# 第一章 有状态的流式处理简介

Apache Flink是一个分布式流处理器，具有直观和富有表现力的API，可实现有状态的流处理应用程序。它以容错的方式有效地大规模运行这些应用程序。 Flink于2014年4月加入Apache软件基金会作为孵化项目，并于2015年1月成为顶级项目。从一开始，Flink就拥有一个非常活跃且不断增长的用户和贡献者社区。到目前为止，已有超过五百人为Flink做出贡献，并且它已经发展成为最复杂的开源流处理引擎之一，并得到了广泛采用的证明。 Flink为不同行业和全球的许多公司和企业提供大规模的商业关键应用。

流处理技术在大大小小的公司中越来越受欢迎，因为它为许多已建立的用例（如数据分析，ETL和事务应用程序）提供了卓越的解决方案，同时也促进了新颖的应用程序，软件架构和商机。接下来我们将讨论，为什么有状态流处理变得如此受欢迎并评估其潜力。我们首先回顾传统的数据应用程序架构并指出它们的局限性。接下来，我们介绍基于状态流处理的应用程序设计 与传统方法相比，它具有许多有趣的特征最后，我们简要讨论开源流处理器的发展，并在本地Flink实例上运行流应用程序。

## 1.1 传统数据处理架构

数十年来，数据和数据处理在企业中无处不在。多年来，数据的收集和使用一直在增长，公司已经设计并构建了基础架构来管理数据。大多数企业实施的传统架构区分了两种类型的数据处理：事务处理（OLTP）和分析处理（OLAP）。

### 事务处理

公司将各种应用程序用于日常业务活动，例如企业资源规划（ERP）系统，客户关系管理（CRM）软件和基于Web的应用程序。这些系统通常设计有单独的层，用于数据处理（应用程序本身）和数据存储（事务数据库系统），如图1-1所示。

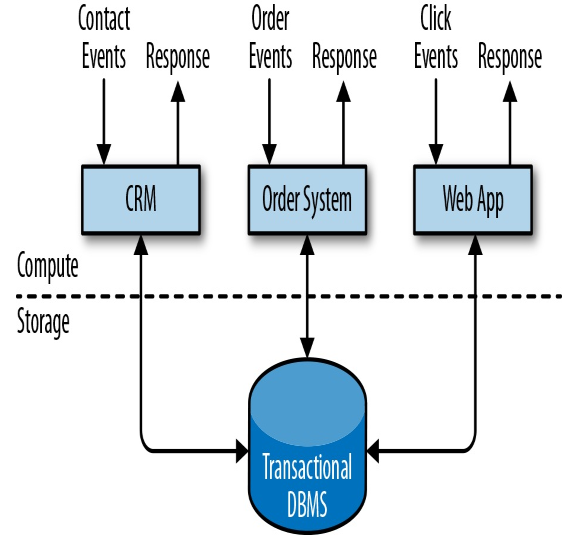


图 1-1 传统应用的架构设计（将数据存储在远程数据库）

应用程序通常连接到外部服务或直接面向用户，并持续处理传入的事件，如网站上的订单，电子邮件或点击。处理事件时，应用程序将会读取远程数据库的状态，或者通过运行事务来更新它。通常，一个数据库系统可以服务于多个应用程序，它们有时会访问相同的数据库或表。

当应用程序需要扩展时，这样的设计可能会导致问题。由于多个应用程序可能会同时用到相同的数据表示，或者共享相同的基础设施，因此想要更改表的结构或扩展数据库，就需要仔细的规划和大量的工作。克服紧耦合应用程序的最新方法是微服务设计模式。微服务被设计为小型、完备且独立的应用程序。他们遵循UNIX的理念，即“只做一件事并且把它做好”。通过将几个微服务相互连接来构建更复杂的应用程序，这些微服务仅通过标准化接口（例如RESTful HTTP连接）进行通信。由于微服务严格地彼此分离并且仅通过明确定义的接口进行通信，因此每个微服务都可以用不同技术栈来实现，包括编程语言、类库和数据存储。微服务和所有必需的软件和服务通常捆绑在一起并部署在独立的容器中。图1-2描绘了一种微服务架构。

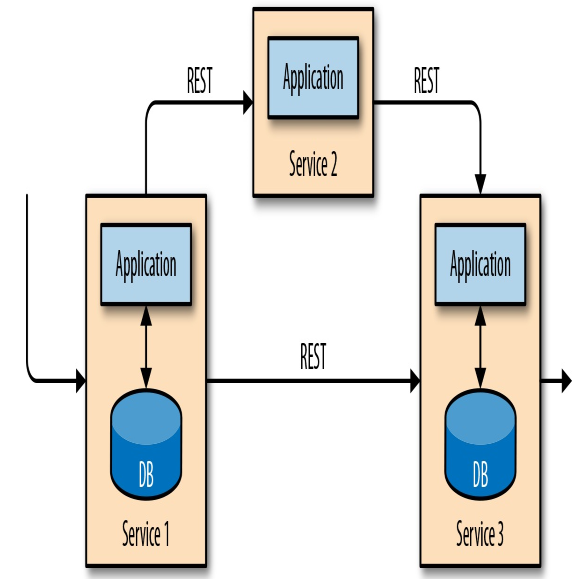


图1-2 微服务架构

### 分析处理

大量数据存储在公司的各种事务数据库系统中，它们可以为公司业务运营提供宝贵的参考意见。例如，分析订单处理系统的数据，可以获得销量随时间的增长曲线；可以识别延迟发货的原因；还可以预测未来的销量以便提前调整库存。但是，事务数据通常分布在多个数据库中，它们往往汇总起来联合分析时更有价值。而且，数据通常需要转换为通用格式。

所以我们一般不会直接在事务数据库上运行分析查询，而是复制数据到数据仓库。数据仓库是对工作负载进行分析和查询的专用数据存储。为了填充数据仓库，需要将事务数据库系统管理的数据复制过来。将数据复制到数据仓库的过程称为extract-transform-load（ETL）。 ETL过程从事务数据库中提取数据，将其转换为某种通用的结构表示，可能包括验证，值的规范化，编码，重复数据删除（去重）和模式转换，最后将其加载到分析数据库中。 ETL过程可能非常复杂，并且通常需要技术复杂的解决方案来满足性能要求。 ETL过程需要定期运行以保持数据仓库中的数据同步。

将数据导入数据仓库后，可以查询和分析数据。通常，在数据仓库上执行两类查询。第一种类型是定期报告查询，用于计算与业务相关的统计信息，比如收入、用户增长或者输出的产量。这些指标汇总到报告中，帮助管理层评估业务的整体健康状况。第二种类型是即席查询，旨在提供特定问题的答案并支持关键业务决策，例如收集统计在投放商业广告上的花费，和获取的相应收入，以评估营销活动的有效性。两种查询由批处理方式由数据仓库执行，如图1-3所示。

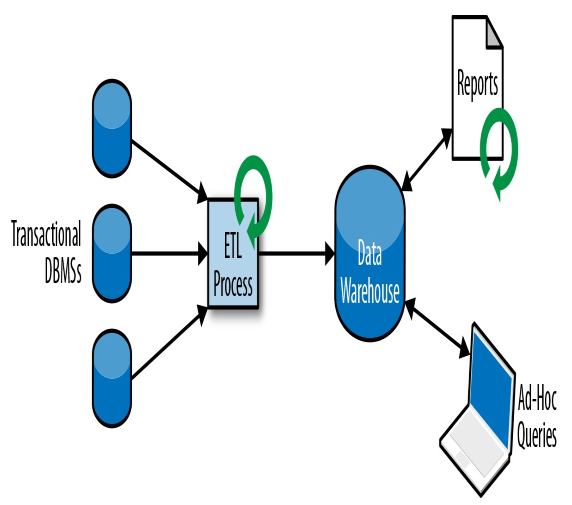


图1-3 用于数据分析的传统数仓架构

如今，Apache Hadoop生态系统的组件，已经是许多企业IT基础架构中不可或缺的组成部分。现在的做法不是直接将所有数据都插入关系数据库系统，而是将大量数据（如日志文件，社交媒体或Web点击日志）写入Hadoop的分布式文件系统（HDFS）、S3或其他批量数据存储库，如Apache HBase，以较低的成本提供大容量存储容量。驻留在此类存储系统中的数据可以通过SQL-on-Hadoop引擎查询和处理，例如Apache Hive，Apache Drill或Apache Impala。但是，基础结构与传统数据仓库架构基本相同。

## 1.2 有状态的流式处理

日常生活中，所有数据都是作为连续的事件流创建的。比如网站或者移动应用中的用户交互动作，订单的提交，服务器日志或传感器测量数据：所有这些都是事件流。实际上，很少有应用场景，能一次性地生成所需要的完整（有限）数据集。实际应用中更多的是无限事件流。**有状态的流处理**就是用于处理这种**无限事件流**的应用程序设计模式，在公司的IT基础设施中有广泛的应用场景。在我们讨论其用例之前，我们将简要介绍有状态流处理的工作原理。

如果我们想要无限处理事件流，并且不愿意繁琐地每收到一个事件就记录一次，那这样的应用程序就需要是有状态的，也就是说能够存储和访问中间数据。当应用程序收到一个新事件时，它可以从状态中读取数据，或者向该状态写入数据，总之可以执行任何计算。原则上讲，我们可以在各种不同的地方存储和访问状态，包括程序变量（内存）、本地文件，还有嵌入式或外部数据库。

Apache Flink将应用程序状态，存储在内存或者嵌入式数据库中。由于Flink是一个分布式系统，因此需要保护本地状态以防止在应用程序或计算机故障时数据丢失。 Flink通过定期将应用程序状态的一致性检查点（check point）写入远程且持久的存储，来保证这一点。状态、状态一致性和Flink的检查点将在后面的章节中更详细地讨论，但是，现在，图1-4显示了有状态的流式Flink应用程序。

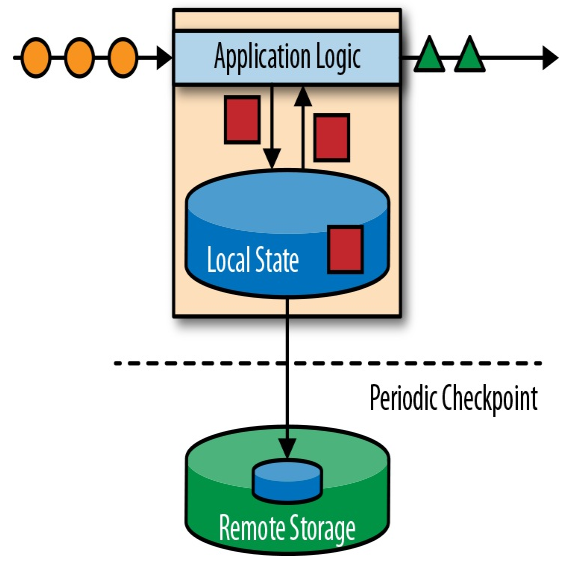


图1-4 有状态的流式应用程序

有状态的流处理应用程序，通常从事件日志中提取输入事件。事件日志就用来存储和分发事件流。事件被写入持久的仅添加（append-only）日志，这意味着无法更改写入事件的顺序。写入事件日志的流，可以被相同或不同的消费者多次读取。由于日志的仅附加（append-only）属性，事件始终以完全相同的顺序发布给所有消费者。现在已有几种事件日志系统，其中Apache Kafka是最受欢迎的，可以作为开源软件使用，或者是云计算提供商提供的集成服务。

在Flink上运行的有状态的流处理应用程序，是很有意思的一件事。在这个架构中，事件日志会按顺序保留输入事件，并且可以按确定的顺序重播它们。如果发生故障，Flink将从先前的检查点（check point）恢复其状态，并重置事件日志上的读取位置，这样就可以恢复整个应用。应用程序将重放（并快进）事件日志中的输入事件，直到它到达流的尾部。此技术一般用于从故障中恢复，但也可用于更新应用程序、修复bug或者修复以前发出的结果，另外还可以用于将应用程序迁移到其他群集，或使用不同的应用程序版本执行A / B测试。

如前所述，有状态的流处理是一种通用且灵活的设计架构，可用于许多不同的场景。在下文中，我们提出了三类通常使用有状态流处理实现的应用程序：（1）事件驱动应用程序，（2）数据管道应用程序，以及（3）数据分析应用程序。

我们将应用程序分类描述，是为了强调有状态流处理适用于多种业务场景；而实际的应用中，往往会具有以上多种情况的特征。

### 事件驱动应用程序（Event-Driven Applications）

事件驱动的应用程序是有状态的流应用程序，它们使用特定的业务逻辑来提取事件流并处理事件。根据业务逻辑，事件驱动的应用程序可以触发诸如发送警报、或电子邮件之类的操作，或者将事件写入向外发送的事件流以供另一个应用程序使用。

事件驱动应用程序的典型场景包括：

* 实时推荐（例如，在客户浏览零售商网站时推荐产品）
* 行为模式检测或复杂事件处理（例如，用于信用卡交易中的欺诈检测）
* 异常检测（例如，检测侵入计算机网络的尝试）

事件驱动应用程序是微服务的演变。它们通过事件日志而不是REST调用进行通信，并将应用程序数据保存为本地状态，而不是将其写入外部数据存储区（例如关系数据库或键值数据库）。图1-5显示了由事件驱动的流应用程序组成的服务架构。

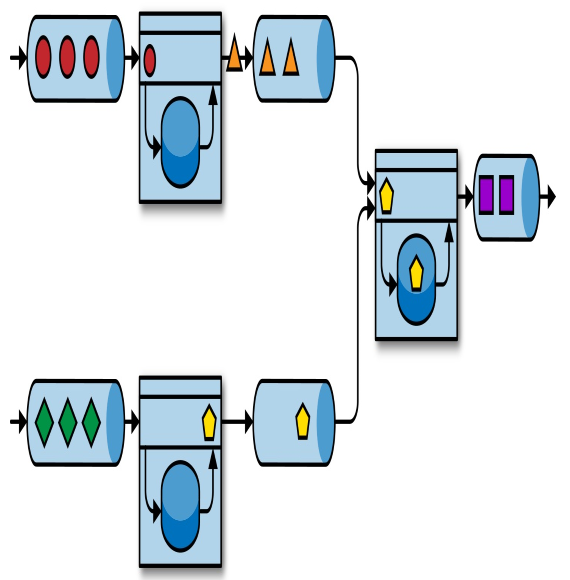
  
图1-5 事件驱动应用程序架构

图1-5中的应用程序通过事件日志连接。一个应用程序将其输出发送到事件日志通道（kafka），另一个应用程序使用其他应用程序发出的事件。事件日志通道将发送者和接收者分离，并提供异步、非阻塞的事件传输。每个应用程序都可以是有状态的，并且可以本地管理自己的状态而无需访问外部数据存储。应用程序也可以单独处理和扩展。

与事务性应用程序或微服务相比，事件驱动的应用程序具有多种优势。与读写远程数据库相比，本地状态访问提供了非常好的性能。扩展性和容错性都由流处理器来保证，并且以事件日志作为输入源，应用程序的整个输入数据可以可靠地存储，并且可以确定性地重放。此外，Flink可以将应用程序的状态重置为先前的保存点（save point），从而可以在不丢失状态的情况下更新或重新扩展应用程序。

事件驱动的应用程序对运行它们的流处理器有很高的要求，并不是所有流处理器都适合运行事件驱动的应用程序。 API的表现力，以及对状态处理和事件时间支持的程度，决定了可以实现和执行的业务逻辑。这方面取决于流处理器的API，主要看它能提供什么样的状态类型，以及它对事件时间处理的支持程度。此外，精确一次（exactly-once）的状态一致性和扩展应用程序的能力是事件驱动应用程序的基本要求。 Apache Flink符合所有的这些要求，是运行此类应用程序的一个非常好的选择。

### 数据管道（Data Pipelines）

当今的IT架构包括许多不同的数据存储，例如关系型数据库和专用数据库系统、事件日志、分布式文件系统，内存中的缓存和搜索索引。所有这些系统都以不同的格式和数据结构存储数据，为其特定的访问模式提供最佳性能。公司通常将相同的数据存储在多个不同的系统中，以提高数据访问的性能。例如，网上商店中提供的产品的信息，可以存储在交易数据库中，同时也存储在缓存（如redis）和搜索索引（如ES）中。由于数据的这种复制，数据存储必须保持同步。

在不同存储系统中同步数据的传统方法是定期ETL作业。但是，它们不能满足当今许多场景的延迟要求。另一种方法是使用事件日志（event log）来发布更新。更新将写入事件日志并由事件日志分发。日志的消费者获取到更新之后，将更新合并到受影响的数据存储中。根据使用情况，传输的数据可能需要标准化、使用外部数据进行扩展，或者在目标数据存储提取之前进行聚合。

以较低的延迟，来提取、转换和插入数据是有状态流处理应用程序的另一个常见应用场景。这种类型的应用程序称为**数据管道（data pipeline）**。数据管道必须能够在短时间内处理大量数据。操作数据管道的流处理器还应具有许多源（source）和接收器（sink）的连接器，以便从各种存储系统读取数据并将数据写入各种存储系统。当然，同样地，Flink完成了所有这些功能。

### 流分析

ETL作业定期将数据导入数据存储区，数据的处理是由即席查询（用户自定义查询）或设定好的通常查询来做的。无论架构是基于数据仓库还是基于Hadoop生态系统的组件，这都是**批处理**。多年来最好的处理方式就是，定期将数据加载到数据分析系统中，但它给分析管道带了的延迟相当大，而且无法避免。

根据设定好的时间间隔，可能需要数小时或数天才能将数据点包含在报告中。我们前面已经提到，数据管道可以实现低延迟的ETL，所以在某种程度上，可以通过使用数据管道将数据导入存储区来减少延迟。但是，即使持续不停地进行ETL操作，在用查询来处理事件之前总会有延迟。虽然这种延迟在过去可能是可以接受的，但是今天的应用程序，往往要求必须能够实时收集数据，并立即对其进行操作（例如，在手机游戏中去适应不断变化的条件，或者在电商网站中提供个性化的用户体验）。

流式分析应用程序不是等待定期触发，而是连续地提取事件流，并且通过纳入最新事件来更新其计算结果，这个过程是低延迟的。这有些类似于数据库中用于更新视图（views）的技术。通常，流应用程序将其结果存储在支持更新的外部数据存储中，例如数据库或键值（key-value）存储。流分析应用程序的实时更新结果可用于驱动监控仪表板（dashboard）应用程序，如图1-6所示。

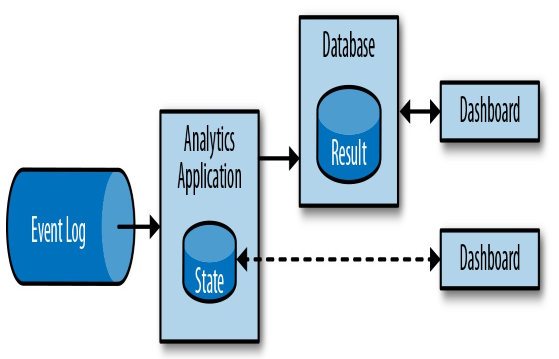


图1-6 流分析应用程序架构

流分析应用程序最大的优势就是，将每个事件纳入到分析结果所需的时间短得多。除此之外，流分析应用程序还有另一个不太明显的优势。传统的分析管道由几个独立的组件组成，例如ETL过程、存储系统、对于基于Hadoop的环境，还包括用于触发任务（jobs）的数据处理和调度程序。相比之下，如果我们运行一个有状态流应用程序，那么**流处理器**就会负责所有这些处理步骤，包括事件提取、带有状态维护的连续计算以及更新结果。此外，流处理器可以从故障中恢复，并且具有精确一次（exactly-once）的状态一致性保证，还可以调整应用程序的计算资源。像Flink这样的流处理器还支持事件时间（event-time）处理，这可以保证产生正确和确定的结果，并且能够在很短的时间内处理大量数据。

流分析应用程序通常用于：

* 监控手机网络的质量分析
* 移动应用中的用户行为
* 实时数据的即席分析

虽然我们不在此处介绍，但Flink还提供对流上的分析SQL查询的支持。

## 1.3 开源流处理的演进

数据流处理并不是一项新技术。一些最初的研究原型和商业产品可以追溯到20世纪90年代（1990s）。然而，在很大程度上，过去采用的流处理技术是由成熟的开源流处理器驱动的。如今，分布式开源流处理器在不同行业的许多企业中，处理着核心业务应用，比如电商、社交媒体、电信、游戏和银行等。开源软件是这一趋势的主要驱动力，主要原因有两个：

1. 开源流处理软件是大家每一个人都可以评估和使用的产品。

2. 由于许多开源社区的努力，可扩展流处理技术正在迅速成熟和发展。

仅仅一个Apache软件基金会就支持了十几个与流处理相关的项目。新的分布式流处理项目不断进入开源阶段，并不断增加新的特性和功能。开源社区不断改进其项目的功能，并正在推动流处理的技术边界。我们将简要介绍一下过去，看看开源流处理的起源和今天的状态。

### 流处理的历史

第一代分布式开源流处理器（2011）专注于具有毫秒延迟的事件处理，并提供了在发生故障时防止事件丢失的保证。这些系统具有相当低级的API，并且对于流应用程序的准确性和结果的一致性，不提供内置支持，因为结果会取决于到达事件的时间和顺序。另外，即使事件没有丢失，也可能不止一次地处理它们。与批处理器相比，第一代开源流处理器牺牲了结果准确性，用来获得更低的延迟。为了让当时的数据处理系统，可以同时提供快速和准确的结果，人们设计了所谓的lambda架构，如图1-7所示。

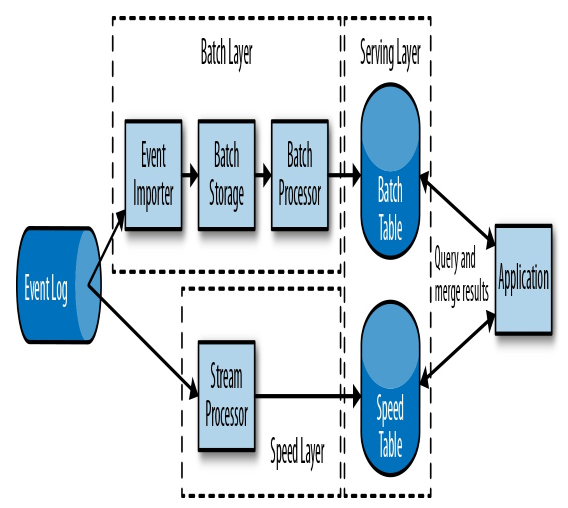


图1-7 lambda 应用架构

lambda架构增强了传统的批处理架构，其“快速层”（speed layer）由低延迟的流处理器来支持。数据到达之后由流处理器提取出来，并写入批处理存储。流处理器近乎实时地计算近似结果并将它们写入“快速表”（speed table）。批处理器定期处理批量存储中的数据，将准确的结果写入批处理表，并从速度表中删除相应的不准确结果。应用程序会合并快速表中的近似结果和批处理表中的准确结果，然后消费最终的结果。

lambda架构现在已经不再是最先进的，但仍在许多地方使用。该体系结构的最初目标是改善原始批处理分析体系结构的高延迟。但是，它有一些明显的缺点。首先，它需要对一个应用程序，做出两个语义上等效的逻辑实现，用于两个独立的、具有不同API的处理系统。其次，流处理器计算的结果只是近似的。第三，lambda架构很难建立和维护。

通过在第一代基础上进行改进，下一代分布式开源流处理器（2013）提供了更好的故障保证，并确保在发生故障时，每个输入记录仅对结果产生一次影响（exactly -once）。此外，编程API从相当低级的操作符接口演变为高级API。但是，一些改进（例如更高的吞吐量和更好的故障保证）是以将处理延迟从毫秒增加到几秒为代价的。此外，结果仍然取决于到达事件的时间和顺序。

第三代分布式开源流处理器（2015）解决了结果对到达事件的时间和顺序的依赖性。结合精确一次（exactly-once）的故障语义，这一代系统是第一个具有计算一致性和准确结果的开源流处理器。通过基于实际数据来计算结果（“重演”数据），这些系统还能够以与“实时”数据相同的方式处理历史数据。另一个改进是解决了延迟/吞吐量无法同时保证的问题。先前的流处理器仅能提供高吞吐量或者低延迟（其中之一），而第三代系统能够同时提供这两个特性。这一代的流处理器使得lambda架构过时了。当然，这一代流处理以flink为代表。

除了目前讨论的特性，例如容错、性能和结果准确性之外，流处理器还不断添加新的操作功能，例如高可用性设置，与资源管理器（如YARN或Kubernetes）的紧密集成，以及能够动态扩展流应用程序。其他功能包括：支持升级应用程序代码，或将作业迁移到其他群集或新版本的流处理器，而不会丢失当前状态。

## 1.4 Flink 简介

Apache Flink是第三代分布式流处理器，它拥有极富竞争力的功能。它提供准确的大规模流处理，具有高吞吐量和低延迟。特别的是，以下功能使Flink脱颖而出：

* 事件时间（event-time）和处理时间（processing-tme）语义。即使对于无序事件流，事件时间（event-time）语义仍然能提供一致且准确的结果。而处理时间（processing-time）语义可用于具有极低延迟要求的应用程序。
* 精确一次（exactly-once）的状态一致性保证。
* 每秒处理数百万个事件，毫秒级延迟。 Flink应用程序可以扩展为在数千个核（cores）上运行。
* 分层API，具有不同的权衡表现力和易用性。本书介绍了DataStream API和过程函数（process function），为常见的流处理操作提供原语，如窗口和异步操作，以及精确控制状态和时间的接口。本书不讨论Flink的关系API，SQL和LINQ风格的Table API。
* 连接到最常用的存储系统，如Apache Kafka，Apache Cassandra，Elasticsearch，JDBC，Kinesis和（分布式）文件系统，如HDFS和S3。
* 由于其高可用的设置（无单点故障），以及与Kubernetes，YARN和Apache Mesos的紧密集成，再加上从故障中快速恢复和动态扩展任务的能力，Flink能够以极少的停机时间7\*24全天候运行流应用程序。
* 能够更新应用程序代码并将作业（jobs）迁移到不同的Flink集群，而不会丢失应用程序的状态。
* 详细且可自定义的系统和应用程序指标集合，以提前识别问题并对其做出反应。
* 最后但同样重要的是，Flink也是一个成熟的批处理器。

除了这些功能之外，Flink还是一个非常易于开发的框架，因为它易于使用的API。嵌入式执行模式，可以在单个JVM进程中启动应用程序和整个Flink系统，这种模式一般用于在IDE中运行和调试Flink作业。在开发和测试Flink应用程序时，此功能非常有用。

### 运行你的第一个Flink应用程序

接下来，我们将启动一个本地群集，然后运行一个简单的流应用程序，对Flink有一个初步的了解。我们将要运行的应用程序，会按时间转换并聚合温度传感器的读数。对于这个例子，你的系统需要安装Java 8。我们将介绍UNIX环境下的步骤，如果你的系统是Windows，建议配置一个Linux虚拟机，或者Cygwin（适用于Windows的Linux环境），或是Windows的Linux子系统（Windows 10之后引入）。以下步骤将会展示如何启动本地Flink集群并提交执行应用程序。

1. 到Flink官网下载免Hadoop依赖的Flink发行版1.7.1，要求Scala版本为2.12。

2．提取压缩文件

$ tar xvfz flink-1.7.1-binscala\_2.12.tgz

3. 启动一个Flink本地集群：

$ cd flink-1.7.1

$ ./bin/start-cluster.sh

Starting cluster.

Starting standalonesession daemon on host xxx.

Starting taskexecutor daemon on host xxx

3. 通过在浏览器访问URL http://localhost:8081，打开Flink的Web UI。如图1-8 所示，你将会看到刚刚启动的Flink集群的一些统计信息。我们可以看到，一个单独的TaskManager（Flink的工作进程）已经连接上，一个单独的任务插槽（TaskManager提供的资源单位）已经是可用状态了。

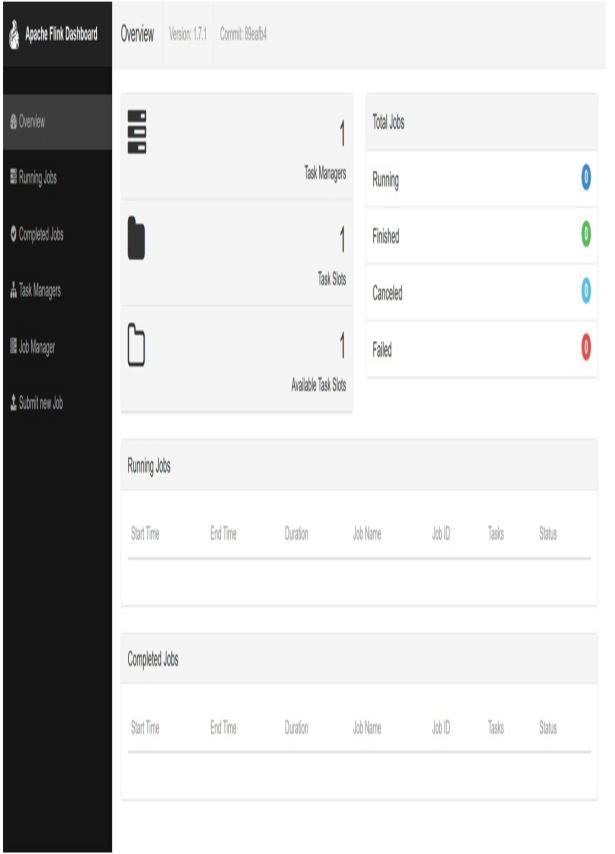


图1-8 显示了概况的Flink网页监控页面

5. 下载有本书中示例的JAR包：

$ wget https://streaming-withflink.github.io/\

examples/download/examples-scala.jar

6. 在本地集群上运行示例程序，需要指明应用的入口类和JAR文件名：

$ ./bin/flink run \ -c io.github.streamingwithflink.chapter1.Av erageSensorReadings \ examples-scala.jar

Starting execution of program Job has been submitted with JobID cfde9dbe315ce162444c475a08cf93d9

7. 查看Web控制台，可以看到在“Running Jobs”（正在运行的作业）中列出了一个作业。如果单击该作业，就可以看到正在运行的作业算子的数据流和实时指标，应该与图1-9所示类似。

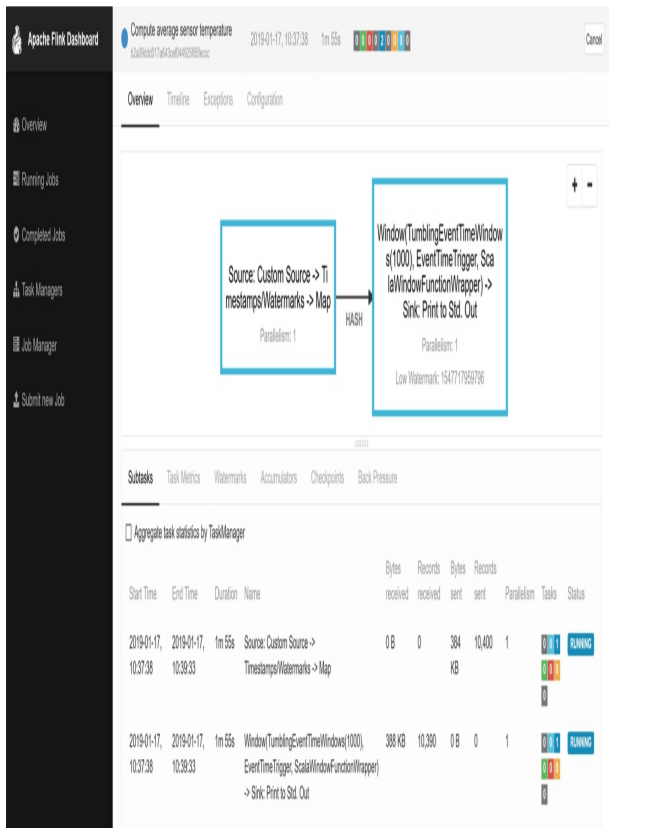


图1-9 显示了一个正在运行任务的Flink网页监控页面

8. 作业的输出会写到Flink工作进程的标准输出里，它被默认重定向到了./log目录下的文件中。我们也可以用tail命令来监控不断生成的输出数据：

$ tail -f ./log/flink-<user>taskexecutor-<n>-<hostname>.out

你应该看到类似这样的数据一行一行写入文件中：

SensorReading(sensor\_1,1547718199000,35.80018327300259)

SensorReading(sensor\_6,1547718199000,15.402984393403084)

SensorReading(sensor\_7,1547718199000,6.720945201171228)

SensorReading(sensor\_10,1547718199000,38.101067604893444)

SensorReading的第一个字段表示传感器ID；第二个字段是时间戳，表示从1970-01-01-00:00:00.000 以来的毫秒数；第三个字段是5秒之内收集到的平均温度值。

9. 我们运行的是流处理程序，所以除非你去手动取消，否则它会一直运行。你可以在Web控制台里选中作业，然后点击页面上方的“Cancel”（取消）按钮来取消它。

10. 最后，我们可以停掉整个Flink集群：

$ ./bin/stop-cluster.sh

这就是整个流程。你已经创建并启动了自己的Flink本地集群，而且运行了自己的第一个Flink流处理程序！当然，关于Flink的流式处理我们还有很多内容要继续学习，这正是这本书要讨论的。

## 1.5 总结

在本章中，我们介绍了有状态的流处理，讨论了它的使用场景，并初步了解了Apache Flink。我们首先概述了传统的数据基础架构，业务应用通常如何设计，以及当今大多数公司如何收集和分析数据。然后，我们介绍了有状态的流处理的概念，并解释了它如何处理各种应用场景，从业务应用和微服务，到ETL和数据分析。我们还讨论了自2010年初开始以来，开源流处理系统如何发展，以及流处理如何成为当今企业的可行解决方案。最后，我们了解了一下Apache Flink及其提供的广泛功能，并展示了如何搭建本地Flink集群，以及运行了第一个流处理应用程序。