Flink事件时间何时触发窗口计算

目录

思考：

什么时候才会触发窗口计算?

既然使用的是事件时间那么必然会涉及到水位线(water_mark)，水位线在其中扮演的角色是什么?

此时我们带着疑问，一步一步的探究

注意：

1、本篇博客中的所有解释都是在滚动窗口的前提下

2、浏览本博客前观看本栏另外一篇博客“Flink时间概念与水位线”尤佳。

Q：为什么要在滚动窗口的前提下进行解释?

A：因为滚动窗口相比较滑动和会话来说更容易让大家理解，在本篇博客中着重的是讨论水位线在窗口触发下的场景，因此当然采用大家较容易理解的窗口来为大家解释。

Q：那我要是想了解水位线在其他窗口下的场景呢?

A：在本栏的其他博客有详细介绍。

1.1 浏览本博客前你需要了解的知识点

flink内部是如何划分窗口的?

首先Windows的时间范围是一个自然时间范围，比如你定义了一个窗口：timeWindow(Time.seconds(3))；那么其windows会将窗口中的事件按照3S进行划分(左闭右开)

[10:11:00,10:11:03)

[10:11:03,10:11:06)

… …
[10:11:21,10:11:24)
… …

当一个Event Time = 10:11:22的记录到来时就会生成如下窗口，此时这条消息就存放在这个窗口中；
[10:11:21,10:11:24]

触发的条件?

a、water_mark时间 >= window_end_time只是第一个条件
b、在[window_start_time,window_end_time)区间中还需要有数据存在，如果没有数据同样是不会触发的。

何时第一次触发?

当water_marker >= windows_end_time窗口结束时间，就会触发窗口操作。
（最新的water_marker时间戳会在过去的windows_end_time窗口结束时间中逐一进行比较，如果发现有 >= 的情况就会触发窗口操作）

1.2 示例：触发窗口计算

示例
最大乱序事件10秒，窗口时间3秒

第一次触发计算

输入数据:

```
1,1538359882000
2,1538359886000
3,1538359892000
4,1538359893000
5,1538359894000
```

定义下方输出数据中的water_mark

```
  val watermark: DataStream[(String, Long, String, Int)] = inputMap.assi
gnTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks[(String, Lo
ng, String, Int)] {
    val maxOutOfOrderness = 10000L //最大允许的乱序时间是10s
    var currentMaxTimestamp = 0L
    var a: Watermark = _

    override def getCurrentWatermark: Watermark = {
      a = new Watermark(currentMaxTimestamp - maxOutOfOrderness)
      a
    }
```

```
12
13
14      override def extractTimestamp(element: (String, Long, String, Int),
15  previousElementTimestamp: Long): Long = {
16        val timestamp = element._2
17        currentMaxTimestamp = Math.max(timestamp, currentMaxTimestamp)
18        val end = if (!a.toString.contains("-")) {
19          val regEx = "[^0-9]";
20          val p = Pattern.compile(regEx);
21          val m = p.matcher(a.toString);
22          val L_number = m.replaceAll("").trim()
23          format.format(L_number.toLong)
24        } else a.toString
25        println("timestamp:" + element._1 + "," + element._2 + "," + eleme
26  nt._3 + "|" +
27          s"${a.toString}($end)" + "|" + format.format(currentMaxTimestamp
28   - maxOutOfOrderness))
29        val lll: Long = System.currentTimeMillis()

          timestamp
        }
      })
```

当我们依次输入数据的时候，在输入完`5,1538359894000`这条数据后，最新的water_marker是：10:11:23。由于该条记录的水位线为：10:11:24 > 10:11:23，因此水位线会进行更新，变成10:11:24，而10:11:24 >= window end time(10:11:24)，所以此时就会触发计算操作。

此时的触发操作计算的是event_time(事件时间)在[10:11:21,10:11:24)窗口之间的数据`(也就是只会计算 1,1538359882000 这一条数据)`

> 输出数据为：
> 当前数据 | 当前数据的水位线，计算并更新水位线
> timestamp: `1,1538359882000` ,10:11:22|Watermark@-10000(Watermark@-0000),10:11:12
> timestamp: `2,1538359886000` ,10:11:26|Watermark@1538359872000(10:11:12),10:11:16
> timestamp: `3,1538359892000` ,10:11:32|Watermark@1538359876000(10:11:16),10:11:22
> timestamp: `4,1538359893000` ,10:11:33|Watermark@1538359882000(10:11:22), 10:11:23
> timestamp: `5,1538359894000` ,10:11:34|Watermark@1538359883000(10:11:23), 10:11:24
> 第一次触发计算： (1,1538359882000, 10:11:22,1)

何时第二次触发计算

> 输入数据:
> `1,1538359882000`

```
2,1538359886000
3,1538359892000
4,1538359893000
5,1538359894000
6,1538359896000
7,1538359897000
```

输出数据为：
timestamp:1,1538359882000,10:11:22|Watermark@-10000(Watermark@-0000),10:11:12
timestamp:2,1538359886000,10:11:26|Watermark@1538359872000(10:11:12),10:11:16
timestamp:3,1538359892000,10:11:32|Watermark@1538359876000(10:11:16),10:11:22
timestamp:4,1538359893000,10:11:33|Watermark@1538359882000(10:11:22), 10:11:23
timestamp:5,1538359894000,10:11:34|Watermark@1538359883000(10:11:23), 10:11:24
第一次触发计算：> (1,1538359882000, 10:11:22,1)
timestamp: `6,1538359896000` ,10:11:36|Watermark@1538359884000(10:11:24),10:11:26
timestamp: `7,1538359897000` ,10:11:37|Watermark@1538359886000(10:11:26), 10:11:27
第二次触发计算：> (2,1538359886000, 10:11:26,1)

第一次的窗口[10:11:21,10:11:24)已经被使用了，那么程序下次触发的窗口则是[10:11:24,10:11:27)，因此我输入的`7,1538359897000`这条数据，而该条数据中的water_mark则会更新成10:11:27而10:11:27 >= window_end_time(10:11:27)，因此会触发操作。

（此时计算的是[10:11:24,10:11:27)窗口内的数据，在上面的输入数据中只有2,1538359886000这条数据属于这个窗口，因此计算的是该条数据）

何时触发多个窗口计算?

输入数据：
```
1,1538359882000
2,1538359886000
3,1538359892000
4,1538359893000
5,1538359894000
6,1538359907000
```

输出数据：
timestamp:1,1538359882000,10:11:22|Watermark@-10000(Watermark@-0000),10:11:12
timestamp:2,1538359886000,10:11:26|Watermark@1538359872000(10:11:12),10:11:16
timestamp:3,1538359892000,10:11:32|Watermark@1538359876000(10:11:16),10:11:22
timestamp:4,1538359893000,10:11:33|Watermark@1538359882000(10:11:22), 10:11:23
timestamp:5,1538359894000,10:11:34|Watermark@1538359883000(10:11:23), 10:11:24
第一次触发计算：> (1,1538359882000, 10:11:22,1)
timestamp: `6,1538359907000` ,10:11:47|Watermark@1538359884000(10:11:24),10:11:37

```
触发计算：> (2,1538359886000, 10:11:26,1)
触发计算：> (3,1538359892000, 10:11:32,1)
触发计算：> (4,1538359893000, 10:11:33,1)
触发计算：> (5,1538359894000, 10:11:34,1)
```

`窗口[10:11:21,10:11:24)之间的数据已经在第一次进行了触发 后面的 6,1538359907000 数据之所以会导致窗口操作是因为:10:11:37 >= 窗口时间中的windows_end_Time [10:11:34,10:11:37);`

而此时在[10:11:34,10:11:37)这个窗口之上还有

[10:11:21,10:11:24)已经在第一次进行触发了
[10:11:24,10:11:27)
[10:11:27,10:11:31)
[10:11:31,10:11:34)
[10:11:34,10:11:37)
… ….

由于timestamp 2,3,4,5的event_time 分别在这些窗口之间，因此输出的话就会在这次触发操作中全部进行输出了；计算的是每个窗口内的数据。