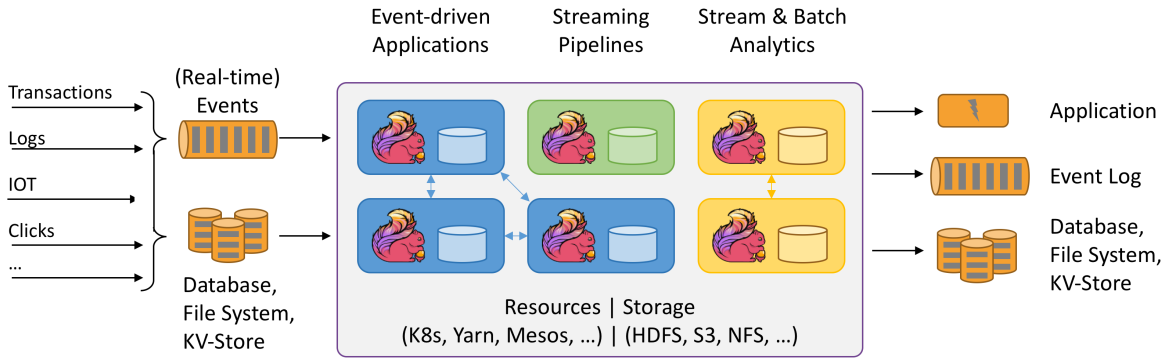
Apache Flink基础及架构

Objective(本课目标)

- [x] 安装Flink集群
- [x] 理解Flink的基本原理
- [x] 掌握Flink的编程模型
- [x] 掌握Flink的资源管理

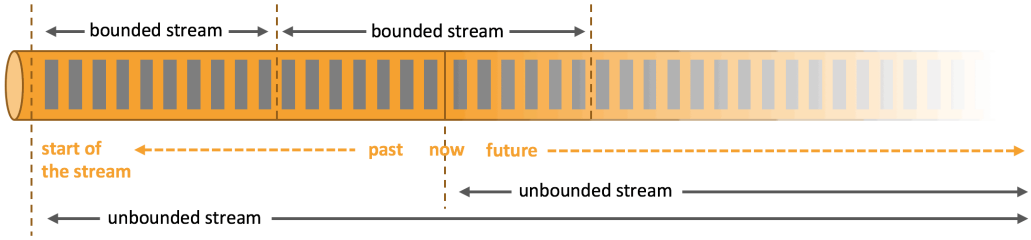
1.Apache Flink简介

- Apache Flink® — Stateful Computations over Data Streams（有状态的数据流）
- Apache Flink 是一个分布式处理引擎框架，用于在无边界和有边界数据流上进行有状态的计算。Flink能在所有常见集群环境中运行，并能以内存速度和任意规模进行计算。
- Events -> 一般指的是消息系统（kafka，roketMQ,activeMQ），偏向于实时数据处理
- DB,FS,Nosql -> 偏向于数据的批处理
- SparkStreaming -> 批处理
- Flink -> 完全的实时流数据处理



2.特性1-处理无界和有界数据

- 任何类型的数据都可以形成一种事件流。信用卡交易、传感器测量、机器日志、网站或移动应用程序上的用户交互记录，所有这些数据都形成一种流。
- 数据可以被作为 无界 或者 有界 流来处理。
  - 无界流 有定义流的开始，但没有定义流的结束。它们会无休止地产生数据。无界流的数据必须持续处理，即数据被摄取后需要立刻处理。我们不能等到所有数据都到达再处理，因为输入是无限的，在任何时候输入都不会完成。处理无界数据通常要求以特定顺序摄取事件，例如事件发生的顺序，以便能够推断结果的完整性。

    - 有界流 有定义流的开始，也有定义流的结束。有界流可以在摄取所有数据后再进行计算。有界流所有数据可以被排序，所以并不需要有序摄取。有界流处理通常被称为批处理
    - 

- Apache Flink 擅长处理无界和有界数据集，精确的时间控制和状态化使得 Flink 的运行时(runtime)能够运行任何处理无界流的应用。有界流则由一些专为固定大小数据集特殊设计的算法和数据结构进行内部处理，产生了出色的性能。

3.特性2-部署应用到任意地方

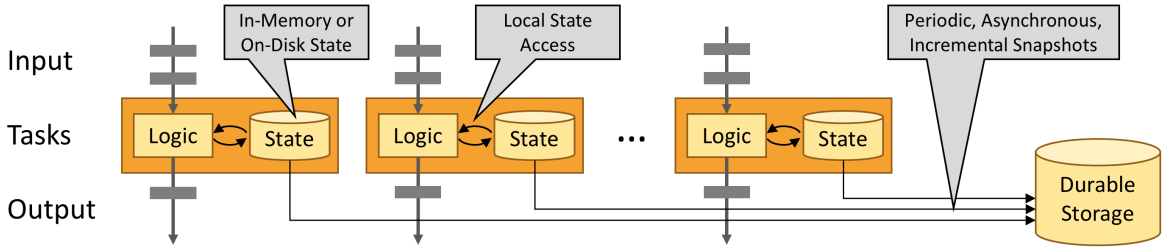
- Apache Flink 是一个分布式系统，它需要计算资源来执行应用程序。Flink 集成了所有常见的集群资源管理器，例如 Hadoop YARN、 Apache Mesos 和 Kubernetes，但同时也可以作为独立集群运行（Standalone模式）。Flink 被设计为能够很好地工作在上述每个资源管理器中，这是通过资源管理器特定(resourcemanager-specific)的部署模式实现的。Flink 可以采用与当前资源管理器相适应的方式进行交互。部署 Flink 应用程序时，Flink 会根据应用程序配置的并行性自动标识所需的资源，并从资源管理器请求这些资源。在发生故障的情况下，Flink 通过请求新资源来替换发生故障的容器。提交或控制应用程序的所有通信都是通过 REST 调用进行的，这可以简化 Flink 与各种环境中的集成

4.特性3-运行任意规模应用

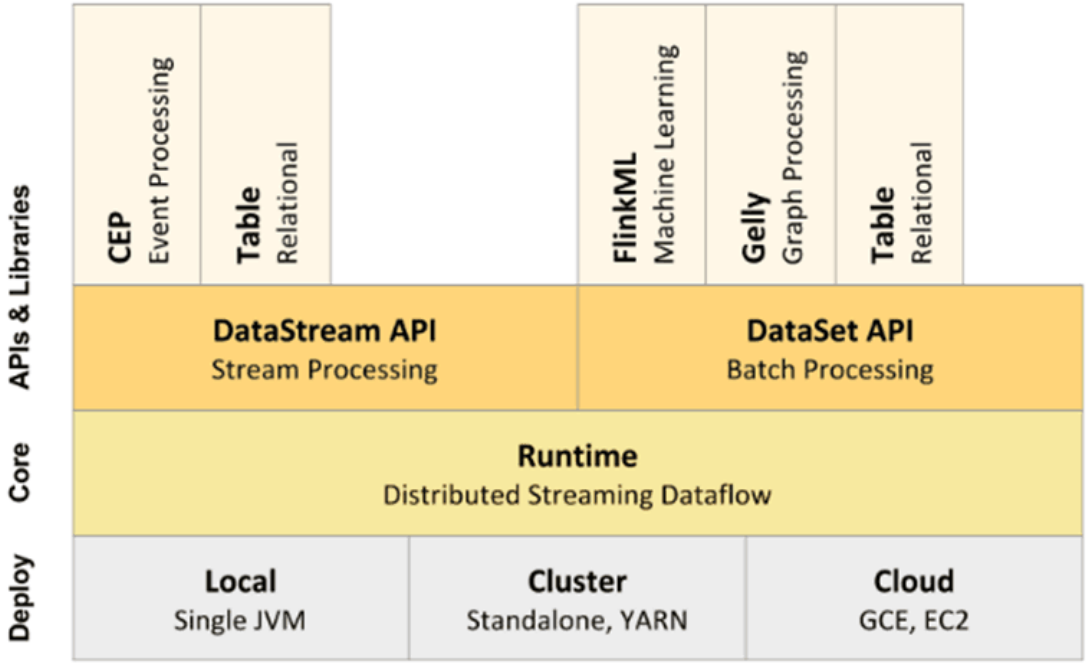
- Flink 旨在任意规模上运行有状态流式应用。因此，应用程序被并行化为可能数千个任务，这些任务分布在集群中并发执行。所以应用程序能够充分利用无尽的 CPU、内存、磁盘和网络 IO。而且 Flink 很容易维护非常大的应用程序状态。其异步和增量的检查点算法对处理延迟产生最小的影响，同时保证精确一次状态的一致性。
- Flink 用户报告了其生产环境中一些令人印象深刻的扩展性数字：
    - 每天处理数万亿的事件
    - 可以维护几TB大小的状态
    - 可以部署上千个节点的集群

5.特性4-利用内存性能

- 有状态的 Flink 程序针对本地状态访问进行了优化。任务的状态始终保留在内存中，如果状态大小超过可用内存，则会保存在能高效访问的磁盘数据结构中。任务通过访问本地（通常在内存中）状态来进行所有的计算，从而产生非常低的处理延迟。Flink 通过定期和异步地对本地状态进行持久化存储来保证故障场景下精确一次的状态一致性。



6.Flink架构图

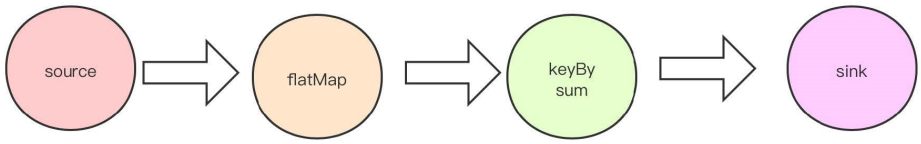


7.案例演示

```
见课堂案例
```

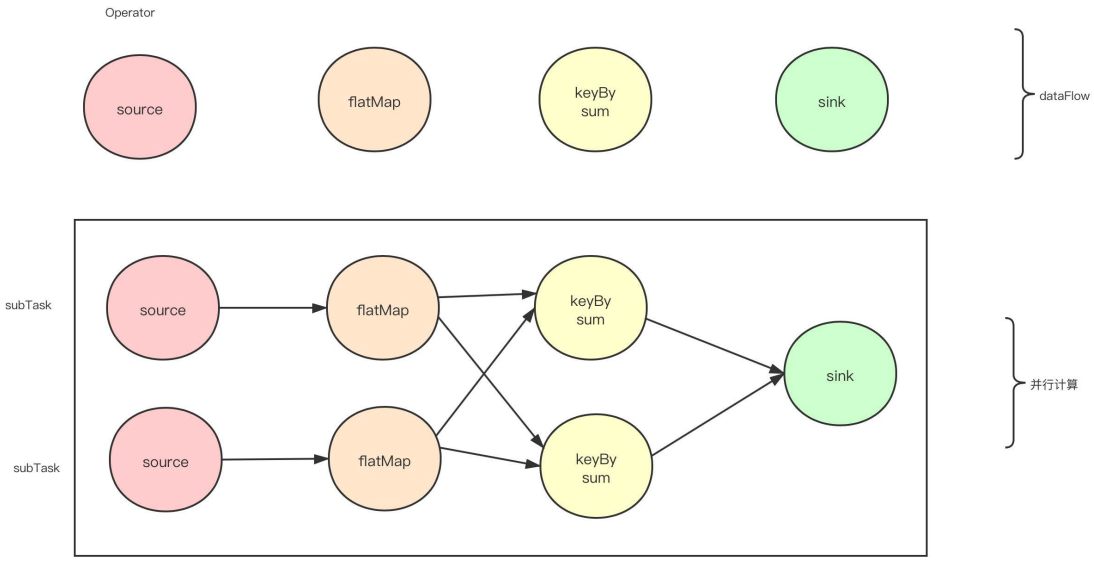
8.Flink核心概念

- Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale.（Apache Flink是一个框架和分布式处理引擎，用于在无界和有界数据流上进行有状态计算。Flink被设计成可以在所有常见的集群环境中运行，以内存速度和任何规模执行计算）
- 有状态：每一步计算都是有状态的，会根据前面的状态进行累加



8.1 分布式

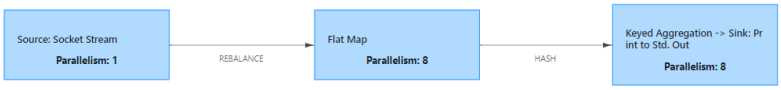
- 算子(`Operator`)将一个或多个 `DataStream` 转换为新的 `DataStream` 。程序可以将多个转换组合成复杂的数据流拓扑



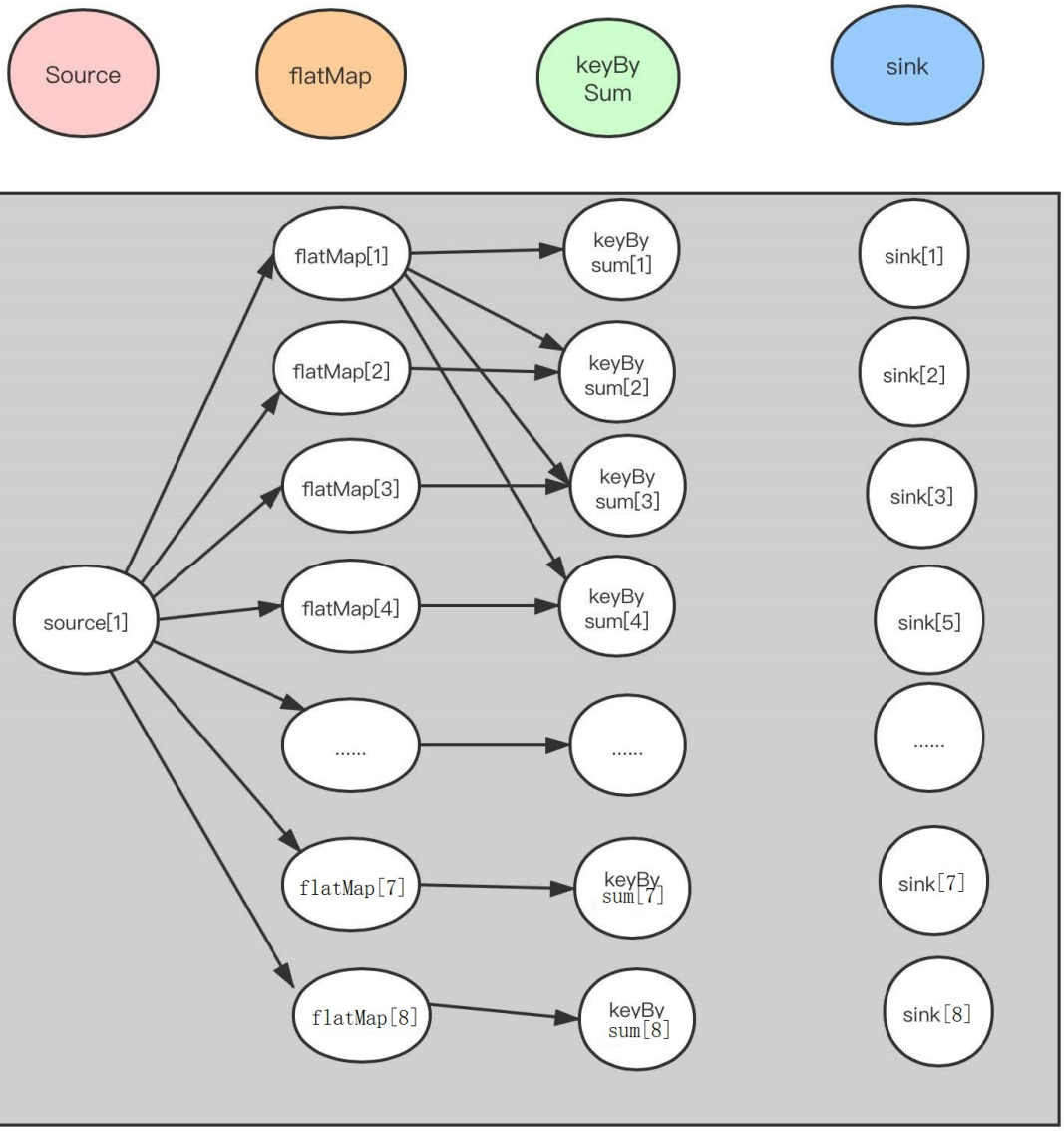
8.2 并行度

```
#案例6，使用默认并行度和设置并行度
/**
* 单词计数:计算并行度
  1. 使用默认并行度
  2. 自定义并行度为2
* 打开页面: http://localhost:8081
*/
public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.createLocalEnvironmentWithWebUI(new Configuration());

//        env.setParallelism(2);
```

```
        ParameterTool parameterTool = ParameterTool.fromArgs(args);

        String hostname = parameterTool.get("hostname");
        int port = parameterTool.getInt("port");

        /**
         * socket无论如何就只能有一个并行度
         *
         */
        DataStreamSource<String> myDataStream = env.socketTextStream(hostname,
port);


        SingleOutputStreamOperator<WordAndOne> result = myDataStream.flatMap(new
StringSplitTask())
                .keyBy("word")
                .sum("count");

        result.print();
        env.execute("word count...");
    }
```

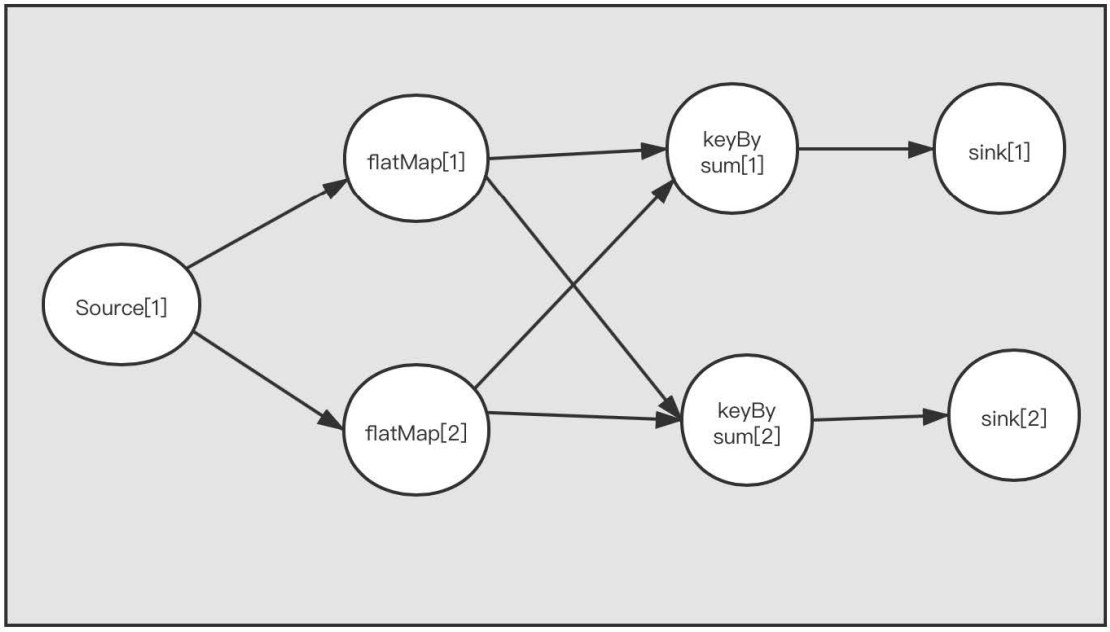


Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Tasks
Source: Socket Stream	RUNNING	0 B	0	0 B	0	1	1
Flat Map	RUNNING	0 B	0	0 B	0	8	8
Keyed Aggregation -> Sink: Print to Std. Out	RUNNING	0 B	0	0 B	0	8	8



设置并行度为2：env.setParallelism(2);

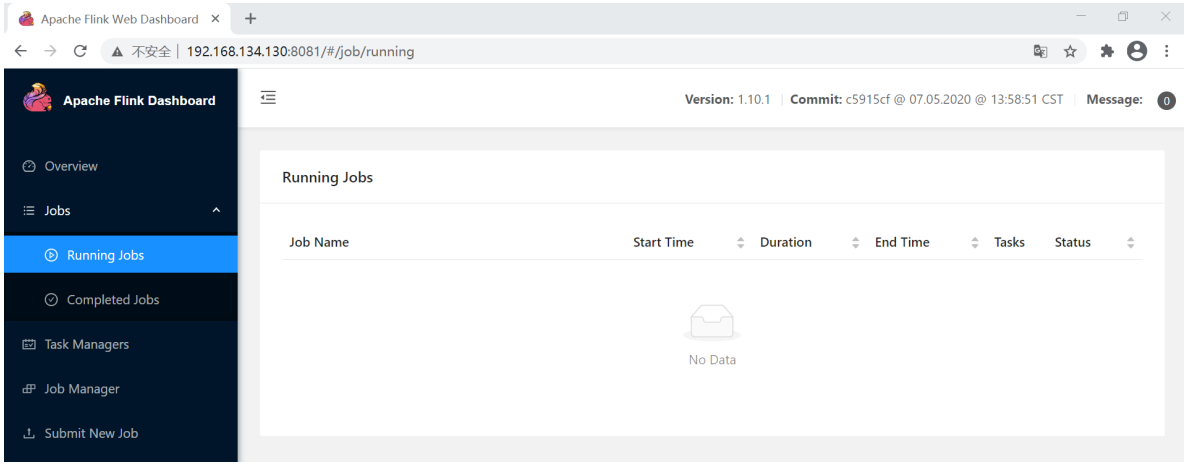
Name	Status	Bytes Received	Records Received	Bytes Sent	Tasks
Source: Socket Stream	RUNNING	⌣	⌣	⌣	1
Flat Map	RUNNING	⌣	⌣	⌣	2
Keyed Aggregation -> Sink: Print to Std. Out	RUNNING	⌣	⌣	⌣	2



9.集群模式和本地模式安装

9.1 Local模式安装（无需任何配置）

```
#安装jdk，配置JAVA_HOME，建议使用jdk1.8以上

#安装包下载地址： https://www.apache.org/dyn/closer.lua/flink/flink-1.10.1/flink-1.10.1-bin-scala_2.11.tgz

#解压安装包：tar -zxvf f flink-1.10.1-bin-scala_2.11.tgz

#配置FLINK_HOEM环境变量

#启动服务()
  start-cluster.sh 启动服务
  stop-cluster.sh 停止服务

#Web页面浏览 http://hadoop1:8081
```



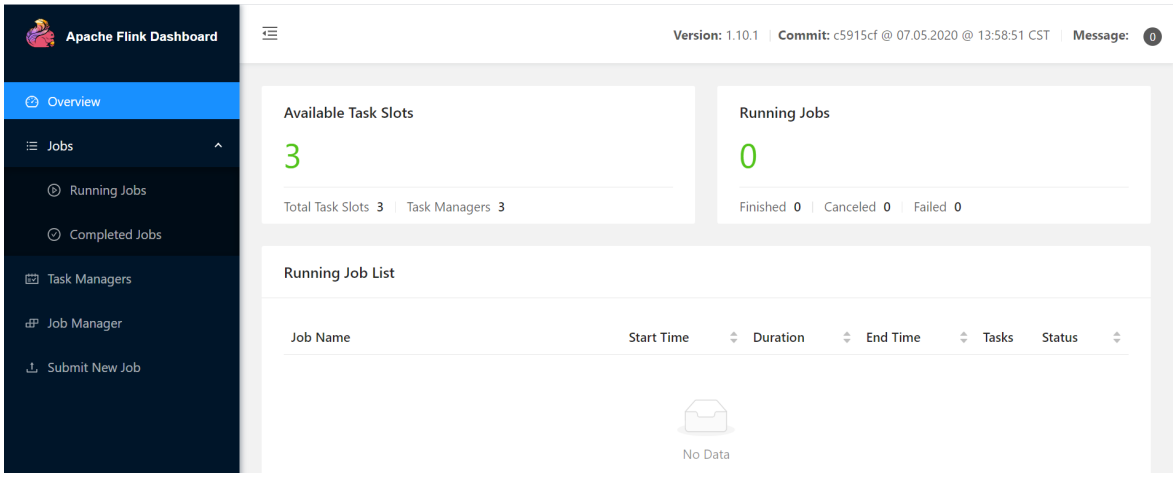
9.2 集群模式安装

1.集群规划

主机名	JobManager	TaskManager
Hadoop1	yes	yes
Hadoop2		yes
hadoop3		yes

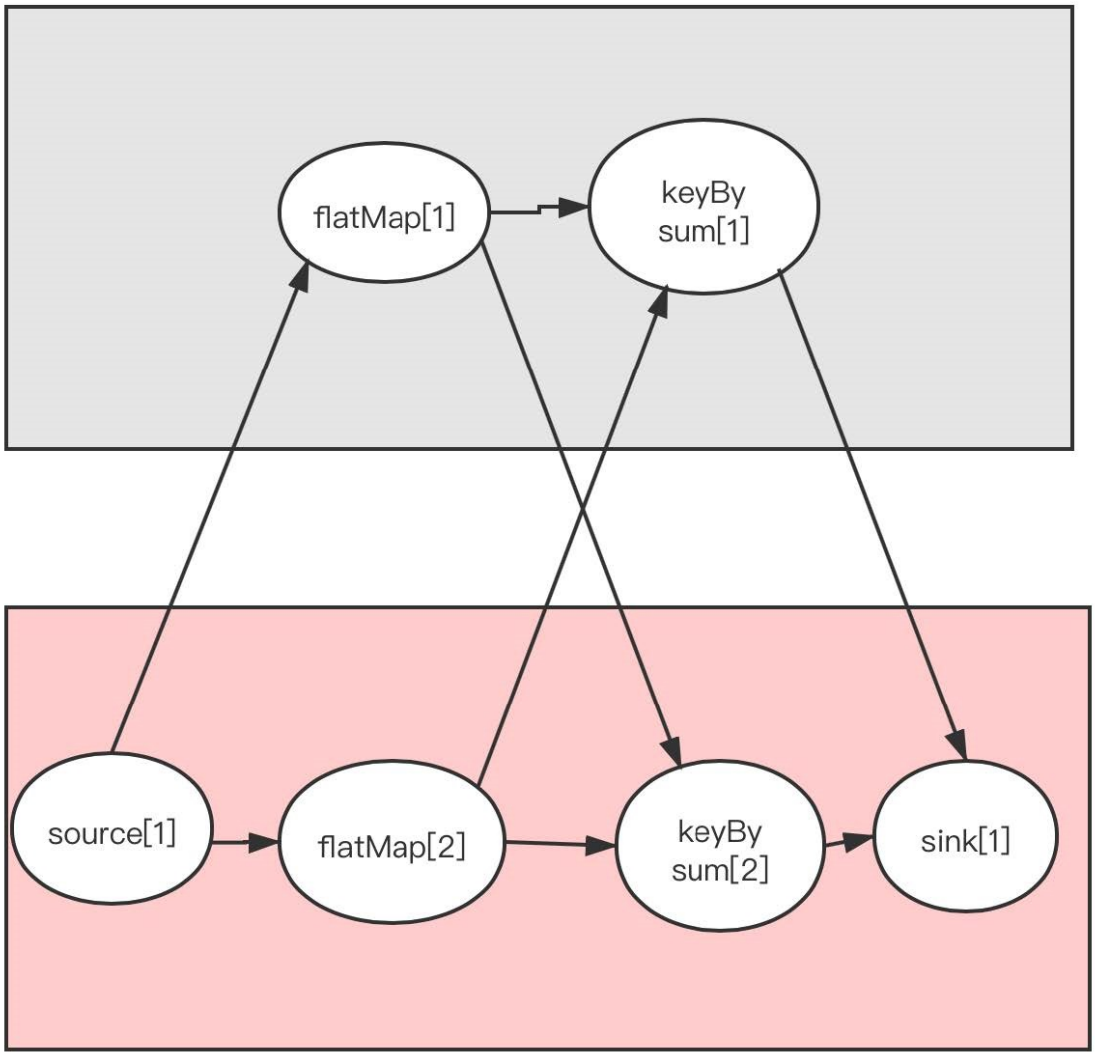
2.安装步骤

```
#前置要求，jdk1.8以上，配置JAVA_HOME

#安装步骤
1.修改conf/flink-conf.yaml jobmanager.rpc.address: hadoop1

2.修改conf/slaves
hadoop1
hadoop2
hadoop3

3.拷贝到其他节点
scp -r flink-1.10.1 hadoop2:/root/install
在hadoop1节点启动：start-cluster.sh
访问http://hadoop1:8081


4.优化StandAlone模式需要考虑的参数
jobmanager.heap.mb：jobmanager节点可用的内存大小
taskmanager.heap.mb：taskmanager节点可用的内存大小
taskmanager.numberOfTaskSlots：每台机器可用的cpu数量
parallelism.default：默认情况下任务的并行度
taskmanager.tmp.dirs：taskmanager的临时数据存储目录
```



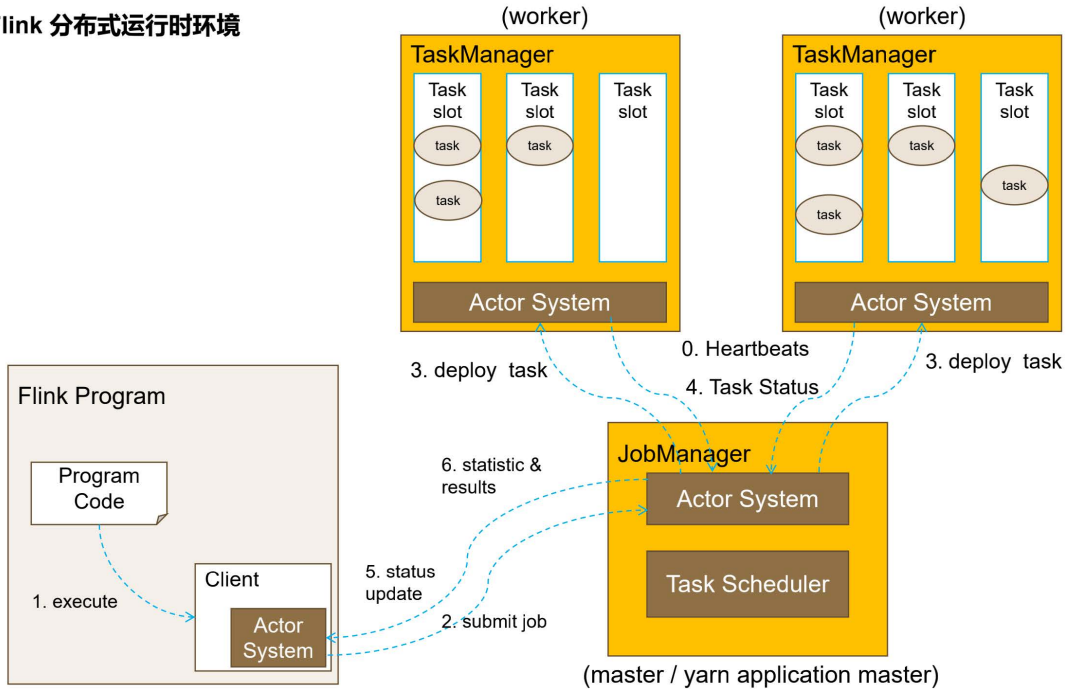
• 10.分布式运行环境

```
(1) 运行
在hadoop1 上执行 nc -lk 8888
flink run -c com.bw.flinkstreaming.job5.WordCount flink.jar --hostname hadoop1 --
port 8888

(2) 停止任务
方式一：页面上停止
方式二：命令停止
flink cancel job-id

#案例1讲解
提交：flink run -c com.bw.flinkstreaming.job7.WordCount flink.jar --hostname
hadoop1 --port 9999
该案例运行结果？ 1 2 2 1

#案例3讲解，如果设置并行度都为1那么总共需要几个task
提交：flink run -c com.bw.flinkstreaming.job9.WordCount flink.jar --hostname
hadoop1 --port 9999
该案例运行结果？ 1 2 1
合并了 task
```



- JobManager和TaskManager保持通信
  - TaskManager里面的Taskslot是运行task的资源（内存，CPU，网络资源等）
  - 默认情况下一个TakaManager只有一个Taskslot
  - task任务运行在Taskslot
  - JobManager会更新task的状态到Client
- ActorSystem
  - 保证系统之间通信方式，心跳检测。
- TaskManager -> TaskSlot -> task -> 并行度

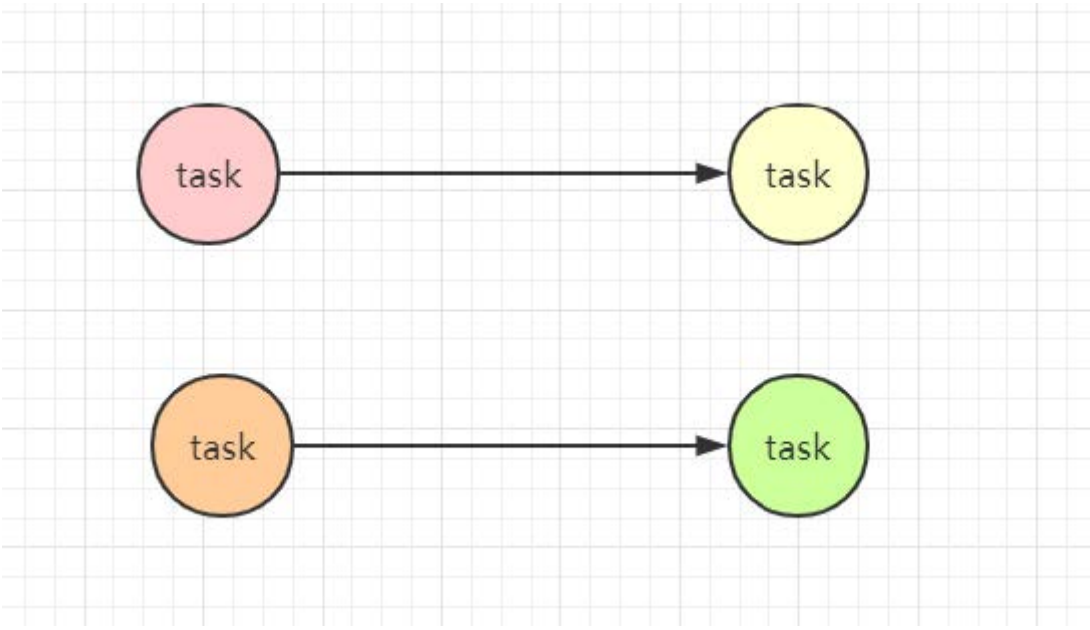
Flink 分布式运行时环境



11.数据传输的策略

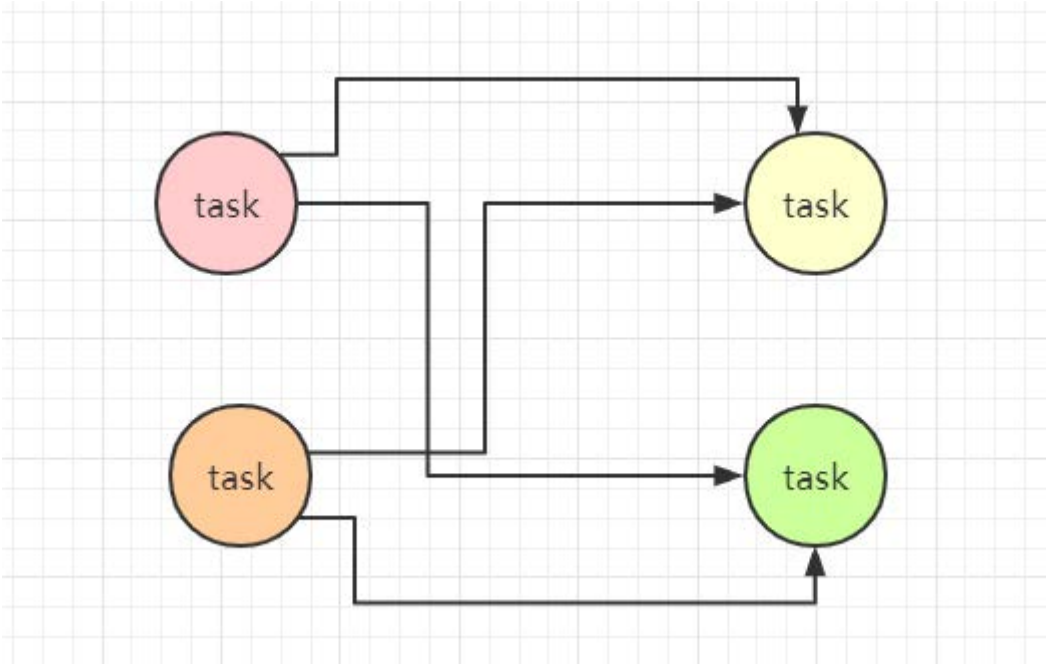
11.1 forward strategy

- 一个 task 的输出只发送给一个 task 作为输入
- 如果两个 task 都在一个 JVM 中的话，那么就可以避免网络开销
- 如果产生forward操作就会合并成为一个task



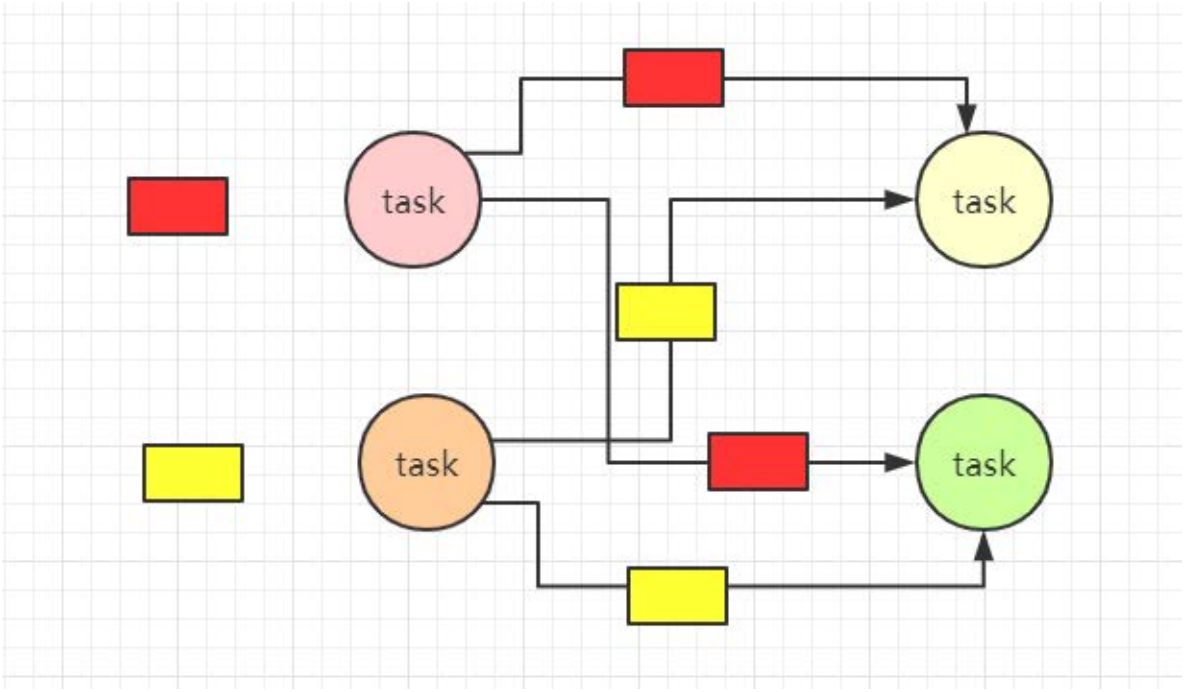
11.2 key based strategy (key by)

- 数据需要按照某个属性(我们称为 key)进行分组(或者说分区)
- 相同 key 的数据需要传输给同一个 task，在一个 task 中进行处理



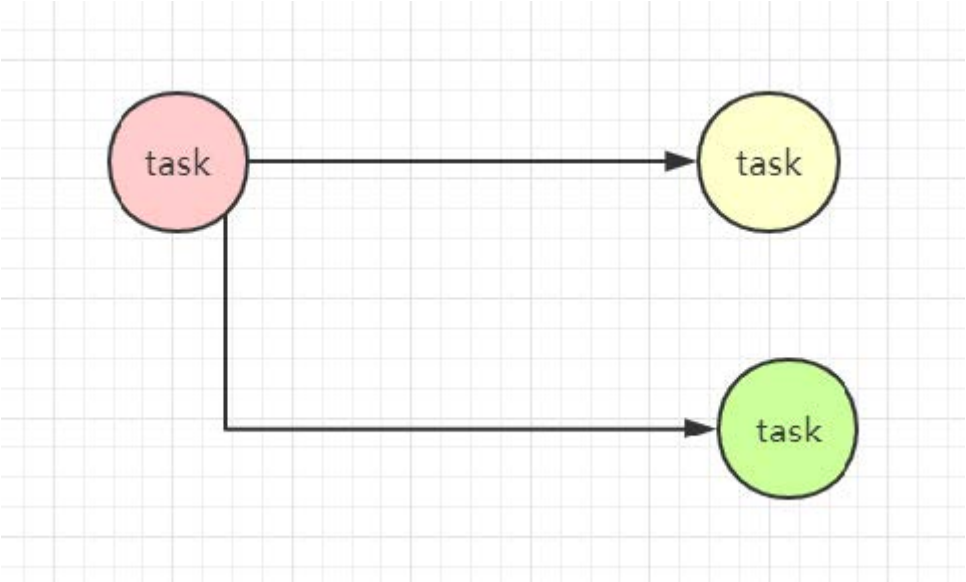
11.3 broadcast strategy

- 广播的方式，当前task的数据会传送到下游的所有task里面。



11.4 random strategy

- 数据随机的从一个 task 中传输到下游的operator所有的subtask
- 保证数据能均匀的传输给所有的subtask



与Kafka进行整合

1. 数据源是Kafka

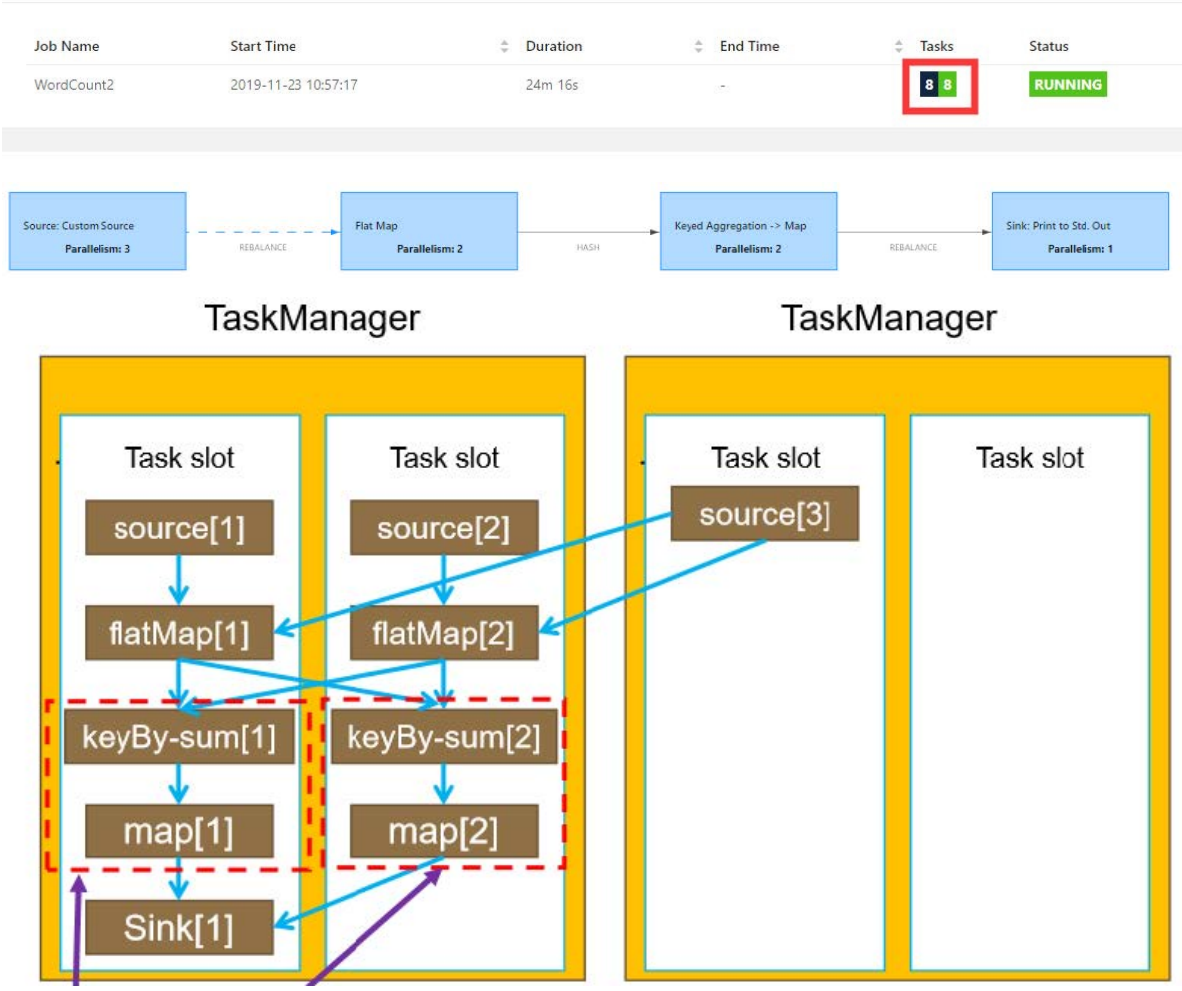
Maven Dependency	Supported since	Consumer and Producer Class name	Kafka version	Notes
flink-connector-kafka-0.8_2.11	1.0.0	FlinkKafkaConsumer08 FlinkKafkaProducer08	0.8.x	Uses the SimpleConsumer API of Kafka internally. Offsets are committed to ZK by Flink.
flink-connector-kafka-0.9_2.11	1.0.0	FlinkKafkaConsumer09 FlinkKafkaProducer09	0.9.x	Uses the new Consumer API Kafka.
flink-connector-kafka-0.10_2.11	1.2.0	FlinkKafkaConsumer010 FlinkKafkaProducer010	0.10.x	This connector supports Kafka messages with timestamps both for producing and consuming.
flink-connector-kafka-0.11_2.11	1.4.0	FlinkKafkaConsumer011 FlinkKafkaProducer011	0.11.x	Since 0.11.x Kafka does not support scala 2.10. This connector supports Kafka transactional messaging to provide exactly once semantic for the producer.
flink-connector-kafka_2.11	1.7.0	FlinkKafkaConsumer FlinkKafkaProducer	>= 1.0.0	This universal Kafka connector attempts to track the latest version of the Kafka client. The version of the client it uses may change between Flink releases. Starting with Flink 1.9 release, it uses the Kafka 2.2.0 client. Modern Kafka clients are backwards compatible with broker versions 0.10.0 or later. However for Kafka 0.11.x and 0.10.x versions, we recommend using dedicated flink-connector-kafka-0.11_2.11 and flink-connector-kafka-0.10_2.11 respectively.

• 12.Task和Slot的关系

```
#课堂案例job10
1.创建topic
kafka-topic.sh --create --zookeeper hadoop1:2181 --replication-factor 1 --
partitions 3 --topic job10
2.运行flink任务
flink run -c com.bw.flinkstreaming.job10.WordCount flink1.jar
```

#注意:
如果task的任务数量也就是并行度大于slot那么程序会无法运行。
1.一个TaksManager里面默认只有一个slot
2.在task运行的过程中会进行数据合并，比如说如下图的 KeyBy->Map:会产生 operator Chain的情况

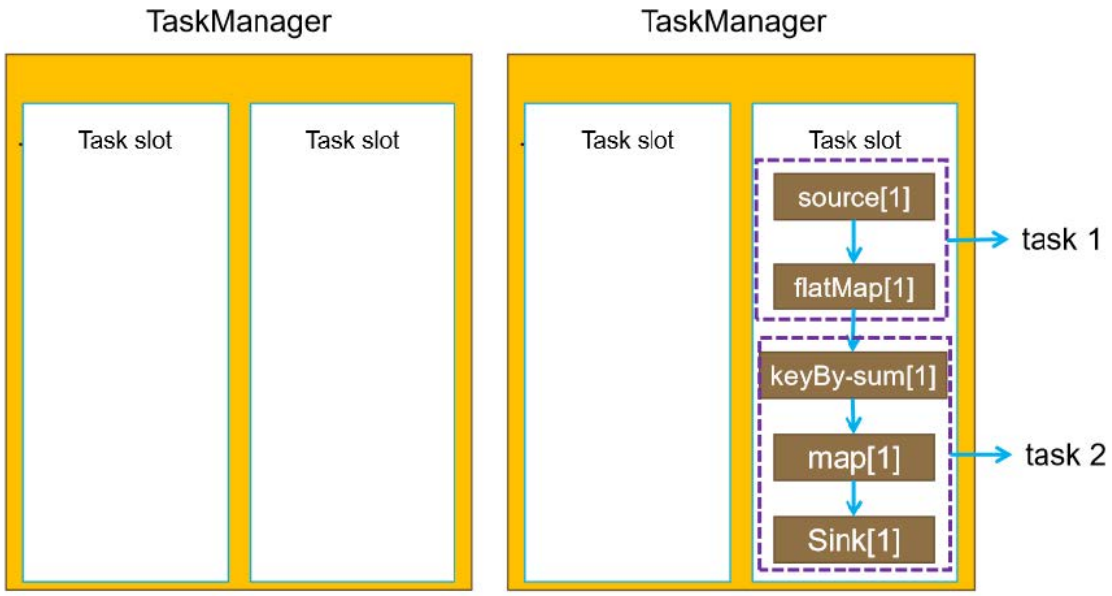
Operator Chain的条件:
1.数据传输策略是 forward strategy
2.在同一个 TaskManager 中运行
3.TaskManager会尽量保证task在同一个 JVM里面运行，有利于提升效率。



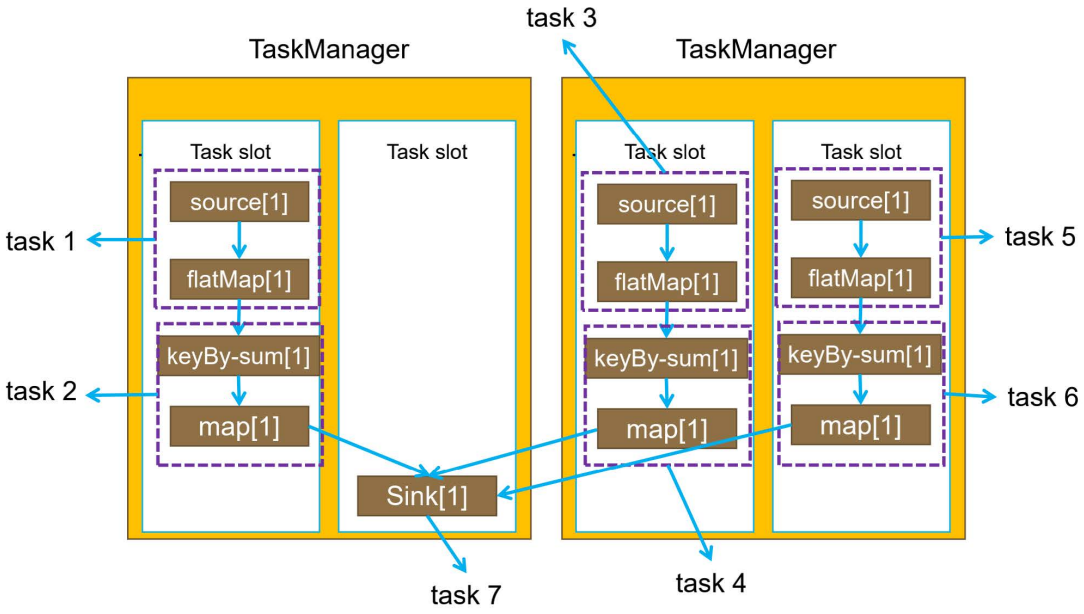
#课程案例job11
1.如果设置并行度都为1那么总共需要几个task
2.提交: flink run -c com.bw.flinkstreaming.job11.WordCount --hostname hadoop1 --port 9999
该案例运行结果?
    1   1   1   1
[socket flatMap] keybased [keyby|sum sink]
source -> flatmap 属于 operator Chain的情况，会将 souce->flatMap进行合并为一个task
keyby sum -> sink也是属于 oprator chain的情况，会将 keyby sum -> sink进行合并为一个task
上游task -> 下游task 因为中间有 keyby所以数据传输的策略是 keybased strategy

Running Jobs

Job Name	Start Time	Duration	End Time	Tasks	Status
WordCount2	2019-11-23 11:37:49	25s	-	2 2	RUNNING

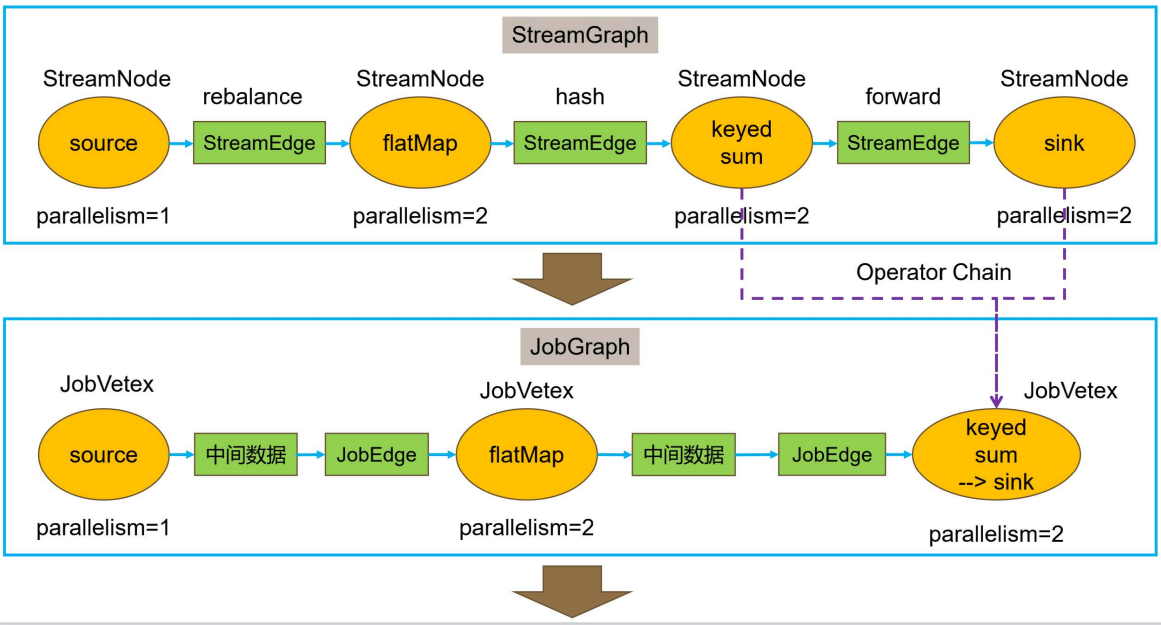


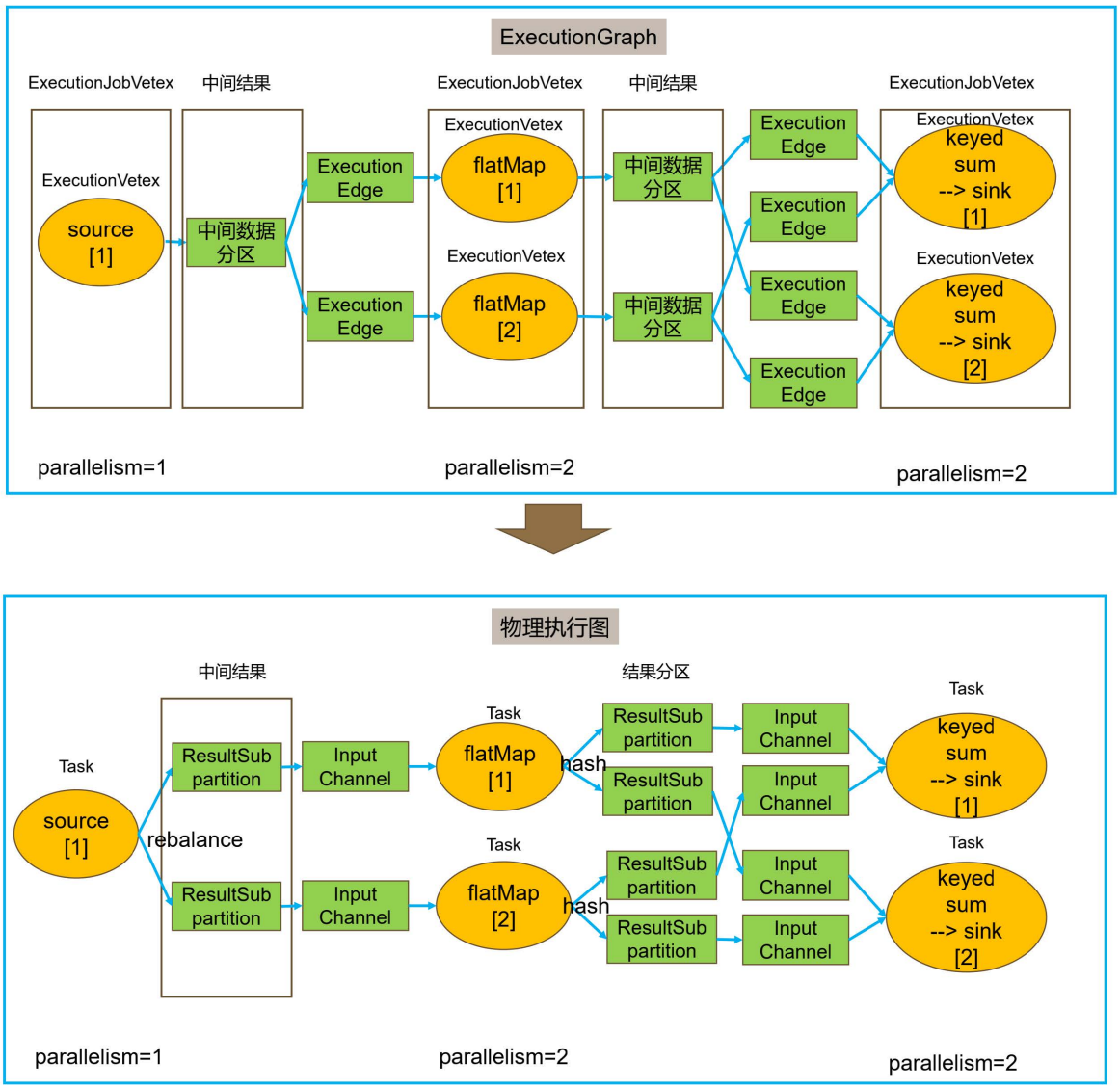

```
#课堂案例12：总共是多少个task
 *  source 3
 *  *  flatmap 3
 *  *  keyby sum 3
 *  *  map 3
 *  *  print 1
```



13.Flink四层图结构

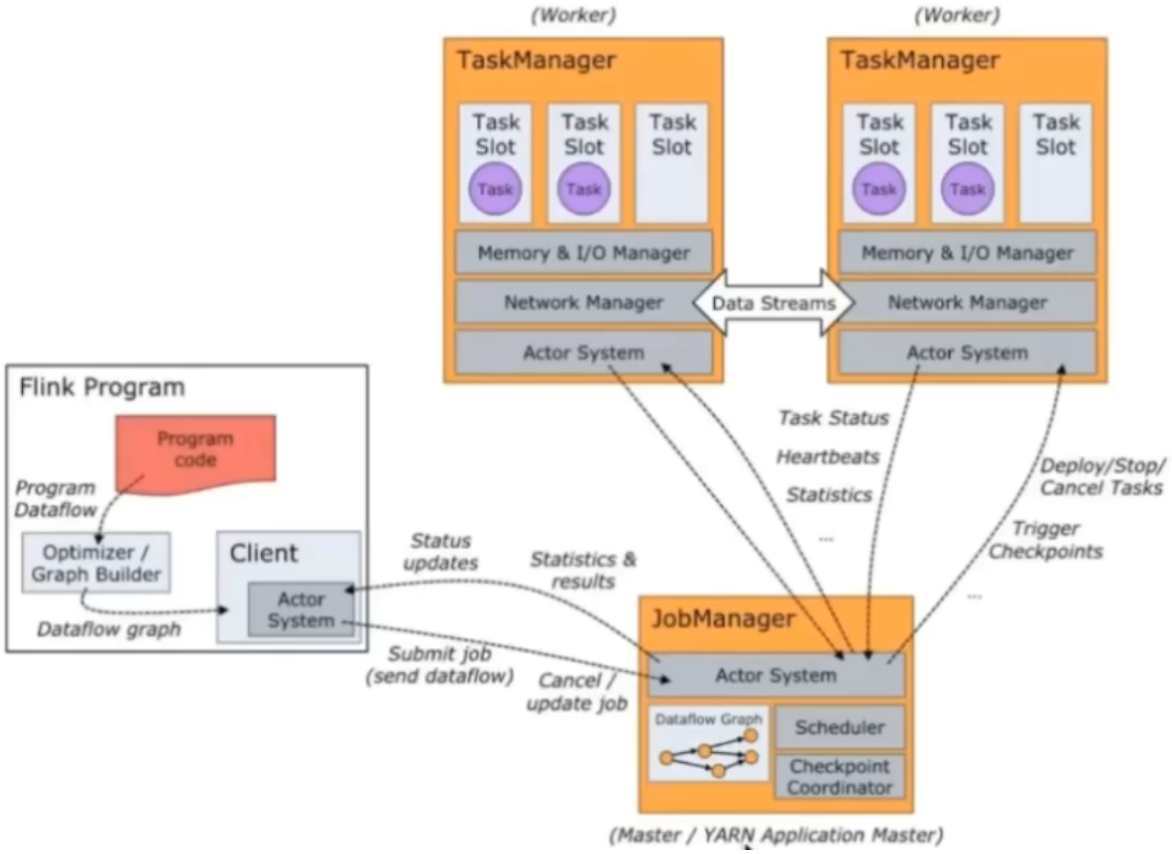
- Flink程序从提交到运行，需要经历如下4个阶段：
  - Stream Graph
  - Job Graph
  - Execution Graph
  - Physical Execution Graph
- Stream Graph和Job Graph是在客户端阶段完成的，然后client通过ActorSystem把JobGraph提交到JobManager
- Execution Graph阶段会引入并行度
- Physical Execution Graph：逻辑理解就可以了





14. 深入剖析Flink执行原理

- 1.Client相当于是Spark的Driver
- 2.ActorSystem相当于HadoopRPC，是用来进行节点之间通信
- 3.Client的ProgramDataFlow相当于是StreamGraph，然后经过优化成JobGraph
- 4.由ActorSystem把JobGraph提交到JobManager节点
- 5.JobManager对JobGraph引入并行度形成ExecutionGraph
- 6.然后由TaskScheduler进行task之间的任务调度，将task分配到TaskSlot里面运行，底层也是基于ActorSystem进行交互
- 7.TaskManager会返回task运行的状态信息给JobManager，JobManager会将task运行的状态信息返回给客户端。
- 8.在页面上可以看到任务的执行情况，是由TaskManager返回给JobManager，然后JobManager返回给客户端的。


(Worker)
(Worker)
TaskManager
TaskManager
Task Slot
Task
Memory & I/O Manager
Network Manager
Actor System
Data Streams
Flink Program
Program code
Program Dataflow
Optimizer / Graph Builder
Client
Actor System
Dataflow graph
Status updates
Statistics & results
Submit job (send dataflow)
Cancel / update job
Task Status
Heartbeats
Statistics
...
Deploy/Stop/ Cancel Tasks
Trigger Checkpoints
JobManager
Dataflow Graph
Scheduler
Checkpoint Coordinator
(Master / YARN Application Master)