链滴

Flink SQL 1.11 新功能详解

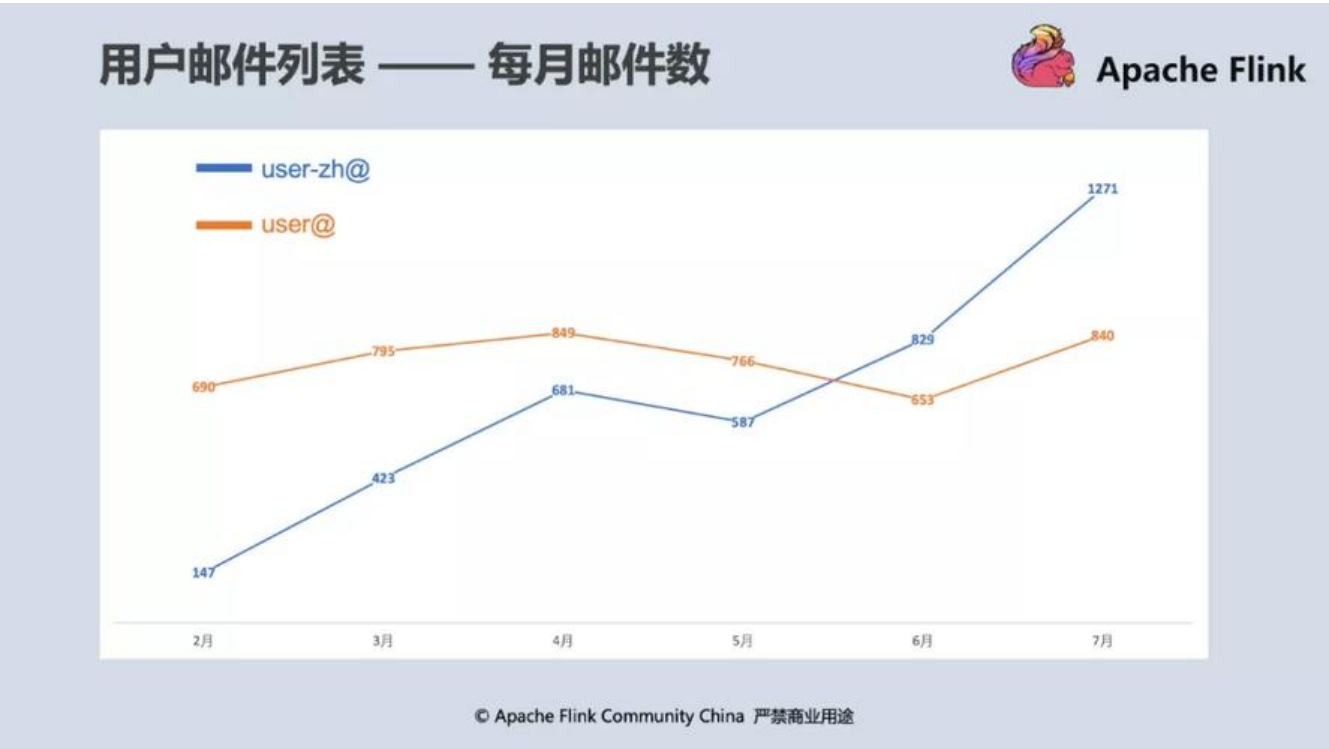
作者：fc13240

原文链接：https://ld246.com/article/1600421223203

来源网站：链滴

1 Flink 1.8 ~ 1.11 社区发展趋势回顾

自 2019 年初阿里巴巴宣布向 Flink 社区贡献 Blink 源码并在同年 4 月发布 Flink 1.8 版本后，Flink 社区的活跃程度犹如坐上小火箭般上升，每个版本包含的 git commits 数量以 50% 的增速持续上涨，吸引了一大批国内开发者和用户参与到社区的生态发展中来，中文用户邮件列表（user-zh@）更是今年 6 月首次超出英文用户邮件列表（user@），在 7 月超出比例达到了 50%。对比其它 Apache 源社区如 Spark、Kafka 的用户邮件列表数（每月约 200 封左右）可以看出，整个 Flink 社区的发展然非常健康和活跃。



2 Flink SQL 新功能解读

原文链接：Flink SQL 1.11 新功能详解

在了解 Flink 整体发展趋势后，我们来看下最近发布的 Flink 1.11 版本在 connectivity 和 simplicity 方面都带来了哪些令人耳目一新的功能。

FLIP-122：简化 connector 参数

整个 Flink SQL 1.11 在围绕易用性方面做了很多优化，比如 FLIP-122[1] 。

优化了 connector 的 property 参数名称冗长的问题。以 Kafka 为例，在 1.11 版本之前用户的 DDL 需要声明成如下方式：

```
CREATE TABLE user_behavior ( ...) WITH ( 'connector.type'='kafka', 'connector.version'='universal', 'connector.topic'='user_behavior', 'connector.startup-mode'='earliest-offset', 'connector.properties.zookeeper.connect'='localhost:2181', 'connector.properties.bootstrap.servers'='localhost:9092', 'format.type'='json');
```

而在 Flink SQL 1.11 中则简化为：

```
CREATE TABLE user_behavior ( ...) WITH ( 'connector'='kafka', 'topic'='user_behavior', 'scan.tartup.mode'='earliest-offset', 'properties.zookeeper.connect'='localhost:2181', 'properties.bootstrap.servers'='localhost:9092', 'format'='json');
```

DDL 表达的信息量丝毫未少，但是看起来清爽许多 :)。Flink 的开发者们为这个优化做了很多讨论，兴趣可以围观 FLIP-122 Discussion Thread[2]。

FLINK-16743：内置 connectors

Flink SQL 1.11 新加入了三种内置的 connectors，如下表所示:

**connector **	描述	使用场景
'connector'='datagen'	用于生成随机数据的source	用于测试
'connector'='blackhole'	不做任何处理的 sink	用于性能测试
'connector'='print'	打印到标准输出流(.out文件)的 sink	用于调试

在外部 connector 环境还没有 ready 时，用户可以选择 datagen source 和 print sink 快速构建 pipline 熟悉 Flink SQL；对于想要测试 Flink SQL 性能的用户，可以使用 blackhole 作为 sink；对于调排错场景，print sink 会将计算结果打到标准输出（比如集群环境下就会打到 taskmanager.out 文），使得定位问题的成本大大降低。

FLIP-110：LIKE 语法

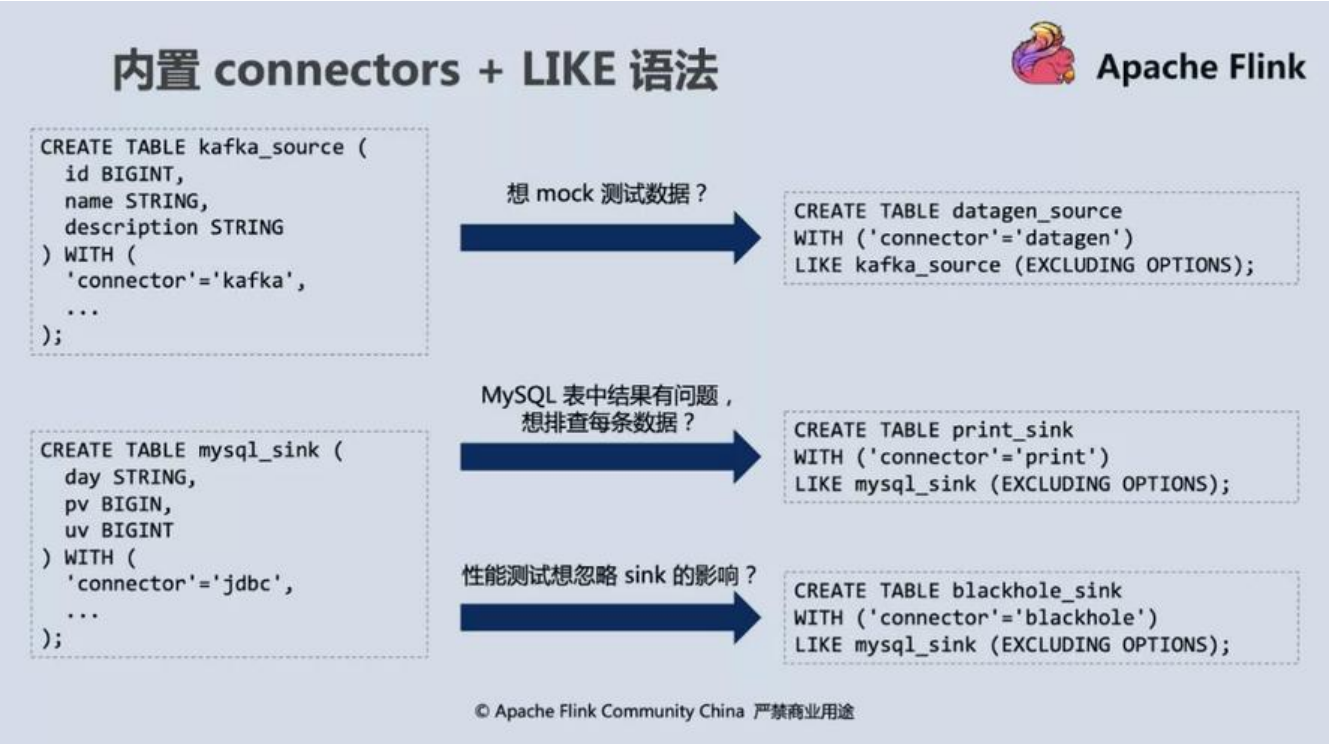
Flink SQL 1.11 支持用户从已定义好的 table DDL 中快速 “fork” 自己的版本并进一步修改 watermark 或者 connector 等属性。比如下面这张 base_table 上想加一个 watermark，在 Flink 1.11 版本前，用户只能重新将表声明一遍，并加入自己的修改，可谓 “牵一发而动全身”。

```
-- before Flink SQL 1.11CREATE TABLE base_table ( id BIGINT, name STRING, ts TIMESTAMP) WITH ( 'connector.type'='kafka', ...);
CREATE TABLE derived_table ( id BIGINT, name STRING, ts TIMESTAMP, WATERMARK FORs AS ts - INTERVAL '5' SECOND) WITH ( 'connector.type'='kafka', ...);
```

从 Flink 1.11 开始，用户只需要使用 CREATE TABLE LIKE 语法就可以完成之前的操作。

```
-- Flink SQL 1.11CREATE TABLE base_table (  id BIGINT,  name STRING,  ts TIMESTAMP) WITH
  'connector'='kafka',  ...);
CREATE TABLE derived_table (  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND) LIKE base_
able;
```

而内置 connector 与 CREATE TABLE LIKE 语法搭配使用则会如下图一般产生“天雷勾地火”的效果极大提升开发效率。



FLIP-113：动态 Table 参数

对于像 Kafka 这种消息队列，在声明 DDL 时通常会有一个启动点位去指定开始消费数据的时间，如需要更改启动点位，在老版本上就需要重新声明一遍新点位的 DDL，非常不方便。

```
CREATE TABLE user_behavior (  user_id BIGINT,  behavior STRING,  ts TIMESTAMP(3)) WITH (
connector'='kafka',  'topic'='user_behavior',  'scan.startup.mode'='timestamp',  'scan.startup.t
mestamp-millis'='123456',  'properties.bootstrap.servers'='localhost:9092',  'format'='json');
```

从 Flink 1.11 开始，用户可以在 SQL client 中按如下方式设置开启 SQL 动态参数（默认是关闭的）如此即可在 DML 里指定具体的启动点位。

```
SET 'table.dynamic-table-options.enabled' = 'true';
SELECT user_id, COUNT(DISTINCT behaviro)FROM user_behavior /*+ OPTIONS('scan.startup.t
mestamp-millis'='1596282223') */GROUP BY user_id;
```

除启动点位外，动态参数还支持像 sink.partition、scan.startup.mode 等更多运行时参数，感兴趣移步 FLIP-113[3]，获得更多信息。

**FLIP-84：重构优化 TableEnvironment 接口 **

Flink SQL 1.11 以前的 TableEnvironment 接口定义和行为有一些不够清晰，比如:

● TableEnvironment#sqlUpdate() 方法对于 DDL 会立即执行，但对于 INSERT INTO DML 语句却 buffer 住的，直到调用 TableEnvironment#execute() 才会被执行，所以在用户看起来顺序执行的

句，实际产生的效果可能会不一样。

- 触发作业提交有两个入口，一个是 TableEnvironment#execute()，另一个是 StreamExecutionEnvironment#execute()，于用户而言很难理解应该使用哪个方法触发作业提交。
- 单次执行不接受多个 INSERT INTO 语句。

针对这些问题，Flink SQL 1.11 提供了新 API，即 TableEnvironment#executeSql()，它统一了执行 QL 的行为， 无论接收 DDL、查询 query 还是 INSERT INTO 都会立即执行。针对多 sink 场景提供了 StatementSet 和 TableEnvironment#createStatementSet() 方法，允许用户添加多条 INSERT 语 一起执行。

除此之外，新的 execute 方法都有返回值，用户可以在返回值上执行 print，collect 等方法。

新旧 API 对比如下表所示：

Current Interface	New Interface
tEnv.sqlUpdate("CREATE TABLE...”);	TableResult result = tEnv.executeSql("CREATE TABLE...”);
tEnv.sqlUpdate("INSERT INTO...SELECT...”); tEnv.execute();	TableResult result = tEnv.executeSql("INSERT INTO ... SELECT...”);
tEnv.sqlUpdate("insert into xx ...”); tEnv.sqlUpdate("insert into yy ...”); tEnv.execute();	StatementSet ss =tEnv.createStatementSet(); ss.addInsertSql("insert into xx ...”); ss.addInsertSql("insert into yy ...”); TaleResult result = ss.execute();

对于在 Flink 1.11 上使用新接口遇到的一些常见问题，云邪做了统一解答，可在 Appendix 部分查看。

FLIP-95：TableSource & TableSink 重构

开发者们在 Flink SQL 1.11 版本花了大量经历对 TableSource 和 TableSink API 进行了重构，核心化点如下：

- 移除类型相关接口，简化开发，解决迷惑的类型问题，支持全类型
- 寻找 Factory 时，更清晰的报错信息
- 解决找不到 primary key 的问题
- 统一了流批 source，统一了流批 sink
- 支持读取 CDC 和输出 CDC
- 直接高效地生成 Flink SQL 内部数据结构 RowData

新 DynamicTableSink API 去掉了所有类型相关接口，因为所有的类型都是从 DDL 来的，不需要 TableSink 告诉框架是什么类型。而对于用户来说，最直观的体验就是在老版本上遇到各种奇奇怪怪报错概率降低了很多，比如不支持的精度类型和找不到 primary key / table factory 的诡异报错在新版本都不复存在了。关于 Flink 1.11 是如何解决这些问题的详细可以在附录部分阅读。

FLIP-123：Hive Dialect

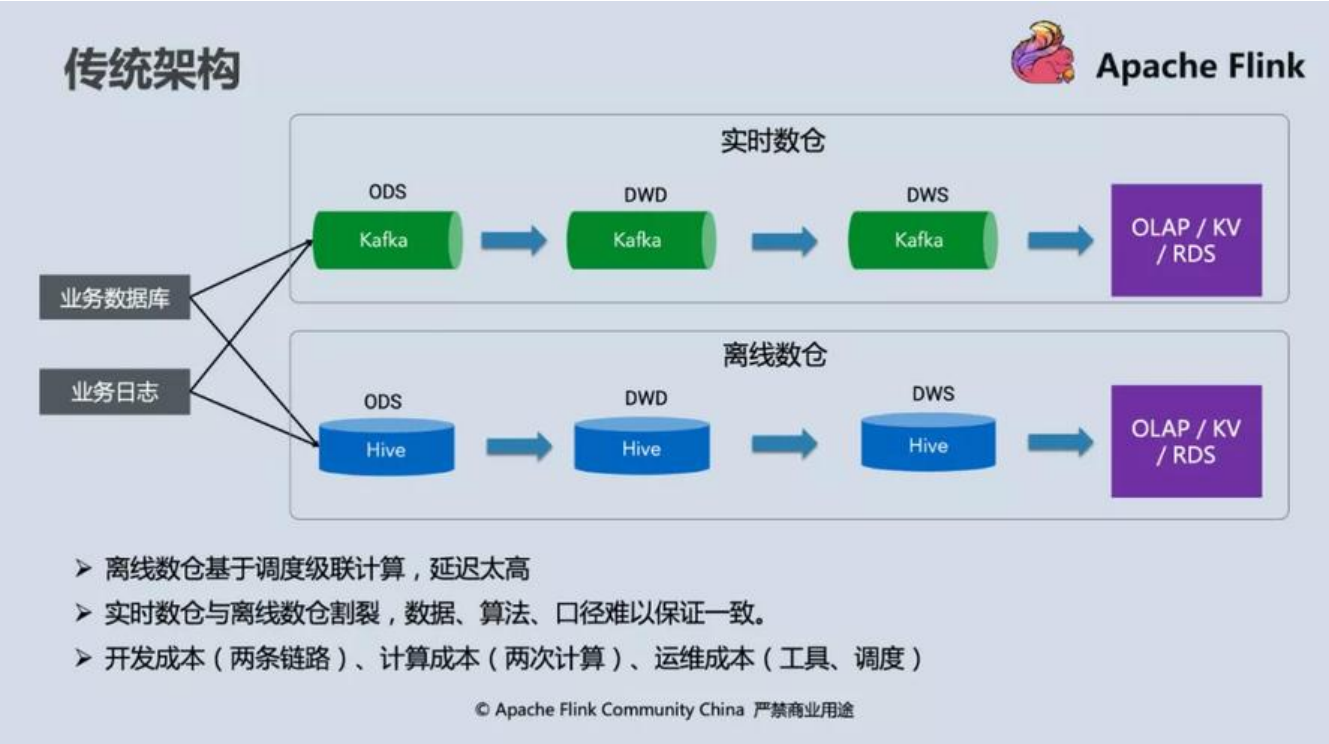
Flink 1.10 版本对 Hive connector 的支持达到了生产可用，但是老版本的 Flink SQL 不支持 Hive DL 及使用 Hive syntax，这无疑限制了 Flink connectivity。在新版本中，开发者们为支持 HiveQL 引了新 parser，用户可以在 SQL client 的 yaml 文件中指定是否使用 Hive 语法，也可以在 SQL client 中通过 set table.sql-dialect=hive/default 动态切换。更多信息可以参考 FLIP-123[4]。

以上简要介绍了 Flink 1.11 在**减少用户不必要的输入和操作**方面对 connectivity 和 simplicity 方面出的优化。下面会重点介绍在**外部系统和数据生态**方面对 connectivity 和 simplicity 的两个核心优，并附上最佳实践介绍。

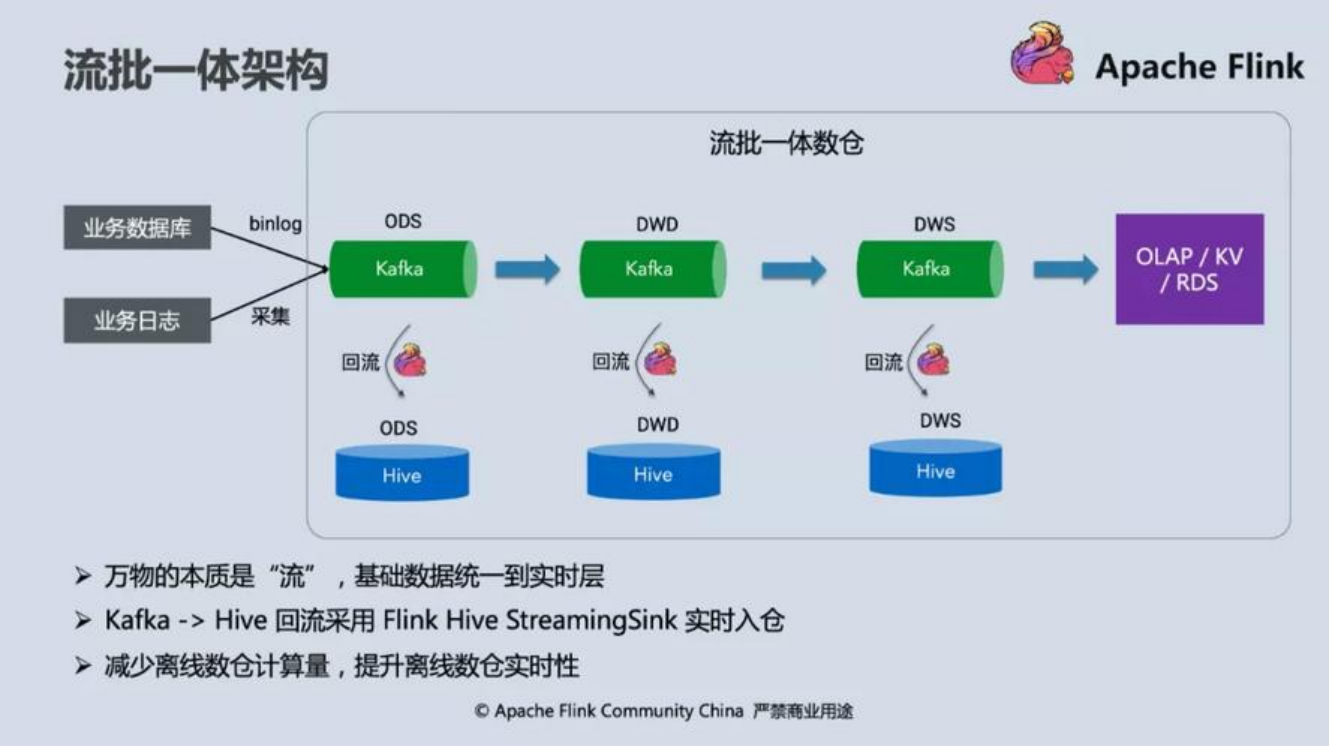
3 Hive 数仓实时化 & Flink SQL + CDC 最佳实践

Hive 数仓实时化

下图是一张非常经典的 Lambda 数仓架构，在整个大数据行业从批处理逐步拥抱流计算的许多年里表“最先进的生产力”。然而随着业务发展和规模扩大，两套单独的架构所带来的开发、运维、计算本问题已经日益凸显。



而 Flink 作为一个流批一体的计算引擎，在最初的设计上就认为“万物本质皆是流”，批处理是流计的特例，如果能够在自身提供高效批处理能力的同时与现有的大数据生态结合，则能以最小侵入的方改造现有的数仓架构使其支持流批一体。在新版本中，Flink SQL 提供了开箱即用的 “Hive 数仓同”功能，即所有的数据加工逻辑由 Flink SQL 以流计算模式执行，在数据写入端，自动将 ODS，DWD 和 DWS 层的已经加工好的数据**实时回流**到 Hive table。One size (sql) fits for all suites (tables) 设计，使得在 batch 层不再需要维护任何计算 pipeline。



对比传统架构，它带来的好处和解决的问题有哪些呢?

● 计算口径与处理逻辑统一，降低开发和运维成本

传统架构维护两套数据 pipeline 最大的问题在于需要保持它们**处理逻辑的等价性**，但由于使用了不的计算引擎（比如离线使用 Hive，实时使用 Flink 或 Spark Streaming），SQL 往往不能直接套用存在**代码上的差异性**，经年累月下来，离线和实时处理逻辑很可能会完全 diverge，有些大的公司甚会存在两个团队分别去维护实时和离线数仓，人力物力成本非常高。Flink 支持 Hive Streaming Sink 后，实时处理结果可以实时回流到 Hive 表，离线的计算层可以完全去掉，处理逻辑由 Flink SQL 统维护，离线层只需要使用回流好的 ODS、DWD、DWS 表做进一步 ad-hoc 查询即可。

● 离线对于“数据漂移”的处理更自然，离线数仓“实时化”

离线数仓 pipeline 非 data-driven 的调度执行方式，在跨分区的数据边界处理上往往需要很多 trick 保证分区数据的完整性，而在两套数仓架构并行的情况下，有时会存在对 late event 处理差异导致数对比不一致的问题。而实时 data-driven 的处理方式和 Flink 对于 event time 的友好支持本身就意着以业务时间为分区（window），通过 event time + watermark 可以统一定义实时和离线数据的整性和时效性，Hive Streaming Sink 更是解决了离线数仓同步的“最后一公里问题”。

下面会以一个 Demo 为例，介绍 Hive 数仓实时化的最佳实践。

■ 实时数据写入 Hive 的最佳实践

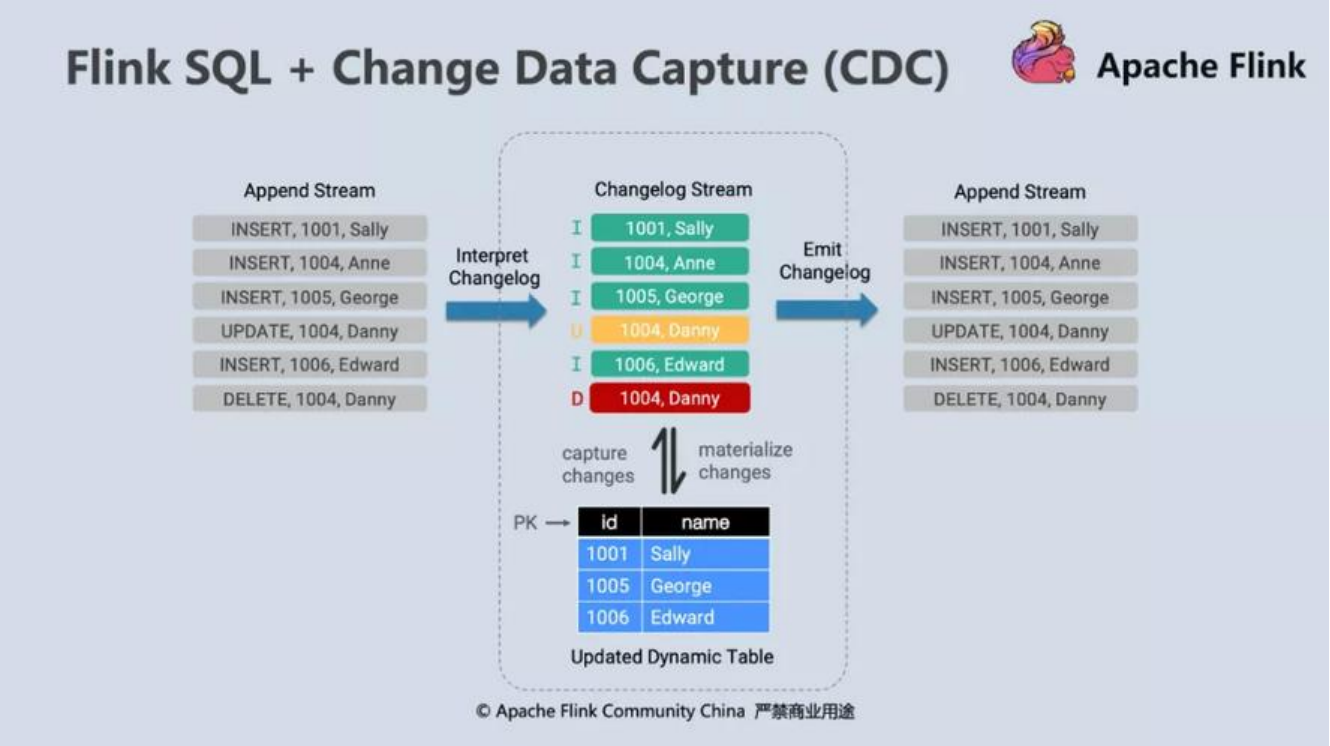
<iframe class="video_iframe rich_pages" data-vidtype="2" data-mpvid="wxv_1507565286744425027" data-cover="http%3A%2F%2Fmmbiz.qpic.cn%2Fmmbiz_jpg%2F8AsYBicEePu7xMOYdV3qRzE2AWad523QeNbToqDiaYZzOnvBg8XhEwibzaLKW5zrFibMLaGWdtuworoCEAlicTwLp%2F0%3Fwx_fmt%3Djpeg" allowfullscreen="" frameborder="0" data-ratio="1.7777777777777777" data-w="1280" data-src="https://mp.weixin.qq.com/mp/readtemplate?t=pages/video_player_tmpl&auto=0&vid=wxv_1507565286743425027" width="677" height="393" data-vh="380.8125" data-vw="677" scrolling="no" marginwidth="0" marginheight="0" src="https://mp.weixin.qq.com/mp/videoplayer?video_h=380.8125&video_w=677&scene=&random_num=585&article_title=Flink+SQL+1.11+%E6%96%B0%E5%8A%9F%E8%83%BD%E4%B8%8E%E6%9C

80%E4%BD%B3%E5%AE%9E%E8%B7%B5&source=4&vid=wxv_1507565286743425027&mid2247489141&idx=1&__biz=MzU3Mzg4OTMyNQ==&nodetailbar=0&uin=&key=&pass_ticke=&version=/mmbizwap/zh_CN/htmledition/js/appmsg/index507425.js&devicetype=&wxtokn=777&sessionid=svr_be1cc3dac8e&preview=0&is_in_pay_subscribe=0&nickname=Flink+%4%B8%AD%E6%96%87%E7%A4%BE%E5%8C%BA&roundHeadImg=http://mmbiz.qpic.cn/mbiz_png/8AsYBicEePu6FJHxaI14AsXuzeg4SybT0hiaCSohrIY75oiaOMzhQU7RouiafjNa76k2CtDxxB2JqnawqFqV3zg3A/0?wx_fmt=png&enterid=1600415037&subscene="></iframe>

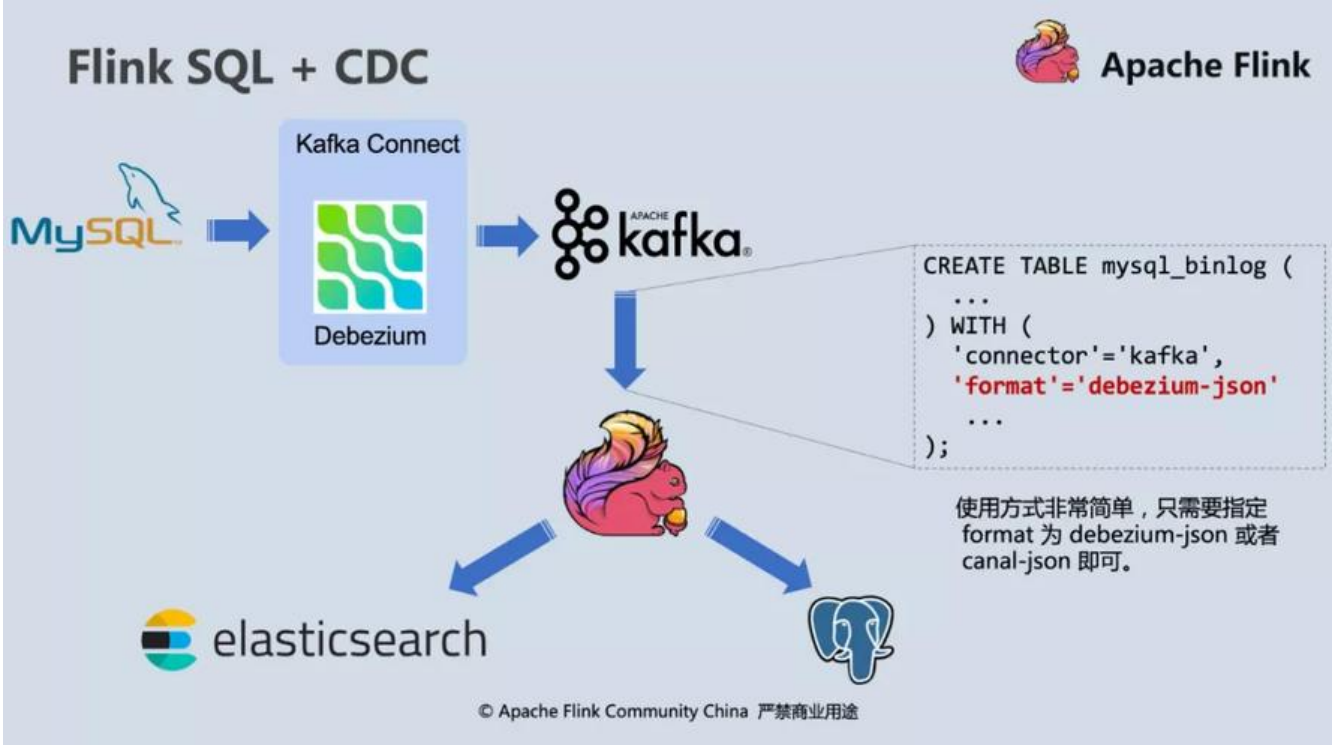
FLIP-105：支持 Change Data Capture (CDC)

除了对 Hive Streaming Sink 的支持，Flink SQL 1.11 的另一大亮点就是引入了 CDC 机制。CDC 的称是 Change Data Capture，用于 tracking 数据库表的增删改查操作，是目前非常成熟的同步数据变更的一种方案。在国内常见的 CDC 工具就是阿里开源的 Canal，在国外比较流行的有 Debezium。link SQL 在设计之初就提出了 Dynamic Table 和“流表二象性”的概念，并且在 Flink SQL 内部完支持了 Changelog 功能，相对于其他开源流计算系统是一个重要优势。本质上 Changelog 就等价一张一直在变化的数据库的表。Dynamic Table 这个概念是 Flink SQL 的基石， Flink SQL 的各个子之间传递的就是 Changelog，完整地支持了 Insert、Delete、Update 这几种消息类型。

得益于 Flink SQL 运行时的强大，Flink 与 CDC 对接只需要将外部的数据流转为 Flink 系统内部的 Insrt、Delete、Update 消息即可。进入到 Flink 内部后，就可以灵活地应用 Flink 各种 query 语法了。



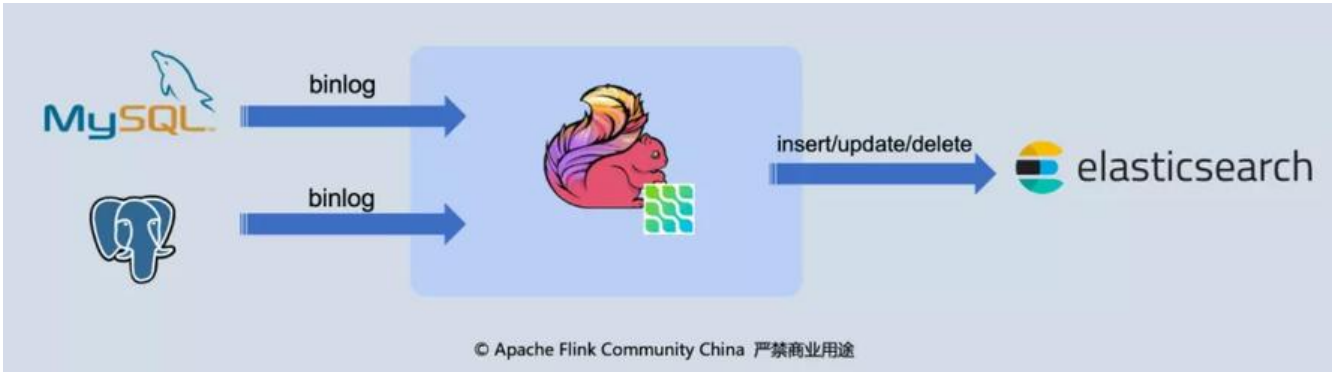
在实际应用中，把 Debezium Kafka Connect Service 注册到 Kafka 集群并带上想同步的数据库表息，Kafka 则会自动创建 topic 并监听 Binlog，把变更同步到 topic 中。在 Flink 端想要消费带 CDC 的数据也很简单，只需要在 DDL 中声明 format = debezium-json 即可。



在 Flink 1.11 上开发者们还做了一些有趣的探索，既然 Flink SQL 运行时能够完整支持 Changelog 那是否有可能不需要 Debezium 或者 Canal 的服务，直接通过 Flink 获取 MySQL 的变更呢？答案然是可以，Debezium 类库的良好设计使得它的 API 可以被封装为 Flink 的 Source Function，不需再起额外的 Service，目前这个项目已经开源，支持了 MySQL 和 Postgres 的 CDC 读取，后续也会持更多类型的数据库，可移步到下方链接解锁更多使用姿势。

https://github.com/ververica/flink-cdc-connectors

下面的 Demo 会介绍如何使用 flink-cdc-connectors 捕获 MySQL 和 Postgres 的数据变更，并利用 Flink SQL 做多流 join 后实时同步到 Elasticsearch 中。



假设你在一个电商公司，订单和物流是你最核心的数据，你想要实时分析订单的发货情况。因为公司经很大了，所以商品的信息、订单的信息、物流的信息，都分散在不同的数据库和表中。我们需要创一个流式 ETL，去实时消费所有数据库全量和增量的数据，并将他们关联在一起，打成一个大宽表。而方便数据分析师后续的分析。

■ 使用 Flink SQL CDC 的最佳实践展示

**

**

<iframe class="video_iframe rich_pages" data-vidtype="2" data-mpvid="wxv_1507569416954599906" data-cover="http%3A%2F%2Fmmbiz.qpic.cn%2Fmmbiz_jpg%2F8AsYBicEePu7xMOYdV3qRzE2AWad523QZKklpicPQlW8rQwGn1MiaAibMxy7icBoxPt8PicPm8JsLtEicxcmibic9A4lK%2F0%3Fwx_fmt%3Djpeg" allowfullscreen="" frameborder="0" data-ratio="1.7777777777777777" data-w="1920" data-src="https://mp.weixin.qq.com/mp/readtemplate?t=pages/video_payer_tmpl&auto=0&vid=wxv_1507569416958459906" width="677" height="393" data-vh="30.8125" data-vw="677" scrolling="no" marginwidth="0" marginheight="0" src="https://mp.eixin.qq.com/mp/videoplayer?video_h=380.8125&video_w=677&scene=&random_num=585&article_title=Flink+SQL+1.11+%E6%96%B0%E5%8A%9F%E8%83%BD%E4%B8%8E%E6%9C80%E4%BD%B3%E5%AE%9E%E8%B7%B5&source=4&vid=wxv_1507569416958459906&mid2247489141&idx=1&__biz=MzU3Mzg4OTMyNQ==&nodetailbar=0&uin=&key=&pass_ticke=&version=/mmbizwap/zh_CN/htmledition/js/appmsg/index507425.js&devicetype=&wxtokn=777&sessionid=svr_be1cc3dac8e&preview=0&is_in_pay_subscribe=0&nickname=Flink+%4%B8%AD%E6%96%87%E7%A4%BE%E5%8C%BA&roundHeadImg=http://mmbiz.qpic.cn/mbiz_png/8AsYBicEePu6FJHxaI14AsXuzeg4SybT0hiaCSohrIY75oiaOMzhQU7RouiafjNa76k2CtDxxB2JqnawqFqV3zg3A/0?wx_fmt=png&enterid=1600415037&subscene="></iframe>

4 Flink SQL 1.12 未来规划

以上介绍了 Flink SQL 1.11 的核心功能与最佳实践，对于下个版本，云邪也给出了一些 ongoing 的划，并欢迎大家在社区积极提出意见 & 建议。

- FLIP-132[5]：Temporal Table DDL （Binlog 模式的维表关联）
- FLIP-129[6]：重构 Descriptor API （Table API 的 DDL）
- 支持 Schema Registry Avro 格式
- CDC 更完善的支持（批处理，upsert 输出到 Kafka 或 Hive）
- 优化 Streaming File Sink 小文件问题
- N-ary input operator （Batch 性能提升）

5 附录

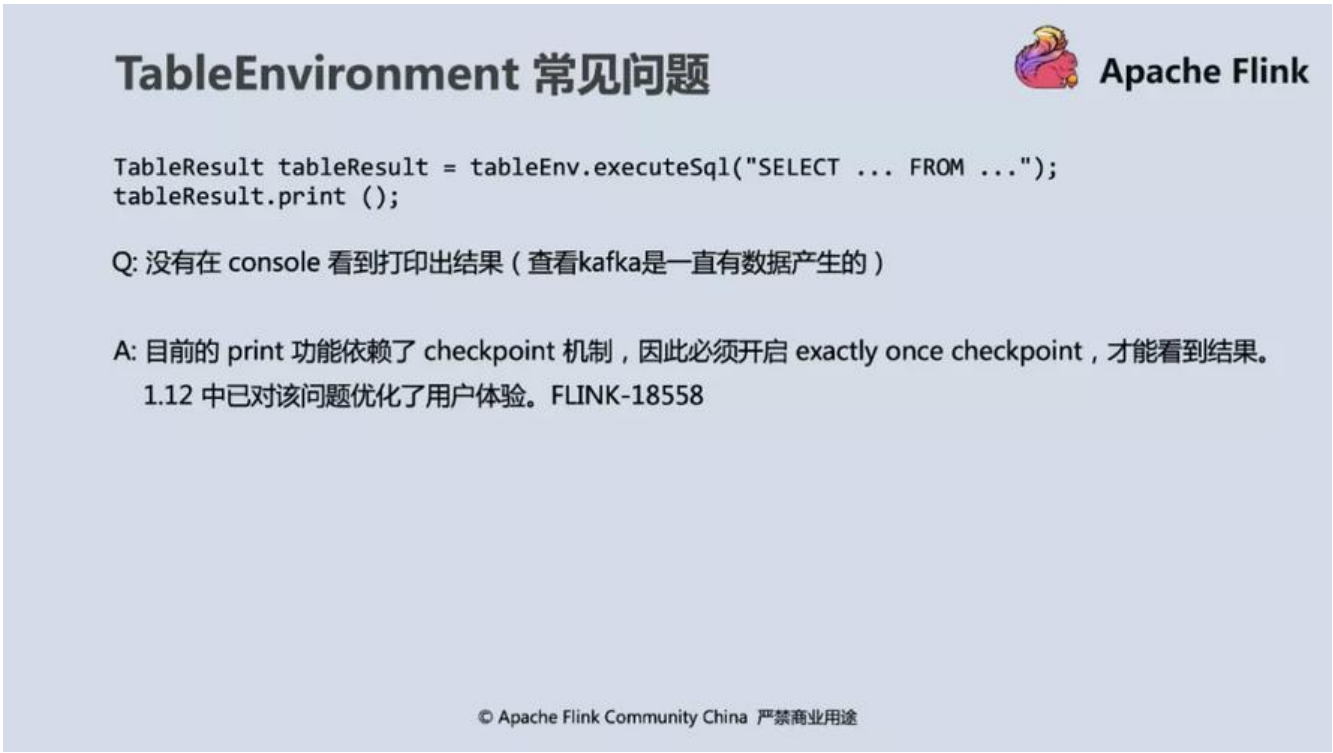
使用新版本 TableEnvironment 遇到的常见报错及原因

第一个常见报错是 No operators defined in streaming topolog。遇到这个问题的原因是在老版本执行 INSERT INTO 语句的下面两个方法：

TableEnvironment#sqlUpdate()TableEnvironment#execute()

在新版本中没有完全向前兼容（方法还在，执行逻辑变了），如果没有将 Table 转换为 AppendedSteam/RetractStream 时（通过StreamExecutionEnvironment#toAppendStream/toRetractStrea），上面的代码执行就会出现上述错误；与此同时，一旦做了上述转换，就必须使用 StreamExecutioEnvironment#execute() 来触发作业执行。所以建议用户还是迁移到新版本的 API 上面，语义上也更清晰一些。

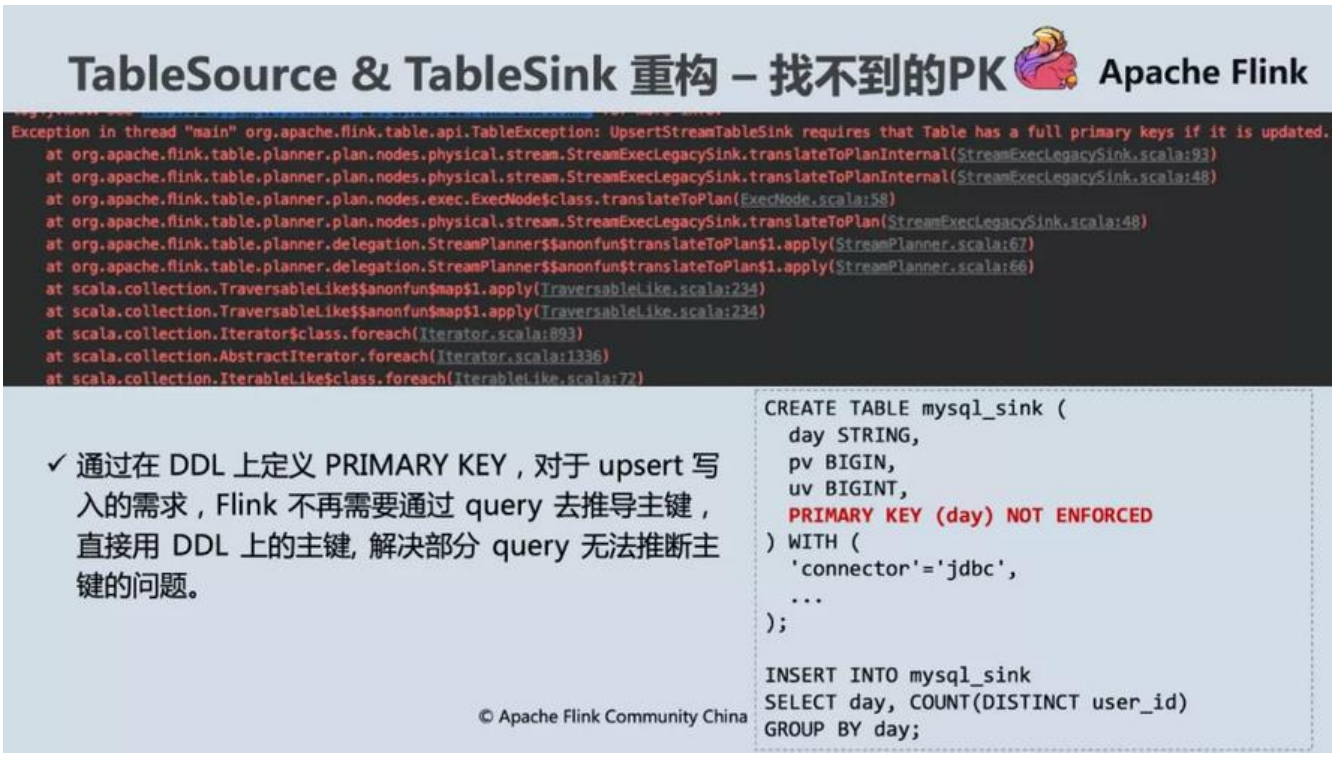
第二个问题是调用新的 TableEnvironemnt#executeSql() 后 print 没有看到返回值，原因是因为目前 print 依赖了 checkpoint 机制，开启 exactly-onece 后就可以了，新版本会优化此问题。



老版本的 StreamTableSource、StreamTableSink 常见报错及新版本优化

第一个常见报错是不支持精度类型，经常出现在 JDBC 或者 HBase 数据源上，在新版本上这个问题不会再出现了。

第二个常见报错是 Sink 时找不到 PK，因为老的 StreamSink 需要通过 query 去推导出 PK，当 query 变得复杂时有可能会丢失 PK 信息，但实际上 PK 信息在 DDL 里就可以获取，没有必要通过 query 推导，所以新版本的 Sink 就不会再出现这个错误啦。



第三个常见报错是在解析 Source 和 Sink 时，如果用户少填或者填错了参数，框架返回的报错信息模糊，“找不到 table factory”，用户也不知道该怎么修改。这是因为老版本 SPI 设计得比较通用没有对 Source 和 Sink 解析的逻辑做单独处理，当匹配不到完整参数列表的时候框架已经默认当前的 able factory 不是要找的，然后遍历所有的 table factories 发现一个也不匹配，就报了这个错。在新的加载逻辑里，Flink 会先判断 connector 类型，再匹配剩余的参数列表，这个时候如果必填的参数失或填错了，框架就可以精准报错给用户。

TableSource & TableSink 重构 – 找不到Factory

Apache Flink

```
org.apache.flink.table.api.NoMatchingTableFactoryException: Could not find a suitable table factory for 'org.apache.flink.table.factories
  .TableSinkFactory' in
the classpath.

Reason: No factory supports all properties.
```

⬆ 老 API：如果有任何一个 property 不满足要求，框架就抓瞎了

⬇ 新 API：先匹配 "connector"，再匹配剩余的，精确提醒哪个参数填错了。

```
Caused by: org.apache.flink.table.api.ValidationException: 'scan.startup.timestamp-millis' is required in 'timestamp' startup mode but missing.
    at org.apache.flink.streaming.connectors.kafka.table.KafkaOptions.lambda$validateScanStartupMode$0(KafkaOptions.java:103)
    at java.util.Optional.ifPresent(Optional.java:159)
    at org.apache.flink.streaming.connectors.kafka.table.KafkaOptions.validateScanStartupMode(KafkaOptions.java:172)
    at org.apache.flink.streaming.connectors.kafka.table.KafkaOptions.validateTableOptions(KafkaOptions.java:164)
    at org.apache.flink.streaming.connectors.kafka.table.KafkaDynamicTableFactoryBase.createDynamicTableSource(KafkaDynamicTableFactoryBase.java:83)
    at org.apache.flink.table.factories.FactoryUtil.createTableSource(FactoryUtil.java:122)
    ... 26 more
```


原文链接：Flink SQL 1.11 新功能详解