

# 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

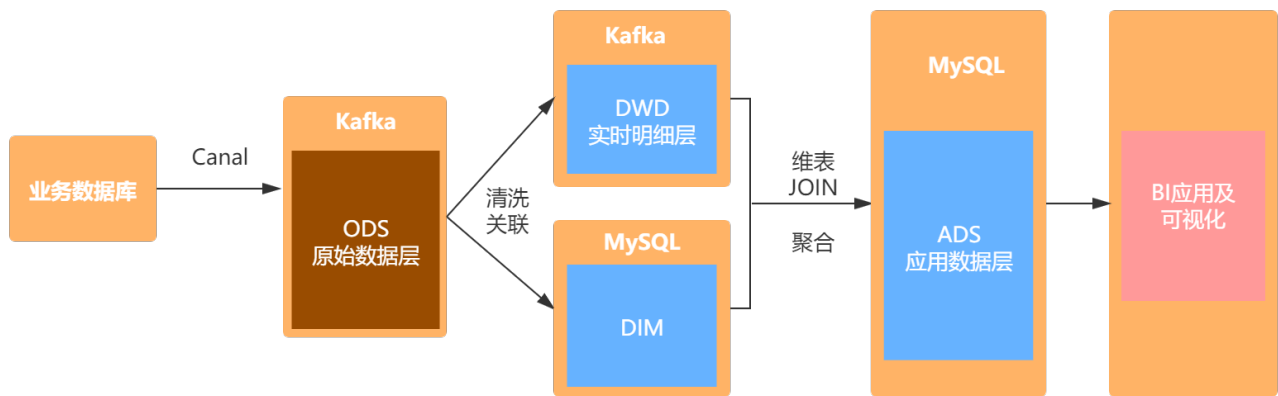
古人学问无遗力，少壮工夫老始成。  
纸上得来终觉浅，绝知此事要躬行。

## 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)

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

- 订单详情表 (order\_detail)

```
1 CREATE TABLE `order_detail` (  
2   `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',  
3
```

```

4  `order_id` bigint(20) DEFAULT NULL COMMENT '订单编号',
5  `sku_id` bigint(20) DEFAULT NULL COMMENT 'sku_id',
6  `sku_name` varchar(200) DEFAULT NULL COMMENT 'sku名称 (冗余)',
7  `img_url` varchar(200) DEFAULT NULL COMMENT '图片名称 (冗余)',
8  `order_price` decimal(10,2) DEFAULT NULL COMMENT '购买价格(下单时sku价格)
9  ',
10 `sku_num` varchar(200) DEFAULT NULL COMMENT '购买个数',
11 `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)

```

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

```

- 商品一级类目表 (base\_category1)

```

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

```

- 商品二级类目表 (base\_category2)

```

1  CREATE TABLE `base_category2` (
2  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '编号',
3  `name` varchar(200) NOT NULL COMMENT '二级分类名称',
4  `category1_id` bigint(20) DEFAULT NULL COMMENT '一级分类编号',
5

```

```
6 | 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)

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

- 省份表 (base\_province)

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

- 区域表 (base\_region)

```
1 | CREATE TABLE `base_region` (  
2 |   `id` int(20) NOT NULL COMMENT '大区id',  
3 |   `region_name` varchar(20) DEFAULT NULL COMMENT '大区名称',  
4 |   PRIMARY KEY (`id`)  
5 | ) 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：

```
1  -- -----
2  --  省份
3  --  kafka Source
4  -- -----
5  DROP TABLE IF EXISTS `ods_base_province`;
6  CREATE TABLE `ods_base_province` (
7    `id` INT,
8    `name` STRING,
9    `region_id` INT ,
10   `area_code` STRING
11  ) WITH(
12    'connector' = 'kafka',
13    'topic' = 'mydw.base_province',
14    'properties.bootstrap.servers' = 'kms-3:9092',
15    'properties.group.id' = 'testGroup',
16    'format' = 'canal-json' ,
17    'scan.startup.mode' = 'earliest-offset'
18  );
19
20
```

```

21  -- -----
22  -- 省份
23  -- MySQL Sink
24  -- -----
25  DROP TABLE IF EXISTS `base_province`;
26  CREATE TABLE `base_province` (
27      `id` INT,
28      `name` STRING,
29      `region_id` INT ,
30      `area_code` STRING,
31      PRIMARY KEY (id) NOT ENFORCED
32  ) WITH (
33      'connector' = 'jdbc',
34      'url' = 'jdbc:mysql://kms-1:3306/dim',
35      'table-name' = 'base_province', -- MySQL 中的待插入数据的表
36      'driver' = 'com.mysql.jdbc.Driver',
37      'username' = 'root',
38      'password' = '123qwe',
39      'sink.buffer-flush.interval' = '1s'
40  );
41
42  -- -----
43  -- 省份
44  -- MySQL Sink Load Data
45  -- -----
46  INSERT INTO base_province
47  SELECT *
48  FROM ods_base_province;
49
50  -- -----
51  -- 区域
52  -- kafka Source
53  -- -----
54  DROP TABLE IF EXISTS `ods_base_region`;
55  CREATE TABLE `ods_base_region` (
56      `id` INT,
57      `region_name` STRING
58  ) WITH(
59      'connector' = 'kafka',
60      'topic' = 'mydw.base_region',
61      'properties.bootstrap.servers' = 'kms-3:9092',
62      'properties.group.id' = 'testGroup',
63      'format' = 'canal-json' ,
64      'scan.startup.mode' = 'earliest-offset'
65  ) ;
66
67  -- -----
68  -- 区域
69  -- MySQL Sink

```

```

70  -- -----
71  DROP TABLE IF EXISTS `base_region`;
72  CREATE TABLE `base_region` (
73      `id` INT,
74      `region_name` STRING,
75      PRIMARY KEY (id) NOT ENFORCED
76  ) WITH (
77      'connector' = 'jdbc',
78      'url' = 'jdbc:mysql://kms-1:3306/dim',
79      'table-name' = 'base_region', -- MySQL 中的待插入数据的表
80      'driver' = 'com.mysql.jdbc.Driver',
81      'username' = 'root',
82      'password' = '123qwe',
83      'sink.buffer-flush.interval' = '1s'
84  );
85
86  -- -----
87  -- 区域
88  -- MySQL Sink Load Data
89  -- -----
90  INSERT INTO base_region
91  SELECT *
92  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：

```

1  -- -----
2  -- DIM层, 区域维表,
3  -- 在MySQL 中创建视图
4  -- -----
5  DROP VIEW IF EXISTS dim_province;
6  CREATE VIEW dim_province AS
7  SELECT
8      bp.id AS province_id,
9      bp.name AS province_name,
10     br.id AS region_id,
11     br.region_name AS region_name,
12     bp.area_code AS area_code
13  FROM base_region br
14       JOIN base_province bp ON br.id= bp.region_id
15  ;

```

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：

```
1  -- -----
2  -- 一级类目表
3  --  kafka Source
4  -- -----
5  DROP TABLE IF EXISTS `ods_base_category1`;
6  CREATE TABLE `ods_base_category1` (
7      `id` BIGINT,
8      `name` STRING
9  )WITH(
10     'connector' = 'kafka',
11     'topic' = 'mydw.base_category1',
12     'properties.bootstrap.servers' = 'kms-3:9092',
13     'properties.group.id' = 'testGroup',
14     'format' = 'canal-json' ,
15     'scan.startup.mode' = 'earliest-offset'
16 );
17
18  -- -----
19  -- 一级类目表
20  --  MySQL Sink
21  -- -----
22  DROP TABLE IF EXISTS `base_category1`;
23  CREATE TABLE `base_category1` (
24      `id` BIGINT,
25      `name` STRING,
26      PRIMARY KEY (id) NOT ENFORCED
27  ) WITH (
28      'connector' = 'jdbc',
29      'url' = 'jdbc:mysql://kms-1:3306/dim',
30      'table-name' = 'base_category1', -- MySQL 中的待插入数据的表
31      'driver' = 'com.mysql.jdbc.Driver',
32      'username' = 'root',
33      'password' = '123qwe',
34      'sink.buffer-flush.interval' = '1s'
35  );
36
37  -- -----
38  -- 一级类目表
39  --  MySQL Sink Load Data
40  -- -----
41
42  INSERT INTO base_category1
43  SELECT *
44  FROM ods_base_category1;
```



```

45
46 -- -----
47 -- 二级类目表
48 -- kafka Source
49 -- -----
50 DROP TABLE IF EXISTS `ods_base_category2`;
51 CREATE TABLE `ods_base_category2` (
52     `id` BIGINT,
53     `name` STRING,
54     `category1_id` BIGINT
55 )WITH(
56     'connector' = 'kafka',
57     'topic' = 'mydw.base_category2',
58     'properties.bootstrap.servers' = 'kms-3:9092',
59     'properties.group.id' = 'testGroup',
60     'format' = 'canal-json' ,
61     'scan.startup.mode' = 'earliest-offset'
62 );
63
64 -- -----
65 -- 二级类目表
66 -- MySQL Sink
67 -- -----
68 DROP TABLE IF EXISTS `base_category2`;
69 CREATE TABLE `base_category2` (
70     `id` BIGINT,
71     `name` STRING,
72     `category1_id` BIGINT,
73     PRIMARY KEY (id) NOT ENFORCED
74 ) WITH (
75     'connector' = 'jdbc',
76     'url' = 'jdbc:mysql://kms-1:3306/dim',
77     'table-name' = 'base_category2', -- MySQL 中的待插入数据的表
78     'driver' = 'com.mysql.jdbc.Driver',
79     'username' = 'root',
80     'password' = '123qwe',
81     'sink.buffer-flush.interval' = '1s'
82 );
83
84 -- -----
85 -- 二级类目表
86 -- MySQL Sink Load Data
87 -- -----
88 INSERT INTO base_category2
89 SELECT *
90 FROM ods_base_category2;
91
92 -- -----
93 -- 三级类目表

```

```

94  -- kafka Source
95  -- -----
96  DROP TABLE IF EXISTS `ods_base_category3`;
97  CREATE TABLE `ods_base_category3` (
98      `id` BIGINT,
99      `name` STRING,
100     `category2_id` BIGINT
101 )WITH(
102     'connector' = 'kafka',
103     'topic' = 'mydw.base_category3',
104     'properties.bootstrap.servers' = 'kms-3:9092',
105     'properties.group.id' = 'testGroup',
106     'format' = 'canal-json' ,
107     'scan.startup.mode' = 'earliest-offset'
108 );
109
110 -- -----
111 -- 三级类目表
112 -- MySQL Sink
113 -- -----
114 DROP TABLE IF EXISTS `base_category3`;
115 CREATE TABLE `base_category3` (
116     `id` BIGINT,
117     `name` STRING,
118     `category2_id` BIGINT,
119     PRIMARY KEY (id) NOT ENFORCED
120 ) WITH (
121     'connector' = 'jdbc',
122     'url' = 'jdbc:mysql://kms-1:3306/dim',
123     'table-name' = 'base_category3', -- MySQL中的待插入数据的表
124     'driver' = 'com.mysql.jdbc.Driver',
125     'username' = 'root',
126     'password' = '123qwe',
127     'sink.buffer-flush.interval' = '1s'
128 );
129
130 -- -----
131 -- 三级类目表
132 -- MySQL Sink Load Data
133 -- -----
134 INSERT INTO base_category3
135 SELECT *
136 FROM ods_base_category3;
137
138 -- -----
139 -- 商品表
140 -- Kafka Source
141 -- -----
142

```

```

143 DROP TABLE IF EXISTS `ods_sku_info`;
144 CREATE TABLE `ods_sku_info` (
145     `id` BIGINT,
146     `spu_id` BIGINT,
147     `price` DECIMAL(10,0),
148     `sku_name` STRING,
149     `sku_desc` STRING,
150     `weight` DECIMAL(10,2),
151     `tm_id` BIGINT,
152     `category3_id` BIGINT,
153     `sku_default_img` STRING,
154     `create_time` TIMESTAMP(0)
155 ) WITH(
156     'connector' = 'kafka',
157     'topic' = 'mydw.sku_info',
158     'properties.bootstrap.servers' = 'kms-3:9092',
159     'properties.group.id' = 'testGroup',
160     'format' = 'canal-json' ,
161     'scan.startup.mode' = 'earliest-offset'
162 );
163
164 -- -----
165 -- 商品表
166 -- MySQL Sink
167 -- -----
168 DROP TABLE IF EXISTS `sku_info`;
169 CREATE TABLE `sku_info` (
170     `id` BIGINT,
171     `spu_id` BIGINT,
172     `price` DECIMAL(10,0),
173     `sku_name` STRING,
174     `sku_desc` STRING,
175     `weight` DECIMAL(10,2),
176     `tm_id` BIGINT,
177     `category3_id` BIGINT,
178     `sku_default_img` STRING,
179     `create_time` TIMESTAMP(0),
180     PRIMARY KEY (tm_id) NOT ENFORCED
181 ) WITH (
182     'connector' = 'jdbc',
183     'url' = 'jdbc:mysql://kms-1:3306/dim',
184     'table-name' = 'sku_info', -- MySQL 中的待插入数据的表
185     'driver' = 'com.mysql.jdbc.Driver',
186     'username' = 'root',
187     'password' = '123qwe',
188     'sink.buffer-flush.interval' = '1s'
189 );
190
191 -- -----

```

```

192 -- 商品
193 -- MySQL Sink Load Data
194 -- -----
195 INSERT INTO sku_info
196 SELECT *
197 FROM ods_sku_info;
198

```

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。

```

1  -- -----
2  -- DIM层, 商品维表,
3  -- 在MySQL 中创建视图
4  -- -----
5  CREATE VIEW dim_sku_info AS
6  SELECT
7      si.id AS id,
8      si.sku_name AS sku_name,
9      si.category3_id AS c3_id,
10     si.weight AS weight,
11     si.tm_id AS tm_id,
12     si.price AS price,
13     si.spu_id AS spu_id,
14     c3.name AS c3_name,
15     c2.id AS c2_id,
16     c2.name AS c2_name,
17     c3.id AS c1_id,
18     c3.name AS c1_name
19 FROM
20 (
21     sku_info si
22     JOIN base_category3 c3 ON si.category3_id = c3.id
23     JOIN base_category2 c2 ON c3.category2_id = c2.id
24     JOIN base_category1 c1 ON c2.category1_id = c1.id
25 );

```

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：

```

1  -- -----
2  -- 订单详情
3  --  Kafka Source
4  -- -----
5
6  DROP TABLE IF EXISTS `ods_order_detail`;
7  CREATE TABLE `ods_order_detail` (
8      `id` BIGINT,
9      `order_id` BIGINT,
10     `sku_id` BIGINT,
11     `sku_name` STRING,
12     `img_url` STRING,
13     `order_price` DECIMAL(10,2),
14     `sku_num` INT,
15     `create_time` TIMESTAMP(0)
16 ) WITH(
17     'connector' = 'kafka',
18     'topic' = 'mydw.order_detail',
19     'properties.bootstrap.servers' = 'kms-3:9092',
20     'properties.group.id' = 'testGroup',
21     'format' = 'canal-json' ,
22     'scan.startup.mode' = 'earliest-offset'
23 );
24
25 -- -----
26 -- 订单信息
27 --  Kafka Source
28 -- -----
29 DROP TABLE IF EXISTS `ods_order_info`;
30 CREATE TABLE `ods_order_info` (
31     `id` BIGINT,
32     `consignee` STRING,
33     `consignee_tel` STRING,
34     `total_amount` DECIMAL(10,2),
35     `order_status` STRING,
36     `user_id` BIGINT,
37     `payment_way` STRING,
38     `delivery_address` STRING,
39     `order_comment` STRING,
40     `out_trade_no` STRING,
41     `trade_body` STRING,
42     `create_time` TIMESTAMP(0) ,
43     `operate_time` TIMESTAMP(0) ,
44     `expire_time` TIMESTAMP(0) ,
45     `tracking_no` STRING,
46     `parent_order_id` BIGINT,
47     `img_url` STRING,
48     `province_id` INT

```

```

49 ) WITH(
50 'connector' = 'kafka',
51 'topic' = 'mydw.order_info',
52 'properties.bootstrap.servers' = 'kms-3:9092',
53 'properties.group.id' = 'testGroup',
54 'format' = 'canal-json' ,
55 'scan.startup.mode' = 'earliest-offset'
56 ) ;
57
58 -- -----
59 -- DWD层, 支付订单明细表dwd_paid_order_detail
60 -- -----
61 DROP TABLE IF EXISTS dwd_paid_order_detail;
62 CREATE TABLE dwd_paid_order_detail
63 (
64     detail_id BIGINT,
65     order_id BIGINT,
66     user_id BIGINT,
67     province_id INT,
68     sku_id BIGINT,
69     sku_name STRING,
70     sku_num INT,
71     order_price DECIMAL(10,0),
72     create_time TIMESTAMP(0),
73     pay_time TIMESTAMP(0)
74 ) WITH (
75     'connector' = 'kafka',
76     'topic' = 'dwd_paid_order_detail',
77     'scan.startup.mode' = 'earliest-offset',
78     'properties.bootstrap.servers' = 'kms-3:9092',
79     'format' = 'changelog-json'
80 );
81 -- -----
82 -- DWD层, 已支付订单明细表
83 -- 向dwd_paid_order_detail装载数据
84 -- -----
85 INSERT INTO dwd_paid_order_detail
86 SELECT
87     od.id,
88     oi.id order_id,
89     oi.user_id,
90     oi.province_id,
91     od.sku_id,
92     od.sku_name,
93     od.sku_num,
94     od.order_price,
95     oi.create_time,
96     oi.operate_time
97 FROM

```

```

98      (
99      SELECT *
100     FROM ods_order_info
101     WHERE order_status = '2' -- 已支付
102     ) oi JOIN
103     (
104     SELECT *
105     FROM ods_order_detail
106     ) od
107     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

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' ) 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

Overview Exceptions TimeLine Checkpoints Configuration

## 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**

```

1 CREATE TABLE ads.ads_province_index(
2     province_id INT(10),
3     area_code VARCHAR(100),
4     province_name VARCHAR(100),
5     region_id INT(10),
6     region_name VARCHAR(100),
7     order_amount DECIMAL(10,2),
8

```

```

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

```

向MySQL的ADS层目标装载数据:

```

1  -- Flink SQL Cli操作
2
3  -- -----
4  -- 使用 DDL 创建MySQL 中的ADS层表
5  -- 指标: 1. 每天每个省份的订单数
6  --        2. 每天每个省份的订单金额
7  -- -----
8  CREATE TABLE ads_province_index(
9      province_id INT,
10     area_code STRING,
11     province_name STRING,
12     region_id INT,
13     region_name STRING,
14     order_amount DECIMAL(10,2),
15     order_count BIGINT,
16     dt STRING,
17     PRIMARY KEY (province_id, dt) NOT ENFORCED
18 ) WITH (
19     'connector' = 'jdbc',
20     'url' = 'jdbc:mysql://kms-1:3306/ads',
21     'table-name' = 'ads_province_index',
22     'driver' = 'com.mysql.jdbc.Driver',
23     'username' = 'root',
24     'password' = '123qwe'
25 );
26 -- -----
27 -- dwd_paid_order_detail已支付订单明细宽表
28 -- -----
29 CREATE TABLE dwd_paid_order_detail
30 (
31     detail_id BIGINT,
32     order_id BIGINT,
33     user_id BIGINT,
34     province_id INT,
35     sku_id BIGINT,
36     sku_name STRING,
37     sku_num INT,
38     order_price DECIMAL(10,2),
39     create_time STRING,
40     pay_time STRING
41 ) WITH (

```



```

42     'connector' = 'kafka',
43     'topic' = 'dwd_paid_order_detail',
44     'scan.startup.mode' = 'earliest-offset',
45     'properties.bootstrap.servers' = 'kms-3:9092',
46     'format' = 'changelog-json'
47 );
48
49 -- -----
50 -- tmp_province_index
51 -- 订单汇总临时表
52 -- -----
53 CREATE TABLE tmp_province_index(
54     province_id INT,
55     order_count BIGINT, -- 订单数
56     order_amount DECIMAL(10,2), -- 订单金额
57     pay_date DATE
58 )WITH (
59     'connector' = 'kafka',
60     'topic' = 'tmp_province_index',
61     'scan.startup.mode' = 'earliest-offset',
62     'properties.bootstrap.servers' = 'kms-3:9092',
63     'format' = 'changelog-json'
64 );
65 -- -----
66 -- tmp_province_index
67 -- 订单汇总临时表数据装载
68 -- -----
69 INSERT INTO tmp_province_index
70 SELECT
71     province_id,
72     count(distinct order_id) order_count, -- 订单数
73     sum(order_price * sku_num) order_amount, -- 订单金额
74     TO_DATE(pay_time, 'yyyy-MM-dd') pay_date
75 FROM dwd_paid_order_detail
76 GROUP BY province_id, TO_DATE(pay_time, 'yyyy-MM-dd')
77 ;
78 -- -----
79 -- tmp_province_index_source
80 -- 使用该临时汇总表，作为数据源
81 -- -----
82 CREATE TABLE tmp_province_index_source(
83     province_id INT,
84     order_count BIGINT, -- 订单数
85     order_amount DECIMAL(10,2), -- 订单金额
86     pay_date DATE,
87     proctime as PROCTIME() -- 通过计算列产生一个处理时间列
88 ) WITH (
89     'connector' = 'kafka',
90     'topic' = 'tmp_province_index',

```

```

91     'scan.startup.mode' = 'earliest-offset',
92     'properties.bootstrap.servers' = 'kms-3:9092',
93     'format' = 'changelog-json'
94 );
95
96 -- -----
97 -- DIM层, 区域维表,
98 -- 创建区域维表数据源
99 -- -----
100 DROP TABLE IF EXISTS `dim_province`;
101 CREATE TABLE dim_province (
102     province_id INT,
103     province_name STRING,
104     area_code STRING,
105     region_id INT,
106     region_name STRING ,
107     PRIMARY KEY (province_id) NOT ENFORCED
108 ) WITH (
109     'connector' = 'jdbc',
110     'url' = 'jdbc:mysql://kms-1:3306/dim',
111     'table-name' = 'dim_province',
112     'driver' = 'com.mysql.jdbc.Driver',
113     'username' = 'root',
114     'password' = '123qwe',
115     'scan.fetch-size' = '100'
116 );
117
118 -- -----
119 -- 向ads_province_index装载数据
120 -- 维表JOIN
121 -- -----
122
123 INSERT INTO ads_province_index
124 SELECT
125     pc.province_id,
126     dp.area_code,
127     dp.province_name,
128     dp.region_id,
129     dp.region_name,
130     pc.order_amount,
131     pc.order_count,
132     cast(pc.pay_date as VARCHAR)
133 FROM
134 tmp_province_index_source pc
135 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：

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:

```
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

Overview

Exceptions

TimeLine

Checkpoints

Configuration

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**

```

1 CREATE TABLE ads_sku_index
2 (
3     sku_id BIGINT(10),
4     sku_name VARCHAR(100),
5     weight DOUBLE,
6     tm_id BIGINT(10),
7     price DOUBLE,
8     spu_id BIGINT(10),
9     c3_id BIGINT(10),
10    c3_name VARCHAR(100) ,
11    c2_id BIGINT(10),
12    c2_name VARCHAR(100),
13    c1_id BIGINT(10),
14    c1_name VARCHAR(100),

```

```

15     order_amount DOUBLE,
16     order_count BIGINT(10),
17     sku_count BIGINT(10),
18     dt varchar(100),
19     PRIMARY KEY (sku_id,dt)
20 );

```

向MySQL的ADS层目标装载数据：

```

1  -- -----
2  -- 使用 DDL 创建MySQL中的ADS层表
3  -- 指标: 1. 每天每个商品对应的订单个数
4  --        2. 每天每个商品对应的订单金额
5  --        3. 每天每个商品对应的数量
6  -- -----
7  CREATE TABLE ads_sku_index
8  (
9      sku_id BIGINT,
10     sku_name VARCHAR,
11     weight DOUBLE,
12     tm_id BIGINT,
13     price DOUBLE,
14     spu_id BIGINT,
15     c3_id BIGINT,
16     c3_name VARCHAR ,
17     c2_id BIGINT,
18     c2_name VARCHAR,
19     c1_id BIGINT,
20     c1_name VARCHAR,
21     order_amount DOUBLE,
22     order_count BIGINT,
23     sku_count BIGINT,
24     dt varchar,
25     PRIMARY KEY (sku_id,dt) NOT ENFORCED
26 ) WITH (
27     'connector' = 'jdbc',
28     'url' = 'jdbc:mysql://kms-1:3306/ads',
29     'table-name' = 'ads_sku_index',
30     'driver' = 'com.mysql.jdbc.Driver',
31     'username' = 'root',
32     'password' = '123qwe'
33 );
34
35 -- -----
36 -- dwd_paid_order_detail已支付订单明细宽表
37 -- -----
38 CREATE TABLE dwd_paid_order_detail
39

```

```
40  (
41    detail_id BIGINT,
42    order_id BIGINT,
43    user_id BIGINT,
44    province_id INT,
45    sku_id BIGINT,
46    sku_name STRING,
47    sku_num INT,
48    order_price DECIMAL(10,2),
49    create_time STRING,
50    pay_time STRING
51  ) WITH (
52    'connector' = 'kafka',
53    'topic' = 'dwd_paid_order_detail',
54    'scan.startup.mode' = 'earliest-offset',
55    'properties.bootstrap.servers' = 'kms-3:9092',
56    'format' = 'changelog-json'
57  );
58
59  -----
60  -- tmp_sku_index
61  -- 商品指标统计
62  -----
63  CREATE TABLE tmp_sku_index(
64    sku_id BIGINT,
65    order_count BIGINT, -- 订单数
66    order_amount DECIMAL(10,2), -- 订单金额
67    order_sku_num BIGINT,
68    pay_date DATE
69  ) WITH (
70    'connector' = 'kafka',
71    'topic' = 'tmp_sku_index',
72    'scan.startup.mode' = 'earliest-offset',
73    'properties.bootstrap.servers' = 'kms-3:9092',
74    'format' = 'changelog-json'
75  );
76  -----
77  -- tmp_sku_index
78  -- 数据装载
79  -----
80  INSERT INTO tmp_sku_index
81  SELECT
82    sku_id,
83    count(distinct order_id) order_count, -- 订单数
84    sum(order_price * sku_num) order_amount, -- 订单金额
85    sum(sku_num) order_sku_num,
86    TO_DATE(pay_time, 'yyyy-MM-dd') pay_date
87  FROM dwd_paid_order_detail
88  GROUP BY sku_id, TO_DATE(pay_time, 'yyyy-MM-dd')
```

```

89 ;
90
91 -- -----
92 -- tmp_sku_index_source
93 -- 使用该临时汇总表, 作为数据源
94 -- -----
95 CREATE TABLE tmp_sku_index_source(
96     sku_id BIGINT,
97     order_count BIGINT, -- 订单数
98     order_amount DECIMAL(10,2), -- 订单金额
99     order_sku_num BIGINT,
100    pay_date DATE,
101    proctime as PROCTIME() -- 通过计算列产生一个处理时间列
102 ) WITH (
103     'connector' = 'kafka',
104     'topic' = 'tmp_sku_index',
105     'scan.startup.mode' = 'earliest-offset',
106     'properties.bootstrap.servers' = 'kms-3:9092',
107     'format' = 'changelog-json'
108 );
109 -- -----
110 -- DIM层, 商品维表,
111 -- 创建商品维表数据源
112 -- -----
113 DROP TABLE IF EXISTS `dim_sku_info`;
114 CREATE TABLE dim_sku_info (
115     id BIGINT,
116     sku_name STRING,
117     c3_id BIGINT,
118     weight DECIMAL(10,2),
119     tm_id BIGINT,
120     price DECIMAL(10,2),
121     spu_id BIGINT,
122     c3_name STRING,
123     c2_id BIGINT,
124     c2_name STRING,
125     c1_id BIGINT,
126     c1_name STRING,
127     PRIMARY KEY (id) NOT ENFORCED
128 ) WITH (
129     'connector' = 'jdbc',
130     'url' = 'jdbc:mysql://kms-1:3306/dim',
131     'table-name' = 'dim_sku_info',
132     'driver' = 'com.mysql.jdbc.Driver',
133     'username' = 'root',
134     'password' = '123qwe',
135     'scan.fetch-size' = '100'
136 );
137 -- -----

```

```

138 -- 向ads_sku_index装载数据
139 -- 维表JOIN
140 -- -----
141 INSERT INTO ads_sku_index
142 SELECT
143     sku_id ,
144     sku_name ,
145     weight ,
146     tm_id ,
147     price ,
148     spu_id ,
149     c3_id ,
150     c3_name,
151     c2_id ,
152     c2_name ,
153     c1_id ,
154     c1_name ,
155     sc.order_amount,
156     sc.order_count ,
157     sc.order_sku_num ,
158     cast(sc.pay_date as VARCHAR)
159 FROM
160 tmp_sku_index_source sc
161 JOIN dim_sku_info FOR SYSTEM_TIME AS OF sc.proctime as ds
162 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

[Overview](#) | Exceptions | TimeLine | Checkpoints | Configuration

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...])

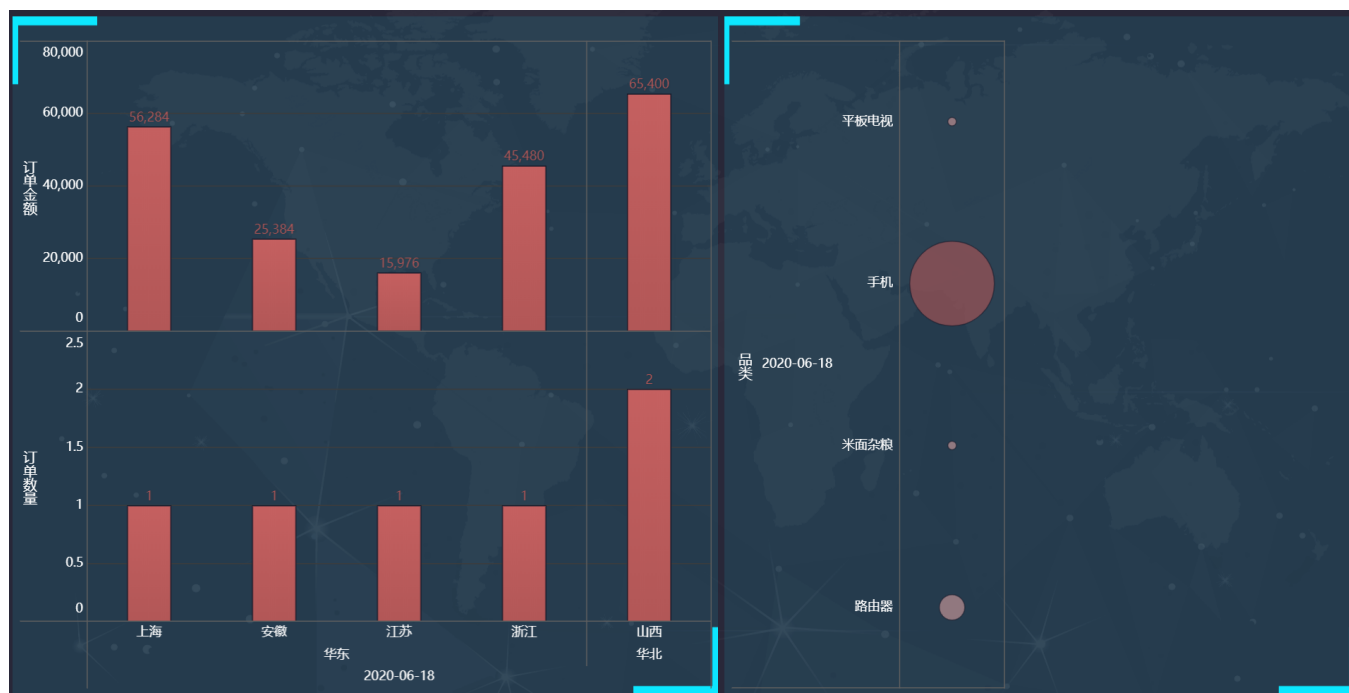
Parallelism: 1

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 幻影...	手机	4440.0	1	2	2020-06-18
+	TCL 55A950C 55英寸...	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 幻影...	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) 小米路由...	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：

```
1 [ERROR] Could not execute SQL statement. Reason:
2 org.apache.flink.table.api.TableException: Provided trait [BEFORE_AND_AFT
3 ER] can't satisfy required trait [ONLY_UPDATE_AFTER]. This is a bug in p
lanner, please file an issue.
Current node is TableSourceScan(table=[[default_catalog, default_databas
e, 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。