

# Flink-day1笔记

## Flink简介

### 前言

#### Apache Flink<sup>®</sup> — Stateful Computations over Data Streams

在数据流上做有状态的计算

实时计算

Apache Spark<sup>™</sup> is a unified analytics engine for large-scale data processing.

在大规模数据上进行统一分析处理

批处理 == 离线分析/数仓

大数据的飞速发展，也促使了各种计算引擎的诞生。2006年2月诞生的Hadoop中的MapReduce，2014年9月份诞生的Storm以及2014年2月诞生的Spark都有着各自专注的应用场景。特别是Spark开启了内存计算的先河，其计算速度比Hadoop的MapReduce快100倍，以此赢得了内存计算的飞速发展。或许因为早期人们对大数据的分析认知不够深刻，亦或许当时业务场景大都局限在批处理领域，从而使得Spark的火热或多或少的掩盖了其他分布式计算引擎的身影。但就在这个时候，有些分布式计算引擎也在默默发展着，直到2016年，人们才开始慢慢意识到流计算的重要性。

整个的分布式计算引擎中，通常被划分为三代

- **第一代** Hadoop的MapReduce做静态计算、Storm流计算。两套独立的计算引擎，使用难度比较大
- **第二代** Spark RDD做静态批处理、DStream|StructuredStreaming流计算；统一技术引擎，使用难度小
- **第三代** Flink DataStream做流计算、DataSet批处理；统一技术引擎，使用难度一般

## What is Apache Flink

<https://flink.apache.org/flink-architecture.html>

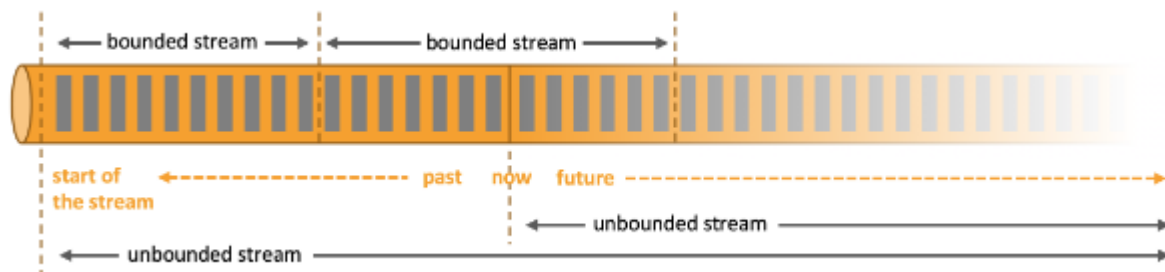
Apache Flink是2014年12月份诞生的一个流计算引擎。

Apache Flink是一个用于在无界和有界数据流上进行有状态计算的框架和分布式处理引擎。Flink被设计成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。

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.

Flink具有以下特点

1. 可以处理无界以及有界数据流

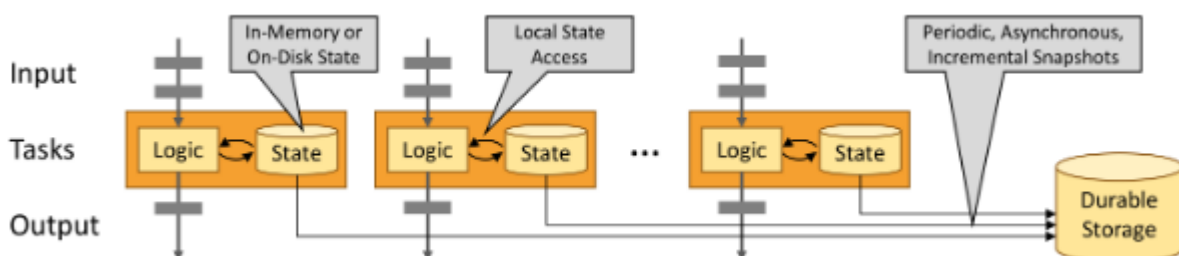


## 2. 随处部署应用程序

- 命令执行
- 远程部署
- 图形界面（比较常用的）

## 3. 以任何规模运行应用程序

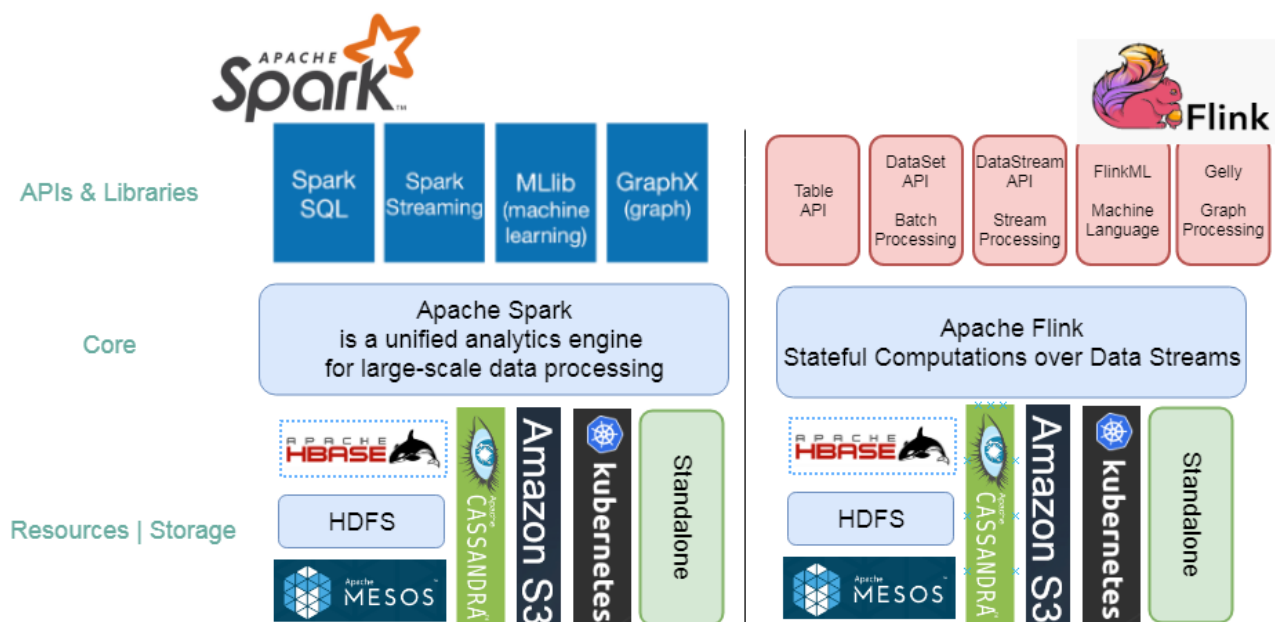
## 4. 充分利用内存性能



# Spark VS Flink

Flink与Spark设计理念恰好相反

- Spark底层计算是批处理模型，在批处理基础上模拟流，从而导致了流计算实时性较低
- Flink底层计算是连续的流计算模型，在流计算上模拟批处理，既可以保证流的实时性又可以实现批处理



## Flink应用场景

系统监控、舆情监控、交通预测、国家电网、疾病预测、金融行业风控、电商实时搜索优化等

## 环境安装

---

参考==>[flink安装.pdf](#)

1. 如果启动有问题，就通过日志信息查看问题原因 (flink\_home/log)
2. flink现在直接启动可以不需要hadoop
3. 在安装笔记中，需要写主机名的地方写主机的主机名

```
jobmanager.rpc.address: flink
taskmanager.numberOfTaskSlots: 1
parallelism.default: 3
```

主机名

### 5. 修改配置文件slaves

```
[root@flink conf]# p
/opt/install/flink-1
[root@flink conf]# v
```

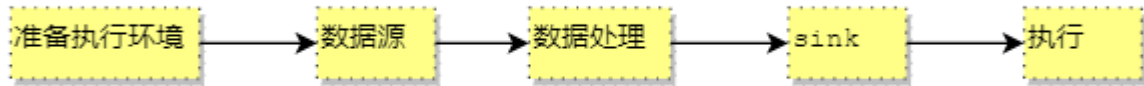
```
flink
```

主机名

## 快速入门

---

### 编程模型



## 代码

### 1. 导入依赖

```
<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-streaming-scala -->
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-streaming-scala_2.11</artifactId>
  <version>1.10.0</version>
</dependency>
```

### 2. 引入插件

```
<build>
  <plugins>
    <!--scala编译插件-->
    <plugin>
      <groupId>net.alchim31.maven</groupId>
      <artifactId>scala-maven-plugin</artifactId>
      <version>4.0.1</version>
      <executions>
        <execution>
          <id>scala-compile-first</id>
          <phase>process-resources</phase>
          <goals>
            <goal>add-source</goal>
            <goal>compile</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <!--创建fatjar插件-->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>2.4.3</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
            <filters>
              <filter>
```

```

        <artifact>*:*</artifact>
        <excludes>
            <exclude>META-INF/*.SF</exclude>
            <exclude>META-INF/*.DSA</exclude>
            <exclude>META-INF/*.RSA</exclude>
        </excludes>
    </filter>
</filters>
</configuration>
</execution>
</executions>
</plugin>
<!--编译插件-->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.2</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
    </configuration>
    <executions>
        <execution>
            <phase>compile</phase>
            <goals>
                <goal>compile</goal>
            </goals>
        </execution>
    </executions>
</plugin>
</plugins>
</build>

```

### 3. 客户端程序

```

package com.baizhi.flink

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

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

        //1.创建执行环境
        val environment = StreamExecutionEnvironment.getExecutionEnvironment

        //本地运行环境
        //val environment = StreamExecutionEnvironment.createLocalEnvironment(3)

        //2.创建DataStream
        val text = environment.socketTextStream("flink.baizhiedu.com", 9999)
    }
}

```

```

//3.对text数据进行常规转换
val result = text.flatMap(line=>line.split("\\s+"))
    .map(word=>(word,1))
//keyBy就是对上面的数据做分组处理，根据0号元素分组。也就是根据输入的单词分组
    .keyBy(0) //类似于spark中的reducebykey; groupbykey
    .keyBy(0)
    .sum(1);

//4.控制台打印结果
result.print();

//5.执行流计算任务
environment.execute("myDataStreamJobTask")

}

}

```

## 程序部署

### 1. 本地 执行（直接在idea《开发工具》中运行代码）

- 通过nc命令监听端口号9999

```
nc -lk 9999
```

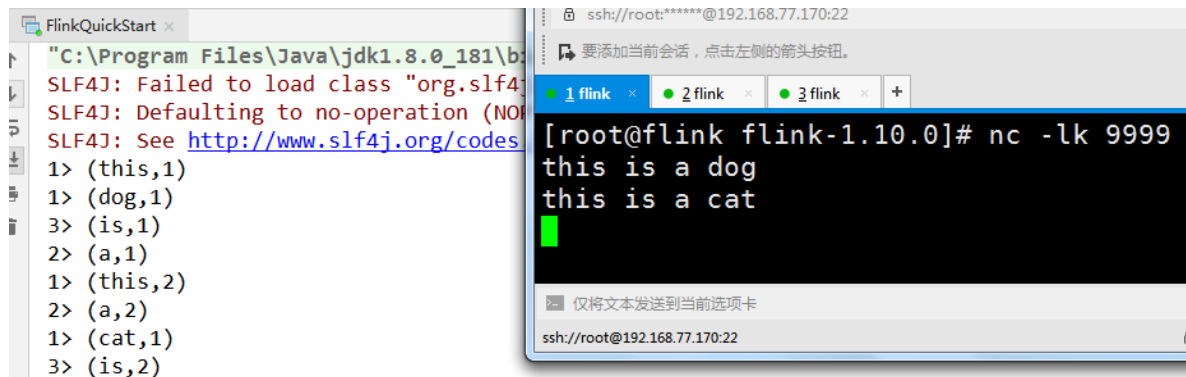
Note:必须保证nc已经安装完成

- 客户端程序修改为本地运行环境

也可以直接使用 `StreamExecutionEnvironment.getExecutionEnvironment`

```
val environment = StreamExecutionEnvironment.createLocalEnvironment(3)
```

- 运行程序：执行main方法
- 在Linux的nc界面输入内容，在开发工具的控制台查看输出结果

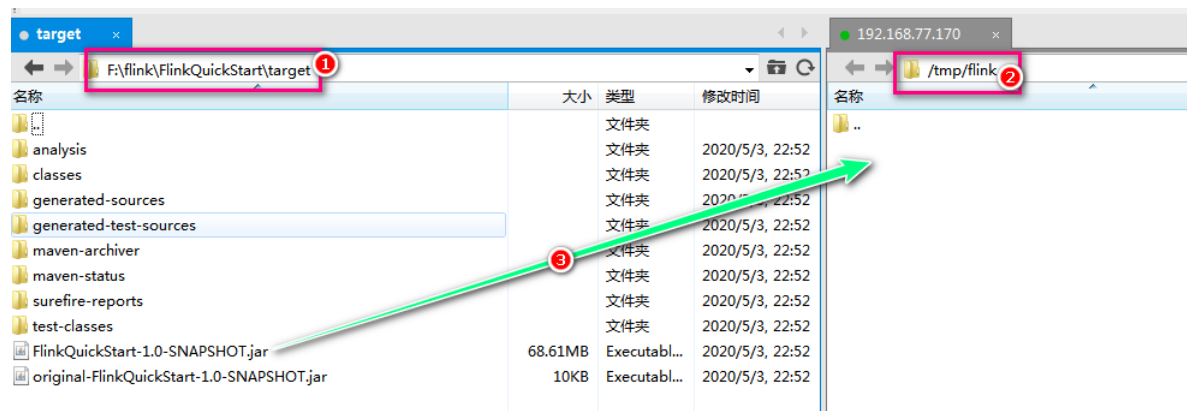


### 2. 远程脚本部署

- 客户端程序修改为自动识别运行环境

```
val environment = StreamExecutionEnvironment.getExecutionEnvironment
```

- 通过mvn package生成jar包
- 把生成的jar传输到Linux系统的/tmp/flink目录下



- 通过执行flink\_home/bin目录下的flink文件的run action, 提交job

```
[root@flink bin]# pwd
/opt/install/flink-1.10.0/bin
[root@flink bin]# ./flink run -c com.baizhi.flink.FlinkQuickStart -p 3 -d -m flink.baizhiedu.com:8081 /
tmp/flink/FlinkQuickStart-1.0-SNAPSHOT.jar
Job has been submitted with JobID 44de5b615e221c3601995ff4a2e415fb
```

#### 说明

`-c,--class <classname>`  
Class with the program entry point ("main()" method). Only needed if the JAR file does not specify the class in its manifest.  
指定启动类

`-d,--detached`  
If present, runs the job in detached mode  
后台提交

`-p,--parallelism <parallelism>`  
The parallelism with which to run the program. Optional flag to override the default value specified in the configuration.  
指定并行度

`-m,--jobmanager <arg>`  
Address of the JobManager (master) to which to connect. Use this flag to connect to a different JobManager than the one specified in the configuration.  
提交目标主机

必须保证在监听9999端口号

所有的action/option/args都不需要记忆。通过./flink --help查看帮助文档

也可以通过./flink --help查看对应action的帮助文档。其中位置可以是run/list/info/cancel...

- 查看运行中的job

```
[root@flink bin]# ./flink list --running --jobmanager flink.baizhiedu.com:8081
Waiting for response...
----- Running/Restarting Jobs -----
03.05.2020 23:38:55 : 44de5b615e221c3601995ff4a2e415fb : myDataStreamJobTask (RUNNING)
-----
```

```
[root@flink bin]# ./flink list --running --jobmanager flink.baizhiedu.com:8081
```

- 查看所有job

```
[root@flink bin]# ./flink list --all --jobmanager flink.baizhiedu.com:8081
Waiting for response...
----- Running/Restarting Jobs -----
03.05.2020 23:38:55 : 44de5b615e221c3601995ff4a2e415fb : myDataStreamJobTask (RUNNING)
-----
No scheduled jobs.
----- Terminated Jobs -----
03.05.2020 23:04:12 : a6eb79d83fdaa38376f113c8494467f9 : myDataStreamJobTask (FAILED)
03.05.2020 23:20:44 : bf78bebeed94f509e9df0e6c6f65b7bf : myDataStreamJobTask (FAILED)
03.05.2020 23:23:35 : 59bb442e385b4c77ab63386227a548e3 : myDataStreamJobTask (FAILED)
03.05.2020 23:28:56 : ac719f564d572f071d8570fad3b8ad7 : myDataStreamJobTask (CANCELED)
03.05.2020 23:33:55 : 70e5548ac2c822a652a5800061bfd532 : myDataStreamJobTask (FAILED)
-----
```

```
[root@flink bin]# ./flink list --all --jobmanager flink.baizhiedu.com:8081
```

#### ○ 取消指定job

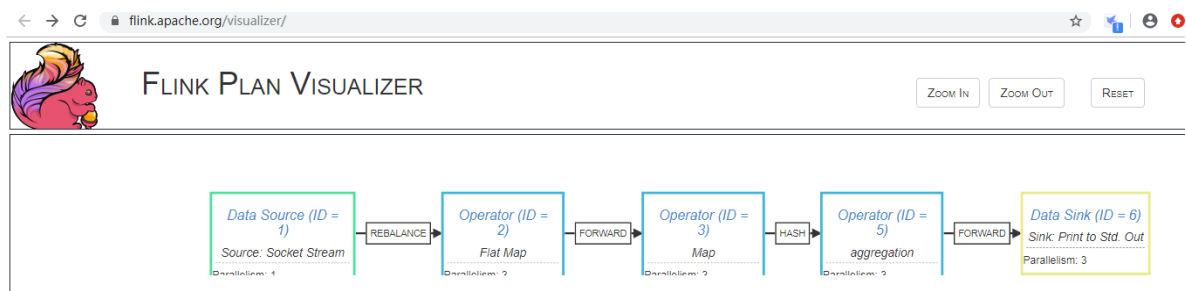
```
[root@flink bin]# ./flink cancel --jobmanager flink.baizhiedu.com:8081 44de5b615e221c3601995ff4a2e415fb
Cancelling job 44de5b615e221c3601995ff4a2e415fb.
Cancelled job 44de5b615e221c3601995ff4a2e415fb.
```

#### ○ 查看程序执行计划

```
[root@flink bin]# ./flink info -c com.baizhi.flink.FlinkQuickStart -p 3 /tmp/flink/FlinkQuickStart-1.0-SNAPSHOT.jar
----- Execution Plan -----
{"nodes":[{"id":1,"type":"Source: Socket Stream","pact":"Data Source","contents":"Source: Socket Stream","parallelism":1}, {"id":2,"type":"Flat Map","pact":"Operator","contents":"Flat Map","parallelism":3,"predecessors":[{"id":1,"ship_strategy":"REBALANCE","side":"second"}]}, {"id":3,"type":"Map","pact":"Operator","contents":"Map","parallelism":3,"predecessors":[{"id":2,"ship_strategy":"FORWARD","side":"second"}]}, {"id":5,"type":"aggregation","pact":"Operator","contents":"aggregation","parallelism":3,"predecessors":[{"id":3,"ship_strategy":"HASH","side":"second"}]}, {"id":6,"type":"Sink: Print to Std. Out","pact":"Data Sink","contents":"Sink: Print to Std. Out","parallelism":3,"predecessors":[{"id":5,"ship_strategy":"FORWARD","side":"second"}]}]}
-----
No description provided.
```

```
[root@flink bin]# ./flink info -c com.baizhi.flink.FlinkQuickStart -p 3
/tmp/flink/FlinkQuickStart-1.0-SNAPSHOT.jar
```


访问<https://flink.apache.org/visualizer/>，将上述命令执行之后的json复制过去查看程序执行计划



### 3. Web UI部署

通过访问flink的web界面，提交job完成部署



 **Apache Flink Dashboard**

Overview

Jobs

Running Jobs

Completed Jobs

Task Managers

Job Manager

Submit New Job




### Available Task Slots

4

Total Task Slots 4 | Task Managers 1

### Running Job List

Job Name

 **Apache Flink Dashboard**

Overview

Jobs

Running Jobs

Completed Jobs

Task Managers

Job Manager

Submit New Job



Version: 1.10.0 | Commit: aa4eb8f @ 07.02.2020 @ 19:18:19 CET | Message: ⓘ

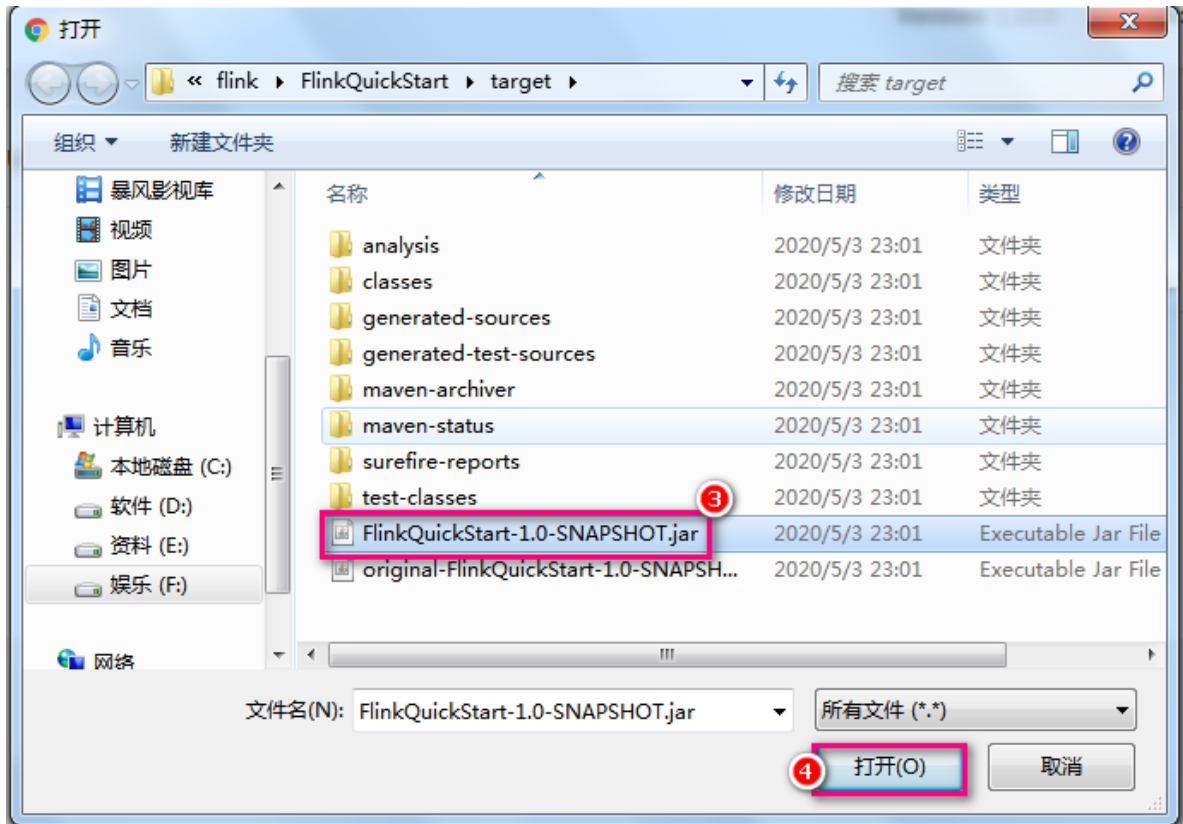
### Uploaded Jars

+ Add New

Name	Upload Time	Entry Class
------	-------------	-------------



No Data



Uploaded Jars [+ Add New](#)

鼠标点击

Name	Upload Time	Entry Class
FlinkQuickStart-1.0-SNAPSHOT.jar	2020-05-03, 23:54:05	-

Name Upload Time Entry Class

FlinkQuickStart-1.0-SNAPSHOT.jar 2020-05-03, 23:54:05 - Delete

Entry Class

Program Arguments

☐ Allow Non Restored State

myDataStreamJobTask RUNNING 3

ID: 6ad9dc3019e909c4e94810cc033ac885 | Start Time: 2020-05-03 23:56:38 | Duration: 15s [Cancel Job](#)

[Overview](#) [Exceptions](#) [TimeLine](#) [Checkpoints](#) [Configuration](#)

```
graph LR; A[Source: Socket Stream  
Parallelism: 1] -- REBALANCE --> B[Fiat Map -> Map  
Parallelism: 3]; B -- HASH --> C[aggregation -> Sink: Print to S  
td. Out  
Parallelism: 3];
```

#### 4. 跨平台部署

- 修改程序的运行环境代码，并指定并行度

```
//创建跨平台执行环境
val jar = "F:\\flink\\FlinkQuickStart\\target\\FlinkQuickStart-1.0-SNAPSHOT.jar";
val environment =
StreamExecutionEnvironment.createRemoteEnvironment("flink.baizhiedu.com",8081,jar);
//设置并行度
environment.setParallelism(3);
```

- 通过mvn package将程序打包
- 运行main方法完成部署

## Flink运行架构

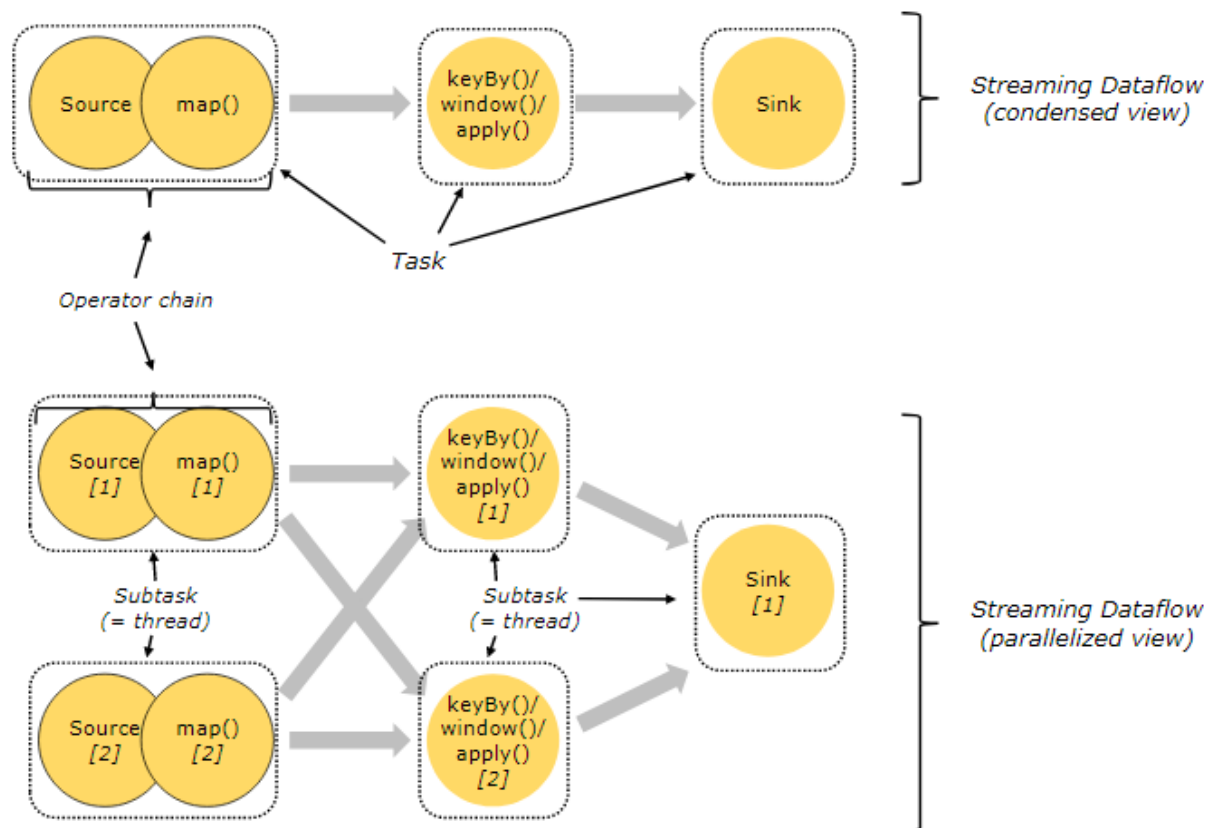
参考: <https://ci.apache.org/projects/flink/flink-docs-release-1.10/concepts/runtime.html>

### Tasks and Operator Chains

Flink是一个分布式流计算引擎，该引擎将一个计算job拆分成若干个Task(等价于Spark中的Stage)。每个Task都有自己的并行度，每个并行度都由一个线程表示，因此一个Task底层对应一系列的线程，Flink称为这些线程为该Task的subtask。

与Spark不同的地方在于Spark是通过RDD的依赖关系实现Stage的划分而Flink是通过 Operator Chain的概念实现Task的拆分。该方式将多个算子归并到一个task中，从而优化计算，减少Thread-to-Thread的通信成本。

For distributed execution, Flink chains operator subtasks together into tasks. Each task is executed by one thread. Chaining operators together into tasks is a useful optimization: it reduces the overhead of thread-to-thread handover and buffering, and increases overall throughput while decreasing latency. The chaining behavior can be configured;



- **Task** - 等价spark中的Stage，每个Task都有若个Subtask
- **Subtask** - 等价于一个线程，是Task中的一个子任务
- **Operator Chain** - 将多个算子归并到一个Task的机制，归并原则类似于SparkRDD的宽窄依赖

## Job Managers, Task Managers, Clients

Flink在运行的过程中，有两种类型的进程组成

- **Job Manager** - (也称为master) 负责协调分布式执行。负责任务调度，协调检查点，协调故障恢复等。等价于Spark中的Master+Driver的功能。通常一个集群中至少有1个Active的JobManager，如果在HA模式下其他处于StandBy状态。
- **Task Manager** - (也称为Worker) 真正负责Task执行计算节点，同时需要向JobManager汇报自身状态信息和工作负荷。通常一个集群中有若干个TaskManager，但必须至少一个。

The Flink runtime consists of two types of processes:

The JobManagers (also called masters) coordinate the distributed execution. They schedule tasks, coordinate checkpoints, coordinate recovery on failures, etc.

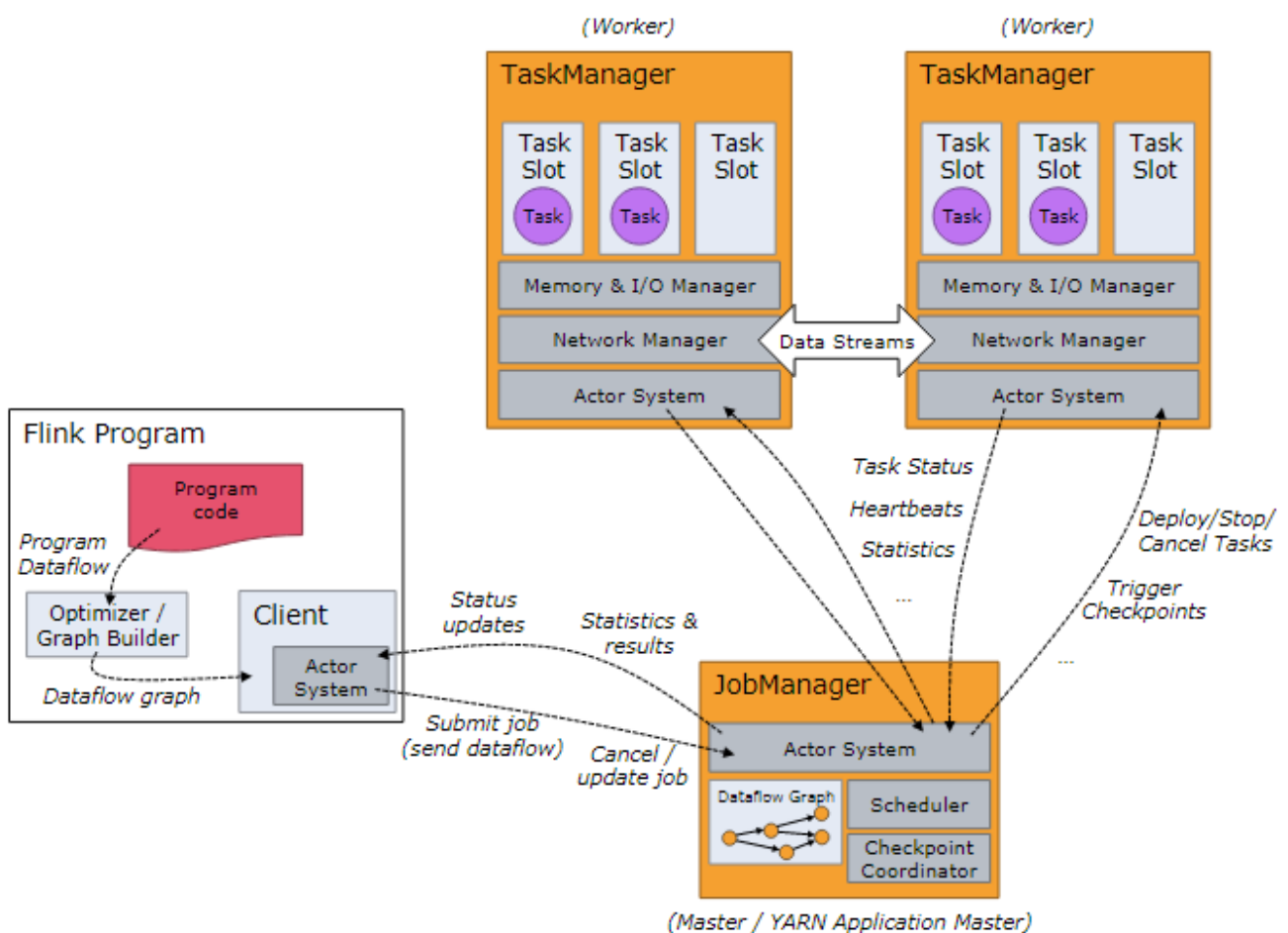
There is always at least one Job Manager. A high-availability setup will have multiple JobManagers, one of which one is always the leader, and the others are standby.

The TaskManagers (also called workers) execute the tasks (or more specifically, the subtasks) of a dataflow, and buffer and exchange the data streams.

There must always be at least one TaskManager.

- **Client** - Flink中的Client并不是集群计算的一部分，Client仅仅准备和提交dataflow给JobManager。提交完成之后，可以直接退出，也可以保持连接以接收执行进度，因此Client并不负责任务执行过程中调度。Client可以作为触发执行的Java/Scala程序的一部分运行，也可以在命令行窗口中运行。类似Spark中的Driver，但又特别不同，因为Spark Driver负责任务调度和恢复

The client is not part of the runtime and program execution, but is used to prepare and send a dataflow to the JobManager. After that, the client can disconnect, or stay connected to receive progress reports. The client runs either as part of the Java/Scala program that triggers the execution, or in the command line process `./bin/flink run ....`



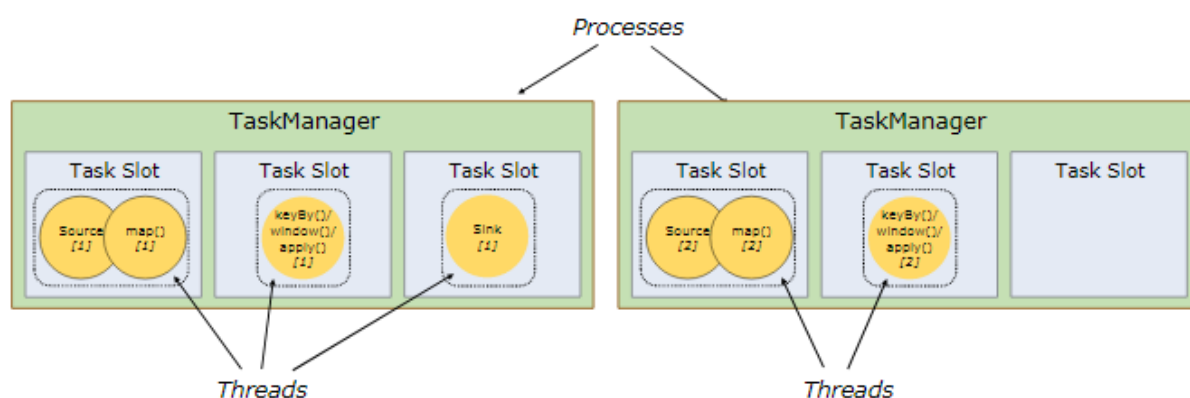
## Task Slots and Resources(难点)

每一个Worker (TaskManager) 是一个JVM进程，可以执行一个或者多个子任务 (Thread/SubTask)。为了控制Worker节点能够接收多少个Task任务，提出了所谓Task slot用于表达一个计算节点的计算能力（每个计算节点至少有一个Task slot）。

每个Task slot表示的是TaskManager计算资源的固定子集。例如，一个TaskManager拥有3个Task slot，则每个Task slot占用TaskManager所管理内存资源的1/3。每个Job启动的时候都拥有固定的Task Slot，这些被分配的Task Slot资源只能被当前job的所有Task使用，不同Job的Task之间不存在资源共享和抢占问题。

但是每个Job会被拆分成若干个Task，每个Task由若干个SubTask构成(取决于Task并行度)。默认Task slot所对应的内存资源只能在**同一个Job下的不同Task的subtask间进行共享**，也就意味着同一个Task的不同subtask不能运行在同一个Taskslot中。

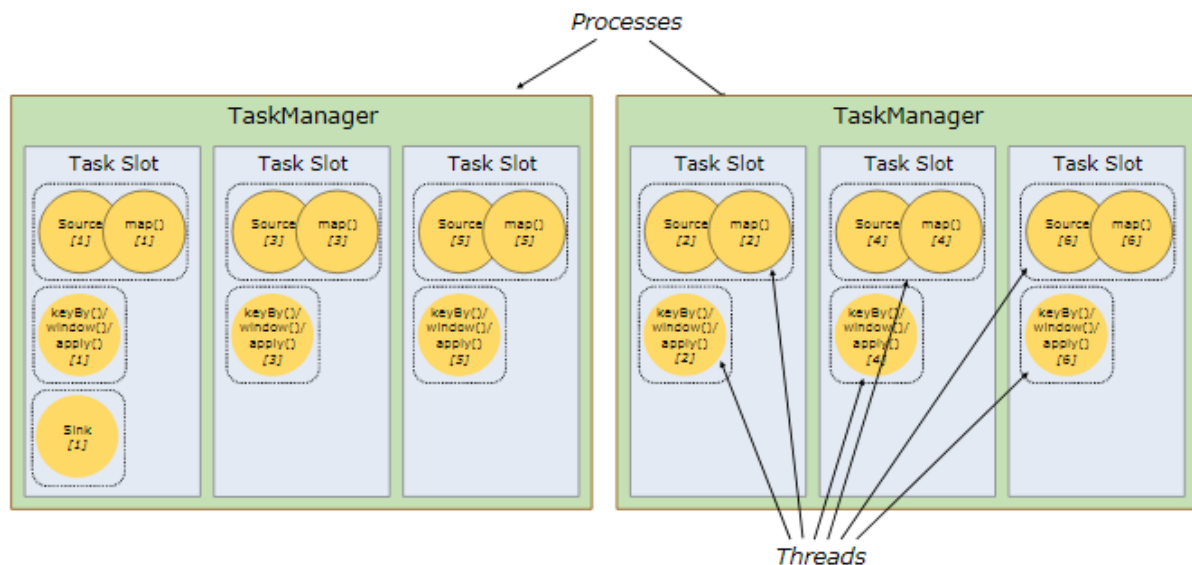
如果同一个Job下的不同Task的subtask间不能共享slot，就会造成非密集型subtask的阻塞，从而浪费内存。



Note:

- 非密集型任务：source()/map()。操作占用内存量小
- 密集型任务：keyBy()/window()/apply()。操作过程中涉及shuffle，会占用大量内存

因此，Flink的底层设计为同一Job下的不同Task的subtask间共享slot。可以将并行度调整从而充分利用资源。将上述示例的并行度从2调整为6，Flink底层会确保heavy subtasks 均衡分布于TaskManager之间的slots



Flink默认行为：同一Job下的不同Task的subtask间共享slot，就意味着一个job运行需要的Task slot个数应该等于该job中Task的并行度最大值。当然用户可以设置Task slot的共享策略

**conclusion:**

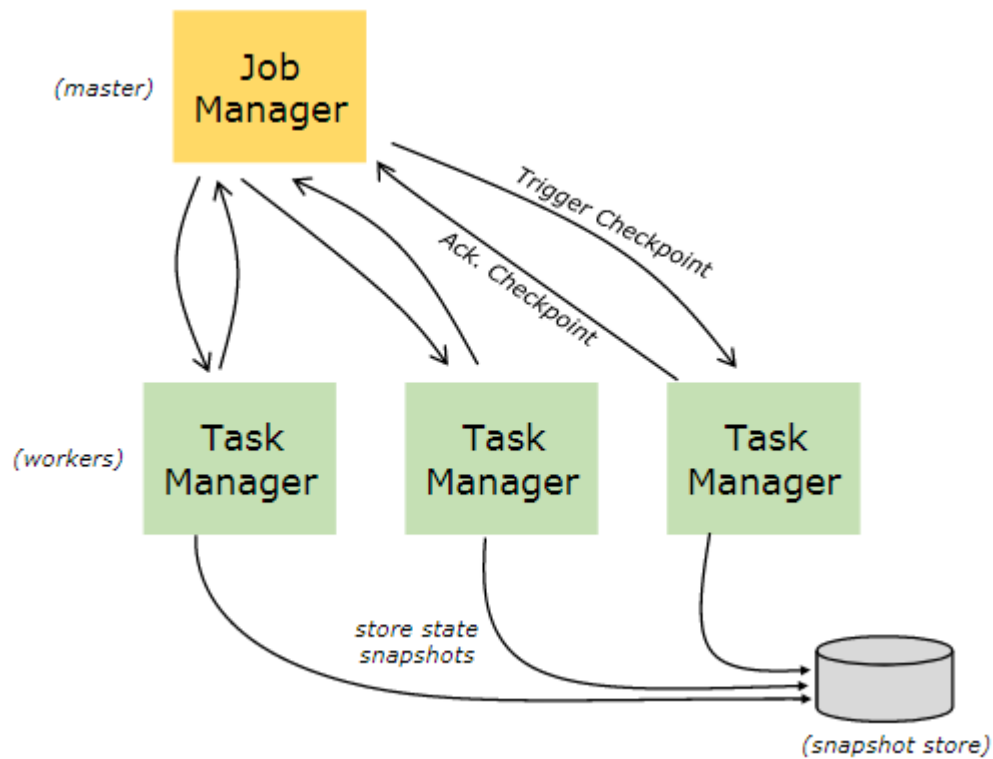
在Flink的应用中，用户只需要指定job的并行度即可，无需指定运行需要的资源数

## State Backends

Flink是一个基于状态计算的流计算引擎。存储的key/value状态索引的确切数据结构取决于所选的State Backends. 除了定义保存状态的数据结构外，State Backend还实现了获取key/value状态时间点快照 并且将该快照储存在checkpoint一部分的逻辑。

Flink中定义了三种State Backend

- The MemoryStateBackend：内存
- The FsStateBackend：文件系统，比如hdfs
- The RocksDBStateBackend：DB;



## Checkpoint/Savepoints

checkpoint是由flink定期的，自动的进行数据的持久化（把状态中的数据写入到磁盘（HDFS））。新的checkpoint执行完成之后，会把老的checkpoint丢弃掉

用Data Stream API编写的程序可以从savepoint恢复执行。Savepoint允许在不丢失任何状态的情况下更新程序和Flink集群。

Savepoint是手动触发的checkpoint，它获取程序的快照并将其写入state backend。Checkpoint依赖于常规的检查点机制：在执行过程中，程序会定期在TaskManager上快照并且生成checkpoint。为了恢复，只需要最后生成的checkpoint。旧的checkpoint可以在新的checkpoint完成后安全地丢弃。

Savepoint与上述的定期checkpoint类似，只是他们由用户触发，并且在新的checkpoint完成时不会自动过期。Savepoint可以通过命令行创建，也可以通过REST API在取消job时创建。

## 作业

1. flink安装完成
2. 四种部署方式操作一下
3. 理解运行架构