第5章 数据流的操作

在当今互连的设备和服务的世界中，我们每天需要花费数小时查看社交媒体上的最新消息、电商平台上的产品优惠信息，或者查看最新新闻或体育更新。无论是完成手头的工作，还是浏览信息或发送电子邮件，都依赖于智能设备和互联网。从发展的趋势看，应用程序、服务的数量和种类只会随着时间的推移而增长，所以智能终端设备无处不在，并且一直在生成大量数据，这种被广泛称之的物联网不断地改变了数据处理的动力。每当我们以某种形式使用智能手机上的任何服务或应用程序时，实时数据处理就会起作用。而且这种实时数据处理能力很大程度上取决于应用程序的质量和价值，因此，很多互联网公司都将重点放在如何应对数据的实用性和及时性等方面的复杂挑战。

互联网服务提供商正在研究和采用一种非常前沿的平台或基础架构，在此基础上构建可扩展性强、接近实时或实时的处理框架。一切都必须是快速的，并且必须对变化和异常做出反应，至关重要的是数据流的处理和使用都必须尽可能接近实时。这些平台或系统每天会处理大量数据，而且是不确定的连续事件流。

与任何其他数据处理系统一样，在数据收集、存储和数据处理方面也面临着相同的基本挑战，但是由于平台的实时需求，因此增加了复杂性。为了收集此类不确定事件流，随后处理所有此类事件，以生成可以利用的数据价值，我们需要使用高度可扩展的专业架构处理大量事件，因此数十年来已经构建了许多系统用来处理实时的连续数据流，包括AMQ、RabbitMQ、Storm、Kafka、Spark、Flink、Gearpump、Apex。

构建用于处理大量流数据的现代系统具有非常灵活和可扩展的技术，这些技术不仅非常高效，而且比以前能更好地帮助实现了业务目标。使用这些技术，可以从各种数据源中获取处理数据，然后根据需要在各种情景中使用。

另外，流式处理是用于从无限制数据中提取信息的技术。鉴于我们的信息系统是基于有限资源（例如内存和存储容量）的硬件构建的，因此它们可能无法容纳无限制的数据集。取而代之的是，我们观察到的数据形式是在处理系统中接收到的，随时间流逝的事件，我们称其为数据流。相反，我们将有界数据视为已知大小的数据集，可以计算有界数据集中的元素数量。如何处理两种类型的数据集？对于批处理，指的是有限数据集的计算分析。实际上，这意味着可以从某种形式的存储中整体上获得和检索这些数据集，我们在计算过程开始时知道数据集的大小，并且处理过程的持续时间受到限制。相反，在流处理中，我们关注数据到达系统时的处理。考虑到数据流的无限性，只要流中一直传递新数据流式处理，就需要持续运行，理论上讲，这可能是永远的。

总体来说，流式处理系统应用编程和操作技术，以利用有限数量的计算资源处理潜在

的无限数据流成为可能。

5.1　处 理 范 例

现在，为了了解这种实时流传输体系结构如何工作以提供有价值的信息，需要了解流传输体系结构的基本原则。一方面，对于实时流体系结构而言，能够以非常高的速率获取大量数据；另一方面，还应确保所摄取的数据也得到处理。图 5-1 显示了一个通用的流处理系统，其中生产者将事件放入消息传递系统中，同时消费者从消息传递系统中读取消息。

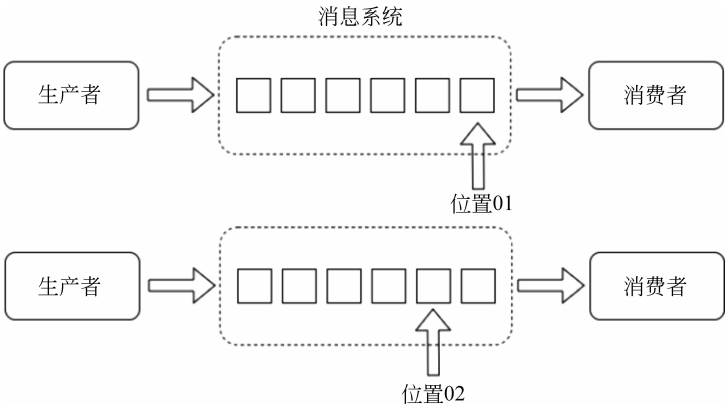

图 5-1　流处理系统

实时流数据的处理可以分为以下三个基本范例：

(1) 至少一次。

(2) 最多一次。

(3) 恰好一次。

下面看一下这三个流处理范例对我们的业务用例意味着什么。虽然对实时事件的恰好处理一次对我们来说是最终的目标，但要始终在不同的情况下实现此目标非常困难，如果实时的复杂性超过这种保证的益处，必须妥协选择其他基本范例。

5.1.1　至少一次

至少一次范例涉及一种机制，当实际处理完事件并且获得结果之后，才保存刚才接收了最后一个事件的位置，这样，如果发生故障并且事件消费者重新启动，消费者将重新读取旧事件并对它们进行处理。但是，由于不能保证接收的事件被全部或部分处理，因此如果事件再次被提取，则可能导致事件重复。这导致事件至少被处理一次的行为。理论上，至少一次范例适用的应用程序涉及更新某些瞬时行情的自动收录器。任何累积总和、计数器或对准确性有依赖的聚合计算(例如求和、分组等)都不适合这种处理范例，因为重复事件将导致不正确的结果。消费者的操作顺序如下：①保存结果；②保存偏移量。图 5-2 显示了如果发生故障并且消费者重新启动时将发生的情况。由于事件已被处理但尚未保

存偏移量，因此消费者将读取先前保存的偏移量，从而导致重复。图 5-2 中，事件 0 被处理两次。

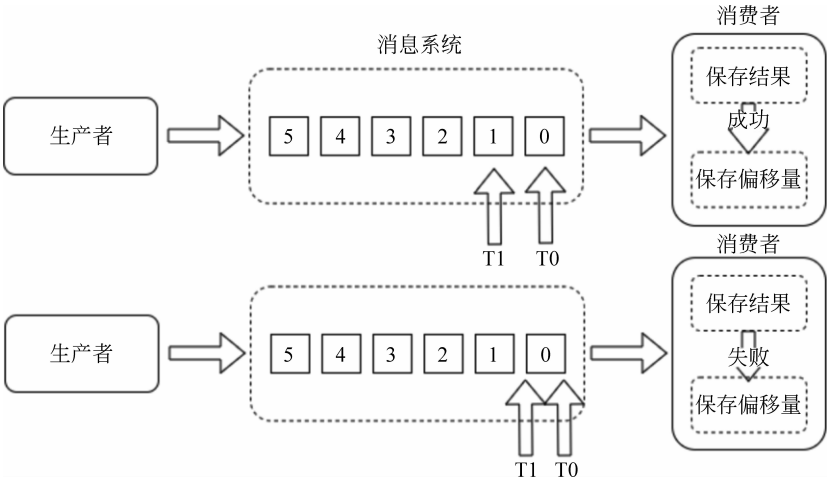

图 5-2　至少一次

5.1.2　最多一次

最多一次范例涉及一种机制，该机制在实际处理事件并将结果保留下来之前，先保存接收到的最后一个事件的位置，这样，如果发生故障并且消费者重新启动，消费者将不会尝试再次读取旧的事件。但是，由于不能保证已接收到的事件都已全部处理，而且它们再也不会被提取，因此可能导致事件丢失，所以导致事件最多处理一次或根本不处理的行为。

理想情况下，最多一次适用的任何应用程序涉及更新即时行情自动收录器，以显示当前值，以及任何累积的总和、计数器或其他汇总的应用程序，要求的条件是精度不是强制性的，或者应用程序不是绝对需要所有事件。任何丢失的事件都将导致错误的结果或结果丢失。消费者的操作顺序如下：①保存偏移量；②保存结果。图 5-3 显示了如果发生故障并且消费者重新启动后会发生的情况。由于存在尚未处理事件但保存了偏移量，因此消费者将从已保存的偏移量中读取数据，从而导致消耗的事件出现间隔。图 5-3 中出现了从未处理的事件 0。

5.1.3　恰好一次

恰好一次范例与至少一次使用范例相似，并且涉及一种机制，该机制仅在事件已被实际处理并且结果被持久化之后，保存最后接收到的事件位置，以便在发生故障时并且消费者重新启动后，消费者将再次读取旧事件并进行处理。但是，由于不能保证接收的事件被全部或部分处理，因此再次提取事件时可能导致事件重复。但是，与至少一次范例不同，重复事件不会被处理并被丢弃，从而导致恰好一次范例。恰好一次范例适用于涉及精确计数器、聚合或通常只需要每个事件仅处理一次且肯定要处理一次（无损失）的任何应用

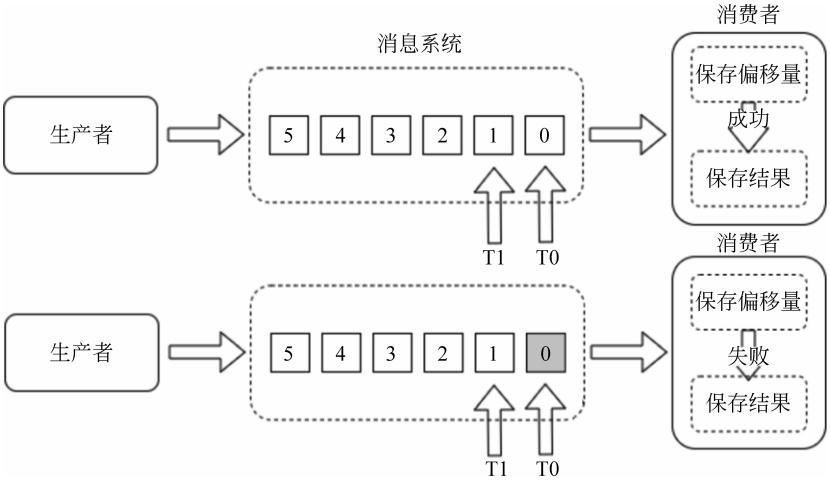

图 5-3　最多一次

程序。消费者的操作顺序如下：①保存结果；②保存偏移量。图 5-4 显示了如果发生故障并且消费者重新启动会发生的情况。由于事件已被处理，但偏移量尚未保存，因此消费者将从先前保存的偏移量中读取数据，从而导致重复。在图 5-4 中事件 0 仅处理一次，因为消费者删除了重复的事件 0。

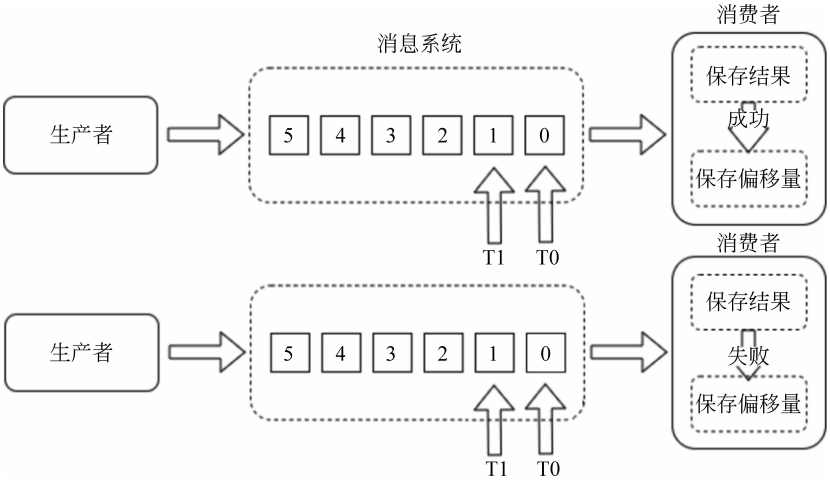

图 5-4　恰好一次

恰好一次范例如何删除重复项？这里有两种技术可以起作用：幂等更新和事务更新。幂等更新涉及基于生成的某些唯一 ID 保存结果，如果存在重复，则生成的唯一 ID 已经存在于结果中（例如数据库），消费者可以删除副本，而无须更新结果。因为并非总是可能而且方便地生成唯一 ID，而且这还需要在消费者上进行额外处理，所以这个过程很复杂。另一点是数据库可以针对结果和偏移量进行分离。事务更新将结果保存在具有事务开始和事务提交阶段的批处理中，以便在发生提交时知道事件已成功处理，因此，当收

到重复事件时，可以删除它们而不更新结果。这种技术比幂等更新复杂得多，因为现在我们需要一些事务性数据存储。另一点是数据库针对结果和偏移量必须一致。

Spark Streaming 在 Spark 2.x 中实现了结构化流传输并且支持恰好一次范例，本章后面将会介绍结构化流。

5.2 理解时间

我们可能遇到两种形式的数据：一种是静止的，以文件的形式、数据库的内容或者各种记录的形式；另一种是运动的，作为连续生成的信号序列，如传感器的测量或来自移动车辆的 GPS 信号。

我们已经讨论过，流式处理程序是一个假定其输入数据大小可能无限的程序，更具体地说，流式处理程序假定其输入数据是随时间推移观察到的不确定长度的信号序列。从时间轴的角度看，静止数据是过去的数据，可以说是有界数据集。无论是存储在文件中，还是包含在数据库中，最初都是随着时间推移收集到某个存储中的数据流，例如用户数据库中的上一季度的所有订单、城市出租车行驶的 GPS 坐标等，都是从单个事件开始被收集到存储库中。

但是，更具挑战性的是处理运动的数据。在最初生成数据的时刻与数据被处理的时刻之间存在时间差，该时间增量可能很短，例如在同一数据中心内生成和处理的 Web 日志事件；该时间增量也可能更长，例如汽车通过隧道时的 GPS 数据，只有当车辆离开隧道后重新建立无线连接时，GPS 数据才会被调度。可以看到，其中包含一个事件发生的时间轴，还包含另一个事件经过流式处理系统的时间轴。这些时间表非常重要，我们为它们指定了特定的名称。

■ 事件时间

创建事件的时间，时间信息由生成事件设备的本地时钟提供。

■ 处理时间

流式系统处理事件的时间，这是服务器运行处理逻辑的时钟，通常与技术原因相关，例如计算处理延迟，或作为标准确定重复输出。

当需要相互关联、排序或聚合事件时，这些时间线之间的区别变得非常重要。数据流中的元素始终具有处理时间，因为流处理系统观察到来自数据源新事件然后进行处理，处理运行时可以记录一个时间，这个时间完全独立于流元素的内容。但是，对于大多数数据流，我们另外提到事件时间的概念，即数据流事件实际发生的时间。如果流处理系统具有检测和记录事件的能力，通常将此事件时间作为流中消息有效负载的一部分。在事件中定义时间戳就是在消息生成时添加一个时间寄存器，该时间将成为数据流的一部分，例如某些不起眼的嵌入式设备（一般都有时钟系统）以及金融交易系统中的日志中都存在定义时间戳的做法，都可以作为事件时间。

时间戳的重要性在于，可以考虑使用数据生成的时间分析，例如跑步时使用可穿戴设备，回到家时将设备中的数据同步到手机，查看刚才穿过公园时的心率和速度等详细信息，在将数据上传到某些云服务器时，这些数据是具有时间戳的。时间戳为数据提供了时

间的上下文，根据事件发生时记录的时间戳进行分析才更有意义。因此，基于时间戳的日志构成了当今正在分析数据流的很大一部分，因此这些时间戳有助于弄清楚特定时间在给定系统上发生了什么。当将数据从创建数据的各种系统或设备传输到处理该数据的集群，通常会出现令人难以捉摸的情况，这是因为跨系统之间的传输操作容易发生不同形式的故障，如延迟、重新排序或丢失。通常，用户希望框架具有容错机制为这种可能发生的故障提供技术解决，而且不牺牲系统的响应能力。为了实现这种愿景，基于事件时间的流处理系统需要解决两个原则问题：其一是可以清楚标记正确和重新排序的结果；其二是可以产生中间预期结果。这两个原则构成事件时间处理的基础。在 Spark 中，此功能仅由结构化流提供，离散化流缺乏对事件时间处理的内置支持。

5.3　离 散 化 流

Spark Streaming 是 Spark 核心的扩展组件之一，可扩展地实现实时数据流的高吞吐量、容错处理。数据可以从诸如 Kafka、Flume、Kinesis 或 TCP 套接字的许多来源中获取，并且可以使用由高级功能表达的复杂算法进行处理。处理后的数据可以推送到文件系统、数据库和实时仪表板，也可以将 Spark 的机器学习和图处理算法应用于数据流。

Spark Streaming 总体框架如图 5-5 所示。首先是要处理的数据必须来自某个外部动态数据源，如传感器、移动应用程序、Web 客户端、服务器日志等，这个数据通过消息机制传送给数据采集系统，如 Kafka、Flume 等，递送或沉积在文件系统中。

图 5-5　Spark Streaming 总体框架

然后是流处理过程，获得的数据由 Spark Streaming 系统进行处理，接下来是基于 NoSQL 的数据存储，如 HBase 等用于存储处理的数据，该系统必须能够实现低延迟的、快速的读写操作，最后是通过终端应用程序显示或分析。终端应用程序可以包括仪表板、商业智能工具和其他使用已处理的流数据进行分析的应用程序，输出的数据也可以存储在数据库中，以便稍后进一步处理。

Spark Streaming 的工作原理如图 5-6 所示。Spark 数据流接收实时输入数据流并将数据分成批，然后由 Spark 引擎进行处理，以批量生成最终的结果流。

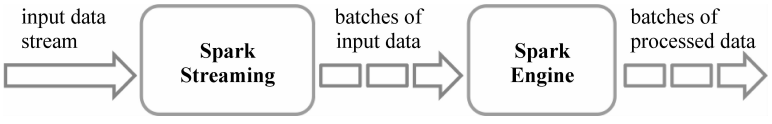

图 5-6　Spark Streaming 的工作原理

Spark Streaming 提供称为离散化数据流(Discretized Stream,DStream)的高级抽象,可以简称离散流,它代表连续产生的数据流。可以从诸如 Kafka、Flume 和 Kinesis 等来源的输入数据流中创建离散流,或者通过对其他离散流应用高级操作创建。在内部,离散流可以表示为一个批次接着一个批次以 RDD 为底层结构的数据流。

数据流本身是连续的,为了处理数据流,需要批量化。Spark Streaming 将数据流分割成 x 毫秒的批次,这些批次总称离散流。离散流是这种批次的一组序列,其中序列中的每个小批量表示为 RDD,数据流被分解成时间间隔相同的 RDD 段。按照 Spark 批处理的间隔,在离散流中的每个 RDD 包含了由 Spark Streaming 应用程序接收的记录。

有两种类型的离散流操作:转换和输出。在 Spark 应用程序中,在离散流上应用转换操作,例如 map()、reduce()和 join()等,处理其中的每个 RDD,在这个过程中创造新的 RDD,施加在离散流上的任何转换会应用到上一级离散流,然后依次施加转换到每个 RDD 上。输出是类似 RDD 操作的动作,因为它们将数据写到外部系统。在 Spark 数据流中,它们在每个时间步长周期性运行,批量生成输出。

5.3.1 一个例子

在详细介绍 Spark 数据流程序之前,先看一个简单的 Spark 数据流程序,这个程序通过 Spark Streaming 的 TCP 套接字接口侦听 NetCat 发生的数据,统计接收到的文本数据中的字数,这段代码的主程序为

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}

object NetworkWordCount {
  def main(args: Array[String]) {
    if (args.length <2) {
      System.err.println("Usage: NetworkWordCount <hostname><port>")
      System.exit(1)
    }
    val sparkConf=new SparkConf().setAppName("NetworkWordCount")
.setMaster("local[2]")
    val ssc=new StreamingContext(sparkConf, Seconds(10))
    val lines=ssc.socketTextStream(args(0), args(1).toInt)
    val words=lines.flatMap(_.split(" "))
    val wordCounts=words.map(x =>(x, 1)).reduceByKey(_ +_)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

代码 5-1

这段代码是一个简单的 Spark 应用程序，首先导入与 Spark 数据流相关的类，主要是 SparkConf 和 StreamingContext。SparkConf 用来设置启动 Spark 应用程序的参数，创建的应用的名称为 NetworkWordCount，并带有两个执行线程（local[2]）的本地 StreamingContext，批处理时间间隔为 10s。StreamingContext 是所有完成 Spark Streaming 功能的主要入口点，使用 ssc.socketTextStream 可以创建一个离散流，代表一个来自 TCP 套接字源的流数据，通过参数传入，args(0)指定为主机名（如 localhost）和 args(1)指定为端口（如 9999）。lines 为离散流对象，表示将从 NetCat 数据服务器接收的数据流，此离散流中的每条记录都是一行文本。接下来，_.split(" ")将包含空格字符的行分割成单词，flatMap()将包含多个单词的集合扁平化拆分成包含独立单词的离散流，通过从源离散流中的每条输入记录生成多个新记录创建新的输出离散流。在这种情况下，每一行将被分割成多个单词并且创建 words 离散流。接下来，通过在 words 离散流上应用聚合操作统计这些单词的数量。首先，通过 map()操作将 words 一对一转换成包含键值对（word，1）的离散流，然后通过 reduceByKey()获得每批数据中的单词统计离散流 wordCounts。最后，wordCounts.print()将打印每秒输入的单词计数。注意，当描述完这些操作过程后，这个单词计数的数据流应用程序仅定义了需要执行的计算过程，但是尚未开始实际处理。在所有转换操作设置完成后如果要开始处理，最终需要调用 ssc.start。

在虚拟实验环境中已经编译和打包了上面的应用程序，我们需要通过 spark-submit 启动这个应用程序包。首先需要运行 Netcat 作为数据服务器，使用 Docker exec 命令进入容器中打开一个终端界面。

```
root@48feaa001420:~# { while :; do echo "Hello Apache Spark"; sleep 0.05; done; }
| netcat -l -p 9999
```

代码 5-2

使用 Docker exec 命令进入容器中打开另一个终端界面，运行 Spark 应用程序。

```
root@48feaa001420:~# spark-submit --class NetworkWordCount /data/
application/simple-streaming/target/scala-2.11/simple-streaming_2.11-0.1.
jar localhost 9999
20/03/26 08:28:39 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
-------------------------------------------
Time: 1585211330000 ms
-------------------------------------------
(Hello,1028)
(Apache,1028)
(Spark,1028)

-------------------------------------------
Time: 1585211340000 ms
-------------------------------------------
```

```
(Hello,188)
(Apache,188)
(Spark,188)
```

代码 5-3

就这样，第一个终端窗口负责发送数据（见代码 5-2），第二个终端窗口负责接收处理数据（见代码 5-3）。

5.3.2 StreamingContext

StreamingContext 是流传输的主要入口点，本质上负责流传输应用程序，包括检查点、转换和对 RDD 的 DStreams 的操作。StreamingContext 是所有数据流功能切入点，提供了访问方法，可以创建来自各种输入源的离散流。StreamingContext 可以从现有 SparkContext 或 SparkConf 创建，其指定了 Master URL 和应用程序名称等其他配置信息。

➢ new StreamingContext(conf：SparkConf，batchDuration：Duration)

通过提供新的 SparkContext 所需的配置创建 StreamingContext。

➢ new StreamingContext(sparkContext：SparkContext，batchDuration：Duration)

使用现有的 SparkContext 创建一个 StreamingContext。

上面 StreamingContext 两个构造的第二个参数都是 batchDuration，这是数据流被分批的时间间隔。无论使用 Spark 交互界面或创建一个独立的应用程序，都需要创建一个新的 StreamingContext。要初始化 Spark 数据流程序，必须创建一个 StreamingContext 对象，它是所有 Spark 数据流功能的主要入口点。可以通过两种方式创建新的 StreamingContext。

（1）如果是在 Spark 应用程序中，StreamingContext 对象可以从 SparkConf 对象创建。

```
import org.apache.spark._
import org.apache.spark.streaming._

val conf=new SparkConf().setAppName(appName).setMaster(master)
val ssc=new StreamingContext(conf, Seconds(1))
```

代码 5-4

appName 参数是应用程序在集群监控界面上显示的名称。master 可以是 Spark、Mesos 或 YARN 集群 URL，或者以本地模式运行的特殊字符串 local［＊］。实际上，当在集群上运行时，不需要在应用程序中硬编码 master，而是使用 spark-submit 启动应用程序并设置 master 参数。但是，对于本地测试和单元测试，可以通过 local［＊］运行 Spark Streaming（检测本地系统中的核心数）。注意，这在内部创建一个 SparkContext（所有 Spark 功能的起始点），可以通过 ssc.sparkContext 进行访问。批处理间隔必须根据应用程序的延迟要求和可用的集群资源进行设置。

(2) 如果通过 spark-shell 打开交互界面，StreamingContext 对象也可以从现有的 SparkContext 对象创建。

```
scala>import org.apache.spark.streaming._
import org.apache.spark.streaming._

scala>val ssc=new StreamingContext(sc, Seconds(10))
ssc: org.apache.spark.streaming.StreamingContext =
org.apache.spark.streaming.StreamingContext@3c4231e5
```

代码 5-5

定义好 StreamingContext 后，必须执行以下操作：

(1) 通过创建输入离散流定义输入源。

(2) 通过将转换和输出操作应用于 DStream 定义流式计算。

(3) 使用 StreamingContext.start 开始接收数据。

(4) 使用 StreamingContext.awaitTermination 等待处理停止(手动或由于错误导致)。

(5) 可以使用 StreamingContext.stop 手动停止处理。

注意，一旦 StreamingContext 对象已经开始启动，就不能建立或添加新的数据流操作，只能按照定义好的操作运行；一旦当前的 StreamingContext 对象被停止，就无法重新启动这个 StreamingContext 对象；只有一个 StreamingContext 对象可以同时在 JVM 中处于活动状态；StreamingContext 对象上的 stop()方法也会停止 SparkContext 对象；如果仅停止 StreamingContext 对象，可以将 stop()方法的可选参数 stopSparkContext 设置为 false；只要先前的 StreamingContext 对象在创建下一个之前停止，而且不停止 SparkContext 对象，就可以使用这个 SparkContext 对象重复创建 StreamingContext 对象。

➢ stop(stopSparkContext：Boolean = ...)：Unit

这个方法立即停止 StreamingContext()的执行，不等待所有接收的数据被处理。默认情况下，如果没有指定 stopSparkContext 参数，SparkContext 对象将被停止，也可以使用 SparkConf 对象配置 spark.streaming.stopSparkContextByDefault 参数配置此隐式行为。

5.3.3　输入流

可以使用 StreamingContext 创建多种类型的输入流，例如 receiverStream 和 fileStream。在代码 5-1 中，lines 是一个输入离散流，通过 socketTextStream 从 NetCat 服务器接收数据流。每个输入流与接收器(Receiver)对象相关联，该对象接收数据并将其存储在内存中进行处理。

➢ abstract class Receiver[T] extends Serializable

这是接收外部数据的抽象类，接收器可以在 Spark 集群的工作节点上运行，可以通过定义方法 onStart()和 onStop()定义自定义接收器，onStart()定义开始接收数据所需的