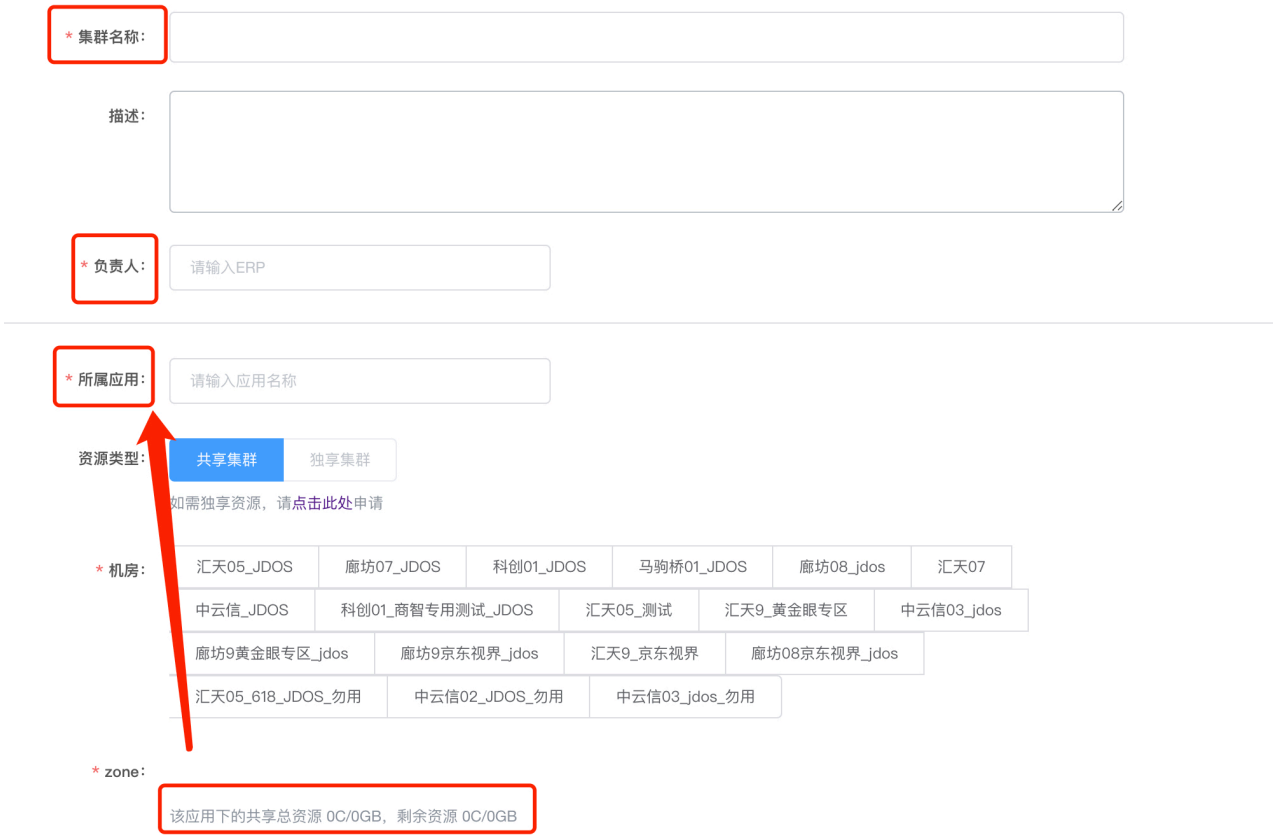
BDP Flink SQL使用入门

1. BDP Flink SQL平台简介

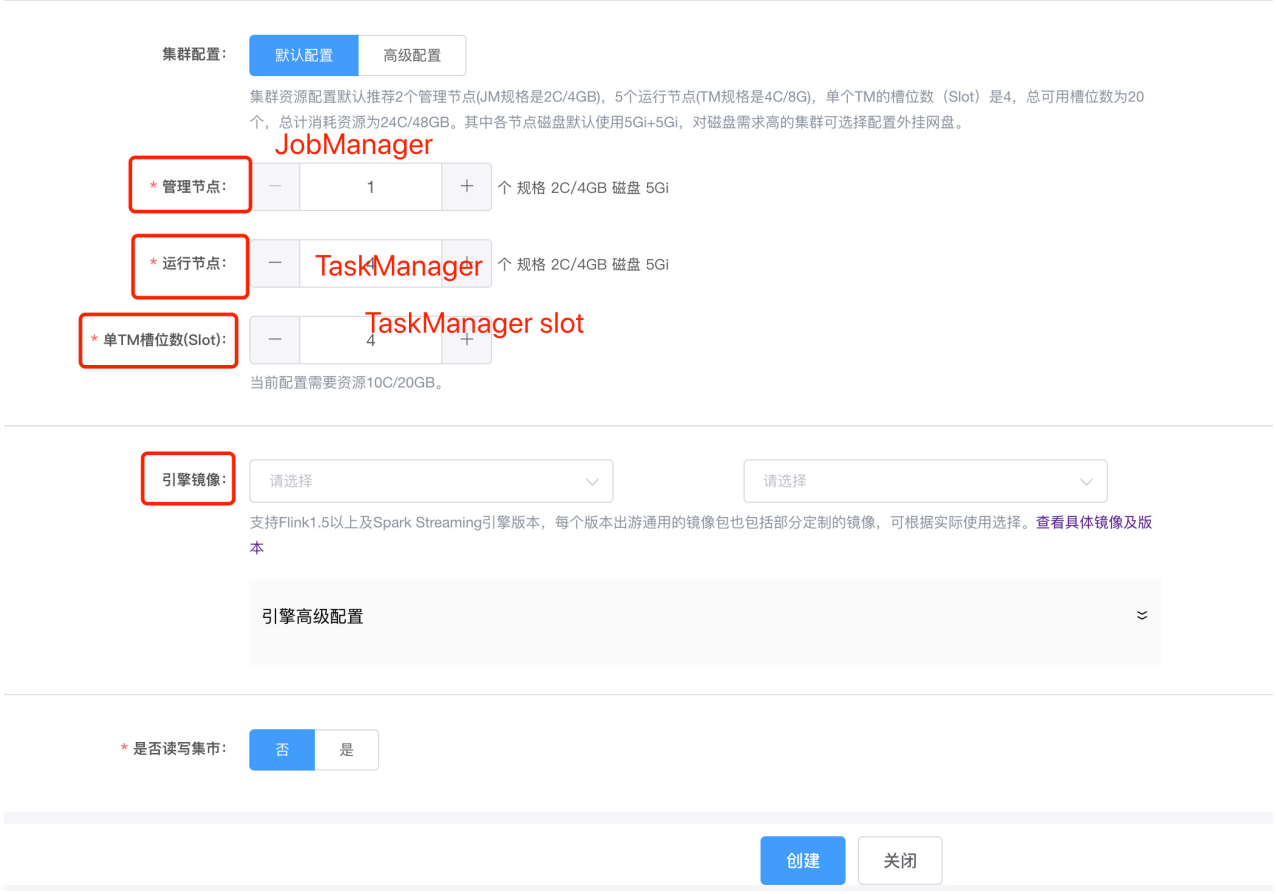
BDP Flink SQL平台网址：http://bdp.jd.com/jrdw/jrctask2/flink/management/titleMenuNoLeft.html?url=/flink

1.1 Flink SQL集群建立

1. 进入BDP 实时计算(新)页面，选择左侧菜单栏的集群管理；
2. 点击新建集群，跳转到新建集群界面；



依次填写新建必须的数据；**注意：**资源挂靠在应用上，若选择了应用后，zone下面区域出现了资源为0或剩余CPU/内存不满足需求，需要走资源申请流程。



1. 选择JM(JobManager)节点数量，为了集群HA建议选择JobManager数量选择两个；

2. 选择TM(TaskManager)节点数量，需要提前评估任务运行所需资源；计算资源数 = TM个数 * TM硬件规格
3. 选择TM的slot数量
4. 选择BDP flink镜像版本
5. 为了保障生产环境的稳定性，引擎高级配置至少需要配置以下参数：

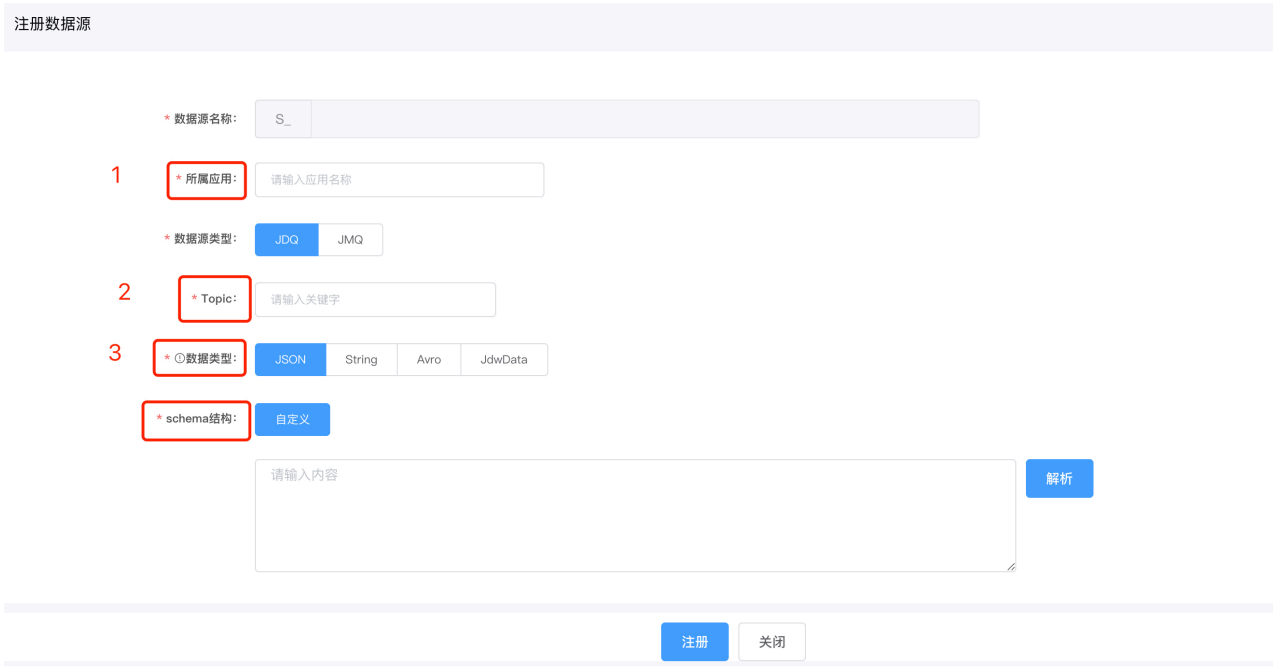
参数名称	参数值	参数说明
state_backend	rocksdb	flink state_backend类型，默认使用FsStateBackend
taskmanager.memory.jvm-metaspace.size	256m	配置JVM metaspace容量
state_backend_incremental	true	开启RocksDBStateBackend的增量checkpoint机制

6. 点击创建，即可完成集群创建

1.2 数据源/目标源创建

1.2.1 数据源创建

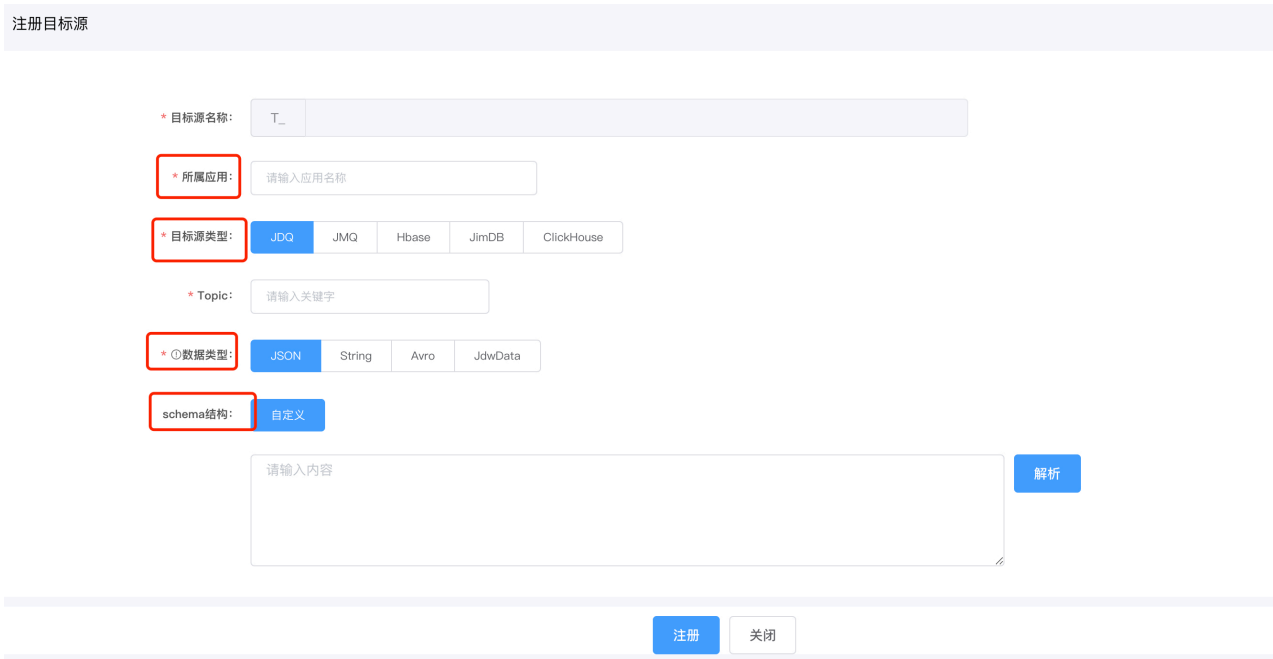
1. 点击左侧菜单的数据源管理进入数据源界面；
2. 点击注册数据源进入数据源注册界面



1. 选择数据源所属应用；
2. 数据源类型目前只支持MessageQueue中间件，根据数据来源选择JDQ or JMQ
3. 选择Topic，数据源名称会自动生成
4. 数据类型的选择与配置，相见BDP平台帮助文档：http://bdp.jd.com/helpCenter/front/showDocumentList.html?docId=458
5. 点击注册完成数据源注册

1.2.2 目标创建

1. 选择左侧菜单栏数据源管理；
2. 点击注册目标源，跳转进入目标源(sink)注册界面



1. 选择所属应用
2. 选择目标源类型，flink SQL将Stream流先抓换为动态表，进行计算，计算完毕，将动态表写入外部系统时，需要转换为不同类型的Stream流(目前flink Table &。SQL支持`Append-only`流、`Retract`流、`Upsert`流)，根据目标源类型特性(例如：是否支持update)与业务场景选择合适的目标源；
3. 根据选择的目标源，配置连接参数
4. 点击注册

2. Flink SQL任务基本组成与详解

一个完成的Flink SQL Job由source，transform，sink组成；与flink Java任务一致

source，sink 以及各个参数含义

transform语法

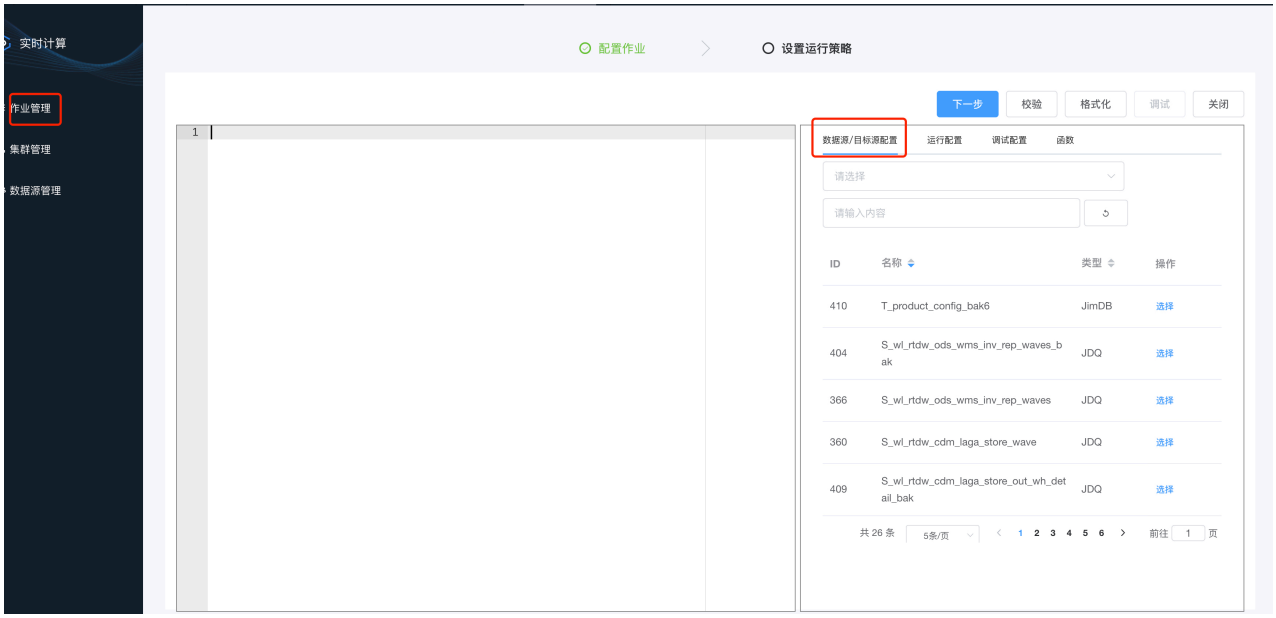
2.1 source

2.1.1 source简介

CREATE 语句用于向当前或指定的 Catalog 中注册表、视图或函数。注册后的表、视图和函数可以在 SQL 查询中使用。目前BDP平台支持flink 1.9版本，因此不支持创建视图，待BDP flink版本升级即可支持。

BDP目前数据源支持JDQ与JMQ，实时数仓基本使用JDQ作为数据管道，因此以JDQ为例解析Source语句。

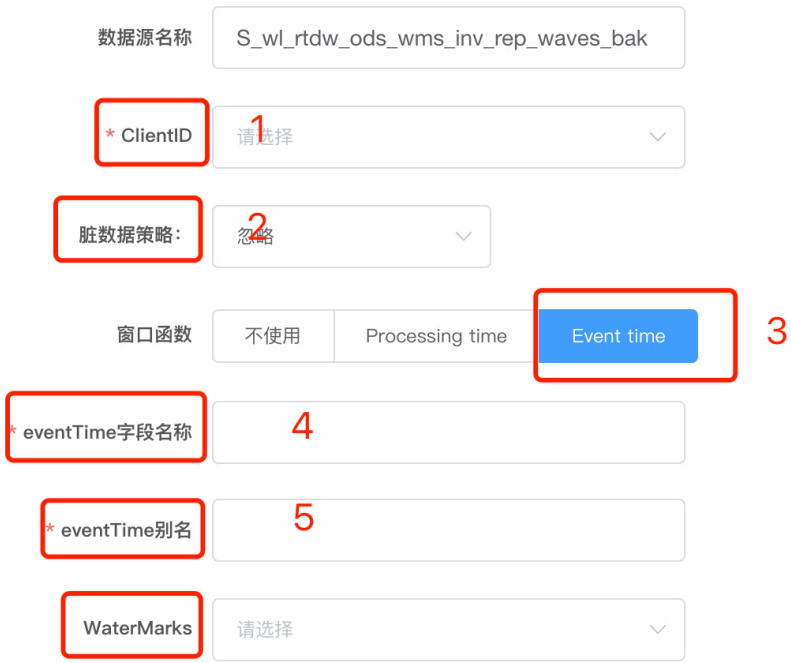
2.1.2 SQL中创建source table



注意：T开头的是目标源；S开头的是数据源

配置source详细信息：

引入数据源



字段名称	字段类型	字段描述
mid	long	
db	string	
sch	string	
tab	string	
opt	string	

1. 选择JDQ 消费者的clientId；
2. 选择脏数据处理策略：忽略/抛出异常；如果业务场景容忍部分脏数据的丢失，可以选择忽略；
3. 时间语义根据业务场景需求选择：processing time/event_time
4. flink SQL event_time时间列的名称
5. Event_time字段列别名，可以在flink sql中使用别名做查询
6. 选择watermark生成方式
7. 点击完成后即可在SQL界面生成下面的 `CREATE SOURCE TABLE` 语句

   PS：数据拉取(`startmode`)模式需要修改为：`groupoffsets`，这样回放数据才会生效。这个需要在source生成之后**直接在SQL中修改即可**。

```
CREATE SOURCE TABLE source_1(
 mid long,
 db string,
 sch string,
```

```
 tab string,
 opt string,
 ts long,
 ddl string,
 err string,
 src map,
 cur map,
 cus map
)
with(
 password='xxxxxx',  # JDQ accesskey
 clientid='xxxxxx',  # JDQ consumer clientid
 sourceType='jdq',
 startmode='latest',  # 任务启动时从JDQ数据拉取方式
 watermarktype='watermarksPeriodicAscending', # watermark生成方式
 timecolumnalias='rowtime', # event_time模式下，时间列的别名
 processmethod='0', # 对于脏数据处理模式
 topic='xxxxxx', # source来源topic名称
 timetype='eventtime', # 时间语义选择event_time
 timecolumn='cur['ts']', # 时间语义对应的字段
 username='xxxxxxxx' # JDQ 应用域名
);
```

2.2 Sink

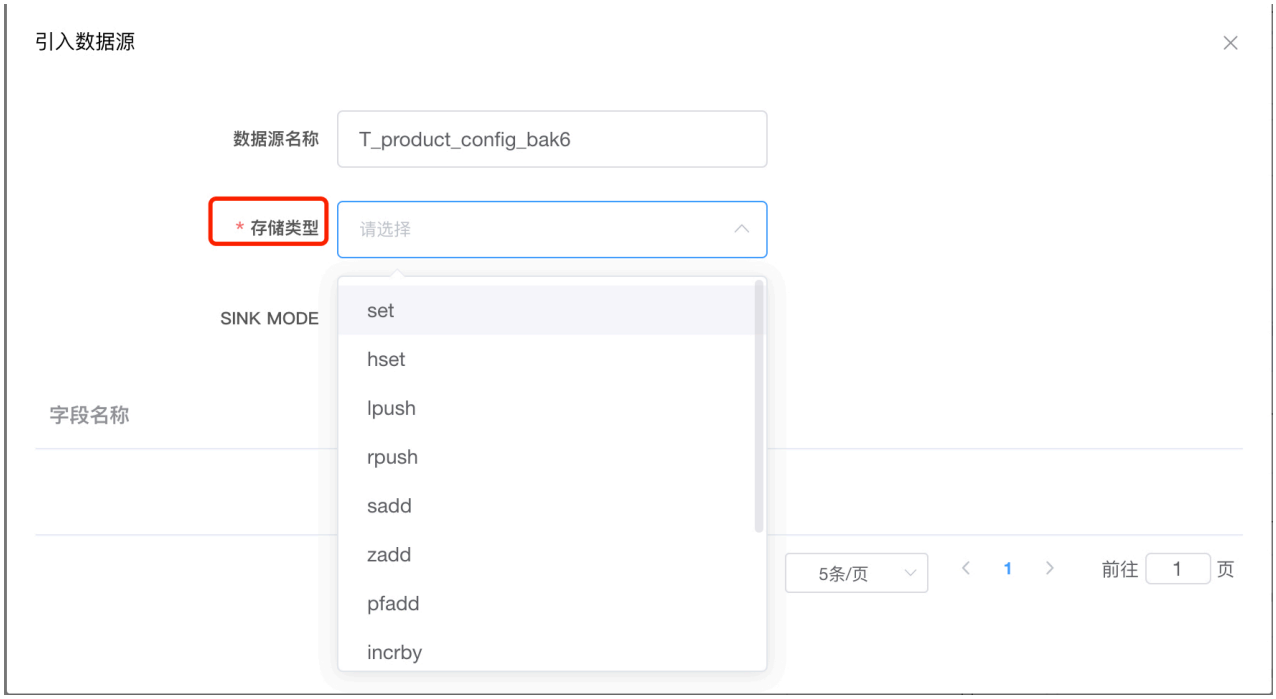
2.2.1 Sink源简介

BDP目前支持的sink源有：JMQ，JDQ，JimDB(Redis)，HBase，ClickHouse

目前项目中仅使用过JimDB和JDQ作为目标源。后续会探索ClickHouse和Elasticsearch。

2.2.2 Sink源创建

选择以T开头的sink数据源，以jimDB源为例创建Sink源。



- **存储类型**：对应redis的数据结构和redis的命令。
- **SINK MODE**：append，retract，upsert，它们的区别和说明如下，根据业务场景和数据使用要求选择适合的sink mode。
    - **Append-only 流：** 仅通过 `INSERT` 操作修改的动态表可以通过输出插入的行转换为流。
    - **Retract 流：** retract 流包含两种类型的 message： *add messages* 和 *retract messages* 。通过将`INSERT` 操作编码为 add message、将 `DELETE` 操作编码为 retract message、将 `UPDATE` 操作编码为更新(先前)行的 retract message 和更新(新)行的 add message，将动态表转换为 retract 流。
    - **Upsert流：** upsert 流包含两种类型的 message： *upsert messages* 和*delete messages*。转换为 upsert 流的动态表需要(可能是组合的)唯一键。通过将 `INSERT` 和 `UPDATE` 操作编码为 upsert message，将 `DELETE` 操作编码为 delete message ，将具有唯一键的动态表转换为流。消费流的算子需要知道唯一键的属性，以便正确地应用 message。与 retract 流的主要区别在于 `UPDATE` 操作是用单个 message 编码的，因此效率更高。

配置完成点击引入数据源，在SQL界面生成如下SQL DDL语句：

```
CREATE SINK TABLE sink_1(
 key string,
 mapkey string,
 mapvalue string
)
with(
 storeType='hset', # redis数据结构与命令
 sinkmode='retract', # 动态表转stream策略
 sinkType='jimdb', # 目标源类型
 jimurl='xxxxxxxx', # jimdb url
 ttl='1', # redis key 过期时间设置
 ttlunit='DAYS'
);
```

2.3 Flink SQL 数据计算

Flink SQL支持标准SQL语法，并使用ANSI SQL标准的Apache Calcite解析、优化SQL。此外flink SQL最终被转换为DataStream/DataSet，用于实现批流一体处理。

SQL语法如下：

```
SELECT [ ALL | DISTINCT ]
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]
  [ HAVING booleanExpression ]
  [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]
```

与标准SQL不同的是，Flink SQL支持window(窗口)函数，用于实时加工：

- 滚动窗口（TUMBLE）
- 滑动窗口（HOP）
- 会话窗口（SESSION）
- OVER窗口（OVER Window）

实际开发中的SQL数据计算部分如下：

```
select
    case
    when company_code = '1' THEN '1'
    else '9' end as entrust_owner_type,
    wh_no,
    cur_status,
    src_status,
    order_no,
    date_str,
    rowtime
```

```
from(select
    opt,
    if(cur['order_no'] IS NOT NULL,cur['order_no'],src['order_no']) as
order_no,
    if(cur['company_code'] IS NOT NULL,cur['company_code'],src['company_code'])
as company_code,
    if(cur['wh_no'] IS NOT NULL,cur['wh_no'],src['wh_no']) as wh_no,
    if(cur['yn'] IS NOT NULL,cur['yn'],src['yn']) as yn,
    if(cur['create_time'] IS NOT NULL,cur['create_time'], '') as create_time,
    if(cur['status'] IS NOT NULL,cur['status'], '') as cur_status,
    if(src['status'] IS NOT NULL,src['status'], '') as src_status,
    rowtime,
    cast(DATE_FORMAT(cast(TIMESTAMPADD(HOUR,8,rowtime) as varchar), 'yyyyMMdd')
as varchar)  as date_str
from S_wl_rtdw_ods_wms_inv_rep_waves_bak) where yn = '1' and create_time <> ''
and REPLACE(SUBSTRING(create_time, 0, 10),'-','') = date_str;
```

2.4 完成的Flink SQL任务

source，transform，sink组成了一个完整的Flink Job。

```
CREATE SOURCE TABLE source_1(
 mid long,
 db string,
 sch string,
 tab string,
 opt string,
 ts long,
 ddl string,
 err string,
 src map,
 cur map,
 cus map
)
with(
 password='xxxxxx',  # JDQ accesskey
 clientid='xxxxxx',  # JDQ consumer clientid
 sourceType='jdq',
 startmode='latest',  # 任务启动时从JDQ数据拉取方式
 watermarktype='watermarksPeriodicAscending', # watermark生成方式
 timecolumnalias='rowtime', # event_time模式下，时间列的别名
 processmethod='0', # 对于脏数据处理模式
 topic='xxxxxx', # source来源topic名称
 timetype='eventtime', # 时间语义选择event_time
 timecolumn='cur['ts']', # 时间语义对应的字段
 username='xxxxxxxx' # JDQ 应用域名
);
```

```
CREATE SINK TABLE sink_1(
 key string,
 mapkey string,
 mapvalue string
)
with(
 storeType='hset', # redis数据结构与命令
 sinkmode='retract', # 动态表转stream策略
 sinkType='jimdb', # 目标源类型
 jimurl='xxxxxxxx', # jimdb url
 ttl='1', # redis key 过期时间设置
 ttlunit='DAYS'
);

CREATE TABLE transfrom_1
select
    case
    when company_code = '1' THEN '1'
    else '9' end as entrust_owner_type,
    wh_no,
    cur_status,
    src_status,
    order_no,
    date_str,
    rowtime
from(select
    opt,
    if(cur['order_no'] IS NOT NULL,cur['order_no'],src['order_no']) as
order_no,
    if(cur['company_code'] IS NOT NULL,cur['company_code'],src['company_code'])
as company_code,
    if(cur['wh_no'] IS NOT NULL,cur['wh_no'],src['wh_no']) as wh_no,
    if(cur['yn'] IS NOT NULL,cur['yn'],src['yn']) as yn,
    if(cur['create_time'] IS NOT NULL,cur['create_time'], '') as create_time,
    if(cur['status'] IS NOT NULL,cur['status'], '') as cur_status,
    if(src['status'] IS NOT NULL,src['status'], '') as src_status,
    rowtime,
    cast(DATE_FORMAT(cast(TIMESTAMPADD(HOUR,8,rowtime) as varchar), 'yyyyMMdd')
as varchar)  as date_str
from source_1) where yn = '1' and create_time <> '' and
REPLACE(SUBSTRING(create_time, 0, 10),'-','') = date_str;

INSERT INTO sink_1
select
    'test',
    CONCAT_WS('_',company_type,store_id),
    CAST(out_store_daily_num as varchar)
from transfrom_1;
```

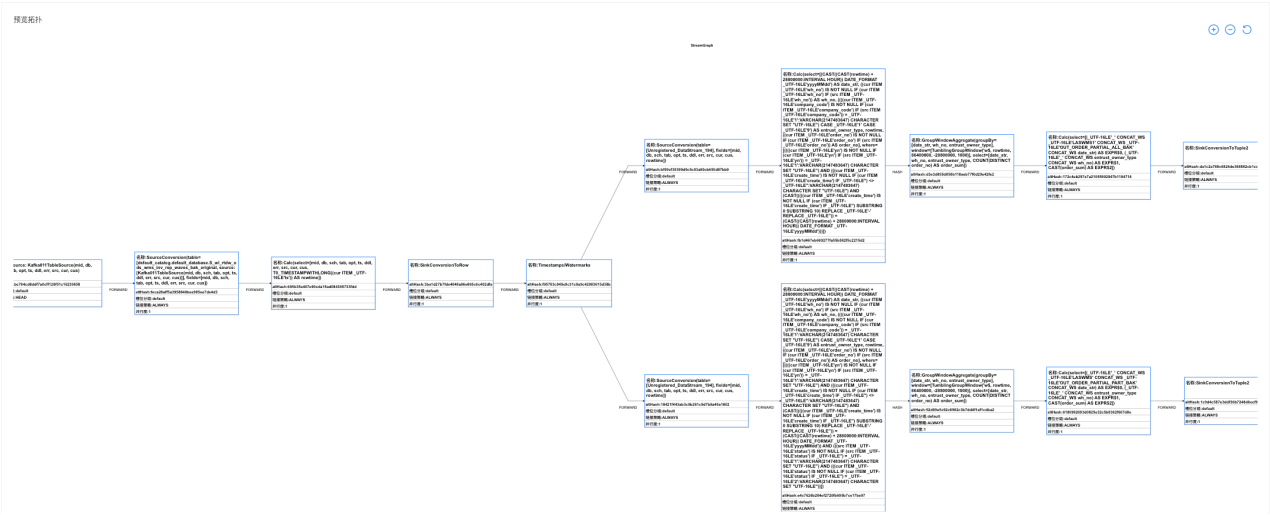
3. BDP Flink SQL任务发布

任务发布流程，启动参数，任务配置，集群配置，job拓扑图修改

1. 点击SQL界面右侧的运行配置，进入flink job运行时的配置：
    - **集群：**上面创建好的集群，可以在同一个集群上部署多个flink SQL Job；
    - **默认并行度：**由于SQL中无法配置各个算子的并行度，因此需要设置默认并行度；在提交任务，形成StreamGraph后可以修改各个算子并行度
    - **checkpoint MODE：**EXACTLY_ONCE，AT_LEAST_ONCE
    - **checkponit间隔(ms)**：两次checkpoint之间的时间间隔
    - **注意：**需要勾选**任务异常告警**和**使用Blink Planner**；Blink Planner是阿里内部开发SQL优化器，flink planner 是Flink老版优化器，1.9版本之后阿里贡献给flink开源社区，1.11成为默认的SQL优化器

- **任务配置：**任务的一些高级配置，使用<K, V>的数据结构

2. 点击校验，检查SQL语法，通过校验即可点击下一步
3. 点击下一步提交任务，构建拓扑图，拓扑图如下所示，可以在拓扑上调整各个算子的并行度



4. 点击上线审批，即可完成一次发布上线

4. Flink SQL调试

目前Flink SQL没有办法加日志调试，因此想到的方法有：

- 输出各个中间结果到第三方存储介质
- 写UDF将关键计算结果输出到日志或者其他地方