Flink 开发环境搭建和应用的配置、部署及运行

版本	*日期*
v1.0	*2019.03.17*

前言

本课程主要面向于初次接触 Flink、或者对 Flink 有了解但是没有实际操作过的同学。

本课程的目标是，希望帮助初次使用 Flink 的同学更顺利地上手使用、以及着手相关开发调试工作。

课程内容包括：

- Flink 开发环境的部署和配置
- 运行 Flink 应用（包括：单机 standalone 模式、standalone 集群模式和 Yarn 集群模式）

一、Flink 开发环境部署和配置

Flink 是一个以 Java 及 Scala 作为开发语言的开源大数据项目，代码开源在 github 上，并使用 maven 来编译和构建项目。对于大部分开发或使用 Flink 的同学来说，Java、Maven 和 Git 这三个工具是必不可少的，因此我们会首先介绍一下这三个工具的安装和配置。另外，一个强大的 IDE 有助于我们更快的阅读代码、开发新功能以及修复 bug，因此这里也会简单介绍 IDE 的相关配置。

根据我们之前的调查，大部分开发者使用 Mac OS 作为本地开发环境，所以在本章节我们主要在 Mac 上演示配置。对于使用 Windows 系统的同学，推荐使用 Win10 系统的 Linux 子系统来编译和运行，这样既可以有 windows 上开发和日常使用的便捷，又可以以 Linux 的方式运行程序，达到接近于在 Linux 服务器端运行的效果。另外，使用 Ubuntu 或者 CentOS 这些热门 Linux 操作系统作为本地开发环境也是完全可行的。

第一章节中的案例，如无特殊说明，默认指的是在 mac 系统上的安装和配置。

1. Java 的安装和配置

在各个操作系统上安装和配置 Java 的教程有很多，这里有三个要点需要注意：

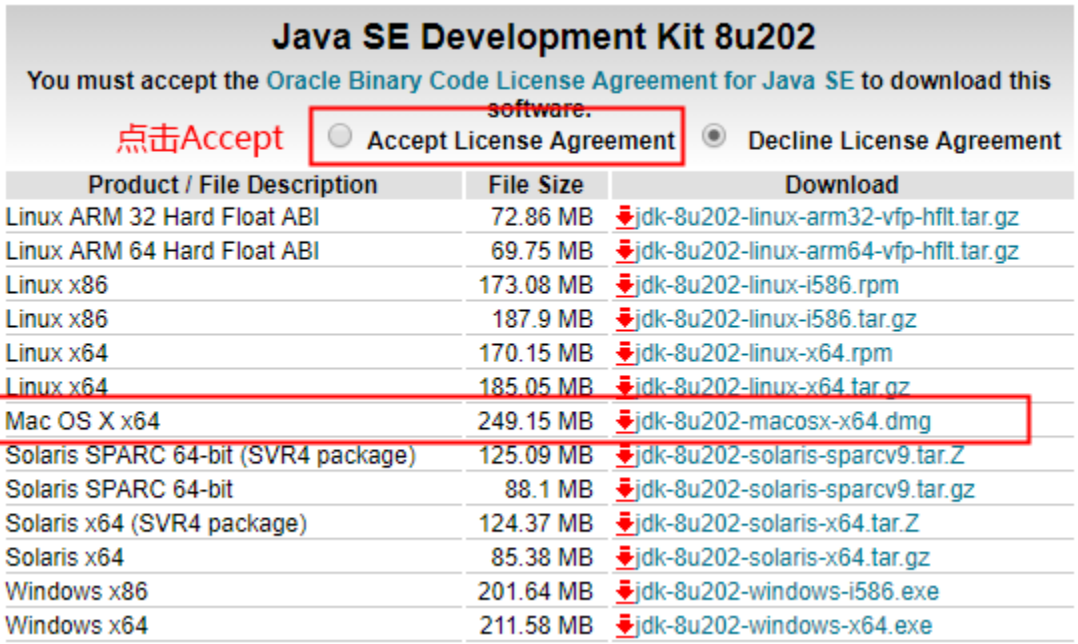
- Flink 编译和运行要求 Java 版本至少是 Java 8，且最好选用 Java 8u51 及以上版本
- 如果要能够编译 Flink 代码，需要安装 JDK
- 安装好 Java 后，还需要配置 JAVA_HOME 和 PATH

这三个要点在 mac 系统、Linux 系统及 Windows 系统上都是适用的。

Mac OS 上安装 JKD8 方法如下：

在下面这个下载链接中下载并安装 Mac OS 对应的安装包

https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html



Java SE Development Kit 8u202

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

点击Accept ○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.86 MB	jdk-8u202-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.75 MB	jdk-8u202-linux-arm64-vfp-hflt.tar.gz
Linux x86	173.08 MB	jdk-8u202-linux-i586.rpm
Linux x86	187.9 MB	jdk-8u202-linux-i586.tar.gz
Linux x64	170.15 MB	jdk-8u202-linux-x64.rpm
Linux x64	185.05 MB	jdk-8u202-linux-x64.tar.gz
Mac OS X x64	249.15 MB	jdk-8u202-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	125.09 MB	jdk-8u202-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	88.1 MB	jdk-8u202-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	124.37 MB	jdk-8u202-solaris-x64.tar.Z
Solaris x64	85.38 MB	jdk-8u202-solaris-x64.tar.gz
Windows x86	201.64 MB	jdk-8u202-windows-i586.exe
Windows x64	211.58 MB	jdk-8u202-windows-x64.exe

安装完成后查看 java8 的安装目录

```
/usr/libexec/java_home -V
```

根据 java_home 命令的结果在~/.bashrc 文件中配置 JAVA_HOME 和 PATH 这两个环境变量

PS：如果使用 zsh 作为默认 shell 的话，则在~/.zshrc 中配置

```
export JAVA_HOME=${your_java_home}
export PATH=$JAVA_HOME/bin:$PATH
```

使环境变量生效，如果使用 zsh，则使用~/.zshrc

```
source ~/.bashrc
```

检查 java 版本

```
java -version
```

补充：

（1）Mac 上也可以通过 HomeBrew 下载 java8

不过需要注意的是，这种方式下载的 java 版本是 openjdk 版的。命令如下：

查看 java8 的信息

```
brew cask info java8
```

安装 java8

```
brew cask install java8
```

（2）关于 Linux 和 Windows 上 JDK 8 的安装和配置方式，网上也有许多介绍的文章，故不详述。

2. Maven 的安装和配置

编译 Flink 要求必须使用 Maven 3，推荐使用 Maven 3.2.5。Maven 3.3.x 能够编译成功，但是在 shade 一些 dependencies 的过程中有些问题，故不推荐使用。

具体步骤如下：

直接下载 Maven 3.2.5 的 binary 包即可

```
wget https://archive.apache.org/dist/maven/maven-3/3.2.5/binaries/apache-maven-3.2.5-bin.tar.gz
```

下载完成后解压到指定目录中

```
tar zxvf apache-maven-3.2.5-bin.tar.gz -C ${your_application_install_dir}
```

环境变量配置：

将 maven 的安装目录配置为 MAVEN_HOME，并把 maven 的 bin 目录加到 PATH 中

```
export MAVEN_HOME=${your_application_install_dir}
export PATH=$JAVA_HOME/bin:$MAVEN_HOME/bin:$PATH
```

使环境变量生效，如果使用 zsh，则使用~/.zshrc

```
source ~/.bashrc
```

查看 maven 的版本

```
mvn -v
```

PS：在 Mac 上还可以使用 brew 下载指定版本的 maven

```
brew install maven@3.2
```

3. Git 的安装和配置

Git 的安装可以参考这篇文章：https://git-scm.com/book/en/v1/Getting-Started-Installing-Git

对于 Mac 用户可以直接用 HomeBrew 安装，命令如下：

```
brew install git
```

（可选）另外可以配置 git alias 来简化命令，创建~/.gitconfig 文件，加入如下内容：

```
[alias]
    co = checkout
    st = status
    ci = commit
    br = branch
```

4. 下载 flink 代码

当我们完成上述安装配置后，我们就可以从 github 上下载 Flink 代码了。github 上 flink 的代码仓库是 https://github.com/apache/flink

（可选）对于国内的用户，下载 github 上的代码可能比较慢，可以在/etc/hosts 中增加如下配置，可以显著提升 github 的下载速度：

```
151.101.72.133 assets-cdn.github.com
151.101.73.194 github.global.ssl.fastly.net
192.30.253.113 github.com
11.238.159.92 git.node5.mirror.et2sqa
```

如果使用 Windows 系统，则是配置在“C:\Windows\System32\drivers\etc\hosts”文件中。

如果使用 Win10 Linux 子系统，建议也配置在“C:\Windows\System32\drivers\etc\hosts”文件中，然后重启 Linux 子系统，因为 Linux 子系统中的/etc/hosts 文件是根据 Window 系统中的“C:\Windows\System32\drivers\etc\hosts”这个文件生成的。

使用 Win10 Linux 子系统还可以通过删除 Linux 子系统的/etc/hosts 文件中的这一行来阻止 Linux 子系统启动的时候覆盖修改过的 hosts 文件：

```
# This file was automatically generated by WSL. To prevent automatic generation of this file, remove this line.
```

下载 Flink 代码到本地

```
git clone https://github.com/apache/flink.git
```

（可选）代码下载完后，默认是在 master 分支，考虑到代码质量，一般会选择合适的发布分支使用，比如 release-1.6 或者 release-1.7，在本次演示中，我们选用阿里巴巴最新开源的 blink 做演示。PS：blink 分支的代码会逐步合并到 master 上。

```
git checkout release-1.6

git checkout release-1.7

git checkout blink
```

5. 编译 flink 代码

Flink 代码使用 maven 构建项目，编译代码的时候 maven 默认会根据当前用户下的“~/.m2/settings.xml”文件中的配置信息下载 Flink 的依赖包，也可以在 mvn 命令中增加“--settings=${your_maven_settings_file}”来指定 maven settings 文件的位置。

如果你之前已有合适的 maven settings 的配置，可以直接使用已有的配置即可。

一个可用的 maven settings.xml 配置文件见链接：
https://drive.google.com/file/d/1cUq9BaHSxEeIKBPKYE8YyFQ9LD6EW8yB/view?usp=sharing
打开链接后，直接复制文件内容，然后粘贴到“~/.m2/settings.xml”文件（或其他 settings.xml 文件）中。

如果需要指定 maven 的 local repository 的路径，可以在 settings.xml 文件中配置“localRepository”这个参数。默认情况下会下载到“~/.m2/repository/”（即当前用户 home 目录下的“.m2/repository”目录）。

重要的配置片段如下所示。

```
<mirror>
<id>nexus-aliyun</id>
<mirrorOf>*,!jeecg,!jeecg-snapshots,!mapr-releases</mirrorOf>
<name>Nexus aliyun</name>
<url>http://maven.aliyun.com/nexus/content/groups/public</url>
</mirror>

<mirror>
<id>mapr-public</id>
<mirrorOf>mapr-releases</mirrorOf>
<name>mapr-releases</name>
<url>https://maven.aliyun.com/repository/mapr-public</url>
</mirror>
```

简要说明一下，第一个 mirror 使用的是 aliyun 提供的 maven 镜像仓库，能够为国内用户加速 maven repository 的访问，你也可以配置成国内其他的 maven 镜像仓库或者自己搭建的仓库。最重要的是下面片段中红色标注的内容。由于 flink 中的 flink-filesystems/flink-mapr-fs 模块依赖

mapr-releases repository 提供的 jar 包，然而由于国内访问 mapr-releases repository 比较慢，而且所依赖的 maprfs-5.2.1-mapr.jar 这个 jar 包有 48MB，flink 依赖中最大的一个 jar 包，故初次编译 flink 时，往往会由于下载 mapr 相关依赖超时导致编译失败。因此，aliyun 专门有一个镜像仓库代理 mapr-releases repository，以期望能让用户更容易地下载 mapr 相关的 jar 包。

可以通过这个链接查看 aliyun 提供的镜像仓库的 meta 信息：https://maven.aliyun.com/mvn/view

在我们配置好之前的几个工具后，编译 flink 就非常简单了，执行如下命令即可：

```
# 删除已有的 build，编译 flink binary
# 接着把 flink binary 安装在 maven 的 local repository（默认是~/.m2/repository）中
mvn clean install -DskipTests

# 另一种编译命令，相对于上面这个命令，主要的确保是：
# 不编译 tests、QA plugins 和 JavaDocs，因此编译要更快一些
mvn clean install -DskipTests -Dfast
```

另外，在一些情况下，我们可能并不想把编译后的 flink binary 安装在 maven 的 local repository 下，我们可以使用下面的命令：

```
# 删除已有的 build，编译 flink binary
mvn clean package -DskipTests

# 另一种编译命令，相对于上面这个命令，主要的确保是：
# 不编译 tests、QA plugins 和 JavaDocs，因此编译要更快一些
mvn clean package -DskipTests -Dfast
```

如果你需要使用指定 hadoop 的版本，可以通过指定“-Dhadoop.version”来设置，编译命令如下：

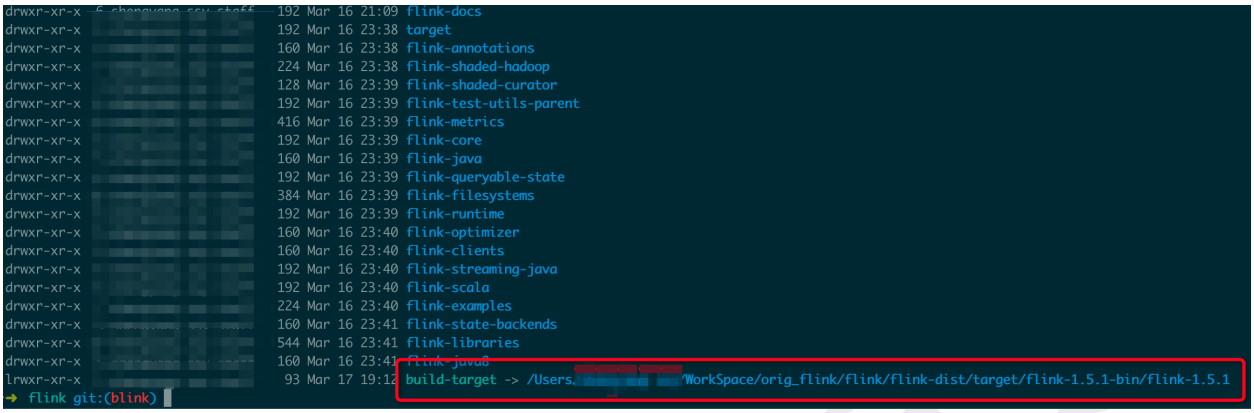
```
mvn clean install -DskipTests -Dhadoop.version=2.6.1
# 或者
mvn clean package -DskipTests -Dhadoop.version=2.6.1
```

当成功编译完成后，上述几种编译方式最终都能在当前 flink 的 code path 下编译出完整的 flink binary，可以在 flink-dist/target/目录中看到：

其中有三个文件可以留意一下（在之后的章节中，我们会介绍 flink binary 的用法）：

- flink binary 目录，本例中是 flink-dist/target/flink-1.5.1-bin/flink-1.5.1
- flink binary 目录的压缩包，本例中是 flink-dist/target/flink-1.5.1.tar.gz
- 包含 flink 核心功能的 jar 包，本例中是 flink-dist/target/flink-dist_2.11-1.5.1.jar

另外，为了方便用户使用，编译时会为 flink binary 目录在 flink 当前 code path 下建一个名为“build-target”的软链接。



在 flink 的 code path 的根目录，执行：

```
cd build-target

# 查看 flink 的版本
./bin/flink -v
```

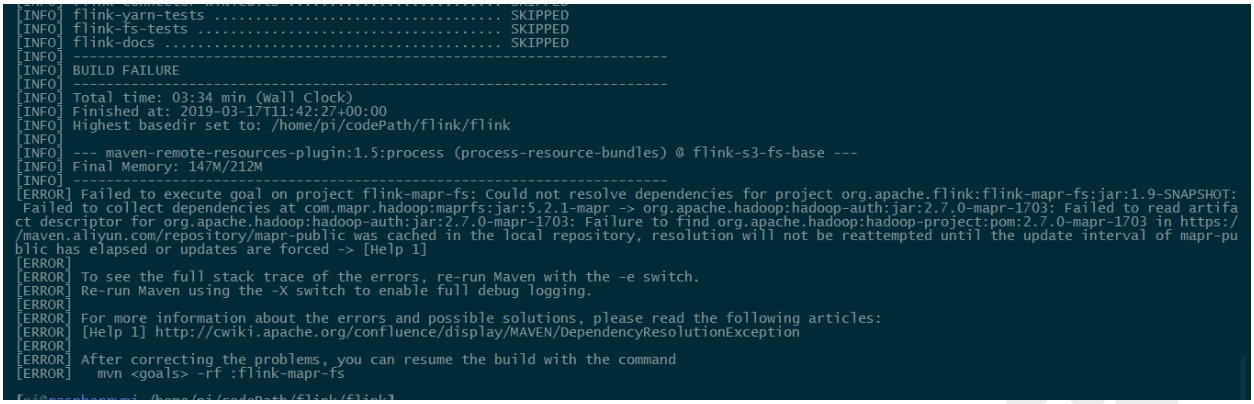
在编译中可能遇到的问题

- 问题 1：编译失败“BUILD FAILURE”，失败信息中有 mapr 相关信息

这种错误一般都和 mapr 相关的依赖包的下载失败有关，在实际测试时，即使配置了之前说的 aliyun 代理的 mapr-releases 镜像，还是可能出现下载失败的情况，问题可能还是和 mapr 的 jar 包比较大，容易下载失败有关。

遇到这些问题时，重试即可。在重试之前，要先根据失败信息删除 maven local repository 中对应的目录，否则需要等待 maven 下载的超时时间才能再次出发下载依赖到本地。

比如下面这个编译失败：



失败信息显示 com.mapr.hadoop:maprfs:jar:5.2.1-mapr 和它依赖的 org.apache.hadoop:hadoop-auth:jar:2.7.0-mapr-1703 有问题，就直接把这两个包对应在 maven local repository 中的目录删掉，然后重新编译即可。

```
rm -rf ~/.m2/repository/com/mapr/hadoop/maprfs/5.2.1-mapr
rm -rf ~/.m2/repository/org/apache/hadoop/hadoop-auth/2.7.0-mapr-1703
```

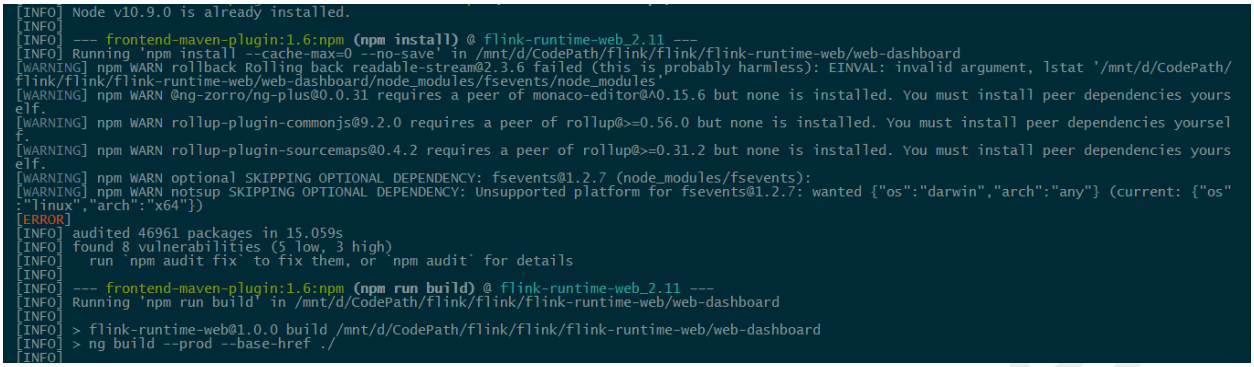
我还遇到过上面这种情况，直接删除 maprfs 的 jar 包后重试即可。

```
rm -rf ~/.m2/repository/com/mapr/hadoop/maprfs/5.2.1-mapr
```

这些问题等到编译成功后，相关的 mapr 的 jar 包就保存在本地的 local repository 目录下了，之后的编译就没问题了。

- 问题 2：发现在 Win10 的 Linux 子系统中编译 flink 比较耗时

我在 Win10 的 Linux 子系统中编译 flink 发现，编译 flink-runtime-web 过程中执行“ng build --prod --base-href ./”命令非常慢，最后虽然编译过了，但差不多花了一个小时的时间。



单独执行“ng build --prod --base-href ./”这个命令时，会长时间停留在“92% chunk asset optimization”处。不确定是否和 Linux 子系统有关，也可能和我的 Win10 机器的内存比较少有关（我的 Win10 机器编译之前的剩余内存只有 3GB 左右，执行这个 ng 命令比较耗内存，整机内存差不多用完了）。这个问题目前没有结论，有兴趣和条件的同学，也可以试一试。

6. 开发环境准备

一个好的 IDE 不仅能有效的提高开发者的开发效率，而且对于不做代码开发但是希望通过代码学习 Flink 的人来说，也非常有助于其对代码的理解。

推荐使用 IntelliJ IDEA IDE 作为 Flink 的 IDE 工具。官方的说法是，不建议使用 Eclipse IDE，主要原因是 Eclipse 的 Scala IDE 和 Flink 用 scala 的不兼容。

（1）下载安装 Intellij IDEA

Intellij IDEA IDE 的下载地址：https://www.jetbrains.com/idea/，下载最新版本安装即可。

（2）安装 Scala plugin

Flink 项目使用了 Java 和 Scala 开发，Intellij 自带 Java 的支持，在导入 Flink 代码前，还需要确保安装 Intellij 的 Scala plugin。安装方法如下：

1. IntelliJ IDEA -> Preferences -> Plugins，点击“Install Jetbrains plugin…”
2. 搜索“scala”，点击“install”
3. 重启 Intellij

（3）检查 Intellij 的 Maven 配置

1. IntelliJ IDEA -> Preferences -> Build, Execution, Deployment -> Build Tools -> Maven
2. 检查“Maven home directory”是否符合预期，如果不是，则选择正确的 maven 路径，然后 apply
3. 检查“User settings file”是否符合预期，默认是“${your_home_dir}/.m2/settings.xml”，如果之前没有特殊配置，则无需更改
4. 检查“Local directory”是否符合预期，默认是“${your_home_dir}/.m2/repository”，如果之前没有特殊配置，则无需更改

（4）导入 Flink 代码

1. IntelliJ IDEA -> File -> New -> Project from existing sources…，选择 Flink 代码的根路径
2. 在“Import project from external model”中选择“Maven”，然后一路点击“next”直到结束

3. IntelliJ IDEA -> File -> Project Structure… -> Project Settings -> Project，检查 Project SDK 是否符合预期（因为在之前的步骤中我们已经配置了 JAVA_HOME，所以一般是符合预期的），如果不是就点击“New”，然后选择之前步骤中安装的 JDK home 目录

PS：代码导入完后，Intellij 会自动 sync 代码并创建 index 用于代码查找。如果之前代码没有编译过，则需要做一次代码全编译，然后 Intellij 经过一次 sync 后，就能这样 Intellij 就能识别所有的代码。

（5）添加 Java 的 Checkstyle

在 Intellij 中添加 Checkstyle 是很重要的，因为 Flink 在编译时会强制代码风格的检查，如果代码风格不符合规范，可能会直接编译失败。对于需要在开源代码基础上做二次开发的同学，或者有志于向社区贡献代码的同学来说，及早添加 checkstyle 并注意代码规范，能帮你节省不必要的修改代码格式的时间。

Intellij 内置对 Checkstyle 的支持，可以检查一下 Checkstyle-IDEA plugin 是否安装（IntelliJ IDEA -> Preferences -> Plugins，搜索“Checkstyle-IDEA”）。

配置 Java Checkstyle：

1. IntelliJ IDEA -> Preferences -> Other Settings -> Checkstyle
2. 设置 “Scan Scope”为“Only Java sources (including tests)”
3. 在“Checkstyle Version”下拉框中选择“8.9”
4. 在“Configuration File”中点击“+”新增一个 flink 的配置：
    a. “Description”填“Flink”
    b. “Use a local Checkstyle file”选择本代码下的 tools/maven/checkstyle.xml 文件
    c. 勾选“Store relative to project location”，然后点击“Next”
    d. 配置“checkstyle.suppressions.file” 的值为"suppressions.xml"，然后点击“Next”和“Finish”
    e. 勾选上“Flink”作为唯一生效的 checkstyle 配置，点击“Apply”和“OK”

5. IntelliJ IDEA -> Preferences -> Editor -> Code Style -> Java，点击⚙齿轮按钮，选择“Import Scheme” -> “Checkstyle Configuration”，选择 checkstyle.xml 文件。这样配置后，Intellij 在自动 import 的时候会按照规则，把 import 代码添加到正确的位置。

需要说明的是，Flink 中的一些模块并不能完全 checkstyle 通过，包括 flink-core、flink-optimizer 和 flink-runtime。但无论如何，还是应当保证你新增或修改的代码遵守 checkstyle 的规范。

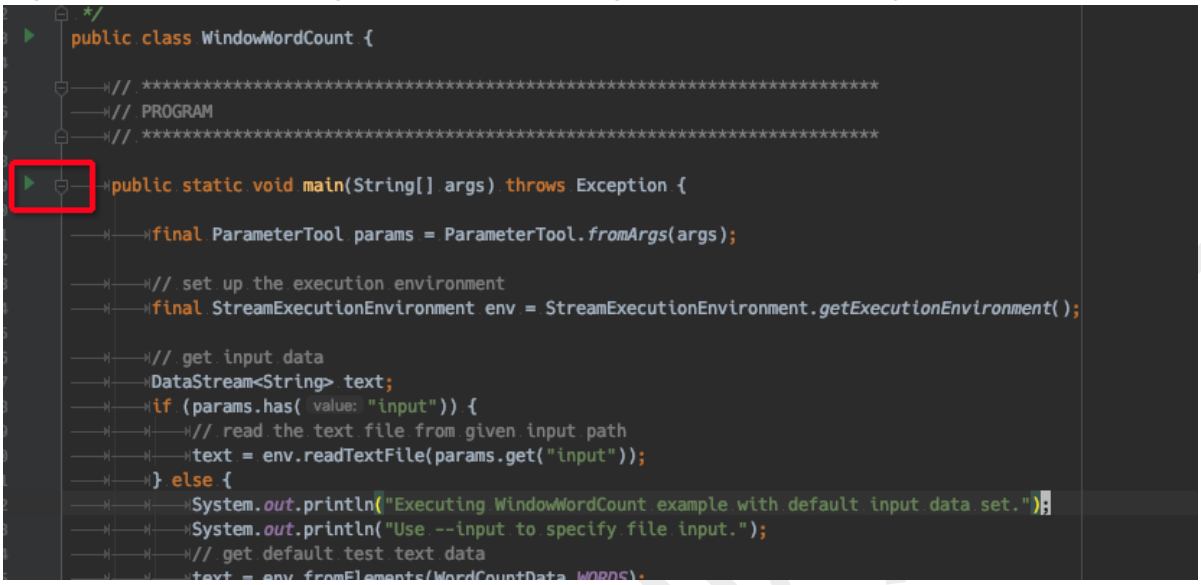
（6）添加 Scala 的 Checkstyle

1. 将“tools/maven/scalastyle-config.xml”文件拷贝到 flink 代码根目录的“.idea”子目录中
2. IntelliJ IDEA -> Preferences -> Editor -> Inspections，搜索“Scala style inspections”，勾选这一项

（7）小试牛刀：在 Intellij 中运行 example

flink 代码编译完成后，直接选择一个 example 即可运行，如：

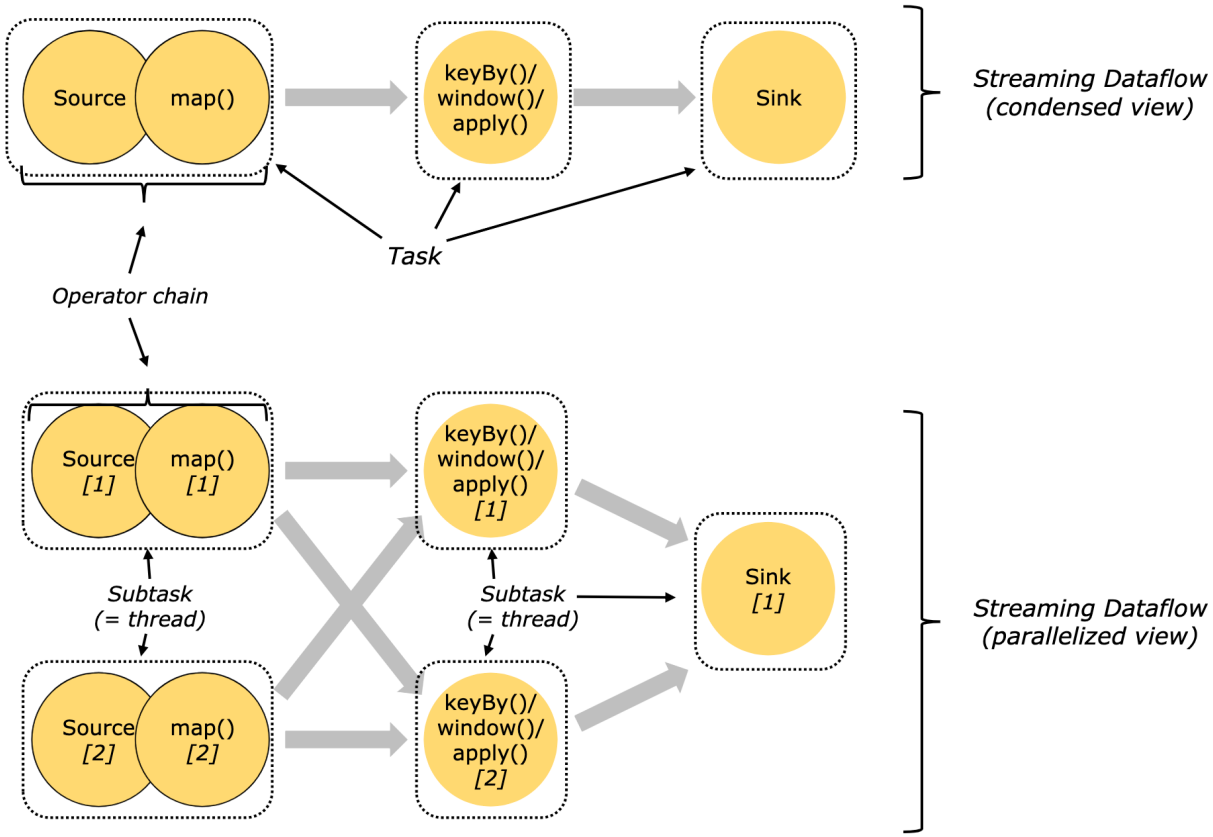
org.apache.flink.streaming.examples.windowing.WindowWordCount.java



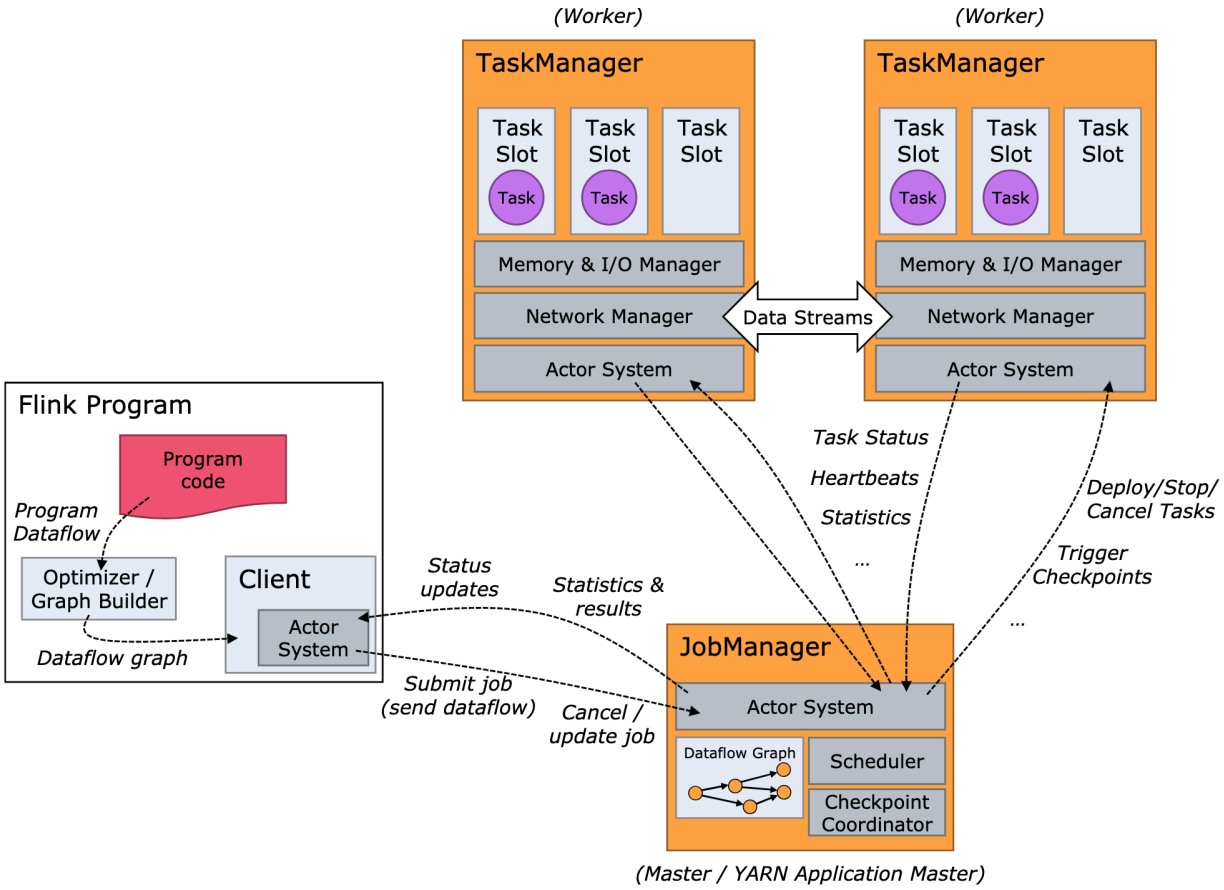
二、运行 Flink 应用

1. 基本概念

运行 Flink 应用其实非常简单，但是在运行 Flink 应用之前，还是有必要了解 Flink 运行时的各个组件，因为这涉及到 Flink 应用的配置问题。

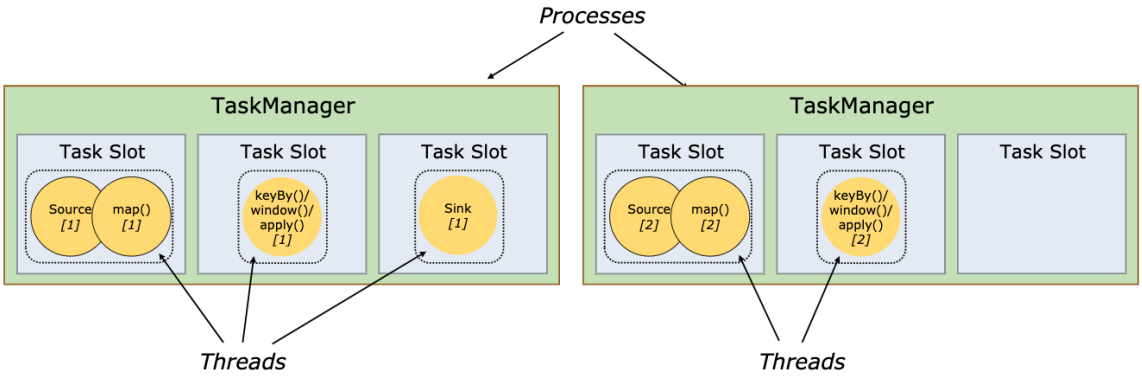


通过这张图我们可以看到，在一个 DAG 图中，不能被 chain 在一起的 operator 会被分隔到不同的 Task 中，也就是说，Task 是 Flink 中资源调度的最小单位。



Flink 运行时包括两类进程：

- JobManager（又称为 JobMaster）：协调 Task 的分布式执行，包括调度 Task、协调创 checkpoint 以及当 job failover 时协调各个 Task 从 checkpoint 恢复等。
- TaskManager（又称为 Worker）：执行 dataflow 中的 Tasks，包括内存 buffer 的分配、Data Stream 的传递等。

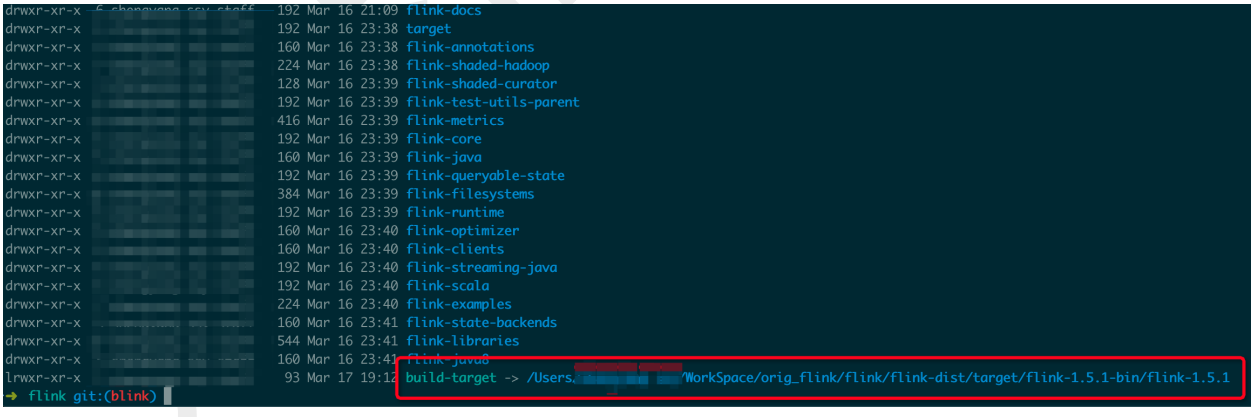


2. 运行环境准备

（1）准备 Flink binary

接下来我们需要准备 Flink binary，主要有两种方法：

- 直接从 Flink 官网上下载 Flink binary 的压缩包: https://flink.apache.org/downloads.html
- 从 Flink 源码编译而来（参考第一章节的第 5 点“编译 flink 代码”）
    - 下图所示“flink-dist/target/flink-1.5.1.tar.gz”是 flink binary 的压缩包
    - Flink 的 code path 下的 build-target 目录就是一个可用的 flink binary 目录



将下载或编译出来的 Flink binary 的压缩包解压后，和编译出来的 build-target 目录是等价的。

```
tar zxvf xxx.tar.gz -C ${dir_for_decompression}
```

（2）安装 Java，并配置 JAVA_HOME 环境变量

第一章中主要介绍的是 Flink 开发环境的部署，而如果只是需要运行 Flink binary，则只需要安装和配置好 Java 即可。

详情请参考第一章第 1 小节。

3. 单机 standalone 的方式运行 flink

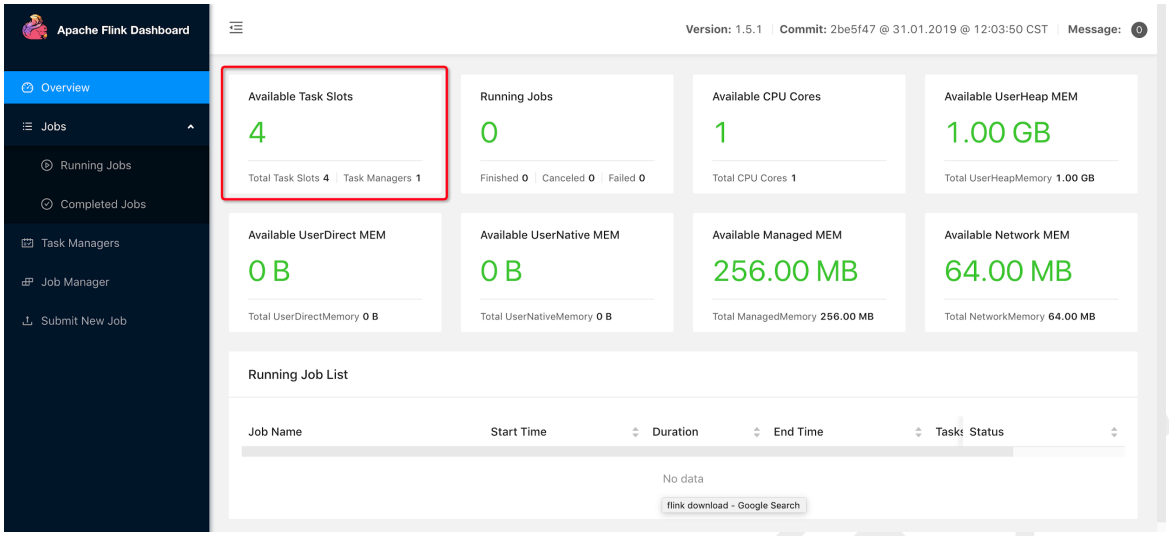
（1）基本的启动流程

最简单的运行 flink 应用的方法就是以单机 standalone 的方式运行。进入 Flink binary 的目录下（一般情况下不需要修改配置文件），执行：

```
./bin/start-cluster.sh
```

输出结果如下，表示启动正常：

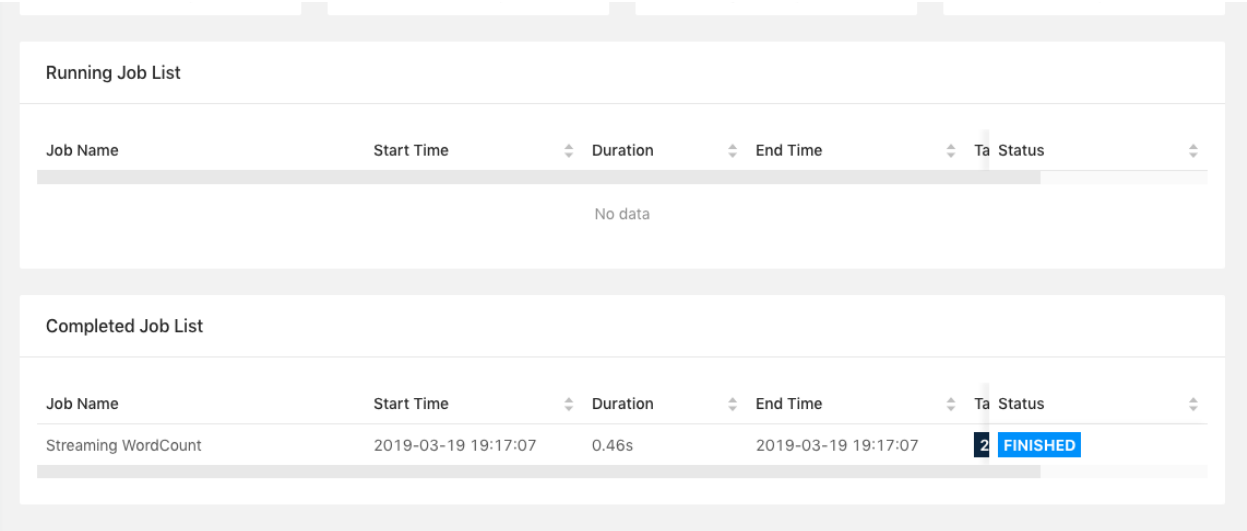
启动成功后，打开 http://127.0.0.1:8081/ 就能看到 Flink 的 web 界面：


Apache Flink Dashboard
Overview
Jobs
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit New Job
Version: 1.5.1 | Commit: 2be5f47 @ 31.01.2019 @ 12:03:50 CST | Message: 0
Available Task Slots
4
Total Task Slots 4 | Task Managers 1
Running Jobs
0
Finished 0 | Canceled 0 | Failed 0
Available CPU Cores
1
Total CPU Cores 1
Available UserHeap MEM
1.00 GB
Total UserHeapMemory 1.00 GB
Available UserDirect MEM
0 B
Total UserDirectMemory 0 B
Available UserNative MEM
0 B
Total UserNativeMemory 0 B
Available Managed MEM
256.00 MB
Total ManagedMemory 256.00 MB
Available Network MEM
64.00 MB
Total NetworkMemory 64.00 MB
Running Job List
Job Name
Start Time
Duration
End Time
Tasks
Status
No data
flink download - Google Search

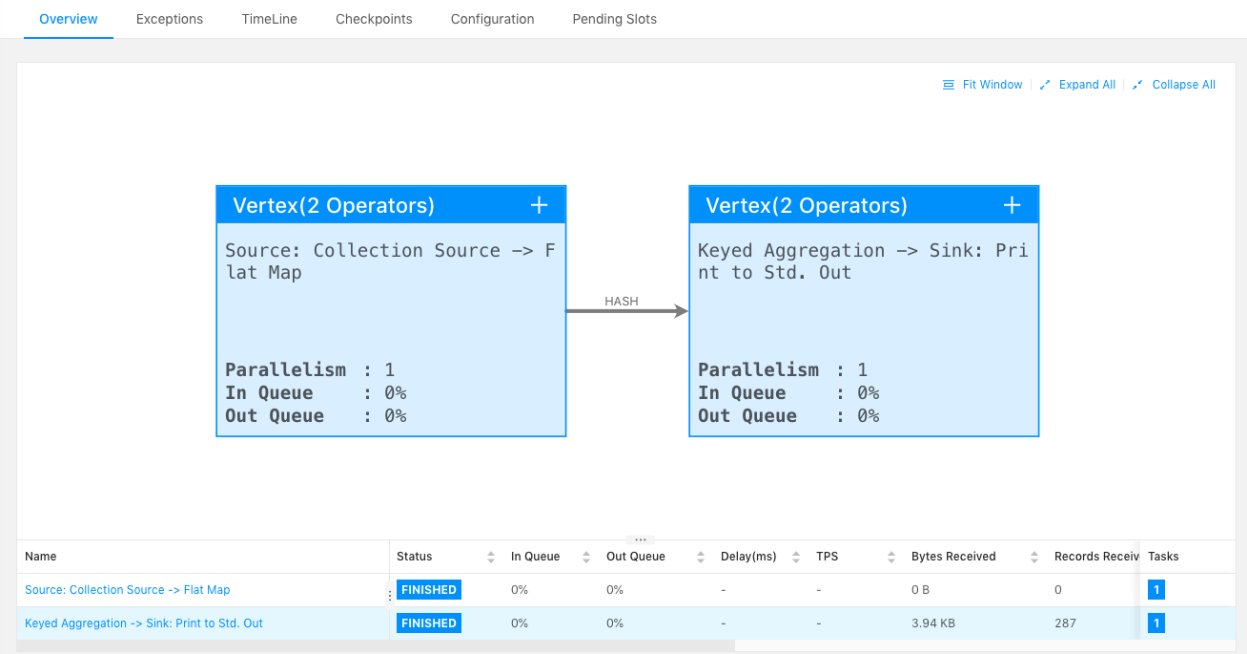
提交一个 Word Count 的任务，命令如下：

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

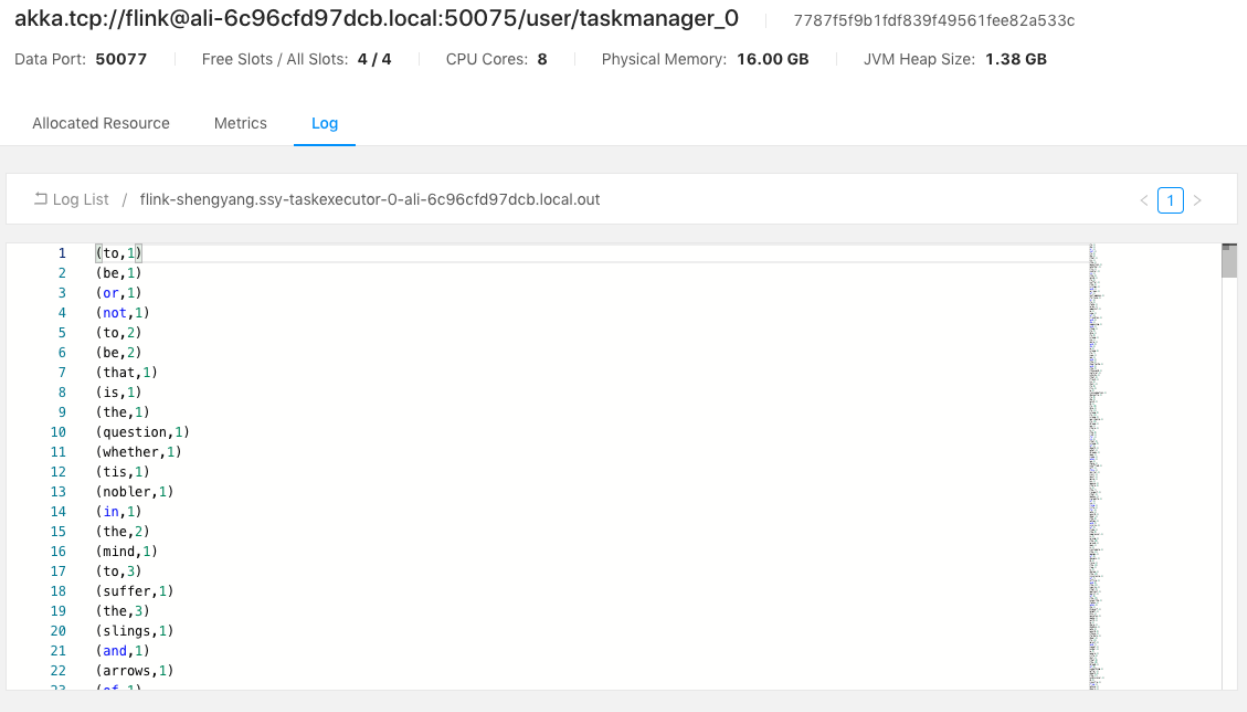
可以在 web 界面上看到 job 完成了



点击 job 信息，可以看到 Word Count 的执行图：



点进最下面的 vertex，可以通过 stdout 信息看到 Word Count 的结果：



结合 Flink 代码：

flink-examples -> flink-examples-streaming -> wordcount -> class WordCount

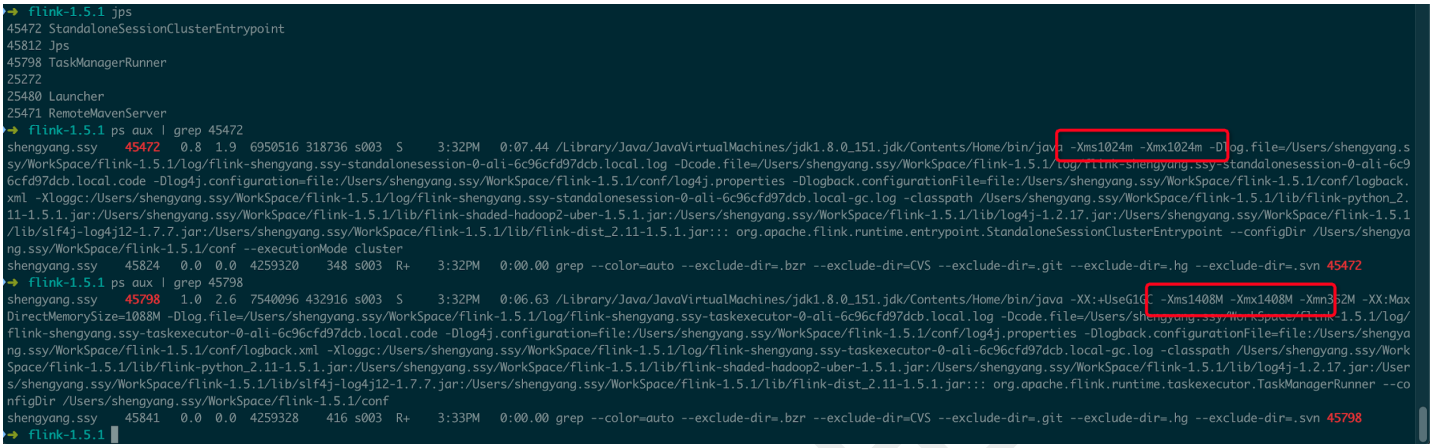
可以看到，不传任何参数时，input 的输入是代码中的一个字符串。

接着我们尝试通过“--input”参数指定我们自己的本地文件作为输入，然后执行：

```
./bin/flink run examples/streaming/WordCount.jar --input ${your_source_file}
```

（2）常用配置介绍

我们在本机上执行 jps 命令，可以看到 Flink 相关的进程主要有两个，一个是 JobManager 进程，另一个是 TaskManager 进程。我们可以进一步用 ps 命令看看进程的启动参数：



接着我们结合 Flink binary 目录下的 conf 子目录中的 flink-conf.yaml 文件，常用的配置有下面几个：

```
# The heap size for the JobManager JVM
jobmanager.heap.mb: 1024

# The heap size for the TaskManager JVM
taskmanager.heap.mb: 1024

# The number of task slots that each TaskManager offers. Each slot runs one parallel pipeline.
taskmanager.numberOfTaskSlots: 4

# the managed memory size for each task manager.
taskmanager.managed.memory.size: 256
```

我们可以先用下面这个命令停掉 standalone 集群：

```
./bin/stop-cluster.sh
```

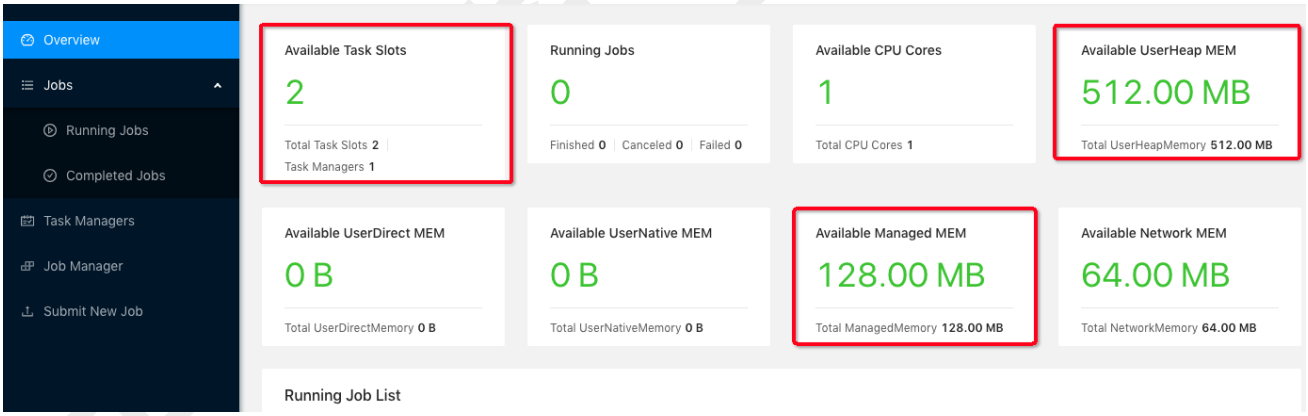
然后修改 flink-conf.yaml 中的这几个配置如下：

```
# The heap size for the JobManager JVM
jobmanager.heap.mb: 1024

# The heap size for the TaskManager JVM
taskmanager.heap.mb: 512

# The number of task slots that each TaskManager offers. Each slot runs one parallel pipeline.
taskmanager.numberOfTaskSlots: 2

# the managed memory size for each task manager.
taskmanager.managed.memory.size: 128
```

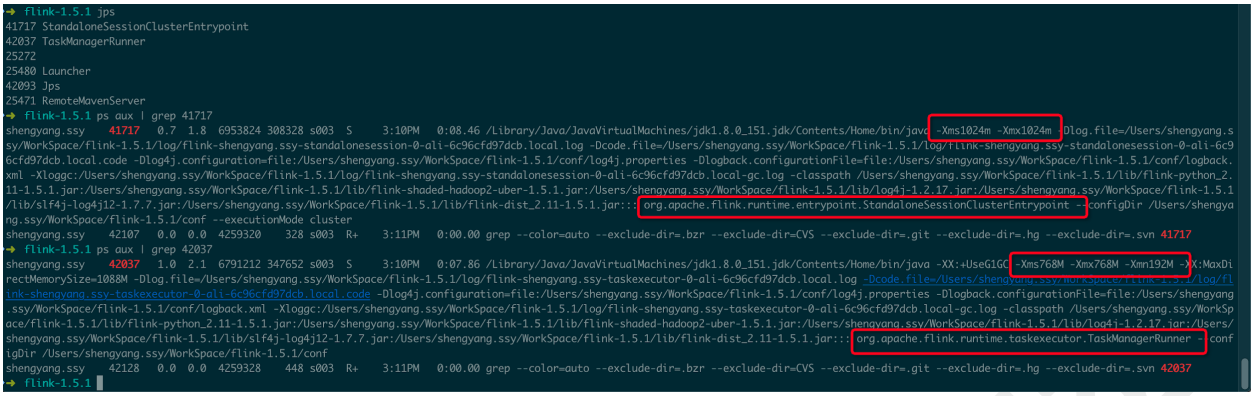
然后重启 standalone 集群：

```
./bin/start-cluster.sh
```

启动成功后，打开 http://127.0.0.1:8081/ 就能看到 Flink 的 web 界面：



我们再看一下 Flink 进程的启动参数：



在 blink 开源分支上，TaskManager 的内存计算上相对于现在的社区版本要更精细化，TaskManager 进程的堆内存限制（-Xmx）一般的计算方法是：

> TotalHeapMemory = taskmanager.heap.mb + taskmanager.managed.memory.size + taskmanager.process.heap.memory.mb（默认值为 128MB）

相对地，最新的 flink 社区版本 release-1.7 中 JobManager 和 TaskManager 的默认内存配置改为：

```
# The heap size for the JobManager JVM
jobmanager.heap.size: 1024m

# The heap size for the TaskManager JVM
taskmanager.heap.size: 1024m
```

flink 社区 release-1.7 版本中的“taskmanager.heap.size”配置实际上指的不是 Java heap 的内存限制，而是 TaskManager 进程总的内存限制。我们可以同样用上述方法查看 release-1.7 版本的 Flink binary 启动的 standalone 集群的 TaskManager 的进程-Xmx 配置，会发现实际进程上的-Xmx 要小于配置的“taskmanager.heap.size”的值，原因在于从中扣除了 network buffer 用的内存，因为 network buffer 用的内存一定是 direct memory，所以不应该算在堆内存限制中。

（3）日志的查看和配置

JobManager 和 TaskManager 的启动日志可以在 Flink binary 目录下的 log 子目录中找到：

```
➜ flink-1.5.1 ls -lrt log
total 32
-rw-r--r-- 1 shengyang.ssy staff     0 Mar 19 11:36 flink-shengyang.ssy-standalonesession-0-ali-6c96cfd97dcb.local.out
-rw-r--r-- 1 shengyang.ssy staff     0 Mar 19 11:36 flink-shengyang.ssy-taskexecutor-0-ali-6c96cfd97dcb.local.out
-rw-r--r-- 1 shengyang.ssy staff   966 Mar 19 11:36 flink-shengyang.ssy-taskexecutor-0-ali-6c96cfd97dcb.local-gc.log
-rw-r--r-- 1 shengyang.ssy staff 12005 Mar 19 11:36 flink-shengyang.ssy-taskexecutor-0-ali-6c96cfd97dcb.local.log
-rw-r--r-- 1 shengyang.ssy staff 12287 Mar 19 11:38 flink-shengyang.ssy-standalonesession-0-ali-6c96cfd97dcb.local.log
-rw-r--r-- 1 shengyang.ssy staff   780 Mar 19 11:41 flink-shengyang.ssy-standalonesession-0-ali-6c96cfd97dcb.local-gc.log
➜ flink-1.5.1
```

log 目录中以“flink-${user}-standalonesession-${id}-${hostname}”为前缀的文件对应的即是 JobManager 的输出，其中有三个文件：

- flink-${user}-standalonesession-${id}-${hostname}.log：代码中的日志输出
- flink-${user}-standalonesession-${id}-${hostname}.out：进程执行时的 stdout 输出
- flink-${user}-standalonesession-${id}-${hostname}-gc.log：JVM 的 GC 的日志

log 目录中以“flink-${user}-taskexecutor-${id}-${hostname}”为前缀的文件对应的是 TaskManager 的输出，也包括三个文件，和 JobManager 的输出一致。

日志的配置文件在 Flink binary 目录的 conf 子目录下：

```
➜ flink-1.5.1 ls conf -lrt
total 64
-rwxr-xr-x 1 shengyang.ssy staff  1434 Mar 18 17:43 zoo.cfg
-rwxr-xr-x 1 shengyang.ssy staff    10 Mar 18 17:43 slaves
-rwxr-xr-x 1 shengyang.ssy staff    15 Mar 18 17:43 masters
-rwxr-xr-x 1 shengyang.ssy staff  2331 Mar 18 17:43 logback.xml
-rwxr-xr-x 1 shengyang.ssy staff  1550 Mar 18 17:43 logback-yarn.xml
-rwxr-xr-x 1 shengyang.ssy staff  2294 Mar 18 17:43 logback-console.xml
-rwxr-xr-x 1 shengyang.ssy staff  1939 Mar 18 17:43 log4j.properties
-rwxr-xr-x 1 shengyang.ssy staff  1709 Mar 18 17:43 log4j-yarn-session.properties
-rwxr-xr-x 1 shengyang.ssy staff  1505 Mar 18 17:43 log4j-kubernetes.properties
-rwxr-xr-x 1 shengyang.ssy staff  1884 Mar 18 17:43 log4j-console.properties
-rwxr-xr-x 1 shengyang.ssy staff  2179 Mar 18 17:43 log4j-cli.properties
-rwxr-xr-x 1 shengyang.ssy staff  4227 Mar 18 17:43 sql-client-defaults.yaml
-rwxr-xr-x 1 shengyang.ssy staff 10011 Mar 19 15:32 flink-conf.yaml
```

其中：

- log4j-cli.properties：用 Flink 命令行时用的 log 配置，比如执行“flink run”命令

- log4j-yarn-session.properties：是用 yarn-session.sh 启动时命令行执行时用的 log 配置
- log4j.properties：无论是 standalone 还是 yarn 模式，JobManager 和 TaskManager 上用的 log 配置都是 log4j.properties

这三个“log4j.*properties”文件分别有三个“logback.*xml”文件与之对应，如果想使用 logback 的同学，之需要把与之对应的“log4j.*properties”文件删掉即可，对应关系如下：

- log4j-cli.properties -> logback-console.xml
- log4j-yarn-session.properties -> logback-yarn.xml
- log4j.properties -> logback.xml

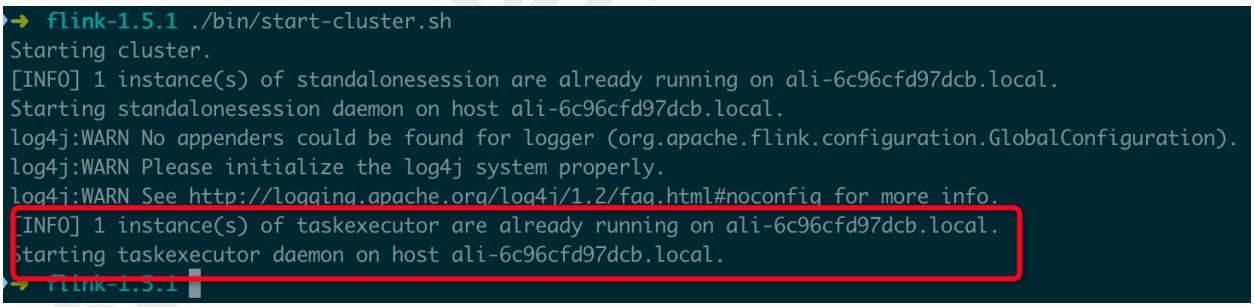
需要注意的是，“flink-${user}-standalonesession-${id}-${hostname}”和“flink-${user}-taskexecutor-${id}-${hostname}”都带有“${id}”，“${id}”表示本进程在本机上该角色（JobManager 或 TaskManager）的所有进程中的启动顺序，默认从 0 开始。

（4）进一步探索

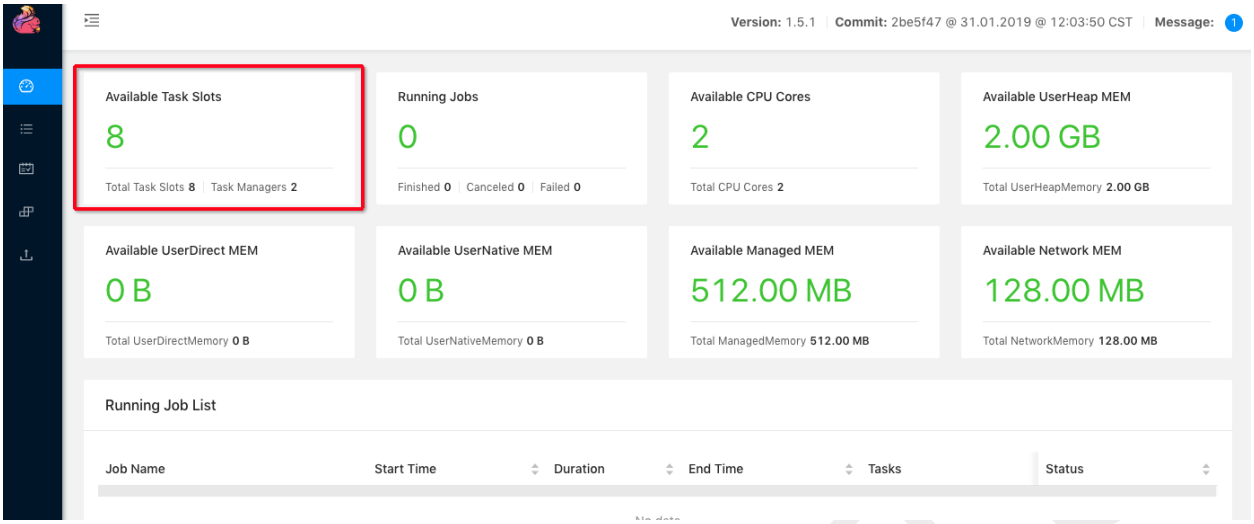
接下来我们可以在上面的基础上再次调用这个命令（重复调用 start-cluster.sh 命令）：

```
./bin/start-cluster.sh
```

命令的输出结果如下所示：



再打开 http://127.0.0.1:8081/ 就能看到 Flink 的 web 界面：



可以看到 TaskManager 的数量增加了一个，各个维度的资源也相应的翻倍了。

再看看 log 目录下的日志，会发现出现了“*-standalonesession-1-*”和“*-taskexecutor-1-*”这两种日志。我们打开“*-standalonesession-1-*.log”这个文件，翻到最下面，可以看到：

```
2019-03-19 20:59:01,234 INFO  org.apache.flink.runtime.security.modules.HadoopModule        - Hadoop user set to shengyang.ssy (auth:SIMPLE)
2019-03-19 20:59:01,266 INFO  org.apache.flink.runtime.entrypoint.ClusterEntrypoint         - Initializing cluster services.
2019-03-19 20:59:01,288 INFO  org.apache.flink.util.NetUtils                                - Unable to allocate on port 6123, due to error: Address already in use (Bind failed)
2019-03-19 20:59:01,289 ERROR org.apache.flink.runtime.entrypoint.ClusterEntrypoint         - Cluster initialization failed.
java.net.BindException: Unable to allocate further port in port range: 6123
    at org.apache.flink.runtime.clusterframework.BootstrapTools.startActorSystem(BootstrapTools.java:120)
    at org.apache.flink.runtime.entrypoint.ClusterEntrypoint.createRpcService(ClusterEntrypoint.java:396)
    at org.apache.flink.runtime.entrypoint.ClusterEntrypoint.initializeServices(ClusterEntrypoint.java:265)
    at org.apache.flink.runtime.entrypoint.ClusterEntrypoint.runCluster(ClusterEntrypoint.java:226)
    at org.apache.flink.runtime.entrypoint.ClusterEntrypoint.lambda$startCluster$0(ClusterEntrypoint.java:190)
    at java.security.AccessController.doPrivileged(Native Method)
    at javax.security.auth.Subject.doAs(Subject.java:422)
    at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1692)
    at org.apache.flink.runtime.security.HadoopSecurityContext.runSecured(HadoopSecurityContext.java:41)
    at org.apache.flink.runtime.entrypoint.ClusterEntrypoint.startCluster(ClusterEntrypoint.java:189)
    at org.apache.flink.runtime.entrypoint.StandaloneSessionClusterEntrypoint.main(StandaloneSessionClusterEntrypoint.java:104)
2019-03-19 20:59:01,291 INFO  org.apache.flink.runtime.entrypoint.ClusterEntrypoint         - Shut down and terminate StandaloneSessionClusterEntrypoint with return code 1 and application status FAILED
2019-03-19 20:59:01,291 INFO  org.apache.flink.runtime.entrypoint.ClusterEntrypoint         - Stopping StandaloneSessionClusterEntrypoint.
                                                                                                                                                54,1          Bot
```

再用 jps 命令看看：

```
➜ flink-1.5.1 jps
99872 Jps
52178 RemoteMavenServer
98869 TaskManagerRunner
25272
25480 Launcher
99592 TaskManagerRunner
98570 StandaloneSessionClusterEntrypoint
➜ flink-1.5.1
```

我们就能看到，重复执行 start-cluster 命令后，JobManager 启动失败，原因是端口冲突。而 TaskManager 则能够正常启动。

在默认配置下，单机 standalone 模式其实就是先执行“./bin/jobmanager.sh start”，后执行“./bin/taskmanager.sh start”。因此我们还可以直接通过“./bin/taskmanager.sh start”这个命令直接启动一个新的 TaskManager。

./bin/taskmanager.sh start|start-foreground|stop|stop-all

我们会发现每次启动一个新的 TM 或者 JM，其日志文件的 id 就会加一，日志的 pattern 为：“flink-${user}-standalonesession-${id}-${hostname}”，这个${id}是如何生成的呢？

```
➜  flink-1.5.1 ls /tmp/*.pid -lrt
-rw-r--r-- 1 shengyang.ssy wheel 12 Mar 19 20:59 /tmp/flink-shengyang.ssy-standalonesession.pid
-rw-r--r-- 1 shengyang.ssy wheel 16 Mar 19 21:08 /tmp/flink-shengyang.ssy-taskexecutor.pid
➜  flink-1.5.1
```

这是通过在/tmp 目录下每个角色（TM 或者 JM）都有一个 pid 文件，追加记录着每个启动过的进程 pid。需要注意的是，因为 pid 文件在/tmp 这个公共的目录下，所以即使是你在用不同 Flink binary 执行 start-cluster 都会发现这个情况，特别是在之前忘了 stop-cluster 的情况下。

pid 文件中的内容如下：

```
➜ flink-1.5.1 cat /tmp/flink-shengyang.ssy-standalonesession.pid
98570
99291
➜ flink-1.5.1 cat /tmp/flink-shengyang.ssy-taskexecutor.pid
98869
99592
569
```

接下来，我们看停 standalone 集群后会发生什么：

```
./bin/stop-cluster.sh
```

```
➜ flink-1.5.1 cat /tmp/flink-shengyang.ssy-standalonesession.pid
98570
➜ flink-1.5.1 cat /tmp/flink-shengyang.ssy-taskexecutor.pid
98869
99592
➜ flink-1.5.1
```

可以看到，每个 pid 文件中最末尾的 pid 被取出来并被 kill 了。

4. 多机部署 Flink standalone 集群

部署前要注意的要点：

- 每台机器上配置好 java 以及 JAVA_HOME 环境变量
- 最好挑选一台机器，和其他机器 ssh 打通
- **每台机器上部署的 Flink binary 的目录要保证是同一个目录**
- 如果需要用 hdfs，需要配置 HADOOP_CONF_DIR 环境变量配置上

JobManager 机器：z05f06378.sqa.zth.tbsite.net

TaskManager 机器：z05f06378.sqa.zth.tbsite.net、z05c19426.sqa.zth.tbsite.net 和 z05f10219.sqa.zth.tbsite.net

修改 Flink binary 目录的 conf 子目录中的 masters 和 slaves 两个文件：

```
$cat conf/masters
z05f06378.sqa.zth.tbsite.net:8081

$cat conf/slaves
z05f06378.sqa.zth.tbsite.net
z05c19426.sqa.zth.tbsite.net
z05f10219.sqa.zth.tbsite.net
```

修改 conf/flink-conf.yaml 配置：

```
jobmanager.rpc.address: z05f06378.sqa.zth.tbsite.net
```

然后把修改后的这三个文件同步到其他机器的相同 conf 目录下

```
conf/masters
conf/slaves
conf/flink-conf.yaml
```

然后启动 flink 集群：

```
./bin/start-cluster.sh
```

提交 WordCount 作业

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

上传 WordCount 的 input 文件：

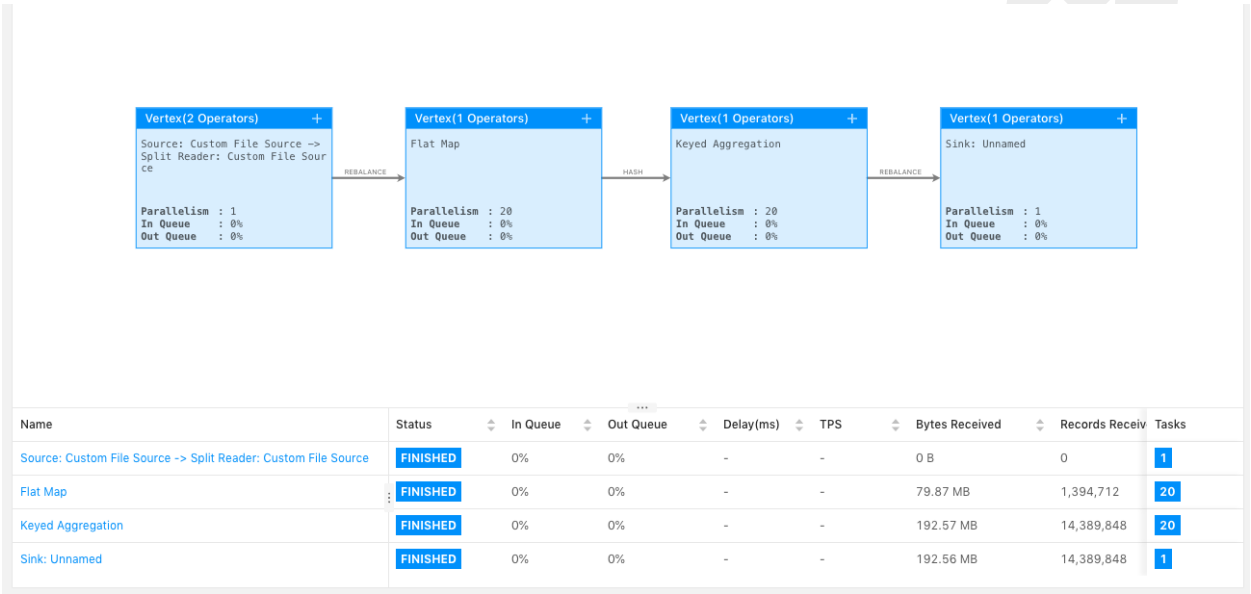
```
hdfs dfs -copyFromLocal story /test_dir/input_dir/story
```

提交读写 hdfs 的 WordCount 作业：

```
./bin/flink run examples/streaming/WordCount.jar --input hdfs:///test_dir/input_dir/story --output hdfs:///test_dir/output_dir/output
```

增加 WordCount 作业的并发度（注意输出文件重名会提交失败）：

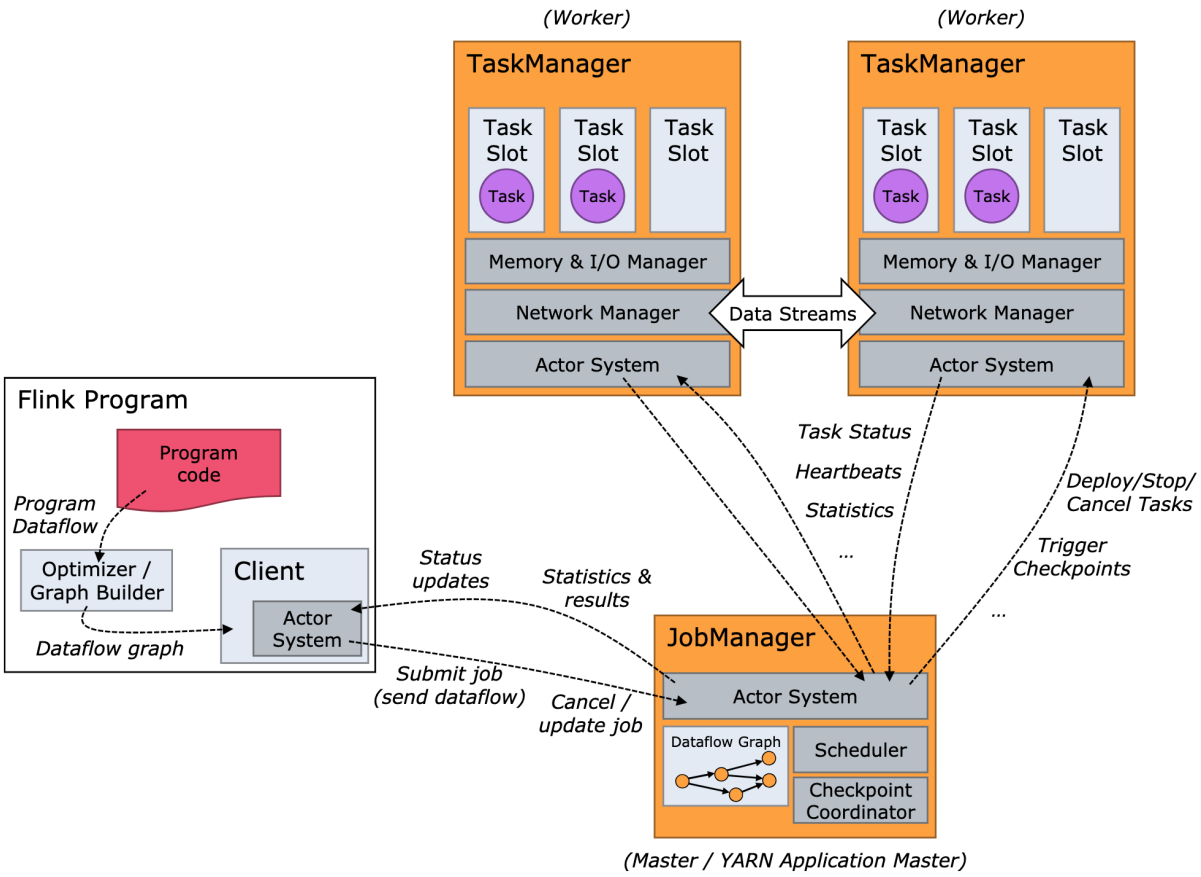
```
./bin/flink run examples/streaming/WordCount.jar --input hdfs:///test_dir/input_dir/story --output hdfs:///test_dir/output_dir/output --parallelism 20
```



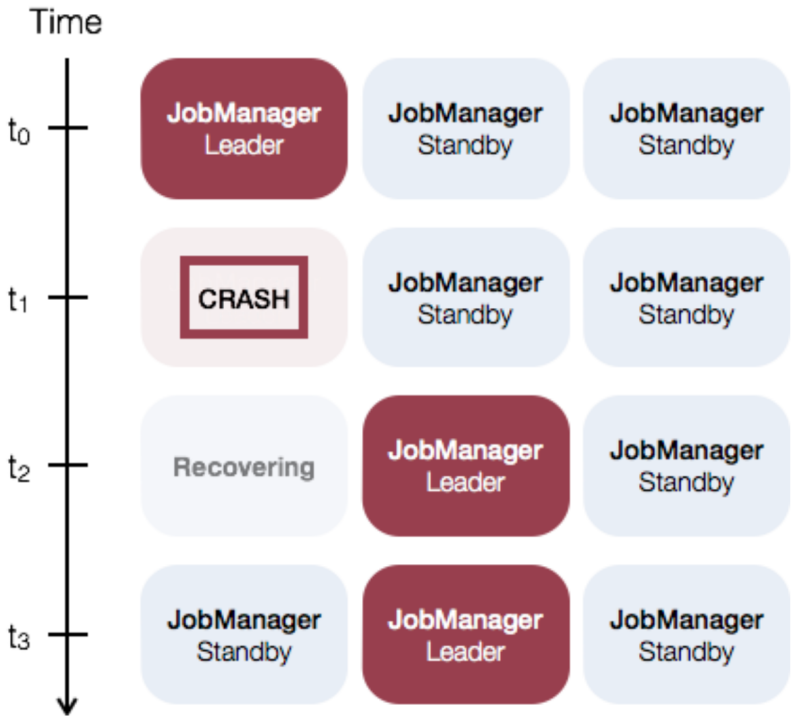
Name	Status	In Queue	Out Queue	Delay(ms)	TPS	Bytes Received	Records Receiv	Tasks
Source: Custom File Source -> Split Reader: Custom File Source	FINISHED	0%	0%	-	-	0 B	0	1
Flat Map	FINISHED	0%	0%	-	-	79.87 MB	1,394,712	20
Keyed Aggregation	FINISHED	0%	0%	-	-	192.57 MB	14,389,848	20
Sink: Unnamed	FINISHED	0%	0%	-	-	192.56 MB	14,389,848	1

5. standalone 模式的 HighAvailability（HA）部署和配置

由下面这幅架构图，我们可以看到 JobManager 是整个系统中最可能导致系统不可用的角色。一个 TaskManager 挂了，如果资源足够（空闲 TaskSlot 足够）的话，则只需要把相关 task 调度到其他空闲 TaskSlot 上，然后 job 从 checkpoint 中恢复即可。而如果当前集群中只配置了一个 JobManager，则一旦 JobManager 挂了，就必须等待这个 JobManager 重新恢复，如果恢复时间过长，就可能导致整个 job 失败。



因此如果在生产业务使用 standalone 模式，则需要部署配置 HighAvailability，这样同时可以有多个 JobManager 待命，从而使得 JobManager 能够持续服务。



如果想尝试试用 Flink standalone HA 模式，需要确保基于 flink release-1.6.1 及以上版本，因为这里社区有个 bug 会导致这个模式下主 JobManager 不能正常工作。因为 flink 中的 blink 开源分支是基于 flink release-1.5.1，所以也存在这个问题。问题的详情见：https://stackoverflow.com/questions/53869850/flink-ha-standalone-cluster-failed

可以直接从 Flink 官网上下载最新版本的 Flink binary 的压缩包试用：https://flink.apache.org/downloads.html ，下载时候需要注意，接下来的实验中需要用到 hdfs，所以需要下载带有 hadoop 支持的 flink binary 包。否则，接下来初始化 standalone 集群的时候会报错，初始化 hdfs filesystem 失败。

	Scala 2.11	Scala 2.12
Apache Flink 1.7.2 only	Download (asc, sha512)	Download (asc, sha512)
Apache Flink 1.7.2 with Hadoop® 2.8	Download (asc, sha512)	Download (asc, sha512)
Apache Flink 1.7.2 with Hadoop® 2.7	Download (asc, sha512)	Download (asc, sha512)
Apache Flink 1.7.2 with Hadoop® 2.6	Download (asc, sha512)	Download (asc, sha512)
Apache Flink 1.7.2 with Hadoop® 2.4	Download (asc, sha512)	Download (asc, sha512)
Apache Flink 1.6.4 only	Download (asc, sha512)	Not supported.
Flink 1.6.4 with Hadoop® 2.8	Download (asc, sha512)	Not supported.
Flink 1.6.4 with Hadoop® 2.7	Download (asc, sha512)	Not supported.
Flink 1.6.4 with Hadoop® 2.6	Download (asc, sha512)	Not supported.
Flink 1.6.4 with Hadoop® 2.4	Download (asc, sha512)	Not supported.

（1）（可选）使用 flink 自带的脚本部署 zk

Flink 现在的 HA 需要基于 zookeeper，如果你的集群中没有，Flink 提供了启动 zookeeper 集群的脚本。首先修改配置文件“conf/zoo.cfg”，根据你要部署的 zookeeper server 的机器数来配置“server.X=addressX:peerPort:leaderPort”，其中“X”是一个 zookeeper server 的唯一 ID，且必须是数字。比如我们配置（注意修改后的配置也要推到其他机器到配置目录中去）：

```
# The port at which the clients will connect
clientPort=3181

server.1=z05f06378.sqa.zth.tbsite.net:4888:5888
server.2=z05c19426.sqa.zth.tbsite.net:4888:5888
server.3=z05f10219.sqa.zth.tbsite.net:4888:5888
```

（PS：端口一般不需要配置，因为我的环境上已经部署了一套 zookeeper，所以会导致端口占用而启动不起来，所以改了默认端口。）

然后启动 zookeeper：

```
./bin/start-zookeeper-quorum.sh
```

jps 命令看到 zk 进程已经启动：

```
$jps
20348 Jps
220415 FlinkZooKeeperQuorumPeer
```

停掉 zookeeper 集群

```
./bin/stop-zookeeper-quorum.sh
```

（2）修改 flink standalone 集群的配置

停掉之前启动到 standalone 集群：

```
./bin/stop-cluster.sh
```

修改 conf/masters 文件，增加一个 JobManager：

```
$cat conf/masters
z05f06378.sqa.zth.tbsite.net:8081
z05c19426.sqa.zth.tbsite.net:8081
```

之前修改过的 conf/slaves 文件保持不变：

```
$cat conf/slaves
z05f06378.sqa.zth.tbsite.net
z05c19426.sqa.zth.tbsite.net
z05f10219.sqa.zth.tbsite.net
```

修改 conf/flink-conf.yaml 文件：

```
# 配置 high-availability mode
high-availability: zookeeper

# 配置 zookeeper quorum（hostname 和端口需要依据对应 zk 的实际配置）
high-availability.zookeeper.quorum:
z05f02321.sqa.zth.tbsite.net:2181,z05f10215.sqa.zth.tbsite.net:2181

#（可选）设置 zookeeper 的 root 目录
high-availability.zookeeper.path.root: /test_dir/test_standalone2_root

#（可选）相当于是这个 standalone 集群中创建的 zk node 的 namespace
high-availability.cluster-id: /test_dir/test_standalone2

# JobManager 的 meta 信息放在 dfs，在 zk 上主要会保存一个指向 dfs 路径的指针
high-availability.storageDir: hdfs:///test_dir/recovery2/
```

需要注意的是，在 HA 模式下，conf/flink-conf.yaml 中的这两个配置都失效了（想想看为什么）。

```
jobmanager.rpc.address
jobmanager.rpc.port
```

修改完成后，再把这几个文件同步到不同机器到相同 conf 目录下。

启动 zookeeper 集群：

```
./bin/start-zookeeper-quorum.sh
```

```
$./bin/start-zookeeper-quorum.sh
Starting zookeeper daemon on host z05f02320.sqa.zth.
Starting zookeeper daemon on host z05c19409.sqa.zth.
Starting zookeeper daemon on host z05c17425.sqa.zth.
```

再启动 standalone 集群：

```
./bin/start-cluster.sh
```

```
$./bin/start-cluster.sh
Starting HA cluster with 2 masters.
Starting standalonesession daemon on host z05f02320.sqa.zth.
Starting standalonesession daemon on host z05c19409.sqa.zth.
Starting taskexecutor daemon on host z05f02320.sqa.zth.
Starting taskexecutor daemon on host z05c19409.sqa.zth.
Starting taskexecutor daemon on host z05c17425.sqa.zth.
```

分别打开：

http://z05f06378.sqa.zth.tbsite.net:8081
http://z05c19426.sqa.zth.tbsite.net:8081

可以看到两个页面最后都转到了同一个地址上，这个地址就是当前主 JobManager 所在机器，另一个就是 standby JobManager。

当我们知道主 JobManager 后，我们可以把主 JobManager 进程 kill 掉，比如当前主 JobManager 在 z05c19426.sqa.zth.tbsite.net 这个机器上，就把这个进程杀掉：

```
[appadmin@z05c19409.sqa.zth /home/appadmin/shengyang.ssy/flink-1.7.2]
$jps
243845 TaskManagerRunner
243050 StandaloneSessionClusterEntrypoint
20330 Jps
242029 FlinkZooKeeperQuorumPeer

[appadmin@z05c19409.sqa.zth /home/appadmin/shengyang.ssy/flink-1.7.2]
$kill -9 243050

[appadmin@z05c19409.sqa.zth /home/appadmin/shengyang.ssy/flink-1.7.2]
$jps
20528 Jps
243845 TaskManagerRunner
242029 FlinkZooKeeperQuorumPeer
```

接着，再打开这两个链接：

http://z05f06378.sqa.zth.tbsite.net:8081
http://z05c19426.sqa.zth.tbsite.net:8081

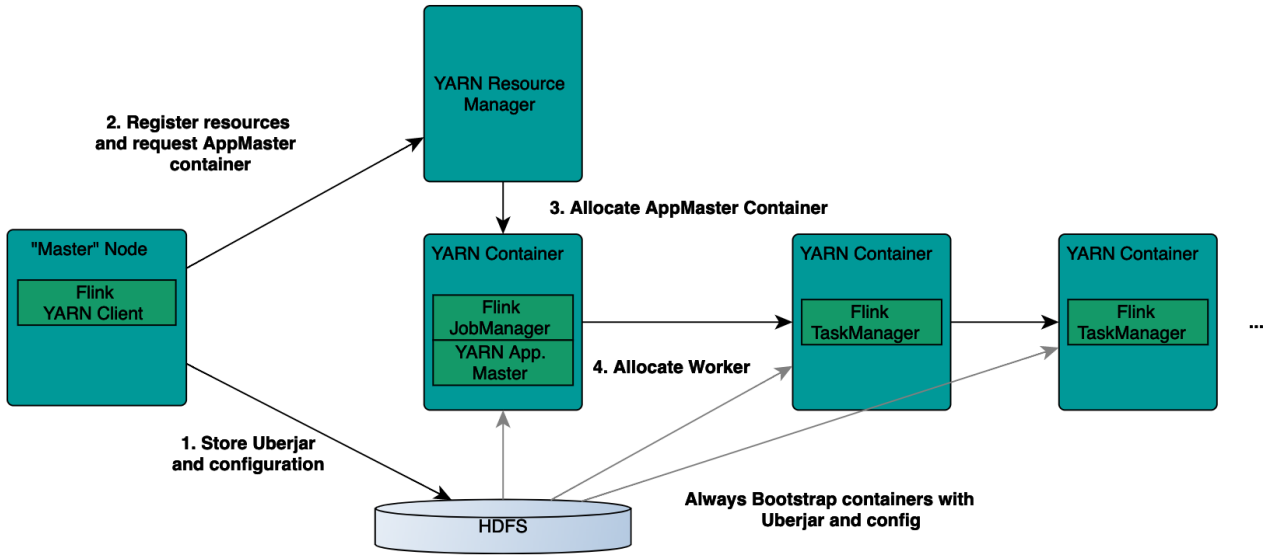
可以发现后一个链接已经不能展示了，而前一个链接可以展示，说明发生主备切换。

然后我们再重启 JobManager：

```
./bin/jobmanager.sh start z05c19426.sqa.zth.tbsite.net 8081
```

再打开 http://z05c19426.sqa.zth.tbsite.net:8081 这个链接，会发现现在这个链接可以转到 http://z05f06378.sqa.zth.tbsite.net:8081 这个页面上了。说明这个 JobManager 完成了一个 Failover recovery。

6. 使用 yarn 模式跑 flink job



什么情况下适合使用 yarn 模式跑 flink job？

相对于 standalone 模式，yarn 模式允许 flink job 的好处有：

- 资源按需使用，提高集群的资源利用率
- 任务有优先级，根据优先级运行作业
- 基于 YARN 调度系统，能够自动化地处理各个角色的 failover
    - JobManager 进程和 TaskManager 进程都由 Yarn NodeManager 监控
    - 如果 JobManager 进程异常退出，则 Yarn ResourceManager 会重新调度 JobManager 到其他机器
    - 如果 TaskManager 进程异常退出，JobManager 会收到消息并重新向 Yarn ResourceManager 申请资源，重新启动 TaskManager

（1）在 YARN 上启动 long running 的 flink 集群（yarn session）

我们演示用的 YARN 集群：http://z05c19394.sqa.zth.tbsite.net:8088/cluster

查看命令参数：

```
./bin/yarn-session.sh -h
```

创建一个 YARN 模式的 flink 集群

```
./bin/yarn-session.sh -n 4 -jm 1024m -tm 4096m
```

其中用到的参数是：

- -n,--container <arg> Number of TaskManagers
- -jm,--jobManagerMemory <arg> Memory for JobManager Container with optional unit (default: MB)
- -tm,--taskManagerMemory <arg> Memory per TaskManager Container with optional unit (default: MB)
- -qu,--queue <arg> Specify YARN queue.
- -s,--slots <arg> Number of slots per TaskManager
- -t,--ship <arg> Ship files in the specified directory (t for transfer)

提交一个 flink job 到 flink 集群：

```
./bin/flink run examples/streaming/WordCount.jar --input hdfs:///test_dir/input_dir/story --output hdfs:///test_dir/output_dir/output
```

这次提交 flink job，虽然没有指定对应 yarn application 的信息，确可以提交到对应的 flink 集群，原因在于“/tmp/.yarn-properties-${user}”文件中保存了上一次创建 yarn session 的集群信息。所以如果同一用户在同一机器上再次创建一个 yarn session，则这个文件会被覆盖掉。

那如果删掉“/tmp/.yarn-properties-${user}”或者在另一个机器上提交作业能否提交到预期到 yarn session 中呢？

这也是可以的，如果配置了 HighAvailability，则可以根据 cluster-id，从 zookeeper 上获取到 JobManager 的地址和端口，从而提交作业。

```
high-availability.cluster-id
```

如果 Yarn session 没有配置 HA，又该如何提交呢？

这个时候就必须要在提交 flink job 的命令中指明 YARN 上的 application id，通过“-yid”参数传入：

```
/bin/flink run -yid application_1548056325049_0048 examples/streaming/WordCount.jar --input hdfs:///test_dir/input_dir/story --output hdfs:///test_dir/output_dir/output
```

我们可以发现，每次跑完任务不久，TaskManager 就没有了，下次在提交任务的时候，TaskManager 又会重新拉起来。如果希望 TaskManager 启动后就持续运行，可以在 conf/flink-conf.yaml 文件中配置下面这个参数，单位是 milliseconds，默认值是 30000L，即 30 秒：

```
resourcemanager.taskmanager-timeout
```

（2）在 YARN 上运行单个 flink job

如果你只想运行单个 flink job 后就退出，那么可以用下面这个命令：

```
./bin/flink run -m yarn-cluster -yn 2 examples/streaming/WordCount.jar --input hdfs:///test_dir/input_dir/story --output hdfs:///test_dir/output_dir/output
```

常用的配置有：

- -yn,--yarncontainer <arg> Number of Task Managers
- -yqu,--yarnqueue <arg> Specify YARN queue.
- -ys,--yarnslots <arg> Number of slots per TaskManager
- -yqu,--yarnqueue <arg> Specify YARN queue.

可以通过 help 命令查看 run 的可用参数：

```
./bin/flink run -h
```

我们可以看到，“./bin/flink run -h”看到的“Options for yarn-cluster mode”中的“-y”和“--yarn”为前缀的参数其实和“./bin/yarn-session.sh -h”命令是一一对应的，语义上也基本一致。

关于“-n”（在 yarn session 模式下）、“-yn”在（yarn single job 模式下）与“-p”参数的关系：

1. “-n”和“-yn”在社区版本中（release-1.5 ~ release-1.7）中没有实际的控制作用，实际的资源是根据“-p”参数来申请的，并且 TM 使用完后就会归还
2. 在 blink 的开源版本中，“-n”（在 yarn session 模式下）的作用就是一开始启动指定数量的 TaskManager，之后即使 job 需要更多的 slot，也不会申请新的 TaskManager

3. 在 blink 的开源版本中，yarn single job 模式“-yn”表示的是初始 TaskManager 的数量，不设置 TaskManager 的上限。（需要特别注意的是，只有加上“-yd”参数才能用 single job 模式（例如：命令“./bin/flink run -yd -m yarn-cluster xxx”）

7. Yarn 模式下的 HighAvailability 配置

首先要确保启动 Yarn 集群用的“yarn-site.xml”文件中的这个配置，这个是 YARN 集群级别 AM 重启的上限。

```
 <property>
  <name>yarn.resourcemanager.am.max-attempts</name>
  <value>100</value>
 </property>
```

然后在 conf/flink-conf.yaml 文件中配置这个 flink job 的 JobManager 能够重启的次数（1+ 9 retries）

```
yarn.application-attempts: 10
```

最后再在 conf/flink-conf.yaml 文件中配置上 zk 相关配置，这几个配置的配置方法和 standalone 的 HA 配置方法基本一致，如下所示。

需要特别注意的是：“high-availability.cluster-id”这个配置要去掉，因为在 Yarn（以及 mesos）模式下，cluster-id 如果不配置的话，会配置成 Yarn 上的 application id，从而可以保证唯一性。否则，在提交作业的时候需要用户去保证 cluster-id 的全局唯一性。

```
# 配置 high-availability mode
high-availability: zookeeper

# 配置 zookeeper quorum（hostname 和端口需要依据对应 zk 的实际配置）
high-availability.zookeeper.quorum:
z05f02321.sqa.zth.tbsite.net:2181,z05f10215.sqa.zth.tbsite.net:2181

# （可选）设置 zookeeper 的 root 目录
high-availability.zookeeper.path.root: /test_dir/test_standalone2_root

# （可选）相当于是这个 standalone 集群中创建的 zk node 的 namespace
# high-availability.cluster-id: /test_dir/test_standalone2

# JobManager 的 meta 信息放在 dfs，在 zk 上主要会保存一个指向 dfs 路径的指针
high-availability.storageDir: hdfs:///test_dir/recovery2/
```