

一步一步学 Silverlight 2 系列 (1): 创建一个基本的 Silverlight 应用 .....	2
一步一步学 Silverlight 2 系列 (2): 基本控件 .....	9
一步一步学 Silverlight 2 系列 (3): 界面布局 .....	16
一步一步学 Silverlight 2 系列 (4): 鼠标事件处理 .....	26
一步一步学 Silverlight 2 系列 (5): 实现简单的拖放功能 .....	35
一步一步学 Silverlight 2 系列 (6): 键盘事件处理 .....	40
一步一步学 Silverlight 2 系列 (7): 全屏模式支持 .....	45
一步一步学 Silverlight 2 系列 (8): 使用样式封装控件观感 .....	50
一步一步学 Silverlight 2 系列 (9): 使用控件模板 .....	55
一步一步学 Silverlight 2 系列 (10): 使用用户控件 .....	63
一步一步学 Silverlight 2 系列 (11): 数据绑定 .....	70
一步一步学 Silverlight 2 系列 (12): 数据与通信之 WebClient .....	81
一步一步学 Silverlight 2 系列 (13): 数据与通信之 WebRequest .....	90
一步一步学 Silverlight 2 系列 (14): 数据与通信之 WCF .....	97
一步一步学 Silverlight 2 系列 (15): 数据与通信之 ASMX .....	107
一步一步学 Silverlight 2 系列 (16): 数据与通信之 JSON .....	117
一步一步学 Silverlight 2 系列 (17): 数据与通信之 ADO.NET Data Services .....	126
一步一步学 Silverlight 2 系列 (18): 综合实例之 RSS 阅读器 .....	137
一步一步学 Silverlight 2 系列 (19): 如何在 Silverlight 中与 HTML DOM 交互 (上) .....	147
一步一步学 Silverlight 2 系列 (20): 如何在 Silverlight 中与 HTML DOM 交互 (下) .....	156
一步一步学 Silverlight 2 系列 (21): 如何在 Silverlight 中调用 JavaScript .....	168

## 一步一步学 Silverlight 2 系列（1）：创建一个基本的 Silverlight 应用

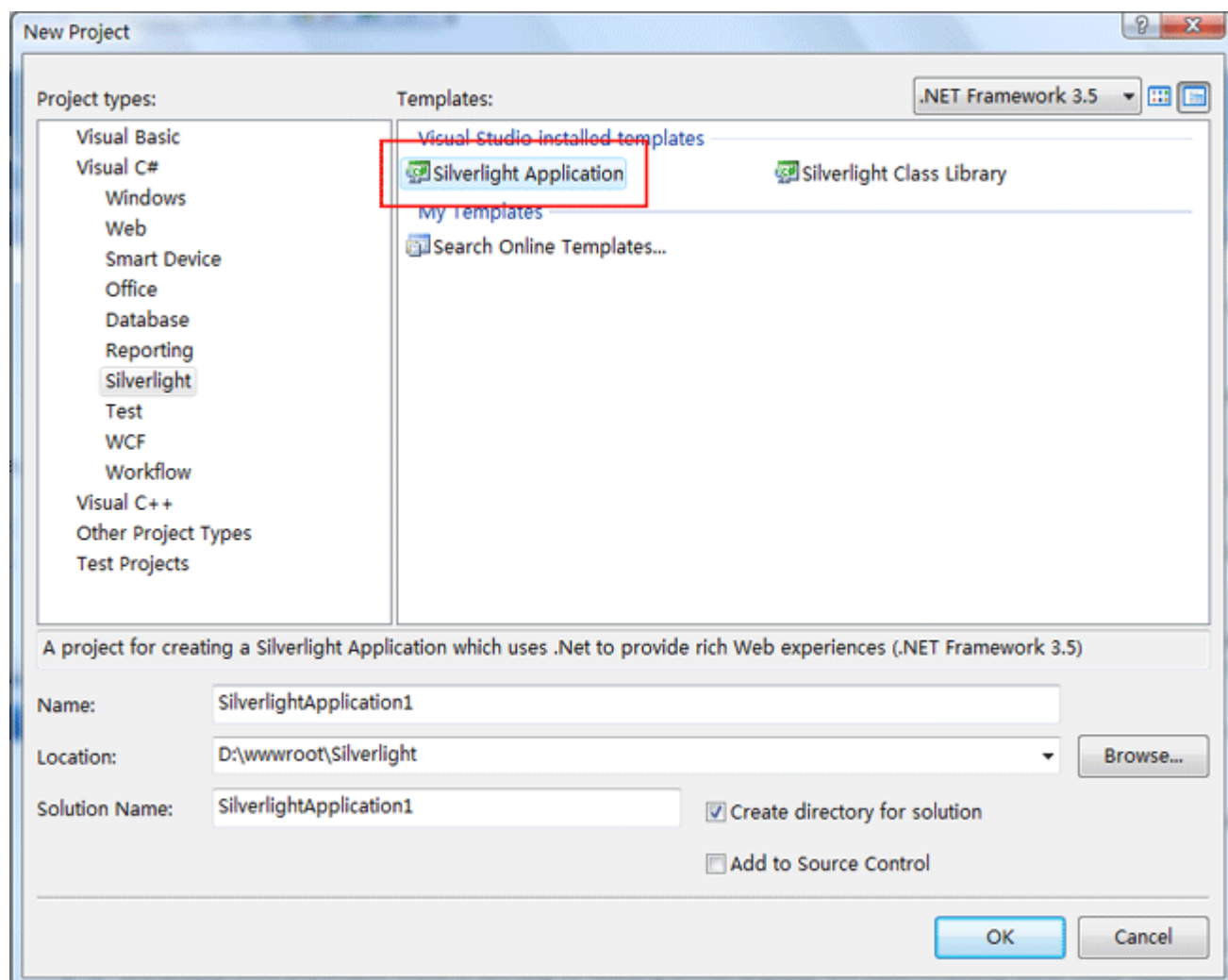
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

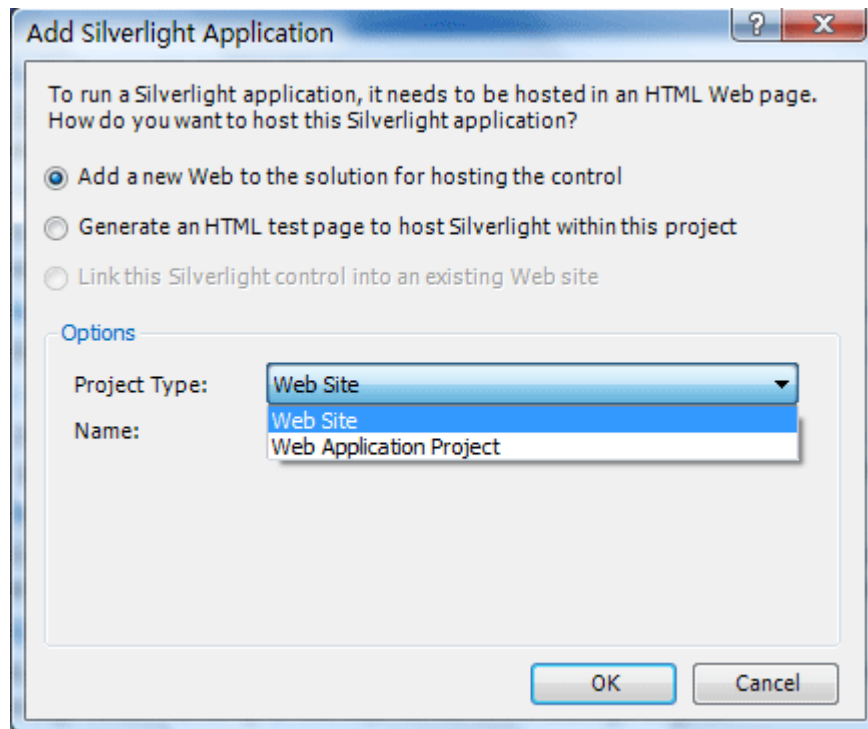
本文为系列第一篇创建一个基本的 Silverlight 2 应用，不能免俗，从最简单的 Hello Word 开始。

### 建立项目

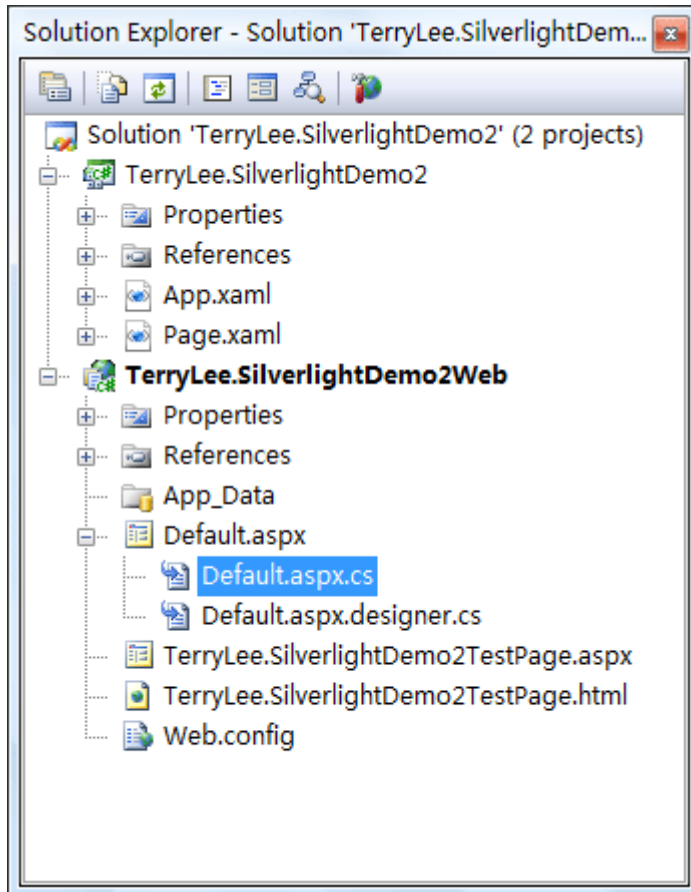
安装完 Silverlight 2 Beta 1 之后打开 VS2008，打开新建项目对话框，可以看到 Silverlight Application 项目模板。



Silverlight 应用不能够独立运行，之后弹出的对话框中可供我们选择创建一个 ASP.NET Web Site 或者 Web Application Project 用来托管 Silverlight 应用程序。

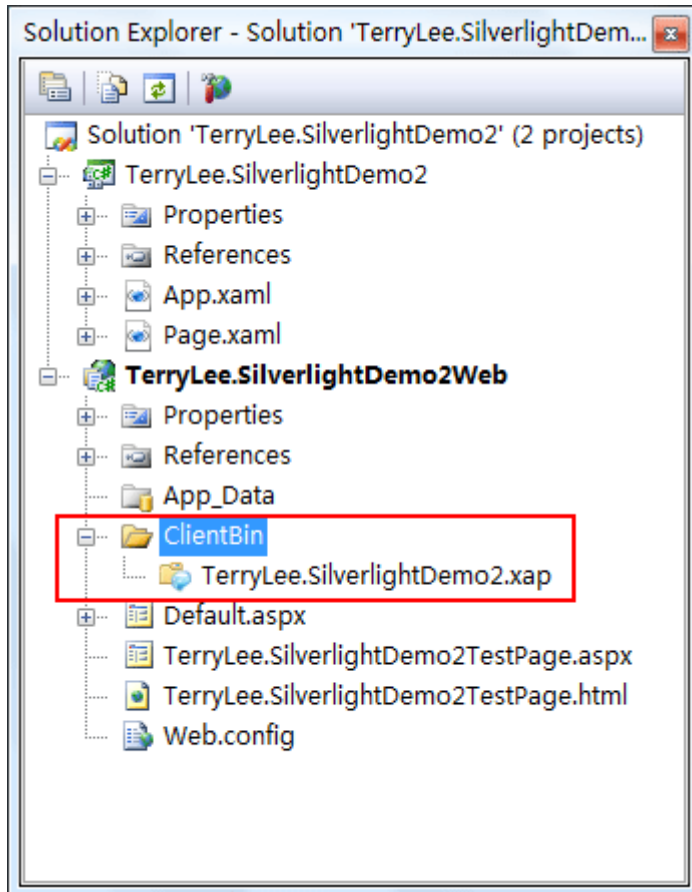


这里我们选择创建一个 Web Application Project，创建完成后的项目结构如下所示：



## 理解.xap 文件

在建立一个 Silverlight 应用程序后，我们什么都不做，直接编译一下整个解决方案，可以看到在资源管理器中多出了一个 ClientBin 的文件夹，并在下面添加了一个 TerryLee.SilverlightDemo2.xap 的文件。



该文件是一个标准的 .NET 程序集, 在编译的时候所有的 XAML 标识和资源文件如图片等都会包含在里面, 采用了标准的 Zip 压缩算法, 以减少客户端下载的文件体积。拷贝一份该文件, 并且修改后缀名 .xap 为 .zip, 并且解压缩, 可以看到里面包含了一些 dll 文件和一个 AppManifest.xaml:

名称	拍摄日期	标记	大小	分级
AppManifest.xaml			1 KB	☆☆☆☆☆
System.Windows.Controls.dll			264 KB	☆☆☆☆☆
System.Windows.Controls.Extended.dll			192 KB	☆☆☆☆☆
TerryLee.SilverlightDemo2.dll			6 KB	☆☆☆☆☆

再打开 TerryLee.SilverlightDemo2TestPage.aspx 文件, 在页面的顶部引入了 System.Web.Silverlight 程序集, 支持 <asp:Silverlight/> 控件:

```
<%@ Register Assembly="System.Web.Silverlight" Namespace="System.Web.UI.Silverlight
Controls"

TagPrefix="asp" %>
```

<asp:Silverlight/>控件的声明如下，其中属性 Source 属性指定了刚才编译生成的.xap 文件的路径：

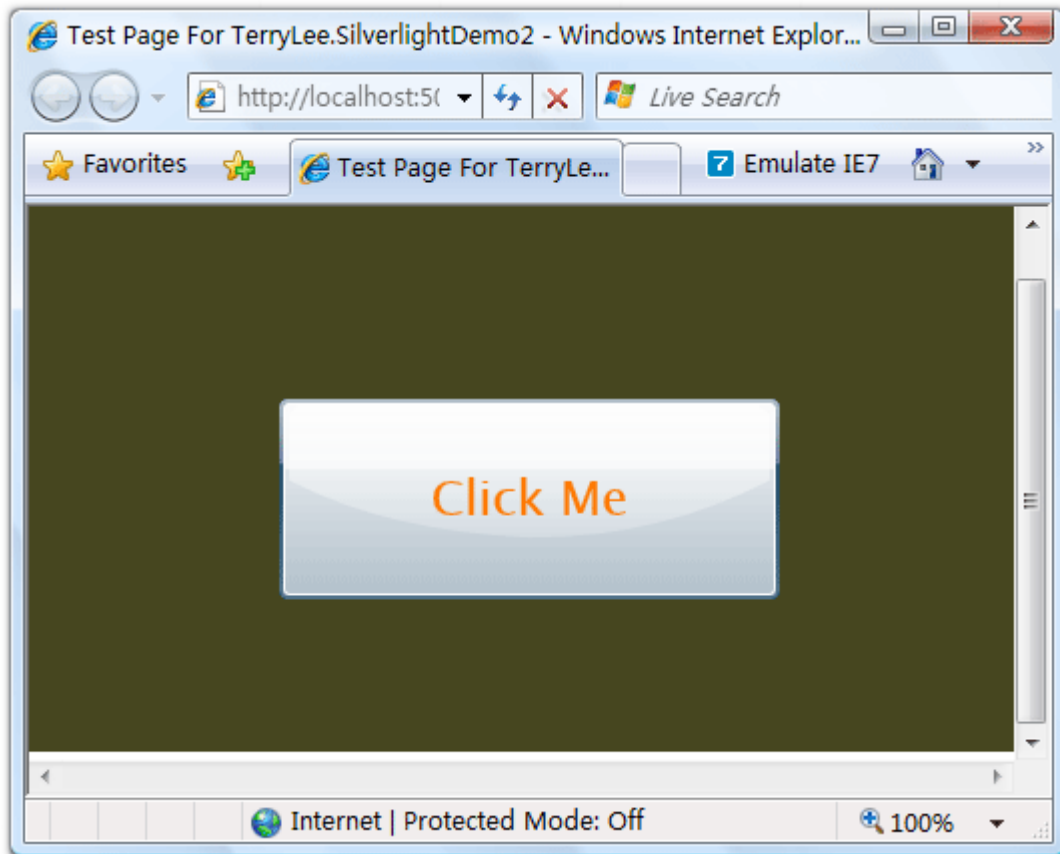
```
<asp:Silverlight ID="Xaml1" runat="server"
    Source="~/ClientBin/TerryLee.SilverlightDemo2.xap"
    Version="2.0" Width="100%" Height="100%" />
```

## 创建一个 Hello Word 程序

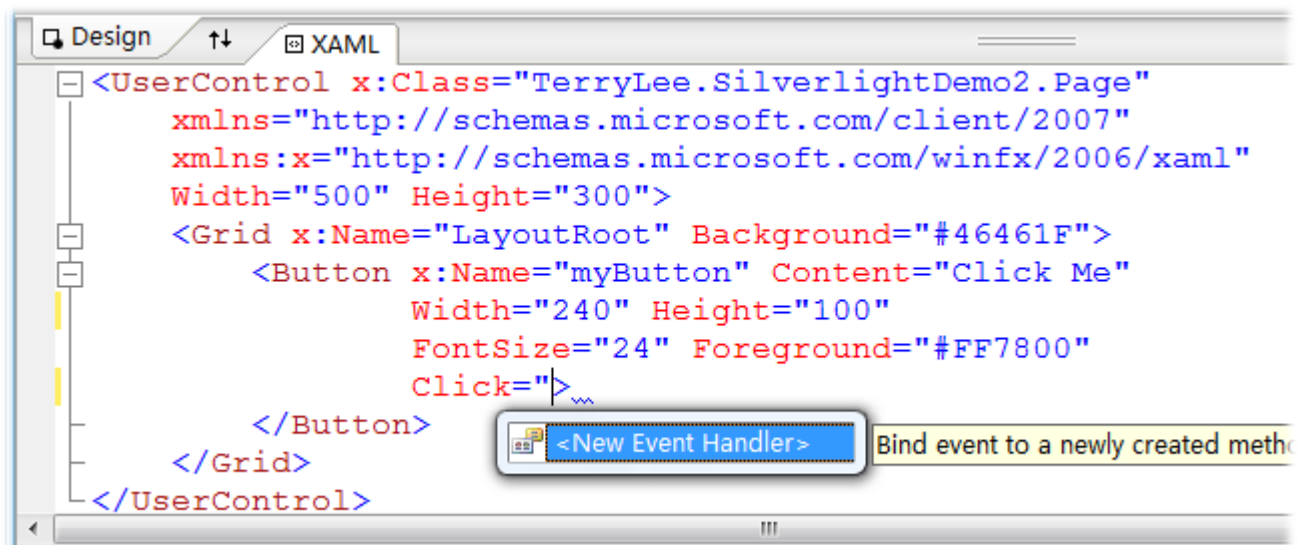
现在我们创建一个简单的 Hello Word Silverlight 程序，使用如下 XAML 创建一个简单的按钮：



运行后效果如下：



为按钮添加 Click 事件，在 XAML 编辑器中输入事件名称 Click 之后，再按 Tab 键将会使用默认的命名方法生成事件处理方法：

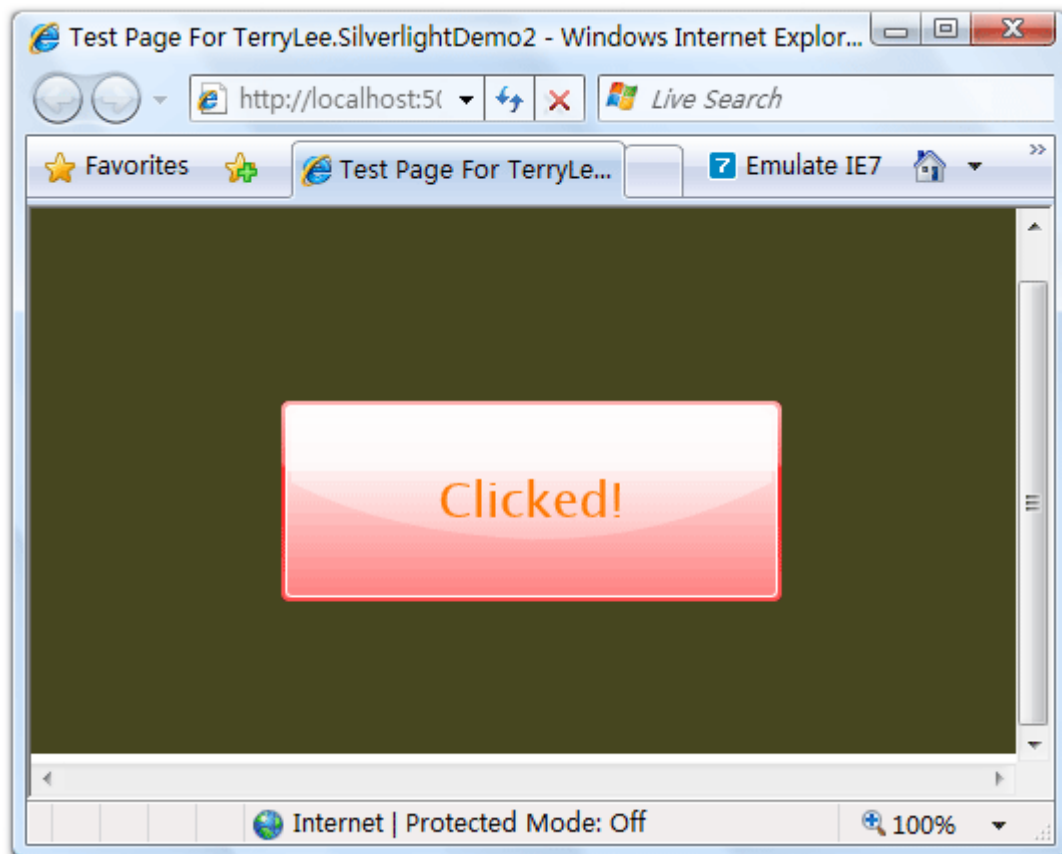


打开 Page.xaml.cs 文件后，可以看到已经生成了对应的事件处理方法，现在就可以用熟悉的 C# 来编写处理程序了，如单击按钮时我们改变按钮的背景色和文字：

```
private void myButton_Click(object sender, RoutedEventArgs e)
```

```
{  
  
    this.myButton.Content = "Clicked!";  
  
    this.myButton.Background = new SolidColorBrush(Colors.Red);  
  
}
```

再运行上面的程序并单击按钮，按钮的文字及背景色发生了变化：



## 结束语

本篇文章是使用 Visual Studio 2008 开发 Silverlight 2 应用程序的一个入门，相信大家已经看过 ScottGu 的文章已经有所了解。但是为了整个系列完整起见，还是做了一下重复的劳动。



## 一步一步学 Silverlight 2 系列（2）：基本控件

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

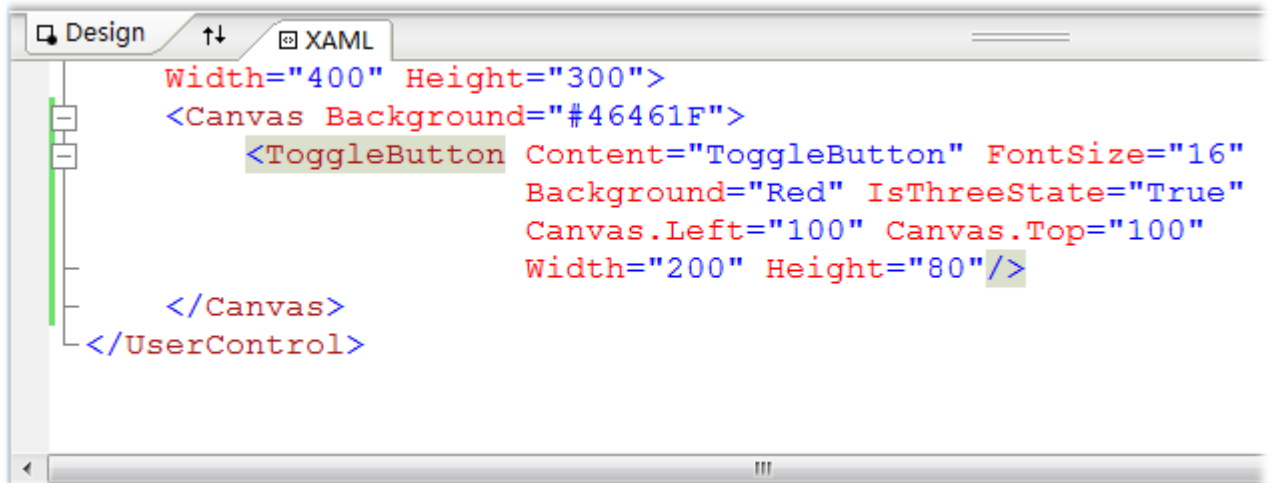
本文为系列文章第二篇学习几个基本的控件。

在 Silverlight 2 中，提供了大量的控件，包括 Button、Calendar、CheckBox、DataGrid、DatePicker、GridSplitter、HyperlinkButton、ListBox、RadioButton、ScrollViewer、Slider、ToggleButton、ToolTip、WatermarkedTextBox 等，本文将讲述其中的几个控件之用法。

### 控件之 **ToggleButton**

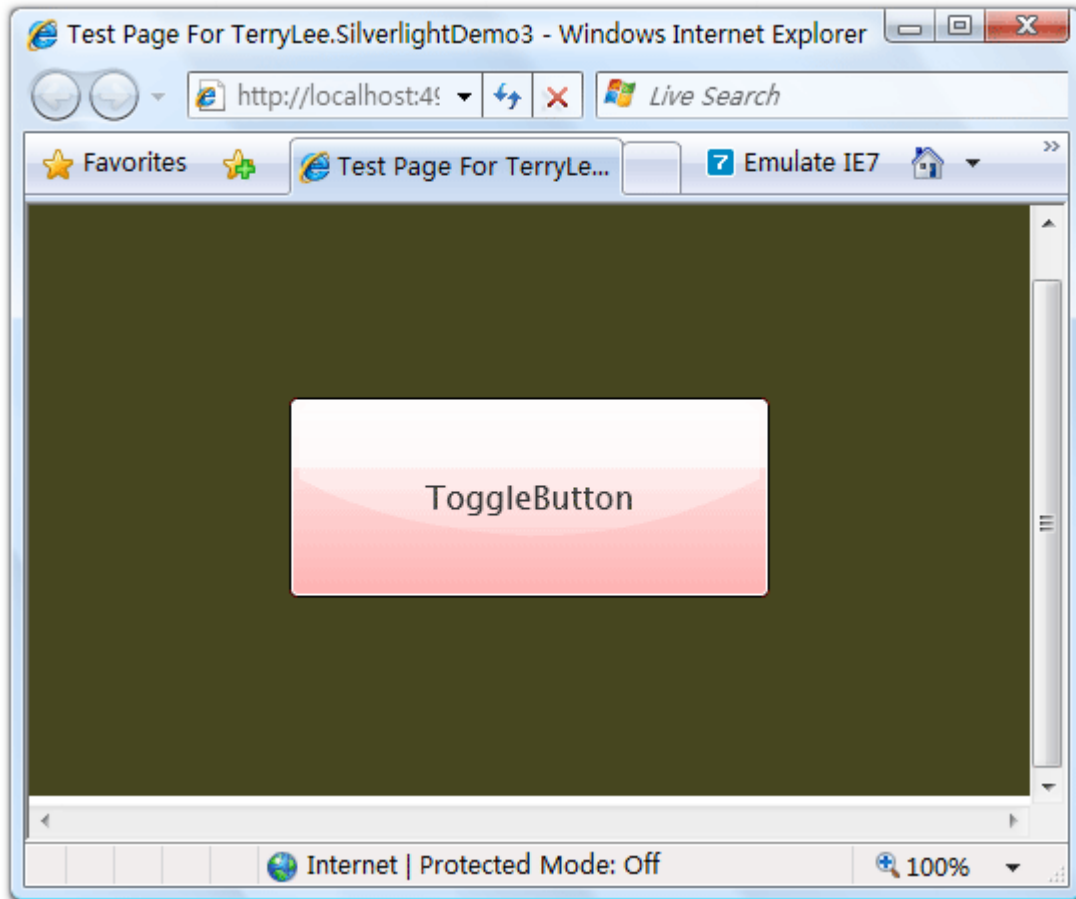
翻转效果在 AJAX 时代已经相当多了，Silverlight 中内置了 ToggleButton 控件，可以使用如下 XAML

代码声明一个 ToggleButton:

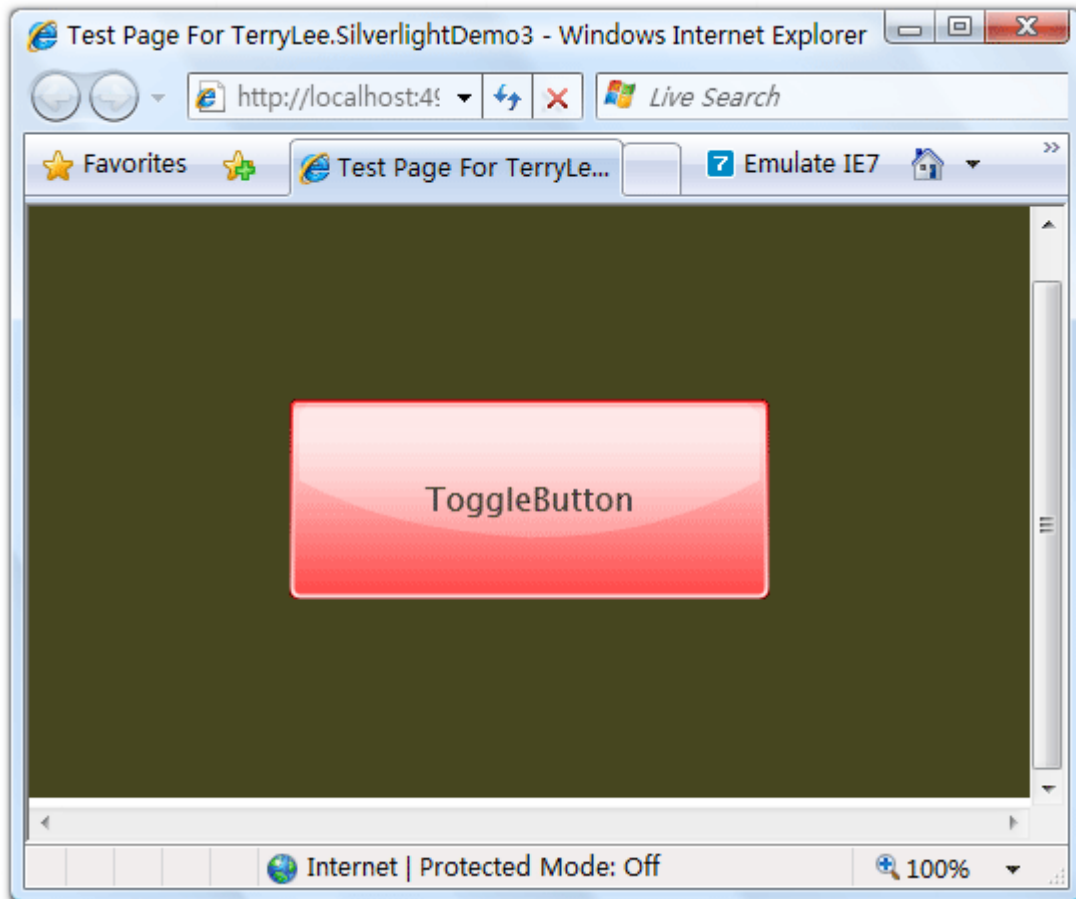


```
Width="400" Height="300">
  <Canvas Background="#46461F">
    <ToggleButton Content="ToggleButton" FontSize="16"
      Background="Red" IsThreeState="True"
      Canvas.Left="100" Canvas.Top="100"
      Width="200" Height="80"/>
  </Canvas>
</UserControl>
```

运行后界面如下:



单击按钮后，控件效果外观效果将会改变：



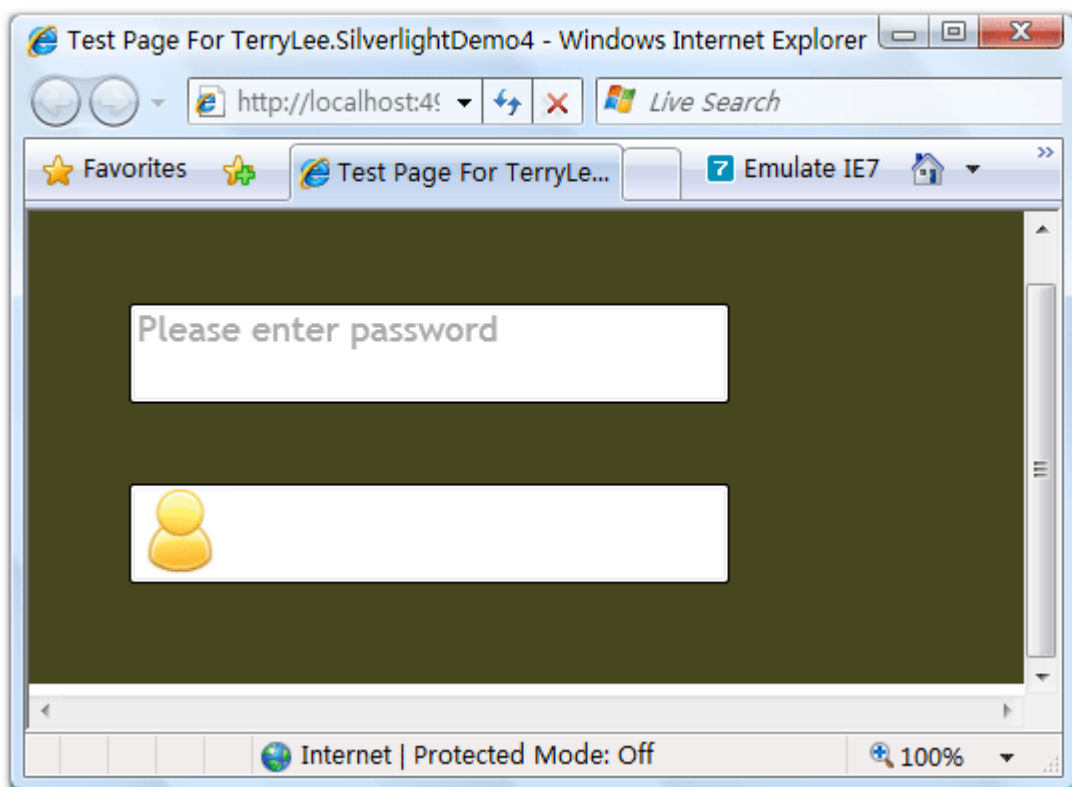
ToggleButton 控件有一个很重要的属性 IsThreeState，指示控件是否保持三种状态，如设为 false，则只会保持两种状态。

## 控件之 WatermarkedTextBox

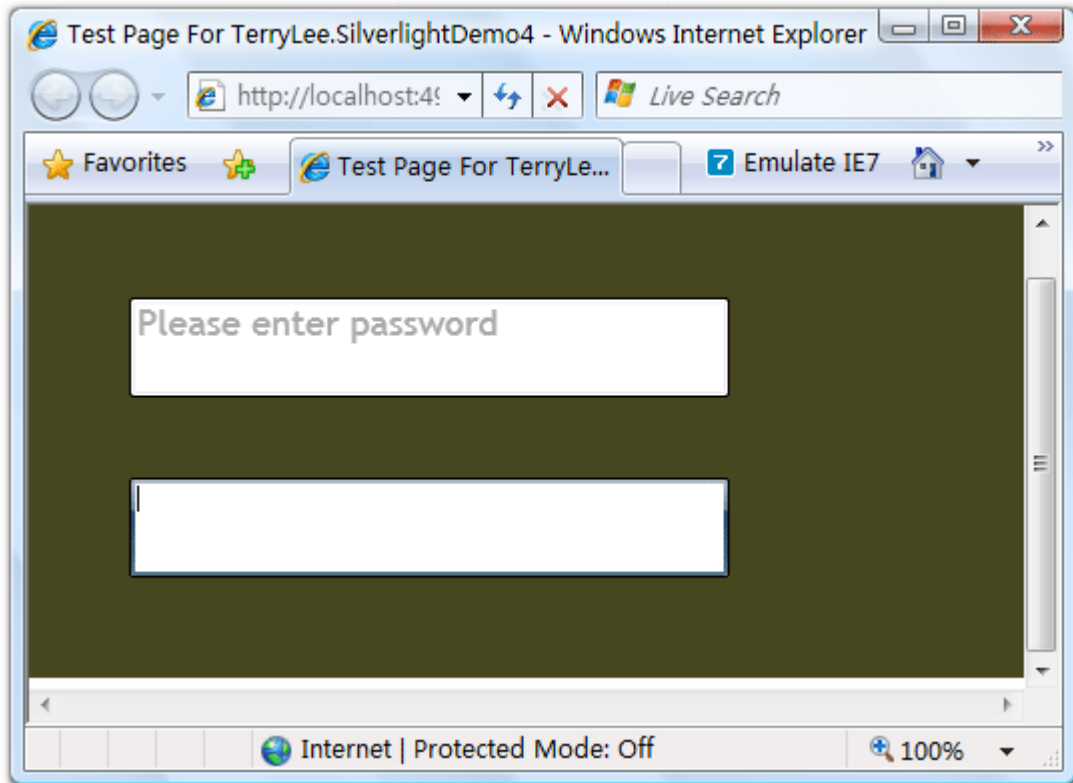
WatermarkedTextBox 即水印效果文本框，可以在文本框未获得焦点之前显示一段文字提示信息，也可以显示其它的控件。如下面的 XAML 中，在第一个 WatermarkedTextBox 中指定水印效果为显示一段文字提示“Please enter password”，而第二个则指定水印效果为一张图片：

```
Design XAML
Width="500" Height="240">
<Canvas Background="#46461F">
    <WatermarkedTextBox Canvas.Top="50" Canvas.Left="50"
        Width="300" Height="50" FontSize="1
        Watermark="Please enter password"/>
    |
    <WatermarkedTextBox Canvas.Top="140" Canvas.Left="50"
        Width="300" Height="50">
        <WatermarkedTextBox.Watermark>
            <Image Source="admin.png" HorizontalAlignment="
        </WatermarkedTextBox.Watermark>
    </WatermarkedTextBox>
</Canvas>
```

运行后效果如下所示:



单击其中一个文本框:

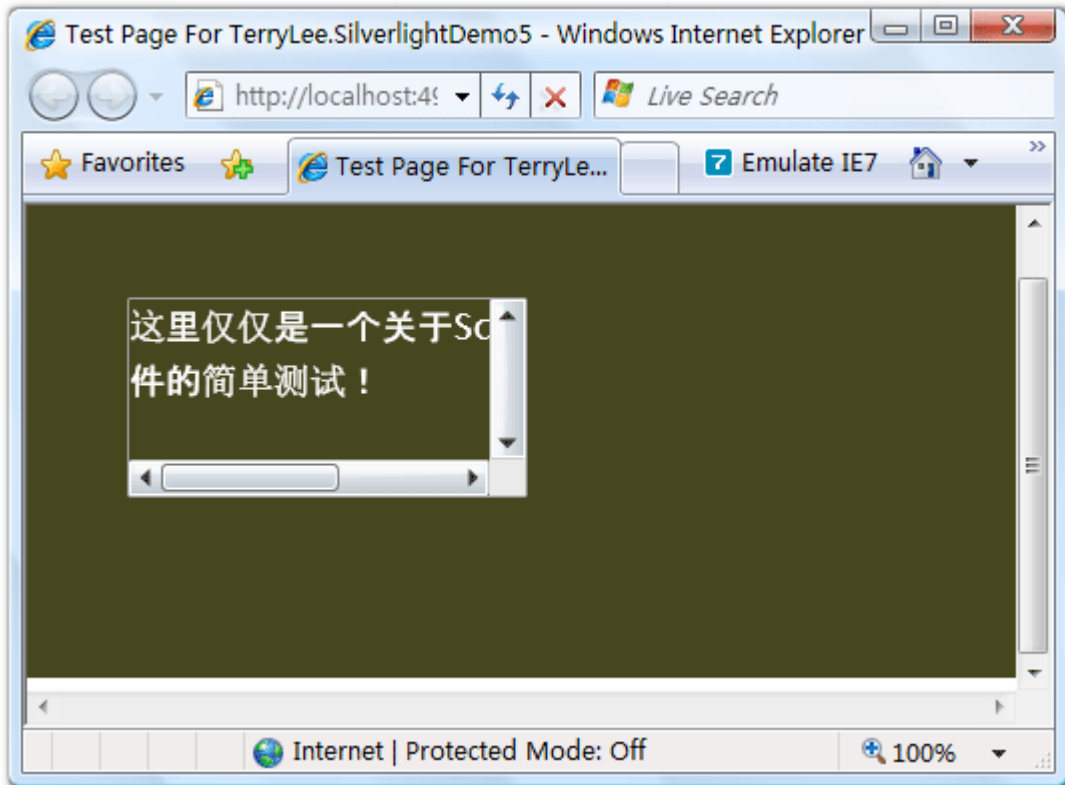


## 控件之 **ScrollViewer**

ScrollViewer 控件使用非常简单，当其中显示的内容超过它自身的大小时，就会有滚动条出现。通过属性 HorizontalScrollBarVisibility 和 VerticalScrollBarVisibility 来控制纵向和横向滚动条是否出现：

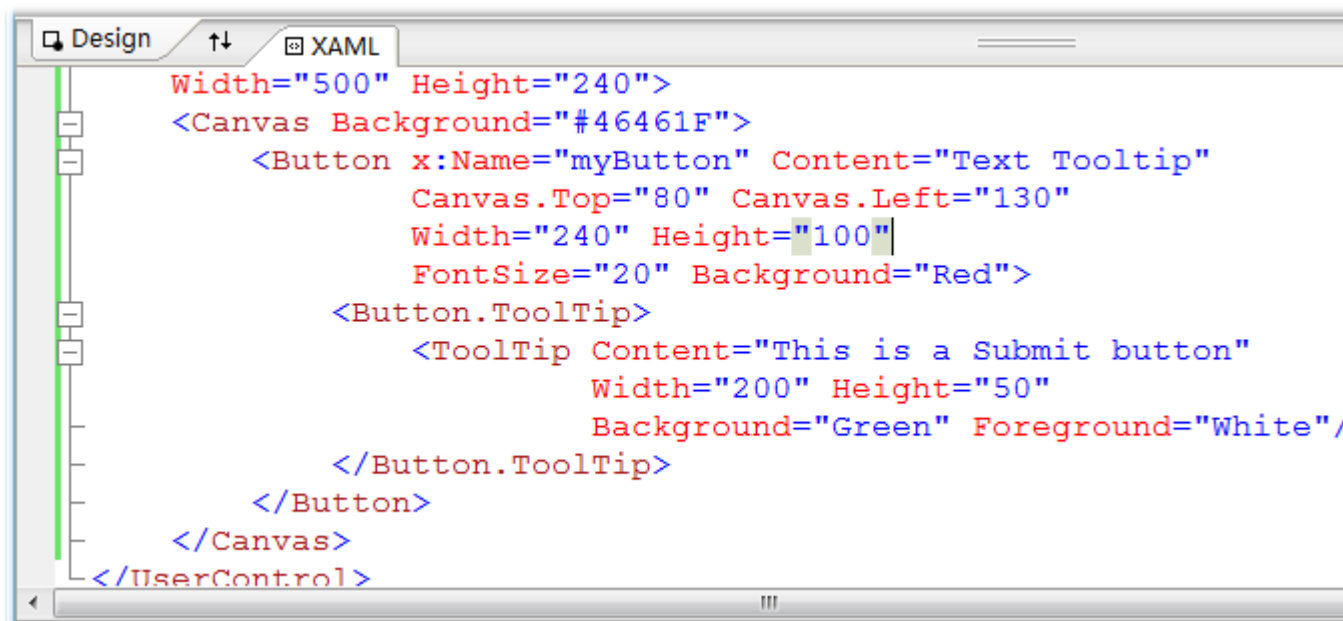
```
Design XAML
<Canvas Background="#46461F">
  <ScrollViewer Width="200" Height="100"
    Canvas.Top="50" Canvas.Left="50"
    HorizontalScrollBarVisibility="Auto"
    VerticalScrollBarVisibility="Visible">
    <TextBlock Width="300" TextWrapping="Wrap"
      Text="这里仅仅是一个关于ScrollViewer控件的简单测试！"
      Foreground="White" FontSize="18"/>
  </ScrollViewer>
</Canvas>
```

运行上面的示例：

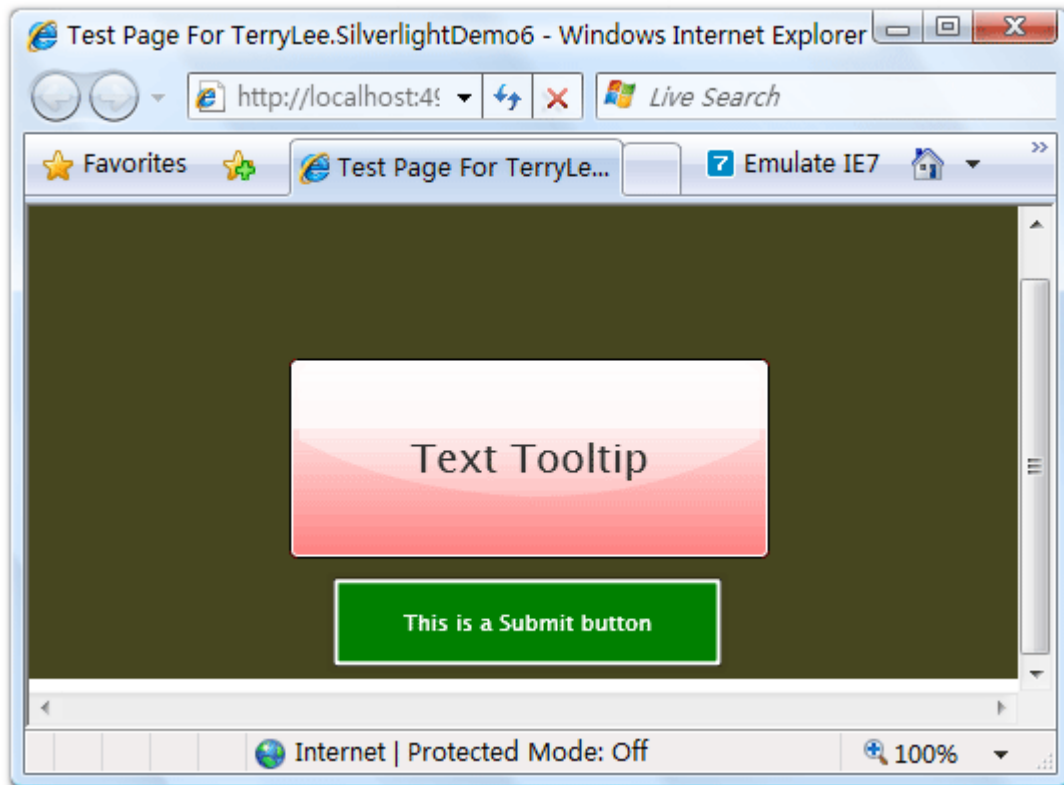


## 控件之 **ToolTip**

ToolTip 控件很多时候都用于其它控件的内嵌控件，如 Button 控件的 ToolTip 附加属性等。声明 ToolTip 控件如下面的 XAML 所示，当鼠标放上按钮时显示一个简单的信息提示：



运行后鼠标放上按钮时效果：



## 结束语

本文简单的演示了 Silverlight 2 中的几个控件的使用，对于 DataGrid 和 ListBox 等控件一般用来显示列表数据，将会在后面的数据绑定中讲述，而其它的诸如 Button、TextBlock 等控件的使用非常简单，这里不再讲述。

## 一步一步学 Silverlight 2 系列（3）：界面布局

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第三篇，学习 Silverlight 2 中的界面布局，Silverlight 2 中新增加了 Grid 和 Panel 两个布局容器，使得界面布局更加的强大和灵活。

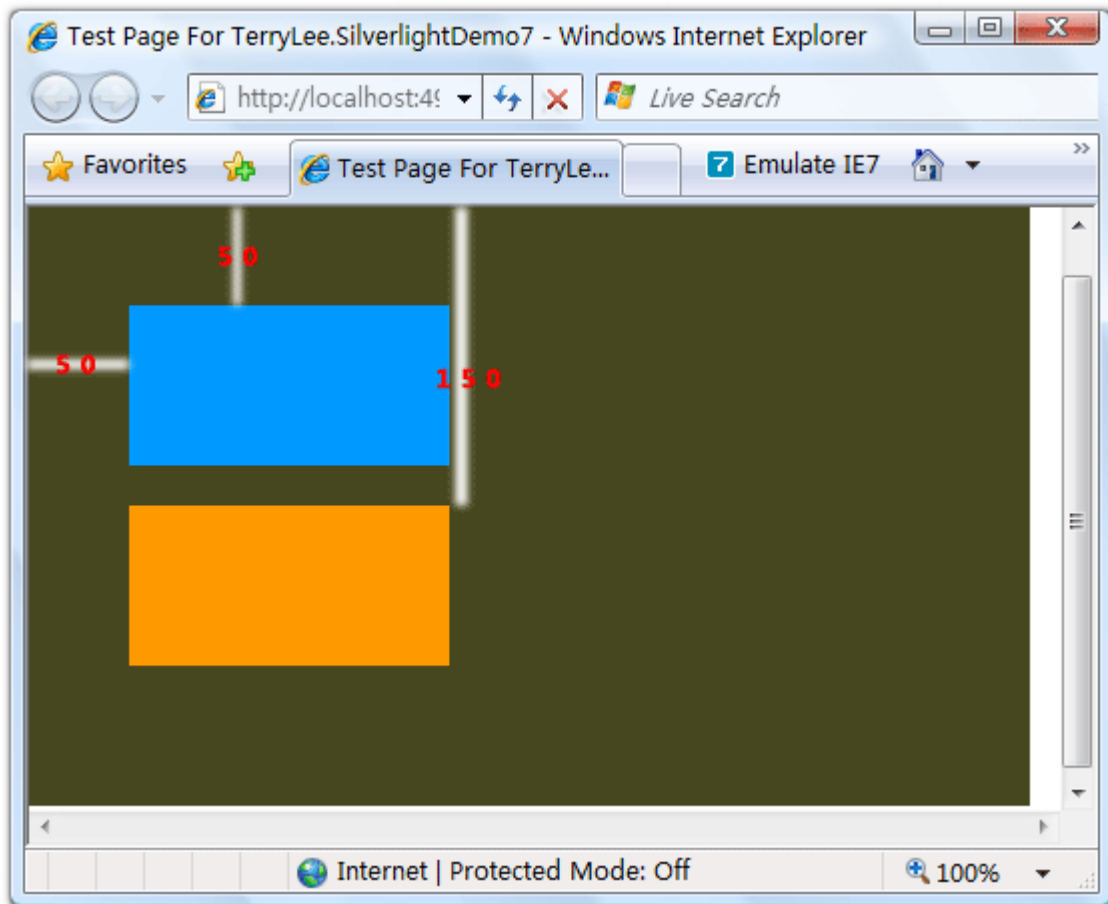
### Canvas 面板

Canvas 是在 Silverlight 1.0 时代就有的一种基础布局面板，它采用绝对坐标定位。可以使用附加属性 (Attached Property) 对 Canvas 中的元素进行定位，通过附加属性我们指定控件相对于其直接父容器 Canvas 控件的上、下、左、右坐标的位置。如下面的 XAML 声明了两个矩形，它们分别相对于父容器 Canvas 的左边距是 50，相对于父容器 Canvas 的上边距分别是 50 和 150：

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="500" Height="300">
<Canvas Background="#46461F">
    <Rectangle Fill="#0099FF" Width="160" Height="80"
        Canvas.Top="50" Canvas.Left="50"/>
    |
    <Rectangle Fill="#FF9900" Width="160" Height="80"
        Canvas.Top="150" Canvas.Left="50"/>
</Canvas>
</UserControl>
```

运行后界面的效果如下所示：

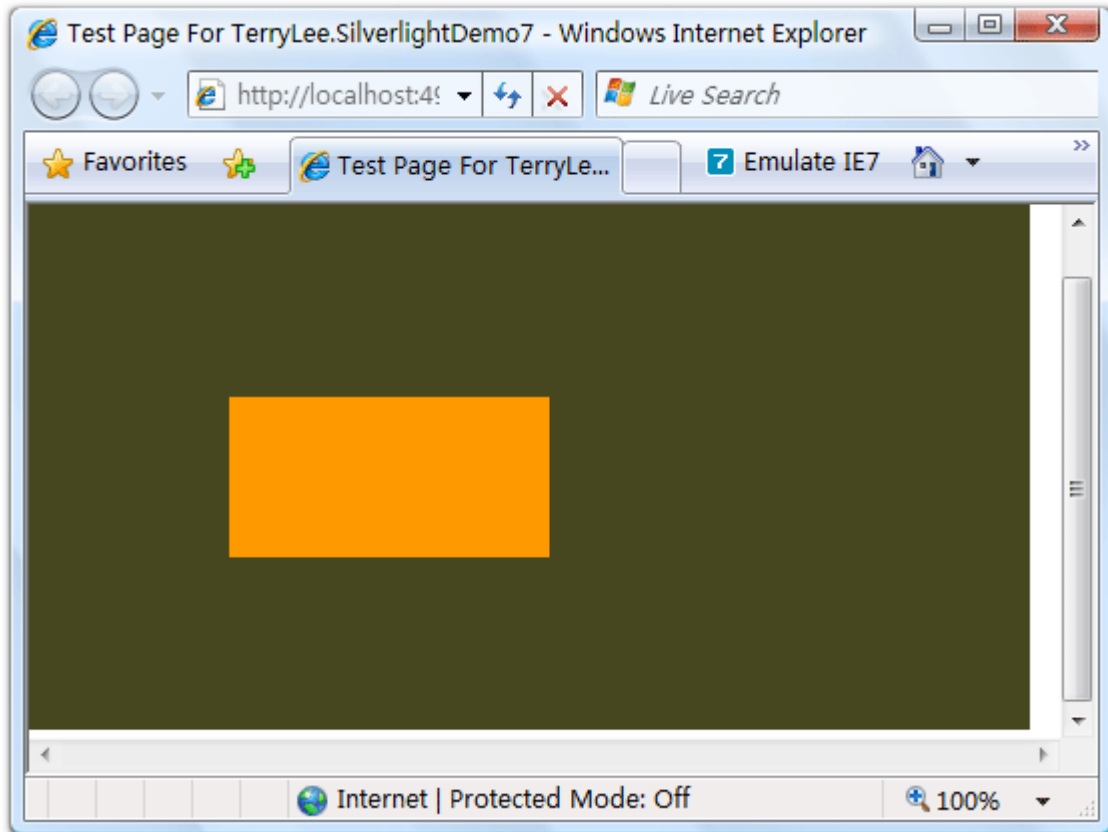




除了上面我们用到的 `Canvas.Top` 和 `Canvas.Left` 两个附加属性外，还有一个 `Canvas.ZIndex` 附加属性。如果指定了两个控件相对于父容器 `Canvas` 同样的边距，则后面声明的控件会覆盖前面声明的控件。这时我们可以使用 `Canvas.ZIndex` 属性来改变它们的显示顺序，如下面的 XAML 声明：

```
<Canvas Background="#46461F">
    <Rectangle Fill="#0099FF" Width="160" Height="80"
        Canvas.Top="100" Canvas.Left="100">
    <Rectangle Fill="#FF9900" Width="160" Height="80"
        Canvas.Top="100" Canvas.Left="100"/>
</Canvas>
```

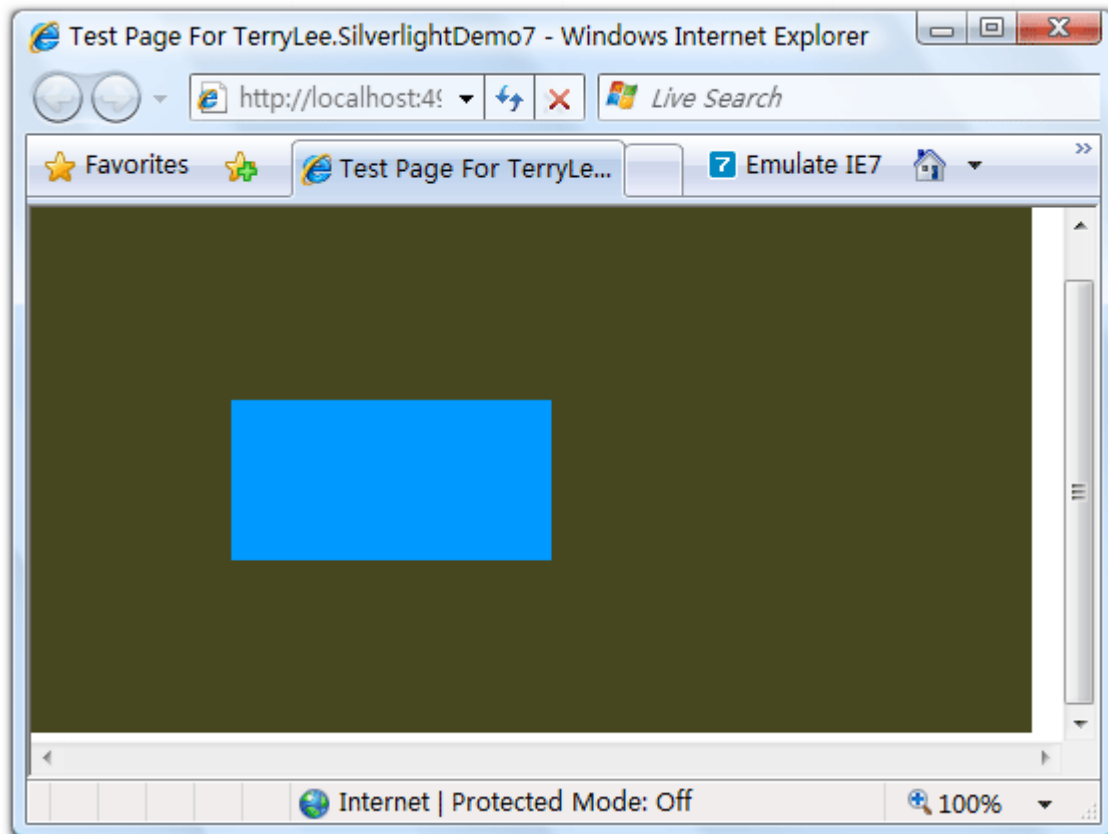
指定两个矩形相对于父容器 `Canvas` 的边距相同，这时默认的后声明的橙色矩形会覆盖蓝色矩形：



指定 Canvas.ZIndex 为 1

```
<Canvas Background="#46461F">  
    <Rectangle Fill="#0099FF" Width="160" Height="80"  
        Canvas.Top="100" Canvas.Left="100" Canvas.ZIndex="1"/>  
  
    <Rectangle Fill="#FF9900" Width="160" Height="80"  
        Canvas.Top="100" Canvas.Left="100"/>  
</Canvas>
```

将会让蓝色矩形显示在上面，值最大的显示在最上面：

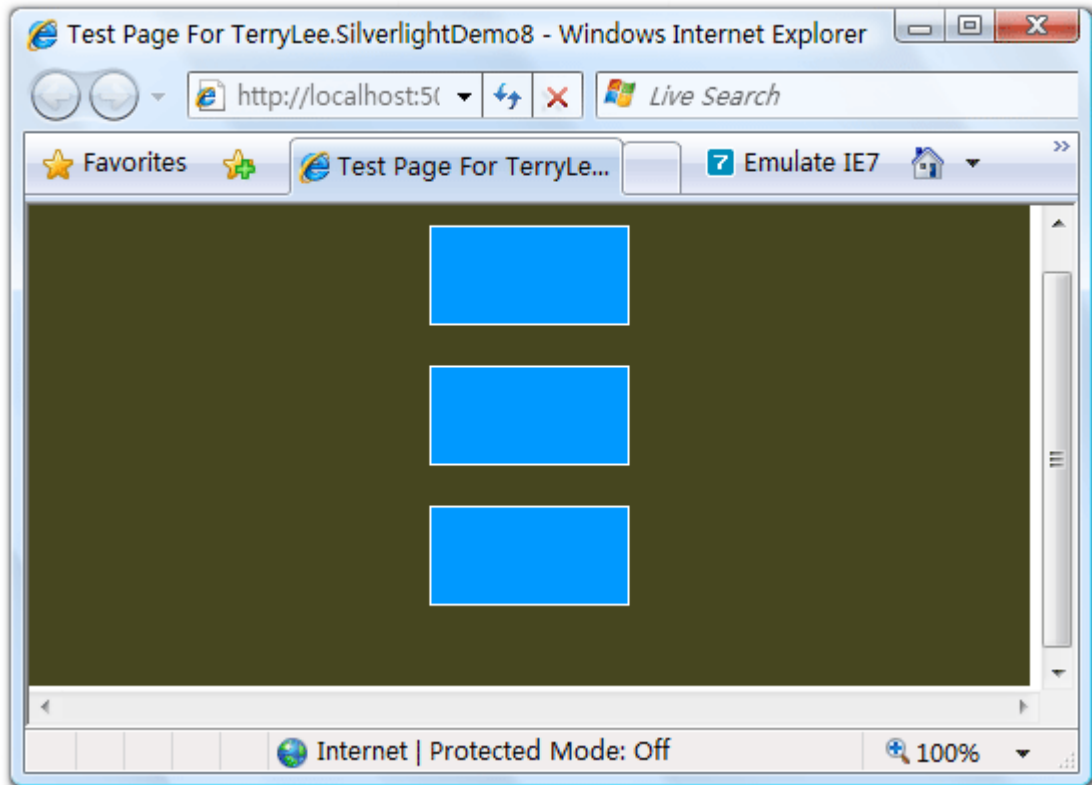


## **StackPanel**

StackPanel 支持用行或列的方式来进行页面布局，默认情况下所有的子元素会垂直的排列显示，如下面的 XAML 声明三个矩形：

```
<StackPanel Background="#46461F">
    <Rectangle Fill="#0099FF" Stroke="White"
        Width="100" Height="50" Margin="10"/>
    <Rectangle Fill="#0099FF" Stroke="White"
        Width="100" Height="50" Margin="10"/>
    <Rectangle Fill="#0099FF" Stroke="White"
        Width="100" Height="50" Margin="10"/>
</StackPanel>
```

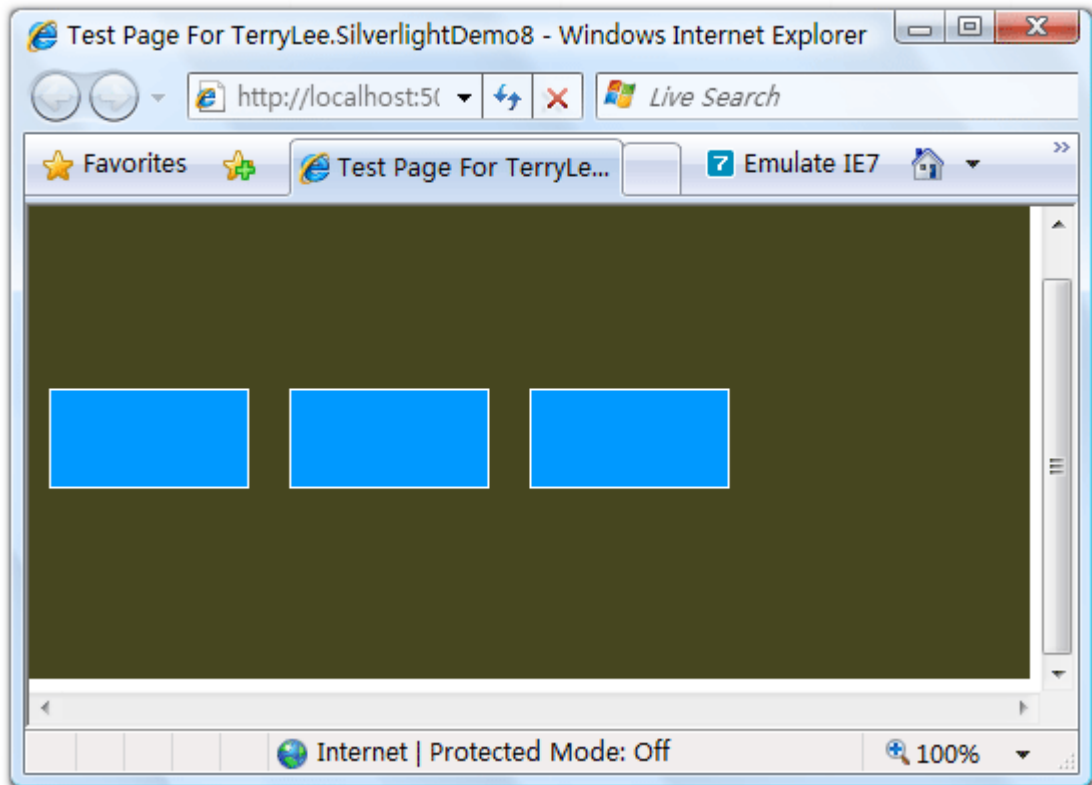
运行后在界面显示效果如下：



当然我们也可以指定为水平排列，通过 Orientation 属性指定：

```
<StackPanel Background="#46461F" Orientation="Horizontal">
    <Rectangle Fill="#0099FF" Stroke="White"
        Width="100" Height="50" Margin="10"/>
    <Rectangle Fill="#0099FF" Stroke="White"
        Width="100" Height="50" Margin="10"/>
    <Rectangle Fill="#0099FF" Stroke="White"
        Width="100" Height="50" Margin="10"/>
</StackPanel>
```

运行后界面显示效果如下：



在这里为了让各个控件之间有一定的距离，使用了 Margin 属性，该属性类似于 HTML 中的 Margin。

## **Grid**

Grid 控件类似与 HTML 中的 Table，只不过子元素不用放在单元格中。通过 <Grid.RowDefinitions> 和 <Grid.ColumnDefinitions> 来定义 Grid 的行和列，使用 Grid.Row 和 Grid.Column 两个附加属性指定子元素在 Grid 中显示的位置，这是一种非常灵活的布局方式。如下面的 XAML 声明：

```
<Grid x:Name="LayoutRoot" Background="#46461F" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height="120"/>
        <RowDefinition Height="*/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100"/>
        <ColumnDefinition Width="*/>
    </Grid.ColumnDefinitions>
```

```
<TextBlock Grid.Row="0" Grid.Column="0" Text="UserName:" VerticalAlignment="Center" Foreground="White"></TextBlock>

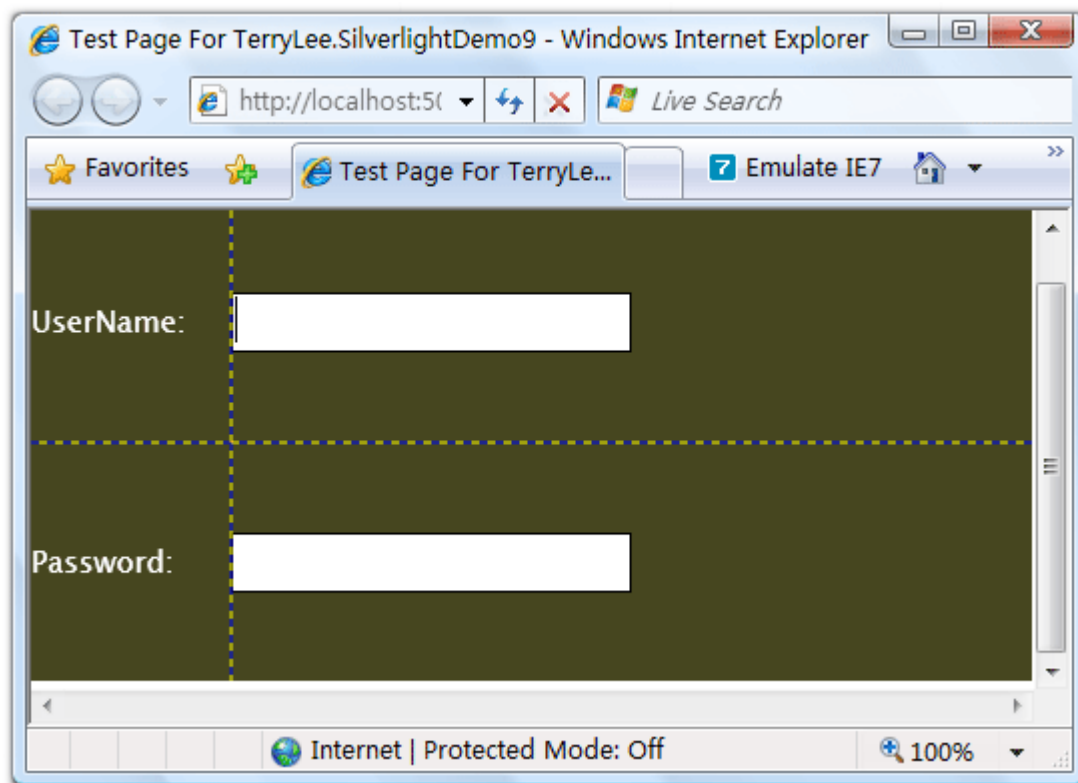
<TextBlock Grid.Row="1" Grid.Column="0" Text="Password:" VerticalAlignment="Center" Foreground="White"></TextBlock>

<TextBox Grid.Row="0" Grid.Column="1" Width="200" Height="30" HorizontalAlignment="Left"></TextBox>

<TextBox Grid.Row="1" Grid.Column="1" Width="200" Height="30" HorizontalAlignment="Left"></TextBox>

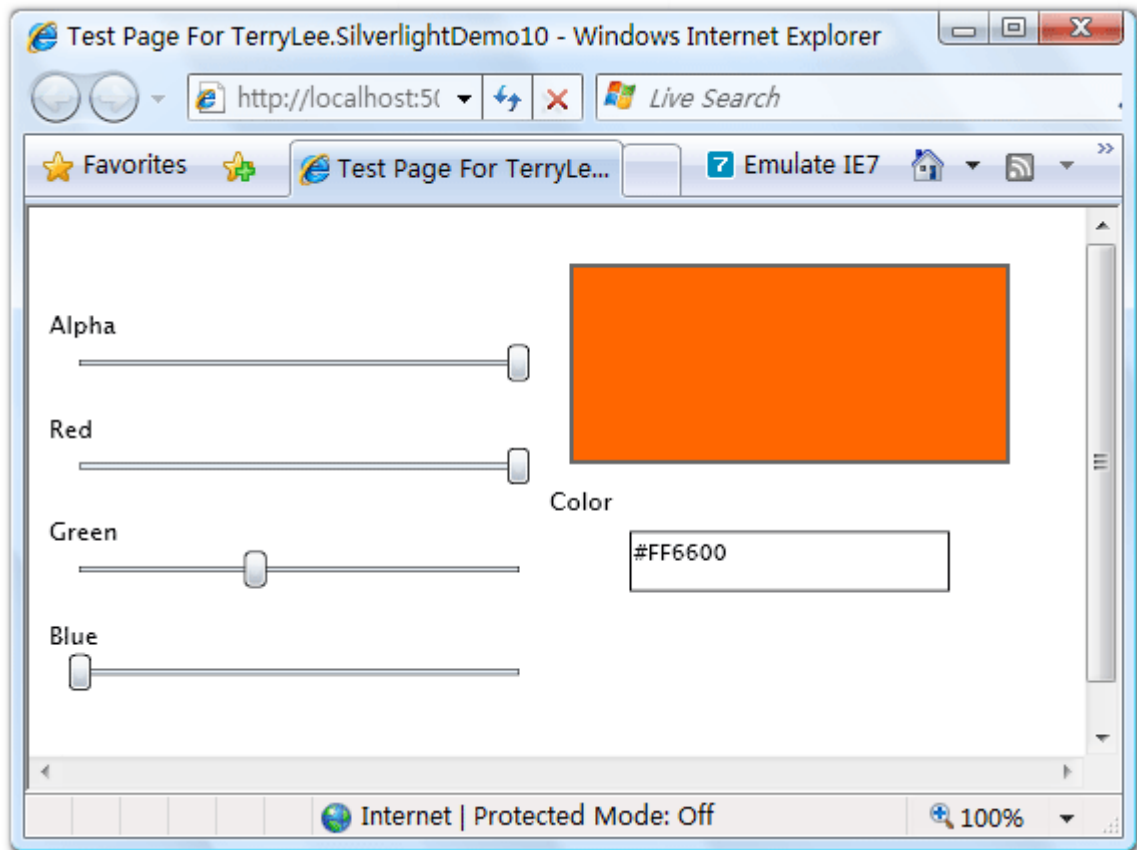
</Grid>
```

定义一个两行两列的 Grid，做一个简单的用户登录的布局，为了明显起见，把 ShowGridLines 属性设为 True，以便能够显示出边框线。同时，我们指定了第一行的高度为 120，而第二行的则是剩余的高度，用 \* 来指定。运行后效果如下：



## 综合实例

分别了解了上面的三个布局控件，接下来我们看一个综合实例，如何完成如下的一个取色器：



首先我们添加一个两行两列的 Grid 控件，分别指定行高和列宽：

```
<Grid x:Name="LayoutRoot" Background="White">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="260" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="120" />
    <RowDefinition Height="120" />
  </Grid.RowDefinitions>
</Grid>
```

添加颜色显示区域，用一个矩形显示，放入 Grid 的第 0 行第 1 列：

```
<Rectangle Grid.Row="0" Grid.Column="1" x:Name="PreviewColor">
```

```
Fill="#FF6600" Margin="10" Stroke="#666666" StrokeThickness="2"
/>
```

再添加颜色值显示区，嵌套一个 StackPanel 控件，让它里面的子控件垂直显示：

```
<StackPanel Grid.Row="1" Grid.Column="1" >
    <TextBlock FontSize="12">Color</TextBlock>
    <TextBox x:Name="HexColor" Width="160" Height="30" Text="#FF6600" Margin="10,5"
    FontSize="11"/>
</StackPanel>
```

左边用四个 Slider 控件和四个 TextBlock 控件显示，需要对 Grid 的行进行合并 Grid.RowSpan 属性：

```
<StackPanel Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" VerticalAlignment="Center"
">
    <TextBlock Text="Alpha" FontSize="12" Margin="10,15,0,0"/>
    <Slider x:Name="AlphaSlider" Margin="20,0,10,0" Maximum="255" Value="255" ValueC
hanged="RedSlider_ValueChanged"/>
    <TextBlock Text="Red" FontSize="12" Margin="10,15,0,0"/>
    <Slider x:Name="RedSlider" Margin="20,0,10,0" Maximum="255" Value="255" ValueCha
nged="RedSlider_ValueChanged"/>
    <TextBlock Text="Green" FontSize="12" Margin="10,15,0,0"/>
    <Slider x:Name="GreenSlider" Margin="20,0,10,0" Maximum="255" Value="102" ValueC
hanged="RedSlider_ValueChanged"/>
    <TextBlock Text="Blue" FontSize="12" Margin="10,15,0,0"/>
    <Slider x:Name="BlueSlider" Margin="20,0,10,0" Maximum="255" Value="0" ValueChan
ged="RedSlider_ValueChanged"/>
</StackPanel>
```

这样我们就完成了上面这样相对复杂的界面布局，对 Slider 控件添加事件处理程序：

```
private void RedSlider_ValueChanged(object sender, RoutedEventArgs<d
ouble> e)
{
```

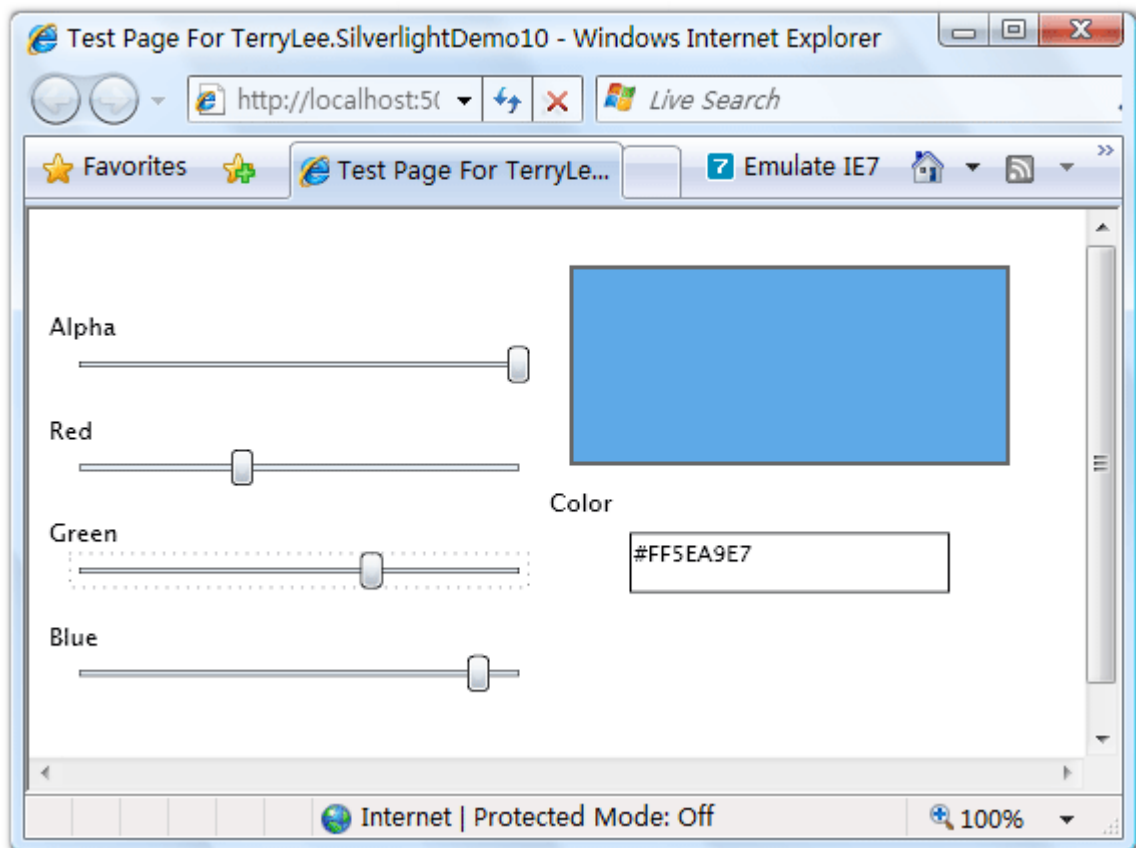


```
Color color = Color.FromArgb((byte)AlphaSlider.Value, (byte)RedSlider.Value, (byte)GreenSlider.Value, (byte)BlueSlider.Value);

PreviewColor.Fill = new SolidColorBrush(color);

HexColor.Text = color.ToString();
}
```

运行后，可以选取不同的颜色值：



## 结束语

关于界面布局就说到这里，在 Silverlight 2 中，通过上面的三种布局控件相结合，可以进行非常强大和灵活的界面布局。

## 一步一步学 Silverlight 2 系列（4）：鼠标事件处理

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了许多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第四篇，学习 Silverlight 2 中的鼠标事件处理，支持的鼠标事件包括 MouseMove 、 MouseEnter 、 MouseLeave 、 MouseLeftButtonDown、 MouseLeftButtonUp。

### 声明事件

对于鼠标事件我们可以附加到任何 Silverlight 对象上面，如下面的 XAML 声明，为两个圆形添加上 MouseEnter 和 MouseLeave 事件：

```
<Canvas Background="#46461F">
    <Ellipse Width="120" Height="120" Fill="Orange"
        Canvas.Top="60" Canvas.Left="80"
        MouseEnter="OnMouseEnter"
        MouseLeave="OnMouseLeave"/>

    <Ellipse Width="120" Height="120" Fill="Orange"
        Canvas.Top="60" Canvas.Left="280"
        MouseEnter="OnMouseEnter"
        MouseLeave="OnMouseLeave"/>
</Canvas>
```

编写事件处理程序，鼠标放上去时和鼠标移开时分别改变圆形的填充色：

```
void OnMouseEnter(object sender, MouseEventArgs e)
{
```

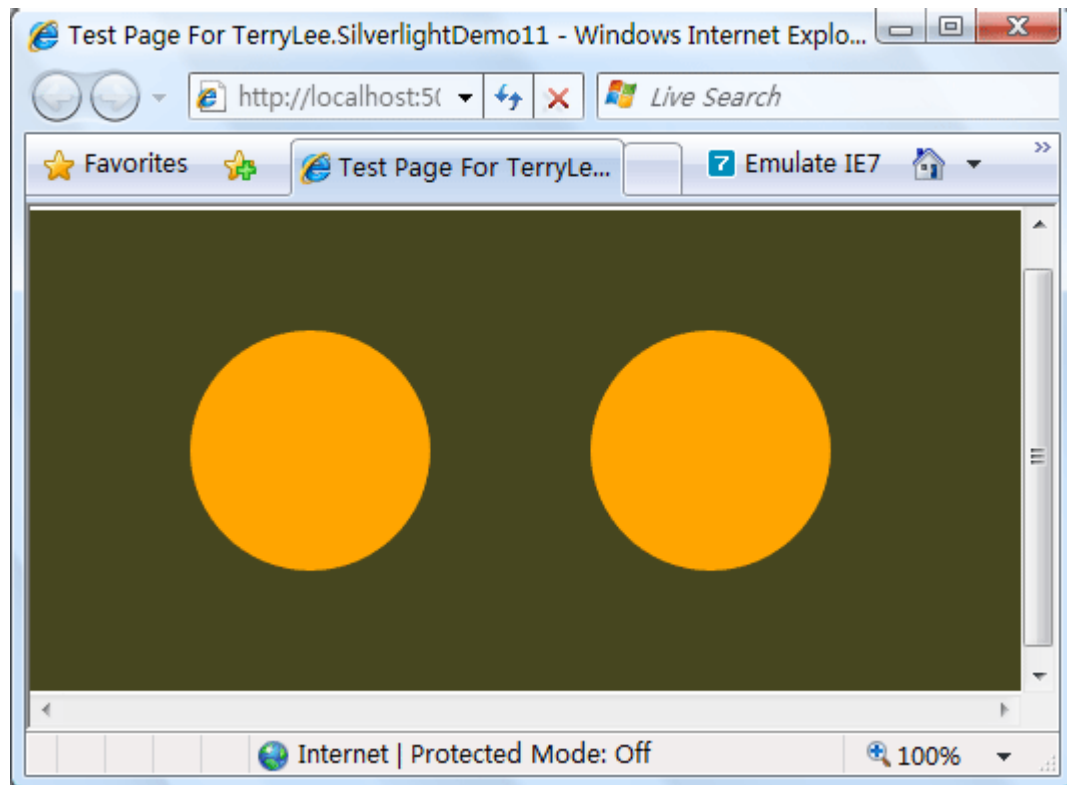
```
Ellipse ell = sender as Ellipse;

ell.Fill = new SolidColorBrush(Colors.Yellow);
}

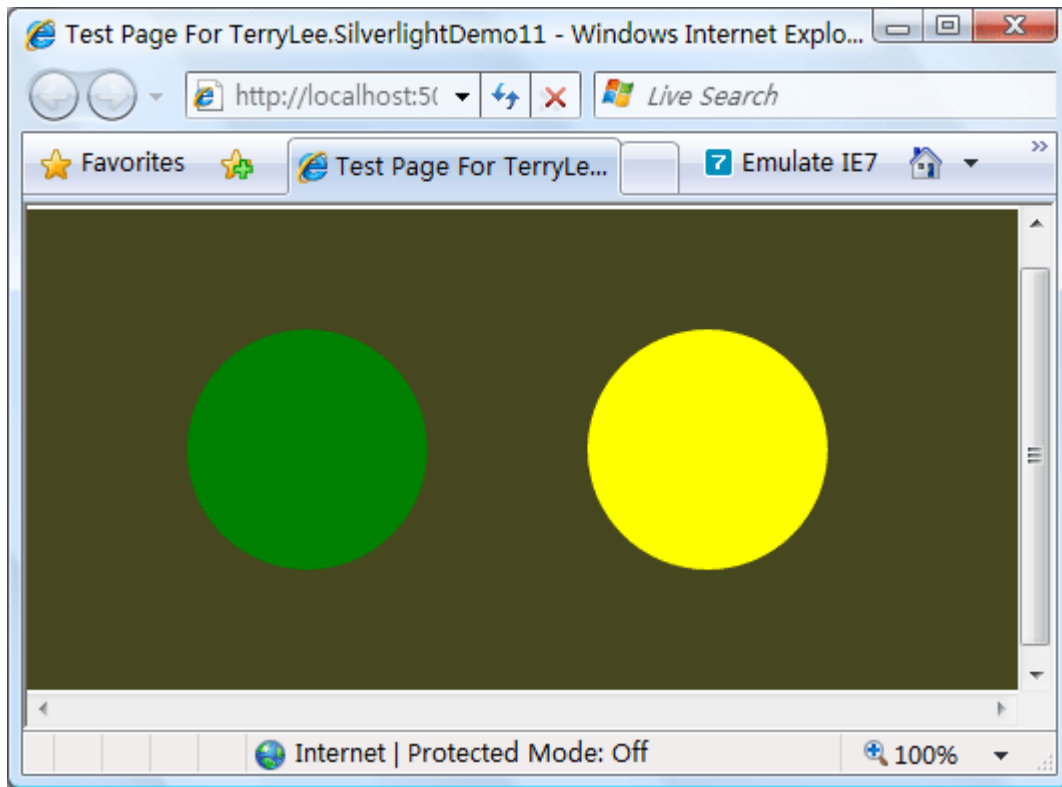
void OnMouseLeave(object sender, MouseEventArgs e)
{
    Ellipse ell = sender as Ellipse;

    ell.Fill = new SolidColorBrush(Colors.Green);
}
}
```

运行后效果如下：



分别在两个圆形上放上鼠标并移开后如下所示：



## 使用代码管理事件

除了在 XAML 中声明事件外，也可以直接使用代码来注册事件，简单的修改一下上面的 XAML 文件，去掉事件的声明并为两个圆形分别加上 Name:

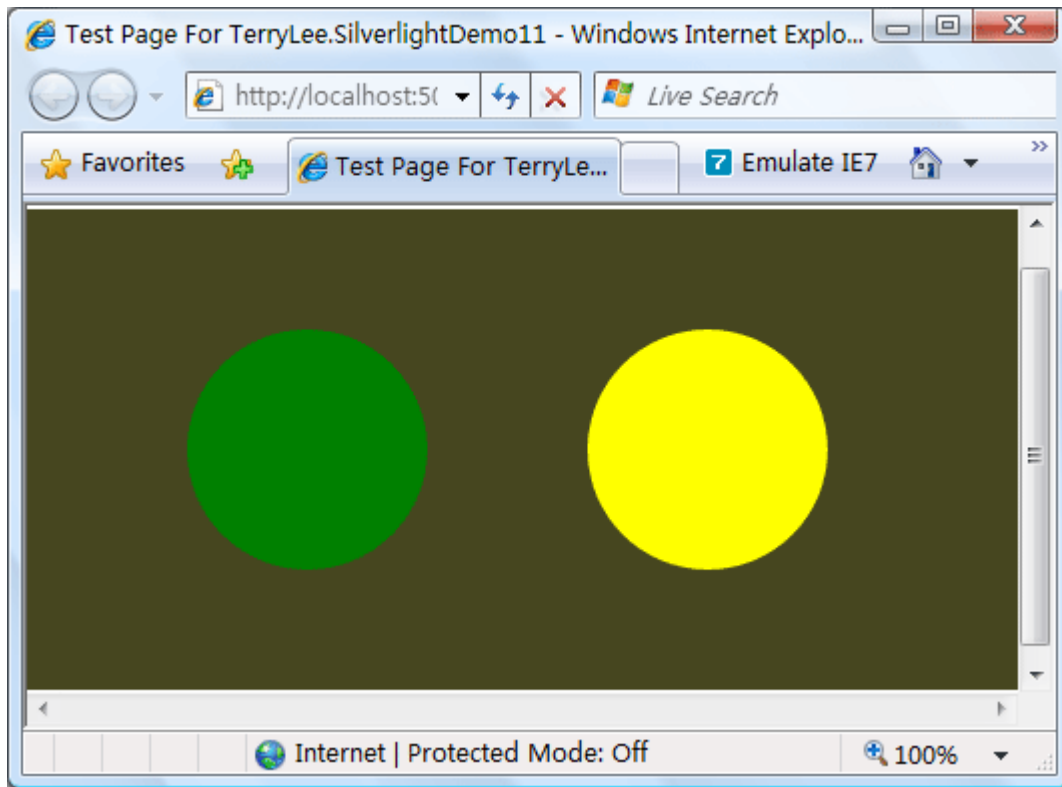
```
<Canvas Background="#46461F">
    <Ellipse x:Name="ellipse1" Width="120" Height="120" Fill="Orange"
        Canvas.Top="60" Canvas.Left="80"/>
    <Ellipse x:Name="ellipse2" Width="120" Height="120" Fill="Orange"
        Canvas.Top="60" Canvas.Left="280"/>
</Canvas>
```

在代码中进行事件注册:

```
public partial class Page : UserControl
{
    public Page()
```

```
{  
    InitializeComponent();  
    ellipse1.MouseEnter += new MouseEventHandler(OnMouseEnter);  
    ellipse1.MouseLeave += new MouseEventHandler(OnMouseLeave);  
    ellipse2.MouseEnter += new MouseEventHandler(OnMouseEnter);  
    ellipse2.MouseLeave += new MouseEventHandler(OnMouseLeave);  
}  
  
void OnMouseEnter(object sender, MouseEventArgs e)  
{  
    Ellipse ell = sender as Ellipse;  
    ell.Fill = new SolidColorBrush(Colors.Yellow);  
}  
  
void OnMouseLeave(object sender, MouseEventArgs e)  
{  
    Ellipse ell = sender as Ellipse;  
    ell.Fill = new SolidColorBrush(Colors.Green);  
}  
}
```

运行后可以看到跟上面一样的效果：



## 事件数据

所有的鼠标事件都使用 `MouseButtonEventArgs` 和 `MouseEventArgs` 作为事件数据，通过这两个参数可以获取相关事件数据，使用 `GetPosition` 方法或者 `Source`、`Handled` 属性。如下面的 XAML 声明：

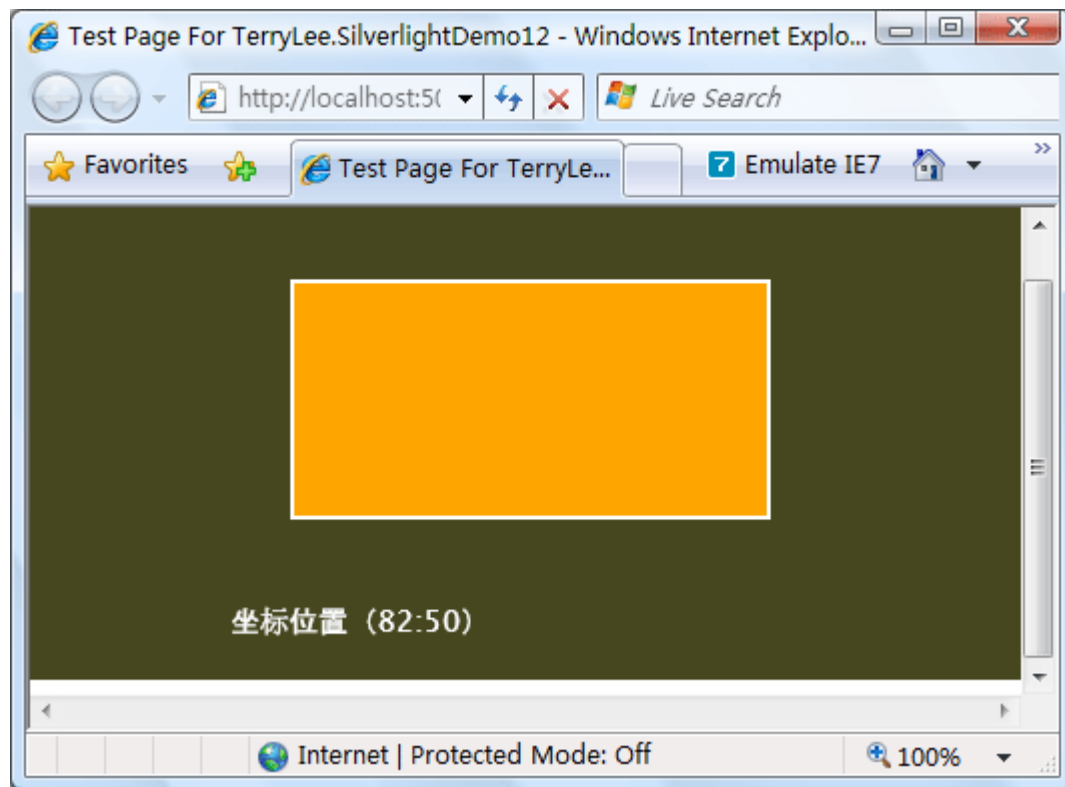
```
<Canvas Background="#46461F">
    <Rectangle Fill="Orange" Stroke="White" StrokeThickness="2"
        Canvas.Top="40" Canvas.Left="130"
        Width="240" Height="120"
        MouseMove="Rectangle_MouseMove"/>
    <TextBlock x:Name="Status" Foreground="White" Text="Status"
        Canvas.Left="100" Canvas.Top="200"/>
</Canvas>
```

为矩形添加 `MouseMove` 事件处理，在鼠标移动时我们获取当前坐标位置，并显示出来：

```
private void Rectangle_MouseMove(object sender, MouseEventArgs e)
{
```

```
Point p = e.GetPosition(e.Source as FrameworkElement);  
  
Status.Text = String.Format("坐标位置 ( {0} : {1} ) ", p.X, p.Y);  
  
}
```

运行后在矩形中移动鼠标，效果如下：



## 路由事件

在 Silverlight 中，提供了事件路由，使得我们可以在父节点上接收和处理来自于子节点的事件，Silverlight 中的路由事件采用了冒泡路由策略。在鼠标事件中 `MouseDown`、`MouseUp`、`MouseMove` 三个事件都支持路由事件，而 `MouseEnter`、`MouseLeave` 两个事件不支持。下面的 XAML 中我们为 Canvas 对象声明了一个 `MouseDown` 事件：

```
<Canvas x:Name="ParentCanvas" Background="#46461F"  
        MouseLeftButtonDown="ParentCanvas_MouseLeftButtonDown">  
    <Rectangle x:Name="RecA" Fill="Orange" Stroke="White" StrokeThickness="2"  
        Canvas.Top="40" Canvas.Left="60"  
        Width="160" Height="100"/>
```

```
<Rectangle x:Name="RecB" Fill="LightBlue" Stroke="White" StrokeThickness="2"
           Canvas.Top="40" Canvas.Left="240"
           Width="160" Height="100"/>

<TextBlock x:Name="Status" Foreground="White" Text="Status"
           Canvas.Left="100" Canvas.Top="200"/>

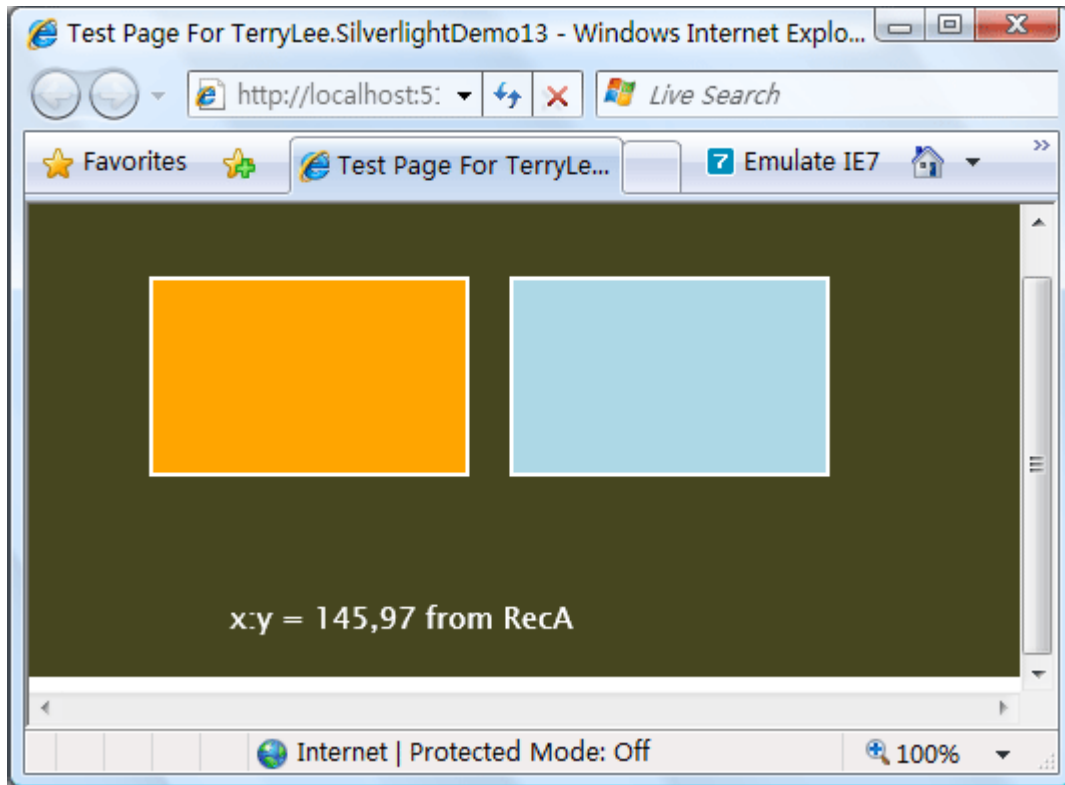
</Canvas>
```

添加 MouseLeftButtonDown 事件处理程序，显示当前鼠标按下时的坐标，并显示源控件名称：

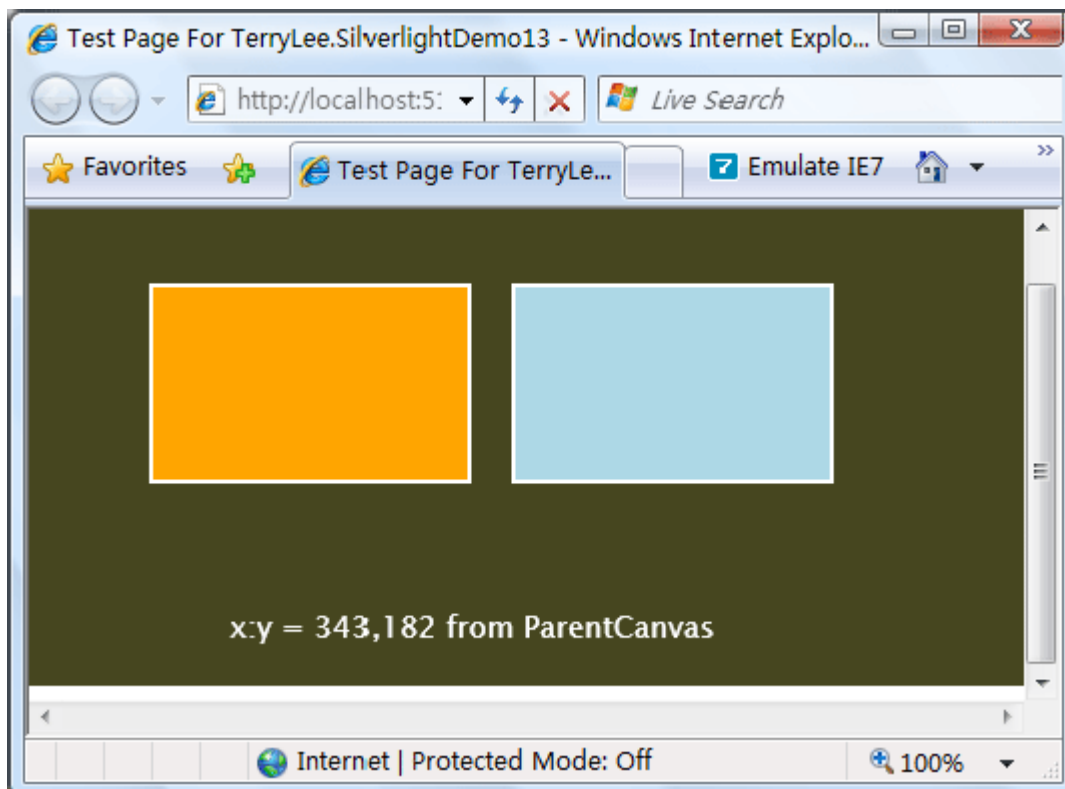
```
private void ParentCanvas_MouseLeftButtonDown(object sender, MouseButtonEventArgs
e)
{
    String msg = "x:y = " + e.GetPosition(sender as FrameworkElement).ToString();
    msg += " from " + (e.Source as FrameworkElement).Name;
    Status.Text = msg;
}
```

运行后在 RecA 上按下鼠标：





在 Canvas 上按下鼠标:



结束语

本文简单介绍了 Silverlight 2 中关于鼠标事件处理的一些知识，包括事件注册、获取事件数据、路由事件等。在下一篇中，我们将使用这些鼠标事件来实现一个简单的拖放功能。

## 一步一步学 Silverlight 2 系列（5）：实现简单的拖放功能

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第五篇，利用前面讲过的鼠标事件处理实现简单的拖放功能。

### 准备 XAML

在实现拖放功能中，分为三个步骤：

1. 按下鼠标，触发 MouseLeftButtonDown 事件，选择要拖动的对象。
2. 移动鼠标，触发 MouseMove 事件，移动选择的对象。
3. 放开鼠标，触发 MouseLeftButtonUp 事件，停止捕捉事件。

做一个简单的界面，用一个按钮来显示拖放，如下 XAML 声明：

```
<Canvas Background="#46461F">
    <Button
        MouseLeftButtonDown="OnMouseDown"
        MouseMove="OnMouseMove"
        MouseLeftButtonUp="OnMouseUp"
        Canvas.Left="50" Canvas.Top="50" Background="Red"
        FontSize="18"
        Width="160" Height="80">
        <Button.Content>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
                VerticalAlignment="Center">
                <Image Source="smile_6.png"></Image>
            </StackPanel>
        </Button.Content>
    </Button>
</Canvas>
```

```
        <TextBlock Text="拖动我" VerticalAlignment="Center" Margin="10"></T
extBlock>

        </StackPanel>

        </Button.Content>

    </Button>

</Canvas>
```

这里为了界面显示效果，使用了控件模板，后续会专门讲到。

## 开始拖放操作

开始拖放操作，实现 `MouseLeftButtonDown` 事件处理程序，用两个全局变量来记录当前鼠标的位置和鼠标是否保持移动。

```
bool trackingMouseMove = false;

Point mousePosition;

void OnMouseDown(object sender, MouseButtonEventArgs e)
{
    FrameworkElement element = sender as FrameworkElement;

    mousePosition = e.GetPosition(null);

    trackingMouseMove = true;

    if (null != element)
    {
        element.CaptureMouse();

        element.Cursor = Cursors.Hand;
    }
}
```

## 移动对象

移动对象，实现 `MouseMove` 事件处理程序，计算元素的位置并更新，同时更新鼠标的位置。

```

void OnMouseMove(object sender, MouseEventArgs e)
{
    FrameworkElement element = sender as FrameworkElement;

    if (trackingMouseMove)
    {
        double deltaV = e.GetPosition(null).Y - mousePosition.Y;
        double deltaH = e.GetPosition(null).X - mousePosition.X;

        double newTop = deltaV + (double)element.GetValue(Canvas.TopProperty);
        double newLeft = deltaH + (double)element.GetValue(Canvas.LeftProperty);

        element.SetValue(Canvas.TopProperty, newTop);
        element.SetValue(Canvas.LeftProperty, newLeft);

        mousePosition = e.GetPosition(null);
    }
}

```

## 完成拖放操作

完成拖放操作，实现 MouseLeftButtonUp 事件处理程序。

```

void OnMouseUp(object sender, MouseButtonEventArgs e)
{
    FrameworkElement element = sender as FrameworkElement;

    trackingMouseMove = false;

    element.ReleaseMouseCapture();

    mousePosition.X = mousePosition.Y = 0;

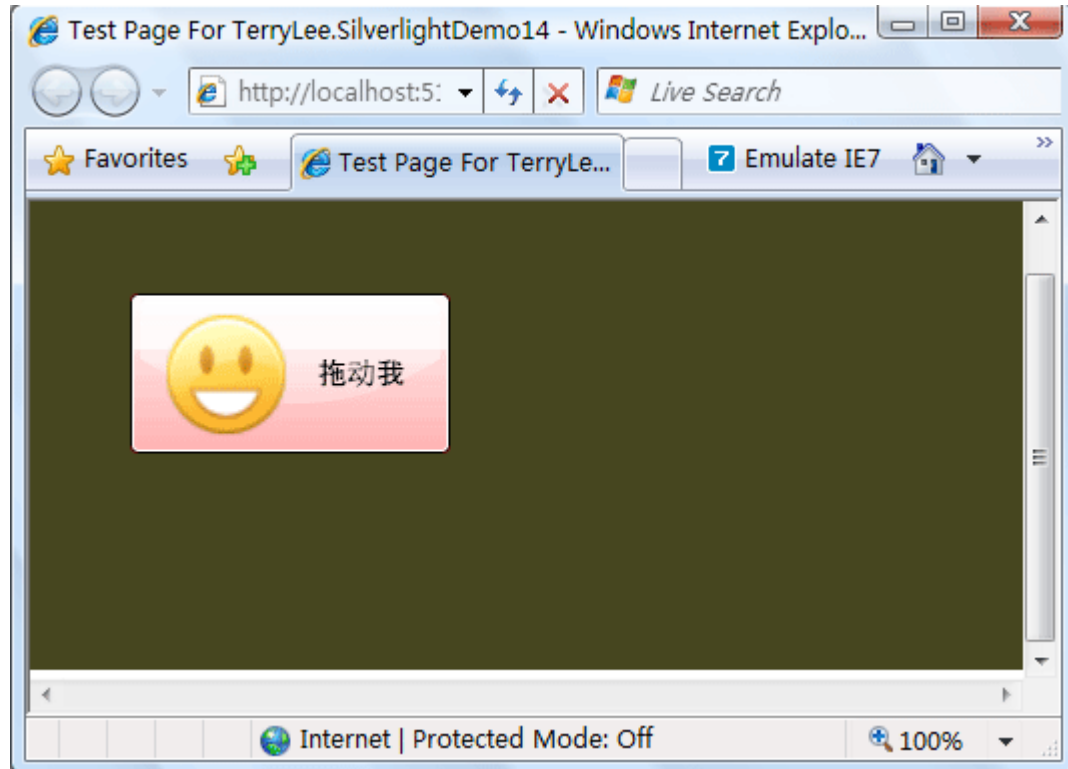
    element.Cursor = null;
}

```

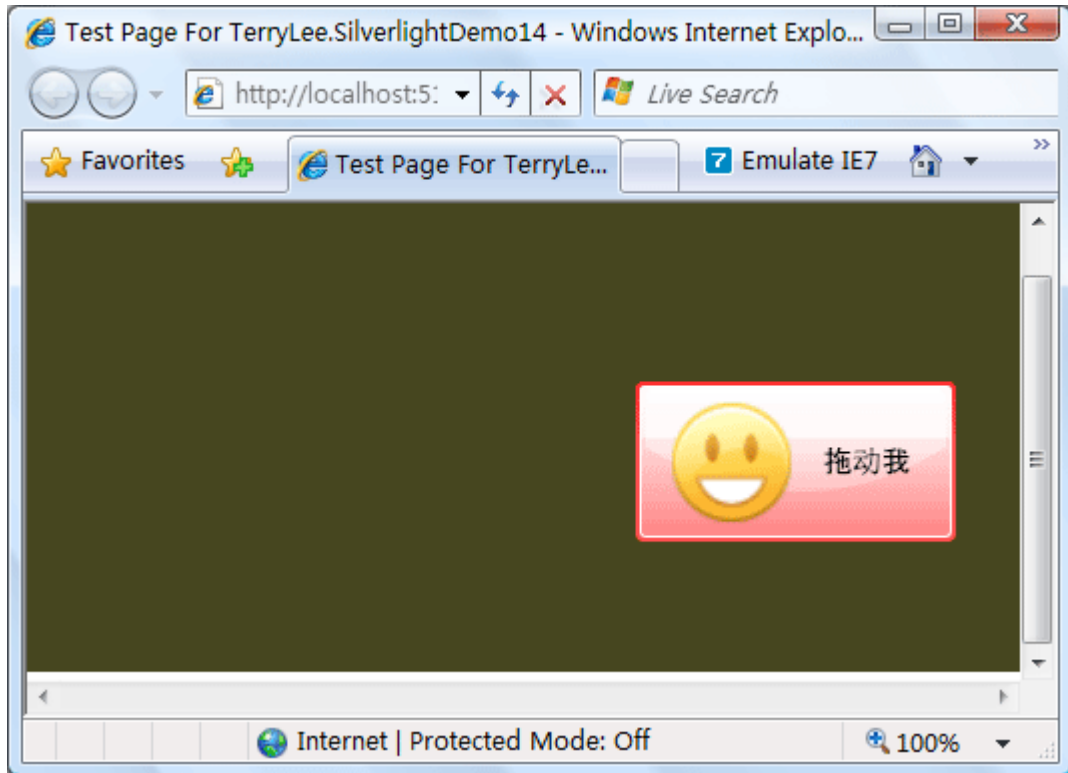
```
}
```

## 效果显示

最终，完成后的效果如下



拖动按钮



## 结束语

本文实现了一个简单的拖放功能（示例来自于 Silverlight 2 SDK），点击[下载](#)文本示例代码。

## 一步一步学 Silverlight 2 系列（6）：键盘事件处理

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了许多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第六篇，介绍 Silverlight 中的键盘处理事件，在 Silverlight 2 中，支持 KeyDown 和 KeyUp 两个事件。

### 声明事件

所有的事件声明过程都是一样的，在 XAML 中或者是在代码中进行注册。

```
<Canvas x:Name="LayoutRoot" Background="#46461F">
    <Ellipse x:Name="ellipse" Width="120" Height="120" Fill="Orange"
        Canvas.Top="50" Canvas.Left="160"
        Stroke="White" StrokeThickness="2"
        KeyUp="ellipse_KeyUp"
        KeyDown="ellipse_KeyDown"/>
</Canvas>
```

或者在代码中注册：

```
public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
        this.ellipse.KeyUp += new KeyEventHandler(ellipse_KeyUp);
        this.ellipse.KeyDown += new KeyEventHandler(ellipse_KeyDown);
    }
}
```



```

}

private void ellipse_KeyUp(object sender, KeyEventArgs e)
{

}

private void ellipse_KeyDown(object sender, KeyEventArgs e)
{

}
}

```

## 使用事件参数 KeyEventArgs

使用事件参数可以获取到事件数据，可以使用的属性有 Key、PlatformKeyCode、Handled、Source。

```

private void ellipse_KeyUp(object sender, KeyEventArgs e)
{
    if (e.Key == Key.R)
    {
        //.....
    }
    else if (e.Key == Key.Ctrl && e.Key == Key.U)
    {
        //.....
    }
}
}

```

在事件数据中，Handled 有时候非常有用，可以用来判断事件是否已经处理。

## 键盘路由事件

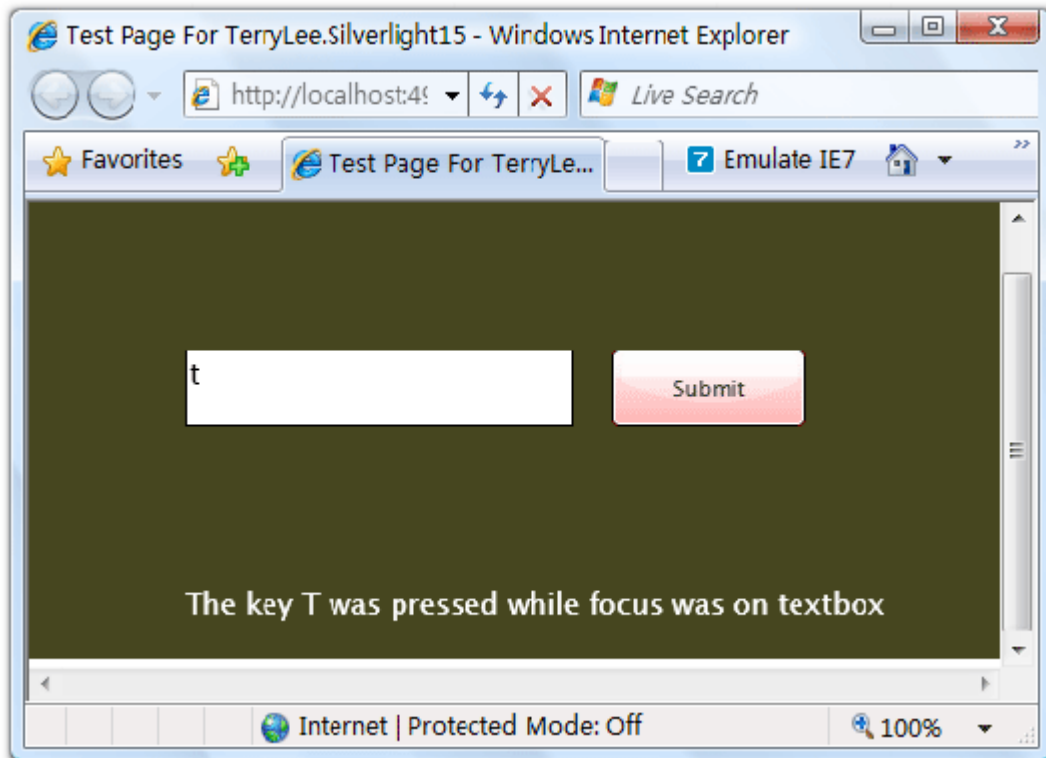
键盘事件 KeyDown 和 KeyUp 都支持路由事件，如下面的示例。

```
<Canvas x:Name="LayoutRoot" Background="#46461F" KeyUp="LayoutRoot_KeyUp">
    <TextBox x:Name="textbox" Width="200" Height="40"
        Canvas.Top="80" Canvas.Left="80"/>
    <Button x:Name="button" Width="100" Height="40"
        Canvas.Top="80" Canvas.Left="280"
        Background="Red" Margin="20 0 0 0" Content="Submit"/>
    <TextBlock x:Name="Status" Foreground="White" Text="Status"
        Canvas.Left="80" Canvas.Top="200"/>
</Canvas>
```

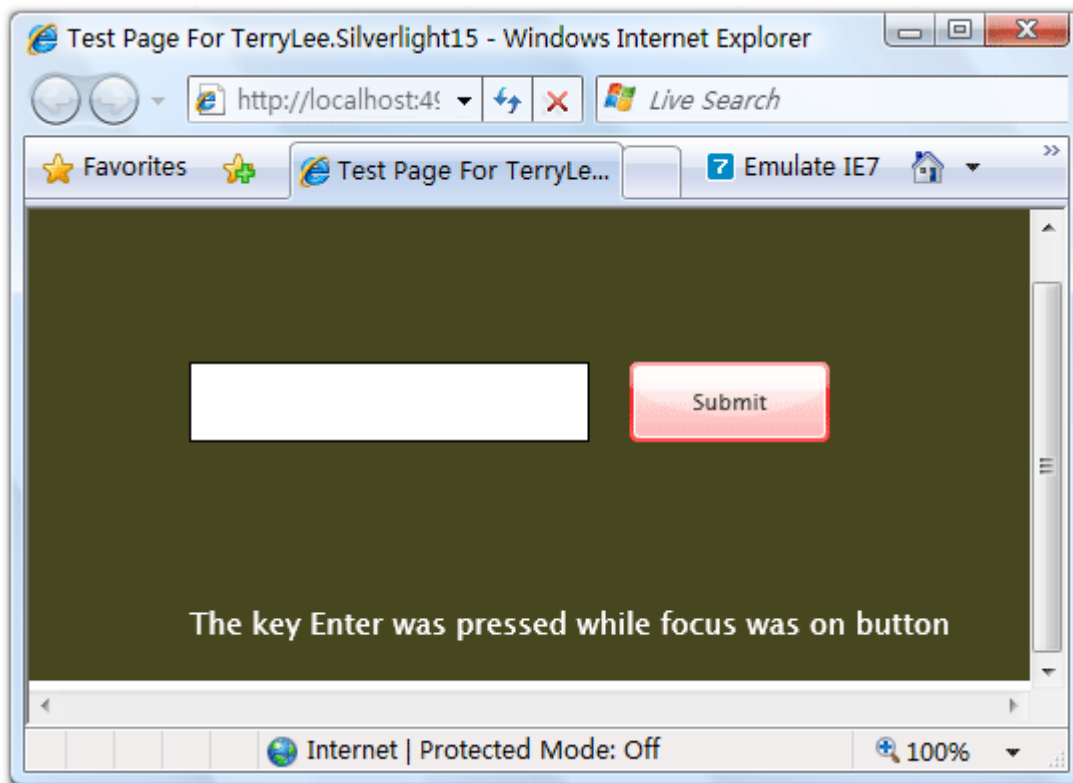
为 Canvas 注册了一个 KeyUp 事件，编写事件处理程序。

```
private void LayoutRoot_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key != Key.Unknown)
    {
        String msg = "The key " + e.Key.ToString();
        msg += " was pressed while focus was on " + (e.Source as FrameworkElement).
Name;
        statusTextBlock.Text = msg;
    }
}
```

运行程序，当文本框获得焦点并输入 t 时



按钮获得焦点



结束语

关于键盘事件都简单的介绍到这儿，希望对大家有用。

## 一步一步学 Silverlight 2 系列（7）：全屏模式支持

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了许多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第七篇，介绍如何在 Silverlight 2 中使用全屏模式。

### 实现全屏模式

全屏模式有时候是非常有用的，在 Silverlight 中，提供了很好的支持。实现起来也非常的简单，其实只有一行代码，编写一个简单的 XAML。

```
<Canvas Background="#46461F">
    <Button x:Name="toggleButton" Background="Red" Width="200" Height="80"
        Canvas.Top="80" Canvas.Left="150" Content="Toggle Full Screen"
        FontSize="20" Click="toggleButton_Click"/>
    <Image x:Name="image" Source="smile_6.png"
        Canvas.Top="100" Canvas.Left="40"></Image>
</Canvas>
```

引入命名空间

```
using System.Windows.Interop;
```

在按钮单击事件中添加实现代码。

```
private void toggleButton_Click(object sender, RoutedEventArgs e)
{
    Content contentObject = Application.Current.Host.Content;
    contentObject.IsFullScreen = !contentObject.IsFullScreen;
}
```

```
}
```

获取当前的 Silverlight 插件“Content”对象，并设置 IsFullScreen 属性。运行后单击按钮将会变为全屏模式，再次单击按钮（或者按 Esc 键）返回普通模式。



Press ESC to exit full-screen mode.  
<http://localhost>

## 捕获相关事件

有时候，我们需要在全屏模式和普通模式之间切换时，添加一个其它的代码，这时可以使用事件 FullScreenChanged。

```
public Page()  
{  
    InitializeComponent();  
    Application.Current.Host.Content.FullScreenChanged += new EventHandler(Content_  
    FullScreenChanged);  
}
```

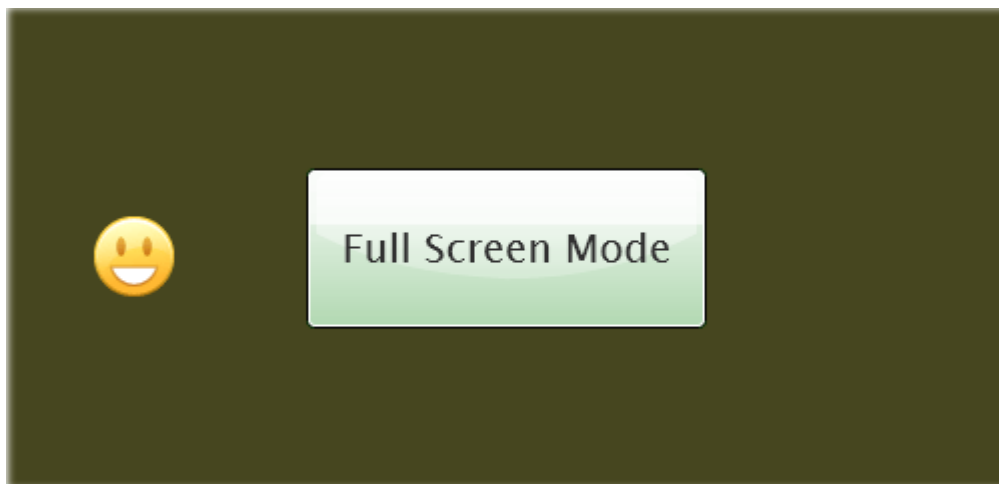
实现事件处理

```
private void Content_FullScreenChanged(object sender, EventArgs e)  
{
```

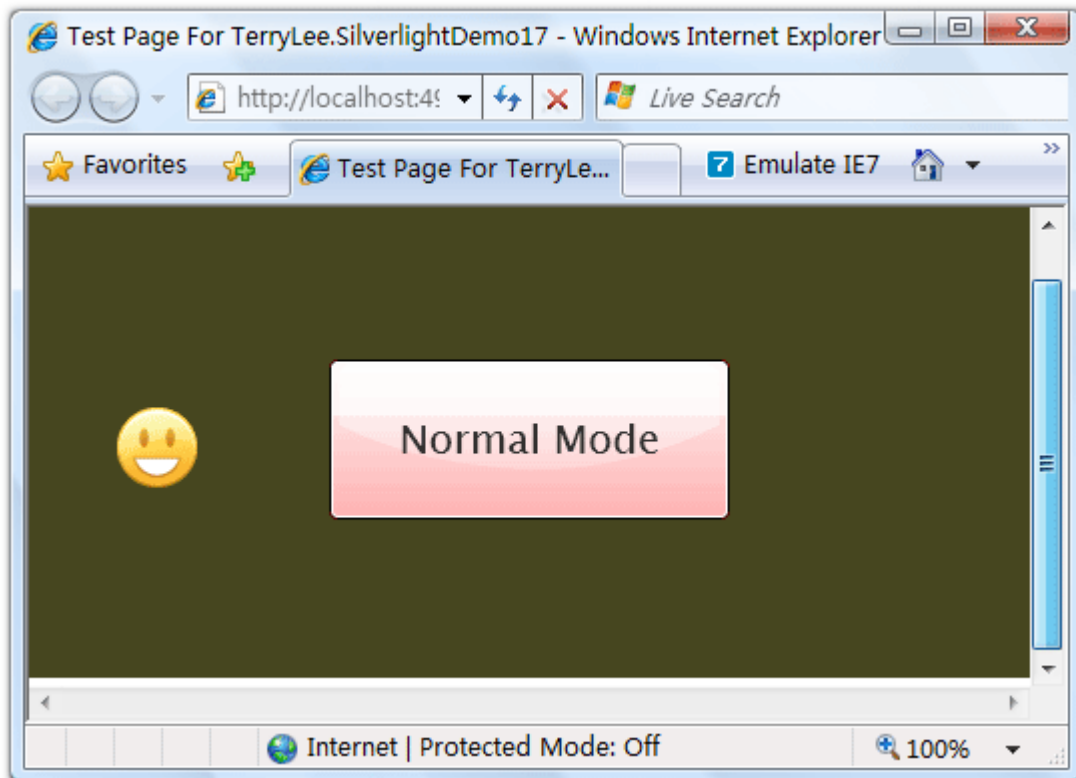
```
Content contentObject = Application.Current.Host.Content;

if (contentObject.IsFullScreen)
{
    toggleButton.Background = new SolidColorBrush(Colors.Green);
    toggleButton.Content = "Full Screen Mode";
}
else
{
    toggleButton.Background = new SolidColorBrush(Colors.Red);
    toggleButton.Content = "Normal Mode";
}
}
```

在普通模式和全屏模式之间切换时，改变按钮的背景色和文字。运行后点击按钮：



切换为普通模式：



完整的代码如下：

```
public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();

        Application.Current.Host.Content.FullScreenChanged += new EventHandler(Content_FullScreenChanged);
    }

    private void toggleButton_Click(object sender, RoutedEventArgs e)
    {
        Content contentObject = Application.Current.Host.Content;
        contentObject.IsFullScreen = !contentObject.IsFullScreen;
    }
}
```



```
private void Content_FullScreenChanged(object sender, EventArgs e)
{
    Content contentObject = Application.Current.Host.Content;
    if (contentObject.IsFullScreen)
    {
        toggleButton.Background = new SolidColorBrush(Colors.Green);
        toggleButton.Content = "Full Screen Mode";
    }
    else
    {
        toggleButton.Background = new SolidColorBrush(Colors.Red);
        toggleButton.Content = "Normal Mode";
    }
}
}
```

## 结束语

本文简单介绍了 Silverlight 2 中对于全屏模式的支持，你可以从[这里](#)下载本文示例代码。

## 一步一步学 Silverlight 2 系列（8）：使用样式封装控件观感

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了许多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第八篇，主要介绍在 Silverlight 中使用 Style 元素封装控件观感

Silverlight 支持一种 Style 机制，它允许我们把控件的属性值封装成可重用的资源。我们可以把这些样式声明保存在独立于页面的其他文件中，然后就可以在一个应用程序中跨控件和页面重用（甚至跨多个应用程序重用）。在做一些基本定制的场景下，概念上类似于在 HTML 中重用 CSS。

### 内联样式

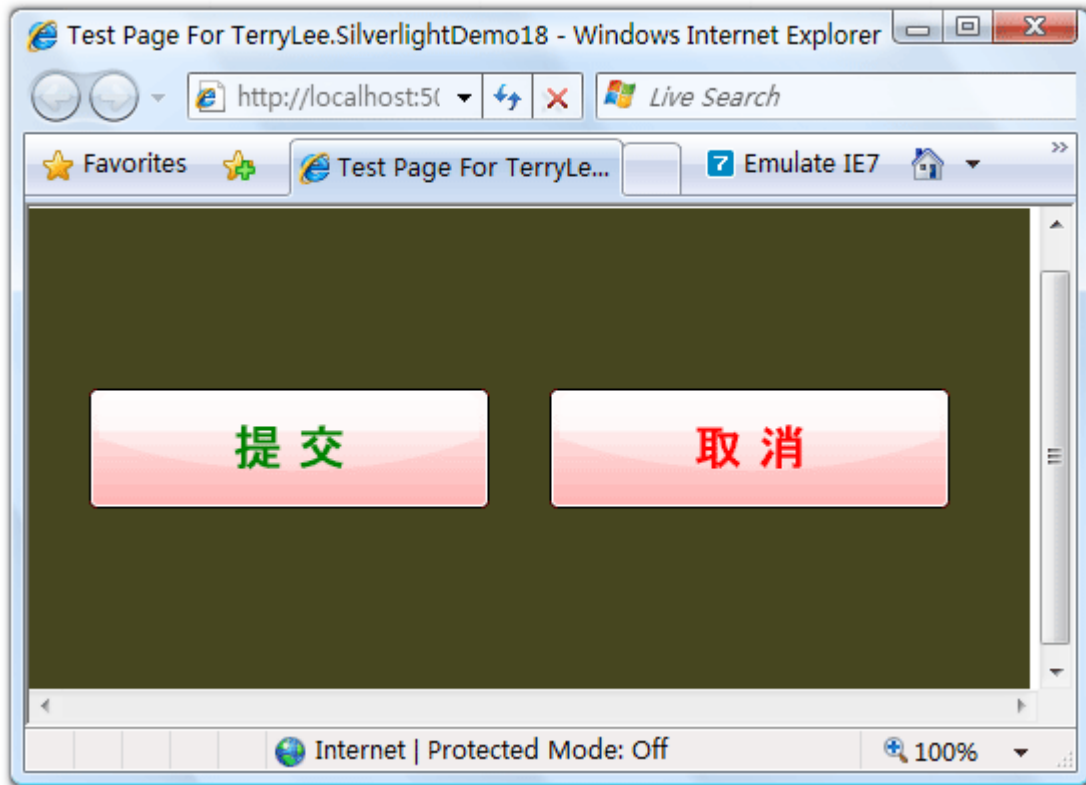
内联样式这个概念其实跟我们在 HTML 中指定元素的样式一样，在 XAML 中通过属性来设置，如下面这段 XAML，我们添加了两个按钮，并在其中设置字体的样式：

```
<Canvas Background="#46461F">
    <Button Width="200" Height="60" Background="Red"
        Canvas.Top="90" Canvas.Left="30" Content="提交"
        FontFamily="微软雅黑"
        FontSize="24"
        FontWeight="Bold"
        Foreground="Green"/>

    <Button Width="200" Height="60" Background="Red"
        Canvas.Top="90" Canvas.Left="260" Content="取消"
        FontFamily="微软雅黑"
        FontSize="24"
        FontWeight="Bold"/>
```

```
Foreground="Red"/>
</Canvas>
```

运行后显示效果如下：



使用内联样式不是一种很好的做法，样式不可重用，页面 XAML 代码混乱等，这些缺点其实类似于在 HTML 中直接设置元素的样式。一种推荐的方式是应该使用全局的样式。

## 全局样式

为了更好使样式能够重用，并且减少 XAML 中的代码，推荐使用全局样式。在 App.xaml 中定义两个样式

```
<Application.Resources>
    <Style x:Key="button1" TargetType="Button">
        <Setter Property="FontFamily" Value="微软雅黑"></Setter>
        <Setter Property="FontSize" Value="24"></Setter>
        <Setter Property="Foreground" Value="Green"></Setter>
        <Setter Property="Background" Value="Red"></Setter>
    </Style>
</Application.Resources>
```

```

</Style>

<Style x:Key="button2" TargetType="Button">

    <Setter Property="FontFamily" Value="微软雅黑"></Setter>

    <Setter Property="FontSize" Value="24"></Setter>

    <Setter Property="Foreground" Value="Red"></Setter>

    <Setter Property="Background" Value="Red"></Setter>

</Style>

</Application.Resources>

```

通过 Style 元素指定,需要设置唯一的一个 Key,类似于 CSS 中的类名或者 ASP.NET 2.0 中 Skin 功能,并且通过 TargetType 指定该样式将使用在哪类控件上,每一个属性都用 Setter 来指定。在 XAML 中,通过 StaticResource 标记句法来指定具体的样式:

```

<Canvas Background="#46461F">

    <Button Width="200" Height="60"

        Canvas.Top="90" Canvas.Left="30" Content="提交"

        Style="{StaticResource button1}"/>

    <Button Width="200" Height="60"

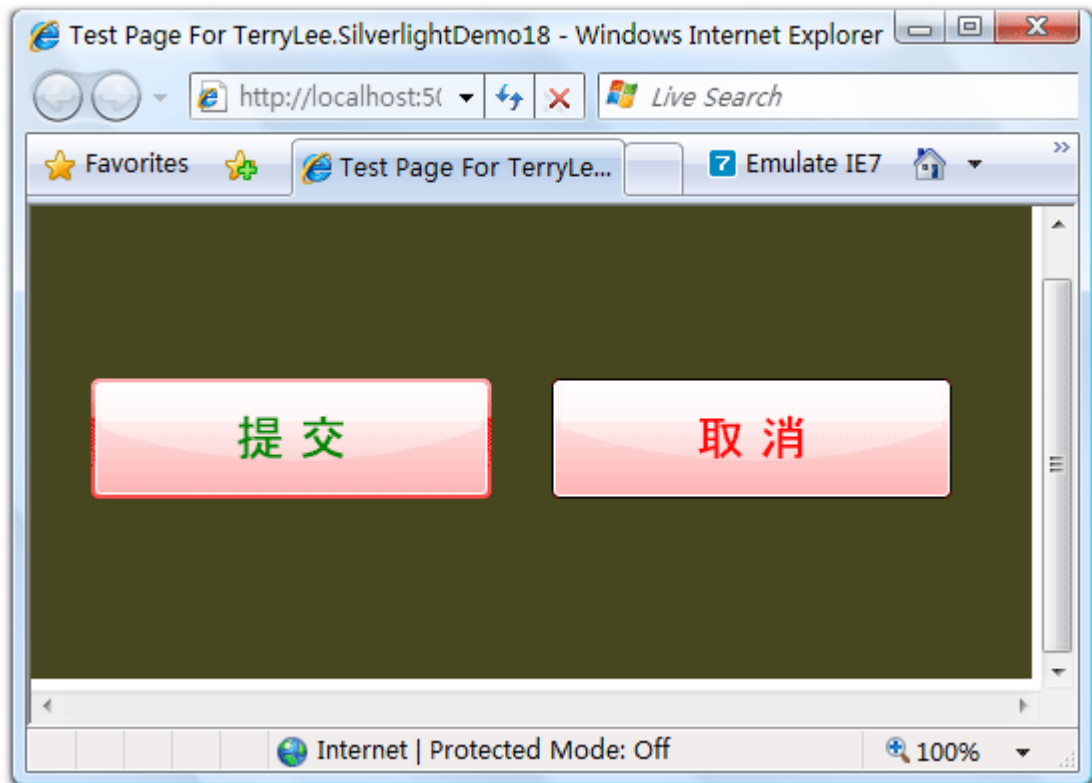
        Canvas.Top="90" Canvas.Left="260" Content="取消"

        Style="{StaticResource button2}"/>

</Canvas>

```

相比较上面的 XAML 文件,现在代码已经干净多了,这使得我们可以只专注于应用程序的业务,而无需考虑它的外观(在 Beta1 中似乎有些属性设置后会报错)。运行后效果如下:

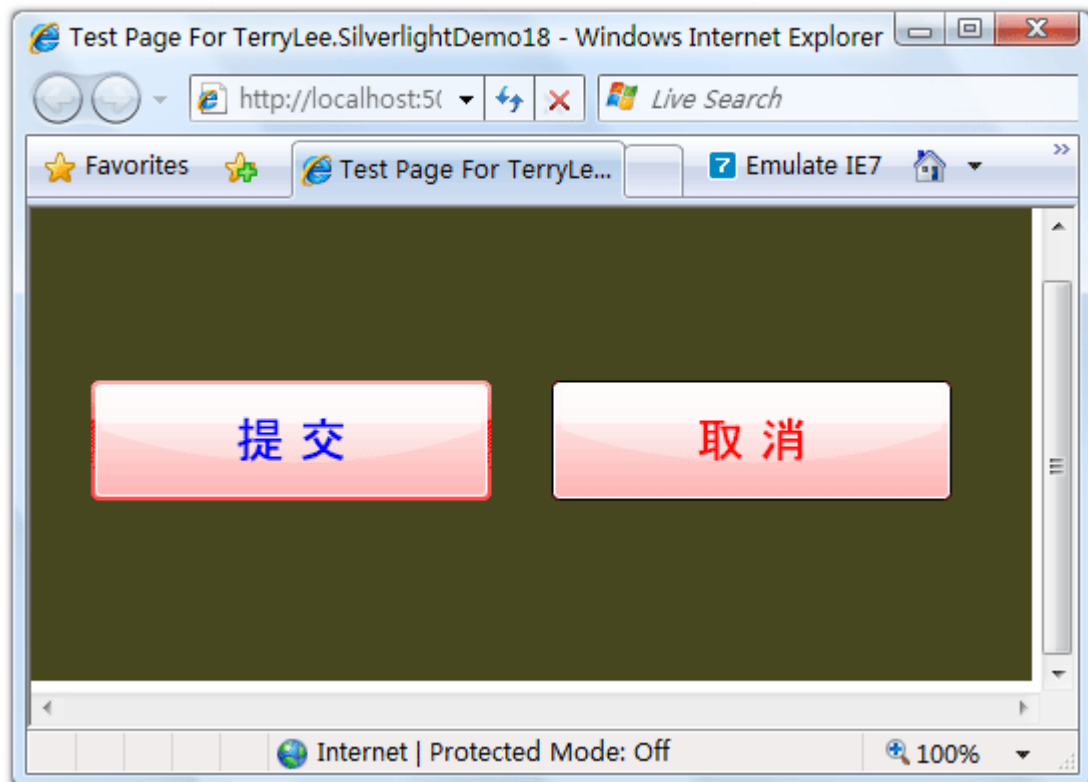


## 样式重写

定义了全局样式之后，样式能够被重写，即内联样式的优先级高于全局样式。如上面的示例中，我们在 XAML 中通过属性 `Foreground` 指定第一个按钮的前景色为蓝色：

```
<Canvas Background="#46461F">
    <Button Width="200" Height="60"
        Canvas.Top="90" Canvas.Left="30" Content="提交"
        Style="{StaticResource button1}"
        Foreground="Blue"
    />
    <Button Width="200" Height="60"
        Canvas.Top="90" Canvas.Left="260" Content="取消"
        Style="{StaticResource button2}"/>
</Canvas>
```

尽管我们在全局样式中指定第一个按钮的前景色为绿色，通过内联样式重写后，它显示为蓝色：



## 结束语

本文简单的介绍了 Silverlight 2 中使用样式来封装控件观感,对任何控件都可以使用全局样式进行封装。

## 一步一步学 Silverlight 2 系列（9）：使用控件模板

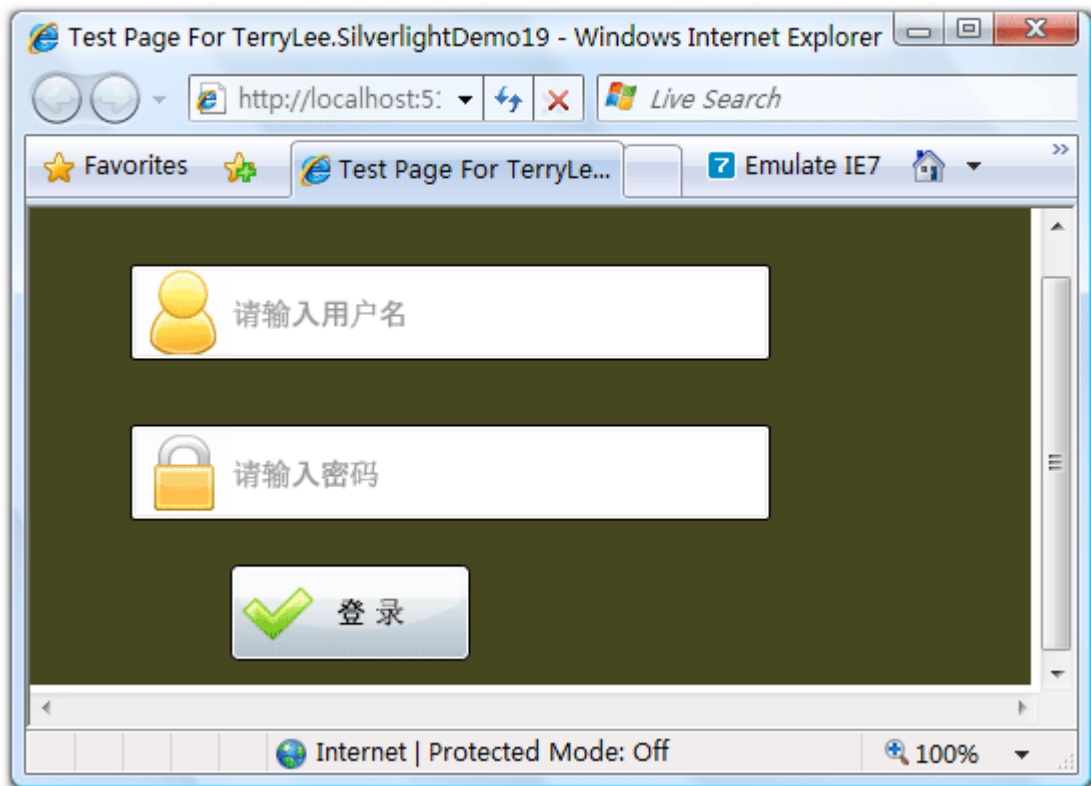
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第九篇，主要介绍如何使用控件模板定制控件的观感。Silverlight 提供了极其强大的功能，允许用户完全定制控件的外观。

### 定制控件内容

在 Silverlight 中，WatermarkedTextBox 控件可以为用户的输入提供一段提示信息，如果只是简单的一点文字信息，有时候未免显得单调，如果加上相应的图片说明效果会更好，如下图所示的一个简单的用户登录界面：



这样看起来界面显的就生动多了，XAML 声明如下：

```
<Canvas Background="#46461F">
    <WatermarkedTextBox x:Name="UserName" Canvas.Top="30" Canvas.Left="50"
        Width="320" Height="48">
        <WatermarkedTextBox.Watermark>
            <StackPanel Width="320" Height="48" Orientation="Horizontal">
                <Image Source="admin.png" HorizontalAlignment="Left"></Image>
                <TextBlock Text="请输入用户名" VerticalAlignment="Center" Foreground=
d="#999999"/>
            </StackPanel>
        </WatermarkedTextBox.Watermark>
    </WatermarkedTextBox>

    <WatermarkedTextBox x:Name="Password" Canvas.Top="110" Canvas.Left="50"
        Width="320" Height="48" HorizontalAlignment="Left">
        <WatermarkedTextBox.Watermark>
            <StackPanel Width="320" Height="48" Orientation="Horizontal">
                <Image Source="lock.png" HorizontalAlignment="Left"></Image>
                <TextBlock Text="请输入密码" VerticalAlignment="Center" Foreground=
"#999999"/>
            </StackPanel>
        </WatermarkedTextBox.Watermark>
    </WatermarkedTextBox>

    <Button Canvas.Top="180" Canvas.Left="100"
        Width="120" Height="48">
        <Button.Content>
            <StackPanel Orientation="Horizontal">
                <Image Source="apply.png" HorizontalAlignment="Left"></Image>
```



```
        <TextBlock Text="登录" VerticalAlignment="Center" Margin="10 0 0 0
    "></TextBlock>

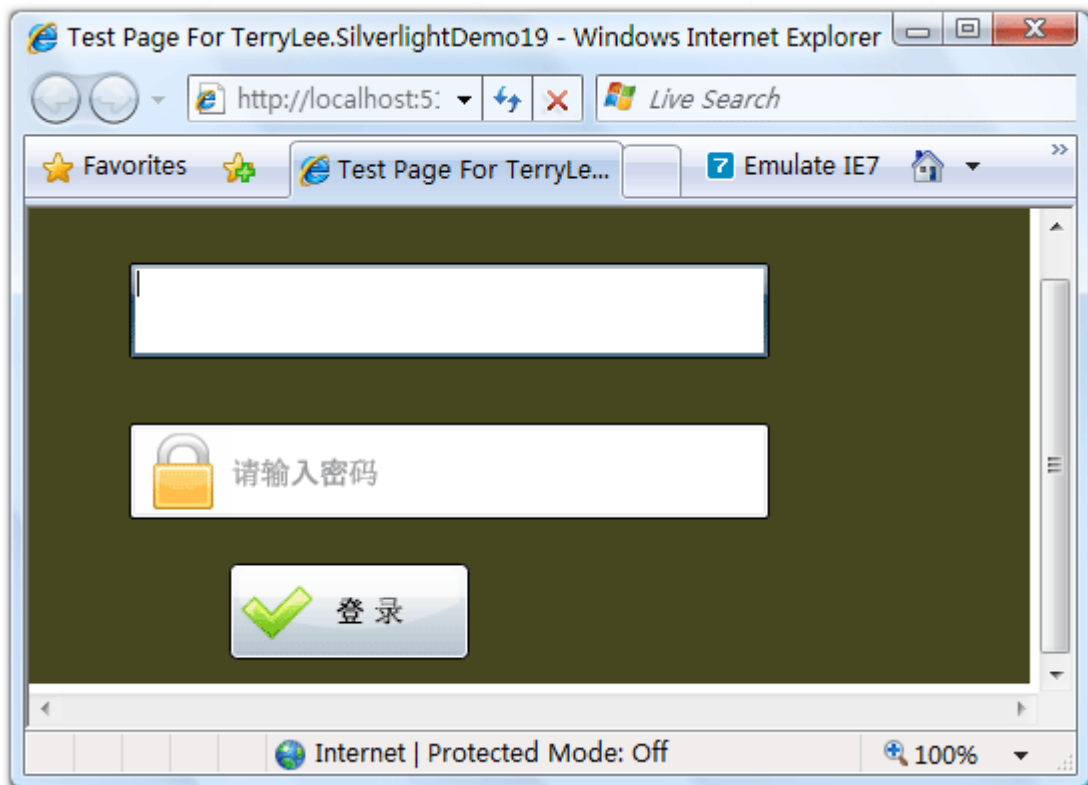
    </StackPanel>

    </Button.Content>

</Button>

</Canvas>
```

很多控件都有 Content 或者 Text 属性，我们完全可以充分发挥自己的想象力去进行定制，定制后控件仍然具有原来的功能行为，如上面的示例，当输入用户名控件获得焦点时文字和图片都将消失：



## 使用控件模板定制控件

前面的示例中我们只是定制了控件的内容，Silverlight 允许我们完全对控件进行定制，而不仅仅是内容。

下面的示例中我们定制一个渐变色的圆角矩形按钮。首先我们在 App.xaml 中创建一个 RoundButton 样式，改写按钮的 Template 属性：

```
<Style x:Key="RoundButton" TargetType="Button">

    <Setter Property="Template">
```

```

<Setter.Value>
  <ControlTemplate TargetType="Button">
    <Grid x:Name="RootElement">
      <Rectangle Width="200" Height="80" RadiusX="15" RadiusY="15">
        <Rectangle.Fill>
          <LinearGradientBrush StartPoint="0,0">
            <GradientStop Color="#FFFFFF" Offset="0.0" />
            <GradientStop Color="#EC04FA" Offset="1.0" />
          </LinearGradientBrush>
        </Rectangle.Fill>
        <Rectangle.Stroke>
          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
            <GradientStop Color="#FCB2FD" Offset="0" />
            <GradientStop Color="#FFFFFF" Offset="1" />
          </LinearGradientBrush>
        </Rectangle.Stroke>
      </Rectangle>
      <TextBlock Text="提交" FontSize="26" Foreground="White"
        HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Grid>
  </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

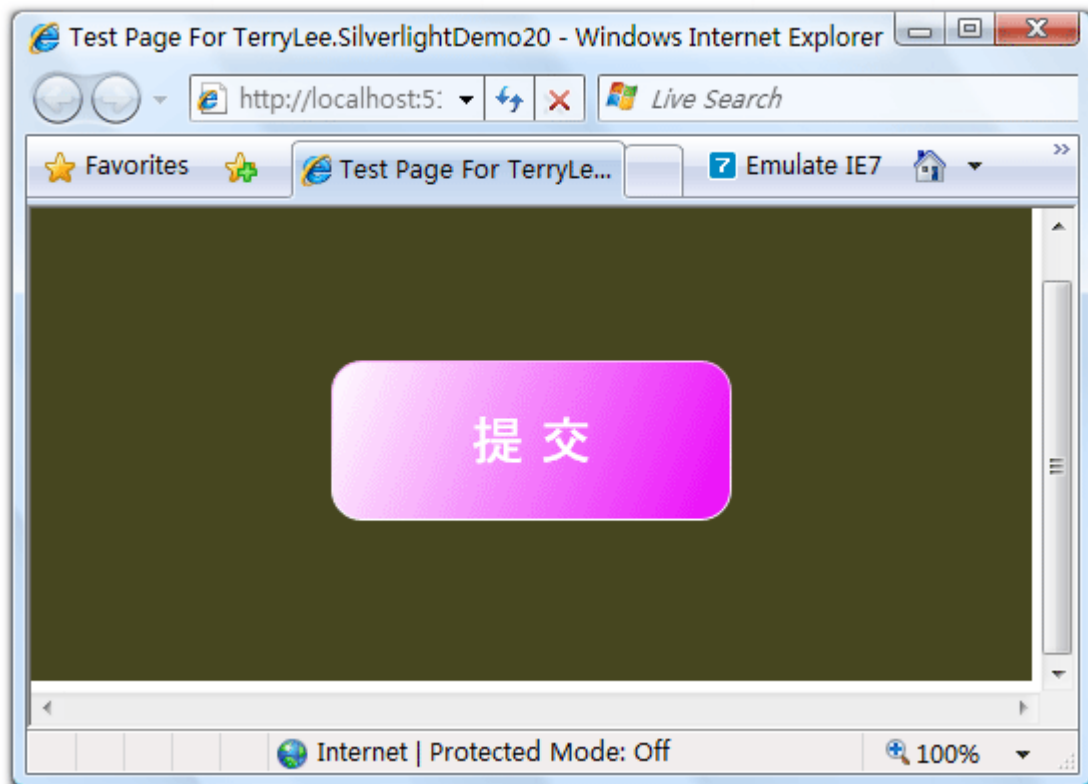
```

其中的渐变等内容在 Graphics 相关内容里将会写到。现在在 XAML 中使用该样式：

```
<Canvas Background="#46461F">
```

```
<Button x:Name="button1" Style="{StaticResource RoundButton}"
        Canvas.Top="80" Canvas.Left="150"/>
</Canvas>
```

运行后就可以看到下面的效果：



## 创建模板

上面的示例中，控件的文字以及控件的大小都是固定的，我们希望在开发人员使用中再设定，可以在控件模板中通过使用 {TemplateBinding ControlProperty} 的标识扩展句法来绑定到控件的属性来实现，使用 ContentPresenter 控件可以灵活的设置各个属性。修改 RoundButton 样式如下所示：

```
<Style x:Key="RoundButton" TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Grid x:Name="RootElement">
```

```

        <Rectangle Width="{TemplateBinding Width}" Height="{TemplateBinding Height}"
            RadiusX="15" RadiusY="15">
            <Rectangle.Fill>
                <LinearGradientBrush StartPoint="0,0">
                    <GradientStop Color="#FFFFFF" Offset="0.0" />
                    <GradientStop Color="#EC04FA" Offset="1.0" />
                </LinearGradientBrush>
            </Rectangle.Fill>
            <Rectangle.Stroke>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#EC04FA" Offset="0" />
                    <GradientStop Color="#FFFFFF" Offset="1" />
                </LinearGradientBrush>
            </Rectangle.Stroke>
        </Rectangle>
        <ContentPresenter
            Content="{TemplateBinding Content}"
            FontSize="{TemplateBinding FontSize}"
            HorizontalContentAlignment="{TemplateBinding HorizontalContentAlignment}"
            VerticalContentAlignment="{TemplateBinding VerticalContentAlignment}"
            Foreground="{TemplateBinding Foreground}">
        </ContentPresenter>
    </Grid>
</ControlTemplate>
</Setter.Value>

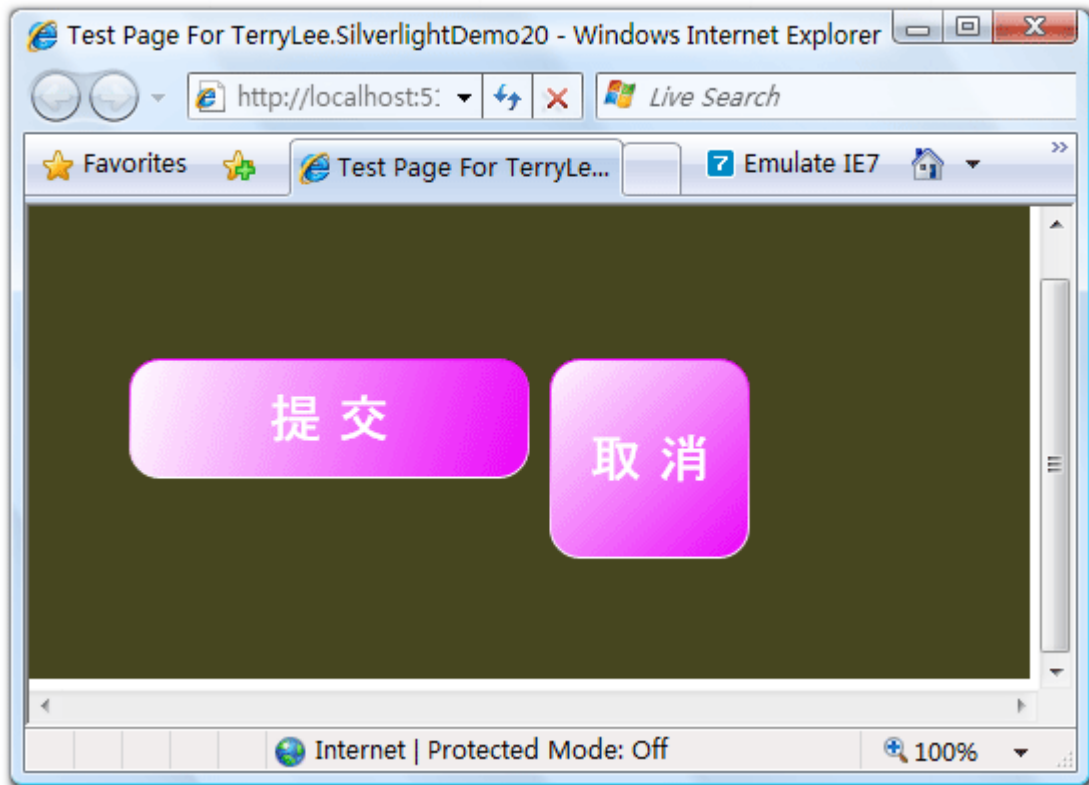
```

```
</Setter>  
</Style>
```

这样在使用 RoundButton 时我们可以设定控件的文本及控件的大小：

```
<Canvas Background="#46461F">  
    <Button x:Name="button1" Style="{StaticResource RoundButton}"  
        Canvas.Top="80" Canvas.Left="50"  
        Content="提 交" FontSize="26"  
        HorizontalContentAlignment="Center"  
        VerticalContentAlignment="Center"  
        Foreground="White" Width="200" Height="60"/>  
  
    <Button x:Name="button2" Style="{StaticResource RoundButton}"  
        Canvas.Top="80" Canvas.Left="260"  
        Content="取 消" FontSize="26"  
        HorizontalContentAlignment="Center"  
        VerticalContentAlignment="Center"  
        Foreground="White" Width="100" Height="100"/>  
</Canvas>
```

运行时效果如下：



## 结束语

本文简单的介绍了如何定制控件的内容以及通过控件模板完全定制控件，你可以从[这里](#)下载本文示例代码。

## 一步一步学 Silverlight 2 系列（10）：使用用户控件

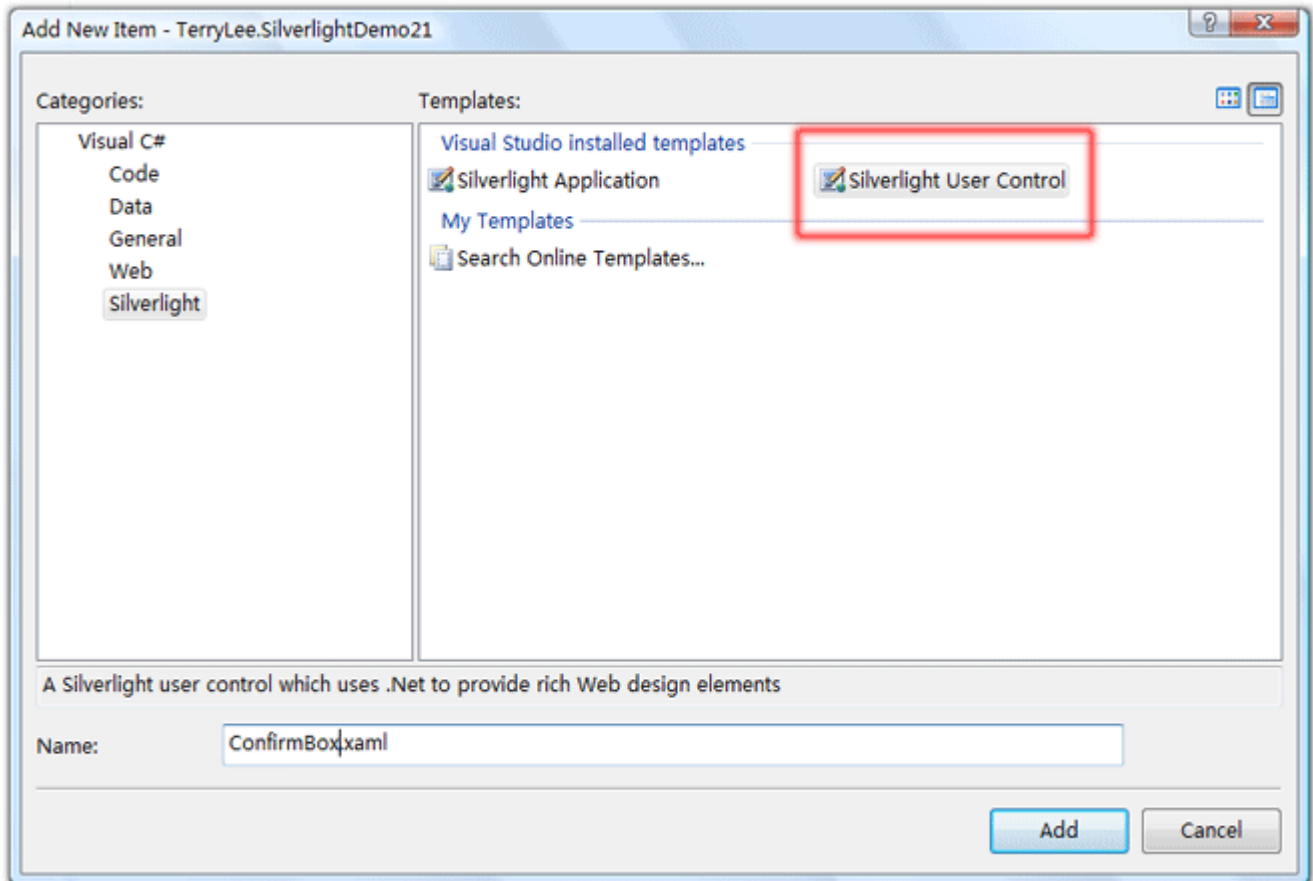
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了许多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第 10 篇，主要介绍 Silverlight 2 中的用户控件使用。

### 创建用户控件

在 Silverlight 2 中，我们可以根据开发自定义控件或者创建用户控件，以达到控件重用的目的，添加一个新的用户控件：



编写用户控件实现代码：

```
<Grid x:Name="LayoutRoot" Background="White">
```

```

<Rectangle HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
           Opacity="0.7" Fill="#FF8A8A8A"/>

<Border CornerRadius="15" Width="400" Height="150" Background="LightPink" Opacity="0.9">

    <StackPanel Orientation="Horizontal" Height="50">

        <Image Source="info.png" Margin="10 0 0 0"></Image>

        <Button Background="Red" Width="120" Height="40"
                Content="OK" Margin="10 0 0 0" FontSize="18"/>

        <Button Background="Red" Width="120" Height="40"
                Content="Cancel" Margin="50 0 0 0" FontSize="18"/>

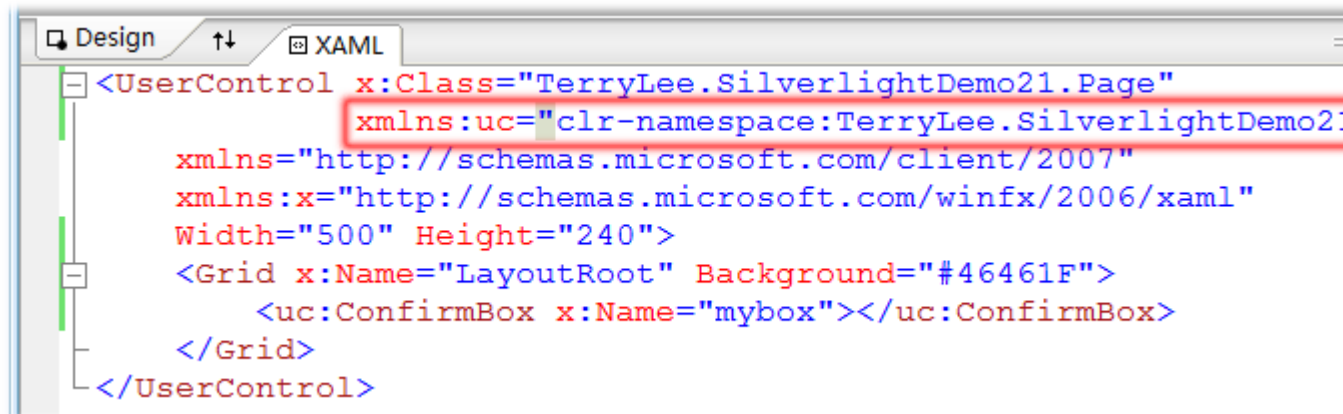
    </StackPanel>

</Border>

</Grid>

```

在需要使用该用户控件的页面 XAML 中注册命名空间：



使用用户控件：

```

<Grid x:Name="LayoutRoot" Background="#46461F">

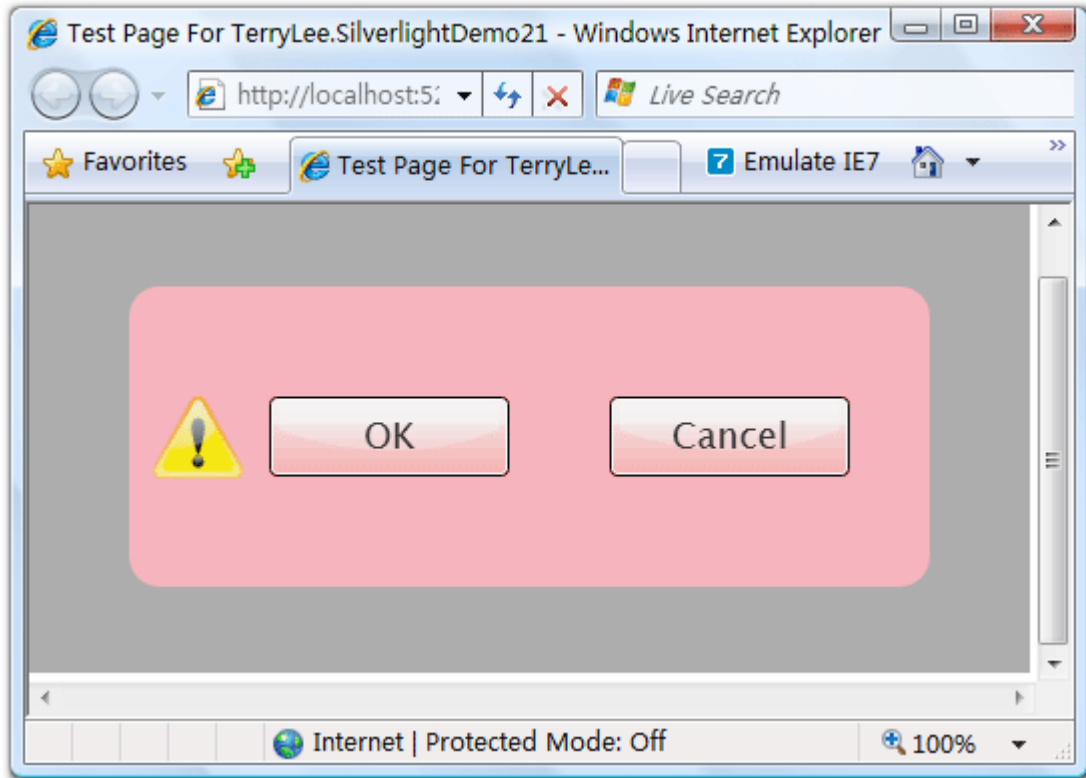
    <uc:ConfirmBox x:Name="mybox"></uc:ConfirmBox>

</Grid>

```

整个过程就这么简单，运行后效果如下：





## 为用户控件添加属性

简单的修改一下上面示例中的 XAML 文件，添加一个文本块控件，用它来显示文字提示信息。

```
<Grid x:Name="LayoutRoot" Background="White">
    <Rectangle HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
        Opacity="0.7" Fill="#FF8A8A"/>
    <Border CornerRadius="15" Width="400" Height="150" Background="LightPink" Opacity="0.9">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="60"></RowDefinition>
                <RowDefinition Height="90"></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition></ColumnDefinition>
            </Grid.ColumnDefinitions>
        </Grid>
    </Border>
</Grid>
```

```

        <TextBlock x:Name="message" FontSize="18" Foreground="White"
                HorizontalAlignment="Left" VerticalAlignment="Center"
                Margin="50 20 0 0"/>

        <StackPanel Orientation="Horizontal" Height="50" Grid.Row="1">
            <Image Source="info.png" Margin="10 0 0 0"></Image>
            <Button Background="Red" Width="120" Height="40"
                    Content="OK" Margin="10 0 0 0" FontSize="18"/>
            <Button Background="Red" Width="120" Height="40"
                    Content="Cancel" Margin="50 0 0 0" FontSize="18"/>
        </StackPanel>
    </Grid>
</Border>
</Grid>

```

定义属性:

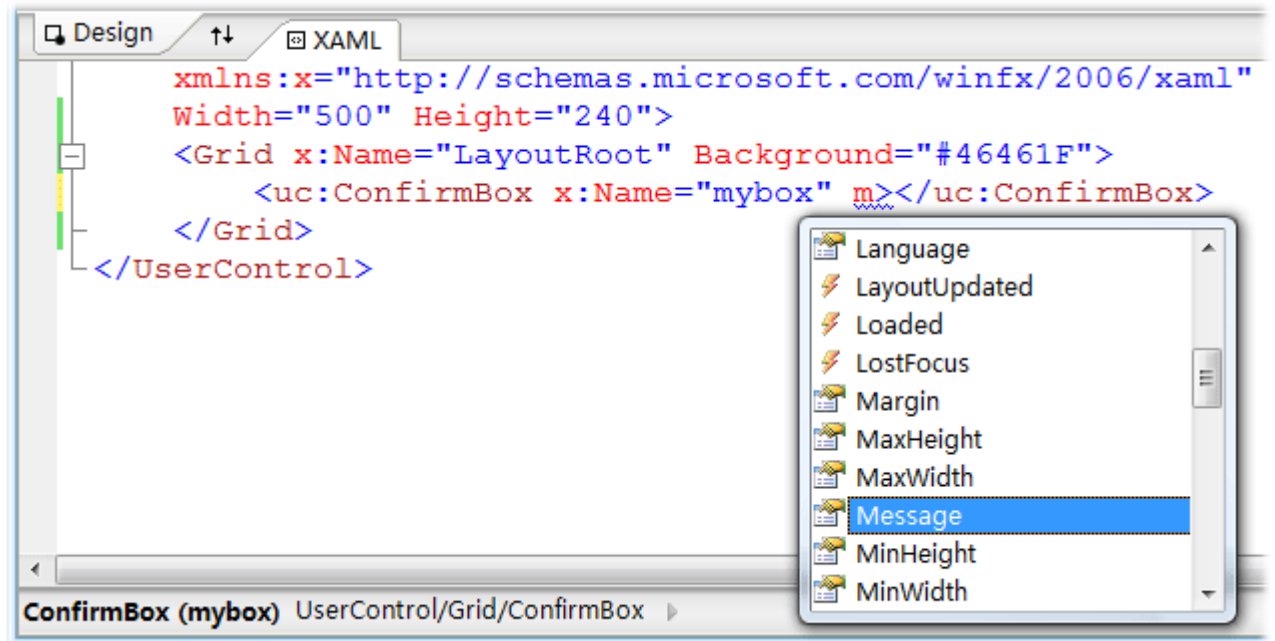
```

public partial class ConfirmBox : UserControl
{
    public ConfirmBox()
    {
        InitializeComponent();
    }

    public String Message
    {
        get { return this.message.Text; }
        set { this.message.Text = value; }
    }
}

```

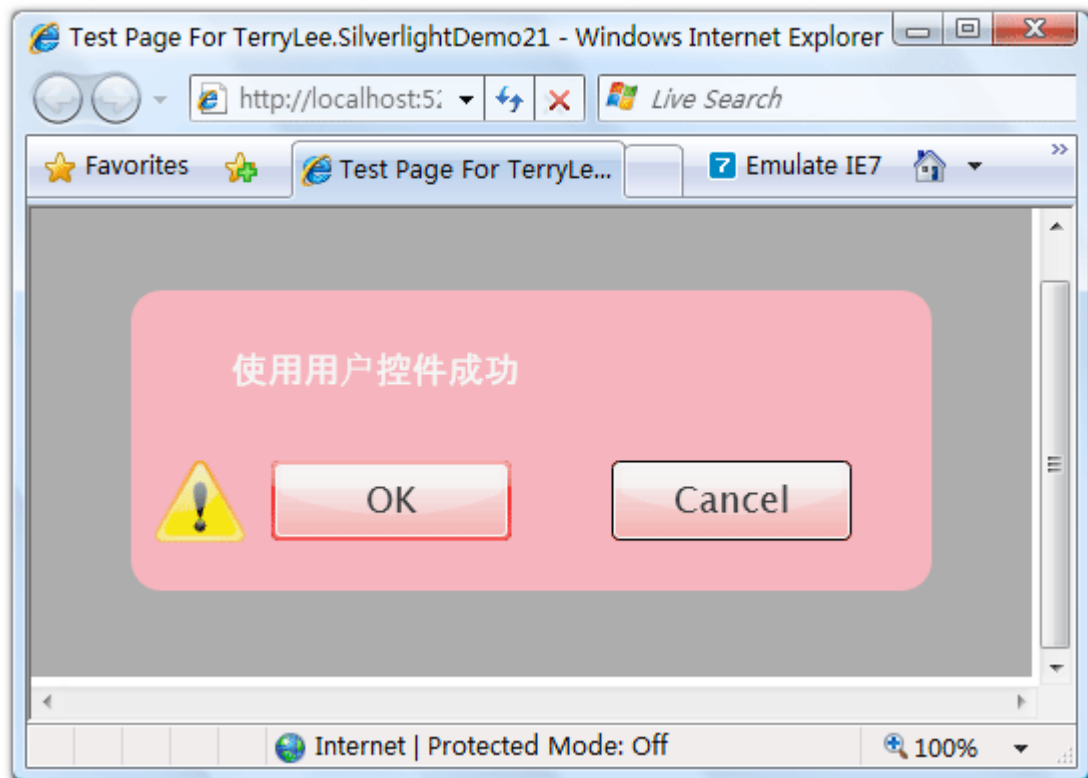
在页面使用用户控件的属性，XAML 编辑器能够识别出属性并提示：



为 ConfirmBox 控件的 Message 属性赋值：

```
<Grid x:Name="LayoutRoot" Background="#46461F">
    <uc:ConfirmBox x:Name="mybox" Message="使用用户控件成功"></uc:ConfirmBox>
</Grid>
```

运行后效果如下所示：



## 动态添加用户控件

用户控件可以动态的添加到页面中，修改一下 Page.xaml 中的 XAML 代码，放入一个 Canvas 作为用户控件的容器。

```
<Grid x:Name="LayoutRoot" Background="#46461F">
    <Canvas x:Name="ContainerCanvas">

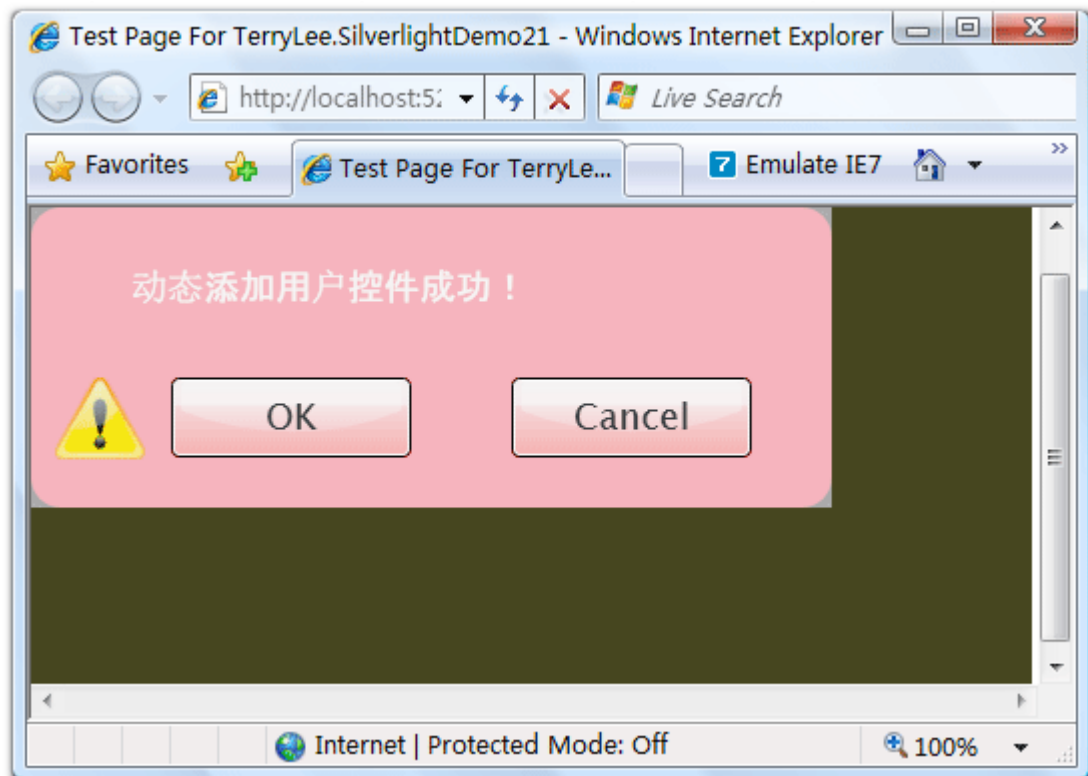
    </Canvas>
</Grid>
```

编写添加用户控件代码：

```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
    ConfirmBox confirmbox = new ConfirmBox();
```

```
confirmbox.Message = "动态添加用户控件成功!";  
  
ContainerCanvas.Children.Add(confirmbox);  
}
```

运行后效果如下所示，当然我们也可以控制用户控件显示的位置等。



## 结束语

本文简单介绍了在 Silverlight 2 中使用用户控件，包括创建用户控件、添加属性、动态添加用户控件等内容，你可以从[这里](#)下载本文示例代码。

## 一步一步学 Silverlight 2 系列（11）：数据绑定

### 概念

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文为系列文章第十一篇，主要介绍 Silverlight 2 中的数据绑定。

### 数据绑定模式

在 Silverlight 2 中，支持三种模式的数据绑定。

- 1.OneTime: 一次绑定，在绑定创建时使用源数据更新目标，适用于只显示数据而不进行数据的更新。
- 2.OneWay: 单向绑定，在绑定创建时或者源数据发生变化时更新到目标，适用于显示变化的数据。
- 3.TwoWay: 双向绑定，在任何时候都可以同时更新源数据和目标。

Jesse Liberty 举的例子非常的形象，使用 Silverlight 开发一个在线书店，显示书籍的书名、作者等信息，使用 OneTime 模式，这些数据一般不会发生变化的；显示价格信息时使用 OneWay 模式，因为管理员可能会在一天内调整价格；显示书籍的剩余数量时用 TwoWay 模式，数量随着用户的订购会随时发生变化，即目标和源数据都要进行更新。

### 简单数据绑定

在本示例中我们将做一个简单的数据绑定，用来显示用户信息，XAML 如下：

```
<Grid x:Name="LayoutRoot" Background="#46461F">
    <Grid.RowDefinitions>
        <RowDefinition Height="160"></RowDefinition>
        <RowDefinition Height="40"></RowDefinition>
        <RowDefinition Height="40"></RowDefinition>
    </Grid.RowDefinitions>
```

```

<Grid.ColumnDefinitions>
    <ColumnDefinition Width="150"></ColumnDefinition>
    <ColumnDefinition Width="*"></ColumnDefinition>
</Grid.ColumnDefinitions>
<Image Source="terrylee.jpg" Width="78" Height="100"
    HorizontalAlignment="Left" Grid.Row="0" Grid.Column="1"/>
<TextBlock Foreground="White" FontSize="18" Text="姓名: "
    Grid.Row="1" Grid.Column="0" HorizontalAlignment="Right"/>
<TextBlock x:Name="lblName" Foreground="White" FontSize="18"
    Grid.Row="1" Grid.Column="1" HorizontalAlignment="Left"/>
<TextBlock Foreground="White" FontSize="18" Text="位置: "
    Grid.Row="2" Grid.Column="0" HorizontalAlignment="Right"/>
<TextBlock x:Name="lblAddress" Foreground="White" FontSize="18"
    Grid.Row="2" Grid.Column="1" HorizontalAlignment="Left"/>
</Grid>

```

添加一个简单 User 类，它具有 Name 和 Address 两个属性：

```

public class User
{
    public string Name { get; set; }

    public string Address { get; set; }
}

```

使用绑定句法{Binding Property}进行数据绑定，注意下面的两个 TextBlock 控件 Text 属性：

```

<Grid x:Name="LayoutRoot" Background="#46461F">
    <Grid.RowDefinitions>
        <RowDefinition Height="160"></RowDefinition>

```

```

        <RowDefinition Height="40"></RowDefinition>
        <RowDefinition Height="40"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="150"></ColumnDefinition>
        <ColumnDefinition Width="*"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Image Source="terrylee.jpg" Width="78" Height="100"
        HorizontalAlignment="Left" Grid.Row="0" Grid.Column="1"/>
    <TextBlock Foreground="White" FontSize="18" Text="姓名: "
        Grid.Row="1" Grid.Column="0" HorizontalAlignment="Right"/>
    <TextBlock x:Name="lblName" Foreground="White" FontSize="18"
        Grid.Row="1" Grid.Column="1" HorizontalAlignment="Left"
        Text="{Binding Name}"/>
    <TextBlock Foreground="White" FontSize="18" Text="位置: "
        Grid.Row="2" Grid.Column="0" HorizontalAlignment="Right"/>
    <TextBlock x:Name="lblAddress" Foreground="White" FontSize="18"
        Grid.Row="2" Grid.Column="1" HorizontalAlignment="Left"
        Text="{Binding Address}"/>
</Grid>

```

指定数据源，注意这里是创建一个 User 的实例并赋值后，把 user 实例绑定到了 TextBlock 的 DataContext 上，而不是向之前我们所做的示例中那样，直接指定 Text 属性：

```

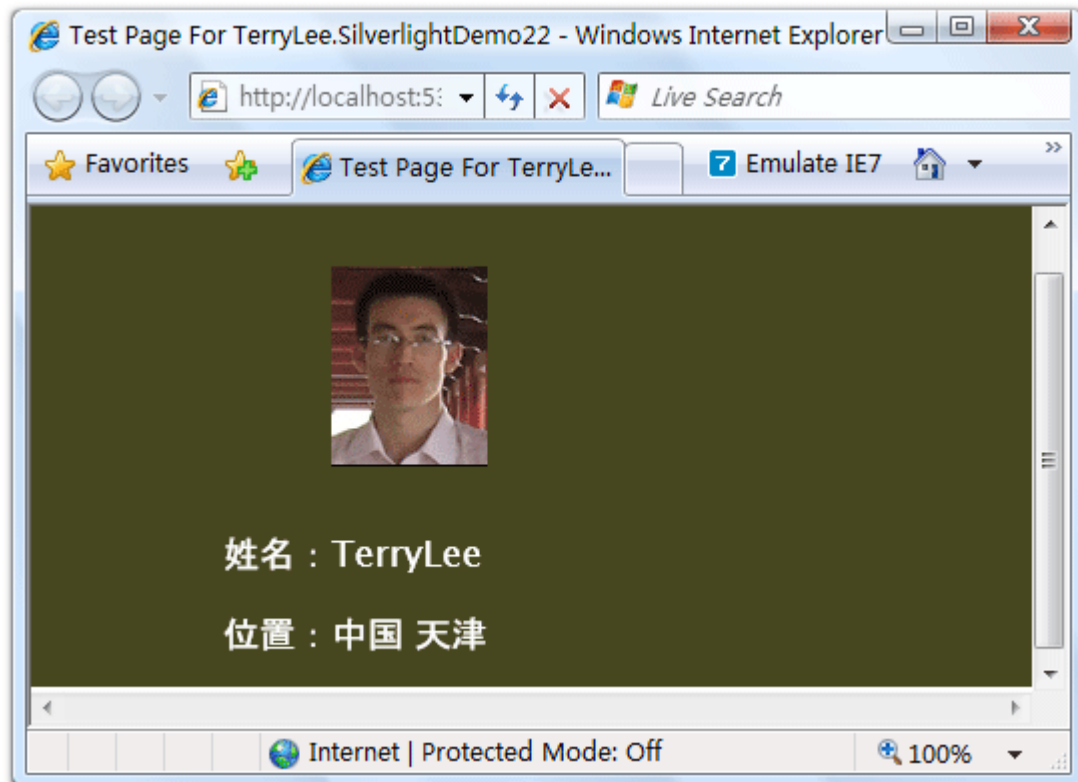
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
    User user = new User();
    user.Name = "TerryLee";
    user.Address = "中国 天津";
}

```



```
lblName.DataContext = user;  
  
lblAddress.DataContext = user;  
  
}
```

运行示例后，可以看到：



上面这种数据绑定模式，只是显示数据而不对数据做任何修改，默认的绑定模式是一次绑定 OneTime。

## 单向绑定示例

如果需要在数据源发生变化时能够通知 UI 进行相应的更新，即使用单向绑定 OneWay 或者双向绑定 TwoWay，则业务实体需要实现接口 INotifyPropertyChanged。在本示例中，我们加上一个更新按钮，当单击按钮时更新 user 实例的属性值，会看到界面上的数据也会发生变化。

修改一下 User 类，使其实现 INotifyPropertyChanged 接口。

```
public class User : INotifyPropertyChanged  
{
```

```
public event PropertyChangedEventHandler PropertyChanged;

private string _name;

public string Name
{
    get { return _name; }
    set
    {
        _name = value;
        if(PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs("Name"));
        }
    }
}

private string _address;

public string Address
{
    get { return _address; }
    set
    {
        _address = value;
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs("Address"));
        }
    }
}
```

```
}  
  
}
```

修改数据绑定模式，使用单向绑定 OneWay 模式，如{ Binding Address, Mode=OneWay}

```
<Grid x:Name="LayoutRoot" Background="#46461F">  
    <Grid.RowDefinitions>  
        <RowDefinition Height="160"></RowDefinition>  
        <RowDefinition Height="40"></RowDefinition>  
        <RowDefinition Height="40"></RowDefinition>  
    </Grid.RowDefinitions>  
    <Grid.ColumnDefinitions>  
        <ColumnDefinition Width="150"></ColumnDefinition>  
        <ColumnDefinition Width="*"></ColumnDefinition>  
    </Grid.ColumnDefinitions>  
    <Image Source="terrylee.jpg" Width="78" Height="100"  
        HorizontalAlignment="Left" Grid.Row="0" Grid.Column="1"/>  
    <Button x:Name="btnUpdate" Width="100" Height="40"  
        Content="Update" Click="btnUpdate_Click"/>  
    <TextBlock Foreground="White" FontSize="18" Text="姓名: "  
        Grid.Row="1" Grid.Column="0" HorizontalAlignment="Right"/>  
    <TextBlock x:Name="lblName" Foreground="White" FontSize="18"  
        Grid.Row="1" Grid.Column="1" HorizontalAlignment="Left"  
        Text="{Binding Name, Mode=OneWay}"/>  
    <TextBlock Foreground="White" FontSize="18" Text="位置: "  
        Grid.Row="2" Grid.Column="0" HorizontalAlignment="Right"/>  
    <TextBlock x:Name="lblAddress" Foreground="White" FontSize="18"  
        Grid.Row="2" Grid.Column="1" HorizontalAlignment="Left"  
        Text="{Binding Address, Mode=OneWay}"/>
```

```
</Grid>
```

编写事件处理程序，为了演示把 user 声明为一个全局的，并在按钮的单击事件中修改其属性值：

```
public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
    }

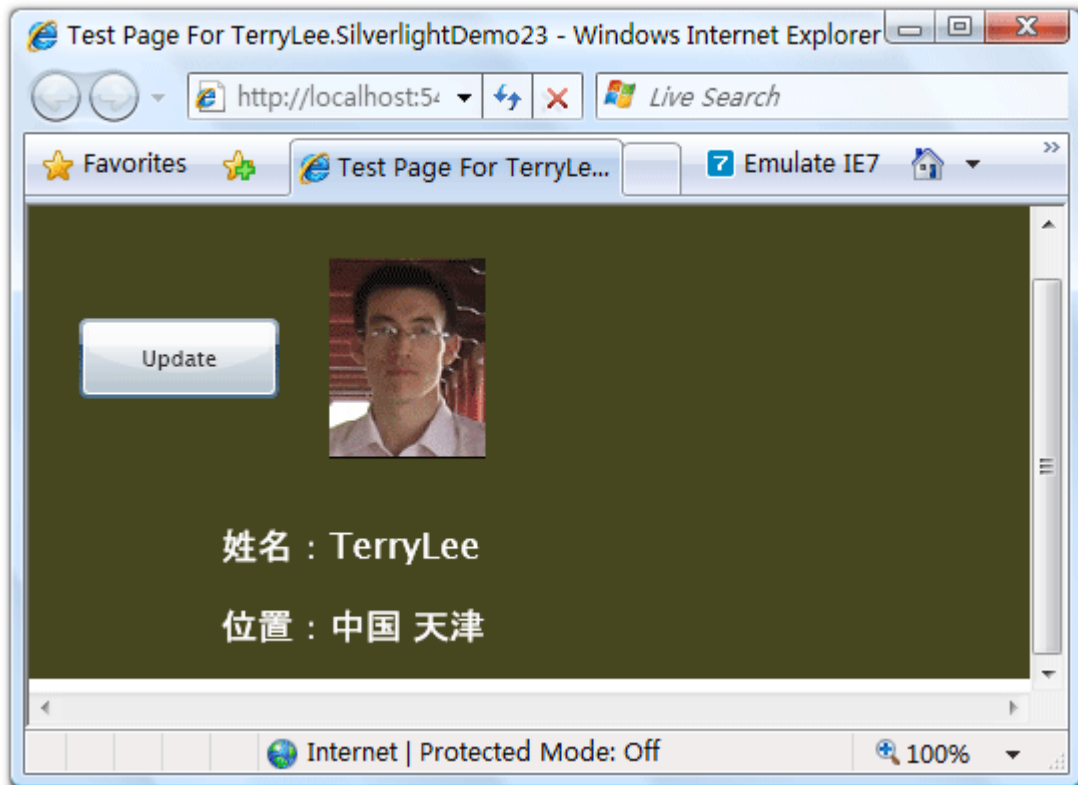
    User user;

    private void UserControl_Loaded(object sender, RoutedEventArgs e)
    {
        user = new User();
        user.Name = "TerryLee";
        user.Address = "中国 天津";

        lblName.DataContext = user;
        lblAddress.DataContext = user;
    }

    private void btnUpdate_Click(object sender, RoutedEventArgs e)
    {
        user.Name = "李会军";
        user.Address = "China Tianjin";
    }
}
```

运行后如下所示：



单击 Update 按钮后:



## 绑定到列表

下面再看一个绑定到列表的简单例子，一般都会使用 DataGrid 或者 ListBox 来进行列表数据的显示。下面的示例我们显示一个文章列表：

```
<Grid Background="#46461F">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Border Grid.Row="0" Grid.Column="0" CornerRadius="15"
        Width="240" Height="36" Background="Orange"
        Margin="20 0 0 0" HorizontalAlignment="Left">
        <TextBlock Text="文章列表" Foreground="White"
            HorizontalAlignment="Left" VerticalAlignment="Center"
            Margin="20 0 0 0"></TextBlock>
    </Border>
    <ListBox x:Name="PostList" Grid.Column="0" Grid.Row="1"
        Margin="40 10 10 10"
        HorizontalContentAlignment="Left" VerticalContentAlignment="Bottom"
        ItemsSource="{Binding Posts}">
    </ListBox>
</Grid>
```

编写一个简单的业务类：

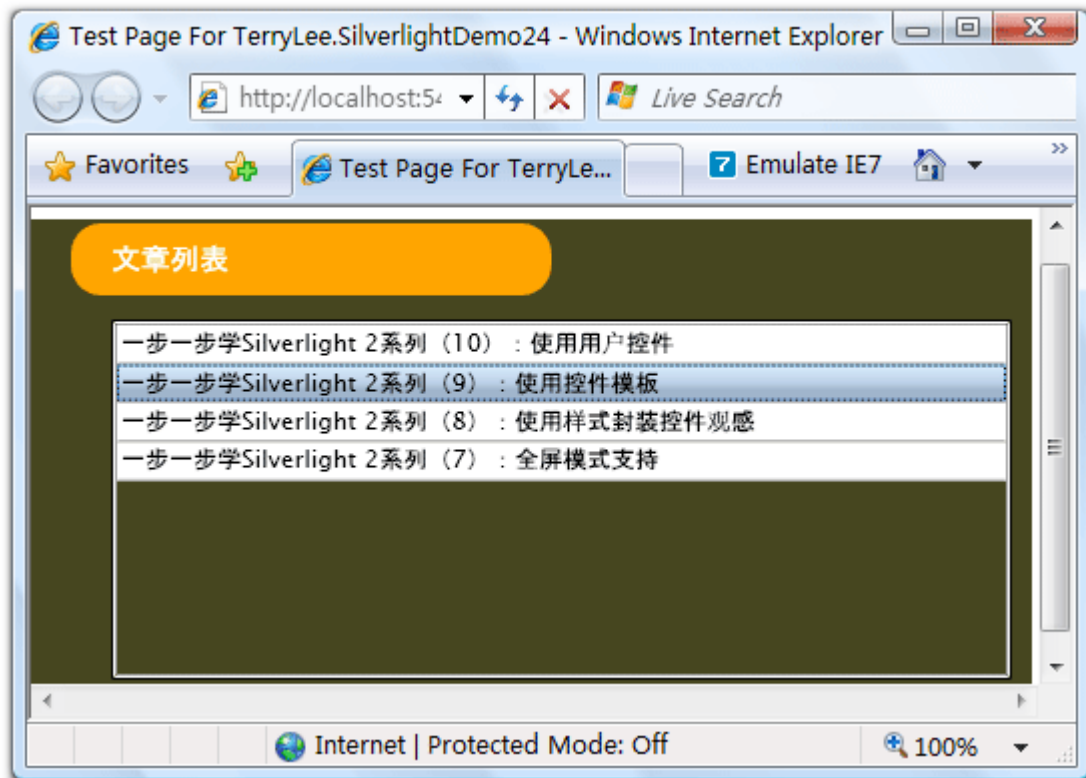
```
public class Blog
{
```

```
public List<String> Posts { get; set; }  
}
```

初始化集合数据并进行绑定

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)  
{  
    Blog blog = new Blog();  
    blog.Posts = new List<String>  
    {  
        "一步一步学 Silverlight 2 系列 (10) : 使用用户控件",  
        "一步一步学 Silverlight 2 系列 (9) : 使用控件模板",  
        "一步一步学 Silverlight 2 系列 (8) : 使用样式封装控件观感",  
        "一步一步学 Silverlight 2 系列 (7) : 全屏模式支持"  
    };  
  
    PostList.DataContext = blog;  
}
```

最终运行的结果如下所示:



当然我们也可以使用 `ListBox` 的 `ItemsSource` 属性进行绑定，

## 结束语

本文简单介绍了 Silverlight 2 中的数据绑定，你可以从[这里](#)下载文章示例代码。



## 一步一步学 Silverlight 2 系列（12）：数据与通信之 WebClient

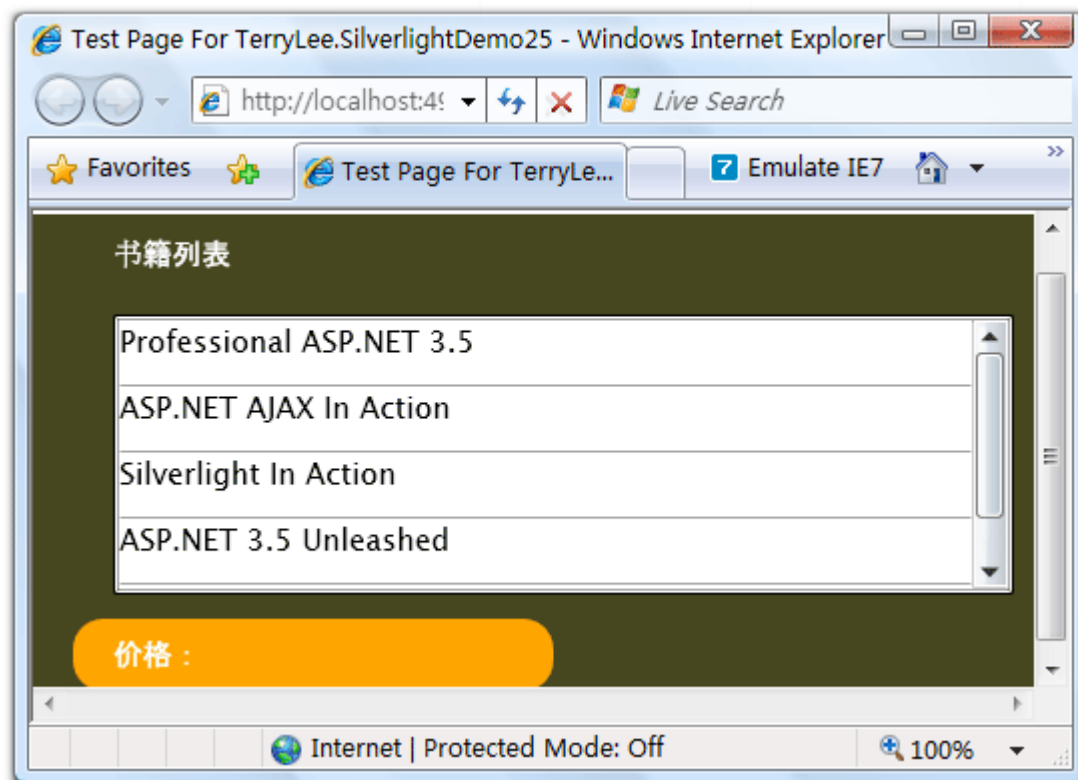
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文将介绍如何在 Silverlight 2 中使用 Web Client 进行通信。

### 简单示例

编写一个简单的示例，在该示例中，选择一本书籍之后，我们通过 Web Client 去查询书籍的价格，并显示出来，最终的效果如下：



编写界面布局，XAML 如下：

```
<Grid Background="#46461F">  
    <Grid.RowDefinitions>
```

```

    <RowDefinition Height="40"></RowDefinition>

    <RowDefinition Height="*"></RowDefinition>

    <RowDefinition Height="40"></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

    <ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<Border Grid.Row="0" Grid.Column="0" CornerRadius="15"

    Width="240" Height="36"

    Margin="20 0 0 0" HorizontalAlignment="Left">

    <TextBlock Text="书籍列表" Foreground="White"

        HorizontalAlignment="Left" VerticalAlignment="Center"

        Margin="20 0 0 0"></TextBlock>

</Border>

<ListBox x:Name="Books" Grid.Row="1" Margin="40 10 10 10"

    SelectionChanged="Books_SelectionChanged">

    <ListBox.ItemTemplate>

        <DataTemplate>

            <StackPanel>

                <TextBlock Text="{Binding Name}" Height="32"></TextBlock>

            </StackPanel>

        </DataTemplate>

    </ListBox.ItemTemplate>

</ListBox>

<Border Grid.Row="2" Grid.Column="0" CornerRadius="15"

    Width="240" Height="36" Background="Orange"

    Margin="20 0 0 0" HorizontalAlignment="Left">

    <TextBlock x:Name="lblPrice" Text="价格: " Foreground="White"

```

```
HorizontalAlignment="Left" VerticalAlignment="Center"  
Margin="20 0 0 0"></TextBlock>  
  
</Border>  
  
</Grid>
```

为了模拟查询价格，我们编写一个 `HttpHandler`，接收书籍的 No，并返回价格：

```
public class BookHandler : IHttpHandler  
{  
    public static readonly string[] PriceList = new string[] {  
        "66.00",  
        "78.30",  
        "56.50",  
        "28.80",  
        "77.00"  
    };  
    public void ProcessRequest(HttpContext context)  
    {  
        context.Response.ContentType = "text/plain";  
        context.Response.Write(PriceList[Int32.Parse(context.Request.QueryString["No"])]);  
    }  
  
    public bool IsReusable  
    {  
        get  
        {  
            return false;  
        }  
    }  
}
```

```
}  
  
}
```

在界面加载时绑定书籍列表，关于数据绑定可以参考[一步一步学 Silverlight 2 系列（11）：数据绑定](#)。

```
void UserControl_Loaded(object sender, RoutedEventArgs e)  
{  
  
    List<Book> books = new List<Book>() {  
        new Book("Professional ASP.NET 3.5"),  
        new Book("ASP.NET AJAX In Action"),  
        new Book("Silverlight In Action"),  
        new Book("ASP.NET 3.5 Unleashed"),  
        new Book("Introducing Microsoft ASP.NET AJAX")  
    };  
  
    Books.ItemsSource = books;  
  
}
```

接下来当用户选择一本书籍时，需要通过 Web Client 去获取书籍的价格，在 Silverlight 2 中，所有的网络通信 API 都设计为了异步模式。在声明一个 Web Client 实例后，我们需要为它注册 DownloadStringCompleted 事件处理方法，在下载完成后将会被回调，然后再调用 DownloadStringAsync 方法开始下载。

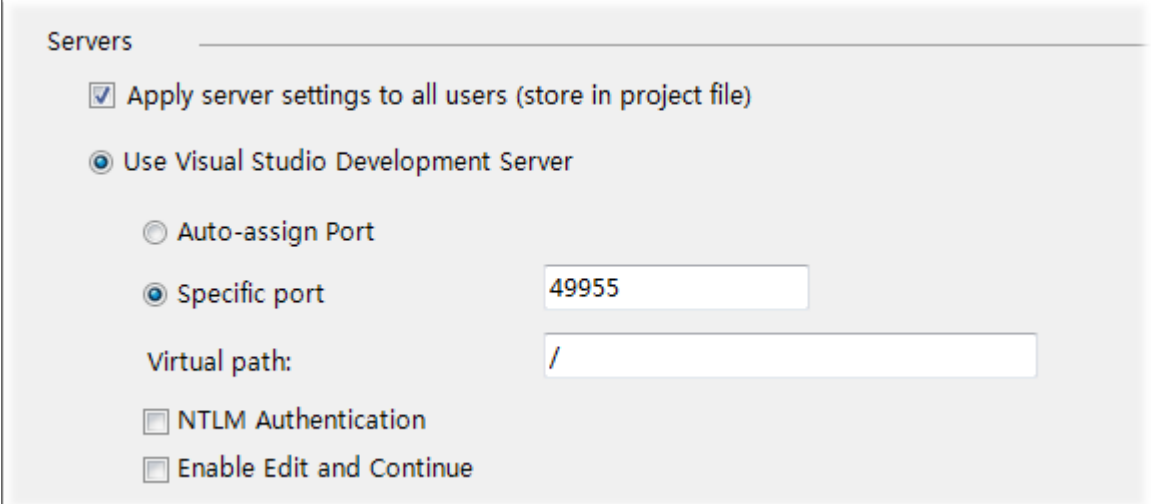
```
void Books_SelectionChanged(object sender, SelectionChangedEventArgs e)  
{  
  
    Uri endpoint = new Uri(String.Format("http://localhost:49955/BookHandler.ashx?No={0}", Books.SelectedIndex));  
  
    WebClient client = new WebClient();
```

```
        client.DownloadStringCompleted += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);

        client.DownloadStringAsync(endpoint);
    }

    void client_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
    {
        if (e.Error == null)
        {
            lblPrice.Text = "价格: " + e.Result;
        }
        else
        {
            lblPrice.Text = e.Error.Message;
        }
    }
}
```

注意大家可以在 Web Application Project 的属性页中, 把 ASP.NET Development Server 的端口号设置为一个固定的端口号:



The screenshot shows the 'Servers' configuration window in Visual Studio. It is titled 'Servers' and has a sub-header 'Servers'. The window contains several options:

- Apply server settings to all users (store in project file)
- Use Visual Studio Development Server
  - Auto-assign Port
  - Specific port: 49955
  - Virtual path: /
- NTLM Authentication
- Enable Edit and Continue

最后完整的代码如下：

```
public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
    }

    void UserControl_Loaded(object sender, RoutedEventArgs e)
    {
        List<Book> books = new List<Book>() {
            new Book("Professional ASP.NET 3.5"),
            new Book("ASP.NET AJAX In Action"),
            new Book("Silverlight In Action"),
            new Book("ASP.NET 3.5 Unleashed"),
            new Book("Introducing Microsoft ASP.NET AJAX")
        };

        Books.ItemsSource = books;
    }

    void Books_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        Uri endpoint = new Uri(String.Format("http://localhost:49955/BookHandler.ashx?No={0}", Books.SelectedIndex));

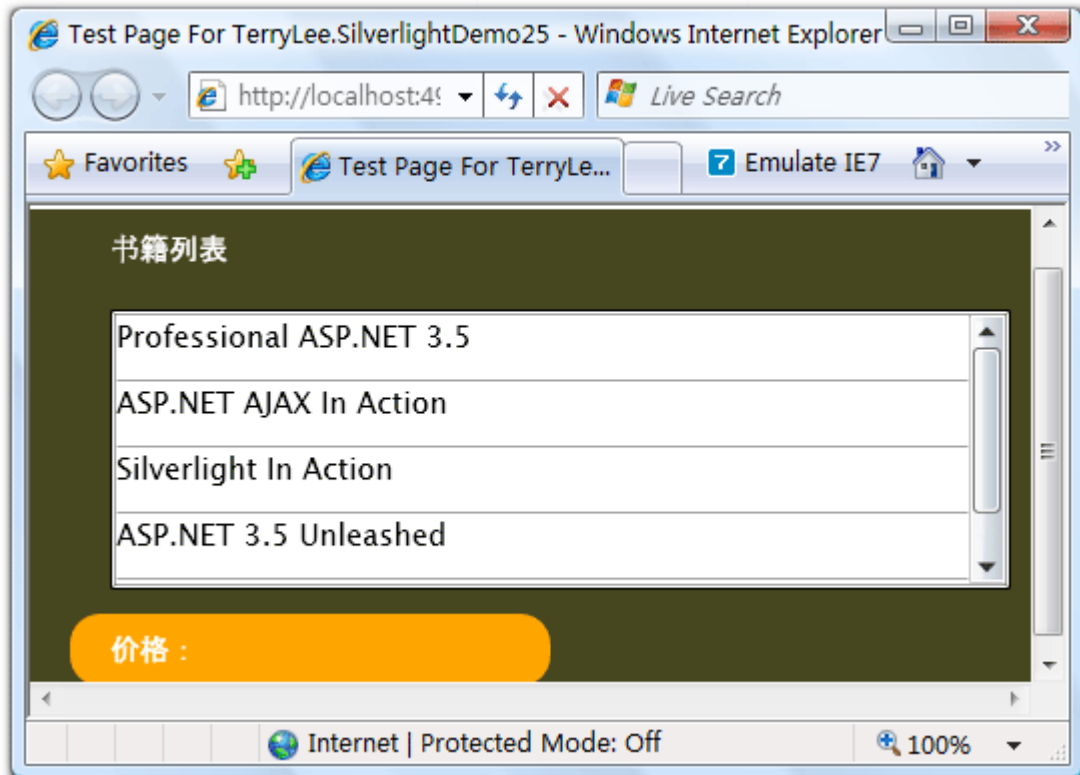
        WebClient client = new WebClient();
    }
}
```

```
        client.DownloadStringCompleted += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);

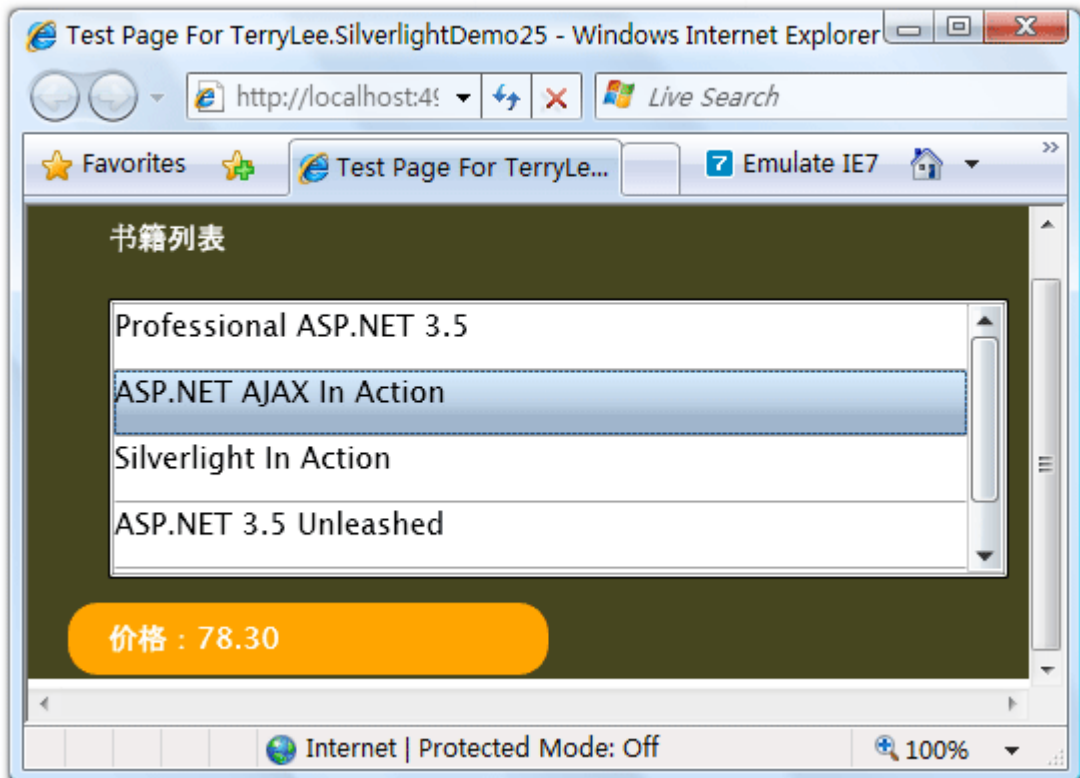
        client.DownloadStringAsync(endpoint);
    }

    void client_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
    {
        if (e.Error == null)
        {
            lblPrice.Text = "价格: " + e.Result;
        }
        else
        {
            lblPrice.Text = e.Error.Message;
        }
    }
}
```

运行后效果如下:



当我们选择其中一本书籍时，将会显示出它的价格：





## 结束语

本文简单介绍了 Silverlight 2 中使用 Web Client 进行通信的知识，在 Silverlight 2 中，提供的通信 API 非常丰富，后面将会介绍其他的方式。你可以从[这里](#)下载本文示例代码。

## 一步一步学 Silverlight 2 系列（13）：数据与通信之 WebRequest

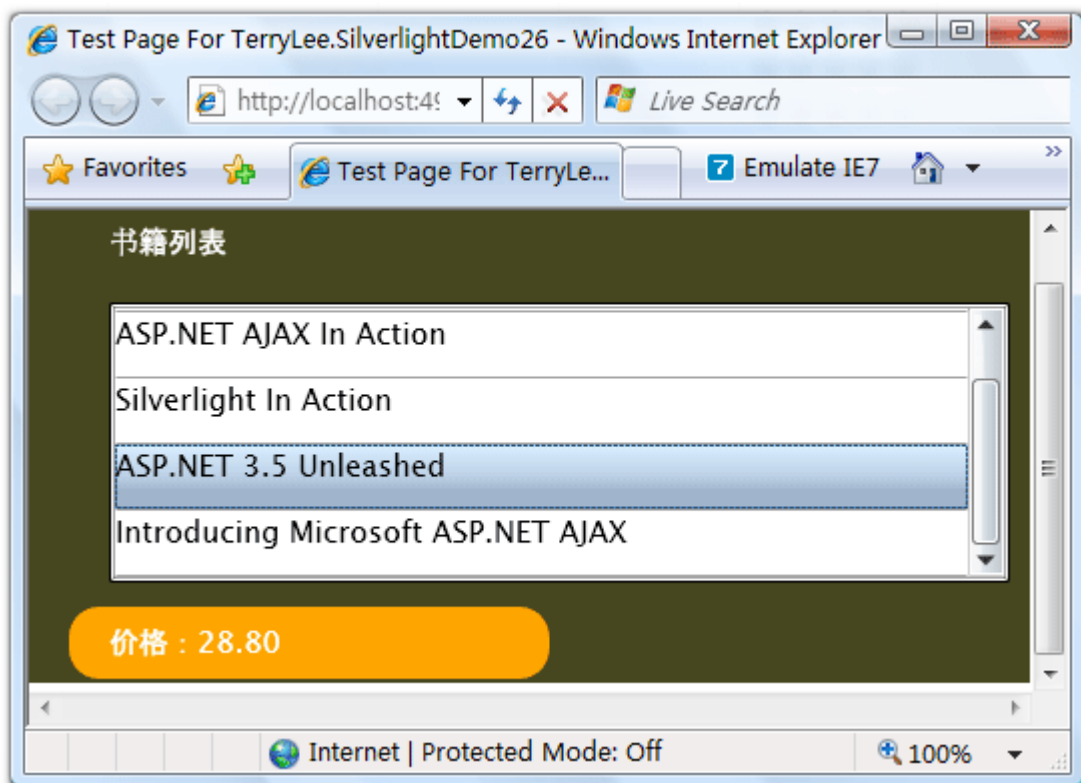
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文将简单介绍在 Silverlight 2 中如何使用 WebRequest 进行数据的提交和获取。

### 简单示例

在本文中，我们仍然使用在一步一步学 Silverlight 2 系列（12）：数据与通信之 WebClient 中用过的示例，只不过稍微做一点小的改动，使用 WebRequest 提交书籍编号数据，并根据书籍号返回价格信息。最终运行的结果如下图：



编写界面布局，XAML 如下：

```
<Grid Background="#46461F">
```

```

<Grid.RowDefinitions>
    <RowDefinition Height="40"></RowDefinition>
    <RowDefinition Height="*"></RowDefinition>
    <RowDefinition Height="40"></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>
<Border Grid.Row="0" Grid.Column="0" CornerRadius="15"
        Width="240" Height="36"
        Margin="20 0 0 0" HorizontalAlignment="Left">
    <TextBlock Text="书籍列表" Foreground="White"
        HorizontalAlignment="Left" VerticalAlignment="Center"
        Margin="20 0 0 0"></TextBlock>
</Border>
<ListBox x:Name="Books" Grid.Row="1" Margin="40 10 10 10"
        SelectionChanged="Books_SelectionChanged">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="{Binding Name}" Height="32"></TextBlock>
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
<Border Grid.Row="2" Grid.Column="0" CornerRadius="15"
        Width="240" Height="36" Background="Orange"
        Margin="20 0 0 0" HorizontalAlignment="Left">

```

```
<TextBlock x:Name="lblPrice" Text="价格: " Foreground="White"
           HorizontalAlignment="Left" VerticalAlignment="Center"
           Margin="20 0 0 0"></TextBlock>

</Border>

</Grid>
```

编写 `HttpHandler`，注意我使用了 `context.Request.Form["No"]`，在后面我们将使用 `WebRequest` 在 `RequestReady` 方法中将数据写入请求流：

```
public class BookHandler : IHttpHandler
{
    public static readonly string[] PriceList = new string[] {
        "66.00",
        "78.30",
        "56.50",
        "28.80",
        "77.00"
    };

    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "text/plain";
        context.Response.Write(PriceList[Int32.Parse(context.Request.Form["No"])]);
    }

    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}
```

```
}  
  
}  
  
}
```

在界面加载时绑定书籍列表，关于数据绑定可以参考[一步一步学 Silverlight 2 系列（11）：数据绑定](#)。

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)  
{  
  
    List<Book> books = new List<Book>() {  
        new Book("Professional ASP.NET 3.5"),  
        new Book("ASP.NET AJAX In Action"),  
        new Book("Silverlight In Action"),  
        new Book("ASP.NET 3.5 Unleashed"),  
        new Book("Introducing Microsoft ASP.NET AJAX")  
    };  
  
    Books.ItemsSource = books;  
  
}
```

接下来在 SelectionChanged 事件中实现用户选择书籍时，我们使用 WebRequest 提交书籍编号，并且获得价格数据，仍然采用异步模式，提供 RequestReady 和 ResponseReady 两个回调函数：

```
private string bookNo;  
  
void Books_SelectionChanged(object sender, SelectionChangedEventArgs e)  
{  
  
    bookNo = Books.SelectedIndex.ToString();  
  
    Uri endpoint = new Uri("http://localhost:49955/BookHandler.ashx");
```

```
WebRequest request = WebRequest.Create(endpoint);

request.Method = "POST";

request.ContentType = "application/x-www-form-urlencoded";

request.BeginGetRequestStream(new AsyncCallback(RequestReady), request);

request.BeginGetResponse(new AsyncCallback(ResponseReady), request);

}
```

实现 RequestReady 方法，将书籍的编号写入请求流中。

```
void RequestReady(IAsyncResult asyncResult)
{
    WebRequest request = asyncResult.AsyncState as WebRequest;
    Stream requestStream = request.EndGetRequestStream(asyncResult);

    using (StreamWriter writer = new StreamWriter(requestStream))
    {
        writer.Write(String.Format("No={0}", bookNo));
        writer.Flush();
    }
}
```

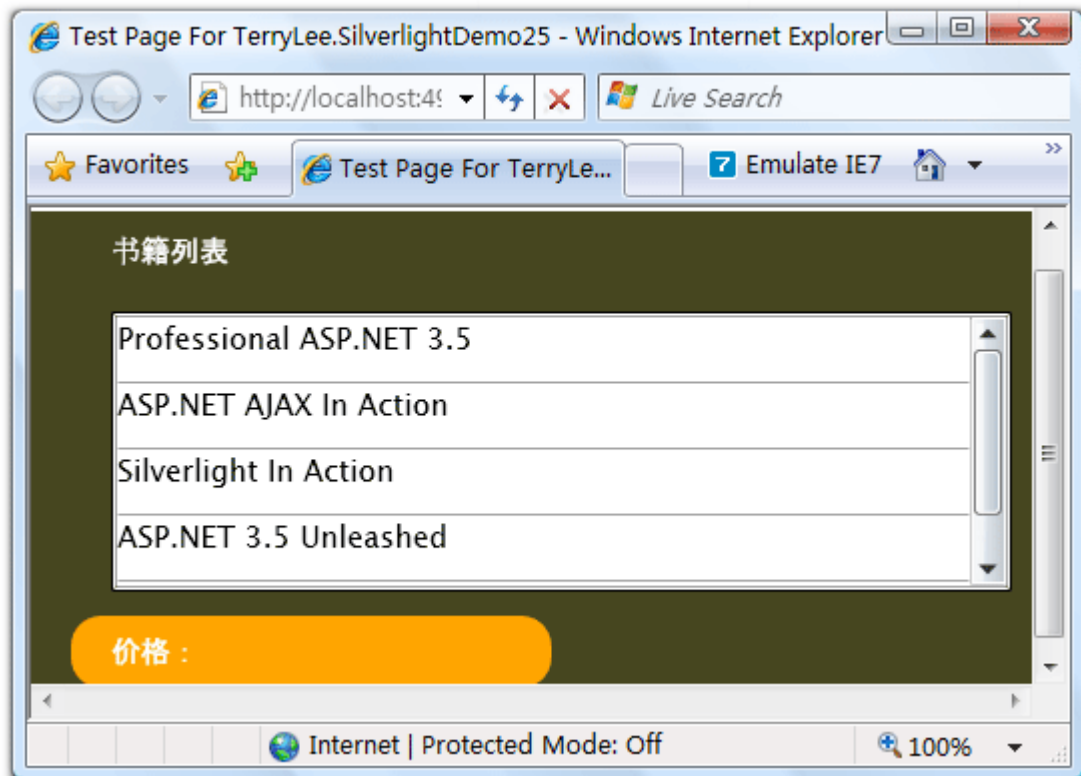
实现 ResponseReady 方法，显示返回的结果。

```
void ResponseReady(IAsyncResult asyncResult)
{
    WebRequest request = asyncResult.AsyncState as WebRequest;
    WebResponse response = request.EndGetResponse(asyncResult);

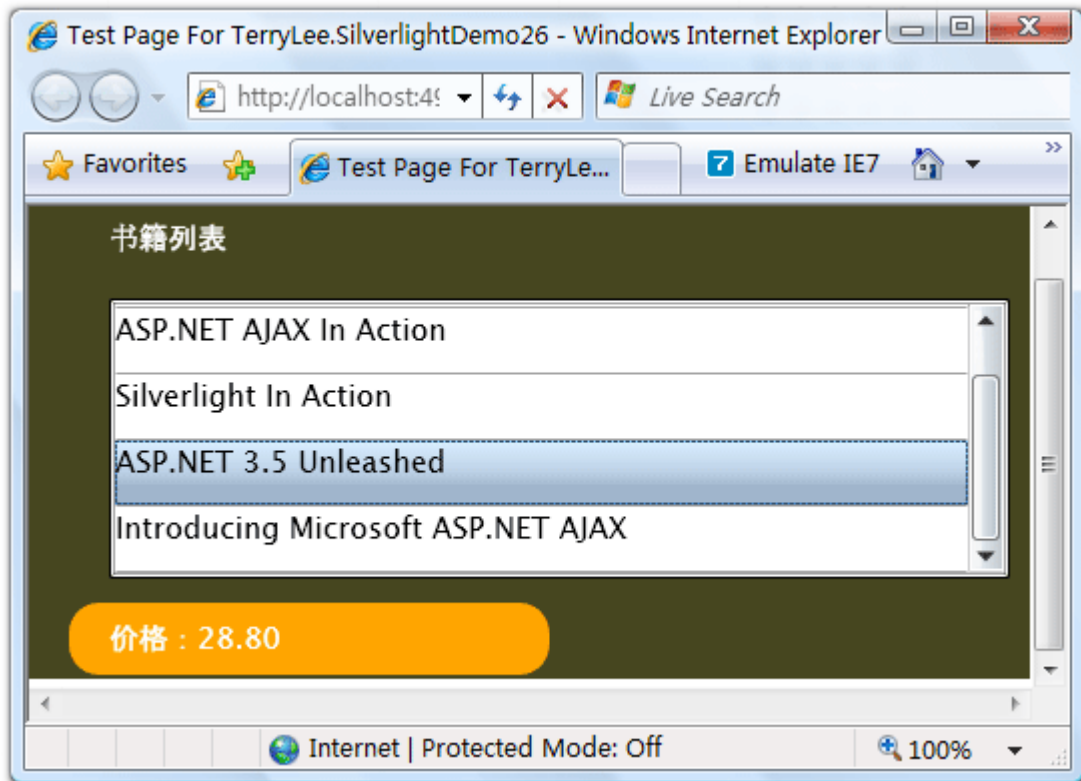
    using (Stream responseStream = response.GetResponseStream())
    {
```

```
StreamReader reader = new StreamReader(responseStream);  
  
lblPrice.Text = "价格: " + reader.ReadToEnd();  
  
}  
  
}
```

最后运行的结果如下：



用户选择一本书籍后，将显示其价格：



## 结束语

本文简单介绍了在 Silverlight 2 中如何使用 WebRequest 提交和获取数据,你可以从[这里](#)下载示例程序。



## 一步一步学 Silverlight 2 系列（14）：数据与通信之 WCF

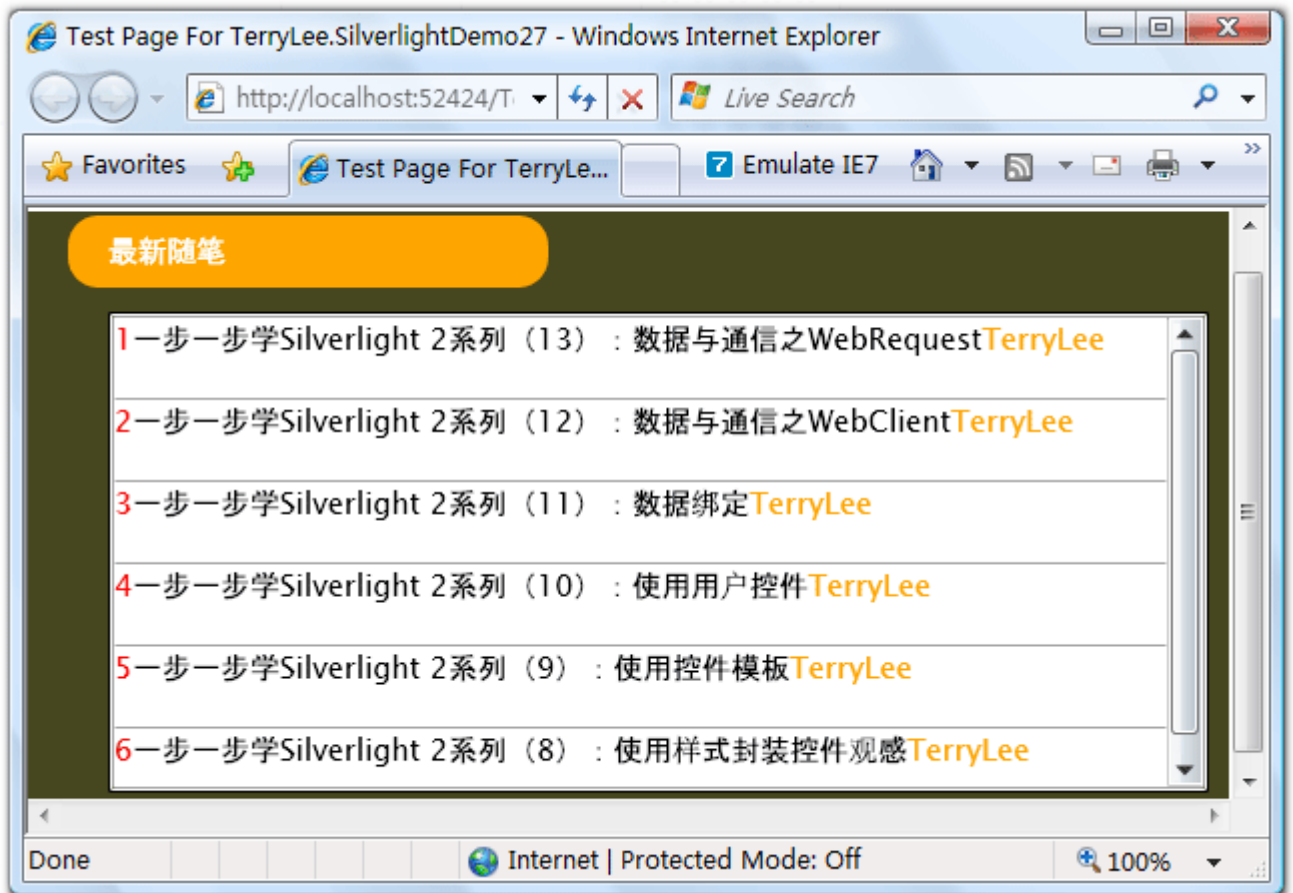
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发。

本文将简单介绍在 Silverlight 2 中如何与 WCF 进行通信。

### 简单示例

在本示例中，我们将通过 WCF 来获取一个最新随笔的列表，在 Silverlight 中显示出来，最终完后效果如下所示。



先定义一个数据契约：

```
[DataContract]
```

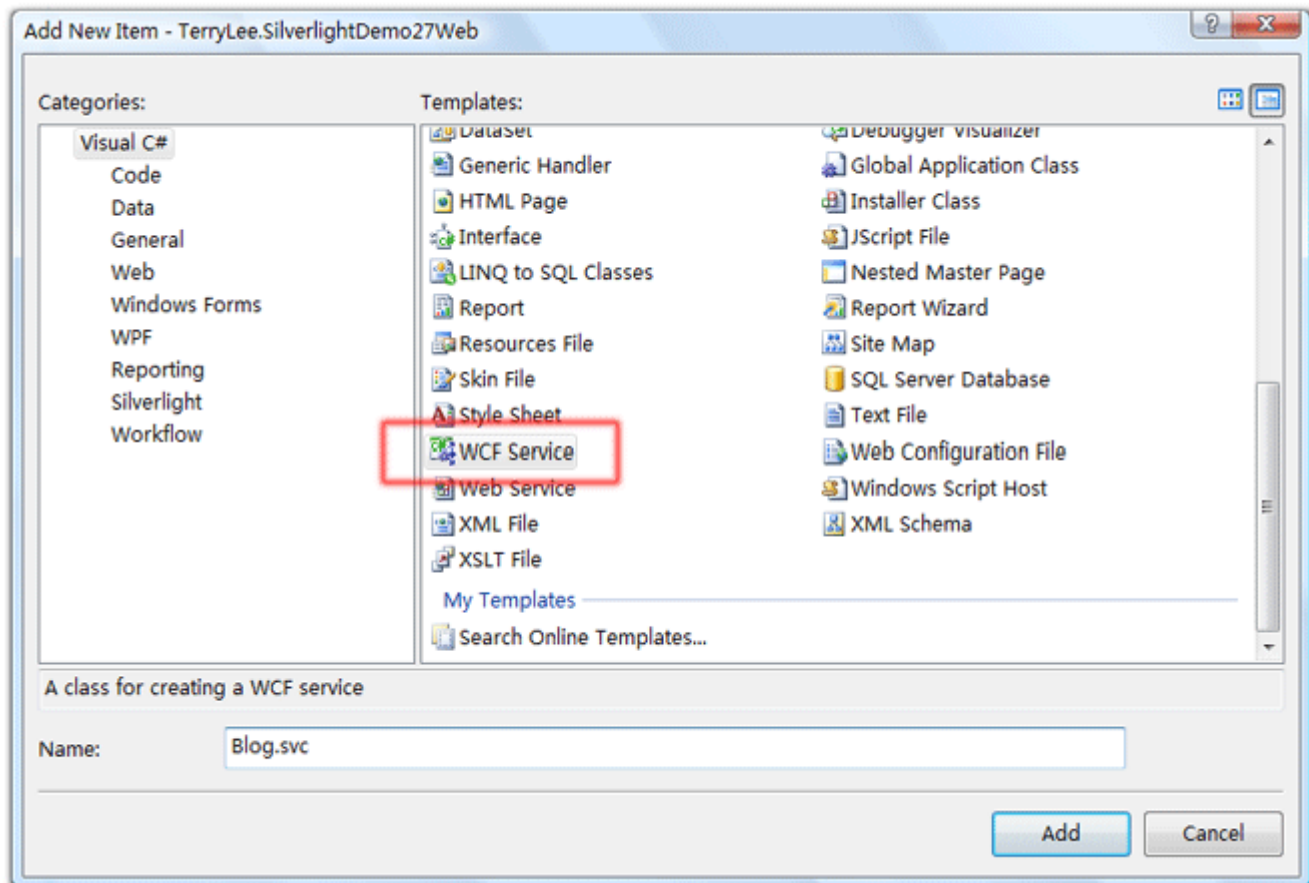
```
public class Post
{
    public Post(int id, string title, string author)
    {
        this.Id = id;
        this.Title = title;
        this.Author = author;
    }

    [DataMember]
    public int Id { get; set; }

    [DataMember]
    public string Title { get; set; }

    [DataMember]
    public string Author { get; set; }
}
```

在 Web 项目中添加一个 WCF Service 文件，命名为 Blog.svc



定义服务契约:

```
[ServiceContract]
public interface IBlog
{
    [OperationContract]
    Post[] GetPosts();
}
```

实现服务, 这里可以从数据库或者其他数据源读取, 为了演示方便, 我们直接初始化一个集合:

```
public class Blog : IBlog
{
    public Post[] GetPosts()
    {
```

```

List<Post> posts = new List<Post>()
{
    new Post(1, "一步一步学 Silverlight 2 系列 (13) : 数据与通信之 WebRequest",
    "TerryLee"),
    new Post(2, "一步一步学 Silverlight 2 系列 (12) : 数据与通信之 WebClient",
    "TerryLee"),
    new Post(3, "一步一步学 Silverlight 2 系列 (11) : 数据绑定", "TerryLee"),
    new Post(4, "一步一步学 Silverlight 2 系列 (10) : 使用用户控件", "TerryLee
"),
    new Post(5, "一步一步学 Silverlight 2 系列 (9) : 使用控件模板", "TerryLee"),
    new Post(6, "一步一步学 Silverlight 2 系列 (8) : 使用样式封装控件观感", "T
erryLee")
};

return posts.ToArray();
}
}

```

修改 Web.config 中的服务配置，这里使用 basicHttpBinding 绑定，并且开启 httpGetEnabled，以便后面我们可以在浏览器中查看服务：

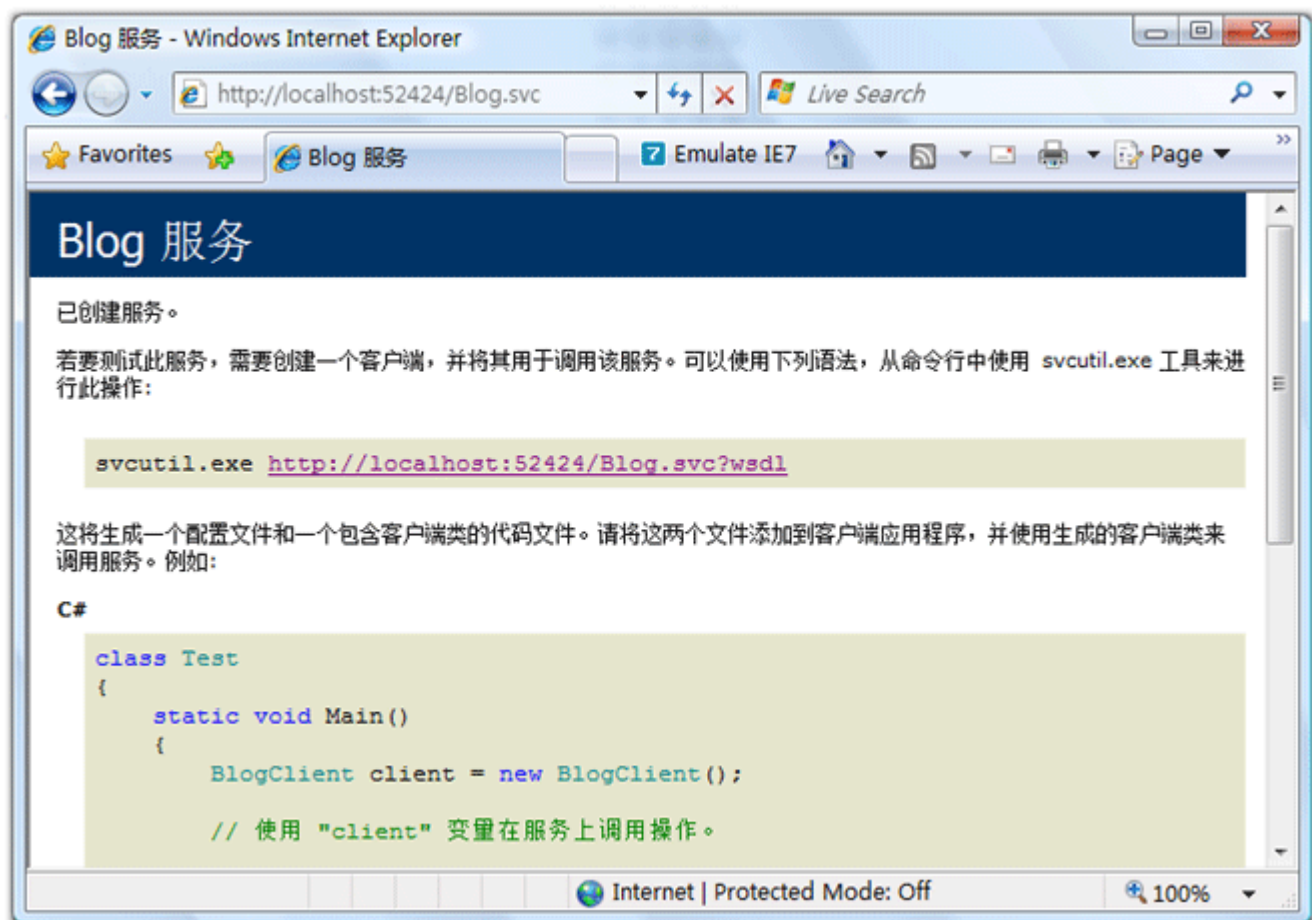
```

<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="TerryLee.SilverlightDemo27Web.BlogBehavior">
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>

```

```
<services>
  <service behaviorConfiguration="TerryLee.SilverlightDemo27Web.BlogBehavior"
    name="TerryLee.SilverlightDemo27Web.Blog">
    <endpoint address="" binding="basicHttpBinding" contract="TerryLee.SilverlightDemo27Web.IBlog">
    </endpoint>
  </service>
</services>
</system.serviceModel>
```

设置一下 Web 应用程序的端口号为固定端口 52424，在浏览器中输入 <http://localhost:52424/Blog.svc>，看看服务是否正常：



好了，现在服务端我们就实现完成了。现在编写界面展示部分，XAML 如下：

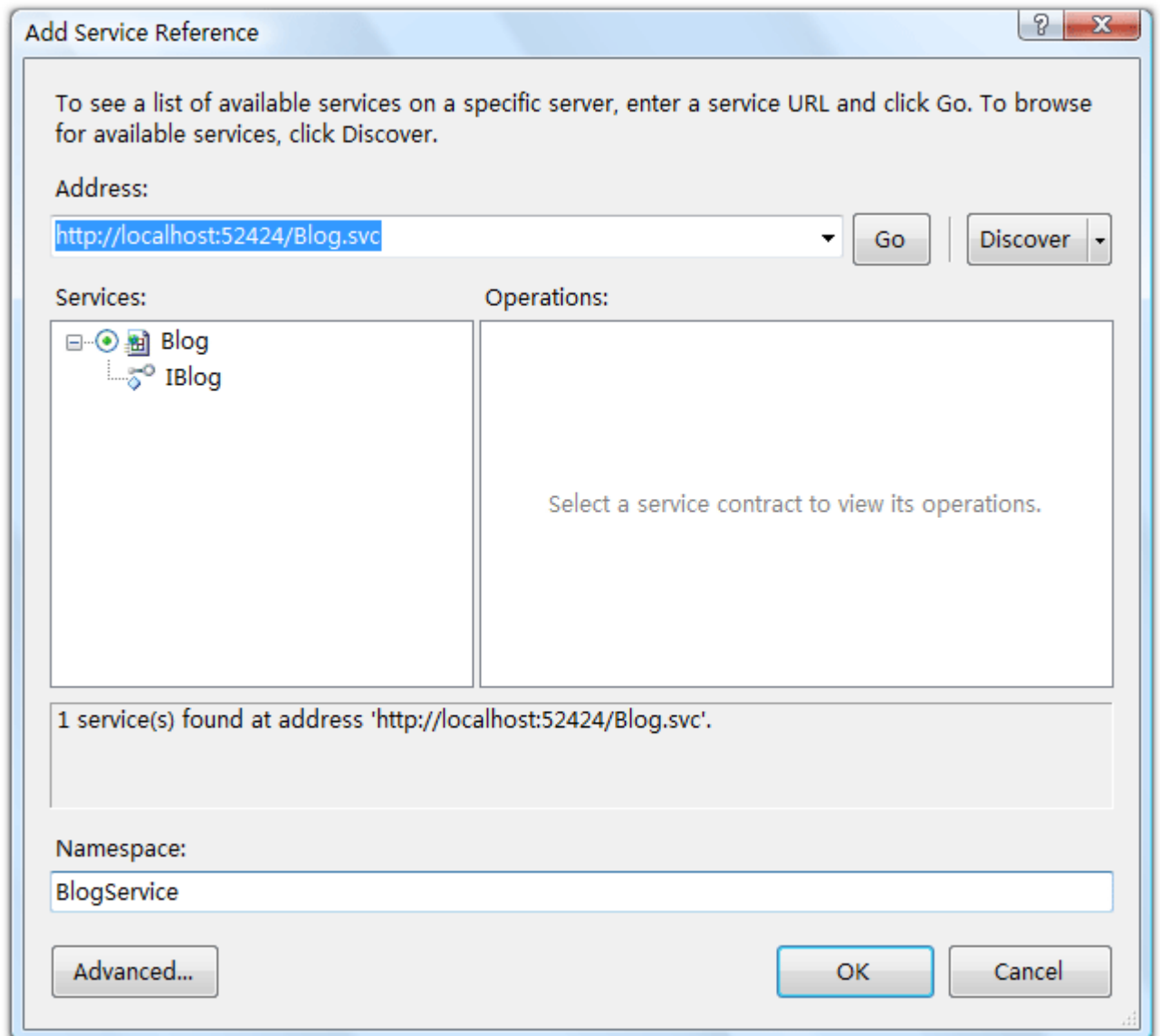
```

<Grid Background="#46461F">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Border Grid.Row="0" Grid.Column="0" CornerRadius="15"
        Width="240" Height="36" Background="Orange"
        Margin="20 0 0 0" HorizontalAlignment="Left">
        <TextBlock Text="最新随笔" Foreground="White"
            HorizontalAlignment="Left" VerticalAlignment="Center"
            Margin="20 0 0 0"></TextBlock>
    </Border>
    <ListBox x:Name="Posts" Grid.Row="1" Margin="40 10 10 10">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="{Binding Id}" Height="40" Foreground="Red"></TextBlock>
                    <TextBlock Text="{Binding Title}" Height="40"></TextBlock>
                    <TextBlock Text="{Binding Author}" Height="40" Foreground="Orange"></TextBlock>
                </StackPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>

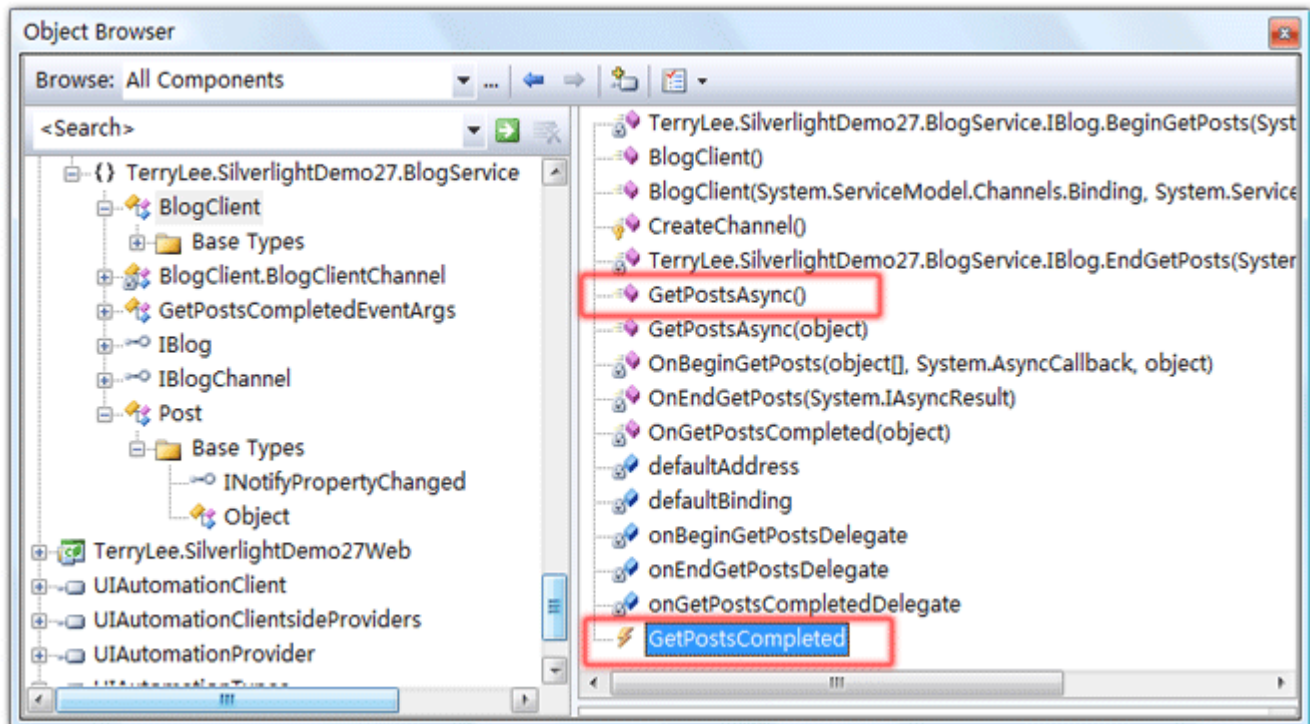
```

```
</Grid>
```

在 Silverlight 项目中添加服务引用，输入地址 <http://localhost:52424/Blog.svc>，输入命名空间 Blog Service。



添加完成后，我们可以在对象浏览器中浏览一下生成的客户端对象：



当然大家也可以手工去编写客户端的代码，请参考 WCF 的相关内容，这里不再赘述。下面编写调用服务并获取数据，这里仍然是采用异步模式，由于在 WCF 服务的配置中我们采取了 BasicHttpBinding，客户端也要采用 BasicHttpBinding。我们需要注册 GetPostsCompleted 事件处理方法，以便完成后回调，同时调用 GetPostsAsync() 方法获取数据。完整的代码如下所示：

```
public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
    }

    private void UserControl_Loaded(object sender, RoutedEventArgs e)
    {
        Binding binding = new BasicHttpBinding();
        EndpointAddress endPoint = new EndpointAddress(
            "http://localhost:52424/Blog.svc");
```



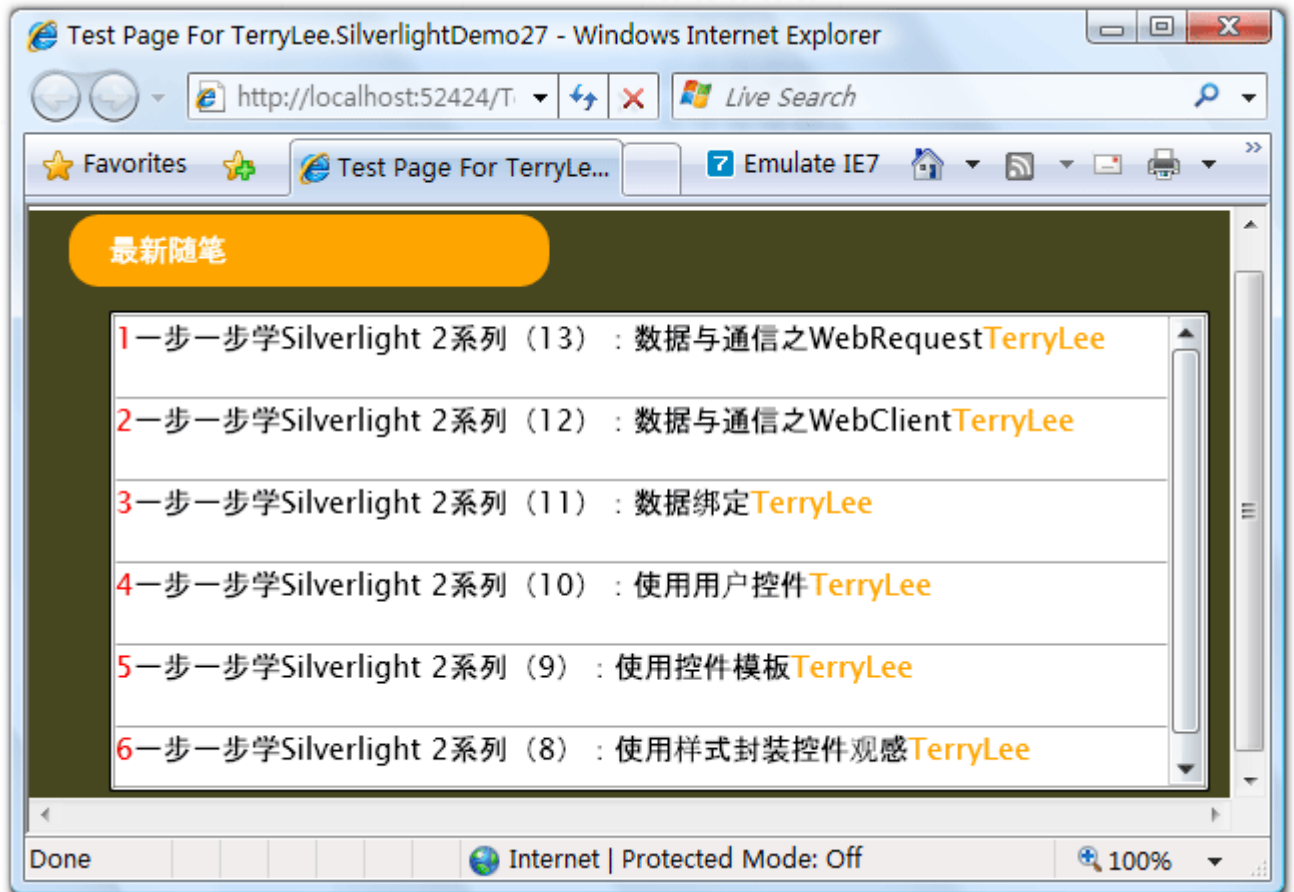
```
BlogClient client = new BlogClient(binding, endPoint);

client.GetPostsCompleted += new EventHandler<GetPostsCompletedEventArgs>(client_GetPostsCompleted);

client.GetPostsAsync();
}

void client_GetPostsCompleted(object sender, GetPostsCompletedEventArgs e)
{
    if (e.Error == null)
    {
        Posts.ItemsSource = e.Result;
    }
}
}
```

至此，一个完整的在 Silverlight 2 中调用 WCF 的示例就完成了，运行后效果如下：



## 结束语

本文简单演示了在 Silverlight 2 中如何与 WCF 进行通信，你可以从[这里](#)下载示例代码。

## 一步一步学 Silverlight 2 系列（15）：数据与通信之 ASMX

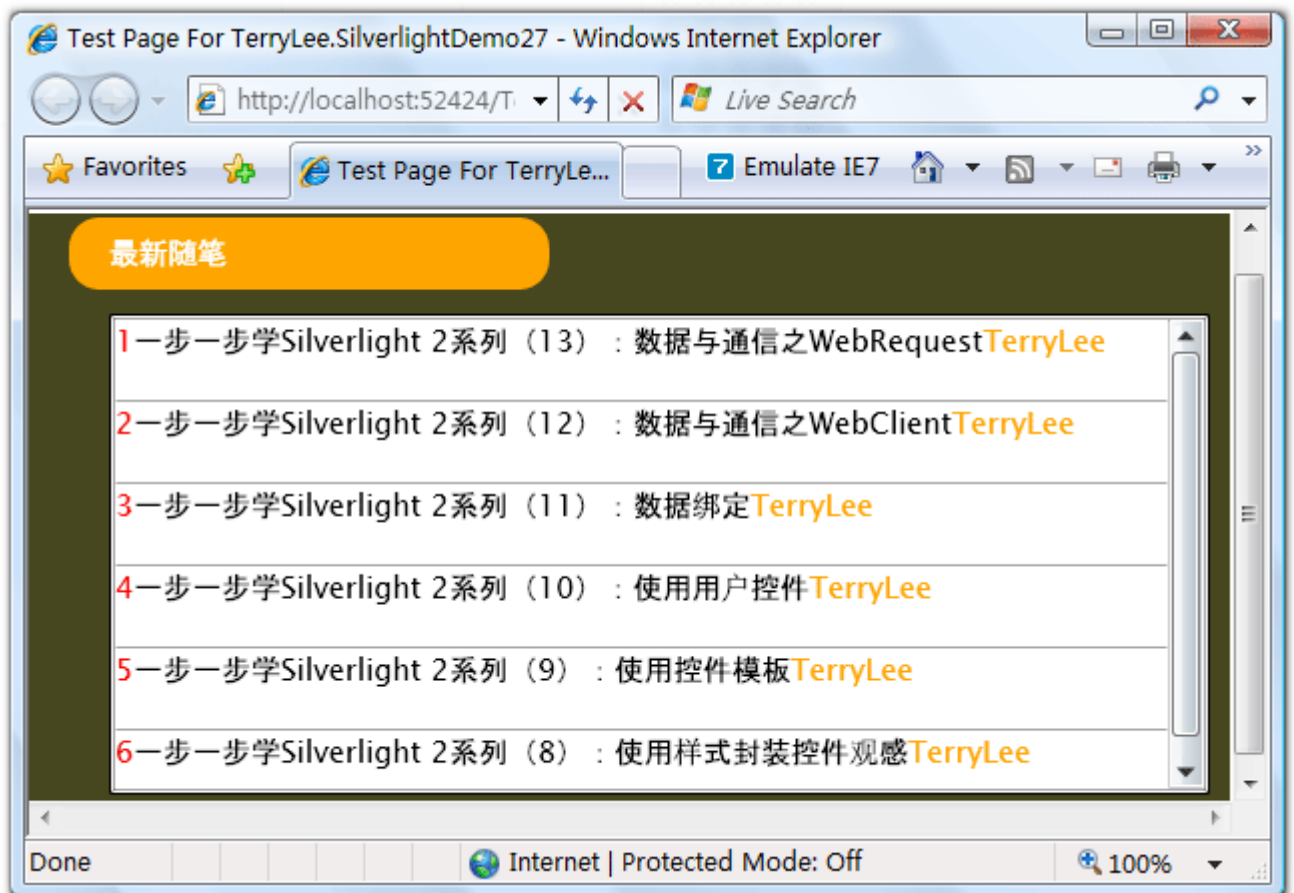
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发。

本文将简单介绍在 Silverlight 2 中如何与 ASMX 进行通信。

### 简单示例

本文的示例非常简单，其过程也跟我们在[一步一步学 Silverlight 2 系列（14）：数据与通信之 WCF](#) 中差不多，我们仍然显示一个最新随笔的列表，最终完成后效果如下所示：

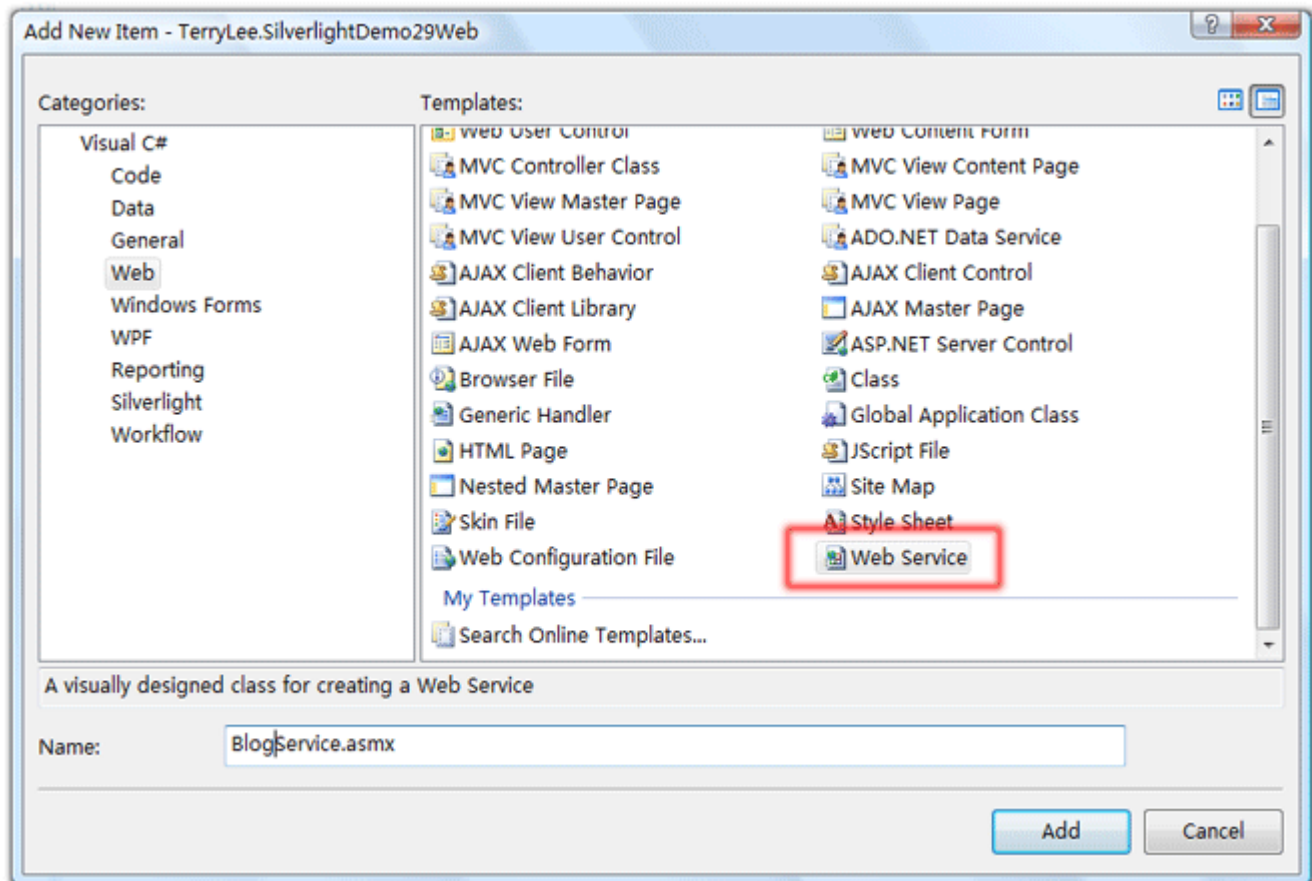


定义一个业务实体 Post。

```
public class Post
```

```
{  
  
    public int Id { get; set; }  
  
    public string Title { get; set; }  
  
    public string Author { get; set; }  
  
}
```

在 Web 项目中添加一个 Web Service 文件，命名为 BlogService.asmx



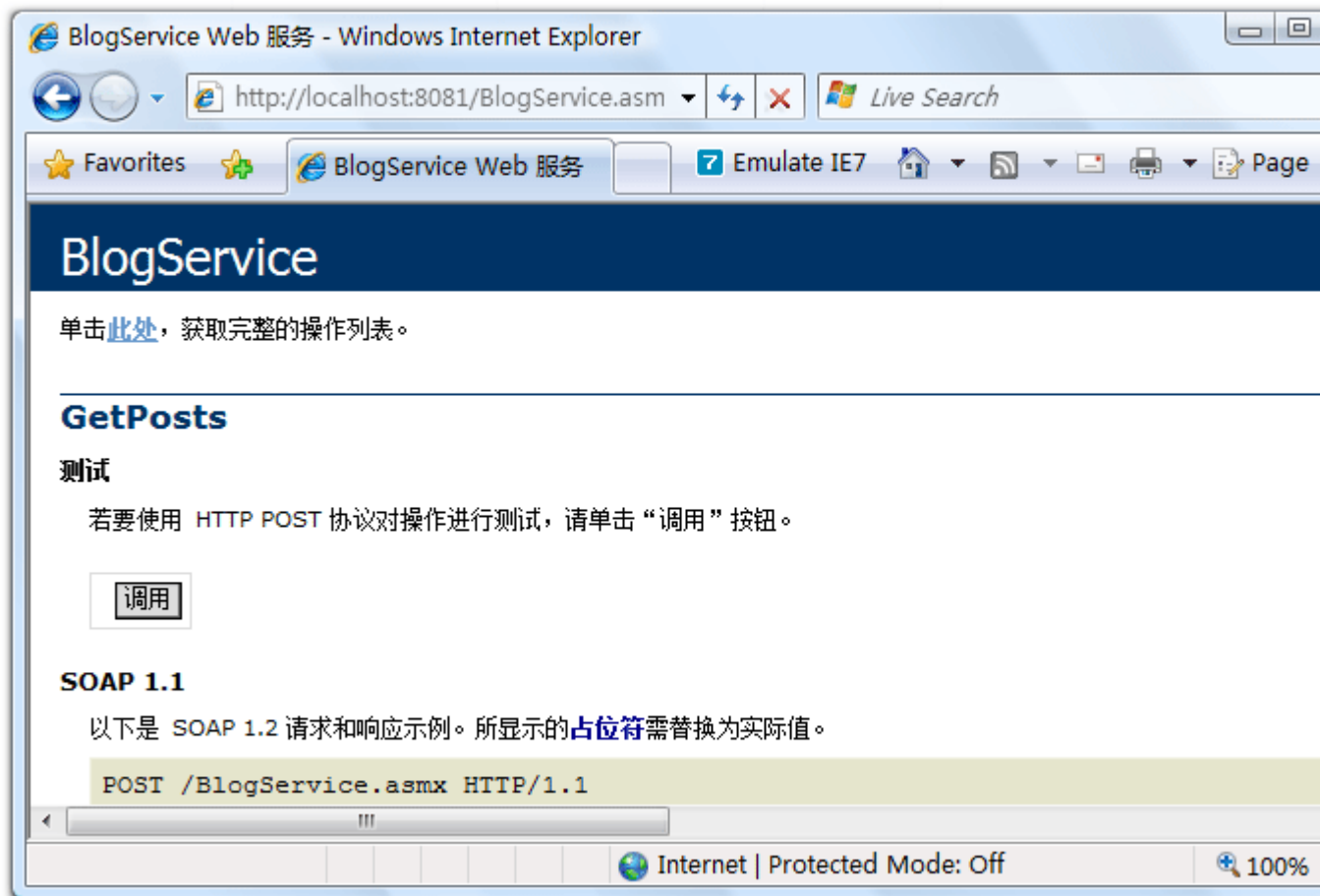
实现该服务，定义一个 GetPosts 方法：

```
public class BlogService : WebService  
{  
  
    [WebMethod]
```

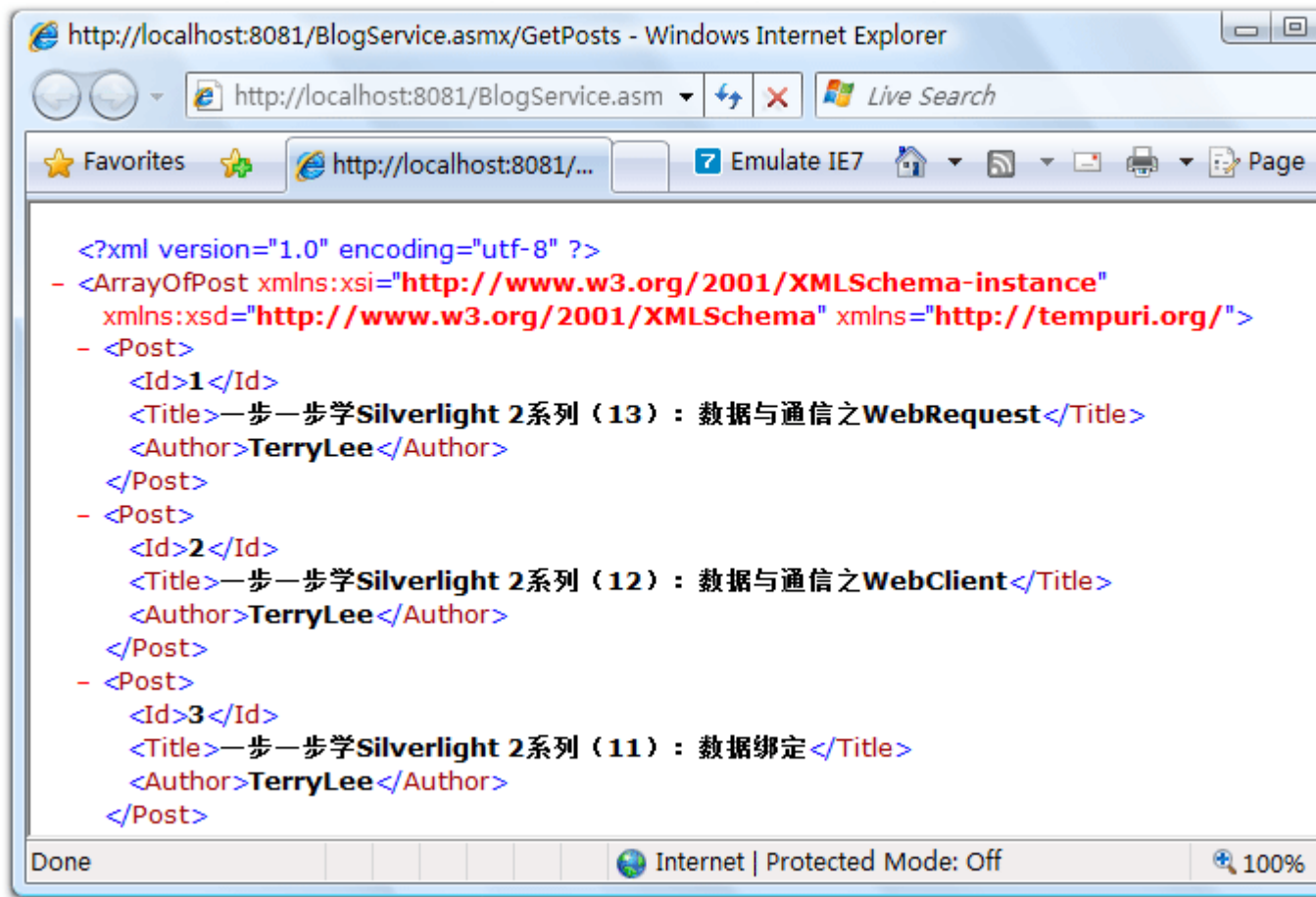
```
public Post[] GetPosts()
{
    List<Post> posts = new List<Post>()
    {
        new Post{ Id=1, Title="一步一步学 Silverlight 2 系列 (13) : 数据与通信之 WebRequest", Author="TerryLee" },
        new Post{ Id=2, Title="一步一步学 Silverlight 2 系列 (12) : 数据与通信之 WebClient", Author="TerryLee" },
        new Post{ Id=3, Title="一步一步学 Silverlight 2 系列 (11) : 数据绑定", Author="TerryLee" },
        new Post{ Id=4, Title="一步一步学 Silverlight 2 系列 (10) : 使用用户控件", Author="TerryLee" },
        new Post{ Id=5, Title="一步一步学 Silverlight 2 系列 (9) : 使用控件模板", Author="TerryLee" },
        new Post{ Id=6, Title="一步一步学 Silverlight 2 系列 (8) : 使用样式封装控件观感", Author="TerryLee" }
    };

    return posts.ToArray();
}
}
```

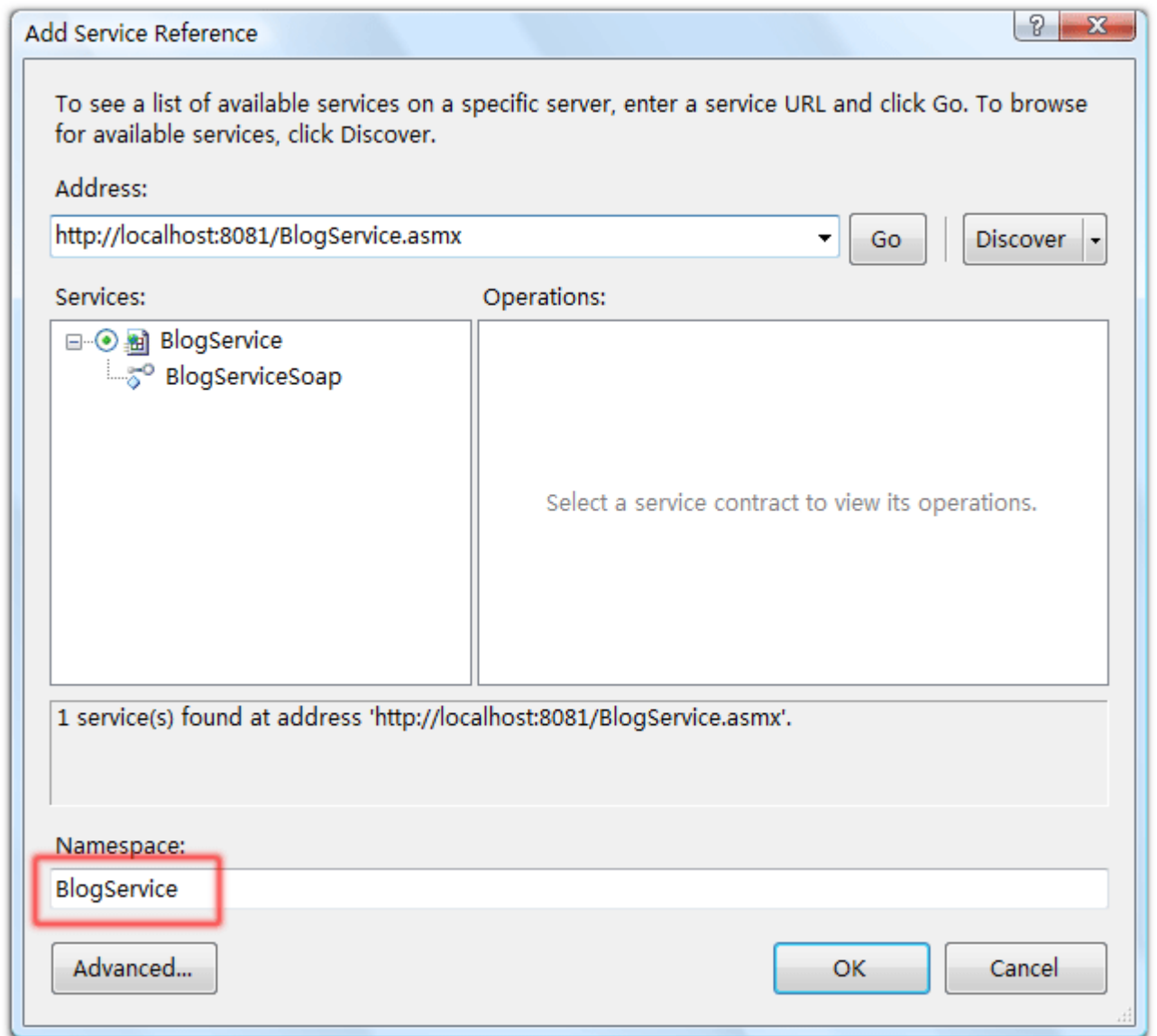
同样设置 Web Development Server 的端口号为一个固定值，这里设为 8081，然后在浏览器中测试服务是否正确：



点击调用后测试服务正确

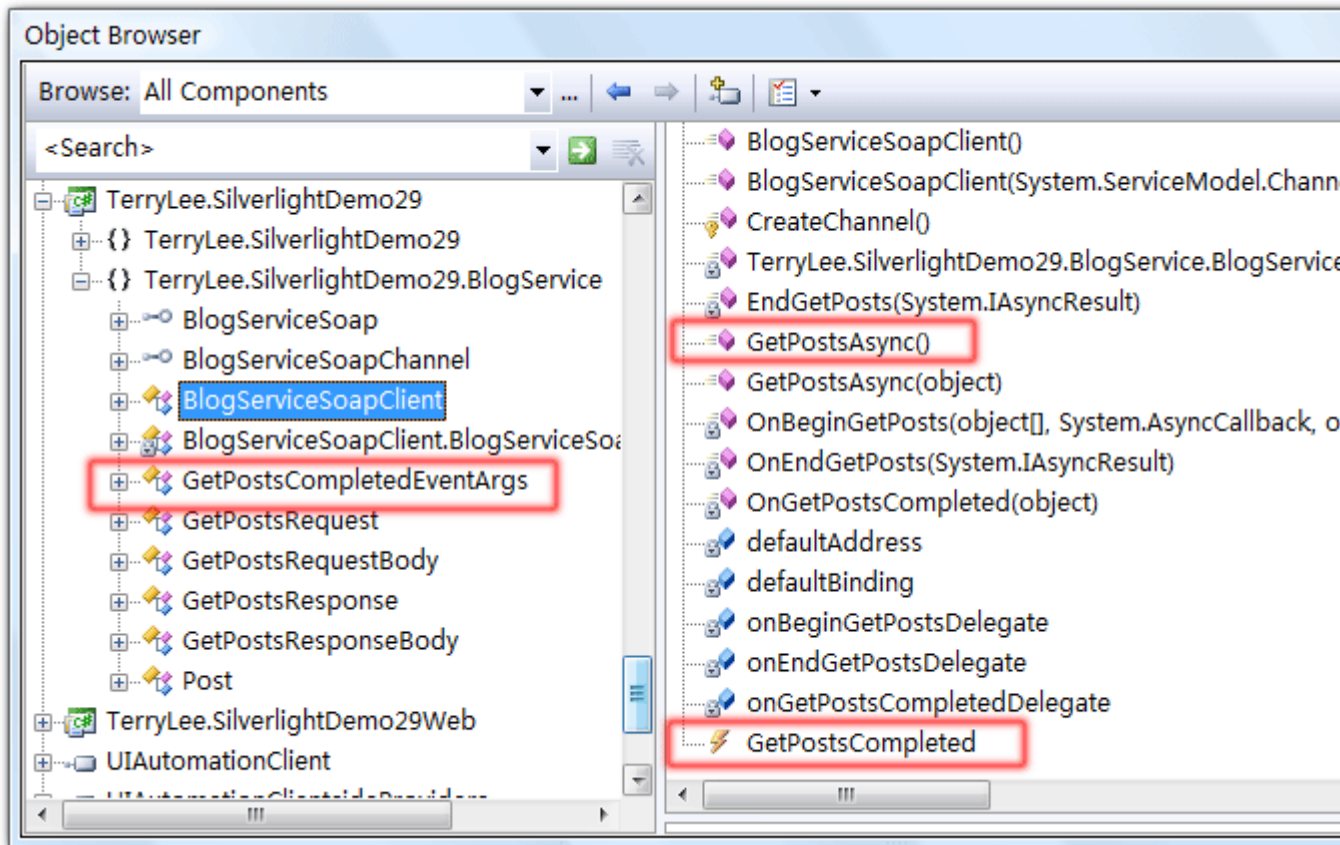


在 Silverlight 项目中，添加对服务引用，



使用对象浏览器查看一下生成客户端代理类中的对象：





编写展示界面，XAML 如下，与上一篇中的示例一样：

```

<Grid Background="#46461F">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Border Grid.Row="0" Grid.Column="0" CornerRadius="15"
        Width="240" Height="36" Background="Orange"
        Margin="20 0 0 0" HorizontalAlignment="Left">
        <TextBlock Text="最新随笔" Foreground="White"
            HorizontalAlignment="Left" VerticalAlignment="Center"
    
```

```

        Margin="20 0 0 0"></TextBlock>

</Border>

<ListBox x:Name="Posts" Grid.Row="1" Margin="40 10 10 10">

    <ListBox.ItemTemplate>

        <DataTemplate>

            <StackPanel Orientation="Horizontal">

                <TextBlock Text="{Binding Id}" Height="40" Foreground="Red"></TextBlock>

                <TextBlock Text="{Binding Title}" Height="40"></TextBlock>

                <TextBlock Text="{Binding Author}" Height="40" Foreground="Orange"></TextBlock>

            </StackPanel>

        </DataTemplate>

    </ListBox.ItemTemplate>

</ListBox>

</Grid>

```

实现调用 ASMX，并进行数据的绑定。仍然采用异步模式，所使用的方法如上图红色框中的部分。过程与 WCF 通信差不多，只不过不再需要指定 Binding 等信息：

```

public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
    }

    private void UserControl_Loaded(object sender, RoutedEventArgs e)
    {
        BlogServiceSoapClient client = new BlogServiceSoapClient();
    }
}

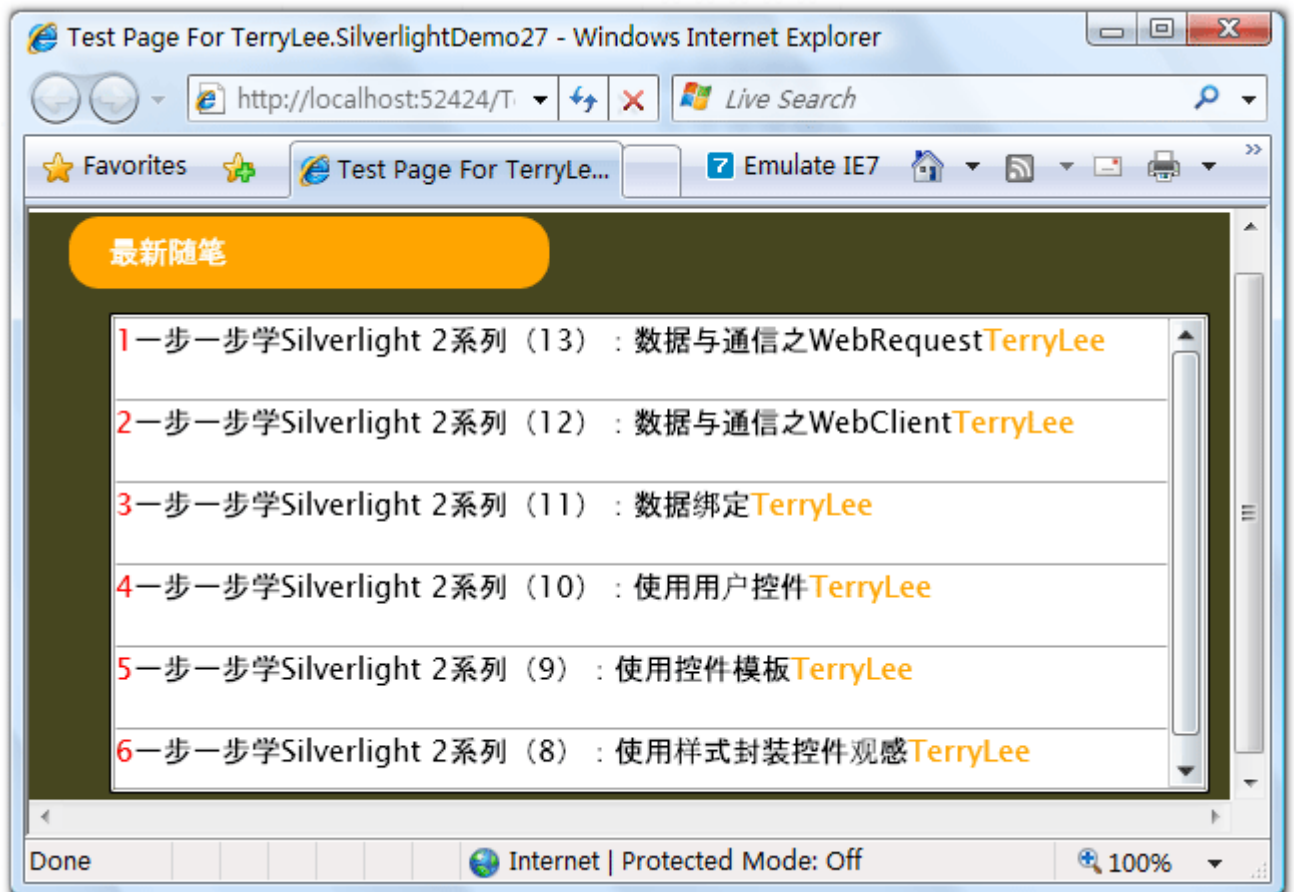
```

```
        client.GetPostsCompleted += new EventHandler<GetPostsCompletedEventArgs>(client_GetPostsCompleted);

        client.GetPostsAsync();
    }

    void client_GetPostsCompleted(object sender, GetPostsCompletedEventArgs e)
    {
        if (e.Error == null)
        {
            Posts.ItemsSource = e.Result;
        }
    }
}
```

一个完整的 Silverlight 2 中调用 ASMX 的示例就完成了，运行后效果如下：



## 结束语

本文简单介绍了在 Silverlight 2 中如何调用 ASMX，你可以从[这里](#)下载示例代码。

## 一步一步学 Silverlight 2 系列（16）：数据与通信之 JSON

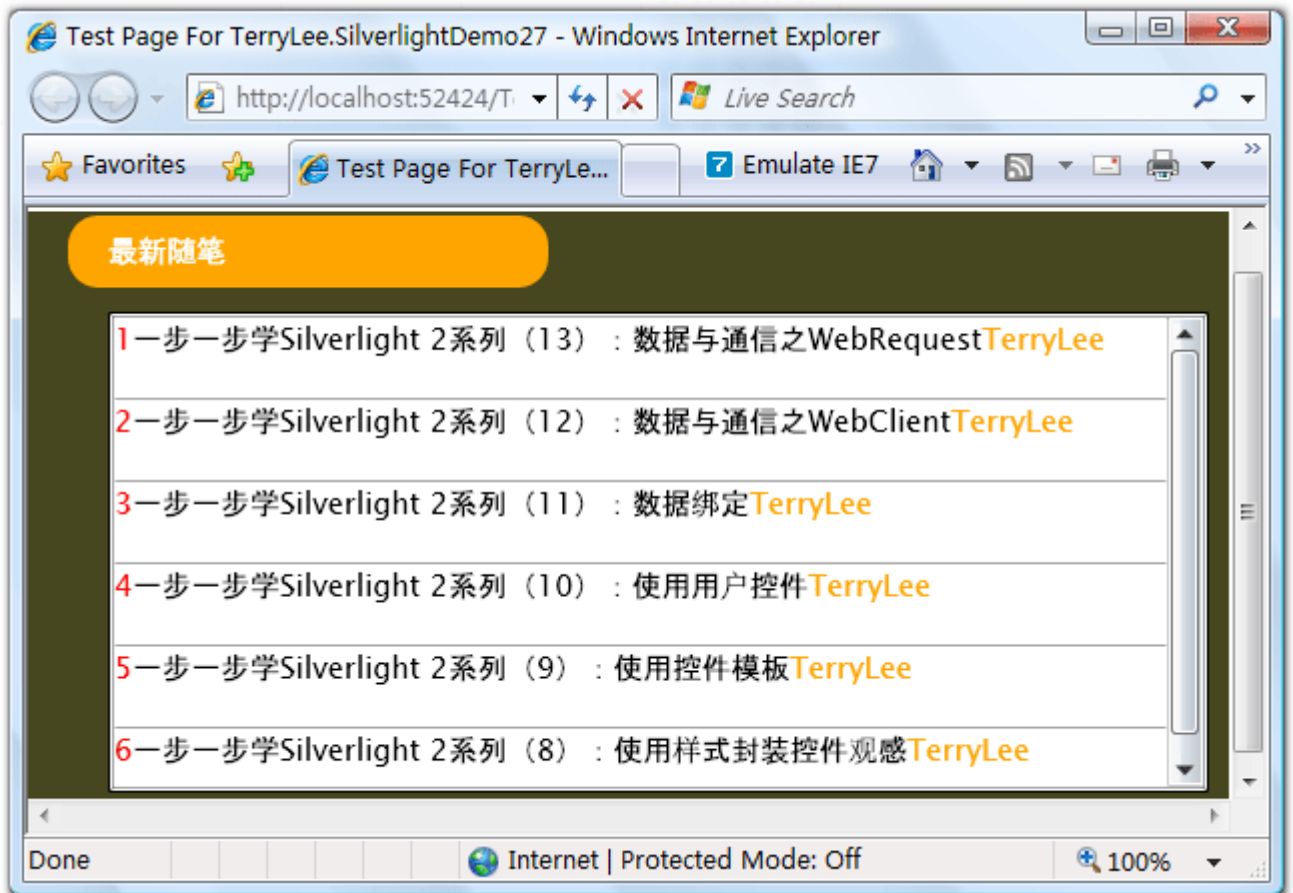
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章带您快速进入 Silverlight 2 开发。

本文将简单介绍在 Silverlight 2 中对于 JSON 的支持。

### 简单示例

在本文中我们仍然采用前面两篇文章中用过的显示最新随笔这样一个示例（举一反三嘛:)), 最终完成的效果如下图所示:



首先我们建立服务端，以便能够提供 JSON 格式的数据。在这里为了产生 JSON 格式的数据，我们借助于一个开源项目 [Json.NET](#)。建立两个实体类型：

```

public class Post
{
    public int Id { get; set; }

    public string Title { get; set; }

    public string Author { get; set; }
}

public class Blog
{
    public List<Post> Posts { get; set; }
}

```

在 Silverlight 项目中我们也会使用到这两个实体类，新建一个 `HttpHandler`，产生 JSON 格式数据，我们使用 `Json.NET` 中的 `JavaScriptConvert.SerializeObject` 方法即可序列化一个对象为 JSON 格式：

```

public class BlogHandler : IHttpHandler
{

    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "text/plain";

        List<Post> posts = new List<Post>()
        {
            new Post{ Id=1, Title="一步一步学 Silverlight 2 系列 (13) : 数据与通信之 WebRequest", Author="TerryLee" },
            new Post{ Id=2, Title="一步一步学 Silverlight 2 系列 (12) : 数据与通信之 WebClient", Author="TerryLee" },
        }
    }
}

```

```

        new Post{ Id=3, Title="一步一步学 Silverlight 2 系列 (11) : 数据绑定", Author="TerryLee" },

        new Post{ Id=4, Title="一步一步学 Silverlight 2 系列 (10) : 使用用户控件", Author="TerryLee" },

        new Post{ Id=5, Title="一步一步学 Silverlight 2 系列 (9) : 使用控件模板", Author="TerryLee" },

        new Post{ Id=6, Title="一步一步学 Silverlight 2 系列 (8) : 使用样式封装控件观感", Author="TerryLee" }

    };

    Blog blog = new Blog();

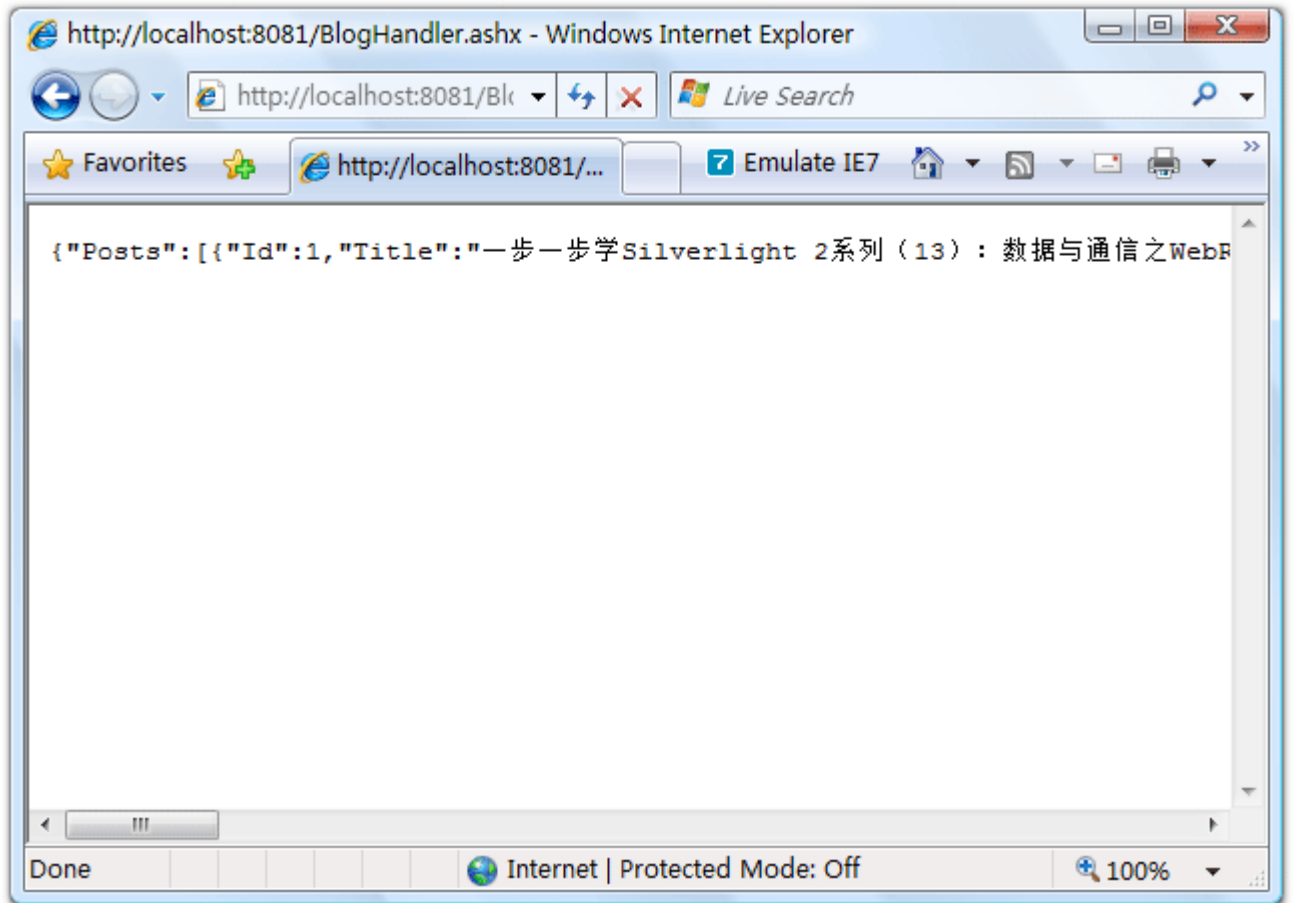
    blog.Posts = posts;

    context.Response.Write(JavaScriptConvert.SerializeObject(blog));
}

public bool IsReusable
{
    get
    {
        return false;
    }
}
}
}

```

现在测试一下 HttpHandler，查看一下生成的数据格式：



对这些数据格式化一下，看起来更明显，这里推荐一个在线 JSON 数据格式化工具 <http://www.curiousconcept.com/jsonformatter/>:



## JSON Formatter

**JSON Data:**

```
学Silverlight 2系列 (9) : 使用控件模板", "Author": "TerryLee"}, {"Id": 6, "Title": "一步一步:
```

**JSON From URL:**

格式化后的数据如下:

## Result

[Close](#)

### Formatted JSON Data:

```
{
  "Posts": [
    {
      "Id": 1,
      "Title": "一步一步学Silverlight 2系列 (13) : 数据与通信之WebRequest",
      "Author": "TerryLee"
    },
    {
      "Id": 2,
      "Title": "一步一步学Silverlight 2系列 (12) : 数据与通信之WebClient",
      "Author": "TerryLee"
    },
    {
      "Id": 3,
      "Title": "一步一步学Silverlight 2系列 (11) : 数据绑定",
      "Author": "TerryLee"
    },
    {
      "Id": 4,
      "Title": "一步一步学Silverlight 2系列 (10) : 使用用户控件",
      "Author": "TerryLee"
    },
    {
      "Id": 5,
      "Title": "一步一步学Silverlight 2系列 (9) : 使用控件模板",
      "Author": "TerryLee"
    }
  ]
}
```

现在实现在 Silverlight 中获取 JSON 数据，并进行反序列化，界面布局 XAML 就不再贴出来了，跟前面两篇的示例一样。在 Silverlight 2 中，内置了对于 JSON 的支持，通过命名空间 `System.Runtime.Serialization.Json` 提供，位于 `System.ServiceModel.Web.dll` 中。

我们使用 `WebRequest` 获取数据：

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
    Uri endpoint = new Uri("http://localhost:8081/BlogHandler.ashx");

    WebRequest request = WebRequest.Create(endpoint);

    request.Method = "POST";

    request.ContentType = "application/x-www-form-urlencoded";
}
```

```
request.BeginGetResponse(new AsyncCallback(ResponseReady), request);  
}  
  
void ResponseReady(IAsyncResult asyncResult)  
{  
    WebRequest request = asyncResult.AsyncState as WebRequest;  
    WebResponse response = request.EndGetResponse(asyncResult);  
  
    using (Stream responseStream = response.GetResponseStream())  
    {  
        DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer  
(typeof(Blog));  
  
        Blog blog = jsonSerializer.ReadObject(responseStream) as Blog;  
  
        Posts.ItemsSource = blog.Posts;  
    }  
}
```

DataContractJsonSerializer 用于将对象序列化为 JSON 或者反序列化为对象实例，分别使用方法 WriteObject 和 ReadObject。

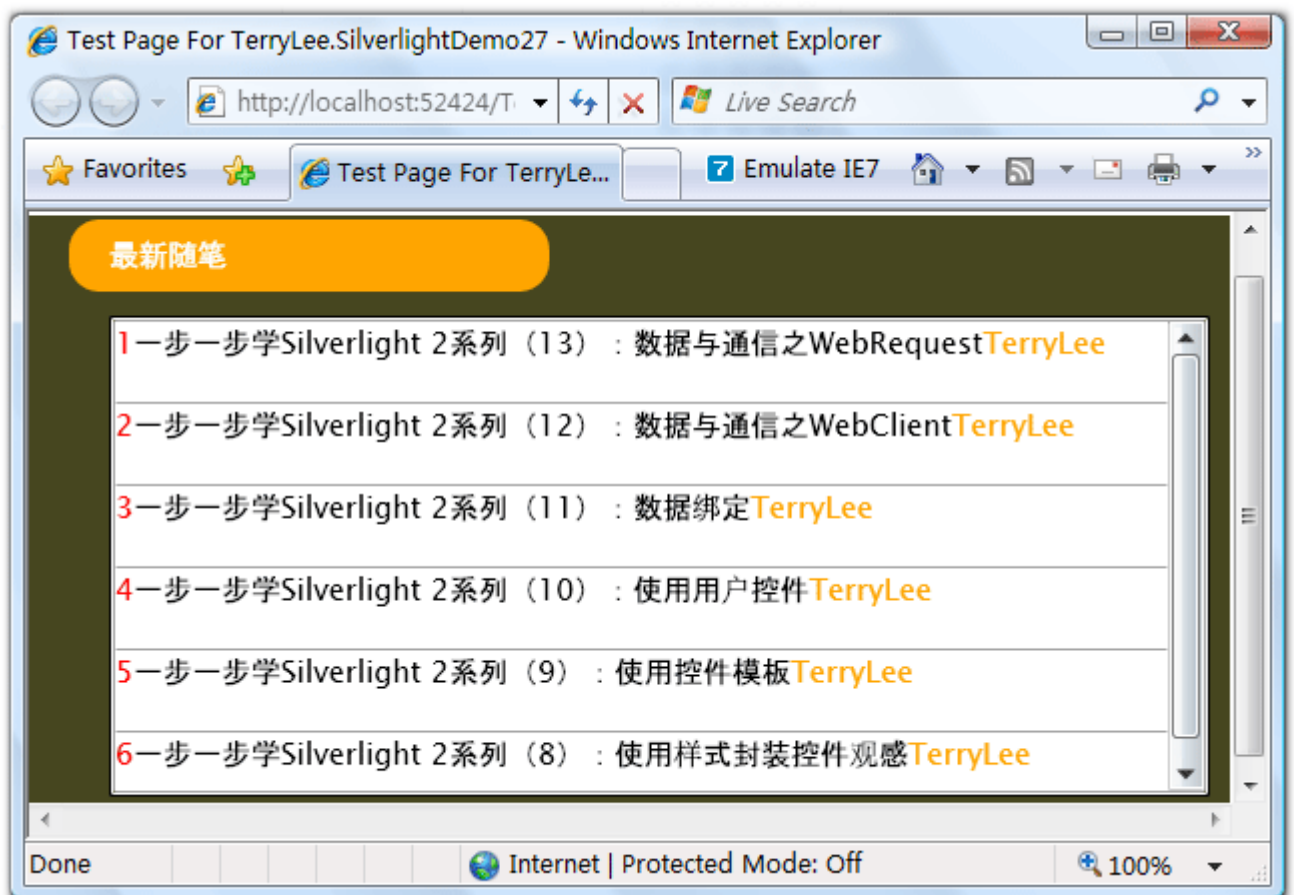
```
void ResponseReady(IAsyncResult asyncResult)
{
    WebRequest request = asyncResult.AsyncState as WebRequest;
    WebResponse response = request.EndGetResponse(asyncResult);

    using (Stream responseStream = response.GetResponseStream())
    {
       DataContractJsonSerializer jsonSerializer =
            new DataContractJsonSerializer(typeof(Blog));

        Blog blog = jsonSerializer.ReadObject(responseStream) as Blog;

        Posts.ItemsSource = blog.Posts;
    }
}
```

至此一个完整的在 Silverlight 2 对于 JSON 的支持示例就完成了。运行后的效果与前面的示例一样：



结束语

本文简单介绍了在 Silverlight 2 中对于 JSON 的支持,DataContractJsonSerializer 用于将对象序列化为 JSON 或者反序列化为对象实例, 你可以从[这里](#)下载本文示例代码。

## 一步一步学 Silverlight 2 系列（17）：数据与通信之 ADO.NET Data Services

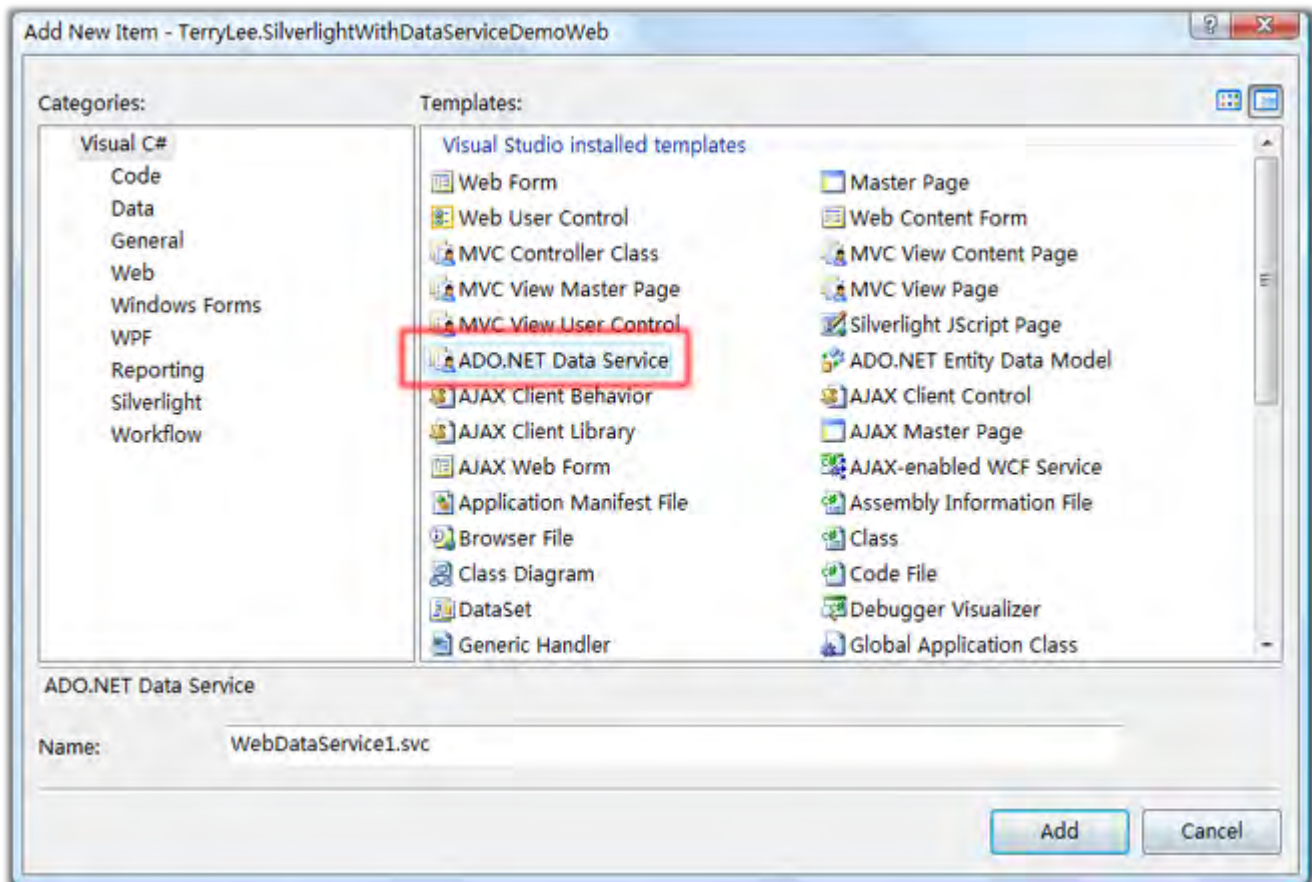
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发。

本文将简单介绍在 Silverlight 2 中如何调用 ADO.NET Data Services。

### 准备知识

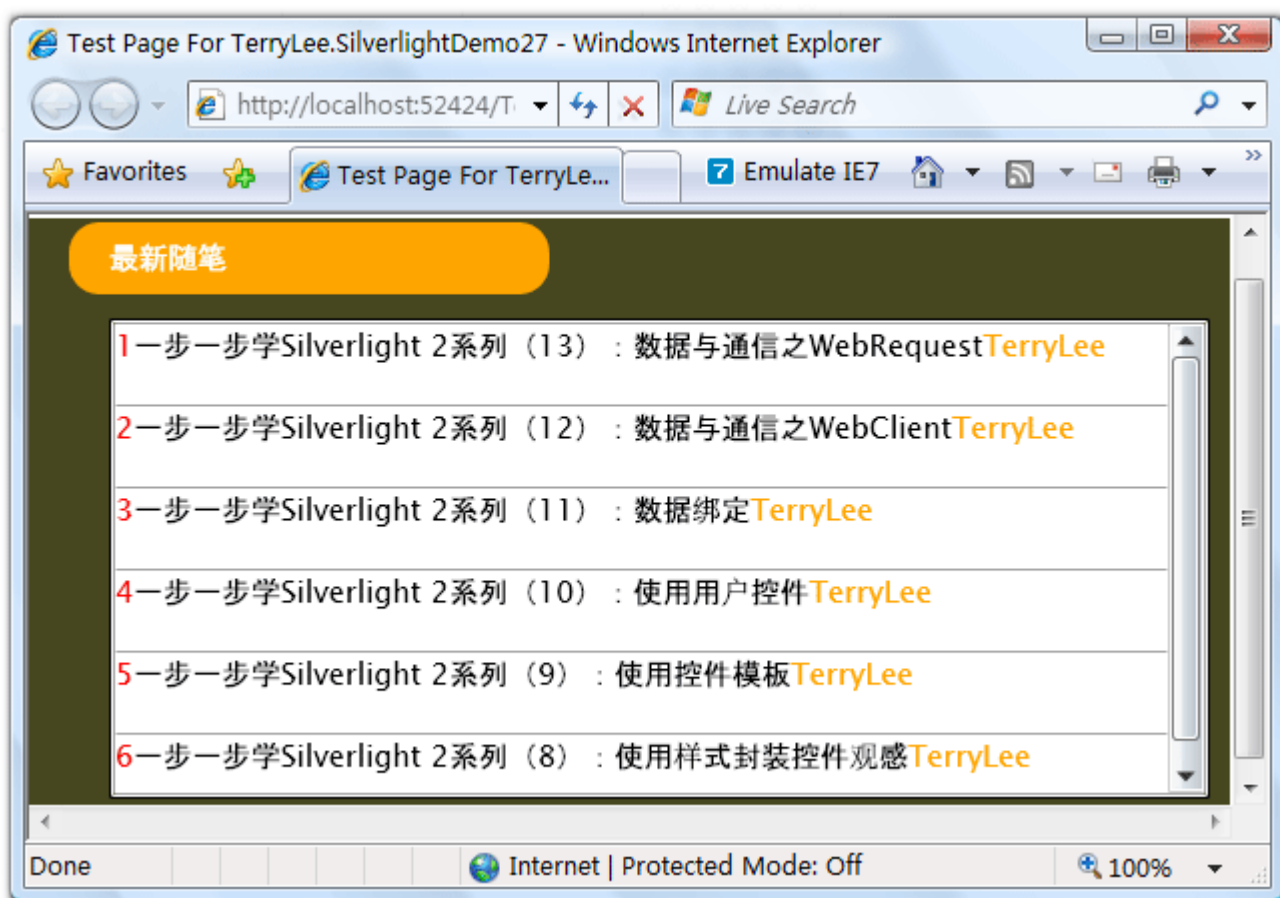
由于 ADO.NET Data Services 是在 ASP.NET 3.5 Extensions 中，所以在开始本文示例之前，首先要安装一下 ASP.NET 3.5 Extensions 最新版本，你可以从[这里](#)下载。安装完成后，在添加新项对话框中应该能够看到 ADO.NET Data Service 项：



ADO.NET Data Service 允许应用程序把数据以服务的形式公开，这样我们就可以通过浏览器来直接访问数据，它支持开放的业界标准，如 AtomPub 和 JSON。它支持标准的 HTTP 动作如 POST、GET、PUT、DELETE，用来完成数据的创建、更新、删除和读取。ADO.NET Data Service 的知识这里不再多说，大家可以去查看相关的资料。

## 简单示例

如果大家看了前面三篇文章的话，可能对于下面的这个界面已经很烦了，不过在本文我会仍然采用这个示例进行演示:)



建立完 Silverlight 2 项目之后，我们在 Web 项目中添加一个 Post 类：

```
public class Post
{
    public int Id { get; set; }
```

```
public string Title { get; set; }

public string Author { get; set; }

}
```

我们用 Id 作为 Post 的主键，这里需要添加对于 Microsoft.Data.Web.dll 程序集的引用，位于<盘符>\Program Files\Reference Assemblies\Microsoft\Framework\ASP.NET 3.5 Extensions 下面，引入命名空间 using Microsoft.Data.Web，并且为 Id 加上[DataWebKey]特性，最终完成后代码应该如下：

```
public class Post
{
    [DataWebKey]
    public int Id { get; set; }

    public string Title { get; set; }

    public string Author { get; set; }
}
```

再添加一个 Blog 类，它有一个返回类型为 IQueryable<Post>的属性 Posts：

```
public class Blog
{
    public Blog()
    {
        _post.Add(new Post { Id = 1, Title = "一步一步学 Silverlight 2 系列 (13) : 数据与通信之 WebRequest", Author = "TerryLee" });
        _post.Add(new Post { Id = 2, Title = "一步一步学 Silverlight 2 系列 (12) : 数据与通信之 WebClient", Author = "TerryLee" });
    }
}
```



```

        _post.Add(new Post { Id = 3, Title = "一步一步学 Silverlight 2 系列 (11) : 数
数据绑定", Author = "TerryLee" });

        _post.Add(new Post { Id = 4, Title = "一步一步学 Silverlight 2 系列 (10) : 使
用用户控件", Author = "TerryLee" });

        _post.Add(new Post { Id = 5, Title = "一步一步学 Silverlight 2 系列 (9) : 使
用控件模板", Author = "TerryLee" });

        _post.Add(new Post { Id = 6, Title = "一步一步学 Silverlight 2 系列 (8) : 使
用样式封装控件观感", Author = "TerryLee" });

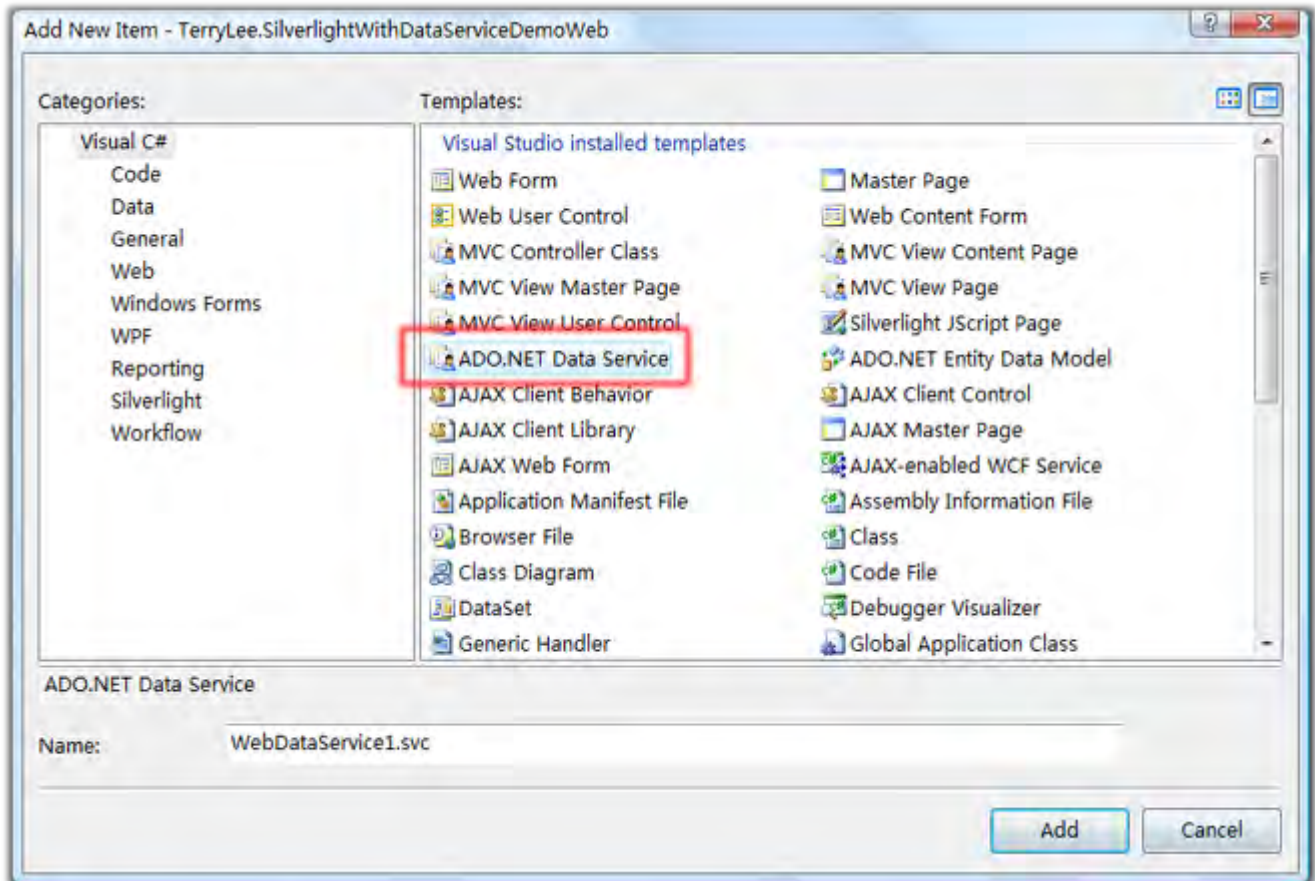
    }

    List<Post> _post = new List<Post>();

    public IQueryable<Post> Posts
    {
        get { return _post.AsQueryable<Post>(); }
    }
}

```

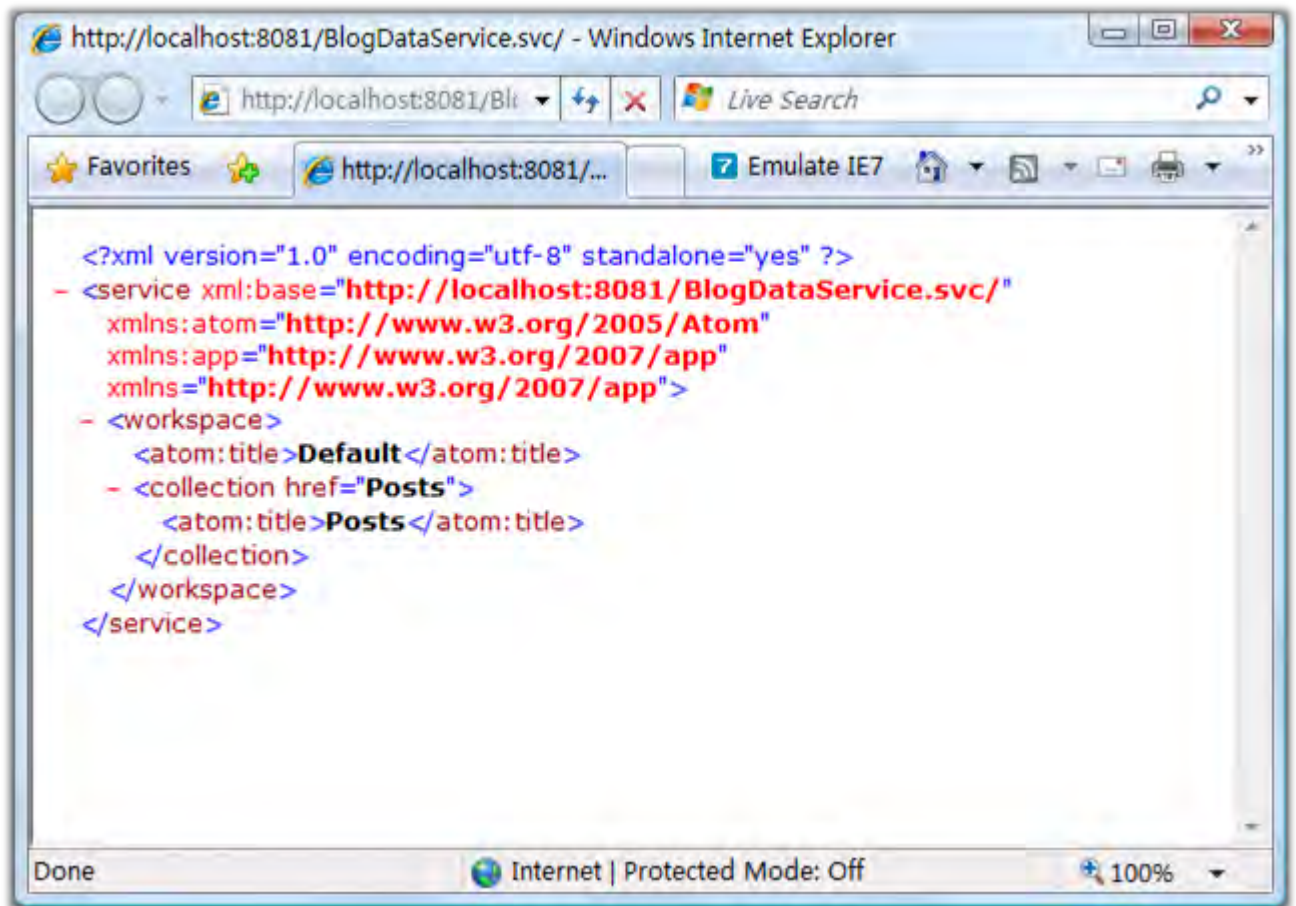
添加一个 ADO.NET Data Service, 取名 BlogDataService.svc:



实现服务，让它继承于泛型的 WebDataService，并且设置访问权限。

```
public class BlogDataService : WebDataService<Blog>
{
    public static void InitializeService(IWebDataServiceConfiguration config)
    {
        config.SetResourceContainerAccessRule("*", ResourceContainerRights.AllRead);
    }
}
```

现在我们的服务端就完成了，现在我们可以浏览器中访问 BlogDataService.svc，应该可以看到如下界面：



现在还看不到所有的 Posts，我们可以在地址栏中输入 <http://localhost:8081/BlogDataService.svc/Posts>，浏览器会默认为 Feed 打开，可以查看源代码，将会看到所有内容，XML 内容如下（只列出片段）：

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="http://localhost:8081/BlogDataService.svc/" .....>
  <id>http://localhost:8081/BlogDataService.svc/Posts</id>
  <updated />
  <title>Posts</title>
  <link rel="self" href="Posts" title="Posts" />
  <entry adsm:type="TerryLee.SilverlightWithDataServiceDemoWeb.Post">
    <id>http://localhost:8081/BlogDataService.svc/Posts(1)</id>
    <updated />
    <title />
    <author>
```

```
<name />

</author>

<link rel="edit" href="Posts(1)" title="Post" />

<content type="application/xml">

  <ads:Id adsm:type="Int32">1</ads:Id>

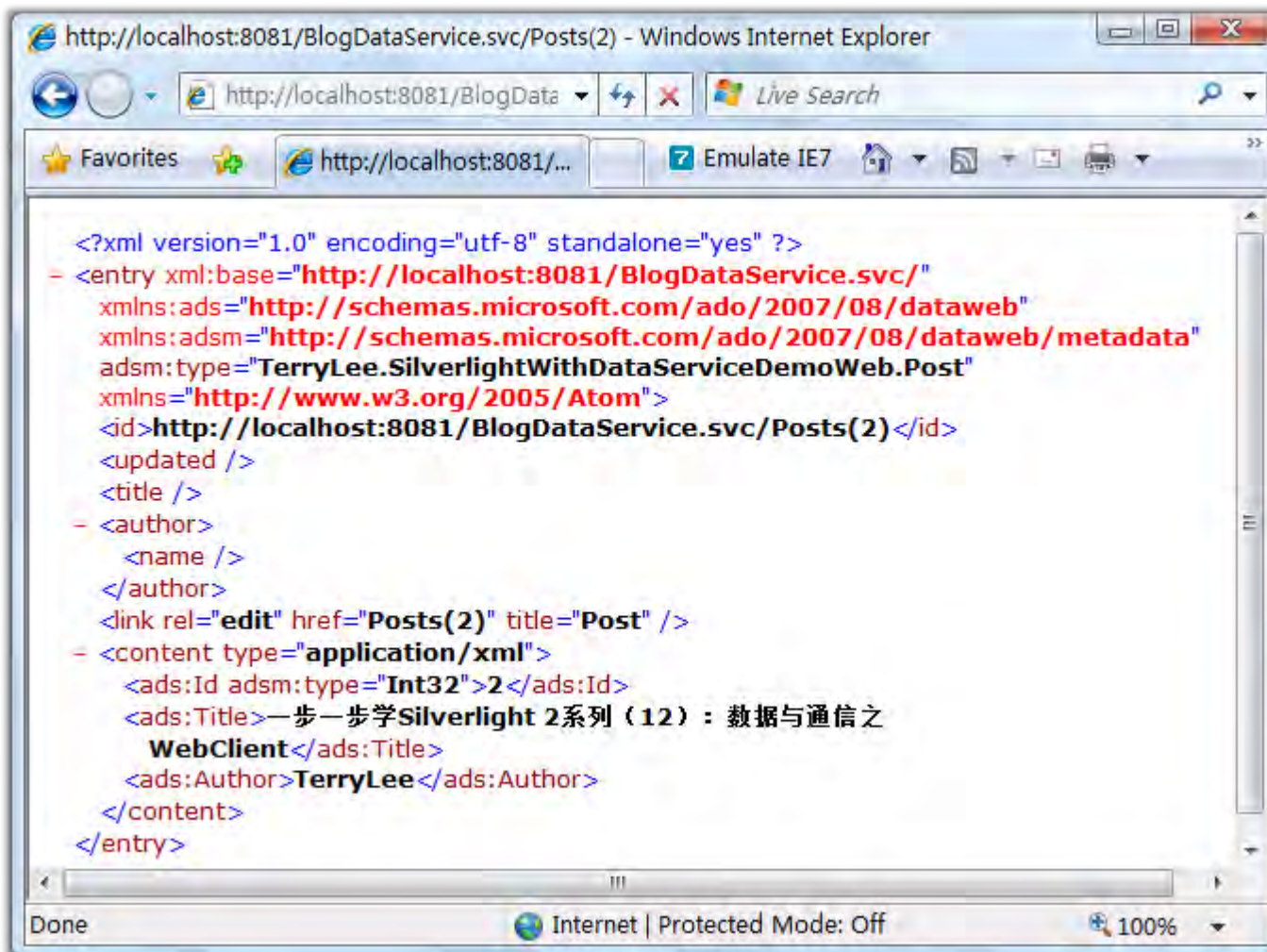
  <ads:Title>一步一步学 Silverlight 2 系列 (13) : 数据与通信之 WebRequest</ads:Ti
tle>

  <ads:Author>TerryLee</ads:Author>

</content>

</entry>
```

如果要查看某一条文章的内容，可以输入 [http://localhost:8081/BlogDataService.svc/Posts\(2\)](http://localhost:8081/BlogDataService.svc/Posts(2)) 进行查看，如下图所示。



当然还可以进行其他的查询，使用 filter 和 orderby 等，如 [http://localhost:8081/BlogDataService.svc/Posts?\\$filter=Id eq 1&\\$orderby=Id](http://localhost:8081/BlogDataService.svc/Posts?$filter=Id eq 1&$orderby=Id)，这里不在介绍。至此我们的数据服务端就算完成了。下面再实现客户端，XAML 不再贴出来，大家可以参考前面的几篇文章，使用 WebClient 获取数据，返回的结果是一个 XML 文件：

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
    Uri uri = new Uri("http://localhost:8081/BlogDataService.svc/Posts");
    WebClient client = new WebClient();

    client.OpenReadCompleted += new OpenReadCompletedEventHandler(client_OpenReadCompleted);

    client.OpenReadAsync(uri);
}

void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    if (e.Error == null)
    {
    }
}
}
```

我们可以使用 LINQ to XML 进行数据的读取，在 Silverlight 项目中建立一个 Post 类，跟上面的 Post 类一样，然后使用 LINQ to XML 读取：

```
XmlReader reader = XmlReader.Create(e.Result);

XDocument postdoc = XDocument.Load(reader);

XNamespace xmlns = "http://www.w3.org/2005/Atom";
```

```

XNamespace ads = "http://schemas.microsoft.com/ado/2007/08/dataweb";

var posts = from x in postdoc.Descendants(xmlns + "entry")
            select new Post
            {
                Id = int.Parse(x.Descendants(ads + "Id").First().Value),
                Title = x.Descendants(ads + "Title").First().Value,
                Author = x.Descendants(ads + "Author").First().Value
            };

Posts.ItemsSource = posts;

```

完成的代码如下所示:

```

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
    Uri uri = new Uri("http://localhost:8081/BlogDataService.svc/Posts");
    WebClient client = new WebClient();

    client.OpenReadCompleted += new OpenReadCompletedEventHandler(client_OpenReadCompleted);

    client.OpenReadAsync(uri);
}

void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    if (e.Error == null)
    {
        XmlReader reader = XmlReader.Create(e.Result);
    }
}

```

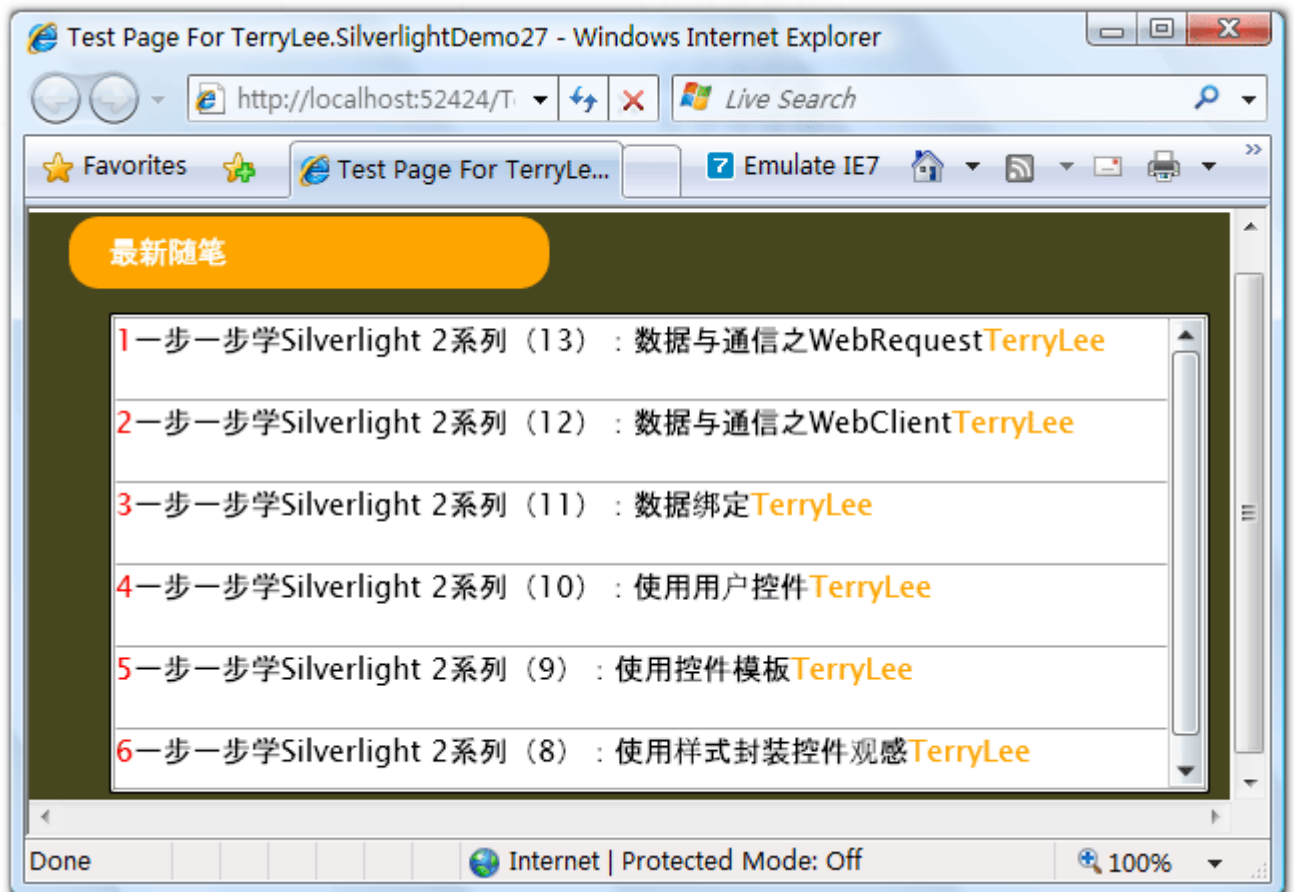
```
XDocument postdoc = XDocument.Load(reader);

XNamespace xmlns = "http://www.w3.org/2005/Atom";
XNamespace ads = "http://schemas.microsoft.com/ado/2007/08/dataweb";

var posts = from x in postdoc.Descendants(xmlns + "entry")
            select new Post
            {
                Id = int.Parse(x.Descendants(ads + "Id").First().Value),
                Title = x.Descendants(ads + "Title").First().Value,
                Author = x.Descendants(ads + "Author").First().Value
            };

Posts.ItemsSource = posts;
}
}
```

完整的示例就到这里了，运行后的结果与前面的一样。



## 结束语

本文简单介绍了在 Silverlight 2 调用 ADO.NET Data Services，由于对 ADO.NET Data Services 了解不多，有错误的地方还请大家斧正，你可以从[这里](#)下载示例代码。



## 一步一步学 Silverlight 2 系列（18）：综合实例之 RSS 阅读器

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了许多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发

本文将综合前面十七篇讲过的界面布局、样式、控件模板、数据绑定、网络通信等几个方面，来开发一个综合实例——简易 RSS 阅读器。

### 界面布局

我们最终完成的 RSS 阅读器界面如下：



定义一个三行两列的 Grid，分别放置顶部信息、分割线和下面的内容区：

```
<Grid.RowDefinitions>
    <RowDefinition Height="50"></RowDefinition>
    <RowDefinition Height="20"></RowDefinition>
    <RowDefinition Height="*"></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="240"></ColumnDefinition>
    <ColumnDefinition Width="*"></ColumnDefinition>
</Grid.ColumnDefinitions>
```

设计顶部输入区域，对 Grid 第一行做合并，并且放置一个 StackPanel：

```
<StackPanel x:Name="Header" Orientation="Horizontal"
    Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2">
    <Image Source="Rss.png" Width="32" Height="32" Margin="10 0 10 0"></Image>
    <Border Style="{StaticResource titleBorder}">
        <TextBlock Text="基于 Silverlight 的 RSS 阅读器" Foreground="#FFFFFF"
            VerticalAlignment="Center" Margin="12 0 0 0"></TextBlock>
    </Border>
    <WatermarkedTextBox x:Name="feedAddress" Width="300" Height="35"
        FontSize="16" Margin="10 0 10 0">
        <WatermarkedTextBox.Watermark>
            <TextBlock Text="请输入有效的 RSS 地址" VerticalAlignment="Center"
                Foreground="#FBA430" FontSize="16"></TextBlock>
        </WatermarkedTextBox.Watermark>
    </WatermarkedTextBox>
    <Button x:Name="displayButton" Style="{StaticResource button}"
        Content="显示" Click="displayButton_Click"></Button>
```

```
<Button x:Name="fullScreenButton" Style="{StaticResource button}"
        Content="全屏" Click="fullScreenButton_Click"></Button>
</StackPanel>
```

鉴于两个按钮的风格一致，在 App.xaml 中定义一个 button 样式：

```
<Style x:Key="button" TargetType="Button">
    <Setter Property="Width" Value="100"></Setter>
    <Setter Property="Height" Value="35"></Setter>
    <Setter Property="Background" Value="#FBA430"></Setter>
    <Setter Property="Foreground" Value="#FBA430"></Setter>
    <Setter Property="FontSize" Value="16"></Setter>
</Style>
<Style x:Key="titleBorder" TargetType="Border">
    <Setter Property="CornerRadius" Value="10"></Setter>
    <Setter Property="Width" Value="220"></Setter>
    <Setter Property="Height" Value="40"></Setter>
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0">
                <GradientStop Color="#FBA430" Offset="0.0" />
                <GradientStop Color="#FEF4E7" Offset="0.5" />
                <GradientStop Color="#FBA430" Offset="1.0" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>
```

定义分割线，用 Rectangle 来表示：

```

<StackPanel Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2" VerticalAlignment="Center">
    <Rectangle Style="{StaticResource rectangle}"/>
</StackPanel>

```

为了显示出渐变的样式，我们定义样式如下：

```

<Style x:Key="rectangle" TargetType="Rectangle">
    <Setter Property="Width" Value="780"></Setter>
    <Setter Property="Height" Value="5"></Setter>
    <Setter Property="RadiusX" Value="3"></Setter>
    <Setter Property="RadiusY" Value="3"></Setter>
    <Setter Property="Fill">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0">
                <GradientStop Color="#FEF4E7" Offset="0.0" />
                <GradientStop Color="#FBA430" Offset="1.0" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>

```

定义左边的列表区，用 ListBox 来显示，并且定义 ItemTemplate：

```

<ListBox x:Name="PostsList" Grid.Column="0" Grid.Row="2"
    Margin="10 5 5 10" SelectionChanged="PostsList_SelectionChanged">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Title.Text}"
                TextWrapping="Wrap" Width="200"/>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>

```

```
</ListBox.ItemTemplate>
</ListBox>
```

最后定义右边的详细信息区域，在 StackPanel 中垂直放置三个 Border：

```
<StackPanel x:Name="Detail" Grid.Column="1" Grid.Row="2">
  <Border CornerRadius="10" Background="#CDFCAE" Margin="10 5 10 10"
    Width="540" Height="40">
    <TextBlock Text="{Binding Title.Text}" TextWrapping="Wrap"
      VerticalAlignment="Center" Foreground="Red"/>
  </Border>
  <Border CornerRadius="10" Background="#CDFCAE" Margin="10 5 10 10"
    Width="540" Height="300">
    <TextBlock Text="{Binding Summary.Text}" TextWrapping="Wrap"/>
  </Border>
  <Border CornerRadius="10" Background="#CDFCAE" Margin="10 5 10 10"
    Width="540" Height="40">
    <StackPanel Orientation="Horizontal">
      <TextBlock Text="评论日期: " TextWrapping="Wrap"
        Foreground="Red" VerticalAlignment="Center"/>
      <TextBlock Text="{Binding PublishDate}" TextWrapping="Wrap"
        Foreground="Red" VerticalAlignment="Center"/>
    </StackPanel>
  </Border>
</StackPanel>
```

界面布局到此大功告成。

## 实现功能

下面实现数据的获取，采用 WebRequest 来实现，也可以使用其他方式。

```

/// <summary>
/// 显示列表
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void displayButton_Click(object sender, RoutedEventArgs e)
{
    Uri uri = new Uri(feedAddress.Text);
    WebRequest request = (WebRequest)WebRequest.Create(uri);
    request.BeginGetResponse(new AsyncCallback(responseReady), request);
}

void responseReady(IAsyncResult asyncResult)
{
    WebRequest request = (WebRequest)asyncResult.AsyncState;
    WebResponse response = (WebResponse)request.EndGetResponse(asyncResult);

    XmlReader reader = XmlReader.Create(response.GetResponseStream());
    SyndicationFeed feed = SyndicationFeed.Load(reader);

    PostsList.ItemsSource = feed.Items;
}

```

显示详细信息:

```

/// <summary>
/// 查看详细信息
/// </summary>
/// <param name="sender"></param>

```

```
/// <param name="e"></param>
private void PostsList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    SyndicationItem item = PostsList.SelectedItem as SyndicationItem;

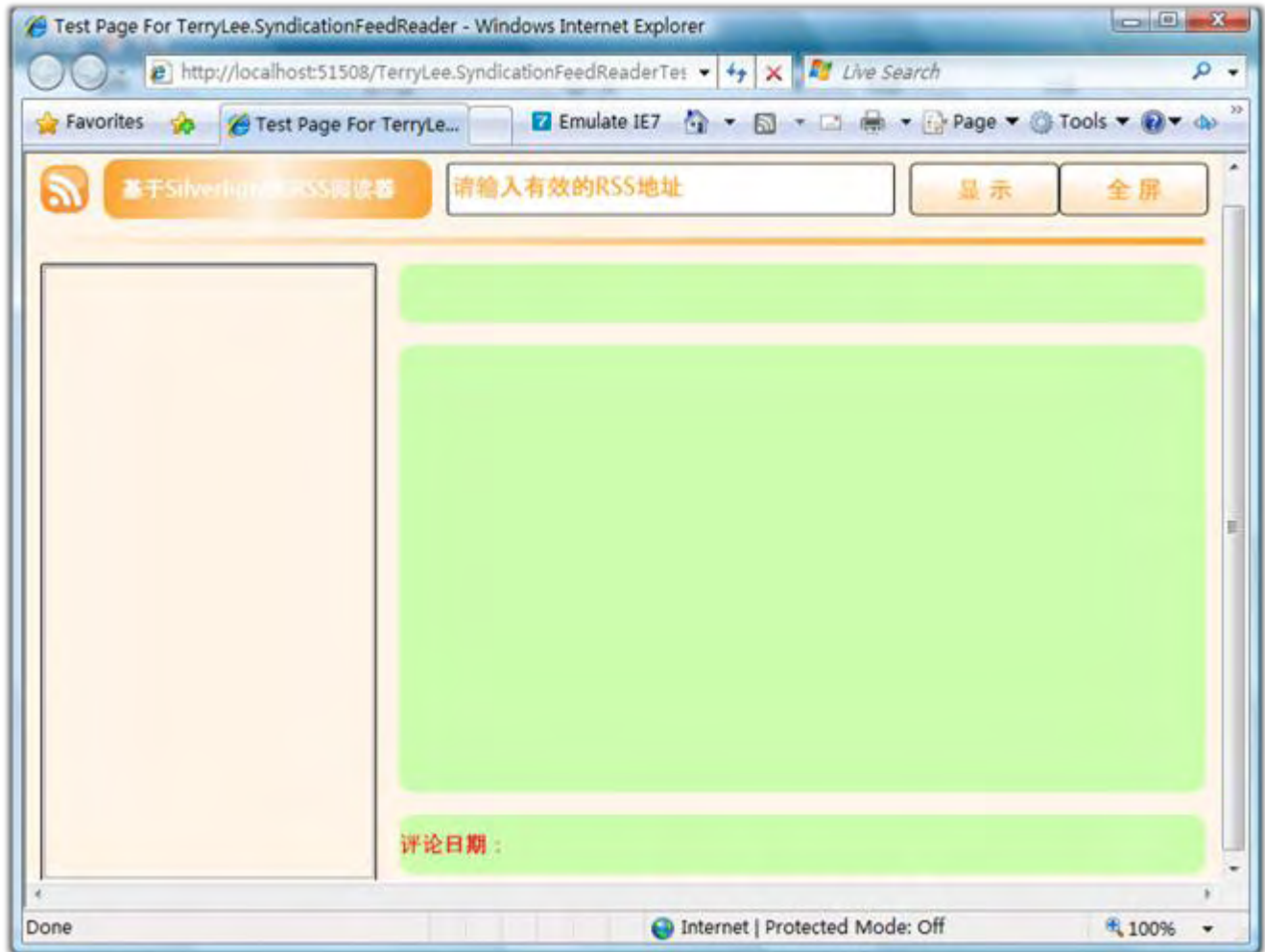
    Detail.DataContext = item;
}
```

实现全屏按钮的代码：

```
/// <summary>
/// 全屏显示
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void fullScreenButton_Click(object sender, RoutedEventArgs e)
{
    Content contentObject = Application.Current.Host.Content;
    contentObject.IsFullScreen = !contentObject.IsFullScreen;
}
```

## 运行效果

运行后界面如下：



输入豆瓣的最新影评 Feed:





选择其中一项后，将显示出详细信息：



## 结束语

本文对前面十七篇内容做了一个小结, 并开发出了一个简易 RSS 阅读器, 你可以从[这里](#)下载本文示例代码。

## 一步一步学 Silverlight 2 系列（19）：如何在 Silverlight 中与 HTML DOM 交互（上）

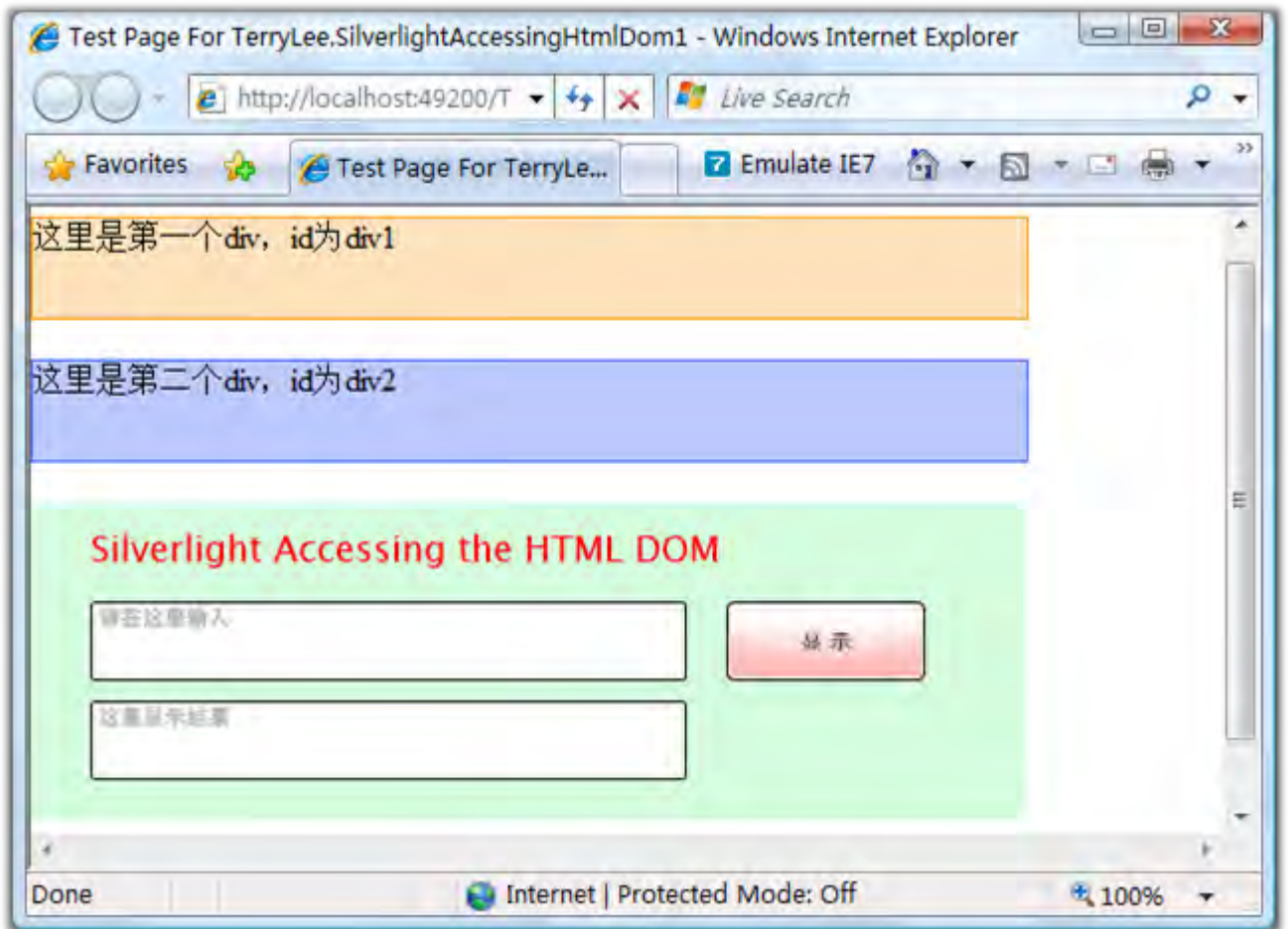
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython，对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发。

Silverlight 中内置了对于 HTML、客户端脚本等的支持，本文为如何在 Silverlight 2 与 HTML DOM 进行交互第一部分，访问和修改 DOM 元素。

### 访问 DOM 元素

我们先来看一个简单的示例，如何访问 HTML DOM。最终完成的效果如下，我们将在界面放置两个 div，分别为 div1 和 div2，下面绿色的区域为 Silverlight 部分，在第一个文本框中输入 div 的 id 并点击显示，将在下面显示出对应 div 上的文本信息。



首先我们需要对测试页做一下修改，因为默认的 Silverlight 插件所占的高度是 100%，修改为 200px。

```
<div style="height:200px">  
    <asp:Silverlight ID="Xaml1" runat="server"  
        Source="~/ClientBin/TerryLee.SilverlightAccessingHtmlDom1.xap"  
        Version="2.0" Width="100%" Height="200px" />  
</div>
```

同时放置两个 div:

```
<div id="div1">这里是第一个 div, id 为 div1</div>  
<div id="div2">这里是第二个 div, id 为 div2</div>
```

为了看起来明显起见，给它们定义简单的样式:

```
#div1
```

```

{
    background:#FCE2BC;
    border:solid 1px #FF9900;
    width:500px;
    height:50px;
    margin-bottom:20px;
}
#div2
{
    background:#BCC8FC;
    border:solid 1px #4769F9;
    width:500px;
    height:50px;
    margin-bottom:20px;
}

```

实现 Silverlight 的界面布局，使用 Canvas，给它的背景定义为浅绿色，XAML 如下：

```

<Canvas Background="#D5FCDF">
    <TextBlock Text="Silverlight Accessing the HTML DOM" Foreground="Red"
        Canvas.Top="10" Canvas.Left="30" FontSize="18">
    </TextBlock>
    <WatermarkedTextBox x:Name="input" Watermark="请在这里输入"
        Height="40" Width="300"
        Canvas.Left="30" Canvas.Top="50">
    </WatermarkedTextBox>
    <WatermarkedTextBox x:Name="result" Watermark="这里显示结果"
        Height="40" Width="300"
        Canvas.Left="30" Canvas.Top="100">

```

```
</WatermarkedTextBox>

<Button x:Name="displayButton" Background="Red"
        Height="40" Width="100" Content="显示"
        Canvas.Top="50" Canvas.Left="350"
        Click="displayButton_Click">

</Button>

</Canvas>
```

实现对 HTML DOM 的访问。Silverlight 2 在命名空间 System.Windows.Browser 下内置了很多对于 HTML DOM 访问和操作的支持，我们最常用的一个对象是 HtmlElement，通过 HtmlPage 静态类可以获取到当前页面的文档模型，最后再调用 GetElementsByTagName 或者 GetElementById 方法。

```
HtmlElement element = HtmlPage.Document.GetElementById(this.input.Text);
```

这样我们就获取到了一个 DOM 元素，再使用它的 GetAttribute 可以获取到相关属性：

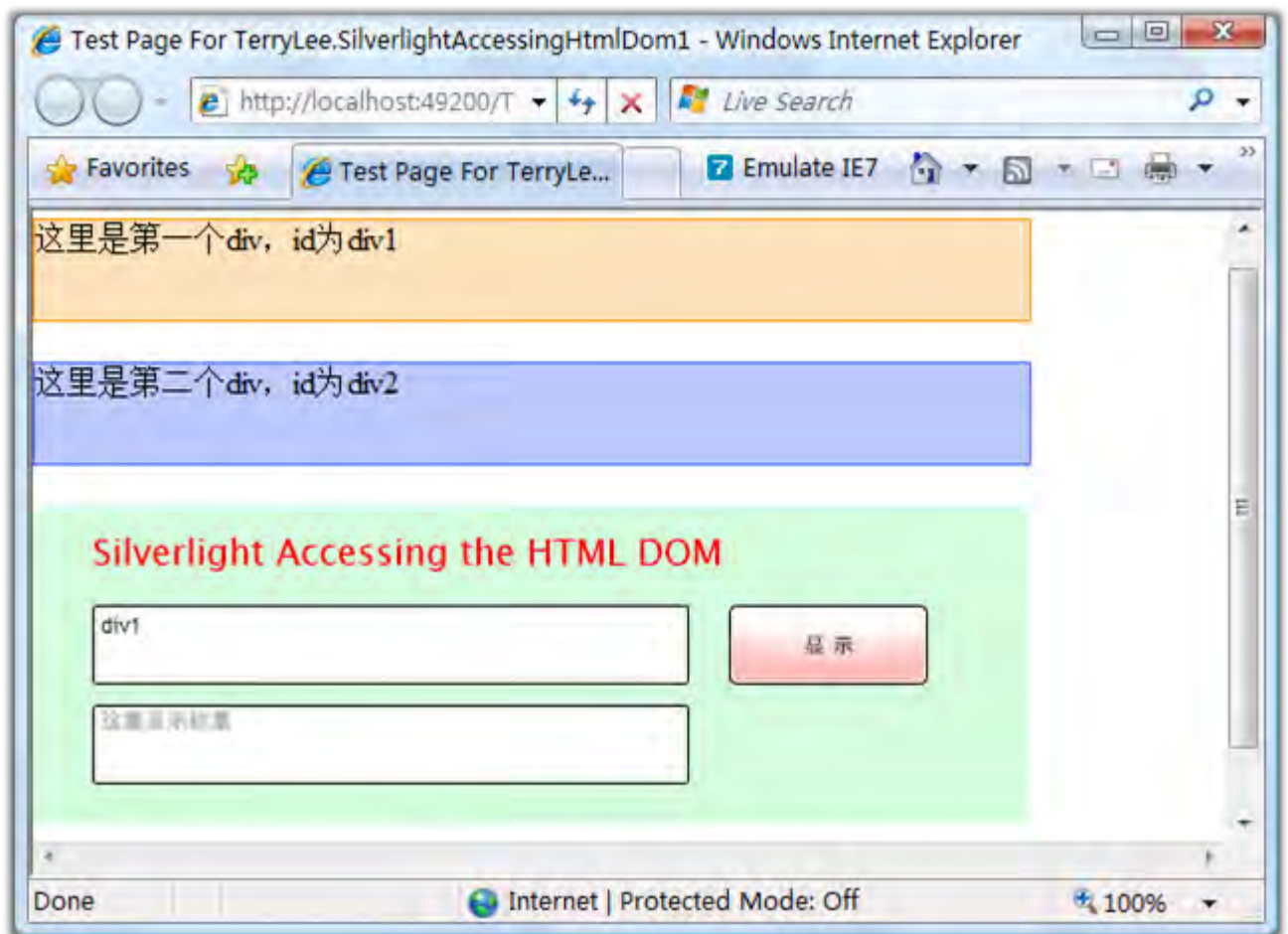
```
this.result.Text = element.GetAttribute("innerText");
```

完整的代码如下：

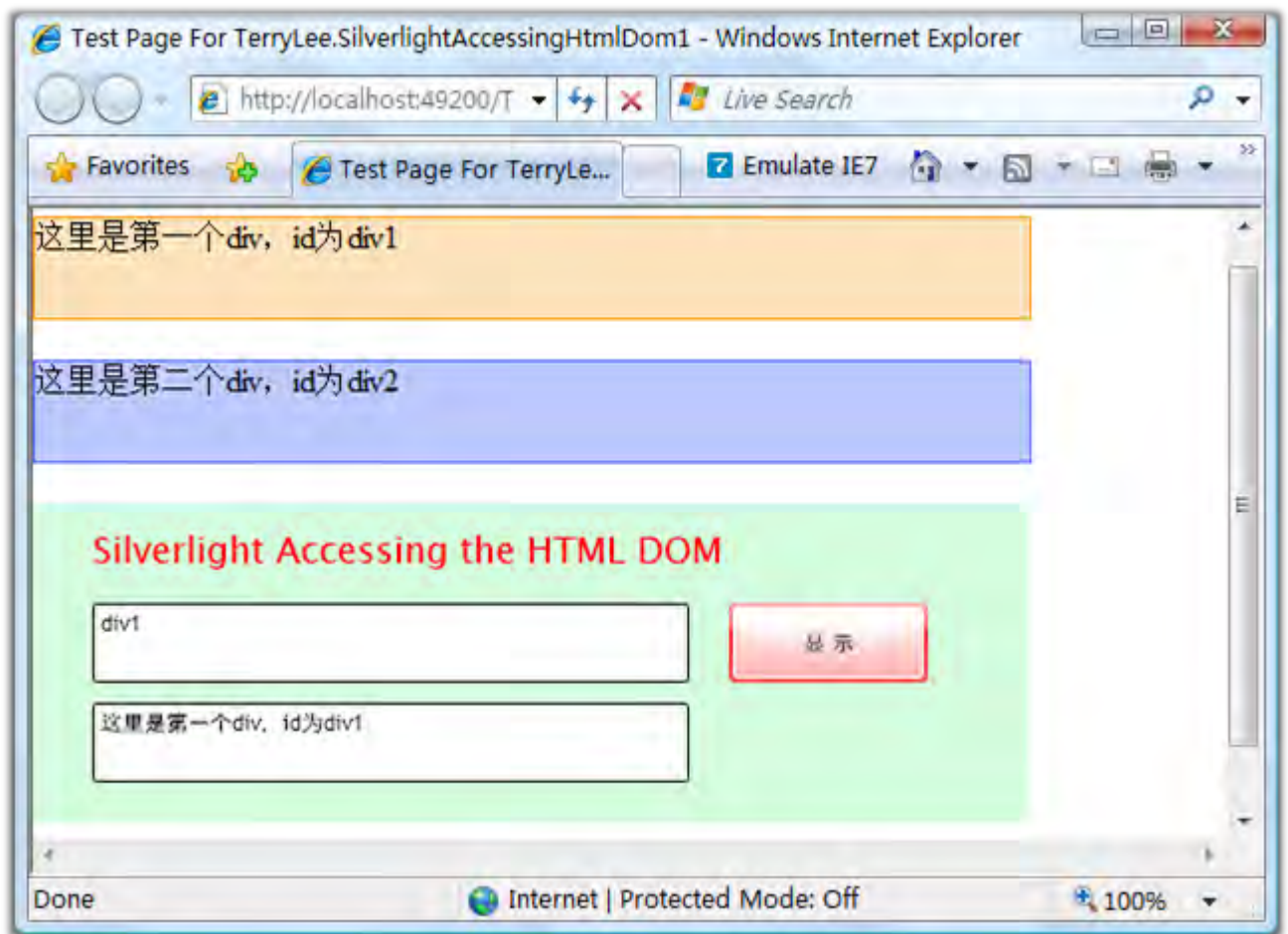
```
private void displayButton_Click(object sender, RoutedEventArgs e)
{
    HtmlElement element = HtmlPage.Document.GetElementById(this.input.Text);

    this.result.Text = element.GetAttribute("innerText");
}
```

运行后我们在第一个文本框中输入 div1：



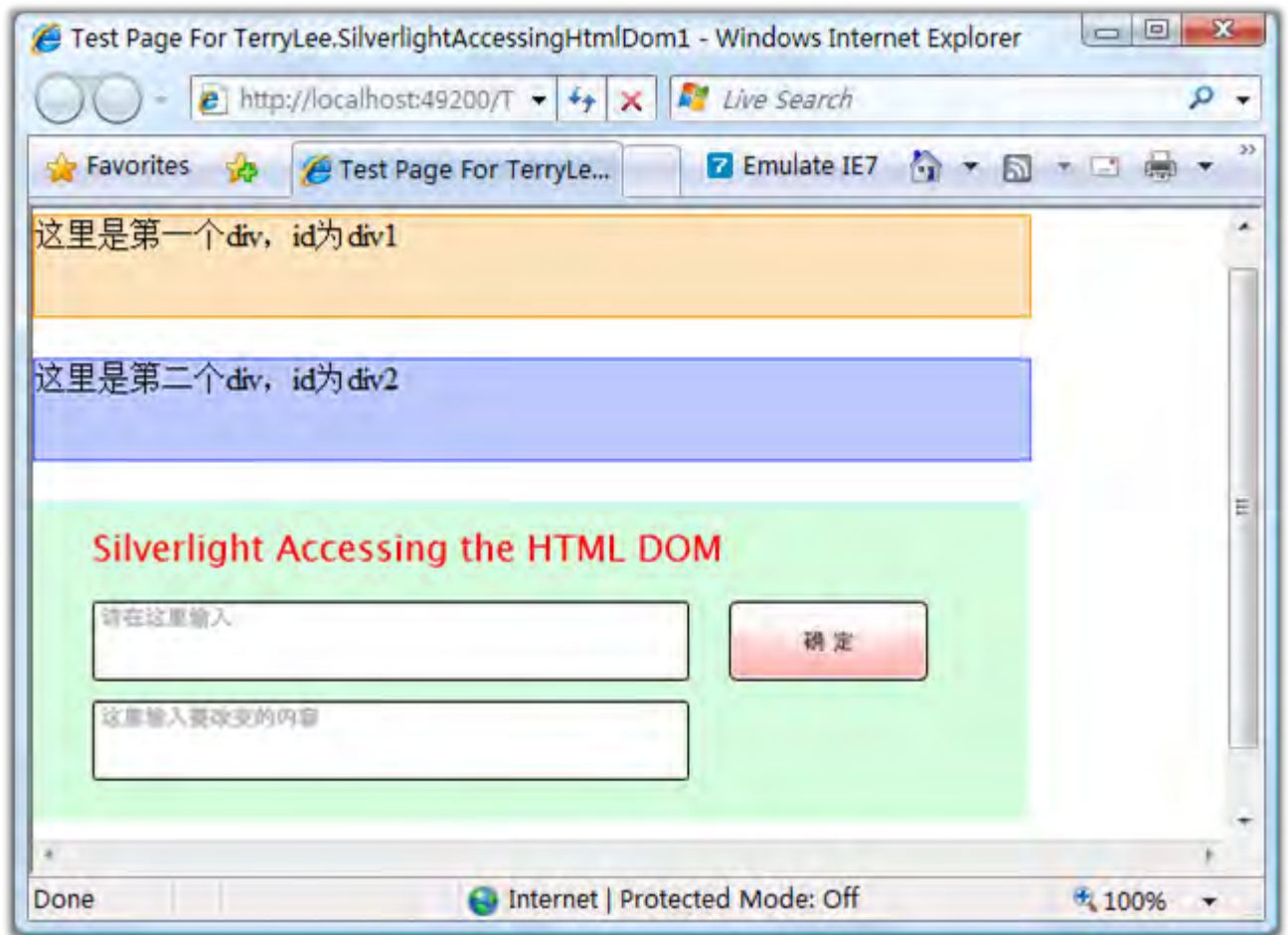
单击显示后，将在第二个文本框中显示出 div1 的文本内容：



## 操作 **DOM** 元素

通过上面的示例我们已经知道了如何去访问 HTML DOM，现在再看一下对 HTML DOM 进行操作，如我们如何改变 DOM 的内容等，还是使用上面的示例，稍作一下修改，在第一个文本框中输入 DOM 的 id，在第二个文本框中输入要修改的内容。



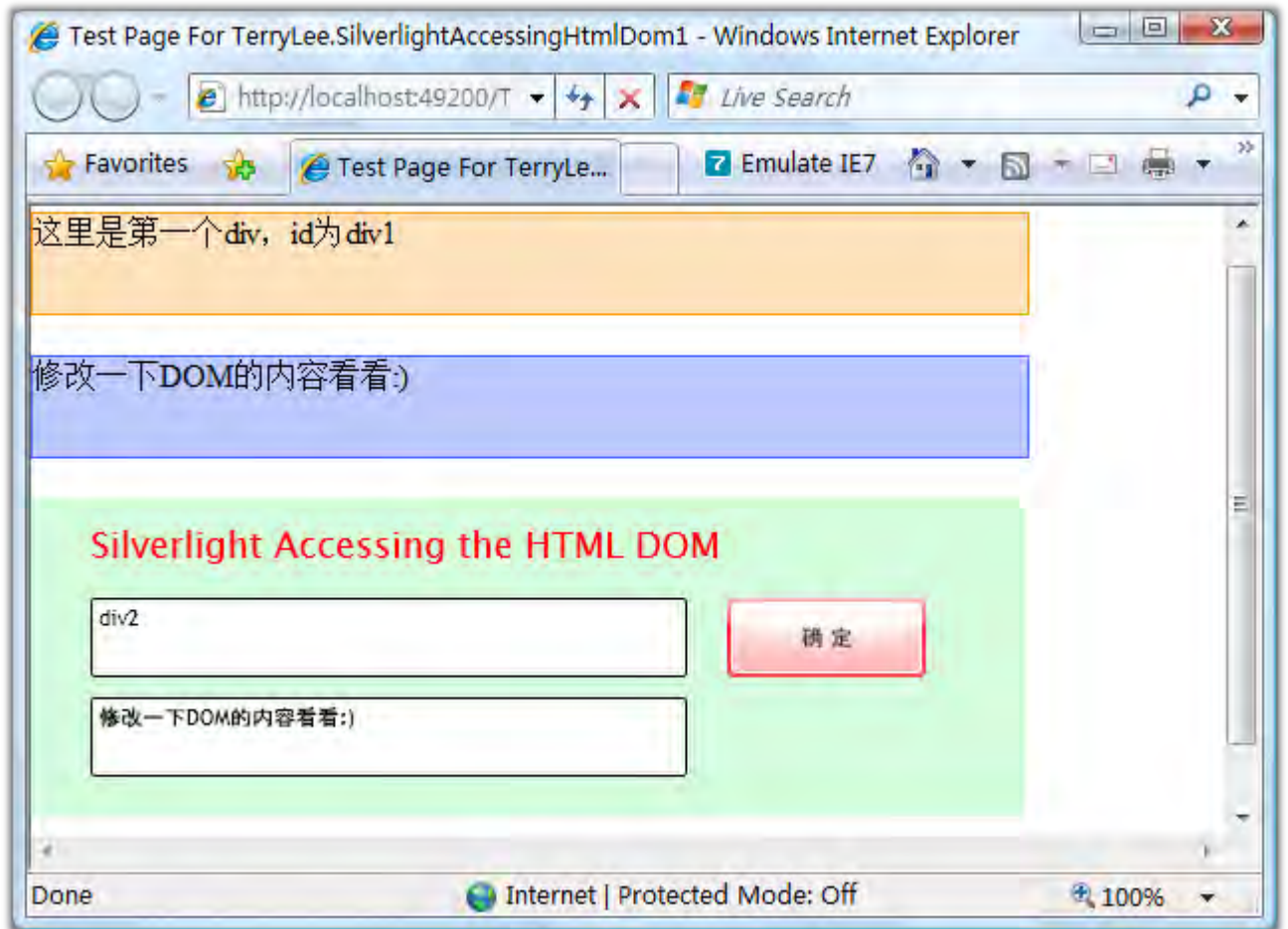


其实这里非常简单，只要对代码做一下小的改动，使用 `SetAttribute` 方法就可以了，第一个参数指定属性名，第二个参数指定属性值。如：

```
private void displayButton_Click(object sender, RoutedEventArgs e)
{
    HtmlElement element = HtmlPage.Document.GetElementById(this.input.Text);

    element.SetAttribute("innerText", this.result.Text);
}
```

运行后，输入 `div2` 和一些内容，单击确定：



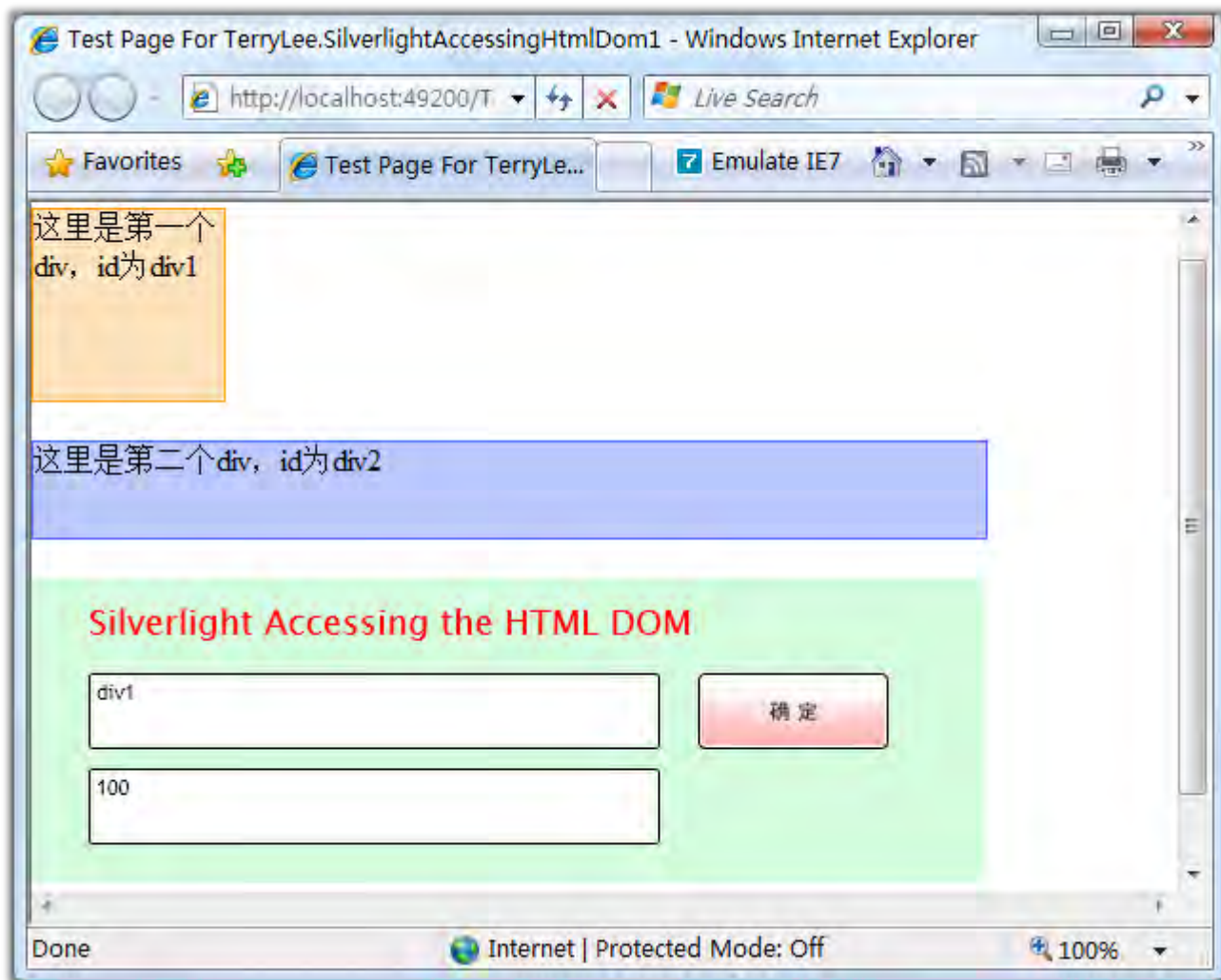
## 修改 DOM 元素样式

除了上面提到的 `GetAttribute` 和 `SetAttribute` 这一组方法之外, `HtmlElement` 还提供了一组 `GetStyleAttribute` 和 `SetStyleAttribute` 方法, 用于获取和设置 DOM 的样式, 如:

```
private void displayButton_Click(object sender, RoutedEventArgs e)
{
    HtmlElement element = HtmlPage.Document.GetElementById(this.input.Text);

    element.SetStyleAttribute("width", this.result.Text);
    element.SetStyleAttribute("height", this.result.Text);
}
```

运行后, 输入 `div1` 和 `100`, 效果如下:



## 结束语

本文介绍了如何在 Silverlight 中访问 DOM 以及修改 DOM 的属性等，下一篇我将介绍如何改变 DOM 的结构，如添加和移除 DOM 元素以及为 DOM 元素注册事件等。

## 一步一步学 Silverlight 2 系列（20）：如何在 Silverlight 中与 HTML DOM 交互（下）

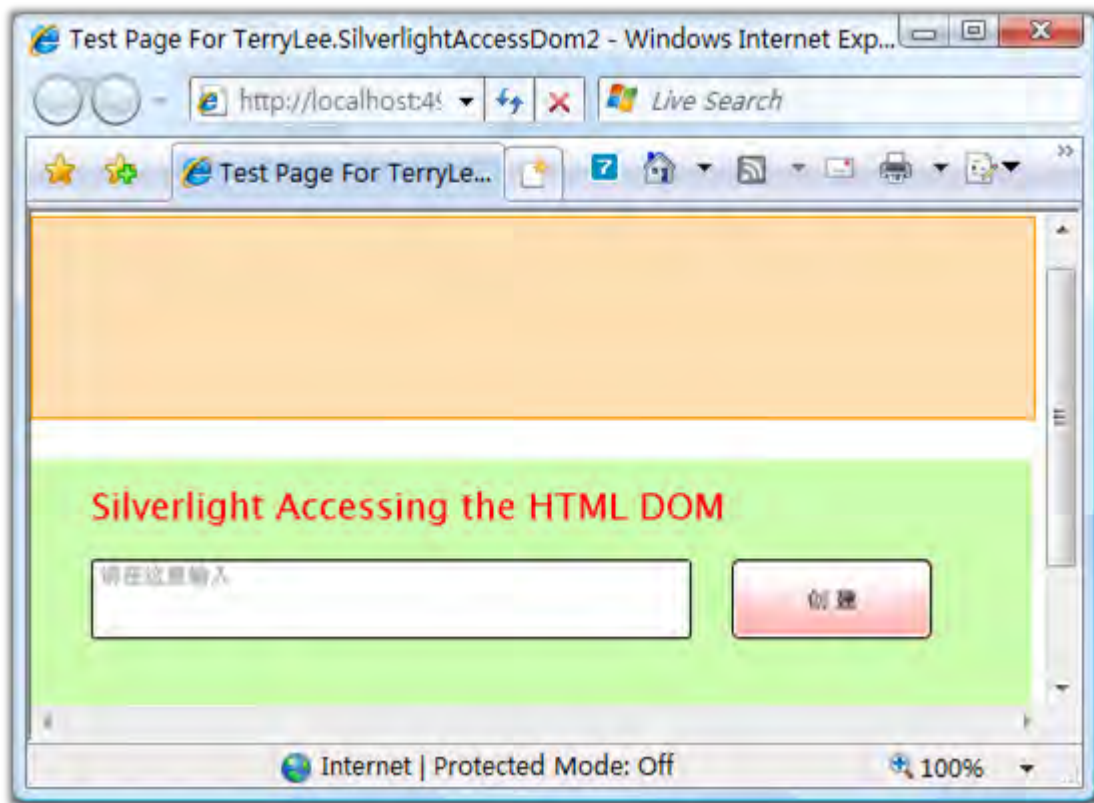
### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython，对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发。

Silverlight 中内置了对于 HTML、客户端脚本等的支持，本文为如何在 Silverlight 2 中与 HTML DOM 交互第二部分。在第一部分中主要介绍了如何访问和修改已有的 HTML DOM，我们还可以完全创建一个新的 DOM 元素或者移除一个已有的 DOM 元素，除此之外，我们还可以为 DOM 元素添加事件处理。

### 创建 DOM 元素

首先我们来看如何创建一个新的 DOM 元素，最终的效果如下，当我们在文本框中输入文字后，单击创建，将在上面的区域中创建一个 li 元素。



先来定义一下 HTML 页面，甚至 Silverlight 插件的高度。

```
<div id="parentdiv">
    <ul id="list">
    </ul>
</div>
<div style="height:200px;">
    <asp:Silverlight ID="Xaml1" runat="server"
        Source="~/ClientBin/TerryLee.SilverlightAccessDom2.xap"
        Version="2.0" Width="100%" Height="200px" />
</div>
```

并且为上面的 div 定义一个简单的样式，以示与 Silverlight 区分

```
#parentdiv
```

```
{  
  
    background:#FCDFB3;  
  
    border:solid 1px #FF9900;  
  
    width:500px;  
  
    height:100px;  
  
    margin-bottom:20px;  
  
}
```

在 Silverlight 中进行界面布局，XAML 如下：

```
<Canvas Background="#CDFCAE">  
    <TextBlock Text="Silverlight Accessing the HTML DOM" Foreground="Red"  
        Canvas.Top="10" Canvas.Left="30" FontSize="18">  
    </TextBlock>  
    <WatermarkedTextBox x:Name="input" Watermark="请在这里输入"  
        Height="40" Width="300"  
        Canvas.Left="30" Canvas.Top="50">  
    </WatermarkedTextBox>  
    <Button x:Name="createButton" Background="Red"  
        Height="40" Width="100" Content="创建"  
        Canvas.Top="50" Canvas.Left="350"  
        Click="createButton_Click">  
    </Button>  
</Canvas>
```

编写创建按钮的代码，首先获取到要往里面添加元素的父元素，即我们定义的 ul：

```
HtmlElement parent = HtmlPage.Document.GetElementById("list");
```

创建一个新的元素 li，并设置属性

```
HtmlElement child = HtmlPage.Document.CreateElement("li");  
child.SetAttribute("innerText", this.input.Text);
```

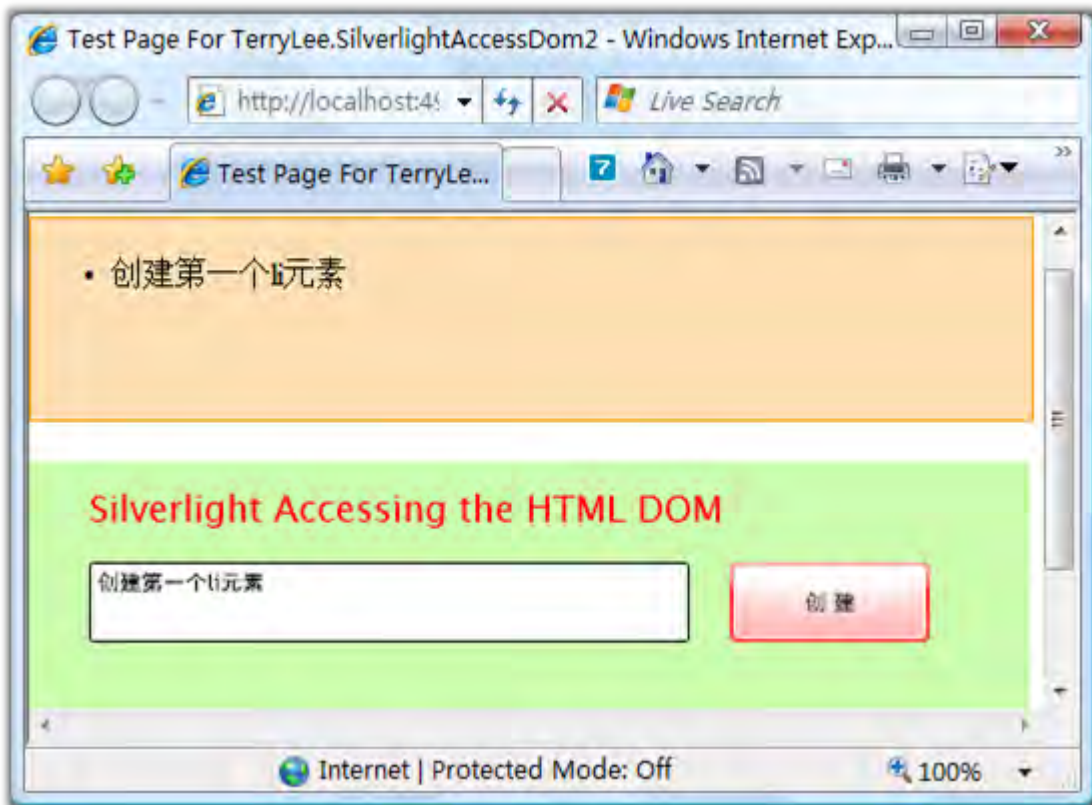
添加新元素到 parent 中

```
parent.AppendChild(child);
```

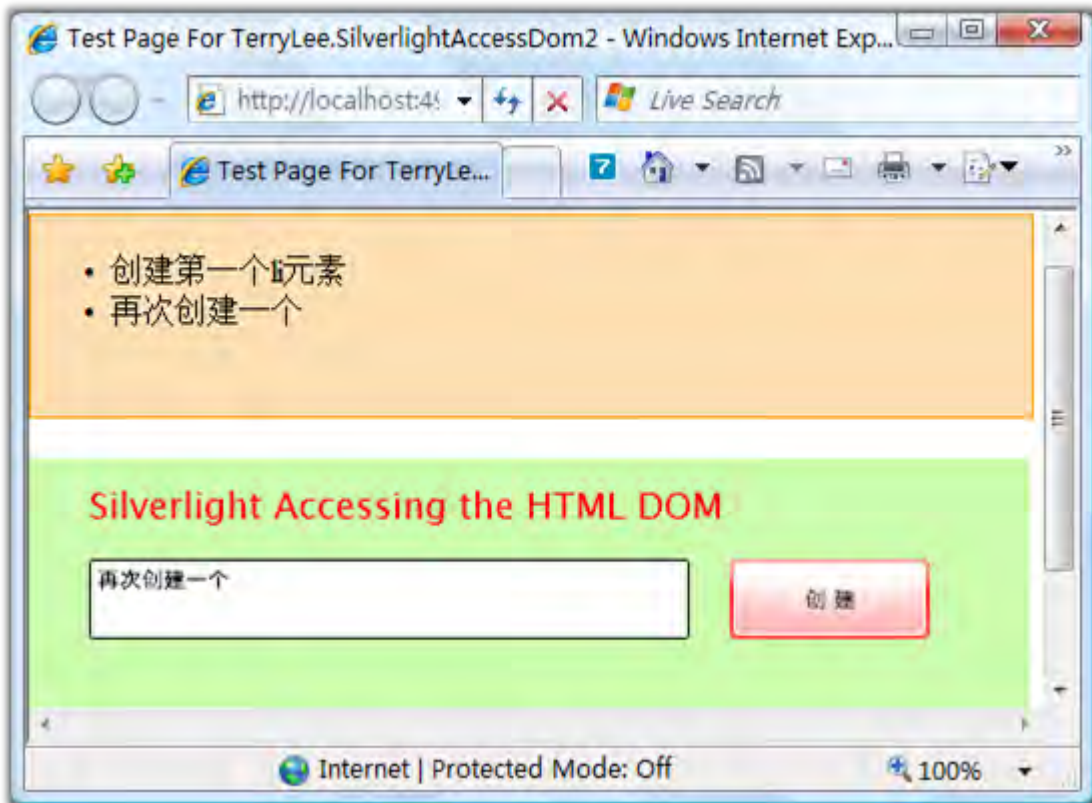
完整的代码如下：

```
private void createButton_Click(object sender, RoutedEventArgs e)  
{  
    HtmlElement parent = HtmlPage.Document.GetElementById("list");  
  
    HtmlElement child = HtmlPage.Document.CreateElement("li");  
    child.SetAttribute("innerText", this.input.Text);  
  
    parent.AppendChild(child);  
}
```

运行后创建第一个元素



再次创建一个





## 移除 DOM 元素

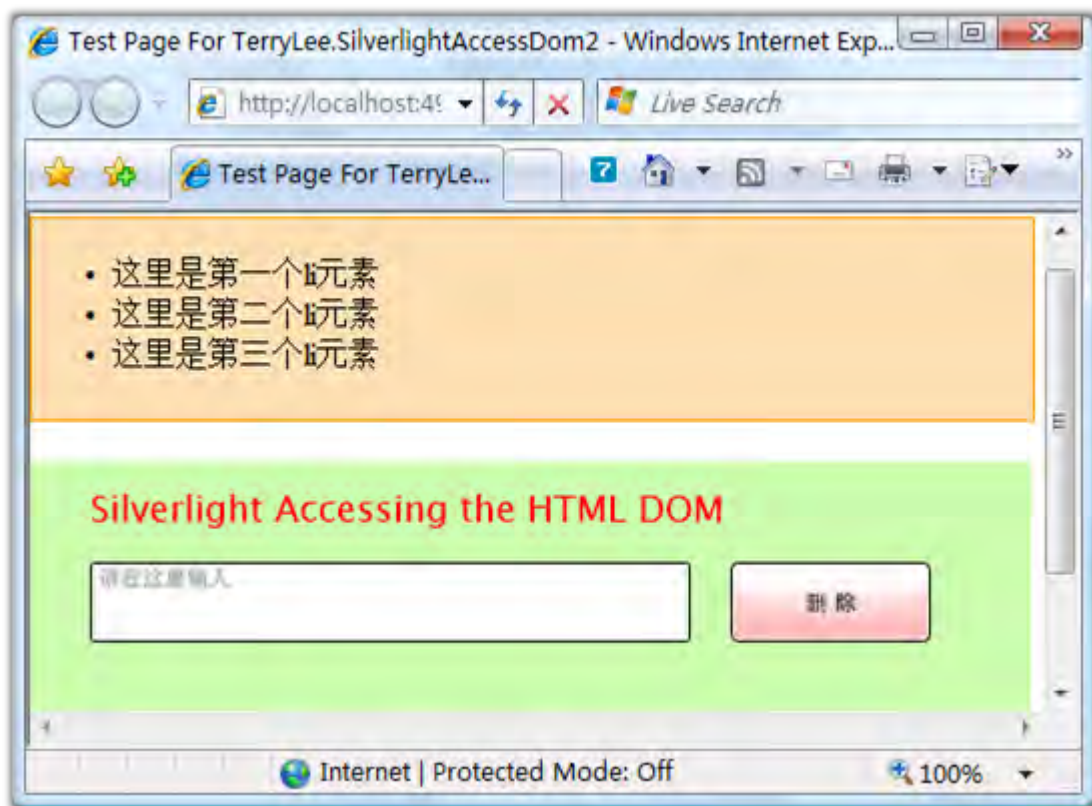
既然可以创建元素，对应的也可以删除元素，我们在页面上放上几个<li>元素，然后输入 id 进行删除。代码如下：

```
private void deleteButton_Click(object sender, RoutedEventArgs e)
{
    HTMLElement parent = HtmlPage.Document.GetElementById("list");

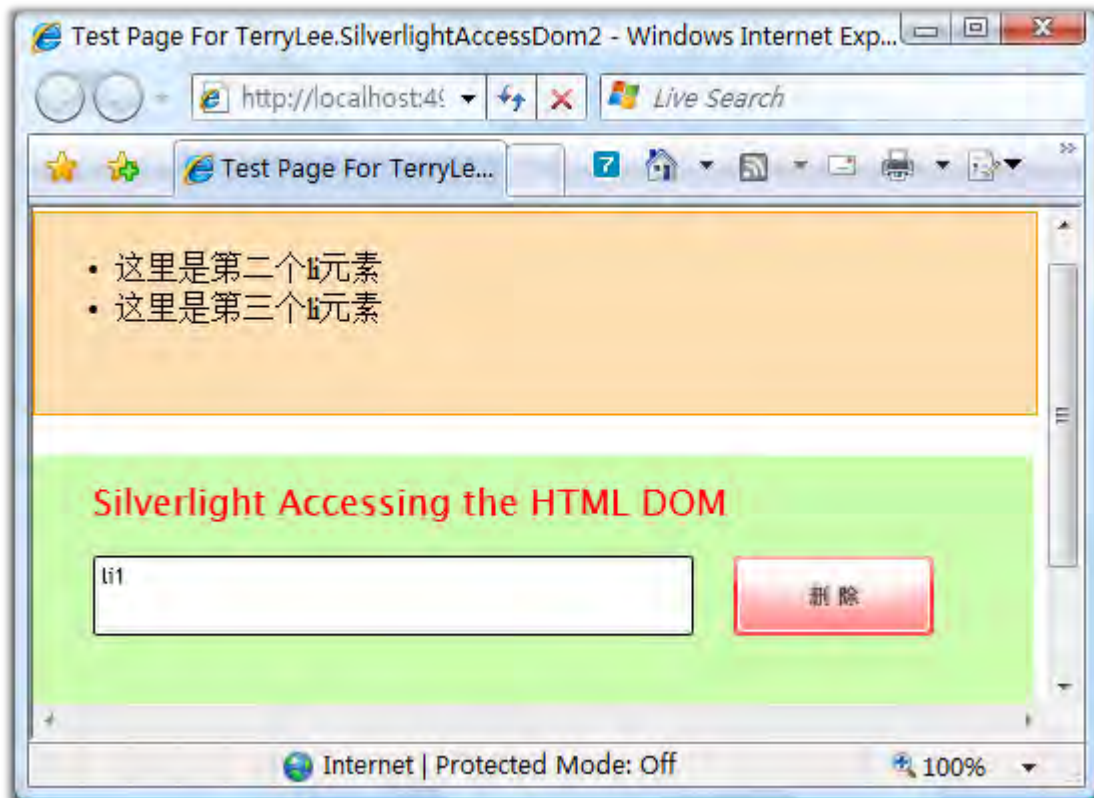
    HTMLElement child = HtmlPage.Document.GetElementById(this.input.Text);

    parent.RemoveChild(child);
}
```

运行后界面上有三个<li>



删除其中一个



## 为 DOM 注册事件

除了添加和移除 DOM 元素外，我们还可以为 DOM 元素附加事件，在下面的例子中我们将通过 Silverlight 动态创建一个 DOM 元素 `<a>`，并未它注册单击事件，单击时修改 Silverlight 中的矩形背景色。先准备相关的 HTML。

```
<div id="parent">

</div>

<div style="height:200px;">

    <asp:Silverlight ID="Xaml1" runat="server"

        Source="~/ClientBin/TerryLee.SilverlightAccessingDom3.xap"

        Version="2.0" Width="100%" Height="200px" />

</div>
```

并且定义两个样式，其中 newstyle 我们将在 Silverlight 中为新添加的 a 元素使用。

```

#parent
{
    background:#FCDFB3;
    border:solid 1px #FF9900;
    width:500px;
    height:100px;
    margin-bottom:20px;
}
.newstyle
{
    background:#0099FF;
    border:solid 1px #0000FF;
}

```

做一个简单的界面，放置一个按钮和矩形：

```

<Canvas Background="#CDFCAE">
    <TextBlock Text="Silverlight Accessing the HTML DOM" Foreground="Red"
        Canvas.Top="10" Canvas.Left="30" FontSize="18">
    </TextBlock>
    <Rectangle x:Name="result" Height="40" Width="300" Fill="Red"
        Canvas.Left="30" Canvas.Top="50"
        RadiusX="5" RadiusY="5">
    </Rectangle>
    <Button x:Name="addButton" Background="Red"
        Height="40" Width="100" Content="添加"
        Canvas.Top="50" Canvas.Left="350"
        Click="addButton_Click">
    </Button>

```

```
</Canvas>
```

添加 DOM 元素，创建一个 a 元素，并为它设置属性，其中用 `CssClass` 来定义它的样式：

```
HtmlElement parent = HtmlPage.Document.GetElementById("parent");

HtmlElement button = HtmlPage.Document.CreateElement("a");
button.SetAttribute("innerText", "改变 Silverlight 中的颜色");
button.SetAttribute("href", "#");
button.CssClass = "newstyle";

parent.AppendChild(button);
```

为 a 元素附加 `onclick` 事件，`HtmlElement` 提供了 `AttachEvent` 方法来附加事件，使用泛型的 `EventHandler`，在 a 元素单击时我们改变 Silverlight 中的矩形填充色和边框。

```
button.AttachEvent("onclick", new EventHandler<HtmlEventArgs>(button_Click));

void button_Click(object sender, HtmlEventArgs e)
{
    result.Stroke = new SolidColorBrush(Colors.Black);
    result.Fill = new SolidColorBrush(Colors.Green);
    result.StrokeThickness = 2;
}
```

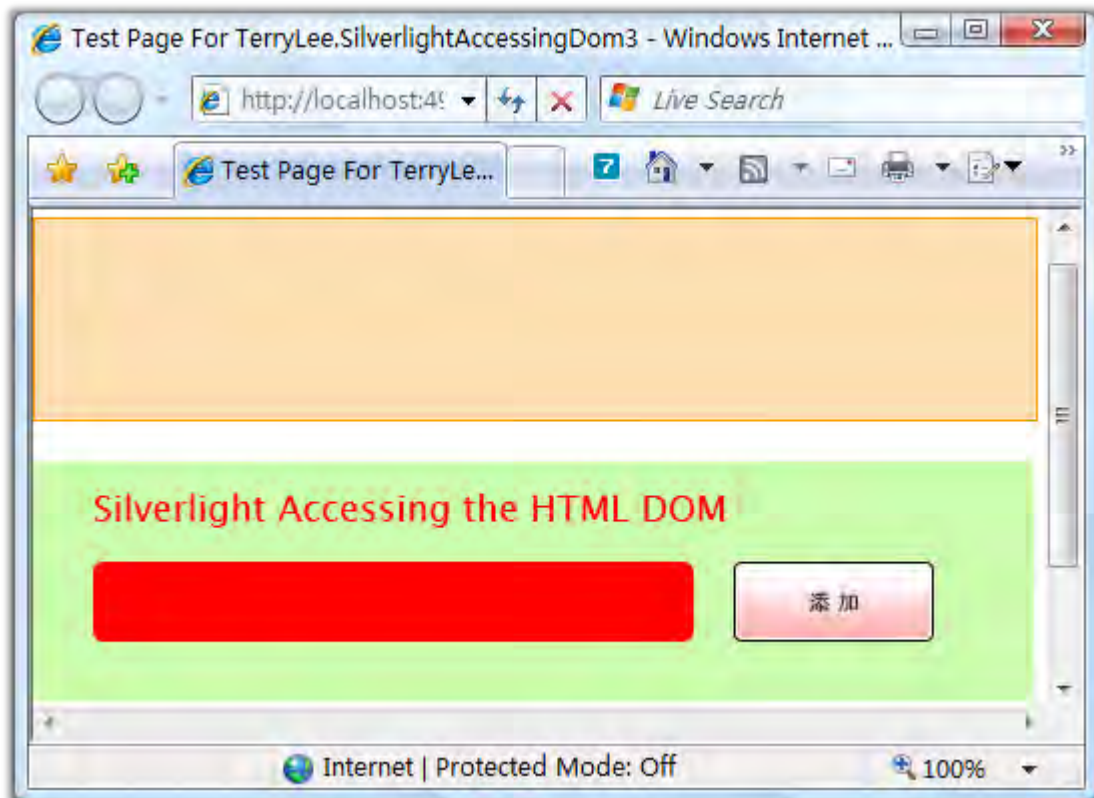
完整的代码如下：

```
private void addButton_Click(object sender, RoutedEventArgs e)
{
    HtmlElement parent = HtmlPage.Document.GetElementById("parent");

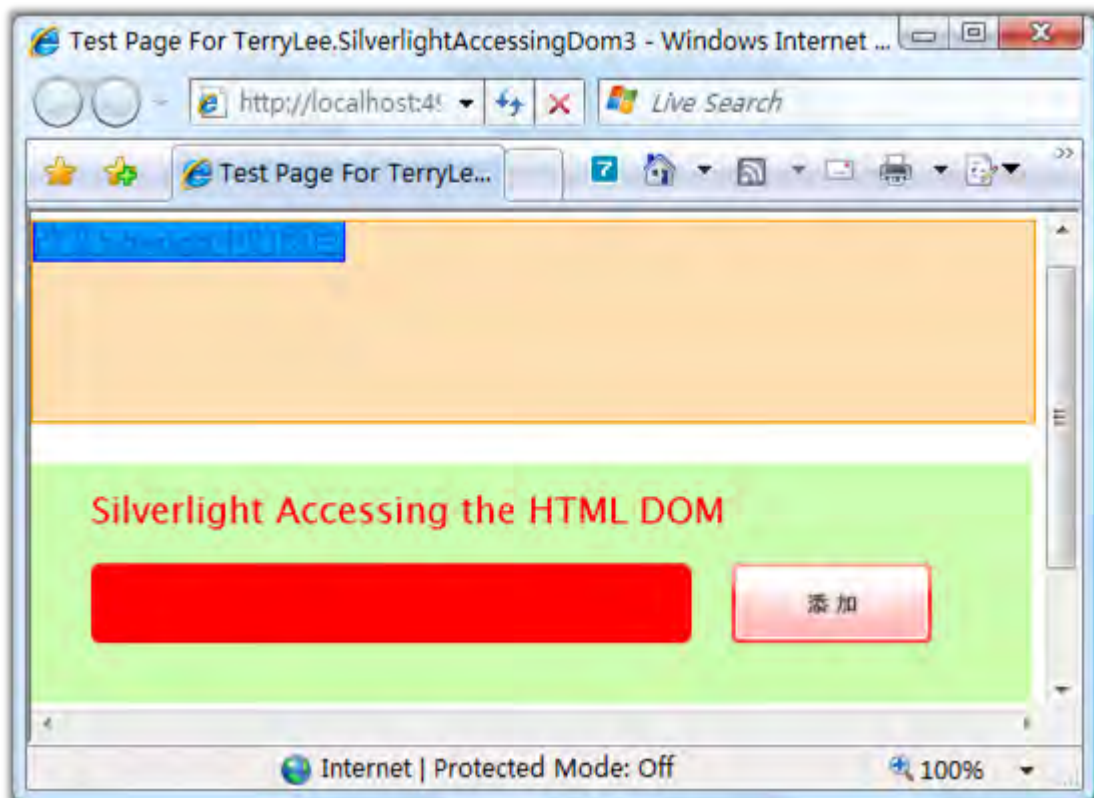
    HtmlElement button = HtmlPage.Document.CreateElement("a");
```

```
button.SetAttribute("innerText", "改变 Silverlight 中的颜色");  
button.SetAttribute("href", "#");  
button.CssClass = "newstyle";  
  
parent.AppendChild(button);  
  
button.AttachEvent("onclick", new EventHandler<HtmlEventArgs>(button_Click));  
}  
  
void button_Click(object sender, HtmlEventArgs e)  
{  
    result.Stroke = new SolidColorBrush(Colors.Black);  
    result.Fill = new SolidColorBrush(Colors.Green);  
    result.StrokeThickness = 2;  
}
```

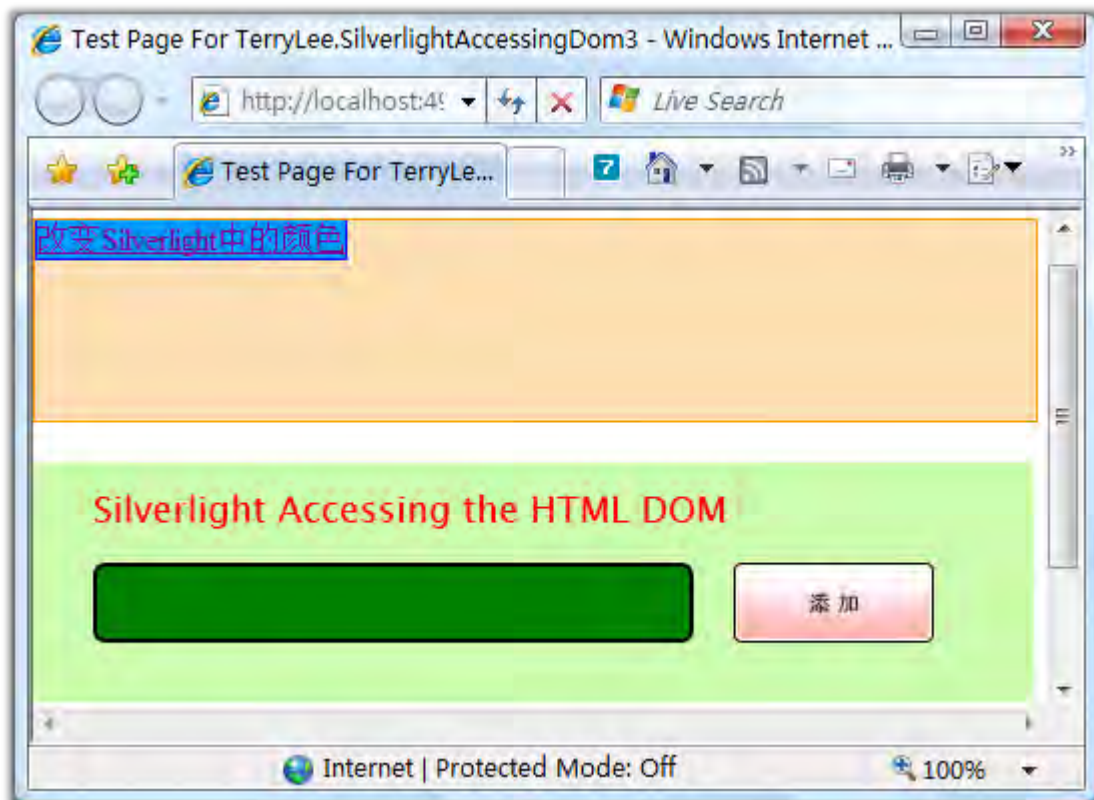
运行一下看看效果如何，起始界面



添加新元素 a



单击改变矩形的背景颜色



HtmlElement 也提供了 DetachEvent 方法，可以取消注册事件。

## 结束语

本文简单介绍了如何在 Silverlight 中添加和移除 DOM 元素，以及为 DOM 元素添加、取消事件处理程序。

## 一步一步学 Silverlight 2 系列（21）：如何在 Silverlight 中调用 JavaScript

### 概述

Silverlight 2 Beta 1 版本发布了，无论从 Runtime 还是 Tools 都给我们带来了很多的惊喜，如支持框架语言 Visual Basic, Visual C#, IronRuby, Ironpython, 对 JSON、Web Service、WCF 以及 Sockets 的支持等一系列新的特性。《一步一步学 Silverlight 2 系列》文章将从 Silverlight 2 基础知识、数据与通信、自定义控件、动画、图形图像等几个方面带您快速进入 Silverlight 2 开发。

Silverlight 中内置了对于 HTML、客户端脚本等的支持。很多情况下，我们编写的 Web 应用程序中用了一些 JavaScript 或者 AJAX 框架，我们希望能够在 Silverlight 调用某些脚本方法，或者说在 Silverlight 中触发某个脚本的执行，这时就需要用到在 Silverlight 中调用 JavaScript，本文将简单介绍这一内容。

### 使用 **GetProperty** 获取脚本对象

先来看一个简单的例子，在 Silverlight 测试页面中放入一个 div 用作显示信息：

```
<div id="result"></div>
```

编写一段简单的 JavaScript 代码：

```
<script type="text/javascript">
    function Hello(message)
    {
        var resultSpan = $get("result");
        resultSpan.innerHTML = "Hello " + message;
    }
</script>
```

再编写一个简单的输入信息界面：

```
<StackPanel Background="#CDFCAE" Orientation="Vertical">
    <StackPanel Height="40">
        <TextBlock Text="Calling Browser Script from Silverlight"
```



```

        Foreground="Red"></TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal">

    <TextBox x:Name="input" Width="340" Height="40" Margin="20 0 20 0"></TextBo
x>

    <Button x:Name="submit" Width="120" Height="40" Background="Red"

        Content="调用" FontSize="20" Foreground="Red" Click="submit_Click"></B
utton>

</StackPanel>

</StackPanel>

```

实现对脚本的调用：

```

private void submit_Click(object sender, RoutedEventArgs e)
{
    ScriptObject hello = HtmlPage.Window.GetProperty("Hello") as ScriptObject;
    hello.InvokeSelf(this.input.Text);
}

```

ScriptObject 提供了任何客户端脚本的封装，不仅仅是 JavaScript，使用其他的 AJAX 框架也可以，如 jQuery 等。然后调用 InvokeSelf() 方法，传入参数，这里 ScriptObject 总共提供了两个方法，Invoke 和 InvokeSelf，如果我们只调用脚本对象的自身，就可以使用 InvokeSelf，如果脚本对象中还有其它的函数等，可以使用 Invoke 传入名称进行调用，两个方法的定义如下：

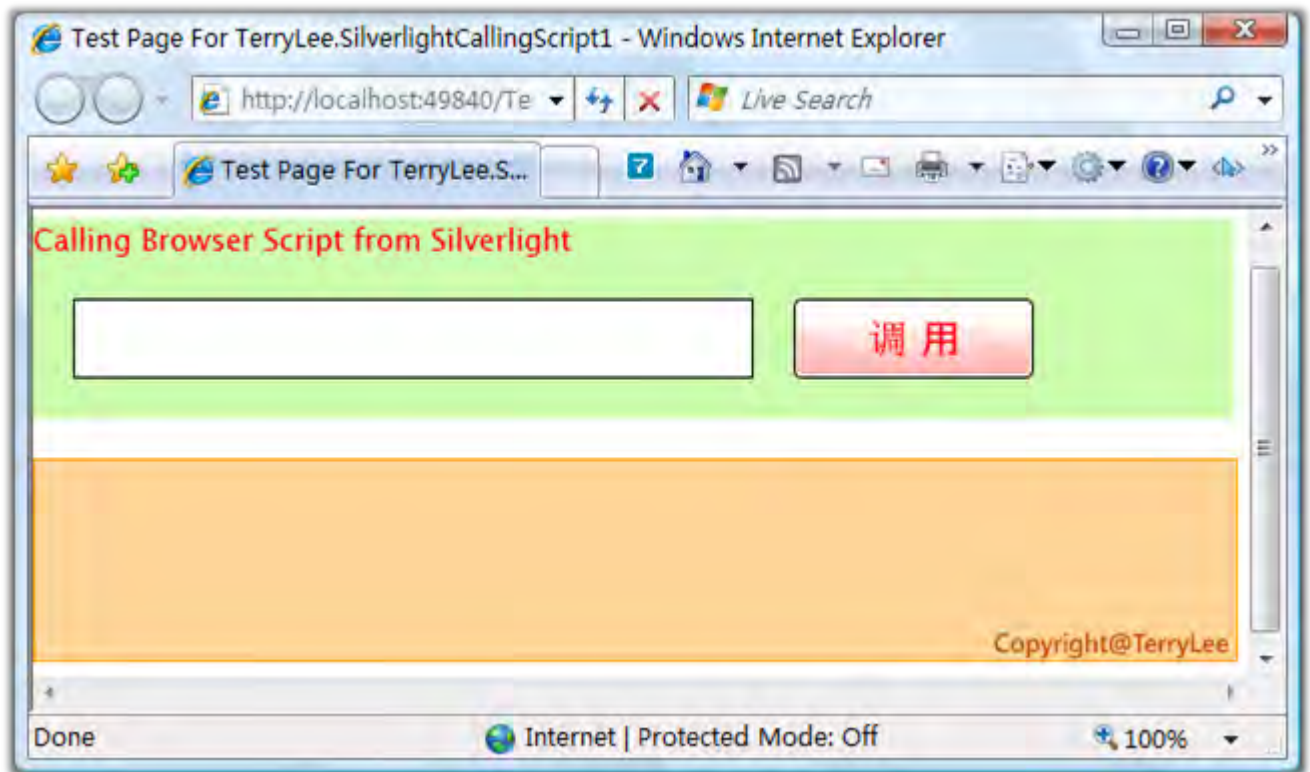
```

[SecuritySafeCritical]
public virtual object Invoke(string name, params object[] args);

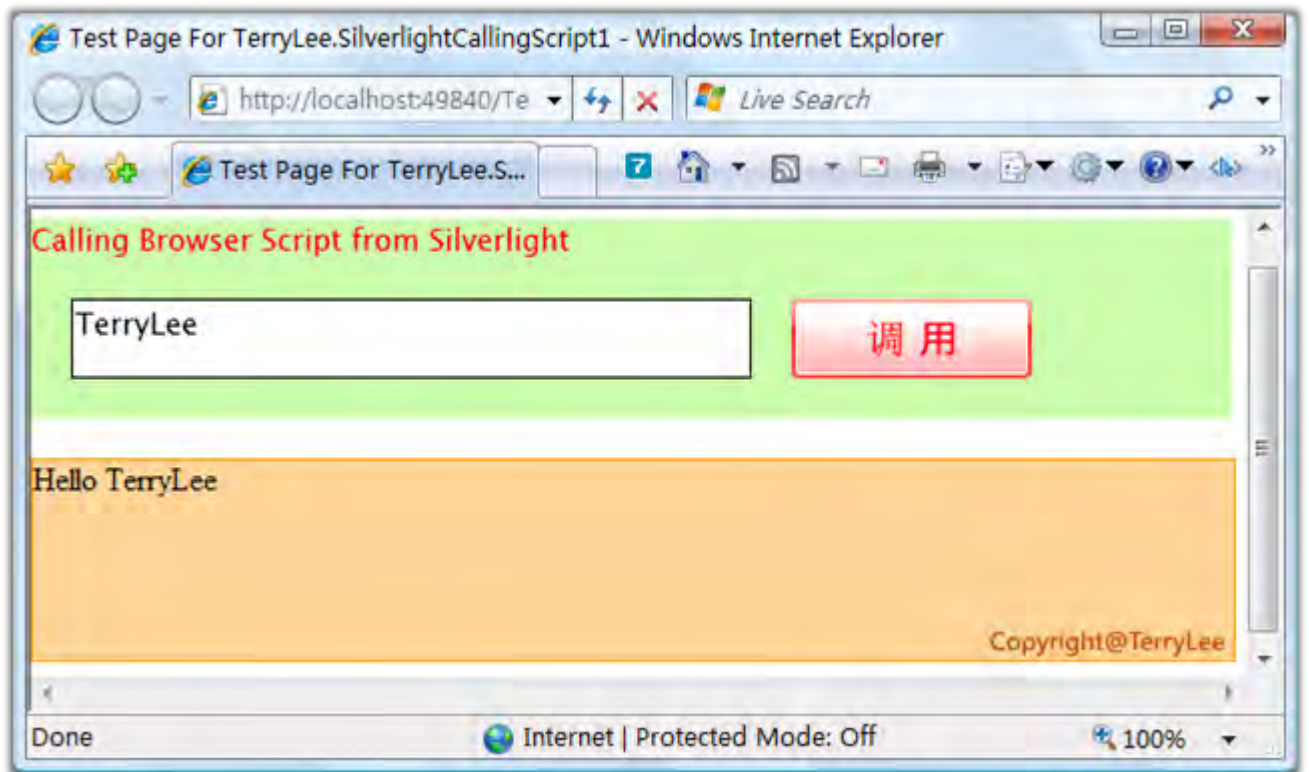
[SecuritySafeCritical]
public virtual object InvokeSelf(params object[] args);

```

运行上面的示例：



输入 TerryLee 后点击调用，可以看到确实调用了客户端脚本：



## 使用 `CreateInstance` 创建脚本对象

除了使用上面所说的使用 `HtmlPage.Window.GetProperty` 方法获取脚本对象之外，还有一种替代方法，即使用 `HtmlPage.Window` 属性的 `CreateInstance` 方法。还是使用上面的示例，我们在测试页中加入如下一段脚本，使用 `prototype` 为 `myHello` 添加了显示的功能：

```
<script type="text/javascript">

    myHello = function(message)
    {
        this.Message = message;
    }

    myHello.prototype.Display = function()
    {
        var resultSpan = $get("result");
        resultSpan.innerHTML = "Hello " + this.Message;
    }
}
```

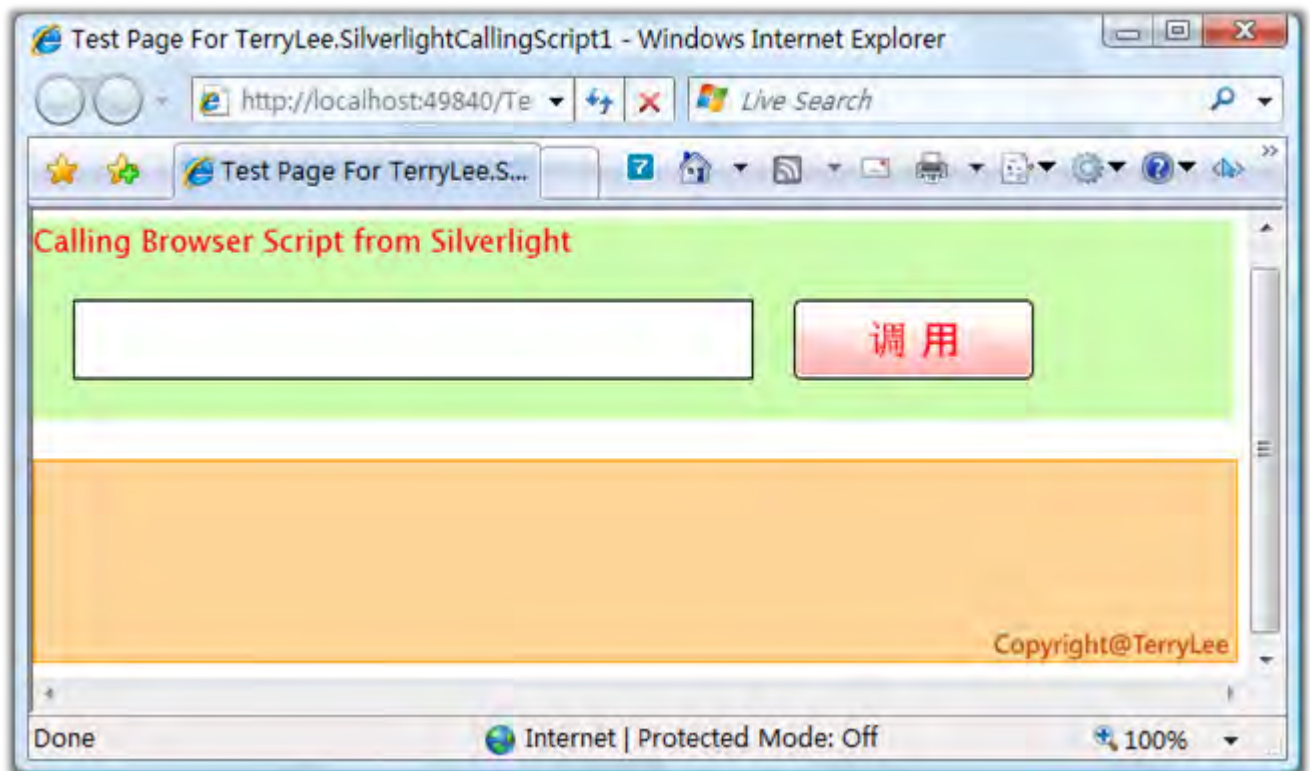
```
</script>
```

使用 `HtmlPage.Window.CreateInstance` 创建脚本对象

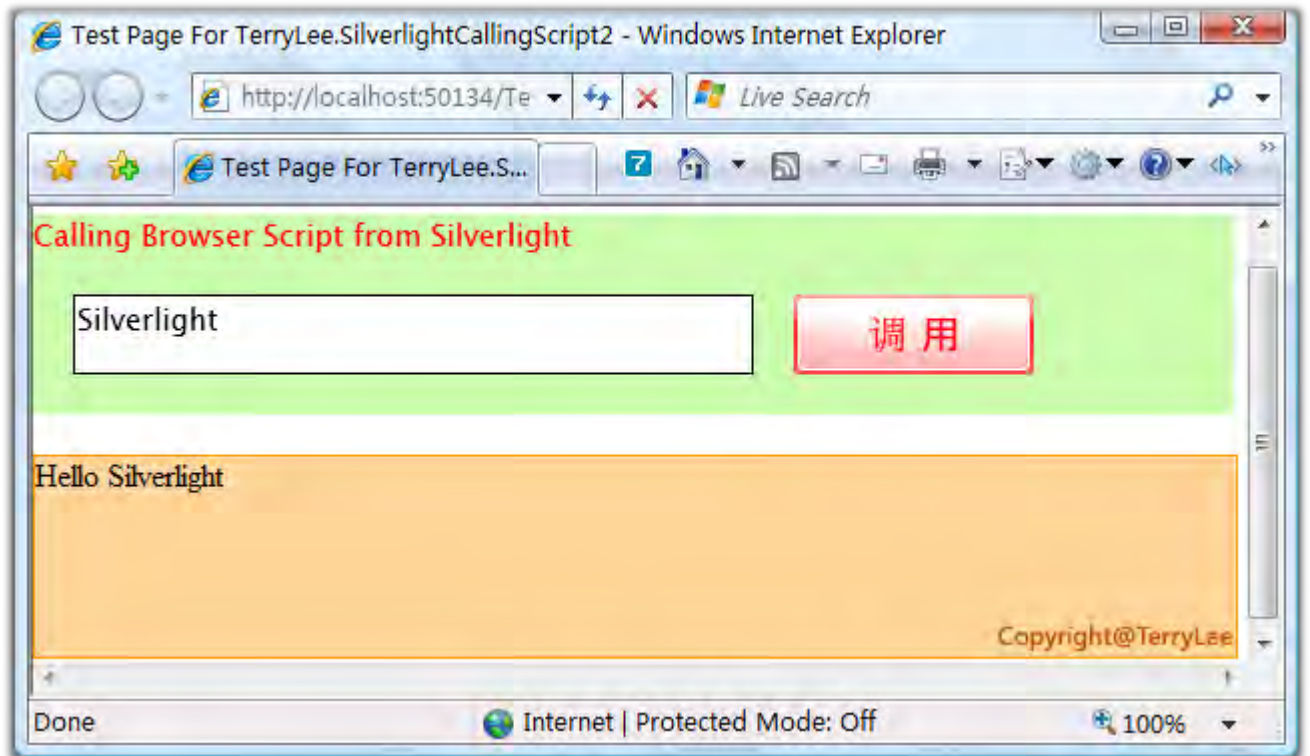
```
private void submit_Click(object sender, RoutedEventArgs e)
{
    ScriptObject script = HtmlPage.Window.CreateInstance("myHello", this.input.Text);

    object result = script.Invoke("Display");
}
```

运行后的效果跟上面的示例是一样的，如：



输入文本信息后：

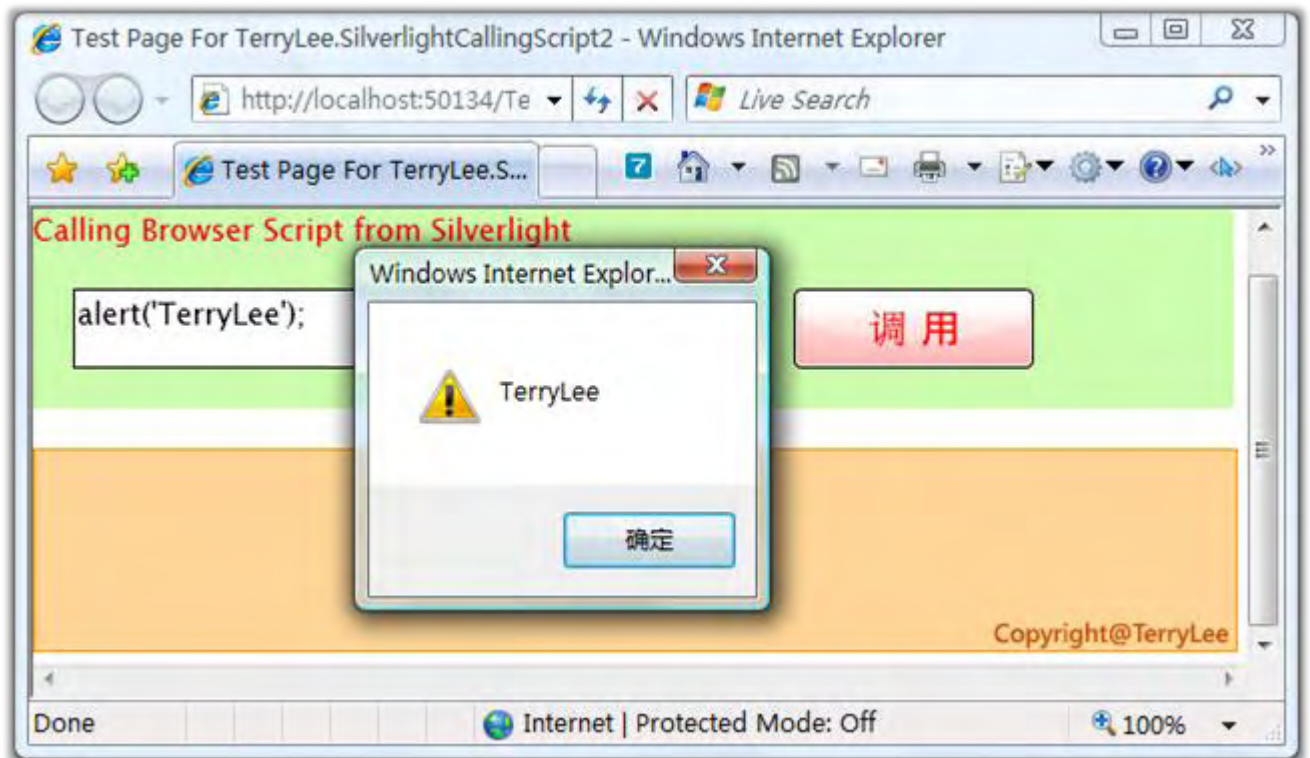


## 使用 `HtmlPage.Window.Eval()`

最后还有一种机制，就是使用 `HtmlPage.Window.Eval()` 方法，只要我们给该方法传入一段字符串，它都会作为 JavaScript 来执行。做一个简单的测试，我们再修改一下上面的示例代码：

```
private void submit_Click(object sender, RoutedEventArgs e)
{
    HtmlPage.Window.Eval(this.input.Text);
}
```

运行后我们在文本框中输入一段脚本 `alert('TerryLee')`；效果如下所示：



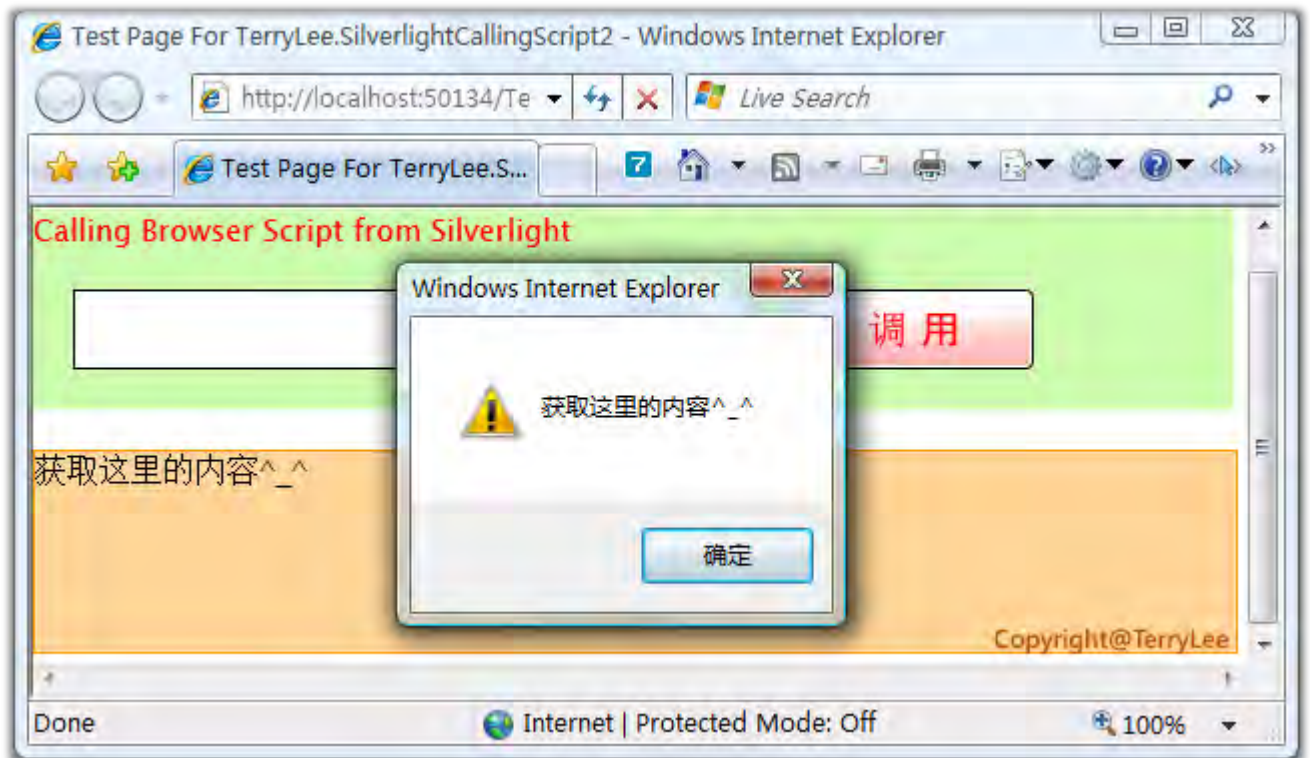
既然 `HtmlPage.Window.Eval()` 可以执行一段脚本，并且将执行的结果以对象形式返回，我们可以使用它来获取 DOM 元素。如下面这段代码：

```
private void submit_Click(object sender, RoutedEventArgs e)
{
    HTMLElement result = HtmlPage.Window.Eval("document.getElementById('result')")
as HTMLElement;

    string message = result.GetAttribute("innerHTML");

    HtmlPage.Window.Alert(message);
}
```

运行后效果如下，获取的 `result` 确实就是我们定义的 `div`。



## 对 **AJAX** 框架的支持

前面说过，ScriptObject 不仅仅是对 JavaScript 的封装，也支持其它的 AJAX 框架，现在我们用 jQuery 来测试一下，编写一小段代码：

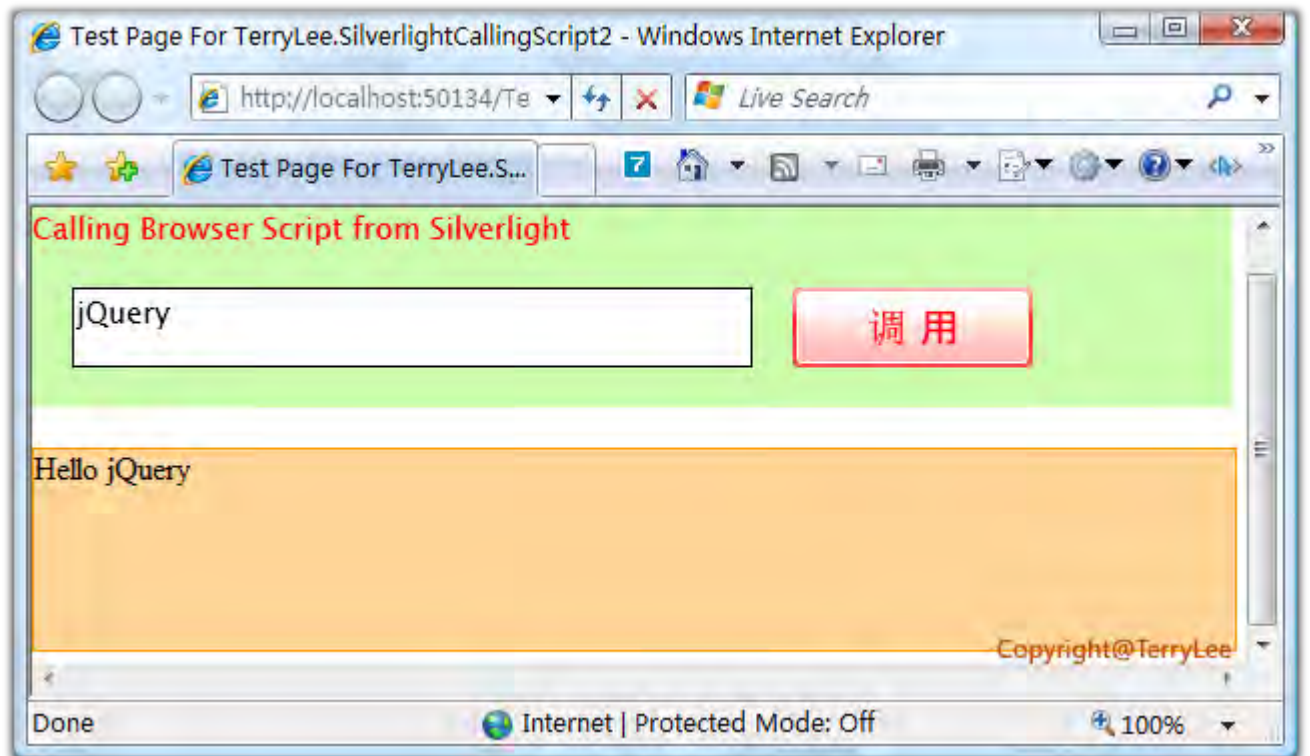
```
<script type="text/javascript">
    function myHello(message)
    {
        $("#result").text("Hello " + message);
    }
</script>
```

调用脚本

```
private void submit_Click(object sender, RoutedEventArgs e)
{
    ScriptObject script = HtmlPage.Window.GetProperty("myHello") as ScriptObject;
```

```
script.InvokeSelf(this.input.Text);  
}
```

运行后的结果与前面的示例是一样的：



## 结束语

本文介绍了在 Silverlight 中调用 JavaScript 的几种方法，下一篇我将介绍如何在 JavaScript 中调用 Silverlight。