**一**.**javaFX Preloader**: JavaFX在Application加载的时候提供了Proloader应用，可以展现loading process bar或者others notification。

**二.** [**Model-View-Controller** (MVC)](http://zh.wikipedia.org/wiki/MVC)是一个非常重要的软件设计原则。按照MVC模式可以将我们的应用程序划分成3个部分，然后为这每一部分建立自己的包 (在源代码文件夹上右键， 选择 新建 | 包):

* ch.makery.address - 放置所有的控制器类(也就是应用程序的业务逻辑)
* ch.makery.address.model - 放置所有的模型类
* ch.makery.address.view - 放置所有界面和控件类

**注意:** view包里可能会包含一些控制器类，它可以直接被单个的view引用，我们叫它 **视图-控制器**。

**三**. 有两种方式来创建用户界面，一终是能过XML文件来定义，另外一种就是直接通过java代码来创建. 我们这里将使用XML的方式来创建大部分的界面，这种方式将会更好的将业务逻辑和界面分开，以保持代码的简洁。使用Scene Builder(所见即所得)来编辑我们的XML布局文件，它可以避免我们直接去修改XML文件。

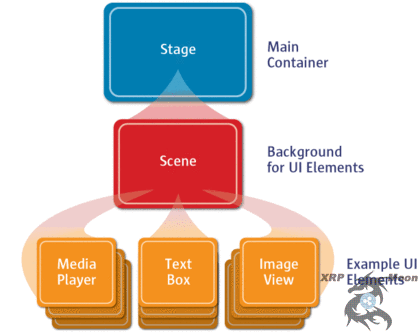
**四**. 点击菜单中的 Preview | Show Preview in Window 可以预览你设计好的界面，试着缩放预览的界面，你会发现TableView会随着窗口的缩放而变化。

**五**. 选择这个 TableView ，在右边面板中将它的 Column Resize Policy（列调整策略） 修改成 constrained-resize（约束调整） (同样是在属性设置项里面)。确保这个TableView的列能够铺满所有的可用空间。

**六**. 注意: 添加一个控件到已经存在的行里面去，你可在这行的行号上右键选择 “Add Row”。

**七**. 你可能注意到了IDE生成的 MainApp.java 继承自 Application 同时包含了两个方法， 这是一个JavaFX应用程序的最基本的代码结构，这里最重要的方法是 start(Stage primaryStage) ，它将会在应用程序运行时通过内部的 main 方法自动调用。

正如你所看到的，这个start(...) 方法会接收一个 Stage 类型的参数，下面的图向你展示了一个JavaFX应用程序的基本结构。

  
\*Image Source: http://www.oracle.com\*

**一切看起来象是剧场里表演**: 这里的 Stage 是一个主容器，它就是我们通常所认为的窗口（有边，高和宽，还有关闭按钮）。在这个 Stage 里面，你可以放置一个 Scene，当然你可以切换别的 Scene，而在这个 Scene 里面，我们就可以放置各种各样的控件。

**八.**对整个项目进行clean build，否则会找不到加载的fxml的路径

**九**. 在JavaFX中,对一个模型类的所有属性使用 [Properties](http://docs.oracle.com/javase/8/javafx/api/javafx/beans/property/Property.html)是很常见的. 一个 Property 允许我们, 打个比方, 当 lastName 或其他属性被改变时自动收到通知, 这有助于我们保持视图与数据的同步，阅读 [Using JavaFX Properties and Binding](http://docs.oracle.com/javase/8/javafx/properties-binding-tutorial/binding.htm) 学习更多关于Properties 的内容。

[LocalDate](http://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html), 我们使用了 birthday 类型, 这是一个新的部分在 [Date and Time API for JDK 8](http://docs.oracle.com/javase/tutorial/datetime/iso/).

### 十. ObservableList

我们处理JavaFX的view classes需要在人员列表发生任何改变时都被通知.  
这是很重要的,不然视图就会和数据不同步.为了达到这个目的,JavaFX引入了一些新的[集合类](http://docs.oracle.com/javase/8/javafx/collections-tutorial/collections.htm).

在这些集合中, 我们需要的是ObservableList. 将以下代码增加到MainApp类的开头去创建一个新的ObservableList. 我们也会增加一个构造器去创建一些样本数据和一个公共的getter方法:  
**十一**. 所有fxml文件需要访问的属性和方法必须加上 @FXML 注解.实际上,只有在私有的情况下才需要, 但是让它们保持私有并且用注解标记的方式更好!

* initialize() 方法在fxml文件完成载入时被自动调用. 那时, 所有的FXML属性都应已被初始化.
* 我们在表格列上使用setCellValueFactory(...) 来确定为特定列使用Person对象的某个属性. 箭头 -> 表示我们在使用Java 8的 *Lambdas* 特性. (另一个选择是使用 [PropertyValueFactory](http://docs.oracle.com/javase/8/javafx/api/), 但它不是类型安全的).=

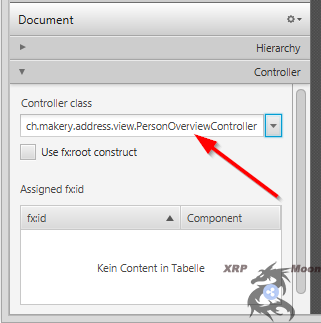
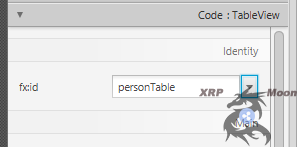
**十二.**

### 连接 MainApp 和 PersonOverviewController

setMainApp(...) 必须被 MainApp 类调用. 这让我们可以访问MainApp对象并得到Persons的列表和其他东西. 用以下代码替换showPersonOverview() 方法. 它包含了新增的两行:

## 将View与Controller挂钩

我们快要完成了! 但是有件小事被遗漏了: 至今没有告诉 PersonOverview.fxml使用的是哪个控制器以及元素与控制器中的属性的对应关系.

1. 使用\*SceneBuilder\* 打开 PersonOverview.fxml.
2. 打开左边的 Controller 组选择PersonOverviewController 作为 **controller class**.  
   
3. 在 Hierarchy 组选择TableView 并选择 Code 组将 personTable 作为 **fx:id**.  
   
4. 对列做相同的事并且将 firstNameColumn and lastNameColumn分别作为 **fx:id** .
5. 对在第二列的 **each label** , 选择对应的 **fx:id**.  
   
6. 重要事项: 回到eclipse并且 **refresh the entire AddressApp project** (F5). 这是必要的因为有时候eclipse并不知道在Scene Builder中作出的改变.

**十三.**

## 删除按钮

我们的用户接口已经包含一个删除按钮，但是没有任何功能。我们能在\*SceneBuilder\*中的按钮上选择动作。在我们控制器中的任何使用@FXML（或者它是公用的）注释的方法都可以被\*Scene Builder\*访问。因此，让我们在PersonOverviewController类的最后添加一个删除方法。

**十四.**

### 为什么是XML？

持久性数据的一种最常用的方法是使用数据库。数据库通常包含一些类型的关系数据（例如：表），当我们需要保存的数据是对象时。这称[object-relational impedance mismatch](http://wikipedia.org/wiki/Object-relational_impedance_mismatch)。匹配对象到关系型数据库表有很多工作要做。这里有一些框架帮助我们匹配（例如：[Hibernate](http://www.hibernate.org/)，最流行的一个）。但是它仍然需要相当多的设置工作。

对于简单的数据模型，非常容易使用XML。我们使用称为[JAXB](https://jaxb.java.net/)（\*\*J\*\*ava **A**rchitecture for **X**ML **B**inding）的库。只需要几行代码，JAXB将允许我们生成XML输出，如下所示：

**十五.**

### 使用JAXB

JAXB已经包含在JDK中。这意味着我们不需要包含任何其它的库。

JAXB提供两个主要特征：\*\*编列(marshal)\*\*Java对象到XML的能力，\*\*反编列(unmarshal)\*\*XML到Java对象。

为了让JAXB能够做转换，我们需要准备我们的模型。

**十六.**

现在，再次测试。试着保存和加载XML文件。在重启之后，它应该自动加载最后使用的文件。

## 它如何工作

让我们看下它是如何一起工作的：

1. 应用程序使用MainApp中的main(…)方法启动。
2. 调用public MainApp()构造函数添加一些样例数据。
3. 调用MainApp的start(…)方法，调用initRootLayout()从RootLayout.fxml中初始化根布局。fxml文件有关于使用控制器的信息，连接视图到RootLayoutController。
4. MainApp从fxml加载器中获取RootLayoutController，传递自己的引用到控制器中。使用这些引用，控制器随后可以访问MainApp的公开方法。
5. 在initRootLayout方法结束，我们试着从Perferences中获取\*最后打开的人员文件\*。如果Perferences知道有这样一个XML文件，我们将从这个XML文件中加载数据。这显然会覆盖掉构造函数中的样例数据。

#### 图表控制器如何工作

1. 控制器需要从FXML文件中访问两个元素:

* barChar：它有String和Integer类型。String用于x轴上的月份，Integer用于指定月份中人员的数量。
* xAxis：我们使用它添加月字符串

1. initialize() 方法使用所有月的列表填充x-axis。
2. setPersonData(…)方法将由MainApp访问，设置人员数据。它遍历所有人员，统计出每个月生日的人数。然后它为每个月添加XYChart.Data到数据序列中。每个XYChart.Data对象在图表中表示一个条形。

 WebView：它是一个使用了WebKitHTML技术的Web组件，可用于在JavaFX应用程序中嵌入Web页面。在WebView中运行的JavaScript可以方便地调用JavaAPI，并且JavaAPI也可以调用WebView中的JavaScript。对附加的HTML5特性的支持，包括Web Socket、Web Worker、Web Font、打印功能等都被添加到了JavaFX8中。参考《增加HTML内容到JavaFX应用程序中（Adding HTML Content to JavaFX Applications）》章节来了解更多信息。

与Swing互操作：现有的Swing程序可以通过JavaFX的新特性升级，例如多媒体播放和Web 内容嵌入。在JavaFX8中加入了SwingNode类，它可以将Swing内容嵌入到JavaFX程序中。参考SwingNode API Javadoc和《在JavaFX应用程序中嵌入Swing 内容（Embedding Swing Content in JavaFX Applications）》章节来了解更多信息。

内置的UI控件和CSS：JavaFX提供了开发一个全功能应用程序所需的所有主要控件。这些组件可以使用标准的Web技术如CSS来进行装饰。在JavaFX8中，DatePicker和TreeView UI控件是可用的，并且可以使用标准的Web技术如CSS来进行美化。参考《使用JavaFX UI控件（Using JavaFX UI Controls）》章节来了解更多信息。另外CSS样式控制类都变成了公开API，它们可以使用CSS来为对象增加样式。

Modena主题：在JavaFX8中，提供了新的Modena主题来替换原来的Caspian主题。不过在Application的start()方法中，可以通过加入setUserAgentStylesheet(STYLESHEET\_CASPIAN)代码行来继续使用Caspian主题。在com的[Modena blog](http://fxexperience.com/2013/01/modena-new-theme-for-javafx-8/)中找到更多信息。

3D图像处理能力：在JavaFX8中的3D图像处理API中加入了一些新的API，包括Shape3D (Box, Cylinder, MeshView和Sphere 子类)，SubScene, Material, PickResult, LightBase (AmbientLight 和PointLight子类)，SceneAntialiasing等。在本次发布中Camera类API也得到了更新。要了解更多信息，可以参考《开始学习JavaFX 3D图形（Getting Started with JavaFX 3D Graphics）》文档和对应的JavaDoc，包括scene.shape.Shape3D，javafx.scene.SubScene，javafx.scene.paint.Material，javafx.scene.input.PickResult和javafx.scene.SceneAntialiasing。

**十七**. **图2-1说明了JavaFX平台的组件架构**。后面的章节将会对每个组件及其之间的联系进行说明。在JavaFX公开的API之下JavaFX代码运行引擎。它由几大部分组成：一个JavaFX 高性能图形引擎，名为Prism；一个简洁高效的窗体系统，名为Glass；一个媒体引擎；一个web引擎。尽管这些组件并没有公开对外暴露，但是下面的描述将有助于你理解一个JavaFX应用是如何运行的。

● 场景图(Scene Graph)

● JavaFX功能的公开API(Java Public APIs for JavaFX Features)

● 图形系统(Graphics System)

● Glass窗体工具包(Glass Windowing Toolkit)

● 多媒体和图像(Media and Images)

● Web组件(Web Component)

● CSS

● UI控件(UI Controls)

● 布局(Layout)

● 2-D和3-D转换(2-D and 3-D Transformations)

● 视觉特效(Visual Effects)

**图2-1 JavaFX架构图**

[](http://www.javafxchina.net/blog/wp-content/uploads/2015/06/2_1-jfxar_dt_001_arch-diag.png)**十八.场景图**

JavaFX场景图(Scene Graph)位于图2-1中的顶层部分，它是构建JavaFX应用的入口。它是一个层级结构的节点树，表示了所有用户界面的视觉元素。它可以处理输入，并且可以被渲染。

在场景图中的一个元素被称为一个节点(Node)。每个节点都有一个ID、样式类和包围盒(bounding volume)。除了根节点之外，在场景图中的所有节点都有一个父节点、0个或多个子节点。节点还可以有如下特性：

● 效果(Effects)，例如模糊和阴影

● 不透明度(Opacity)

● 变换(Transforms)

● 事件处理器(Event handlers，例如鼠标、键盘和输入法)

● 应用相关的状态(Application-specific state)

与Swing和AWT不同，JavaFX场景图还包括图元，例如矩形、文本，还有控件、布局容器、图像、多媒体。

对于大多数用户来说，场景图简化了UI设计，尤其是对富客户端应用来说。对场景图中使用动画可以很容易地通过javafx.animation API和声明式方法(例如XML文档)来实现。

javafx.scene API允许创建和定义各种内容，例如：

● 节点(Nodes)：包括各种形状(2D或3D)、图像、多媒体、内嵌的Web浏览器、文本、UI控件、图表、组和容器

● 状态(State)：变换(节点的定位和定向)、视觉效果、以及内容的其它视觉状态。

● 效果(Effects)：可以改变场景图节点的外观的简单对象。例如模糊、阴影、图像调整。

**十九**.JAVAFX功能的JAVA 公开API

在图2-1中描述的JavaFX架构的顶层提供了一个支持富客户端应用开发的完整Java API集合。这些API为构建富客户端程序提供了无比的自由和弹性。JavaFX平台将Java平台上众多最佳功能以及全面、沉浸式的多媒体功能集成到了一个直观、全面的一站式开发环境之中。这些JavaFX功能API包括：

● 允许使用Java的强大特性，例如泛型、注解、多线程、Lamda表达式(从Java8开始引入)

● 允许Web开发者通过其他基于JVM的动态语言来使用JavaFX，例如Groovy和JavaScript。

● 允许Java开发者使用其他系统语言来开发大型、复杂的JavaFX应用，例如Groovy。

● 允许使用绑定技术，包括高性能的lazy binding、binding expressions、bound sequence expressions、partial bind reevaluation。其它语言(像Groovy)可以通过与JavaFX 脚本类似的语法来使用这些绑定库。

● **扩展了Java 集合库，包括可观察的list和map，允许应用程序将用户界面和数据模型进行连接，观察数据模型的改变并同时更新对应的UI控件。**

JavaFX API和编程模型是JavaFX 1.x 产品线的延续。大多数JavaFX API已经被直接用到了Java之中。有一些API，例如布局和多媒体，由于具有更多的细节，它们已经基于JavaFX 1.x的用户反馈进行了改进和简化。JavaFX采用了很多Web 标准，例如CSS用于控件的样式化、ARIA用于易用性标准化。对更多Web标准的应用正处在review状态。

#### 二十. 图形系统

JavaFX图形系统(Graphics System)是在图2-1中蓝色部分，它是在JavaFX场景图层之下的实现细节。它支持2D和3D场景图。当系统中的图形硬件无法支持硬件加速渲染时，它将提供软件渲染技术。

在JavaFX平台中实现了两套图形加速流水线：

● Prism用于处理渲染工作。它可以在硬件和软件渲染器之上工作，包括3D。它负责将JavaFX场景进行光栅化和渲染。下面的各种渲染方式都有可能被用到：

○ 在Windows XP和Vista上的DirectX 9

○ 在Windows 7上的DirectX 11

○ Mac、Linux、嵌入式设备上的OpenGL

○ 当硬件加速技术不支持时使用软件渲染

如果可能将会优先使用硬件加速，但是如果硬件加速不可用则会使用软件渲染，软件渲染技术已经内置于JRE之中。这点在展示3D场景时尤其重要。当然，使用硬件加速时性能将会更好。

● Quantum Toolkit 将Prism和Glass Windowing ToolKit绑在一起，使得它们可以被其上层的JavaFX层使用。它也负责管理与渲染有关的事件处理的线程规则。

#### 二十一.GLASS窗体工具包

Glass窗体工具包(Glass Windowing Toolkit)在图2-1中处于整体架构的中间位置，它处于JavaFX图形技术栈的最底层。其主要职责是提供本地操作服务，例如窗体、计时器、皮肤。它是连接JavaFX层与本地操作系统的平台无关层。

它还负责管理事件队列。与AWT自行管理事件队列不同，它使用本地操作系统的事件队列来调度线程。同样与AWT不同的是Glass Toolkit与JavaFX应用运行在一个线程之上，而在AWT中本地系统中的那部分代码运行在一个线程中，而Java部分的运行在另外一个线程上。这导致了很多问题，而这些问题在JavaFX由于采用了统一的线程而都得到了解决。

##### 线程

在系统运行时会有下面的两个或更多线程：

● JavaFX应用程序线程：这是JavaFX应用开发者使用的主要线程。任何“活动”的场景都是窗体的一部分，它们都必需通过此线程来访问。场景图可以通过一个后台线程来创建和控制，但是如果其根节点与任何活动对象相关，则该场景图必需通过JavaFX应用程序线程来访问。这允许开发者在背景线程上创建复杂的场景图，而与此同时在“活动”场景中保持动画的流畅。由于JavaFX应用程序线程与Swing和AWT事件调度线程(EDT)不同，所以在将JavaFX代码嵌入到Swing应用程序中时必需小心。

● Prism渲染线程：此线程处理渲染工作，使其与事件调度器独立开来。它使得第N帧在被渲染时可以对第N+1帧进行处理。这种并行处理机制具有很大的优势，尤其是在具有多处理器的现代操作系统上。Prism渲染线程可以带有多个光栅化线程来协助处理渲染相关的工作。

● 多媒体线程:此线程会在后台运行，通过使用JavaFX应用程序线程来在场景图中同步最近的帧。

##### 脉冲

脉冲(Pulse)是一个事件，用于通知JavaFX场景图何时该使用Prism来同步场景图之上的元素状态。一个脉冲被限制为最多60帧每秒(fps)，并且会在场景图上运行动画时被触发。即使动画没有运行，如果场景图中的内容发生了改变，也会使脉冲被调度。例如，如果一个Button的位置被改变了，那么一个脉冲将会被调度。

如果一个脉冲被触发，在场景图之上的元素状态将会与渲染层进行同步。脉冲向开发者提供了一种异步处理消息的方法。这种特性对于系统在脉冲上进行批处理和执行事件非常重要。

Layout和CSS也与脉冲事件相关。场景图中的改变可以导致很多布局或CSS的更新，这些更新有可能会影响性能。系统在每个脉冲之上自动进行一次CSS和布局处理，以此降低对系统性能的影响。应用开发者也可以根据需要手工在脉冲之前触发布局。

Glass WindowToolkit负责使用高分辨率的本地计时器来执行脉冲事件。

#### 二十二. 布局

布局容器(Layoutcontainer)或面板(Pane)允许对JavaFX应用程序场景图中的UI控件进行灵活、动态的排布。JavaFX Layout API包括下列容器类：

● BorderPane类将其内容节点放到上、下、左、右、中各个区域中。

● HBox类将其内容节点横向排成一行。

● VBox类将其内容节点纵向排成一列。

● StackPane类将其内容节点摞在一起。

● GridPane类允许开发者创建一个灵活的网格，按行列来布局其内容节点。

● FlowPane类将其内容按行或列进行”流式“布局，当遇到横向或纵向的边界时自动进行换行或换列。

● TilePane类将其内容放到统一大小的单元格中。

● AnchorPane类允许开发者创建锚节点，将控件停靠于布局的上下左右各边，也可以居中停靠。

为了获得理想的布局结构，可以在JavaFX应用中嵌套使用各类布局。

#### 二十三. 2D和3D变换

在JavaFX场景图中的每个节点都可以使用下面的javafx.scene.tranform包中的类来进行x-y坐标系变换：

● translate——将一个节点在xyz坐标系中从一个位置移动另外一个位置。

● scale——将一个节点在xyz坐标系中根据缩放因子进行缩放。

● shear——旋转一个坐标轴，这样x轴和y轴不再是垂直的了。节点的坐标值会根据制定的倍数进行变换。

● rotate——根据scene中指定的一个支点对节点进行旋转。

● affine——执行从一个2D/3D坐标系到另外一个2D/3D坐标系的线性映射，同时保留线条的 ‘straight’ 和’parallel’ 属性。这个类应与Translate、Scale、Rotate、Shear变换类同时使用，一般不要直接使用。

#### 二十四.视觉效果

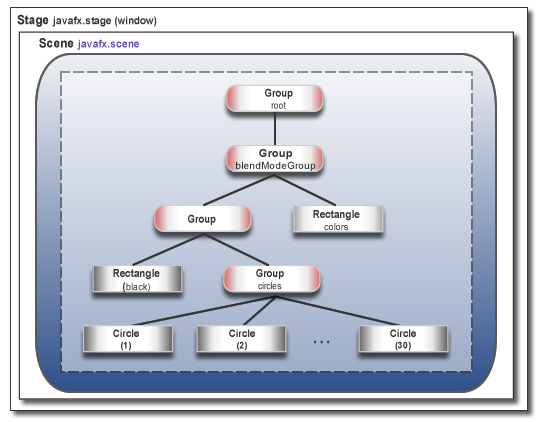
在JavaFX场景图中开发富客户端界面包括使用视觉效果来实时地美化JavaFX应用程序的外观。JavaFX的视觉效果主要是基于像素的图像，因此它们需要先获取场景图中节点渲染成图像，再将视觉效果添加上去。

JavaFX中通过下面的类来提供一些常用的视觉效果：

● Drop Shadow——应用视觉效果后将为给定的内容渲染一个在它的后面的阴影。

● Reflection——在真实的内容后面渲染一个反射倒影。

● Lighting——模仿一个光源的照射效果，使一个平面的对象看起来更真实、具有三维效果。

****