

17 | Structured Streaming: 如何用DataFrame API进行实时数据分析?

2019-05-27 蔡元楠

大规模数据处理实战

[进入课程 >](#)



讲述：巴莫

时长 11:13 大小 10.27M



你好，我是蔡元楠。

上一讲中，我们介绍了 Spark 中的流处理库 Spark Streaming。它将无边界的流数据抽象成 DStream，按特定的时间间隔，把数据流分割成一个个 RDD 进行批处理。所以，DStream API 与 RDD API 高度相似，也拥有 RDD 的各种性质。

在第 15 讲中，我们比较过 RDD 和 DataSet/DataFrame。你还记得 DataSet/DataFrame 的优点吗？你有没有想过，既然已经有了 RDD API，我们为什么还要引入 DataSet/DataFrame 呢？

让我们来回顾一下 DataSet/DataFrame 的优点（为了方便描述，下文中我们统一用 DataFrame 来代指 DataSet 和 DataFrame）：

DataFrame 是**高级 API**，提供类似于**SQL**的 query 接口，方便熟悉关系型数据库的开发人员使用；

Spark SQL 执行引擎会自动优化 DataFrame 程序，而用 RDD API 开发的程序本质上需要工程师自己构造 RDD 的 DAG 执行图，所以依赖于工程师自己去优化。

那么我们自然会想到，如果可以拥有一个基于 DataFrame API 的流处理模块，作为工程师的我们就不要去用相对 low level 的 DStream API 去处理无边界数据，这样会大大提升我们的开发效率。

基于这个思想，2016 年，Spark 在其 2.0 版本中推出了结构化流数据处理的模块 Structured Streaming。

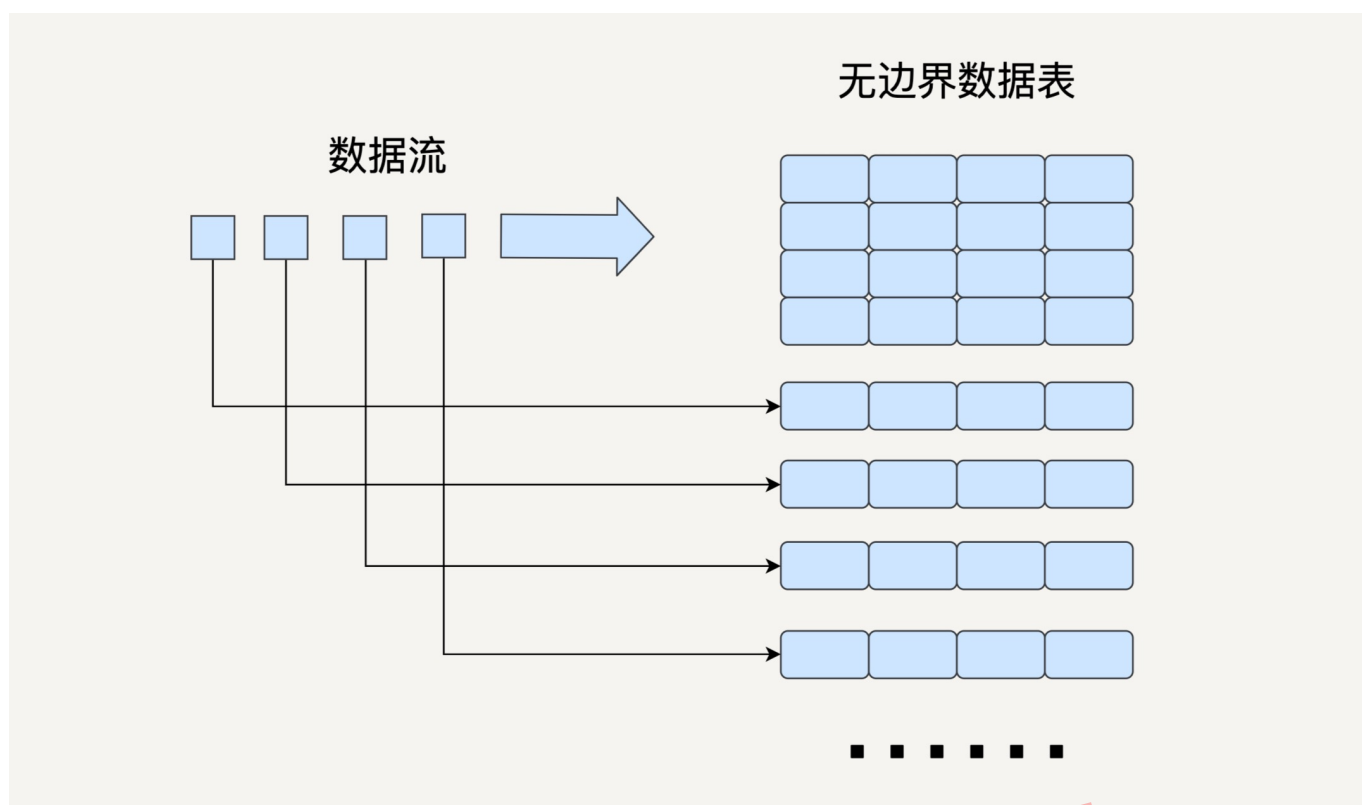
Structured Streaming 是基于 Spark SQL 引擎实现的，依靠 Structured Streaming，在开发者眼里，流数据和静态数据没有区别。我们完全可以像批处理静态数据那样去处理流数据。随着流数据的持续输入，Spark SQL 引擎会帮助我们持续地处理新数据，并且更新计算结果。

今天，就让我们一起来一起学习 Structured Streaming 的原理以及应用。

Structured Streaming 模型

流数据处理最基本的问题就是如何对不断更新的无边界数据建模。

之前讲的 Spark Streaming 就是把流数据按一定的时间间隔分割成许多个小的数据块进行批处理。在 Structured Streaming 的模型中，我们要把数据看成一个无边界的关系型的数据表。每一个数据都是表中的一行，不断会有新的数据行被添加到表里来。我们可以对这个表做任何类似批处理的查询，Spark 会帮我们不断对新加入的数据进行处理，并更新计算结果。



与 Spark Streaming 类似，Structured Streaming 也是将输入的数据流按照时间间隔（以一秒为例）划分成数据段。每一秒都会把新输入的数据添加到表中，Spark 也会每秒更新输出结果。输出结果也是表的形式，输出表可以写入硬盘或者 HDFS。

这里我要介绍一下 Structured Streaming 的三种输出模式。

1. 完全模式 (Complete Mode)：整个更新过的输出表都被写入外部存储；
2. 附加模式 (Append Mode)：上一次触发之后新增加的行才会被写入外部存储。如果老数据有改动则不适合这个模式；
3. 更新模式 (Update Mode)：上一次触发之后被更新的行才会被写入外部存储。

需要注意的是，Structured Streaming 并不会完全存储输入数据。每个时间间隔它都会读取最新的输入，进行处理，更新输出表，然后把这次的输入删除。Structured Streaming 只会存储更新输出表所需要的信息。

Structured Streaming 的模型在根据事件时间 (Event Time) 处理数据时十分方便。

我们在第六讲中曾经讲过事件时间和处理时间 (Processing Time) 的区别。这里我再简单说一下。事件时间指的是事件发生的时间，是数据本身的属性；而处理时间指的是 Spark 接收到数据的时间。

很多情况下，我们需要基于事件时间来处理数据。比如说，统计每个小时接到的订单数量，一个订单很有可能在 12:59 被创建，但是到了 13:01 才被处理。


在 Structured Streaming 的模型中，由于每个数据都是输入数据表中的一行，那么事件时间就是行中的一列。依靠 DataFrame API 提供的类似于 SQL 的接口，我们可以很方便地执行基于时间窗口的查询。

Streaming DataFrame API

在 Structured Streaming 发布以后，DataFrame 既可以代表静态的有边界数据，也可以代表无边界数据。之前对静态 DataFrame 的各种操作同样也适用于流式 DataFrame。接下来，让我们看几个例子。

创建 DataFrame

SparkSession.readStream() 返回的 DataStreamReader 可以用于创建流 DataFrame。它支持多种类型的数据流作为输入，比如文件、Kafka、socket 等。

 复制代码

```
1 socketDataFrame = spark
2   .readStream
3   .format("socket")
4   .option("host", "localhost")
5   .option("port", 9999)
6   .load()
```

上边的代码例子创建了一个 DataFrame，用来监听来自 localhost:9999 的数据流。

基本的查询操作

流 DataFrame 同静态 DataFrame 一样，不仅支持类似 SQL 的查询操作（如 select 和 where 等），还支持 RDD 的转换操作（如 map 和 filter）。让我们一起来看下面的例子。

假设我们已经有一个 DataFrame 代表一个学生的数据流，即每个数据是一个学生，每个学生有名字（name）、年龄（age）、身高（height）和年级（grade）四个属性，我们可以用 DataFrame API 去做类似于 SQL 的 Query。

[复制代码](#)

```
1 df = ... // 这个 DataFrame 代表学校学生的数据流, schema 是{name: string, age: number, height
2 df.select("name").where("age > 10") // 返回年龄大于 10 岁的学生名字列表
3 df.groupBy("grade").count() // 返回每个年级学生的人数
4 df.sort_values(['age'], ascending=False).head(100) // 返回 100 个年龄最大的学生
```

在这个例子中, 通过第二行我们可以得到所有年龄在 10 岁以上的学生名字, 第三行可以得到每个年级学生的人数, 第四行得到 100 个年龄最大的学生信息。此外, DataFrame 还支持很多基本的查询操作, 在此不做赘述。

我们还可以通过 `isStreaming` 函数来判断一个 DataFrame 是否代表流数据。

[复制代码](#)

```
1 df.isStreaming()
```

基于事件时间的时间窗口操作

在学习 Spark Streaming 的时间窗口操作时, 我们举过一个例子, 是每隔 10 秒钟输出过去 60 秒的前十热点词。这个例子是基于处理时间而非事件时间的。

现在让我们设想一下, 如果数据流中的每个词语都有一个时间戳代表词语产生的时间, 那么要怎样实现, 每隔 10 秒钟输出过去 60 秒内产生的前十热点词呢? 你可以看看下边的代码。

[复制代码](#)

```
1 words = ... # 这个 DataFrame 代表词语的数据流, schema 是 { timestamp: Timestamp, word: St
2
3
4 windowedCounts = words.groupBy(
5     window(words.timestamp, "1 minute", "10 seconds"),
6     words.word
7 ).count()
8 .sort(desc("count"))
9 .limit(10)
```


基于词语的生成时间，我们创建了一个窗口长度为 1 分钟，滑动间隔为 10 秒的 window。然后，把输入的词语表根据 window 和词语本身聚合起来，并统计每个 window 内每个词语的数量。之后，再根据词语的数量进行排序，只返回前 10 的词语。


在 Structured Streaming 基于时间窗口的聚合操作中，groupBy 是非常常用的。

输出结果流

当经过各种 SQL 查询操作之后，我们创建好了代表最终结果的 DataFrame。下一步就是开始对输入数据流的处理，并且持续输出结果。

我们可以用 Dataset.writeStream() 返回的 DataStreamWriter 对象去输出结果。它支持多种写入位置，如硬盘文件、Kafka、console 和内存等。

```
1 query = wordCounts
2     .writeStream
3     .outputMode("complete")
4     .format("csv")
5     .option("path", "path/to/destination/dir")
6     .start()
7
8
9 query.awaitTermination()
```

 复制代码

在上面这个代码例子中，我们选择了完全模式，把输出结果流写入了 CSV 文件。

Structured Streaming 与 Spark Streaming 对比

接下来，让我们对比一下 Structured Streaming 和上一讲学过的 Spark Streaming。看看同为流处理的组件的它们各有什么优缺点。

简易度和性能

Spark Streaming 提供的 DStream API 与 RDD API 很类似，相对比较低 level。

当我们编写 Spark Streaming 程序的时候，本质上就是要去构造 RDD 的 DAG 执行图，然后通过 Spark Engine 运行。这样开发者身上的担子就很重，很多时候要自己想办法去提

高程序的处理效率。这不是 Spark 作为一个数据处理框架想看到的。对于好的框架来说，开发者只需要专注在业务逻辑上，而不用操心别的配置、优化等繁杂事项。

Structured Streaming 提供的 DataFrame API 就是这么一个相对高 level 的 API，大部分开发者都很熟悉关系型数据库和 SQL。**这样的数据抽象可以让他们用一套统一的方案去处理批处理和流处理**，不用去关心具体的执行细节。

而且，DataFrame API 是在 Spark SQL 的引擎上执行的，Spark SQL 有非常多的优化功能，比如执行计划优化和内存管理等，所以 Structured Streaming 的应用程序性能很好。

实时性

在上一讲中我们了解到，Spark Streaming 是准实时的，它能做到的最小延迟在一秒左右。

虽然 Structured Streaming 用的也是类似的微批处理思想，每过一个时间间隔就去拿来最新的数据加入到输入数据表中并更新结果，但是相比起 Spark Streaming 来说，它更像是实时处理，能做到用更小的时间间隔，最小延迟在 100 毫秒左右。

而且在 Spark 2.3 版本中，Structured Streaming 引入了连续处理的模式，可以做到真正的毫秒级延迟，这无疑大大拓展了 Structured Streaming 的应用广度。不过现在连续处理模式还有很多限制，让我们期待它的未来吧。

对事件时间的支持

就像我们在前边讲过的，Structured Streaming 对基于事件时间的处理有很好的支持。

由于 Spark Streaming 是把数据按接收到的时间切分成一个个 RDD 来进行批处理，所以它很难基于数据本身的产生时间来进行处理。如果某个数据的处理时间和事件时间不一致的话，就容易出问题。比如，统计每秒的词语数量，有的数据先产生，但是在下一个时间间隔才被处理，这样几乎不可能输出正确的结果。

Structured Streaming 还有很多其他优点。比如，它有更好的容错性，保证了端到端 exactly once 的语义等等。所以综合来说，Structured Streaming 是比 Spark Streaming 更好的流处理工具。

思考题

在基于事件时间的窗口操作中，Structured Streaming 是怎样处理晚到达的数据，并且返回正确结果的呢？

比如，在每十分钟统计词频的例子中，一个词语在 1:09 被生成，在 1:11 被处理，程序在 1:10 和 1:20 都输出了对应的结果，在 1:20 输出时为什么可以把这个词语统计在内？这样的机制有没有限制？

欢迎你把自己的答案写在留言区，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。

 极客时间

大规模数据处理实战

Google 一线工程师的大数据架构实战经验

蔡元楠

Google Brain 资深工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 16 | Spark Streaming: Spark的实时流计算API

下一篇 18 | Word Count: 从零开始运行你的第一个Spark应用



Ming

2019-05-27

👍 3

我不确定有没有完全理解问题..

我想大概是因为，输出时间所对应的窗口可以故意设置的比输出时间稍微早一点，这样可以对数据延迟有一定的抗性。不然例子中的1:09分的数据就没机会被使用了。

...

展开 ▾



leben kri...

2019-05-27

👍 1

一般是处理滞后一定时间的数据，超过了这个时间范围，就会舍弃

展开 ▾



cricket198...

2019-05-27

👍 1

spark structure streaming有没有类似flink的sideOutput机制？支持超过watermark的事件被处理到



方伟

2019-05-27

👍 1

我知道在flink中可以通过watermark来处理这样的场景，在Structured Streaming中应该也是这样的方式来处理吧。



Rainbow

2019-05-27

👍 1

10分钟统计一次，按照处理时间分1:00-1:10，1:10-1:20；所以单词的处理时间位于第二个区间会被第二次统计到；如果按照事件时间，sql里time>1:00 and time<1:10就可以把单词归类到第一个区间，这么理解对吗，老师？

展开 ▾



张凯江

2019-05-29

👍

输出模式支持呀。

完全模式和更新模式哈。



青石

2019-05-29



watermark, $\text{process time} - \text{event time} > \text{watermark}$ 则直接丢失, $\text{process time} - \text{event time} < \text{watermark}$ 则接收数据处理, 更新结果表。

展开 ▾



锦

2019-05-29



我觉得可能是通过冗余计算上一个时间窗口中的数据来实现的。
局限性就是不支持迟到太久的数据



周凯

2019-05-29



程序在1:10处理的是1:09之前生成的数据, 往后推10分钟, 那1:20处理的是1:19之前生成的数据



胡鹏

2019-05-29



老师, 我最近遇到个问题还望帮忙提点一下:

1. 需求: 统计实时订单量(类似)
2. 通过maxwell读取binlog数据同步到kafka
3. spark-streaming处理kafka里面的数据
4. spark-sql定义不同的实时报表...

展开 ▾



RocWay

2019-05-28



我的理解是: 虽然设立了窗口, 但是有些事件可能由于网络或其他原因迟到了, 这些迟到的事件也要被计算在内。否则这段窗口内的数据计算就会“不准”。当然也不能无限允许迟到, 所以Spark也设立了watermark。如果窗口的结束时间减去watermark, 比某个事件的时间还“晚”, 那这个事件就不能算在这个窗口里。

展开 ▾



安

2019-05-27



因为1:20是处理时间，1:09是事件时间，1点20输出数据的事件时间还没到1点20。是这样吗？不太确定



Mr.

2019-05-27



不知道老师能否也讲解一些关于flink方面的东西呢？

展开 ▾