亚太研究院

Spark 官方文档翻译

实时流处理编程指南
(v1.1.0)

翻译者　Jack Niu

Spark 官方文档翻译团成员

前 言

世界上第一个Spark 1.1.0 中文文档问世了！

伴随着大数据相关技术和产业的逐步成熟，继Hadoop之后，Spark技术以集大成的无可比拟的优势，发展迅速，将成为替代Hadoop的下一代云计算、大数据核心技术。

Spark是当今大数据领域最活跃最热门的高效大数据通用计算平台，基于RDD，Spark成功的构建起了一体化、多元化的大数据处理体系，在“One Stack to rule them all”思想的引领下，Spark成功的使用Spark SQL、Spark Streaming、MLLib、GraphX近乎完美的解决了大数据中Batch Processing、Streaming Processing、Ad-hoc Query等三大核心问题，更为美妙的是在Spark中Spark SQL、Spark Streaming、MLLib、GraphX四大子框架和库之间可以无缝的共享数据和操作，这是当今任何大数据平台都无可匹敌的优势。

在实际的生产环境中，世界上已经出现很多一千个以上节点的Spark集群，以eBay为例，eBay的Spark集群节点已经超过2000个，Yahoo 等公司也在大规模的使用Spark，国内的淘宝、腾讯、百度、网易、京东、华为、大众点评、优酷土豆等也在生产环境下深度使用Spark。2014 Spark Summit上的信息，Spark已经获得世界 20 家顶级公司的支持，这些公司中包括Intel、IBM等，同时更重要的是包括了最大的四个Hadoop发行商，都提供了对Spark非常强有力的支持。

与Spark火爆程度形成鲜明对比的是Spark人才的严重稀缺，这一情况在中国尤其严重，这种人才的稀缺，一方面是由于Spark技术在 2013、2014 年才在国内的一些大型企业里面被逐步应用，另一方面是由于匮乏Spark相关的中文资料和系统化的培训。为此，Spark亚太研究院和 51CTO联合推出了“Spark亚太研究院决胜大数据时代 100 期公益大讲堂”，来推动Spark技术在国内的普及及落地。

具体视频信息请参考 http://edu.51cto.com/course/course_id-1659.html

与此同时，为了向Spark学习者提供更为丰富的学习资料，Spark亚太研究院发起并号召，结合网络社区的力量构建了Spark中文文档专家翻译团队，历经 1 个月左右的艰苦努力和反复修改，Spark中文文档V1.1 终于完成。尤其值得一提的是，在此次中文文档的翻译期间，Spark官方团队发布了Spark 1.1.0 版本，为了让学习者了解到最新的内容，Spark中文文档专家翻译团队主动提出基于最新的Spark 1.1.0 版本，更新了所有已完成的翻译内容，在此，我谨代表Spark亚太研究院及广大Spark学习爱好者向专家翻译团队所有成员热情而专业的工作致以深刻的敬意！

当然，作为世界上第一份相对系统的Spark中文文档，不足之处在所难免，大家有任何建议或者意见都可以发邮件到marketing@sparkinchina.com ;同时如果您想加入Spark中文文档翻译团队，也请发邮件到marketing@sparkinchina.com进行申请；Spark中文文档的翻译是一个持续更新的、不断版本迭代的过程，我们会尽全力给大家

提供更高质量的Spark中文文档翻译。

最后，也是最重要的，请允许我荣幸的介绍一下我们的Spark中文文档第一个版本翻译的专家团队成员，他们分别是（排名不分先后）：

- 傅智勇，《快速开始(v1.1.0)》（和唐海东翻译的是同一主题，大家可以对比参考）
- 吴洪泽,《Spark机器学习库（v1.1.0)》（其中聚类和降维部分是蔡立宇翻译）
- 武扬，《在Yarn上运行Spark (v1.1.0)》《Spark 调优(v1.1.0)》
- 徐骄，《Spark配置(v1.1.0)》《Spark SQL编程指南(v1.1.0)》（Spark SQL和韩保礼翻译的是同一主题，大家可以对比参考）
- 蔡立宇，《Bagel 编程指南(v1.1.0)》
- harli，《Spark 编程指南（v1.1.0)》
- 吴卓华，《图计算编程指南(1.1.0)》
- 樊登贵，《EC2(v1.1.0)》《Mesos(v1.1.0)》
- 韩保礼，《Spark SQL编程指南(v1.1.0)》（和徐骄翻译的是同一主题，大家可以对比参考）
- 颜军，《文档首页(v1.1.0)》
- Jack Niu，《Spark实时流处理编程指南(v1.1.0)》
- 俞杭军，《sbt-assembly》《使用Maven编译Spark(v1.1.0)》
- 唐海东，《快速开始(v1.1.0)》（和傅智勇翻译的是同一主题，大家可以对比参考）
- 刘亚卿，《硬件配置(v1.1.0)》《Hadoop 第三方发行版(v1.1.0)》《给Spark提交代码(v1.1.0)》
- 耿元振《集群模式概览(v1.1.0)》《监控与相关工具(v1.1.0)》《提交应用程序(v1.1.0)》
- 王庆刚，《Spark作业调度(v1.1.0)》《Spark安全(v1.1.0)》
- 徐敬丽，《Spark Standalone 模式 (v1.1.0)》

另外关于Spark API的翻译正在进行中，敬请大家关注。

Life is short, You need Spark!

Spark亚太研究院院长 王家林
2014 年 10 月

Spark 亚太研究院决胜大数据时代 100 期公益大讲堂

简 介

作为下一代云计算的核心技术，Spark性能超Hadoop百倍，算法实现仅有其 1/10 或 1/100,是可以革命Hadoop的目前唯一替代者，能够做Hadoop做的一切事情，同时速度比Hadoop快了 100 倍以上。目前Spark已经构建了自己的整个大数据处理生态系统，国外一些大型互联网公司已经部署了Spark。甚至连Hadoop的早期主要贡献者Yahoo现在也在多个项目中部署使用Spark；国内的淘宝、优酷土豆、网易、Baidu、腾讯、皮皮网等已经使用Spark技术用于自己的商业生产系统中，国内外的应用开始越来越广泛。Spark正在逐渐走向成熟，并在这个领域扮演更加重要的角色，刚刚结束的 2014 Spark Summit上的信息，Spark已经获得世界 20 家顶级公司的支持，这些公司中包括Intel、IBM等，同时更重要的是包括了最大的四个Hadoop发行商都提供了对非常强有力的支持Spark的支持.

鉴于Spark的巨大价值和潜力，同时由于国内极度缺乏Spark人才，Spark亚太研究院在完成了对Spark源码的彻底研究的同时，不断在实际环境中使用Spark的各种特性的基础之上，推出了Spark亚太研究院决胜大数据时代 100 期公益大讲堂，希望能够帮助大家了解Spark的技术。同时，对Spark人才培养有近一步需求的企业和个人，我们将以公开课和企业内训的方式，来帮助大家进行Spark技能的提升。同样，我们也为企业提供一体化的顾问式服务及Spark一站式项目解决方案和实施方案。

Spark亚太研究院决胜大数据时代 100 期公益大讲堂是国内第一个Spark课程免费线上讲座，每周一期，从 7 月份起，每周四晚 20:00-21:30，与大家不见不散！老师将就Spark内核剖析、源码解读、性能优化及商业实战案例等精彩内容与大家分享，干货不容错过！

时间：从 7 月份起，每周一期，每周四晚 20:00-21:30
形式：腾讯课堂在线直播
学习条件：对云计算大数据感兴趣的技术人员
课程学习地址：http://edu.51cto.com/course/course_id-1659.html

实时流处理编程指南(v1.1.0)

（翻译者：Jack Niu）

Spark Streaming Programming Guide，原文档链接：

http://spark.apache.org/docs/latest/streaming-programming-guide.html

目录

第1章 Spark Streaming 编程指南

1.1 概述

Spark Streaming是Spark 核心API的一种扩展，它实现了对实时流数据的高吞吐量，低容错率的流处理。数据可以有许多来源，如Kafka, Flume, Twitter,ZeroMQ或传统TCP套接字,可以使用复杂算法对其处理实现高层次的功能，如map,reduce,join和window。最后，经处理的数据可被输出到文件系统，数据库，和实时仪表盘。事实上，你可以申请使用Spark公司在内置 机器学习算法，以及 图形处理的数据流算法。

它的内部工作原理如下: Spark Streaming 接收实时输入数据流并将数据划分为批次，其然后由 Spark Enigne 分批处理用来生成结果的最终流。

Spark Streaming提供了一个称为 *离散流* 或 *DStream*的高层次的抽象，它代表一个持续的流数据。DStreams可以从诸如Kafka 和 Flume的输入数据流中创建，或者通过应用高级操作在其他DStreams中创建。在内部，DSTREAM代表 RDDS中的一个序列。

本指南将告诉您如何开始用 DStreams 编写 Spark Streaming 程序。您可以使用 Scala 和 Java，这两者本指南中均会给出。你会发现选项卡贯穿本指南，从而可以让你在 Scala 和 Java 代码片段之间进行选择。

1.2 一个简单的例子

在我们进入如何编写自己的 Spark Streaming 程序的细节前，让我们快速浏览一下一个简单的 Spark Streaming 程序是什么样子。比方说，我们要计算从数据服务器监听 TCP 套接字接收的文本数据字数。你只需要做如下操作:

首先，我们导入 Spark Streaming 类名，以及添加需要（如 DStream）其他有用的方法以及一些从 StreamingContext 到我们环境的隐式转换.

StreamingContext是所有Spark Streaming功能的主入口点。

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

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

接下来，我们创建了一个 StreamingContext对象。除了Spark配置，我们还需指定会在 1 秒钟内进行批处理的任何的DStream。

```
import org.apache.spark.api.java.function._

import org.apache.spark.streaming._

import org.apache.spark.streaming.api._

// Create a StreamingContext with a local master

// Spark Streaming needs at least two working thread

val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
```

接下来 ,我们需要创建一个新的 DStream 对象,并通过 context 对象指定数据服务器的 IP 地址和端口。

```
// Create a DStream that will connect to serverIP:serverPort, like localhost:
9999

val lines = ssc.socketTextStream("localhost", 9999)
```

lines 对象表示将被从数据服务器接收的数据的流。在这个DStream中的每个记录就是一行文字。接下来，我们要根据空格把每一行分割成单词。

```
// Split each line into words

val words = lines.flatMap(_.split(" "))
```

flatMap是一对多的DStream操作.它从源DStream中每条记录中生成多个新记录到新创建的DStream中。在这种情况下，每一行将被分割为多个单词并且 words DStream表示单词流。接下来，我们将要计算这些单词的个数。

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

// Count each word in each batch

val pairs = words.map(word => (word, 1))

val wordCounts = pairs.reduceByKey(_ + _)


// Print a few of the counts to the console

wordCounts.print()
```

words DStream进一步映射（一对一的转换）到一个DStream（word，1）键值对，然后将其reduce得到在每批次的数据单词的频率。最后，wordCounts.print（）将打印数每秒产生的计数。

需要注意的是，当这些行被执行的时候，Spark Streaming 只设置了计算启动时它才会执行，而没开始有真正的处理。所有的转换完成后才开始处理，我们最终调用:

```
ssc.start()           // Start the computation

ssc.awaitTermination()  // Wait for the computation to terminate
```

可以在Spark Streaming示例中可以找到 NetworkWordCount的完整的代码。

如果您已经 下载并 编译 Spark，就可以运行这个如下示例。您首先需要使用如下代码运行Netcat（在大多数类Unix系统中一个小工具）作为数据服务器

```
$ nc -lk 9999
```

接下来,你可以在不同的终端通过运行如下代码启动实例

```
$ ./bin/run-example org.apache.spark.examples.streaming.NetworkWordCount loc
alhost 9999
```

接下来，作为终端 netcat 服务器每秒钟会计算并打印出所有的行。如下所示:

```
# TERMINAL 1:

# Running Netcat

$ nc -lk 9999

hello world

...
```

```
# TERMINAL 2: RUNNING NetworkWordCount or JavaNetworkWordCount


$ ./bin/run-example org.apache.spark.examples.streaming.NetworkWordCount localhost 9999

...

-------------------------------------------

Time: 1357008430000 ms

-------------------------------------------

(hello,1)

(world,1)

...
```

你也可以直接使用 Spark Shell 来调用 Spark Streaming

```
$ bin/spark-shell
```

也通过包装现有的交互式 shell SparkContext 对象创建 StreamingContext，SC：

```
val ssc = new StreamingContext(sc, Seconds(1))
```

当使用 Spark Shell 时，你可能还需要发送 a ^D 到 netcat 会话，强制使用 pipeline 打印字计数到控制台。

1.3 基础知识

接下来，我们跳过简单的例子，详细阐述如何根据 Spark Streaming 编写 streaming 应用的基础知识。

1.3.1 链接

您需要将以下依赖添加到您的 SBT 或 Maven 项目以便开始编写自己的 Spark Streaming 程序：

```
groupId = org.apache.spark

artifactId = spark-streaming_2.10

version = 1.0.2
```

对于来源于 Spark Streaming 的核心 API 中不存在的数据,如 Kafka 和 Flume,则必须到相应的 artifact 添加到 spark-streaming-xyz_2.10 依赖。一些常见的示例是如下:

Source	Artifact
Kafka	spark-streaming-kafka_2.10
Flume	spark-streaming-flume_2.10
Twitter	spark-streaming-twitter_2.10
ZeroMQ	spark-streaming-zeromq_2.10
MQTT	spark-streaming-mqtt_2.10

最新列表，请参阅 Apache的版本库 的支持来源的完整列表。

13.2 初始化 StreamingContext

使用 Scala 初始化 Spark Streaming 程序，需要创建 StreamingContext 对象,它是所有 Spark Streaming 功能的主要切入点。

可以通过使用如下创建 StreamingContext对象:

```
new StreamingContext(master, appName, batchDuration, [sparkHome], [jars])
```

master 参数是一个标准的 Spark cluster URL ，可以用 “local作为本地测试。该 appName 的 是你的程序名，它将显示在群集的Web UI。如前面解释的,batchInterval是批次的大小.最后，如果运行在分布式模式下的代码部署到集群，需要在最后两个参数，如在描述 星火编程指南。此外，底层SparkContext可以通过 ssc.sparkContext访问。

该批次间隔必须根据你的应用程序的延迟要求和可用群集资源进行设置。更多细节部分请参考 性能优化。

1.3.3 离散流(DStreams)

*离散流*或*DStream* 是Spark Streaming提供的基本抽象。它代表了一个连续的数据流，或者从源端接收到的，或通过将输入流中产生处理后的数据流。在内部，它是由RDDS的连续序列表示，这是Spark的一个不可改变的，分布式的数据集抽象。在DStream中每个RDD包含数据从某一个时间间隔，如图所示。

任何在DStream上使用的操作都会转化为底层RDDS操作。例如，在 前面的示例从行stream中转换为单词，在flatmap操作是在每个RDD施加在lines DStream生成的该RDDS上的 words DStream。如下图所示。

这些底层的RDD转换是由Spark engine计算的到。为使用方便,该DStream操作隐藏了大部分的细节，并提供给开发者更高层的API。这些操作细节在后面的章节中讨论。

1.3.4 输入离散流

我们已经看了 简单的例子中使用 ssc.socketTextStream（...）创建一个接收到TCP套接字连接文本数据的DSTREAM。除了套接字，核心Spark Streaming API提供了从文件和Akka actors 创建DStreams作为输入源的方法。

具体地，对于文件的 DSTREAM 可被创建为

```
ssc.fileStream(dataDirectory)
```

Spark Streaming 将监视 Hadoop 兼容的文件系统的任何 dataDirectory 目录并处理在该目录中创建的任何文件。注意

- 该文件必须具有相同的数据格式。
- 该文件必须在 dataDirectory中 创建,它可以由原子 *移动* 或 *重命名* 到数据目录中。
- 一旦移动这些文件不能被修改。

有关数据流从文件，Akka actors和安全套接字的详细信息，请参考在相关功能的API,其中 StreamingContext为Scala和 JavaStreamingContext 为Java。

从来源如Kafka, Flume和Twitter中创建DStreams附加功能可以通过导入正确的依赖关系,已在 早期章节中讲述。以Kafka举例，添加 spark-streaming-kafka_2.10 artifact 到项目的依赖后，你就可以创建一个从Kafka的DStream

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

KafkaUtils.createStream(ssc, kafkaParams, ...)
```

有关这些其他来源的详细信息，请参见相应的 API文档。此外，你也可以实现自己的自定义接收器的来源。请参阅 自定义接收器指南。

操作

有两种 DStream 操作- *转换*和*输出操作*。类似 RDD 转换，DStream 转换操作是在一个或多个 DStreams 上创建变换后的数据的新的 DStreams。在输入数据流序列转换后，调用输出操作,将数据写入到外部数据接收器，诸如一个文件系统或数据库中。

1.3.5 离散流转换

DStreams 支持多种变换的基本 Spark RDD 的使用。常见如下。

Transformation	Meaning
map(*func*)	源 DSTREAM 的每个元素通过函数 func 返回一个新的 DSTREAM。

flatMap(*func*)	类似的图 ,但每个输入项可以被映射到 0 或者更多的输出项。
filter(*func*)	在源 DSTREAM 上选择 FUNC 函数返回仅为 true 的记录,最终返回一个新的 DSTREAM 。
repartition(*numPartitions*)	通过创建更多或更少的分区改变平行于这个 DSTREAM 的层次。
union(*otherStream*)	返回一个包含源 DStream 与其他 DStream 元素联合后新的 DSTREAM。
count()	在源 DSTREAM 的每 RDD 的数目计数并返回包含单元素 RDDS 新的 DSTREAM。
reduce(*func*)	使用函数 func (有两个参数并返回一个结果)将源 DStream 中的每个 RDD 进行元素聚合,返回一个单元素 RDDs 新的 DStream.该函数应该关联，使得它可以并行地进行计算。
countByValue()	当请求 DStream 类型为 K 元素的 DStream,返回类型为(K , Long)的键值对的新 DSTREAM,其中每个键的值就是它的源 DSTREAM 每个 RDD 频率。

reduceByKey(*func*, [*numTasks*])	当一个类型为（K，V）键值对的 DStream 被调用的时候,返回类型为类型为（K，V）键值对的新 DStream,其中每个键的值都是使用给定的 reduce 函数汇总。注意：默认情况下，使用 Spark 的并行任务默认号码（2 为本地模式，并且在集群模式的数目是由配置属性确定 spark.default.parallelism）进行分组。你可以通过一个可选 numTasks 参数设置不同数量的任务。
join(*otherStream*, [*numTasks*])	当被调用类型分别为的（K，V）和（K，W）键值对的 2 个 DStream 时，返回包含所有键值对每个键的元素,类型为（K，（V，W））键值对的一个新 DSTREAM。
cogroup(*otherStream*, [*numTasks*])	当被调用的 DStream 含有(K, V) 和(K, W)键值对时,返回(K, Seq[V], Seq[W])元组的新 DStream.
transform(*func*)	通过源 DSTREAM 的每 RDD 应用 RDD-to-RDD 函数返回一个新的 DSTREAM。这可以用来在 DStream 做任意 RDD 操作。
updateStateByKey(*func*)	返回一个新 “状态”的 DStream,所在每个键的状态是根据键的前一个状态和键的新值应用给定函数后的更新。这可以

被用来维持每个键的任何状态数据。

最后两个转换是值得再次高亮显示。

UpdateStateByKey 操作

该 updateStateByKey 操作可以让你保持任意状态，同时不断有新的信息进行更新。要使用此功能，你就必须做两个步:

1. 定义状态 - 状态可以是任意的数据类型。
2. 定义状态更新函数 - 用一个函数指定如何使用先前的状态，从输入流中的新值更新状态。

让我们用一个例子来说明。假设你要维护可见的文本数据流中每个字的运行计数。在这里，正在运行的计数是状态而且它是一个整数。我们定义了更新功能:

```
def updateFunction(newValues: Seq[Int], runningCount: Option[Int]): Option[Int] = {

    val newCount = ...  // add the new values with the previous running count to get the new count

    Some(newCount)

}
```

此函数用于含有字的DStream（也就是说 前面的示例中，在DSTREAM含（word，1）键值对）。

```
val runningCounts = pairs.updateStateByKey[Int](updateFunction _)
```

每一个字都将调用更新函数，newValues 具有序列 1（从（word，1）键值 对）且 runningCount表示以前的计数。有关完整的Scala代码，看看这个例子 StatefulNetworkWordCount。

转换操作

该 转换 操作（连同其变化如 transformWith）允许DStream上应用任意RDD到RDD函数。它可以被应用于未在DStream API中暴露任何RDD操作。例如，在每批次的数据流与另一数据集的连接功能不直接暴露在DSTREAM API中。但是，您可以轻松地使用 转换 来

做到这一点。这使得功能非常强大。例如，如果你想通过连接预先计算的垃圾邮件信息的输入数据流（可能也有Spark生成的），然后基于此做实时数据清理的筛选。

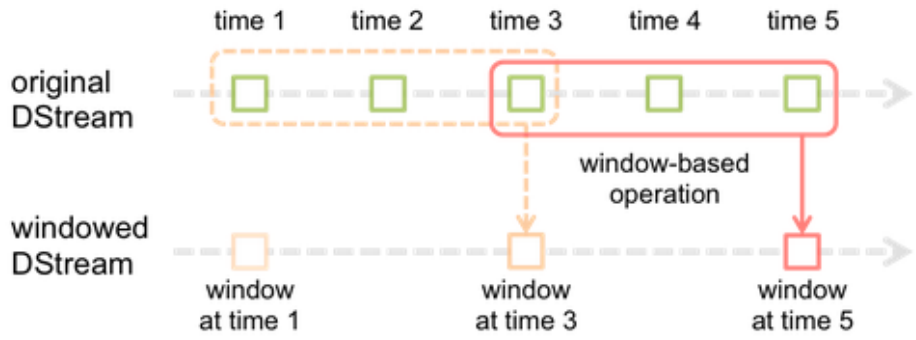
```
val spamInfoRDD = ssc.sparkContext.newAPIHadoopRDD(...) // RDD containing spam information


val cleanedDStream = wordCounts.transform(rdd => {
  rdd.join(spamInfoRDD).filter(...) // join data stream with spam information to do data cleaning
  ...
})
```

事实上，你也可以在转换方法中使用 机器学习和 图形计算的算法。

.

窗口操作

最后，Spark Streaming还提供了 *窗口的计算* ，它允许你通过滑动窗口对数据进行转换。下图阐述了这种滑动窗口。



如该图所示，每次当窗口 *滑动* 过源DStream，落入窗口内源RDDS被组合和操作时，产生的加窗DStream的RDDS。在此特定情况下，该操作是由 2 个滑动时间单元施加在数据的最后 3 个时间单位。这表明任何窗口操作需要指定两个参数。

- *窗口长度* -该窗口（图中 3）的持续时间
- *滑动间隔* - （图 2），在其上执行的窗口操作的时间间隔。

这两个参数必须是源 DSTREAM（1 中所示）的批次间隔的倍数。

让我们来说明窗口的操作用一个例子。比方说，你想扩展 前面的例子所产生过的数据，每 10 秒的最后 30 秒字数技术。要做到这一点，我们在数据的最后 30 秒的DStream的键值对(word ,1)上应 用 reduceByKey操作 。这是通过使用操作 reduceByKeyAndWindow。

```
// Reduce last 30 seconds of data, every 10 seconds

val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int) => (a + b),
 Seconds(30), Seconds(10))
```

一些常见的窗口的操作如下所述。所有这些操作用到所述两个参数- windowLength 和 slideInterval。

Transformation	Meaning
window(*windowLength*, *slideInterval*)	返回一个基于源 DStream 的窗口批次计算得到新的 DStream。
countByWindow(*windowLength*,*slideInterval*)	返回一个流中的元素的滑动窗口计数。
reduceByWindow(*func*, *windowLength*,*slideInterval*)	返回一个新的单件流 ,通过 FUNC 及在使用滑动间隔对数据流中的元素聚合。该函数应该联合以便它可以正确地在并行计算。
reduceByKeyAndWindow(*func*,*windowLength*, *slideInterval*, [*numTasks*])	当调用 DStream(K ,V)对时 ,返回新(K ,V)对的 DSTREAM ，其中每个键的值是根据给定的 reduce 函数 FUNC 和滑动窗口批次汇总。注意：默认情况下，这里使用 Spark 的并行任务默认号码（2 本地模式，并在集群模式下的数量由配置属性决定

	spark.default.parallelism ）做分组。你可以通过一个可选 numTasks 参数设置不同数量的任务。
reduceByKeyAndWindow(*func*, *invFunc*,*windowLength*, *slideInterval*, [*numTasks*])	一种比之前更有效的版本是 reduceByKeyAndWindow（ ），其中使用减少先前窗口的值的增量来计算每个窗口的减少值。这是通过减少输入的滑动窗口中的新数据完成的，而“逆减少”即离开窗口的旧数据。一个例子是“添加”和“减”键作为滑动窗口的数量。然而，它仅适用于“可逆的 reduce 函数”，即，那些 reduce 函数具有一个对应的“逆 reduce”函数（作为参数函数 invFunc 比如在 reduceByKeyAndWindow，减少任务的数量是通过一个可选的参数进行配置。
countByValueAndWindow(*windowLength*,*slideInterval*, [*numTasks*])	当被调用的（K，V）对的DStream，返回新的（K，Long）对的DSTREAM ,对其中每个键的值是其在一个滑动窗口频率。和 reduceByKeyAndWindow 一样 ，减少任务的数量是通过一个可选的参数进行配置。

API文档中提供DSTREAM转换的完整列表。对于Scala的API，参考 DSTREAM 和 PairDStreamFunctions。对于Java的API，参考 JavaDStream和 JavaPairDStream。

1.3.6 在 DStreams 输出操作

输出操作允许 DSTREAM 的数据被推送出外部系统，如数据库或文件系统。由于输出操作实际上使变换后的数据通过外部系统被使用，它们触发所有 DSTREAM 转换的实际执行（类似于 RDDS 操作）。目前，以下输出操作被定义为：

Output Operation	Meaning
print()	首先在 Dirver 上打印每一批 DStream 数据中的 10 个元素。这对于开发和调试。
saveAsObjectFiles(*prefix*, [*suffix*])	将 DSTREAM 的内容为序列化对象并保存为 SequenceFile 文件。根据产生在每个批次间隔的文件名前缀和后缀：“前缀 TIME_IN_MS [.后缀]”。
saveAsTextFiles(*prefix*, [*suffix*])	保存此 DSTREAM 的内容作为文本文件。根据产生在每个批次间隔的文件名前缀和后缀：“前缀 TIME_IN_MS [.后缀]”。
saveAsHadoopFiles(*prefix*, [*suffix*])	保存此 DSTREAM 的内容为 Hadoop 的文件。根据产生在每个批次间隔的文件名前缀和后缀：“前缀 TIME_IN_MS [.后缀]”。
foreachRDD(*func*)	Note that the function *func* is executed at the driver, and will usually have RDD actions in it that will force the computation of the streaming RDDs.

	最通用的输出操作，是使用函数 FUNC，从流中生成的每个 RDD。这个函数可以将每个 RDD 的数据推送到外部系统，如保存 RDD 为文件，或通过网络写到数据库。注意，函数 FUNC 是在 driver 执行 ,并且通常将具有在它的 RDD 行为，将强制执行 RDDS 的计算。

设计模式中使用 foreachRDD

dstream.foreachRDD是一个强大的原语，它允许数据发送到外部系统。但是，要了解如何正确，有效地使用这种原语是很重要的。为避免一些常见的错误现报告如下:

- 通常将数据写入到外部系统需要创建一个连接对象（如 TCP 连接到远程服务器），并用它来发送数据到远程系统。出于这个目的，开发者可能在不经意间尝试在 Spark driver 创建连接对象 ,并尝试使用它保存 RDDS 记录到 Spark worker。例如(在 Scala)

```
dstream.foreachRDD(rdd => {
    val connection = createNewConnection()  // executed at the driver
    rdd.foreach(record => {
        connection.send(record) // executed at the worker
    })
})
```

这是不正确的，这需要连接对象进行序列化并从 driver 发送到 worker。跨机器间很少转换这种连接对象。此错误可能表现为序列错误（连接对不可序列化），初始化错误（连接对象在 worker 中需要初始化），等等。正确的解决办法是在 worker 创建的连接对象。

- 但是，这可能会导致另一个常见的错误 - 为每一个记录的一个新的连接。例如，

```
dstream.foreachRDD(rdd => {
    rdd.foreach(record => {
        val connection = createNewConnection()
```

```
            connection.send(record)
            connection.close()
        })
    })
```

通常情况下，创建一个连接对象有时间和资源开销。因此，创建和销毁的每个记录中的连接对象可以招致不必要的高开销，并可以显著降低系统的整体吞吐量。一个更好的解决方案是使用 rdd.foreachPartition -创建一个单独的连接对象，并使用该连接发送的所有记录在RDD分区。

```
    dstream.foreachRDD(rdd => {
       rdd.foreachPartition(partitionOfRecords => {
          val connection = createNewConnection()
          partitionOfRecords.foreach(record => connection.send(record))
          connection.close()
       })
    })
```

这分摊了接创建多条记录连接的开销。

- 最后，这可以进一步通过重用在多个 RDDS / batches 的连接对象进行了优化。一种能够保持连接对象的静态池，可以重用作多个 batches 的 RDDS 推到外部系统，从而进一步降低了开销。

```
    dstream.foreachRDD(rdd => {
        rdd.foreachPartition(partitionOfRecords => {
            // ConnectionPool is a static, lazily initialized pool of connecti
ons
            val connection = ConnectionPool.getConnection()
            partitionOfRecords.foreach(record => connection.send(record))
            ConnectionPool.returnConnection(connection)  // return to the pool
 for future reuse
        })
    })
```

需要注意的是，在池中的连接应该按需延迟创建和超时(如果一段时间内不使用)。这实现了最有效的数据发送到外部系统。

另一点要记住：

- DStreams由输出操作延迟方式执行的，就像RDDS由RDD actions延迟方式执行。具体来讲，DSTREAM输出操作内部的RDD actions迫使所接收的数据的处理。因此，如果你的应用程序没有任何输出操作，或有在内部没有任何RDD action输出操作，如 dstream.foreachRDD（ ），那么什么都不会得到执行。系统将简单地接收的数据，并丢弃它。
- 默认情况下，输出操作一个在一次一执行。并且它们在它们的应用中定义的顺序执行。

1.3.7 缓存/持久化

类似RDDS，DStreams还允许开发者把流的数据持久化到内存中。也就是说，在DStream用 persist() 方法 将自动持久化DSTREAM中每一个RDD到内存中。如果在DSTREAM的数据将被计算多次（例如，相同的数据上做多个操作），这是很有用的。对于像基于窗口的操作 reduceByWindow和 reduceByKeyAndWindow和基于状态的操作例如 updateStateByKey，这是隐含为真的。因此，通过基于窗口的操作产生DStreams会自动持久化到内存中，无需开发人员调用 persist()。。

对于通过网络接收数据（例如，Kafka，Flume，安全套接字等）的输入数据流，默认持久化等级被设定为将数据复制到用于容错的两个节点。

需要注意的是，不像RDDS，DStreams的默认持久化等级将序列化数据到内存中。这是在进一步讨论 性能调优部分。在不同的持久化等级的详细信息，可以查找 Spark编程指南。

1.3.8 检查点

状态操作是数据的多个批次操作的其中之一。这包括所有基于窗口的操作和 updateStateByKey 操作。由于状态操作对数据的前一批次的依赖，随着时间的推移他们不断积累的元数据。要清除这些元数据，Spark streaming 通过中间数据保存到 HDFS 支持周期性的*检查点*。需要注意的是检查点也带来保存到 HDFS 的开销，这可能会导致相应的批处理需要更长的时间来处理。因此，需要检查点的时间间隔进行精心设置。在小批量生产（比如 1 秒），检查点每批次可显著减少操作的吞吐量。相反，检查点速度过慢造成的任务规模增长可能有不利影响。通常情况下，一个检查点的时间间隔-滑动 DSTREAM 间隔的 5-10 倍是好的设置试试。

启用检查点，开发人员须提供 RDD 将被保存的 HDFS 路径。通过使用:

```
ssc.checkpoint(hdfsPath) // assuming ssc is the StreamingContext or JavaStreamingContext
```

DStream 的检查点的间隔可以通过设置

```
dstream.checkpoint(checkpointInterval)
```

对于必须检查的 DStreams(即通过 updateStateByKe 和 reduceByKeyAndWindow 与逆函数创建的 DStreams ），该 DSTREAM 的检查点间隔为默认设置为 DSTREAM 的滑动区间的倍数，使得其至少 10 秒。

1.3.9 部署应用程序

Spark Streaming应用程序以同样的方式与任何其他Spark应用部署在集群上。请参阅部署指南了解更多详情。

请注意，使用 先进源（如Kafka, Flume, Twitter）的应用程序也需要包装他们链接以及同他们依赖的额外工件到JAR中,用来部署应用程序。例如，使用一个应用程序 TwitterUtils将必须包括 spark-streaming-twitter_2.10 和其所有传递依赖在应用程序 JAR中。

如果正在运行的 Spark Streaming 应用程序需要升级（使用新的应用程序代码），那么有两种可能的机制。

- 升级后的 Spark Streaming 应用程序启动和并行运行现有的应用程序。一旦新的（与旧的接收到相同的数据）已被预热并就绪，旧的程序就可以停用。请注意，这种方式适合于支持数据发送到两个目的地（即早期和升级的应用程序）的数据源来完成。
- 现有的应用程序是正常关闭（见 StreamingContext.stop（...） 或 JavaStreamingContext.stop（...） 为正常关机的选项），确保已接收关闭之前已经完全处理数据。然后在升级应用程序可以启动，这将从之前程序停止的同一点开始处理。注意，这只能由输入信号源完成,当之前的程序停止和升级的应用尚不起来的时候,它支持输入源侧缓冲（如Kafka和 Flume）.

1.3.10 监控应用

除了 Spark的 监控能力，也有具体到Spark Streaming的附加功能。当使用 StreamingContext时， Spark web UI 显示了一个额外的 Streaming 选项卡，显示有关运行数据的接收器（接收器是否处于活动状态，记录数收到，接收错误等），并已完成的批次（批次处理时间，排队时延等）等统计信息。这可以用于监测streaming应用程序的进度。

在Web UI中的以下两个指标就显得尤为重要- *处理时间*和 *调度延迟*（*批量处理统计下*）。第一个是要处理的每个批次数据的时间，第二个是在batch队列中等待先前批次的处理结束时间。如果批次处理时间比批次间隔持续多和/或排队延迟不断增加，则说明该系统是无法处理并落后于他们正在生成的批次。在这种情况下，可以考虑 减少批次处理时间。

Spark Streaming程序的进度，也可以通过监控 StreamingListener接口，它可以让你得到接收机的状态和处理时间。注意，这是一个开发的API，它很可能会在在未来（即，更多报告的信息）来加以改进。

1.4 性能优化

获取一个集群上的 a Spark Streaming 应用程序的最佳性能需要一些调整的。本节介绍了一些可以调整的参数和配置，以提高你的应用程序的性能。在较高的角度上，你需要考虑两件事情：

1. 通过有效地利用群集资源减少每批数据的处理时间。
2. 设置合适的批次大小，使得数据的批次可以尽可能快地接收并被处理（即，数据处理紧跟在数据摄取）
3.

1.4.1 减少每个批次的处理时间

在Spark中可以做很多优化从而减少每个批次的处理时间。这些都被详细讨论在 调优指南中。本节重点介绍一些最重要的。

并行结构的数据接收

通过网络接收（如 Kafka, Flume, socket 等）数据需要数据被反序列化并存储在 Spark 中。如果数据接收成为系统中的瓶颈，那么考虑并行数据接收。请注意，每个输入 DStream 创建单个接收器（运行在 worker 的机器上）用来接收单一的数据流。接收多个数据流可因此通过创建多个输入 DStreams 并配置它们以从源数据流中的不同分区接收数据来实现。例如，一个单一的 Kafka 输入 DStream 接收数据的两个主题可以被分成两个 Kafka 输入流，每个仅接收一个主题。这将在两个 worker 运行两个 receiver，因而允许并行接收数据，并提高整体的吞吐量。这些多 DSTREAM 可以联合在一起创建一个单一的 DSTREAM。然后这个单一输入 DSTREAM 转换为统一的流。可以做如下操作。

```
val numStreams = 5
```

```
val kafkaStreams = (1 to numStreams).map { i => KafkaUtils.createStream(...)
 }

val unifiedStream = streamingContext.union(kafkaStreams)

unifiedStream.print()
```

应当考虑的另一个参数是receiver的阻塞间隔。对于大多数的receiver ,在存储到Spark之前接收到的数据被合并在一起形成大块数据。在每批次的块的数量决定了将被用于处理在地图样变换的那些接收数据的任务数量。这种阻塞间隔由 的配置参数 spark.streaming.blockInterval确定,其默认值是 200 毫秒。

另一种从多个输入流/receiver接受数据是显示地对输入数据流重新分区（使用 inputStream.repartition（ <分区数> ） ）。在集群中进一步处理之前,它将数据分发接收批次到指定数目的机器上。

并行结构的数据处理

如果在计算的任何阶段使用的并行任务的数量不够高,则群集资源利用不足。例如，对于分布式reduce的操作如reduceByKey 和reduceByKeyAndWindow ,并行任务的默认数量由[配置属性] ((configuration.html#spark-properties) spark.default.parallelism) 决定。您可以通过并行的等级作为自变量（见[PairDStreamFunctions]（ api/scala/index.html#org.apache.spark.streaming.dstream.PairDStreamFunctions ）文档），或者设置 配置属性 spark.default.parallelism来更改默认值。

数据序列化

数据序列化的开销显著，特别是当要实现亚秒级的 batch 规模。有两个方面:

- Spark RDD数据序列化 :请参阅 调优指南中数据序列化的详细讨论。但是 ,请注意，不像Spark ,默认情况下RDDS是以序列化的字节数组持久化 ,以减少相关的GC暂停。
- 输入数据的序列化 ：将外部数据导入Spark，收到的字节数据（例如，从网络）需要从字节反序列化并按照Spark的序列化格式重新序列化。因此，输入数据的反序列化的开销可能是一个瓶颈。
-

任务启动管理开销

如果每秒发起的任务数目很高（例如，每秒 50 或更多），那么给从站发送的任务开销可能显著和将难以达到亚秒级延迟的。以下更改可以减少开销：

- 任务序列化 :使用Kryo序列化任务可降低任务的大小 ,因此降低其发送到从站的时间。

- 执行模式 :在独立模式或粗粒Mesos模式下运行Spark带来更好的任务启动时间比细粒度Mesos模式。更多细节请参考 Mesos运行指南。
  这些变化可能会缩短 100S 毫秒批处理时间，从而使亚秒级 batch 规模是可行的。

1.4.2 设定正确的批次大小

为了使一个集群上Spark Streaming应用程序稳定运行，系统处理与接收数据速度应该能够一致。换句话说，batch数据应该被处理与被产生时那样快。可以通过在Spark Streaming web用户界面找到 监视的处理时间,从而判断这对于应用程序是否正确设置，其中所述批量的处理时间应小于批次间隔。

根据流计算的性质 ,所用的间歇时间间隔可能对在一组固定的群集资源的应用程序上持续的数据传输速率产生显著影响。例如，让我们考虑之前的 WordCountNetwork 例子。对于一个特定的数据速率，系统能够跟上每 2 秒（即 2 秒间歇间隔）报告字计数，,而不是每 500 毫秒。因此需要设置这样的批次间隔，使得在生产中的预期数据速率可被维持。

为你的应用程序找出正确的batch大小一个好的方法是用一个保守的批次时间间隔（比如 5-10 秒）和低数据速率进行测试。为了验证该系统是否能够跟上数据传输速率，可以检查每一个处理batch所经历的端到端延迟的值（或是查找Spark driver log4j的log4j日志中“延迟总计”的值，或使用 StreamingListener 接口）。如果延迟保持与batch大小一致，则系统是稳定的。否则 ,如果延迟不断增加 ,这意味着该系统是无法跟上 ,因此是不稳定的。一旦你有一个稳定的配置的想法，你可以尝试提高数据速率和/或减少批量大小。请注意，由于临时数据率的增加导致延迟的瞬时增大也是可接受的 ,只要延迟会降低到较低的值(即，低于批量大小）。

1.4.3 内存优化

Spark应用的内存优化和GC的性能已经在 调优指南中很详细的讨论了。我们建议您阅读。在本节中，我们强调的是几个自定义,即强烈建议在Spark Streaming应用尽量减少GC相关的暂停并实现更加一致的批处理时间。

- DStreams的默认持久化等级 ：不像RDDS，DStreams的默认的持久化级别会序列化内存中的数据（也就是 DSTREAM为 StorageLevel.MEMORY_ONLY_SER而 RDDS为 StorageLevel.MEMORY_ONLY）。即使保持数据序列化会带来更高的序列化/反序列化的开销，但它可以显著减少GC暂停。
- 清除持久化的RDDS：默认情况下，按照Spark的内置策略（LRU）,Spark Streaming产生的所有持续化的RDDS会从内存中清除。如果设置了 spark.cleaner.ttl，那么持久化的RDDS时间超过该值时会定期清除。正如 早些时候提到的，这需要在Spark Streaming程序使用的操作的基础上小心设置。然而，通过设置 配置属

性 spark.streaming.unpersist为true来启用 更智能的非持久化 RDDS。这使得系统找出哪些RDDS是没有必要的维持并非持久化他们。这可能会减少Spark的RDD内存使用情况，也能潜在的提高GC的性能。

- 并发垃圾收集器 ：使用并发mark-and-sweep GC进一步减少GC暂停的可变性。即使我们知道并发GC会减小整个系统处理的吞吐量，但其仍然建议使用以实现更一致的批次处理时间。

1.5 容错特性

在本节中，我们将要讨论一个节点发生故障时的 Spark Streaming 应用程序的行为。要理解这一点，让我们记住 Spark 的 RDDS 基本容错特性。

1. RDD 是一个不可变的，唯一的可重新计算的，分布式的数据集。每个 RDD 记得在容错输入数据集中创建它的唯一操作的血统。
2. 如果 RDD 任何分区因 worker 节点故障而丢失，那么这个分区可以从原来的容错数据集使用操作的血统重新计算。

因为在 Spark Streaming 的所有数据转换是基于 RDD 操作，只要输入数据集存在，所有的中间数据可以重新计算。牢记这些特性，我们将详细讨论失败的语义。

1.5.1 Worker 节点的故障

基于输入源的使用有两种故障行为。

1. *使用 HDFS 文件作为输入源* -由于数据是可靠地存储在 HDFS 中，所有的数据可以重新计算，因此没有数据会丢失导致任何故障。
2. *使用通过网络的接收的数据任何输入源* -对于像 Kafka 和 Flume 基于网络的数据源，所接收的输入数据被复制在内存中的簇节点之间（默认复制因子为 2）。因此，如果一个 worker 节点发生故障，则该系统可以拷贝丢失输入数据重新计算所述遗留问题。然而，如果 worker 节点的网络接收器正在运行时发生故障，则数据可能丢失一部分，也就是说，收到的系统中的数据，但还没有被复制到其他节点（次）。接收器将其他节点上启动从而继续接收数据。

由于所有数据都被建模为 RDDS 唯一操作的血统，任何重新计算总是会带来同样的结果。这样一来，所有的 DSTREAM 转换是保证有 恰好一次语义。也就是说，即使有是一个 worker 节点故障,最终的转化的结果将是一样的。然而，输出操作（如 foreachRDD）具有在-至少一次 的语义，即 worker 出现故障时转换后的数据可以被不止一次地写入到一个外

部实体。虽然保存到 HDFS 使用 的 saveAs *文件操作（如，文件将简单地重写用相同的数据）是可以接受的，为达到恰好一次语义输出操作,附加事务机制可能是必要的。

1.5.2 Driver 节点的故障

对于 streaming 应用程序全天候运行，Spark Streaming 允许 streaming 计算即使 driver 节点失败后也可进行恢复。Spark Streaming 通过 StreamingContext 定期写入 DStreams 设置的元数据信息到 HDFS 的目录(可以是任何的 Hadoop 兼容的文件系统)。如之前所述,这种周期性 *的检查点*可以通过启用 ssc.checkpoint（ <检查点目录> ）来设置检查点目录。driver 节点故障发生时，丢失 StreamingContext 可以从该信息中恢复，并重新启动。

为了让 Spark Streaming 程序能够恢复，必须写入的方式，使得它具有以下特性：

1. 当程序正在第一次启动时 ,它会创建一个新的 StreamingContext 建立所有的数据流，然后调用 start（ ）。
2. 当程序被破坏后重新启动时，它会在检查点目录的检查点数据中重新创建 StreamingContext。
3.

这种行为是通过使用 StreamingContext.getOrCreate 变得简单。使用方法如下。

```
// Function to create and setup a new StreamingContext

def functionToCreateContext(): StreamingContext = {

    val ssc = new StreamingContext(...)   // new context

    val lines = ssc.socketTextStream(...) // create DStreams

    ...

    ssc.checkpoint(checkpointDirectory)   // set checkpoint directory

    ssc

}


// Get StreamingContext from checkpoint data or create a new one

val context = StreamingContext.getOrCreate(checkpointDirectory, functionToCr
eateContext _)


// Do additional setup on context that needs to be done,

// irrespective of whether it is being started or restarted
```

```
context. ...


// Start the context

context.start()

context.awaitTermination()
```

如果 checkpointDirectory 存在，则上下文将从检查点数据重建。如果目录不存在（如第一次运行），则该函数 functionToCreateContext 将被调用来创建一个新的上下文，并建立 DStreams。看到 Scala 的例子 RecoverableNetworkWordCount。这个例子追加网络数据的计数到一个文件中。

您也可以从检查点的数据显式地创建 StreamingContext，并使用 new StreamingContext（checkpointDirectory）开始计算。

注：如果 Spark Streaming 和/或 Spark Streaming 程序被重新编译，你*必须*创建一个新的 StreamingContext 或 JavaStreamingContext，而不是从检查点数据重新创建。这是因为，如果类重新编译之前数据被生成,试图从检查点数据加载上下文可能会失败。所以，如果你使用的是 getOrCreate，则确保检查点目录被显示删除每当重新编译代码启动的时候。

这种故障恢复可以使用 Spark 的进行 独立的集群模式自动完成，它允许任何 Spark 应用程序的驱动程序在集群内启动，并在失败时重新启动（见 监督模式）。这可以在本地通过使用本地独立集群的监管模式启动上面的例子，并杀死运行 driver 的 Java 进程进行测试（当 JPS 运行，显示所有活动的 Java 进程时将显示为 *DriverWrapper*）。driver 应自动重新启动，并且计数将继续

对于像 Mesos 和 Yarn 等部署环境中，你必须通过其他机制来重新启动 driver。

Recovery Semantics

基于输入源的使用有两种故障行为。

1. *使用 HDFS 文件作为输入源* -由于数据是可靠地存储在 HDFS 中，所有的数据可以重新计算，因此没有数据会丢失导致任何故障。
2. *使用接收的数据通过网络的任何输入源* -接收到的输入数据在内存中被复制到多个节点。由于当 Spark driver 出现故障时所有在 Spark worker 的内存中的数据均丢失，过去的输入数据将无法访问和 driver 恢复。因此，如果使用有状态和基于窗口的操作（如

updateStateByKey，window，countByValueAndWindow 等），那么在中间状态将不会完全恢复。

在将来的版本中，我们将支持完全恢复所有输入源。请注意，对于像非状态转换如图，map, count, 和 reduceByKey，该系统中，一旦重新启动，*所有*输入流，将继续接收和处理新的数据。

为了更好地理解一个 HDFS 的源 driver 发生故障时系统的行为，让我们考虑一个文件输入流将发生什么。具体地，在该文件输入流的情况下，driver 宕机时创建的新文件可以被正确地识别所，并它们如同没有故障时以同样的方式被处理。为了进一步说明在文件输入流，我们将用一个例子。比方说，每秒生成文件，和 Spark Streaming 程序每次读取新的文件和输出文件中的行数。这是具有和不具 driver 故障时输出的序列。

Time	Number of lines in input file	Output without driver failure	Output with driver failure
1	10	10	10
2	20	20	20
3	30	30	30
4	40	40	[DRIVER FAILS] no output
5	50	50	no output
6	60	60	no output
7	70	70	[DRIVER RECOVERS] 40, 50, 60, 70
8	80	80	80

9	90	90	90
10	100	100	100

如果 driver 在时间 3 的处理的中间崩溃，那么将处理时间 3 和恢复后输出 30。

1.6 从 0.9.1 或以下到 1.x 的迁移指南

星火 0.9.1 和 Spark 1.0 之间，还有以确保未来 API 的稳定做了几个变化。本节详细阐述将现有代码迁移到 1.0 所需的步骤。

输入DStreams：

创建一个输入流(例如,StreamingContext.socketStream，FlumeUtils.createStream 等）的所有操作现在在Scala中返回 InputDStream / ReceiverInputDStream （而不是 DSTREAM），在Java中返回 JavaInputDStream / JavaPairInputDStream /JavaReceiverInputDStream / JavaPairReceiverInputDStream (而不是 JavaDStream)。这确保了输入数据流的功能特性在未来可被添加到这些类而不破坏二进制兼容性。请注意，您现有的Spark Streaming应用程序不应该要求任何变化（因为这些新类是DSTREAM / JavaDStream的子类），但可能需要重新编译以Spark 1.0。

定制网络接收器 ：

在之前的Spark Streaming，自定义网络接收器可以在Scala中使用类NetworkReceiver定义。然而，在API中的错误处理和报告方面是有限的，并且不能从Java调用。从Starting Spark 1.0，这个类已经被取代为 Receiver并有以下优点。

- 方法如 stop 和 restart 已添加到更好地控制接收器的生命周期。更多细节请参考自定义的接收器指导。
- 自定义接收器可以同时使用 Scala 和 Java 实现。
  从现有的早期 NetworkReceiver 迁移到新的自定义接收器，你需要做到以下几点。
- 使你的自定义接收器类继承 org.apache.spark.streaming.receiver.Receiver 而不是 org.apache.spark.streaming.dstream.NetworkReceiver。
- 此前，BlockGenerator 对象必须通过 custom receiver 创建，接收到的数据被存储在 Spark。它必须通过 OnStart( )和的 onStop( ) 方法显式启动和停止。新的 Receiver 类使得这种不必要的，因为它增加了一套名为方法 store(<data>) ，可以调用它来存储

数据。因此，迁移您的自定义网络接收器，删除任何 BlockGenerator 对象（反正星火 1.0 后不再存在），并使用 store(...) 方法存储接收到的数据。

基于Actor的接收器 ：数据可能已被使用任何Akka Actors通过继承自actor 类 org.apache.spark.streaming.receivers.Receiver。这已更名 为 org.apache.spark.streaming.receiver.ActorHelper 和 pushBlock（...）方法来存储接收到的数据已被重新命名为 store(...)。在其他辅助 类 org.apache.spark.streaming.receivers包也被搬到 了 org.apache.spark.streaming.receiver 包，并为更容易区分而更名。

1.7 从这里到哪

- API 文档
  - Scala 的文档
    - StreamingContext和 DSTREAM
    - KafkaUtils， FlumeUtils， KinesisUtils， TwitterUtils， ZeroMQUtils 和 MQTTUtils
  - Java 的文档
    - JavaStreamingContext， JavaDStream和 PairJavaDStream
    - KafkaUtils， FlumeUtils， KinesisUtils TwitterUtils， ZeroMQUtils 和 MQTTUtils
- 在Scala和 Java的更多实例
- 文献和 视频描述 Spark Streaming。

Spark 亚太研究院

Spark 亚太研究院是中国最专业的一站式大数据 Spark 解决方案供应商和高品质大数据企业级完整培训与服务供应商，以帮助企业规划、架构、部署、开发、培训和使用 Spark 为核心，同时提供 Spark 源码研究和应用技术训练。针对具体 Spark 项目，提供完整而彻底的解决方案。包括 Spark 一站式项目解决方案、Spark 一站式项目实施方案及 Spark 一体化顾问服务。

官网： www.sparkinchina.com

近期活动

- 2014 年亚太地区规格最高的 Spark 技术盛会！
- 面向大数据、云计算开发者、技术爱好者的饕餮盛宴！
- 云集国内外 Spark 技术领军人物及灵魂人物！
- 技术交流、应用分享、源码研究、商业案例探询！

时间：2014 年 12 月 6-7 日
地点：北京珠三角万豪酒店
Spark亚太峰会网址：http://www.sparkinchina.com/meeting/2014yt/default.asp

- 如果你是对 Spark 有浓厚兴趣的初学者，在这里你会有绝佳的入门和实践机会！
- 如果你是 Spark 的应用高手，在这里以“武”会友，和技术大牛们尽情切磋！
- 如果你是对 Spark 有深入独特见解的专家，在这里可以尽情展现你的才华！

比赛时间：

2014 年 9 月 30 日—12 月 3 日

Spark开发者大赛网址：http://www.sparkinchina.com/meeting/2014yt/dhhd.asp

视频课程：

《大数据 Spark 实战高手之路》 国内第一个 Spark 视频系列课程

从零起步，分阶段无任何障碍逐步掌握大数据统一计算平台 Spark，从 Spark 框架编写和开发语言 Scala 开始，到 Spark 企业级开发，再到 Spark 框架源码解析、Spark 与 Hadoop 的融合、商业案例和企业面试，一次性彻底掌握 Spark，成为云计算大数据时代的幸运儿和弄潮儿，笑傲大数据职场和人生！

- 第一阶段：熟练的掌握 Scala 语言
  课程学习地址：http://edu.51cto.com/pack/view/id-124.html
- 第二阶段：精通 Spark 平台本身提供给开发者 API
  课程学习地址：http://edu.51cto.com/pack/view/id-146.html
- 第三阶段：精通 Spark 内核
  课程学习地址：http://edu.51cto.com/pack/view/id-148.html
- 第四阶段：掌握基于 Spark 上的核心框架的使用
  课程学习地址：http://edu.51cto.com/pack/view/id-149.html
- 第五阶段：商业级别大数据中心黄金组合：Hadoop+ Spark
  课程学习地址：http://edu.51cto.com/pack/view/id-150.html
- 第六阶段：Spark 源码完整解析和系统定制
  课程学习地址：http://edu.51cto.com/pack/view/id-151.html

近期公开课：

《决胜大数据时代：Hadoop、Yarn、Spark 企业级最佳实践》

集大数据领域最核心三大技术：Hadoop 方向 50%：掌握生产环境下、源码级别下的 Hadoop 经验，解决性能、集群难点问题；Yarn 方向 20%：掌握最佳的分布式集群资源管理框架，能够轻松使用 Yarn 管理 Hadoop、Spark 等；Spark 方向 30%：未来统一的大数据框架平台，剖析 Spark 架构、内核等核心技术，对未来转向 SPARK 技术，做好技术储备。课程内容落地性强，即解决当下问题，又有助于驾驭未来。

开课时间： 2014 年 10 月 26-28 日北京、2014 年 11 月 1-3 日深圳

咨询电话：4006-998-758

QQ 交流群： 1 群：317540673 （已满）
2 群：297931500

微信公众号：spark-china