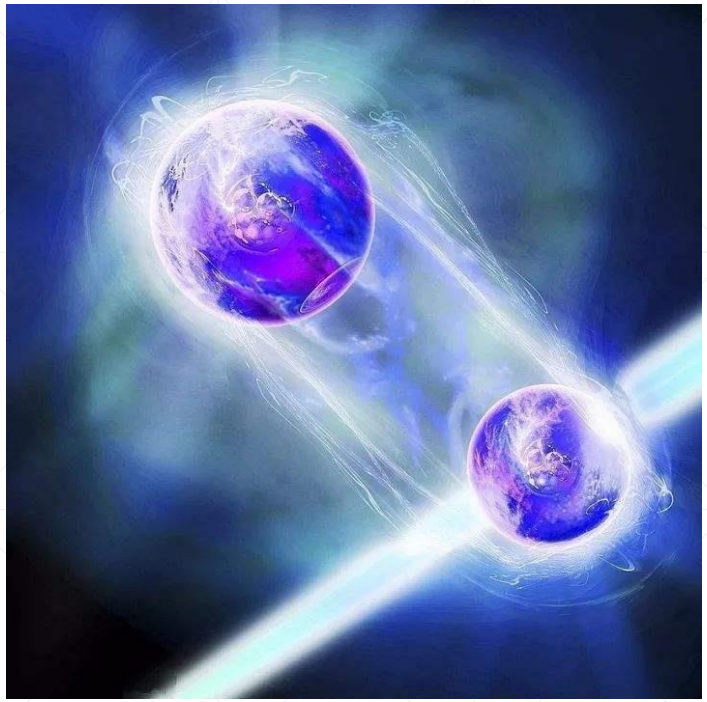
WPF系列之四

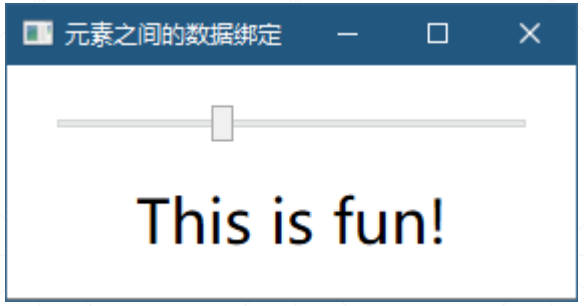


WPF数据绑定技术基础

北京理工大学计算机学院
金旭亮

数据绑定是什么？

从示例中理解“数据绑定”

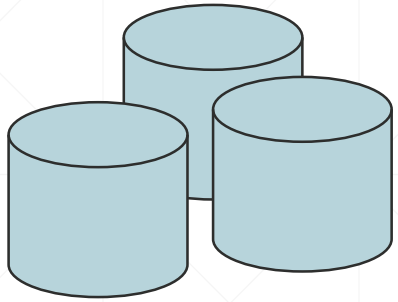


引例：WhatIsDataBind1

```
<Slider x:Name="mySlider" Minimum="12" Maximum="64" />
<TextBlock  Text="This is fun!"
      FontSize="{ Binding ElementName = mySlider,Path = Value }" />
```

WPF实现数据绑定的基本方式：**数据绑定表达式**

“数据绑定”的定义



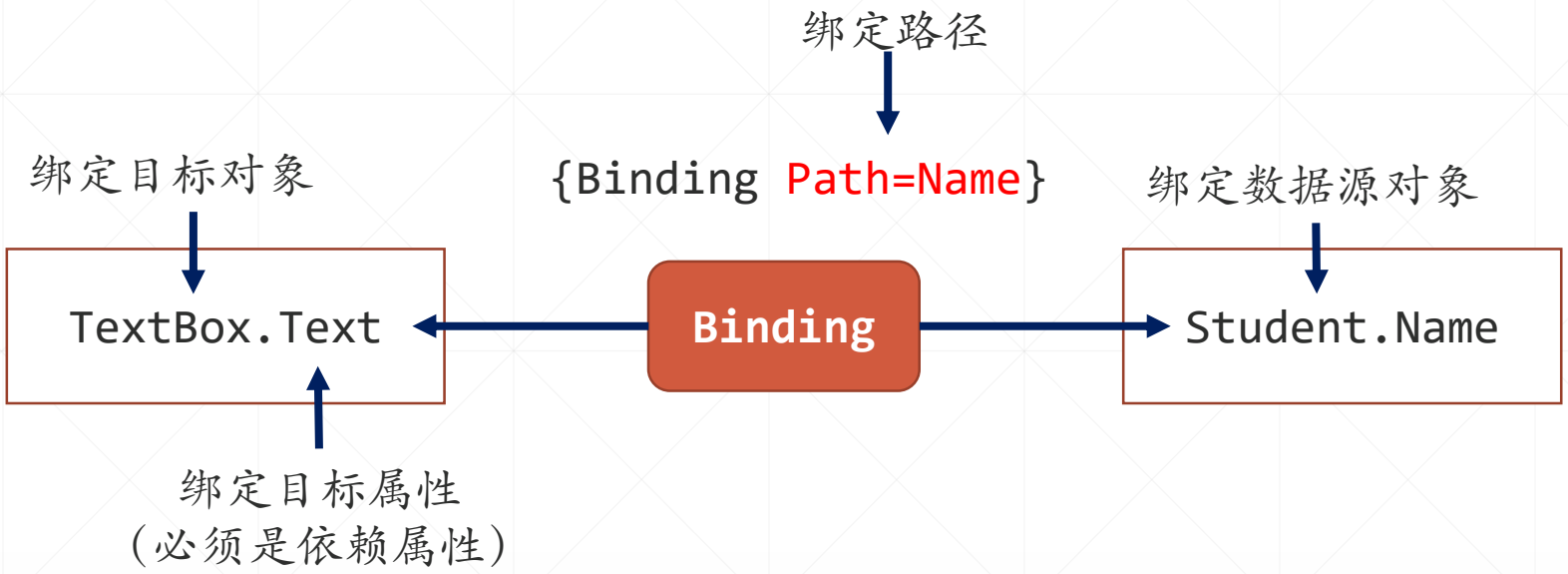
原始数据
(Source)

数据绑定

UI界面
(Target)

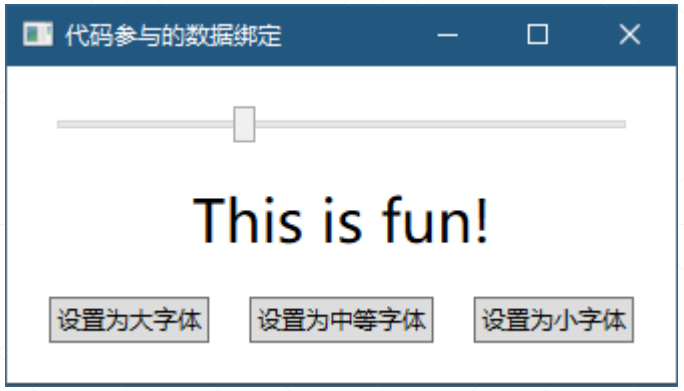
> 数据绑定在提供数据的源对象和目标对象之间建立一种关联，目标对象可以显示和修改源对象的数据，两者自动维持同步。

“数据绑定”四要素



绑定目标对象显示**数据源对象**中的信息。
绑定目标属性是指目标对象中用于显示信息的那个属性，而**绑定路径（Path）**则是一个字串，根据它可以到源对象中提取信息。

应用实例：控件状态的自动同步




```
1 reference
private void btnSetBig_Click(object sender, RoutedEventArgs e)
{
    //设置TextBlock文本块控件的字体大小为30
    txtFont.FontSize = 30;
    //滑块控件会自动滑动到30这个位置，不需要单独控制
}
```

示例：WhatIsDataBind2

三个按钮的事件响应代码中使用代码设置了TextBlock控件的大小，运行之后，点击三个按钮，会发现Slider控件也会**自动**滑动到相应的位置，无需手写代码实现这一功能。

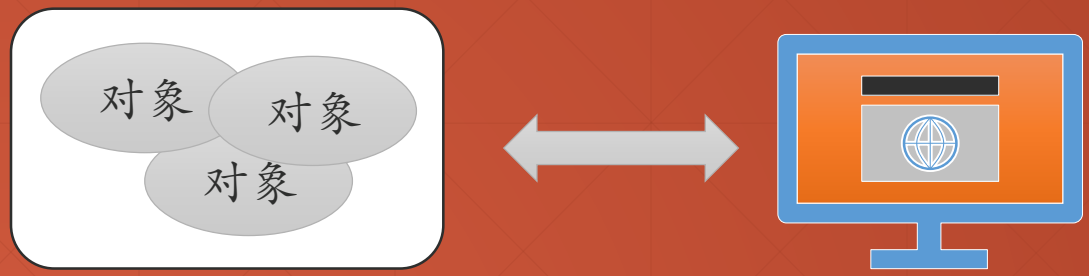
多控件之间的相互协作

示例：WhatIsDataBind3

滑动最上部滑块，可以看到下面的TextBlock控件文本大小随之改变。

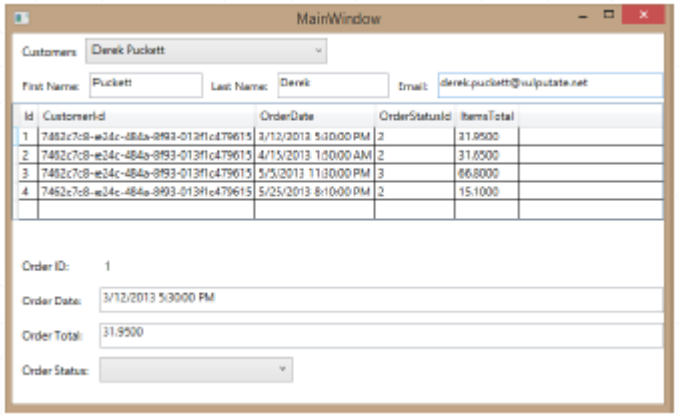
在文本框中输入文本，可以看到最下面的TextBlock控件能同步显示用户输入的文本。

点击中部ListBox，选中不同的选项，可以看到下面的TextBlock控件文本颜色同步改变。



对象集合的显示

实际开发中应用数据绑定



Read

Update
Create
Delete

绑定数据源
对象

Database

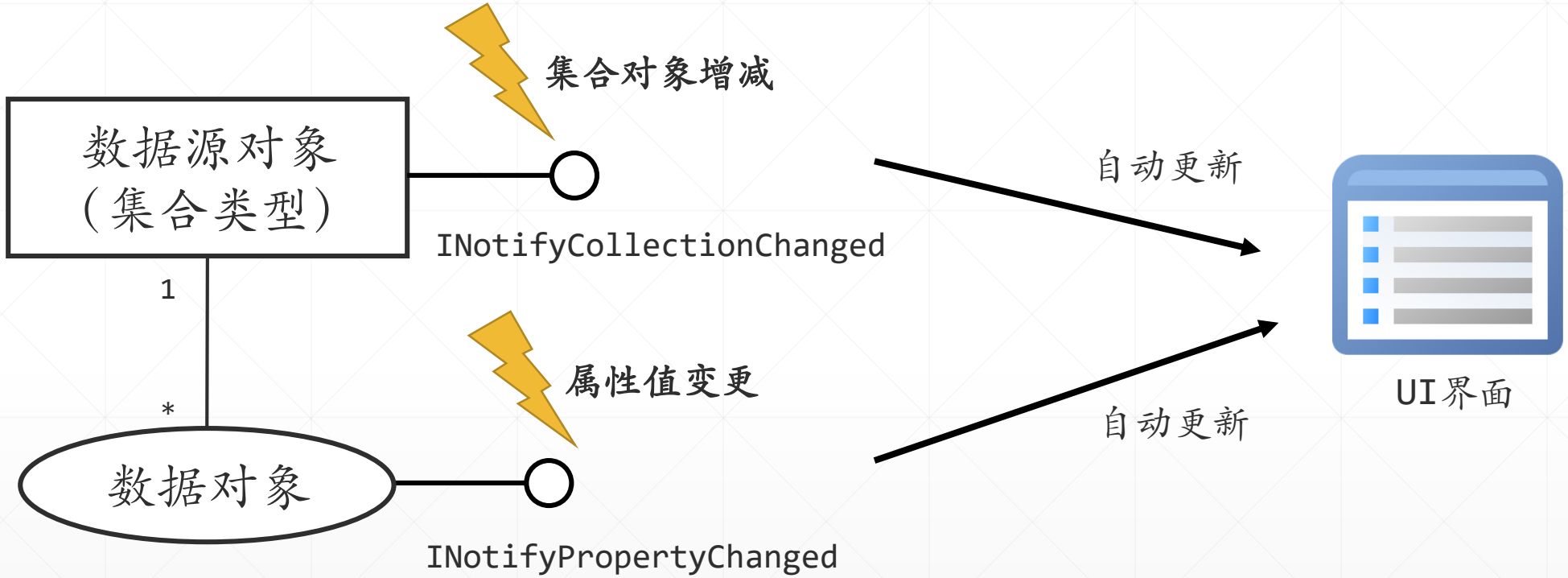
Web

File

数据绑定技术

数据存取技术

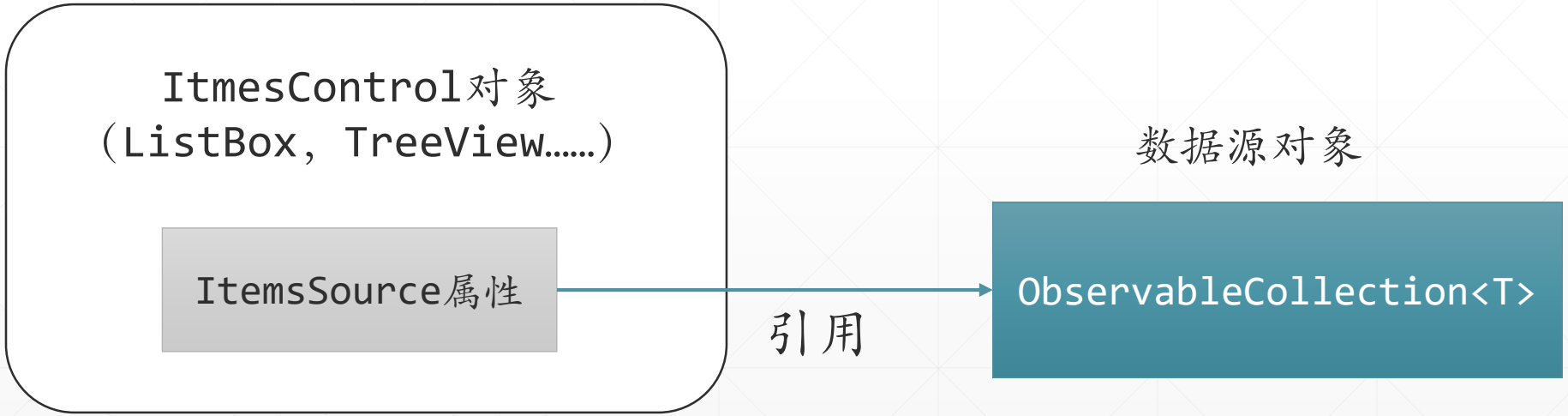
对象集合作为数据源应该满足的两个条件



推荐使用ObservableCollection<T>作为数据绑定源（它实现了INotifyCollectionChanged接口），而集合中的对象应该实现INotifyPropertyChanged，这是一种最常见的编程套路。

使用WPF数据绑定控件显示对象集合

通常使用派生自**ItemsControl**的控件（比如ListBox，DataGrid等）“批量”显示数据源集合中的数据对象。



支持“属性值改变通知”的数据对象

```
//被绑定的数据对象，应该实现INotifyPropertyChanged接口，
//这样一来，属性值变化时WPF的UI界面也能自动刷新
class DataItem : INotifyPropertyChanged
{
    //此属性变更事件是实现界面自动刷新的关键
    public event PropertyChangedEventHandler PropertyChanged;

    private string _itemvalue = "";
    public string ItemValue
    {
        get
        {
            return _itemvalue;
        }
        set
        {
            _itemvalue = value;
            //触发属性值变更事件（真实程序中应该要检查新值和老值是否一致，不一致时才触发事件）
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("ItemValue"));
        }
    }

    private int _itemID = -1;
    public int ItemID...
}
```

注意把握典型的“编程套路”

示例：BindToCollection

支持“对象数目增减通知”的数据源

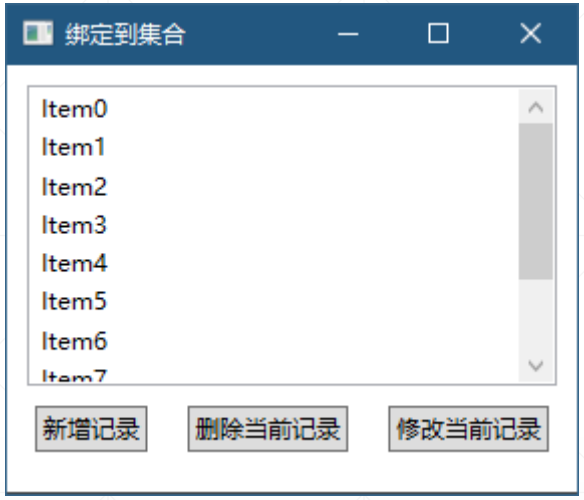
```
1 reference
class MyDataProvider
{
    /// <summary>
    /// 用于生成用于展示WPF数据绑定特性的测试用数据对象集合
    /// </summary>
    /// <returns></returns>
    1 reference
    public static ObservableCollection<DataItem> GetAllItems()
    {
        //使用ObservableCollection来保存数据源
        var items = new ObservableCollection<DataItem>();
        //创建10条测试用数据
        for (int i = 0; i < 10; i++)
        {
            items.Add(new DataItem()
            {
                ItemID = i,
                ItemValue = $"Item{i}"
            });
        }
        return items;
    }
}
```

```
public class ObservableCollection<T> :
    Collection<T>,
    INotifyCollectionChanged,
    INotifyPropertyChanged
{
}
```

基类库的ObservableCollection<T>集合实现了INotifyCollectionChanged接口，当集合中的对象数目有增减时，会自动触发相应的CollectionChanged事件，通知WPF数据绑定引擎更新显示。

示例：BindToCollection

绑定到集合示例



示例：BindToCollection

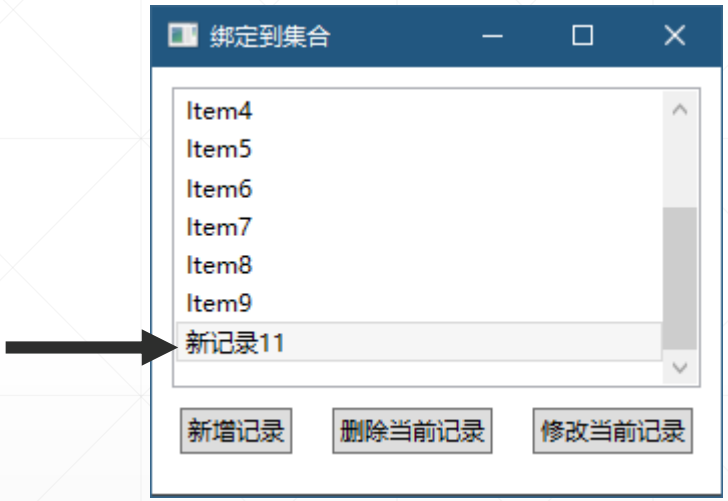
```
public partial class MainWindow : Window
{
    //绑定数据源
    ObservableCollection<DataItem> items;
    public MainWindow()
    {
        InitializeComponent();
        //提取数据
        items = MyDataProvider.GetAllItems();
        //实现绑定
        lstDataItems.ItemsSource = items;
    }
```

```
<ListBox x:Name="lstDataItems"
  DisplayMemberPath="ItemValue"
  ……
/>
```

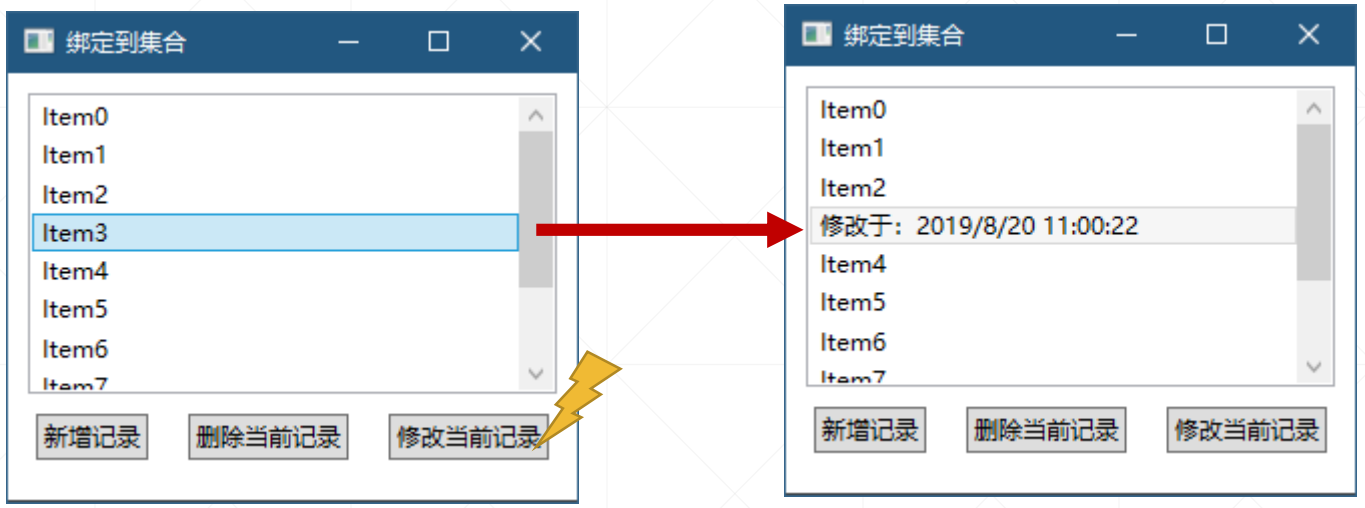
ListBox的ItemsSource属性用于接收数据源对象，它的DisplayMemberPath属性用于指定显示数据对象的哪个属性值

新增记录

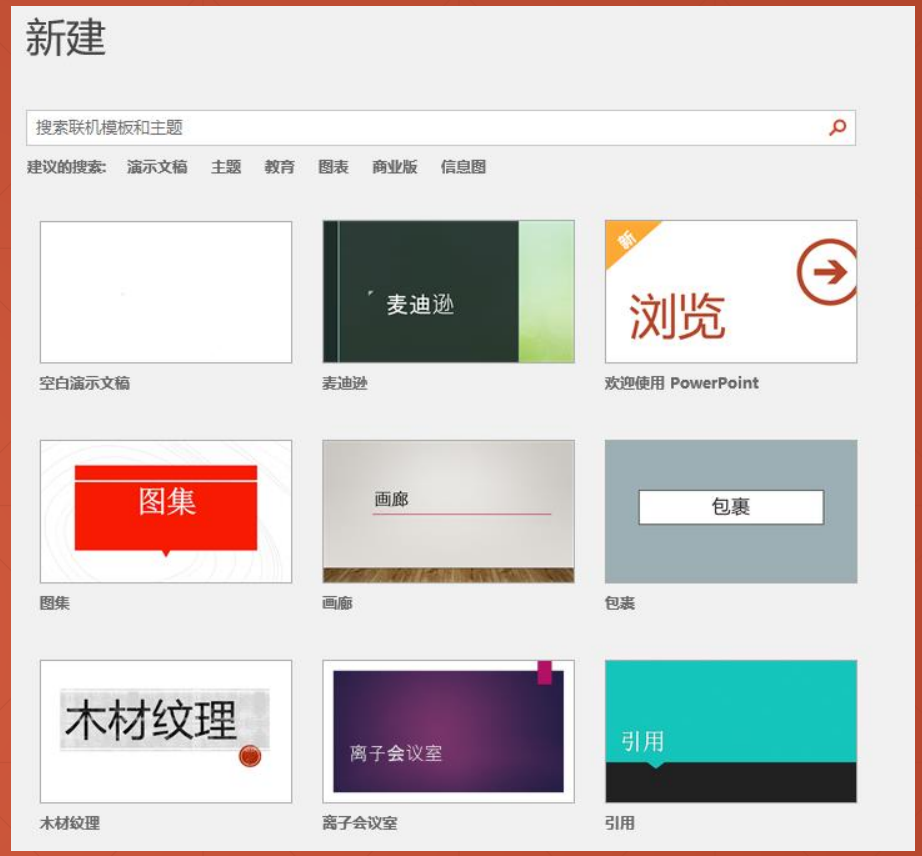
```
private void btnAdd_Click(object sender, RoutedEventArgs e)
{
    //向对象集合中添加一个数据对象
    DataItem obj = new DataItem();
    obj.ItemID = items.Count;
    obj.ItemValue = $"Item{items.Count}";
    items.Add(obj);
    //ListBox滚动显示新加入的记录
    lstDataItems.ScrollIntoView(obj);
    lstDataItems.SelectedIndex = items.Count - 1;
}
```



修改记录



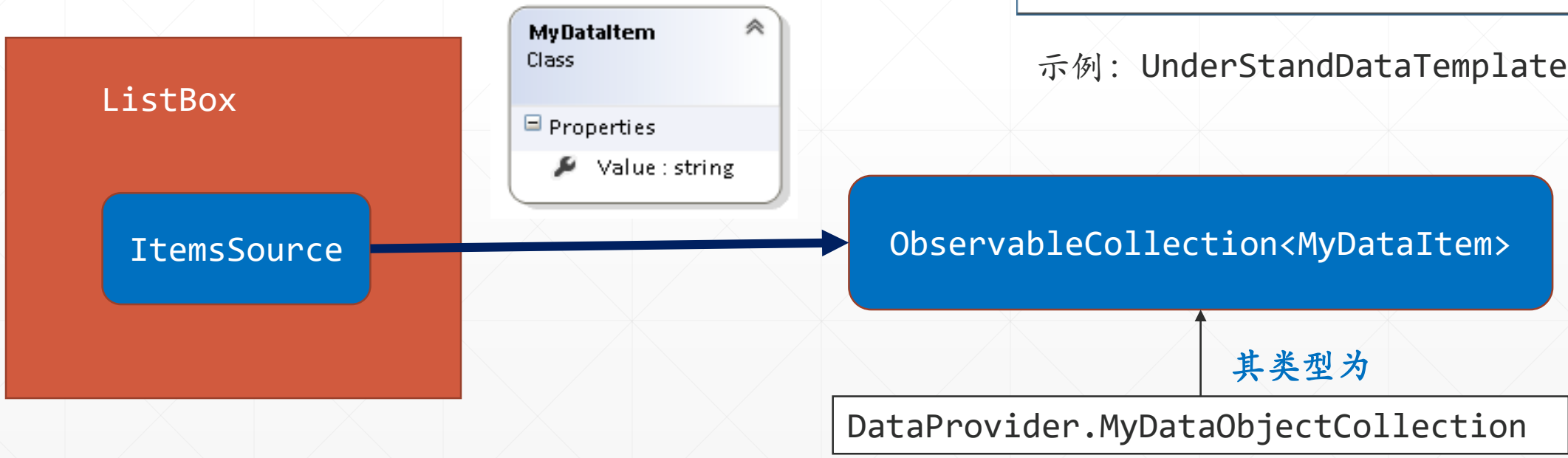

```
private void btnModify_Click(object sender, RoutedEventArgs e)
{
    //确保有选中的当前记录
    if (lstDataItems.SelectedIndex != -1)
    {
        //取出对应的数据对象
        var dataItem = lstDataItems.SelectedItem as DataItem;
        //属性值的修改，会导致ListBox相关记录的自动刷新
        dataItem.ItemValue = $"修改于: {DateTime.Now.ToLocalTime()}";
    }
}
```


新建
搜索联机模板和主题
建议的搜索: 演示文稿 主题 教育 图表 商业版 信息图
空白演示文稿
麦迪逊
麦迪逊
新
浏览
欢迎使用 PowerPoint
图集
图集
画廊
画廊
包裹
包裹
木材纹理
木材纹理
离子会议室
离子会议室
引用
引用

数据绑定模板的妙用

通过示例掌握数据模板的使用技巧

示例：UnderStandDataTemplate

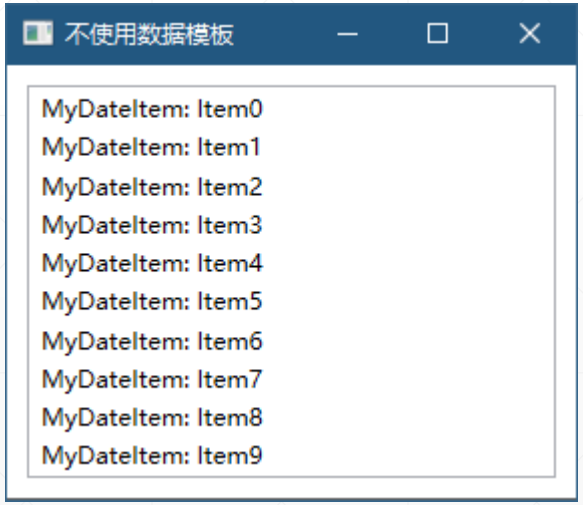


默认行为特性

仅仅设定ListBox的ItemsSource属性绑定到MyDataItem集合对象，其它不做任何设置，……

```
<ListBox ItemsSource="{Binding Source={x:Static local:DataProvider.MyDataObjectCollection}}"/>
```

运行结果

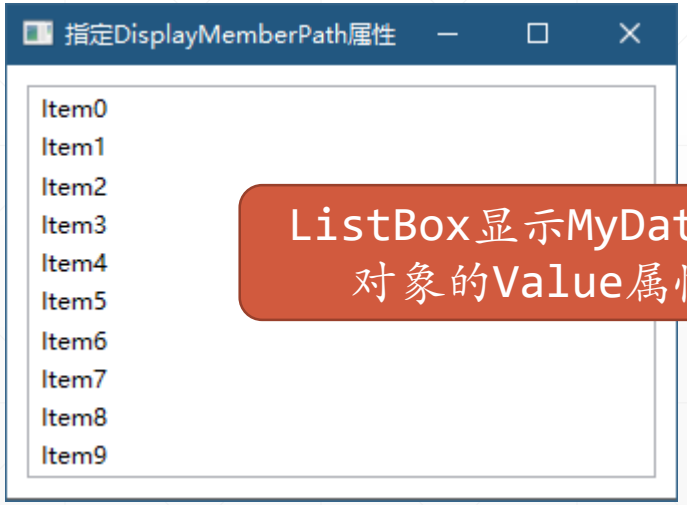


分析：
由于没有任何特定设置，ListBox 在显示集合中的对象时，默认情况下会调用**MyDataItem.ToString()**方法显示一个字符串，因此，可以通过重写数据对象的ToString()方法让ListBox显示一个有意义的字符串。

使用DisplayMemberPath属性

如果设置ListBox控件的DisplayMemberPath属性，可以让ListBox显示出指定属性的值。

```
<ListBox ItemsSource="{Binding Source=…}}" DisplayMemberPath="Value"/>
```



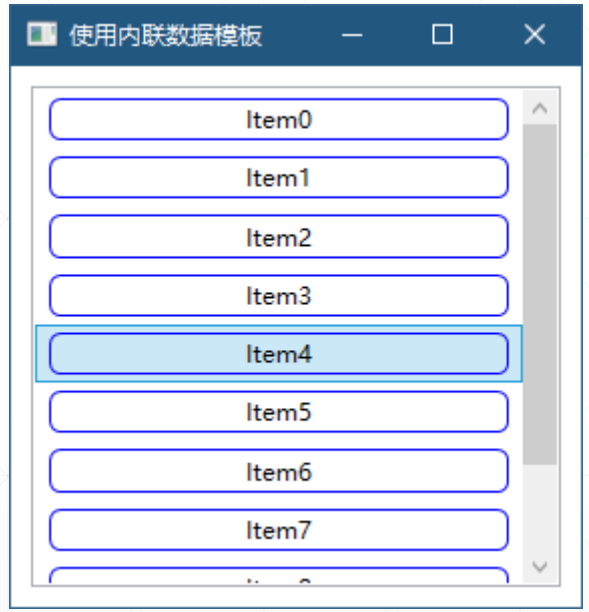
ListBox显示MyDataItem对象的Value属性值

缺陷：
这种方式无法同时显示数据对象的多个属性。

使用数据模板

使用数据模板，可以自定义每个数据对象的显示样式……

```
<ListBox ……>
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Border BorderBrush="Blue" ……>
                <TextBlock Text="{Binding Path=Value}" />
            </Border>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```



在需要的时候，可以将多个控件放到数据模板中，就能同时显示同一个数据对象的多个属性值。

资源中的数据模板

在实际开发中，Data Template通常放到资源中。使其成为一个可重用的对象：

```
<Window.Resources>
        <DataTemplate  x:Key="MyDataTemplate">
            ……
        </DataTemplate>
 </Window.Resources>
```

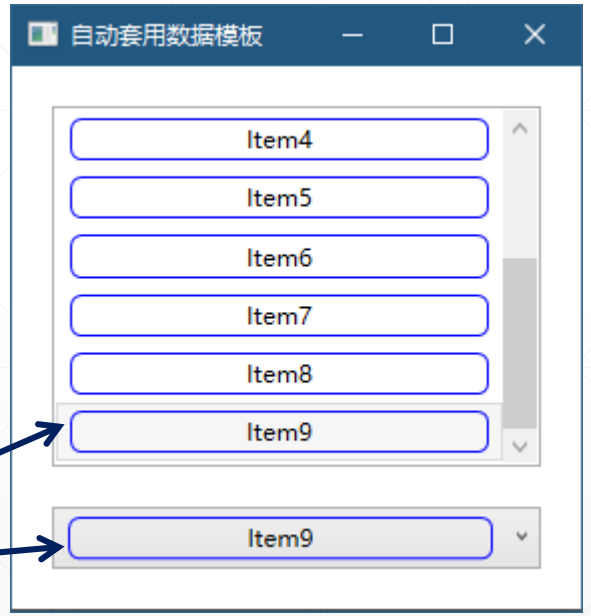
放置在UI界面上的各种数据绑定控件（ListBox，DataGrid之类），现在就可使用这个存放于资源中的数据模板：

```
<ListBox ItemTemplate="{StaticResource MyDataTemplate}"/>
```

自动套用数据模板

如果设定了DataTemplate的DataType属性，此模板会自动应用于所有指定类型的数据源对象，只需为ListBox和ComboBox设定ItemsSource即可。

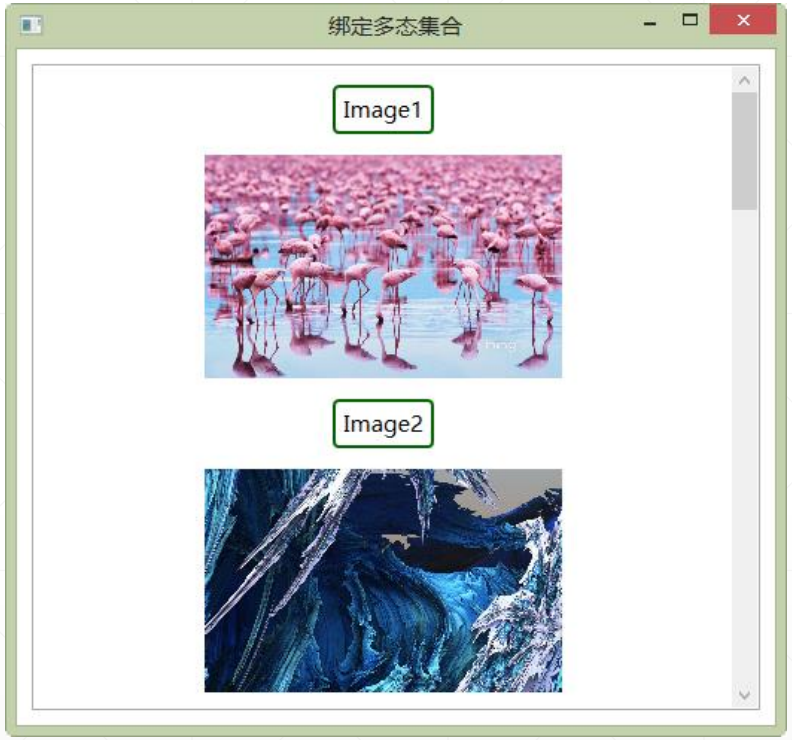
```
<Window.Resources>
    <DataTemplate  DataType="{x:Type local:MyDataItem}">
        ……
    </DataTemplate>
</Window.Resources>
```

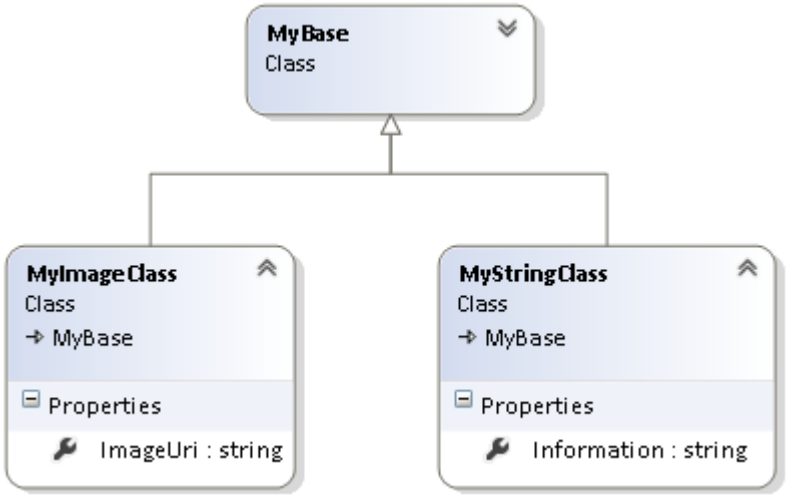


MyDataItem对象将自动应用指定的模板。而这个模板会让ListBox和ComboBox两个控件的选项拥有一致的外观。

另外，设置ListBox和ComboBox控件的IsSynchronizedWithCurrentItem=“True”，可以让这两个控件的当前选中项自动同步。

使用数据模板显示多态数据集合

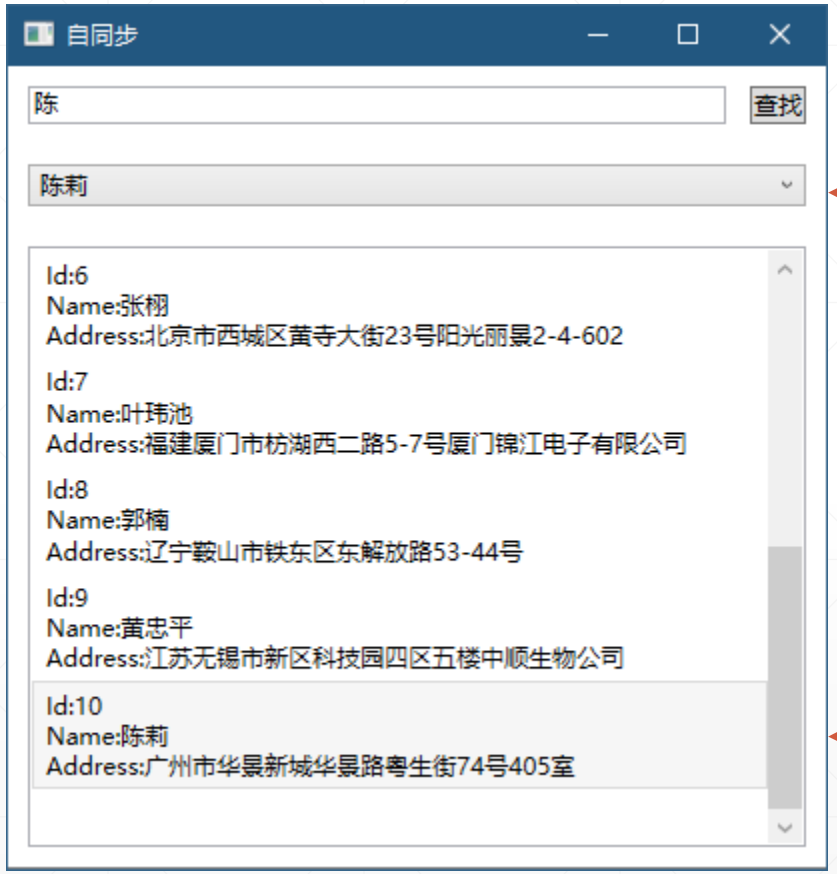




示例：ShowPolymorphismCollection

```
<Window.Resources>
    <DataTemplate DataType="{x:Type local:MyStringClass}">
        <Border HorizontalAlignment="Center"...>
    </DataTemplate>
    <DataTemplate DataType="{x:Type local:MyImageClass}">
        <Image HorizontalAlignment="Center".../>
    </DataTemplate>
</Window.Resources>
```

多个数据绑定控件之间的同步



如果找到某个符合条件的用户，下拉框会自动选择这一用户……

在下拉框中选择某个客户，列表框中自动滚动显示

示例：IsSynchronizedWithCurrentItemDemo

IsSynchronizedWithCurrentItemDemo示例分析

```
<ComboBox x:Name="cboClient"
          ItemsSource="{Binding}"
          DisplayMemberPath="Name"
          SelectedValuePath="Id"
          IsSynchronizedWithCurrentItem="true"
          SelectionChanged="cboClient_SelectionChanged"
          Margin="10"
          />

<ListBox x:Name="lstClient"
         ItemsSource="{Binding}"
         IsSynchronizedWithCurrentItem="true"
          Margin="10"
         MaxHeight="300"
         >
    <ListBox.ItemTemplate...>
</ListBox>
```

诸如ListBox、ComboBox等数据绑定控件，都有一个**IsSynchronizedWithCurrentItem**属性，当其值为true时，会自动设置自己的**SeletedItem**属性为绑定的对象集合的当前项。

IsSynchronizedWithCurrentItemDemo示例分析

```
2 references
public partial class MainWindow : Window
{
    List<Client> Clients = null;
    0 references
    public MainWindow()
    {
        InitializeComponent();
        //设定绑定数据源
        Clients = ClientRepository.GetAllClients();
        DataContext = Clients;
    }
    1 reference
    private void btnFind_Click(object sender, RoutedEventArgs e)
    {
        //查找用户
        var client = Clients.FirstOrDefault(
            c => c.Name.StartsWith(txtFind.Text.Trim()));
        if (client != null)
        {
            //自动选中第一个匹配项
            cboClient.SelectedItem = client;
            //自动滚动显示选中项
            lstClient.ScrollIntoView(client);
        }
    }

    1 reference
    private void cboClient_SelectionChanged(object sender,
        SelectionChangedEventArgs e)
    {
        lstClient.ScrollIntoView(cboClient.SelectedItem);
    }
}
```

给父类控件的DataContext属性赋值，其子控件（ListBox和ComboBox）都能接收到。

使用LINQ，可以很方便地在数据集合中查找记录，之后，设置ComboBox的SelectedItem项，并调用ListBox的ScrollIntoView方法自动滚动显示找到的记录。

使用ListBox的**ScrollIntoView**方法，可以让其滚动显示特定的项

小结

- 本讲PPT介绍了WPF数据绑定机制的一些重要特性，可以让你领略到它的强大与灵活之处。
- WPF数据绑定机制的技术特性其实很丰富，本讲只是展示了其中的一部分，更多的技术特性请课后通过互联网自学，掌握了本讲所介绍的内容，自学更多的知识应该是没问题的。
- WPF数据绑定机制的一个重要应用是用它来实现MVVM设计模式，由于本课程时间有限，所以未作介绍，这个任务留为作业。