# Flink Sql on Zeppelin (6) ——Flink1.11预览 (上)

## Flink 1.11 Table&Sql预览

### 概述

- 之前就和大家说过要做一期Flink 1.11的超前点映版,加上Flink 1.11 release在即,所以这次的内容就和大家简单分享一些Table&Sql方面的新特性
- 就像标题一样,还是会在Zeppelin中演示,由于两个项目都没release,所以我自己编了下,不过大家用的时候可能有点问题,可以钉钉联系我解决,钉钉号是rnodvmd。链接: https://pan.baidu.com/s/1qCsv9nf35mrqLSTMq9f1fQ 密码: 4mkh
- 我会挑一些我觉得比较好用的新特性来分享,如果大家想要了解更多的话,可以看看 Flink社区分享的文章重磅! Apache Flink 1.11 功能前瞻来啦
- 这次我会从Catalog、Table、Function这3个地方来看(Databases没啥变化,暂时跳过)

## Catalog

- 社区从Flink 1.9就开始支持Hive Catalog,全新的版本不止支持Hive Catalog,也支持了Postgres Catalog。毕竟大部分公司的实时计算集群和离线计算集群是分开的,hive相对于Postgres还是比较重的。
- 在使用Hive Catalog之后,可以做到一次建表,终生使用。当集群重启之后,我们再也不用去复制一堆的DDL语句了。当然,美中不足的地方是,如果想要修改其中的参数怎么办? 比如我建了一个 Kafka Source Table ,指定了消费位点是 earliest-offset ,那我在集群重启之后不想从这个位点消费了,是否依然只能先 Drop Table 再 Create Table 呢? 先保留一下,下面说到Table的地方会给大家讲解
- Postgres Catablog目前尚未支持建表的操作,可能是来不及发布了吧。毕竟群里天 天有同学问什么时候能发布,给社区大佬们的紧迫感太强了吧哈哈
- 下面开始演示一下Catalog的相关操作

```
1 %flink111.ssql
2 —— 建一个Postgres Catalog, 注意! url后面不能跟库名!
3 CREATE CATALOG mypg WITH(
4 'type' = 'jdbc',
```

```
'default-database' = 'dijie',
'username' = 'postgres',
'password' = '123456',
'base-url' = 'jdbc:postgresql://localhost:5432/'
);
```

```
1 %flink.ssql
2 —— 看看结果
3 show catalogs;
```

%flink.ssql	
show catalogs;	
■ Ind	
catalog	
default_catalog	
hive	
турд	
Took 2 sec. Last updated by anonymous at July 04 2020, 9:33:47 PM.	

- 因为我还配置了Hive,所以也能看到Hive Catalog,至于如何在Zeppelin中让Flink 集成 Hive可以看环境准备
- 另外还有一种Catalog是基于内存的,也是默认的Catalog,集群shutdown之后数据就会消失,我们也可以创建一个基于内存的Catalog

- 结果我就不展示给大家看了, 毕竟图片还是挺费流量的~
- 吐槽一句,由于不知道Postgres语法,我觉得还是挺难用的,不知道为啥不先实现 Mysql Catalog,这个我之后会自己实现一下,然后给大家分享一下。
- 别的Catalog相关的DDL也不演示了,挑挑重点说一下

#### **Table**

• 简化了With中的参数、举个例子吧

```
1
     -- Flink 1.10
 2
     CREATE TABLE t3(
 3
 4
     ) WITH (
 5
         'update-mode' = 'append',
 6
         'connector.type' = 'kafka',
 7
         'connector.version' = 'universal',
 8
         'connector.topic' = 'zeppelin_01_test',
 9
         'connector.properties.zookeeper.connect' = '127.0.0.1:2181',
10
         'connector.properties.bootstrap.servers' = '127.0.0.1:9092',
11
         'connector.properties.group.id' = 'zeppelin_01_test',
12
         'connector.startup-mode' = 'earliest-offset',
13
         'format.type'='json'
14
15
```

```
1
     -- Flink 1.11
 2
     CREATE TABLE t3(
 3
 4
     ) WITH (
 5
      'connector' = 'kafka',
 6
      'topic' = 'user_behavior',
 7
      'properties.bootstrap.servers' = 'localhost:9092',
 8
      'properties.group.id' = 'testGroup',
 9
      'format' = 'json',
10
      'scan.startup.mode' = 'earliest-offset'
11
     );
```

- 很明显的能够看出来,减少了Key的部分内容,以前实在是太长太累了,而且没得必要啊
- 上面我们提到过,当我们把建的表持久化到Hive Catalog之后,如果想要修改一些参数,是否是只能重新 Create Table 呢? 其实是不需要的,这就是我们要说的Table第二个特件: Table Hints
- 光说不练假把式, 我们来看看如何使用, 并对比看看结果

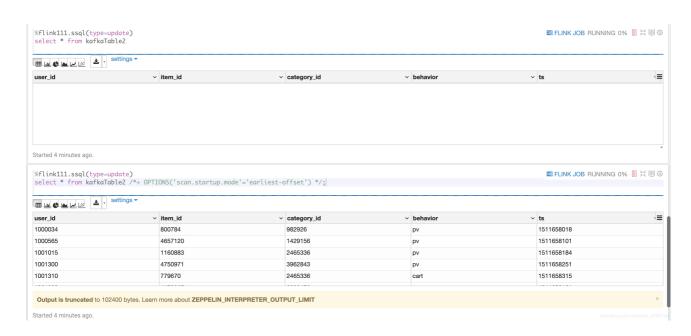
```
1 %flink
2 // 在开始之前,先打开这个配置,不然无法使用Table Hints
3 stenv.getConfig().getConfiguration().setBoolean("table.dynamic-table-options.enabled",true);
```

```
1
    %flink111.ssql
 2
    -- with中的key缩短了真是太好了,记得指定消费位点为'latest',方便我们观察现象
 3
     DROP TABLE IF EXISTS kafkaTable2;
 4
 5
     CREATE TABLE kafkaTable2 (
 6
     user_id BIGINT,
 7
        item_id BIGINT,
8
        category_id BIGINT,
9
        behavior STRING,
10
        ts BIGINT
11
     ) WITH (
12
      'connector' = 'kafka-0.11',
13
      'topic' = 'zeppelin_01_test',
14
      'properties.bootstrap.servers' = '127.0.0.1:9092',
15
      'properties.group.id' = 'testGroup',
16
      'format' = 'json',
17
      'scan.startup.mode' = 'latest-offset'
18
```

```
1 —— 从最新消费,因为我没有灌数据,所以应该是没有结果输出
2 %flink111.ssql(type=update)
3 select * from kafkaTable2
```

```
1 -- 使用Table Hits去OverWrite 启动位点,能够读到数据
2 %flink111.ssql(type=update)
3 select * from kafkaTable2 /*+ OPTIONS('scan.startup.mode'='earliest-offset') */;
```

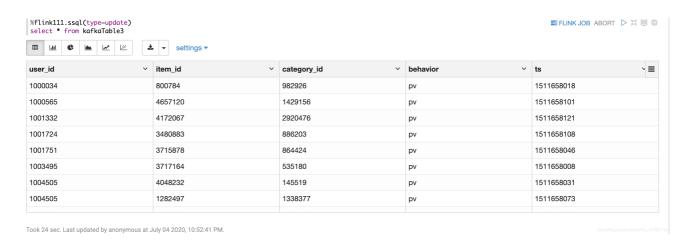
• 接下来让我们看看结果是否如预期一样



- 很明显,没有使用Table Hints的查询无法输出结果,而另一个使用了的,能够从最早的位点读到数据,完美!
- 当然,很多时候这样用确实很方便很爽,但是显得够不够规范化,那怎么办呢?又不想写那么多又长又臭的DDL。那么,就可以使用 Like 语句

```
1 %flink111.ssql
2 create table kafkaTable3 with('scan.startup.mode'='earliest-offset')
like kafkaTable2
```

• 再让我们从这个表里面捞点数据看看吧



- 很完美
- Flink1.11中又新增了三种类型的connector
  - DataGen SQL Connector: 一个可以帮我们模拟数据的Source, 当我们没法自己造数据的时候,可以使用这个来模拟数据,很实用

• Print SQL Connector: 很好用的一个功能,当你不知道到底是哪一步丢了数据,可以通过将每一步的结果插入 Print Sink Table ,更优秀的一点是我们可以用 Like 的语法来减少繁琐的字段映射:

```
1 | CREATE TABLE print_table WITH ('connector' = 'print') LIKE source
_table (EXCLUDING ALL)
```

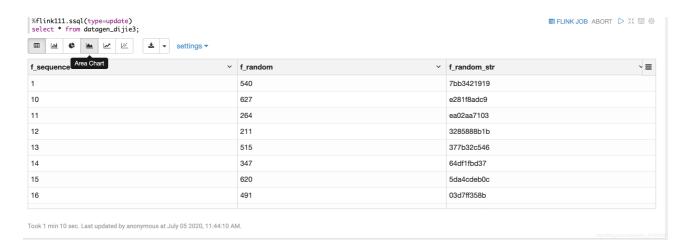
• BlackHole SQL Connector:并不是一个实质性的 Sink Table ,它就像 Unix /dev/null ,通常用于丢弃不需要的数据输出,一般用于压测或者UDF,当你不知道压力到底在sink还是前面的计算,可以用它来接收数据。也支持 Like 语句

```
1 | CREATE TABLE blackhole_table WITH ('connector' = 'blackhole') LIK
E source_table (EXCLUDING ALL)
```

• 给大家简单演示一下DataGen SQL Connector,别的两个connector就不演示了

```
1
     %flink111.ssql
 2
     CREATE TABLE datagen_dijie3 (
 3
      f_sequence INT,
 4
      f random INT,
 5
      f_random_str STRING
 6
     ) WITH (
 7
      'connector' = 'datagen',
 8
      'rows-per-second'='5',
 9
      'fields.f_sequence.kind'='sequence',
10
      'fields.f_sequence.start'='1',
11
      'fields.f_sequence.end'='1000',
12
      'fields.f_random.min'='1',
13
      'fields.f_random.max'='1000',
14
      'fields.f_random_str.length'='10'
15
     );
```

#### • 看看结果吧



- 相当利于我们测试使用了
- 重构了Source&Sink接口、终于不会再遇到
  - java.io.IOException: org.apache.flink.table.api.NoMatchingTableFactory
    Exception: Could not find a suitable table factory for 'org.apache.flin
    k.table.factories.TableSinkFactory' in the classpath.
- 之前遇到这个歌报错,会有两种情况:第一种是少包,第二种是Properties写的有问题,有时候很难排查
- 重构之后,已经能很明显的看出来是哪个问题了
  - Caused by: org.apache.flink.table.api.ValidationException: Could not find any factory for identifier 'kafka-0.11' that implements 'org.apache.flink.table.factories.DynamicTableSourceFactory' in the classpath.

```
Caused by: org.apache.flink.table.api.ValidationException: Unsupported
 2
      options found for connector 'print'.
 3
 4
     Unsupported options:
 5
 6
     id
 7
 8
     Supported options:
 9
10
     connector
11
     print-identifier
12
     property-version
     standard-error
```

• 大家应该很明显的就能看出来,哪有问题了

- 同样的, 之前的版本还有一个让人摸不清头脑的地方
  - java.io.IOException: org.apache.flink.table.api.TableException: Upsert StreamTableSink requires that Table has a full primary keys if it is updated.
- 这个报错是什么问题上次我已经说过了,不太了解的同学可以看聚合结果写入Kafka
- 新版本再也不是从Sql中推断主键,而是支持在表中定义 Primay Key ,我们再也不会出现这个问题了(除非你的表不支持定义 Primay Key ),简单的演示一下吧
  - 1 %flink111.ssql
    -- 将一个撤回流写入Mysql, 之前的版本这样写是会报错的,因为无法推断出主键
    insert into MyUserTable select a.f\_random\_str,a.f\_sequence from datag
    en\_dijie3 a left join datagen\_dijie3 b on a.f\_sequence = b.f\_sequence

```
☆▼
         Query Favorites
                                   Query History
name
                cnt
281c744d16
                    9
44b8d3e8f8
                    8
996ef8e494
                    15
e1aba4e05d
                    2
80c9bbe381
                    13
94a212a31e
                    4
b3b3243b4c
                    1
a1c0a094a1
                    11
f0c599418a
                    12
1ed6ceaa28
                    3
9dc3d76e82
                    6
c99d6e682c
                    5
44df3174eb
                    10
79d02a52d2
                    14
b961140929
                     7
```

select \* from

- 看来数据成功的写入了! 这个更新实在太棒了,之前的版本只能通过 Group By + Last\_value() 强行生成主键来执行语句
- 最后,说一下Change Log。之前的博客给大家分享过如何将撤回流写入Kafka,在 Flink 1.11中,终于支持CDC类型的数据,比如Canal-Json。但是美中不足的是,目 前只是将接口完善了,真正的实现还没来得及写,所以就不给大家演示了,不过等过 几天没这么忙了,我会自己实现一下 CanalJsonFormatFactory,可以期待一下~

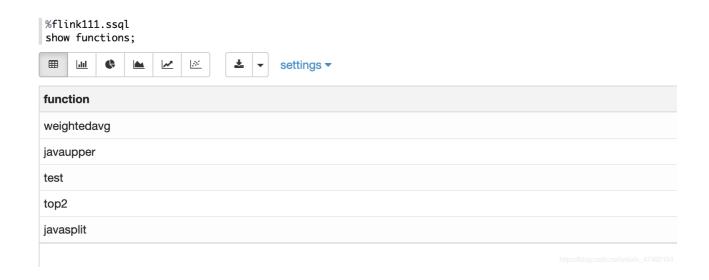
https://blog.csdn.net/weixin\_47482194

• Table部分就没有什么了,至于Hive部分的内容,我准备留到下期再说吧,因为新的 Hive连接器太强了

#### **Function**

• 其实Function的部分改动并不大,虽然之前就支持了Function的DDL,不过只能通过 Java/Scala代码来玩,现在终于在纯Sql的环境支持了DDL。演示一下吧

```
1 %flink111.ssql
create function test AS 'org.apache.zeppelin.flink.udf.JavaSplit'
```



• 别的DDL就不演示了~毕竟用法都很简单

## 写在最后

- 其实大家能看出来,Table部分是改动最大的,而且是给我们带来最多便利的地方, 真的是得感谢一下社区开发者们~
- 至于Hive相关的内容,准备下期单独拎出来讲,会整一套全链路的东西,期待一下吧
- 最后呢,可能有些地方讲的还是有所不足,或者有些地方讲的有误,希望大家谅解,也欢迎大家指出我的不足
- 参考资料: Flink 社区钉钉群的视频回放"大数据+AI meetup 6.14下午场"之信老师讲解的部分