Flink流处理简介

主要内容

- Flink是什么
- 为什么要用Flink
- 流处理的发展和演变
- Flink的主要特点
- Flink vs Spark Streaming

Flink是什么

Apache Flink是一个框架和**分布式**处理引擎，用于对**无界**和**有界**数据**流**进行**状态**计算。

Flink 目前在国内企业的应用

所有大厂都在**重度**使用Flink。特别是阿里，收购了Flink母公司，并为Flink贡献了海量的代码，双十一大屏的指标全部使用Flink计算。

为什么选择 Flink

- 流数据更真实地反映了我们的生活方式
- 传统的数据架构是基于有限数据集的
- 我们的目标
    - 低延迟（Spark Streaming 的延迟是秒级，Flink 延迟是毫秒级，由于操作系统的时钟的精度是毫秒级，所以可以认为Flink是没有延迟的）
    - 高吞吐
    - 结果的准确性和良好的容错性（EXACTLY-ONCE）

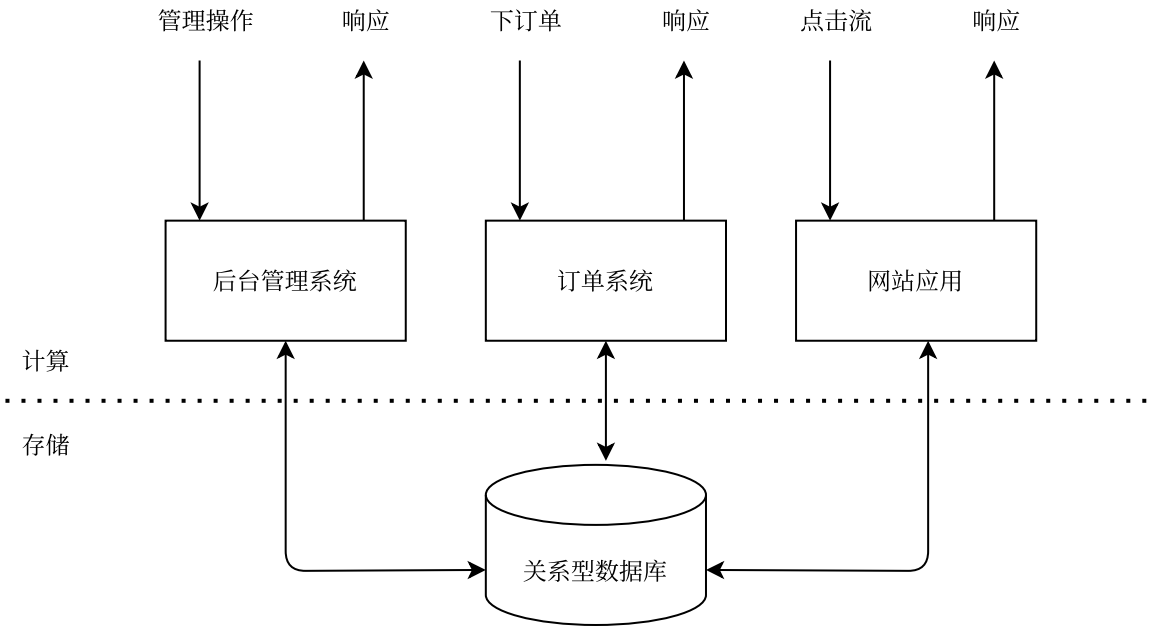
哪些行业需要处理流数据

- 所有行业都需要处理流数据，因为数据本质上是流的形式。
- 电商：计算PV（Page View）、UV（Unique Visitor），以及实时热门商品（每过5分钟计算一次过去1小时的PV最多的商品）等指标。
- 物联网：温度传感器连续1秒钟温度上升的检测。
- 风控：连续三次登录失败的检测，信用卡欺诈检测（第一笔消费小于1元，第二笔消费大于500元），超时未支付订单的检测

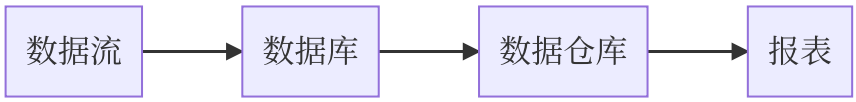
传统数据处理架构

- OLTP（在线事务处理）
- OLAP（在线分析处理）
- LAMBDA架构

OLTP

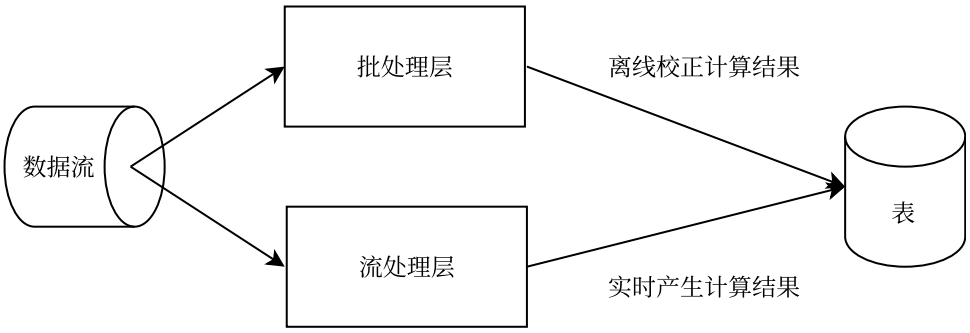


OLAP

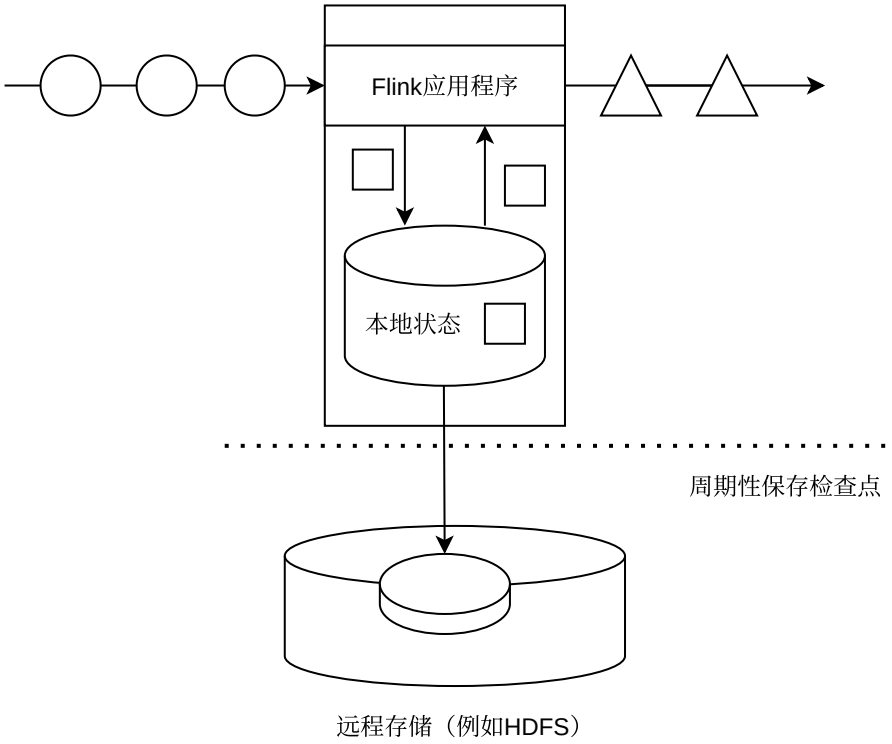


LAMBDA架构

- 用两套系统，同时保证低延迟和结果准确



有状态的流处理



流处理的演变

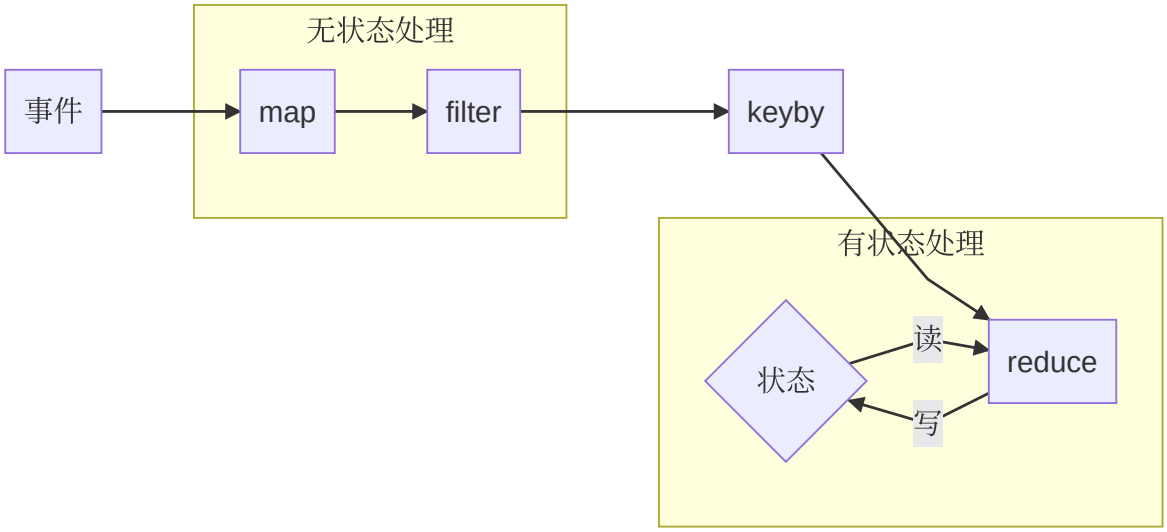
- Apache Storm：低延迟
- Apache Spark Stream：高吞吐
- Apache Flink：低延迟，高吞吐，时间正确/语义化窗口，计算结果的正确性（EXACTLY-ONCE）

Flink的主要特点

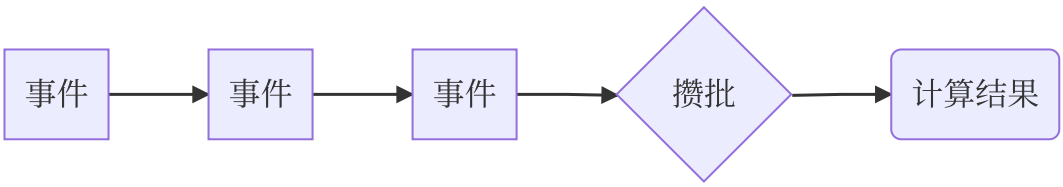
- 事件驱动
- 基于流的世界观
- 分层API
- 支持事件时间（EventTime）和处理时间（ProcessingTime）语义
- 精确一次（EXACTLY-ONCE）的状态一致性保证
- 低延迟，每秒处理数百万个事件，毫秒级延迟
- 与众多常用存储系统的连接（ES，HBase，MySQL，Redis...）
- 高可用（Zookeeper），动态扩展，实现7*24小时全天候运行

事件驱动

- 来一条数据就处理一次，每来一条数据就会驱动DAG中算子的运行，也可以看作数据在DAG里面流动。

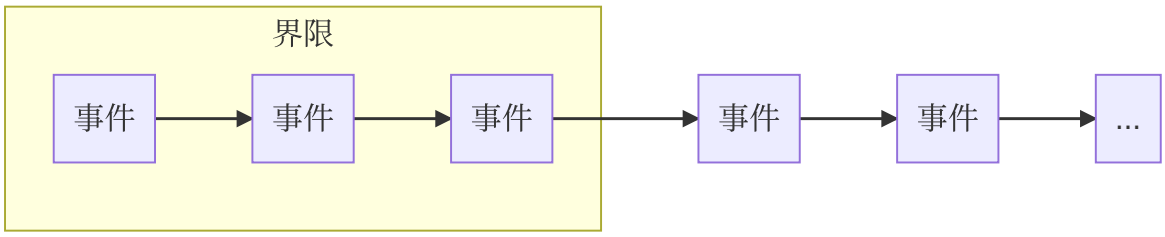


- 事件到达之后立即驱动`MAP`的运行，`MAP`处理完事件之后，将`ETL`后的数据发送给`FILTER`算子，就会立刻驱动`FILTER`算子的运行，依次类推。
- 由于Flink是有状态的流处理，所以可能会有算子会维护和操作内部状态，例如`REDUCE`算子。而`MAP`和`FILTER`是无状态的计算。
- 传统批处理示意图如下：

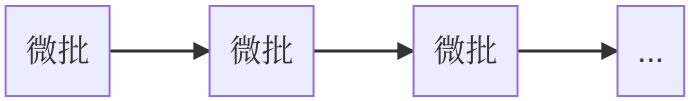


基于流的世界观

- 在 Flink 的世界观中，一切都是由流组成的，离线数据是有界的流；实时数据是一个没有界限的流：这就是所谓的有界流和无界流。

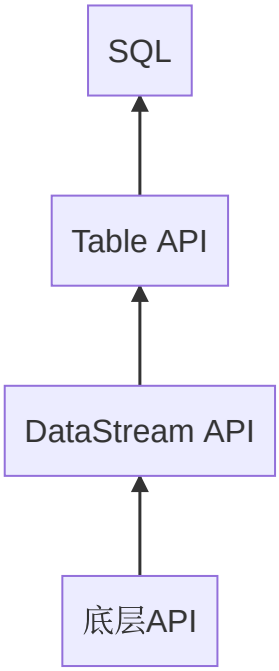


- 在Spark Streaming的世界观中，一切都是由批组成的，离线数据是一批数据；实时数据是无数个微小批次组成的数据。



- 流的世界观最终要的一点其实是在静态的离线数据上面加了一个**维度：时间**
- 这个观点来自爱因斯坦的狭义相对论，批处理类似牛顿力学（坐标系：`x`，`y`，`z`），流处理类似狭义相对论力学（坐标系：`x`，`y`，`z`，`t`）。

分层API



- 我们的学习重点：底层API，DataStream API
- DataStream API对应Spark中RDD的各种算子
- 底层API在Spark中无对应，是学习的**难点**，主要是处理函数，用来处理时间语义和状态。

Flink中最重要的三个核心概念

我们在学习Flink时，只需要彻底理解下面三个概念，就能够很好的理解Flink的本质：

- 时间语义：事件时间，逻辑时钟（水位线）
- 状态：分清有状态和无状态的区别
- 事件驱动：来一条数据处理一次

分布式系统重要概念

- 分区：物理分区和逻辑分区的区别是什么
- 时钟：物理时钟和逻辑时钟的区别是什么
- 并行和并发的区别

函数式编程的重要概念

- 不可变数据结构（所以不能编写循环，只能使用递归）
- 无状态和有状态
- 惰性求值、懒加载（编写Flink程序只是定义好DAG，然后扔到集群中去，在需要的时候执行）
- 副作用（赋值、异常）
- 函数作为一等公民

Flink vs Spark Streaming

- 流 vs 微批
- 数据模型
  - Spark：RDD，Spark Streaming 的 DStream 实际上也就是一组组小批数据 RDD 的集合。
  - Flink 基本数据模型是数据流，以及事件（Event）序列（Integer、String、Long、POJO Class）
- 运行时架构
  - Spark是批计算，将DAG划分为不同的Stage，一个Stage完成后才可以计算下一个Stage。
  - Flink是标准的流执行模式，一个事件在一个节点处理完后可以直接发往下一个节点进行处理。
- Spark Streaming的延迟是Flink的1000倍。
- Flink支持事件时间

单词计数程序

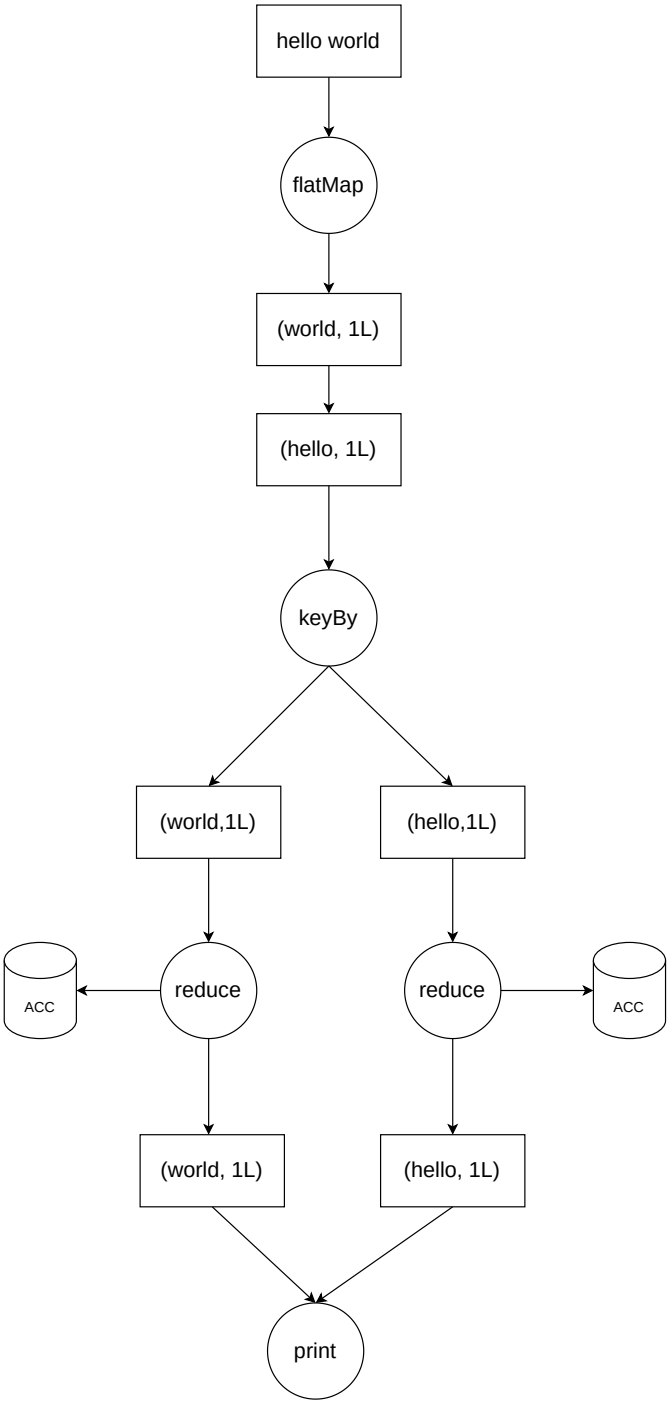
Flink程序典型结构

1. 获取流执行环境
2. 设置并行任务的数量
3. 读取数据源
4. 进行计算
5. 输出
6. 执行程序

算子的语义

- map的语义：针对流中的每一个元素，输出一个元素
- flatMap的语义：针对流中的每个元素，输出0个、1个或者多个元素
- filter的语义：针对流中的每个元素，输出0个或者1个元素
- reduce的语义：reduce会初始化一个空累加器（类型和流中的元素类型相同），第一条元素到来，直接作为累加器保存，并将累加器输出。第二条以及之后的元素到来，和累加器进行累加操作并更新累加器，然后将累加器输出。reduce函数定义的是输入元素和累加器的累加规则。

单词计数程序示意图


hello world
flatMap
(world, 1L)
(hello, 1L)
keyBy
(world,1L)
(hello,1L)
ACC
reduce
reduce
ACC
(world, 1L)
(hello, 1L)
print

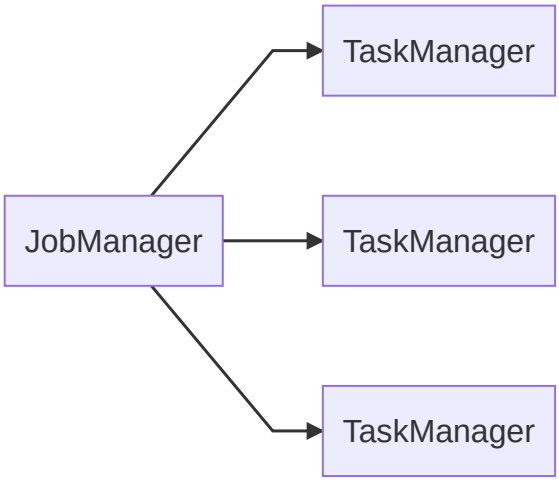
Flink运行架构

主要内容

- Flink 运行时的组件
- 任务提交流程
- 任务调度原理

Flink架构

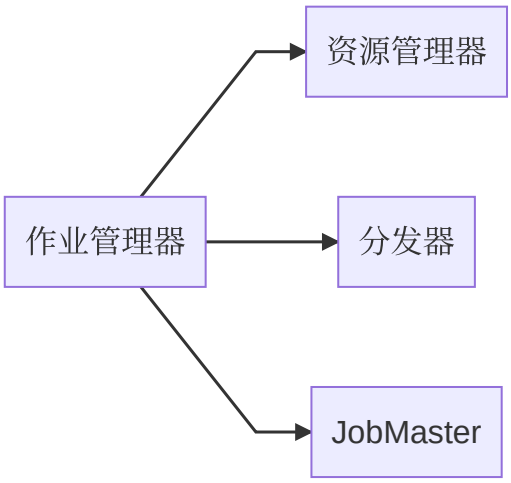
- Flink 运行时由两种类型的进程组成：一个JobManager（作业管理器）和一个或者多个TaskManager（任务管理器）。
- 典型的Master-Slave架构。



作业管理器

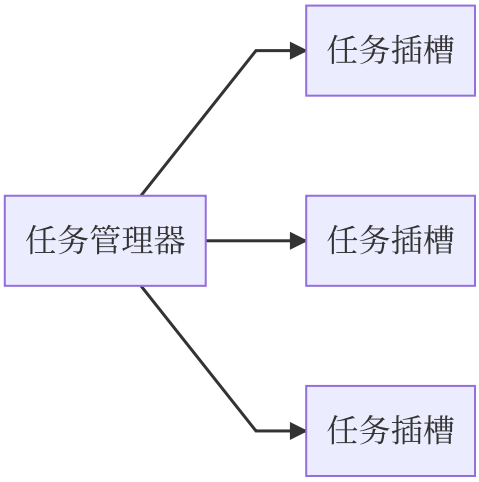
三类线程

- 资源管理器：资源是**任务槽**
- 分发器（WebUI）
- JobMaster（每个作业对应一个）



任务管理器

- 任务插槽（task slot）
- 不同的任务插槽就是不同的物理分区
- 每个任务插槽是一个内存分片



并行度的设置

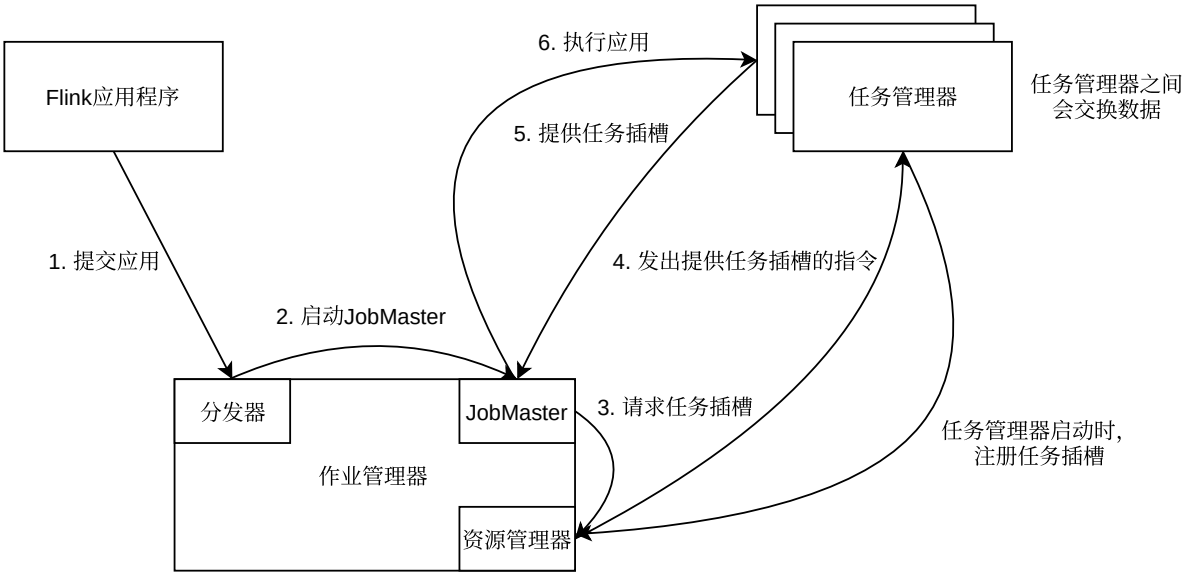
从上到下，优先级升高

1. 任务管理器的配置文件里面
2. 在命令行提交任务时指定并行度
3. 全局并行度
4. 针对算子设置并行度

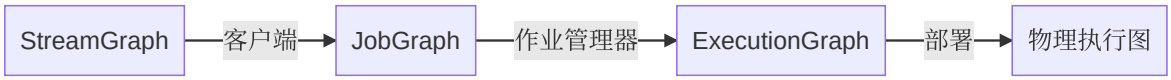
并行度设置的最佳实践

1. 不要设置全局并行度
2. 针对某些算子设置并行度，例如数据源，为了不改变数据的顺序
3. 在命令行设置，可以动态扩容

任务提交流程


Flink应用程序
6. 执行应用
任务管理器
任务管理器之间
会交换数据
5. 提供任务插槽
1. 提交应用
4. 发出提供任务插槽的指令
2. 启动JobMaster
分发器
JobMaster
3. 请求任务插槽
任务管理器启动时,
注册任务插槽
作业管理器
资源管理器

不同的DAG结构



自定义数据源

- SourceFunction: 并行度只能设置为1
- ParallelSourceFunction: 并行自定义数据源

基本转换算子

无状态的算子

- map
- filter
- flatMap

同步和异步

- 同步：事件执行顺序是确定的
- 异步：事件执行顺序是不确定的