WPF的学习 2021/7/25

笔记本: 暑期实习
创建时间: 2021/7/25 15:42 **更新时间:** 2021/7/25 17:35
作者: 134exetj717
URL: http://wiki.suncaper.net/pages/viewpage.action?pageId=41877554

- x命名空间：将xaml中的编译结果和c#中的编译结果进行合并。相当于设置了一个字典，将x:后面的内容放到字典C#中进行查询

x:Key
在WPF中，几乎每个元素都有自己的Resource属性，这个属性就是“key-value”的集合。只要把元素放进这个集合里，这个元素就成了资源字典中的一个条目。当然，为了能检索到这个条件，就必须为它添加x:Key。x:key的作用就是使用为资源贴上用于检索的索引，它为资源字典中的每个资源设置一个唯一键。
x:Class
这个Attribute是告诉XAML编译器将XAML编译器编译的结果和后台编译结果的哪一个类进行合并。后台相同类名前使用了partial关键字，这样由XAML解析成的类和C#代码文件里定义的部分就合二为一。正是这种机制我们才可以把类的逻辑代码留在.CS文件里，用C#语言来实现，而把哪些与声明及布局UI元素相关的代码分离出去，实现UI与逻辑分离。而后者使用XAML和XAML编辑工具就能轻松搞定。
x:Name
它表示处理 XAML 中定义的对象元素后，为运行时代码中存在的实例指定运行时对象名称。在实际项目中，控件元素和资源的命名规则是只在需要的时候对控件和资源进行命名操作，绝大多数XAML元素都没有明确命名，因为它们不需要被其他代码所访问，其存在仅仅是提供静态作用。
x:Type
一般情况下，我们在编程中操作数据类型的实例或者实例的引用，但有的时候我们也需要用到数据类型本身。
x:Type顾名思义应该是一个数据类型的名称。当我们想在XAML中表达某一数据类型就需要用到x:Type标记扩展。比如某个类的一个属性，它的值要求的是一个数据类型，当我们在XAML中为这个属性赋值是就需要用到x:Type。
x:Null
在XAML里面表示空值就是x:Null。多数时候我们不需要为属性赋一个Null值，但如果一个属性开始就有默认值而我们有不需要这个默认值就需要用到Null值了。在WPF中，Style是按照一个特定的审美规格设置控件的各个属性，程序员可以为控件逐个设置style，也可以指定一个style目标控件类型，一旦指定了目标类型，所有的这类控件都将使用这个style----除非你显示的将某个实例的Style设置为Null。
x:Code
x:Code的作用是可以在XAML文档中可以编写后置的C#后台逻辑代码，当然这样写的问题破坏了XAML和CS文件的分工，使得代码不容易维护，不宜调试。
（类似于ASP.NET在ASPX文件中嵌入服务器代码。）
同时WPF中对它的使用有严格的限制，并不建议使用。

- XAML对象可以相互嵌套：

```
<Canvas>
    <Rectangle Canvas.Left="50" Canvas.Top="50"  Width="200"  Height="150"
RadiusX="10"  RadiusY="10"  Fill="Gold"/>
</Canvas>
```

- 树：
  - 逻辑树：WPF的各个方面
  - 可视树：将逻辑树打散，重新组合，暴露了视觉的实现细节
  - 树图：我们设计的内容

- WPF事件：
  - 路由事件：添加路由事件就是把相应的事件名称作为一个特性添加到某个元素中，比如，鼠标按下，然后自己定义函数

```
<Grid MouseLeftButtonDown="Click_Img">
    <Border CornerRadius="20" Width="80" Height="30" BorderBrush="Black"
BorderThickness="1">
        <Image Source="Images/back1.png" x:Name="ccPic"/>
    </Border>
</Grid>
```

  - 事件类型

（1）WPF事件类型

尽管每个元素都提供了许多事件，但最重要的事件通常包括如下5类：

1.**生命周期事件**：在元素被初始化、加载或卸载时发生这些事件。

2.**鼠标事件**：在鼠标进行使用时的动作引发这些事件。

3.**键盘事件**：伴随键盘上的键被按而引发的事件。

4.**手写笔**：主要针对在触控环境下使用手写笔引发的事件。

5.**多点触控事件**：一个或多个屏幕触控引发的事件。

（2）WPF生命周期事件

WPF元素从被创建以及直到被释放的整个生命周期中，会有很多阶段性的状态，由这些状态触发的事件统称为生命周期事件，通常可以利用这些事件进行初始化或者事务善后工作。

（3）WPF输入事件

当使用外设和计算机交互时引发，典型情况是鼠标、键盘、手写笔和触屏对计算机输入时触发。这类事件都可以通过继承自InputEventArgs的自定义事件参数类传递额外的信息。不同的输入设备其输入事件的参数类派生后也不一样，但是它们都有共性，都能够获取该设备的一些状态数据，如键盘事件中的按键值、按键状态等等。

名称	路由类型	说明
PreviewKeyDown	隧道	当按下一个键时发生
KeyDown	冒泡	同上
PreviewTextInput	隧道	当按键完成并且元素正在接收文本输入时发生。对于那些不会产生文本输入的功能键，不会引发该事件。
PreviewKeyUp	隧道	当按键后松开时触发
KeyUp	冒泡	同上

参数属性	说明
e.Key	按下的普通字符或数字键，它的值是一个枚举类型，可以用来判断或者处理一般按键。
e. KeyboardDevice	键盘上所有的键的集合，通常用于处理功能键

```
<TextBox x:Name="txtInput" Margin="5" Width="150" PreviewKeyDown="KeyEvent"
KeyDown="KeyEvent" />

private void KeyEvent(object sender, KeyEventArgs e)
{
    string msg = "事件:" + e.RoutedEvent.ToString() + ",Key:" + e.Key.ToString();
    lstOutput.Items.Add(msg);
    if (e.Key == Key.A && e.KeyboardDevice.IsKeyDown(Key.LeftCtrl))
    {
         MessageBox.Show("Ctrl+A组合键被按下！");
    }
}
```

（2）焦点

在Windows世界中，用户每次只能用一个控件。当前接收用户按键的控件是具有焦点的控件，通常获得焦点的控件的外观和其他控件略有差别。在 WPF 中，有两个与焦点有关的主要概念：键盘焦点和逻辑焦点。 键盘焦点指接收键盘输入的元素，而逻辑焦点指焦点范围中具有焦点的元素。

键盘焦点指当前正在接收键盘输入的元素。 在整个窗体中只能有一个具有键盘焦点的元素。 在 WPF 中，具有键盘焦点的元素会将 IsKeyboardFocused 设置为 true。 Keyboard 类的静态属性 FocusedElement 获取当前具有键盘焦点的元素。

为了使元素能够获取键盘焦点，基元素的 Focusable 和 IsVisible 属性必须设置为 true。 有些类（如 Panel 基类）默认情况下将 Focusable 设置为 false；因此，如果您希望此类元素能够获取键盘焦点，必须将 Focusable 设置为 true。

可以通过用户与 UI 交互（例如，按 Tab 键定位到某个元素或者在某些元素上单击鼠标）来获取键盘焦点。 还可以通过使用 Keyboard 类的 Focus 方法，以编程方式获取键盘焦点。 Focus 方法尝试将键盘焦点给予指定的元素。 返回的元素是具有键盘焦点的元素，如果有旧的或新的焦点对象阻止请求，则具有键盘焦点的元素可能不是所请求的元素。