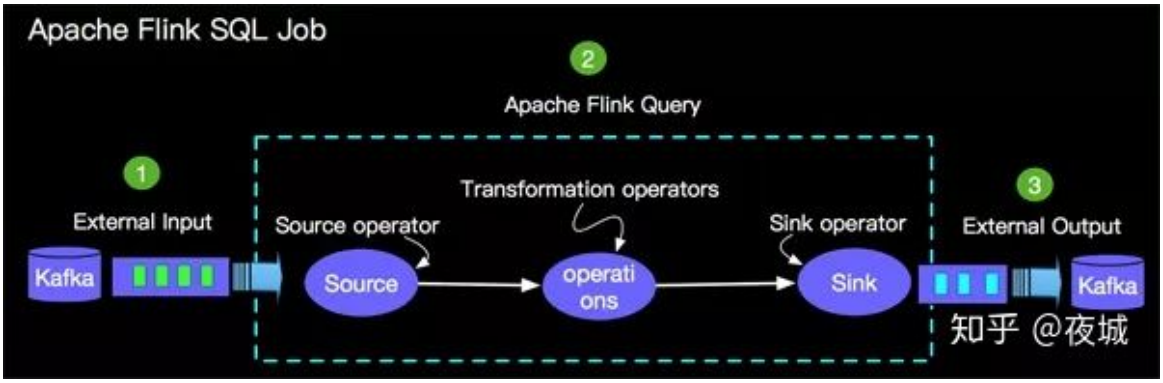
Apache Flink SQL

本篇核心目标是让大家概要了解一个完整的 Apache Flink SQL Job 的组成部分，以及 Apache Flink SQL 所提供的核心算子的语义，最后会应用 TumbleWindow 编写一个 End-to-End 的页面访问的统计示例。

1.Apache Flink SQL Job 的组成

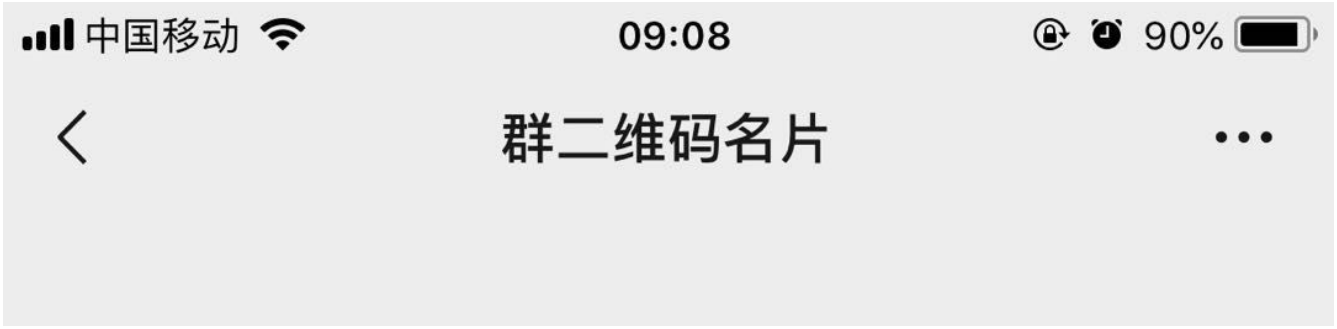
我们做任何数据计算都离不开读取原始数据，计算逻辑和写入计算结果数据三部分，当然基于 ApacheFlink SQL 编写的计算 Job 也离不开这个三部分，如下所所示：



如上所示，一个完整的 Apache Flink SQL Job 由如下三部分：

Source Operator – Soruce operator 是对外部数据源的抽象, 目前 Apache Flink 内置了很多常用的数据源实现，比如上图提到的 Kafka。
Query Operators – 查询算子主要完成如图的 Query Logic，目前支持了Union，Join，Projection,Difference, Intersection 以及 window 等大多数传统数据库支持的操作。
Sink Operator – Sink operator 是对外结果表的抽象，目前 Apache Flink 也内置了很多常用的结果表的抽象，比如上图提到的 Kafka。

大数据Flink讨论群



大数据Flink交流群

该二维码7天内(7月2日前)有效，重新进入将更新

2.Apache Flink SQL 核心算子

SQL 是 StructuredQuevy Language 的缩写，最初是由美国计算机科学家 Donald D. Chamberlin 和 Raymond F. Boyce 在 20 世纪 70 年代早期从 Early History of SQL 中了解关系模型后在 IBM 开发的。该版本最初称为[SEQUEL: A Structured EnglishQuery Language]（结构化英语查询语言），旨在操纵和检索存储在 IBM 原始准关系数据库管理系统 System R 中的数据。直到 1986 年， ANSI 和 ISO 标准组正式采用了标准的”数据库语言 SQL”语言定义。Apache Flink SQL 核心算子的语义设计也参考了 1992 、2011 等 ANSI-SQL 标准。接下来我们将简单为大家介绍 Apache Flink SQL 每一个算子的语义。

2.1 SELECT

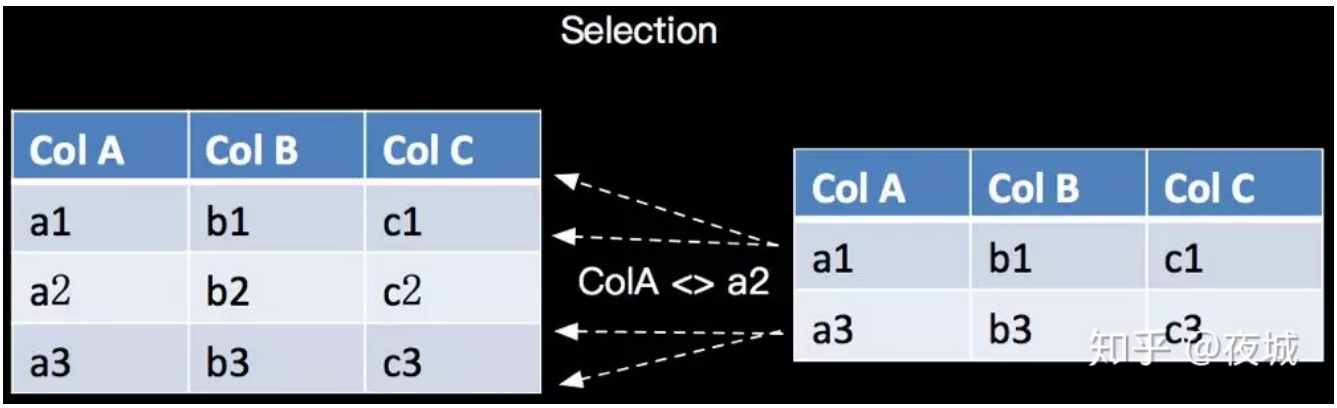
SELECT 用于从数据集/流中选择数据，语法遵循 ANSI-SQL 标准，语义是关系代数中的投影(Projection),对关系进行垂直分割，消去某些列。

一个使用 Select 的语句如下：

```
SELECT ColA, ColC FROME tab ;
```

2.2 WHERE

WHERE 用于从数据集/流中过滤数据，与 SELECT 一起使用，语法遵循 ANSI-SQL 标准，语义是关系代数的 Selection，根据某些条件对关系做水平分割，即选择符合条件的记录，如下所示：

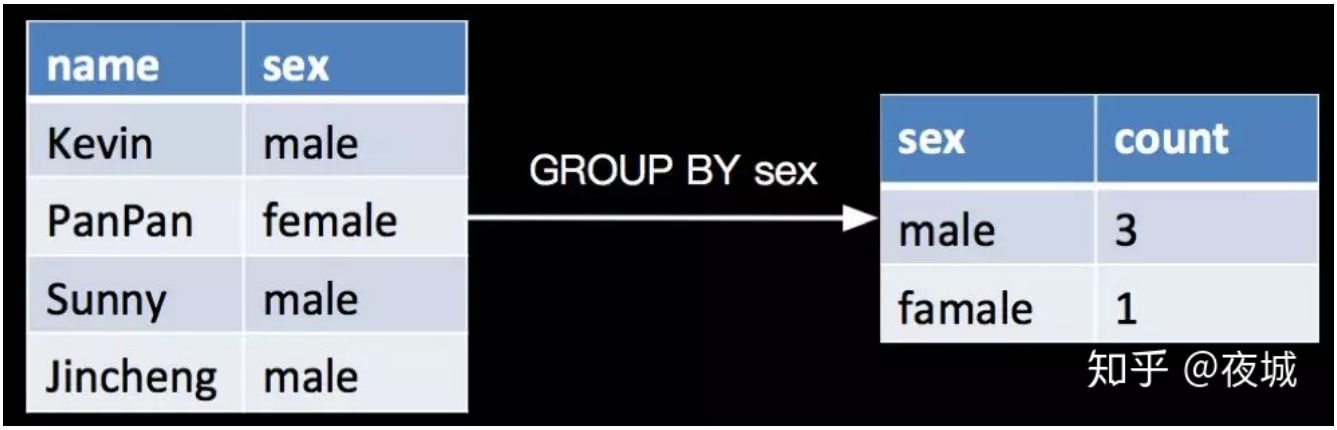


对应的 SQL 语句如下：

```
SELECT * FROM tab WHERE ColA <> 'a2' ;
```

2.3 GROUP BY

GROUP BY 是对数据进行分组的操作，比如我需要分别计算一下一个学生表里面女生和男生的人数分别是多少，如下：

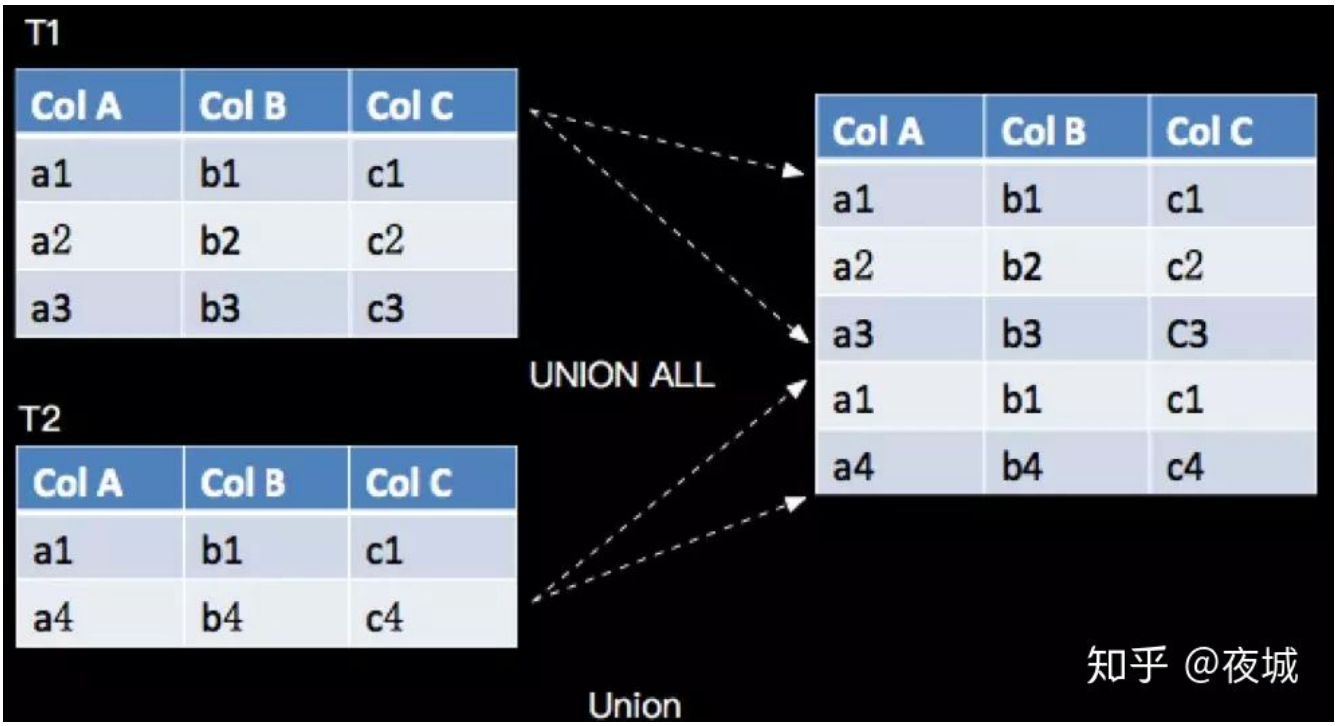


对应的 SQL 语句如下：

```
SELECT sex, COUNT(name) AS count FROM tab GROUP BY sex ;
```

2.4 UNION ALL

UNION ALL 将两个表合并起来，要求两个表的字段完全一致，包括字段类型、字段顺序,语义对应关系代数的 Union，只是关系代数是 Set 集合操作，会有去重复操作，UNION ALL 不进行去重，如下所示：

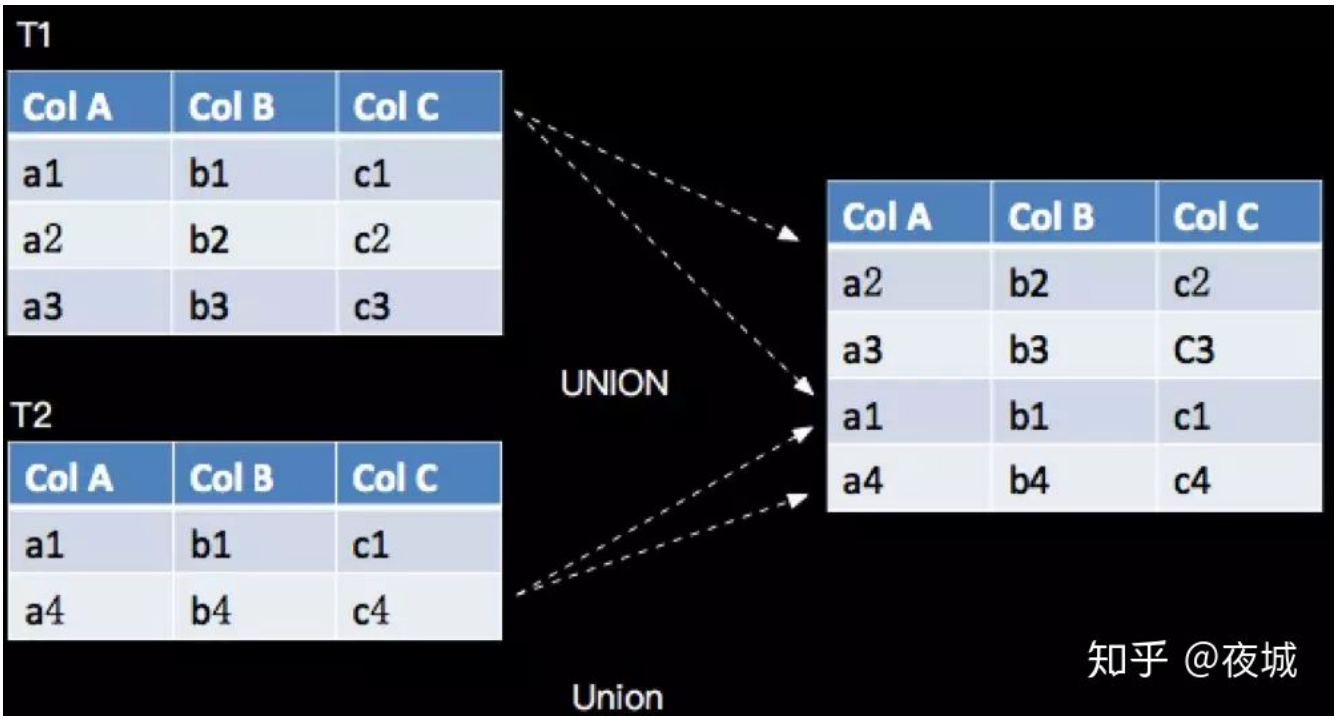


对应的 SQL 语句如下：

```
SELECT * FROM T1 UNION ALL SELECT * FROM T2
```

2.5 UNION

UNION 将两个流给合并起来，要求两个流的字段完全一致，包括字段类型、字段顺序，并其 UNION 不同于 UNION ALL，UNION 会对结果数据去重,与关系代数的 Union 语义一致，如下：



对应的 SQL 语句如下：

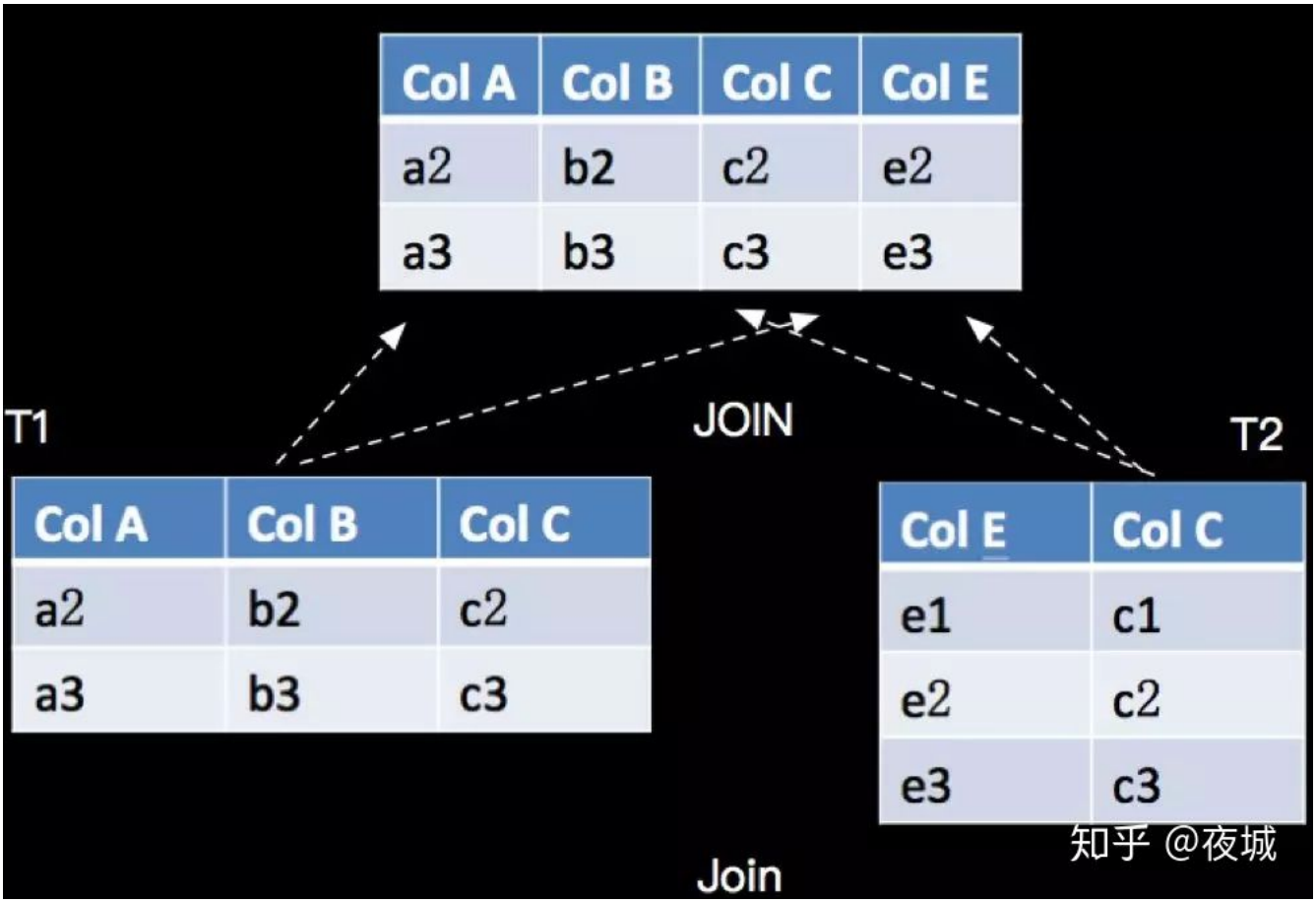
```
SELECT * FROM T1 UNION SELECT * FROM T2
```

2.6 JOIN

JOIN 用于把来自两个表的行联合起来形成一个宽表，Apache Flink 支持的 JOIN 类型：

JOIN – INNER JOIN
LEFT JOIN – LEFT OUTER JOIN
RIGHT JOIN – RIGHT OUTER JOIN
FULL JOIN – FULL OUTER JOIN

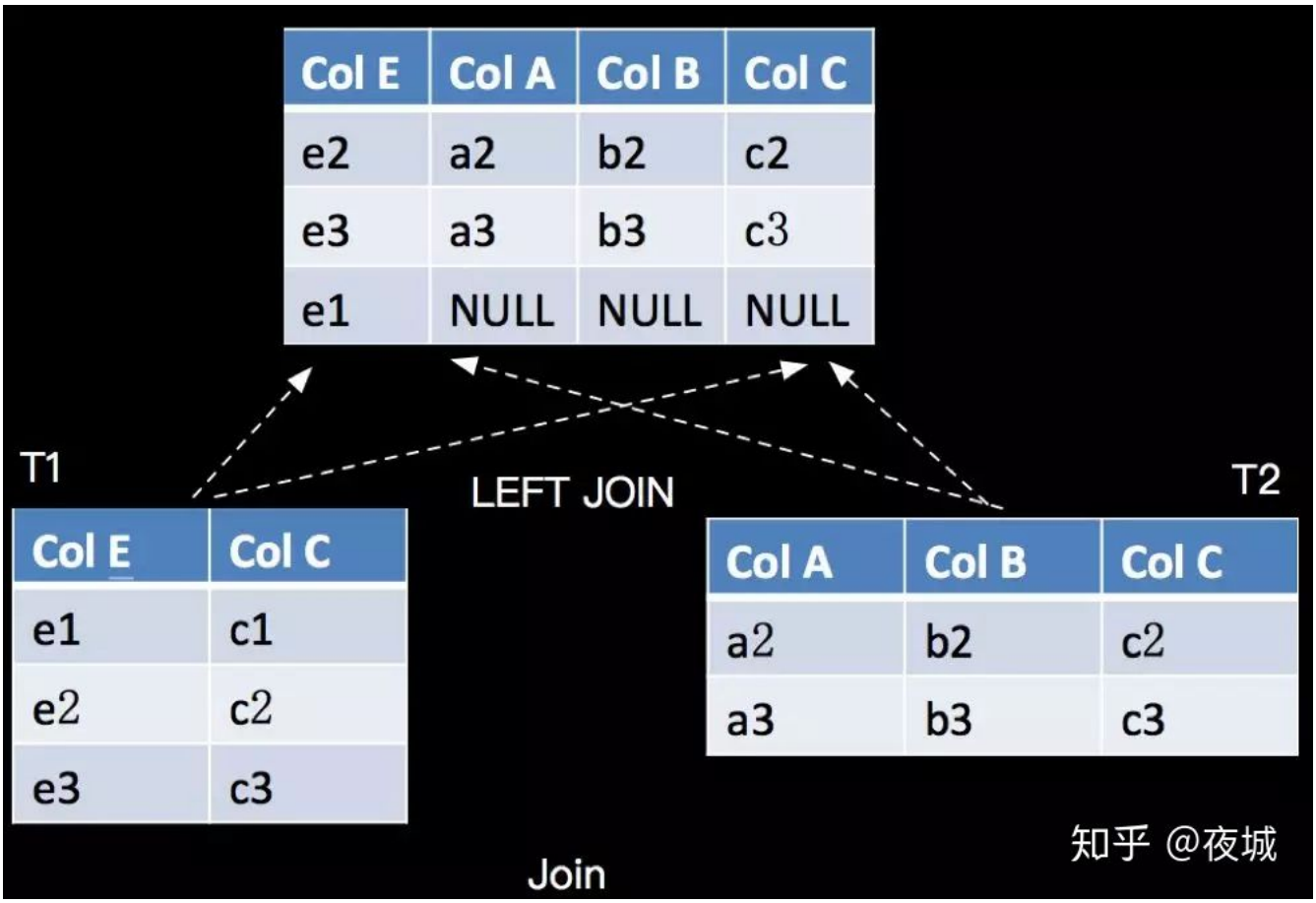
JOIN 与关系代数的 Join 语义相同，具体如下：



对应的 SQL 语句如下(INNERJOIN)：

```
SELECT ColA, ColB, T2.ColC, ColE FROM TI JOIN T2 ON T1.ColC = T2.ColC ;
```

LEFT JOIN 与 INNERJOIN 的区别是当右表没有与左边相 JOIN 的数据时候，右边对应的字段补 NULL 输出，如下：



对应的 SQL 语句如下(LEFTJOIN)：

```
SELECT ColA, ColB, T2.ColC, ColE FROM TI LEFT JOIN T2 ON T1.ColC = T2.ColC ;
```

说明：

细心的读者可能发现上面 T2.ColC 是添加了前缀 T2 了，这里需要说明一下，当两张表有字段名字一样的时候，我需要指定是从那个表里面投影的。
RIGHT JOIN 相当于 LEFT JOIN 左右两个表交互一下位置。FULL JOIN 相当于 RIGHT JOIN 和 LEFT JOIN 之后进行 UNION ALL 操作。

2.7 Window

在 Apache Flink 中有 2 种类型的 Window，一种是 OverWindow，即传统数据库的标准开窗，每一个元素都对应一个窗口。一种是 GroupWindow，目前在SQL中 GroupWindow 都是基于时间进

行窗口划分的。

2.7.1 OverWindow

OVER Window 目前支持由如下三个元素组合的 8 种类型：

时间 – ProcessingTime 和 EventTime
数据集 – Bounded 和 UnBounded
划分方式 – ROWS 和 RANGE 我们以的Bounded ROWS 和 Bounded RANGE 两种常用类型，想大家介绍 Over Window 的语义

Bounded ROWS Over Window

Bounded ROWS OVER Window 每一行元素都视为新的计算行，即，每一行都是一个新的窗口。

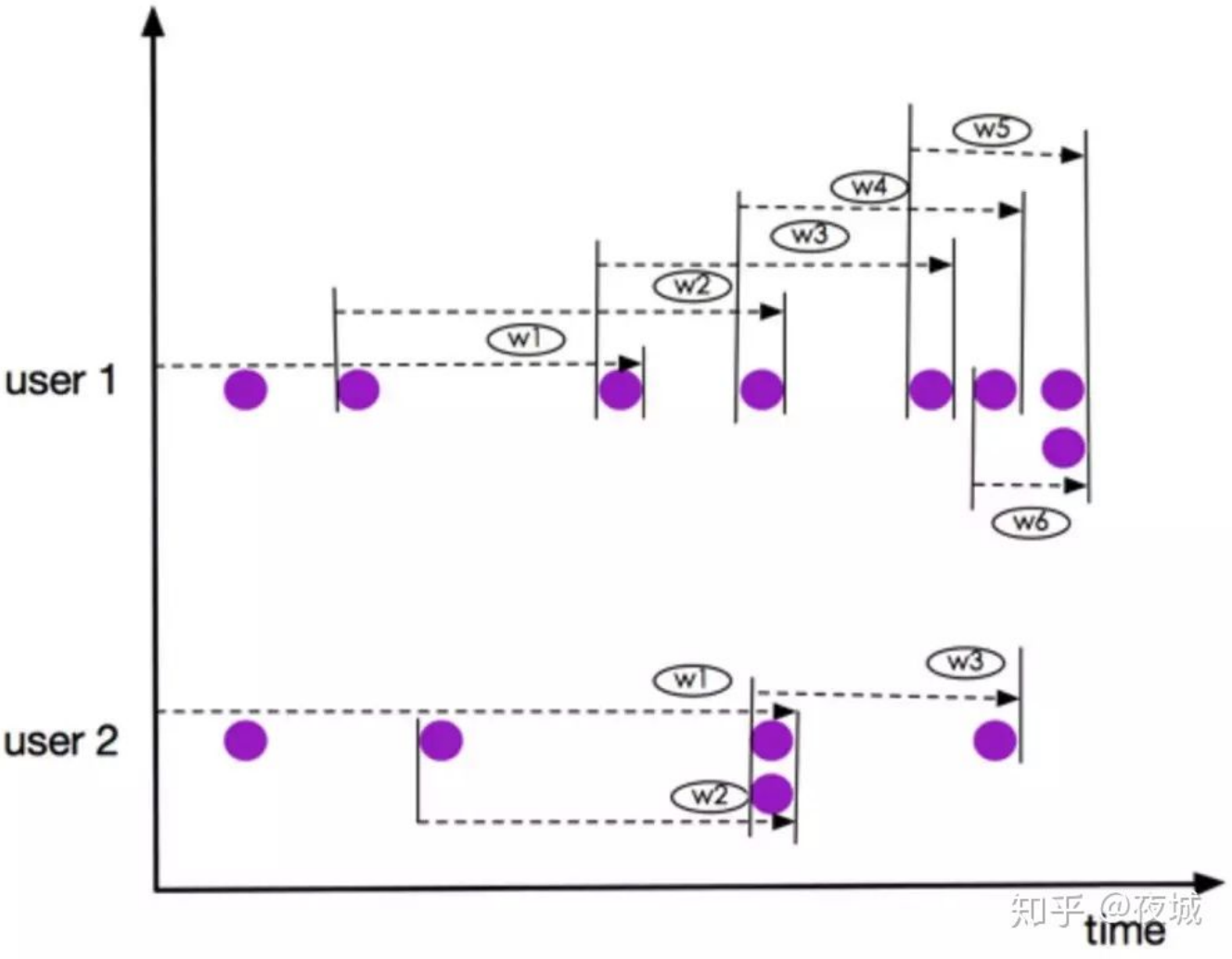
语法

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     ROWS
     BETWEEN (UNBOUNDED | rowCount) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1
```

value_expression – 进行分区的字表达式；
timeCol – 用于元素排序的时间字段；
rowCount – 是定义根据当前行开始向前追溯几行元素；

语义

我们以 3 个元素(2PRECEDING)的窗口为例，如下图:



上图所示窗口 user 1 的 w5 和 w6， user 2 的 窗口 w2 和 w3，虽然有元素都是同一时刻到达，但是他们仍然是在不同的窗口，这一点有别于 RANGEOVER Window.

Bounded RANGE Over Window

Bounded RANGE OVER Window 具有相同时间值的所有元素行视为同一计算行，即，具有相同时间值的所有行都是同一个窗口；

语法

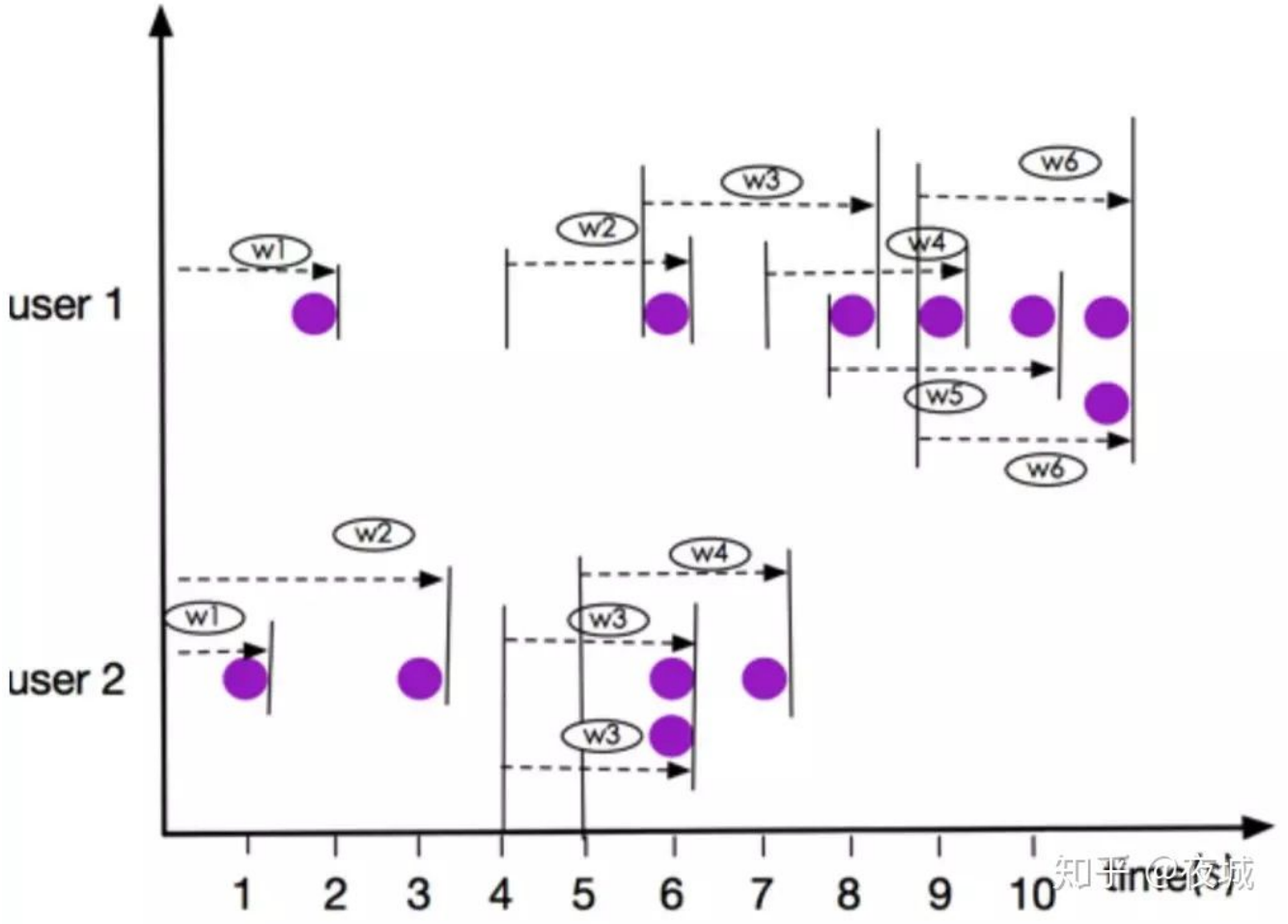
Bounded RANGE OVER Window 的语法如下：

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     RANGE
     BETWEEN (UNBOUNDED | timeInterval) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1
```

value_expression – 进行分区的字表达式；
timeCol – 用于元素排序的时间字段；
timeInterval – 是定义根据当前行开始向前追溯指定时间的元素行；

语义

我们以 3 秒中数据(INTERVAL‘2’ SECOND)的窗口为例，如下图：



注意: 上图所示窗口 user 1 的 w6， user 2的 窗口 w3，元素都是同一时刻到达,他们是在同一个窗口，这一点有别于 ROWS OVER Window.

2.7.2 GroupWindow

根据窗口数据划分的不同，目前 Apache Flink 有如下 3 种 Bounded Winodw:

Tumble – 滚动窗口，窗口数据有固定的大小，窗口数据无叠加；
Hop – 滑动窗口，窗口数据有固定大小，并且有固定的窗口重建频率，窗口数据有叠加；
Session – 会话窗口，窗口数据没有固定的大小，根据窗口数据活跃程度划分窗口，窗口数据无叠加；

说明：Aapche Flink 还支持 UnBounded的 Group Window，也就是全局 Window，流上所有数据都在一个窗口里面，语义非常简单，这里不做详细介绍了。

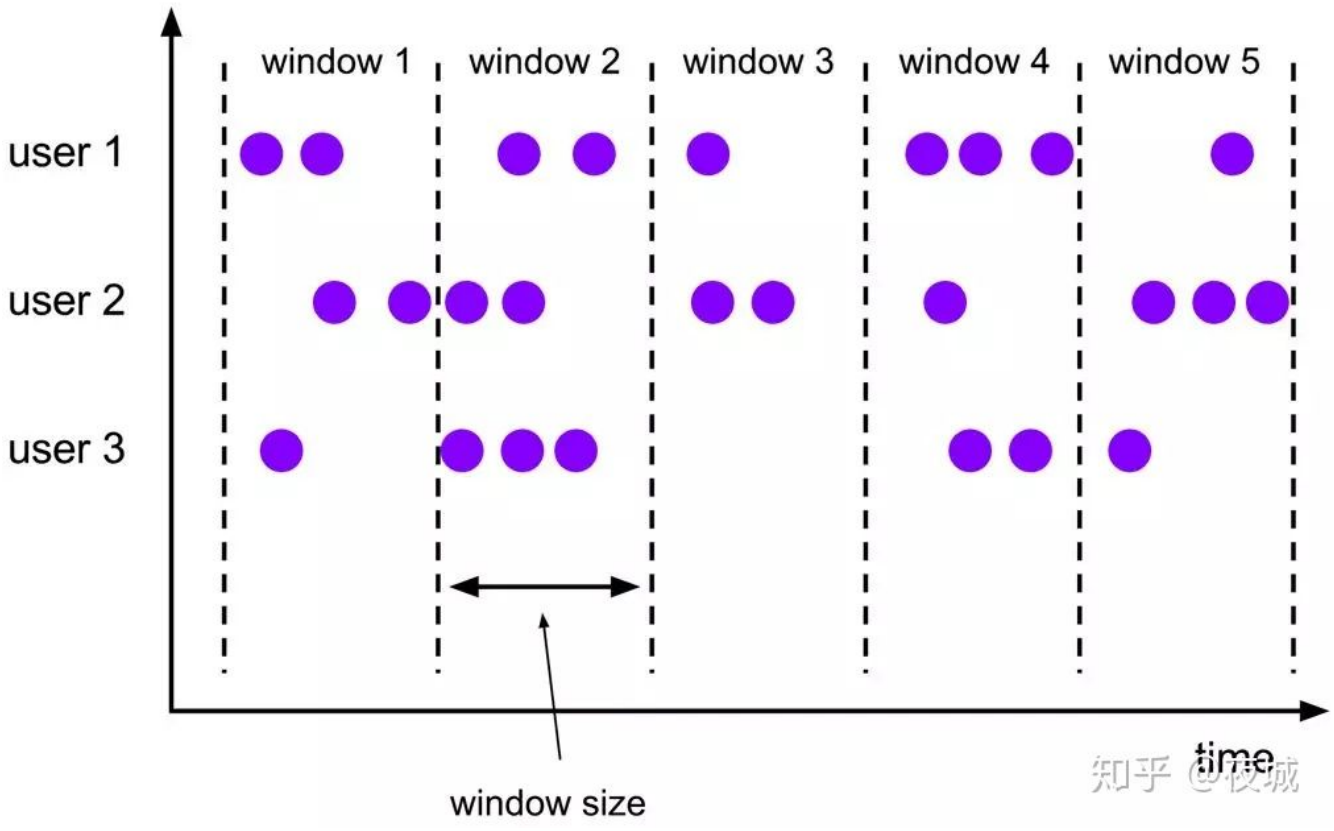
GroupWindow 的语法如下：

```
SELECT
    [gk],
    agg1(col1),
     ...
    aggN(colN)
FROM Tab1
GROUP BY [WINDOW(definition)], [gk]
```

[WINDOW(definition)] – 在具体窗口语义介绍中介绍。

Tumble Window

Tumble 滚动窗口有固定 size，窗口数据不重叠，具体语义如下：

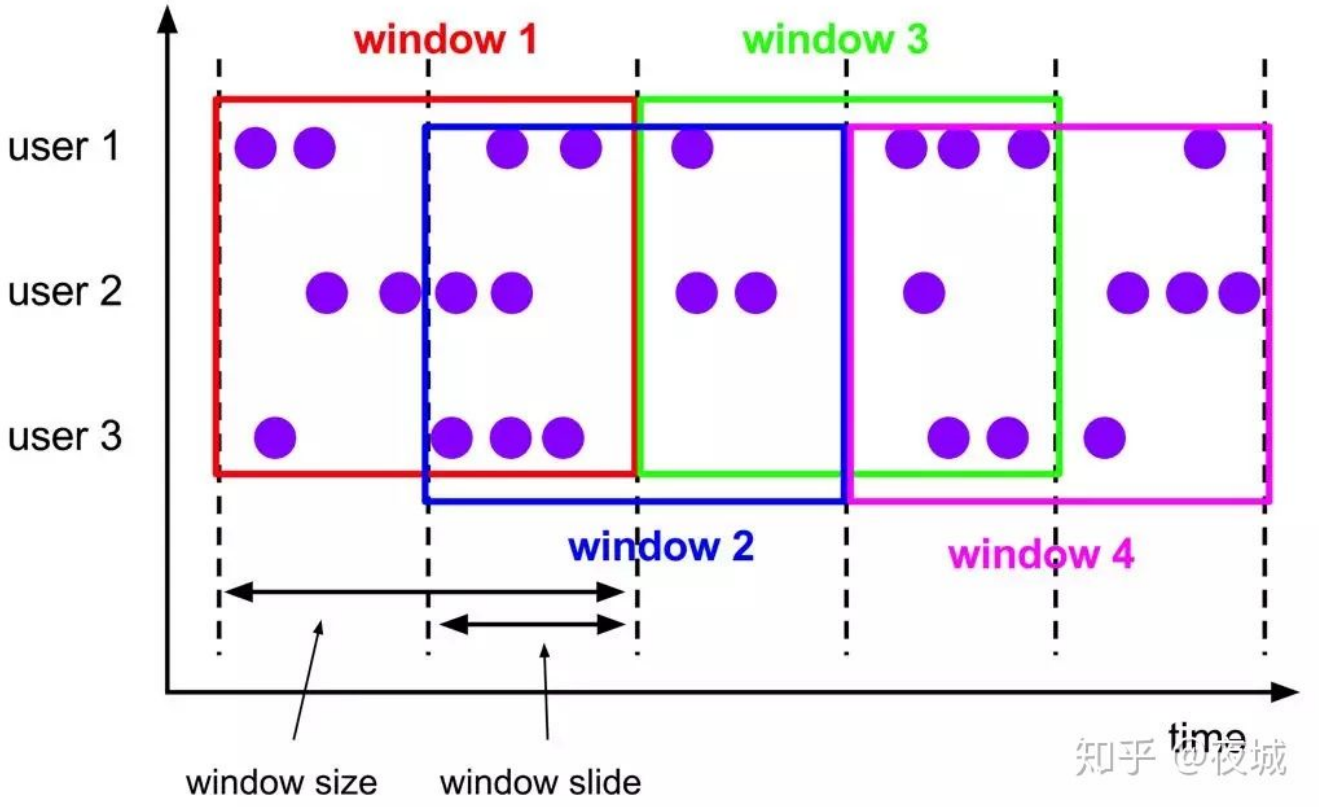


假设我们要写一个 2 分钟大小的 Tumble，示例SQL如下：

```
SELECT gk, COUNT(*) AS pv
  FROM tab
    GROUP BY TUMBLE(rowtime, INTERVAL '2' MINUTE), gk
```

Hop Window

Hop 滑动窗口和滚动窗口类似，窗口有固定的 size，与滚动窗口不同的是滑动窗口可以通过 slide 参数控制滑动窗口的新建频率。因此当 slide 值小于窗口 size 的值的时候多个滑动窗口会重叠，具体语义如下：

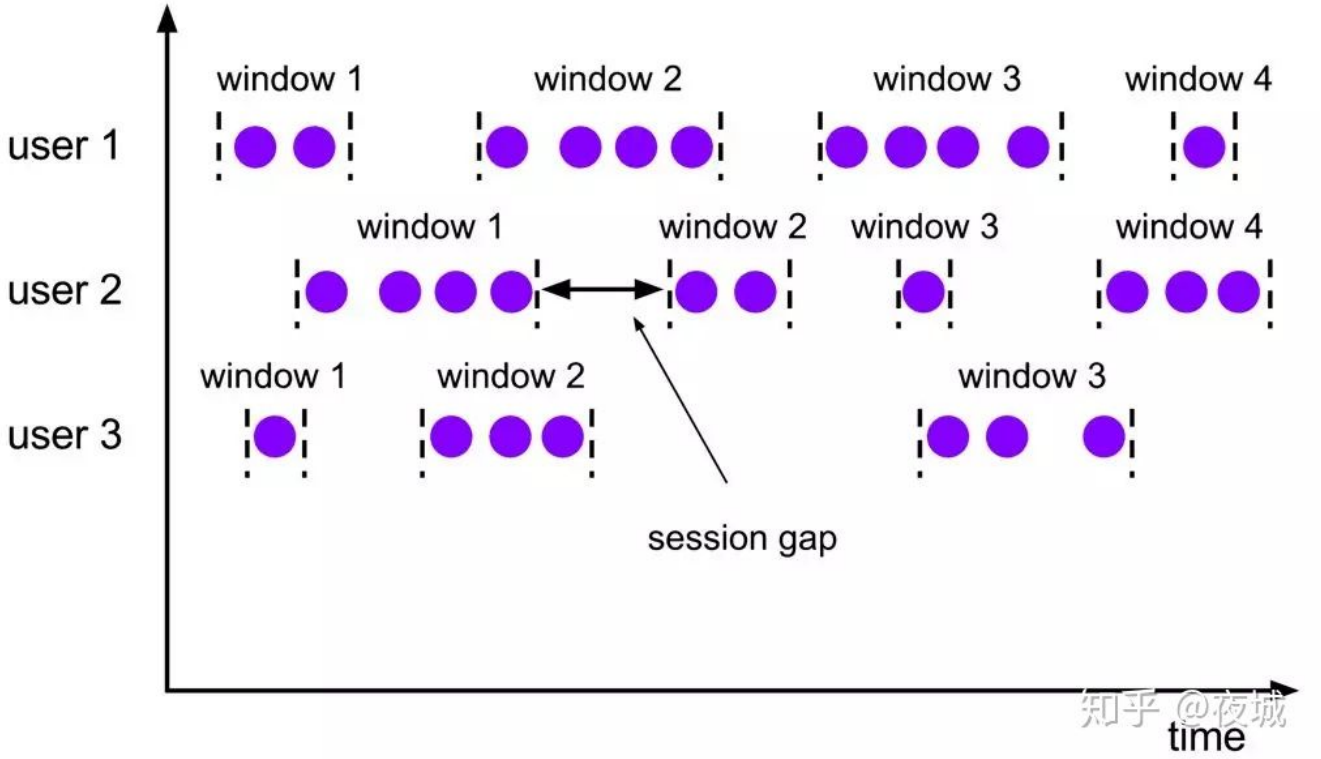


假设我们要写一个统计连续的两个访问用户之间的访问时间间隔不超过 3 分钟的的页面访问量(PV).

```
SELECT gk, COUNT(*) AS pv
  FROM tab
    GROUP BY HOP(rowtime, INTERVAL '5' MINUTE, INTERVAL '10' MINUTE), gk
```

Session Window

Session 会话窗口 是没有固定大小的窗口，通过 session 的活跃度分组元素。不同于滚动窗口和滑动窗口，会话窗口不重叠,也没有固定的起止时间。一个会话窗口在一段时间内没有接收到元素时，即当出现非活跃间隙时关闭。一个会话窗口 分配器通过配置 session gap 来指定非活跃周期的时长,具体语义如下：



假设我们要写一个统计连续的两个访问用户之间的访问时间间隔不超过 3 分钟的的页面访问量(PV).

```
SELECT gk, COUNT(*) AS pv
  FROM pageAccessSession_tab
    GROUP BY SESSION(rowtime, INTERVAL '3' MINUTE), gk
```

说明：很多场景用户需要获得 Window 的开始和结束时间，上面的 GroupWindow的SQL 示例中没有体现，那么窗口的开始和结束时间应该怎样获取呢? Apache Flink 我们提供了如下辅助函数：

TUMBLE_START/TUMBLE_END
HOP_START/HOP_END
SESSION_START/SESSION_END

这些辅助函数如何使用，请参考如下完整示例的使用方式。

3.完整的 SQL Job 案例

上面我们介绍了 Apache Flink SQL 核心算子的语法及语义，这部分将选取Bounded EventTime Tumble Window 为例为大家编写一个完整的包括 Source 和 Sink 定义的 ApacheFlink SQL Job。假设有一张淘宝页面访问表(PageAccess_tab)，有地域，用户 ID 和访问时间。我们需要按不同地域统计每 2 分钟的淘宝首页的访问量(PV). 具体数据如下：

region	userId	accessTime
ShangHai	U0010	2017-11-11 10:01:00
BeiJing	U1001	2017-11-11 10:01:00
BeiJing	U2032	2017-11-11 10:10:00
BeiJing	U1100	2017-11-11 10:11:00
ShangHai	U0011	2017-11-11 12:10:00

3.1 Source 定义

自定义 Apache Flink Stream Source 需要实现 StreamTableSource, StreamTableSource 中通过 StreamExecutionEnvironment 的 addSource 方法获取 DataStream, 所以我们需要自定义一个 SourceFunction, 并且要支持产生 WaterMark，也就是要实现 DefinedRowtimeAttributes 接口。出于代码篇幅问题，我们如下只介绍核心部分，完整代码 请查看: EventTimeTumbleWindowDemo.scala

3.1.1 Source Function 定义

支持接收携带 EventTime 的数据集合，Either 的数据结构 Right 是 WaterMark，Left 是元数据:

```
class MySourceFunction[T](dataList: Seq[Either[(Long, T), Long]]) extends SourceF
  override def run(ctx: SourceContext[T]): Unit = {
    dataList.foreach {
      case Left(t) => ctx.collectWithTimestamp(t._2, t._1)
      case Right(w) => ctx.emitWatermark(new Watermark(w)) // emit watermark
    }
  }
}
```

3.1.2 定义 StreamTableSource

我们自定义的 Source 要携带我们测试的数据，以及对应的 WaterMark 数据，具体如下:

```
class MyTableSource extends StreamTableSource[Row] with DefinedRowtimeAttributes

  // 页面访问表数据 rows with timestamps and watermarks
  val data = Seq(
     // Data
     Left(1510365660000L, Row.of(new JLong(1510365660000L), "ShangHai", "U0010"))
     // Watermark
     Right(1510365660000L),
    ..
  )

  val fieldNames = Array("accessTime", "region", "userId")
  val schema = new TableSchema(fieldNames, Array(Types.SQL_TIMESTAMP, Types.STRIN
  val rowType = new RowTypeInfo(
    Array(Types.LONG, Types.STRING, Types.STRING).asInstanceOf[Array[TypeInformat
    fieldNames)

  override def getDataStream(execEnv: StreamExecutionEnvironment): DataStream[Row
    // 添加数据源实现
    execEnv.addSource(new MySourceFunction[Row](data)).setParallelism(1).returns(
  }
  ...
}
```

3.4 Sink 定义

我们简单的将计算结果写入到 Apache Flink 内置支持的 CSVSink 中，定义 Sink 如下：

```
def getCsvTableSink: TableSink[Row] = {
    val tempFile = ...
    new CsvTableSink(tempFile.getAbsolutePath).configure(
      Array[String]("region", "winStart", "winEnd", "pv"),
```

```
      Array[TypeInformation[_]](Types.STRING, Types.SQL_TIMESTAMP, Types.SQL_TIME
  }
```

3.5 构建主程序

主程序包括执行环境的定义，Source / Sink 的注册以及统计查 SQL 的执行，具体如下：

```
def main(args: Array[String]): Unit = {
    // Streaming 环境
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    val tEnv = TableEnvironment.getTableEnvironment(env)

    // 设置EventTime
    env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)

    //方便我们查出输出数据
    env.setParallelism(1)

    val sourceTableName = "mySource"
    // 创建自定义source数据结构
    val tableSource = new MyTableSource

    val sinkTableName = "csvSink"
    // 创建CSV sink 数据结构
    val tableSink = getCsvTableSink

    // 注册source
    tEnv.registerTableSource(sourceTableName, tableSource)
    // 注册sink
    tEnv.registerTableSink(sinkTableName, tableSink)

    val sql =
      "SELECT  " +
      "  region, " +
      "  TUMBLE_START(accessTime, INTERVAL '2' MINUTE) AS winStart," +
      "  TUMBLE_END(accessTime, INTERVAL '2' MINUTE) AS winEnd, COUNT(region) AS
      " FROM mySource " +
      " GROUP BY TUMBLE(accessTime, INTERVAL '2' MINUTE), region"

    tEnv.sqlQuery(sql).insertInto(sinkTableName);
    env.execute()
  }
```

3.6 执行并查看运行结果

执行主程序后我们会在控制台得到 Sink 的文件路径，如下：

```
Sink path : /var/folders/88/8n406qmx2z73qvrzc_rbtv_r0000gn/T/csv_sink_8025014910
```

Cat 方式查看计算结果，如下：

```
jinchengsunjcdeMacBook-Pro:FlinkTableApiDemo jincheng.sunjc$ cat /var/folders/88/
ShangHai,2017-11-11 02:00:00.0,2017-11-11 02:02:00.0,1
BeiJing,2017-11-11 02:00:00.0,2017-11-11 02:02:00.0,1
BeiJing,2017-11-11 02:10:00.0,2017-11-11 02:12:00.0,2
ShangHai,2017-11-11 04:10:00.0,2017-11-11 04:12:00.0,1
```

4.小结

本篇概要的介绍了 Apache Flink SQL 的所有核心算子，并以一个 End-to-End 的示例展示了如何编写 Apache Flink SQL 的 Job . 希望对大家有所帮助。

孙金城，本期作者；淘宝花名"金竹"，Apache Flink Committer，阿里巴巴高级技术专家。目前就职于阿里巴巴计算平台事业部，自2015年以来一直投入于基于Apache Flink的新一代大数据计算平台实时计算的设计研发工作。

发布于 2019-06-19