**Flink入门实战(上)**

**一、Flink 简介**

**1、初识 Flink**

Flink 起源于 Stratosphere 项目，Stratosphere 是在 2010~2014 年由 3 所地处柏林

的大学和欧洲的一些其他的大学共同进行的研究项目，2014 年 4 月 Stratosphere 的

代 码被 复制 并捐赠 给了 Apache 软件基 金会， 参加 这个 孵化项 目的 初始 成员 是

Stratosphere 系统的核心开发人员，2014 年 12 月，Flink 一跃成为 Apache 软件基金

会的顶级项目。

在德语中，Flink 一词表示快速和灵巧，项目采用一只松鼠的彩色图案作为 logo，

这不仅是因为松鼠具有快速和灵巧的特点，还因为柏林的松鼠有一种迷人的红棕色，

而 Flink 的松鼠 logo 拥有可爱的尾巴，尾巴的颜色与 Apache 软件基金会的 logo 颜

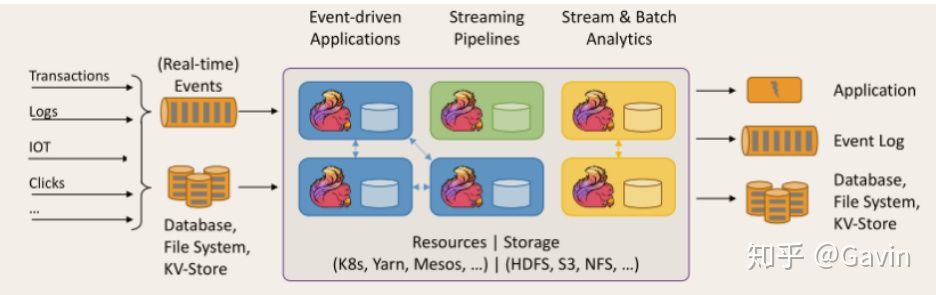
色相呼应，也就是说，这是一只 Apache 风格的松鼠。



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有

状态计算。Flink 被设计在所有常见的集群环境中运行，以内存执行速度和任意规模

来执行计算。

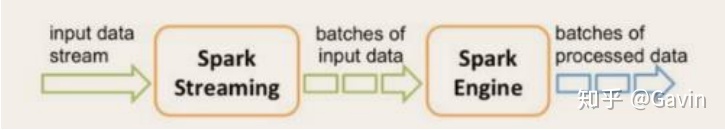


**2、Flink 的重要特点**

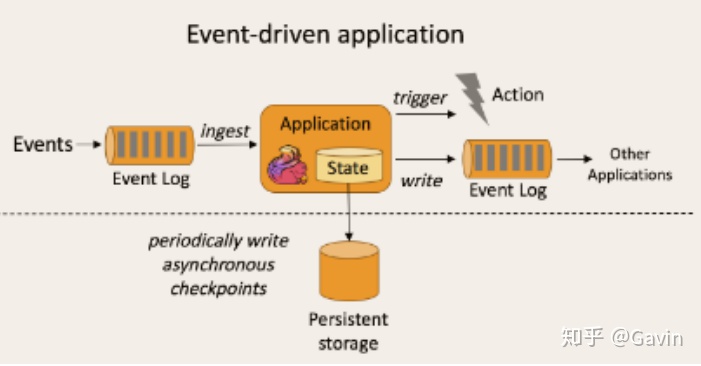
（1）**事件驱动型(Event-driven)**

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流提取数据，并   
根据到来的事件触发计算、状态更新或其他外部动作。比较典型的就是以 kafka 为   
代表的消息队列几乎都是事件驱动型应用。

与之不同的就是 SparkStreaming 微批次，如图：

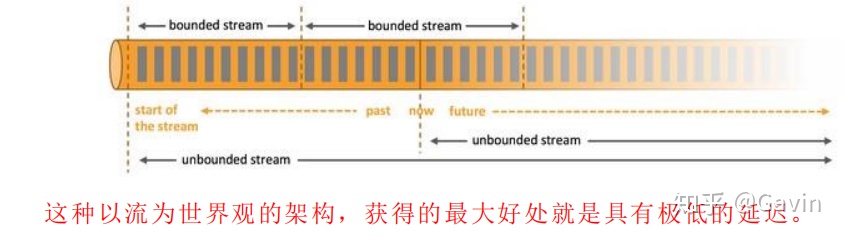


事件驱动型：

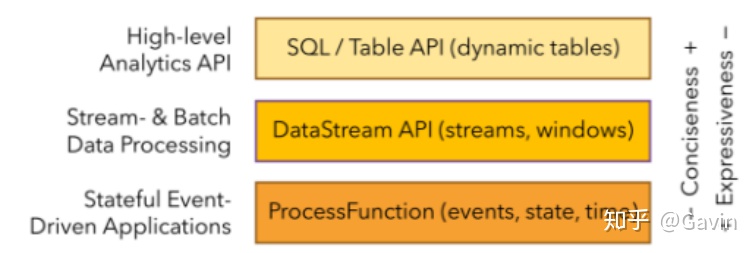


（2）**流与批的世界观**

**批处理**的特点是有界、持久、大量，非常适合需要访问全套记录才能完成的计   
算工作，一般用于离线统计。   
**流处理**的特点是无界、实时, 无需针对整个数据集执行操作，而是对通过系统   
传输的每个数据项执行操作，一般用于实时统计。  
在 spark 的世界观中，一切都是由批次组成的，离线数据是一个大批次，而实   
时数据是由一个一个无限的小批次组成的。   
而在 flink 的世界观中，一切都是由流组成的，离线数据是有界限的流，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。   
**无界数据流**：无界数据流有一个开始但是没有结束，它们不会在生成时终止并   
提供数据，必须连续处理无界流，也就是说必须在获取后立即处理 event。对于无界   
数据流我们无法等待所有数据都到达，因为输入是无界的，并且在任何时间点都不   
会完成。处理无界数据通常要求以特定顺序（例如事件发生的顺序）获取 event，以   
便能够推断结果完整性。   
**有界数据流**：有界数据流有明确定义的开始和结束，可以在执行任何计算之前   
通过获取所有数据来处理有界流，处理有界流不需要有序获取，因为可以始终对有   
界数据集进行排序，有界流的处理也称为批处理。



（3） **分层 api**



最底层级的抽象仅仅提供了有状态流，它将通过过程函数（Process Function）   
被嵌入到 DataStream API 中。底层过程函数（Process Function） 与 DataStream API    
相集成，使其可以对某些特定的操作进行底层的抽象，它允许用户可以自由地处理   
来自一个或多个数据流的事件，并使用一致的容错的状态。除此之外，用户可以注   
册事件时间并处理时间回调，从而使程序可以处理复杂的计算。   
实际上，大多数应用并不需要上述的底层抽象，而是针对核心 API（Core APIs）   
进行编程，比如 DataStream API（有界或无界流数据）以及 DataSet API（有界数据   
集）。这些 API 为数据处理提供了通用的构建模块，比如由用户定义的多种形式的  
转换（transformations），连接（joins），聚合（aggregations），窗口操作（windows）   
等等。DataSet API 为有界数据集提供了额外的支持，例如循环与迭代。这些 API   
处理的数据类型以类（classes）的形式由各自的编程语言所表示。   
Table API 是以表为中心的声明式编程，其中表可能会动态变化（在表达流数据   
时）。Table API 遵循（扩展的）关系模型：表有二维数据结构（schema）（类似于   
关系数据库中的表），同时 API 提供可比较的操作，例如 select、project、join、group-by、   
aggregate 等。Table API 程序声明式地定义了什么逻辑操作应该执行，而不是准确地   
确定这些操作代码的看上去如何。   
尽管 Table API 可以通过多种类型的用户自定义函数（UDF）进行扩展，其仍不   
如核心 API 更具表达能力，但是使用起来却更加简洁（代码量更少）。除此之外，   
Table API 程序在执行之前会经过内置优化器进行优化。   
你可以在表与 DataStream/DataSet 之间无缝切换，以允许程序将 Table API 与   
DataStream 以及 DataSet 混合使用。   
Flink 提供的最高层级的抽象是 SQL 。这一层抽象在语法与表达能力上与   
Table API 类似，但是是以 SQL 查询表达式的形式表现程序。SQL 抽象与 Table API   
交互密切，同时 SQL 查询可以直接在 Table API 定义的表上执行。   
目前 Flink 作为批处理还不是主流，不如 Spark 成熟，所以 DataSet 使用的并不   
是很多。Flink Table API 和 Flink SQL 也并不完善，大多都由各大厂商自己定制。所   
以我们主要学习 DataStream API 的使用。实际上 Flink 作为最接近 Google DataFlow   
模型的实现，是流批统一的观点，所以基本上使用 DataStream 就可以了。

Flink 几大模块

* λ Flink Table & SQL(还没开发完)
* λ Flink Gelly(图计算)
* λ Flink CEP(复杂事件处理)

**二、快速上手**

**搭建 maven 工程 FlinkTutorial**

（1）pom 文件引入依赖文件

**<?**xml version**=**"1.0" encoding**=**"UTF-8"**?>**

**<**project xmlns**=**"http://maven.apache.org/POM/4.0.0"

xmlns**:**xsi**=**"http://www.w3.org/2001/XMLSchema-instance"

xsi**:**schemaLocation**=**"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**

**<**modelVersion**>**4**.**0**.**0**</**modelVersion**>**

**<**groupId**>**com**.**apache**.**flink**</**groupId**>**

**<**artifactId**>**FlinkTutorial**</**artifactId**>**

**<**version**>**1**.**0**-**SNAPSHOT**</**version**>**

**<**dependencies**>**

**<**dependency**>**

**<**groupId**>**org**.**apache**.**flink**</**groupId**>**

**<**artifactId**>**flink**-**scala\_2**.**11**</**artifactId**>**

**<**version**>**1**.**7**.**2**</**version**>**

**</**dependency**>**

**<!--** https**:***//mvnrepository.com/artifact/org.apache.flink/flink-streaming-scala -->*

**<**dependency**>**

**<**groupId**>**org**.**apache**.**flink**</**groupId**>**

**<**artifactId**>**flink**-**streaming**-**scala\_2**.**11**</**artifactId**>**

**<**version**>**1**.**7**.**2**</**version**>**

**</**dependency**>**

**</**dependencies**>**

**<**build**>**

**<**plugins**>**

**<!--** 该插件用于将 Scala 代码编译成 **class** **文件** **-->**

**<**plugin**>**

**<**groupId**>**net**.**alchim31**.**maven**</**groupId**>**

**<**artifactId**>**scala**-**maven**-**plugin**</**artifactId**>**

*//<version>3.4.6</version>*

**<**executions**>**

**<**execution**>**

**<!--** 声明绑定到 maven 的 compile 阶段 **-->**

**<**goals**>**

**<**goal**>**testCompile**</**goal**>**

**</**goals**>**

**</**execution**>**

**</**executions**>**

**</**plugin**>**

**<**plugin**>**

**<**groupId**>**org**.**apache**.**maven**.**plugins**</**groupId**>**

**<**artifactId**>**maven**-**assembly**-**plugin**</**artifactId**>**

**<**version**>**3**.**0**.**0**</**version**>**

**<**configuration**>**

**<**descriptorRefs**>**

**<**descriptorRef**>**jar**-**with**-**dependencies**</**descriptorRef**>**

**</**descriptorRefs**>**

**</**configuration**>**

**<**executions**>**

**<**execution**>**

**<**id**>**make**-**assembly**</**id**>**

**<**phase**>**package**</**phase**>**

**<**goals**>**

**<**goal**>**single**</**goal**>**

**</**goals**>**

**</**execution**>**

**</**executions**>**

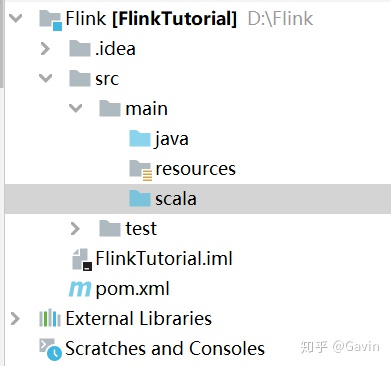
**</**plugin**>**

**</**plugins**>**

**</**build**>**

**</**project**>**

（2）添加 scala 框架 和 scala 文件夹



（3）批处理 wordcount

* 先在resource创建hello文本，编辑以下内容

hello world

hello flink

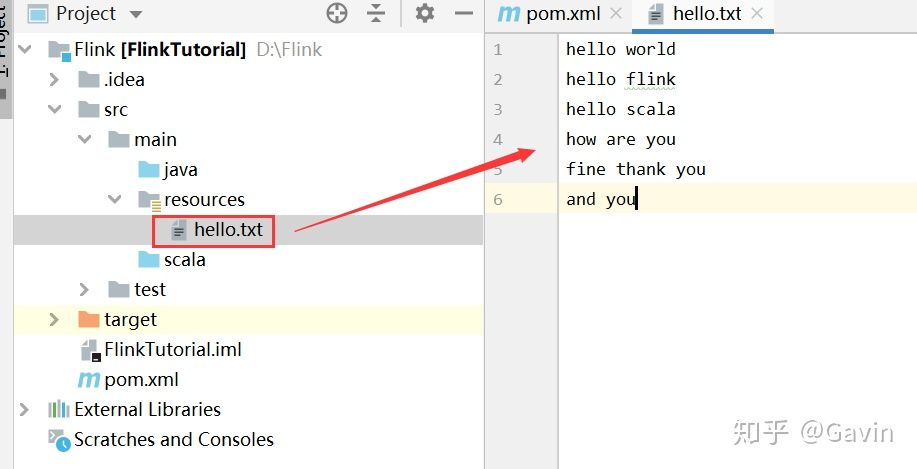
hello scala

how are you

fine thank you

and you

如图所示:



* 编写**批处理**wordcount

**package** com.apache.fink

**import** org.apache.flink.api.scala.\_

*/\*\**

*\* 批处理 wordcount*

*\*/*

object WordCount **{**

def **main(**args**:** Array**[**String**]):** Unit **=** **{**

*// 创建一个批处理的执行环境*

val env **=** ExecutionEnvironment**.**getExecutionEnvironment

*// 从文件中读取数据*

val inputPath **=** "D:\\Flink\\src\\main\\resources\\hello.txt"

val inputDataSet **=** env**.**readTextFile**(**inputPath**)**

*// 分词之后做count*

val wordCountDataSet **=** inputDataSet**.**flatMap**(**\_**.**split**(**" "**))**

**.**map**(** **(**\_**,** 1**)** **)**

**.**groupBy**(**0**)**

**.**sum**(**1**)**

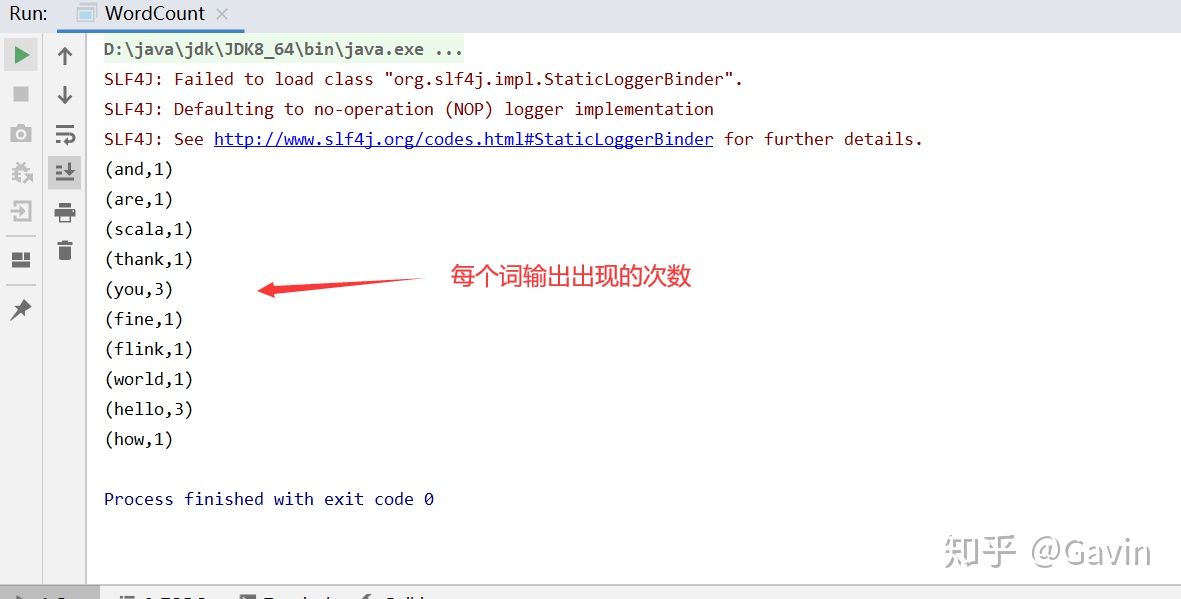
*// 打印输出*

wordCountDataSet**.**print**()**

**}**

**}**

启动程序，控制台打印数据的信息如下：



* 编写**流处理（**实时处理**）**

**package** com.apache.flink

**import** org.apache.flink.streaming.api.scala.\_

*/\*\**

*\* 流处理*

*\*/*

object StreamWordCount **{**

def **main(**args**:** Array**[**String**]):** Unit **=** **{**

*//创建流处理环境*

val env**:** StreamExecutionEnvironment **=** StreamExecutionEnvironment**.**getExecutionEnvironment

*//接收socket文本流*

val dataStream**:** DataStream**[**String**]** **=** env**.**socketTextStream**(**"spark2.x"**,** 7777**)**

*//处理 分组并且sum聚合*

val sumStream**:** DataStream**[(**String**,** Int**)]** **=** dataStream

**.**flatMap**(**\_**.**split**(**" "**))**

**.**filter**(**\_**.**nonEmpty**)**

**.**map**((**\_**,** 1**))**

**.**keyBy**(**0**)**

**.**sum**(**1**)**

*//打印*

sumStream**.**print**()**

env**.**execute**()**

**}**

**}**

在虚拟机执行命令

*//发现nc命令没有安装*

**[**root@spark2 **~]**# nc **-**lk 7777

**-**bash**:** nc**:** command not found

*//安装命令：*

**[**root@spark2 **~]**# yum install **-**y nc

*//执行7777命令*

**[**root@spark2 **~]**# nc **-**lk 7777

*//启动idea程序后，即可在虚拟机输出消费的数据，比如：*

helloworld

flink

spark

**//**idea程序也将会打印出来。如下图所示：



当然，有实时处理也同样有离线处理，接下来简单演示一个离线处理的简单demo

* 编写**离线处理**

具体代码实现

**package** com.apache.flink

**import** org.apache.flink.api.scala.**{**AggregateDataSet**,** DataSet**,** ExecutionEnvironment**}**

**import** org.apache.flink.api.scala.\_

*/\*\**

*\* 离线处理*

*\*/*

object DateSetWcApp **{**

def **main(**args**:** Array**[**String**]):** Unit **=** **{**

*//构造执行环境*

val env**:** ExecutionEnvironment **=** ExecutionEnvironment**.**getExecutionEnvironment

*//读取文件*

val txtDataSet**:** DataSet**[**String**]** **=** env**.**readTextFile**(**"D:\\Flink\\src\\main\\resources\\hello.txt"**)**

*//经过groupby进行分组，sum进行聚合*

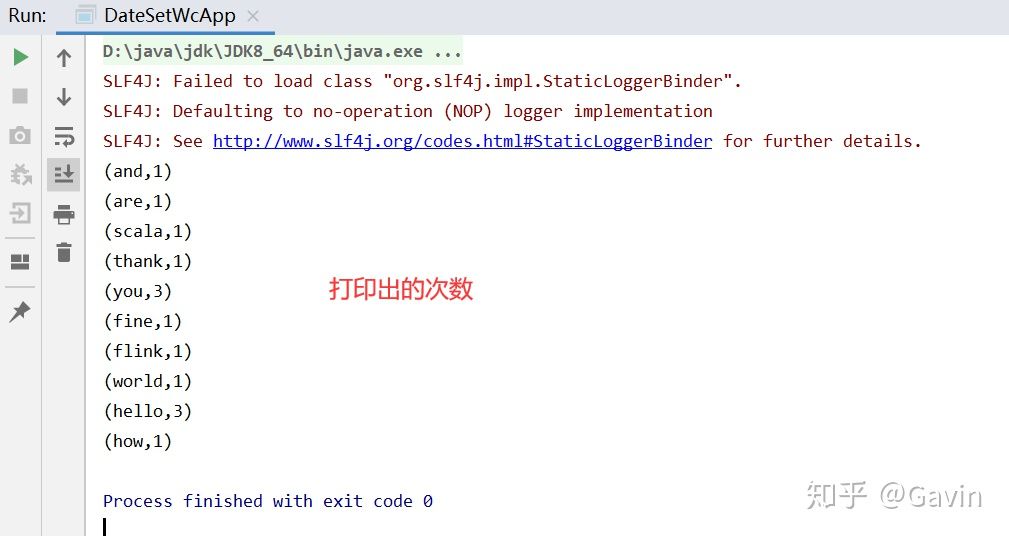
val aggSet**:** AggregateDataSet**[(**String**,** Int**)]** **=** txtDataSet**.**flatMap**(**\_**.**split**(**" "**)).**map**((**\_**,** 1**)).**groupBy**(**0**).**sum**(**1**)**

aggSet**.**print**()**

**}**

**}**

启动程序，控制台打印数据的信息如下：



以上可以对比一下：离线处理与批处理打印输出的结果是不一样的

**三、Flink 部署**

**1、Standalone 模式**

**（1）安装**

* 解压缩 flink-1.7.2-bin-hadoop27-scala\_2.11.tgz，进入 conf 目录中。

**[**root@spark1 module**]**# tar **-**zxvf flink**-**1**.**7**.**2**-**bin**-**scala\_2**.**11**.**tgz

**[**root@spark1 module**]**# cd flink**-**1**.**7**.**2**/**conf**/**

* 修改 flink/conf/flink-conf.yaml 文件

**[**root@spark1 conf**]**# vim flink**-**conf**.**yaml

jobmanager**.**rpc**.**address**:** hadoop1**.**x

* 修改 /conf/slave 文件

**[**root@spark1 conf**]**# vim slave

hadoop2**.**x

hadoop3**.**x

* 分发到其他两台上(hadoop2.x、hadoop3.x)

**[**root@hadoop1 module**]**# scp **-**r flink**-**1**.**7**.**2**/** hadoop2**.**x**:/**usr**/**local**/**etc**/**hadoop**/**module**/**

**[**root@hadoop1 module**]**# scp **-**r flink**-**1**.**7**.**2**/** hadoop3**.**x**:/**usr**/**local**/**etc**/**hadoop**/**module**/**

* 启动

**[**root@hadoop1 bin**]**# pwd

**/**usr**/**local**/**etc**/**hadoop**/**module**/**flink**-**1**.**7**.**2**/**bin

**[**root@hadoop1 bin**]**# **./**start**-**cluster**.**sh

Starting cluster**.**

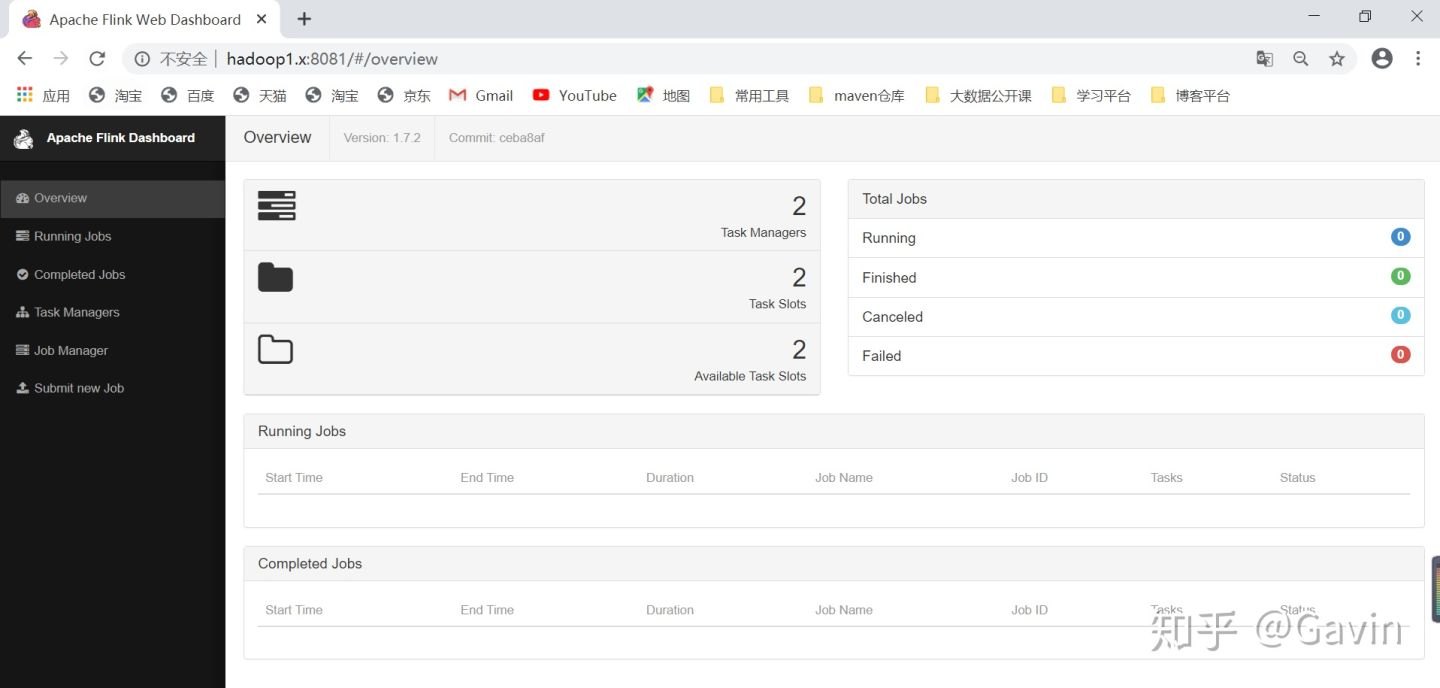
Starting standalonesession daemon on host hadoop1**.**x**.**

Starting taskexecutor daemon on host hadoop2**.**x**.**

Starting taskexecutor daemon on host hadoop3**.**x**.**

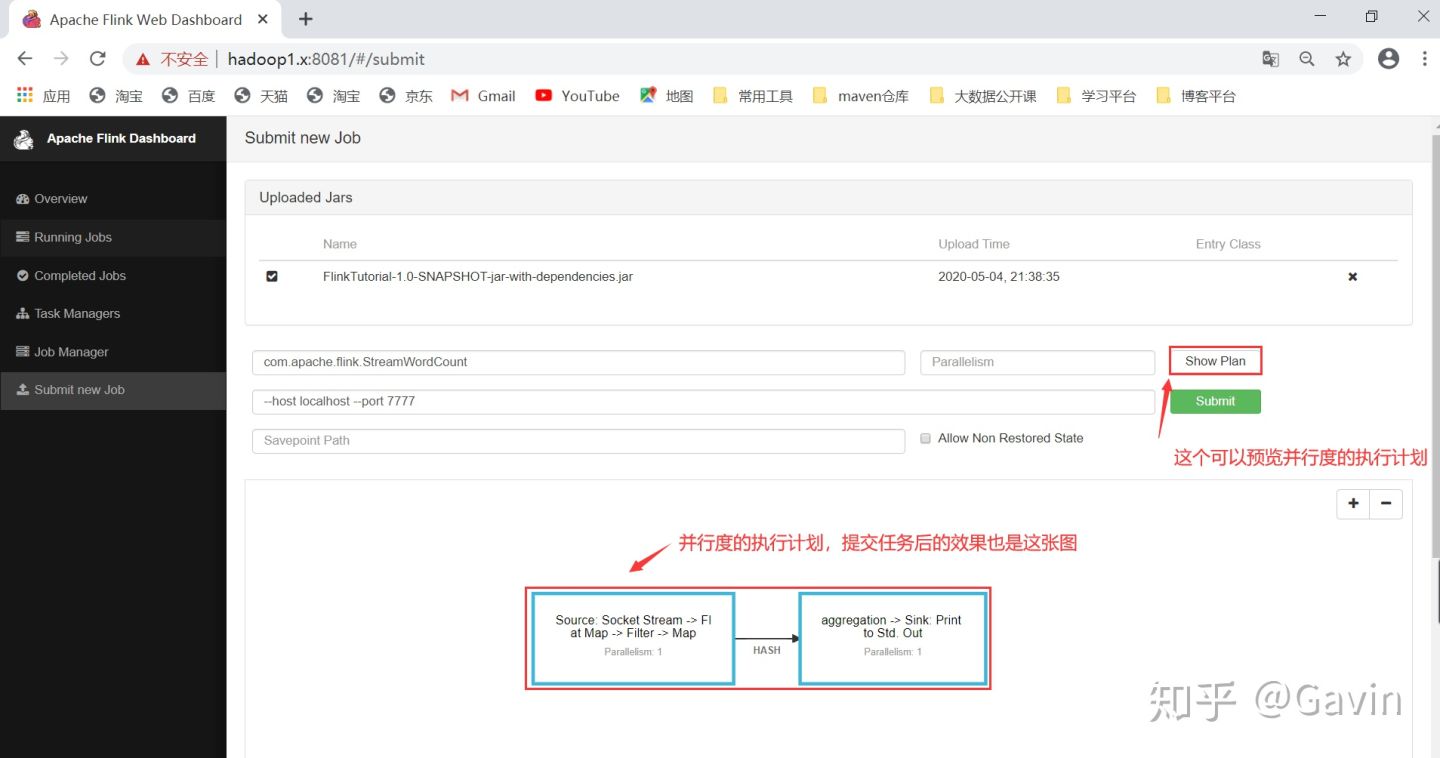
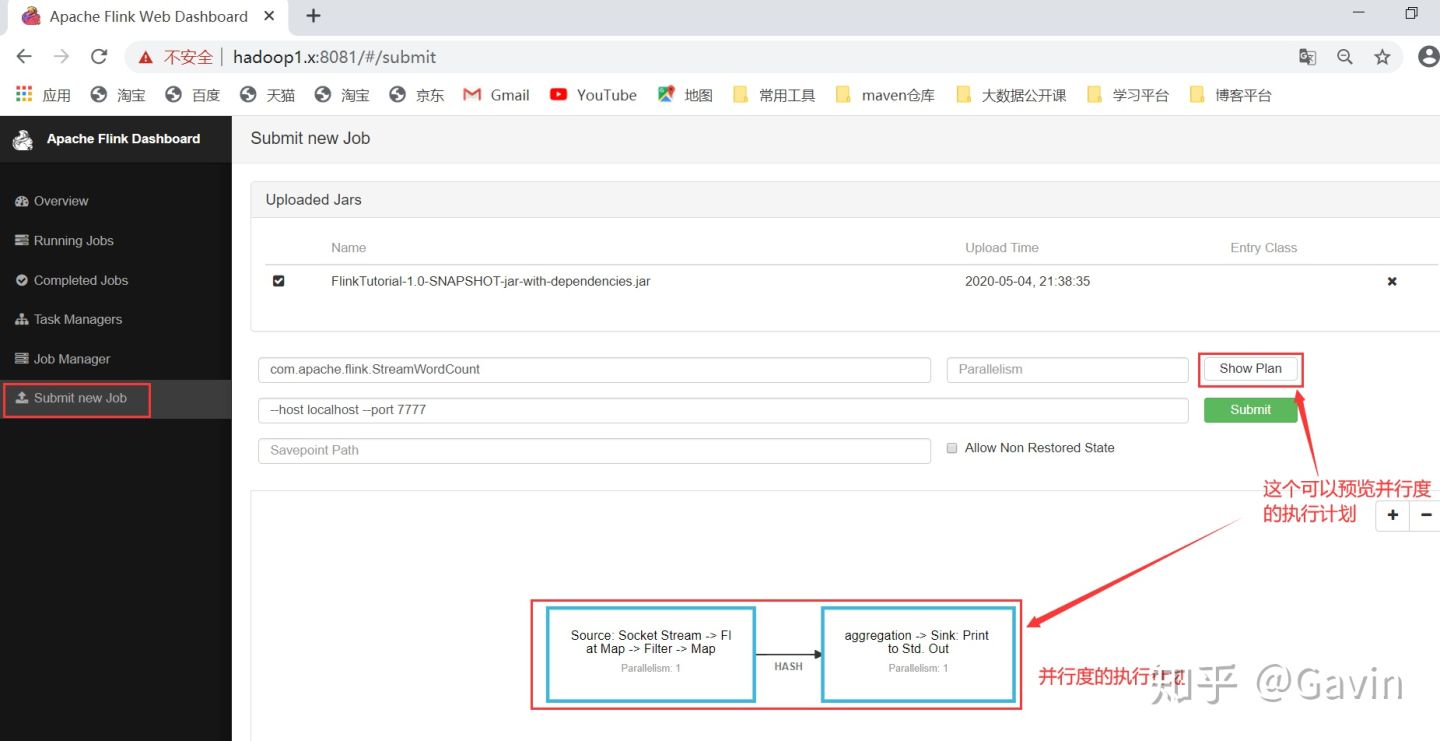
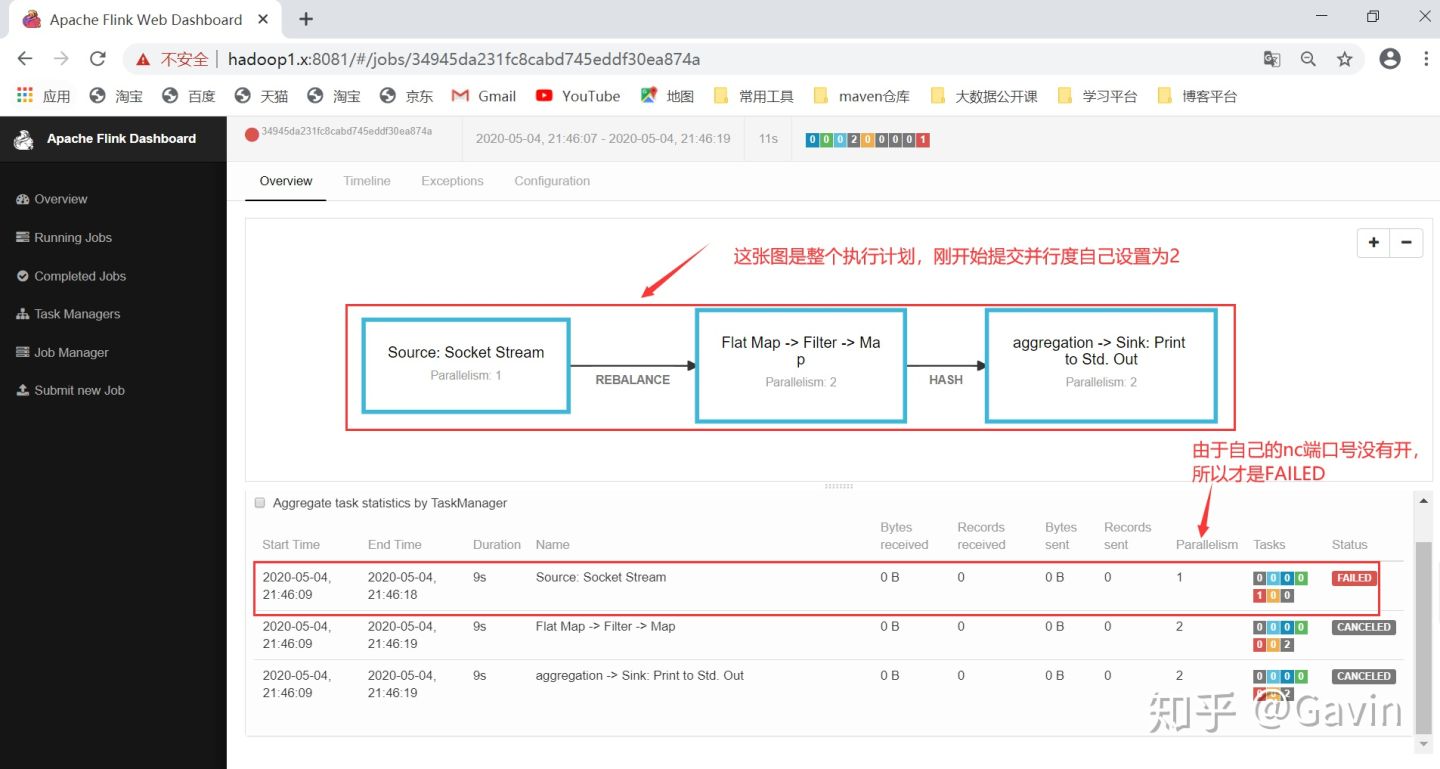
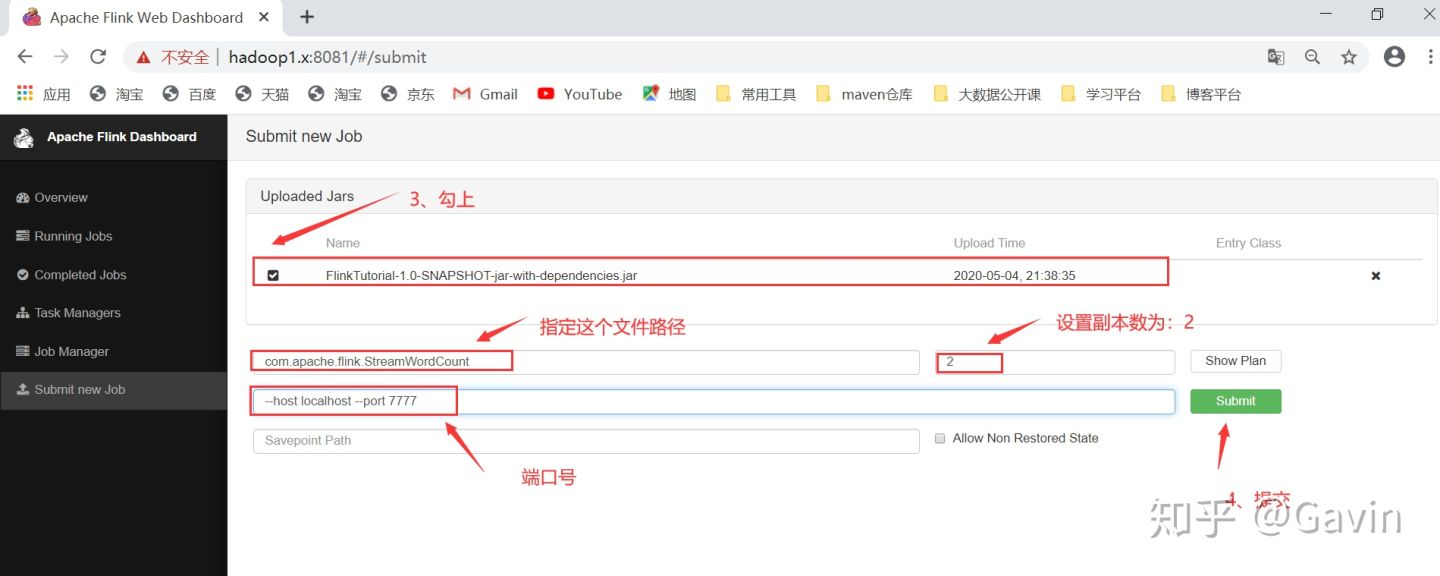
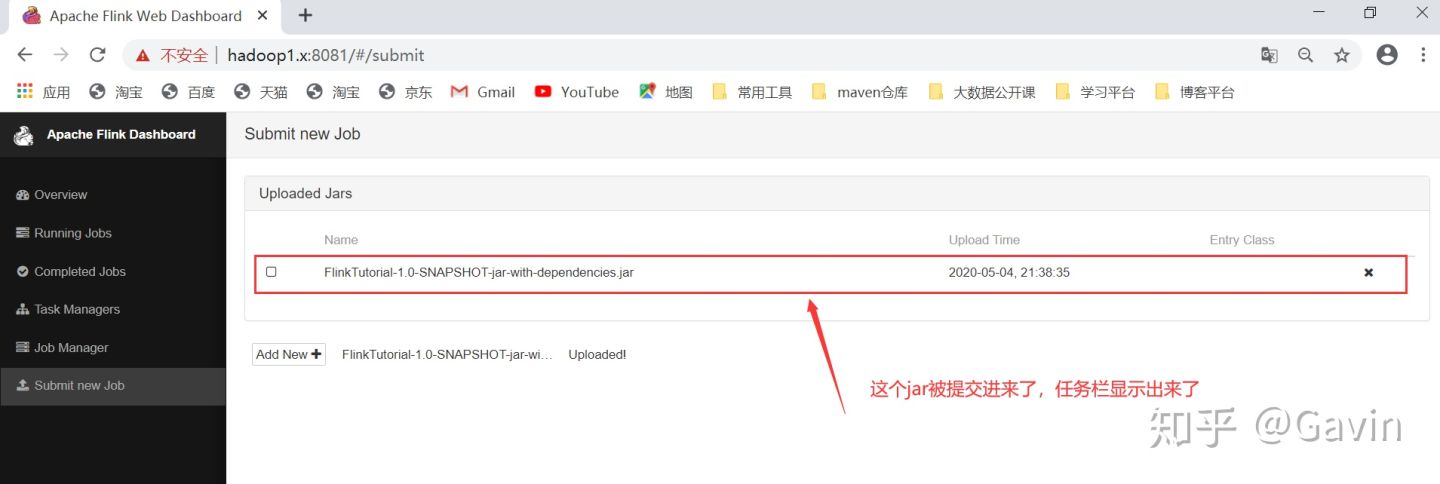
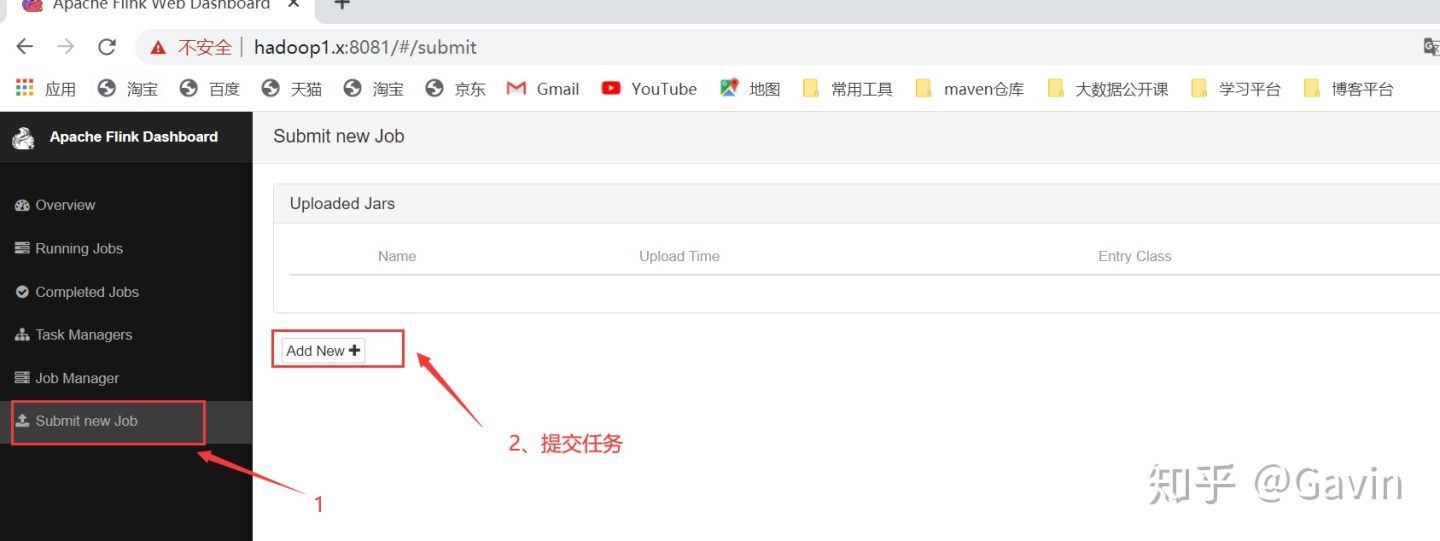
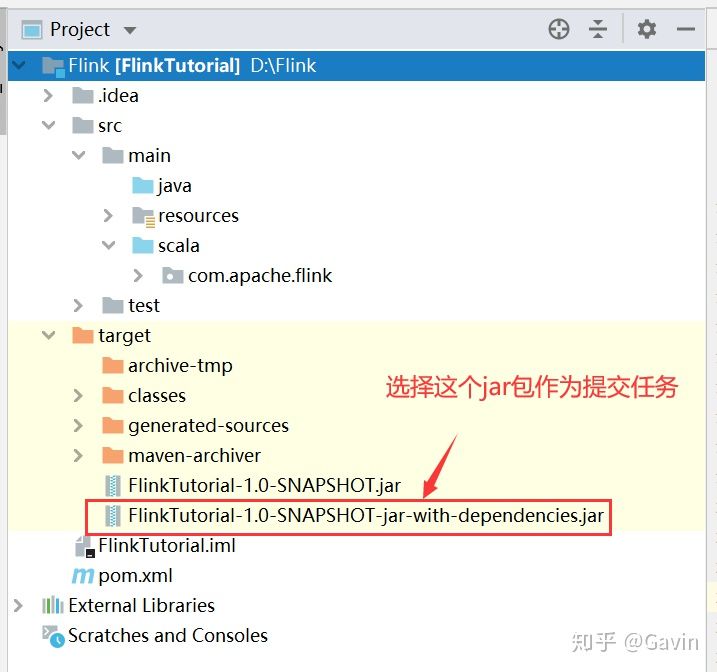
**[**root@hadoop1 bin**]**#

* 访问 [http://hadoop1.x:8081](https://link.zhihu.com/?target=http%3A//hadoop1.x%3A8081) 可以对 flink 集群和任务进行监控管理,如图所示：



**提交任务**

（1）在idea代码打包，上传到flink监控管理Web页面



**视频演示：**

Standalone 模式提交任务

其中，想要启动Running Jobs运行状态

首先，在 flink/conf/flink-conf.yaml 文件

[root@spark1 conf]# vim flink-conf.yaml

taskmanager.numberOfTaskSlots: 3

执行这条命令：

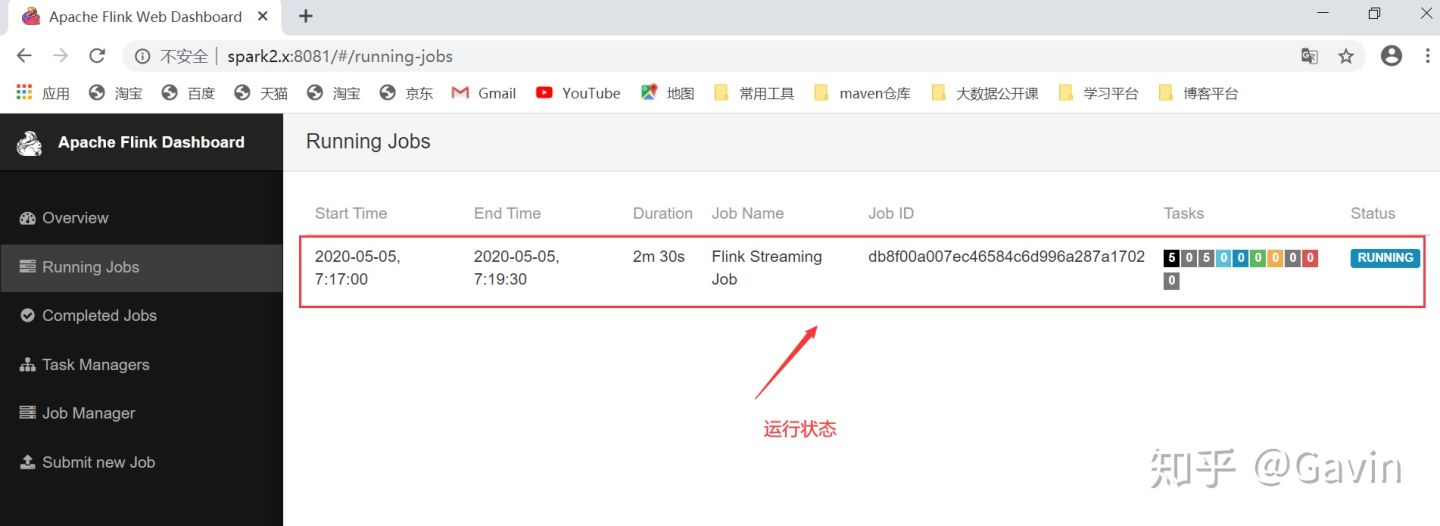
**[**root@spark2 bin**]**# **./**flink run **-**c com**.**apache**.**flink**.**StreamWordCount **-**p 2 **/**usr**/**local**/**etc**/**hadoop**/**module**/**flink**-**1**.**7**.**2**/**FlinkTutorial**-**1**.**0**-**SNAPSHOT**-**jar**-**with**-**dependencies**.**jar **--**host localhost **--**port 7777

Starting execution of program

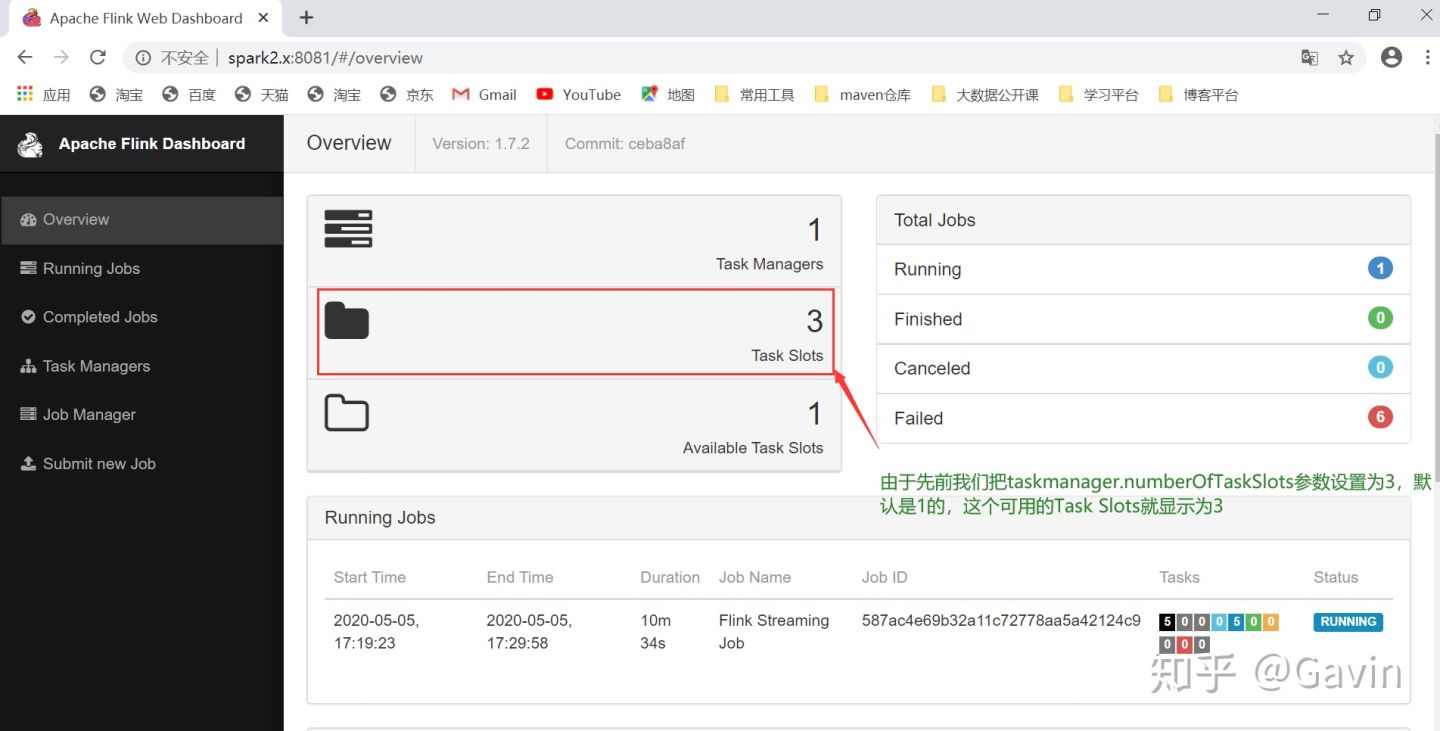
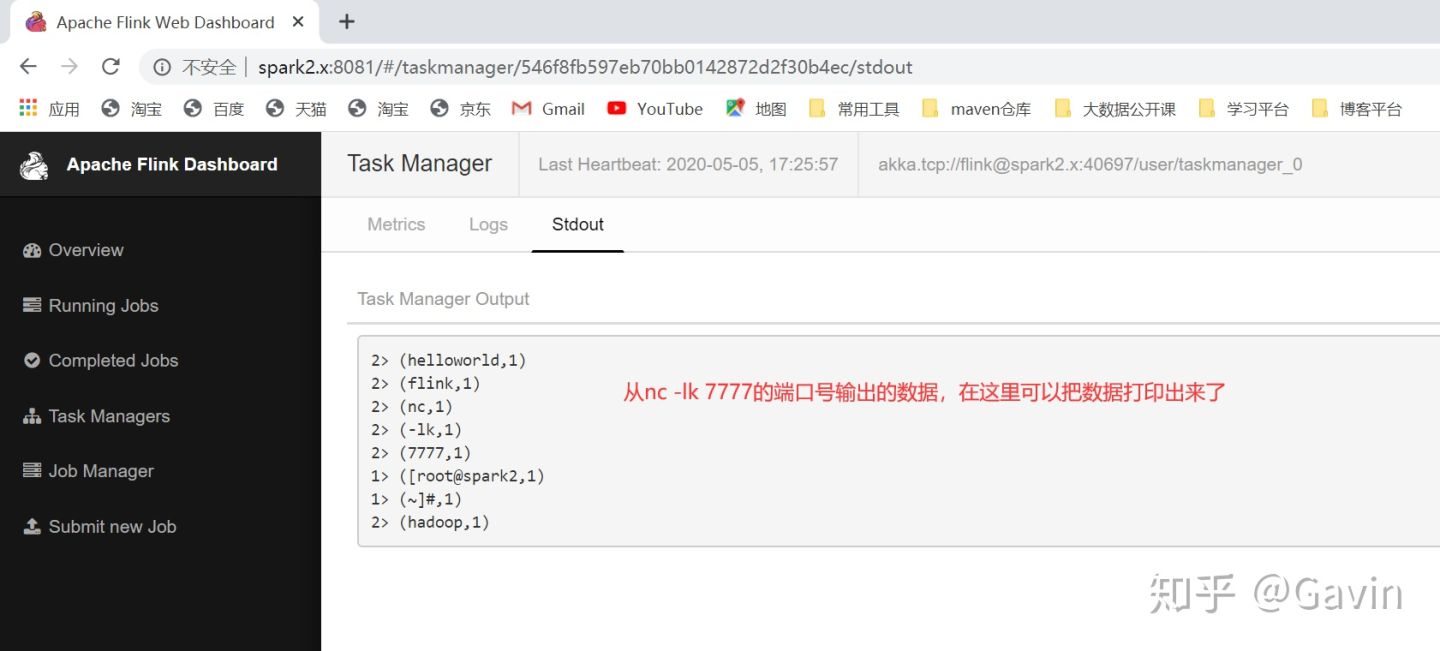
*//执行这条命令之前，需要监听端口号，防止出错*

**[**root@spark2 **~]**# nc **-**lk 7777

查看：



Task Manager查看数据，输出的并行度信息



当然，若想把当前的任务取消，我们可以执行这条命令：

**[**root@spark2 bin**]**# pwd

**/**usr**/**local**/**etc**/**hadoop**/**module**/**flink**-**1**.**7**.**2**/**bin

*//列出所有正在运行的job*

**[**root@spark2 bin**]**# **./**flink list

Waiting **for** response**...**

**------------------** Running**/**Restarting Jobs **-------------------**

05**.**05**.**2020 09**:**19**:**23 **:** 587ac4e69b32a11c72778aa5a42124c9 **:** Flink Streaming **Job** **(**RUNNING**)**

**--------------------------------------------------------------**

No scheduled jobs**.**

*//取消正在运行的job*

**[**root@spark2 bin**]**# **./**flink cancel 587ac4e69b32a11c72778aa5a42124c9

Cancelling job 587ac4e69b32a11c72778aa5a42124c9**.**

Cancelled job 587ac4e69b32a11c72778aa5a42124c9**.**

*//再次查看所有正在运行的job，发现没有了*

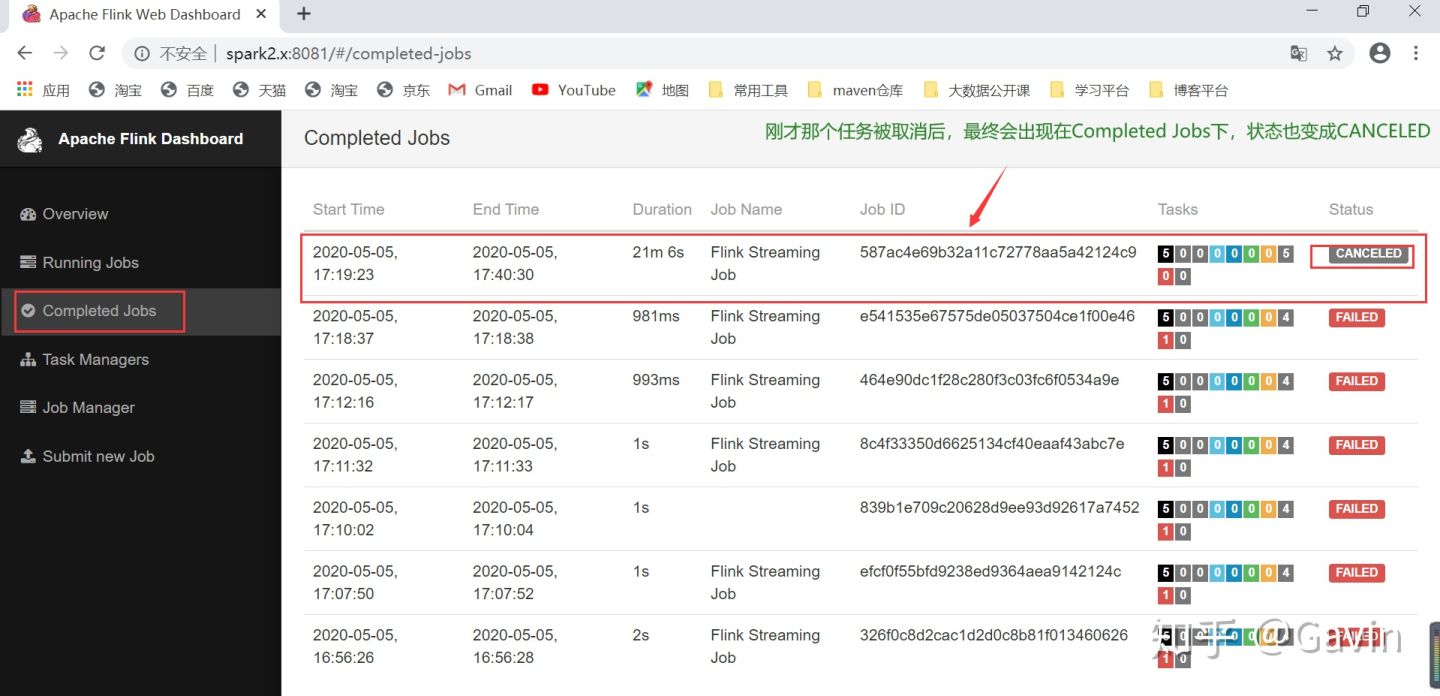
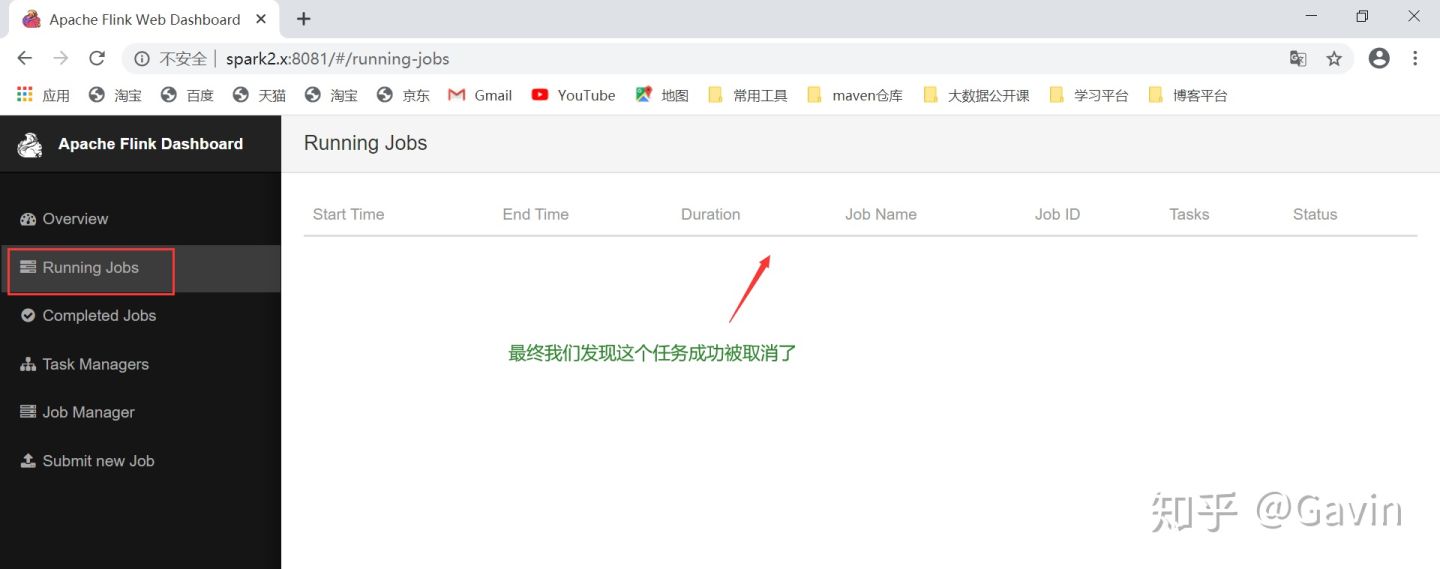
**[**root@spark2 bin**]**# **./**flink list

Waiting **for** response**...**

No running jobs**.**

No scheduled jobs**.**

**[**root@spark2 bin**]**#



当然，也可以查看被取消的Jobs有哪些，可以执行这条命令查看

**[**root@spark2 bin**]**# **./**flink list **--**all

Waiting **for** response**...**

No running jobs**.**

No scheduled jobs**.**

**----------------------** Terminated Jobs **-----------------------**

05**.**05**.**2020 08**:**56**:**26 **:** 326f0c8d2cac1d2d0c8b81f013460626 **:** Flink Streaming **Job** **(**FAILED**)**

05**.**05**.**2020 09**:**07**:**50 **:** efcf0f55bfd9238ed9364aea9142124c **:** Flink Streaming **Job** **(**FAILED**)**

05**.**05**.**2020 09**:**10**:**02 **:** 839b1e709c20628d9ee93d92617a7452 **:** **(**FAILED**)**

05**.**05**.**2020 09**:**11**:**32 **:** 8c4f33350d6625134cf40eaaf43abc7e **:** Flink Streaming **Job** **(**FAILED**)**

05**.**05**.**2020 09**:**12**:**16 **:** 464e90dc1f28c280f3c03fc6f0534a9e **:** Flink Streaming **Job** **(**FAILED**)**

05**.**05**.**2020 09**:**18**:**37 **:** e541535e67575de05037504ce1f00e46 **:** Flink Streaming **Job** **(**FAILED**)**

05**.**05**.**2020 09**:**19**:**23 **:** 587ac4e69b32a11c72778aa5a42124c9 **:** Flink Streaming **Job** **(**CANCELED**)**

**--------------------------------------------------------------**

**[**root@spark2 bin**]**#

**2、Yarn 模式**

以 Yarn 模式部署 Flink 任务时，要求 Flink 是有 Hadoop 支持的版本，Hadoop   
环境需要保证版本在 2.2 以上，并且集群中安装有 HDFS 服务。

（1）启动 hadoop 集群

**[**root@spark2 bin**]**# start**-**all**.**sh

This script is Deprecated**.** Instead use start**-**dfs**.**sh and start**-**yarn**.**sh

Starting namenodes on **[**spark2**.**x**]**

spark2**.**x**:** starting namenode**,** logging to **/**usr**/**local**/**etc**/**hadoop**/**module**/**hadoop**-**2**.**7**.**2**/**logs**/**hadoop**-**root**-**namenode**-**spark2**.**x**.**out

localhost: starting datanode**,** logging to **/**usr**/**local**/**etc**/**hadoop**/**module**/**hadoop**-**2**.**7**.**2**/**logs**/**hadoop**-**root**-**datanode**-**spark2**.**x**.**out

Starting secondary namenodes **[**0**.**0**.**0**.**0**]**

0**.**0**.**0**.**0**:** starting secondarynamenode**,** logging to **/**usr**/**local**/**etc**/**hadoop**/**module**/**hadoop**-**2**.**7**.**2**/**logs**/**hadoop**-**root**-**secondarynamenode**-**spark2**.**x**.**out

starting yarn daemons

starting resourcemanager**,** logging to **/**usr**/**local**/**etc**/**hadoop**/**module**/**hadoop**-**2**.**7**.**2**/**logs**/**yarn**-**root**-**resourcemanager**-**spark2**.**x**.**out

localhost: starting nodemanager**,** logging to **/**usr**/**local**/**etc**/**hadoop**/**module**/**hadoop**-**2**.**7**.**2**/**logs**/**yarn**-**root**-**nodemanager**-**spark2**.**x**.**out

**[**root@spark2 bin**]**#

（2）启动*启动 yarn-session*

**[**root@spark2 bin**]**# **./**yarn**-**session**.**sh **-**n 2 **-**s 2 **-**jm 1024 **-**tm 1024 **-**nm test **-**d

Error: A JNI error has occurred**,** please check your installation and **try** again

Exception in thread "main" java**.**lang**.**NoClassDefFoundError**:** org**/**apache**/**hadoop**/**conf**/**Configuration

at java**.**lang**.**Class**.**getDeclaredMethods0**(**Native Method**)**

at java**.**lang**.**Class**.**privateGetDeclaredMethods**(**Class**.**java**:**2701**)**

at java**.**lang**.**Class**.**privateGetMethodRecursive**(**Class**.**java**:**3048**)**

at java**.**lang**.**Class**.**getMethod0**(**Class**.**java**:**3018**)**

at java**.**lang**.**Class**.**getMethod**(**Class**.**java**:**1784**)**

at sun**.**launcher**.**LauncherHelper**.**validateMainClass**(**LauncherHelper**.**java**:**544**)**

at sun**.**launcher**.**LauncherHelper**.**checkAndLoadMain**(**LauncherHelper**.**java**:**526**)**

Caused by**:** java**.**lang**.**ClassNotFoundException**:** org**.**apache**.**hadoop**.**conf**.**Configuration

at java**.**net**.**URLClassLoader**.**findClass**(**URLClassLoader**.**java**:**382**)**

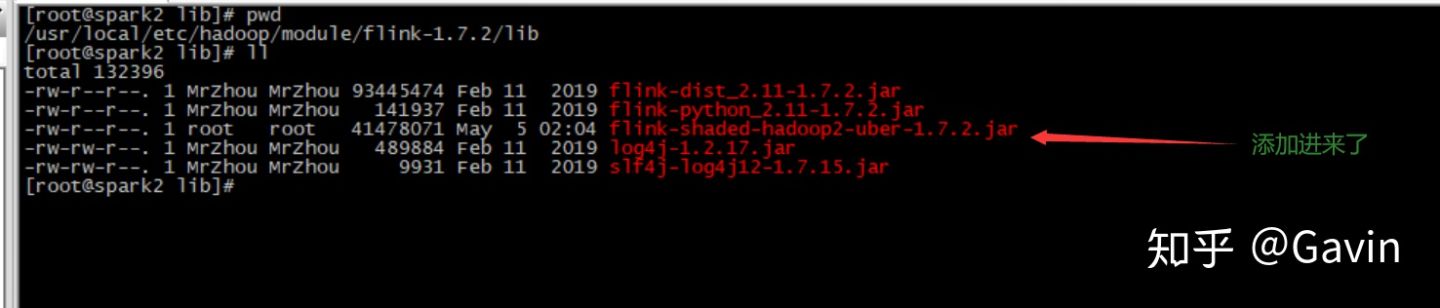
at java**.**lang**.**ClassLoader**.**loadClass**(**ClassLoader**.**java**:**424**)**

at sun**.**misc**.**Launcher$AppClassLoader**.**loadClass**(**Launcher**.**java**:**349**)**

at java**.**lang**.**ClassLoader**.**loadClass**(**ClassLoader**.**java**:**357**)**

**...** 7 more

报错，发现flink的lib下少了**flink-shaded-hadoop2-uber-1.7.2.jar**包



再次去启动，没问题的

**[**root@spark2 bin**]**# **./**yarn**-**session**.**sh **-**n 2 **-**s 2 **-**jm 1024 **-**tm 1024 **-**nm test **-**d

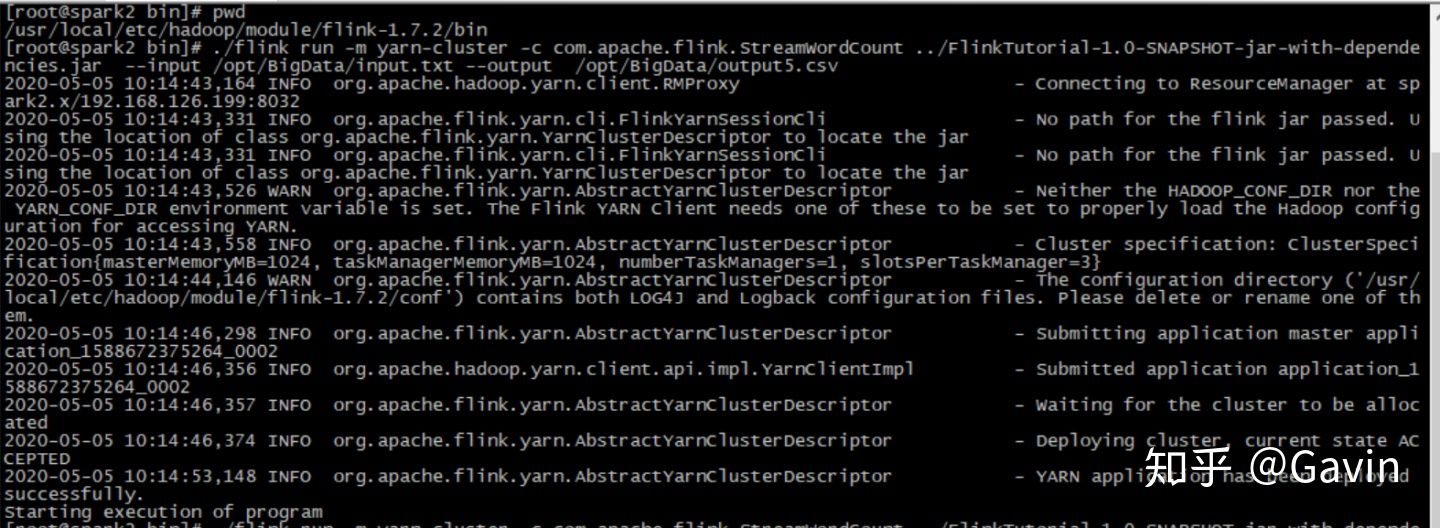
其中：

**-n(--container)**：TaskManager 的数量。   
**-s(--slots)**： 每个 TaskManager 的 slot 数量，默认一个 slot 一个 core，默认每个   
taskmanager 的 slot 的个数为 1，有时可以多一些 taskmanager，做冗余。   
**-jm**：JobManager 的内存（单位 MB)。   
**-tm**：每个 taskmanager 的内存（单位 MB)。   
**-nm**：yarn 的 appName(现在 yarn 的 ui 上的名字)。   
**-d**：后台执行

（3）执行任务

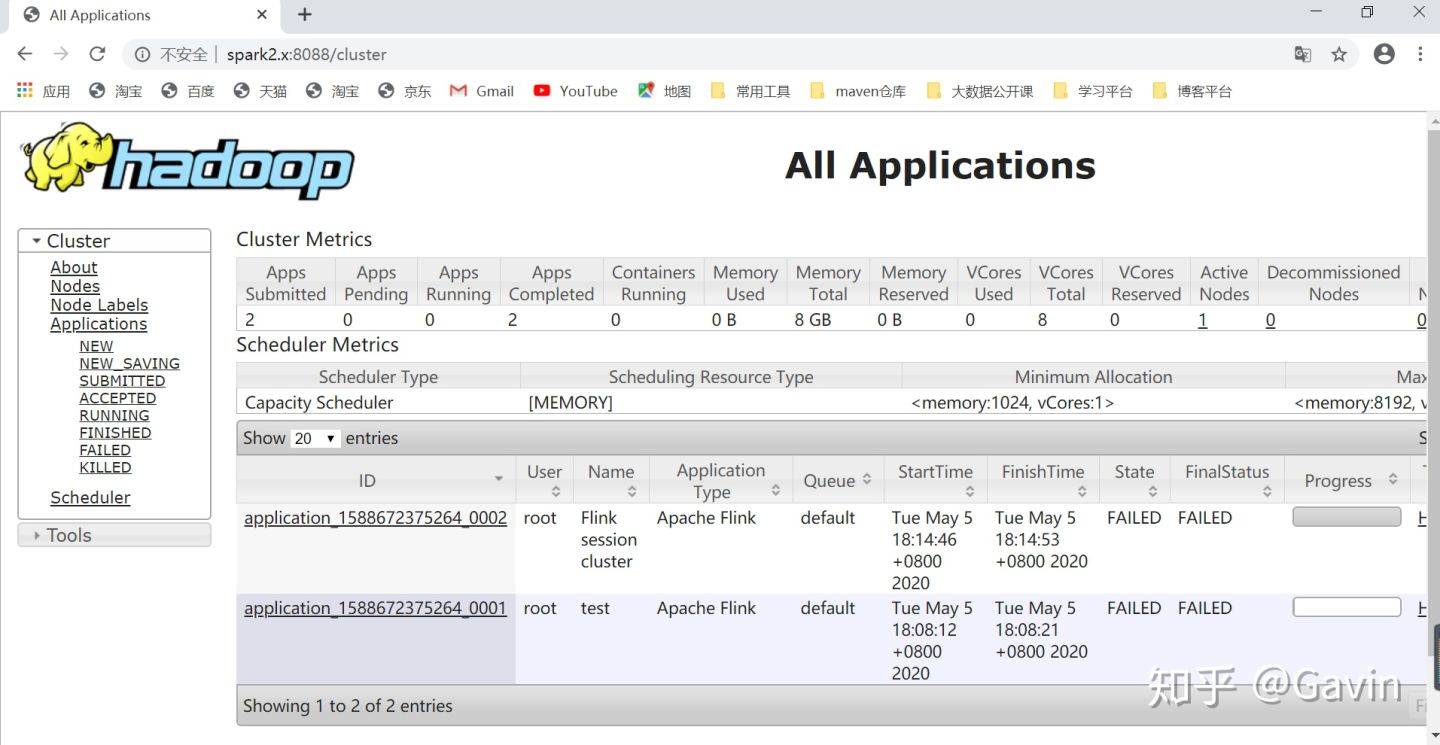
[root@spark2 bin]# ./flink run -m yarn-cluster -c com.apache.flink.StreamWordCount ../FlinkTutorial-1.0-SNAPSHOT-jar-with-dependencies.jar --input /opt/BigData/input.txt --output /opt/BigData/output5.csv

加载过程，如图所示：



（4）去 yarn 控制台查看任务状态

[http://spark2.x:8088](https://link.zhihu.com/?target=http%3A//spark2.x%3A8088/cluster)



**3、Kubernetes 部署**

容器化部署时目前业界很流行的一项技术，基于 Docker 镜像运行能够让用户更   
加 方 便地 对应 用进 行管 理 和运 维。 容器 管理 工 具中 最为 流行 的就 是 Kubernetes   
（k8s），而 Flink 也在最近的版本中支持了 k8s 部署模式。

（1）搭建 Kubernetes 集群（略）

（2）配置各组件的 yaml 文件

在 k8s 上构建 Flink Session Cluster，需要将 Flink 集群的组件对应的 docker 镜像   
分别在 k8s 上启动，包括 JobManager、TaskManager、JobManagerService 三个镜像   
服务。每个镜像服务都可以从中央镜像仓库中获取。

（3）启动 Flink Session Cluster

*// 启动 jobmanager-service 服务*

kubectl create **-**f jobmanager**-**service**.**yaml

*// 启动 jobmanager-deployment 服务*

kubectl create **-**f jobmanager**-**deployment**.**yaml

*// 启动 taskmanager-deployment 服务*

kubectl create **-**f taskmanager**-**deployment**.**yaml

（4）访问 Flink UI 页面

集群启动后，就可以通过 JobManagerServicers 中配置的 WebUI 端口，用浏览器   
输入以下 url 来访问 Flink UI 页面了：   
http://*{JobManagerHost:Port}*/api/v1/namespaces/default/services/flink-jobmanage   
r:ui/proxy

**四、 Flink 运行架构**

**1、 Flink 运行时的组件**

Flink 运行时架构主要包括四个不同的组件，它们会在运行流处理应用程序时协同工作：   
作业管理器（JobManager）、资源管理器（ResourceManager）、任务管理器（TaskManager），以及分发器（Dispatcher）。因为 Flink 是用 Java 和 Scala 实现的，所以所有组件都会运行在

Java 虚拟机上。每个组件的职责如下：

* **作业管理器（JobManager）**

控制一个应用程序执行的主进程，也就是说，每个应用程序都会被一个不同的   
JobManager 所控制执行。JobManager 会先接收到要执行的应用程序，这个应用程序会包括： 作业图（JobGraph）、逻辑数据流图（logical dataflow graph）和打包了所有的类、库和其它 资源的 JAR 包。JobManager 会把 JobGraph 转换成一个物理层面的数据流图，这个图被叫做 “执行图”（ExecutionGraph），包含了所有可以并发执行的任务。JobManager 会向资源管 理器（ResourceManager）请求执行任务必要的资源，也就是任务管理器（TaskManager）上 的插槽（slot）。一旦它获取到了足够的资源，就会将执行图分发到真正运行它们的 TaskManager 上。而在运行过程中，JobManager 会负责所有需要中央协调的操作，比如说检 查点（checkpoints）的协调。

* **资源管理器（ResourceManager）**

主要负责管理任务管理器（TaskManager）的插槽（slot），TaskManger 插槽是 Flink 中 定义的处理资源单元。Flink 为不同的环境和资源管理工具提供了不同资源管理器，比如   
YARN、Mesos、K8s，以及 standalone 部署。当 JobManager 申请插槽资源时，ResourceManager 会将有空闲插槽的 TaskManager 分配给 JobManager。如果 ResourceManager 没有足够的插槽 来满足 JobManager 的请求，它还可以向资源提供平台发起会话，以提供启动 TaskManager 进程的容器。另外，ResourceManager 还负责终止空闲的 TaskManager，释放计算资源。

* **任务管理器（TaskManager）**

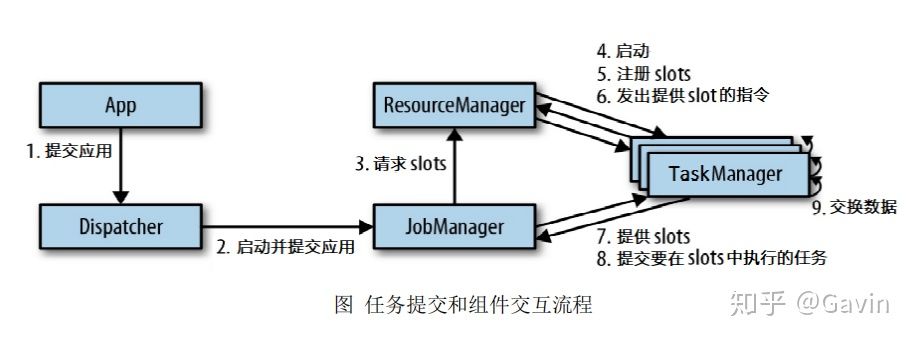
Flink 中的工作进程。通常在 Flink 中会有多个 TaskManager 运行，每一个 TaskManager都包含了一定数量的插槽（slots）。插槽的数量限制了 TaskManager 能够执行的任务数量。   
启动之后，TaskManager 会向资源管理器注册它的插槽；收到资源管理器的指令后，   
TaskManager 就会将一个或者多个插槽提供给 JobManager 调用。JobManager 就可以向插槽 分配任务（tasks）来执行了。在执行过程中，一个 TaskManager 可以跟其它运行同一应用程 序的 TaskManager 交换数据。

* **分发器（Dispatcher）**

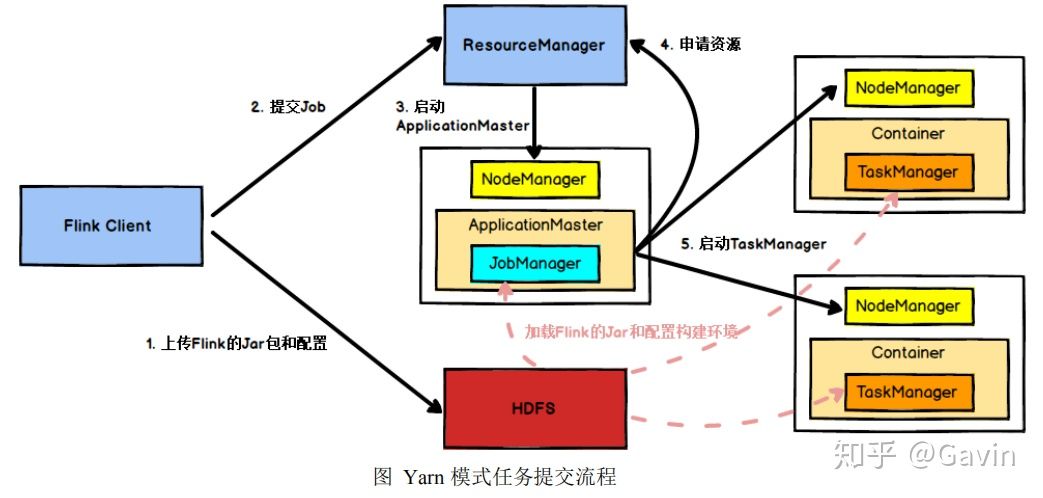
可以跨作业运行，它为应用提交提供了 REST 接口。当一个应用被提交执行时，分发器   
就会启动并将应用移交给一个 JobManager。由于是 REST 接口，所以 Dispatcher 可以作为集 群的一个 HTTP 接入点，这样就能够不受防火墙阻挡。Dispatcher 也会启动一个 Web UI，用 来方便地展示和监控作业执行的信息。Dispatcher 在架构中可能并不是必需的，这取决于应 用提交运行的方式。

**2、任务提交流程**

我们来看看当一个应用提交执行时，Flink 的各个组件是如何交互协作的：

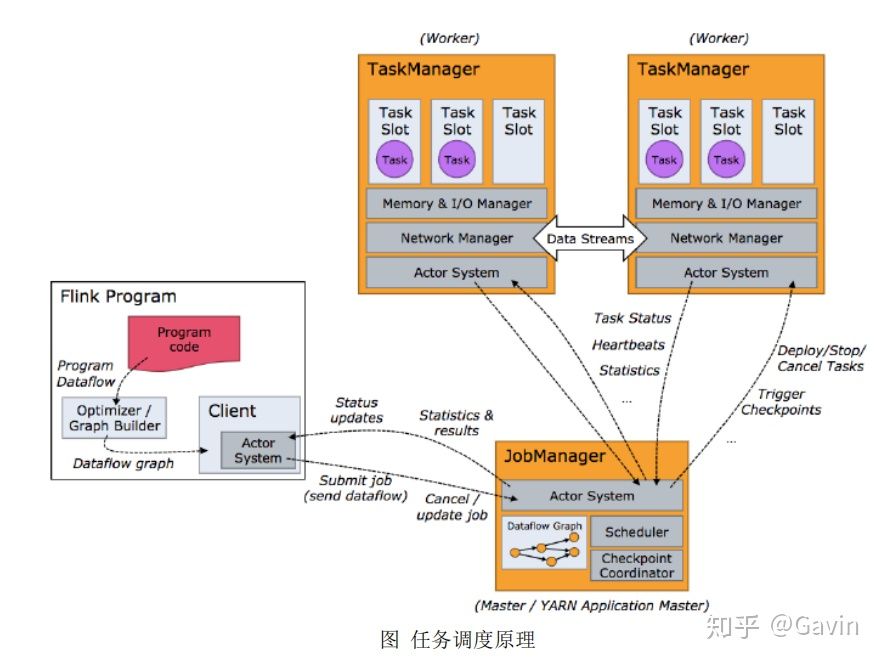


上图是从一个较为高层级的视角，来看应用中各组件的交互协作。如果部署的集群环境   
不同（例如 YARN，Mesos，Kubernetes，standalone 等），其中一些步骤可以被省略，或是 有些组件会运行在同一个 JVM 进程中。 具体地，如果我们将 Flink 集群部署到 YARN 上，那么就会有如下的提交流程：



Flink 任务提交后，Client 向 HDFS 上传 Flink 的 Jar 包和配置，之后向 Yarn    
ResourceManager 提交任务，ResourceManager 分配 Container 资源并通知对应的   
NodeManager 启动 ApplicationMaster，ApplicationMaster 启动后加载 Flink 的 Jar 包 和配置构建环境，然后启动 JobManager，之后 ApplicationMaster 向 ResourceManager 申请资源启动 TaskManager ， ResourceManager 分 配 Container 资 源 后 ， 由 ApplicationMaster 通 知 资 源 所 在 节 点 的 NodeManager 启 动 TaskManager ， NodeManager 加载 Flink 的 Jar 包和配置构建环境并启动 TaskManager，TaskManager 启动后向 JobManager 发送心跳包，并等待 JobManager 向其分配任务。

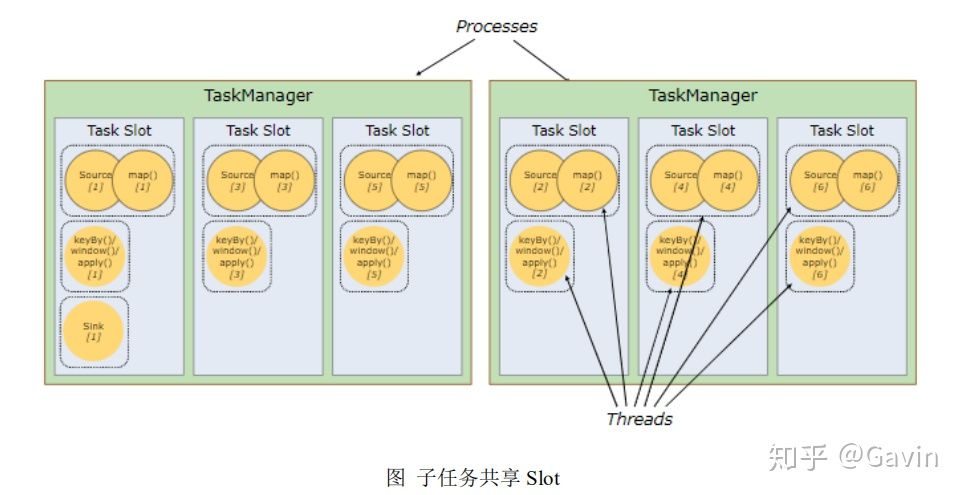
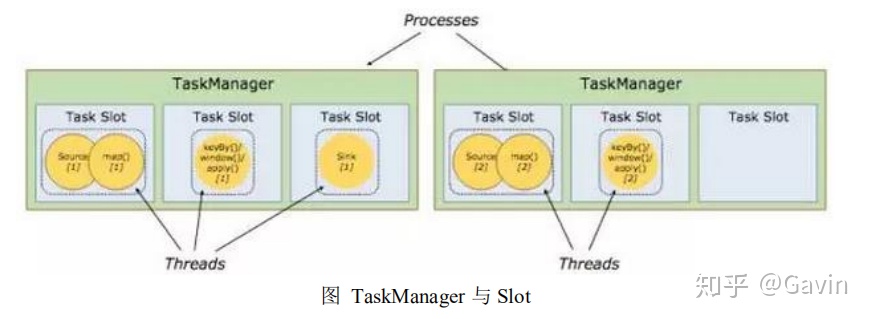
**3、任务调度原理**



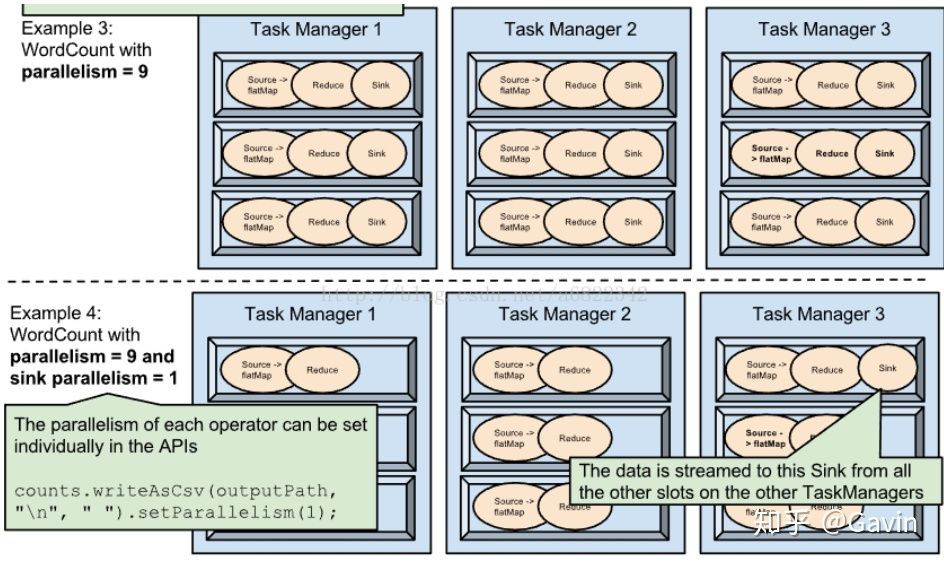
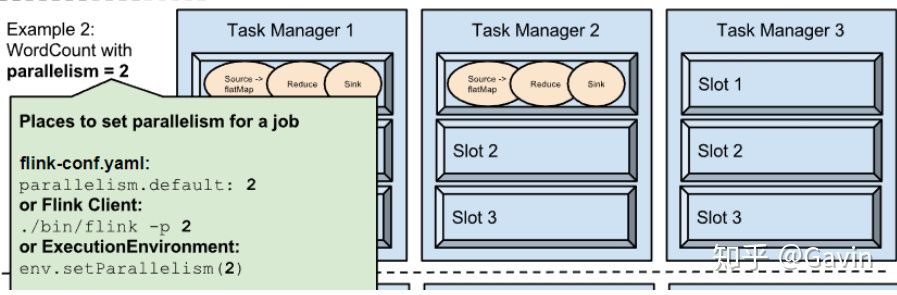
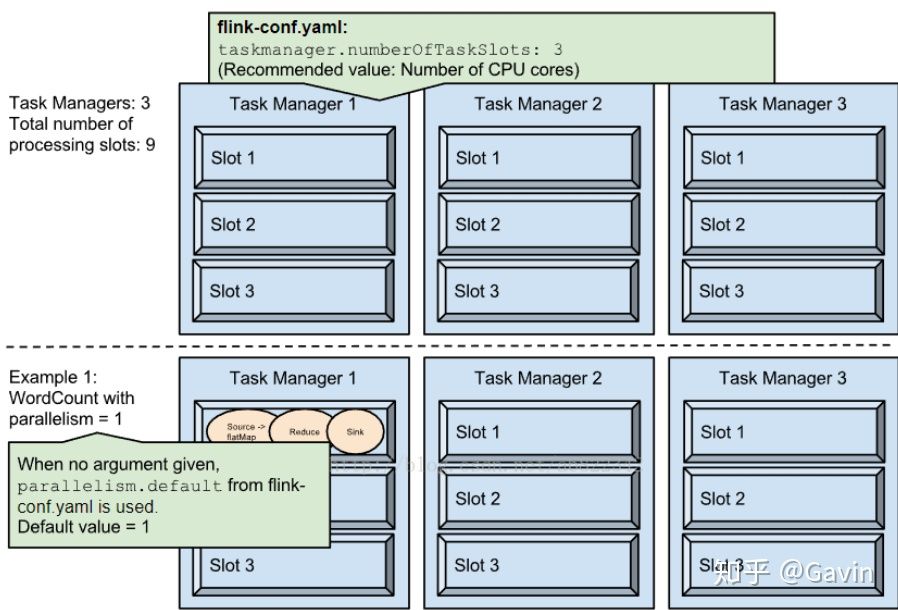
客户端不是运行时和程序执行的一部分，但它用于准备并发送   
dataflow(JobGraph)给 Master(JobManager)，然后，客户端断开连接或者维持连接以   
等待接收计算结果。   
当 Flink 集 群 启 动 后 ， 首 先 会 启 动 一 个 JobManger 和一个或多个的 TaskManager。  
由 Client 提交任务给 JobManager，JobManager 再调度任务到各个 TaskManager 去执行，然后 TaskManager 将心跳和统计信息汇报给 JobManager。   
TaskManager 之间以流的形式进行数据的传输。上述三者均为独立的 JVM 进程。   
**Client**为提交 Job 的客户端，可以是运行在任何机器上（与 JobManager 环境   
连通即可）。  
提交 Job 后，Client 可以结束进程（Streaming 的任务），也可以不   
结束并等待结果返回。   
**JobManager**主 要 负 责 调 度 Job 并 协 调 Task 做 checkpoint， 职 责 上 很 像   
Storm 的 Nimbus。从 Client 处接收到 Job 和 JAR 包等资源后，会生成优化后的   
执行计划，并以 Task 的单元调度到各个 TaskManager 去执行。   
**TaskManager**在启动的时候就设置好了槽位数（Slot），每个 slot 能启动一个   
Task，Task 为线程。  
从 JobManager 处接收需要部署的 Task，部署启动后，与自   
己的上游建立 Netty 连接，接收数据并处理。

* **TaskManger 与 Slots**

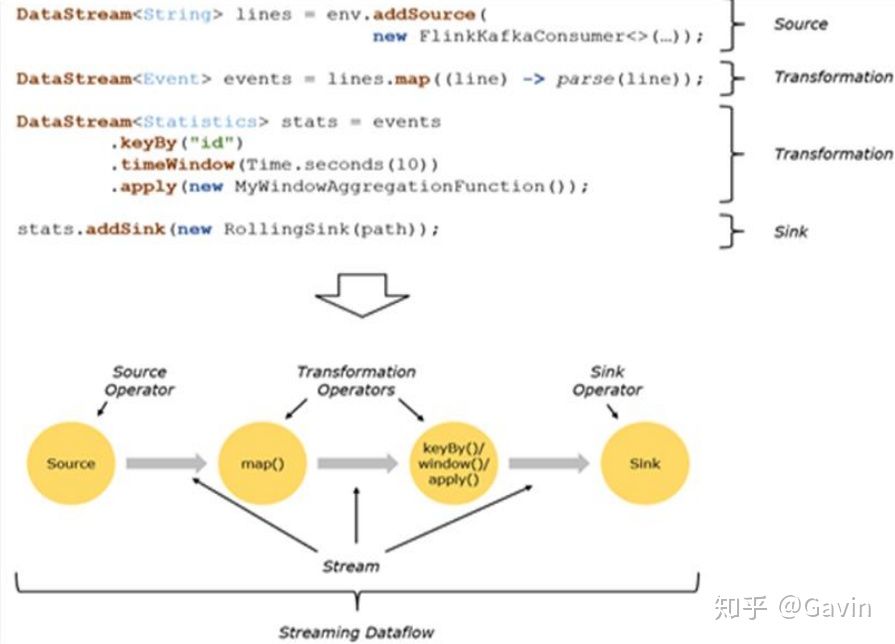
Flink 中每一个 worker(TaskManager)都是一个 **JVM 进程**，它可能会在**独立的线**  
**程**上执行一个或多个 subtask。为了控制一个 worker 能接收多少个 task，worker 通   
过 task slot 来进行控制（一个 worker 至少有一个 task slot）。   
每个 task slot 表示 TaskManager 拥有资源的**一个固定大小的子集**。假如一个 TaskManager 有三个 slot，那么它会将其管理的内存分成三份给各个 slot。资源 slot   
化意味着一个 subtask 将不需要跟来自其他 job 的 subtask 竞争被管理的内存，取而   
代之的是它将拥有一定数量的内存储备。需要注意的是，这里不会涉及到 CPU 的隔   
离，slot 目前仅仅用来隔离 task 的受管理的内存。   
通过调整 task slot 的数量，允许用户定义 subtask 之间如何互相隔离。如果一个   
TaskManager 一个 slot，那将意味着每个 task group 运行在独立的 JVM 中（该 JVM   
可能是通过一个特定的容器启动的），而一个 TaskManager 多个 slot 意味着更多的   
subtask 可以共享同一个 JVM。而在同一个 JVM 进程中的 task 将共享 TCP 连接（基   
于多路复用）和心跳消息。它们也可能共享数据集和数据结构，因此这减少了每个   
task 的负载。



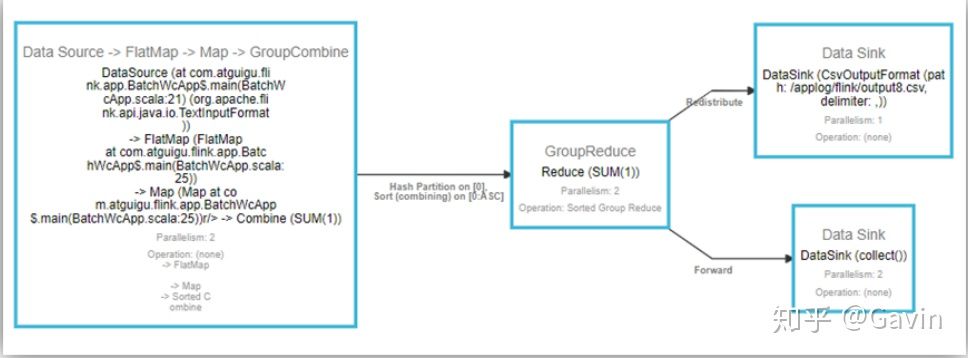
默认情况下，Flink 允许子任务共享 slot，即使它们是不同任务的子任务（前提   
是它们来自同一个 job）。 这样的结果是，一个 slot 可以保存作业的整个管道。   
**Task Slot 是静态的概念，是指 TaskManager 具有的并发执行能力**，可以通过   
参数 taskmanager.numberOfTaskSlots 进行配置；而**并行度 parallelism 是动态概念，**  
**即 TaskManager 运行程序时实际使用的并发能力**，可以通过参数 parallelism.default   
进行配置。   
也就是说，假设一共有 3 个 TaskManager，每一个 TaskManager 中的分配 3 个   
TaskSlot，也就是每个 TaskManager 可以接收 3 个 task，一共 9 个 TaskSlot，如果我   
们设置 parallelism.default=1，即运行程序默认的并行度为 1，9 个 TaskSlot 只用了 1   
个，有 8 个空闲，因此，设置合适的并行度才能提高效率。



* **程序与数据流（DataFlow）**

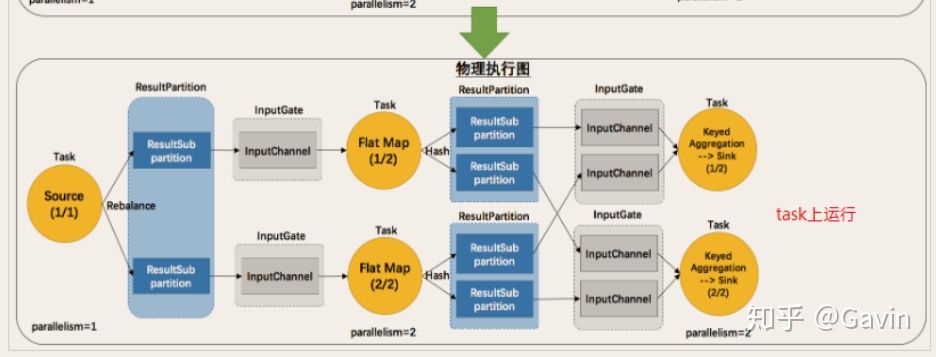
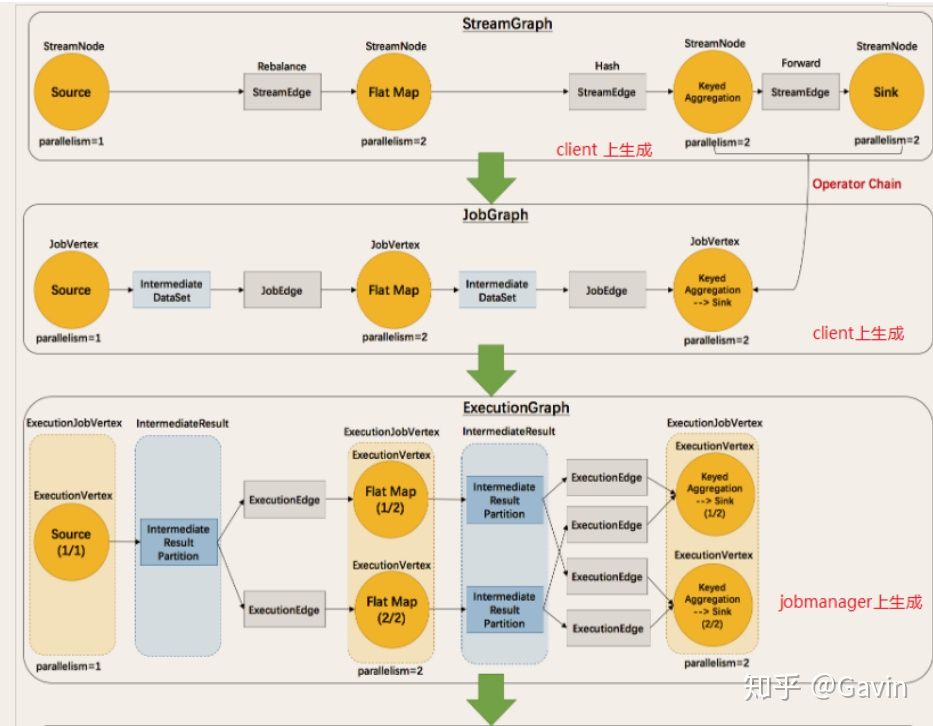


所有的 Flink 程序都是由三部分组成的： **Source**、**Transformation**和 **Sink**。   
Source 负责读取数据源，Transformation 利用各种算子进行处理加工，Sink 负   
责输出。  
在运行时，Flink 上运行的程序会被映射成“逻辑数据流”（dataflows），它包   
含了这三部分。**每一个 dataflow 以一个或多个 sources 开始以一个或多个 sinks 结**  
**束**。dataflow 类似于任意的有向无环图（DAG）。在大部分情况下，程序中的转换   
运算（transformations）跟 dataflow 中的算子（operator）是一一对应的关系，但有   
时候，一个 transformation 可能对应多个 operator。



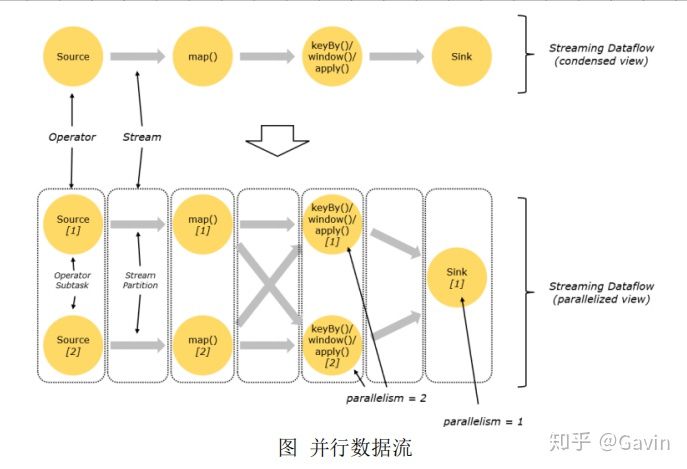
* **执行图（ExecutionGraph）**

由 Flink 程序直接映射成的数据流图是 StreamGraph，也被称为逻辑流图，因为   
它们表示的是计算逻辑的高级视图。为了执行一个流处理程序，Flink 需要将逻辑流   
图转换为物理数据流图（也叫执行图），详细说明程序的执行方式。   
Flink 中的执行图可以分成四层：StreamGraph -> JobGraph -> ExecutionGraph ->    
物理执行图。   
**StreamGraph**：是根据用户通过 Stream API 编写的代码生成的最初的图。用来   
表示程序的拓扑结构。   
**JobGraph**：StreamGraph 经过优化后生成了 JobGraph，提交给 JobManager 的   
数据结构。主要的优化为，将多个符合条件的节点 chain 在一起作为一个节点，这   
样可以减少数据在节点之间流动所需要的序列化/反序列化/传输消耗。   
**ExecutionGraph**： JobManager 根 据 JobGraph 生 成 ExecutionGraph 。   
ExecutionGraph 是 JobGraph 的并行化版本，是调度层最核心的数据结构。   
**物理执行图**：JobManager 根据 ExecutionGraph 对 Job 进行调度后，在各个   
TaskManager 上部署 Task 后形成的“图”，并不是一个具体的数据结构。



* **并行度（Parallelism）**

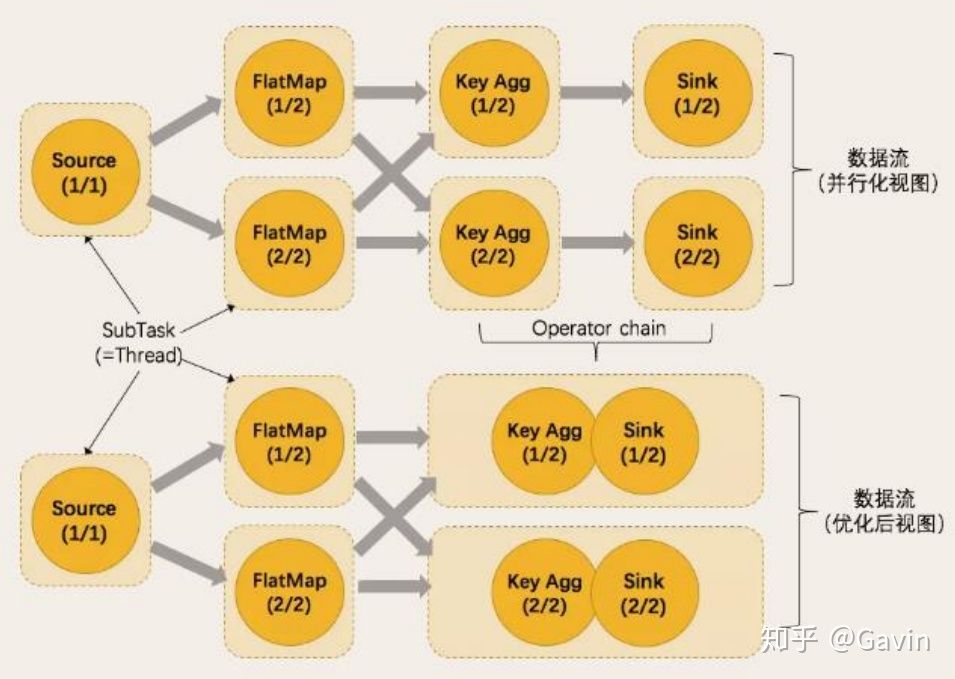
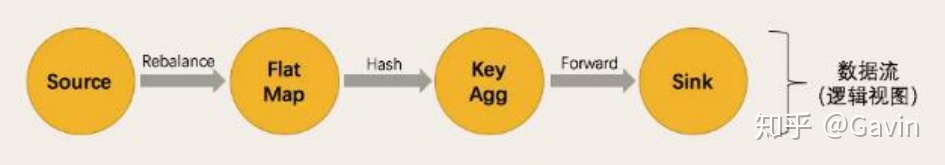
Flink 程序的执行具有**并行、分布式**的特性。   
在执行过程中，一个流（stream）包含一个或多个分区（stream partition），而   
每一个算子（operator）可以包含一个或多个子任务（operator subtask），这些子任   
务在不同的线程、不同的物理机或不同的容器中彼此互不依赖地执行。   
**一个特定算子的子任务（subtask）的个数被称之为其并行度（parallelism）**。   
一般情况下，一个流程序的并行度，可以认为就是其所有算子中最大的并行度。一   
个程序中，不同的算子可能具有不同的并行度。



Stream 在算子之间传输数据的形式可以是 one-to-one(forwarding)的模式也可以   
是 redistributing 的模式，具体是哪一种形式，取决于算子的种类。   
**One-to-one**：stream(比如在 source 和 map operator 之间)维护着分区以及元素的   
顺序。那意味着 map 算子的子任务看到的元素的个数以及顺序跟 source 算子的子   
任务生产的元素的个数、顺序相同，map、fliter、flatMap 等算子都是 one-to-one 的   
对应关系。   
¬ 类似于 spark 中的**窄依赖**  
**Redistributing**：stream(map()跟 keyBy/window 之间或者 keyBy/window 跟 sink   
之间)的分区会发生改变。每一个算子的子任务依据所选择的 transformation 发送数   
据到不同的目标任务。例如，keyBy() 基于 hashCode 重分区、broadcast 和 rebalance   
会随机重新分区，这些算子都会引起 redistribute 过程，而 redistribute 过程就类似于   
Spark 中的 shuffle 过程。   
¬ 类似于 spark 中的**宽依赖**

* **任务链（Operator Chains）**

**相同并行度的 one to one 操作**，Flink 这样相连的算子链接在一起形成一个 task，   
原来的算子成为里面的一部分。将算子链接成 task 是非常有效的优化：它能减少线   
程之间的切换和基于缓存区的数据交换，在减少时延的同时提升吞吐量。链接的行   
为可以在编程 API 中进行指定。



发布于 05-05