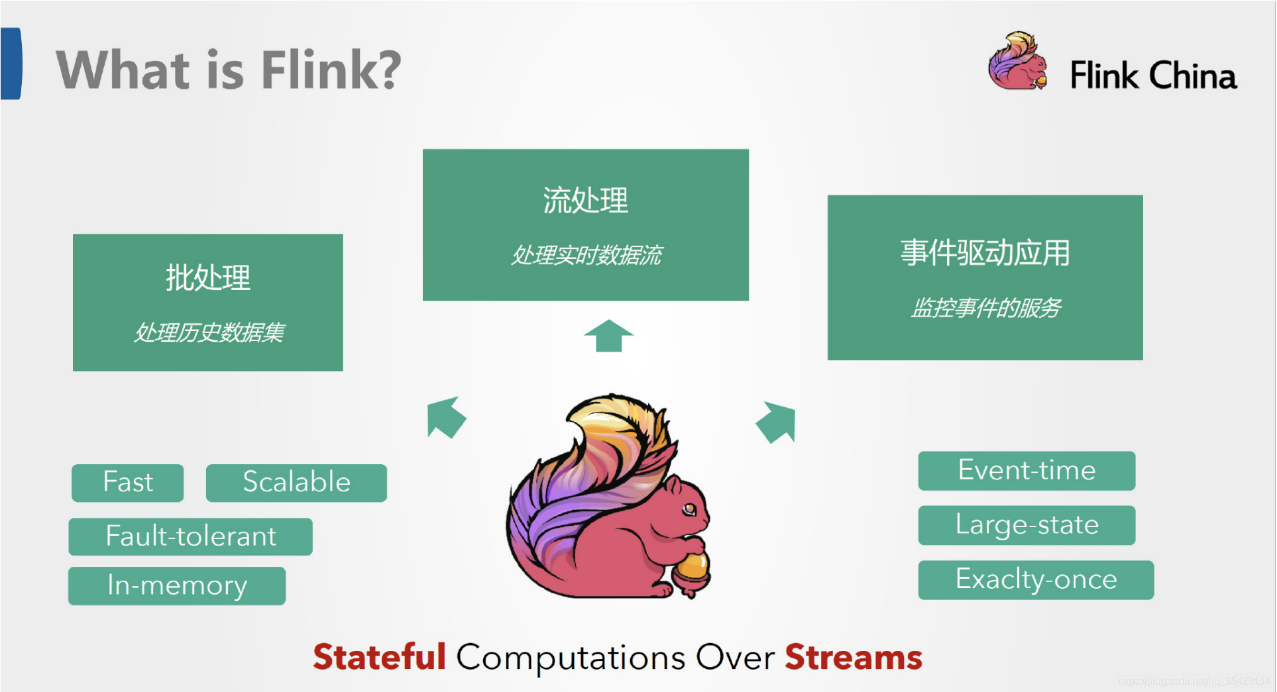
**[Apache Flink简介与安装](https://www.cnblogs.com/chq3272991/p/16670591.html)**

https://www.cnblogs.com/chq3272991/p/16670591.html

**什么是 Apache Flink？**

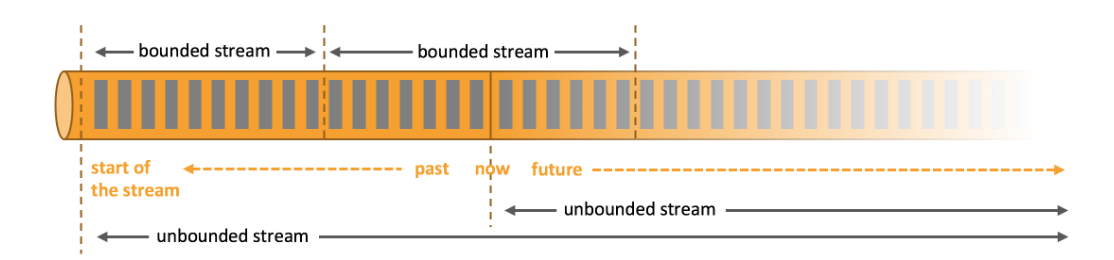
[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220909091606007-358139906.png)

Apache Flink是一个框架和分布式处理引擎，用于对无界数据和有界数据流进行状态计算。Flink被设计为在所有常见的集群环境中运行，以内存中的速度和任何规模执行计算。

**处理无界和有界数据**

任何类型的数据都是作为事件流产生的。信用卡交易、传感器测量、机器日志或网站或移动应用程序上的用户交互，所有这些数据都以流的形式生成。

数据可以作为无界或有界流进行处理。

1. 无界流有一个开始但没有定义的结束。它们不会终止并在生成数据时提供数据。必须连续处理无界流，即事件必须在被摄取后立即处理。不可能等待所有输入数据到达，因为输入是无界的并且不会在任何时间点完成。处理无界数据通常需要以特定顺序（例如事件发生的顺序）摄取事件，以便能够推断结果的完整性。
2. 有界流具有定义的开始和结束。可以通过在执行任何计算之前摄取所有数据来处理有界流。处理有界流不需要有序摄取，因为始终可以对有界数据集进行排序。有界流的处理也称为批处理。  
   [](https://img2022.cnblogs.com/blog/900293/202209/900293-20220908185948175-170917603.png)

**随处应用部署**[**#**](https://www.cnblogs.com/chq3272991/p/16670591.html#%E9%9A%8F%E5%A4%84%E5%BA%94%E7%94%A8%E9%83%A8%E7%BD%B2)

Apache Flink 是一个分布式系统，需要计算资源才能执行应用程序。Flink 集成了所有常见的集群资源管理器，例如Hadoop YARN和Kubernetes，但也可以设置为作为独立集群运行。

Flink 旨在很好地工作于前面列出的每个资源管理器。这是通过特定于资源管理器的部署模式实现的，该模式允许 Flink 以惯用的方式与每个资源管理器进行交互。

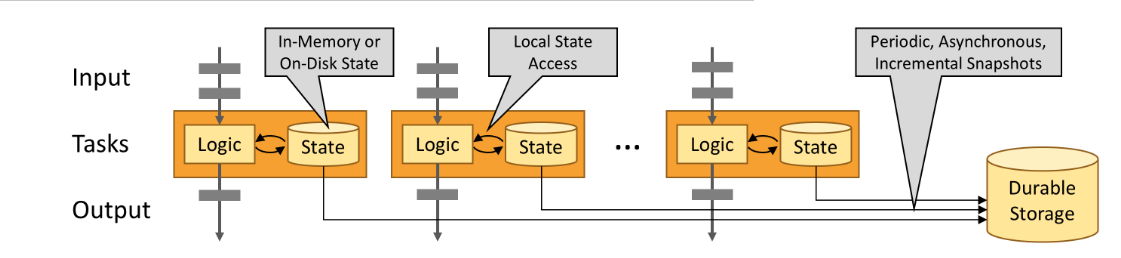
在部署 Flink 应用程序时，Flink 会根据应用程序配置的并行度自动识别所需资源，并向资源管理器请求。在失败的情况下，Flink 通过请求新资源来替换失败的容器。提交或控制应用程序的所有通信都是通过 REST 调用发生的。这简化了 Flink 在许多环境中的集成。

**以任何规模运行应用程序**[**#**](https://www.cnblogs.com/chq3272991/p/16670591.html#%E4%BB%A5%E4%BB%BB%E4%BD%95%E8%A7%84%E6%A8%A1%E8%BF%90%E8%A1%8C%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)

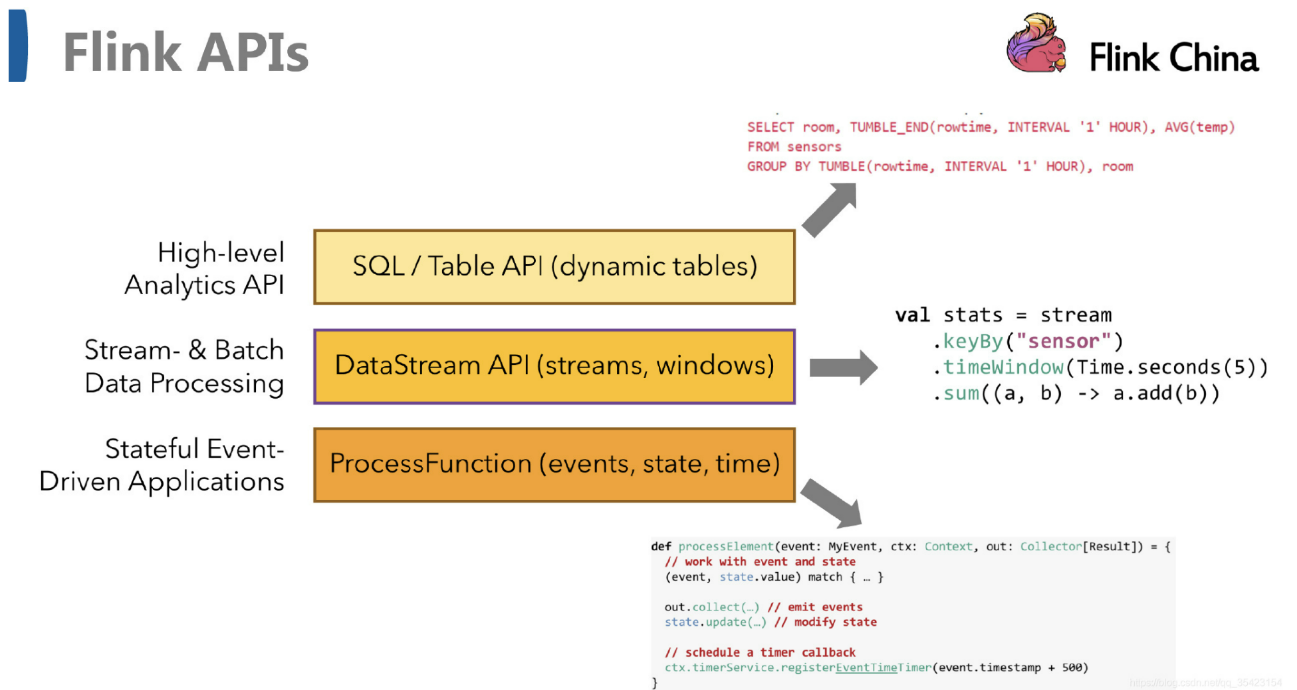
Flink 旨在以任何规模运行有状态的流应用程序。应用程序被并行化为可能在集群中分布和并发执行的数千个任务。因此，应用程序可以利用几乎无限量的 CPU、主内存、磁盘和网络 IO。而且，Flink 很容易维护非常大的应用程序状态。其异步和增量检查点算法确保对处理延迟的影响最小，同时保证精确一次的状态一致性。  
用户报告了在其生产环境中运行的 Flink 应用程序的可扩展性数据，例如

* 应用程序每天处理数万亿个事件，
* 维护多个 TB 状态的应用程序，以及
* 在数千个内核上运行的应用程序。

**利用内存性能[#](https://www.cnblogs.com/chq3272991/p/16670591.html" \l "%E5%88%A9%E7%94%A8%E5%86%85%E5%AD%98%E6%80%A7%E8%83%BD)**

有状态的 Flink 应用程序针对本地状态访问进行了优化。任务状态始终保存在内存中，或者如果状态大小超过可用内存，则保存在访问高效的磁盘数据结构中。因此，任务通过访问本地（通常是在内存中）状态来执行所有计算，从而产生非常低的处理延迟。Flink 通过定期和异步检查本地状态到持久存储来保证在发生故障时的一次性状态一致性。  
[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220908190744448-1712502710.png)

**分层API**[**#**](https://www.cnblogs.com/chq3272991/p/16670591.html#%E5%88%86%E5%B1%82api)

Flink 根据抽象程度分层，提供了三种不同的 API。每一种 API 在简洁性和表达力上有着不同的侧重，并且针对不同的应用场景。  
[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220909093428207-1414613883.png)

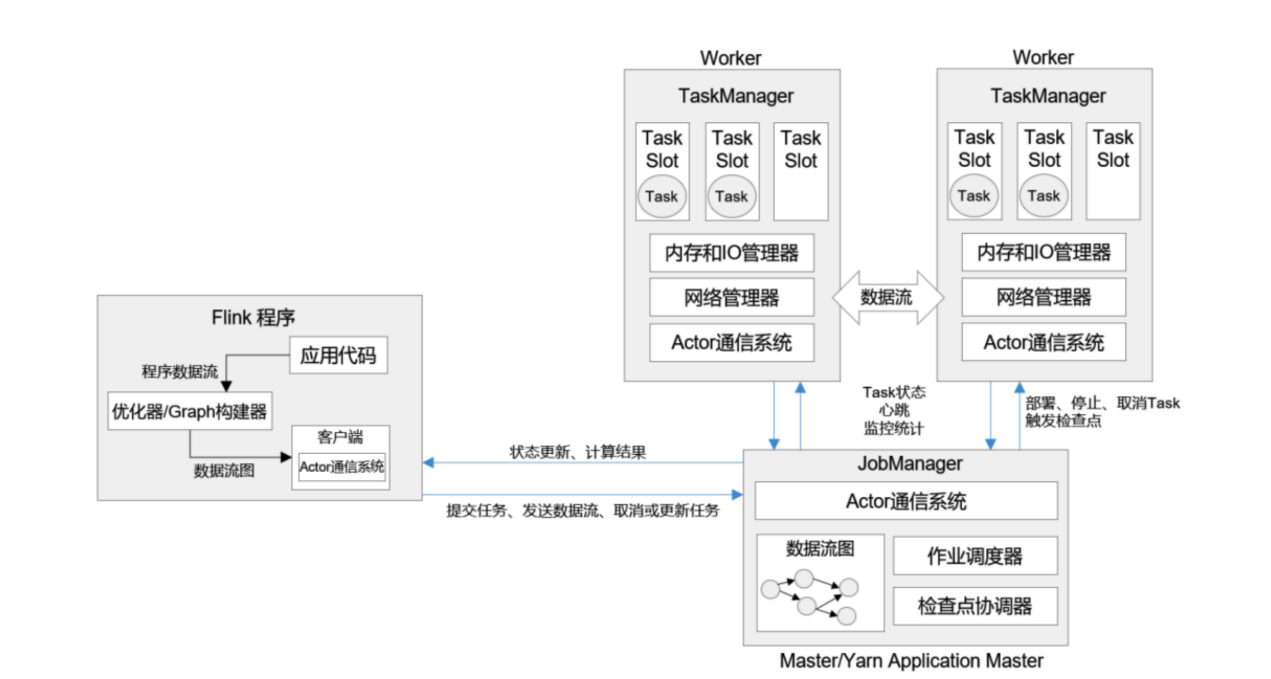
* ProcessFunction: 可以处理一或两条输入数据流中的单个事件或者归入一个特定窗口内的多个事件。它提供了对于时间和状态的细粒度控制。开发者可以在其中任意地修改状态，也能够注册定时器用以在未来的某一时刻触发回调函数。因此，你可以利用ProcessFunction实现许多有状态的事件驱动应用所需要的基于单个事件的复杂业务逻辑。
* DataStream API：为许多通用的流处理操作提供了处理原语。这些操作包括窗口、逐条记录的转换操作，在处理事件时进行外部数据库查询等。DataStream API支持Java和Scala语言，预先定义了例如map()、reduce()、aggregate()等函数。你可以通过扩展实现预定义接口或使用Java、Scala的lambda表达式实现自定义的函数。
* SQL & Table API: Flink支持两种关系型的API、Table API和SQL。这两个API都是批处理和流处理统一的API，这意味着在无边界的实时数据流和有边界的历史记录数据流上，关系型 API 会以相同的语义执行查询，并产生相同的结果。Table API和SQL借助了 Apache Calcite来进行查询的解析，校验以及优化。它们可以与DataStream和DataSet API无缝集成，并支持用户自定义的标量函数，聚合函数以及表值函数。Flink 的关系型 API 旨在简化数据分析、数据流水线和 ETL 应用的定义。

Flink VS Spark Streaming

* 数据模型  
  Flink基本数据模型是数据流，以及事件序列  
  Spark采用 RDD模型，Spark Streaming的DStream实际上也就是一组组小批数据RDD的集合
* 运行时架构  
  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是批计算，将DAG划分为不同的Stage，一个完成后才可以计算下一个。

**Flink系统架构[#](https://www.cnblogs.com/chq3272991/p/16670591.html" \l "flink%E7%B3%BB%E7%BB%9F%E6%9E%B6%E6%9E%84)**

Flink的运行时架构中，最重要的就是两大组件：作业管理器（JobManger）和任务管理器（TaskManager）。对于一个提交执行的作业，JobManager是真正意义上的“管理者”（Master）, 负责管理调度，所以在不考虑高可用的情况下只能有一个；而TaskManager是“工作者”（Worker、Slave）, 负责执行任务处理数据，所以可以有一个或多个。

[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220909142849100-131972740.png)

客户端并不是处理系统的一部分，它只负责作业的提交。具体来说，就是调用程序的 main 方法，将代码转换成“数据流图”（Dataflow Graph），并最终生成作业图（JobGraph），一并发送给 JobManager。提交之后，任务的执行其实就跟客户端没有关系了；

**作业管理器（JobManager）**[**#**](https://www.cnblogs.com/chq3272991/p/16670591.html#%E4%BD%9C%E4%B8%9A%E7%AE%A1%E7%90%86%E5%99%A8jobmanager)

JobManager 是一个 Flink 集群中任务管理和调度的核心，是控制应用执行的主进程。也就是说，每个应用都应该被唯一的 JobManager 所控制执行。JobManger 又包含 3 个不同的组件：

**JobMaster**

JobMaster 是 JobManager 中最核心的组件，负责处理单独的作业（Job）。所以 JobMaster和具体的 Job 是一一对应的，多个 Job 可以同时运行在一个 Flink 集群中, 每个 Job 都有一个自己的 JobMaster。需要注意在早期版本的 Flink 中，没有 JobMaster 的概念；而 JobManager的概念范围较小，实际指的就是现在所说的 JobMaster。

在作业提交时，JobMaster 会先接收到要执行的应用。这里所说“应用”一般是客户端提交来的，包括：Jar 包，数据流图（dataflow graph），和作业图（JobGraph）。

JobMaster 会把 JobGraph 转换成一个物理层面的数据流图，这个图被叫作“执行图”（ExecutionGraph），它包含了所有可以并发执行的任务。 JobMaster 会向资源管理器（ResourceManager）发出请求，申请执行任务必要的资源。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，JobMaster 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）**

ResourceManager 主要负责资源的分配和管理，在 Flink 集群中只有一个。所谓“资源”，主要是指 TaskManager 的任务槽（task slots）。任务槽就是 Flink 集群中的资源调配单元，包含了机器用来执行计算的一组 CPU 和内存资源。每一个任务（Task）都需要分配到一个 slot 上执行。

这里注意要把 Flink 内置的 ResourceManager 和其他资源管理平台（比如 YARN）的ResourceManager 区分开。

Flink 的 ResourceManager，针对不同的环境和资源管理平台（比如 Standalone 部署，或者YARN），有不同的具体实现。在 Standalone 部署时，因为 TaskManager 是单独启动的（没有Per-Job 模式），所以 ResourceManager 只能分发可用 TaskManager 的任务槽，不能单独启动新TaskManager。

而在有资源管理平台时，就不受此限制。当新的作业申请资源时，ResourceManager 会将有空闲槽位的 TaskManager 分配给 JobMaster。如果 ResourceManager 没有足够的任务槽，它还可以向资源提供平台发起会话，请求提供启动 TaskManager 进程的容器。另外，ResourceManager 还负责停掉空闲的 TaskManager，释放计算资源。

**分发器（Dispatcher）**

Dispatcher 主要负责提供一个 REST 接口，用来提交应用，并且负责为每一个新提交的作业启动一个新的 JobMaster 组件。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 在架构中并不是必需的，在不同的部署模式下可能会被忽略掉。

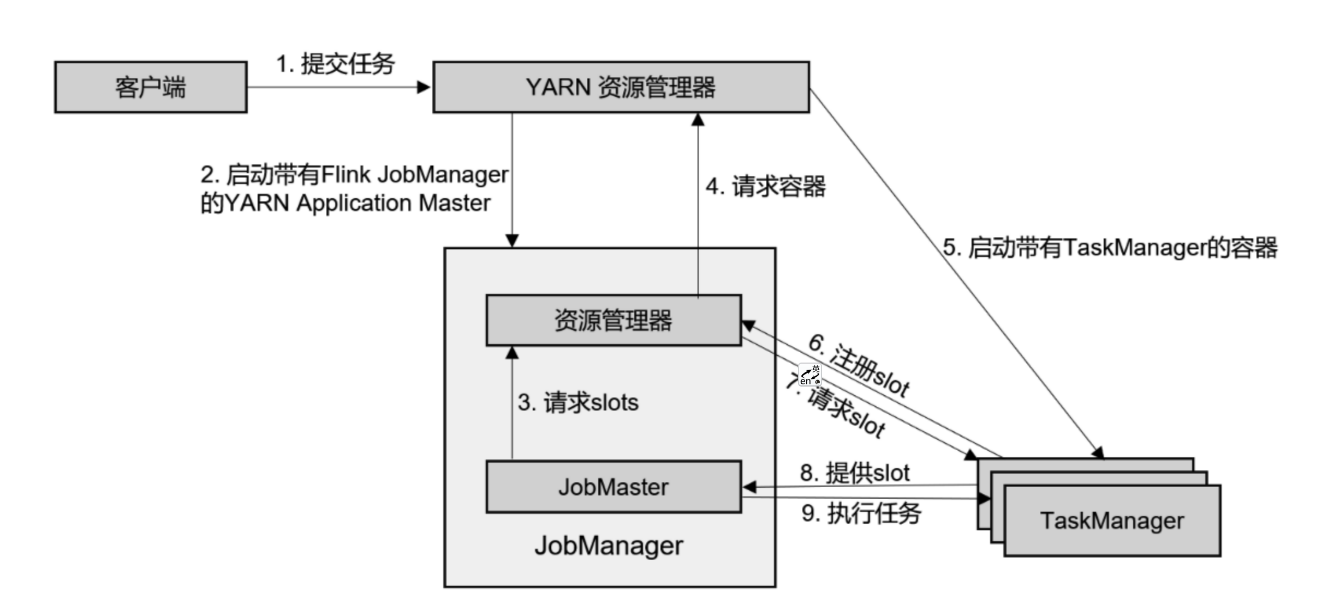
**任务管理器（TaskManager）**[**#**](https://www.cnblogs.com/chq3272991/p/16670591.html#%E4%BB%BB%E5%8A%A1%E7%AE%A1%E7%90%86%E5%99%A8taskmanager)

TaskManager 是 Flink 中的工作进程，数据流的具体计算就是它来做的，所以也被称为“Worker”。Flink 集群中必须至少有一个TaskManager；当然由于分布式计算的考虑，通常会有多个 TaskManager 运行，每一个 TaskManager 都包含了一定数量的任务槽（task slots）。Slot是资源调度的最小单位，slot 的数量限制了 TaskManager 能够并行处理的任务数量。

启动之后，TaskManager 会向资源管理器注册它的 slots；收到资源管理器的指令后，TaskManager 就会将一个或者多个槽位提供给 JobMaster 调用，JobMaster 就可以分配任务来执行了。

在执行过程中，TaskManager 可以缓冲数据，还可以跟其他运行同一应用的 TaskManager交换数据。

**YARN 集群 单作业（Per-Job）模式[#](https://www.cnblogs.com/chq3272991/p/16670591.html" \l "yarn-%E9%9B%86%E7%BE%A4-%E5%8D%95%E4%BD%9C%E4%B8%9Aper-job%E6%A8%A1%E5%BC%8F)**

[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220909144018774-1036383015.png)

（1）客户端将作业提交给 YARN 的资源管理器，这一步中会同时将 Flink 的 Jar 包和配置  
上传到 HDFS，以便后续启动 Flink 相关组件的容器。  
（2）YARN 的资源管理器分配 Container 资源，启动 Flink JobManager，并将作业提交给  
JobMaster。这里省略了 Dispatcher 组件。  
（3）JobMaster 向资源管理器请求资源（slots）。  
（4）资源管理器向 YARN 的资源管理器请求 container 资源。  
（5）YARN 启动新的 TaskManager 容器。  
（6）TaskManager 启动之后，向 Flink 的资源管理器注册自己的可用任务槽。  
（7）资源管理器通知 TaskManager 为新的作业提供 slots。  
（8）TaskManager 连接到对应的 JobMaster，提供 slots。  
（9）JobMaster 将需要执行的任务分发给 TaskManager，执行任务。  
可见，区别只在于 JobManager 的启动方式，以及省去了分发器。当第 2 步作业提交给JobMaster，之后的流程就与会话模式完全一样了。

**安装测试**[**#**](https://www.cnblogs.com/chq3272991/p/16670591.html#%E5%AE%89%E8%A3%85%E6%B5%8B%E8%AF%95)

检查java环境，需要java 11

tar -zxvf flink-1.14.5-bin-scala\_2.11.tgz -C /home/hp/

切换到flink目录下，

[hp@node-4 ~]$ ls

flink-1.14.5 hadoop-3.2.4 hadoop-3.2.4.tar.gz hadoop-data jdk1.8.0\_311 jdk-8u311-linux-x64.tar.gz spark-3.2.2-bin-hadoop3.2 test word.txt

其中，有几个文件夹需要特别了解的：

* bin/ 目录包含 flink 二进制文件以及几个管理各种作业和任务的 bash 脚本
* conf/ 目录包含一些配置文件，包括flink-conf.yaml
* examples/ 目录包含可以与 Flink 一起使用的示例应用程序

**启动和停止本地集群**

启动本地集群，请运行Flink附带的bash脚本：

./bin/start-cluster.sh

可以看到一些输出信息：

Starting cluster.

Starting standalonesession daemon on host node-4.

Starting taskexecutor daemon on host node-4.

此时，Flink作为后台进场运行，可以通过ps命令查看：

ps aux | grep flink

您应该能够导航到 localhost:8081 的 Web UI 以查看 Flink 仪表板并看到集群已启动并正在运行。  
要快速停止集群和所有正在运行的组件，您可以使用提供的脚本：

./bin/stop-cluster.sh

**提交Flink作业[#](https://www.cnblogs.com/chq3272991/p/16670591.html" \l "%E6%8F%90%E4%BA%A4flink%E4%BD%9C%E4%B8%9A)**

Flink 提供了一个 CLI 工具 bin/flink，它可以运行打包为 Java ARchives (JAR) 的程序并控制它们的执行。提交作业是指将作业的 JAR 文件和相关依赖上传到正在运行的 Flink 集群并执行。

Flink 版本附带示例作业，您可以在示例/文件夹中找到这些示例作业。

要将示例字数统计作业部署到正在运行的集群，请发出以下命令：

./bin/flink run examples/streaming/WordCount.jar

您可以通过查看日志来验证输出：

tail ./log/flink-hp-taskexecutor-0-node-4.out

(nymph,1)

(in,3)

(thy,1)

(orisons,1)

(be,4)

(all,2)

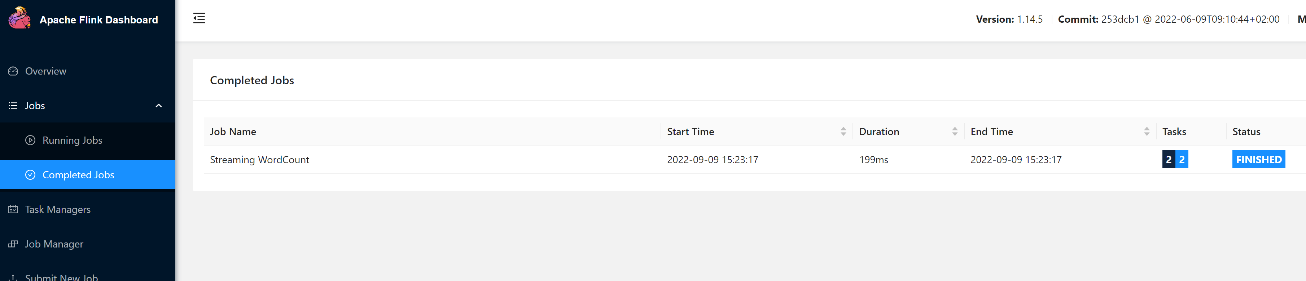
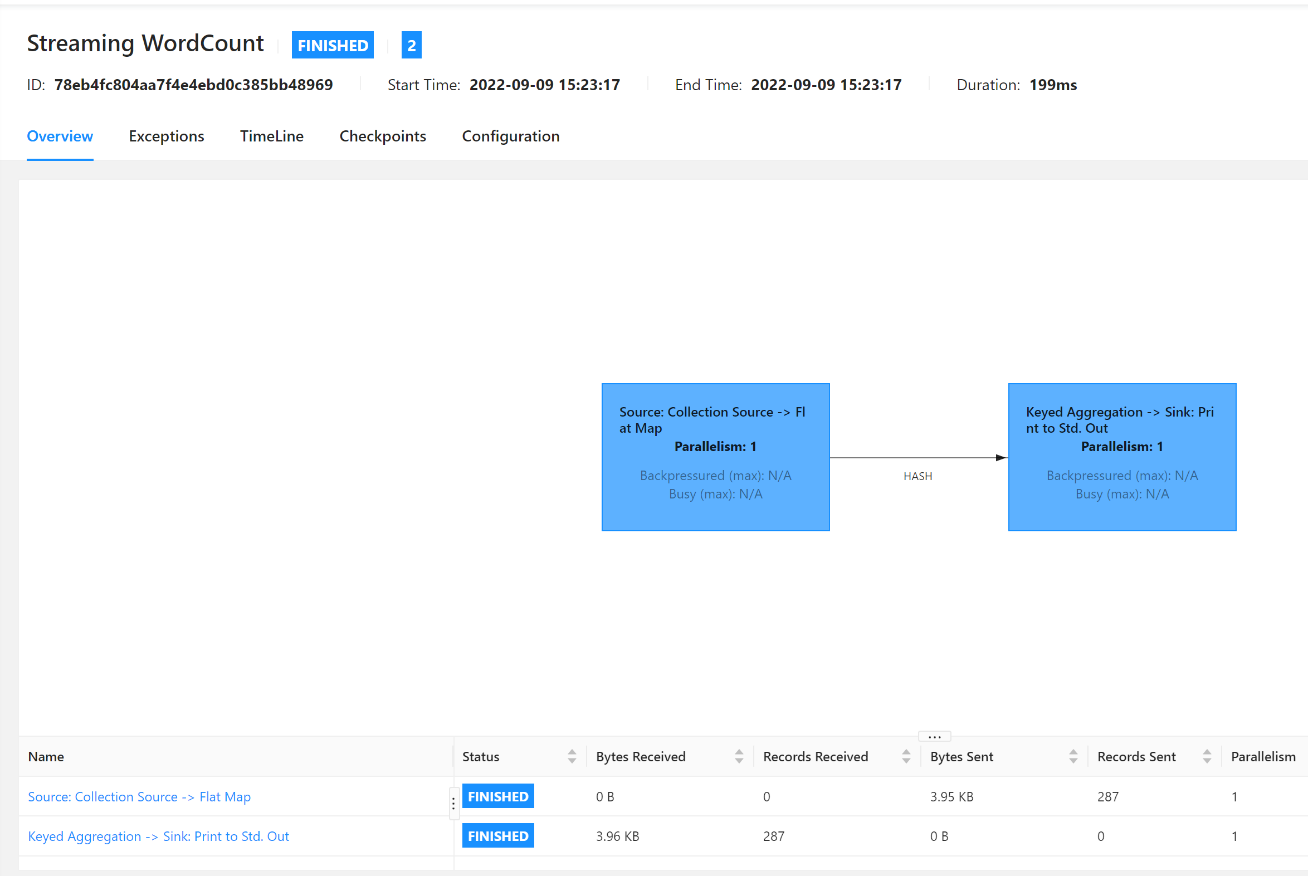
(my,1)

(sins,1)

(remember,1)

(d,4)

此外，您可以查看 Flink 的 Web UI 来监控集群的状态和正在运行的作业。

您可以查看执行的数据流计划：  
[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220909152337741-2016255719.png)  
[](https://img2022.cnblogs.com/blog/900293/202209/900293-20220909152407772-655606298.png)

参考：  
<https://flink.apache.org/flink-architecture.html>  
[https://nightlies.apache.org/flink/flink-docs-release-1.15//docs/try-flink/local\_installation/](https://nightlies.apache.org/flink/flink-docs-release-1.15/docs/try-flink/local_installation/)  
<https://blog.csdn.net/qq_35423154/article/details/113759891>  
<https://blog.csdn.net/weixin_47491957/article/details/124285190>