大数据计算

产品文档

文档目录

产品简介
流计算产品概述

最近更新时间: 2019-11-26 14:47:19

流计算产品致力于给用户提供更高效、更稳定、更易用的流计算处理服务，解决实时数据处理需求，如网站点击流量分析、电商精准推荐、金融实时风控等流计算应用。

流计算组件通过借助数据开发、智能调度等组件的能力，提供流计算开发测试环境到部署运维等一站式服务。在使用流计算服务时，需要依赖数据采集组件将大数据云外的流式数据采集到Kafka中统一存储。同时，通过数据管理组件获取元数据信息后，在数据开发组件提供的IDE环境中完成流计算应用的开发，最终通过流计算引擎完成实时数据处理。

本文档主要描述流计算服务的功能简介、快速入门、操作指南、常见问题等内容。

相关概念

最近更新时间: 2019-11-26 14:47:19

使用流计算产品时，通常会涉及到以下概念：

- 实时流数据：实时、持续生成的数据，如业务日志、系统日志等各类日志信息。
- 主题（topic）：流计算服务订阅和发布的最小单位，用户可以用topic表示一种流数据，类似与数据库中的表（table）。在大数据云服务中，一个流计算服务的数据源对应一个topic，单个topic可以存储一个或多个日志中的流式数据。
- 流连接分区（partion）：topic存储数据的最小单元，对于吞吐较高的topic，可以创建多个分区。
- 状态保存点（SavePoint）：SavePoint由用户手动触发，可以支持程序升级后，继续从升级前的那个点开始执行计算， 保证数据不中断。
- CU（Compute Unit）：流计算所提供计算资源的最小单位，1CU的具体含义为：CPU 1核、内存 4GB。

产品优势

最近更新时间: 2019-11-26 14:47:19

流计算产品具有以下优势：

- 优良的计算性能 流计算支持每秒千万级数据处理，响应延迟达到毫秒级。
- 简单易用的计算服务 上层大数据加工提供这些服务的可视化开发界面，降低使用和开发门槛。借助数据管理服 务，能迅速获取元数据信息，解耦与数据源的依赖，实现端到端的数据处理。
- 灵活的资源规划 用户可根据自己的需求，灵活划分资源组，满足个性需求，不浪费。

使用与约束

最近更新时间: 2021-09-15 15:59:08

支持的流计算引擎：流计算服务支持Flink、Spark Streaming两种计算引擎，其中Flink为全功能支持， Spark Streaming只支持上传JAR包方式使用，推荐用户优先使用Flink引擎，两种引擎的对比如下：

流计算引擎对比	Flink	Spark Streaming
引擎版本	Flink 1.11.3	Spark 2.3.0
产品生态	Apache顶级项目，Flink 产品发展速度快，在流处理方面处于业界最前沿水平	归属于Spark的子项目，Spark社区规模大，产品相对更加成熟
运行模式	事件驱动	时间驱动（微批处理，运行的时候需要指定批处理的时间，每次运行 任务时处理一个批次的数据）
时间机制	支持三种时间机制：事件时间，注入时间，处理时间	只支持处理时间
支持程度	在大数据云服务中，支持可视化拖拽、上传JAR包两种开发方式	在大数据云服务中，只支持JAR包的开发方式

支持的source数据源、sink数据源：流计算服务支持的source源、sink源如下：

	开源组件	组件版本
支持的source源	Kafka	0.10及以上版本
支持的sink源	Kafka	0.10及以上版本
	Oracle	11.2.0.1.0
	MySQL	5.6及以上版本
	HBase	2.0
	Redis	4.0
	Elasticsearch	6.4.2
	InfluxDB	2.1.5
	Phoenix	5.0.0
	Hive	3.1.0

应用场景

最近更新时间: 2019-11-26 14:47:19

流计算产品具有以下应用场景：

- 互联网点击流分析：用户在网站浏览时会产生许多的点击行为，对这些点击行为加以分析可精确把握热点趋势。借助于云端流计算服务，构建实时分析可达到分钟级，对用户行为数据进行实时汇聚分析，持续地挖掘出有价值信息，帮助更好地做出运营决策，改进用户体验。
- 金融实时风控：在众多金融风险中，及早探测到风险往往能有效地减少损失，将金融 交易大数据与流计算服务相结合，引入特征模型算法，及早地过滤出诸如盗刷卡等异 常交易行为，实施风险控制，提升金融安全性。
- 物联网监控：在工业设备的运转过程中，及早发现潜在故障会极大降低维修成本。借 助于云端流计算服务，及时收集设备传感器数据，并进行聚合、分析筛选，可实现秒 级设备异常告警，提升设备利用率。
- 电商精准推荐：在电商交易中，借助于云端流计算服务，实时提取特征变量，及时跟 踪用户关注品类，预测用户消费趋势，为精准推荐提供基础能力。从而提升用户购物 体验，促进消费行为。

快速入门
查看服务

最近更新时间: 2019-11-11 07:47:14

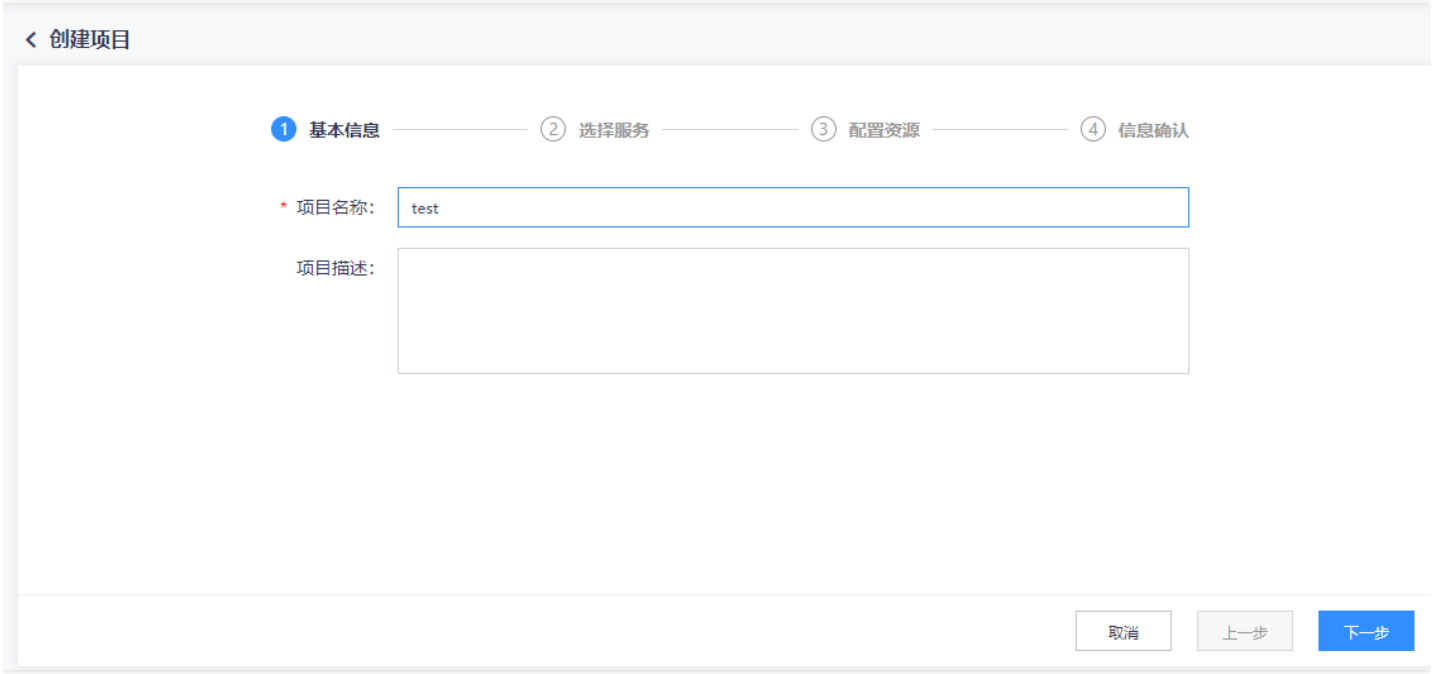
登录后，点击左侧的“资源管理-产品与服务”菜单，或点击上方的“产品与服务”，可查看所有开通的服务。

创建项目

最近更新时间: 2019-11-11 07:47:14

点击页面左侧“项目空间-我的项目”选项卡，可以看到当前所有的项目列表。

点击蓝色的“新建项目”按钮，弹出“新建项目”对话框。



这里可以根据需要选择各项服务，本次新建项目需包含“流计算”，其余服务按需选择。点击“下一步”配置项目的各项属性参数。

点击下一步，在项目资源组页面，通过下拉框中，为所选的服务选择一个资源分组。配置完成后，点击“下一步”。

对以上配置

信息进行确认，无误后点击“确定”按钮。

操作指南

进入项目

最近更新时间: 2019-11-11 07:47:14

创建成功的项目会显示在项目列表的最上面。点击“修改服务”可以对项目中的服务进行新建、修改配置、删除等操作。点击“开通的服务”列中的流计算，进入此项目下流计算的开发页面。

创建数据源和topic（以Kafka举例）

新建数据源

最近更新时间: 2019-11-11 07:47:14

为了将数据从指定表中读写，需要按照业务需求创建对应的数据源/库/表。在“数据管理”→“数据源管理”中新建数据源。

serverAddress填写对应Kafka集群地址，可以进行连通性测试，最后点击“确认”。


建行大数据云
控制台首页
产品与服务
帮助与文档
数据管理
数据源管理
元数据管理
数据权限管理
权限审批
数据授权
申请记录
数据质量
规则配置
质量查询
质量告警
数据管理 > 数据源管理
新增数据源
*数据源名称:
test_kafka
*数据源环境:
测试环境
生产环境
测试和生产
*生产环境配置:
*serverAddress:
127.0.0.1:9092
连通性测试
*是否托管:
是
托管元数据请配置管理员账号,允许平台创建库表并管理数据权限
数据源描述:
上一步
完成
输入数据源名称查询
数据源描述
操作
编辑

设计态–新建topic

最近更新时间: 2019–11–11 07:47:14

在“数据管理”→“元数据管理”→“库表管理”中新建数据库和数据表（Kafka不需要新建数据库，直接新建topic即可）。

新建topic时需指定topic归属的项目，即此topic可在归属项目下使用。

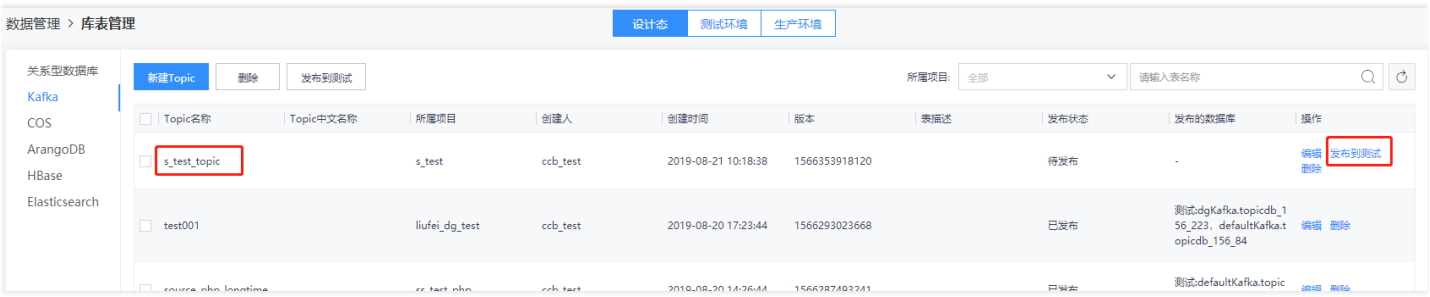

创建topic
1 配置基本信息
2 字段设置
* Topic名称：
仅支持英文、数字、下划线，且不能以数字和下划线开头，最大100字符
Topic中文名：
* 所属项目：
请选择
* Topic数据格式：
json
CSV
* Topic分区数：
生命周期：
不填写默认7天
Topic描述：
取消
下一步


创建topic
1 配置基本信息
2 字段设置
字段名称
字段中文名
字段类型
字段长度
字段密级
字段脱敏
字段描述
操作
id
字符串
50
5
不脱敏
下移 删除
name
字符串
50
5
不脱敏
上移 删除
新增字段
上一步
完成

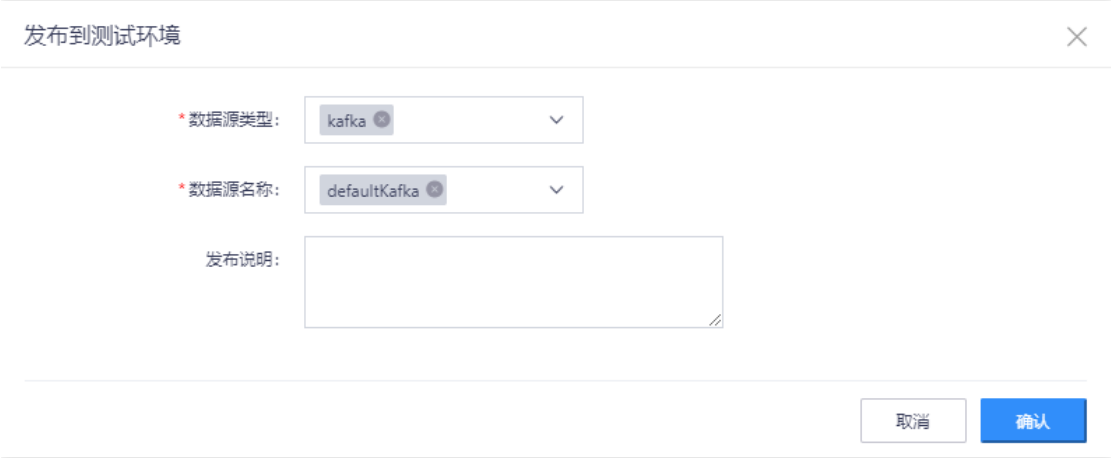
发布到测试态

最近更新时间: 2019-11-11 07:47:11

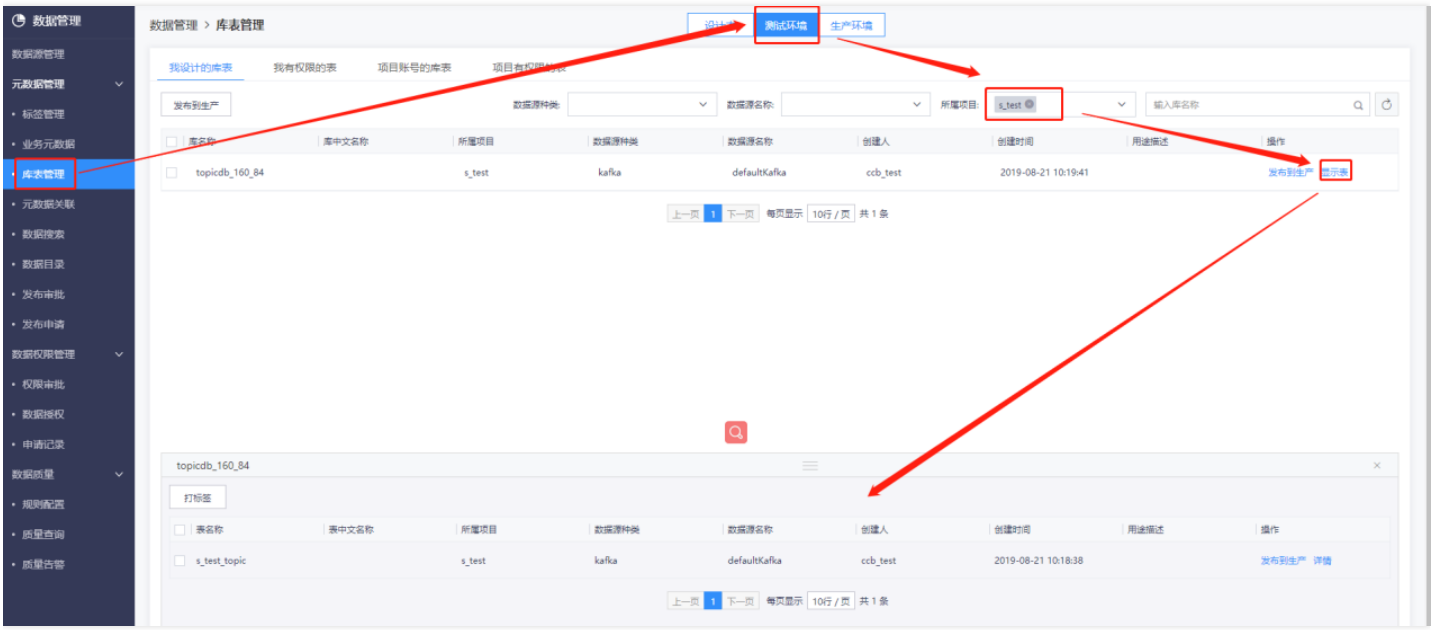
完成topic创建动作后，topic信息可在“库表管理的”的发布态查看，点击列表操作中的“发布至测试”，可申请将topic发布到测试环境，并在测试环境使用。



发布测试时，需指定此topic归属的数据源。



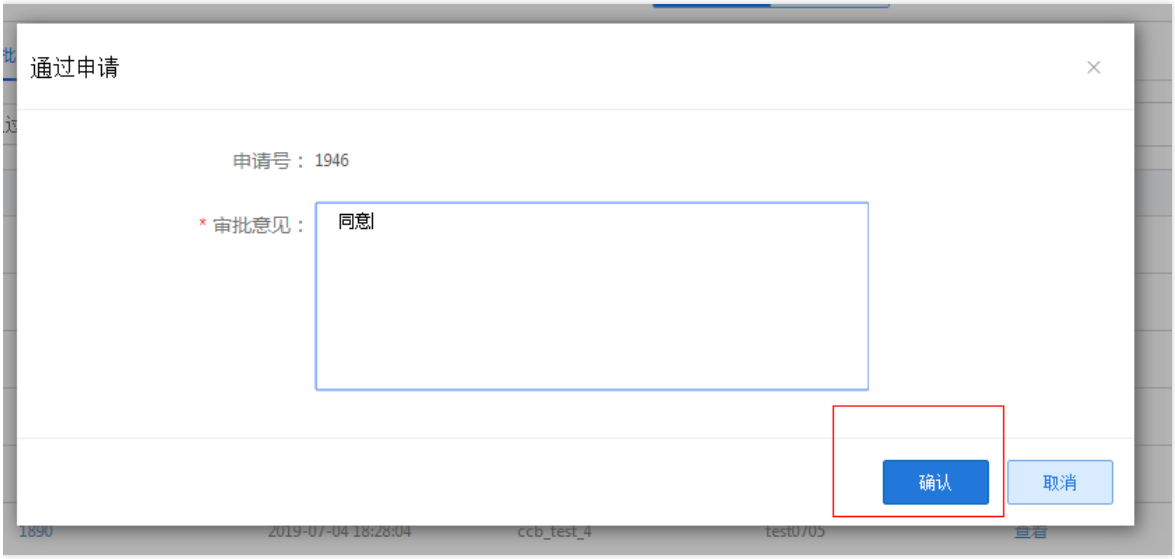
发布测试完毕后，可在测试环境中筛选查看:



测试态审批

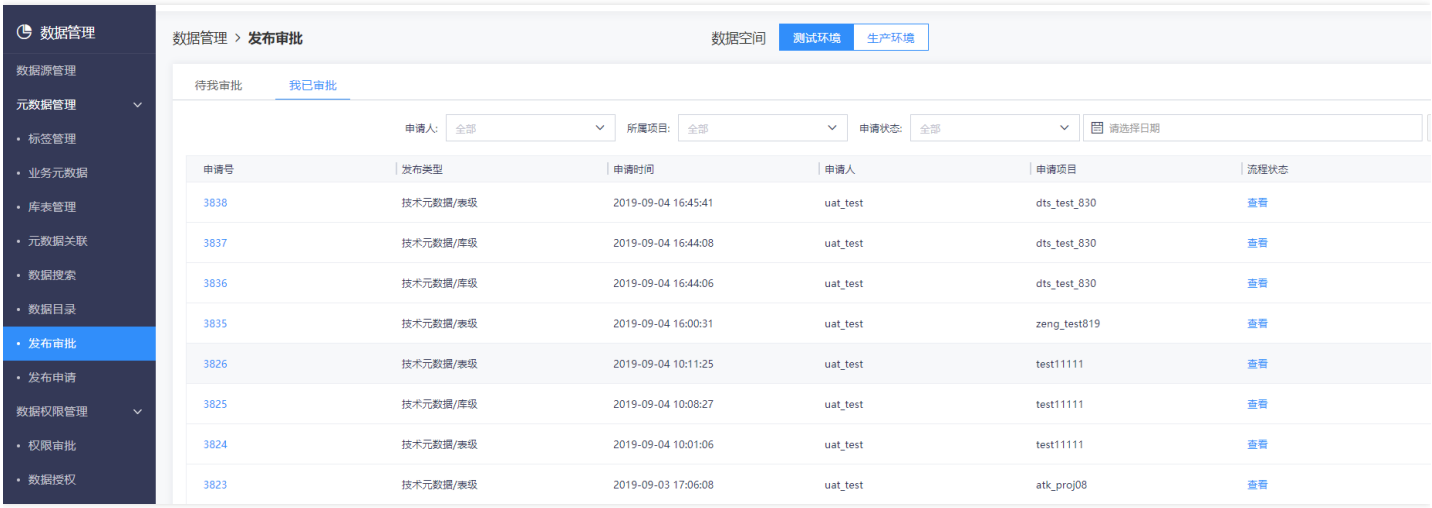
最近更新时间: 2019-11-11 07:47:11

使用审批账号登陆大数据云平台后，在“数据管理”→“元数据管理”→“发布审批”→“测试环境”→“待我审批”中可以查看待审批的数据库/表，点击审批通过，并在弹出的窗口中，填写审批意见后，点击“确认”，即完成了审批流程。审批后，待发布的数据库/表成功发布至测试环境。



在“测试环境”选项卡下的“我

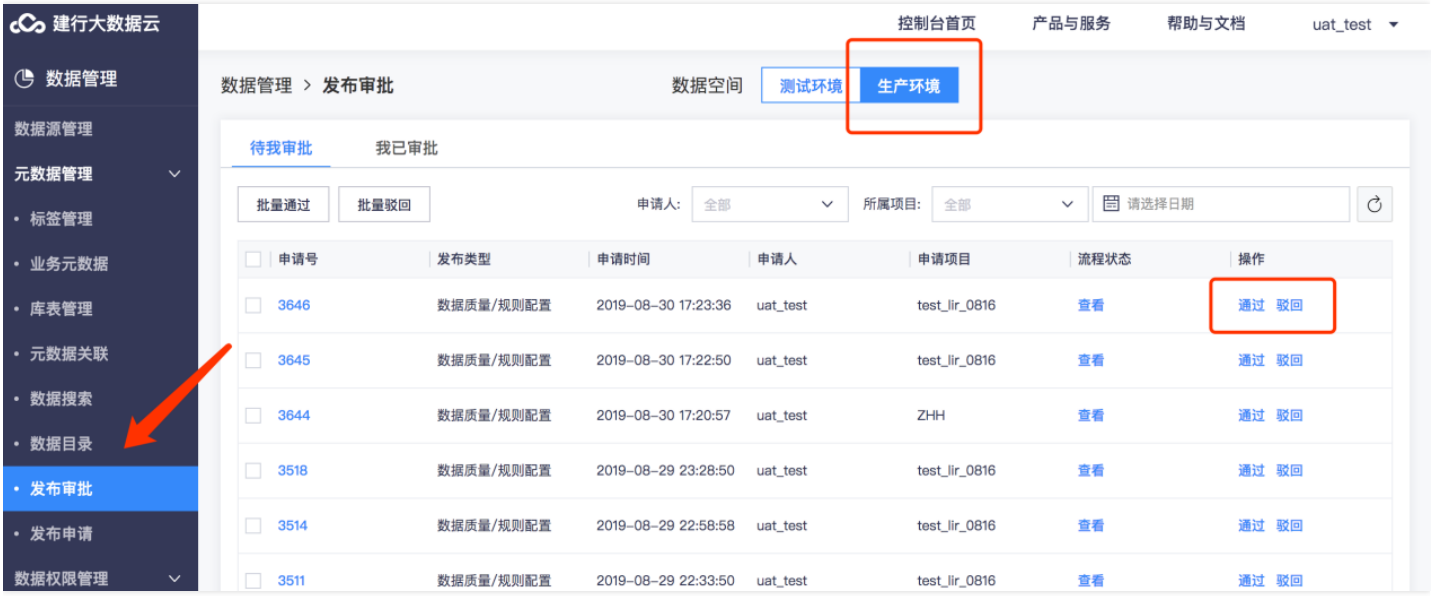
已审批”选项卡下，可以看到所有已审批通过的数据库/表。



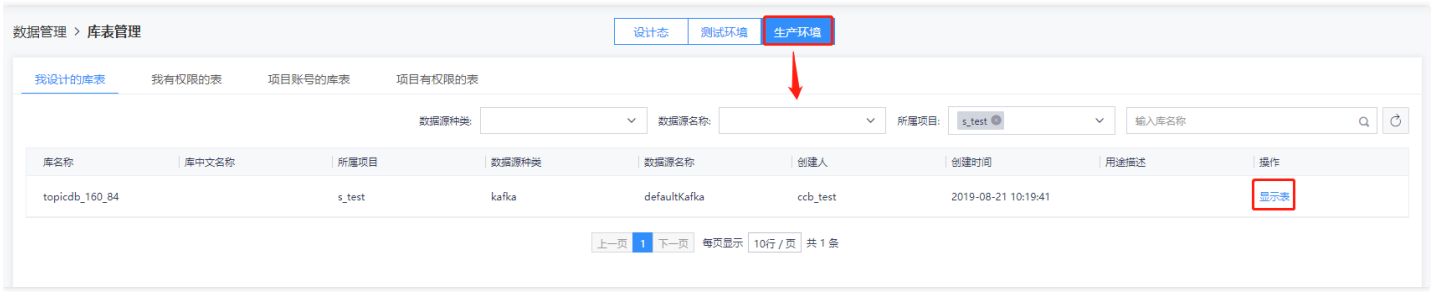
发布到生产态

最近更新时间: 2019-11-11 07:47:11

将topic发布到测试后，点击“操作”列“显示表”，页面下方会显示该数据库中所有的表，其中每个表对应一个topic。同样，点击“操作”列“发布到生产”，经过同样的审批流程后，即可将该topic发布到生产。



审批通过后，点击“生产环境”，在“我设计的库表”选项卡下，按所属项目进行筛选后，即可看到已发布到生产的库，点击“显示表”，即可看到已发布到生产的topic。



申请数据权限（跨项目的数据库表）

申请数据权限

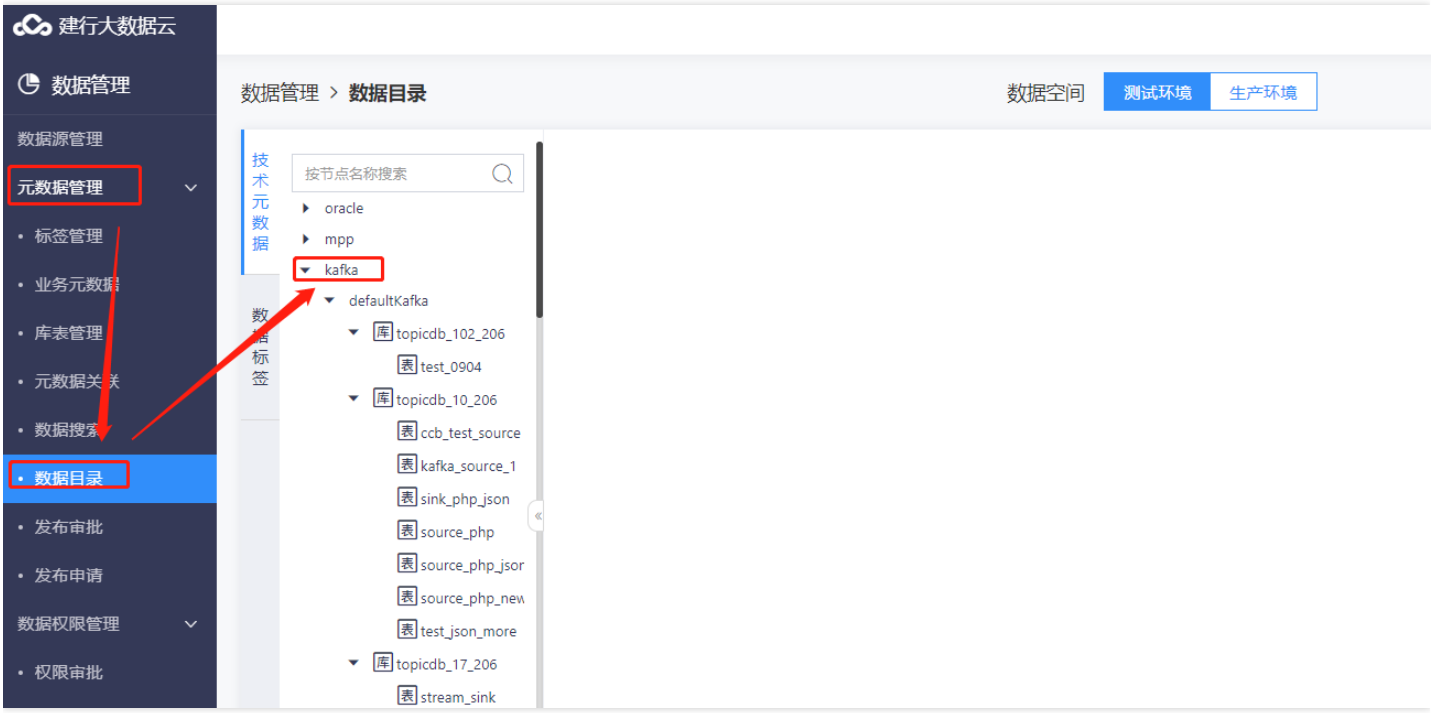
最近更新时间: 2019-11-11 07:48:26

流计算作业如果需要使用到其他项目的数据表均需要提前通过数据管理组件进行申请和注册。

申请权限

最近更新时间: 2019-11-11 07:47:11

可通过如下步骤申请权限：（1）除数据管理员角色以外的其他角色的人员（包括：项目管理员，开发人员，运维人员，业务人员等）进入数据管理模块，点击“元数据管理”下面的“数据目录”。



(2) 选中数据源Kafka，搜索表名称，点击对应的表，会展示出表详情。(3) 点击【权限申请】按钮，弹出权限申请的页签，选择申请对象，选择权限，选择申请对象，选择申请项目，点击【确认】按钮，权限申请成功。

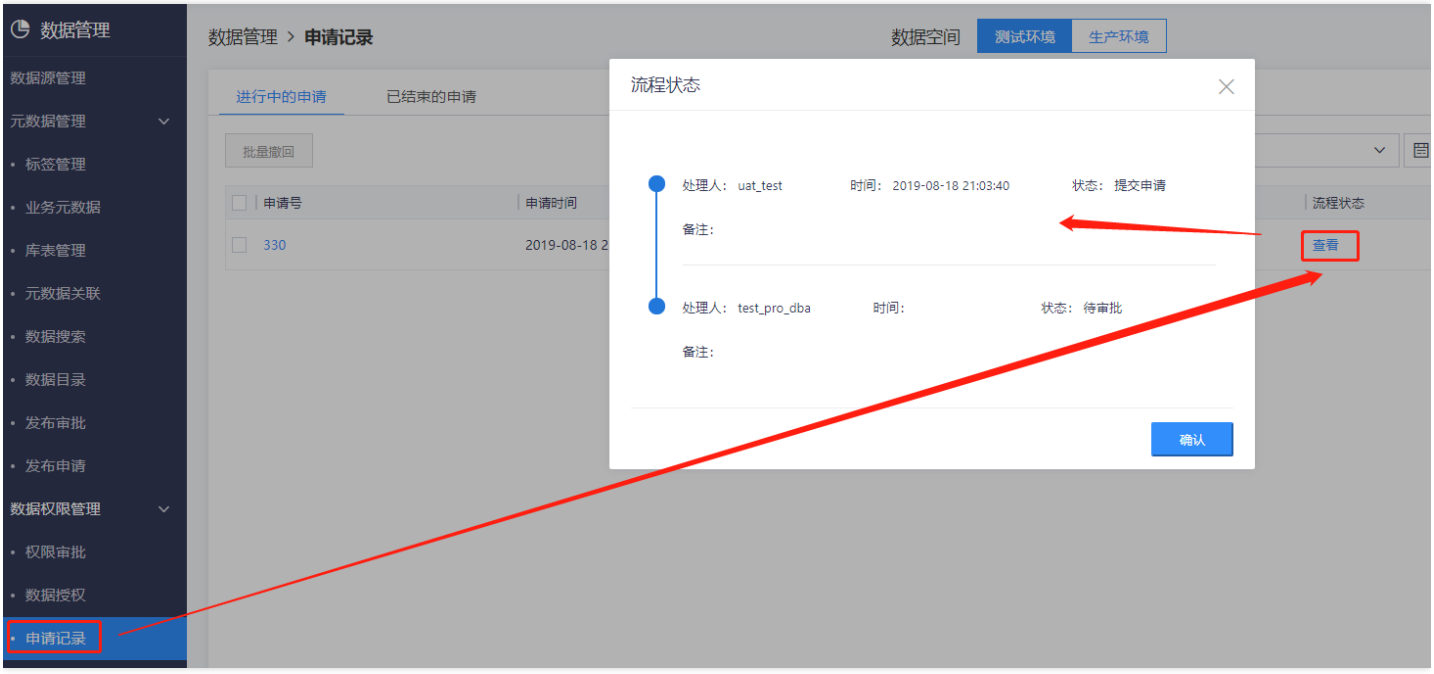
说明： 1. 申请类型为列权限时，申请对象只能为个人；申请类型为表权限时，申请对象可以为项目，个人或者项目和个人。 2. Kafka类型数据源，没有列权限，只有表权限，不需要选择申请类型；Kafka类型数据源的权限选择只有两个选项：insert和select。 3. 申请个人和项目的表权限，会同时生成两条申请记录，审批记录也是两条。

权限审批

最近更新时间: 2019-11-11 07:47:11

可通过如下步骤审批权限：（1）使用数据管理员账号登录进入数据管理模块，点击“数据权限管理”，展开下拉列表中点击“权限审批”。

(2) 根据申请号选择提交的申请，点击“查看”可查看提交申请的用户，备注中显示提交原因。

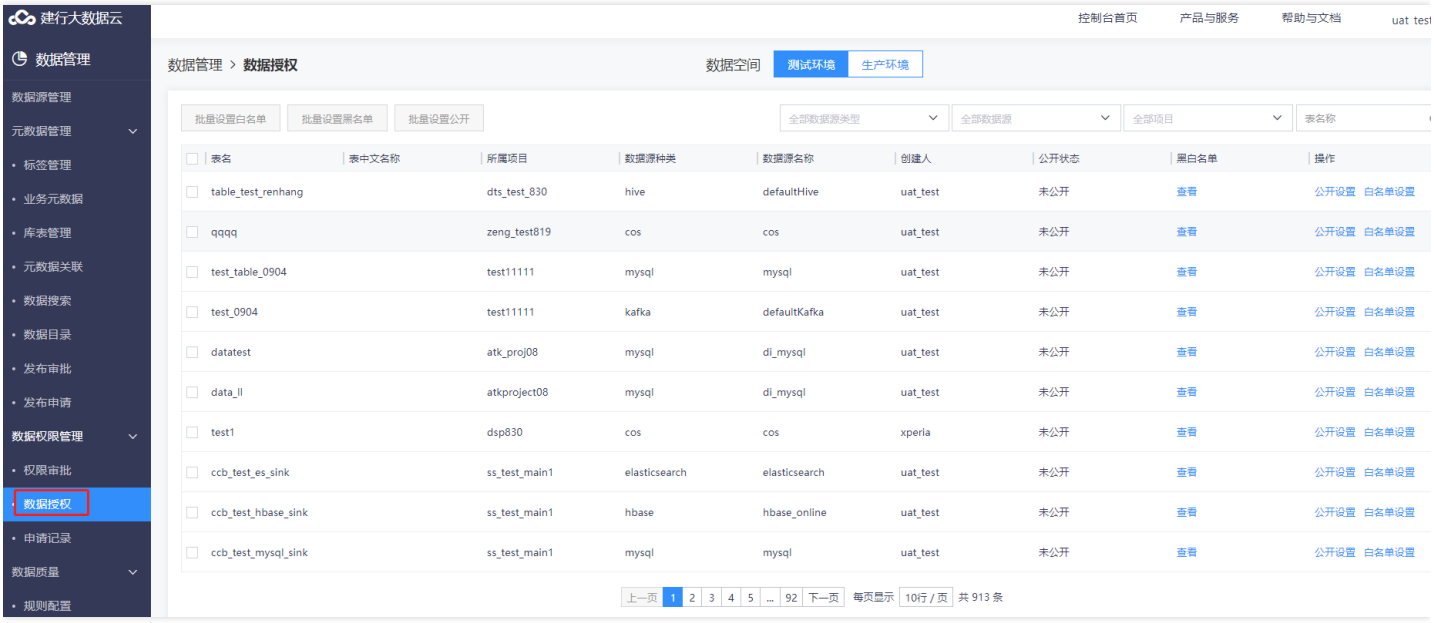


(3) 点击“通过”（或者驳回）审批该申请。 说明：通过申请，如果申请的时候选择的是项目，该表的项目白名单会新增一条数据，项目管理员对该数据表有对应的权限；如果选择的是个人，用户白名单新增一条数据，申请人对该表有对应的权限。

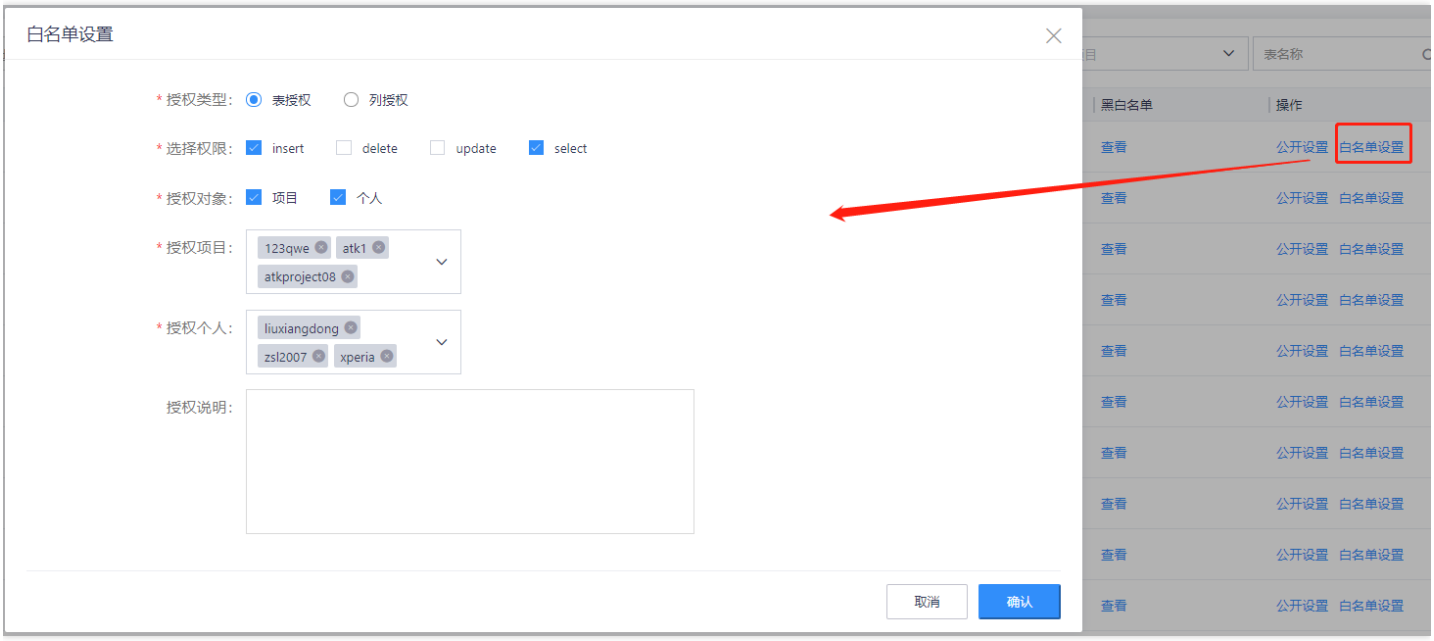
数据授权

最近更新时间: 2019-11-11 07:47:11

可通过如下步骤进行数据授权：（1）使用数据管理员或者租户的账号登录，进入数据管理模块，点击“数据权限管理”，展开的下拉框中点击“数据授权”，数据空间显示数据授权列表。



(2) 选中需要授权的表，点击“白名单设置”按钮，弹出白名单设置标签，选择授权类型，选择权限，选择授权对象，授权项目和个人都支持多选，点击“确认”。 说明：授权类型为表权限时，授权对象可以为项目，个人或者项目和个人；授权对象为列权限时，授权对象只可以为个人；个人和项目均支持多选和搜索。

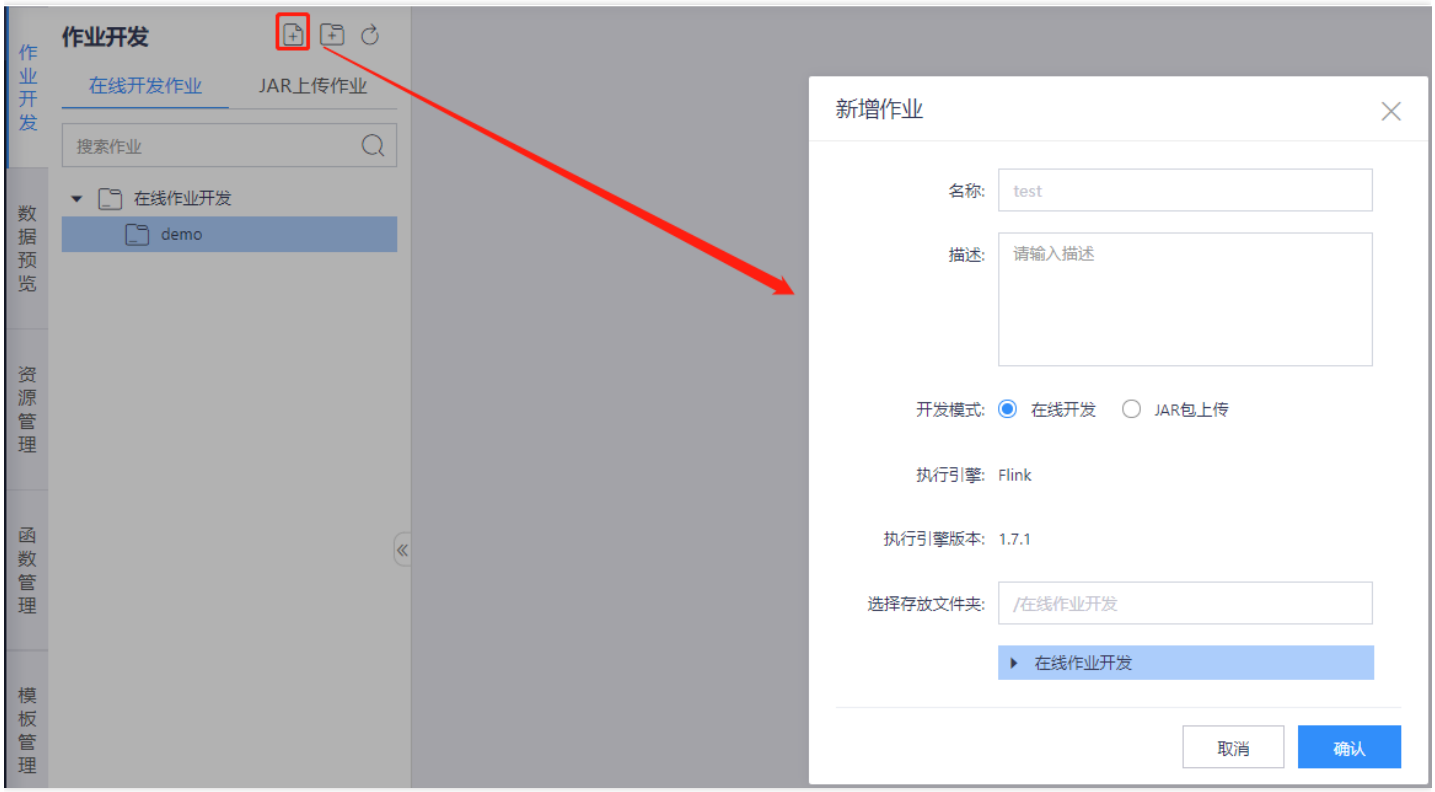


(3) 授权成功后，点击“查看”按钮，可以看到授权的项目白名单和个人白名单。 说明：查看项目白名单和用户白名单，点击“新增白名单”按钮，可以给表增加白名单，操作类似白名单设置。

流计算作业开发
新建作业

最近更新时间: 2019-11-11 07:47:11

进入流计算组件，点击“新建文件夹”和“新建作业”，创建一个新的流计算作业，创建完毕后即进入流计算作业的开发IDE界面。 左侧为本项目流计算任务树，右侧为任务组件和画布区。可以通过拖拽算子和连线轻松编辑流计算作业。 创建文件夹/作业时，支持：在线开发和JAR包上传2种模式，其中在线开发模式支持Flink引擎；JAR包上传模式，依赖线下开发并上传至云平台的JAR包，支持Flink和Spark Streaming两种引擎。



在线开发模式

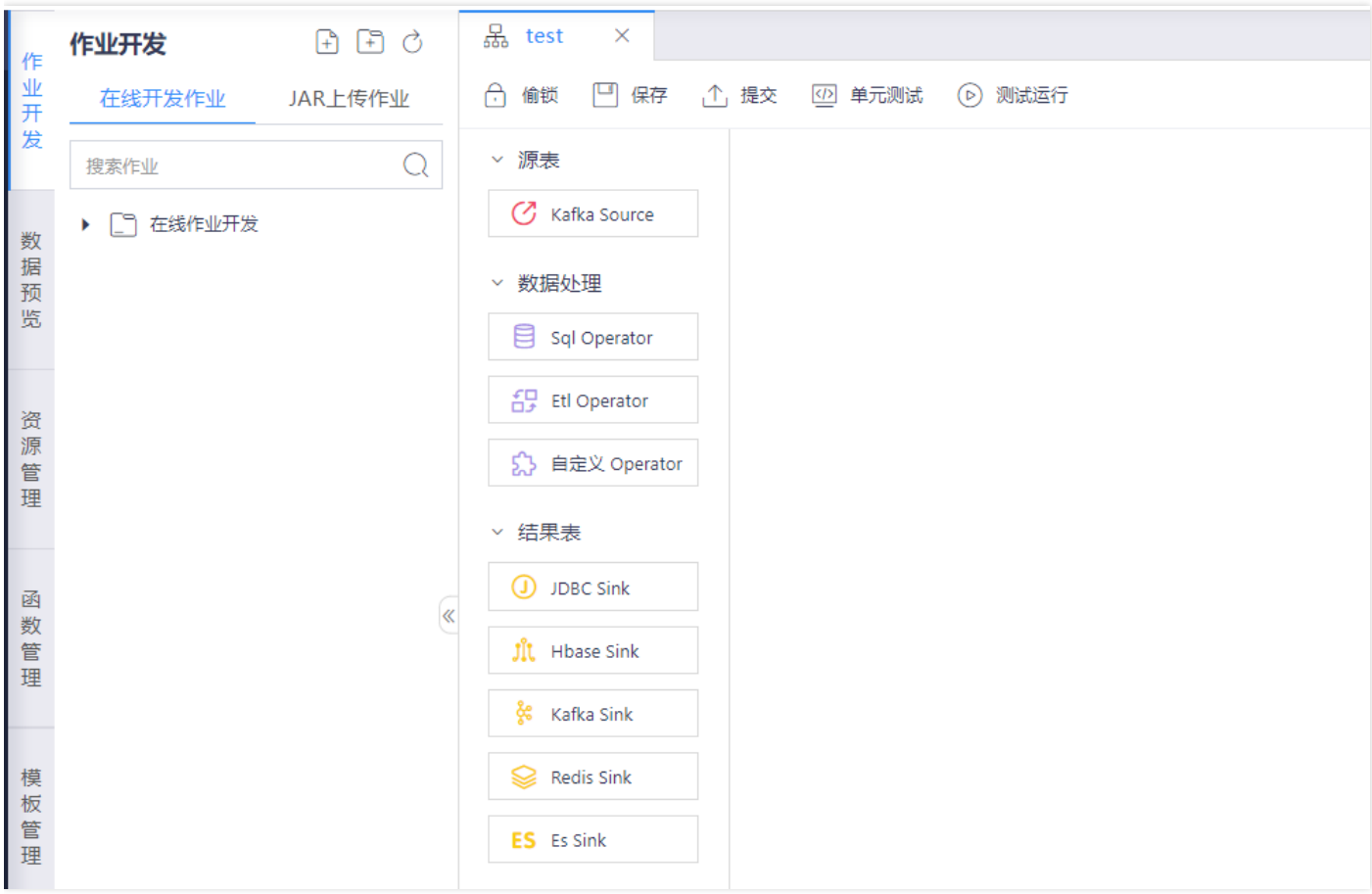
在线开发模式

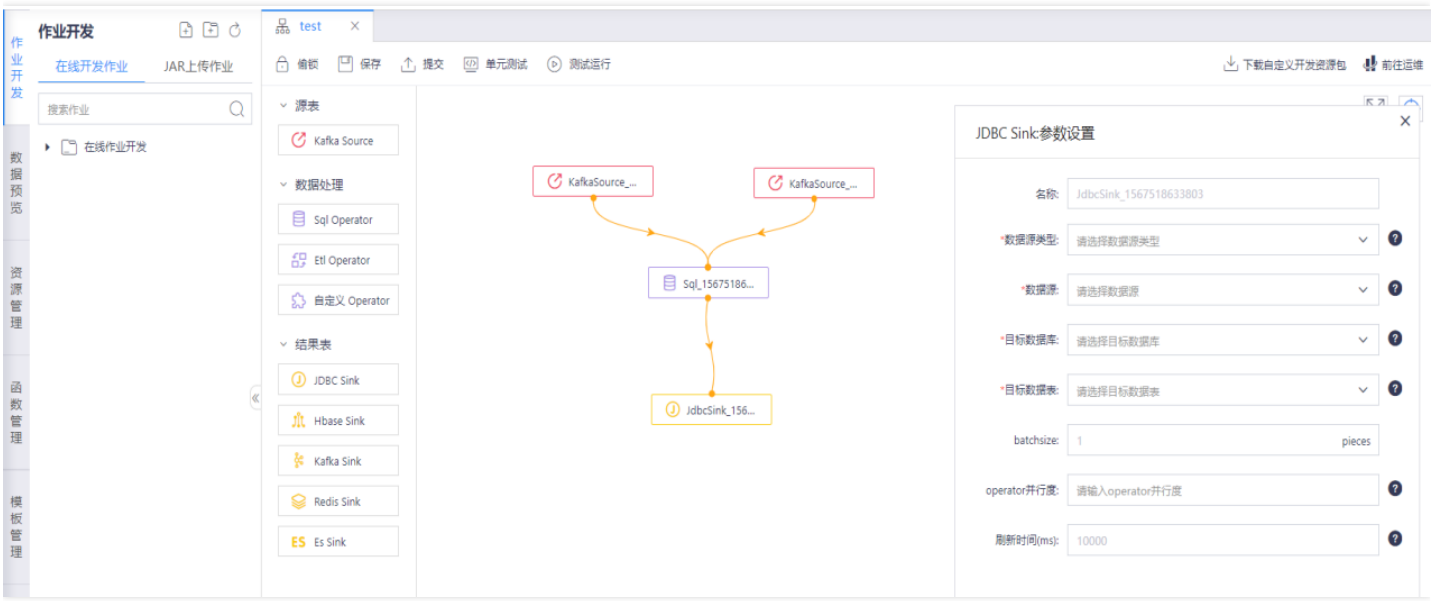
最近更新时间: 2019-11-26 14:47:12

新建在线开发模式的流计算作业，将进入在线开发的IDE界面，中间的“源表”“数据处理”“结果表”等Operator支持拖拽。拖拽后，将自动弹出算子的参数设置栏，完成参数配置，连接各Operator后，即可生成流作业。流计算支持：

- 双流joining（支持2个Kafka数据源）、
- 维表管理（支持Kafka数据源关联MySQL、Oracle等其他数据源 ）、
- JAR包依赖（通过上传的JAR包进行ETL等操作）、
- 单元测试（支持线下上传数据包，进行乱序、延时测试）

等功能，灵活支持各种业务场景。




作业开发
在线开发作业
JAR上传作业
搜索作业
在线作业开发
test
锁
保存
提交
单元测试
测试运行
下载自定义开发资源包
前往运维
源表
Kafka Source
数据处理
Sql Operator
Etl Operator
自定义 Operator
结果表
JDBC Sink
Hbase Sink
Kafka Sink
Redis Sink
Es Sink
KafkaSource_...
KafkaSource_...
Sql_15675186...
JdbcSink_156...
JDBC Sink:参数设置
名称:
JdbcSink_1567518633803
*数据源类型:
请选择数据源类型
*数据源:
请选择数据源
*目标数据库:
请选择目标数据库
*目标数据表:
请选择目标数据表
batchsize:
1
pieces
operator并行度:
请输入operator并行度
刷新时间(ms):
10000

作业参数配置

最近更新时间: 2019-11-26 14:47:12

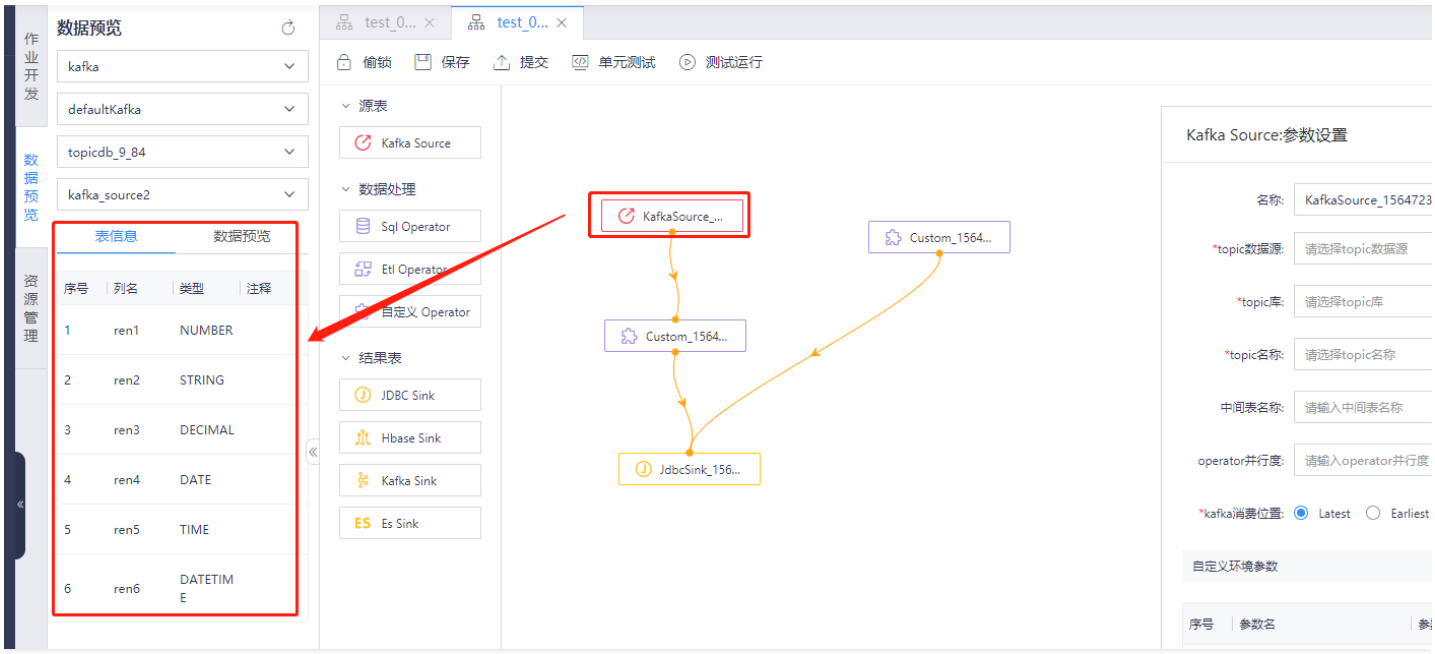
新建作业后，会自动弹出作业参数配置弹窗，可先对流作业的信息进行配置：

- 时间属性：支持Event Time和Processing Time 2种，分别表示流计算应用中，按照什么时间进行数据处理。
- CheckPoint触发间隔：即设置检查点的时间间隔。
- 默认并行数：即流作业的并行数。
- 自定义环境参数：非必须，可配置Flink引擎的其他环境参数，缓存属性等。

数据预览

最近更新时间: 2019-11-11 07:47:11

数据预览功能可支持提前对source、sink的数据结构和数据进行预览，辅助开发。

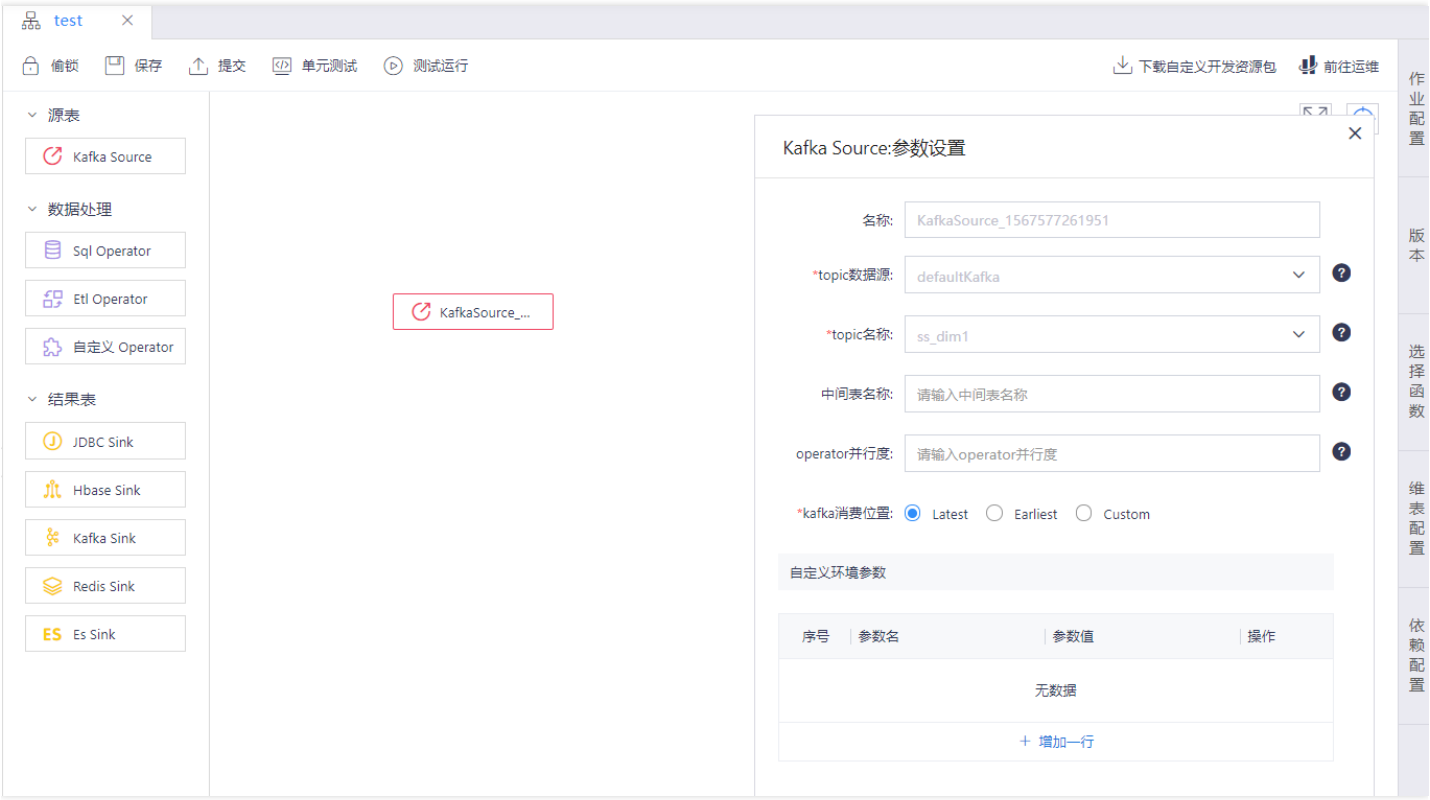


添加数据源

最近更新时间: 2019-11-26 14:47:12

在IDE界面拖拽数据源插件“Kafka source”，双击拖拽的Kafka source弹出参数设置框（首次拖拽插件时，自动弹出），配置参数后，点击右上角的“X”进行保存。配置参数说明如下：

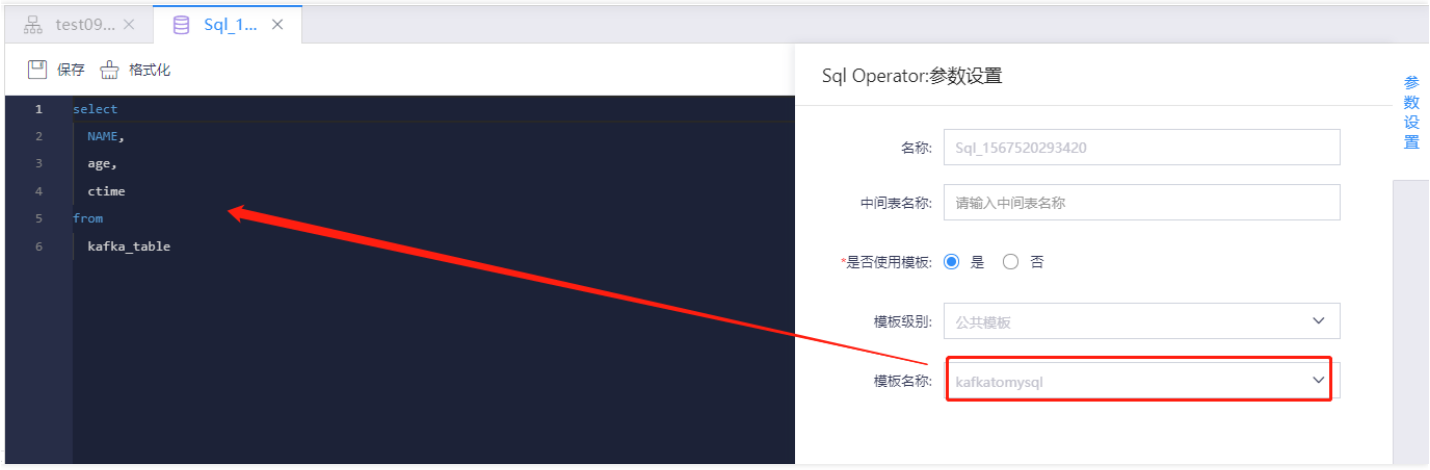
- 名称：即插件名称，可修改
- 数据源：可选择有项目下有权限的数据源
- topic名称：可选择已选数据源下的topic
- 中间表名称：非必填，为将Kafka中的数据进行结构化后注册表的名称；
- Operator并发度：非必填，如不指定将按照作业默认并行数执行；
- Kafka消费位置：支持Latest（最近时间）、Earliest（最早时间）、Custom（自定义选择开始时间）三种方式，默认Latest time执行。



SQL Operator

最近更新时间: 2019-11-11 07:47:04

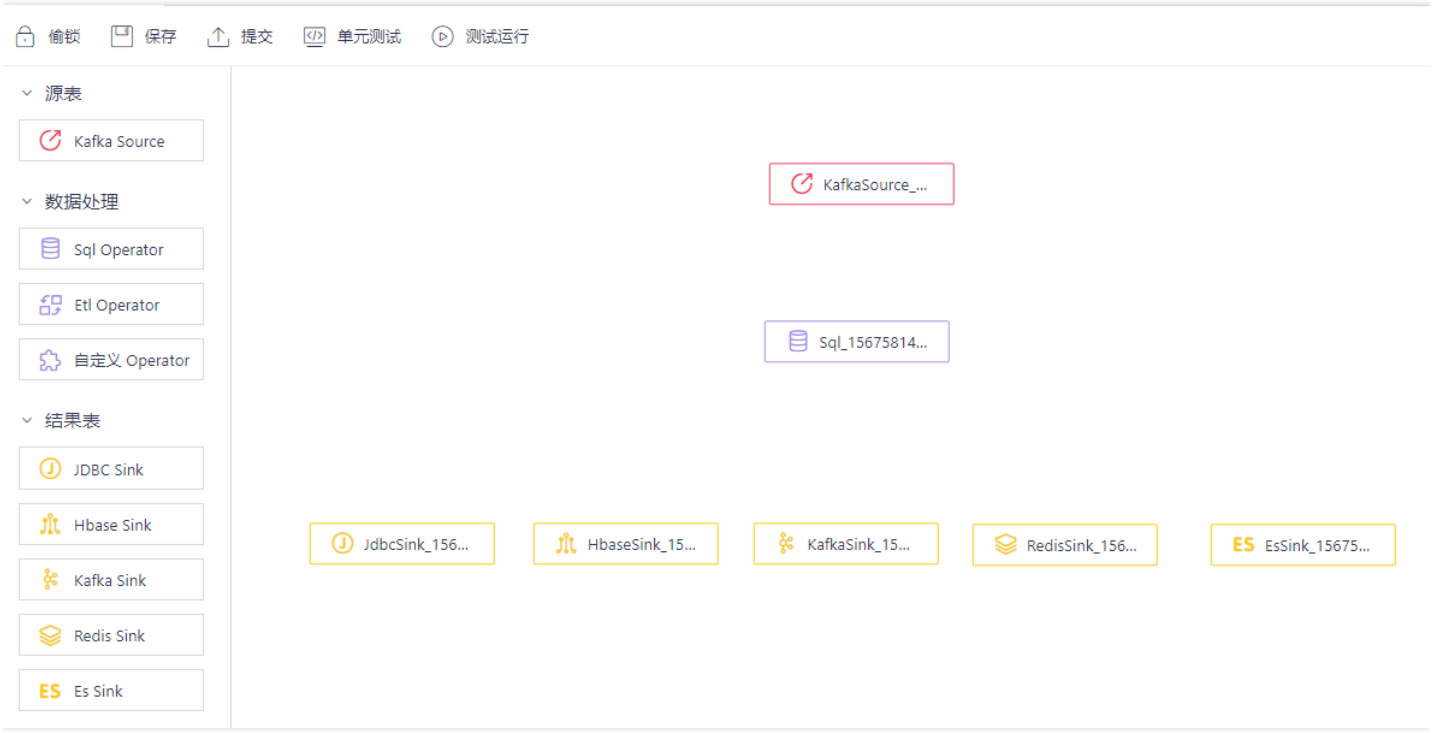
在IDE界面拖拽数据源插件“SQL Opertaor”，编写具体的流计算业务逻辑，SQL Operator不能独立存在，只能在source和sink之间。 双击拖拽的“SQL Opertaor”，即可通过编写Flink SQL进行数据处理，SQL编译框默认提供编写规范，并支持添加预制的模板（具体详见模板管理）。



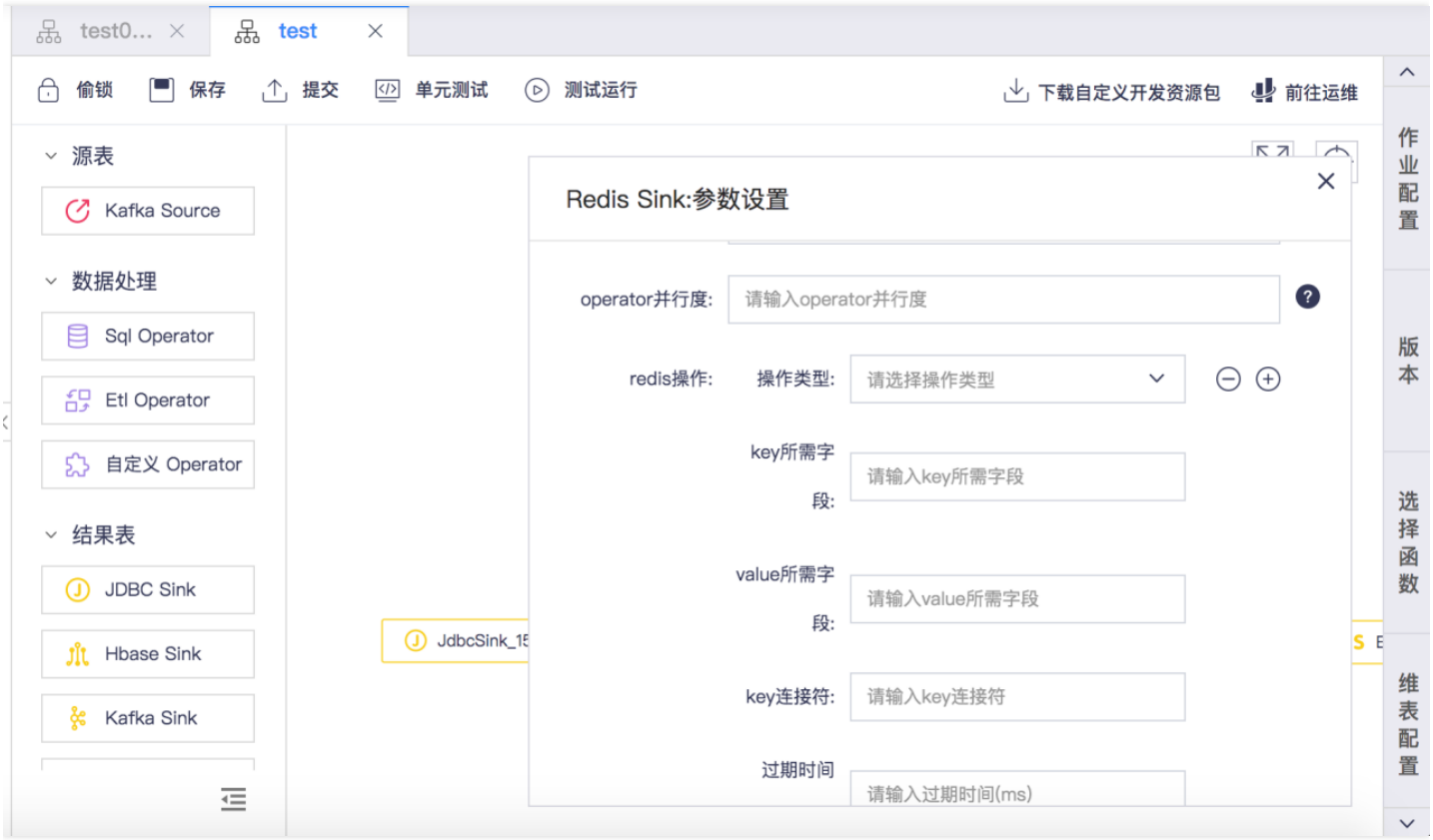
添加目标表

最近更新时间: 2021-09-15 15:59:08

拖拽目标表算子，即可进行添加目标表的操作，流计算服务支持将数据输出至：JDBC（含：MySQL、Oracle二种数据源）、HBase、Kafka、Redis、Elasticsearch七种数据源。



JDBC sink、HBase sink、ES sink中都有数据写入批次概念： 数据按照批次写入，有两个条件限制：批次大小batchsize和刷新时间flushIntervalms，只要满足其中一个条件，就会触发写数据操作。batchsize表示这一批次的大小，也就是有多少条数，刷新时间表示这一批次等待多长时间写入。



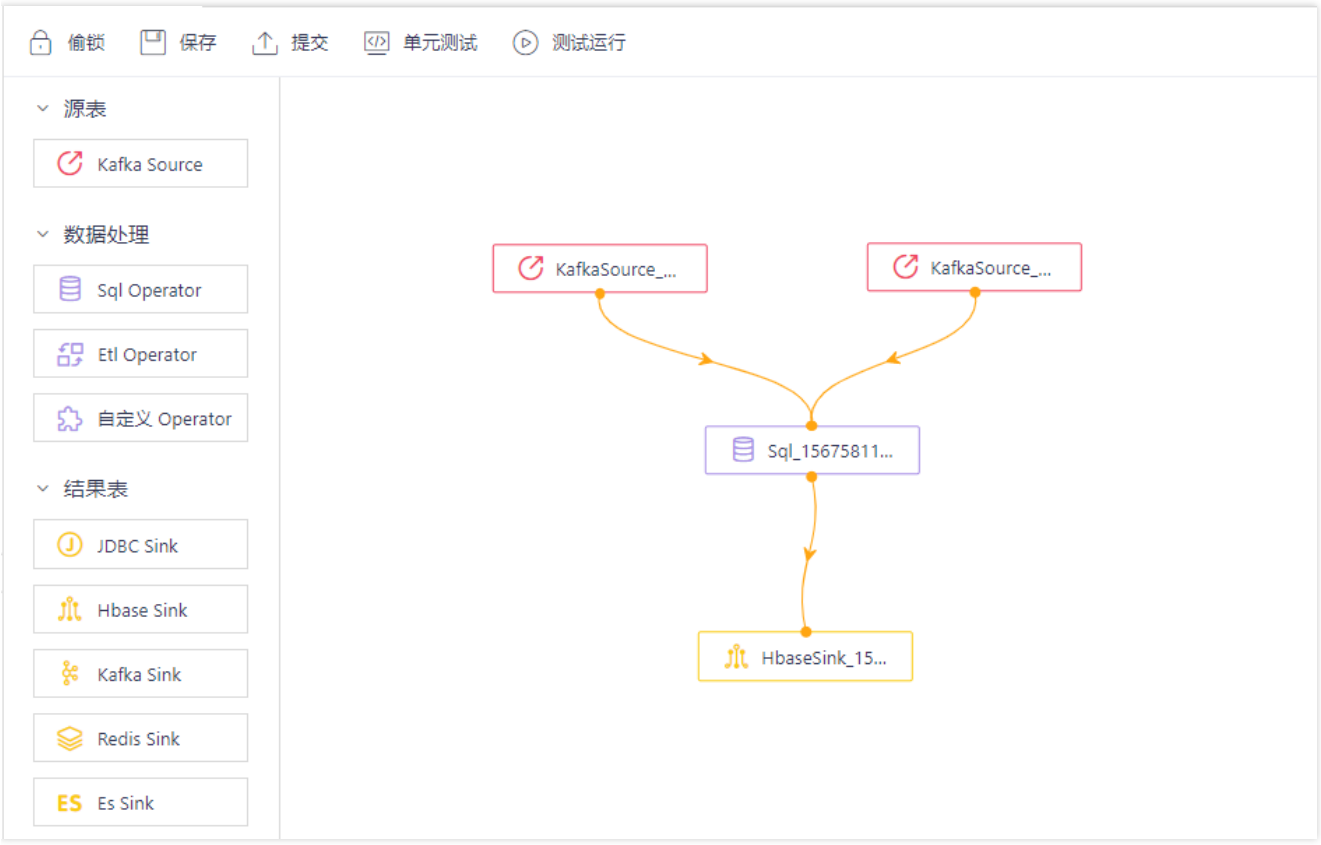
用户配置Redis sink时，支持多种操作，incr、incrByFloat、set、sadd、zadd、rpush、lpush、hset。 key所需字段: key是从上游表中取的某个或多个字段对应的值，如果输入多个字段时用逗号分隔。 value所需字段：value的值从上游表中取的某个字段对应的值。 key连接符: 拼接key时候的连接符。

连接source、Operator、sink插件

最近更新时间: 2019-11-11 07:47:04

将鼠标悬停在算子上，可浮现连线点，可选择算子的连接顺序，生成作业流。 在IDE界面通过拖拽方式将源表Kafka source ，SQL Operator，Kafka sink 进行连接，多个作业的时候也可灵活连接。

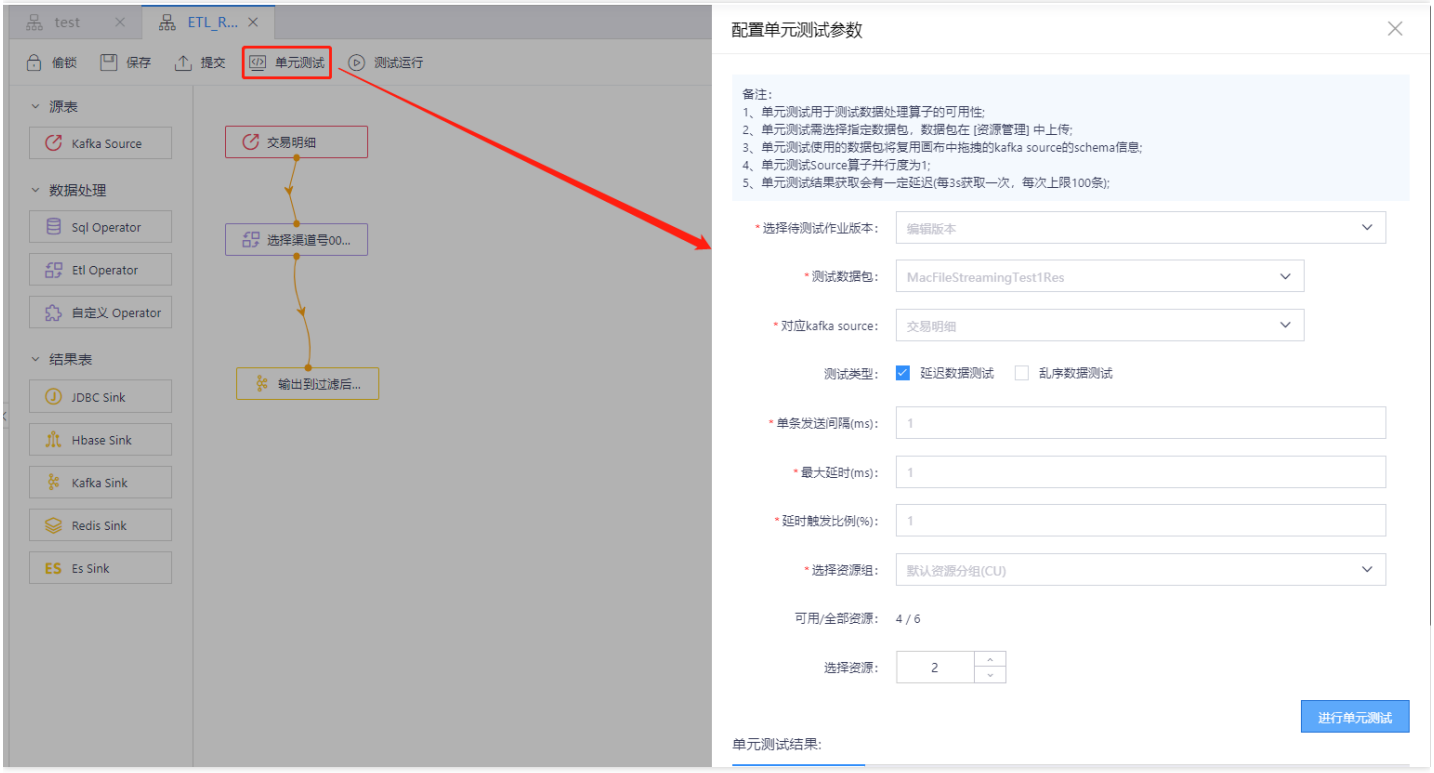
流计算支持同时对2个Kafka source进行关联处理（双流joining）。



单元测试

最近更新时间: 2019-11-11 07:47:04

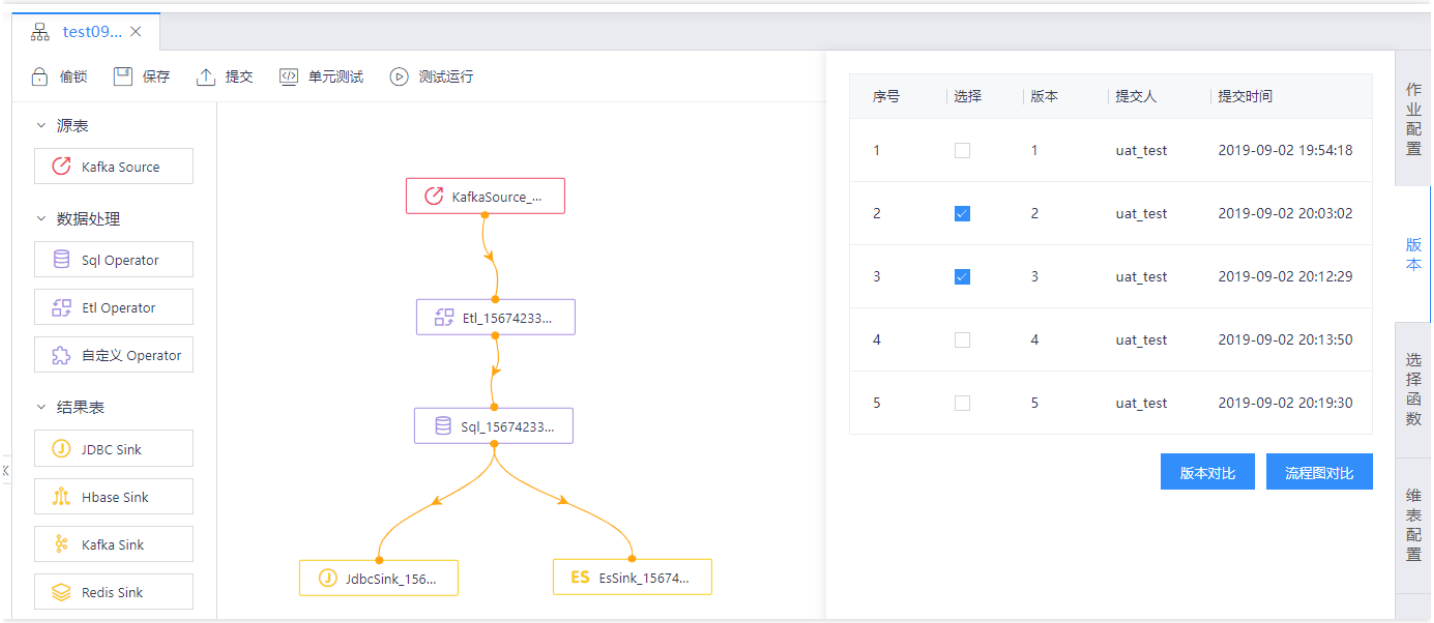
单元测试可以快速进行作业测试，支持乱序测试、延迟测试2种方式。 读取线下上传的测试数据包中的数据，代替从source中读取数据，并且可以复用画板中source的Schema信息，将测试结果数据输出到开发界面，代替了数据输出到sink。 1、单元测试用于测试数据处理算子的可用性; 2、单元测试需选择指定数据包，数据包在[资源管理] 中上传; 3、单元测试使用的数据包将复用画布中拖拽的Kafka source的Schema信息; 4、单元测试source算子并行度为1; 5、单元测试结果获取会有一定延迟(每3s获取一次，每次上限100条)。



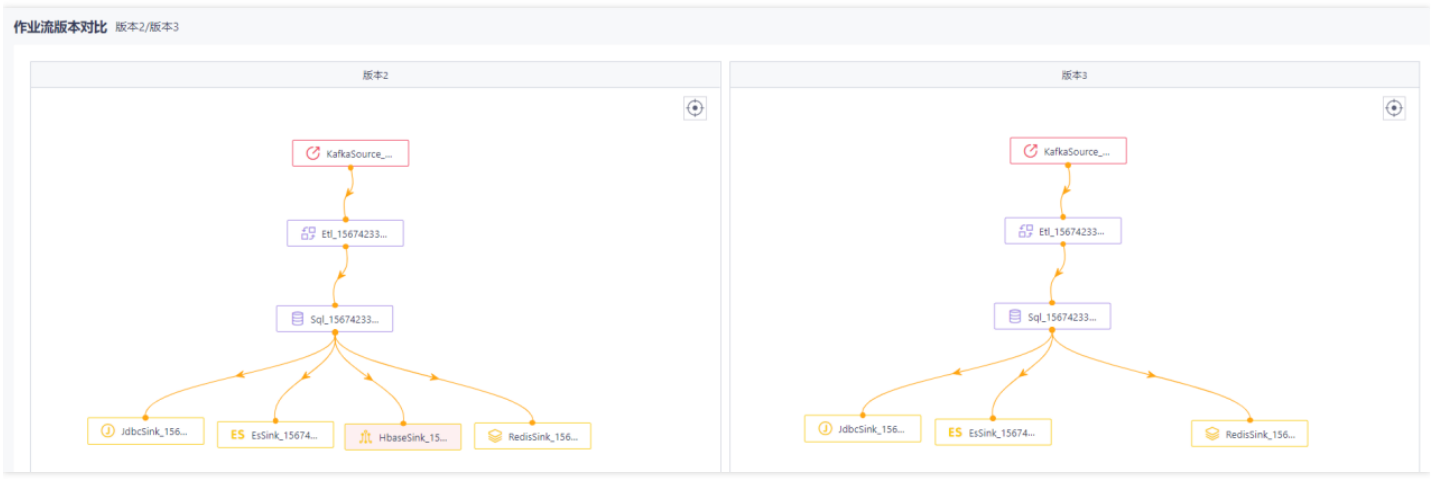
版本管理

最近更新时间: 2019-11-11 07:47:04

支持生成多个版本的流作业（点击“提交”按钮生成新版本），点击“版本管理”，可以查看各个版本的信息。



当多版本时，通过点击“流程图对比”或“版本对比”，可对任意2个版本的流程图和参数进行对比，内容不一致的部分会被标注出来：


作业流版本对比 版本2/版本3
版本2
KafkaSource_...
Etl_15674233...
Sql_15674233...
JdbcSink_156...
ES EsSink_15674...
HbaseSink_15...
RedisSink_156...
版本3
KafkaSource_...
Etl_15674233...
Sql_15674233...
JdbcSink_156...
ES EsSink_15674...
RedisSink_156...


作业版本对比--- 版本2 / 版本3
流计算内容对比
2
3
},
{
"jobName":"HbaseSink_1567423474810",
"layout":{"x":223,"y":301},
"jobId":1909021924825,
"jobType":"HbaseSink",
"params":{
"commitType":"put",
"columnFamily":"cf",
"commitColumns":"",
"dsId":"239",
"parallelism":"",
"rowkey":"name",
"batchSize":"1",
"db":"ss_hbase_tx",
"table":"ss_test_create_hbase",
"flushIntervalMs":"10000"
}
},
{
"jobName":"JdbcSink_1567423473468",
"layout":{"x":-86,"y":294},
"jobId":1909021924532,
"jobType":"JdbcSink",
"params":{
"dsId":"215",

自定义开发说明

自定义开发说明

最近更新时间: 2019-11-11 07:46:40

自定义开发相关功能包括ETL、Custom Operator、UDF、维表等，提供JAR包上传和在线代码编写两种实现方式。下面将为每一种功能提供详细说明和并附注案例。

ETL Operator

ETL Operator

最近更新时间: 2019-11-11 07:45:53

ETL功能以算子形式，将Stream流暴露给用户，进行细力度的操作。系统提供JAR/在线开发两种实现方式。

JAR开发

最近更新时间: 2019-11-26 14:47:12

- 代码编写 ETL功能需要继承基类ETLFunction，并重写ETL和tableSchema两个方法。

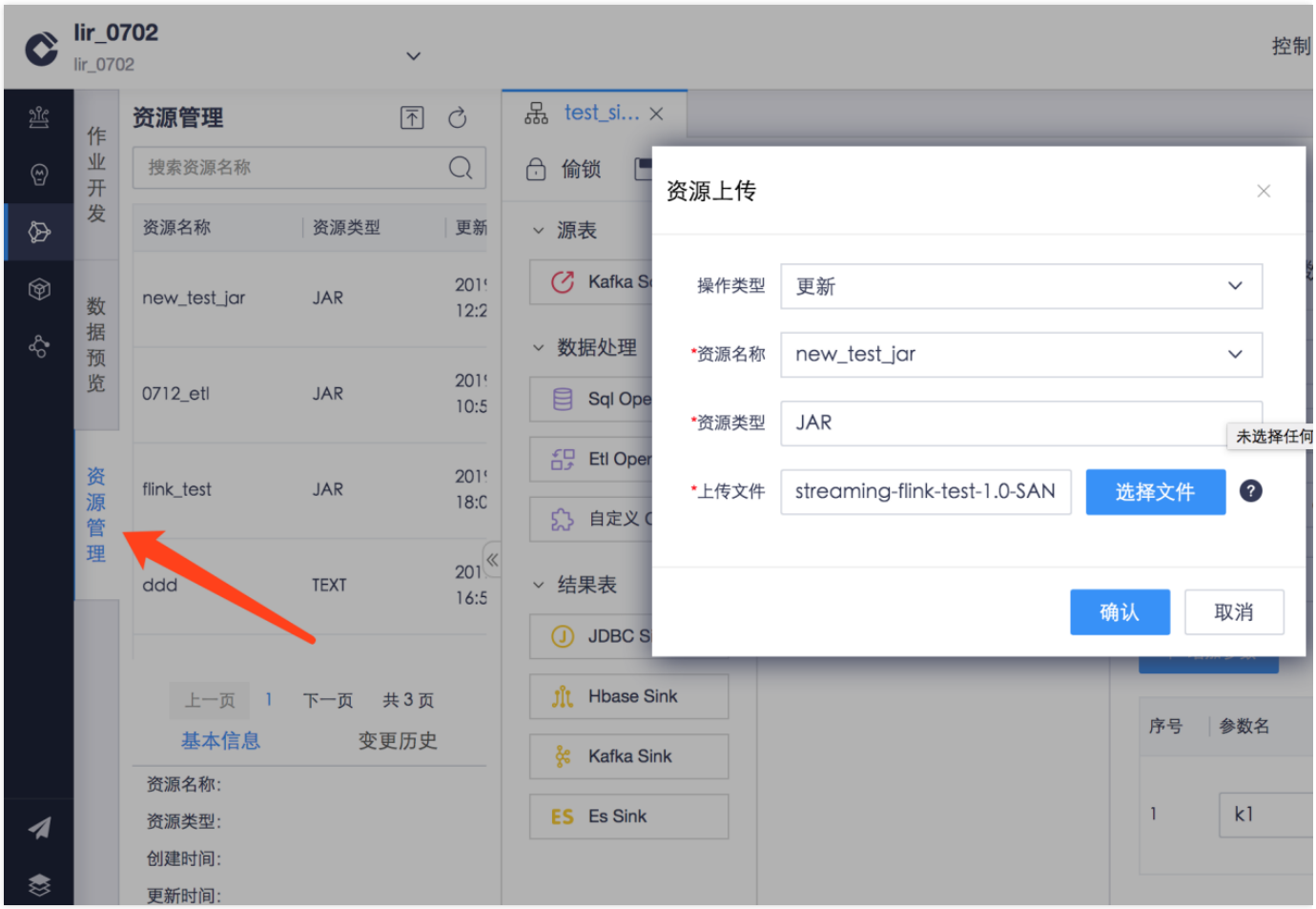
```
trait ETLFunction extends Serializable {
/**
* 把row暴露给用户，进行自定义操作
*
* @param row 任何文本
*/
def ETL(row: Row): Seq[Array[Any]]

/**
* 定义ETL输出的Schema
* 使用JSONRowSchemaConverter.convert(tableScheme())解析
*/
def tableScheme(): String

}
```

举例ETLTestSimple为一个ETL简单实现： 1. params为页面设置的自定义参数，以map形式输入，此处只简单输出。 2. with Logging，代码中可直接使用log实例，日志会打印在TaskManager的log信息中。 3. ETL函数，重载父类函数。每一条流消息调用一次本方法。本例中将第一列取出，并拼接一个固定列值。实际使用时以具体业务逻辑替换。 4. tableSchema函数，重载父类函数。因处理逻辑不确定，所以返回Schema需算子自己定义。以标准JSON Schema返回。 注意：必须要和ETL函数的返回值对应上，不然后续转换过程和报错。

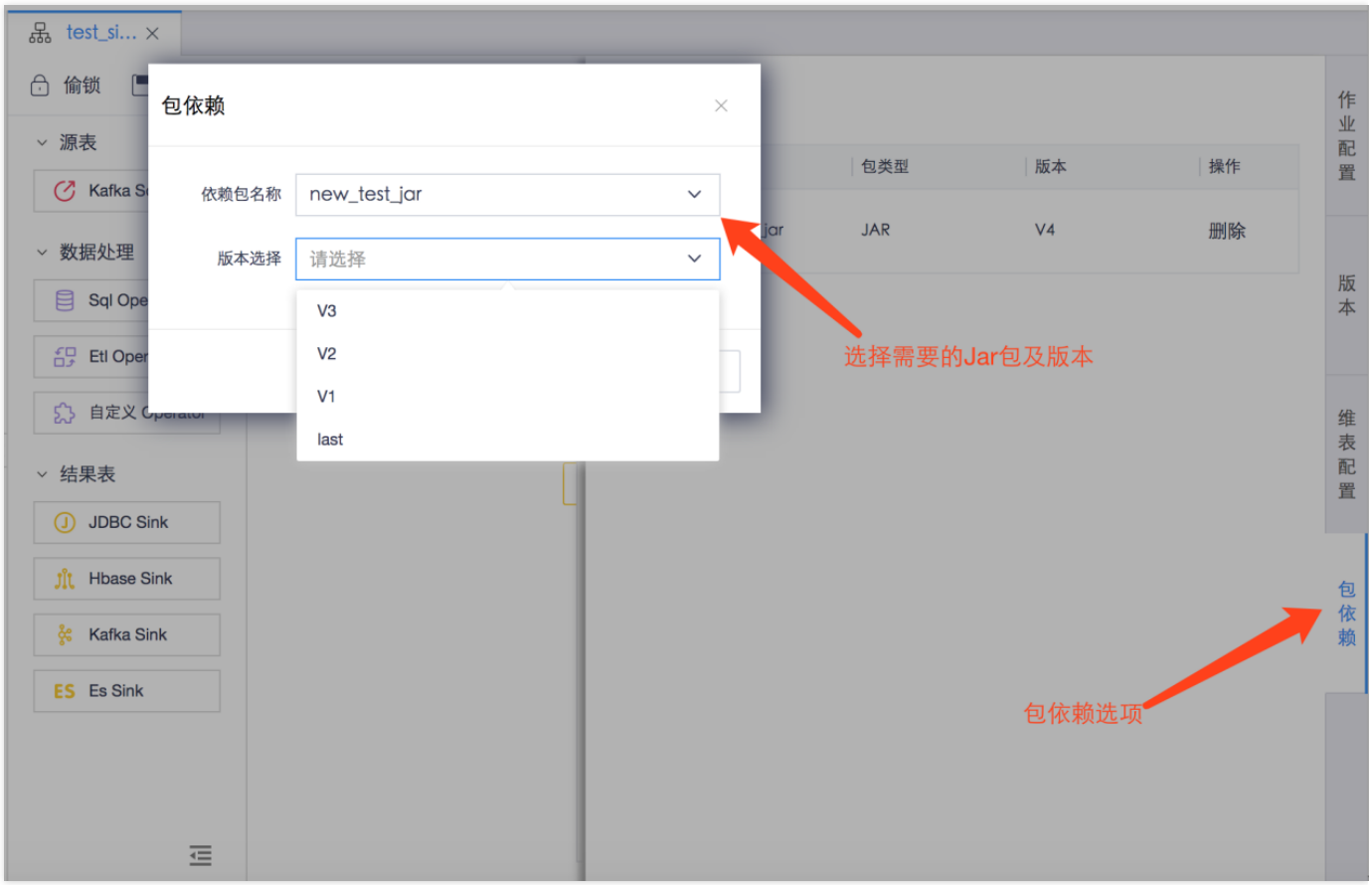
```
class ETLTestSimple (dac: AppConfig, params: Map[String, Any]) extends ETLFunction with Logging {
params.foreach(kv => {
log.info(kv._1 + " =>"+ kv._2)
})
override def ETL(row: Row): Seq[Array[Any]] = {
log.info("in: "+ row.toString)
val rESult: Array[Any] = Array(
row.getField(0),"fieldAddInETL"
)
log.info("out:"+ rESult.mkString(","))
Seq(rESult)
}

/**
* 定义ETL输出的Schema
*/
override def tableScheme(): String = {
val Schema = "{'type':'object','propertiES':{'id':{'type':'integer'},'name':{'type':'string'}}}"
Schema
}
}
```

- 代码打包 mvn clean package –DskipTests –Dcheckstyle.skip=true
- 页面配置

1. 在资源管理功能中将打包后的【streaming-flink-test-1.0-SANPSHOT.jar】上传。



2. 在作业开发界面选择依赖的JAR。



3. 拖拽算子生成graph，并设置ETL Operator参数。其中类名称需为全路径，在执行时从依赖JAR包中反射注入。

在线开发

最近更新时间: 2019-11-11 08:00:19

- **代码编写** 在线代码编写功能在原ETL算子中，增加了“在线开发”选项，并提供Java/Scala两种语法支持。默认会展示最基本的代码框架，如需业务相关模板可在“是否使用模板”功能中选择。

双击ETL算子后进入编辑页面。按上图进行操作，其中需注意点如下：

1. 模板需预先在左侧“模板管理”功能中设置，且公共模板整个租户可见。
2. 模板具体参见《Flink自定义开发模板.doc》，只需编写“模板内容”部分，系统会自动填充成“补全后内容”运行。
3. “添加参数”输入的参数于ETL功能中使用，通过params: Java.util.Map[String, Any]直接获取。
4. 如需在ETL中调用维表，请参见《维表与ETL结合方式》。

Custom Operator

最近更新时间: 2019-11-26 14:47:12

Custom Operator功能提供一个自定义的全新算子，可以充当ETL，也可以充当source或者sink。需要自己维护TypeInformation并通过flatMap函数实现业务逻辑，Custom Operator只支持JAR开发模式。

- 代码编写 ETL功能需要继承基类tableOperateProcESsor，并重写innerbuild函数。举例CustomTestSimple为一个Custom Operator简单实现：
- 构造器中定义了四个参数： name：本算子的name，通常设置为在jobGraph中显示名称。 childs：子算子集合 configs: Map格式参数集合 sm: StreamingMate可获取环境变量、维表等全局参数
- with Logging，代码中可直接使用log实例，日志会打印在TaskManager的log信息中。
- innerBuild函数，重载父类函数。将dataStream直接暴露给用户，本例中不做处理直接转给下一个算子。实际使用时以具体业务逻辑替换。
- dataStream.dataType.asInstanceOf[RowTypeInfo]获取输入流的Schema 注意：必须要隐式声明implicit val tpe: TypeInformation[Row]，供下游算子使用

```
class CustomTestSimple(name: String, childs: mutable.MutableList[ProcESsor[DataStream[Row]]], configs: Map[Any, Any], sm: StreamingMate)
extends tableOperateProcESsor(name, childs, configs, sm) {

override def innerBuild(dataStream1: DataStream[Row],dataStream2: DataStream[Row]): DataStream[Row] = {
configs.foreach(kv => {
println(kv._1 + " =>"+ kv._2)
})

val rowType = dataStream1.dataType.asInstanceOf[RowTypeInfo]
implicit val tpe: TypeInformation[Row] = rowType

val finalStream = dataStream1.flatMap[Row](new MapMapper(rowType)).setParallelism(1)
finalStream
}
}
class MapMapper(rowTypeInfo: RowTypeInfo) extends RichFlatMapFunction[Row, Row] {
override def flatMap(row: Row, collector: Collector[Row]): Unit = {
println("CustomTestSimple:"+ row)
collector.collect(row)
}
}
```

- 代码打包 同ETL
- 页面配置 同ETL

自定义UDF
JAR开发

最近更新时间: 2019-11-11 08:00:19

- 代码编写 UDF功能需要继承基类ScalarFunction，并重写eval方法。

```
import org.Apache.Flink.table.functions.ScalarFunction

class UDFTestScala extends ScalarFunction{
def eval(s: String): String = {s}
}
```

- 代码打包

```
mvn clean package -DskipTests -Dcheckstyle.skip=true
```

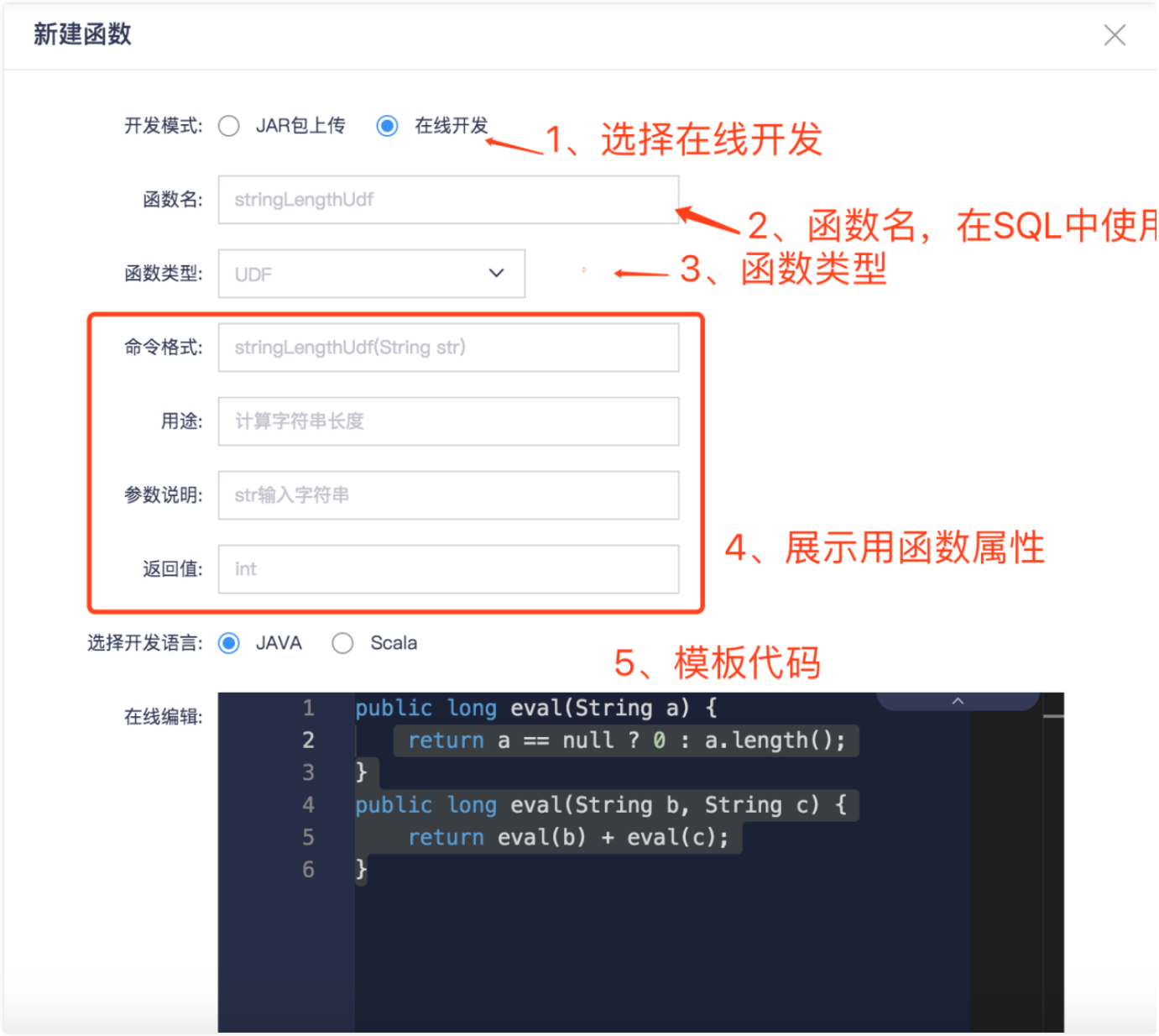
- 页面配置 在资源管理功能中将打包后的streaming-flink-test-1.0-SANPSHOT.jar上传。

在线开发

最近更新时间: 2019-11-11 08:00:19

- 代码编写 在左侧菜单栏添加了“函数管理”功能，支持UDF/UDAF的添加和修改：

双击新增按钮后进入编辑页面。



按上图进行操作，其中需要注意点如下： 1. 函数名为最终register的函数名称，SQL中使用这个名称。 2. 函数属性用以展示，在选择函数时使用。 3. 模板具体参见《Flink自定义开发模板.doc》，只需编写“模板内容”部分，系统会自动填充成“补全后内容”运行。

自定义UDAF JAR开发

最近更新时间: 2019-11-11 08:00:19

UDAF功能采用Flink自身UDF语法，详细说明参见https://ci.Apache.org/projects/Flink/Flink-docs-release-1.7/dev/table/UDFs.html

- 代码编写 UDAF功能需要继承基类AggregateFunction，并重写createAccumulator、getValue、accumulate、getRESultType四个方法（有些可选方法见官网）。

```
import org.Apache.Flink.table.functions.AggregateFunction
import Java.lang.Long
import Java.util.concurrent.atomic.AtomicLong

import org.Apache.Flink.api.common.typeinfo.{TypeInformation, TypES}
class UDAFTestScala/*占位符*/ extends AggregateFunction[Long, AtomicLong] {
//初始化count UDAF的accumulator
override def createAccumulator: AtomicLong = new AtomicLong(0L)

//getValue提供了如何通过存放状态的accumulator计算count UDAF的结果的方法
override def getValue(accumulator: AtomicLong): Long = accumulator.get()

//accumulate提供了如何根据输入的数据更新count UDAF存放状态的accumulator
def accumulate(accumulator: AtomicLong, iValue: Long): Unit = {
accumulator.addAndGet(iValue)
}

override def getRESultType: TypeInformation[Long] = TypES.LONG
}
```

- 代码打包

```
mvn clean package -DskipTests -Dcheckstyle.skip=true
```

- 页面配置 在资源管理功能中将打包后的streaming-flink-test-1.0-SANPSHOT.jar上传。

在线开发

最近更新时间: 2019-11-11 08:04:05

在左侧菜单栏添加了“函数管理”功能，支持UDF/UDAF的添加和修改：

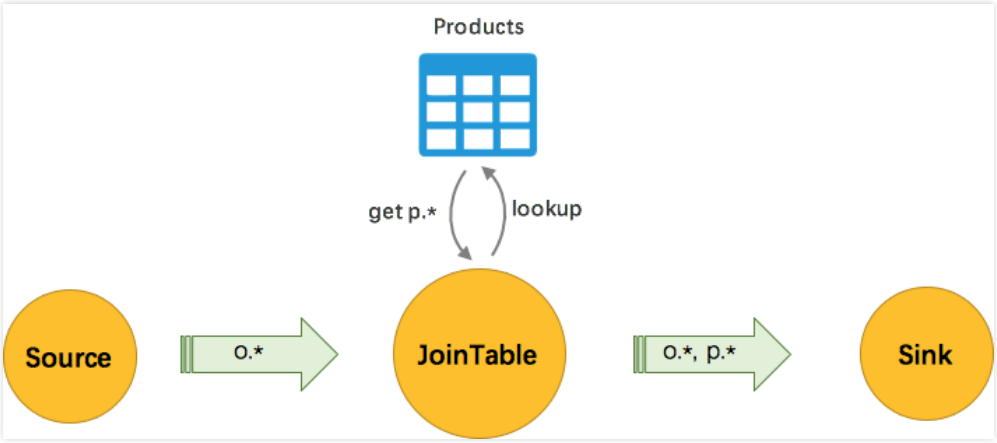
双击新增按钮后进入编辑页面。按上图进行操作，其中需要注意点如下： 1. 函数名为最终register的函数名称，SQL中使用这个名称。 2. 函数属性用以展示，在选择

函数时使用。 3. 模板具体参见《Flink自定义开发模板.doc》，只需编写“模板内容”部分，系统会自动填充成“补全后内容”运行。

维表配置
功能说明

最近更新时间: 2019-11-26 14:47:12

当前Flink版本只支持流式数据源，当流数据需要关联外部数据库（如MySQL、Oracle、Redis等），需要采用维表的形式支持。如下图所示，在ETL操作中，每一条



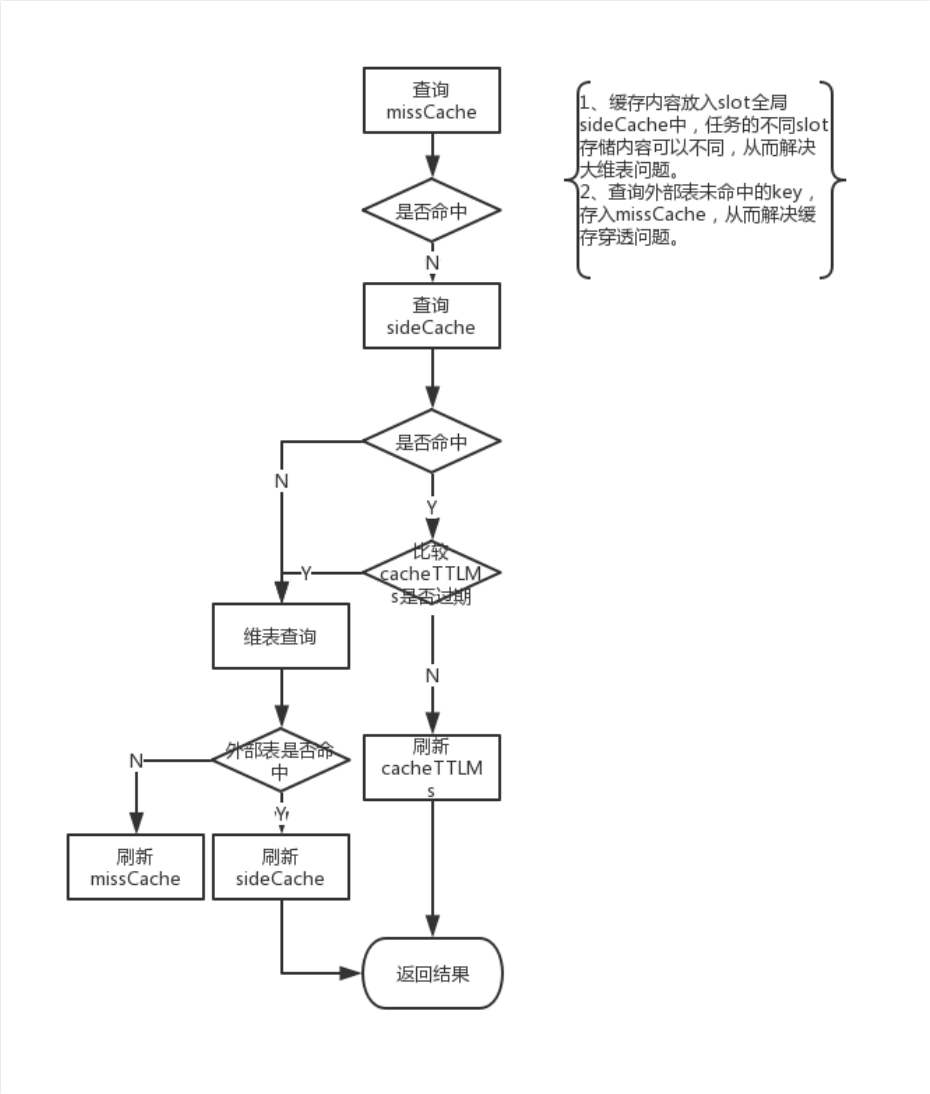
消息会lookup一下外部数据库。 数据库的维表查询请求，有大量相同 key 的重复请求。如何减少重复请求？本地缓存是常用的方案。本方案目前提供两种缓存方案：LRU 和 ALL。 cache = ALL（默认）：全量内存缓存 cacheTTLMs:缓存的过期时间(ms) cache = LRU: LRU内存缓存 cachESize: 缓存的条目数量 cacheTTLMs:缓存的过期时间(ms)

- All缓存 Async 和 LRU-Cache 能极大提高吞吐率并降低数据库的读压力，但是仍然会有大量的 IO 请求存在，尤其是当 miss key（维表中不存在的 key）很多的时候。如果维表数据不大（通常百万级以内），那么其实可以将整个维表缓存到本地。那么 miss key 永远不会去请求数据库。因为本地缓存就是维表的镜像，缓存中不存在那么远程数据库中也不存在。 ALL cache 可以通过 cache='ALL'参数开启，通过cacheTTLMs控制缓存的刷新间隔。会为 updater 节点起一个异步线程去同步缓存。在 Job 刚启动时，会先阻塞主数据流，直到缓存数据加载完毕，保证主数据流流过时缓存就已经 ready。在之后的更新缓存的过程中，不会阻塞主数据流。因为异步更新线程会将维表数据加载到临时缓存中，加载完毕后再与主缓存做原子替换。只有替换操作是加了锁的。 因为几乎没有 IO 操作，所以使用 cache ALL 的维表性能可以非常高。但是由于内存需要能同时容纳下两份维表拷贝，因此需要加大内存的配置。 全量内存缓存模式，用户需要实现一个updater，用以定期全量刷新数据；getKey用以生成关联缓存的key。由框架来管理缓存cacheTTLMs到期和updater调用。 Datasource.select()为具体的查询操作。

```
class MySQLUpdater(config: Map[String, Any])
extends AppConfigUpdater(config) with Logging {
val MySQLOptions = Map("url"-> config("url").toString,
"drivername"-> config("drivername").toString,
"username"-> config("username").toString,
"password"-> config("password").toString)

val dbKey = MySQLOptions.get("url").toString
val SQL= config("SQL").toString
val rowkeyFields = config("rowkeyFields").toString.split(",")

Datasource.initIfNeeded(dbKey, MySQLOptions)

override def update(): Map[String, Any] = {
Datasource.select(dbKey, SQL, rowkeyFields)
}

override def getKey(params: Map[String, Any]): String = {
Datasource.getKey(params)
}
}

object MySQLUpdater {
val key = "MySQLUpdater"
}
```

- LRU缓存 通过 cache='LRU'参数可以开启 LRU 缓存优化，会为每个slot节点创建一个 LRU 本地缓存Map。当每个数据进来的时候，先去缓存中查询，如果存在则直接关联输出，减少了一次 IO 请求。如果不存在，再发起数据库查询请求（异步或同步方式，本系统默认全部采用异步方式处理），请求返回的结果会先存入缓存中以备下次查询。 为了防止缓存无限制增长，所以使用的是 LRU 缓存，并且可以通过 cachESize 调整缓存的大小。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 是作用于每条缓存数据上的，也就是某条缓存数据在指定 timeout 时间内没有被访问，则会从缓存中移除。 在使用 LRU 缓存时，如果存在大量的 invalid key ，或者数据库中不存在的 key。由于命中不了缓存，导致缓存的收益较低，仍然会有大量请求打到数据库。因此我们将未命中的 key 也加进了缓存，提高了未命中 key 和 invalid key 情况下的缓存命中率。 LRU内存缓存，用户需要实现一个updater，用以单条刷新数据，每一次查询都会尝试调用updater方法。由框架来管理缓存cacheTTLMs到期，查询是否命中及updater调用。 Datasource.select()为具体的查询操作。

```
class MySQLQueryer(config: Map[String, Any])
extends AppConfigQueryer(config) with Logging {
val MySQLOptions = Map("url"-> config("url").toString,
"drivername"-> config("drivername").toString,
"username"-> config("username").toString,
"password"-> config("password").toString)

val dbKey = MySQLOptions.get("url").toString
val SQL = config("SQL").toString
val rowkeyFields = config.get("rowkeyFields").getOrElse("").toString.split(",")

Datasource.initIfNeeded(dbKey, MySQLOptions)

override def query(params: collection.Map[String, Any]): Any = {
Datasource.select(dbKey, SQL, params)
}

override def getKey(params: collection.Map[String, Any]): String = {
Datasource.getKey(params)
}
}
object MySQLQueryer {
val key = "MySQLQueryer"
}
```


查询
missCache
是否命中
N
查询
sideCache
是否命中
N
Y
比较
cacheTTLM
s是否过期
Y
维表查询
N
外部表是否命
中
N
Y
刷新
cacheTTLM
s
刷新
missCache
刷新
sideCache
返回结果
1、缓存内容放入slot全局
sideCache中，任务的不同slot
存储内容可以不同，从而解决
大维表问题。
2、查询外部表未命中的key，
存入missCache，从而解决缓
存穿透问题。

- 执行流程图如下：

与ETL结合方式

最近更新时间: 2019-11-11 08:08:56

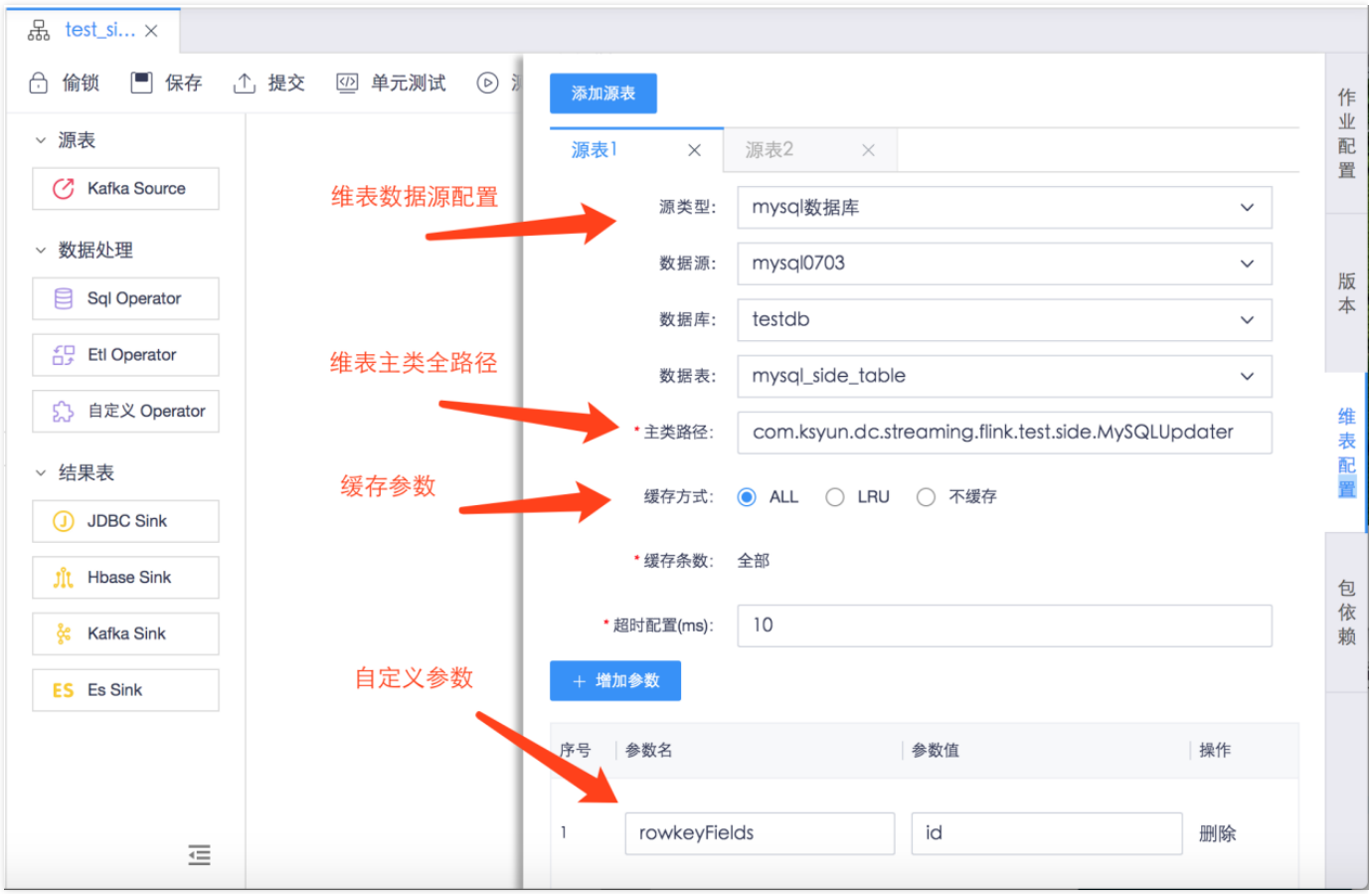
本版本维表功能只支持在ETL算子中使用。本文提供MySQL版的维表实现案例，如需其他版本请用户自定义实现。

- 代码编写 dac.getConfigByKey(MySQLUpdater.key)为从运行时环境获取缓存对象，其中MySQLUpdater.key为对应实现的key。

```
class ETLWithSide (dac: AppConfig, params: Map[String, Any]) extends ETLFunction with Logging {

/**
* 把一个字符串解析成表(多行 多列)
*
* @param row 任何文本
*/
override def ETL(row: Row): Seq[Array[Any]] = {
//ALL 类型返回ALLCACHE
val cache1: Cache = dac.getConfigByKey(MySQLUpdater.key).get.asInstanceOf[AllCache]
//LRU和NONE 类型返回LRUCache
val cache2: Cache = dac.getConfigByKey(MySQLQueryer.key).get.asInstanceOf[LRUCache]

val result: Array[Any] = Array(
row.getField(0),
cache1.get(Map("id"->row.getField(0))).getOrElse(Map()).asInstanceOf[Map[String,Any]].getOrElse("name",""),
cache2.get(Map("id"->row.getField(0))).getOrElse(Map()).asInstanceOf[Map[String,Any]].getOrElse("name","")
)
println("ETLWithSide:"+ result.mkString(","))
Seq(result)
}

/**
* 定义ETL输出的Schema
*/
override def tableScheme(): String = {
val Schema = "{'type':'object','properties':{'id':{'type':'integer'},'nameFromAllCache':{'type':'string'},'nameFromLRUCache':{'type':'string'}}}"
Schema
}
}
```

- 代码打包 同ETL

- 页面配置 其中数据源非必选参数，如使用外部数据源，可完全由自定义参数配置数据库连接。维表可设置多个。



如选择内部数据源，对应的参数对照表见【维表参数说明】

维表参数说明

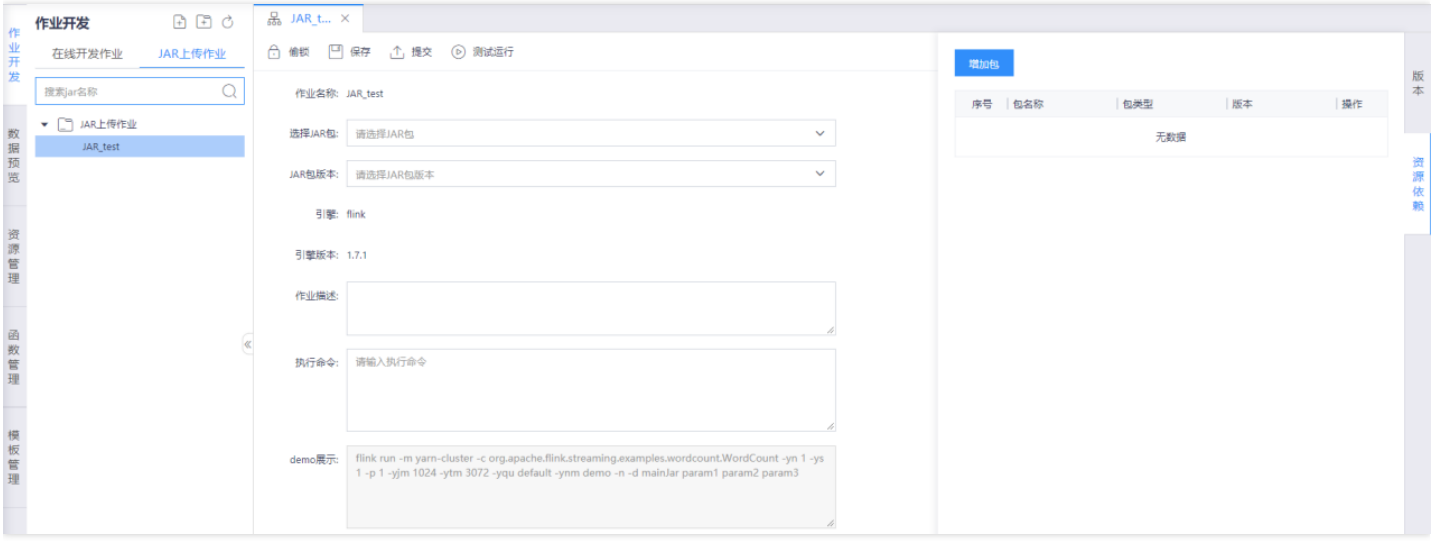
最近更新时间: 2019-10-28 01:42:05

类型	值	说明	demo
公用参数	type	数据库类型	MySQL
	cachetype	缓存类型	ALL/LRU/NONE
	cacheTimeout	缓存超时时间	1000S
	cachESize	缓存大小（LRU时有效）	1000
	dsId	流实例ID	
MySQL	drivername	数据库驱动类型	com.MySQL.JDBC.Driver
	url	数据库地址	JDBC:MySQL://127.0.0.1:3306/Flink
	db	库名	testdb
	tablename	表名	MySQL_side_table
	username	用户名	root
	password	密码	1234
Redis	masterNode	主节点	10.77.0.6:6379
	connectType	连接类型	masterSlave
	password	密码	123456
	operations	操作描述	[{“keyFields”:”ren1,ren2”,”opType”:”set”,”expire”:””,”scoreField”:””,”valueFields”:”ren1”,”keyDelimiter”:”|”}]

JAR包上传模式

最近更新时间: 2019-11-11 08:08:56

JAR包上传模式支持线下上传JAR包进行流作业处理，JAR包上传模式支持Flink和Spark Streaming两种引擎。

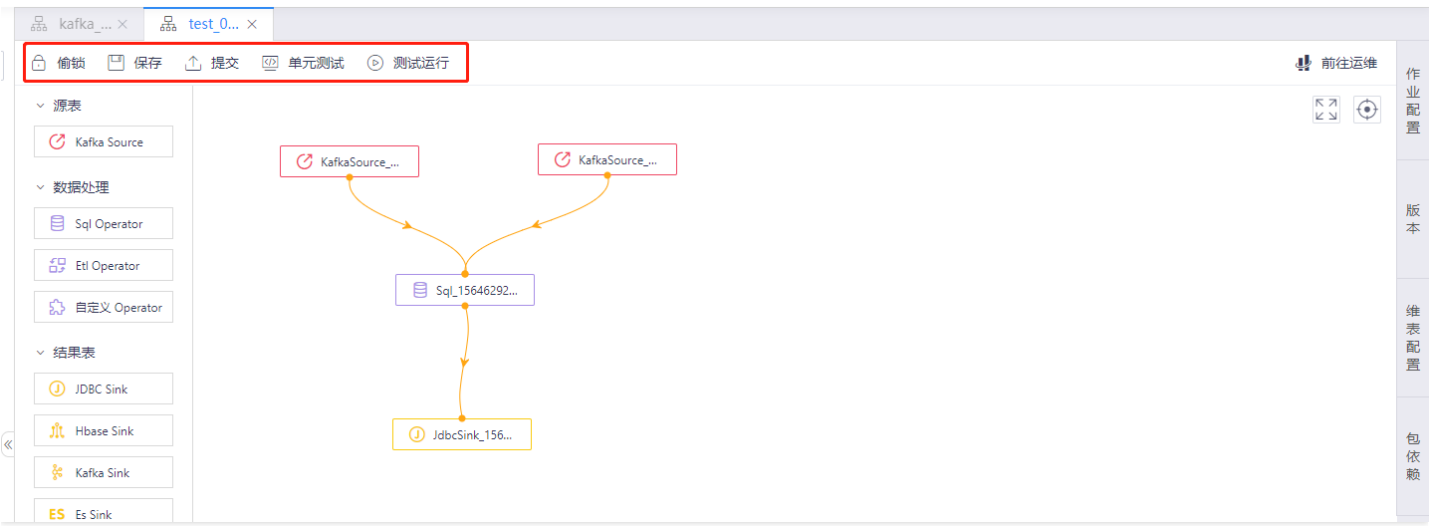


测试及运行

发布测试

最近更新时间: 2019-11-11 08:24:36

完成作业开发配置后，点击“保存”按钮保存当前任务，点击“提交”按钮生成新版本的作业，点击“运行测试”将作业发布到“运维中心”进行测试运行。

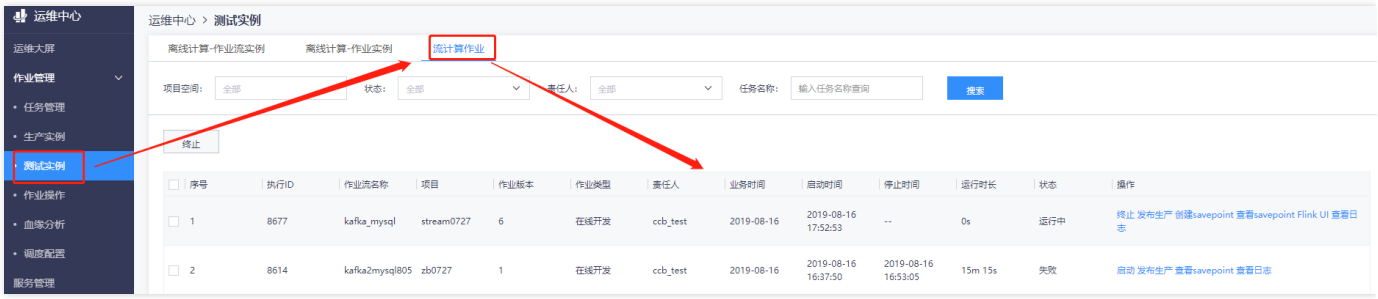


测试运行

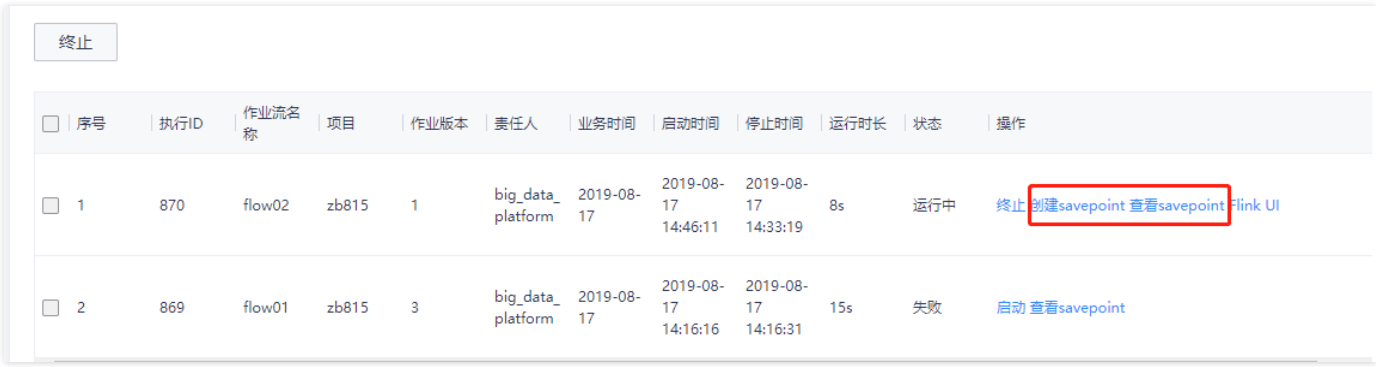
最近更新时间: 2019-11-11 08:24:36

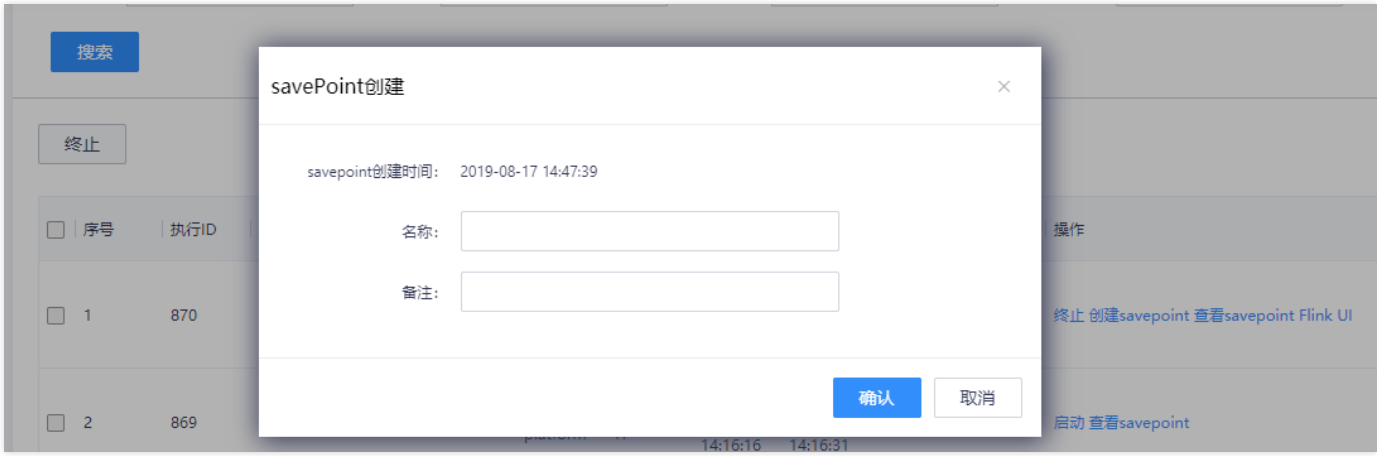
提交作业后，会在“运维中心-测试实例-流计算作业”生成一条流作业数据，可进行：【启动】【终止】【创建SavePoint】【查看SavePoint】【查看Flink UI】等操作。

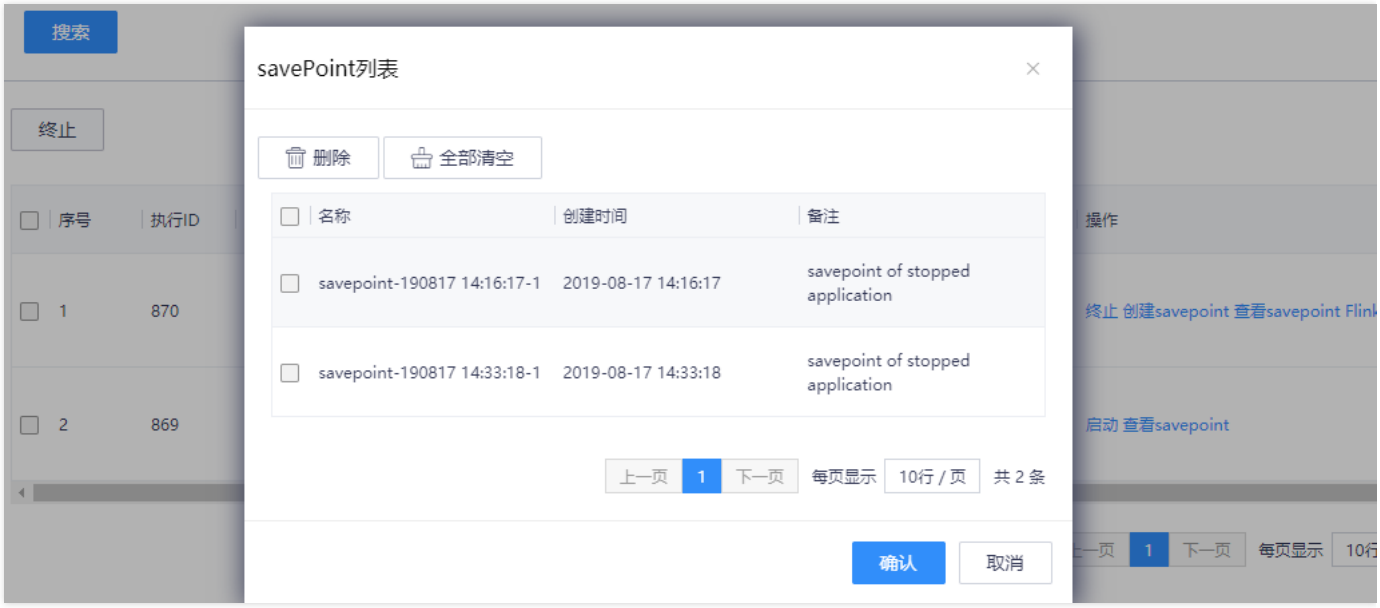
- 【启动】：点击启动，并选择启动方式（可选择立即启动或SavePoint启动）、配置作 业参数后，可启动作业。
- 【终止】：点击终止，可停止作业；



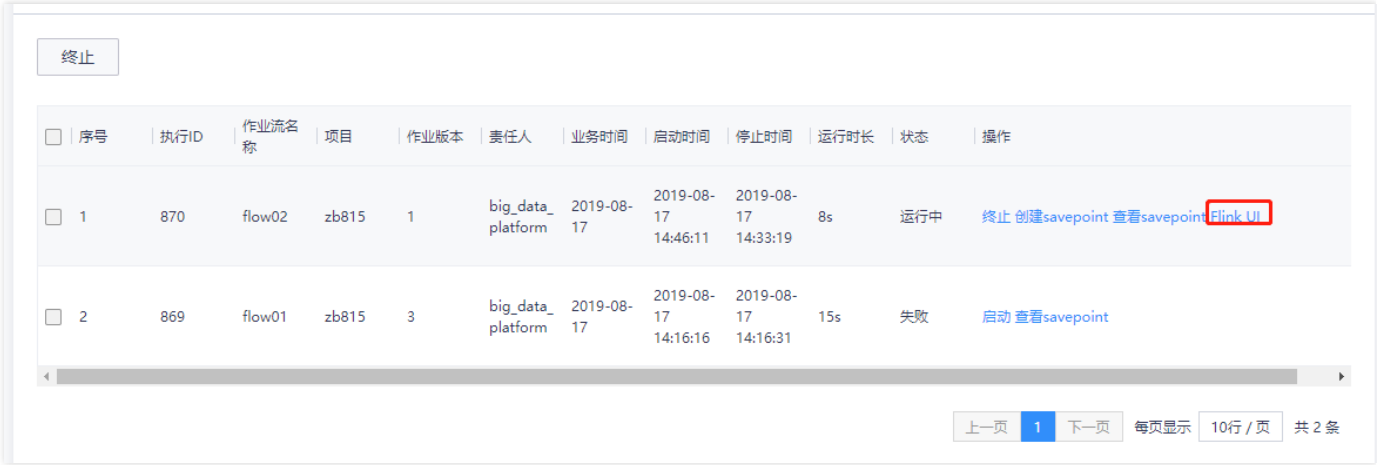
- 【创建SavePoint】：创建后，再次启动作业时，可选择从SavePoint标识的点启动作 业；
- 【查看SavePoint】：可以查看所有的SavePoint列表







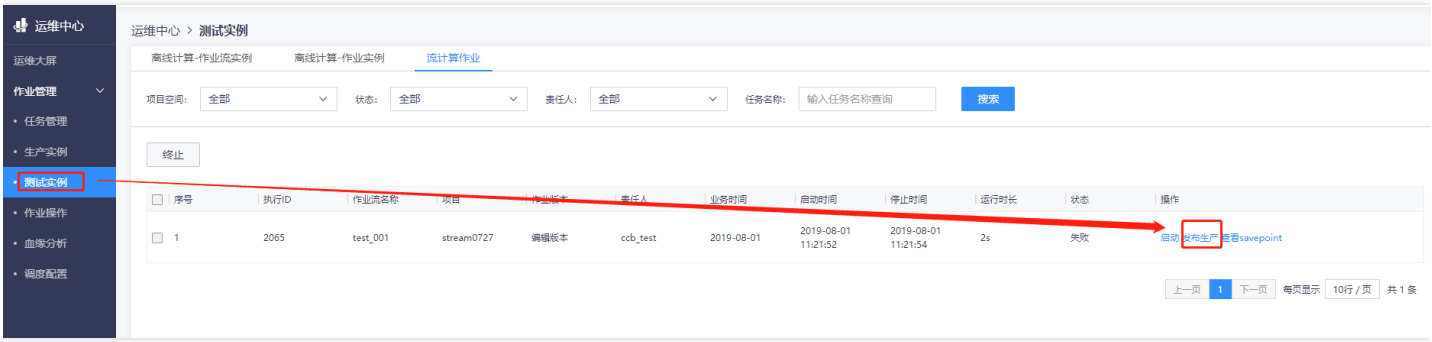
- 【查看Flink UI】 备注：查看Flink UI前，需要提前配置部署集群的hosts，否则页面无法正常跳转。



发布生产

最近更新时间: 2019-11-11 08:24:36

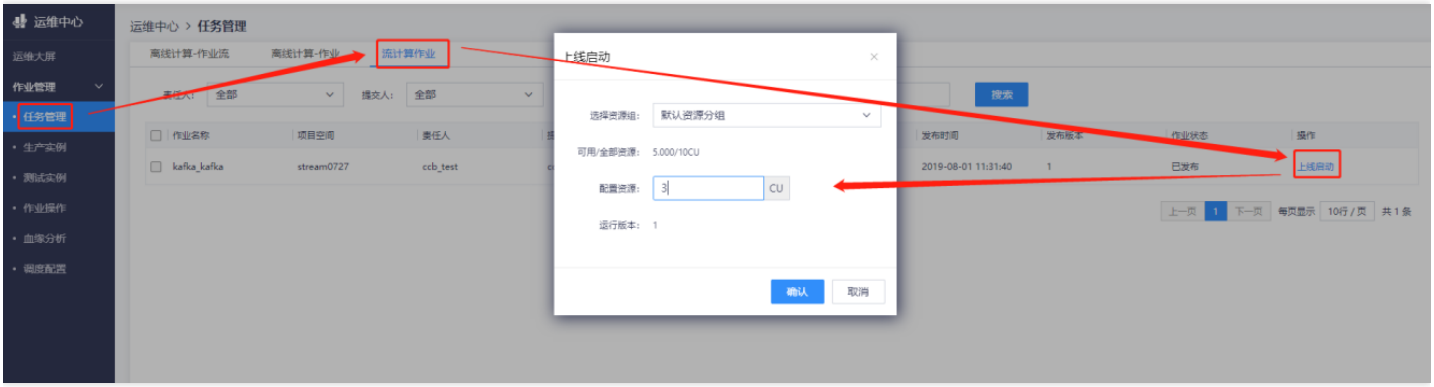
提交作业后，会在【运维中心-测试实例-流计算作业】生成一条流作业数据，可进行：【启动】【终止】【创建SavePoint】【查看SavePoint】【查看Flink UI】等操作，对于执行正常的作业，点击【发布生产】，可进入发布生产的审批流程。



上线启动

最近更新时间: 2019-11-11 08:23:52

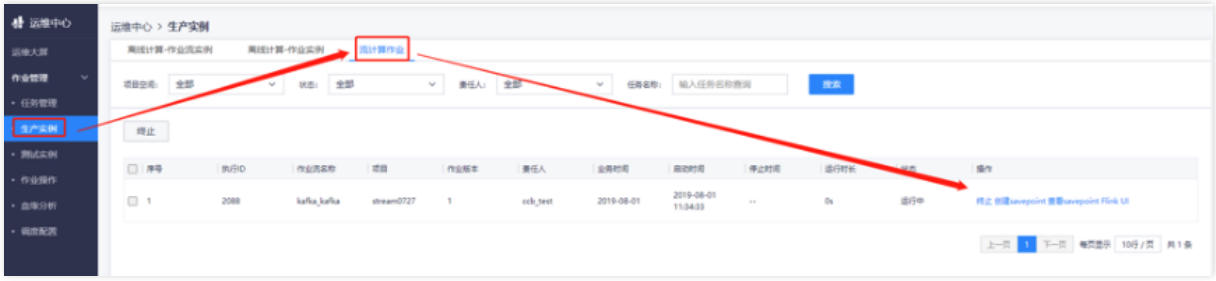
已通过审批的作业流可以在【运维中心-任务管理】页面中看到，并可以进行相应操作，已拒绝的作业流不可以在【运维中心-任务管理】中看到。 点击【上线启动】按钮，需要对待上线的作业分配运行资源。完成资源分配的作业，将提交至【运维中心-生产实例-流计算作业】中，进行调度运行：启动、停止、创建SavePoint、查看SavePoint、查看Flink UI等。



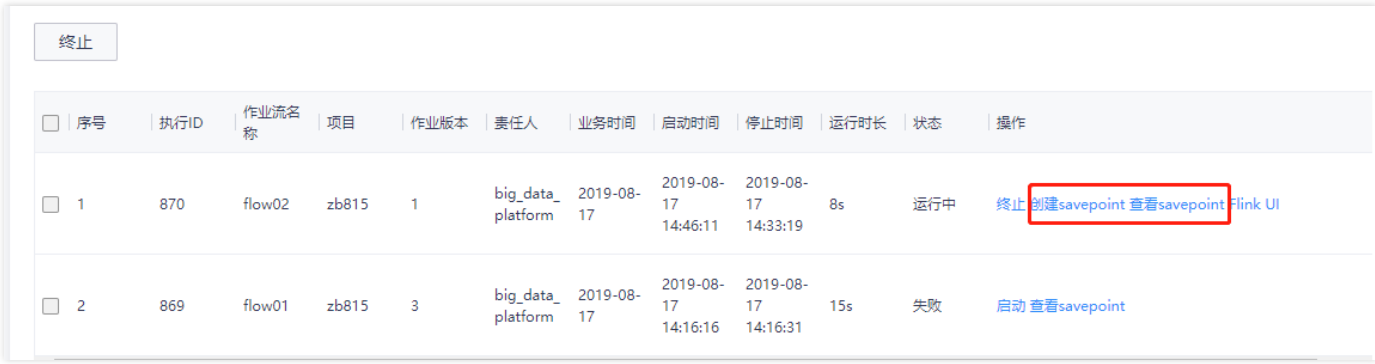
生产运行

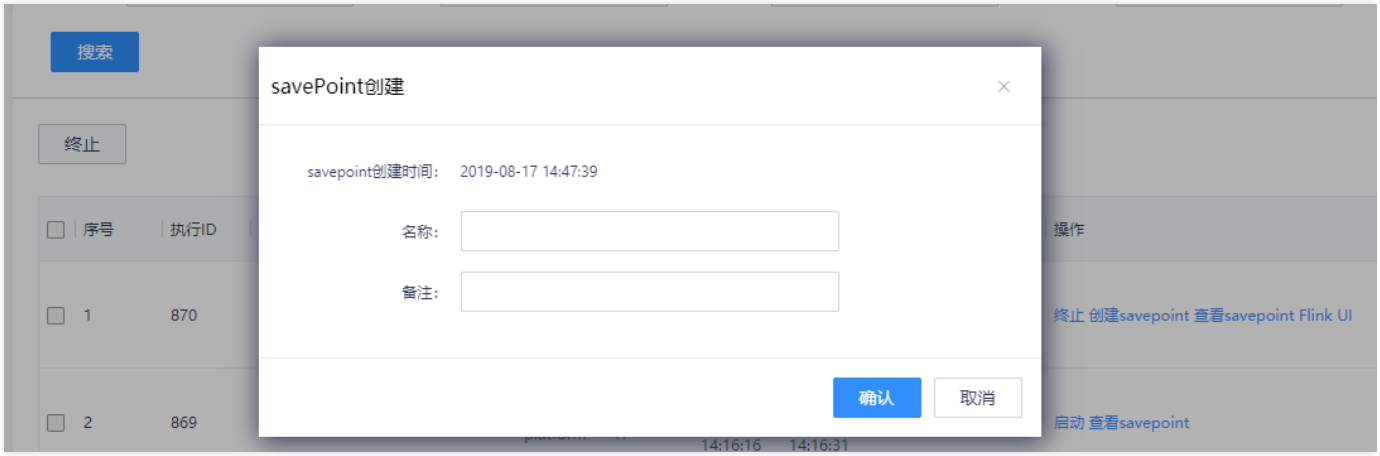
最近更新时间: 2019-11-11 08:23:52

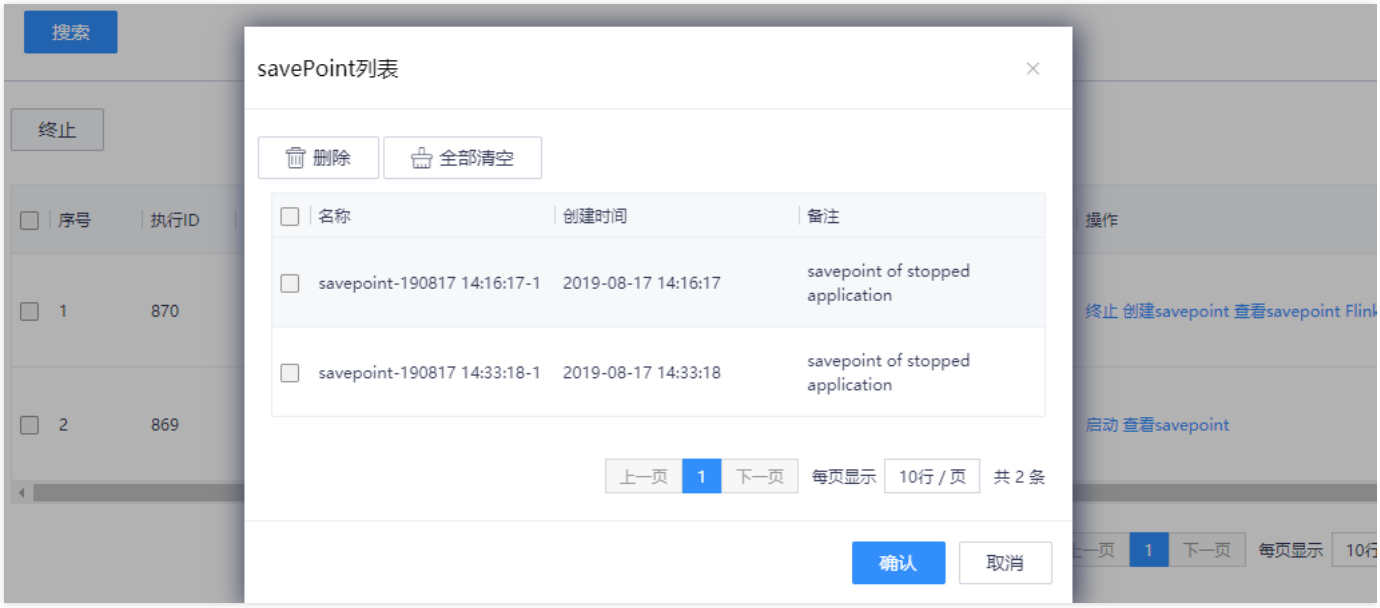
- 【启动】：选择启动位点，点击按以上配置启动，作业即提交至计算集群运行。流计 算作业启动时候您可以指定启动时间。表示从源头数据存储的指定时间点开始读取数 据。例如设置启动时间为当前时间或者一个小时之前，也可以立即启动
- 【终止】：点击终止，可停止作业；



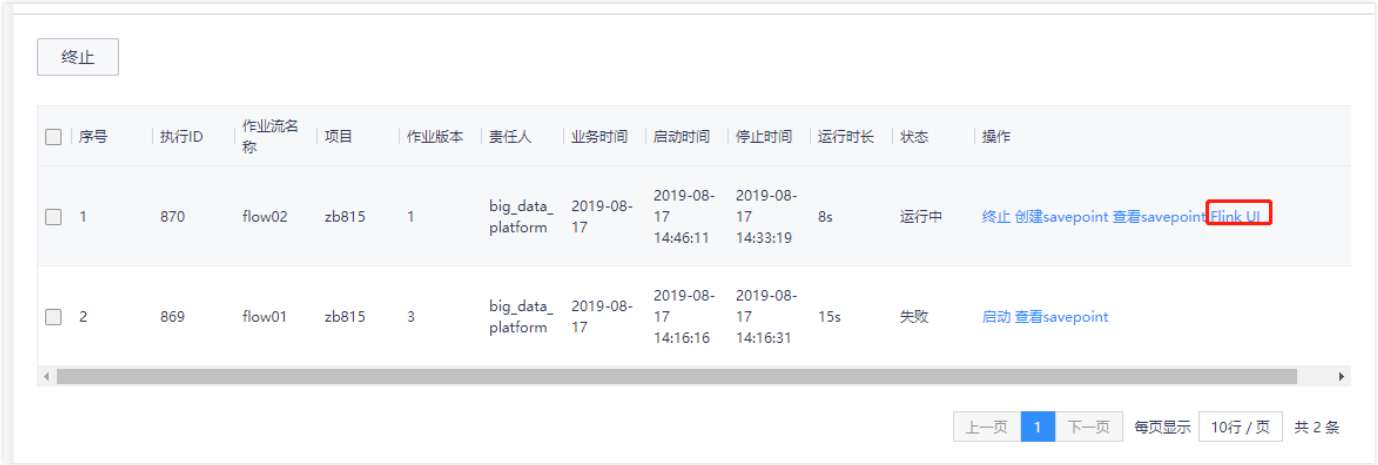
- 【创建SavePoint】：创建后，再次启动作业时，可选择从SavePoint标识的点启动作 业；
- 【查看SavePoint】：可以查看所有的SavePoint列表。

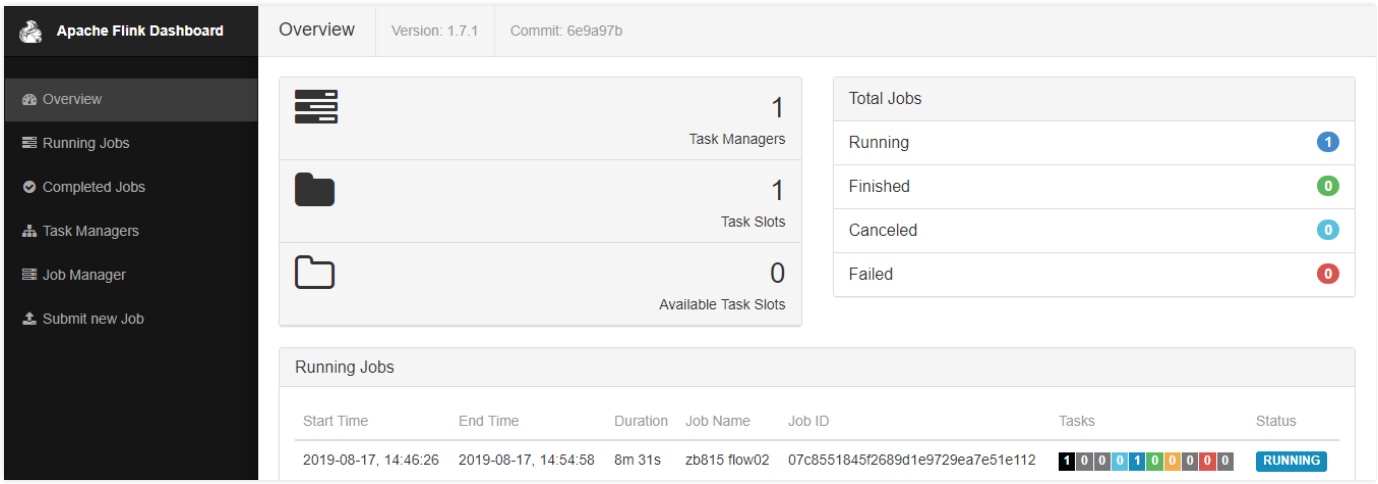






- 查看Flink UI 备注：查看Flink UI前，需要提前配置部署集群的hosts，否则页面无法正常跳转。




Apache Flink Dashboard
Overview
Version: 1.7.1
Commit: 6e9a97b
Overview
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit new Job
1
Task Managers
1
Task Slots
0
Available Task Slots
Total Jobs
Running 1
Finished 0
Canceled 0
Failed 0
Running Jobs
Start Time
End Time
Duration
Job Name
Job ID
Tasks
Status
2019-08-17, 14:46:26
2019-08-17, 14:54:58
8m 31s
zb815 flow02
07c8551845f2689d1e9729ea7e51e112
RUNNING

最佳实践
场景实践
背景

最近更新时间: 2019-11-26 14:47:12

随着互联网络技术的发展，直播的概念有了新的拓展和发展，现在更多的人关注网络直播，特别是视频直播生态链更受关注。通过网络信号，在线收看球赛、体育赛事、重大活动和新闻等，让大众有了广阔且自由的选择空间，我们能够真正的随时随地的体验直播的快乐和便捷。

在当前体验为王的时代，任何体验不佳均会导致客户群体的大规模流式。对于网站直播平台而言，需要重点关注用户(主播和粉丝)的使用体验，重点关注的系统指标包括音视频卡顿率、延迟率、丢包情况，同时由于直播平台时效性，平台方更需要实时了解到系统周边问题，并先于用户发现并定位问题和故障。另外，作为平台运营方还应该对于整体网站客户运营情况、视屏爆款情况有及时的跟踪和了解。

下面我们以某个视屏直播平台方为例，讲解下如何使用实时计算进行系统稳定性以及平台运营情况进行实时监控和展现。

业务

业务

最近更新时间: 2019-11-26 14:47:12

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。

主播和粉丝各自持有直播APP软件，该APP软件将每10s向服务器打点，服务器接收到打点信息会输出到本地磁盘，并通过阿里云日志服务采集端将打点数据到日志服务，实时计算通过订阅该日志，实时计算客户端视频播放情况。

业务目标

最近更新时间: 2019-11-26 14:47:12

针对客户端APP的监控，获取以下指标：

- 房间故障，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）。

数据格式

最近更新时间: 2019-11-11 08:28:13

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
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",
"afcnt": "34",
"adrop": "3",
"afts": "1515922566",
"alat": "123",
"vbytES": "123456",
"vfcnt": "34",
"vdrop": "4",
"vfts": "1515922566",
"vlat": "123",
"ublock": "1",
"dblock": "2",
"timEStamp": "15151922566",
"region": "beijing"
}
```

场景

最近更新时间: 2019-10-28 01:42:05

- 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。

```
SELECT
CAST(TUMBLE_START(proctime, INTERVAL '10' MINUTE) asVARCHAR) 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
GROUPBY
TUMBLE(proctime, 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。

```
SELECT
CAST(TUMBLE_START(proctime, INTERVAL '10' MINUTE) asVARCHAR) as app_ts,
region,
SUM(alat)/COUNT(alat) as alat,
SUM(vlat)/COUNT(vlat) as vlat,
FROM
view_app_heartbeat_Stream
GROUPBY
TUMBLE(proctime, 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。

```
SELECT
CAST(TUMBLE_START(proctime, INTERVAL '10' MINUTE) asVARCHAR) as app_ts,
SUM(IF(ublock <> 0 OR dblock <> 0, 1, 0)) / CAST(COUNT(DISTINCT userid) ASDOUBLE) as block_rate,
FROM
view_app_heartbeat_Stream
GROUPBY
TUMBLE(proctime, INTERVAL '10' MINUTE);
```

技术实践

最近更新时间: 2019-10-27 08:04:46

1. Kafka source并发度设置不要超过topic Partitions数量；Kafka Consumer数量不能超过topic Partitions数量，超过的部分会占用slot但不会消费数据。
2. State量大的算子，尽量在前一个算子进行keyBy，将相同key的数据归集到同一个slot。使用Flink Tuples。
3. 当使用类似于groupBy、join或keyBy这些操作时，Flink提供了多种方式以便用户在数据集中选择主键。如果现在使用的是Flink Tuple类型，那么只要简单地指定字段元组的位置，就可以被用作主键。
4. 复用Flink对象，当从用户定义的函数返回数据时，最好使用可变对象。
5. 使用注解功能。由于Flink无法解析和理解代码，所以可以提供一些有利于构建更有效执行计划的重要信息。可以使用以下三个注解： @ForwardedFields：指定输入值中哪些字段保持不变，哪些字段是用于输出的。 @NotForwardedFields：指定在输出中未保留相同位置的字段。 @ReadFields：指定用来计算结果值的字段。指定的字段应该只在计算中使用，而不仅仅是复制到输出参数中。
6. Select Join Type。 BROADCAST_HASH_SECOND：第二个数据集要小得多 REPARTITION_HASH_FIRST：第一个数据集稍微小一些 REPARTITION_HASH_SECOND：第二个数据集要小一点 REPARTITION_SORT_MERGE：使用排序和合并策略对数据集进行重新分配 **OPTIMIZER_CHOOSES：**Flink优化器将决定如何join数据集

故障指南

最近更新时间: 2019-10-28 01:42:05

- SQL不规范- 带关键字的SQL FlinkSQL的关键字，要使用反引号``引起来。 例如： select name, sum(1) as count from tmp group by id，其中count为关键字，要改为 select name, sum(1) as `count` from tmp group by id。
- 上下游Schema不匹配 经过SQL查询出的Schema要和下游算子的Schema匹配，包括字段个数、类型、名称。 例如：SQL中查询出name，age，score三个字段：select name, age,score from tmp 下游算子MySQL sink表中只有两个字段name、age。这种不匹配的情况将会报错。
- SQL不规范-窗口函数 时间类型支持Processing Time和Event Time。格式规范为： Processing Time时，时间字段使用proctime：TUMBLE(proctime,interval '10' second), Event Time时，时间字段使用rowtime：TUMBLE(rowtime,interval '10' second)。 例如：ProcessingTime时，select sum(score) as cnt from tmp group by name, TUMBLE(ctime,interval '10' second) 应该改为select sum(score) as cnt from tmp group by name, TUMBLE(proctime,interval '10' second)
- 使用Kafka中多级JSON报错 获取数组内容时，下标从1开始。例如 select score[0] from tmp 应该改为 select score[1] from tmp
- 元数据信息无法获取 1、刷新页面，重新加载； 2、退出当前用户登录，重新登录； 3、如果前两步操作仍然无法正常进入数据开发页面，则可能是服务问题，可以通过以下步骤检查： （1）浏览器进入开发者模式； （2）打开network选项框，同时点击元信息下拉菜单，查看 http://console.bigdata.yun.ccb.com/i/ide/dataManager/ideGateWay 调用返回值； （3）如果相应state不是200，反馈给后台运维人员处理。
- 资源列表无法获取 1、刷新页面，重新加载； 2、退出当前用户登录，重新登录； 3、如果前两步操作仍然无法正常进入数据开发页面，则可能是服务问题，可以通过以下步骤检查： （1）浏览器进入开发者模式； （2）打开network选项框，同时点击元信息下拉菜单，查看 http://console.bigdata.yun.ccb.com/i/ide/dataManager/getResourceGroup 调用返回值； （3）如果相应state不是200，反馈给后台运维人员处理。
- UDF中函数参数与SQL中传入的格式不匹配 在“运维中心–运行日志–YARN日志”功能中直接查看日志，查找“SQL validation failed. Given parameters of function do not match any signature. Actual: () Expected: ()”异常，如果有则修改UDF或者业务处理逻辑，重新调度运行。
- Job无报错长时间无数据输出 流计算开发界面，检测batchsize、flushInterval参数设置，可能因为设置的值过大，还没触发数据写入，调整成合适值后，重新调度运行。

常见问题
产品介绍常见问题

最近更新时间: 2021-09-15 15:59:08

Q：什么是实时流计算?

A：“实时”指实时处理，计算框架支持按消息时间逐条处理；“流”指数据如水流，一个接着一个；“计算”指数学运算、数据分析、算法模型执行等。“实时流计算”指实时处理当下正在发生的数据流，逐条大数据分析或算法运算。

Q：目前要流计算引擎支持哪几种？在使用上有什么区别?

A：目前支持Flink、Spark Streaming两种引擎。其中Flink支持上传JAR包、通过IDE拖拽插件的方式进行流计算作业的开发，Spark Streaming只支持上传JAR包的方式。

Q：目前流计算支持哪几种数据源?

A：source支持：Kafka；sink支持：Kafka、MySQL、Oracle、Elasticsearch、HBase、Redis

产品使用常见问题

最近更新时间: 2021-09-15 15:59:08

Q：如何快速测试流计算作业是否有问题?
A：可通过以下步骤进行测试。

- 在IDE开发页面进行在线开发，开发完毕后，可点击“单元测试”快速进行作业测试；
- 单元测试使用线下上传的测试数据包充当数据源，支持对数据进行延迟、乱序测试，并快速生成结果；
- 单元测试无误后，可将作业发布到测试环境，进行试运行。经过数据验证后，可将该流计算作业发布至生产环境进行正式运行。

Q：流计算作业怎么更新?
A：可通过以下步骤进行测试。

- 流计算作业支持多版本，版本生成动作通过开发IDE界面的“提交”进行触发。
- 作业提交前，可以在流计算的开发IDE页面进行作业编辑。
- 作业提交后，为保障已有版本作业的稳定，以后版本不可再进行编辑操作，但可以通过生成新版本的方式替换老版本，实现流作业的更新。

Q：SavePoint有什么作用，如何创建?
A：SavePoint可以支持程序升级后，继续从升级前的那个点开始执行计算，保证数据不中断。SavePoint需要用户手动创建。

Q：是否支持自定义函数，自定义函数的作用范围多大?
A：流计算支持用户自定义函数，函数统一作用在租户级别。

Q：流计算是否支持用户自定义扩展插件?
A：流计算支持Kafka source、SQL Operator、ETL Operator、JDBC sink、HBase sink、ElasticSearch sink、Redis sink、Kafka sink 这8个插件，可灵活满足用户的各种需求。插件由平台统一维护，不支持租户自定义扩展。

Q：流计算有哪些使用限制?
A：Flink SQL不支持TopN、Emit等语法，可采用ETL实现。Kafka source多流Join，不可同时支持多个Kafka 版本。