使用内存（MB）
UsedVCore	Integer	250	已使用VCore
RequestId	String	2127A47D-CB91-44AE-8277-3FC645A761CE	请求ID

示例

请求示例

```
/api/v2/projects/project1/queueresource
```

正常返回示例

XML 格式


```
<RequestId>2127A47D-CB91-44AE-8277-3FC645A761CE</RequestId>
<Queues>
  <Queue>
    <MaxMem>24576</MaxMem>
    <UsedGpu>0</UsedGpu>
    <UsedMem>10496</UsedMem>
    <ClusterId>d6wxwo5tnrmuamx2ly3m****</ClusterId>
    <MinGpu>0</MinGpu>
    <MaxVCore>600</MaxVCore>
    <MinVCore>599</MinVCore>
    <MaxGpu>0</MaxGpu>
    <MinMem>24535</MinMem>
    <QueueName>root.project1</QueueName>
    <UsedVCore>250</UsedVCore>
  </Queue>
</Queues>
```

JSON 格式

```
{
  "RequestId": "2127A47D-CB91-44AE-8277-3FC645A761CE",
  "Queues": {
    "Queue": [
      {
        "MaxMem": 24576,
        "UsedGpu": 0,
        "UsedMem": 10496,
        "ClusterId": "d6wxwo5tnrmuamx2ly3m****",
        "MinGpu": 0,
        "MaxVCore": 600,
        "MinVCore": 599,
        "MaxGpu": 0,
        "MinMem": 24535,
        "QueueName": "root.project1",
        "UsedVCore": 250
      }
    ]
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.9.8. UpdateQueue

您可以通过UpdateQueue修改指定Queue的某些参数，例如maxVcore、maxMemMB等。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/clusters/[clusterId]/queue HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Path	是	h6272cj4etgqe7oe****	集群ID。  说明 您可以使用 <code>listcluster</code> 获取集群ID信息。
maxMemMB	Integer	Body	是	16	Queue的最大内存，单位为MB。
maxVcore	Integer	Body	是	4	Queue的最大CPU个数。  说明 100Vcore=1core。
queueName	String	Body	是	root.cloudpay_dev	Queue的名称。
RegionId	String	Header	否	cn-hangzhou	地域。
gpu	Integer	Body	否	1	GPU个数。

返回数据

名称	类型	示例值	描述
RequestId	String	9F1CAD8D-E80E-45AF-82D7-8314FE17A34A	请求ID。

示例

请求示例

```
/api/v2/clusters/h6272cj4etgqe7oe****/queue
```

正常返回示例

XML 格式

```
<RequestId>9F1CAD8D-E80E-45AF-82D7-8314FE17A34A</RequestId>
```

JSON 格式

```
{  "RequestId": "9F1CAD8D-E80E-45AF-82D7-8314FE17A34A"}
```


错误码

访问[错误中心](#)查看更多错误码。

8.10. Cluster

8.10.1. CreateCluster

您可以在提交实时计算独享模式购买订单后，通过CreateCluster OpenAPI创建集群。

 **说明** 仅独享模式支持CreateCluster。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/clusters HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
description	String	Body	是	cluster1	集群描述。
displayName	String	Body	是	cluster_name	集群名称。
orderId	String	Body	是	blinkonecs_1234	购买订单生成的实例ID。

名称	类型	位置	是否必选	示例值	描述
userOssBucket	String	Body	是	12345	您OSS中Bucket名称。
userVpcId	String	Body	是	vpc-abcde	您的集群所在VPC名称。 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> <p>? 说明 实时计算和连通的上下游数据存储必须部署于同一个VPC，否则会造成数据无法连通的问题。</p> </div>
userVSwitch	String	Body	是	vsw-abcde	交换机名称。
zoneId	String	Body	是	cn-shanghai-f	可用区。
RegionId	String	Header	否	cn-hangzhou	区域。

返回数据

名称	类型	示例值	描述
ClusterId	String	pgr1rnrldry05mhsarcj****	集群ID。 <div style="border: 1px solid #ccc; padding: 5px; background-color: #e6f2ff;"> <p>? 说明 您可以使用 <code>listcluster</code> 获取集群ID信息。</p> </div>
RequestId	String	9F1CAD8D-E80E-45AF-82D7-8314FE17A34A	请求ID。

示例

请求示例

```
{
  "zoneId": "cn-shanghai-f",
  "displayName": "cluster_name",
  "description": "cluster1",
  "userOssBucket": "12345",
  "userVpcId": "vpc-abcde",
  "userVSwitch": "vsw-abcde",
  "orderId": "blinkonecs_1234"}

```

正常返回示例

XML 格式

```
<RequestId>9F1CAD8D-E80E-45AF-82D7-8314FE17A34A</RequestId>
<ClusterId>pgr1rnrldry05mhsarcj****</ClusterId>
```

JSON 格式

```
{
  "RequestId": "9F1CAD8D-E80E-45AF-82D7-8314FE17A34A",
  "ClusterId": "pgr1rnrldry05mhsarcj****"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.2. CreateCellClusterOrder

您可以通过CreateCellClusterOrder创建一个实时计算的实例，仅限独享集群。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/realtime-compute/cell/buy HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
masterNum	Integer	Body	是	3	<p>Master机器的数量，仅支持1台或者3台。</p> <div style="border: 1px solid #ccc; background-color: #e6f2ff; padding: 5px;"> <p>? 说明</p> <ul style="list-style-type: none"> 您需要3台master才能有效保障服务SLA。 选择该参数后不支持修改。 </div>

名称	类型	位置	是否必选	示例值	描述
masterSpec	String	Body	是	Ecs_4c16g	Master机器型号。
payModel	String	Body	是	pre	实例类型： <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> <p>? 说明</p> <ul style="list-style-type: none"> pre（包年包月）。 post（按量计费）。 </div>
region	String	Body	是	cn-hangzhou	地域。
slaveNum	Integer	Body	是	3	Slave机器数量，至少2台。
slaveSpec	String	Body	是	Ecs_8c32g	Slave机器型号。
period	Integer	Body	否	8	仅pre（包年包月）填写，表示购买时间，最多可以购买12个月，单位是月。
RegionId	String	Header	否	cn-shanghai	地域。

返回数据

名称	类型	示例值	描述
OrderId	String	blinkonecs_post-cn-6ja214s9y06	订单号。
RequestId	String	6676DF01-0533-4CB0-8EEB-74BDB1848A3B	请求ID。

示例

请求示例

```
/api/v2/realtime-compute/cell/buy
{"region":"cn-shanghai","masterSpec":"Ecs_4c16g","masterNum":3,"slaveSpec":"Ecs_8c32g",
"slaveNum":3,"payModel":"post"}
```

正常返回示例

XML 格式

```
<RequestId>6676DF01-0533-4CB0-8EEB-74BDB1848A3B</RequestId>
<OrderId>blinkonecs_post-cn-6ja214s9y06</OrderId>
```

JSON 格式

```
{
  "RequestId": "6676DF01-0533-4CB0-8EEB-74BDB1848A3B",
  "OrderId": "blinkonecs_post-cn-6ja214s9y06"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.3. DestroyCluster

您可以通过DestroyCluster OpenAPI注销集群，集群注销后不再产生费用。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/clusters/[clusterId] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
clusterId	String	是	cmy99ugusuco66x9qc6k****	集群ID  说明 您可以使用listcluster获取集群ID信息。
RegionId	String	是	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{"requestId":"C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.4. ExpandCluster

您可以通过ExpandCluster OpenAPI对已有集群进行Slave机型的扩容。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/clusters/[clusterId]/expand HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
----	----	------	-----	----

名称	类型	是否必选	示例值	描述
clusterId	String	是	cmy99ugusuco66x9qc6k****	集群ID <p>说明 您可以使用listcluster获取集群ID信息。</p>
RegionId	String	是	cn-hangzhou	区域ID <p>说明 公共云用户请忽略此参数。</p>
model	String	否	Ecs_4c16g	机器型号 <p>说明 扩容机型必须和已有Slave机型一致。</p>
count	Integer	否	5	扩容后的机器台数
userVSwitch	String	否	vsw-abcd	交换机名称 <p>说明 扩容提示vSwitch不足时，需要填写该参数。</p>

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/expand
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{"requestId":"C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.5. ShrinkCluster

您可以通过ShrinkCluster来减少集群Slave实例台数。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。


请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
PUT /api/v2/clusters/[clusterId]/shrink HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Path	是	cmy99ugusuco66x9qc6k****	集群ID。  说明 您可以使用 <code>listcluster</code> 获取集群ID信息。
instanceIds	String	Body	是	403936	集群Slave实例ID。指定多个实例需用逗号(,)分割。 您可以通过 <code>getClusterDetails</code> 接口查看已创建的集群的配置信息。 如果同时配置 <code>instanceIds</code> 和 <code>modelTargetCount</code> 参数时，以 <code>instanceIds</code> 为准。
RegionId	String	Header	否	cn-hangzhou	区域ID。

名称	类型	位置	是否必选	示例值	描述
modelTargetCount	String	Body	否	{Ecs_4c16g:2}	指定缩容实例的型号和台数。 ? 说明 缩容过程中随机去除Slave实例。

返回数据

名称	类型	示例值	描述
RequestId	String	5A0AF0E2-A715-40FD-8A2B-1EDBF4213489	请求ID。

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/shrink
{"instanceIds":["403936"]}
```

正常返回示例

XML 格式

```
<RequestId>5A0AF0E2-A715-40FD-8A2B-1EDBF4213489</RequestId>
```

JSON 格式

```
{
  "RequestId": "5A0AF0E2-A715-40FD-8A2B-1EDBF4213489"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.6. ListCluster

您可以通过ListCluster OpenAPI查询集群信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/clusters HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Query	否	rulpqd442sgxbtw9736r****	集群ID。
displayName	String	Query	否	demo	集群名称。
state	String	Query	否	RUNNING	集群状态： <ul style="list-style-type: none"> STARTING：集群创建中。 EXPANDING：集群扩容中（增加Slave机型台数）。 UPGRADING：集群变配中（提高或者降低Master）。 DESTROYING：集群注销中。 DESTROYED：集群已销毁。 REDUCING：集群缩容中（减少Slave机型台数）。 MAINTAINING：集群维护中。
region	String	Query	否	cn-beijing	区域。
pageSize	Integer	Query	否	10	每页显示的集群数。
pageIndex	Integer	Query	否	1	指定查询的页码。
RegionId	String	Header	否	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
Clusters	Array of Cluster		集群详情。
Cluster			

名称	类型	示例值	描述
ClusterId	String	rulpqd442sgxbtw9736r****	集群ID。
Description	String	demo	集群备注。
DisplayName	String	demo	集群名称。
GmtCreate	Long	1605609438000	集群VPC创建时间。
GmtModified	Long	1605610377000	集群VPC修改时间。
Operator	String	27395716024545****	集群最后操作者UID。
OwnerId	String	107992689699****	集群拥有者UID。
RegionId	String	cn-beijing	集群所属地区。
State	String	RUNNING	集群状态： <ul style="list-style-type: none"> STARTING：集群创建中。 EXPANDING：集群扩容中（增加Slave机型台数）。 UPGRADING：集群变配中（提高或者降低Master）。 DESTROYING：集群注销中。 DESTROYED：集群已销毁。 REDUCING：集群缩容中（减少Slave机型台数）。 MAINTAINING：集群维护中。
ZoneId	String	cn-beijing-e	区域ID。
PageIndex	Integer	1	指定查询的页码。
PageSize	Integer	10	每页显示的集群数。
RequestId	String	10C4F123-351D-4D2B-94A4-D6FF456AEA6E	请求ID。

名称	类型	示例值	描述
TotalCount	Long	3	总集群数。
TotalPage	Integer	10	总页数。

示例

请求示例

```
http(s)://[Endpoint]/?RegionId=cn-hangzhou&<公共请求参数>
```

正常返回示例

XML 格式

```
<TotalCount>3</TotalCount>
<TotalPage>1</TotalPage>
<PageSize>10</PageSize>
<RequestId>10C4F123-351D-4D2B-94A4-D6FF456AEA6E</RequestId>
<Clusters>
  <Cluster>
    <Operator>27395716024545****</Operator>
    <GmtCreate>1605609438000</GmtCreate>
    <Description>demo</Description>
    <ZonId>cn-beijing-e</ZonId>
    <OwnerId>107992689699****</OwnerId>
    <ClusterId>rulpqd442sgxbtw9736r****</ClusterId>
    <State>RUNNING</State>
    <DisplayName>demo</DisplayName>
    <GmtModified>1605610377000</GmtModified>
    <RegionId>cn-beijing</RegionId>
  </Cluster>
  <Cluster>
    <Operator>23996037709276****</Operator>
    <GmtCreate>1597654146000</GmtCreate>
    <Description>test</Description>
    <ZonId>cn-hangzhou-f</ZonId>
    <OwnerId>107992689699****</OwnerId>
    <ClusterId>rcmp9x37ztfb63g1x7lt****</ClusterId>
    <State>RUNNING</State>
    <DisplayName>niuna</DisplayName>
    <GmtModified>1601051659000</GmtModified>
    <RegionId>cn-hangzhou</RegionId>
  </Cluster>
</Clusters>
<PageIndex>1</PageIndex>
```

JSON 格式

```
{
  "TotalCount": 3,
  "TotalPage": 1,
  "PageSize": 10,
  "RequestId": "10C4F123-351D-4D2B-94A4-D6FF456AEA6E",
  "Clusters": {
    "Cluster": [
      {
        "Operator": "27395716024545****",
        "GmtCreate": 1605609438000,
        "Description": "demo",
        "Zoneld": "cn-beijing-e",
        "OwnerId": "107992689699****",
        "ClusterId": "rulpqd442sgxbtw9736r****",
        "State": "RUNNING",
        "DisplayName": "demo",
        "GmtModified": 1605610377000,
        "RegionId": "cn-beijing"
      },
      {
        "Operator": "23996037709276****",
        "GmtCreate": 1597654146000,
        "Description": "test",
        "Zoneld": "cn-hangzhou-f",
        "OwnerId": "107992689699****",
        "ClusterId": "rcmp9x37ztfb63g1x7lt****",
        "State": "RUNNING",
        "DisplayName": "niuna",
        "GmtModified": 1601051659000,
        "RegionId": "cn-hangzhou"
      }
    ]
  },
  "PageIndex": 1
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.7. GetClusterDetails

您可以通过GetClusterDetails查询已创建的集群的配置信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

GET /api/v2/clusters/[clusterId]/details HTTP/1.1

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Path	是	cmy99ugusuco66x9qc6k****	集群ID。  说明 您可以使用listcluster获取集群ID信息。
RegionId	String	Header	是	cn-hangzhou	区域ID。

返回数据

名称	类型	示例值	描述
Details	Struct		集群配置详情。
ClusterId	String	7cmedvslzs8q9n7pkfx5****	集群ID。
Description	String	test	集群描述。
DisplayName	String	test	集群名称。
GmtCreate	Long	1612693397000	集群VPC创建时间。
GmtModified	Long	1612693397000	集群VPC修改时间。
InstanceInfos	String	{k:v}	集群Master机型和Slave机型的相关信息，包括实例ID，ENI，IP和机器型号等。
IsMixDeploy	Boolean	false	是否为混部集群（存在多种型号Slave机型）。
Operator	String	137677185440****	集群最后操作人UID。
OwnerId	String	137677185440****	集群拥有者UID。

名称	类型	示例值	描述
RegionId	String	cn-shanghai	区域ID。
State	String	STARTING	<p>集群状态：</p> <ul style="list-style-type: none"> • 集群创建中：STARTING。 • 集群扩容中（增加Slave机型台数）：EXPANDING。 • 集群变配中（提高或者降低Master型号）：UPGRADING。 • 集群注销中：DESTROYING。 • 集群已销毁：DESTROYED。 • 集群缩容中（减少Slave机型台数）：REDUCING。 • 集群维护中：MAINTAINING。
StorageType	String	HDFS	存储类型。
UserOssInfo	String	<pre>{\"endpoint\": \"oss-cn-shanghai.aliyuncs.com\", \"vpcEndpoint\": \"oss-cn-shanghai-internal.aliyuncs.com\", \"ossEndpoint\": \"oss-cn-shanghai-internal.aliyuncs.com\", \"accessId\": null, \"accessKey\": null, \"bucket\": \"fblink-test-oss-beijing\"}</pre>	您的OSS信息。以JSON形式返回，包括Bucket, Endpoint等。
UserSGId	String	sg-abcd	安全组ID。
UserVSwitchList	String	[\"vsw-uf6v128q9khll8e9ucyhj\"]	您的VSW列表。
UserVpcId	String	vpc-uf6aq156h1gy06dzf****	VPC ID。
ZoneId	String	cn-shanghai-f	可用地域ID。

名称	类型	示例值	描述
RequestId	String	A35C7747-AE0B-4688-AD84-82CAAD512FD3	请求ID。

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/details
```

正常返回示例

XML 格式

```
<Details>
  <Operator>137677185440****</Operator>
  <UserOssInfo>{"endpoint":"oss-cn-shanghai.aliyuncs.com","vpcEndpoint":"oss-cn-shanghai-internal.aliyuncs.com","oxsEndpoint":"oss-cn-shanghai-internal.aliyuncs.com","accessId":null,"accessKey":null,"bucket":"flink-test-oss-beijing"}</UserOssInfo>
  <Description>test</Description>
  <StorageType>HDFS</StorageType>
  <ZoneId>cn-shanghai-f</ZoneId>
  <UserVSwitchList>["vsw-uf6v128q9khl8e9ucyhj"]</UserVSwitchList>
  <ClusterId>7cmedvslzs8q9n7pkfx5****</ClusterId>
  <IsMixDeploy>>false</IsMixDeploy>
  <GmtModified>1612693397000</GmtModified>
  <GmtCreate>1612693397000</GmtCreate>
  <OwnerId>137677185440****</OwnerId>
  <State>STARTING</State>
  <UserVpcId>vpc-uf6aq156h1gy06dzf****</UserVpcId>
  <DisplayName>test</DisplayName>
  <RegionId>cn-shanghai</RegionId>
</Details>
<RequestId>A35C7747-AE0B-4688-AD84-82CAAD512FD3</RequestId>
```

JSON 格式

```
{
  "Details": {
    "Operator": "137677185440****",
    "UserOssInfo": "{\"endpoint\":\"oss-cn-shanghai.aliyuncs.com\",\"vpcEndpoint\":\"oss-cn-shanghai-internal.aliyuncs.com\",\"oxsEndpoint\":\"oss-cn-shanghai-internal.aliyuncs.com\",\"accessId\":null,\"accessKey\":null,\"bucket\":\"flink-test-oss-beijing\"}",
    "Description": "test",
    "StorageType": "HDFS",
    "ZoneId": "cn-shanghai-f",
    "UserVSwitchList": "[\"vsw-uf6v128q9khl8e9ucyhj\"]",
    "ClusterId": "7cmedvslzs8q9n7pkfx5****",
    "IsMixDeploy": false,
    "GmtModified": 1612693397000,
    "GmtCreate": 1612693397000,
    "OwnerId": "137677185440****",
    "State": "STARTING",
    "UserVpcId": "vpc-uf6aq156h1gy06dzf****",
    "DisplayName": "test",
    "RegionId": "cn-shanghai"
  },
  "RequestId": "A35C7747-AE0B-4688-AD84-82CAAD512FD3"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.8. GetClusterEngineVersions

您可以通过GetClusterEngineVersions OpenAPI获取集群上可使用的引擎版本号。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/clusters/[clusterId]/engineversions HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
----	----	------	-----	----

名称	类型	是否必选	示例值	描述
clusterId	String	是	cmy99ugusuco66x9qc6k****	集群ID ? 说明 您可以使用listcluster获取集群ID信息。
RegionId	String	是	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
EngineVersions	List	current--blink-3.4.4	引擎版本
RequestId	String	7203295B-3726-4BEF-B3E1-9EE33BF2FECF	请求ID

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/engineversions
```

正常返回示例

XML 格式

```
<RequestId>7203295B-3726-4BEF-B3E1-9EE33BF2FECF</RequestId>
<EngineVersions>
  <EngineVersion>current--blink-3.4.4</EngineVersion>
</EngineVersions>
```

JSON 格式

```
{
  "RequestId": "7203295B-3726-4BEF-B3E1-9EE33BF2FECF",
  "EngineVersions": {
    "EngineVersion": [
      "current--blink-3.4.4"
    ]
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.9. GetClusterResource

您可以通过GetClusterResource OpenAPI查看集群总资源和当前已使用的资源情况。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/clusters/[clusterId]/resource HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
clusterId	String	是	cmy99ugusuco66x9qc6k****	集群ID ? 说明 您可以使用listcluster获取集群ID信息。
RegionId	String	是	cn-hangzhou	区域ID ? 说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	1F721BC4-6298-42BB-9289-794E7B0349B9	请求ID
Resource	Struct		资源详情
AllocatedMB	Long	10496	已使用的内存

名称	类型	示例值	描述
AllocatedVirtualCores	Long	250	已使用的VCore数
AvailableMB	Long	14080	可使用的内存
AvailableVirtualCores	Long	350	可使用的VCore数
TotalMB	Long	24576	集群总内存
TotalVirtualCores	Long	600	集群总VCore数

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/resource
```

正常返回示例

XML 格式

```
<RequestId>1F721BC4-6298-42BB-9289-794E7B0349B9</RequestId>
<Resource>
  <TotalVirtualCores>600</TotalVirtualCores>
  <AllocatedMB>10496</AllocatedMB>
  <AllocatedVirtualCores>250</AllocatedVirtualCores>
  <AvailableVirtualCores>350</AvailableVirtualCores>
  <TotalMB>24576</TotalMB>
  <AvailableMB>14080</AvailableMB>
</Resource>
```

JSON 格式


```
{
  "RequestId": "1F721BC4-6298-42BB-9289-794E7B0349B9",
  "Resource": {
    "TotalVirtualCores": 600,
    "AllocatedMB": 10496,
    "AllocatedVirtualCores": 250,
    "AvailableVirtualCores": 350,
    "TotalMB": 24576,
    "AvailableMB": 14080
  }
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.10. ModifyMasterSpec

您可以通过ModifyMasterSpec OpenAPI更改集群Master机型的规格，实现Master机型的升配和降配。

 **说明** ModifyMasterSpec仅支持在独享模式的按量付费模式下使用。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

PUT /api/v2/clusters/[clusterId]/specification HTTPS

请求参数

名称	类型	是否必选	示例值	描述
clusterId	String	是	cmy99ugusuco66x9qc6k****	集群ID  说明 您可以使用listcluster获取集群ID信息。
masterTargetModel	String	是	Ecs_4c16g	Master机型变配的目标机器型号
RegionId	String	否	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2	请求ID

示例

请求示例

```
/api/v2/clusters/cmy99ugusuco66x9qc6k****/specification
{"masterTargetModel":"Ecs_4c16g"}
```

正常返回示例

XML 格式

```
<requestId>C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2</requestId>
```

JSON 格式

```
{"requestId":"C5C6746D-2FFC-4D5D-A6A8-FA8DF7585DC2"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.10.11. GetClusterMetrics

您可以通过GetClusterMetrics获取集群的CPU、存储或YARN的资源分配状况等信息。仅独享模式支持GetClusterMetrics接口。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

该接口使用公共请求头，无特殊请求头。请参见公共请求参数文档。

请求语法

```
POST /api/v2/clusters/[clusterId]/metrics HTTPS
```

请求参数

名称	类型	位置	是否必选	示例值	描述
clusterId	String	Path	是	yourClusterID	集群ID。<note>不是集群的名称。</note>

名称	类型	位置	是否必选	示例值	描述
metricjson	String	Body	是	{ "end":1606291550061, "limit":"avg:sample:50", "queries":[{"aggregator":"avg", "downsample":"avg", "granularity":"60s", "metric":"ClusterID.system.cpu.user"}], "start":1606291390000}	查询Metric提交的JSON信息。
RegionId	String	Header	是	cn-hangzhou	地域ID

返回数据

名称	类型	示例值	描述
Metrics	Array of Metric		Metric信息。
Metric			
Dps	Map	k:v	时间点和对应的Metric值。
MetricName	String	delay	Metric名称。
Summary	Float	10.2	聚合后的Metric值。
Tags	Map	k:v	Metric标记。
RequestId	String	FD0FF8C0-779A-45EB-9674-FF3E127B10D2	请求ID。

示例

请求示例

```
POST /api/v2/clusters/[clusterId]/metrics HTTP/1.1
RegionId: cn-hangzhou
公共请求头
{
  "clusterId": "<yourClusterID>"
  "metricJson": "{metric: "clusterid.system.mem.memtotal", aggregator: "avg", downsample: "avg"}"
}
```

正常返回示例

XML 格式

```
<RequestId>A017B343-33DC-4A97-BAC5-4B57872D5B61</RequestId>
<Metrics>
  <metricName>et2hunbu.yarn.resourcemanager.jvmmetrics.MemHeapUsed</metricName>
  <dps>
    <1574244620>3675.839599609375</1574244620>
    <1574244640>3817.712646484375</1574244640>
    <1574244660>3864.84130859375</1574244660>
    <1574244680>3973.963134765625</1574244680>
    <1574244700>4121.57958984375</1574244700>
    <1574244720>4170.6123046875</1574244720>
    <1574244740>4279.62109375</1574244740>
    <1574244760>4427.05615234375</1574244760>
    <1574244780>4472.11474609375</1574244780>
  </dps>
  <summary>6700.7467978583445</summary>
  <tags/>
</Metrics>
```

JSON 格式

```
{
  "RequestId": "A017B343-33DC-4A97-BAC5-4B57872D5B61",
  "Metrics": [
    {
      "metricName": "et2hunbu.yarn.resourceanager.jvmmetrics.MemHeapUsed",
      "dps": {
        "1574244620": 3675.839599609375,
        "1574244640": 3817.712646484375,
        "1574244660": 3864.84130859375,
        "1574244680": 3973.963134765625,
        "1574244700": 4121.57958984375,
        "1574244720": 4170.6123046875,
        "1574244740": 4279.62109375,
        "1574244760": 4427.05615234375,
        "1574244780": 4472.11474609375
      },
      "summary": 6700.7467978583445,
      "tags": {
      }
    }
  ]
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.11. OpenAPI DEMO

本文为您介绍公共云环境下，使用实时计算OpenAPI方式操作实时计算作业所需的POM依赖和OpenAPI DEMO。

POM依赖

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-foas</artifactId>
  <version>2.8.0</version>
</dependency>
```

 **说明** 如果需要依赖 aliyun-java-sdk-core，请使用4.3.0及以上版本。4.3.0版本依赖示例如下。

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-core</artifactId>
  <version>4.3.0</version>
</dependency>
```

OpenAPI DEMO

```

package sample;
import com.aliyuncs.AcsRequest;
import com.aliyuncs.AcsResponse;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.foas.model.v20181111.*;
import com.aliyuncs.http.FormatType;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import com.google.gson.Gson;
import java.util.HashMap;
import java.util.Map;
/**
 * 适用于公共云环境。
 *
 * 作业的一般流程是：
 * 如果有依赖package，先创建package。
 * 创建Job->获取planjson->更新planjson到job->上线job->启动job->停止instance。
 * Instance的状态一般如下（流作业为例，括号里的为动作）：
 * Job上线->UNKNOWN[启动job]->WAITING[等待]->RUNNING[暂停]->PAUSED[恢复]->RUNNING[停止]->TERMINAT
ED
 *
 * Note:以下示例仅供演示foas接口的调用方式，不保证逻辑上的严谨性，请按照自己的场景重新组织调用逻辑。
 */
public class PublicSample {
    //填写购买的foas所在的regionId，根据您的实际情况修改。
    private static final String regionId = "cn-shanghai";
    //云账号AK，支持子账号，需要自行保证该账号具有访问实时计算服务的权限（子账号需要得到主账号访问实时计
算服务的权限授权）。
    private static final String accessId = "<yourAccessId>";
    private static final String accessKey = "<yourAccessSecret>";
    private static final String projectName = "<yourProjectName>";
    private static final String jobName = "<yourJobName>";
    //文件夹与Linux文件系统格式相同，以/开始，表示根目录，只支持绝对路径，不支持相对路径。
    private static final String folderName = "/新手任务";
    private static final String clusterId = "<yourClusterId>";//可以通过listProjectBindQueue接口来获得。
    private static final String queueName = "root.<queueName>";//可以通过listProjectBindQueue接口来获得。
    private static final String engineVersion = "current";//Blink或者Flink的版本，current表示使用推荐版本，beta
表示使用测试版本。
    private static final String packageName = "<packageName>";
    private static final String sql =
        "--SQL\n" +
        "--*****-\n" +
        "--CreateTime: 2018-12-29 14:35:04\n" +
        "--Comment: 请输入业务注释信息\n" +
        "--*****-\n" +
        "\n" +
        "CREATE TABLE datahub_source (`name` INT, num VARCHAR) WITH (\n" +
        "  type = 'random'\n" +
        ");\n" +
        "CREATE TABLE datahub_sink (`name` INT, cnt BIGINT) WITH (\n" +
        "  type = 'PRINT'\n" +
        ");\n" +

```

```

"INSERT INTO\n" +
" datahub_sink\n" +
"SELECT\n" +
" `name`,\n" +
" COUNT (1)\n" +
"FROM\n" +
" datahub_source\n" +
"GROUP BY\n" +
" `name`";
private static final String properties =
"#checkpoint模式: EXACTLY_ONCE或者AT_LEAST_ONCE\n" +
"blink.checkpoint.mode=EXACTLY_ONCE\n" +
"#checkpoint间隔时间, 单位毫秒\n" +
"blink.checkpoint.interval.ms=180000\n" +
"#rocksdb的数据生命周期, 单位毫秒\n" +
"#state.backend.rocksdb.ttl.ms=129600000\n";
private static <T extends AcsResponse> T getResponse(IAcsClient client, AcsRequest<T> request) {
AcsResponse response = null;
try {
//必须设置为json。
request.setHttpContentType(FormatType.JSON);
request.setAcceptFormat(FormatType.JSON);
response = client.getAcsResponse(request);
} catch (Exception e) {
//处理异常的逻辑, 请注意异常中的requestId字段, 排查问题时, 需要提供该值给服务端。
if (e instanceof ServerException) {
ServerException serverException = (ServerException) e;
System.out.println(serverException.getRequestId());//本次请求的requestId。
System.out.println(serverException.getErrCode());//本次请求失败的错误码, 用于排查错误。
}
System.out.println(e.getMessage());
}
return (T) response;
}
private static String getClusterMetricJson(String clusterId) {
return String.format(
"{\n" +
"\t\"queries\": [\n" +
"\t\t\t\"metric\": \"%s.system.cpu.user\",\n" +
"\t\t\t\"aggregator\": \"avg\",\n" +
"\t\t\t\"granularity\": \"20s\",\n" +
"\t\t\t\"downsample\": \"avg\"\n" +
"\t},\n" +
"\t\t\t\"metric\": \"%s.system.cpu.iowait\",\n" +
"\t\t\t\"aggregator\": \"avg\",\n" +
"\t\t\t\"granularity\": \"20s\",\n" +
"\t\t\t\"downsample\": \"avg\"\n" +
"\t}],\n" +
"\t\"start\": %s,\n" +
"\t\"end\": %s,\n" +
"\t\"limit\": \"avg:sample:50\"\n" +
"}",
clusterId, clusterId, System.currentTimeMillis() - 3600 * 1000, System.currentTimeMillis()
);

```

```
}
public static void main(String[] args) throws Exception {
    IClientProfile profile = DefaultProfile.getProfile(regionId, accessId, accessKey);
    DefaultAcsClient client = new DefaultAcsClient(profile);
    client.setAutoRetry(false); // 不进行自动重试。
    // 创建一个独享集群，该接口是异步接口，调用返回后需要调用listCluster接口来获取集群的状态来判断是否创建完成。
    CreateClusterRequest createClusterRequest = new CreateClusterRequest();
    createClusterRequest.setDescription("test cluster");
    createClusterRequest.setDisplayName("aaaa");
    createClusterRequest.setOrderId("yyyy"); // 填写购买独享集群下单后生成的订单ID。
    createClusterRequest.setUserOssBucket("myBucket");
    createClusterRequest.setUserVpcId("xxxxx");
    createClusterRequest.setUserVSwitch("yyyy");
    createClusterRequest.setZoneId("cn-shanghai-f");
    CreateClusterResponse createClusterResponse = PublicSample.getResponse(client, createClusterRequest);
    System.out.println(new Gson().toJson(createClusterResponse)); // 该response中含有clusterId信息，这个是集群的索引，后面的大部分接口都需要使用到。
    // 根据条件搜索集群，不设置的条件留null。
    ListClusterRequest listClusterRequest = new ListClusterRequest();
    listClusterRequest.setClusterId(clusterId);
    listClusterRequest.setRegion(regionId);
    ListClusterResponse listClusterResponse = PublicSample.getResponse(client, listClusterRequest);
    System.out.println(new Gson().toJson(listClusterResponse));
    // 在该集群上创建queue，只能创建一级queue。
    CreateQueueRequest createQueueRequest = new CreateQueueRequest();
    createQueueRequest.setClusterId(clusterId);
    createQueueRequest.setQueueName(queueName);
    createQueueRequest.setMaxVcore(300);
    createQueueRequest.setMaxMemMB(4 * 1024);
    CreateQueueResponse createQueueResponse = PublicSample.getResponse(client, createQueueRequest);
    System.out.println(new Gson().toJson(createQueueResponse));
    // 获取集群的引擎版本信息。
    GetClusterEngineVersionsRequest getClusterEngineVersionsRequest = new GetClusterEngineVersionsRequest();
    getClusterEngineVersionsRequest.setClusterId(clusterId);
    GetClusterEngineVersionsResponse getClusterEngineVersionsResponse = PublicSample.getResponse(client, getClusterEngineVersionsRequest);
    System.out.println(new Gson().toJson(getClusterEngineVersionsResponse));
    // 获取该集群的资源信息。
    GetClusterResourceRequest getClusterResourceRequest = new GetClusterResourceRequest();
    getClusterResourceRequest.setClusterId(clusterId);
    GetClusterResourceResponse getClusterResourceResponse = PublicSample.getResponse(client, getClusterResourceRequest);
    System.out.println(new Gson().toJson(getClusterResourceResponse));
    // 获取集群的queue信息，包括所有的queue和各个queue的资源信息。
    GetClusterQueueInfoRequest getClusterQueueInfoRequest = new GetClusterQueueInfoRequest();
    getClusterQueueInfoRequest.setClusterId(clusterId);
    GetClusterQueueInfoResponse getClusterQueueInfoResponse = PublicSample.getResponse(client, getClusterQueueInfoRequest);
    System.out.println(new Gson().toJson(getClusterQueueInfoResponse));
    // 获取集群的detail信息。
```

```
GetClusterDetailsRequest getClusterDetailsRequest = new GetClusterDetailsRequest();
getClusterDetailsRequest.setClusterId(clusterId);
GetClusterDetailsResponse getClusterDetailsResponse = PublicSample.getResponse(client, getClusterDe
tailsRequest);
System.out.println(new Gson().toJson(getClusterDetailsResponse));
//获取集群的metrics曲线，最多只能获取最近两小时的数据。
GetClusterMetricsRequest getClusterMetricsRequest = new GetClusterMetricsRequest();
getClusterMetricsRequest.setClusterId(clusterId);
getClusterMetricsRequest.setMetricJson(getClusterMetricJson(clusterId));//指标的名称请查看文档。
GetClusterMetricsResponse getClusterMetricsResponse = PublicSample.getResponse(client, getClusterM
etricsRequest);
System.out.println(new Gson().toJson(getClusterMetricsResponse));
//注意：以下操作独享集群的接口只支持按量付费的独享集群。
//扩容集群，该接口是异步接口，需要通过集群状态来判断是否操作完成。
ExpandClusterRequest expandClusterRequest = new ExpandClusterRequest();
expandClusterRequest.setClusterId(clusterId);
expandClusterRequest.setModel("Ecs_4c8g");//该参数可选，当前不支持混布，只能填与原有的机型一致的机
型，或者不填写。
expandClusterRequest.setCount(20);//目标集群台数。
ExpandClusterResponse expandClusterResponse = PublicSample.getResponse(client, expandClusterRe
quest);
System.out.println(new Gson().toJson(expandClusterResponse));
//修改集群master的机型配置，只能更改master机型，不能修改master数量。
//注意：该接口是异步接口，需要通过集群状态来判断是否操作完成。
ModifyMasterSpecRequest modifyMasterSpecRequest = new ModifyMasterSpecRequest();
modifyMasterSpecRequest.setClusterId(clusterId);
modifyMasterSpecRequest.setMasterTargetModel("Ecs_8c16g");
ModifyMasterSpecResponse modifyMasterSpecResponse = PublicSample.getResponse(client, modifyMa
sterSpecRequest);
System.out.println(new Gson().toJson(modifyMasterSpecResponse));
//缩容集群，缩容有两种参数提供形式（当前仅支持第一种形式）：
//1.直接填需要去除的实例instanceId（可以通过getClusterDetails接口获得）。
//2.提供每种机型的目标台数。
//注意：该接口是异步接口，需要通过集群状态来判断是否操作完成。
ShrinkClusterRequest shrinkClusterRequest = new ShrinkClusterRequest();
shrinkClusterRequest.setClusterId(clusterId);
shrinkClusterRequest.setInstanceIds("111,222,333");
//shrinkClusterRequest.setModelTargetCount("{\"Ecs_4c8g\":2,\"Ecs_8c16g\":7}");
ShrinkClusterResponse shrinkClusterResponse = PublicSample.getResponse(client, shrinkClusterReques
t);
System.out.println(new Gson().toJson(shrinkClusterResponse));
//销毁集群，只能销毁按量付费的集群。注意：该接口操作风险高，请务必谨慎操作。
DestroyClusterRequest destroyClusterRequest = new DestroyClusterRequest();
destroyClusterRequest.setClusterId(clusterId);
DestroyClusterResponse destroyClusterResponse = PublicSample.getResponse(client, destroyClusterRe
quest);
System.out.println(new Gson().toJson(destroyClusterResponse));
//创建project，该接口分两种场景：
//1.购买的共享集群，需要提供订单ID。
//2.独享集群上创建project，需要提供独享集群的clusterId。
CreateProjectRequest createProjectRequest = new CreateProjectRequest();
//createProjectRequest.setOrderId();//在共享集群上创建project时必须，填写购买的订单id。
createProjectRequest.setDescription("test project");
createProjectRequest.setDeployType("cell");//在共享集群上创建project时填public。
createProjectRequest.setClusterId(clusterId);//在共享集群上创建project时可不填
```

```
createProjectRequest.setClusterId(clusterId); // 在共享集群上创建project时可选项。
createProjectRequest.setName(projectName);
CreateProjectResponse createProjectResponse = PublicSample.getResponse(client, createProjectRequest);
System.out.println(new Gson().toJson(createProjectResponse));
// 绑定queue到project, 该操作类似于将资源（某个queue）指定给某个project使用, 只有将资源绑定到了某个project上,
// 该project里的作业才能使用该资源。当前只支持一对一的关系, 即一个project只能绑定一个queue, 一个queue只能被绑定至一个project上。
BindQueueRequest bindQueueRequest = new BindQueueRequest();
bindQueueRequest.setProjectName(projectName);
bindQueueRequest.setClusterId(clusterId);
bindQueueRequest.setQueueName(queueName);
BindQueueResponse bindQueueResponse = PublicSample.getResponse(client, bindQueueRequest);
System.out.println(new Gson().toJson(bindQueueResponse));
// 获取指定project已绑定的资源。
ListProjectBindQueueRequest listProjectBindQueueRequest = new ListProjectBindQueueRequest();
listProjectBindQueueRequest.setProjectName(projectName);
ListProjectBindQueueResponse listProjectBindQueueResponse = PublicSample.getResponse(client, listProjectBindQueueRequest);
System.out.println(new Gson().toJson(listProjectBindQueueResponse));
// 获取project详情。
GetProjectRequest getProjectRequest = new GetProjectRequest();
getProjectRequest.setProjectName(projectName);
GetProjectResponse getProjectResponse = PublicSample.getResponse(client, getProjectRequest);
System.out.println(new Gson().toJson(getProjectResponse));
// 根据条件搜索project。
ListProjectRequest listProjectRequest = new ListProjectRequest();
listProjectRequest.setName(projectName);
listProjectRequest.setDeployType("cell");
listProjectRequest.setRegion("cn-shanghai");
ListProjectResponse listProjectResponse = PublicSample.getResponse(client, listProjectRequest);
System.out.println(new Gson().toJson(listProjectResponse));
// 从project解绑资源。注意：解绑前, 请先将使用了该资源的作业全部下线。
UnbindQueueRequest unbindQueueRequest = new UnbindQueueRequest();
unbindQueueRequest.setProjectName(projectName);
unbindQueueRequest.setClusterId(clusterId);
unbindQueueRequest.setQueueName(queueName);
UnbindQueueResponse unbindQueueResponse = PublicSample.getResponse(client, unbindQueueRequest);
System.out.println(new Gson().toJson(unbindQueueResponse));
// 删除project。注意:删除project是高危操作, 请务必谨慎操作。删除的条件是该project下的所有作业均已下线。
DeleteProjectRequest deleteProjectRequest = new DeleteProjectRequest();
deleteProjectRequest.setProjectName(projectName);
DeleteProjectResponse deleteProjectResponse = PublicSample.getResponse(client, deleteProjectRequest);
System.out.println(new Gson().toJson(deleteProjectResponse));
// 获取文件夹列表。
ListChildFolderRequest listChildFolderRequest = new ListChildFolderRequest();
listChildFolderRequest.setPath("/");
listChildFolderRequest.setProjectName(projectName);
ListChildFolderResponse listChildFolderResponse = PublicSample.getResponse(client, listChildFolderRequest);
System.out.println(new Gson().toJson(listChildFolderResponse));
```



```
System.out.println(new Gson().toJson(getFolderResponse));  
//获取文件夹。  
GetFolderRequest getFolderRequest = new GetFolderRequest();  
getFolderRequest.setPath(folderName);  
getFolderRequest.setProjectName(projectName);  
GetFolderResponse getFolderResponse = PublicSample.getResponse(client, getFolderRequest);  
System.out.println(new Gson().toJson(getFolderResponse));  
Long folderId;  
GetFolderResponse.Folder folder = getFolderResponse.getFolder();  
if (folder == null || folder.getFolderId() == null) {  
    //创建文件夹。  
    CreateFolderRequest createFolderRequest = new CreateFolderRequest();  
    createFolderRequest.setPath(folderName);  
    createFolderRequest.setProjectName(projectName);  
    CreateFolderResponse createFolderResponse = PublicSample.getResponse(client, createFolderRequest);  
    System.out.println(new Gson().toJson(createFolderResponse));  
    folderId = createFolderResponse.getFolderId();  
} else {  
    folderId = folder.getFolderId();  
}  
// sdk的Package存在您的OSS上，需要将Package通过OSS console或者OSS sdk上传到OSS上。  
// 此处需要将OSS meta信息提供给foas，  
// 并对foas授权Bucket读取权限(ram role:AliyunStreamDefaultRole)，foas在使用时会从用户指定的OSS和Bucket上下载Package。  
// 请确保OSS的region与foas所在的region一致。  
// 如果project建在独享集群上，则不需要提供Endpoint，Bucket等信息，只需要提供package的OSSPath。  
// 用户的OSS信息在创建独享集群的时候已经提供过了，不需要重复提供。  
CreatePackageRequest createPackageRequest = new CreatePackageRequest();  
createPackageRequest.setProjectName(projectName);  
createPackageRequest.setPackageName(packageName);  
createPackageRequest.setType("jar");  
createPackageRequest.setDescription("test package");  
createPackageRequest.setOssEndpoint("oss-cn-hangzhou.aliyuncs.com");//提供的endpoint需要保证网络可通达。  
createPackageRequest.setOssBucket("your bucket");  
createPackageRequest.setOssOwner("11111111");//OSS的主账号uid。  
createPackageRequest.setOssPath("aaa/bbb/ccc.jar");  
createPackageRequest.setMd5("123456");//可不填。不填，则跳过md5校验；填写，则在下载时进行完整性校验。  
CreatePackageResponse createPackageResponse = PublicSample.getResponse(client, createPackageRequest);  
System.out.println(new Gson().toJson(createPackageResponse));  
//更新package。  
UpdatePackageRequest updatePackageRequest = new UpdatePackageRequest();  
updatePackageRequest.setProjectName(projectName);  
updatePackageRequest.setPackageName(packageName);  
updatePackageRequest.setOssPath("aaa/bbb/ddd.jar");  
UpdatePackageResponse updatePackageResponse = PublicSample.getResponse(client, updatePackageRequest);  
System.out.println(new Gson().toJson(updatePackageResponse));  
//获取package详情。  
GetPackageRequest getPackageRequest = new GetPackageRequest();  
getPackageRequest.setProjectName(projectName);  
getPackageRequest.setPackageName(packageName);
```

```
GetPackageResponse getPackageResponse = PublicSample.getResponse(client, getPackageRequest);
System.out.println(new Gson().toJson(getPackageResponse));
//根据条件搜索package。
ListPackageRequest listPackageRequest = new ListPackageRequest();
listPackageRequest.setProjectName(projectName);
listPackageRequest.setPackageName("aa");//模糊匹配。
listPackageRequest.setType("jar");
listPackageRequest.setPageIndex(1);//可选，默认1。
listPackageRequest.setPageSize(30);//可选，默认10。
ListPackageResponse listPackageResponse = PublicSample.getResponse(client, listPackageRequest);
System.out.println(new Gson().toJson(listPackageResponse));
//列举project绑定的cluster和queue。
ListProjectBindQueueRequest listProjectBindQueueRequest2 = new ListProjectBindQueueRequest();
listProjectBindQueueRequest2.setProjectName(projectName);
ListProjectBindQueueResponse listProjectBindQueueResponse2 = PublicSample.getResponse(client, list
ProjectBindQueueRequest2);
System.out.println(new Gson().toJson(listProjectBindQueueResponse2));
//查询project下绑定的queue的资源。
ListProjectBindQueueResourceRequest listProjectBindQueueResourceRequest = new ListProjectBindQue
ueResourceRequest();
listProjectBindQueueResourceRequest.setProjectName(projectName);
ListProjectBindQueueResourceResponse listProjectBindQueueResourceResponse = PublicSample.getRes
ponse(client, listProjectBindQueueResourceRequest);
System.out.println(new Gson().toJson(listProjectBindQueueResourceResponse));
//根据条件搜索project下的作业。
ListJobRequest listJobRequest = new ListJobRequest();
listJobRequest.setProjectName(projectName);
listJobRequest.setJobType("flink_stream");
listJobRequest.setApiType("sql");
//该参数控制返回的job信息中是否返回较长的字段（例如code）。填false，则返回的只有部分字段；不填或者
填true，则返回所有字段。
listJobRequest.setIsShowFullField(false);
ListJobResponse listJobResponse = PublicSample.getResponse(client, listJobRequest);
System.out.println(new Gson().toJson(listJobResponse));
//获取作业。
GetJobRequest getJobRequest = new GetJobRequest();
getJobRequest.setProjectName(projectName);
getJobRequest.setJobName(jobName);
GetJobResponse getJobResponse = PublicSample.getResponse(client, getJobRequest);
System.out.println(new Gson().toJson(getJobResponse));
GetJobResponse.Job job = getJobResponse.getJob();
if (job == null || job.getJobName() == null) {
    //创建作业。
    CreateJobRequest createJobRequest = new CreateJobRequest();
    createJobRequest.setProjectName(projectName);
    createJobRequest.setJobName(jobName);
    createJobRequest.setCode(sql);
    createJobRequest.setProperties(properties);//无，则不用设置。
    //clusterId和queueName可以不配置。如果不配置，则随机从项目绑定的cluster和queue中选择一个。
    createJobRequest.setClusterId(clusterId);
    createJobRequest.setQueueName(queueName);
    //engineVersion可以不配置。如果不配置，则使用current版本（推荐版本）。
    createJobRequest.setEngineVersion(engineVersion);
    createJobRequest.setDescription("test");
}
```

```
createJobRequest.setPackages(packageName);//多个package之间用英文逗号分隔。
createJobRequest.setJobType("flink_stream");//flink_stream/flink_batch大小写不敏感。
createJobRequest.setApiType("sql");//datastream/sql大小写不敏感。
createJobRequest.setFolderId(folderId);
CreateJobResponse createJobResponse = PublicSample.getResponse(client, createJobRequest);
System.out.println(new Gson().toJson(createJobResponse));
} else {
    //修改作业。
    UpdateJobRequest updateJobRequest = new UpdateJobRequest();
    updateJobRequest.setProjectName(projectName);
    updateJobRequest.setJobName(jobName);
    updateJobRequest.setCode(sql);//设置需要修改的字段，不修改的字段不需要设置。
    UpdateJobResponse updateJobResponse = PublicSample.getResponse(client, updateJobRequest);
    System.out.println(new Gson().toJson(updateJobResponse));
}
//返回引用指定package的job，该接口查询上线后的job引用关系，未上线的job不会返回。
GetRefPackageJobRequest getRefPackageJobRequest = new GetRefPackageJobRequest();
getRefPackageJobRequest.setProjectName(projectName);
getRefPackageJobRequest.setPackageName(packageName);
GetRefPackageJobResponse getRefPackageJobResponse = PublicSample.getResponse(client, getRefPackageJobRequest);
System.out.println(new Gson().toJson(getRefPackageJobResponse));
//校验job是否存在语法错误。
ValidateJobRequest validateJobRequest = new ValidateJobRequest();
validateJobRequest.setProjectName(projectName);
validateJobRequest.setJobName(jobName);
ValidateJobResponse validateJobResponse = PublicSample.getResponse(client, validateJobRequest);
System.out.println(new Gson().toJson(validateJobResponse));
//获取job的planjson。该接口是异步接口，提交完成之后，需要调用CheckRawPlanJson接口获取结果:Session
in run/success/fail。
GetRawPlanJsonRequest getRawPlanJsonRequest = new GetRawPlanJsonRequest();
getRawPlanJsonRequest.setProjectName(projectName);
getRawPlanJsonRequest.setJobName(jobName);
getRawPlanJsonRequest.setAutoconfEnable(true);//可以不填，默认为false。
//设置这个job预期使用多少资源来运行，这个参数会影响到生成plan的大小，不提供的话，底层引擎会生成默认
资源配置。
getRawPlanJsonRequest.setExpectedCore(2f);//这个参数与下面的参数可以同时提供，也可以同时不提供，
不允许一个提供，一个不提供。
getRawPlanJsonRequest.setExpectedGB(9f);
GetRawPlanJsonResponse getRawPlanJsonResponse = PublicSample.getResponse(client, getRawPlanJsonRequest);
CheckRawPlanJsonRequest checkRawPlanJsonRequest = new CheckRawPlanJsonRequest();
checkRawPlanJsonRequest.setProjectName(projectName);
checkRawPlanJsonRequest.setJobName(jobName);
checkRawPlanJsonRequest.setSessionId(getRawPlanJsonResponse.getSessionId());
String planJson;
CheckRawPlanJsonResponse checkRawPlanJsonResponse;
while (true) {
    checkRawPlanJsonResponse = PublicSample.getResponse(client, checkRawPlanJsonRequest);
    System.out.println(checkRawPlanJsonResponse.getPlanJsonInfo().getStatus());
    if (checkRawPlanJsonResponse.getPlanJsonInfo().getStatus().equals("success")) {
        planJson = checkRawPlanJsonResponse.getPlanJsonInfo().getPlanJson();
        break;
    } else if (checkRawPlanJsonResponse.getPlanJsonInfo().getStatus().equals("fail")) {
```

```
        System.out.println(checkRawPlanJsonResponse.getPlanJsonInfo().getErrorMessage());
        return;
    }
    Thread.sleep(1000);//每秒查询一次。
}
//修改作业planjson。
UpdateJobRequest updateJobRequest = new UpdateJobRequest();
updateJobRequest.setProjectName(projectName);
updateJobRequest.setJobName(jobName);
updateJobRequest.setPlanJson(planJson);
UpdateJobResponse updateJobResponse = PublicSample.getResponse(client, updateJobRequest);
System.out.println(new Gson().toJson(updateJobResponse));
//预估当前作业所配置的planjson需要的资源数，为上线接口需要提供的maxCU数值提供参考，仅blink-3.2.1及以上版本支持该功能。
CalcPlanJsonResourceRequest calcPlanJsonResourceRequest = new CalcPlanJsonResourceRequest();
calcPlanJsonResourceRequest.setProjectName(projectName);
calcPlanJsonResourceRequest.setJobName(jobName);
CalcPlanJsonResourceResponse calcPlanJsonResourceResponse = PublicSample.getResponse(client, calcPlanJsonResourceRequest);
System.out.println(new Gson().toJson(calcPlanJsonResourceResponse));
//获取作业上一次运行记录的最后一次自动调优的planjson，注意:是上一次的运行记录，包括停止的和暂停的。
如果不存在，则返回null。
GetJobLatestAutoScalePlanRequest getJobLatestAutoScalePlanRequest = new GetJobLatestAutoScalePlanRequest();
getJobLatestAutoScalePlanRequest.setProjectName(projectName);
getJobLatestAutoScalePlanRequest.setJobName(jobName);
GetJobLatestAutoScalePlanResponse getJobLatestAutoScalePlanResponse = PublicSample.getResponse(client, getJobLatestAutoScalePlanRequest);
System.out.println(new Gson().toJson(getJobLatestAutoScalePlanResponse));
//上线作业。
CommitJobRequest commitJobRequest = new CommitJobRequest();
commitJobRequest.setProjectName(projectName);
commitJobRequest.setJobName(jobName);
commitJobRequest.setIsOnOff(true);//是否开启autoscale自动调优。不填，表示不开启，仅blink-3.2.1及以上版本支持该功能。
commitJobRequest.setMaxCU(100);//设置最大资源使用上限。
commitJobRequest.setConfigure("{\"fetch_delay\":44.5}");//json格式，当前仅支持这一个配置项，大小写敏感，value是double型。
CommitJobResponse commitJobResponse = PublicSample.getResponse(client, commitJobRequest);
System.out.println(new Gson().toJson(commitJobResponse));
//获取实例。
GetInstanceRequest getInstanceRequest = new GetInstanceRequest();
getInstanceRequest.setProjectName(projectName);
getInstanceRequest.setJobName(jobName);
// -1表示获取流作业的当前运行实例（因为流作业某一时刻最多只会存在一个实例）。注意：批作业需要填实际的运行实例id。
getInstanceRequest.setInstanceId(-1L);
GetInstanceResponse getInstanceResponse = PublicSample.getResponse(client, getInstanceRequest);
System.out.println(new Gson().toJson(getInstanceResponse));
GetInstanceResponse.Instance instance = getInstanceResponse.getInstance();
{
    //启动作业。
    if (instance == null || "UNKNOWN".equals(instance.getActualState()) || "TERMINATED".equals(instance.getActualState())) {
```

```

StartJobRequest startJobRequest = new StartJobRequest();
startJobRequest.setProjectName(projectName);
startJobRequest.setJobName(jobName);
Map map = new HashMap<String, String>();
//startOffset表示启动点位。
map.put("startOffset", String.valueOf(System.currentTimeMillis()));//精确到ms时间戳。
startJobRequest.setParameterJson(new Gson().toJson(map));//json格式参数。
StartJobResponse startJobResponse = PublicSample.getResponse(client, startJobRequest);
System.out.println(new Gson().toJson(startJobResponse));
//等待启动成功。
while (true) {
    GetInstanceRunSummaryRequest getInstanceRunSummaryRequest = new GetInstanceRunSummaryRequest();
    getInstanceRunSummaryRequest.setProjectName(projectName);
    getInstanceRunSummaryRequest.setJobName(jobName);
    getInstanceRunSummaryRequest.setInstanceId(-1L);
    GetInstanceRunSummaryResponse getInstanceRunSummaryResponse = PublicSample.getResponse(client, getInstanceRunSummaryRequest);
    System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
    if ("WAITING".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
        System.out.println(String.format("LastErrorTime[%s], lastErrorMessage[%s]",
            getInstanceRunSummaryResponse.getRunSummary().getLastErrorTime(),
            getInstanceRunSummaryResponse.getRunSummary().getLastErrorErrorMessage());
        Thread.sleep(1000);
    } else {
        break;
    }
}
}
}
{
    // 作业启动成功后，可以调用如下的接口进行运维。
    //获取instance的运行dag图。
    GetInstanceDetailRequest getInstanceDetailRequest = new GetInstanceDetailRequest();
    getInstanceDetailRequest.setProjectName(projectName);
    getInstanceDetailRequest.setJobName(jobName);
    getInstanceDetailRequest.setInstanceId(-1L);
    GetInstanceDetailResponse getInstanceDetailResponse = PublicSample.getResponse(client, getInstanceDetailRequest);
    System.out.println(new Gson().toJson(getInstanceDetailResponse));
    //获取instance的运行配置。
    GetInstanceConfigRequest getInstanceConfigRequest = new GetInstanceConfigRequest();
    getInstanceConfigRequest.setProjectName(projectName);
    getInstanceConfigRequest.setJobName(jobName);
    getInstanceConfigRequest.setInstanceId(-1L);
    GetInstanceConfigResponse getInstanceConfigResponse = PublicSample.getResponse(client, getInstanceConfigRequest);
    System.out.println(new Gson().toJson(getInstanceConfigResponse));
    //获取instance的运行checkpoint信息。
    GetInstanceCheckpointRequest getInstanceCheckpointRequest = new GetInstanceCheckpointRequest();
    getInstanceCheckpointRequest.setProjectName(projectName);
    getInstanceCheckpointRequest.setJobName(jobName);
    getInstanceCheckpointRequest.setInstanceId(-1L);
    GetInstanceCheckpointResponse getInstanceCheckpointResponse = PublicSample.getResponse(client

```

```

    getInstanceCheckpointResponse getInstanceCheckpointResponse = PublicSample.getResponse(client
, getInstanceCheckpointRequest);
    System.out.println(new Gson().toJson(getInstanceCheckpointResponse));
    //获取instance的运行时异常信息(failover信息)。
    GetInstanceExceptionsRequest getInstanceExceptionsRequest = new GetInstanceExceptionsRequest()
;
    getInstanceExceptionsRequest.setProjectName(projectName);
    getInstanceExceptionsRequest.setJobName(jobName);
    getInstanceExceptionsRequest.setInstanceId(-1L);
    GetInstanceExceptionsResponse getInstanceExceptionsResponse = PublicSample.getResponse(client,
getInstanceExceptionsRequest);
    System.out.println(new Gson().toJson(getInstanceExceptionsResponse));
    //如果上线时作业开启了自动调优，调优效果不理想，可中途关闭自动调优，关闭之后也可以再次打开。
    UpdateAutoScaleConfigRequest updateAutoScaleConfigRequest = new UpdateAutoScaleConfigRequ
est();
    updateAutoScaleConfigRequest.setProjectName(projectName);
    updateAutoScaleConfigRequest.setJobName(jobName);
    updateAutoScaleConfigRequest.setInstanceId(-1L);
    updateAutoScaleConfigRequest.setConfigJson("{\"autoscale.enable\":false}");//json格式，当前只支持
这个key，大小写敏感，取值为true或者false。
    UpdateAutoScaleConfigResponse updateAutoScaleConfigResponse = PublicSample.getResponse(clie
nt, updateAutoScaleConfigRequest);
    System.out.println(new Gson().toJson(updateAutoScaleConfigResponse));
    //查询instance的运行时各项指标的数据曲线信息。
    //当前只支持查询最近两小时的曲线，metric的格式为：blink.{projectName}.{jobName}.{metricName}。
    String metricJson = "{\n" +
        "\t\t\"start\": 1547637620000,\n" + //精确到ms时间戳。
        "\t\t\"limit\": \"avg:sample:50\",\n" +
        "\t\t\"end\": 1547638420000,\n" +
        "\t\t\"queries\": [{\n" +
        "\t\t\t\t\"downsample\": \"20s-avg\",\n" +
        "\t\t\t\t\"metric\": \"blink.bayes_team.huayuan_test_job.delay\",\n" +
        "\t\t\t\t\"granularity\": \"20s\",\n" +
        "\t\t\t\t\"aggregator\": \"max\"\n" +
        "\t\t\t},\n" +
        "\t\t\t\t\"downsample\": \"20s-avg\",\n" +
        "\t\t\t\t\"metric\": \"blink.bayes_team.huayuan_test_job.fetched_delay\",\n" +
        "\t\t\t\t\"granularity\": \"20s\",\n" +
        "\t\t\t\t\"aggregator\": \"max\"\n" +
        "\t\t\t}]\n" +
        "}";
    GetInstanceMetricRequest getInstanceMetricRequest = new GetInstanceMetricRequest();
    getInstanceMetricRequest.setProjectName(projectName);
    getInstanceMetricRequest.setJobName(jobName);
    getInstanceMetricRequest.setInstanceId(-1L);
    getInstanceMetricRequest.setMetricJson(metricJson);
    GetInstanceMetricResponse getInstanceMetricResponse = PublicSample.getResponse(client, getInsta
nceMetricRequest);
    System.out.println(new Gson().toJson(getInstanceMetricResponse.getMetrics()));
    //获取instance实际使用的资源信息（cpu/memory）。
    GetInstanceResourceRequest getInstanceResourceRequest = new GetInstanceResourceRequest();
    getInstanceResourceRequest.setProjectName(projectName);
    getInstanceResourceRequest.setJobName(jobName);
    getInstanceResourceRequest.setInstanceId(-1L);
    GetInstanceResourceResponse getInstanceResourceResponse = PublicSample.getResponse(client, ae

```

```
tInstanceResourceRequest);
    System.out.println(new Gson().toJson(getInstanceResourceResponse.getResource()));
    //批量获取多个作业的运行状态。
    BatchGetInstanceRunSummaryRequest batchGetInstanceRunSummaryRequest = new BatchGetInstanceRunSummaryRequest();
    batchGetInstanceRunSummaryRequest.setProjectName(projectName);
    batchGetInstanceRunSummaryRequest.setJobType("flink_stream");
    batchGetInstanceRunSummaryRequest.setJobNames("job1,job2,job3");//多个job之间用英文逗号分隔
    , 请确保都是stream job, 并且job已存在且已上线, 否则会忽略错误的job。
    BatchGetInstanceRunSummaryResponse batchGetInstanceRunSummaryResponse = PublicSample.getResponse(client, batchGetInstanceRunSummaryRequest);
    System.out.println(new Gson().toJson(batchGetInstanceRunSummaryResponse.getRunSummaries()));
}
{
    //暂停作业。
    GetInstanceRunSummaryRequest getInstanceRunSummaryRequest = new GetInstanceRunSummaryRequest();
    getInstanceRunSummaryRequest.setProjectName(projectName);
    getInstanceRunSummaryRequest.setJobName(jobName);
    getInstanceRunSummaryRequest.setInstanceId(-1L);
    GetInstanceRunSummaryResponse getInstanceRunSummaryResponse = PublicSample.getResponse(client, getInstanceRunSummaryRequest);
    System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
    if ("RUNNING".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
        ModifyInstanceStateRequest modifyInstanceStateRequest = new ModifyInstanceStateRequest();
        modifyInstanceStateRequest.setProjectName(projectName);
        modifyInstanceStateRequest.setJobName(jobName);
        modifyInstanceStateRequest.setInstanceId(-1L);
        modifyInstanceStateRequest.setExpectState("PAUSED");
        ModifyInstanceStateResponse modifyInstanceStateResponse = PublicSample.getResponse(client, modifyInstanceStateRequest);
        System.out.println(new Gson().toJson(modifyInstanceStateResponse));
        //等待暂停成功。
        while (true) {
            getInstanceRunSummaryRequest = new GetInstanceRunSummaryRequest();
            getInstanceRunSummaryRequest.setProjectName(projectName);
            getInstanceRunSummaryRequest.setJobName(jobName);
            getInstanceRunSummaryRequest.setInstanceId(-1L);
            getInstanceRunSummaryResponse = PublicSample.getResponse(client, getInstanceRunSummaryRequest);
            System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
            if ("RUNNING".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
                System.out.println(String.format("lastErrorTime[%s], lastErrorMessage[%s]",
                    getInstanceRunSummaryResponse.getRunSummary().getLastErrorTime(),
                    getInstanceRunSummaryResponse.getRunSummary().getLastErrorMessage()));
                Thread.sleep(1000);
            } else {
                break;
            }
        }
    }
}
{
    //恢复作业。
```

```
GetInstanceRunSummaryRequest getInstanceRunSummaryRequest = new GetInstanceRunSummaryR
equest();
getInstanceRunSummaryRequest.setProjectName(projectName);
getInstanceRunSummaryRequest.setJobName(jobName);
getInstanceRunSummaryRequest.setInstanceId(-1L);
GetInstanceRunSummaryResponse getInstanceRunSummaryResponse = PublicSample.getResponse(
client, getInstanceRunSummaryRequest);
System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
if ("PAUSED".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
    ModifyInstanceStateRequest modifyInstanceStateRequest = new ModifyInstanceStateRequest();
    modifyInstanceStateRequest.setProjectName(projectName);
    modifyInstanceStateRequest.setJobName(jobName);
    modifyInstanceStateRequest.setInstanceId(-1L);
    modifyInstanceStateRequest.setExpectState("RUNNING");
    ModifyInstanceStateResponse modifyInstanceStateResponse = PublicSample.getResponse(client,
modifyInstanceStateRequest);
    System.out.println(new Gson().toJson(modifyInstanceStateResponse));
    //等待恢复成功。
    while (true) {
        getInstanceRunSummaryRequest = new GetInstanceRunSummaryRequest();
        getInstanceRunSummaryRequest.setProjectName(projectName);
        getInstanceRunSummaryRequest.setJobName(jobName);
        getInstanceRunSummaryRequest.setInstanceId(-1L);
        getInstanceRunSummaryResponse = PublicSample.getResponse(client, getInstanceRunSummar
yRequest);
        System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
        if ("PAUSED".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
            System.out.println(String.format("lastErrorTime[%s], lastErrorMessage[%s]",
                getInstanceRunSummaryResponse.getRunSummary().getLastErrorTime(),
                getInstanceRunSummaryResponse.getRunSummary().getLastErrorMessage()));
            Thread.sleep(1000);
        } else {
            break;
        }
    }
}
}
{
    //停止作业。
    GetInstanceRunSummaryRequest getInstanceRunSummaryRequest = new GetInstanceRunSummaryR
equest();
    getInstanceRunSummaryRequest.setProjectName(projectName);
    getInstanceRunSummaryRequest.setJobName(jobName);
    getInstanceRunSummaryRequest.setInstanceId(-1L);
    GetInstanceRunSummaryResponse getInstanceRunSummaryResponse = PublicSample.getResponse(
client, getInstanceRunSummaryRequest);
    System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
    if ("RUNNING".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
        ModifyInstanceStateRequest modifyInstanceStateRequest = new ModifyInstanceStateRequest();
        modifyInstanceStateRequest.setProjectName(projectName);
        modifyInstanceStateRequest.setJobName(jobName);
        modifyInstanceStateRequest.setInstanceId(-1L);
        modifyInstanceStateRequest.setExpectState("TERMINATED");
        ModifyInstanceStateResponse modifyInstanceStateResponse = PublicSample.getResponse(client,
```



```
modifyInstanceStateRequest);
System.out.println(new Gson().toJson(modifyInstanceStateResponse));
//等待停止成功。
while (true) {
    getInstanceRunSummaryRequest = new GetInstanceRunSummaryRequest();
    getInstanceRunSummaryRequest.setProjectName(projectName);
    getInstanceRunSummaryRequest.setJobName(jobName);
    getInstanceRunSummaryRequest.setInstanceId(-1L);
    getInstanceRunSummaryResponse = PublicSample.getResponse(client, getInstanceRunSummaryRequest);
    System.out.println(new Gson().toJson(getInstanceRunSummaryResponse));
    if ("RUNNING".equals(getInstanceRunSummaryResponse.getRunSummary().getActualState())) {
        System.out.println(String.format("lastErrorTime[%s], lastErrorMessage[%s]",
            getInstanceRunSummaryResponse.getRunSummary().getLastErrorTime(),
            getInstanceRunSummaryResponse.getRunSummary().getLastErrorMessage()));
        Thread.sleep(1000);
    } else {
        break;
    }
}
//下线作业。
OfflineJobRequest offlineJobRequest = new OfflineJobRequest();
offlineJobRequest.setProjectName(projectName);
offlineJobRequest.setJobName(jobName);
OfflineJobResponse offlineJobResponse = PublicSample.getResponse(client, offlineJobRequest);
System.out.println(new Gson().toJson(offlineJobResponse));
//删除作业。
DeleteJobRequest deleteJobRequest = new DeleteJobRequest();
deleteJobRequest.setProjectName(projectName);
deleteJobRequest.setJobName(jobName);
DeleteJobResponse deleteJobResponse = PublicSample.getResponse(client, deleteJobRequest);
System.out.println(new Gson().toJson(deleteJobResponse));
//删除package。
DeletePackageRequest deletePackageRequest = new DeletePackageRequest();
deletePackageRequest.setProjectName(projectName);
deletePackageRequest.setPackageName(packageName);
DeletePackageResponse deletePackageResponse = PublicSample.getResponse(client, deletePackageRequest);
System.out.println(new Gson().toJson(deletePackageResponse));
}
```

8.12. Project

8.12.1. CreateProject

您可以通过CreateProject OpenAPI在集群中创建项目。

 说明 不能通过子账号的AK信息以openAPI方式创建项目。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
POST /api/v2/projects HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
deployType	String	是	public	集群类型： <ul style="list-style-type: none">独享集群：cell共享集群：public
name	String	否	project1	项目名称
orderId	String	否	stream-abcd	共享模式实例ID ? 说明 仅共享模式需要填写。
clusterId	String	否	cmy99ugusuco66x9qc6k****	集群ID
managerIds	String	否	1234567	主账号ID
description	String	否	测试项目	项目描述

返回数据

名称	类型	示例值	描述
RequestId	String	B661F03F-0F9D-4EFF-A40F-4ED1519A549A	请求ID

示例

请求示例

```
/api/v2/projects
{"name":"project1","orderId":"stream-abcd","clusterId":"cmy99ugusuco66x9qc6k****","managerIds":"1234567","description":"测试项目","deployType":"cell"}
```

正常返回示例

XML 格式

```
<requestId>B661F03F-0F9D-4EFF-A40F-4ED1519A549A</requestId>
```

JSON 格式

```
{"requestId":"B661F03F-0F9D-4EFF-A40F-4ED1519A549A"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.12.2. DeleteProject

您可以通过DeleteProject OpenAPI删除集群中的项目。如果需要删除的项目已经和Queue绑定，也会将项目和Queue解绑（不会删除Queue）。为了防止资源浪费，请及时将Queue绑定新的项目。

 说明 DeleteProject仅支持独享模式。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
DELETE /api/v2/projects/[projectName] HTTPS
```

请求参数

名称	类型	是否必选	示例值	描述
projectName	String	是	project1	项目名称

名称	类型	是否必选	示例值	描述
RegionId	String	是	cn-hangzhou	区域ID  说明 公共云用户请忽略此参数。

返回数据

名称	类型	示例值	描述
RequestId	String	3F4ADBCF-B116-4680-80B3-E9A139841A0D	请求ID

示例

请求示例

```
/api/v2/projects/project1
```

正常返回示例

XML 格式

```
<requestId>3F4ADBCF-B116-4680-80B3-E9A139841A0D</requestId>
```

JSON 格式

```
{"requestId": "3F4ADBCF-B116-4680-80B3-E9A139841A0D"}
```

错误码

访问[错误中心](#)查看更多错误码。

8.12.3. GetProject

您可以通过GetProject OpenAPI查看指定项目的详细信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。


请求语法

GET /api/v2/projects/[projectName] HTTP/1.1

请求参数

名称	类型	位置	是否必选	示例值	描述
projectName	String	Path	是	project1	项目名称。
RegionId	String	Header	否	cn-hangzhou	区域ID

返回数据

名称	类型	示例值	描述
Project	Struct		项目详情。
ClusterId	String	rcmp9x37ztfb63g1x7lt****	集群ID。  说明 您可以使用 listcluster 获取集群ID信息。
CreateTime	Long	1597655424000	项目创建时间。
Creator	String	23996037709276****	项目创建者的UID。
DeployType	String	cell	集群类型： <ul style="list-style-type: none"> 独享集群：cell。 共享集群：public。
Description	String	test	项目描述。
GlobalJobConfig	String	sql.type=print datastream.main.class=test	项目级别系统参数。
Id	String	7ht9ed87rtd48bof9mu8****	请求ID。
ManagerIds	String	107992689699****	管理员ID列表。多个ID使用英文逗号（,）分隔。

名称	类型	示例值	描述
Modifier	String	23996037709276****	项目修改者UID。
ModifyTime	Long	1597655424000	项目修改时间。
Name	String	project1	项目名称。
Region	String	cn-hangzhou	区域ID。
State	String	ENABLE	项目状态。
RequestId	String	4A6BFFEE-6C96-4727-A620-A109F50284DA	请求ID。

示例

请求示例

```
/api/v2/projects/project1
```

正常返回示例

XML 格式

```
<Project>
  <DeployType>CELL</DeployType>
  <ModifyTime>1597655424000</ModifyTime>
  <ManagerIds>107992689699****</ManagerIds>
  <Description>test</Description>
  <ClusterId>rcmp9x37ztfb63g1x7lt****</ClusterId>
  <CreateTime>1597655424000</CreateTime>
  <Creator>23996037709276****</Creator>
  <Name>project1</Name>
  <State>ENABLE</State>
  <Region>cn-hangzhou</Region>
  <Id>7ht9ed87rtd48bof9mu8****</Id>
  <Modifier>23996037709276****</Modifier>
</Project>
<RequestId>4A6BFFEE-6C96-4727-A620-A109F50284DA</RequestId>
```

JSON 格式

```
{
  "Project": {
    "DeployType": "CELL",
    "ModifyTime": 1597655424000,
    "ManagerIds": "107992689699****",
    "Description": "test",
    "ClusterId": "rcmp9x37ztfb63g1x7lt****",
    "CreateTime": 1597655424000,
    "Creator": "23996037709276****",
    "Name": "project1",
    "State": "ENABLE",
    "Region": "cn-hangzhou",
    "Id": "7ht9ed87rtd48bof9mu8****",
    "Modifier": "23996037709276****"
  },
  "RequestId": "4A6BFEE-6C96-4727-A620-A109F50284DA"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.12.4. ListProject

您可以通过ListProject OpenAPI查询多个项目的信息。查询的项目信息可根据参数选择，进行自由组合。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

您可以使用公共请求参数和该接口必填参数。公共请求参数请参见OpenAPI>调用方式>公共请求参数文档。

请求语法

```
GET /api/v2/projects HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
RegionId	String	Header	否	cn-hangzhou	区域ID
name	String	Query	否	job1	项目名称

名称	类型	位置	是否必选	示例值	描述
deployType	String	Query	否	cell	集群类型： <ul style="list-style-type: none"> CELL：独享集群 PUBLIC：共享集群
clusterId	String	Query	否	cmy99ugusuco66x9qc6k****	集群ID ? 说明 您可以使用listcluster获取集群ID信息。
region	String	Query	否	cn-hangzhou	区域ID
pageSize	Integer	Query	否	1	每页显示的项目数
pageIndex	Integer	Query	否	2	查询的页码

返回数据

名称	类型	示例值	描述
PageIndex	Integer	1	查询的页码
PageSize	Integer	10	每页显示的项目数
Projects	Array of Project		项目详情
Project			
ClusterId	String	rcmp9x37ztfb63g1x7lt****	集群ID ? 说明 您可以使用listcluster获取集群ID信息。
CreateTime	Long	1597655424000	项目创建时间
Creator	String	23996037709276****	项目创建者的UID

名称	类型	示例值	描述
DeployType	String	CELL	集群类型： <ul style="list-style-type: none">CELL：独享集群PUBLIC：共享集群
Description	String	test	项目描述
Id	String	e8edisrtou71a2nero i5f3kc	请求ID
ManagerIds	String	1079926896999421	管理员ID列表。多个ID请使用逗号(,)分隔。
Modifier	String	27395716024545****	项目修改者UID
ModifyTime	Long	1605610424000	项目修改时间
Name	String	project1	项目名称
Region	String	cn-hangzhou	区域ID
State	String	ENABLE	项目状态
RequestId	String	5A0EA261-AA2F- 47FF-B7D9- A785ED1D0ADE	请求ID
TotalCount	Long	3	总项目数
TotalPage	Integer	1	总页码数

示例

请求示例

```
/api/v2/projects
```

正常返回示例

XML 格式

```
<TotalCount>3</TotalCount>
<TotalPage>1</TotalPage>
<PageSize>10</PageSize>
<RequestId>5A0EA261-AA2F-47FF-B7D9-A785ED1D0ADE</RequestId>
<Projects>
  <Project>
    <DeployType>CELL</DeployType>
    <ModifyTime>1597655424000</ModifyTime>
    <ManagerIds>1079926896999421</ManagerIds>
    <Description>test</Description>
    <ClusterId>rcmp9x37ztfb63g1x7lt****</ClusterId>
    <State>ENABLE</State>
    <CreateTime>1597655424000</CreateTime>
    <Region>cn-hangzhou</Region>
    <Creator>23996037709276****</Creator>
    <Id>7ht9ed87rtd48bof9mu8895z</Id>
    <Modifier>23996037709276****</Modifier>
    <Name>project1</Name>
  </Project>
  <Project>
    <DeployType>CELL</DeployType>
    <ModifyTime>1605610424000</ModifyTime>
    <ManagerIds>1079926896999421</ManagerIds>
    <Description>stream4demo</Description>
    <ClusterId>rulpqd442sgxbtw9736r****</ClusterId>
    <State>ENABLE</State>
    <CreateTime>1605610424000</CreateTime>
    <Region>cn-beijing</Region>
    <Creator>27395716024545****</Creator>
    <Id>e8edisrtou71a2neroi5f3kc</Id>
    <Modifier>27395716024545****</Modifier>
    <Name>stream4demo</Name>
  </Project>
</Projects>
<PageIndex>1</PageIndex>
```

JSON 格式

```
{
  "TotalCount": 3,
  "TotalPage": 1,
  "PageSize": 10,
  "RequestId": "5A0EA261-AA2F-47FF-B7D9-A785ED1D0ADE",
  "Projects": {
    "Project": [
      {
        "DeployType": "CELL",
        "ModifyTime": 1597655424000,
        "ManagerIds": "1079926896999421",
        "Description": "test",
        "ClusterId": "rcmp9x37ztfb63g1x7lt****",
        "State": "ENABLE",
        "CreateTime": 1597655424000,
        "Region": "cn-hangzhou",
        "Creator": "23996037709276****",
        "Id": "7ht9ed87rtd48bof9mu8895z",
        "Modifier": "23996037709276****",
        "Name": "project1"
      },
      {
        "DeployType": "CELL",
        "ModifyTime": 1605610424000,
        "ManagerIds": "1079926896999421",
        "Description": "stream4demo",
        "ClusterId": "rulpqd442sgxbtw9736r****",
        "State": "ENABLE",
        "CreateTime": 1605610424000,
        "Region": "cn-beijing",
        "Creator": "27395716024545****",
        "Id": "e8edisrtou71a2neroi5f3kc",
        "Modifier": "27395716024545****",
        "Name": "stream4demo"
      }
    ]
  },
  "PageIndex": 1
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.12.5. UpdateProject

您可以通过UpdateProject OpenAPI修改项目参数，项目参数可应用于项目下的所有作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

该接口使用公共请求头，无特殊请求头。请参见公共请求参数文档。

名称	类型	是否必选	示例	描述
RegionId	String	否	cn-hangzhou	地域

请求语法

```
PUT /api/v2/projects/{projectName} HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
globalJobConfig	String	Form Data	否	sql.\${key}=\${value}	项目参数
projectName	String	Path	是	city_brain	项目名称

返回数据

名称	类型	示例值	描述
RequestId	String	A017B343-33DC-4A97-BAC5-4B57872D5B61	请求ID

示例

请求示例

```
PUT /api/v2/projects/city_brain HTTP/1.1
Host:foas.aliyuncs.com
Content-Type:application/json
```

正常返回示例

XML 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<UpdateProjectResponse>
  <RequestId>A017B343-33DC-4A97-BAC5-4B57872D5B61</RequestId>
</UpdateProjectResponse>
```

JSON 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "RequestId" : "A017B343-33DC-4A97-BAC5-4B57872D5B61"
}
```

错误码

访问[错误中心](#)查看更多错误码。

8.12.6. UpdateProjectConfig

您可以通过UpdateProjectConfig OpenAPI为项目开启或停止支持批SQL作业。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求头

该接口使用公共请求头，无特殊请求头。请参见公共请求参数文档。

请求语法

```
PUT /api/v2/project/config HTTP/1.1
```

请求参数

名称	类型	位置	是否必选	示例值	描述
IsOpenBatchSQL	Boolean	Query	是	true	是否开启批SQL作业，参数取值如下： <ul style="list-style-type: none"> true：项目开启支持批SQL作业。 false：项目停止支持批SQL作业。
ProjectName	String	Query	是	city_brain	项目名

返回数据

名称	类型	示例值	描述
Data	Boolean	true	批SQL开启或停用成功
RequestId	String	2E62CA21-83FE-4CE1-B7E2-E868F7272E6B	请求ID

示例

请求示例

```
PUT /api/v2/project/config?IsOpenBatchSQL=true&ProjectName=city_brain HTTP/1.1
Host:foas.aliyuncs.com
Content-Type:application/json
```

正常返回示例

XML 格式

```
HTTP/1.1 200 OK
Content-Type:application/xml
<UpdateProjectConfigResponse>
  <Data>true</Data>
  <RequestId>2E62CA21-83FE-4CE1-B7E2-E868F7272E6B</RequestId>
</UpdateProjectConfigResponse>
```

JSON 格式

```
HTTP/1.1 200 OK
Content-Type:application/json
{
  "Data" : true,
  "RequestId" : "2E62CA21-83FE-4CE1-B7E2-E868F7272E6B"
}
```

错误码

访问[错误中心](#)查看更多错误码。

9. 常见问题

9.1. 计量计费

本章节汇总了使用实时计算过程中计量计费的常见问题。

- 为什么计算的CU跟显示消耗的CU不一致？
- 阿里云实时计算如何计量和收费？
- 阿里云实时计算是否带来其他费用？
- 为什么已使用CU和资源消耗总和不一致？
- 实时计算到期不续费，作业信息是否会被保留？
- 按量付费和包年包月两种付费模式可以相互转换吗？
- 已经完成购买的独享集群如何进行资源配置变更？
- 非高可用集群如何获得高可用性保障SLA服务？
- 初次购买的独享集群中Slave型号为8核32g，能否将Slave型号升配为16核64g或降配为4核16g？

为什么计算的CU跟显示消耗的CU不一致？

配置界面配置的资源只是单个Operator的计算资源，不包括框架额外消耗的资源。所以，Operator消耗CU数值相加，和最后运行消耗的CU数值不相同。

框架额外消耗的资源包括但不限于：

- 网络资源。
- 为Operator优化所消耗的其他资源。
- 各个JOIN、GROUP BY等节点分配的state_size所消耗的资源。

```
{
  "id": 17,
  "uid": "17",
  "pact": "Operator",
  "name": "JoinTable(...)",
  "parallelism": 16,
  "maxParallelism": 32768,
  "vcore": 0.02,
  "heap_memory": 8,
  "native_memory": 2,
  "direct_memory": 0,
  "state_size": 1,
  "chainingStrategy": "ALWAYS"
},
```

② 说明 由于state_size=1，需要额外的堆外内存来进行缓存（需要运行的时候才知道缓存的量），这些资源无法体现到资源配置中。

阿里云实时计算如何计量和收费？

实时计算的基本计量单位为CU（Compute Unit），即计算资源。一个CU对应于实时计算底层系统是一个CPU的计算能力。实时计算底层使用虚拟化技术进行资源隔离，保证一个基本的CU消费且最大消费仅能为一个CPU的计算能力。

一个CU描述了一个实时计算作业最小运行能力，即在限定的CPU、内存、I/O情况下对于事件流处理的能力。一个实时计算作业可以指定在1个或多个CU上运行。

当前实时计算定义1CU的处理能力约为1000条数据/秒。

实时计算支持包年包月购买，详情请参见 [开通服务和创建项目](#)。

说明 一个CU一天的单价是人民币6元，您可以根据业务以月或者年为单位选择购买的时长。例如：购买了10个CU、购买时长是一个月30天，则购买的费用为 $6*10*30=1800$ RMB。

阿里云实时计算是否带来其他费用？

阿里云实时计算独立于其中读取数据的流源和向其中写入处理数据的目标。因此您需要分别为输入和输出流相关的上下游付费。例如，您向LogHub写入数据，需要提供相应的存储费用。请查看所涉及对应存储产品的收费细则。

为什么已使用CU和资源消耗总和不一致？

● 问题描述：

为什么总览界面上已使用CU与项目中所有作业的资源消耗总和不一致？如下图，已使用CU为22.93CU，而所有作业资源消耗总和为24.11CU。



● 解释说明：

作业资源计算方法和已使用CU计算方法不同，导致了二者的显示结果不同。

- 作业资源计算方法：

CU是真实物理资源的一个抽象， $1\text{CU}=1\text{核CPU}+4\text{G MEM}$ （即内存）。一个作业真实消耗的CU数= MAX （作业申请的CPU，作业申请的MEM/4G）。例如，一个作业申请了1核CPU+5G MEM，那么实际消耗资源为1.25CU。

- 已使用CU计算方法：

已使用CU= MAX （所有作业申请的CPU总和，所有作业申请的MEM总和/4G）。

- 示例说明：

例如，1个项目有2个作业A和B。作业A申请的资源为（1核CPU，6G MEM）。作业B申请的资源为（1核CPU，2G MEM）。

- 作业实际消耗资源总和为2.5CU。单作业实际资源消耗如下：

- 作业A实际消耗资源为1.5CU。
- 作业B实际消耗资源为1CU。

- 整个项目实际的资源申请数为（2核CPU，8GMEM），即项目的已使用CU=2CU。

因此，项目总览界面上已使用CU（2CU）与项目中所有作业的资源消耗总和（2.5CU）有可能不一致。

实时计算到期不续费，作业信息是否会被保留？

实时计算到期释放后，不会保留数据。如果您再次购买实时计算服务后新建项目，可以导入原有项目的作业。

按量付费和包年包月两种付费模式可以相互转换吗？

按量付费和包年包月两种付费模式不可相互转换。

已经完成购买的独享集群如何进行资源配置变更？

已购买独享集群仅支持Master升配和Slave扩容或缩容（Slave数量的增减）。

非高可用集群如何获得高可用性保障SLA服务？

非高可用集群（一台Master）和高可用集群（三台Master）不能够进行相互转换。仅高可用集群（三台Master）支持高可用性保障SLA服务。如果您购买的是非高可用集群（一台Master），需要重新购买高可用集群（三台Master）后，才可以获得高可用性保障SLA服务。

初次购买的独享集群中Slave型号为8核32g，能否将Slave型号升配为16核64g或降配为4核16g？

无法对已购买的独享集群中Slave的型号进行变更，只能对Slave的数量进行调整。需要重新购买集群实现对Slave型号的升降配需求。

9.2. 基本概念

本文汇总了实时计算Flink版基本概念、作业操作、调试运维等方面的常见问题。

基本概念

- 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版适用哪些场景？
- 阿里云实时计算Flink版使用有哪些限制？
- 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？

作业操作

- 实时计算Flink版的全链路示意图是什么样的？
- 实时计算Flink产品全链路使用流程是怎样的？
- 单个阿里云实时计算Flink版作业包含几个部分？
- 实时计算Flink版数据的状态保存时间是多久？
- 如何加载数据到实时计算Flink版？
- 实时计算Flink版处理后的结果数据如何继续应用？
- 实时计算Flink版中一个作业里可以编写多个INSERT语句、插入到多个数据结果表吗？

调试与运维

- 开发调试和生产运维区别是什么？
- 作业暂停恢复和停止重启的区别是什么？
- 什么是启动位点？

其他

- 阿里云实时计算Flink版提供的编程接口是什么？
- 实时计算Flink版SQL作业支持全局ORDER BY吗？
- 无窗口和窗口这两类SQL的业务含义是什么？

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（Continuous）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是基于批量数据处理模型：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。



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计算SQL后再次提交作业，您甚至可以使用AdHoc查询做到即时修改即时查询。

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。

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版适用哪些场景？

阿里云实时计算Flink版提供标准的Flink SQL语义可以协助您完成流式计算逻辑的处理，但无法满足某些特定场景的业务需求。阿里云实时计算Flink版为部分授信用户提供全功能的UDF函数，帮助授信用户完成业务定制化的数据处理需求。所以，在流数据分析领域，您可以直接使用Flink SQL+UDF即可完成大部分流式数据分析处理逻辑。


阿里云实时计算Flink版优势和限制如下：

- 阿里云实时计算Flink版更擅长于进行流式数据分析、统计和处理。
- 阿里云实时计算Flink版不适合于SQL不能够解决的领域。例如，复杂的迭代数据处理和复杂的规则引擎告警。

阿里云实时计算Flink版使用有哪些限制？

请您评估以下限制对您业务的影响，并做相应的调整。

- 阿里云实时计算Flink版购买指导，请参见 [如何购买](#)。
- 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版支持华东、华北、华南地域，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Project下属的Task、Task版本、IDE打开Task页面均有不同限制，包括：
 - 一个阿里云账号可以购买多个Project项目。
 - 单个Project下允许最多创建100个job。
 - 单个Project下最多允许50个文件夹，深度最大不超过5层。
 - 单个Project下最多允许50个UDF/JAR。
 - 单个Project下最多允许50个数据存储注册。
 - 单个Task最多允许20个历史版本保存数。

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。

实时计算Flink版的全链路示意图是什么样的？

不同于现有的离线或批量计算模型，实时计算Flink版全链路整体上更加强调数据的实时性，包括数据的实时采集、实时计算和实时集成。三大类数据的实时处理逻辑在全链路上保证了流式计算的低时延。全链路实时计算Flink版示意图如下。



1. 数据采集

使用流式数据采集工具将数据实时地采集并传输到大数据消息Pub/Sub系统，该系统将为下游实时计算Flink版提供源源不断的事件源去触发流式计算作业的运行。

2. 数据触发

流数据作为实时计算Flink版的触发源，驱动实时计算Flink版运行。因此，一个实时计算Flink版作业必须至少使用一个流数据作为源。进入的数据流将直接触发实时计算Flink版进行计算。

3. 数据集成

实时计算Flink版将计算的结果数据直接写入目的数据存储，这其中包括多种数据存储系统、消息投递系统，甚至直接对接业务规则告警系统以发出告警信息。不同于批量计算（阿里云MaxCompute或开源Hadoop），实时计算Flink版天生自带数据集成模块，可以将结果数据直接写入到目的数据存储。

4. 数据消费

一旦结果数据从实时计算Flink版投递到目的数据源后，数据消费就和实时计算Flink版完全解耦。您可以使用数据存储系统访问数据，使用消息投递系统进行信息接收，或者直接使用告警系统进行告警。

实时计算Flink产品全链路使用流程是怎样的？

1. 验证网络连通性。

使用网络探测功能验证，详情请参见 [存储注册方式](#) 中的网络探测。

2. 引用外部存储。

详情请参见 [概述](#) 文档引用外部存储。

 说明 如果不支持注册存储功能，请使用 [明文方式](#)。

3. 编写源表、结果表、维表（可选）的DDL和DML语句。

```
CREATE TABLE test_input (  
  id BIGINT,  
  `name` VARCHAR,  
  score INT,  
  `time` BIGINT,  
  tbirus AS TO_TIMESTAMP(`time`/1000), --使用计算列，将16位时间戳转换为13位时间戳。  
  WATERMARK FOR ts AS WITHOFFSET(ts, 1000) --使用窗口，其中CEP、OVER窗口需要声明。  
)  
with(  
  type='datahub',  
  ...  
);  
CREATE TABLE test_output (  
  window_start TIMESTAMP,  
  window_end TIMESTAMP,  
  ct BIGINT  
)  
with(  
  type='rds',  
  ...  
);  
--简单的滚动窗口示例。  
INSERT INTO  
  test_output  
SELECT  
  TUMBLE_START(ts, INTERVAL '1' MINUTE),  
  TUMBLE_END(ts, INTERVAL '1' MINUTE),  
  count(*)  
FROM test_input  
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE);
```

4. 语法检查。

详情请参见[开发](#)。

5. 调试。

- 调试结果仅供展示，不输出至结果表。
- 部分外部存储支持数据源表抽取功能。例如，DataHub源表、TableStore源表。如果不支持数据源表抽取功能，请下载模板自行上传数据完成调试。
- Kafka中VARBINARY类型不支持调试。
- 部分函数例如EMIT语法不支持调试。
- 调试输出结果中，同时会显示过程数据。

6. 上线运行。

详情请参见[如何利用报错信息快速定位作业问题?](#)

7. 调优。

- 优化SQL逻辑。

详情请参见[高性能Flink SQL优化技巧](#)。

- 添加WITH参数。
- 添加作业参数。

详情请参见：

- [高性能Flink SQL优化技巧](#)。
- [手动配置调优](#)。

- 资源调优。

详情请参见：

- [AutoConf自动配置调优](#)（实时计算Flink版1.0和2.0版本支持）。
- [AutoScale自动配置调优](#)（实时计算Flink版3.2.2及以上版本支持该功能）。
- [手动配置调优](#)。

单个阿里云实时计算Flink版作业包含几个部分？

- SQL代码：SQL代码是计算作业的核心业务逻辑，其中包括输入表DDL声明（流式输入表、静态输入表）、输出表DDL声明，以及执行业务逻辑的DML语句。
- 参数：参数用来描述作业运行时指标。例如并发量、批处理数据量等信息。
- 属性：作业的业务信息。例如创建人、创建时间等相关记录。
- 资源配置：配置作业的资源。

实时计算Flink版数据的状态保存时间是多久？

实时计算Flink版数据的状态默认保存36小时，如果某个状态超过36小时没有更新，之前的状态可能会被清除。状态设置步骤如下：

1. 登录[实时计算控制台](#)。
2. 单击页面顶部的[开发](#)。

3. 在作业开发区域，双击目标文件夹或目标作业名，进入作业编辑页面。
4. 在作业编辑区域右侧导航栏，单击作业参数。
5. 在作业参数页面输入状态配置参数，示例如下。

重要

- 状态（state）的生命周期（ttl）设置过小，会导致retract时产生NullPointerException的报错。
- 状态（state）的生命周期（ttl）设置过大，会导致子线资源的消耗增大。

- 实时计算Flink版1.0版本。

```
#使用rocksdb作为statebackend。  
state.backend.type=rocksdb  
#rocksdb的数据生命周期，单位为毫秒。  
state.backend.rocksdb.ttl.ms=129600000
```

- 实时计算Flink版2.0及以上版本。

```
#使用niagara作为statebackend。  
state.backend.type=niagara  
#niagara的数据生命周期，单位为毫秒。  
state.backend.niagara.ttl.ms=129600000
```

- 实时计算Flink版3.0及以上版本。

```
#使用Gemini作为statebackend。。  
state.backend.type=gemini  
#gemini的数据生命周期，单位为毫秒。  
state.backend.gemini.ttl.ms=129600000
```

如何加载数据到实时计算Flink版？

实时计算Flink版支持以下两种加载数据的方式：

- 流式数据输入：触发实时计算Flink版进行数据处理，推动实时计算Flink版持续进行数据计算。每个实时计算作业必须至少声明一个流式数据输入源。
- 静态存储：静态数据输入通常是维表，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版提供了数据关联查询。

实时计算Flink版处理后的结果数据如何继续应用？

实时计算Flink版将计算的结果数据输出到目的数据表，并且为下游数据继续应用提供各类读写接口。

实时计算Flink版中一个作业里可以编写多个INSERT语句、插入到多个数据结果表吗？

- 可以编写存在多个结果表。使用多个INSERT（DML）语句的作业代码，分别插入数据结果表。
- 可以编写存在多个源表的作业代码。多数据源表可以进行双流JOIN、UNION等操作。

重要 不建议将完全不相关的链路写在同一个作业中。如果某条链路出现问题，则会影响整个作业，其他不相关的链路也会受到影响。

开发调试和生产运维区别是什么？

开发调试和生产运维主要有以下两点区别：

- 开发调试对线上作业和数据存储不造成任何影响。

开发调试发生在调试环境，所有的Flink SQL运行将在独立的调试容器运行，且所有的输出将被直接改写到调试结果屏幕，因此不会对线上生产的实时计算Flink版作业、线上生产的数据存储系统造成任何影响。

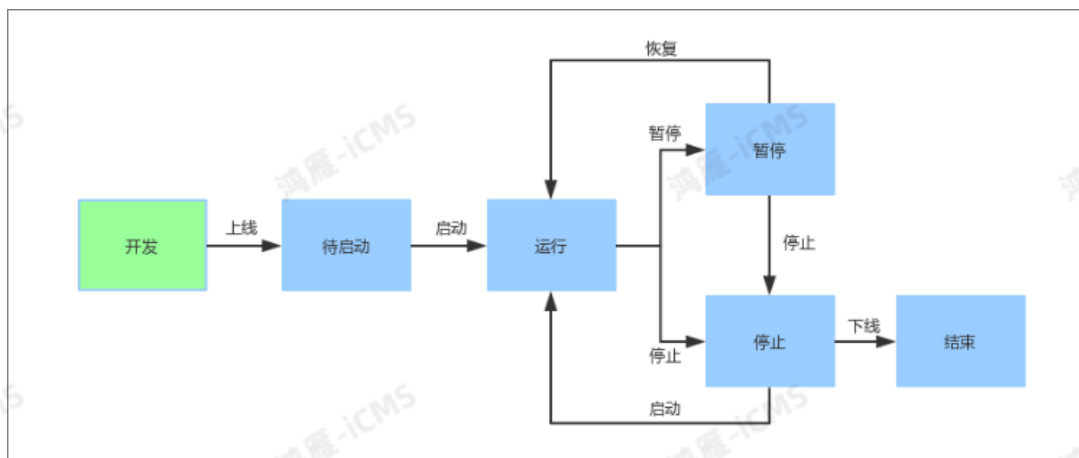
- 开发调试过程无法完全检测部分实际生产过程中的异常。

开发调试过程实际上不是将调试数据写入到外部数据源，而是被实时计算Flink版拦截输出至调试结果页面。因此在实时计算Flink版的调试代码是在调试容器中运行的，所以线上运行过程中可能由于对目标数据源写入格式的不兼容，导致作业运行失败。这类错误在开发调试阶段无法被发现和处理，只能到线上运行后才能发现。

例如，您的结果数据输出到RDS系统，其中某些字段输出字符串数据长度大于RDS建表长度最大值，在调试环境下系统无法测试出该类问题，但实际生产运行过程中会引发异常。后续，实时计算Flink版提供将调试结果写入真实数据源的本地调试功能，有效辅助您缩短调试和生产的差距，尽可能在调试阶段发现并解决问题。

作业暂停恢复和停止重启的区别是什么？

实时计算Flink版作业生命周期状态



暂停和停止的区别：

- 停止操作会清除作业的所有状态，停止后再启动相当于全新的一个作业。
- 暂停操作会保留作业的所有状态：
 - 如果状态可以兼容，暂停后会复用暂停前的作业状态。
 - 如果状态无法兼容，则会报错无法恢复。

重要 如果无法兼容，需要停止作业后，启动作业，否则作业状态将会被清理，请谨慎操作。

暂停作业后，再恢复作业，作业状态是否兼容主要取决于实际生成的物理执行图是否一致，和具体的底层优化策略相关。暂停恢复后作业的兼容性如下表所示。

操作变更类型	暂停恢复后状态是否兼容
修改SQL	不兼容
修改资源配置	兼容
修复作业参数	<ul style="list-style-type: none"> 增删作业参数不兼容。 修改作业参数值一般情况下兼容，但改变部分 optimizer参数值，修改statebackend类型等会导致不兼容。
修改WITH参数	<ul style="list-style-type: none"> 增删WITH参数不兼容。 修改WITH参数值一般情况下兼容，但类似于维表 async true <-> false等情况会导致不兼容。
修改资源引用	通常情况下兼容，但类似于用UDF变更影响Schema或者UDAGG则不兼容。
修改作业版本	自Blink 3.2.3以后，最后一位版本号修改可兼容。不兼容其它版本切换。

操作变更生效的前提是作业重新上线，并且恢复时选择 **按最新配置恢复**。如果选择 **按之前配置恢复**，则所有操作变更均不生效，效果等同于原作业暂停恢复。

什么是启动位点？

启动位点是读取数据的时间点。启动位点的时间点以后产生的数据，都将按照时间的顺序输入至新的作业中进行计算。

阿里云实时计算Flink版提供的编程接口是什么？

阿里云实时计算Flink版提供Flink SQL编程接口编写业务逻辑，为流式数据分析定制多种数据处理函数和操作符。以Word Count为例，通过Flink SQL编写的业务逻辑如下所示。

```
--声明一个流式源表。
create table stream_source(word string);
--声明一个目标表。
create table stream_result(word string, cnt bigint);
--统计word次数。
insert into stream_result
select t.word,
       count(1)
from stream_source t
group by t.word;
```

实时计算Flink版SQL作业支持全局ORDER BY吗？

实时计算Flink版作业不支持全局ORDER BY，ORDER BY需要在OVER窗口中使用。详情请参见 [TopN语句](#)。

无窗口和窗口这两类SQL的业务含义是什么？

- 窗口函数Group Window Aggregation

Group Window Aggregation是一种在窗口上的聚合。Group Window Aggregation在每个窗口结束且无提前的观测值（early firing）时，会发出的一条结果数据，例如MicroBatch。时间窗口有以下几种：

- TUMBLE：滚动窗口
- HOP：滑动窗口
- SESSION：会话窗口

例如，统计过去的1分钟内，多少用户单击了某个网页。在这种情况下，我们可以定义一个窗口，用来收集最近一分钟内的数据，并对这个窗口内的数据进行计算。

- 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

```
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM window_input
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

● 无窗口函数Group Aggregation

Group Aggregation是一种无限流的聚合。因为没有窗口，所以每到达一条数据就会增量计算一次，并发出更新后的结果。实时计算Flink版状态保存在以下三层存储媒介：

- 第一层：内存，作为缓存来使用。
- 第二层：硬盘，实时计算使用SSD作为存储。

- 第三层：HDFS。

因此，不存在内存占满的问题。

- 测试数据

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

```
SELECT Customer,SUM(OrderPrice) FROM XXXX  
GROUP BY Customer;
```

- 测试结果

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

9.3. 如何使用

本章节汇总了实时计算使用、网络连通性、效率提升等相关的常见问题。

使用

- 如何查看作业的系统版本信息？
- 如何查看集群名称、项目名称和作业名称？
- 如何查看AccessKey ID和AccessKey Secret信息？
- 如何查看更新后的SQL是否生效？
- 如何查看维表读取数据的进度？
- 如何切换系统版本？
- 如何在SQL代码中动态获取作业参数？
- 如何处理业务数据中的特殊字符？
- 如何重新获取配置资源？
- 如何使用系统时间进行分组聚合计算？
- 重启作业时，是否能保证数据不丢失且不重复？

网络连通性

- [如何查看独享集群白名单和VPC信息？](#)
- [独享集群如何访问公网？](#)
- [如何对网络连通性问题进行排查？](#)
- [如何访问跨VPC里的存储资源？](#)

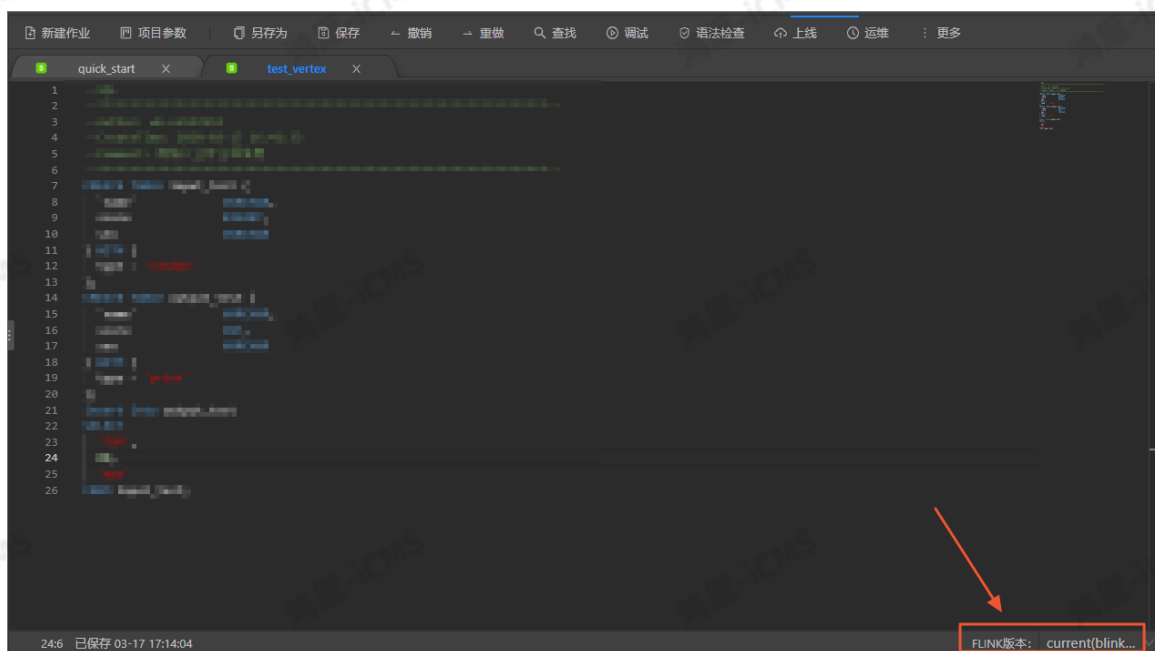
效率提升

- [如何快速编写和测试SQL？](#)
- [如何优化实时计算各个节点资源配置以减少延迟？](#)

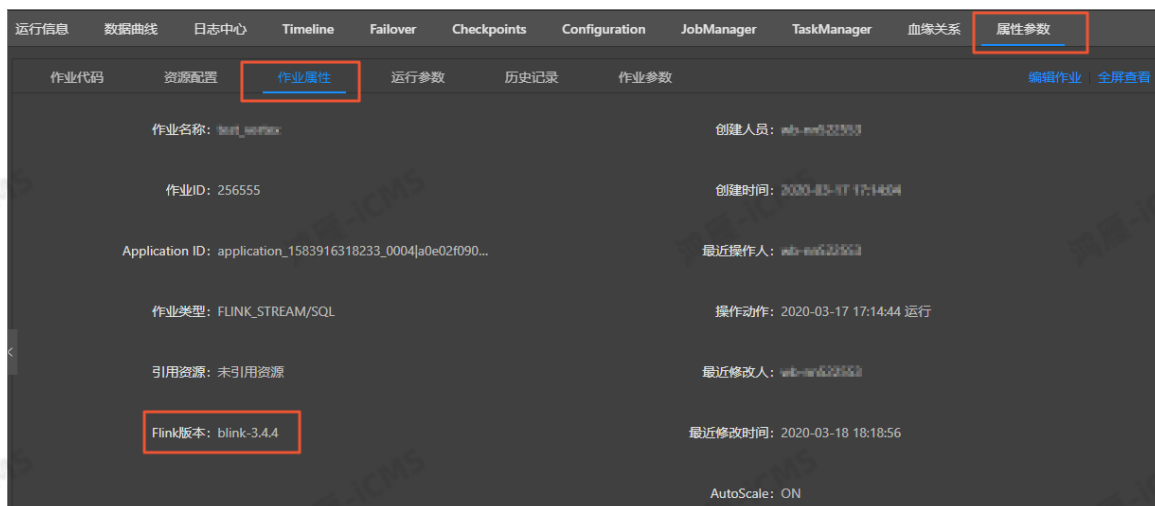
如何查看作业的系统版本信息？

实时计算支持不同的系统版本，您可以在 [实时计算管理控制台](#) 的 [开发页面](#) 或者 [运维页面](#) 查看当前作业的系统版本信息。

- 在 [开发页面](#) 查看系统版本信息
 - i. 登录 [实时计算控制台](#)。
 - ii. 单击顶部的 [开发](#)。
 - iii. 在 [开发页面](#) 的右下角的 Flink 版本，查看系统版本信息。



- 在 [运维页面](#) 查看系统版本信息
 - i. 登录 [实时计算控制台](#)。
 - ii. 单击顶部的 [运维](#)。
 - iii. 在 [运维页面](#)，单击目标作业名称。
 - iv. 在 [作业运维](#) > [属性参数](#) > [作业属性](#) > [Flink版本](#) 页面，查看系统版本信息。

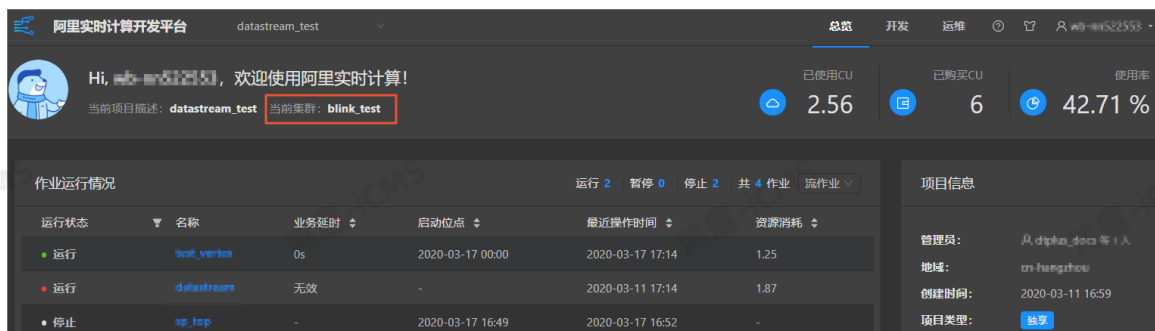


如何查看集群名称、项目名称和作业名称？

1. 登录**实时计算控制台**。
2. 将鼠标悬停至页面右上角账号信息位置，选择**项目管理**。
3. 在项目列表页面，单击目标项目名称，进入项目**概览**页面，查看以下信息：

- o 集群名称

您可以在实时计算控制台的**总览**页面查看实时计算项目的名称和项目所归属的集群名称。



- o 项目名称

您可以在实时计算控制台的**总览**页面查看实时计算项目的名称。

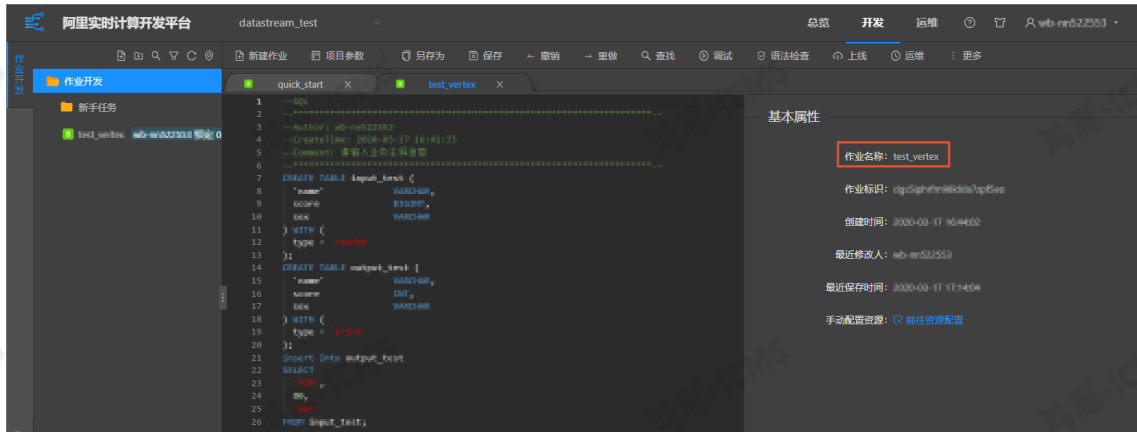


- o 作业名称

您可以在实时计算控制台的**开发**或**运维**页面查看实时计算作业的名称。

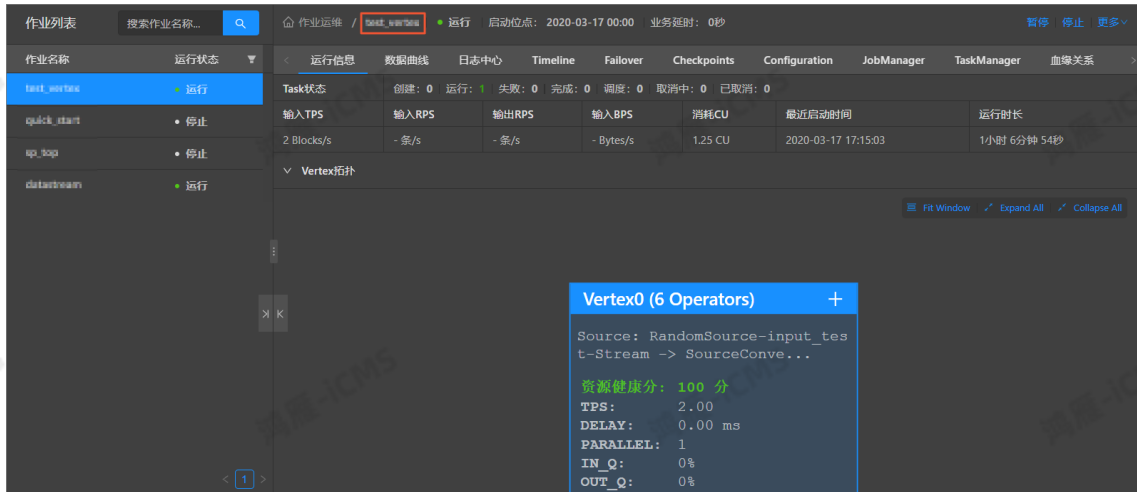
■ 在开发页面查看作业名称

您可以在作业编辑页面右侧的基本属性页面中，查看作业的名称。



■ 在运维页面查看作业名称

您可以在作业运维页面的顶部查看作业的名称。



○ 地域信息

在总览页面右侧项目信息中，查看项目地域信息。



如何查看AccessKey ID和AccessKey Secret信息？

您可以通过阿里云管理控制台获取您的密钥信息，查询步骤如下：

1. 登录 [阿里云控制台](#)。
2. 将鼠标悬停至页面右上角的头像，单击 **Accesskey 管理**。

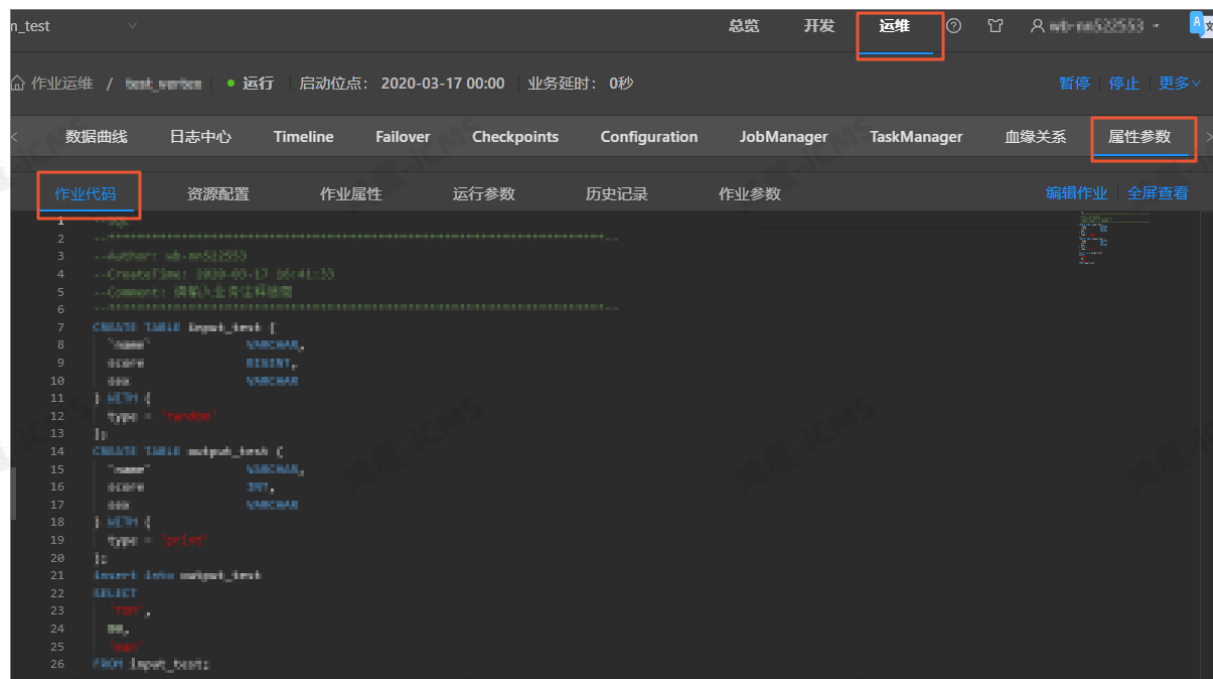


3. 在安全信息管理页面查看您的AccessKey ID和Access Key Secret信息。



如何查看更新后的SQL是否生效？

您可以在运维界面的属性参数 > 作业代码中查看更新后的SQL代码，确认您的变更是否生效。

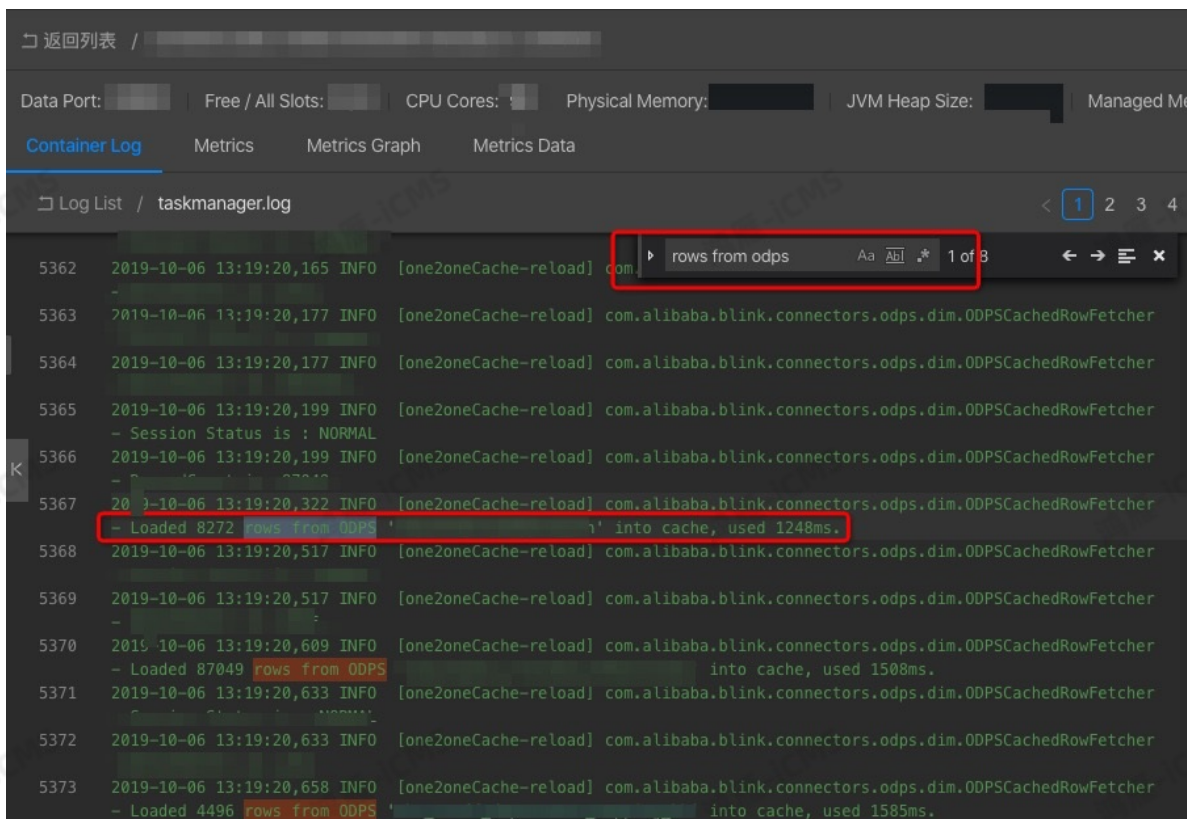


说明 查看作业SQL变更生效前，请确保您按照作业生效的完整流程完成了SQL变更：[停止作业 > 更新SQL > 上线作业 > 启动作业](#)。

如何查看维表读取数据的进度？

您可以通过实时计算控制台中的 `taskmanager.log` 日志信息，查看维表读取数据的进度。具体操作步骤如下：

1. 登录 [实时计算控制台](#)。
2. 单击顶部的 **运维**。
3. 在运维页面，单击目标作业名称。
4. 单击对应的JOIN节点拓扑图。
5. 单击SubTasks列表右侧的LOGO，进入 `taskmanager.log` 日志。
6. 按下 `Ctrl + F`，检索 `rows from <维表类型>`，例如 `rows from odps`。

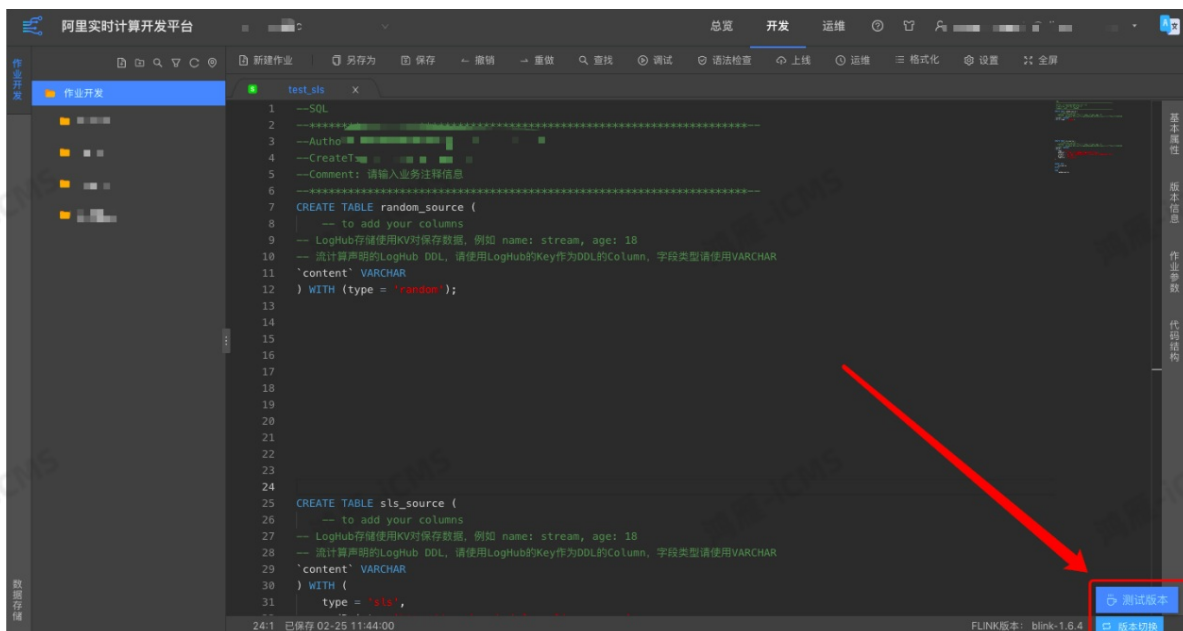


- ② 说明 如果通过以上步骤无法检索到相应的进度信息，请从以下方面进行排查：
- 维表是否具备相应的权限（例如，MaxCompute需要具备 download等权限）。
 - WITH参数是否正确。
 - taskmanager.log日志中是否存在异常信息。taskmanager.log日志中异常信息查询方法参见[如何从taskmanager.log中查找脏数据等报错信息？](#)

如何切换系统版本？

实时计算支持多种系统版本，您可以在实时计算管理控制台的 [开发](#) 页面对系统版本进行切换，操作步骤如下：

1. 登录 [实时计算控制台](#)。
2. 单击顶部的 [开发](#)。
3. 在开发页面的右下角，选择对应的版本号。



4. 单击确定，完成切换。



说明

- 如果您使用的是独享集群，且只安装了一个系统版本，您需要先安装新版本再进行切换。详情请参见[管理独享集群Blink版本](#)。
- 系统版本切换完成后，需要进行重新上线（生成新的planJSON）和启动作业操作，才能使更新后的系统版本在作业中生效。

如何在SQL代码中动态获取作业参数？

您可以对作业中指定的参数进行配置，以达到作业更新后，作业SQL代码中WITH参数信息自动同步的目的。操作步骤如下：

1. 登录实时计算控制台。
2. 单击顶部的开发。
3. 在作业编辑页面，单击右侧的作业参数。
4. 在作业参数页面，输入WITH参数的动态配置信息。

```

1 #checkpoint模式: EXACTLY_ONCE 或者 AT_LEAST_ONCE
2 #blink.checkpoint.mode=EXACTLY_ONCE
3
4 #checkpoint间隔时间, 单位毫秒
5 #blink.checkpoint.interval.ms=180000
6
7 #rocksdb的数据生命周期, 单位毫秒
8 #state.backend.rocksdb.ttl.ms=129600000
9
10 status.topic=...
11 status.accessId=...
12 status.accessKey=...
13 status.consumerGroup=...
14 status.endPoint=...

```

5. 在对应的作业编辑页面，在作业代码中，使用 ``${}`` 声明WITH参数。

```

) WITH (
  type = 'mq',
  topic = `${status.topic}`,
  pullIntervalMs = '100',
  consumerGroup = `${status.consumerGroup}`,
  endPoint = `${status.endPoint}`,
  accessId = `${status.accessId}`,
  accessKey = `${status.accessKey}`
);

```

如何处理业务数据中的特殊字符？

如果您的业务数据中存在特殊字符（即系统使用的字符或者SQL关键字），则需要使用 `REGEXP_REPLACE` 函数对特殊字符进行转义处理。详情请参见 [REGEXP_REPLACE](#)。

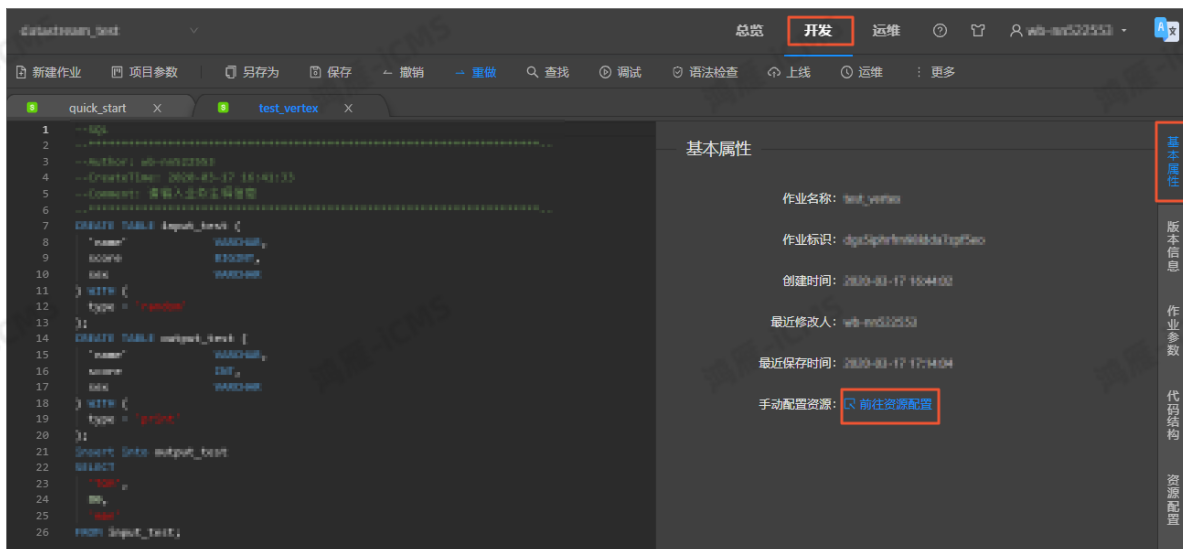
常见的特殊字符和转义方式如下表所示。

特殊字符	字符的转义方式
'	\'
\	\\
.	\\.
\$	\\\$

如何重新获取配置资源？

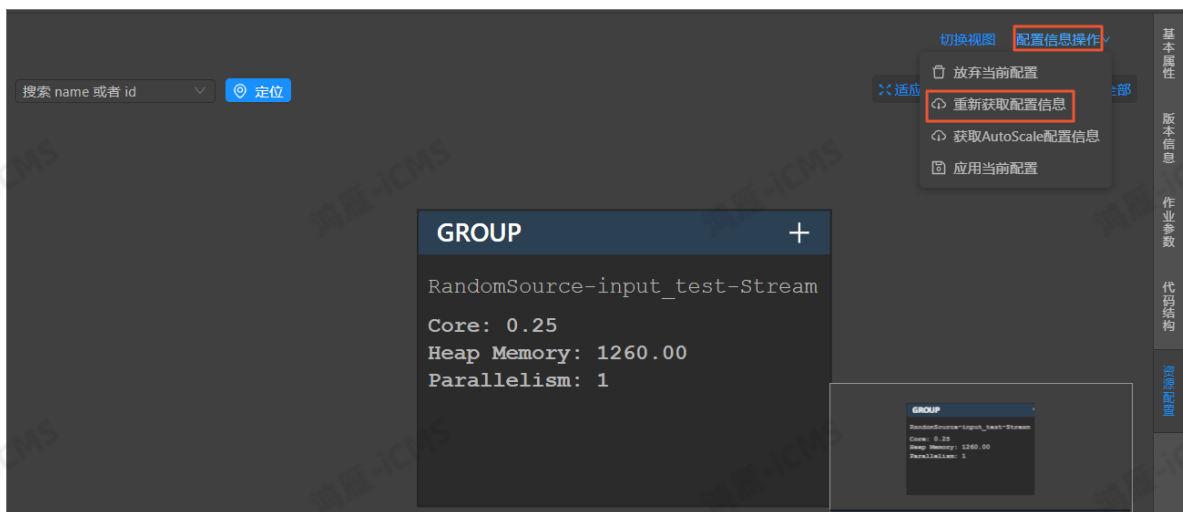
对作业的配置进行更新前，您需要进行重新获得获取配置资源的操作。操作步骤如下：

1. 登录[实时计算控制台](#)。
2. 单击页面底部的开发。
3. 在作业开发页面，单击基本属性 > 前往资源配置。



说明 如果在操作中出现报错，请参见[重新获取配置资源报错，该如何处理？](#)。

4. 单击配置信息操作 > 重新获取配置信息。



如何使用系统时间进行分组聚合计算？

如果您有使用系统时间做分组聚合计算的需求，建议您使用 `GROUP BY DATE_FORMAT(TO_TIMESTAMP(NOW()*1000), 'yyyyMMdd') as `day`` 的方式来代替 `GROUP BY CURRENT_DATE`。

因为，如果使用 `GROUP BY CURRENT_DATE` 方式进行分组聚合，运行结果可能不符合您的预期。

重启作业时，是否能保证数据不丢失且不重复？

- 如果您是手动停止作业后，再重启作业，您可以在重启设置启动位点时，调整启动时间到停止时间之前，并建议您使用带主键唯一特性的数据库做结果表，即使上游存储有部分重复消费的数据，最终结果也能接

近准确。因为实时计算独享和共享模式本身不支持Savepoint，所以手动停止并重启的位点无法做到非常精准。

- 如果系统出现故障后重启作业，您无需担心数据丢失问题。因为实时计算存在容错机制。容错机制的核心是持续创建分布式数据流及其状态的快照。当系统出现故障时，这些快照充当可以回退的一致性检查点（Checkpoint）。作业Failover会从Failover前一次成功的Checkpoint位点恢复。

如何查看独享集群白名单和VPC信息？

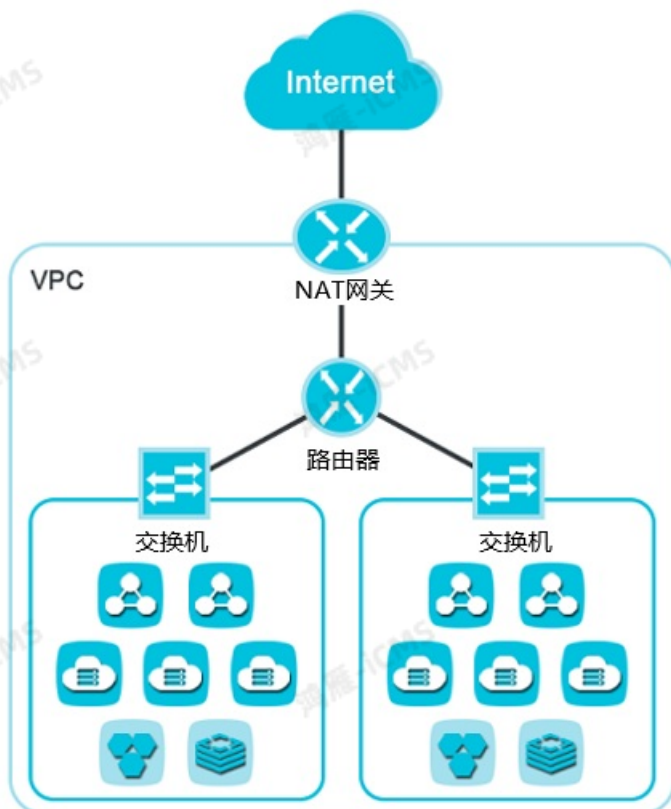
独享集群引用的外部存储必须和实时计算处于相同的VPC。如果引用的数据存储（例如RDS）需要配置白名单，则需要将ENI中的IP添加到外部数据存储的白名单中。

1. 登录[实时计算控制台](#)。
2. 将鼠标悬停至页面右上角账号信息位置，选择 **项目管理**。
3. 单击左侧导航栏中 **集群列表**。
4. 在**集群列表**页面，名称下单击目标集群名称。
5. 在**集群信息**窗口查看白名单和VPC信息。
 - **ENI**：引用的外部存储白名单中需要添加的IP地址。
 - **VPC**：集群VPC ID。

独享集群如何访问公网？

- 背景说明

独享集群默认不具备访问公网的能力，但阿里云提供的NAT网关可实现VPC网络与公网网络互通，以满足部分独享集群用户通过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（如下图所示）将独享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：

- [创建NAT网关](#)。
- [创建和管理SNAT条目](#)。
- [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：

- 退掉存储服务后，重新购买一个与实时计算相同VPC的存储资源服务。
- 释放Blink集群后，重新购买一个和存储服务相同VPC的实时计算服务。
- 开通实时计算的公网访问能力，通过公网访问存储服务。因为在延迟性方面，公网不如内网，所以不推荐使用此方式。独享集群默认不能访问公网，如有需求，请参见[独享集群如何访问公网？](#)

如何对网络连通性问题进行排查？

如果出现网络连通性问题，请先参见 [网络探测](#) 完成网络连通性的初步验证。本文为您介绍不同场景应该如何处理网络连通性问题：

- 网络连接失败
 - 未设置白名单

请参见[数据存储白名单配置](#)完成上下游数据存储设备的白名单设置。

- 白名单设置不正确

- 独享集群

- 检查相关数据存储和实时计算是否处于不同VPC。如果处于不同VPC，请使用 [高速通道](#) 等产品进行连通，或者重新购买与实时计算处于相同VPC的存储资源。

- 共享集群

- 检查Endpoint填写是否正确。共享模式访问VPC下的存储需要先进行VPC授权，详细步骤请参见 [VPC访问授权](#)。

❓ **说明** Kafka的连接信息可以进行VPC授权，但是Meta数据不能进行授权，因此共享模式不能使用Kafka作为上下游数据存储。

- 端口连接失败

- 请参见 [安全组概述](#) 完成安全组设置。

- 域名连接失败

- 如果IP连接正常，但域名连接失败，请检测域名解析是否正确。可以使用 [PrivateZone](#) 产品对域名进行解析。

如何快速编写和测试SQL？

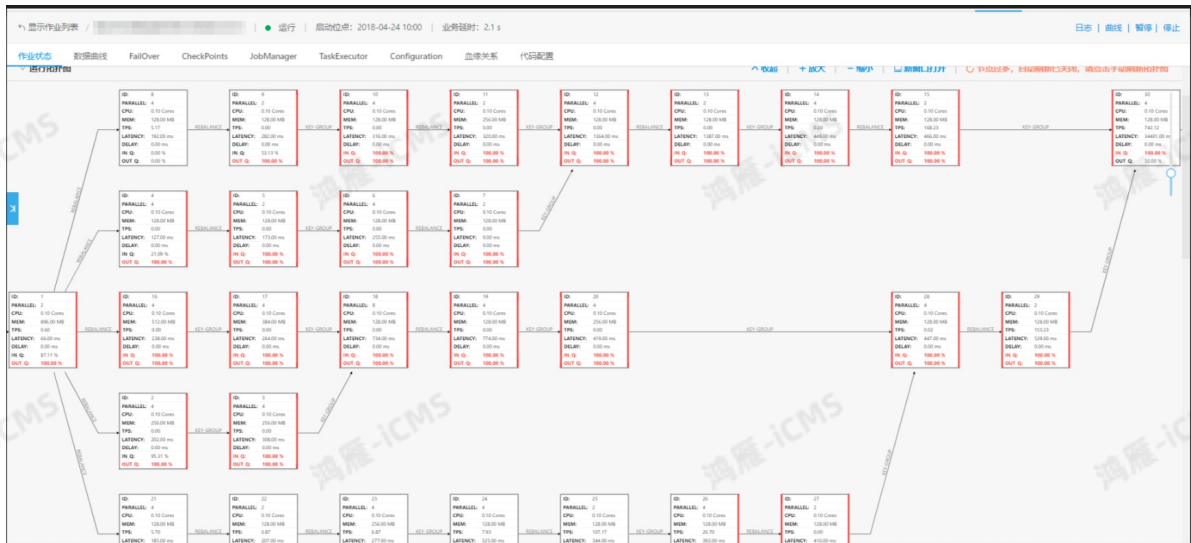
实时计算提供整套开发工具辅助您进行开发测试工作，推荐流程如下：

1. 在数据开发页面进行SQL开发，使用SQL编辑器的DDL生成、SQL智能提示、语法检测等工具实现快速开发。
2. 在数据开发的在线调试功能，进行模拟容器调试。该调试过程不会影响线上代码和数据上下游存储。
3. 将发布调试完成的代码，到生产运维试运行。经过真实业务和数据验证后，该SQL作业进入生产状态。
4. 在数据开发页面完成修改代码后，不会影响生产作业，需要您单击 [上线后](#)，重新启动该作业，该修改才会生效。

如何优化实时计算各个节点资源配置以减少延迟？

- 问题描述

实时计算作业启动后，多个节点的延迟会迅速增加。示例如下图。



- 原因分析
 - 资源不足
 - 启动作业的时候选择了追历史数据，作业将会有一定的延时。
- 解决方案
 - 资源不足：进行AutoConf自动配置调优或手动配置调优。
 - 当启动作业的时，指定读取数据时间选择当前时间之前：历史数据追取完成后，延时将恢复正常。

9.4. 分场景排错指引

本文汇总了实时计算Flink版服务和作业上线、启动、输出、延时等的常见问题。

购买

- 购买实时计算Flink版服务时，遇到Inventory check failed报错，应该如何处理？
- 控制台提示项目参数错误，项目不存在，应该如何处理？

上线启动

- 作业无法启动，应该如何处理？
- 作业上线后，作业的运维页面看不到拓扑图、瞬时值的指标或曲线指标，应该如何处理？
- 新作业提交后，项目中现存作业出现异常的原因是什么？

数据输出

- 为什么作业没有输出？
- 为什么使用滑动窗口没有数据输出？

作业延时

- 作业延时增加，应该如何处理？
- 作业延迟过大，应该如何处理？
- 业务延迟显示作业无效，应该如何处理？

问题定位思路

- 如何对实时计算Flink版3.0以上版本的作业进行反压检测？
- 如何通过调试模式（Debug）查看作业的输出信息？
- 如何处理调试过程、窗口或计算过程中的脏数据？
- 如何利用报错信息快速定位作业问题？
- 如何从taskmanager.log中查找脏数据等报错信息？
- Vertex排查数据量问题
- 重新获取配置资源报错，该如何处理？
- 在使用UDX函数时，引用JAR包内的本地文件报错
- GROUP BY数据出现热点、数据倾斜
- OpenAPI DEMO 报错
- 云监控控制台页面配置监控告警时，显示无数据，应该如何处理？

购买实时计算Flink版服务时，遇到Inventory check failed报错，应该如何处理？

- 问题场景

在购买实时计算Flink版服务时，遇到 Inventory check failed 报错。



- 问题原因
 - 您不小心删除了阿里云实时计算Flink版在RAM的角色 AliyunStreamDefaultRole。
 - 您不小心修改了 AliyunStreamDefaultRole 的授权策略且无法恢复。

- 解决方案

请您参见 [共享模式角色授权](#) 恢复 AliyunStreamDefaultRole 角色。

重要

- 删除 AliyunStreamDefaultRole 角色后，导致实时计算Flink版正在运行的作业瞬间无法读写上下游外部存储，直到后续重新初始化 AliyunStreamDefaultRole 角色后才可恢复正常。请务必确认这项操作对于线上实时计算Flink版业务是否会造成影响。
- RAM 删除角色和实时计算Flink版重新初始化操作，均需要您的主账号或者经主账号授权的 RAM 账号来完成。

控制台提示项目参数错误，项目不存在，应该如何处理？

- 问题描述

在阿里云实时计算Flink版控制台出现错误提示窗口。



- 原因分析

系统下线了已经到期但是没有续费的项目。

- 解决方案

您需要对项目进行续费，详情请参见 [手动续费](#)。

作业无法启动，应该如何处理？

您在完成作业开发上线以后，在运维页面无法启动作业。原因有以下几点：

- 作业中存在Failover

- 排错指引

查看Failover报错信息，分析Job运行异常原因，通常Failover有以下报错。

```
Slot allocation request timed out
```

- 解决方案

这个问题通常是由资源不足引起的，请先检查资源是否充足，不足请先扩容。

- 资源不足

- 排错指引

在运行信息中，查看是否有如下的报错提示。

```
Submit blink job failed, name: test_copy, errcode:30011, errmsg:code:[30011], brief info:[error occur while run app], context info:[details:[资源不足, 请手动停止任务提交, 并联系管理员扩容.(yarn resource not available, so here try to kill it to avoid long time waiting)]java.lang.Exception: shell cmd execute failed
```

- 解决方案

在已使用CU接近已购买CU的时，如果您的作业所需CU较大，会导致作业无法启动，需要您重新 [购买资源](#)。

- 获取资源配置失败

排错指引

在运行信息中，查看是否有如下的报错提示。

```
errcode:30011, errmsg:code:[30011], brief info:[error occur while run app], context info:[details:[任务提交失败, 请检查详细日志输出. (submit app failed)]java.lang.UnsupportedOperationException: Cannot set chaining strategy on Union Transformation.
```

Table with columns: 运行信息, 数据曲线, FailOver, CheckPoints, JobManager, TaskExecutor, 血缘关系, 告警设置, 属性参数. It contains a detailed stack trace for the error.

解决方案

重新获取资源配置。BlinkSQL任务开发完成后，需要单击 资源配置，通过获取自动生成JSON配置来生成一份默认的配置文件。

无作业指标

问题现象

作业运维曲线上没有任何指标，作业一直处于Created状态，也没有Failover报错。

原因分析

JM没运行起来。

解决方案

先检查资源是否足够，如果资源足够，请您手动指增加CU后，重启作业。

作业上线后，作业的运维页面看不到拓扑图、瞬时值的指标或曲线指标，应该如何处理？

- 1. 首先确认是否使用了自定义的Source，如果是自定义的Source，Source的输入曲线为空是正常的，因为实时计算Flink版无法监控到自定义的Source。
2. 请查看是否有Failover错误信息，如果有Failover错误信息，请参见 运行Failover。

3. 如果没有拓扑图和瞬时值的数据，或者没有使用自定义的插件但曲线指标不存在，请提供问题描述和作业链接联系我们。

新作业提交后，项目中现存作业出现异常的原因是什么？

独享模式采用的是CGroup（Control Group）非严格资源隔离模式，在集群资存在空闲时，单个作业（作业A）资源可以超用（即实际使用资源可超过申请资源，运维界面对应显示资源健康分小于等于20分）。如果新提交作业（作业B）申请的资源大于实际剩余资源，将要求超用资源的作业A退还部分资源。现存作业（作业A）的资源减少后可能会出现业务延迟增大等异常。

说明

- 仅独享模式存在此问题。
- 新作业提交运行后，项目使用率过高（超过95%以上）可能会出现以上问题。

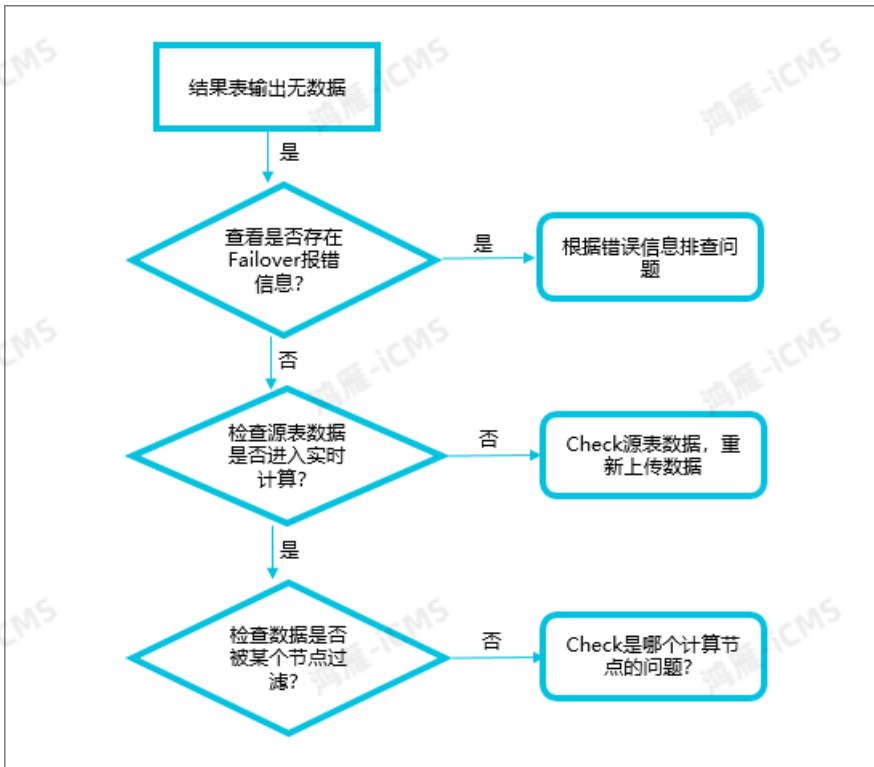
例如，1个项目的总资源为10CU，已经启动了两个作业，分别申请了2.5CU。由于资源有超用，其中一个作业A实际使用了3CU。总项目上来看，已使用CU（即实际申请CU）=5CU，但实际使用的资源为5.5CU。如果此时再提交运行1个申请5CU的作业，由于总资源为10CU，此时资源超用的作业A需要退还超用的0.5CU。因此，作业A实际使用的资源为2.5CU（与实际申请的一致），相比之前的实际使用的资源3CU有所减少，其处理能力也有所下降，可能会出现业务延迟增大的现象。

为什么作业没有输出？

场景描述

上线运行作业后，下游结果表中没有数据。

排错流程图



i. 检查作业中是否存在 Failover

- 排错指引

- 查看Failover报错信息，分析作业运行异常原因。

- 解决方案

- 解决Failover问题，使作业正常运行。

- ii. 检查源表数据是否进入实时计算Flink版

- 排错指引

- 这种情况下没有Failover，但数据延时会很大，请查看数据曲线，检查各Source输入是否有数据。

- 解决方案

- 检查源表，保证上游有数据进入实时计算Flink版。

- iii. 检查数据是否被某个节点过滤

- 排错指引

- 在运行信息页面，查看每个节点的IN_Q（输入）和OUT_Q（输出）。如果输入有数据，输出为0，说明数据被这个节点过滤了。单击对应节点vertex顶部深蓝色区域后，再单击Metrics，查看数据输入和数据输出指标的详细信息。

- 在数据曲线页面，查看数据流入流出情况。

- a. 有数据流入：检查是否使用了数据存储的方式引入。

- b. 没有数据流出：可能是启动位点设置的有误。

- 解决方案

- 请参见[Vertex排查数据量问题](#)。


- Join过滤了数据。

- Window过滤了数据（检查Watermark的字段是否有效）。

- Where条件过滤了数据。

- iv. 检查下游是否由于默认缓存机制缓存了数据

- 解决方案：排除JOB的业务逻辑异常后，调整下游存储的batchsize的大小，设置为1。

 说明 该参数可能会造成下游数据库 I/O 压力过大、存在性能瓶颈的风险。

```
with(  
  type = 'DATAHUB',  
  endpoint = 'XXXX',  
  batchsize = '1'  
)
```

- v. 检查下游RDS，是否存在死锁

- 解决方案：请参见[写MySQL（TDDL/RDS）时，出现死锁（DeadLock）](#)。

另外，您可以使用调试模式（Debug），将计算结果打印到日志中，对日志进行分析，判断无输出结果的原因。详情请参见[如何通过调试模式（Debug）查看作业的输出信息？](#)

为什么使用滑动窗口没有数据输出？

- 问题描述：

使用 **滑动窗口** 没有数据输出。

- 解决方案

- 检查源表Watermark的时间字段是不是TIMESTAMP类型，如果不是，请您使用 **计算列** 进行转换。

说明

- a. 请参见 **概述** 的时间类型中关于Watermark的说明了解更多。
- b. 请参见 **数据源表概述** 数据列章节了解如何进行转换。

- 检查数据源表的并发是不是都有数据，只要有一个并发没有数据都不会触发Watermark形成窗口输出。如下图所示。

ID	操作	InQueue	OutQueue	RecvCnt	SendCnt	TPS	Retries	Status
2	跳转TaskExecutor日志	0 (0.00%)	0 (0.00%)	0	0	0	0	RUNNING
1	跳转TaskExecutor日志	0 (0.00%)	4 (100.00%)	0	78209	0.75	0	RUNNING
0	跳转TaskExecutor日志	0 (0.00%)	3 (50.00%)	0	79062	0.90	0	RUNNING

作业延时增加，应该如何处理？

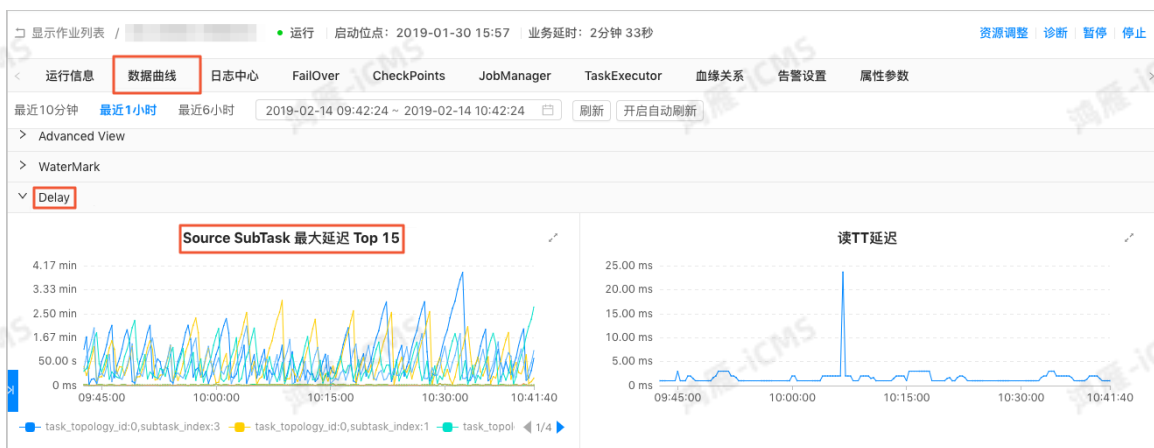
- 排查指引

○ 数据间隔时间较大

数据间隔时间： $no_data_delay = \text{当前系统时间} - \text{最后一条数据到达实时计算Flink版的时间}$ ，即数据源中的数据到达实时计算Flink版的时间间隔。



数据间隔延时过大表明上游可能没有数据进入实时计算Flink版系统。实时计算Flink版数据间隔时间等于所有并发中的最大的延迟。您可以参见运维页面数据曲线 > Delay中的Source SubTask 最大延迟 Top 15曲线分析源表各并发的延迟。



○ 作业性能不佳

SQL语法或者资源配置的不合理可能导致作业性能不佳，从而造成数据处理的滞后和作业延时的增加。

● 处理方法

- 检查作业链路，确保上游所有Shard（Queue）都有最新的数据流入实时计算Flink版。
- 如果在数据间隔时间较小的情况下，仍然存在作业延时增加的现象，则可能是作业性能不佳所导致。处理方法参见[高性能Flink SQL优化技巧](#)。SQL优化后，建议您先进行3~5次[自动配置调优](#)，如果调优效果不明显，再进行[手动配置调优](#)。

🔍 说明

- 第一次进行自动配置调优时，请不要指定CU数。
- 建议您先优化SQL，再进行资源配置，优化效率会更高。

作业延迟过大，应该如何处理？

线上作业延迟过大，表现为曲线图中业务延迟呈水平趋势，或逐步呈上升趋势，或在追历史数据（启动位点调整到当前时间之前）的过程中，曲线图中业务延迟下降非常缓慢。常见的原因有以下几种：

- 源数据中存在脏数据
 - 排错指引
 - 在曲线图中，查看各Source的脏数据，脏数据会在相应时间节点上显示数据量，没有则显示为0。
 - 查看Failover或Taskmanager.log日志中的报错信息，脏数据会在错误信息中标明。
 - 解决方案
 - 更改SQL，过滤脏数据后，重新上线作业。
- 没有源表数据进入实时计算Flink版
 - 排错指引
 - 这种情况下没有Failover，请查看数据曲线，检查各Source输入是否有数据。
 - 解决方案
 - 检查源表，确保上游有数据进入实时计算Flink版。详情请参见数据曲线业务延迟3条曲线。
- RDS写入瓶颈
 - 排错指引
 - 在RDS写入过慢时，建议您调大并发或增加资源，观察输出是否仍然为0或RPS极小。
 - 解决方案
 - 新建一个Schema相同的结果表，重新注册存储引入。
- 源数据存在问题
 - 排错指引
 - 请确认是否存在以下情况：
 - 是否存在历史数据。
 - 数据周期性输入为0。
 - 输入数据不稳定，表现为输入曲线有抖动。
 - 解决方案
 - 历史数据处理完成后（例如，追历史数据完成后），延迟会下降。
 - 确认数据生产机制是否异常。
 - 数据输入波动大，检查Source源。
- 部分节点存在反压
 - 排错指引
 - 详情请参见[如何对实时计算Flink版3.0以上版本的作业进行反压检测？](#)

- 解决方案
 - a. 优化SQL。详情请参见 [高性能Flink SQL优化技巧](#)。
 - b. 优化上下游WITH参数。详情请参见上下游存储文档的WITH参数介绍。
 - c. 优化作业参数。详情请参见 [高性能Flink SQL优化技巧](#)。
 - d. 资源调优。[AutoScale自动配置调优](#)后，再进行[手动调优](#)。

业务延迟显示作业无效，应该如何处理？

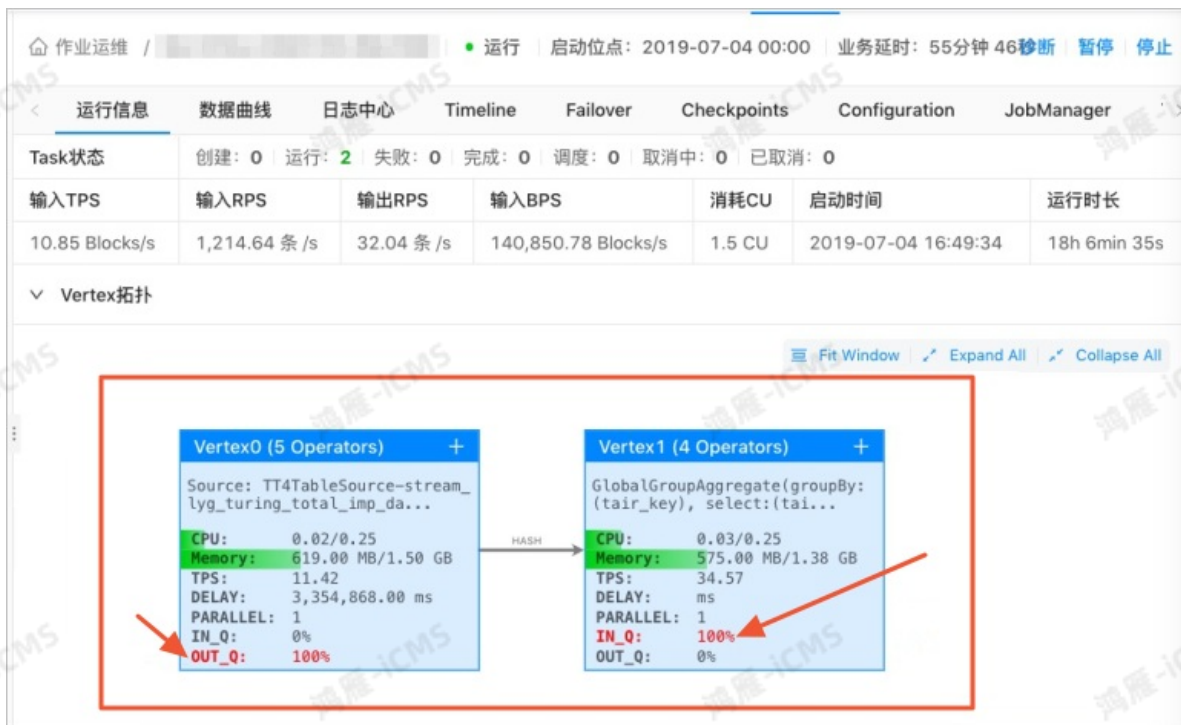
业务延迟显示无效的原因包括如下几种，您可以根据不同的原因进行对应的处理：

- 作业运行失败，无法获取稳定的埋点信息。
根据Failover报错信息提示修复问题，使作业正常运行。
- 您没有对自定义源表或Datastream作业进行埋点，或者埋点方式和实时计算Flink版数据存储不兼容。
请参见 [Metric](#) 文档编写埋点代码，确保和实时计算Flink版数据存储相互兼容。
- 用户数据的延迟和实时计算Flink版获取埋点数据的延迟重合或多个作业显示无效。
当用户数据的延迟和实时计算Flink版获取埋点数据的延迟重合或多个作业显示无效可能是集群出现问题，请联系我们。

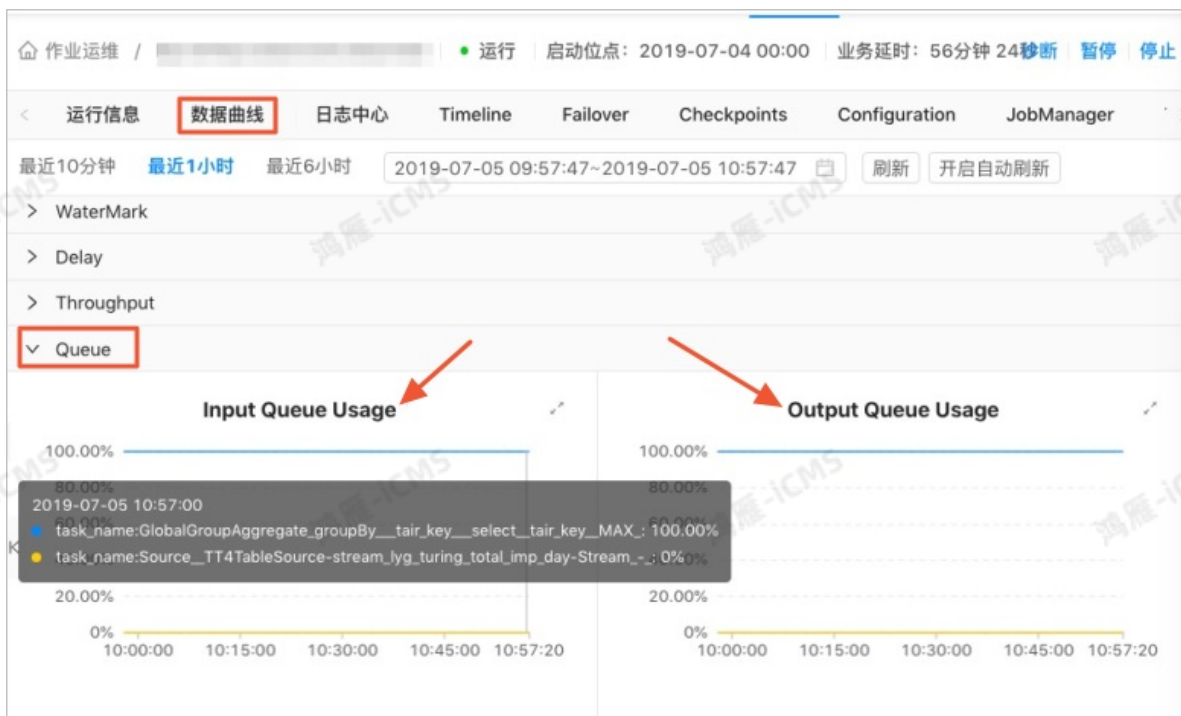
如何对实时计算Flink版3.0以上版本的作业进行反压检测？

反压检测是指当下游处理能力不足时，通知上游停止发送数据，从而避免数据丢失。实时计算Flink版3.0以上版本作业的反压检测步骤如下：

1. 查看Vertex拓扑中的IN_Q与OUT_Q的参数值是否到达100%。



- IN_Q与OUT_Q值间歇性到达100%：作业不存在反压。
 - IN_Q与OUT_Q值长时间维持在100%：进行下一步检测。
2. 查看数据曲线 > Queue中的Input Queue Usage和Output Queue Usage是否达到100%。



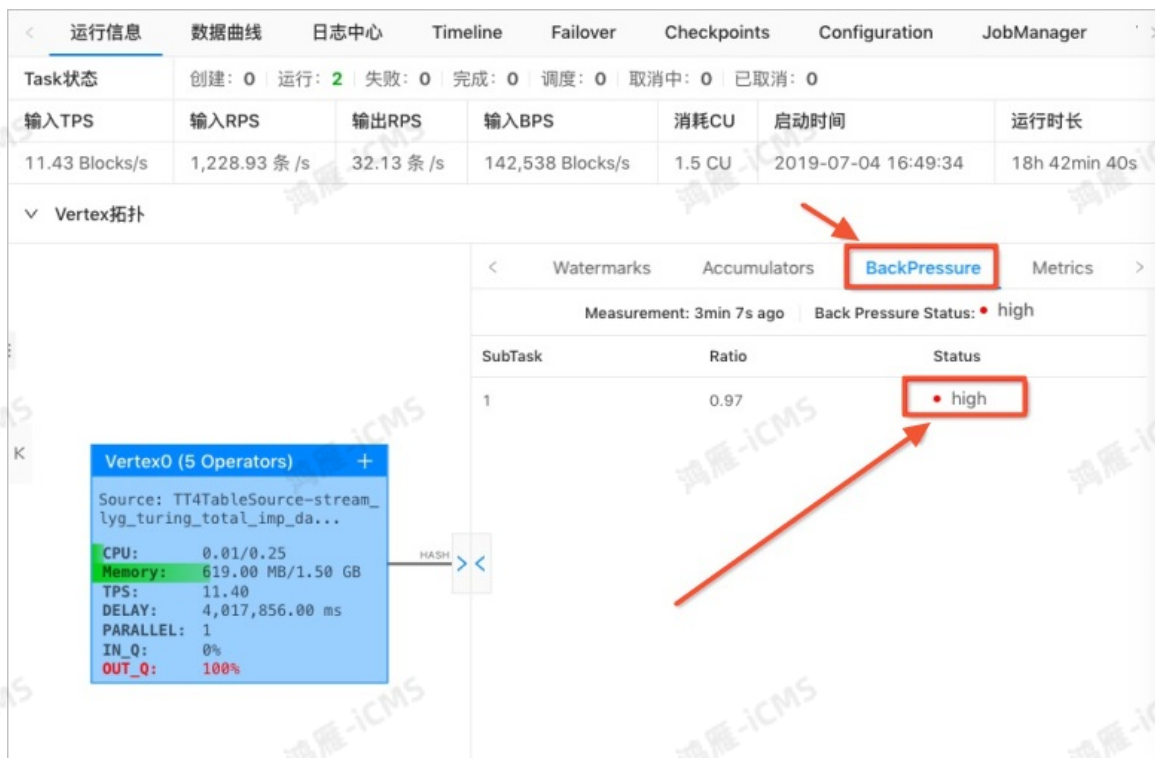
- 未达到100%：作业不存在反压。
- 达到100%：进行下一步检测。

3. 查看BackPressure的状态是否为high。

- i. 在运维 > 运行信息 > Vertex拓扑中，单击vertex顶部深蓝色区域。



ii. 在右侧Task信息区域，查看BackPressure参数。



- 红色high：该节点存在反压。
- 绿色ok：该节点不存在反压。

如何通过调试模式（Debug）查看作业的输出信息？

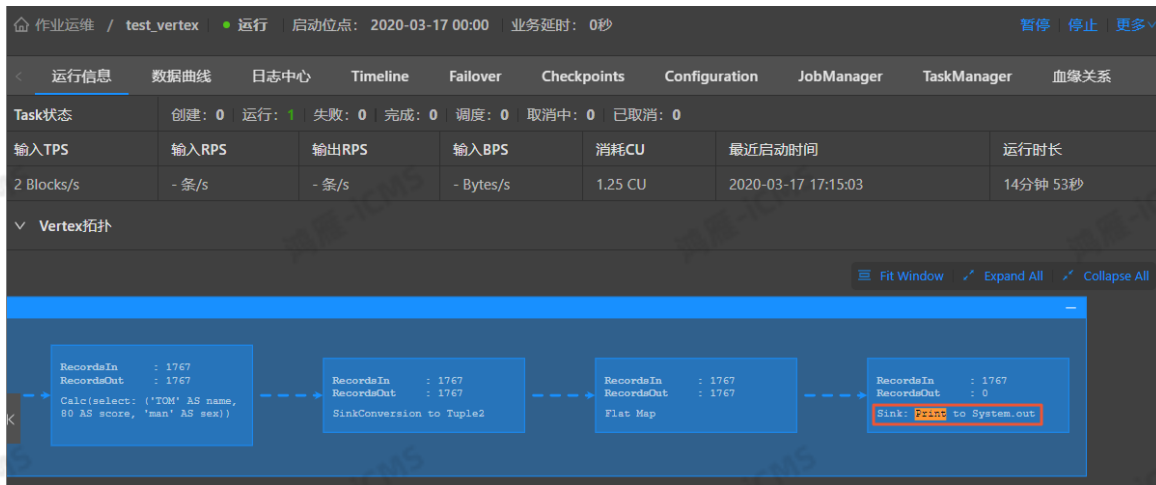
当结果表没有输出时，建议您使用调试模式（Debug），将计算结果打印到日志中，对日志进行分析，判断无输出结果的原因。例如存在脏数据等。根据作业是否包含UDX，日志查看的方法分为以下两种：

● 无UDX

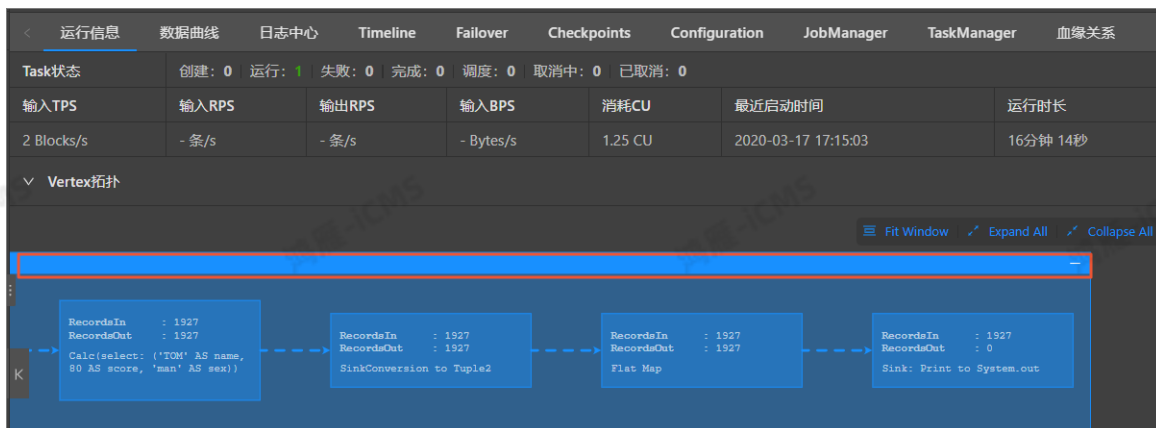
i. 将结果表中的WITH参数改为 `type='print'`。

```
CREATE table result_infor(
  id bigint,
  phoneNumber bigint,
  name VARCHAR
)with(
  type='print'
);
```

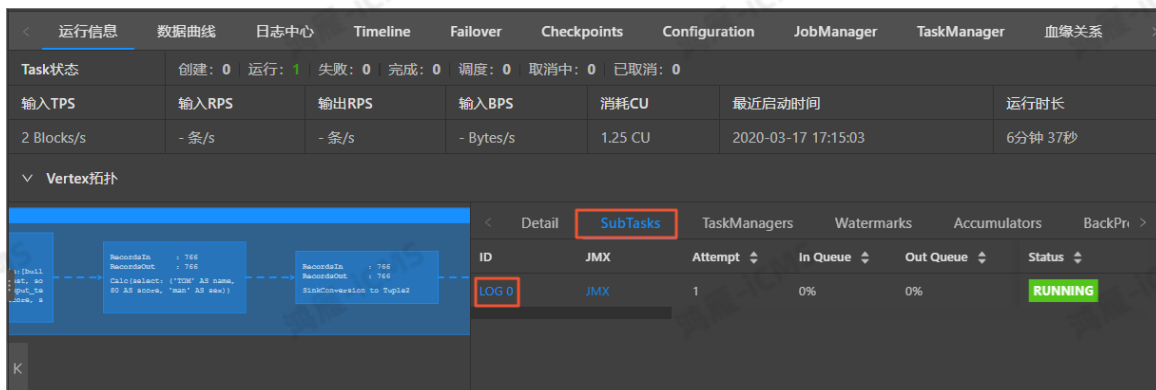
- ii. 重新上线并且启动作业。
- iii. 单击作业名称，进入运行信息页面。
- iv. 单击Expand all。
- v. 按住CTRL+F，输入print，找到Print节点所在的Vertex。



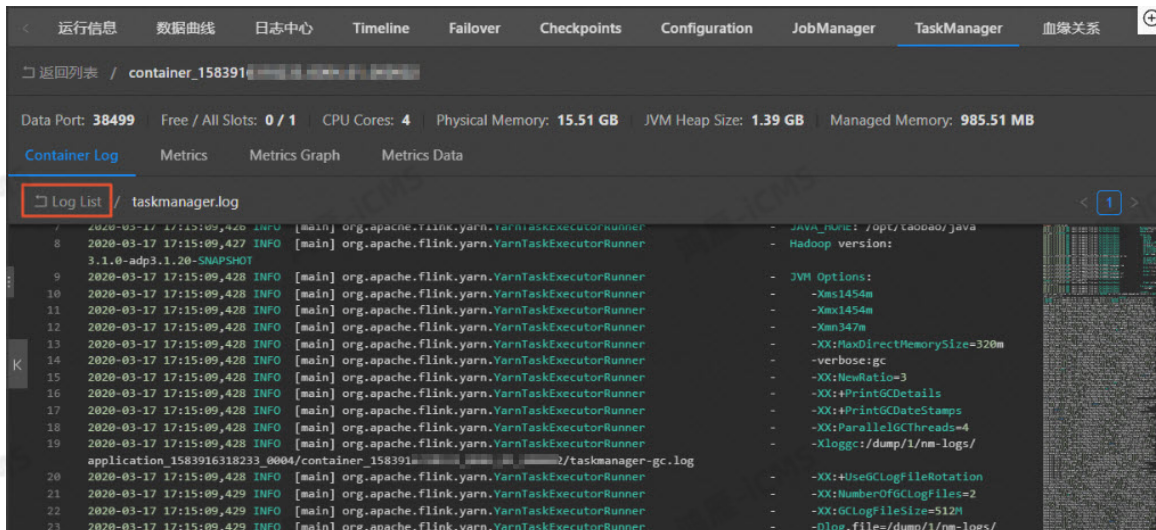
vi. 单击目标vertex顶部深蓝色区域。



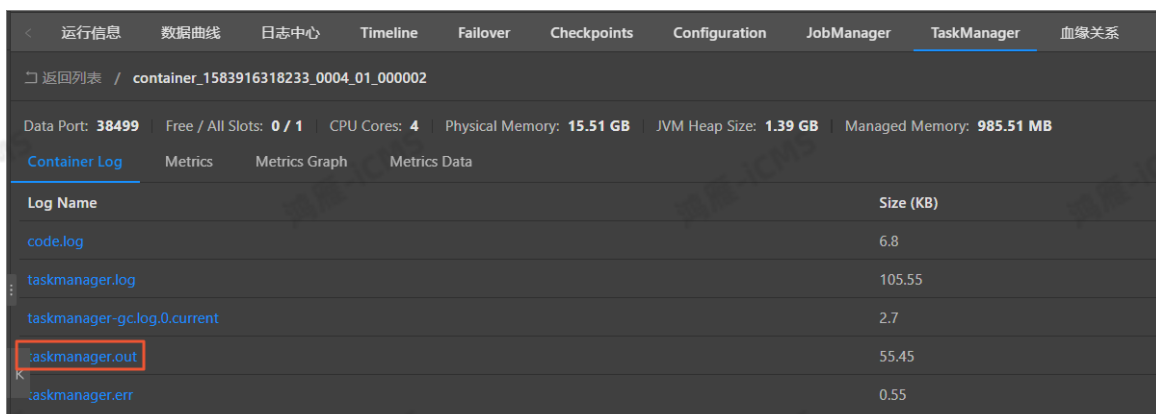
vii. 在SubTasks页签，单击LOGO，跳转至Container Log页面。



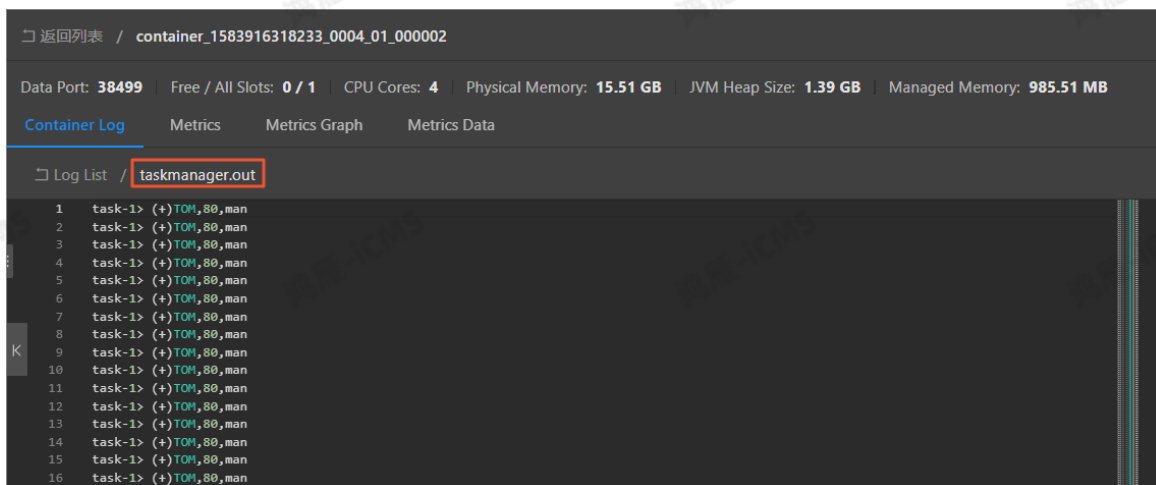
viii. 单击logList，返回日志列表。



ix. 单击taskmanager.out查看日志结果。



日志结果如下图所示。

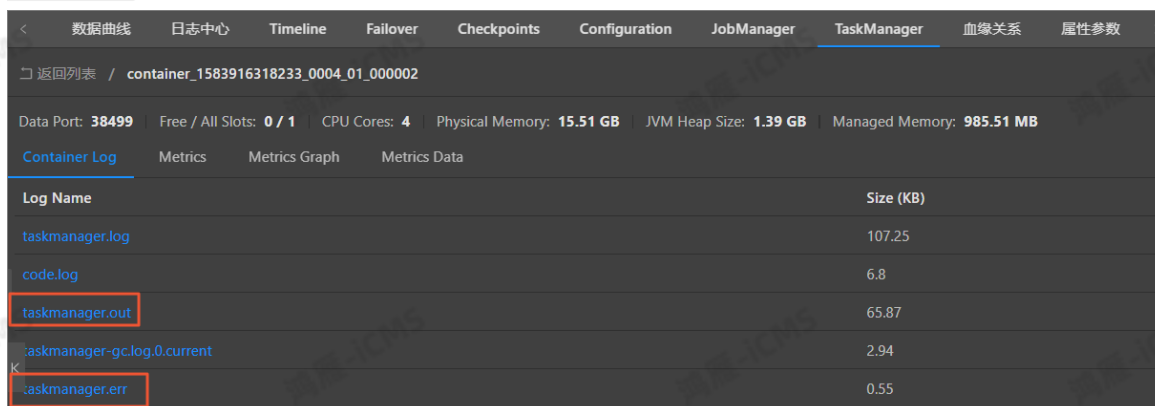


- 存在UDX

如果您使用UDX，在Java代码中又可以分为如下两种Debug方法：

o system out/err方法

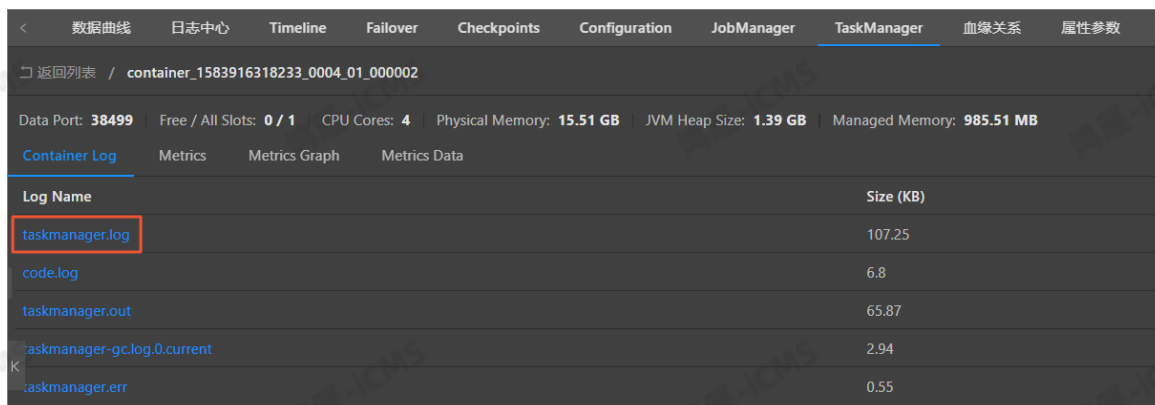
在Java代码中使用 `system.out` 或 `system.err` 方法，把调试的日志打印到 `taskmanager.out` 或 `taskmanager.err`。



Log Name	Size (KB)
taskmanager.log	107.25
code.log	6.8
taskmanager.out	65.87
taskmanager-gc.log.0.current	2.94
taskmanager.err	0.55

o SLF4J的Logger方法

在Java代码中使用SLF4J的Logger功能，把调试的日志打印到 `taskmanager.log`。



Log Name	Size (KB)
taskmanager.log	107.25
code.log	6.8
taskmanager.out	65.87
taskmanager-gc.log.0.current	2.94
taskmanager.err	0.55

如何处理调试过程、窗口或计算过程中的脏数据？

实际操作中您可能会遇到脏数据中断业务开展的情况，本文为您介绍在不同的场景中如何处理脏数据。

● 调试过程中的脏数据

o 错误详情

作业调试过程中出现报错信息：操作错误，所有的数据均被跳过，请检查DDL中的 `LENGTHCHECK`项。

o 错误原因

单行字段条数检查策略 `LENGTHCHECK`的默认值为 `NONE`，即解析出的字段数小于定义字段数时，跳过这行数据。如果您的数据源存在脏数据，不符合单行字段条数检查的默认策略，则调试线上抽取数据时会报错，导致无法抽取到线上数据。

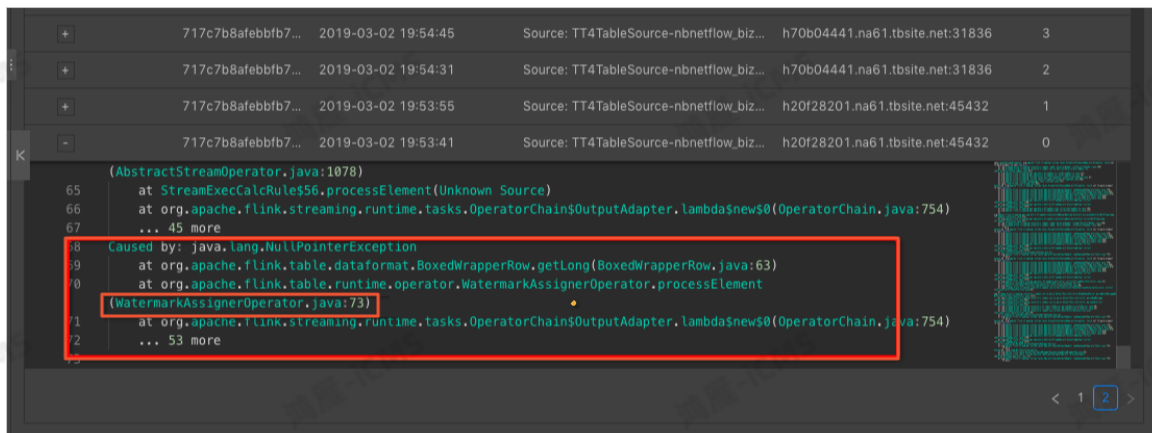
o 解决方案

建议在源表的 `WITH`参数中增加 `lengthcheck='PAD'`参数，即当解析出的字段数小于定义字段数时，系统会使用 `null`在行尾填充缺少的字段。

● 窗口中的脏数据

错误详情

报错信息： Caused by: java.lang.NullPointerException



报错原因

WaterMark中存在NULL值导致NullPointerException报错。

解决方案

如果您使用Window函数，Event time的时间字段中存在NULL值，需要使用 **计算列** 进行过滤，示例如下。

```
ts as case when `datetime` is null
then to_timestamp('1970-01-01 00:00:00')
else `datetime` end
```

计算过程中的脏数据

错误详情

java.lang.NumberFormatException: For input string:"null"

```

(StreamTask.java:298)
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:747)
    at java.lang.Thread.run(Thread.java:834)
2019-03-04 13:58:36,031 WARN [Topology-0 (5/5)]
com.alibaba.blink.streaming.connectors.common.source.parse.
DefaultSourceCollector - Data format error, field type: LongTypefield data:
null, index: 18, data: [2019-03-04 13:57:00,5060.0,0.0,0.0,7.0,0.0,0.0,0.0,
4980.0,0.0,1.0,7.5,0.0,2.0,1.0,1996213839601445,26842,cn-beijing,null,null,
null]
5 java.lang.NumberFormatException: For input string: "null"
    at java.lang.NumberFormatException.forInputString(
NumberFormatException.java:65)
    at java.lang.Long.parseLong(Long.java:589)
    at java.lang.Long.parseLong(Long.java:631)
    at
com.alibaba.blink.streaming.connectors.common.util.StringSerializer.deserialize(
StringSerializer.java:154)
    
```

java.lang.NumberFormatException: For input string:""

```

    at org.apache.flink.streaming.api.operators.ProcessOperator.processElement(ProcessOperator.java:62)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
    ... 36 more
1 Caused by: java.lang.NumberFormatException: For input string: ""
2   at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
3   at java.lang.Integer.parseInt(Integer.java:592)
4   at java.lang.Integer.valueOf(Integer.java:766)
5   at StreamExecCalcRule$715.processElement(Unknown Source)
6   at org.apache.flink.table.runtime.BaseRowProcessRunner.processElement(BaseRowProcessRunner.scala:74)
7   ... 39 more
    
```

报错原因

输入值是NULL字符串或空字符串，不能作为数字类型进行计算。

解决方案

确定脏数据后，通过WHERE或者CASE WHEN的方法对脏数据进行过滤。

说明 实时计算Flink版2.2及以上版本支持在日志中查看脏数据，具体步骤请参见 [如何从taskmanager.log中查找脏数据等报错信息？](#)

使用WHERE对脏数据进行过滤，示例如下。

```

select avg(test_id) from input
where test_id IS NOT NULL;
    
```

使用CASE WHEN对脏数据进行过滤，示例如下。

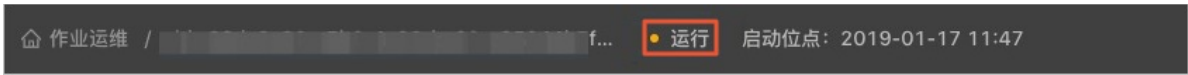
```

select avg(case when test_id IS Null
then 0
else test_id
end as test_id)
from input;
    
```

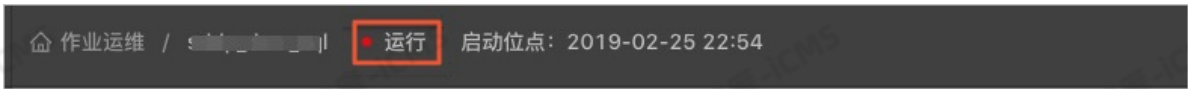
如何利用报错信息快速定位作业问题？

当作业状态运行前的标点显示为红色或者黄色时，表明作业的运行状态不正常。

- 黄色：作业正在调度。（可能因为资源不足导致作业运行失败。）

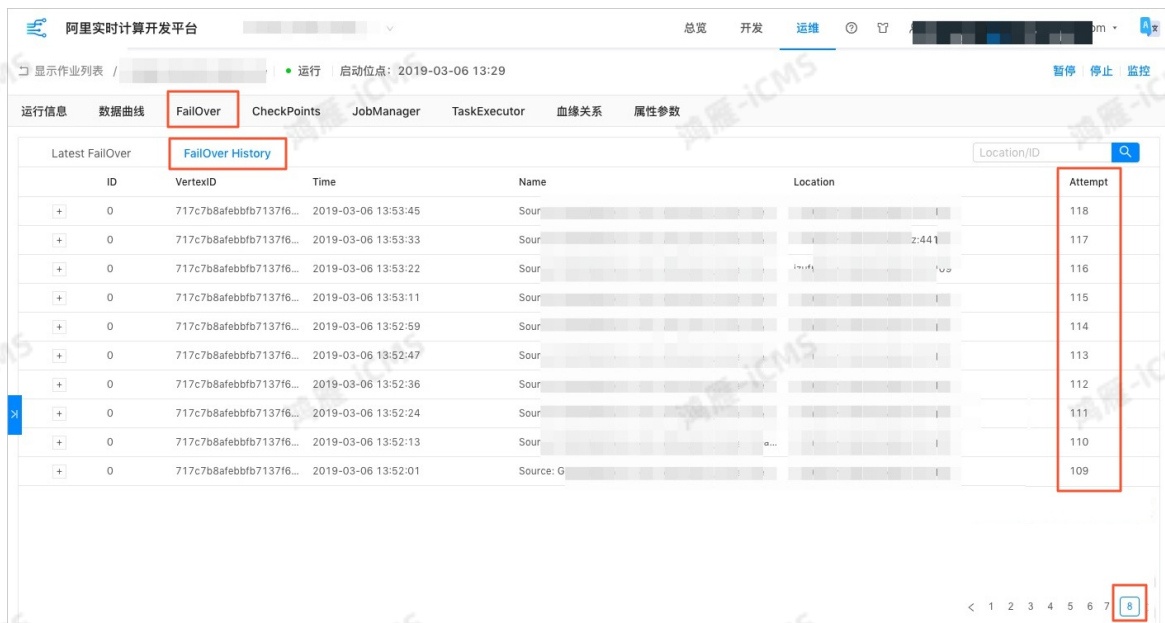


- 红色：作业存在错误。



通过以下两种方法快速定位作业问题：

- 查看Attempt为0的Failover历史信息。
 - i. 在Failover > Failover History页面，查看Attempt为0的Failover历史信息。



- ii. 查看Failover当前页面最后一个Caused by信息。找到最开始的Failover，并将页面拉至最下端，由下至上，找到Cause by信息。

```

java.lang.RuntimeException: java.lang.RuntimeException: Could not forward element to next operator
    at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:131)
    at com.alibaba.blink.streaming.connectors.common.source.AbstractParallelSource.run(AbstractParallelSource.java:120)
    at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:142)
    at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:78)
    at org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:56)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:307)
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
    at java.lang.Thread.run(Thread.java:834)
Caused by: java.lang.RuntimeException: Could not forward element to next operator
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:611)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:941)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:919)
    at org.apache.flink.streaming.api.operators.StreamSource$1.collect(StreamSource.java:127)
    at org.apache.flink.streaming.api.operators.StreamSource$1.collect(StreamSource.java:114)
    at org.apache.flink.streaming.api.operators.StreamSource$1.collect(StreamSource.java:98)
    at org.apache.flink.streaming.api.operators.StreamSourceContexts$ManualWatermarkContext.collectWithTimestamp(StreamSourceContexts.java:273)
    at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.runImpl(ParallelReader.java:223)
    at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:126)
    ... 7 more
Caused by: com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException: 900085

Column For Parser Plugin Error!
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndCollect(DefaultSourceCollector.java:366)
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndCollect(DefaultSourceCollector.java:75)
    at com.alibaba.blink.streaming.connectors.common.source.parse.SourceParser.FlatMap(SourceParser.java:47)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
    ... 16 more
Caused by: java.lang.RuntimeException: Could not forward element to next operator
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:611)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:941)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:919)
    at org.apache.flink.streaming.api.operators.TimestampedCollector.collect(TimestampedCollector.java:51)
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndMessage(DefaultSourceCollector.java:408)
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndCollect(DefaultSourceCollector.java:362)
    ... 26 more
Caused by: java.lang.NullPointerException
    at com.alibaba.blink.streaming.connectors.common.source.watermark.WatermarkUtils$BayerPunctuatedTimestampAndWatermarkWithOffset.extractTimestamp(WatermarkUtils.java:114)
    at org.apache.flink.streaming.runtime.operators.TimestampsAndPunctuatedWatermarksOperator.processElement(TimestampsAndPunctuatedWatermarksOperator.java:58)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
    ... 26 more

```

说明 第1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。

• 无法查看Attempt为0的Failover信息：

- i. 在数据曲线页面，通过最近6小时、最近1天或最近1周维度，查看Failover曲线中第一次Failover的时间点。

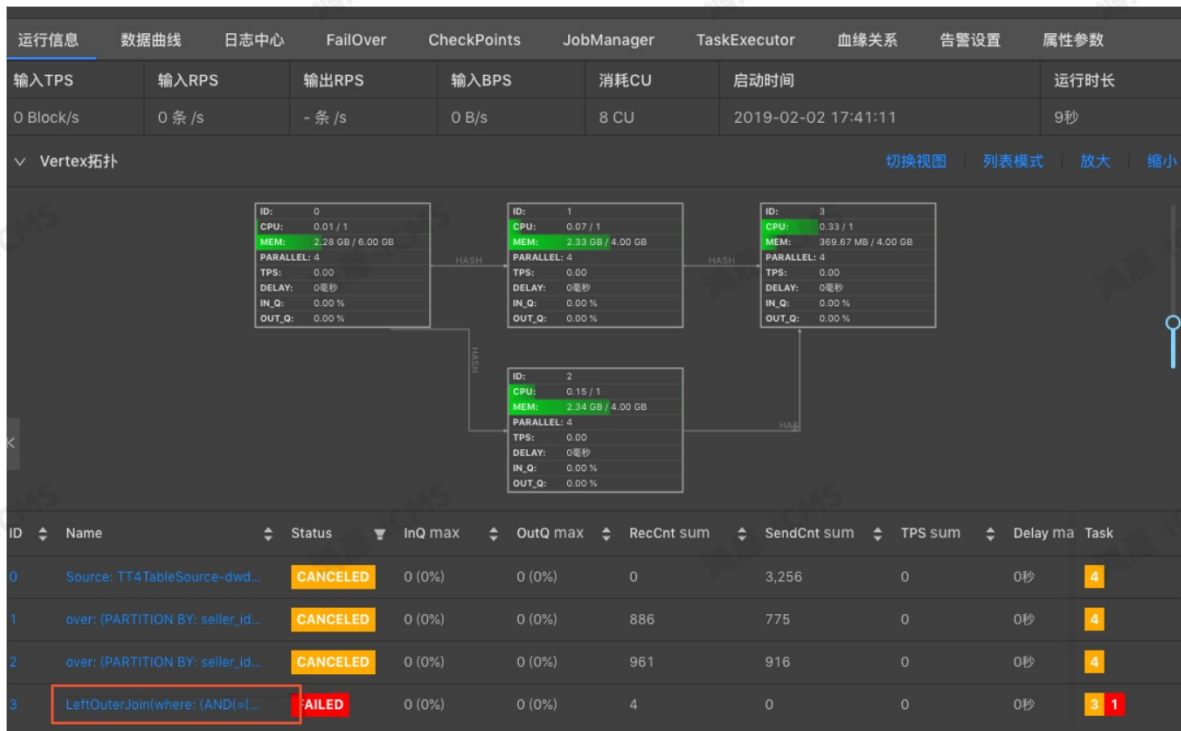


- ii. 在taskmanager.log日志中查看报错的原因。报错原因查询方法参见 [如何从taskmanager.log中查找脏数据等报错信息？](#)

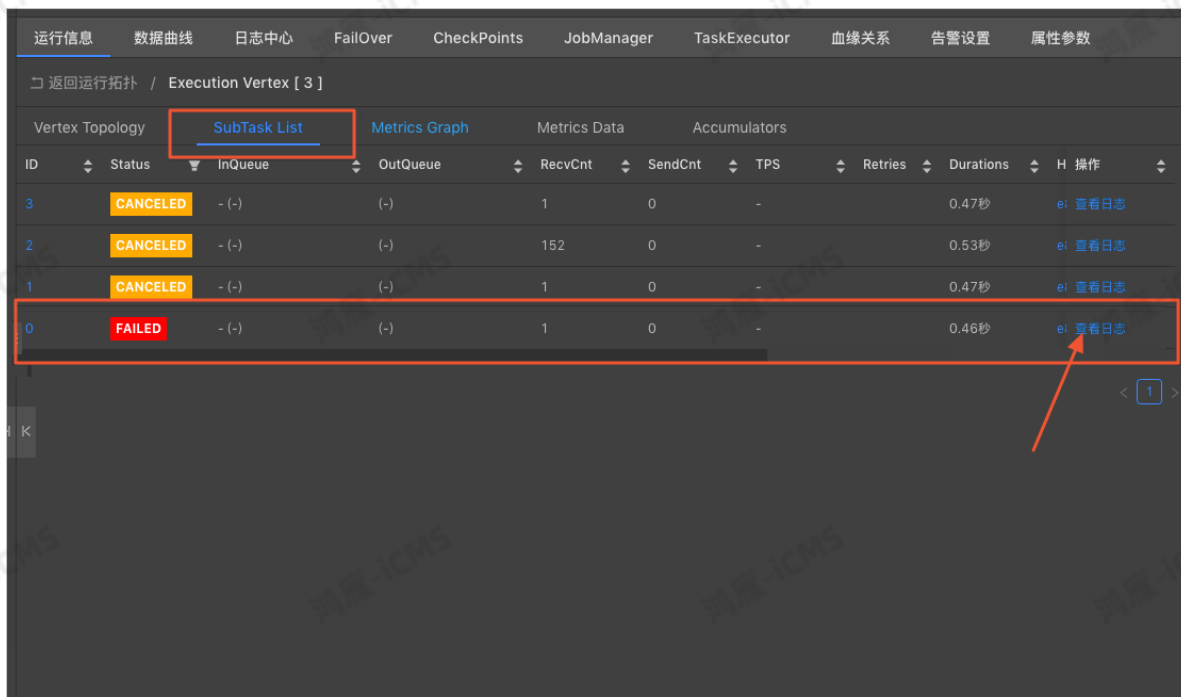
如何从taskmanager.log中查找脏数据等报错信息？

作业运行所产生的信息会记录到 taskmanager.log 日志文件中，如果您的作业出现运行失败的状态，您可以在 taskmanager.log 日志文件中查看可能的报错原因，例如脏数据。在 taskmanager.log 日志文件中查看报错信息的步骤如下：

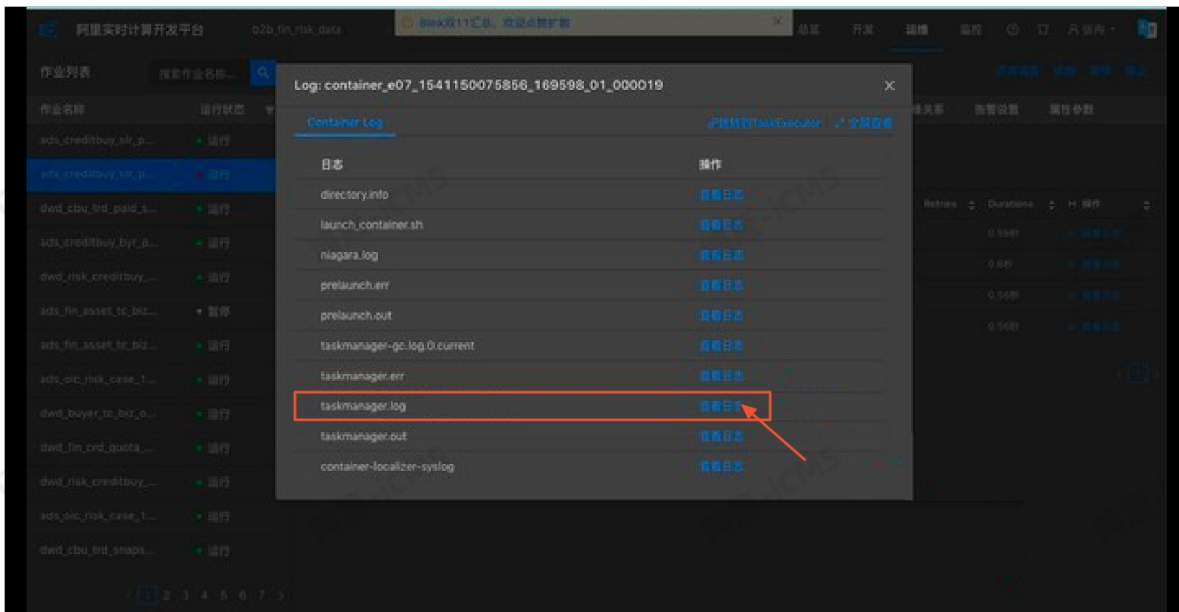
1. 单击Status为FAILED的节点或者Source节点。



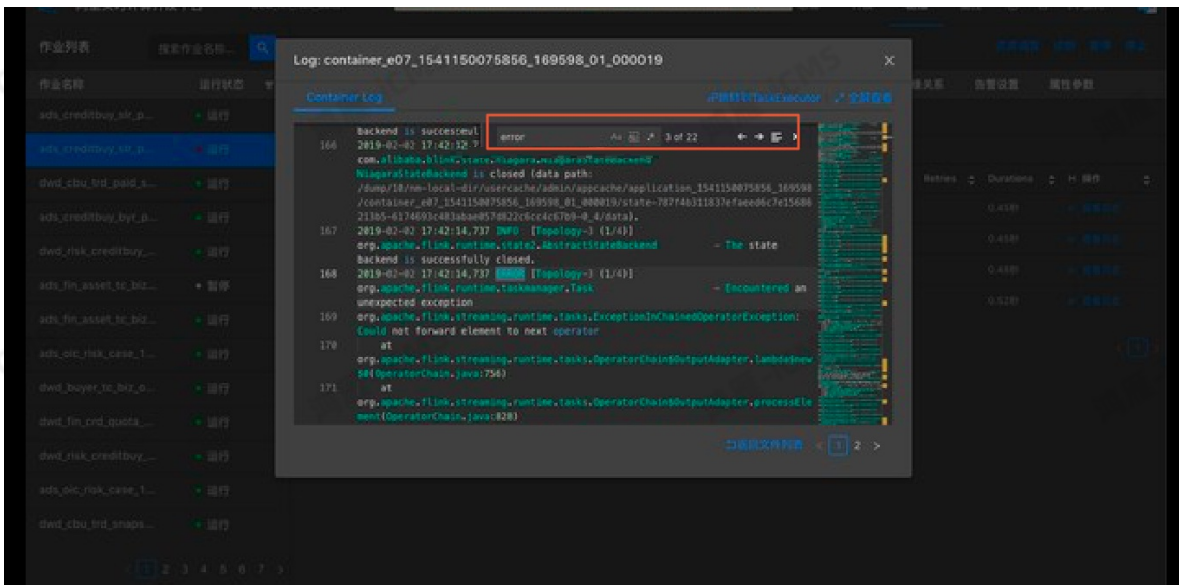
2. 在运行信息 > SubTask List中，单击Status为FAILED的Subtask操作列下的查看日志。



3. 单击taskmanager.log日志操作列下的查看日志。



4. 按下 **ctrl+F** 搜索报错关键字（例如error），并查看报错信息。

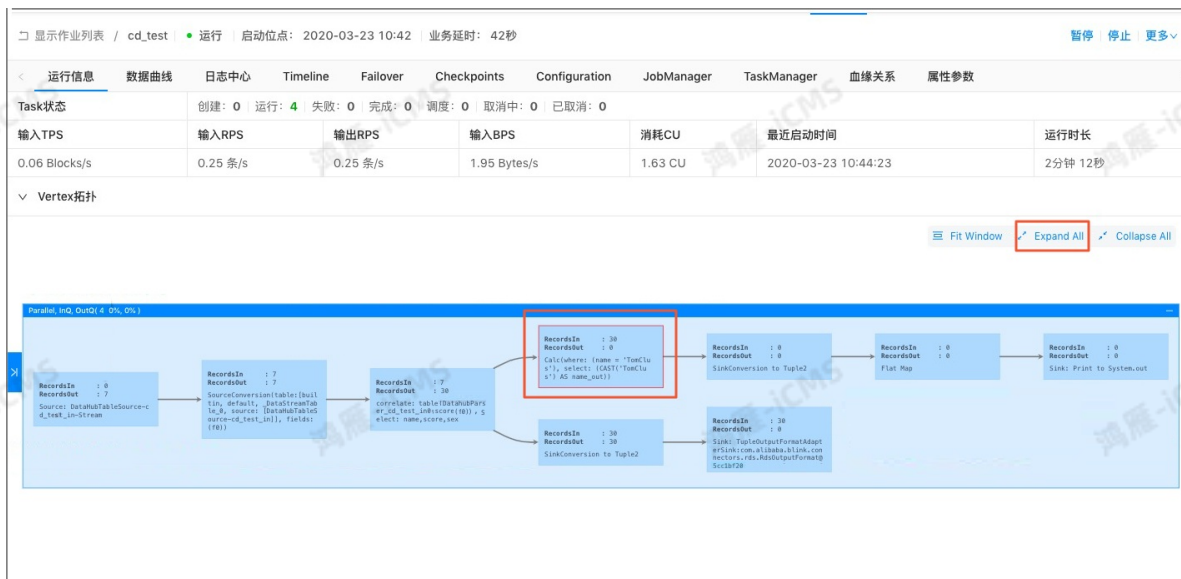


说明 建议从最近的时间开始查看，最近的有效报错信息通常是导致作业运行失败的根本原因。

Vertex排查数据量问题

- 现象分析

数据经过JOIN、WHERE或WINDOW等节点，数据量变少为正常现象，因为数据可能因为条件限制被过滤或JOIN不上。当某个Operator为红色时，代表数据被该节点过滤。



● 过滤场景

○ Window节点过滤



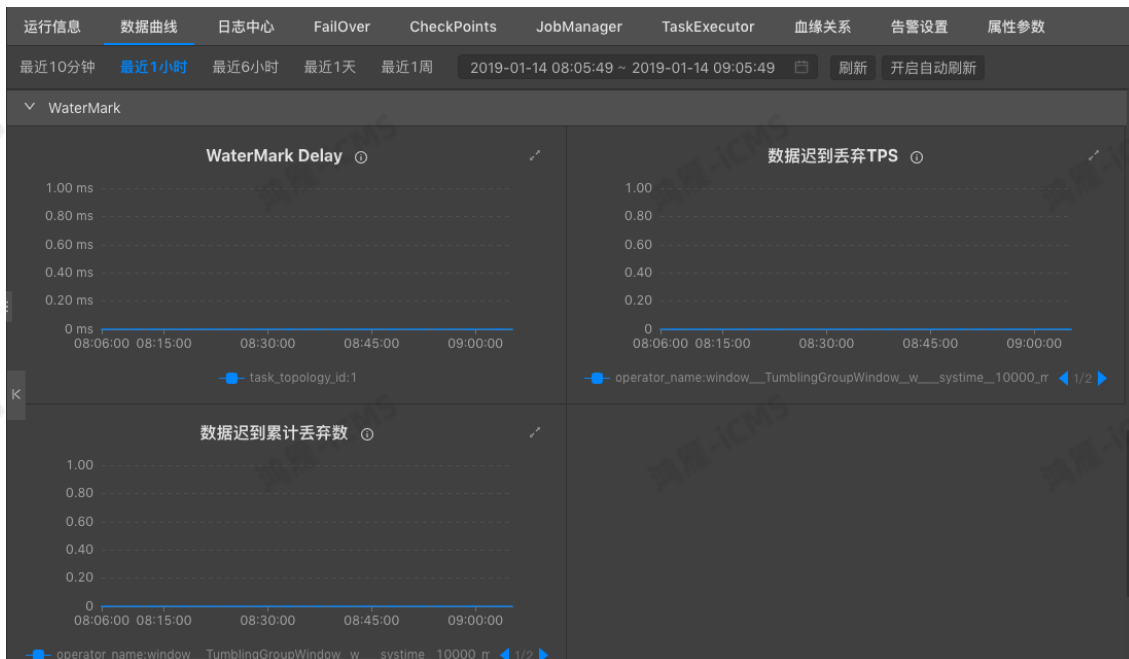
排查方案：

a. 检查开窗的字段ts是否正常：

- 如果您的开窗时间是1970年（时间戳为10位）、15XXX年（时间戳为16位）等非13位时间戳的开窗时间，则需要使用**计算列**转换为13位时间戳后，再进行开窗操作。
- 如果您不确定您的ts字段是否正常，建议您写一个虚拟的Sink，将ts字段打印到日志中，详情请参见[如何通过调试模式（Debug）查看作业的输出信息？](#)

```
CREATE TABLE test_output (
  ts      timestamp
) WITH (
  type = 'print'
);
INSERT into test_output
SELECT
  ts
FROM 源表;
```

b. 查看数据曲线中有关WaterMark的三个曲线。详情请参见数据曲线。



2.x以下的版本：查看数据源表的并发是否都有数据，只要一个并发没有数据，就不会触发WaterMark形成窗口输出。

ID	操作	InQueue	OutQueue	RecvCnt	SendCnt	TPS	Retries	Status
2	跳转TaskExecutor日志	0 (0.00%)	0 (0.00%)	0	0	0	0	RUNNING
1	跳转TaskExecutor日志	0 (0.00%)	4 (100.00%)	0	78209	0.75	0	RUNNING
0	跳转TaskExecutor日志	0 (0.00%)	3 (50.00%)	0	79062	0.90	0	RUNNING

说明 假设，窗口设置为1分钟的滚动窗口，数据被窗口过滤。检查下一个窗口是否有数据到来，如果下个窗口没有数据到来，则不会触发上一个窗口的结束，因此窗口没有输出。

Where节点过滤

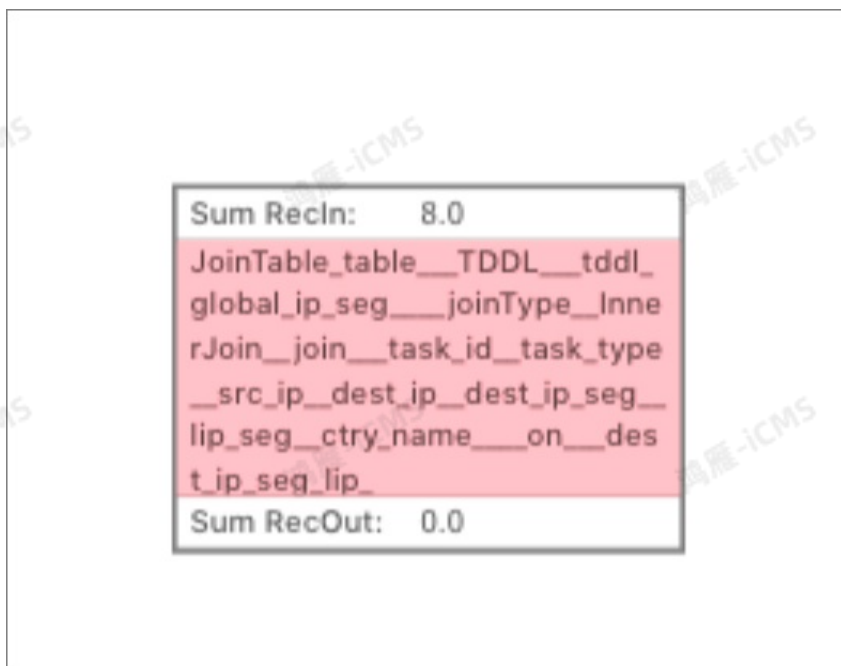


排查方案

- a. 将Where条件注释掉后，再将没有Where条件的数据打印到日志中，检查是否有数据满足Where条件。详情请参见[如何通过调试模式（Debug）查看作业的输出信息？](#)
- b. 当有符合Where条件的数据时，请检查Where条件的写法。例如，在Where条件中使用了to_timestamp、date_format等函数时，您可以将Where条件中使用的函数打印到虚拟Sink中，检查函数使用的方法是否正确。

```
CREATE TABLE test_output (  
  t1      varchar  
) WITH (  
  type = 'print'  
);  
INSERT into test_output  
SELECT  
  DATE_FORMAT(datetime1, 'yyMMdd')  
FROM 源表;
```

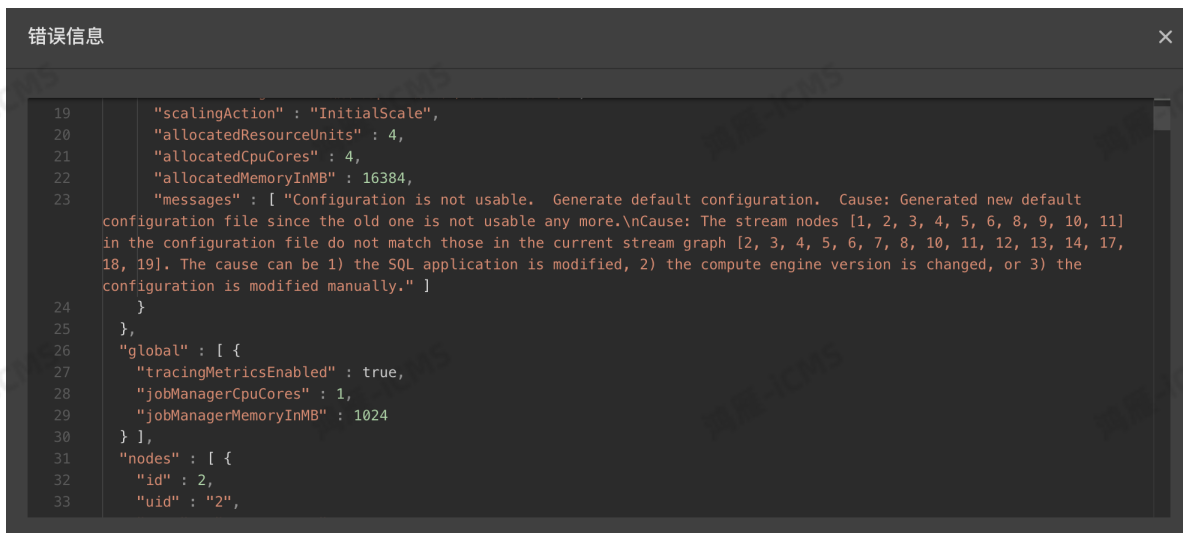
○ JOIN节点过滤



排查方案：检查ON条件，如果您不确定ON条件是否有问题，您可以将左右两个表的数据打印到日志中检查。详情请参见[如何通过调试模式（Debug）查看作业的输出信息？](#)

重新获取配置资源报错，该如何处理？

● 报错信息

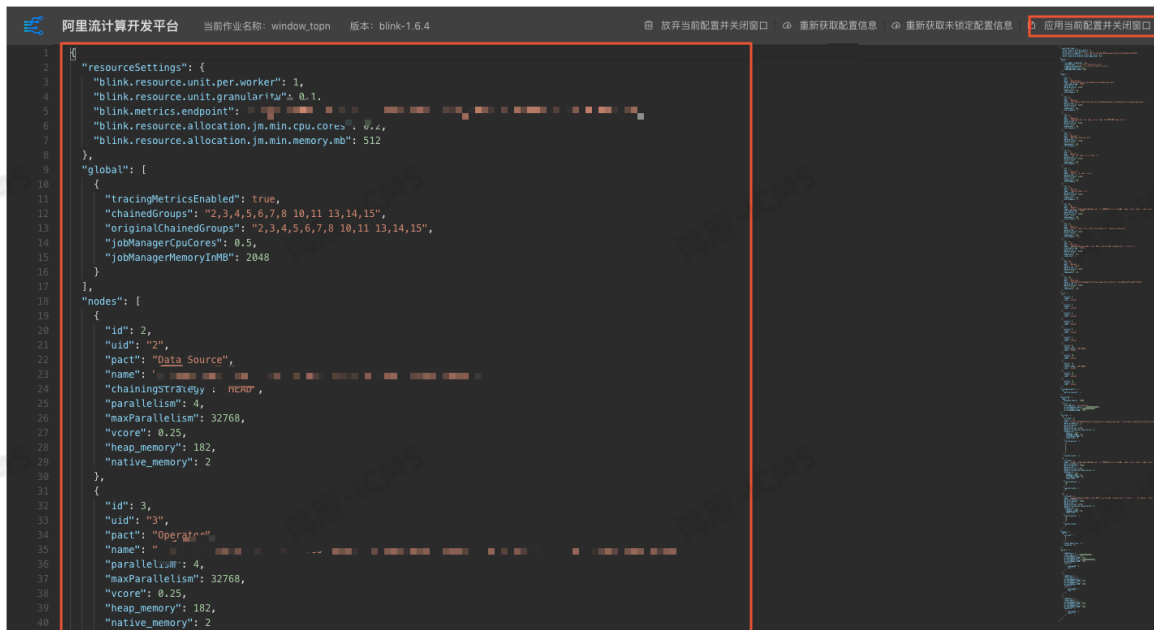


● 报错原因

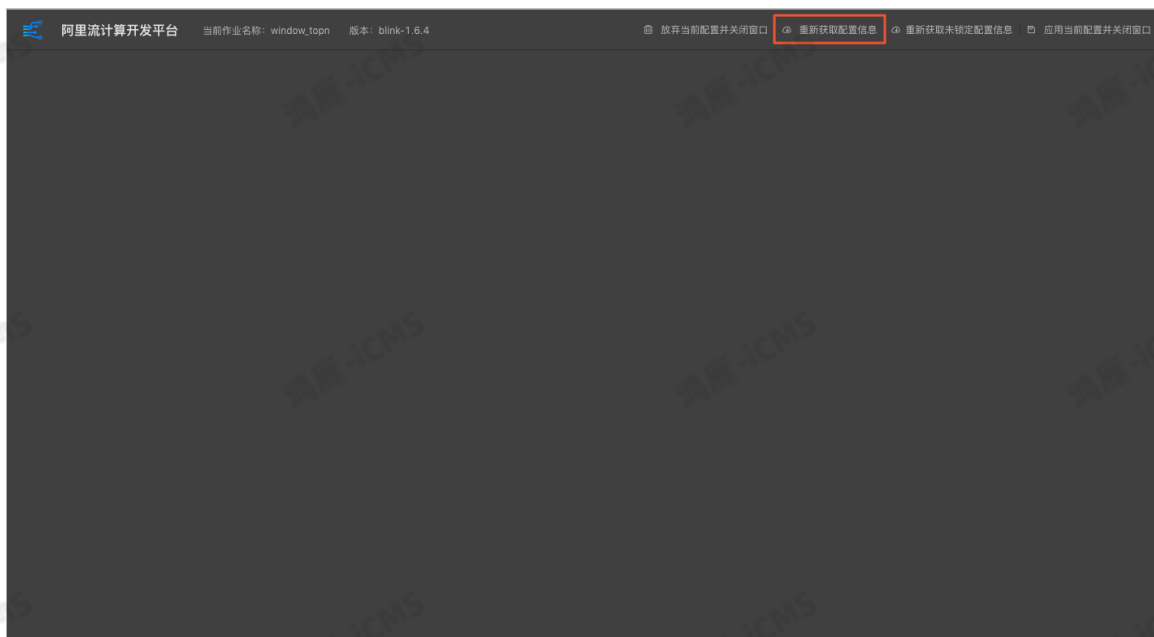
无法正常获取当前资源配置信息。

● 解决方案

- i. 在资源配置页面，按住 CTRL+E 将资源配置信息删除后，应用当前配置并关闭窗口。



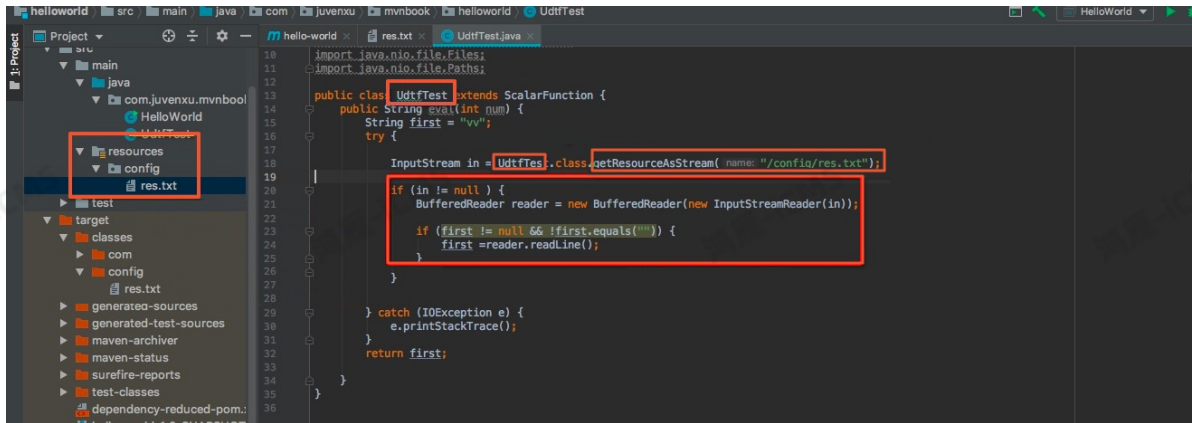
ii. 重新获取配置资源，再应用当前配置并关闭窗口。



在使用UDX函数时，引用JAR包内的本地文件报错

如果引用的JAR包内的本地文档报错信息为空指针，您可以从以下4个方面进行确认：

- 本地文件是否存放在resources文件夹下。
- 是否使用当前加载路径（UdtfTest）。
- 获得相对路径下资源的方法是否为 `getResourceAsStream(name:"/config/res.txt")`。
- 是否正确读取文件内的内容。



GROUP BY数据出现热点、数据倾斜

- 问题描述

在作业处理的速度很慢时，增加了资源后，作业处理速度并没有明显提升。

- 问题原因

数据出现热点、数据倾斜。

- 解决方案

- 优化前

```

SELECT
ID,
COUNT(distinct NAME)
FROM AA
GROUP BY ID;
    
```

- 优化后

```

CREATE VIEW SSS AS
SELECT
ID,
NAME
FROM AA
GROUP BY ID,NAME;
INSERT INTO SS
SELECT
ID,
COUNT(NAME)
FROM SSS
GROUP BY ID;
    
```

说明 把SQL拆分成两次进行GROUP BY，第一次去重，第二次聚合。

OpenAPI DEMO报错

- 报错信息

```

\inputVertices\ : [ ],\n      \targetTps\ : 0.0\n    },\n    \2\ : {\n      \verti
3 ],\n    \inputFromVertices\ : [ ],\n    \targetTps\ : 0.0\n    },\n    \4\ : {\n
\vertices\ : [ 5 ],\n    \inputFromVertices\ : [ ],\n    \targetTps\ : 0.0\n    }\n
\estimatedMemoryInMB\ : 2868.0,\n    \allocatedCpuCores\ : 0.7,\n    \allocatedMemoryInMB\
"engineVersion": "blissmall-2.2.7", "engineJobHandler": "application_151054905049_10028|1ee6912f3a849
ServiceUnavailable: The request has failed due to a temporary failure of the server.
RequestId DD304876-E624-4CCE-8748-72EA81EFBA06
null
Disconnected from the target VM. address: '127.0.0.1:2100', transport: 'socket'
{"es": "C6444C0", "runSummary": {"id": "000007", "actualState": "TER
\jobName": "blissmall_dim_clean"}}

```

- 报错原因
您使用了4.3.0以下版本的 `aliyun-java-sdk-core` 。
- 解决方案
请使用4.3.0及以上版本的 `aliyun-java-sdk-core` 。

云监控控制台页面配置监控告警时，显示无数据，应该如何处理？

- 问题描述
实时计算Flink版作业运维页面有数据曲线，且有数据传给云监控消费的日志信息，但在云监控控制台配置监控告警时，显示无数据。
- 问题原因
子账号没有授予相关权限。
- 解决方案
添加AliyunCloudMonitorFullAccess和AliyunCloudMonitorReadOnlyAccess权限。

* 选择权限

系统策略 自定义策略 [新建权限策略](#)

云监控 ↻

权限策略名称	备注
AliyunCloudMonitorFullAccess	管理云监控（CloudMonitor）的权限
AliyunCloudMonitorReadOnlyAccess	只读访问云监控（CloudMonitor）的权限

9.5. 上下游存储

本文汇总了实时计算上下游存储常见报错与解决方案。

- 如何避免结果表写入不同步问题？
- 存放的数据产品不在实时计算支持的存储列表，应该如何处理？

- DataHub作为结果表存在大量重复数据，应该如何处理？
- DataHub中Shard在只读状态下可以被读取数据吗？
- 实时计算每处理一条记录都会去新增或者更新RDS数据库表吗？将实时计算结果写入数据库，是否会对RDS产生性能影响？
- 实时计算上游数据源是否支持RDS数据库？
- 如何使用Java连接实时计算与消息队列？
- 如何把结果数据输出到一个存在主键限制的外部存储？
- 使用SINK WITH PRIMARY KEY有什么限制？
- 哪些类型的结果表支持在DDL中定义PRIMARY KEY，并能够按照PRIMARY KEY进行INSERT、UPDATE和DELETE？
- 在Sink节点的并发度不为1，多个并发中存在相同PRIMARY KEY的数据时，Sink节点如何保证多个并发间对相同PRIMARY KEY的数据更新顺序一致？
- 为什么相同PRIMARY KEY的数据写到外部存储的顺序和Sink的输入流相同PRIMARY KEY的数据顺序不一致？

如何避免结果表写入不同步问题？

- 问题描述

把相同的数据保存到属于不同数据库但表结构相同的两个结果表里，两张表数据的写入速率不同，但最终的结果一致。
- 问题原因

由于结果表的并发量不同，数据分发出现数据乱序。
- 解决方案
 - 对作业进行自动配置调优，详情请参见 [自动配置调优](#)。
 - 为两个并发不同的结果表设置相同的节点，让两个节点串联（Chain）在一起。详情请参见 [手动配置调优](#)。

存放的数据产品不在实时计算支持的存储列表，应该如何处理？

如果您使用了自建开源存储，需要将您的数据迁移到实时计算支持的列表存储中。

DataHub作为结果表存在大量重复数据，应该如何处理？

- 问题描述

从DataHub的Topic A中读取数据，通过实时计算处理后写入到DataHub中的Topic B，发现有大量重复数据。
- 原因分析

上述问题是由DataHub存储机制导致的。DataHub是一种消息队列服务，写入一条数据，就存储一条数据，不会对数据的重复性进行判断，而会将每一条输入的数据当成新数据进行储存。在此场景中，运算过程中的计算数据同样被DataHub储存，导致产生重复数据。
- 解决方案

请您将结果表换成RDS。RDS会根据主键更新结果，不会储存计算过程数据。

DataHub中Shard在只读状态下可以被读取数据吗？

• 问题描述

DataHub中Shard在只读状态下可以被读取数据吗？

创建时间	2018-04-03 17:37:46		
修改时间	2018-04-03 17:37:46		
Shard数量	12		
类型	TUPLE		
注释	stg_product_logs		

Shards	DataConnector	Metric	Schema	Subscription
	状态	最早数据时间	最新数据时间	
	CLOSED	2018-05-08 15:28:45	2018-05-15 14:21:47	
	CLOSED	2018-05-08 15:29:02	2018-05-15 14:22:48	

• 问题解析

Shard在可读写状态下是可以被读取数据的。上图中Shard状态为CLOSED，而CLOSED是只读状态，并非可读写状态，因此无法被读取。关于Shard状态说明，请参见[Shard状态说明](#)。

实时计算每处理一条记录都会去新增或者更新RDS数据库表吗？将实时计算结果写入数据库，是否会对RDS产生性能影响？

实时计算的每条结果数据会触发对RDS的写入。如果存在KEY，重复的值会更新（UPDATE），不重复的值会新增（INSERT）。您可以通过设置Batchsize参数来降低对RDS性能的影响。

实时计算上游数据源是否支持RDS数据库？

不支持，实时计算支持以下两类上游数据源来驱动计算：

- 消息队列类型。例如，消息队列Kafka、消息队列RocketMQ版和DataHub等。
- 非消息队列类型。例如，大数据计算服务（MaxCompute，原名ODPS）和表格存储Tablestore等。

如何使用Java连接实时计算与消息队列？

您可以使用Flink Datastream的方式实现实时计算读取消息队列RocketMQ版中的数据。详情请参见 [概述](#)。

如何把结果数据输出到一个存在主键限制的外部存储？

A：例如，输出结果数据到一张存在PRIMARY KEY限制的MySQL表，MySQL建表语句如下。

```
CREATE TABLE words (
  word VARCHAR(30) NOT NULL,
  cnt BIGINT,
  CONSTRAINT word_pk PRIMARY KEY(word)
);
```

Q: Blink统计每个word的出现频次后，把结果输出到真实的物理表words中。Blink定义的结果表rds_output的DDL中声明了PRIMARY KEY为word，Sink处理每条输入数据的逻辑为：

- 如果words表中不存在和rds_output主键一致的数据(w,c1)，则在words表插入数据(w,c1)。
- 如果words表中存在和rds_output主键一致的数据(w,c1)，则在words表中更新该条记录，即替换(w,c)为(w,c1)。

说明

- Blink结果表rds_output定义中的PRIMARY KEY必须和MySQL真实的物理表的PRIMARY KEY一致。
- 如果rds_output表DDL里未声明PRIMARY KEY，在作业运行阶段，向words表中插入一条word已经存在的记录，则会因为MySQL的主键限制引发运行异常。

---Split用于把句子切分成单词。

```
CREATE FUNCTION split AS 'com.hjc.test.blink.sql.udx.SplitUdtf';
CREATE TABLE tt_input(
  sentences VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>'
);
CREATE TABLE rds_output(
  word VARCHAR,
  cnt BIGINT,
  PRIMARY KEY(word)
) with (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='words',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT word, COUNT(1) AS cnt FROM
(
  SELECT word FROM tt_input, LATERAL TABLE(split(sentences)) as T(word)
) GROUP BY word;
```

使用SINK WITH PRIMARY KEY有什么限制？

SINK WITH PRIMARY KEY用于把数据输出到一个有主键限制的外部存储中，例如MySQL。使用SINK WITH PRIMARY KEY时，必须同时满足以下两个条件：

- Sink的外部存储必须具有PRIMARY KEY限制，且Blink结果表定义里声明的PRIMARY KEY必须和物理存储的PRIMARY KEY一致。
- Sink必须具有按主键进行INSERT、UPDATE和DELETE的能力。

哪些类型的结果表支持在DDL中定义PRIMARY KEY，并能够按照PRIMARY KEY进行INSERT、UPDATE和DELETE？

首先，结果表对应的外部存储必须具有PRIMARY KEY限制。其次，Connector Sink支持单个并发间按照PRIMARY KEY进行INSERT、UPDATE和DELETE。同时满足以上两个条件，才可以在Sink对应的DDL中声明PRIMARY KEY。

Blink内置的Connectors中，仅以下Connectors支持按照PRIMARY KEY进行INSERT、UPDATE和DELETE：

- 云数据库HBase版
- 交互式分析Hologres
- 分析型数据库MySQL3.0
- 分析型数据库MySQL
- 表格存储TableStore
- 云数据库RDS SQL Server版
- 云数据HybridDB for MySQL
- 云数据库Redis版
- 云数据库RDS PostgreSQL版
- 云数据库RDS版
- Oracle
- Elasticsearch
- Phoenix5

在Sink节点的并发度不为1，多个并发中存在相同PRIMARY KEY的数据时，Sink节点如何保证多个并发间对相同PRIMARY KEY的数据更新顺序一致？

不能保证多并发间对相同PRIMARY KEY的数据更新顺序一致。因为多个并发间如果存在相同PRIMARY KEY的数据，最终写入到外部存储的数据取决于各个并发间完成写动作的前后顺序，这将导致结果的不确定性。

为了避免因为Sink节点多并发导致的数据不确定性，需要将相同PRIMARY KEY的数据落在同一个Sink并发节点上，以下Pattern都可以让Sink的输入数据满足要求：

- {Aggregate -> Sink}或{Aggregate -> [非状态节点，例如calculate] -> Sink}
- {双流 Join -> Sink}或{双流 Join -> [非状态节点，例如calculate] -> Sink}

其中Aggregate的GroupKey或双流Join的JoinKey需要满足以下条件：

- SINK PRIMARY KEY需要和JoinKey或GroupKey任何一个相同。
- SINK PRIMARY KEY必须是JoinKey或GroupKey任何一个的子集。
- 如果SINK PRIMARY KEY不和JoinKey或GroupKey任何一个相同，且不是它们任何一个的子集，需要显式的从JoinKey或GroupKey转换到SINK PRIMARY KEY，最直接的方式就是加一层Aggregate，从而以PRIMARY KEY作为Aggregate的GroupKey。


为什么相同PRIMARY KEY的数据写到外部存储的顺序和Sink的输入流相同PRIMARY KEY的数据顺序不一致？

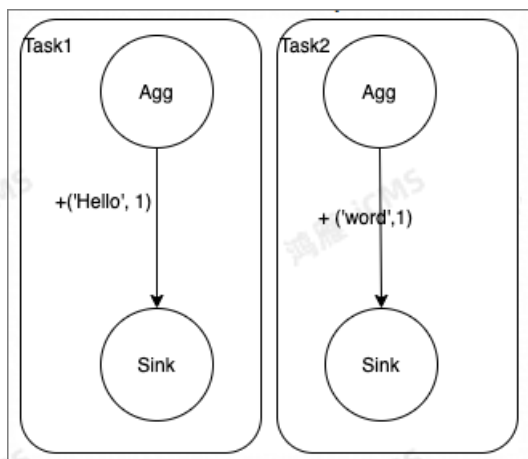
- 作业SQL有问题

例如，由于Sink的PRIMARY KEY和上游的Aggregate的GroupKey并无关联，因此，相同PRIMARY KEY的数据落在不同的并发，最终导致外部存储中的数据顺序和Sink的输入流的不一致，因而数据结果也不正确。

○ 错误示例1

```
---Split用于把句子切分成单词。
CREATE FUNCTION split AS 'com.hjc.test.blink.sql.udx.SplitUdtf';
CREATE TABLE tt_input(
  sentences VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>'
);
---错误示例：Sink的PRIMARY KEY和Aggregate的GroupKey不一致。
CREATE TABLE rds_output(
  word VARCHAR,
  cnt BIGINT,
  PRIMARY KEY(cnt)
) with (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='words',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT word, COUNT(1) AS cnt FROM
(
  SELECT word FROM tt_input, LATERAL TABLE(split(sentences)) as T(word)
) GROUP BY word
```

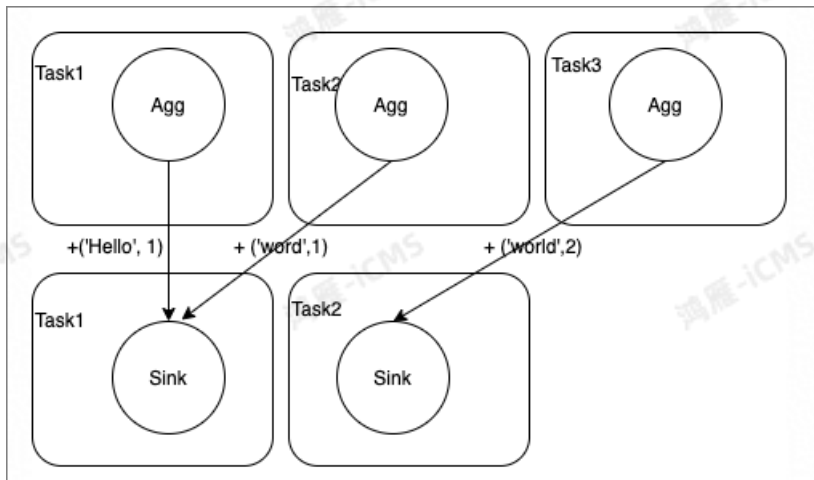
 **说明** 该示例中，Sink的PRIMARY KEY和Aggregate的GroupKey不一致，导致相同cnt的数据落在不同的并发上，最终写到Sink里的数据不确定。PRIMARY KEY为1的记录可能为('Hello',1)或('word',1)，这取决于哪条记录最后写到RDS。



o 错误示例2

```
---Split用于把句子切分成单词。
CREATE FUNCTION split AS 'com.hjc.test.blink.sql.udx.SplitUdtf';
CREATE TABLE tt_input(
  sentences VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>'
);
--- 错误示例：Sink的Primary和Aggregate的GroupKey不一致。
CREATE TABLE rds_output(
  word VARCHAR,
  cnt BIGINT,
  PRIMARY KEY(cnt)
) with (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='words'
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
-- 声明了partitionBy是cnt, Sink前强制按照partitionBy进行一次Shuffle。
  partitionBy = 'cnt'
);
INSERT INTO rds_output
SELECT word, COUNT(1) AS cnt FROM
(
  SELECT word FROM tt_input, LATERAL TABLE(split(sentences)) as T(word)
) GROUP BY word
;
```

❓ 说明 该示例中，Sink的Primary和Aggregate的GroupKey仍然不一致，和错误示范1唯一的区别为Sink声明了PartitionedKey，导致写入Sink前会按照partitionKey进行一次Shuffle。但是这样的结果仍然具有不确定性，PRIMARY KEY为1的记录可能为('Hello',1)或('word',1)，这取决于哪条记录最后写到RDS。

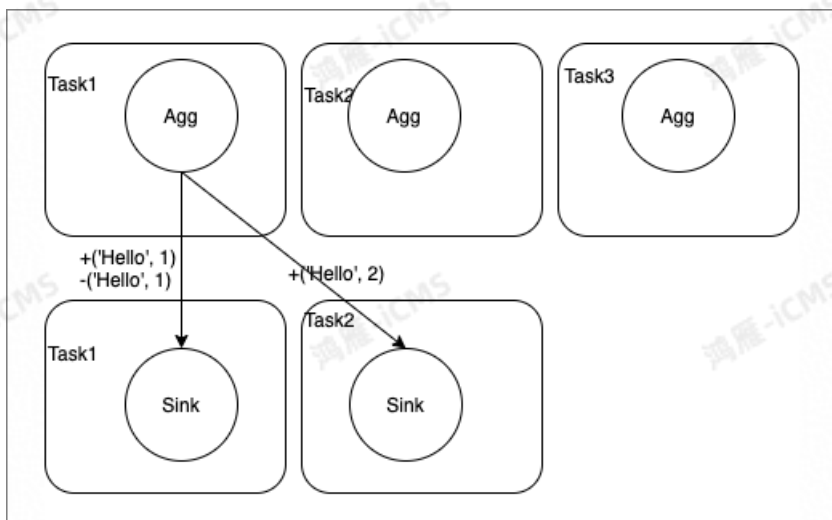


● 作业PlanJson被破坏

作业的PlanJson被手动修改了Sink或Aggregate并发度，导致Aggregate和下游的Sink不能Chain起来，破坏了Sink输入数据的Distribution属性。

```
CREATE TABLE tt_input(
  word VARCHAR
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>'
);
CREATE TABLE rds_output(
  word BIGINT,
  cnt BIGINT,
  PRIMARY KEY(word)
) with (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='words',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT word, COUNT(1) AS cnt FROM tt_input GROUP BY word;
```

说明 该示例中，SQL本身没有问题，但在PlanJson上手动调整Sink节点的并发度，导致原来的Forward的边变成了Rebalance的边。最终PRIMARY KEY为Hello对应的记录可能为('Hello',2)，也可能不存在该PRIMARY KEY，这取决于哪条记录最后写到RDS。



9.6. SQL常见问题

9.6.1. 语法检查报错

本文汇总了实时计算语法检查常见报错与解决方案。

SQL编写完成后，请您先进行语法检查。如果出现报错信息，请更改SQL后，再上线运行您的作业。常见的语法检查报错信息如下：

- 报错：column '<columnName>' is ambiguous
- 报错：Failed to encode 'xxxx' in character set 'ISO-8859-1'
- 报错：'Rds' is not supported as a source table
- 报错：Query result and target table 'table_name' field length not match
- 报错：Interval field value 100 exceeds precision of MINUTE(2) field
- 报错：ODPS *** Partition null table count ODPS tables row count exceeds maxRowCount limit {2}
- 报错：SQL validation failed: index (55) must be less than size (55)

报错：column '<columnName>' is ambiguous

- 报错信息

column '<columnName>' is ambiguous 。

- 问题原因

关联的两张表存在同名字段，在引用时未在字段前指定表名。

- 解决方案

○ 修改前

```
select
column_name
.....
join table_A.column_name = table_B.column_name
```

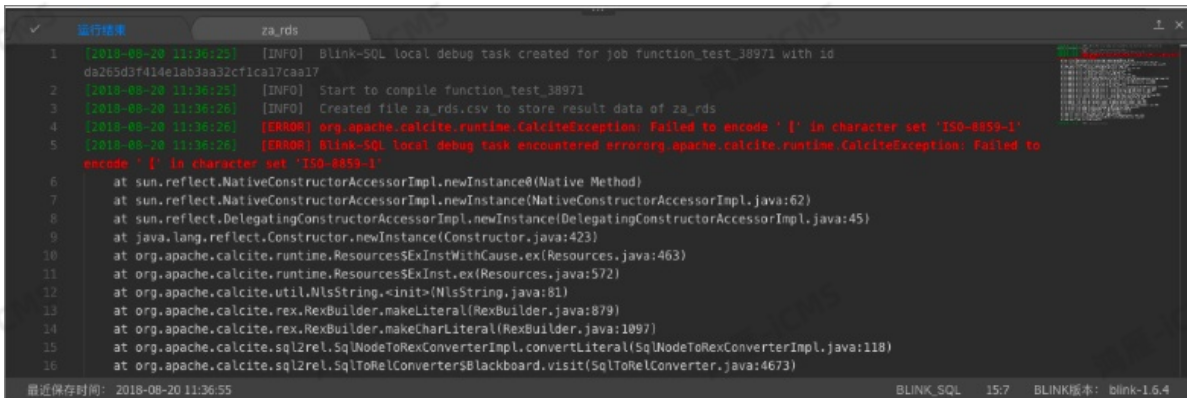
○ 修改后

```
select
table_A.column_name
.....
join table_A.column_name =table_B.column_name
```

报错：Failed to encode 'xxxx' in character set 'ISO-8859-1'

● 报错信息

```
[ERROR] org.apache.calcite.runtime.CalciteException: Failed to encode '[' in character set 'ISO-8859-1'[E
RROR] Blink-SQL local debug task encountered errororg.apache.calcite.runtime.CalciteException: Failed t
o encode '[' in character set 'ISO-8859-1'
```



● 报错原因

Flink SQL中不支持直接使用中文字符，包括中文文字和中文标点符号。

● 解决方案

在中文字符前面添加 `_UTF16`。

```
_UTF16 '['
_UTF16'您好'
```

说明 常量都需要添加 `_UTF16`。

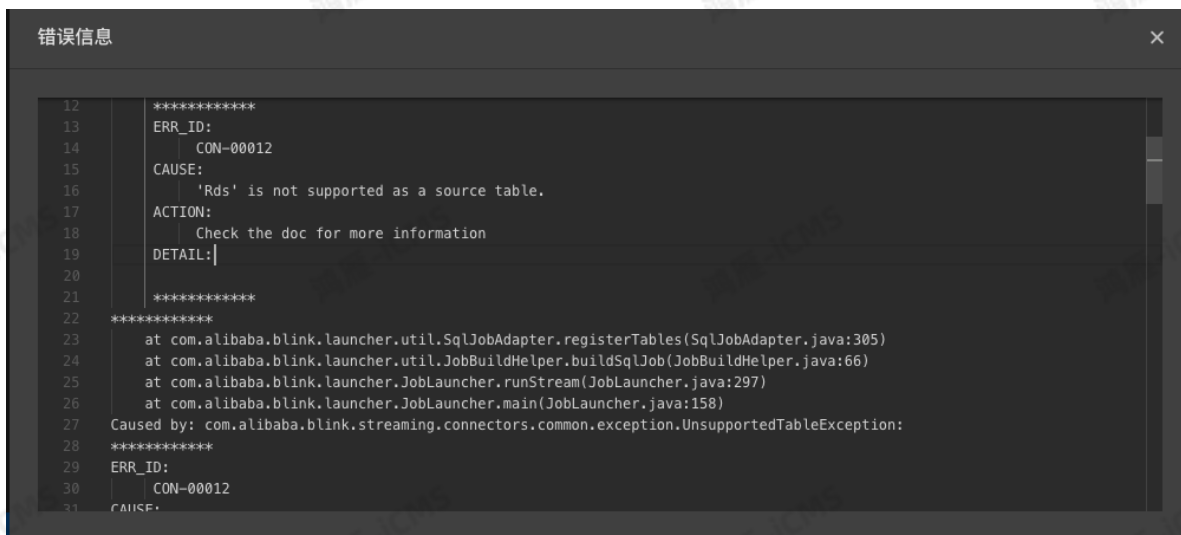
报错：'Rds' is not supported as a source table

● 报错信息

```

ERR_ID:
  CON-00012
CAUSE:
  'Rds' is not supported as a source table.
ACTION:
  Check the doc for more information

```



```

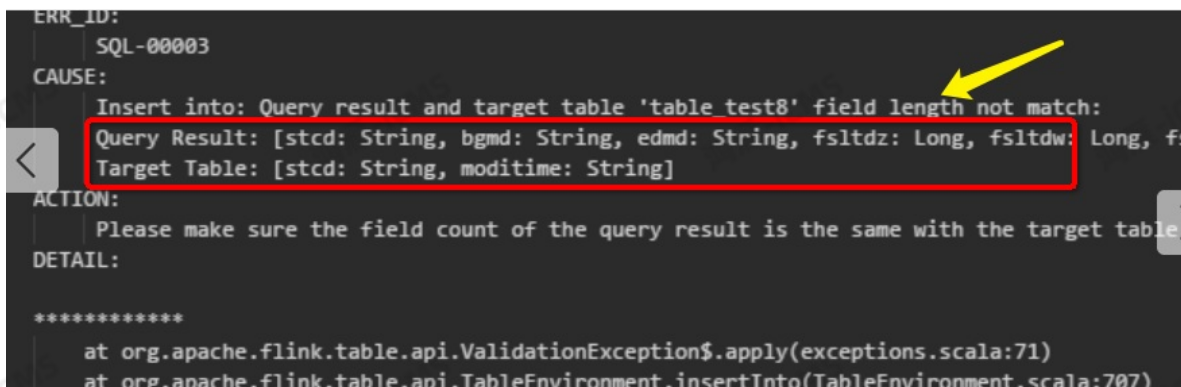
错误信息
12 *****
13 ERR_ID:
14   CON-00012
15 CAUSE:
16   'Rds' is not supported as a source table.
17 ACTION:
18   Check the doc for more information
19 DETAIL:|
20
21 *****
22 *****
23   at com.alibaba.blink.launcher.util.SqlJobAdapter.registerTables(SqlJobAdapter.java:305)
24   at com.alibaba.blink.launcher.util.JobBuildHelper.buildSqlJob(JobBuildHelper.java:66)
25   at com.alibaba.blink.launcher.JobLauncher.runStream(JobLauncher.java:297)
26   at com.alibaba.blink.launcher.JobLauncher.main(JobLauncher.java:158)
27 Caused by: com.alibaba.blink.streaming.connectors.common.exception.UnsupportedTableException:
28 *****
29 ERR_ID:
30   CON-00012
31 CAUSE:

```

- 报错原因
 - 作业中声明了RDS的表，但是没有INSERT该表。
 - RDS作为维表，但是没有声明维表PERIOD FOR SYSTEM_TIME。
- 解决方案
 - INSERT作业中声明的RDS表。
 - 在作业中的维表中写入声明PERIOD FOR SYSTEM_TIME。

报错：Query result and target table 'table_name' field length not match

- 报错信息



```

ERR_ID:
  SQL-00003
CAUSE:
  Insert into: Query result and target table 'table test8' field length not match:
  Query Result: [stcd: String, bgmd: String, edmd: String, fsldtz: Long, fsldtw: Long, fs
  Target Table: [stcd: String, moditime: String]
ACTION:
  Please make sure the field count of the query result is the same with the target table.
DETAIL:
  *****
  at org.apache.flink.table.api.ValidationException$.apply(exceptions.scala:71)
  at org.apache.flink.table.api.TableEnvironment.insertInto(TableEnvironment.scala:707)

```

🔍 说明 报错提示中的table_name指代具体实例中的表名。

- 报错原因

写入的字段与结果表中声明字段的数量、类型、顺序不一致。

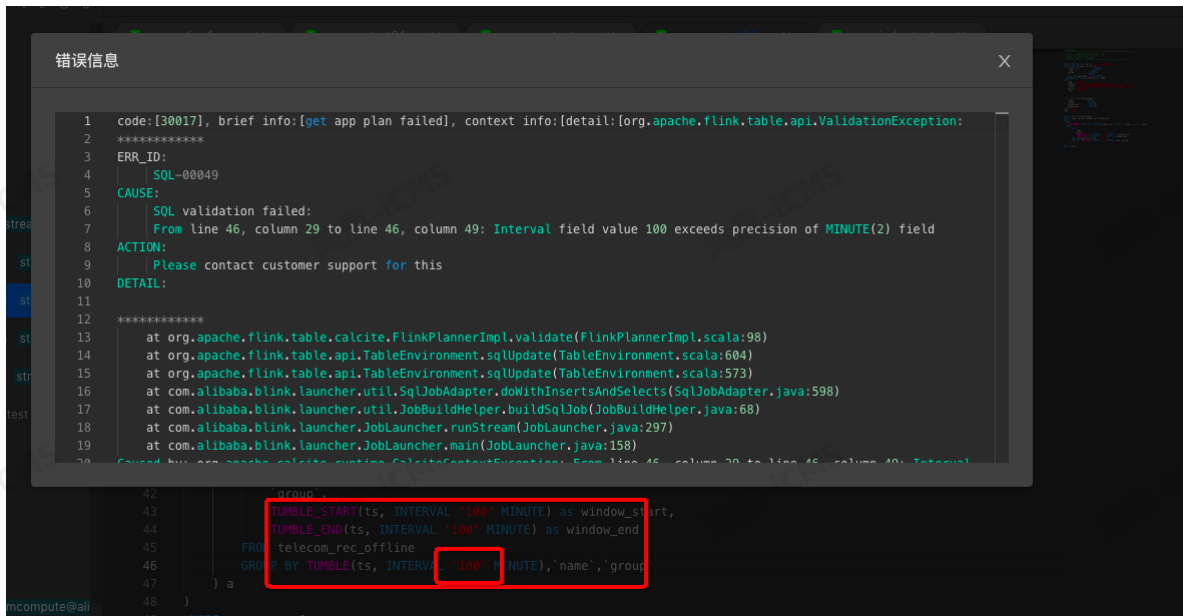
• 解决方案

修改语句使写入语句的字段数量、类型、顺序都与结果表保持一致。

报错：Interval field value 100 exceeds precision of MINUTE(2) field

• 报错信息

在窗口里的年、月、日、时、分、秒设置超过两位数时，语法检查报错。

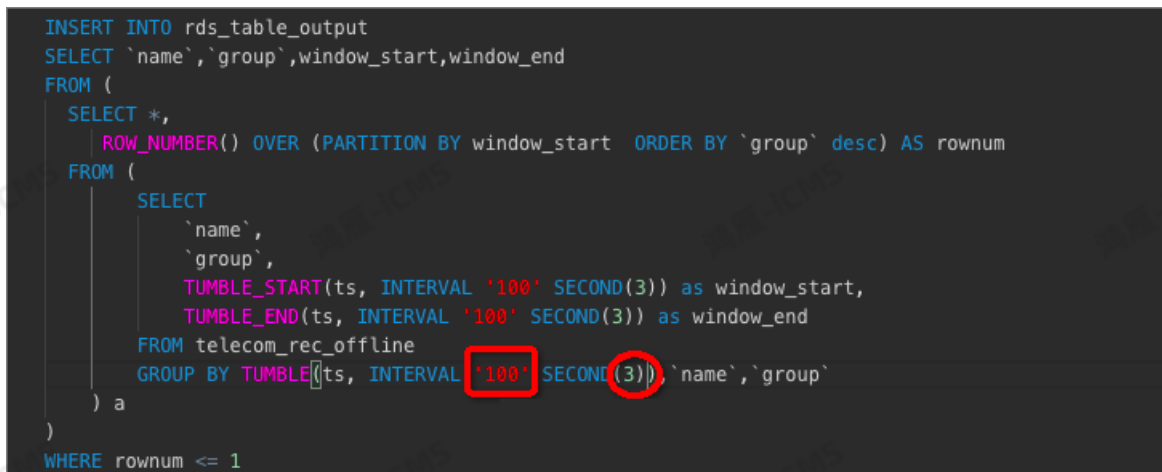


• 报错原因

默认的时间间隔精度为两位数。

• 解决方案

建议您在时间单位中更改精度的位数。



说明 100是三位数，需要在SECOND后面括号内写为3。

报错：ODPS *** Partition null table count ODPS tables row count exceeds maxRowCount limit {2}

- 报错信息

在MaxCompute（ODPS）作为维表时，语法检查报错。



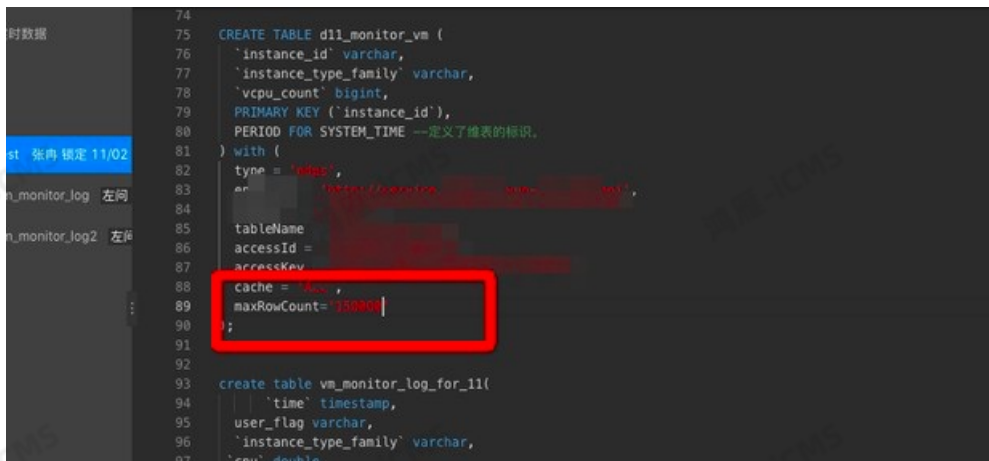
```
2 *****
3 ERR_ID:
4 SQL-00010009
5 CAUSE:
6 Could not create table 'd11_monitor_vm' as dim table
7 ACTION:
8 Please refer to details section for hint.
9 If it doesn't help, please contact customer support
10 DETAIL:
11 com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException:
12 *****
13 ERR_ID:
14 CON-04000054
15 CAUSE:
16 ODPS d11_monitor_vm Partition null table count ODPS tables row count exceeds maxRowCount limit {2}
17 ACTION:
18 please switch to a smaller table
19 DETAIL:
20 *****
21 *****
```

- 报错原因

MaxCompute（ODPS）维表数据记录数超过了maxRowCount默认值 100000。

- 解决方案

修改maxRowCount参数值，如下图所示。



```
74
75 CREATE TABLE d11_monitor_vm (
76   'instance_id' varchar,
77   'instance_type_family' varchar,
78   'vcpu_count' bigint,
79   PRIMARY KEY ('instance_id'),
80   PERIOD FOR SYSTEM_TIME 定义了维表的标识。
81 ) with (
82   type = 'ODPS',
83   ...
84
85   tableName
86   accessId =
87   accessKey
88   cache = '...',
89   maxRowCount='150000'
90 ;
91
92
93 create table vm_monitor_log_for_111(
94   'time' timestamp,
95   user_flag varchar,
96   'instance_type_family' varchar,
97   'cpu' double,
```

报错：SQL validation failed: index (55) must be less than size (55)

- 报错信息

在使用MaxCompute（ODPS）结果表的动态分区功能时，语法检查报错。

```
code:[30017], brief info:[get app plan failed], context info:[detail:
>>>>
org.apache.flink.table.api.ValidationException:
*****
ERR_ID:
  SQL-00120001
CAUSE:
  SQL validation failed:
  index (55) must be less than size (55)
ACTION:
  Please see descriptions above. If it doesn't help, please contact customer support for this.
DETAIL:
*****
<<<<<]
```

- 报错原因

MaxCompute（ODPS）结果表DDL语句中没有声明分区列。

- 解决方案

在DDL语句中声明MaxCompute（ODPS）的分区列。例如示例中的`partition` = 'ds=2018****'。

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

MaxCompute（ODPS）结果表DDL语句详情，请参见[创建MaxCompute（ODPS）结果表](#)。

9.6.2. 调试报错

本文汇总了实时计算调试常见报错与解决方案。

- 报错：Query result field length not match TableSink analysis_result field length
- 报错：Cannot apply 'EXTRACT' to arguments of type 'EXTRACT(<INTERVAL HOUR> FROM <CHAR(19)>)'
- 报错：originated by DoubleParser:NUMERIC_VALUE_FORMAT_ERROD
- 报错：Query result field type(s) not match TableSink T2 field type(s)
- 报错：The type '[B]' is not supported for the CSV input format.
- 报错：检测到调试数据与schema字段数量不符
- 报错：操作错误，所有数据均被跳过，请检查DDL中的LENGTHCHECK项

报错：Query result field length not match TableSink analysis_result field length

• 报错信息

```
[INFO] Blink-SQL local debug task created for job warn_url_leak_31880 with id f0f911f19cbe523de6d4027cdac04ad
[INFO] Start to compile warn_url_leak_31880
[INFO] Created file analysis_result.csv to store result data of analysis_result
[ERROR] Blink-SQL local debug task encountered error org.apache.flink.table.api.ValidationException: Query result field length not match TableSink analysis_result field length.
Query fields: [id: String, EXPR$1: String, EXPR$2: String, EXPR$3: String, EXPR$4: String, EXPR$5: String, EXPR$6: String, EXPR$7: String, EXPR$8: String, EXPR$9: String, EXPR$10: String, EXPR$11: String, EXPR$12: String, EXPR$13: String, EXPR$14: String, EXPR$15: String, EXPR$16: String, EXPR$17: String, EXPR$18: String, EXPR$19: String, EXPR$20: String, EXPR$21: String, EXPR$22: String, EXPR$23: String, EXPR$24: String, EXPR$25: String, EXPR$26: String, EXPR$27: String, user_name: String, user_id: String, area_code: String]
TableSink fields: [id: String, risk_event_name: String, rule_id: String, rule_name: String, trigger_time: String, analysis_type: Integer, status_enum: Integer, process_type_enum: Integer, know_ids: String, know_names: String, process_relate_id: String, weight: Integer, related_ips: String, security_domain_guids: String, security_domain_names: String, logs_info: String, dst_ips: String, dst_asset_guids: String, dst_security_domain_guids: String, dst_security_domain_names: String, marks: String, risk_event_code: String, priority_level: String, asset_guids: String, event_type: String, event_type_level: String, device_type: String, device_type_level: String, result_guid: String, rule_code: String, username: String, user_id: String, area_code: String]
at org.apache.flink.table.api.ValidationException$.apply(exceptions.scala:71)
at org.apache.flink.table.api.TableEnvironment.insertInto(TableEnvironment.scala:696)
at org.apache.flink.table.api.TableEnvironment.sqlUpdate(TableEnvironment.scala:611)
at org.apache.flink.table.api.TableEnvironment.sqlUpdate(TableEnvironment.scala:569)
at com.alibaba.blink.launcher.util.SqlJobAdapter.dealInserts(SqlJobAdapter.java:482)
at com.aliyun.blink.debug.BlinkSqlLocalDebug.run(BlinkSqlLocalDebug.java:115)
at com.aliyun.blink.debug.BlinkDebugTask.call(BlinkDebugTask.java:66)
at com.aliyun.blink.debug.BlinkDebugTask.call(BlinkDebugTask.java:12)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:627)
at java.lang.Thread.run(Thread.java:852)
```

• 报错原因

查询结果字段长度与结果表字段长度不匹配。

• 解决方法

- 调整INSERT语句中的字段数量，与结果表一致。
- 调整INSERT语句中插入的字段数据类型和字段顺序，与结果表一致。
- 如果使用前两种方法排查后，仍然报错，请使用二分法定位问题。

报错：Cannot apply 'EXTRACT' to arguments of type 'EXTRACT(<INTERVAL HOUR> FROM <CHAR(19)>)'

• 报错信息

```
运行结果 T2
1 [2018-07-09 11:45:13] [INFO] Blink-SQL local debug task created for job test_bxy_34673 with id 248c93855576a52359c5ec214411dac8
2 [2018-07-09 11:45:13] [INFO] Start to compile test_bxy_34673
3 [2018-07-09 11:45:13] [INFO] Created file T2.csv to store result data of T2
4 [2018-07-09 11:45:13] [ERROR] org.apache.calcite.runtime.CalciteContextException: Cannot apply 'EXTRACT' to arguments of type 'EXTRACT(<INTERVAL HOUR> FROM <CHAR(19)>)', Supported forms(s): 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME_INTERVAL>)'
5 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME>)'
6 [2018-07-09 11:45:13] [ERROR] org.apache.calcite.runtime.CalciteContextException: From line 20, column 3 to line 20, column 29: Cannot apply 'EXTRACT' to arguments of type 'EXTRACT(<INTERVAL HOUR> FROM <CHAR(19)>)', Supported form(s): 'EXTRACT (<DATETIME_INTERVAL> FROM <DATETIME_INTERVAL>)'
7 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME>)'
8 [2018-07-09 11:45:13] [ERROR] Blink-SQL local debug task encountered error org.apache.flink.table.api.ValidationException: SQL validation failed. From line 20, column 3 to line 20, column 29: Cannot apply 'EXTRACT' to arguments of type 'EXTRACT(<INTERVAL HOUR> FROM <CHAR(19)>)', Supported form(s): 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME_INTERVAL>)'
9 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME>)'
10 Caused by: org.apache.calcite.runtime.CalciteContextException: From line 20, column 3 to line 20, column 29: Cannot apply 'EXTRACT' to arguments of type 'EXTRACT(<INTERVAL HOUR> FROM <CHAR(19)>)', Supported form(s): 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME_INTERVAL>)'
11 'EXTRACT(<DATETIME_INTERVAL> FROM <DATETIME>)'
最近保存时间: 2018-07-09 11:45:13 BLINK_SQL 20:28 BLINK版本: blink-1.6.4
```

• 报错原因

EXTRACT函数的参数类型使用不正确。

• 解决方案

使用正确的EXTRACT函数参数类型，详情请参见 [EXTRACT](#)。

报错：originated by DoubleParser:NUMERIC_VALUE_FORMAT_ERROD

• 报错信息

在DataHub中存在NULL值时，如果从DataHub里抽取数据，报错如下。

```

161 [2018-05-18 17:14:29] [INFO] Stopping Akka RPC service.
162 [2018-05-18 17:14:29] [INFO] Flink Mini Cluster is shut down
163 [2018-05-18 17:14:29] [ERROR] Blink-SQL local debug task encountered errororg.apache.flink.runtime.client.JobExecutionException: Job execution
failed.
164 org.apache.flink.api.common.io.ParseException: Parsing error for column 10 of row
null,null,null,null,null,48145-05-30 07:13:20,null,null' originated by DoubleParser: NUMERIC_VALUE_FORMAT_ERROR.
165 at org.apache.flink.runtime.jobmaster.JobMaster.jobStatusChanged(JobMaster.java:1304)
166 at org.apache.flink.runtime.jobmaster.JobMaster.access$3600(JobMaster.java:143)
167 at org.apache.flink.runtime.jobmaster.JobMaster$JobManagerJobStatusListener$1.run(JobMaster.java:1512)
168 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.handleRunAsync(AkkaRpcActor.java:273)
169 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.handleMessage(AkkaRpcActor.java:131)
170 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.access$000(AkkaRpcActor.java:72)
171 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor$1.apply(AkkaRpcActor.java:110)
172 at akka.actor.ActorCell$$anonfun$become$1.applyOrElse(ActorCell.scala:534)
173 at akka.actor.Actor$class.aroundReceive(Actor.scala:467)
174 at akka.actor.UntypedActor.aroundReceive(UntypedActor.scala:97)
    
```

● 报错原因

CSV文件会把NULL值转化为字符串NULL，系统在处理NULL时，发现记录类型与字符类型不一致，就会提示格式转换错误。在抽取DataHub数据作为样本调试时，通常会出现此错误，如果项目正式上线，则不会出现类似于下图的错误。



● 解决方案

- 过滤DataHub中NULL值。
- 处理NULL为您期望的值。例如，使用CASE WHEN函数处理NULL值。

报错：Query result field type(s) not match TableSink T2 field type(s)

● 报错信息

```

运行结束 T2
1 [2018-07-09 11:52:40] [INFO] Blink-SQL local debug task created for job test_bxy_34674 with id 339125b7b419203952b7e9659a0acf7b
2 [2018-07-09 11:52:40] [INFO] Start to compile test_bxy_34674
3 [2018-07-09 11:52:40] [INFO] Created file T2.csv to store result data of T2
4 [2018-07-09 11:52:40] [ERROR] Blink-SQL local debug task encountered errororg.apache.flink.table.api.ValidationException: Query
result field type(s) not match TableSink T2 field type(s).
5 Query mismatched fields: [EXPR$0: String]
6 TableSink mismatched fields: [var1: Long]
7 at org.apache.flink.table.api.ValidationExceptions$.apply(exceptions.scala:71)
8 at org.apache.flink.table.api.TableEnvironment.insertInto(TableEnvironment.scala:716)
9 at org.apache.flink.table.api.TableEnvironment.sqlUpdate(TableEnvironment.scala:611)
10 at org.apache.flink.table.api.TableEnvironment.sqlUpdate(TableEnvironment.scala:569)
11 at com.alibaba.blink.launcher.util.SqlJobAdapter.dealInserts(SqlJobAdapter.java:482)
12 at com.aliyun.blink.debug.BlinkSqlLocalDebug.run(BlinkSqlLocalDebug.java:115)
13 at com.aliyun.blink.debug.BlinkDebugTask.call(BlinkDebugTask.java:66)
14 at com.aliyun.blink.debug.BlinkDebugTask.call(BlinkDebugTask.java:12)
15 at java.util.concurrent.FutureTask.run(FutureTask.java:266)
16 at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)
17 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:627)
    
```

● 报错原因

类型转换异常。

● 解决方案

查看INSERT语句中的数据类型和结果表声明的数据类型是否一致。

报错：The type '[B' is not supported for the CSV input format.

• 报错信息

```

110 [2021-09-12 14:22:08] [INFO] Successfully got table 'kafka_stream_input_src' from catalog 'builtin'
111 [2021-09-12 14:22:08] [ERROR] Blink-SQL local debug task encountered error, org.apache.flink.table.api.ValidationException:
112 *****
113 ERR_ID:
114     SQL-00120001
115 CAUSE:
116     SQL validation failed:
117     The type '[B' is not supported for the CSV input format.
118 ACTION:
119     Please see descriptions above. If it doesn't help, please contact customer support for this.
120 DETAIL:
121
122 *****
123 Caused by: java.lang.IllegalArgumentException: The type '[B' is not supported for the CSV input format.
124     at org.apache.flink.table.calcite.FlinkPlannerImpl.validate(FlinkPlannerImpl.scala:114)
125     at org.apache.flink.table.util.TableEnvironmentUtil$.queryToRel(TableEnvironmentUtil.scala:38)
126     at org.apache.flink.table.api.TableEnvironment.sqlQuery(TableEnvironment.scala:1392)
127     at com.alibaba.blink.launcher.util.SqlJobAdapter.registerViews(SqlJobAdapter.java:482)
128     at com.alibaba.blink3x.debug.Blink3xSqlDbg.doRun(Blink3xSqlDbg.java:161)
129     at com.alibaba.blink.debug.common.BlinkSqlDebug.run(BlinkSqlDebug.java:95)
130     at com.alibaba.blink.debug.common.BlinkDebugTask.call(BlinkDebugTask.java:26)
131     at com.alibaba.blink.debug.common.BlinkDebugTask.call(BlinkDebugTask.java:7)
132     at java.util.concurrent.FutureTask.run(FutureTask.java:266)
133     at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1147)
134     at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
135     at java.lang.Thread.run(Thread.java:834)
136
137 [2021-09-12 14:22:08] [INFO] dbg process is finish!
138 [2021-09-12 14:22:08] [INFO] process is finish!
139 [2021-09-12 14:22:08] [ERROR] requestId: 202103121422-7Wk5EVH25bayes channel handler exception happened, java.lang.NumberFormatException: nu
140

```

• 报错原因

二进制数据类型不支持本地调试。

• 解决方案

语法检查没有问题请进行线上调试，详情请参见 [线上调试](#)。

报错：检测到调试数据与schema字段数量不符

• 报错信息

发生错误

操作错误：检测到调试数据与schema字段数量不符，请重新上传或抽样调试数据 921025 (201901171137-7JP2EONP4T)

数据预览

随机抽样线上数据 | 顺序抽样线上数据 | 下载模板

key	value
未上传数据或上传数据为空，请上传以后查看	

数据上传

调试数据：

• 报错原因

在调试时，默认使用逗号分隔字段，如果您的数据中存在JSON数据，则数据会被解析成多个字段。

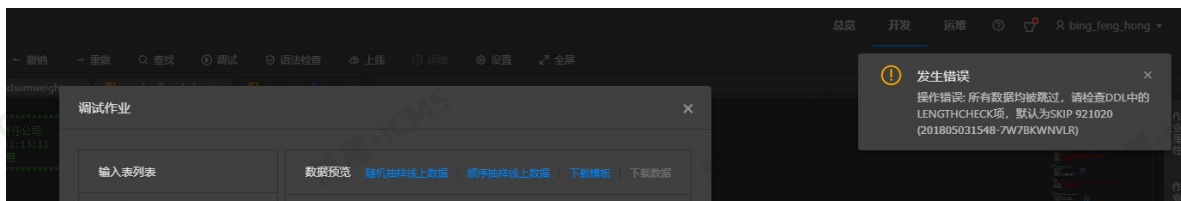
- 解决方案

在作业参数中设置分隔符为非逗号。

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

报错：操作错误，所有数据均被跳过，请检查DDL中的LENGTHCHECK项

- 报错信息



- 错误原因

源表中存在脏数据，导致无法抽取到任何线上数据。

- 解决方法

在源表的WITH参数中增加 `lengthcheck='PAD'` 语句。

9.6.3. 上线报错

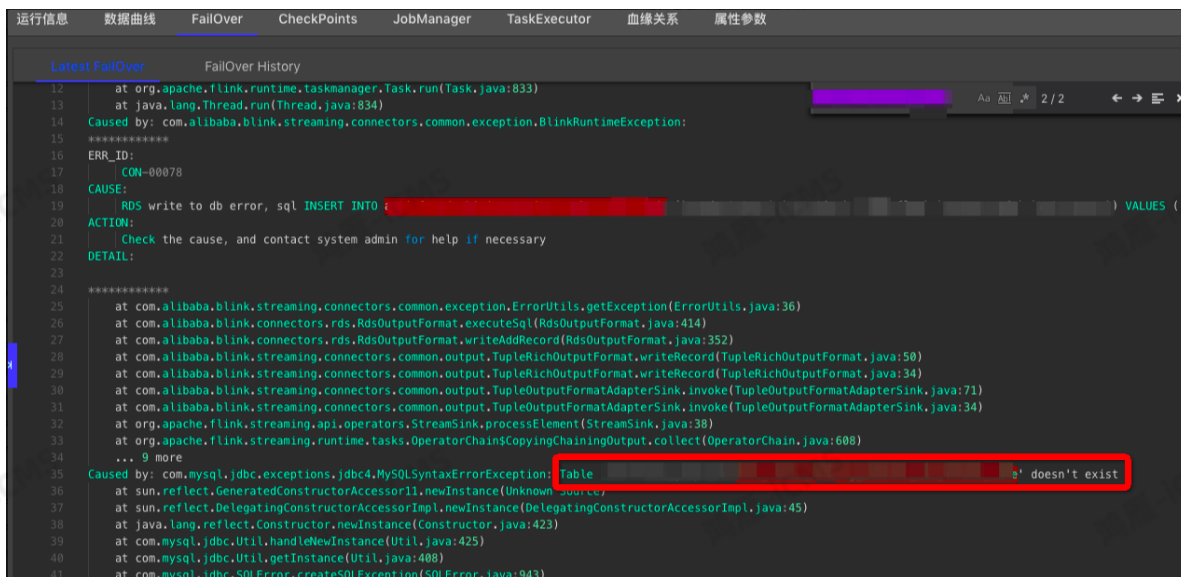
本文汇总了实时计算上线常见报错与解决方案。

- 报错：Table 'database.table' doesn't exist
- 报错：Could not forward element to next operator,null pointer Exception
- 报错：ERROR in next method The response is not valid json
- 报错：ERROR in next method AccessKeyId STS.NK1p3aqjkkagcUHiTpc**** is inactive
- 报错：open catalog connection failed
- 报错：获取数据超时，请重试或尝试手动编辑
- 警告：WARNING（不影响当前流程）

报错：Table 'database.table' doesn't exist

- 报错信息

说明 报错中 `database.table` 指代具体数据库名和表名。



```
Latest FailOver
FailOver History
12 at org.apache.flink.runtime.taskmanager.Task.run(Task.java:833)
13 at java.lang.Thread.run(Thread.java:834)
14 Caused by: com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException:
15 *****
16 ERR_ID:
17 CON-00078
18 CAUSE:
19 RDS write to db error, sql INSERT INTO (VALUES ('
20 ACTION:
21 Check the cause, and contact system admin for help if necessary
22 DETAIL:
23 *****
24
25 at com.alibaba.blink.streaming.connectors.common.exception.ErrorUtils.getException(ErrorUtils.java:36)
26 at com.alibaba.blink.connectors.rds.RdsOutputFormat.executeSql(RdsOutputFormat.java:414)
27 at com.alibaba.blink.connectors.rds.RdsOutputFormat.writeAddRecord(RdsOutputFormat.java:352)
28 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:50)
29 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:34)
30 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:71)
31 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:34)
32 at org.apache.flink.streaming.api.operators.StreamSink.processElement(StreamSink.java:38)
33 at org.apache.flink.streaming.runtime.tasks.OperatorChainCopyingChainingOutput.collect(OperatorChain.java:608)
34 ... 9 more
35 Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table '...' doesn't exist
36 at sun.reflect.GeneratedConstructorAccessor11.newInstance(Unknown Source)
37 at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
38 at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
39 at com.mysql.jdbc.Util.handleNewInstance(Util.java:425)
40 at com.mysql.jdbc.Util.getInstance(Util.java:408)
41 at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:943)
```

- 报错原因
数据库中不存在作业中声明的表。
- 解决方案
 - 检查数据存储中的表是否引用成功。如果不成功，请再次尝试或手动输入。
 - 检查您的数据库中是否存在报错信息中的表，如果没有，请自行创建。

🔍 说明 阿里云实时计算不具有数据存储功能，所有涉及表创建的DDL操作，都是对外部数据表、存储的引用声明。

报错：Could not forward element to next operator,null pointer Exception

- 报错信息

```

java.lang.RuntimeException: java.lang.RuntimeException: Could not forward element to next operator
    at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:131)
    at com.alibaba.blink.streaming.connectors.common.source.AbstractParallelSource.run(AbstractParallelSource.java:120)
    at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:142)
    at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:78)
    at org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:56)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:387)
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
    at java.lang.Thread.run(Thread.java:834)
Caused by: java.lang.RuntimeException: Could not forward element to next operator
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:611)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:941)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:919)
    at org.apache.flink.streaming.api.operators.StreamSource$1.collectWithMetrics(StreamSource.java:127)
    at org.apache.flink.streaming.api.operators.StreamSource$1.collect(StreamSource.java:114)
    at org.apache.flink.streaming.api.operators.StreamSource$1.collect(StreamSource.java:98)
    at org.apache.flink.streaming.api.operators.StreamSourceContexts$ManualWatermarkContext.collectWithTimestamp(StreamSourceContexts.java:273)
    at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.runImpl(ParallelReader.java:223)
    at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:126)
    ... 7 more
Caused by: com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException: 900005

Column For Parser Plugin Error:
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndCollect(DefaultSourceCollector.java:366)
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndCollect(DefaultSourceCollector.java:75)
    at com.alibaba.blink.streaming.connectors.common.source.parse.SourceParser.flatMap(SourceParser.java:47)
    at org.apache.flink.streaming.api.operators.StreamFlatMap.processElement(StreamFlatMap.java:47)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
    ... 16 more
Caused by: java.lang.RuntimeException: Could not forward element to next operator
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:611)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:941)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:919)
    at org.apache.flink.streaming.api.operators.TimestampedCollector.collect(TimestampedCollector.java:51)
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseRawMessage(DefaultSourceCollector.java:400)
    at com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector.parseAndCollect(DefaultSourceCollector.java:362)
    ... 20 more
Caused by: java.lang.NullPointerException
    at com.alibaba.blink.streaming.connectors.common.source.watermark.WatermarkUtils$BayesPunctuatedTimestampAndWatermarkWithOffset.extractTimestamp(WatermarkUtils.java:114)
    at org.apache.flink.streaming.runtime.operators.timestampsAndPunctuatedWatermarksOperator.processElement(TimestampsAndPunctuatedWatermarksOperator.java:50)
    at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
    ... 26 more

```

- 报错原因

数据源端存在NULL值。

- 解决方案

使用计算列过滤NULL值。通常Watermark里的Event time时间字段里存在NULL值。

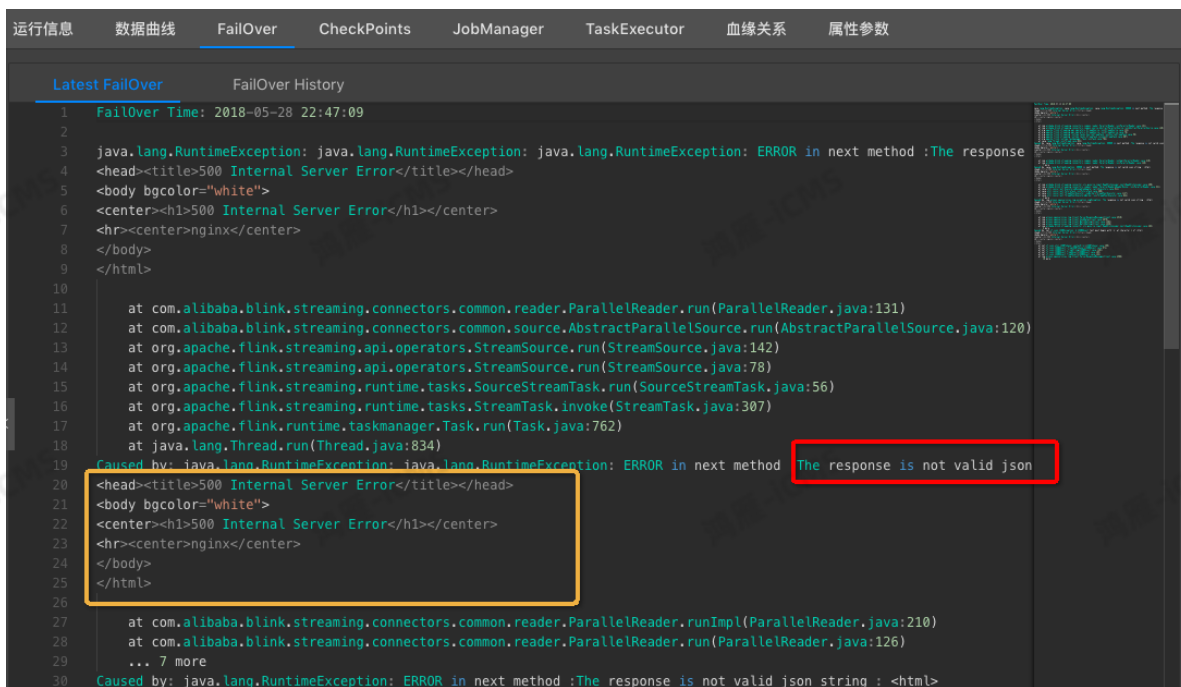
```

ts as case when `datetime` is null
then timestamp '1970-01-01 00:00:00'
else `datetime` end

```

报错：ERROR in next method The response is not valid json

- 报错信息



- 报错原因
SLS发生了网络震荡，导致连接实时计算失败。
- 解决方案
重新**启动**作业。

报错：ERROR in next method AccessKeyId STS.NK1p3aqjkkagcUHiTpc**** is inactive

- 报错信息
-
- ```

1 Latest FailOver FailOver History
2 FailOver Time: 2018-06-11 06:01:47
3 java.lang.RuntimeException: java.lang.RuntimeException: java.lang.RuntimeException:
4 *****
5 ERR_ID:
6 CON-00084
7 CAUSE:
8 Read data from sls error,ERROR in next method :AccessKeyId STS.NK1p3aqjkkagcUHiTpc**** is inactive
9 ACTION:
10 Check the cause, and contact system admin for help if necessary
11 DETAIL:
12 *****
13 *****
14 at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:132)
15 at com.alibaba.blink.streaming.connectors.common.source.AbstractParallelSource.run(AbstractParallelSource.java:180)
16 at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:142)
17 at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:78)
18 at org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:56)
19 at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:307)
20 at org.apache.flink.runtime.taskmanager.Task.run(Task.java:833)
21 at java.lang.Thread.run(Thread.java:834)
22 Caused by: java.lang.RuntimeException: java.lang.RuntimeException:
23 *****
24 ERR_ID:
25 CON-00084
26 CAUSE:
27 Read data from sls error,ERROR in next method :AccessKeyId STS.NK1p3aqjkkagcUHiTpc**** is inactive
28 ACTION:
29 Check the cause, and contact system admin for help if necessary
30 DETAIL:

```

- 报错原因  
SLS的Token过期。
- 解决方案  
设置参数stsUpdateSeconds='3600'。

## 报错：open catalog connection failed

- 报错信息

```
code:[100004], brief info:[open catalog connection failed], context info:[details:
[properties:{"endpoint":"???", "name":"blink", "type":"default"}.exception:Failed to create Hive Metastore cl
ient]]
```

- 报错原因

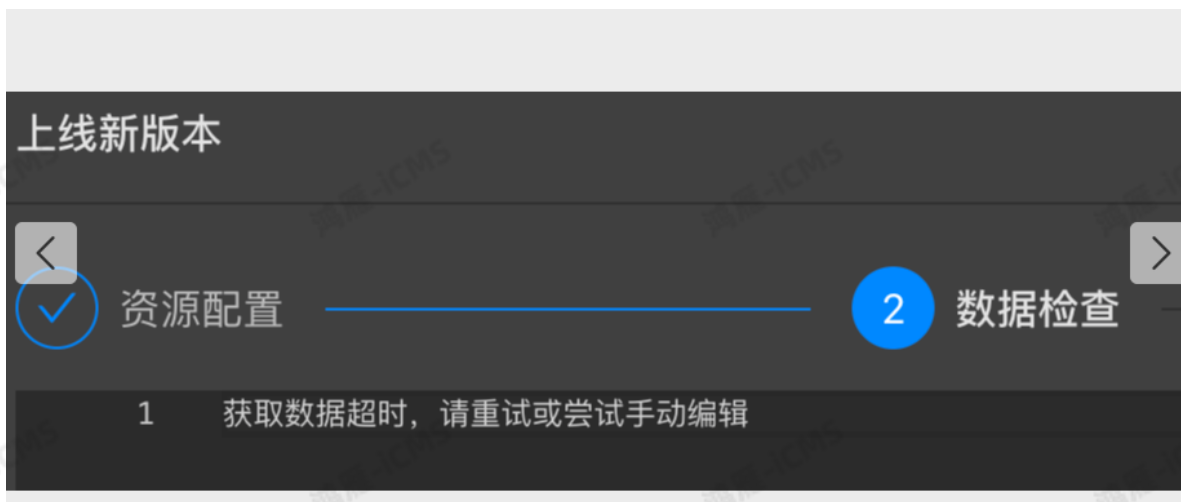
DDL或DML语句中，表名或字段大小写不一致。

- 解决方案

建议您仔细检查并修改表名或字段的大小写。

## 报错：获取数据超时，请重试或尝试手动编辑

- 报错信息



- 报错原因

- 自定义表的参数传入失败。
- 由于Case When判断过多，需要加长编译时间。

- 解决方案

- 检查自定义表的参数是否传入成功。
- 在作业参数中配置blink.job.submit.timeoutInSeconds参数（默认为30s，可以适当增加，但不能超过90s）。

## 警告：WARNING（不影响当前流程）

- 警告详情

○ 详情1

```
1 WARNING(不影响当前流程):
2 Autoconf cannot scale up parallelism of source (vertexId: 0) to 30, because it is
 constrained to the number of source partitions 29.
3 com.alibaba.blink.launcher.metrics.MetricsException: Error fetching metrics for job
 cdnlog.alicacheolog with time between Wed Nov 21 10:35:18 CST 2018 and Wed Nov 21
 10:38:18 CST 2018. Previous job statistics and configuration will be used.
```

○ 详情2

```
1 WARNING:
2 Configuration is not usable. Generate default configuration. Cause:
 The stream nodes in the configuration [1, 2, 3, 4, 5, 6, 7, 8, 10, 11,
 14, 15, 16, 17, 18, 19, 21, 22, 25, 26, 27, 28], do not match those in
 stream graph [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 14, 15, 16, 17, 18, 19].
 The could be caused by updating the application without updating the
 configuration, or incorrect modification to the configuration. Please
 check the configuration and try again
```

● 警告原因

○ 详情1

- 因为上游Shard的限制，所以并发不能扩展到30。
- 因为取不到上一次对应时间点的指标曲线，所以使用了上一次的资源配置，即本次调优实际并没有生效。

○ 详情2

可能是在不更新配置的情况下更新了应用程序，或者对配置的修改不正确，导致与作业资源配置不匹配。

● 解决方案

○ 详情1

不影响作业运行，可以直接上线运行该作业。

○ 详情2

重新获取配置信息后，反复执行上线 > 调优。

### 9.6.4. 启动报错

本文汇总了实时计算启动常见报错与解决方案。

- 报错：No such edge in stream graph:1 -&g
- 报错：Slot allocation request timed out for
- 报错：Submit blink job failed
- 报错：CLI-00000001
- 报错：NoClassDefFoundError或NoSuchMethodError
- 报错：cp path NOT exist or is empty!

**报错：No such edge in stream graph:1 -&g**

- 报错信息

从Blink2.x版本升级到Blink3.x版本时，启动报错。

```
The program finished with the following exception:

org.apache.flink.client.program.ProgramInvocationException: The main method caused an error.
 at org.apache.flink.client.program.PackagedProgram.callMainMethod(PackagedProgram.java:607)
 at org.apache.flink.client.program.PackagedProgram.invokeInteractiveModeForExecution(PackagedProgram.java:466)
 at org.apache.flink.client.program.OptimizerPlanEnvironment.getOptimizedPlan(OptimizerPlanEnvironment.java:94)
 at org.apache.flink.client.program.PackagedProgramUtils.createJobGraph(PackagedProgramUtils.java:76)
 at org.apache.flink.client.cli.CliFrontend.runProgram(CliFrontend.java:239)
 at org.apache.flink.client.cli.CliFrontend.run(CliFrontend.java:216)
 at org.apache.flink.client.cli.CliFrontend.parseParameters(CliFrontend.java:1055)
 at org.apache.flink.client.cli.CliFrontend.lambda$main$9(CliFrontend.java:1131)
 at java.security.AccessController.doPrivileged(Native Method)
 at javax.security.auth.Subject.doAs(Subject.java:422)
 at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1682)
 at org.apache.flink.runtime.security.HadoopSecurityContext.runSecured(HadoopSecurityContext.java:41)
 at org.apache.flink.client.cli.CliFrontend.main(CliFrontend.java:1131)
Caused by: java.lang.RuntimeException: No such edge in stream graph: 1 -> 2
 at org.apache.flink.streaming.api.graph.StreamGraph.getStreamEdges(StreamGraph.java:731)
 at com.alibaba.blink.launcher.autoconfig.StreamGraphConfigurer.configure(StreamGraphConfigurer.java:59)
 at com.alibaba.blink.launcher.JobLauncher.execute(JobLauncher.java:535)
 at com.alibaba.blink.launcher.JobLauncher.runStream(JobLauncher.java:469)
 at com.alibaba.blink.launcher.JobLauncher.main(JobLauncher.java:242)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.flink.client.program.PackagedProgram.callMainMethod(PackagedProgram.java:590)
 ... 12 more
```

- 报错原因

Blink2.x和Blink3.x的PLAN JSON不兼容。

- 解决方案

您需要重新获取配置资源后，重新上线并启动作业。详情请参见 [上线](#)和[启动](#)。

## 报错：Slot allocation request timed out for

- 报错信息

```
Message

java.util.concurrent.TimeoutException: Slot allocation request timed out for ad466a9c8f1a82b48e9d81df3476cb7e
 at org.apache.flink.runtime.instance.SlotPool.checkTimeoutSlotAllocation(SlotPool.java:460)
 at org.apache.flink.runtime.instance.SlotPool.access$500(SlotPool.java:91)
 at org.apache.flink.runtime.instance.SlotPool$3.run(SlotPool.java:442)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.handleRunAsync(AkkaRpcActor.java:273)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.handleMessage(AkkaRpcActor.java:131)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.access$000(AkkaRpcActor.java:72)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor$1.apply(AkkaRpcActor.java:110)
 at akka.actor.ActorCell$$anonfun$become$1.applyOrElse(ActorCell.scala:534)
 at akka.actor.Actor$class.aroundReceive(Actor.scala:467)
 at akka.actor.UntypedActor.aroundReceive(UntypedActor.scala:97)
 at akka.actor.ActorCell.receiveMessage(ActorCell.scala:516)
 at akka.actor.ActorCell.invoke(ActorCell.scala:487)
 at akka.dispatch.Mailbox.processMailbox(Mailbox.scala:238)
 at akka.dispatch.Mailbox.run(Mailbox.scala:220)
 at akka.dispatch.ForkJoinExecutorConfigurator$AkkaForkJoinTask.exec(AbstractDispatcher.scala:397)
 at scala.concurrent.forkjoin.ForkJoinTask.doExec(ForkJoinTask.java:260)
 at scala.concurrent.forkjoin.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1339)
 at scala.concurrent.forkjoin.ForkJoinPool.runWorker(ForkJoinPool.java:1979)
 at scala.concurrent.forkjoin.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:107)
```

• 报错原因

在作业CU不足时，无法分配资源，导致请求超时。

• 解决方案

重新设置作业的CU值。如果您项目中的CU不足，请在 项目管理中对项目进行扩容，详情请参见 [变配指导](#)和[续费指导](#)。

### 报错：Submit blink job failed

• 报错信息

```

数据曲线 FailOver CheckPoints JobManager TaskExecutor Configuration 血缘关系 代码配置
run instance failed, 操作错误: 系统处理BLINK作业异常。
错误原因: Submit blink job failed, name: user_day_activity_by_namespace, code:30010, msg:code:[30010], brief info:[error occur while app], context info:[details:[任务提交超时, 请检查详细日志输出. (submit app exceed time, maybe consider to tune up submit timeout)
java.lang.Exception:
2018-01-04 17:09:04,241 INFO org.apache.flink.yarn.cli.FlinkYarnCLI - No path for the flink jar passed. Us
the location of class org.apache.flink.yarn.YarnClusterDescriptorV2 to locate the jar
2018-01-04 17:09:04,294 INFO org.apache.hadoop.yarn.client.RMPProxy - Connecting to ResourceManager at
b21d11599.cloud.et2.tbsite.net/11.219.228.171:8050
Cluster started: Yarn cluster with application id null
Using address localhost:0 to connect to JobManager.
JobManager web interface address http://
Using the parallelism provided by the remote cluster (0). To use another parallelism, set it at the ./bin/flink client.
Starting execution of program
2018-01-04 17:09:04,422 INFO org.apache.flink.yarn.YarnClusterClientV2 - Starting program in interactive mode
[streamEnv] update k=blink.checkpoint.external.path,
v=hdfs://et2cloudprod/blink_rest_server/root.syncstore/user_day_activity_by_namespace/external_cp
[streamEnv] update k=state.backend.rocksdb.ttl.ms, v=129600000
[streamEnv] update k=state.backend.rocksdb.checkpointpath,
v=hdfs://et2cloudprod/blink_rest_server/root.syncstore/user_day_activity_by_namespace/rocksdb_cp
[streamEnv] update k=stsAccesskey, v=v91V042KRCL7mFEYfLP02INwSv71m0Jz18RUsNbRzFM=
[streamEnv] update k=stsRegionId, v=cn-shanghai
[streamEnv] update k=__inner_rg_sample_data_conf_accessKey__, v=prc1bmIam4hRyPQ9Q66oPZd9gidXDY
[streamEnv] update k=stsAccessId, v=9juusi0kMFGRs8c
[streamEnv] update k=__inner_projectName__, v=syncstore
[streamEnv] update k=blink.checkpoint.mode, v=EXACTLY_ONCE
[streamEnv] update k=blink.checkpoint.external.cleanup, v=RETAIN_ON_CANCELLATION


```

• 报错原因

修改了SQL代码，但未重新生成JSON资源配置。

• 解决方案

修改SQL代码后，需要重新生成JSON配置。详情请参见 [上线](#)。

 说明 如果只修改了资源参数，则不需要重新生成JSON配置。

### 报错：CLI-00000001

• 报错信息

```

ERR_ID
CLI-00000001
CAUSE:
AutoConf:an error occurred while applying the resource configuration file.
Node or edge number in stream graph and json file not matched,please generate a new json.
ACTION:
The reason may be that the content of the SQL has been modified or the blink version number has been
changed.
Please regenerate the resource configuration file.
DETAIL:

```

```
Caused by: com.alibaba.blink.launcher.autoconfig.UnexpectedConfigurationException:

ERP_ID:
 CLI-00000001
CAUSE:
 AutoConf: an error occurred while applying the resource configuration file.
 Node or edge number in stream graph and json file not matched, please generate a new json.
ACTION:
 The reason may be that the content of the SQL has been modified or the blink version number has been changed.
 Please regenerate the resource configuration file.
DETAIL:

at com.alibaba.blink.launcher.autoconfig.StreamGraphConfigurer.configure(StreamGraphConfigurer.java:44)
at com.alibaba.blink.launcher.JobLauncher.execute(JobLauncher.java:547)
at com.alibaba.blink.launcher.JobLauncher.runStream(JobLauncher.java:481)
at com.alibaba.blink.launcher.JobLauncher.main(JobLauncher.java:254)
```

- 报错原因  
更改SQL版本后，导致新SQL的PLAN JSON和资源配置不兼容。

- 解决方案  
i. 在实时计算控制台的开发页面，单击右侧的资源配置。



- ii. 单击资源配置右上角切换视图，进入配置信息页面。
- iii. 删除所有JSON信息。

```
1 {
2 "parameters": {
3 "blink.target_resource_units": 5,
4 "blink.metrics.job.name": " ",
5 "blink.metrics.start.time.ts": 1566814528851,
6 "blink.metrics.end.time.ts": 1566818128851,
7 "blink.metrics.start.time": "2019-08-26T18:15:28+08:00",
8 "blink.metrics.end.time": "2019-08-26T19:15:28+08:00"
9 },
10 "resourceSettings": {
11 "blink.resource.unit.per.worker": 1,
12 "blink.resource.unit.granularity": 0.1,
13 "blink.metrics.endpoint": "https://100.118.129.166:9000",
14 "blink.resource.allocation.jm.min.cpu.cores": 0.2,
15 "blink.resource.allocation.jm.min.memory.mb": 512
16 },
17 "autoConfig": {
18 "goal": {
19 "maxResourceUnits": 10000,
20 "targetResourceUnits": 5
21 },
22 "result": {
23 "attemptId": 4,
24 "attempts": 4,
25 "attemptsCurrentPlan": 4,
26 "statusMessage": "Unable to get running metrics, continue to use last configuration.",
27 "scalingAction": "ScaleToTargetResource",
28 "allocatedResourceUnits": 0.7800000000000001,
29 "allocatedCpuCores": 0.7800000000000001,
30 "allocatedMemoryInMB": 2866,
31 "messages": [
32 "com.alibaba.blink.launcher.metrics.MetricsException: No metric result for [redacted] 7:48:00 CST 2019-Mon Aug 26 19:15:20 CST 2019/parserTps.rate/AVG/
33 container_type=0/false]. Previous job statistics and configuration will be used."
34],
35 "history": {
36 "1": {
37 "attemptId": 1,
38 "statusMessage": "New job, using default configuration."
39 },
40 "2": {
41 "attemptId": 2,
42 "statusMessage": "Unable to get running metrics, continue to use last configuration."
43 }
44 }
45 }
46 }
47 }
```

- iv. 在资源配置右上角，单击配置新增操作 > 重新获取配置信息。
- v. 在资源配置右上角，单击配置新增操作 > 应用当前配置。
- vi. 重新上线作业。



说明 在重新上线时，请选择手动资源配置。

### 报错：NoClassDefFoundError或NoSuchMethodError

- 报错信息

java.lang.NoSuchMethodError

java.lang.NoClassDefFoundError

- 报错原因

依赖包冲突。

- 解决方案

请参见[概述](#)和[Datastream开发](#)中的POM文件，配置依赖。

② 说明

- POM依赖的版本需要和开发页面右下角的版本保持一致。
- Blink相关的依赖请使用 `<scope>provided</scope>` 的方式进行打包。
- 自定义的依赖请使用Shade的方式进行打包，避免与Blink底层依赖发生冲突。

## 报错：cp path NOT exist or is empty!

- 报错信息

作业暂停恢复时，报错如下。

```
Submit job for the [1] times,last error occurred at [Tue Nov 10 00:00:00 CST 2020],lastErrorMessage is as follows:
run instance failed, BLINK error: bayes submit job failed.job name:[random_print], brs errorCode:[30026], brs errorMessage:
code:[30026], brief info:[hdfs://a****/]ification failed], context info:[details:cp path NOT exist or is empty!]
```

- 报错原因

在作业Checkpoint未成功时，进行暂停恢复操作，导致作业恢复时，无法查询到对应的HDFS路径。

- 解决方案

您需要停止再启动作业，详情请参见[停止和启动](#)。

## 9.6.5. 运行Failover

本文汇总了实时计算Flink版运行Failover常见报错与解决方案。

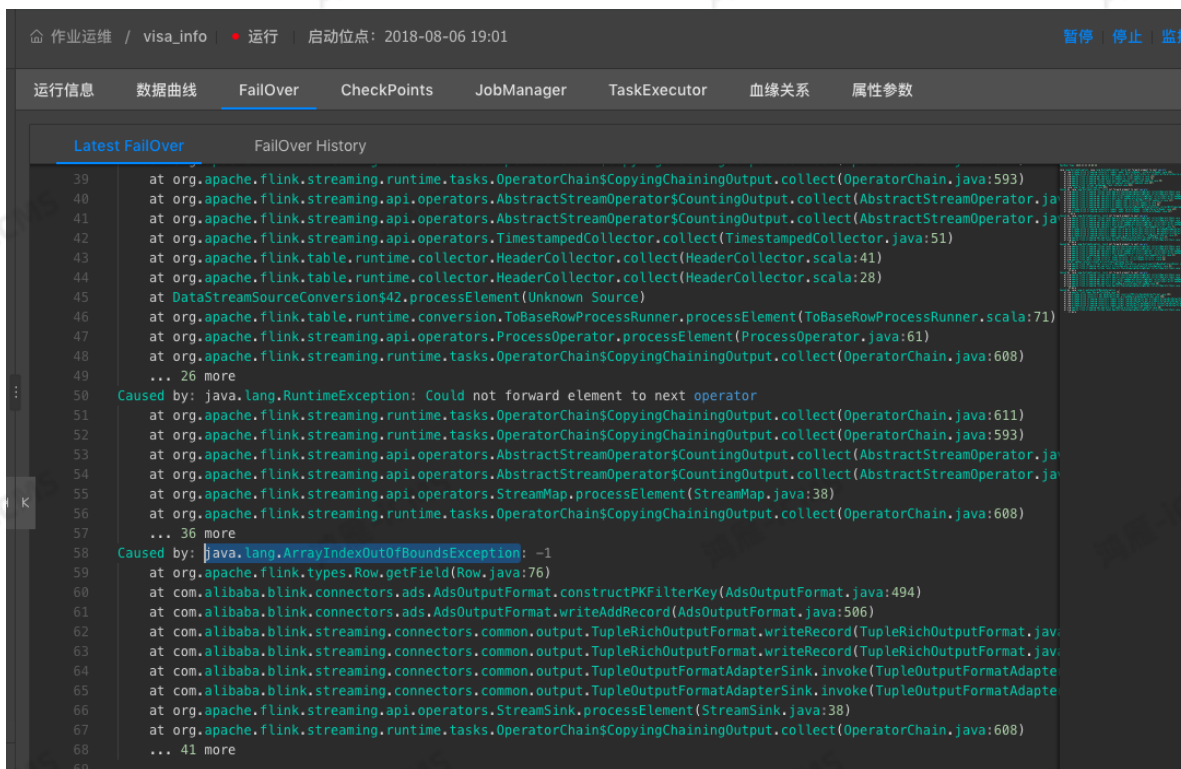
- 报错： `java.lang.ArrayIndexOutOfBoundsException -1`
- 报错： `java.util.concurrent.TimeoutException`
- 报错： `the message body size over max value`
- 报错： `Caused by: org.codehaus.janino.JaninoRuntimeException: Code of method...64KB`
- 报错： `Caused by: java.net.SocketException: Connection reset`
- 报错： `Seek out of range`
- 报错： `pool.GetConnectionTimeoutException`
- 报错： `the last packet sent successfully to the server was milliseconds ago`
- 报错： `Incorrect String value:'\xF0\x9F...'for column`
- 报错： `Some executions status are not reported and failing the job now`
- 报错： `The heartbeat of TaskManager with id container...timed out`
- 报错： `cause by:java.lang.IndexOutOfBoundsException`
- 报错： `No pooled slot available and request to ResourceManager for new slot failed`



- 报错：Ots attribute key type error, table ots\_sink column key uv
- 报错：OutOfMemory
- 报错：OutOfMemoryError: Java heap space
- 报错：java.lang.StackOverflowError
- 报错：Suspend the old job to update it
- 报错：MySQLSyntaxErrorException: Unknown column
- 报错：TaskManager lost/killed
- 报错：The sts token has been expired
- 报错：OTSPParameterInvalid, Message: Invalid update row request: missing cells in request
- 报错：There is an invalid character in tagvalue
- 报错：Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLTransactionRollbackException: Deadlock found when trying to get lock; try restarting transaction
- 报错：Value '0000-00-00 00:00:00' can not be represented as java.sql.Timestamp
- 报错：Failover:offset (0) + length (8) exceed the capacity of the array: 1
- 作业启动一段时间后报错停止，Failover里出现报错信息，应该如何处理？

### 报错：java.lang.ArrayIndexOutOfBoundsException -1

- 报错信息



```
作业运维 / visa_info · 运行 | 启动位点: 2018-08-06 19:01 | 暂停 | 停止 | 监控
运行信息 数据曲线 FailOver CheckPoints JobManager TaskExecutor 血缘关系 属性参数
Latest FailOver FailOver History
39 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
40 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:593)
41 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:593)
42 at org.apache.flink.streaming.api.operators.TimestampedCollector.collect(TimestampedCollector.java:51)
43 at org.apache.flink.table.runtime.collector.HeaderCollector.collect(HeaderCollector.scala:41)
44 at org.apache.flink.table.runtime.collector.HeaderCollector.collect(HeaderCollector.scala:28)
45 at DataStreamSourceConversion$42.processElement(Unknown Source)
46 at org.apache.flink.table.runtime.conversion.ToBaseRowProcessRunner.processElement(ToBaseRowProcessRunner.scala:71)
47 at org.apache.flink.streaming.api.operators.ProcessOperator.processElement(ProcessOperator.java:61)
48 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
49 ... 26 more
50 Caused by: java.lang.RuntimeException: Could not forward element to next operator
51 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:611)
52 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
53 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:593)
54 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:593)
55 at org.apache.flink.streaming.api.operators.StreamMap.processElement(StreamMap.java:38)
56 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
57 ... 36 more
58 Caused by: java.lang.ArrayIndexOutOfBoundsException: -1
59 at org.apache.flink.types.Row.getField(Row.java:76)
60 at com.alibaba.blink.connectors.ads.AdsOutputFormat.constructPKFilterKey(AdsOutputFormat.java:494)
61 at com.alibaba.blink.connectors.ads.AdsOutputFormat.writeAddRecord(AdsOutputFormat.java:506)
62 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:506)
63 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:506)
64 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:38)
65 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:38)
66 at org.apache.flink.streaming.api.operators.StreamSink.processElement(StreamSink.java:38)
67 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
68 ... 41 more
```

- 报错原因  
实时计算Flink版引用的外部数据源主键字段大小写敏感。
- 解决方案

修改结果表的主键字段大小写，和外部数据源表保持一致。例如，如果分析型数据库MySQL版结果表的字段是大写的，则在实时计算Flink版的DDL语句中，结果表的主键字段也必须是大写的。

## 报错：java.util.concurrent.TimeoutException

- 报错信息

AutoScale启动时，报错如下。

```
java.util.concurrent.TimeoutException
 at org.apache.flink.runtime.concurrent.FutureUtils$Timeout.run(FutureUtils.java:834)
 at java.util.concurrent.Executors$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

作业上线时指定的CU数，不足以支持AutoScale进行迭代。

- 解决方案

在作业开启上线时，如果开启AutoScale，则提高到最大CU数。详情请参见[AutoScale自动配置调优](#)。

## 报错：the message body size over max value

- 报错信息

```
ERR_ID:
 CON-00066
CAUSE:
 Can not write to metaq/mq, retry attempts 3, errMsg Retrying cost [4002] ms, exceed timeout threshold [3] *
 [1000] ms. Error Message: [the message body size over max value, MAX: 4194304]
ACTION:
 Check the cause, and contact system admin for help if necessary
DETAIL:

 at com.alibaba.blink.streaming.connectors.common.exception.ErrorUtils.getException(ErrorUtils.java:39)
 at com.alibaba.blink.streaming.connectors.common.exception.ErrorUtils.throwException(ErrorUtils.java:53)
 at com.alibaba.blink.streaming.connector.metaq.MetaQOutputFormat.writeAddRecord(MetaQOutputFormat.java:193)
 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord
 (TupleRichOutputFormat.java:50)
 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord
 (TupleRichOutputFormat.java:34)
 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke
 (TupleOutputFormatAdapterSink.java:71)
 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke
 (TupleOutputFormatAdapterSink.java:34)
```

- 错误原因

在数据写入MQ结果表时，MQ消息超过了Body的长度限制。详情请参见[消息队列RocketMQ版消息Body的长度限制是多少？](#)。

- 解决方案

调整MQ消息写入Body的长度，限制在Body长度范围内。

## 报错：Caused by: org.codehaus.janino.JaninoRuntimeException: Code of method...64KB

- 报错信息

```
Caused by: java.lang.RuntimeException: Compiling "StreamExecCalcRule$6109": Code of method "calcProjectApply_1$(LStreamExecCalcRule$6109;Lorg/apache/flink/table/dataformat/BaseRow;)V" of class "StreamExecCalcRule$6109" grows beyond 64 KB
 at org.codehaus.janino.UnitCompiler.compileUnit(UnitCompiler.java:361)
 at org.codehaus.janino.SimpleCompiler.cook(SimpleCompiler.java:234)
 at org.codehaus.janino.SimpleCompiler.compileToClassLoader(SimpleCompiler.java:446)
 at org.codehaus.janino.SimpleCompiler.cook(SimpleCompiler.java:213)
 at org.codehaus.janino.SimpleCompiler.cook(SimpleCompiler.java:204)
 at org.codehaus.commons.compiler.Cookable.cook(Cookable.java:80)
 at org.codehaus.commons.compiler.Cookable.cook(Cookable.java:75)
 at org.apache.flink.table.codegen.CodeGenUtils$.compile(CodeGenUtils.scala:997)
 ... 18 more
Caused by: org.codehaus.janino.JaninoRuntimeException: Code of method "calcProjectApply_1$(LStreamExecCalcRule$6109;Lorg/apache/flink/table/dataformat/BaseRow;)V" of class "StreamExecCalcRule$6109" grows beyond 64 KB
 at org.codehaus.janino.CodeContext.makeSpace(CodeContext.java:974)
 at org.codehaus.janino.CodeContext.write(CodeContext.java:884)
 ...
```

- 报错原因

JVM对某个方法的字节编码长度限制为64KB。而此作业中，单个字段的字节编码长度就已超过了64KB。

- 解决方案

使用 **自定义标量函数（UDF）** 方法修改此作业。

## 报错：Caused by: java.net.SocketException: Connection reset

- 报错信息

```
The last packet successfully received from the server was 3,991 milliseconds ago. The last packet sent suc
cessfully to the server was 3,992 milliseconds ago.
 at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
 at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
 at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java
:45)
 at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
 at com.mysql.jdbc.Util.handleNewInstance(Util.java:425)
 at com.mysql.jdbc.SQLError.createCommunicationsException(SQLError.java:989)
 at com.mysql.jdbc.MysqlIO.send(MysqlIO.java:3743)
 at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:2506)
 at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2677)
 at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2545)
 at com.mysql.jdbc.ConnectionImpl.setAutoCommit(ConnectionImpl.java:4842)
 at com.alibaba.druid.pool.DruidPooledConnection.setAutoCommit(DruidPooledConnection.java:709)
 ... 6 more
Caused by: java.net.SocketException: Connection reset
 at java.net.SocketOutputStream.socketWrite(SocketOutputStream.java:113)
 at java.net.SocketOutputStream.write(SocketOutputStream.java:153)
 at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:82)
 at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:140)
 at com.mysql.jdbc.MysqlIO.send(MysqlIO.java:3725)
 ... 11 more
```

• 报错原因

链接超时导致了Failover。

• 解决方案

- i. 检查RDS是否为高可用模式，如果不是，请切换至高可用模式。
- ii. With参数中的 URL 中设置 autoReconnect=true ，如下图所示。

```
cs.com:3306/fotoabledb?autoReconnect=true',
```

如果有多个参数，请用and（&）符号进行连接。

```
type='rds',
url='jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=gbk&autoReconne
```

**报错：Seek out of range**

• 报错信息

```
sage (2017-12-22 09:53:23)
lang.RuntimeException: java.lang.RuntimeException: com.aliyun.datahub.exception.InvalidParameterException: wifiana_test/group_rssi_data_part_tenminutes/0: Seek Out of Range (Status
at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:131)
at com.alibaba.blink.streaming.connectors.common.source.AbstractParallelSource.run(AbstractParallelSource.java:107)
at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:142)
at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:78)
at org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:56)
at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:307)
at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
at java.lang.Thread.run(Thread.java:834)
Caused by: java.lang.RuntimeException: com.aliyun.datahub.exception.InvalidParameterException: wifiana_test/group_rssi_data_part_tenminutes/0: Seek Out of Range (Status Code: 400; Error
at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.runImpl(ParallelReader.java:210)
at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:126)
... 7 more
Caused by: com.aliyun.datahub.exception.InvalidParameterException: wifiana_test/group_rssi_data_part_tenminutes/0: Seek Out of Range (Status Code: 400; Error Code: InvalidParameter; Re
at com.aliyun.datahub.model.serialize.JsonCursorParser.parse(JsonCursorParser.java:31)
at com.aliyun.datahub.model.serialize.GetCursorResult.JsonDeser.deserialize(GetCursorResult.JsonDeser.java:19)
at com.aliyun.datahub.model.serialize.GetCursorResult.JsonDeser.deserialize(GetCursorResult.JsonDeser.java:14)
at com.aliyun.datahub.DatahubClient.getCursor(DatahubClient.java:627)
at com.aliyun.datahub.DatahubClient.getCursor(DatahubClient.java:699)
at com.alibaba.blink.streaming.connectors.datahub.input.RawDatahubRecordReader.seek(RawDatahubRecordReader.java:332)
at com.alibaba.blink.streaming.connectors.datahub.input.RawDatahubRecordReader.seek(RawDatahubRecordReader.java:51)
at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader$ReaderRunner.run(ParallelReader.java:336)
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1147)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
... 3 more
```

- 原因分析
  - 实时计算Flink版处理数据的速率小于数据进入DataHub的速率，会造成数据堆积，延时增加。当延时大于DataHub生命周期时，实时计算Flink版读取的数据已经过期，最终导致系统报错。
  - 您的本地设备系统时间大于DataHub所在机器的时间。实时计算Flink版作业启动位点默认时间为您本地设备的系统时间，如果启动位点在DataHub机器时间之后，则会导致 Seek out of Range 的报错。
- 解决方案
  - 对实时计算Flink版的性能进行调优，使实时计算Flink版处理数据的速率大于等于数据进入DataHub的速率。调优方法请参见：
    - [高性能Flink SQL优化技巧](#)
    - [AutoConf自动配置调优](#)
    - [AutoScale自动配置调优](#)
    - [手动配置调优](#)
  - 调整用户本地设备系统时间，使其和DataHub所在机器时间保持一致。

## 报错：pool.GetConnectionTimeoutException

- 报错信息

```
Message (2018-03-16 11:54:19)

java.lang.RuntimeException: Cannot create connection to RDS.
 at com.alibaba.blink.connectors.rds.dim.RdsRowFetcher.connectToTable(RdsRowFetcher.java:67)
 at com.alibaba.blink.connectors.mysql.dim.MySqlRowFetcherBase.reconnectToTable(MySqlRowFetcherBase.java:147)
 at com.alibaba.blink.connectors.mysql.dim.MySqlRowFetcherBase.open(MySqlRowFetcherBase.java:133)
 at org.apache.flink.api.common.functions.util.FunctionUtils.openFunction(FunctionUtils.java:36)
 at com.alibaba.blink.table.runtime.JoinTableProcessRunner.open(JoinTableProcessRunner.scala:57)
 at com.alibaba.blink.table.runtime.JoinTableWithCalcProcessRunner.open(JoinTableWithCalcProcessRunner.scala:46)
 at org.apache.flink.api.common.functions.util.FunctionUtils.openFunction(FunctionUtils.java:36)
 at org.apache.flink.streaming.api.operators.AbstractUdfStreamOperator.open(AbstractUdfStreamOperator.java:156)
 at org.apache.flink.streaming.api.operators.ProcessOperator.open(ProcessOperator.java:51)
 at org.apache.flink.streaming.runtime.tasks.StreamTask.openAllOperators(StreamTask.java:445)
 at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:297)
 at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
 at java.lang.Thread.run(Thread.java:834)
Caused by: com.alibaba.druid.pool.GetConnectionTimeoutException: wait millis 16429, active 30, maxActive 30, creating 0
 at com.alibaba.druid.pool.DruidDataSource.getConnectionInternal(DruidDataSource.java:1329)
 at com.alibaba.druid.pool.DruidDataSource.getConnectionDirect(DruidDataSource.java:1148)
 at com.alibaba.druid.pool.DruidDataSource.getConnection(DruidDataSource.java:1128)
 at com.alibaba.druid.pool.DruidDataSource.getConnection(DruidDataSource.java:1118)
 at com.alibaba.blink.connectors.rds.dim.RdsRowFetcher.connectToTable(RdsRowFetcher.java:63)
 ... 12 more
```

• 报错原因

Pool的链接池最大值为30个，而您的链接数大于30，导致数据写入失败。

• 解决方案

调大维表或流表JOIN节点的 Core参数。例如，Core的默认值是0.1，可以调成0.25或1。详情请参见 [手动配置调优](#)。

### 报错：the last packet sent successfully to the server was milliseconds ago

• 报错信息

- the last packet successfully received from the server was milliseconds ago
- the last packet sent successfully to the server was milliseconds ago

• 报错原因

如果RDS的 net\_read\_timeout 和 net\_write\_timeout 参数设置过小，在网络条件较差或客户端处理Block耗时较长时，会导致RDS连接实时计算Flink版提示连接超时的报错。

| 参数                                   | 作用                                                               |
|--------------------------------------|------------------------------------------------------------------|
| net_read_timeout 和 net_write_timeout | 服务器端等待客户端发送的网络包和发送给客户端网络包的超时时间，该参数仅对TCP/IP链接并且是Activity状态下的线程有效。 |


• 解决方案

调整 net\_read\_timeout 参数和 net\_write\_timeout 参数值，操作步骤如下：

- i. 登录[RDS管理控制台](#)。
- ii. 在页面左上角，选择实例所在区域。
- iii. 单击目标实例ID。

- iv. 在左侧导航栏中单击 **参数设置**。
- v. 在可修改参数页签，查找 `net_read_timeout` 和 `net_write_timeout`。

 **说明** 可以使用ctrl+f或command+f的组合键在页面中搜索您需要修改的参数。

- vi. 单击 `net_read_timeout` 和 `net_write_timeout` 后的 。
- vii. 输入目标值，并单击确定。
- viii. 单击可修改参数页签右上角的提交参数。
- ix. 在弹出的对话框中，单击确定，完成参数修改。


## 报错：Incorrect String value: '\xF0\x9F...'for column

- 报错信息

```
系统错误 内部错误：在批处理写入时发生错误 sql template sql_INSERT INTO
at com.alibaba.blink.connectors.rds.RdsExceptionUtil.getBlinkException(RdsExceptionUtil.jav
at com.alibaba.blink.connectors.rds.RdsOutputFormat.execBatchAdd(RdsOutputFormat.java:599)
at com.alibaba.blink.connectors.rds.RdsOutputFormat.batchWrite(RdsOutputFormat.java:447)
at com.alibaba.blink.connectors.rds.RdsOutputFormat.sync(RdsOutputFormat.java:399)
at com.alibaba.blink.connectors.rds.RdsOutputFormat$1.run(RdsOutputFormat.java:284)
at java.util.TimerThread.mainLoop(Timer.java:555)
at java.util.TimerThread.run(Timer.java:585)
Caused by: java.sql.BatchUpdateException: Incorrect string value: '\xF0\x9F\x92\xA4' for column '...' at row 42
at sun.reflect.GeneratedConstructorAccessor14.newInstance(ConstructorSource)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
at com.mysql.jdbc.Util.handleNewInstance(Util.java:425)
at com.mysql.jdbc.Util.getInstance(Util.java:408)
at com.mysql.jdbc.SQLException.createBatchUpdateException(SQLException.java:1162)
at com.mysql.jdbc.PreparedStatement.executeBatchedInserts(PreparedStatement.java:1582)
at com.mysql.jdbc.PreparedStatement.executeBatchInternal(PreparedStatement.java:1248)
at com.mysql.jdbc.StatementImpl.executeBatch(StatementImpl.java:958)
at com.alibaba.druid.pool.DruidPooledPreparedStatement.executeBatch(DruidPooledPreparedStatement.java:559)
at com.alibaba.blink.connectors.rds.RdsOutputFormat.execBatchAdd(RdsOutputFormat.java:482)
... 5 more
Caused by: java.sql.SQLException: Incorrect string value: '\xF0\x9F\x92\xA4' for column '...' at row 42
at com.mysql.jdbc.SQLException.createSQLException(SQLException.java:964)
at com.mysql.jdbc.MySQLIO.checkErrorPacket(MySQLIO.java:3978)
at com.mysql.jdbc.MySQLIO.checkErrorPacket(MySQLIO.java:3986)
at com.mysql.jdbc.MySQLIO.sendCommand(MySQLIO.java:2524)
at com.mysql.jdbc.MySQLIO.sqlQueryDirect(MySQLIO.java:2677)
at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2549)
at com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1861)
at com.mysql.jdbc.PreparedStatement.executeUpdateInternal(PreparedStatement.java:2073)
at com.mysql.jdbc.PreparedStatement.executeUpdateInternal(PreparedStatement.java:2089)
at com.mysql.jdbc.PreparedStatement.executeLargeUpdate(PreparedStatement.java:5898)
at com.mysql.jdbc.PreparedStatement.executeBatchedInserts(PreparedStatement.java:1543)
... 9 more
```

- 错误原因  
数据中存在Emoji表情，数据库编码不能解析。

- 解决方案  
您需要在RDS控制台对应实例的 **参数设置** 中，修改 `character_set_server` 参数值为 `utf8mb4`。此外，数据库和表的字符集也要同样设置为 `utf8mb4`。

 **说明**

- `character_set_server` 参数修改后，需要重启实例才能生效。
- 如果需要将 `\xF0\x9F...` 字符串存入数据库，则将反斜线 (`\`) 替换为双反斜线 (`\\`)。

## 报错：Some executions status are not reported and failing the job now

- 报错信息  
Some executions status are not reported and failing the job now.

- 错误原因

因为作业没有启动成功，所以没有记录启动时间。默认启动时间为UNIX起始时间 1970-01-01 08:00:00。

● 解决方案

请重新[上线](#)并启动作业。

## 报错：The heartbeat of TaskManager with id container...timed out

● 报错信息

```

Latest FailOver FailOver History
1 FailOver Time: 2019-01-23 16:31:03
2
3 java.util.concurrent.TimeoutException: Heartbeat of TaskManager with id container_e08_1548199152109_0107_01_000018
 timed out.
4 at org.apache.flink.runtime.jobmaster.JobMaster$TaskManagerHeartbeatListener.notifyHeartbeatTimeout
 (JobMaster.java:1545)
5 at org.apache.flink.runtime.heartbeat.HeartbeatManagerImpl$HeartbeatMonitor.run(HeartbeatManagerImpl.java:358)
6 at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
7 at java.util.concurrent.FutureTask.run(FutureTask.java:266)
8 at org.apache.flink.runtime.concurrent.akka.ActorSystemScheduledExecutorAdapter$ScheduledFutureTask.run
 (ActorSystemScheduledExecutorAdapter.java:154)
9 at akka.dispatch.TaskInvocation.run(AbstractDispatcher.scala:39)
10 at akka.dispatch.ForkJoinExecutorConfigurator$AkkaForkJoinTask.exec(AbstractDispatcher.scala:415)
11 at scala.concurrent.forkjoin.ForkJoinTask.doExec(ForkJoinTask.java:260)
12 at scala.concurrent.forkjoin.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1339)
13 at scala.concurrent.forkjoin.ForkJoinPool.runWorker(ForkJoinPool.java:1979)
14 at scala.concurrent.forkjoin.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:107)
15

```

● 报错原因

- 分布式物理机网络失联。
- Failover的节点对应Taskmanager的内存设置太小，垃圾回收机制会导致心跳超时。

● 解决方案

- 如果此问题出现的不频繁，则不用关注，通常Failover后作业能正常恢复。
- 调大Failover的节点对应Taskmanager的内存值。

## 报错：cause by:java.lang.IndexOutOfBoundsException

● 报错信息

引用MaxCompute维表时，报错如下。

```

22 at org.apache.flink.streaming.api.operators.StreamMap.processElement(StreamMap.java:38)
23 at org.apache.flink.streaming.runtime.tasks.OperatorChainCopyingChainingOutput.collect(OperatorChain.java:608)
24 ... 12 more
25 Caused by: java.lang.RuntimeException: com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException: 821109
26
27 系统错误-内部错误: 在批量写入时发生错误 sql template sql:INSERT INTO `rds_dim` (`d`,`e`,`f`) values (?,?,?) ON DUPLICATE KEY UPDATE e= values(e),f= values(f),preparedStatement
 at com.alibaba.blink.connectors.rds.RdsOutputFormat.writeAddRecord(RdsOutputFormat.java:278)
28
29 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:30)
30 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:34)
31 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:70)
32 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:33)
33 at org.apache.flink.streaming.api.operators.StreamSink.processElement(StreamSink.java:38)
34 at org.apache.flink.streaming.runtime.tasks.OperatorChainCopyingChainingOutput.collect(OperatorChain.java:608)
35 ... 17 more
36 Caused by: com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException: 821109
37
38 系统错误-内部错误: 在批量写入时发生错误 sql template sql:INSERT INTO `rds_dim` (`d`,`e`,`f`) values (?,?,?) ON DUPLICATE KEY UPDATE e= values(e),f= values(f),preparedStatement
 at com.alibaba.blink.connectors.rds.RdsExceptionUtil.getBlinkException(RdsExceptionUtil.java:33)
39
40 at com.alibaba.blink.connectors.rds.RdsOutputFormat.execBatchAdd(RdsOutputFormat.java:509)
41 at com.alibaba.blink.connectors.rds.RdsOutputFormat.batchWrite(RdsOutputFormat.java:447)
42 at com.alibaba.blink.connectors.rds.RdsOutputFormat.sync(RdsOutputFormat.java:399)
43 at com.alibaba.blink.connectors.rds.RdsOutputFormat$1.run(RdsOutputFormat.java:204)
44 at java.util.TimerThread.mainLoop(Timer.java:555)
45 at java.util.TimerThread.run(Timer.java:505)
46 Caused by: java.sql.BatchUpdateException: Unknown column 'd' in 'field list'
47 at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

```

● 报错原因

不支持Schema定义中写入分区列。





```
at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:34)
at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:71)
at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:34)
at org.apache.flink.streaming.api.functions.sink.SinkFunction.invoke(SinkFunction.java:52)
at org.apache.flink.streaming.api.operators.StreamSink.processElement(StreamSink.java:56)
at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingWithOneInputStreamOperatorOutput.pushToOperator(OperatorChain.java:635)
at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingWithOneInputStreamOperatorOutput.collect(OperatorChain.java:612)
at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingWithOneInputStreamOperatorOutput.collect(OperatorChain.java:575)
at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:763)
at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:741)
at DataStreamSinkConversion$50.processElement(Unknown Source)
at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingWithOneInputStreamOperatorOutput.pushToOperator(OperatorChain.java:635)
at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingWithOneInputStreamOperatorOutput.collect(OperatorChain.java:612)
at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingWithOneInputStreamOperatorOutput.collect(OperatorChain.java:575)
at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:763)
at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:741)
at org.apache.flink.table.runtime.util.StreamRecordCollector.collect(StreamRecordCollector.java:44)
at org.apache.flink.table.runtime.aggregate.GroupAggFunction.processElement(GroupAggFunction.scala:152)
at org.apache.flink.table.runtime.aggregate.GroupAggFunction.processElement(GroupAggFunction.scala:44)
at org.apache.flink.table.runtime.KeyedProcessOperator.processElement(KeyedProcessOperator.java:167)
at org.apache.flink.streaming.runtime.io.StreamInputProcessor.processRecordOrMark(StreamInputProcessor.java:286)
at org.apache.flink.streaming.runtime.io.StreamInputProcessor.processInput(StreamInputProcessor.java:231)
at org.apache.flink.streaming.runtime.tasks.OneInputStreamTask.run(OneInputStreamTask.java:112)
at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:348)
at org.apache.flink.runtime.taskmanager.Task.run(Task.java:732)
at java.lang.Thread.run(Thread.java:834)
```

- 报错原因
  - valueColumns参数中包含主键。
  - valueColumns参数中各字段之间存在空格。
- 解决方案
  - 去除设置在valueColumns参数中的主键。
  - 删除valueColumns参数中各字段之间的空格。

## 报错：OutOfMemory

- 报错信息

在线上阶段和调试阶段出现内存溢出（OutOfMemory）的报错。

- 报错原因

- 调试阶段

- 调试分配的内存不足。
- 测试引用的源表过多。
- 数据量过大（每个上传文件里面的测试数据均为1000条以上）。

- 线上阶段

作业的内存分配不足导致内存溢出。

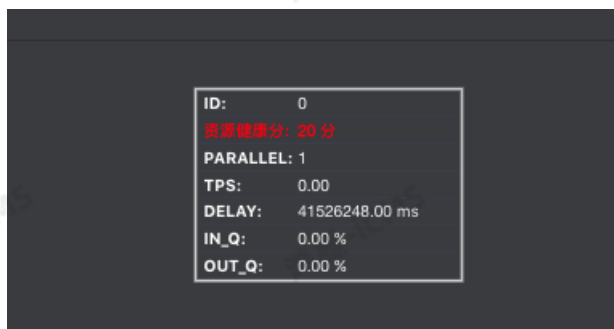
- 解决方案

- 调试阶段

在不影响测试结果情况下，建议您缩减测试数据为每个文件20条左右。

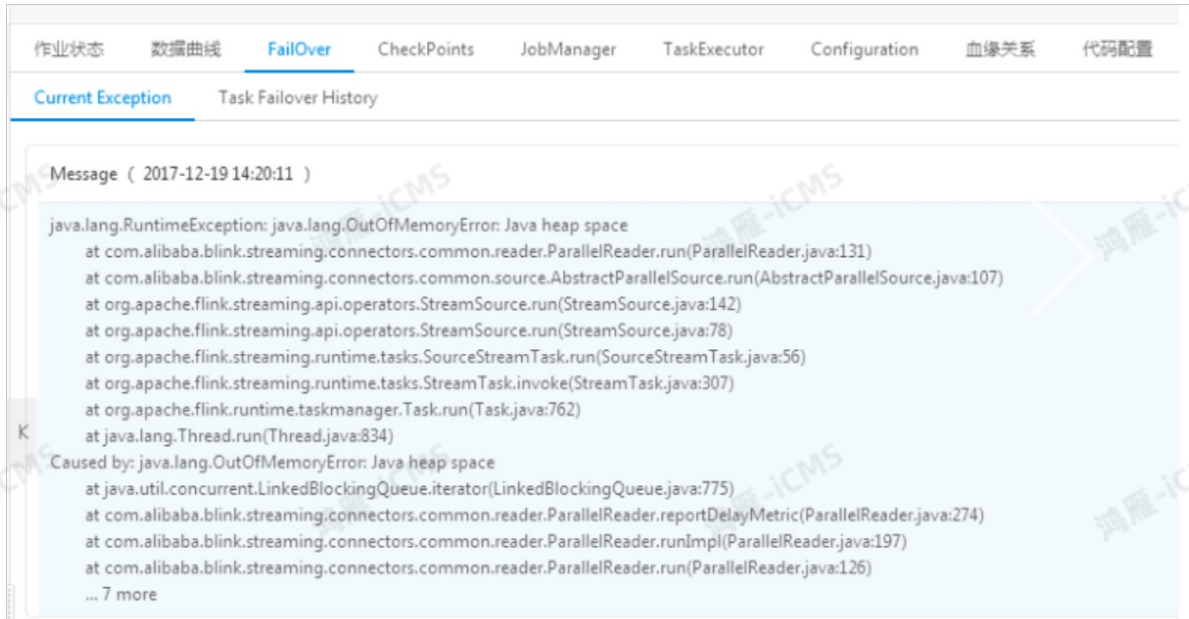
- 线上阶段

在运行拓扑图中查看资源健康分在60分以下的节点，上调该节点的CU数。具体步骤请参看 [手动配置调优](#)。



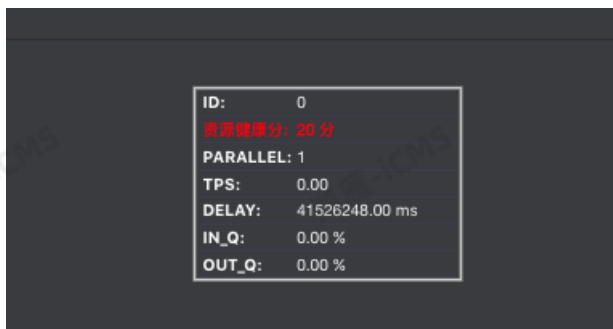
## 报错：OutOfMemoryError: Java heap space

- 报错信息



- 报错原因  
内存不足。
- 解决方案

在运行拓扑图中查看资源健康分在60分以下的节点，上调该节点的CU数。具体步骤请参看 [手动配置调优](#)。



### 报错：java.lang.StackOverflowError

- 报错信息



此报错不影响作业，AutoScale功能完成后，作业即可恢复运行。

## 报错：MySQLSyntaxErrorException: Unknown column

- 报错信息

```
Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column 'd' in 'field list'
```

- 报错原因

Flink SQL中字段的定义和您上传的表字段的定义不一致。

- 解决方案

请您根据Failover界面的报错提示，修改Flink SQL字段的定义，确保和您上传的表字段的定义一致。

## 报错：TaskManager lost/killed

- 报错信息

```
java.lang.Exception: The assigned slot: SimpleSlot (1) - container_e05_1505041177764_25978_01_000033 @ hdpet2mainse011132138130.et2.tbsite.net (dataPort=59317) - ALLOCATED/ALIVE is asked to release from TaskManager: container_e05_1505041177764_25978_01_000033 @ hdpet2mainse011132138130.et2.tbsite.net (dataPort=59317), probably due to TaskManager lost/killed
 at org.apache.flink.runtime.instance.SimpleSlot.releaseSlot(SimpleSlot.java:217)
 at org.apache.flink.runtime.instance.SlotPool.releaseTaskManager(SlotPool.java:699)
 at sun.reflect.GeneratedMethodAccessor44.invoke(Unknown Source)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.handleRpcInvocation(AkkaRpcActor.java:183)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.handleMessage(AkkaRpcActor.java:135)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor.access$000(AkkaRpcActor.java:72)
 at org.apache.flink.runtime.rpc.akka.AkkaRpcActor$1.apply(AkkaRpcActor.java:110)
 at akka.actor.ActorCell$$anonfun$become$1.applyOrElse(ActorCell.scala:534)
 at akka.actor.Actor$class.aroundReceive(Actor.scala:467)
 at akka.actor.UntypedActor.aroundReceive(UntypedActor.scala:97)
 at akka.actor.ActorCell.receiveMessage(ActorCell.scala:516)
 at akka.actor.ActorCell.invoke(ActorCell.scala:487)
 at akka.dispatch.Mailbox.processMailbox(Mailbox.scala:238)
 at akka.dispatch.Mailbox.run(Mailbox.scala:220)
 at akka.dispatch.ForkJoinExecutorConfigurator$AkkaForkJoinTask.exec(AbstractDispatcher.scala:397)
 at scala.concurrent.forkjoin.ForkJoinTask.doExec(ForkJoinTask.java:260)
 at scala.concurrent.forkjoin.ForkJoinPool$WorkQueue.runTask(ForkJoinPool.java:1339)
 at scala.concurrent.forkjoin.ForkJoinPool.runWorker(ForkJoinPool.java:1979)
 at scala.concurrent.forkjoin.ForkJoinWorkerThread.run(ForkJoinWorkerThread.java:107)
```

- 报错原因

由于GROUP BY、JOIN等节点的state\_size默认为0，不会申请Native Memory。当节点的Native Memory没有申请或太小时，就被YARN断开了。

- 解决方案

- 使用[手动配置调优](#)进行资源调优。
- 配置GROUP BY、JOIN等节点的state\_size为1。每个Group，建议Native Memory配置为至少20M。

## 报错：The sts token has been expired

- 报错信息

```
java.lang.RuntimeException: java.lang.RuntimeException: com.aliyun.datahub.exception.NoPermissionException: The sts token has been expired. (Status Code: 403; Error Code: NoPermission; Request ID: 201802230613304db99a0a76d8a13b)
 at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:131)
 at com.alibaba.blink.streaming.connectors.common.source.AbstractParallelSource.run(AbstractParallelSource.java:107)
 at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:142)
 at org.apache.flink.streaming.api.operators.StreamSource.run(StreamSource.java:78)
 at org.apache.flink.streaming.runtime.tasks.SourceStreamTask.run(SourceStreamTask.java:56)
 at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:307)
 at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
 at java.lang.Thread.run(Thread.java:834)
Caused by: java.lang.RuntimeException: com.aliyun.datahub.exception.NoPermissionException: The sts token has been expired. (Status Code: 403; Error Code: NoPermission; Request ID: 201802230613304db99a0a76d8a13b)
 at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.runImpl(ParallelReader.java:210)
 at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader.run(ParallelReader.java:126)
 ... 7 more
Caused by: com.aliyun.datahub.exception.NoPermissionException: The sts token has been expired. (Status Code: 403; Error Code: NoPermission; Request ID: 201802230613304db99a0a76d8a13b)
 at com.aliyun.datahub.model.serialize.JsonErrorParser.parse(JsonErrorParser.java:44)
 at com.aliyun.datahub.model.serialize.GetCursorResultJsonDeser.deserialize(GetCursorResultJsonDeser.java:19)
 at com.aliyun.datahub.model.serialize.GetCursorResultJsonDeser.deserialize(GetCursorResultJsonDeser.java:14)
 at com.aliyun.datahub.DatahubClient.getCursor(DatahubClient.java:627)
 at com.aliyun.datahub.DatahubClient.getCursor(DatahubClient.java:699)
 at com.alibaba.blink.streaming.connectors.datahub.input.RawDatahubRecordReader.seek(RawDatahubRecordReader.java:397)
 at com.alibaba.blink.streaming.connectors.datahub.input.RawDatahubRecordReader.next(RawDatahubRecordReader.java:278)
 at com.alibaba.blink.streaming.connectors.common.reader.ParallelReader$ReaderRunner.run(ParallelReader.java:341)
 at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
 at java.util.concurrent.FutureTask.run(FutureTask.java:266)
 at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1147)
 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)
 ... 1 more
```

- 报错原因

为了保证用户的数据安全，STS Token会有一些的生命周期，系统会自动重新获取Token。

- 解决方案

建议您在With参数中设置 `stsUpdateSeconds='3600'`。

```
WITH (
 type='datahub',
 url='',
 username='',
 password='',
 tableName='',
 stsUpdateSeconds='3600'
)
```

- stsUpdateSeconds主动更新roleArn的时间，时间单位为秒。
- stsUpdateSeconds设置的值，需比各存储中，登录Session的过期时间值小。例如，如果登录Session过期时间设置为6小时，则stsUpdateSeconds参数建议设置为stsUpdateSeconds='10800'。

允许自主管理多因素设备

登录Session过期时间： 小时（有效值：6 - 24）

登录掩码设置：

网络掩码决定哪些IP地址会受到登录控制台的影响，包括密码登录和SSO登录，但使用AccessKey发起的API访问并不受影响。如果指定掩码，子用户必须只能从指定的IP地址进行登录。如果不指定任何掩码，登录控制台功能将适用于整个网络。当需要配置多个掩码时，请使用分号来分隔掩码，例如：  
42.120.66.0/24; 42.120.74.98

## 报错：OTSPParameterInvalid, Message: Invalid update row request: missing cells in request

- 报错信息

数据写入OTS结果表时，报错如下。


```
Code: OTSPParameterInvalid, Message: Invalid update row request: missing cells in request.
```

- 报错原因

通常是因为valueColumns整行数据都为NULL。

- 排查思路

设置结果表Type为Print，查找valueColumns整行全部都是NULL的数据。

 说明 如果数据量大时，您可以添加WHERE条件快速排查valueColumns整行全部都是NULL的数据。

## 报错：There is an invalid character in tagvalue

- 报错信息



```
java.lang.IllegalArgumentException: There is an invalid character in tagvalue. the tag is + <method:getFollower$401>, the char is '$'
 at com.alibaba.hitsdb.client.value.request.Point.checkTag(Point.java:355)
 at com.alibaba.hitsdb.client.value.request.Point.checkPoint(Point.java:313)
 at com.alibaba.hitsdb.client.value.request.Point$MetricBuilder.build(Point.java:278)
 at com.alibaba.blink.connectors.hitsdb.DefaultRowPointConverter.convert(DefaultRowPointConverter.java:63)
 at com.alibaba.blink.connectors.hitsdb.HITSDBOutputFormat.writeAddRecord(HITSDBOutputFormat.java:130)
 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:50)
 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:34)
 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:71)
 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:34)
 at org.apache.flink.streaming.api.functions.sink.SinkFunction.invoke(SinkFunction.java:52)
 at org.apache.flink.streaming.api.operators.StreamSink.processElement(StreamSink.java:56)
 at org.apache.flink.streaming.runtime.tasks.OperatorChain$OutputAdapter.lambda$new$0(OperatorChain.java:767)
 at org.apache.flink.streaming.runtime.tasks.OperatorChain$OutputAdapter.processElement(OperatorChain.java:841)
 at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingOutput.pushToOperator(OperatorChain.java:689)
 at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingOutput.collect(OperatorChain.java:673)
 at org.apache.flink.streaming.runtime.tasks.OperatorChain$ChainingOutput.collect(OperatorChain.java:639)
 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:1106)
 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:1084)
 at DataStreamSinkConversion$373.processElement(Unknown Source)
 at org.apache.flink.streaming.runtime.io.StreamInputProcessor.processInput(StreamInputProcessor.java:255)
 at org.apache.flink.streaming.runtime.tasks.OneInputStreamTask.run(OneInputStreamTask.java:89)
 at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:298)
 at org.apache.flink.runtime.taskmanager.Task.run(Task.java:747)
 at java.lang.Thread.run(Thread.java:834)
```

- 报错原因

TSDB输入数据中存在非法字符 \$ 。

- 解决方案


TSDB输入数据只可包含大小写英文字母、中文、数字，以及特殊字符 -、\_、.、/、()、:、,、[]、=、'、#。详情请参见 [写入数据](#)。

 说明 建议您在写入数据前进行字符替换。

## 报错：Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLTransactionRollbackException: Deadlock found when trying to get lock; try restarting transaction

### • 报错信息

写MySQL（TDDL/RDS）时，出现死锁（DeadLock）。

 **说明** 在实时计算Flink中，下游数据库使用MySQL等关系数据库（对应的Connector为TDDL/RDS），当实时计算Flink版频繁写某个表或者资源时，存在死锁风险。

### • 死锁形成的示例

假设完成一次INSERT需要依次抢占(A,B)2个锁。A是一个范围锁，有2个事务(T1,T2)，表的Schema为(id(自增主键),nid(唯一键))。T1包含2条insert(null,2),(null,1)，T2包含1条insert(null,2)。

- i. t时刻，T1第一条INSERT插入，此时T1持有(A,B)2个锁。
- ii. t+1时刻T2开始插入，需要等待锁A来锁住(-inf,2]，此时A被T1拥有，且锁住了(-inf,2]，区间存在包含关系，所以T2依赖T1释放A。
- iii. t+2时刻T1第二条INSERT执行，需要A锁住(-inf,1]，该区间属于(-inf,2]，所以需要排队等T2释放锁，所以T1依赖T2释放A。

当T1和T2相互依赖且相互等待时死锁形成。

### • RDS/TDDL、OTS数据库引擎锁的区别

- **RDS/TDDL**: InnoDB的行锁是针对索引加的锁，不是针对单条记录加的锁。所以虽然是访问不同行的记录，但是如果使用相同的索引键，会出现锁冲突，造成了整个区域的数据都无法更新。
- **OTS**: 单行锁，不影响其他数据更新。

### • 死锁的解决方案

高QPS/TPS或高并发写入情况场景，建议使用OTS作为结果表，可以解决死锁的问题。通常，不建议使用TDDL或者RDS作为Flink Job的结果表。

如果必须要使用MySQL等关系数据库作为Sink节点，有以下建议：

- 确保没有其他读写业务方的干扰。
- 如果Job的数据量不大可以尝试单并发写入。但是在高QPS/TPS、高并发情况下，写入性能会降低。
- 尽可能不使用UniqueKey（唯一主键），带UniqueKey表的写入可能会导致死锁。如果业务要求表必须包含UniqueKey，请按照字段区分能力从大到小排列来定义UniqueKey，可以大幅降低死锁出现概率。例如，您可以把MD5函数放在day\_time(20171010)前面，就可以使得字段区分能力从大到小排列来定义UniqueKey，从而解决死锁问题。
- 根据业务特点做分库分表，尽可能避免单表写入，实施细节请联系对应的数据库管理员。

## 报错：Value '0000-00-00 00:00:00' can not be represented as java.sql.Timestamp

### • 报错信息

```
Caused by: java.sql.SQLException: Value '0000-00-00 00:00:00' can not be represented as java.sql.Timestamp
 at com.mysql.jdbc.SQLException.createSQLException(SQLException.java:964)
 at com.mysql.jdbc.SQLException.createSQLException(SQLException.java:897)
 at com.mysql.jdbc.SQLException.createSQLException(SQLException.java:886)
 at com.mysql.jdbc.SQLException.createSQLException(SQLException.java:860)
 at com.mysql.jdbc.ResultSetRow.getTimestampFast(ResultSetRow.java:937)
 at com.mysql.jdbc.ByteArrayRow.getTimestampFast(ByteArrayRow.java:130)
 at com.mysql.jdbc.ResultSetImpl.getTimestampInternal(ResultSetImpl.java:5918)
 at com.mysql.jdbc.ResultSetImpl.getTimestamp(ResultSetImpl.java:5588)
 at com.alibaba.druid.pool.DruidPooledResultSet.getTimestamp(DruidPooledResultSet.java:220)
 at com.alibaba.blink.connectors.mysql.dim.MySqlRowFetcherBase.deserializeToInnerObject(MySqlRowFetcherBase.java:316)
 at com.alibaba.blink.connectors.mysql.dim.MySqlRowFetcherBase.toResultRow(MySqlRowFetcherBase.java:251)
 at com.alibaba.blink.connectors.mysql.dim.MySqlRowFetcherBase.flatMap(MySqlRowFetcherBase.java:218)
 at com.alibaba.blink.connectors.mysql.dim.MySqlRowFetcherBase.flatMap(MySqlRowFetcherBase.java:58)
 at com.alibaba.blink.streaming.connectors.common.LookupFunctionWrapper.eval(LookupFunctionWrapper.java:56)
 at LookupFunction$146.flatMap(Unknown Source)
 at org.apache.flink.table.runtime.join.TemporalTableJoinProcessRunner.processElement(TemporalTableJoinProcessRunner.scala:85)
 at org.apache.flink.table.runtime.join.TemporalTableJoinProcessRunner.processElement(TemporalTableJoinProcessRunner.scala:33)
 at org.apache.flink.streaming.api.operators.ProcessOperator.processElement(ProcessOperator.java:66)
)
```

- 问题原因

RDS维表中TIMESTAMP类型的字段中存在非法格式的数据0000-00-00 00:00:00，维表JOIN非法格式的数据就会报错。

- 解决方案

在RDS维表WITH参数 URL 后添加 `?zeroDateTimeBehavior=convertToNull`，输出为null。

## 报错：Failover:offset (0) + length (8) exceed the capacity of the array: 1

- 报错信息

```
Caused by: java.lang.IllegalArgumentException: offset (0) + length (8) exceed the capacity of the array: 1
 at org.apache.hadoop.hbase.util.Bytes.explainWrongLengthOrOffset(Bytes.java:548)
 at org.apache.hadoop.hbase.util.Bytes.toLong(Bytes.java:522)
 at org.apache.hadoop.hbase.util.Bytes.toLong(Bytes.java:495)
```

- 问题原因

云数据库HBase版的数据类型定义错误。

- 解决方案

修改云数据库HBase版的数据类型和物理表一致。

## 作业启动一段时间后报错停止，Failover里出现报错信息，应该如何处理？

## ● STS Token过期

```
1 FailOver Time: 2018-04-23 23:04:01
2
3 java.lang.RuntimeException: Could not forward element to next operator
4 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:611)
5 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:593)
6 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:941)
7 at org.apache.flink.streaming.api.operators.AbstractStreamOperator$CountingOutput.collect(AbstractStreamOperator.java:919)
8 at org.apache.flink.streaming.api.operators.StreamMap.processElement(StreamMap.java:38)
9 at org.apache.flink.streaming.runtime.io.StreamInputProcessor.processInput(StreamInputProcessor.java:282)
10 at org.apache.flink.streaming.runtime.tasks.OneInputStreamTask.run(OneInputStreamTask.java:66)
11 at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:307)
12 at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
13 at java.lang.Thread.run(Thread.java:834)
14 Caused by: com.alibaba.blink.streaming.connectors.common.exception.BlinkRuntimeException: 821304
15
16 用户错误-连接错误：连续4次往Datahub写入数据失败，请联系Datahub系统管理员进行排查。
17 错误信息: The sts token has been expired. (Status Code: 403; Error Code: NoPermission; Request ID: 2018042323035229b99a0a8b030437)。
18 at com.alibaba.blink.streaming.connectors.datahub.DatahubExceptionUtil.getBlinkException(DatahubExceptionUtil.java:28)
19 at com.alibaba.blink.streaming.connectors.datahub.sink.DatahubOutputFormat.writeDataHubPack(DatahubOutputFormat.java:386)
20 at com.alibaba.blink.streaming.connectors.datahub.sink.DatahubOutputFormat.batchWrite(DatahubOutputFormat.java:406)
21 at com.alibaba.blink.streaming.connectors.datahub.sink.DatahubOutputFormat.sync(DatahubOutputFormat.java:238)
22 at com.alibaba.blink.streaming.connectors.datahub.sink.DatahubOutputFormat.writeAddRecord(DatahubOutputFormat.java:460)
23 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:50)
24 at com.alibaba.blink.streaming.connectors.common.output.TupleRichOutputFormat.writeRecord(TupleRichOutputFormat.java:34)
25 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:70)
26 at com.alibaba.blink.streaming.connectors.common.output.TupleOutputFormatAdapterSink.invoke(TupleOutputFormatAdapterSink.java:33)
27 at org.apache.flink.streaming.api.operators.StreamSink.processElement(StreamSink.java:38)
28 at org.apache.flink.streaming.runtime.tasks.OperatorChain$CopyingChainingOutput.collect(OperatorChain.java:608)
29 ... 9 more
30
```

STS Token过期导致的报错，不影响线上作业。详情请参见 [报错：The sts token has been expired](#)。

## ● 脏数据处理失败

### ○ 排错指引

在Failover中查看脏数据具体内容。

### ○ 解决方案

修改SQL，过滤脏数据后，重新 [上线](#) 作业。

## ● 写入MYSQL类型数据库死锁

### ○ 排错指引

- 手动调优后，检查Sink节点拆分后的部分节点与同时写入的其他节点之间，是否出现了Rebalance。
- 检查Sink节点是否多并发写入数据库。
- 检查出现死锁的表中PK与数据库表定义的UK是否一致，包括顺序。
- 检查定义的PK是否按照区分度高低进行排列。

### ○ 解决方案

详情请参见 [写MySQL（TDDL/RDS）时，出现死锁（DeadLock）](#)。

# 10.最佳实践

## 10.1. 电商行业最佳实践

### 10.1.1. 电商场景实战之多类目成交总额管理

本文通过案例为您介绍如何使用实时计算完成数据的实时处理、多类目的管理。

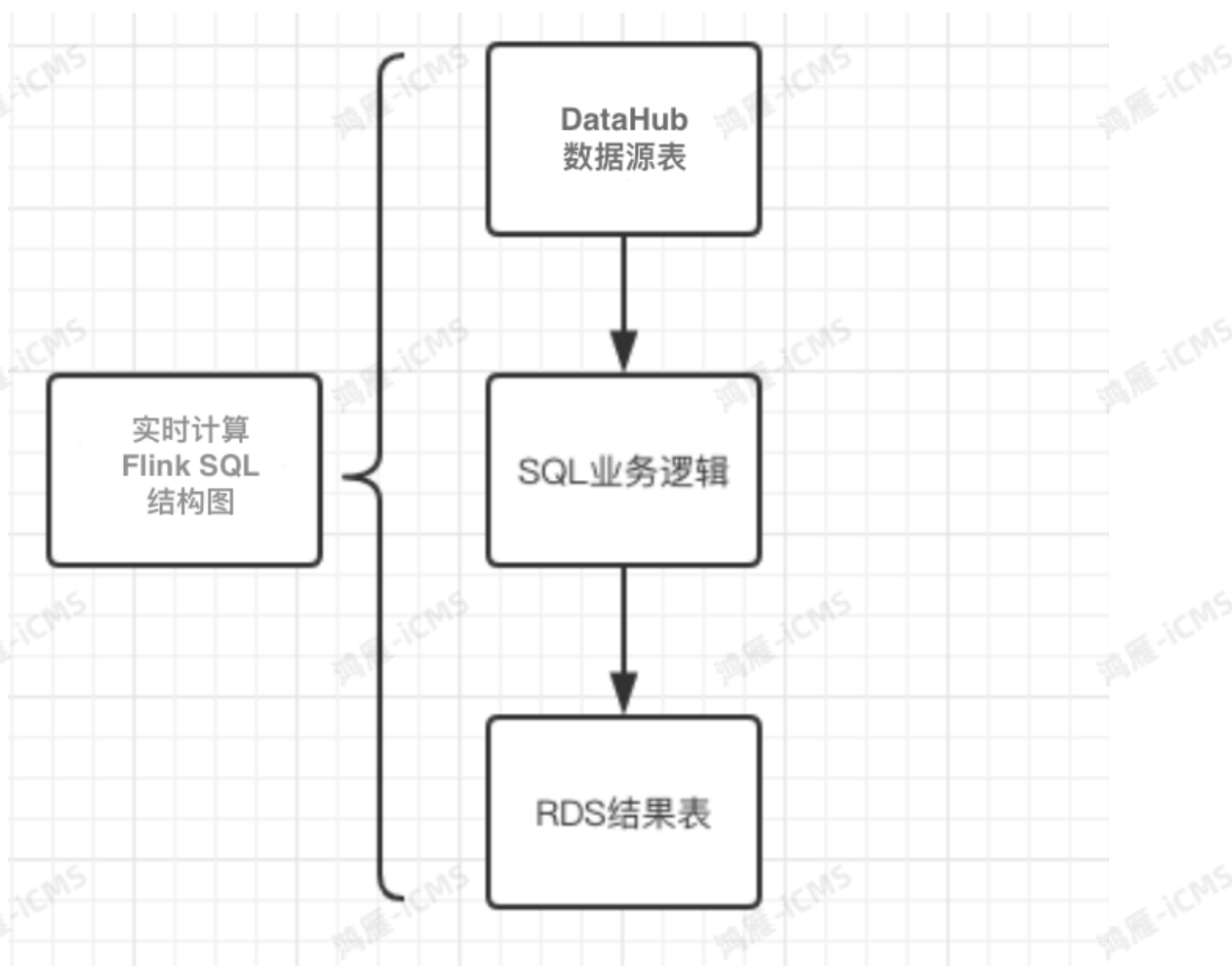
#### 背景信息

以下是天猫工程师利用实时计算实现实时多类目成交总额管理的案例。

#### 多类目成交总额管理案例

② 说明 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

#### SQL结构图



操作步骤如下：

1. 创建数据总线DataHub源表

系统订单是实时产生的，源表语句如下。

```
CREATE TABLE source_order (
 id VARCHAR,-- 商品ID。
 buyer_id VARCHAR, --买家ID。
 site VARCHAR,--商品类别。
 pay_time VARCHAR,--订单支付时间。
 buy_amount DOUBLE, --订单金额。
 wap VARCHAR--购买方式。
) WITH (
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
 project='yourProjectName',--DataHub中Project的名称。
 topic='yourTopicName',--DataHub Project中Topic的名称。
 roleArn='yourRoleArn',--指定角色的roleArn。
 batchReadSize='500'
);
```

## 2. 创建RDS结果表

RDS结果表语句如下。

```
CREATE TABLE ads_site_block_trd_pay_ri (
 id VARCHAR,
 site VARCHAR,
 data_time VARCHAR,
 all_alipay BIGINT,
 all_ord_cnt BIGINT,
 primary key(id,site,data_time)
) WITH (
 type='rds',
 url = 'yourRDSDatabaseURL',--RDS数据库URL。
 tableName = 'yourDatabaseTableName',--RDS数据库中的表名。
 userName = 'yourDatabaseUserName',--登录数据库的用户名。
 password = 'yourDatabasePassword'--登录数据库的密码。
);
```

## 3. 创建一个View视图

根据商品ID、商品类别、买家ID和订单支付时间做分组操作。

```
CREATE VIEW tmp_ads_site_block_trd_pay_ri AS
SELECT
 id id,
 mod(HASH_CODE(`a`.buyer_id),4096) hash_id,
 site site,
 date_format(`a`.pay_time, 'yyyy-MM-dd HH:mm:ss', 'yyyyMMddHH') data_time,
 SUM(cast(buy_amount as bigint)) all_alipay,
 count(distinct `a`.buyer_id) all_ord_cnt
FROM
 source_order `a`
GROUP BY id, site, mod(HASH_CODE(`a`.buyer_id),4096), date_format(`a`.pay_time, 'yyyy-MM-dd HH:mm:ss', 'yyyyMMddHH');
```

## 4. 数据聚合后插入RDS结果表

把分组好的数据聚合后插入RDS结果表中，语句如下。

```
INSERT INTO ads_site_block_trd_pay_ri
SELECT
id,
site,
`a`.data_time,
CAST(sum(all_alipay) AS BIGINT) as all_alipay,
CAST(sum(all_ord_cnt) AS BIGINT) as all_ord_cnt
FROM
tmp_ads_site_block_trd_pay_ri `a`
GROUP BY id , site , `a`.data_time ;
```

## 常见问题

Q: 如何解决数据倾斜?

A: 假如您的ID数据非常大，根据您的ID进行分组计算可能会造成机器热点从而导致数据倾斜，计算性能会很差。

```
mod(HASH_CODE(`a`.buyer_id),4096) hash_id
```

使用HASH\_CODE这个离散函数来分离数据热点，然后使用MOD函数对哈希值进行分组操作。这样做的优点是规范每个节点数据的数量，避免大量数据的堆积导致数据倾斜（4096指的是分组的数量，可以根据数据的大小进行分组操作）。HASH\_CODE函数详情请参见[HASH\\_CODE](#)。

Q: 您得到的UV量是否准确?

A: 直接使用 `count(distinct,buyer_id)` 会出现计算不准的问题。请使用 `GROUP BY mod(HASH_CODE(a.buyer_id),4096)`，把相同 `buyer_id` 过滤去重后，再进行 `SUM` 操作，就可以避免这样的错误产生。

```
CREATE VIEW tmp_ads_site_block_trd_pay_ri AS
SELECT
id id,
mod(HASH_CODE(`a`.buyer_id),4096) hash_id,
site site,
date_format(`a`.pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMddHH') data_time,
SUM(cast(buy_amount as bigint)) all_qty_cnt,
count(distinct `a`.buyer_id) all_ord_cnt
FROM
source_order `a`
GROUP BY id , site , mod(HASH_CODE(`a`.buyer_id),4096) , date_format(`a`.pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMddHH');
```

## DEMO示例以及源代码

根据上文介绍的成交总额管理解决方案，为您创建了一个包含完整链路的 **DEMO代码**，您可以参考示例代码，注册上下游数据，制定您的多类目成交总额管理解决方案。使用示例时，您需要使用DataHub作为源表提供流数据，RDS作为结果表来存储结果数据。

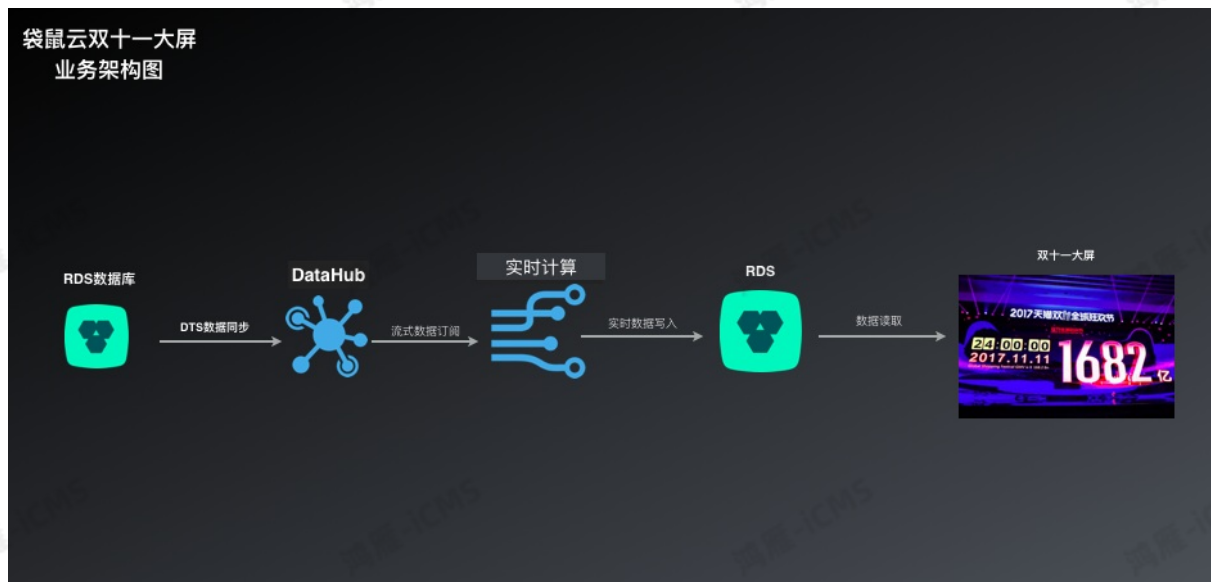
### 10.1.2. 电商场景实战之订单与销量统计

本文通过案例为您介绍如何使用实时计算完成订单与销量的统计。

## 背景信息

以下案例是实时计算的合作伙伴袋鼠云通过阿里云实时计算来完成电商订单管理的案例。

## 业务架构图



业务流程：

1. 使用数据传输服务DTS把您的数据同步到大数据总线（DataHub）。具体步骤请参见 [MySQL到DataHub数据实时同步](#)。
2. 阿里云实时计算订阅大数据总线（DataHub）的数据进行实时计算。
3. 将实时数据插入到RDS的云数据库。
4. 通过阿里云的DataV或者是其他的大屏完成数据展示。

## 准备工作

将RDS MySQL产生的增量数据实时同步到DataHub中的Topic。由RDS经过DTS数据同步到大数据总线（DataHub）Schema表的信息。

具体步骤请参见 [MySQL到DataHub数据实时同步](#)。

orders\_real源表

| 字段名                | 数据类型    | 详情     |
|--------------------|---------|--------|
| dts_ordercodeofsys | VARCHAR | 订单编号   |
| dts_paytime        | BIGINT  | 订单付款时间 |
| dts_deliveredtime  | VARCHAR | 订单发货时间 |
| dts_storecode      | VARCHAR | 店铺编号   |
| dts_warehousecode  | VARCHAR | 仓库code |



| 字段名                  | 数据类型    | 详情            |
|----------------------|---------|---------------|
| dts_cancelled        | BIGINT  | 是否取消          |
| dts_delivered        | BIGINT  | 是否发货          |
| dts_receivercity     | VARCHAR | 收货人城市         |
| dts_receiverprovince | VARCHAR | 收货人省份         |
| dts_record_id        | VARCHAR | 记录ID          |
| dts_operation_flag   | VARCHAR | 操作Flag        |
| dts_instance_id      | VARCHAR | 数据库instanceId |
| dts_db_name          | VARCHAR | 数据库名          |
| dts_table_name       | VARCHAR | 数据表           |
| dts_utc_timestamp    | VARCHAR | 更新时间          |
| dts_before_flag      | VARCHAR | 变更前标识         |
| dts_after_flag       | VARCHAR | 变更后标识         |

orderdetail\_real源表

| 字段名                          | 数据类型    | 详情            |
|------------------------------|---------|---------------|
| dts_ordercodeofsys           | VARCHAR | 订单编号          |
| dts_sku_name                 | VARCHAR | 商品名字          |
| dts_sku_code                 | VARCHAR | 商品编号          |
| dts_quantity                 | BIGINT  | 数量            |
| dts_dividedamount            | DOUBLE  | 发货金额          |
| dts_salechanneldividedamount | DOUBLE  | 渠道销售金额        |
| dts_initialcost              | DOUBLE  | 成本            |
| dts_record_id                | VARCHAR | 记录ID          |
| dts_operation_flag           | VARCHAR | 操作Flag        |
| dts_instance_id              | VARCHAR | 数据库instanceId |
| dts_db_name                  | VARCHAR | 数据库名字         |
| dts_table_name               | VARCHAR | 表名            |

| 字段名               | 数据类型    | 详情    |
|-------------------|---------|-------|
| dts_utc_timestamp | VARCHAR | 更新时间  |
| dts_before_flag   | VARCHAR | 变更前标识 |
| dts_after_flag    | VARCHAR | 变更后标识 |

## 编写业务逻辑

```
--数据的订单源表。
create table orders_real(
 dts_ordercodeofsys varchar,
 dts_paytime bigint,
 dts_deliveredtime varchar,
 dts_storecode varchar,
 dts_warehousecode varchar,
 dts_cancelled bigint,
 dts_delivered bigint,
 dts_receivercity varchar,
 dts_receiverprovince varchar,
 dts_record_id varchar,
 dts_operation_flag varchar,
 dts_instance_id varchar,
 dts_db_name varchar,
 dts_table_name varchar,
 dts_utc_timestamp varchar,
 dts_before_flag varchar,
 dts_after_flag varchar
) with (
 type='datahub',
 endPoint='http://dh-cn-****.com',
 project='your',
 topic='表名',
 accessId='您的ID',
 accessKey='您的KEY'
);
create table orderdetail_real(
 dts_ordercodeofsys varchar,
 dts_skuname varchar,
 dts_skucode varchar,
 dts_quantity bigint,
 dts_dividedamount double,
 dts_salechanneldividedamount double,
 dts_initialcost double,
 dts_record_id varchar,
 dts_operation_flag varchar,
 dts_instance_id varchar,
 dts_db_name varchar,
 dts_table_name varchar,
 dts_utc_timestamp varchar,
 dts_before_flag varchar,
 dts_after_flag varchar
);
```

```
) with (
 type='datahub',
 endPoint='http://dh-cn-****.com',
 project='yourPorjectName',
 topic='yourTableName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret'
);
create table ads_all_count_amount(
 bill_date varchar,--下单时间。
 bill_count bigint,--总的订单总数。
 qty bigint,--总的销售量。
 primary key (bill_date)
) with (
 type='rds',
 url='jdbc:mysql://rm-XXXX.mysql.rds.aXXXXcs.com:3306/XXXX',
 tableName='yourDatabaseTableName',
 userName='yourDatabaseAccount',
 password='yourDatabasePassword'
);
--订单源表，最新交易时间的商品编号。
CREATE VIEW new_paytime AS
SELECT
 dts_ordercodeofsys,
 MAX(dts_paytime) AS dts_paytime
FROM orders_real
GROUP BY dts_ordercodeofsys ;
--订单详情表，有效订单的订单编码、商品名称、商品编号、数量的信息。
CREATE VIEW new_orderdetail AS
SELECT
 dts_ordercodeofsys,
 dts_skuname,
 dts_skucode,
 CASE WHEN dts_operation_flag='U'
 AND dts_before_flag='Y'
 AND dts_after_flag='N' THEN -1*dts_quantity
 WHEN dts_operation_flag='U'
 AND dts_before_flag='N'
 AND dts_after_flag='Y' THEN dts_quantity
 WHEN dts_operation_flag='D' THEN -1*dts_quantity
 ELSE dts_quantity
 END AS dts_quantity
FROM
 orderdetail_real;
--订单总单数，总销售量。
INSERT INTO ads_all_count_amount
SELECT
 FROM_UNIXTIME(cast(a.dts_paytime/1000000 AS bigint),'yyyyMMdd') AS bill_date,
 COUNT(DISTINCT a.dts_ordercodeofsys) AS bill_count,
 SUM(b.dts_quantity) AS qty
from
 new_paytime a
 join
 new_orderdetail b
ON a.dts_ordercodeofsys=b.dts_ordercodeofsys
```

```
ON a.dts_ordercodeofsys--b.dts_ordercodeofsys
GROUP BY
FROM_UNIXTIME(CAST(a.dts_paytime/1000000 AS bigint),'yyyyMMdd');
```

## 难点解析

为了方便您理解结构化代码和代码维护，推荐使用View（[数据视图概念](#)）将业务逻辑拆分为三个模块。

- 模块一：根据订单编号进行分组

同一个编号订单会有多次业务操作（例如下单、付款、发货），并在Binlog日志中形成多条同一订单编号的订单流水记录。使用 `MAX(dts_paytime)` 获取同一编号的最后一次操作数据库最终付款交易时间。

```
CREATE VIEW new_paytime AS
SELECT
 dts_ordercodeofsys,
 MAX(dts_paytime) AS dts_paytime
FROM orders_real
GROUP BY dts_ordercodeofsys;
```

- 模块二：生成有效订单的信息

--订单详情表，有效订单的订单编码、商品名称、商品编号、数量的信息。

```
CREATE VIEW new_orderdetail AS
SELECT
 dts_ordercodeofsys,
 dts_skuname,
 dts_skucode,
 CASE WHEN dts_operation_flag='U'
 AND dts_before_flag='Y'
 AND dts_after_flag='N' THEN -1*dts_quantity
 WHEN dts_operation_flag='U'
 AND dts_before_flag='N'
 AND dts_after_flag='Y' THEN dts_quantity
 WHEN dts_operation_flag='D' THEN -1*dts_quantity
 ELSE dts_quantity
 END AS dts_quantity
FROM orderdetail_real;
```

数据库日志会获取所有的数据记录的变更，而每个订单是有状态的，如下表所示。

| 字段名                | 数据类型    | 详情                                                                                                                         |
|--------------------|---------|----------------------------------------------------------------------------------------------------------------------------|
| dts_record_id      | VARCHAR | 记录ID。增量日志的唯一标识，唯一递增。如果变更类型为Update，那么增量更新会被拆分成2条。一条Insert，一条Delete。这两条记录具有相同的 record_id。                                    |
| dts_operation_flag | VARCHAR | 标示这条增量日志的操作类型，取值为： <ul style="list-style-type: none"> <li>○ I: Insert</li> <li>○ D: Delete</li> <li>○ U: Update</li> </ul> |
| dts_instance_id    | VARCHAR | 数据库 instanceid。这条增量日志所对应的数据库的Server ID。                                                                                    |

| 字段名               | 数据类型    | 详情                                                                                                                 |
|-------------------|---------|--------------------------------------------------------------------------------------------------------------------|
| dts_db_name       | VARCHAR | 这条增量更新日志更新的表所在的数据库库名。                                                                                              |
| dts_table_name    | VARCHAR | 这条增量更新日志更新的表名。                                                                                                     |
| dts_utc_timestamp | VARCHAR | 这条增量日志的操作时间戳，为这个更新操作记录Binlog的时间戳。时间戳为UTC时间。                                                                        |
| dts_before_flag   | VARCHAR | 表示这条增量日志后面带的各个Column值是否为更新前的值。取值包括：Y 和 N。当后面的Column为更新前的值时，dts_before_flag=Y，当后面的Column值为更新后的值时，dts_before_flag=N。 |
| dts_after_flag    | VARCHAR | 表示这条增量日志后面带的各个Column值是否为更新后的值。取值包括：Y 和 N。当后面的Column为更新前的值时，dts_after_flag=N，当后面的Column值为更新后的值时，dts_after_flag=Y。   |

对于不同的操作类型，增量日志中的 dts\_before\_flag 和 dts\_after\_flag 定义如下：

i. 操作类型为Insert

所有Column值为新插入的记录值，即为更新后的值，所以 dts\_before\_flag=N，dts\_after\_flag=Y。

| dts_record_id | dts_instance_id | dts_db_name | dts_table_name | dts_operation_flag | dts_utc_timestamp | dts_before_flag | dts_after_flag | dts_col1 | ....  | dts_colN   |
|---------------|-----------------|-------------|----------------|--------------------|-------------------|-----------------|----------------|----------|-------|------------|
| 1             | 234             | db1         | sbtest1        | I                  | 1476258462        | N               | Y              | 1        | ..... | JustInsert |

ii. 操作类型为Update

Update操作被拆为2条增量日志。这两条增量日志的 dts\_record\_id，dts\_operation\_flag 及 dts\_utc\_timestamp 相同。第一条日志记录更新前的值，所以 dts\_before\_flag=Y，dts\_after\_flag=N。第二条日志记录了更新后的值，所以 dts\_before\_flag=N，dts\_after\_flag=Y。

| dts_record_id | dts_instance_id | dts_db_name | dts_table_name | dts_operation_flag | dts_utc_timestamp | dts_before_flag | dts_after_flag | dts_col1 | ....  | dts_colN   |
|---------------|-----------------|-------------|----------------|--------------------|-------------------|-----------------|----------------|----------|-------|------------|
| 2             | 234             | db1         | sbtest1        | I                  | 1476258463        | Y               | N              | 1        | ..... | JustInsert |
| 2             | 234             | db1         | sbtest1        | I                  | 1476258463        | N               | Y              | 1        | ..... | JustUpdate |

iii. 操作类型为Delete

所有Column值为被删除的记录值，即为更新前的值。所以 `dts_before_flag=Y` , `dts_after_flag=N` 。

| dts_record_id | dts_instance_id | dts_db_name | dts_table_name | dts_operation_flag | dts_utc_timestamp | dts_before_flag | dts_after_flag | dts_col1 | ....  | dts_colN   |
|---------------|-----------------|-------------|----------------|--------------------|-------------------|-----------------|----------------|----------|-------|------------|
| 3             | 234             | db1         | sbtest1        | D                  | 1476258464        | Y               | N              | 1        | ..... | JustUpdate |

- 模块三：统计总订单数和销售额

```
SELECT
 from_unixtime(CAST(a.dts_paytime/1000000 AS bigint),'yyyyMMdd') AS bill_date,
 COUNT(DISTINCT a.dts_ordercodeofsys) AS bill_count,
 SUM(b.dts_quantity) AS qty
from
 new_paytime as a
join
 new_orderdetail as b
ON
 a.dts_ordercodeofsys=b.dts_ordercodeofsys
GROUP BY
 from_unixtime(CAST(a.dts_paytime/1000000 AS bigint),'yyyyMMdd');
```

## DEMO示例以及源代码

根据上文介绍的订单与销量统计解决方案，为您创建了一个包含完整链路的DEMO示例，如下所示：

- DataHub作为源表。
- RDS作为结果表。

DEMO代码完整，您可参考示例代码，注册上下游数据，制定自己的订单与销量统计解决方案。单击 [DEMO代码](#) 进行下载。

## 常见问题

- Q：模块二中，如何判断有效交易订单？

A：首先是要满足 `dts_operation_flag=U` 或者 `dts_operation_flag=I`，然后 `dts_before_flag` 代表的是变更前订单状态，`dts_after_flag` 是变更后订单状态。所以有效交易订单如下。

```
dts_operation_flag='U'
AND dts_before_flag='N'
AND dts_after_flag='Y' THEN dts_quantity
```

- Q：模块二中，为什么 `THEN -1*dts_quantity` ？

A：订单的取消或者是交易没有成功，在总的销量里也会记录。为了保证总销量的正确性，所以把没有成交的订单数量设为负数，在计算总的销量会减去这个数量。

- Q：模块三中，为什么订单源表和订单详情需要进行JOIN操作？

A：`new_paytime` 查出的是最新交易的时间的所有的订单编号。`new_orderdetail` 查询的是所有有效订单的订单编码、商品名称、商品编号、数量的信息。两张表JOIN是为了方便用户来统计订单总数和总的销量。

### 10.1.3. 电商场景实战之实时PV和UV曲线

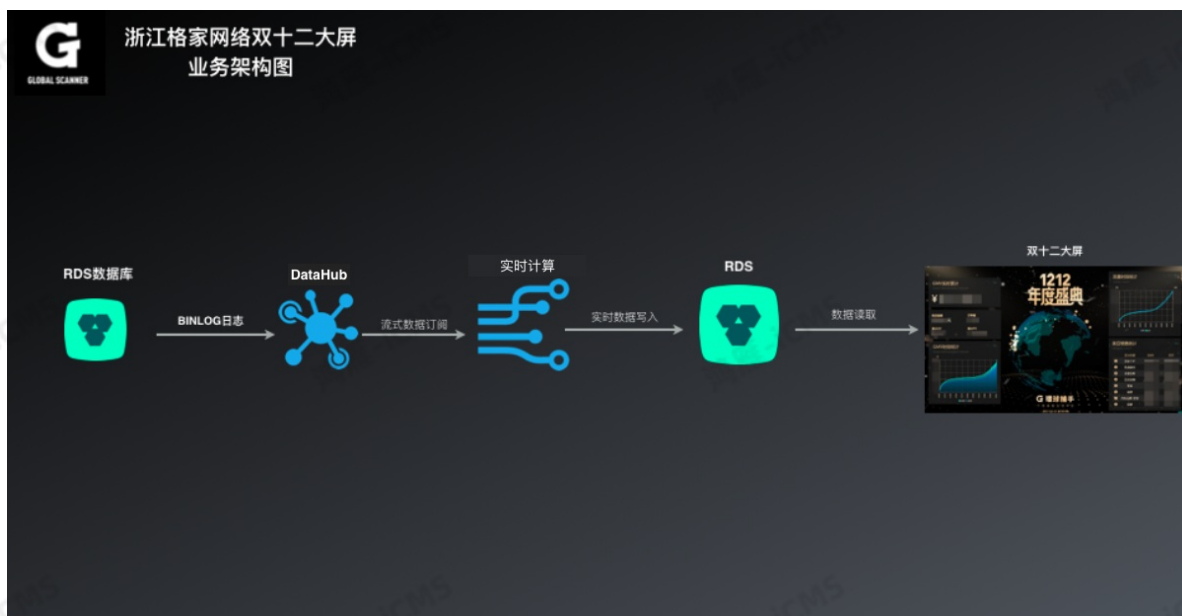
本文以实时计算合作伙伴格格家的案例为例，为您介绍如何使用实时计算制作实时PV和UV曲线图。

#### 背景

随着新零售的概念慢慢崛起，互联网电商行业竞争越来越激烈。实时数据信息对于电商行业尤为重要，例如实时地统计网页PV和UV的总量。

#### 案例

- 业务架构图



- 业务流程

- 通过数据总线（DataHub）提供的SDK，同步Binlog日志数据到DataHub。
- 阿里云实时计算订阅DataHub的数据进行实时计算。
- 插入实时数据到云数据库RDS。
- 通过阿里云DataV数据可视化或其他大屏展示结果数据。

- 准备工作 日志源表

| 字段          | 数据类型    | 说明     |
|-------------|---------|--------|
| account_id  | VARCHAR | 用户ID   |
| client_ip   | VARCHAR | 客户端IP  |
| client_info | VACHAR  | 设备机型信息 |
| platform    | VARHCAR | 系统版本信息 |
| imei        | VARCHAR | 设备唯一标识 |
| version     | BIGINT  | 版本号    |

| 字段                    | 数据类型    | 说明     |
|-----------------------|---------|--------|
| action                | BIGINT  | 页面跳转描述 |
| gpm                   | VARCHAR | 埋点链路   |
| c_time                | VARCHAR | 请求时间   |
| target_type           | VARCHAR | 目标类型   |
| target_id             | VARCHAR | 目标ID   |
| udata                 | VARCHAR | 扩展信息   |
| session_id            | VARHCAR | 会话ID   |
| product_id_chain      | VARHCAR | 商品ID串  |
| cart_product_id_chain | VARCHAR | 加购商品ID |
| tag                   | VARCHAR | 特殊标记   |
| position              | VARCHAR | 位置信息   |
| network               | VARCHAR | 网络使用情况 |
| p_dt                  | VARCHAR | 时间分区   |
| p_platform            | VARCHAR | 系统版本信息 |

数据库RDS结果表

| 字段           | 数据类型      | 说明                                                                                                                       |
|--------------|-----------|--------------------------------------------------------------------------------------------------------------------------|
| summary_date | BIGINT    | 统计日期                                                                                                                     |
| summary_min  | VARCHAR   | 统计分钟                                                                                                                     |
| p_v          | BIGINT    | 单击量                                                                                                                      |
| uv           | BIGINT    | 访客量<br><br> 说明 一天内同个访客多次访问网站计为一个UV。 |
| currenttime  | TIMESTAMP | 当前时间                                                                                                                     |

- 业务逻辑

```
--数据的订单源表。
```

```
CREATE TABLE source_ods_fact_log_track_action (
```



```
account_id VARCHAR,--用户ID。
client_ip VARCHAR,--客户端IP。
client_info VARCHAR,--设备机型信息。
platform VARCHAR,--系统版本信息。
imei VARCHAR,--设备唯一标识。
`version` VARCHAR,--版本号。
`action` VARCHAR,--页面跳转描述。
gpm VARCHAR,--埋点链路。
c_time VARCHAR,--请求时间。
target_type VARCHAR,--目标类型。
target_id VARCHAR,--目标ID。
udata VARCHAR,--扩展信息，JSON格式。
session_id VARCHAR,--会话ID。
product_id_chain VARCHAR,--商品ID串。
cart_product_id_chain VARCHAR,--加购商品ID。
tag VARCHAR,--特殊标记。
`position` VARCHAR,--位置信息。
network VARCHAR,--网络使用情况。
p_dt VARCHAR,--时间分区天。
p_platform VARCHAR --系统版本信息。
) WITH (
 type='datahub',
 endPoint='yourEndpointURL',
 project='yourProjectName',
 topic='yourTopicName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret',
 batchReadSize='1000'
);
CREATE TABLE result_cps_total_summary_pvuv_min (
 summary_date bigint,--统计日期。
 summary_min varchar,--统计分钟。
 pv bigint,--单击量。
 uv bigint,--一天内同个访客多次访问计算为一个UV。
 currenttime timestamp,--当前时间。
 primary key (summary_date,summary_min)
) WITH (
 type='rds',
 url = 'yourRDSDatabaseURL',
 userName = 'yourDatabaseUserName',
 password = 'yourDatabasePassword',
 tableName = 'yourTableName'
);
CREATE VIEW result_cps_total_summary_pvuv_min_01 AS
select
cast(p_dt as bigint) as summary_date --时间分区。
,count(client_ip) as pv --客户端的IP。
,count(distinct client_ip) as uv --客户端去重。
,cast(max(c_time) as TIMESTAMP) as c_time --请求的时间。
from source_ods_fact_log_track_action
group by p_dt;
INSERT into result_cps_total_summary_pvuv_min
select
a.summary_date, --时间分区。
```

```
cast(DATE_FORMAT(c_time,'HH:mm') as varchar) as summary_min, --取出小时分钟级别的时间。
a.pv,
a.uv,
CURRENT_TIMESTAMP as currenttime --当前时间。
from result_cps_total_summary_pvuv_min_01 AS a
;
```

- 难点解析

为了方便理解结构化代码和代码维护，我们推荐使用View（[数据视图概念](#)）把业务逻辑拆分成两个模块。

- 模块一

```
CREATE VIEW result_cps_total_summary_pvuv_min_01 AS
select
cast(p_dt as bigint) as summary_date, --时间分区。
count(client_ip) as pv, --客户端的IP。
count(distinct client_ip) as uv, --客户端去重。
cast(max(c_time) as TIMESTAMP) as c_time --请求的时间。
from source_ods_fact_log_track_action
group by p_dt;
```

- PV是客户访问网站后的单击次数，UV是去重的客户IP数。
- cast(max(c\_time) as TIMESTAMP)为最后请求时间。
- 以p\_dt作为时间分区，单位为天，以max(c\_time)作为访问网站的截止时间，向数据库插入一条PV和UV。

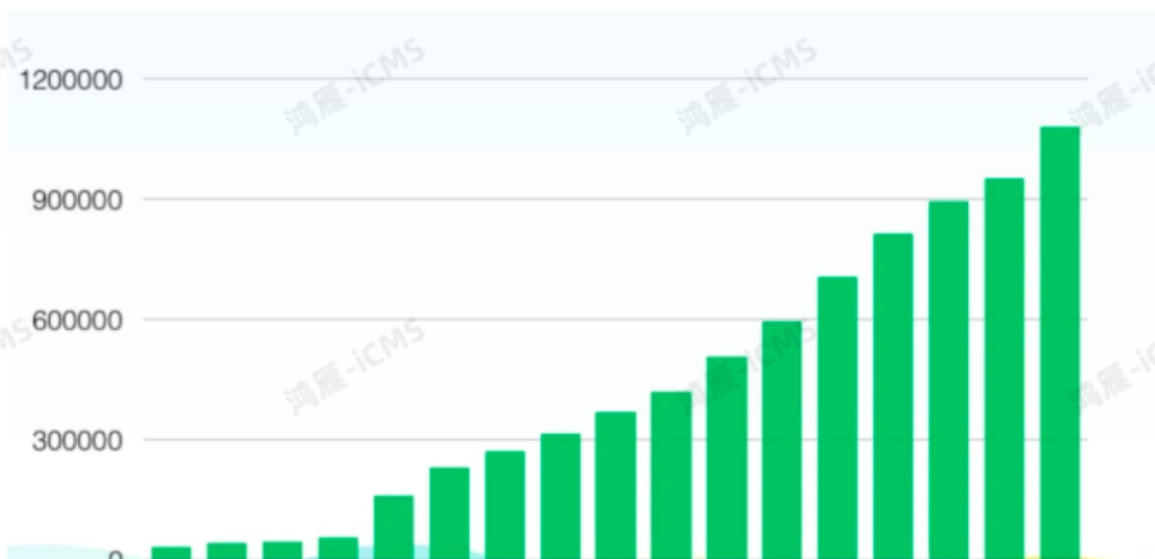
#### 结果

| p_dt       | pv   | uv  | max(c_time)        |
|------------|------|-----|--------------------|
| 2017-12-12 | 1000 | 100 | 2017-12-12 9:00:00 |
| 2017-12-12 | 1500 | 120 | 2017-12-12 9:01:00 |
| 2017-12-12 | 2200 | 200 | 2017-12-12 9:02:00 |
| 2017-12-12 | 3300 | 320 | 2017-12-12 9:03:00 |

模块二

```
INSERT into result_cps_total_summary_pvuv_min
select
a.summary_date, --时间分区, 天单位为。
cast(DATE_FORMAT(c_time,'HH:mm') as varchar) as summary_min, --取出小时分钟级别的时间。
a.pv,
a.uv,
CURRENT_TIMESTAMP as currenttime --当前时间。
from result_cps_total_summary_pvuv_min_01 AS a;
```

把模块一的数据精确到小时分钟级别取出后，根据数据得出PV和UV的增长曲线。



### 示例及源代码

根据上文介绍的PV和UV曲线解决方案，阿里云为您创建了一个包含完整链路的DEMO示例，您可以参考DEMO示例，注册上下游存储，从而得出您的PV和UV曲线图。单击[代码示例](#)下载完整示，使用示例时，上下游存储请注意以下两点：

- DataHub作为源表。
- RDS作为结果表。

## 10.1.4. 电商场景实战之营销红包

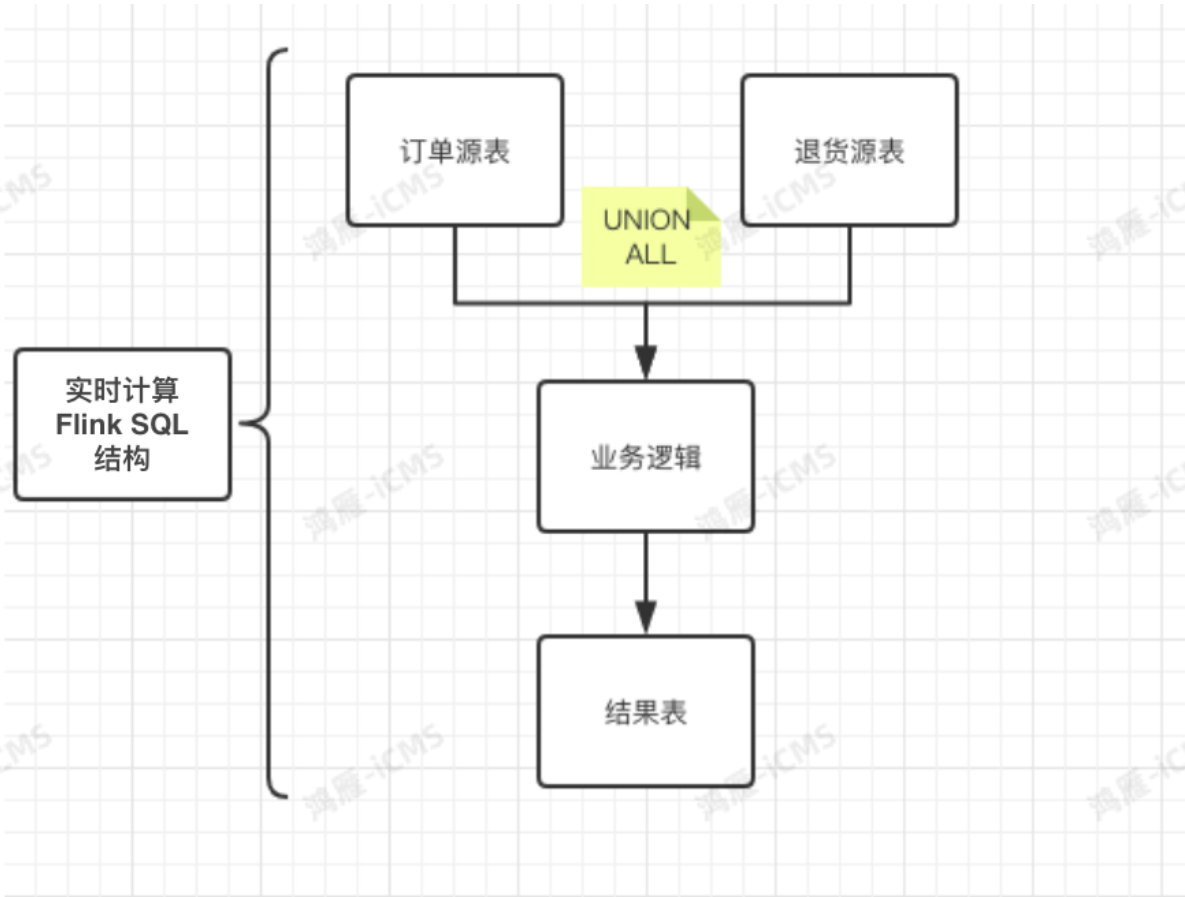
本文为您介绍实时计算如何在营销红包的策略中筛选满足营销红包发放条件的用户。

### 背景

某商家双十一活动期间的“回血红包”营销方案。用户在消费达到一定金额后，商家为了促进用户继续消费会返给用户一定金额的“回血红包”。实时计算能够为您实时的去监控用户的消费金额，筛选满足营销红包发放条件的用户。

### 解决方案

- SQL结构图



- 数据源表

○ 系统订单源表

② 说明 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

```
CREATE TABLE dwd_tb_trd_pay_ri(
 biz_order_id VARCHAR, -- '订单ID'
 auction_id VARCHAR, -- '商品ID'
 auction_title VARCHAR, -- '商品标题'
 buyer_id VARCHAR, -- '买家ID'
 buyer_nick VARCHAR, -- '买家昵称'
 pay_time VARCHAR, -- '支付时间'
 gmt_create VARCHAR, -- '创建时间'
 gmt_modified VARCHAR, -- '修改时间'
 biz_type VARCHAR, -- '交易类型'
 pay_status VARCHAR, -- '支付状态'
 `attributes` VARCHAR, -- '订单标记'
 from_group VARCHAR, -- '订单来源'
 div_idx_actual_total_fee DOUBLE -- '成交金额'
) WITH (
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
 project='yourProjectName', -- '您的project'
 topic='yourTopicName', -- '您的topic'
 roleArn='yourRoleArn', -- '您的roleArn'
 batchReadSize='500'
);
```

## ○ 退货源表

② 说明 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

```
CREATE TABLE dwd_tb_trd_rfd_ri(
 biz_order_id VARCHAR, -- '订单ID'
 auction_id VARCHAR, -- '商品ID'
 auction_title VARCHAR, -- '商品标题'
 buyer_id VARCHAR, -- '买家ID'
 buyer_nick VARCHAR, -- '买家昵称'
 pay_time VARCHAR, -- '支付时间'
 gmt_create VARCHAR, -- '创建时间'
 gmt_modified VARCHAR, -- '修改时间'
 biz_type VARCHAR, -- '交易类型'
 pay_status VARCHAR, -- '支付状态'
 `attributes` VARCHAR, -- '订单标记'
 from_group VARCHAR, -- '订单来源'
 refund_fee DOUBLE -- '退款金额'
) WITH (
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
 project='yourProjectName', -- '您的project'
 topic='yourTopicName', -- '您的topic'
 roleArn='yourRoleArn', -- '您的roleArn'
 batchReadSize='500'
);
```

## ● 数据结果表

## 关系型数据库RDS结果表

```
CREATE TABLE tddl_output(
 gmt_create VARCHAR, -- '创建时间'
 gmt_modified VARCHAR, -- '修改时间'
 buyer_id BIGINT, -- '买家ID'
 cumulate_amount BIGINT, -- '金额'
 effect_time BIGINT, -- '支付时间'
 primary key(buyer_id, effect_time)
) WITH (
 type='rds',
 url = 'yourDatabaseURL', -- '您的数据库url'
 tableName = 'yourTableName', -- '您的表名'
 userName = 'yourUserName', -- '您的用户名'
 password = 'yourDatabasePassword' -- '您的密码'
);
```

## ● 业务逻辑

把订单表和退货表做UNION ALL操作，获取到用户购买的所有商品的信息，统计用户的真实的消费金额和明细。

```
CREATE VIEW dwd_tb_trd_and_rfd_pay_ri
AS
SELECT
*
FROM(
(SELECT
`a`.biz_order_id biz_order_id,
`a`.auction_id auction_id,
`a`.auction_title auction_title,
`a`.buyer_id buyer_id,
`a`.buyer_nick buyer_nick,
`a`.pay_time pay_time,
`a`.gmt_create gmt_create,
`a`.gmt_modified gmt_modified,
`a`.biz_type biz_type,
`a`.pay_status pay_status,
`a`.`attributes` `attributes`,
`a`.from_group,
`a`.div_idx_actual_total_fee div_idx_actual_total_fee
FROM
dwd_tb_trd_pay_ri `a`
)
UNION ALL
(
SELECT
`b`.biz_order_id biz_order_id,
`b`.auction_id auction_id,
`b`.auction_title auction_title,
`b`.buyer_id buyer_id,
`b`.buyer_nick buyer_nick,
`b`.pay_time pay_time,
`b`.gmt_create gmt_create,
`b`.gmt_modified gmt_modified,
`b`.biz_type biz_type,
`b`.pay_status pay_status,
`b`.`attributes` `attributes`,
`b`.from_group,
`b`.refund_fee div_idx_actual_total_fee --退款取负值
FROM
dwd_tb_trd_rfd_ri `b`
)
);
```

- 去重操作

在订单表和退货表中可能会存在大量重复的数据。例如商品ID、商品名称等。用MIN函数是为了只取第一次来的参数值，从而过滤掉其他的信息，然后再通过订单号和支付状态做分组取出需要的商品ID、商品名称等。

```
CREATE VIEW filter_hxhb_dwd_tb_trd_and_rfd_pay_ri_distinct AS
SELECT
 biz_order_id biz_order_id,
 pay_status pay_status,
 MIN(auction_id) auction_id,
 MIN(auction_title) auction_title,
 MIN(buyer_id) buyer_id,
 MIN(buyer_nick) buyer_nick,
 MIN(pay_time) pay_time,
 MIN(gmt_create) gmt_create,
 MIN(gmt_modified) gmt_modified,
 MIN(biz_type) biz_type,
 MIN(attributes) attributes,
 MIN(div_idx_actual_total_fee) div_idx_actual_total_fee
FROM
 dwd_tb_trd_and_rfd_pay_ri
GROUP BY biz_order_id ,pay_status;
```

- 数据汇总所有数据做汇总处理

```
CREATE VIEW ads_tb_trd_and_rfd_pay_ri AS
SELECT
 MIN(gmt_create) gmt_create, --'创建时间'
 MAX(gmt_modified) gmt_modified, --'修改时间'
 CAST (buyer_id AS BIGINT) buyer_id, --'购买ID'
 CAST (date_format(pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMdd') AS BIGINT) AS effect_time, --'购买时间'
 SUM(CAST(div_idx_actual_total_fee/100 AS BIGINT)) cumulate_amount --'总的金额'
FROM
 filter_hxhb_dwd_tb_trd_and_rfd_pay_ri_distinct
GROUP BY
 buyer_id,date_format(pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMdd');
```

Q: 为什么取MAX、MIN?

```
MIN(gmt_create) gmt_create, --'创建时间'
MAX(gmt_modified) gmt_modified, --'修改时间'
```

A: MIN(gmt\_create)是指的订单的第一笔订单的创建时间， MAX(gmt\_modified)是指的最后一比订单的时间。根据订单的业务逻辑您可以用MAX和MIN来取相应的值。

 说明 如果时间字段不是BIGINT类型的可以用相应的内置函数做转换，详情请参见 [内置函数](#)。

- 数据入库

把统计好的数据插入RDS结果，再由其他的推送软件把相应的红包数发放给满足条件的用户。



```
INSERT INTO tddl_output
SELECT
 gmt_create,
 gmt_modified,
 buyer_id,
 cumulate_amount,
 effect_time
from ads_tb_trd_and_rfd_pay_ri
where cumulate_amount>0;
```

Q: 为什么要取总的金额大于0的数?

```
cumulate_amount>0
```

A: 是为了过滤掉在上一步做UNION ALL的退货商品的金额。

## 总结

Q: 在大量的订单和退货单中怎样才能清洗出我们所需要的数据?

A: 在真实的购买记录中会存在大量的购买量和退货量，用UNION ALL把两张或者是多张表合并成一张大表，然后再通过具体的业务逻辑去重、聚合操作。最后把用户所有的真实订单交易金额写入存储中为后续的红包推送做准备。

## DEMO示例以及源代码

根据上文介绍的营销红包解决方案，为您创建了一个包含完整链路的DEMO示例。

- DataHub作为源表
- RDS作为结果表

DEMO代码完整，您可参考示例代码，注册上下游数据，制定自己的营销红包解决方案。单击 [DEMO代码](#) 进行下载。

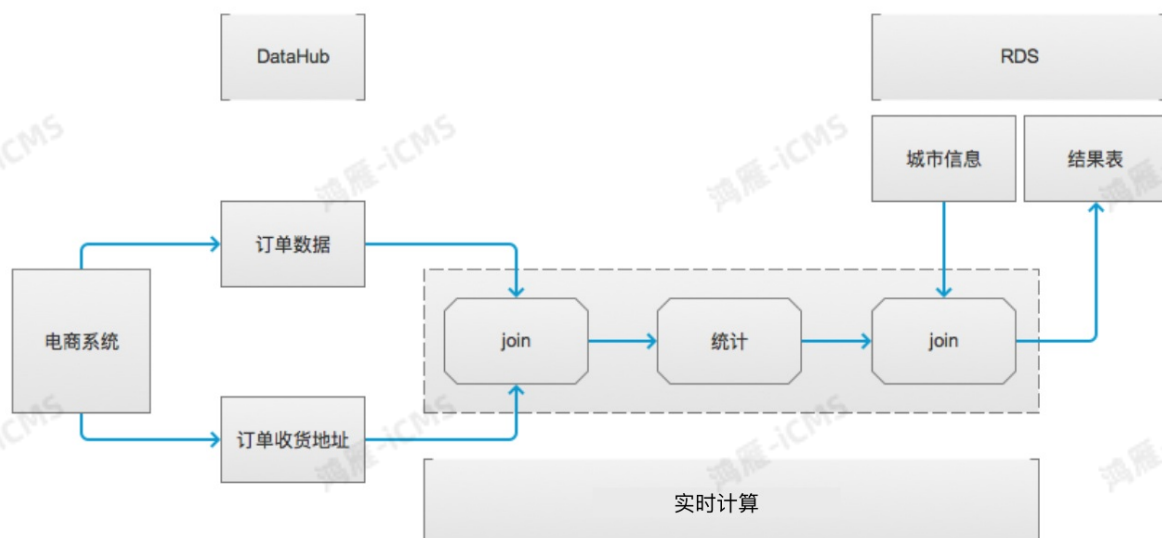
## 10.1.5. 电商场景实战之实时态势感知和订单地理分布

本文通过案例为您介绍如何使用实时计算Flink版完成实时态势感知和订单地理分布。

### 背景信息

实时态势感知和订单地理分布有助于企业及时的优化产品品类的分配和发布。以下是某食品电商企业通过实时计算Flink版完成产品的实时态势感知和订单地理分布的案例。

### 案例



说明 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

创建数据存贮

在电商系统里，订单与订单地址是分开存储（下单人可以给多个下单地址），所以在订单创建时并没有收货地址，只有在订单提交时才真正的知道收货地址。订单地址里保存的是城市的id (city\_id)，为了获取地理信息，还需要一张城市表（存储城市地理信息）。目标是按日统计不同地域订单（总销售额）的分布情况。

订单

```

CREATE TABLE source_order (
 id VARCHAR,-- 订单ID。
 seller_id VARCHAR, --卖家ID。
 account_id VARCHAR,--买家ID。
 receive_address_id VARCHAR,--收货地址ID。
 total_price VARCHAR,--订单金额。
 pay_time VARCHAR --订单支付时间。
) WITH (
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
 project='yourProjectName',--您的project。
 topic='yourTopicName',--您的topic。
 roleArn='yourRoleArn',--您的roleArn。
 batchReadSize='500'
);

```

o 订单地址

```
CREATE TABLE source_order_receive_address (
 id VARCHAR,--收货地址ID。
 full_name VARCHAR,--收货人全名。
 mobile_number VARCHAR,--收货人手机号。
 detail_address VARCHAR,--收货详细地址。
 province VARCHAR,--收货省份。
 city_id VARCHAR,--收货城市。
 create_time VARCHAR --创建时间。
) WITH (
 type='datahub',
 endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
 project='yourProjectName',--您的project。
 topic='yourTopicName',--您的topic。
 roleArn='yourRoleArn',--您的roleArn。
 batchReadSize='500'
);
```

o 城市表

```
CREATE TABLE dim_city (
 city_id varchar,
 city_name varchar,--城市名。
 province_id varchar,--所属省份ID。
 zip_code varchar,--邮编。
 lng varchar,--经度。
 lat varchar,--纬度。
 PRIMARY KEY (city_id),
 PERIOD FOR SYSTEM_TIME --定义为维表。
) WITH (
 type='rds',
 url = 'yourDatabaseURL',--您的数据库URL。
 tableName = 'yourTableName',--您的表名。
 userName = 'yourDatabaseName',--您的用户名。
 password = 'yourDatabasePassword'--您的密码。
);
```

- 按日统计不同地域订单（总销售额）的分布。

```
CREATE TABLE result_order_city_distribution (
 summary_date varchar,--统计日期。
 city_id bigint,--城市ID。
 city_name varchar,--城市名。
 province_id bigint,--所属省份ID。
 gmv double,--总销售额。
 lng varchar,--经度。
 lat varchar,--纬度。
 primary key (summary_date,city_id)
) WITH (
 type= 'rds',
 url = 'yourDatabaseURL',--您的数据库URL。
 tableName = 'yourTableName',--您的表名。
 userName = 'yourDatabaseName',--您的用户名。
 password = 'yourDatabasePassword'--您的密码。
);
```

- 编辑业务逻辑

```
INSERT INTO result_order_city_distribution
SELECT
 d.summary_date,
 cast(d.city_id as BIGINT),
 e.city_name ,
 cast(e.province_id as BIGINT),
 d.gmv,
 e.lng,
 e.lat
from (
 SELECT
 DATE_FORMAT(a.pay_time,'yyyyMMdd') as summary_date ,
 b.city_id as city_id ,
 round(sum(cast(a.total_price as double)),2) as gmv
 from source_order as a
 join source_order_receive_address as b on a.receive_address_id =b.id
 group by DATE_FORMAT(a.pay_time,'yyyyMMdd'),b.city_id
 --双流JOIN，并根据日期和城市ID得到销售额分。
)d join dim_city FOR SYSTEM_TIME AS OF PROCTIME() as e on d.city_id = e.city_id
-- JOIN维表，补齐城市地理信息，得到最终结果。
;
```

## DEMO示例以及源代码

根据上文介绍的实时态势感知和订单地理分布解决方案，阿里云为您创建了一个包含完整链路的DEMO示例，您可以参考DEMO示例，注册上下游数据，制定自己的解决方案。单击[DEMO代码](#)进行下载完整示例。使用示例时，上下游存储请注意以下两点：

- DataHub作为源表
- RDS作为结果表

## 10.1.6. 电商场景实战之最新交易记录获取

本文为您介绍如何利用实时计算获取最新交易记录。

## 背景

在互联网电商的真实交易记录里，订单交易表里会出现不同时间对一笔订单出现多次操作的记录。例如您修改了购买的数量、退货等一系列的操作，但是商家只想获取有效的交易记录。下文为您介绍如何利用实时计算获取最新交易记录。

## 操作流程

- 语法

```
SELECT col1, col2, col3
FROM (
 SELECT col1, col2, col3
 ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
 ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
FROM table_name)
WHERE rownum <= N [AND conditions]
```

| 参数                                             | 说明                      |
|------------------------------------------------|-------------------------|
| ROW_NUMBER()                                   | 计算行号的OVER窗口函数，行号计算从1开始。 |
| PARTITION BY col1[, col2..]                    | 指定分区的列，可选。              |
| ORDER BY col1 [asc desc][, col2 [asc desc]...] | 指定排序的列，可以多列不同排序方向。      |

- 测试案例

- 测试数据

| id (BIGINT) | TIME (VARCHAR) | VALUE (BIGINT) |
|-------------|----------------|----------------|
| 1           | 1517898096     | 5              |
| 1           | 1517887296     | 44             |
| 1           | 1517872896     | 32             |
| 2           | 1517872896     | 10             |

## ○ 测试语句

```
create table source_table (
 id BIGINT,
 `TIME` VARCHAR,
 `VALUE` BIGINT
)with(
 type='datahub',
 endPoint='yourEndpointURL',
 project='yourProjectName',
 topic='yourTableName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret'
);
CREATE TABLE result_table (
 id BIGINT,
 `TIME` VARCHAR,
 `VALUE` BIGINT
) WITH (
 type = 'rds',
 url = 'yourRDSDatabaseURL',
 userName = 'yourDatabaseName',
 password = 'yourDatabasePassword',
 tableName = 'yourTableName'
);
INSERT INTO result_table
SELECT id,`TIME`,`VALUE`
FROM (
 SELECT *,
 ROW_NUMBER() OVER (PARTITION BY id ORDER BY `TIME` desc) AS rownum
 FROM
 source_table
)
WHERE rownum <= 1
;
```

## ○ 测试结果

| id (BIGINT) | TIME (VARCHAR) | VALUE (BIGINT) |
|-------------|----------------|----------------|
| 1           | 1517898096     | 5              |
| 2           | 1517872896     | 10             |

## ● 难点解析

```
SELECT id, `TIME`, `VALUE`
FROM (
 SELECT *,
 ROW_NUMBER() OVER (PARTITION BY id ORDER BY `TIME` desc) AS rownum
 FROM
 source_table
)
WHERE rownum <= 1
;
```

在订单表里有多个业务时间处理的订单，如果您只想取最后时间的订单里面的数据，需要使用 `row_number() OVER (PARTITION BY id ORDER BY TIME DESC)` 表示根据ID分组，在分组内部根据 TIME 降序排列。此函数计算的值表示每组内部排序后的顺序编号（组内是连续且唯一的），这样我们取的就是最新时间的订单里面的数据。over窗口函数的使用，详细请参见[OVER窗口](#)。

## DEMO示例以及源代码

根据上文介绍的最新交易记录获取解决方案，为您创建了一个包含完整链路的DEMO示例：

- DataHub作为源表。
- RDS作为结果表。

DEMO代码完整，您可参考示例代码，注册上下游数据，制定自己的解决方案。单击 [DEMO代码](#) 进行下载。

# 10.2. 视频直播行业最佳实践

## 10.2.1. 视频直播解决方案之视频核心指标监控

本文通过案例为您介绍如何使用实时计算完成视频核心指标监控。

### 背景信息

随着互联网络技术的发展，网络直播受到越来越多人的关注，特别是视频直播生态链。通过网络信号，我们在线收看体育赛事、重大活动和新闻等，真正地体验直播的乐趣。

在体验为王的时代，任何体验不佳均会导致客户群体大规模的流失。对于网站直播平台而言，需要重点关注以下几点：

- 用户（主播和粉丝）的使用体验，重点关注系统指标包括音视频卡顿率、延迟率、丢包等情况。
- 直播平台具有时效性，平台方更需要实时了解系统周边问题，并先于用户发现并定位问题和故障。
- 平台运营方还应该及时跟踪和了解网站整体的客户运营情况，以及视频爆款等情况。

下面我们以某个视频直播平台方为例，讲解如何使用实时计算监控系统稳定性和平台运营情况。

### 业务详情

为了最大程度活跃用户社群，覆盖更多场景直播频道，最终实现平台方盈利，业务通常是这样实现的：

- 同一个视频直播网站有多个主播。
- 每个主播向一个频道内的用户进行广播。
- 用户可以看到当前频道内的主播视频，并听到其声音。
- 主播可以与频道内的多个用户进行私聊。

#### 业务流程



1. 主播和粉丝各自持有直播App软件，该App软件每10秒向服务器打点。
2. 服务器接收到打点信息会输出到本地磁盘，阿里云日志服务采集打点数据。
3. 实时计算订阅该日志服务。
4. 实时计算客户端视频播放情况。

## 业务目标

针对客户端App的监控，获取以下指标：

- 房间故障，包括卡顿、丢帧、音视频不同步等。
- 分地域统计数据端到端延迟平均情况。
- 统计实时整体卡顿率（出现卡顿的在线用户数/在线总用户数\*100%，通过此指标可以衡量当前卡顿影响的人群范围）。
- 统计人均卡顿次数（在线卡顿总次数/在线用户数，通过此指标可以从卡顿频次上衡量整体的卡顿严重程度）。

最终我们希望将上述数据实时计算并写入阿里云关系型数据库RDS（Relational Database Service），并提供在线数据报表展示、大屏展示、甚至部分监控告警。

## 数据格式

客户端App向服务器打点日志格式如下。

| 字段     | 含义       |
|--------|----------|
| ip     | 用户端IP    |
| agent  | 端类型      |
| roomid | 房间号      |
| userid | 用户ID     |
| abytes | 音频码率     |
| afcnt  | 音频帧数     |
| adrop  | 音频丢帧数量   |
| afts   | 音频时间戳    |
| alat   | 音频帧端到端延迟 |



| 字段        | 含义       |
|-----------|----------|
| vbytes    | 视频码率     |
| vfcnt     | 视频帧数     |
| vdrop     | 视频丢帧数量   |
| vfts      | 视频时间戳    |
| vlat      | 视频帧端到端延迟 |
| ublock    | 上行卡顿次数   |
| dblock    | 下行卡顿次数   |
| timestamp | 打点时间戳    |
| region    | 地域       |

日志服务内部是半结构化存储，上述数据将展现出来格式如下。

```
{
 "ip": "ip",
 "agent": "agent",
 "roomid": "123456789",
 "userid": "123456789",
 "abytes": "123456",
 "afcnc": "34",
 "adrop": "3",
 "afts": "1515922566",
 "alat": "123",
 "vbytes": "123456",
 "vfcnc": "34",
 "vdrop": "4",
 "vfts": "1515922566",
 "vlat": "123",
 "ublock": "1",
 "dblock": "2",
 "timestamp": "15151922566",
 "region": "hangzhou"
}
```

## SQL编写

- 数据清洗

首先在实时计算中声明源表。

```
CREATE TABLE app_heartbeat_stream_source (
 ip VARCHAR,
 agent VARCHAR,
 roomid VARCHAR,
 userid VARCHAR,
 abytes VARCHAR,
 afcnt VARCHAR,
 adrop VARCHAR,
 afts VARCHAR,
 alat VARCHAR,
 vbytes VARCHAR,
 vfcnt VARCHAR,
 vdrop VARCHAR,
 vfts VARCHAR,
 vlat VARCHAR,
 ublock VARCHAR,
 dblock VARCHAR,
 `timestamp` VARCHAR,
 app_ts AS TO_TIMESTAMP(CAST(`timestamp` AS BIGINT)), --定义生成WATERMARK的字段。
 WATERMARK FOR app_ts AS withOffset(app_ts, 10000) --WATERMARK比数据时间线性增加10秒。
) WITH (
 type ='sls',
 endPoint ='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId ='xxxxxxxxxx',
 accessKey ='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
 project ='xxxx',
 logStore ='app_heartbeat_stream_source',
);
```

源端业务为考虑业务方便，将所有的数据均处理为VARCHAR类型。为了方便后续处理，我们将上述源表进行数据清洗，清洗目标为：

- i. 格式转换。将部分VARCHAR类型转为BIGINT。
- ii. 业务补充。例如，填写地域相关信息。

```
CREATE VIEW view_app_heartbeat_stream AS
SELECT
 ip,
 agent,
 CAST(roomid AS BIGINT),
 CAST(userid AS BIGINT),
 CAST(abytes AS BIGINT),
 CAST(afcmt AS BIGINT),
 CAST(adrop AS BIGINT),
 CAST(afts AS BIGINT),
 CAST(alat AS BIGINT),
 CAST(vbytes AS BIGINT),
 CAST(vfcm AS BIGINT),
 CAST(vdrop AS BIGINT),
 CAST(vfts AS BIGINT),
 CAST(vlat AS BIGINT),
 CAST(ublock AS BIGINT),
 CAST(dblock AS BIGINT),
 app_ts,
 region
FROM
 app_heartbeat_stream_source;
```

- 房间故障统计

统计房间故障，故障包括卡顿、丢帧、音视频不同步信息，使用10分钟一个窗口进行统计。

```
CREATE VIEW room_error_statistics_10min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
 roomid,
 SUM(ublock) as ublock, --统计10分钟内上行卡顿次数。
 SUM(dblock) as dblock, --统计10分钟内下行卡顿次数。
 SUM(adrop) as adrop, --统计10分钟内音频丢包次数。
 SUM(vdrop) as vdrop, --统计10分钟内视频丢包次数。
 SUM(alat) as alat, --统计10分钟内音频延迟。
 SUM(vlat) as vlat, --统计10分钟内音频延迟。
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '10' MINUTE), roomid;
```

- 分地域统计延迟情况

分地域统计数据端到端延迟平均情况，每10分钟统计音频、视频平均延迟情况。

```
CREATE VIEW region_lat_statistics_10min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
 region,
 SUM(alat)/COUNT(alat) as alat,
 SUM(vlat)/COUNT(vlat) as vlat,
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '10' MINUTE), region;
```

- 实时整体卡顿率

统计实时整体卡顿率，即出现卡顿的在线用户数/在线总用户数\*100%，通过此指标可以衡量当前卡顿影响的人群范围。

```
CREATE VIEW block_total_statistics_10min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
 SUM(IF(ublock <> 0 OR dblock <> 0, 1, 0)) / CAST(COUNT(DISTINCT userid) AS DOUBLE) as block_rate, --C
COUNT(DISTINCT) 需要在1.4.4版本以上Blink才能支持
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '10' MINUTE);
```

- 统计人均卡顿次数

统计人均卡顿次数，即在线卡顿总次数/在线用户数，通过此指标可以从卡顿频次上衡量整体的卡顿严重程度。

```
CREATE VIEW block_peruser_statistics_10min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
 SUM(ublock+dblock) / CAST(COUNT(DISTINCT userid) AS DOUBLE) as block_peruser, --COUNT(DISTINCT)
需要在1.4.4版本以上Blink才能支持
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '10' MINUTE);
```

## Demo示例以及源代码

根据上文介绍的视频直播解决方案之视频核心指标监控，阿里云为您创建了一个包含完整链路的Demo示例，参考示例代码，您可注册上下游数据，制定自己的视频核心指标监控的解决方案。单击下载[示例](#)。使用示例时，您需要选择CSV文件上传至DataHub作为源表，RDS作为结果表来存储结果数据。

f

## 10.2.2. 视频直播解决方案之直播数字化运营

本文通过案例为您介绍如何使用实时计算完成直播数字化运营的视频直播解决方案。

### 直播数字化运营业务

本文主要关注的是直播数据化运营业务。为您介绍使用实时计算完成对当前直播间运营情况实时的监控，包括热门视频、用户走势等等。

## 解决方案

- 业务目标
  - 全站观看直播总人数以及走势。
  - 房间直播总人数以及走势。
  - 热门直播房间及主播Top10，分类目主播Top10。

- 数据格式

参考上面的数据格式，使用App打点日志作为原始数据进行统计。

客户端App向服务器打点日志格式如下。

| 字段        | 含义       |
|-----------|----------|
| ip        | 用户端IP    |
| agent     | 端类型      |
| roomid    | 房间号      |
| userid    | 用户ID     |
| abytes    | 音频码率     |
| afcmt     | 音频帧数     |
| adrop     | 音频丢帧数量   |
| afts      | 音频时间戳    |
| alat      | 音频帧端到端延迟 |
| vbytes    | 视频码率     |
| vcmt      | 视频帧数     |
| vdrop     | 视频丢帧数量   |
| vfts      | 视频时间戳    |
| vlat      | 视频帧端到端延迟 |
| ublock    | 上行卡顿次数   |
| dblock    | 下行卡顿次数   |
| timestamp | 打点时间戳    |
| region    | 地区       |

日志服务内部是半结构化存储，上述数据将展现出来格式如下。

```
{
 "ip": "ip",
 "agent": "agent",
 "roomid": "123456789",
 "userid": "123456789",
 "abytes": "123456",
 "afcmt": "34",
 "adrop": "3",
 "afts": "1515922566",
 "alat": "123",
 "vbytes": "123456",
 "vfcmt": "34",
 "vdrop": "4",
 "vfts": "1515922566",
 "vlat": "123",
 "ublock": "1",
 "dblock": "2",
 "timestamp": "1515922566",
 "region": "hangzhou"
}
```

- SQL编写

- 全站总人数及走势

使用每分钟一个窗口统计全站观看人数走势，走势最近的1分钟计算结果就是当前总人数。

```
CREATE VIEW view_app_total_visit_1min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
 COUNT(DISTINCT userid) as app_total_user_cnt
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '1' MINUTE);
```

- 房间直播总人数以及走势

同样，使用每分钟一个窗口统计房间内观看人数走势，走势最近的1分钟计算结果就是当前房间人数。

```
CREATE VIEW view_app_room_visit_1min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
 roomid as room_id,
 COUNT(DISTINCT userid) as app_room_user_cnt
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '1' MINUTE), roomid;
```

- 热门直播房间排名

统计热门直播实际上为首页推荐用户使用，把热门的房间放在首页推广能够获得更多流量和单击。

```
CREATE VIEW view_app_room_visit_top10 AS
SELECT
 app_ts,
 room_id,
 app_room_user_cnt,
 ranking
FROM
(
 SELECT
 app_ts,
 room_id,
 app_room_user_cnt,
 ROW_NUMBER() OVER (PARTITION BY 1 ORDER BY app_room_user_cnt desc) AS ranking
 FROM
 view_app_room_visit_1min
) WHERE ranking <= 10;
```

- 分类目统计热门直播房间排行

为了最大程度活跃用户社群，覆盖更多场景直播频道，最终实现平台方盈利，通常同一个视频直播网站会开设大量的不同频道，以迎合各类不同人群的观看需求。例如，淘宝直播就会有美妆、男装、汽车、健身等多个不同类目。

类目和房间关系表是在RDS存储的映射表。

```
CREATE TABLE dim_category_room (
 id BIGINT,
 category_id BIGINT,
 category_name VARCHAR,
 room_id BIGINT
 PRIMARY KEY (room_id),
 PERIOD FOR SYSTEM_TIME --定义为维表。
) WITH (
 type= 'rds',
 url = 'xxxx',--您的数据库url。
 tableName = 'xxx',--您的表名。
 userName = 'xxx',--您的用户名。
 password = 'xxx'--您的密码。
);
```

针对不同的房间，关联类目表，并分类统计各自的排名情况。

```
CREATE VIEW view_app_room_visit_1min AS
SELECT
 CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
 roomid as room_id,
 COUNT(DISTINCT userid) as app_room_user_cnt
FROM
 view_app_heartbeat_stream
GROUP BY
 TUMBLE(app_ts, INTERVAL '1' MINUTE), roomid;
--关联类目表。
CREATE VIEW view_app_category_visit_1min AS
SELECT
 r.app_ts,
 r.room_id,
 d.category_id,
 d.category_name,
 r.app_room_user_cnt
FROM
 view_app_room_visit_1min r
JOIN
 dim_category_room d
ON
 r.room_id = d.room_id;
--计算不同类目下房间号排名。
CREATE VIEW view_app_category_visit_top10 AS
SELECT
 app_ts,
 category_id,
 category_name,
 app_room_user_cnt,
 ranking
FROM
 (
 SELECT
 app_ts,
 room_id,
 category_id,
 category_name,
 app_room_user_cnt,
 ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY app_room_user_cnt desc) AS ranking
 FROM
 view_app_category_visit_1min
) WHERE ranking <= 10;
```

## Demo示例以及源代码

阿里云为您创建了一个包含完整链路的Demo示例，您可参考 [代码示例](#)，注册上下游数据，制定自己的直播数字化运营的解决方案。在使用示例时，上下游存储请注意以下几点：

- 选择CSV文件上传至DataHub作为源表。
- RDS作为维表。
- RDS作为结果表。



# 11.相关协议

## 11.1. 实时计算服务等级协议（SLA）

实时计算服务等级协议（SLA）的详情，请参见 [阿里云实时计算服务等级协议](#)。