# 一、理解路由事件

路由事件是一种可以针对元素树中的多个侦听器而不是仅仅针对引发该事件的对象调用处理程序的事件，也就是说，触发事件源的父级或子级如果都有对该事件的监听，则都能触发事件。路由事件与一般事件的区别在于：路由事件是一种用于元素树的事件，当路由事件触发后，它可以向上或向下遍历可视树和逻辑树，他用一种简单而持久的方式在每个元素上触发，而不需要任何定制的代码（如果用传统的方式实现一个操作，执行整个事件的调用则需要执行代码将事件串联起来）

# 二、路由事件一般使用以下三种路由策略：

路由事件的特殊性在于其传递性，WPF中的路由事件分为三种。

与普通的.NET事件类似的直接路由事件(Direct event)。它源自一个元素，并且不传递给其他元素。例如，

MouseEnter事件(当鼠标移动到一个元素上面时触发)就是一个直接路由事件。

在包含层次中向上传递的冒泡路由事件(Bubbling event)。例如，MouseDown事件就是一个冒泡路由事件。

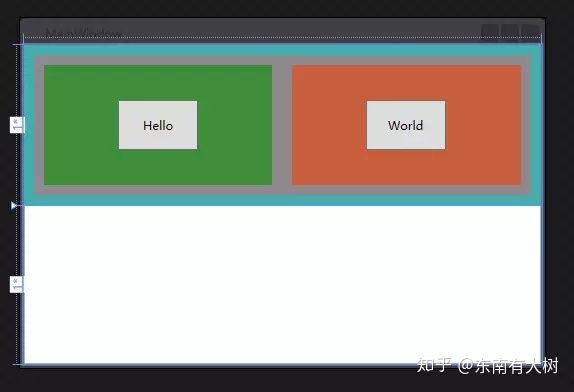
它首先被单击的元素触发，接下来就是该元素的父元素触发，依此类推，直到WPF到达元素树的顶部为止。

在包含层次中向下传递的隧道路由事件(Tunneling event)。例如PreviewKeyDown就是一个隧道路由事件。

在一个窗口上按下某个键，首先是窗口，然后是更具体的容器，直到到达按下键时具有焦点的元素。

1. **冒泡：由事件源向上传递一直到根元素**
2. 程序设计思路是，将多个Grid嵌套起来，构成父子结构，在最底层支Grid上定义一个按钮，绑定一个单击事件，并且按钮的所有低级元素都绑定该事件，它们使用同一个事件处理程序；再定义一个ListBox，当事件处理程序执行的时候，打印出事件激发者的名字，这样便能看到事件执行的顺序

下图是界面设计：



1. 下面是XMAL代码，按钮和每一个Grid都绑定了Button.Click="Btn\_Click"

<Window x:Class="WPF\_CODE.MainWindow"

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

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

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

<!--最外层的Grid-->

<Grid Button.Click="Btn\_Click" Name="Grid\_1" Background="#FF43AEAE">

<!--定义两行-->

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<!--第二层Grid-->

<Grid Button.Click="Btn\_Click" Name="Grid\_2" Margin="10" Background="#FF8D888D" Grid.Row="0">

<!--定义两列-->

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<!--第三层左侧Grid-->

<Grid Button.Click="Btn\_Click" Name="Grid\_3\_Left" Grid.Column="0" Background="#FF3D8F3A" Margin="10">

<!--添加一个按钮-->

<Button Button.Click="Btn\_Click" Name="ButtonLeft" Width="80" Height="50" Content="Hello"/>

</Grid>

<!--第三层右侧Grid-->

<Grid Button.Click="Btn\_Click" Name="Grid\_3\_Right" Grid.Column="1" Background="#FFC95E3E" Margin="10">

<!--添加一个按钮-->

<Button Button.Click="Btn\_Click" Name="ButtonRight" Width="80" Height="50" Margin="10" Content="World"></Button>

</Grid>

</Grid>

<!--定义一个ListBox，用于输出结果-->

<ListBox Name="Print\_List" Grid.Row="1"/>

</Grid>

</Window>

1. 下面是后端代码

**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** WPF\_CODE

{

*/// <summary>*

*/// MainWindow.xaml 的交互逻辑*

*/// </summary>*

**public** **partial** **class** **MainWindow** : Window

{

**public** MainWindow()

{

InitializeComponent();

}

*// 事件处理程序*

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

{

*//sender是指由谁激发了这个事件处理程序，它可以获取到触发对象*

*//通过(sender as FrameworkElement).Name转换，将触发控件的名称拿出来，*

**string** message = "触发者："+(sender **as** FrameworkElement).Name.ToString();

**this**.Print\_List.Items.Add(message);

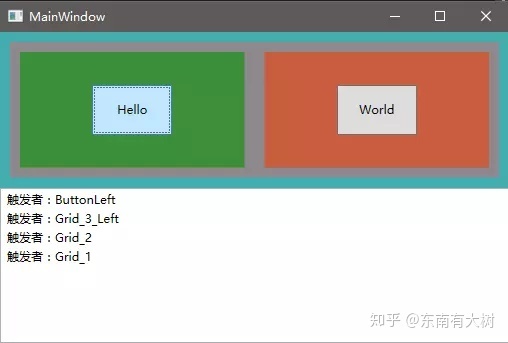
*//e包含了与事件相关的一些参数，e.Handled如果设置为True，则表示冒泡不再继续*

*//e.Handled = true;*

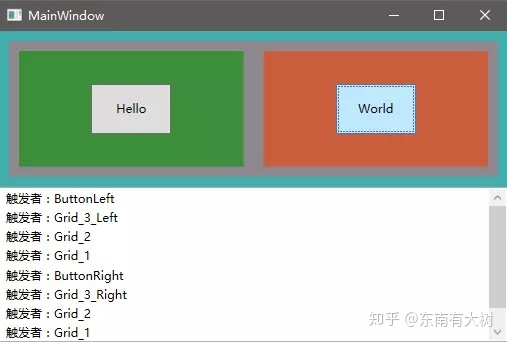
}

}

}



点击右侧按钮，结果如下：

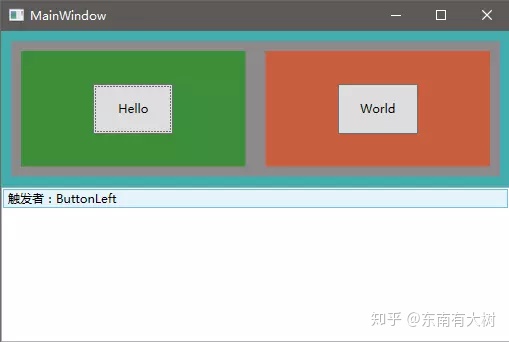


可以看出，事件首先在**源元素**上触发，然后从每一个元素向上沿着树传递，直到到达根元素为止（或者直到处理程序把事件标记为已处理为止），从而调用这些元素中的路由事件

如果将事件处理程序里的e.Handled = true;代码放开，效果如下

*//e包含了与事件相关的一些参数，e.Handled如果设置为True，则表示冒泡不再继续*

e.Handled = **true**;



即事件处理程序只要被触发一次，并不会发生冒泡

**直接：只有事件源才有机会响应事件**

事件仅仅在源元素上触发，这个与普通的.Net事件的行为相同，不同的是这样的事件仍然会参与一些路由事件的特定机制，如事件触发器等；

该事件唯一可能的处理程序是与其挂接的委托

**隧道：从元素树的根部调用事件处理程序并依次向下深入直到事件源**

隧道的执行顺序与冒泡正好相反，直接看例子，然后再解释，将上便中所有的Click事件改为PreviewMouseDown事件，而绑定的事件处理程序不变

XAML代码如下：

<Window x:Class="WPF\_CODE.MainWindow"

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

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

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

<!--最外层的Grid-->

<Grid PreviewMouseDown="Btn\_Click" Name="Grid\_1" Background="#FF43AEAE">

<!--定义两行-->

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<!--第二层Grid-->

<Grid PreviewMouseDown="Btn\_Click" Name="Grid\_2" Margin="10" Background="#FF8D888D" Grid.Row="0">

<!--定义两列-->

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<!--第三层左侧Grid-->

<Grid PreviewMouseDown="Btn\_Click" Name="Grid\_3\_Left" Grid.Column="0" Background="#FF3D8F3A" Margin="10">

<!--添加一个按钮-->

<Button PreviewMouseDown="Btn\_Click" Name="ButtonLeft" Width="80" Height="50" Content="Hello"/>

</Grid>

<!--第三层右侧Grid-->

<Grid PreviewMouseDown="Btn\_Click" Name="Grid\_3\_Right" Grid.Column="1" Background="#FFC95E3E" Margin="10">

<!--添加一个按钮-->

<Button PreviewMouseDown="Btn\_Click" Name="ButtonRight" Width="80" Height="50" Margin="10" Content="World"></Button>

</Grid>

</Grid>

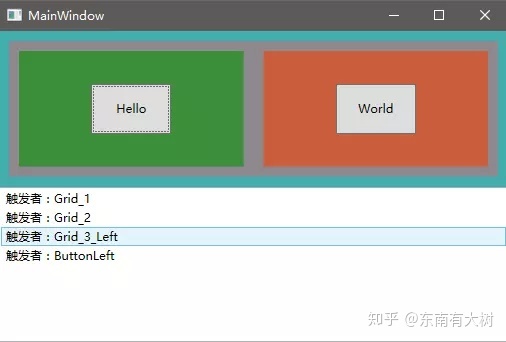
<!--定义一个ListBox，用于输出结果-->

<ListBox Name="Print\_List" Grid.Row="1"/>

</Grid>

</Window>

运行程序，查看效果：



可以看出，隧道是指事件首先是从根元素上被触发，然后从每一个元素向下沿着树传递，直到到达根元素为止

（或者直到到达处理程序把事件标记为已处理为止），他的执行方式正好与冒泡策略相反

所有的隧道事件都以Preview开头

后台代码记得将e.Handled=true注释掉

**RoutedEventArgs属性：**

**\* 1.Source：指定哪个对象引发了事件，对于键盘事件来说，是指具有输入焦点的对象，而对于鼠标事件来说，**

**\* 是指鼠标下面最顶层的，Enable为True的元素。由于路由事件的传递特性，Source的值将与事件实参类**

**\* 的sender形参的值有可能不同。**

**\* 2.OriginalSource:指定哪个对象最初引发了事件，通常与Source属性的值相同，然而在一些情况下OriginalSource**

**\* 可能比Source属性引用的元素更精细。例如，如果单击窗口边缘，Source属性时Window对象，而OriginalSource则是Border**

**\* 对象，最初是Border对象触发了单击事件。通常较少使用OriginalSource属性来获取事件源。**

**\* 3.RouteEvent:可以使用该属性获取当前触发的事件的类型，比如UIElement.MouseUpEvent对象，在使用相同的事件**

**\* 处理器处理不同的事件时非常有用。**

**\* 4.Handled:允许停止冒泡或隧道事件处理。如果将Handled设置为TRUE，事件将不再继续传递，也不会被触发**