



MyBox：简易工具箱

开发指南

作者：Mara

版本：2.0

2019-11-18

内容目录

1 关于 MyBox.....	4
1.1 项目资源.....	4
1.2 项目文档.....	5
2 源码.....	6
2.1 开发资源.....	6
2.1.1 需要单独下载的软件	6
2.1.2 通过 Maven 自动下载的依赖.....	6
2.1.3 直接引用的第三方代码.....	7
2.1.4 图标.....	7
2.2 利用 Maven 构建 jar 包.....	8
2.2.1 在 win 平台上构建.....	8
2.2.2 在 Linux 平台上构建.....	9
2.2.3 在 Mac 平台上构建.....	9
2.2.4 构建跨平台的 jar 包.....	9
2.2.5 构建不含 javafx 的 jar 包.....	9
2.3 制作自包含包的脚本.....	10
2.3.1 工具: jpackage.....	10
2.3.2 制作 exe 包.....	10
2.3.3 制作 linux 可执行包.....	10
2.3.4 制作 mac 可执行包.....	10
3 Netbeans 编程.....	12
3.1 配置 Derby 数据库.....	12
3.1.1 添加 Derby 驱动器.....	12
3.1.2 以嵌入模式启动/连接 Derby 数据库.....	13
3.1.3 以客户端模式连接数据库.....	15
3.2 项目管理.....	16
3.2.1 切换配置文件.....	16
3.2.2 快速调试.....	16
3.2.3 构建包含所有依赖的 jar 包.....	16
3.3 查找和下载第三方源码.....	17
3.4 异常处理.....	18
3.4.1 模块化后再去模块化, Netbeans 异常.....	18
3.4.2 系统突然关机后, Netbeans 异常.....	18
3.4.3 双击 fxml 打开编辑代码界面而不是 JavaFX Scene Builder.....	19
4 JavaFx 编程.....	20
4.1 官方指南.....	20
4.2 最新版 JavaFx 的配置.....	20
4.3 利用 CSS 制作界面风格.....	20
4.4 开启/关闭 hidpi.....	20
4.5 控件截图的分辨率.....	20
4.6 动态设置 Fxml 的 Controller.....	23
4.7 动态修改 chart 的 legend.....	23

4.8 “setAlwaysOnTop”带来的危险.....	23
4.9 TableView/ListView 的“幽灵数据”	24
4.10 内嵌页面监听事件.....	24
4.11 强制 ListView 刷新.....	25
4.12 一句话事项.....	25
5 图像处理.....	26
5.1 Java 的图像处理技术.....	26
5.2 图像的元数据.....	26
5.3 多帧图像文件.....	26
5.4 Tiff 文件格式.....	26
5.5 动画 Gif.....	26
5.6 大图像.....	27
5.7 图像采样.....	27
5.8 图像的灰度.....	28
5.9 颜色距离.....	28
5.10 如何把图像转变成褐色.....	28
5.11 图像的混合模式.....	28
5.12 卷积.....	28
5.13 “漫灌”算法.....	28
5.14 图像尺寸.....	28
5.15 图像量化.....	29
5.16 滤镜.....	29
6 Java 编程.....	30
6.1 数据精度.....	30
6.2 字符集和编码.....	30
6.3 问题处理.....	30
6.3.1 “cannot access class ... because module ... does not export ...”	30
6.3.2 “某些输入文件使用了未经检查或不安全的操作”	31
6.4 一句话事项.....	32

1 关于 MyBox

1.1 项目资源

这是利用 JavaFx 开发的图形化界面程序，目标是提供简单易用的功能，免费开源。

项目主页：

<https://github.com/Mararsh/MyBox>

每个版本的源代码、编译好的包、和文档都在 Release 目录下：

<https://github.com/Mararsh/MyBox/releases>

欢迎在线提交软件需求和问题报告：

<https://github.com/Mararsh/MyBox/issues>

云盘地址：

https://pan.baidu.com/s/1fWMRzym_jh075OCX0D8y8A#list/path=%2F

Set of Easy Tools, including PDF tools, Image Tools, File Tools, Network Tools.

javafx-desktop-apps pdf-converter pdfbox jai-imageio image-viewer Manage topics

82 commits 1 branch 26 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Description	Commit	Last Commit
MyBox	v3.0	4f7d685	8 days ago
docs	u		8 days ago
.gitignore	u		3 months ago
LICENSE	Initial commit		4 months ago
README.md	v3.0		8 days ago

User Guide in English

MyBox: 简易工具集

1.2 项目文档

本文档介绍作者开发 MyBox 的一些总结，下载地址：

<https://github.com/Mararsh/MyBox/releases/download/v5.8/MyBox-DevGuide-2.0-zh.pdf>

MyBox 的用户手册包括：

《MyBox 用户手册-概述》

<https://github.com/Mararsh/MyBox/releases/download/v5.0/MyBox-UserGuide-5.0-Overview-zh.pdf>

《MyBox 用户手册-PDF 工具》

<https://github.com/Mararsh/MyBox/releases/download/v5.0/MyBox-UserGuide-5.0-PdfTools-zh.pdf>

《MyBox 用户手册-图像工具》

<https://github.com/Mararsh/MyBox/releases/download/v5.0/MyBox-UserGuide-5.0-ImageTools-zh.pdf>

《MyBox 用户手册-桌面工具》

<https://github.com/Mararsh/MyBox/releases/download/v5.0/MyBox-UserGuide-5.0-DesktopTools-zh.pdf>

《MyBox 用户手册-网络工具》

<https://github.com/Mararsh/MyBox/releases/download/v5.0/MyBox-UserGuide-5.0-NetworkTools-zh.pdf>

2 源码

2.1 开发资源

MyBox 的开发和运行只依赖开源软件。

目前版本的 MyBox 基于 Java 13 + openjfx 13 + derby 15 + NetBean 11。

2.1.1 需要单独下载的软件

开发 MyBox 需要下载以下软件：

Java 13.0.1 或更高版本(Oracle JDK 或 openJDK)

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

<http://openjdk.java.net/>

NetBean 11.0 或更高版本

<https://netbeans.org/>

JavaFX Scene Builder 2.0

<https://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>

如果需要制作自包含程序包，则还需要下载 jpackage（这是未发布的 jdk14 的一部分）：

<https://jdk.java.net/jpackage/>

2.1.2 通过 Maven 自动下载的依赖

MyBox 还依赖以下软件，不过开发者不必单独下载它们，而是通过 Maven 构造 MyBox 的源码来自动下载这些依赖：

JavaFx （从 Java 11 以后，Java 不再包含 JavaFx，因此 JavaFx 是作为依赖加入项目）

<https://gluonhq.com/products/javafx/>

<https://openjfx.io/>

Derby:

<http://db.apache.org/derby/>

jai-imageio

<https://github.com/jai-imageio/jai-imageio-core>

PdfBox

<https://pdfbox.apache.org/>

Pdf2dom

<http://cssbox.sourceforge.net/pdf2dom/>

javazoom

<http://www.javazoom.net/index.shtml>

log4j

<https://logging.apache.org/log4j/2.x/>

tess4j

<https://github.com/nguyenq/tess4j>

tesseract

<https://github.com/tesseract-ocr/tesseract>

barcode4j

<http://barcode4j.sourceforge.net>

zxing

<https://github.com/zxing/zxing>

flexmark-java

<https://github.com/vsch/flexmark-java>

commons-compress

<http://commons.apache.org/proper/commons-compress/index.html>

2.1.3 直接引用的第三方代码

以下代码被复制为单独的 class 文件，在目录“MyBox\src\main\java\thridparty\”下：

GifDecoder:

<https://github.com/DhyanB/Open-Imaging>

EncodingDetect:

<https://www.cnblogs.com/ChurchYim/p/8427373.html>

2.1.4 图标

以下网站的资源对开源项目免费：

<https://icons8.com/icons/set/home>

MyBox 包含五套不同颜色的图标：浅蓝色图标直接取自以上网站，其它颜色的图标由文件“MainApp.java”的方法“makeIcons()”自动生成。

图标主色	获得方式	颜色值
浅蓝	网站下载	#4788c7 #dff0fe
红色	浅蓝图标色相减少 215	#c94d58 #feb6be
粉红	浅蓝图标红色通道增加 151	#de88c7 #ffd9e6
蓝色	浅蓝图标饱和度增加 50%	#0066cc #92cbfe
橘色	浅蓝图标色相增加 171	#c8754b #fecdb6

2.2 利用 Maven 构建 jar 包

项目文件“pom.xml”针对不同的平台定义了不同的 profile，因此可以针对不同的平台编译打包。

2.2.1 在 win 平台上构建

在 MyBox 源码根目录下执行以下命令：(缺省 profile 是 win 平台)

`mvn clean`

`mvn package`

```
命令提示符

D:\MyBox>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----< mara:MyBox >-----
[INFO] Building MyBox 5.3
[INFO]           [ jar ]-
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ MyBox ---
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 0.258 s
[INFO] Finished at: 2019-08-08T10:35:24+08:00
[INFO]

D:\MyBox>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< mara:MyBox >-----
[INFO] Building MyBox 5.3
[INFO]           [ jar ]-
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ MyBox ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 753 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ MyBox ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 245 source files to D:\MyBox\target\classes
[INFO] /D:/MyBox/src/main/java/mara/mybox/value/CommonValues.java: 某些输入文件使用了未经检查或不安全的操作。
[INFO] /D:/MyBox/src/main/java/mara/mybox/value/CommonValues.java: 有关详细信息，请使用 -Xlint:unchecked 重新编译。
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ MyBox ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\MyBox\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ MyBox ---
[INFO] Changes detected - recompiling the module!
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ MyBox ---
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ MyBox ---
[INFO] Building jar: D:\MyBox\target\MyBox-5.3.jar
[INFO]
```

2.2.2 在 Linux 平台上构建

确保环境变量 JAVA_HOME 指向 jdk13.0.1，如 “/usr/java/openjdk-13.0.1”。

在 MyBox 源码根目录下执行以下命令：

```
mvn clean  
mvn -P linux package
```

2.2.3 在 Mac 平台上构建

确保环境变量 JAVA_HOME 指向 jdk13.0.1，如 “/Library/Java/JavaVirtualMachines/jdk-13.0.1.jdk/Contents/Home”。

在 MyBox 源码根目录下执行以下命令：

```
mvn clean  
mvn -P mac package
```

2.2.4 构建跨平台的 jar 包

可以在任意平台上构建跨平台的 jar 包，它包含了所有平台需要的资源，因此较大。

在 MyBox 源码根目录下执行以下命令：

```
mvn clean  
mvn -P cross-platform package
```

2.2.5 构建不含 javafx 的 jar 包

如果用户环境已安装 openjfx 12.0.1 以上版本，则可以制作不包含 javafx 的 jar 包提供给用户。这样做好处是：MyBox 的安装包较小，坏处是：要求用户安装 javafx 并且运行时需要额外配置引用 javafx 的参数。

好处小于坏处，所以目前没有制作这样的包。

2.3 制作自包含包的脚本

自包含包无需 java 环境即可运行。

2.3.1 工具: jpackage

Java 8 包含打包工具 “javapackager”，但是 Java 9 把它移除了。Open jdk 14 正在开发新的替代工具 jpackage，并且开放了下载：

<https://jdk.java.net/jpackage/>

实测 jpackage 很好用，不需要安装，可以放在任意目录下。

注意：jpackage 自身基于 jdk14(还未发布)，打包时可以用参数 “`--runtime-image`” 指定要打的包所用的 jdk 版本。

2.3.2 制作 exe 包

在源码的目录 “pack” 下有文件 “**make-win-exe.bat**”，双击这个文件可以自动打包。其中的目录名需要修改成开发者自己的环境。

其中的核心语句是：

```
D:\Programs\jdk-14\bin\jpackage --package-type app-image --app-version %version% --vendor Mara
--verbose --runtime-image D:\Programs\Java\openjdk-13.0.1 --dest D:\tmp\make-mybox\out --name
MyBox --input D:\tmp\make-mybox\src --main-jar MyBox-%version%.jar --icon D:\tmp\make-
mybox\res\MyBox.ico
```

生成的启动文件是 “D:\tmp\make-mybox\out\MyBox\MyBox.exe”。

2.3.3 制作 linux 可执行包

在源码的目录 “pack” 下有文件 “**make-linux-bin.sh**”，运行这个文件可以自动打包。其中的目录名需要修改成开发者自己的环境。

其中的核心语句是：

```
./jdk-14/bin/jpackage --package-type app-image --app-version $version --vendor Mara --verbose
--runtime-image /usr/java/openjdk-13.0.1 --dest out --name MyBox --input src --main-jar MyBox-
$version.jar --icon res/MyBox.png
```

生成的启动文件是 “out/MyBox/bin/MyBox”。

2.3.4 制作 mac 可执行包

在源码的目录 “pack” 下有文件 “**make-mac-dmg.sh**”，运行这个文件可以自动打包。其中的目录名需要修改成开发者自己的环境。

其中的核心语句是：

```
./jdk-14/contents/Home/bin/jpackage --package-type dmg --app-version $version --vendor Mara --verbose
--runtime-image /Library/Java/JavaVirtualMachines/jdk-13.0.1.jdk/Contents/Home --dest out --name
MyBox --input src --main-jar MyBox-$version.jar --icon res/MyBox.icns
```

生成的启动文件是 “out/MyBox-\$version.dmg”。

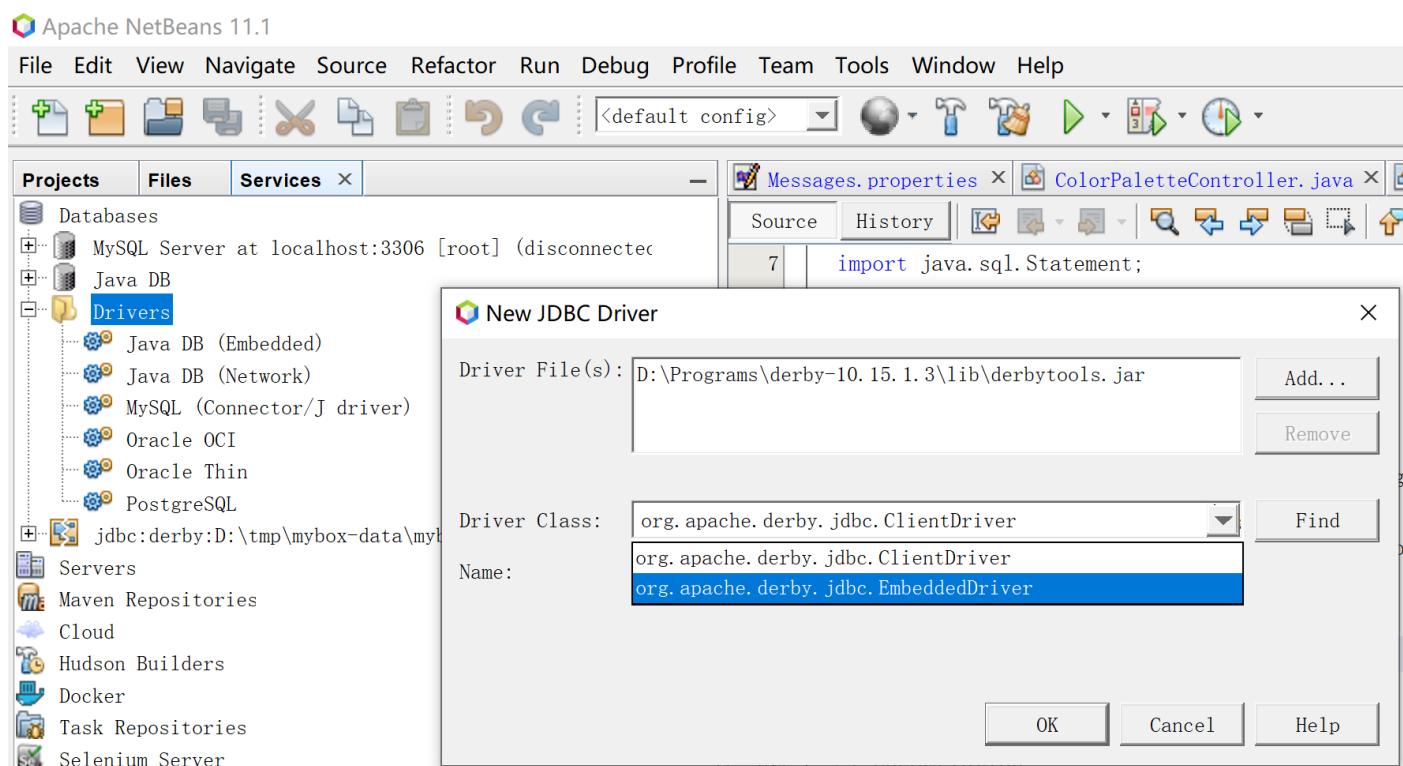
注意，在 mac 上打包需要制作 icns 文件，pack 目录下的文件 “**generate-mac-icns.sh**”可以自动制作 icns，其中的文件/目录名需要修改成开发者自己的环境。

3 Netbeans 编程

3.1 配置 Derby 数据库

3.1.1 添加 Derby 驱动器

在 Derby 的安装包中找到文件“Derbytools.jar”，用 NetBeans 的“Services”-“Databases”-“Drivers”添加 derby 的两个 jdbc 驱动：“Client Driver”和“Embedded Driver”

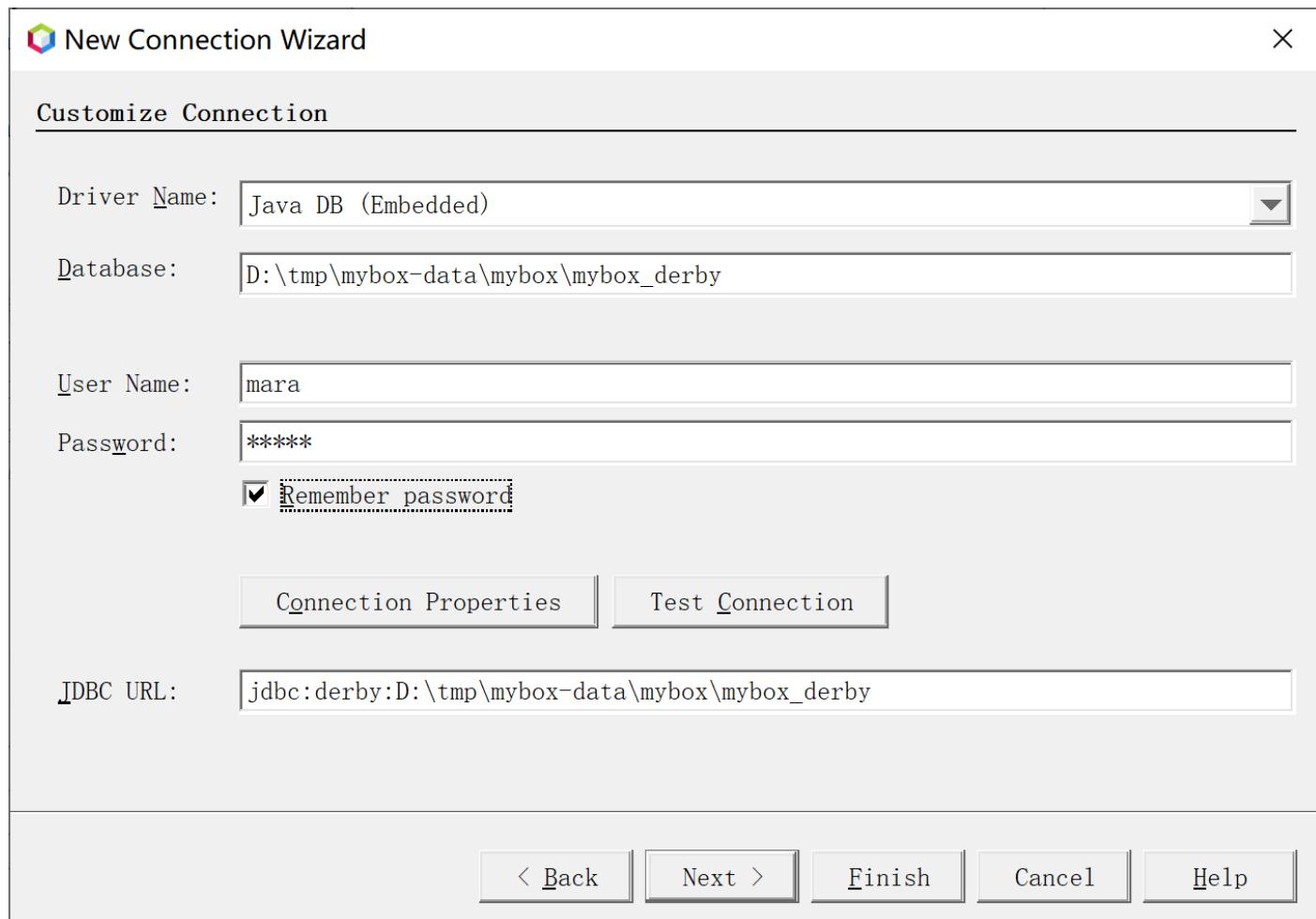


3.1.2 以嵌入模式启动/连接 Derby 数据库

注意：这种模式只能互斥地访问数据库，即同时只能有一个应用或 jvm 访问数据库。若已有其它应用或 jvm 连接到数据库，则 NetBeans 不能用这种模式访问数据库；而若 NetBeans 用这种模式连接了数据库，则其它应用或 jvm 就不能连接数据库。

当 MyBox 没有启动时，可以用这种方式启动数据库来检查数据。

在调试 MyBox 之前，应当断开嵌入模式的连接，否则 MyBox 无法连接数据库。



Apache NetBeans 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects Files Services ×

Databases MySQL Server at localhost:3306 [root] Java DB Drivers jdbc:derby:D:\tmp\mybox_derby [mara on MARA]

MARA Tables ALARM_CLOCK BROWSER_URLS CONVOLUTION_KERNEL FLOAT_MATRIX IMAGE_HISTORY IMAGE_SCOPE SR...

View Data... Execute Command... Add Column... Refresh Delete Delete Grab Structure... Recreate Table... Properties

Messages.properties × ImageManufactureBatchReplaceColor

Connection: jdbc:derby:D:\tmp\mybox_derby [mara on MARA]

```
1 SELECT * FROM MARA.SRGB FETCH FIRST 100 ROWS ONLY;
2
```

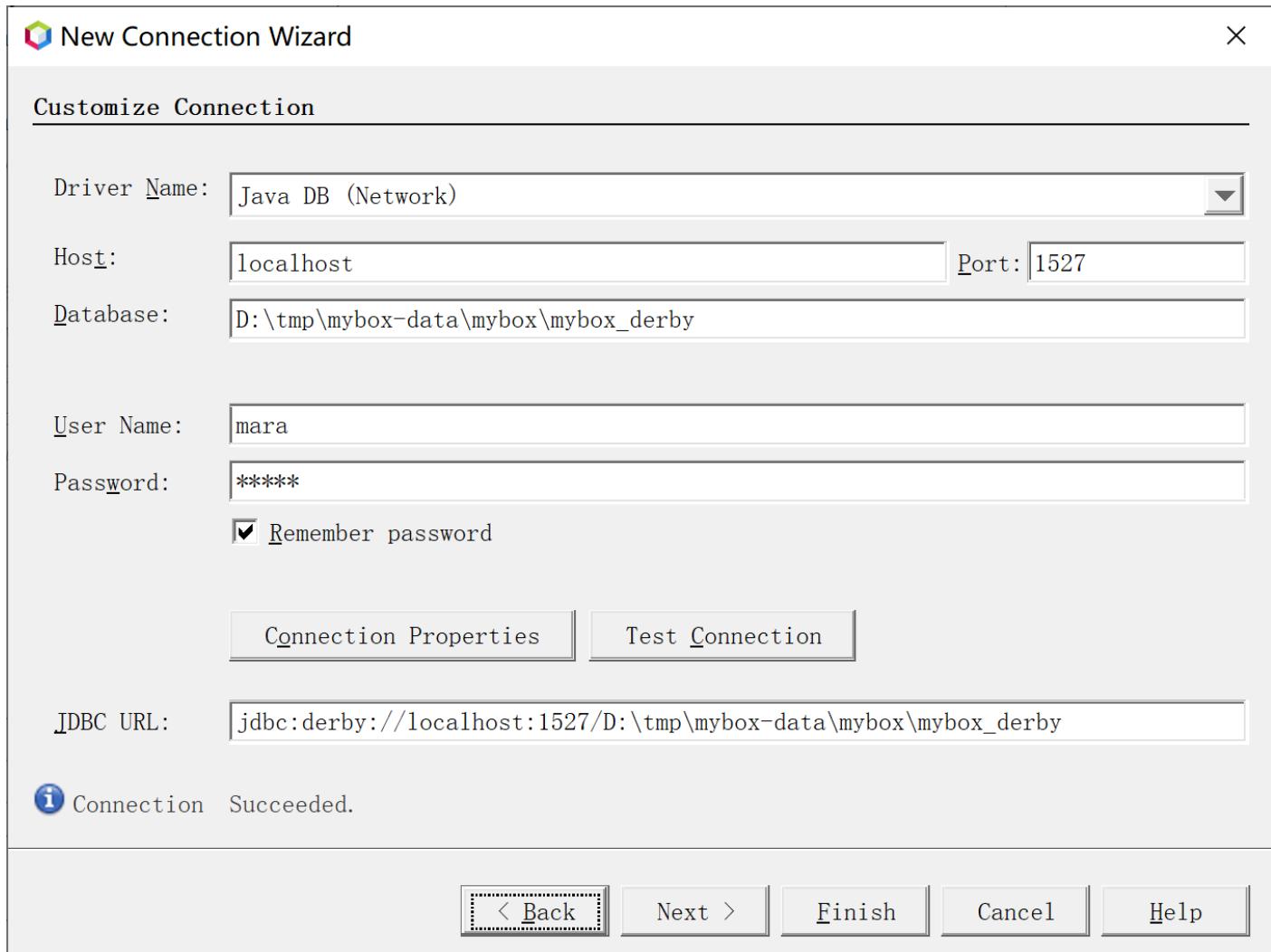
SELECT * FROM MARA.SRGB F...

#	COLOR_VALUE	COLOR_NAME
1	0xff0000ff	red
2	0x0000ffff	blue
3	0xffffffff	white
4	0xd2b48cff	tan
5	0xffff00ff	yellow
6	0x20b2aaff	lightseagreen
7	0x48d1ccff	mediumturquoise
8	0xf0ffff	azure
9	0xe6e6faaff	lavender
10	0x191970ff	midnightblue
11	0xffff0f5ff	lavenderblush
12	0xdb7093ff	palevioletred
13	0xdc143cff	crimson
14	0xffc0cbff	pink
15	0xffb6c1ff	lightpink
16	0xf0fff0ff	honeydew
17	0xfa8072ff	salmon
18	0xfffffe0ff	lightyellow
19	0xdab987ff	brownwood

3.1.3 以客户端模式连接数据库

当 Derby 数据库已以网络模式（服务器）被启动以后，可以用客户端模式连接到数据库。

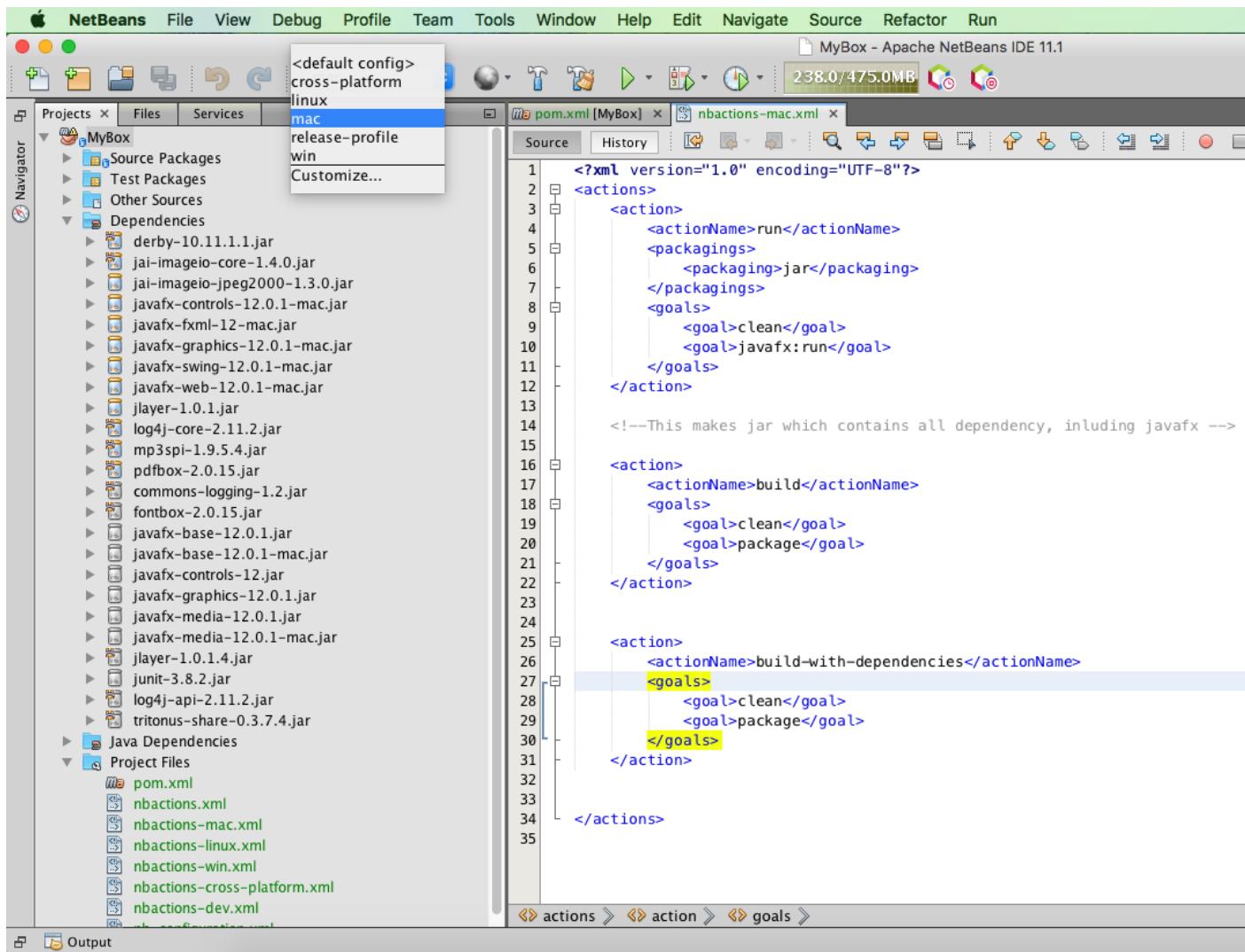
当 MyBox 已启动时，可以用这种方式连接数据库来检查数据。



3.2 项目管理

3.2.1 切换配置文件

项目文件“nbactions*.xml”是针对不同的 profile（不同平台）的 netbeans 配置文件，在 netbeans 中可以方便地切换配置文件以构建运行当前平台的程序：



3.2.2 快速调试

根据官网的指南，在“nbactions*.xml”中，“run”被定义为“javafx:run”，这样点击“run”按钮就是直接在 IDE 环境下运行 MyBox，响应速度很快，而不生成 jar 包。

3.2.3 构建包含所有依赖的 jar 包

所有“nbactions*.xml”都把“build”定义为“package”，因此对项目执行“build”时就针对当前 profile（平台）自动生成包含所有依赖的 jar 包，但是响应速度较慢。

3.3 查找和下载源码

通常在 Netbeans 中通过 Maven 可以找到并下载成功多数最新源码，但也有版本未更新或者 IDE 下载不顺利的时候，此时可以直接到以下网址查找下载：

<https://mvnrepository.com/>

The screenshot shows the mvnrepository.com website. At the top, there's a navigation bar with icons for back, forward, search, and home. The URL in the address bar is https://mvnrepository.com/artifact/org.openjfx/javafx-controls/13.0.1. Below the address bar, the MVNREPOSITORY logo is on the left, and a search bar on the right says "Search for groups, artifacts, categories".

The main content area has a title "JavaFX Controls » 13.0.1" with a subtitle "JavaFX Controls". To the left of the main content, there's a sidebar titled "Indexed Artifacts (15.5M)" which includes a line graph showing the number of projects indexed over time from 2006 to 2018, and a list of "Popular Categories" such as Aspect Oriented, Actor Frameworks, Application Metrics, etc.

The main content area contains the following details for the artifact:

License	GPL 2.0
Date	(Oct 18, 2019)
Files	jar (306 bytes) View All
Repositories	Central
Used By	151 artifacts

A note at the bottom of this section says "Note: There is a new version for this artifact" and a link to "New Version".

Below this, there's a section for dependency declarations with tabs for Maven, Gradle, SBT, Ivy, Grape, Leiningen, and Buildr. The Maven tab is selected, showing the following XML code:

```
<!-- https://mvnrepository.com/artifact/org.openjfx/javafx-controls -->
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>13.0.1</version>
</dependency>
```

At the bottom of this section is a checkbox labeled "Include comment with link to declaration".

点击“View All”即可进入 maven 库目录：



org/openjfx/javafx-controls/13.0.1

.. /			
javafx-controls-13.0.1-javadoc.jar	2019-10-18 18:43	10171195	
javafx-controls-13.0.1-javadoc.jar.asc	2019-10-18 18:43	475	
javafx-controls-13.0.1-javadoc.jar.asc.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-javadoc.jar.asc.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-javadoc.jar.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-javadoc.jar.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-linux.jar	2019-10-18 18:43	2508887	
javafx-controls-13.0.1-linux.jar.asc	2019-10-18 18:43	475	
javafx-controls-13.0.1-linux.jar.asc.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-linux.jar.asc.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-linux.jar.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-linux.jar.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-mac.jar	2019-10-18 18:43	2508838	
javafx-controls-13.0.1-mac.jar.asc	2019-10-18 18:43	475	
javafx-controls-13.0.1-mac.jar.asc.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-mac.jar.asc.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-mac.jar.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-mac.jar.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-sources.jar	2019-10-18 18:43	1420630	
javafx-controls-13.0.1-sources.jar.asc	2019-10-18 18:43	475	
javafx-controls-13.0.1-sources.jar.asc.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-sources.jar.asc.sha1	2019-10-18 18:43	40	
javafx-controls-13.0.1-sources.jar.md5	2019-10-18 18:43	32	
javafx-controls-13.0.1-sources.jar.sha1	2019-10-18 18:43	40	

3.4 异常处理

3.4.1 模块化后去模块化，Netbeans 异常

加入模块化描述文件 module-info.java 以后，如果去掉它，再用 Netbeans 打开项目，则所有源文件报错。

解决办法：删除 Netbeans 的缓存，如目录 “C:\Users\mara\AppData\Local\NetBeans\Cache\11.1”。

3.4.2 系统突然关机后，Netbeans 异常

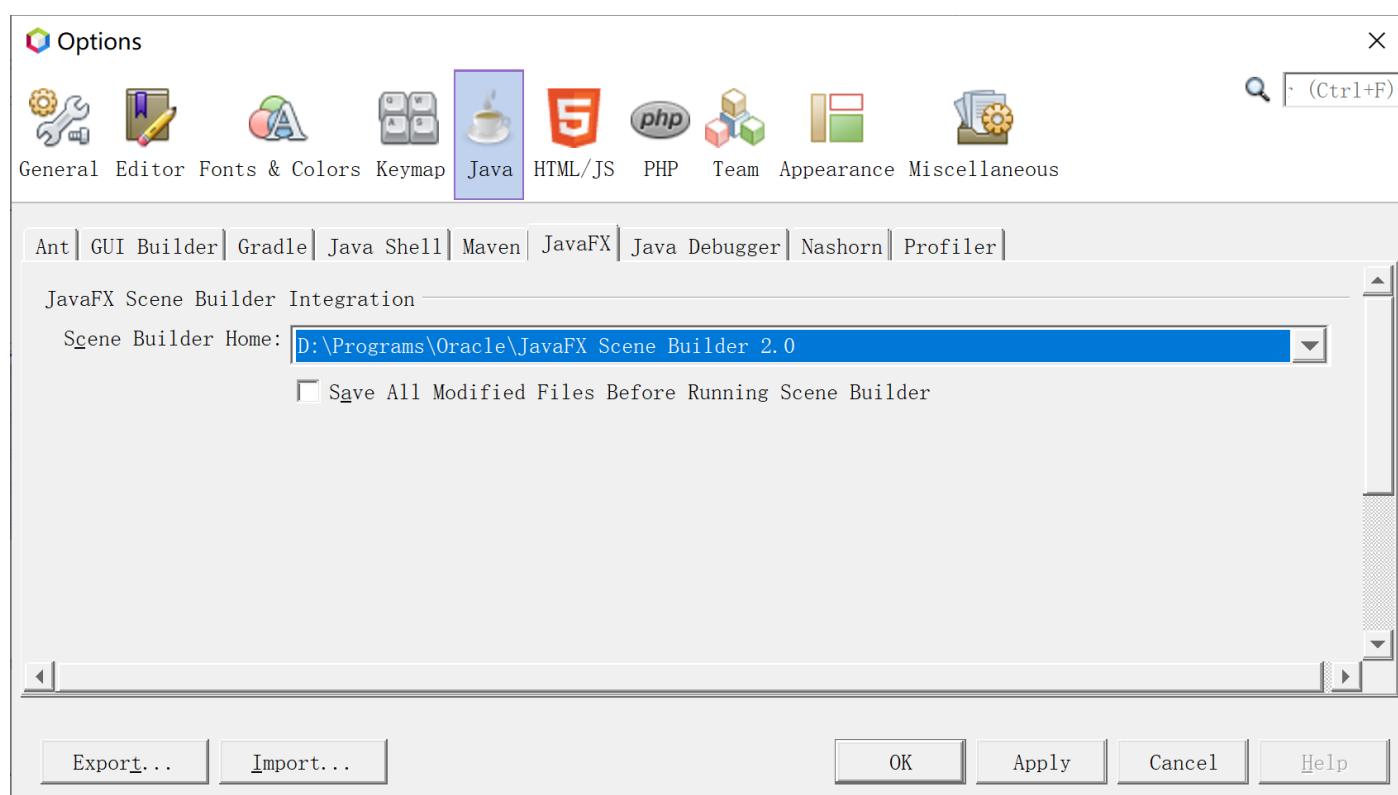
系统突然关机时 Netbeans 内存数据可能没有更新到系统缓存，Netbeans 可能表现异常：在我的环境中，左边的项目树再也显示不出来了。

解决办法：删除 Netbeans 的配置数据，如目录 “C:\Users\mara\AppData\Roaming\NetBeans\11.1”。

3.4.3 双击 fxml 打开编辑代码界面而不是 JavaFX Scene Builder

检查 “Tools”-“Options”-“Java”-“JavaFX”，确保路径正确。

有时退出 NetBeans 后再入，这个选项会失效，需要重新设置。



4 JavaFx 编程

4.1 官方指南

<https://docs.oracle.com/javafx/2/>

4.2 最新版 JavaFx 的配置

<https://openjfx.io/openjfx-docs/>

4.3 利用 CSS 制作界面风格

JavaFX CSS 参考指南：

<https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>

编程指南：

https://docs.oracle.com/javafx/2/get_started/css.htm

https://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm

4.4 开启/关闭 hidpi

Java 9 以后 JavaFx 已实现了 dpi-aware，在 Java12 中缺省是开启的。

可以通过以下属性来开启/关闭 JavaFx 的 hidpi：

```
System.setProperty("prism.allowhidpi", "false");
```

注意：这个属性需要在启动 JavaFx 之前设置，否则无效。

4.5 控件截图的分辨率

利用方法 `snapshot()` 可以对控件当前状态进行截图。Java 8 的控件截图总是所见即所得，而 Java9 以后的控件截图分辨率很低。

以下网文解释了这个现象：

<http://news.kynosarges.org/2017/02/01/javafx-snapshot-scaling/>

在我的环境中：

"Toolkit.getDefaultToolkit().getScreenResolution()" 返回 216

"Screen.getPrimary().getDpi()" 返回 72

"Screen.getPrimary().getOutputScaleX()" 返回 2.25

$72 * 2.25 = 162$

$96 * 2.25 = 216$

$216 / 72 = 3$

所以 MyBox 中的代码为了得到清晰的截图，只能采用这种办法把控件先放大 3 倍再截图。坏处是截图的尺寸很大。以下是一个例子：

```
Bounds bounds = webView.getLayoutBounds();
double scale = Toolkit.getDefaultToolkit().getScreenResolution()
    / Screen.getPrimary().getDpi();
if (scale < 1) scale = 1;
int imageWidth = (int) Math.round(bounds.getWidth() * scale);
int imageHeight = (int) Math.round(bounds.getHeight() * scale);
SnapshotParameters snapPara = new SnapshotParameters();
snapPara.setFill(Color.TRANSPARENT);
snapPara.setTransform(javafx.scene.transform.Transform.scale(scale, scale));
WritableImage snapshot = new WritableImage(imageWidth, imageHeight);
snapshot = webView.snapshot(snapPara, snapshot);
```

MyBox JVM属性

终端风格



当前用户名称	mara
当前用户的根目录	C:\Users\mara
当前程序的根目录	D:\MyBox
MyBox数据目录	D:\tmp\mybox-data\mybox
MyBox数据库	jdbc:derby://localhost:1527/ D:\tmp\mybox-data\mybox\mybox_derby ;user=mara;password=mybox;create=false
Java虚拟机名称	12.0.1
Java虚拟机供应商	Oracle Corporation
Java虚拟机名称	Java HotSpot(TM) 64-Bit Server VM
Java虚拟机信息	mixed mode, sharing
Java根目录	D:\Programs\Java\jdk-12.0.1
Java临时文件目录	D:\SysTemp\
JavaFX运行时刻版本	13.0.1+1
物理内存	8102MB
JVM最大可用内存	2026MB
本地文件编码	GBK
Unicode输入输出编码	UnicodeLittle
CPU字节序	little
Sun桌面	windows
物理屏幕	216dpi 解析度:3,840 x 2,160 刷新率:60 位深:32
JavaFX屏幕	72dpi 尺寸: 1,707 x 960 横向拉伸比: 2.25 纵向拉伸比: 2.25
分辨率感知已关闭	false
文件编码	GBK

4.6 动态设置 Fxml 的 Controller

参考:

<https://stackoverflow.com/questions/23132302/invocationtargetexception-when-running-a-javafx-program/53129147?r=SearchResults#53129147>

例子:

```
FXMLLoader loader = new FXMLLoader(getClass().getResource("main.fxml"));
loader.setController(new MainController(path));
Pane mainPane = loader.load();
```

可见，这种方法还可以为 Controller 设置参数。

但是，fxml 中所有控件的 action 都不能直接定义了。

4.7 动态修改 chart 的 legend

参考:

<https://stackoverflow.com/questions/37634769/dynamically-change-chart-colors-using-colorpicker/37646943?r=SearchResults#37646943>

以下是 MyBox 的一段代码:

```
Set<Node> legendItems = histogramChart.lookupAll("Label.chart-legend-item");
if (legendItems.isEmpty()) {
    return;
}
for (Node legendItem : legendItems) {
    Label legendLabel = (Label) legendItem;
    Node legend = legendLabel.getGraphic();
    if (legend != null) {
        String colorString = FxmlColor.rgb2Hex(colorTable.get(legendLabel.getText()));
        legend.setStyle("-fx-background-color: " + colorString);
    }
}
```

4.8 “setAlwaysOnTop”带来的危险

考虑以下场景:

- 1) 窗口 A 被设置为 “setAlwaysOnTop(true)”
- 2) 窗口 B 弹出提示框 “alert.showAndWait()”，此时焦点在窗口 B 的提示框上，在用户作出选择之前其它任何窗口都得不到焦点。

3) 窗口 A 遮住了窗口 B 的提示框，因为窗口 A 得不到焦点，所以用户无法把窗口 A 挪开。于是程序处于死局，只能被强制退出了。

参考以下网文，避免这种冲突的方法是：程序应当总是把提示框自动提到最上面。

<https://stackoverflow.com/questions/38799220/javafx-how-to-bring-dialog-alert-to-the-front-of-the-screen?r=SearchResults>

```
Stage stage = (Stage) alert.getDialogPane().getScene().getWindow();
stage.setAlwaysOnTop(true);
stage.toFront();                                <---- 这是关键
```

以下问题报告已提交给 JDK 开发组：

https://bugs.java.com/bugdatabase/view_bug.do?bug_id=JDK-8230127

4.9 TableView/ListView 的“幽灵数据”

定制 TableView/ListView 的 Cell 时会改写方法“UpdateItem”，如果不处理“empty”分支则一定会碰到“幽灵数据”，即一些不存在的数据行显示在底部。

<https://stackoverflow.com/questions/26821319/javafx-listview-and-treeview-controls-are-not-repainted-correctly>

解决办法是必须写对于 empty 的处理：

```
@Override
protected void updateItem(final T persistentObject, final boolean empty) {
    super.updateItem(persistentObject, empty);
    if (empty) {                                <---- 必须有这个处理
        setText(null);
        setGraphic(null);
    } else {
        // ... rest of your code.
    }
}
```

4.10 内嵌页面监听事件

内嵌页面监听事件的特点是：当焦点不在内嵌页面时，它收不到事件通知；当焦点在内嵌页面时，主页面和内嵌页面同时收到事件通知。

这样会造成两种麻烦：1) 若主页面和内嵌页面因为继承同一基础类而有相同的处理逻辑和监听控件，则事件可能被处理两次。2) 焦点不在内嵌页面时有的事件不会被触发。

一个解决办法是把所有事件处理都放在主页面，但是这样主页面代码很繁杂。更好的方案是：内嵌页面不直接接收事件通知，而是由主页面转发事件通知。

4.11 强制 ListView 刷新

当添加或删除数据时，ListView 总是更新，但是当数据属性变化时，ListView 无法监测到因而不能更新。程序逻辑会要求必须刷新 ListView

<https://stackoverflow.com/questions/13906139/javafx-update-of-listview-if-an-element-of-observablelist-changes?r=SearchResults>

```
for (int i = 0; i < listView.getItems().size(); i++) {
    listView.getItems().set(i, listView.getItems().get(i));
}
```

4.12 一句话事项

1. 总是假设用户对编程一无所知。
2. 不要指望用户会看用户手册，一些人连屏幕上的提示都不会读或想（包括我）。
3. 必要的提示信息可以帮助用户理解程序的意图。
4. 二义性的提示不如没有提示。
5. 总是响应用户的操作，如弹出信息、发出声音、或更新界面。
6. 尽可能多地提供选择，并设置缺省选择。
7. 记住用户的输入或选择，下次自动填写或设置。
8. 保持界面干净的同时，减少用户的输入或选择。
9. 只要有地方放单选钮/多选纽，就不用下拉框。
10. 选择的层级不应超过 3。
11. 界面控件较多时，考虑目前流行的布局：左右幕布式收放、上下风箱式开闭、页签堆叠。
12. 用户屏幕分辨率可能从 1K 到 4K，所以 MyBox 多数界面的设计尺寸小于 1100*720。
13. 善用“会折行”的容器，如 FlowPane，以兼顾小屏和大屏。
14. 只能在 JavaFx 线程中修改界面控件。在控件的事件处理中若要修改控件本身，则应用 Platform.runLater 来执行。
15. 对于耗时的操作应当显示进度信息。
16. 耗时的操作不应在 JavaFx 线程中执行，否则会阻塞界面的更新。
17. 后端线程要时时检查 task 是否已被取消，尤其在循环中或耗时操作前后。
18. 鼠标事件监听时，event.getX() 和 event.getY() 返回的是相对于 event.getSource()（监听控件）的坐标。如：在 ImageView 上监听鼠标点击，则取到的是相对于图像的坐标。
19. 以下场景可能触发多次迭代调用：同时监听 onMouseEntered 和 onMouseExited，然后在事件处理中改变鼠标的焦点（如弹出上下文菜单）或改变控件本身尺寸。
20. 当双击鼠标时，总是先触发单击事件、然后继续触发双击事件，因此需要注意事件处理中可能重复或冲突的操作。

5 图像处理

5.1 Java 的图像处理技术

请参考以下网文：

<https://docs.oracle.com/javase/8/docs/technotes/guides/imageio/>
https://docs.oracle.com/javase/8/docs/technotes/guides/imageio/spec/imageio_guideTOC.fm.html
<https://docs.oracle.com/javase/tutorial/2d/index.html>
<https://www.javaworld.com/article/2076764/java-se/image-processing-with-java-2d.html>

5.2 图像的元数据

请参考官方定义：

https://docs.oracle.com/javase/10/docs/api/javax/imageio/metadata/doc-files/standard_metadata.html
https://docs.oracle.com/javase/10/docs/api/javax/imageio/metadata/doc-files/gif_metadata.html
https://docs.oracle.com/javase/10/docs/api/javax/imageio/metadata/doc-files/jpeg_metadata.html
https://docs.oracle.com/javase/10/docs/api/javax/imageio/metadata/doc-files/png_metadata.html
https://docs.oracle.com/javase/10/docs/api/javax/imageio/metadata/doc-files/tiff_metadata.html

5.3 多帧图像文件

即一个文件中保存有多个独立的图像。

目前 MyBox 支持的多帧图像文件格式是：动画 gif 和 tiff/tif。

5.4 Tiff 文件格式

关于 Tiff/Tif:

<https://en.wikipedia.org/wiki/TIFF>
<https://en.wikipedia.org/wiki/GeoTIFF>
<https://www.adobe.io/open/standards/TIFF.html>

5.5 动画 Gif

请参考以下网文：

<http://glib.sourceforge.net/whatsinagif/index.html>
<https://www.jianshu.com/p/df52f1511cf8>
<https://stackoverflow.com/questions/22259714/arrayindexoutofboundsexception-4096-while-reading-gif-file>
<https://github.com/DhyanB/Open-Imaging>
<https://programtalk.com/python-examples/com.sun.media.imageioimpl.plugins.gif.GIFImageWriterSpi/>

5.6 大图像

对内存造成压力的是图像的像素总数、而不是图像文件的字节数。

例如，一个 42M 的 jpg 文件，包含 65500×4504 个像素，每个像素占 3 字节，故图像数据占用内存 844M 字节。若要把整个图像显示在 JavaFX 界面上，数据需要在文件、`BufferedImage`、和 `WritableImage` 之间倒腾，本以为需要内存约三倍多一点即 2.6G，但是结果是 4.5G 都不行，最后大约需要申请 5.2G 内存才加载成功，而运行时显示只占 2.3G。

又如，一个 52M 的 png 文件，像素 8101×4557 ，图像数据是 147M 字节，只需要内存大约 750M 就可以了。

MyBox 处理大图像的原则是：

- 1) 评估加载整个图像所需内存, 判断能否加载整个图像。（按像素数据的 5 倍+200M）
- 2) 若可用内存足够载入整个图像，则读取图像所有数据做下一步处理。尽可能内存操作而避免文件读写。
- 3) 若内存可能溢出，则采样读取图像数据做下一步处理。
- 4) 采样比的选择：即要保证采样图像足够清晰、又要避免采样数据占用过多内存。
- 5) 采样图像主要用于显示图像。已被采样的大图像，不适用于图像整体的操作和图像合并操作。
- 6) 一些操作，如分割图像、降采样图像，可以局部读取图像数据、边读边写，因此适用于大图像：显示采样图像、处理原图像。
- 7) 无法整体加载处理的图像，未必不能批量处理。例如，一个 500M 像素数据的图像，无法在 1.8G 可用内存的限制下全部加载显示，因此无法实现交互式的剪裁、缩放和颜色处理；但是做批量剪裁、缩放和颜色处理时，因为图像不显示到界面上，操作所需内存可能只需 1.7G，从而执行成功。

连续处理较大的图像，会影响内存占用情形：前一次操作的内存可能还没有被回收，则下一次操作的可用内存就变少了。所以，最好重新打开工具来处理大图像，这样它可以就占用绝大多数可用内存。

为了能交互式处理大图像，用户可以扩展 JVM 的最大内存。

5.7 图像采样

图像采样分为：降采样（Downsampling，也叫子采样 Subsampling）、和升采样（Upsampling，也叫插值 interpolating）。

当图像像素值非常大时，降采样有助于在可用内存的有限范围内加载和显示整个图像的样子。

降采样的算法很简单：指定宽度和高度的采样比率，则图像的像素被选择提取出来。例如，比率为 3，表示宽度 3 高度 3 的相邻矩阵只取一个像素点。

图像降采样主要用来处理大图像。当采样率为 1 时，相当于剪裁功能。它与“图像处理”工具的“剪裁”功能和“大小”功能的差别在于：对于大图像，此工具是只读采样区域、边读边写，从而避免内存溢出。对于小图像，此工具与“图像处理”工具一样：都是读取图像所有数据、在内存里处理数据。

5.8 图像的灰度

请参考以下网文：

https://en.wikipedia.org/wiki/HSL_and_HSV

<https://en.wikipedia.org/wiki/Grayscale>

5.9 颜色距离

请参考以下网文：

https://en.wikipedia.org/wiki/Color_difference

5.10 如何把图像转变成褐色

请参考以下网文：

<https://stackoverflow.com/questions/21899824/java-convert-a-greyscale-and-sepia-version-of-an-image-with-bufferedimage/21900125#21900125>

5.11 图像的混合模式

请参考以下网文：

https://en.wikipedia.org/wiki/Blend_modes

<https://baike.baidu.com/item/混合模式/6700481?fr=aladdin>

<https://blog.csdn.net/bravebean/article/details/51392440>

<https://www.cnblogs.com/bigdream6/p/8385886.html>

5.12 卷积

请参考以下网文：

<https://en.wikipedia.org/wiki/Convolution>

[https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

<http://colah.github.io/posts/2014-07-Understanding-Convolutions/>

5.13 “漫灌” 算法

https://en.wikipedia.org/wiki/Flood_fill

5.14 图像尺寸

请区分以下概念：

原始尺寸: 图像文件中保存的像素值

加载尺寸: 图像在内存中的像素值。加载、剪裁、变形都会改变加载尺寸。

显示尺寸: 用户在界面上缩放图像、最终显示在屏幕上的像素值。

选择尺寸: 用户在图像上选择的区域，其像素值被自动映射到内存区域、并按原始尺寸的比例来计算。

例如，图像原始尺寸为 1000x500，加载尺寸为 800x400，显示尺寸为 600x300，选择尺寸为 700x200。

利用加载宽度，可以避免因为内存限制而无法查看大图像，也可以放大较小的图片读取进来。

加载宽度不同于界面上的缩放：加载像素决定了处理图像所需内存，而界面缩放影响屏幕上实际显示的像素。

5.15 图像量化

减少图像中颜色数量的技术称为“颜色量化”。 “抖动处理”可以改善颜色量化后的图像的质量。

为了不同的目的可以选择不同的算法：

对于多数图像的 256 色颜色量化，RGB 均等量化最快并且足够好。

K-Means 聚类适用于计算图像中最不同的颜色。

统计量化适用于计算图像中出现最多的颜色。

对于一些量化算法，颜色空间将被分割成一些区间以缩小数据集。参数“区间位深”决定预处理的数据范围大小。例如，4 位位深意味着从 256*256*256 的颜色空间映射到 16*16*16 个颜色值。在图像的像素被映射到新的区间后，量化将运行于这个更小的值集。所以实际上将发生两级量化，因为区间映射也是一种量化。

注意：较大的位深并不意味着更好的结果，它总是花费更长的运行时间并且可能给出更糟糕的输出。结果的质量依赖于图像属性、算法、和参数。例如，当图像包含较少的不同颜色，则小的位深更好。

请参考以下网文：

http://web.cs.wpi.edu/~matt/courses/cs563/talks/color_quant/CQindex.html

<http://tpgit.github.io/UnOfficialLeptDocs/leptonica/color-quantization.html>

https://www.researchgate.net/publication/220502178_On_spatial_quantization_of_color_images

http://rosettacode.org/wiki/Color_quantization

<http://www.imagemagick.org/Usage/quantize/>

5.16 滤镜

参考以下网站

<http://www.jhlabs.com/ip/index.html>

6 Java 编程

6.1 数据精度

关于是否使用 BigDecimal 来确保精度，以下网文很有启发：

<https://stackoverflow.com/questions/322749/retain-precision-with-double-in-java>

大致说：“不要浪费精力在使用 BigDecimal 上。绝大多数情况下你不需要它。”

“BigDecimal 比 double 慢很多”

“解决方案依赖于问题本身：

- 若只是减少小数位数，使用字符串格式。不要显示超过 15 位有效数字或 7 位浮点。
- 若是判断数字范围，应当用 `if (abs(x - 7.3) < TOLERANCE)` 而不是 `if (x == 7.3)`。
- 若处理货币，应当用整型保存各个货币单位。
- 若需要超过 53 位有效位数的精度，则用高精度浮点类型，如 BigDecimal。”

6.2 字符集和编码

字符集指：字符（显示/打印）与字节（计算机存储）的对应关系。而编码指：字节在载体（文件/内存）中存放方式。

<https://en.wikipedia.org/wiki/UTF-8>

<https://www.cnblogs.com/ChurchYim/p/8427373.html>

<https://www.cnblogs.com/maohuidong/p/8044568.html>

6.3 问题处理

6.3.1 “cannot access class ... because module ... does not export ...”

曾经在 netbean11.1 + openjdk 12.0.1 + openjfx12.0.1 环境下，碰到了以下错误：

```
Exception in thread "JavaFX Application Thread" java.lang.IllegalAccessError: class
mara.mybox.controller.ImageViewController (in unnamed module @0x1ec2591a) cannot access class
com.sun.javafx.charts.Legend (in module javafx.controls) because module javafx.controls does not export
com.sun.javafx.charts to unnamed module @0x1ec2591a
```

这个错误只出现在一处：显示图像数据的直方图，当增删颜色通道时需要刷新 legends，引用到未开放的内部类 com.sun.javafx.charts。

从 Java 9 以后代码包都分布在不同的 module 中，它们之间引用时需要配置环境：

<https://github.com/javafxports/openjdk-jfx/issues/236>

<http://mail.openjdk.java.net/pipermail/openjfx-dev/2018-June/021977.html>

<https://stackoverflow.com/questions/53237287/module-error-when-running-javafx-media-application>

<https://github.com/openjfx/javafx-maven-plugin>

<https://openjfx.io/openjfx-docs/#modular>

<https://community.oracle.com/blogs/vincentvauban/2018/07>

一些类虽然是 public 的，但是它们不在所在模块的 export 列表中，因此其它模块无法访问。

以下网文的介绍了这个问题及其解决办法：

<https://stackoverflow.com/questions/56459093/openjfx-custom-runtime-image-using-maven-and-jlink-module-exports-or-command-l/56467911?r=SearchResults#56467911>

MyBox 还没有模块化，解决办法是在 pom.xml 中添加以下语句：

```
<plugin>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-maven-plugin</artifactId>
    <version>0.0.3</version>
    <configuration>
        <mainClass>mara.mybox.MyBox</mainClass>
        <options>
            <option>--add-opens</option>
            <option>javafx.controls/com.sun.javafx.charts=ALL-UNNAMED</option>
        </options>
    </configuration>
</plugin>
```

最终的解决办法是：不再使用未公开的内部类，而用公开接口的方法替代实现。

6.3.2 “某些输入文件使用了未经检查或不安全的操作”

可参考：

<https://stackoverflow.com/questions/8215781/how-do-i-compile-with-xlintunchecked>

<https://stackoverflow.com/questions/262367/type-safety-unchecked-cast?r=SearchResults>

例如，问题代码：

```
List<String> names = new ArrayList();
Map<String, ImageInformation> oldList = new HashMap();
```

改为：

```
List<String> names = new ArrayList<>();
Map<String, ImageInformation> oldList = new HashMap<>();
```

利用批量替换可以轻易解决问题。

在 pom.xml 中加入编译选项以检查不合规的地方：

```
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
    <compilerArgument>-Xlint:unchecked</compilerArgument>
</configuration>
```

可以对方法添加注释以抑制检查：

```
@SuppressWarnings("unchecked")
public Object getValue() {
    return value;
}
```

6.4 一句话事项

1. 总是假设最极端的输入和最离奇的错误操作。
2. 假设用户硬件配置（cpu/内存/硬盘）略低于平均水平。
3. 文件读写用 BufferedInputStream 和 BufferedOutputStream。
4. 文件读写用 try-with-resource 语句。
5. 读写文本文件时，注意字符集。
6. 字符串与字节转换时，注意字符集。
7. 当类的参数比较多时，不要写很多构造方法来初始化不同参数集，而是用静态方法创建、然后各个 set 方法返回 this，来实现链式赋值。
8. 匿名类非常适合以下场景：大量重复代码，只有少数方法的实现不同。
9. 泛型非常适合以下场景：大量重复代码，只有数据类型不同。
10. 循环里尽可能不执行浮点计算。如果可以，把浮点计算转换为整型计算。
11. 监听数据变化时，需要判断是否真的发生变化，可以设置最小变化阈值以避免频繁处理。
12. 若修改 JVM 参数，或想粗暴清理应用环境时，需要重启应用。
13. 注意索引参数是基于 0 (0-based) 还是基于 1 (1-cased)，尤其是对第三方库。自己写的方法最好注明。一般默认基于 0，若是基于 1 则必须注明。

<文档结束>