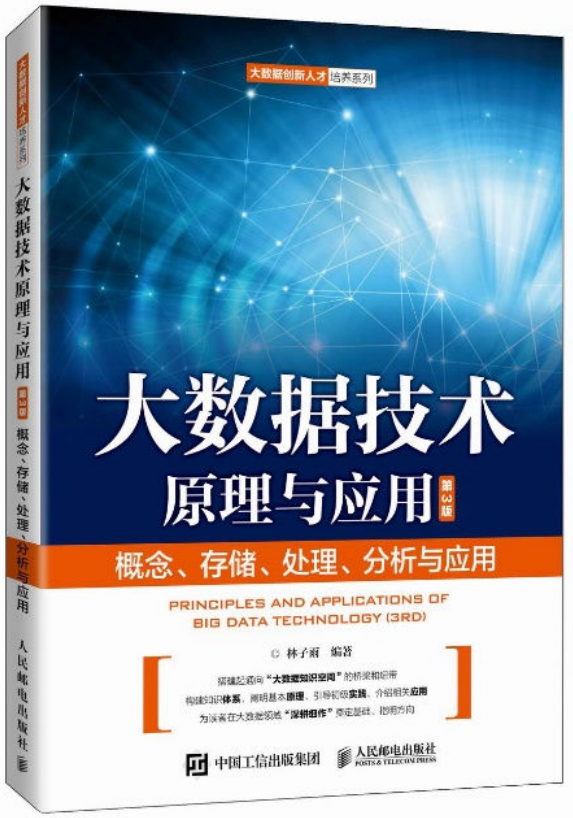
大数据技术原理与应用

12. Flink

陈建文
电子信息与通信学院
chenjw@hust.edu.cn

12. Flink

12.1 Flink 简介
12.2 为什么选择Flink
12.3 Flink应用场景
12.4 Flink中统一数据处理
12.5 Flink技术栈
12.6 Flink工作原理
12.7 Flink编程模型
12.8 Flink应用程序结构
12.9 Flink 编程实践



12.1 Flink简介

- **Flink是Apache软件基金会的一个顶级项目，是分布式、高性能、随时可用以及准确的流处理应用程序打造的开源流处理框架，并且可以同时支持实时计算和批量计算。**
- **Flink起源于Stratosphere 项目，该项目是在2010年到2014年间由柏林工业大学、柏林洪堡大学和哈索普拉特纳研究所联合开展的。**
- **2014年4月，Stratosphere代码被贡献给Apache软件基金会，成为Apache软件基金会孵化器项目。**
- **2014年12月，Flink项目成为Apache软件基金会顶级项目。目前，Flink是Apache软件基金会的5个最大的大数据项目之一。**

- 在全球范围内拥有350多位开发人员，并在越来越多的企业中得到了应用，在国内，包括阿里巴巴、美团、滴滴等在内的知名互联网企业，都已经开始大规模使用Flink作为企业的分布式大数据处理引擎。
- 在阿里巴巴，基于Flink搭建的平台于2016年正式上线，并从阿里巴巴的搜索和推荐这两大场景开始实现。目前，阿里巴巴所有的业务，包括阿里巴巴所有子公司都采用了基于Flink搭建的实时计算平台，服务器规模已经达到数万台，这种规模等级在全球范围内也是屈指可数。
- 阿里巴巴的Flink平台内部积累起来的状态数据，已经达到PB级别规模，每天在平台上处理的数据量已经超过万亿条，在峰值期间可以承担每秒超过4.72亿次的访问，最典型的应用场景是阿里巴巴“双11”大屏。

- **Flink**具有十分强大的功能，可以支持不同类型的应用程序。**Flink**的主要特性包括：批流一体化、精确的状态管理、事件时间支持以及精确一次的状态一致性保障等。
- **Flink** 不仅可以运行在包括 **YARN**、 **Mesos**、**Kubernetes** 等在内的多种资源管理框架上，还支持在裸机集群上独立部署，在启用高可用选项的情况下，它不存在单点失效问题。
- 事实证明，**Flink** 已经可以扩展到数千核心，其状态可以达到 **TB** 级别，且仍能保持高吞吐、低延迟的特性有很多要求严苛的流处理应用都运行在 **Flink** 之上。

12.2 为什么选择Flink

- 12.2.1 传统数据处理架构
- 12.2.2 大数据Lambda架构
- 12.2.3 流处理架构
- 12.2.4 Flink是理想的流计算框架
- 12.2.5 Flink的优势

12.2.1 传统数据处理架构

传统数据处理架构的一个显著特点就是采用一个中心化的数据库系统，用于存储事务性数据。

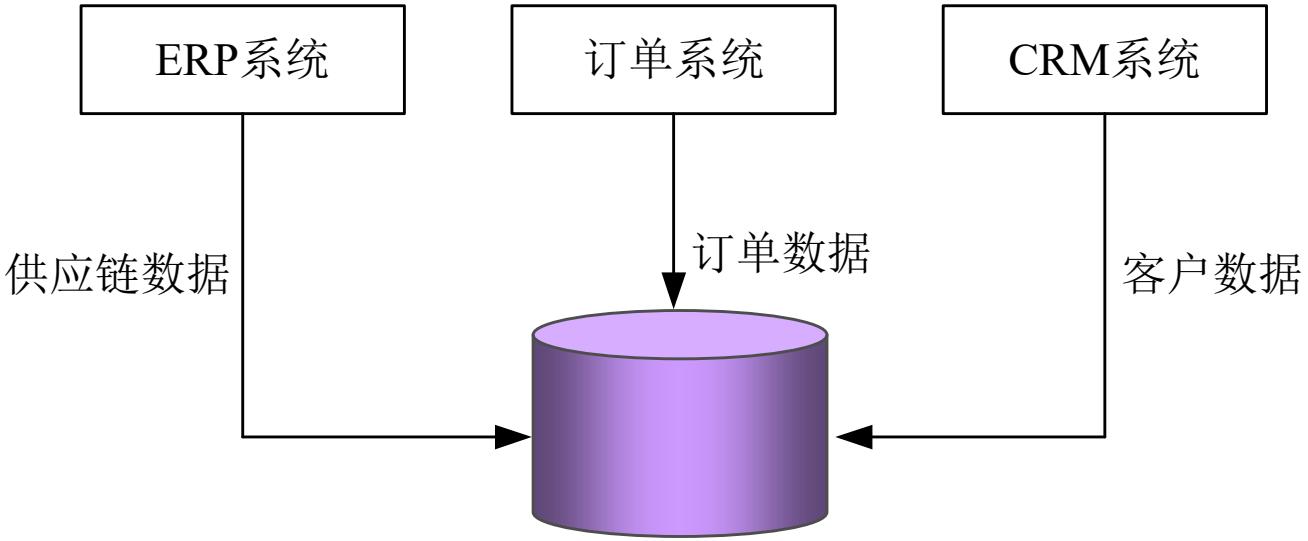


图12-1 传统数据处理架构

12.2.2 大数据Lambda架构

大数据Lambda架构主要包含两层，即批处理层和实时处理层，在批处理层中，采用MapReduce、Spark等技术进行批量数据处理，而在实时处理层中，则采用Storm、Spark Streaming等技术进行数据的实时处理。

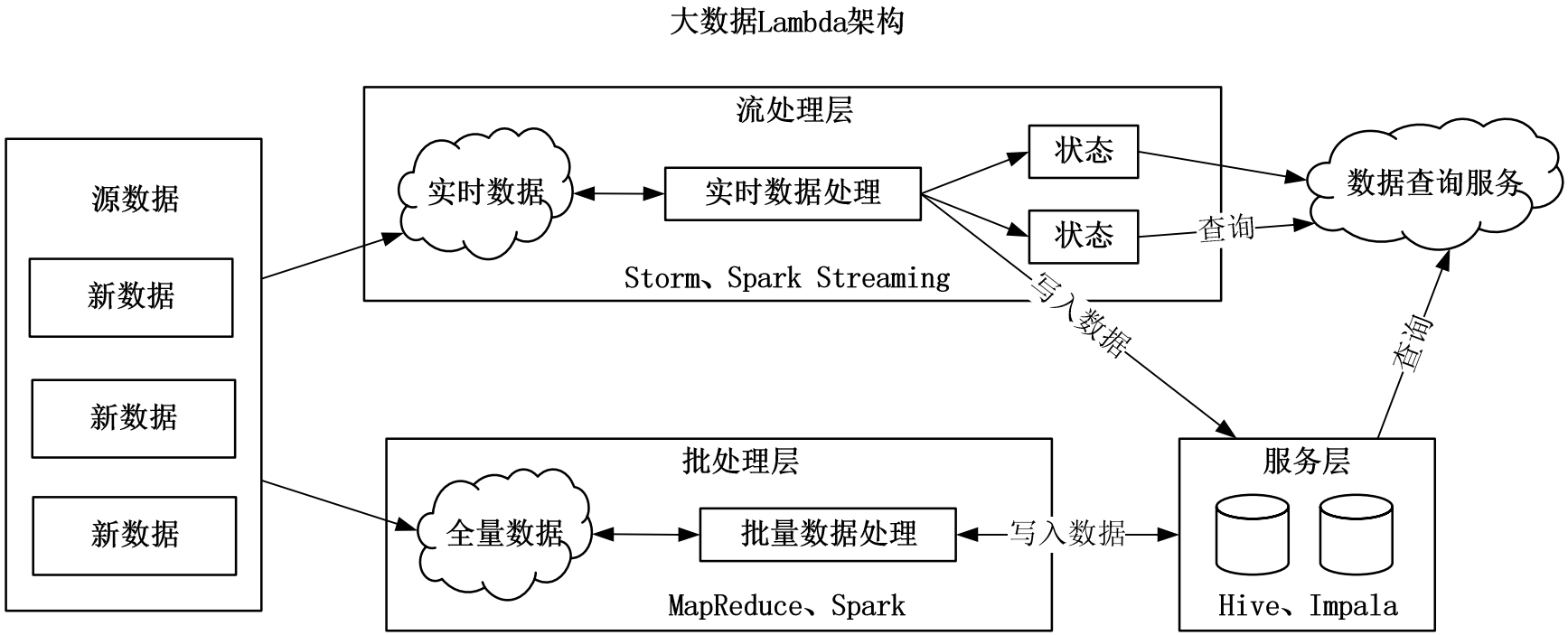


图 12.2大数据Lambda架构

12.2.3 流处理架构

为了高效地实现流处理架构，一般需要设置消息传输层和流处理层（如图**12-3**所示）。消息传输层从各种数据源采集连续事件产生的数据，并传输给订阅了这些数据的应用程序；流处理层会持续地将数据在应用程序和系统间移动，聚合并处理事件，并在本地维持应用程序的状态。

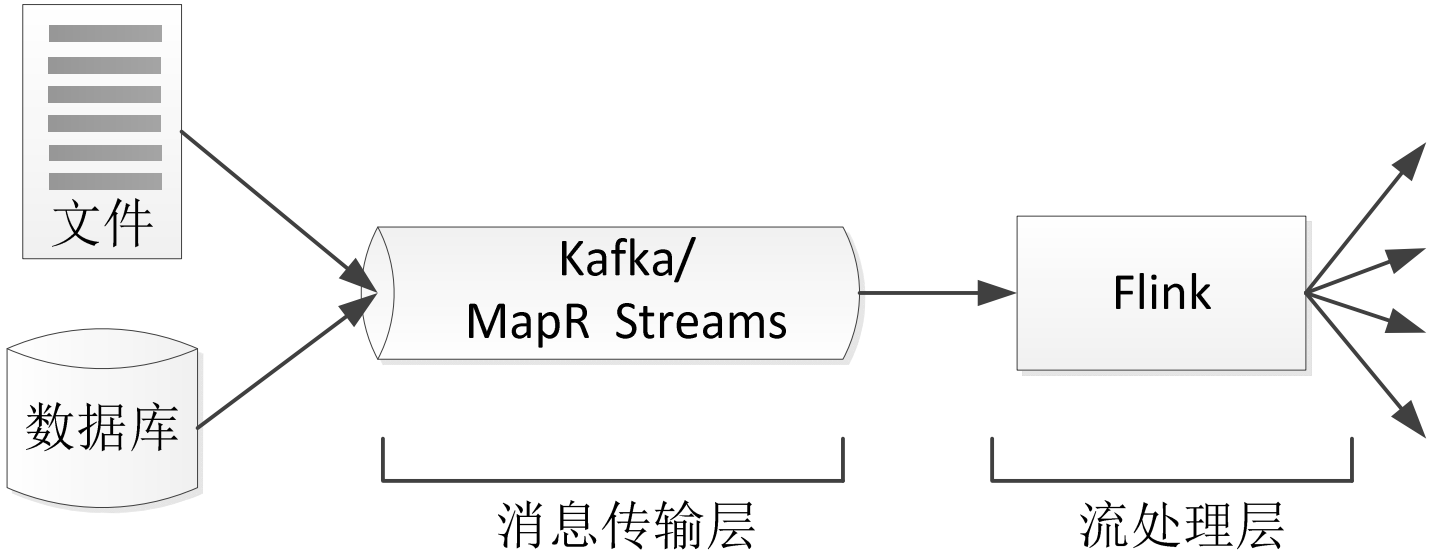


图12-3 流处理架构

流处理架构的核心是使各种应用程序互连在一起的消息队列，消息队列连接应用程序，并作为新的共享数据源，这些消息队列取代了从前的大型集中式数据库。如图**12-4**所示，流处理器从消息队列中订阅数据并加以处理，处理后的数据可以流向另一个消息队列，这样，其他应用程序都可以共享流数据。在一些情况下，处理后的数据会被存放到本地数据库中。

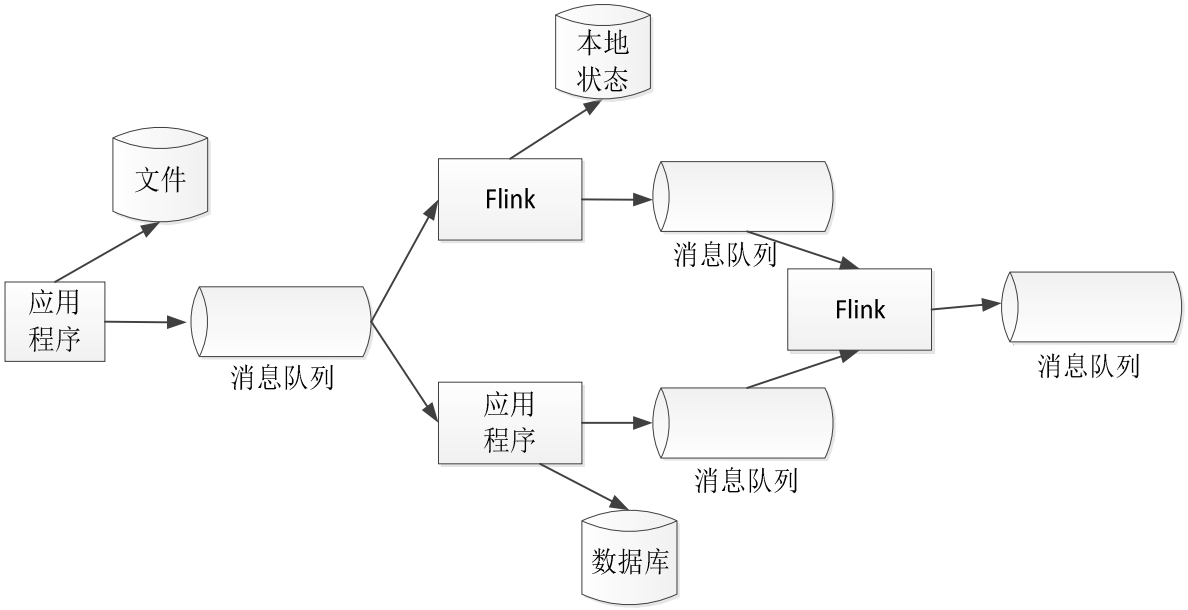


图12-4 流处理架构中的消息队列

- 流处理架构正在逐步取代传统数据处理架构和**Lambda**架构，成为大数据处理架构的一种新趋势。
- 一方面，由于流处理架构中不存在一个大型集中式数据库，因此，避免了传统数据处理架构中存在的“数据库不堪重负”的问题。
- 另一方面，在流处理架构中，批处理被看成是流处理的一个子集，因此，就可以用面向流处理的框架进行批处理，这样就可以用一个流处理框架来统一处理流计算和批量计算，避免了**Lambda**架构中存在的“多个框架难管理”的问题。

12.2.4 Flink是理想的流计算框架

- 流处理架构需要具备低延迟、高吞吐和高性能的特性，而目前从市场上已有的产品来看，只有**Flink**可以满足要求。
- **Storm**虽然可以做到低延迟，但是无法实现高吞吐，也不能在故障发生时准确地处理计算状态。
- **Spark Streaming**通过采用微批处理方法实现了高吞吐和容错性，但是牺牲了低延迟和实时处理能力。
- **Spark**的另一个流计算组件**Structured Streaming**，包括微批处理和持续处理两种处理模型。采用微批处理时，最快响应时间需要**100**毫秒，无法支持毫秒级别响应。采用持续处理模型时，可以支持毫秒级别响应，但是，只能做到“至少一次”的一致性，无法做到“精确一次”的一致性。

- **Flink**实现了**Google Dataflow**流计算模型，是一种兼具高吞吐、低延迟和高性能的实时流计算框架，并且同时支持批处理和流处理。此外，**Flink**支持高度容错的状态管理，防止状态在计算过程中因为系统异常而出现丢失。因此，**Flink**就成为了能够满足流处理架构要求的理想的流计算框架。

表3-1 不同流计算框架的对比

产品	消息保证机制	容错机制	状态管理	延时	吞吐量
Storm	至少一次	Acker机制	无	低	低
Spark Streaming	精确一次	基于RDD的检查点	基于DStream	中	高
Flink	精确一次	检查点	基于操作	低	高

12.2.5 Flink的优势

Flink具有以下优势：

（1）同时支持高吞吐、低延迟、高性能

（2）同时支持流处理和批处理

（3）高度灵活的流式窗口

（4）支持有状态计算

（5）具有良好的容错性

（6）具有独立的内存管理

（7）支持迭代和增量迭代

12.3 Flink应用场景

- 12.3.1 事件驱动型应用
- 12.3.2 数据分析应用
- 12.3.3 数据流水线应用

12.3.1 事件驱动型应用

（1）什么是事件驱动型应用

事件驱动型应用是一类具有状态的应用，它从一个或多个事件数据流中读取事件，并根据到来的事件做出反应，包括触发计算、状态更新或其他外部动作等。事件驱动型应用是在传统的应用设计基础上进化而来的。在传统的设计中，通常都具有独立的计算和数据存储层，应用会从一个远程的事务数据库中读写数据。而事务驱动型应用是建立在有状态流处理应用的基础之上的。在这种设计中，数据和计算不是相互独立的层，而是放在一起的，应用只需访问本地（内存或磁盘）即可获取数据。系统容错性是通过定期向远程持久化存储写入检查点来实现的。

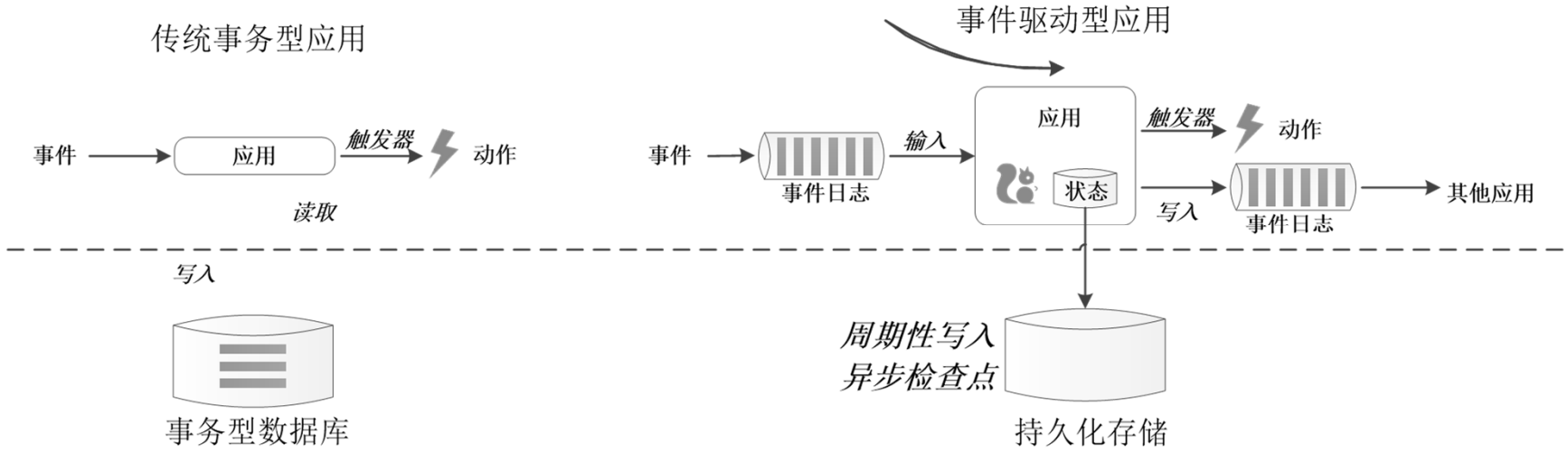


图12-5 传统应用和事件驱动型应用架构的区别

典型的事件驱动型应用包括反欺诈、异常检测、基于规则的报警、业务流程监控、**Web** 应用（社交网络）等。

（2）事件驱动型应用的优势

事件驱动型应用都是访问本地数据，而无需查询远程的数据库，这样，无论是在吞吐量方面，还是在延迟方面，都可以获得更好的性能。向一个远程的持久化存储周期性地写入检查点，可以采用异步和增量的方式来实现。因此，检查点对于常规的事件处理的影响是很小的。事件驱动型应用的优势不仅限于本地数据访问。在传统的分层架构中，多个应用共享相同的数据库，是一个很常见的现象。因此，数据库的任何变化，比如，由于一个应用的更新或服务的升级而导致的数据布局的变化，都需要谨慎协调。由于每个事件驱动型应用都只需要考虑自身的数据，对数据表示方式的改变或者应用的升级，都只需要很少的协调工作。

（3）Flink是如何支持事件驱动型应用的

- 一个流处理器如何能够很好地处理时间和状态，决定了事件驱动型应用的局限性。Flink许多优秀的特性都是围绕这些方面进行设计的。Flink提供了丰富的状态操作原语，它可以管理大量的数据（可以达到TB级别），并且可以确保“精确一次”的一致性。而且，Flink还支持事件时间、高度可定制的窗口逻辑和细粒度的时间控制，这些都可以帮助实现高级的商业逻辑。Flink还拥有一个复杂事件处理（CEP）类库，可以用来检测数据流中的模式。
- Flink中针对事件驱动应用的突出特性当属“保存点”（savepoint）。保存点是一个一致性的状态镜像，它可以作为许多相互兼容的应用的一个初始化点。给定一个保存点以后，就可放心对应用进行升级或扩容，还可以启动多个版本的应用来完成 A/B 测试。

12.3.2 数据分析应用

（1）什么是数据分析应用

分析作业会从原始数据中提取信息，并得到富有洞见的观察。传统的数据分析通常先对事件进行记录，然后在这个有界的数据集上执行批量查询。为了把最新的数据融入到查询结果中，就必须把这些最新的数据添加到被分析的数据集中，然后重新运行查询。查询的结果会被写入到一个存储系统中，或者形成报表。

一个高级的流处理引擎，可以支持实时的数据分析。这些流处理引擎并非读取有限的数据集，而是获取实时事件流，并连续产生和更新查询结果。这些结果或者被保存到一个外部数据库中，或者作为内部状态被维护。仪表盘应用可以从这个外部的数据库中读取最新的结果，或者直接查询应用的内部状态。

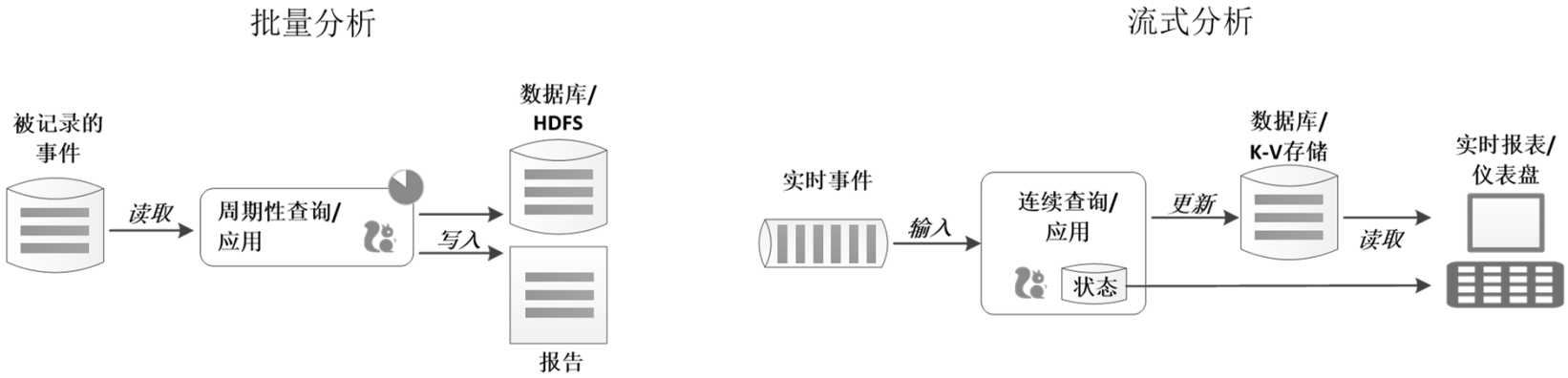


图12-6 Flink同时支持流式及批量分析应用

典型的数据分析应用包括电信网络质量监控、移动应用中的产品更新及实验评估分析、消费者技术中的实时数据即席分析、大规模图分析等。

（2）流式分析应用的优势

与批量分析相比，连续流式分析的优势是，由于消除了周期性的导入和查询，因而从事件中获取洞察结果的延迟更低。此外，流式查询不需要处理输入数据中的人为产生的边界。

另一方面，流式分析具有更加简单的应用架构。一个批量分析流水线会包含一些独立的组件来周期性地调度数据提取和查询执行。如此复杂的流水线，操作起来并非易事，因为，一个组件的失败就会直接影响到流水线中的其他步骤。相反，运行在一个高级流处理器（比如**Flink**）之上的流式分析应用，会把从数据提取到连续结果计算的所有步骤都整合起来，因此，它就可以依赖底层引擎提供的故障恢复机制。

（3）Flink是如何支持数据分析应用的

Flink可以同时支持批处理和流处理。Flink提供了一个符合ANSI规范的SQL接口，它可以为批处理和流处理提供一致的语义。不管是运行在一个静态的数据集上，还是运行在一个实时的数据流上，SQL查询都可以得到相同的结果。Flink还提供了丰富的用户自定义函数，使得用户可以在SQL查询中执行自定义代码。如果需要进一步定制处理逻辑，Flink的DataStream API和DataSet API提供了更加底层的控制。此外，Flink 的 Gelly 库为基于批量数据集的大规模高性能图分析提供了算法和构建模块支持。

12.3.3 数据流水线应用

（1）什么是数据流水线

Extract-transform-load（ETL）是一个在存储系统之间转换和移动数据的常见方法。通常而言，**ETL**作业会被周期性地触发，从而把事务型数据库系统中的数据复制到一个分析型数据库或数据仓库中。

数据流水线可以实现和**ETL**类似的功能，它们可以转换、清洗数据，或者把数据从一个存储系统转移到另一个存储系统中。但是，它们是以一种连续的流模式来执行的，而不是周期性地触发。因此，当数据源中源源不断地生成数据时，数据流水线就可以把数据读取过来，并以较低的延迟转移到目的地。比如，一个数据流水线可以对一个文件系统目录进行监控，一旦发现有新的文件生成，就读取文件内容并写入到事件日志中。再比如，将事件流物化到数据库或增量构建和优化查询索引。

图12-7 周期性 ETL 作业和持续数据流水线的差异

典型的数据流水线应用包括电子商务中的实时查询索引构建、电子商务中的持续**ETL**等。

（2）数据流水线的优势

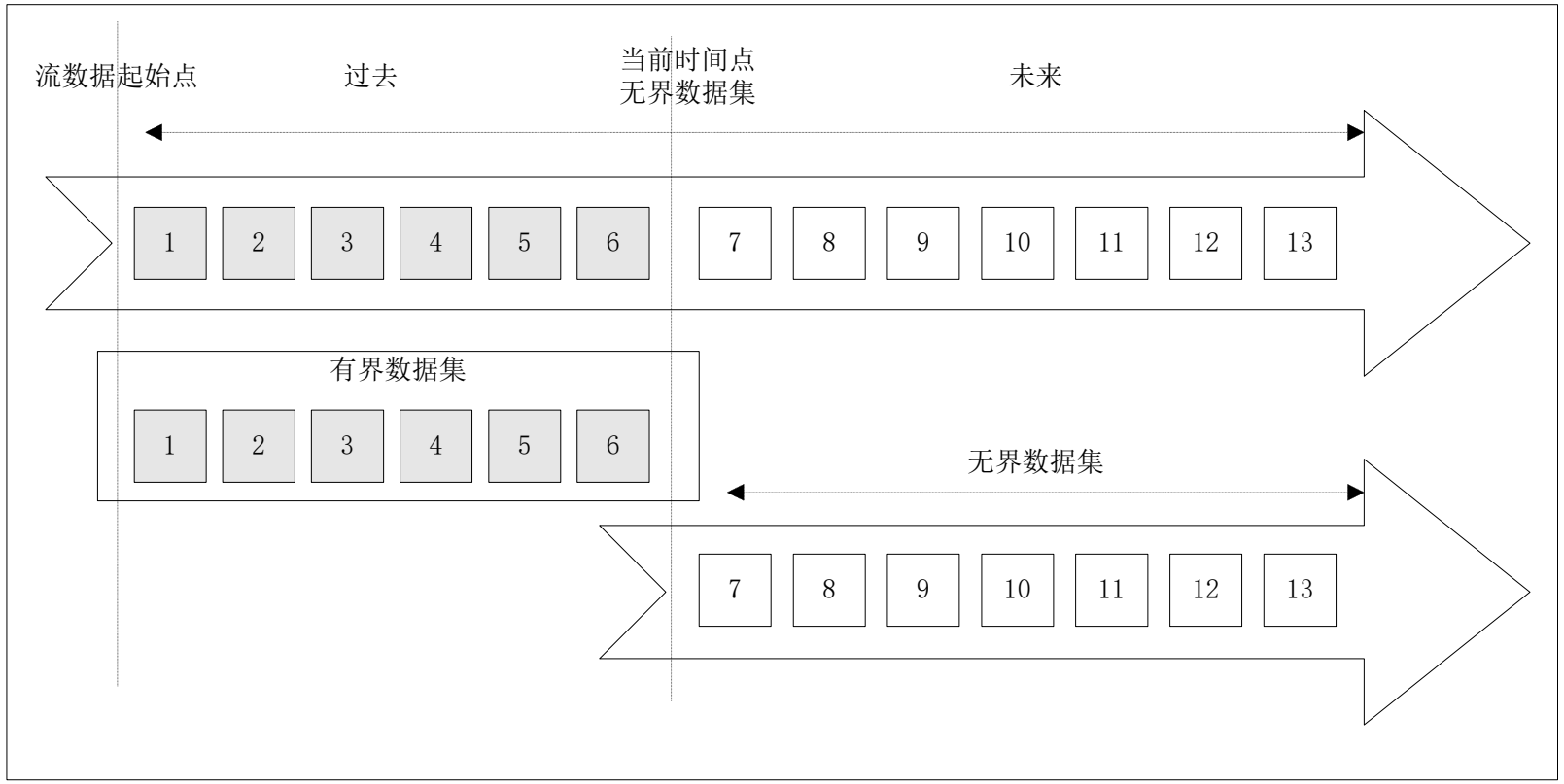
相对于周期性的**ETL**作业而言，连续的数据流水线的优势是，减少了数据转移过程的延迟。此外，由于它能够持续消费和发送数据，因此用途更广，支持用例更多。

（3）Flink如何支持数据流水线应用

Flink的SQL接口（或者Table API）以及丰富的用户自定义函数，可以解决许多常见的数据转换问题。通过使用更具通用性的DataStream API，还可以实现具有更加强大功能的数据流水线。Flink提供了大量的连接器，可以连接到各种不同类型的数据存储系统，比如Kafka、Kinesis、Elasticsearch和JDBC数据库系统。同时，Flink供了面向文件系统的连续型数据源，可用来监控目录变化，并提供了数据槽（sink），支持以时间分区的方式写入文件。

12.4 Flink中统一数据处理

根据数据的产生方式，我们可以把数据集分为两种类型：有界数据集和无界数据集



- 有界数据集具有时间边界，在处理过程中数据一定会在某个时间范围内起始和结束，有可能是一小时，也有可能是一天内的交易数据。有界数据集的特点是，数据是静止不动的，不会存在数据的追加操作。对有界数据集的数据处理方式被称为批处理
- 对于无界数据集，数据从开始生成就一直持续不断地产生新的数据，因此数据是没有边界的，例如服务器信令、网络传输流、传感器信号数据、实时日志信息等。和批量数据处理方式对应，对无界数据集的数据处理方式被称为流处理。
- 有界数据集与无界数据集是一个相对模糊的概念。对于有界数据集而言，如果数据一条一条地经过处理引擎，那么也可以认为是无界的。反过来，对于无界数据集而言，如果每间隔一分钟、一小时、一天进行一次计算，那么也可以认为这一段时间内的数据又相对是有界的。

- 对于**Spark**而言，它会使用一系列连续的微小批处理来模拟流处理，也就是说，它会在特定的时间间隔内发起一次计算，而不是每条数据都触发计算，这就相当于把无界数据集切分为多个小量的有界数据集。

- 对于**Flink**而言，它把有界数据集看成无界数据集的一个子集，因此，将批处理与流处理混合到同一套引擎当中，用户使用**Flink**引擎能够同时实现批处理与流处理任务。

12.5 Flink技术栈

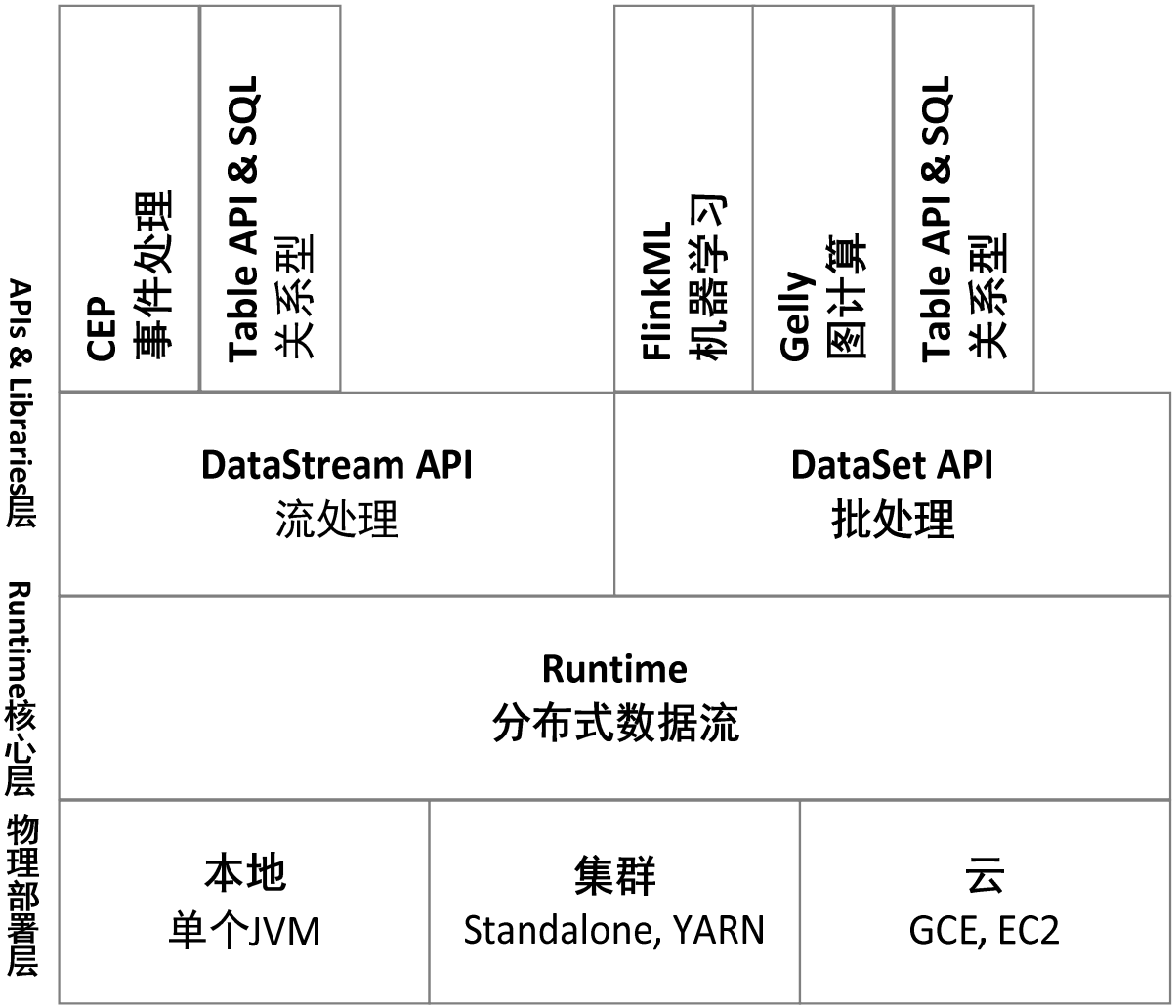


图12-8 Flink核心组件栈

12.6 Flink工作原理

Flink系统主要由两个组件组成，分别为JobManager和TaskManager，Flink 架构也遵循Master-Slave架构设计原则，JobManager为Master节点，TaskManager为Slave节点。

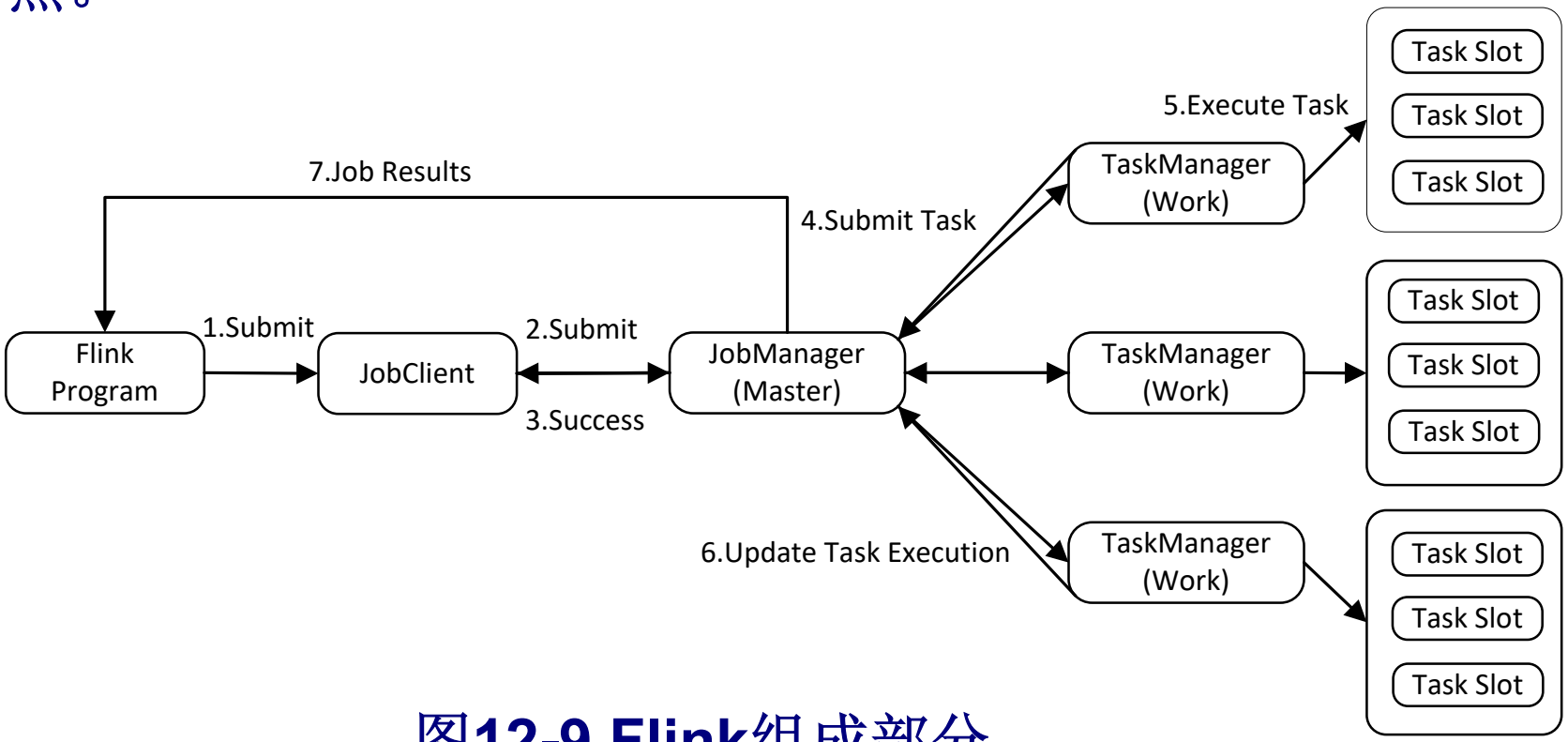


图12-9 Flink组成部分

12.7 Flink编程模型

Flink 提供了不同级别的抽象（如图12-10所示），以开发流或批处理作业。

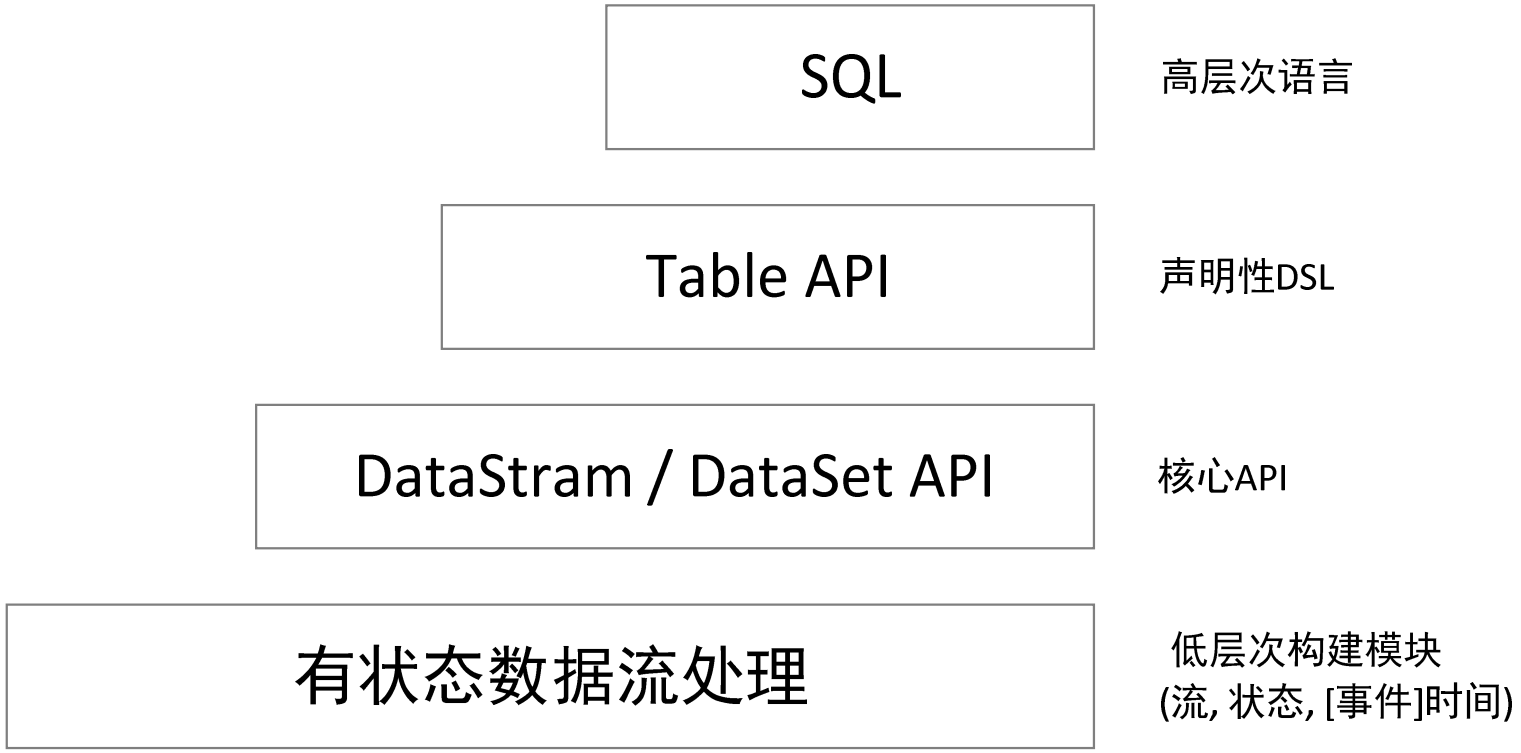


图12-10 Flink编程模型

12.8 Flink应用程序结构

如图所示，一个完整的Flink应用程序结构包含如下三个部分：

（1）数据源（Source）：Flink 在流处理和批处理上的数据源大概有4类：基于本地集合的数据源、基于文件的数据源、基于网络套接字的数据源、自定义的数据源。常见的自定义数据源包括Apache kafka、Amazon Kinesis Streams、RabbitMQ、Twitter Streaming API、Apache NiFi等，当然用户也可以定义自己的数据源。

（2）数据转换（Transformation）：数据转换的各种操作包括map、 flatMap、filter、keyBy、reduce、aggregation、window、windowAll、union、select等，可以将原始数据转换成满足要求的数据。

（3）数据输出（Sink）：数据输出是指Flink将转换计算后的数据发送的目的地。常见的数据输出包括写入文件、打印到屏幕、写入Socket 、自定义Sink等 。常见的自定义Sink有Apache kafka、RabbitMQ、MySQL、ElasticSearch、Apache Cassandra、Hadoop FileSystem 等。

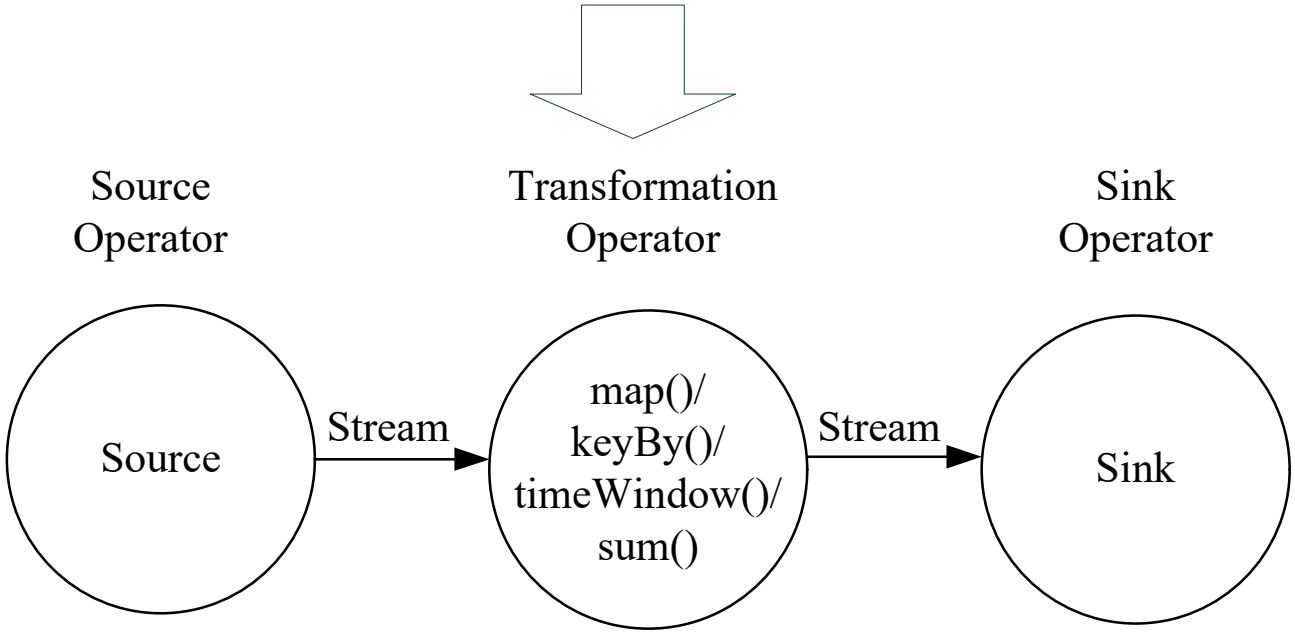
下图以一段简单代码为实例，演示了Flink的应用程序结构。

```
val source = env.socketTextStream("localhost",9999,'\n')

val dataStream = source.flatMap(_.split(" "))
    .map((_,1))
    .keyBy(0)
    .timeWindow(Time.seconds(2),Time.seconds(2))
    .sum(1)

dataStream.print()
```

Source

Transformation

Sink



12.9 Flink编程实践

- 12.9.1 安装Flink
- 12.9.2 编程实现WordCount程序

12.9.1 安装Flink

Flink的运行需要Java环境的支持，因此，在安装Flink之前，请先参照相关资料安装Java环境（比如Java8）。然后，到Flink官网（https://flink.apache.org/downloads.html）下载安装包，比如下载得到的安装文件为flink-1.9.1-bin-scala_2.11.tgz，并且保存在Linux系统中（假设当前使用hadoop用户名登录了Linux系统，并且安装文件被保存到了“/home/hadoop/Downloads”目录下），然后，使用如下命令对安装文件进行解压缩：

```
$ cd /home/hadoop/Downloads
$ sudo tar -zxvf flink-1.9.1-bin-scala_2.11.tgz -C /usr/local
```

修改目录名称，并设置权限，命令如下：

```
$ cd /usr/local
$ sudo mv ./ flink-1.9.1 ./flink
$ sudo chown -R hadoop:hadoop ./flink
```

Flink对于本地模式是开箱即用的，如果要修改**Java**运行环境，可以修改“**/usr/local/flink/conf/flink-conf.yaml**”文件中的**env.java.home**参数，设置为本地**Java**的绝对路径。使用如下命令添加环境变量：

```
$ vim ~/.bashrc
```

在**.bashrc**文件中添加如下内容：

```
export FLNK_HOME=/usr/local/flink
export PATH=$FLINK_HOME/bin:$PATH
```

保存并退出**.bashrc**文件，然后执行如下命令让配置文件生效：

```
$ source ~/.bashrc
```

使用如下命令启动**Flink**：

```
$ cd /usr/local/flink
$ ./bin/start-cluster.sh
```

使用**jps**命令查看进程：

```
$ jps
17942 TaskManagerRunner
18022 Jps
17503 StandaloneSessionClusterEntrypoint
```

如果能够看到**TaskManagerRunner**和**StandaloneSessionClusterEntrypoint**这两个进程，就说明启动成功。

Flink的JobManager同时会在8081端口上启动一个Web前端，可以在浏览器中输入“http://localhost:8081”来访问。

Flink安装包中自带了测试样例，这里可以运行WordCount样例程序来测试Flink的运行效果，具体命令如下：

```
$ cd /usr/local/flink/bin
$ ./flink run /usr/local/flink/examples/batch/WordCount.jar
```

执行上述命令以后，如果执行成功，应该可以看到类似如下的屏幕信息：

```
Starting execution of program
Executing WordCount example with default input data set.
Use --input to specify file input.
Printing result to stdout. Use --output to specify output path.
(a,5)
(action,1)
(after,1)
(against,1)
(all,2)……
```

12.9.2 编程实现WordCount程序

编写WordCount程序主要包括以下几个步骤：

（1）安装Maven

（2）编写代码

（3）使用Maven打包Java程序

（4）通过flink run命令运行程序

（1）安装Maven

Ubuntu中没有自带安装Maven，需要手动安装Maven。可以访问Maven官网下载安装文件，下载地址如下：
http://apache.fayea.com/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.zip

下载到**Maven**安装文件以后，保存到“**~/Downloads**”目录下。然后，可以选择安装在“**/usr/local/maven**”目录中，命令如下：

```
$ sudo unzip ~/Downloads/apache-maven-3.3.9-bin.zip -d /usr/local
$ cd /usr/local
$ sudo mv apache-maven-3.3.9/ ./maven
$ sudo chown -R hadoop ./maven
```

（2） 编写代码

在**Linux**终端中执行如下命令，在用户主文件夹下创建一个文件夹**flinkapp**作为应用程序根目录：

```
$ cd ~ #进入用户主文件夹
$ mkdir -p ./flinkapp/src/main/java
```

然后，使用**vim**编辑器在“**./flinkapp/src/main/java**”目录下建立三个代码文件，即**WordCountData.java**、**WordCountTokenizer.java**和**WordCount.java**。

WordCountData.java用于提供原始数据，其内容如下：

```
package cn.edu.xmu;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
public class WordCountData {
   public static final String[] WORDS=new String[]{"To be, or not to be,--that is the question:--", "Whether \'tis nobler in the mind to suffer", "The slings and arrows of outrageous fortune", "Or to take arms against a sea of troubles,", "And by opposing end them?--To die,--to sleep,--", "No more; and by a sleep to say we end", "The heartache, and the thousand natural shocks", "That flesh is heir to,--\'tis a consummation", "Devoutly to be wish\'d. To die,--to sleep;--", "To sleep! perchance to dream:--ay, there\'s the rub;", "For in that sleep of death what dreams may come,", "When we have shuffled off this mortal coil,", "Must give us pause: there\'s the respect", "That makes calamity of so long life;", "For who would bear the whips and scorns of time,", "The oppressor\'s wrong, the proud man\'s contumely,", "The pangs of despis\'d love, the law\'s delay,", "The insolence of office, and the spurns", "That patient merit of the unworthy takes,", "When he himself might his quietus make", "With a bare bodkin? who would these fardels bear,", "To grunt and sweat under a weary life,", "But that the dread of something after death,--", "The undiscover\'d country, from whose bourn", "No traveller returns,--puzzles the will,", "And makes us rather bear those ills we have", "Than fly to others that we know not of?", "Thus conscience does make cowards of us all;", "And thus the native hue of resolution", "Is sicklied o\'er with the pale cast of thought;", "And enterprises of great pith and moment,", "With this regard, their currents turn awry,", "And lose the name of action.--Soft you now!", "The fair Ophelia!--Nymph, in thy orisons", "Be all my sins remember\'d."};
   public WordCountData() {
   }
   public static DataSet<String> getDefaultTextLineDataset(ExecutionEnvironment env){
      return env.fromElements(WORDS);
   }
}
```

WordCountTokenizer.java用于切分句子，其内容如下：

```
package cn.edu.xmu;
import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.util.Collector;
public class WordCountTokenizer implements FlatMapFunction<String, Tuple2<String,Integer>>{
    public WordCountTokenizer(){}
    public void flatMap(String value, Collector<Tuple2<String, Integer>> out) throws Exception {
        String[] tokens = value.toLowerCase().split("\\W+");
        int len = tokens.length;
        for(int i = 0; i<len;i++){
            String tmp = tokens[i];
            if(tmp.length()>0){
                out.collect(new Tuple2<String, Integer>(tmp,Integer.valueOf(1)));
            }
        }
    }
}
```

WordCount.java提供主函数，其内容如下：

```
package cn.edu.xmu;
import org.apache.flink.api.java.DataSet;
import org.apache.flink.api.java.ExecutionEnvironment;
import org.apache.flink.api.java.operators.AggregateOperator;
import org.apache.flink.api.java.utils.ParameterTool;
public class WordCount {
   public WordCount(){}
   public static void main(String[] args) throws Exception {
      ParameterTool params = ParameterTool.fromArgs(args);
      ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
      env.getConfig().setGlobalJobParameters(params);
      Object text;
      //如果没有指定输入路径，则默认使用WordCountData中提供的数据
      if(params.has("input")){
         text = env.readTextFile(params.get("input"));
      }else{
         System.out.println("Executing WordCount example with default input data set.");
         System.out.println("Use -- input to specify file input.");
         text = WordCountData.getDefaultTextLineDataset(env);
      }
      AggregateOperator counts = ((DataSet)text).flatMap(new WordCountTokenizer()).groupBy(new int[]{0}).sum(1);
      //如果没有指定输出，则默认打印到控制台
      if(params.has("output")){
         counts.writeAsCsv(params.get("output"),"\n", " ");
         env.execute();
      }else{
         System.out.println("Printing result to stdout. Use --output to specify output path.");
         counts.print();
      }
   }
}
```

该程序依赖Flink Java API，因此，我们需要通过Maven进行编译打包。需要新建文件pom.xml，然后，在pom.xml文件中添加如下内容，用来声明该独立应用程序的信息以及与Flink的依赖关系：

```
<project>
  <groupId>cn.edu.xmu</groupId>
  <artifactId>simple-project</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>Simple Project</name>
  <packaging>jar</packaging>
  <version>1.0</version>
  <repositories>
    <repository>
      <id>jboss</id>
      <name>JBoss Repository</name>
      <url>http://repository.jboss.com/maven2/</url>
    </repository>
  </repositories>
  <dependencies>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-java</artifactId>
      <version>1.9.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-streaming-java_2.11</artifactId>
                  <version>1.9.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-clients_2.11</artifactId>
      <version>1.9.1</version>
    </dependency>
  </dependencies>
</project>
```

（3）使用Maven打包Java程序

为了保证Maven能够正常运行，先执行如下命令检查整个应用程序的文件结构：

```
$ cd ~/flinkapp
$ find .
```

文件结构应该是类似如下的内容：

```
.
./src
./src/main
./src/main/java
./src/main/java/WordCountData.java
./src/main/java/WordCount.java
./src/main/java/WordCountTokenizer.java
./pom.xml
```

接下来，我们可以通过如下代码将整个应用程序打包成**JAR**包（注意：计算机需要保持连接网络的状态，而且首次运行打包命令时，**Maven**会自动下载依赖包，需要消耗几分钟的时间）：

```
$ cd ~/flinkapp    #一定把这个目录设置为当前目录
$ /usr/local/maven/bin/mvn package
```

如果屏幕返回的信息中包含“**BUILD SUCCESS**”，则说明生成**JAR**包成功。

（4）通过flink run命令运行程序

最后，可以将生成的JAR包通过flink run命令提交到Flink中运行（请确认已经启动Flink），命令如下：

```
$ /usr/local/flink/bin/flink run --class
cn.edu.xmu.WordCount ~/flinkapp/target/simple-project-
1.0.jar
```

执行成功后，可以在屏幕上看到词频统计结果。