实时数仓|基于Flink1.11的SQL构建实时数仓探索实践

实时数仓主要是为了解决传统数仓数据时效性低的问题，实时数仓通常会用在实时的OLAP分析、实时的数据看板、业务指标实时监控等场景。虽然关于实时数仓的架构及技术选型与传统的离线数仓会存在差异，但是关于数仓建设的基本方法论是一致的。本文会分享基于Flink SQL从0到1搭建一个实时数仓的demo，涉及数据采集、存储、计算、可视化整个处理流程。通过本文你可以了解到：

- 实时数仓的基本架构
- 实时数仓的数据处理流程
- Flink1.11的SQL新特性
- Flink1.11存在的bug
- 完整的操作案例

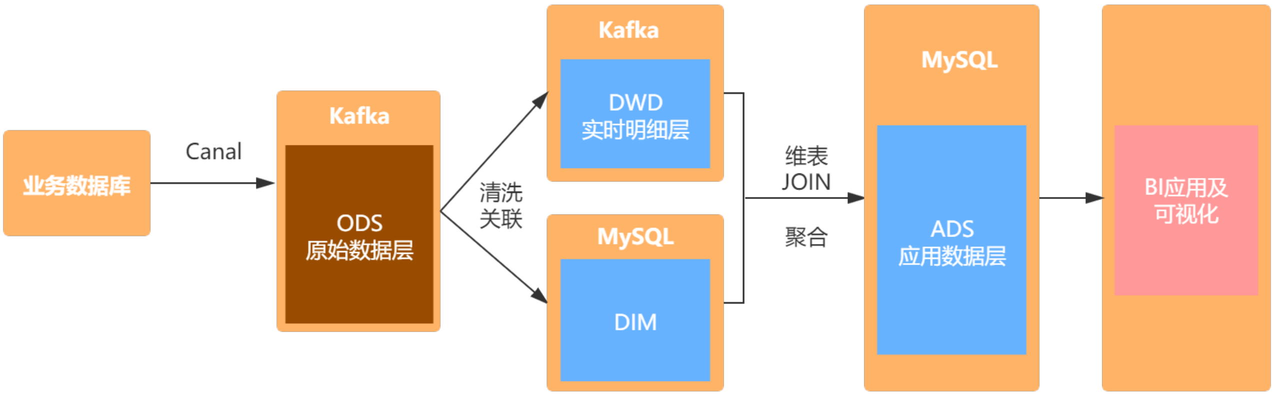
> 古人学问无遗力，少壮工夫老始成。
> 纸上得来终觉浅，绝知此事要躬行。

案例简介

本文会以电商业务为例，展示实时数仓的数据处理流程。另外，本文旨在说明实时数仓的构建流程，所以不会涉及太复杂的数据计算。为了保证案例的可操作性和完整性，本文会给出详细的操作步骤。为了方便演示，本文的所有操作都是在Flink SQL Cli中完成的。

架构设计

具体的架构设计如图所示：首先通过canal解析MySQL的binlog日志，将数据存储在Kafka中。然后使用Flink SQL对原始数据进行清洗关联，并将处理之后的明细宽表写入kafka中。维表数据存储在MySQL中，通过Flink SQL对明细宽表与维表进行JOIN，将聚合后的数据写入MySQL，最后通过FineBI进行可视化展示。



业务数据准备

- 订单表（order_info）

```sql
CREATE TABLE `order_info` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
  `consignee` varchar(100) DEFAULT NULL COMMENT '收货人',
  `consignee_tel` varchar(20) DEFAULT NULL COMMENT '收件人电话',
  `total_amount` decimal(10,2) DEFAULT NULL COMMENT '总金额',
  `order_status` varchar(20) DEFAULT NULL COMMENT '订单状态',
  `user_id` bigint(20) DEFAULT NULL COMMENT '用户id',
  `payment_way` varchar(20) DEFAULT NULL COMMENT '付款方式',
  `delivery_address` varchar(1000) DEFAULT NULL COMMENT '送货地址',
  `order_comment` varchar(200) DEFAULT NULL COMMENT '订单备注',
  `out_trade_no` varchar(50) DEFAULT NULL COMMENT '订单交易编号（第三方支付用)',
  `trade_body` varchar(200) DEFAULT NULL COMMENT '订单描述(第三方支付用)',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `operate_time` datetime DEFAULT NULL COMMENT '操作时间',
  `expire_time` datetime DEFAULT NULL COMMENT '失效时间',
  `tracking_no` varchar(100) DEFAULT NULL COMMENT '物流单编号',
  `parent_order_id` bigint(20) DEFAULT NULL COMMENT '父订单编号',
  `img_url` varchar(200) DEFAULT NULL COMMENT '图片路径',
  `province_id` int(20) DEFAULT NULL COMMENT '地区',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='订单表';
```

- 订单详情表（order_detail）

```sql
CREATE TABLE `order_detail` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
  `order_id` bigint(20) DEFAULT NULL COMMENT '订单编号',
  `sku_id` bigint(20) DEFAULT NULL COMMENT 'sku_id',
  `sku_name` varchar(200) DEFAULT NULL COMMENT 'sku名称（冗余)',
  `img_url` varchar(200) DEFAULT NULL COMMENT '图片名称（冗余)',
  `order_price` decimal(10,2) DEFAULT NULL COMMENT '购买价格(下单时sku价格)
',
  `sku_num` varchar(200) DEFAULT NULL COMMENT '购买个数',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='订单详情表'
;
```

- 商品表（sku_info）

```
CREATE TABLE `sku_info` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT 'skuid(itemID)',
  `spu_id` bigint(20) DEFAULT NULL COMMENT 'spuid',
  `price` decimal(10,0) DEFAULT NULL COMMENT '价格',
  `sku_name` varchar(200) DEFAULT NULL COMMENT 'sku名称',
  `sku_desc` varchar(2000) DEFAULT NULL COMMENT '商品规格描述',
  `weight` decimal(10,2) DEFAULT NULL COMMENT '重量',
  `tm_id` bigint(20) DEFAULT NULL COMMENT '品牌(冗余)',
  `category3_id` bigint(20) DEFAULT NULL COMMENT '三级分类id（冗余)',
  `sku_default_img` varchar(200) DEFAULT NULL COMMENT '默认显示图片(冗余)',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='商品表';
```

- 商品一级类目表（base_category1）

```
CREATE TABLE `base_category1` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
  `name` varchar(10) NOT NULL COMMENT '分类名称',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='一级分类表'
;
```

- 商品二级类目表（base_category2）

```
CREATE TABLE `base_category2` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
  `name` varchar(200) NOT NULL COMMENT '二级分类名称',
  `category1_id` bigint(20) DEFAULT NULL COMMENT '一级分类编号',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='二级分类表'
;
```

- 商品三级类目表（base_category3）

```
CREATE TABLE `base_category3` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
  `name` varchar(200) NOT NULL COMMENT '三级分类名称',
  `category2_id` bigint(20) DEFAULT NULL COMMENT '二级分类编号',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='三级分类表'
;
```

- 省份表（base_province）

```
CREATE TABLE `base_province` (
  `id` int(20) DEFAULT NULL COMMENT 'id',
  `name` varchar(20) DEFAULT NULL COMMENT '省名称',
  `region_id` int(20) DEFAULT NULL COMMENT '大区id',
  `area_code` varchar(20) DEFAULT NULL COMMENT '行政区位码'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- 区域表（base_region）

```
CREATE TABLE `base_region` (
  `id` int(20) NOT NULL COMMENT '大区id',
  `region_name` varchar(20) DEFAULT NULL COMMENT '大区名称',
   PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

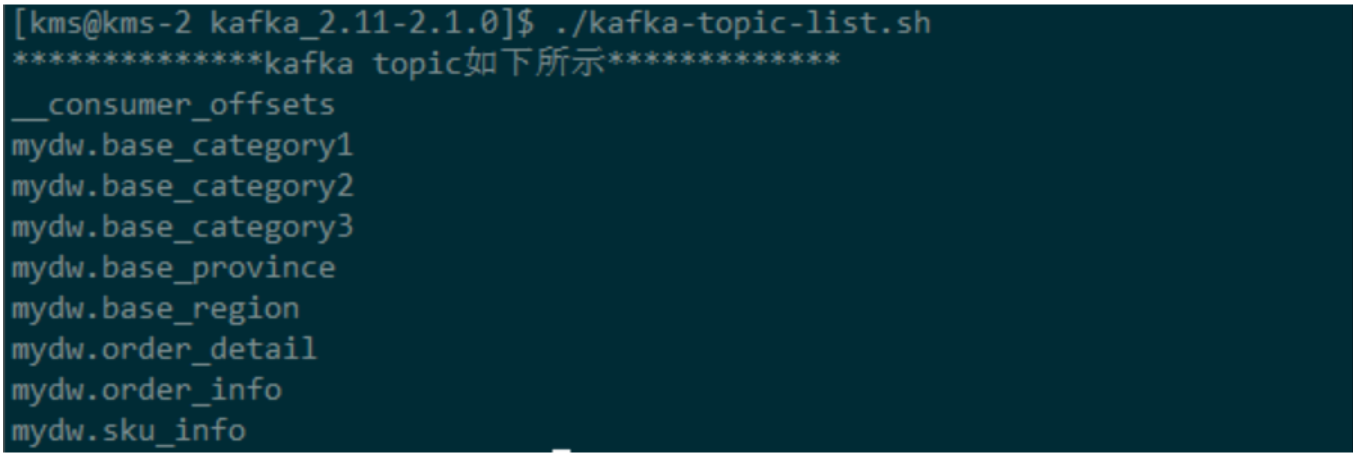
> 注意：以上的建表语句是在MySQL中完成的，完整的建表及模拟数据生成脚本见：
> 链接：https://pan.baidu.com/s/1fcMgDHGKedOpzqLbSRUGwA 提取码：zuqw

数据处理流程

ODS层数据同步

关于ODS层的数据同步参见我的另一篇文章基于Canal与Flink实现数据实时增量同步(一)。主要使用canal解析MySQL的binlog日志，然后将其写入到Kafka对应的topic中。由

于篇幅限制，不会对具体的细节进行说明。同步之后的结果如下图所示：



```
[kms@kms-2 kafka_2.11-2.1.0]$ ./kafka-topic-list.sh
**************kafka topic如下所示*************
__consumer_offsets
mydw.base_category1
mydw.base_category2
mydw.base_category3
mydw.base_province
mydw.base_region
mydw.order_detail
mydw.order_info
mydw.sku_info
```

DIM层维表数据准备

本案例中将维表存储在了MySQL中，实际生产中会用HBase存储维表数据。我们主要用到两张维表：**区域维表**和**商品维表**。处理过程如下：

- 区域维表

首先将`mydw.base_province`和`mydw.base_region`这个主题对应的数据抽取到MySQL中，主要使用Flink SQL的Kafka数据源对应的canal-json格式，注意：在执行装载之前，需要先在MySQL中创建对应的表，本文使用的MySQL数据库的名字为**dim**，用于存放维表数据。如下：

```
-- -------------------------
--   省份
--   kafka Source
-- -------------------------
DROP TABLE IF EXISTS `ods_base_province`;
CREATE TABLE `ods_base_province` (
  `id` INT,
  `name` STRING,
  `region_id` INT ,
  `area_code`STRING
) WITH(
'connector' = 'kafka',
 'topic' = 'mydw.base_province',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;


```

```
-- -------------------------
--   省份
--   MySQL Sink
-- -------------------------
DROP TABLE IF EXISTS `base_province`;
CREATE TABLE `base_province` (
    `id` INT,
    `name` STRING,
    `region_id` INT ,
    `area_code`STRING,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'base_province', -- MySQL中的待插入数据的表
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'sink.buffer-flush.interval' = '1s'
);

-- -------------------------
--   省份
--   MySQL Sink Load Data
-- -------------------------
INSERT INTO base_province
SELECT *
FROM ods_base_province;

-- -------------------------
--   区域
--   kafka Source
-- -------------------------
DROP TABLE IF EXISTS `ods_base_region`;
CREATE TABLE `ods_base_region` (
  `id` INT,
  `region_name` STRING
) WITH(
'connector' = 'kafka',
 'topic' = 'mydw.base_region',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- -------------------------
--   区域
--   MySQL Sink
-- -------------------------
DROP TABLE IF EXISTS `base_region`;
CREATE TABLE `base_region` (
    `id` INT,
    `region_name` STRING,
     PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'base_region', -- MySQL中的待插入数据的表
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'sink.buffer-flush.interval' = '1s'
);

-- -------------------------
--   区域
--   MySQL Sink Load Data
-- -------------------------
INSERT INTO base_region
SELECT *
FROM ods_base_region;
```

经过上面的步骤，将创建维表所需要的原始数据已经存储到了MySQL中，接下来就需要在MySQL中创建维表，我们使用上面的两张表，创建一张视图：`dim_province`作为维表：

```
-- ---------------------------------
-- DIM层,区域维表,
-- 在MySQL中创建视图
-- ---------------------------------
DROP VIEW IF EXISTS dim_province;
CREATE VIEW dim_province AS
SELECT
  bp.id AS province_id,
  bp.name AS province_name,
  br.id AS region_id,
  br.region_name AS region_name,
  bp.area_code AS area_code
FROM base_region br
     JOIN base_province bp ON br.id= bp.region_id
;
```

这样我们所需要的维表：dim_province就创建好了，只需要在维表join时，使用Flink SQL创建JDBC的数据源，就可以使用该维表了。同理，我们使用相同的方法创建商品维表，具体如下：

```sql
-- -------------------------
--   一级类目表
--    kafka Source
-- -------------------------
DROP TABLE IF EXISTS `ods_base_category1`;
CREATE TABLE `ods_base_category1` (
  `id` BIGINT,
  `name` STRING
)WITH(
 'connector' = 'kafka',
 'topic' = 'mydw.base_category1',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- -------------------------
--   一级类目表
--    MySQL Sink
-- -------------------------
DROP TABLE IF EXISTS `base_category1`;
CREATE TABLE `base_category1` (
    `id` BIGINT,
    `name` STRING,
     PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'base_category1', -- MySQL中的待插入数据的表
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'sink.buffer-flush.interval' = '1s'
);

-- -------------------------
--   一级类目表
--    MySQL Sink Load Data
-- -------------------------

INSERT INTO base_category1
SELECT *
FROM ods_base_category1;
```

```sql

-- -------------------------
--  二级类目表
--   kafka Source
-- -------------------------
DROP TABLE IF EXISTS `ods_base_category2`;
CREATE TABLE `ods_base_category2` (
  `id` BIGINT,
  `name` STRING,
  `category1_id` BIGINT
)WITH(
'connector' = 'kafka',
 'topic' = 'mydw.base_category2',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- -------------------------
--  二级类目表
--   MySQL Sink
-- -------------------------
DROP TABLE IF EXISTS `base_category2`;
CREATE TABLE `base_category2` (
    `id` BIGINT,
    `name` STRING,
    `category1_id` BIGINT,
     PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'base_category2', -- MySQL中的待插入数据的表
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'sink.buffer-flush.interval' = '1s'
);

-- -------------------------
--  二级类目表
--   MySQL Sink Load Data
-- -------------------------
INSERT INTO base_category2
SELECT *
FROM ods_base_category2;

-- -------------------------
-- 三级类目表
--   kafka Source
-- -------------------------
DROP TABLE IF EXISTS `ods_base_category3`;
CREATE TABLE `ods_base_category3` (
  `id` BIGINT,
  `name` STRING,
  `category2_id` BIGINT
)WITH(
'connector' = 'kafka',
 'topic' = 'mydw.base_category3',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- -------------------------
--  三级类目表
--   MySQL Sink
-- -------------------------
DROP TABLE IF EXISTS `base_category3`;
CREATE TABLE `base_category3` (
    `id` BIGINT,
    `name` STRING,
    `category2_id` BIGINT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'base_category3', -- MySQL中的待插入数据的表
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'sink.buffer-flush.interval' = '1s'
);

-- -------------------------
--  三级类目表
--   MySQL Sink Load Data
-- -------------------------
INSERT INTO base_category3
SELECT *
FROM ods_base_category3;

-- -------------------------
--   商品表
--   Kafka Source
-- -------------------------

```

```sql
DROP TABLE IF EXISTS `ods_sku_info`;
CREATE TABLE `ods_sku_info` (
  `id` BIGINT,
  `spu_id` BIGINT,
  `price` DECIMAL(10,0),
  `sku_name` STRING,
  `sku_desc` STRING,
  `weight` DECIMAL(10,2),
  `tm_id` BIGINT,
  `category3_id` BIGINT,
  `sku_default_img` STRING,
  `create_time` TIMESTAMP(0)
) WITH(
 'connector' = 'kafka',
 'topic' = 'mydw.sku_info',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- -------------------------
--   商品表
--   MySQL Sink
-- -------------------------
DROP TABLE IF EXISTS `sku_info`;
CREATE TABLE `sku_info` (
  `id` BIGINT,
  `spu_id` BIGINT,
  `price` DECIMAL(10,0),
  `sku_name` STRING,
  `sku_desc` STRING,
  `weight` DECIMAL(10,2),
  `tm_id` BIGINT,
  `category3_id` BIGINT,
  `sku_default_img` STRING,
  `create_time` TIMESTAMP(0),
   PRIMARY KEY (tm_id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'sku_info', -- MySQL中的待插入数据的表
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'sink.buffer-flush.interval' = '1s'
);

-- -------------------------
--   商品
--   MySQL Sink Load Data
-- -------------------------
INSERT INTO sku_info
SELECT *
FROM ods_sku_info;

```

经过上面的步骤，我们可以将创建商品维表的基础数据表同步到MySQL中，同样需要提前创建好对应的数据表。接下来我们使用上面的基础表在mySQL的dim库中创建一张视图：`dim_sku_info`，用作后续使用的维表。

```sql
-- ---------------------------------
-- DIM层,商品维表,
-- 在MySQL中创建视图
-- ---------------------------------
CREATE VIEW dim_sku_info AS
SELECT
  si.id AS id,
  si.sku_name AS sku_name,
  si.category3_id AS c3_id,
  si.weight AS weight,
  si.tm_id AS tm_id,
  si.price AS price,
  si.spu_id AS spu_id,
  c3.name AS c3_name,
  c2.id AS c2_id,
  c2.name AS c2_name,
  c3.id AS c1_id,
  c3.name AS c1_name
FROM
(
  sku_info si
  JOIN base_category3 c3 ON si.category3_id = c3.id
  JOIN base_category2 c2 ON c3.category2_id =c2.id
  JOIN base_category1 c1 ON c2.category1_id = c1.id
);
```

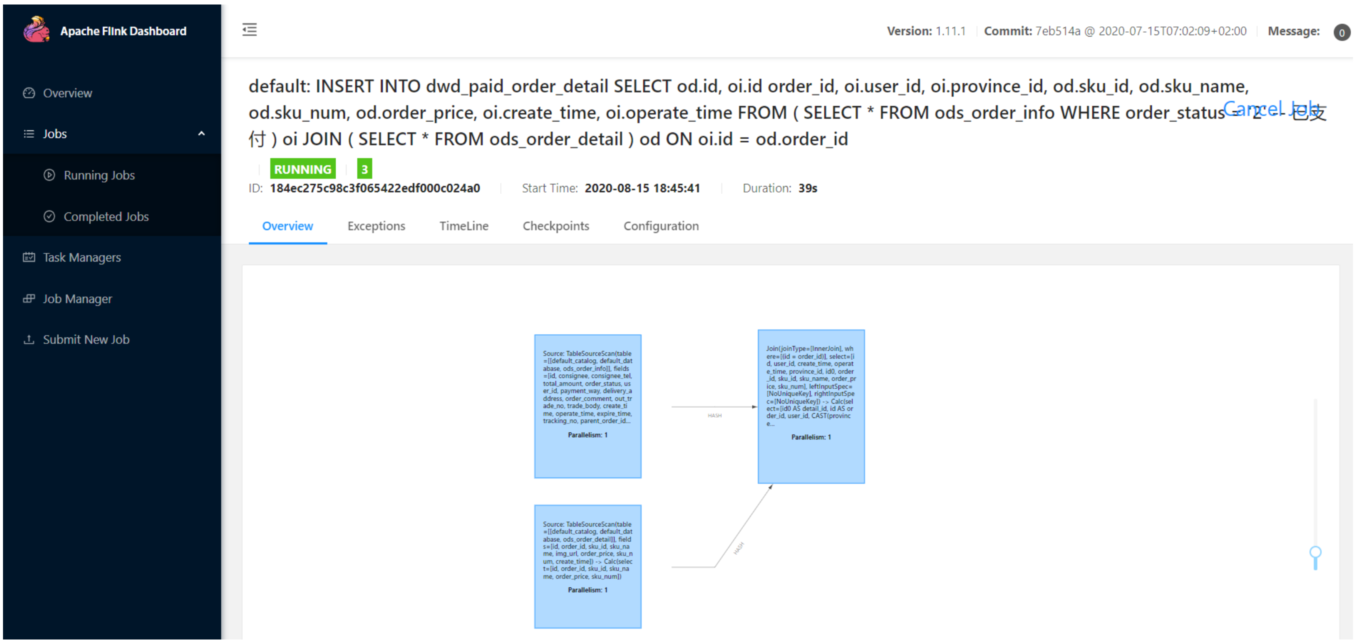
至此，我们所需要的维表数据已经准备好了，接下来开始处理DWD层的数据。

DWD层数据处理

经过上面的步骤，我们已经将所用的维表已经准备好了。接下来我们将对ODS的原始数据进行处理，加工成DWD层的明细宽表。具体过程如下：

```sql
-- -------------------------
--   订单详情
--   Kafka Source
-- -------------------------

DROP TABLE IF EXISTS `ods_order_detail`;
CREATE TABLE `ods_order_detail`(
  `id` BIGINT,
  `order_id` BIGINT,
  `sku_id` BIGINT,
  `sku_name` STRING,
  `img_url` STRING,
  `order_price` DECIMAL(10,2),
  `sku_num` INT,
  `create_time` TIMESTAMP(0)
) WITH(
 'connector' = 'kafka',
 'topic' = 'mydw.order_detail',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- -------------------------
--   订单信息
--   Kafka Source
-- -------------------------
DROP TABLE IF EXISTS `ods_order_info`;
CREATE TABLE `ods_order_info` (
  `id` BIGINT,
  `consignee` STRING,
  `consignee_tel` STRING,
  `total_amount` DECIMAL(10,2),
  `order_status` STRING,
  `user_id` BIGINT,
  `payment_way` STRING,
  `delivery_address` STRING,
  `order_comment` STRING,
  `out_trade_no` STRING,
  `trade_body` STRING,
  `create_time` TIMESTAMP(0) ,
  `operate_time` TIMESTAMP(0) ,
  `expire_time` TIMESTAMP(0) ,
  `tracking_no` STRING,
  `parent_order_id` BIGINT,
  `img_url` STRING,
  `province_id` INT
) WITH(
'connector' = 'kafka',
 'topic' = 'mydw.order_info',
 'properties.bootstrap.servers' = 'kms-3:9092',
 'properties.group.id' = 'testGroup',
 'format' = 'canal-json' ,
 'scan.startup.mode' = 'earliest-offset'
) ;

-- ---------------------------------
-- DWD层,支付订单明细表dwd_paid_order_detail
-- ---------------------------------
DROP TABLE IF EXISTS dwd_paid_order_detail;
CREATE TABLE dwd_paid_order_detail
(
  detail_id BIGINT,
  order_id BIGINT,
  user_id BIGINT,
  province_id INT,
  sku_id BIGINT,
  sku_name STRING,
  sku_num INT,
  order_price DECIMAL(10,0),
  create_time TIMESTAMP(0),
  pay_time TIMESTAMP(0)
 ) WITH (
    'connector' = 'kafka',
    'topic' = 'dwd_paid_order_detail',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);
-- ---------------------------------
-- DWD层,已支付订单明细表
-- 向dwd_paid_order_detail装载数据
-- ---------------------------------
INSERT INTO dwd_paid_order_detail
SELECT
  od.id,
  oi.id order_id,
  oi.user_id,
  oi.province_id,
  od.sku_id,
  od.sku_name,
  od.sku_num,
  od.order_price,
  oi.create_time,
  oi.operate_time
FROM
```

```
    (
    SELECT *
    FROM ods_order_info
    WHERE order_status = '2' -- 已支付
    ) oi JOIN
    (
    SELECT *
    FROM ods_order_detail
    ) od
    ON oi.id = od.order_id;
```


Apache Flink Dashboard
Overview
Jobs
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit New Job
Version: 1.11.1
Commit: 7eb514a @ 2020-07-15T07:02:09+02:00
Message: 0
Cancel Job
RUNNING
3
Overview
Exceptions
TimeLine
Checkpoints
Configuration
Source: TableSourceScan(table=[[default_catalog, default_database, ods_order_info]], fields=[id, consignee, consignee_tel, total_amount, order_status, user_id, payment_way, delivery_address, order_comment, out_trade_no, trade_body, create_time, operate_time, expire_time, tracking_no, parent_order_id...
Parallelism: 1
HASH
Join(joinType=[InnerJoin], where=[(id = order_id)], select=[id, user_id, create_time, operate_time, province_id, id0, order_id, sku_id, sku_name, order_price, sku_num], leftInputSpec=[NoUniqueKey], rightInputSpec=[NoUniqueKey]) -> Calc(select=[id0 AS detail_id, id AS order_id, user_id, CAST(province...
Parallelism: 1
Source: TableSourceScan(table=[[default_catalog, default_database, ods_order_detail]], fields=[id, order_id, sku_id, sku_name, img_url, order_price, sku_num, create_time]) -> Calc(select=[id, order_id, sku_id, sku_name, order_price, sku_num])
Parallelism: 1
HASH

default: INSERT INTO dwd_paid_order_detail SELECT od.id, oi.id order_id, oi.user_id, oi.province_id, od.sku_id, od.sku_name, od.sku_num, od.order_price, oi.create_time, oi.operate_time FROM (SELECT * FROM ods_order_info WHERE order_status = '2' -- 已支付) oi JOIN (SELECT * FROM ods_order_detail) od ON oi.id = od.order_id

ID: 184ec275c98c3f065422edf000c024a0 Start Time: 2020-08-15 18:45:41 Duration: 39s

ADS层数据

经过上面的步骤，我们创建了一张dwd_paid_order_detail明细宽表，并将该表存储在了Kafka中。接下来我们将使用这张明细宽表与维表进行JOIN，得到我们ADS应用层数据。

- **ads_province_index**

首先在MySQL中创建对应的ADS目标表：**ads_province_index**

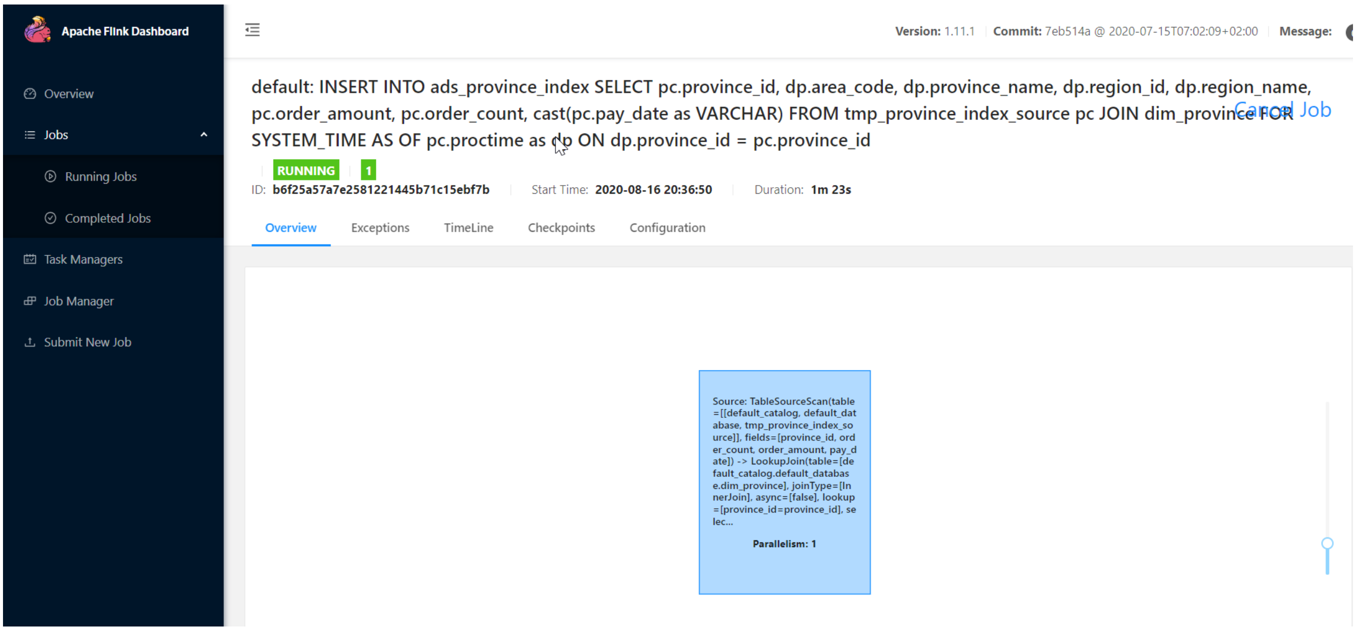
```
CREATE TABLE ads.ads_province_index(
  province_id INT(10),
  area_code VARCHAR(100),
  province_name VARCHAR(100),
  region_id INT(10),
  region_name VARCHAR(100),
  order_amount DECIMAL(10,2),

  order_count BIGINT(10),
  dt VARCHAR(100),
  PRIMARY KEY (province_id, dt)
) ;
```

向MySQL的ADS层目标装载数据：

```
-- Flink SQL Cli操作
-- ---------------------------------
-- 使用 DDL创建MySQL中的ADS层表
-- 指标：1.每天每个省份的订单数
--      2.每天每个省份的订单金额
-- ---------------------------------
CREATE TABLE ads_province_index(
  province_id INT,
  area_code STRING,
  province_name STRING,
  region_id INT,
  region_name STRING,
  order_amount DECIMAL(10,2),
  order_count BIGINT,
  dt STRING,
  PRIMARY KEY (province_id, dt) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/ads',
    'table-name' = 'ads_province_index',
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe'
);
-- ---------------------------------
-- dwd_paid_order_detail已支付订单明细宽表
-- ---------------------------------
CREATE TABLE dwd_paid_order_detail
(
  detail_id BIGINT,
  order_id BIGINT,
  user_id BIGINT,
  province_id INT,
  sku_id BIGINT,
  sku_name STRING,
  sku_num INT,
  order_price DECIMAL(10,2),
  create_time STRING,
  pay_time STRING
 ) WITH (

```

```sql
    'connector' = 'kafka',
    'topic' = 'dwd_paid_order_detail',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);

-- ---------------------------------
-- tmp_province_index
-- 订单汇总临时表
-- ---------------------------------
CREATE TABLE tmp_province_index(
    province_id INT,
    order_count BIGINT,-- 订单数
    order_amount DECIMAL(10,2), -- 订单金额
    pay_date DATE
)WITH (
    'connector' = 'kafka',
    'topic' = 'tmp_province_index',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);
-- ---------------------------------
-- tmp_province_index
-- 订单汇总临时表数据装载
-- ---------------------------------
INSERT INTO tmp_province_index
SELECT
      province_id,
      count(distinct order_id) order_count,-- 订单数
      sum(order_price * sku_num) order_amount, -- 订单金额
      TO_DATE(pay_time,'yyyy-MM-dd') pay_date
FROM dwd_paid_order_detail
GROUP BY province_id,TO_DATE(pay_time,'yyyy-MM-dd')
;
-- ---------------------------------
-- tmp_province_index_source
-- 使用该临时汇总表，作为数据源
-- ---------------------------------
CREATE TABLE tmp_province_index_source(
    province_id INT,
    order_count BIGINT,-- 订单数
    order_amount DECIMAL(10,2), -- 订单金额
    pay_date DATE,
    proctime as PROCTIME()   -- 通过计算列产生一个处理时间列
 ) WITH (
    'connector' = 'kafka',
    'topic' = 'tmp_province_index',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);

-- ---------------------------------
-- DIM层,区域维表,
-- 创建区域维表数据源
-- ---------------------------------
DROP TABLE IF EXISTS `dim_province`;
CREATE TABLE dim_province (
  province_id INT,
  province_name STRING,
  area_code STRING,
  region_id INT,
  region_name STRING ,
  PRIMARY KEY (province_id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'dim_province',
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'scan.fetch-size' = '100'
);

-- ---------------------------------
-- 向ads_province_index装载数据
-- 维表JOIN
-- ---------------------------------

INSERT INTO ads_province_index
SELECT
  pc.province_id,
  dp.area_code,
  dp.province_name,
  dp.region_id,
  dp.region_name,
  pc.order_amount,
  pc.order_count,
  cast(pc.pay_date as VARCHAR)
FROM
tmp_province_index_source pc
  JOIN dim_province FOR SYSTEM_TIME AS OF pc.proctime as dp
  ON dp.province_id = pc.province_id;
```

当提交任务之后：观察Flink WEB UI：



Version: 1.11.1 | Commit: 7eb514a @ 2020-07-15T07:02:09+02:00 | Message:

default: INSERT INTO ads_province_index SELECT pc.province_id, dp.area_code, dp.province_name, dp.region_id, dp.region_name, pc.order_amount, pc.order_count, cast(pc.pay_date as VARCHAR) FROM tmp_province_index_source pc JOIN dim_province FOR SYSTEM_TIME AS OF pc.proctime as dp ON dp.province_id = pc.province_id

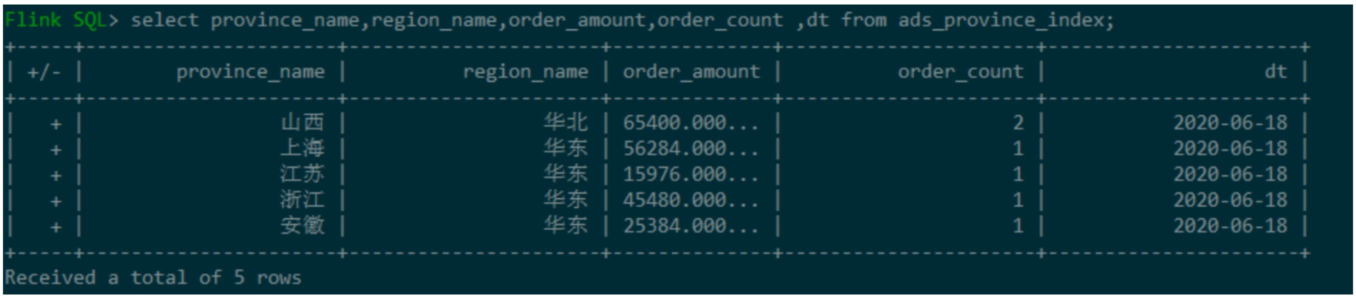
RUNNING | 1

ID: b6f25a57a7e2581221445b71c15ebf7b | Start Time: 2020-08-16 20:36:50 | Duration: 1m 23s

Source: TableSourceScan(table=[[default_catalog, default_database, tmp_province_index_source]], fields=[province_id, order_count, order_amount, pay_date]) -> LookupJoin(table=[default_catalog.default_database.dim_province], joinType=[InnerJoin], async=[false], lookup=[province_id=province_id], selec...

Parallelism: 1

查看ADS层的ads_province_index表数据：



```
Flink SQL> select province_name,region_name,order_amount,order_count ,dt from ads_province_index;
+-----+---------------------+---------------------+----------+--------------------+--------------------+
| +/- |        province_name |          region_name | order_amount |        order_count |                 dt |
+-----+---------------------+---------------------+----------+--------------------+--------------------+
|   + |                 山西 |                 华北 | 65400.000... |                  2 |         2020-06-18 |
|   + |                 上海 |                 华东 | 56284.000... |                  1 |         2020-06-18 |
|   + |                 江苏 |                 华东 | 15976.000... |                  1 |         2020-06-18 |
|   + |                 浙江 |                 华东 | 45480.000... |                  1 |         2020-06-18 |
|   + |                 安徽 |                 华东 | 25384.000... |                  1 |         2020-06-18 |
+-----+---------------------+---------------------+----------+--------------------+--------------------+
Received a total of 5 rows
```

- **ads_sku_index**

首先在MySQL中创建对应的ADS目标表：**ads_sku_index**

```
CREATE TABLE ads_sku_index
(
  sku_id BIGINT(10),
  sku_name VARCHAR(100),
  weight DOUBLE,
  tm_id BIGINT(10),
  price DOUBLE,
  spu_id BIGINT(10),
  c3_id BIGINT(10),
  c3_name VARCHAR(100) ,
  c2_id BIGINT(10),
  c2_name VARCHAR(100),
  c1_id BIGINT(10),
  c1_name VARCHAR(100),
  order_amount DOUBLE,
  order_count BIGINT(10),
  sku_count BIGINT(10),
  dt varchar(100),
  PRIMARY KEY (sku_id,dt)
);
```

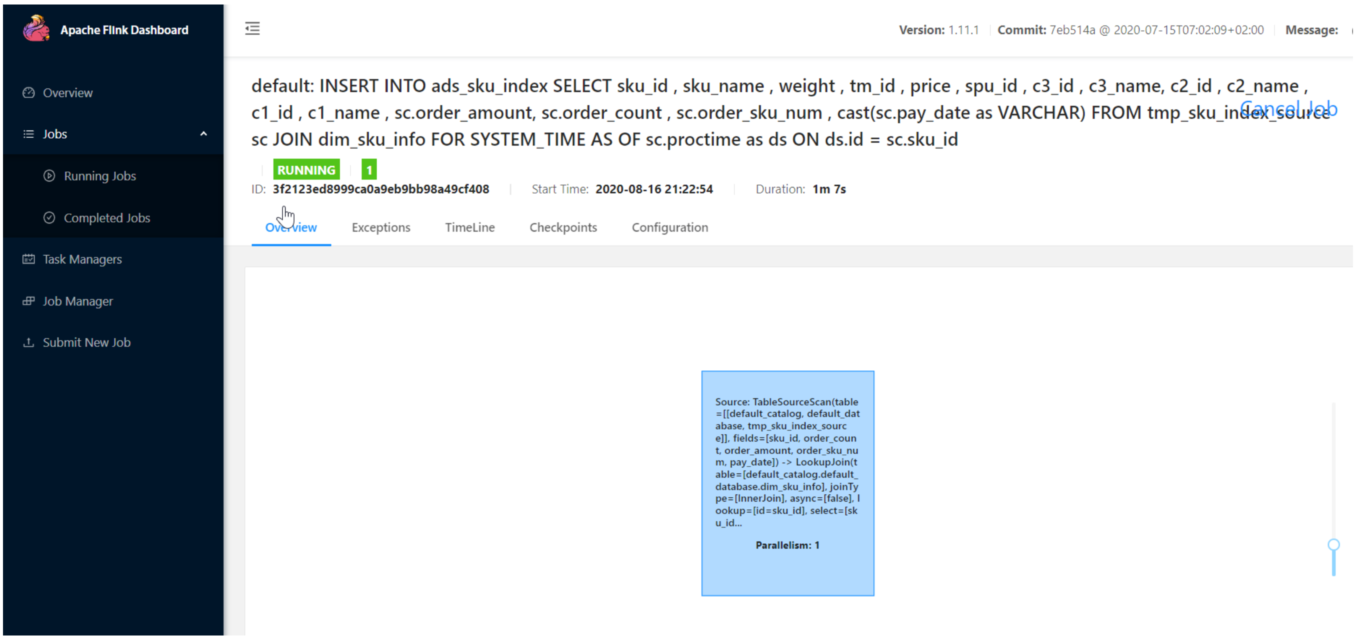
向MySQL的ADS层目标装载数据：

```
-- ---------------------------------
-- 使用 DDL创建MySQL中的ADS层表
-- 指标：1.每天每个商品对应的订单个数
--       2.每天每个商品对应的订单金额
--       3.每天每个商品对应的数量
-- ---------------------------------
CREATE TABLE ads_sku_index
(
  sku_id BIGINT,
  sku_name VARCHAR,
  weight DOUBLE,
  tm_id BIGINT,
  price DOUBLE,
  spu_id BIGINT,
  c3_id BIGINT,
  c3_name VARCHAR ,
  c2_id BIGINT,
  c2_name VARCHAR,
  c1_id BIGINT,
  c1_name VARCHAR,
  order_amount DOUBLE,
  order_count BIGINT,
  sku_count BIGINT,
  dt varchar,
  PRIMARY KEY (sku_id,dt) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/ads',
    'table-name' = 'ads_sku_index',
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe'
);

-- ---------------------------------
-- dwd_paid_order_detail已支付订单明细宽表
-- ---------------------------------
CREATE TABLE dwd_paid_order_detail

```

```sql
(
  detail_id BIGINT,
  order_id BIGINT,
  user_id BIGINT,
  province_id INT,
  sku_id BIGINT,
  sku_name STRING,
  sku_num INT,
  order_price DECIMAL(10,2),
  create_time STRING,
  pay_time STRING
 ) WITH (
    'connector' = 'kafka',
    'topic' = 'dwd_paid_order_detail',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);

-- ---------------------------------
-- tmp_sku_index
-- 商品指标统计
-- ---------------------------------
CREATE TABLE tmp_sku_index(
    sku_id BIGINT,
    order_count BIGINT,-- 订单数
    order_amount DECIMAL(10,2), -- 订单金额
        order_sku_num BIGINT,
    pay_date DATE
)WITH (
    'connector' = 'kafka',
    'topic' = 'tmp_sku_index',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);
-- ---------------------------------
-- tmp_sku_index
-- 数据装载
-- ---------------------------------
INSERT INTO tmp_sku_index
SELECT
      sku_id,
      count(distinct order_id) order_count,-- 订单数
      sum(order_price * sku_num) order_amount, -- 订单金额
          sum(sku_num) order_sku_num,
      TO_DATE(pay_time,'yyyy-MM-dd') pay_date
FROM dwd_paid_order_detail
GROUP BY sku_id,TO_DATE(pay_time,'yyyy-MM-dd')
```

```sql
;

-- ---------------------------------
-- tmp_sku_index_source
-- 使用该临时汇总表，作为数据源
-- ---------------------------------
CREATE TABLE tmp_sku_index_source(
    sku_id BIGINT,
    order_count BIGINT,-- 订单数
    order_amount DECIMAL(10,2), -- 订单金额
    order_sku_num BIGINT,
    pay_date DATE,
    proctime as PROCTIME()   -- 通过计算列产生一个处理时间列
 ) WITH (
    'connector' = 'kafka',
    'topic' = 'tmp_sku_index',
    'scan.startup.mode' = 'earliest-offset',
    'properties.bootstrap.servers' = 'kms-3:9092',
    'format' = 'changelog-json'
);
-- ---------------------------------
-- DIM层,商品维表,
-- 创建商品维表数据源
-- ---------------------------------
DROP TABLE IF EXISTS `dim_sku_info`;
CREATE TABLE dim_sku_info (
  id BIGINT,
  sku_name STRING,
  c3_id BIGINT,
  weight DECIMAL(10,2),
  tm_id BIGINT,
  price DECIMAL(10,2),
  spu_id BIGINT,
  c3_name STRING,
  c2_id BIGINT,
  c2_name STRING,
  c1_id BIGINT,
  c1_name STRING,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://kms-1:3306/dim',
    'table-name' = 'dim_sku_info',
    'driver' = 'com.mysql.jdbc.Driver',
    'username' = 'root',
    'password' = '123qwe',
    'scan.fetch-size' = '100'
);
-- ---------------------------------
-- 向ads_sku_index装载数据
-- 维表JOIN
-- ---------------------------------
INSERT INTO ads_sku_index
SELECT
  sku_id ,
  sku_name ,
  weight ,
  tm_id ,
  price ,
  spu_id ,
  c3_id ,
  c3_name,
  c2_id ,
  c2_name ,
  c1_id ,
  c1_name ,
  sc.order_amount,
  sc.order_count ,
  sc.order_sku_num ,
  cast(sc.pay_date as VARCHAR)
FROM
tmp_sku_index_source sc
  JOIN dim_sku_info FOR SYSTEM_TIME AS OF sc.proctime as ds
  ON ds.id = sc.sku_id
  ;
```

当提交任务之后：观察Flink WEB UI：



Version: 1.11.1 | Commit: 7eb514a @ 2020-07-15T07:02:09+02:00 | Message:

default: INSERT INTO ads_sku_index SELECT sku_id , sku_name , weight , tm_id , price , spu_id , c3_id , c3_name, c2_id , c2_name , c1_id , c1_name , sc.order_amount, sc.order_count , sc.order_sku_num , cast(sc.pay_date as VARCHAR) FROM tmp_sku_index_source sc JOIN dim_sku_info FOR SYSTEM_TIME AS OF sc.proctime as ds ON ds.id = sc.sku_id

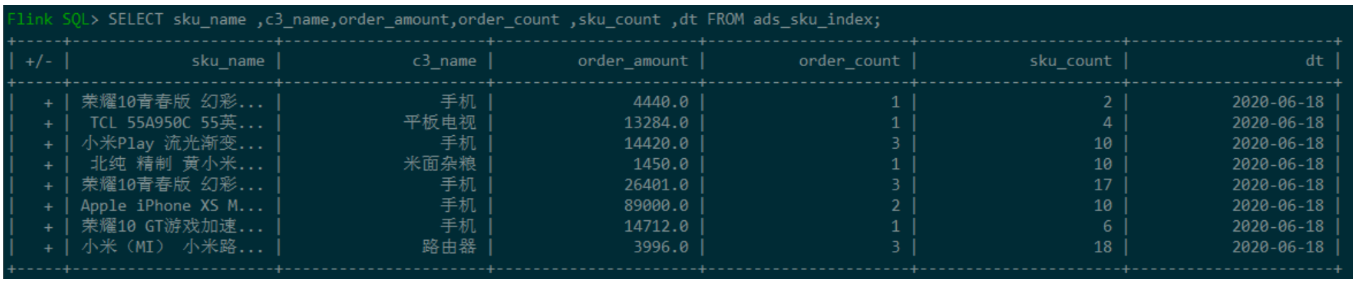
RUNNING 1

ID: 3f2123ed8999ca0a9eb9bb98a49cf408 | Start Time: 2020-08-16 21:22:54 | Duration: 1m 7s

Source: TableSourceScan(table=[[default_catalog, default_database, tmp_sku_index_source]], fields=[sku_id, order_count, order_amount, order_sku_num, pay_date]) -> LookupJoin(table=[default_catalog.default_database.dim_sku_info], joinType=[InnerJoin], async=[false], lookup=[id=sku_id], select=[sku_id...

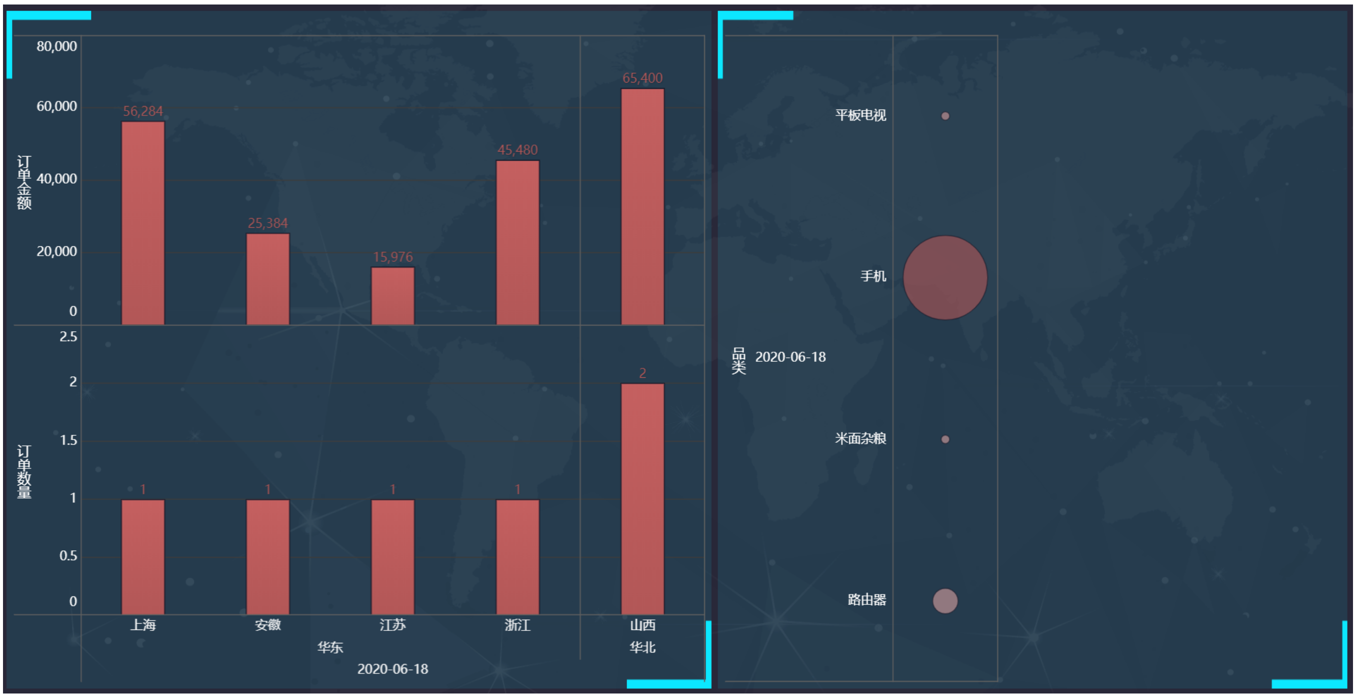
Parallelism: 1

查看ADS层的ads_sku_index表数据：



```
Flink SQL> SELECT sku_name ,c3_name,order_amount,order_count ,sku_count ,dt FROM ads_sku_index;
+-----+----------------------+----------------------+----------------------+----------------------+----------------------+----------------------+
| +/- |             sku_name |              c3_name |         order_amount |          order_count |            sku_count |                   dt |
+-----+----------------------+----------------------+----------------------+----------------------+----------------------+----------------------+
|   + | 荣耀10青春版 幻彩... |                 手机 |               4440.0 |                    1 |                    2 |           2020-06-18 |
|   + |  TCL 55A950C 55英... |             平板电视 |              13284.0 |                    1 |                    4 |           2020-06-18 |
|   + | 小米Play 流光渐变... |                 手机 |              14420.0 |                    3 |                   10 |           2020-06-18 |
|   + |  北纯 精制 黄小米... |             米面杂粮 |               1450.0 |                    1 |                   10 |           2020-06-18 |
|   + | 荣耀10青春版 幻彩... |                 手机 |              26401.0 |                    3 |                   17 |           2020-06-18 |
|   + | Apple iPhone XS M... |                 手机 |              89000.0 |                    2 |                   10 |           2020-06-18 |
|   + | 荣耀10 GT游戏加速... |                 手机 |              14712.0 |                    1 |                    6 |           2020-06-18 |
|   + | 小米（MI） 小米路... |               路由器 |               3996.0 |                    3 |                   18 |           2020-06-18 |
+-----+----------------------+----------------------+----------------------+----------------------+----------------------+----------------------+
```

FineBI结果展示



其他注意点

Flink1.11.0存在的bug

当在代码中使用Flink1.11.0版本时，如果将一个change-log的数据源insert到一个upsert sink时，会报如下异常：

```
[ERROR] Could not execute SQL statement. Reason:
org.apache.flink.table.api.TableException: Provided trait [BEFORE_AND_AFTER] can't satisfy required trait [ONLY_UPDATE_AFTER]. This is a bug in planner, please file an issue.
Current node is TableSourceScan(table=[[default_catalog, default_database, t_pick_order]], fields=[order_no, status])
```

该bug目前已被修复，修复可以在Flink1.11.1中使用。

总结

本文主要分享了构建一个实时数仓的demo案例，通过本文可以了解实时数仓的数据处理流程，在此基础之上，对Flink SQL的CDC会有更加深刻的认识。另外，本文给出了非常详细的使用案例，你可以直接上手进行操作，在实践中探索实时数仓的构建流程。