toc: true title: 《从0到1学习Flink》—— 介绍Flink中的Stream Windows date: 2018-12-08 tags:

- Flink
- 大数据
- 流式计算

前言

目前有许多数据分析的场景从批处理到流处理的演变， 虽然可以将批处理作为流处理的特殊情况来处理，但是分析无穷集的流数据通常需要思维方式的转变并且具有其自己的术语（例如，“windowing（窗口化）”、“at-least-once（至少一次）”、“exactly-once（只有一次）”）。

对于刚刚接触流处理的人来说，这种转变和新术语可能会非常混乱。 Apache Flink 是一个为生产环境而生的流处理器，具有易于使用的 API，可以用于定义高级流分析程序。

Flink 的 API 在数据流上具有非常灵活的窗口定义，使其在其他开源流处理框架中脱颖而出。

在这篇文章中，我们将讨论用于流处理的窗口的概念，介绍 Flink 的内置窗口，并解释它对自定义窗口语义的支持。

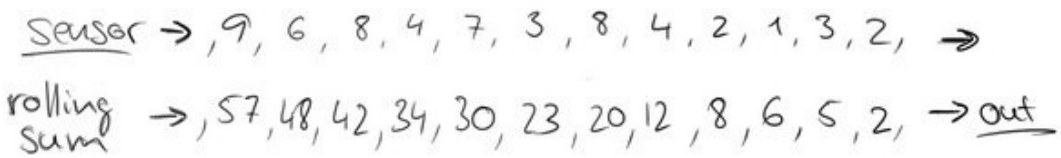
什么是 Windows?

下面我们结合一个现实的例子来说明。

就拿交通传感器的示例：统计经过某红绿灯的汽车数量之和?

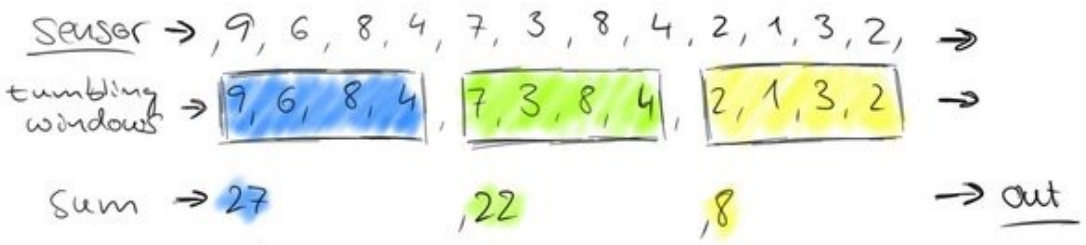
假设在一个红绿灯处，我们每隔 15 秒统计一次通过此红绿灯的汽车数量，如下图：

可以把汽车的经过看成一个流，无穷的流，不断有汽车经过此红绿灯，因此无法统计总共的汽车数量。但是，我们可以换一种思路，每隔 15 秒，我们都将与上一次的结果进行 sum 操作（滑动聚合），如下：



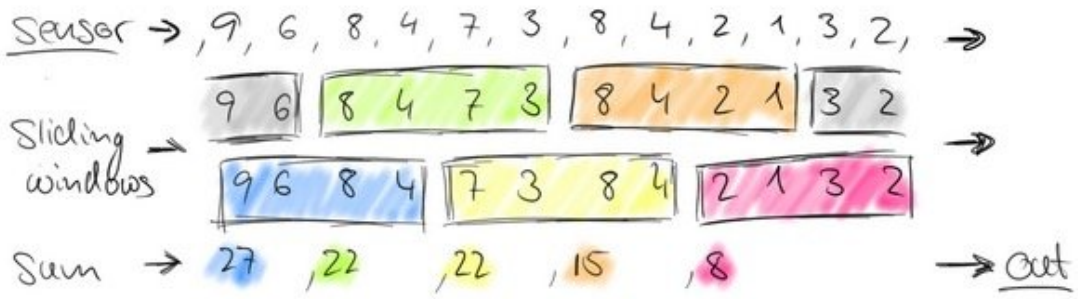
这个结果似乎还是无法回答我们的问题，根本原因在于流是无界的，我们不能限制流，但可以在有一个有界的范围内处理无界的流数据。

因此，我们需要换一个问题的提法：每分钟经过某红绿灯的汽车数量之和？ 这个问题，就相当于一个定义了一个 Window（窗口），window 的界限是1分钟，且每分钟内的数据互不干扰，因此也可以称为翻滚（不重合）窗口，如下图：



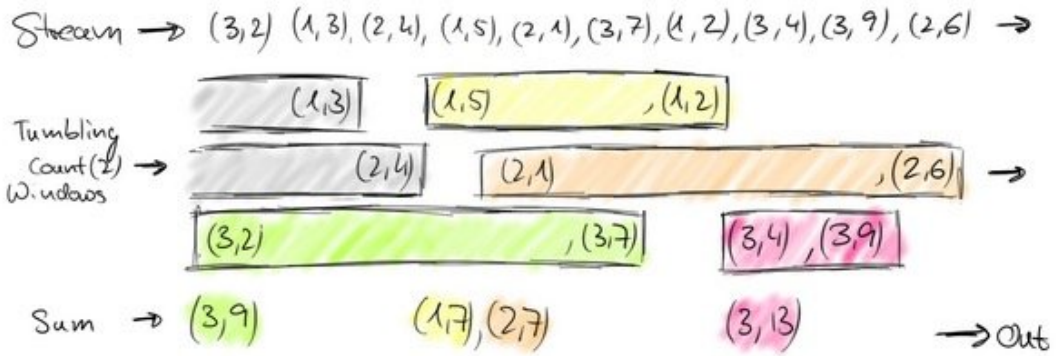
第一分钟的数量为8，第二分钟是22，第三分钟是27。。。这样，1个小时内会有60个window。

再考虑一种情况，每30秒统计一次过去1分钟的汽车数量之和：



此时，window 出现了重合。这样，1个小时内会有120个 window。

扩展一下，我们可以在某个地区，收集每一个红绿灯处汽车经过的数量，然后每个红绿灯处都做一次基于1分钟的window统计，即并行处理：



它有什么作用?

通常来讲，Window 就是用来对一个无限的流设置一个有限的集合，在有界的数据集上进行操作的一种机制。window 又可以分为基于时间（Time-based）的 window 以及基于数量（Count-based）的 window。

Flink 自带的 window

Flink DataStream API 提供了 Time 和 Count 的 window，同时增加了基于 Session 的 window。同时，由于某些特殊的需要，DataStream API 也提供了定制化的 window 操作，供用户自定义 window。

下面，主要介绍 Time-Based window 以及 Count-Based window，以及自定义的 window 操作，Session-Based Window 操作将会在后续的文章中讲到。

Time Windows

正如命名那样，Time Windows 根据时间来聚合流数据。例如：一分钟的 tumbling time window 收集一分钟的元素，并在一分钟过后对窗口中的所有元素应用于一个函数。

在 Flink 中定义 tumbling time windows(翻滚时间窗口) 和 sliding time windows(滑动时间窗口) 非常简单：

tumbling time windows(翻滚时间窗口)

输入一个时间参数

```
data.keyBy(1)
    .timeWindow(Time.minutes(1)) //tumbling time window 每分钟统计一次数量和
    .sum(1);
```

sliding time windows(滑动时间窗口)

输入两个时间参数

```
data.keyBy(1)
    .timeWindow(Time.minutes(1), Time.seconds(30)) //sliding time window 每隔 30s 统计过去一分钟的数量和
    .sum(1);
```

有一点我们还没有讨论，即“收集一分钟的元素”的确切含义，它可以归结为一个问题，“流处理器如何解释时间?”

Apache Flink 具有三个不同的时间概念，即 processing time, event time 和 ingestion time。

这里可以参考我下一篇文章：

《从0到1学习Flink》—— 介绍Flink中的Event Time、Processing Time和Ingestion Time

Count Windows

Apache Flink 还提供计数窗口功能。如果计数窗口设置的为 100 ，那么将会在窗口中收集 100 个事件，并在添加第 100 个元素时计算窗口的值。

```
/**
 * Windows this {@code KeyedStream} into tumbling count windows.
 *
 * @param size The size of the windows in number of elements.
 */
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size) {
    return window(GlobalWindows.create()).trigger(PurgingTrigger.of(CountTrigger.of(size)));
}

/**
 * Windows this {@code KeyedStream} into sliding count windows.
 *
 * @param size The size of the windows in number of elements.
 * @param slide The slide interval in number of elements.
 */
public WindowedStream<T, KEY, GlobalWindow> countWindow(long size, long slide) {
    return window(GlobalWindows.create())
            .evictor(CountEvictor.of(size))
            .trigger(CountTrigger.of(slide));
}
```

在 Flink 的 DataStream API 中，tumbling count window 和 sliding count window 的定义如下:

tumbling count window

输入一个时间参数

```
data.keyBy(1)
    .countWindow(100) //统计每 100 个元素的数量之和
    .sum(1);
```

sliding count window

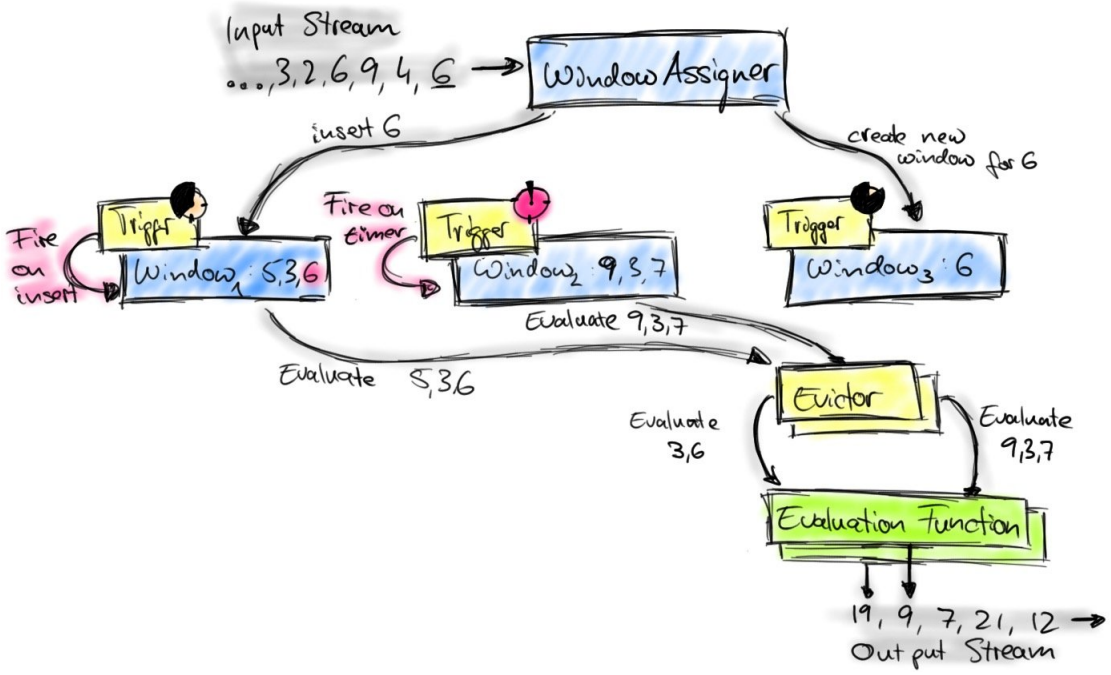
输入两个时间参数

```
data.keyBy(1)
    .countWindow(100, 10) //每 10 个元素统计过去 100 个元素的数量之和
    .sum(1);
```

解剖 Flink 的窗口机制

Flink 的内置 time window 和 count window 已经覆盖了大多数应用场景，但是有时候也需要定制窗口逻辑，此时 Flink 的内置的 window 无法解决这些问题。为了还支持自定义 window 实现不同的逻辑，DataStream API 为其窗口机制提供了接口。

下图描述了 Flink 的窗口机制，并介绍了所涉及的组件：



到达窗口操作符的元素被传递给 WindowAssigner。WindowAssigner 将元素分配给一个或多个窗口，可能会创建新的窗口。 窗口本身只是元素列表的标识符，它可能提供一些可选的元信息，例如 TimeWindow 中的开始和结束时间。注意，元素可以被添加到多个窗口，这也意味着一个元素可以同时在多个窗口存在。

每个窗口都拥有一个 Trigger(触发器)，该 Trigger(触发器) 决定何时计算和清除窗口。当先前注册的计时器超时时，将为插入窗口的每个元素调用触发器。在每个事件上，触发器都可以决定触发(即、清除(删除窗口并丢弃其内容)，或者启动并清除窗口。一个窗口可以被求值多次，并且在被清除之前一直存在。注意，在清除窗口之前，窗口将一直消耗内存。

当 Trigger(触发器) 触发时，可以将窗口元素列表提供给可选的 Evictor，Evictor 可以遍历窗口元素列表，并可以决定从列表的开头删除首先进入窗口的一些元素。然后其余的元素被赋给一个计算函数，如果没有定义 Evictor，触发器直接将所有窗口元素交给计算函数。

计算函数接收 Evictor 过滤后的窗口元素，并计算窗口的一个或多个元素的结果。DataStream API 接受不同类型的计算函数，包括预定义的聚合函数，如 sum（），min（），max（），以及 ReduceFunction，FoldFunction 或 WindowFunction。

这些是构成 Flink 窗口机制的组件。 接下来我们逐步演示如何使用 DataStream API 实现自定义窗口逻辑。 我们从 DataStream [IN] 类型的流开始，并使用 key 选择器函数对其分组，该函数将 key 相同类型的数据分组在一块。

```
SingleOutputStreamOperator<xxx> data = env.addSource(...);
data.keyBy()
```

如何自定义 Window?

1、Window Assigner

负责将元素分配到不同的 window。

Window API 提供了自定义的 WindowAssigner 接口，我们可以实现 WindowAssigner 的

```
public abstract Collection<W> assignWindows(T element, long
timestamp)
```

方法。同时，对于基于 Count 的 window 而言，默认采用了 GlobalWindow 的 window assigner，例如：

```
keyBy.window(GlobalWindows.create())
```

2、Trigger

Trigger 即触发器，定义何时或什么情况下移除 window

我们可以指定触发器来覆盖 WindowAssigner 提供的默认触发器。 请注意，指定的触发器不会添加其他触发条件，但会替换当前触发器。

3、Evictor（可选）

驱逐者，即保留上一 window 留下的某些元素

4、通过 apply WindowFunction 来返回 DataStream 类型数据。

利用 Flink 的内部窗口机制和 DataStream API 可以实现自定义的窗口逻辑，例如 session window。

结论

对于现代流处理器来说，支持连续数据流上的各种类型的窗口是必不可少的。Apache Flink 是一个具有强大功能集的流处理器，包括一个非常灵活的机制，可以在连续数据流上构建窗口。 Flink 为常见场景提供内置的窗口运算符，以及允许用户自定义窗口逻辑。

参考

1、https://flink.apache.org/news/2015/12/04/Introducing-windows.html

2、https://blog.csdn.net/lmalds/article/details/51604501

关注我

转载请务必注明原创地址为：http://www.54tianzhisheng.cn/2018/12/08/Flink-Stream-Windows/

另外我自己整理了些 Flink 的学习资料，目前已经全部放到微信公众号了。你可以加我的微信：zhisheng_tian，然后回复关键字：Flink 即可无条件获取到。

Github 代码仓库

https://github.com/zhisheng17/flink-learning/

以后这个项目的所有代码都将放在这个仓库里，包含了自己学习 flink 的一些 demo 和博客

相关文章

1、《从0到1学习Flink》—— Apache Flink 介绍

2、《从0到1学习Flink》—— Mac 上搭建 Flink 1.6.0 环境并构建运行简单程序入门

3、《从0到1学习Flink》—— Flink 配置文件详解

4、《从0到1学习Flink》—— Data Source 介绍

5、《从0到1学习Flink》—— 如何自定义 Data Source ？

6、《从0到1学习Flink》—— Data Sink 介绍

7、《从0到1学习Flink》—— 如何自定义 Data Sink？

8、《从0到1学习Flink》—— Flink Data transformation(转换)

9、《从0到1学习Flink》—— 介绍Flink中的Stream Windows

10、《从0到1学习Flink》—— Flink 中的几种 Time 详解

11、《从0到1学习Flink》—— Flink 写入数据到 ElasticSearch