第14章 Java FX基础

目 录

contents

14.1 AWT、Swing和Java Fx

- AWT是抽象窗口工具包
- 重UI组件
- 和平台相关：

- Java FX可开发富因特网应用(RIA)
- 能在桌面应用和浏览器中独立运行
- 支持触摸设备

AWT

抽象窗口工具集
(Abstract Window Toolkit)

Swing

更健壮、功能更齐全和更灵活

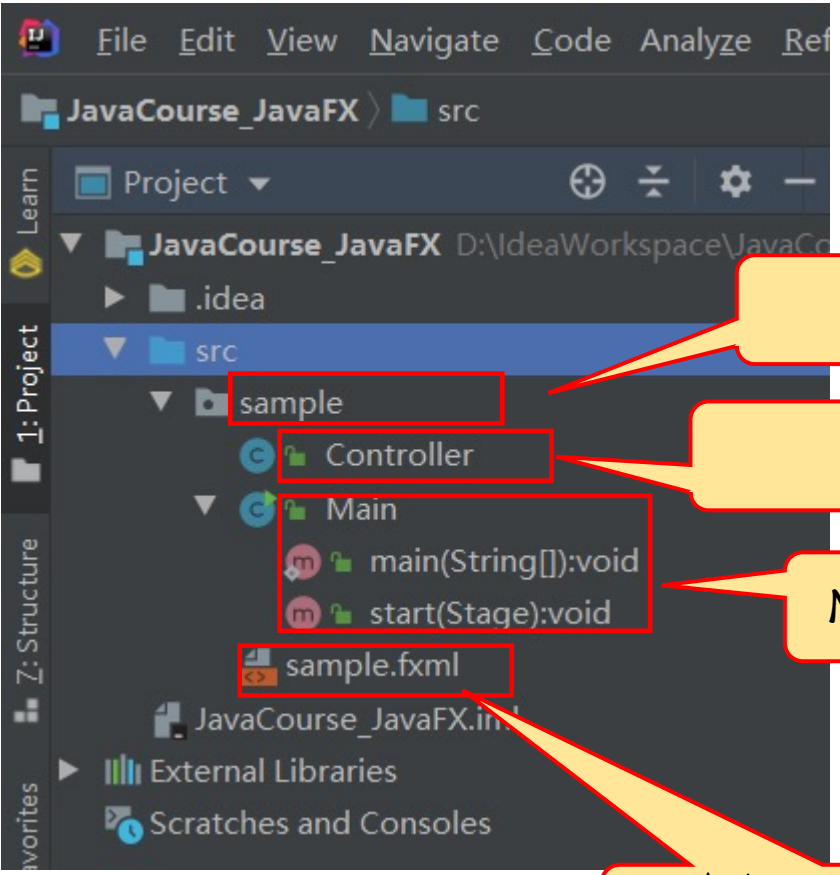
- Swings是一个轻组件,面向桌面开发
- 使用Java代码在画布上直接绘制
- 不依赖平台、占用较少的本地资源

Java Fx

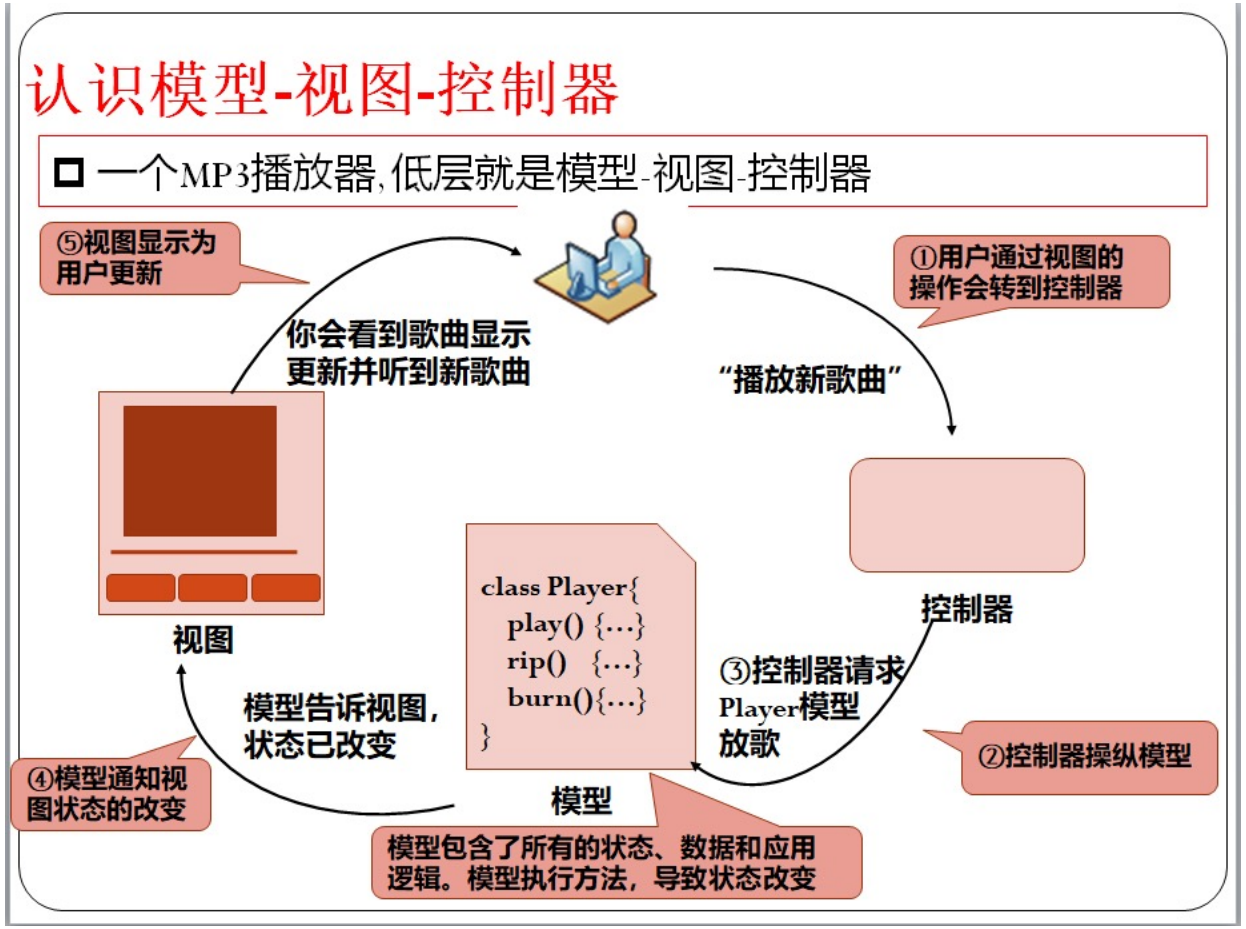
桌面应用
Web应用
统一开发模式

14.2 JavaFX 程序的基本结构

创建工程时工程类型要选择“Java FX”



Java FX工程结构图如左所示
Java FX采用MVC（Model-View-Control）设计模式：
降低了界面展现、应用逻辑、用户对视图的操作控制三部分的耦合度。



14.2 JavaFX 程序的基本结构


```
// Main.java里加载sample.fxml
FXMLLoader loader = new FXMLLoader(getClass().getResource( “sample.fxml"));
Pane root = loader.load();
```

视图的界面布局由xml来描述，保存在.fxml文件里

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.control.TextField?>
```

<Pane>、<Button>、<TextField>这些标签都对应了Java FX的UI组件，标签名为对应的类名

最外面节点是容器Pane

```
<Pane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller= “sample.Controller">
  <children>
    <Button fx:id="btn1" layoutX="499.0" layoutY="339.0" mnemonicParsing="false" onAction="#onClick" text="Button" />
    <TextField fx:id="text" layoutX="35.0" layoutY="35.0" />
  </children>
</Pane>
```

根节点属性指定视图对应的控制器，注意要加包名限定

如果需要在Java程序里访问控件，需要在xml里指定控件的fx:id属性

onAction属性指定了用户点击按钮时，会调用控制器里的onClick方法

14.2 JavaFX 程序的基本结构

```
public class Main extends Application {
        @Override
        public void start(Stage primaryStage) {
                try {
                        FXMLLoader loader = new FXMLLoader(getClass().getResource( “sample.fxml"));
                        Pane root = loader.load();

                        Controller controller = loader.getController();
                        controller.setMyApp(this);

                        Scene scene = new Scene(root,400,600);
                        primaryStage.setScene(scene);
                        primaryStage.show();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }
        public static void main(String[] args) {
                launch(args);
        }
}
```

Main必须继承Application，并覆盖start方法

加载sample.fxml

加载视图的根节点（fxml里根节点），自动加载根节点下面所有的UI控件

得到控制器实例

将Main对象传进控制器，这样控制器就可以访问Main对象的成员

Scene构造函数第一个参数指定了场景里的根节点对象root（前面已加载）

Stage就是舞台（窗体），可以随时切换场景(Scene)

14.2 JavaFX 程序的基本结构

```
public class Controller {
        @FXML
        private TextField text;

        private Main myApp;

        public void setMyApp(Main myApp) {
                this.myApp = myApp;
        }

        @FXML
        public void onClick(ActionEvent e) {
                text.setText("Button Cliked");
        }
}
```

@FXML标注将实例变量名与xml中fx:id属性值相同的控件绑定起来，在Java程序里通过text引用，可以控制TextField对象（读写），比如onClick方法里的text.setText方法

Main对象引用，引用Main的实例，这样控制器就可以访问Main对象的成员

我们讨论了Controller（控制器）、fxml配置文件（视图）。那么Model呢？Main不要作为模型类，应该在工程里添加Model类。Main只是JavaFX应用的启动类。控制器如何拿到Model的对象？定义一个类似于setMyApp的方法，例如：
public void setModel(Model model)。
然后在Main里实例化Model对象，通过拿到的Controller实例调用setModel方法。

@FXML标注将onClick函数与xml中定义的onAction="#onClick"绑定

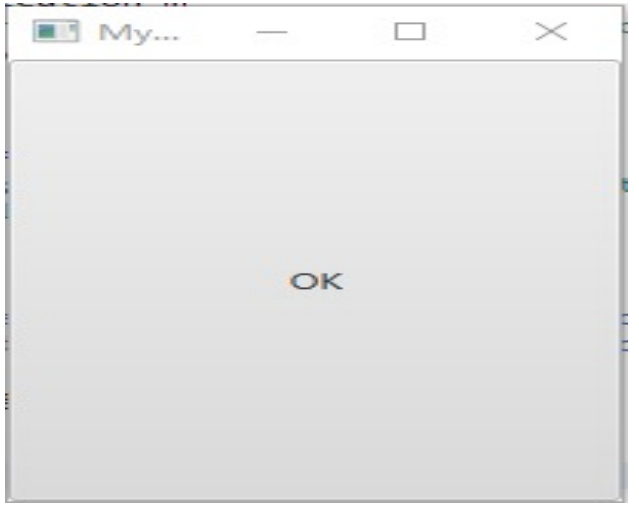
```
<Pane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="sample.Controller">
  <children>
    <Button fx:id="btn1" layoutX="499.0" layoutY="339.0" mnemonicParsing="false" onAction="#onClick" text="Button" />
    <TextField fx:id="text" layoutX="35.0" layoutY="35.0" />
  </children>
</Pane>
```

14.2 JavaFX 程序的基本结构

```
import javafx.application.Application ;
import javafx.scene.Scene;
import javafx.scene.control .Button;
import javafx.stage.Stage;
public class MyJavaFX extends Application {
  public void start(Stage primaryStage) {
    Button btOK = new Button("OK") ;
    Scene scene = new Scene(btOK, 200, 250) ;
    primaryStage.setTitle("MyJavaFX") ;
    primaryStage.setScene(scene) ;
    primaryStage.show() ;
  }//注意按钮占满了了整个场景
  public static void main(String[] args) {
            Application.launch(args);
  }
}
```

需从Application类派生
重写start方法：用于启动主窗体(Stage类型)。

如果想启动两个窗体（Stage），
可在start中创建新的Stage对象。

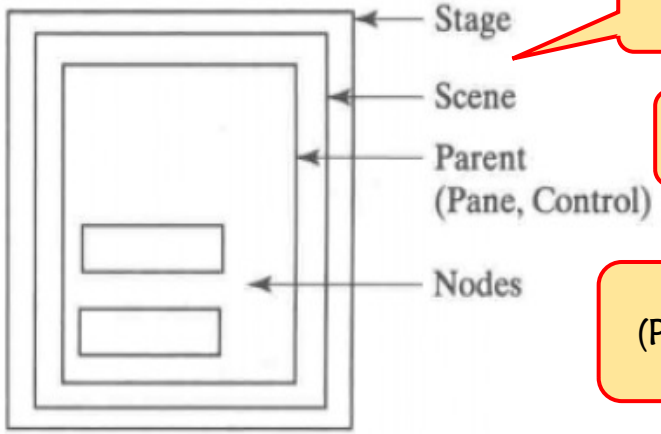


Stage就像一个舞台，
由场景Scene组成。
场景可以切换（Stage的setScene方法）

launch 方法是一个定义在Application 类中的静态方法，用于启动一个独立的JavaFX 应用：激活start方法。

注意也可以完全脱离fxml，完全在Java程序里定义视图的内容，这个例子就是如此。但不推荐这样做，用fxml定义视图，可以将代码与视图定义分离。有的同学担心fxml写起来太麻烦，不用担心。利用UI设计工具SceneBuilder，用拖拽控件的方式设计好UI后，会自动产生fxml配置文件。

14.3 面板、UI组件以及形状



界面元素的包含关系

Stage就是窗口
里面包含Scene

Stage包含多个Scene

Scene包含多个Parent
(Pane或Control)，其中一
个是root

Node是所有
可视化组件的祖先类

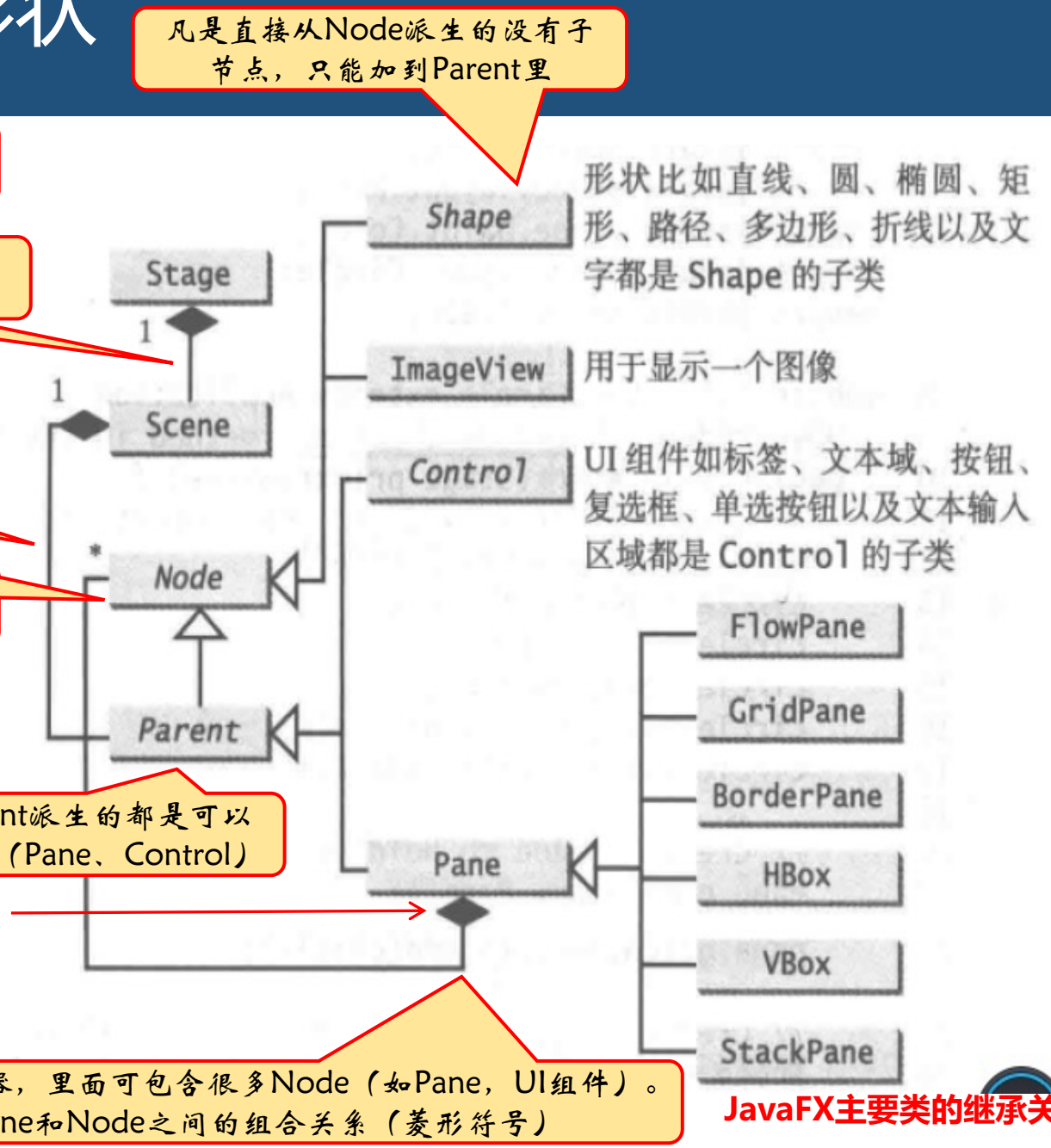
面板作为容器用于布置若干节点

注意：

(1)派生：Node->Parent->Pane，
故Pane可包含Node的任何子类。

(2)Pane由Node构成->故可由Pane构成，
故Pane可包含多个Pane。

(3)Scene可包含Control或Pane，但不能
包含Shape和ImageView。

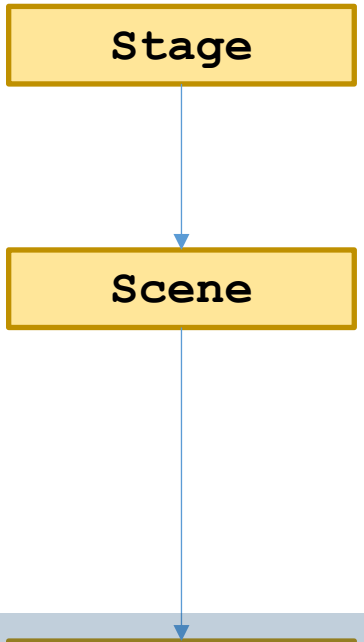


JavaFX主要类的继承关系

14.3 面板、UI组件以及形状

JavaFX API： https://docs.oracle.com/javafx/2/api/index.html?overview-summary.html

Stage、Scene、Scene里面的可视化控件之间的包含关系形成了树状关系



③primaryStage.setScene(scene);
设置Stage的第一个Scene

②Scene scene = new Scene(root,400,600);
把Pane开始的子树放进去

①Pane root = loader.load();从fxml加载所有的UI组件，root就是fxml文件里的根节点Pane，TextField和Button在fxml里是Pane的子节点。因此这条语句构建了下面的子树。

注意Pane作为容器，负责里面UI组件的布局(Layout). Pane可以相互嵌套，实现任意复杂的布局。

如果需要在Pane里加入更多UI组件
在fxml文件的Pane节点里添加子节点

这个子树里每个节点的祖先类是Node

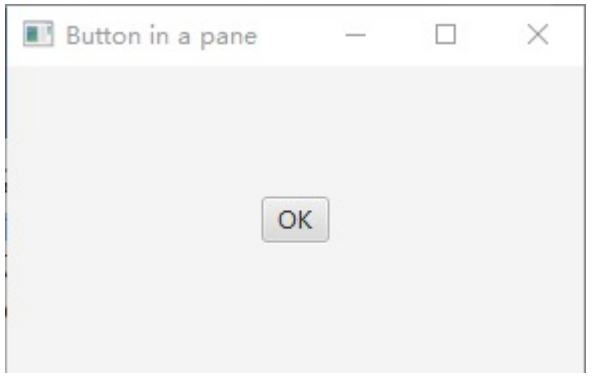
14.3 面板、UI组件以及形状

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class ButtonInPane extends Application
{
 public void start(Stage primaryStage){
 StackPane pane = new StackPane();
 pane.getChildren().add(new Button("OK"));
 Scene scene = new Scene(pane, 200,50);
 primaryStage.setTitle("Button in a pane");
 primaryStage.setScene(scene);
 primaryStage.show(); // Display the stage
}
```

StackPane：Pane的子类，它实现的布局方式是：控件在同一中央位置堆叠存放，一个控件可能覆盖其它控件。

程序将Button控件加入StackPane。getChildren返回javafx.collections.ObservableList 的一个实例。

ObservableList类似于ArrayList, 用于存储元素集合。调用add(e)可将一个元素加入集合的列表。

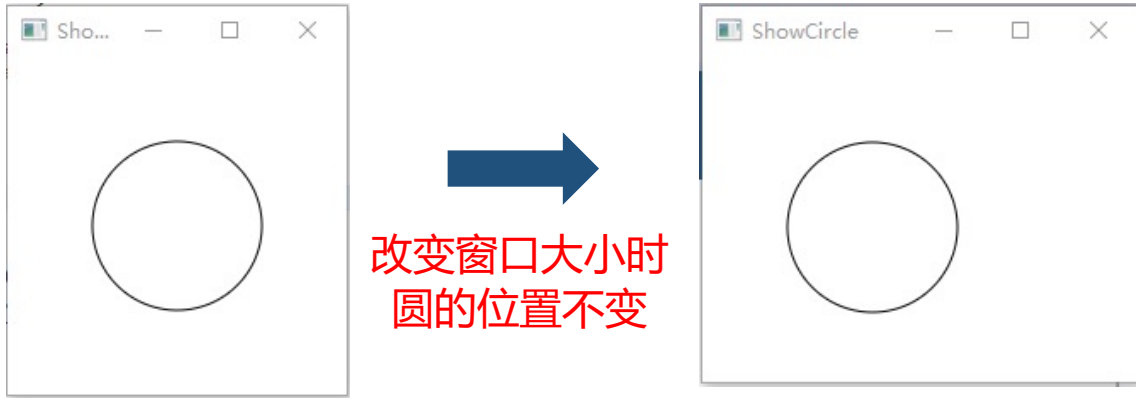


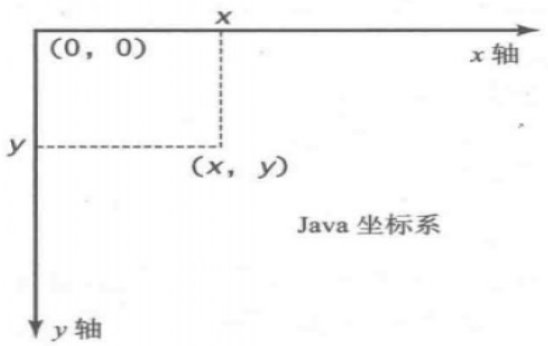
pane.getChildren().add(构造好的UI组件对象);是将一个UI组件加入面板的方法。

这个代码不需要fxml文件可以运行。因此，我们可以不依赖于fxml，直接在程序里创建UI并布局（当然比较麻烦）

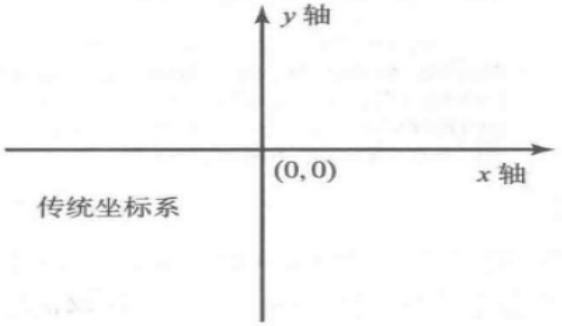
14.3 面板、UI组件以及形状

```
import javafx.application .Application ;
import javafx.scene.Scene ;
import javafx.scene.layout .Pane ;
import javafx.scene.paint .Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage ;
public class ShowCircle extends Application {
 public void start (Stage primaryStage) {
 Circle circle = new Circle(100,100,50);
 circle.setStroke(Color.BLACK) ;
 circle.setFill (Color .WHITE) ;
 Pane pane = new Pane() ;
 pane.getChildren0.add(circle) ;
 primaryStage.setTitle("ShowCircle");
 primaryStage.setScene(new Scene(pane , 200, 200));
 primaryStage.show() ; }}
```





Java坐标系



传统坐标系

如果希望窗体改变大小时，圆依然在中心，咋办？属性绑定

```
circle.centerXProperty( ).bind(pane.widthProperty( ).divide(2)); // widthProperty被绑定
circle.centerYProperty( ).bind(pane.heightProperty( ).divide(2)); // heightProperty被绑定
```

14.4 属性绑定

- **属性绑定**：可以将一个目标对象属性和一个源对象属性绑定。（观察者模式实现）

```
circle.centerXProperty( ).bind(pane.widthProperty( ).divide(2));
```

目标对象属性　　源对象属性

- 如果源对象中（被观察者）的值改变了，目标对象（观察者）也将自动改变。
- 被绑定目标采用bind方法和源进行绑定：target.bind(source);

bind方法在javafx.beans.property.Property接口中定义。

目标target是javafx.beans.property.Property接口的一个实例（实现了Property接口的类的实例为观察者）。

源Source是javafx.beans.value.ObservableValue接口的一个实例（ 实现了ObservableValue接口的类的实例为被观察者）。

- JavaFX已为基本类型和字符串定义了绑定属性。对于double/float/long/int/boolean类型的值，对应的绑定属性类型是DoubleProperty/FloatProperty/LongProperty/IntegerProperty/BooleanProperty, String类型值的绑定属性是StringProperty。这些属性同时也实现了ObservableValue接口，因此也可以作为绑定的Source。
  - 即这些属性即可以作为观察者绑定其他源对象属性，也可以作为被观察者（绑定的Source）

14.4 属性绑定

```
import javafx.application .Application ;
import javafx.scene.Scene ;
import javafx.scene.layout .Pane ;
import javafx.scene.paint .Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage ;
public class ShowCircle extends Application {
public void start (Stage primaryStage) {
Circle circle = new Circle(100,100,50);
circle.setStroke(Color.BLACK) ;
circle.setFill (Color .WHITE) ;
Pane pane = new Pane() ;
circle.centerXProperty().bind(pane.widthProperty().divide(2));
circle.centerYProperty().bind(pane.heightProperty().divide(2));
pane.getChildren0.add(circle) ;
primaryStage.setTitle("ShowCircle");
primaryStage.setScene(new Scene(pane , 200, 200));
primaryStage.show() ; }}
```

14.4 属性绑定

- 一般而言，JavaFX的每个绑定属性都有一个get方法和set方法（用于获取和设置属性值本身），还有一个获取绑定属性的方法。
- 绑定属性获取方法的命名习惯是属性名称后面加上Property（如centerXProperty方法）
- 例如Circle有个绑定属性centerX，有getCenterX，setCenterX方法获取和设置属性值本身，centerXProperty为绑定属性获取方法，返回的是DoubleProperty，要绑定属性，必须用该方法

```
public class SomeClassName {

  private PropertyType x;

  /** Value getter method */
  public propertyValueType getX() { ... }

  /** Value setter method */
  public void setX(propertyValueType value) { ... }

  /** Property getter method */
  public PropertyType
    xProperty() { ... }
}
```

a) x 是一个绑定属性

```
public class Circle {

  private DoubleProperty centerX;

  /** Value getter method */
  public double getCenterX() { ... }

  /** Value setter method */
  public void setCenterX(double value) { ... }

  /** Property getter method */
  public DoubleProperty centerXProperty() { ... }
}
```

b) centerX 是一个绑定属性

注意double绑定属性的类型：DoubleProperty
绑定属性类型必须是Property子类型
DoubleProperty是包装类型（包装了double）
注意和普通get和set方法的返回类型、形参类型的区别

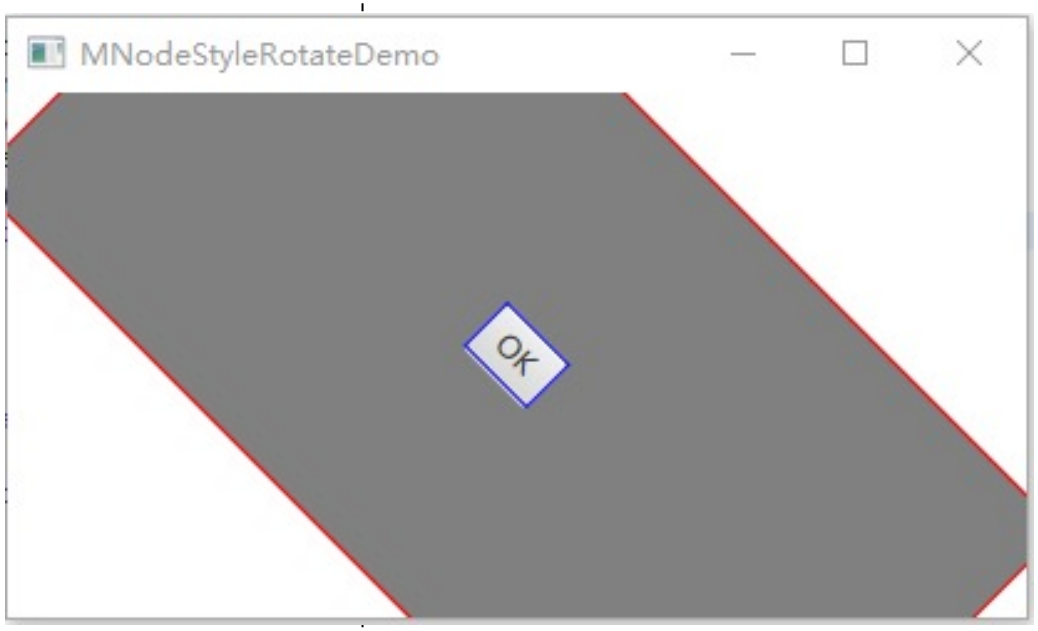
14.5 节点（Node）的通用属性和方法

- JavaFX Node类（所有UI组件的父类）的样式属性设置方法和在Web页面中指定HTML元素样式一样，可通过层叠样式表 CSS来设置（当然也可在Java程序里设置）。 因此，JavaFX 的样式表称为JavaFX CSS。样式决定了UI的外观。（比如UI组件大小，位置，边框颜色…）
- JavaFX 中，样式属性名使用前缀-fx-进行定义。每个子类(Node的具体子类，代表不同的UI元素）拥有它自己的一些特定的样式属性（公共的样式属性在Node或其他父类里定义）。
- 设定样式的语法是styleName:value。节点的多个样式属性可以一起设置，通过分号进行分隔。设置样式属性的方法是setStyle。如circle.setStyle("-fx-stroke: black;-fx-fill: red;");

  等价于下面两个语句：circle.setStroke(Color.BLACK)；circle.setFill(Color.RED);
- 如果使用了不正确的JavaFX CSS, 程序依然可以编译和运行，但是样式将被忽略。
- rotate样式属性设定一个以度为单位的角度，让节点围绕它的中心旋转该角度。如果设置的角度是正的，表示旋转是顺时针；否则，逆时针。

14.5 节点的通用属性和方法

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage ;
import javafx.scene.layout.StackPane;
public class NodeStyleRotateDemo extends Application {
public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    Button btOK = new Button("OK");
    btOK.setStyle("-fx-border-color: blue;");
    pane.getChildren( ).add(btOK);    pane.setRotate(45);
    pane.setStyle( "-fx-border-color: red; -fx-background-color: gray;");
    primaryStage.setTitle("MNodeStyleRotateDemo");
    primaryStage.setScene(new Scene(pane, 200, 250));
    primaryStage.show();
 }
```

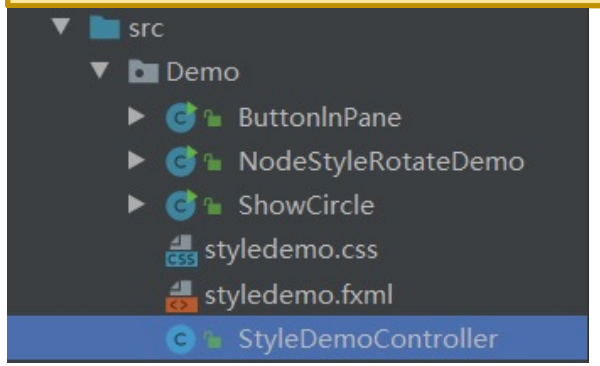


14.5 Java FX CSS Reference ： https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

- 更好的方式是在样式表里设置UI组件的属性，这样可以代码与样式分离。同时组件的创建也应该在fxml里完成（如同本章第一个例子）。这个旋转的例子用fxml和样式的实现如下

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<StackPane xmlns="http://javafx.com/javafx"
        xmlns:fx="http://javafx.com/fxml"
        fx:controller="Demo.StyleDemoController"
        stylesheets="/Demo/styledemo.css"
        prefHeight="400.0" prefWidth="600.0" fx:id="stackPane">
   <children>
      <Button fx:id="btnOK"  text="OK" />
   </children>
</StackPane>
```

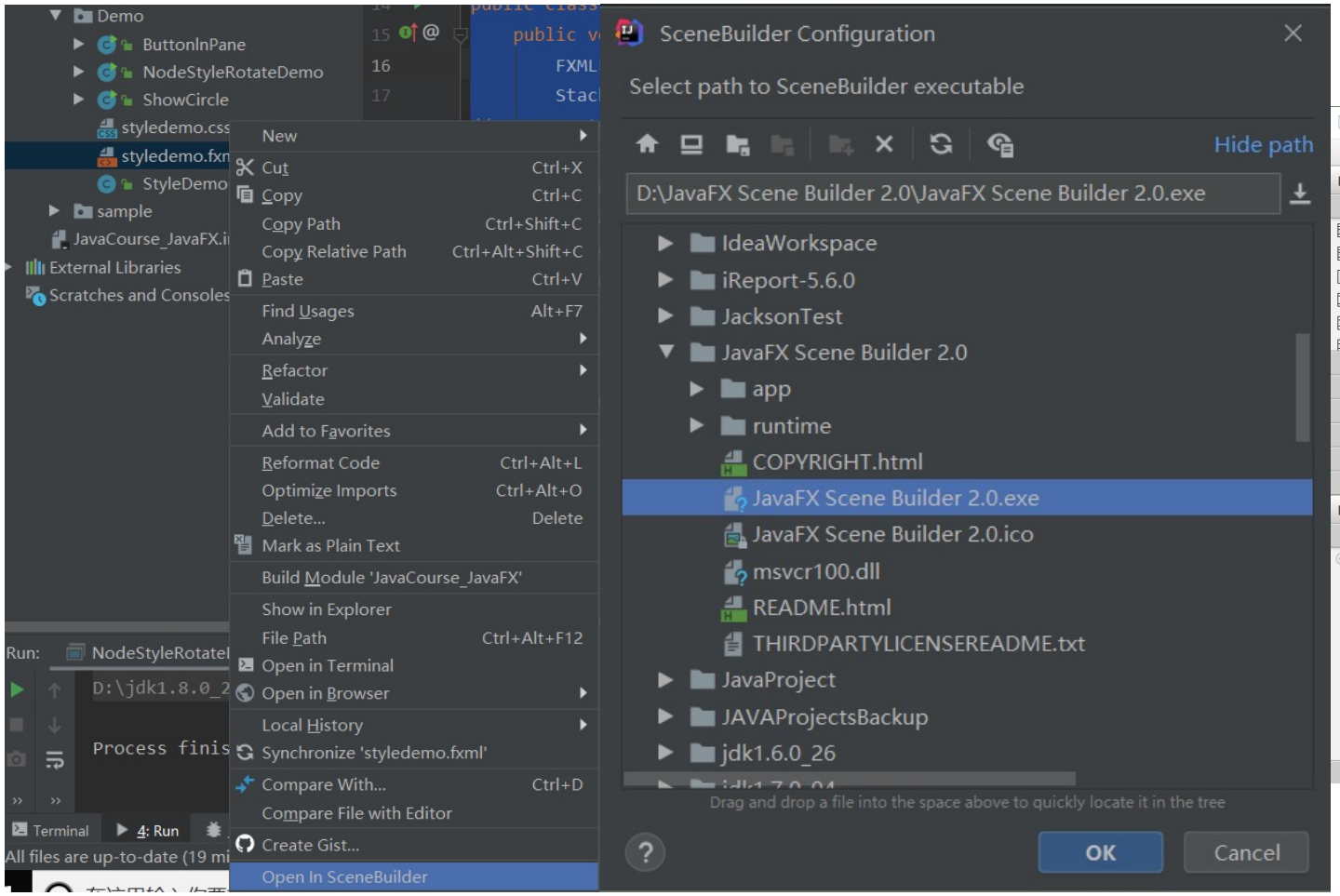
```
/*#id是id选择器，根据fxml定义的节点id选择组件应用样式*/
#btnOK{
    -fx-border-color: blue;
}
#stackPane{
    -fx-border-color: red;
    -fx-background-color: gray;
    -fx-rotate: -45;
}
```

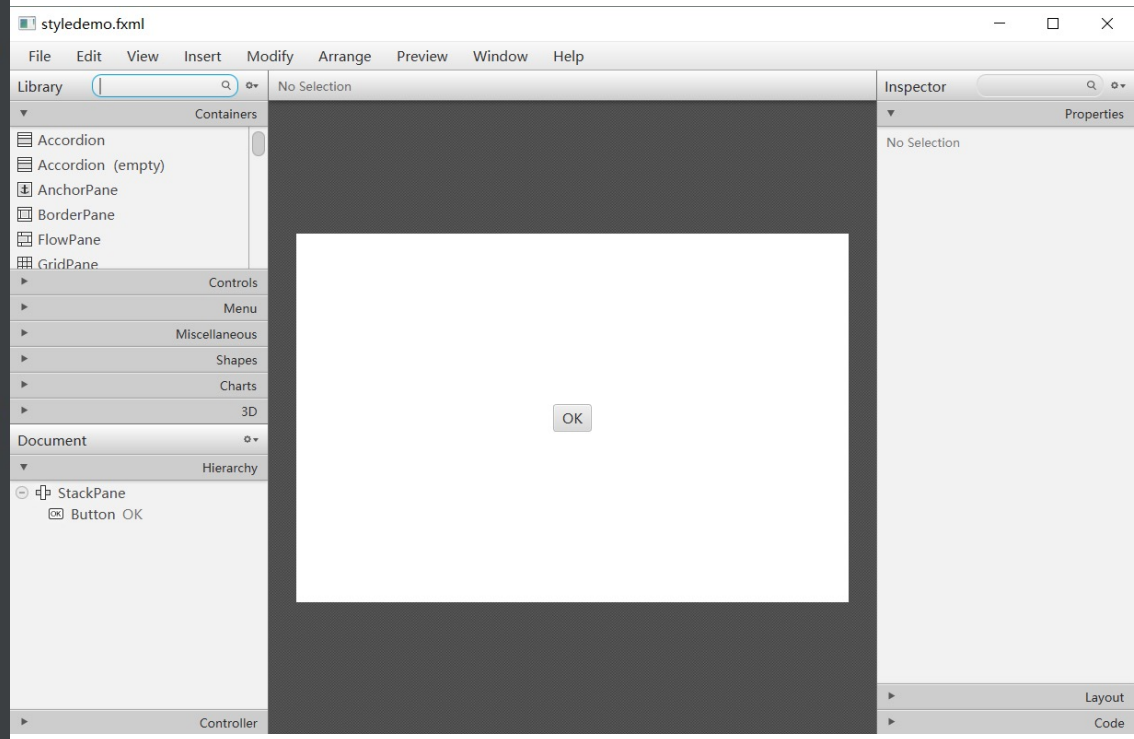



```
public class NodeStyleRotateDemo extends Application{
    public void start(Stage primaryStage) throws IOException {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("styledemo.fxml"));
        StackPane pane = loader.load(); //类名与fxml里根节点名一致
        Scene scene = new Scene(pane,400,600);
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(scene);primaryStage.show();
    }
}
```

14.5 节点的通用属性和方法

- 设计JavaFX UI界面可以使用设计工具SceneBuilder：https://gluonhq.com/products/scene-builder/





14.6 布局面板

类	描述
Pane	布局面板的基类，它有 getChildren（ ）方法来返回面板中的节点列表
StackPane	节点放置在面板中央，并且叠加在其他节点之上
FlowPane	节点以水平方式一行一行放置，或者垂直方式 一列一列放
GridPane	节点放置在一个二维网格的单元格中
Border Pane	将节点放置在顶部、右边、底部、左边以及中间区域
HBox	节点放在单行中
VBox	节点放在单列 中

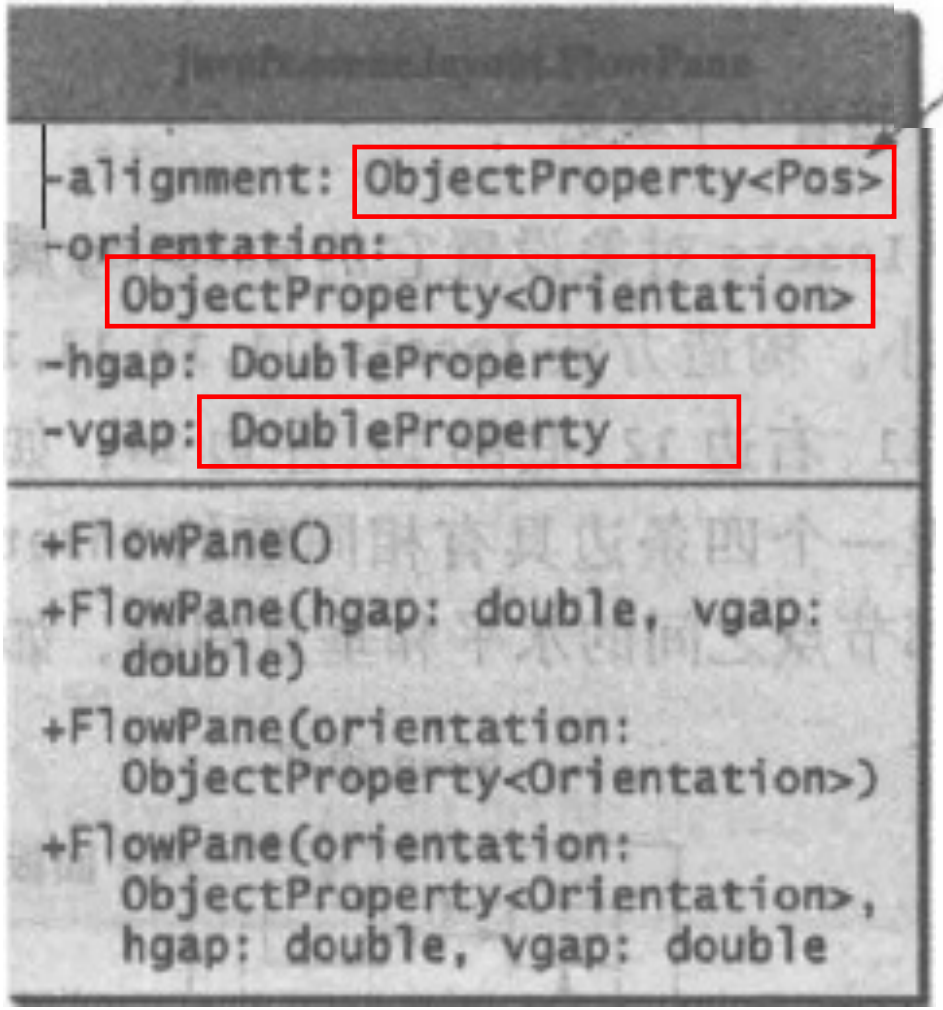
14.6 布局面板

FlowPane

- FlowPane 将节点按照加入的次序，从左到右水平或者从上到下垂直组织。
- 当一行或者一列排满的时候，开始新的一行或者一列。
- 除了继承的可被绑定属性如widthProperty等外，下列数据成员还有自定义的可被绑定属性：alignment、orientation、hgap、vgap
- Orientation.HORIZONTAL:节点水平排列
- Orientation.VERTICAL：节点垂直排列
- 以使用像素为单位指定节点之间的距离。

14.6 布局面板

FlowPane



在类中提供了属性值的获取和设置方法，以及属性本身的获取方法，为简洁起见，故在 UML 图中省略

该面板内容的整体对齐方式（默认：Pos.LEFT）
面板中的方向（默认：Orientation.HORIZONTAL）

节点之间的水平间隔（默认：0）
节点之间的垂直间隔（默认：0）

创建一个默认的 FlowPane
使用给定的水平和垂直间隔创建一个 FlowPane

使用给定的方向创建一个 FlowPane

使用给定的方向、水平间隔以及垂直间隔创建一个 FlowPane

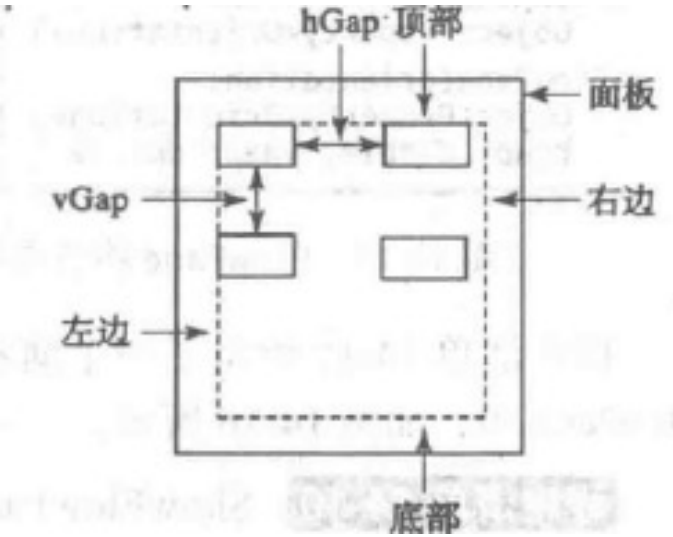
14.6 布局面板

```
public class ShowFlowPane extends Application{
   public void start(Stage primaryStage) {
      FlowPane pane = new FlowPane() ;
      pane.setPadding(new Insets(11, 12, 13, 14));
      pane.setHgap(5);   pane.setVgap(5);
      pane.getChildren().addAll(new Label("First Name:"),
                  new TextField(), new Label( “MI:” ));  //向面板添加三个元素
      TextField tfi = new TextField();
      tfi.setPrefColumnCount(1) ;
      pane.getChildren( ).addAll(tfi,
            new Label ( “Last Name:” ) ,new TextField()); //继续添加三个元素
      Scene scene = new Scene(pane,200, 250);
      primaryStage.setTitle("ShowFlowPane");
      primaryStage.setScene(scene);  primaryStage.show();
   }
}
```

一个Insets对象指定了一个面板边框的四周padding。构造方法InSetS(ll,12,13,14)指定了面板padding为顶部11、右边12、底部13、左边14。还可以使用构造方法Insets(value)来创建一个四条边具有相同值的Insets。

hGap和vGap属性指定了面板中两个相邻节点之间的水平和垂直间隔

tfi .setPrefColumnCount(l)将tfi文本域的期望列数设置为1



14.6 布局面板

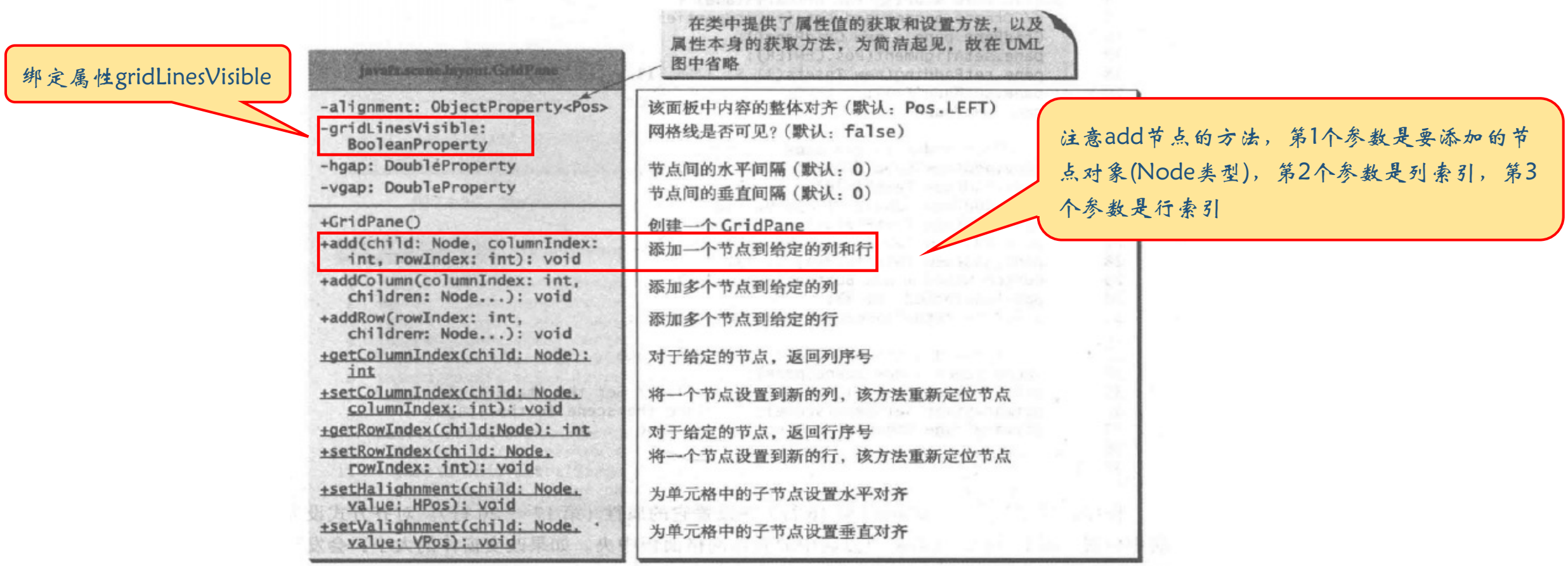
```
public class ShowFlowPane extends Application{
   public void start(Stage primaryStage) {
      FlowPane pane = new FlowPane() ;
      pane.setPadding(new Insets(11, 12, 13, 14));
      pane.setHgap(5);   pane.setVgap(5);
      pane.getChildren().addAll(new Label("First Name:"),
                  new TextField(), new Label( “MI:” )); //向面板添加三个元素
      TextField tfi = new TextField();
      tfi.setPrefColumnCount(1) ;
      pane.getChildren( ).addAll(tfi,
            new Label ( “Last Name:” ) ,new TextF
      Scene scene = new Scene(pane,200, 250);
      primaryStage.setTitle("ShowFlowPane");
      primaryStage.setScene(scene);  primaryStage.s
   }
}
```

默认方式创建的FlowPane将里面的UI控件从左到右依次排列，如果一排放不下自动换行再从左到右继续排列。
可以看到，当改变窗体大小会改变内部元素的布局

14.6 布局面板

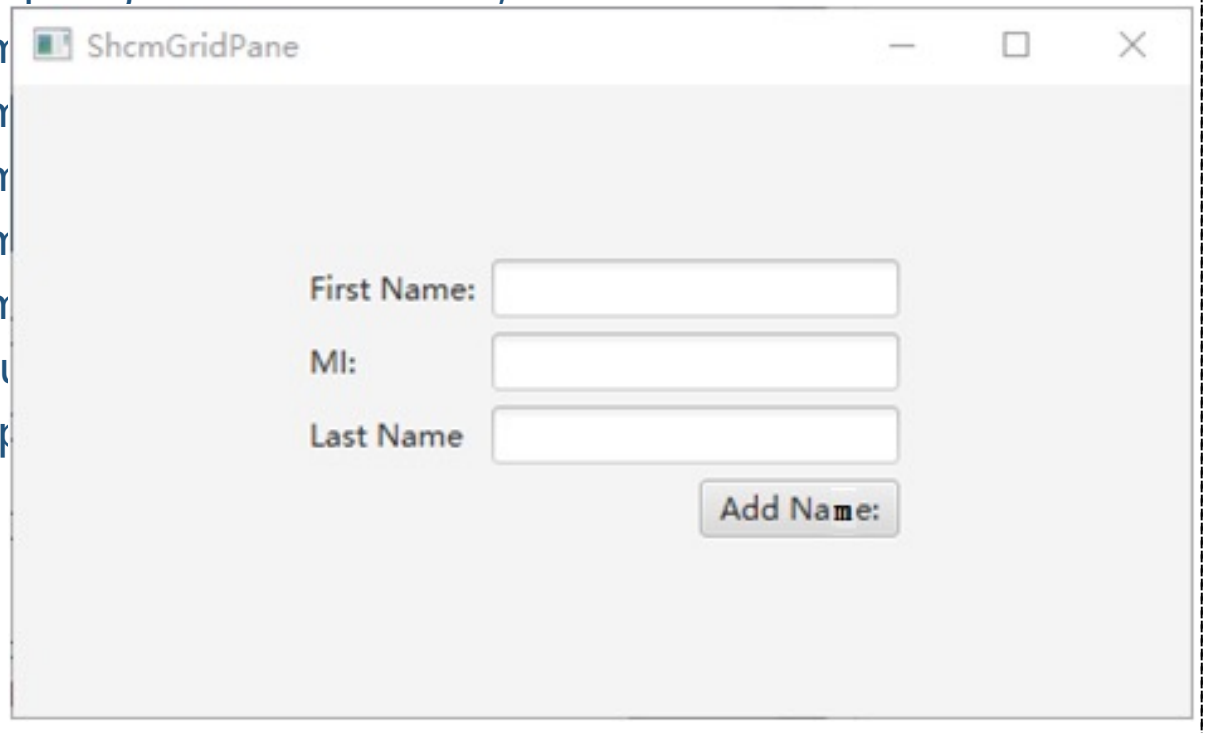
GridPane

- GridPane 将节点布局在面板里一个网格（矩阵）中。节点位置由指定的列和行下标指定



14.6 布局面板

```
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
im
im
im
im
im
pu
```




```
    pane.add(new Label( “First Name:"), 0,0); //0列0行
    pane.add(new TextField(), 1, 0);  //1列0行
    pane.add(new Label( “MI:” ), 0, 1); //0列1行
    pane.add(new TextField(),1,1); //1列1行
    pane.add(new Label( “Last Name” ), 0, 2); //0列2行
    pane.add(new TextField(), 1, 2); //1列2行
    Button btnAdd = new Button("Add Name:");
    pane.add(btnAdd, 1,3);  //1列3行（注意这时一共2列4行）
    //将Button对象在单元格水平右对齐（静态方法）
    GridPane.setHalignment(btnAdd, HPos.RIGHT);

    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShcmGridPane");
    primaryStage.setScene(scene);
    primaryStage.show();
    }
}
```

14.6 布局面板

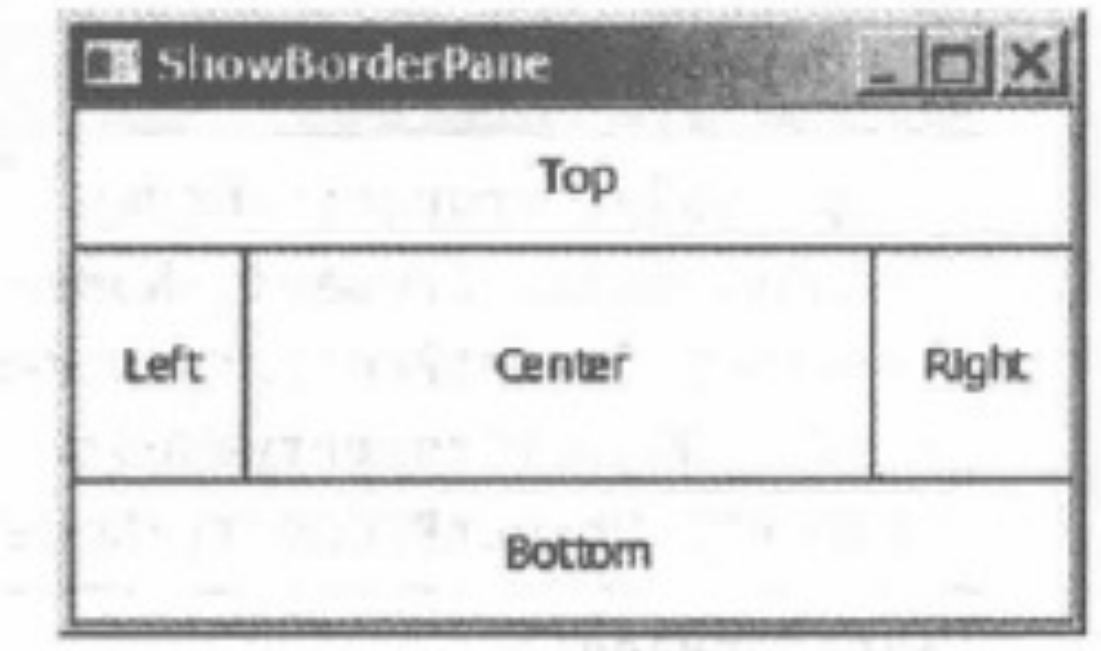
注意Pane作为容器，负责里面UI组件的布局(Layout)。Pane可以相互嵌套，实现任意复杂的布局。

BorderPane

- BorderPane 可以将节点放置在五个区域：顶部、底部、左边、右边以及中间，分别使用setTop(node)、setBottom(node)、setLeft(node)、setRight(node)和setCenter(node)方法。

HBox和VBox

- HBox 将它的子节点（ children) 布局在单个水平行中。VBox 将它的子节点布局在单个垂直列中。



看教材第14章程序清单14-12，14-13

14.6 JavaFX使用注意的问题

□ JavaFX开发策略：Main和controller关联， fxml界面和controller关联，Model和controller关联

- JavaFX Project通常包含Main.java, css样式表，fxml界面配置文件、控制器controller、Model类（多个）
- css样式表用来设计fxml界面中的UI样式，可用默认样式。
- Main.java负载启动程序，可以存储一些应用总体信息：数据库连接，登录用户及口令等
- 控制器controller，用来处理界面中的事件
- 控制器应和Main对象建立关联（把Main对象set到控制器），以便获得其总体信息用于数据库操作等。
- fxml界面也和controller建立关联，以便controller能绑定界面里UI组件（通过fx:id 和@FXML）
- Model类也应该和controller关联（把Model对象set到控制器里），以便controller可以调用模型的方法

□ 注意:UI控件的类型必须import自JavaFX（不要import错了，注意不要import swing里面的控件）

14.6 JavaFX使用注意的问题

JavaFX如何建立关联：Main和controller

- 在controller中定义变量myApp保存Main的对象，并定义函数setUp供Main调用。

```
private Main myApp;
public void setUp(Main application) {    myApp = application;    }
```

- Main通过FXMLLoader加载fxml后，找到controller，调用controller.setUp(this)建立关联
- Model和controller的关联方法类似

JavaFX如何建立关联：fxml和controller

- 在fxml中指定控件的fx:id 属性来和controller绑定。如<TextField fx:id= *"inputText"* …>。在controller中则通过@FXML标记将fxml里的UI控件绑定到controller里的实例变量

```
@FXML
private TextField inputText  //该实例变量绑定到fxml里的对应控件，控制器通过该实例变量访问或设置控件的属性
```

- 在fxml中通过控件的属性 onAction= *"#handleButtonAction"* 给按钮的处理函数取名。在controller中通过@FXML

```
private void handleButtonAction(ActionEvent event)定义处理函数;
```