Apache Flink Eventtime源码阅读

向怡帆 2019K8009918008

Apache Flink是在当前这个大数据时代，对于流的处理相当优异的引擎，在这里希望能通过自己的学习有所收获的同时，也能为 读者提供一些启发。

一、Apache Flink简介

1.1 什么是Apache Flink

Flink是一款分布式的计算引擎，它可以用来做批处理，即处理静态的数据集、历史的数据集**;**也可以用来做流处理，即实时地处理一些实时数据流，实时地产生数据的结果**;**也可以用来做一些基于事件的应用， 比如说滴滴通过**Flink CEP**实现实时监测用户及司机的行为流来判断用户或司机的行为是否正当。

1.2 Apache Flink历史

Flink起源于一个叫做**Stratosphere**的研究项目，目标是建立下一代大数据分析引擎，其在**2014**年**4**月**16** 日成为**Apache**的孵化项目，从**Stratosphere 0.6**开始，正式更名为**Flink**。**Flink 0.7**中介绍了最重要的特性：**Streaming API**。最初只支持**Java API**，后来增加了**Scala API**。

1.3 Apache Flink机制

Flink有四大机制，奠定了它能如此流行并广受大众青睐的基础。分别是**Checkpoint**、**State**、**Time**、**Window**。本文主要涉及的部分有**Time**和**Window**这两部分。

Checkpoint是**flink**提供的快照机制。**Flink**基于**Chandy-Lamport**算法实现了一个分布式的一致性的快照，从而提供了一致性的语义。

提供了一致性的语义之后，**Flink**为了让用户在编程时能够更轻松、更容易地去管理状态，还提供了一套非常简单明了的**State API**，这就是**state**机制。

之后，**Flink**还实现了**Time**的机制（主要依赖于**watermark**），能够支持基于事件的时间的处理，或者说 基于系统时间的处理，能够容忍数据的延时、容忍数据的迟到、容忍乱序的数据。

由于流计算一般在对流数据进行操作之前都会先进行开窗，并在窗口上计算，**flink**提供了这些窗口功能供用户使用。

1.4 Apache Flink用途

在国内，包括携程、唯品会、饿了么、滴滴、头条等等都曾使用**Apache Flink**做大数据流处理。他们的应用场景包括实时的机器学习，实时的统计分析，实时的异常监测等等。这些实践案例的共同点就是都 用来做实时性的任务。其中阿里巴巴用**Flink**的一个派生**Blink**实时排名优化。

在国外，像亚马逊公司用它做云服务的流处理等。其中著名的公司如**uber**，**epay**等也用**flink**开源引擎来做自己的流数据处理

1.5 Apache Flink优势

- **Flink**不会抛出**OOM**（内存不足）异常减
- 少垃圾收集
- 非常有效的磁盘溢出和网络传输
- 无需运行时调整
- 性能更稳定可靠
- 内置的基于成本的优化器
- 自定义状态维护

二、Window机制

2.1 什么是window

> 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**。

——**Imalds**李麦迪

> 在流处理应用中，数据是连续不断的，因此我们不可能等到所有数据都到了才开始处理。当然我们 可以每来一个消息就处理一次，但是有时我们需要做一些聚合类的处理，例如：在过去的**1**分钟内 有多少用户点击了我们的网页。在这种情况下，我们必须定义一个窗口，用来收集最近一分钟内的 数据，并对这个窗口内的数据进行计算。

——伍 翀

实际上**window**的出现就是我们对于无限数据流中，**“**无限**”**这一难题提出的处理方式。我们没法等无限的数据停止再去做处理，我们必须做实时的处理，来了一些我们就处理一些。就如同工厂的流水线，有永 远流不完的产品等待着我们包装，我们就是来一个包一个，处理一个。对于这个小范围一定数据量的聚 合处理，我们需要一个概念，需要一个规则，比如这个范围有多小？按照什么单位划分数据量？该做什 么处理？等等问题。于是窗口的概念应运而生，我们定义出这个对象**window**，我们给它规定一些**feature**（范围，数据量，划分，处理等），以便我们更精确，更有效的处理数据。

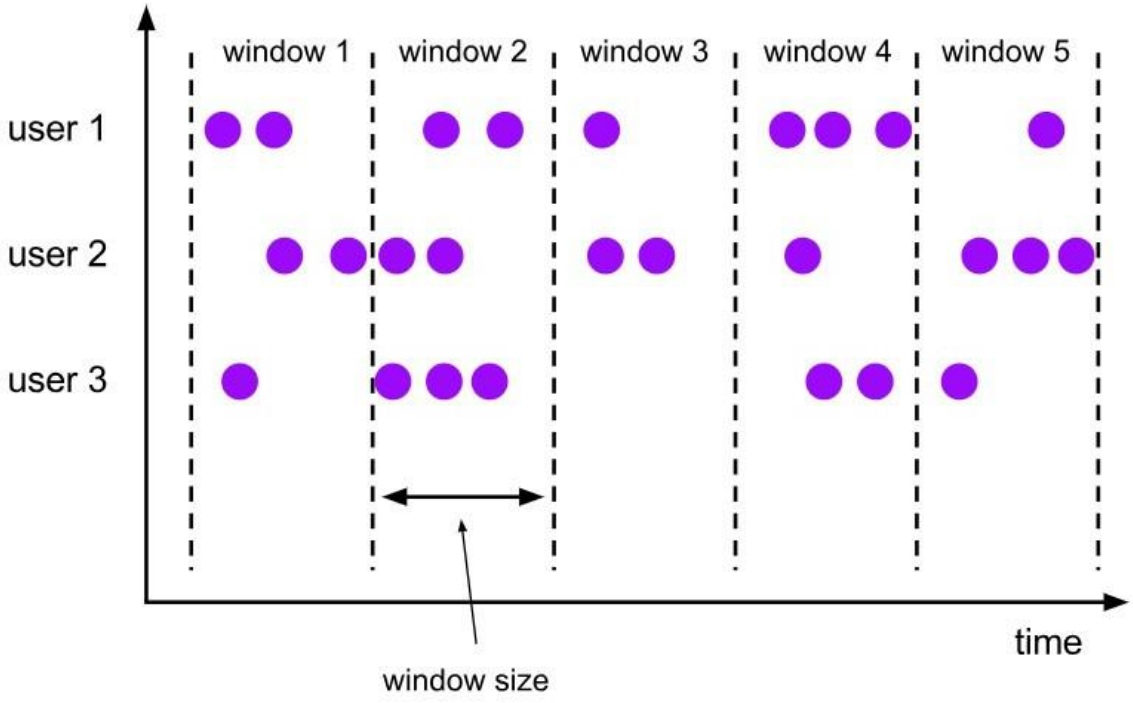
2.2 不同种类的window

窗口可以有多种驱动方式，实际上就是我们我们前文所提的划分规则。首先最自然地想法，按照时间切 片，那么这就是时间驱动的**Time Window**（比如每**10**秒钟）。当然也可以是数据驱动，按照数据量划分，这就是**Count Window**（比如每**100**个元素）。同时除了按照驱动类型分类窗口，还可以有按照其他性质分类的窗口：翻滚窗口（**Tumbling Window**，无重叠），滚动窗口（**Sliding Window**，有重叠）， 和会话窗口（**Session Window**，活动间隙）以及全局窗口。下面我们会分别依次介绍这些窗口。

2.2.1 Time Window

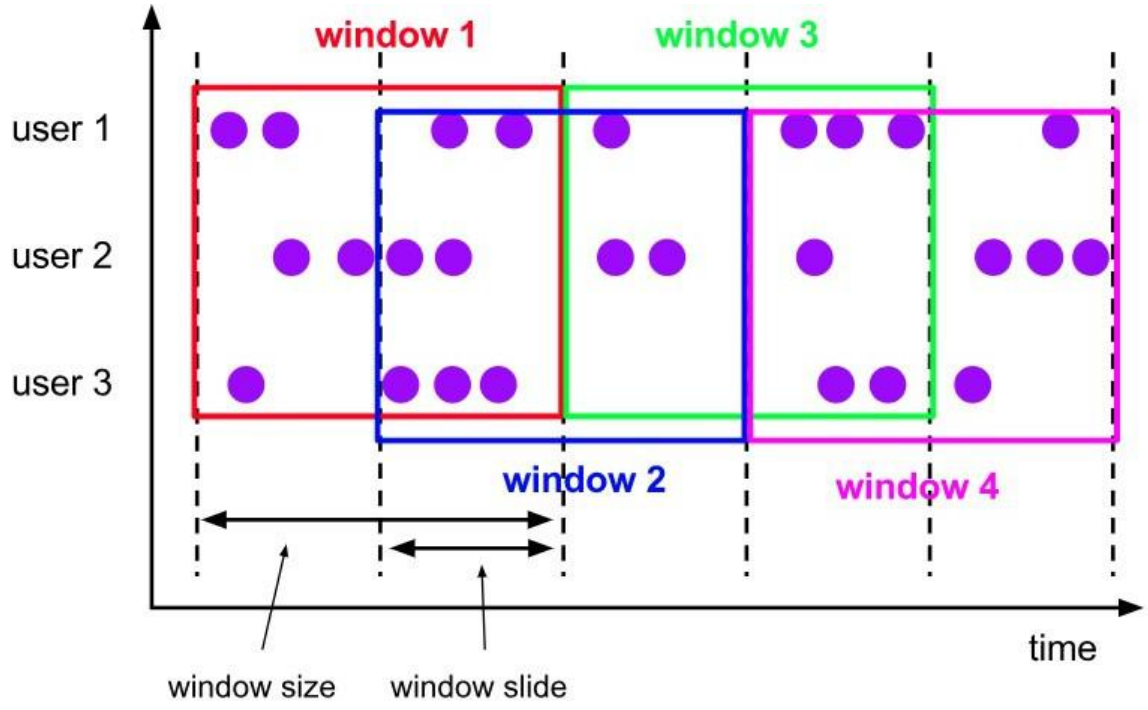
Time Window 是根据时间对数据流进行分组的。其实这里就涉及到了我们分析源码的主要部分时间的概念。在**flink**中时间有三种概念：**event time**（事件时间：事件发生时的时间），**ingestion time**（摄取时间：事件进入流处理系统的时间），**processing time**（处理时间：消息被计算处理的时间）。后文在分析**Eventtime**的时候会详细去探讨。当然**Flink** 中窗口机制和时间类型是完全解耦的，也就是说当需要改变时间类型时不需要更改窗口逻辑相关的代码。

2.2.1.1 Tumbling Time Window



由上图可以看出，固定**window size**以后，不重叠的切分数据流即是**Tumbling Time Window**。例如， 我们需要统计每一分钟中用户购买的商品的总数，需要将用户的行为事件按每一分钟进行切分，这种切 分被成为翻滚时间窗口（**Tumbling Time Window**）。

2.2.1.2 Sliding Time Window

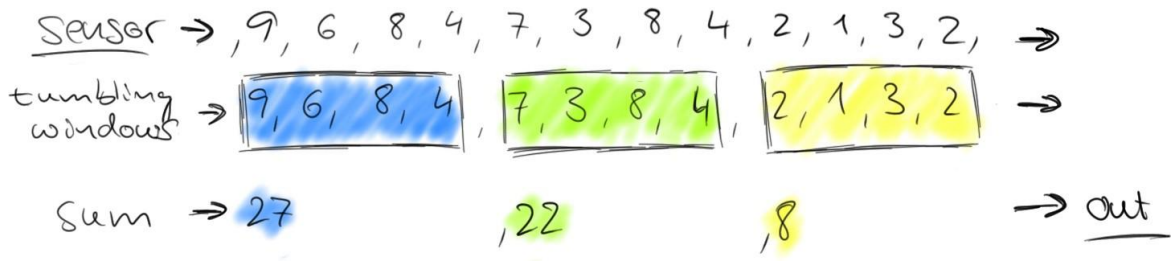


由上图可以看出，固定**window size**以后，可重叠的的切分数据流即是**Sliding Time Window**。比如，我们可以每**30**秒计算一次最近一分钟用户购买的商品总数。

2.2.2 Count Window

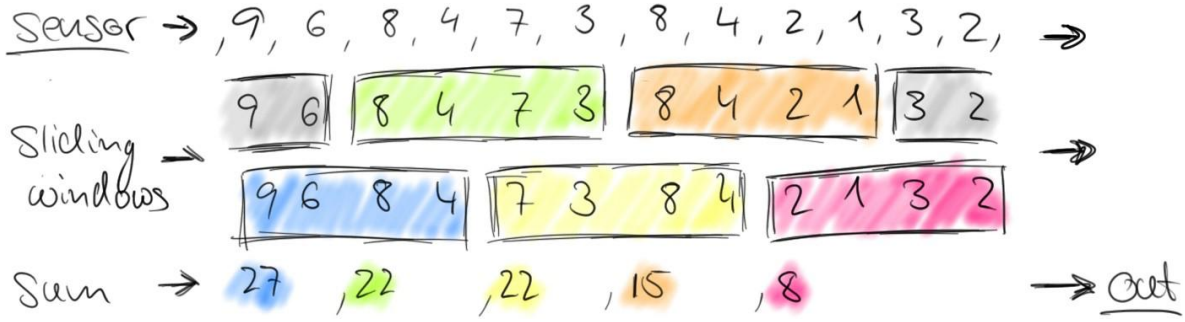
Count Window 是根据元素个数对数据流进行分组的。

2.2.2.1 Tumbling Count Window



上图所示即为固定对四个数据进入后进行加和处理，由于对于数据的切分是不重叠的，所以是**Tumbling Count Window**。

2.2.2.2 Sliding Count Window



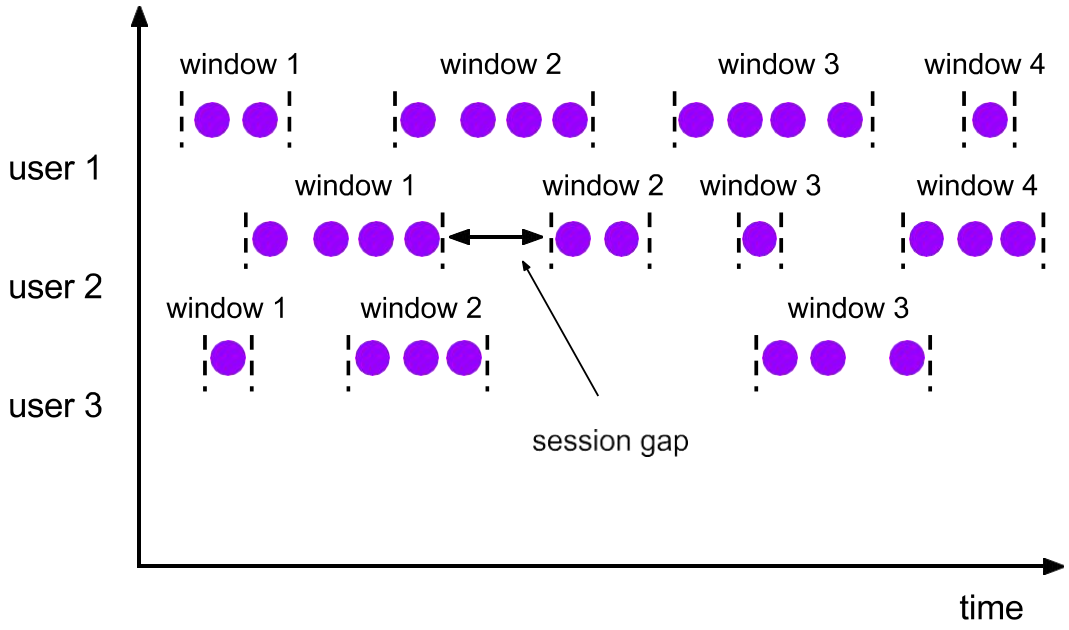
上图所示即为每过两个数据，对前四个数据进行加和处理，由于对于数据的切分是有重叠的，所以是**Sliding Count Window**。

2.2.3 Session Window

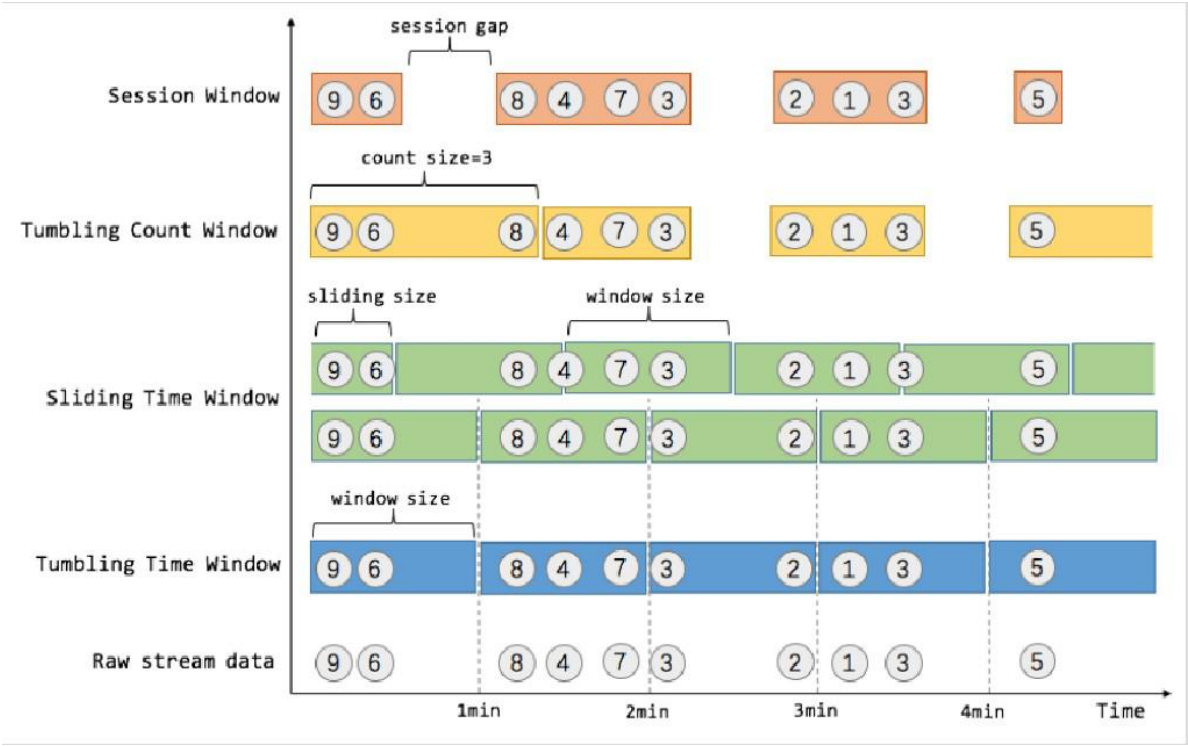
我们可以想到在现实生活中我们其实可以想到，假如我们要统计一个客户购买量，实际上他不是一直在 购物的，他分为活跃期和不活跃期，在活跃期大量购物，之后便处于长时间的不活跃期，所以如果我们 还是按照常规的窗口划分，那么将浪费相当多资源，自此，**Session Window**应运而生。

> **The** *session windows* **assigner groups elements by sessions of activity. Session windows do not overlap and do not have a fixed start and end time, in contrast to** *tumbling windows* **and** *sliding windows***. Instead a session window closes when it does not receive elements for a certain period of time,** *i.e.*, **when a gap of inactivity occurred. A session window assigner can be configured with either a static** *session gap* **or with a** *session gap extractor* **function which defines how long the period of inactivity is. When this period expires, the current session closes and subsequent elements are assigned to a new session window.**

根据**apache flink**文档中的描述，我们可以规定在一段时间内未接收到元素。**Session Window**将会关闭，之后的元素将会分配给新的**Session Window**，如下图所示。

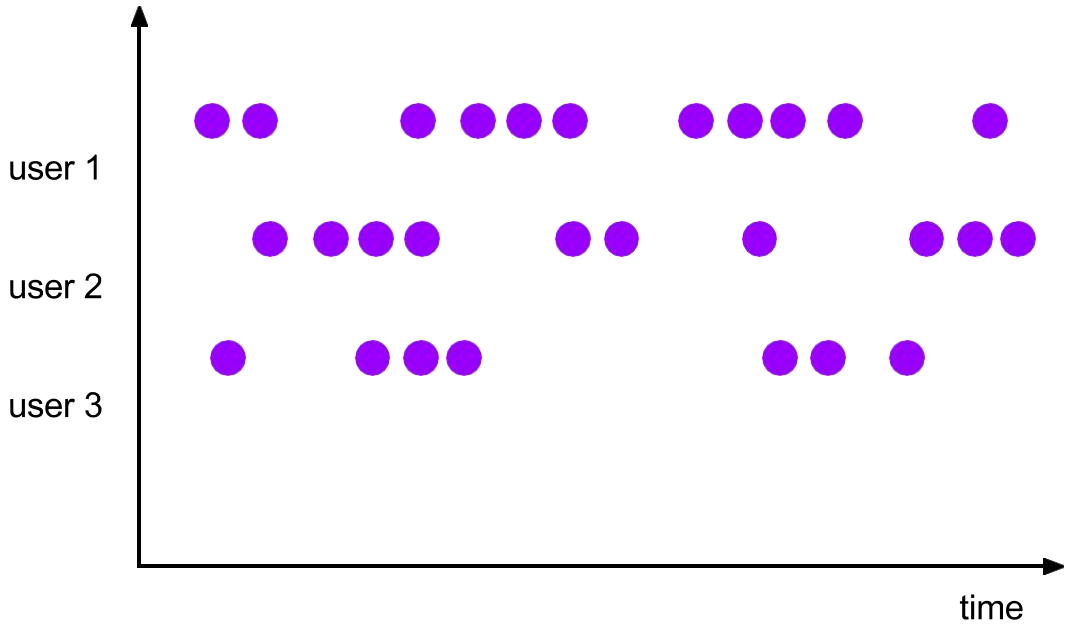


下面我们给出一张图集成的大部分**window**的特性，可以从该图中看出不同**window**的区别。

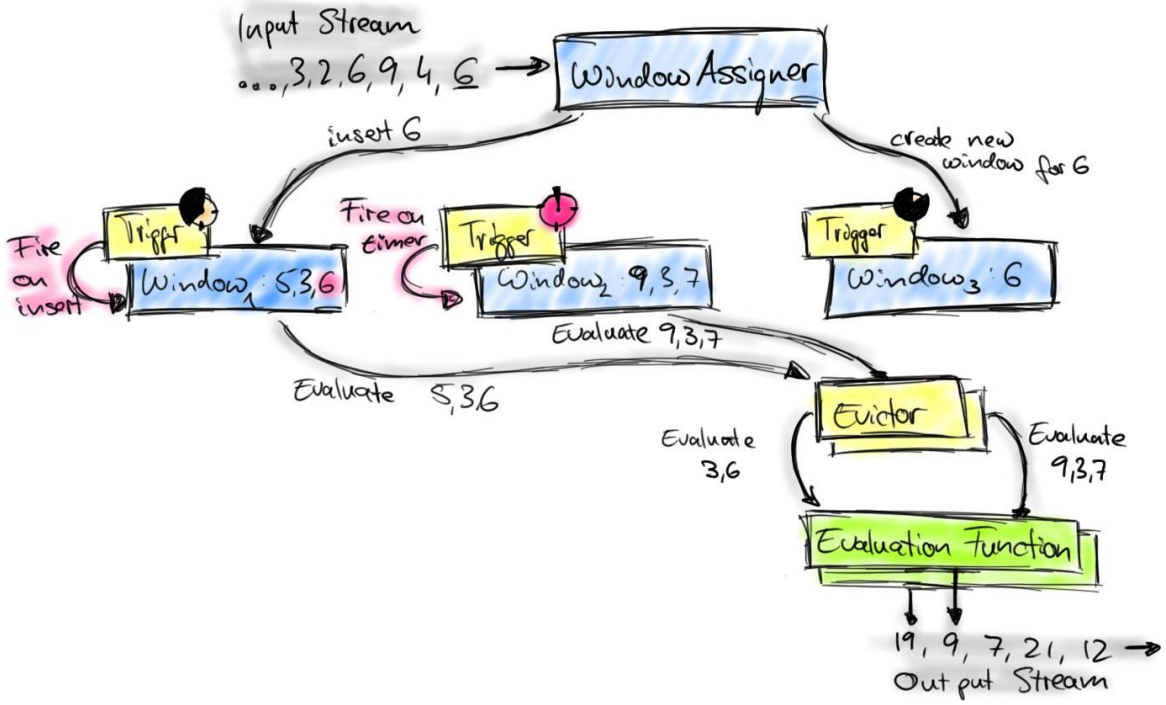


2.2.4 Global Window

顾名思义，全局窗口，所有在窗口中的元素共享一个**key**。这类窗口只有在特定情况下才有意义。



2.3 window工作原理



上图对于**window**的工作机制流程进行了很好地诠释。

到达窗口运算符的元素将传递给`WindowAssigner`。`WindowAssigner`将元素分配给一个或多个窗口，可能会创建新窗口。`Window`本身仅仅是元素的列表的标识符，并且可以提供一些可选的元信息，如在`TimeWindow`的情况下的开始时间和结束时间。请注意，可以将元素添加到多个窗口，这也意味着可以同时存在多个窗口。

每个窗口都拥有一个`Trigger`，它决定何时评估或清除该窗口。对于每个插入到窗口中的元素以及先前注册的计时器超时时，将触发该触发器。对于每个事件，触发器可以决定触发（即评估），清除（删除窗口并丢弃其内容），或者触发然后清除窗口。刚刚触发的触发器将评估窗口并保持其原样，即所有元素保留在窗口中，并在下次触发时再次评估。一个窗口可以被评估多次，并且一直存在直到清除为止。 请注意，在清除窗口之前，窗口会消耗内存。

触发触发器时，可以将窗口元素列表提供给可选的`Evictor`。`Evictor`可以遍历列表，并决定从列表的开头删除一些元素，即删除一些首先进入窗口的元素。其余元素提供给评估功能。如果未定义退出者，则触发器将所有窗口元素直接移交给评估函数。

评估函数接收一个窗口的元素（可能由**Evictor**过滤），并为该窗口计算一个或多个结果元素。数据流**API**接受不同类型的评价函数，包括预定义的聚合功能，例如`sum()`，`min()`，`max()`，以及一个`ReduceFunction`，`FoldFunction`或`WindowFunction`。`WindowFunction`是最通用的评估函数，它接收窗口对象（即，窗口的元数据），窗口元素列表以及窗口键（如果是键控窗口）作为参数。