32-BeamWindow：打通流处理的任督二脉

你好，我是蔡元楠。

今天我要与你分享的主题是“Beam Window：打通流处理的任督二脉”。

在上一讲中，我们一起用Beam编写了第一个完整的WordCount项目，我们所用的例子是统计莎士比亚的文集中最常使用到的一些单词。

这里我们所用到的“莎士比亚文集”这种类型的数据集是一个静态的数据集。也就是说，我们在生成输入数据集的时候，就已经知道了这个数据集是完整的，并不需要再等待新的数据进来。

根据前面的内容，我们可以把这种数据集归类为有界数据集（Bounded Dataset）。这里我们的数据流水线就是一个批处理的数据流水线。

这个时候你可能会有一个疑问，如果我们想要统计的内容是一个正在连载的小说，我们在编写数据流水线的时候，这个小说还并没有完结，也就是说，未来还会不断有新的内容作为输入数据流入我们的数据流水线，那我们需要怎么做呢？

这个时候我们就需要用到窗口（Window）这个概念了。

窗口

在Beam的世界中，窗口这个概念将PCollection里的每个元素根据时间戳（Timestamp）划分成为了不同的有限数据集合。

当我们要将一些聚合操作（Aggregation）应用在PCollection上面的时候，或者我们想要将不同的PCollections连接（Join）在一起的时候，其实Beam是将这些操作应用在了这些被窗口划分好的不同数据集合上的。

无论是有界数据集还是无界数据集，Beam都会一视同仁，按照上面所说的规则进行处理。

你可能又会有一个疑问，我们在上一讲的例子当中，根本就没有提到过窗口这个概念，但是我刚刚又说，Beam同样会将有界数据集根据窗口划分成不同的有限数据集合来处理，那这些窗口、PCollection中每个元素的时间戳又是从哪里来的呢？

其实，我们在用I/O连接器读取有界数据集的过程中，Read Transform会默认为每个元素分配一个相同的时间戳。在一般情况下，这个时间戳就是你运行数据流水线的时间，也就是处理时间（Processing Time）。而Beam也会为这个数据流水线默认地分配一个全局窗口（Global Window），你可以把它理解为是一个从无限小到无限大的时间窗口。

如果你想要显式地将一个全局窗口赋予一个有界数据集的话，可以使用如下代码来完成：

Java

```
PCollection<String> input = p.apply(TextIO.read().from(filepath));
```

```
PCollection<String> batchInputs = input.apply(Window.<String>into(new GlobalWindows()));
```

需要注意的是，你在处理有界数据集的时候，可以不用显式地将一个窗口分配给一个PCollection数据集。但是，在处理无边界数据集的时候，你必须要显式地分配一个窗口给这个无边界数据集。而这个窗口不可以是前面提到的全局窗口，否则在运行数据流水线的时候会直接抛出异常错误。

在了解过窗口的基本概念之后，接下来我来给你讲讲在Beam中存在的不同窗口类型。

固定窗口（Fixed Window）

固定窗口在有的数据处理框架中又被称为滚动窗口（Tumbling Window）。固定窗口通常是由一个静态的窗口大小定义而来的。

例如，要是我们定义一个每小时的窗口，那这个窗口大小就是固定的一个小时，如果我们按照2019年7月8号的0时作为时间的起始点，那么这个固定窗口就可以分为类似下面这样的形式：

```
[July 8, 2019 0:00:00 AM, July 8, 2019 1:00:00 AM),
[July 8, 2019 1:00:00 AM, July 8, 2019 2:00:00 AM)
[July 8, 2019 2:00:00 AM, July 8, 2019 3:00:00 AM)
……
```

而一个PCollection中的所有元素，就会根据它们各自自身的时间戳被分配给相应的固定窗口中。

这里你需要注意一点，因为固定窗口本质上并不可能会重叠在一起，如果我们定义的窗口是固定窗口的话，PCollection中的每一个元素只会落入一个，且是唯一一个窗口中。

在Beam中，如果要定义一个上述所说的按照每小时分割的窗口，我们可以使用一个Window Transform来完成，如下所示：

Java

```
PCollection<String> input = p.apply(KafkaIO.<Long, String>read()).apply(Values.<String>create());

PCollection<String> fixedWindowedInputs = input.apply(Window.<String>into(FixedWindows.of(Duration.standard
```

滑动窗口（Sliding Window）

滑动窗口通常是由一个静态的窗口大小和一个滑动周期（Sliding Period）定义而来的。

例如，我们可以定义一个窗口大小为一个小时，滑动周期为30分钟的一个滑动窗口。我们还是以2019年7月8号的0时作为时间的起始点，那这个滑动窗口可以分为下面这样的形式：

```
[July 8, 2019 0:00:00 AM, July 8, 2019 1:00:00 AM)
[July 8, 2019 0:30:00 AM, July 8, 2019 1:30:00 AM)
[July 8, 2019 1:00:00 AM, July 8, 2019 1:30:00 AM)
[July 8, 2019 1:30:00 AM, July 8, 2019 2:00:00 AM)
……
```

因为Beam对于滑动周期的大小并没有做任何限制，所以你可以看到，滑动窗口和固定窗口不同的是，当滑动周期小于窗口大小的时候，滑动窗口会有部分重叠。也就是说，在一个PCollection里面，同一个元素是可以被分配到不同的滑动窗口中的。

可能你也会发现到，当滑动窗口的窗口大小和滑动周期一样的时候，这个滑动窗口的性质其实就和固定窗口一样了。

在Beam中，如果要定义一个上述所说，窗口大小为一个小时而滑动周期为30分钟的一个滑动窗口，我们同样可以使用一个Window Transform来完成，如下所示：

Java

```
PCollection<String> input = p.apply(KafkaIO.<Long, String>read()).apply(Values.<String>create());

PCollection<String> slidingWindowedInputs = input.apply(Window.<String>into(SlidingWindows.of(Duration.stan
```

会话窗口（Sessions Window）

会话窗口和上面所说的两个窗口有点不一样，它并没有一个固定的窗口长度。

会话窗口主要是用于记录持续了一段时间的活动数据集。在一个会话窗口中的数据集，如果将它里面所有的元素按照时间戳来排序的话，那么任意相邻的两个元素它们的时间戳相差不会超过一个定义好的静态间隔时间段（Gap Duration）。

怎么理解这个定义呢？我想用一个例子来解释会比较清晰。

假设，我们现在正在一个视频流的平台中处理无界数据集，我们想要分析在这个平台中的一些用户行为习惯。

为了方便说明，我们想要分析的问题非常简单，就是一个用户在线看视频一般会在每天的什么时候开始看多长时间的视频。同时，我们假设只会有一个用户的数据流入我们的输入数据集中，这个数据会带有用户对视频平台发送视频流请求的时间戳。

我们希望定义一个会话窗口来处理这些数据，而这个会话窗口的间隔时间段为5分钟。

所有的数据假设都是发生在2019年7月8号中的，流入的数据集如下：

```
(key1, value1, [7:44:00 AM, 7:44:00 AM))
(key1, value2, [7:45:00 AM, 7:45:00 AM))
(key1, value3, [7:49:00 AM, 7:49:00 AM))
(key1, value4, [8:01:00 AM, 8:01:00 AM))
(key1, value5, [8:02:00 AM, 8:02:00 AM))
```

那么，这5个数据会形成两个会话窗口，分别是：

```
(key1, [(value1, [7:44:00 AM, 7:44:00 AM)), (value2, [7:45:00 AM, 7:45:00 AM)), (value3, [7:49:00 AM, 7:49:0
```

```
(key1, [(value4, [8:01:00 AM, 8:01:00 AM)), (value5, [8:02:00 AM, 8:02:00 AM))])
```

你可以看到，在第一个会话窗口中，数据的时间戳分别是7:44:00AM，7:45:00AM和7:49:00AM，这个窗口的总长度有5分钟。任意相邻的两个元素之间的时间间隔不会超过我们之前定义好的5分钟。

而在第二个会话窗口中，数据的时间戳分别是8:01:00AM和8:02:00AM，这个窗口的总长度有1分钟，同样任意相邻的两个元素之间的时间间隔不会超过5分钟。每一个会话窗口都记录了一个用户的在线活跃点和时长。

在Beam中，如果要定义一个上述所说会话窗口的话，你可以使用以下代码来完成：

Java

```
PCollection<String> input = p.apply(KafkaIO.<Long, String>read()).apply(Values.<String>create());

PCollection<String> sessionWindowedInputs = input.apply(Window.<String>into(Sessions.withGapDuration(Durati
```

小结

今天我们一起学习了在处理无界数据集的时候，我们需要显示定义到的一个概念——窗口。

窗口的定义是后面我们编写流处理数据流水线的一个基石。而窗口这个概念其实就是用来回答我们在第23讲中“WWWH”问题里“Where in event time they are being computed”这个问题的。除去全局窗口，Beam里面总共让我们定义三种不同的窗口类型，分别是固定窗口，滑动窗口和会话窗口。

思考题

在今天介绍的三种时间窗口类型中，你觉得这三种时间窗口分别适合使用在什么样的应用场景中呢？

欢迎你把答案写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

精选留言：

- cricket1981 2019-07-08 09:04:30
  beam支持动态session gap定义吗？全局窗口的作用和使用场景是什么？beam支持自定义窗口吗？