Flink x Zeppelin ， Hive Streaming 实战解析

发布于: 2020 年 08 月 04 日

Flink 1.11 正式发布已经三周了，其中最吸引我的特性就是 Hive Streaming。正巧 Zeppelin-0.9-preview2 也在前不久发布了，所以就写了一篇 Zeppelin 上的 Flink Hive Streaming 的实战解析。本文主要从以下几部分跟大家分享：

Hive Streaming 的意义
Checkpoint & Dependency
写入 Kafka
Hive Streaming Sink
Hive Streaming Source
Hive Temporal Table

Hive Streaming 的意义

很多同学可能会好奇，为什么 Flink 1.11 中，Hive Streaming 的地位这么高？它的出现，到底能给我们带来什么？

其实在大数据领域，一直存在两种架构 **Lambda** 和 **Kappa**：

Lambda 架构——流批分离，静态数据通过定时调度同步到 Hive 数仓，实时数据既会同步到 Hive，也会被实时计算引擎消费，这里就引出了一点问题。
数据口径问题
离线计算产出延时太大
数据冗余存储
Kappa 架构——全部使用实时计算来产出数据，历史数据通过回溯消息的消费位点计算，同样也有很多的问题，毕竟没有一劳永逸的架构。
消息中间件无法保留全部历史数据，同样数据都是行式存储，占用空间太大

实时计算计算历史数据力不从心
无法进行 Ad-Hoc 的分析

为了解决这些问题，行业内推出了实时数仓，解决了大部分痛点，但是还是有些地方力不从心。比如涉及到历史数据的计算怎么办？我想做 Ad-Hoc 的分析又怎么玩？所以行业内现在都是实时数仓与离线数仓并行存在，而这又带来了更多的问题：模型需要多份、数据产出不一致、历史数据的计算等等 。

而 Hive Streaming 的出现就可以解决这些问题！再也不用多套模型了；也不需要同一个指标因为涉及到历史数据，写一遍实时 SQL 再写一遍离线 SQL；Ad-Hoc 也能做了，怎么做？读 Hive Streaming 产出的表就行！

接下来，让我们从参数配置开始，接着流式的写入 Hive，再到流式的读取 Hive 表，最后再 Join 上 Hive 维表吧。这一整套流程都体验后，想必大家对 Hive Streaming 一定会有更深入的了解，更能够体会到它的作用。

Checkpoint & Dependency

因为只有在完成 Checkpoint 之后，文件才会从 In-progress 状态变成 Finish 状态，所以，我们需要合理的去配置 Checkpoint，在 Zeppelin 中配置 Checkpoint 很简单。

```
%flink.conf

# checkpoint 配置

pipeline.time-characteristic EventTime
execution.checkpointing.interval 120000
execution.checkpointing.min-pause 60000
execution.checkpointing.timeout 60000
execution.checkpointing.externalized-checkpoint-retention RETAIN_ON_CANCELLATION

# 依赖jar包配置

flink.execution.packages org.apache.flink:flink-connector-kafka_2.11:1.11.0,org.a
```

又因为我们需要从 Kafka 中读取数据，所以将 Kafka 的依赖也加入进去了。

写入Kafka

我们的数据来自于天池数据集，是以 CSV 的格式存在于本地磁盘，所以需要先将他们写入 Kafka。

先建一下 CSV Source 和 Kafka Sink 的表：

```
%flink.ssql
SET table.sql-dialect=default;
DROP TABLE IF EXISTS source_csv;
CREATE TABLE source_csv (
user_id string,
theme_id string,
item_id string,
leaf_cate_id string,
cate_level1_id string,
clk_cnt int,
reach_time string
) WITH (
 'connector' = 'filesystem',
 'path' = 'file:///Users/dijie/Downloads/Cloud_Theme_Click/theme_click_log.csv',
 'format' = 'csv'

 )
```

```
%flink.ssql
SET table.sql-dialect=default;
DROP TABLE IF EXISTS kafka_table;
CREATE TABLE kafka_table (
user_id string,
theme_id string,
item_id string,
leaf_cate_id string,
cate_level1_id string,
clk_cnt int,
reach_time string,
ts AS localtimestamp,
WATERMARK FOR ts AS ts - INTERVAL '5' SECOND
) WITH (
'connector' = 'kafka',
'topic' = 'theme_click_log',
'properties.bootstrap.servers' = '10.70.98.1:9092',
```

```
    'properties.group.id' = 'testGroup',
    'format' = 'json',
    'scan.startup.mode' = 'latest-offset'

    )
```

因为注册的表即可以读又可以写，于是我在建表时将 Watermark 加上了；又因为源数据中的时间戳已经很老了，所以我这里采用当前时间减去5秒作为我的 Watermark。

大家可以看到，我在语句一开始指定了 SQL 方言为 Default，这是为啥呢？还有别的方言吗？别急，听我慢慢说。

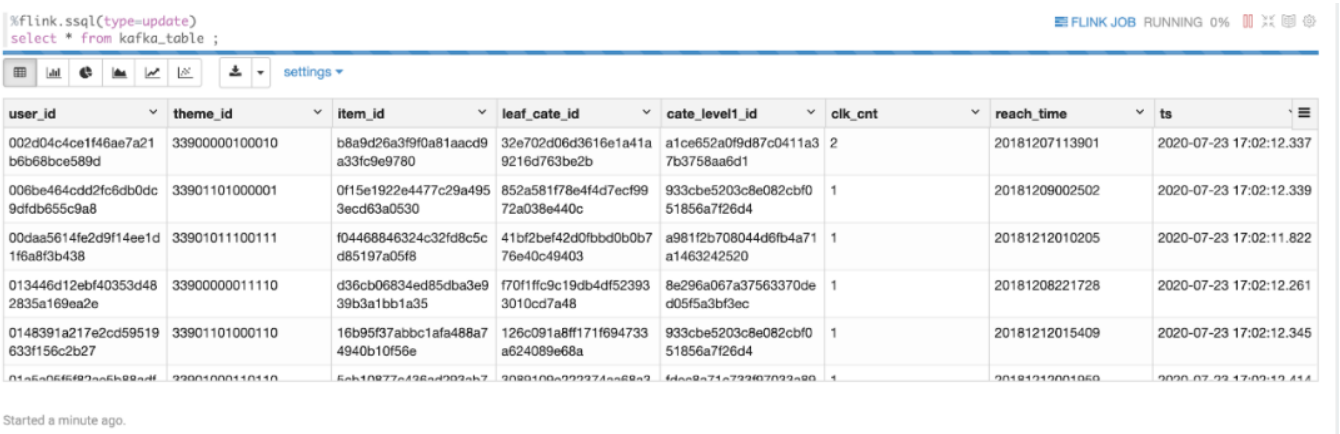
其实在之前的版本，Flink 就已经可以和 Hive 打通，包括可以把表建在 Hive 上，但是很多语法和 Hive 不兼容，包括建的表在 Hive 中也无法查看，主要原因就是方言不兼容。所以，在 Flink 1.11 中，为了减少学习成本（语法不兼容），可以用 DDL 建 Hive 表并在 Hive 中查询，Flink 支持了方言，默认的就是 Default 了，就和之前一样，如果想建 Hive 表，并支持查询，请使用 Hive 方言，具体可以参考下方链接。

> **Hive 方言：**
> https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/hive/hive_catalog.html

再把数据从 CSV 中读取后写入 Kafka。

```
%flink.ssql(type=update)

insert into kafka_table select * from source_csv ;
```



看来没问题，那么接下来让我们写入 Hive。

Hive Streaming Sink

建一个Hive Sink Table，记得将方言切换到 Hive，否则会有问题。

```
%flink.ssql
SET table.sql-dialect=hive;
DROP TABLE IF EXISTS hive_table;
CREATE TABLE hive_table (
user_id string,
theme_id string,
item_id string,
leaf_cate_id string,
cate_level1_id string,
clk_cnt int,
reach_time string
) PARTITIONED BY (dt string, hr string, mi string) STORED AS parquet TBLPROPERTIE

 'partition.time-extractor.timestamp-pattern'='$dt $hr:$mi:00',
 'sink.partition-commit.trigger'='partition-time',
 'sink.partition-commit.delay'='1 min',
 'sink.partition-commit.policy.kind'='metastore,success-file'

);
```

参数给大家稍微解释一下：

partition.time-extractor.timestamp-pattern：分区时间抽取器，与 DDL 中的分区字段保持一致；

sink.partition-commit.trigger：分区触发器类型，可选 process-time 或partition-time。process-time：不需要上面的参数，也不需要水印，当当前时间大于分区创建时间 +sink.partition-commit.delay 中定义的时间，提交分区；partition-time：需要 Source 表中定义 watermark，当 watermark > 提取到的分区时间 +sink.partition-commit.delay 中定义的时间，提交分区；

sink.partition-commit.delay：相当于延时时间；

sink.partition-commit.policy.kind：怎么提交，一般提交成功之后，需要通知 metastore，这样 Hive 才能读到你最新分区的数据；如果需要合并小文件，也可以自定义 Class，通过实现 PartitionCommitPolicy 接口。

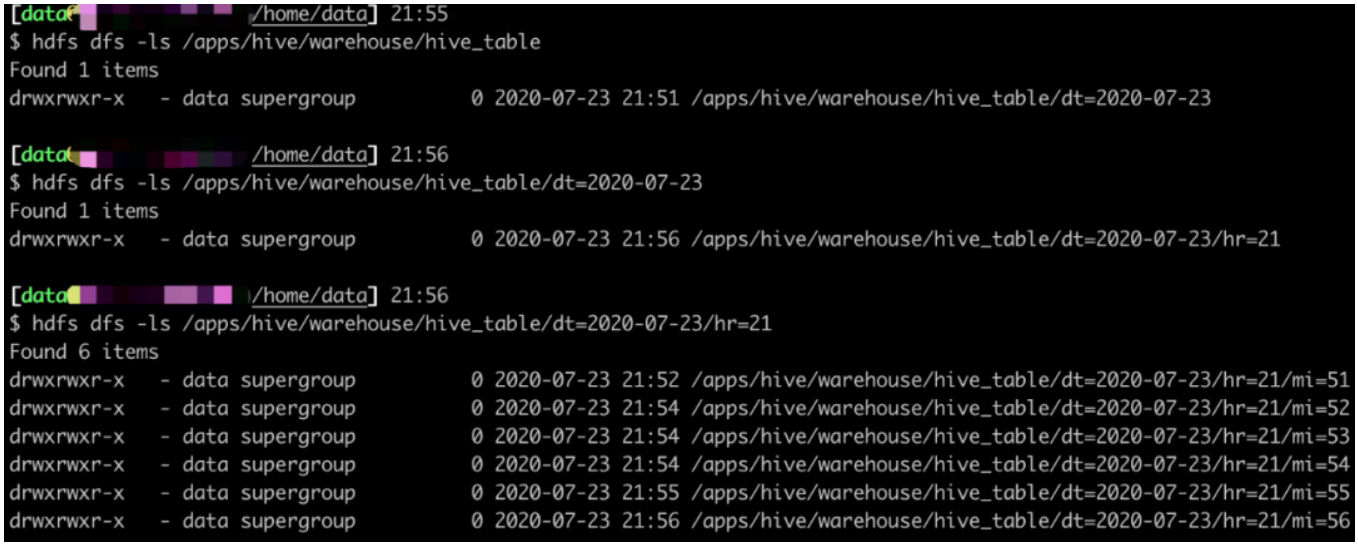
接下来让我们把数据插入刚刚创建的 Hive Table：

```
%flink.ssql

insert into hive_table select  user_id,theme_id,item_id,leaf_cate_id,cate_level1_
```

让程序再跑一会儿~我们先去倒一杯 95 年的 Java☕ 。

然后再看看我们的 HDFS，看看路径下的东西。



大家也可以用 Hive 自行查询看看，我呢就先卖个关子，一会儿用 Hive Streaming 来读数据。

Hive Streaming Source

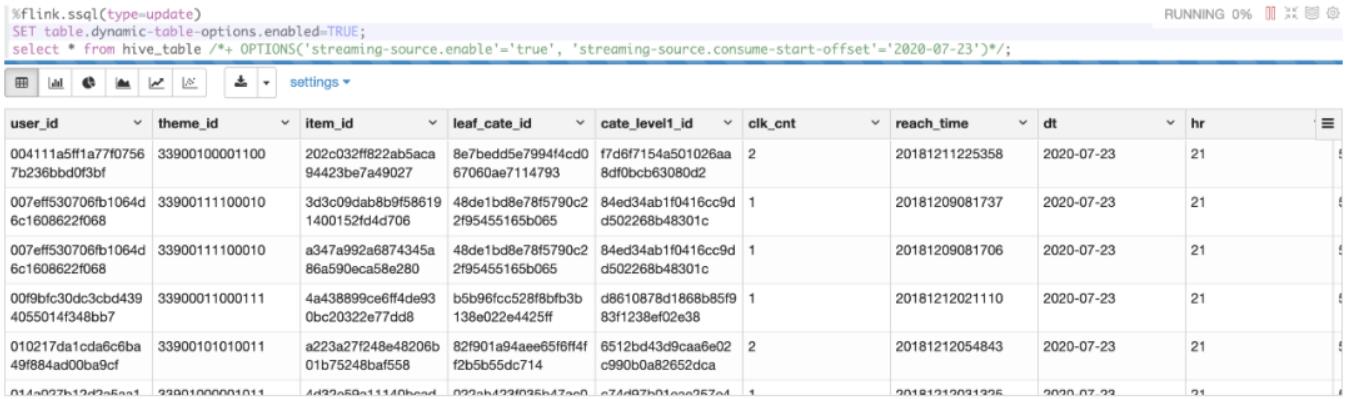
因为 Hive 表上面已经创建过了，所以这边读数据的时候直接拿来用就行了，不同的地方是需要使用 Table Hints 去覆盖参数。

Hive Streaming Source 最大的不足是，无法读取已经读取过的分区下新增的文件。简单来说就是，读过的分区，就不会再读了。看似很坑，不过仔细想想，这样才符合流的特性。

照旧给大家说一下参数的意思：

stream-source.enable：显而易见，表示是否开启流模式。
stream-source.monitor-interval：监控新文件/分区产生的间隔。
stream-source.consume-order：可以选 create-time 或者 partition-time；create-time 指的不是分区创建时间，而是在 HDFS 中文件/文件夹的创建时间；partition-time 指的是分区的时间；对于非分区表，只能用 create-time。官网这边的介绍写的有点模糊，会让人误以为可以查到已经读过的分区下新增的文件，其实经过我的测试和翻看源码发现并不能。
stream-source.consume-start-offset：表示从哪个分区开始读。

光说不干假把式，让我们捞一把数据看看~



```
%flink.ssql(type=update)
SET table.dynamic-table-options.enabled=TRUE;
select * from hive_table /*+ OPTIONS('streaming-source.enable'='true', 'streaming-source.consume-start-offset'='2020-07-23')*/;
```

RUNNING 0%

settings

user_id	theme_id	item_id	leaf_cate_id	cate_level1_id	clk_cnt	reach_time	dt	hr
004111a5ff1a77f07567b236bbd0f3bf	33900100001100	202c032ff822ab5aca94423be7a49027	8e7bedd5e7994f4cd067060ae7114793	f7d6f7154a501026aa8df0bcb63080d2	2	20181211225358	2020-07-23	21
007eff530706fb1064d6c1608622f068	33900111100010	3d3c09dab8b9f586191400152fd4d706	48de1bd8e78f5790c22f95455165b065	84ed34ab1f0416cc9dd502268b48301c	1	20181209081737	2020-07-23	21
007eff530706fb1064d6c1608622f068	33900111100010	a347a992a6874345a86a590eca58e280	48de1bd8e78f5790c22f95455165b065	84ed34ab1f0416cc9dd502268b48301c	1	20181209081706	2020-07-23	21
00f9bfc30dc3cbd4394055014f348bb7	33900011000111	4a438899ce6ff4de930bc20322e77dd8	b5b96fcc528f8bfb3b138e022e4425ff	d8610878d1868b85f983f1238ef02e38	1	20181212021110	2020-07-23	21
010217da1cda6c6ba49f884ad00ba9cf	33900101010011	a223a27f248e48206b01b75248baf558	82f901a94aee65f6ff4ff2b5b55dc714	6512bd43d9caa6e02c990b0a82652dca	2	20181212054843	2020-07-23	21

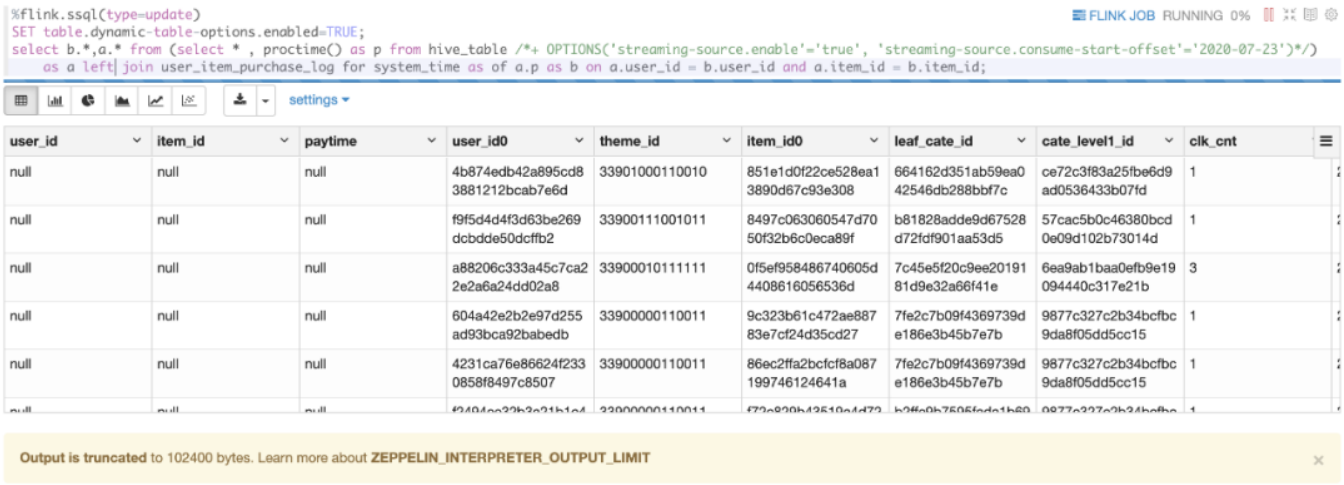
SET 那一行得带着，不然无法使用 Table Hints。

Hive Temporal Table

看完了 Streaming Source 和 Streaming Sink，让我们最后再试一下 Hive 作为维表吧。

其实用 Hive 维表很简单，只要是在 Hive 中存在的表，都可以当做维表使用，参数完全可以用 Table Hints 来覆盖。

lookup.join.cache.ttl：表示缓存时间；这里值得注意的是，因为 Hive 维表会把维表所有数据缓存在 TM 的内存中，如果维表量很大，那么很容易就 OOM；如果 ttl 时间太短，那么会频繁的加载数据，性能会有很大影响。

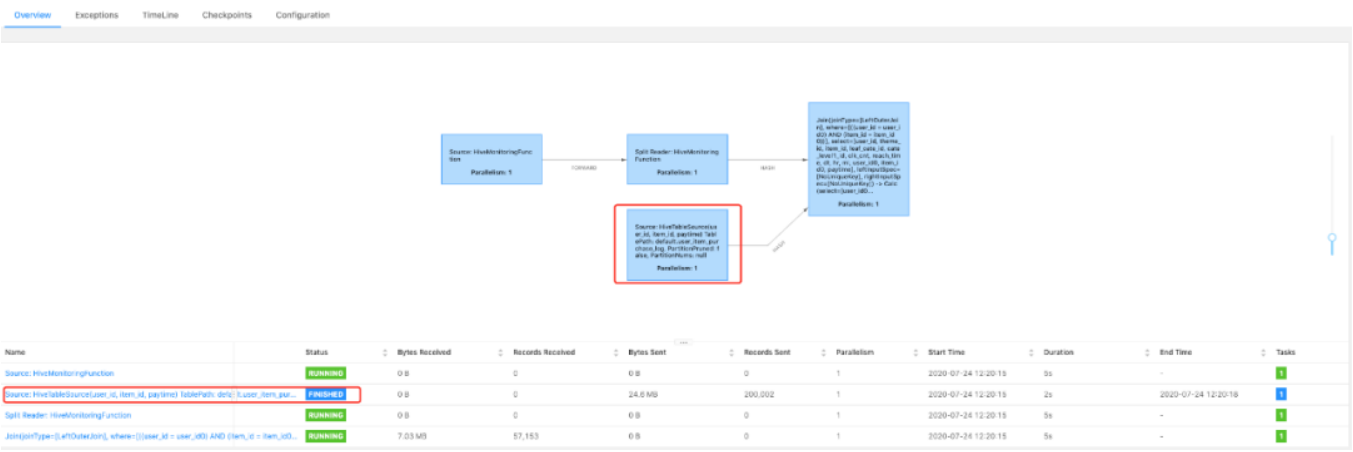


因为是 LEFT JOIN，所以维表中不存在的数据会以 NULL 补全。

再看一眼 DAG 图：

大家看一下画框的地方，能看到这边是使用的维表关联 LookupJoin。

如果大家 SQL 语句写错了，丢了 for system_time as of a.p，那么 DAG 图就会变成这样：



这种就不是维表 JOIN 其实更像是流和批在 JOIN。

写在最后

Hive Streaming 的完善意味着打通了流批一体的最后一道壁垒，既可以做到历史数据的 OLAP 分析，又可以实时吐出结果，这无疑是 ETL 开发者的福音，想必接下来的日子，会有更多的企业完成他们实时数仓的建设。

参考文档:

[1]https://ci.apache.org/projects/flink/flink-docs-release-1.11/dev/table/hive/
[2]https://github.com/apache/zeppelin/blob/master/docs/interpreter/flink.md