

1. System.Threading.**DispatcherObject**类,通过继承此类，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。

*WPF应用程序使用的是STA模型（Single-Thread Affinity，单线程亲和），以为着只有创建元素的线程可以操作元素（一般就是UI线程）*

2.System.Windows.**DependencyObject**类，提供对依赖属性的支持。

3.System.Windows.Media.**Visual**类，每个元素本质都是一个Visual对象，封闭绘图指令、绘画额外细节。

*任何继承自Visual的类，都能在窗口上显示出来。*

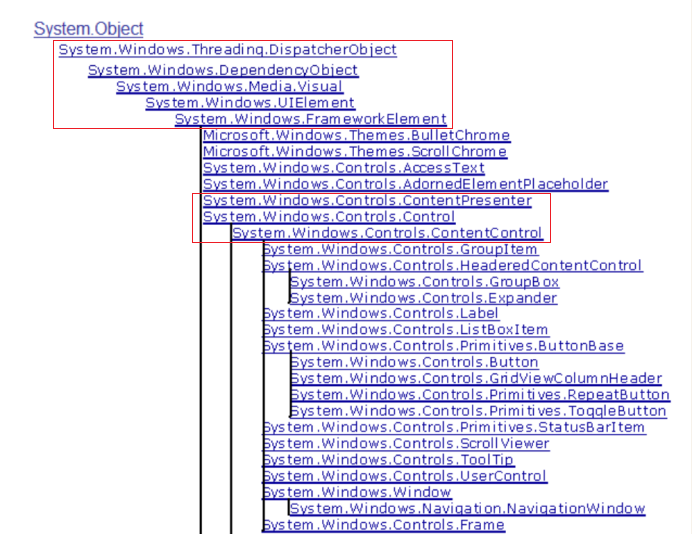
1. System.Windows.**UIElement**类，提供WPF本质特征的支持，如布局、输入、焦点、及事件

***LIFE****: Layout | Input | Focus | Event*

1. System.Windows.**FrameworkElement**类，该类实现了一些全部由UIElement类定义的成员；提供数据绑定、动画及样式的支持

*如：UIElement类为WPF布局系统设置了基础，但FrameworkElement类提供了支持它的重要属性（HorizontalAlignment，Margin等）*

* 1. System.Windows.Shapes.**Shape**类，基本形状类继承于此类。
  2. System.Windows.Controls.**Control**类，是可与用户进行交互的元素；为字体、前背景色及模板提供支持。
     1. System.Windows.Controls.**ContentControl**类，所有具有单一内容类控件的基类。
     2. System.Windows.Controls.**ItemsControl**类，所有显示选项集合控件的基类。
  3. System.Windows.Controls.**Panel**类，所有布局窗口的基类。
  4. System.Windows.Controls.**Decorator**类，所有装饰类的基类，







## FAQ

### 基本特征

1. 元素、控件怎么理解？

继承自UIElement或者FrameworkElement的对象，称为**元素**，是更广泛的含义，比如按钮、图形、边框；

继承自Control的对象一般称为**控件**，其特点是可以和用户交互，控件本身也继承自元素，所以是更特殊的一类元素；比如按钮；

*另外，在WPF开放领域，自定义元素和自定义控件，一般可以互换使用，因为绝大多数情况下，你需要的都是“自定义控件”*

1. WPF为什么要设计依赖项属性而不是用传统属性

更高效的保持机制（静态类，属性值继承等减少对象的存储）

支持附加功能：更改通知，属性值继承

1. WPF为什么要设计路由事件而不是用传统事件

根据桌面程序常常需要的业务需求，设置路由事件以支持**隧道**和**冒泡**；

1. WPF布局的原则

不应显式设置元素（如控件）的尺寸

不应使用屏幕坐标指定元素的位置

布局容器的子元素“共享”可用的空间

可嵌套的布局容器

### XAML语法相关

1. XAML中使用花括号是什么含义？

表示其值得计算使用的是标记扩展，在代码内部即是调用静态扩展方法

如{x:static xxx}，实际就是调用StaticExtension的相关方法获取值；

1. 属性值的优先级：从低到高

默认值（FrameworkPropertyMetadata对象设置的值）

继承而来的值（设置了FrameworkPropertyMetadata.Inherits，并且包含层次中某个元素提供了值）

来自主题样式的值（模板中使用的themes/generic.xaml）

来自项目样式的值：style中定义

本地值（XAML或者代码中直接设置的值）

1. XAML属性设置值的方法有哪几种？

**简单属性和类型转换器**：字符串设置，WPF选择合适的类型转换器进行转换；

**属性元素语法**：使用【Parent.PropertyName】添加子元素；

**扩展标记**：使用【{标记扩展类 参数}】的绑定；实际实现是基于C#中的静态扩展方法；

**附加属性**：使用【定义类型.属性名】的方式；

**嵌套元素**：使用一个集合标签，内部添加多个子元素的方式；

1. XAML绑定表达式的方式有哪几种？

绑定到元素对象： { Binding ElementName=xxx, Path=Xxx }

绑定到非元素对象： { Binding Source={x:Static SystemFonts.IconFontFamily},

Path=**Source**} // this **Source** is a property of IconFontFamily

{ Binding RelativeSource={RelativeSource FindAcestor,

AncestorType={x:Type Windows}},

Path=Title }

Use DataContext to set the same binging object in parent scope

### 样式

1. 静态资源和动态资源的区别？

ToDo

1. 设置器有哪几种？在什么时候使用？

在样式定义中使用。

**属性设置器**：只能设置依赖项属性；

<Setter Property=”XXX”>

<Setter.Value>

Xxx

</Setter.Value>

</Setter>

**事件设置器**：只能设置路由事件；（WPF不推荐使用事件设置器，建议使用**事件触发器**）

<EventSetter Event=”Xxx\_Event” Handler=”Xxx\_event\_hanlder” />

1. 触发器有哪几种？

**属性触发器**：关联在**依赖项属性**的触发器；

**事件触发器**：关联在**路由事件**的触发器。

1. 什么是行为？

封装了一些UI功能，从而可以不必编写任何代码就能够应用到元素上；

可以这样想象，类似AOP，从一个横切方向提取共性的特征，从而实现某类通用的功能，最常见的行为就是拖拽功能。

### 控件模板

1. 为什么说WPF控件被称为无外观控件？

控件只定义了**功能**，但**没固定**的外观；

*外观可以用控件模板动态改变*；（<ControlTemplate>）

1. 什么是控件模板？

设置控件的基本结构和基本外观的模板；

1. TemplateBinding作用？

在控件模板中获取运行时绑定对象的属性的方法；

（仅限控件模板中使用，<ControlTemplate>标签内部）

1. TemplateVisualState和TemplatePart特性

在自定义控件时，用于表示该自定义控件应该具有的某些特性；

TemplateVisualState：具有特定可视化状态

TemplatePart：具有特定名称的元素；

1. 什么元素可以使用自定义模板？

Control元素及其子类。

因为Template属性的定义是在Control中的，因此可以使用自定义的ControlTemplate设置其Template属性进行替换。ContentControl和ItemsControl都是其子类，因此都可以使用自定义的控件模板

1. 扩展标准控件的方式有哪几种？有什么不同？

样式：修改控件的公开属性；

内容控件：组合不同的标准控件，使其支持更多的功能，从而实现扩展；

控件模板：改变控件的内部结构

数据模板：是一块定义如何显示绑定的数据对象的XAML标记；

ContentControl.ContentTemplate

ItemsControl.ItemTempate(*实际上就是对每个项设置ContentControl.ContentTemplate，最终替换的是ContentPresenter*)

样式可以修改控件**公开的属性**，从而改变控件的外观；

模板可以修改控件**内部的结构**，从而彻底改变控件的显示方式；

### 自定义元素

1. 什么时候需要自定义元素？

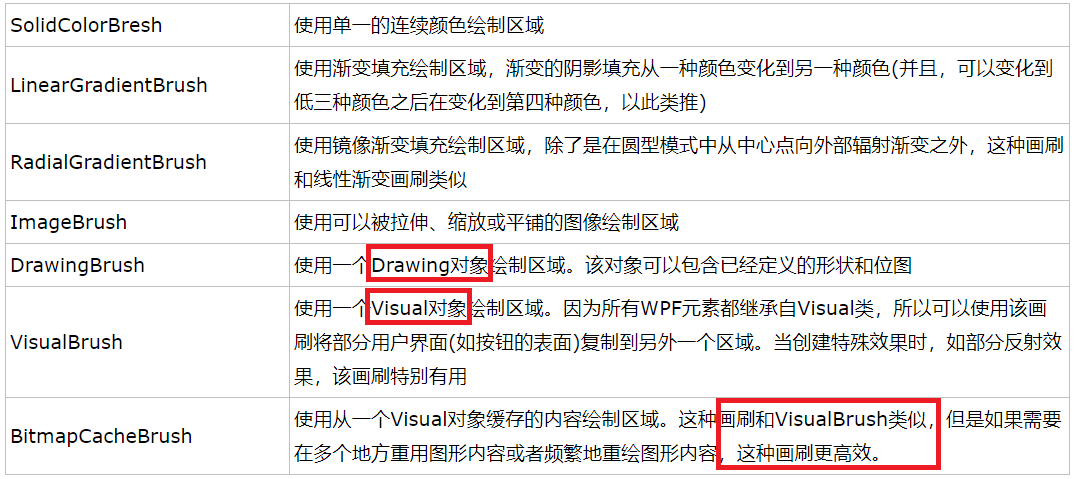
当需要测定改变底层的功能时，或者实现一个标准控件不支持的全新功能的时候，考虑使用自定义元素；（或者也可称为自定义控件）

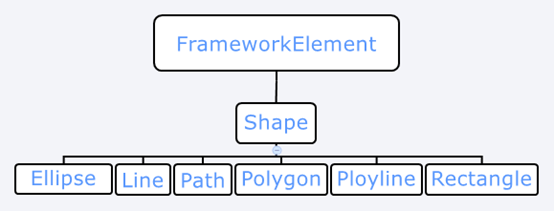
1. 自定义元素和自定义控件模板有什么区别？
   1. 自定义控件模板只能用于Control子类，因为绑定的是Template属性；
   2. 自定义控件模板改变的是外观（虽然可以是结构上重新改变外观），但是自定义元素是从根本上提供一个新的功能，是提供新的业务功能。
2. 自定义元素的基类一般是什么？

一般情况下，是**Control**类，其内部实现还是定义一个默认的ControlTemplate；自定义元素内部实现模板与逻辑的分离。这时候更适合称为自定义“**控件**”；

如果需要更低级的自定义元素，则是**FrameworkElement**，这时候需要重写OnRender()方法，是真正意义上的自定义“**元素**”。

### 画刷、图画与动画





Path.Data : **Geometry**对象；



PathGeometry.Figures: **PathFigure**对象集合（包含大量线段的不间断线条绘制的形状）

PathFigure.Segments: **PathSegment**对象集合



1. 绘制矢量形状的相关对象有哪些，如何绘制？

Shape： 形状

*【Eclipse | Line |......】*

Path： 路径

**Geometry**： 图形（几何图形）

*【EclipseGeometry | LineGeometry | ......】*

**PathGeometry**

PathFigure: 画像（*自定义的名称，为了和Geometry图形区分*）

PathSegment: 线段

**应用WPF标准机制，实现WPF标准元素的绘制**

1. 将矢量形状绘制到图像中的相关对象有哪些，如何绘制？

DrawingBrush：继承自Brush，因此可以设置画刷，使用画刷绘制到元素上；

DrawingVisual：在可视化对象（visual）中放置图画；

DrawingImage：在Image元素中用于设置Image.Source属性；

Drawing： 图画

GeometryDrawing 几何图画

**Geometry：这里可以使用上面定义的PathGeometry对象**

ImageDrawing 位图图画

VideoDrawing 视频图画

GlyphRunDrawing 字形图画

DrawingGroup 图画组合

1. 如何实现绘制可视化对象？

自定义类继承Visual类；

重写VisualChildCount，GetVisualChild，并调用AddVisualChild和AddLogicalChild方法；

使用DrawingVisual类的RenderOpen()方法，获取DrawingContext；

调用DrawingContext的DrawLine、DrawRectangle、DrawGemotry等方法，实现形状的绘制；

调用DrawingContext.Close()方法关闭绘制；

1. 绘制图画的性能比较？

超过1000个图形时，使用WriteableBitmap；

--> 使用一个元素封装多个Visual对象；

--> 使用Path元素， Geometry中使用StreamGeometry

Geometry中使用PathGeometry

--> 使用其他shape元素

### 数据绑定

1. 相关用法

Set Parent’s DataContext;

{Binding Path=xxx}

绑定的属性需要支持更改通知，才能双向绑定（INotifyPropertyChanged）

只要属性实现IEnumerable接口，则可以绑定到ItemsControl.ItemsSource

1. 格式化绑定数据的集中方法

* StringFormat：{Binding Path=xxx, StringFormat={}{0:C}} (*两个中括号是需要的*)
* IValueConverter：{Binding Path=Xxx, Convert={StaticResource YourConverter}}

1. ItemsControl常用的属性

|  |  |
| --- | --- |
| ItemSource | 数据源 |
| DisplayMemberPath | 设置显示的数据的相关属性 |
| ItemStringFormat | 格式化转换器 |
| ItemContainerStyle | 每个项应用的样式 |
| ItemContainerStyleSelector | 每个项可根据该类返回的值选择应用什么样式 |
| ItemTemplate | 每个项应用的数据模板 |
| ItemTemplateSelector | 每个项可根据该类返回的值选择应用什么数据模板 |
| ItemsPanel | 设置列表项放到什么容器中 |
| GroupStyle | 数据源如果实现了分组，可设置分组的样式（具体样式在该对象的各个属性中定义，如ContainerStyle、ContainerStyleSelector、HeaderTemplate、HeaderTemplateSelector和Panel） |
| GroupStyleSelector | 每个分组可根据该类返回的值选择应用什么样的GroupStyle |

1. ItemContainerStyle和ItemTemplate使用的差异？

ItemContainerStyle应用的是样式，需要样式本身公开了属性才能修改，如应用在ListBoxItem上，则只能修改ListBoxItem本身公开的属性；

ItemTemplate则是将每一项的数据模板进行了替换，实际上等于重新设置了ListBoxItem的ContentTemplate属性；

1. 什么是数据视图？

WPF实际并不是直接将数据源和XAML元素绑定的，而是生成了一个中间件，CollectionView，数据视图层；该类的作用是分离数据源和元素之间的直接依赖，这样就可以在数据视图层进行过滤、分组、排序等操作而不影响数据源。

1. DataTemplate实际上是替换可视化视图中的哪一部分？

替换ContentPresenter部分。

### 模板总结

1. 占位符都有哪些？

ContentPresenter，ContentControl的内容占位符；**在ContentControl的ControlTemplate中使用。**

ItemsPresenter，ItemsControl的内容占位符，最终会被使用ItemsControl.ItemsPanel设置的元素替换该占位符；**在ItemsControl的ControlTemplate中使用。**

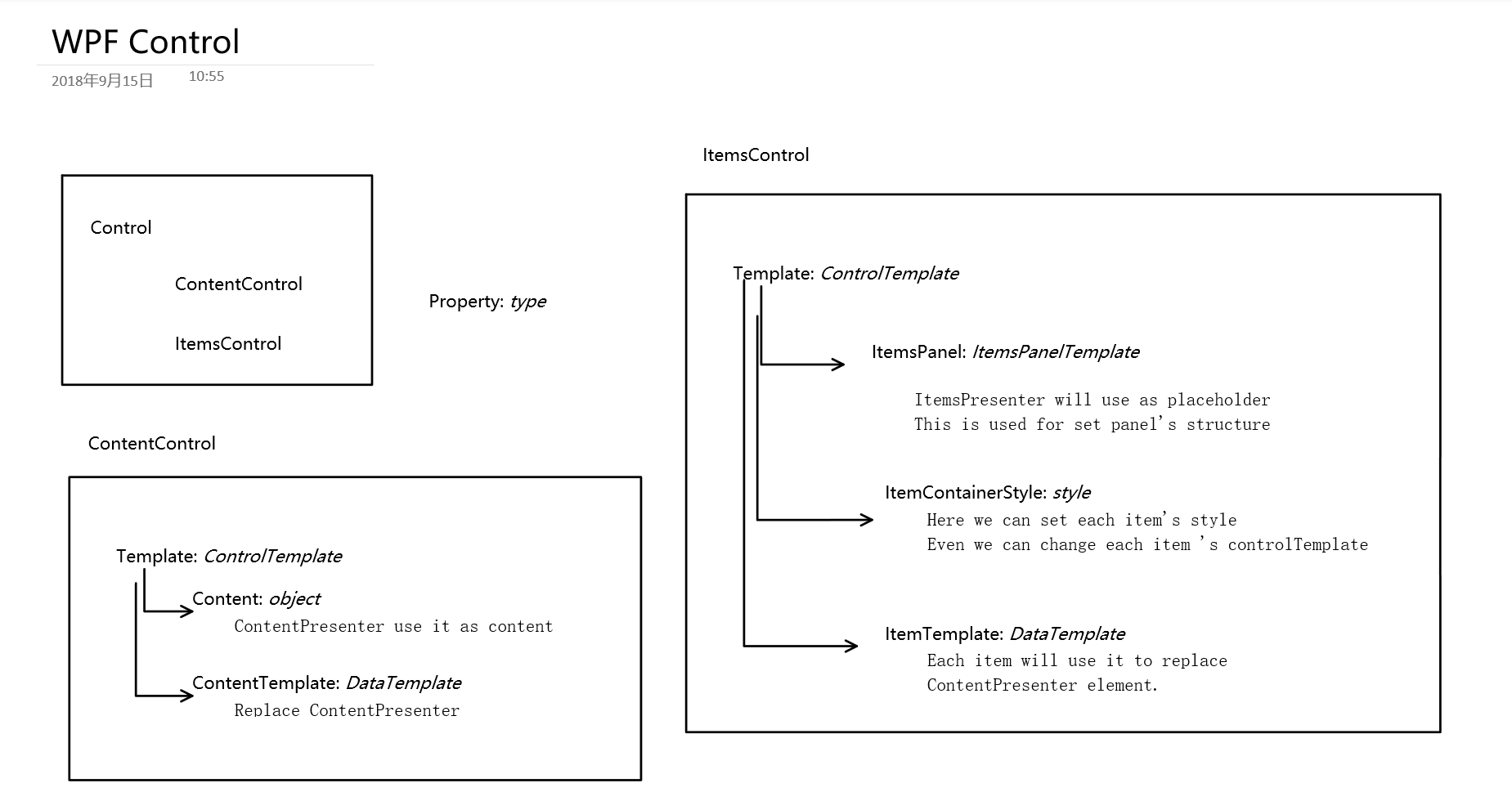
1. 几个模板之间的关系？

模板的类型有三种：

ControlTemplate：定于于Control基类的属性，用于设置控件的外观；

ItemsPanelTemplate：只用于ItemsControl，用于设置所有项容器面板的控件模板属性。(*ItemsPanelTemplate describes how ItemsPresenter creates the panel that manages layout of containers for an ItemsControl*.)，主要是为了管理列表项的布局；

DataTemplate：数据模板，可以认为是用于替换ContentPresenter的内容；



### 动画

1. WPF提供了哪几种动画模式

两种

1. 基于属性

本质上，这种WPF动画只不过是在一段时间间隔内修改依赖项属性值的一种方式。

* 1. 基于线性插值的动画（线性值变化）
  2. 基于关键帧的动画（类似枚举值变化）
  3. 基于路径的动画（基于PathGeometry）

1. 基于帧

一种更低级但是有更多控制的动画模式，一般不建议使用，除非以上几种动画模式都无法实现；例如模型碰撞动画效果等。

1. WPF动画的实现效果和应用情况？

基本的效果会使用到，比如鱼眼，登录动画等，但是复杂的动画，一般使用第三方软件实现，如使用视频内嵌，或者3D软件编辑的程序内嵌。

### 流文档与打印

WPF的流文档在实际使用中并不多见。其实现类似于HTML中的各个标签，如<p><b><H1>等等，当然还提供一些其他高级功能，如编辑等。

但是实际应用中很少看到有保存为.xaml的文档；虽然WPF的流文档还支持打开一些通用兼容格式，如.rtf，.xps；但是依然很少使用。

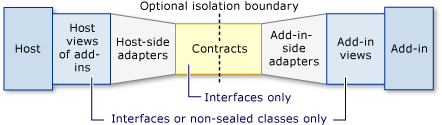
### 多线程

WPF是单线程亲和程序，意味着后台线程不能更新UI元素，因此WPF自身提供一个BackgroundWorker类，可以支持后台线程进度通知，切换线程更新数据等功能。

### 插件模型

MEF：Managed Extensibility Framework 或 MEF 是一个用于创建可扩展的轻型应用程序的库。通过简单地给代码附加“[Import]”和“[Export]”标记，我们就可以清晰地表明组件之间的“服务消费”与“服务提供”关系，MEF在底层使用反射动态地完成组件识别、装配工作。

MAF：Managed Add-in Framework是一个更加可靠但也更加复杂的插件模型。通过两层适配，使得框架有更大的灵活性（没有什么是不能通过增加一层来解决的）和安全性（每个插件被加载到新的应用程序域中）；



插件与宿主交互称为自动化（automation），一般是通过初始化组件时传递一个宿主适配器到插件中。

WPF还支持插件可视化，即宿主应用程序中显示实际运行在插件应用程序域中的控件，并与之交互。

WPF 单元测试？