> 由 KiOii (*_EM_Cpper*)整理。 (KiOii (*_EM_Cpper*) makes this note.)

Getting Started

In this chapter

- **What JavaFX is**
- **The history of JavaFX**
- **How to write your first JavaFX application**
- **How to use the NetBeans Integrated Development Environment to work with a JavaFX application**
- **How to pass parameters to a JavaFX application**
- **How to launch a JavaFX application**
- **The life cycle of a JavaFX application**
- **How to terminate a JavaFX Application**

What Is JavaFX ?

- *define:* open source Java-based framework for developing **rich client applications**

  > The JavaFX library is available as a public Java application programming interface(API)

- **features**
  - JavaFX is **written** in Java
  - JavaFX supports **data binding** through its libraries
  - JavaFX code can be written **using** any Java virtual machine (JVM)-supported scripting languages such as Visage, Groovy, and Scala
  - two ways to build **UI** : **Java** code and **FXML**
  - multimedia support
  - JavaFX provides out-of-the-box support for applying **effects and animations**

GUI

- *define:* JavaFX GUI 由 scene graph构成

scene graph

- *define:* a collection of visual elements

  > 虚拟元素集合，虚拟元素被称为节点(node)

- 场景图使用JavaFX API构建
- *use:* nodes in a scene graph can handle user input and user gestures

  > 场景图中的节点可以处理用户输入和用户动作
  >
  > 它们可以有效果(effect),转换(transformation),状态(state)

node

- 作用：处理用户输入和用户动作
- 特点：可以有效果，转换，状态
- 类型：UI控件 （按钮、文本域、2D、3D、图像、媒体、网页内容、图表）

JavaFX component

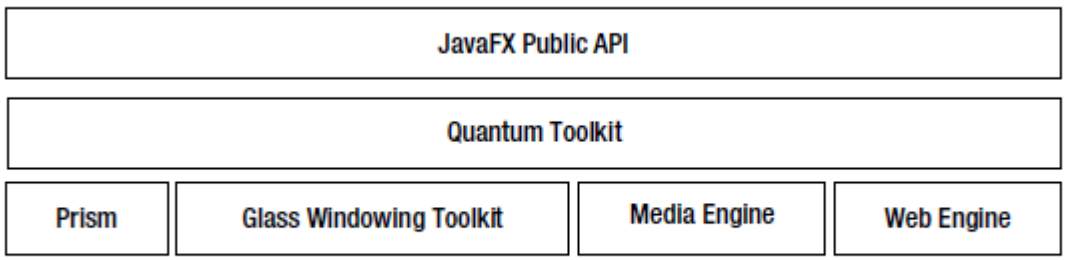

Figure 1-1. Components of the JavaFX platform

> 介绍各部分

prism

- *define:* hardware-accelerated graphics pipeline

  > 硬件加速图形渲染管道

- *use:* rendering the scene graph

  > 渲染场景图

- *wraning:* 如果平台没有硬件加速图形渲染，则回退到Java 2D渲染模式

  > 硬件加速图形渲染：DirectX on Windows and OpenGL on Mac，Linux，embedded platforms

- 为什么能进行渲染?

  > 使用一个独立的线程，区别于处理用户输入的 Java Application Thread

- 如何加速处理?

  > 通过在渲染当前帧时处理下一帧来加快处理

- 如何实现对场景图的重新渲染

  > **机制**：一旦屏幕内容被修改，Prism 使用 pulse event来实现对场景图的同步
  >
  > **过程**：当屏幕需要重新渲染时，一个pulse event在JavaFX Application Thread的事件队列中排队

> **pulse event** : 指出了Prism中的渲染层和场景图是不同步的这样一条事件
>
> **结果**：least frame should be redered：Prism level中的最新帧必须被渲染
>
> **注意**：pulse event 被限制为 最大每秒60 frames

Glass Windowing Toolkit

- 功能: provides graphics and windowing sevice

  > 提供图形和窗口服务，例如使用本地操作系统的窗口和定时器

- 功能：manages event queues

  > 负责事件队列
  >
  > 在JavaFX中，事件队列由一个单独的操作系统来管理（JavaFX Application Thread)
  >
  > 用户输入事件被分配到 JavaFX Application Thread 上

- *wraning：* JavaFX需要一个只能在 JavaFX Application Thread 上被修改的实时场景图（live scene graph）

media engine

- *define:* providing media support in JavaFX
- 特点：利用了平台上可用的编码解码器
- 机制：使用单独的线程去处理 media frames，使用JavaFX Application Thread去实现frames和scene graph的同步
- 底层：media engine is based on `GStreamer`

  > GStreamer 是开源多媒体框架

web engine

- *define:* processing web content (HTML)

  > 处理嵌入在 scene graph中的web内容

- 如何渲染?

  > Prism负责渲染web content

- 底层：web engine is based on `WebKit`

  > WebKit 是一个开源的web浏览器引擎
  >
  > 支持：HTML5、CSS、JavaScript、Document Object Model（DOM）

Quantum toolkit

- *define:* 是对底层（Prism，Glass，Media Engine，Web Engine）的抽象，并且促进了这些底层东西的协调

History of JavaFX

> From Java 8, the version numbers of Java SE and JavaFX will be the same

System Requirements

- Java Development Kit 8

JavaFX Runtime Library

- All JavaFX classes are packaged in a JAR file name : `jfxrt.jar`

JavaFX Source Code

- `javafx-src.zip`

Your First JavaFX Application

Creating the HelloJavaFX Class

- A JavaFX application is a class

  > must inherit form the **Application** class in **javafx.application** package

```
package com.javafx.test;

import javafx.application.*;

public class HelloFXApp extends Application{
    // ...
}
```

Override the start() method

- *wraning:* Application class 包含一个abstract start(Stage stage) method

  > 你必须对其进行重写

- ***start*** method : `public abstract void start(Stage stage) throws java.lang.Exception`

```
package com.javafx.test;

import javafx.application.Application; // Application class
import javafx.stage.Stage;             // Stage class

public class HelloFXApp extends Application
{
    @Override
    public void start(Stage stage)
    {

        // JavaFX Application的入口点方法
```

```
        // stage : JavaFX runtime提供给程序的 primary stage
    }
}
```

Showing the stage

- Be used to display a scene

  > 在JavaFX中 stage是scene的容器
  >
  > JavaFX的运行时负责处理所有的环境细节，在桌面的显示和在浏览器中的显示是不同的

- *wraning:* 一开始，stage上面是没有scene的，因此你需要去创建

  > 你必须去show the stage以便于显现 scene上面的visuals（各种特效组件等等）

- ***show*** method : show stage
- ***setTitle*** method : set a title

```
package com.javafx.test;

import javafx.application.Application; // Application class
import javafx.stage.Stage;             // Stage class

public class HelloFXApp extends Application
{
    /**
     *  程序的入口点
     */
    @Override
    public void start(Stage stage)
    {
        // 我们可以设置标题
        stage.setTitle("Hello JavaFX Appliation");
        // stage 负责 呈现 scene
        stage.show();
    }
}
```

Launching the Application

- Two ways:
  - 如果没有main则在命令行上也可运行（自动运行javaFX程序）
  - 如果有main，需要`Application.launch(args);`这样会调用start方法

```
package com.javafx.test;

import javafx.application.Application; // Application class

import javafx.stage.Stage;             // Stage class
```

```
public class HelloFXApp extends Application
{

    public static void main(String... args)
    {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage)
    {
        // 我们可以设置标题
        stage.setTitle("Hello JavaFX Appliation");
        // stage 负责 呈现 scene
        stage.show();
    }
}
```

> 当应用程序退出时，launch会调用**Platform.exit()**method

Adding a Scene to the Stage

> setScene 加载场景

- **Scene** class : `javafx.scene` package

  > A stage contains one scene, and a scene contains visual contents

- **VBox** class: 根节点包含孩子节点，孩子节点包含它们的孩子节点，等等。VBox就是根节点

  > veritical box,竖框

  - **getChild** method : 获取孩子节点
    - **add** method : 添加节点

      ```
      root.getChild().add(msg);
      ```

- **Parent** class: Any node that inherits from the `javafx.scene.Parent` class can used as the root node fo a scene

  > 任何继承自Parent的节点都是根节点，VBox, HBox, Pane, FlowPane, GridPane, or TilePane都是已知的根节点

```
package com.javafx.test;

import javafx.application.Application; // Application class
import javafx.stage.Stage;             // Stage class
import javafx.scene.Scene;      // scene
import javafx.scene.text.Text;  // child node
import javafx.scene.layout.VBox;// root node
```

```
public class HelloFXApp extends Application
{

    public static void main(String... args)
    {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage)
    {
        // 我们可以设置标题
        stage.setTitle("Hello JavaFX Appliation");
        // 设置 Text node
        VBox root = new VBox();
        Text msg = new Text("Hello JavaFX!" + " 完美");
        root.getChildren().add(msg);
        // 设置 scene
        Scene scene = new Scene(root,400,100);
        stage.setScene(scene);

        // stage 负责 呈现 scene
        stage.show();
    }
}
```

Improving the HelloFX Application

button and text field

```
// Create a button with "Exit" text
Button exitBtn = new Button("Exit");
```

> 当按钮被点击时，ActionEvent 被触发，你可以添加一个ActionEvent处理程序来处理事件

- **setOnAction** method : set an ActionEvent handler for the button

```
// Using a lambda expression
exitBtn.setOnAction(e -> Platform.exit());
// Using an anonymous class  (匿名类)
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
...
exitBtn.setOnAction(new EventHandler<ActionEvent>() {
@Override
public void handle(ActionEvent e) {
Platform.exit();
}
});
```

```
package com.javafx.test;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;

import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class ImproveHelloFXApp extends Application{
    public static void main(String... args)
    {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage)
    {
        // name Label node
        Label nameLbl = new Label("你的名字");
        // textfield node
        TextField nameFld = new TextField();

        // exit button node
        Button exitBtn = new Button("Exit");


        exitBtn.setOnAction(
                new EventHandler<ActionEvent>()
                {
                    @Override
                    public void handle(ActionEvent e)
                    {
                        Platform.exit();
                    }
                }
                );
        // root node
        VBox root = new VBox();
        // add child node
        root.setSpacing(3);
        root.getChildren().addAll(nameLbl,nameFld,exitBtn);

        // scene
        Scene scene = new Scene(root,400,100);
```

```
        // stage
        stage.setScene(scene);
        stage.show();
    }
}
```

The Lift Cycle of a JavaFX Application

- JavaFX在运行时创建几个线程，在应用程序的不同stage中，线程被用来执行不同的任务
  - **JavaFX-Launcher**
  - **JavaFX Application Thread**

> launch() 创建了这些线程

- JavaFX运行时按顺序调用JavaFXApplication的如下方法
  - The **no-args constructor**
  - The **init**() method
  - The **start**() method
  - The **stop**() method

关于如上两个线程

- JavaFX运行时在**JavaFX Application Thread**线程中创建一个特别的**Application**类的实例
- **JavaFX Launcher Thread** 调用那个实例的**init**()方法，**Application**类中的**init**方法是空的(empty body)

> 你可以重写该init方法

- wraning: 不允许在 **JavaFX Launcher Thread**中创建**Stage**或者**Scene**对象

> UI controls are ok. （buttons or shapes)

> 它们必须在**JavaFX Application Thread**中创建
> 因此**你不能在init()方法中创建Stage或者Scene**

- **JavaFX Application Thread** 调用那个实例的**start**()方法

> 该方法是abstract的，因此你必须override

  - 此时，**launch**()方法等待 JavaFX application结束，结束时 **JavaFX Application Thread**调用那个实例的**stop**() method

> 注意：stop和init默认实现都是empty body

执行过程

> FXLifeCycleApp() constructor: JavaFX Application Thread init() method : JavaFX-Launcher start() method : JavaFX Application Thread stop() method : JavaFX Application Thread

Terminating a JavaFX Application

> JavaFX application可能被显示或者隐式退出（terminated）

- **explicitly terminate**
  - **use :** 通过调用`Platform.exit()`方法 在start方法中或者后，stop（）被调用执行退出。
    > 这时，如果只有守护线程（daemon threads ）在运行，JVM将退出（exit）
  - **wraning :** 如果`Platform.exit()`在 constructor 或者 init 方法总被调用，stop()方法可能不会被调用（要注意！）
  - **wraning :** JavaFX application如果在web brower上面运行，`Platform.exit()`可能没有效果
- **implicitly terminate**
  - 当最后的窗口被关闭时自动退出
  - 打开默认退出的开关： using static `setImplicitExit(boolean implicitExit)`
    > 该静态方法由Platform类拥有
    - use : `setImplicitExir( true );`打开开关
      > 传递 false 则关闭，默认是开启的（因为目前大多数应用都是关闭窗口然后退出的）
      >
      > **一旦开关被打开，当JavaFX Application Thread terminate时，stop被调用（也就是默认情况下是这样）**
- **wraning :** Terminating the **JavaFX Application Thread** does not always terminate the JVM
  > JavaFX Application Thread的退出不意味着JVM的退出
  >
  > **之后，当所有正在运行的非守护线程退出时，JVM终止（terminate）**
- **wraning :** 一旦你关闭了默认exit开关，需要显式调用exit（Platform.exit）