**非常好的学习网站：** [**http://tech.ddvip.com/**](http://tech.ddvip.com/%20)

**高手：** [**http://www.cnblogs.com/gnielee/archive/2010/04/20/wpf4-datagrid-control-adv2.html**](http://www.cnblogs.com/gnielee/archive/2010/04/20/wpf4-datagrid-control-adv2.html)

[**http://tech.ddvip.com/2012-10/1350027629183372\_4.html**](http://tech.ddvip.com/2012-10/1350027629183372_4.html)

[**http://tech.ddvip.com/2013-02/1361980516191131\_2.html**](http://tech.ddvip.com/2013-02/1361980516191131_2.html)

[**http://tech.ddvip.com/2012-10/1350026359183369.html**](http://tech.ddvip.com/2012-10/1350026359183369.html)

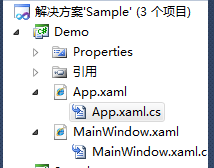
**WPF Source Code:**

[**https://onedrive.live.com/?cid=c75f4e27adfe5bbc&id=C75F4E27ADFE5BBC!937&authkey=!**](https://onedrive.live.com/?cid=c75f4e27adfe5bbc&id=C75F4E27ADFE5BBC!937&authkey=!)

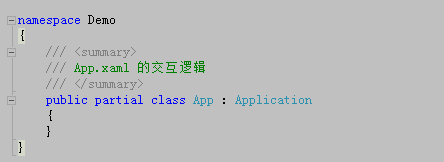
**8天入门wpf—— 第一天 基础概念介绍**

一：App环境承载

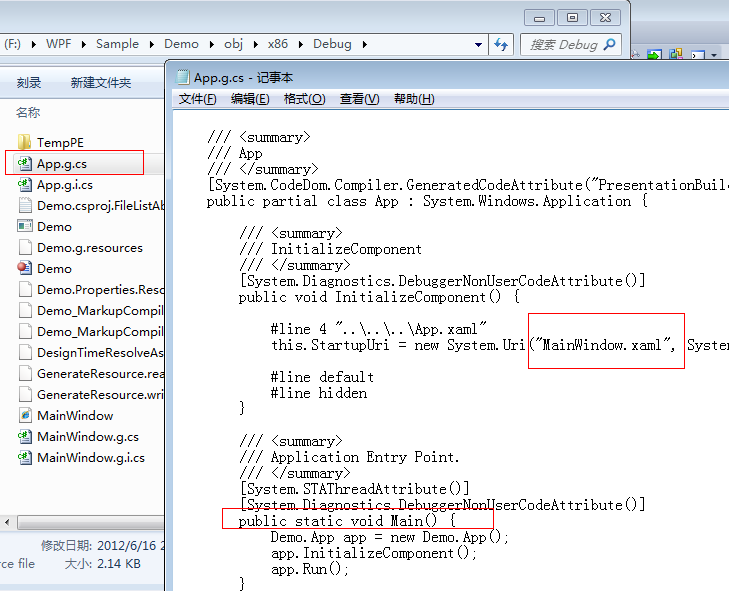
    我们都知道，console和winform程序的入口函数都是main，wpf同样也不例外，好了，我们新建一个wpf的程序，vs自动给我们生成了一个MainWindow.xaml和App.xaml文件。



微软官方说wpf程序是从Application开始的，既然是开始总有个入口点吧，奇怪的是我们并没有发现Main函数，程序又是如何Run起来的呢？



其 实，wpf为了简化我们的工作，把一些机械性的代码透明了，那么我们如何找到这个Main函数呢？很简单，我们编译一下程序，发现App.xaml最后生 成了App.g.cs的部分类，并且发现StartupUri是MainWindow.xaml，也就是说程序一运行，MainWindow.xaml将 会启动。

[](http://img.ddvip.com/2012/1012/201210120319037279.png)

二：Wpf中Application的生命周期

   我们知道webform中的Global文件定义了一个应用程序的全局生命周期，或许有人问，生命周期能够干些什么，其实干的事情可多着呢，比如我们可以做一些身份验证，或者一些信息的初始化，那么wpf中到底有哪些对应的方法和事件呢？

1：OnStartup方法    =>   Startup 事件

    这个就见名识意了，也就是上面一幅图中的app.Run()的时候触发。

2: OnSessionEnding方法 => SessionEnding 事件

    系统关机前调用。

3：OnExit方法 => Exit事件

    应用程序关闭前调用。

4：OnActivated方法 =>  Activated 事件

    应用程序获得焦点的时候触发。

5：OnDeactivated方法 => DeActivated事件

    应用程序失去焦点的时候触发。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | using System;   using System.Collections.Generic;   using System.Configuration;   using System.Data;   using System.Linq;   using System.Windows;     namespace Demo   {       /// <summary>       /// App.xaml 的交互逻辑       /// </summary>       public partial class App : Application       {           protected override void OnActivated(EventArgs e)           {               base.OnActivated(e);                 //TODO  your code           }             protected override void OnDeactivated(EventArgs e)           {               base.OnDeactivated(e);                 //TODO  your code           }             protected override void OnExit(ExitEventArgs e)           {               base.OnExit(e);                 //TODO  your code           }             protected override void OnStartup(StartupEventArgs e)           {               base.OnStartup(e);                 //TODO  your code           }             protected override void OnSessionEnding(SessionEndingCancelEventArgs e)           {               base.OnSessionEnding(e);                 //TODO  your code           }       }   } |

三：全局异常获取

    在webform中的Global文件中有一个Application\_Error方法，专门用来捕获整个应用程序的异常，以至于不会出现“黄白页”给用 户，以此来提高系统的健壮性和安全性，那么wpf中也有类似的方法吗？当然，wpf跟webform神似，他有的我也有，这里是一个 DispatcherUnhandledException事件，然后我们在OnStartup注册一下就Ok了。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | namespace Demo   {       /// <summary>       /// App.xaml 的交互逻辑       /// </summary>       public partial class App : Application       {           protected override void OnStartup(StartupEventArgs e)           {               base.OnStartup(e);                 //注册Application\_Error               this.DispatcherUnhandledException += new DispatcherUnhandledExceptionEventHandler(App\_DispatcherUnhandledException);             }             //异常处理逻辑           void App\_DispatcherUnhandledException(object sender, System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)           {               //处理完后，我们需要将Handler=true表示已此异常已处理过               e.Handled = true;           }       }   } |

好，下面我们做了示例：

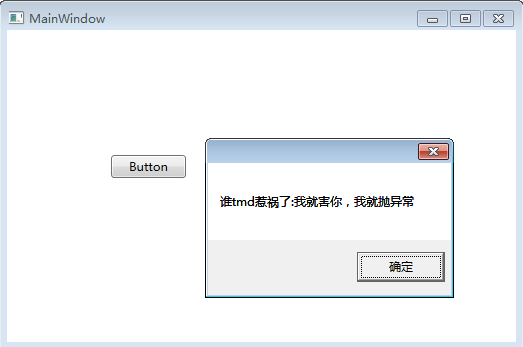
首先我们拖一个button，事件处理中故意抛出异常。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | namespace Demo   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();           }             private void button1\_Click(object sender, RoutedEventArgs e)           {               throw new Exception("我就害你，我就抛异常");           }       }   } |

然后我们在Application\_Error中进行处理，当然实际应用中应该是记一些log日志。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | //异常处理逻辑           void App\_DispatcherUnhandledException(object sender, System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)           {               MessageBox.Show("谁tmd惹祸了:" + e.Exception.Message);                 //处理完后，我们需要将Handler=true表示已此异常已处理过               e.Handled = true;           } |

最后看一下效果，注意，我们的程序并没有崩溃。



# 8天入门wpf—— 第二天 xaml详解

首先我们还是新建一个空项目，看一下VS给我们默认生成的xaml结构。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | <Window x:Class="WpfApplication1.MainWindow"          xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"          xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"          Title="MainWindow" Height="350" Width="525">      <Grid>        </Grid>  </Window> |

然后我们来对比一下webform中的page默认生成页面

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication1.WebForm1" %>     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>">   <html xmlns="<http://www.w3.org/1999/xhtml>">   <head runat="server">       <title></title>   </head>   <body>       <form id="form1" runat="server">       <div>       </div>       </form>   </body>   </html> |

我们发现xaml很像xml结构，是的，它是一种遵循xml规范的界面描述语言。

一：xaml简述

  首先我默认大家都是熟悉webform的开发者。

 1：x:Class

      这个标记在webform中有对应的标记吗？有的，也就是这里的CodeBehind。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <span style="color:#0000FF;"><</span><span style="color:#800000;">%@ Page </span><span style="color:#FF0000;">Language</span><span style="color:#0000FF;">="C#"</span><span style="color:#FF0000;"> AutoEventWireup</span><span style="color:#0000FF;">="true"</span><span style="color:#FF0000;"> CodeBehind</span><span style="color:#0000FF;">="WebForm1.aspx.cs"</span><span style="color:#FF0000;"> Inherits</span><span style="color:#0000FF;">="WebApplication1.WebForm1"</span><span style="color:#FF0000;"> %</span><span style="color:#0000FF;">></span> |

 2：xmlns

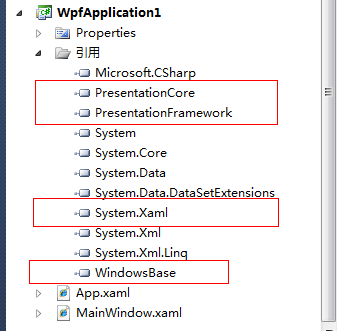
     这个在webform中也有对应标记吗？首先我们要知道xmlns是干嘛的？哦，导入命名空间用的，那我们马上就想到webform中的对应标记import。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <span style="color:#0000FF;"><</span><span style="color:#800000;">%@ Import </span><span style="color:#FF0000;">Namespace</span><span style="color:#0000FF;">="System.IO"</span><span style="color:#FF0000;"> %</span><span style="color:#0000FF;">></span> |

那么下一个问题就是两者有什么不同吗？我们知道webform中导入命名空间需要一个一个的导入，4个命名空间就要写4个import，而xaml可以做到多个命名空间做为一个导入。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>" |

其实也就导入了如下4个wpf开发必备的dll，这个命名空间也是xaml中默认的命名空间。



3：xmlns:x

   如果我们需要导入一些自定义的命名空间，那么我们就需要加上用“**: + 自定义名称**”的形式，这里微软导入了一个自定义的命名空间。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>" |

下面我们也来导入一个命名空间，实际开发中我们当然我们不会做成url的形式，这里我就取名为sys前缀

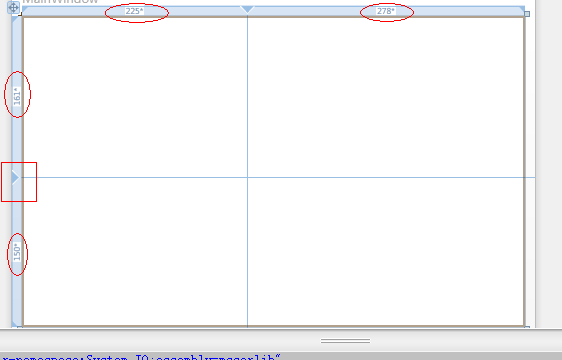
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | <Window x:Class="WpfApplication1.MainWindow"          xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"          xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"          xmlns:sys="clr-namespace:System.IO;assembly=mscorlib"          Title="MainWindow" Height="350" Width="525">      <Grid>      </Grid>  </Window> |

4：Grid

   我们都知道在n年前的html网页布局中都采用table的形式，table嵌套table，table跨行跨列等手段构建了漂亮的网页，这种排版思路也应用到了wpf中。

<1>：划分位置

     我们画个两行两列的界面布局，这里我们只要将”鼠标“放在”红色方框“中，就会出现小三角，我们点击就可生成一个分割线，红色小圈圈标记着“行列”的分割比列。

[](http://img.ddvip.com/2012/1012/201210120324170104.png)

然后我们看一下生成的代码

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <Window x:Class="WpfApplication1.MainWindow"          xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"          xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"          xmlns:sys="clr-namespace:System.IO;assembly=mscorlib"          Title="MainWindow" Height="350" Width="525">      <Grid>          <Grid.RowDefinitions>              <RowDefinition Height="161\*" />              <RowDefinition Height="150\*" />          </Grid.RowDefinitions>          <Grid.ColumnDefinitions>              <ColumnDefinition Width="225\*" />              <ColumnDefinition Width="278\*" />          </Grid.ColumnDefinitions>      </Grid>  </Window> |

我们奇怪的发现为什么宽度有“\*”号，这里要解释一下wpf中对“宽高度”的设置有三个形式。

①：绝对尺寸         <RowDefinition Height="161" />

②：自动尺寸         <RowDefinition Height="auto" />

③：按比例尺寸      <RowDefinition Height="161\*" />

那我们就有疑问了，到底161\* 是多少呢？计算规则如下：

我们这里的窗体Height=350。

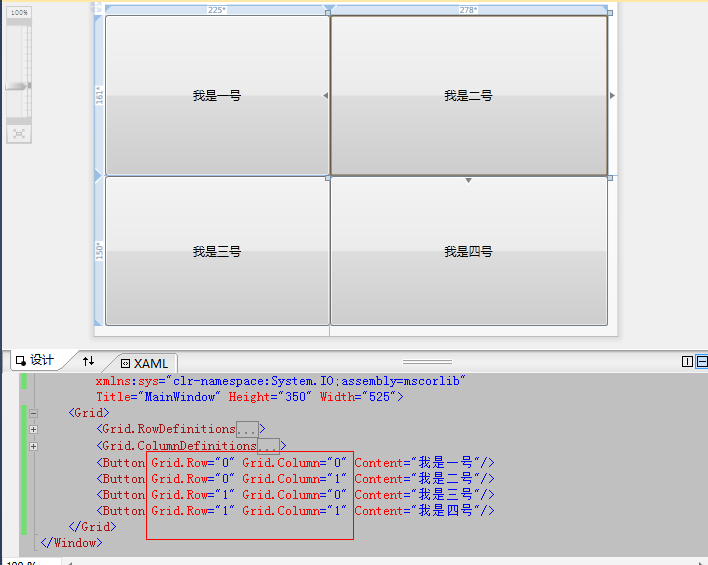
第一行高度为: 350/(161+150)\*161

第二行高度为：350(161+150)\*150

<2>: UI元素布局

  ①：UI元素对号入座

          很简单，我们只要在button的Grid属性上设置button应该放在哪一个单元格即可。

[](http://img.ddvip.com/2012/1012/201210120324171790.png)

  ②：UI元素跨行跨列

[](http://img.ddvip.com/2012/1012/201210120324204431.png)

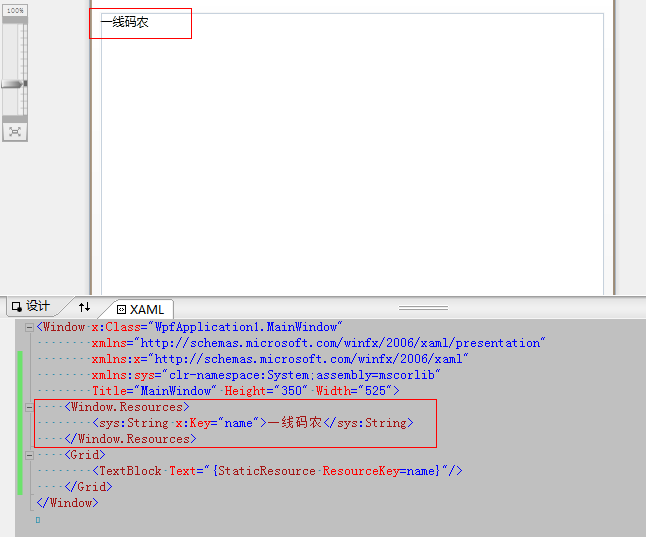
二：xaml中扩展标记

   扩展标记分为两种：wpf级别和xaml级别。

<1> wpf级别扩展标记

①： StaticResource

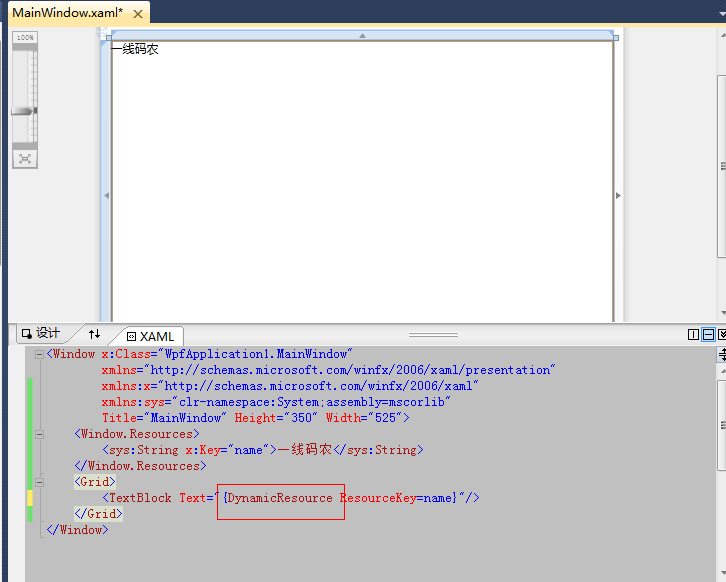
      用于获取资源的值，值获取在xaml编译的时候完成，什么意思呢？先举个例子。

[](http://img.ddvip.com/2012/1012/201210120324206538.png)

首先，我们发现有一个window.Resources，这东西我们可以认为是在MainWindow类中定义的全局变量，这里我就定义个name=“一线码农”的变量，那么textblock获取变量的方式就可以通过StaticResource。

②：DynamicResource

      跟StaticResource唯一不同的是，它是在运行时获取的，如果大家知道C#里面的dynamic关键字，我想就不用解释了，上代码。

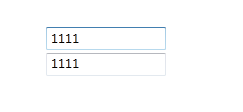
[](http://img.ddvip.com/2012/1012/201210120324218361.png)

③：Binding

    还是在webform中找一下关键字吧，相当于webform中的Eval，上代码说话。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11 | <Window x:Class="WpfApplication1.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:sys="clr-namespace:System;assembly=mscorlib"           Title="MainWindow" Height="350" Width="525">       <Grid>           <TextBox Height="23" Margin="87,75,0,0" Name="textBox1"  Width="120" />           <TextBox Height="23" Margin="87,126,0,0" Name="textBox2"  Width="120"                    Text="{Binding ElementName=textBox1, Path=Text}" />       </Grid>   </Window> |

这里我把textbox2的text绑定到了textbox1的text上，最后的效果就是我在textbox1上输入，textbox2也会相应的变化，很有意思。



④：TemplateBinding

    这个被称为模板绑定，可以把对象的属性和模板的属性绑定起来，详细的介绍放在后续文章中。

<2>xaml级别扩展标记

①  x:Type

  将模板或者样式指定在哪一种对象上时需要用type指定。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <Window x:Class="WpfApplication1.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:sys="clr-namespace:System;assembly=mscorlib"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style TargetType="{x:Type TextBox}">               <Setter Property="Background" Value="Red"/>           </Style>       </Window.Resources>       <Grid>           <TextBox Height="23"                    Margin="87,75,0,0" Name="textBox1"  Width="120" />       </Grid>   </Window> |

如这里我定义的css样式，将background=red指定到textbox控件上。



②：x:Static

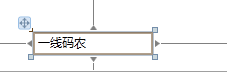
   主要用于在xaml中获取某个对象的静态值，上代码说话。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | namespace WpfApplication1  {      /// <summary>      /// MainWindow.xaml 的交互逻辑      /// </summary>      public partial class MainWindow : Window      {          public static string name = "一线码农";            public MainWindow()          {              InitializeComponent();          }      }  } |

xaml代码:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10 | <Window x:Class="WpfApplication1.MainWindow"          xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"          xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"          xmlns:local="clr-namespace:WpfApplication1"          Title="MainWindow" Height="350" Width="525">      <Grid>          <TextBox Height="23"  Text="{x:Static local:MainWindow.name}"                   Margin="87,75,0,0" Name="textBox1"  Width="120" />      </Grid>  </Window> |

最后效果：



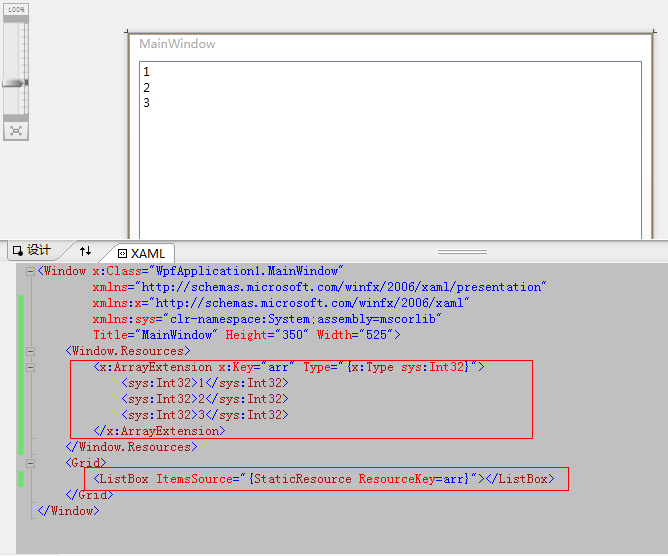
③：x:null

   这个就比较简单了，xaml中某某控件设为null就靠它了。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4 | <Grid>      <TextBox Height="23"  Text="{x:Null}"               Margin="87,75,0,0" Name="textBox1"  Width="120" />  </Grid> |

④：x:Array

 这个主要就是在xaml中创建数组，还是举个例子。

[](http://img.ddvip.com/2012/1012/201210120324224689.png)

# 8天入门wpf—— 第三天 样式

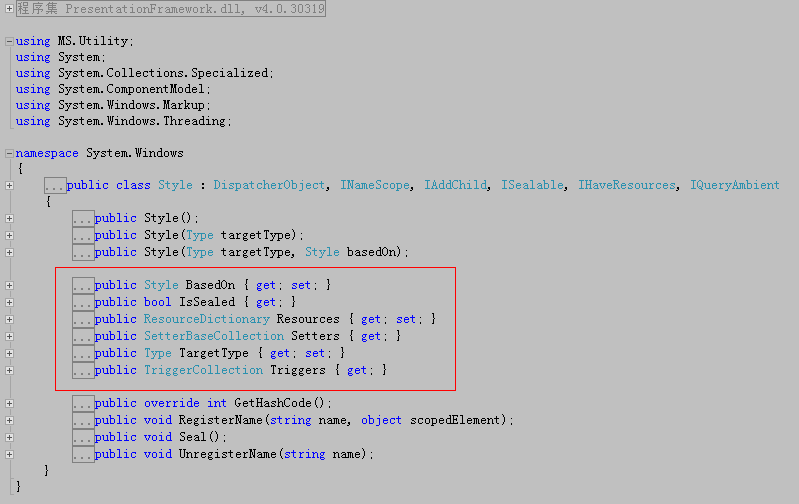
说起样式，大家第一反应肯定是css，好的，先上一段代码。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | html{border:0;}   ul,form{margin:0; padding:0}   body,div,th,td,li,dd,span,p,a{font-size:12px; font-family:Verdana,Arial,"宋体";color:#575757;}   h3,input{font-size:12px; font-family:Verdana,Arial,"宋体";color:#4465a2;}     body {       /\*background-color:#eaeaea;\*/       /\*e5e5e5\*/       /\*BACKGROUND: url(../images/header\_bg.jpg) #fff repeat-x;\*/       BACKGROUND: url(../images/color\_1.png) #fff repeat-x 0px -233px;       margin:0px;       padding:0px;   }     ul{list-style:none;}   h1,h2,h4,h5,h6{ font-size:14px; color:#333;}   img{border:0;}   a {color:#333333;text-decoration:none;}   a:hover{color:#ff0000;text-decoration:underline;} |

我们知道css实现了内容与样式的分离，既然wpf跟webform非常类似，那么肯定也有一套能够实现css的功能，是的。这就是wpf的style。

一：Style类

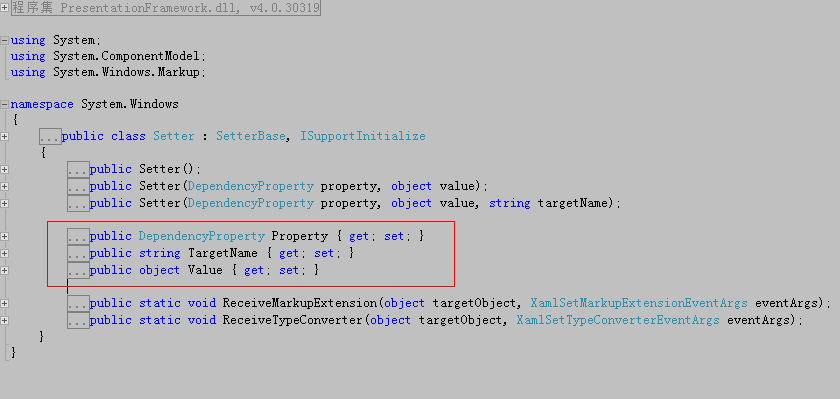
    首先我们看看Style里面有哪些东西，在vs里面我们可以通过按F12查看类的定义。

[](http://img.ddvip.com/2012/1012/201210120333188070.png)

下面我们一一解读下:

1：Setters

   从上图我们知道Setters的类型是SetterBaseCollection，可以看得出是一个存放SetterBase的集合，SetterBase派生出了两个类Setter和EventSetter，下面我们看看Setter类的定义。

[](http://img.ddvip.com/2012/1012/201210120333190242.png)

这里我们看到了两个非常重要KV属性Property和Value，我们拿css找找对应关系。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <span style="color:#800000;">html</span>{<span style="color:#FF0000;">border</span>:<span style="color:#0000FF;">0</span>;} |

html    => Style.TargetType

border =>   Property

0        =>   Value

估计大家想迫不及待的试一试，好了，我先做一个简单的demo。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <Window x:Class="WpfApplication1.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:sys="clr-namespace:System;assembly=mscorlib"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style TargetType="Button">               <Setter  Property="Background" Value="Pink"/>               <Setter Property="FontSize" Value="22"/>           </Style>       </Window.Resources>       <Grid>           <Button Content="一线码农"/>       </Grid>   </Window> |

最后效果：

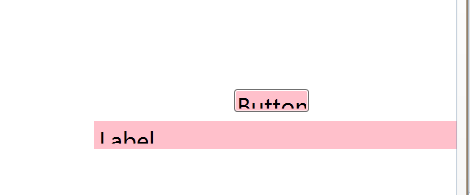


仔细看看，是不是找到了css的感觉，有人肯定要问，这不就是标签选择器吗？能不能做成“id选择器”，当然可以，我们只需要给style取一个名字，然后在控件上引用一下就ok了。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="mystyle" TargetType="Button">               <Setter  Property="Background" Value="Pink"/>               <Setter Property="FontSize" Value="22"/>           </Style>       </Window.Resources>       <Grid>           <Button Style="{StaticResource ResourceKey=mystyle}" Content="一线码农"/>       </Grid>   </Window> |

现 在我们添加一个label，如果我们也需要同样的“背景色”和“字体”，那么我们是否要重新写一个label的样式吗？答案肯定是否定的，聪明的你肯定会 想到”基类“。我们发现label和button都是继承自ContentControl,都属于内容控件，那么何不在TargetType中定义为 ContentControl不就ok了吗？

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <Window x:Class="WpfApplication1.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:sys="clr-namespace:System;assembly=mscorlib"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="mystyle" TargetType="ContentControl">               <Setter  Property="Background" Value="Pink"/>               <Setter Property="FontSize" Value="22"/>           </Style>       </Window.Resources>       <Grid>           <Button Style="{StaticResource ResourceKey=mystyle}"                   Content="Button" Height="23" Margin="132,99,0,0" Name="button1"  Width="75" />           <Label Style="{StaticResource ResourceKey=mystyle}"                  Content="Label" Height="28" Margin="140,168,0,0" Name="label1"  />       </Grid>   </Window> |



2：TargetType

  我们在说Setter的时候也提到了，其实TargetType也就是将样式施加到某一个对象上，具体的也没什么好说的。

3：BaseOn

 我们知道css具有“继承和覆盖”的特性，同样我们的wpf中也是具有的。

<1>：继承

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="baseStyle" TargetType="Button">               <Setter Property="FontSize" Value="22"/>           </Style>           <Style x:Key="childStyle" TargetType="Button"                  BasedOn="{StaticResource ResourceKey=baseStyle}">               <Setter  Property="Background" Value="Pink"/>           </Style>       </Window.Resources>       <Grid>           <Button Style="{StaticResource ResourceKey=childStyle}" Content="一线码农"/>       </Grid>   </Window> |

效果：

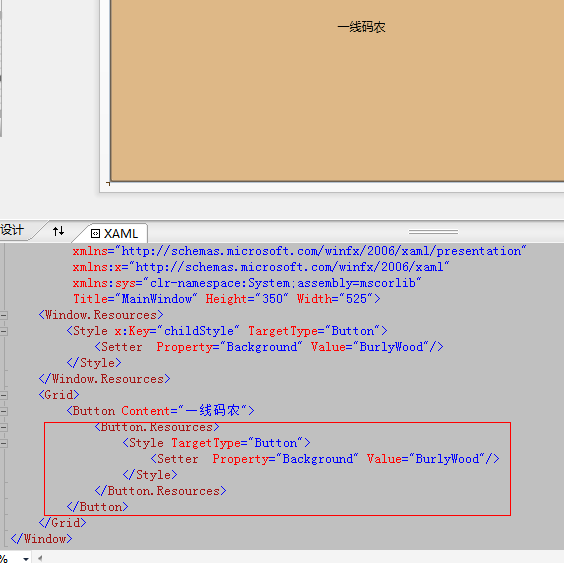


从上例中，我们看到childStyle继承到了baseStyle中的fontSize，最终的效果也是我们期望看到的。

<2>：覆盖

     我们知道css遵循“就近原则”。

①：“行内”覆盖“嵌入”，“嵌入”覆盖“外部”

[](http://img.ddvip.com/2012/1012/201210120333195860.png)

我们可以清楚的看到，行内样式覆盖了嵌入样式。

②：同级别遵循“就近”。

[](http://img.ddvip.com/2012/1012/201210120333198074.png)

从button的颜色上看，我们可以获知Pink已经被BurlyWood覆盖。

4：Triggers

   顾名思义，是触发器的意思，我们可以认为是wpf在style中注入了一些很简单，很sb的js代码。

wpf中有5种trigger，都是继承自TriggerBase类。

<1> Trigger,MuliTrigger

   我们知道js是事件驱动机制的，比如触发mouseover，mouseout，click等事件来满足我们要处理的逻辑，那么wpf在不用写C#代码的情况下用trigger就能够简单的满足这些事件处理。

下面举个例子

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="childStyle" TargetType="Button">               <Setter  Property="Background" Value="BurlyWood"/>               <Style.Triggers>                   <!-- 当IsMouseOver的时候，Button颜色变成粉色 -->                   <Trigger Property="IsMouseOver" Value="True">                       <Setter  Property="Background" Value="Pink"/>                   </Trigger>               </Style.Triggers>           </Style>       </Window.Resources>       <Grid>           <Button Style="{StaticResource ResourceKey=childStyle}" Content="一线码农">           </Button>       </Grid>   </Window> |

最后的效果就是当isMouseOver=true的情况下，button的Background变成Pink。

然而trigger只能满足在单一的条件下触发，那么我想在多个条件同时满足的情况下才能触发有没有办法做到呢？刚好MuliTrigger就可以帮你实现。

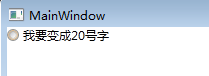
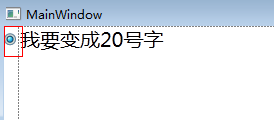
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="childStyle" TargetType="Button">               <Setter  Property="Background" Value="BurlyWood"/>               <Style.Triggers>                   <MultiTrigger>                       <MultiTrigger.Conditions>                           <Condition Property="IsMouseOver" Value="True"></Condition>                           <Condition Property="IsPressed" Value="True"></Condition>                       </MultiTrigger.Conditions>                       <Setter  Property="Background" Value="Pink"/>                   </MultiTrigger>               </Style.Triggers>           </Style>       </Window.Resources>       <Grid>           <Button  Style="{StaticResource ResourceKey=childStyle}" Content="一线码农">           </Button>       </Grid>   </Window> |

<2>DataTrigger,MultiDataTrigger

    这个跟上面的Trigger有什么不同呢？其实也就是DataTrigger多了一个Binding的属性，当然它的实际应用也是最广泛的。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="childStyle" TargetType="Control">               <Setter  Property="Background" Value="BurlyWood"/>               <Style.Triggers>                   <!-- 绑定当前的radio单选框,如果按钮选中,触发字体设置 -->                   <DataTrigger Binding="{Binding ElementName=radio, Path=IsChecked}" Value="True">                       <Setter Property="FontSize" Value="20"/>                   </DataTrigger>               </Style.Triggers>           </Style>       </Window.Resources>       <Grid>           <RadioButton Style="{StaticResource ResourceKey=childStyle}"                        Name="radio" Content="我要变成20号字"></RadioButton>       </Grid>   </Window> |

效果：

          =>            

当我们选中radio的时候，字体变大，同样MultiDataTrigger这个多条件的使用道理也是一样的，这里就不介绍了。

<3>EventTrigger

  这个trigger与动画有关，目前项目中还没接触到，留给大家自己研究研究。

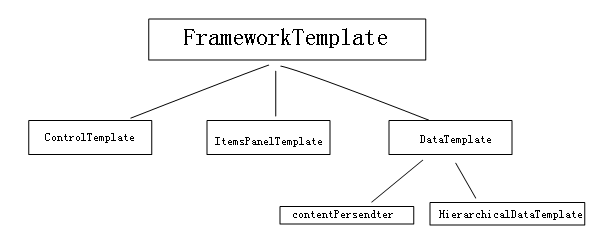
5：IsSealed

   用于标记style是只读的，类似我们在C#中的Seal关键字，来达到不允许让继承类使用，wpf使用seal常常在C#代码里面控制，在xaml中我们是找不到的，有兴趣的话，大家自己研究研究。

# 8天入门wpf—— 第四天 模板

今天说下wpf中的模板，前面一篇中我们讲到了style，但是style所能做的仅仅是在现有控件的基础上进行修修补补，但是如果我们想彻底颠覆控件样式，那么我们就必须使用这一篇所说的模板。

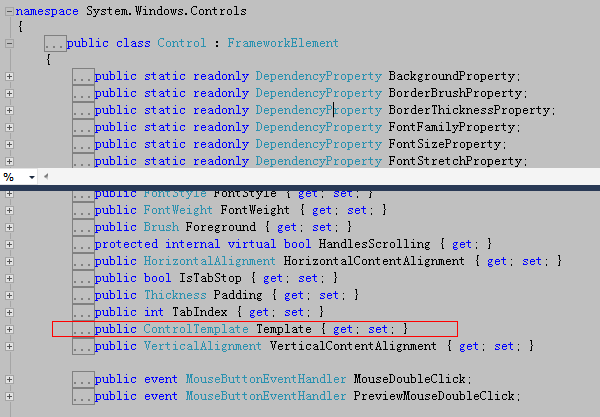
  老外写书都喜欢在篇头搞一个类图，方便我们宏观认识，这里我也上一个。

[](http://img.ddvip.com/2012/1012/201210120339594542.png)

一：控件模板

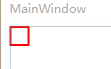
1：ControlTemplate

  我们知道wpf的控件都是继承自Control，在Control类中有一个Template属性，类型就是ControlTemplate。

[](http://img.ddvip.com/2012/1012/201210120339596113.png)

那么利用这个ControlTemplate就可以彻底的颠覆控件的默认外观，这里我把一个checkbox变成一个小矩形，蛮有意思的。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ControlTemplate x:Key="rect" TargetType="{x:Type CheckBox}">               <StackPanel>                   <Rectangle Name="breakRectangle" Stroke="Red" StrokeThickness="2" Width="20" Height="20">                       <Rectangle.Fill>                           <SolidColorBrush Color="White"/>                       </Rectangle.Fill>                   </Rectangle>               </StackPanel>           </ControlTemplate>       </Window.Resources>       <Canvas>           <CheckBox Template="{StaticResource ResourceKey=rect}" Content="我是CheckBox"/>       </Canvas>   </Window> |



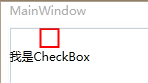
确实，我们干了一件漂亮的事情，把checkbox变成了“小矩形”，但是我们发现了一个小问题，为什么我的Content=“xxx”没有显示到模板上？

很简单，我们已经重定义了控件模板，默认模板将会被覆盖...

2：ContentPresenter

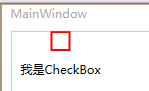
    幸好，wpf给我们提供了一个ContentPresenter，它的作用就是把原有模板的属性原封不动的投放到自定义模板中。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ControlTemplate x:Key="rect" TargetType="{x:Type CheckBox}">               <StackPanel>                   <Rectangle Name="breakRectangle" Stroke="Red" StrokeThickness="2" Width="20" Height="20">                       <Rectangle.Fill>                           <SolidColorBrush Color="White"/>                       </Rectangle.Fill>                   </Rectangle>                   <ContentPresenter/>               </StackPanel>           </ControlTemplate>       </Window.Resources>       <Canvas>           <CheckBox Template="{StaticResource ResourceKey=rect}" Content="我是CheckBox"/>       </Canvas>   </Window> |



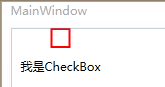
当然你也可以玩一些小技巧，比如我想在"矩形“和”文字”中间设置边距，那么我们可以设置ContentPresenter的margin。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ControlTemplate x:Key="rect" TargetType="{x:Type CheckBox}">               <StackPanel>                   <Rectangle Name="breakRectangle" Stroke="Red" StrokeThickness="2" Width="20" Height="20">                       <Rectangle.Fill>                           <SolidColorBrush Color="White"/>                       </Rectangle.Fill>                   </Rectangle>                   <ContentPresenter Margin="10" />               </StackPanel>           </ControlTemplate>       </Window.Resources>       <Canvas>           <CheckBox Template="{StaticResource ResourceKey=rect}"  Content="我是CheckBox"/>       </Canvas>   </Window> |



如果你够聪明，你会发现我设置的margin是一个非常呆板的事情，意思就是说能不能根据具体控件灵活控制margin呢？答案肯定是没问题的，因为我们记得一个控件可以绑定到另一个控件上，比如这里我将模板中的Margin绑定到原控件中的Padding上去。

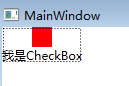
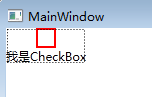
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ControlTemplate x:Key="rect" TargetType="{x:Type CheckBox}">               <StackPanel>                   <Rectangle Name="breakRectangle" Stroke="Red" StrokeThickness="2" Width="20" Height="20">                       <Rectangle.Fill>                           <SolidColorBrush Color="White"/>                       </Rectangle.Fill>                   </Rectangle>                   <ContentPresenter Margin="{TemplateBinding Padding}" />               </StackPanel>           </ControlTemplate>       </Window.Resources>       <Canvas>           <CheckBox Template="{StaticResource ResourceKey=rect}"  Content="我是CheckBox" Padding="10"/>       </Canvas>   </Window> |



3：Trigger

    我们知道style里面也是有trigger的，废话不多说，上代码说话。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ControlTemplate x:Key="rect" TargetType="{x:Type CheckBox}">               <ControlTemplate.Resources>                   <SolidColorBrush x:Key="redBrush" Color="Red"/>               </ControlTemplate.Resources>               <StackPanel>                   <Rectangle Name="breakRectangle" Stroke="Red" StrokeThickness="2" Width="20" Height="20">                       <Rectangle.Fill>                           <SolidColorBrush Color="White"/>                       </Rectangle.Fill>                   </Rectangle>                   <ContentPresenter/>               </StackPanel>               <ControlTemplate.Triggers>                   <Trigger Property="IsChecked" Value="True">                       <Setter TargetName="breakRectangle" Property="Fill" Value="{StaticResource ResourceKey=redBrush}">                       </Setter>                   </Trigger>               </ControlTemplate.Triggers>           </ControlTemplate>       </Window.Resources>       <Canvas>           <CheckBox Template="{StaticResource ResourceKey=rect}"  Content="我是CheckBox"/>       </Canvas>   </Window> |

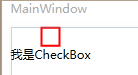


最后形成的效果就是当checkbox选中时为实心框，不选中为空心框。

4：与Style混搭

  可能刚才我也说了，style只能在原有的控件基础上修修补补，如果我们让Style修补Control控件的Template属性时，此时我们是不是就可以实现ControlTemplate和Style的混搭呢？

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <Style x:Key="cbx" TargetType="{x:Type CheckBox}">               <Setter Property="Template">                   <Setter.Value>                       <ControlTemplate TargetType="{x:Type CheckBox}">                           <ControlTemplate.Resources>                               <SolidColorBrush x:Key="redBrush" Color="Red"/>                           </ControlTemplate.Resources>                           <StackPanel>                               <Rectangle Name="breakRectangle" Stroke="Red" StrokeThickness="2" Width="20" Height="20">                                   <Rectangle.Fill>                                       <SolidColorBrush Color="White"/>                                   </Rectangle.Fill>                               </Rectangle>                               <ContentPresenter/>                           </StackPanel>                           <ControlTemplate.Triggers>                               <Trigger Property="IsChecked" Value="True">                                   <Setter TargetName="breakRectangle" Property="Fill" Value="{StaticResource ResourceKey=redBrush}">                                   </Setter>                               </Trigger>                           </ControlTemplate.Triggers>                       </ControlTemplate>                   </Setter.Value>               </Setter>           </Style>         </Window.Resources>       <Canvas>           <CheckBox Style="{StaticResource ResourceKey=cbx}" Content="我是CheckBox"/>       </Canvas>   </Window> |



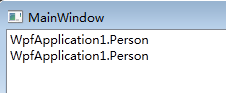
二：数据模板

  现在我们已经知道“控件模板”是用于改变控件外观，那么“数据模板”顾名思义就是控制数据的显示方式，下面做个demo让person绑定到listbox上。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | namespace WpfApplication1   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public static string name = "一线码农";             public MainWindow()           {               InitializeComponent();           }       }         public class PersonList : ObservableCollection<Person>       {           public PersonList()           {               this.Add(new Person() { Name = "一线码农", Age = 24, Address = "上海" });               this.Add(new Person() { Name = "小师妹", Age = 20, Address = "上海" });           }       }         public class Person       {           public string Name { get; set; }             public int Age { get; set; }             public string Address { get; set; }       }   } |

xaml：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"           xmlns:src="clr-namespace:WpfApplication1"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ObjectDataProvider x:Key="personList" ObjectType="{x:Type src:PersonList}"/>       </Window.Resources>       <Grid>           <ListBox ItemsSource="{Binding Source={StaticResource ResourceKey=personList}}"></ListBox>       </Grid>   </Window> |



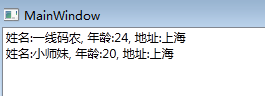
最后我们发现，listbox中并没有呈现我们需要的数据，只是呈现了当前类的ToString()方法，很简单，因为我们绑定的不是简单的数据类型集合，而是多字段的复杂类型，更重要的是我们并没有告诉wpf该如何呈现person数据。

<1>重写Tostring()

   既然wpf在Render数据的时候呈现的是当前的ToString()形式，那下面我们来重写ToString()试试看。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class Person       {           public string Name { get; set; }             public int Age { get; set; }             public string Address { get; set; }             public override string ToString()           {               return string.Format("姓名:{0}, 年龄:{1}, 地址:{2}", Name, Age, Address);           }       } |

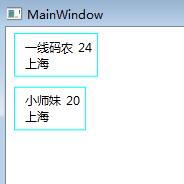
最后看看效果，如我们所愿，person信息已经呈现。



<2>DataTemplate重写

  或许有的人比较苛刻，他需要person是作为矩形一块一块的呈现，而不是这些简单的形式，那么此时我们就可以用DataTemplate来颠覆。

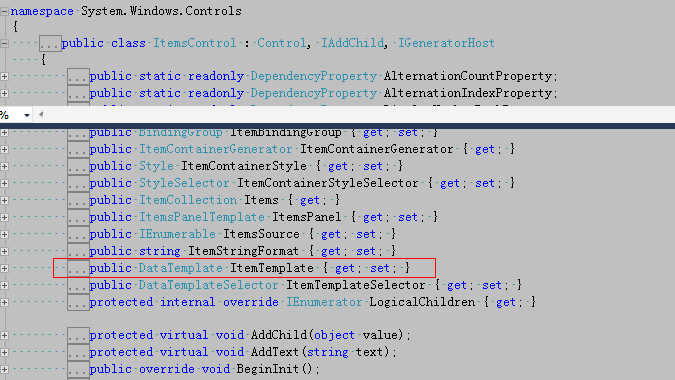
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"           xmlns:src="clr-namespace:WpfApplication1"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ObjectDataProvider x:Key="personList" ObjectType="{x:Type src:PersonList}"/>           <DataTemplate x:Key="rect">               <Border Name="border" BorderBrush="Aqua" BorderThickness="1" Padding="5" Margin="5">                   <StackPanel>                       <StackPanel Orientation="Horizontal">                           <TextBlock Text="{Binding Name}" Margin="5,0,0,0"/>                           <TextBlock Text="{Binding Age}" Margin="5,0,0,0"/>                       </StackPanel>                       <StackPanel Orientation="Horizontal">                           <TextBlock Text="{Binding Address}" Margin="5,0,0,0"/>                       </StackPanel>                   </StackPanel>               </Border>           </DataTemplate>       </Window.Resources>       <Grid>           <ListBox ItemsSource="{Binding Source={StaticResource ResourceKey=personList}}"                     ItemTemplate="{StaticResource ResourceKey=rect}"></ListBox>       </Grid>   </Window> |



哈哈，果然是以一块一块的形式展现，大功告成，当然这里的”触发器“和”style混搭“跟ConrolTemplate非常相似，我想应该不需要累赘了。

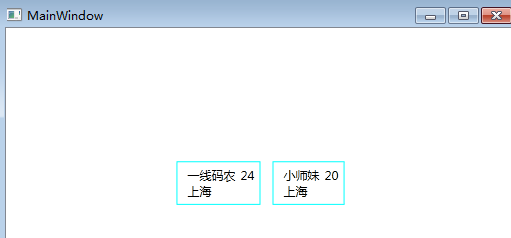
三： ItemsPanelTemplate

     在条目控件（ItemControl)里面，有一个属性叫ItemPanel,类型是ItemPanelTemplate。

[](http://img.ddvip.com/2012/1012/201210120340011856.png)

那 么ItemsPanelTemplate主要用来干什么的呢？首先我们要知道常见的条目控件有：ListBox,Menu,StatusBar，比如拿 ListBox来说，我们经过仔细研究，发现ItemBox的ItemPanel其实是一个VisualizingStackPanel，就是说 ListBox的每一项的排列方式是遵循StackPanel的原则，也就是从上到下的排列方式，比如”一线码农“和”小师妹“是按照竖行排列方式，好， 我现在的要求就是能够”横排“，该如何做到呢？

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"           xmlns:src="clr-namespace:WpfApplication1"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ObjectDataProvider x:Key="personList" ObjectType="{x:Type src:PersonList}"/>           <DataTemplate x:Key="rect">               <Border Name="border" BorderBrush="Aqua" BorderThickness="1" Padding="5" Margin="5">                   <StackPanel>                       <StackPanel Orientation="Horizontal">                           <TextBlock Text="{Binding Name}" Margin="5,0,0,0"/>                           <TextBlock Text="{Binding Age}" Margin="5,0,0,0"/>                       </StackPanel>                       <StackPanel Orientation="Horizontal">                           <TextBlock Text="{Binding Address}" Margin="5,0,0,0"/>                       </StackPanel>                   </StackPanel>               </Border>           </DataTemplate>           <ItemsPanelTemplate x:Key="items">               <StackPanel Orientation="Horizontal" VerticalAlignment="Center" HorizontalAlignment="Center"/>           </ItemsPanelTemplate>       </Window.Resources>       <Grid>           <ListBox ItemsSource="{Binding Source={StaticResource ResourceKey=personList}}"                     ItemTemplate="{StaticResource ResourceKey=rect}" ItemsPanel="{StaticResource ResourceKey=items}"></ListBox>       </Grid>   </Window> |

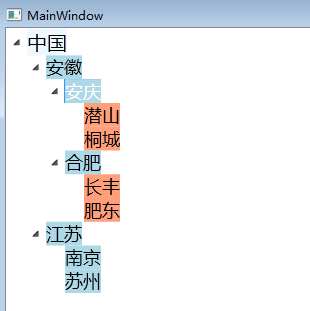


哈哈，确实有意思，我们改变了ListBox中Item的默认排序方向，当然在menu，statusBar中我们也可以用同样的方式来更改。

四： HierarchicalDataTemplate

      它是针对具有分层数据结构的控件设计的，比如说TreeView，相当于可以每一个层级上做DataTemplate，很好很强大。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            xmlns:sys="clr-namespace:System;assembly=mscorlib"           xmlns:src="clr-namespace:WpfApplication1"            Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <XmlDataProvider x:Key="Info" XPath="Nations">               <x:XData>                   <Nations xmlns="">                       <Nation Name="中国">                           <Provinces>                               <Province Name="安徽">                                   <Citys>                                       <City Name="安庆">                                           <Countrys>                                               <Country Name="潜山"/>                                               <Country Name="桐城"/>                                           </Countrys>                                       </City>                                       <City Name="合肥">                                           <Countrys>                                               <Country Name="长丰"/>                                               <Country Name="肥东"/>                                           </Countrys>                                       </City>                                   </Citys>                               </Province>                               <Province Name="江苏">                                   <Citys>                                       <City Name="南京">                                           <Countys>                                               <Country Name="溧水"/>                                               <Country Name="高淳"/>                                           </Countys>                                       </City>                                       <City Name="苏州">                                           <Countys>                                               <Country Name="常熟"/>                                           </Countys>                                       </City>                                   </Citys>                               </Province>                           </Provinces>                       </Nation>                   </Nations>               </x:XData>           </XmlDataProvider>           <HierarchicalDataTemplate DataType="Nation" ItemsSource="{Binding XPath=Provinces/Province}">               <StackPanel Background="AliceBlue">                   <TextBlock FontSize="20" Text="{Binding XPath=@Name}"/>               </StackPanel>           </HierarchicalDataTemplate>           <HierarchicalDataTemplate DataType="Province" ItemsSource="{Binding XPath=Citys/City}">               <StackPanel Background="LightBlue">                   <TextBlock FontSize="18" Text="{Binding XPath=@Name}"/>               </StackPanel>           </HierarchicalDataTemplate>           <HierarchicalDataTemplate DataType="City" ItemsSource="{Binding XPath=Countrys/Country}">               <StackPanel Background="LightBlue">                   <TextBlock FontSize="18" Text="{Binding XPath=@Name}"/>               </StackPanel>           </HierarchicalDataTemplate>           <HierarchicalDataTemplate DataType="Country">               <StackPanel Background="LightSalmon">                   <TextBlock FontSize="18" Text="{Binding XPath=@Name}"/>               </StackPanel>           </HierarchicalDataTemplate>       </Window.Resources>       <TreeView ItemsSource="{Binding Source={StaticResource ResourceKey=Info},XPath=Nation}"></TreeView>   </Window> |

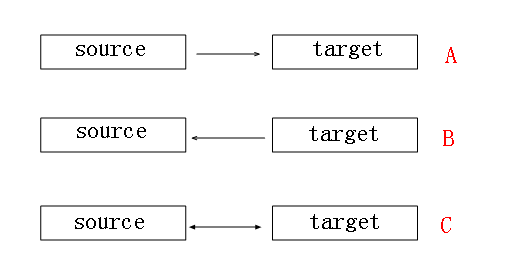


# 8天入门wpf—— 第五天 数据绑定

在webform中，如果提到“绑定”二字，相信大家都不会陌生，绑定，让我们的代码更加的简洁优美，在wpf中也存在各种神马的绑定，当然使用上都是行隔理不隔。

一： 控件到控件的绑定

    既然是绑定，那么肯定就有”源对象“和”目标对象“两种状态实体，从图的角度上来说存在三种状态：



确实在wpf中存在这三种模式的对应方式，

1：OneWay

   正如图A所说，Source影响着Target，但是Target却影响不到Source。

2：OneWayToSource

  也正如图B中所表述的一样，Target影响Source，而Source却影响不到Target。

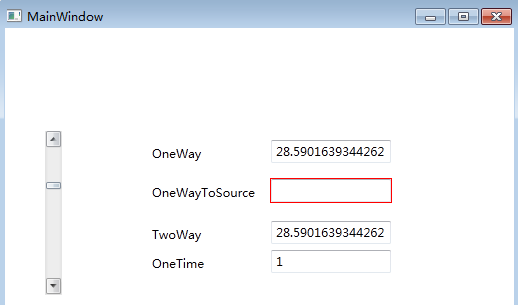
3：TwoWay

  这个也就相当于无向图的边，Source与Target相互影响。

4：OneTime

  在OneWay的基础上延伸了一个OneTime，仅绑定一次。如果大家属性Jquery中的one函数我想就可以不用表述了。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <Window x:Class="WpfApplication1.MainWindow"            xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"            xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"            Title="MainWindow" Height="350" Width="525">       <Canvas>           <ScrollBar Height="24" Name="scrollBar1" Width="237" Orientation="Horizontal" Canvas.Left="103" Canvas.Top="51"  Minimum="1" Maximum="100" SmallChange="1" />           <Label Canvas.Left="41" Canvas.Top="121" Content="OneWay" Height="28" Name="label1" />           <TextBox Canvas.Left="165" Canvas.Top="121" Height="23"                    Text="{Binding ElementName=scrollBar1, Path=Value, Mode=OneWay}"                    Name="textBox1" Width="120" />           <Label Canvas.Left="41" Canvas.Top="160" Content="OneWayToSource" Height="28" Name="label2" />           <TextBox Canvas.Left="165" Canvas.Top="160" Height="23"                      Text="{Binding ElementName=scrollBar1, Path=Value, Mode=OneWayToSource}"                    Name="textBox2" Width="120" />           <Label Canvas.Left="41" Canvas.Top="202" Content="TwoWay" Height="28" Name="label3" />           <TextBox Canvas.Left="165" Canvas.Top="202" Height="23"                      Text="{Binding ElementName=scrollBar1, Path=Value, Mode=TwoWay}"                    Name="textBox3" Width="120" />           <Label Canvas.Left="41" Canvas.Top="231" Content="OneTime" Height="28" Name="label4" />           <TextBox Canvas.Left="165" Canvas.Top="231" Height="23"                      Text="{Binding ElementName=scrollBar1, Path=Value, Mode=OneTime}"                    Name="textBox4" Width="120" />       </Canvas>   </Window> |

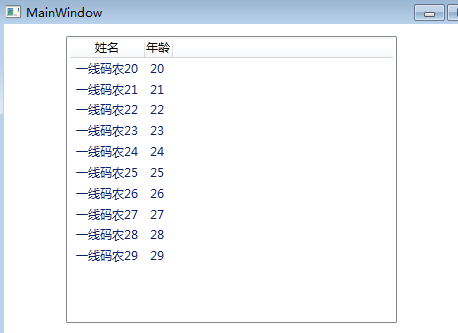


最终的结果，还是大家自己拖拖滚动条吧，有图有真相。

二：.net对象与控件的绑定

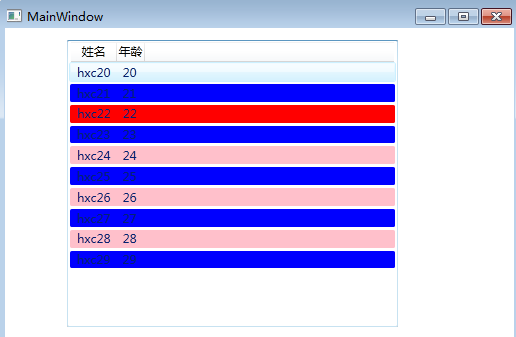
    这种绑定还是经常使用的，在WebForm中我们常用的Eval就是此种绑定，因为俺从数据库里好不容易捞了点数据总要呈现在UI上面吧，好，不多说，上代码说话。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <Window x:Class="WpfApplication3.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:local="clr-namespace:WpfApplication3"           Title="MainWindow" Height="350" Width="525">       <Grid>           <ListView  Height="287" HorizontalAlignment="Left" Margin="62,12,0,0" Name="listView1" VerticalAlignment="Top" Width="331">               <ListView.View>                   <GridView>                       <GridView.Columns>                           <GridViewColumn Header="姓名"  DisplayMemberBinding="{Binding Name}"/>                           <GridViewColumn Header="年龄" DisplayMemberBinding="{Binding Age}"/>                       </GridView.Columns>                   </GridView>               </ListView.View>           </ListView>       </Grid>   </Window> |



首 先谢天谢地，我们的数据出来了，好，现在我们有需求了，我现在需要给奇偶行填充不同底色，并且age=22的这行数据标红，那在wpf中该怎么做呢？我们 依稀的记得在webform中我们会在“行事件”上做手脚，在数据的绑定上wpf给我们提供了一个口子，也就是在绑定时可以插入自己的”事件处理代码“, 但必须要继承自IValueConverter。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | <Window x:Class="WpfApplication3.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:local="clr-namespace:WpfApplication3"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <local:ColorConvert x:Key="myConvert"/>           <Style x:Key="item" TargetType="{x:Type ListViewItem}">               <Setter Property="Background">                   <Setter.Value>                       <Binding RelativeSource="{RelativeSource Self}"   Converter="{StaticResource myConvert}"/>                   </Setter.Value>               </Setter>           </Style>       </Window.Resources>       <Grid>           <ListView ItemContainerStyle="{StaticResource ResourceKey=item}" Height="287" HorizontalAlignment="Left" Margin="62,12,0,0" Name="listView1" VerticalAlignment="Top" Width="331">               <ListView.View>                   <GridView>                       <GridView.Columns>                           <GridViewColumn Header="姓名"  DisplayMemberBinding="{Binding Name}"/>                           <GridViewColumn Header="年龄" DisplayMemberBinding="{Binding Age}"/>                       </GridView.Columns>                   </GridView>               </ListView.View>           </ListView>       </Grid>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72 | using System;   using System.Collections.Generic;   using System.Linq;   using System.Text;   using System.Windows;   using System.Windows.Controls;   using System.Windows.Data;   using System.Windows.Documents;   using System.Windows.Input;   using System.Windows.Media;   using System.Windows.Media.Imaging;   using System.Windows.Navigation;   using System.Windows.Shapes;     namespace WpfApplication3   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();                 List<Student> list = new List<Student>();                 for (int i = 20; i < 30; i++)               {                   list.Add(new Student() { Name = "hxc" + i, Age = i });               }                 listView1.ItemsSource = list;           }       }         public class ColorConvert : IValueConverter       {           public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)           {               //这里的value既为当前的行对象               var item = value as ListViewItem;                 //获取当前的item在当前的Listview中的位置               var view = ItemsControl.ItemsControlFromItemContainer(item) as ListView;                 var index = view.ItemContainerGenerator.IndexFromContainer(item);                 //当Age=22是红色标示               if ((view.Items[index] as Student).Age ==22)                   return Brushes.Red;                 if (index % 2 == 0)                   return Brushes.Pink;               else                   return Brushes.Blue;           }             public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)           {               return null;           }       }           public class Student       {           public string Name { get; set; }             public int Age { get; set; }       }   } |



快看，效果出来了，这里要稍微解释下IValueConverter的使用步骤：

①：自定义一个类继承自IValueConverter，其中Convert方法的value 为绑定参数，parameter参数为绑定参数的附带值。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <span style="color:#008080;">1</span><span style="color:#0000FF;">public</span><span style="color:#0000FF;">object</span> Convert(<span style="color:#0000FF;">object</span> value, Type targetType, <span style="color:#0000FF;">object</span> parameter, System.Globalization.CultureInfo culture) |

②：我们需要在xaml中引用并写入资源。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11 | <Window.Resources>           <local:ColorConvert x:Key="myConvert"/>           <Style x:Key="item" TargetType="{x:Type ListViewItem}">               <Setter Property="Background">                   <Setter.Value>                       <Binding RelativeSource="{RelativeSource Self}"   Converter="{StaticResource myConvert}"/>                   </Setter.Value>               </Setter>           </Style>       </Window.Resources> |

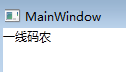
③：最后也就是在Binding中使用Convert,wpf在绑定数据的时候会自动调用我们自定义的myConvert方法。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11 | <Window.Resources>           <local:ColorConvert x:Key="myConvert"/>           <Style x:Key="item" TargetType="{x:Type ListViewItem}">               <Setter Property="Background">                   <Setter.Value>                       <Binding RelativeSource="{RelativeSource Self}"   Converter="{StaticResource myConvert}"/>                   </Setter.Value>               </Setter>           </Style>       </Window.Resources> |

三： .net方法与控件的绑定

   在做wpf时，有时我们需要在xaml中绑定.net中的方法，当然这在实际开发中也是很常用的，不过方法必要由ObjectDataProvider来封装。

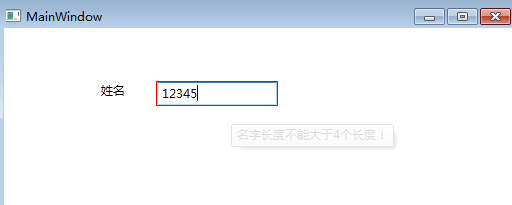
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <Window x:Class="WpfApplication5.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:local="clr-namespace:WpfApplication5"           xmlns:sys="clr-namespace:System;assembly=mscorlib"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <ObjectDataProvider x:Key="Test" ObjectType="{x:Type local:Student}" MethodName="GetName">           </ObjectDataProvider>       </Window.Resources>       <Grid>           <TextBlock Text="{Binding Source={StaticResource ResourceKey=Test}, Mode=OneWay}"/>       </Grid>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | namespace WpfApplication5   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();           }       }         public class Student       {           //前台要引用的方法           public string GetName()           {               return "一线码农";           }       }   } |



四：wpf中的验证

   我们知道不管在什么体系架构中都有属于自己的一套验证体系，比如webform中的验证控件，mvc中的特性验证，当然wpf也是有的，为了验证的灵活 性，实际开发中我们用的比较多的还是”自定义验证“，其实只需要实现ValidationRule接口就行了，然后写上自定义的验证逻辑。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <Window x:Class="WpfApplication4.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"          xmlns:local="clr-namespace:WpfApplication4"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <local:Student x:Key="student"/>       </Window.Resources>       <Grid>           <TextBlock Height="23" HorizontalAlignment="Left" Margin="97,54,0,0" Name="textBlock1" Text="姓名" VerticalAlignment="Top" />           <TextBox DataContext="{StaticResource ResourceKey=student}" Height="23" HorizontalAlignment="Left" Margin="153,54,0,0" Name="textBox1" VerticalAlignment="Top" Width="120">               <TextBox.Text>                   <Binding Path="Name" UpdateSourceTrigger="PropertyChanged">                       <!-- 自定义的验证规格，当然可以是多个Check -->                       <Binding.ValidationRules>                           <local:NameCheck />                       </Binding.ValidationRules>                   </Binding>               </TextBox.Text>               <TextBox.ToolTip>                   <!--将当前的错误信息显示在tooltip上-->                   <Binding RelativeSource="{RelativeSource Self}"  Path="(Validation.Errors)[0].ErrorContent"/>               </TextBox.ToolTip>           </TextBox>       </Grid>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | using System;   using System.Collections.Generic;   using System.Linq;   using System.Text;   using System.Windows;   using System.Windows.Controls;   using System.Windows.Data;   using System.Windows.Documents;   using System.Windows.Input;   using System.Windows.Media;   using System.Windows.Media.Imaging;   using System.Windows.Navigation;   using System.Windows.Shapes;     namespace WpfApplication4   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();           }       }         public class NameCheck : ValidationRule       {           public override ValidationResult Validate(object value, System.Globalization.CultureInfo cultureInfo)           {               var name = Convert.ToString(value);                 //如果名字长度大于4则是非法               if (name.Length > 4)                   return new ValidationResult(false, "名字长度不能大于4个长度！");                 return ValidationResult.ValidResult;           }       }         public class Student       {           public string Name { get; set; }             public int Age { get; set; }       }   } |



同样这里也需要注意的就是：

①   实现ValidationRule接口，重写Validate方法，其中的逻辑，你懂的。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <span style="color:#008080;">1</span><span style="color:#0000FF;">public</span><span style="color:#0000FF;">override</span> ValidationResult Validate(<span style="color:#0000FF;">object</span> value, System.Globalization.CultureInfo cultureInfo) |

② 然后我们在需要验证的控件上追加Rule验证, 其中的UpdateSourceTrigger设定为字段改变时触发，当然可选值有很多...

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | <TextBox.Text>                   <Binding Path="Name" UpdateSourceTrigger="PropertyChanged">                       <!-- 自定义的验证规格，当然可以是多个Check -->                       <Binding.ValidationRules>                           <local:NameCheck />                       </Binding.ValidationRules>                   </Binding>               </TextBox.Text> |

③ 最后要将实体写入到验证控件的DataContext上，最后大功告成。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <span style="color:#008080;">1</span>  <span style="color:#0000FF;"><</span><span style="color:#800000;">TextBox </span>DataContext<span style="color:#0000FF;">="</span><span style="color:#808000;">{StaticResource ResourceKey=student}</span><span style="color:#0000FF;">"</span><span style="color:#FF0000;"> Height</span><span style="color:#0000FF;">="23"</span><span style="color:#FF0000;"> HorizontalAlignment</span><span style="color:#0000FF;">="Left"</span><span style="color:#FF0000;"> Margin</span><span style="color:#0000FF;">="153,54,0,0"</span><span style="color:#FF0000;"> Name</span><span style="color:#0000FF;">="textBox1"</span><span style="color:#FF0000;"> VerticalAlignment</span><span style="color:#0000FF;">="Top"</span><span style="color:#FF0000;"> Width</span><span style="color:#0000FF;">="120"</span><span style="color:#0000FF;">></span> |

# 8天入门wpf—— 第六天 细说控件

这篇我们来大概的看一下WPF的各种神马控件，首先我们要知道所有的wpf控件都是继承自Control，从用途上可以分为四种

        1:内容控件(Content Controls)

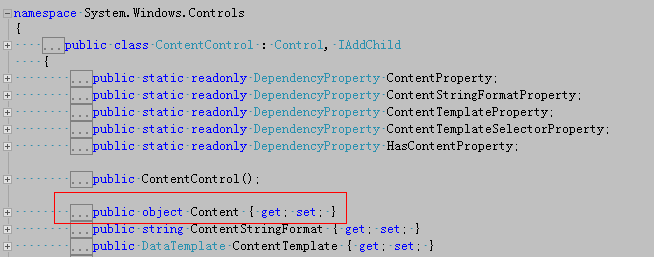
        2:条目控件(Items Controls)

        3:文本控件(Text Controls)

        4:范围控件(Range Controls)

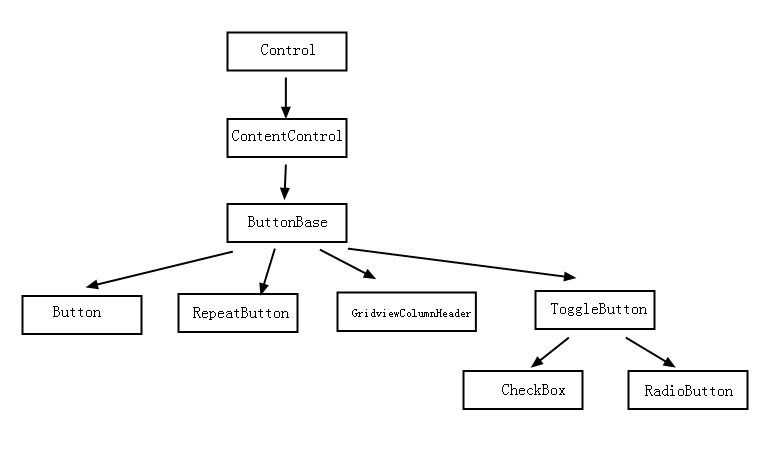
一：内容控件

   内容控件的最大的特征就是有一个Content属性，从前面的文章中，我们多多少少也知道Content接收的是一个Object类型，或许我们会立即 想到莫非Button就是一个内容控件，确实，Button算是一个内容控件，凡是内容控件都继承自ContentControl，因为Content属 性就是属于ContentControl。

[](http://img.ddvip.com/2012/1012/201210120355311667.png)

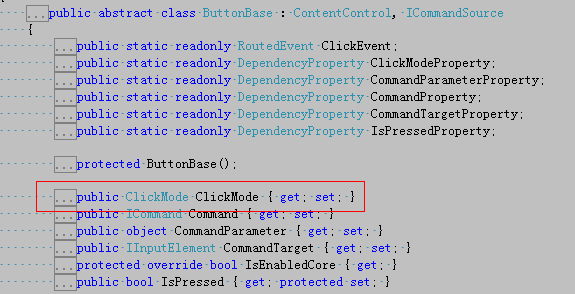
1：ButonBase

    要说Button，我们需要从ButtonBase说起，首先看一下类图。

[](http://img.ddvip.com/2012/1012/201210120355316205.png)

<1>Button

 从图中我们可以看出，Button是继承自ButtonBase的，Button有个很有趣的地方就是ButtonBase中存在一个 ClickMode属性，什么意思呢？也就是说Click事件是用什么方式触发？触发方式在ClickMode中以枚举的方式展 现，Hover，Press和Release，默认也就是Press。

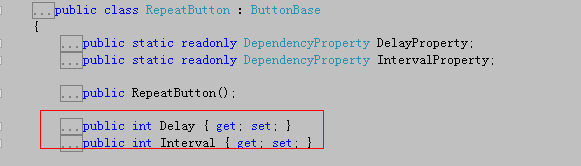
[](http://img.ddvip.com/2012/1012/201210120355324852.png)

那么下面我用Hover的形式来触发Click事件，蛮有意思的，嘿嘿。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | <Window x:Class="ButtonDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <Button Content="Button" Height="23" IsDefault="True" Margin="124,77,304,211" ClickMode="Hover" Name="button1"  Width="75" Click="button1\_Click" />       </Grid>   </Window> |

<2>RepeatButton

   首先这玩意我们在webform或者winform中貌似都没有见过，在wpf中也是一个新增的控件，那么它的用途是什么呢？很简单，我们在看 video的时候都有“快进”，“快退”，你懂的，首先我们看下RepeatButton中的定义，我们发现有一个Delay：作用就是按下时第一次触发 Click的时间延迟，Interval：每次click发生的时间间隔，如果大家玩转了Timer控件都应该很清楚。

[](http://img.ddvip.com/2012/1012/201210120355327011.png)

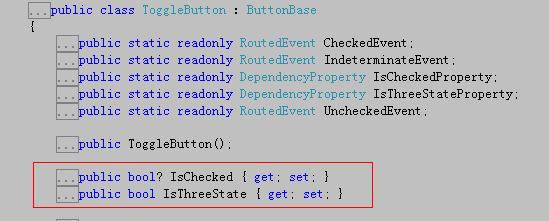
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10 | <Window x:Class="RepeatButtonDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Canvas>           <TextBox Canvas.Left="151" Canvas.Top="69" Height="33" Name="textBox1" Width="172" Text="0" />           <RepeatButton x:Name="test" Delay="100" Click="test\_Click" Width="172"                         Content="确定" Height="61" Canvas.Left="151" Canvas.Top="121" />       </Canvas>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | namespace RepeatButtonDemo   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();           }             private void test\_Click(object sender, RoutedEventArgs e)           {               var num = Convert.ToInt32(textBox1.Text);                 textBox1.Text = (++num).ToString();           }       }   } |

<3>GridViewColumnHeader

  这个是与GridView控件一起搭配使用，放在后面一起讲。

<4>ToggleButton

    从图中我们看到ToggleButton是CheckBox和RadioButton的基类，大家一看，这玩意我们早就会了，是的，大家都会，这里我就说点有新鲜味道的东西，首先我们看下ToggleButton的类定义。

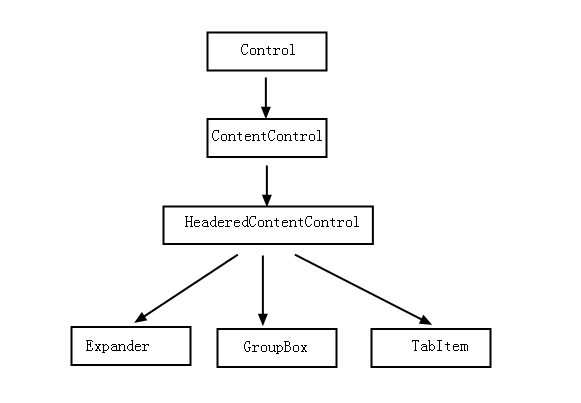


嘿 嘿，兴趣点来了，怎么IsChecked是可空类型？而且还存在王八IsThreeState属性，难道还有第三种状态？是的，这是在Html中没有的， 这里我们要知道，实际上我们最终的UI呈现的要么是CheckBox,要么是Radiobutton，要使第三种状态有效，我们只需要设置 IsThreeState属性和Indeterminate事件即可。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | <Window x:Class="CheckBoxDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <CheckBox Content="CheckBox" Height="16" HorizontalAlignment="Left" Margin="96,137,0,0" Name="checkBox1" VerticalAlignment="Top" IsThreeState="True" Indeterminate="checkBox1\_Checked" />       </Grid>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | namespace CheckBoxDemo   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();           }             private void checkBox1\_Checked(object sender, RoutedEventArgs e)           {               MessageBox.Show("不错");           }       }   } |

2: HeaderedContentControl

   顾名思义，这是一个带有标题的内容控件，或许大家第一个反应过来的估计就是GroupBox，是的，这玩意我们在Html中用的太多了，老规矩，我们还是看看类图。

[](http://img.ddvip.com/2012/1012/201210120355330262.png)

<1> Expander

    首先得要申明，Expander是wpf中新增的一个控件，在html中我们经常会看到一个伸缩控件，折叠起来只能看到标题，伸展开才能看到内容，继续上代码说话。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <Window x:Class="ExpanderDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <Expander Header="年龄组" Height="208" Margin="39,33,154,70" Name="expander1" Width="310">               <StackPanel>                   <RadioButton Content="RadioButton1" Height="16" Name="radioButton1" />                   <RadioButton Content="RadioButton2" Height="16"  Name="radioButton2" />               </StackPanel>           </Expander>         </Grid>   </Window> |

<2> GroupBox

      关于GroupBox的使用，我想我也不用罗嗦了，太简单不过了。

<3>TabItem

      TabItem控件是与TabControl控件搭配使用，这个放到条目控件上说。

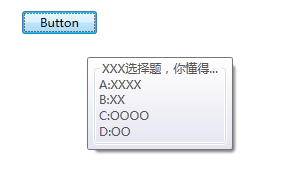
3：ToolTip

   首先我们要知道ToolTip也是继承自ContentControl，在使用ToolTip的时候我们要注意两点。

<1>: ToolTip有点特殊，它不能独立的作为一个控件使用，而是与其他具体控件的ToolTip联合使用。

<2>:ToolTip提供了一个ToolTipSerivce类，可用于设计Tooltip显示的相对位置，提示时间，嘿嘿，蛮有意思。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <Window x:Class="ToolTipDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <Button Content="Button" Height="23" HorizontalAlignment="Left" Margin="130,103,0,0" Name="button1" VerticalAlignment="Top" Width="75"                   ToolTipService.HorizontalOffset="20"                   ToolTipService.VerticalOffset="20" >               <Button.ToolTip>                   <StackPanel>                       <GroupBox Header="XXX选择题，你懂得...">                           <GroupBox.Content>                               <StackPanel>                                   <TextBlock x:Name="A">A:XXXX</TextBlock>                                   <TextBlock x:Name="B">B:XX</TextBlock>                                   <TextBlock x:Name="C">C:OOOO</TextBlock>                                   <TextBlock x:Name="D">D:OO</TextBlock>                               </StackPanel>                           </GroupBox.Content>                       </GroupBox>                   </StackPanel>               </Button.ToolTip>           </Button>       </Grid>   </Window> |

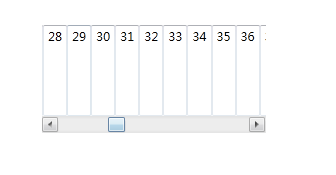


4：ScrollViewer

  在内容控件中很常用的一个莫过于ScrollViewer，因为我们在界面布局时，永远都是“内容”大于界面，那么内容超出了我们该怎么办呢？

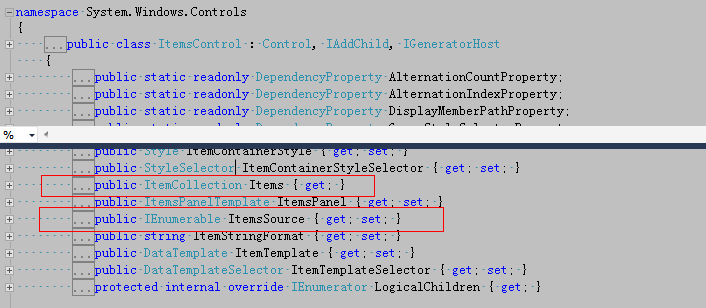
我们知道Html中Div具有裁剪功能，当内容超出，自动就有滚动条，那么在wpf中的ScrollViewer也能够实现同样的功能。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <Window x:Class="ScrollViewerDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <ScrollViewer Height="108" HorizontalAlignment="Left" Margin="98,63,0,0"                         Name="scrollViewer1" VerticalAlignment="Top" Width="224"                         VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">               <StackPanel x:Name="Test" Orientation="Horizontal">                 </StackPanel>           </ScrollViewer>       </Grid>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | namespace ScrollViewerDemo   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();                 for (int i = 0; i < 100; i++)               {                   TextBox tbx = new TextBox();                     tbx.Text = i.ToString();                     Test.Children.Add(tbx);               }           }       }   } |



二：条目控件

  条目控件首先都是继承自ItemsControl，在ItemsControl中我们发现有两个比较有意思的属性，Items和ItemsSource。

[](http://img.ddvip.com/2012/1012/201210120355335361.png)

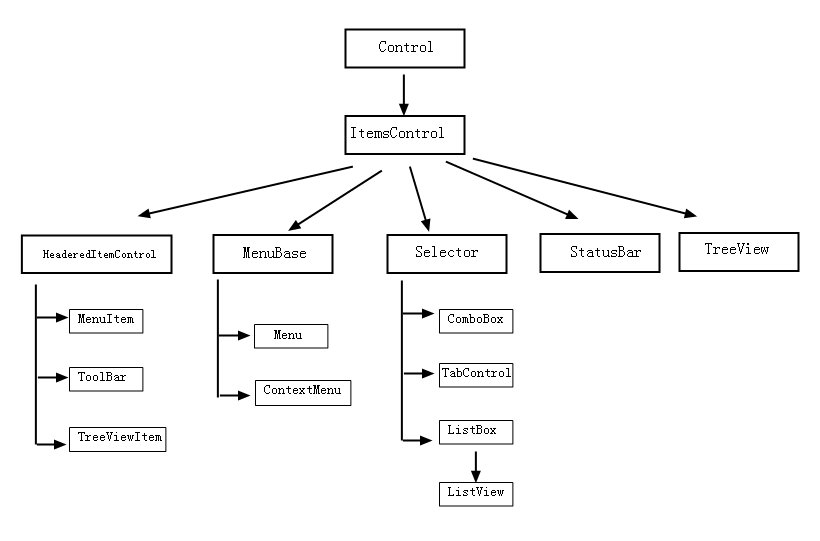
Items：

从 图中可以看出Items属于ItemCollection的集合类型，所以每一个Item里面都可以放入一个Object类型对象，这里有意思的地方就 是，如果我放入的是一个UI元素，那么很好，wpf会调用UI的OnRender方法将UI元素呈现，如果说是一个没有OnRender方法的元素，那该 怎么办呢？wpf很智能，它会创建一个TextBlock，然后调用该对象的ToString()将字符串呈现在TextBlock上。

ItemsSource:

从前面文章中我们也看到，ItemsSource常用于数据绑定，所以是一个非常实用的属性。

好，接下来我们看一下条目控件的类图

[](http://img.ddvip.com/2012/1012/201210120355338920.png)

从图中，我们大致可以看出这些控件可以分为两大类。

第一类：就是“条目容器”，比如Menu。

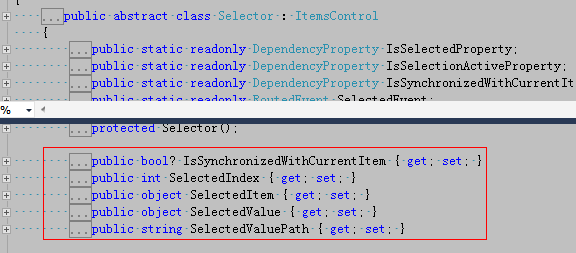
第二类：就是“具体条目”，比如MenuItem，但是在MenuItem中又可以分为“带标题”和“不带标题”的两类具体条目”。

<1>MenuBase

   从图中我们可以看出MenuBase的子类有两个Menu和ContextMenu,在Winform中我想大家肯定玩烂了，这里我也不多说了。

<2>Selector

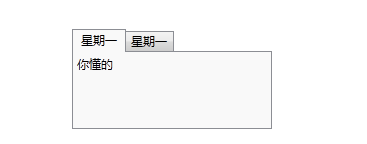
既 然是选择性的控件，那么难免少不了SelectedIndex或者SelectedItem，可以我们反应就是Listbox，嘿嘿，关于ListBox 和ComboBox这里就不多说了，我们具体的还是看下TabControl和ListView,先还是看下Selector类中的定义。

[](http://img.ddvip.com/2012/1012/201210120355342318.png)

TabControl:

 这个控件我们在Html中用的还是比较多的，顾名思义就是选项卡，因为我们知道用ListBox是很占用空间的，而TabControl是具有更小的地方展现更多的内容，其实TabControl的每一个标签页都是一个TabItem。

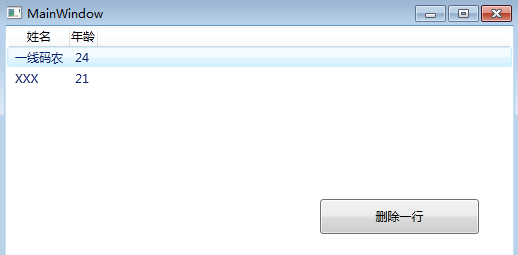
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="TabControlDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <TabControl Height="100" HorizontalAlignment="Left" Margin="103,51,0,0"                       Name="tabControl1" VerticalAlignment="Top" Width="200"                        TabStripPlacement="Top" >               <TabItem Header="星期一" Name="tabItem1">                   <TabItem.Content>                       <TextBlock x:Name="test1" Text="你懂的"/>                   </TabItem.Content>               </TabItem>               <TabItem Header="星期一" Name="tabItem2">                   <TabItem.Content>                       <TextBlock x:Name="test2" Text="你不懂的"/>                   </TabItem.Content>               </TabItem>           </TabControl>       </Grid>   </Window> |



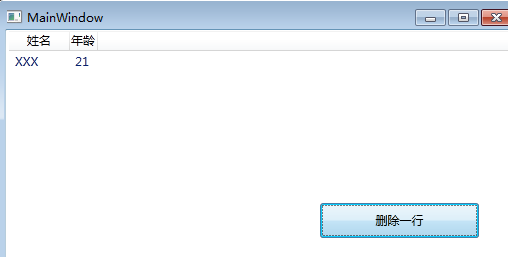
ListView：

这 个控件我们在实际开发中经常用于数据绑定，它是继承自ListBox，ListBox默认只能显示一列，而ListView则可以用于显示多列，这里我提 一个很有兴趣的玩意ObservableCollection<T>。它有什么用呢？其实ObservableCollection可以允许 一个UI元素作为观察者对它进行监视，也就是说如果ObservableCollection中的元素有变动，作为观察的UI元素也会相应的改变，下面举 个例子。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <Window x:Class="ListViewDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           DataContext="{Binding RelativeSource={RelativeSource Self}}"           Title="MainWindow" Height="350" Width="525">       <Grid>           <ListView ItemsSource="{Binding PersonList}">               <ListView.View>                   <GridView>                       <GridViewColumn Header="姓名" DisplayMemberBinding="{Binding Path=Name}"/>                       <GridViewColumn Header="年龄" DisplayMemberBinding="{Binding Path=Age}"/>                   </GridView>               </ListView.View>           </ListView>           <Button Content="删除一行" Click="Button\_Click" Margin="315,174,35,103" />       </Grid>   </Window> |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | namespace ListViewDemo   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           private ObservableCollection<Person> personList = new ObservableCollection<Person>();             public ObservableCollection<Person> PersonList           {               get { return personList; }               set { personList = value; }           }             public MainWindow()           {               InitializeComponent();                 personList.Add(new Person() { Name = "一线码农", Age = 24 });                 personList.Add(new Person() { Name = "XXX", Age = 21 });           }             private void Button\_Click(object sender, RoutedEventArgs e)           {               personList.RemoveAt(0);           }       }         public class Person       {           public string Name { get; set; }             public int Age { get; set; }       }   } |



             <====  单击删除一行后  ====>



<3>StatusBar

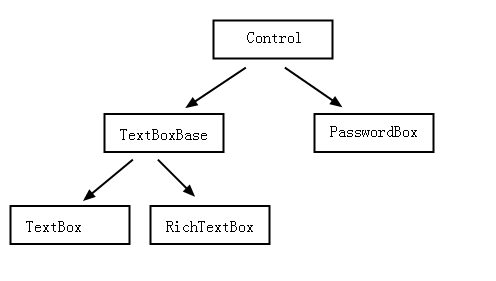
关于状态栏控件，我想大家在WinForm中已经玩烂了，此处也就不多说了。

<4>TreeView

我们知道TreeView是一个树形控件，在Html中如果想展现一个树形结构，我们只要将数据结构“深度优先”一下就OK了，关于TreeView的数据绑定，我的前一篇文章也说过，这里也就走马观花一下。

三：文本控件

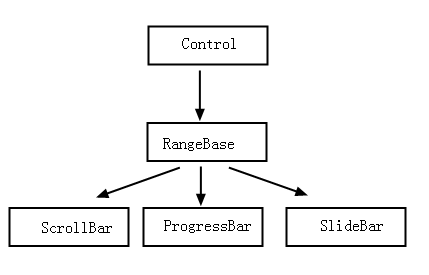
在wpf中，文本控件有三个，分别是：TextBox,RichTextBox和PasswordBox，先不管怎么样，上类图



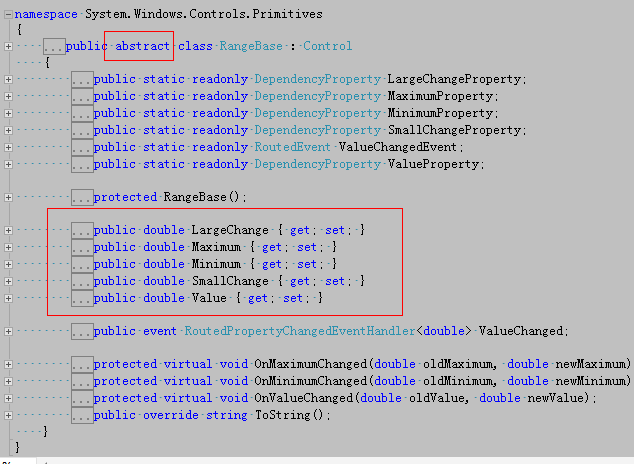
这几个控件，我想在winform中用的还是比较熟的，这里也就不罗嗦了。

四：范围控件

还是先上图：



下面我们看看RangeBase的类定义：

[](http://img.ddvip.com/2012/1012/201210120355403827.png)

图中可以看出RangeBase是一个抽象类，定义了4个基本属性：LargeChange,SmallChange,Maximum,Minimum，有了这些东西我们才能方便快捷的使用范围控件。

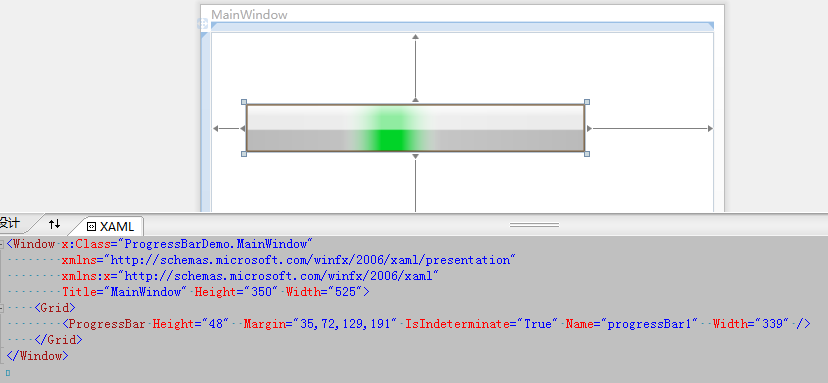
<1> ScrollBar

在先前的例子中，我们经常用一个控件来绑定ScrollBar的Value来形成联动，也就可以避免在后台的不必要代码，灵活方便。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <Window x:Class="ScrollBar.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <StackPanel Height="100" Margin="97,61,206,150" Name="stackPanel1" Width="200">               <ScrollBar Name="test" Orientation="Horizontal" Maximum="100" Minimum="5" SmallChange="2" Height="17" Width="186" />               <Label Content="滑动块值"/>               <TextBox Name="txtScrollValue" Text="{Binding ElementName=test, Path=Value}"/>           </StackPanel>       </Grid>   </Window> |

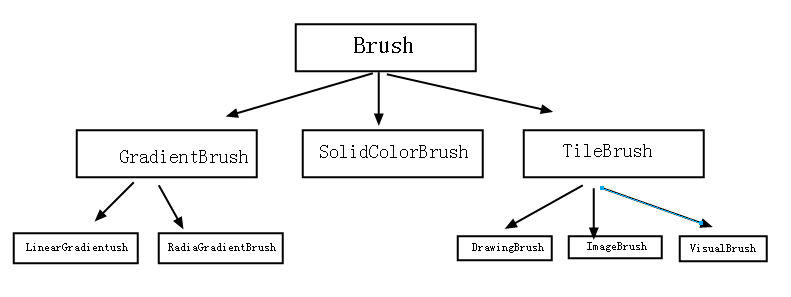
<2>ProgressBar

 这个控件我们在实际应用上使用的还是比较多的，因为我们在完成一个任务时，可能需要等待数十秒或者数分钟，所以为了不让用户认为系统处于假死状态，就要 用一个等待进度条，这里有意思的地方就是，如果我们不知道任务何时完成或者说不在乎任务何时结束，我们可以设一个无限等待的进度条，也就是说进度条上有一 个“小矩形”在不停的滚动，我们要做的也就是设置IsIndeterminate=true即可。

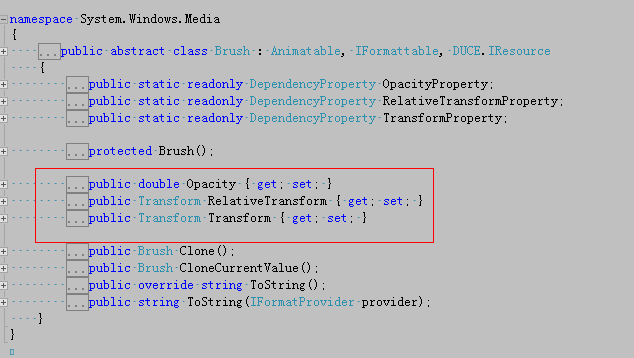
[](http://img.ddvip.com/2012/1012/201210120355405764.png)

# 8天入门wpf—— 第七天 画刷

这一篇我们聊聊wpf中的画刷，在wpf中如果想玩各种花哨，那么如何使用画刷则是我们的基本功，首先看一下类图

[](http://img.ddvip.com/2012/1012/201210120358297368.png)

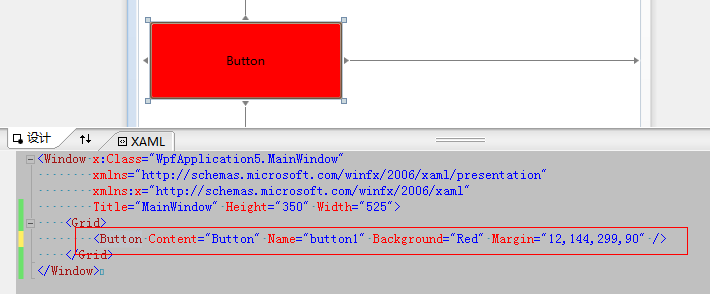
从图中可以看出，wpf有5种画刷和1种自定义画刷，都是继承自基类Brush，我们看看基类中有哪些好玩的东西。

[](http://img.ddvip.com/2012/1012/201210120358299441.png)

这里有3个比较感兴趣的属性，分别属于”透明度“和”图像转换“，好，下面我们一一解说。

一：SolidColorBrush（实心画刷）

   实心画刷是我们用的最多的，也是最简单的一个，其实也就是填充色的意思，一个很简单的例子：

[](http://img.ddvip.com/2012/1012/201210120358302306.png)

其实这里的Background=Red使用的就是SolidColorBrush，xaml进行解析时，发现Background是Brush类型，刚才我也说了

Brush具有图形转换的能力，最后xaml就会通过Transform把”Red"字符串解析成SolidColorBrush，更直观一点的话，我们可以

用C#代码来描述。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9 | public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();                 button1.Background = new SolidColorBrush(Colors.Red);           }       } |

二：GradientBrush(梯度画刷)

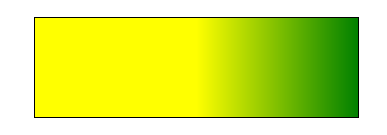
如果我们使用过ps或者freehand，我们肯定知道在填充色方面有一个叫做“渐变色”的概念，我们使用的最多的渐变色要么是“线性”的，

要么是“圆形”的，刚好这里对应wpf中的“LinearGradientBrush”和“RadialGradientBrush”。

1： LinearGradientBrush（线性梯度画刷）

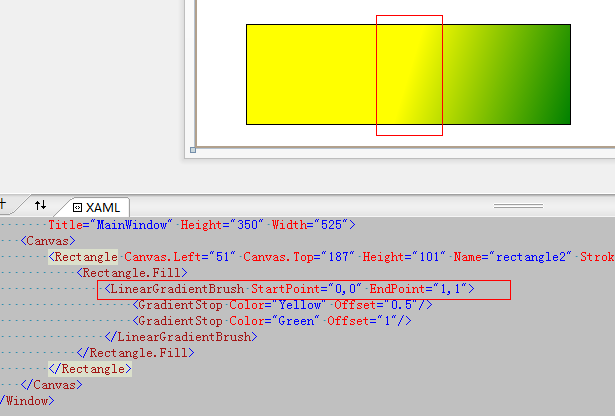
    线性画刷也是比较简单的，一般情况下我们只要设定一个“StartPoint”和“EndPoint”即可。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <Window x:Class="WpfApplication2.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Canvas>           <Rectangle Canvas.Left="51" Canvas.Top="187" Height="101" Name="rectangle2" Stroke="Black" Width="325">               <Rectangle.Fill>                   <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">                       <GradientStop Color="Yellow" Offset="0.5"/>                       <GradientStop Color="Green" Offset="1"/>                   </LinearGradientBrush>               </Rectangle.Fill>           </Rectangle>       </Canvas>   </Window> |



这里要注意的就是，我设定的坐标是(0,0),(0,1)，我们知道两点一条直线，这条直线与X轴平行，我们可以看到颜色的分布是垂直于Y轴的，

如果说我们把坐标改为(0,0)(1,1)，那么颜色分割线还是与(0,0),(1,1)这条斜线垂直吗？最后发现，严格垂直。

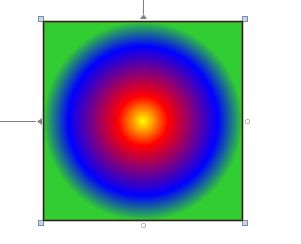
[](http://img.ddvip.com/2012/1012/201210120358356031.png)

2：RadialgradientBrush(圆形梯度画刷)

   在ps中我们玩”圆形渐变“的时候，只需要设定圆心坐标和X坐标和Y坐标的值就可以画一个圆形渐变，在wpf中同样需要这三个元素，

分别对应设Center,RadiusX,RadiusY,当然在wpf中还存在一个“梯度原点“：GradientOrigin。

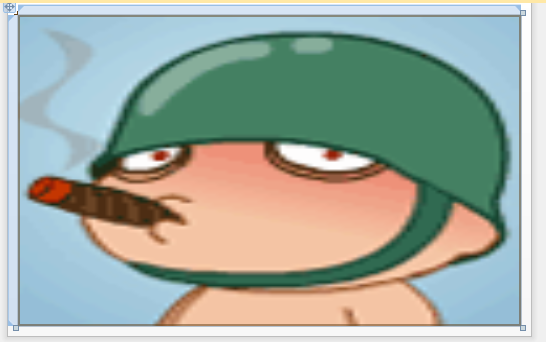
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <Window x:Class="WpfApplication3.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <Rectangle Height="200" HorizontalAlignment="Left" Margin="128,45,0,0" Name="rectangle1" Stroke="Black" VerticalAlignment="Top" Width="200">               <Rectangle.Fill>                   <RadialGradientBrush GradientOrigin="0.5,0.5" Center="0.5,0.5" RadiusX="0.5" RadiusY="0.5">                       <RadialGradientBrush.GradientStops>                           <GradientStop Color="Yellow" Offset="0"/>                           <GradientStop Color="Red" Offset="0.25"/>                           <GradientStop Color="Blue" Offset="0.75"/>                           <GradientStop Color="LimeGreen" Offset="1"/>                       </RadialGradientBrush.GradientStops>                   </RadialGradientBrush>               </Rectangle.Fill>           </Rectangle>       </Grid>   </Window> |



三：ImageBrush（图像画刷）

     这种画刷也是很有意思的，有时我们在炫时需要用图片做装饰，那么此时ImageBrush就可以祝你一臂之力。

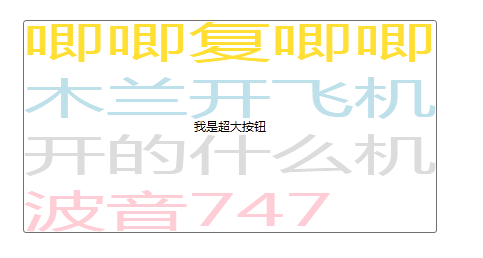
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11 | <Window x:Class="WpfApplication7.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:my="clr-namespace:WpfApplication7"           Title="MainWindow" Height="350" Width="525">       <Grid>           <Grid.Background>               <ImageBrush x:Name="landBrush" ImageSource="C:UsersAdministratorDesktopweibo64512.gif"/>           </Grid.Background>       </Grid>   </Window> |



四：VisualBrush（控件画刷）

    这种画刷是作用在控件级别上的，也就是说任何控件都可以作为画刷，很神奇的说。

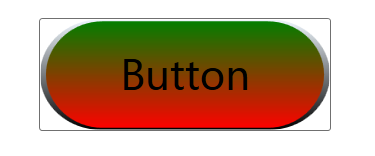
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | <Window x:Class="WpfApplication1.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <VisualBrush x:Key="test" TileMode="Tile" Opacity="0.8">               <VisualBrush.Visual>                   <StackPanel>                       <TextBlock Foreground="Gold">                           唧唧复唧唧                       </TextBlock>                       <TextBlock Foreground="LightBlue">                          木兰开飞机                       </TextBlock>                       <TextBlock Foreground="LightGray">                          开的什么机                       </TextBlock>                       <TextBlock Foreground="Pink">                          波音747                       </TextBlock>                   </StackPanel>               </VisualBrush.Visual>           </VisualBrush>       </Window.Resources>       <Grid>           <Button Content="我是超大按钮" Height="213" HorizontalAlignment="Left" Margin="32,34,0,0" Name="button1"                   VerticalAlignment="Top" Width="414" Background="{StaticResource ResourceKey=test}"/>       </Grid>   </Window> |



五：DrawingBrush（自定义画刷）

    最灵活，最复杂的也就是这种自定义画刷，毕竟wpf不能给我们满足所有的要求，就必须留一道口子给我们程序员自定义实现。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | <Window x:Class="WpfApplication4.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <DrawingBrush x:Key="test">               <DrawingBrush.Drawing>                   <DrawingGroup>                       <DrawingGroup.Children>                           <GeometryDrawing>                               <!-- 绘制矩形 -->                               <GeometryDrawing.Geometry>                                   <RectangleGeometry RadiusX="0.2" RadiusY="0.5"                                                          Rect="0.02,0.02,0.96,0.96" />                               </GeometryDrawing.Geometry>                               <!-- 矩形填充色 -->                               <GeometryDrawing.Brush>                                   <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">                                       <GradientStop Color="Green" Offset="0" />                                       <GradientStop Color="Red" Offset="1" />                                   </LinearGradientBrush>                               </GeometryDrawing.Brush>                               <!-- 矩形边框 -->                               <GeometryDrawing.Pen>                                   <Pen Thickness="0.02">                                       <Pen.Brush>                                           <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">                                               <GradientStop Color="AliceBlue" Offset="0" />                                               <GradientStop Color="Black" Offset="1" />                                           </LinearGradientBrush>                                       </Pen.Brush>                                   </Pen>                               </GeometryDrawing.Pen>                           </GeometryDrawing>                       </DrawingGroup.Children>                   </DrawingGroup>               </DrawingBrush.Drawing>           </DrawingBrush>       </Window.Resources>       <Grid>           <Button Background="{StaticResource ResourceKey=test}" FontSize="40" Content="Button" Height="113" HorizontalAlignment="Left" Margin="89,80,0,0" Name="button1" VerticalAlignment="Top" Width="292" />       </Grid>   </Window> |



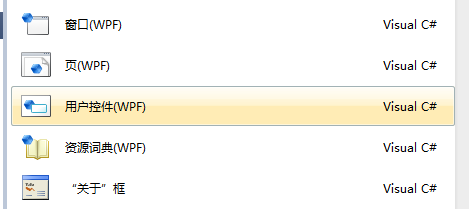
# 8天入门wpf—— 第八天 最后的补充

从这一篇往前看，其实wpf中还有很多东西没有讲到，不过我的原则还是将比较常用的知识点过一遍，如果大家熟悉了这些知识，基本功也就打的差不多了，后续可以等待老邓的wpf细说系列，这里我先顶老邓一下。

一：用户控件(UserControl）

     对于用户控件的认识，我想大家还是很熟悉的，因为这玩意我们在webform或者在mvc中用的可多了，我们看看wpf中怎么使用，先我们要知道"用户 控件“继承自UserControl,而UserControl继承自ContentControl,也就是上上一篇说的”内容控件”。

第一步：在vs中的添加项中找到一个“用户控件WPF”，点击添加即可。



第二步：我们发现其实UserControl和Window是一个层次上的，都有xaml和cs文件，然后我们在xaml中拖几个控件。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <UserControl x:Class="WpfApplication8.AddProduct"                xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"                xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"                xmlns:mc="<http://schemas.openxmlformats.org/markup-compatibility/2006>"                xmlns:d="<http://schemas.microsoft.com/expression/blend/2008>"                mc:Ignorable="d"                d:DesignHeight="200" d:DesignWidth="300">       <Grid Height="171" Width="262">           <TextBlock Height="20" HorizontalAlignment="Left" Margin="28,57,0,0" Name="textBlock1" Text="名称:" VerticalAlignment="Top" Width="42" />           <TextBlock Height="20" HorizontalAlignment="Left" Margin="28,92,0,0" Name="textBlock2" Text="价格:" VerticalAlignment="Top" Width="42" />           <TextBox Height="23" HorizontalAlignment="Left" Margin="76,54,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" />           <TextBox Height="23" HorizontalAlignment="Left" Margin="76,92,0,0" Name="textBox2" VerticalAlignment="Top" Width="120" />       </Grid>   </UserControl> |



第三步：我们在MainWindow中引用，跟webform中使用套路一模一样，最后也就ok了。

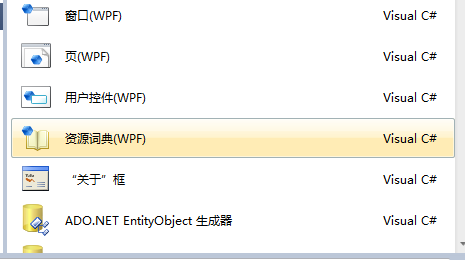
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9 | <Window x:Class="WpfApplication8.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:local="clr-namespace:WpfApplication8"           Title="MainWindow" Height="350" Width="525">       <Grid>           <local:AddProduct x:Name="test"/>       </Grid>   </Window> |



二：资源文件

    先前文章我也说过，资源就类似于webform中的css，但是实际应用中，css都是一个个单独的文件来实现内容与样式的分离，当然wpf中也主张这么做。

第一步：vs中新建项 -> 资源字典->点击确定



第二步：这里我就将默认生成的Dictionary1.xaml放在解决方案的Style文件夹下，然后我们写上一段简单的style。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6 | <ResourceDictionary xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"                      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>">      <Style x:Key="backColor" TargetType="{x:Type Button}">          <Setter Property="Background" Value="Red"/>      </Style>  </ResourceDictionary> |

第三步：在Resources上引用，指定资源文件路径，跟webform中的css文件引用一样一样的。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <Window x:Class="WpfApplication9.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Window.Resources>           <!-- 引用外部资源文件 -->           <ResourceDictionary>               <ResourceDictionary.MergedDictionaries>                   <ResourceDictionary Source="/Style/Dictionary1.xaml"/>               </ResourceDictionary.MergedDictionaries>           </ResourceDictionary>       </Window.Resources>       <Grid>           <Button Content="Button" Style="{StaticResource ResourceKey=backColor}" Height="23" HorizontalAlignment="Left" Margin="104,58,0,0" Name="button1" VerticalAlignment="Top" Width="75" />       </Grid>   </Window> |

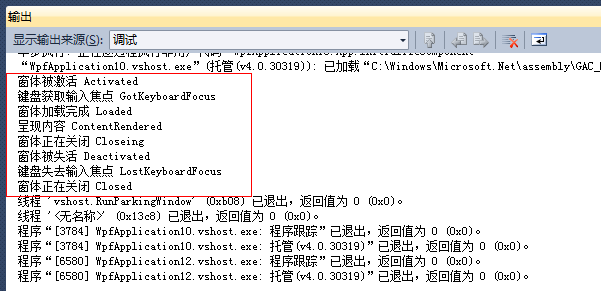
http://img.ddvip.com/2012/1012/201210120403133655.png

三：了解wpf中Window的生命周期

  了解生命周期，可以让我们更好的控制生命周期内各个阶段发生的行为，具体怎么灵活运用，得要看大家灵活发挥了。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85 | using System;   using System.Collections.Generic;   using System.Linq;   using System.Text;   using System.Windows;   using System.Windows.Controls;   using System.Windows.Data;   using System.Windows.Documents;   using System.Windows.Input;   using System.Windows.Media;   using System.Windows.Media.Imaging;   using System.Windows.Navigation;   using System.Windows.Shapes;   using System.Diagnostics;     namespace WpfApplication10   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();                 //初始化               this.Initialized += (sender, e) =>               {                   Debug.WriteLine("窗体初始化完成 Initialized");               };                 //激活               this.Activated += (sender, e) =>               {                   Debug.WriteLine("窗体被激活 Activated");               };                 //加载               this.Loaded += (sender, e) =>               {                   Debug.WriteLine("窗体加载完成 Loaded");               };                 //呈现内容               this.ContentRendered += (sender, e) =>               {                   Debug.WriteLine("呈现内容 ContentRendered");               };                 //失活               this.Deactivated += (sender, e) =>               {                   Debug.WriteLine("窗体被失活 Deactivated");               };                 //窗体获取输入焦点               this.GotFocus += (sender, e) =>               {                   Debug.WriteLine("窗体获取输入焦点 GotFocus");               };                 //窗体失去输入焦点               this.LostFocus += (sender, e) =>               {                   Debug.WriteLine("窗体失去输入焦点 LostFocus");               };                 //键盘获取输入焦点               this.GotKeyboardFocus += (sender, e) =>               {                   Debug.WriteLine("键盘获取输入焦点 GotKeyboardFocus");               };                 //键盘失去输入焦点               this.LostKeyboardFocus += (sender, e) =>               {                   Debug.WriteLine("键盘失去输入焦点 LostKeyboardFocus");               };                 //正在关闭               this.Closing += (sender, e) =>               {                   Debug.WriteLine("窗体正在关闭 Closeing");               };                 //关闭               this.Closed += (sender, e) =>               {                   Debug.WriteLine("窗体正在关闭 Closed");               };             }       }   } |

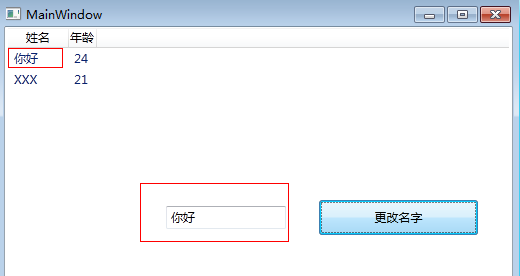
从窗体的开启到关闭，我们可以在“输出窗口”中看到如下的事件发生顺序流。

[](http://img.ddvip.com/2012/1012/201210120403134805.png)

四：属性更改通知（INotifyPropertyChanged）

    我们在开发webform中，如果删除GridView里面的一行，我们的作法肯定就是在数据库中删除掉选定的记录然后重新绑定GridView控件来实 现我们的需求，注意，这里有“重新绑定”一词，但是在wpf中有一个突破，前一篇文章我也提到过wpf中的 ObservableCollection<T>，MSDN中说，在添加项，移除项时此集合通知控件，我们知道对一个集合的操作是CURD， 但是恰恰没有Update的时候提供集合通知，也就是说当我Update的时候，虽然"集合内容“已被修改，但是"控件“却没有实现同步更新，怎么办 呢？INotifyPropertyChanged提供了解决方案。

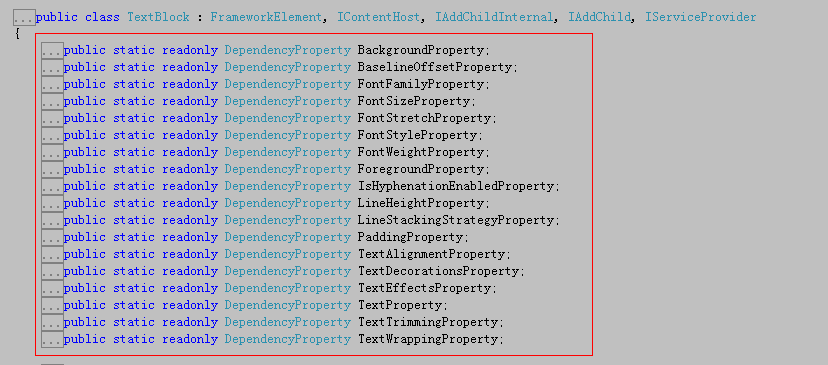
|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | using System;   using System.Collections.Generic;   using System.Linq;   using System.Text;   using System.Windows;   using System.Windows.Controls;   using System.Windows.Data;   using System.Windows.Documents;   using System.Windows.Input;   using System.Windows.Media;   using System.Windows.Media.Imaging;   using System.Windows.Navigation;   using System.Windows.Shapes;   using System.Collections.ObjectModel;   using System.Windows.Controls.Primitives;   using System.ComponentModel;     namespace ListViewDemo   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           private ObservableCollection<Person> personList = new ObservableCollection<Person>();             public MainWindow()           {               InitializeComponent();                 personList.Add(new Person() { Name = "一线码农", Age = 24 });                 personList.Add(new Person() { Name = "XXX", Age = 21 });                 listview1.ItemsSource = personList;           }             private void Button\_Click(object sender, RoutedEventArgs e)           {               var first = personList.FirstOrDefault();                 first.Name = textBox1.Text;           }       }         public class Person : INotifyPropertyChanged       {           public string name;             public string Name           {               get               {                   return name;               }               set               {                   name = value;                   NotifyPropertyChange("Name");               }           }             public int age;             public int Age           {               get               {                   return age;               }               set               {                   age = value;                   NotifyPropertyChange("Age");               }           }             public event PropertyChangedEventHandler PropertyChanged;             private void NotifyPropertyChange(string propertyName)           {               if (PropertyChanged != null)                   PropertyChanged(this, new PropertyChangedEventArgs(propertyName));           }       }   } |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <Window x:Class="ListViewDemo.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           Title="MainWindow" Height="350" Width="525">       <Grid>           <ListView x:Name="listview1">               <ListView.View>                   <GridView>                       <GridViewColumn Header="姓名" DisplayMemberBinding="{Binding Path=Name}"/>                       <GridViewColumn Header="年龄" DisplayMemberBinding="{Binding Path=Age}"/>                   </GridView>               </ListView.View>           </ListView>           <Button Content="更改名字" Click="Button\_Click" Margin="315,174,35,103" />           <TextBox Height="23" HorizontalAlignment="Left" Margin="162,180,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" />       </Grid>   </Window> |



我们只要输入名字，然后点击”button按钮”，最后ListView同步更新了，是不是很神奇的说。

五：依赖属性

   依赖属性是wpf中独有的一种属性，前面文章中或许我们发现WPF的类定义中满是这些玩意，比如我们看一个TextBlock。

[](http://img.ddvip.com/2012/1012/201210120403139885.png)

这 些Property为后缀的都是叫做依赖属性，不过依赖属性这些东西深究起来内容还是比较多的，不过我还是讲究应用方面，有时候我们可能有这样的需求，就 是希望能在TextBlock上显示当前时间，这时我们就可以扩展TextBlock，在其中增加一个TimeProperty的依赖属性来显示当前时 间。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | using System;   using System.Collections.Generic;   using System.Linq;   using System.Text;   using System.Windows;   using System.Windows.Controls;   using System.Windows.Data;   using System.Windows.Documents;   using System.Windows.Input;   using System.Windows.Media;   using System.Windows.Media.Imaging;   using System.Windows.Navigation;   using System.Windows.Shapes;     namespace WpfApplication12   {       /// <summary>       /// MainWindow.xaml 的交互逻辑       /// </summary>       public partial class MainWindow : Window       {           public MainWindow()           {               InitializeComponent();           }       }         public class CustomTextBlock : TextBlock       {           //自定义一个依赖项属性           public static DependencyProperty TimeProperty = DependencyProperty.Register("Timer", typeof(DateTime),                                                          typeof(CustomTextBlock),                                                          new PropertyMetadata(DateTime.Now, OnTimerPropertyChanged),                                                          ValidateTimeValue);           /// <summary>           /// 对依赖属性进行设置值           /// </summary>           public DateTime Time           {               get               {                   //获取当前属性值                   return (DateTime)GetValue(TimeProperty);               }               set               {                   //给当前的属性赋值                   SetValue(TimeProperty, value);               }           }               static void OnTimerPropertyChanged(DependencyObject obj, DependencyPropertyChangedEventArgs args)           {             }             static bool ValidateTimeValue(object obj)           {               DateTime dt = (DateTime)obj;                 if (dt.Year > 1990 && dt.Year < 2200)                   return true;               return false;           }         }   } |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9 | <Window x:Class="WpfApplication12.MainWindow"           xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"           xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"           xmlns:local="clr-namespace:WpfApplication12"           Title="MainWindow" Height="350" Width="525">       <Grid>           <local:CustomTextBlock Text="{Binding RelativeSource={RelativeSource Self}, Path=Timer}"/>       </Grid>   </Window> |

http://img.ddvip.com/2012/1012/201210120403149816.png

# WPF换肤之一：创建圆角窗体

我们都期望自己的软件能够有一套看上去很吸引人眼球的外衣，使得别人看上去既专业又有美感。这个系列就带领着大家一步一步的讲解如何设计出一套自己的WPF的窗体皮肤，如果文中有任何错误或者不足，还请指出。

   WPF是微软大战略中的一个重心所在，学习WPF可谓是一举多得：首先，学习WPF可以让你了解SilverLight的80%；其次，XAML语言可 以让你快速的入手WCF和WF；更甚者，就是WPF给予DX渲染核心，抛弃了传统WINFORM以GDI+为主的渲染方式，使得界面更新更流畅，彻底解决 了闪烁问题，同时也使得代码和界面显示分离，这和以前WINFORM方式完全不同。

   那么，本节开始就以最简单的方式来介绍如何创建一个圆角的窗体。

简单介绍XAML

   首先，让我来说一下XAML，因为只有了解了它，我们才能够更加深入的来介绍如何创建圆角窗体。

   所谓的XAML就是一坨类似xml的可扩展应用程序的标记语言，它能够详细的描述用户界面元素，比如说TextBox，Button等等的信息，以便做 到精确的设置或者定位。具体的大家还是参考一些其他的资料吧，反正记住XAML和HTML类似，都是描述界面元素的特定语言而已。

   在WINFORM中，我们自绘圆角窗体的时候，都是先把窗体置为FormBorderStyle为None的状态，即无窗体模式；然后我们会结合后台代 码先绘制出窗体的区域，然后通过Win32函数里面的CreateRoundRectRgn()方法进行切割；最后通过WndProc(ref Message m)处理窗体界面消息即可。

创建圆角窗体

   在WPF中，创建圆角窗体相当简单，根本不需要那么多的步骤，下面看我的操作：

   首先，我们需要设置WPF的窗体样式为None，这个和Winform设计差不多，就是设置WindowStyle为None即可，这样当我们运行F5就可以看到一个无边框的窗体。

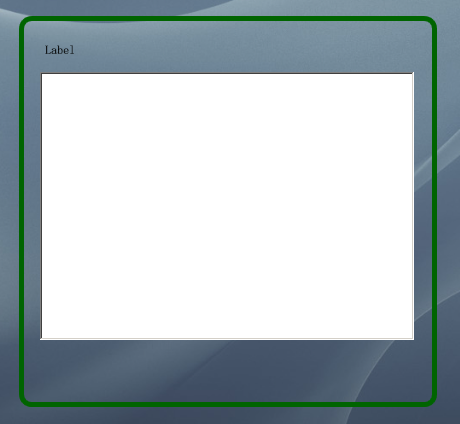
   其次，我们需要设置AllowTransParency为True，BackGround为Transparent，OpacityMask为 White，这样设置才能保证当我们设置为圆角的时候，四个角能够透明显示，这样当我们F5运行的时候，将看不到任何窗体，因为已经透明了。

   最后，怎么添加圆角呢，这个很简单，直接利用Border元素即可，加入如下代码：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <Window x:Class="WpfApplication1.MsgWindow"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      Title="TestWindow" Height="391" Width="418" WindowStyle="None" AllowsTransparency="True" Background="Transparent" OpacityMask="White" ResizeMode="NoResize" PreviewMouseMove="ResetCursor" WindowStartupLocation="CenterScreen">      <Grid Background="Transparent">              <Border BorderThickness="5" BorderBrush="DarkGreen"  CornerRadius="10,10,10,10" MouseMove="DisplayResizeCursor" PreviewMouseDown="Resize" Name="top">              <Grid>                  <ListBox Margin="16,51,18,62" Name="uniqueMessages" />                  <Label Height="28" HorizontalAlignment="Left" Margin="16,17,0,0" Name="label1" VerticalAlignment="Top" Width="120">Label</Label>              </Grid>              </Border>      </Grid>  </Window> |

其中，BorderThickness用来表示Border的粗细程度，BorderBrush用来表示Border的颜色，CornerRadius表明四个角的弯曲度。

   效果如下：



   可以看到，整个窗体居然是透明的，圆角部分显示的很完美。控件放在了透明的窗体上，这样也就预示我们成功了一半。

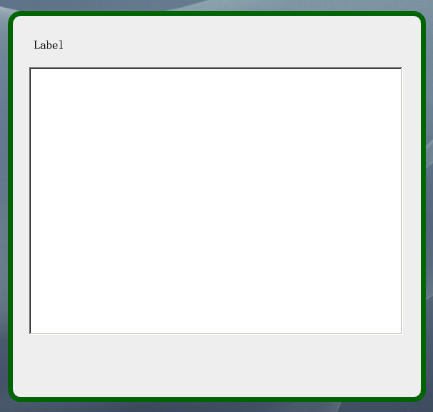
   接下来，我们设置Border的Background，用以遮蔽窗体中无需透明的部分。

代码如下：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <Window x:Class="WpfApplication1.MsgWindow"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      Title="TestWindow" Height="391" Width="418" WindowStyle="None" AllowsTransparency="True" Background="Transparent" OpacityMask="White" ResizeMode="NoResize" PreviewMouseMove="ResetCursor" WindowStartupLocation="CenterScreen">      <Grid Background="Transparent">              <Border BorderThickness="5" BorderBrush="DarkGreen"  CornerRadius="10,10,10,10" MouseMove="DisplayResizeCursor" PreviewMouseDown="Resize" Name="top">              <Border.Background>                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                      <GradientStop Color="#eee"/>                  </LinearGradientBrush>              </Border.Background>              <Grid>                  <ListBox Margin="16,51,18,62" Name="uniqueMessages" />                  <Label Height="28" HorizontalAlignment="Left" Margin="16,17,0,0" Name="label1" VerticalAlignment="Top" Width="120">Label</Label>              </Grid>              </Border>      </Grid>  </Window> |

其中，LinearGradientBrush用来绘制border区域的背景，以便遮蔽透明窗体部分。

   这样当我们再运行的时候，就可以看到效果了：



好了，这个窗体虽然完成了，但是如何进行拖拽而改变其大小呢？下节将继续讲解。

让我们接着上一章： WPF换肤之一：创建圆角窗体 来继续。

在这一章，我主要是实现对圆角窗体的拖动，改变大小功能。

拖动自绘窗体的步骤

首先，通过上节的设计，我们知道了如何设计一个圆角窗体，通过XAML代码量，我们发现设置这个窗体是多么的简单。但是如何让窗体能够进行Resize呢？

在Winform时代，我们通过WndProc(ref Message m)处理窗体界面消息来实现，那么在WPF中是否也是如此呢？

其实在WPF中，虽说封装比较紧密，但是对于处理界面消息这块，和WINFORM一样，未有所改变。下面请看具体设计：

首先，由于要涉及到和Win32交互，我们需要订阅SourceInitialized事件。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5 | public MsgWindow()          {              InitializeComponent();              this.SourceInitialized += new EventHandler(WSInitialized);          } |

然后，由于涉及到SourceInitialized Event，我们就需要使用到HwndSource，它主要功能就是WPF放入到Win32窗体中。让我们看看WindowSourceInitialized事件：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5 | void WSInitialized(object sender, EventArgs e)          {              hs = PresentationSource.FromVisual(this) as HwndSource;              hs.AddHook(new HwndSourceHook(WndProc));          } |

接下来我们看到，窗体Hook了一个 HwndSourceHook的委托，这个委托能够接受所有经过Windows的消息。我们来看看WndProc函数：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Dictionary<int, int> messages = new Dictionary<int, int>();            private IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)          {              Debug.Print(string.Format("窗体消息: {0}, wParam: {1} , lParam: {2}", msg.ToString(), wParam.ToString(), lParam.ToString()));              if (messages.ContainsKey(msg) == false)              {                  messages.Add(msg, msg);                  // 添加接收到的WIndows信息到ListBox中                  listMsg.Items.Add(msg);              }              return new IntPtr(0);          } |

这个函数会接收到所有windows消息，打印到Debug台上。

接下来，知道了事件处理流程，我们开始讨论拖拉窗体的问题。

首先，我们先给窗体添加一个ResetCursor事件，以便于拖拉结束后，恢复鼠标指针：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4 | <Window x:Class="WpfApplication1.MsgWindow"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      Title="TestWindow" Height="391" Width="418" WindowStyle="None" AllowsTransparency="True" Background="Transparent" OpacityMask="White" ResizeMode="NoResize" PreviewMouseMove="ResetCursor" WindowStartupLocation="CenterScreen"> |

其次，我们给Border元素添加一个MouseMove事件，用来显示鼠标特定情况下的鼠标指针形状（如达到了窗体边缘，则变换为拖拉的鼠标形状），同时添加一个PreviewMouseDown事件，用来进行Resize操作（也就是鼠标左键按下，开始进行拖放）：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <Border BorderThickness="5" BorderBrush="DarkGreen"  CornerRadius="10,10,10,10" MouseMove="DisplayResizeCursor" PreviewMouseDown="Resize" Name="top"> |

这样，当事件添加好以后，我们转换到后台代码：

由于窗体总共有八个地方可以进行拖拉，分别是Top，TopRight，Right，BottomRight,Bottom,BottomLeft,Left,TopLeft,那么我们先声明一个Enum：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11 | public enum ResizeDirection          {              Left = 1,              Right = 2,              Top = 3,              TopLeft = 4,              TopRight = 5,              Bottom = 6,              BottomLeft = 7,              BottomRight = 8,          } |

在Win32中，由于61440+1 代表左边，61440+2代表右边，一次类推，所以我们需要进行如下设计：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7 | [DllImport("user32.dll", CharSet = CharSet.Auto)]          private static extern IntPtr SendMessage(IntPtr hWnd, uint Msg, IntPtr wParam, IntPtr lParam);            private void ResizeWindow(ResizeDirection direction)          {              SendMessage(hs.Handle, WM\_SYSCOMMAND, (IntPtr)(61440 + direction), IntPtr.Zero);          } |

其中，WM\_SYSCOMMAND为Int类型，初始值为0x112，它的解释如下：

|  |  |  |
| --- | --- | --- |
| **WM\_SYSCOMMAND** | 0x112 | A window   receives this message when the user chooses a command from the Window menu   (formerly known as the system or control menu) or when the user chooses the   maximize button, minimize button, restore button, or close button. |

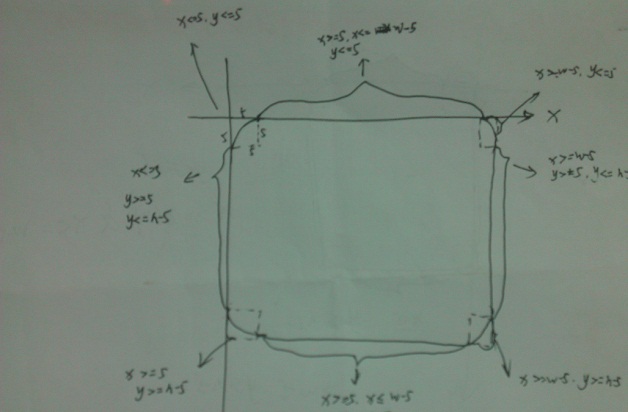
这样，通过上面的函数，我们就可以实现窗体的Resize，下面我们来响应鼠标事件：

首先是窗体的ResetCursor事件，这个主要是用来恢复鼠标形状：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7 | private void ResetCursor(object sender, MouseEventArgs e)          {              if (Mouse.LeftButton != MouseButtonState.Pressed)              {                  this.Cursor = Cursors.Arrow;              }          } |

然后我们来看看DisplayResizeCursor事件，它主要是用来改变鼠标形状，当鼠标达到一定区域，则显示拖拉的鼠标形状(<->)：

其计算方式，请参看下图：

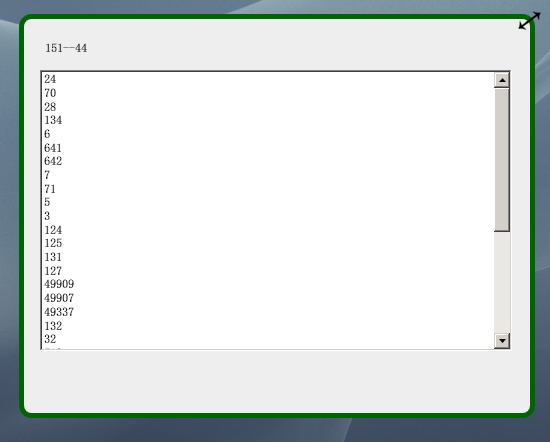
[](http://img.ddvip.com/2012/0802/201208021224517493.jpg)

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | private void DisplayResizeCursor(object sender, MouseEventArgs e)          {              Border clickBorder = sender as Border;                Point pos = Mouse.GetPosition(this);              double x = pos.X;              double y = pos.Y;              double w= this.Width;              double h= this.Height;                this.label1.Content = x + "--" + y;                if (x <= relativeClip & y <= relativeClip) // left top              {                  this.Cursor = Cursors.SizeNWSE;              }              if (x >= w - relativeClip & y <= relativeClip) //right top              {                  this.Cursor = Cursors.SizeNESW;              }                if (x >= w - relativeClip & y >= h - relativeClip) //bottom right              {                  this.Cursor = Cursors.SizeNWSE;              }                if (x <= relativeClip & y >= h - relativeClip)  // bottom left              {                  this.Cursor = Cursors.SizeNESW;              }                if ((x >= relativeClip & x <= w - relativeClip) & y <= relativeClip) //top              {                  this.Cursor = Cursors.SizeNS;              }                if (x >= w - relativeClip & (y >= relativeClip & y <= h - relativeClip)) //right              {                  this.Cursor = Cursors.SizeWE;              }                if ((x >= relativeClip & x <= w - relativeClip) & y > h - relativeClip) //bottom              {                  this.Cursor = Cursors.SizeNS;              }                if (x <= relativeClip & (y <= h - relativeClip & y >= relativeClip)) //left              {                  this.Cursor = Cursors.SizeWE;              }          } |

最后就是Resize的函数，和上面的计算方式类似，只是拖拉的时候需要调用ResizeWindow函数来改变大小：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | private void Resize(object sender, MouseButtonEventArgs e)         {             Border clickedBorder = sender as Border;               Point pos = Mouse.GetPosition(this);             double x = pos.X;             double y = pos.Y;             double w = this.Width;             double h = this.Height;               if (x <= relativeClip & y <= relativeClip) // left top             {                 this.Cursor = Cursors.SizeNWSE;                 ResizeWindow(ResizeDirection.TopLeft);             }             if (x >= w - relativeClip & y <= relativeClip) //right top             {                 this.Cursor = Cursors.SizeNESW;                 ResizeWindow(ResizeDirection.TopRight);             }               if (x >= w - relativeClip & y >= h - relativeClip) //bottom right             {                 this.Cursor = Cursors.SizeNWSE;                 ResizeWindow(ResizeDirection.BottomRight);             }               if (x <= relativeClip & y >= h - relativeClip)  // bottom left             {                 this.Cursor = Cursors.SizeNESW;                 ResizeWindow(ResizeDirection.BottomLeft);             }               if ((x >= relativeClip & x <= w - relativeClip) & y <= relativeClip) //top             {                 this.Cursor = Cursors.SizeNS;                 ResizeWindow(ResizeDirection.Top);             }               if (x >= w - relativeClip & (y >= relativeClip & y <= h - relativeClip)) //right             {                 this.Cursor = Cursors.SizeWE;                 ResizeWindow(ResizeDirection.Right);             }               if ((x >= relativeClip & x <= w - relativeClip) & y > h - relativeClip) //bottom             {                 this.Cursor = Cursors.SizeNS;                 ResizeWindow(ResizeDirection.Bottom);             }               if (x <= relativeClip & (y <= h - relativeClip & y >= relativeClip)) //left             {                 this.Cursor = Cursors.SizeWE;                 ResizeWindow(ResizeDirection.Left);             }         } |

最后效果图如下所示：



在上篇文章中，我有提到过WndProc中可以处理所有经过窗体的事件，但是没有具体的来说怎么可以处理的。

其实，在WPF中，要想 利用WndProc来处理所有的事件，需要利用到SourceInitialized  Event，首先需要创建一个HwndSource对象，然后利用其AddHook方法来将所有的windows消息附加到一个现有的事件中，这个就是 WndProc。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5 | void WSInitialized(object sender, EventArgs e)          {              hs = PresentationSource.FromVisual(this) as HwndSource;              hs.AddHook(new HwndSourceHook(WndProc));          } |

这样，我们就成功地添加了一个可以接收所有windows消息的函数，那么有了它，就让我们用它来做一些有意义的事情吧。

在WPF设计过程中，我是利用一个无边框窗体进行了重绘。所以当我设置其最大化时，肯定是要遮蔽任务栏的：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | this.WindowState = (this.WindowState == WindowState.Normal ? WindowState.Maximized : WindowState.Normal); |
| 双击代码全选 | |
| 1 | 下面就让我们来实现不遮蔽任务栏（参考文章：<a href="<http://blogs.msdn.com/b/llobo/archive/2006/08/01/maximizing-window-_2800_with-windowstyle_3d00_none_2900_-considering-taskbar.aspx>">Maximizing window (with WindowStyle=None) considering Taskbar</a>）。 |

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157 | #region 这一部分用于最大化时不遮蔽任务栏          private static void WmGetMinMaxInfo(System.IntPtr hwnd, System.IntPtr lParam)          {                MINMAXINFO mmi = (MINMAXINFO)Marshal.PtrToStructure(lParam, typeof(MINMAXINFO));                // Adjust the maximized size and position to fit the work area of the correct monitor              int MONITOR\_DEFAULTTONEAREST = 0x00000002;              System.IntPtr monitor = MonitorFromWindow(hwnd, MONITOR\_DEFAULTTONEAREST);                if (monitor != System.IntPtr.Zero)              {                    MONITORINFO monitorInfo = new MONITORINFO();                  GetMonitorInfo(monitor, monitorInfo);                  RECT rcWorkArea = monitorInfo.rcWork;                  RECT rcMonitorArea = monitorInfo.rcMonitor;                  mmi.ptMaxPosition.x = Math.Abs(rcWorkArea.left - rcMonitorArea.left);                  mmi.ptMaxPosition.y = Math.Abs(rcWorkArea.top - rcMonitorArea.top);                  mmi.ptMaxSize.x = Math.Abs(rcWorkArea.right - rcWorkArea.left);                  mmi.ptMaxSize.y = Math.Abs(rcWorkArea.bottom - rcWorkArea.top);              }                Marshal.StructureToPtr(mmi, lParam, true);          }            /// <summary>          /// POINT aka POINTAPI          /// </summary>          [StructLayout(LayoutKind.Sequential)]          public struct POINT          {              /// <summary>              /// x coordinate of point.              /// </summary>              public int x;              /// <summary>              /// y coordinate of point.              /// </summary>              public int y;                /// <summary>              /// Construct a point of coordinates (x,y).              /// </summary>              public POINT(int x, int y)              {                  this.x = x;                  this.y = y;              }          }            [StructLayout(LayoutKind.Sequential)]          public struct MINMAXINFO          {              public POINT ptReserved;              public POINT ptMaxSize;              public POINT ptMaxPosition;              public POINT ptMinTrackSize;              public POINT ptMaxTrackSize;          };          /// <summary> Win32 </summary>          [StructLayout(LayoutKind.Sequential, Pack = 0)]          public struct RECT          {              /// <summary> Win32 </summary>              public int left;              /// <summary> Win32 </summary>              public int top;              /// <summary> Win32 </summary>              public int right;              /// <summary> Win32 </summary>              public int bottom;                /// <summary> Win32 </summary>              public static readonly RECT Empty = new RECT();                /// <summary> Win32 </summary>              public int Width              {                  get { return Math.Abs(right - left); }  // Abs needed for BIDI OS              }              /// <summary> Win32 </summary>              public int Height              {                  get { return bottom - top; }              }                /// <summary> Win32 </summary>              public RECT(int left, int top, int right, int bottom)              {                  this.left = left;                  this.top = top;                  this.right = right;                  this.bottom = bottom;              }                  /// <summary> Win32 </summary>              public RECT(RECT rcSrc)              {                  this.left = rcSrc.left;                  this.top = rcSrc.top;                  this.right = rcSrc.right;                  this.bottom = rcSrc.bottom;              }                /// <summary> Win32 </summary>              public bool IsEmpty              {                  get                  {                      // BUGBUG : On Bidi OS (hebrew arabic) left > right                      return left >= right || top >= bottom;                  }              }              /// <summary> Return a user friendly representation of this struct </summary>              public override string ToString()              {                  if (this == RECT.Empty) { return "RECT {Empty}"; }                  return "RECT { left : " + left + " / top : " + top + " / right : " + right + " / bottom : " + bottom + " }";              }                /// <summary> Determine if 2 RECT are equal (deep compare) </summary>              public override bool Equals(object obj)              {                  if (!(obj is Rect)) { return false; }                  return (this == (RECT)obj);              }                /// <summary>Return the HashCode for this struct (not garanteed to be unique)</summary>              public override int GetHashCode()              {                  return left.GetHashCode() + top.GetHashCode() + right.GetHashCode() + bottom.GetHashCode();              }                  /// <summary> Determine if 2 RECT are equal (deep compare)</summary>              public static bool operator ==(RECT rect1, RECT rect2)              {                  return (rect1.left == rect2.left && rect1.top == rect2.top && rect1.right == rect2.right && rect1.bottom == rect2.bottom);              }                /// <summary> Determine if 2 RECT are different(deep compare)</summary>              public static bool operator !=(RECT rect1, RECT rect2)              {                  return !(rect1 == rect2);              }          }            [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]          public class MONITORINFO          {              /// <summary>              /// </summary>              public int cbSize = Marshal.SizeOf(typeof(MONITORINFO));                /// <summary>              /// </summary>              public RECT rcMonitor = new RECT();                /// <summary>              /// </summary>              public RECT rcWork = new RECT();                /// <summary>              /// </summary>              public int dwFlags = 0;          }            [DllImport("user32")]          internal static extern bool GetMonitorInfo(IntPtr hMonitor, MONITORINFO lpmi);            [DllImport("User32")]          internal static extern IntPtr MonitorFromWindow(IntPtr handle, int flags);          #endregion |

上面这部分主要就是通过显示器信息来确定窗体显示的WorkArea和MonitorArea。

上面的函数准备好以后，下面就开始处理最大化按钮事件：

首先，让我们来看一个常量：

|  |  |  |
| --- | --- | --- |
| **WM\_GETMINMAXINFO** | 0x24 | The   WM\_GETMINMAXINFO message is sent to a window when the size or position of the   window is about to change. An application can use this message to override   the window's default maximized size and position, or its default minimum or   maximum tracking size. |

从字面意思看来就是这个消息是用来处理窗体大小或者是位置变化的。程序可以使用这个消息来重载原本存在的最大化信息，位置信息等等。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12 | private IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)          {              switch (msg)              {                  case 0x0024:/\* WM\_GETMINMAXINFO \*/                      WmGetMinMaxInfo(hwnd, lParam);                      handled = true;                      break;  default: break;              }              return (System.IntPtr)0;          } |
| 双击代码全选 | |
| 1 | 上面的代码就是通过处理<strong>WM\_GETMINMAXINFO</strong>消息来实现最大化时不遮蔽任务栏。 我们来看看效果： |

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <a href="<http://img.ddvip.com/2012/0802/201208021231425210.png>" target="\_blank"><img src="<http://img.ddvip.com/2012/0802/201208021231425210.png>" height="393" width="700"></a> |
| 双击代码全选 | |
| 1 | 需要补充一点的是，在WindowForm中，我们可以通过Point p = new Point(lParam.ToInt32())来确定我们的鼠标坐标在窗体的哪个位置上，但是在WPF中，Point没有带有单个参数的方法，这里只能通过 |

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2 | int x = lParam.ToInt32() & 0xffff;                     int y = lParam.ToInt32() >> 16; |

来获取。

希望这篇文章对你有用。

# WPF换肤之四：界面设计和代码设计分离

说起WPF来，除了总所周知的图形处理核心的变化外，和Winform比起来，还有一个巨大的变革，那就是真正意义上做到了界面设计和代码设计的分离。这样可以让美工和程序分开进行，而不是糅合在一块，这样做的好处当然也是显而易见的：提高了开发效率。

原先的设计方式

在我们之前设计的代码中，每当添加一个新的窗体的时候，我总是会在这个新的窗体的XAML文件中加入如下的代码，以便使样式能够应用上去：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | View Code    <Window x:Class="WpfApplication1.MsgWindow"      xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"      Title="TestWindow" Height="391" Width="418" WindowStyle="None" AllowsTransparency="True" Background="Transparent" OpacityMask="White" ResizeMode="NoResize" PreviewMouseMove="ResetCursor" WindowStartupLocation="CenterScreen">      <Grid Background="Transparent">              <Border BorderThickness="5" BorderBrush="DarkGreen"  CornerRadius="10,10,10,10" MouseMove="DisplayResizeCursor" PreviewMouseDown="Resize" Name="top">              <Border.Background>                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                      <GradientStop Color="#eee"/>                  </LinearGradientBrush>              </Border.Background>              <Grid>                 <!--这里放置UIElement.-->              </Grid>              </Border>      </Grid>  </Window> |

然后，在后台中，为了使窗体能够在最大化时不遮蔽任务栏，拖拉窗体边缘能够改变窗口大小，点按窗体可以实现拖拉的时候，在后台加入了如下的代码：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343 | View Code    using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Data;  using System.Windows.Documents;  using System.Windows.Input;  using System.Windows.Media;  using System.Windows.Media.Imaging;  using System.Windows.Shapes;  using System.Windows.Interop;  using System.Diagnostics;  using System.Runtime.InteropServices;    namespace WpfApplication1  {      /// <summary>      /// Interaction logic for TestWindow.xaml        /// </summary>      public partial class MsgWindow : Window      {          private const int WM\_SYSCOMMAND = 0x112;          private HwndSource hs;          IntPtr retInt = IntPtr.Zero;            public MsgWindow()          {              InitializeComponent();              this.SourceInitialized += new EventHandler(WSInitialized);          }            void WSInitialized(object sender, EventArgs e)          {              hs = PresentationSource.FromVisual(this) as HwndSource;              hs.AddHook(new HwndSourceHook(WndProc));          }           public double relativeClip = 10;            public enum ResizeDirection          {              Left = 1,              Right = 2,              Top = 3,              TopLeft = 4,              TopRight = 5,              Bottom = 6,              BottomLeft = 7,              BottomRight = 8,          }            [DllImport("user32.dll", CharSet = CharSet.Auto)]          private static extern IntPtr SendMessage(IntPtr hWnd, uint Msg, IntPtr wParam, IntPtr lParam);            private void ResizeWindow(ResizeDirection direction)          {              SendMessage(hs.Handle, WM\_SYSCOMMAND, (IntPtr)(61440 + direction), IntPtr.Zero);          }            private void ResetCursor(object sender, MouseEventArgs e)          {              if (Mouse.LeftButton != MouseButtonState.Pressed)              {                  this.Cursor = Cursors.Arrow;              }          }            private void Resize(object sender, MouseButtonEventArgs e)          {              Border clickedBorder = sender as Border;                Point pos = Mouse.GetPosition(this);              double x = pos.X;              double y = pos.Y;              double w = this.ActualWidth;              double h = this.ActualHeight;                if (x <= relativeClip & y <= relativeClip) // left top              {                  this.Cursor = Cursors.SizeNWSE;                  ResizeWindow(ResizeDirection.TopLeft);              }              if (x >= w - relativeClip & y <= relativeClip) //right top              {                  this.Cursor = Cursors.SizeNESW;                  ResizeWindow(ResizeDirection.TopRight);              }                if (x >= w - relativeClip & y >= h - relativeClip) //bottom right              {                  this.Cursor = Cursors.SizeNWSE;                  ResizeWindow(ResizeDirection.BottomRight);              }                if (x <= relativeClip & y >= h - relativeClip)  // bottom left              {                  this.Cursor = Cursors.SizeNESW;                  ResizeWindow(ResizeDirection.BottomLeft);              }                if ((x >= relativeClip & x <= w - relativeClip) & y <= relativeClip) //top              {                  this.Cursor = Cursors.SizeNS;                  ResizeWindow(ResizeDirection.Top);              }                if (x >= w - relativeClip & (y >= relativeClip & y <= h - relativeClip)) //right              {                  this.Cursor = Cursors.SizeWE;                  ResizeWindow(ResizeDirection.Right);              }                if ((x >= relativeClip & x <= w - relativeClip) & y > h - relativeClip) //bottom              {                  this.Cursor = Cursors.SizeNS;                  ResizeWindow(ResizeDirection.Bottom);              }                if (x <= relativeClip & (y <= h - relativeClip & y >= relativeClip)) //left              {                  this.Cursor = Cursors.SizeWE;                  ResizeWindow(ResizeDirection.Left);              }          }            private void DisplayResizeCursor(object sender, MouseEventArgs e)          {              Border clickBorder = sender as Border;                Point pos = Mouse.GetPosition(this);              double x = pos.X;              double y = pos.Y;              double w= this.ActualWidth;              double h= this.ActualHeight;                this.label1.Content = x + "--" + y;                if (x <= relativeClip & y <= relativeClip) // left top              {                  this.Cursor = Cursors.SizeNWSE;              }              if (x >= w - relativeClip & y <= relativeClip) //right top              {                  this.Cursor = Cursors.SizeNESW;              }                if (x >= w - relativeClip & y >= h - relativeClip) //bottom right              {                  this.Cursor = Cursors.SizeNWSE;              }                if (x <= relativeClip & y >= h - relativeClip)  // bottom left              {                  this.Cursor = Cursors.SizeNESW;              }                if ((x >= relativeClip & x <= w - relativeClip) & y <= relativeClip) //top              {                  this.Cursor = Cursors.SizeNS;              }                if (x >= w - relativeClip & (y >= relativeClip & y <= h - relativeClip)) //right              {                  this.Cursor = Cursors.SizeWE;              }                if ((x >= relativeClip & x <= w - relativeClip) & y > h - relativeClip) //bottom              {                  this.Cursor = Cursors.SizeNS;              }                if (x <= relativeClip & (y <= h - relativeClip & y >= relativeClip)) //left              {                  this.Cursor = Cursors.SizeWE;              }          }            private void button1\_Click(object sender, RoutedEventArgs e)          {              this.WindowState = (this.WindowState == WindowState.Normal ? WindowState.Maximized : WindowState.Normal);          }          #region 这一部分用于最大化时不遮蔽任务栏          private static void WmGetMinMaxInfo(System.IntPtr hwnd, System.IntPtr lParam)          {                MINMAXINFO mmi = (MINMAXINFO)Marshal.PtrToStructure(lParam, typeof(MINMAXINFO));                // Adjust the maximized size and position to fit the work area of the correct monitor              int MONITOR\_DEFAULTTONEAREST = 0x00000002;              System.IntPtr monitor = MonitorFromWindow(hwnd, MONITOR\_DEFAULTTONEAREST);                if (monitor != System.IntPtr.Zero)              {                    MONITORINFO monitorInfo = new MONITORINFO();                  GetMonitorInfo(monitor, monitorInfo);                  RECT rcWorkArea = monitorInfo.rcWork;                  RECT rcMonitorArea = monitorInfo.rcMonitor;                  mmi.ptMaxPosition.x = Math.Abs(rcWorkArea.left - rcMonitorArea.left);                  mmi.ptMaxPosition.y = Math.Abs(rcWorkArea.top - rcMonitorArea.top);                  mmi.ptMaxSize.x = Math.Abs(rcWorkArea.right - rcWorkArea.left);                  mmi.ptMaxSize.y = Math.Abs(rcWorkArea.bottom - rcWorkArea.top);              }                Marshal.StructureToPtr(mmi, lParam, true);          }            /// <summary>          /// POINT aka POINTAPI          /// </summary>          [StructLayout(LayoutKind.Sequential)]          public struct POINT          {              /// <summary>              /// x coordinate of point.              /// </summary>              public int x;              /// <summary>              /// y coordinate of point.              /// </summary>              public int y;                /// <summary>              /// Construct a point of coordinates (x,y).              /// </summary>              public POINT(int x, int y)              {                  this.x = x;                  this.y = y;              }          }            /// <summary>          /// 窗体大小信息          /// </summary>          [StructLayout(LayoutKind.Sequential)]          public struct MINMAXINFO          {              public POINT ptReserved;              public POINT ptMaxSize;              public POINT ptMaxPosition;              public POINT ptMinTrackSize;              public POINT ptMaxTrackSize;          };          /// <summary> Win32 </summary>          [StructLayout(LayoutKind.Sequential, Pack = 0)]          public struct RECT          {              /// <summary> Win32 </summary>              public int left;              /// <summary> Win32 </summary>              public int top;              /// <summary> Win32 </summary>              public int right;              /// <summary> Win32 </summary>              public int bottom;                /// <summary> Win32 </summary>              public static readonly RECT Empty = new RECT();                /// <summary> Win32 </summary>              public int Width              {                  get { return Math.Abs(right - left); }  // Abs needed for BIDI OS              }              /// <summary> Win32 </summary>              public int Height              {                  get { return bottom - top; }              }                /// <summary> Win32 </summary>              public RECT(int left, int top, int right, int bottom)              {                  this.left = left;                  this.top = top;                  this.right = right;                  this.bottom = bottom;              }                  /// <summary> Win32 </summary>              public RECT(RECT rcSrc)              {                  this.left = rcSrc.left;                  this.top = rcSrc.top;                  this.right = rcSrc.right;                  this.bottom = rcSrc.bottom;              }                /// <summary> Win32 </summary>              public bool IsEmpty              {                  get                  {                      // BUGBUG : On Bidi OS (hebrew arabic) left > right                      return left >= right || top >= bottom;                  }              }              /// <summary> Return a user friendly representation of this struct </summary>              public override string ToString()              {                  if (this == RECT.Empty) { return "RECT {Empty}"; }                  return "RECT { left : " + left + " / top : " + top + " / right : " + right + " / bottom : " + bottom + " }";              }                /// <summary> Determine if 2 RECT are equal (deep compare) </summary>              public override bool Equals(object obj)              {                  if (!(obj is Rect)) { return false; }                  return (this == (RECT)obj);              }                /// <summary>Return the HashCode for this struct (not garanteed to be unique)</summary>              public override int GetHashCode()              {                  return left.GetHashCode() + top.GetHashCode() + right.GetHashCode() + bottom.GetHashCode();              }                  /// <summary> Determine if 2 RECT are equal (deep compare)</summary>              public static bool operator ==(RECT rect1, RECT rect2)              {                  return (rect1.left == rect2.left && rect1.top == rect2.top && rect1.right == rect2.right && rect1.bottom == rect2.bottom);              }                /// <summary> Determine if 2 RECT are different(deep compare)</summary>              public static bool operator !=(RECT rect1, RECT rect2)              {                  return !(rect1 == rect2);              }          }            [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]          public class MONITORINFO          {              /// <summary>              /// </summary>              public int cbSize = Marshal.SizeOf(typeof(MONITORINFO));                /// <summary>              /// </summary>              public RECT rcMonitor = new RECT();                /// <summary>              /// </summary>              public RECT rcWork = new RECT();                /// <summary>              /// </summary>              public int dwFlags = 0;          }            [DllImport("user32")]          internal static extern bool GetMonitorInfo(IntPtr hMonitor, MONITORINFO lpmi);            [DllImport("User32")]          internal static extern IntPtr MonitorFromWindow(IntPtr handle, int flags);          #endregion            private void MyMacClass\_SourceInitialized(object sender, EventArgs e)          {              hs = PresentationSource.FromVisual((Visual)sender) as HwndSource;              hs.AddHook(new HwndSourceHook(WndProc));          }            private IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)          {              switch (msg)              {                  case 0x0024:/\* WM\_GETMINMAXINFO \*/                      WmGetMinMaxInfo(hwnd, lParam);                      handled = true;                      break;                  default: break;              }              return (System.IntPtr)0;          }        }  } |

如果按照上面的设计，那么每加入一个新的窗体，都要重复上面的两个步骤的话，加入一个工程中需要加入的新窗体特别多，估计这种操作足以让一个正常人疯掉了。并且假如以后border的颜色要修改，那得修改多少页面啊~~~

改进的设计方式

所以，为了便于设计和维护，实现所谓的UI和代码分析，让我们提出一个假设的方案来：

首先，所有的公共样式放到一个样式文件中，所有新加的窗体都能共享这个公共的样式文件。

其次，所有的公共事件（窗体最大化，最小化，关闭等），都放到一个公共的基类中，所有新加的窗体只要继承该基类，即可继承系统公用的事件操作。

那么，本着这个假设，我们开始来进行。

首先，对于公共样式，我们需要添加一个“资源词典”的页面，用来设计公共样式：

添加完成后，会看到如下XAML：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4 | <ResourceDictionary xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"      xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>">    </ResourceDictionary> |

下面我们在这个文件中添加样式：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80 | View Code    <ResourceDictionary xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"        xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"                        x:Class="MyOwnerDrawnWindow.Resource\_Dictionaries.MyTheme"                        >            <!-- Border defining the frame of the Window -->          <Style x:Key="MywindowBorder" TargetType="Border">            <Setter Property="CornerRadius" Value="10, 10, 10, 10" />            <Setter Property="BorderBrush" Value="DarkGreen"></Setter>            <Setter Property="BorderThickness" Value="5" />            <Setter Property="HorizontalAlignment" Value="Stretch"></Setter>            <Setter Property="VerticalAlignment" Value="Stretch"></Setter>            <Setter Property="Background" Value="#ababab"></Setter>        </Style>            <ControlTemplate x:Key="MyWindowTemplate" TargetType="{x:Type Window}">            <Grid>                <Border x:Name="MyBorder"  Style="{StaticResource MywindowBorder}" >                    <Border.Background>                        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                            <GradientStop Color="#eee"/>                        </LinearGradientBrush>                    </Border.Background>                    <!--这一句很重要，主要用于放置界面元素，和asp.net中的masterpage有点像-->                     <AdornerDecorator>                        <ContentPresenter />                    </AdornerDecorator>                </Border>            </Grid>        </ControlTemplate>            <!-- My Window Style -->        <Style x:Key="MyWindowStyle" TargetType="Window">            <Setter Property="Background" Value="Transparent" />            <Setter Property="WindowStyle" Value="None" />            <Setter Property="AllowsTransparency" Value="True" />            <Setter Property="Template" Value="{StaticResource MyWindowTemplate}" />        </Style>    </ResourceDictionary> |

好了，这就是我们的样式文件，接下来我们需要处理这个样式中的Border事件，让其支持鼠标左键拖拉功能。

新建一个类，命名为MyThemeClass.cs，让其继承自Window基类。在MyThemeClass类中，我们主要处理两个内容，一个是支持鼠标左键拖拉以便改变窗体大小，另一个是使窗体不遮蔽任务栏。

具体代码如下：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695 | View Code    using System;    using System.Collections.Generic;    using System.Linq;    using System.Text;    using System.Windows;    using System.Windows.Media;    using System.Windows.Interop;    using System.Runtime.InteropServices;    using System.Windows.Controls;    using System.Windows.Input;        namespace MyOwnerDrawnWindow    {        public class MyThemeClass:Window        {             private const int WM\_SYSCOMMAND = 0x112;            public const int WM\_LBUTTONUP = 0x0202;            private HwndSource hs;            IntPtr retInt = IntPtr.Zero;            public double relativeClip = 10;                public MyThemeClass()            {                this.Loaded += delegate                {                    InitializeEvent();                };                this.SourceInitialized +=new EventHandler(MyMacClass\_SourceInitialized);            }                private void MyMacClass\_SourceInitialized(object sender, EventArgs e)            {                hs = PresentationSource.FromVisual((Visual)sender) as HwndSource;                hs.AddHook(new HwndSourceHook(WndProc));            }                private IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)            {                switch (msg)                {                    case 0x0024:/\* WM\_GETMINMAXINFO \*/                        WmGetMinMaxInfo(hwnd, lParam);                        handled = true;                        break;                    default: break;                }                return (System.IntPtr)0;            }                #region 这一部分用于最大化时不遮蔽任务栏            private static void WmGetMinMaxInfo(System.IntPtr hwnd, System.IntPtr lParam)            {                    MINMAXINFO mmi = (MINMAXINFO)Marshal.PtrToStructure(lParam, typeof(MINMAXINFO));                    // Adjust the maximized size and position to fit the work area of the correct monitor                int MONITOR\_DEFAULTTONEAREST = 0x00000002;                System.IntPtr monitor = MonitorFromWindow(hwnd, MONITOR\_DEFAULTTONEAREST);                    if (monitor != System.IntPtr.Zero)                {                        MONITORINFO monitorInfo = new MONITORINFO();                    GetMonitorInfo(monitor, monitorInfo);                    RECT rcWorkArea = monitorInfo.rcWork;                    RECT rcMonitorArea = monitorInfo.rcMonitor;                    mmi.ptMaxPosition.x = Math.Abs(rcWorkArea.left - rcMonitorArea.left);                    mmi.ptMaxPosition.y = Math.Abs(rcWorkArea.top - rcMonitorArea.top);                    mmi.ptMaxSize.x = Math.Abs(rcWorkArea.right - rcWorkArea.left);                    mmi.ptMaxSize.y = Math.Abs(rcWorkArea.bottom - rcWorkArea.top);                }                    Marshal.StructureToPtr(mmi, lParam, true);            }                /// <summary>            /// POINT aka POINTAPI            /// </summary>            [StructLayout(LayoutKind.Sequential)]            public struct POINT            {                /// <summary>                /// x coordinate of point.                /// </summary>                public int x;                /// <summary>                /// y coordinate of point.                /// </summary>                public int y;                    /// <summary>                /// Construct a point of coordinates (x,y).                /// </summary>                public POINT(int x, int y)                {                    this.x = x;                    this.y = y;                }            }                [StructLayout(LayoutKind.Sequential)]            public struct MINMAXINFO            {                public POINT ptReserved;                public POINT ptMaxSize;                public POINT ptMaxPosition;                public POINT ptMinTrackSize;                public POINT ptMaxTrackSize;            };            [StructLayout(LayoutKind.Sequential, Pack = 0)]            public struct RECT            {                /// <summary> Win32 </summary>                public int left;                /// <summary> Win32 </summary>                public int top;                /// <summary> Win32 </summary>                public int right;                /// <summary> Win32 </summary>                public int bottom;                    /// <summary> Win32 </summary>                public static readonly RECT Empty = new RECT();                    /// <summary> Win32 </summary>                public int Width                {                    get { return Math.Abs(right - left); }  // Abs needed for BIDI OS                }                /// <summary> Win32 </summary>                public int Height                {                    get { return bottom - top; }                }                    /// <summary> Win32 </summary>                public RECT(int left, int top, int right, int bottom)                {                    this.left = left;                    this.top = top;                    this.right = right;                    this.bottom = bottom;                }                        /// <summary> Win32 </summary>                public RECT(RECT rcSrc)                {                    this.left = rcSrc.left;                    this.top = rcSrc.top;                    this.right = rcSrc.right;                    this.bottom = rcSrc.bottom;                }            }                [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]            public class MONITORINFO            {                /// <summary>                /// </summary>                public int cbSize = Marshal.SizeOf(typeof(MONITORINFO));                    /// <summary>                /// </summary>                public RECT rcMonitor = new RECT();                    /// <summary>                /// </summary>                public RECT rcWork = new RECT();                    /// <summary>                /// </summary>                public int dwFlags = 0;            }                [DllImport("user32")]            internal static extern bool GetMonitorInfo(IntPtr hMonitor, MONITORINFO lpmi);                [DllImport("User32")]            internal static extern IntPtr MonitorFromWindow(IntPtr handle, int flags);            #endregion                #region 这一部分是四个边加上四个角            public enum ResizeDirection            {                Left = 1,                Right = 2,                Top = 3,                TopLeft = 4,                TopRight = 5,                Bottom = 6,                BottomLeft = 7,                BottomRight = 8,            }            #endregion                #region 用于改变窗体大小            [DllImport("user32.dll", CharSet = CharSet.Auto)]            private static extern IntPtr SendMessage(IntPtr hWnd, uint Msg, IntPtr wParam, IntPtr lParam);                private void ResizeWindow(ResizeDirection direction)            {                SendMessage(hs.Handle, WM\_SYSCOMMAND, (IntPtr)(61440 + direction), IntPtr.Zero);            }            #endregion                #region 为元素注册事件            private void InitializeEvent()            {                ControlTemplate baseWindowTemplate = (ControlTemplate)App.Current.Resources["MyWindowTemplate"];                Border borderClip = (Border)baseWindowTemplate.FindName("MyBorder", this);                    borderClip.MouseMove += delegate                {                    DisplayResizeCursor(null,null);                };                    borderClip.PreviewMouseDown += delegate                {                    Resize(null,null);                };                    borderClip.MouseLeftButtonDown += delegate                {                    DragMove();                };                    this.PreviewMouseMove += delegate                {                    ResetCursor(null,null);                };            }            #endregion                #region 重写的DragMove，以便解决利用系统自带的DragMove出现Exception的情况            public new void DragMove()            {                if (this.WindowState == WindowState.Normal)                {                    SendMessage(hs.Handle, WM\_SYSCOMMAND, (IntPtr)0xf012, IntPtr.Zero);                    SendMessage(hs.Handle, WM\_LBUTTONUP, IntPtr.Zero, IntPtr.Zero);                }            }            #endregion                #region 显示拖拉鼠标形状            private void DisplayResizeCursor(object sender, MouseEventArgs e)            {                Point pos = Mouse.GetPosition(this);                double x = pos.X;                double y = pos.Y;                double w = this.ActualWidth;  //注意这个地方使用ActualWidth,才能够实时显示宽度变化                double h = this.ActualHeight;                    if (x <= relativeClip & y <= relativeClip) // left top                {                    this.Cursor = Cursors.SizeNWSE;                }                if (x >= w - relativeClip & y <= relativeClip) //right top                {                    this.Cursor = Cursors.SizeNESW;                }                    if (x >= w - relativeClip & y >= h - relativeClip) //bottom right                {                    this.Cursor = Cursors.SizeNWSE;                }                    if (x <= relativeClip & y >= h - relativeClip)  // bottom left                {                    this.Cursor = Cursors.SizeNESW;                }                    if ((x >= relativeClip & x <= w - relativeClip) & y <= relativeClip) //top                {                    this.Cursor = Cursors.SizeNS;                }                    if (x >= w - relativeClip & (y >= relativeClip & y <= h - relativeClip)) //right                {                    this.Cursor = Cursors.SizeWE;                }                    if ((x >= relativeClip & x <= w - relativeClip) & y > h - relativeClip) //bottom                {                    this.Cursor = Cursors.SizeNS;                }                    if (x <= relativeClip & (y <= h - relativeClip & y >= relativeClip)) //left                {                    this.Cursor = Cursors.SizeWE;                }            }            #endregion                #region  还原鼠标形状            private void ResetCursor(object sender, MouseEventArgs e)            {                if (Mouse.LeftButton != MouseButtonState.Pressed)                {                    this.Cursor = Cursors.Arrow;                }            }            #endregion                #region 判断区域，改变窗体大小            private void Resize(object sender, MouseButtonEventArgs e)            {                Point pos = Mouse.GetPosition(this);                double x = pos.X;                double y = pos.Y;                double w = this.ActualWidth;                double h = this.ActualHeight;                    if (x <= relativeClip & y <= relativeClip) // left top                {                    this.Cursor = Cursors.SizeNWSE;                    ResizeWindow(ResizeDirection.TopLeft);                }                if (x >= w - relativeClip & y <= relativeClip) //right top                {                    this.Cursor = Cursors.SizeNESW;                    ResizeWindow(ResizeDirection.TopRight);                }                    if (x >= w - relativeClip & y >= h - relativeClip) //bottom right                {                    this.Cursor = Cursors.SizeNWSE;                    ResizeWindow(ResizeDirection.BottomRight);                }                    if (x <= relativeClip & y >= h - relativeClip)  // bottom left                {                    this.Cursor = Cursors.SizeNESW;                    ResizeWindow(ResizeDirection.BottomLeft);                }                    if ((x >= relativeClip & x <= w - relativeClip) & y <= relativeClip) //top                {                    this.Cursor = Cursors.SizeNS;                    ResizeWindow(ResizeDirection.Top);                }                    if (x >= w - relativeClip & (y >= relativeClip & y <= h - relativeClip)) //right                {                    this.Cursor = Cursors.SizeWE;                    ResizeWindow(ResizeDirection.Right);                }                    if ((x >= relativeClip & x <= w - relativeClip) & y > h - relativeClip) //bottom                {                    this.Cursor = Cursors.SizeNS;                    ResizeWindow(ResizeDirection.Bottom);                }                    if (x <= relativeClip & (y <= h - relativeClip & y >= relativeClip)) //left                {                    this.Cursor = Cursors.SizeWE;                    ResizeWindow(ResizeDirection.Left);                }            }              #endregion        }    } |

这样，我们的Theme和类就创建好了，那么，在主窗体中，我们该如何应用上去呢？当然，这个不是难事，并且异常简单：

首先，创建一个新的窗体Window1, 它需要继承自MyThemeClass类：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | public partial class Window1 : MyThemeClass |

然后再XAML中只需要修改两个地方，首先是添加入一个新的引用：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | xmlns:src="clr-namespace:MyOwnerDrawnWindow" |

添加完这个引用后，把<Window….></Window>修改成<src:MyThemeClass……></ MyThemeClass> 这样做的目的是防止由于继承基类的不同而造成XAML的识别错误。

最后，只需要添加这句Style="{StaticResource MyWindowStyle}"  就行了。 这样就算是添加完成了新的窗体，每一个新添加的窗体都按照这种方式添加即可，是不是简洁了许多？ 并且后期维护也简单多了。

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | View Code    <src:MyThemeClass x:Class="MyOwnerDrawnWindow.Window1"        xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"        xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"        xmlns:src="clr-namespace:MyOwnerDrawnWindow"        Title="Window1" Height="335" Width="706"        Style="{StaticResource MyWindowStyle}">        <Grid>           <!--这里放置你的UIElement-->        </Grid>    </src:MyThemeClass> |

好了，最后一步就是讲Theme文件关联起来，在APP.xaml文件中添加对资源文件的引用即可：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | View Code    <Application x:Class="MyOwnerDrawnWindow.App"        xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>"        xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"        StartupUri="Window1.xaml">       <Application.Resources>            <ResourceDictionary>                <ResourceDictionary.MergedDictionaries>                    <ResourceDictionary Source="MyTheme.xaml"></ResourceDictionary>                </ResourceDictionary.MergedDictionaries>            </ResourceDictionary>        </Application.Resources>    </Application> |

这样运行起来以后，达到的预期的效果。

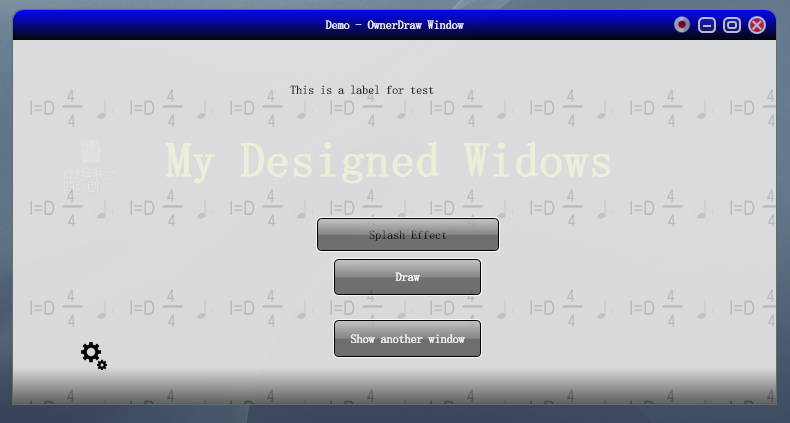
# WPF换肤之五：创建漂亮的窗体

换肤效果

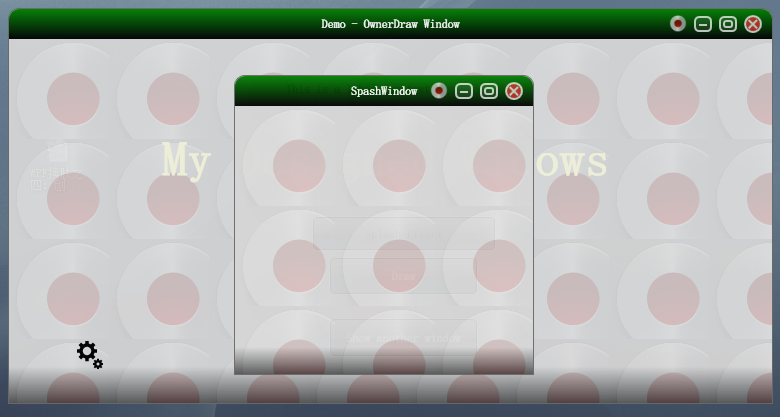
下面请看换肤效果（首先声明，窗体样式和按钮样式均来自于CodeProject，本人美工太差）：

[](http://img.ddvip.com/2012/0802/201208021248126477.png)

[](http://img.ddvip.com/2012/0802/201208021248129677.png)

[](http://img.ddvip.com/2012/0802/201208021248132888.png)

[](http://img.ddvip.com/2012/0802/201208021248136401.png)

[](http://img.ddvip.com/2012/0802/201208021248139676.png)

支持效果：

1. 窗体拖拉，可以任意拖动窗体:重写了自带的DragMove类，使得拖拉和Resize不冲突。
2. 窗体大小改变:通过WndProc接收所有的Windows消息来处理。
3. 最大化，最小化，关闭窗体:这个系统自带，但是最大化的时候会遮蔽任务栏，通过Win32函数处理。
4. 改变窗体背景图片，抛弃千篇一律的重复感：这个涉及到ImageBrush的内容，ViewPort的设置有关。

后续还需要改进，包括实现动画效果等等。

这一系列准备写下去，包括之后的线程和UI交互，异步处理长时间的操作等等，敬请关注，谢谢。

# WPF换肤之六：酷炫的时区浏览小精灵

由于工作需要,经常要查看到不同地区的 当前时间,以前总是对照着时区表来进行加减运算,现在有了这个小工具以后,感觉省心了不少.下面是软件的截图:

效果图赏析

[](http://img.ddvip.com/2012/0828/201208281222500262.jpg)

[](http://img.ddvip.com/2012/0828/201208281222503238.jpg)

[](http://img.ddvip.com/2012/0828/201208281222506119.png)

[](http://img.ddvip.com/2012/0828/201208281222509028.png)

在 界面上,有能够冉冉升起的太阳或者月亮,有缓慢飘动的浮云,有青葱翠绿的花叶,　当然,也有显目的时区显示.如果要是放在WinForm时代,要实现这样 的界面,真的是繁琐和复杂,但是在WPF中,利用XAML控制前台界面,利用CodeBehind控制窗口拖动,日月变换等等逻辑,真的是简便而且效果强 大.其基于DX渲染的界面,流畅而且逼真.

实现原理

首先,参照前面几章的讲解,我们知道要想实现窗体自绘,我们需要进行以下几项设置:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4 | AllowTransparency = “True”  WindowStyle = “None”  Background = “Transparent”  OpacityMask = “White” |

这样之后,我们需要做的是放置一个Grid容器,设置Grid容器行和列,这个我不消多说,因为和设置ASP.NET中的网格类似.

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | View Code    <Grid.ColumnDefinitions>    <ColumnDefinition/>    <ColumnDefinition/>    <ColumnDefinition/>    <ColumnDefinition/>    <ColumnDefinition/>    <ColumnDefinition/>  </Grid.ColumnDefinitions>    <Grid.RowDefinitions>    <RowDefinition Height="30"/>    <RowDefinition Height="30"/>    <RowDefinition/>  </Grid.RowDefinitions> |

上面的代码就是声明了一个3行6列的Grid容器,其中第一行和第二行的高度均为30.容器有了,下面我们开始往容器里面放东西.

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3 | <Border Grid.Column="0" Grid.Row="1">  <TextBlock x:Name="HK" Padding="0,5,0,0" Background="#FFA04C00" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock>  </Border> |

上面的就是往容器放置的东西,其中Border是用于画边线(Grid自带的边线太难看), Grid.Column="0" Grid.Row="1"　是指将这个内容放置在第１列,第二行的位置上,依次类推,然后填充即可.

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | View Code    <Border Grid.Column="0">  <TextBlock FontFamily="@微软雅黑" Padding="0,5,0,0" FontSize="12pt" Foreground="Black" FontWeight="bold" Background="#FFF87800"  TextAlignment="Center"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="1">  <TextBlock Background="#FFB0D428" Padding="0,5,0,0" FontFamily="@微软雅黑" FontSize="16" FontWeight="Bold" Foreground="Black"    Text="New York" TextAlignment="Center"/></Border>  <Border Grid.Column="2">  <TextBlock Background="#FF2283E4" Padding="0,5,0,0" FontFamily="@微软雅黑" FontSize="16" FontWeight="Bold" Foreground="Black"    Text="London" TextAlignment="Center"/></Border>  <Border Grid.Column="3">  <TextBlock Background="#FF20D4D0" Padding="0,5,0,0" FontFamily="@微软雅黑" FontSize="16" FontWeight="Bold" Foreground="Black"    Text="Pairs" TextAlignment="Center"/></Border>  <Border Grid.Column="4">  <TextBlock Background="#FF28D428" Padding="0,5,0,0" FontFamily="@微软雅黑" FontSize="16" FontWeight="Bold" Foreground="Black"    Text="Sydney" TextAlignment="Center"/></Border>  <Border Grid.Column="5">  <TextBlock Background="#FFD040F8" Padding="0,5,0,0" FontFamily="@微软雅黑" FontSize="16" FontWeight="Bold" Foreground="Black"    Text="Brasilia" TextAlignment="Center"/></Border>  <Border Grid.Column="0" Grid.Row="1">  <TextBlock x:Name="HK" Padding="0,5,0,0" Background="#FFA04C00" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="1" Grid.Row="1">  <TextBlock x:Name="NY" Padding="0,5,0,0" Background="#FF708418" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="2" Grid.Row="1">  <TextBlock x:Name="UK" Padding="0,5,0,0" Background="#FF184880" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="3" Grid.Row="1">  <TextBlock x:Name="PS" Padding="0,5,0,0" Background="#FF188480" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="4" Grid.Row="1">  <TextBlock x:Name="SD" Padding="0,5,0,0" Background="#FF188418" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="5" Grid.Row="1">  <TextBlock x:Name="BR" Padding="0,5,0,0" Background="#FF7800A8" FontFamily="@微软雅黑" FontSize="12" Foreground="White"   TextAlignment="Center" TextWrapping="NoWrap" TextTrimming="None"><Run Text="Hong Kong"/></TextBlock></Border>  <Border Grid.Column="0" Grid.Row="2">  <TextBlock x:Name="HKT" Padding="0,5,0,0" FontFamily="@Gill Sans MT" FontSize="21pt" Foreground="White"   TextAlignment="Center" FontWeight="Bold"><Run Text="Load"/></TextBlock></Border>  <Border Grid.Column="1" Grid.Row="2">  <TextBlock x:Name="NYT" Padding="0,5,0,0" FontFamily="@Gill Sans MT" FontSize="21pt" Foreground="White"   TextAlignment="Center" FontWeight="Bold"><Run Text="Load"/></TextBlock></Border>  <Border Grid.Column="2" Grid.Row="2">  <TextBlock x:Name="UKT" Padding="0,5,0,0" FontFamily="@Gill Sans MT" FontSize="21pt" Foreground="White"   TextAlignment="Center" FontWeight="Bold"><Run Text="Load"/></TextBlock></Border>  <Border Grid.Column="3" Grid.Row="2">  <TextBlock x:Name="PST" Padding="0,5,0,0" FontFamily="@Gill Sans MT" FontSize="21pt" Foreground="White"   TextAlignment="Center" FontWeight="Bold"><Run Text="Load"/></TextBlock></Border>  <Border Grid.Column="4" Grid.Row="2">  <TextBlock x:Name="SDT" Padding="0,5,0,0" FontFamily="@Gill Sans MT" FontSize="21pt" Foreground="White"   TextAlignment="Center" FontWeight="Bold"><Run Text="Load"/></TextBlock></Border>  <Border Grid.Column="5" Grid.Row="2">  <TextBlock x:Name="BRT" Padding="0,5,0,0" FontFamily="@Gill Sans MT" FontSize="21pt" Foreground="White"   TextAlignment="Center" TextDecorations="None" FontWeight="Bold" Background="Transparent"><Run Text="Load"/></TextBlock></Border> |

这样填充完成后,就能看见上面的图片了,是不是开始有型了?

[](http://img.ddvip.com/2012/0828/201208281222512335.jpg)

然后我们还需要控制其能够计算时间,这个需要用到WPF中的DispatcherTimer对象,具体的用法如下:

首先设置DispatcherTimer对象的各种属性和事件:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4 | timer = new DispatcherTimer();  timer.Interval = TimeSpan.Parse("00:00:01");  timer.IsEnabled = true;  timer.Tick+=new EventHandler(timer\_Tick); |

这里我设置的是一秒钟跳动一次,然后注册了其Tick事件,在这个事件中,我们控制界面上时间的显示:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11 | View Code    private void timer\_Tick(object sender, EventArgs e)  {  HKT.Text = DateTime.Now.ToString("HH:mm");  NYT.Text = DateTime.Now.AddHours(-12).ToString("HH:mm");  UKT.Text = DateTime.Now.AddHours(-7).ToString("HH:mm");  PST.Text = DateTime.Now.AddHours(-6).ToString("HH:mm");  SDT.Text = DateTime.Now.AddHours(2).ToString("HH:mm");  BRT.Text = DateTime.Now.AddHours(-11).ToString("HH:mm");  } |

这样每隔一分钟,时间就改变一次.

这部分介绍完之后,看上去都已经有模有样了,下面开始添加特效图片了,首先我们需要添加的是花叶效果:　将花叶的图片放到项目中,设置其Build Action为Resource模式,然后在代码中将花叶翻转180度居中放置,XAML代码如下:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12 | View Code    <Image Height="82" Margin="248,0,118,0" Source="Images/yezi2.png" Stretch="Fill" VerticalAlignment="Top"  RenderTransformOrigin="0.5,0.5">  <Image.RenderTransform>  <TransformGroup>  <ScaleTransform/>  <SkewTransform/>  <RotateTransform Angle="-180"/>  <TranslateTransform/>  </TransformGroup>  </Image.RenderTransform>  </Image> |

之后是添加飘动的白云,这个稍微复杂点,我们得先添加一个Canvas画板,以便能够附加其Left属性,然后利用StoryBoard来控制其左右移动效果:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10 | <Canvas x:Name="MyCanvas">  <Canvas.Triggers>  <EventTrigger RoutedEvent="FrameworkElement.Loaded">  <BeginStoryboard >  <Storyboard Storyboard.TargetName="img1" Storyboard.TargetProperty="(Canvas.Left)">  <DoubleAnimation From="0" To="514" Duration="0:0:20" RepeatBehavior="23:59:59" AutoReverse="True"/>  </Storyboard>  </BeginStoryboard>  </EventTrigger>  </Canvas.Triggers> |

其中需要注意的是, Duration代表动画时间, AutoReverse为True代表按照原路径返回.这样设置之后,就可以看见白云飘飘了.

最后的就是那个太阳和月亮的标记,当移动上面去,能够缓慢下降和升起的效果,这个利用Code来完成.

对应的XAML代码:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1 | <Image x:Name="DayMark" HorizontalAlignment="Left" Height="53" Source="Images/sun2.png" Stretch="Fill" VerticalAlignment="Top" Width="105" Margin="0,15,0,0" MouseEnter="DayMark\_MouseEnter" /> |

对应的后台事件:

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8 | private void DayMark\_MouseEnter(object sender, MouseEventArgs e)  {      TranslateTransform trans = new TranslateTransform();      DayMark.RenderTransform = trans;      DoubleAnimation animation = new DoubleAnimation(10,TimeSpan.FromSeconds(1));      animation.AutoReverse = true;      trans.BeginAnimation(TranslateTransform.YProperty,animation);  } |

好了,这样之后,我们的主要设计完成,接下来是调调位置,调调大小,添加鼠标拖拽事件,鼠标关闭事件等等.　希望对你有用哦.

# WPF换肤之七：异步

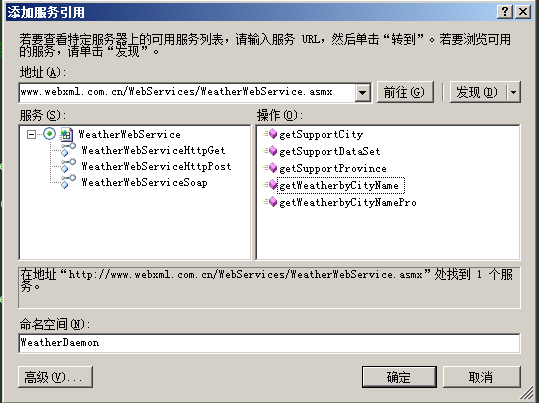
在WinForm时代，相信大家都遇到过这种情形，如果在程序设计过程中遇到了耗时的操作，不使用异步会导致程序假死。当然，在WPF中，这种情况也是存在的，所以我们就需要寻找一种解决方法来让程序界面响应和耗时操作异步进行，那么上述假死的情况就不会发生了。

这一节就着重讲解异步以及线程和界面交互。

异步使用方式（APM模式）

在上节中，我们给一个普通的Window窗口做了换肤处理，呈现出了一个非常酷的时区浏览小工具。当然，这一节，我们还是以那个工具为主，为其增加天气预报功能，而天气预报的数据来源，则通过WebService来获取。

首先，我们在程序中添加WebService服务引用，添加效果如下图所示，我们需要用到其中的GetWeatherByCityName方法来获取天气预报信息。



添加完成后，我们就可以通过下面的代码来获取城市的天气信息：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | static WeatherWebServiceSoapClient weatherClient;   //获取气象信息的WebService对象  private string[] GetWeather(string cityName)  {              string[] weatherInfoList = null;              if (weatherClient == null) weatherClient = new WeatherWebServiceSoapClient("WeatherWebServiceSoap"); //实例化服务调用           try              {                  weatherInfoList = weatherClient.getWeatherbyCityName(cityName);              }              catch (System.Net.WebException webException)              {                  throw webException;              }              catch (System.Net.Sockets.SocketException socketException)              {                  throw socketException;              }              catch (System.NullReferenceException nullException)              {                  throw nullException;              }              catch (System.Exception exception)              {                  throw exception;              }              finally              {                  if (weatherClient != null) weatherClient = null;              }              return weatherInfoList;  } |

返回的数组中包含的数据信息如下：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | View Code     #region content  //<string>直辖市</string>  //<string>上海</string>  //<string>58367</string>  //<string>58367.jpg</string>  //<string>2012-8-10 23:58:13</string>  //<string>27℃/33℃</string>  //<string>8月11日 阵雨转多云</string>  //<string>东南风4-5级</string>  //<string>3.gif</string>  //<string>1.gif</string>  //<string>今日天气实况：气温：28℃；风向/风力：北风 1级；湿度：80%；空气质量：良；紫外线强度：中等</string>  //<string> 穿衣指数：天气炎热，建议着短衫、短裙、短裤、薄型T恤衫、敞领短袖棉衫等清凉夏季服装。 感冒指数：暂无。 运动指数：有降水，风力较强，较适宜在户内开展低强度运动，若坚持户外运动，请选择避雨防风地点。 洗车指数：不宜洗车，未来24小时内有雨，如果在此期间洗车，雨水和路上的泥水可能会再次弄脏您的爱车。 晾晒指数：有降水，可能会淋湿晾晒的衣物，不太适宜晾晒。请随时注意天气变化。 旅游指数：有阵雨，气温较高，但风较大，能缓解湿热的感觉，还是适宜旅游，您仍可陶醉于大自然的美丽风光中。 路况指数：有降水，路面潮湿，车辆易打滑，请小心驾驶。 舒适度指数：天气较热，虽然有降水，但仍然无法削弱较高气温给人们带来的暑意，这种天气会让您感到不很舒适。 空气污染指数：气象条件有利于空气污染物稀释、扩散和清除，可在室外正常活动。 紫外线指数：属中等强度紫外线辐射天气，外出时建议涂擦SPF高于15、PA+的防晒护肤品，戴帽子、太阳镜。</string>  //<string>27℃/34℃</string>  //<string>8月12日 多云</string>  //<string>南风3-4级</string>  //<string>1.gif</string>  //<string>1.gif</string>  //<string>28℃/34℃</string>  //<string>8月13日 阵雨</string>  //<string>南风3-4级</string>  //<string>3.gif</string>  //<string>3.gif</string>  //<string> 上海简称：沪，位置：上海地处长江三角洲前缘，东濒东海，南临杭州湾，西接江苏，浙江两省，北界长江入海，正当我国南北岸线的中部，北纬31°14′，东 经121°29′。面积：总面积7823.5平方公里。人口：人口1000多万。上海丰富的人文资源、迷人的城市风貌、繁华的商业街市和欢乐的节庆活动形 成了独特的都市景观。游览上海，不仅能体验到大都市中西合壁、商儒交融、八方来风的氛围，而且能感受到这个城市人流熙攘、车水马龙、灯火璀璨的活力。上海 在中国现代史上占有着十分重要的地位，她是中国\*\*党的诞生地。许多震动中外的历史事件在这里发生，留下了众多的革命遗迹，处处为您讲述着一个个使人永不 忘怀的可歌可泣的故事，成为包含民俗的人文景观和纪念地。在上海，每到秋祭，纷至沓来的人们在这里祭祀先烈、缅怀革命历史,已成为了一种风俗。大上海在中 国近代历史中，曾是风起云涌可歌可泣的地方。在这里荟萃多少风云人物，散落在上海各处的不同住宅建筑，由于其主人的非同寻常，蕴含了耐人寻味的历史意义。 这里曾留下许多革命先烈的足迹。瞻仰孙中山、宋庆龄、鲁迅等故居，会使您产生抚今追昔的深沉遐思，这里还有无数个达官贵人的住宅，探访一下李鸿章、蒋介石 等人的公馆，可以联想起主人那段显赫的发迹史。</string>  #endregion |

现 在，问题来了，如果我们在程序中直接调用这个接口来获取天气信息的话，会发现主界面快则五六秒，慢则二十秒后才能够显现出来，这就说明，当程序获取天气信 息的时候，主界面被阻塞住了。为什么会被阻塞，是因为程序本身只有一条主线程，当程序获取天气信息的时候，线程占用，界面显示当然不能进行了。解决方法就 是使用异步。

关于异步的文章，请参看我之前的这篇博文：[我所知道的.NET异步](http://www.cnblogs.com/scy251147/archive/2012/03/03/2378477.html)， 由于我是APM模式(就是BeginXXXX和EndXXXX成对出现)的忠实粉丝，所以采用的代码如下：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | private void BeginInvokeWeather(string citiName)  {        try        {              Func<string, string[]> func = new Func<string, string[]>(GetWeather);              IAsyncResult iar = func.BeginInvoke(citiName, new AsyncCallback(EndInvokeWeather), func);              lblLoadingText.Dispatcher.Invoke(new Action(delegate()              {                      lblLoadingText.Opacity = 1;                      lblLoadingText.Content = "加载天气中...";               }));          }          catch(Exception ex)          {                throw ex;          }  }    private void EndInvokeWeather(IAsyncResult iar)  {          Func<string, string[]> func = (Func<string, string[]>)iar.AsyncState;  //还原状态        string[] weatherDaemonList = func.EndInvoke(iar);  //获取值        weatherInfoParamValue = weatherDaemonList;           if (weatherDaemonList != null)           {               if (weatherDaemonList.Length > 0)  //获取成功           {                    //进行处理              if (weatherDaemonList.Length < 9) return;                      string imgNameWithoutExtension = GetImgNameWithOutExtension(weatherDaemonList[8]);                    if (!imgNameWithoutExtension.Equals("NA")) isSuccess = true;                      string uriStringParam = "<pack://application:>,,,/TimeZoneDaemonApp;component/Images/Weather/" + imgNameWithoutExtension + ".png";                     //重新初始化一下，避免多次加载造成的资源冲突               weatherImg.Dispatcher.Invoke(new Action(delegate()                      {                          weatherImg = new BitmapImage();                      }));                      weatherImg.Dispatcher.Invoke(new Action(delegate()                      {                          weatherImg.BeginInit();                            weatherImg.UriSource = new Uri(uriStringParam);                          weatherImg.EndInit();                          DayMark.Width = weatherImgWidth;                          DayMark.Height = weatherImgHeight;                          DayMark.Source = weatherImg;                          lblLoadingText.Content = "调用结束...";                          lblLoadingText.Opacity = 0;                      }));                  }              }          } |

这样，当程序启动的时候，便会异步获取天气信息，界面阻塞的问题得以解决，请看图示：

[](http://img.ddvip.com/2012/0828/201208281225195500.jpg)

加载完成之后，我们就可以看到原来现在我在的地方是朗朗晴天呢... :D

[](http://img.ddvip.com/2012/0828/201208281225198415.jpg)

当 然，这里还涉及到一个问题，就是线程和UI交互的问题，在Winform中我们可以通过Control.Invoke的方式来进行，在WPF中，只是多了 一个Dispatcher而已，具体用法就是Control. Dispatcher.Invoke来进行，比如加载天气的Label就是利用这种方式进行交互的：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5 | lblLoadingText.Dispatcher.Invoke(new Action(delegate()  {         lblLoadingText.Opacity = 1;         lblLoadingText.Content = "加载天气中...";  })); |

希望本文对你有用。

源码下载

[点击这里下载源码](http://files.cnblogs.com/scy251147/TimeZoneDaemonApp%28WithAsyncMode%29.rar)   由于工程中图片体积太大，就拿出来单独上传，用的时候直接覆盖掉Images文件夹即可。 [点击这里下载资源文件](http://files.cnblogs.com/scy251147/Images.rar)

# WPF换肤之八：创建3D浏览效果

上节中，我们展示了WPF中的异步以及界面线程交互的方式，使得应用程序的显示更加的流畅。这节我们主要讲解如何设计一个具有3D浏览效果的天气信息浏览器。

效果显示

下面我们看截图：



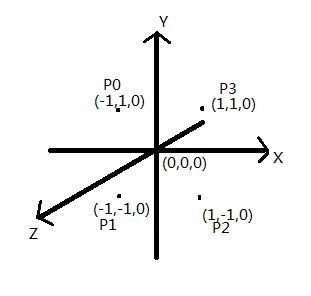


是不是能够感受到一种与众不同的感觉。如果你能够感受到它的与众不同，这也是我本节所要达到的目标。

实现方式

上 面的只是一个简单的3D图形，它的产生需要依赖于WPF中的MeshGeometry3D对象，这个对象按照微软官方的解释就是用于生成3D形状的三角形 基元，它有三个比较重要的属性：Positions（这个是必须的），TextureCoordinates以及TriangleIndices。其中 Positions是指定当前的界面坐标，也就是由这些坐标形成什么样的界面形状；TextureCoordinates，官方说法是材质被映射到构成网 格的顶点，通俗的说来，就是改变图像的显示顺序的，比如是正向显示，横向显示等等，也就是能够使界面倒个个儿；TriangleIndices则代表当前 显示图形的正反面。

我们先看一个坐标图：



在这个坐标图中，P0就是(-1,1,0)坐标，P1就是(-1,-1,0)坐标，P2就是(1,-1,0)坐标，P3就是(1,1,0)坐标，这四个象限按照逆时针方向来。那么在接下来的讲解中，这些坐标将会有辅助作用。

先对应这坐标说说TextureCoordinates， 在XAML代码中，可以看到其设置如下：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | <!-- front side-->  <Viewport2DVisual3D Material="{StaticResource  CubeSideMaterial }">      <Viewport2DVisual3D.Geometry>          <MeshGeometry3D Positions="0,1,0 0,0,0 1,0,0 1,1,0"                          TextureCoordinates="0,0 0,1 1,1 1,0"                          TriangleIndices="0 1 2  0 2 3"/>      </Viewport2DVisual3D.Geometry>      ....  </Viewport2DVisual3D>    <!--  left side -->  <Viewport2DVisual3D Material="{StaticResource CubeSideMaterial}">      <Viewport2DVisual3D.Geometry>          <MeshGeometry3D Positions="1,1,0 1,0,0 1,0,-1 1,1,-1"                          TextureCoordinates="0,0 0,1 1,1 1,0"                          TriangleIndices="0 1 2  0 2 3"/>      </Viewport2DVisual3D.Geometry>      ....  </Viewport2DVisual3D>    <!--Back side-->  <Viewport2DVisual3D Material="{StaticResource CubeSideMaterial}">      <Viewport2DVisual3D.Geometry>        <MeshGeometry3D Positions="1,1,-1 1,0,-1 0,0,-1 0,1,-1"                          TextureCoordinates="0,0 0,1 1,1 1,0"                          TriangleIndices="0 1 2  0 2 3"/>      </Viewport2DVisual3D.Geometry>          ....  </Viewport2DVisual3D>    <!--Right side-->  <Viewport2DVisual3D Material="{StaticResource CubeSideMaterial}">      <Viewport2DVisual3D.Geometry>          <MeshGeometry3D Positions="0,1,-1 0,0,-1 0,0,0 0,1,0"                          TextureCoordinates="0,0 0,1 1,1 1,0"                          TriangleIndices="0 1 2  0 2 3"/>  </Viewport2DVisual3D.Geometry>                          ....  </Viewport2DVisual3D> |

可 以看到，在这是个面中，它的值都是0,0 0,1 1,1 1,0, 这组值代表界面正向显示（默认情况下），其实,按照映射方式来的话（按照上图的P0，P1，P2，P3）,也就是0,1被映射到了P0;0,0被映射到了 P1;1,0被映射到了P2;1,1被映射到了P3。假如稍微改变下，为0,1 1,1 1,0 0,0 那么我们就可以看到界面显示如下：



图像侧着显示了。

再 来说说TriangleIndices,这个主要用来显示图形的正反面的，如果按照逆时针，即P0，P1，P2，P3的方向来的话，则可以显示正常的图 形，也就是假如它的值为(0 1 2 0 2 3 ) 或者( 0 1 2  2 3 0 ) 或者 (1 2 3  3 0 1 )均可以显示出正面的图形来，需要注意的是，由于计算机的图形都是用三角形表示的，所以这里0 1 2为一组代表一个三角形，2 3 0 为一组，代表另一个三角形，这两个三角形就组成了一个矩形。

如果我们让它显示一半正面，一半反面，则可以使用(0 1 2  0 3 2 )来表示(一个为逆时针，一个为顺时针)，得到的结果如下：



由于其背面透明且无任何界面元素，所以看不到。

最 后来说说Positions，它用来表示显示的图形，由于我们这里是一个正方体，所以每个面应该有4个点，4个点按照逆时针方向(P0 - > P1 - > P2 -> P3)方向排列即可。 如果一个正方体，它的正面的四个点按照逆序肯定是(0,1,0 0,0,0 1,0,0 1,1,0),依此类推，那么它的左边侧面的点肯定是(1,1,0 1,0,0 1,0,-1 1,1,-1)。 按照这样的点推下去，就能够得到正确的Positions的值。需要说明的是，复杂的图形，Positions的值是非常复杂的，那是推理不来的。

下面说说如何切换四个面：

首先，3D图形中，我们都需要有视点，这里称作(Camera)，然后还得需要有光源(Light)，这样图像才不至于黑乎乎一篇，在WPF中也是如此：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <Viewport3D x:Name="view" ClipToBounds="True" RenderOptions.EdgeMode="Aliased">              <!--Camera-->              <Viewport3D.Camera>                  <PerspectiveCamera x:Name="camera" FieldOfView="59" Position="0.5,0.5,2" LookDirection="0,0,-1">                      <PerspectiveCamera.Transform>                          <RotateTransform3D x:Name="rot" CenterY="0.5" CenterX="0.5" CenterZ="-0.5">                              <RotateTransform3D.Rotation>                                  <!-- rotation -->                                  <AxisAngleRotation3D x:Name="camRotation" Axis="0,1,0" Angle="0"/>                              </RotateTransform3D.Rotation>                          </RotateTransform3D>                      </PerspectiveCamera.Transform>                  </PerspectiveCamera>              </Viewport3D.Camera>              <!--Light-->              <ModelVisual3D>                  <ModelVisual3D.Content>                      <AmbientLight Color="White" />                  </ModelVisual3D.Content>              </ModelVisual3D> |

其中FieldOfView用于切换视角的远近，而Position用于控制物体的方位，LookDirection用控制视角查看的方向。

接下来，我们就可以通过上面的讲解设计出具体的界面代码了：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146 | <!-- front side-->              <Viewport2DVisual3D Material="{StaticResource  CubeSideMaterial }">                  <Viewport2DVisual3D.Geometry>                      <MeshGeometry3D Positions="0,1,0 0,0,0 1,0,0 1,1,0"                                      TextureCoordinates="0,0 0,1 1,1 1,0"                                      TriangleIndices="0 1 2  0 2 3"/>                  </Viewport2DVisual3D.Geometry>                  <Border BorderThickness="1" x:Name="FrontSide" BorderBrush="White" CornerRadius="4"  PreviewMouseDown="FrontSide\_PreviewMouseDown" >                      <Border.Background>                          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                              <GradientStop Color="Black"  />                          </LinearGradientBrush>                      </Border.Background>                      <StackPanel  Height="450" Width="450" OpacityMask="White" >                          <Button  PreviewMouseLeftButtonDown="Button\_MouseLeftButtonDown" Style="{StaticResource CloseRadialButton}" HorizontalAlignment="Right" Margin="0,2,2,0"></Button>                          <Border Margin="15,0,15,15" BorderThickness="1" CornerRadius="8" Height="30" VerticalAlignment="Top" PreviewMouseDown="Border\_PreviewMouseDown">                              <Border.Background>                                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                                      <GradientStop Color="#FF333333" Offset=".1"/>                                      <GradientStop Color="Red" Offset="1"/>                                  </LinearGradientBrush>                              </Border.Background>                              <TextBlock Foreground="White" HorizontalAlignment="Center" FontSize="16" Margin="3" Text="天气信息"/>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White"  >                              <TextBlock Foreground="White"  Margin="0,3,0,0" >                                  直辖市:上海                              </TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White"  >                              <TextBlock Foreground="White"  Margin="0,3,0,0">  -8-10 23:58:13 最低气温:27℃/最高气温:33℃                              </TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White" >                              <TextBlock Foreground="White"  Margin="0,3,0,0" TextWrapping="Wrap" TextAlignment="Left">                                  今日天气实况：气温：28℃；风向/风力：北风 1级；湿度：80%；空气质量：良；紫外线强度：中等 穿衣指数：天气炎热，建议着短衫、短裙、短裤、薄型T恤衫、敞领短袖棉衫等清凉夏季服装。                              </TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White" >                              <TextBlock Foreground="White"  Margin="0,3,0,0" TextWrapping="Wrap"  TextAlignment="Left">                                  感 冒指数：暂无。 运动指数：有降水，风力较强，较适宜在户内开展低强度运动，若坚持户外运动，请选择避雨防风地点。 洗车指数：不宜洗车，未来24小时内有雨，如果在此期间洗车，雨水和路上的泥水可能会再次弄脏您的爱车。 晾晒指数：有降水，可能会淋湿晾晒的衣物，不太适宜晾晒。请随时注意天气变化。 旅游指数：有阵雨，气温较高，但风较大，能缓解湿热的感觉，还是适宜旅游，您仍可陶醉于大自然的美丽风光中。 路况指数：有降水，路面潮湿，车辆易打滑，请小心驾驶。 舒适度指数：天气较热，虽然有降水，但仍然无法削弱较高气温给人们带来的暑意，这种天气会让您感到不很舒适。                              </TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White" >                              <TextBlock Foreground="White"  Margin="0,3,0,0" TextWrapping="Wrap" TextAlignment="Left">                                  空气污染指数：气象条件有利于空气污染物稀释、扩散和清除，可在室外正常活动。 紫外线指数：属中等强度紫外线辐射天气，外出时建议涂擦SPF高于15、PA+的防晒护肤品，戴帽子、太阳镜。                              </TextBlock>                          </Border>                      </StackPanel>                  </Border>              </Viewport2DVisual3D>                <!--  left side -->              <Viewport2DVisual3D Material="{StaticResource CubeSideMaterial}">                  <Viewport2DVisual3D.Geometry>                      <MeshGeometry3D Positions="1,1,0 1,0,0 1,0,-1 1,1,-1"                                      TextureCoordinates="0,0 0,1 1,1 1,0"                                      TriangleIndices="0 1 2  0 2 3"/>                  </Viewport2DVisual3D.Geometry>                  <Border BorderThickness="1" x:Name="LeftSide" BorderBrush="White" CornerRadius="4" PreviewMouseDown="LeftSide\_PreviewMouseDown"  >                      <Border.Background>                          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                              <GradientStop Color="Black" />                          </LinearGradientBrush>                      </Border.Background>                      <StackPanel  Height="450" Width="450" OpacityMask="White">                          <Button Style="{StaticResource CloseRadialButton}" HorizontalAlignment="Right" Margin="0,2,2,0" PreviewMouseLeftButtonDown="Button\_MouseLeftButtonDown"></Button>                          <Border Margin="15,0,15,15" BorderThickness="1" CornerRadius="8" Height="30" VerticalAlignment="Top">                              <Border.Background>                                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                                      <GradientStop Color="#FF333333" Offset=".1"/>                                      <GradientStop Color="Red" Offset="1"/>                                  </LinearGradientBrush>                              </Border.Background>                              <TextBlock Foreground="White" HorizontalAlignment="Center" FontSize="16" Margin="3">未来天气信息</TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White"  >                              <TextBlock Foreground="White"  Margin="0,3,0,0" >  月11日 阵雨转多云 东南风4-5级 27℃/34℃                              </TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White"  >                              <TextBlock Foreground="White"  Margin="0,3,0,0">  月12日 多云 南风3-4级 28℃/34℃                              </TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White" >                              <TextBlock Foreground="White"  Margin="0,3,0,0" TextWrapping="Wrap" TextAlignment="Left">  月13日 阵雨 南风3-4级                              </TextBlock>                          </Border>                      </StackPanel>                  </Border>              </Viewport2DVisual3D>                <!--Back side-->              <Viewport2DVisual3D Material="{StaticResource CubeSideMaterial}">                  <Viewport2DVisual3D.Geometry>                      <MeshGeometry3D Positions="1,1,-1 1,0,-1 0,0,-1 0,1,-1"                                      TextureCoordinates="0,0 0,1 1,1 1,0"                                      TriangleIndices="0 1 2  0 2 3"/>                  </Viewport2DVisual3D.Geometry>                  <Border BorderThickness="1" x:Name="BackSide" BorderBrush="White" CornerRadius="4" PreviewMouseDown="BackSide\_PreviewMouseDown" >                      <Border.Background>                          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                              <GradientStop Color="Black" />                          </LinearGradientBrush>                      </Border.Background>                      <StackPanel  Height="450" Width="450" OpacityMask="White">                         <Button  PreviewMouseLeftButtonDown="Button\_MouseLeftButtonDown" Style="{StaticResource CloseRadialButton}" HorizontalAlignment="Right" Margin="0,2,2,0"></Button>                          <Border Margin="15,0,15,15" BorderThickness="1" CornerRadius="8" Height="30" VerticalAlignment="Top">                              <Border.Background>                                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                                      <GradientStop Color="#FF333333" Offset=".1"/>                                      <GradientStop Color="Red" Offset="1"/>                                  </LinearGradientBrush>                              </Border.Background>                              <TextBlock Foreground="White" HorizontalAlignment="Center" FontSize="16" Margin="3">城市简介</TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White"  >                              <TextBlock Foreground="White"  Margin="0,3,0,0" TextWrapping="Wrap" TextAlignment="Left" >                                  上 海简称：沪，位置：上海地处长江三角洲前缘，东濒东海，南临杭州湾，西接江苏，浙江两省，北界长江入海，正当我国南北岸线的中部，北纬31°14′，东经 121°29′。面积：总面积7823.5平方公里。人口：人口1000多万。上海丰富的人文资源、迷人的城市风貌、繁华的商业街市和欢乐的节庆活动形成 了独特的都市景观。游览上海，不仅能体验到大都市中西合壁、商儒交融、八方来风的氛围，而且能感受到这个城市人流熙攘、车水马龙、灯火璀璨的活力。上海在 中国现代史上占有着十分重要的地位，她是中国\*\*党的诞生地。许多震动中外的历史事件在这里发生，留下了众多的革命遗迹，处处为您讲述着一个个使人永不忘 怀的可歌可泣的故事，成为包含民俗的人文景观和纪念地。在上海，每到秋祭，纷至沓来的人们在这里祭祀先烈、缅怀革命历史,已成为了一种风俗。大上海在中国 近代历史中，曾是风起云涌可歌可泣的地方。在这里荟萃多少风云人物，散落在上海各处的不同住宅建筑，由于其主人的非同寻常，蕴含了耐人寻味的历史意义。这 里曾留下许多革命先烈的足迹。瞻仰孙中山、宋庆龄、鲁迅等故居，会使您产生抚今追昔的深沉遐思，这里还有无数个达官贵人的住宅，探访一下李鸿章、蒋介石等 人的公馆，可以联想起主人那段显赫的发迹史。                              </TextBlock>                          </Border>                      </StackPanel>                  </Border>              </Viewport2DVisual3D>                <!--Right side-->              <Viewport2DVisual3D Material="{StaticResource CubeSideMaterial}">                  <Viewport2DVisual3D.Geometry>                      <MeshGeometry3D Positions="0,1,-1 0,0,-1 0,0,0 0,1,0"                                      TextureCoordinates="0,0 0,1 1,1 1,0"                                      TriangleIndices="0 1 2  0 2 3"/>                  </Viewport2DVisual3D.Geometry>                  <Border BorderThickness="1" x:Name="RightSide" BorderBrush="White" CornerRadius="4" PreviewMouseDown="RightSide\_PreviewMouseDown" >                      <Border.Background>                          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                              <GradientStop Color="Black" />                          </LinearGradientBrush>                      </Border.Background>                      <StackPanel  Height="450" Width="450" OpacityMask="White">                          <Button  PreviewMouseLeftButtonDown="Button\_MouseLeftButtonDown" Style="{StaticResource CloseRadialButton}" HorizontalAlignment="Right" Margin="0,2,2,0"></Button>                          <Border Margin="15,0,15,15" BorderThickness="1" CornerRadius="8" Height="30" VerticalAlignment="Top">                              <Border.Background>                                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">                                      <GradientStop Color="#FF333333" Offset=".1"/>                                      <GradientStop Color="Red" Offset="1"/>                                  </LinearGradientBrush>                              </Border.Background>                              <TextBlock Foreground="White" HorizontalAlignment="Center" FontSize="16" Margin="3">图表显示</TextBlock>                          </Border>                          <Border Margin="15,0,15,0" HorizontalAlignment="Left" BorderThickness="0,0,0,1" BorderBrush="White"  >                              <TextBlock Foreground="White"  Margin="0,3,0,0" TextWrapping="Wrap" TextAlignment="Left" >                                  这个地方是图表显示温度                              </TextBlock>                          </Border>                      </StackPanel>                  </Border>              </Viewport2DVisual3D>              </Viewport3D> |

那么在后台如何做到点击切换呢？

在后台我们是通过一个DispatcherTimer来控制切换的动态显示，并且通过设置RotateTransform.的Angle来控制四个 面的逐一显现的。当Angle为0时，显现的是第一面；为90时，显现的是左侧面；为180时，显现的是背面；为270时，显现的是右侧面；为360时， 回到原位，这就相当于摄像机的位置和视角改变一样。

具体代码如下：

|  |  |
| --- | --- |
| 双击代码全选 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | public \_3DWeatherWindow(string[] WeatherList)         {             InitializeComponent();             weatherList = WeatherList;               if (clock == null) clock = new DispatcherTimer();             clock.Tick += new EventHandler(clock\_Tick);             clock.Interval = new TimeSpan(0, 0, 0, 0, 10);         }           private string[] weatherList;           DispatcherTimer clock = null;         double rotAngle = 90;           private void clock\_Tick(object sender, EventArgs e)         {             camRotation.Angle += 5;             if (camRotation.Angle >= rotAngle) clock.Stop();         }           private void FrontSide\_PreviewMouseDown(object sender, MouseButtonEventArgs e)         {             //初始化值             camRotation.Angle = 0;             rotAngle = 90;               clock.Start();         }           private void LeftSide\_PreviewMouseDown(object sender, MouseButtonEventArgs e)         {             rotAngle = 180;             clock.Start();         }           private void BackSide\_PreviewMouseDown(object sender, MouseButtonEventArgs e)         {             rotAngle = 270  ;             clock.Start();         }             private void RightSide\_PreviewMouseDown(object sender, MouseButtonEventArgs e)         {             rotAngle = 360;             clock.Start();         } |

好了，希望对你有用。