Chapter 3 第3章 JavaFX 控件——Image、ImageView 与 TreeView

JavaFX 的控件中允许包含图片，还可以在场景中直接嵌入独立的图片。JavaFX 对图片支持的基础是 Image 和 ImageView 这两个类。Image 封装了图片，而 imageView 管理图片的显示。这两个类包含在 javafx.scene.image 包中。在 JavaFX 中，TreeView 以树状形式显示数据的分层视图。本章将介绍基于 NetBeans IDE 开发包含图片以及 TreeView 的 JavaFX 应用的方法。

3.1 Image 和 ImageView 控件

Image 类可以从 InputStream、URL 或图片文件的路径中加载图片。Image 类的构造方法为 Image(String url)。其中，url 指定 URL 或图片文件的路径；如果参数的格式不正确，则认为该参数指向一个路径，否则从 URL 位置加载图片。注意：Image 没有继承 Node，所以它不能作为场景图的一部分。ImageView 的构造方法为 ImageView(Image image)。

【例 3.1】 加载一幅清华大学出版社网站的图片，使用 ImageView 将该图片显示出来。

```
1.  package imagedemo;
2.  import javafx.application.Application;
3.  import javafx.scene.Scene;
4.  import javafx.scene.layout.StackPane;
5.  import javafx.stage.Stage;
6.  import javafx.scene.image.Image;
7.  import javafx.scene.image.ImageView;
8.  public class imageDemo extends Application {
9.      @Override
```

```
10.     public void start(Stage primaryStage) {
11.         //创建 Image 与 ImageView 对象
12.          Image image = new Image("http://www.tup.com.cn/upload/
kindeditor/image/20220505/lidh2022 0505134151_8151.jpg");
13.         ImageView imageView = new ImageView();
14.         imageView.setImage(image);
15.         //显示图片
16.         StackPane root = new StackPane();
17.         root.getChildren().add(imageView);
18.         Scene scene = new Scene(root, 300, 250);
19.         primaryStage.setTitle("Image Read Test");
20.         primaryStage.setScene(scene);
21.         primaryStage.show();
22.     }
23.     public static void main(String[] args) {
24.         launch(args);
25.     }
26. }
```

【执行结果】

执行结果如图 3.1 所示。

图 3.1　程序的执行结果

【分析讨论】

- 第 12 句创建了一个 Image。但是，图片是不能添加到场景中的，必须先嵌入一个 ImageView 中(第 13 句)。

3.2　TreeView 控件

在 JavaFX 中，TreeView 以树状形式显示数据的分层视图。这里，分层是指一些条目是其他条目的子项。例如，在树用于显示文件系统的情形下，单独的文件从属于包含它们的目录。在 TreeView 中，用户可以根据需要展开或收缩树枝，这样就可以以一种紧凑但可以展开的形式显示分层数据。TreeView 实现了一种概念上的基于树的数据结构。树从根节点开始，根节点指出树的起点。在根节点下有一个或多个子节点。子节点分为叶子节点(终端节点不包含子节点)和树枝节点(构成子树的根节点，子树是包含在更大的树结构中的树)。从根节点到某个特定节点的节点序列称为路径。当树的大小超出视图的尺寸时，TreeView 将会自动提供滚动条。根据需要自动添加滚动条能够节省空间。

TreeView 是泛型类，其声明如下。

```
class TreeView<T>
```

- T——指定树中条目保存值的类型。一般为 String 类型。

TreeView 的构造方法定义如下。

```
TreeView(TreeItem<T> rootNode)
```

- rootNode——子树的根节点。因为所有的节点都派生自根节点，所以根节点是唯一需要传递给 TreeView 的节点。
- TreeItem——构成树的条目是 TreeItem 类型的对象。TreeItem 没有继承 Node，所以 treeItem 对象不是通用对象，它可以用在 TreeView 中，但不能作为独立控件使用。

TreeItem 类的声明为 class TreeItem＜T＞

- T——指定了 TreeItem 保存值的类型。

使用 TreeView 的方法如下。

① 构造要显示的树。首先，创建根节点；然后，向根节点添加其他节点，可以通过对 getChildren( )方法返回的列表调用 add( )或 addAll( )方法实现，添加的节点可以是叶子节点或子树。

② 构造完成树以后，将其根节点传递给 TreeView 的构造方法以创建 TreeView 对象。

③ 处理 TreeView 中选择的事件。首先，调用 getSelectinModel( )方法获

得选择模式，然后调用 selectItemProperty( )方法获得选中的属性，最后，通过对该方法的返回值调用 addListener( )方法以添加事件监听器。每次做出选择时，就将对新选项的引用作为新值传递给 changed( )方法。

④ 通过调用 getValue( )方法可以获得 treeItem 的值。还可以前向或后向沿着某个条目的树路径前进。

⑤ 通过调用 getParent( )方法可以得到某个父节点。调用 getChildren( )方法可以得到某个节点的子节点。

【例 3.2】 创建一个树，显示一个食物层次。树中存储 String 类型的条目。根节点的标签是 Food。根节点有 3 个直接子节点——水果、蔬菜和坚果。水果节点包含 3 个子节点——苹果、梨和橘子。苹果节点下有 3 个叶子节点——富士、国光和红玉。每次做出选择时都显示所选项的名称。

```
1.  package treeviewdemo;
2.  import javafx.application.*;
3.  import javafx.scene.*;
4.  import javafx.stage.*;
5.  import javafx.scene.layout.*;
6.  import javafx.scene.control.*;
7.  import javafx.event.*;
8.  import javafx.beans.value.*;
9.  import javafx.geometry.*;
10. public class TreeViewDemo extends Application {
11. Label response;
12. public static void main(String[ ] args) {
13.     //通过调用 launch()方法启动 JavaFX 应用
14.     launch(args);
15. }
16. //重写 start()方法
17. public void start(Stage myStage) {
18.     //设置舞台的标题
19.     myStage.setTitle("Demonstrate a TreeView");
20.     //使用 FlowPane 布局
21.     //指定场景中元素周围的水平和垂直间隙
22.     FlowPane rootNode = new FlowPane(10, 10);
23.     //中心对齐
24.     rootNode.setAlignment(Pos.CENTER);
25.     //创建一个场景
```

```
26.     Scene myScene = new Scene(rootNode, 310, 460);
27.     //在舞台中设置场景
28.     myStage.setScene(myScene);
29.     //创建一个标签提示用户的选择项
30.     response = new Label("No Selection");
31.     //创建树的根节点
32.     TreeItem<String> tiRoot = new TreeItem<String>("Food");
33.     //创建水果子节点
34.     TreeItem<String> tiFruit = new TreeItem<String>("水果");
35.     //构造苹果子树
36.     TreeItem<String> tiApples = new TreeItem<String>("苹果");
37.     //将不同品种的苹果添加到苹果子树节点
38.     tiApples.getChildren().add(new TreeItem<String>("富士"));
39.     tiApples.getChildren().add(new TreeItem<String>("国光"));
40.     tiApples.getChildren().add(new TreeItem<String>("红玉"));
41.     //将不同的水果添加到水果子树节点 Add varieties to the fruit node
42.     tiFruit.getChildren().add(tiApples);
43.     tiFruit.getChildren().add(new TreeItem<String>("梨"));
44.     tiFruit.getChildren().add(new TreeItem<String>("橘子"));
45.     //最后,将水果子节点添加到根节点
46.     tiRoot.getChildren().add(tiFruit);
47.     //现在,用同样的方法构造蔬菜子树
48.     TreeItem<String> tiVegetables = new TreeItem<String>("蔬菜");
49.     tiVegetables.getChildren().add(new TreeItem<String>("玉米"));
50.     tiVegetables.getChildren().add(new TreeItem<String>("豌豆"));
51.     tiVegetables.getChildren().add(new TreeItem<String>("西兰花"));
52.     tiVegetables.getChildren().add(new TreeItem<String>("豆颈"));
53.     tiRoot.getChildren().add(tiVegetables);
54.     //构造坚果子树节点
55.     TreeItem<String> tiNuts = new TreeItem<String>("坚果");
56.     tiNuts.getChildren().add(new TreeItem<String>("核头"));
57.     tiNuts.getChildren().add(new TreeItem<String>("花生"));
58.     tiNuts.getChildren().add(new TreeItem<String>("山核头"));
59.     tiRoot.getChildren().add(tiNuts);
60.     //用创建的树创建 TreeView
61.     TreeView<String> tvFood = new TreeView<String>(tiRoot);
62.     //设置 TreeView 的选择模式
63.      MultipleSelectionModel < TreeItem < String > > tvSelModel =
    tvFood.getSelectionModel();
```

```
64.     //用变化监听器响应用户选择的一条 TreeView
65.     tvSelModel.selectedItemProperty().addListener(new
    ChangeListener<TreeItem<String>>() {
66.     public void changed(ObservableValue<? extends TreeItem<
    String>> changed,
67.                         TreeItem<String> oldVal, TreeItem<
    String> newVal) {
68.         //显示用户的选择以及子树路径
69.         if(newVal != null) {
70.           //构造入口路径与选择的条目
71.           String path = newVal.getValue();
72.           TreeItem<String> tmp = newVal.getParent();
73.           while(tmp != null) {
74.             path = tmp.getValue() + " -> " + path;
75.             tmp = tmp.getParent();
76.           }
77.           //显示用户选择的条目以及路径
78.            response.setText("Selection is " + newVal.getValue()
    + "\nComplete path is " + path);
79.       }
80.     }});
81.       //将树根节点添加到场景中
82.       rootNode.getChildren().addAll(tvFood, response);
83.       //在舞台中显示场景
84.       myStage.show();
85.     }
86. }
```

【执行结果】

执行结果如图 3.2 所示。

【分析讨论】

- 第 32 句创建了树的根节点;其次,创建了根节点之下的节点,这些节点构成了子树的根节点。一个表示水果(第 34 句),一个表示蔬菜(第 48 句),一个表示坚果(第 55 句)。然后,为这些子树添加叶子节点。其中,水果子树还包含一个子树,它包含不同品牌的苹果(第 38～40 句)。这里关键的知识点是树中的每个树枝要么走向一个叶子节点,要么走向一个子树的根节点。
- 构造所有的节点之后,通过对根节点调用 add( )方法,就可以将每个子

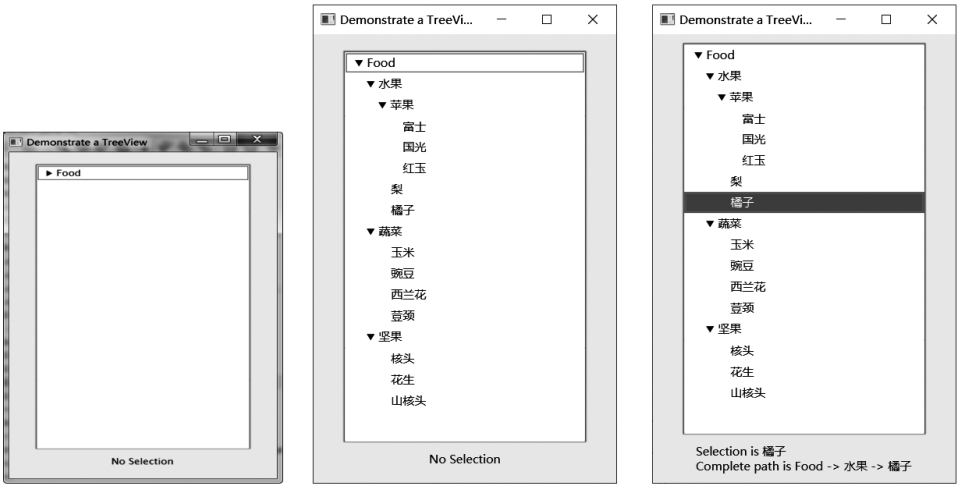

图 3.2　程序的执行结果

树的根节点添加到树的根节点(第 38～40 句,第 42～44 句,第 49～53 句,第 56～59 句)。

- 在事件处理监听程序中,从根节点到选定节点的路径通过第 71～75 句实现。首先,获取选中节点的值(一个字符串,即节点的名称)。然后,创建一个 TreeItem<String>类型的变量,并将其初始化为引用新选中节点的父节点。如果新选中的节点没有父节点,那么其值为 NULL。否则,进入循环,将每个父节点的值添加到 path 中。这个过程不断循环进行,直到找到树的根节点。

3.3　小结

JavaFX 对图片支持的基础是 Image 和 ImageView 这两个类。Image 封装了图片,而 imageView 则负责管理图片的显示。在 JavaFX 中,TreeView 以树状形式显示数据的分层视图。在 TreeView 中,用户可以根据需要展开或收缩树枝,这样就可以用一种紧凑但可以展开的形式显示分层数据。TreeView 实现了一种概念上的基于树的数据结构。本章介绍了基于 NetBeans IDE 开发包含图片以及 TreeView 的 JavaFX 应用程序的方法。希望读者认真领会理解,通过实际的上机调试掌握这些示例程序的编程方法。