

## 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 <a href="#">Oracle Binary Code License Agreement for Java SE</a> to download this software.		
点击Accept	<input checked="" type="radio"/> Accept License Agreement	<input type="radio"/> Decline License Agreement
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.86 MB	<a href="#">jdk-8u202-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.75 MB	<a href="#">jdk-8u202-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	173.08 MB	<a href="#">jdk-8u202-linux-i586.rpm</a>
Linux x86	187.9 MB	<a href="#">jdk-8u202-linux-i586.tar.gz</a>
Linux x64	170.15 MB	<a href="#">jdk-8u202-linux-x64.rpm</a>
Linux x64	185.05 MB	<a href="#">jdk-8u202-linux-x64.tar.gz</a>
Mac OS X x64	249.15 MB	<a href="#">jdk-8u202-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	125.09 MB	<a href="#">jdk-8u202-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	88.1 MB	<a href="#">jdk-8u202-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	124.37 MB	<a href="#">jdk-8u202-solaris-x64.tar.Z</a>
Solaris x64	85.38 MB	<a href="#">jdk-8u202-solaris-x64.tar.gz</a>
Windows x86	201.64 MB	<a href="#">jdk-8u202-windows-i586.exe</a>
Windows x64	211.58 MB	<a href="#">jdk-8u202-windows-x64.exe</a>

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/1cUq9BaHSxEelKBPKYE8YyFQ9LD6EW8yB/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：



```
→ flink git:(blink) ls -lrt flink-dist/target
total 584340
-rw-r--r-- 1 22212 Mar 16 21:08 checkstyle-checker.xml
-rw-r--r-- 1 2466 Mar 16 21:08 checkstyle-suppressions.xml
-rw-r--r-- 1 367 Mar 16 21:08 checkstyle-result.xml
drwxr-xr-x 4 128 Mar 16 21:08 classes
drwxr-xr-x 3 96 Mar 16 21:08 maven-shared-archive-resources
drwxr-xr-x 3 96 Mar 16 21:08 generated-test-sources
drwxr-xr-x 5 160 Mar 16 21:08 test-classes
drwxr-xr-x 3 96 Mar 16 21:08 maven-archiver
drwxr-xr-x 2 64 Mar 16 21:08 archive-tmp
drwxr-xr-x 3 96 Mar 16 21:08 flink-1.5.1-bin
-rw-r--r-- 1 16334 Mar 17 19:12 original-flink-dist_2.11-1.5.1.jar
-rw-r--r-- 1 179602798 Mar 17 19:12 flink-dist_2.11-1.5.1.jar
-rw-r--r-- 1 409476789 Mar 17 19:12 flink-1.5.1.tar.gz
→ flink git:(blink)
```

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。

```
drwxr-xr-x 5 chengman 2017-03-16 19:21 flink-docs
drwxr-xr-x 2 2017-03-16 23:38 target
drwxr-xr-x 2 2017-03-16 23:38 flink-annotations
drwxr-xr-x 2 2017-03-16 23:38 flink-shaded-hadoop
drwxr-xr-x 2 2017-03-16 23:39 flink-shaded-curator
drwxr-xr-x 2 2017-03-16 23:39 flink-test-utils-parent
drwxr-xr-x 2 2017-03-16 23:39 flink-metrics
drwxr-xr-x 2 2017-03-16 23:39 flink-core
drwxr-xr-x 2 2017-03-16 23:39 flink-java
drwxr-xr-x 2 2017-03-16 23:39 flink-queryable-state
drwxr-xr-x 2 2017-03-16 23:39 flink-fileSystems
drwxr-xr-x 2 2017-03-16 23:39 flink-runtime
drwxr-xr-x 2 2017-03-16 23:40 flink-optimizer
drwxr-xr-x 2 2017-03-16 23:40 flink-clients
drwxr-xr-x 2 2017-03-16 23:40 flink-streaming-java
drwxr-xr-x 2 2017-03-16 23:40 flink-scala
drwxr-xr-x 2 2017-03-16 23:40 flink-examples
drwxr-xr-x 2 2017-03-16 23:41 flink-state-backends
drwxr-xr-x 2 2017-03-16 23:41 flink-libraries
drwxr-xr-x 2 2017-03-16 23:41 flink-javag8
lrwxr-xr-x 1 2017-03-17 19:12 build-target -> /Users/.../Workspace/orig_flink/flink/flink-dist/target/flink-1.5.1-bin/flink-1.5.1
```

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：

```

[INFO] flink-yarn-tests ..... SKIPPED
[INFO] flink-fs-tests ..... SKIPPED
[INFO] flink-docs ..... SKIPPED
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 03:34 min (Wall Clock)
[INFO] Finished at: 2019-03-17T11:42:27+00:00
[INFO] Highest basedir set to: /home/pi/codePath/flink/flink
[INFO] --- maven-remote-resources-plugin:1.5:process (process-resource-bundles) @ flink-s3-fs-base ---
[INFO] Final Memory: 147M/212M
[INFO] -----
[ERROR] Failed to execute goal on project flink-mapr-fs: Could not resolve dependencies for project org.apache.flink:flink-mapr-fs:jar:1.9-SNAPSHOT:
Failed to collect dependencies at com.mapr.hadoop:maprfs:jar:5.2.1-mapr -> org.apache.hadoop:hadoop-auth:jar:2.7.0-mapr-1703: Failed to read artifact descriptor for org.apache.hadoop:hadoop-auth:jar:2.7.0-mapr-1703: Failure to find org.apache.hadoop:hadoop-project:pom:2.7.0-mapr-1703 in https://maven.aliyun.com/repository/mapr-public was cached in the local repository, resolution will not be reattempted until the update interval of mapr-public has elapsed or updates are forced -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/DependencyResolutionException
[ERROR]
[ERROR] After correcting the problems, you can resume the build with the command
[ERROR] mvn <goals> -rf :flink-mapr-fs
[INFO] -----
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ flink-swift-fs-hadoop ---

```

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
```

```

[INFO] Final Memory: 147M/212M
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-compile) on project flink-mapr-fs: Compilation failure: Compilation failure:
[ERROR] /home/pi/codePath/flink/flink/flink-fileSystems/flink-mapr-fs/src/main/java/org/apache/flink/runtime/fs/maprfs/MapRFileSystem.java:[70,44] package org.apache.hadoop.fs does not exist
[ERROR] /home/pi/codePath/flink/flink/flink-fileSystems/flink-mapr-fs/src/main/java/org/apache/flink/runtime/fs/maprfs/MapRFileSystem.java:[73,45] cannot find symbol
[ERROR] symbol:   class Configuration
[ERROR] location: package org.apache.hadoop.conf
[ERROR] /home/pi/codePath/flink/flink/flink-fileSystems/flink-mapr-fs/src/main/java/org/apache/flink/runtime/fs/maprfs/MapRFileSystem.java:[73,93] cannot find symbol
[ERROR] symbol:   class Configuration
[ERROR] location: package org.apache.hadoop.conf
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
[ERROR]
[ERROR] After correcting the problems, you can resume the build with the command
[ERROR] mvn <goals> -rf :flink-mapr-fs
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ flink-swift-fs-hadoop ---

```

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。

```
[INFO] Node v10.9.0 is already installed.
[INFO] --- frontend-maven-plugin:1.6:npm (npm install) @ flink-runtime-web_2.11 ---
[INFO] Running 'npm install --cache-max=0 --no-save' in /mnt/d/CodePath/flink/flink/flink-runtime-web/web-dashboard
[WARNING] npm WARN rollback Rolling back readable-stream@2.3.6 failed (this is probably harmless): EINVAL: invalid argument, lstat '/mnt/d/CodePath/flink/flink/flink-runtime-web/web-dashboard/node_modules/fsevents/node_modules'
[WARNING] npm WARN @ng-zorro/ng-plus@0.0.31 requires a peer of monaco-editor@>=0.15.6 but none is installed. You must install peer dependencies yourself.
[WARNING] npm WARN rollup-plugin-commonjs@9.2.0 requires a peer of rollup@>=0.56.0 but none is installed. You must install peer dependencies yourself.
[WARNING] npm WARN rollup-plugin-sourcemaps@0.4.2 requires a peer of rollup@>=0.31.2 but none is installed. You must install peer dependencies yourself.
[WARNING] npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules/fsevents):
[WARNING] npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
[ERROR]
[INFO] audited 46961 packages in 15.059s
[INFO] found 8 vulnerabilities (5 low, 3 high)
[INFO] run 'npm audit fix' to fix them, or 'npm audit' for details
[INFO] --- frontend-maven-plugin:1.6:npm (npm run build) @ flink-runtime-web_2.11 ---
[INFO] Running 'npm run build' in /mnt/d/CodePath/flink/flink/flink-runtime-web/web-dashboard
[INFO] > flink-runtime-web@1.0.0 build /mnt/d/CodePath/flink/flink/flink-runtime-web/web-dashboard
[INFO] > ng build --prod --base-href ./
```

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



### (3) 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。

### (4) 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。

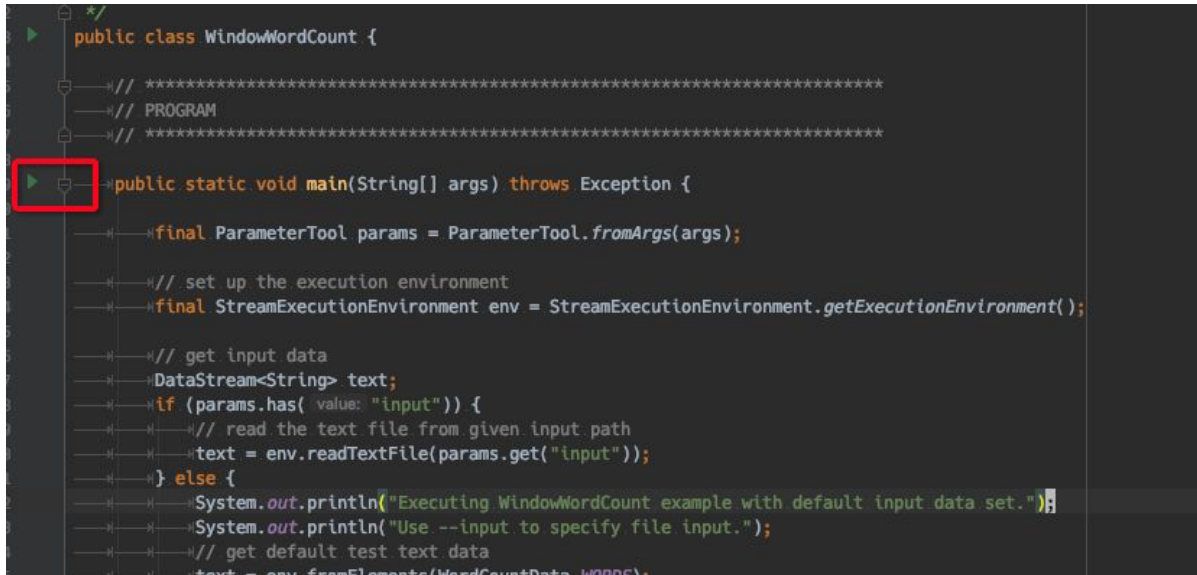
#### (5) 添加Scala的Checkstyle

- 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

(6) 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

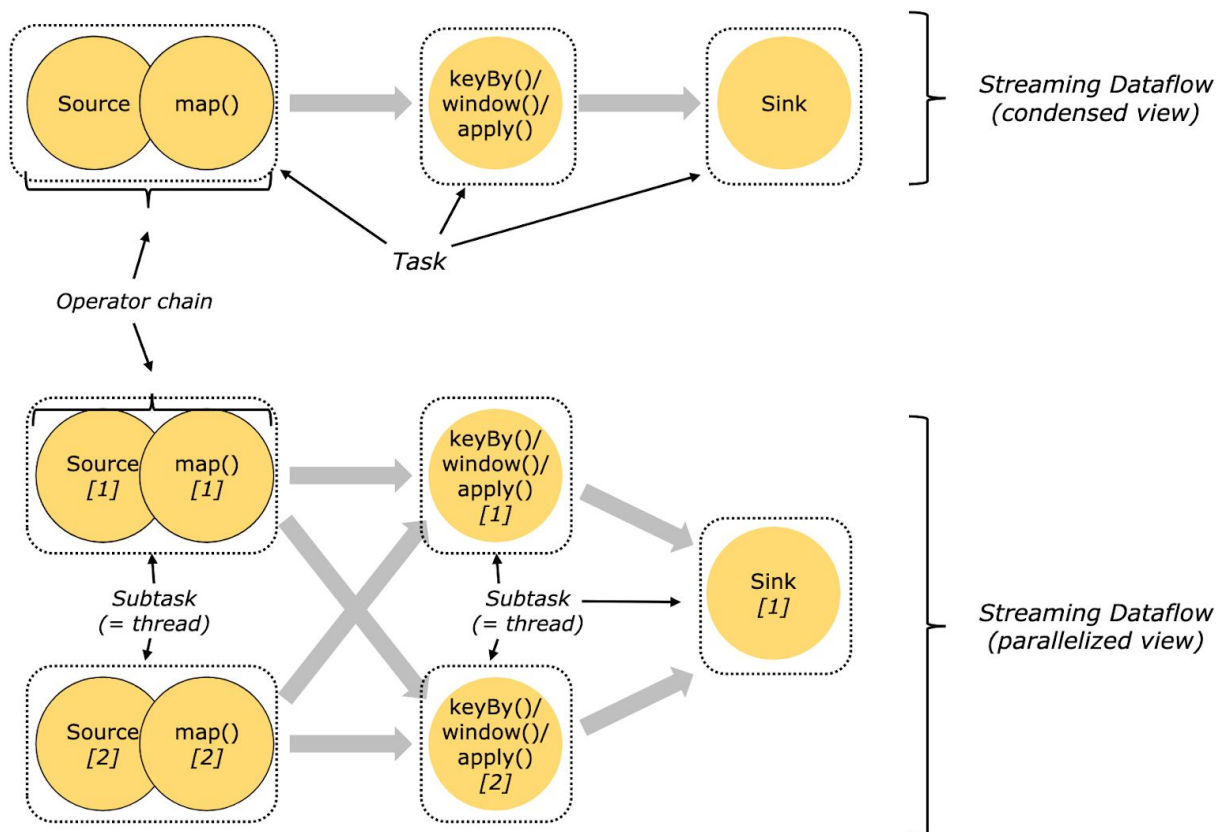


```
public class WindowWordCount {  
    /**  
     * *****  
     * // PROGRAM  
     * *****  
     */  
    public static void main(String[] args) throws Exception {  
        final ParameterTool params = ParameterTool.fromArgs(args);  
        // set up the execution environment  
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
        // get input data  
        DataStream<String> text;  
        if (params.has( "value: \"input\"")) {  
            // read the text file from given input path  
            text = env.readTextFile(params.get("input"));  
        } else {  
            System.out.println("Executing WindowWordCount example with default input data set.");  
            System.out.println("Use --input to specify file input.");  
            // get default test text data  
            text = env.fromElements(WordCountData.WORDS);  
        }  
    }  
}
```

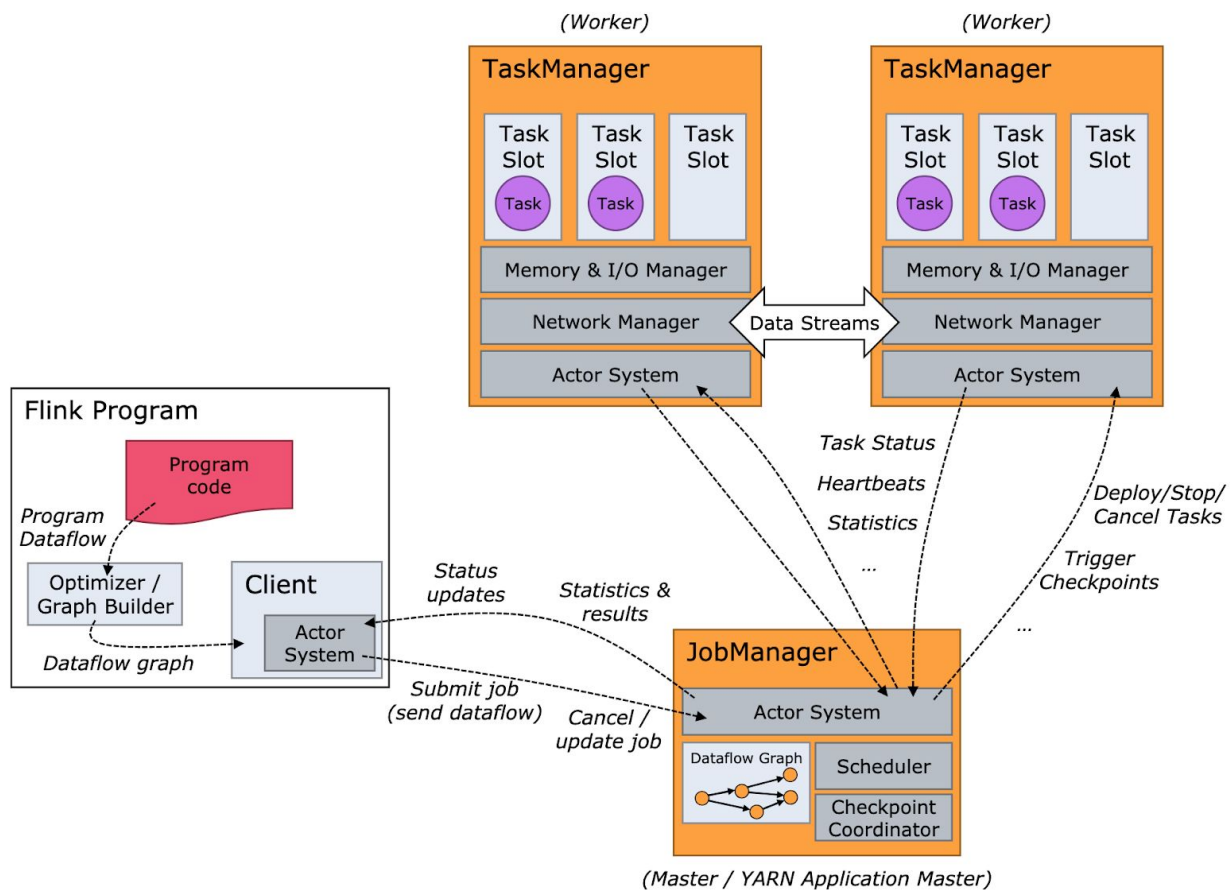
## 二、运行Flink应用

### 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。

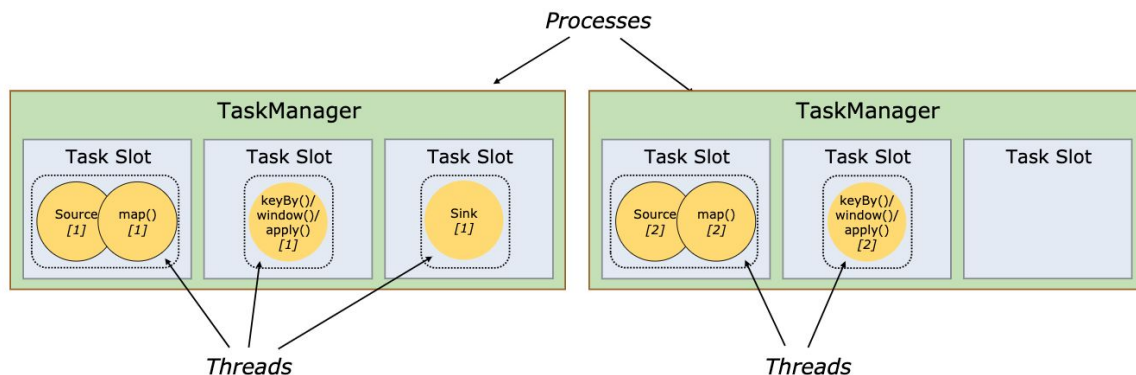


通过这张图我们可以看到，在一个DAG图中，不能被chain在一起的operator会被分隔到不同的Task中，也就是说，Task是Flink中资源调度的最小单位。



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、协调创建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

```
➔ flink git:(blink) ls -lrt flink-dist/target
total 584340
-rw-r--r-- 1 22212 Mar 16 21:08 checkstyle-checker.xml
-rw-r--r-- 1 2466 Mar 16 21:08 checkstyle-suppressions.xml
-rw-r--r-- 1 367 Mar 16 21:08 checkstyle-result.xml
drwxr-xr-x 4 128 Mar 16 21:08 classes
drwxr-xr-x 3 96 Mar 16 21:08 maven-shared-archive-resources
drwxr-xr-x 3 96 Mar 16 21:08 generated-test-sources
drwxr-xr-x 5 160 Mar 16 21:08 test-classes
drwxr-xr-x 3 96 Mar 16 21:08 maven-archiver
drwxr-xr-x 2 64 Mar 16 21:08 archive-tmp
drwxr-xr-x 3 96 Mar 16 21:08 flink-1.5.1-bin
-rw-r--r-- 1 16334 Mar 17 19:12 original-flink-dist_2.11-1.5.1.jar
-rw-r--r-- 1 179602798 Mar 17 19:12 flink-dist_2.11-1.5.1.jar
-rw-r--r-- 1 409476789 Mar 17 19:12 flink-1.5.1.tar.gz
➔ flink git:(blink)
```

```
drwxr-xr-x 6 shengmao 4096 Mar 16 21:09 flink-docs
drwxr-xr-x 2 Mar 16 23:38 target
drwxr-xr-x 2 Mar 16 23:38 flink-annotations
drwxr-xr-x 2 Mar 16 23:38 flink-shaded-hadoop
drwxr-xr-x 2 Mar 16 23:39 flink-shaded-curator
drwxr-xr-x 2 Mar 16 23:39 flink-test-utils-parent
drwxr-xr-x 2 Mar 16 23:39 flink-metrics
drwxr-xr-x 2 Mar 16 23:39 flink-core
drwxr-xr-x 2 Mar 16 23:39 flink-java
drwxr-xr-x 2 Mar 16 23:39 flink-queryable-state
drwxr-xr-x 2 Mar 16 23:39 flink-fileSystems
drwxr-xr-x 2 Mar 16 23:39 flink-runtime
drwxr-xr-x 2 Mar 16 23:40 flink-optimizer
drwxr-xr-x 2 Mar 16 23:40 flink-clients
drwxr-xr-x 2 Mar 16 23:40 flink-streaming-java
drwxr-xr-x 2 Mar 16 23:40 flink-scala
drwxr-xr-x 2 Mar 16 23:40 flink-examples
drwxr-xr-x 2 Mar 16 23:41 flink-state-backends
drwxr-xr-x 2 Mar 16 23:41 flink-libraries
drwxr-xr-x 2 Mar 16 23:41 flink-javag
lrwxr-xr-x 1 Mar 17 19:12 build-target -> /Users/.../Workspace/orig_flink/Flink/Flink-dist/target/Flink-1.5.1-bin/Flink-1.5.1
```

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

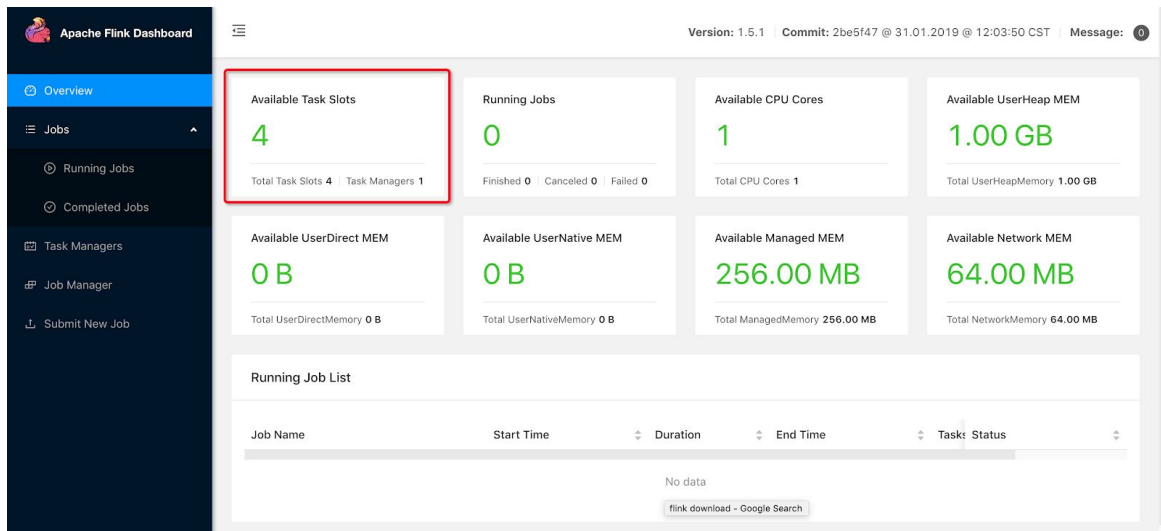
最简单的运行flink应用的方法就是以单机standalone的方式运行。进入Flink binary的目录下（一般情况下不需要修改配置文件），执行：

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

输出结果如下，表示启动正常：

```
➔ flink-1.5.1 ./bin/start-cluster.sh
Starting cluster.
Starting standalone session daemon on host ali-6c96cfd97dcb.local.
log4j:WARN No appenders could be found for logger (org.apache.flink.configuration.GlobalConfiguration).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Starting taskexecutor daemon on host ali-6c96cfd97dcb.local.
➔ flink-1.5.1
```

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





提交一个Word Count的任务， 命令如下：

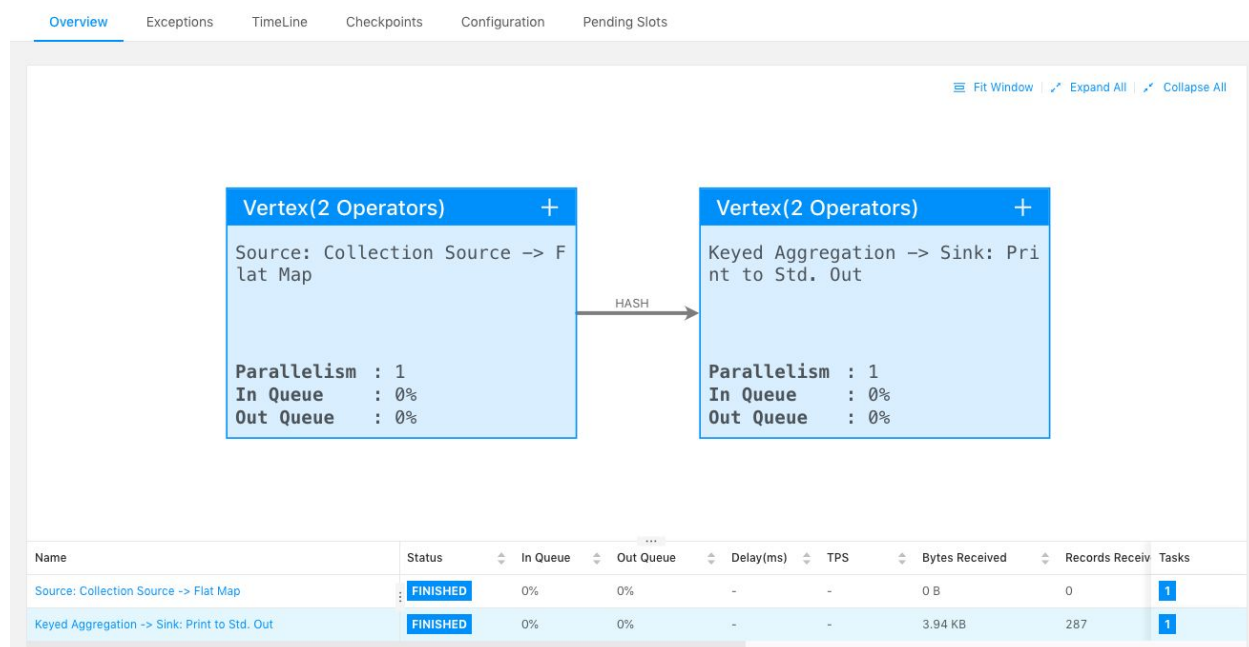
```
./bin/flink run examples/streaming/WordCount.jar
```

可以在web界面上看到job完成了

Running Job List						
Job Name	Start Time	Duration	End Time	Ta	Status	
No data						

Completed Job List						
Job Name	Start Time	Duration	End Time	Ta	Status	
Streaming WordCount	2019-03-19 19:17:07	0.46s	2019-03-19 19:17:07	2	FINISHED	

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：

akka.tcp://flink@ali-6c96cfd97dcb.local:50075/user/taskmanager\_0 | 7787f5f9b1fdf839f49561fee82a533c

Data Port: **50077** | Free Slots / All Slots: **4 / 4** | CPU Cores: **8** | Physical Memory: **16.00 GB** | JVM Heap Size: **1.38 GB**

Allocated Resource Metrics **Log**

Log List / flink-shengyang.ssy-taskexecutor-0-ali-6c96cfd97dcb.local.out < 1 >

```
1 (to,1)
2 (be,1)
3 (or,1)
4 (not,1)
5 (to,2)
6 (be,2)
7 (that,1)
8 (is,1)
9 (the,1)
10 (question,1)
11 (whether,1)
12 (tis,1)
13 (nobler,1)
14 (in,1)
15 (the,2)
16 (mind,1)
17 (to,3)
18 (suffer,1)
19 (the,3)
20 (slings,1)
21 (and,1)
22 (arrows,1)
23 (of,1)
```

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：

```
→ flink-1.5.1 jps
45472 StandaloneSessionClusterEntrypoint
45812 Jps
45798 TaskManagerRunner
25272
25480 Launcher
25471 RemoteMavenServer
→ flink-1.5.1 ps aux | grep 45472
shengyang.ssy 45472 0.8 1.9 6950516 318736 s003 S 3:32PM 0:07.44 /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java -Xms1024m -Xmx1024m -Dlog.file=/Users/shengyang.ssy/Workspace/Flink-1.5.1/log/flink-shengyang.ssy-standalonesession-0-ali-6c96cf9d97dcb.local.log -Dcode.file=/Users/shengyang.ssy/Workspace/Flink-1.5.1/log/flink-shengyang.ssy-standalonesession-0-ali-6c96cf9d97dcb.local.code -Dlog4j.configuration=file:/Users/shengyang.ssy/Workspace/Flink-1.5.1/conf/log4j.properties -Dlogback.configurationFile=file:/Users/shengyang.ssy/Workspace/Flink-1.5.1/conf/logback.xml -Xloggc:/Users/shengyang.ssy/Workspace/Flink-1.5.1/log/flink-shengyang.ssy-standalonesession-0-ali-6c96cf9d97dcb.local-gc.log -classpath /Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/flink-python_2.11-1.5.1.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/flink-shaded-hadoop2-uber-1.5.1.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/log4j-1.2.17.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/slf4j-log4j12-1.7.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/flink-dist_2.11-1.5.1.jar:: org.apache.flink.runtime.entrypoint.StandaloneSessionClusterEntrypoint --configDir /Users/shengyang.ssy/Workspace/Flink-1.5.1/conf --executionMode cluster
shengyang.ssy 45824 0.0 0.0 4259320 348 s003 R+ 3:32PM 0:00.00 grep --color=auto --exclude-dir=.bzz --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn 45472
→ flink-1.5.1 ps aux | grep 45798
shengyang.ssy 45798 1.0 2.6 7540096 432916 s003 S 3:32PM 0:06.63 /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java -XX:+UseG1GC -Xms1408M -Xmx1408M -Xmn382M -XX:MaxDirectMemorySize=1088M -Dlog.file=/Users/shengyang.ssy/Workspace/Flink-1.5.1/log/flink-shengyang.ssy-taskexecutor-0-ali-6c96cf9d97dcb.local.log -Dcode.file=/Users/shengyang.ssy/Workspace/Flink-1.5.1/log/flink-shengyang.ssy-taskexecutor-0-ali-6c96cf9d97dcb.local.code -Dlog4j.configuration=file:/Users/shengyang.ssy/Workspace/Flink-1.5.1/conf/log4j.properties -Dlogback.configurationFile=file:/Users/shengyang.ssy/Workspace/Flink-1.5.1/conf/logback.xml -Xloggc:/Users/shengyang.ssy/Workspace/Flink-1.5.1/log/flink-shengyang.ssy-taskexecutor-0-ali-6c96cf9d97dcb.local-gc.log -classpath /Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/flink-python_2.11-1.5.1.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/flink-shaded-hadoop2-uber-1.5.1.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/log4j-1.2.17.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/slf4j-log4j12-1.7.jar:/Users/shengyang.ssy/Workspace/Flink-1.5.1/lib/flink-dist_2.11-1.5.1.jar:: org.apache.flink.runtime.taskexecutor.TaskManagerRunner --configDir /Users/shengyang.ssy/Workspace/Flink-1.5.1/conf
shengyang.ssy 45841 0.0 0.0 4259328 416 s003 R+ 3:33PM 0:00.00 grep --color=auto --exclude-dir=.bzz --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn 45798
→ flink-1.5.1
```

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

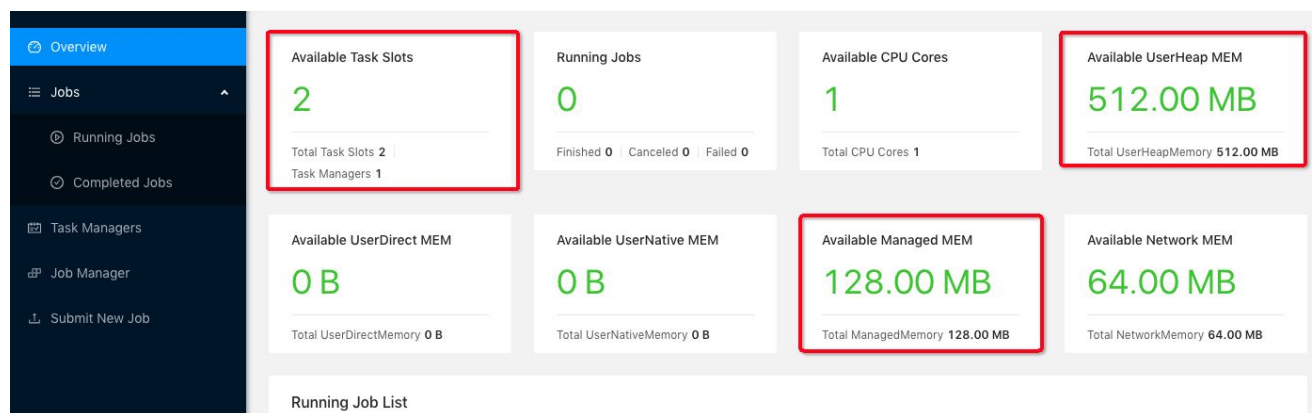
# The number of task slots that each TaskManager offers. Each slot runs
one parallel pipeline.
taskmanager.numberOfTaskSlots: 2

# 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：

```
→ flink-1.5.1 jps
41717 StandaloneSessionClusterEntrypoint
42037 TaskManagerRunner
25272
25480 Launcher
42093 Jps
25471 RemoteMavenServer
→ flink-1.5.1 ps aux | grep 41717
shengyang.ssy  41717  0.7 1.8 6953824 308328 s003 S   3:10PM  0:08.46 /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java -Xms1024m -Xmx1024m -Dlog.file=/Users/shengyang.ssy/Workspace/flink-1.5.1/log/flink-shengyang.ssy-standalonesession-0-ali-6c96cf9d97dcb.local.log -Dcode.file=/Users/shengyang.ssy/Workspace/flink-1.5.1/log/flink-shengyang.ssy-standalonesession-0-ali-6c96cf9d97dcb.local_code -Dlog4j.configuration=file:/Users/shengyang.ssy/Workspace/flink-1.5.1/conf/log4j.properties -Dlogback.configurationFile=file:/Users/shengyang.ssy/Workspace/flink-1.5.1/conf/logback.xml -Xloggc:/Users/shengyang.ssy/Workspace/flink-1.5.1/log/flink-shengyang.ssy-standalonesession-0-ali-6c96cf9d97dcb.local-gc.log -classpath /Users/shengyang.ssy/Workspace/flink-1.5.1/lib/flink-python_2.11-1.5.1.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/flink-shaded-hadoop2-uber-1.5.1.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/log4j-1.2.17.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/slf4j-log4j12-1.7.7.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/flink-dist-2.11-1.5.1.jar:: org.apache.flink.runtime.entrypoint.StandaloneSessionClusterEntrypoint -configDir /Users/shengyang.ssy/Workspace/flink-1.5.1/conf --executionMode cluster
shengyang.ssy  42107  0.0 0.0 4259320 328 s003 R+   3:11PM  0:00.00 grep --color=auto --exclude-dir=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn 41717
→ flink-1.5.1 ps aux | grep 42037
shengyang.ssy  42037  1.0 2.1 6791212 347652 s003 S   3:10PM  0:07.86 /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java -XX:+UseG1GC -Xms768M -Xmx768M -Xmn192M -XX:MaxDirectMemorySize=1088M -Dlog.file=/Users/shengyang.ssy/Workspace/flink-1.5.1/log/flink-shengyang.ssy-taskexecutor-0-ali-6c96cf9d97dcb.local.log -Dcode.file=/Users/shengyang.ssy/Workspace/flink-1.5.1/log/flink-shengyang.ssy-taskexecutor-0-ali-6c96cf9d97dcb.local_code -Dlog4j.configuration=file:/Users/shengyang.ssy/Workspace/flink-1.5.1/conf/log4j.properties -Dlogback.configurationFile=file:/Users/shengyang.ssy/Workspace/flink-1.5.1/conf/logback.xml -Xloggc:/Users/shengyang.ssy/Workspace/flink-1.5.1/log/flink-shengyang.ssy-taskexecutor-0-ali-6c96cf9d97dcb.local-gc.log -classpath /Users/shengyang.ssy/Workspace/flink-1.5.1/lib/flink-python_2.11-1.5.1.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/flink-shaded-hadoop2-uber-1.5.1.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/log4j-1.2.17.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/slf4j-log4j12-1.7.7.jar:/Users/shengyang.ssy/Workspace/flink-1.5.1/lib/flink-dist-2.11-1.5.1.jar:: org.apache.flink.runtime.taskexecutor.TaskManagerRunner -configDir /Users/shengyang.ssy/Workspace/flink-1.5.1/conf
shengyang.ssy  42128  0.0 0.0 4259328 448 s003 R+   3:11PM  0:00.00 grep --color=auto --exclude-dir=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn 42037
→ flink-1.5.1
```

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

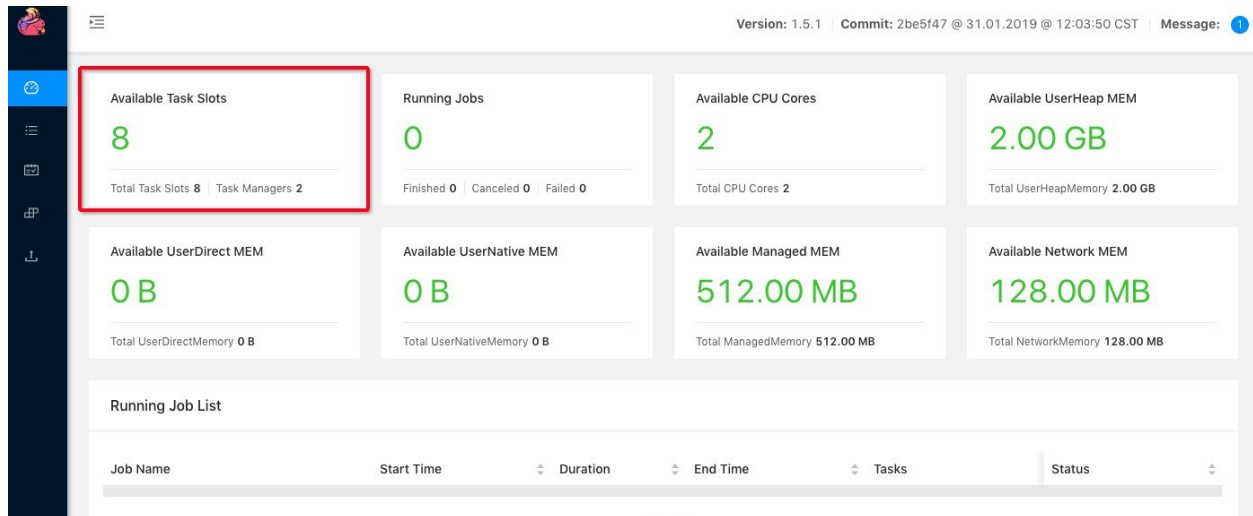
接下来我们可以在上面的基础上再次调用这个命令（重复调用start-cluster.sh命令）：

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

命令的输出结果如下所示：

```
→ flink-1.5.1 ./bin/start-cluster.sh
Starting cluster.
[INFO] 1 instance(s) of standalone-session are already running on ali-6c96cfd97dcb.local.
Starting standalone-session daemon on host ali-6c96cfd97dcb.local.
log4j:WARN No appenders could be found for logger (org.apache.flink.configuration.GlobalConfiguration).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[INFO] 1 instance(s) of taskexecutor are already running on ali-6c96cfd97dcb.local.
Starting taskexecutor daemon on host ali-6c96cfd97dcb.local.
→ flink-1.5.1
```

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“\*-standalone-session-1-\*”和“\*-taskexecutor-1-\*”这两种日志。我们打开“\*-standalone-session-1-\*.log”这个文件，翻到最下面，可以看到：

```

2019-03-19 20:59:01,194 INFO org.apache.flink.runtime.entrypoint.ClusterEntrypoint - Install security context.
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.

```

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：

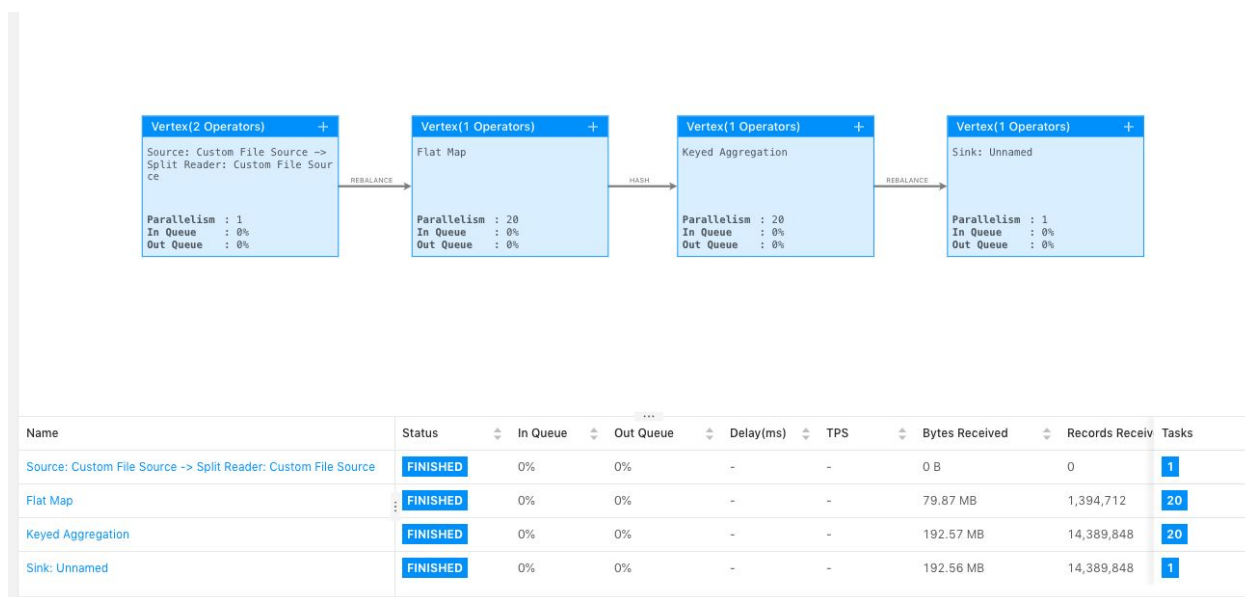
```
hdfs dfs -copyFromLocal story /test_dir/input_dir/story
```

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）：

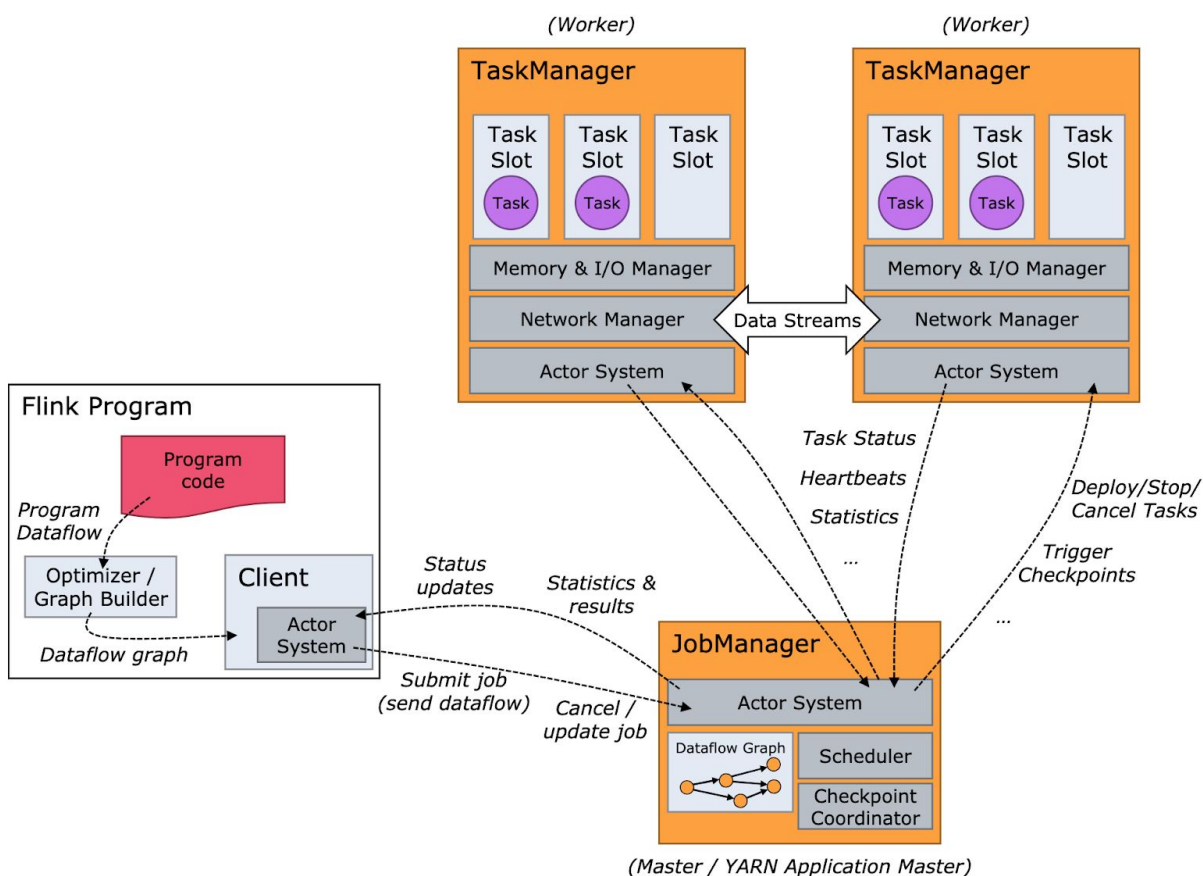
```
./bin/flink run examples/streaming/WordCount.jar --input  
hdfs:///test_dir/input_dir/story --output hdfs:///test_dir/output_dir/output  
--parallelism 20
```



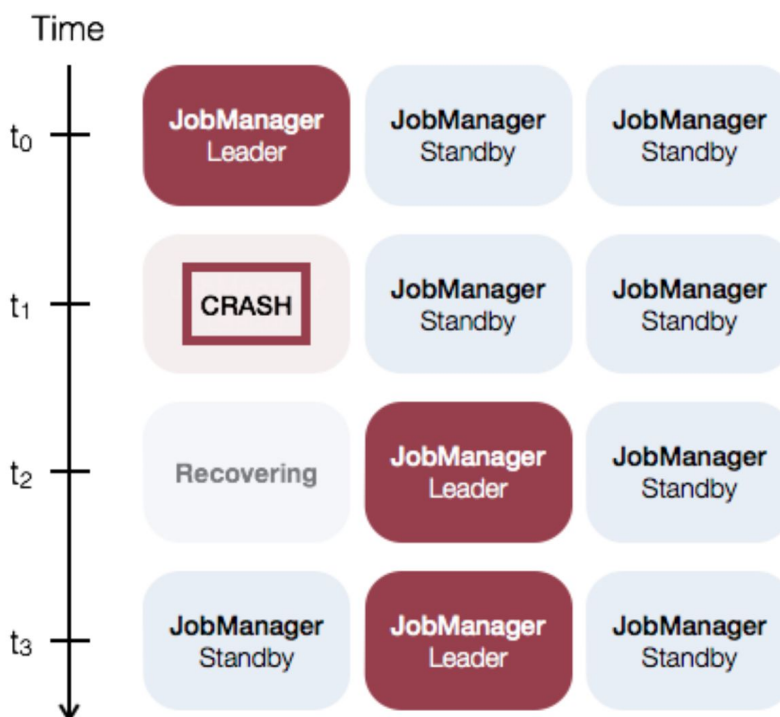


## 5. standalone模式的HighAvailability (HA) 部署和配置

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	<a href="#">Download (asc, sha512)</a>	<a href="#">Download (asc, sha512)</a>
Apache Flink 1.7.2 with Hadoop® 2.8	<a href="#">Download (asc, sha512)</a>	<a href="#">Download (asc, sha512)</a>
Apache Flink 1.7.2 with Hadoop® 2.7	<a href="#">Download (asc, sha512)</a>	<a href="#">Download (asc, sha512)</a>
Apache Flink 1.7.2 with Hadoop® 2.6	<a href="#">Download (asc, sha512)</a>	<a href="#">Download (asc, sha512)</a>
Apache Flink 1.7.2 with Hadoop® 2.4	<a href="#">Download (asc, sha512)</a>	<a href="#">Download (asc, sha512)</a>
Apache Flink 1.6.4 only	<a href="#">Download (asc, sha512)</a>	Not supported.
Flink 1.6.4 with Hadoop® 2.8	<a href="#">Download (asc, sha512)</a>	Not supported.
Flink 1.6.4 with Hadoop® 2.7	<a href="#">Download (asc, sha512)</a>	Not supported.
Flink 1.6.4 with Hadoop® 2.6	<a href="#">Download (asc, sha512)</a>	Not supported.
Flink 1.6.4 with Hadoop® 2.4	<a href="#">Download (asc, sha512)</a>	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 standalone session daemon on host z05f02320.sqa.zth.
Starting standalone session 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>

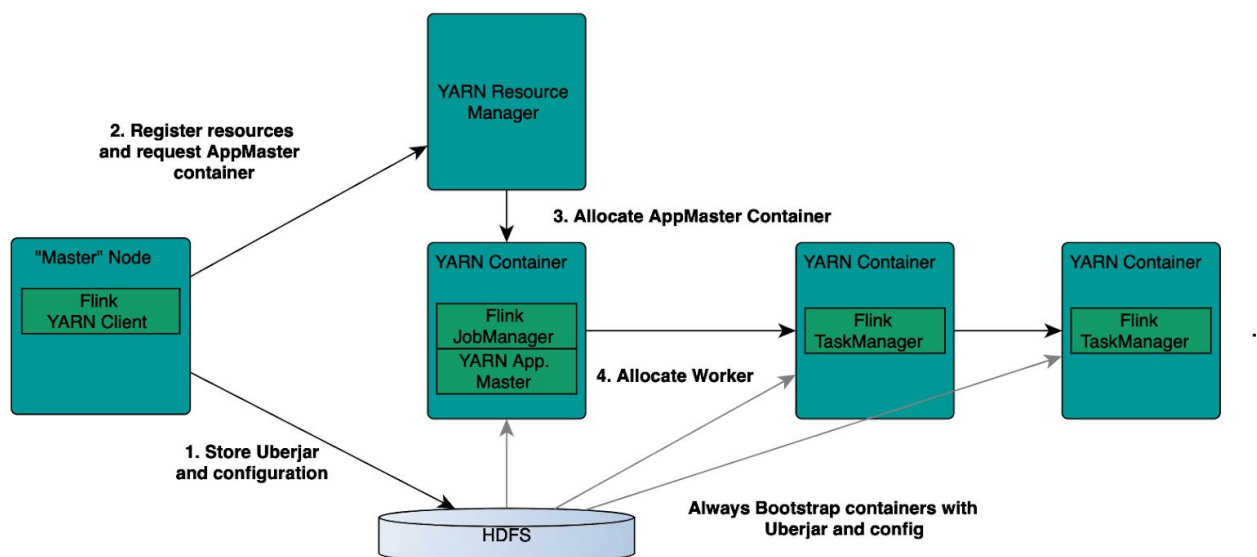
可以发现后一个链接已经不能展示了，而前一个链接可以展示，说明发生主备切换。

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/