结构化流编程指南

[Overview](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#overview)

[Quick Example](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#quick-example)

[Programming Model](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#programming-model)

[Basic Concepts](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#basic-concepts)

[Handling Event-time and Late Data](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#handling-event-time-and-late-data)

[Fault Tolerance Semantics](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#fault-tolerance-semantics)

[API using Datasets and DataFrames](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#api-using-datasets-and-dataframes)

[Creating streaming DataFrames and streaming Datasets](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#creating-streaming-dataframes-and-streaming-datasets)

[Input Sources](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#input-sources)

[Schema inference and partition of streaming DataFrames/Datasets](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#schema-inference-and-partition-of-streaming-dataframesdatasets)

[Operations on streaming DataFrames/Datasets](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#operations-on-streaming-dataframesdatasets)

[Basic Operations - Selection, Projection, Aggregation](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#basic-operations---selection-projection-aggregation)

[Window Operations on Event Time](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time)

[Handling Late Data and Watermarking](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#handling-late-data-and-watermarking)

[Join Operations](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#join-operations)

[Unsupported Operations](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#unsupported-operations)

[Starting Streaming Queries](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#starting-streaming-queries)

[Output Modes](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#output-modes)

[Output Sinks](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#output-sinks)

[Using Foreach](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach)

[Managing Streaming Queries](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#managing-streaming-queries)

[Monitoring Streaming Queries](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#monitoring-streaming-queries)

[Interactive APIs](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#interactive-apis)

[Asynchronous API](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#asynchronous-api)

[Recovering from Failures with Checkpointing](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#recovering-from-failures-with-checkpointing)

[Where to go from here](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#where-to-go-from-here)

# Overview

结构化流是一种基于Spark SQL引擎构建的可扩展且容错的流处理引擎。 您可以使用与表达静态数据的批量计算相同的方式来表达流式计算。 Spark SQL引擎将逐渐持续运行它，并在流数据持续到达时更新最终结果。 您可以使用Scala，Java或Python中的Dataset / DataFrame API来表达流聚合，事件时间窗口，流到批处理连接等。计算在相同的优化的Spark SQL引擎上执行。 最后，系统通过检查点和预写日志确保端到端的一次容错保证。 简而言之，结构化流提供快速，可扩展，容错，端到端的一次性流处理，而无需用户理解流。

结构化Streaming仍然是Spark 2.1中的ALPHA，API仍然是实验性的。 在本指南中，我们将介绍编程模型和API。 首先，我们从一个简单的例子开始 - 一个流字数。

## 快速示例

假设您希望保持从监听TCP套接字的数据服务器接收的文本数据的运行字数。 我们来看看如何使用结构化流式表达这一点。 您可以在Scala / Java / Python中看到完整的代码。 如果您下载Spark，可以直接运行该示例。 在任何情况下，让我们逐步了解示例并了解它的工作原理。 首先，我们必须导入必要的类并创建一个本地SparkSession，这是与Spark相关的所有功能的起点。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_0)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_0)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_0)

from pyspark.sql import SparkSession

from pyspark.sql.functions import explode

from pyspark.sql.functions import split

spark = SparkSession \

.builder \

.appName("StructuredNetworkWordCount") \

.getOrCreate()

接下来，我们创建一个流DataFrame，它表示从本地主机侦听的服务器接收的文本数据：9999，并转换DataFrame以计算字数。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_1)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_1)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_1)

# Create DataFrame representing the stream of input lines from connection to localhost:9999

lines = spark \

.readStream \

.format("socket") \

.option("host", "localhost") \

.option("port", 9999) \

.load()

# Split the lines into words

words = lines.select(

explode(

split(lines.value, " ")

).alias("word")

)

# Generate running word count

wordCounts = words.groupBy("word").count()

这行DataFrame表示包含流文本数据的无界表。 此表包含一列名为“value”的字符串，并且流文本数据中的每一行都将成为表中的一行。 请注意，这并不是正在收到任何数据，因为我们只是设置转换，还没有开始。 接下来，我们使用了两个内置的SQL函数 - 拆分和分解，将每行分成多行，每行都有一个单词。 此外，我们使用函数别名将新列命名为“word”。 最后，我们已经通过将数据集中的唯一值分组并对它们进行计数来定义wordCounts DataFrame。 请注意，这是一个流式DataFrame，它表示流的运行字数。

我们现在已经设置了关于流数据的查询。 剩下的就是实际开始接收数据并计算计数。 为此，我们将其设置为在每次更新时将控制台的整套计数（由outputMode（“complete”）指定）打印出来。 然后使用start（）开始流式计算。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_2)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_2)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_2)

# Start running the query that prints the running counts to the console

query = wordCounts \

.writeStream \

.outputMode("complete") \

.format("console") \

.start()

query.awaitTermination()

执行此代码后，流式计算将在后台启动。 查询对象是该活动流查询的句柄，我们已经决定使用query.awaitTermination（）等待查询的终止，以防止查询处于活动状态时退出进程。

要实际执行此示例代码，您可以在自己的Spark应用程序中编译代码，或者一旦下载了Spark即可运行该示例。 我们正在展示后者。 您将首先需要运行Netcat（大多数类Unix系统中的一个小型实用程序）作为数据服务器

$ nc -lk 9999

Then, in a different terminal, you can start the example by using

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_3)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_3)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_3)

|  |  |
| --- | --- |
| TERMINAL 1:  # Running Netcat  $ nc -lk 9999  apache spark  apache hadoop  ... | [**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_4)  [**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_4)  [**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_4)  # TERMINAL 2: RUNNING structured\_network\_wordcount.py  $ ./bin/spark-submit examples/src/main/python/sql/streaming/structured\_network\_wordcount.py localhost 9999  -------------------------------------------  Batch: 0  -------------------------------------------  +------+-----+  | value|count|  +------+-----+  |apache| 1|  | spark| 1|  +------+-----+  -------------------------------------------  Batch: 1  -------------------------------------------  +------+-----+  | value|count|  +------+-----+  |apache| 2|  | spark| 1|  |hadoop| 1|  +------+-----+  ... |

$ ./bin/spark-submit examples/src/main/python/sql/streaming/structured\_network\_wordcount.py localhost 9999

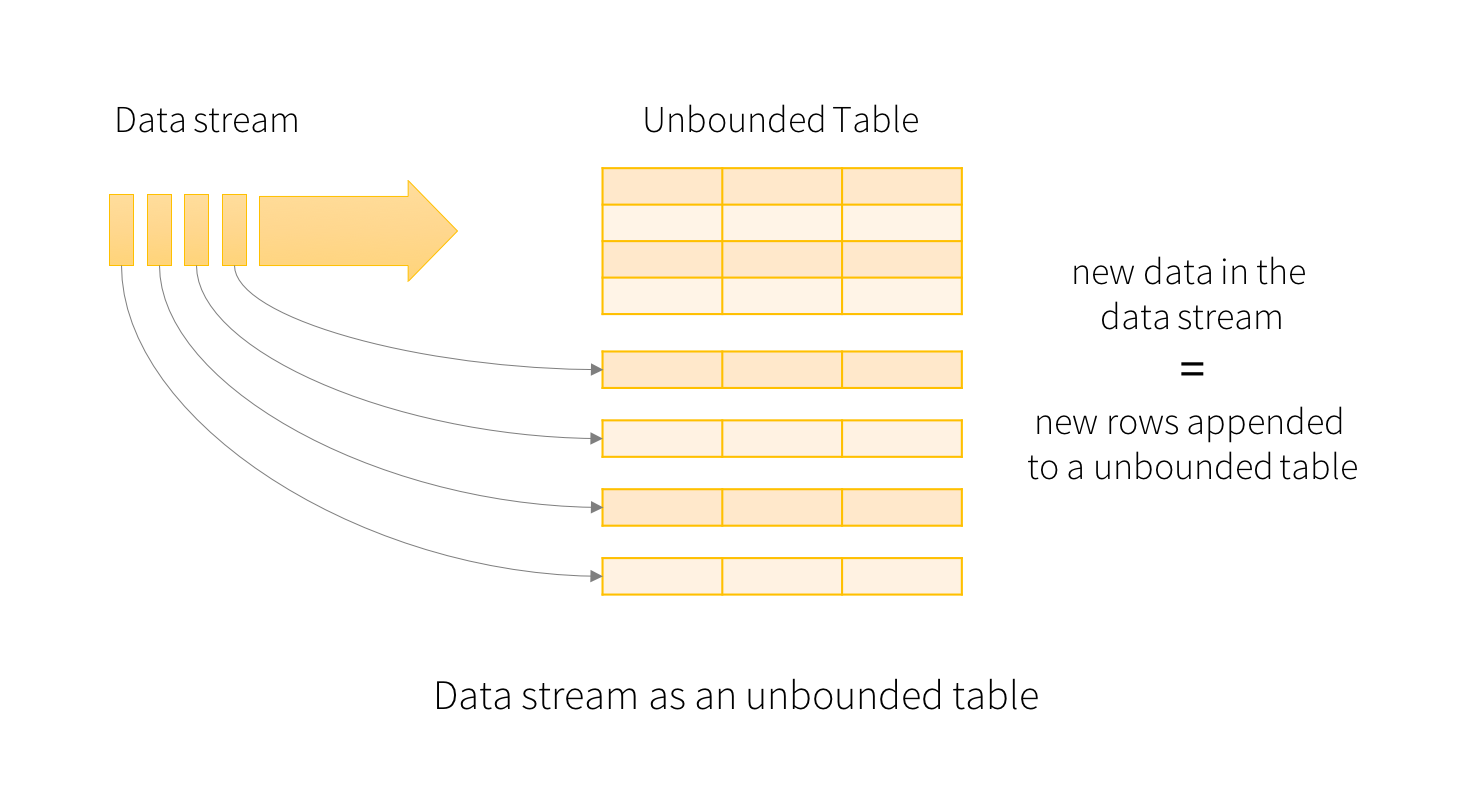
然后，在运行netcat服务器的终端中输入的任何行将每秒计数并打印在屏幕上。 它会看起来像下面这样。

# 编程模型

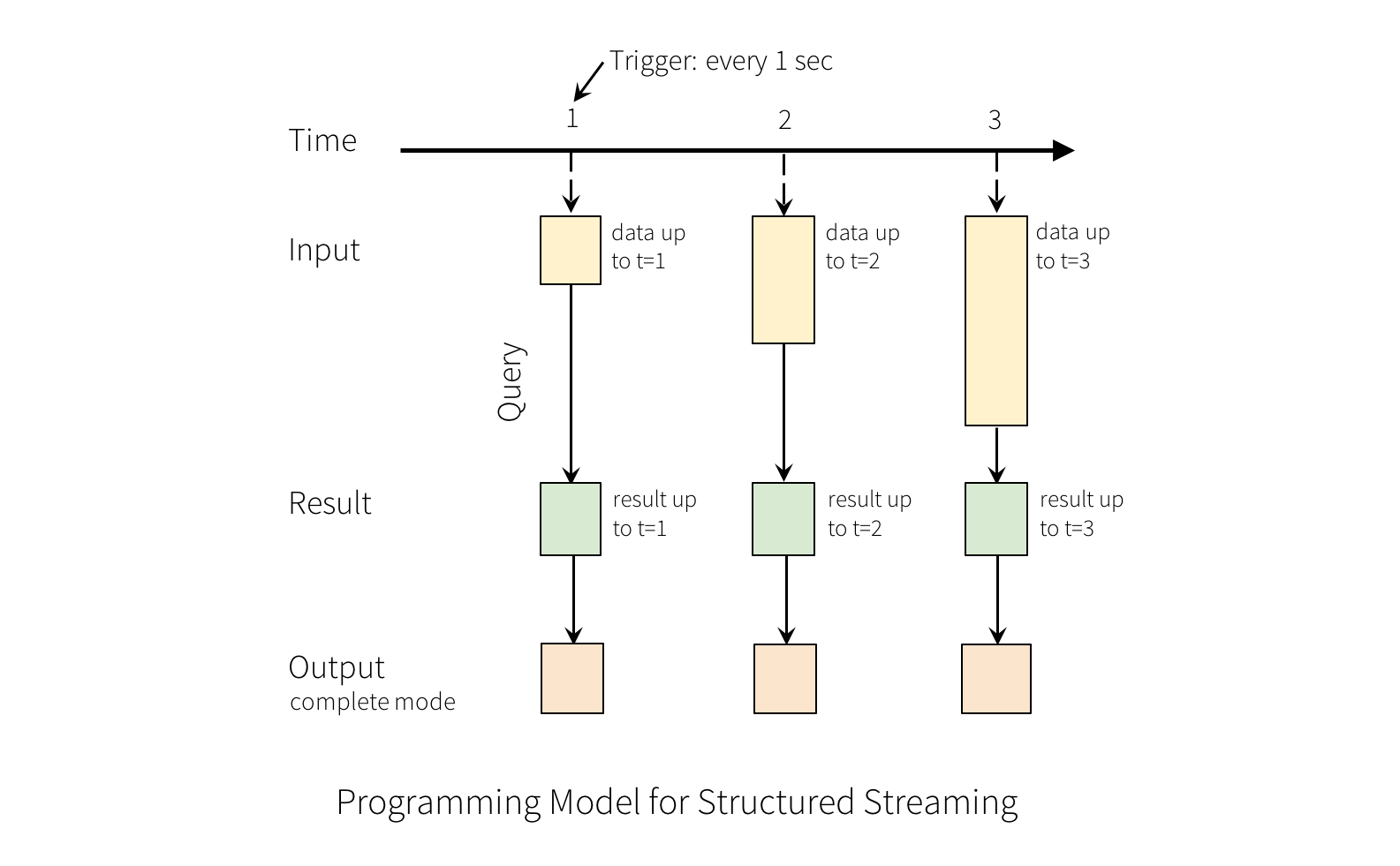
结构化流式传输的关键思想是将实时数据流视为不断附加的表格。 这导致与批处理模型非常相似的新流处理模型。 您将在静态表格上将流式计算表示为标准批量查询，而Spark会在无界输入表上将其作为增量查询来运行。 让我们更详细地了解这个模型。

## 基本概念

将输入数据流视为“输入表”。 到达流的每个数据项都像追加到输入表的新行一样。



对输入的查询将生成“结果表”。 每个触发间隔（例如，每1秒），新行将附加到输入表，最终更新结果表。 无论何时更新结果表，我们都希望将更改的结果行写入外部接收器。



“输出”定义为写入外部存储器的内容。可以以不同的模式定义输出：

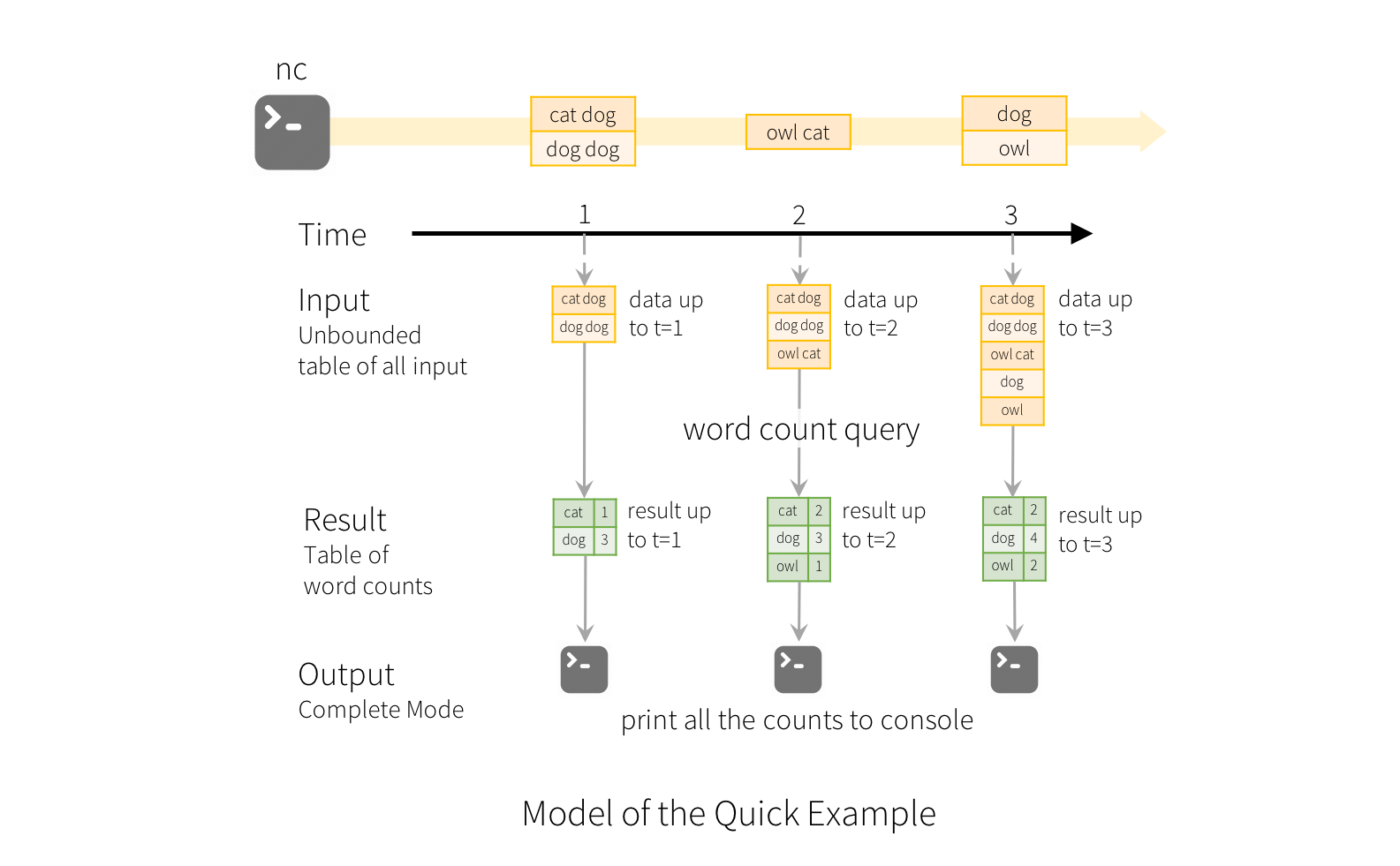
完整模式 - 整个更新的结果表将写入外部存储。由存储连接器决定如何处理整个表的写入。

附加模式 - 只有结果表中自上次触发后附加的新行将被写入外部存储。这仅适用于不期望更改结果表中现有行的查询。

更新模式 - 只有自上次触发后结果表中更新的行将被写入外部存储（可用于Spark 2.1.1）。请注意，这与完全模式不同，因为此模式仅输出自上次触发以来更改的行。如果查询不包含聚合，它将等效于附加模式。

请注意，每个模式适用于某些类型的查询。这将在后面详细讨论。

为了说明这个模型的使用，让我们在上面的快速示例的上下文中了解模型。第一行DataFrame是输入表，最后的wordCounts DataFrame是结果表。请注意，流行线DataFrame生成wordCounts的查询与静态DataFrame完全相同。但是，当该查询启动时，Spark将不断地从套接字连接检查新数据。如果有新数据，Spark将运行一个“增量”查询，将先前的运行计数与新数据相结合，以计算更新的计数，如下所示。



这种模式与许多其他流处理引擎显着不同。 许多流媒体系统需要用户自己维护运行聚合，因此必须对容错和数据一致性（至少一次，或最多一次，或一次）进行说明。 在这个模型中，当有新数据时，Spark负责更新结果表，从而减轻用户对它的推理。 例如，我们来看看这个模型如何处理基于事件时间的处理和迟到的数据。

## 处理事件时间和延迟数据

事件时间是嵌入数据本身的时间。对于许多应用程序，您可能希望在此事件时间进行操作。例如，如果要每分钟获取IoT设备生成的事件数，那么您可能希望使用数据生成的时间（即数据中的事件时间），而不是Spark接收到的时间他们。这个事件时间在这个模型中非常自然地表现出来 - 来自设备的每个事件都是表中的一行，事件时间是行中的列值。这允许基于窗口的聚合（例如每分钟的事件数）仅仅是偶数列上的特殊类型的分组和聚合 - 每个时间窗口是组，并且每行可以属于多个窗口/组。因此，可以在静态数据集（例如来自收集的设备事件日志）以及数据流上一致地定义基于事件时间窗口的聚合查询，从而使用户的使用寿命更加容易。

此外，该模型自然地处理基于事件时间晚于预期的数据。由于Spark正在更新结果表，它可以完全控制更新旧的聚合，当有迟到的数据，以及清理旧的聚合以限制中间状态数据的大小。由于Spark 2.1，我们支持水印，允许用户指定后期数据的阈值，并允许引擎相应地清理旧状态。稍后将在“窗口操作”部分中详细介绍这些。

## 容错语义

提供端到端的一次性语义是结构化流媒体设计背后的关键目标之一。 为了实现这一点，我们设计了结构化流源，汇和执行引擎，以便可靠地跟踪处理的确切进度，以便它可以通过重新启动和/或重新处理来处理任何类型的故障。 假设每个流源具有偏移量（类似于Kafka偏移量或Kinesis序列号）以跟踪流中的读取位置。 引擎使用检查点和写入日志记录每个触发器中正在处理的数据的偏移范围。 流宿设计为处理后处理的幂等。 一起使用可重放源和幂等接收器，结构化流可以在任何故障下确保端到端完全一次的语义。

## API使用Datasets和DataFrames

由于Spark 2.0，DataFrames和Datasets可以表示静态，有界数据，以及流式传输，无界数据。 与静态数据集/ DataFrames类似，您可以使用通用入口点SparkSession（Scala / Java / Python文档）从流源创建流式DataFrames / Datasets，并将其作为静态DataFrames / Datasets应用相同的操作。 如果您不熟悉Datasets / DataFrames，强烈建议您使用DataFrame / Dataset编程指南来熟悉它们。

# 创建流数据帧和流数据集

Streaming DataFrames可以通过SparkSession.readStream（）返回的DataStreamReader接口（Scala / Java / Python文档）创建。与创建静态DataFrame的读取界面类似，您可以指定源 - 数据格式，模式，选项等的详细信息。

## 输入源

在Spark 2.0中，有几个内置源代码。

文件源 - 将目录中写入的文件作为数据流读取。支持的文件格式为text，csv，json，parquet。有关更多最新列表，请参阅DataStreamReader界面的文档，以及每个文件格式的支持选项。请注意，文件必须以原子方式放置在给定的目录中，这在大多数文件系统中可以通过文件移动操作实现。

Kafka来源 - 来Kafka的投票数据。它与Kafka代理商版本0.10.0或更高版本兼容。有关详细信息，请参阅“卡夫卡集成指南”。

套接字源（用于测试） - 从套接字连接读取UTF8文本数据。侦听服务器插座位于驱动程序中。请注意，这仅应用于测试，因为这不提供端到端容错保证。

某些来源不是容错的，因为它们不能保证在故障后可以使用检查点偏移量重播数据。请参阅上一节关于容错语义的部分。以下是Spark中所有资源的详细信息。

|  |  |  |  |
| --- | --- | --- | --- |
| Source | Options | Fault-tolerant | Notes |
| **File source** | path：路径到输入目录，并且与所有文件格式通用。  有关特定于文件格式的选项，请参阅DataStreamReader（Scala / Java / Python）中的相关方法。 例如。 对于“HIVE”格式选项见DataStreamReader  ([Scala](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.sql.streaming.DataStreamReader)/[Java](http://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/streaming/DataStreamReader.html)/[Python](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.streaming.DataStreamReader)). E.g. for "parquet" format options see DataStreamReader.parquet() | Yes | 支持glob路径，但不支持多个逗号分隔的路径/ globs。 |
| **Socket Source** | host: 指定的连接地址  port: 指定通信的端口 | No |  |
| **Kafka Source** | See the [Kafka Integration Guide](http://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html). | Yes |  |
|  |  |  |  |

Here are some examples.

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_5)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_5)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_5)

spark = SparkSession. ...

# Read text from socket

socketDF = spark \

.readStream \

.format("socket") \

.option("host", "localhost") \

.option("port", 9999) \

.load()

socketDF.isStreaming() # Returns True for DataFrames that have streaming sources

socketDF.printSchema()

# Read all the csv files written atomically in a directory

userSchema = StructType().add("name", "string").add("age", "integer")

csvDF = spark \

.readStream \

.option("sep", ";") \

.schema(userSchema) \

.csv("/path/to/directory") # Equivalent to format("csv").load("/path/to/directory")

这些示例生成无类型的流DataFrames，这意味着DataFrame的架构在编译时未被检查，只在运行时在查询提交时进行检查。 一些操作，如map，flatMap等，需要在编译时知道该类型。 要做到这一点，您可以使用与静态DataFrame相同的方法将这些未类型化的流DataFrames转换为类型化的流数据集。 有关详细信息，请参阅SQL编程指南。 此外，有关支持的流媒体源的更多详细信息将在文档后面讨论。

## 流式DataFrames / Datasets的模式推理和分区

默认情况下，基于文件的源的结构化流式传输需要您指定模式，而不是依靠Spark自动推断。这种限制确保了即使在出现故障的情况下，一致的模式将被用于流式查询。对于特殊用例，您可以通过将spark.sql.streaming.schemaInference设置为true来重新启用模式推断。

当存在名为/ key = value /的子目录时，会发生分区发现，并且列表将自动递归到这些目录中。如果这些列显示在用户提供的模式中，则它们将根据正在读取的文件的路径由Spark填充。构成分区方案的目录必须在查询启动时存在，并且必须保持静态。例如，可以添加/ data / year = 2016 / when / data / year = 2015 /存在，但更改分区列无效（即通过创建目录/ data / date = 17 /）。

## 流数据帧/数据集的操作

您可以对流式DataFrames / Datasets应用各种操作，从无类型，类似SQL的操作（例如select，where，groupBy）到类型类似RDD的操作（例如，map，filter，flatMap）。 有关详细信息，请参阅SQL编程指南。 我们来看一下可以使用的几个示例操作。

# 基本操作 - 选择，投影，聚合

DataFrame / Dataset的大多数常见操作都支持流式传输。 本节稍后将讨论不支持的少数操作。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_6)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_6)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_6)

df = ... # streaming DataFrame with IOT device data with schema { device: string, deviceType: string, signal: double, time: DateType }

# Select the devices which have signal more than 10

df.select("device").where("signal > 10")

# Running count of the number of updates for each device type

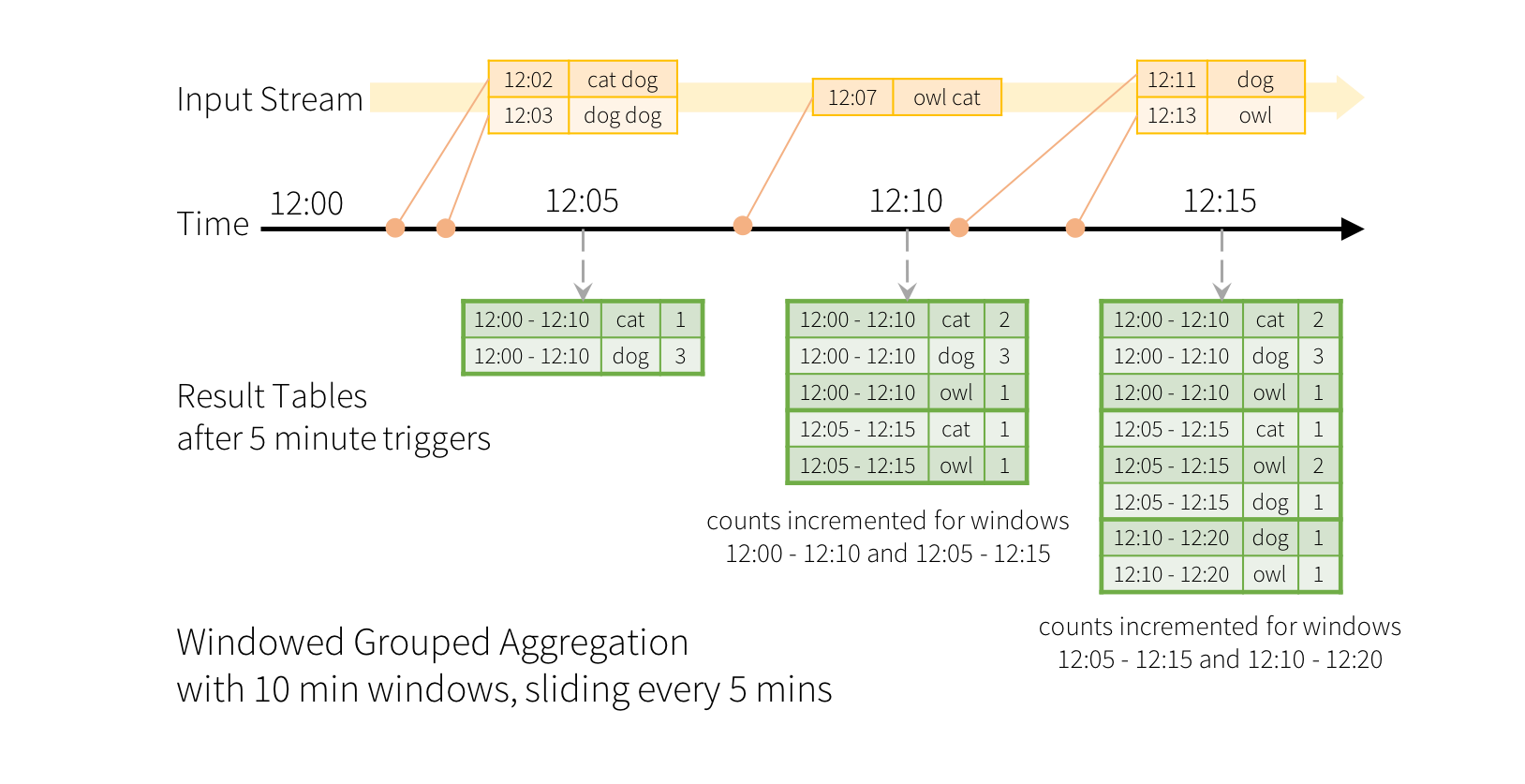
df.groupBy("deviceType").count()

# 事件时间窗口操作

通过结构化流式，滑动事件时间窗口的聚合很简单。了解基于窗口的聚合的关键思想与分组聚合非常相似。在分组聚合中，为用户指定的分组列中的每个唯一值维护聚合值（例如计数）。在基于窗口的聚合的情况下，对于行的事件时间的每个窗口，维护聚合值。让我们以一个例证了解这一点。

想象一下，我们的快速示例被修改，并且流现在包含生成行的时间和行。我们不想运行字数，而是要在10分钟的窗口内计数单词，每5分钟更新一次。也就是说，在10分钟的窗口12:00 - 12:10，12:05 - 12:15，12:10 - 12:20等之间收到的单词计数。请注意，12:00 - 12:10表示数据12点后到12点10分抵达。现在，考虑在12:07收到的一个字。这个词应该增加对应于两个窗口的计数12:00 - 12:10和12:05 - 12:15。因此，计数将被二者分组，分组密钥（即单词）和窗口（可以从事件时间计算）来索引。

结果表将如下所示。



由于此窗口类似于分组，因此在代码中可以使用groupBy（）和window（）操作来表示窗口聚合。 您可以在Scala / Java / Python中看到以下示例的完整代码。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_7)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_7)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_7)

words = ... # streaming DataFrame of schema { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of each group

windowedCounts = words.groupBy(

window(words.timestamp, "10 minutes", "5 minutes"),

words.word

).count()

# 处理晚数据和水印

现在考虑如果一个事件迟到应用程序会发生什么。 例如，例如，在12:04（即事件时间）生成的单词可以由应用程序在12:11接收。 应用程序应该使用时间12:04而不是12:11更新窗口的较旧计数12:00 - 12:10。 这在我们基于窗口的分组中自然发生 - 结构化流可以长时间维持部分聚合的中间状态，以便后期数据可以正确地更新旧窗口的聚合，如下所示。

但是，为了运行这个查询几天，系统必须绑定其累积的内存中间状态的数量。这意味着系统需要知道何时可以从内存状态中删除旧聚合，因为应用程序不会再为该集合接收到较晚的数据。为了实现这一点，在Spark 2.1中，我们引入了水印，让引擎自动跟踪数据中的当前事件时间，并尝试相应地清除旧状态。您可以通过指定事件时间列来定义查询的水印，并根据事件时间来确定数据预计的延迟。对于从时间T开始的特定窗口，引擎将保持状态，并允许延迟数据更新状态，直到发动机 - 后期阈值> T看到的最大事件时间为止。换句话说，阈值内的晚期数据将被聚合，但是晚于阈值的数据将被丢弃。让我们以一个例子来理解这一点。我们可以使用Watermark（）轻松定义上一个例子中的水印，如下所示。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_8)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_8)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_8)

words = ... # streaming DataFrame of schema { timestamp: Timestamp, word: String }

# Group the data by window and word and compute the count of each group

windowedCounts = words \

.withWatermark("timestamp", "10 minutes") \

.groupBy(

window(words.timestamp, "10 minutes", "5 minutes"),

words.word) \

.count()

在这个例子中，我们正在定义“timestamp”列的查询的水印，并将“10分钟”定义为允许数据延迟的阈值。 如果此查询在更新输出模式下运行（稍后在“输出模式”部分中讨论），则引擎将继续更新Resule表中窗口的计数，直到窗口比水印早，在“ 时间戳“10分钟。 这是一个例子。

如图所示，引擎跟踪的最大事件时间是蓝色虚线，每个触发开始时设置为（最大事件时间 - '10分钟）的水印是红线。例如，当引擎观察数据（12:14，狗），它为下一个触发器设置水印为12:04。该水印允许发动机保持中间状态另外10分钟，以允许后期数据计数。例如，数据（12:09，猫）是不正常的，晚了，它落在Windows 12:05 - 12:15和12:10 - 12:20。由于它仍然在触发器的水印12:04之前，引擎仍然将中间计数保持为状态，并正确更新相关窗口的计数。然而，当水印更新为12:11时，窗口的中间状态（12:00 - 12:10）被清除，所有后续数据（例如（12:04，驴））被认为是“太晚”，因此被忽略。请注意，按照更新模式的规定，每次触发后，更新的计数（即紫色行）将被写入作为触发输出进行吸收。

某些接收器（例如文件）可能不支持更新模式所需的细粒度更新。要与他们一起工作，我们还支持附加模式，只有最后的计数被写入下沉。这如下所示。

与之前的更新模式类似，引擎维护每个窗口的中间计数。 但是，部分计数不会更新到结果表，而不是写入sink。 发动机等待迟到的“10分钟”进行计数，然后将窗口<水印的中间状态丢弃，并将最终计数附加到结果表/汇点。 例如，只有在将水印更新为12:11之后，窗口12:00 - 12:10的最终计数才附加到结果表中。

水印清理聚合状态的条件重要的是要注意，为了清除聚合查询中的状态（从Spark 2.1.1开始，将来会更改），必须满足以下条件。

输出模式必须是追加或更新。完整模式要求保留所有聚合数据，因此不能使用水印来中断状态。有关每种输出模式的语义的详细说明，请参见“输出模式”部分。

聚合必须具有事件时间列或事件时间列上的窗口。

必须在与聚合中使用的时间戳列相同的列上调用水印。例如，df.withWatermark（“time”，“1 min”）。groupBy（“time2”）。在附加输出模式中count（）无效，因为水印在不同的列上定义为聚合列。

必须在聚合之前调用水印才能使用水印细节。例如，在附加输出模式下，df.groupBy（“time”）。count（）。withWatermark（“time”，“1 min”）无效。

# 合并业务

流式数据帧可以与静态数据帧连接，以创建新的流数据帧。 这里有几个例子。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_9)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_9)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_9)

staticDf = spark.read. ...

streamingDf = spark.readStream. ...

streamingDf.join(staticDf, "type") # inner equi-join with a static DF

streamingDf.join(staticDf, "type", "right\_join") # right outer join with a static DF

## 不支持的操作

DataFrames / Datasets不支持DataFrame / Dataset操作。其中一些如下。

流数据集不支持多个流聚合（即流DF上的聚合链）。

流数据集不支持限制并采取前N行。

不支持流数据集上的不同操作。

只有在聚合和完全输出模式下，流数据集才支持排序操作。

有条件地支持流和静态数据集之间的外连接。

不支持与流数据集完全外连接

不支持左侧外连接与右侧的流数据集

右侧外连接与左侧的流数据集不支持

两个流数据集之间的任何形式的连接都不支持。

此外，还有一些Dataset方法将不适用于流数据集。它们是立即运行查询并返回结果的操作，这在流数据集上没有意义。相反，这些功能可以通过显式启动流式查询来完成（参见下一节）。

**count（） - 无法从流数据集返回单个计数。而是使用ds.groupBy.count（）返回一个包含运行计数的流数据集。**

**foreach（） - 而不是使用ds.writeStream.foreach（...）（见下一节）。**

**show（） - 而是使用控制台接收器（请参阅下一节）。**

如果您尝试任何这些操作，您将看到一个AnalysisException，如“操作XYZ不支持流数据帧/数据集”。虽然其中一些可能在未来版本的Spark中得到支持，但还有其他一些基本上很难在流数据上高效地实现。例如，不支持对输入流进行排序，因为它需要跟踪流中接收到的所有数据。因此，从根本上难以有效执行。

# 开始流式查询

一旦定义了最终结果DataFrame / Dataset，剩下的就是开始流计算。为此，您必须使用通过Dataset.writeStream（）返回的DataStreamWriter（Scala / Java / Python文档）。您将必须在此界面中指定以下一个或多个。

输出接收器的详细信息：数据格式，位置等

输出模式：指定写入输出接收器的内容。

查询名称：可选地，指定用于标识的查询的唯一名称。

触发间隔：可选，指定触发间隔。如果未指定，系统将在上一次处理完成后立即检查新数据的可用性。如果由于先前的处理尚未完成而导致触发时间错误，则系统将尝试在下一个触发点触发，而不是在处理完成后立即触发。

检查点位置：对于可以保证端到端容错的某些输出接收器，请指定系统将写入所有检查点信息的位置。这应该是与HDFS兼容的容错文件系统中的目录。检查点的语义将在下一节中进行更详细的讨论。

## 输出模式

有几种类型的输出模式。

附加模式（默认） - 这是默认模式，只有从上次触发后添加到结果表的新行将被输出到接收器。 只有那些添加到结果表中的行从不会改变的查询才支持这一点。 因此，该模式保证每行只能输出一次（假定容错接收器）。 例如，只有select，where，map，flatMap，filter，join等的查询将支持Append模式。

完成模式 - 每次触发后，整个结果表将被输出到接收器。 聚合查询支持这一点。

更新模式 - （自Spark 2.1.1起可用）只有结果表中自上次触发后更新的行将被输出到接收器。 更多信息将在以后的版本中添加。

不同类型的流式查询支持不同的输出模式。 以下是兼容性矩阵。

|  |  |  |  |
| --- | --- | --- | --- |
| Query Type |  | Supported Output Modes | Notes |
| Queries without aggregation | | Append, Update | 不支持完全模式，因为将所有数据保存在结果表中是不可行的。 |
| Queries with aggregation | Aggregation on event-time with watermark | Append, Update, Complete | 附加模式使用水印来删除旧的聚合状态。 但是，窗口化聚合的输出会延迟“withWatermark（）”中指定的晚期阈值，因为模式语义可以在结果表中定义后将其添加到结果表中一次（即在水印被交叉之后）。 有关详细信息，请参阅后期数据部分。  更新模式使用水印来删除旧的聚合状态。  完成模式不会丢弃旧的聚合状态，因为根据定义，此模式保留结果表中的所有数据。 |
| Other aggregations | Complete, Update | 由于没有定义水印（仅在其他类别中定义），旧的聚合状态不会被丢弃。  不支持附加模式，因为聚合可以更新，从而违反了此模式的语义。 |
|  |  |  |  |

# 输出接收端

有几种类型的内置输出接收器。

文件宿 - 将输出存储到目录中。

writeStream

.format("parquet") // can be "orc", "json", "csv", etc.

.option("path", "path/to/destination/dir")

.start()

Foreach sink - Runs arbitrary computation on the records in the output. See later in the section for more details.

writeStream

.foreach(...)

.start()

控制台接收器（用于调试） - 每次触发时，将输出打印到控制台/ stdout。 都支持“追加”和“完成”输出模式。 这应该用于低数据量的调试目的，因为在每次触发后，整个输出被收集并存储在驱动程序的存储器中。

writeStream

.format("console")

.start()

存储器接收器（用于调试） - 输出作为内存表存储在存储器中。 都支持“追加”和“完成”输出模式。 由于整个输出被收集并存储在驱动程序的存储器中，因此应用于低数据量的调试目的。 因此，请谨慎使用。

writeStream

.format("memory")

.queryName("tableName")

.start()

某些接收器不容错，因为它们不保证输出的持久性，仅用于调试目的。 请参阅上一节关于容错语义的部分。 以下是Spark中所有接收器的详细信息。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sink | Supported Output Modes | Options | Fault-tolerant | Notes |
| **File Sink** | Append | path：路径到输出目录，必须指定。 maxFilesPerTrigger：每个触发器中要考虑的最大新文件数（默认值：无最大值）  latestFirst：是否首先处理最新的新文件，当有大量的文件积压（default：false）时很有用  有关特定于文件格式的选项，请参阅DataFrameWriter（Scala / Java / Python）中的相关方法。 例如。 对于“镶木地板”格式选项请参阅DataFrameWriter.parquet（） | Yes | 支持写入分区表。 按时间划分可能是有用的。 |
| **Foreach Sink** | Append, Update, Compelete | None | Depends on ForeachWriter implementation | More details in the [next section](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#using-foreach) |
| **Console Sink** | Append, Update, Complete | numRows：打印每个触发器的行数（默认值：20）  truncate：是否截断输出太长（默认值：true） | No |  |
| **Memory Sink** | Append, Complete | None | 否。但在完全模式下，重新启动的查询将重新创建完整的表。 | Table name is the query name. |
|  |  |  |  |  |

请注意，您必须调用start（）来实际启动查询的执行。 这将返回一个StreamingQuery对象，它是持续运行的执行的句柄。 您可以使用此对象来管理查询，我们将在下一小节中讨论。 现在，让我们通过几个例子了解所有这些。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_10)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_10)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_10)

# ========== DF with no aggregations ==========

noAggDF = deviceDataDf.select("device").where("signal > 10")

# Print new data to console

noAggDF \

.writeStream \

.format("console") \

.start()

# Write new data to Parquet files

noAggDF \

.writeStream \

.format("parquet") \

.option("checkpointLocation", "path/to/checkpoint/dir") \

.option("path", "path/to/destination/dir") \

.start()

# ========== DF with aggregation ==========

aggDF = df.groupBy("device").count()

# Print updated aggregations to console

aggDF \

.writeStream \

.outputMode("complete") \

.format("console") \

.start()

# Have all the aggregates in an in memory table. The query name will be the table name

aggDF \

.writeStream \

.queryName("aggregates") \

.outputMode("complete") \

.format("memory") \

.start()

spark.sql("select \* from aggregates").show() # interactively query in-memory table

## 使用Foreach

foreach操作允许在输出数据上计算任意操作。从Spark 2.1起，这只适用于Scala和Java。要使用它，您将必须实现接口ForeachWriter（Scala / Java文档），该接口具有在触发器之后生成作为输出的行序列时调用的方法。请注意以下要点。

作者必须是可序列化的，因为它将被序列化并发送给执行者执行。

所有三种方法，开放，进程和关闭将被称为执行者。

只有当打开方法被调用时，作者必须执行所有的初始化（例如打开连接，启动事务等）。请注意，如果在创建对象之后，类中有任何初始化，那么该初始化将在驱动程序中发生（因为这是正在创建的实例），这可能不是您打算的。

版本和分区是打开的两个参数，唯一表示一组需要推出的行。版本是每个触发器增加的单调递增的id。分区是表示输出分区的id，因为输出是分布式的，并且将在多个执行器上处理。

打开可以使用版本和分区来选择是否需要写入行的顺序。因此，它可以返回true（继续写入）或false（不需要写入）。如果返回false，则不会在任何行上调用进程。例如，在部分故障之后，失败的触发器的一些输出分区可能已经被提交到数据库。基于存储在数据库中的元数据，作者可以识别已经被提交的分区，并因此返回false以跳过再次提交它们。

每当打开被调用时，关闭也将被调用（除非JVM由于某些错误而退出）。即使打开返回false也是如此。如果在处理和写入数据时有任何错误，则关闭将被调用与错误。您有责任清理在open中创建的状态（例如，连接，事务等），以免资源泄漏。

## 管理流式查询

查询启动时创建的StreamingQuery对象可用于监视和管理查询。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_11)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_11)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_11)

query = df.writeStream.format("console").start() # get the query object

query.id() # get the unique identifier of the running query that persists across restarts from checkpoint data

query.runId() # get the unique id of this run of the query, which will be generated at every start/restart

query.name() # get the name of the auto-generated or user-specified name

query.explain() # print detailed explanations of the query

query.stop() # stop the query

query.awaitTermination() # block until query is terminated, with stop() or with error

query.exception() # the exception if the query has been terminated with error

query.recentProgress() # an array of the most recent progress updates for this query

query.lastProgress() # the most recent progress update of this streaming query

您可以在单个SparkSession中启动任意数量的查询。 他们都将同时运行共享群集资源。 您可以使用sparkSession.streams（）获取可用于管理当前活动查询的StreamingQueryManager（Scala / Java / Python文档）。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_12)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_12)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_12)

spark = ... # spark session

spark.streams().active # get the list of currently active streaming queries

spark.streams().get(id) # get a query object by its unique id

spark.streams().awaitAnyTermination() # block until any one of them terminates

监控流式查询

有两个API用于监视和调试活动查询 - 以交互方式和异步方式。

## 互动API

您可以使用streamingQuery.lastProgress（）和streamingQuery.status（）直接获取活动查询的当前状态和指标。 lastProgress（）在Scala和Java中返回一个StreamingQueryProgress对象，并在Python中返回与该字段相同的字典。 它具有关于流的最后触发器的进展的所有信息 - 处理哪些数据，处理速率，延迟等等。还有streamingQuery.recentProgress返回最后几个进度的数组。

此外，streamingQuery.status（）在Scala和Java中返回StreamingQueryStatus对象，并在Python中返回与字段相同的字典。 它提供有关查询立即执行的信息 - 触发器是活动的，正在处理的数据等。

这里有几个例子。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_13)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_13)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_13)

query = ... # a StreamingQuery

print(query.lastProgress)

'''

Will print something like the following.

{u'stateOperators': [], u'eventTime': {u'watermark': u'2016-12-14T18:45:24.873Z'}, u'name': u'MyQuery', u'timestamp': u'2016-12-14T18:45:24.873Z', u'processedRowsPerSecond': 200.0, u'inputRowsPerSecond': 120.0, u'numInputRows': 10, u'sources': [{u'description': u'KafkaSource[Subscribe[topic-0]]', u'endOffset': {u'topic-0': {u'1': 134, u'0': 534, u'3': 21, u'2': 0, u'4': 115}}, u'processedRowsPerSecond': 200.0, u'inputRowsPerSecond': 120.0, u'numInputRows': 10, u'startOffset': {u'topic-0': {u'1': 1, u'0': 1, u'3': 1, u'2': 0, u'4': 1}}}], u'durationMs': {u'getOffset': 2, u'triggerExecution': 3}, u'runId': u'88e2ff94-ede0-45a8-b687-6316fbef529a', u'id': u'ce011fdc-8762-4dcb-84eb-a77333e28109', u'sink': {u'description': u'MemorySink'}}

'''

print(query.status)

'''

Will print something like the following.

{u'message': u'Waiting for data to arrive', u'isTriggerActive': False, u'isDataAvailable': False}

'''

异步API

您还可以通过附加StreamingQueryListener（Scala / Java文档）异步监视与SparkSession关联的所有查询。 一旦您使用sparkSession.streams.attachListener（）附加您的自定义StreamingQueryListener对象，您将在查询启动和停止时以及在活动查询中进行进度时获得回调。 这是一个例子，

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_14)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_14)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_14)

Not available in Python.

从故障恢复与检查点

如果发生故障或故意关机，您可以恢复之前的查询的进度和状态，并继续停止。 这是使用检查点和写入日志完成的。 您可以使用检查点位置配置查询，并且查询将将所有进度信息（即在每个触发器中处理的偏移范围）和运行聚合（例如，快速示例中的字计数）保存到检查点位置。 此检查点位置必须是HDFS兼容文件系统中的路径，并且可以在启动查询时将其设置为DataStreamWriter中的选项。

[**Scala**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_scala_15)

[**Java**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_java_15)

[**Python**](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#tab_python_15)

aggDF \

.writeStream \

.outputMode("complete") \

.option("checkpointLocation", "path/to/HDFS/dir") \

.format("memory") \

.start()

Where to go from here

Examples: See and run the [Scala](https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples/sql/streaming)/[Java](https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples/sql/streaming)/[Python](https://github.com/apache/spark/tree/master/examples/src/main/python/sql/streaming) examples.

Spark Summit 2016 Talk - [A Deep Dive into Structured Streaming](https://spark-summit.org/2016/events/a-deep-dive-into-structured-streaming/)