#### **Input Sources （输入源）**

在 Spark 2.0 中，有一些内置的 sources 。

* File source（文件源） - 以文件流的形式读取目录中写入的文件。支持的文件格式为 text ， csv ， json ， parquet 。有关更多的 up-to-date 列表，以及每种文件格式的支持选项，请参阅 DataStreamReader interface 的文档。请注意，文件必须以 atomically （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 source（Kafka 源） - 来自 Kafka 的 Poll 数据。它与 Kafka broker 的 0.10.0 或者更高的版本兼容。有关详细信息，请参阅 [Kafka Integration 指南](http://spark.apachecn.org/docs/cn/2.2.0/structured-streaming-kafka-integration.html) 。
* Socket source (for testing) （Socket 源（用于测试）） - 从一个 socket 连接中读取 UTF8 文本数据。 listening server socket （监听服务器 socket）位于 driver 。请注意，这只能用于测试，因为它不提供 end-to-end fault-tolerance （端到端的容错）保证。

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Source | Options（选项） | Fault-tolerant（容错） | Notes（说明） |
| File source（文件源） | path: 输入路径的目录，并且与所有文件格式通用。  maxFilesPerTrigger: 每个 trigger （触发器）中要考虑的最大新文件数（默认是: 无最大值）  latestFirst: 是否先处理最新的新文件，当有大量积压的文件时有用（默认: false）  fileNameOnly: 是否仅根据文件名而不是完整路径检查新文件（默认值: false）。将此设置为 `true` ，以下文件将被视为相同的文件，因为它们的文件名 "dataset.txt" 是相同的:  · "file:///dataset.txt"  · "s3://a/dataset.txt"  · "s3n://a/b/dataset.txt"  · "s3a://a/b/c/dataset.txt"  有关特定于 file-format-specific （文件格式）的选项，请参阅DataStreamReader(Scala/Java/Python/R)  中的相关方法。例如，对于 "parquet" 格式选项请参阅 DataStreamReader.parquet() | Yes | 支持 glob 路径，但是不支持多个逗号分隔的 paths/globs 。  基于文件的 sources 的 Structured Streaming 需要您指定 schema （模式），而不是依靠 Spark 自动 infer 。 |
| Socket Source（Socket 源） | host: 连接到的 host ，必须指定  port: 连接的 port （端口），必须指定 | No |  |
| Kafka Source（Kafka 源） | 请查看 Kafka Integration 指南. | Yes |  |

**这些示例生成无类型的 streaming DataFrames ，这意味着在编译时不会检查 DataFrame 的模式，仅在运行时在 query is submitted （查询提交）的时候进行检查。像 map ，flatMap 等这样的操作需要在编译时知道这个类型。要做到这一点，您可以使用与 static DataFrame 相同的方法将这些 untyped （无类型）的 streaming DataFrames 转换为 typed streaming Datasets （类型化的 streaming Datasets ）。**

Window

import spark.implicits.\_

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

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

val windowedCounts = words.groupBy(

window($"timestamp", "10 minutes", "5 minutes"),

$"word"

).count()

WaterMark

import spark.implicits.\_

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

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

val windowedCounts = words

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

.groupBy(

window($"timestamp", "10 minutes", "5 minutes"),

$"word")

.count()

### Join 操作

Streaming DataFrames 可以与 static DataFrames 连接，以创建新的 streaming DataFrames 。