

Flink-分组窗口 | Over Windows | SQL 中的 Group Windows | SQL 中的 Over Windows

窗口 (Windows)

1. 时间语义，要配合窗口操作才能发挥作用。最主要的用途，当然就是开窗口、根据时间段做计算了。下面我们就来看看Table API和SQL中，怎么利用时间字段做窗口操作。
2. 在Table API和SQL中，主要有两种窗口：**Group Windows**和**Over Windows**

➤ Group Windows (分组窗口)

- 根据时间或行计数间隔，将行聚合到有限的组 (Group) 中，并对每个组的数据执行一次聚合函数

➤ Over Windows

- 针对每个输入行，计算相邻行范围内的聚合

https://blog.csdn.net/qq_40180229

分组窗口 (Group Windows)

1. Group Windows 是使用 window (w:GroupWindow) 子句定义的，并且必须由as子句指定一个别名。
2. 为了按窗口对表进行分组，窗口的别名必须在 group by 子句中，像常规的分组字段一样引用
3. Table API 提供了一组具有特定语义的预定义 Window 类，这些类会被转换为底层 DataStream 或 DataSet 的窗口操作
4. 分组窗口分为三种：滚动窗口、滑动窗口、会话窗口

滚动窗口 (Tumbling windows) :

1. 滚动窗口 (Tumbling windows) 要用Tumble类来定义
2. over: 定义窗口长度

3. on: 用来分组（按时间间隔）或者排序（按行数）的时间字段
4. as: 别名，必须出现在后面的groupBy中

```
1 // Tumbling Event-time Window (事件时间字段rowtime)
2 .window(Tumble over 10.minutes on 'rowtime as 'w)
3
4 // Tumbling Processing-time Window (处理时间字段proctime)
5 .window(Tumble over 10.minutes on 'proctime as 'w)
6
7 // Tumbling Row-count Window (类似于计数窗口, 按处理时间排序, 10行一组)
8 .window(Tumble over 10.rows on 'proctime as 'w)
9
```

滑动窗口 (Sliding windows) :

1. 滑动窗口 (Sliding windows) 要用Slide类来定义
2. over: 定义窗口长度
3. every: 定义滑动步长
4. on: 用来分组（按时间间隔）或者排序（按行数）的时间字段
5. as: 别名，必须出现在后面的groupBy中

```
1 // Sliding Event-time Window
2 .window(Slide over 10.minutes every 5.minutes on 'rowtime as 'w)
3
4 // Sliding Processing-time window
5 .window(Slide over 10.minutes every 5.minutes on 'proctime as 'w)
6
7 // Sliding Row-count window
8 .window(Slide over 10.rows every 5.rows on 'proctime as 'w)
```

会话窗口 (Session windows) :

1. 会话窗口 (Session windows) 要用Session类来定义
2. withGap: 会话时间间隔
3. on: 用来分组（按时间间隔）或者排序（按行数）的时间字段
4. as: 别名，必须出现在后面的groupBy中

```
1 // Session Event-time Window
2 .window(Session withGap 10.minutes on 'rowtime as 'w)
3
```

```

4
5 // Session Processing-time Window
   .window(Session withGap 10.minutes on 'proctime as 'w)

```

Over Windows

1. Over window 聚合是标准 SQL 中已有的（over 子句），可以在查询的 SELECT 子句中定义
2. Over window 聚合，会针对每个输入行，计算相邻行范围内的聚合
3. Over windows 使用 window（w:overwindows*）子句定义，并在 select（）方法中通过别名来引用
4. Table API 提供了 Over 类，来配置 Over 窗口的属性
5. 可以在事件时间或处理时间，以及指定为时间间隔、或行计数的范围内，定义 Over windows
6. 无界的 over window 是使用常量指定的

```

1 val table = input
2   .window([w: OverWindow] as 'w)
3   .select('a, 'b.sum over 'w, 'c.min over 'w)

```

无界 Over Windows

```

1 // 无界的事件时间over window (时间字段 "rowtime")
2 .window(Over partitionBy 'a orderBy 'rowtime preceding UNBOUNDED_RANGE as
3   s 'w)
4
5 // 无界的处理时间over window (时间字段"proctime")
6 .window(Over partitionBy 'a orderBy 'proctime preceding UNBOUNDED_RANGE
7   as 'w)
8
9 // 无界的事件时间Row-count over window (时间字段 "rowtime")
10 .window(Over partitionBy 'a orderBy 'rowtime preceding UNBOUNDED_ROW as
11   'w)

```

```

// 无界的处理时间Row-count over window (时间字段 "rowtime")
.window(Over partitionBy 'a orderBy 'proctime preceding UNBOUNDED_ROW as
  'w)

```

有界的over window

```

1 // 有界的事件时间over window (时间字段 "rowtime", 之前1分钟)
2 .window(Over partitionBy 'a orderBy 'rowtime preceding 1.minutes as 'w)
3
4 // 有界的处理时间over window (时间字段 "rowtime", 之前1分钟)
5 .window(Over partitionBy 'a orderBy 'proctime preceding 1.minutes as 'w)
6
7 // 有界的事件时间Row-count over window (时间字段 "rowtime", 之前10行)
8 .window(Over partitionBy 'a orderBy 'rowtime preceding 10.rows as 'w)
9
10 // 有界的处理时间Row-count over window (时间字段 "rowtime", 之前10行)
11 .window(Over partitionBy 'a orderBy 'proctime preceding 10.rows as 'w)

```

SQL 中的 Group Windows

另外还有一些辅助函数，可以用来选择Group Window的开始和结束时间戳，以及时间属性。

这里只写TUMBLE_，滑动和会话窗口是类似的（HOP_，SESSION_*）。

1. TUMBLE_START(time_attr, interval)
2. TUMBLE_END(time_attr, interval)
3. TUMBLE_ROWTIME(time_attr, interval)
4. TUMBLE_PROCTIME(time_attr, interval)

- Group Windows 定义在 SQL 查询的 Group By 子句中

➤ TUMBLE(time_attr, interval)

- 定义一个滚动窗口，第一个参数是时间字段，第二个参数是窗口长度

➤ HOP(time_attr, interval, interval)

- 定义一个滑动窗口，第一个参数是时间字段，第二个参数是窗口滑动步长，第三个是窗口长度

➤ SESSION(time_attr, interval)

- 定义一个会话窗口，第一个参数是时间字段，第二个参数是窗口间隔

https://blog.csdn.net/qq_40180229

SQL 中的 Over Windows

1. 用 Over 做窗口聚合时，所有聚合必须在同一窗口上定义，也就是说必须是相同的分区、排序和范围
2. 目前仅支持在当前行范围之前的窗口
3. ORDER BY 必须在单一的时间属性上指定

```
1 SELECT COUNT(amount) OVER (  
2     PARTITION BY user  
3     ORDER BY proctime  
4     ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)  
5 FROM Orders  
6  
7 // 也可以做多个聚合  
8 SELECT COUNT(amount) OVER w, SUM(amount) OVER w  
9 FROM Orders  
10 WINDOW w AS (  
11     PARTITION BY user  
12     ORDER BY proctime  
13     ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
```

代码实操

```
1 import com.atguigu.bean.SensorReading  
2 import org.apache.flink.streaming.api.TimeCharacteristic  
3 import org.apache.flink.streaming.api.functions.timestamps.BoundedOutOfOrdernessTimestampExtractor  
4  
5 import org.apache.flink.streaming.api.scala._  
6 import org.apache.flink.streaming.api.windowing.time.Time  
7 import org.apache.flink.table.api.{Over, Table, Tumble}  
8 import org.apache.flink.table.api.scala._  
9 import org.apache.flink.types.Row  
10  
11 object TimeAndWindowTest {  
12     def main(args: Array[String]): Unit = {  
13         val env = StreamExecutionEnvironment.getExecutionEnvironment  
14         env.setParallelism(1)  
15         env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)  
16  
17         // 创建表执行环境  
18         val tableEnv = StreamTableEnvironment.create(env)  
19  
20         val inputStream: DataStream[String] = env.readTextFile("D:\\MyWork\\  
21 WorkspaceIDEA\\flink-tutorial\\src\\main\\resources\\SensorReading.txt")  
22  
23         // map成样例类类型  
24         val dataStream: DataStream[SensorReading] = inputStream
```

```

25     .map(data => {
26         val dataArray = data.split(",")
27         SensorReading(dataArray(0), dataArray(1).toLong, dataArray(2).to
28 Double)
29     })
30     .assignTimestampsAndWatermarks( new BoundedOutOfOrdernessTimestamp
31 Extractor[SensorReading]
32     (Time.seconds(1)) {
33         override def extractTimestamp(element: SensorReading): Long = el
34 ement.timestamp * 1000L
35     } )
36
37     // 将流转换成表, 直接定义时间字段
38     val sensorTable: Table = tableEnv
39     .fromDataStream(dataStream, 'id, 'temperature, 'timestamp.rowtime
40 as 'ts)
41
42     // 1. Table API
43     // 1.1 Group Window聚合操作
44     val resultTable: Table = sensorTable
45     .window( Tumble over 10.seconds on 'ts as 'tw )
46     .groupBy( 'id, 'tw )
47     .select( 'id, 'id.count, 'tw.end )
48
49     // 1.2 Over Window 聚合操作
50     val overResultTable: Table = sensorTable
51     .window( Over partitionBy 'id orderBy 'ts preceding 2.rows as 'ow
52 )
53     .select( 'id, 'ts, 'id.count over 'ow, 'temperature.avg over 'ow )
54
55     // 2. SQL实现
56     // 2.1 Group Windows
57     tableEnv.createTemporaryView("sensor", sensorTable)
58     val resultSqlTable: Table = tableEnv.sqlQuery(
59         """
60         |select id, count(id), hop_end(ts, interval '4' second, interval
61 '10' second)
62         |from sensor
63         |group by id, hop(ts, interval '4' second, interval '10' second)
64         """.stripMargin)
65
66     // 2.2 Over Window
67     val orderSqlTable: Table = tableEnv.sqlQuery(
68         """
69         |select id, ts, count(id) over w, avg(temperature) over w
70         |from sensor
71         |window w as (
72         |    partition by id
73         |    order by ts

```

```

74         | rows between 2 preceding and current row
75         |)
76         """".stripMargin)
77         // sensorTable.printSchema()
        // 打印输出
        // resultTable.toRetractStream[Row].print("agg")
        // overResultTable.toAppendStream[Row].print("over result")
        orderSqlTable.toAppendStream[Row].print("order sql")

        env.execute("time and window test job")
    }
}

```

The screenshot shows the IntelliJ IDEA IDE with a Flink project. The main editor displays the Scala code for `TimeAndWindowTest.scala`. The code defines a windowed aggregation over sensor data. The output window shows the results of the query, including sensor IDs, timestamps, and aggregated values. The process finished successfully with exit code 0.

```

val orderSqlTable: Table = tableEnv.sqlQuery(
    """
    |select id, ts, count(id) over w, avg(temperature) over w
    |from sensor
    |window w as (
    |  partition by id
    |  order by ts
    |  rows between 2 preceding and current row
    |)
    """".stripMargin)
// sensorTable.printSchema()
// 打印输出
// resultTable.toRetractStream[Row].print("agg")
// overResultTable.toAppendStream[Row].print("over result")
orderSqlTable.toAppendStream[Row].print("order sql")

env.execute(jobName = "time and window test job")

```

Run: TimeAndWindowTest

```

order sql> sensor_1,2019-01-17 09:43:19.0,5,42.03333333333333
order sql> sensor_6,2019-01-17 09:43:21.0,1,15.4
order sql> sensor_7,2019-01-17 09:43:22.0,1,6.7
order sql> sensor_7,2019-01-17 09:43:22.0,2,18.85
order sql> sensor_10,2019-01-17 09:43:25.0,1,38.1
order sql> sensor_10,2019-01-17 09:43:25.0,2,35.150000000000006
order sql> sensor_10,2019-01-17 09:43:25.0,3,35.233333333333334

```

Process finished with exit code 0

只把我当做你的路人甲
不怪你隔着时差