链滴

JavaFX 官方文档翻译

作者：MingGH

原文链接：https://ld246.com/article/1619279958522

来源网站：链滴

官方文档地址：JavaFX
<h2 id="介绍">介绍</h2>
JavaFX 允许你创建具有现代、硬件加速的用户界面的 Java 应用程序，并且具有高度的可移植性
JavaFX 有详细的参考文档，这个简短的教程将告诉你如何编写一个 JavaFX 15 应用程序。
关于如何在移动平台上运行 JavaFX 应用程序的信息，请参见《Gluon Mobile 入门》
关于 JavaFX 11 的长期支持（LTS）的信息，请参见 JavaFX 长期支持选项。
JavaFX 建立在 JDK 之上，是一个独立的组件。开发 JavaFX 应用程序有两种不同的选择。

使用 JavaFX SDK（选择 11 LTS、最新版本 15.0.1 或早期访问构建）。
使用构建系统（如 maven/gradle），从 Maven 中心下载所需的模块（也可在上述版本中选择。

在任何情况下，对于这两个选项，都要求有最新版本的 JDK 15，或者至少是 JDK 11。
<h2 id="安装Java">安装 Java</h2>
跳过，记得安装 Java11 ，或者 Java15
<h2 id="使用JavaFX-SDK运行HelloWorld">使用 JavaFX SDK 运行 HelloWorld</h2>
如果你想使用 JavaFX SDK 而不是构建工具，请为你的操作系统下载一个合适的 JavaFX 运行时并将其解压到所需位置。在本教程中，我们将使用 JavaFX 15.0.1。
添加一个环境变量，指向运行时的 lib 目录。
windows 下
<pre><code class="highlight-chroma">set PATH_TO_FX="path\to\javafx-sdk-15.0.1\lib"
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">export PATH_TO_FX=path/to/javafx-sdk-15.0.1/lib
</code></pre>
现在你可以使用 JavaFX 运行时从命令行编译和运行 JavaFX 应用程序。
编译应用程序（例如，使用本示例中的 HelloFX.java），使用。
windows 下
<pre><code class="highlight-chroma">javac --module-path %PATH_TO_FX% --add-modules javafx.controls HelloFX.java
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">javac --module-path $PATH_TO_FX --add-modules javafx.controls HelloFX.java
</code></pre>
重要提示：请确保添加所需的模块，同时考虑到横向依赖关系的自动解决例如，没有必要添加 javafx.graphics 模块，因为它是 javafx.controls 模块的过渡性需求）。但是如你的应用程序使用 FXML，你将需要添加 javafx.fxml 模块，如下所示。
windows 下
<pre><code class="highlight-chroma">javac --module-path %PATH_TO_FX% --add-modules javafx.controls,javafx.fxml HelloFX.java
</code></pre>
linux 下
<pre><code class="highlight-chroma">javac --module-path $PATH_TO_FX --add-modules javafx.controls,javafx.fxml HelloFX.java
</code></pre>
最后，用以下方法运行该应用程序。
windows 下

<pre><code class="highlight-chroma">java --module-path %PATH_TO_FX% --add-modules
avafx.controls HelloFX
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">java --module-path $PATH_TO_FX --add-modules ja
afx.controls HelloFX
</code></pre>
<h2 id="使用Maven运行HelloWorld">使用 Maven 运行 HelloWorld</h2>
如果您想使用 Maven 开发 JavaFX 应用程序，您不必下载 JavaFX SDK。只需在 pom.xml 中指
你想要的模块和版本，构建系统就会下载所需的模块，包括你所在平台的本地库。
这里有一个 <a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com
2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FHelloFX%2FMaven%2Fhellofx%2Fpom.xml" tar
et="_blank" rel="nofollow ugc">pom.xml 文件，显示了如何实现这一目标，包括在这个<a h
ef="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsampl
s%2Fblob%2Fmaster%2FHelloFX%2FMaven" target="_blank" rel="nofollow ugc">例子
。
担心有的同学上不了 github,这里贴出来
pom 文件
<pre><code class="language-xml highlight-chroma"><projec
 xmlns="http://mav
n.apache.org/POM/4.0.0" xmlns:xsi=<span clas
="highlight-s">"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http:/
maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd"<span cla
s="highlight-nt">>
<modelVersion>4.0.0<span class="highlight-nt"
</modelVersion>
<groupId>org.openjfx<span class="highlight-nt"
</groupId>
<artifactId>hellofx&lt
/artifactId>
<packaging>jar</
ackaging>
<version>1.0-SNAPSHOT<span class="highlight-
t"></version>
<name>demo</n
me>
<url>http://maven.apache.org<span class="highl
ght-nt"></url>

<properties>

<project.build.sourceEncoding>
/span>UTF-8</project.build.sourceEncoding>
/span>

<javafx.version>16<span clas
="highlight-nt"></javafx.version>

<javafx.maven.plugin.version>
/span>0.0.6</javafx.maven.plugin.version>
/span>

</properties>

<dependencies>

<dependency>
<groupId>org.openjfx</groupId>
<artifactId>javafx-controls</artifactId>
<version>{javafx.version}</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.openjfx</groupId>
<artifactId>javafx-maven-plugin</artifactId>
<version>{javafx.maven.plugin.version}</version>
<configuration>
<mainClass>HelloFX</mainClass>
</configuration>
</plugin>
</plugins>
</build>
</project>
</code></pre>

另外，我们还创建了 JavaFX Maven Archetypes 来快速创建 Maven 项目。通过执行以下命令可以创建一个简单的 JavaFX 项目。
<pre><code class="highlight-chroma">mvn archetype:generate \
    -DarchetypeGroupId=org.openjfx \
    -DarchetypeArtifactId=javafx-archetype-simple \
    -DarchetypeVersion=0.0.3 \
    -DgroupId=org.openjfx \
    -DartifactId=sample \
    -Dversion=1.0.0 \
    -Djavafx-version=15.0.1
</code></pre>
该 Pom 使用了 JavaFX Maven 件。

<pre><code class="highlight-chroma"><plugins>
    <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.0.5</version>
        <configuration>
            <mainClass>HelloFX</mainClass>
        </configuration>
    </plugin>
</plugins>
</code></pre>
添加 maven 依赖
<pre><code class="highlight-chroma"><dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>15.0.1</version>
  </dependency>
</dependencies>
</code></pre>
重要提示：请注意过渡性的依赖关系是自动解决的（例如，不需要为 <code>javafx.graphics</ode> 模块添加依赖关系，因为它是由 javafx.controls 模块过渡解决的）。但是如果你的应用程序使 <code>FXML</code>，你将需要为 <code>javafx.fxml</code> 模块添加一个依赖关系。
最后，运行应用程序（例如，基于参考样本中的 HelloFX.java）。
<pre><code class="highlight-chroma">mvn clean javafx:run
</code></pre>
注意：请确保将 JAVA_HOME 环境变量设置为正确的 JDK 位置。
<h2 id="使用Gradle运行HelloWorld">使用 Gradle 运行 HelloWorld</h2>
与 Maven 类似，我们可以在 build.gradle 文件中声明需要的 JavaFX 模块。然而，对于 Gradl，我们需要应用 JavaFX gradle 插件。
<pre><code class="highlight-chroma">plugins {
  id 'application'
  id 'org.openjfx.javafxplugin' version '0.0.9'
}
</code></pre>
接下来，我们添加所需的模块。例如，如果我们只需要 javafx.controls 模块，我们将包括：

<pre><code class="highlight-chroma">javafx {
    version = "15.0.1"
    modules = ['javafx.controls']
}
</code></pre>
重要提示：请注意过渡性的依赖关系是自动解决的（例如，不需要为 <cod>javafx.graphics</code> 模块添加依赖关系，因为它是由 <code>javafx.controls</code> 模块渡解决的）。但是如果你的应用程序使用 <code>FXML</code>，你将需要同时添加 <code>javafx fxml</code> 模块。
你可以指定一个不同的 JavaFX 版本。例如，如果你想坚持使用 JavaFX 11.0.2。
<pre><code class="highlight-chroma">javafx {
    version = "11.0.2"
    modules = ['javafx.controls']
}
</code></pre>
这里有一个 <a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FHelloFX%2FGradle%2Fhellofx%2Fbuild.gradle"

arget="_blank" rel="nofollow ugc">build.gradle 文件，显示了如何实现这一目标，它取自个 sample.。
运行应用程序（例如，使用给定样本中的 HelloF.java），使用：
windows 下
<pre><code class="highlight-chroma">gradlew run
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">./gradlew run
</code></pre>
注意：我们对每个 JDK 的最小 Gradle 版本的建议如下。
<table>
<thead>
<tr>
<th>JDK version</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradle Version</td>
<td>5.0</td>
<td>5.0</td>
<td>6.0</td>
<td>6.3</td>
<td>6.3</td>
</tr>
</tbody>
</table>
<h2 id="Runtime-images">Runtime images</h2>
JavaFX 项目的运行时映像是一个自定义的 JRE，它只包含你的应用程序所需的平台模块。
如果你想为你的 JavaFX 项目创建一个运行时镜像，请遵循以下说明。
为你的操作系统下载一个合适的 JavaFX rntime and JavaFX jmods，并将其解压到一个想要的位置。
添加这些环境变量，指向运行时的 lib 目录和 jmods 目录。
windows 下
<pre><code class="highlight-chroma">set PATH_TO_FX="path\to\javafx-sdk-15.0.1\lib"
set PATH_TO_FX_MODS="path\to\javafx-jmods-15.0.1"
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">export PATH_TO_FX=path/to/javafx-sdk-15.0.1/lib
export PATH_TO_FX_MODS=path/to/javafx-jmods-15.0.1
</code></pre>
<h3 id="命令行">命令行</h3>
你可以从命令行中运行或创建你的 JavaFX 项目的 Runtime image。在这个 <a href="https://li

k.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FCommandLine%2FModular%2FCLI" target="_blank" rel="nofollow ugc">project</> 中可以找到一个简单的例子。
该应用程序有一个单一的 HelloFX.java 类，其 module-info 文件定义了 <code>hellofx</code> 模块。编译时可以使用 JavaFX SDK。
windows 下：
<pre><code class="highlight-chroma">dir /s /b src*.java > sources.txt & javac --module-path %PATH_TO_FX% -d mods/hellofx @sources.txt & del sources.txt
</code></pre>
linux 下
<pre><code class="highlight-chroma">javac --module-path $PATH_TO_FX -d mods/hellofx (find src/ -name "*.java")
</code></pre>
或使用 JavaFX 的 jmods。
windows 下
<pre><code class="highlight-chroma">dir /s /b src*.java > sources.txt & javac --module-path %PATH_TO_FX_MODS% -d mods/hellofx @sources.txt & del sources.txt
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">javac --module-path $PATH_TO_FX_MODS -d mods/ellofx $(find src/ -name "*.java")
</code></pre>
注意：注意，为了方便，输出被放在 mods/hellofx 下，但它可以使用任其他名称。
要用 java 命令运行模块化的应用程序，你需要 JavaFX SDK 模块和你的模块添加到模块路径中
windows 下
<pre><code class="highlight-chroma">java --module-path "%PATH_TO_FX%;mods" -m hellox/hellofx.HelloFX
</code></pre>
linux 下
<pre><code class="highlight-chroma">java --module-path $PATH_TO_FX:mods -m hellofx/ellofx.HelloFX
</code></pre>
<h3 id="jlink">jlink</h3>
通过一个模块化的项目，jlink 可以被用来创建一个使用 JavaFX jmods 的自定义 Runtime imags
windows 下
<pre><code class="highlight-chroma">jlink --module-path "%PATH_TO_FX_MODS%;mods" -add-modules hellofx --output hellofx
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">$JAVA_HOME/bin/jlink --module-path $PATH_TO_FXMODS:mods --add-modules hellofx --output hellofx
</code></pre>
而在 image 建立后，你可以运行它。
windows 下
<pre><code class="highlight-chroma">hellofx\bin\java -m hellofx/hellofx.HelloFX
</code></pre>

linux/mac 下
<pre><code class="highlight-chroma">hellofx/bin/java -m hellofx/hellofx.HelloFX
</code></pre>
注意：这个定制的 JRE 是特定平台的。
<h3 id="Maven">Maven</h3>
您可以运行或创建您的 JavaFX Maven 项目的 runtime。在这个 project.中以找到一个简单的例子。
该应用程序有一个主类 MainApp.java，其 module-info 文件定义了 <code>hellofx</code> 模块。它还使用了 FXML，需要在 pom.xml 中添加 <code>javafx.fxml</code> 依赖项。
它可以在命令行上用 <code>javac</code> 编译和运行，但使用 Maven 我们可以简单地做到：
<pre><code class="highlight-chroma">mvn clean javafx:run
</code></pre>
要创建一个自定义 runtime，使用 JavaFX Maven 插件，你可以这样做：
<pre><code class="highlight-chroma">mvn clean javafx:jlink
</code></pre>
注意该插件允许像 <code>jlink</code> 命令一样的常规选项，以及创建一个启动器或带有自定义 image 的压缩包。
而在 image 建立后，你可以从命令行中运行它。
windows 下
<pre><code class="highlight-chroma">target\hellofx\bin\launcher
</code></pre>
linux 下
<pre><code class="highlight-chroma">target/hellofx/bin/launcher
</code></pre>
<h3 id="Gradle">Gradle</h3>
你可以运行或创建一个你的 JavaFX Gradle 项目的 Runtime。在这个 project 中可以找到一个简单的例子。
这个应用程序有一个主类 HelloFX.java，它的 module-info 文件定义了 <code>hellofx</code> 模块，还有必要的 build.gradle 文件。
它可以在命令行上用 <code>javac</code> 编译和运行，但使用 Gradle 我们可以简单地做。<

p>
windows 下
<pre><code class="highlight-chroma">gradlew run
</code></pre>
linux/mac 下
<pre><code class="highlight-chroma">./gradlew run
</code></pre>
要创建一个自定义的运行时，可以使用上述步来生成一个 gradle 任务。另外，有一个插件可以为我们做这件事：org.beryx.jlink。它可以和 JavaFX 的 Gradle 插件相合：
<pre><code class="highlight-chroma">plugins {
  id 'application'
  id 'org.openjfx.javafxplugin' version '0.0.9'
  id 'org.beryx.jlink' version '2.23.1'
}

javafx {
version = "15.0.1"
modules = ['javafx.controls']
}

jlink {
launcher {
name = 'hellofx'
}
}
</code></pre>

来生成和运行自定义 image
windows
<pre><code class="highlight-chroma">gradlew jlink
build\image\bin\hellofx
</code></pre>
linux/mac
<pre><code class="highlight-chroma">./gradlew jlink
build/image/bin/hellofx
</code></pre>
<h3 id="自定义JDK-JavaFX-image">自定义 JDK+JavaFX image</h3>
你可以使用 <code>jlink</code> 来创建一个包括部分或全部 JavaFX 模块的 runtime image 而不附在某个项目上。
有了这个 image，你就可以运行 JavaFX 模块化或非模块化项目，把它作为 IDE 中的新 JRE 来建常规的 JavaFX 项目，甚至可以使用它的 <code>jlink</code> 命令来为你的项目创建一个新的定义 image。
例如，你可以创建一个包含 <code>java.se</code> 模块和 JavaFX 块的镜像，通过运行。

windows
<pre><code class="highlight-chroma">set PATH_TO_FX_MODS="path\to\javafx-jmods-15.0.
"
%JAVA_HOME%\bin\jlink --module-path %PATH_TO_FX_MODS% \
    --add-modules java.se,javafx.fxml,javafx.web,javafx.media,javafx.swing \
    --bind-services --output \path\to\jdkfx-15.0.1
</code></pre>
linux/mac
<pre><code class="highlight-chroma">export PATH_TO_FX_MODS=path/to/javafx-jmods-15
0.1
$JAVA_HOME/bin/jlink --module-path $PATH_TO_FX_MODS \
    --add-modules java.se,javafx.fxml,javafx.web,javafx.media,javafx.swing \
    --bind-services --output /path/to/jdkfx-15.0.1.jdk
</code></pre>
<h3 id="使用自定义的image">使用自定义的 image</h3>
通过这个自定义 image，你可以设置一个新的 JAVA_HOME。
windows
<pre><code class="highlight-chroma">set JAVA_HOME="path\to\jdkfx-15.0.1"
</code></pre>
linux/mac
<pre><code class="highlight-chroma">export JAVA_HOME=/path/to/jdkfx-15.0.1.jdk
</code></pre>
由于 JavaFX 模块已经是运行时的一部分，现在你可以运行 HelloFX <a href="https://link.ld246
com/forward?goto=https%3A%2F%2Fopenjfx.cn%2Fopenjfx-docs%2F" target="_blank" rel="
ofollow ugc">例子而不需要再添加 <code>PATH_TO_FX</code>，比如：
windows
<pre><code class="highlight-chroma">dir /s /b src*.java > sources.txt & javac -d m
ds/hellofx @sources.txt & del sources.txt
java --module-path mods -m hellofx/hellofx.HelloFX
</code></pre>
linux/mac
<pre><code class="highlight-chroma">javac -d mods/hellofx $(find src/ -name "*.java")
java --module-path mods -m hellofx/hellofx.HelloFX
</code></pre>
或将 image 添加到你的 IDE 中。
<img src="https://ld246.com/images/img-loading.svg" alt="Adding JDK+FX platform" d
ta-src="https://b3logfile.com/file/2021/04/solo-fetchupload-5086702051251022129-d7d91f8
.png?imageView2/2/interlace/1/format/jpg">
并开始用 JavaFX 类创建 Java 项目。
<img src="https://ld246.com/images/img-loading.svg" alt="Create JavaFX project" data-
rc="https://b3logfile.com/file/2021/04/solo-fetchupload-16818116661026459 27-4b5eb926.p
g?imageView2/2/interlace/1/format/jpg">
并在没有任何额外选项的情况下运行它们。
<img src="https://ld246.com/images/img-loading.svg" alt="Run JavaFX project" data-sr
="https://b3logfile.com/file/2021/04/solo-fetchupload-2921463436245105153-88c544e7.png
imageView2/2/interlace/1/format/jpg">
注意：这个自定义的 JRE 是针对特定平台的，它不打算用于分发，如果有
个新的 JDK 或新的 JavaFX SDK，它必须重新创建。不过，它的 jlink 工具可以用来创建一个带有项
的自定义镜像，可以进行分发。而 <a href="https://link.ld246.com/forward?goto=https%3A%2
%2Fdocs.oracle.com%2Fen%2Fjava%2Fjavase%2F15%2Fdocs%2Fspecs%2Fman%2Fjpackage.
tml" target="_blank" rel="nofollow ugc">jpackage tool 可用于分发带有 jlinked 项目的安
程序。
<h3 id="非模块化应用">非模块化应用</h3>
从 Java 9 开始，应用程序应该是模块化的，并且用 <a href="https://link.ld246.com/forward?

oto=https%3A%2F%2Fopenjfx.cn%2Fopenjfx-docs%2Findex.html%23modular" target="_blan
" rel="nofollow ugc">jlink 这样的工具来分发。然而，如果你有一个非模块化的 JavaFX 项目
或者你不能使用 <code>jlink</code>，因为你有非模块化的依赖关系，而自动模块命名惯例并不适
，你仍然可以创建一个 fat jar。
正如<a href="https://link.ld246.com/forward?goto=http%3A%2F%2Fmail.openjdk.java.ne
%2Fpipermail%2Fopenjfx-dev%2F2018-June%2F021977.html" target="_blank" rel="nofollow
gc">这里所解释的，为了创建一个包含所有必要的 JavaFX 依赖项的可运行 jar，你需要使用一
不从 <code>Application</code> 扩展的启动器类。
<blockquote>
这个问题的原因

</blockquote>
<h4 id="Maven-">Maven</h4>
如果你使用 Maven 开发 JavaFX 应用程序，你就不必下载 JavaFX SDK 了。只需在 pom.xml
件中指定你想要的模块和版本，构建系统就会下载所需的模块，包括你所在平台的本地库。
在这个 <a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2F
penjfx%2Fsamples%2Fblob%2Fmaster%2FCommandLine%2FNon-modular%2FMaven" targe
="_blank" rel="nofollow ugc">project 中可以找到一个简单的 Maven 例子。<a href="https
//link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples%2Fblo
%2Fmaster%2FCommandLine%2FNon-modular%2FMaven%2Fhellofx%2Fpom.xml" target="_
lank" rel="nofollow ugc">pom.xml 文件展示了如何用 Maven shade 插件为这个 <code>hel
ofx</code> 项目创建可运行的 fat jar。
运行该插件以创建 fat jar。
<pre><code class="highlight-chroma">mvn compile package
</code></pre>
运行该应用程序，使用:
windows
<pre><code class="highlight-chroma">java -jar shade\hellofx.jar
</code></pre>
linux/mac
<pre><code class="highlight-chroma">java -jar shade/hellofx.jar
</code></pre>
<h4 id="跨平台jar">跨平台 jar</h4>
你可以通过在你的 pom 文件中加入需要本地库的三个平台的依赖项来创建一个跨平台的 fat jar
在这个例子中只有 <code>javafx.graphics</code> 中的依赖项：
<pre><code class="highlight-chroma"><dependencies>
    ...
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>15.0.1</version>
        <classifier>win</classifier>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>15.0.1</version>
        <classifier>linux</classifier>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-graphics</artifactId>
        <version>15.0.1</version>
        <classifier>mac</classifier>

```
        &lt;/dependency&gt;
    &lt;/dependencies&gt;
</code></pre>
<h4 id="Gradle-">Gradle</h4>
<p>如果你使用 Gradle 开发你的 JavaFX 应用程序，你不必下载 JavaFX SDK。只要在 <code>build.gradle</code> 文件中指定你想要的模块和版本，构建系统就会下载所需的模块，包括你平台的本库。然而，对于 Gradle，我们需要找到并指定平台/操作系统作为分类器。</p>
<p>在这个 <a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FCommandLine%2FNon-modular%2FGradle" target="_blank" rel="nofollow ugc">project</a> 中可以找到一个简单的 Gradle 例子。为了用 Gradle 这个项目创建一个可运行的 fat jar，请修改 <a href="https://link.ld246.com/forward?goto=http%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FCommandLine%2FNon-modular%2FGradle%2Fhellofx%2Fbuild.gradle" target="_blank" rel="nofollow ugc">build.gradle</a> 文件中的 jar 任务，以包括启动器类:</p>
<pre><code class="highlight-chroma">mainClassName = 'hellofx.HelloFX'
jar {
    manifest {
        attributes 'Main-Class': 'hellofx.Launcher'
    }
    from {
        configurations.runtimeClasspath.collect { it.isDirectory() ? it : zipTree(it) }
    }
}
</code></pre>
<p>现在运行这个任务来创建 fat jar 并且运行它</p>
<p>windows</p>
<pre><code class="highlight-chroma">gradlew.bat jar

java -jar build\libs\hellofx.jar
</code></pre>

<p>linux/mac</p>
<pre><code class="highlight-chroma">./gradlew jar

java -jar build/libs/hellofx.jar
</code></pre>

<h4 id="跨平台jar-">跨平台 jar</h4>
<p>你可以通过在构建文件中添加需要本地库的三个平台的依赖项来创建一个跨平台的 fat jar，在这例子中只有 <code>javafx.graphics</code> 中的依赖项。</p>
<pre><code class="highlight-chroma">runtimeOnly "org.openjfx:javafx-graphics:$javafx.version:win"
runtimeOnly "org.openjfx:javafx-graphics:$javafx.version:linux"
runtimeOnly "org.openjfx:javafx-graphics:$javafx.version:mac"
</code></pre>
<p>现在再次运行 jar 任务以创建跨平台的 fat jar。</p>
<h4 id="命令行-">命令行</h4>
<p>最后，你也可以在命令行上为你的 JavaFX 项目创建一个可运行的 fat jar。</p>
<p><strong>警告:</strong> 这是一个令人沮丧的、乏味的、容易出错的手工过程，在 <code>jlink</code> 不适用的情况下，应该使用 Maven 的 shade 插件或 Gradle 的 jar 任务来避免。</p>
<p>在这个 <a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FCommandLine%2FNon-modular%2FCLI" target="_blank" rel="nofollow ugc">project</a>.中可以找到一个例子。为你的操作系统下载相应的 <a href
```

"https://link.ld246.com/forward?goto=https%3A%2F%2Fgluonhq.com%2Fproducts%2Fjavaf%2F" target="_blank" rel="nofollow ugc">JavaFX SDK，并将其解压到所需位置。
添加一个环境变量，指向运行时的 lib 目录
windows
<pre><code class="highlight-chroma">set PATH_TO_FX="path\to\javafx-sdk-15.0.1\lib"
</code></pre>
linux/mac
<pre><code class="highlight-chroma">export PATH_TO_FX=/path/to/javafx-sdk-15.0.1/lib
</code></pre>
编译这个项目
windows
<pre><code class="highlight-chroma">dir /s /b src\main\java*.java > sources.txt &
avac --module-path %PATH_TO_FX% --add-modules=javafx.controls -d out @sources.txt &a
p; del sources.txt
</code></pre>
linux
<pre><code class="highlight-chroma">javac --module-path $PATH_TO_FX --add-modules=j
avafx.controls -d out $(find src/main/java -name "*.java")
</code></pre>
并创建 fat jar，加入 JavaFX 需要的 jar 和特定平台的本地库。对于 hellofx 项目。
windows
<pre><code class="highlight-chroma">cd out & jar xf "%PATH_TO_FX%\javafx.base.jar"
& jar xf "%PATH_TO_FX%\javafx.graphics.jar" & jar xf "%PATH_TO_FX%\javafx.contr
ls.jar" & cd ..
copy "%PATH_TO_FX%\..\bin\prism*.dll" out & copy "%PATH_TO_FX%\..\bin\javafx*.dll"
ut & copy "%PATH_TO_FX%\..\bin\glass.dll" out && copy "%PATH_TO_FX%\..\b
n\decora_sse.dll" out
del out\META-INF\MANIFEST.MF & del out\module-info.class
mkdir libs
jar --create --file=libs/hellofx.jar --main-class=hellofx.Launcher -C out .
</code></pre>
linux
<pre><code class="highlight-chroma">find $PATH_TO_FX/{javafx.base.jar,javafx.graphics.jar,
avafx.controls.jar} -exec unzip -nq {} -d out \;

#uncomment for Linux:
#cp $PATH_TO_FX/{libprism*.so,libjavafx*.so,libglass*.so,libdecora_sse.so} out

#uncomment for Mac:
#cp $PATH_TO_FX/{libprism*.dylib,libjavafx*.dylib,libglass.dylib,libdecora_sse.dylib} out

rm out/META-INF/MANIFEST.MF out/module-info.class
mkdir libs
jar --create --file=libs/hellofx.jar --main-class=hellofx.Launcher -C out .
</code></pre>

而在 jar 建立后，你可以运行它。
<pre><code class="highlight-chroma">java -jar libs\hellofx.jar
</code></pre>
linux/mac
<pre><code class="highlight-chroma">java -jar libs/hellofx.jar
</code></pre>

注意：这个 JRE 是特定平台的。然而，如果每个平台的不同本地库都包含内，它可以很容易地成为一个多平台的 jar。这需要下载所有平台的 JavaFX SDK，提取所有必要的 ja（例如，javafx.graphics 在每个平台都是不同的），并复制所有的本地库，如上所示。正如上面提到，这个过程不应该手动完成。
<h2 id="JavaFX-and-IntelliJ-IDEA">JavaFX and IntelliJ IDEA</h2>
本节介绍如何在 IntelliJ IDEA 中创建一个 JavaFX 应用程序。在 IDE 的截图中使用了 JavaFX 15.0 1 和 IntelliJ IDEA 2020.1。
为你的操作系统下载一个合适的 JDK，并将 <code>JAVA_HOME </code> 设为 JDK 目录。多信息请参考 Install Java 部分。
你可以创建一个 JavaFX 模块化或非模块化项目，并使用 IDE 工具、Maven 或 Gradle 构建工。
<h3 id="非模块化应用-">非模块化应用</h3>
<h4 id="IDE">IDE</h4>
按照这些步骤创建一个 JavaFX 非模块化项目，并使用 IDE 工具来构建和运行它。或者，你也可从这里下载一个类似的项目。
为你的操作系统下载相应的 <a href="https://link.ld246.com/forward?goto=https%3A%2F%Fgluonhq.com%2Fproducts%2Fjavafx%2F" target="_blank" rel="nofollow ugc">JavaFX SDK<a>，并将其解压到所需位置，例如 <code>/Users/your-user/Downloads/javafx-sdk-15</code>。
<h5 id="1--创建JavaFX-project">1. 创建 JavaFX project</h5>
<img src="https://ld246.com/images/img-loading.svg" alt="Create a JavaFX project" dat-src="https://b3logfile.com/file/2021/04/solo-fetchupload-4841701116047135522-b720b688png?imageView2/2/interlace/1/format/jpg">
为项目提供一个名称，如 <code>HelloFX</code>，和一个位置。当项目打开时，JavaFX 类无法识别的。
<img src="https://ld246.com/images/img-loading.svg" alt="Missing JavaFX classes" datasrc="https://b3logfile.com/file/2021/04/solo-fetchupload-7058461564890044011-825050df.ng?imageView2/2/interlace/1/format/jpg">
<h5 id="2--Set-JDK-15">2. Set JDK 15</h5>
进入 <code>File -> Project Structure -> Project</code>，并将项目的 SDK 设置为 1。你也可以把语言级别设置为 11 或更高。
<img src="https://ld246.com/images/img-loading.svg" alt="Set JDK 11" data-src="https:/b3logfile.com/file/2021/04/solo-fetchupload-9001326708557096131-1d04a102.png?imageVew2/2/interlace/1/format/jpg">
<h5 id="3--Create-a-library">3. Create a library</h5>
进入 <code>File -> Project Structure -> Libraries</code>，将 JavaFX 15 SDK 作为个库添加到项目中。指向 JavaFX SDK 的 <code>lib</code> 文件夹。
<img src="https://ld246.com/images/img-loading.svg" alt="Create Library" data-src="hps://b3logfile.com/file/2021/04/solo-fetchupload-8748622722498065774-0e745f29.png?imaeView2/2/interlace/1/format/jpg">
一旦库被应用，JavaFX 类将被 IDE 识别。
<img src="https://ld246.com/images/img-loading.svg" alt="JavaFX classes are recognize" data-src="https://b3logfile.com/file/2021/04/solo-fetchupload-47194922350564164 74-d4ee1ba.png?imageView2/2/interlace/1/format/jpg">
警告:如果你现在运行该项目，它将被编译，但你会得到这个错误。
<blockquote>
<pre><code class="highlight-chroma">Caused by: java.lang.RuntimeException: Exception in Application start method
    at javafx.graphics/com.sun.javafx.application.LauncherImpl.launchApplication1(LauncherIml.java:900)</code></pre>

...
Caused by: java.lang.IllegalAccessError: class com.sun.javafx.fxml.FXMLLoaderHelper (in unnamed module @0xXXXXX) cannot access class com.sun.javafx.util.Utils (in module javafx.graphics because module javafx.graphics does not export com.sun.javafx.util to unnamed module @0xXXXXX
...
</code></pre>
</blockquote>
发生这个错误是因为 IntelliJ 只把 <code>javafx.base</code> 和 <code>javafx.graphics</code> 模块添加到模块路径中，但对于这个例子来说，还需要添加 <code>javafx.controls</code> 和 <code>javafx.fxml</code> 模块。
<h5 id="4--添加-VM-选项">4. 添加 VM 选项</h5>
要解决这个问题，请点击 "<code>Run -> Edit Configurations...</code> "并添加这些虚机选项。
windows
<pre><code class="highlight-chroma">--module-path "\path\to\javafx-sdk-15.0.1\lib" --add-modules javafx.controls,javafx.fxml
</code></pre>
linux/mac
<pre><code class="highlight-chroma">--module-path /path/to/javafx-sdk-15.0.1/lib --add-modules javafx.controls,javafx.fxml
</code></pre>
注意: IntelliJ 创建的默认项目使用 FXML，所以 <code>javafx.fxml</code> 和 <code>javafx.controls</code> 是必需的。如果你的项目使用其他模块，你也需要添加它们。
<img src="https://ld246.com/images/img-loading.svg" alt="VM options" data-src="https://b3logfile.com/file/2021/04/solo-fetchupload-5435927485935205531-704e6b6c.png?imageview2/2/interlace/1/format/jpg">
点击应用并关闭对话框。
另外，你也可以定义一个全局变量，可以在未来的项目中使用。进入 <code>Preferences (File > Settings) -> Appearance & Behavior -> Path Variables</code>，将变量的名定义为 <code>PATH_TO_FX</code>，并浏览到 JavaFX SDK 的 lib 文件夹来设置其值，然后点击用。
<img src="https://ld246.com/images/img-loading.svg" alt="Path Variable" data-src="https://b3logfile.com/file/2021/04/solo-fetchupload-4186708749476897974-954a4b47.png?imageView2/2/interlace/1/format/jpg">
然后你可以在设置虚拟机选项时参考这个全局变量，如：
<pre><code class="highlight-chroma">--module-path ${PATH_TO_FX} --add-modules javafx.controls,javafx.fxml
</code></pre>
<img src="https://ld246.com/images/img-loading.svg" alt="Path Variable in VM options" data-src="https://b3logfile.com/file/2021/04/solo-fetchupload-5597348085736815526-8fdd97b.png?imageView2/2/interlace/1/format/jpg">
<h5 id="5--运行项目">5. 运行项目</h5>
点击 <code>Run -> Run...</code> 来运行该项目，现在它应该可以正常工作了。
<h4 id="Maven--">Maven</h4>
按照这些步骤创建一个 JavaFX 非模块化项目，并使用 Maven 工具来构建和运行它。另外，你可以从这里下载一个类似的项目。
<h5 id="1--创建一个Maven项目">1. 创建一个 Maven 项目</h5>
选择 <code>File -> New -> Project -> Maven </code> 然后 选中 <code>Create from archetype</code>. 如果 JavaFX <a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fjavafx-maven-archetypes" target="_blank" rel="nofollow

gc">archetype 尚未安装，请选择 <code>add archetype</code>...并设置 groupId（<cod
>org.openjfx</code>）、artifactId（<code>javafx-maven-archetypes</code>）和版本（<st
ong>0.0.5），然后按下确定。
<img src="https://ld246.com/images/img-loading.svg" alt="JavaFX Archetype" data-src=
https://b3logfile.com/file/2021/04/solo-fetchupload-7735496218735457940-da0c3538.png?i
ageView2/2/interlace/1/format/jpg">
安装之后，选择这个 artifact,按 <code>Next</code>，并提供项目的名称，如 <code>Hello
X</code> 和项目的位置。也可以选择提供 groupId，比如 <code>org.openjfx</code> 和 <cod
>artifactId</code>，比如 <code>hellofx</code>
根据项目中是否使用 FXML，在 <code>javafx-archetype-fxml</code> 或 <code>javafx-ar
hetype-simple</code> 之间选择 artifactId 的原型。
<img src="https://ld246.com/images/img-loading.svg" alt="Select archetype" data-src=
https://b3logfile.com/file/2021/04/solo-fetchupload-1010939983269238972-4211e91e.png?
mageView2/2/interlace/1/format/jpg">
你也可以为 <code>javafx-version</code> 创建一个属性，并将其设置为 15.0.1。
<h5 id="2--验证项目">2. 验证项目</h5>
你可以在<a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2
openjfx%2Fsamples%2Fblob%2Fmaster%2FIDE%2FIntelliJ%2FModular%2FMaven%2Fhellofx
2Fpom.xml" target="_blank" rel="nofollow ugc">这里找到生成的 pom 文件。
确认它包括 <code>javafx.controls</code> 和 <code>javafx.fxml</code> 依赖项，并包括
<code>javafx-maven-plugin</code>
<img src="https://ld246.com/images/img-loading.svg" alt="Generated pom" data-src="
ttps://b3logfile.com/file/2021/04/solo-fetchupload-3238279849953155547-e3ca18c7.png?im
geView2/2/interlace/1/format/jpg">
打开 <code>module-info</code> 类，它包括了所需的模块 <code>javafx.controls</code>
和 <code>javafx.fxml</code>。由于 FXML 使用反射来访问模块中的控制器，它已经被打开到 <c
de>javafx.fxml</code>。
<img src="https://ld246.com/images/img-loading.svg" alt="module-info" data-src="htt
s://b3logfile.com/file/2021/04/solo-fetchupload-8530242654424605938-f2ed4b99.png?imag
View2/2/interlace/1/format/jpg">
<h5 id="3--运行项目">3. 运行项目</h5>
你可以打开 Maven 项目窗口，点击 <code>HelloFX -> Plugins -> compiler -> co
piler:compile</code> 来编译项目，点击 <code>HelloFX -> Plugins -> javafx -> javafx
run</code> 来执行该项目。
注意：如果 JAVA_HOME 没有设置为 11 或更高，从 Maven 项目窗口运行可能会失败。为避免
种情况，您可以给 Maven Runner 添加一个环境变量。
<img src="https://ld246.com/images/img-loading.svg" alt="JavaHome" data-src="https:
/b3logfile.com/file/2021/04/solo-fetchupload-8138267097508945154-c847179d.png?imageV
ew2/2/interlace/1/format/jpg">
或向 <code>javafx-maven-plugin</code> 设置正确的 <code>java</code> 命令。
<pre><code class="language-xml highlight-chroma"><config
ration>
    <executable>/path/to/jdk-15/bin/java<span clas
="highlight-nt"></executable>
</configuration>
</code></pre>
该项目也可以从终端运行。确保 <code>JAVA_HOME</code> 被设置为 15，然后运行 <cod
>mvn clean javafx:run</code>
<h5 id="4--创建一个自定义Runtime-image">4. 创建一个自定义 Runtime image</h5>
要创建一个自定义的运行时，使用 <a href="https://link.ld246.com/forward?goto=https%3
%2F%2Fgithub.com%2Fopenjfx%2Fjavafx-maven-plugin" target="_blank" rel="nofollow ugc"
JavaFX Maven plugin，你可以点击 <code>HelloFX -> Plugins -> javafx -> javafx:
link</code>，或者从终端将 <code>JAVA_HOME</code> 设置为 15 就可以运行。
<pre><code class="highlight-chroma">mvn clean javafx:jlink

</code></pre>
注意该插件允许像 <code>jlink</code> 命令一样的常规选项，以及创建一个启动器或带有自义 image 的压缩包。
而在 image 建立后，你可以从命令行中运行它。
windows
<pre><code class="highlight-chroma">target\hellofx\bin\launcher
</code></pre>
linux/mac
<pre><code class="highlight-chroma">target/hellofx/bin/launcher
</code></pre>
<h4 id="Gradle--">Gradle</h4>
按照这些步骤来创建一个 JavaFX 模块化项目，并使用 Gradle 工具来构建和运行它。或者，你可以从这里下载一个类似的项目。
<h5 id="1--创建一个-Gradle-项目">1. 创建一个 Gradle 项目</h5>
用 Java 创建一个 Gradle 项目。为项目提供一个名字，比如 HelloFX 和项目的位置。可以选择供 groupId，如 org.openjfx，artifactId，如 hellofx。当项目打开时，添加一个包 <code>org.opejfx</code> 和一个空的 <code>MainApp</code> 类。
<h5 id="2--修改构建">2. 修改构建</h5>
编辑 build.gradle 文件，用这个 build 文件替它，相应地将 mainClassName 设置为 org.openjfx.MainApp。
注意 org.openjfx.javafxplugin 插件的使用，它消除了添加 JavaFX 依赖项和为它们的编译和运任务设置模块路径的必要性。
<img src="https://ld246.com/images/img-loading.svg" alt="Update the build" data-src=https://b3logfile.com/file/2021/04/solo-fetchupload-187311930873386880-fce28ffe.png?imaeView2/2/interlace/1/format/jpg">
<h5 id="3--添加模块信息类">3. 添加模块信息类</h5>
添加 <code>module-info</code> 类，包括需要的模块 <code>javafx.controls</code> 和code>javafx.fxml</code>。因为 FXML 使用反射来访问模块中的控制器，所以必须将其开放给 <cde>javafx.fxml</code>。最后，导出包 <code>org.openjfx.com</code>。
<h5 id="4--添加源代码">4. 添加源代码</h5>
Based on this MainApp class, add its content to the project main class. Then add the <a href"https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples2Fblob%2Fmaster%2FIDE%2FIntelliJ%2FModular%2FGradle%2Fhellofx%2Fsrc%2Fmain%2Fjav%2Forg%2Fopenjfx%2FFXMLController.java" target="_blank" rel="nofollow ugc">controller<a> and the FXML and and the css files.
基于这个 MainApp 类，将其内容添加到项目主类中。然后添加 <a href="https://link.ld246.com/foward?goto=https%3A%2F%2Fgithub.com%2Fopenjfx%2Fsamples%2Fblob%2Fmaster%2FID%2FIntelliJ%2FModular%2FGradle%2Fhellofx%2Fsrc%2Fmain%2Fjava%2Forg%2Fopenjfx%2F

XMLController.java" target="_blank" rel="nofollow ugc">controller 和 FXML 和和 css 文件。
<img src="https://ld246.com/images/img-loading.svg" alt="source code" data-src="http://b3logfile.com/file/2021/04/solo-fetchupload-3485876056651419688-3168f2f8.png?imageView2/2/interlace/1/format/jpg">
<h5 id="5-运行这个项目">5.运行这个项目</h5>
你可以打开 Gradle 窗口，点击 <code>hellofx->Tasks->build->build</code> 来建项目，点击 <code>hellofx->Tasks->application->run</code> 来执行该项目。你也以打开终端并运行。
windows
<pre><code class="highlight-chroma">
gradlew run
</code></pre>
linux/mac
<pre><code class="highlight-chroma">./gradlew run
</code></pre>
<h5 id="6--创建一个自定义的runtime-image">6. 创建一个自定义的 runtime image</h5>
要创建一个 runtime image，你可以使用 <code>org.beryx.jlink</code> 插件。它可以很容地与 JavaFX Gradle 插件相结合：
<pre><code class="language-groovy highlight-chroma">
plugins {
  id 'application'
  id 'org.openjfx.javafxplugin version '0.0.9'
  id 'org.beryx.jlink' version '2.23.1'
}

javafx {
version = "15.0.1"
modules = ['javafx.controls', 'javafx.fxml']
}

jlink {
launcher {
name = 'hellofx'
}
}

</code></pre>

来生成自定义的 image。运行 <code>hellofx -> Tasks -> build -> jlink</code>
务来创建 image
要运行该 image，请在终端上输入
windows
<pre><code class="highlight-chroma">build\image\bin\hellofx
</code></pre>
linux/mac
<pre><code class="highlight-chroma">build/image/bin/hellofx
</code></pre>
<h2 id="JavaFX-和-NetBeans">JavaFX 和 NetBeans</h2>
改日再更
<h2 id="JavaFX-和-Eclipse">JavaFX 和 Eclipse</h2>
改日再更
<h2 id="接下来的步骤">接下来的步骤</h2>
恭喜你成功创建并运行了你的第一个 JavaFX 应用程序。
如果您想为 JavaFX 做出贡献，请访问我们的 <a href="https://link.ld246.com/forward?goto
https%3A%2F%2Fgithub.com%2Fopenjdk%2Fjfx%2F" target="_blank" rel="nofollow ugc">Gi
hub 资源库。
请通过我们的<a href="https://link.ld246.com/forward?goto=http%3A%2F%2Fgluonhq.c
m%2Fabout-us%2Fcontact-us%2F" target="_blank" rel="nofollow ugc">电子邮件支持与
们联系。我们很乐意听到您的意见!