**第一回 JavaFX2.0简介**

javaFX2.0是JavaFX最新的主要升级发布版。它的很多新特性并不能和1.3版本兼容。如果想要用javaFX做开发，推荐使用2.0版本。

建立SDK

开始使用javafx的第一步就是在你的电脑上下载和安装javafx SDK。你可以安装完整版SDK，也可以只安装javaFX运行时环境。详见第二回。

创建应用程序框架

每个JavaFX应用都有相似的框架。程序的主类继承了Application类，main方法调用luanch方法，start方法建立并显示舞台（stage，包括了应用的UI）。

定义UI

应用接口的舞台包括了一个或多个场景（scene，依次包含了控件、形状、图形来构建接口。）

布局UI控件

决定了在UI中使用什么控件后，使用内置的布局面板来管理控件的大小和位置。

部署

JavaFX应用运行在桌面、浏览器和网络上。完成应用后，将它们做成jar、html、jnlp文件供用户使用。

# 第二回 安装JavaFX2.0 SDK

通过本文，你讲学会如何在windows系统中安装JavaFX2.0。你有两个选择：

安装JavaFX SDK, 包括JavaFX运行时

仅安装JavaFX运行时

同时本文还指导了如何运行样例程序和怎样搭建JavaFX的NB环境。

Mac系统也有自己的 JavaFX 2.0开发者预览版，详见  第三回.

安装JavaFX SDK

如果你想要用IDE或者命令行开发javafx应用，请安装JavaFX SDK。SDK包括了Javafx的类库和工程文件。

如何安装JavaFX SDK:

检查你的系统要求

到  <http://www.oracle.com/technetwork/java/javafx/downloads/index.html>找到Javafx sdk针对你系统的版本下载保存。

运行.exe文件安装。

如何卸载JavaFX SDK:

使用标准卸载程序进行卸载。

安装JavaFX Runtime

若你想在你桌面上、浏览器中或使用Java Web Start通过网络运行Javafx应用，可以只安装运行时。

注意;

如果你已经成功安装了JavaFX SDK，就无需再单独安装运行时了。

如何安装JavaFX Runtime:

检查系统要求

找到并下载你系统对应的版本。

运行.exe文件。

如何卸载JavaFX Runtime

使用标准卸载程序进行卸载

运行样例程序

下载javafx样例程序，里面有一系列用javafx构建的应用。要运行它们，必须安装好运行时。

到  <http://www.oracle.com/technetwork/java/javafx/downloads/index.html.>点击JavaFX 2.0 Samples的链接,保存javafx\_samples-2\_0.zip到本地.

从zip文件中解压样例文件到本地目录。

进入目录。

双击其中任意一个可执行文件即可运行。比如，要在桌面运行Ensemble应用，双击Ensemble.jar。要在浏览器中运行，双击html。要通过网络运行，打开Ensemble.jnlp。

每个样例的源代码都在javafx-samples-2.0\src目录下。想要查看源代码，到javafx-samples-2.0\src\sample目录下，里面有你感兴趣的应用。每个样例的源代码都是NB工程。

搭建JavaFX的NetBeans环境

按照第四回搭建NB环境。

# 第三回 JavaFX2.0在mac系统的安装

JavaFX 2.0平台预览版包括了JavaFX SDK, 它提供了开发应用的工具和技术。

这次发布为JavaFX提供了Java API, 这为Java开发者提供了开放的功能，使得大量Java开发工具能用来开发javafx应用。此外，本次发布还提供了：

高性能图像引擎，为制作简易、平滑、快捷的富图像提供了高级支持。

媒体引擎，支持网络多媒体内容后台运行。

网络组件，是HTML能绑定在javafx应用中。

UI控件的扩展集，譬如Charts, Tables, Menus, 和 Panes.

FXML（XML-based markup language ） ，可以定义用户接口。

样例程序，快速展示了该技术的特性。

注意:

当前JavaFX应用仅能在Mac系统的桌面上运行。

系统要求

Mac OS下的JavaFX 2.0 开发者预览版需要以下软件:

操作系统 - Mac OS X 10.6 或更高

JDK - JDK 6 u 26 或更高

安装

Mac系统下的javafx文件是zip文件.

安装方法:

从<http://www.oracle.com/technetwork/java/javafx/downloads/javafx2-macosx-487281.html>下载JavaFX SDK zip文件。

从zip文件解压SDK文件到本地目录。

样例

JavaFX样例程序在<http://www.oracle.com/technetwork/java/javafx/downloads/javafx2-macosx-487281.html.> 下载该zip文件并解压到包含SDK的目录中，目录结构类似这样：

javafx-sdk2.0\

    bin\

    docs\

   javafx-samples-2.0\

    rt\

    .

    .

要在桌面运行应用，双击.jar文件。

每 个样例的源代码都在javafx-samples-2.0\src目录下。想要查看源代码，到javafx-samples-2.0\src \sample目录下，里面有你感兴趣的应用。每个样例的源代码都是NB工程。按照 Setting Up NetBeans IDE with JavaFX 2.0在NB中创建应用。

已知BUG和事项

以下问题被发现存在。后面的数字参考JavaFX issuesdashboard中的报告。

在Dock和菜单栏中应用名称只能是"Java"，无法更改。"关于"和"退出"中的名称是"Glass"，也不能改。[RT-12113, RT-13010]

当窗口第一次显示时，出现窗口黑屏，原因是不协调的视觉闪光。[RT-16804]

拖拉操作没有用户反馈。[RT-16307, RT-14624]

工作区:收到设置一个游标，并使用一个浮动窗口显示图像来反映拖动操作。

退出时可能异常[RT-14118, RT-15599]

工作区添加应用时悬挂或破碎不太可能。

已知渲染时有掺混。 [RT-12852]

不支持3D渲染。 [RT-13234, RT-13230]

Mac OS渐变支持不好. [RT-10565]

不能向用户展示错误对话框. [RT-13739]

Caps Lock 在 TextField 和TextArea中无效. [RT-16616]

在TextField中不能剪切、复制、粘贴. [RT-16475]

# 第四回 在NB中创建JavaFX2.0应用

NetBeans IDE需要一个 JavaFX 2.0可用的Java平台。本部分详述了如何在NB上建立JavaFX2.0可用Java平台。

当你首次新建应用或者打开预加载向导时NetBeans IDE会尝试创建 JavaFX可用的Java平台。如果NB不能自动创建会有警告出现，这时候你必须手动创建. 你可以创建额外的Java平台，比如你想要使用不同的jdk。

这里分两步介绍自动和手动创建Java平台：

打开新建JavaFX应用向导.这是全局的起点，当打开向导时， NetBeans IDE 尝试创建一个JavaFX可用的Java平台。如果 NetBeans 成功创建，你就完成任务了.

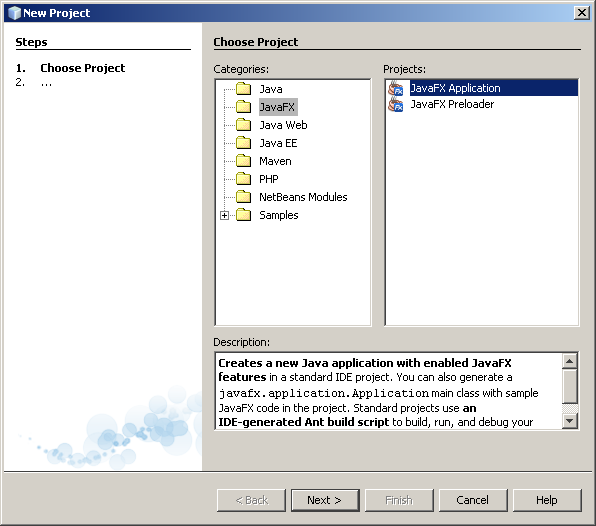
手动创建JavaFX可用的平台.如果自动创建失败 , 或者想要添加新的JavaFX可用平台, 你必须手动创建。

打开新建JavaFX应用向导

创 建 JavaFX可用的Java平台的第一步是打开NetBeans IDE的新建JavaFX应用向导。 (或者你可以打开新建JavaFX Preloader 向导。)如果NB没有找到JavaFX可用的Java平台， 会尝试创建。如果成功了，你的新建就完成了。否则你要手动创建。

注意:Windows平台自动创建到默认目录 (C:\Program Files\Oracle\).

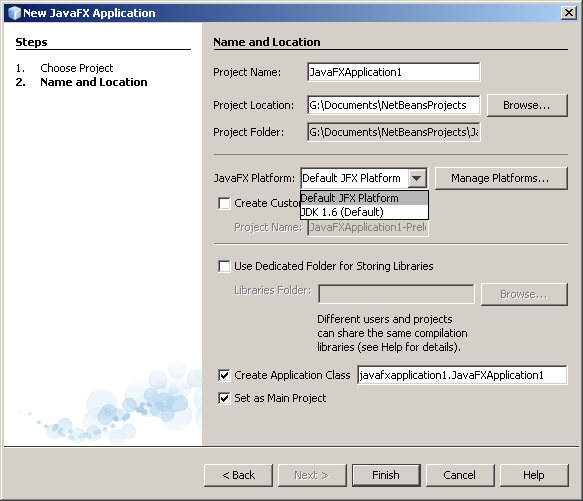
打开新建JavaFX Application向导:

[](http://web.ahzz.net/attachment/Mon_1111/22_2_95adc9d4ce0e7a9.png?33)

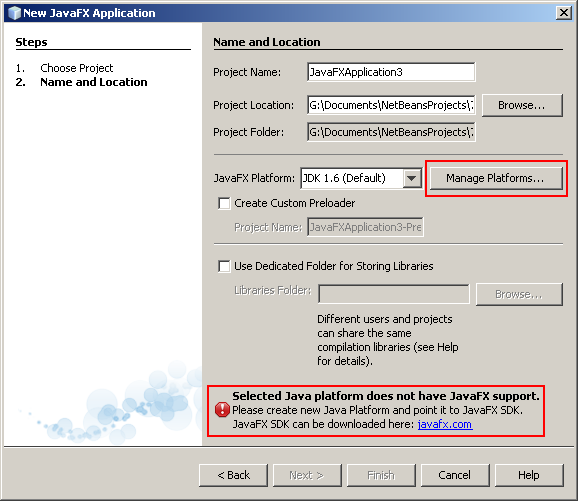
选择 JavaFX 分类. 在 Projects窗口, 选择JavaFX Application， 点击Next，打开Name and Location面板 。IDE自动寻找JavaFX SDK并尝试生成 JavaFX可用的Java平台。

NB创建环境的成功与否会出现不同的屏幕:

生 成成功:这个 JavaFX平台使用和IDE相同的 JDK，平台名称叫 Default JFX Platform。 IDE自动选择了这个平台。你成功了，除非你想用不同的jdk创建另外一个可用的平台。若你想这样，按照Creating a JavaFX-Enabled Platform Manually.

[](http://web.ahzz.net/attachment/Mon_1111/22_2_ffc44d75ed2f398.png?32)

生成失败:JavaFX Platform列表没有显示任何可用平台。查看  Creating a JavaFX-Enabled Platform Manually.

[](http://web.ahzz.net/attachment/Mon_1111/22_2_4a50aec5e87d39e.png?29)

手动创建JavaFX可用的平台

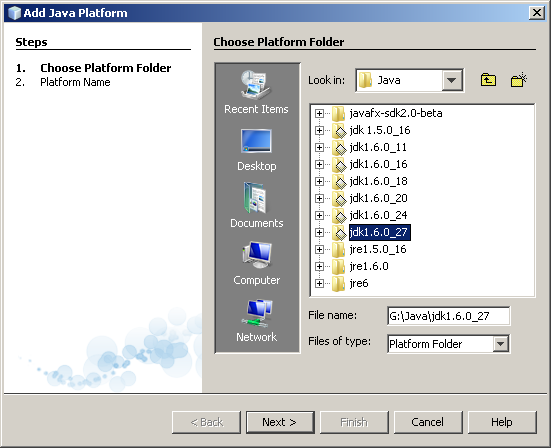
打开Java平台管理器NetBeans IDE Java Platform Manager：

在新建 JavaFX Application或者新建JavaFX Preloader向导中，点击Manage Platforms...；或者

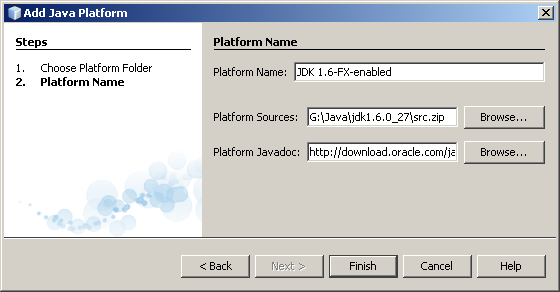
点开Tools 菜单选择Java Platforms。或者

打开任意Java工程的Project Properties，切换到Libraries标签，点击Manage Platforms...

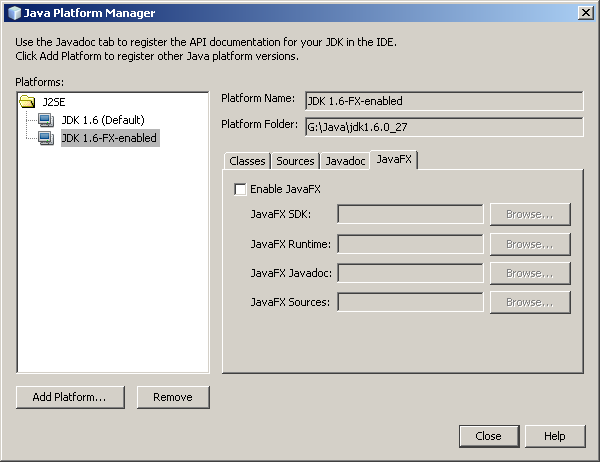
点击Add Platform...  Add Java Platform向导中出现Choose Java Platform 面板。找到你想要的jdk.



选中JDK。必须选择JDK 1.6 u 26或更高 (或JDK 7)。点 NexT，打开Platform Name面板。

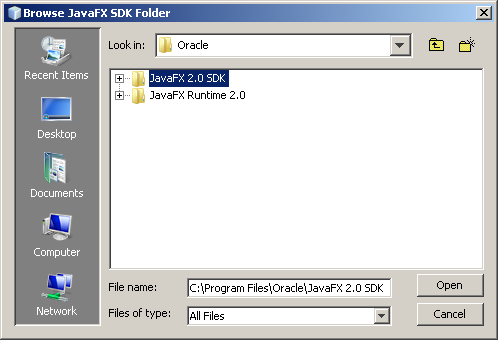


为你的平台命名后点Finish，返回到 Platform Manager。你创建的平台已经被加入列表了



选中你的平台，打开JavaFX标签。 现在，JavaFX 对该平台还不可用。选中Enable JavaFX框。警告出现了：“the JavaFX platform is invalid.”

点击JavaFX SDK 框旁边的Browse，找到 JavaFX 2.0 SDK文件夹。



点 击Open返回到Java Platform Manager。现在JavaFX SDK 和JavaFX Javadoc 框都填好了。如果JavaFX Runtime 2.0和JavaFX 2.0 SDK在同一目录，JavaFX Runtime 框也会填好的。若JavaFX Runtime 在其他目录，就收到找到Sources狂不用选。

点击Close。你已经完成了。

# 第五回 开始使用JavaFX2.0

如果想用 JavaFX快速开发富用户体验的应用，就好好读本文吧。我们将创建一个简单应用并了解到用少量代码实现复杂图形效果是多么容易。当然了，JavaFX不仅仅只是异常漂亮生动的形状而已。读完本文后去看看样例将大有好处。

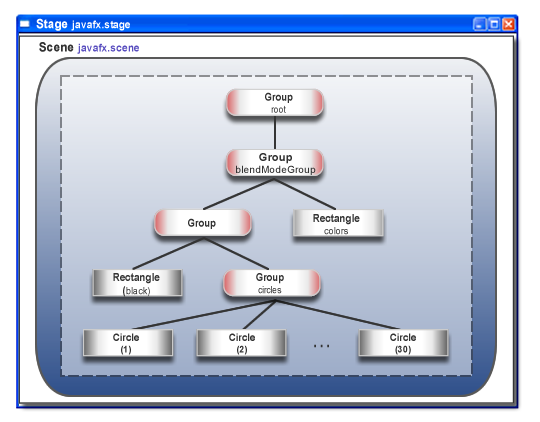
Figure 1 Colorful Circles Application



如果你很熟悉JavaFX场景图形编程模型，那就很容易理解程序的代码。 舞台（ stage）是应用的最高层容器，场景（ scene）是应用中内容的绘图表面。内容都被组织为场景图形，是一颗节点的层次树。

Figure 2 为应用 ColorfulCircles展示了场景图形。 分支节点是 Group 类的实例化。非分支节点，即叶子节点，是 Rectangle 和Circle 类的实例化。

Figure 2 Colorful Circles Scene Graph



建立应用

你可以使用任何为开发Java设计的工具来构建javaFX应用。我们推荐使用NetBeans IDE。 开始之前，请确保你的NB版本已经支持了javaFX2.0。详见 System Requirements。

安装如下步骤用NB做开发：

从 File 菜单 , 选择New Project .

从JavaFX application 分类中, 选择JavaFX Application ， 点击Next .

输入工程名 ColorfulCircles 后点 Finish .

打开 ColorfulCircles.java 文件,复制 import 声明粘贴进你的工程覆盖NB自动生成的语句。

或者你可以使用NB的代码完成功能或Fix Imports 命令导入Import语句，不过记得确保包都以javafx开头。

创建应用基础

删除 NetBeans IDE生成的ColorfulCircles 类并用 Example 1 中的代码代替。以下是运行javaFX应用所需的最少代码：

Example 1 Basic Application

public class ColorfulCircles extends Application {

    public static void main(String[] args) {

        launch(args);

    }

    @Override

    public void start(Stage primaryStage) {

        primaryStage.show();

    }

}

ColorfulCircles类继承了 Application 类，包含两个方法。第一个方法是 main() 方法， 用来调用 launch() 方法。作为JavaFX最佳实践，launch() 方法是被  main() 方法调用的唯一方法。

然后， ColorfulCircles 类重写了抽象的 start() 方法。start() 方法的参数是应用的初始舞台。最后一行使得舞台可见。

现在你可以编译运行ColorfulCircles了，每一步都记得查看下中间结果。如果出问题了，查看下 ColorfulCircles.java 文件。

添加场景

现在为舞台添加场景：增加三行代码，参见Example 2 。这有两个最佳实践： 代码中为场景创建了一个Group节点做根节点，并设置了场景的宽和高（这里是800和600）。

在 primaryStage.show() 这一行之前添加场景及其所有内容。这是另一个最佳实践。

Example 2 Scene

@Override

public void start(Stage primarystage) {

    Group root = new Group();

    Scene scene = new Scene(root, 800, 600, Color.BLACK);

    primaryStage.setScene(scene);

    primaryStage.show()

添加图形

接下来在 primaryStage.show()行之前添加 Example 3 中的代码来创建30个圆。

Example 3 30 Circles

Group circles = new Group();

for (int i = 0; i < 30; i++) {

   Circle circle = new Circle(150, Color.web("white", 0.05));

   circle.setStrokeType(StrokeType.OUTSIDE);

   circle.setStroke(Color.web("white", 0.16));

   circle.setStrokeWidth(4);

   circles.getChildren().add(circle);

}

root.getChildren().add(circles);

这些代码创建了称为 circles 的Group，然后使用一个ofr循环向其中添加30个圆。每个圆半径到是150，并用white颜色填充，此外不透明度是5%，所以基本是透明的。

然后为这些圆创建边框，代码中包含了 StrokeType 类。 描边类型的 OUTSIDE 标明圆的边界向外扩展 StrokeWidth 的值，也就是4。描边的颜色是 white ，不透明度是16%，使得它比圆的颜色浅。

最后一行把 circles 加到根节点上。这只是临时结构，稍候将修改场景图形使得它匹配 Figure 2 展示那样。

Figure 4 展示了当前应用。由于代码没有为每个圆指定特定位置，它们都叠加在一起，并且窗口的左上角是圆心。层叠的圆的不透明度和黑色北京作用使得圆看起来是灰色的。

Figure 4 Circles



增加视觉效果

继续为圆应用盒子模糊效果使得它们看起来柔和。代码是 Example 4 。 在primaryStage.show() 之前添加这些代码。

Example 4 Box Blur Effect

circles.setEffect(new BoxBlur(10, 10, 3));

代码设置了模糊半径，宽和高都是10并且迭代3次，使它接近高斯模糊。 这样便在圆的边缘出现了平滑效果，看Figure 5 .

Figure 5 Box Blur on Circles



创建背景渐变

现在创建一个矩形并用线性渐变填充，代码见 Example 5 .

在 root.getChildren().add(circles) 之前添加这些代码，这样矩形才能在圆下面。

Example 5 Linear Gradient

Rectangle colors = new Rectangle(scene.getWidth(), scene.getHeight(),

     new LinearGradient(0f, 1f, 1f, 0f, true, CycleMethod.NO\_CYCLE, new

         Stop[]{

            new Stop(0, Color.web("#f8bd55")),

            new Stop(0.14, Color.web("#c0fe56")),

            new Stop(0.28, Color.web("#5dfbc1")),

            new Stop(0.43, Color.web("#64c2f8")),

            new Stop(0.57, Color.web("#be4af7")),

            new Stop(0.71, Color.web("#ed5fc2")),

            new Stop(0.85, Color.web("#ef504c")),

            new Stop(1, Color.web("#f2660f")),}));

root.getChildren().add(colors);

代 码创建了称为colors的矩形。 矩形和场景同宽高，并从左下角的（0，0）点开始到右上角的（1，1）点应用线性渐变。 true 表示渐变在矩形中是成比例的，NO\_CYCLE 表示颜色循环不会重复， Stop[] 序列表明了渐变颜色序列。最后一行把colors 添加到根节点。

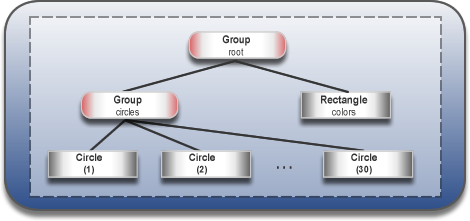
现在边缘模糊的灰色圆出现在了彩虹色的上面，见 Figure 6 .

Figure 6 Linear Gradient



Figure 7 展示了中间的场景图。现在circles 组和colors 矩形都是根节点的孩子。

Figure 7 Intermediate Scene Graph



应用混合模式

现在通过增加混合覆盖效果给圆增加颜色并使场景变暗。这个任务需要一点家务活，你需要从场景中移除 circles组和渐变的矩形，并把它们添加到新的混合覆盖组中。

删除下面两行代码:

root.getChildren().add(colors);

root.getChildren().add(circles);

添加 Example 6 中的代码到上面删除代码的位置。

Example 6 Blend Mode

Group blendModeGroup =

    new Group(new Group(new Rectangle(scene.getWidth(), scene.getHeight(),

        Color.BLACK), circles), colors);

colors.setBlendMode(BlendMode.OVERLAY);

root.getChildren().add(blendModeGroup);

blendModeGroup 组为混合覆盖组建立了结构。组中包含了两个孩子。第一个是一个新建匿名Group ，包含一个新建的匿名黑色矩形和以前创建的circles 组。第二个孩子是以前创建的colors 矩形。

setBlendMode() 方法把混合覆盖应用到了colors 矩形。最后一行代码把blendModeGroup 添加到场景作为根节点的孩子，如Figure 2 .

混合覆盖效果是图形设计程序中的常规效果。它可以暗化图像或高亮它们，这取决于混合组中的颜色。这里，我们把线性渐变矩形用作覆盖，黑色矩形用来保持背景黑暗，而接近透明的圆从矩形中取了色不过依然变暗了。

Figure 8 展示了结果。下一步活化了圆之后将能看到完整的混合覆盖效果。

Figure 8 Overlay Blend



添加动画

最后一步是使用javaFX动画来移动圆:

如果没准备好，增加 import static java.lang.Math.random; 到导入声明。

在 primaryStage.show() 之前增加Example 7 中的活化代码。

Example 7 Animation

Timeline timeline = new Timeline();

for (Node circle: circles.getChildren()) {

    timeline.getKeyFrames().addAll(

        new KeyFrame(Duration.ZERO, // set start position at 0

            new KeyValue(circle.translateXProperty(), random() \* 800),

            new KeyValue(circle.translateYProperty(), random() \* 600)

        ),

        new KeyFrame(new Duration(40000), // set end position at 40s

            new KeyValue(circle.translateXProperty(), random() \* 800),

            new KeyValue(circle.translateYProperty(), random() \* 600)

        )

    );

}

// play 40s of animation

timeline.play();

动 画是由时间线驱动的，所以这里创建了时间线，然后使用一个for 循环为30个圆增加两个关键帧。第一个关键帧在0秒时使用translateXProperty 和translateYProperty 属性设置窗口内的一个随机位置。第二个关键帧在40秒时同样做。这样，当时间线play()后，所有圆就在40秒内从一个随机位置到另一个。

Figure 9 展示了运动中的30个圆。完成后查看 ColorfulCircles.java 文件 .

Figure 9 Animated Circles



# 第六回 JavaFX2.0 UI 控件

JavaFX控件是通过API在场景图形中使用结点构建的，所以它们可以使用javaFX平台的富视觉特点。由于javaFX API是完全用Java语言实现的，所以可以轻松将javaFX UI 控件集成进已存在的Java应用中。

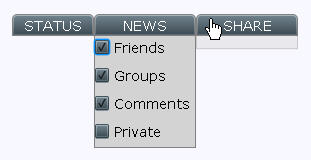
**JavaFX 2.0中支持的UI控件**

UI控件的构造类位于API的javafx.scene.control 包中。

控件列表包括了你可能很熟悉的用Java开发客户端应用的典型UI组件。不过，JavaFX 2.0 SDK引入了新的Java UI 控件，比如TitledPane 和TableView .

Figure 1-1 是一副屏幕截图，有三个TitledPane 元素和一个社交类型列表，并且列表可以滑入 (retract)和滑出 (extend).

Figure 1-1 Titled Panes



可以从 API文档查看全部UI控件。

UI 控件类比Control类提供了更多的变量和方法，这样就能以直观的方式支持典型的用户交互。你可以使用层叠样式表（CSS）为你的UI组件设计特殊样 式。对于某些个别任务，还可能要继承Control 类来创建定制的UI 组件，或者使用Skin 接口为已存在的控件定义一个新皮肤。

从样例中的Ensemble 应用试着了解下控件的范围、它们的行为、可以实现的样式。

**特性和效果**

由于javafx.scene.control包中的 UI 控件都继承了 Node 类，所以它们可以和场景图形的渲染、动画、变换及动画过度进行整合。

考虑创建一个按钮，为它添加倒影并通过时间线修改它的透明度来让它闪动。

Figure 1-2 展示了这个按钮的三个不同时间线上的状态。左边的图像是不透明度设为1.0时 ，中间的图像是不透明度 0.8 ，最右边的不透明度是0.5 .

Figure 1-2 Animated Button



通过使用JavaFX API只用少量代码就能实现这个任务。

Example 1-1 创建了一个无限的时间线并开始了它，里面有一个600毫秒的关键帧设置按钮的不透明度从默认值(1.0)向 0.0变化。setAutoReverse 使得时间线可以自动反向。

Example 1-1 Creating an Animated Button

[复制代码](javascript:)

1. import javafx.animation.KeyFrame;
2. import javafx.animation.KeyValue;
3. import javafx.animation.Timeline;
4. import javafx.util.Duration;
5. import javafx.scene.control.Button;
6. import javafx.scene.text.Font;
7. import javafx.scene.effect.Reflection;
8. ...
9. Button button = new Button();
10. button.setText("OK");
11. button.setFont(new Font("Tahoma", 24));
12. button.setEffect(new Reflection());
13. final Timeline timeline = new Timeline();
14. timeline.setCycleCount(Timeline.INDEFINITE);
15. timeline.setAutoReverse(true);
16. final KeyValue kv = new KeyValue(button.opacityProperty(), 0);
17. final KeyFrame kf = new KeyFrame(Duration.millis(600), kv);
18. timeline.getKeyFrames().add(kf);
19. timeline.play();

...

你也可以应用 javafx.scene.effect 包中的其他效果，比如shadow, lighting, 或者 motion blur.

**为UI控件添加CSS装饰**

通过定义自己的Cascading Style Sheets (CSS)可以定制内建的UI控件。在JavaFX 应用中使用CSS很像在HTML中使用，因为都必须遵循相同的CSS 规范。控件的视觉效果由.css文件定义，见代码Example 1-2 .

Example 1-2 Defining Styles for UI Controls in the CSS File

/\*controlStyle.css \*/

.scene{

    -fx-font: 14pt "Cambria Bold";

    -fx-color: #e79423;

    -fx-background: #67644e;

}

.button{

    -fx-text-fill: #006464;

    -fx-background-color: #e79423;

    -fx-border-radius: 20;

    -fx-background-radius: 20;

    -fx-padding: 5;

}

可以通过Scene类中的 getStylesheets 方法应用该效果，见Example 1-3 .

Example 1-3 Applying CSS

Scene scene = new Scene();

scene.getStylesheets().add("uicontrolssample/controlStyle.css");

//译者注：添加外部css文件时，即使文件和类在同意目录下，也要加上css所在的包名。

另外，你可以通过使用setStyle方法直接定义控件风格。 Example 1-4 中的 -fx-base 属性为场景中新增的双态按钮定义，它重写了CSS文件中对应的属性。

Example 1-4 Defining the Style of a Toggle Button in the JavaFX Application

ToggleButton tb3 = new ToggleButton ("I don't know");

tb3.setStyle("-fx-base: #ed1c24;");

Figure 1-3 展示了双态按钮的效果。

Figure 1-3 Applying CSS Style to a Toggle Button

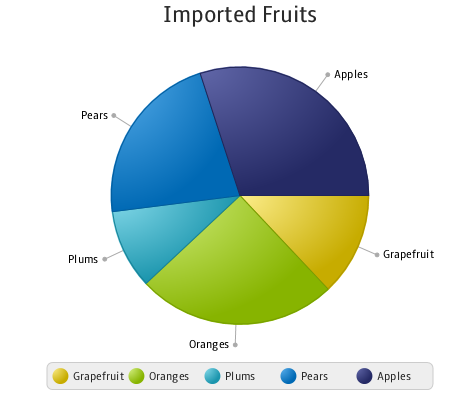
http://www.javafxfan.net/attachment/Mon_1111/3_2_a7e58f017af48cd.png?4000

图表

除 了为用户接口提供典型元素外， JavaFX SDK在 javafx.scene.chart包中 提供了预置图表。以下类型图表已经可以支持了：area chart, bar chart, bubble chart, line chart, pie chart, and scatter chart。一个图表可以包含几个系类的数据。

Figure 1-4 是一个进口水果饼图。

Figure 1-4 Pie Chart



和其他Java客户端工具不同，使用JavaFX SDK 只需要在应用中添加几行代码就能构建这样的图表。你也可以定义一系列的颜色表和风格、应用视觉效果、处理鼠标事件、创建动画等。

来 Using JavaFX Charts 了解更多的图表特性和功能信息。

**集成JavaFX 2.0 UI 控件和 Swing**

可以将 JavaFX UI 控件集成进已存在的用Swing构建的Java客户端应用。

要集成JavaFX内容和Swing，请安装以下步骤：

将JavaFX UI 控件一个一个地添加到javafx.scene.Scene 对象中的布局容器中，比如一个group.

把Scene 对象加入Swing 应用.

即使把一个单独的JavaFX 2.0 控件加入到已存在的Swing代码中也要做上面的两个步骤。

尽管它们被集成进了Swing程序，JavaFX 2.0 UI 控件也依然被菱镜图形库(Prism graphical library）渲染 ，并具有全部的高级渲染能力。

到第七回 了解更多二者的集成信息。

# 第七回 JavaFX2.0和Swing的集成

JavaFX 2.0 发布版引入了JFXPanel类，它位于 javafx.embed.swing 包中，使你能够将JavaFX内容绑定进Swing 程序。

本文教你怎么将JavaFX内容加进Swing 程序中并指导你如何在一个同时具备Swing 和JavaFX 的应用中正确使用线程。

**添加JavaFX到Swing组件**

要达到本文的目的，你要创建一个JFrame 组件，并添加一个JFXPanel 对象，JFXPanel组件的图形场景要包含JavaFX 内容。

和 所有的Swing程序一样，你要在 Event Dispatch Thread (事件调度线程EDT)上创建图形用户接口(GUI) 。Example 1 展示了initAndShowGUI 方法，它创建了一个JFrame 组件并添加了JFXPanel 对象。创建JFXPanel 类的实例会在后台开始JavaFX 运行时。GUI 创建后，调用initFX 方法在JavaFX线程上创建JavaFX场景。

Example 1

[复制代码](javascript:)

1. public class Test {
2. private static void initAndShowGUI() {
3. // This method is invoked on the EDT thread
4. JFrame frame = new JFrame("FX");
5. final JFXPanel fxPanel = new JFXPanel();
6. frame.add(fxPanel);
7. frame.setVisible(true);
8. Platform.runLater(new Runnable() {
9. @Override
10. public void run() {
11. initFX(fxPanel);
12. }
13. });
14. }
15. private static void initFX(JFXPanel fxPanel) {
16. // This method is invoked on the JavaFX thread
17. Scene scene = createScene();
18. fxPanel.setScene(scene);
19. }
20. private static Scene createScene() {
21. //Code to create the JavaFX scene
22. }
24. public static void main(String[] args) {
25. SwingUtilities.invokeLater(new Runnable() {
26. @Override
27. public void run() {
28. initAndShowGUI();
29. }
30. });
31. }
32. }

**JavaFX-Swing的互操作性和线程**

由于JavaFX和Swing的数据存在与一个程序中，你可能遇到以下互操作的情况:

一个JavaFX的数据改变是由Swing的数据改变引起的.

反之.

**改变JavaFX的数据来回应Swing数据的改变**

记住JavaFX的数据只能通过javaFX用户线程访问。不论何时要改变JavaFX 数据都要把你的代码用一个Runnable 对象包围起来并调用Platform.runLater 方法。见Example  2 .

Example 2

[复制代码](javascript:)

1. jbutton.addActionListener(new ActionListener() {
2. public void actionPerformed(ActionEvent e) {
3. Platform.runLater(new Runnable() {
4. @Override
5. public void run() {
6. fxlabel.setText("Swing button clicked!");
7. }
8. });
9. }
10. });

**改变Swing数据来回应JavaFX 数据的改变**

记住Swing数据的改变要通过EDT。确保你的代码实现了EDT,把它用Runnable 对象环绕并调用SwingUtilities.invokeLater 方法。见Example  3 .

Example 3

[复制代码](javascript:)

1. SwingUtilities.invokeLater(new Runnable() {
2. @Override
3. public void run() {
4. //Code to change Swing data.
5. }
6. });

**SwingBrowser2: 一个集成了JavaFX组件的Swing应用**

通 过SwingBrowser2 可以看到 Swing - JavaFX是怎么互操作的，它为浏览器提供了常用的基本功能。你可以在它的地址栏中键入一个URL 来查看应用窗口加载的页面。你可以通过点页面链接去到新页面、返回前一页、打开标签页、加入书签、全页面检索。 Figure 1是该应用的窗口。

Figure 1 The SwingBrowser2 application window



**初始化Swing数据**

你可以从一个NB工程中的侧边栏点链接来下载swingbrowser2.zip 文件。解压到本地并从Netbeans IDE中运行为工程。确保你的NetBeans IDE版本是支持的。

SwingBrowser2打开后，它的GUI创建就在 EDT上。通过边栏的链接查看 Main.java 文件。

应用的顶层窗口是一个 JFrame 组件，包含了很多Swing 组件，比如一个tabbed面板，一个menu，几个text field、button，还有一个要显示JavaFX内容的JFX 面板。

**加载 JavaFX内容**

刚开始运行，JFX面板包含一个空的WebView 对象。当在地址栏输入一个URL后， AddressBar.java中的 action listener 就开始加载页面。代码见Example 4 .

Example 4

[复制代码](javascript:)

1. txtURL.addActionListener(new ActionListener() {
2. @Override public void actionPerformed(ActionEvent e) {
3. browser.load(txtURL.getText());
4. }
5. });

Browser.java文件中的load 方法验证URL并调用call方法，见 Example 5 .

Example 5

[复制代码](javascript:)

1. public void load(String str) {
2. if (str != null {
3. str = str.trim();
4. if (str.isEmpty()) return;
5. String url = toURL(str);
6. if (url == null) {
7. url = toURL("http://" + str);
8. }
9. if (url != null) {
10. Platform.runLater (new Runnable() {
11. @Override
12. public void run () {
14. call(url);
15. }
16. }
17. }
18. private static String toURL(String str) {
19. try {
20. return new URL(str).toExternalForm();
21. } catch (MalformedURLException exception) {
22. return null;
23. }
24. }

JavaFX数据应该只能在JavaFX线程上访问。call方法验证了线程并确保特点URL的页面被加载在 JavaFX线程上。Callback.java 文件中有完整代码。Example 6 是表明了实现方案的代码块。

Example 6

[复制代码](javascript:)

1. public final void call(final String value) {
2. if (Platform.isFXApplicationThread()) {
3. callImpl(value);
4. }
5. else {
6. Platform.runLater(new Runnable() {
7. @Override public void run() {
8. callImpl(value);
9. }
10. });
11. }
12. }
13. protected void callImpl(String value) {
14. getEngine().load(value);
15. }

更新Swing数据

当 WebView 组件加载了新页面后，页面标题就从JavaFX数据中取回并传递给Swing GUI。然后页面标题就显示在标签页上并加入应用标题。 TabbedBrowser.java 中有完整代码。Example 7 是表明了实现方案的代码块。

Example 7

[复制代码](javascript:)

1. public WebPane addNewTab(final String url, boolean selected) {
2. ...
3. final WebPane wp = new WebPane(url);
4. wp.getBrowser().getEngine().titleProperty().addListener(
5. new javafx.beans.value.ChangeListener<String>() {
6. @Override
7. public void changed(ObservableValue<? extends String> observable,
8. String oldValue, final String title) {
9. EventQueue.invokeLater(new Runnable() {
10. @Override public void run() {
11. setTitleAt(indexOfComponent(wp), title);
12. setToolTipTextAt(indexOfComponent(wp), title);
13. if (getSelectedComponent() == wp) {
14. setWindowTitle(title);
15. }
16. }
17. )};
18. }
19. )};
20. }

想了解怎么部署二者的整合应用，到 [Deploying JavaFX Applications](http://download.oracle.com/javafx/2.0/deployment/deploy_swing_apps.htm)  查看。

# 第八回 JavaFX2.0 标签Label

Label 类位于 JavaFX API的 javafx.scene.control 包中，它继承了 Labeled 类。Label 类用来显示一个文本元素。你可以让一个文本换行来适应特定大小的空间，也可以加入图像。

Figure 2-1 展示了三个标签的常规用法。左边是带有图像的文本元素，中间的是转动后的文本，右边的是换行文本。

Figure 2-1 Sample Application with Labels



**创建Label**

JavaFX API提供了三个Label 类的构造方法来创建标签，见代码Example 2-1 .

Example 2-1 Creating Labels

//空标签

Label label1 = new Label();

//有文本的标签

Label label2 = new Label("Search");

//有文本有图像

Image image = new Image(getClass().getResourceAsStream("labels.jpg"));

Label label3 = new Label("Search", new ImageView(image));

创建标签后就可以用下面Labeled 类的方法向其中添加文本和图像内容。

setText(String text) – 为标签指定一个标题。

setGraphic(Node graphic) – 指定图标

setTextFill方法为标签的文本元素指定了颜色。研究下 Example 2-2：先创建了一个文本标签，又添加了一个图标，再指定文本的填充颜色。

Example 2-2 Adding an Icon and Text Fill to a Label

Label label1 = new Label("Search");

Image image = new Image(getClass().getResourceAsStream("labels.jpg"));

label1.setGraphic(new ImageView(image));

label1.setTextFill(Color.web("#0076a3"));

这块代码加入到程序中后，就产生了一个标签，见Figure 2-2 .

Figure 2-2 Label with Icon



当为按钮定义文本和图像内容时，可以用setGraphicTextGap 方法在之间产生空白。

另 外，可以在标签的布局设置区域使用setTextAlignment 方法来改变标签的位置。你也可以通过 setContentDisplay 方法为图像定义针对文本的相对位置，指定下面任意一个ContentDisplay常量： LFFT , RIGHT , CENTER , TOP , BOTTOM .

**设置字体**

比较一下 Figure 2-1 和 Figure 2-2 中的搜索标签，注意Figure 2-1 中的标签是大字体。这是因为Example 2-2 中的代码块没有为标签指定任何字体，它被默认文字大小渲染的。

使用Labeled 类的 setFont 方法为标签提供不同于默认值的文字大小。Example 2-3 在的代码块将label1 的文字大小设置为30号并且字体名称是Arial。为label2 设置的是32号和Cambria字体。

Example 2-3 Applying Font Settings

//Use a constructor of the Font class

label1.setFont(new Font("Arial", 30));

//Use the font method of the Font class

label2.setFont(Font.font("Cambria", 32));

**换行文本**

创建标签后，有时候必须让文本适应这个比它小的标签。 这时必须要打断文本 (换行) 来使它适应布局区域，为 setWrapText 方法设置true值即可。见 Example 2-4。

Example 2-4 Enable Text Wrapping

Label label3 = new Label("A label that needs to be wrapped");

label3.setWrapText(true);

把label3加入程序后，效果如Figure 2-3 .

Figure 2-3 Label with Wrapped Text



但 是如果标签的布局区域不仅仅是被限制了宽，还限制了高呢？当标签不可能渲染全部文本串时你可以为它指定行为。使用Labeled 类的setTextOverrun 方法和任一OverrunStyle 类型定义如何合适的处理只能部分显示的文本。查询API 文档来了解更多关于OverrunStyle 类型的信息。

使用特效

尽管标签是静态内容不能修改，不过依然可以应用特效或者变换它。 Example 2-5 中的代码块将label2 转动了270 °并且垂直方向平移了50。

Example 2-5 Rotating a Label

Label label2 = new Label ("Values");

label2.setFont(new Font("Cambria", 32));

label2.setRotate(270);

label2.setTranslateY(50);

旋 转和平移是 JavaFX API中的典型变换。此外，你可以为标签设置当用户让鼠标悬停时具有变焦效果 (放大)。Example 2-6 中的代码块为label3应用了变焦效果。 当标签的MOUSE\_ENTERED 事件被激发时，setScaleX 和setScaleY 方法设置其缩放比例是1.5。当用户的鼠标离开标签时MOUSE\_EXITED 时间发生，缩放比例设为1.0标签就变成原始大小了。

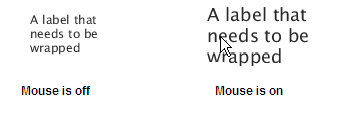
Example 2-6 Applying the Zoom Effect

[复制代码](javascript:)

1. label3.setOnMouseEntered(new EventHandler<MouseEvent>() {
2. @Override public void handle(MouseEvent e) {
3. label3.setScaleX(1.5);
4. label3.setScaleY(1.5);
5. }
6. });
7. label3.setOnMouseExited(new EventHandler<MouseEvent>() {
8. @Override public void handle(MouseEvent e) {
9. label3.setScaleX(1);
10. label3.setScaleY(1);
11. }
12. });

Figure 2-4   展示了label3的两个状态。

Figure 2-4 Zooming a Label



# 第九回 JavaFX2.0按钮Button

Button类可通过JavaFX API让用户通过点击一个按钮来处理某个行为。Button类是Labeled 类的子类，它可以显示文本、图片（当然也可以同时显示二者）。Figure 3-1展示了具有各种效果的按钮。通过本文你就能学会怎么创建这样的按钮。

Figure 3-1 Types of Buttons



**创建按钮**

在JavaFX应用中有三种Button类的构造方法创建 Button 控件，见Example 3-1 .

Example 3-1 Creating a Button

[复制代码](javascript:)

1. //A button with an empty text caption.
2. Button button1 = new Button();
3. //A button with the specified text caption.
4. Button button2 = new Button("Accept");
5. //A button with the specified text caption and icon.
6. Image imageOk = new Image(getClass().getResourceAsStream("ok.png"));
7. Button button3 = new Button("Accept", new ImageView(imageOk));

由于 Button 类继承了Labeled 类，所以你可以用下面的方法为没有图标和标题的按钮指定内容。

setText(String text) 方法–为按钮指定标题

The setGraphic(Node graphic) 方法– 指定图标

Example 3-2展示了如何创建无标题的图标按钮。

Example 3-2 Adding an Icon to a Button

Image imageDecline = new Image(getClass().getResourceAsStream("not.png"));

Button button5 = new Button();

button5.setGraphic(new ImageView(imageDecline));

将代码块加入应用后产生的效果如 Figure 3-2 .

Figure 3-2 Button with Icon

http://www.javafxfan.net/attachment/Mon_1111/3_2_0bd034a23a50b0c.png?2503

Example 3-2 和Figure 3-2 中的图标是一个ImageView 对象。不过，也可以使用其他的图形对象，如javafx.scene.shape 包中的各种形状。当按钮同时具有文本和图形内容时，可以使用setGraphicTextGap方法在二者之间设置间隙。

Button类的默认皮肤有如下的视觉区别。Figure 3-3 用一个图标按钮展示了这种区别。

Figure 3-3 Button States



**分配行为**

按钮的最基本功能就是在按下时处理行为。用Button 类的setOnAction方法定义用户按下按钮时会发生什么。 Example 3-3 是为button2定义行为的代码块。

Example 3-3 Defining an Action for a Button

button2.setOnAction(new EventHandler<ActionEvent>() {

    @Override public void handle(ActionEvent e) {

        label.setText("Accepted");

    }

});

ActionEvent 是由EventHandler处理的一种事件类型。EventHandler对象提供了handle方法处理按钮触发的行为。Example 3-3展示了如何重写handle 方法，所以当用户按下button2 时label 的标题就变为了"Accepted."

可以用Button 类设置任意多你需要引起特定行为或应用视效的事件处理方法。

**应用特效**

由于 Button类继承了Node 类，你可以使用javafx.scene.effect包中的任意特效改善按钮的视觉呈现。在Example 3-4中，当onMouseEntered 事件发生时为button3应用了DropShadow效果。

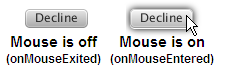
Example 3-4 Applying the DropShadow Effect

[复制代码](javascript:)

1. DropShadow shadow = new DropShadow();
2. //Adding the shadow when the mouse cursor is on
3. button3.addEventHandler(MouseEvent.MOUSE\_ENTERED,
4. new EventHandler<MouseEvent>() {
5. @Override public void handle(MouseEvent e) {
6. button3.setEffect(shadow);
7. }
8. });
9. //Removing the shadow when the mouse cursor is off
10. button3.addEventHandler(MouseEvent.MOUSE\_EXITED,
11. new EventHandler<MouseEvent>() {
12. @Override public void handle(MouseEvent e) {
13. button3.setEffect(null);
14. }
15. });

Figure 3-4是鼠标在button3 上移入移出的效果。

Figure 3-4 Button with Drop Shadow



**为Button造型**

下面对按钮的美化工作是通过Skin 类定义的CSS风格来实现的。在JavaFX 2.0 中使用CSS和在HTML中使用很类似，因为它们都基于相同的CSS规范。

可 以先定义一个单独的CSS文件，然后用 setStyleClass 方法在应用中使用它。这个方法也是继承自Node类的且对全部UI控件可用。当然你也可以使用setStyle方法直接在代码中定义按钮的风格。 Example 3-5 和Figure 3-4演示了后面这种方法。

Example 3-5 Styling a Button

button1.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9;");

-fx-font属性为button1设置了字体大小，-fx-base属性改变了按钮默认的颜色。这样，button1就具有了浅绿色和大字体，如Figure 3-5 .

Figure 3-5 Button Styled with CSS

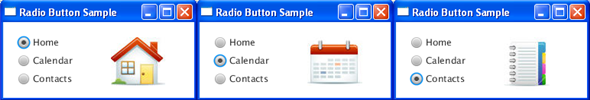
http://www.javafxfan.net/attachment/Mon_1111/3_2_e421a41ba4c69d6.png?4514

# 第十回 JavaFX2.0单选框Radio Button

RadioButton类是ToggleButton类的一个专业实现。一个单选按钮控件可以被选中和取消选中。典型的单选按钮是被放置在一个组里面，组里每次只能有一个按钮被选中。这种行为将它们和开关按钮区别开了，因为一个组中的所有开关按钮能同时被取消选中。

Figure 4-1是三幅RadioButton例子的截图，里面的三个单选按钮在同一个组中。

Figure 4-1 RadioButton Sample



通过研习下文能够了解更多关于在应用中实现单选按钮的信息。

**创建RadioButton**

RadioButton 类位于JavaFX SDK的javafx.scene.control包中，提供了两个创建单选按钮的构造方法。Example 4-1是创建两个单选按钮。无参数构造方法用来创建rb1，它的标题通过setText方法设置。而rb2的标题直接定义在相应的构造方法中。

Example 4-1 Creating Radio Buttons

[复制代码](javascript:)

1. //A radio button with an empty string for its label
2. RadioButton rb1 = new RadioButton();
3. //Setting a text label
4. rb1.setText("Home");
5. //A radio button with the specified label
6. RadioButton rb2 = new RadioButton("Calendar");

你可以通过为setSelected方法指定true值来明确地让一个单选按钮是选中状态。如果你想要检查一个特定的单选按钮是否被用户选中了，使用isSelected方法。

由于 RadioButton 类继承了Labeled 类，所以你不仅可以为其指定文本标题，还可以是图片。使用setGraphic方法来指定一副图片。Example 4-2演示了如何在应用中实现带图像的单选按钮。

Example 4-2 Creating a Graphical Radio Button

[复制代码](javascript:)

1. Image image = new Image(getClass().getResourceAsStream("ok.jpg"));
2. RadioButton rb = new RadioButton("Agree");
3. rb.setGraphic(new ImageView(image));

**将Radio Button加入到组**

单 选按钮的典型用法是在组中使用来提供几个互斥选项。ToggleGroup对象为所有的单选按钮提供了引用来关联自身，并且管理单选按钮来实现每次只能有 一个被选中。Example 4-3创建了一个开关按钮组、三个单选按钮，把每个按钮都加入到组中，并指定了在程序启动后哪个要被选中。

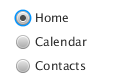
Example 4-3 Creating a Group of Radio Buttons

[复制代码](javascript:)

1. final ToggleGroup group = new ToggleGroup();
2. RadioButton rb1 = new RadioButton("Home");
3. rb1.setToggleGroup(group);
4. rb1.setSelected(true);
5. RadioButton rb2 = new RadioButton("Calendar"); rb2.setToggleGroup(group);
6. RadioButton rb3 = new RadioButton("Contacts");
7. rb3.setToggleGroup(group);

当这些单选按钮被它们的布局管理器添加到应用的内容上以后，输出应该类似于Figure 4-2.

Figure 4-2 Three Radio Buttons Combined in a Group



**处理Radio Button事件**

当组中的某个单选按钮被选中时程序会处理该行为。研读Example 4-4中的代码块来了解怎么根据哪个单选按钮被选中来改变图标。

Example 4-4 Processing Action for Radio Buttons

[复制代码](javascript:)

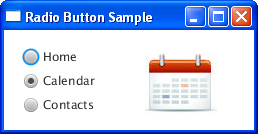
1. ImageView image = new ImageView();
2. rb1.setUserData("Home");
3. rb2.setUserData("Calendar");
4. rb3.setUserData("Contacts");
5. final ToggleGroup group = new ToggleGroup();
6. group.selectedToggleProperty().addListener(
7. new ChangeListener<Toggle>()
8. { public void changed(ObservableValue<? extends Toggle> ov,
9. Toggle old\_toggle, Toggle new\_toggle)
10. { if (group.getSelectedToggle() != null)
11. { final Image image =
12. new Image( getClass().getResourceAsStream( group.getSelectedToggle().getUserData().toString() + ".jpg" ) );
13. icon.setImage(image); } } });

比如，当rb3被选中时，getSelectedToggle方法返回"rb3,"getUserData方法返回"Contacts"。因此，getResourceAsStream方法接收了"Contacts.jpg."Figure 4-1是应用的输出。

**为Radio Button请求焦点**

在单选按钮组中，默认第一个按钮具有焦点。当你为组中的第二个单选按钮使用setSelected 方法后，你期望的结果是像Figure 4-3.

Figure 4-3 Default Focus Settings



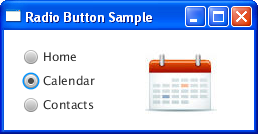
第二个按钮被选中了，但焦点依然在第一个按钮上。使用requestFocus函数可以改变焦点位置，见Example 4-5.

Example 4-5 Requesting Focus for the Second Radio Button

rb2.setSelected(true); rb2.requestFocus();

这样，代码产生的结果如Figure 4-4.

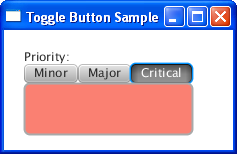
Figure 4-4 Setting Focus for the Selected Radio Button



# 第十一回 JavaFX2.0开关按钮ToggleButton

ToggleButton类代表了可以通过JavaFX API创建的另一类按钮。 2个或更多的这种按钮被加入一个组中，但是每次只能有一个被选中，或者一个都没有。 Figure 5-1是在一个组中有3个开关按钮的应用截图。这个应用根据哪个开关按钮别按下来决定绘制何种颜色的矩形。

Figure 5-1 Three Toggle Buttons



创建Toggle Button

可以通过ToggleButton类的3个构造方法的任意一个来创建开关按钮。见Example 5-1.

[复制代码](javascript:)

1. //A toggle button without any caption or icon
2. ToggleButton tb1 = new ToggleButton();
3. //A toggle button with a text caption
4. ToggleButton tb2 = new ToggleButton("Press me");
5. //A toggle button with a text caption and an icon
6. Image image = new Image(getClass().getResourceAsStream("icon.png"));
7. ToggleButton tb3 = new ToggleButton ("Press me", new ImageView(image));

ToggleButton类继承了Labeled类，因而你可以为其指定文本标题、图像、文本加图像。可以使用Labeled  类的setText和setGraphic方法来给开关按钮指定文本和图像。

在代码中定义了开关按钮后，就可以把它们放进组中并指定特定的行为。

**把Toggle Button加入组**

ToggleButton类的实现和RadioButton类实现相当类似。不过和单选框不同的是，开关按钮并不要求每次必须最少有一个按钮被选中在组中。也就是说，点击选中的开关按钮会使其取消选中，但是点击组中的单选按钮没任何反应。

t花点时间看下 Example 5-2中的代码吧.

[复制代码](javascript:)

1. Example 5-2 Combining Toggle Buttons in a Group
2. final ToggleGroup group = new ToggleGroup();
3. ToggleButton tb1 = new ToggleButton("Minor");
4. tb1.setToggleGroup(group);
5. tb1.setSelected(true);
6. ToggleButton tb2 = new ToggleButton("Major");
7. tb2.setToggleGroup(group);
8. ToggleButton tb3 = new ToggleButton("Critical");
9. tb3.setToggleGroup(group);

Example 5-2创建了3个开关按钮并把它们加入开关组中。tb1调用了setSelected方法，所以应用打开后它会被按下。但是，你也可以按起Minor按钮使得没有任何开关按钮被按下。见 Figure 5-2.

Figure 5-2 Three Toggle Buttons in a Group



Description of "Figure 5-2 Three Toggle Buttons in a Group"

一般使用开关按钮的组来为每个按钮分配特定行为。下一部分将解释如何使用这些开关按钮改变矩形的颜色。

**设置行为**

ToggleButton类从Node 类继承了setUserData方法，该方法可以让你给任意选中的选项赋特定的值。在 Example 5-3中，用户数据指明了要用什么颜色来绘制矩形。

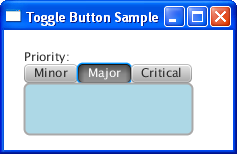
[复制代码](javascript:)

1. Example 5-3 Setting User Data for the Toggle Buttons
2. tb1.setUserData(Color.LIGHTGREEN);
3. tb2.setUserData(Color.LIGHTBLUE);
4. tb3.setUserData(Color.SALMON);
5. final Rectangle rect = new Rectangle(145, 50);
6. final ToggleGroup group = new ToggleGroup();
7. group.selectedToggleProperty().addListener(new ChangeListener<Toggle>(){
8. public void changed(ObservableValue<? extends Toggle> ov,
9. Toggle toggle, Toggle new\_toggle) {
10. if (new\_toggle == null)
11. rect.setFill(Color.WHITE);
12. else
13. rect.setFill(
14. (Color) group.getSelectedToggle().getUserData()
15. );
16. }
17. });

ChangeListener<Toggle>对象检查了组中被按下的开关。如果没有开关按钮被按下，矩形就用白色绘制。如果某个按钮被按下，getSelectedToggle和getUserData方法会连续调用来返回一种颜色绘制矩形。

比如用户按下了tb2按钮， setSelectedToggle().getUserData()的调用返回 Color.LIGHTBLUE结果如图 Figure 5-3.

Figure 5-3 Using Toggle Buttons to Paint a Rectangle



Description of "Figure 5-3 Using Toggle Buttons to Paint a Rectangle"

查看 ToggleButtonSample.java文件检查应用的完整代码。

**美化Toggle Button**

可以通过为开关按钮应用CSS来改善应用的视效。在JavaFX 2.0应用中使用CSS和在HTML中使用几乎一样，因为它们都基于相同的CSS规范。Example 5-4使用setStyle方法改变了开关按钮的 -fx-base CSS熟悉。

Example 5-4 Applying CSS Styles to Toggle Buttons

tb1.setStyle("-fx-base: lightgreen;");

tb2.setStyle("-fx-base: lightblue;");

tb3.setStyle("-fx-base: salmon;");

程序中加入这些代码后开关按钮的变化见 Figure 5-4.

Figure 5-4 Painted Toggle Buttons



Description of "Figure 5-4 Painted Toggle Buttons"

你可以尝试ToggleButton类的其他CSS属性，或者应用动画、转换、 JavaFX API中的各种视效。

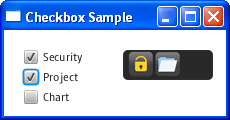
# 第十二回 JavaFX2.0 复选框CheckBox

CheckBox类让你可以在应用中创建复选框。尽管复选框看起来很想单选框

，但是它们并不能被放置进一个开关组中来实现很多选项的选择。你可以复习一下前面关于Radio Button 和Toggle Button相关的内容。

Figure 6-1  是有三个复选框的一个应用，这些复选框用来控制工具栏中的图标是否显示。

Figure 6-1 Checkbox Sample



**创建Checkbox**

Example 6-1   creates two simple checkboxes.

[复制代码](javascript:)

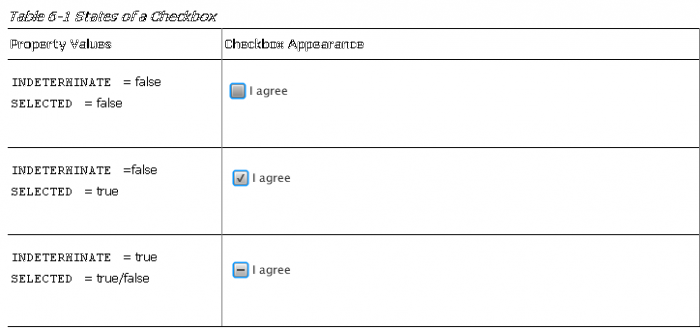
1. //Example 6-1 Creating Checkboxes
2. //A checkbox without a caption
3. CheckBox cb1 = new CheckBox();
4. //A checkbox with a string caption
5. CheckBox cb2 = new CheckBox("Second");
6. cb1.setText("First");
7. cb1.setSelected(true);

创建复选框后，就可以使用JavaFX API中的方法对它们进行设置。在Example 6-1 中 setText方法为 c1这个复选框定义了标题，设置setSelected方法为 true以使程序运行后 cb1是选中的。

**定义状态**

复 选框能被定义为明确或不明确。被定义为明确后就可以被选中或取消选中，但是不明确的话就不能被选中或取消选中。结合使用CheckBox类的 setSelected   和setIndeterminate方法来为复选框指定状态。 Table 6-1 是三个不同状态的复选框，它们的 INDETERMINATE   和SELECTED属性不相同。

Table 6-1 States of a Checkbox



当 复选框要呈现多种状态的UI元素时要为它们指定三种状态，如"Yes", "No", "Not Applicable"。CheckBox  类的 allowIndeterminatePropety设置了复选框对象是否能在全部三种状态（选中、取消选中、不明确）中循环 。如果设置了 ，这个控件就可以在三种状态中循环。否则 ，这控件只能在选中和取消选中两种状态中切换。下一部分的应用中构造了三个复选框，并且只有两个状态。

**设置行为**

Example 6-2 中的代码块创建了三个复选框，这样当一个复选框被选中后，相应的图标就显示出来。

[复制代码](javascript:)

1. //Example 6-2 Setting the Behavior for the Checkboxes
2. final String[] names = new String[]{"Security", "Project", "Chart"};
3. final Image[] images = new Image[names.length];
4. final ImageView[] icons = new ImageView[names.length];
5. final CheckBox[] cbs = new CheckBox[names.length];
6. for (int i = 0; i < names.length; i++)
7. { final Image image = images[i] =
8. new Image(getClass().getResourceAsStream(names[i] + ".png"));
9. final ImageView icon = icons[i] = new ImageView();
10. final CheckBox cb = cbs[i] = new CheckBox(names[i]);
11. cb.selectedProperty().addListener
12. (new ChangeListener<Boolean>() {
13. public void changed(ObservableValue<? extends Boolean> ov, Boolean old\_val, Boolean new\_val)
14. { icon.setImage(new\_val ? image : null);
15. } });
16. }

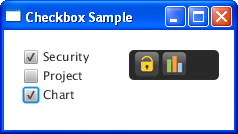
names这个数组使用了一个 for循环来创建复选框和相应的图标。比如说，

cbs[0]是第一个复选框，被分配了"Security"这个标题；同时image[0]接收了"Security.png"作为 getResourceStream方法的文件名。如果一个特定的复选框被选中后，相应的图片就被分配成它的图标。如果一个复选框被取消选中，图标接收

null图片所以没有图标分配。

Figure 6-2  是应用中的 Security 和Chart 复选框被选中，Project 取消选中。

Figure 6-2 Checkbox Application in Action



**美化Checkbox**

Figure 6-2  中的复选框有CheckBox  类 默认的外观和感觉。可以通过使用 setStyle  方法来改变它的样子，见 Example 6-3 .

[复制代码](javascript:)

1. //Example 6-3 Styling a Checkbox
2. cb1.setStyle( "-fx-border-color: lightblue; "
3. + "-fx-font-size: 20;"
4. + "-fx-border-insets: -5; "
5. + "-fx-border-radius: 5;"
6. + "-fx-border-style: dotted;"
7. + "-fx-border-width: 2;" );

新风格包括了一条浅蓝色点边框和一个扩大字体的标题。Figure 6-3  是使用了这种风格的 cb1复选框。

Figure 6-3 Styled Checkbox

http://www.javafxfan.net/attachment/Mon_1111/3_2_0dce5758fdf4873.png?2545

Description of "Figure 6-3 Styled Checkbox"

要为应用中的所有复选框设置特定的风格，按照下面的步骤：

创建一个 .css文件；

在 .css文件中创建 checkbox  CSS 类。

在checkbox   CSS 类中定义所有需要的风格。

在你的JavaFX应用中，使用setStyleClass  方法来使用CSS文件。

# 第十三回 JavaFX2.0 选项框ChoiceBox

ChoiceBox类为在几个选项中快速选择提供了支持。看看 Figure 7-1 中的选项框的这个简单实现怎么样。

Figure 7-1 Creating a Choice Box with Three Items



Description of "Figure 7-1 Creating a Choice Box with Three Items"

**创建 Choice Box**

Example 7-1 创建了带有三个条目的选项框。

Example 7-1 Creating a Choice Box

ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(

    "First", "Second", "Third")

);

Example 7-1 是创建在ChoiceBox类的构造函数中条目列表。这个条目列表是用一个可见的数组指定的。当然你可以选择使用该类的空构造方法，然后使用setItems方法设置条目列表。见Example 7-2.

Example 7-2 Choice Box with Text Elements and a Separator

ChoiceBox cb = new ChoiceBox();

cb.setItems(FXCollections.observableArrayList(

    "New Document", "Open ",

    new Separator(), "Save", "Save as")

);

不过选项框不单单能包含文本元素，还可以是其他对象。  Example 7-2中用一个Separator控件分开了条目。当应用中加入这些代码后，效果见 Figure 7-2.

Figure 7-2 Menu Created by Using a Choice Box



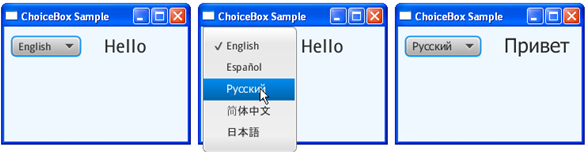
Description of "Figure 7-2 Menu Created by Using a Choice Box"

实际项目中，选项框用来创建多选列表。

**为Choice Box设置行为**

Figure 7-3 中的应用提供了一个带有5个选项的多选框。当一种特定的语言被选中后，相应的问候语就显示出来。

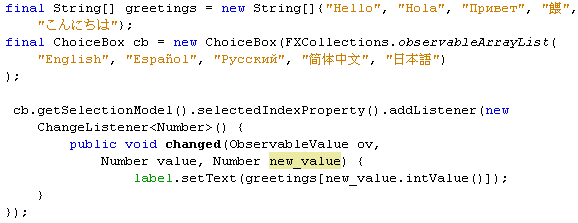
Figure 7-3 Multiple-Choice List



Description of "Figure 7-3 Multiple-Choice List"

Figure 7-4 中的代码块解释了选项框中的条目被选中后是怎么定义哪中语言的问候语应该显示的。

Figure 7-4 Selecting a Choice Box Item



通 过连续不断的调用getSelectionModel和selectedIndexProperty方 法，ChangeListener<Number> 对象侦测当前被选中条目的索引。getSelectionModel方法返回被选中的条目，selectedIndexProperty方法返回选项框的 SELECTED\_INDEX属性。这样，索引的整型值定义了问候数组中的元素，并为标签指定一个 String文本。如果一个用户选择了第二个条目，它对应的西班牙语，SELECTED\_INDEX就是1，"Hola"就被从问候语数字中选了出来。这 样，标签就显示"Hola."

可以通过为ChoiceBox控件分派提示条使它更有引导性。提示条是javafx.scene.control包中的一个UI控件。提示条能被用于任何JavaFX UI控件。

**使用Tooltip**

Tooltip类为在选项框（还有其他所有控件）添加提示条提供了预制方法，可以通过调用setTooltip方法很容易的使用。见Example 7-3.

Example 7-3 Adding a Tooltip to a Choice Box

cb.setTooltip(new Tooltip("Select the language"));

一般用户是在Tooltip类的构造方法内定义提示内容。不过，如果你的应用逻辑要求UI动态地设置文本，你可以使用空构造方法使用它，然后使setText方法为它设置文本。

选项框使用了提示条后，用户把鼠标放在选项框上就看到一个图像条。见 Figure 7-5.

Figure 7-5 Choice Box with the Applied Tooltip



Description of "Figure 7-5 Choice Box with the Applied Tooltip"

要改善应用的外观，可以给选项框使用CSS，或者使用视觉效果和变换。

# 第十四回 JavaFX2.0 文本框TextField

TextField类实现了一种可以接受和显示文本输入的UI控件，它提供了接受用户输入的功能。和另一个文本输入控件PasswordField一起都继承了TextInput这个类，TextInput是所有文本控件的父类。

Figure 8-1 是一个带有标签的典型文本框。

Figure 8-1 Label and Text Field

http://www.javafxfan.net/attachment/Mon_1111/3_2_5c5c6a535ff5718.png?1861

Description of "Figure 8-1 Label and Text Field"

**创建Text Field**

在 Example 8-1中，一个文本框和一个标签被用来显示输入的内容类型。

Example 8-1 Creating a Text Field

[复制代码](javascript:)

1. Label label1 = new Label("Name:");
2. TextField textField = new TextField ();
3. HBox hb = new HBox();
4. hb.getChildren().addAll(label1, textField);
5. hb.setSpacing(10);

你 可以像 Example 8-1中那样创建空文本框或者是创建带有特定文本数据文本框。要创建带有预定义文本的文本框，使用下面这个TextField类的构造方 法：TextField("Hello World!")。任何时候你都可以通过getText 方法获得一个文本框的值。

可以使用TextInput 类的setPrefColumnCount方法设置文本框的大小，定义文本框一次显示的最多字符数。

**用Text Field构建UI**

一般地， TextField对象被用来创建几个文本框。  Figure 8-2中的应用显示了三个文本框并且处理用户在它们当中输入的数据。

Figure 8-2 TextFieldSample Application



Description of "Figure 8-2 TextFieldSample Application"

Example 8-2 中的代码块创建了三个文本框和两个按钮，并把使用GridPane 容器他们加入到应用的屏幕上。当你要为你的UI控件实现灵活的布局时这个容器相当方便。

Example 8-2 Adding Text Fields to the Application

[复制代码](javascript:)

1. //Creating a GridPane container
2. GridPane grid = new GridPane();
3. grid.setPadding(new Insets(10, 10, 10, 10));
4. grid.setVgap(5);
5. grid.setHgap(5);
6. //Defining the Name text field
7. final TextField name = new TextField();
8. name.setPromptText("Enter your first name.");
9. name.setPrefColumnCount(10);
10. name.getText();
11. GridPane.setConstraints(name, 0, 0);
12. grid.getChildren().add(name);
13. //Defining the Last Name text field
14. final TextField lastName = new TextField();
15. lastName.setPromptText("Enter your last name.");
16. GridPane.setConstraints(lastName, 0, 1);
17. grid.getChildren().add(lastName);
18. //Defining the Comment text field
19. final TextField comment = new TextField();
20. comment.setPrefColumnCount(15);
21. comment.setPromptText("Enter your comment.");
22. GridPane.setConstraints(comment, 0, 2);
23. grid.getChildren().add(comment);
24. //Defining the Submit button
25. Button submit = new Button("Submit");
26. GridPane.setConstraints(submit, 1, 0);
27. grid.getChildren().add(submit);
28. //Defining the Clear button
29. Button clear = new Button("Clear");
30. GridPane.setConstraints(clear, 1, 1);
31. grid.getChildren().add(clear);

花 点时间来研究下这块代码。 name, lastName, 和comment文本框使用了TextField 类的空构造方法来创建。和 Example 8-1不同，这里文本框没有使用标签，而是使用提示语提醒用户在文本框中要输入什么类型的数据。setPromptText方法定义了当应用启动后显示在 文本框中的字符串。把 Example 8-2 中的代码加入到应用中，运行效果如 Figure 8-3.

Figure 8-3 Three Text Fields with the Prompt Messages



Description of "Figure 8-3 Three Text Fields with the Prompt Messages"

文本框中的提示语和文本的区别是提示语不能通过getText方法获得。

实际应用中，文本框中输入的文本是根据特定的业务任务决定的应用逻辑来处理的。 下一部分解释了如何使用文本框处理用户输入并向用户反馈。

**处理Text Field数据**

前面提到，用户输入文本框的内容能通过TextInput 类的getText方法获得。

研究Example 8-3 中的代码学习怎么处理TextField对象的数据。

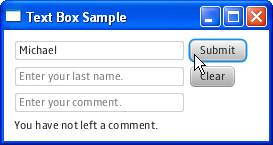
Example 8-3 Defining Actions for the Submit and Clear Buttons

[复制代码](javascript:)

1. //Adding a Label
2. final Label label = new Label();
3. GridPane.setConstraints(label, 0, 3);
4. GridPane.setColumnSpan(label, 2);
5. grid.getChildren().add(label);
6. //Setting an action for the Submit button
7. submit.setOnAction(new EventHandler<ActionEvent>() {
8. @Override
9. public void handle(ActionEvent e) {
10. if ((comment.getText() != null && !comment.getText().isEmpty())) {
11. label.setText(name.getText() + " " + lastName.getText() + ", "
12. + "thank you for your comment!");
13. } else {
14. label.setText("You have not left a comment.");
15. }
16. }
17. });
18. //Setting an action for the Clear button
19. clear.setOnAction(new EventHandler<ActionEvent>() {
20. @Override
21. public void handle(ActionEvent e) {
22. name.setText("");
23. lastName.setText("");
24. comment.setText("");
25. label.setText(null);
26. }
27. });

GridPane容器中的Label控件用来显示应用对用户的回应。当用户点击Submit按钮时，setOnAction方法检查comment文本框。如果它是非空字符串，一条感谢信息就显示出来。否则，应用会提醒用户还没有添加评论。见 Figure 8-4.

Figure 8-4 The Comment Text Field Left Blank



Description of "Figure 8-4 The Comment Text Field Left Blank"

当用户点击Clear按钮时，三个文本框的内容都将被清除。

.回顾一下你可能用到的文本框使用函数。

copy()– 将当前选择的文本范围转移到剪贴板，保留选择文本。

cut()– 将当前选择的文本范围转移到剪贴板，删除选择文本。

paste()– 将剪贴板内容转移到文本中，取代当前选择文本。

# 第十五回 JavaFX2.0 密码框PasswordField

PasswordField 类实现了一种特定的文本框：用户向其中输入的字符都被隐藏，代之显示的是特殊的回显串。Figure 9-1 是一个带有提示语的密码框。

Figure 9-1 Password Field with a Prompt Message

http://www.javafxfan.net/attachment/Mon_1111/3_2_d9fc9accf1e0a94.png?1962

Description of "Figure 9-1 Password Field with a Prompt Message"

**创建Password Field**

一个入门级的任务是创建一个密码框，见Example 9-1 .

Example 9-1 Creating a Password Field

PasswordField passwordField = new PasswordField();

passwordField.setPromptText("Your password");

由 于是用户接口，所以最好给密码框配上一条提示语或者是一个指示标签。和 TextField   类相同，PasswordField类也提供了 setText 方法来设置在应用启动后要在控件中显示的字符串。但是，setText 方法指定的字符串被密码框的回显字符覆盖了。默认地，密码框的回显字符是星号。Figure 9-2 是带有预定义文本的密码框。

Figure 9-2 Password Field with the Set Text

http://www.javafxfan.net/attachment/Mon_1111/3_2_b294dc5e2b336db.png?965

Description of "Figure 9-2 Password Field with the Set Text"

密码框中的值也可以用getText 方法获得。您可以在您的应用程序处理这个值，并设置适当的身份验证逻辑。

处理Password的值

花点时间看Example 9-2 中的代码，它实现了一个密码框。

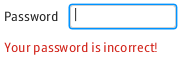
Example 9-2 Implementing the Authentication Logic

[复制代码](javascript:)

1. final Label message = new Label("");
2. VBox vb = new VBox();
3. vb.setPadding(new Insets(10, 0, 0, 10));
4. vb.setSpacing(10);
5. HBox hb = new HBox();
6. hb.setSpacing(10);
7. hb.setAlignment(Pos.CENTER\_LEFT);
8. Label label = new Label("Password");
9. final PasswordField pb = new PasswordField();
10. pb.setOnAction(new EventHandler<ActionEvent>() {
11. @Override public void handle(ActionEvent e) {
12. if (!pb.getText().equals("T2f$Ay!")) {
13. message.setText("Your password is incorrect!");
14. message.setTextFill(Color.rgb(210, 39, 30));
15. } else {
16. message.setText("Your password has been confirmed");
17. message.setTextFill(Color.rgb(21, 117, 84));
18. }
19. pb.setText(" ");
20. }
21. });
22. hb.getChildren().addAll(label, pb);
23. vb.getChildren().addAll(hb, message);

这里通过setOnAction 方法为密码框定义了身份验证逻辑。该方法调用的时间是提交密码后，它新建了一个EventHandler 对象来处理键入的值。如果输入值和要求值不同，相应的信息就以红色显示出来，见Figure 9-3 .

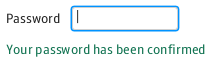
Figure 9-3 Password is Incorrect



Description of "Figure 9-3 Password is Incorrect"

如果输入值符合要求，确认信息就显示出来，见Figure 9-4 .

Figure 9-4 Password is Correct



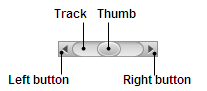
Description of "Figure 9-4 Password is Correct"

出于安全考虑，最佳实践是键入文本后清空密码框。在Example 9-2 中，身份验证后会将密码框置空 。

# 第十六回 JavaFX2.0滚动栏ScrollBar

ScrollBar 类使你可以创建滚动栏面板和视图。Figure 10-1 展示了滚动栏的三部分：滚动条（thumb），左右按钮（或上下按钮），可滚动区。

Figure 10-1 Elements of the scroll bar



**创建Scroll Bar**

花点时间看下 Example 10-1 中的代码块。

Example 10-1 Simple Scroll Bar

ScrollBar sc = new ScrollBar();

sc.setMin(0);

sc.setMax(100);

sc.setValue(50);

setMin 和setMax 方法定义了滚动栏呈现的最小值和最大值。当用户移动滚动条时，它的值会改变。在 Example 10-1 中，它的值是50，所以应用启动后滚动条位于滚动栏的中间。默认滚动栏是水平方向的，你要用setOrientation 方法把它设为垂直方向。

用户可以点击左右按钮（垂直滚动栏是上下按钮）按照增长单位来改变值。UNIT\_INCREMENT属性指定了当点击按钮时滚动条调整的幅度。也可以点击滚动区来大幅改变。 BLOCK\_INCREMENT属性定义了当点击滚动区时改变的幅度。

在您的应用程序，您可以使用几个滚动条来滚动超出可用空间边界的图形内容。

**在应用中使用Scroll Bar**

实战体验下。  Example 10-2 中创建的应用实现了一个可滚动的视窗来查看图片。这个应用的目的是让用户能够查看垂直盒子（vertical box）中的内容，它们比视窗好高度要长。

Example 10-2 Scrolling Through Multiple Images

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Orientation;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.ScrollBar;
8. import javafx.scene.effect.DropShadow;
9. import javafx.scene.image.Image;
10. import javafx.scene.image.ImageView;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.paint.Color;
13. import javafx.stage.Stage;
14. public class Main extends Application {
15. final ScrollBar sc = new ScrollBar();
16. final Image[] images = new Image[5];
17. final ImageView[] pics = new ImageView[5];
18. final VBox vb = new VBox();
19. DropShadow shadow = new DropShadow();
20. @Override
21. public void start(Stage stage) {
22. Group root = new Group();
23. Scene scene = new Scene(root, 180, 180);
24. scene.setFill(Color.BLACK);
25. stage.setScene(scene);
26. stage.setTitle("Scrollbar");
27. root.getChildren().addAll(vb, sc);
28. shadow.setColor(Color.GREY);
29. shadow.setOffsetX(2);
30. shadow.setOffsetY(2);
31. vb.setLayoutX(5);
32. vb.setSpacing(10);
33. sc.setLayoutX(scene.getWidth()-sc.getWidth());
34. sc.setMin(0);
35. sc.setOrientation(Orientation.VERTICAL);
36. sc.setPrefHeight(180);
37. sc.setMax(360);
38. for (int i = 0; i < 5; i++) {
39. final Image image = images[i] =
40. new Image(getClass().getResourceAsStream(
41. "fw" +(i+1)+ ".jpg")
42. );
43. final ImageView pic = pics[i] =
44. new ImageView(images[i]);
45. pic.setEffect(shadow);
46. vb.getChildren().add(pics[i]);
47. }
48. sc.valueProperty().addListener(new ChangeListener<Number>() {
49. public void changed(ObservableValue<? extends Number> ov,
50. Number old\_val, Number new\_val) {
51. vb.setLayoutY(-new\_val.doubleValue());
52. }
53. });
54. stage.show();
55. }
56. public static void main(String[] args) {
57. launch(args);
58. }
59. }

代码的第一行向场景中增加了一个带有图片的垂直盒子和一个滚动栏。

垂直盒子的Y坐标会随着滚动条 VALUE 属性的变化而变化。 所以每次滚动条移动时（或者按钮、滚动区）被点击时，垂直盒子就跟着动。见Example 10-3 .

Example 10-3 Implementing the Scrolling of the Vertical Box

[复制代码](javascript:)

1. sc.valueProperty().addListener(new ChangeListener<Number>() {
2. public void changed(ObservableValue<? extends Number> ov,
3. Number old\_val, Number new\_val) {
4. vb.setLayoutY(-new\_val.doubleValue());
5. }
6. });

编译并运行应用，效果如Figure 10-2 .

Figure 10-2 Scroll Bar Sample



这个应用是ScrollBar 类的典型应用。你也可以在场景内用定制化的该类创建滚动区域。因为对于每个UI控件和节点（node，有Node这个类的）,滚动栏都可以从默认实现中修改外观。

# 第十七回 JavaFX2.0 滚动窗Scroll Pane

滚动窗为UI元素提供了一个可以滚动查看的视图。该控件让用户可以通过移动视口或者滚动条来查看。Figure 11-1 是一个添加了图片的默认设置的滚动窗。

Figure 11-1 Scroll Pane



**创建Scroll Pane**

Example 11-1 演示了如何在应用中创建滚动窗。

Example 11-1 Using a Scroll Pane to View an Image

Image roses = new Image(getClass().getResourceAsStream("roses.jpg"));

ScrollPane sp = new ScrollPane();

sp.setNode(new ImageView(roses));

setNode 方法定义了滚动窗的结点是什么内容，可以只指定一个结点。要创建具有多个组件的滚动窗，得用布局容器或者Group类。可以为 setPannable 方法设置true值，这样当点击和移动鼠标时能预览图像，滚动条的位置也会相应改变。

**为Scroll Pane设置滚动条策略**

ScrollPane 类提供了一种策略来决定何时显示滚动条：总是、从不、需要时( always, never,needed）。分别使用 setHbarPolicy 和setVbarPolicy 方法为水平滚动条和垂直滚动条指定策略。这样，Example 11-2 中的垂直滚动条会一直显示，而水平的不会显示。

Example 11-2 Setting the Horizontal and Vertical Scroll Bar Policies

sp.setHbarPolicy(ScrollBarPolicy.NEVER);

sp.setVbarPolicy(ScrollBarPolicy.ALWAYS);

结果是只能垂直地滚动图片，见 Figure 11-2 .

Figure 11-2 Disabling the Horizontal Scroll Bar



**改变Scroll Pane中组件的大小**

设计UI接口时可能需要能够改变组件的大小已让它们适合滚动窗的宽和高。为 setFitToWidth 或 setFitToHeight 方法设置true 值来匹配特定的方向。

Figure 11-3 中的滚动窗包含单选按钮、文本框和密码框。这些内容的大小超过了滚动窗预先定义的尺寸，所以垂直滚动条显示了出来。然而，由于setFitToWidth 方法被设为true，视窗宽度会伸缩使水平方向无滚动条。

Figure 11-3 Fitting the Width of the Scroll Pane



默认FIT\_TO\_WIDTH 和 FIT\_TO\_HEIGHT 属性都是false，可改变大小的内容也保持原始大小。从上面代码移除 setFitToWidth 方法后，显示如Figure 11-4 .

Figure 11-4 Default Properties for Fitting the Content



ScrollPane 类可以取回和设置内容在水平和垂直方向的当前、最小、最大值。学习怎么使用吧

**使用Scroll Pane的样例程序**

Example 11-3 使用滚动窗显示一个带有图片的垂直盒子。ScrollPane 类的VVALUE属性帮助辨识当然显示的图片并显示它的名称。

Example 11-3 Using a Scroll Pane to View Images

[复制代码](javascript:)

1. package scrollpanesample;
2. import javafx.application.Application;
3. import javafx.beans.value.ChangeListener;
4. import javafx.beans.value.ObservableValue;
5. import javafx.scene.Scene;
6. import javafx.scene.control.Label;
7. import javafx.scene.control.ScrollPane;
8. import javafx.scene.image.Image;
9. import javafx.scene.image.ImageView;
10. import javafx.scene.layout.Priority;
11. import javafx.scene.layout.VBox;
12. import javafx.stage.Stage;
13. public class Main extends Application {
14. final ScrollPane sp = new ScrollPane();
15. final Image[] images = new Image[5];
16. final ImageView[] pics = new ImageView[5];
17. final VBox vb = new VBox();
18. final Label fileName = new Label();
19. final String [] imageNames = new String [] {"fw1.jpg", "fw2.jpg",
20. "fw3.jpg", "fw4.jpg", "fw5.jpg"};
21. @Override
22. public void start(Stage stage) {
23. VBox box = new VBox();
24. Scene scene = new Scene(box, 180, 180);
25. stage.setScene(scene);
26. stage.setTitle("Scroll Pane");
27. box.getChildren().addAll(sp, fileName);
28. VBox.setVgrow(sp, Priority.ALWAYS);
29. fileName.setLayoutX(30);
30. fileName.setLayoutY(160);
31. for (int i = 0; i < 5; i++) {
32. images[i] = new Image(getClass().getResourceAsStream(imageNames[i]));
33. pics[i] = new ImageView(images[i]);
34. pics[i].setFitWidth(100);
35. pics[i].setPreserveRatio(true);
36. vb.getChildren().add(pics[i]);
37. }
38. sp.setVmax(440);
39. sp.setPrefSize(115, 150);
40. sp.setContent(vb);
41. sp.vvalueProperty().addListener(new ChangeListener<Number>() {
42. public void changed(ObservableValue<? extends Number> ov,
43. Number old\_val, Number new\_val) {
44. fileName.setText(imageNames[(new\_val.intValue() - 1)/100]);
45. }
46. });
47. stage.show();
48. }
49. public static void main(String[] args) {
50. launch(args);
51. }
52. }

Figure 11-5 是编译并运行的效果。

Figure 11-5 Scrolling Images



垂直滚动条的最大值等于垂直盒子的高度。 Example 11-4 中的代码块显示了当前显示图片的名称。

Example 11-4 Tracking the Change of the Scroll Pane's Vertical Value

sp.vvalueProperty().addListener(new ChangeListener<Number>() {

    public void changed(ObservableValue<? extends Number> ov,

        Number old\_val, Number new\_val) {

            fileName.setText(imageNames[(new\_val.intValue() - 1)/100]);

        }

});

ImageView对象限制了图片高度是100点。所以， new\_val.intValue() - 1 除以100的结果给出了当前图片的索引。

可以在应用中改变水平滚动条的最小值和最大值来动态更新用户接口。

# 第十八回 JavaFX2.0 列表框ListView

ListView类呈现一个可滚动的项目列表。 Figure 12-1 展示了一个住宿类型列表。

Figure 12-1 Simple List View



可以通过setItems 方法定义项目来产生列表。也可以使用setCellFactory 方法为列表中项目创建一个视图。

**创建List View**

Example 12-1中的代码块实现了Figure 12-1 中带有String 类项目的列表。

Example 12-1 Creating a List View Control

ListView<String> list = new ListView<String>();

ObservableList<String> items =FXCollections.observableArrayList (

    "Single", "Double", "Suite", "Family App");

list.setItems(items);

使用setPrefHeight 和setPrefWidth 方法来改变列表视图控件的大小和高度。Example 12-2限制垂直列表具有100点宽度和70点高度，效果见Figure 12-2。

Example 12-2 Setting Height and Width for a List View

list.setPrefWidth(100);

list.setPrefHeight(70);

Figure 12-2 Resized Vertical List



Description of "Figure 12-2 Resized Vertical List"

要 将ListView对象设置为水平方向的可以通过将方向属性设为 Orientation.HORIZONTAL ，这样做即可：list.setOrientation(Orientation.HORIZONTAL) 。 Figure 12-1 和Figure 12-3 中的水平列表具有相同的项目。

Figure 12-3 Horizontal List View Control

http://www.javafxfan.net/attachment/Mon_1111/3_2_fa6f5e24ae8c622.png?3567

Description of "Figure 12-3 Horizontal List View Control"

可以用下面的组合方法获得每个项目当前的状态：

getSelectionModel().selectedIndexProperty() – 返回当前被选中项目的索引。

getSelectionModel().selectedItemProperty() – 返回当前被选中项目。

getFocusModel().getFocusedIndex() – 返回当前有焦点的项目索引。

getFocusModel().getFocusedItem() – 返回当前有焦点的项目。

注意，选中的和有焦点的项目都是只读的，应用启动后是不能为项目指定这些属性的。

前面的代码样例讲解了怎么创建具有文本项目的列表。然而，列表视图控件可以包含Node对象。

**用数据产生List View**

研究下面的代码学习怎么用细胞工厂（cell factory）产生列表项目。Example 12-3 中的应用创建了一个颜色模式列表。

Example 12-3 Creating a Cell Factory

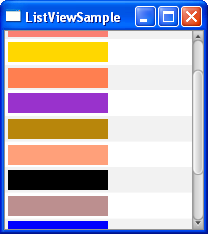
[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.collections.FXCollections;
3. import javafx.collections.ObservableList;
4. import javafx.scene.Scene;
5. import javafx.scene.control.ListCell;
6. import javafx.scene.control.ListView;
7. import javafx.scene.layout.Priority;
8. import javafx.scene.layout.VBox;
9. import javafx.scene.paint.Color;
10. import javafx.scene.shape.Rectangle;
11. import javafx.stage.Stage;
12. import javafx.util.Callback;
13. public class Main extends Application {
14. ListView<String> list = new ListView<String>();
15. ObservableList<String> data = FXCollections.observableArrayList(
16. "chocolate", "salmon", "gold", "coral", "darkorchid",
17. "darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue",
18. "blueviolet", "brown");
19. @Override
20. public void start(Stage stage) {
21. VBox box = new VBox();
22. Scene scene = new Scene(box, 200, 200);
23. stage.setScene(scene);
24. stage.setTitle("ListViewSample");
25. box.getChildren().addAll(list);
26. VBox.setVgrow(list, Priority.ALWAYS);
27. list.setItems(data);
28. list.setCellFactory(new Callback<ListView<String>, ListCell<String>>() {
29. @Override public ListCell<String> call(ListView<String> list) {
30. return new ColorRectCell();
31. }
32. });
33. stage.show();
34. }
36. static class ColorRectCell extends ListCell<String> {
37. @Override
38. public void updateItem(String item, boolean empty) {
39. super.updateItem(item, empty);
40. Rectangle rect = new Rectangle(100, 20);
41. if (item != null) {
42. rect.setFill(Color.web(item));
43. setGraphic(rect);
44. }
45. }
46. }
48. public static void main(String[] args) {
49. launch(args);
50. }
51. }

细胞工厂产生了 ListCell 对象。每个细胞都关联一个单一的数据项目并显示列表中视图的一“行”。细胞呈现的内容通过setGraphic方法可以包含其他控件、文本、形状、图像。该应用中，列表细胞放的是矩形。

Figure 12-4 是该应用编译运行后产生的效果。

Figure 12-4 List of Color Patterns



你可以滚动列表，选择或取消选择项目，也可以扩展应用来用颜色填充文本标签。

**处理选中的List Item**

按照 Example 12-4 修改应用的代码，以使其能处理特定项目被选中的事件。

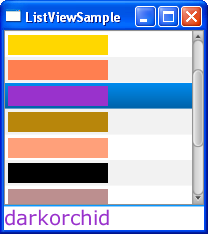
Example 12-4 Processing Events for a List Item

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.collections.FXCollections;
5. import javafx.collections.ObservableList;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Label;
8. import javafx.scene.control.ListCell;
9. import javafx.scene.control.ListView;
10. import javafx.scene.layout.Priority;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.paint.Color;
13. import javafx.scene.shape.Rectangle;
14. import javafx.scene.text.Font;
15. import javafx.stage.Stage;
16. import javafx.util.Callback;
17. public class Main extends Application {
18. ListView<String> list = new ListView<String>();
19. ObservableList<String> data = FXCollections.observableArrayList(
20. "chocolate", "salmon", "gold", "coral", "darkorchid",
21. "darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue",
22. "blueviolet", "brown");
23. final Label label = new Label();
24. @Override
25. public void start(Stage stage) {
26. VBox box = new VBox();
27. Scene scene = new Scene(box, 200, 200);
28. stage.setScene(scene);
29. stage.setTitle("ListViewSample");
30. box.getChildren().addAll(list, label);
31. VBox.setVgrow(list, Priority.ALWAYS);
32. label.setLayoutX(10);
33. label.setLayoutY(115);
34. label.setFont(Font.font("Verdana", 20));
35. list.setItems(data);
36. list.setCellFactory(new Callback<ListView<String>, ListCell<String>>() {
37. @Override public ListCell<String> call(ListView<String> list) {
38. return new ColorRectCell();
39. }
40. });
41. list.getSelectionModel().selectedItemProperty().addListener(
42. new ChangeListener<String>() {
43. public void changed(ObservableValue<? extends String> ov,
44. String old\_val, String new\_val) {
45. label.setText(new\_val);
46. label.setTextFill(Color.web(new\_val));
47. }
48. });
49. stage.show();
50. }
52. static class ColorRectCell extends ListCell<String> {
53. @Override
54. public void updateItem(String item, boolean empty) {
55. super.updateItem(item, empty);
56. Rectangle rect = new Rectangle(100, 20);
57. if (item != null) {
58. rect.setFill(Color.web(item));
59. setGraphic(rect);
60. }
61. }
62. }
64. public static void main(String[] args) {
65. launch(args);
66. }
67. }

addListener 方法调用后为 selectedItemProperty 新建了一个ChangeListener<String> 对象来绑定选中项目的改变。比如说，深紫色项目被选中了，标签接收到了"darkorchid"标题并用相应的颜色填充。修改后应用的效果见Figure 12-5 .

Figure 12-5 Selecting a Dark Orchid Color Pattern



# 第二十回 JavaFX2.0 分割线Separator

JavaFX API中的 Separator类呈现的是一条水平或者垂直的分隔线。它是用来分隔应用中用户接口的元素的，并不提供任何行为

创建Separator

Example 14-1 中的代码块创建了一条水平的一条垂直的分隔线。

Example 14-1 Vertical and Horizontal Separators

[复制代码](javascript:)

1. //Horizontal separator
2. Separator separator1 = new Separator();
3. //Vertical separator
4. Separator separator2 = new Separator();
5. separator2.setOrientation(Orientation.VERTICAL);

Separator 类继承了Node类，所以，分隔线就继承了 Node类的所有常量和变量。

一般分隔线是用来分隔UI控件成组的。研究下Example 14-2 中的代码，它将春天的月份和夏天的分开了

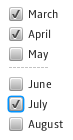
Example 14-2 Using a Separator Between Checkbox Categories

[复制代码](javascript:)

1. final String[] names = new String[]{"March", "April", "May",
2. "June", "July", "August"};
3. final CheckBox[] cbs = new CheckBox[names.length];
4. final Separator separator = new Separator();
5. final VBox vbox = new VBox();
6. for (int i = 0; i < names.length; i++) {
7. cbs[i] = new CheckBox(names[i]);
8. }
10. separator.setMaxWidth(40);
11. separator.setAlignment(Pos.CENTER\_LEFT);
12. vbox.getChildren().addAll(cbs);
13. vbox.setSpacing(5);
14. vbox.getChildren().add(3, separator);

运行效果见Figure 14-1 .

Figure 14-1 Checkboxes and a Separator



分隔线会占据分配给它的全部水平或垂直空间。setMaxWidth方法用来指定一个特定的宽度， setValignment 方法指定了分隔线在其分配的布局空间内的垂直位置。类似地，也可以使用setHalignment 方法来设置其水平位置。

在Example 14-2 中，分隔线被使用专用的方法add(index, node)加入了垂直盒子。可以使用这种方法在创建UI后或者动态改变UI时包括分隔线。

**为UI添加分隔线**

前面提到，分隔线可以将UI控件分组。可以使用它们构建用户接口，就像下面这样显示天气预报Figure 14-2 .

Figure 14-2 Structuring Weather Forecast Data with Separators



Description of "Figure 14-2 Structuring Weather Forecast Data with Separators"

Figure 14-2 中的效果是，分隔线用来分开Label和 ImageView对象。研究下下面的代码 Example 14-3 .

Example 14-3 Using Separators in a Weather Forecast Application

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.geometry.Insets;
3. import javafx.geometry.Orientation;
4. import javafx.geometry.VPos;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.\*;
8. import javafx.scene.image.Image;
9. import javafx.scene.image.ImageView;
10. import javafx.scene.layout.GridPane;
11. import javafx.scene.text.Font;
12. import javafx.stage.Stage;
13. public class Main extends Application {
14. Label caption = new Label("Weather Forecast");
15. Label friday = new Label("Friday");
16. Label saturday = new Label("Saturday");
17. Label sunday = new Label("Sunday");
18. @Override
19. public void start(Stage stage) {
20. Group root = new Group();
21. Scene scene = new Scene(root, 500, 300);
22. stage.setScene(scene);
23. stage.setTitle("Separator Sample");
24. GridPane grid = new GridPane();
25. grid.setPadding(new Insets(10, 10, 10, 10));
26. grid.setVgap(2);
27. grid.setHgap(5);
28. scene.setRoot(grid);
29. Image cloudImage = new Image(getClass().getResourceAsStream("cloud.jpg"));
30. Image sunImage = new Image(getClass().getResourceAsStream("sun.jpg"));
31. caption.setFont(Font.font("Verdana", 20));
32. GridPane.setConstraints(caption, 0, 0);
33. GridPane.setColumnSpan(caption, 8);
34. grid.getChildren().add(caption);
35. final Separator sepHor = new Separator();
36. sepHor.setValignment(VPos.CENTER);
37. GridPane.setConstraints(sepHor, 0, 1);
38. GridPane.setColumnSpan(sepHor, 7);
39. grid.getChildren().add(sepHor);
40. friday.setFont(Font.font("Verdana", 18));
41. GridPane.setConstraints(friday, 0, 2);
42. GridPane.setColumnSpan(friday, 2);
43. grid.getChildren().add(friday);
44. final Separator sepVert1 = new Separator();
45. sepVert1.setOrientation(Orientation.VERTICAL);
46. sepVert1.setValignment(VPos.CENTER);
47. sepVert1.setPrefHeight(80);
48. GridPane.setConstraints(sepVert1, 2, 2);
49. GridPane.setRowSpan(sepVert1, 2);
50. grid.getChildren().add(sepVert1);
51. saturday.setFont(Font.font("Verdana", 18));
52. GridPane.setConstraints(saturday, 3, 2);
53. GridPane.setColumnSpan(saturday, 2);
54. grid.getChildren().add(saturday);
55. final Separator sepVert2 = new Separator();
56. sepVert2.setOrientation(Orientation.VERTICAL);
57. sepVert2.setValignment(VPos.CENTER);
58. sepVert2.setPrefHeight(80);
59. GridPane.setConstraints(sepVert2, 5, 2);
60. GridPane.setRowSpan(sepVert2, 2);
61. grid.getChildren().add(sepVert2);
62. sunday.setFont(Font.font("Verdana", 18));
63. GridPane.setConstraints(sunday, 6, 2);
64. GridPane.setColumnSpan(sunday, 2);
65. grid.getChildren().add(sunday);
66. final ImageView cloud = new ImageView(cloudImage);
67. GridPane.setConstraints(cloud, 0, 3);
68. grid.getChildren().add(cloud);
69. final Label t1 = new Label("16");
70. t1.setFont(Font.font("Verdana", 20));
71. GridPane.setConstraints(t1, 1, 3);
72. grid.getChildren().add(t1);
73. final ImageView sun1 = new ImageView(sunImage);
74. GridPane.setConstraints(sun1, 3, 3);
75. grid.getChildren().add(sun1);
76. final Label t2 = new Label("18");
77. t2.setFont(Font.font("Verdana", 20));
78. GridPane.setConstraints(t2, 4, 3);
79. grid.getChildren().add(t2);
80. final ImageView sun2 = new ImageView(sunImage);
81. GridPane.setConstraints(sun2, 6, 3);
82. grid.getChildren().add(sun2);
83. final Label t3 = new Label("20");
84. t3.setFont(Font.font("Verdana", 20));
85. GridPane.setConstraints(t3, 7, 3);
86. grid.getChildren().add(t3);
87. stage.show();
88. }
89. public static void main(String[] args) {
90. launch(args);
91. }
92. }

该应用使用了水平和垂直分隔线，并在 GridPane容器中跨了行和列。 也可以为分隔线设置优先的长度(水平分隔线的宽度和垂直分隔线的高度)这样界面大小改变时就能动态伸缩了。可以为Separator 对象应用CSS来改变分隔线的视觉外观。

**美化Separator**

要为 Example 14-3 中的分隔线应用相同的效果，可以创建CSS文件(例如controlStyle.css) 并保存在应用主类的相同包中。Example 14-4 是一些可以加入到controlStyle文件的CSS类。

Example 14-4 Using CSS Classes to Style Separators

/\*controlStyle.css \*/

.separator{

    -fx-background-color: #e79423;

    -fx-background-radius: 2;

}

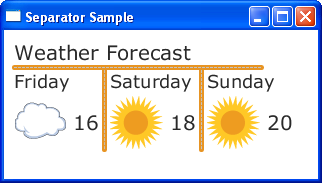
可以使用 Scene 类的getStylesheets 方法来应用分隔线风格，见 Example 14-5 .

Example 14-5 Enabling Style Sheets in a JavaFX Application

scene.getStylesheets().add("separatorsample/controlStyle.css");

Figure 14-3 是应用修改后运行时分隔线的效果。

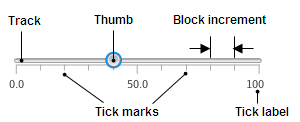
Figure 14-3 Styled Separators



# 第二十一回 JavaFX2.0 滑动条Slider

Slider 类呈现一个控件来显示和回应一个范围的数值。该控件包括一个轨道和一个可以拖动的滑标，也包含刻度和刻度标记来指示数值。Figure 15-1 展示了一个滑动条并指明了其主要元素。

Figure 15-1 Elements of a Slider



Description of "Figure 15-1 Elements of a Slider"

**创建Slider**

花点时间看下 Example 15-1 中的代码，它产生了一个Figure 15-1 展示的滑动条。

Example 15-1 Creating a Slider

[复制代码](javascript:)

1. Slider slider = new Slider();
2. slider.setMin(0);
3. slider.setMax(100);
4. slider.setValue(40);
5. slider.setShowTickLabels(true);
6. slider.setShowTickMarks(true);
7. slider.setMajorTickUnit(50);
8. slider.setMinorTickCount(5);
9. slider.setBlockIncrement(10);

setMin 和 setMax 方法分布定义了滑动条呈现的最小值和最大值。setValue方法指定了滑动条的当前值，当然必须在最小值和最大值之间。 使用该方法来定义应用启动后滑标的位置。

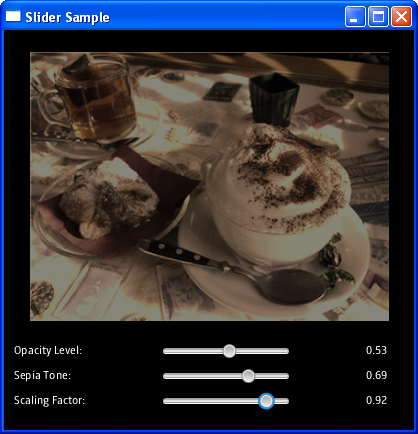
两 个布尔方法 setShowTickMarks 和 setShowTickLabels 定义了滑动条的视觉外观。在 Example 15-1 中，刻度和数值显示了。另外，大刻度之间的单元距离设置为50，大刻度直接的小刻度数量定义为5。把 setSnapToTicks方法设为 true 来保持滑标总和刻度对其。

setBlockIncrement方法定义了用户点击轨道时滑标移动的距离。 Example 15-1 中该值是10，就是说当用户点击轨道时，滑标会向点击方向移动10单位。

**在图形应用中使用滑动条**

现在看一下Figure 15-2。 该应用使用了3个滑动条来白哦几图片的属性。每个滑动条调整一个特定的视觉特点：透明度、褐色调、伸缩比例。

Figure 15-2 Three Sliders



Description of "Figure 15-2 Three Sliders "

Example 15-2   shows the source code of this application.

Example 15-2 Slider Sample

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Insets;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Label;
8. import javafx.scene.control.Slider;
9. import javafx.scene.effect.SepiaTone;
10. import javafx.scene.image.Image;
11. import javafx.scene.image.ImageView;
12. import javafx.scene.layout.GridPane;
13. import javafx.scene.paint.Color;
14. import javafx.stage.Stage;
15. public class Main extends Application {
16. final Slider opacityLevel = new Slider(0, 1, 1);
17. final Slider sepiaTone = new Slider(0, 1, 1);
18. final Slider scaling = new Slider (0.5, 1, 1);
19. final Image image  = new Image(getClass().getResourceAsStream(
20. "cappuccino.jpg")
21. );
22. final Label opacityCaption = new Label("Opacity Level:");
23. final Label sepiaCaption = new Label("Sepia Tone:");
24. final Label scalingCaption = new Label("Scaling Factor:");
25. final Label opacityValue = new Label(
26. Double.toString(opacityLevel.getValue()));
27. final Label sepiaValue = new Label(
28. Double.toString(sepiaTone.getValue()));
29. final Label scalingValue = new Label(
30. Double.toString(scaling.getValue()));
31. final static Color textColor = Color.WHITE;
32. final static SepiaTone sepiaEffect = new SepiaTone();
33. @Override
34. public void start(Stage stage) {
35. Group root = new Group();
36. Scene scene = new Scene(root, 600, 400);
37. stage.setScene(scene);
38. stage.setTitle("Slider Sample");
39. scene.setFill(Color.BLACK);
40. GridPane grid = new GridPane();
41. grid.setPadding(new Insets(10, 10, 10, 10));
42. grid.setVgap(10);
43. grid.setHgap(70);
44. final ImageView cappuccino = new ImageView (image);
45. cappuccino.setEffect(sepiaEffect);
46. GridPane.setConstraints(cappuccino, 0, 0);
47. GridPane.setColumnSpan(cappuccino, 3);
48. grid.getChildren().add(cappuccino);
49. scene.setRoot(grid);
50. opacityCaption.setTextFill(textColor);
51. GridPane.setConstraints(opacityCaption, 0, 1);
52. grid.getChildren().add(opacityCaption);
54. opacityLevel.valueProperty().addListener(new ChangeListener<Number>() {
55. public void changed(ObservableValue<? extends Number> ov,
56. Number old\_val, Number new\_val) {
57. cappuccino.setOpacity(new\_val.doubleValue());
58. opacityValue.setText(String.format("%.2f", new\_val));
59. }
60. });
61. GridPane.setConstraints(opacityLevel, 1, 1);
62. grid.getChildren().add(opacityLevel);
63. opacityValue.setTextFill(textColor);
64. GridPane.setConstraints(opacityValue, 2, 1);
65. grid.getChildren().add(opacityValue);
66. sepiaCaption.setTextFill(textColor);
67. GridPane.setConstraints(sepiaCaption, 0, 2);
68. grid.getChildren().add(sepiaCaption);
69. sepiaTone.valueProperty().addListener(new ChangeListener<Number>() {
70. public void changed(ObservableValue<? extends Number> ov,
71. Number old\_val, Number new\_val) {
72. sepiaEffect.setLevel(new\_val.doubleValue());
73. sepiaValue.setText(String.format("%.2f", new\_val));
74. }
75. });
76. GridPane.setConstraints(sepiaTone, 1, 2);
77. grid.getChildren().add(sepiaTone);
78. sepiaValue.setTextFill(textColor);
79. GridPane.setConstraints(sepiaValue, 2, 2);
80. grid.getChildren().add(sepiaValue);
81. scalingCaption.setTextFill(textColor);
82. GridPane.setConstraints(scalingCaption, 0, 3);
83. grid.getChildren().add(scalingCaption);
84. scaling.valueProperty().addListener(new ChangeListener<Number>() {
85. public void changed(ObservableValue<? extends Number> ov,
86. Number old\_val, Number new\_val) {
87. cappuccino.setScaleX(new\_val.doubleValue());
88. cappuccino.setScaleY(new\_val.doubleValue());
89. scalingValue.setText(String.format("%.2f", new\_val));
90. }
91. });
92. GridPane.setConstraints(scaling, 1, 3);
93. grid.getChildren().add(scaling);
94. scalingValue.setTextFill(textColor);
95. GridPane.setConstraints(scalingValue, 2, 3);
96. grid.getChildren().add(scalingValue);
97. stage.show();
98. }
99. public static void main(String[] args) {
100. launch(args);
101. }
102. }

ImageView 对象的不透明度由第一个滑动条改变，叫opacityLevel。SepiaTone 效果的改变由sepiaTone滑动条控制。第三个滑动条定义了放大倍数，调用的是 setScaleX 和 setScaleY 方法。

Example 15-3   中的代码是一个把Slider类的 getValue方法返回的双精度值转换为 String ，也应用了格式来显示滑动条的值：点后两位小数的浮点类型。

Example 15-3 Formatting the Rendered Slider's Value

scalingValue.setText((Double.toString(value)).format("%.2f", value));

下一步就是为它应用视效或CSS风格来改善外观和感觉。

# 第二十二回 JavaFX2.0 进度条和进度指示器

ProgressIndicator及其直接子类 ProgressBar提供了指示特定任务正在运行并检测其完成进度的能力。 不过ProgressBar类用来显示一个显示进度完成的条，而 ProgressIndicator类则是将进度动态地显示在一个饼图里。见 Figure 16-1 .

Figure 16-1 Progress Bar and Progress Indicator

http://www.javafxfan.net/attachment/Mon_1111/3_2_676c604efe8ba49.png?1938

Description of "Figure 16-1 Progress Bar and Progress Indicator"

**创建进度控件**

Example 16-1 中的代码能够在JavaFX应用中插入一个进度控件。

Example 16-1 Implementing the Progress Bar and Progress Indicator

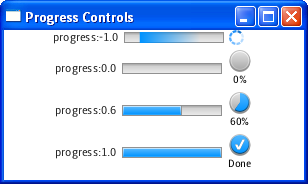
[复制代码](javascript:)

1. ProgressBar pb = new ProgressBar(0.6);
2. ProgressIndicator pi = new ProgressIndicator(0.6);

也 可以使用空构造方法创建进度控件而不指定参数。这时候，可以使用setProgress方法为它分配值。 另一个初始化进度控件的方法是使用 ProgressBarBuilder 类，该类包括诸如build和 progress 之类的方法。可以查看API文档去了解更多。

有时候应用并不能缺地in个任务的完成时间，这时进度控件就保持在非确定模式中直到可以确定。Figure 16-2 中是依赖于不同进度变量值的进度控件。

Figure 16-2 Various States of Progress Controls



Description of "Figure 16-2 Various States of Progress Controls"

Example 16-2   shows the source code of the application shown in  Figure 16-2 .

Example 16-2 Enabling Different States of Progress Controls

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.geometry.Pos;
3. import javafx.scene.Group;
4. import javafx.scene.Scene;
5. import javafx.scene.control.Label;
6. import javafx.scene.control.ProgressBar;
7. import javafx.scene.control.ProgressIndicator;
8. import javafx.scene.layout.HBox;
9. import javafx.scene.layout.VBox;
10. import javafx.stage.Stage;
11. public class Main extends Application
12. { final Float[] values = new Float[] {-1.0f, 0f, 0.6f, 1.0f};
13. final Label [] labels = new Label[values.length];
14. final ProgressBar[] pbs = new ProgressBar[values.length];
15. final ProgressIndicator[] pins = new ProgressIndicator[values.length];
16. final HBox hbs [] = new HBox [values.length];
17. @Override
18. public void start(Stage stage)
19. { Group root = new Group();
20. Scene scene = new Scene(root, 300, 150);
21. scene.getStylesheets().add("progresssample/Style.css");
22. stage.setScene(scene);
23. stage.setTitle("Progress Controls");
24. for (int i = 0; i < values.length; i++)
25. { final Label label = labels[i] = new Label();
26. label.setText("progress:" + values[i]);
27. final ProgressBar pb = pbs[i] = new ProgressBar();
28. pb.setProgress(values[i]);
29. final ProgressIndicator pin = pins[i] = new ProgressIndicator();
30. pin.setProgress(values[i]);
31. final HBox hb = hbs[i] = new HBox();
32. hb.setSpacing(5);
33. hb.setAlignment(Pos.CENTER);
34. hb.getChildren().addAll(label, pb, pin);
35. }
36. final VBox vb = new VBox();
37. vb.setSpacing(5);
38. vb.getChildren().addAll(hbs);
39. scene.setRoot(vb);
40. stage.show();
41. }
42. public static void main(String[] args)
43. { launch(args); }
44. }

一个在0和1之间的正数用来指示进程的百分比。 比如，0.4代表40%。而一个负数表示进度在非确定模式。用方法isIndeterminate 可以检查进度控件是否在非确定模式中。

**在界面上指示进度**

Figure 16-2 曾经简单的显示了进度控件的所以可能状态。实际应用中，进度值可以通过其他UI元素的值获得。

研究 Example 16-3 中的代码学习如何为基于滑标位置的进度条和指示器设置值。

Example 16-3 Receiving the Progress Value from a Slider

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Pos;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.ProgressBar;
8. import javafx.scene.control.ProgressIndicator;
9. import javafx.scene.control.Slider;
10. import javafx.scene.layout.HBox;
11. import javafx.stage.Stage;
12. public class Main  extends Application {
13. @Override
14. public void start(Stage stage) {
15. Group root = new Group();
16. Scene scene = new Scene(root);
17. stage.setScene(scene);
18. stage.setTitle("Progress Controls");
19. final Slider slider = new Slider();
20. slider.setMin(0);
21. slider.setMax(50);
23. final ProgressBar pb = new ProgressBar(0);
24. final ProgressIndicator pi = new ProgressIndicator(0);
25. slider.valueProperty().addListener(new ChangeListener<Number>() {
26. public void changed(ObservableValue<? extends Number> ov,
27. Number old\_val, Number new\_val) {
28. pb.setProgress(new\_val.doubleValue()/50);
29. pi.setProgress(new\_val.doubleValue()/50);
30. }
31. });
32. final HBox hb = new HBox();
33. hb.setSpacing(5);
34. hb.setAlignment(Pos.CENTER);
35. hb.getChildren().addAll(slider, pb, pi);
36. scene.setRoot(hb);
37. stage.show();
38. }
39. public static void main(String[] args) {
40. launch(args);
41. }
42. }

编译运行效果见 Figure 16-3 .

Figure 16-3 Indicating the Progress Set by a Slider



Description of "Figure 16-3 Indicating the Progress Set by a Slider"

一个 ChangeListener 对象决定了是否滑动条在动并计算进度条和指示器的值，所以进度控件值的范围是0.0到1.0.

# 第二十三回 JavaFX2.0 超链接Hyperlink

Hyperlink 类呈现的是Labeled 控件的另一种形式，主要用来格式化超链接文本。Figure 17-1 显示了默认超链接的三个实现状态。

Figure 17-1 Three States of a Hyperlink Control



**创建Hyperlink**

这些代码将产生上面的效果 Example 17-1 .

Example 17-1 Typical Hyperlink

Hyperlink link = new Hyperlink();

link.setText("http://example.com");

link.setOnAction(new EventHandler<ActionEvent>() {

    @Override

    public void handle(ActionEvent e) {

        System.out.println("This link is clicked");

    }

});

setText 实例方法定义了超链接的文本。由于 Hyperlink 类继承了Labeled 类，所以可以为超链接指定特定的字体和内容。setOnAction 方法定义了任何时候点击超链接的行为，和Button控件很像。在 Example 17-1 中，这种行为只是用来打印一个字符串。实际上，它可以实现更多更复杂的认为。

**连接到本地内容**

Figure 17-2 中的应用显示了本地的图片。

Figure 17-2 Viewing Images



Description of "Figure 17-2 Viewing Images"

看下它的代码 Example 17-2 .

Example 17-2 Using Hyperlinks to VIew Images

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.scene.\*;
5. import javafx.scene.control.\*;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.layout.VBox;
9. import javafx.stage.Stage;
10. public class Main extends Application {
11. final static String[] imageFiles = new String[]{
12. "product.png",
13. "education.png",
14. "partners.png",
15. "support.png"
16. };
17. final static String[] captions = new String[]{
18. "Products",
19. "Education",
20. "Partners",
21. "Support"
22. };
23. final ImageView selectedImage = new ImageView();
24. final ScrollPane list = new ScrollPane();
25. final Hyperlink[] hpls = new Hyperlink[captions.length];
26. final Image[] images = new Image[imageFiles.length];
27. public static void main(String[] args) {
28. Application.launch(args);
29. }
30. @Override
31. public void start(Stage stage) {
32. Scene scene = new Scene(new Group());
33. stage.setTitle("Hyperlink Sample");
34. stage.setWidth(300);
35. stage.setHeight(200);
36. selectedImage.setLayoutX(100);
37. selectedImage.setLayoutY(10);
38. for (int i = 0; i < captions.length; i++) {
39. final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
40. final Image image = images[i] = new Image(
41. getClass().getResourceAsStream(imageFiles[i])
42. );
43. hpl.setOnAction(new EventHandler<ActionEvent>() {
44. @Override
45. public void handle(ActionEvent e) {
46. selectedImage.setImage(image);
47. }
48. });
49. }
50. final Button button = new Button("Refresh links");
51. button.setOnAction(new EventHandler<ActionEvent>() {
52. @Override
53. public void handle(ActionEvent e) {
54. for (int i = 0; i < captions.length; i++) {
55. hpls[i].setVisited(false);
56. selectedImage.setImage(null);
57. }
58. }
59. });
60. VBox vbox = new VBox();
61. vbox.getChildren().addAll(hpls);
62. vbox.getChildren().add(button);
63. vbox.setSpacing(5);
64. ((Group) scene.getRoot()).getChildren().addAll(vbox, selectedImage);
65. stage.setScene(scene);
66. stage.show();
67. }
68. }

该应用在for循环中创建了四个 Hyperlink 对象。点击特点的超链接会调用setOnAction 方法产生不同的行为。这样，images数组中相应的图片就设置给 selectedImage 变量。

当点击一个超链接时，它就成为了访问过的（visited）。可以使用Hyperlink 类的setVisited 方法刷新链接。见Example 17-3 中的代码。

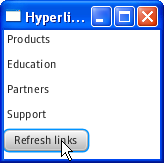
Example 17-3 Refreshing the HyperlInks

[复制代码](javascript:)

1. final Button button = new Button("Refresh links");
2. button.setOnAction(new EventHandler<ActionEvent>() {
3. @Override
4. public void handle(ActionEvent e) {
5. for (int i = 0; i < captions.length; i++) {
6. hpls[i].setVisited(false);
7. selectedImage.setImage(null);
8. }
9. }
10. });

点击Refresh Links按钮就会就把超链接都充值为未访问状态。见Figure 17-3 .

Figure 17-3 Unvisited Hyperlinks



Description of "Figure 17-3 Unvisited Hyperlinks"

由于Hyperlink类继承了 Labeled 类，所以除了文本还可以为其指定图片。下一部分的应用就使用了文本和图片链接并加载远程HTML页面。

**链接到远程内容**

可以在JavaFX应用中显示HTML内容，方法是在场景内绑定WebView 浏览器。WebView 组件提供了网页的基本浏览功能。除此之外，还支持用户交互，如导航和执行JavaScript命令。

研究Example 17-4 中的代码，它创建了带有文本和图像的超链接。点击超链接后，相应的值就作为URL传递给绑定的浏览器。

Example 17-4 Loading Remote Web Pages

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.scene.\*;
5. import javafx.scene.control.\*;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.layout.HBox;
9. import javafx.scene.layout.Priority;
10. import javafx.scene.layout.VBox;
11. import javafx.scene.text.Font;
12. import javafx.scene.web.WebEngine;
13. import javafx.scene.web.WebView;
14. import javafx.stage.Stage;
15. public class Main extends Application {
16. final static String[] imageFiles = new String[]{
17. "product.png",
18. "education.png",
19. "partners.png",
20. "support.png"
21. };
22. final static String[] captions = new String[]{
23. "Products",
24. "Education",
25. "Partners",
26. "Support"
27. };
28. final static String[] urls = new String[]{
29. "http://www.oracle.com/us/products/index.html",
30. "http://education.oracle.com/",
31. "http://www.oracle.com/partners/index.html",
32. "http://www.oracle.com/us/support/index.html"
33. };
35. final ImageView selectedImage = new ImageView();
36. final Hyperlink[] hpls = new Hyperlink[captions.length];
37. final Image[] images = new Image[imageFiles.length];
38. public static void main(String[] args){
39. launch(args);
40. }
41. @Override
42. public void start(Stage stage) {
43. VBox vbox = new VBox();
44. Scene scene = new Scene(vbox);
45. stage.setTitle("Hyperlink Sample");
46. stage.setWidth(570);
47. stage.setHeight(550);
48. selectedImage.setLayoutX(100);
49. selectedImage.setLayoutY(10);
51. final WebView browser = new WebView();
52. final WebEngine webEngine = browser.getEngine();
53. for (int i = 0; i < captions.length; i++) {
54. final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
55. final Image image = images[i] =
56. new Image(getClass().getResourceAsStream(imageFiles[i]));
57. hpl.setGraphic(new ImageView (image));
58. hpl.setFont(Font.font("Arial", 14));
59. final String url = urls[i];
60. hpl.setOnAction(new EventHandler<ActionEvent>() {
61. @Override
62. public void handle(ActionEvent e) {
63. webEngine.load(url);
64. }
65. });
66. }
68. HBox hbox = new HBox();
69. hbox.getChildren().addAll(hpls);
70. vbox.getChildren().addAll(hbox, browser);
71. VBox.setVgrow(browser, Priority.ALWAYS);
73. stage.setScene(scene);
74. stage.show();
75. }
76. }

超链接也在for循环中创建，类似于 Example 17-2 。为超链接设置的行为从urls数组到 WebEngine 对象传递了相应的URL。

编译运行效果如Figure 17-4 .

Figure 17-4 Loading Pages from the Oracle Corporate Site



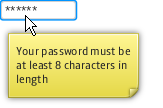
# 第二十四回 JavaFX2.0 提示条ToolTip

Tooltip类产生一个常见的UI控件，一般用来为UI控件添加信息。把鼠标放在控件上提示条就显示出来。任何控件使用 setTooltip方法都能添加提示条。

提示条有2个状态：激活的和显示的。当鼠标放置在控件上时提示条激活，当它显示出来就是“显示的”状态，显示的提示条也是激活的。在提示条激活和显示之间有一些延迟。

带有提示条的密码框见Figure 18-1 .

Figure 18-1 Tooltip Added to a Password Field



Description of "Figure 18-1 Tooltip Added to a Password Field"

**创建Tooltip**

研究 Example 18-1 中的代码，它创建的是上面的应用。

[复制代码](javascript:)

1. Example 18-1 Adding a Tooltip to the Password Field
2. final PasswordField pf = new PasswordField();
3. final Tooltip tooltip = new Tooltip();
4. tooltip.setText(
5. "\nYour password must be\n" +
6. "at least 8 characters in length\n"  +
7. );
8. pf.setTooltip(tooltip);

javafx.scene.control包中的每个控件都具有添加提示条的 setTooltip 方法。可以定义文本，使用Tooltip的构造方法或 setText 方法 。

由于 Tooltip 类继承了Labeled 类，所以不仅可以添加文本，也可以添加图形。 Example 18-2 中的代码块为密码框的提示条添加了图标。

Example 18-2 Adding an Icon to a Tooltip

Image image = new Image(

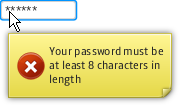
    getClass().getResourceAsStream("warn.png")

);

tooltip.setGraphic(new ImageView(image));

运行效果见 Figure 18-2

Figure 18-2 Tooltip with an Icon



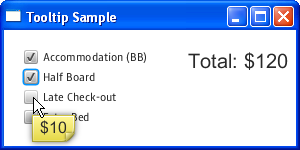
Description of "Figure 18-2 Tooltip with an Icon"

提示条不仅能提供辅助信息，也能呈现数据。

在提示条中呈现数据

Figure 18-3 中的应用使用显示在提示条中的信息来计算酒店住宿的总费用。

Figure 18-3 Calculating Hotel Rates



Description of "Figure 18-3 Calculating Hotel Rates"

Each checkbox is accompanied by a tooltip.每个复选框有一个提示条，每个提示条显示一个特定预定项目的费用。如果用户选择了复选框，相应的值就加到总数中。当然取消选中后也会从总数中减去。

看下该应用的代码Example 18-3 .

Example 18-3 Using Tooltips to Calculate Hotel Rates

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Insets;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.CheckBox;
8. import javafx.scene.control.Label;
9. import javafx.scene.control.Tooltip;
10. import javafx.scene.layout.HBox;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.text.Font;
13. import javafx.stage.Stage;
14. public class Main extends Application {
15. final static String[] rooms = new String[]{
16. "Accommodation (BB)",
17. "Half Board",
18. "Late Check-out",
19. "Extra Bed"
20. };
21. final static Integer[] rates = new Integer[]{
22. 100, 20, 10, 30
23. };
24. final CheckBox[] cbs = new CheckBox[rooms.length];
25. final Label total = new Label("Total: $0");
26. Integer sum = 0;
27. public static void main(String[] args) {
28. launch(args);
29. }
30. @Override
31. public void start(Stage stage) {
32. Scene scene = new Scene(new Group());
33. stage.setTitle("Tooltip Sample");
34. stage.setWidth(300);
35. stage.setHeight(150);
36. total.setFont(new Font("Arial", 20));
38. for (int i = 0; i < rooms.length; i++) {
39. final CheckBox cb = cbs[i] = new CheckBox(rooms[i]);
40. final Integer rate = rates[i];
41. final Tooltip tooltip = new Tooltip("$" + rates[i].toString());
42. tooltip.setFont(new Font("Arial", 16));
43. cb.setTooltip(tooltip);
44. cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
45. public void changed(ObservableValue<? extends Boolean> ov,
46. Boolean old\_val, Boolean new\_val) {
47. if (cb.isSelected()) {
48. sum = sum + rate;
49. } else {
50. sum = sum - rate;
51. }
52. total.setText("Total: $" + sum.toString());
53. }
54. });
55. }
56. VBox vbox = new VBox();
57. vbox.getChildren().addAll(cbs);
58. vbox.setSpacing(5);
59. HBox root = new HBox();
60. root.getChildren().add(vbox);
61. root.getChildren().add(total);
62. root.setSpacing(40);
63. root.setPadding(new Insets(20, 10, 10, 20));
64. ((Group) scene.getRoot()).getChildren().add(root);
65. stage.setScene(scene);
66. stage.show();
67. }
68. }

Example 18-4 中的代码加入到 Example 18-3 中来创建一个提示条并分配了一个文本。项目价格的Integer 值被转化成了String 值。

Example 18-4 Setting the Value for a Tooltip

final Tooltip tooltip = new Tooltip("$" + rates*.toString())*

可以通过使用CSS来改变其外观。

# 第二十五回 JavaFX2.0 Html编辑器

HTMLEditor控件是一个全功能的富文本编辑器。除了基本编辑功能外，它还支持以下特性：

文本格式，包括粗体、斜体、下划线等等

段落设置，比如格式、字体、字号

前景和背景颜色

文本缩进

圆点和数字列表

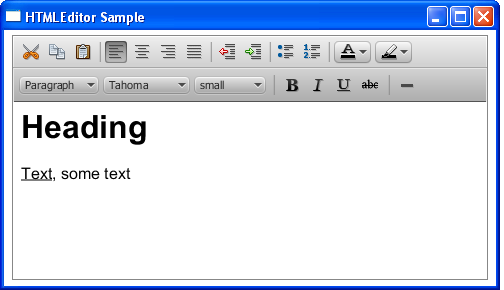
文本对齐

添加水平标尺

复制和粘贴文本块

Figure 19-1 是一个JavaFX应用中的富文本编辑器。

Figure 19-1 HTML Editor



Description of "Figure 19-1 HTML Editor"

HTMLEditor 类呈现编辑内容使用的是HTML字符串形式，比如说，Figure 19-1 中的内容呈现以下字符串："<html><head></head><body

contenteditable="true"><h1>Heading</h1><div><u>Text</u>,

some text</div></body></html> ."

由于HTMLEditor 类继承了Node 类，所以可以为它的实例应用视效和转换。

**添加HTML Editor**

和其他UI控件一样， HTMLEditor组件必须加入场景才能在应用中显示。可以像 Example 19-1 这样直接添加，或者通过布局容器。

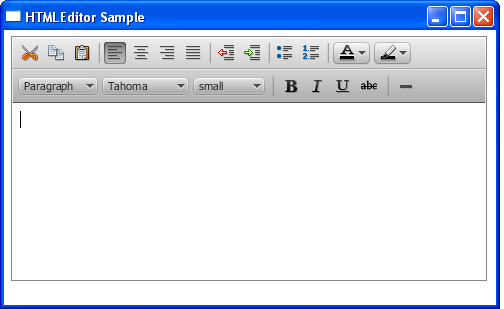
Example 19-1 Adding an HTML Editor to a JavaFX Application

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.scene.Scene;
3. import javafx.scene.web.HTMLEditor;
4. import javafx.stage.Stage;
5. public class HTMLEditorSample extends Application {
6. @Override
7. public void start(Stage stage) {
8. stage.setTitle("HTMLEditor Sample");
9. stage.setWidth(400);
10. stage.setHeight(300);
11. final HTMLEditor htmlEditor = new HTMLEditor();
12. htmlEditor.setPrefHeight(245);
13. Scene scene = new Scene(htmlEditor);
14. stage.setScene(scene);
15. stage.show();
16. }
17. public static void main(String[] args) {
18. launch(args);
19. }
20. }

编译运行上面的代码效果是 Figure 19-2 .

Figure 19-2 Initial View of the HTMLEditor Component



Description of "Figure 19-2 Initial View of the HTMLEditor Component"

实现该组件后就有格式栏，不能关闭它们。不过也可以使用CSS来改变其外观。见Example 19-2 .

Example 19-2 Styling the HTMLEditor

htmlEditor.setStyle(

    "-fx-font: 12 cambria;"

    + "-fx-border-color: brown; "

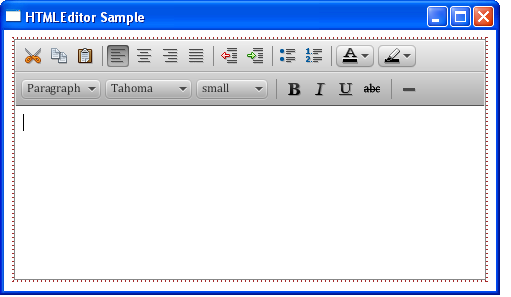
    + "-fx-border-style: dotted;"

    + "-fx-border-width: 2;"

);

把它们加入到 Example 19-1 ，效果是Figure 19-3 .

Figure 19-3 Alternative View of the HTMLEditor Component



Description of "Figure 19-3 Alternative View of the HTMLEditor Component"

现在组件的边框和格式栏的字体改变了。

HTMLEditor 类提供了一个方法来定义应用启动后编辑区显示的内容。使用setHtmlText 方法来设置编辑器的初始文本，见Example 19-3。

Example 19-3 Setting the Text Content

private final String INITIAL\_TEXT = "<html><body>Lorem ipsum dolor sit "

    + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "

    + "in scelerisque cursus, pulvinar at ante. Nulla consequat"

    + "congue lectus in sodales. Nullam eu est a felis ornare "

    + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "

    + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "

    + "Integer congue faucibus dapibus. Integer id nisl ut elit "

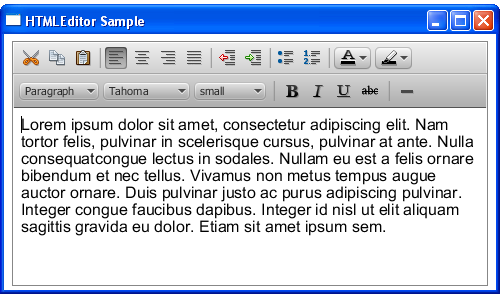
    + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "

    + "sem.</body></html>";

htmlEditor.setHtmlText(INITIAL\_TEXT);

Figure 19-4 是使用setHTMLText 方法后的编辑器。

Figure 19-4 HTMLEditor with the Predefined Text Content



Description of "Figure 19-4 HTMLEditor with the Predefined Text Content"

可以在字符串中使用HTML标签来指定初始显示的文本格式。

用HTML Editor构建用户接口

能够使用 HTMLEditor 控件来实现典型的用户接口，比如说可以实现消息服务、email客户端、甚至内容管理系统。

下面实现一个消息排版窗口，在很多email客户端应用中都可以找到它。

Example 19-4 HTMLEditor Added to the Email Client UI

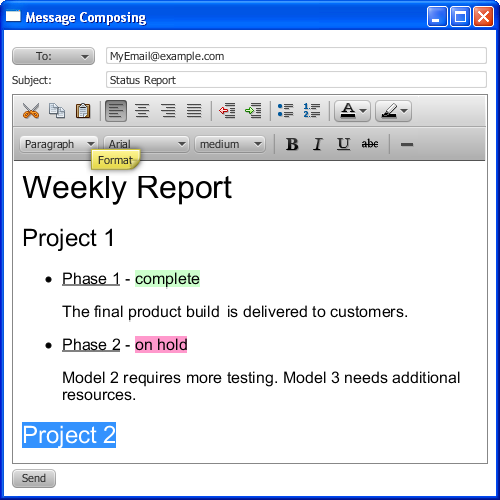
[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.collections.FXCollections;
3. import javafx.geometry.Insets;
4. import javafx.geometry.Pos;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.\*;
8. import javafx.scene.layout.GridPane;
9. import javafx.scene.layout.VBox;
10. import javafx.scene.web.HTMLEditor;
11. import javafx.stage.Stage;
12. public class HTMLEditorSample extends Application {
14. @Override
15. public void start(Stage stage) {
16. stage.setTitle("Message Composing");
17. stage.setWidth(500);
18. stage.setHeight(500);
19. Scene scene = new Scene(new Group());
21. final VBox root = new VBox();
22. root.setPadding(new Insets(8, 8, 8, 8));
23. root.setSpacing(5);
24. root.setAlignment(Pos.BOTTOM\_LEFT);
26. final GridPane grid = new GridPane();
27. grid.setVgap(5);
28. grid.setHgap(10);
30. final ChoiceBox sendTo =
31. new ChoiceBox(FXCollections.observableArrayList(
32. "To:", "Cc:", "Bcc:")
33. );
35. sendTo.setPrefWidth(100);
36. GridPane.setConstraints(sendTo, 0, 0);
37. grid.getChildren().add(sendTo);
39. final TextField tbTo = new TextField();
40. tbTo.setPrefWidth(400);
41. GridPane.setConstraints(tbTo, 1, 0);
42. grid.getChildren().add(tbTo);
44. final Label subjectLabel = new Label("Subject:");
45. GridPane.setConstraints(subjectLabel, 0, 1);
46. grid.getChildren().add(subjectLabel);
48. final TextField tbSubject = new TextField();
49. tbTo.setPrefWidth(400);
50. GridPane.setConstraints(tbSubject, 1, 1);
51. grid.getChildren().add(tbSubject);
53. root.getChildren().add(grid);
55. final HTMLEditor htmlEditor = new HTMLEditor();
56. htmlEditor.setPrefHeight(370);
57. root.getChildren().addAll(htmlEditor, new Button("Send"));
59. final Label htmlLabel = new Label();
60. htmlLabel.setWrapText(true);
62. scene.setRoot(root);
63. stage.setScene(scene);
64. stage.show();
65. }
66. public static void main(String[] args) {
67. launch(args);
68. }
69. }

该接口包括一个选项框来选择接受类型，2个文本框来输入email地址和主题，一个标签来显示主题字段，一个编辑器，还有发送按钮。

使用Grid 和VBox布局容器把这些UI控件加入到应用的场景中。编译运行的效果见 Figure 19-5 ，这是一个用户正在排版周报。

Figure 19-5 Email Client User Interface



Description of "Figure 19-5 Email Client User Interface"

调 用setPrefWidth 或 setPrefHeight方法为 HTMLEditor对象设置宽或高 ，当然根本不指定也行。Example 19-4 中为组件高度指定了值，而宽度由VBox悲剧容器控制了。当内容文本超出了编辑区的宽度和高度时，垂直滚动条就显示出来。

**获取HTML内容**

用JavaFX HTMLEditor控件，你可以编辑和设置初始内容。此外，你还可以以HTML格式获得输入的和编辑的内容。具体实现见 Example 19-5

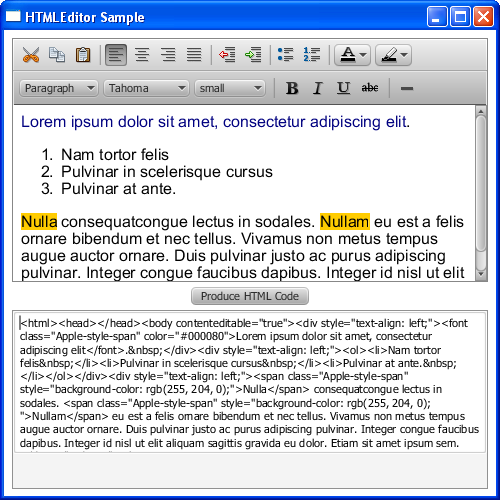
Example 19-5 Retrieving HTML Code

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.geometry.Insets;
5. import javafx.geometry.Pos;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.control.\*;
9. import javafx.scene.layout.VBox;
10. import javafx.scene.web.HTMLEditor;
11. import javafx.stage.Stage;
12. public class HTMLEditorSample extends Application {
13. private final String INITIAL\_TEXT = "Lorem ipsum dolor sit "
14. + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "
15. + "in scelerisque cursus, pulvinar at ante. Nulla consequat"
16. + "congue lectus in sodales. Nullam eu est a felis ornare "
17. + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "
18. + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "
19. + "Integer congue faucibus dapibus. Integer id nisl ut elit "
20. + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "
21. + "sem.";
22. @Override
23. public void start(Stage stage) {
24. stage.setTitle("HTMLEditor Sample");
25. stage.setWidth(500);
26. stage.setHeight(500);
27. Scene scene = new Scene(new Group());
29. VBox root = new VBox();
30. root.setPadding(new Insets(8, 8, 8, 8));
31. root.setSpacing(5);
32. root.setAlignment(Pos.BOTTOM\_LEFT);
34. final HTMLEditor htmlEditor = new HTMLEditor();
35. htmlEditor.setPrefHeight(245);
36. htmlEditor.setHtmlText(INITIAL\_TEXT);
37. final TextArea htmlCode = new TextArea();
38. htmlCode.setWrapText(true);
40. ScrollPane scrollPane = new ScrollPane();
41. scrollPane.getStyleClass().add("noborder-scroll-pane");
42. scrollPane.setContent(htmlCode);
43. scrollPane.setFitToWidth(true);
44. scrollPane.setPrefHeight(180);
45. Button showHTMLButton = new Button("Produce HTML Code");
46. root.setAlignment(Pos.CENTER);
47. showHTMLButton.setOnAction(new EventHandler<ActionEvent>() {
48. @Override public void handle(ActionEvent arg0) {
49. htmlCode.setText(htmlEditor.getHtmlText());
50. }
51. });
53. root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
54. scene.setRoot(root);
55. stage.setScene(scene);
56. stage.show();
57. }
58. public static void main(String[] args) {
59. launch(args);
60. }
61. }

getHTMLText方法获得了编辑内容并以HTML字符串形式呈现。该内容传递给文本区，这样就能查看、复制、粘贴这些HTML代码。 Figure 19-6 就是样例。

Figure 19-6 Obtaining the HTML Content



Description of "Figure 19-6 Obtaining the HTML Content"

类似地，也可以获得HTML代码比in个保存为文件或者发送到WebView对象，以同步编辑器和绑定的浏览器中的内容。 下面实现了这个任务 Example 19-6 .

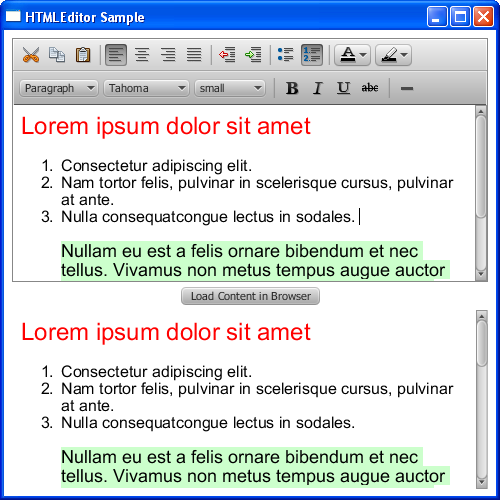
Example 19-6 Rendering Edited HTML Content in a Browser

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.geometry.Insets;
5. import javafx.geometry.Pos;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.control.\*;
9. import javafx.scene.layout.VBox;
10. import javafx.scene.web.HTMLEditor;
11. import javafx.scene.web.WebEngine;
12. import javafx.scene.web.WebView;
13. import javafx.stage.Stage;
14. public class HTMLEditorSample extends Application {
15. private final String INITIAL\_TEXT = "Lorem ipsum dolor sit "
16. + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "
17. + "in scelerisque cursus, pulvinar at ante. Nulla consequat"
18. + "congue lectus in sodales. Nullam eu est a felis ornare "
19. + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "
20. + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "
21. + "Integer congue faucibus dapibus. Integer id nisl ut elit "
22. + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "
23. + "sem.";
24. @Override
25. public void start(Stage stage) {
26. stage.setTitle("HTMLEditor Sample");
27. stage.setWidth(500);
28. stage.setHeight(500);
29. Scene scene = new Scene(new Group());
31. VBox root = new VBox();
32. root.setPadding(new Insets(8, 8, 8, 8));
33. root.setSpacing(5);
34. root.setAlignment(Pos.BOTTOM\_LEFT);
35. final HTMLEditor htmlEditor = new HTMLEditor();
36. htmlEditor.setPrefHeight(245);
37. htmlEditor.setHtmlText(INITIAL\_TEXT);
39. final WebView browser = new WebView();
40. final WebEngine webEngine = browser.getEngine();
42. ScrollPane scrollPane = new ScrollPane();
43. scrollPane.getStyleClass().add("noborder-scroll-pane");
44. scrollPane.setStyle("-fx-background-color: white");
45. scrollPane.setContent(browser);
46. scrollPane.setFitToWidth(true);
47. scrollPane.setPrefHeight(180);
48. Button showHTMLButton = new Button("Load Content in Browser");
49. root.setAlignment(Pos.CENTER);
50. showHTMLButton.setOnAction(new EventHandler<ActionEvent>() {
51. @Override public void handle(ActionEvent arg0) {
52. webEngine.loadContent(htmlEditor.getHtmlText());
53. }
54. });
56. root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
57. scene.setRoot(root);
58. stage.setScene(scene);
59. stage.show();
60. }
61. public static void main(String[] args) {
62. launch(args);
63. }
64. }

从htmlEditor 组件获得HTML代码加载到WebEngine 对象来指定绑定浏览器的内容。每次用户点击Load Content in Browser按钮，编辑的文本就更新到浏览器中。Figure 19-7 是Example 19-6 运行的效果。

Figure 19-7 Loading Content in a Browser



Description of "Figure 19-7 Loading Content in a Browser"

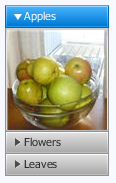
使用Text 组件来添加非编辑文本内容。到[Using Text and Text Effects in JavaFX](http://download.oracle.com/javafx/2.0/text/jfxpub-text.htm) 了解更多Text组件。

# 第二十六回 JavaFX2.0 标题窗格TitledPane和手风琴控件Accordion

标题窗格就是带有标题的面板，可以被打开和关闭，也可以被包进任何Node元素，诸如UI控件、图片、计入布局容器的元素组。

标题窗格可以用手风琴控件来形成组。手风琴控件能创建多个窗格而每次只显示一个。Figure 20-1 是带有3个标题窗格的手风琴控件。

Figure 20-1 Accordion with Three Titled Panes



Description of "Figure 20-1 Accordion with Three Titled Panes"

JavaFX SDK API中的 Accordion 和 TitledPane 类用来实现这样的控件。

**创建Titled Panes**

创建TitledPane 控件要定义一个标题和一些内容。可以使用TitledPane 类的带有两个参数的构造方法，或者单独使用setText 和setContent 方法也行。两种方法都在Example 20-1 中 .

Example 20-1 Declaring a TitledPane Object

//using a two-parameter constructor

TitledPane tp = new TitledPane("My Titled Pane", new Button("Button"));

//applying methods

TitledPane tp = new TitledPane();

tp.setText("My Titled Pane");

tp.setContent(new Button("Button"));

它们的效果系统，都是 Figure 20-2 .

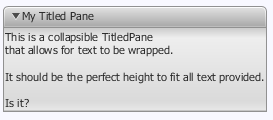
Figure 20-2 Titled Pane with a Button

http://www.javafxfan.net/attachment/Mon_1112/3_2_fba251ec293f7a0.png?3097

Description of "Figure 20-2 Titled Pane with a Button"

标题窗格可以改变大小来适应它的内容。可以添加多行文本，结果见Figure 20-3 .

Figure 20-3 Titled Pane with Some Text



Description of "Figure 20-3 Titled Pane with Some Text"

不要明确指定标题窗格的最小、最大和优先的高度值，这样在打开关闭时可能导致难以预料的行为。

Example 20-2 在的代码添加了几个控件到标题窗格，然后加入到了GridPane 布局容器。

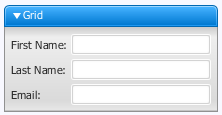
Example 20-2 Titled Pane with the GridPane Layout Container

[复制代码](javascript:)

1. TitledPane gridTitlePane = new TitledPane();
2. GridPane grid = new GridPane();
3. grid.setVgap(4);
4. grid.setPadding(new Insets(5, 5, 5, 5));
5. grid.add(new Label("First Name: "), 0, 0);
6. grid.add(new TextField(), 1, 0);
7. grid.add(new Label("Last Name: "), 0, 1);
8. grid.add(new TextField(), 1, 1);
9. grid.add(new Label("Email: "), 0, 2);
10. grid.add(new TextField(), 1, 2);
11. gridTitlePane.setText("Grid");
12. gridTitlePane.setContent(grid);

运行的结果是 Figure 20-4 。

Figure 20-4 Titled Pane that Contains Several Controls



Description of "Figure 20-4 Titled Pane that Contains Several Controls"

可 以定义标题窗格打开关闭的方式。默认地，标题窗格是可伸缩的，它们的移动也是动画。如果要阻止标题窗格关闭，将setCollapsible方法 设为false。 也可以将 setAnimated 方法设为false来关闭动画打开效果。 Example 20-3 中的代码实现了该任务。

Example 20-3 Adjusting the Style of a Titled Pane

TitledPane tp = new TitledPane();

//prohibit closing

tp.setCollapsible(false);

//prohibit animating

tp.setAnimated(false);

**将Titled Panes加入到Accordion**

在应用中，可以单独使用标题窗格，也可以使用Accordion 把控件编组。同样也不要指定手风琴控件的高度值。

将几个标题窗格加入到手风琴很类似把开关按钮加入到开关组：每次只能打开手风琴中的一个标题窗格。Example 20-4 创建了3个标题窗格并加入到了手风琴中。

Example 20-4 Accordion and Three Titled Panes

[复制代码](javascript:)

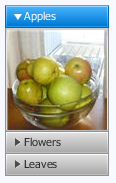
1. import javafx.application.Application;
2. import javafx.scene.Group;
3. import javafx.scene.Scene;
4. import javafx.scene.control.Accordion;
5. import javafx.scene.control.TitledPane;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.paint.Color;
9. import javafx.stage.Stage;
10. public class TitledPaneSample extends Application {
11. final String[] imageNames = new String[]{"Apples", "Flowers", "Leaves"};
12. final Image[] images = new Image[imageNames.length];
13. final ImageView[] pics = new ImageView[imageNames.length];
14. final TitledPane[] tps = new TitledPane[imageNames.length];
16. public static void main(String[] args) {
17. launch(args);
18. }
19. @Override public void start(Stage stage) {
20. stage.setTitle("TitledPane");
21. Scene scene = new Scene(new Group(), 80, 180);
22. scene.setFill(Color.GHOSTWHITE);
24. final Accordion accordion = new Accordion ();

27. for (int i = 0; i < imageNames.length; i++) {
29. images[i] = new
30. Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
31. pics[i] = new ImageView(images[i]);
32. tps[i] = new TitledPane(imageNames[i],pics[i]);
33. }
34. accordion.getPanes().addAll(tps);
35. accordion.setExpandedPane(tps[0]);
36. Group root = (Group)scene.getRoot();
37. root.getChildren().add(accordion);
38. stage.setScene(scene);
39. stage.show();
40. }
41. }

用循环创建了3个标题窗格，每个的内容都是ImageView 对象。把标题窗格加入到手风琴中要使用getPanes 和addAll 方法。可以用add 方法代替addAll 方法来加入单个标题窗格。

默认地，应用启动后所有窗格都关着。setExpandedPane方法指定了带有苹果图片的窗格要打开。见 Figure 20-5 .

Figure 20-5 Accordion with Three Titled Panes



Description of "Figure 20-5 Accordion with Three Titled Panes"

**处理Accordion事件**

可以使用标题窗格和手风琴程序不同的数据。Example 20-5 创建了一个单独的标题窗格放进GridPane 悲剧容器和三个标题窗格放进手风琴中。单独的窗格包含了一个email客户端元素，手风琴使得选择窗格会显示相应的图片。

Example 20-5 Implementing ChangeListener for an Accordion

[复制代码](javascript:)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Insets;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Accordion;
8. import javafx.scene.control.Label;
9. import javafx.scene.control.TextField;
10. import javafx.scene.control.TitledPane;
11. import javafx.scene.image.Image;
12. import javafx.scene.image.ImageView;
13. import javafx.scene.layout.GridPane;
14. import javafx.scene.layout.HBox;
15. import javafx.scene.paint.Color;
16. import javafx.stage.Stage;
17. public class TitledPaneSample extends Application {
18. final String[] imageNames = new String[]{"Apples", "Flowers", "Leaves"};
19. final Image[] images = new Image[imageNames.length];
20. final ImageView[] pics = new ImageView[imageNames.length];
21. final TitledPane[] tps = new TitledPane[imageNames.length];
22. final Label label = new Label("N/A");
24. public static void main(String[] args) {
25. launch(args);
26. }
27. @Override public void start(Stage stage) {
28. stage.setTitle("TitledPane");
29. Scene scene = new Scene(new Group(), 800, 250);
30. scene.setFill(Color.GHOSTWHITE);
32. // --- GridPane container
33. TitledPane gridTitlePane = new TitledPane();
34. GridPane grid = new GridPane();
35. grid.setVgap(4);
36. grid.setPadding(new Insets(5, 5, 5, 5));
37. grid.add(new Label("To: "), 0, 0);
38. grid.add(new TextField(), 1, 0);
39. grid.add(new Label("Cc: "), 0, 1);
40. grid.add(new TextField(), 1, 1);
41. grid.add(new Label("Subject: "), 0, 2);
42. grid.add(new TextField(), 1, 2);
43. grid.add(new Label("Attachment: "), 0, 3);
44. grid.add(label,1, 3);
45. gridTitlePane.setText("Grid");
46. gridTitlePane.setContent(grid);
48. // --- Accordion
49. final Accordion accordion = new Accordion ();
50. for (int i = 0; i < imageNames.length; i++) {
51. images[i] = new
52. Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
53. pics[i] = new ImageView(images[i]);
54. tps[i] = new TitledPane(imageNames[i],pics[i]);
55. }
56. accordion.getPanes().addAll(tps);
57. accordion.expandedPaneProperty().addListener(new
58. ChangeListener<TitledPane>() {
59. public void changed(ObservableValue<? extends TitledPane> ov,
60. TitledPane old\_val, TitledPane new\_val) {
61. if (new\_val != null) {
62. label.setText(accordion.getExpandedPane().getText() +
63. ".jpg");
64. }
65. }
66. });
68. HBox hbox = new HBox(10);
69. hbox.setPadding(new Insets(20, 0, 0, 20));
70. hbox.getChildren().setAll(gridTitlePane, accordion);
71. Group root = (Group)scene.getRoot();
72. root.getChildren().add(hbox);
73. stage.setScene(scene);
74. stage.show();
75. }
76. }

当打开手风琴中的标题窗格时，手风琴的 expandedPaneProperty 属性就会改变。ChangeListener对象通报了该变化，而手风琴中打开的标题窗格就构建一个文件名，该文件名就作为相应 Label对象的文本。

Figure 20-6 是应用启动后的样子，Attachment标签是"N/A,"因为没有窗格被选中。

Figure 20-6 Initial View of the TitledPaneSample Application



Description of "Figure 20-6 Initial View of the TitledPaneSample Application"

如果打开的是Leaves标题窗格，Attachment标签就变成"Leaves.jpg,"见Figure 20-7 .

Figure 20-7 TitledPaneSample Application with the Leaves Titled Pane Expanded



Description of "Figure 20-7 TitledPaneSample Application with the Leaves Titled Pane Expanded"

TitledPane 和Accordion 类都继承了Node类，所以可以应用特效和使用CSS。

# 第二十七回 JavaFX2.0 文本Text及其特效

文本讲述如何在JavaFX2.0应用中加入文本和如何为文本提供花俏的效果。

**引子**

JavaFX 2.0应用的图形内容包含一些对象，它们被组织在一个成为场景图的类树结构中。场景图中的每个元素成为一个结点，结点可以管理很多不同种类的内容，包括文 本。结点可以转换和移动，也可以应用多种效果。为所有结点类型使用共同特点使得可以提供复杂的文本内容来满足现在的富网络应用(RIAs).

JavaFX 2.0发布版提供了javafx.scene.text.Text类用来显示文本。 Text类继承自 Node 类，所以可以为其应用特效、动画、转换，和其他结点一样的。而又Node类继承自 Shape 类，可以像其他形状一样为其设置描边和填充效果。

**添加Text**

To add a text object to your application, 要添加文本，使用下面任一构造方法。Example 1 到Example 3 .

Example 1

Text t = new Text();

t.setText("This is a text sample");

Example 2

Text t = new Text("This is a text sample");

Example 3

Text t = new Text (10, 20, "This is a text sample");

也可以使用javafx.scene.text.TextBuilder 类创建，见Example 4 .

Example 4

Text t = TextBuilder.create().text("This is a text sample").build();

**设置字体和颜色**

添加文本后就可以设置一些属性了。要设置使用的字体，实例化javafx.scene.text.Font 类。Font.font()方法可以指定字体和字号，也可以像下面这样设置颜色 Example 5 .

Example 5

t.setText("This is a text sample");

t.setFont(Font.font ("Verdana", 20));

t.setFill(Color.RED);

或者使用系统字体，它是依赖于不同OS平台的。这样的话，使用Font.getDefault() 方法。

不使用单一颜色，也可以使用线性渐变填充，见Example 6 .

Example 6

Text text = TextBuilder.create().text("Stroke and Fill").

                    font(Font.font("Tahoma", 100)).build();

text.setFill(new LinearGradient(0, 0, 1, 2, true, CycleMethod.REPEAT, new

         Stop[]{new Stop(0, Color.AQUA), new Stop(0.5f, Color.RED)}));

text.setStrokeWidth(1);

text.setStroke(Color.BLACK);

效果如下 Figure 1 .

Figure 1 Text with a Linear Gradient Filling



Description of "Figure 1 Text with a Linear Gradient Filling"

**设置粗体和斜体**

font 方法的 FontWeight 常数可以设置粗体，见Example 7 .

Example 7

t.setFont(Font.font("Verdana", FontWeight.BOLD, 70));

FontPosture 常数来设置斜体，见Example 8 .

Example 8

t.setFont(Font.font("Verdana", FontPosture.ITALIC, 20));

**使用定制Font**

如果需要使用某种独特的字体而其他机器上很可能没有安装，可以在 JavaFX 2.0应用中包含TrueType字体 (.ttf) 或OpenType字体 (.otf)。

要包含 TrueType或 OpenType字体，这样做:

在工程目录下创建 resources/fonts文件夹

把字体文件复制到fonts子文件夹下

在代码中找下面这样加载字体。Example 9 .

Example 9

text.setFont(Font.loadFont("file:resources/fonts/isadoracyr.ttf", 120));

该字体如下 Figure 2 .

Figure 2 Custom Font



Description of "Figure 2 Custom Font"

**应用Effect**

JavaFX 2.0发布版在javafx.scene.effect 包中提供了大量特效。前面提到，可以为文本结点应用特效。完整的效果集合，查看API文档。可以通过TextEffects示例应用查看一些效果。该应用中文本结点使用了一系列效果。从<http://download.oracle.com/javafx/2.0/text/TextEffects.java.html>

下载 texteffects.zip 文件并解压到本地，用NB打开为工程。

**透视效果**

PerspectiveTransform 类使得可以在二维内容中模拟三维效果。透视转换可以将一个任意的四边形映射为另一个四边形。输入就是一个结点，而输出依赖于指定的四个角的X和Y坐标。 在TextEffects 应用中，PerspectiveTransform 效果是为一组包含矩形和文本的group应用的，见Example 10 。

Example 10

[复制代码](javascript:)

1. PerspectiveTransform pt = new PerspectiveTransform();
2. pt.setUlx(10.0f);
3. pt.setUly(10.0f);
4. pt.setUrx(310.0f);
5. pt.setUry(40.0f);
6. pt.setLrx(310.0f);
7. pt.setLry(60.0f);
8. pt.setLlx(10.0f);
9. pt.setLly(90.0f);
10. g.setEffect(pt);
11. g.setCache(true);
12. Rectangle r = new Rectangle();
13. r.setX(10.0f);
14. r.setY(10.0f);
15. r.setWidth(280.0f);
16. r.setHeight(80.0f);
17. r.setFill(Color.BLUE);
18. Text t = new Text();
19. t.setX(20.0f);
20. t.setY(65.0f);
21. t.setText("Perspective");
22. t.setFill(Color.YELLOW);
23. t.setFont(Font.font(null, FontWeight.BOLD, 36));
24. g.getChildren().add(r);
25. g.getChildren().add(t);
26. return g;

效果如下 Figure 3 .

Figure 3 Text with a Perspective Effect



Description of "Figure 3 Text with a Perspective Effect"

**模糊效果**

GaussianBlur 类提供了基于高斯卷积内核的模糊效果。

Example 11 是一个应用了模糊效果的文本结点，见TextEffects 应用。

Example 11

[复制代码](javascript:)

1. Text t2 = new Text();
2. t2.setX(10.0f);
3. t2.setY(140.0f);
4. t2.setCache(true);
5. t2.setText("Blurry Text");
6. t2.setFill(Color.RED);
7. t2.setFont(Font.font(null, FontWeight.BOLD, 36));
8. t2.setEffect(new GaussianBlur());
9. return t2;

效果如下 Figure 4 .

Figure 4 Text with a Blur Effect



Description of "Figure 4 Text with a Blur Effect "

**外部阴影效果**

要实现外部阴影效果，使用 DropShadow 类。可以为文本阴影指定一种颜色和偏移量。在TextEffects 应用中，文本是红色的，而外边阴影效果是3点的灰色。代码见Example 12 .

Example 12

[复制代码](javascript:)

1. DropShadow ds = new DropShadow();
2. ds.setOffsetY(3.0f);
3. ds.setColor(Color.color(0.4f, 0.4f, 0.4f));
4. Text t = new Text();
5. t.setEffect(ds);
6. t.setCache(true);
7. t.setX(10.0f);
8. t.setY(270.0f);
9. t.setFill(Color.RED);
10. t.setText("JavaFX drop shadow...");
11. t.setFont(Font.font(null, FontWeight.BOLD, 32));

效果如下 Figure 5 .

Figure 5 Text with a Drop Shadow Effect

http://www.javafxfan.net/attachment/Mon_1112/3_2_e97484395af3e8e.jpg?6979

Description of "Figure 5 Text with a Drop Shadow Effect"

**内部阴影效果**

内部阴影效果在内容的内边缘显示阴影。使用时也要指定颜色也偏移量。下面是在x和y方向都是4点偏移的内部阴影效果代码Example 13 .

Example 13

[复制代码](javascript:)

1. InnerShadow is = new InnerShadow();
2. is.setOffsetX(4.0f);
3. is.setOffsetY(4.0f);
4. Text t = new Text();
5. t.setEffect(is);
6. t.setX(20);
7. t.setY(100);
8. t.setText("InnerShadow");
9. t.setFill(Color.YELLOW);
10. t.setFont(Font.font(null, FontWeight.BOLD, 80));
11. t.setTranslateX(300);
12. t.setTranslateY(300);
13. return t;

Figure 6 Text with an Inner Shadow Effect



Description of "Figure 6 Text with an Inner Shadow Effect"

**反射**

Reflection类呈现的是原始文本的倒影。也可以通过提供额外的参数来调整文本反射视图，如 底透明度、反射可见比、与原文距离、顶透明度。更多细节，查看API文档。

TextEffects 应用中的反射效果实现见Example 14 .

Example 14

[复制代码](javascript:)

1. Text t = new Text();
2. t.setX(10.0f);
3. t.setY(50.0f);
4. t.setCache(true);
5. t.setText("Reflections on JavaFX...");
6. t.setFill(Color.RED);
7. t.setFont(Font.font(null, FontWeight.BOLD, 30));
8. Reflection r = new Reflection();
9. r.setFraction(0.7f);
10. t.setEffect(r);
11. t.setTranslateY(400);
12. return t;

运行后见 Figure 7 .

Figure 7 Text with a Reflection Effect



Description of "Figure 7 Text with a Reflection Effect"

**整合多个效果**

前面学了如何使用单一效果，要丰富文本内容就要组合多种效果，应用一个效果链来获得特殊的视觉感触。看下NeonSign 应用（点击下载） 的效果Figure 8 .

Figure 8



Description of "Figure 8 "

NeonSign 应用的图形场景中包含了以下元素：

背景使用的是一副砖墙图片

一个矩形提供了放射渐变填充

一个文本结点使用了效果链

一个文本域用于输入数据

该应用使用了一个绑定机制来设置文本结点显示输入的文本值。文本结点的文本属性（textProperty）绑定到了文本域的文本属性，见Example 15 .

Example 15

Text text = new Text();

TextField textField = new TextField();

textField.setText("Neon Sign");

text.textProperty().bind(textField.textProperty());

可以看到文本框输入的变化会引起文本结点的变化

文 本结点使用了效果链。主要的效果是一个混合效果，使用了MULTIPLY模式来结合两种输入：一个外部阴影效果和另一个混合效果 blend1。类似地， blend1 效果结合了一个外部阴影效果(ds1 )和一个混合效果 (blend2 )。blend2 效果结合了两种内部阴影效果。使用这个效果链和不同的颜色为文本对象应用了精细和复杂的颜色模式。下面是效果链的代码Example 16 .

Example 16

[复制代码](javascript:)

1. Blend blend = new Blend();
2. blend.setMode(BlendMode.MULTIPLY);
3. DropShadow ds = new DropShadow();
4. ds.setColor(Color.rgb(254, 235, 66, 0.3));
5. ds.setOffsetX(5);
6. ds.setOffsetY(5);
7. ds.setRadius(5);
8. ds.setSpread(0.2);
9. blend.setBottomInput(ds);
10. DropShadow ds1 = new DropShadow();
11. ds1.setColor(Color.web("#f13a00"));
12. ds1.setRadius(20);
13. ds1.setSpread(0.2);
14. Blend blend2 = new Blend();
15. blend2.setMode(BlendMode.MULTIPLY);
16. InnerShadow is = new InnerShadow();
17. is.setColor(Color.web("#feeb42"));
18. is.setRadius(9);
19. is.setChoke(0.8);
20. blend2.setBottomInput(is);
21. InnerShadow is1 = new InnerShadow();
22. is.setColor(Color.web("#f13a00"));
23. is.setRadius(5);
24. is.setChoke(0.4);
25. blend2.setTopInput(is1);
26. Blend blend1 = new Blend();
27. blend1.setMode(BlendMode.MULTIPLY);
28. blend1.setBottomInput(ds1);
29. blend1.setTopInput(blend2);
30. blend.setTopInput(blend1);
31. text.setEffect(blend);

通过本文学习了如何添加文本和应用各种效果。更全面的信息，查看API文档。如果要在应用中实现一个文本编辑区，使用HTMLEditor组件。关于 HTMLEditor 控件的信息，查看第二十五回 。

# [第二十八回 使用JavaFX布局窗](http://somefuture.iteye.com/blog/1222292)

JavaFX SDK提供了布局窗来支持不同的布局风格。本指南提供使用这些布局窗在JavaFX中创建图形化用户接口的信息。

假设读者具有中级的Java编程技术。

本指南包括以下主题:

* [Using Built-In Layout Panes](http://somefuture.iteye.com/blog/1222293) - 讲述内置的布局窗并提供实例
* [Tips for Sizing and Aligning Nodes](http://somefuture.iteye.com/blog/1222294) - 提供覆盖结点默认大小和位置的例子

# 第二十九回 使用JavaFX2.0内置布局窗格

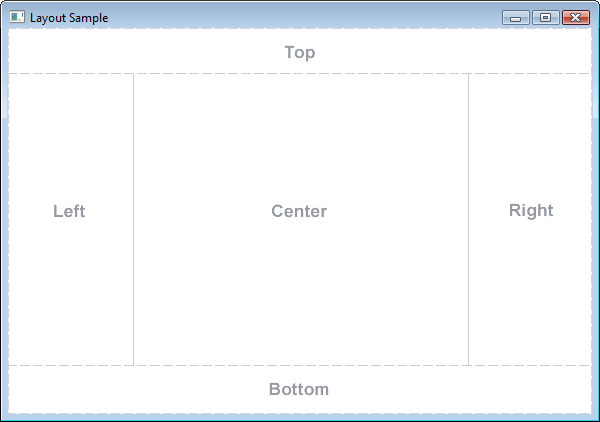
在JavaFX应用中，当然可以通过指定UI元素的位置和大小属性来手动布局。不过，更简单的方法是使用布局窗格。JavaFX SDK提高了多种布局容器类(称为窗格)来方便的建立和管理经典布局,如行、列、堆、拼贴等。由于窗口是可以改变大小的，所以布局窗格会根据其包含的结点 自动修改位置和大小。

本文是JavaFX布局窗格的概览，并为每个窗格提供了小例子。

**边框窗格BorderPane**

BorderPane布局窗格提供了5块放置结点的区域：顶部、底部、座部、右部、中部。 Figure 1-1 是能用该布局窗格创建的布局类型。区域可以是任意大小的，如果不需要某一块，可以不定义。

Figure 1-1 Sample Border Pane



Description of "Figure 1-1 Sample Border Pane"

边框窗格对于经典布局很有用，像顶部的工具栏，底部的状态栏，左边的导航面板，右边的补充信息，中间的工作区。

Example 1-1 创建了一个每个区域是有色矩形的边框窗格。

Example 1-1 Create a Border Pane

[复制代码](javascript:)

1. BorderPane layout = new BorderPane();
2. layout.setTop(new Rectangle(200, 50, Color.DARKCYAN));
3. layout.setBottom(new Rectangle(200, 50, Color.DARKCYAN));
4. layout.setCenter(new Rectangle(100, 100, Color.MEDIUMAQUAMARINE));
5. layout.setLeft(new Rectangle(50, 100, Color.DARKTURQUOISE));
6. layout.setRight(new Rectangle(50, 100, Color.DARKTURQUOISE));

**水平盒子HBox**

HBox 布局窗格提供了一种简单的方法来把一些列结点放进单行里面。Figure 1-2 是一个HBox 窗格的例子。

Figure 1-2 Sample HBox Pane

http://www.javafxfan.net/attachment/Mon_1112/3_2_0bc5f72675c00ec.png?2356

Description of "Figure 1-2 Sample HBox Pane"

Padding属性用来设置结点和HBox的边缘间距离。 Spacing属性用来设置结点间距离。style用来改变背景色。

Example 1-2 创建了一个工具栏的HBox 窗格，里面有两个按钮。

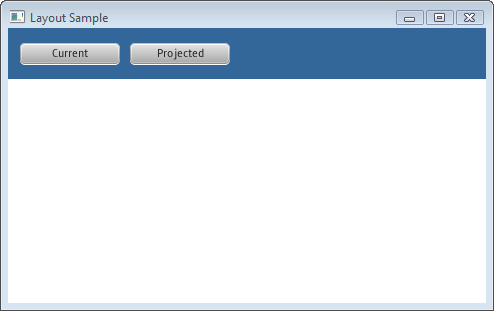
Example 1-2 Create an HBox Pane

[复制代码](javascript:)

1. HBox hbox = new HBox();
2. hbox.setPadding(new Insets(15, 12, 15, 12));
3. hbox.setSpacing(10);
4. hbox.setStyle("-fx-background-color: #336699");
5. Button buttonCurrent = new Button("Current");
6. buttonCurrent.setPrefSize(100, 20);
7. Button buttonProjected = new Button("Projected");
8. buttonProjected.setPrefSize(100, 20);
9. hbox.getChildren().addAll(buttonCurrent, buttonProjected);
10. BorderPane border = new BorderPane();
11. border.setTop(hbox);

Example 1-2 中的最后一行创建了一个边框布局，并把HBox加入到顶部区域。结果见 Figure 1-3 .

Figure 1-3 HBox Pane in a Border Pane



Description of "Figure 1-3 HBox Pane in a Border Pane"

**垂直盒子VBox**

VBox 布局窗格和HBox 布局很类似，区别仅仅是垂直盒子的结点是组织进一列中。Figure 1-4 是一个VBox窗格的例子。

Figure 1-4 Sample VBox Pane



Description of "Figure 1-4 Sample VBox Pane"

Padding属性用来设置结点和VBox的边缘间距离。 Spacing属性用来设置结点间距离。

Example 1-3 创建了一个选项列表VBox。

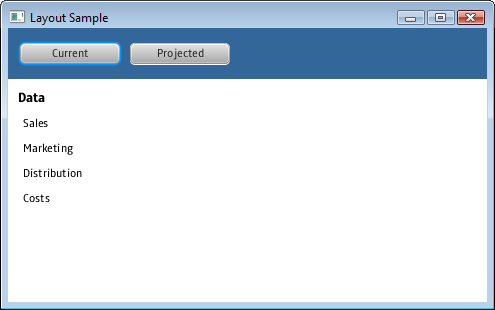
Example 1-3 Create a VBox Pane

[复制代码](javascript:)

1. VBox vbox = new VBox();
2. vbox.setPadding(new Insets(10, 10, 10, 10));
3. vbox.setSpacing(10);
4. Text title = new Text("Data");
5. title.setFont(Font.font("Amble CN", FontWeight.BOLD, 14));
6. vbox.getChildren().add(title);
7. Text options[] = new Text[] {
8. new Text("  Sales"),
9. new Text("  Marketing"),
10. new Text("  Distribution"),
11. new Text("  Costs")};
12. for (int i=0; i<4; i++) {
13. vbox.getChildren().add(options[i]);
14. }
15. border.setLeft(vbox);       // Add to BorderPane from Example 1-2

Example 1-3 中最后一行把VBox窗格加入到了边框布局中的左部。结果见 Figure 1-5 .

Figure 1-5 VBox Pane in a Border Pane



Description of "Figure 1-5 VBox Pane in a Border Pane"

**堆栈窗格StackPane**

StackPane布局窗格把所有结点放进一个堆栈中，新结点都在前一个结点上面。该布局模式可以方便地 在形状和图片上叠加文字或将多种简单形状组合成一个复杂形状。Figure 1-6 是一个帮助图标，是将一个问号放在了具有渐变背景的矩形上面。

Figure 1-6 Sample Stack Pane

http://www.javafxfan.net/attachment/Mon_1112/3_2_c8ec25554eff199.png?1004

Description of "Figure 1-6 Sample Stack Pane"

Example 1-4 为帮助图标创建了堆栈窗格。

Example 1-4 Create a Stack Pane

[复制代码](javascript:)

1. StackPane stack = new StackPane();
2. Rectangle helpIcon = new Rectangle(35.0, 25.0);
3. helpIcon.setFill(new LinearGradient(0,0,0,1, true, CycleMethod.NO\_CYCLE,
4. new Stop[]{
5. new Stop(0,Color.web("#4977A3")),
6. new Stop(0.5, Color.web("#B0C6DA")),
7. new Stop(1,Color.web("#9CB6CF")),}));
8. helpIcon.setStroke(Color.web("#D0E6FA"));
9. helpIcon.setArcHeight(3.5);
10. helpIcon.setArcWidth(3.5);
11. Text helpText = new Text("?   ");
12. helpText.setFont(Font.font("Amble Cn", FontWeight.BOLD, 18));
13. helpText.setFill(Color.WHITE);
14. helpText.setStroke(Color.web("#7080A0"));
15. stack.getChildren().addAll(helpIcon, helpText);
16. stack.setAlignment(Pos.CENTER\_RIGHT);     // Right-justify nodes in stack
17. HBox.setHgrow(stack, Priority.ALWAYS);    // Give stack any extra space
18. hbox.getChildren().add(stack);            // Add to HBox from Example 1-2

Example 1-4 的最后一行把堆栈窗格加入到了HBox 中，并且让它永远在最右边。结果见Figure 1-7 .

Figure 1-7 Stack Pane in an HBox Pane

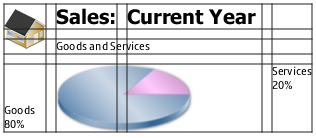
http://www.javafxfan.net/attachment/Mon_1112/3_2_4ddd61636d5564f.png?3070

Description of "Figure 1-7 Stack Pane in an HBox Pane"

**网格窗格GridPane**

GridPane 布局窗格可以灵活的创建放置结点的行和列的网络。结点可以放在任何网格细胞中，并且需要时还可以跨细胞。网格窗格用来创建表格或者是任何用行和列组织的布 局。Figure 1-8 是一个包含一个图标、小标题、文本和饼图的网格窗格。在图中，gridLinesVisible 属性用来设置显示网格线以看出行和列。该属性对于调试GridPane 布局很有用。

Figure 1-8 Sample Grid Pane



Description of "Figure 1-8 Sample Grid Pane"

Gap属性来控制行和列直接的空间。padding属性来控制结点和网格窗格边缘的距离。

Example 1-5 creates the grid pane shown in Figure 1-8 .

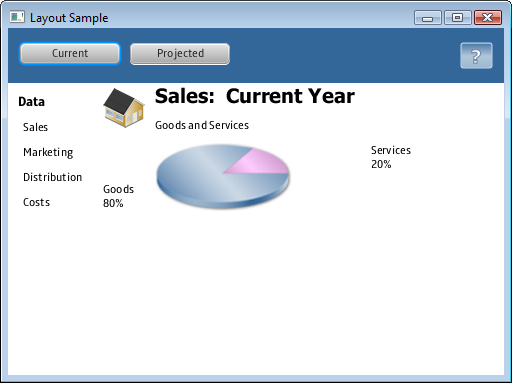
Example 1-5 Create a Grid Pane

[复制代码](javascript:)

1. GridPane grid = new GridPane();
2. grid.setHgap(10);
3. grid.setVgap(10);
4. grid.setPadding(new Insets(0, 0, 0, 10));
5. // Category in column 2, row 1
6. Text category = new Text("Sales:");
7. category.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
8. grid.add(category, 1, 0);
9. // Title in column 3, row 1
10. Text chartTitle = new Text("Current Year");
11. chartTitle.setFont(Font.font("Tahoma", FontWeight.BOLD, 20));
12. grid.add(chartTitle, 2, 0);
13. // Subtitle in columns 2-3, row 2
14. Text chartSubtitle = new Text("Goods and Services");
15. grid.add(chartSubtitle, 1, 1, 2, 1);
16. // House icon in column 1, rows 1-2
17. ImageView imageHouse = new ImageView(
18. new Image(LayoutSample.class.getResourceAsStream("graphics/house.png")));
19. grid.add(imageHouse, 0, 0, 1, 2);
20. // Left label in column 1 (bottom), row 3
21. Text goodsPercent = new Text("Goods\n80%");
22. GridPane.setValignment(goodsPercent, VPos.BOTTOM);
23. grid.add(goodsPercent, 0, 2);
24. // Chart in columns 2-3, row 3
25. ImageView imageChart = new ImageView(
26. new Image(LayoutSample.class.getResourceAsStream("graphics/piechart.png")));
27. grid.add(imageChart, 1, 2, 2, 1);
28. // Right label in column 4 (top), row 3
29. Text servicesPercent = new Text("Services\n20%");
30. GridPane.setValignment(servicesPercent, VPos.TOP);
31. grid.add(servicesPercent, 3, 2);
32. border.setCenter(grid);       // Add to BorderPane from Example 1-2

Example 1-5 的最后一行把网格布局放到了边框布局的中间。结果见Figure 1-9 .

Figure 1-9 Grid Pane in a Border Pane



Description of "Figure 1-9 Grid Pane in a Border Pane"

由于窗口大小的变化，网格成功的结点会根据布局限制重置大小。

**流窗格FlowPane**

FlowPane布局窗格中的结点会连续放置在窗格的边界集中。结点可以垂直流向 (columns) 或水平流向(rows)。垂直流向窗格具有较高的分界线，水平流向窗格具有较宽的分界线。Figure 1-10 是一个使用了数字图标的水平窗格例子。相反，垂直流向窗格中，第一列会包含1到4，第二列包含5到8。

Figure 1-10 Sample Horizontal Flow Pane



Description of "Figure 1-10 Sample Horizontal Flow Pane"

Gap属性来控制行和列直接的空间。padding属性来控制结点和网格窗格边缘的距离。Example 1-6 创建了一些列图标的水平流窗格。

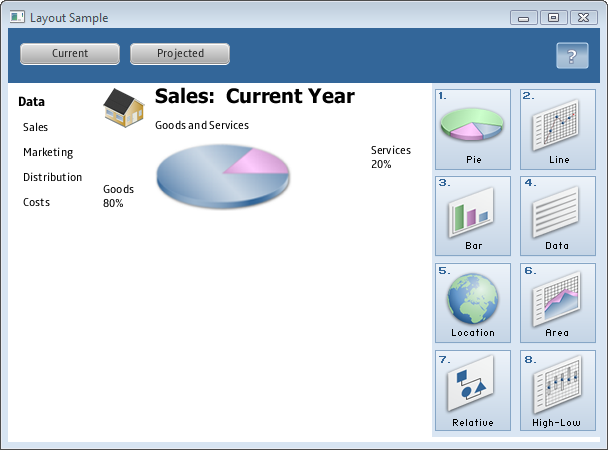
Example 1-6 Create a Flow Pane

[复制代码](javascript:)

1. FlowPane flow = new FlowPane();
2. flow.setPadding(new Insets(5, 0, 5, 0));
3. flow.setVgap(4);
4. flow.setHgap(4);
5. flow.setPrefWrapLength(170); // preferred width allows for two columns
6. flow.setStyle("-fx-background-color: DAE6F3");
7. ImageView pages[] = new ImageView[8];
8. for (int i=0; i<8; i++) {
9. pages[i] = new ImageView(
10. new Image(LayoutSample.class.getResourceAsStream("graphics/chart\_"+i+".png")));
11. flow.getChildren().add(pages[i]);
12. }
13. border.setRight(flow);       // Add to BorderPane from Example 1-2

Example 1-6 把流窗格放到了边框窗格的右部。结果是Figure 1-11 .

Figure 1-11 Flow Pane in a Border Pane



Description of "Figure 1-11 Flow Pane in a Border Pane"

**瓦片窗格TilePane**

瓦片窗格和流窗格很类似，TilePane 布局窗格中的所有结点都大小相同，放在网格中。结点可以水平放置(in rows) 或垂直放置(in columns)。水平放置的瓦片在宽度宽度方向而垂直的在高度方向。使用prefColumns和 prefRows属性来设置瓦片窗格的首选大小。

Gap属性来控制行和列直接的空间。padding属性来控制结点和网格窗格边缘的距离。

Example 1-7 创建了一个水平瓦片窗格，其效果和Figure 1-10 相同。

Example 1-7 Create a Tile Pane

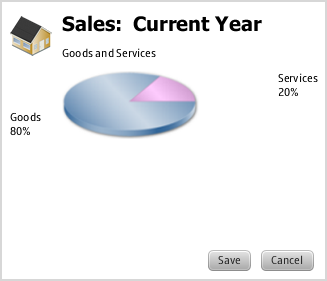
[复制代码](javascript:)

1. TilePane tile = new TilePane();
2. tile.setPadding(new Insets(5, 0, 5, 0));
3. tile.setVgap(4);
4. tile.setHgap(4);
5. tile.setPrefColumns(2);
6. tile.setStyle("-fx-background-color: DAE6F3");
7. ImageView pages[] = new ImageView[8];
8. for (int i=0; i<8; i++) {
9. pages[i] = new ImageView(
10. new Image(LayoutSample.class.getResourceAsStream("graphics/chart\_"+i+".png")));
11. tile.getChildren().add(pages[i]);
12. }

**锚窗格AnchorPane**

AnchorPane 布局窗格用来在窗格的上下左右中固定结点。当窗口大小改变时，结点会维持它们原来的相对位置。一个结点可以赋予多个位置，一个位置也可以赋予多个结点。Figure 1-12 是一个锚窗格，网格窗格在顶部，有两个按钮的HBox窗格在底部偏右。

Figure 1-12 Sample Anchor Pane



Description of "Figure 1-12 Sample Anchor Pane"

Example 1-8 照上面创建了一个锚窗格。

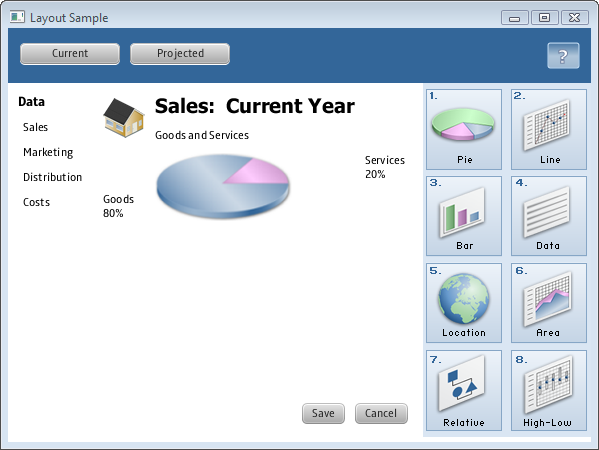
Example 1-8 Create an Anchor Pane

[复制代码](javascript:)

1. AnchorPane anchorpane = new AnchorPane();
2. Button buttonSave = new Button("Save");
3. Button buttonCancel = new Button("Cancel");
4. HBox hbox = new HBox();
5. hbox.setPadding(new Insets(0, 10, 10, 10));
6. hbox.setSpacing(10);
7. hbox.getChildren().addAll(buttonSave, buttonCancel);
8. anchorpane.getChildren().addAll(grid,hbox);   // Add grid from Example 1-5
9. AnchorPane.setBottomAnchor(hbox, 8.0);
10. AnchorPane.setRightAnchor(hbox, 5.0);
11. AnchorPane.setTopAnchor(grid, 10.0);
12. border.setCenter(anchorpane);       // Add to BorderPane from Example 1-2

Example 1-8 最后一行把锚窗格放在了边框窗格中间，取代了先前的结点。结果见Figure 1-13 .

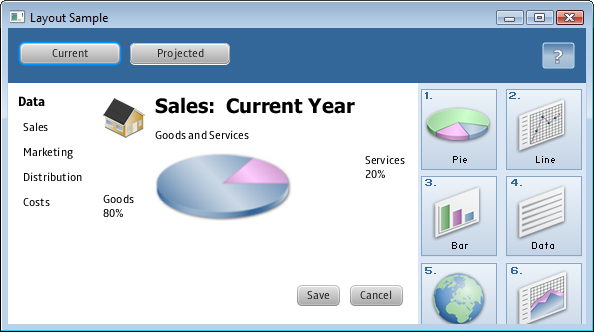
Figure 1-13 Anchor Pane in a Border Pane



Description of "Figure 1-13 Anchor Pane in a Border Pane"

当窗口大小改变时，结点会维持它们原来的相对位置。 Figure 1-14 在按钮在底部，当窗口变低时，按钮也向上移动了。

Figure 1-14 Resized Anchor Pane



# 第三十回 重置JavaFX2.0结点

使用JavaFX内置布局窗格最大的好处是窗格帮你管理结点的大小和对其性质。窗格改变大小时，结点大小十分改变要根据窗格的性质。注意不是所有的结点类 都可以改变大小。UI控件和布局窗格可以，但是形状、Text对象、 Group对象不可以，它们在布局中是刚性对象。如果你想要更多的控制控件大小，请直接指定其尺寸。布局窗格会根据你的设置来决定控件的大小。

**改变结点大小**

布 局窗格通过调用prefWidth(height)和 prefHeight(width) 方法查询结点的首选尺寸。默认地，UI控件根据其内容计算它们的首选尺寸。比如，Button 对象的尺寸是根据文本长度和标签中字体的尺寸（可能还有图片）计算的。一般的，计算出来的尺寸都是刚好够大以使标签能完全看见。

UI控件也提供根据典型用法的默认最小和最大尺寸。比如，Button 对象的最大尺寸是首选尺寸，因为不太可能让其任意大。然而，ScrollPane 对象的最大尺寸是不确定的，因为总是希望它们会随着内容增长。

既可以使用结点的默认尺寸，也可以随心而设置。比如，Figure 2-1 是边框窗格中一些按钮和列表视图的默认尺寸。

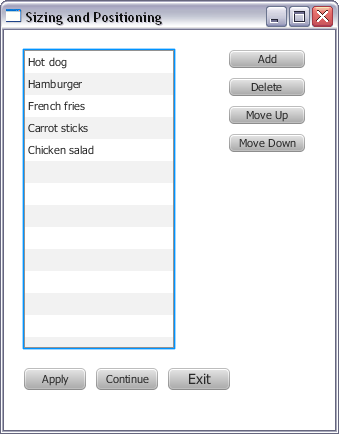
Figure 2-1 Computed Sizes



Description of "Figure 2-1 Computed Sizes"

如果想要 Figure 2-2 中的效果呢？里面UI控件的尺寸都是预设的。

Figure 2-2 Desired Sizes



Description of "Figure 2-2 Desired Sizes"

应用程序通常需要直接设置控件的最大、最小、首选尺寸。下面将讲解如何修改控件的外观来是自己满意。

**让按钮同样大**

要实现该目标，可以为每个按钮指定高度和宽度，然后将它们的首选尺寸改成最大的宽度和高度。一种更简单的方法是让布局窗格来完成该工作。布局窗格由想要达到的效果来决定。

**使用垂直盒子**

Figure 2-1 中的场景为按钮使用了一个VBox布局窗格且按钮尺寸是默认的。按钮已经具有相同高度了，所以只要改变其宽度即可。

Figure 2-2 中的场景使用一个VBox窗格来利用一下默认行为：让 VBox窗格的宽度和最宽元素的宽度相同。要让按钮都和 VBox窗格宽度相同，就要把每个 按钮的最大宽度设为Double.MAX\_VALUE常量，这样控件的变化就没有了限制。当然，把最大值指定为具体指也行，比如 80.0.

Example 2-1 在实现了任何让VBox窗格中的按钮都一样宽。

Example 2-1 Set a Column of Buttons to the Same Width

[复制代码](javascript:)

1. BorderPane root = new BorderPane();
2. root.setPadding(new Insets(20, 0, 20, 20));
3. Button btnAdd = new Button("Add");
4. Button btnDelete = new Button("Delete");
5. Button btnMoveUp = new Button("Move Up");
6. Button btnMoveDown = new Button("Move Down");
7. btnAdd.setMaxWidth(Double.MAX\_VALUE);
8. btnDelete.setMaxWidth(Double.MAX\_VALUE);
9. btnMoveUp.setMaxWidth(Double.MAX\_VALUE);
10. btnMoveDown.setMaxWidth(Double.MAX\_VALUE);
11. VBox vbButtons = new VBox();
12. vbButtons.setSpacing(10);
13. vbButtons.setPadding(new Insets(0, 30, 10, 25));
14. vbButtons.getChildren().addAll(btnAdd, btnDelete, btnMoveUp, btnMoveDown);
15. root.setright(vbButtons);

小提示：

把按钮列放在了边框窗格的右部就限制了最宽按钮的首选宽度。边框窗格的中部会尽量去填满所以的控件，所以如果把Vbox放在中间，它和按钮都会扩大。

**使用瓦片窗格**

Figure 2-1 中的场景为底部的按钮使用了一个HBox布局窗格且按钮尺寸是默认的。 按钮的款和搞都不同。

Figure 2-2 中的场景使用了一个水平TilePane布局窗格来利用其默认行为：每个细胞或瓦片都相同大小。 每个瓦片的尺寸都需要是瓦片窗格中最大结点的大小。

要让按钮和瓦片尺寸一样，把它们的最大宽度和高度设为Double.MAX\_VALUE常量。 Example 2-2 中实现了该目标。

Example 2-2 Set a Row of Buttons to the Same Size

[复制代码](javascript:)

1. Button btnApply = new Button("Apply");
2. Button btnContinue = new Button("Continue");
3. Button btnExit = new Button("Exit");
4. btnApply.setMaxSize(Double.MAX\_VALUE, Double.MAX\_VALUE);
5. btnContinue.setMaxSize(Double.MAX\_VALUE, Double.MAX\_VALUE);
6. btnExit.setMaxSize(Double.MAX\_VALUE, Double.MAX\_VALUE);
7. btnExit.setStyle("-fx-font-size: 14pt;");
8. TilePane tileButtons = new TilePane(Orientation.HORIZONTAL);
9. tileButtons.setPadding(new Insets(20, 10, 20, 0));
10. tileButtons.setHgap(10.0);
11. tileButtons.getChildren().addAll(btnApply, btnContinue, btnExit);

即使窗口改变大小瓦片也不会变，所以瓦片窗格中的按钮不会改变的。

**让结点保持首选尺寸**

当 舞台改变大小时，其中的布局窗格可能会给包含的控件产生过多或不足的空间。每个布局窗格都有其空间分派原则，根据的是控件最小、最大、首选尺寸。一般地， 最大尺寸是Double.MAX\_VALUE的话控件会扩张来填充空间而有限定大小的就不会。比如， ListView对象最大值不能确定。如果要限制其首选高度，可以设置为 Control.USE\_PREF\_SIZE 常量，见Example 2-3 .

Example 2-3 Set Maximum Height to Preferred Height

[复制代码](javascript:)

1. ListView<String> lvList = new ListView<String>();
2. ObservableList<String> items = FXCollections.observableArrayList (
3. "Hot dog", "Hamburger", "French fries",
4. "Carrot sticks", "Chicken salad");
5. lvList.setItems(items);
6. lvList.setMaxHeight(Control.USE\_PREF\_SIZE);

按钮默认只扩张到首选尺寸。但是如果最小宽度没有指定，可能会出现三个点(... ) 。要防止按钮比首选宽度小，把最小宽度设置为其首选宽度。见Example 2-4 .

Example 2-4 Set Minimum Width to Preferred Width

Button btnMoveDown = new Button("Move Down");

btnMoveDown.setMinWidth(Control.USE\_PREF\_SIZE);

控件的首选尺寸首先基于经过计算（内容）的值。可以覆盖其首选值为自己的选择。下面的语句覆盖了列表的宽度。

lvList.setPrefWidth(150.0);

**阻止改变大小**

如果不想要结点改变大小，把其最大、最小、首选值设为一样的。要只阻止宽度或高度改变，就把宽度或高度的限制设为相同。在Example 2-5 中，列表创建时要求了限制值相同，所以即使窗口改变其尺寸也不变。一个按钮也限制了宽度的三个值相同。

Example 2-5 Set Size Constraints to Prevent Resizing

[复制代码](javascript:)

1. ListView<String> lvList = new ListView<String>();
2. lvList.setMinSize(150.0, Control.USE\_PREF\_SIZE);
3. lvList.setMaxSize(150.0, Control.USE\_PREF\_SIZE;
4. Button btnDelete = new Button("Delete");
5. btnDelete.setMinWidth(80.0);
6. btnDelete.setPrefWidth(80.0);
7. btnDelete.setMaxWidth(80.0);

**文本对齐**

每个布局窗格都有自己的方法来对其结点。比如，HBox和 VBox 布局窗格中，结点是顶端对齐和左对齐的。TilePane和 FlowPane窗格中，结点是居中的。窗格自身默认是顶端对齐和左对齐的。

可以使用setAlignment()方法设置结点和窗格的对其方式。对其常量位于 javafx.geometry包中的 enum类型中。

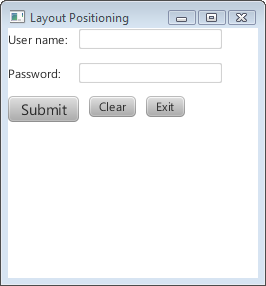
HPos - 指定水平对齐的值

Pos - 指定垂直和水平对齐的值。下划线左边指定的是垂直对齐值，下划线右边指定的是水平对齐值。比如Pos.BOTTOM\_LEFT要结点垂直方向在底部而水平方向在左边。

VPos - 指定垂直对齐的值。

Figure 2-3 是代码  Example 2-6 的效果.由于没有对齐限制，布局窗格在左上角。

Figure 2-3 Default Positions



Description of "Figure 2-3 Default Positions"

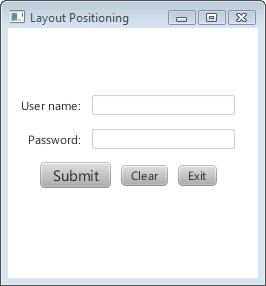
Example 2-6 Create a UI with Default Alignment

[复制代码](javascript:)

1. public void start(Stage primaryStage) {
2. GridPane grid = new GridPane();
3. grid.setHgap(10);
4. grid.setVgap(12);
5. HBox hbButtons = new HBox();
6. hbButtons.setSpacing(10.0);
7. Button btnSubmit = new Button("Submit");
8. Button btnClear = new Button("Clear");
9. Button btnExit = new Button("Exit");
10. btnSubmit.setStyle("-fx-font-size: 11pt;");
11. Label lblName = new Label("User name:");
12. TextField tfName = new TextField();
13. Label lblPwd = new Label("Password:");
14. PasswordField pfPwd = new PasswordField();
15. hbButtons.getChildren().addAll(btnSubmit, btnClear, btnExit);
16. grid.add(lblName, 0, 0);
17. grid.add(tfName, 1, 0);
18. grid.add(lblPwd, 0, 1);
19. grid.add(pfPwd, 1, 1);
20. grid.add(hbButtons, 0, 2, 2, 1);
21. Scene scene = new Scene(grid, 250, 250);
22. primaryStage.setTitle("Layout Positioning");
23. primaryStage.setScene(scene);
24. primaryStage.show();
25. }

如果要 Figure 2-4 中的效果呢？窗格在屏幕中央。

Figure 2-4 Desired Positions



Description of "Figure 2-4 Desired Positions"

下面就讲述任何覆盖默认位置设置。

**将网格居中**

要让 Example 2-6 中的网格居中，这样做：

grid.setAlignment(Pos.CENTER);

**对齐列中的控件**

在我们的理想布局中，标签要右对齐而文本框左对齐。要在网格中达到该目的，为每列使用ColumnConstraints类并设置水平对其方式。 Example 2-7 为 Example 2-6 中的网格定义了列属性。

Example 2-7 Define the Columns in the Grid

[复制代码](javascript:)

1. GridPane grid = new GridPane();
2. grid.setAlignment(Pos.CENTER);
3. grid.setHgap(10);
4. grid.setVgap(12);
5. ColumnConstraints column1 = new ColumnConstraints();
6. column1.setHalignment(HPos.RIGHT);
7. grid.getColumnConstraints().add(column1);
8. ColumnConstraints column2 = new ColumnConstraints();
9. column2.setHalignment(HPos.LEFT);
10. grid.getColumnConstraints().add(column2);
11. 使用VBox窗格也可以实现控件右对齐。照下面这样使用 setAlignment() 方法：
12. VBox vbox = new VBox;
13. vbox.setAlignment(Pos.CENTER\_RIGHT);

**按钮居中**

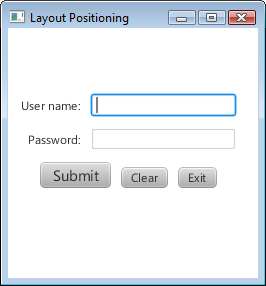
HBox中的按钮跨越了网格的列。下面的代码让按钮居中：

HBox hbButtons = new HBox()

hbButtons.setAlignment(Pos.CENTER);

HBox 的setAlignment() 方法居中了HBox窗格和其中的结点。要是更喜欢将 HBox窗格行居中而其中的按钮底部对齐，就像 Figure 2-5 .

Figure 2-5 Override Positions and Bottom-Justify the Buttons



Description of "Figure 2-5 Override Positions and Bottom-Justify the Buttons"

要这样安排，把 HBox 窗格放在一个只有一个细胞的内网格中，再把它放在外网格的第三行。把内网格设置为居中，把HBox窗格设为底部对齐，见 Example 2-8 .

Example 2-8 Center and Bottom-Justify the Buttons

[复制代码](javascript:)

1. HBox hbButtons = new HBox();
2. hbButtons.setSpacing(10.0);
3. hbButtons.setAlignment(Pos.BOTTOM\_CENTER);
4. hbButtons.getChildren().addAll(btnSubmit, btnClear, btnExit);
5. GridPane innergrid = new GridPane();
6. innergrid.setAlignment(Pos.CENTER);
7. innergrid.add(hbButtons, 0, 0);
8. grid.add(innergrid, 0, 2, 2, 1);

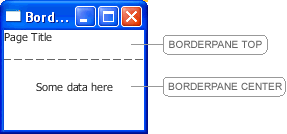
# JavaFX高级教程：JavaFX2.0的FXML语言

FXML是JavaFX 2.0新引入的。你可能会问"What is FXML?" 和"Is FXML for me?" FXML 是基于XML的一种声明性标记语言，用来定义应用的用户接口。FXML对于定义静态的布局很便利，诸如form, control, 和table。使用FXML也可以动态构造布局，不过要结合脚本。

FXML是一个优势是基于XML，所以多数开发者，尤其是web开发者和其他RIA平台的开发者会很熟悉它。另一个优势是FXML不是编译型语言，不需要 编译后才能看出变化。第三个好处是可以很简单的看到应用场景的结构。反过来，也就很简单地可以在组内进行合作开发用户接口。

要对比JavaFX和FXML,看图Figure 1   .构成该应用的场景包括一个边框布局，在它的顶部和中间各有一个标签。

Figure 1 Border Pane Simple Example



Description of "Figure 1 Border Pane Simple Example"

Example 1   是相应的JavaFX代码.

Example 1 JavaFX Scene Graph

BorderPane border = new BorderPane();

Label toppanetext = new Label("Page Title");

border.setTop(toppanetext);

Label centerpanetext = new Label ("Some data here");

border.setCenter(centerpanetext);

Example 2   是相应的FXML.

Example 2 FXML Scene Graph

<BorderPane>

    <top>

        <Label text="Page Title"/>

    </top>

    <center>

        <Label text="Some data here"/>

    </center>

</BorderPane>

展示FXML优势的最好方法是例子。本指南讲解如何创建Figure 2   中的登陆界面 .

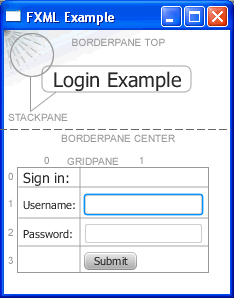
Figure 2 Login User Interface



Description of "Figure 2 Login User Interface"

开始之前先熟悉一下Figure 3   中的用户接口. 该接口使用了一个包含两部分的边框布局。顶区域包括一个堆栈布局，里面是用文本Label Example覆盖一副图片。   中部区域是一个网格布局，有标签、文本框、密码框和按钮。

Figure 3 Layout of Login User Interface



Description of "Figure 3 Layout of Login User Interface"

要创建该界面，我们需完成以下任务:

Prepare for This Tutorial

Set Up the Project

Set Up the Application Basics

Create the Properties File

Create the FXML File

Define a Border Pane Layout

Stack Text Over an Image

Add a Grid Layout and Controls

Add a Button Event

Use a Scripting Language

Use a Style Sheet

**准备**

本指南使用NetBeans IDE.请确保NetBeans IDE的版本支持JavaFX 2.0。

要完成本文，应熟悉用JavaFX编用户接口。尤其要知道场景的知识，因为FXML的语法结构和JavaFX的场景很像。

**建立工程**

第一步是建立JavaFX工程。

从File菜单选择   New Project   .

在  JavaFX   application category中，选择JavaFX FXML Application，点   Next   .

输入名称 FXMLExample点击  Finish   . NetBeans IDE打开了工程。有三个文件：FXMLExample.java, Sample.fxml, and Sample.java.

下载浅蓝色渐变的  fx\_boxback.jpg   图片到桌面，用来当背景。然后把它拖到fxmlexample文件夹下。

**建立基础**

每个 FXML 应用必须包含一些JavaFX代码，最少也有创建舞台和场景还有启动的代码。

打开FXMLExample.java   ,删除NetBeans IDE生成的代码，用下面的代码代替。Example 3   .

Example 3 FXMLExample.java

package fxmlexample;

[复制代码](javascript:)

1. import java.util.ResourceBundle;
2. import javafx.application.Application;
3. import javafx.fxml.FXMLLoader;
4. import javafx.scene.Parent;
5. import javafx.scene.Scene;
6. import javafx.stage.Stage;
7. public class FXMLExample extends Application {
8. @Override
9. public void start(Stage stage) throws Exception {
10. stage.setTitle("FXML Example");
12. Parent root = FXMLLoader.load(getClass().getResource("fxml\_example.fxml"),
13. ResourceBundle.getBundle("fxmlexample.fxml\_example"));
14. Scene scene = new Scene(root, 226, 264);
15. stage.setScene(scene);
16. stage.show();
17. }
18. public static void main(String[] args) {
19. launch(args);
20. }
21. }

作为一个JavaFX编程者，应该很熟悉创建舞台和场景。而然，这一行是FXML特有的:

Parent root = FXMLLoader.load(getClass().getResource("fxml\_example.fxml"), ResourceBundle.getBundle("fxmlexample.fxml\_example"));

FXMLLoader.load() 方法从文件   fxml\_example.fxml加载了目标层次并分配给变量root。 getResources参数增加了一个资源束 来为用户接口具体化，使得诸如定位任务更简单。后面将使用一个配置文件回到资源束并创建FXML源文件。 总之，场景已经创建，root变量是场景根元素。 FXML的根元素标记为场景的根。

**创建配置文件**

最佳实践是将字符串外部化，把它们放进配置文件。按照以下步骤创建用户登录界面的配置文件。

在Projects窗口，右击 fxmlexample文件夹，选择New   ,然后是Other   .

在New File对话框，点击Other   ,然后是Properties File   ,点  Next   .

输入fxml\_example作为名称，点Finish。 IDE打开了该文件，其扩展名是.properties。

输入资源名称和值，如 Example 4   .

Example 4 Resource names in fxml\_example.properties

loginExample=Login Example

signIn=Sign in:

userName=Username:

password=Password:

submit=Submit

**创建FXML文件**

现在创建fxml\_example.fxml文件并插入XML声明和导入语句。

在Projects窗口，右击Sample.fxm，选择Rename   .

输入fxml\_example后点OK   .

打开fxml\_example文件，删除自动生成代码，用下面的取代 Example 5   .

Example 5 Declaration and Import Statements

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.\*?>

<?import javafx.scene.image.\*?>

所有FXML文件必须以XML声明开始。它定义了XML版本(1.0)和编码类型(UTF-8).

在JavaFX中，类名称可以被完整描述（包括包名），或者是使用导入语句，见 Example 5   .只要你喜欢，你可以完整导入所以相关类。

**定义边框布局**

下面开始构建用户界面。在导入语句后面插入下面代码 Example 6   .

Example 6 Border Pane Layout

<BorderPane fx:controller="fxmlexample.FXMLExampleController" xmlns:fx="http://javafx.com/fxml">

     <top>

     </top>

     <center>

     </center>

</BorderPane>

本例中，场景根是BorderPane布局类，定义了两快区域，顶部和中部。

fx:controller属性定义了控制器文件，必须要声明在FXML根元素中。后面还会更多的了解控制器。

xmlns:fx="http://javafx.com/fxml"属性将命名空间映射到 <http://javafx.com/fxml>。命名空间也必须声明在FXML根元素中。该属性让你可以使用JavaFX API相关的FXML标签。

**在图像上覆盖文字**

现在，在边框布局的顶区域放置一个堆栈布局。该布局包括一个标签和被覆盖的图片，见 Figure 4   .

Figure 4 Top Region of Border Pane, Including Stack Pane



Description of "Figure 4 Top Region of Border Pane, Including Stack Pane"

堆栈布局的代码是 Example 7   .把它们插入在 <top>   和  </top>标签之间。

Example 7 Stack Text Over an Image

       <StackPane>

            <children>

                <ImageView>

                   <image>

                        <Image url="@fx\_boxback.jpg"/>

                   </image>

                </ImageView>

                <Label text="%loginExample" style="-fx-font: NORMAL 20 Tahoma;"/>

            </children>

        </StackPane>

StackPane布局将其孩子结点放进一个栈中，后面的会放在之前的上面。该布局可以很方便的把文本放在图片上。

<children>  标签把孩子结点加入到场景中，就像使用getChildren().add()方法。

Image类加载了图片 fx\_boxback.jpg，图片放在当前FXML文件路径下。 ImageView类用来显示图片。

Label类有一个文本属性来自资源 loginExample。在 FXML中，资源名称使用%来指定。加载时， FXML 加载器使用 loginExample  的值Login Example来取代它。

FXML定义风格时很像setStyle()方法的CSS风格。 在Example 7  中 Label类使用  style  标签来设置字型为normal，字体是Tahoma，字号是20。

另一种定义风格的方法和Java一样，加载样式表。使用样式表的话对于以后修改比较方便。看本文的 《使用样式表》  了解相关信息。

**添加网格布局和控件**

下面在边框布局的中部添加网格布局。使用网格布局可以在屏幕上垂直和水平的放置控件。见  Figure 5   .

Figure 5 Grid Pane in Center Region of Border Pane



Description of "Figure 5 Grid Pane in Center Region of Border Pane"

Example 8  是网格布局的代码，放在 <center> 和  </center>  标签中。

Example 8 Grid Layout with Controls

        <GridPane alignment="top\_center" hgap="8" vgap="8"

            style="-fx-padding: 40 0 0 0">

            <children>

                <Label text="%signIn"

                    style="-fx-font: NORMAL 14 Tahoma;"

                    GridPane.columnIndex="0" GridPane.rowIndex="0"/>

                <Label text="%userName"

                    GridPane.columnIndex="0" GridPane.rowIndex="1"

                    labelFor="$usernameField"/>

                <TextField fx:id="usernameField" prefColumnCount="10"

                    GridPane.columnIndex="1" GridPane.rowIndex="1"/>

                <Label text="%password"

                    GridPane.columnIndex="0" GridPane.rowIndex="2"

                    labelFor="$passwordField"/>

                <PasswordField fx:id="passwordField" prefColumnCount="10"

                    GridPane.columnIndex="1" GridPane.rowIndex="2"

                    onAction="#handlePasswordFieldAction"/>

                <Button fx:id="submitButton" text="%submit"

                    GridPane.columnIndex="1" GridPane.rowIndex="3"

                    onAction="#handleSubmitButtonAction"/>

                <Label fx:id="buttonStatusText"

                    GridPane.columnIndex="1" GridPane.rowIndex="4"

                    style="-fx-text-fill: #ff0000;"/>

            </children>

        </GridPane>

应 该对这些代码很熟悉了吧，不过还是要解释某些属性。GridPane.columnIndex   和  GridPane.rowIndex属性对应GridPane类的 setColumnIndex()   和setRowIndex()  方法。行列的值在网格开始时是0。比如，第一个Label控件的坐标是 (0,0)表示它在第一行第一列。或者说左上角。prefColumnCount 用于  TextField 和  PasswordField控件，来设置首选列宽是10 .

元素的fx:id属性会在文档的命名空间中创建一个变量，可以在后面 引用。变量引用时使用  $来标识。在  Example 8  中,  TextField的ID是 usernameField   .该ID分配给Label控件的labelFor属性， 为  TextField控件添加了一个label.

PasswordField和

Button控件有一个 onAction属性。该符号引用一个定制化方法，会在后面创建 FXMLExampleController类时创建。

尽管FXML创建界面很方便，但却不能实现行为。行为必须用Java代码实现（下一节）或者使用脚本，见 《使用脚本语言》.

**添加按钮事件**

现在创建控制器来管理按钮事件。本例演示如何把FXML和Java代码编写的事件控制器相关联。

在Projects窗口，右击 Sample.java，选择 Refactor--  Rename   .

输入FXMLExampleController后点击Refactor   .

打开FXMLExampleController.java删除代码，用下面的代替 Example 9   .

Example 9 FXMLExampleController.java

package fxmlexample;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.control.Label;

public class FXMLExampleController {

    @FXML private Label buttonStatusText;

    @FXML protected void handleSubmitButtonAction(ActionEvent event) {

        buttonStatusText.setText("Submit button pressed");

    }

    @FXML protected void handlePasswordFieldAction(ActionEvent event) {

        buttonStatusText.setText("Enter key pressed");

    }

}

@FXML注解用来标识非公开控制器成员和方法。

作为Java的补充，也可以使用其他编译型语言如Scala来实现控制器。.

现在可以运行了。输入内容点击Submit试试。

想要完整代码，请下载 FXMLExample.zip

.

**使用脚本语言**

除了使用Java创建控制器，可以使用任何提供JSR223-可编译脚本引擎的语言。比如 JavaScript, Groovy, Jython, 还有Clojure.下面是用 JavaScript编码FXML的。

在fxml\_example.fxml文件中，添加JavaScript声明到XML声明后。

<?language javascript?>

在

Button标记中，改变方法名称 :

onAction="handleSubmitButtonAction(event);"

更新标记PasswordField :

onAction="handlePasswordFieldAction(event);"

移除fx:controller 从  BorderPane标记中，在  <script>中添加js方法 ，见Example 10   .

Example 10 JavaScript in FXML

    <BorderPane xmlns:fx="http://javafx.com/fxml">

         <fx:script>

             function handleSubmitButtonAction() {

                 buttonStatusText.setText("Calling the JavaScript");

             }

             function handlePasswordFieldAction(event) {

                 buttonStatusText.text = "More JavaScript";

             }

         </fx:script>

当然也可以把js方法写进外部文件 (比如 fxml\_example.js)然后像下面这样引用:

<fx:script source="fxml\_example.js"/>

使用脚本编写FXML，调试时可能不会步入函数体。

**使用样式表**

使用内联样式，也可以添加样式表然后为结点设置关联。下面创建了样式表来定义网格布局和标签控件的样式。

创建样式表.

在Projects窗口，右击fxmlexample文件夹，选择New -- Other   .

在 New File对话框中，选择Other -- Cascading Style Sheet   , 然后  Next   .

输入 fxmlstylesheet后点击Finish   .

用下面的代码代替原来的代码 Example 11   .

Example 11 Contents of Style Sheet

@charset "utf-8";

/\*

    Document   : FXMLstylesheet.css

\*/

.grid-pane {

  -fx-padding: 80 0 0 0;

}

.label {

    -fx-font: normal 36px Tahoma;

}

打开 FXMLExample.java，把样式表加入到下面代码前面 s tage.show()   .scene.getStylesheets().add("fxmlexample/fxmlstylesheet.css");

打开 fxml\_example.fxml添加样式类.

为<String>元素添加导入语句。

<?import java.lang.\*?>

用下面代码代替GridPane Example 12   .

Example 12 Style Class for Grid Pane

<GridPane alignment="top\_center" hgap="8" vgap="8">

           <styleClass>

                <String fx:value="grid-pane"/>

            </styleClass>

用下面代码代替"Sign In" Label 。 Example 13   .

Example 13  Style class for Label

<Label text="%signIn"

    GridPane.columnIndex="0" GridPane.rowIndex="0">

         <styleClass>

               <String fx:value="label"/>

         </styleClass>

</Label>

使 用 <styleClass>  标签时，风格会应用到所有的类，除非它有自己的内联风格。所以Example 13  中的更改不仅应用到了 Sign In标签，也赋予了Username 和Password标签。但不会应用到Login Example标签，因为它有自己的内联样式，覆盖了其他风格

# 详解富客户端平台JavaFX2.0架构与框架

JavaFX 2.0平台是基于Java技术的富客户端平台。它使应用程序开发者更加容易的开发和部署跨平台的富互联网应用(RIA)。JavaFX 2.0文档包含了JavaFX 2.0所提供的功能的概述。

图 1描述了JavaFX 2.0平台的架构组件。后面的部分将对每一个组件进行逐一的描述。在JavaFX通用API的下面是用来运行JavaFX代码的引擎。这个引擎包括以下子 组件：JavaFX高性能图形引擎(Prism);新的更小但更有效率的窗体系统(Glass);媒体引擎和Web引擎。虽然这些组件不是包括在外的，但 是他们的描述能帮助你更好地理解JavaFX应用是如何运行的。

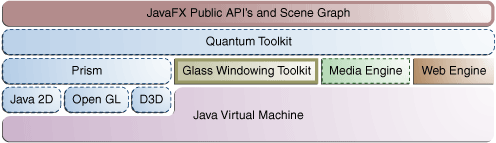


图1 JavaFX 2.0 架构图

**场景(Scene Graph)**

JavaFX的场景(Scene Graph)是JavaFX应用程序创建的开始点。它是一棵层次树，其每个节点都代表了一个应用程序UI的视觉元素。它可以处理输入并被渲染。

在 场景中的一个元素叫做节点(node)。每个节点都有其ID，样式类别和包围盒(不熟悉计算机图形学.....原文为bounding volume，我挺想翻译为基准线或者轮廓线的)。在场景中的每个节点都有一个唯一的双亲节点和零到多个孩子结点。一个节点还可以包含：

◆ 效果(Effect)，比如模糊或阴影效果

◆ 透明

◆ 变换

◆ 事件处理(比如鼠标、键盘或其他输入方式)

◆ 应用程序定义的状态

不同于Swing或者AWT，JavaFX的场景除了包含控制、布局管理器、图像和媒体外，也有基本图元，如矩形或者文字。

对于大多数用户来说，场景简化了用户实现UI的工作，特别是用户实现富UI的时候。在场景中绘制各种图形可以被快捷地完成，而且可以使用如XML文档一样的声明式方法。

javafx.scene API允许创建和定义许多种内容，如：

◆ 节点(Nodes)：2D或者3D的图形、图像、媒体、嵌入式web浏览器、文字、UI控件、图标、组和容器。

◆ 状态(State)：变换(节点的位置和方向)，视觉效果和其他内容的视觉状态。

◆ 动画(Animation)：场景对象在一段时间中的属性变化

◆ 效果(Effects)：可以修改场景节点的外观，如模糊、阴影和颜色调整。

你可以从Working with the JavaFX Scene Graph文档获取更多的信息。

**为实现JavaFX特征提供的Java API**

如 图1中的最顶层显示的那样，JavaFX 2.0平台包括一组完整的公共API。这些API为创建富客户端应用提供无可比拟的自由度和灵活性。JavaFX平台将Java平台的如身临其境般的媒体 功能融入直观的一站式综合开发环境之中。这些新的Java API实现了JavaFX的以下特点：

◆ 允许使用强大的Java语言特征，如泛型、注解和多线程技术

◆ 对于Web开发人员，JavaFX提供使用动态语言编写方式，如使用JRuby、Groovy和JavaScript

◆ 允许Java开发人员使用其他的系统语言去编写庞大复杂的JavaFX应用，如Groovy

◆ 允许使用与JavaFX脚本语言类似的绑定机制(binding)。这包括提供高性能的延迟绑定机制，绑定表达式和序列表达式，部分绑定重求值(不理解....)。其他语言(如Groovy)可以使用如JavaFX脚本语言一样的绑定库来实现语法级别的绑定。

◆ JavaFX扩展了Java集合库，加入了observable lists和maps。这允许应用程序将UI和数据模型进行连接，UI可以观测数据模型的改变并进行直接更新。

JavaFX 2.0 API和编程模型是JavaFX 1.x的延续。大部分JavaFX API都与Java直接相连。一些API已经根据JavaFX 1.x使用者的反馈进行了更新(包括性能和易用性的提升)，如布局管理(Layout)和媒体(Media)。而且JavaFX 2.0更加支持Web标准，如可以使用CSSS作为样式控制和使用无障碍规范建立ARIA。更多的Web标准也正在审核之中。

**图形系统**

在图1中用蓝色的部分是JavaFX图形系统。它是JavaFX场景层的下层细节实现。它为2D和3D的场景提供支持。当系统的图形硬件设备无法支持硬件渲染的时候，它提供软件渲染支持。

在JavaFX平台上实现了两个图形加速管道：

Prism：Prism处理渲染工作。他的渲染工作，包括3D渲染，都可以运行在硬件和软件渲染单元之上。它负责处理光栅和渲染JavaFX场景。基于所使用的设备的不同，可能有以下的渲染途径：

◆ 在Windows XP和Vista下使用DirectX 9

◆ 在Windows 7下使用DirectX 11

◆ 在Mac,Linux,嵌入式系统上使用OpenGL

◆ 如果硬件加速不可用，使用Java2D

如果可用时，所有的硬件加速途径都将被使用。但如果不可用，Java2D渲染途径将被使用。这是由于该渲染技术已经包含在JRE中。这可能在进行3D场景处理时很重要，但是一旦硬件渲染途径可用，性能就会比较好。

l Quantum Toolkit：Quantum Toolkit将Prism和Glass窗口工具结合在一起并使在整个JavaFX层次结构中上层组件可用。它也同时管理着与事件处理相对的渲染线程规则。

**Glass窗口工具**

在图1中间所示的Glass窗口工具是JavaFX 2.0图形栈中最底层的框架。他的主要责任是提供本地系统服务，比如说控制窗口，计时器或外观。它作为平台依赖层衔接着JavaFX平台和操作系统。

Glass 工具包也为事件队列提供管理功能。不同于AWT管理其自身所具有的事件队列，Glass工具包使用操作系统事件队列去管理线程。同样不同于AWT的 是，Glass工具包与JavaFX应用运行于同一线程。而在AWT中，原生AWT运行于一个线程而Java层次运行于另一个线程。这样会带来许多问题， 而JavaFX使用单一JavaFX线程来解决这类问题。

**线程**

任何时候，整个JavaFX系统都运行于两个以上的线程：

◆ JavaFX应用程序线程(JavaFX Application Thread)：这是JavaFX应用开发人员所要使用的主要线程。任何“活着的”场景(指场景是窗口的一部分)都只能从这个线程访问。然而，场景也可以 从后台线程创建，这使得开发人员可以从后台线程创建复杂的场景而同时保证“活着的”场景中的动画平滑而快速地播放。JavaFX应用程序线程与Swing 或AWT得事件派发线程(EDT)不同，所以当把JavaFX代码嵌入Swing应用中时，必须格外当心。

◆ Prism渲染线程(Prism Render Thread)：这个线程只处理渲染工作，而不干预事件派发线程。它允许第N帧播放时的同时，N+1帧正在被处理。这种能力对于提高并发处理能力有着极大 地好处，尤其是在现代的多处理器系统中尤为出色。Prism渲染线程也可以开启多个光栅线程去帮助分担渲染工作。

◆ 媒体线程(Media Thread)：这个线程运行于后台并通过JavaFX应用程序线程所使用的场景来同步最新的帧。

**脉搏(Pulse)**

脉 搏(Pulse，先这么翻译吧，等官方翻译出来再修改，其本质就是Repaint信号)是JavaFX应使用Prism去同步场景和场景中的元素状态的标 志。脉搏的最大速度为60帧/秒(FPS)，而且无论何时只要场景中的动画进行，那么就会发送脉搏通知。即使动画没有进行，当场景发生改变，也会发送脉搏 通知。例如，当一个按钮的位置发生了变化，那么就会安排一个脉搏通知。

当发送脉搏信号后，场景中的元素的状态就会根据渲染层进行同步。脉搏可以使应用程序开发者对于事件进行异步处理。这个重要的特点使得系统可以在一个脉搏来临前将事件进行批量处理。

布局管理和CSS也关联于脉搏信号。场景的众多变化可能导致多次布局或CSS的更新，这种更新会导致性能的降低。JavaFX系统会自动地在每次脉搏信号来临时将CSS或布局进行发送，从而避免系统性能下降。应用程序开发人员也可以手动触发布局或CSS变更通知的发送。

Glass窗口工具负责使用高性能的系统原生计时器来发送脉搏信号。

**媒体与图像**

JavaFX 媒体功能是通过javafx.scene.media API实现的。JavaFX提供显示和音频媒体功能。支持MP3，AIFF，WAV音频文件和FLV视频文件。JavaFX媒体功能由三个组件提供：媒体 对象代表着媒体文件，媒体播放器用来播放媒体文件，媒体视图是代表着媒体的节点。

在图1中绿色的部分是媒体引擎组件，这部分已经在 JavaFX 2.0中被完全重新设计了，增加了性能和稳定性，并提供了跨平台一致性。你可以阅读Incorporating Media Assets into JavaFX Applications文档来获取更多的信息。

**嵌入式浏览器**

JavaFX嵌入式浏览器是JavaFX新的UI组件，它通过API提供了 Web显示和访问的全部功能。图1中橙色的部分是基于WebKit的 Web引擎组件(WebKit是一款支持HTML5、CSS、Javascript、Dom和SVG的开源Web浏览器引擎)。它使开发人员可以在他们的 Java应用中实现以下功能：

◆ 渲染本地或远程的HTML内容

◆ 支持历史功能，并提供前进和后退导航

◆ 重新载入内容

◆ 向Web组件增加效果

◆ 编辑HTML内容

◆ 执行JavaScript命令

◆ 处理事件

这个嵌入式浏览器组件包括一下的类：

◆ WebEngine 提供基本的Web页访问功能

◆ WebView 封装了一个WebEngine对象，可将HTML内容放入应用程序场景中，并提供域和方法来应用效果和变换。它是Node类的子类。

你可以阅读Adding HTML Content to JavaFX Applications文档来获取更多的关于JavaFX嵌入式浏览器的细节。

CSS

在JavaFX中CSS提供了对于JavaFX中UI应用定制样式而无需改变代码的能力。CSS可以被异步地应用于任何JavaFX场景中的节点。JavaFX CSS样式也可以在运行期间被简单地赋予场景，并允许动态修改应用程序外观。

图2显示了同一应用程序在两个不同CSS样式下的效果：

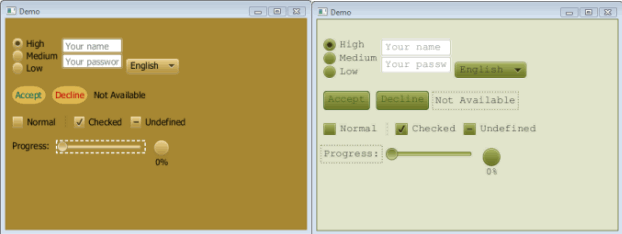


图2 CSS样式范例

JavaFX CSS基于W3C CSS ver2.1版本规范与现在正在使用的ver3版本的一部分。JavaFX CSS可以使用任何CSS解析器，甚至那些不支持JavaFX扩展的部分的。这样可以将为JavaFX和为其他目的(如HTML页)所写的CSS混合为一 个文件。由于JavaFX具有轻微不同的词法，所有的JavaFX属性名都以“-fx-”开头，包括那些可兼容于标准HTML的CSS。

你可以阅读 Skinning JavaFX Applications with CSS文章来获取更多的信息。

**UI控件**

JavaFX中UI控件可以通过使用场景中的节点的API来实现。它们从JavaFX平台的丰富的可视化特点中获得优势并在跨平台时有着轻微的不同。JavaFX CSS允许为UI控件定制主题和皮肤。

图3显示了一些现在支持的控件。新的控件，如TitlePane或Accordion，都将在JavaFX 2.0 SDK中被引入。这些控件位于javafx.scene.control包中。



图3 JavaFX UI控件范例

你可以阅读Using JavaFX UI Controls文档和API文档中的javafx.scene.control 包来获取更多的细节信息。

**布局**

布局容器或窗格可被用于在JavaFX应用程序场景中动态、灵活地管理UI控件。JavaFX布局API包括以下容器类，这些类可以自动进行布局：

◆ BorderPane将其内容节点分布于上、下、左、右、中间位置。

◆ HBox将其内容节点水平布局于一行中

◆ VBox将其内容节点垂直布局于一列中

◆ StackPane将其内容节点以从后到前的顺序布局于一个栈中

◆ GridPane允许开发人员创建灵活行列数目的格子，并在格子中存放内容节点

◆ FlowPane将其内容节点在水平方向或垂直方向上按顺序布局，并以特定宽度和高度的边界进行包裹。

◆ TilePane将其内容节点布局于一致大小的单元中。

◆ AnchorPane允许开发人员在上下左右或者中间创建锚节点。

在JavaFX应用中，想要实现希望的布局效果，可以综合使用不同的容器。

你可以阅读Working with Layouts in JavaFX文献或查看API中的javafx.scene.layout包来获取更多信息。

**2D和3D变换**

在JavaFX场景中的每个节点都可以使用javafx.scene.transform(原文这里应该是打错了)进行二维坐标变换：

◆ 平移变换(translate)：将一个节点从一个地方根据初始位置移动到另一个地方

◆ 拉伸变换(scale)：将一个节点根据缩放因子进行缩放

◆ 扭曲变换(shear)：将一个节点根据不垂直的x、y轴进行旋转。

◆ 旋转变换(rotate)：将一个节点根据场景中的某个点进行旋转

◆ 仿射变换(affine)：(若干balabala...看计算机图形学教材吧...)该变换不应被直接使用，而是应该使用Translate、Scale、Rotate、Shear来代替。

你可以阅读Applying Transformations in JavaFX文档或查看API中javafx.scene.transform包来获取更多信息。

**视觉效果**

在JavaFX场景中使用视觉效果可以提升JavaFX应用的观赏度。JavaFX效果主要是基于图像点的，因此他可以应用于场景中的一组节点，像图像一样渲染节点并对其应用指定效果。

在JavaFX中有一些视觉效果，下面是一部分：

◆ 阴影(Drop Shadow)：在给定的内容背后渲染阴影。

◆ 反射(Reflection)：在内容的下面渲染反射效果

◆ 光照(Lighting)：模拟光源照射在给定的内容上，可以将扁平的对象变得更加真实，3D化。

在Creating Visual Effects文档中有更多的如何使用视觉效果的例子。更多的视觉效果类参见API文档中的javafx.scene.effect包。

**部署**

JavaFX应用可以通过三种不同的方式被部署于桌面或在浏览器中：

◆ 独立(Standalone)：JavaFX应用被安装在本地磁盘中，并通过双击JAR文件来启动。当用户无需进行线上访问或应用没有线上功能时，这种模式是最理想的。

◆ 浏览器(Browser)：在这种模式下，JavaFX应用被嵌入一个Web页中并随着Web页被访问而自动启动。它可以跟Web页中JavaScript进行交互。

◆ 网页启动(Web Start)：这种模式下，用户下载应用程序并在本地运行。一旦被下载，用户在此之后就可以通过双击JNLP文件来启动应用。

你可以阅读Deploying JavaFX Applications document文档来查看更多关于三种模式和如何准备JavaFX应用部署的信息。

# [JavaFX的2.0常见问题合集](http://c-zhiwu.iteye.com/blog/1390246)

**1。 在JavaFX 2.0什么是新的？**JavaFX的2.0是在Java作为一个富客户端平台的演进的下一步。 它的目的是为企业和商业应用提供一个轻量级的，硬件加速Java UI平台。   
功能亮点：   
• 为JavaFX的Java API   
• 新的硬件加速图形管道   
•  FXML一个新的基于XML定义的用户界面的标记语言   
• 超过50的用户界面控制和图表CSS3的造型   
• 在Java应用程序的Web组件来呈现HTML和JavaScript内容   
• 秋千/ JavaFX的无缝集成到Swing应用程序的互操作性   
• 新媒体的稳定，一致的媒体播放引擎   
• 对于一个功能更完整的清单，看看功能概览   
  
**2。 我为什么要选择的JavaFX来开发应用程序？**  
JavaFX的设计，以解决现代企业的客户端应用程序的需要。 使用JavaFX的一些好处是：   
•  利用Java开发技能：JavaFX应用程序是完全在Java编程语言中，最广泛采用的技术来开发和部署应用软件，它利用基于标准的编程实践和设计模式之一。   
• 零没有时间在60：一个专注于简化的API设计使得开发复杂的用户界面，非常容易在Java编程语言。此外，JavaFX的发展FXML的一个脚本，定义 用户界面的基于XML的标记语言的用户界面。熟悉Web技术或其他标记的UI技术的开发者会发现FXML简单易学，功能强大的各种应用。   
•  创建无国界：JavaFX提供高性能的硬件加速的图形和媒体引擎的丰富的图形和媒体的API集，以简化身临其境的视觉应用的发展。   
•  保留在Java投资：JavaFX开发者可以保留现有的投资，在他们的JavaFX应用程序中重用Java库。 他们甚至可以通过Java本地接口访问本地系统的功能，无缝连接到基于服务器的Java EE中间件应用。   
•  熟悉的Java开发工具：开发人员可以使用自己喜爱的的Java开发工具，如NetBeans和Eclipse的集成开发环境，为开发JavaFX应用程序。  JavaFX开发的所有标准的IDE功能，包括语法高亮，自动完成，互动步调试和分析。   
• 部署在桌面上或在浏览器中：JavaFX应用程序可以部署为桌面应用程序或浏览器applets的Java浏览器插件的帮助。改进的Java浏览器插件提 供了一种安全的方式在浏览器中运行JavaFX应用程序。当部署在桌面上，JavaFX应用程序，得到更好的性能和执行本土的系统集成的好处。   
  
**3。 由JavaFX的支持哪些平台？**  
• 上目前有32位和64位版本的微软Windows XP，Windows Vista中，和Windows 7的JavaFX 2.0。   
• 一个用于Mac OS X的JavaFX 2.0的开发者预览Oracle技术网下载。 预计将达到在2012年上半年全面上市。   
•  Linux版本的JavaFX 2.0将跟随，预期在2012年下半年全面上市。   
•  JavaFX支持的Java SE 6和Windows上的Java SE 7中，并预计这些版本的Java SE支持在Mac OS X和Linux。 对于系统要求的详细清单，请参阅的JavaFX 2.0支持的系统配置。   
 **4。 开源的JavaFX平台吗？**  
2011年在JavaOne会议上，甲骨文公司宣布，他们打算提交一份建议，开源JavaFX平台作为一个新项目下的OpenJDK。 甲骨文打算最初作出贡献的JavaFX UI控件和相关的库;其他JavaFX组件将按照多个阶段。   
JavaFX的源代码将被发布在GPL下V2 Classpath的许可证，到OpenJDK中的其他项目类似。 甲骨文JavaFX运行时和SDK将继续Java二进制代码许可协议下发布。   
订阅项目邮件列表和访问源代码库<http://openjdk.java.net/> 。   
  
**5。 是JavaFX在Java SE？**  
号JavaFX运行时是一个单独的JRE下载，从现有的<http://java.com/javafx> 。 想尽可能顺利地为最终用户的JavaFX运行时安装的应用程序开发提供多种部署选项：   
• 摇匀JavaFX运行Java应用程序紧密集成。   
• 部署Web Start应用程序，将下载的最新版本的JavaFX运行时按需。   
• 使用Java部署工具包，以检测是否最终用户他的系统上安装最新版本的JRE和JavaFX运行时。   
对于JavaFX的部署方案的更多细节，请参阅 <http://download.oracle.com/javafx/2.0/deployment/deployment_toolkit.htm> 。   
  
**6。  JavaFX的更换为新的客户端用户界面的Java SE库秋千？**  
是 的，没有。 秋千可预见的将来将保持在Java SE规范的一部分，并包含在JRE。一方面，Swing是广泛地使用在现有的Java桌面应用程序，而是依赖于一个古老的建筑，这需要一定的专业知识和专 业化水平。另一方面，JavaFX的功能了一套现代的UI控件使用标准的CSS技术，可以剥皮。 虽然我们建议尽可能开发利用的JavaFXAPI的建设新的应用程序时，它可以在同一应用程序中使用Swing和JavaFX的，允许开发者来扩展现有的 Swing应用程序。   
  
**7。  JavaFX的音频和视频编解码器提供支持？**  
JavaFX提供了一套通用的API，可以很容易，包括JavaFX应用程序内的任何媒体播放。 媒体目前支持的格式如下：   
• 音频：MP3，AIFF格式包含未压缩的PCM格式的WAV包含未压缩的PCM   
• 视频：FLV含有VP6视频和MP3音频   
  
**8。 的JavaFX 2.0是否支持JavaFX脚本？**  
JavaFX Script是使用JavaFX 2.0开始，不再支持。 但是，您也可以使用在JVM上运行，其他脚本语言，比如Groovy或Scala，如果你喜欢使用声明性语言。   
  
**9。  JavaFX的以前的版本仍然可用？**  
产品的JavaFX 1.x的家庭将继续提供一个有限的时间内。 我们强烈鼓励，带来了以前版本的JavaFX开始规划其移植到JavaFX 2.0的市场应用的公司。   
 **10。 是JavaFX 2.0提供什么类型的牌照？**  
的JavaFX 2.0是相同的许可证和商业模式的JavaSE下，与再分配的条款，目前不允许第三方开发商发布JavaFX运行他们的应用程序（或SDK）（S）除外。 这个限制是由于在JavaFX的第三方技术，这是我们在更换过程中使用。我们预计将在几个月内解决这个问题，这样您就可以分发您的应用程序相同的方式，它 有可能多年的JRE（S）的JavaFX运行时库。   
 **11。 多久JavaFX的更新版本呢？**  
JavaFX是预计将遵循相同的Java SE更新时间表。bbs.heiba95.com  
 **12。 是JavaFX支持通过甲骨文的Java SE支持的产品？**  
目前还没有JavaFX的支持合同，虽然这种解决方案很可能在未来成为可用。 目前，我们建议您访问的JavaFX 论坛 ，或登录的JIRA提交bug报告或要求的新功能。   
  
**13。 我如何提交bug报告或请求功能呢？**  
如 果您认为在JavaFX中，你发现了一个缺陷，并希望提交一个bug报告，或者如果您想提出一个新的功能或增强功能的平台，你可以做访问，并创建一个帐户 HTTP： / / JavaFX的jira.kenai.com 。 一旦你登录后，选择“问题”，然后“创建期”从顶部的导航栏在页面上。下一步，选择“运行”项目，并选择您是否正在提交一个bug或功能要求。 给我们多的信息，你可以按照页面上的指示。如果我们需要其他任何东西，从你，我们会联系了解更多详情，但你可以告诉我们时间提前，更好！

# [javafx2.0 监听树和表的选择变化](http://c-zhiwu.iteye.com/blog/1345179)

Swing中的组件都有对应的选择模型 (SelectionModel)和一些对应的方法(形如:add/remove SelectionChangedListener)来监听组件选择的变化, 如JTree中有TreeSelectionModel和addTreeSelectionListener /removeTreeSelectionListener.

JavaFx也有同样的选择模型,可以通过改模型来监听选择的变化.下面是两个监听选择变化的示例(以TreeView为例, 其它组件(List Table等)同样可以使用这些方法):

方法一:

Java代码  [收藏代码](javascript:void())

1. tree.getSelectionModel().selectedItemProperty().addListener(new ChangeListener() {
2. //tree.getSelectionModel().selectedIndexProperty().addListener(new ChangeListener() {
3. @Override
4. public void changed(ObservableValue observable,Object oldValue, Object newValue) {
5. system.out.println("selection change");
6. }
7. });

方法二:

也通过监听javafx组件的FocusModel(焦点模型)来获取组件选择的变化的通知.

下面是一个示例:

Java代码  [收藏代码](javascript:void())

1. treegetFocusModel().focusedItemProperty().addListener(new ChangeListener<TreeItem>() {
2. @Override
3. public void changed(
4. ObservableValue<? extends TreeItem> observable,
5. TreeItem oldValue, TreeItem newValue) {
6. system.out.println("tree selection changed");
7. }
8. });

**1**   
顶

**0**   
踩

# [JavaFX2.0 Calendar 日历控件汇总](http://remotejavasky.iteye.com/blog/1337633)

JavaFX2的推出，对于RCP无疑又多了一种选择，其控件模型，CSS特性等，让Java开发者也可以很容易开发出modern的RCP应用。像 Flex，JQuery UI等RIA技术已实现的丰富组件，利用JavaFX可以很方便的重写。关于日历控件，Java平台下在Swing中就没有加入到标准组件中，虽然有几个 开源实现。然而在JavaFX2中，日历控件将会出现在标准组件中，只是很多耐不住寂寞的开发者已经带来了几个实现。   
**注:以下网址大都需要代理，贴一个**[**http://www.niudaili.com/**](http://www.niudaili.com/)

1. 基于WebView的（[http://jewelsea.wordpress.com/2011/12/05/jqueryui-based-datepicker-for-javafx）](http://jewelsea.wordpress.com/2011/12/05/jqueryui-based-datepicker-for-javafx%EF%BC%89)这充分利用现有资源jQuery UI DatePicker控件，根据JavaFX与JavaScript的交互特性，很方便的实现了日期选择的功能，但感觉使用WebView这个控件有点炮打苍蝇，尤其是对于NativeApp 可能就很不妥了。
2. 来自DZone的一个小示例（<http://www.dzone.com/links/r/creating_a_datechooser_control_with_javafx_20.html>）虽然不可以使用，但对学习JavaFX的控件模型，及CSS知识,了解一个简单的日历控件的实现倒是很有帮助的
3. 参考ExtJS Calendar的实现（<http://calendarcontrolinjavafx.blogspot.com/2012/01/calendar-control-in-javafx-20.html>） 该实现已经相对很完整了，自己修改一下完全可以满足需求了。只是其实现的代码没有完全遵循JavaFX2的行为 控件 皮肤的设计理念，然而它确实是目前实现的最好的一个，其作者也将加入JFXtras（JFXtras的日历控件实现还不完善，包括其它的一些针对 JavaFX2.0的控件）开源项目，相信这个日历控件实现的会更好。其使用方法也很简单，所贴链接的博客已有介绍，当然具体实现，主要看源码了（<http://code.google.com/p/javafx-calendar/>）。

    贴图看个效果先，这个是默认的样式   
      
    目前只支持一个主题，当然这个自己根据Ext的添加就是copy的事情了   
    

# [javafx2.0 修改控件默认鼠标键盘监听](http://c-zhiwu.iteye.com/blog/1392638)

JFX为所有空间提供了默认的鼠标键盘监听,以符合一般使用习惯. 比如表格/列表/树等控件中可以使用方向键选择表单元,空格/回车/F2开始编辑. 但有时候我们需要自定义的键盘监听来适合自己的需要, 比如在表中按下回车将自动提交编辑并移到右边单元.

屏蔽控件的默认鼠标键盘监听, 可以通过添加一个事件过滤器(EventFilter)来实现.下面的例子实现了屏蔽TableView的Enter键默认监听, 并在Enter键按下时选择表格的右边单元.

Java代码  [收藏代码](javascript:void())

1. table.addEventFilter(KeyEvent.KEY\_PRESSED, new EventHandler<KeyEvent>() {
2. @Override
3. public void handle(KeyEvent e) {
4. // e.consume(); //blocks all key bingdings
5. table.requestFocus();
6. KeyCode code =e.getCode();
7. if(code == KeyCode.ENTER){
8. try {
9. // your code here
10. new Robot().keyPress(java.awt.event.KeyEvent.VK\_RIGHT);//XXX  javafx  has no robot for key type...
11. } catch (Exception e1) {
12. }
13. e.consume();
14. }
16. }
17. });

事件过虑参考阅读:

<http://docs.oracle.com/javafx/2.0/api/javafx/scene/Node.html#addEventFilter%28javafx.event.EventType,%20javafx.event.EventHandler%29>

<http://docs.oracle.com/javafx/2.0/events/jfxpub-events.htm>

# [javafx2.0 获取TableView 正在编辑的单元TableCell](http://c-zhiwu.iteye.com/blog/1391411)

JFX使用CellFactory(Callback)来获取每个单元格 (TableCell)的控件, 因而表格(TableView)并没有直接提供获取正在编辑的单元格(TableCell)的方法, 仅仅支持获取当前正在编辑的单位的位置(TablePosition), 如果需要获取正在编辑器的单元格(如手动提交编辑(commitEdit())), 可以通过如下方法来实现:

Java代码  [收藏代码](javascript:void())

1. public class MyCell<S,T> extends TableCell<S, T> {
3. @Override
4. public void startEdit() {
5. if((!(isEditable())) || (!(getTableView().isEditable())) || (!(getTableColumn().isEditable()))){
6. return ;
7. }
8. //在这里记录正在编辑的单元格. 作为示例, 这里使用Cell对应的TableView的userData来记录.
9. getTableView().setUserData(this);
10. super.startEdit();
11. }
12. @Override
13. public void cancelEdit() {
14. super.cancelEdit();
15. getTableView().setUserData(null);
16. }
18. }

Java代码  [收藏代码](javascript:void())

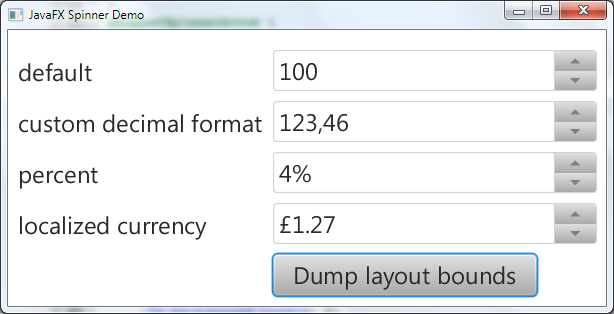
1. //外部获取TableCell
2. table.getUserData();

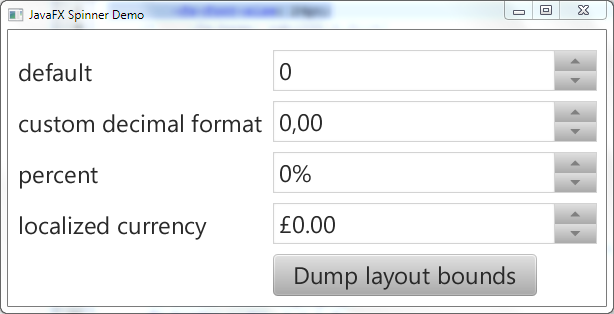
# [javafx2 : 支持使用微调(spinner)控制的数字的文本框(NemberTextField)](http://c-zhiwu.iteye.com/blog/1407930)

* 我 最近花了一些时间学习javaFX, 要更深入地理解新GUI包, 自定义控制器可能是一个比较好的方法. 由于我是开发财务软件的, 所以我当然希望javaFX中也有类似JFormattedTextField和JSpinner的控件. 这对我来说确实是个不错的选择.
* 这是我的控制器:

* 数字文本框(NumberTextField): 可以配置任意格式的数字;
* 微调控制器( NumberSpinner ): 可以使用键盘方向键或箭头按钮来控制数值;它也是控制器的一部分;

控制器及其示例可以在这里下载(可直接导入到netbeans,见附件).  示例中还包含一个css样式文件,  它用于控制Spinner的风格是直角或圆角.





### NumberTextField

NumberTextField 的实现很容易,以致我认为这算不上自定义控制器, 而仅仅是改变一个已存在的控制器的一些行为而已.  NumberTextField  扩展自JFX中的文本框(TextField), 添加一个使用BigDecimal(由于财务软件需要精确的类型)的NumberProperty作为模型, 并做一些格式化和解析处理. 就这样, 不复杂.

Java代码  [收藏代码](javascript:void())

1. package de.thomasbolz.javafx;
3. import java.math.BigDecimal;
4. import java.text.NumberFormat;
5. import java.text.ParseException;
6. import javafx.beans.property.ObjectProperty;
7. import javafx.beans.property.SimpleObjectProperty;
8. import javafx.beans.value.ChangeListener;
9. import javafx.beans.value.ObservableValue;
10. import javafx.event.ActionEvent;
11. import javafx.event.EventHandler;
12. import javafx.scene.control.TextField;
14. /\*\*
15. \* Textfield implementation that accepts formatted number and stores them in a
16. \* BigDecimal property The user input is formatted when the focus is lost or the
17. \* user hits RETURN.
18. \*
19. \* @author Thomas Bolz
20. \*/
21. public class NumberTextField extends TextField {
23. private final NumberFormat nf;
24. private ObjectProperty<BigDecimal> number = new SimpleObjectProperty<>();
26. public final BigDecimal getNumber() {
27. return number.get();
28. }
30. public final void setNumber(BigDecimal value) {
31. number.set(value);
32. }
34. public ObjectProperty<BigDecimal> numberProperty() {
35. return number;
36. }
38. public NumberTextField() {
39. this(BigDecimal.ZERO);
40. }
42. public NumberTextField(BigDecimal value) {
43. this(value, NumberFormat.getInstance());
44. }
46. public NumberTextField(BigDecimal value, NumberFormat nf) {
47. super();
48. this.nf = nf;
49. initHandlers();
50. setNumber(value);
51. }
53. private void initHandlers() {
55. // try to parse when focus is lost or RETURN is hit
56. setOnAction(new EventHandler<ActionEvent>() {
58. @Override
59. public void handle(ActionEvent arg0) {
60. parseAndFormatInput();
61. }
62. });
64. focusedProperty().addListener(new ChangeListener<Boolean>() {
66. @Override
67. public void changed(ObservableValue<? extends Boolean> observable, Boolean oldValue, Boolean newValue) {
68. if (!newValue.booleanValue()) {
69. parseAndFormatInput();
70. }
71. }
72. });
74. // Set text in field if BigDecimal property is changed from outside.
75. numberProperty().addListener(new ChangeListener<BigDecimal>() {
77. @Override
78. public void changed(ObservableValue<? extends BigDecimal> obserable, BigDecimal oldValue, BigDecimal newValue) {
79. setText(nf.format(newValue));
80. }
81. });
82. }
84. /\*\*
85. \* Tries to parse the user input to a number according to the provided
86. \* NumberFormat
87. \*/
88. private void parseAndFormatInput() {
89. try {
90. String input = getText();
91. if (input == null || input.length() == 0) {
92. return;
93. }
94. Number parsedNumber = nf.parse(input);
95. BigDecimal newValue = new BigDecimal(parsedNumber.toString());
96. setNumber(newValue);
97. selectAll();
98. } catch (ParseException ex) {
99. // If parsing fails keep old number
100. setText(nf.format(number.get()));
101. }
102. }
103. }

 NumberSpinner

NumberSpinner好像复杂一点. 它构建在NumberTextField 上, 并使用递增和递减按钮来改变文本框中数字的值, 每次改变步长为stepwidth.

stepwidth和NumberFormat的初始值在构造器中指定. 文本框和按钮的大小取决于文本的大小. 文本的大小可在.css文件中设置.

Java代码  [收藏代码](javascript:void())

1. package de.thomasbolz.javafx;
3. import java.math.BigDecimal;
4. import java.text.NumberFormat;
5. import javafx.beans.binding.NumberBinding;
6. import javafx.beans.property.ObjectProperty;
7. import javafx.beans.property.SimpleObjectProperty;
8. import javafx.event.ActionEvent;
9. import javafx.event.EventHandler;
10. import javafx.geometry.Pos;
11. import javafx.scene.control.Button;
12. import javafx.scene.input.KeyCode;
13. import javafx.scene.input.KeyEvent;
14. import javafx.scene.layout.HBox;
15. import javafx.scene.layout.StackPane;
16. import javafx.scene.layout.VBox;
17. import javafx.scene.shape.LineTo;
18. import javafx.scene.shape.MoveTo;
19. import javafx.scene.shape.Path;
20. import javax.swing.JSpinner;
22. /\*\*
23. \* JavaFX Control that behaves like a {@link JSpinner} known in Swing. The
24. \* number in the textfield can be incremented or decremented by a configurable
25. \* stepWidth using the arrow buttons in the control or the up and down arrow
26. \* keys.
27. \*
28. \* @author Thomas Bolz
29. \*/
30. public class NumberSpinner extends HBox {
32. public static final String ARROW = "NumberSpinnerArrow";
33. public static final String NUMBER\_FIELD = "NumberField";
34. public static final String NUMBER\_SPINNER = "NumberSpinner";
35. public static final String SPINNER\_BUTTON\_UP = "SpinnerButtonUp";
36. public static final String SPINNER\_BUTTON\_DOWN = "SpinnerButtonDown";
37. private final String BUTTONS\_BOX = "ButtonsBox";
38. private NumberTextField numberField;
39. private ObjectProperty<BigDecimal> stepWitdhProperty = new SimpleObjectProperty<>();
40. private final double ARROW\_SIZE = 4;
41. private final Button incrementButton;
42. private final Button decrementButton;
43. private final NumberBinding buttonHeight;
44. private final NumberBinding spacing;
46. public NumberSpinner() {
47. this(BigDecimal.ZERO, BigDecimal.ONE);
48. }
50. public NumberSpinner(BigDecimal value, BigDecimal stepWidth) {
51. this(value, stepWidth, NumberFormat.getInstance());
52. }
54. public NumberSpinner(BigDecimal value, BigDecimal stepWidth, NumberFormat nf) {
55. super();
56. this.setId(NUMBER\_SPINNER);
57. this.stepWitdhProperty.set(stepWidth);
59. // TextField
60. numberField = new NumberTextField(value, nf);
61. numberField.setId(NUMBER\_FIELD);
63. // Enable arrow keys for dec/inc
64. numberField.addEventFilter(KeyEvent.KEY\_PRESSED, new EventHandler<KeyEvent>() {
66. @Override
67. public void handle(KeyEvent keyEvent) {
68. if (keyEvent.getCode() == KeyCode.DOWN) {
69. decrement();
70. keyEvent.consume();
71. }
72. if (keyEvent.getCode() == KeyCode.UP) {
73. increment();
74. keyEvent.consume();
75. }
76. }
77. });
79. // Painting the up and down arrows
80. Path arrowUp = new Path();
81. arrowUp.setId(ARROW);
82. arrowUp.getElements().addAll(new MoveTo(-ARROW\_SIZE, 0), new LineTo(ARROW\_SIZE, 0),
83. new LineTo(0, -ARROW\_SIZE), new LineTo(-ARROW\_SIZE, 0));
84. // mouse clicks should be forwarded to the underlying button
85. arrowUp.setMouseTransparent(true);
87. Path arrowDown = new Path();
88. arrowDown.setId(ARROW);
89. arrowDown.getElements().addAll(new MoveTo(-ARROW\_SIZE, 0), new LineTo(ARROW\_SIZE, 0),
90. new LineTo(0, ARROW\_SIZE), new LineTo(-ARROW\_SIZE, 0));
91. arrowDown.setMouseTransparent(true);
93. // the spinner buttons scale with the textfield size
94. // TODO: the following approach leads to the desired result, but it is
95. // not fully understood why and obviously it is not quite elegant
96. buttonHeight = numberField.heightProperty().subtract(3).divide(2);
97. // give unused space in the buttons VBox to the incrementBUtton
98. spacing = numberField.heightProperty().subtract(2).subtract(buttonHeight.multiply(2));
100. // inc/dec buttons
101. VBox buttons = new VBox();
102. buttons.setId(BUTTONS\_BOX);
103. incrementButton = new Button();
104. incrementButton.setId(SPINNER\_BUTTON\_UP);
105. incrementButton.prefWidthProperty().bind(numberField.heightProperty());
106. incrementButton.minWidthProperty().bind(numberField.heightProperty());
107. incrementButton.maxHeightProperty().bind(buttonHeight.add(spacing));
108. incrementButton.prefHeightProperty().bind(buttonHeight.add(spacing));
109. incrementButton.minHeightProperty().bind(buttonHeight.add(spacing));
110. incrementButton.setFocusTraversable(false);
111. incrementButton.setOnAction(new EventHandler<ActionEvent>() {
112. @Override
113. public void handle(ActionEvent ae) {
114. increment();
115. ae.consume();
116. }
117. });
119. // Paint arrow path on button using a StackPane
120. StackPane incPane = new StackPane();
121. incPane.getChildren().addAll(incrementButton, arrowUp);
122. incPane.setAlignment(Pos.CENTER);
124. decrementButton = new Button();
125. decrementButton.setId(SPINNER\_BUTTON\_DOWN);
126. decrementButton.prefWidthProperty().bind(numberField.heightProperty());
127. decrementButton.minWidthProperty().bind(numberField.heightProperty());
128. decrementButton.maxHeightProperty().bind(buttonHeight);
129. decrementButton.prefHeightProperty().bind(buttonHeight);
130. decrementButton.minHeightProperty().bind(buttonHeight);
132. decrementButton.setFocusTraversable(false);
133. decrementButton.setOnAction(new EventHandler<ActionEvent>() {
135. @Override
136. public void handle(ActionEvent ae) {
137. decrement();
138. ae.consume();
139. }
140. });
142. StackPane decPane = new StackPane();
143. decPane.getChildren().addAll(decrementButton, arrowDown);
144. decPane.setAlignment(Pos.CENTER);
146. buttons.getChildren().addAll(incPane, decPane);
147. this.getChildren().addAll(numberField, buttons);
148. }
150. /\*\*
151. \* increment number value by stepWidth
152. \*/
153. private void increment() {
154. BigDecimal value = numberField.getNumber();
155. value = value.add(stepWitdhProperty.get());
156. numberField.setNumber(value);
157. }
159. /\*\*
160. \* decrement number value by stepWidth
161. \*/
162. private void decrement() {
163. BigDecimal value = numberField.getNumber();
164. value = value.subtract(stepWitdhProperty.get());
165. numberField.setNumber(value);
166. }
168. public final void setNumber(BigDecimal value) {
169. numberField.setNumber(value);
170. }
172. public ObjectProperty<BigDecimal> numberProperty() {
173. return numberField.numberProperty();
174. }
176. public final BigDecimal getNumber() {
177. return numberField.getNumber();
178. }
180. // debugging layout bounds
181. public void dumpSizes() {
182. System.out.println("numberField (layout)=" + numberField.getLayoutBounds());
183. System.out.println("buttonInc (layout)=" + incrementButton.getLayoutBounds());
184. System.out.println("buttonDec (layout)=" + decrementButton.getLayoutBounds());
185. System.out.println("binding=" + buttonHeight.toString());
186. System.out.println("spacing=" + spacing.toString());
187. }
188. }

 number\_spinner.css

 最 后, 控制器的样式可在css文件中设置. 我实现了圆角和直角两种风格(见上文截图). 你可以通过修改 #NumberField, #ButtonBox, #SpinnerButtonUp 和#SpinnerButtonDown 中的border/background-radiuses来切换不同的风格.

Java代码  [收藏代码](javascript:void())

1. .root{
2. -fx-font-size: 24pt;
3. /\*    -fx-base: rgb(255,0,0);\*/
4. /\*    -fx-background: rgb(50,50,50);\*/
5. }
6. #NumberField {
7. -fx-border-width: 1;
8. -fx-border-color: lightgray;
9. -fx-background-insets:1;
10. -fx-border-radius:3 0 0 3;
11. /\*    -fx-border-radius:0 0 0 0;\*/
12. }
13. #NumberSpinnerArrow {
14. -fx-fill: gray;
15. -fx-stroke: gray;
16. /\*        -fx-effect: innershadow( gaussian , black , 2 , 0.6 , 1 , 1 )\*/
17. }
18. #ButtonsBox {
19. -fx-border-color:lightgray;
20. -fx-border-width: 1 1 1 0;
21. -fx-border-radius: 0 3 3 0;
22. /\*    -fx-border-radius: 0 0 0 0;\*/
23. }
24. #SpinnerButtonUp {
25. -fx-background-insets: 0;
26. -fx-background-radius:0 3 0 0;
27. /\*    -fx-background-radius:0;\*/
28. }
29. #SpinnerButtonDown {
30. -fx-background-insets: 0;
31. -fx-background-radius:0 0 3 0;
32. /\*    -fx-background-radius:0;\*/
33. }

**结论**

从上面的例子可以看出在javaFx中自定义控制器并不困难. JavaFX自2.0版本以后作为一个纯粹的java API, 其比以前的任何版本更好地整合诸如groovy(BigDecimal的乐土)的语言. 这将是财务桌面应用的黄金组合.

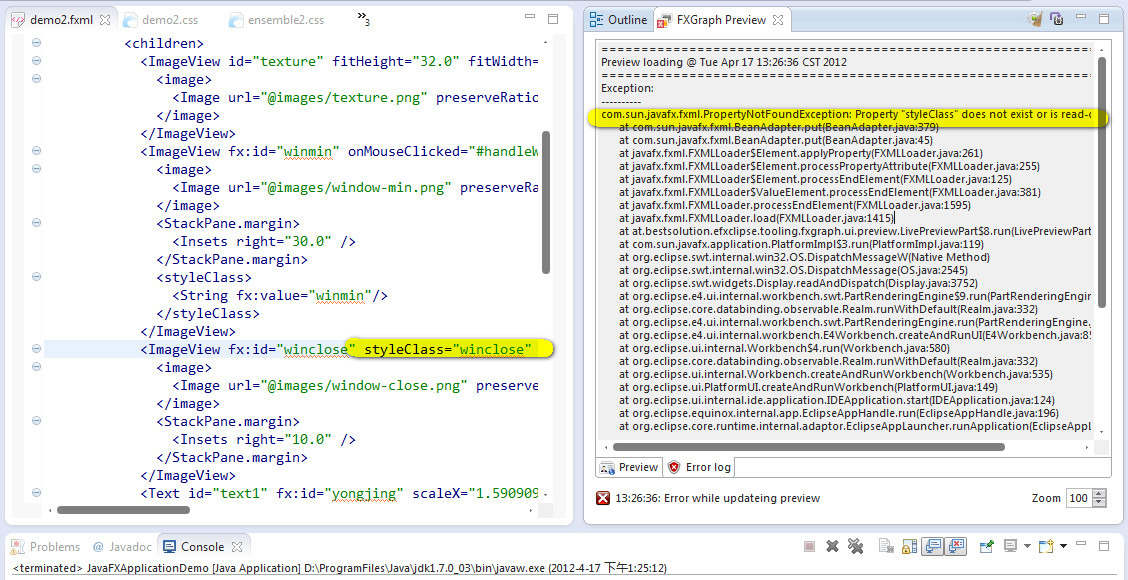
* [JavaFXSpinner\_1.zip](http://dl.iteye.com/topics/download/155b5b21-7cd9-3707-b346-d07b820870ac) (24.5 KB)
* 下载次数: 1

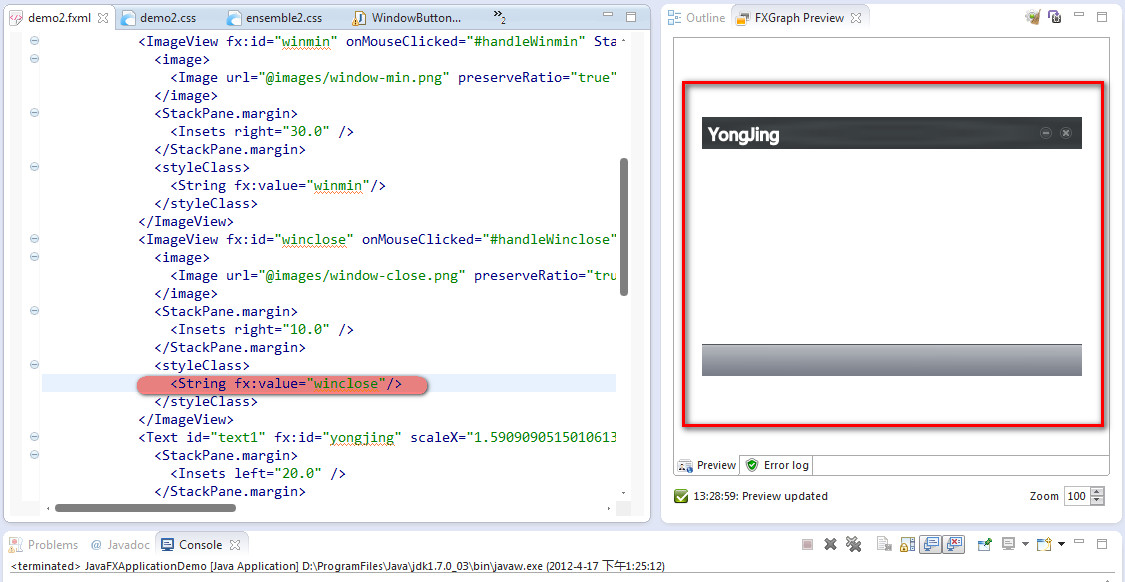
# [JavaFX Scene Builder使用总结](http://www.cnblogs.com/yinger/archive/2012/04/17/2453522.html)

持续更新我在使用JavaFX Scene Builder的过程中遇到的问题和解决办法！

我一般的开发步骤：首先是在Javafx Scene Builder中设计好界面的布局，将界面中要做的事情做完，包括定义控制器类，定义简单的style，定义事件方法，然后将整个FXML文件和用到的资 源文件（图片等）一起复制到Eclipse（安装了Javafx开发插件的版本，详情请看我的Javafx第一篇中所需要的软件）中，最后就是在程序中用 FXMLLoader去加载它显示出来，查看效果如何。

1.一般不要在JavaFX Scene Builder中添加样式！它会在相应的元素element中添加一个styleClass的属性，但是如果在程序中运行时总是会报一个错误，如果想用CSS来设置每个元素的样式的话建议手动操作！在FXML文件中要设置样式的元素添加一个子节点，如下所示即可！





但是纠结的是，如果在程序中改好了之后，重新在JavaFX Scene Builder中进行界面设计时，JavaFX Scene Builder会将你添加的styleclass删除掉，很多时候处理他们之间的

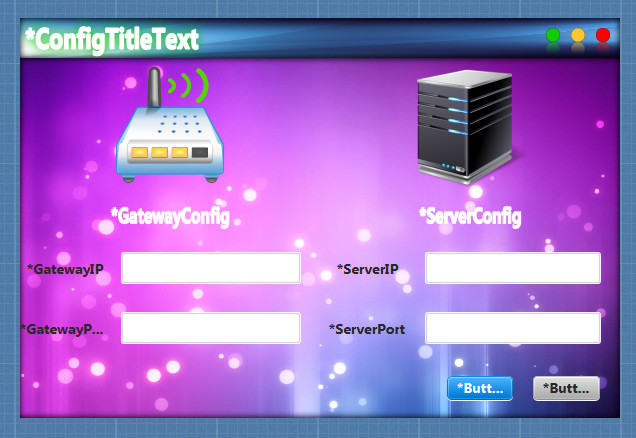
同步很困难，所以最好做好同步！

2.在使用JavaFX Scene Builder的过程中，经常会出现右边的属性面板出现假死现象，可以resize一下该面板就好了，这个是官网上发现了的已知的bug

3.在本地化的时候，一般我们都是在FXML文件中使用 %BundleText 来绑定特定的properties文件中的某个键值对，但是这里会出现一些问题。例如，将一个Label的text设置为

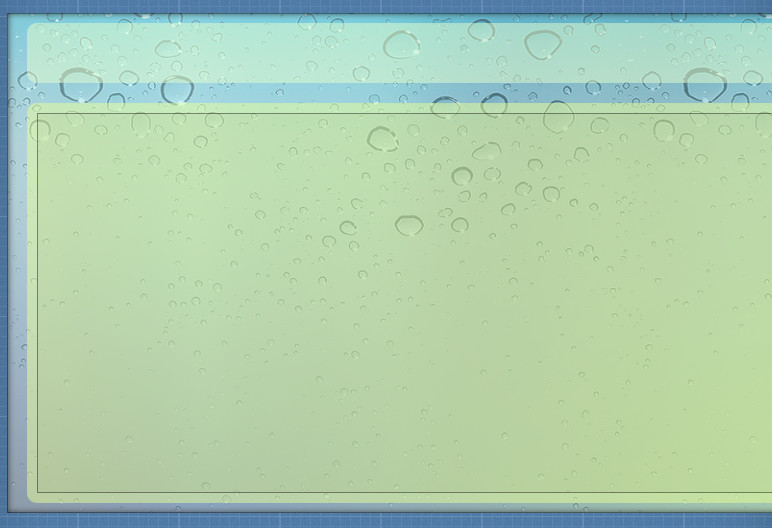
%Message，那么对应的FXML文件将生成的代码是 text="\%Message"， 因为%是特殊字符，所以要用反斜线来转义，但是这样一来，就会很麻烦！因为复制到Eclipse中时，Eclipse会报错！原因就在于用于转义的 “\”，它会发现这个斜线是多余的，所以我想了一个办法：%是需要转移的，但是\*是不需要的，而且界面是文本中一般不会出现\*，所以，在开发界面时如果是 要从属性文件中读取相应的文字的话，那么我就定义成 \*Message 形式的文本，然后复制到Eclipse中之后，再利用Eclipse的替换功能，注意不能是直接将所有的\*换成%，因为FXML文件头部的import元 素中也还存在\*，所以最好是使用text="%替换text="\*，保证万无一失！同理，如果你只是修改了Eclipse中的FXML文件，如果想要复制文件内容同步到Javafx scene builder中的话你就要将%替换成\*即可！呵呵，很方便吧！不过还是希望Javafx scenebuilder能够更加的智能！

还有一个需要注意的问题就是，如果你希望某个label或者text开始时为空字符串，（比如这个label就是显示错误信息的，但是初始时肯定是 没有的），一般是设置了text为“”，不过最好是能够使用“\*Message”，然后在properties文件中设置一下它的初值为空字符串，以免出 错！

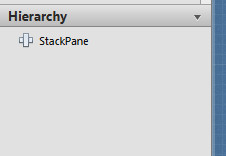


4.在Javafx scene builder中使用图片：点击File，选择Import，然后选中一张图片即可，工具会自动生成一个ImageView！注意，图片文件名最好不要有 空格，不然这次显示正常，下次可能就会出错！如果这张图片在当前的FXML文件所在的目录下或者子目录下，那么对应的url使用的是相对路径，一般是这样 子： url="@background-shadow.png" ，但是如果是在当前的FXML文件的父目录或者其他的目录中的话，生成的url是使用绝对路径！例如：url="file:/D:/yinger /My%20DBank/workspace\_fxml/apple.png"

同理，要想在JavaFx Scene Builder中使用视频或者音频等等，也是使用同样的方法！更加有用的是，你可以导入其他的FXML文件到当前的FXML文件中，这个十分方便我们实现 组件复用！工具会自动的生成相应的导入代码，例如：<fx:include source="PageView.fxml" />，除此之外，更新被导入的FXML文件中的内容会同步到这个文件中！

5.一般默认情况下新建的FXML文件的根节点都是AnchorPane，但是有时候我们并不想让根节点是它，比如想让根节点是 StackPane，但是如果在JavaFx Scene Builder中却是不允许删除根节点的，怎么办呢？很简单的，修改FXML源文件！直接将源文件中根节点的AnchorPane改成StackPane 即可，然后就可以看到效果了！



6.Javafx scene builder有一个很好的功能，点击View，选中Show Outlines可以查看布局的网格线！



7.点击Hierarchy面板右方的小小的向下的箭头，快捷显示fx:id/id/info功能，info一般是显示GridPane的表格大小（列\*行）



分类: [JavaFx](http://www.cnblogs.com/yinger/category/372015.html)

# [JavaFx 2.0总结](http://www.cnblogs.com/yinger/archive/2012/04/18/2456173.html)

1.如果是button的onAction事件，在FXML文件中定义方式： onAction="#processLogin"

在它对应的控制器中方法的签名是：@FXML protected void processLogin(ActionEvent event)   【也可以是private】

但是如果是其他的事件类型，一般形式是：在FXML文件中定义方式：onMouseClicked="#handleWinclose"

对应的方法签名是：@FXML  private void handleWinclose(MouseEvent event)   【也可以是protected】

2.Controller控制器类的public void initialize(URL location, ResourceBundle resources)方法是在FXML文件加载完成之后要调用的，所以可以在这里给界面中的一些组件

添加事件监听，或者初始化界面中某些组件的值

3.TilePane：在Javafx scene builder中使用TilePane时，右侧的属性面板出现了两个Alignment！一个是Alignment，另一个是Tile Alignment，注意区别！

首先，Tile这个单词是瓦片的意思，想想瓦片都是并排着一排一排的排列，而且都是一样大小的，这就是瓦片布局，TilePane！

Alignment：设置tilepane中tiles的对齐方式，这个类似在word中设置文字的对齐方式，一般是设置为TOP\_LEFT，表示从左上角开始放组件（tile）；

TileAlignment：设置每个瓦片（tile）内部的布局方式，我喜欢居中对齐，设置为CENTER。



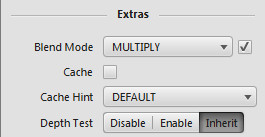
4.ScrollPane：它是可以显示滚动条的pane，但是可以通过设置BarPolicy来设置bar显示的方式：NEVER?ALWAYS?NEEDED?一般是用needed！

另外还有一个属性pannable，如果设置为true的话，那么通过移动scroll pane content就可以查看内容，滚动条会自动的移动到相应的位置！

5.BlendMode：blend的意思是叠加，blendmode的意思就是叠加的方式，这个有很多，常见的ADD，MUTIPLY，BLUE，GREEN，RED。。。这个很有用！

一般默认情况下，pane是有自己的背景颜色的，但是有时候我们希望pane没有它的默认的背景颜色，而是使用原来的背景色，如下面的右图所示，设置为MUTIPLY即可，

其他的一些也可以，不过要看情况而定选择你最合适的，因为有些模式会导致pane上的内容显示不太一样，因为它是叠加了的！

 6.

# [JavaFx2.0---Binding](http://www.cnblogs.com/yinger/archive/2012/04/20/2459803.html)

1.首先阅读文档，了解Javafx2.0中的属性和绑定：[Using JavaFX Properties and Binding](http://docs.oracle.com/javafx/2.0/binding/jfxpub-binding.htm%20)

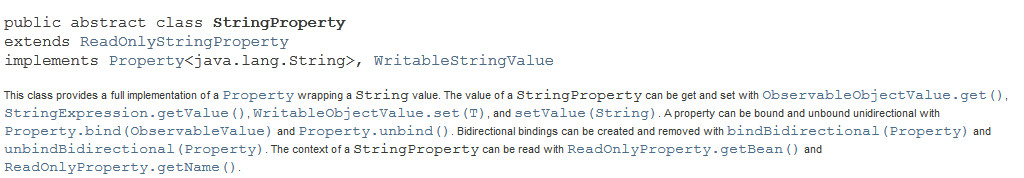
2.简单总结：

（1）JavaBean不再是以前的pojo了，Javafx添加了一系列的封装类，进一步封装了Java中的基本类型的封装类，使得它可以被绑定或者绑定，也就是它实现了Observable接口，具体请看API。



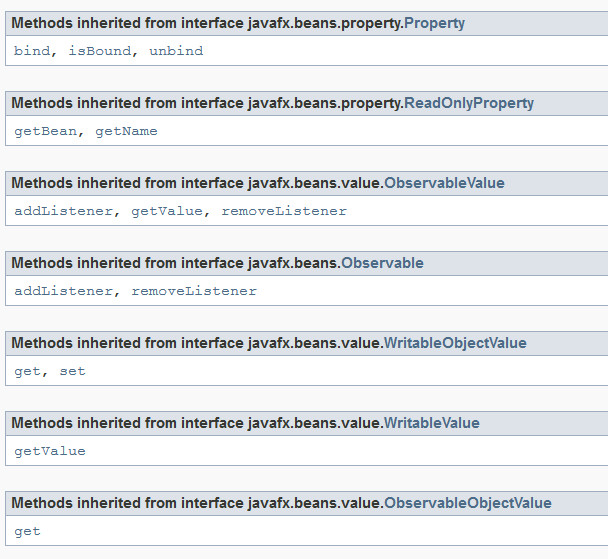
（2）上面的以Simple开头的是相应的property的简单实现类，所以在类（Javabean）中一般是使用初始化为simple...

（3）property都有一些方法用于绑定特定的对象，例如，绑定其他的property，或者其他的property组合而成的，例如，StringProperty



（4）除了使用bind方法绑定其他的property之外，它还可以添加listener，比如，当这个属性发生变化的时候就可以通知给其他的对象！

下面显示了StringProperty的实现的接口



3.实践

编写一个JavaBean，符合JavaFx2.0的规范

LightButton.java

[复制代码](javascript:void(0);)

package light;

import util.ProtocolUtil;

import javafx.beans.property.IntegerProperty;

import javafx.beans.property.SimpleIntegerProperty;

import javafx.beans.property.SimpleStringProperty;

import javafx.beans.property.StringProperty;

public class LightButton {

private IntegerProperty id = new SimpleIntegerProperty();// init here

private StringProperty name = new SimpleStringProperty();

private IntegerProperty state = new SimpleIntegerProperty(ProtocolUtil.STATE\_OFF);

// public LightButton(){//可以没有

//

// }

public LightButton(int id, String name) {

setId(id);

setName(name);

}

public void setState(int value) {

state.set(value);

}

public int getState() {

return state.get();

}

public IntegerProperty stateProperty() {

return state;

}

public void setId(int value) {

id.set(value);

}

public int getId() {

return id.get();

}

public IntegerProperty idProperty() {

return id;

}

public void setName(String value) {

name.set(value);

}

public String getName() {

return name.get();

}

public StringProperty nameProperty() {

return name;

}

}

[复制代码](javascript:void(0);)

LightView.fxml

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.\*?>

<?import java.util.\*?>

<?import javafx.geometry.\*?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.effect.\*?>

<?import javafx.scene.image.\*?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.paint.\*?>

<?import javafx.scene.text.\*?>

<?import light.Light?>

<AnchorPane id="AnchorPane" fx:id="lightAnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="150.0" prefWidth="150.0" xmlns:fx="http://javafx.com/fxml" fx:controller="light.LightViewController">

<children>

<StackPane id="StackPane" fx:id="lightStackPane" onMouseEntered="#handleShowButtons" onMouseExited="#handleHideButtons" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">

<children>

<ImageView fx:id="lightImageView" onMouseClicked="#handleLight" style="-fx-cursor:hand;" StackPane.alignment="TOP\_CENTER">

<effect>

<Reflection bottomOpacity="0.2" fraction="0.5" />

</effect>

<image>

<Image url="@light\_off.png" preserveRatio="true" smooth="true" />

</image>

<StackPane.margin>

<Insets top="20.0" />

</StackPane.margin>

</ImageView>

<Text id="text1" fx:id="textLightName" boundsType="VISUAL" scaleX="1.3438942202036481" scaleY="1.5467894807101856" stroke="#0c9900" strokeLineCap="ROUND" strokeLineJoin="ROUND" text="LightName" StackPane.alignment="TOP\_CENTER">

<effect>

<Reflection bottomOpacity="0.2" fraction="0.6" />

</effect>

<StackPane.margin>

<Insets top="120.0" />

</StackPane.margin>

</Text>

<Button fx:id="btnOpen" alignment="CENTER" contentDisplay="CENTER" defaultButton="true" onAction="#handleOpenLight" text="\*Open" StackPane.alignment="CENTER\_LEFT">

<StackPane.margin>

<Insets left="4.0" />

</StackPane.margin>

</Button>

<Button fx:id="btnClose" onAction="#handleCloseLight" text="\*Close" StackPane.alignment="CENTER\_RIGHT">

<StackPane.margin>

<Insets right="4.0" />

</StackPane.margin>

</Button>

</children>

</StackPane>

</children>

</AnchorPane>

[复制代码](javascript:void(0);)

LightViewController.java

[复制代码](javascript:void(0);)

package light;

import java.net.URL;

import java.util.ArrayList;

import java.util.List;

import java.util.ResourceBundle;

import util.ProtocolUtil;

import javafx.animation.FadeTransition;

import javafx.beans.value.ChangeListener;

import javafx.beans.value.ObservableValue;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.input.MouseEvent;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.StackPane;

import javafx.scene.text.Text;

import javafx.util.Duration;

import app.AppControllers2;

public class LightViewController implements Initializable {

private LightButton lightButton;

private Image lightOnImage;

private Image lightOffImage;

@FXML

private AnchorPane lightAnchorPane;

@FXML

private StackPane lightStackPane;

@FXML

private ImageView lightImageView;

@FXML

private Button btnOpen;

@FXML

private Button btnClose;

@FXML

private Text textLightName;

private List<Button> buttons = new ArrayList<Button>();

@Override

public void initialize(URL arg0, ResourceBundle arg1) {

lightOnImage = new Image(getClass().getResourceAsStream("light\_on.png"));

lightOffImage = new Image(getClass().getResourceAsStream("light\_off.png"));

lightButton = AppControllers2.getNewLightButton();

changeImage();

textLightName.setText(lightButton.getName());

// lightButton.nameProperty().bind(textLightName.textProperty());//(1)fails!

// java.lang.RuntimeException: A bound value cannot be set.

// textLightName.textProperty().bind(lightButton.nameProperty());//(2)ok

textLightName.textProperty().bindBidirectional(lightButton.nameProperty());// (3)ok

lightButton.nameProperty().bindBidirectional(textLightName.textProperty());// (4)ok

// (3)+(4) ok

// lightButton.nameProperty().bind(textLightName.textProperty());//(1)+(2) fails

// can not do this!StackOverflowError

lightButton.stateProperty().addListener(new ChangeListener<Number>() {

public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number newValue) {

changeImage();

}

});

buttons.add(btnOpen);

buttons.add(btnClose);

handleHideButtons(null);

}

private void changeImage() {

if (lightButton.getState() == ProtocolUtil.STATE\_OFF) {

lightImageView.setImage(lightOffImage);

} else {

lightImageView.setImage(lightOnImage);

}

}

@FXML

private void handleOpenLight(ActionEvent event) {

System.out.println("handle open");

lightButton.setName("open");

lightButton.setState(ProtocolUtil.STATE\_ON);

}

@FXML

private void handleCloseLight(ActionEvent event) {

System.out.println("handle close");

lightButton.setName("close");

lightButton.setState(ProtocolUtil.STATE\_OFF);

}

@FXML

private void handleLight(MouseEvent event) {

System.out.println("handle light");

lightButton.setName("name");

lightButton.setState(1 - lightButton.getState());

}

@FXML

private void handleShowButtons(MouseEvent me) {

System.out.println("handle show buttons");

for (Button button : buttons) {

FadeTransition f = new FadeTransition(Duration.millis(250), button);

f.setFromValue(0.2);

f.setToValue(0.8);

f.play();

}

}

@FXML

// TODO:bug here!if the mouse moves very fast,the buttons will not disappear!

private void handleHideButtons(MouseEvent me) {

System.out.println("handle hide buttons");

for (Button button : buttons) {

button.setOpacity(0);

}

}

}

[复制代码](javascript:void(0);)

然后用FXMLLoader加载fxml文件，显示在stage中，即可看到效果：



文字会发生变化，图片也会发生变化！其中文字使用的是bind，而图片使用的是addListener

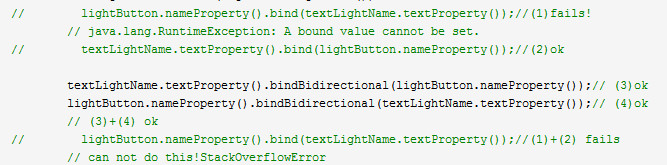


注意：请注意Controller类中的注释代码，这个很重要，绑定是有方向性的，一般是UI控件的值绑定到了bean的某个属性对象上。

并且要注意，不要以为 a.bind(b) 再加上 b.bind(a) 就可以是想双向绑定，这个是错误的，会发生栈溢出错误！例如（1）+（2）

代码片段反映了不同的情况下的情况，注意两个不同的方法：bind 和 bindBidirectional 方法，后者是由StringProperty提供的，可以实现双向绑定（貌似是，没有看过源码）

所以，使用（3）和使用（4）是一样的，写上一个就可以了。



4.但是，实际开发中，我们的bean对象中不仅仅只是int，float，double，string等等这些基本类型吧！那如果是一些比较复杂的Java类怎么进行绑定呢？

仔细查看API，不能发现还有一个ObjectProperty，它对应一个简单的实现类SimpleObjectProperty，看看它怎么使用吧，其实和其他的XXXProperty是一样的

下面是一个例子：目的是让rectangle的color和bill中的color属性一致！

[复制代码](javascript:void(0);)

package demo;

import javafx.beans.InvalidationListener;

import javafx.beans.Observable;

import javafx.beans.binding.Bindings;

import javafx.beans.binding.NumberBinding;

import javafx.beans.property.DoubleProperty;

import javafx.beans.property.ObjectProperty;

import javafx.beans.property.SimpleDoubleProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.scene.paint.Color;

import javafx.scene.shape.Rectangle;

public class BindingDemo {

public static void main(String[] args) {

Bill bill1 = new Bill();

Bill bill2 = new Bill();

NumberBinding total = Bindings.add(bill1.dpProperty(), bill2.dpProperty());

total.addListener(new InvalidationListener() {

public void invalidated(Observable arg0) {

System.out.println("now invalid!");

}

});

bill1.setDp(12);//valid->invalid

// System.out.println(total.getValue());// (1)//invalid->valid

bill2.setDp(13);//valid->invalid

System.out.println(total.getValue());// (2)//invalid->valid

// if (1)+(2)

// now invalid!

// 12.0

// now invalid!

// 25.0

//if (1) commented

// now invalid!

// 25.0

Rectangle rectangle = new Rectangle(20, 20, Color.RED);

System.out.println(rectangle.getFill());// Color[red=255,green=0,blue=0,opacity=1.0]

rectangle.fillProperty().bind(bill1.colorProperty());

bill1.setColor(Color.GREEN);

System.out.println(rectangle.getFill());// Color[red=0,green=128,blue=0,opacity=1.0]

// ChangeListener<Object> right and ChangeListener<Color> wrong

// rectangle.fillProperty().addListener(new ChangeListener<Object>() {

// public void changed(ObservableValue<?> observable, Object oldValue, Object newValue) {

// //when the rectangle color changes,this method will be invoked

// }

// });

}

}

class Bill {

private DoubleProperty dp = new SimpleDoubleProperty();

private ObjectProperty<Color> color = new SimpleObjectProperty<Color>();

public final Color getColor() {

return color.get();

}

public final void setColor(Color color) {

this.color.set(color);

}

public ObjectProperty<Color> colorProperty() {

return color;

}

public final double getDp() {

return dp.get();

}

public final void setDp(double dp) {

this.dp.set(dp);

}

public DoubleProperty dpProperty() {

return dp;

}

}

[复制代码](javascript:void(0);)

# 使用JavaFX2.0的控件

# 关于本教程

本教程覆盖了JavaFX API中内置的JavaFX UI控件。

本文包含以下章节：

* JavaFX的UI控件
* 标签
* 按钮
* 单选按钮
* 切换按钮
* 复选框
* 选择框
* 文本框
* 密码框
* 滚动条
* 滚动窗格
* 列表视图
* 表格
* 分隔条
* 滑块
* 进度条和进度指示器
* 超链接
* 工具提示
* HTML编辑器
* 标题面板和可折叠面板

每章提供了代码示例和应用程序来说明实际的UI控件的功能。你可以找到上表列出的应用程​的源文件和对应的NetBeans项目文件。

# JavaFX UI控件

         通过JavaFX API提供的JavaFX UI控件通过场景中的结点来构建，因此，控件可以使用JavaFX平台的丰富的视觉功能。由于JavaFX API是完全实现自Java的，因此你可以方便地将JavaFX UI控件集成到已有的Java应用程序中。

# 1 JavaFX2.0中支持的UI控件

         构建UI的类存在于API的javafx.scene.control包中。图1-1展示了使用包中一些控件构建的例子的截屏图。

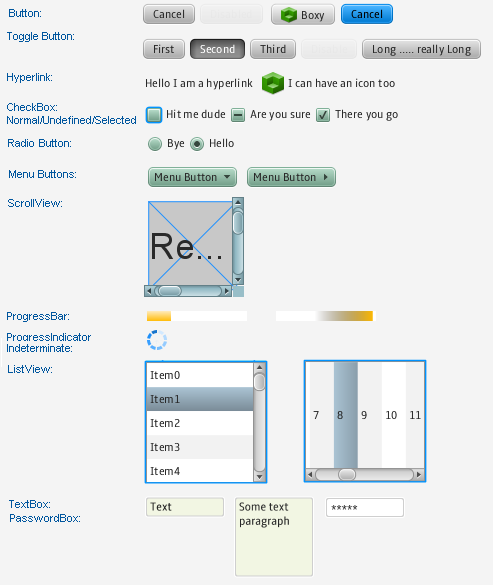


图1-1 JavaFXUI的例子

         该例子使用用以下的类：

* 标签
* 按钮
* 单选按钮
* 切换按钮
* 复选框
* 多选框
* 文本框
* 密码框
* 列表框
* 滚动面板
* 进度条
* 进度指示器
* 超链接

这个UI列表包含了以往你在开发Java客户端程序时可能用到的典型控件，无论如何，JavaFX2.2 SDK还包含了一些新的控件，比如选项面板和表格控件。

        图1-2展示了三个包含社交网络列表的选项面板程序的截屏图。列表可以上下收缩。

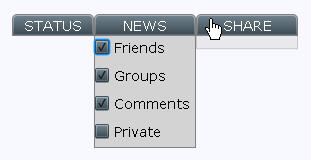


图1-2 选项面板

要查看完整的UI控件的列表，请参阅API文档。

UI控件类相比典型的用户交互类提供了额外的变量和方法来直观地实现及户交互，你可以通过层叠样式表（CSS）为控件指定特殊的风格。对一些不常用的任务，你可能需要通过继承控件类创建自定义的UI组件或通过皮肤接口为控件定义新的外观。

尝试JavaFX产品中提供的设定程序来测试控件的功能、表现和风格。

## 功能和效果

         因为javafx.scene.control包中的控件都继承自Node类，它们都可以使用场景渲染、动画、变换和动画过渡来宗合展示。

         考虑一个创建按钮的任务，它使用了反射，将它的透明度连续动态地从最大变到最小。

         图1-3通过动画展示了按钮的三种状态，左边的图是一个透明度为1.0的按钮，中的的按钮透明度为0.8，右边的透明度为0.5.



图1-3 动画按钮

通过使用JavaFX API，你可以使用很少的代码来实现这些功能。

例一创建并开始了一个无限期时间线，关键帧的时间间隔600毫秒，按钮的透明度从默认值（1.0）逐渐变为0.0。setAutoReverse方法可以反向变化。

例1：创建动画按钮

 import javafx.animation.KeyFrame;

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.animation.KeyValue;
2. import javafx.animation.Timeline;
3. import javafx.util.Duration;
4. import javafx.scene.control.Button;
5. import javafx.scene.text.Font;
6. import javafx.scene.effect.Reflection;
8. Button button = new Button();
9. button.setText("OK");
10. button.setFont(newFont("Tahoma", 24));
11. button.setEffect(newReflection());
13. final Timeline timeline = new Timeline();
14. //将循环次数设为永久
15. timeline.setCycleCount(Timeline.INDEFINITE);
16. timeline.setAutoReverse(true);
17. final KeyValue kv = newKeyValue(button.opacityProperty(), 0);
18. final KeyFrame kf = newKeyFrame(Duration.valueOf(600), kv);
19. timeline.getKeyFrames().add(kf);
20. timeline.play();

你还可以在上例中使用javafx.scene.effect包中的其他可视效果，如阴影、光照或者动态模糊。

## 用CSS设定UI控件的风格

         你可以通过层叠样式表（CSS）定制内置的UI控件的外观，在JavaFX中使用CSS与在HTML中基本相同，因为都使用相同的CSS规范。控件的可视状态通过例1-2的.css文件来定义。

         例1-2：在CSS文件中定义UI控件的样式

**[css]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. /\*controlStyle.css \*/
3. .scene{
4. -fx-font: 14pt "CambriaBold";
5. -fx-color: #e79423;
6. -fx-background: #67644e;
7. }
9. .button{
10. -fx-text-fill: #006464;
11. -fx-background-color: #e79423;
12. -fx-border-radius: 20;
13. -fx-background-radius: 20;
14. -fx-padding: 5;
15. }

         你可以通过例1-3所示的方法，用Scene类中的getStylesheets方法在应用程序中开启特定的样式。

         例1-3：应用CSS

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Scene scene = new Scene();
2. scene.getStylesheets().add("/uicontrolssample/controlStyle.css");

         因为HTML中的每个元素有自已的ID和类属性，因此JavaFX程序中的每个控件都可通过setId（）和setStyleClass（）方法获得ID和样式类。此外，你还可以直接在程序中定义控件的样式。

         例1-4中为切换按钮定义的-fx-base属性覆盖了在.css文件中为场景中所有控件统一定义的相应属性。

         例1-4：在JavaFX程序中定义切换按钮的样式

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ToggleButton tb3 = new ToggleButton ("I don't know");
2. tb3.setStyle("-fx-base: #ed1c24");

         图1-4展示了切换按钮加入到程序中时的样式。

http://hi.csdn.net/attachment/201111/3/0_1320290025yrWy.gif

图1-4 将CSS样式应用到切换按钮

## 图表

         作为对典型UI界面元素的补充，JavaFXSDK在javafx.scene.chart包中提供了现成的图表控件。该控件支持的图表种类为：面积图、柱状图、气泡图、折线图、饼图和散点图。图表可包含多个系列。

         图1-5展示了进口水果的饼图

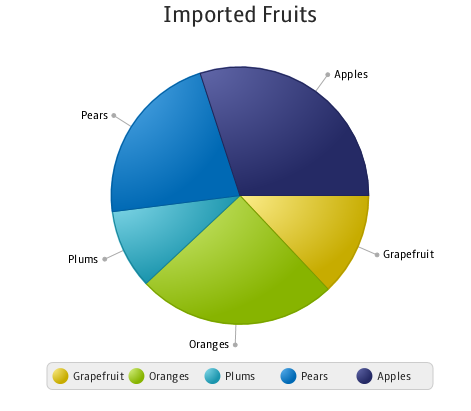


图1-5 饼图

         不同于其他的Java客户端工具包。使用JavaFXSDK，你可以在你的程序中仅用很少的代码就可以创建这样的图表。例1-5展示了这种特性。

例1-5：创建图表

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ObservableList<PieChart.Data> pieChartData =
2. FXCollections.observableArrayList(
3. newPieChart.Data("Grapefruit", 13),
4. newPieChart.Data("Oranges", 25),
5. newPieChart.Data("Plums", 10),
6. newPieChart.Data("Pears", 22),
7. newPieChart.Data("Apples", 30)
8. );
9. PieChart chart = new PieChart(pieChartData);
10. chart.setTitle("Imported Fruits");

以上的代码段包含以容：

* 通过PieChart.Data类生成数据
* 图表通过PieChart类的实例来生成
* 创建的图表包含了标题

## 将JavaFX2.0控件集成到Swing中

你可以将JavaFX2.0控件集成到现有的使用Swing工具包创建的Java客户端应用程序中。

         要将JavaFX内容集成到Swing程序中，其步骤为：

1、  将使用到的所有JavaFXUI控件逐一添加javafx.scene.Scene对象中，使用容器或组来管理它们。

2、  将Scene对象添加到Swing程序的内容窗格中。

如果你需要将JavaFX的单一控件定位到已有的Swing程序中，你也必须完成上述的两个步骤。

即使JavaFX控件被集成到了Swing程序中，JavaFX UI控件仍然使用Prism图形库来绘制，并且拥有该引擎所拥有的全站优势。

关于JavaFX和Swing的集成，请参考“在Swing中使用JavaFX”教程来了解更多信息。

# 2 标签控件

标签控件位于JavaFX API的javafx.scene.control包中，它扩展了标签类。使用标签可以显示文本元素。你可以将一段文本包装在一个特定的空间内，还可以为文本添加一个图像。

图2-1展示了三种普通的标签用例，左边的标签是一个带图像的文本，中间的展示了旋转后的文本，右边的标签绘制了一段文本。



图2-1 标签例子

## 创建标签

JavaFX API提供了三种标签类的构造器来创建标签，如例2-1所示。

例2-1 创建标签

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //空标签
2. Label label1 = new Label();
3. //纯文字标签
4. Label label2 = new Label("Search");
5. //带图标和文字的标签
6. Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
7. Label label3 = new Label("Search", new ImageView(image));

一旦在代码中创建了标签，你可以通过标签类的以下方法向标签对象增加文字或图像内容。

* setText（String text）方法—为标签对象指定文字内容
* setGraphic（Node graphic）—指定图标

setTextFill方法指定颜色来绘制标签的文本内容，研究一下例2-2，它首先创建了一个标签，然后加了一个图标、并指定了文本的绘制颜色。

例2-2 向标签增加图标和文本颜色

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Label label1 = new Label("Search");
2. Image image = newImage(getClass().getResourceAsStream("labels.jpg"));
3. label1.setGraphic(new ImageView(image));
4. label1.setTextFill(Color.web("#0076a3"));

将这段代码加到程序中后，将产生图2-2所示的标签效果。



图2-2带图标的标签

如果你同时为标签定义了文本和图像内容，你可以通过setGraphicTextGap方法来指定文本与图像间的间距。

另外，你还可以通过setVPos和 setHPos方法来改变标签在其布局区内的位置，或者通过setTextAlignment方法来设定文本元素的对齐方式。你还可以通过 setContentDisplay方法，再指定常量：LEFT，RIGHT，CENTER，TOP，BOTTPM来定义图像相对于文本的位置。

## 设定字体

对比图2-1与2-2中的搜索标签，可以发现图2-1中的标签字体较大。这是因为例2-2代码段没有为标签指定任何字体，它使用了默认的字体大小来绘制。

要为标签提供一个字体文字大小以替换默认大小，请使用Labeled类中的setFont。在代码片段[2-3](http://download.oracle.com/javafx/2.0/ui_controls/label.htm#CIHEJDHA)中将label1的文本大小设为30点，字体名称为Arial。对于label2，大小为32点，字体名称Cambria。

例2-3 应用字体设置

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //使用字体裁类的构造方法
2. label1.setFont(new Font("Arial", 30));
3. //使用字体类的方法
4. label2.setFont(Font.font("Cambria", 32));

或者，你还可以通过层叠样式表（CSS）为标签控件指定字体设定。研究一下例2-4就可明白如何为label1和label2指定相同的字体设定。

例2-4 设定标签样式

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. label1.setStyle("-fx-font: 30 arial");
2. label2.setStyle("-fx-font: 32 cambria");

**多行文本**

当你创建一个标签时，有时你必须将标签放置在一个比绘制区小的空间里，将文本打断（换行），以便以布局大小相适应，为setWrapText方法指定true值就可实现这种需求。如例2-5所示：

例2-5 开启文本换行

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Label label3 = new Label("A label that needs to be wrapped");
2. label3.setWrapText(true);

当label3被加入到程序的内容中，就会像图2-3所示的效果一样被绘制。



图2-3 换行标签

由于标签的布局大小不仅被它的宽度所限制，也被高 度所限制。当不可能绘制整个字串时，你可以指定它的表现形式。使用标签类的setTextOverrun方法，并选择有效的OverrunStyle类型 来处理不能被正确地绘制的那些文本。请参阅API文档来了解OverrunStyle类型的更多信息。

## 应用特效

虽然标签属静态文本，无法被编辑。但你可以使用可视化效果或者变换。例2-6的代码段将label2旋转了270度，并将变换了纵向位置。

例2-6 旋转标签

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Label label2 = new Label ("Values");
2. label2.setFont(new Font("Cambria", 32));
3. label2.setRotate(270);
4. label2.setTranslateY(50);

旋转和位移都是JavaFX API中典型的变换方法。此外，你还可以设置一种效果，当鼠标停留在标签上时会将标签放大。

例2-7中的代码段对label3使用了放大效 果。当标签的MOUSE\_ENTERED事件被触发时，比例因子1.5应用到了setScaleX和setScaleY方法，当用户将鼠标从标签上移开 时，MOUSE\_EXITED事件发生，比例因子重设为1.0，标签使用原来的大小来绘制。

例2-7 应用放大效果

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. label3.setOnMouseEntered(new EventHandler<MouseEvent>() {
2. @Override public voidhandle(MouseEvent e) {
3. label3.setScaleX(1.5);
4. label3.setScaleY(1.5);
5. }
6. });
8. label3.setOnMouseExited(new EventHandler<MouseEvent>() {
9. @Override public voidhandle(MouseEvent e) {
10. label3.setScaleX(1);
11. label3.setScaleY(1);
12. }
13. });

图2-4展示了label3的两种状态

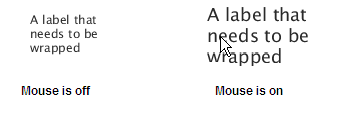


图2-4 放大标签

## 相关API

* Label
* Labeled

# 3 按钮控件

通过JavaFX API提供的Button类的方法，开发一员可以在用户按下一个按钮时来处理相应的动作。Button类继承自Label类，它可以显示文本、图像者两者相结合。图3-1展示了不同的效果。本章将学习如何创建这些不同类型的按钮。



图3-1 按钮的种类

## 创建一个按钮

         你可以在JavaFX程序中通过例3-1展示的三种构造方法来创建按钮控件。

         例3-1 创建按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //空按钮
2. Button button1 = new Button();
3. //使用特定本文为标题的按钮
4. Button button2 = new Button("Accept");
5. //使用标题和图标的按钮
6. Image imageOk = newImage(getClass().getResourceAsStream("ok.png"));
7. Button button3 = new Button("Accept", new ImageView(imageOk));

         由于Button类继承自Label类，你可以通过以下的方法来为未指定图标或标题的按钮指定按钮的内容。

* setText(String text)—为按钮指定标题
* setGraphic（Node graphic）—指定图标

例3-2展示了如何创建一个只有图标没有标题的按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Image imageDecline = newImage(getClass().getResourceAsStream("not.png"));
2. Button button5 = new Button();
3. button5.setGraphic(new ImageView(imageDecline));

当按钮被加入到程 序中，这段代码产生的按钮如图3-2.

http://hi.csdn.net/attachment/201111/3/0_13202903508ifN.gif

图3-2 带图标的按钮

在例3-2和图3-2中，图标是一个ImageView对象。然而，你可以使用其他的图形对象，例如，可以使用javafx.scene.shape包中的形状。当同时为按钮定义文本和图形后，你可以使用setGraphicTextGap方法来设定两者的间距。

Button类的默认皮肤区分了以下的可视化状态。图3-3展示了一个带图标的按钮的默认状态。

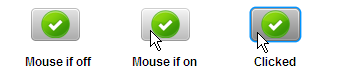


图3-3 按钮状态

## 赋予动作

按钮的主要功能是在单击时产生一个动作，使用Button类的setOnAction方法可以定义当用户单击按钮时将要完成的事情，例3-3展示了为button2定义动作的代码片段。

         例3-3 为按钮定义动作

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. button2.setOnAction(new EventHandler<ActionEvent>() {
2. @Override public voidhandle(ActionEvent e) {
3. label.setText("Accepted");
4. }
5. });

ActionEvent是一个被EventHandler处理的事件类型，EventHandler对象提供了handle方法来处理按钮触发的动作，例3-3展示了如何覆写handle方法，以更当用户按下button2时标签文本可以设为“Accepted”。

你可以使用Button类来根据需要设定很多事件—处理（event--handling）方法，以便引发特定的行为或应用可视化特效。

## 应用特效

因为Button类继承自Node类，你可以使作javafx.scene.effect包中的任何特效来增强按钮的可视化效果。在例3-4中，当button3触发了onMouseEntered事件时，该按钮使用了DropShadow（译注：一种阴影效果）效果。

例3-4 应用DropShadow特效

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. DropShadow shadow = new DropShadow();
2. //当鼠标位于按钮上时，增加阴影特效
3. button3.addEventHandler(MouseEvent.MOUSE\_ENTERED,
4. newEventHandler<MouseEvent>() {
5. @Override public voidhandle(MouseEvent e) {
6. button3.setEffect(shadow);
7. }
8. });
9. //鼠标离开时，去除特效
10. button3.addEventHandler(MouseEvent.MOUSE\_EXITED,
11. newEventHandler<MouseEvent>() {
12. @Override public voidhandle(MouseEvent e) {
13. button3.setEffect(null);
14. }
15. });

图3-4展示了当鼠标进入和离开按钮时的状态。



图3-4 有阴影特效的按钮

## 使用样式

改善按钮可视化效果的下一步是应用通过Skin类定义的CSS样式，在JavaFX2.0中使用CSS与在HTML中类似，因为两者都基于同样的规范。

你可以将样式定义在独立的CSS文件中，然后在应用程序中通过setStyleClass方法来开启，此方法继承自Node类，所有的UI控件都起可用。另外，你还可以直接通过setStyle方法直接为一个控件定义样式。例3-5和图3-4演示了这种方法。

例3-5 样式化一个按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. button1.setStyle("-fx-font: 22 arial; -fx-base: #b6e7c9");

-fx-font-size属性为button1设定字大属性，-fx-base属性覆盖了按钮的默认颜色。结果是，button1的颜色是淡绿色的，并且字体稍大，效果如图3-5所示：

http://hi.csdn.net/attachment/201111/3/0_1320290396l2ic.gif

 图3-5 用CSS样式化的按钮

## 相关API

* Button
* Labeled

# 4 单选按钮

单选按钮（RadioButton）类实现自切换按钮（ToggleButton）类，一个单选按钮要么选中，要么不选。一般来说，单选按钮都捆绑成一组，一次只能选择一个。这种行为可以和切换按钮区分开来，因为一组内的所有切换按钮均可以有未选取的状态。

图4-1展示了三个单选按钮例子的屏幕截图，例子中三个单选按钮属同一组。

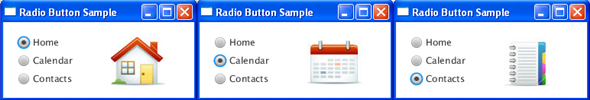


图4-1 单选按钮例子

通过研究后面的单节来学习如何在你的应用程序中实现单选按钮。

## 创建单选按钮

在JavaFX API的Javafx.scene.control包中，单选按钮类提供了两种创建单选按钮的构造方法，例4-1展示了两个按钮，无参数的构造方法用来创 建rb1，rb1的文本标题通过使用setText方法来实现，rb2的文本标题是通过相应的构造方法来定义的。

例4-1 创建单选按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //一个标题为空的按钮
2. RadioButton rb1 = new RadioButton();
3. //设定标题
4. rb1.setText("Home");
5. //一个有特定标题的按钮
6. RadioButton rb2 = new RadioButton("Calendar");

你可以明确地通过setSelected方法将一个单选按钮设定为选中状态，并将它的值设定为true。如果你需要检查一个单选按钮是否处于选中状态，可心使用isSelected方法。

由于单选按钮类是Labeled类的扩展类，你不但可以为它设定标题，还可以设定图像。使用setGraphic方法可以设定图像。例4-2演示了如何在应用程序中实现图像单选按钮。

例4-2 创建图像化单选按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Image image = newImage(getClass().getResourceAsStream("ok.jpg"));
2. RadioButton rb = new RadioButton("Agree");
3. rb.setGraphic(new ImageView(image));

**向组中添加单选按钮**

典型的单选按钮使用于组中，代表几个相互独立的选项。切换按钮类为所有单选按钮提供的参考将它们联系在一起，以便在某一时刻只能选择其中之一。例4-3创建了一个捆绑按钮组，然后创建了三个单选按钮，再把它们添加到组中，最后确定当程序启动后哪一个按钮被选中。

例4-3 创建一个单选按钮组

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final ToggleGroup group = new ToggleGroup();
2. RadioButton rb1 = new RadioButton("Home");
3. rb1.setToggleGroup(group);
4. rb1.setSelected(true);
5. RadioButton rb2 = new RadioButton("Calendar");
6. rb2.setToggleGroup(group);
7. RadioButton rb3 = new RadioButton("Contacts");
8. rb3.setToggleGroup(group);

当这些按钮通过布局容器放置并添加到程序中后，输出如图4-2所示。

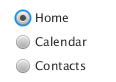


图4-2 绑定在一组中的三个单选按钮

## 处理单选按钮的事件

典型地，当组中的单选按钮被选中时，程序会执行一个动作。查看代码段4-4可以学习如何根据单选按钮的选取状态来改变图标。

例4-4 处理单选按钮的动作

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ImageView image = new ImageView();
2. rb1.setUserData("Home")
3. rb2.setUserData("Calendar");
4. rb3.setUserData("Contacts");
6. final ToggleGroup group = new ToggleGroup();
7. group.selectedToggleProperty().addListener(new ChangeListener<Toggle>(){
8. public voidchanged(ObservableValue<? extends Toggle> ov,
9. Toggle old\_toggle, Togglenew\_toggle) {
10. if(group.getSelectedToggle() != null) {
11. final Image image =new Image(
12. getClass().getResourceAsStream(
13. group.getSelectedToggle().getUserData().toString() +
14. ".jpg"
15. )
16. );
17. icon.setImage(image);
18. }
19. }
20. });

用 户数据被赋予了每个单选按钮，ChangeListener<Toggle>对象检查组中的被选按钮，它用 getSelectedToggle方法来标识当前哪个单选按钮被选中，然后通过getUserData方法提取用户数据，再以用户数据来构造一个供调用 的图像文件名。

例如，当rb3被选中时，getSelectedToogle方法返回“rb3”，getUserData方法返回“Contacts”。这样，getResourceAsStream方法接收值“Contacts.jpg”。程序的输出如图4-1年示。

## 为单选按钮申请焦点

在一组单选按钮中，默认第一个按钮拥有焦点，此时如果你通过setSelected方法将组中的第二个按钮设为选中状态，你将获得如图4-3所示的效果。

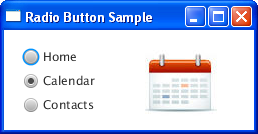


图4-3 默认焦点设置

第二个按钮被选中，但第一个焦点仍保留焦点。使用requestFocus函数可以改变焦点，如例4-5所示。

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. rb2.setSelected(true);
2. rb2.requestFocus();

当此函数被调用后，输出结果如图4-4所示。

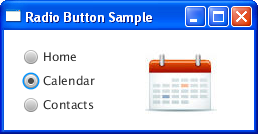


图4-4 为所选按钮设定焦点

## 相关API

* RadioButton
* Labeled

# 5 切换按钮

切换按钮类在JavaFX API中代表另个一种类型的按钮，两个或多个这种类型的按钮可以被捆绑在一组中，但同一时刻只能选择其中之一，或者都不选。图5-1是一个在组中捆绑了三个切换按钮的应用程序的截屏图。程序根据选中的切换按钮来确定颜色绘制一个矩形。

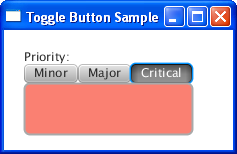


图5-1 三个切换按钮

## 创建一个切换按钮

你可以在应用程序中使用切换按钮类的三个构造函数中的任何一个来创建切换按钮，如例5-1所示。

例5-1 创建切换按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //无标题和图标的切换按钮
2. ToggleButton tb1 = new ToggleButton();
3. //有标题的切换按钮
4. ToggleButton tb2 = new ToggleButton("Press me");
5. //有标题和图标的切换按钮
6. Image image = newImage(getClass().getResourceAsStream("icon.png"));
7. ToggleButton tb3 = new ToggleButton ("Press me", newImageView(image));

切换按钮类是Labeled类的扩展，因此你可以为它指定标题、图标或两者。可以使用Lebeled类的setText和setGraphic为切换按钮指定文本和图像内容。

一旦你在代码中定义了切换按钮，你可以将它们绑定为一组，并设定一个特定的行为。

## 向组中添加切换按钮

实现切换按钮类与单选按钮类非常相似。然而，不同于单选按钮，组中的切换按钮不能强制选择组中的单一按钮。也就是说，当单击选中的切换按钮时会导致它变为非选中状态，但单击一个选中的单选按钮时则不会改变什么。

花点时间来研究一个例5-2的代码段。

例5-2 绑定组中的切换按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final ToggleGroup group = new ToggleGroup();
3. ToggleButton tb1 = new ToggleButton("Minor");
4. tb1.setToggleGroup(group);
5. tb1.setSelected(true);
7. ToggleButton tb2 = new ToggleButton("Major");
8. tb2.setToggleGroup(group);
9. tb2.setUserData(Color.LIGHTBLUE);
11. ToggleButton tb3 = new ToggleButton("Critical");
12. tb3.setToggleGroup(group);

例5-2创建了三个切换按钮并将它们设为一组，tb1调用了setSelected方法以便当程序启动时选中它。然而，你可以不选Minor这个切换按钮，这样程序启动后组中不会有任何按钮选选中。如图5-2所示。



图5-2 组中的三个切换按钮

一般来说，你用一组切换按钮来为每个按钮指定特定的行为。下一部份解释如何利用切换按钮来区分矩形的颜色。

## 设定行为

切换按钮类继承自Node类的setUserData方法帮助你指定任何选项值，在例5-3中，用户数据指示哪种颜色用来绘制矩形。

例5-3 为切换按钮设定用户数据

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. tb1.setUserData(Color.LIGHTGREEN);
2. tb2.setUserData(Color.LIGHTBLUE);
3. tb3.setUserData(Color.SALMON);
5. final Rectangle rect = new Rectangle(167, 50);
7. final ToggleGroup group = new ToggleGroup();
8. group.selectedToggleProperty().addListener(newChangeListener<Toggle>(){
9. public voidchanged(ObservableValue<? extends Toggle> ov,
10. Toggle toggle, Togglenew\_toggle) {
11. if (new\_toggle == null)
12. rect.setFill(Color.WHITE);
13. else
14. rect.setFill(
15. (Color)group.getSelectedToggle().getUserData()
16. );
17. }
18. });

ChangeListener<Toggle>对象检测组中的选择项，如果组中无选择项，则绘制一个白色的方框。如果选择的组中的项目，则后续会调用getSelectedToggle和getUserData方法，根据返回的颜色来绘制方框。

例如，如果用户选择了tb2按钮，getSelectedToggle.getUserData方法返回Color.LIGHTBLUE，结果如图5-3所示。



图5-3 使用选项按钮来绘制方框

查看ToggleButtonSample.java可以检查程序的完整代码。

## 定制选项按钮

你可以将CSS样式应用到切换按钮来改善这个应用程序，在JavaFX中，CSS的用法与在HTML中一样，因为两者都基于同样的规范。例5-4使用setStyle方法以改变切换按钮的-fx-base样式属性。

例5-4 将样式应用到切换按钮

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. tb1.setStyle("-fx-base: lightgreen");
2. tb2.setStyle("-fx-base: lightblue");
3. tb3.setStyle("-fx-base:salmon");

把这几行代码加到程序中后，产生的可视效果如图5-4所示。



图5-4 绘制切换按钮

你可能还想试试其他的CSS样式、或应用JAVAFXAPI中提供的动画、变换各可视化特效。

## 相关API

* ToggleButton
* ToggleGroup

# 6 选择框

CheckBox类提供了在应用程序中创建选择框的能力。虽然选择框看上去与单选按钮类似，然而不能将它们绑定到一个切换组以具备同时选择多个选项的能力（注：原文如此）。更多信息，请参考单选按钮和切换按钮章节。

图6-1展示了一个程序的截图。在这个程序里，使用了三个选择框来打开和关闭程序中工具条的图标。

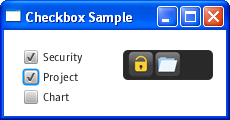


图6-1 选择框例子

**创建选择框**

例6-1 创建2个简单选择框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //第一个没有标签
2. CheckBox cb1 = new CheckBox();
3. //另一个有文字标签
4. CheckBox cb2 = new CheckBox("Second");
5. cb1.setText("First");
6. cb1.setSelected(true);

一旦创建了选择框，你可以通过JAVAFXAPI中提供的方法来修改它。在例6-1中，setText方法定义了c1的标签，setSelected方法用来将cb1的状态设为选中，以便在程序开始时使cb1这个选择框选中。

## 定义状态

选择框可以已定义或未定义，当处于已定义状态时， 使用以选取或不选它。然而，当选择框未定义时，它不能被选择或不选择。可使用CheckBox类的setSelected和setDefined两个方法 组给来设定选择框的状态。表6-1展示了根据它的DEFINED和SELECTED属性来决定的选择框状态。

表6-1 选择框的状态

|  |  |
| --- | --- |
| **Property Values** | **Checkbox Appearance** |
| DEFINED = true  SELECTED = false | http://hi.csdn.net/attachment/201111/3/0_1320292511WA6D.gif |
| DEFINED = true  SELECTED = true | http://hi.csdn.net/attachment/201111/3/0_13202925217F93.gif |
| DEFINED = false  SELECTED = true/false | http://hi.csdn.net/attachment/201111/3/0_13202925346fl1.gif |

**设定行为**

当选择框用来代表UI要素混合的三种状态（例如 “YES”、“NO”、“NOT”）时，你可能需要使应用程序的选择框具备三种状态。选择框的“ALLOW\_TRI\_STATE”属性决定了选择框是否能 在三种状态（“选中”、“未选中”、“未定义”）中循环。如果该属性为true，则选择框可在三种状态中循环，否则控件只能在两种状态中循环。后续部份描 述的程序构造了三个选择框，并且只打开了两种状态。

例6-2的代码片段创建了三个选择框，因此如果一个选择框被选中了，则相应的图标会出现在工具条上。

例6-2 设定选择框的行为

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final String[] names = new String[]{"Security","Project", "Chart"};
2. final Image[] images = new Image[names.length];
3. final ImageView[] icons = new ImageView[names.length];
4. final CheckBox[] cbs = new CheckBox[names.length];
6. for (int i = 0; i < names.length; i++) {
7. final Image image = images[i] =
8. newImage(getClass().getResourceAsStream(names[i] + ".png"));
9. final ImageView icon = icons[i] =new ImageView();
10. final CheckBox cb = cbs[i] = newCheckBox(names[i]);
11. cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
12. public voidchanged(ObservableValue<? extends Boolean> ov,
13. Boolean old\_val, Booleannew\_val) {
14. icon.setImage(new\_val? image : null);
15. }
16. });
17. }

Names数组使用了一个for循环来创建选择框 数组及相应的图标数组。例如，第一个选择框cbs[0]被赋予了“Security”标签。同时，当第一个图标对应的图像创建后，image[0]接收 “Security.png”作为一个文件名提供给getResourceStream方法。如果特定的选择框被选中时，对应的图像被赋予相应的图标。如 果选择框未选中，图标接收一个null值，相应的图标不会被绘制。

图6-2展示了“Security”和“Chart”选项框被选中、“Project”选项框未选中的程序的情况。



图6-2 赋予动作的选择框的程序

## 定制选项框

图6-2中的选择框设为CheckBox类的默认外观，你可以通过例6-3所示的方法来通过setStyle方法改变选择框的外观。

例6-3 定制选择框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. cb1.setStyle(
2. "-fx-border-color:lightblue; "
3. + "-fx-font-size: 20;"
4. + "-fx-border-insets: -5;"
5. + "-fx-border-radius:5;"
6. + "-fx-border-style:dotted;"
7. + "-fx-border-width:2;"
8. );

新的样式包含了点状的浅蓝色边框、增大的字体。图6-3展示了应用这种样式的选择框cb1。

http://hi.csdn.net/attachment/201111/3/0_1320292915w3tq.gif

图6-3 样式化的选择框

要为所有选择框设定特定的样式，请使用下列的过程：

* 创建一个.css文件
* 在.css文件中创建选择框的CSS类
* 在选择框的CSS类中定义需要的样式
* 在你的JavaFX程序中通过使用setStyleClass方法来应用样式单

## 相关API

* CheckBox
* JavaFX CSS Specification

# 7 组合框

组合框提供了在几个选项中快速选择的能力，考虑一下图7-1所示的组合框的实现。



例7-1 创建一个有三个选项的组合框

## 创建组合框

例7-1 创建一个有三个选项的组合框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
2. "First","Second", "Third")
3. );

例7-1展示了一个在ChoiceBox类内部创建项目列表的例子。列表项通过使用一个观察者数组来提供。当然，你可以使用类的空构造器来创建组合档，然后通过setItems方法指定项目列表。如例7-2所示。

例7-2有文本元素和分隔条的组合框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ChoiceBox cb = new ChoiceBox();
2. cb.setItems(FXCollections.observableArrayList(
3. "New Document","Open ",
4. new Separator(),"Save", "Save as")
5. );

注意：组合框不仅可包含文本，也可以包含其他对象。在例7-2中，使用一个分隔条来分开选项。当运行这个代码段时，产生的效果如图7-2所示。

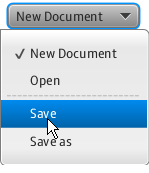


图7-2 通过组合框创建的菜单

在真实的应用中，组合框用来构建多个选项的列表。

## 设定组合框的行为

图7-3所示的应用程序提供了一个有5个选项的组合框，当一个选定特定的语言时，相应的欢迎语被绘制出来。

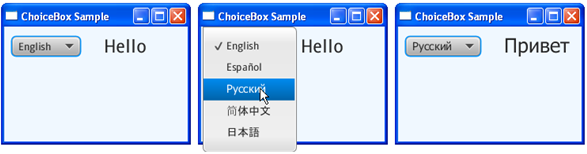
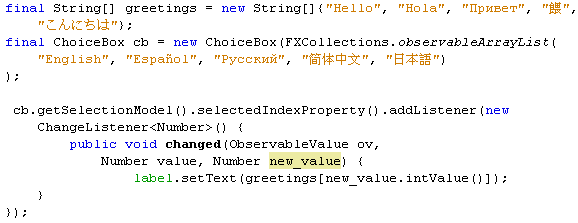


图7-3 多选项列表

例7-4提供了一个代码段来展现如何通过从组合框中选定项目来决定哪个欢迎语被绘制。

例7-4 选择一个组合框项目



ChangeListener<Number> 对象通过连续调用getSelectionModel和getSelectedIndex方法来检测当前选取的项目的序号。 getSelectionModel方法返回被选项目，getSelectedIndex方法返回组合框cb的SELECTED\_INDEX属性。结果 是，作为序号的数值定义了欢迎数组的一个元素，并且为标签指定了一个相应的文本。例如，如果用户选择第二个选项，它对应的是西班牙语，所选序号是1，从 greetings数组中选出第一个值“Hola”作为欢迎语。这样，标签绘制出“Hola”。

你可以通过使用提示来使组合框拥有更多的信息量。提示是javafx.scene.control包中提供的一个UI控件，它可以应用到任何JavaFX UI控件。

## 使用工具提示

Tooltip类提供了一个预定的提示语，它可以通过调用setTooltip方法来方便的为组合框（或其他任何控件）提供提示语。如例7-5所示。

例7-5 为组合框增加提示语

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. cb.setTooltip(new Tooltip("Select the language"));

用户一般通过Tooltip类的构造器来定义提示文本。然而，如果你的应用程序的逻辑需要为UI动态地设定提示文本，你可以先通过一个空构造器来创建工具提示对象，然后再通过它的setText方法来指定文字信息。

组合框cb应用了更改后的提示，当用户将鼠标放在组合框上时，会看到图7-5所示的内容。



图7-5 应用工具提示的组合框

为了进一步改善你的应用程序，你可以通过CSS属性或应用可视化特效来定制组合框。

## 相关API

* ChoiceBox
* Tooltip

# 8 文本框

TextBox类实现了一种接受并显示输入的UI控件，它提供了接受用户输入文本的功能。与之相应的还有另一种输入控件：PasswordBox，它继承自TextInputControl类。所有文本控件的超类都通过JavaFX API来提供。

图8-1展示了一个附标签的典型文本框

http://hi.csdn.net/attachment/201111/3/0_132029336655W5.gif

图8-1 标签和文本框

## 创建文本框

在例8-1中，一个文本框与一个标签组合起来，以便指出应该在文本框中填写什么内容。

例8-1 创建文本框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Label label1 = new Label("Name:");
2. TextBox textBox = new TextBox ();
3. HBox hb = new HBox();
4. hb.getChildren().addAll(label1, textBox);
5. hb.setSpacing(10);

你可以像例8-1一样的创建一个空的文本框，也可以创建一个有实际文字数据的文本框。要创建一个有特定内容的文本框，请使用TextBox类的这个构造器：TextBox(“Hello World”)。你可以在任何时侯通过调用getText方法来获取文本框的值。

你可以使用TextInputControl类的setColumns方法来设定文本框的尺寸，定义它能接受的最大字符数。

## 用文本框来构建UI

一般地，TextBox对象在表单中来使用，以创建几个文本框。图8-2中的程序显示了3个文本框，并且处理用户的输入。



图8-2 文本框的例子

例8-2中的代码段创建了三个文本框和两个按钮，然后通过使用GridPane容器将它们添加到程序的场景中。当你需要为你的UI实现灵活的布局时，使用容器可提供极大的便利。

例8-2 在程序中添加文本框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //Creating a GridPane container
2. GridPane grid = new GridPane();
3. grid.setPadding(new Insets(10, 10, 10, 10));
4. grid.setVgap(5);
5. grid.setHgap(5);
6. //Defining the Name text field
7. final TextBox name = new TextBox();
8. name.setPromptText("Enter your first name.");
9. name.setColumns(18);
10. GridPane.setConstraints(name, 0, 0);
11. grid.getChildren().add(name);
12. //Defining the Last Name text field
13. final TextBox lastName = new TextBox();
14. lastName.setPromptText("Enter your last name.");
15. GridPane.setConstraints(lastName, 0, 1);
16. grid.getChildren().add(lastName);
17. //Defining the Comment text field
18. final TextBox comment = new TextBox();
19. comment.setColumns(24);
20. comment.setPromptText("Enter your comment.");
21. GridPane.setConstraints(comment, 0, 2);
22. grid.getChildren().add(comment);
23. //Defining the Submit button
24. Button submit = new Button("Submit");
25. GridPane.setConstraints(submit, 1, 0);
26. grid.getChildren().add(submit);
27. //Defining the Clear button
28. Button clear = new Button("Clear");
29. GridPane.setConstraints(clear, 1, 1);
30. grid.getChildren().add(clear);

让我们花点时间来学习这个代码段。name、 lastName、comment文本框通过TextBox类的空构造器来创建。与例8-1不同，标签在代码段中没有附着于文本框，取而代之的是使用提示 文字来通知用户应该在文本框中输入何种数据。setPromptText方法定义了程序开始时出现在文本框中的文字。当例8-2被添加到程序中时，会产生 图8-3所示的输出。



图8-3 具有提示的三个文本框

提示文本和用户键入的内容有所不同，提示文本不能通过getText方法获得。

在真实的应用程序中，输入到文本框中的数据根据特定商业需求的程序逻辑来处理。下节将解释如何使用文本框来评估输入的文本，并向用户产生一个回应。

## 处理文本框数据

就像先前提到的一样，用户输入到文本框中的数据可以通过TextInputControl类的getText方法来获得。

研究8-3来学习如何处理文本框对象的数据。

例8-3 为提交和清除按钮定义动作

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //Adding a Label
2. final Label label = new Label();
3. GridPane.setConstraints(label, 0, 3);
4. GridPane.setColumnSpan(label, 2);
5. grid.getChildren().add(label);
7. //Setting an action for the Submit button
8. submit.setOnAction(new EventHandler<ActionEvent>() {
10. @Override
11. public void handle(ActionEvent e){
12. if ((comment.getText() !=null && !comment.getText().isEmpty())) {
13. label.setText(name.getText() + " " + lastName.getText() +", "
14. + "thank you foryour comment!");
15. } else {
16. label.setText("Youhave not left a comment.");
17. }
18. }
19. });
21. //Setting an action for the Clear button
22. clear.setOnAction(new EventHandler<ActionEvent>() {
24. @Override
25. public void handle(ActionEvent e){
26. name.clear();
27. lastName.clear();
28. comment.clear();
29. label.setText(null);
30. }
31. });

添加到GridPane中的标签控件向用户绘制程序的回应。当用户按下提交按钮时，setOnAction方法检查文本框的内容。如果是非容字符，一个感谢信息被绘制。否则，程序通知用户内容还设有确定，如图8-4所示。

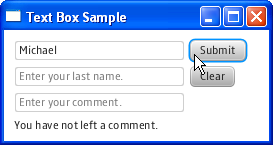


图8-4 文本框为空

当用户按下清除按钮时，清除方法被调用，删除所有三个文本框中的内容。

学习一下你可能在使用文本框时会用到的其他有用的方法。

* copy()–将当前选取的文本传递到剪贴板上，保留所选内容.
* cut()–将当前选取的文本传递到剪贴板上，清除所选内容.
* paste()–将剪贴板中的内容传递到文本框中，替换所选部份.

## 相关API

* TextBox
* TextInput

# 9 密码框

密码框实现了一个特别的文本输入控件，用户键入的字符被隐藏，取而代之的是显示一个回应字串。图9-1展示了一个有提示语的密码框。

http://hi.csdn.net/attachment/201111/3/0_1320293629TK5A.gif

图9-1 有提示的密码框

## 创建密码框

先用例9-1的代码来创建一个初级的密码框。

例9-1 创建密码框

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. PasswordBox passwordBox = new PasswordBox();
2. passwordBox.setColumns(12);
3. passwordBox.setPromptText("Your password");

在你的用户界面上，可 以为密码框指定一个伴随的提示语或增加一个通知标签。和TextBox类一样，PasswordBox提供了setText方法来在控件中绘制文本。然 而，用setText方法设定的字串在密码框中会被回应字符掩盖。默认的回应字符是“\*”号。图9-2展示了有预定义字符的密码框。

http://hi.csdn.net/attachment/201111/3/0_13202936788yBb.gif

图9-2 设定了文字的密码框

在密码框中键入的值可以通过getText方法获得，你可以在你的程序中处理这个值来设定适当的验证逻辑。

## 设定密码

花点时间看下例9-2的代码，它演示了在你的用户界面中实现的密码框。

例9-2 实现验证逻辑

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final Label message = new Label("");
3. VBox vb = new VBox();
4. vb.setPadding(new Insets(10, 0, 0, 10));
5. vb.setSpacing(10);
6. HBox hb = new HBox();
7. hb.setSpacing(10);
8. hb.setAlignment(Pos.CENTER\_LEFT);
10. Label label = new Label("Password");
11. final PasswordBox pb = new PasswordBox();
12. pb.setColumns(12);
13. pb.setAction(new Runnable() {
15. @Override
16. public void run() {
17. if(!pb.getText().equals("T2f$Ay!")) {
18. message.setText("Your password is incorrect!");
19. message.setTextFill(Color.rgb(210, 39, 30));
20. } else {
21. message.setText("Your password has been confirmed.");
22. message.setTextFill(Color.rgb(21, 117, 84));
23. }
24. pb.clear();
25. }
26. });
28. hb.getChildren().addAll(label, pb);
29. vb.getChildren().addAll(hb, message);

密码框的验证逻辑在setAction实例变量中被定义，变理代表一个函数，当键入的值提交后，它就会被调用。一般而言，当按下回车键时就会被调用。如果键入的值与预设的不一致，相应的红色提示信息会出现。如图9-3.

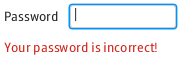
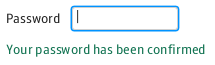


图9-3 密码不正确

如果键入的值满足预定标准，确认信息出现。如图9-4.



因为安全原因，当值被键入后，一般需要清除密码框中的数据。在例9-2中，认证完成后调用了clear方法来清除。

## 设定回应字符

你可以通过调用PasswordBox类的setEchoChar来更改默认回应符。另一种方法是在定义控件的时侯在PasswordBox类的构造器中指定默认的回应字符。例9-3表现了这两种意图。

例9-3 使用不同的回应符

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //Setting an echo character within a constructor
2. PasswordBox pb1 = new PasswordBox("#");
3. //Setting an echo character by using the setEchoChar method
4. PasswordBox pb2 = new PasswordBox();
5. pb2.setEchoChar("{1}quot;);

另外，例9-4中的setHideDelay方法提供了一个机会来设定一个毫秒级的时间延迟，它可以在键入字符时延迟一定的时间后再显示回应符（注：很Cool的改进，我第一次在我的Android智能手机中体验过）。这样，用户可以在继续前确认键入的内容。

例9-4 为回应符设定延迟

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. pb.setHideDelay(500);

例9-4为密码框设定一个500毫秒的延迟，当用户在里面键入时，他分辨率会体验到图图9-5所示的感觉。

http://hi.csdn.net/attachment/201111/3/0_13202938308FXb.gif

图9-5 设定延迟后键入密码

## 相关API

* PasswordBox
* TextInputControl

# 10 滚动条

ScrollBar类允许你在应用程序中使用滚动的面板和视图。图10-1展示了滚动条的三个区域：滑块、左右（上下）按钮和跟踪器。

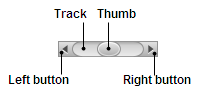


图10-1 滚动条元素

## 创建滚动条

让我们看一看例10-1的代码片段。

例10-1 简单的滚动条

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ScrollBar sc = new ScrollBar();
2. sc.setMin(0);
3. sc.setMax(100);
4. sc.setValue(50);

setMin和setMax方法定义滚动条所代表 的最小、最大值。当用户滑动滑块时，滚动条的值改变。在例10-1中，值是50。因此当程序开始时，滑块位于滚动条的中间。滚动条默认是水平放置的。然 而，你可以通过调用setOrientation方法将它设定为垂直放置。

用户可以单击左右按钮（在垂直模式是上下按钮）来滚动一个单位。UNIT\_INCREMENT属性定义当用户单击时滚动条滚动的量。另一个方法是单击跟踪区来滑动多个单位值，BLOCK\_INCREMENT属性定义了单击跟踪条时滚动条的滚动量。

在你的程序中，你可以使用多种方法来滚动浏览超出可用区域的图形元素。

## 在程序中使用滚动条

从动作来理解滚动条。例10-2的程序实现了一个滚动场景来浏览图像。这个程序的任务是让用户浏览垂面板的内容，它比场景的高度大。

例10-2 滚动浏览多个图像

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Orientation;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.ScrollBar;
8. import javafx.scene.effect.DropShadow;
9. import javafx.scene.image.Image;
10. import javafx.scene.image.ImageView;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.paint.Color;
13. import javafx.stage.Stage;
15. public class Main extends Application {
17. final ScrollBar sc = newScrollBar();
18. final Image[] images = newImage[5];
19. final ImageView[] pics = newImageView[5];
20. final VBox vb = new VBox();
21. DropShadow shadow = newDropShadow();
23. @Override
24. public void start(Stage stage) {
25. Group root = new Group();
26. Scene scene = new Scene(root, 180, 180);
27. scene.setFill(Color.BLACK);
28. stage.setScene(scene);
29. stage.setTitle("Scrollbar");
30. root.getChildren().addAll(vb,sc);
32. shadow.setColor(Color.GREY);
33. shadow.setOffsetX(2);
34. shadow.setOffsetY(2);
36. vb.setLayoutX(5);
37. vb.setSpacing(10);
39. sc.setLayoutX(scene.getWidth()-sc.getWidth());
40. sc.setMin(0);
41. sc.setOrientation(Orientation.VERTICAL);
42. sc.setPrefHeight(180);
43. sc.setMax(360);
45. for (int i = 0; i < 5;i++) {
46. final Image image =images[i] =
47. newImage(getClass().getResourceAsStream(
48. "fw"+(i+1)+ ".jpg")
49. );
50. final ImageView pic =pics[i] =
51. newImageView(images[i]);
52. pic.setEffect(shadow);
53. vb.getChildren().add(pics[i]);
54. }
56. sc.valueProperty().addListener(new ChangeListener<Number>() {
57. public voidchanged(ObservableValue<? extends Number> ov,
58. Number old\_val,Number new\_val) {
59. vb.setLayoutY(-new\_val.doubleValue());
60. }
61. });
63. stage.setVisible(true);
64. }
66. public static void main(String[]args) {
67. Application.launch(args);
68. }
69. }

程序的前几行向场景中添加了一个内含图像和滚动条的垂直面板。

当滚动条的Value属性改变时，垂直面板的Y坐标一起改变。因此当滑块移动、单击按钮或跟踪块时，垂直面板移动。如例10-3。

例10-3 垂直滚动条的实现

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. sc.valueProperty().addListener(new ChangeListener<Number>() {
2. public voidchanged(ObservableValue<? extends Number> ov,
3. Number old\_val, Numbernew\_val) {
4. vb.setLayoutY(-new\_val.doubleValue());
5. }
6. });

编译运行程序，输出如图10-2所示。



图10-2 滚动条例子

本程序演示了滚动条的典型应用，你也可以定制这个类在场景中创建滚动条。与其他任何UI控件和结点一样，滚动条可以通过样式来改变默认的实现风格。

## 样式化滚动条

考虑一下通过层叠样式表来样式化你的滚动条。例10-4演示了如何直接通过代码来样式化滚动条。

例10-4 将CSS样式应用到滚动条

 sc.setStyle(

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. "-fx-base:lemonchiffon;"
2. + "-fx-border-color:darkgreen; "
3. +"-fx-border-insets: -5; "
4. +"-fx-border-radius: 10;"
5. + "-fx-border-style:dotted;"
6. + "-fx-border-width:2;"
7. +"-fx-background-color: #b6e7c9;"
8. +"-fx-background-radius: 10;"
9. );

当把这些样式应用到例10-1中的代码中时，它的外观改变了。请参见图10-3来观察结果。

http://hi.csdn.net/attachment/201111/3/0_1320294093jm85.gif

图10-3 样式化的滚动条

你可以通过在程序中增加css文件，然后在程序中声明相应的类来样式化。

## 相关API

* ScrollBar
* JavaFX CSS规范

# 11 滚动面板

滚动面板为UI要素提供滚动视图。此控件使得用户可以通过平移视图或使用滚动条来滚动组件内容。默认设置的并附加了图像的滚动面板如图11-1所示。



图11-1 滚动面板

## 创建滚动面板

例11-1演示了如何创建滚动面板。

例11-1 使用滚动面板来浏览图像

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Image roses = newImage(getClass().getResourceAsStream("roses.jpg"));
2. ScrollPane sp = new ScrollPane();
3. sp.setNode(new ImageView(roses));

setNode方法定义了作为滚动面板内容使用有结点。你可以仅指定一个结点，要创建有多个组件的滚动视图，可以使用布局容器或Group类。你也可以为setPannable方法指定true值来通过单击并移动鼠标浏览图像，滚动条会作出相应的改变。

## 为滚动面板设定滚动条策略

ScrollPane类提供了一种策略来决定何时出现滚动条：总是可见、隐藏或需要时出现。使用setHbarPolicy和setVbarPolicy方法来分别决定垂直或水平滚动条的滚动条策略。因此，在例11-2中，会出现垂直滚动条而没有水平滚动条。

例11-2 设定水平及垂直滚动条策略

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. sp.setHbarPolicy(ScrollBarPolicy.NEVER);
2. sp.setVbarPolicy(ScrollBarPolicy.ALWAYS);

结果是，你只能垂直滚动图像，如图11-2所示。



图11-2 关闭水平滚动条

## 在滚动面板中重设组件尺寸

设计UI界面的时候，你可能需要去调整组件的尺寸，以更能匹配滚动条的高度和宽度，将setFitToWidth或setFitToHeight方法的参数设为true以匹配实际的尺寸。

图11-3所示的滚动面板包含了单选按钮、文本框和密码框。组件的尺寸超出了滚动面板的预定尺寸，因此滚动条会自动出现。然而，由于将setFitToWidth的参数设为了true，组件在水平位置上自动收缩，所以水平滚动条不会出现。



图11-3 匹配滚动面板的宽度

默认地，FIT\_TO\_WIDTH和FIT\_TO\_HEIGHT的属性均为false，这样可调整大小的组件保持原来的尺寸。如果你在此程序中将setFitToWidth方法删除，你将看到图11-4的结果。



图11-4 匹配组件的默认属性

这个滚动面板能让你在水平、垂直方向检索面板内容当前、最小、最大值，你可以在你的程序中学习使用它。

## 附滚动面板的示例程序

例11-3使用滚动面板来显示内附图像的垂直框，ScrollPane类的VALUE属性帮助你识别当前显示的图像，并绘制图像的文件名。

例11-3 使用滚动面板显示图像

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.scene.Scene;
5. import javafx.scene.control.Label;
6. import javafx.scene.control.ScrollPane;
7. import javafx.scene.image.Image;
8. import javafx.scene.image.ImageView;
9. import javafx.scene.layout.Priority;
10. import javafx.scene.layout.VBox;
11. import javafx.stage.Stage;
13. public class Main extends Application {
15. final ScrollPane sp = newScrollPane();
16. final Image[] images = newImage[5];
17. final ImageView[] pics = newImageView[5];
18. final VBox vb = new VBox();
19. final Label fileName = newLabel();
20. final String [] imageNames = newString [] {"fw1.jpg", "fw2.jpg",
21. "fw3.jpg","fw4.jpg", "fw5.jpg"};
23. @Override
24. public void start(Stage stage) {
25. VBox box = new VBox();
26. Scene scene = new Scene(box,180, 180);
27. stage.setScene(scene);
28. stage.setTitle("ScrollPane");
29. box.getChildren().addAll(sp,fileName);
30. box.setVgrow(sp,Priority.ALWAYS);
32. fileName.setLayoutX(30);
33. fileName.setLayoutY(160);
35. for (int i = 0; i < 5;i++) {
36. final Image image =images[i] =
37. new Image(getClass().getResourceAsStream(imageNames[i]));
38. final ImageView pic =pics[i] =
39. newImageView(images[i]);
40. pics[i].setFitWidth(100);
41. pics[i].setPreserveRatio(true);
42. vb.getChildren().add(pics[i]);
43. }
45. sp.setVmax(440);
46. sp.setPrefSize(115, 150);
47. sp.setNode(vb);
48. sp.vvalueProperty().addListener(new ChangeListener<Number>() {
49. public voidchanged(ObservableValue<? extends Number> ov,
50. Number old\_val,Number new\_val) {
51. fileName.setText(imageNames[(new\_val.intValue() - 1) / 100]);
52. }
53. });
54. stage.setVisible(true);
55. }
57. public static void main(String[]args) {
58. Application.launch(args);
59. }
60. }

编译和运行这段代码产生图11-5的输出。



图11-5 滚动图像

滚动条的最大值与垂直框的高度相等，如下例11-4的代码段绘制已显示图像的文件名称。

例11-4 跟踪滚动面板的垂直移动

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. sp.vvalueProperty().addListener(new ChangeListener<Number>() {
2. public voidchanged(ObservableValue<? extends Number> ov,
3. Number old\_val, Numbernew\_val) {
4. fileName.setText(imageNames[(new\_val.intValue() - 1) / 100]);
5. }
6. });

ImageView对象限制了图像的高度为100像素，这样，new\_val\_intValue-1除以100后，结果返回在imageNames数组中的当前图像。

在实际使用时，你还可以改变水平和垂直滚动条的最大最小值，这样可以动态更新你的用户界面。

## 相关API

* ScrollPane
* ScrollBar

# 12 列表视图

列表视图类代表一个可滚动项目的列表，图12-1展示了所有可选的客房类型。



图12-1 简单列表视图

  你可以通过使用setItems方法定义项目的方式来产生列表，你还可以应用setCellFactory方法来创建列表项的视图。

## 创建列表视图

例12-1的代码段实现了用字符列表项实现了图12-1的功能。

例12-1 创建列表视图

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ListView<String> list = new ListView<String>();
2. ObservableList<String> items =FXCollections.observableArrayList (
3. "Single","Double", "Suite", "Family App");
4. list.setItems(items);

要改变列表控件的尺寸，可使用setPrefHeight和setPrefWidth方法。例12-2将列表大小改为100×70，结果如图12-2所示。

例12-2 设置列表视图的高和宽

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. list.setPrefWidth(100);
2. list.setPrefHeight(70);



图12-2 调整垂直列表大小

你可以将列表视图的定位属性设为 Orientation.HORIZONTAL来将它的朝向更改为水平方向。可以这样 做：list.setOrientation(Orientation.HORIZONTAL)。包含和图12-1所示的相同列表项的水平列表如图 12-3所示。

http://hi.csdn.net/attachment/201111/3/0_13202946158QWi.gif

图12-3 水平列表控制

你可以通过以下方法的组合来获取列表视图中每个列表项的状态。

* getSelectionModel().selectedIndexProperty()—返回当前选中项的序号
* getSelectionModel().selectedItemProperty()—返回当前选中的列表项
* getFocusModel().getFocusIndex()—返加当前有焦点的列表项的序号
* getFocusModel().getFocusItem()—返加当前有焦点的列表项

需要注意的是，选取和聚焦状态是只读的，程序开始后你不能指定哪一个项目被选中和聚焦。

前述的列子代码展示了如何创建一个文字列表项，然而，列表视图可以包含任何结点对象。

## 产生带数据的列表视图

研究下列代码来学习如何使用单元格工厂方法产生列表项。例12-3的所示的程序创建一个颜色列表。

  例12-3 创建单元格工厂

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.collections.FXCollections;
5. import javafx.collections.ObservableList;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Label;
8. import javafx.scene.control.ListCell;
9. import javafx.scene.control.ListView;
10. import javafx.scene.layout.Priority;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.paint.Color;
13. import javafx.scene.shape.Rectangle;
14. import javafx.scene.text.Font;
15. import javafx.stage.Stage;
16. import javafx.util.Callback;
18. public class Main extends Application {
20. ListView<String> list = newListView<String>();
21. ObservableList<String> data= FXCollections.observableArrayList(
22. "chocolate","salmon", "gold", "coral",
23. "darkorchid","darkgoldenrod", "lightsalmon",
24. "black","rosybrown", "blue", "blueviolet",
25. "brown"
26. );
27. final Label label = new Label();
29. @Override
30. public void start(Stage stage) {
31. VBox box = new VBox();
32. Scene scene = new Scene(box,200, 200);
33. stage.setScene(scene);
34. stage.setTitle("ListViewSample");
35. box.getChildren().addAll(list, label);
36. box.setVgrow(list,Priority.ALWAYS);
38. label.setLayoutX(10);
39. label.setLayoutY(115);
40. label.setFont(newFont("Verdana", 20));
42. list.setItems(data);
44. list.setCellFactory(newCallback<ListView<String>,
45. ListCell<String>>() {
46. publicListCell<String> call(ListView<String> p) {
47. final Rectanglerect = new Rectangle(100, 20);
48. finalListCell<String> cell = new ListCell<String>() {
49. @Override publicvoid updateItem(String item,
50. booleanempty) {
51. super.updateItem(item, empty);
52. if (item!= null) {
53. rect.setFill(Color.web(item));
54. setNode(rect);
55. }
56. }
57. };
58. return cell;
59. }
60. });
61. list.setPrefHeight(100);
62. stage.setVisible(true);
63. }
65. public static void main(String[]args) {
66. Application.launch(args);
67. }
68. }

单 元格工厂产生ListCell对象，每全单元格与一个单独的数据项联系在一起，绘制列表视图的“一行”，通过setNode（译者注：在第42个构件版本 中，这个方法被抛弃了。取而代之的是setGraphics方法）方法设置的单元格的内容可以包含其他控件、文字、形状或图形。在此程序中，列表项表现了 矩形的情况。

编译运行这个程序，产生的输出如下：

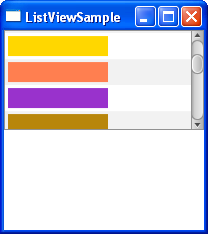


图12-4 颜色列表

你可以滚动列表，选择或取消列表中的任何项。你还可以扩充这个程序，使用颜色来填充文字标签。

## 处理列表项的选取

增加例12-4所示的代码段，可以实际项目被选取时产生产生事件。

例12-4 处理列表项的事件

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final Label label = new Label();
2. list.getSelectionModel().selectedItemProperty().addListener(
3. newChangeListener<String>() {
4. public void changed(ObservableValue<? extends String> ov,
5. String old\_val, Stringnew\_val) {
6. label.setText(new\_val);
7. label.setTextFill(Color.web(new\_val));
8. }
9. });

addListener 方法调用selectedItemProperty创建一个新的ChangeListener<String>对象来处理选择项的改变。例 如，如果深蓝色被选取，标签接收“darkorchid”，并使用相应的颜色来绘制。修改后的程序输出如图12-5所示。

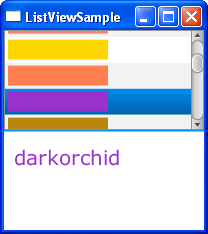


图12-5 选择深紫色的情况

## 相关API

* ListView
* ListCellFactory

# 13 表格视图

JavaFX  SDK中设计了几个用于表现表格化数据的类，在JavaFX中用来创建表格的最重要的类是TableView、TableColumn和TableCell类。你可以通过实现数据模型或应用单元格生成器来创建表格。

表格类提供了内建的功能在需要时按列来排序或调整列的宽度。

图13-1展示了一个地址簿中的联系人的典型表格。



图13-1 表格例子

## 创建表格

例13-1所示的代码段展示了创建三个列的表格，并将它增加到场景中的情况。

例13-1 增加一个表格

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.geometry.Insets;
3. import javafx.scene.Group;
4. import javafx.scene.Scene;
5. import javafx.scene.control.Label;
6. import javafx.scene.control.TableColumn;
7. import javafx.scene.control.TableView;
8. import javafx.scene.layout.VBox;
9. import javafx.scene.text.Font;
10. import javafx.stage.Stage;
11. public class Main extendsApplication {
12. private TableView table = newTableView();
13. public static void main(String[]args) {
14. launch(args);
15. }
17. @Override
18. public void start(Stage stage) {
19. Scene scene = new Scene(newGroup());
20. stage.setTitle("TableView Sample");
21. stage.setWidth(400);
22. stage.setHeight(450);
24. final Label label = newLabel("Address Book");
25. label.setFont(new Font("Arial",20));
27. table.setStyle("-fx-base: #b6e7c9;");
29. TableColumn firstNameCol =new TableColumn("First Name");
30. TableColumn lastNameCol = newTableColumn("Last Name");
31. TableColumn emailCol = newTableColumn("Email");
33. table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);
35. final VBox vbox = new VBox();
36. vbox.setSpacing(5);
37. vbox.getChildren().addAll(label, table);
38. vbox.setPadding(newInsets(10, 0, 0, 10));
40. ((Group)scene.getRoot()).getChildren().addAll(vbox);
42. stage.setScene(scene);
43. stage.setVisible(true);
44. }
45. }

表格控件通过实例化TableView类来创建。在例13-1中，表格由VBox容器来管理，然而，你可以将表格直接加入到应用程序的场景中。

例13-1定义了三个列来存储地址簿中联系人的姓、名和邮件地址信息，这些列通过TableColumn类来创建。

TableView类的getColumns()方法将前面创建的列增加到表格中。在你的程序中，你可以通过该方法来动态地增加或删除表格的列。

这个代码段同时使用了setStyle()方法来改变默认的表格外观。

编译并运行此代码，将产生如图13-2的输出。

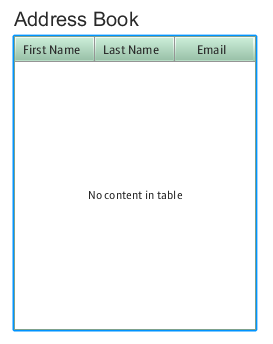


图13-2 没有数据的表格

你可以通过调查setVisible方法来管理列是否可见。例如，如果你的应用程序的业务逻辑需要隐藏用户的邮件列，可以这样来实现：emailCol.setVisible(false)。

如果你的数据结构需要进行更复杂的表现方式，你可以创建一个嵌套列。

例如，假设地址簿中的联系人有两个email地址。那么就需要两列来展现主要地址和次要地址。你可以通过创建两个子列，再像例13-2所示的在emailCol列上调用getColumns方法来增加。

例13-2 创建嵌套列

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. TableColumn firstEmailCol = new TableColumn("Primary");
2. TableColumn secondEmailCol = new TableColumn("Secondary");
4. emailCol.getColumns().addAll(firstEmailCol, secondEmailCol);

  一旦你将这些代码加到例13-1中后，然后编译并运行程序代码，就会产生图13-3的样式。

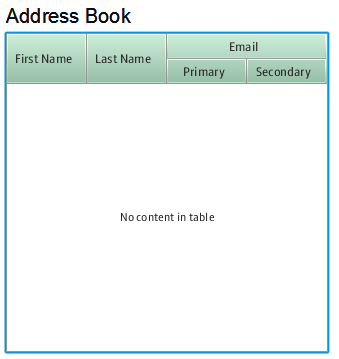


图13-3 有嵌套列的表格

虽然表格加入到了应用程序中，但由于没有数据，所以会出现一个标准表格标题“表格无数据”。如果要要改变这个标题，你可以调用setPlaceholder方法在空表中指定一个Node对象作为占位符。

## 定义表格模型

在程序中创建表格时，最好的方法是实现一个定义了数据模型，并提供了方法和属性来操作表格的类。例13-3创建了Person类来定义地址簿中的数据。

例13-3 创建Person类

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. private final StringProperty firstName;
2. private final StringPropertylastName;
3. private final StringPropertyemail;
4. private Person(String fName,String lName, String email) {
5. this.firstName = newStringProperty(fName);
6. this.lastName = newStringProperty(lName);
7. this.email = newStringProperty(email);
8. }
9. public String getFirstName() {
10. return firstName.get();
11. }
12. public void setFirstName(StringfName) {
13. firstName.set(fName);
14. }
15. public String getLastName() {
16. return lastName.get();
17. }
18. public void setLastName(StringfName) {
19. lastName.set(fName);
20. }
21. public String getEmail() {
22. return email.get();
23. }
24. public void setEmail(StringfName) {
25. email.set(fName);
26. }
27. }

firstName、lastName、和email被创建来引用实际的数据元素。另外，还为每个数据元素提供了get和set方法。因此可以用getFirstName方法来获得firstName属性的值，setFirstName方法来指定这个属性的值。

当数据模型在Person类中被描述后，你可以创建一个可观察列表（ObservableList）来定义你要在表中展示的任意多个数据行。例13-4的代码段实现了这个任务。

例13-4 在ObservableList中定表表格数据

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final ObservableList<Person> data = FXCollections.observableArrayList(
2. new Person("Jacob","Smith", "jacob.smith@example.com"),
3. new Person("Isabella","Johnson", "isabella.johnson@example.com"),
4. new Person("Ethan","Williams", "ethan.williams@example.com"),
5. new Person("Emma","Jones", "emma.jones@example.com"),
6. new Person("Michael","Brown", "michael.brown@example.com")
7. );

下一步是将数据与表格的列关联起来。你可以通过为每个数据元素定义的属性来完成。如例13-5。

例13-5 设置列的数据属性

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. firstNameCol.setProperty("firstName");
2. lastNameCol.setProperty("lastName");
3. emailCol.setProperty("email");

定义了数据模型后，数据被填入，并与列形成了关联，你可以用TableView的setItems方法向表中增加数据。如：setItem(data)。

由于ObservableList对象可以跟踪列表内元素的变化，因此当数据改变时TableView会自动更新其内容。

仔细阅读一下例13-6的代码。

例13-6 创建表格并为表格添加数据

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.property.StringProperty;
3. import javafx.collections.FXCollections;
4. import javafx.collections.ObservableList;
5. import javafx.geometry.Insets;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.control.Label;
9. import javafx.scene.control.TableColumn;
10. import javafx.scene.control.TableView;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.text.Font;
13. import javafx.stage.Stage;
15. public class Main extends Application {
16. public static class Person {
17. private final StringPropertyfirstName;
18. private final StringPropertylastName;
19. private final StringPropertyemail;
21. private Person(String fName,String lName, String email) {
22. this.firstName = newStringProperty(fName);
23. this.lastName = new StringProperty(lName);
24. this.email = newStringProperty(email);
25. }
27. public String getFirstName(){
28. return firstName.get();
29. }
31. public voidsetFirstName(String fName) {
32. firstName.set(fName);
33. }
35. public String getLastName() {
36. return lastName.get();
37. }
39. public voidsetLastName(String fName) {
40. lastName.set(fName);
41. }
43. public String getEmail() {
44. return email.get();
45. }
47. public void setEmail(StringfName) {
48. email.set(fName);
49. }
51. }
52. private TableView<Person>table = new TableView<Person>();
53. private finalObservableList<Person> data =
54. FXCollections.observableArrayList(
55. newPerson("Jacob", "Smith","jacob.smith@example.com"),
56. newPerson("Isabella", "Johnson","isabella.johnson@example.com"),
57. new Person("Ethan","Williams", "ethan.williams@example.com"),
58. newPerson("Emma", "Jones","emma.jones@example.com"),
59. newPerson("Michael", "Brown","michael.brown@example.com")
60. );
62. public static void main(String[]args) {
63. launch(args);
64. }
66. @Override
67. public void start(Stage stage) {
68. Scene scene = new Scene(newGroup());
69. stage.setTitle("TableView Sample");
70. stage.setWidth(400);
71. stage.setHeight(450);
73. final Label label = newLabel("Address Book");
74. label.setFont(newFont("Arial", 20));
76. table.setStyle("-fx-base: #b6e7c9;");
78. TableColumn firstNameCol =new TableColumn("First Name");
79. firstNameCol.setProperty("firstName");
81. TableColumn lastNameCol = newTableColumn("Last Name");
82. lastNameCol.setProperty("lastName");
84. TableColumn emailCol = newTableColumn("Email");
85. emailCol.setMinWidth(200);
86. emailCol.setProperty("email");
88. table.setItems(data);
89. table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

92. final VBox vbox = new VBox();
93. vbox.setSpacing(5);
94. vbox.getChildren().addAll(label, table);
95. vbox.setPadding(newInsets(10, 0, 0, 10));
97. ((Group)scene.getRoot()).getChildren().addAll(vbox);
99. stage.setScene(scene);
100. stage.setVisible(true);
101. }
102. }

当你编译并运行这个程序后，产生的输出如图13-4所示：

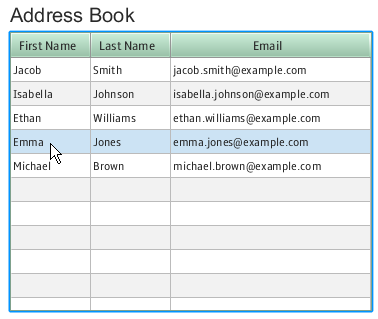


图13-4 装填了数据的表格

## 增加新行

图13-4的表格包含了5行数据，然而，目前为止你还不能修改数据。

你可以使用文本框来键入姓、名、邮件地址的新值。文本框控件可以让你的程序具有接受用户输入的能力。例13-7创建了三个文本框，并为每个文本框定义了掩码，然后创建了一个按钮。

例13-7 使用文本框在表中键入新值

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final TextBox addFirstName = new TextBox();
2. addFirstName.setPromptText("First Name");
3. addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
4. final TextBox addLastName = new TextBox();
5. addLastName.setMaxWidth(lastNameCol.getPrefWidth());
6. addLastName.setPromptText("Last Name");
7. final TextBox addEmail = new TextBox();
8. addEmail.setMinWidth(emailCol.getPrefWidth());
9. addEmail.setPromptText("Email");
11. final Button addButton = new Button("Add");
12. addButton.setOnAction(new EventHandler<ActionEvent>() {
13. @Override public voidhandle(ActionEvent e) {
14. data.add(new Person(
15. addFirstName.getText(),
16. addLastName.getText(),
17. addEmail.getText())
18. );
19. addFirstName.clear();
20. addLastName.clear();
21. addEmail.clear();
22. }
23. });

当用户单击增加按钮时，键入文本框中的值被包含在Person类的构造器中增加到了可观察列表中，这样，新键入的联系人信息就会自动出现在表格中。

请仔细看一下例13-8的代码。

例13-8 包含可键入新联系人的文本框的表格

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.property.StringProperty;
3. import javafx.collections.FXCollections;
4. import javafx.collections.ObservableList;
5. import javafx.event.ActionEvent;
6. import javafx.event.EventHandler;
7. import javafx.geometry.Insets;
8. import javafx.scene.Group;
9. import javafx.scene.Scene;
10. import javafx.scene.control.Button;
11. import javafx.scene.control.Label;
12. import javafx.scene.control.TableColumn;
13. import javafx.scene.control.TableView;
14. import javafx.scene.control.TextBox;
15. import javafx.scene.layout.HBox;
16. import javafx.scene.layout.VBox;
17. import javafx.scene.text.Font;
18. import javafx.stage.Stage;
20. public class Main extends Application {
22. public static class Person {
24. private final StringPropertyfirstName;
25. private final StringPropertylastName;
26. private final StringPropertyemail;
28. private Person(String fName,String lName, String email) {
29. this.firstName = newStringProperty(fName);
30. this.lastName = newStringProperty(lName);
31. this.email = newStringProperty(email);
32. }
34. public String getFirstName(){
35. return firstName.get();
36. }
38. public voidsetFirstName(String fName) {
39. firstName.set(fName);
40. }
42. public String getLastName() {
43. return lastName.get();
44. }
46. public voidsetLastName(String fName) {
47. lastName.set(fName);
48. }
50. public String getEmail() {
51. return email.get();
52. }
54. public void setEmail(StringfName) {
55. email.set(fName);
56. }
58. }
59. private TableView<Person>table = new TableView<Person>();
60. private finalObservableList<Person> data =
61. FXCollections.observableArrayList(
62. newPerson("Jacob", "Smith","jacob.smith@example.com"),
63. newPerson("Isabella", "Johnson","isabella.johnson@example.com"),
64. newPerson("Ethan", "Williams","ethan.williams@example.com"),
65. new Person("Emma","Jones", "emma.jones@example.com"),
66. newPerson("Michael", "Brown","michael.brown@example.com")
67. );
69. private HBox hb = new HBox();
71. public static void main(String[]args) {
72. launch(args);
73. }
75. @Override
76. public void start(Stage stage) {
77. Scene scene = new Scene(newGroup());
78. stage.setTitle("TableView Sample");
79. stage.setWidth(400);
80. stage.setHeight(450);
82. final Label label = newLabel("Address Book");
83. label.setFont(newFont("Arial", 20));
85. table.setStyle("-fx-base: #b6e7c9;");
87. TableColumn firstNameCol =new TableColumn("First");
88. firstNameCol.setProperty("firstName");
90. TableColumn lastNameCol = newTableColumn("Last");
91. lastNameCol.setProperty("lastName");
93. TableColumn emailCol = newTableColumn("Email");
94. emailCol.setMinWidth(200);
95. emailCol.setProperty("email");
97. table.setItems(data);
98. table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);
100. final TextBox addFirstName =new TextBox();
101. addFirstName.setPromptText("First Name");
102. addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
103. final TextBox addLastName =new TextBox();
104. addLastName.setMaxWidth(lastNameCol.getPrefWidth());
105. addLastName.setPromptText("Last Name");
106. final TextBox addEmail = newTextBox();
107. addEmail.setMinWidth(emailCol.getPrefWidth());
108. addEmail.setPromptText("Email");
110. final Button addButton = newButton("Add");
111. addButton.setOnAction(newEventHandler<ActionEvent>() {
112. @Override public voidhandle(ActionEvent e) {
113. data.add(new Person(
114. addFirstName.getText(),
115. addLastName.getText(),
116. addEmail.getText())
117. );
118. addFirstName.clear();
119. addLastName.clear();
120. addEmail.clear();
121. }
122. });
124. hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton);
125. hb.setSpacing(3);
127. final VBox vbox = new VBox();
128. vbox.setSpacing(5);
129. vbox.getChildren().addAll(label,table, hb);
130. vbox.setPadding(newInsets(10, 0, 0, 10));
132. ((Group)scene.getRoot()).getChildren().addAll(vbox);
134. stage.setScene(scene);
135. stage.setVisible(true);
136. }
137. }

Clear方法用于删除文本框中的数据以便增加更多的数据。

这个程序没有提供任何过虑器来检查数据，例如检查邮件格式是否合法。开发你自已的程序时，可以提供这类功能。

现在这个版本也不检查是否有空的输入，如果没有提供值，单击按钮会插入一个空行。

图13-5演示了用户如何增加一行数据。

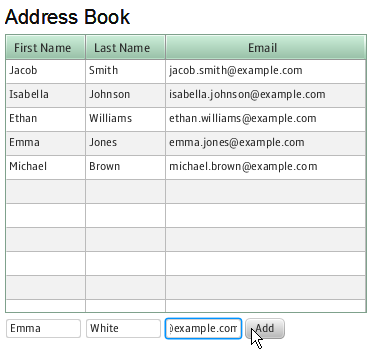


图13-5 向地址簿中增加联系人信息

图13-6展示了单击增加按钮后的情况，Emma White的信息已经出现在表中了。

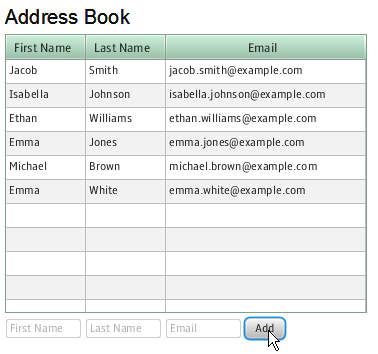


图13-6 新增的条目

## 对列进行排序

TableView类提供了内建的列排序功能，用户可以通过单击列标题来改变数据的顺序。第一次单击按升序排列，再一单击按降序排列，第三次单击取消排序。默认不进行排序。

用户可以对表进行多列排序，并可指定在排序时哪一列是排序的主要列。要对多列进行排序，用户可以在要排序的列标题是单击时按下SHIFT键。

在图13-7中，已对“姓”这一列按升序进行了排列，可以再对“名”这一列按降序进行排列。注意第一列比第二列具有更高的优先级，是主排序列。

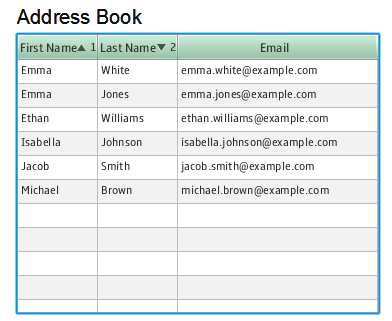


图13-7 多列排序

如果是开发人员，你可以在程序中通过setSortAscending方法来设置各列的排序选项，为此方法传入“true”的参数为升序排列，如果参数为“false”则为降序。

你也可以通过从TableView.sortOrder观察者列表中增加或删除TableColoumn实例来指定对哪一列进行排序，列表中列的顺序代表排序的优先顺序（例如：第一个优先于第二个）。

如果要禁止排序，可对列调用setSortable(false)方法。

你还可以使用JavaFX SDK提供的FilteredList类为你的表格实现一个过滤器。这个类包装了ObservableList类，提供了方便的过滤功能。创建过滤器时， 你要指定数据源和过滤准则（适配器）。它支持LIVE和BATCH过滤模式。FilterableList.FiterMode.LIVE模式为自动过 滤。而FilterableList.FiterMode.BATCH模式通过FilterableList.fliter方法的要求来过滤。

SortedList类是ObservableList类的另一种包装器，当新数据加入到表中时，你可以使用它来排序。

## 在表中编辑数据

TableView类不仅可以绘制表格化的数据， 它还提供了编辑的能力。你可以使用TableView.edit(TablePosition)方法来启动编辑器。要取消编辑，可以为编辑器传一个 null值。你还可以通过TableCell类来编辑表中的数据，如例13-9所示：

例13-9 实现单元格编辑

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. class EditingCell extends TableCell<String> {
2. private final Label label;
3. private TextBox textBox;
5. public EditingCell() {
6. this.label = new Label();
7. }
9. @Override
10. public void startEdit() {
11. super.startEdit();
12. if (isEmpty()) {
13. return;
14. }
16. if (textBox == null) {
17. createTextBox();
18. } else {
19. textBox.setText(getItem());
20. }
21. setNode(textBox);
22. textBox.requestFocus();
23. textBox.selectAll();
24. }
26. @Override
27. public void cancelEdit() {
28. super.cancelEdit();
29. setNode(label);
30. }
32. @Override
33. public void commitEdit(Stringt) {
34. super.commitEdit(t);
35. setNode(label);
36. }
38. @Override
39. public void updateItem(Stringitem, boolean empty) {
40. super.updateItem(item,empty);
41. if (!isEmpty()) {
42. if (textBox != null){
43. textBox.setText(item);
44. }
45. label.setText(item);
46. setNode(label);
47. }
48. }
50. private void createTextBox(){
51. textBox = newTextBox(getItem());
52. textBox.setOnKeyReleased(new EventHandler<KeyEvent>() {
53. @Override public void handle(KeyEventt) {
54. if (t.getCode()== KeyCode.ENTER) {
55. commitEdit(textBox.getRawText());
56. } else if(t.getCode() == KeyCode.ESCAPE) {
57. cancelEdit();
58. }
59. }
60. });
61. }
62. }

在例13-9中，createTextBox方法使用一个私有的textbox变量来分析键入的值序列，然后在输入ENTER或ESCAPE时分别调用commitEdit或cancelEdit方法。

TableColoumn类的 setCellFactory方法用以为单元格安装一个自定义的单元格构造器，这个构造器的作用是在需要时返加新的TableCell实例。例13-10 展示了如何为firstName、lastName和emailCol列实现一个单元格构造器。

例13-10 使用单元格构造器

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Callback<TableColumn, TableCell> cellFactory =
2. new Callback<TableColumn,TableCell>() {
3. public TableCellcall(TableColumn p) {
4. return new EditingCell();
5. }
6. };

如例13-11一样，使用TableColumn类的setOnEditCommit方法来处理单元格内容的变化。这个方法识别修改内容，检索新的值，然后替换可观察列表中当前元素的数据。

例13-11 处理表中的数据编辑

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //Modifying the firstName property
2. firstNameCol.setOnEditCommit(newEventHandler<EditEvent<String>>() {
3. @Override public voidhandle(EditEvent<String> t) {
4. ((Person)t.getTableView().getItems().get(
5. t.getTablePosition().getRow())).setFirstName(t.getNewValue());
6. }
7. });

10. //Modifying the lastName property
11. lastNameCol.setOnEditCommit(newEventHandler<EditEvent<String>>() {
12. @Override public voidhandle(EditEvent<String> t) {
13. ((Person)t.getTableView().getItems().get(
14. t.getTablePosition().getRow())).setLastName(t.getNewValue());
15. }
16. });
18. //Modifying the email property
19. emailCol.setOnEditCommit(new EventHandler<EditEvent<String>>(){
20. @Override public voidhandle(EditEvent<String> t) {
21. ((Person)t.getTableView().getItems().get(
22. t.getTablePosition().getRow())).setEmail(t.getNewValue());
23. }
24. });

在图13-8中，用户正编辑MichaelBrown的名字。要编辑单元格，用户在单元格中键入新的值，然后按下Enter键。如果不按下Enter键，单元格的值不会被修改。这种行为主要是因为单元格编辑器实现了TextBox类。

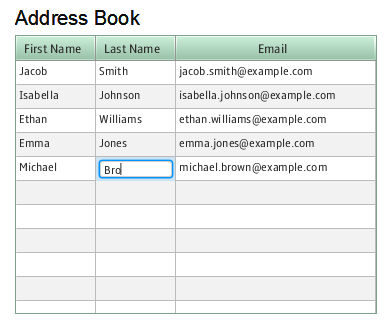


图13-8 单元格编辑

## 相关API

* TableView
* TableColumn
* TableCell
* TextBox
* Button

# 14 分隔条

JavaFX API提供的Separator类代表一个水平或垂直的线条。它用来在用户界面中分隔组件，它不会产生任何动作。不过，你可以对它进行样式化，增加特效或创建动画。默认地，分隔条是水平的，但可以用setOrientation方法来改变方向。

## 创建分隔条

例14-1的代码段创建了一个水平分隔条和一个垂直分隔条。

例14-1 水平和垂直分隔条

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //Horizontal separator
2. Separator separator1 = new Separator();
3. //Vertical separator
4. Separator separator2 = new Separator();
5. separator2.setOrientation(Orientation.VERTICAL);

Separator类是Node类的子类，因此，它继承了Node类的所有实例变更。

一般而言，分隔条用来分隔UI组件的组，研究一下例14-2的代码段，它将春季复选框和夏季复选框分开。

例14-2 在复选类型间使用分隔条

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final String[] names = new String[]{"March", "April","May",
2. "June","July", "August"};
3. final CheckBox[] cbs = new CheckBox[names.length];
4. final Separator separator = new Separator();
5. final VBox vbox = new VBox();
7. for (int i = 0; i < names.length; i++) {
8. cbs[i] = new CheckBox(names[i]);
9. }
11. separator.setMaxWidth(40);
12. separator.setAlignment(Pos.CENTER\_LEFT);
13. vbox.getChildren().addAll(cbs);
14. vbox.setSpacing(5);
15. vbox.getChildren().add(3, separator);

这段代码增加到程序中后，产生图14-1的控件样式。

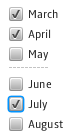


图14-1 复选框和分隔条

分隔条占用了分配给它的水平或垂直位置。setMaxWidth方法用于定义实际的宽度，setVPos方法在已分配的布局空间内指定分隔条的垂直位置。相应的，你可以通过setHpos方法设置水平分隔线的位置。

在例14-2中，分隔条通过党规的add(index,node)方法添加到垂直框中。你可以在你的应用程序中使用这种方法在UI创建后或动态改变后来增加分隔条。

## 在程序中为UI添加分隔条

如前所述，分隔条于来分隔UI控件组，你也可以使用它来构造用户界面。考虑一下创建一个如图14-2所示的绘制一个天气预报信息的情况。

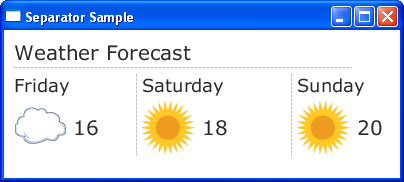


图14-2 用分隔条构建天气预报数据

对图14-2所示的程序来说，分隔条用于分隔Label和ImageView对象，请研究一下例14-3所示的程序的代码。

例14-3 在天气预报程序中使用分隔条

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.geometry.Insets;
3. import javafx.geometry.VPos;
4. import javafx.scene.Group;
5. import javafx.scene.Scene;
6. import javafx.scene.control.\*;
7. import javafx.scene.effect.DropShadow;
8. import javafx.scene.image.Image;
9. import javafx.scene.image.ImageView;
10. import javafx.scene.layout.GridPane;
11. import javafx.scene.text.Font;
12. import javafx.stage.Stage;
14. public class Main extends Application {
16. Button button3 = newButton("Decline");
17. DropShadow shadow = newDropShadow();
18. Label caption = newLabel("Weather Forecast");
19. Label friday = newLabel("Friday");
20. Label saturday = newLabel("Saturday");
21. Label sunday = newLabel("Sunday");
23. @Override
24. public void start(Stage stage) {
25. Group root = new Group();
26. Scene scene = new Scene(root,500, 300);
27. stage.setScene(scene);
28. stage.setTitle("Separator Sample");
30. GridPane grid = newGridPane();
31. grid.setPadding(newInsets(10, 10, 10, 10));
32. grid.setVgap(2);
33. grid.setHgap(5);
35. scene.setRoot(grid);
37. Image cloudImage = newImage(getClass().getResourceAsStream("cloud.jpg"));
38. Image sunImage = newImage(getClass().getResourceAsStream("sun.jpg"));
40. caption.setFont(Font.font("Verdana", 20));
41. GridPane.setConstraints(caption, 0, 0);
42. grid.setColumnSpan(caption,8);
43. grid.getChildren().add(caption);
45. final Separator sepHor = newSeparator();
46. sepHor.setVpos(VPos.CENTER);
47. GridPane.setConstraints(sepHor, 0, 1);
48. grid.setColumnSpan(sepHor,7);
49. grid.getChildren().add(sepHor);
51. friday.setFont(Font.font("Verdana", 18));
52. GridPane.setConstraints(friday, 0, 2);
53. grid.setColumnSpan(friday,2);
54. grid.getChildren().add(friday);
56. final Separator sepVert1 =new Separator();
57. sepVert1.setOrientation(Orientation.VERTICAL);
58. sepVert1.setVpos(VPos.CENTER);
59. sepVert1.setPrefHeight(80);
60. GridPane.setConstraints(sepVert1, 2, 2);
61. grid.setRowSpan(sepVert1, 2);
62. grid.getChildren().add(sepVert1);
64. saturday.setFont(Font.font("Verdana",18));
65. GridPane.setConstraints(saturday, 3, 2);
66. grid.setColumnSpan(saturday,2);
67. grid.getChildren().add(saturday);
69. final Separator sepVert2 =new Separator();
70. sepVert2.setOrientation(Orientation.VERTICAL);
71. sepVert2.setVpos(VPos.CENTER);
72. sepVert2.setPrefHeight(80);
73. GridPane.setConstraints(sepVert2, 5, 2);
74. grid.setRowSpan(sepVert2, 2);
75. grid.getChildren().add(sepVert2);
77. sunday.setFont(Font.font("Verdana", 18));
78. GridPane.setConstraints(sunday, 6, 2);
79. grid.setColumnSpan(sunday,2);
80. grid.getChildren().add(sunday);
82. final ImageView cloud = newImageView(cloudImage);
83. GridPane.setConstraints(cloud,0, 3);
84. grid.getChildren().add(cloud);
86. final Label t1 = newLabel("16");
87. t1.setFont(Font.font("Verdana", 20));
88. GridPane.setConstraints(t1,1, 3);
89. grid.getChildren().add(t1);
91. final ImageView sun1 = newImageView(sunImage);
92. GridPane.setConstraints(sun1,3, 3);
93. grid.getChildren().add(sun1);
95. final Label t2 = newLabel("18");
96. t2.setFont(Font.font("Verdana", 20));
97. GridPane.setConstraints(t2,4, 3);
98. grid.getChildren().add(t2);
100. final ImageView sun2 = newImageView(sunImage);
101. GridPane.setConstraints(sun2,6, 3);
102. grid.getChildren().add(sun2);
104. final Label t3 = new Label("20");
105. t3.setFont(Font.font("Verdana", 20));
106. GridPane.setConstraints(t3,7, 3);
107. grid.getChildren().add(t3);
109. stage.setVisible(true);
110. }
111. public static void main(String[]args) {
112. Application.launch(args);
113. }
114. }

这个程序同时使用了水平和垂直分隔条，并且它们跨 越了GridPane容器的行和列，在你的程序中，也可以为分隔条设定推荐的长度（对水平分隔条是长度，对垂直分隔条是高度），以便当用户界面的大小被调 整时它能动态地改变。你还可以通过Separator对象提供的CSS类来为分隔条增加额外的可视化效果。

## 样式化分隔条

要为例14-3中的所有分隔条使用相同的样式，你可以创建一个CSS文件（例如：controlStyle.css），然后将它保存在与主类（main class）相同的包内。例14-4演示了可以增加到controlSytle文件中的CSS类。

例14-4 使用CSS类样式化分隔条

**[css]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. /\*controlStyle.css \*/
2. .separator{
3. -fx-background-color: #e79423;
4. -fx-background-radius: 2;
5. -fx-background-insets: -2;
6. }

你可以在程序中通过getStylesheets方法来启用样式化的分隔条。如例14-5所示：

例14-5 在JavaFX程序中启用样式表

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. scene.getStylesheets().add("/separatorsample/controlStyle.css");

图14-3 展示了程序修改后分隔条在天气预报程序中的样子。

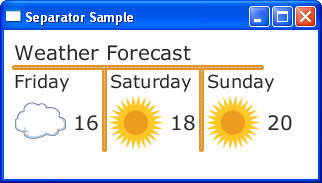


图14-3 样式化的分隔条

## 相关API

* Separator
* JavaFX CSS Specification

# 15 滑动条

滑动条代表一种在表现和交互一定范围内的数值的控件，它包括一个跟踪条和一个滑动块。也可以包含表示数值范围的刻度和刻度标签。图15-1展示了典型的滑动条和它的主要构件。

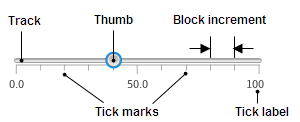


图15-1 滑动条的构件

## 创建滑动条

让我们花点时间来研究一下例15-1的代码段，它产生的输出如图15-1所示。

例15-1 创建滑动条

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Slider slider = new Slider();
2. slider.setMin(0);
3. slider.setMax(100);
4. slider.setValue(40);
5. slider.setShowTickLabels(true);
6. slider.setShowTickMarks(true);
7. slider.setMajorTickUnit(50);
8. slider.setMinorTickCount(5);
9. slider.setBlockIncrement(10);

setMin和setMax方法分别定义了滑动条代表的最大和最小数值。setValue方法指定了滑动条的当前值，这个值永远比最大值小，比最小值大。当程序运行时，使用这个方法来指定滑动块的位置。

两个布尔方法，setShowTickMarks 和setShowTickLabels定义了滑动块的外观。在例15-1中，刻度和标签已经启用。另外，主要刻度的间距设为了50，两个主要刻度间的次要 刻度的间距设为了5。你可以将setSnapToTicks方法的参数设为true来保证滑动块的值正好位于刻度上。

setBlockIncrement方法定义了用户在跟踪条上单击时滑动块移动的距离。在例15-1中，这个值是10，滑动块会向单击方向移动10个单位。

## 在图形程序中使用滑动条

现在查看一下图15-2，这个程序使用了三个滑动条来编辑图片的绘制属性。每个滑动条调整一个实际的可视化属性：透明度，棕褐色调值和比例因子。

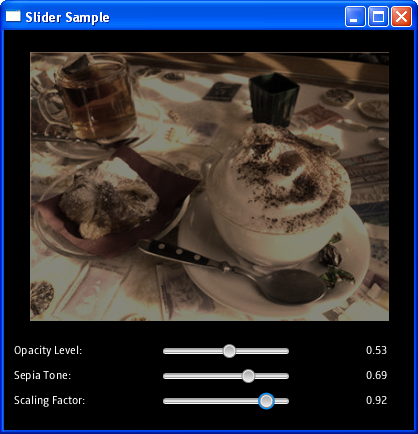


图15-2 三个滑动条

例15-2展示了这个程序的源代码

例15-2 滑动条例子

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Insets;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Label;
8. import javafx.scene.control.Slider;
9. import javafx.scene.effect.SepiaTone;
10. import javafx.scene.image.Image;
11. import javafx.scene.image.ImageView;
12. import javafx.scene.layout.GridPane;
13. import javafx.scene.paint.Color;
14. import javafx.stage.Stage;
16. public class Main extends Application {
18. final Slider opacityLevel = newSlider(0, 1, 1);
19. final Slider sepiaTone = newSlider(0, 1, 1);
20. final Slider scaling = new Slider(0.5, 1, 1);
21. final Image image  = new Image(getClass().getResourceAsStream(
22. "cappuccino.jpg")
23. );
25. final Label opacityCaption = newLabel("Opacity Level:");
26. final Label sepiaCaption = newLabel("Sepia Tone:");
27. final Label scalingCaption = newLabel("Scaling Factor:");
29. final Label opacityValue = newLabel(
30. Double.toString(opacityLevel.getValue()));
31. final Label sepiaValue = newLabel(
32. Double.toString(sepiaTone.getValue()));
33. final Label scalingValue = newLabel(
34. Double.toString(scaling.getValue()));
36. final static Color textColor =Color.WHITE;
37. final static SepiaTonesepiaEffect = new SepiaTone();
39. @Override
40. public void start(Stage stage) {
41. Group root = new Group();
42. Scene scene = new Scene(root,480, 400);
43. stage.setScene(scene);
44. stage.setTitle("SliderSample");
45. scene.setFill(Color.BLACK);
47. GridPane grid = newGridPane();
48. grid.setPadding(newInsets(10, 10, 10, 10));
49. grid.setVgap(10);
50. grid.setHgap(70);
52. final ImageView cappuccino =new ImageView (image);
53. cappuccino.setEffect(sepiaEffect);
54. GridPane.setConstraints(cappuccino, 0, 0);
55. grid.setColumnSpan(cappuccino, 3);
56. grid.getChildren().add(cappuccino);
57. scene.setRoot(grid);
59. opacityCaption.setTextFill(textColor);
60. GridPane.setConstraints(opacityCaption, 0, 1);
61. grid.getChildren().add(opacityCaption);
63. opacityLevel.valueProperty().addListener(newChangeListener<Number>() {
64. public voidchanged(ObservableValue<? extends Number> ov,
65. Number old\_val,Number new\_val) {
66. cappuccino.setOpacity(new\_val.doubleValue());
67. opacityValue.setText(String.format("%.2f", new\_val));
68. }
69. });
71. GridPane.setConstraints(opacityLevel, 1, 1);
72. grid.getChildren().add(opacityLevel);
74. opacityValue.setTextFill(textColor);
75. GridPane.setConstraints(opacityValue, 2, 1);
76. grid.getChildren().add(opacityValue);
78. sepiaCaption.setTextFill(textColor);
79. GridPane.setConstraints(sepiaCaption, 0, 2);
80. grid.getChildren().add(sepiaCaption);
82. sepiaTone.valueProperty().addListener(new ChangeListener<Number>(){
83. public voidchanged(ObservableValue<? extends Number> ov,
84. Number old\_val,Number new\_val) {
85. sepiaEffect.setLevel(new\_val.doubleValue());
86. sepiaValue.setText(String.format("%.2f", new\_val));
87. }
88. });
89. GridPane.setConstraints(sepiaTone, 1, 2);
90. grid.getChildren().add(sepiaTone);
92. sepiaValue.setTextFill(textColor);
93. GridPane.setConstraints(sepiaValue, 2, 2);
94. grid.getChildren().add(sepiaValue);
96. scalingCaption.setTextFill(textColor);
97. GridPane.setConstraints(scalingCaption, 0, 3);
98. grid.getChildren().add(scalingCaption);
100. scaling.valueProperty().addListener(new ChangeListener<Number>() {
101. public voidchanged(ObservableValue<? extends Number> ov,
102. Number old\_val,Number new\_val) {
103. cappuccino.setScaleX(new\_val.doubleValue());
104. cappuccino.setScaleY(new\_val.doubleValue());
105. scalingValue.setText(String.format("%.2f", new\_val));
106. }
107. });
108. GridPane.setConstraints(scaling, 1, 3);
109. grid.getChildren().add(scaling);
111. scalingValue.setTextFill(textColor);
112. GridPane.setConstraints(scalingValue, 2, 3);
113. grid.getChildren().add(scalingValue);
115. stage.setVisible(true);
116. }
118. public static void main(String[]args) {
119. Application.launch(args);
120. }
121. }

ImageView对象的透明度属性根据名为透明级的第一个滑动条的值来改变，棕褐色调特效的级别根据棕褐色调滑动条的值来确定，第三个滑动条通过传递参数给setScaleX和setScaleY方法定义了图片的比例因子。

例13-5的代码段演示了由滑动条的getValue方法返回的Double值转为String的方法，还展示了应用格式化将滑动条的浮点值按两位小数来绘制的方法。

例15-3 格式化绘制滑动条的值

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. scalingValue.setText((Double.toString(value)).format("%.2f",value));

下一步可以通过应用可视化特效或CSS样式来改善滑动条的样式。

## 相关API

* Slider
* SpiaTone

# 16 进度条和进度指示器

ProgressIndicator类和它的直接子类ProgressBar提供了指示特定任务正在运行并检测任务运行进度的能力，ProgressBar将进度绘制为一个进度条，ProgressIndicator将进度绘制为一个动态改奕的饼图，如图16-1所示：

http://hi.csdn.net/attachment/201111/3/0_13202963436ItI.gif

图16-1 进度条和进度指示器

## 创建进度控件

可在你的应用程序中使用例16-1中的代码段来插入一个进度控件。

例16-1 进度条和进度指示器的实现

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. ProgressBar pb = new ProgressBar(0.6);
2. ProgressIndicator pi = new ProgressIndicator(0.6);

你也可以通过无参数的 空构造器来创建进度控件，在这种情况下，可以通过使用setProgress方法来赋值。另上种实例化进度控件的方法是使用 ProgressBarBuilder类，它通过build和progress提供了相同的方法。关于本类的更多信息，请参阅API文档。

有时程序无法知道一个任务的准确的完成时间，在这种情况下，进度控件提供一个动态指示直到任务的长度能准确决定。图16-2展示了根据不同的进度值绘制的各种不同的进度状态。

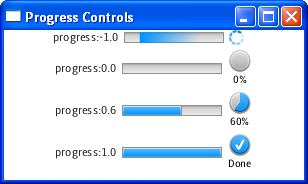


图16-2 进度控件的不同状态

例16-2 展示了这个程序的源代码

例16-2 开启进度控件的不同状态

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.geometry.Pos;
3. import javafx.scene.Group;
4. import javafx.scene.Scene;
5. import javafx.scene.control.Label;
6. import javafx.scene.control.ProgressBar;
7. import javafx.scene.control.ProgressIndicator;
8. import javafx.scene.layout.HBox;
9. import javafx.scene.layout.VBox;
10. import javafx.stage.Stage;
12. public class Main  extendsApplication {
14. final Float[] values = new Float[] {-1.0f, 0f, 0.6f, 1.0f};
15. final Label [] labels = new Label[values.length];
16. final ProgressBar[] pbs = new ProgressBar[values.length];
17. final ProgressIndicator[] pins = new ProgressIndicator[values.length];
18. final HBox hbs [] = new HBox [values.length];
20. @Override
21. public void start(Stage stage) {
22. Group root = new Group();
23. Scene scene = new Scene(root,300, 150);
24. scene.getStylesheets().add("/progresssample/Style.css");
25. stage.setScene(scene);
26. stage.setTitle("ProgressControls");

29. for (int i = 0; i <values.length; i++) {
30. final Label label =labels[i] = new Label();
31. label.setText("progress:" + values[i]);
33. final ProgressBar pb =pbs[i] = new ProgressBar();
34. pb.setProgress(values[i]);
36. final ProgressIndicatorpin = pins[i] = new ProgressIndicator();
37. pin.setProgress(values[i]);
38. final HBox hb = hbs[i] =new HBox();
39. hb.setSpacing(5);
40. hb.setAlignment(Pos.CENTER);
41. hb.getChildren().addAll(label, pb, pin);
42. }
44. final VBox vb = new VBox();
45. vb.setSpacing(5);
46. vb.getChildren().addAll(hbs);
47. scene.setRoot(vb);
48. stage.setVisible(true);
49. }
50. public static void main(String[]args) {
51. Application.launch(args);
52. }
53. }

设为0到1之间的正值用来表表进度控件的百分比，例如：0.4表示40%。负值表示进度条的完成时间不确定。使用isIndeterminate方法来检相进度控制是否属不确定状态。

## 进度指示器

图16-2展现了进度控制的一般状态，在现实的应用程序中，进度值可以通过其他其他UI组件获取。

研究一下例16-3，学习如何根据滑动条位置为进度条和进度指示器设定值。

例16-3 从滑动条接受进度值

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Pos;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.ProgressBar;
8. import javafx.scene.control.ProgressIndicator;
9. import javafx.scene.control.Slider;
10. import javafx.scene.layout.HBox;
11. import javafx.stage.Stage;
13. public class Main  extendsApplication {
15. @Override
16. public void start(Stage stage) {
17. Group root = new Group();
18. Scene scene = newScene(root);
19. stage.setScene(scene);
20. stage.setTitle("ProgressControls");
22. final Slider slider = newSlider();
23. slider.setMin(0);
24. slider.setMax(50);
26. final ProgressBar pb = newProgressBar(0);
27. final ProgressIndicator pi =new ProgressIndicator(0);
29. slider.valueProperty().addListener(new ChangeListener<Number>() {
30. public voidchanged(ObservableValue<? extends Number> ov,
31. Number old\_val,Number new\_val) {
32. pb.setProgress(new\_val.doubleValue()/50);
33. pi.setProgress(new\_val.doubleValue()/50);
34. }
35. });
37. final HBox hb = new HBox();
38. hb.setSpacing(5);
39. hb.setAlignment(Pos.CENTER);
40. hb.getChildren().addAll(slider, pb, pi);
41. scene.setRoot(hb);
42. stage.show();
43. }
44. public static void main(String[]args) {
45. launch(args);
46. }
47. }

可以编译并运行这个程序，产生的输出如图16-3所示。



图16-3 通过滑动条设定进度指示

ChangeListenner对象决定滑动条的值是否改变，并计算进度条和进度指示器的值，以更进度控制的值范围在0.0到1.0之间。

## 相关API

* ProgressBar
* ProgressIndicator

# 17 超链接控件

超链接类代表另一种类型的Labeled控件，用于将格式文本作为超链接使用。图17-1演示了超链接实现的三种状态。



图17-1 超链接控件的三种状态

## 创建超链接

产生超链接的代码如例17-1所示。

例17-1 典型的超链接

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Hyperlink link = new Hyperlink();
2. link.setText("http://example.com");
3. link.setOnAction(new EventHandler<ActionEvent>() {
4. @Override
5. public void handle(ActionEvent e){
6. System.out.println("Thislink is clicked");
7. }
8. });

setText实例方法定义了超链接的标签，由于 超链接继承自Labeled类，你可以设置特定的字体和文本。setOnAction方法设定了动作，当超链接被单击时，动作方法会被调用，与 Button类相应方法类似。在例17-1中，此方法被限制用于打印一个字串，然而，你可以设为实现一个更通用的任务。

## 链接本地内容

图17-2的程序绘制了本地目录中的图像。



图17-2 浏览图像

该程序的代码如下。

例17-2 使用超链接浏览图像

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.scene.\*;
5. import javafx.scene.control.\*;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.layout.VBox;
9. import javafx.stage.Stage;
11. public class Main extends Application {
13. final static String[] imageFiles = new String[]{
14. "product.png",
15. "education.png",
16. "partners.png",
17. "support.png"
18. };
19. final static String[] captions =new String[]{
20. "Products",
21. "Education",
22. "Partners",
23. "Support"
24. };
25. final ImageView selectedImage =new ImageView();
26. final ScrollPane list = newScrollPane();
27. final Hyperlink[] hpls = newHyperlink[captions.length];
28. final Image[] images = newImage[imageFiles.length];
30. public static void main(String[]args) {
31. Application.launch(args);
32. }
34. @Override
35. public void start(Stage stage) {
36. Scene scene = new Scene(newGroup());
37. stage.setTitle("Hyperlink Sample");
38. stage.setWidth(300);
39. stage.setHeight(200);
41. selectedImage.setLayoutX(100);
42. selectedImage.setLayoutY(10);
44. for (int i = 0; i <captions.length; i++) {
45. final Hyperlink hpl =hpls[i] = new Hyperlink(captions[i]);
46. final Image image =images[i] = new Image(
47. getClass().getResourceAsStream(imageFiles[i])
48. );
49. hpl.setOnAction(newEventHandler<ActionEvent>() {
50. @Override
51. public voidhandle(ActionEvent e) {
52. selectedImage.setImage(image);
53. }
54. });
55. }
57. final Button button = newButton("Refresh links");
58. button.setOnAction(newEventHandler<ActionEvent>() {
59. @Override
60. public voidhandle(ActionEvent e) {
61. for (int i = 0; i< captions.length; i++) {
62. hpls[i].setVisited(false);
63. selectedImage.setImage(null);
64. }
65. }
66. });
68. VBox vbox = new VBox();
69. vbox.getChildren().addAll(hpls);
70. vbox.getChildren().add(button);
71. vbox.setSpacing(5);
73. ((Group)scene.getRoot()).getChildren().addAll(vbox, selectedImage);
74. stage.setScene(scene);
75. stage.show();
76. }
77. }

该程序在for循环内创建了4个超链接对象，供每个超链接调用的setOnAction方法定义了用户单击超链接时的行为。在本例中，inages数组中的相应图像被赋给了selectedImage变量。

当用户单击超链接时，它被标记为已访问。你可以使用超链接类的setVisited方法来更新链接。例17-3的代码完成这项工作。

例17-3 更新链接

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final Button button = new Button("Refresh links");
2. button.setOnAction(new EventHandler<ActionEvent>() {
3. @Override
4. public void handle(ActionEvent e){
5. for (int i = 0; i <captions.length; i++) {
6. hpls[i].setVisited(false);
7. selectedImage.setImage(null);
8. }
9. }
10. });

单击更新链接按钮时，所有的超链接变为未访问状态，如图17-3所示。

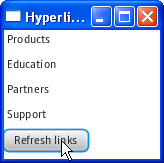


图17-3 未访问链接

由于超链接类继承自Labeled类，你不公可以设定文本标签，还可以设置图像。下节提供的代码使用了文本和图像来创建超链接，并加载了远程HTML页面。

## 链接远程内容

你可以在程序中场景嵌入WebView浏览器来绘制HTML内容。WebView组件提供了基本的Web页面浏览功能。它可以绘制Web页面内容，提供如超链接交互和执行JavaScript代码的能力。

研究一下例17-4的程序，它创建了4个有文本和图像的超链接，当超链接被单击时，相应的值作为URL链接至嵌入的浏览器中。

例17-4 调用远程Web页面。

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.scene.\*;
5. import javafx.scene.control.\*;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.layout.HBox;
9. import javafx.scene.layout.Priority;
10. import javafx.scene.layout.VBox;
11. import javafx.scene.text.Font;
12. import javafx.scene.web.WebEngine;
13. import javafx.scene.web.WebView;
14. import javafx.stage.Stage;
16. public class Main extends Application {
18. final static String[] imageFiles= new String[]{
19. "product.png",
20. "education.png",
21. "partners.png",
22. "support.png"
23. };
24. final static String[] captions =new String[]{
25. "Products",
26. "Education",
27. "Partners",
28. "Support"
29. };
31. final static String[] urls = newString[]{
32. "http://www.oracle.com/us/products/index.html",
33. "http://education.oracle.com/",
34. "http://www.oracle.com/partners/index.html",
35. "http://www.oracle.com/us/support/index.html"
36. };
38. final ImageView selectedImage =new ImageView();
39. final Hyperlink[] hpls = newHyperlink[captions.length];
40. final Image[] images = newImage[imageFiles.length];
42. public static void main(String[]args){
43. launch(args);
44. }
46. @Override
47. public void start(Stage stage) {
48. VBox vbox = new VBox();
49. Scene scene = newScene(vbox);
50. stage.setTitle("Hyperlink Sample");
51. stage.setWidth(570);
52. stage.setHeight(550);
54. selectedImage.setLayoutX(100);
55. selectedImage.setLayoutY(10);
57. final WebView browser = newWebView();
58. final WebEngine webEngine =browser.getEngine();
60. for (int i = 0; i <captions.length; i++) {
61. final Hyperlink hpl =hpls[i] = new Hyperlink(captions[i]);
63. final Image image =images[i] =
64. newImage(getClass().getResourceAsStream(imageFiles[i]));
65. hpl.setGraphic(newImageView (image));
66. hpl.setFont(Font.font("Arial",14));
67. final String url =urls[i];
69. hpl.setOnAction(newEventHandler<ActionEvent>() {
70. @Override
71. public voidhandle(ActionEvent e) {
72. webEngine.load(url);
73. }
74. });
75. }
77. HBox hbox = new HBox();
78. hbox.getChildren().addAll(hpls);
80. vbox.getChildren().addAll(hbox, browser);
81. VBox.setVgrow(browser,Priority.ALWAYS);
83. stage.setScene(scene);
84. stage.show();
85. }
86. }

超链接在for循环中被创建，如例17-2一样。超链接的动作通过ruls数组中相应的URL访问页面，显示于内嵌浏览器的WebEngine对象中。

编译并运行程序，产生的窗口如图17-4所示。

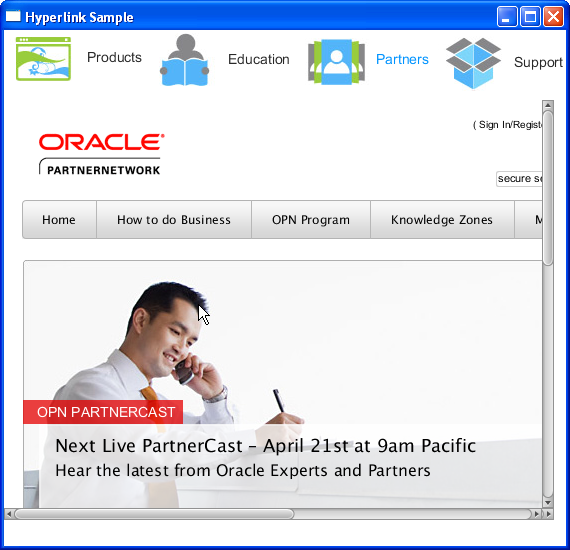


图17-4 从Oracle专业网站调用页面

## 相关API

* Hyperlink
* Labeled
* WebView
* WebEngine

# 18 工具提示

工具提示代表一种常规的UI组件，一般用来显不关于UI控件的其他信息。当鼠标在控件上悬停时，工具提示会显示出来。工具提示只能通过调用控件的setTooltip来设定。

工具提示有二种状态：活动和显示。鼠标在控件上时，它被激活，当工具提供处于显示状态时，它才实际出现。显示出来的工具提示同时也时活动的。从活动到显示通过会有一定的延迟时间。

具有工具提示的密码框如图18-1所示。

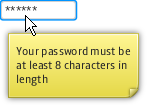


图18-1 加入工具提示的密码框

## 创建工具提示

研究一下例18-1中的代码，它创建了前述的图中所展现的带有工具提示的密码框。

例18-1 为密码框增加工具提示

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final PasswordField pf = new PasswordField();
2. final Tooltip tooltip = new Tooltip();
3. tooltip.setText(
4. "\nYour password mustbe\n" +
5. "at least 8 characters inlength\n"  +
6. );
7. pf.setTooltip(tooltip);

javafx.scene.control包中的每个控件都有setTooltip方法用于设定工具提示。你可以通过Tooptip类的构造器或使用setText方法为工具提示指定提示文本。

由于工具所示类扩展自Labeled类，你还可以设定图标。例18-2的代码为密码框设定了带有图标的工具提示。

例18-2 为工具提示增加图标

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. Image image = new Image(
2. getClass().getResourceAsStream("warn.png")
3. );
4. tooltip.setGraphic(new ImageView(image));

将上述代码增加到程序中编译运行后，图18-2会出现。

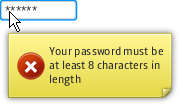


图18-2 带图标的工具提示

工具提示不仅可以包含辅助信息，还可以表现数据。

## 在工具提示中表现程序数据

图18-3的程序使用工具提示中的信息计算旅馆费用。



图18-3 计算旅馆费用

每个复选框都有一个相应的工具提示，每个工具提示显示了实际的登记费用。如果用户选择了一个复选框，相应的值增加到汇总费用中，反之则从汇总费用中扣除。

例18-3 使用工具提示来计算旅馆费用

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Insets;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.CheckBox;
8. import javafx.scene.control.Label;
9. import javafx.scene.control.Tooltip;
10. import javafx.scene.layout.HBox;
11. import javafx.scene.layout.VBox;
12. import javafx.scene.text.Font;
13. import javafx.stage.Stage;

16. public class Main extends Application {
18. final static String[] rooms = newString[]{
19. "Accommodation(BB)",
20. "Half Board",
21. "Late Check-out",
22. "Extra Bed"
23. };
24. final static Integer[] rates =new Integer[]{
25. 100, 20, 10, 30
26. };
27. final CheckBox[] cbs = newCheckBox[rooms.length];
28. final Label total = newLabel("Total: $0");
29. Integer sum = 0;
31. public static void main(String[]args) {
32. launch(args);
33. }
35. @Override
36. public void start(Stage stage) {
37. Scene scene = new Scene(newGroup());
38. stage.setTitle("TooltipSample");
39. stage.setWidth(300);
40. stage.setHeight(150);
42. total.setFont(newFont("Arial", 20));
44. for (int i = 0; i <rooms.length; i++) {
45. final CheckBox cb =cbs[i] = new CheckBox(rooms[i]);
46. final Integer rate =rates[i];
47. final Tooltip tooltip =new Tooltip("{1}quot; + rates[i].toString());
48. tooltip.setFont(newFont("Arial", 16));
49. cb.setTooltip(tooltip);
50. cb.selectedProperty().addListener(new ChangeListener<Boolean>() {
51. public voidchanged(ObservableValue<? extends Boolean> ov,
52. Boolean old\_val,Boolean new\_val) {
53. if(cb.isSelected()) {
54. sum = sum +rate;
55. } else {
56. sum = sum -rate;
57. }
58. total.setText("Total: {1}quot; + sum.toString());
59. }
60. });
61. }
63. VBox vbox = new VBox();
64. vbox.getChildren().addAll(cbs);
65. vbox.setSpacing(5);
66. HBox root = new HBox();
67. root.getChildren().add(vbox);
68. root.getChildren().add(total);
69. root.setSpacing(40);
70. root.setPadding(newInsets(20, 10, 10, 20));
72. ((Group) scene.getRoot()).getChildren().add(root);
74. stage.setScene(scene);
75. stage.show();
76. }
77. }

例18-4中的代码行在例18-3中被用来创建工具提示并将文本赋予它，选项价格的Integer值转换为String值。

例18-4 为工具提示设定值

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. final Tooltip tooltip = new Tooltip("{1}quot; + rates[i].toString())

你可通过使用CSS为工具提示设定样式。

## 相关API

* Tooltip
* Labeled

# 19 HTML编辑器

HTMLEditor控件是一个全功能的富文本编辑器，除基本编辑功能外，它不支持以下特点：

* 文本格式化，包括加粗、倾斜、下划线、穿透线
* 段落设定，如格式和字体
* 前景和背景色
* 文本缩进
* 符号列表和数字列表
* 文本对齐
* 水平标尺
* 拷贝和粘贴

图19-1展示了增加到JavaFX程序中的富文本编辑器

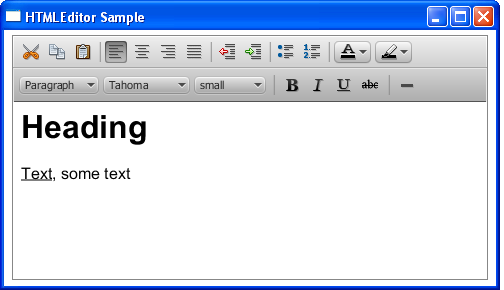


图19-1 HTML编辑器

HTMLEditor类以HTML字串的形式表示 编辑内容，例如：在图19-1的编辑器中键入的内容用下列的字串表 示，"<html><head></head><bodycontenteditable="true"& gt;<h1>Heading</h1><div><u>Text</u>,some text</div></body></html>."

由于HTMLEditor类属Node类的子类，你可以对HTMLEditor类的实例应用可视效果或变换。

## 增加HTML编辑器

和其他UI控件一样，HTMLEditor组件必须增加到场景中，以便它可以出现在应用程序中。你可以像例19-1所示的一样直接将它增加到场景中，也可以像其他程序一样通过布局容器来增加。

例19-1 向JavaFX程序中增加HTML编辑器

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.scene.Scene;
3. import javafx.scene.web.HTMLEditor;
4. import javafx.stage.Stage;
6. public class HTMLEditorSample extends Application {
8. @Override
9. public void start(Stage stage) {
10. stage.setTitle("HTMLEditor Sample");
11. stage.setWidth(400);
12. stage.setHeight(300);
13. final HTMLEditor htmlEditor =new HTMLEditor();
14. htmlEditor.setPrefHeight(245);
15. Scene scene = newScene(htmlEditor);
16. stage.setScene(scene);
17. stage.show();
18. }
20. public static void main(String[]args) {
21. launch(args);
22. }
23. }

编译并运行上述代码，产生的窗口如图19-2所示。

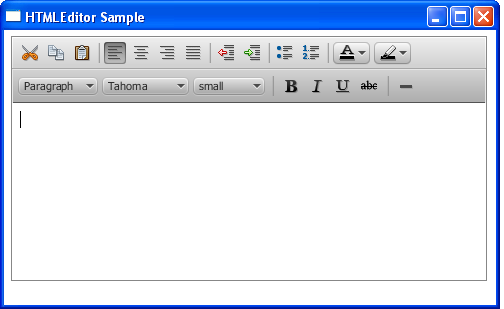


图19-2HTMLEditor组件的初始视图

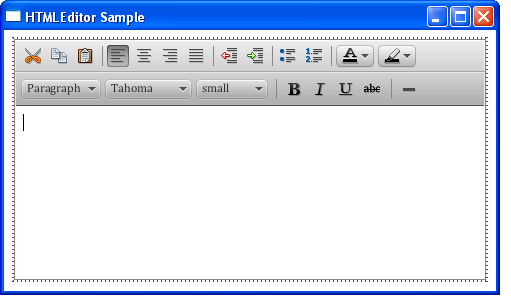
该组件实现了格式化工具条，你不能关闭它。但你仍然可以通过CSS样式定制外观，如例19-2所示。

例19-2 样式化HTML编辑器

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. htmlEditor.setStyle(
2. "-fx-font: 12 cambria;"
3. + "-fx-border-color: brown;"
4. + "-fx-border-style:dotted;"
5. + "-fx-border-width:2;"
6. );

当上述代码增加到例19-1中后，编辑器变化了，如图19-3所示。



 例19-3 另一种HTMLEditor视图

应用的样式改变了组件的边界和工具条的字体。

HTMLEditor类提供了一个方法来定义出现在编辑区的内容，使用setHtmlText方法，如例19-3所示，可以设定编辑器中的初始文本。

例19-3 设定文本内容

**[html]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. private final String INITIAL\_TEXT = "<html><body>Loremipsum dolor sit "
2. + "amet, consecteturadipiscing elit. Nam tortor felis, pulvinar "
3. + "in scelerisque cursus,pulvinar at ante. Nulla consequat"
4. + "congue lectus in sodales.Nullam eu est a felis ornare "
5. + "bibendum et nec tellus.Vivamus non metus tempus augue auctor "
6. + "ornare. Duis pulvinarjusto ac purus adipiscing pulvinar. "
7. + "Integer congue faucibusdapibus. Integer id nisl ut elit "
8. + "aliquam sagittis gravidaeu dolor. Etiam sit amet ipsum "
9. +"sem.</body></html>";
10. htmlEditor.setHtmlText(INITIAL\_TEXT);

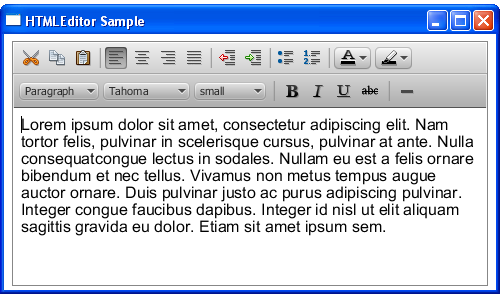


图19-4 具有预定义内容的HTMLEditor

你可以在这些字串中使用HTML标签来为初始绘制的内容使用特定的格式。

## 使用HTML编辑器来创建用户界面

你可以使用HTMLEditor控件在你的应用程序中实现典型的用户界面。例如，你可实现即时通讯服务，email客户端或者内容管理系统。表现在很多email客端程序常见的信息综合窗口的用户界面。

例19-4 将HTMLEditor增加到Email客户端的UI中

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.collections.FXCollections;
3. import javafx.geometry.Insets;
4. import javafx.geometry.Pos;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.\*;
8. import javafx.scene.layout.GridPane;
9. import javafx.scene.layout.VBox;
10. import javafx.scene.web.HTMLEditor;
11. import javafx.stage.Stage;
13. public class HTMLEditorSample extends Application {
15. @Override
16. public void start(Stage stage) {
17. stage.setTitle("MessageComposing");
18. stage.setWidth(500);
19. stage.setHeight(500);
20. Scene scene = new Scene(newGroup());
22. final VBox root = newVBox();
23. root.setPadding(new Insets(8,8, 8, 8));
24. root.setSpacing(5);
25. root.setAlignment(Pos.BOTTOM\_LEFT);
27. final GridPane grid = newGridPane();
28. grid.setVgap(5);
29. grid.setHgap(10);
31. final ChoiceBox sendTo =
32. newChoiceBox(FXCollections.observableArrayList(
33. "To:","Cc:", "Bcc:")
34. );
36. sendTo.setPrefWidth(100);
37. GridPane.setConstraints(sendTo, 0, 0);
38. grid.getChildren().add(sendTo);
40. final TextField tbTo = newTextField();
41. tbTo.setPrefWidth(400);
42. GridPane.setConstraints(tbTo,1, 0);
43. grid.getChildren().add(tbTo);
45. final Label subjectLabel =new Label("Subject:");
46. GridPane.setConstraints(subjectLabel, 0, 1);
47. grid.getChildren().add(subjectLabel);
49. final TextField tbSubject =new TextField();
50. tbTo.setPrefWidth(400);
51. GridPane.setConstraints(tbSubject, 1, 1);
52. grid.getChildren().add(tbSubject);
54. root.getChildren().add(grid);
56. final HTMLEditor htmlEditor =new HTMLEditor();
57. htmlEditor.setPrefHeight(370);
59. root.getChildren().addAll(htmlEditor, newButton("Send"));
61. final Label htmlLabel = newLabel();
62. htmlLabel.setWrapText(true);
64. scene.setRoot(root);
65. stage.setScene(scene);
66. stage.show();
67. }
69. public static void main(String[]args) {
70. launch(args);
71. }
72. }

用户界面包含一个选择框来选择收件人的类型，两个文本框用于录入邮件地址和主题信息，标签用于标识主题字段，还包含了编辑器和发送按钮。

UI控件通过Grid和VBox安排在程序的场景中。

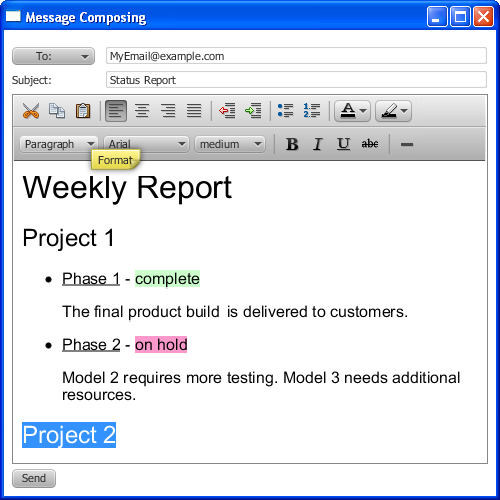


图19-5 邮件客户端界面

可以通过调用setPrefWidth和setPrefHeight方法为HTMLEditor对象指定高和宽。也可以保持默认的高和宽。例19-4指定了组件的高。它的宽度通过Vbox容器来定义。当文本内容超过编辑区的高度时，垂直滚动条会出现。

## 获得HTML内容

通过JavaFX的HTMLEditor控件，你可以编辑文本并设定和初始内容。另外，你可以获得通过HTML格式获得键入和编辑的内容。例19-5所示的程序实现了这个任务。

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.geometry.Insets;
5. import javafx.geometry.Pos;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.control.\*;
9. import javafx.scene.layout.VBox;
10. import javafx.scene.web.HTMLEditor;
11. import javafx.stage.Stage;
13. public class HTMLEditorSample extends Application {
14. private final String INITIAL\_TEXT= "Lorem ipsum dolor sit "
15. + "amet, consecteturadipiscing elit. Nam tortor felis, pulvinar "
16. + "in scelerisquecursus, pulvinar at ante. Nulla consequat"
17. + "congue lectus insodales. Nullam eu est a felis ornare "
18. + "bibendum et nectellus. Vivamus non metus tempus augue auctor "
19. + "ornare. Duispulvinar justo ac purus adipiscing pulvinar. "
20. + "Integer conguefaucibus dapibus. Integer id nisl ut elit "
21. + "aliquam sagittisgravida eu dolor. Etiam sit amet ipsum "
22. + "sem.";
24. @Override
25. public void start(Stage stage) {
26. stage.setTitle("HTMLEditor Sample");
27. stage.setWidth(500);
28. stage.setHeight(500);
29. Scene scene = new Scene(newGroup());
31. VBox root = new VBox();
32. root.setPadding(new Insets(8,8, 8, 8));
33. root.setSpacing(5);
34. root.setAlignment(Pos.BOTTOM\_LEFT);
36. final HTMLEditor htmlEditor =new HTMLEditor();
37. htmlEditor.setPrefHeight(245);
38. htmlEditor.setHtmlText(INITIAL\_TEXT);
40. final TextArea htmlCode = newTextArea();
41. htmlCode.setWrapText(true);
43. ScrollPane scrollPane = newScrollPane();
44. scrollPane.getStyleClass().add("noborder-scroll-pane");
45. scrollPane.setContent(htmlCode);
46. scrollPane.setFitToWidth(true);
47. scrollPane.setPrefHeight(180);
49. Button showHTMLButton = newButton("Produce HTML Code");
50. root.setAlignment(Pos.CENTER);
51. showHTMLButton.setOnAction(new EventHandler<ActionEvent>() {
52. @Override public voidhandle(ActionEvent arg0) {
53. htmlCode.setText(htmlEditor.getHtmlText());
54. }
55. });
57. root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
58. scene.setRoot(root);
60. stage.setScene(scene);
61. stage.show();
62. }
64. public static void main(String[]args) {
65. launch(args);
66. }
67. }

HTMLEditor对象调用getHTMLText方法将编辑的内容和表现分离为HTML字串。这个信息发送至文本区，以便能观察、复制、粘贴产生的HTML代码。图19-6展示了在HTMLEditor编辑器中编辑文本的例子。

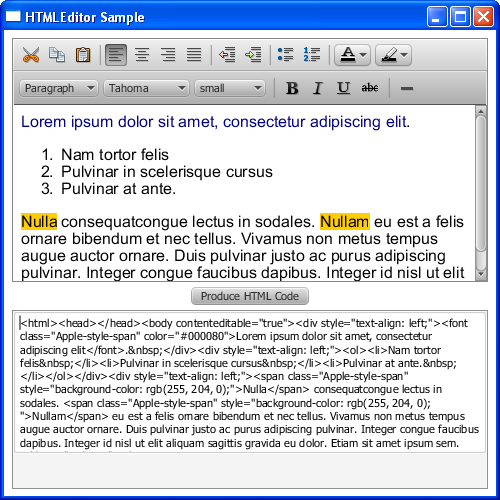


 图19-6 获得HTML内容

同样，你可以获取HTML代码并保存在文件或发送到WebView对象以同步编辑器和嵌入的浏览器。例19-6展示了如何实现这个任务。

例19-6 在浏览器中绘制编辑的HTML内容

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.event.ActionEvent;
3. import javafx.event.EventHandler;
4. import javafx.geometry.Insets;
5. import javafx.geometry.Pos;
6. import javafx.scene.Group;
7. import javafx.scene.Scene;
8. import javafx.scene.control.\*;
9. import javafx.scene.layout.VBox;
10. import javafx.scene.web.HTMLEditor;
11. import javafx.scene.web.WebEngine;
12. import javafx.scene.web.WebView;
13. import javafx.stage.Stage;
15. public class HTMLEditorSample extends Application {
16. private final String INITIAL\_TEXT= "Lorem ipsum dolor sit "
17. + "amet, consecteturadipiscing elit. Nam tortor felis, pulvinar "
18. + "in scelerisquecursus, pulvinar at ante. Nulla consequat"
19. + "congue lectus insodales. Nullam eu est a felis ornare "
20. + "bibendum et nectellus. Vivamus non metus tempus augue auctor "
21. + "ornare. Duispulvinar justo ac purus adipiscing pulvinar. "
22. + "Integer conguefaucibus dapibus. Integer id nisl ut elit "
23. + "aliquam sagittisgravida eu dolor. Etiam sit amet ipsum "
24. + "sem.";
26. @Override
27. public void start(Stage stage) {
28. stage.setTitle("HTMLEditor Sample");
29. stage.setWidth(500);
30. stage.setHeight(500);
31. Scene scene = new Scene(newGroup());
33. VBox root = new VBox();
34. root.setPadding(new Insets(8,8, 8, 8));
35. root.setSpacing(5);
36. root.setAlignment(Pos.BOTTOM\_LEFT);
38. final HTMLEditor htmlEditor =new HTMLEditor();
39. htmlEditor.setPrefHeight(245);
40. htmlEditor.setHtmlText(INITIAL\_TEXT);
42. final WebView browser = newWebView();
43. final WebEngine webEngine =browser.getEngine();
45. ScrollPane scrollPane = newScrollPane();
46. scrollPane.getStyleClass().add("noborder-scroll-pane");
47. scrollPane.setStyle("-fx-background-color: white");
48. scrollPane.setContent(browser);
49. scrollPane.setFitToWidth(true);
50. scrollPane.setPrefHeight(180);
52. Button showHTMLButton = newButton("Load Content in Browser");
53. root.setAlignment(Pos.CENTER);
54. showHTMLButton.setOnAction(new EventHandler<ActionEvent>() {
55. @Override public void handle(ActionEventarg0) {
56. webEngine.loadContent(htmlEditor.getHtmlText());
57. }
58. });
60. root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
61. scene.setRoot(root);
63. stage.setScene(scene);
64. stage.show();
65. }
67. public static void main(String[]args) {
68. launch(args);
69. }
70. }

从htmlEditor组件接收的HTML代码被加载到为嵌入的浏览器指定内容的WebEngine对象中，每次当用户单击“Load Content in Browser”按钮时，编辑内容在浏览器中得到更新。图19-7演示了这个动作。

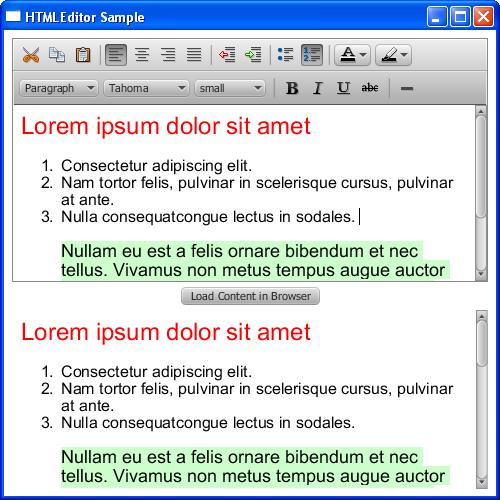


图19-7 在浏览器中加载内容

你可以使用Text组件来添加非编辑内容，参阅“在JavaFX中使用Text和Text效果”获得更多关于Text组件的信息。

## 相关API

* HTMLEditor
* WebView
* WebEngine
* Label
* Button
* TextField
* ChoiceBox
* ScrollPane

# 20 标题面板和可折叠面板

标题面板是一个带标题的面板，它可以打开或关闭，还可以封装任何的结点，比如UI控件和图像，以及增加到布局容器中的UI元素组。

标题面板可以通过折叠控件分组，以更可以创建多个面板并一次显示出来。图20-1展示了一个包含三个标题面板的折叠控件。

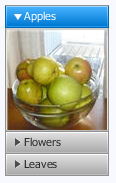


图20-1 有三个标题面板的折叠面板

使用JavaFX API中的Accordion类和TitlePane类可在你的应用程序中实现这些控件。

## 创建标题面板

创建一个标题控件需定义一个标题和一些内容。你可以通过使用TitlePane类的两参数构造器来创建，或者使用setText和setContent方法。两种方法均在例20-1中体现了。

例20-1 声明一个标题面板对象

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. //使用两参数构造器
2. TitledPane tp = new TitledPane("My Titled Pane", newButton("Button"));
3. //使用方法
4. TitledPane tp = new TitledPane();
5. tp.setText("My Titled Pane");
6. tp.setContent(new Button("Button"));

编译和运行上述两种方法之一，均会生成图20-2所示的控件。

http://hi.csdn.net/attachment/201111/3/0_1320297834LEk6.gif

图20-2 有按钮的标题面板

标题面板自动调整尺寸以容纳这些内容的尺寸。你可以增加多行文本来测试在图20-3中的结果。

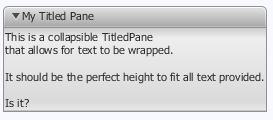


图20-3 带文本的标题面板

不需要明确地设定标题面板的最小、最大或最佳尺寸，因为这些方法会在面板打开或关闭时带来不可预期的行为。

例20-2中所示的代码段通过将它们集成到GridPane中的方式在标题面板中增加了几个控件。

例20-2 带GridPane布局容器的标题面板

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. TitledPane gridTitlePane = new TitledPane();
2. GridPane grid = new GridPane();
3. grid.setVgap(4);
4. grid.setPadding(new Insets(5, 5, 5, 5));
5. grid.add(new Label("First Name: "), 0, 0);
6. grid.add(new TextField(), 1, 0);
7. grid.add(new Label("Last Name: "), 0, 1);
8. grid.add(new TextField(), 1, 1);
9. grid.add(new Label("Email: "), 0, 2);
10. grid.add(new TextField(), 1, 2);
11. gridTitlePane.setText("Grid");
12. gridTitlePane.setContent(grid);

编译并运行上述代码，输出如图20-4所示：

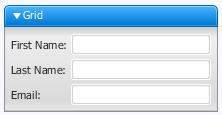


图20-4 包含几个控件的标题面板

你可以定义标题面板打开或关闭的方式。默认地，标题面板是可收缩并且会动态地移动。如果低迷的应用程序禁止关闭标题面板，可向setCollalsible方法传递一个false参数。例20-3中的代码实现了此任务。

例20-3 调整标题面板的样式

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. TitledPane tp = new TitledPane();
2. //prohibit closing
3. tp.setCollapsible(false);
4. //prohibit animating
5. tp.setAnimated(false);

## 将标题面板增加到折叠面板

在你的应用程序中，你可以将标题面板作为独立的元素，也可以通过使用Accordion控件将它们集成到一起。不需要明确地设定折叠面板的最小、最大或最佳尺寸，因为这些方法会在面板打开或关闭时带来不可预期的行为。

将标题面板增加到折叠面板中与在按钮组中增加按钮类似。在折叠面板中，同一时刻只能打开一个标题面板。例20-4创建了三个面板并把它们增加到了折叠面板中。

例20-4 折叠面板和三个标题面板

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.scene.Group;
3. import javafx.scene.Scene;
4. import javafx.scene.control.Accordion;
5. import javafx.scene.control.TitledPane;
6. import javafx.scene.image.Image;
7. import javafx.scene.image.ImageView;
8. import javafx.scene.paint.Color;
9. import javafx.stage.Stage;
11. public class TitledPaneSample extends Application {
12. final String[] imageNames = newString[]{"Apples", "Flowers", "Leaves"};
13. final Image[] images = newImage[imageNames.length];
14. final ImageView[] pics = newImageView[imageNames.length];
15. final TitledPane[] tps = newTitledPane[imageNames.length];
17. public static void main(String[]args) {
18. launch(args);
19. }
21. @Override public void start(Stagestage) {
22. stage.setTitle("TitledPane");
23. Scene scene = new Scene(newGroup(), 80, 180);
24. scene.setFill(Color.GHOSTWHITE);
26. final Accordion accordion =new Accordion ();
28. for (int i = 0; i <imageNames.length; i++) {
29. images[i] = new
30. Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
31. pics[i] = newImageView(images[i]);
32. tps[i] = newTitledPane(imageNames[i],pics[i]);
33. }
34. accordion.getPanes().addAll(tps);
35. accordion.setExpandedPane(tps[0]);
37. Group root =(Group)scene.getRoot();
38. root.getChildren().add(accordion);
39. stage.setScene(scene);
40. stage.show();
41. }
42. }

三个标题面板在循环内创建，标题按钮的内容作为一个ImageView对象创建。标题面板通过使用getPanes和addAll方法增加到折叠面板中，你可以使用add 方法替换addAll方法单独地增加标题面板。

默认情况下，标题面板在程序开始时是收缩起来的。在运行例子中的程序时， setExpandedPane方法指定了装有苹果图片的面板被打开。如图20-5所示：

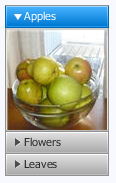


图20-5 有三个标题面板的折叠面板

## 为折叠面板处理事件

你可以在程序中使用标题面板和折叠面板来表现不同的数据，例20-5创建了一个由GridPane布局容器管理的独立面板和用折叠面板集成的三个标题面板。独立的标题面板包含了一个email客户端的UI元件，折叠面板使得选择的图像出现在Grid标题面板的附件栏中。

例20-5 实现折叠面板的ChangeListner

**[java]** [view plaincopy](http://blog.csdn.net/lzmailcn/article/details/6931896)

1. import javafx.application.Application;
2. import javafx.beans.value.ChangeListener;
3. import javafx.beans.value.ObservableValue;
4. import javafx.geometry.Insets;
5. import javafx.scene.Group;
6. import javafx.scene.Scene;
7. import javafx.scene.control.Accordion;
8. import javafx.scene.control.Label;
9. import javafx.scene.control.TextField;
10. import javafx.scene.control.TitledPane;
11. import javafx.scene.image.Image;
12. import javafx.scene.image.ImageView;
13. import javafx.scene.layout.GridPane;
14. import javafx.scene.layout.HBox;
15. import javafx.scene.paint.Color;
16. import javafx.stage.Stage;
18. public class TitledPaneSample extends Application {
19. final String[] imageNames = newString[]{"Apples", "Flowers", "Leaves"};
20. final Image[] images = newImage[imageNames.length];
21. final ImageView[] pics = newImageView[imageNames.length];
22. final TitledPane[] tps = newTitledPane[imageNames.length];
23. final Label label = newLabel("N/A");
25. public static void main(String[]args) {
26. launch(args);
27. }
29. @Override public void start(Stagestage) {
30. stage.setTitle("TitledPane");
31. Scene scene = new Scene(newGroup(), 800, 250);
32. scene.setFill(Color.GHOSTWHITE);
34. // --- GridPane container
35. TitledPane gridTitlePane =new TitledPane();
36. GridPane grid = newGridPane();
37. grid.setVgap(4);
38. grid.setPadding(new Insets(5,5, 5, 5));
39. grid.add(new Label("To:"), 0, 0);
40. grid.add(new TextField(), 1, 0);
41. grid.add(new Label("Cc:"), 0, 1);
42. grid.add(new TextField(), 1,1);
43. grid.add(newLabel("Subject: "), 0, 2);
44. grid.add(new TextField(), 1,2);
45. grid.add(newLabel("Attachment: "), 0, 3);
46. grid.add(label,1, 3);
47. gridTitlePane.setText("Grid");
48. gridTitlePane.setContent(grid);
50. // --- Accordion
51. final Accordion accordion =new Accordion ();
52. for (int i = 0; i <imageNames.length; i++) {
53. images[i] = new
54. Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
55. pics[i] = newImageView(images[i]);
56. tps[i] = newTitledPane(imageNames[i],pics[i]);
57. }
58. accordion.getPanes().addAll(tps);
59. accordion.expandedPaneProperty().addListener(new
60. ChangeListener<TitledPane>() {
61. public voidchanged(ObservableValue<? extends TitledPane> ov,
62. TitledPaneold\_val, TitledPane new\_val) {
63. if (new\_val!= null) {
64. label.setText(accordion.getExpandedPane().getText() +
65. ".jpg");
66. }
67. }
68. });
70. HBox hbox = new HBox(10);
71. hbox.setPadding(newInsets(20, 0, 0, 20));
72. hbox.getChildren().setAll(gridTitlePane, accordion);
74. Group root =(Group)scene.getRoot();
75. root.getChildren().add(hbox);
76. stage.setScene(scene);
77. stage.show();
78. }
79. }

当用户在折叠面板中打开标题面板时，折叠面板的expandedPaneProperty属性改变，ChangeListner对象发送这个变化，然后折叠面板中展开的标题面板用来构造捕获的文件名。这个文件名作为附件标签对象的文本。

图20-6展示了当程序开始后的样式，捕获的标签包含“N/A”，因为还没有标题面板被选中。



图20-6 标题面板的初始视图

如果你展开了叶子标题面板，捕获标签将包含“Leaves.jpg”，如图20-7所示。

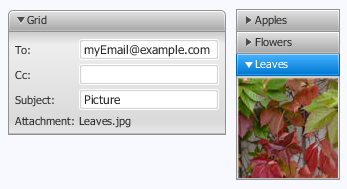


图20-7 叶子标题面板打开时的标题面板的例子

由于TitlePane和Accordion类都属于Node类的子类，你可以对它使用可视化特效或变换。也可以使用CSS样式来改变控件的外观。

## 相关API

* TitlePane
* Acoordion
* Label
* GridPane
* TextField

# [JavaFX2 中CSS使用的几个问题总结](http://aiyooyoo.com/index.php/archives/573/)

1.javaFx中引入CSS报错  
使用如下代码

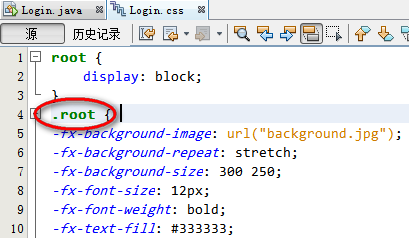
scene.getStylesheets().add(this.getClass().getResource("login.css").toExternalForm());

为JavaFX引入CSS，结果报  
Exception in Application start method Exception in thread "main" java.lang.RuntimeException: Exception in Application start method at com.sun.javafx.application.LauncherImpl.launchApplication1(Unknown Source).  
....  
Caused by: java.lang.NullPointerException  
  
很明显，这样的问题是由于CSS文件找不到造成的。很可能是由于你的IDE不会自动把CSS文件从工程的SRC目录拷贝到build目录造成的。可以手工拷贝一下，或者清理项目重新生成。  
参考网址：<http://stackoverflow.com/questions/9862113/javafx-2-0-xychart-and-css>  
  
2..javaFx中引入CSS无效果  
症状描述：使用代码

scene.getStylesheets().add(this.getClass().getResource("login.css").toExternalForm());

引入CSS，没有报错，但是也没有CSS效果，而是用下面的方式却是可行的：

grid.setStyle("-fx-background-image: url(\"ThghN.png\"); -fx-background-repeat: stretch; -fx-background-size: 300 250;");

问题原因：（1）可能你是用的是较早的javaFx版本，这是其中的一个BUG，解决办法就是升级版本（相关帖子：<https://forums.oracle.com/forums/thread.jspa?threadID=2229859>）。  
（2）还有可能的原因就是CSS写错了。注意CSS中，选择器是.root，而不是root。在使用IDE生成代码时容易忽略，这个小问题害我排查了将近两个多小时。。。郁闷了  
[](http://aiyooyoo.com/index.php/attachment/572/)  
  
最后，附上手册地址：<http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>

# Use JavaFX2 with Spring

Spring is more than a simple IoC container, it gives you access to a whole platform. Besides the benifits of injecting Nodes or parts of your application UI you can benefit of the whole Spring infrastructure. The following examle demonstrates how to define a JavaFX2 / Spring application.

To start a JavaFX 2 application we usually extend from javafx.application.Application and implement a main method to call the static Application.launch() method. Defining a \*.Application bean didn’t succeed for me so let’s create an abstract launcher class which extends from javafx.application.Application and contains the Spring context:

[view source](http://java.dzone.com/articles/use-javafx2-spring#viewSource)

[print](http://java.dzone.com/articles/use-javafx2-spring#printSource)[?](http://java.dzone.com/articles/use-javafx2-spring#about)

01.public abstract class ASpringFX extends Application {

02.private ClassPathXmlApplicationContext context = null;

03.private final String springXML;

04.private Node root;

05.private final String rootName;

06.

07.public ASpringFX(final String springXML) {

08.this.springXML = springXML;

09.this.rootName = null;

10.this.context = new ClassPathXmlApplicationContext(

11.new String[] { springXML });

12.}

13.

14.@Override

15.public void start(Stage stage) throws Exception {

16.root = (Node) this.context.getBean(rootName!=null?rootName:"root");

17.startFXApplication(stage);

18.}

19.

20.public String getSpringXML() {

21.return springXML;

22.}

23.

24.public abstract void startFXApplication(Stage stage);

25.}

The start(Stage stage) method I use to get the root bean defined in the Spring context and to pass the stage to an abstract method. You can easilly adjust the code and pass the root node too…

Now we need a Class with a main method that is bootstrapping the JavaFX Application and the Spring context.

[view source](http://java.dzone.com/articles/use-javafx2-spring#viewSource)

[print](http://java.dzone.com/articles/use-javafx2-spring#printSource)[?](http://java.dzone.com/articles/use-javafx2-spring#about)

01.public class SpringFX extends ASpringFX{

02.

03.public SpringFX() {

04.super("main.xml");

05.}

06.

07.public static void main(final String[] args) {

08.Application.launch(args);

09.}

10.

11.@Override

12.public void startFXApplication(Stage stage) {

13.Group group = new Group();

14.Scene scene = new Scene(group, 800, 600, Color.BLACK);

15.RootComponent root = (RootComponent) this.getRoot();

16.List<Node> leafs = root.getLeafs();

17.root.getItems().addAll(leafs);

18.group.getChildren().add(root);

19.stage.setScene(scene);

20.stage.show();

21.}

22.

23.}

 You can see “this.getRoot()” returning the root node from Spring context. This component is a SplitPane and contains the leaf Nodes injected by Spring.

 So create the root and the leaf components :

[view source](http://java.dzone.com/articles/use-javafx2-spring#viewSource)

[print](http://java.dzone.com/articles/use-javafx2-spring#printSource)[?](http://java.dzone.com/articles/use-javafx2-spring#about)

01.public class RootComponent extends SplitPane {

02.private List<Node> leafs;

03.

04.public RootComponent() {

05.this.setPrefSize(800, 600);

06.}

07.

08.public List<Node> getLeafs() {

09.return leafs;

10.}

11.

12.public void setLeafs(List<Node> leafs) {

13.this.leafs = leafs;

14.}

15.}

 and

[view source](http://java.dzone.com/articles/use-javafx2-spring#viewSource)

[print](http://java.dzone.com/articles/use-javafx2-spring#printSource)[?](http://java.dzone.com/articles/use-javafx2-spring#about)

01.public class LeafOne extends GridPane {

02.

03.public LeafOne() {

04.Pane content = new Pane();

05.GridPane.setHgrow(content, Priority.ALWAYS);

06.GridPane.setVgrow(content, Priority.ALWAYS);

07.content.getChildren().add(new Label("left"));

08.this.getChildren().add(content);

09.}

10.

11.}

 and

[view source](http://java.dzone.com/articles/use-javafx2-spring#viewSource)

[print](http://java.dzone.com/articles/use-javafx2-spring#printSource)[?](http://java.dzone.com/articles/use-javafx2-spring#about)

01.public class LeafTwo extends GridPane {

02.

03.public LeafTwo() {

04.Pane content = new Pane();

05.GridPane.setHgrow(content, Priority.ALWAYS);

06.GridPane.setVgrow(content, Priority.ALWAYS);

07.content.getChildren().add(new Label("right"));

08.this.getChildren().add(content);

09.}

10.

11.}

 Finaly create a spring.xml and define the beans (I prefer xml to show the hierarchie)…

[view source](http://java.dzone.com/articles/use-javafx2-spring#viewSource)

[print](http://java.dzone.com/articles/use-javafx2-spring#printSource)[?](http://java.dzone.com/articles/use-javafx2-spring#about)

01.<?xml version="1.0" encoding="UTF-8"?>

02.<beans xmlns="<http://www.springframework.org/schema/beans>"

03.xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

04.xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>">

05.<bean id="root" class="org.jacp.JavaFXSpring.RootComponent">

06.<property name="leafs">

07.<list>

08.<ref bean="leafOne" />

09.<ref bean="leafTwo" />

10.</list>

11.</property>

12.</bean>

13.<bean id="leafOne" class="org.jacp.JavaFXSpring.LeafOne"/>

14.<bean id="leafTwo" class="org.jacp.JavaFXSpring.LeafTwo"/>

15.</beans>

And voilà … you defined JavaFX 2 Nodes as Spring beans. You can use the root as well as the leaf nodes like avarage Spring beans and use the whole Spring infrastructure.