WPF面试题：入门2道+初级17道+中级23道+高级8道，共计50道8994字

入门篇[2]

1.什么是WPF?

WPF 是微软推出的表现层UI开发框架，全称 Windows Presentation Foundation。 相对Winform来讲，它使用一种全新的桌面应用程序 UI 的开发方式。 除了像Winform那样在“Windows 窗体”上删除控件之外，WPF 还为应用程序开发提供了额外的功能改善，包括丰富的用户界面、动画等等。

简而言之，可以使用 WPF 完成以下操作：

- 绘制普通控件和图形。
- 轻松加载/播放音频和视频文件。
- 提供平滑的图形效果，例如阴影和颜色渐变。
- 使用可跨相同控件使用的共享样式，以提供相同的主题、皮肤和设计。
- 变换对象，包括形状、控件和视频。
- 可以创建和动画 3D 图形。
- 可以轻松绘制可缩放的矢量图形而不会出现锯齿状锯齿。

2.WPF中的XAML是什么？为什么需要它？它只存在于WPF吗?

XAML 是用来组织 WPF UI 的 XML 文件。 以XML标签方式表示UI的重点是编写一次可以在其他地方运行它，比如Blend软件也能正常加载与编辑。

XAML 不仅仅适用于 WPF。 XAML 是一种基于 XML 的语言，它有多种变体。

WPF初级篇[17]

1.简单描述下WPF的样式

WPF 样式的工作方式与 CSS 样式类似

在 CSS 中，我们为控件定义样式，并在应用程序中任何需要的地方重用相同的样式

与 WPF 中的样式允许定义属性并可在应用程序中重用的方式相同。

2.WPF 中的资源是什么?

资源提供了一种简单的方法来重用已定义的对象和值。 WPF 中的资源允许一次设置多个控件的属性。例如，可以使用单个资源在 WPF 应用程序中的多个元素上设置背景属性。

定义资源的最佳方式是在 Window 或 Page 元素级别。 为元素定义的任何资源也适用于该元素的子元素。

3.WPF中的Visibility.Collapsed和Visibility.Hidden有什么区别?

- `Visibility.Hidden` 隐藏控件，但保留它在布局中占用的空间。 所以它呈现空白而不是控件。

- `Visibilty.Collapsed` 不呈现控件并且不保留空格。 控件占用的空间是“折叠的”

4.什么是静态资源和动态资源?

- Static Resource - StaticResource 的值在加载时确定
- Dynamic Resource - 在运行时更改属性值的情况下使用。

5.WPF中控件的分类?

WPF控件可以分为四类：

- **Control: -** 大部分时间使用的基本控件。 例如文本框、按钮等。像按钮、文本框、标签等独立控件的控件被称为内容控件。 还有其他控件可以容纳其他控件，例如 itemscontrols。 Itemscontrol 可以有多个文本框控件、标签控件等。
- **Shape: -** 帮助我们创建简单的图形控件，如椭圆、线条、矩形等。
- **Panel: -** 有助于对齐和定位控件。 例如，grid 帮助我们以表格方式对齐，stack panel 有助于水平和垂直对齐。
- **Content presenter: -** 用于将任何 XAML 内容放入其中。

6.WPF中的命令设计模式是什么

命令设计模式是面向对象设计模式中最强大的设计模式之一。 此模式允许将操作请求与实际执行操作的对象分离，换句话说，命令模式将操作表示为对象。 Command 对象不包含要执行的功能。 这消除了命令定义和功能之间的直接联系，并促进了松散耦合。 当需要根据用户请求实现操作时，命令模式是处理对象的最佳模式。

命令设计模式的成员包括： Client、调用者、命令、具体执行内容、接收者

7.XML和XAML有什么区别?

以下是 XML 和 XAML 之间的区别：

1. 所有 XAML 文档都可以称之为 XML 文档。 然而，反过来说却是不能的。
2. XAML 是一种声明性应用程序语言，而 XML 是一种标记语言。
3. XML 主要用于 Web 应用程序。 相比之下，XAML 用于设计 Windows 和其他 Web 应用程序的控件。
4. XAML 侧重于对象属性、定义以及它们之间的关系。
5. XML 是 W3C 产生的一种标记语言，用于描述其他标记语言。

8.WPF中的xmlns 和xmlns:x有什么区别?

这两个命名空间都有助于定义/解析 XAML UI 元素。

- 第一个命名空间是默认命名空间，有助于解析整体 WPF 元素。
- 第二个命名空间以“x:”为前缀，有助于解析 XAML 语言定义。

例如，对于下面的 XAML 片段，我们有两个东西，一个是“StackPanel”，另一个是“x:name”。“StackPanel”由默认命名空间解析，“x:name”由“xmlns:x”命名空间解析。

```
<StackPanel x:Name="myStack" />
```

9.相对于Winform，WPF有什么优势?

- 1 - 绑定（更简约的编码）
- 2 - 灵活的外观和感受（资源和样式）

- 3 - 声明式编程 (XAML)
- 4 - 表现层混合动画（动画开发简单）
- 5 - 快速加载（硬件加速）
- 6- 图形硬件无关（分辨率无关）

10.什么是WPF的值转换器?

值转换器充当目标和源之间的桥梁，当目标与一个源绑定数据类型不一致时，需要值转换器来做中转。

例如有一个文本框和一个按钮控件， 当文本框的文本被填充或为空时，希望启用或禁用按钮控件。

在这种情况下，需要将字符串数据转换为布尔值。这可以使用值转换器实现。

要实现值转换器，需要继承System.Windows.Data命名空间中的IValueConverter，并实现两个方法Convert和ConvertBack。

11.XAML 文件中的 xmlns 是什么?

“xmlns”代表 XML 命名空间。 它帮助我们避免 XML 文档中的名称冲突和混淆。

12.我们什么时候应该使用“x:name”和“name”?

“x:name”和“name”没有区别，“name”是“x:name”的简写。 但是当对象无法使用“name”属性时，就需要使用“x:name”属性。

[NEW]13.在WPF中进行对齐的各种方法是什么?

FrameworkElement有两个对齐属性：HorizontalAlignment和Vertical alignment。

Horizontal Alignment属性是**HorizontalAlignment**枚举的一种类型，表示子元素如何水平放置在父元素中。

HorizontalAlignment枚举具有四个属性Left、Center、Right和Stretch。“左”、“中”和“右”属性将子元素设置为父元素的左、中和右。Stretch属性拉伸子元素以填充父元素分配的布局空间。

VerticalAlignment属性是**HorizontalAlignment**枚举的一种类型，表示子元素在父元素中的垂直位置。

VerticalAlignment枚举具有四个属性Top、Center、Bottom和Stretch。“顶部”、“中心”和“底部”属性将子元素设置为父元素的顶部、中心或底部。Stretch属性拉伸子元素以垂直填充父元素分配的布局空间。

[NEW]14.描述WPF中的多段线

多段线是由一条或多条直线（或圆弧）线段组成的对象。矩形是您已经熟悉的多段线示例。正如您所看到的，它是一个可以修改的对象，可以使用比四条单独的线更容易的方式进行操作。

换句话说，多段线是连接直线的集合。Polyline对象表示多段线形状，并使用给定点绘制多段线。Points特性表示多段线中的点。Stroke属性设置颜色，StrokeThickness表示多段线的线宽。

[NEW]15.WPF提供了哪些不同类型的画刷?

画笔和笔是用于绘制和填充图形对象的对象。WPF采用XAML进行渲染，每个XAML控件都有一个WPF类。例如，XAML中的标记和WPF中的SolidColorBrush类都表示纯色画笔。

在XAML和WPF模型中，提供了以下画笔对象：

1. SolidColorBrush
2. LinearGradientBrush

3. RadialGradientBrush
4. DrawingBrush
5. Visual Brush
6. ImageBrush

[NEW]16.什么是弹出窗口以及如何打开和关闭弹出窗口

弹出窗口是一个浮动在页面或窗口上的窗口，提供一些快速操作的功能。例如，页面或窗口上的登录控件或控件的动画弹出提示。XAML的Popup元素表示WPF Popup控件。

- 打开：当IsOpen设置为true时，弹出控件显示其内容
- 关闭：当IsOpen设置为false时，Popup控件显示其内容。

[NEW]17.什么是WPF中的旋转变换

RotateTransform将元素绕点顺时针旋转指定角度。WPF中的RotateTransform对象表示RotateTransform。Angle属性表示顺时针旋转的角度（以度为单位）。CenterX和CenterY属性表示中心点的X和Y坐标。默认情况下，ScaleTransform以点（0,0）为中心，该点对应于矩形的左上角。

WPF中级篇[23]

1.描述下WPF对象完整的层次结构?

- **Object:** 由于 WPF 是使用 .NET 创建的，因此 WPF UI 类继承的第一个类是 .NET 对象类。
- **Dispatcher:** 一个抽象基类,用于绑定到一个线程上的类。与Windows窗体类似,WPF也要求仅从创建线程中调用方法和属性。WPF应用程序使用为人熟知的单线程亲和(Single-Thread Affinity,STA)模型,这意味着整个用户界面由单个线程拥有。从另一个线程与用户界面进行交互是不安全的。通过继承自DispathcerObject类,用户界面中的每个元素都可以检查代码是否在正确的线程上运行,并能通过访问调度程序为用户界面线程封送代码。
- **Dependency:** 所有支持依赖属性的类的基类。依赖属性可以依赖其他输入,例如主题和用户喜好。依赖属性与数据绑定，动画，资源和样式一起使用。
- **Visual:** 所有可见元素的基类都是Visual。这个类包含点击测试和转换等特性
- **UI Element:** 所有需要基本显示功能的WPF元素的抽象基类是UIElement。这个类提供了鼠标移动,拖放,按键的通道和起泡事件;提供了可以由派生类重写的虚显示方法;以及布局方法。WPF不再使用Window句柄,这个类就可以用Window句柄
- **FrameworkElement:** FrameworkElement派生自基类UIElement,实现了由基类定义的方法的默认代码

最后，所有 WPF 控件 textbox 、 button 、 grids 以及可以从 WPF 工具箱中想到的任何内容都继承自FrameworkElement类。

2.描述下WPF的总体架构?

- **User32:** 决定了哪个控件显示在屏幕上的哪个位置。
- **DirectX:** WPF内部使用DirectX 与驱动程序对话并渲染呈现内容。
- **Milcore:** 媒体集成库。 此部分是非托管代码，因为它充当 WPF 托管和 DirectX/User32 非托管 API 之间的桥梁。
- **Presentation core :** WPF 公开的低级 API，提供 2D 、 3D 、几何等功能。
- **Presentation framework:** 此部分具有高级功能，如应用程序控件、布局。 帮助您构建应用程序的内容等。

3.Style 和 ControlTemplate的主要区别是什么?

- 样式在控件上设置属性。
- ControlTemplate 是大多数控件的属性，用于指定它们的呈现方式。

详细地说，可以使用一种样式对一组属性的设置进行分组，以便重新使用它来标准化已有控件。 样式可以在控件上显式设置，也可以应用于所有特定类型。

控件模板可以通过样式设置或在控件上显式设置以更改其显示方式。 所有控件都有嵌入在 .net wpf 程序集中的默认模板（和样式）。

4.WPF 是建立在 Windows 窗体之上的还是完全不同的?

这两个是完全不同的技术。 它们确实为两个方向提供了一些互操作性层，但除此之外没有任何共同之处。 Windows 窗体或多或少是 Win32/MFC 之上的轻量级包装器，这意味着它在 .NET 中的可扩展性并非在所有情况下都那么好。 WPF 是一个从头开始实现的新 UI 框架。 在自定义现有类型时，WPF 也更加灵活。

- WPF 更适合创建“华丽”的 GUI。 只是它需要比 WinForms 更新的 .net 框架，并且需要兼容 dx9 或更高的 GPU。
- WinForms 仍然是一项强大的技术，通常可以以比 WPF 更快的速度开发，但是，最终，这两种技术都可以用来实现相同的目标。 WinForms 通常用于开发业务应用程序，而 WPF 通常用于创建更多基于最终用户的软件、应用程序等。

5.如何理解MVVM中的 View 和 ViewModel?

View是客户端界面、输入输出界面或用户界面。 收集了窗口、导航页面、用户控件、资源文件、样式和主题、自定义工具和控件的所有用户界面元素。

View不知道 ViewModel 和 Model，反之亦然，ViewModel 和 Model 不知道 View ，并且控件是完全分离的。

但是ViewModel知道View的需求。 它们通过数据绑定和依赖属性或多个属性进行通信。

ViewModel 是一个非可视类。 MVVM 设计模式不派生自任何基于 WPF 的类。 ViewModel 不直接知道 View。 View 和 ViewModel 之间的通信是通过一些属性和绑定进行的。

一个 View-Model 可以连接到多个模型，像一对多关系一样工作，并为 View 封装业务逻辑和数据。

6.如何在WPF应用程序中全局捕获异常?

使用“Application.DispatcherUnhandledException”事件。

请注意，仍有一些异常会导致应用程序崩溃，例如在尝试保存到数据库时出现堆栈溢出、内存耗尽或网络连接丢失等情况。

7.WPF中的x:Name和Name属性之间有什么区别?

它们不是同一件事。

- `x:Name` 是一个 xaml 概念，主要用于引用元素。当您为元素提供 x:Name xaml 属性时，“指定的 `x:Name` 将成为处理 xaml 时在底层代码中创建的字段的名称，并且该字段保存对对象的引用。”[MSDN],所以，它是一个设计时生成的字段，默认情况下具有内部访问权限。
- `Name` 是 FrameworkElement对象的现有字符串类型的属性, 以其他WPF元素中都包含此属性.

因此，这也意味着 `x:Name` 可以用于更广泛的对象。 这是一种使 xaml 中的任何内容都能够被给定名称引用的技术。

8.ListBox 与 ListView - 如何选择以及何时进行数据绑定?

ListView 是一个专门的 ListBox（继承自 ListBox）。 ListView允许指定不同的视图而不是直接列表。 可以滚动自己的视图，也可以使用 GridView（想想类似资源管理器的"详细信息视图"）。 它基本上是多列列表框，跟 windows 窗体列表视图的表现类似。

如果不需要 ListView 的附加功能，只是显示项目列表（即使模板很复杂），使用 ListBox就足够了。

9.说出使用WPF而不是Windows窗体的一些优点

使用 WPF 代替 Windows 窗体的优点：

- XAML 使更容易的创建和编辑 GUI，并允许在设计模式 (XAML) 和后台代码（C#、VB.NET 等）之间拆分工作。
- 数据绑定，使开发项目可以更清晰地分离数据和布局。
- 使用硬件加速来绘制 GUI，以获得更好的性能。

10.WPF中的命令设计模式和ICommand是什么?

ICommand 是 MVVM 的核心组件。 ICommand 在 MVVM 中经常使用，它提供了View和 ViewModel（用户界面和业务逻辑）之间的分离逻辑。 XAML 提供了一种通过 ICommand 更好地绑定 GUI 事件的方法。 ICommand 要求用户定义两个方法，*bool CanExecute* 和 *void Execute*。 CanExecute 方法只是告诉用户，我可以执行这个 Action 吗? 这对于控制 GUI 元素的可操作性非常有用。

ICommand 非常简单，但是也可以完在更加有趣和复杂的功能。 ICommand 将用户界面集成到业务逻辑中，或者在视图与视图模型之间进行直接通信。 它还为视图提供了更新模型/视图模型的机制。

11.什么是可冻结对象?

Freezable 是一种特殊类型的对象，具有两种状态：*unfrozen* 和 *frozen*。 解冻时，Freezable 的行为与任何其他对象一样。 冻结后，无法再修改 Freezable。

Freezable 提供了一个 `changed` 事件来通知观察者对对象的任何修改。 冻结 Freezable 可以提高其性能，因为它不再需要在更改通知上花费资源。 冻结的 Freezable 也可以跨线程共享，而未冻结的 Freezable 则不能。

尽管 Freezable 类有许多应用程序，但 WPF中的大多数 Freezable 对象都与图形子系统相关。

12.什么是MVVM?

MVVM（Model View ViewModel）是一个在WPF中制作应用的框架。 MVVM 与 MVC 框架相同。 它是一个三层架构，我们可以使用 MVVM 进行松耦合开发。

MVVM 由 John Gossman 于 2005 年推出，当时专门用于 WPF，作为 Martin Fowler 更广泛的演示模型模式的具体应用。 基于 MVVM 模式的应用程序的实现使用各种平台功能，这些功能以某种形式可用于 WPF、Xarmarin 移动端、Web和 Windows。 许多商业应用程序，包括 Microsoft Expression 产品，都是在 MVVM 之后构建的。

MVVM 的优势

- 模块化
- 测试驱动开发
- 分离 UI 和业务层作为视图和视图模型。
- Page和Window之间的代码共享。
- 易于维护。

MVVM 的特性列表

- 它分离了业务层和表示层，如 MVP 和 MVC
- 改进关注点的结构/分离（视图、视图模型和模型）。
- 实现更好的设计/开发人员工作流程。
- 增强简单性和可测试性。
- 通过XAML支持强大数据绑定功能
- 无需使用代码隐藏文件
- 提供多环境的应用开发能力。
- 强大的数据绑定、命令、验证等等。
- 设计者和开发者可以一起工作。

13.WPF中可视化树和逻辑树的区别是什么?

WPF 用户界面的元素是分层相关的。 这种关系称为逻辑树。 一个元素的模板由多个视觉元素组成。 这棵树被称为 VisualTree。

逻辑树 描述了用户界面元素之间的关系。 逻辑树负责：

- 继承 DependencyProperty 值
- 解析 DynamicResources 引用
- 查找绑定的元素名称
- 转发路由事件

视觉树包含所有逻辑元素，包括每个元素模板的所有视觉元素。 视觉树负责：

- 渲染视觉元素
- 传播元素不透明度
- 传播布局和渲染变换
- 传播 IsEnabled 属性。
- 进行命中测试
- 相对来源 (FindAncestor)

14.在WPF应用程序集中添加新文件时，Page和Window有什么区别?

- 页面旨在用于导航应用程序（通常带有后退和前进按钮，例如浏览器）。 页面必须托管在 NavigationWindow 或 Frame 中。
- Windows 只是普通的 WPF 应用程序 Windows，但可以通过 Frame 容器托管页面。

15.WPF中的样式和资源有什么区别?

资源用于针对多种类型的控件的属性，而样式一次只能为一种类型的控件定义属性。 我们还可以将不同的样式定义为一种公共资源的一部分。

这是一个开放式问题。 参考你的经验来提供相关的答案。

16.WPF中Dispatcher对象的用途是什么?

几乎每个 WPF 元素都具有线程关联性。 这意味着只能从创建该元素的线程访问此类元素。 为此，每个需要线程关联的元素最终都是从 DispatcherObject 类派生的。 此类提供名为 Dispatcher 的属性，该属性返回与 WPF 元素关联的 Dispatcher 对象。

Dispatcher 类用于在他的附加线程上执行工作。 它有一个工作项队列，负责在调度程序线程上执行工作项。

17.WPF中StaticResource和DynamicResource之间有什么区别?

- 在实际运行应用程序之前加载 XAML 期间，将解析 **StaticResource** 并将其分配给属性。 它只会被分配一次，并且忽略对资源字典的任何更改。
- **DynamicResource** 在加载期间将一个 Expression 对象分配给该属性，但直到运行时当 Expression 对象被要求提供值时才实际查找资源。 这会导致直到在运动时需要它时才查找资源。一个很好的例子是对稍后在 XAML 中定义的资源的前向引用。 另一个例子是直到运行时才会存在的资源。 如果源资源字典发生更改，它将更新目标。

[NEW]18.描述下WPF中使用的Prism框架

Prism旨在在WPF中构建具有单个代码库的应用程序。它有助于以模块化的方式开发客户端应用程序，以便将大型应用程序的复杂性划分为更简单的模块。

换句话说，“*Prism由Microsoft Patterns and Practices开发，提供指导，旨在帮助您更轻松地设计和构建丰富、灵活且易于维护的Windows Presentation Foundation（WPF）桌面应用程序。*”。

以下是基本架构：

1. **App.XAML:** 调用Application_Startup上的Boot Strapper。
2. **BootStrapper:** 一个调用Shell（Shell.XAML）的类文件，从而创建模块目录。
3. **Shell:** 类似一个具有区域的母版页，一个定义区域的容器
4. **Region:** 视图呈现的区域，类似视图占位符
5. **View:** 具有用户界面的XAML文件
6. **Module:** 每个模块可以有一个或多个视图，这些视图通过区域管理器注册到区域（在Shell中）

[NEW]19.WPF中的虚拟化是什么

WPF中的虚拟化技术提高了UI元素的渲染性能。通过应用虚拟化，布局系统确保在屏幕上只呈现容器的可见项。例如，列表控件可能有数千个项目，但虚拟化将减少对可见项目的渲染。

WPF中的VirtualizingStackPanel控件用于实现虚拟化。VirtualizingStackPanel的IsVirtualizing属性激活虚拟化。默认情况下，IsVirtualization属性设置为true。当IsVirtualization设置为false时，VirtualizingStackPanel的行为与普通StackPanel相同。

VirtualizingStackPanel.VirtualizationMode属性有两个值，Standard和Recycling。VirtualizationMode的默认值为Standard，这意味着VirtualizingStackPanel为每个可见项目创建一个项目容器，并在不再需要时（例如当项目滚动到视图外时）丢弃它。当ItemsControl包含许多项时，创建和丢弃项容器的过程会降低性能。在这种情况下，使用Recycling重用项目容器，而不是每次都创建一个新容器。

[NEW]20.WPF中有多少种位图效果

位图效果使设计人员和开发人员能够将视觉效果应用于渲染的WPF内容。例如，位图效果可以轻松地将阴影效果或模糊效果应用于图像或按钮。

以下是WPF中可用的位图效果：

1. BlurBitmapEffect
2. DropShadowBitmapEffect

[NEW]21.WPF有什么优缺点

优点：

- 丰富的多媒体集成：要在win32或Windows窗体应用程序中使用三维图形、视频、语音和丰富的文档查看，需要分别学习几种独立的技术，并在有限的原生支持的情况下将它们融合在一起。WPF应用程序可以在同一个编程模型中使用所有这些功能。
- 屏幕分辨率无关：WPF允许您缩小或放大屏幕上的元素，与屏幕分辨率无关。它使用矢量图形使应用程序的分辨率独立。
- 硬件加速：WPF构建在Direct3D之上，使用图形处理单元（GPU）而不是中央处理器单元（CPU）进行图形渲染。这为WPF应用程序提供了硬件加速的好处，允许更平滑的图形和增强的性能。
- 声明式编程：WPF使用可扩展应用程序标记语言（XAML）声明性编程来定义应用程序对象的布局和表示三维模型等。这允许设计人员与开发人员直接为同一个WPF应用程序进行协同作业。
- 丰富的组成和定制：WPF控件易于自定义。您不需要编写任何代码来以非常独特的方式自定义控件。WPF还允许您为外观完全不同的应用程序创建外观。
- 易于部署：WPF提供了部署传统Windows应用程序的选项（使用Windows Installer或Click Once）。这一特性并非WPF独有，但仍然是该技术的重要组成部分。
- 国际化与本地化支持：控件中的静态文本和String函数的返回数据根据最终用户操作系统指定的区域性和语言进行修改。

缺点：

- WPF的内置控件远比没有WinForms的丰富多样。
- WPF提供了一条新的学习曲线，相比而言Winform更简单些
- 没有MDI子窗口模式

[NEW]22.如何理解绑定中的“UpdateSourceTrigger”?

这是绑定上的一个属性，用于控制从目标到源的数据流，并用于双向数据绑定。默认模式是焦点改变但有许多其他选项可用时。

UpdateSourceTrigger提供以下选项：

- Default：这是默认值，大多数控件都会失去焦点而触发更新。
- LostFocus：直到焦点移出控件时触发更新
- PropertyChanged：每当目标属性发生更改时，就会进行值更新。
- Explicit：用于延迟源更新，直到用户通过单击按钮等强制执行时进行更新。

[NEW]23.什么是路由事件?

路由事件是基于控件的层次结构的，是WPF提供的一种新的基础结构，它允许事件通过隧道沿视觉树从根元素向内到达目标元素，或者冒泡到根元素。路由事件与正常事件类似。

路由事件有三种类型，如下所示：

- 冒泡事件：在控件层次结构中首先引发的隧道事件。这些事件由Root元素引发。这允许事件沿树向下隧穿。
- 隧道事件：冒泡事件是控件首先引发的事件，而不是控件层次结构中其他控件引发的事件。它允许泡泡上升到树，直到根元素。首先引发隧道事件，然后引发冒泡事件。
- 直接事件：直接事件通常由控件本身引发。此事件的行为与.NET常规事件相同。

WPF高级篇[8]

1.解释SelectedItem、SelectedValue和SelectedValuePath之间的区别?

- **SelectedItem** 属性返回您的列表绑定到的整个对象。 因此，假设您已将一个列表绑定到一组 `Category` 对象（每个 Category 对象都具有 Name 和 ID 属性）。 例如。 `ObservableCollection<Category>`。 `SelectedItem` 属性将返回当前选择的 `Category` 对

象。 然而，出于绑定目的，也可能有另一种情况，并不想得到整个 Category 对象绑定到列表绑定到结果，而是该 Category 对象上的单个属性的值（例如它的 `ID` 属性）。

- 因此，我们将 **SelectedValuePath** 属性和 **SelectedValue** 属性作为另一种绑定方式（将它们相互结合使用）。 假设您有一个 `Product` 对象，您的视图绑定到该对象（具有 ProductName、Weight 等属性）。假设在该 Product 对象上有一个 `CategoryID` 属性，并且希望用户能够从类别列表中为产品选择一个类别。 需要将 Category 对象的 ID 属性分配给 Product 对象的 `CategoryID` 属性。这就是“SelectedValuePath”和“SelectedValue”属性的用武之地。 使用 `SelectedValuePath='ID'` 将 Category 对象上的 ID 属性分配给列表绑定到的 Product 对象上的属性，然后将 `SelectedValue` 属性绑定到 DataContext 上的属性（即产品）。

下面的示例演示了这一点。 有一个 ComboBox 绑定到一个类别列表（通过 ItemsSource）。 将产品上的 CategoryID 属性绑定为选定值（使用 SelectedValue 属性）。 通过 SelectedValuePath 属性将此与类别的 ID 属性相关联。 并且只在 ComboBox 中显示 Name 属性和 DisplayMemberPath 属性）。

```
<ComboBox ItemsSource="{Binding Categories}"
              SelectedValue="{Binding CategoryID, Mode=TwoWay}"
              SelectedValuePath="ID"
              DisplayMemberPath="Name" />
```

2.WPF 中的 ControlTemplate 和 DataTemplate 有什么区别?

通常，控件是为了它自己而呈现的，并不反映底层数据。 例如，一个 `Button` 不会绑定到一个业务对象——它在那里纯粹是为了可以点击它。 但是，通常会出现“ContentControl”或“ListBox”，以便它们可以为用户呈现数据。

- 因此，“DataTemplate”用于为底层数据提供可视化结构，而“ControlTemplate”与底层数据无关，只是为控件本身提供可视化布局。
- “ControlTemplate”通常只包含“TemplateBinding”表达式，绑定回控件本身的属性，而“DataTemplate”将包含标准绑定表达式，绑定到其“DataContext”的属性（业务/域对象或 查看模型）。

3.Freezable.Clone() 和 Freezable.CloneCurrentValue() 方法有什么区别?

- `Clone` 实际上复制绑定表达式。 因此，如果对象的一个属性被绑定，它在副本中仍然是绑定的。
- 另一方面，`CloneCurrentValues` 只复制当前值，顾名思义。 不会保留绑定，因此如果修改了绑定的源，则不会更新副本中的值。

4.ObservableCollection 和 BindingList 有什么区别?

实际的区别在于 `BindingList` 用于WinForms，而 `ObservableCollection` 用于WPF。 从 WPF 的角度来看，BindingList 没有得到正确支持，除非真的必须，否则您永远不会在 WPF 项目中真正使用它。

5.冒泡事件和隧道事件之间的确切区别是什么?

WPF 为我们提供了许多不同的事件处理机制——它们是冒泡、隧道和直接。 这些都称为路由事件。

- **Direct event** - 最符合直观感受的就是直接路由事件了。 这是项目本身处理发生的事件的地方。一个很好的例子是在标准 WinForms 中处理鼠标按钮的 onClick 事件。 这是在 GUI 项中引发事件并由所述 GUI 元素处理的地方。

- **Bubbling Event** - 当事件没有被元素处理（比如文本框）并且事件“冒泡”到包含它的 UI 容器时，就会发生冒泡。 例如，假设有一个包含面板的窗口，在该面板内有一个Grid，在Grid内有一个TextBox。 如果TextBox未处理该事件，则它会移动、传递或“冒泡”到Grid上（因为Grid包含TextBox），如果未在该级别处理，则事件会进一步向上冒泡“ 树”（称为可视化树）到面板，在那里它可能会或可能不会被处理。 这个过程一直持续到它被处理或事件跳出最顶层的元素。
- **Tunneling** - 隧道与冒泡相反。 事件不是沿着可视化树“向上”移动，而是沿着可视化树向下传播到被视为源的元素。 隧道事件的标准 WPF 命名定义是它们都以“Preview”开头，例如 previewdownkey 和 previewmousebuttondown。 可以在它们到达“目标”元素的途中捕获它们并进行处理。

6.Threads 和 Dispatchers 是什么关系?

WPF 应用程序只有一个 UI 线程来处理所有 UI 交互和用户输入。 还有一个“隐藏”线程负责渲染，但通常开发人员不会处理它。

Dispatcher / Thread 关系是一对一的，即一个 Dispatcher 总是与一个线程相关联，可用于将执行分派到该线程。 `Dispatcher.CurrentDispatcher` 返回当前线程的调度程序，也就是说，当在工作线程上调用 `Dispatcher.CurrentDispatcher` 时，将获得该工作线程的调度程序。

Dispatchers 是按需创建的，这意味着如果访问 `Dispatcher.CurrentDispatcher` 并且没有与当前线程关联的调度程序，则会创建一个。

应用程序中的调度程序数量始终小于或等于应用程序中的线程数量。

7.ContentControl 和 ContentPresenter 之间有什么区别?

`ContentControl` 是包含其他元素并具有 `Content` 属性（例如，`Button`）的控件的基类。

- `ContentPresenter` 用于在控件模板中显示内容。
- `ContentControl`，可以直接使用（它应该用作基类），而 ContentPresenter 用来显示其控件模板中的内容部分。

个人经验（并非适用于所有情况，请自行判断）：

1. 在 `ControlTemplate` 中使用 `ContentPresenter`
2. 在 `ControlTemplate` 之外（包括 `DataTemplate` 和外部模板）尽量不要使用它们中的任何一个，如果必须使用，可以 `ContentPresenter` 优先
3. 如果需要创建一个承载内容的自定义“无外观”控件，并且无法通过更改现有控件的模板（这应该是非常罕见的）获得相同的结果，则可以将 `ContentControl` 子类化。

8.为什么需要依赖属性?

主要区别在于，普通 .NET 属性的值是直接从类中的私有成员 读取，而 DependencyProperty 的值在调用 *GetValue( ) 从 DependencyObject* 继承的方法。

当设置依赖属性的值时，它不会存储在对象的字段中，而是存储在基类 DependencyObject 提供的键和值字典中。 条目的键是属性的名称，值是您要设置的值。

依赖属性的优点如下：

- **减少内存占用**

当 UI 控件的 90% 以上的属性通常保持其初始值时，为每个属性存储一个字段是一种巨大的消耗。 依赖属性通过仅在实例中存储修改的属性来解决这些问题。 默认值在依赖属性中存储一次。

- **值继承**

当访问依赖项属性时，将使用值解析策略来解析该值。 如果没有设置本地值，则依赖属性会向上导航逻辑树，直到找到一个值。 当您在根元素上设置 FontSize 时，它适用于下面的所有文本块，除非在元素中覆盖该属性值。

- **更改通知**

依赖属性具有内置的更改通知机制。 通过在属性元数据中注册回调，您会在属性值更改时收到通知。 这在数据绑定中会使用到。