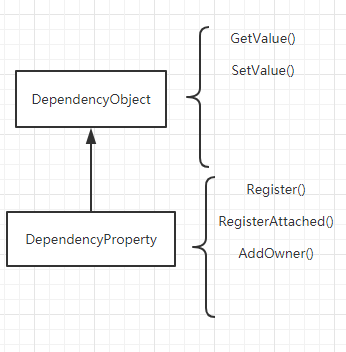
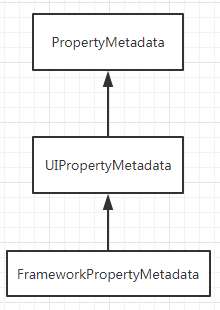
**依赖项属性**：





类继承图

根据多个提供对象（动画、父类元素、绑定、样式和模板）来决定值，同时该值也能及时响应变化。

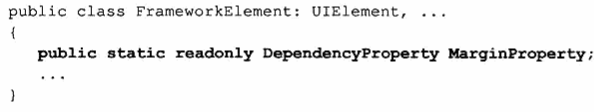
依赖项属性的价值：

节省内存，对象内存不是分配给每个属性和实例，只有值发生改变才分配内存，默认值在所有实例之间共享。

**一 创建依赖项属性**

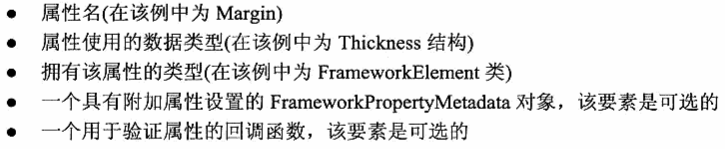
**1定义依赖项属性**（只能为继承自DependencyObject类的依赖对象添加）

定义为静态、只读



**2注册依赖项属性**

在与依赖项属性关联的类的静态构造函数中注册。调用DependencyObject.Register()方法，传入的参数（以最多的为例）：



**3添加属性包装器**

是使用DependencyObject的SetValue(),GetValue()

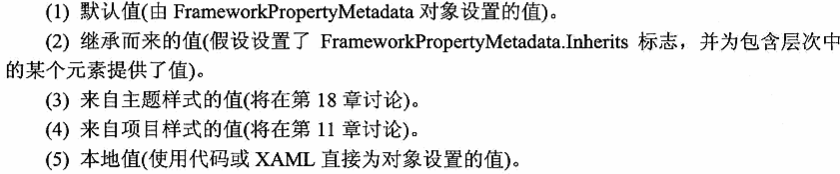
如何使用依赖项属性

1更改通知

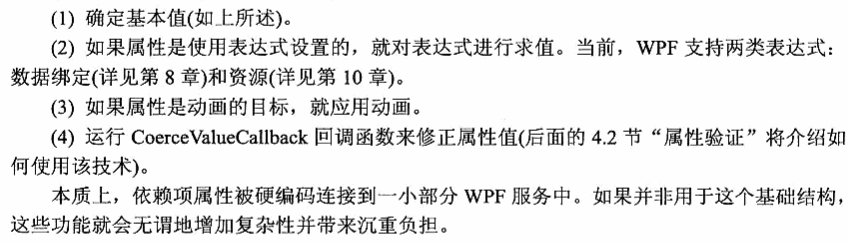
属性值发生变化→触发OnPropertyChangedCallback()→调用PropertyChangedCallback()

2动态值识别

基本值的决定



最终值的决定



共享的依赖项属性

附加的依赖项属性

**二 属性验证**

验证回调

ValidateValueCallback回调函数：接收或拒绝新值

CoerceValueCallback回调函数：将新值修改为更能被接收的值

强制回调

此处可看圣殿骑士的demo

**路由事件**：

**理解路由事件**

1定义、注册和封装路由事件

与注册依赖属性类似

2共享路由事件

与依赖属性类似。如UIElement和ContentElement都有MouseUp事件，而该事件由Windows.Input.Mouse类定义。



3引发路由事件：（非传统的.NET事件封装器引发，而是用RaiseEvent()方法）

事件处理程序的参数解析：

Eg.



object sender:提供引发该事件的对象的引用；

EventArgs e:该对象与其他所有可能很重要的附加细节绑定在一起；MouseButtonEventArgs继承自RoutedEventArgs

4处理路由事件

1. XAML中直接连接
2. 代码连接事件（通过调用UIElement.AddHandler()）



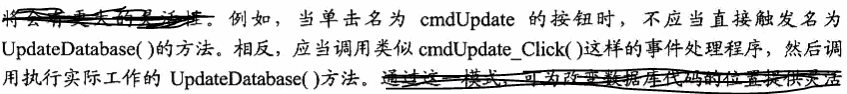
1. 直接使用UIElement.AddHandler()



事件命名：元素名\_事件名 ：img\_MouseUp；如果元素没有Name，可以如下：



额外事件处理代码层：增加灵活性，不同元素可以通过不同的事件处理程序连接到同一个方法。



优缺点：

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| 第一种 | 界面设计与业务逻辑分离； | 容易造成XAML的臃肿；  总是在窗口对象第一次实例化时就关联相应的事件处理程序； |
| 第二种 | 可以在窗口生命周期的某一时间而关联事件； | 无法明确指出MouseUpEvent事件是由Image类提供的； |
| 第三种 |  |  |

**事件路由**

直接路由事件

冒泡路由事件:MouseDown,首先由被单击的元素引发，往父元素向上引发，直到WPF到达元素树的顶部

隧道路由事件:PreviewKeyDown,首先在窗口级别上，然后是更具体的容器，直至到达当按下键时具有焦点的元素

RoutedEventArgs类

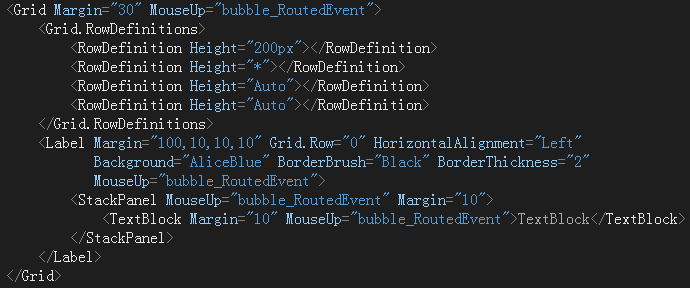
处理挂起的事件

附加事件

**冒泡事件和隧道事件的执行顺序**：

**冒泡事件**：

XAML文件↓



对Label标签MouseUp，输出结果：

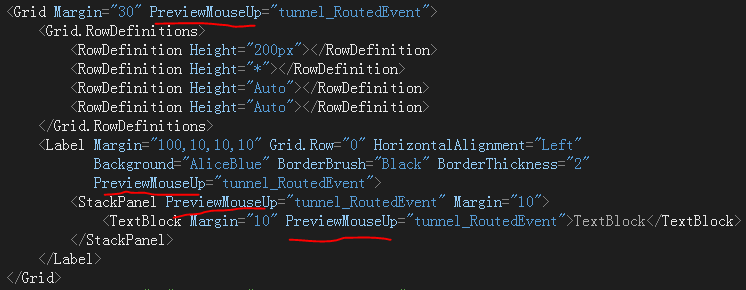


分析：从绑定MouseUp事件的最外层元素开始执行，一直到source元素（最先接收到

MouseUp的元素），本处是Grid→Label

**隧道事件**

XAML文件↓



对Label标签PreviewMouseUp，输出结果：



分析：从source元素开始执行，一直到绑定PreviewMouseUp的最外层元素，本处是Label

→Grid

**WPF事件**

生命周期事件

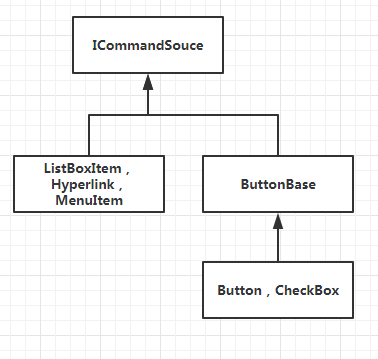
输入事件

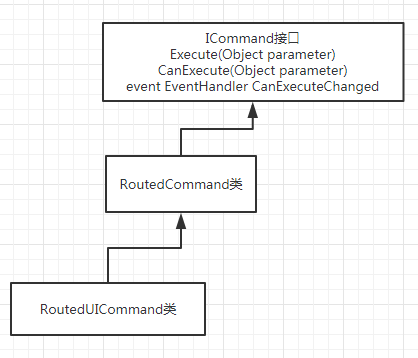
鼠标

键盘

多点触摸

**命令：**





类继承图

**1理解命令**

路由事件中，元素只能连接事件处理程序，有多个事件处理程序调用相同的应用程序方法的情况。有几点不足。1：事件是低级元素，可能无法满

足实际业务需要；2：事件处理程序无法重复使用；3 管理用户界面的状态很难

因此出现了控件连接命令，命令可以重复使用；并且当连接的命令不可用时，可以自动禁

用控件。

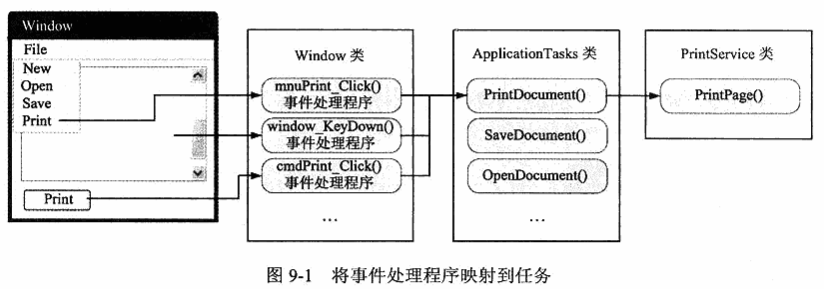
相同的功能可能会有多个路由事件同时连接，就需要编写多个具有相同功能的不同事件处理

程序，如果涉及到状态的改变（是否可用），那将会很麻烦。WPF通过命令模型解决问题：

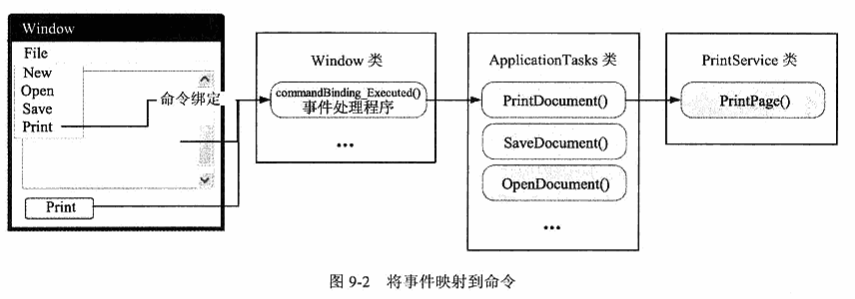
将事件委托到适当的命令

使控件的启用状态和相应命令的状态保持同步（不需要手动改变控件的状态）

**路由事件的控件连接方式**



**命令的控件连接方式**



**1 System.Windows.Input.ICommand：**

Execute(Object parameter)

包括应用程序处理逻辑或者可以触发在应用程序其他地方处理的事件

CanExecute(Object parameter)

返回命令的状态，可用---true

CanExecuteChanged

当命令状态改变时引发CanExecuteChanged事件。指示使用命令的控件调用CanExecute

方法，改变状态。

**2 System.Windows.Input.RoutedCommand ： System.Windows.Input.ICommand**

创建自己的命令时使用，私有地实现了ICommand接口，在Execute(),CanExecute()方法中加了额外的IInputElemenmt类型的参数target，表示开始处理事件的元素。

另外增加了命令名称（Name）、包含命令的类（OwnerType）、触发命令的操作集合（InputGestures）

**3 RoutedUICommand ： RoutedCommand**

添加了Text属性，用于用户界面中的提示。该属性比直接在控件上定义文本的好处是可以在某个位置执行本地化。

**命令库**

WPF自带的命令，

ApplicationCommands ,NavigationCommands ,EditingCommands ,ComponentCommands ,MediaCommands

这些单独的命令对象不具有实际的功能，需要自己去实现，但可以自动绑定快捷键。

**执行命令（一）**

1为派生自ICommands的控件设置Command属性：

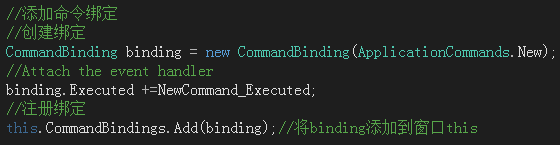


2在后台代码需要

当命令被触发时执行什么操作；

如何确定命令是否能被执行（如连接事件处理程序）

规定命令的作用范围



此处的事件冒泡为什么没有出现，我已经在按钮和外层的label都绑定了同一个Command啊？

**执行命令（二）--直接调用命令**

①并非只能使用实现了ICommandSource接口的类触发命令。可以通过Execute（）方法直接调用任何事件处理程序的方法：



第一个参数：

第二个参数：目标元素

②也可在关联的CommandBinding对象中调用Execute（）方法



这两种方式无法自动改变命令的状态，需要通过RoutedCommand.CanExecuteChanged() , RoutedCommand.CanExecute()方法处理。

**高级命令**

**自定义命令**

实例化一个新的RountedUICommand对象

在不同位置使用相同的命令

**跟踪和翻转命令**

命令

**2 WPF命令模型**

组成部分：

命令：

命令绑定：

命令源：用户操作的元素：如MeauItem，Button

命令目标：用户可以看到操作结果的地方，如Paste命令中的TextBox。

**元素绑定**

**一 元素绑定元素**

数据源：WPF元素 数据目标：WPF元素

绑定方式：

XAML

代码

绑定错误

当出现错误的绑定时，会在输出窗口输出信息。

绑定模式

OneWay, TwoWay, OneTime, OneWayToSource, Default

绑定表达式只能用于设置依赖项属性（也就是数据目标必须是依赖项属性），但通过设置Mode. OneWayToSource，可克服限制，但前提是提供数值的属性本身是依赖项属性。

使用代码检索绑定

1）使用BindingOperation.GetBinding()获取Binding对象



第一个参数：数据目标；第二个参数：：数据目标绑定的属性

2）使用BindingOperation.GetBindingExpression()获取BindingExpression对象

再通过BindingExpression.ResolvedSource属性获取**数据源**对象，而通过数据源可获取其所有属性的值。

多绑定

绑定更新（会影响程序性能）

从源到目标会立即更新，不用失去焦点；

从目标到源不一定。

可通过设置UpdateSourceTrigger枚举类型，设置目标到源的更新方式。

其中UpdateSourceTrigger.Explicit完全由自己控制，比如点击apply按钮之后再更新，通过获取BindingExpression对象设置。

**二 绑定到非元素对象**

必须是公有属性

Source

RelativeSource

DataContext

**控件**

内容控件ContentControl类层次结构

