第 5 章 时间与窗口

学习目标

- 熟悉时间概念，能够描述事件时间和处理时间的区别。
- 熟悉窗口分类，能够描述不同窗口类型的作用。
- 掌握键控窗口和非键控窗口，能够说出这两种窗口计算方式的区别。
- 掌握窗口分配器，能够在 DataStream 程序中定义不同类型的窗口分配器。
- 掌握窗口函数，能够灵活运用不同类型的窗口函数处理窗口内的数据。
- 熟悉什么是水位线，能够详细说出水位线的作用。
- 掌握水位线的使用，能够在 DataStream 程序中灵活应用水位线处理数据乱序。
- 了解窗口触发器的应用，能够在 DataStream 程序中自定义窗口触发器。
- 了解窗口驱逐器的应用，能够在 DataStream 程序中使用内置驱逐器和自定义驱逐器。
- 了解处理延迟数据的方式，能够在 DataStream 程序中使用不同的方式处理延迟数据。

Flink 是一个擅长处理无界流的分布式处理引擎，无界流的特点是数据无休止地产生。在面对无界流的处理时，我们不可能等到所有数据都到达后再进行处理，因此更有效的做法是将无界流切分为若干有限的数据集进行处理，这就是所谓的窗口，为了确保每个数据集切分的精确控制，在 Flink 中通常以时间作为衡量标准进行数据集的切分。本章针对 Flink 的时间与窗口进行详细讲解。

5.1 时间概念

时间对于我们来说是一个再熟悉不过的概念，它是一种宝贵的资源，无论是在工作中还是在学习中，每一分每一秒都代表着生活的独特瞬间。我们应该珍视这样的时光，避免无谓的浪费，并尽可能地充实和利用好这些时间。

日常生活中的每个事件在发生时都拥有其自身的时间属性，借助事件的时间属性可以判断事件发生的先后顺序。例如，小明在 12:00:00 打开了电视，在 12:10:00 吃了一颗糖，此时可以看作打开电视和吃糖是两个事件，12:00:00 和 12:10:00 可以看作这两个事件自身的时间属性，通过事件自身的时间属性，便可以判断小明是先打开电视再吃糖，那么时间和流处理有什么关系呢？

对于流处理而言，其主要功能是无休止的处理无界流，无界流中的每条数据都可以看作一个单独的事件，那么通过流处理对无界流进行处理时，同样可以借助数据的时间属性判断数据产生的先后顺序，以此确保窗口在划分无界流的数据时可以更加精确。

Flink 根据时间产生的位置将数据的时间属性分为事件时间(Event Producer)和处理时间(Processing Time)两种类型,那么什么是时间产生的位置呢? 这里还是通过一个生活中的小例子进行说明,例如,小明给同学发送了一条短信,短信在小明的手机上点击发送时会产生一个时间,发送到小明同学的手机时同样会产生一个时间,这两个时间都可以看作该短信的时间属性,只不过产生的位置不同。为了使读者更好地理解 Flink 中事件时间和处理时间产生的位置,先来看 Flink 流处理的运行流程,如图 5-1 所示。

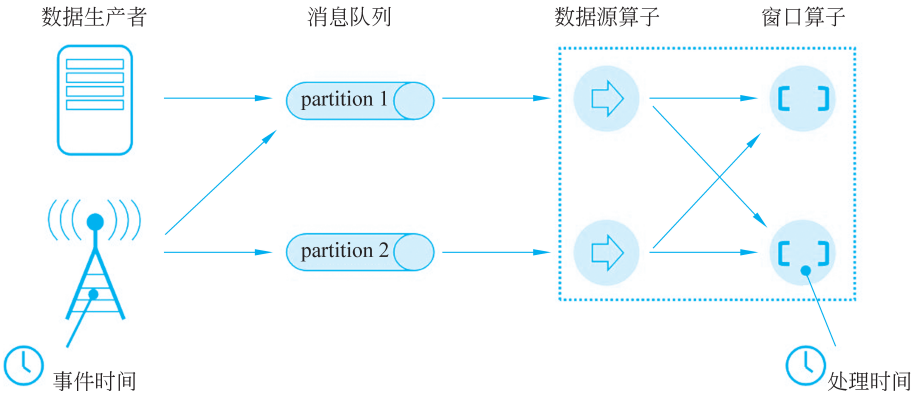

图 5-1　Flink 流处理的运行流程

从图 5-1 可以看出,数据由数据生产者产生,数据生产者可能是服务器、移动设备等,数据经由消息队列传输到 Flink 进行流处理,其中数据在数据生产者产生时的时间属性就是事件时间;数据传输到 Flink 的窗口算子进行流处理时的时间属性就是处理时间。接下来分别对事件时间和处理时间进行介绍,具体内容如下。

1. 事件时间

事件时间指的是无界流的数据在对应的数据生成者产生时的时间属性,也就是数据实际生成的时间。事件时间可以作为一个属性嵌入数据中,并随着数据一并传输到 Flink,因此可以看出,数据一旦生成,那么事件时间自然就确认了。

2. 处理时间

处理时间指的是无界流的数据传输到 Flink 的窗口算子进行处理时产生的时间属性,也就是数据实际被处理的时间。处理时间取决于进行窗口算子操作的服务器系统时间。

那么在实际应用中,该如何从这两种时间属性进行选择呢? 接下来举一个实际应用场景中经常使用的案例进行说明,电商网站经常会对用户的访问量进行统计,假设某电商网站要统计每天的用户访问量,如果某个用户在 23:59:59 访问了该网站,但是该访问记录传输 Flink 的窗口算子进行处理时已经是第二天的 00:00:01 了,那么该用户的访问记录,是应该统计到当天用户的访问量? 还是统计到第二天的用户访问量呢? 很明显,该用户的访问记录应该被统计到当天用户的访问量,不过如果使用处理时间,那么该用户的访问记录会被统计到第二天的用户访问量,这就会导致用户访问量的统计结果与实际情况产生偏差。

通过上面的例子不难发现,使用基于事件时间的窗口算子处理无界流时,其处理结果更加符合实际的业务逻辑,并且处理结果更加精准,所以在实际应用中是较为常用的,不过处理时间并不是一无是处,由于处理时间并不会通过数据产生时嵌入的时间属性判断数据产生的先后顺序,可以看作数据一旦传输到窗口算子就立即进行处理,所以基于处理时间的窗口算子对

无界流进行处理时，效率会更高，通常处理时间用在实时性要求高，而对处理结果准确性要求不高的应用场景。

5.2 窗口分类

在 Flink 中，流处理主要是对无界流进行处理，而无界流的特点是数据无休止地产生，因此不可能等到所有数据都到达之后才进行处理，而是在接收无界流的同时对数据进行处理。通过基础的 DataStream API 实现的 DataStream 程序对无界流进行处理时，只能针对当前处理的结果与无界流传输的新数据进行处理，并输出处理结果，以此来周而复始地处理无界流，但是现实中无界流中的数据并不是缓慢地一条一条传输到 DataStream 程序，通常是同一时间有大量的数据传输到 DataStream 程序，这样频繁地输出结果就会给服务器带来很大的负担。

因此，更加高效的做法是，将无界流划分为多个有界流，并针对每个有界流进行处理，此时的处理结果是针对每个有界流内的所有数据，从而减少了输出处理结果的次数，此过程中产生的每个有界流都视为一个单独的窗口（Window），对于有界流的处理就视为窗口操作。按照不同角度划分，Flink 中的窗口类型是不一样的，下面介绍两种窗口分类方式。

1. 按照驱动类型分类

按照驱动类型分类，可以将窗口分为时间窗口（Time Window）和计数窗口（Count Window），其中时间窗口以时间为衡量标准，按照时间去划分无界流；计数窗口以数量作为衡量标准，按照数据的数量划分无界流。其具体介绍如下。

1）时间窗口

时间窗口可以看作固定时间间隔发车的长途汽车，每经过一段固定的时间，便会有一辆长途汽车离开长途汽车站，每趟长途汽车内乘客的数量是不固定的，但是每趟长途汽车发车的时间间隔是固定的。同样的时间窗口在划分无界流的时候也是如此，DataStream 程序每运行一段固定的时间便生成一个窗口，每个窗口由起始时间和结束时间组成，Flink 通过窗口的起始时间和结束时间之间的时间差来描述窗口大小（Window Size），每个窗口内数据集的大小是不固定的，不过每个窗口的窗口大小是固定的，因此可以理解为时间窗口是无界流某一时间段的数据。

那么使用时间窗口划分无界流的数据时，如何确保每个窗口被分配的数据是其对应时间段内的数据呢？这时就需要借助数据的时间属性来判断每个窗口应该分配的数据，也就是说，每个窗口仅被分配时间属性符合该窗口对应时间段内的数据，不过时间属性等于窗口结束时间的数据并不会被分配到当前窗口中。无界流的数据被分配到窗口的过程中，当数据的时间属性大于或等于某个窗口的结束时间时，便认为无界流中属于当前窗口的数据全部分配完成，此时会触发该窗口关闭，当窗口内的数据计算完成后，便将该窗口销毁。例如，每间隔 5 分钟生成一个窗口的示意图，如图 5-2 所示。

从图 5-2 可以看出，每个窗口都包含了无界流某一时间段的数据，例如，窗口 2 包含了无界流中时间段为[08:05:00,08:10:00)的数据，时间属性为 08:05:15、08:07:36 和 08:08:27 的数据会被分配到窗口 2 中，而时间属性为 08:10:00 的数据并没有分配到该窗口内，并且当无界流中出现时间属性为 08:10:00 的数据时，会触发窗口 2 的关闭。

2）计数窗口

计数窗口可以看作满员就发车的长途汽车，每当长途汽车内的座位坐满乘客时，长途汽车

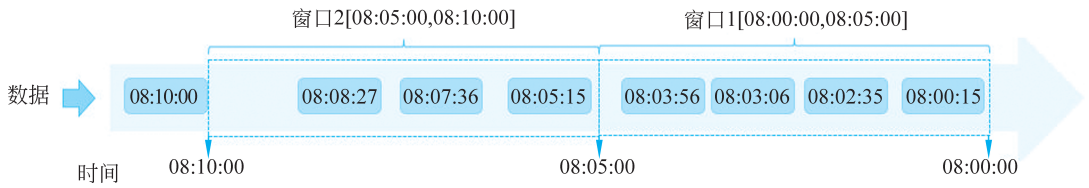

图 5-2　时间窗口

便会离开长途汽车站，每趟长途汽车内乘客的数量是固定的，但是每趟长途汽车发车的时间间隔是不固定的。同样的计数窗口在切分无界流的时候也是如此，DataStream 程序每接收到固定数量的数据便生成一个窗口，因此每个窗口内数据集的大小是固定的，不过每个窗口生成的时间间隔是不固定的。例如，每接收到 8 条数据生成一个窗口的示意图，如图 5-3 所示。

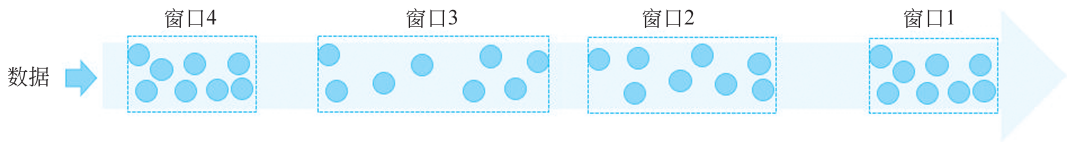

图 5-3　计数窗口

从图 5-3 可以看出，当 DataStream 程序接收到无界流中的第 1 条数据时便开启一个窗口，即窗口 1，待窗口 1 内达到 8 条数据时，便关闭窗口 1，当 DataStream 程序再次接收到无界流的数据时，便会再开启一个窗口，即窗口 2，待窗口 2 内达到 8 条数据时，便关闭窗口 2，以此类推。由此可以推断出，每个窗口生成的时间间隔，以及每个窗口的等待时间，实际上取决于 DataStream 程序接收无界流中每条数据的时间间隔。

需要注意的是，计数窗口在划分无界流时与数据的时间属性无关，一旦出现数据乱序的现象，那么将无法保证无界流中的数据会按照产生的顺序分配到对应的窗口进行处理，这将导致窗口中数据的执行结果不准确，因此在实际应用场景中，为了确保处理结果的准确性，通常使用时间窗口对无界流进行处理。本章后续内容主要以时间窗口为主进行介绍。

2. 按照数据分配规则分类

按照数据分配规则，可以将窗口分为滚动窗口、滑动窗口、会话窗口和全局窗口，具体介绍如下。

1）滚动窗口（Tumbling Windows）

滚动窗口按照固定窗口大小划分窗口，每个窗口之间“无缝衔接”，不存在重叠也不存在间隔，因此无界流的每条数据只会分配到一个窗口。之前举例的时间窗口，使用的数据分配规则就是滚动窗口，在时间窗口中使用滚动窗口划分窗口时，窗口大小就是窗口起始时间和结束时间之间的时间差。例如，时间窗口按照固定窗口大小为 5 分钟的滚动窗口划分窗口的示意图如图 5-4 所示。

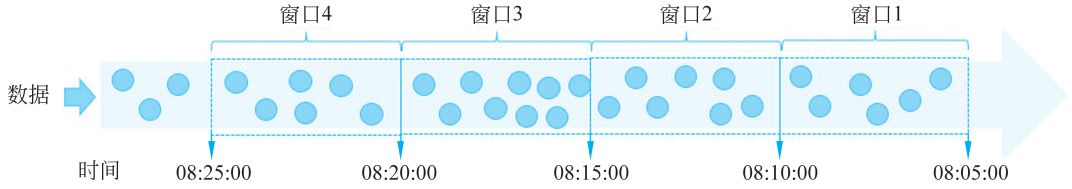

图 5-4　滚动窗口

从图 5-4 可以看出，每个窗口的窗口大小为 5 分钟，它们之间不存在重叠也不存在间隔，无界流的每条数据只会分配到一个窗口中。滚动窗口是最简单的数据分配规则，同时应用也最为广泛。

2）滑动窗口（Sliding Windows）

滑动窗口按照固定窗口大小和滑动步长（Slide）划分窗口，其中滑动步长用于控制窗口向前滑动的长度，在时间窗口中，滑动的长度可以看作当前窗口的起始时间，到下一个窗口起始时间的时间间隔，由于窗口大小是固定的，所以滑动的长度也可以看作当前窗口的结束时间，到下一个窗口结束时间的时间间隔。

通过滑动窗口划分窗口时，每个窗口之间会存在重叠或间隔，这取决于滑动步长和窗口之间的大小关系，如果滑动步长小于窗口大小，则每个窗口之间会存在重叠，无界流中的部分数据会被分配到多个窗口中。例如，时间窗口按照固定窗口大小 15 分钟，滑动步长为 5 分钟的滑动窗口划分窗口的示意图如图 5-5 所示。

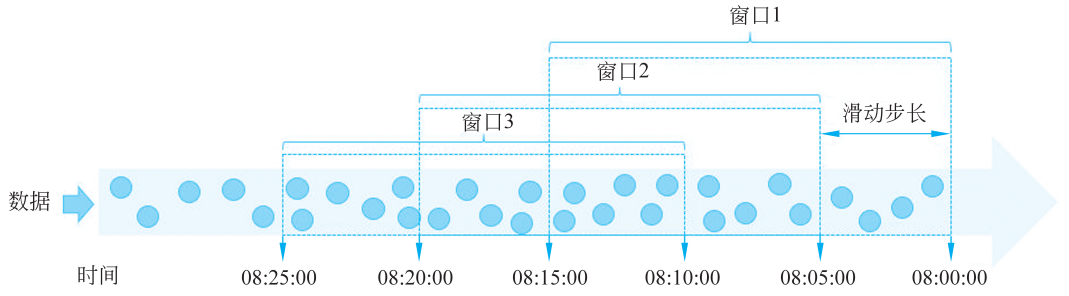

图 5-5 滑动窗口（1）

从图 5-5 可以看出，每个窗口都存在重叠的部分，无界流的部分数据会被分配到多个窗口中。例如，无界流中时间属性在 08:10:00 和 08:15:00 之间的数据会被分配到窗口 1、窗口 2 和窗口 3，因此这 3 个窗口在进行窗口计算时，会出现无界流的这部分数据被重复计算 3 次。

如果滑动步长大于窗口大小，则每个窗口之间会存在间隔，无界流中的部分数据不会被分配到任何窗口中。例如，时间窗口按照固定窗口大小 10 分钟，滑动步长为 15 分钟的滑动窗口划分窗口的示意图如图 5-6 所示。

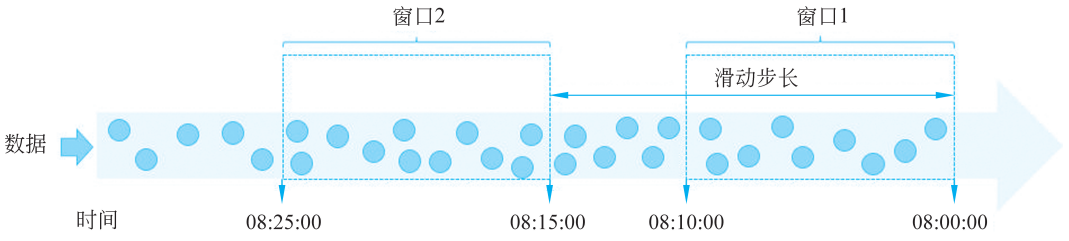

图 5-6 滑动窗口（2）

从图 5-6 可以看出，每个窗口之间都存在间隔，这个间隔的时长取决于滑动步长与窗口大小的差值，无界流的部分数据不会被分配到任何窗口。例如，无界流中时间属性在 08:10:00 和 08:15:00 之间的数据既没有被分配到窗口 1，也没有被分配到窗口 2，因此这部分无界流的数据会丢失。

如果滑动步长等于窗口大小，那么实际上和滚动窗口没有区别，每个窗口之间不存在重叠也不存在间隔，无界流的每条数据只会分配到一个窗口。例如，时间窗口按照固定窗口大小 5 分钟，滑动步长为 5 分钟的滑动窗口划分窗口的示意图与图 5-4 一致。

由于滑动步长等于窗口大小时，与滚动窗口效果一致，滑动步长大于窗口大小时，会出现数据丢失，所以通常情况下，使用滑动窗口时，滑动步长要小于窗口大小，并且滑动步长与窗口大小之间最好保持整数倍的关系。

3）会话窗口（Session Windows）

会话窗口根据会话间隙（Session Gap）划分窗口，所谓会话间隙是指 DataStream 程序的窗口算子接收到无界流传输的两条数据时，这两条数据时间属性的时间差。当窗口算子接收到无界流传输的数据时，会为每条数据生成一个窗口，每个窗口的起始时间为数据的时间属性，并且根据会话间隙确定每个窗口的结束时间，如果两个窗口之间存在交集，即两条数据时间属性的时间差没有超过会话间隙，则将两个窗口合并为一个新的窗口；如果新的窗口与再次生成的窗口之间不存在交集，即两条数据时间属性的时间差超过会话间隙，则新的窗口会自动关闭；以此重复进行创建窗口和合并窗口的操作。有关会话窗口划分窗口的示意图如图 5-7 所示。

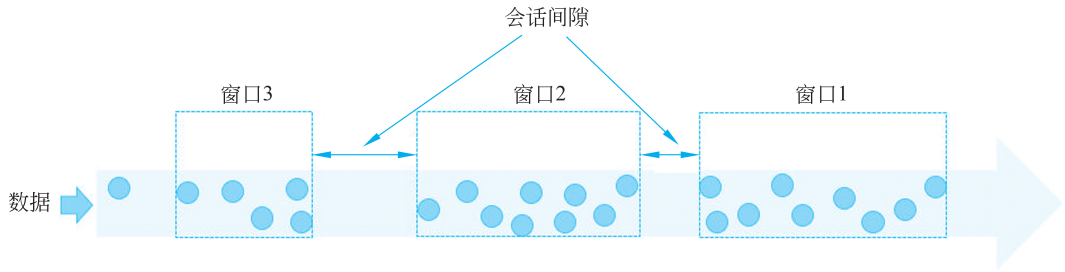

图 5-7　会话窗口

从图 5-7 可以看出，每个窗口的起始时间、结束时间、窗口大小和数据量都是不固定的，虽然每个窗口之间存在间隔，但不会像滑动窗口那样出现数据丢失的问题。需要注意的是，如果会话间隙设置过大，或者无界流的数据比较紧密，会出现窗口一直在合并。

4）全局窗口（Global Windows）

全局窗口将无界流的所有数据都分配到一个窗口中进行处理，除非用户自定义触发器，否则窗口不会关闭，如图 5-8 所示。

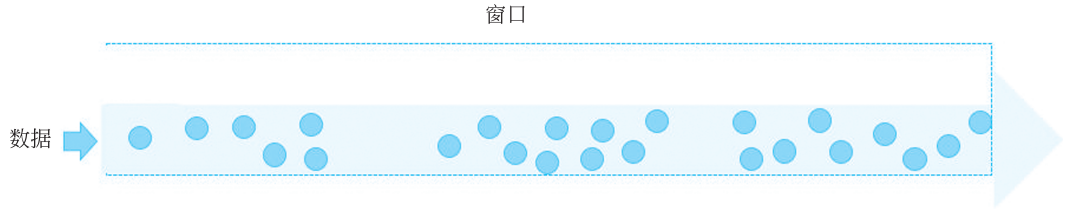

图 5-8　全局窗口

从图 5-8 可以看出，使用全局窗口划分的窗口并没有结束时间。

5.3 键控和非键控窗口

窗口计算是窗口对无界流进行处理的核心，在 DataStream 程序中，按照计算的无界流是否被分区，可以将窗口计算的方式分为键控窗口（Keyed Windows）和非键控窗口（Non-Keyed Windows），其中键控窗口用于对分区后的无界流进行计算；非键控窗口直接对无界流进行计算，5.2 节演示的示意图都是基于非键控窗口。接下来以时间窗口的键控窗口和非键控窗口为例进行详细讲解。

1. 键控窗口

使用键控窗口的方式执行窗口计算之前，首先需要使用转换算子 keyBy 将无界流按照指定 key 进行分区；然后再调用窗口算子 window 执行窗口计算，每个分区内的数据会单独占用一个任务执行计算，多个分区之间执行并行计算（除非并行度为 1），最后得到每个窗口中不同分区内数据的计算结果。键控窗口的示意图如图 5-9 所示。

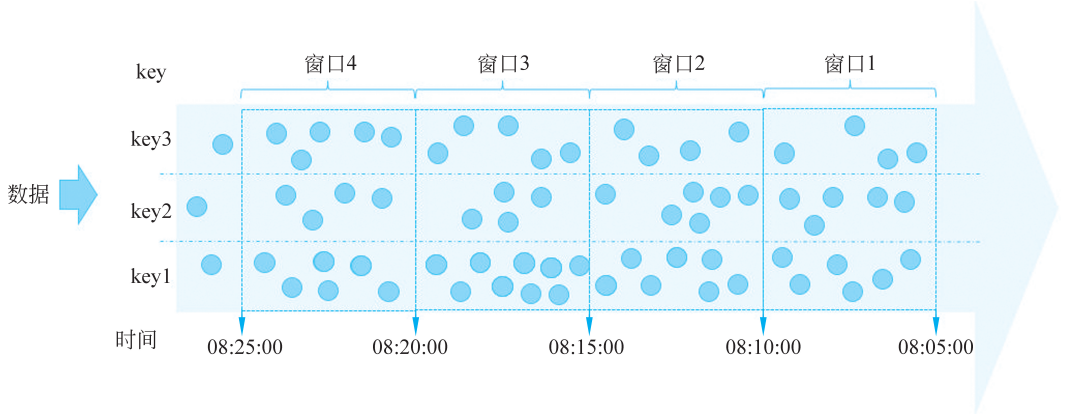

图 5-9 键控窗口的示意图

从图 5-9 可以看出，每个窗口内的数据根据指定 key 被分为 3 个分区，即 key1、key2 和 key3，在执行窗口计算时，每个分区会并行计算包含的数据，得出不同分区的计算结果。

在 DataStream 程序中，以键控窗口的计算方式执行窗口计算的语法格式如下。

```
keyBy(…)     //分区操作
.window(…)   //窗口分配器
[.trigger(…)]  //窗口触发器
[.evictor(…)]  //窗口驱逐器
[.allowedLateness(…)]  //延迟处理数据
[.sideOutputLateData(…)]  //指定旁路输出
.reduce()/aggregate()/process()     //窗口函数
```

针对上述程序结构进行如下讲解。

- window(…)：用于通过窗口算子 window 指定窗口分配器（WindowAssigner），窗口分配器用于指定无界流中数据的分配规则。
- trigger(…)：可选，用于指定窗口触发器，默认情况下，除全局窗口分配器之外，其他窗口分配器都有默认触发器，不需要单独配置，如果窗口分配器的默认触发器无法满足实际应用场景，则可以自定义触发器。

- evictor(…)：可选，用于指定窗口驱逐器，通过自定义窗口驱逐器，可以在无界流的数据进入窗口函数之前移除指定数据，也可以在无界流的数据进入窗口函数之后移除指定数据，默认情况下窗口驱逐器不移除无界流的任何数据。
- allowedLateness(…)：可选，用于指定是否延迟处理数据，默认情况不延迟处理，所谓延迟处理数据是指延长窗口的结束时间，从而等待属于该窗口但延迟到达的数据。
- sideOutputLateData(…)：可选，用于指定 Side Outputs(旁路输出)的标签，该标签可以收集被遗弃的延迟到达数据。
- reduce()/aggregate()/process()：用于指定窗口函数，窗口函数可以根据指定逻辑对窗口内的数据进行计算。

2. 非键控窗口

使用非键控窗口的计算方式执行窗口计算时，直接调用窗口算子 windowsAll 对无界流进行窗口计算即可，每个窗口占用一个任务执行计算，由于 5.2 节演示的示意图都是基于非键控窗口，所以这里不再通过示意图展示。在 DataStream 程序中，以非键控窗口的计算方式执行窗口计算的语法格式如下。

```
windowAll(…)
[.trigger(…)]
[.evictor(…)]
[.allowedLateness(…)]
[.sideOutputLateData(…)]
.reduce()/aggregate()/process()
```

上述程序结构中，windowAll(…)用于通过窗口算子 windowAll 指定窗口分配器，其他内容可参照基于键控窗口的计算方式执行计算的语法格式介绍。

多学一招：计数窗口实现不同窗口计算方式

在 DataStream 程序中，计数窗口以键控窗口的计算方式执行窗口计算的语法格式如下。

```
keyBy(…)
.countWindow(size,slide)
[.trigger(…)]
[.evictor(…)]
[.allowedLateness(…)]
[.sideOutputLateData(…)]
.reduce()/aggregate()/process()
```

上述程序结构中，countWindow(…)表示通过窗口算子 countWindow 指定窗口分配器，其中 size 用于指定窗口大小，即每个窗口的数据量；slide 用于指定滑动步长，即 DataStream 程序接收无界流传输多少条数据时，开启新的窗口，默认 slide 的值等于 size 的值，即 slide 为可选值。例如，size 为 10，slide 为 5，表示每当 DataStream 程序接收无界流传输的 5 条数据时，便开启一个窗口，DataStream 程序会向该窗口分配最近接收的 10 条数据，由此可以看出，计数窗口仅支持滚动窗口和滑动窗口的数据分配规则。

在 DataStream 程序中，计数窗口以非键控窗口的计算方式执行窗口计算的语法格式如下。

```
countWindowAll(size,slide)
[.trigger(…)]
[.evictor(…)]
[.allowedLateness(…)]
[.sideOutputLateData(…)]
.reduce()/aggregate()/process()
```

上述程序结构中，countWindowAll(…)表示通过窗口算子 countWindowAll 指定窗口分配器。

5.4 窗口分配器

在 DataStream 程序中，执行窗口计算的第一步是在构建窗口算子时定义窗口分配器，窗口分配器用于将数据分配到不同的窗口中进行处理。Flink 提供了 4 种类型的窗口分配器，每一种窗口分配器对应了一种窗口类型，即滚动窗口、滑动窗口、会话窗口和全局窗口，其中滚动窗口、滑动窗口和会话窗口类型的窗口分配器可以根据数据的时间属性，定义数据应该被分配到哪个窗口中。接下来详细介绍如何在 DataStream 程序中构建时间窗口的窗口算子时，定义不同类型的窗口分配器，具体内容如下。

1. 定义滚动窗口类型的窗口分配器

由于数据的时间属性分为事件时间和处理时间，所以滚动窗口类型的窗口分配器分为 TumblingEventTimeWindows 和 TumblingProcessingTimeWindows 两种类型，前者用于根据数据的事件时间分配数据；后者用于根据数据的处理时间分配数据。有关在 DataStream 程序中以不同计算方式执行窗口计算时，定义滚动窗口类型的窗口分配器的语法格式如下。

```
//以键控窗口的计算方式执行窗口计算，根据数据的事件时间分配数据
keyBy(…)
.window(TumblingEventTimeWindows.of(size))
……
//以键控窗口的计算方式执行窗口计算，根据数据的处理时间分配数据
keyBy(…)
.window(TumblingProcessingTimeWindows.of(size))
……
//以非键控窗口的计算方式执行窗口计算，根据数据的事件时间分配数据
windowAll(TumblingEventTimeWindows.of(size))
……
//以非键控窗口的计算方式执行窗口计算，根据数据的处理时间分配数据
windowAll(TumblingProcessingTimeWindows.of(size))
……
```

上述示例代码中，size 用于指定窗口大小，可选时间单位为天、时、分、秒和毫秒，对应的实现方式分别是 Time.days(days)、Time.hours(hours)、Time.minutes(minutes)、Time.seconds(seconds)和 Time.milliseconds(milliseconds)。例如，指定窗口大小为 5 秒，则实现方式为 Time.seconds(5)。

2. 定义滑动窗口类型的窗口分配器

滑动窗口类型的窗口分配器同样根据数据的时间属性分为 SlidingEventTimeWindows 和 SlidingProcessingTimeWindows 两种类型，前者用于根据数据的事件时间分配数据，后者

用于根据数据的处理时间分配数据。有关在 DataStream 程序中以不同计算方式执行窗口计算时，定义滑动窗口类型的窗口分配器的语法格式如下。

```
//以键控窗口的计算方式执行窗口计算，根据数据的事件时间分配数据
keyBy(…)
.window(SlidingEventTimeWindows.of(size,slide))
……
//以键控窗口的计算方式执行窗口计算，根据数据的处理时间分配数据
keyBy(…)
.window(SlidingProcessingTimeWindows.of(size,slide))
……
//以非键控窗口的计算方式执行窗口计算，根据数据的事件时间分配数据
windowAll(SlidingEventTimeWindows.of(size,slide))
……
//以非键控窗口的计算方式执行窗口计算，根据数据的处理时间分配数据
windowAll(SlidingProcessingTimeWindows.of(size,slide))
……
```

上述代码中，size 用于指定窗口大小；slide 用于指定滑动步长。size 和 slide 的可选时间单位为天、时、分、秒和毫秒，实现方式与定义滚动窗口类型分配器时 size 的实现方式一致。

3. 定义会话窗口类型的窗口分配器

会话窗口类型的窗口分配器同样根据数据的时间属性分为 EventTimeSessionWindows 和 ProcessingTimeSessionWindows 两种类型，前者用于根据数据的事件时间分配数据，后者用于根据数据的处理时间分配数据。有关在 DataStream 程序中以不同计算方式执行窗口计算时，定义会话窗口类型的窗口分配器的语法格式如下。

```
//以键控窗口的计算方式执行窗口计算，根据数据的事件时间分配数据
keyBy(…)
.window(EventTimeSessionWindows.withGap(size))
……
//以键控窗口的计算方式执行窗口计算，根据数据的处理时间分配数据
keyBy(…)
.window(ProcessingTimeSessionWindows.withGap(size))
……
//以非键控窗口的计算方式执行窗口计算，根据数据的事件时间分配数据
windowAll(EventTimeSessionWindows.withGap(size))
……
//以非键控窗口的计算方式执行窗口计算，根据数据的处理时间分配数据
windowAll(ProcessingTimeSessionWindows.withGap(size))
……
```

上述代码中，size 用于指定会话间隙。size 可选时间单位为天、时、分、秒和毫秒，实现方式与定义滚动窗口类型分配器时 size 的实现方式一致。

4. 定义全局窗口类型的窗口分配器

全局窗口类型的窗口分配器为 GlobalWindows，它是计数窗口的底层实现。由于全局窗口只包含一个窗口，所以无法根据数据的时间属性来分配数据。有关在 DataStream 程序中以不同计算方式执行窗口计算时，定义全局窗口类型的窗口分配器的语法格式如下。