Flink 1.10 SQL、HiveCatalog 与事件时间整合示例

简介： Flink 1.10 与 1.9 相比又是个创新版本，在我们感兴趣的很多方面都有改进，特别是 Flink SQL。本文用根据埋点日志计算 PV、UV 的简单示例来体验 Flink 1.10 的两个重要新特性.

Flink 1.10 与 1.9 相比又是个创新版本，在我们感兴趣的很多方面都有改进，特别是 Flink SQL。本文用根据埋点日志计算 PV、UV 的简单示例来体验 Flink 1.10 的两个重要新特性：

一是 SQL DDL 对事件时间的支持；
二是 Hive Metastore 作为 Flink 的元数据存储（即 HiveCatalog）。

这两点将会为我们构建实时数仓提供很大的便利。

添加依赖项

示例采用 Hive 版本为 1.1.0，Kafka 版本为 0.11.0.2。

要使 Flink 与 Hive 集成以使用 HiveCatalog，需要先将以下 JAR 包放在 ${FLINK_HOME}/lib 目录下。

- flink-connector-hive_2.11-1.10.0.jar
- flink-shaded-hadoop-2-uber-2.6.5-8.0.jar
- hive-metastore-1.1.0.jar
- hive-exec-1.1.0.jar
- libfb303-0.9.2.jar

后三个 JAR 包都是 Hive 自带的，可以在 ${HIVE_HOME}/lib 目录下找到。前两个可以通过阿里云 Maven 搜索 GAV 找到并手动下载（groupId 都是org.apache.flink）。

再在 pom.xml 内添加相关的 Maven 依赖。

> Maven 下载：
> https://maven.aliyun.com/mvn/search

```
<properties>
    <scala.bin.version>2.11</scala.bin.version>
    <flink.version>1.10.0</flink.version>
```

```
    <hive.version>1.1.0</hive.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-table-api-scala_${scala.bin.version}</artifactId>
      <version>${flink.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-table-api-scala-bridge_${scala.bin.version}</artifactId>
      <version>${flink.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-table-planner-blink_${scala.bin.version}</artifactId>
      <version>${flink.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-sql-connector-kafka-0.11_${scala.bin.version}</artifactId
      <version>${flink.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-connector-hive_${scala.bin.version}</artifactId>
      <version>${flink.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-json</artifactId>
      <version>${flink.version}</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hive</groupId>
      <artifactId>hive-exec</artifactId>
      <version>${hive.version}</version>
    </dependency>
  </dependencies>
```

最后，找到 Hive 的配置文件 hive-site.xml，准备工作就完成了。

注册 HiveCatalog、创建数据库

不多废话了，直接上代码，简洁易懂。

```
val streamEnv = StreamExecutionEnvironment.getExecutionEnvironment
    streamEnv.setParallelism(5)
    streamEnv.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
```

```
    val tableEnvSettings = EnvironmentSettings.newInstance()
        .useBlinkPlanner()
        .inStreamingMode()
        .build()
    val tableEnv = StreamTableEnvironment.create(streamEnv, tableEnvSettings)

    val catalog = new HiveCatalog(
      "rtdw",                  // catalog name
      "default",               // default database
      "/Users/lmagic/develop",  // Hive config (hive-site.xml) directory
      "1.1.0"                  // Hive version
    )
    tableEnv.registerCatalog("rtdw", catalog)
    tableEnv.useCatalog("rtdw")

    val createDbSql = "CREATE DATABASE IF NOT EXISTS rtdw.ods"
    tableEnv.sqlUpdate(createDbSql)
```

创建 Kafka 流表并指定事件时间

我们的埋点日志存储在指定的 Kafka topic 里，为 JSON 格式，简化版 schema 大致如下。

```
"eventType": "clickBuyNow",
    "userId": "97470180",
    "shareUserId": "",
    "platform": "xyz",
    "columnType": "merchDetail",
    "merchandiseId": "12727495",
    "fromType": "wxapp",
    "siteId": "20392",
    "categoryId": "",
    "ts": 1585136092541
```

其中 ts 字段就是埋点事件的时间戳（毫秒）。在 Flink 1.9 时代，用 CREATE TABLE 语句创建流表时是无法指定事件时间的，只能默认用处理时间。而在 Flink 1.10 下，可以这样写。

```
CREATE TABLE rtdw.ods.streaming_user_active_log (
  eventType STRING COMMENT '...',
  userId STRING,
  shareUserId STRING,
  platform STRING,
  columnType STRING,
  merchandiseId STRING,
  fromType STRING,
  siteId STRING,
  categoryId STRING,
  ts BIGINT,
  procTime AS PROCTIME(), -- 处理时间
  eventTime AS TO_TIMESTAMP(FROM_UNIXTIME(ts / 1000, 'yyyy-MM-dd HH:mm:ss')), --
```

```
  WATERMARK FOR eventTime AS eventTime - INTERVAL '10' SECOND -- 水印
) WITH (
  'connector.type' = 'kafka',
  'connector.version' = '0.11',
  'connector.topic' = 'ng_log_par_extracted',
  'connector.startup-mode' = 'latest-offset', -- 指定起始offset位置
  'connector.properties.zookeeper.connect' = 'zk109:2181,zk110:2181,zk111:2181',
  'connector.properties.bootstrap.servers' = 'kafka112:9092,kafka113:9092,kafka11
  'connector.properties.group.id' = 'rtdw_group_test_1',
  'format.type' = 'json',
  'format.derive-schema' = 'true', -- 由表schema自动推导解析JSON
  'update-mode' = 'append'
)
```

Flink SQL 引入了计算列（computed column）的概念，其语法为 column_name AS computed_column_expression，它的作用是在表中产生数据源 schema 不存在的列，并且可以利用原有的列、各种运算符及内置函数。比如在以上 SQL 语句中，就利用内置的 PROCTIME() 函数生成了处理时间列，并利用原有的 ts 字段与 FROM_UNIXTIME()、TO_TIMESTAMP() 两个时间转换函数生成了事件时间列。

为什么 ts 字段不能直接用作事件时间呢？因为 Flink SQL 规定时间特征必须是 TIMESTAMP(3) 类型，即形如"yyyy-MM-ddTHH:mm:ssZ"格式的字符串，Unix 时间戳自然是不行的，所以要先转换一波。

既然有了事件时间，那么自然要有水印。Flink SQL 引入了 WATERMARK FOR rowtime_column_name AS watermark_strategy_expression 的语法来产生水印，有以下两种通用的做法：

- 单调不减水印（对应 DataStream API 的 AscendingTimestampExtractor）

```
WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL '0.001' SECOND
```

- 有界乱序水印（对应 DataStream API 的 BoundedOutOfOrdernessTimestampExtractor）

```
WATERMARK FOR rowtime_column AS rowtime_column - INTERVAL 'n' TIME_UNIT
```

上文的 SQL 语句中就是设定了 10 秒的乱序区间。如果看官对水印、AscendingTimestampExtractor 和 BoundedOutOfOrdernessTimestampExtractor 不熟的话，可以参见之前的这篇，就能理解为什么会是这样的语法了。

https://www.jianshu.com/p/c612e95a5028

下面来正式建表。

```
    val createTableSql =
      """
        |上文的SQL语句
        |......
      """.stripMargin
    tableEnv.sqlUpdate(createTableSql)
```

执行完毕后，我们还可以去到 Hive 执行 DESCRIBE FORMATTED ods.streaming_user_active_log 语句，能够发现该表并没有事实上的列，而所有属性（包括 schema、connector、format 等等）都作为元数据记录在了 Hive Metastore 中。

1	# col_name	data_type	comment
2		NULL	NULL
3		NULL	NULL
4	# Detailed Table Information	NULL	NULL
5	Database:	ods	NULL
6	Owner:	null	NULL
7	CreateTime:	Wed Mar 25 22:36:13 CST 2020	NULL
8	LastAccessTime:	UNKNOWN	NULL
9	Protect Mode:	None	NULL
10	Retention:	0	NULL
11	Location:	hdfs://[redacted]/user/hive/warehouse/ods.db/streaming_user_active_log	NULL
12	Table Type:	MANAGED_TABLE	NULL
13	Table Parameters:	NULL	NULL
14		flink.connector.properties.bootstrap.servers	[redacted]
15		flink.connector.properties.group.id	rtdw_group_test_1
16		flink.connector.properties.zookeeper.connect	[redacted]
17		flink.connector.startup-mode	latest-offset
18		flink.connector.topic	ng_log_par_extracted
19		flink.connector.type	kafka
20		flink.connector.version	0.11
21		flink.format.derive-schema	true

flink.generic.table.schema.0.data-type	VARCHAR(2147483647)
flink.generic.table.schema.0.name	eventType
flink.generic.table.schema.1.data-type	VARCHAR(2147483647)
flink.generic.table.schema.1.name	userId
flink.generic.table.schema.10.data-type	TIMESTAMP(3) NOT NULL
flink.generic.table.schema.10.expr	PROCTIME()
flink.generic.table.schema.10.name	procTime
flink.generic.table.schema.11.data-type	TIMESTAMP(3)
flink.generic.table.schema.11.expr	TO_TIMESTAMP(FROM_UNIXTIME(`ts` / 1000, 'yyyy-MM-dd HH:mm:ss'))
flink.generic.table.schema.11.name	eventTime
flink.generic.table.schema.2.data-type	VARCHAR(2147483647)
flink.generic.table.schema.2.name	shareUserId
flink.generic.table.schema.3.data-type	VARCHAR(2147483647)
flink.generic.table.schema.3.name	platform
flink.generic.table.schema.4.data-type	VARCHAR(2147483647)
flink.generic.table.schema.4.name	columnType

Flink SQL 创建的表都会带有一个标记属性 is_generic=true，图中未示出。

开窗计算 PV、UV

用30秒的滚动窗口，按事件类型来分组，查询语句如下。

```
SELECT eventType,
TUMBLE_START(eventTime, INTERVAL '30' SECOND) AS windowStart,
TUMBLE_END(eventTime, INTERVAL '30' SECOND) AS windowEnd,
COUNT(userId) AS pv,
COUNT(DISTINCT userId) AS uv
FROM rtdw.ods.streaming_user_active_log
WHERE platform = 'xyz'
GROUP BY eventType, TUMBLE(eventTime, INTERVAL '30' SECOND)
```

关于窗口在 SQL 里的表达方式请参见官方文档。1.10 版本 SQL 的官方文档写的还是比较可以的。

> SQL 文档：
> https://ci.apache.org/projects/flink/flink-docs-release-1.10/dev/table/sql/queries.html#group-windows

懒得再输出到一个结果表了，直接转换成流打到屏幕上。

```
    val queryActiveSql =
      """
        |......
        |......
      """.stripMargin
    val result = tableEnv.sqlQuery(queryActiveSql)

    result
        .toAppendStream[Row]
        .print()
        .setParallelism(1)
```

敏感数据较多，就不一一截图了。以上是我分享的两个示例，感兴趣的同学也可以动手试试。