依赖属性可以通过使用Binding从数据源获得值（依赖在别人身上）的属性。拥有依赖属性的对象称为“依赖对象”。

传统的.NET Framework属性称为CLR属性，实例的每个CLR属性都包含着一个非静态的字段，这需要占用内存，.NET Framework中的许多类拥有许多属性，其中只有很少一部分是常用的，这样就会造成内存的浪费。

在WPF中，允许对象在被创建的时候并不包含用于存储数据的空间（即字段所占用的空间）、只保留在需要用到数据时能够获得默认值、借用其他对象数据或实时分配空间的能力。这种对象称为依赖对象（Dependency Object）而它这种实时获取数据的能力则依靠依赖属性来实现。

依赖项属性可以像普通属性那样使用。

依赖项属性只能使用在继承了DependencyObject类的类中。

依赖项属性必须为static，且为public。

依赖项属性使用的7个场景：

1. 希望可在样式中设置属性。
2. 希望属性支持数据绑定。
3. 希望可使用动态资源引用设置属性。
4. 希望从元素树中的父元素自动继承属性值。
5. 希望属性可进行动画处理。
6. 希望属性系统在属性系统、环境或用户执行的操作或者读取并使用样式更改了属性以前的值时报告。
7. 希望使用已建立的、WPF 进程也使用的元数据约定，例如报告更改属性值时是否要求布局系统重新编写元素的可视化对象。

一个简单的依赖属性例程：

例：程序dependency\_property\_test1

MainWindow.xaml

<Window x:Class="dependency\_property\_test1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:dependency\_property\_test1"

Title="MainWindow" Height="350" Width="525">

<StackPanel>

<TextBlock x:Name="textBlock" />

<Button Content="Change" Click="Button\_Click" />

</StackPanel>

</Window>

MainWindow.xaml.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Data*;

using *System*.*Windows*.*Documents*;

using *System*.*Windows*.*Input*;

using *System*.*Windows*.*Media*;

using *System*.*Windows*.*Media*.*Imaging*;

using *System*.*Windows*.*Navigation*;

using *System*.*Windows*.*Shapes*;

namespace dependency\_property\_test1

{

/// <summary>

/// MainWindow.xaml 的交互逻辑

/// </summary>

public partial class MainWindow : *Window*

{

public MainWindow()

{

*InitializeComponent*();

simple = new Simple();

// 由于Simple不是UIElement的派生类型，

// 无法在XAML中添加绑定

*Binding* binding = new *Binding*();

binding.*Source* = simple;

binding.*Path* = new *PropertyPath*("YearPublished");

binding.*Mode* = *BindingMode*.*TwoWay*;

textBlock.*SetBinding*(*TextBlock*.*TextProperty*, binding);

}

private Simple simple;

private void Button\_Click(object sender, *RoutedEventArgs* e)

{

++simple.YearPublished;

}

}

}

Simple.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

namespace dependency\_property\_test1

{

class Simple:*DependencyObject*

{

public int YearPublished

{

get { return (int)*GetValue*(YearPublishedProperty); }

set { *SetValue*(YearPublishedProperty, value); }

}

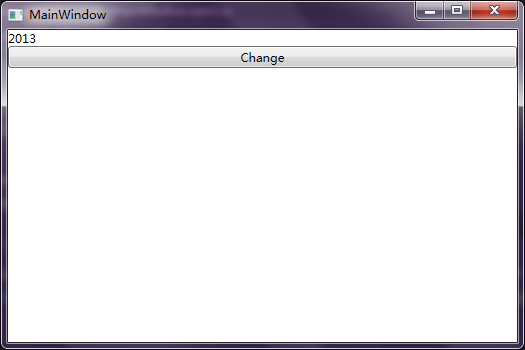
public static readonly *DependencyProperty* YearPublishedProperty =

*DependencyProperty*.*Register*("YearPublished", typeof(int), typeof(Simple), new *PropertyMetadata*(2013));

}

}

在输出结果中，点击Button按钮，TextBlock中的值会跟着变化



依赖属性：

例：程序dependency\_property\_test2

MainWindow.xaml

<Window x:Class="dependency\_property\_test2.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="DependencyProperty" Height="350" Width="525">

<StackPanel>

<TextBox x:Name="textBox1" BorderBrush="Black" Margin="5" />

<TextBox x:Name="textBox2" BorderBrush="Black" Margin="5" />

</StackPanel>

</Window>

Student.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Data*;

namespace dependency\_property\_test2

{

class Student:*DependencyObject*

{

// CLR属性包装器

public string Name

{

get { return (string)*GetValue*(NameProperty); }

set { *SetValue*(NameProperty, value); }

}

// 依赖属性

// Register函数的3个参数

// 第1个参数为string类型，指明以哪个CLR属性作为依赖属性的包装器

// 第2个参数用来指明依赖属性存储什么类型的值

// 第3个参数指明依赖属性的宿主类型或者说依赖属性注册关联到哪个类型

public static readonly *DependencyProperty* NameProperty =

*DependencyProperty*.*Register*("Name", typeof(string), typeof(Student));

// SetBinding包装

public *BindingExpressionBase* SetBinding(*DependencyProperty* dp, *BindingBase* binding)

{

return *BindingOperations*.*SetBinding*(this, dp, binding);

}

}

}

MainWindow.xaml.cs

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

using *System*.*Windows*;

using *System*.*Windows*.*Controls*;

using *System*.*Windows*.*Data*;

using *System*.*Windows*.*Documents*;

using *System*.*Windows*.*Input*;

using *System*.*Windows*.*Media*;

using *System*.*Windows*.*Media*.*Imaging*;

using *System*.*Windows*.*Navigation*;

using *System*.*Windows*.*Shapes*;

namespace dependency\_property\_test2

{

/// <summary>

/// MainWindow.xaml 的交互逻辑

/// </summary>

public partial class MainWindow : *Window*

{

public MainWindow()

{

*InitializeComponent*();

Student student = new Student();

// 设置绑定的数据源和Path

// 数据源为textBox1，Path为Text属性

student.SetBinding(Student.NameProperty, new *Binding*("Text") { *Source* = textBox1});

// 数据源为student，Path为Name属性

textBox2.*SetBinding*(*TextBox*.*TextProperty*, new *Binding*("Name") { *Source* = student});

}

}

}

依赖属性的存储：

参考WPF深入浅出.pdf第161页，可以发现在创建和注册依赖属性DependencyProperty对象时，实际是利用DependencyProperty实例的CLR属性和宿主类型名生成hash code，最后把hash code和DependencyProperty实例作为Key-Value对存入全局的，名为PropertyFromName的Hashtable中。WPF属性系统通过CLR属性名和宿主类型名从这个全局的Hashtable中检索到DependencyProperty实例。