Flink:

# 什么是Flink

# [Flink 应用场景](https://flink.apache.org/usecases.html)

* 1. flink功能

support for stream and batch processing 对流和批处理的支持

sophisticated state management 复杂状态管理

event-time processing semantics 事件-时间处理语义

exactly-once consistency guarantees for state 状态的仅此一次的持续性保证

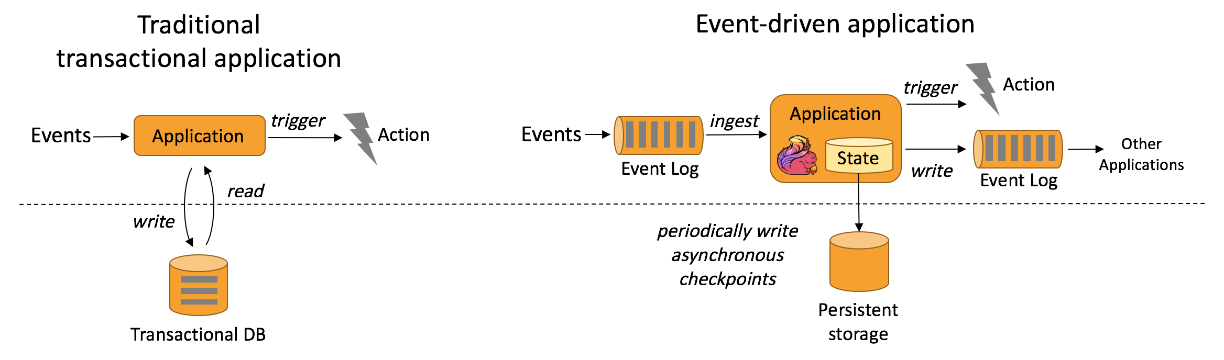
* 1. Flink 最常见的应用场景：
     1. Event-driver Application (事件驱动)

1. 什么是事件驱动型应用

事件驱动型应用是一种有状态的应用，他从一个或多个事件流提取数据，并根据提取到来的数据触发计算、状态更新或者外部action。

事件驱动型应用是一次计算和数据存储层分离的传统应用设计的演变。在此架构中，应用程序能从远程传统交易数据库中读取数据也能将数据持久化到传统库中。

相反，事件驱动型应用是基于有状态的流处理应用，在此设计中，数据和计算协同定位，这将产生局部的数据（基于内存或磁盘）访问。通过阶段性地将将 将checkpoint写入远程持久化存储来实现容错。以下图片说明了传统的应用程序体系结构和事件驱动型应用之间的区别。



1. 事件驱动型应用的优势

本地数据访问使其具有更高的吞吐和更低的延迟

定期地异步、增量式远程持久化checkpoint存储，对正常事件处理影响较小。

减少数据的改变或服务扩容时的协调工作

1. Flink如何支持事件驱动型应用程序的？

事件驱动型应用会受制于底层流处理系统对时间和状态的把控能力。

* + 1. Data Analytics Application（数据分析）
    2. Data Pipeline Application （）

Event-driver Application (事件驱动)

欺诈检测

异常检测

基于规则的告警

业务流程监控

Web应用程序（社交网络）

CEP：复杂事件处理

Data Analytics Application（数据分析）

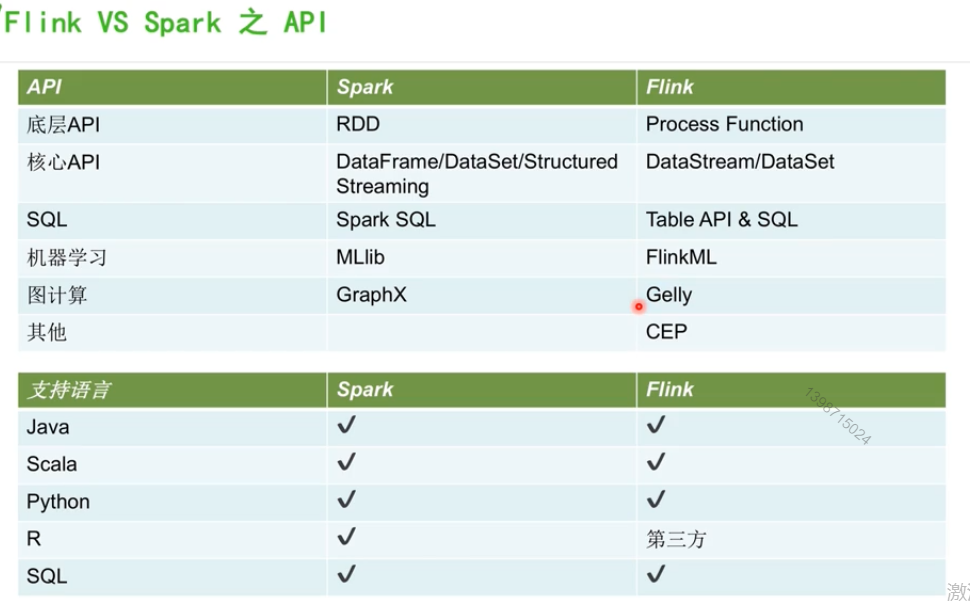
周期性查询

连续性查询

Data Pipeline Application （）

Periodic ETL

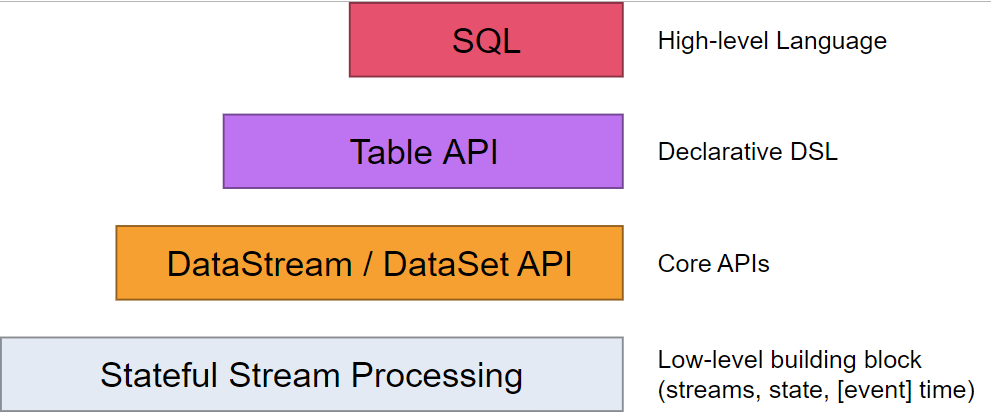
Data Pipeline



# [Flink docs](https://ci.apache.org/projects/flink/flink-docs-release-1.9/)

**Concepts:**

1. Dataflow Programming Model
   1. Levels of Abstraction

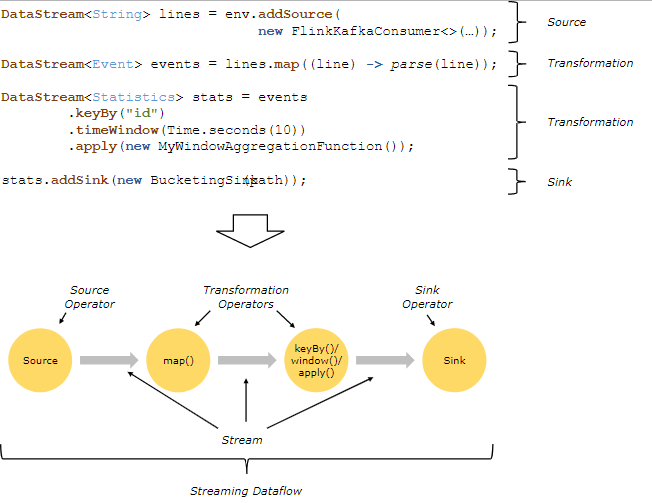


* 1. Programs and Dataflows

流和转换是Flink的基石（请注意：flink的DataSet API中使用的DataSets其内部也是流）。概念上，stream就是数据记录流，transformation 就是一个把一个或多个输入流streams作为输入，然后把产生的一个或多个输出流作为结果的算子。

程序执行时，Flink程序被映射成streaming dataflows，streaming dataflows 由 streams 和transformation 算子组成。每个dataflow以一个或多个source开始而以一个或多个sink结束。Dataflow有点类似于任意的有向无环图（DAG图）。尽管环的特定形式是迭代结构，但是，为了简单起见，大多数情况下我们掩盖这种情况。

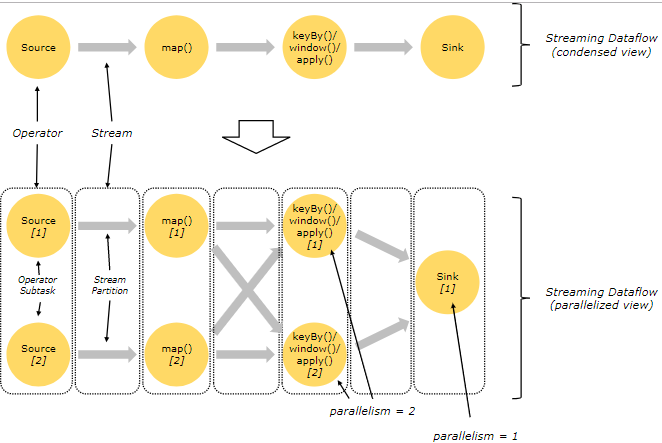
通常程序的转换和数据流间的算子是意义对应的。然而，有时，一个转换也有可能有多个转换算子组成。



* 1. Parallel Dataflows

Flink程序的本质还是并行和分布式，在程序运行时，一个stream有一个或多个流分区（stream partitions），每个operator算子也有一个或多个operator子任务（operator subtasks）。算子的subtasks 间彼此独立，并可能在不同的机器或容器上执行不同的线程。

算子的subtasks的数量就是对应算子的并行度。而一个stream的并行度也取决于产生它的算子。同一个程序的不同算子也可能有不同程度的并行度



流可以在两个算子间以一对一（或转发）或者重新分配的模式传输数据，

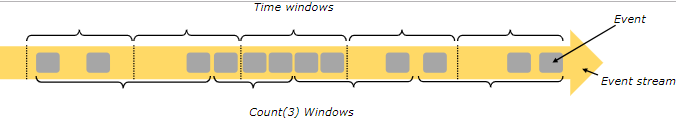
**One-to-one** streams (上图中source 和 map(之间就是one-to-one streams):保持了元素的分区和顺序，这意味着，subtask[1]的map()算子的将看到相同的顺序相同的元件，因为它们是由子任务[1]的源操作者产生的

**Redistributing** streams:

* 1. Windows

流和批处理的event聚合在运行上还是有些差异的。比方说，你不可能对一个stream的所有元素求和，因为stream通常是无界的。但是，stream的聚合（count, sum 等）可以定义在窗口上（windows）,就像对最近5min求count 或者最近100条记录求sum等。

窗口（windows）可以由时间（e.g:每30秒）来驱动也可以有事件（每100条事件）来驱动。我们也可以区分不同窗口之间的差异，例如滚动窗口（无重叠）、滑动窗口（有重叠） 和session窗口（以非活跃的界限作为标记）。

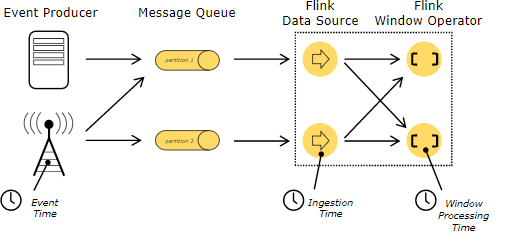


* 1. Time

**Event Time:**即事件产生时的时间。在event中通常以时间戳描述，例如可由传感器或生产服务产生。Flink 通过[timestamp assigners](https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/event_timestamps_watermarks.html)来访问时间时间戳。

**Ingestion time** ：一条事件通过source operator 进入flink.数据流的时间。

**Processing Time** 每一个算子执行基于时间的操作的本地时间。



* 1. Stateful Operations
  2. Checkpoints for Fault Tolerance
  3. Batch on Streaming

Flink以流处理的特殊形式来执行批处理程序，这种情况下stream就是有界的（元素的个数有限）。DataSet在其内部以数据流的形式进行处理，因此，以上的概念同样适用于以流处理来进行批处理的处理方案，除了少数例外：

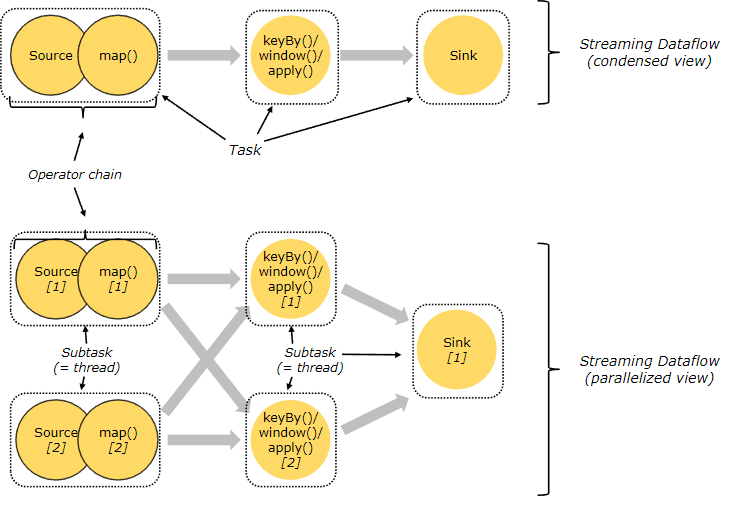
[Fault tolerance for batch programs](https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/batch/fault_tolerance.html) ：批处理程序不使用检查点（checkpoint）的容错。由于输入的有限性，通过重新执行流来恢复是可以的。这一点在程序恢复上开销较大，但是使常规处理更简单，因为它不需要检查点（checkpoint）。

DataSet API中的stateful operations 都是通过使用简单的内存数据结构或者核外数据结构实现，而不是键/值索引。

DataSet API介绍了特殊的同步迭代操作（superstep-based），这一点仅限于有界流。

1. Distributed Runtime Environment
   1. Tasks and Operator Chains

在分布式运行环境中，flink 将operator subtasks和 tasks 关联到一起。一个task由一个线程执行，将operators 和task关联起来是一项优化任务，既降低了交互和缓存的开销，总体上降低延迟的同时也提高了吞吐量。



* 1. Job Managers , Task Managers , Clients

Flink Flink运行时包含两种类型的进程

JobManagers (also called masters):协调分布式的运行，主要包含：task调度，checkpoint协调，以及协调故障恢复等。

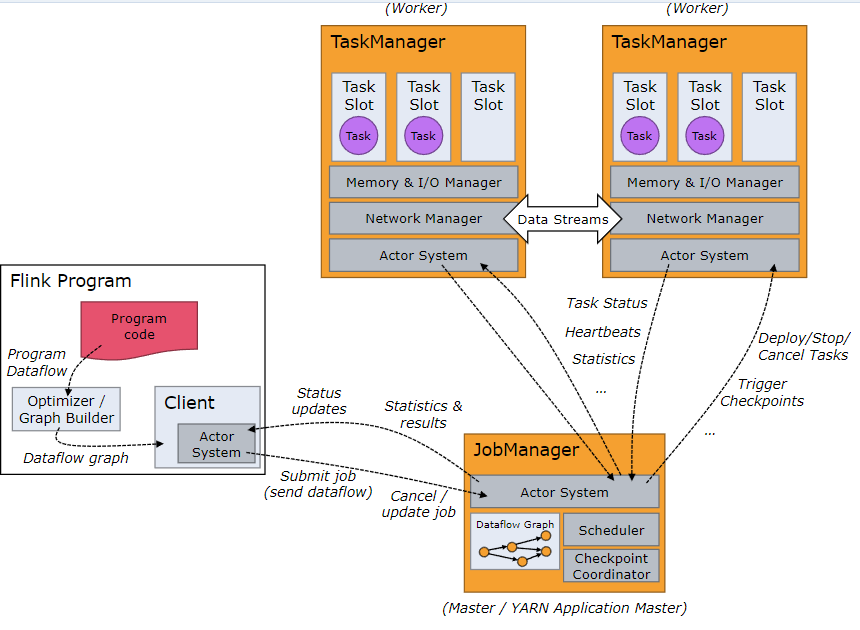
运行时环境中至少有一个Job Manager，在HA环境中可以设置有多个JobManager，其中只有一个是leader，其他都是standby。

TaskManagers (also called workers) ：执行数据流的task(更具体的说是subtask)，缓冲和交换数据流。

至少有一个TaskManagers；

JobManager和TaskManager可以通过多种方式启动：作为 [standalone cluster](https://ci.apache.org/projects/flink/flink-docs-release-1.9/ops/deployment/cluster_setup.html)直接在计算机上，在容器中启动，或者由诸如YARN或Mesos的资源框架进行管理。TaskManager连接到JobManager，宣布自己可用，并被分配工作

客户端不是运行时和程序执行的一部分，而是用于准备数据流并将其发送到JobManager的。或保持联系以接收进度报告。客户端要么作为触发执行的Java / Scala程序的一部分运行，要么在命令行过程中运行./bin/flink run ...



* 1. Task Slots and Resources

每个 worker（TaskManager）都是一个 JVM 进程，并且可以在不同的线程中执行一个或多个 subtasks。为了控制 worker 接收 task 的数量，worker 拥有所谓的 **task slots** （至少一个）。

* 1. State Backends
  2. Savepoints