Chapter 5 第5章 JavaFX 菜单

菜单是 GUI 的重要组成部分，它可以让用户访问程序的核心功能，所以 JavaFX 为菜单提供了广泛的支持。JavaFX 的一个主要优势在于通过使用效果/变换改变控件的精确外观。通过这个功能，可以让 GUI 具有用户期望的复杂现代外观。本章将介绍开发 JavaFX 菜单应用程序以及让 GUI 具有用户期望的外观的原理与方法。

5.1 基础知识

JavaFX 的菜单系统由 javafx.scene.control 包中的一系列相关的类提供支持，如表 5.1 所示。

表 5.1 JavaFX 的核心菜单类

类	主 要 功 能
CheckMenuItem	复选菜单项
ContextMenu	弹出菜单
Menu	标准菜单，由一个或多个 MenuItem 组成
MenuBar	保存程序的顶级菜单的对象
MenuItem	填充菜单的对象
RadioMenuItem	单选菜单项
SeparatorMenuItem	菜单项之间的可视分隔符

- 如果要创建程序的顶级菜单，首先要创建一个 MenuBar 实例，即这个类是菜单的容器。在 MenuBar 实例中，将添加 Menu 实例。每个 Menu 对象定义了一个菜单，也就是说，每个 Menu 对象包含一个或多个可以选择的菜单项。Menu 显示的菜单项是 MenuItem 类型的对象。因此，

MenuItem 定义了用户可以选择的选项。

- 除了标准菜单项，还可以在菜单中包含复选菜单项和单选菜单项，它们的操作与复选框和单选按钮控件类似。复选菜单项用 CheckMenuItem 类创建，单选菜单项用 RadioMenuItem 类创建。这两个类扩展了 MenuItem 类。
- SeparatorMenuItem 类用于在菜单中创建一条分隔线，它继承了 CustomMenuItem 类，后者使得在菜单中嵌入其他类型的控件变得容易。CustomMenuItem 类扩展了 MenuItem 类。
- MenuItem 类没有继承 Node 类。因此，MenuItem 类的实例只能用在菜单中，而不能以其他方式加入场景图中。但是，MenuBar 类继承了 Node 类，所以可以把菜单栏添加到场景图中。MenuItem 是 Menu 的超类，它可以创建子菜单，也就是菜单中的菜单。要创建子菜单，首先要创建一个 Menu 对象，并用 MenuItem 填充它，然后把它添加到另一个 Menu 对象中。
- 选择菜单项后，会生成动作事件。与所选项关联的文本称为选择的名称，所以不需要通过检查名称确定哪个菜单项被选择。
- 也可以创建独立的上下文菜单，它们在激活时会被弹出。首先，需要创建一个 ContextMenu 类的对象。然后，向该对象添加 MenuItem。如果为某个控件定义了上下文菜单，那么激活该菜单的方式通常是在该控件上右击。ContextMenu 类继承了 PopupControl 类。
- 工具栏是与菜单相关的一种特性。工具栏由 ToolBar 类支持。该类创建独立的组件，通常用于快速访问程序菜单中包含的功能。

5.2　MenuBar、Menu 和 MenuItem 概述

要为程序创建菜单，最少要用到 MenuBar、Menu 以及 MenuItem 这 3 个类。上下文菜单也会用到 MenuItem。因此，这 3 个类是菜单系统的基础。

1. MenuBar

Menubar 是菜单的容器，它是为程序提供主菜单的控件。MenuBar 类继承了 Node 类，因此可以把它添加到场景图中。MenuBar 有两个构造方法，第一个是默认的构造方法，需要在使用之前在其中填充菜单；第二个构造方法允许指定初始的菜单栏列表。一般地，程序有且只有一个菜单栏。Menubar 定义的方法中，getMenus( )方法经常被使用，它返回一个由菜单栏管理的菜单列表。创建的菜单将被添加到这个列表中。

```
final ObservableList<Menu> getMenus( )
```

调用 add( )方法可以把 Menu 实例添加到这个菜单列表中，也可以用 addAll( )方法在一次调用中可以添加两个或多个 Menu 实例。添加的菜单将按照添加顺序从左到右排列在菜单中。如果要在特定位置添加一个菜单，则可以使用以下 add( )方法。

```
void add(int idx, Menu menu)
```

- Menu 将被添加到由 idx 指定的索引位置。索引从 0 开始，0 对应最左边的菜单。

当需要删除不再需要的菜单时，可以通过对 getMenus( )方法返回的 ObservableList 调用 remove( )方法实现。该方法的两种定义形式如下：

```
void remove(Menu menu)
void remove(int idx)
```

- Menu——对要删除的菜单的引用，idx 是要删除的菜单的索引，索引从 0 开始。如果找到并删除了菜单项，则第一种形式返回 true，第二种形式返回对删除元素的引用。

2. Menu

Menu 封装了菜单，菜单项用 MenyItem 填充。而 Menu 派生自 MenuItem，这意味着一个 Menu 实例可以是另一个 Menu 实例中的选项，从而能够创建菜单的子菜单。Menu 定义了以下 4 个构造方法。

- Menu(String name)——该构造方法创建的菜单具有 name 指定的名称。
- Menu(String name, Node image)——image 指定了要显示的图片。
- Menu(String name, Node inage, MenuItem... menuItems)——允许指定最初的添加菜单项列表。
- Menu( )——可以用默认的构造方法创建未命名的菜单。然后，创建菜单后再调用 setText( )方法添加名称，调用 setGraphic( )方法添加图片。

每个菜单都维护一个由它包含的菜单项组成的列表。要在菜单中添加菜单项，需要把菜单添加到这个列表中。可以在 Menu 的构造方法中指定它们，或者把它们添加到列表中。为此，首先调用 getItem( )方法：

```
final ObservableList<MenuItem> getItems( )
```

该方法返回当前与菜单相关联的菜单项列表。然后，调用 add( )或 addAll( )方法把菜单项添加到这个列表中。另外，也可以调用 remove( )方法从中删除菜单项，调用 size( )方法获取列表的大小。此外，可以在菜单项列表中添加一条菜单分隔线，该分隔线是 SwparatorMenuItem 类型的对象。分隔线允许相关的菜单项分组，从而有助于组织菜单。分隔线可以帮助突出显示重要的菜单项。

3. MenuItem

MenuItem 封装了菜单中的元素，该元素可以是链接到某个程序动作的选项，也可以用于显示子菜单。MenuItem 定义了以下 3 种构造方法。

- MenuItem( )——创建一个空菜单项。
- MenuItem(String name)——用指定的名称创建菜单项。
- MenuItem(String name, Node image)——用包含的图片创建菜单项。

MenuItem 被选中时，将产生动作事件。通过调用 setOnAction( )方法可以为这种事件注册事件处理程序。

```
final void srtOnAction(EventHandler<ActionEvent> handle)
```

MenuItem 提供的 setDisable( )方法可以用来启用或禁用菜单项。

```
final void setDisable(boolean disable)
```

- 如果 disable 为 true，则禁用菜单项；如果 disable 为 false，则启用菜单项。

5.3　创建主菜单

一般地，主菜单是由菜单栏定义的菜单，也是定义了程序的全部功能的菜单。创建主菜单需要以下几个步骤：

① 创建用于保存菜单的 MenuBar 对象实例；

② 构造将包含在菜单栏中的每个菜单，首先创建一个 Menu 对象，然后向该对象添加 MneuItem；

③ 把菜单栏添加到场景图中；

④ 对于每个菜单项，添加动作事件处理程序，以响应选中菜单项时生成的动作事件。

【例 5.1】 创建一个菜单栏，其中包含 3 个菜单。第一个是标准的 File 菜单，它包含 Open、Close、Save 和 Exit 选项；第二个是 Options 菜单，它包含 Colors 和 Priority 两个子菜单；第三个菜单称为 Help，它只需要 About 一个选项。选中一个菜单项时，将在一个标签中显示所选项的名称。

```
1.  import javafx.application.*;
2.  import javafx.scene.*;
3.  import javafx.stage.*;
4.  import javafx.scene.layout.*;
5.  import javafx.scene.control.*;
6.  import javafx.event.*;
7.  import javafx.geometry.*;
8.  public class MenuDemo extends Application {
9.    Label response;
10.   public static void main(String[ ] args) {
11.     //通过调用 launch()方法启动 JavaFX 应用
12.     launch(args);
13.   }
14.   //重写 start()方法
15.   public void start(Stage myStage) {
16.     //设置舞台的标题
17.     myStage.setTitle("Demonstrate Menus");
18.     //定义根节点
19.     BorderPane rootNode = new BorderPane();
20.     //创建一个场景
21.     Scene myScene = new Scene(rootNode, 300, 300);
22.     //在舞台中设置场景
23.     myStage.setScene(myScene);
24.     //定义一个标签响应用户的选择
25.     response = new Label("Menu Demo");
26.     //创建 MenuBar 对象
27.     MenuBar mb = new MenuBar();
28.     //创建 File 菜单
29.     Menu fileMenu = new Menu("File");
30.     MenuItem open = new MenuItem("Open");
31.     MenuItem close = new MenuItem("Close");
32.     MenuItem save = new MenuItem("Save");
33.     MenuItem exit = new MenuItem("Exit");
```

```
34.     fileMenu.getItems().addAll(open, close, save, new
    SeparatorMenuItem(), exit);
35.     //将 File 菜单添加到 MenuBar 中
36.     mb.getMenus().add(fileMenu);
37.     //创建 Options 菜单
38.     Menu optionsMenu = new Menu("Options");
39.     //创建 Colors 子菜单
40.     Menu colorsMenu = new Menu("Colors");
41.     MenuItem red = new MenuItem("Red");
42.     MenuItem green = new MenuItem("Green");
43.     MenuItem blue = new MenuItem("Blue");
44.     colorsMenu.getItems().addAll(red, green, blue);
45.     optionsMenu.getItems().add(colorsMenu);
46.     //创建 Priority 子菜单
47.     Menu priorityMenu = new Menu("Priority");
48.     MenuItem high = new MenuItem("High");
49.     MenuItem low = new MenuItem("Low");
50.     priorityMenu.getItems().addAll(high, low);
51.     optionsMenu.getItems().add(priorityMenu);
52.     //添加分隔符
53.     optionsMenu.getItems().add(new SeparatorMenuItem());
54.     //创建 Reset 菜单项
55.     MenuItem reset = new MenuItem("Reset");
56.     optionsMenu.getItems().add(reset);
57.     //将 Options 菜单添加到 MenuBar
58.     mb.getMenus().add(optionsMenu);
59.     //创建 Help 菜单
60.     Menu helpMenu = new Menu("Help");
61.     MenuItem about = new MenuItem("About");
62.     helpMenu.getItems().add(about);
63.     //将 Help 菜单添加到 MenuBar 中
64.     mb.getMenus().add(helpMenu);
65.     //定义动作事件处理程序,以响应选中菜单项时生成的动作事件
66.      EventHandler<ActionEvent> MEHandler = new EventHandler<
    ActionEvent>() {
67.       public void handle(ActionEvent ae) {
68.         String name = ((MenuItem)ae.getTarget()).getText();
69.         //如果选择 Exit,则退出程序
```

```
70.          if(name.equals("Exit"))  Platform.exit();
71.            response.setText( name + " selected");
72.      }
73.   };
74.   //针对每个菜单项注册动作事件处理程序
75.   open.setOnAction(MEHandler);
76.   close.setOnAction(MEHandler);
77.   save.setOnAction(MEHandler);
78.   exit.setOnAction(MEHandler);
79.   red.setOnAction(MEHandler);
80.   green.setOnAction(MEHandler);
81.   blue.setOnAction(MEHandler);
82.   high.setOnAction(MEHandler);
83.   low.setOnAction(MEHandler);
84.   reset.setOnAction(MEHandler);
85.   about.setOnAction(MEHandler);
86.   //将 MenuBar 添加到窗口顶部
87.   //响应用户选择的标签显示在窗口中间
88.   rootNode.setTop(mb);
89.   rootNode.setCenter(response);
90.   //在窗口中显示舞台及其场景
91.   myStage.show();
92.   }
93. }
```

【执行结果】

执行结果如图 5.1 所示。

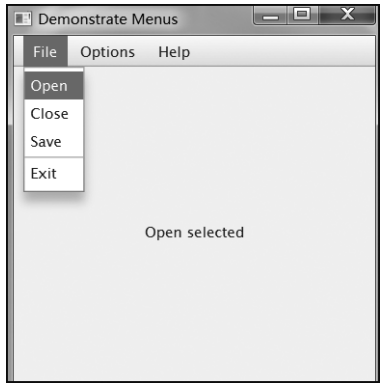

图 5.1　程序的执行结果

【分析讨论】

- 第 19 句创建的根节点的对象类型是 BorderPane，它定义了一个包含 5 个区域的窗口，这 5 个区域分别是顶部、底部、左侧、右侧和中央。
- 第 27 句用来构造菜单栏，此时，菜单栏是空的。第 29～33 句创建 File 菜单及其菜单项。第 34 句将各个菜单项添加到 File 菜单中。第 36 句将 File 菜单添加到菜单栏中。此时，菜单栏中将包含 File 菜单，File 菜单将包含 4 个选项：Open、Close、Save 和 Exit。
- 第 38 句将创建 Options 菜单，它包含 Colors 和 Oriority 两个子菜单，还包含 Reset 菜单项。第 40～44 句构造子菜单，第 45 句将它们添加到 Options 菜单中。第 47 句创建 Priority 子菜单，第 48～49 句创建两个菜单项 High 和 Low。第 50 句将它们添加到 Priority 子菜单中。第 51 句将 Priority 子菜单添加到 Options 菜单中。第 53 句在各个菜单项之间设置分隔符。第 55 句创建 Reset 菜单项。第 56 句将其添加到 Options 子菜单中。第 58 句将 Options 菜单添加到 MenuBar。
- 第 60 句创建 Help 菜单。第 61 句创建菜单项 About。第 62 句将其添加到 Help 菜单中。第 64 句将 Help 菜单添加到 MenuBar 中。
- 第 66～73 句定义动作事件处理程序，以响应选中菜单项时生成的动作事件。在 handle( )方法中，通过调用 getTarget( )方法获得事件的目标。该方法的返回类型是 MenuItem，其名称通过调用 getText( )方法返回。然后，这个字符串被赋值给 name。如果 name 包含字符串"Exit"，就调用 platform.exit( )方法终止程序，否则在 response 标签中显示获得的名称。
- 第 75～85 句将 MEHandler 注册为每个菜单项的动作事件处理程序。第 88 句将菜单栏添加到根节点。

5.4　效果与变换

1. 效果

效果由 javafx.scene.effect 包中的 Effect 类及其子类支持。使用效果可以自定义场景图中节点的外观，如表 5.2 所示。

表 5.2　JavaFX 内置的效果

类	主 要 功 能
Bloom	增加节点中较亮部分的亮度
BoxBlur	让节点变得模糊

续表

类	主 要 功 能
DropShadow	在节点后面显示阴影
Glow	生成发光效果
InnerShadow	在节点内显示阴影
Lighting	创建光源的阴影效果
Reflection	显示倒影

2. 变换

变换由 javafx.scene.transform 包中的抽象类 Transform 支持，它有 4 个子类，分别是 Rotate、Scale、Shear 和 Translate。在节点上可以执行多种变换，例如可以旋转并缩放节点。Note 类支持变换。

【例 5.2】 程序创建了 4 个按钮，分别为 Rotate、Scale、Glow 和 Shadow。每单击一个按钮，就对按钮应用对应的效果或变换。

```
1.  import javafx.application.*;
2.  import javafx.scene.*;
3.  import javafx.stage.*;
4.  import javafx.scene.layout.*;
5.  import javafx.scene.control.*;
6.  import javafx.event.*;
7.  import javafx.geometry.*;
8.  import javafx.scene.transform.*;
9.  import javafx.scene.effect.*;
10. import javafx.scene.paint.*;
11. public class EffectsAndTransformsDemo extends Application {
12.     double angle = 0.0;
13.     double glowVal = 0.0;
14.     boolean shadow = false;
15.     double scaleFactor = 1.0;
16.     //定义一个基本的效果
17.     Glow glow = new Glow(0.0);
18.     InnerShadow innerShadow = new InnerShadow(10.0, Color.RED);
19.     Rotate rotate = new Rotate();
20.     Scale scale = new Scale(scaleFactor, scaleFactor);
21.     //创建 4 个按钮
```

```
22.     Button btnRotate = new Button("Rotate");
23.     Button btnGlow = new Button("Glow");
24.     Button btnShadow = new Button("Shadow off");
25.     Button btnScale = new Button("Scale");
26.     public static void main(String[ ] args) {
27.       //通过调用 launch()方法启动 JavaFX 应用
28.       launch(args);
29.     }
30.     //重写 start()方法
31.     public void start(Stage myStage) {
32.       //设置舞台的标题
33.       myStage.setTitle("Effects and Transforms Demo");
34.       //使用 FlowPane 布局定义根节点,并指定场景中元素周围的水平和垂直
          //间隙
35.       FlowPane rootNode = new FlowPane(10, 10);
36.       //Center the controls in the scene.
37.       rootNode.setAlignment(Pos.CENTER);
38.       //创建一个场景
39.       Scene myScene = new Scene(rootNode, 300, 100);
40.       //在舞台中设置场景
41.       myStage.setScene(myScene);
42.       //设置发光效果
43.       btnGlow.setEffect(glow);
44.       //将 Rotate 按钮添加到变换列表中
45.       btnRotate.getTransforms().add(rotate);
46.       //将 Scale 按钮添加到变换列表中
47.       btnScale.getTransforms().add(scale);
48.       //处理 Rotate 按钮的动作响应事件
49.       btnRotate.setOnAction(new EventHandler<ActionEvent>() {
50.         public void handle(ActionEvent ae) {
51.           //每当按钮被单击,它将旋转 30°
52.           //指定旋转的中心点
53.           angle += 30.0;
54.           rotate.setAngle(angle);
55.           rotate.setPivotX(btnRotate.getWidth()/2);
56.           rotate.setPivotY(btnRotate.getHeight()/2);
57.     }
58.     });
```

```
59.     //定义 scale 按钮的动作响应处理程序
60.     btnScale.setOnAction(new EventHandler<ActionEvent>() {
61.       public void handle(ActionEvent ae) {
62.         //每当按钮被点击,它的大小将发生变换
63.         scaleFactor += 0.1;
64.         if(scaleFactor > 1.0)  scaleFactor = 0.4;
65.         scale.setX(scaleFactor);
66.         scale.setY(scaleFactor);
67.       }
68.     });
69.     //定义 Glow 按钮的动作事件响应程序
70.     btnGlow.setOnAction(new EventHandler<ActionEvent>() {
71.         public void handle(ActionEvent ae) {
72.         //每当按钮被单击,它的颜色将逐渐变浅
73.         glowVal += 0.1;
74.         if(glowVal > 1.0) glowVal = 0.0;
75.         //Set the new glow value.
76.         glow.setLevel(glowVal);
77.       }
78.     });
79.     //定义 Shadow 按钮的动作事件的相应处理程序
80.     btnShadow.setOnAction(new EventHandler<ActionEvent>() {
81.     public void handle(ActionEvent ae) {
82.       //每当按钮被单击,它的颜色将逐渐变深
83.       shadow = !shadow;
84.       if(shadow) {
85.         btnShadow.setEffect(innerShadow);
86.         btnShadow.setText("Shadow on");
87.       } else {
88.         btnShadow.setEffect(null);
89.         btnShadow.setText("Shadow off");
90.       }
91.     }
92.   });
93.   //将标签与按钮添加到场景图中
94.    rootNode.getChildren().addAll(btnRotate, btnScale, btnGlow,
btnShadow);
95.   //显示舞台和场景
```

```
96.   myStage.show();
97.   }
98. }
```

【执行结果】

执行结果如图 5.2 所示。

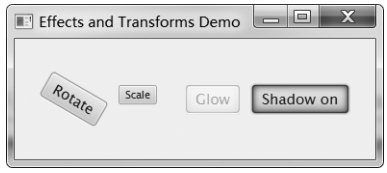

图 5.2　程序的执行结果

【分析讨论】

- 第 17～18 句定义了一个基本的效果。其中，Glow 生成的效果是节点具有发光的外观，构造方法为：

  Glow(double glowLevel)——glowLevel 用于指定光的亮度，取值范围在 0.0～1.0。创建 Glow 实例后，可以调用 setLevel( )方法改变发光的级别。

  final setLevel(double glowlevel)——glowLevel 用于指定光的亮度，取值范围在 0.0～1.0。

- InnerShadow 生成的效果是节点内具有阴影，其构造方法如下：

  InnerShadow(double radius, Color shadowColor)——radius 用于指定节点内阴影的半径，即指定阴影的大小。Shadow 用于指定阴影的颜色。Color 类型是 JavaFX 类型 javafx.scene.paint.Color，该类型定义了 Color.GREEN、ColorRED 和 Color.BLUE 等多个常量。

- 第 19～20 句定义了两个基本的变换。第 22～25 句定义了 4 个按钮。第 45 句将 Rotate 按钮添加到变换列表中。要向节点添加变换，可以把变换添加到节点维护的变换列表中。通过调用 Node 类定义的 getTransform( )方法可以获得该变换列表，如下所示：

  final ObservableList＜Transform＞ getTransform( )——该方法返回对变换列表的引用。要添加变换，只要调用 add( )方法把它添加到这个列表中即可。调用 clear( )方法可以清除该列表。调用 remove( )方法可以从列表中删除特定的元素。

- 为了演示变换，这里使用了 Rotale 类和 Scale 类。Rotale 绕着指定的点

旋转节点。Rotate 类的构造方法如下：

Rotate(double angle, double x, double y)——angle 用于指定旋转的角度。选中的中心点称为轴点，由 x 和 y 指定。

- 在创建 Rotate 实例之后(第 19 句)才设置这 3 个值。设置这 3 个值用到了如下 3 个方法(第 54～56 句)：

```
final void setAngle(double angle)
final void setPivotX(double x)
final void setPivotY(double y)
```

其中，angle 用于指定旋转的角度，x 和 y 用于指定旋转中心点。

- Scale 根据缩放因子缩放节点。Scale 类的构造方法如下：

Scale(double widthFactor, double heightFactor)——widthFactor 用于指定对节点宽度应用缩放因子，heightFactor 用于指定对节点高度应用缩放因子。

- 创建 Scale 实例之后(第 20 句)，可以使用如下两个方法改变这两个因子(第 63～66 句)：

```
final void setX(double widthFactor)
final void setY(double heightFactor)
```

widthFactor 用于指定对节点的宽度应用缩放因子，heightFactor 用于指定对节点的高度应用缩放因子。

5.5 小结

JavaFX 提供了一个强大的、流线化且灵活的框架，简化了视觉效果出色的 GUI 的开发。本章比较详细地分析与讨论了 JavaFX 为菜单提供的支持。JavaFX 的一个主要优势在于通过使用效果/变换改变控件的精确外观。通过这个功能可以让 GUI 具有用户期望的复杂的现代外观。本章详细介绍了开发 JavaFX 菜单应用程序，以及让 GUI 具有用户期望的外观的基本原理与方法。