Bài 6
XỬ LÝ SỰ KIỆN VÀ LỆNH TRONG WPF

Các bài giảng trước chủ yếu giới thiệu về các thành phần trực quan trong WPF và việc làm thế nào để tạo lập giao diện đồ hoạ kết hợp những thành phần đó. Tuy nhiên, một giao diện đồ họa không chỉ mang tính thẩm mỹ cao mà còn phải cho phép người dùng tương tác với các thành phần trên đó. Việc tương tác với ứng dụng của người dùng thông qua giao diện đồ hoạ có liên quan nhiều trên việc viết mã lệnh xử lý *sự kiện* (events) và *lệnh* (commands). Mặc dù các khái niệm này đã được đề cập sơ bộ trong các bài giảng trước, bài giảng này giới thiệu một cách có hệ thống hơn về hai khái niệm quan trọng này trong WPF.

1. Xử lý sự kiện trong WPF

1.1. Sự kiện

Mỗi khi bạn nhắp chuột vào một nút bấm hay gõ dòng văn bản nào đó vào một form, bạn đang sử dụng *sự kiện* (events). Trong lập trình, có thể định nghĩa *sự kiện* là một hành động được phát động bởi người dùng, bởi một thiết bị như đồng hồ đếm (timer) hay bàn phím, hoặc thậm chí là bởi hệ điều hành, tại những thời điểm phần lớn là không theo chu trình nhất định. Ví dụ, với một thiết bị định vị con trỏ như chuột, hành động nhắp phím chuột sẽ gây nên sự kiện "nhắp chuột". Mỗi khi một sự kiện xảy ra, thông thường dữ liệu liên quan đến sự kiện đó được thu thập và chuyển nó tới một *đơn vị xử lý sự kiện* (event handler) để xử lý tiếp. Cũng có khi, sự kiện bị bỏ qua hay chuyển tới nhiều hàm xử lý sự kiện một lúc nếu những hàm xử lý này cùng đồng thời lắng nghe sự kiện đó. Dữ liệu tương ứng với một sự kiện ít nhất xác định loại sự kiện, nhưng đôi khi cũng bao gồm các thông tin khác như sự kiện xảy ra tại thời điểm nào, đối tượng nào phát động nó...

Thông thường, ta hầu như không suy nghĩ về việc sự kiện xảy ra như thế nào, ví dụ làm sao để máy tính nhận biết chuột trái được nhắp, hay một phím trên bàn phím được bấm… Lý do là vì các chi tiết ở mức thấp này đã được framework đồ hoạ trong máy tính xử lý. Ngay cả đối với người phát triển, công việc của ta với sự kiện phần lớn là xử lý phần bề nổi của nhiều vấn đề ở phía sau

mỗi sự kiện. Ngay cả trong trường hợp đó, có rất nhiều phần "bề nổi" cần được xem xét. Trong phần này, trước hết ta tìm hiểu cơ chế xử lý sự kiện trong WPF.

1.2. Đơn vị xử lý sự kiện

Mỗi *đơn vị xử lý sự kiện* (event handler) đơn giản là một phương thức (hàm) nhận đầu vào từ một thiết bị như chuột hay bàn phím và thực hiện một việc nào đó để phản ứng lại với một sự kiện xảy ra trên thiết bị đó. Ví dụ sau đây minh hoạ đoạn mã lệnh C# là một đơn vị xử lý sự kiện có tên `ButtonOkClicked` có tác dụng xử lý sự kiện nút chuột được bấm:

```
private void ButtonOkClicked(object sender, RoutedEventArgs e)
{
     this.Close(); //đóng cửa sổ hiện thời

}
```

Trong các phần tiếp theo, để dễ hiểu, ta dùng từ "*hàm xử lý sự kiện*" với nghĩa tương đương "*đơn vị xử lý sự kiện*"

Thực chất, có hai bước cần thực hiện để xử lý một sự kiện:

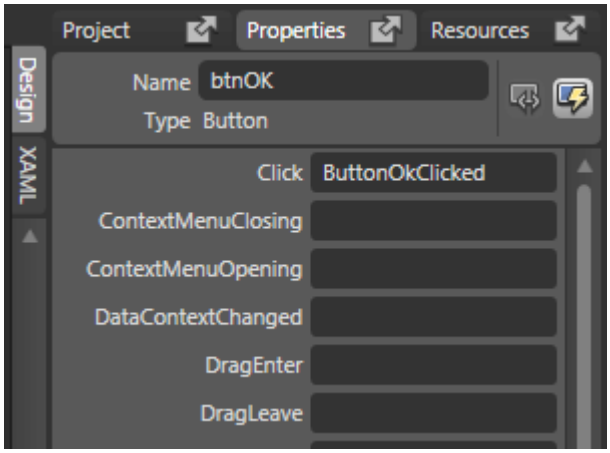
1. Liên kết đơn vị xử lý sự kiện với điều khiển (nút bấm, trường văn bản, thực đơn…), nơi sự kiện tương ứng được phát động.
2. Viết mã lệnh trong đơn vị xử lý sự kiện để lập trình các công việc phản ứng lại với sự kiện.

Có hai cách để liên kết một sự kiện với một đơn vị xử lý sự kiện. Bạn có thể dùng (1) một *môi trường phát triển tích hợp* (IDE) như Expression Blend hoặc WPF Designer của Visual Studio (cách trực quan); hoặc (2) viết mã lệnh trực tiếp.

1.2.1. Cách liên kết trực quan

Để liên kết theo cách này, ta cần có các công cụ thiết kế giao diện GUI dành cho WPF chẳng hạn như Expression Blend hoặc WPF Designer của Visual Studio. Với các công cụ này, với mỗi phần từ UI trên giao diện ta có cửa sổ liệt kế các sự kiện. Với mỗi sự kiện, ta có thể phân định đơn

vị xử lý sự kiện bằng cách khai báo tên hàm xử lý (không gồm đối số) bên cạnh sự kiện ta muốn bắt và xử lý. Hình 6.1 minh hoạ việc khai báo hàm xử lý sự kiện ButtonOkClicked ứng với sự kiện Click của nút bấm btnOK sử dụng Expression Blend.



Hình 6.1 - Phân định trực quan hàm ButtonOkClicked xử lý sự kiện Click của nút btnOK trên Expression Blend

Sau khi khai báo, ta nhấn Enter, môi trường sẽ tự động tạo sinh và chuyển ta đến khuôn rỗng của hàm xử lý sự kiện có tên giống với tên ta đã đặt cho đơn vị xử lý sự kiện khi khai báo, và với danh sách tham số ngầm định tương ứng với loại sự kiện. Nhiệm vụ của người lập trình lúc này là viết mã lệnh thực hiện các hành động phản ứng với sự kiện bên trong hàm xử lý này. Trong ví dụ về nút bấm trên, khuôn dạng tự sinh của hàm xử lý sẽ là:

```
private void ButtonOkClicked(object sender, RoutedEventArgs e)
{
    //viết mã xử lý vào đây

}
```

Khi nhìn lại mã XAML tương ứng, ta sẽ thấy WPF sử dụng XAML để khai báo liên kết giữa sự kiện mà hàm xử lý sự kiện như thế nào:

```
<Button HorizontalAlignment="Left" Margin="130,92,0,86" x:Name="btnOK"
Width="80" Content="OK" Click="ButtonOkClicked"/>
```

Như đã thấy, để gắn kết sự kiện Click với hàm xử lý ButtonOkClicked, ta có thể khai báo `Click="ButtonOkClicked"` trong khai báo tạo lập nút bấm trong mã XAML.

1.2.2. Cách liên kết bằng mã lệnh trực tiếp

Ta cũng có thể liên kết sự kiện vào hàm xử lý bằng mã lệnh với kết quả không đổi. Bạn có thể tự hỏi tại sao không chọn cách trực quan ở trên. Một lý do cơ bản là nếu ta muốn tạo ra các điều khiển một cách linh động, ví dụ sinh ra một hay nhiều nút bấm trong thời gian chạy (runtime) chứ không phải tạo lập sẵn trong thời gian thiết kế form (design-time), thì cách duy nhất để liên kết sự kiện của các điều khiển đó vào hàm xử lý là thông qua mã lệnh. Xét ví dụ sau đây:

Giả sử ta có một nút bấm có tên là btnOK, và mục tiêu của ta là gắn kết một sự kiện của nó với hàm xử lý mà chỉ dùng mã lệnh. Tất cả những việc phải làm là chọn tên sự kiện tương ứng mà ta muốn bắt và liên kết nó với dòng lệnh new RoutedEventHandler với đối số là tên của hàm xử lý của ta. Ví dụ:

```
btnOK.Click += new RoutedEventHandler(ButtonOkClicked);
```

Tiếp theo ta khai báo hàm xử lý với đối số tương ứng với sự kiện. Thông thường mỗi loại sự kiện của mỗi loại điều khiển lại đòi hỏi hàm xử lý sự kiện tương ứng với nó có chứa danh sách tham số xác định (có số lượng, thứ tự và kiểu tham số xác định trước), mặc dù tên gọi của hàm xử lý có thể tuỳ ý. Nếu ta sử dụng cách trực quan, cấu trúc của hàm xử lý sự kiện sẽ được tự động tạo ra. Việc của ta chỉ là viết nội dung xử lý bên trong hàm xử lý. Trong trường hợp viết mã lệnh, ta phải tự viết phần khai báo hàm xử lý, trong đó, cần tuân theo quy tắc định nghĩa về cấu trúc tham số (số lượng, thứ tự, kiểu tham số) tương ứng của sự kiện đó. Để biết được cấu trúc này, không gì khác ngoài việc tìm đọc các tài liệu tham khảo về sự kiện tương ứng, mà MSDN là tài liệu đầy đủ và chính xác nhất.

Trong ví dụ trên, phần nội dung hàm xử lý sự kiện Click trong mã C# sẽ là:

```
private void ButtonOkClicked(object sender, RoutedEventArgs e)
{
    this.Close();
```

```
        }
```

Để ý rằng hàm xử lý sự kiện trong ví dụ chứa 2 tham số mà giá trị của chúng sẽ được lấy từ sự kiện – *sender* tham chiếu đến đối tượng phát động sự kiện (ở đây là nút bấm btnOK) và *event* (*e*) chỉ ra dạng tác động cụ thể để sự kiện bị kích hoạt, chẳng hạn như bấm phím hay nhắp chuột... Trong nhiều trường hợp, bạn không cần phải quan tâm đến các tham số của hàm xử lý sự kiện. Ví dụ, trong đoạn mã ví dụ ở trên, phần nội dung xử lý sự kiện không hề dùng tới tham số sender lẫn tham số e. Tuy nhiên, sẽ có những trường hợp trong đó, bạn muốn sử dụng cùng một hàm xử lý ứng với nhiều sự kiện có cùng bản chất hoặc cho một loại sự kiện của nhiều đối tượng cùng loại. Khi đó, ta phải quan tâm đến điều khiển nào đã gửi sự kiện, lúc đó tham số sender và event có thể sẽ hữu dụng.

1.3 Sự kiện có định tuyến

WPF mở rộng mô hình lập trình hướng sự kiện chuẩn của .NET, bằng việc đưa ra một loại sự kiện mới gọi là *sự kiện có định tuyến* (routed event). Loại sự kiện này nâng cao tính linh hoạt trong các tình huống lập trình hướng sự kiện. Việc thiết lập và xử lý một sự kiện có định tuyến có thể thực hiện với cùng cú pháp với một sự kiện "thường" (CLR event).

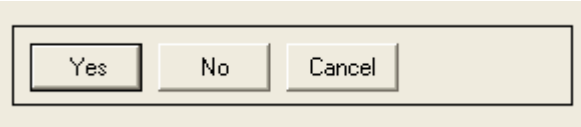
1.3.1 Cây trực quan

Trước khi bàn luận thêm về sự kiện có định tuyến, một khái niệm quan trọng cần biết đó là *cây trực quan* (visual tree). Một giao diện người dùng WPF được xây dựng theo phương thức phân lớp, trong đó một phần tử trực quan không có hoặc có các phần tử con. Cấu trúc phân cấp của các lớp phần tử trực quan như thế trên một giao diện người dùng được gọi là *cây trực quan* của giao diện đó. Ví dụ, xét giao diện được định nghĩa bằng đoạn mã XAML sau:

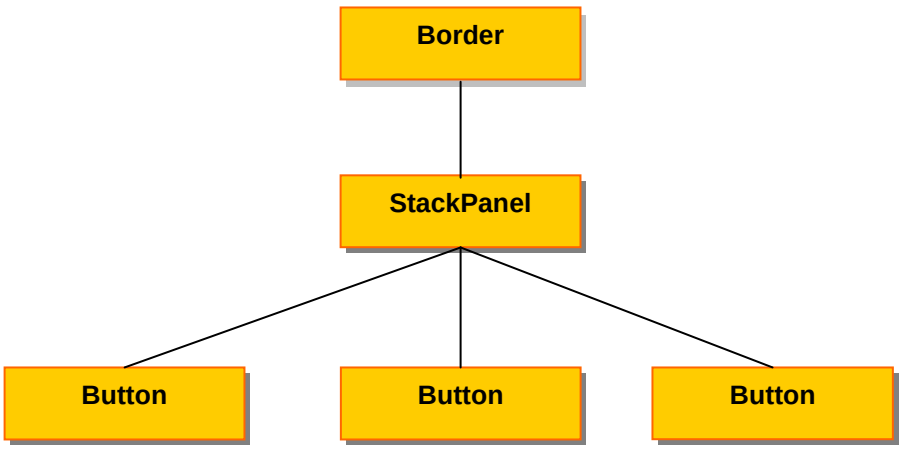
```
      <Border Height="50" Width="300" BorderBrush="Gray" BorderThickness="1">
          <StackPanel Background="LightGray" Orientation="Horizontal"
Button.Click="CommonClickHandler">
            <Button Name="YesButton" Width="Auto" >Yes</Button>
            <Button Name="NoButton" Width="Auto" >No</Button>
            <Button Name="CancelButton" Width="Auto" >Cancel</Button>
          </StackPanel>
```

```
    </Border>
```

Kết quả khi chạy chương trình:



Cây trực quan tương ứng sẽ là:



Hình 6.2 – Ví dụ về cây trực quan

1.3.2 Sự kiện có định tuyến là gì?

Về mặt chức năng, sự kiện có định tuyến là một loại sự kiện có thể kích hoạt nhiều đơn vị xử lý sự kiện thuộc về nhiều điều khiển khác nhau trên cây trực quan, chứ không chỉ trên đối tượng đã phát động sự kiện.

Một ứng dụng WPF điển hình thường chứa nhiều phần tử UI. Bất kể được tạo ra bằng mã lệnh hay được khai báo bằng XAML, các thành phần này tồn tại trong mối quan hệ kiểu cây trực quan với nhau - tạo nên các tuyến quan hệ đi từ thành phần này tới thành phần kia. Theo các tuyến quan hệ đó, có ba phương thức định tuyến sự kiện: lan truyền lên (bubble), lan truyền xuống (tunnel) và trực tiếp (direct).

Lan truyền lên (bubble) là phương thức thường thấy nhất. Nó có nghĩa là một sự kiện sẽ được truyền đi trên cây trực quan từ thành phần nguồn (nơi sự kiện được phát động) cho tới khi nó được xử lý hoặc nó chạm tới nút gốc. Điều này cho phép ta xử lý một sự kiện trên một đối tượng nằm ở cấp trên so với thành phần nguồn. Ví dụ, bạn có thể gắn một hàm xử lý sự kiện Button.Click vào đối tượng Grid có chứa nút bấm thay vì gắn hàm xử lý đó vào bản thân nút bấm. Sự kiện lan truyền lên có tên gọi thể hiện hành động của sự kiện, ví dụ: MouseDown.

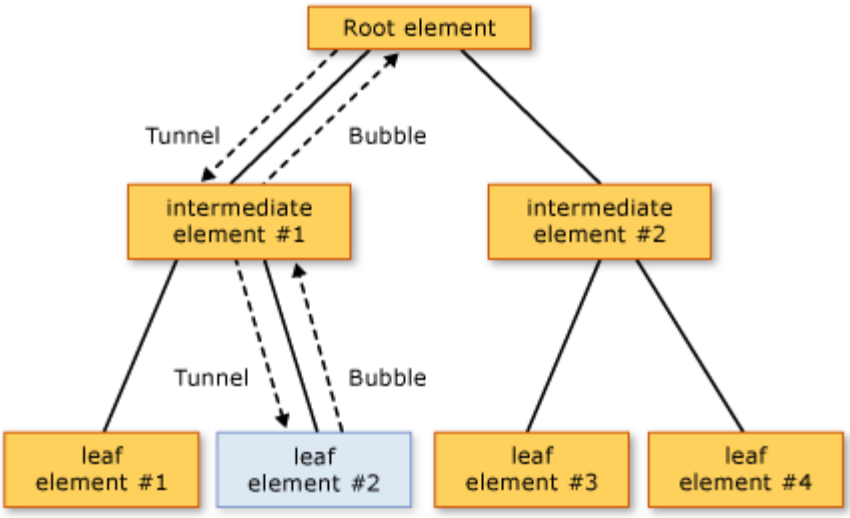
Sự kiện lan truyền xuống (tunnel) đi theo hướng ngược lại, bắt đầu từ nút gốc và truyền xuống cây trực quan cho tới khi nó được xử lý hoặc chạm tới thành phần gốc của sự kiện đó. Điều này cho phép các thành phần cấp trên có thể chặn sự kiện và xử lý nó trước khi sự kiện đó chạm tới thành phần nguồn (nơi dự định xảy ra sự kiện). Các sự kiện lan truyền xuống có tên được gắn thêm tiền tố Preview, ví dụ, sự kiện PreviewMouseDown.

Sự kiện trực tiếp (direct) hoạt động giống như sự kiện thông thường trong .NET Framework. Chỉ có một đơn vị xử lý duy nhất sẽ được gắn với sự kiện trực tiếp.

Thông thường, nếu một sự kiện lan truyền xuống được định nghĩa cho một sự kiện nào đó, đồng thời cũng sẽ có một sự kiện lan truyền lên tương ứng. Trong trường hợp đó, sự kiện lan truyền xuống sẽ được phát động trước, bắt đầu từ gốc và chạy xuống tìm kiếm hàm xử lý trên cây trực quan. Một khi nó đã được xử lý hoặc chạm tới thành phần nguồn, sự kiện lan truyền lên sẽ được phát động, lan truyền từ thành phần nguồn đi ngược lên để tìm tới hàm xử lý nó trên cây trực quan. Sự kiện lan truyền lên hay xuống sẽ không ngừng lan truyền vì một hàm xử lý nó được gọi. Do vậy, nếu ta muốn dừng quá trình truyền xuống hoặc lên, ta phải đánh dấu "đã xử lý" cho tham số sự kiện truyền vào, cụ thể:

```
      private void OnChildElementMouseDown(object sender, MouseButtonEventArgs
e) {
            e.Handled = true;
      }
```

Một khi ta đã đánh dấu "đã xử lý" cho sự kiện (`e.Handled = true`), nó sẽ không được lan truyền tiếp nữa.



Hình 6.3 - Sự kiện có định tuyến trên cây trực quan [xxx]

Trở lại ví dụ trên, nguồn của sự kiện Click là một trong những thành phần nút bấm, và bất kể nút nào được bấm, nó sẽ trở thành thành phần đầu tiên được phép xử lý sự kiện. Tuy nhiên, nếu không có đơn vị xử lý nào tương ứng với sự kiện Click gắn với nút đó, thì sự kiện sẽ được lan truyền lên trên phần tử cha của nút bấm, trong trường hợp này là StackPanel, rồi sau đó, lan truyền tới Border... Nói cách khác, tuyến lan truyền sự kiện Click sẽ là:

Button→StackPanel→Border→...

1.3.3 Các tình huống cơ bản sử dụng sự kiện có định tuyến

Phần sau đây tổng kết những tình huống cần vận dụng khái niệm sự kiện có định tuyến, và tại sao một sự kiện CLR điển hình là không đủ trong những tình huống đó.

a. *Bao đóng và kết hợp điều khiển*

Nhiều điều khiển trong WPF có cấu trúc nội dung phức hợp. Ví dụ, ta có thể đặt một hình ảnh bên trong một nút bấm, làm mở rộng cây trực quan của nút bấm. Tuy nhiên, hình ảnh thêm vào không được phép phá vỡ cơ chế hit-testing, cơ chế khiến nút bấm phản ứng với việc nhấp chuột vào trong nó, ngay cả khi người dùng nhấp chuột vào những pixel là một phần của hình ảnh thêm vào.

b. Các điều khiển sử dụng cùng một đơn vị xử lý sự kiện

Trong Windows Forms, có trường hợp ta cần gán nhiều lần cùng một đơn vị xử lý để xử lý các sự kiện thuộc vào nhiều thành phần khác nhau. Sự kiện có định tuyến cho phép ta gán đơn vị xử lý chỉ một lần trong trường hợp đó. Như trong ví dụ đã nêu trong đoạn mã XAML, sau đây là hàm xử lý tương ứng:

```
private void CommonClickHandler(object sender, RoutedEventArgs e)
{
  FrameworkElement feSource = e.Source as FrameworkElement;
  switch (feSource.Name)
  {
    case "YesButton":
      // do something here ...
      break;
    case "NoButton":
      // do something ...
      break;
    case "CancelButton":
      // do something ...
      break;
  }
  e.Handled=true;
}
```

c. Xử lý lớp:

Sự kiện có định tuyến cho phép một đơn vị xử lý tĩnh (static) được định nghĩa trong lớp. Đơn vị xử lý lớp này có cơ hội xử lý một sự kiện trước khi một đơn vị xử lý gắn với đối tượng cụ thể nào đó của lớp có thể.

d. Tham chiếu đến một sự kiện mà không bị hiện tượng phản xạ:

Các kỹ thuật markup và mã lệnh đòi hỏi phải có cách để định danh một sự kiện. Một sự kiện có định tuyến tạo ra trường RoutedEvent như một định danh, cung cấp một kỹ thuật định danh sự kiện mạnh mà không đòi hỏi hiện tượng phản xạ tĩnh hoặc run-time.

1.3.4 Lợi ích của sự kiện có định tuyến

Cơ chế thông báo sự kiện kiểu định tuyến có nhiều lợi ích. Một lợi ích rất quan trọng của sự kiện có định tuyến là một thành phần UI trực quan không cần móc nối cùng một sự kiện trên tất cả các thành phần con trong nó, chẳng hạn sự kiện MouseMove. Thay vào đó, nó có thể móc nối sự kiện này vào bản thân nó, và khi con chuột di chuyển qua một trong các thành phần con của nó, sự kiện này sẽ được lan truyền tới nó.

Một ưu điểm quan trọng khác của sự kiện có định tuyến là các thành phần ở tất cả các mức trong cây trực quan có thể tự động thực thi mã lệnh để phản ứng lại các sự kiện của các thành phần con của chúng, mà không cần các thành phần con phải thông báo khi sự kiện xảy ra.

1.3.5 Một ví dụ đầy đủ về sự kiện có định tuyến

Form chỉ bao gồm một StackPanel chứa 2 Button và 1 TextBlock có tên xác định. StackPanel được phân định bắt sự kiện Click trên hai nút bấm nằm trong nó. Nhiệm vụ của đơn vị xử lý sự kiện Click là cho biết đối tượng nào đã xử lý sự kiện Click, sự kiện Click phát ra từ loại đối tượng nào, tên gọi là gì nào, và loại lan truyền định tuyến đã được thực hiện. Các thông tin trên được đưa vào nội dung của TextBlock và hiển thị lên màn hình sau mỗi sự kiện Click.

Đoạn mã XAML khai báo giao diện như sau:

```
      <Window x:Class="Lesson6.Window1"
          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
          Title="Lesson6 - Routed Events" Height="300" Width="300"
          >
        <!--Khai báo stack panel làm layout chính.
            Trong đó, có bắt sự kiện Click của Button và xử lý qua hàm
HandleClick-->
        <StackPanel
          Name="My_StackPanel"
          Button.Click="HandleClick"
        >
          <!--Khai báo tạo lập Button 1-->
```

```
        <Button Name="Button1">Nút bấm 1</Button>
        <!--Khai báo tạo lập Button 2-->
        <Button Name="Button2">Nút bấm 2</Button>
        <!--Khai báo tạo lập TextBlock lưu trữ và hiển thị kết quả-->
        <TextBlock Name="Results"/>
      </StackPanel>

    </Window>
```

Đoạn mã lệnh C# cho hàm HandleClick để xử lý sự kiện Click:

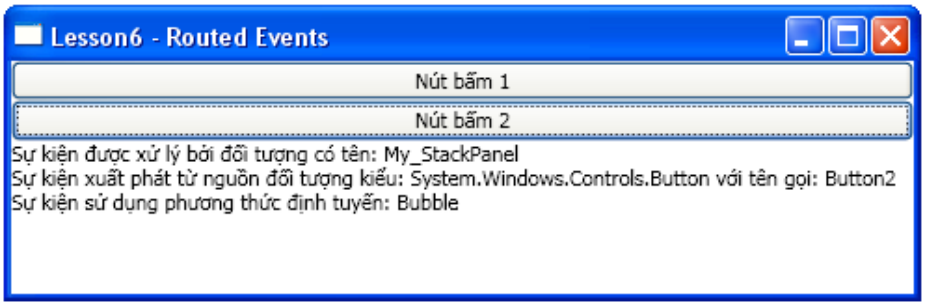
```
      //Dùng một StringBuilder để lưu trữ thông tin kết quả
            StringBuilder eventstr = new StringBuilder();

            //Đơn vị xử lý sự kiện Click của Button
            void HandleClick(object sender, RoutedEventArgs args)
            {
                //Lấy thông tin về đối tượng xử lý sự kiện Click
                FrameworkElement fe = (FrameworkElement)sender;
                eventstr.Append("Sự kiện được xử lý bởi đối tượng có tên: ");
                eventstr.Append(fe.Name);
                eventstr.Append("\n");
                //
                //Lấy thông tin về nguồn phát ra sự kiện CLick:
                FrameworkElement fe2 = (FrameworkElement)args.Source;
                eventstr.Append("Sự kiện xuất phát từ nguồn đối tượng kiểu:
");
                //+ Loại thành phần UI;
                eventstr.Append(args.Source.GetType().ToString());
                //+ Định danh;
                eventstr.Append(" với tên gọi: ");
                eventstr.Append(fe2.Name);
                eventstr.Append("\n");
                //
                //Lấy thông tin về phương thức định tuyến
                eventstr.Append("Sự kiện sử dụng phương thức định tuyến: ");
                eventstr.Append(args.RoutedEvent.RoutingStrategy);
                eventstr.Append("\n");
```

```
            //
            //Đưa thông tin ra màn hình
            Results.Text = eventstr.ToString();

        }
```

Kết quả như sau:



Hình 6.4 – Ví dụ về sự kiện có định tuyến

2. Lệnh (Command) trong WPF

Ra lệnh (commanding) là một cơ chế nhập tin trong WPF cung cấp khả năng xử lý đầu vào ở mức ngữ nghĩa hơn là xử lý đầu vào từ thiết bị. Các ví dụ về command là các hành động Copy, Cut và Paste mà ta đã gặp ở nhiều ứng dụng. Phần tiếp theo sẽ trình bày tổng quan về khái niệm này trong WPF.

2.1 Lệnh là gì?

Điểm khác biệt giữa lệnh và một đơn vị xử lý sự kiện đơn giản gắn với một nút hay một đồng hồ đếm là: lệnh tách bạch giữa *ngữ nghĩa* cũng như *nguồn phát hành động* với *logic thực hiện hành động* đó. Điều này cho phép nhiều nguồn khác biệt nhau hoàn toàn có thể phát động cùng một logic lệnh, đồng thời, cho phép tuỳ biến logic lệnh tuỳ vào các đối tượng bị tác động khác nhau.

Ví dụ điển hình về lệnh là các hành động Copy, Cut và Paste, được thấy ở rất nhiều ứng dụng. Ngữ nghĩa của các lệnh này là nhất quán với tất cả các ứng dụng và lớp khác nhau (Copy - tạo bản sao từ đối tượng được chọn; Cut - tạo bản sao rồi xoá bỏ đối tượng được chọn (cắt); Paste – Chèn đối tượng được copy/cắt vào vị trị được chọn). Tuy nhiên, logic hành động lại tuỳ thuộc vào đối tượng cụ thể mà ta tác động lên. Ví dụ, tổ hợp phím CTRL+X có thể phát động lệnh Cut trên các lớp văn bản, các lớp hình ảnh và trên trình duyệt Web, nhưng logic thực sự thực hiện hành động Cut lại được định nghĩa bởi đối tượng hoặc ứng

dụng mà lệnh cắt tác động lên chứ không phải từ nguồn đã phát ra lệnh. Cụ thể hơn, một đối tượng văn bản có thể cắt đoạn văn bản được chọn vào clipboard, trong khi một đối tượng hình ảnh có thể cắt lấy vùng ảnh được chọn, nhưng nguồn phát lệnh là như nhau - một tổ hợp phím hay một nút bấm trên thanh công cụ.

Một cách đơn giản để sử dụng lệnh trong WPF là sử dụng một RoutedCommand đã được định sẵn trong các lớp thư viện lệnh; sử dụng một điều khiển có hỗ trợ sẵn xử lý lệnh đó và một điều khiển hỗ trợ sẵn khả năng phát động lệnh. Trong ví dụ dưới đây, lệnh Paste là một trong những lệnh định sẵn trong lớp ApplicationCommands. Điều khiển TextBox đã xây dựng sẵn khả năng xử lý lệnh Paste. Và lớp MenuItem hỗ trợ khả năng phát động lệnh.

Ví dụ sau đây minh hoạ cách thức tạo lập một MenuItem để khi nhắp chuột vào nó, lệnh Paste sẽ được phát động trên một TextBox, với giả thiết là hộp TextBox đang nhận được focus.

Mã XAML:

```
<StackPanel>
    <Menu>
      <!--Khai báo hàm xử lý lệnh Paste-->
      <MenuItem Command="ApplicationCommands.Paste" />
    </Menu>
    <TextBox />
  </StackPanel>
```

2.2 Những khái niệm chính trong hệ thống lệnh của WPF

Mô hình lệnh trong WPF có thể được chia thành bốn khái niệm chính: lệnh, nguồn lệnh, đích lệnh, và liên kết lệnh, trong đó:

- *Lệnh* là hành động được thực hiện
- *Nguồn lệnh* là đối tượng phát động lệnh
- *Đích lệnh* là đối tượng mà lệnh tác động lên
- *Liên kết lệnh* là đối tượng ánh xạ logic thực hiện lệnh với lệnh

Trong ví dụ trên đây, Paste là lệnh, MenuItem là nguồn lệnh, TextBox là đích lệnh, và liên kết lệnh được cung cấp bởi điều khiển TextBox (định sẵn). Cần lưu ý rằng không phải lúc nào liên kết lệnh (CommandBinding) cũng được cung cấp bởi điều khiển đóng vai trò đích lệnh. Thông thường,

CommandBinding phải được tạo bởi người lập trình ứng dụng, hoặc CommandBinding có thể được gắn với đối tượng cha của đích lệnh.

2.3 Lệnh có định tuyến

Sự khác biệt giữa lệnh có định tuyến và sự kiện có định tuyến là cách mà lệnh được dẫn đường từ nơi phát động lệnh (nguồn lệnh) tới nơi xử lý lệnh (đích lệnh). Trong mô hình lệnh có định tuyến, sự kiện có định tuyến được sử dụng dưới dạng các thông báo giữa giữa nguồn lệnh vào đích lệnh (thông qua liên kết lệnh).

Trong một thời điểm nhất định, chỉ có một đơn vị xử lý lệnh (gắn với đích lệnh) sẽ được thực sự kích hoạt (Đơn vị xử lý lệnh hoạt động). Đơn vị xử lý lệnh hoạt động được xác định bằng việc kết hợp giữa vị trí của nguồn lệnh và đích lệnh trên cây, và đâu là thành phần UI đang nhận được focus. Khi lệnh được phát đi, sự kiện có định tuyến sẽ được sử dụng để gọi đến đơn vị xử lý lệnh hoạt động, để hỏi xem lệnh này có được cho phép không (thông qua phương thức CanExecute), cũng như thực hiện logic hành động (thông qua phát động phương thức Executed).

Thông thường, nơi phát lệnh sẽ tìm liên kết lệnh giữa vị trí của nó trên cây trực quan và nút gốc của cây trực quan. Nếu nó tìm thấy một liên kết lệnh như thế, đơn vị xử lý lệnh tương ứng sẽ xác định lệnh này có được cho phép thực hiện không. Nếu như lệnh được gắn với một điều khiển trên thanh công cụ hay menu, thì một vài bước logic thêm sẽ được thực hiện để tìm dọc theo đường đi trên cây trực quan từ nút gốc tơi phần tử đang nhận được focus để tìm kiếm một liên kết lệnh.

Một điểm quan trọng cần hiểu về việc định tuyến trong lệnh có định tuyến của WPF là một khi một đơn vị xử lý lệnh đã được kích hoạt, sẽ không có đơn vị xử lý nào khác được gọi.

Để nắm rõ hơn về ưu điểm của việc sử dụng lệnh trong WPF ta xét ví dụ sau:

2.4 Một ví dụ về sử dụng lệnh trong WPF

Ta xét một form gồm một ListBox (có tên `lsbCustomers`) chứa danh sách tên các khách hàng và một menu có chứa mục xoá Delete, có tác dụng xoá mục được chọn trong danh sách. Ta

muốn chắc chắn rằng người sử dụng phải chọn tên khách hàng trong danh sách trước khi có thể bấm mục xoá Delete trên menu.

Đoạn mã sau sẽ vô hiệu hoá mục Delete trên cơ sở có một mục được chọn trong danh sách khách hàng hay không.

```
private void HandleMenus()
{
     menuDelete.IsEnabled = lsbCustomers.SelectedItem != null;
}
```

Đây là cách thông thường để đồng bộ hoá việc cho phép hay vô hiệu một mục menu hay nút bấm ứng với một điều kiện nào đó. Để đạt mục tiêu đã nêu của đầu bài, đoạn mã trên có thể được gọi trong đơn vị xử lý sự kiện SelectionChanged của ListBox như sau:

```
private void lsbCustomers_SelectionChanged(object sender,
       SelectionChangedEventArgs e)
{
          HandleMenus();
}
```

Cách xử lý dựa trên sự kiện kiểu này là bình thường khi trên form chỉ có một ListBox. Tuy nhiên, khi form trở nên phức tạp hơn, ví dụ chứa 2 ListBox, khi đó việc xử lý theo cách trên trở nên phức tạp. Trở lại ví dụ, giả sử ta có thêm một ListBox có chứa danh sách các mặt hàng (có tên `lsbProducts`). Cả hai ListBox chứa tên khách hàng và tên mặt hàng đều chịu tác động của mục Delete khi chúng nhận được focus và một trong các tên được chọn. Trong trường hợp này, để xét xem mục Delete nên bị vô hiệu hoá hay không, điều kiện kiểm tra trở nên phức tạp hơn:

```
private void HandleMenus()
{
      menuDelete.IsEnabled =
         (lsbCustomers.SelectedItem != null &&
         ((ListBoxItem)lsbCustomers.SelectedItem).IsFocused) ||
         (lsbProducts.SelectedItem != null &&
         ((ListBoxItem)lsbProducts.SelectedItem).IsFocused);
}
```

Đồng thời, cũng yêu cầu thêm mã lệnh đối với việc xử lý sự kiện Click lên mục Delete trên menu: Ta phải xác định ListBox nào bị tác động:

```
private void menuDelete_Click(object sender, RoutedEventArgs e)
{
      if (lsbCustomers.SelectedItem != null &&
         ((ListBoxItem)lsbCustomers.SelectedItem).IsFocused)
            lsbCustomers.Items.Remove(lsbCustomers.SelectedItem);
      else if (lsbProducts.SelectedItem != null &&
         ((ListBoxItem)lsbProducts.SelectedItem).IsFocused)
            lsbProducts.Items.Remove(lsbProducts.SelectedItem);

}
```

Hãy tưởng tượng nếu như form chứa khoảng 5 điều khiển cùng chịu tác động của hành động Delete, phần mã lệnh xử lý sẽ trở nên phức tạp đến mức nào. May mắn là WPF cung cấp một phương thức tốt hơn trong trường hợp như vậy. Cơ chế lệnh trong WPF đơn giản hoá mã lệnh trong trường hợp này bởi nó phân tách rõ giữa lệnh với phần triển khai lệnh (logic lệnh), cho phép ta liên kết điều khiển với những lệnh cụ thể, như ta sẽ thấy trong tiếp theo.

Đây là đoạn mã lệnh tương đương cho ví dụ trên sử dụng Command trong WPF.

```
<StackPanel>
  <Menu>
    <MenuItem Command="ApplicationCommands.Delete"
      Header="Delete" />
  </Menu>

  <Label>Khách hàng:</Label>

  <ListBox Name="lsbCustomers">
    <ListBox.CommandBindings>
      <CommandBinding
        Command="ApplicationCommands.Delete"
        CanExecute="DeleteCustomer_CanExecute"
        Executed="DeleteCustomer_Executed" />
    </ListBox.CommandBindings>
```

```
            <ListBoxItem>Bùi Như Lạc</ListBoxItem>
            <ListBoxItem>Ngô Giang Thơm</ListBoxItem>
            <ListBoxItem>Nguyễn Y Vân</ListBoxItem>
          </ListBox>
        </StackPanel>
```

Đoạn mã đầu phân định giá trị cho thuộc tính Command cho mục Delete trên menu. Nó cũng gắn một liên kết lệnh vào ListBox danh sách khách hàng. Trong trường hợp này, menu Delete là *nguồn lệnh*, và ListBox đóng vai trò là *đích lệnh*. CommandBinding xác định hàm thực hiện đối với hai thuộc tính CanExecute và Executed. CanExecute xác định khi nào lệnh Delete có thể được thực hiện, trong khi Executed xác định thực hiện logic lệnh trên đích lệnh như thế nào. Sau đây là mã lệnh cài đặt cho hai hàm này:

```
        private void DeleteCustomer_CanExecute(object sender,
CanExecuteRoutedEventArgs e)
        {
              e.CanExecute = lsbCustomers.SelectedItem != null;
        }

        private void DeleteCustomer_Executed(object sender,
ExecutedRoutedEventArgs e)
        {
              lsbCustomers.Items.Remove(lsbCustomers.SelectedItem);
        }
```

Cho tới đây, ta chưa thấy được ưu điểm của cách tiếp cận này. Tuy nhiên, trong trường hợp thêm vào một ListBox danh sách sản phẩm, lợi ích của phương pháp sẽ thể hiện rõ hơn. Sau đây là đoạn mã XAML khai báo tạo lập ListBox chứa danh sách sản phẩm:

```
        <Label>Sản phẩm sách:</Label>
        <ListBox x:Name="lsbProducts" >
              <ListBox.CommandBindings>
                <CommandBinding

                  Command="ApplicationCommands.Delete"
                  CanExecute="DeleteProduct_CanExecute"
                  Executed="DeleteProduct_Executed" />
```

```
            </ListBox.CommandBindings>
            <ListBoxItem>Nếu còn có ngày mai</ListBoxItem>
            <ListBoxItem>Chiếc lá rơi màu xanh</ListBoxItem>
            <ListBoxItem>Nhân gian chi ngộ</ListBoxItem>
      </ListBox>
```

Một lần nữa, ta chỉ cần cài đặt hai phương thức CanExecute và Executed như sau:

```
      private void DeleteProduct_CanExecute(object sender,
CanExecuteRoutedEventArgs e)
      {
            e.CanExecute = lsbProducts.SelectedItem != null;
      }

      private void DeleteProduct_Executed(object sender, ExecutedRoutedEventArgs
e)
      {
            lsbProducts.Items.Remove(lsbProducts.SelectedItem);
      }
```

Không giống như cách tiếp cận truyền thống, khi sử dụng phương thức lệnh trong WPF, ta không cần phải thay đổi mã của *nguồn lệnh* (menu Delete) khi thêm ListBox thứ hai. Cũng chú ý rằng bạn không cần phải cân nhắc điều khiển nào nhận được focus. Lớp CommandManeger (lớp phối hợp hoạn động của các lệnh trong WPF) sẽ tương tác với FocusManager để xác định điều khiển nào hiện đang nhận focus.

Qua ví dụ trên, ta cũng thấy một đặc điểm quan trọng của lệnh trong WPF đó là: *Lệnh không tự động thực thi logic hành động*. Lệnh trong WPF đơn thuần chỉ thông báo cho các phần tử UI biết rằng có một lệnh có ngữ nghĩa như thế đang được phát động - Bản thân phần tử UI phải triển khai/hiện thực hoá logic hành động phản ứng lại. Việc tách bạch giữa lệnh và logic thực hiện lệnh là một điểm mạnh. Như ta thấy qua ví dụ trên, lệnh Delete được phát động từ cùng nguồn (Delete menu), nhưng việc cài đặt được thực hiện riêng cho 2 đối tượng hoàn toàn khác nhau.

2.5 Lệnh tự tạo

Tự tạo các lệnh của riêng bạn trong nhiều trường hợp là cần thiết. Việc này cũng không quá phức tạp trong WPF. Để làm được điều này, lớp lệnh tự tạo phải hiện thực hoá giao diện ICommand. Tuy nhiên, ta có thể dùng lớp RoutedUICommand là lớp có sẵn trong framework đã hiện thực hoá tốt giao diện ICommand. Ví dụ, sau đây là cách tạo nên một lệnh cho phép chèn thêm một khách hàng.

Trong file code-behind C#, ta tạo một lớp mới có tên là MyCommands chứa một biến public kiểu RoutedUICommand. Lớp này được đặt trong cùng namespace với đối tượng Window chính. Đoạn mã ví dụ như sau:

C#

```
namespace Lesson6
{
    public static class MyCommands
    {
        static MyCommands()
        {
            InsertCustomer = new RoutedUICommand(
               "Insert Customer", "InsertCustomer",
               typeof(MyCommands));
        }

        public readonly static RoutedUICommand InsertCustomer;


    }
}
```

Trong file .xaml, ta thêm một mục menu trên form:

XAML

```
<MenuItem
  Command="local:MyCommands.InsertCustomer"
  Header="Insert Customer" />
```

Sau đó, ta thêm một CommandBinding cho lệnh mới này cho Window chính:

```
<Window.CommandBindings>
    <CommandBinding
      Command="local:MyCommands.InsertCustomer"
      CanExecute="InsertCustomer_CanExecute"
      Executed="InsertCustomer_Executed" />
 </Window.CommandBindings>
```

Lưu ý ở đây, ta phải thay đổi một chút phần khai báo Window chính trong file xaml, cụ thể là thêm dòng:

```
xmlns:local="clr-namespace:Lesson6"
```

Dòng này có nhiệm vụ chỉ ra đường dẫn logic đến lớp MyCommands trong namespace, trong ví dụ là Lesson6, dưới tên tham chiếu là `local`. Nhờ đó, việc gán thuộc tính Command cho mục menu hay Window.Binding mới thực hiện được (`Command="local:MyCommands.InsertCustomer"`):

```
<Window x:Class="Lesson6.Window3"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    xmlns:local="clr-namespace:Lesson6"
    Title="Lesson6" Height="300" Width="300"
    >
```

Cuối cùng, ta cài đặt thêm các phương thức trong CommandBinding:

```
private void InsertCustomer_CanExecute(object sender,
CanExecuteRoutedEventArgs e)
{
        e.CanExecute = true;
}

private void InsertCustomer_Executed(object sender,
            ExecutedRoutedEventArgs e)
{
```

```
            ListBoxItem item = new ListBoxItem();
            item.Content = "New Customer";
            lsbCustomers.Items.Add(item);
        }
```

Câu hỏi Ôn tập

1. Để xử lý một sự kiện cần thực hiện những công việc gì?

A. Viết mã lệnh thực hiện các hành động phản ứng với sự kiện

B. Kết nối sự kiện với hàm xử lý sự kiện

C. Cả hai công việc trên

Trả lời: C

2. Một sự kiện định tuyến có thể là:

A. Sự kiện truyền xuống

B. Sự kiện truyền lên

C. Sự kiện trực tiếp

D. Một trong ba phương án a, b, c, tuỳ thuộc vào chiến lược dẫn tuyến của sự kiện đó

E. Có thể đồng thời hai trong 3 phương án a, b, c

Trả lời: D

3. Một sự kiện định tuyến có thể lan truyền:

A. Từ phần tử nguồn tới phần tử bất kỳ trên cây trực quan

B. Lan truyền theo một trong hai hướng: từ phần từ nguồn đến nút gốc hoặc từ phần tử nguồn đến các nút con của nó

C. Chỉ lan truyền (ngược hay xuôi) qua các phần từ nằm trong đoạn từ nút gốc tới phần tử nguồn mà có quan hệ họ hàng với phần tử nguồn.

Trả lời: C

4. Với mô hình sự kiện có định tuyến, một sự kiện lan truyền xuống được:

A. Lan truyền từ phần tử nguồn lên phần tử gốc trong cây trực quan

B. Lan truyền từ nút gốc đến phần tử nguồn trong cây trực quan

C. Lan truyền từ phần tử nguồn xuống các nút con trong cây trực quan

Trả lời: C

5. Khi gắn kết một lệnh với một đối tượng chịu tác động của lệnh, việc thực hiện lệnh sẽ do:

A. Bản thân lệnh đó tự thực thi hành động tương ứng với ngữ nghĩa của nó, người lập trình không phải tác động thêm gì

B. Việc gắn kết chỉ có tác dụng thiết lập việc phát thông báo cho đối tượng chịu tác động lệnh biết nó được ra lệnh gì mỗi khi lệnh được gọi, còn người lập trình phải viết mã lệnh thực thi lệnh đó như thế nào

C. Nguồn phát lệnh xác định việc thực thi hành động

Trả lời: B

6. Ưu điểm của việc sử dụng lệnh có định tuyến so với xử lý sự kiện có định tuyến:

A. Nguồn lệnh (nơi phát động lệnh) không bó chặt với đích lệnh (nơi xử lý lệnh) – chúng không cần các tham chiếu trực tiếp lẫn nhau như trong trường hợp liên kết bằng đơn vị xử lý sự kiện

B. Lệnh có định tuyến sẽ tự động cho phép hoặc vô hiệu hoá tất cả các điều khiển UI tương ứng khi đích lệnh xác định rằng lệnh đó bị vô hiệu hoá

C. Lệnh có định tuyến cho phép ta liên kết phím nóng và các dạng nhập liệu khác như cơ chế phát động lệnh

D. Cả ba ưu điểm trên.

Trả lời: D

7. Trong mô hình lệnh có định tuyến, một khi một đơn vị xử lý lệnh đã được kích hoạt thực hiện:

A. Giống như sự kiện có định tuyến, lệnh lại được lan truyền tiếp, do vậy, có thể có nhiều đơn vị xử lý lệnh khác sẽ được thực hiện

B. Không đơn vị xử lý nào khác được gọi

C. Còn tuỳ lệnh đó có được đánh dấu "đã xử lý" hay chưa

Trả lời: B

Tài liệu tham khảo

1. Routed Events Overview, http://msdn.microsoft.com/en-us/library/ms742806.aspx

2. Event Handlers in WPF, http://www.kirupa.com/net/event_handlers_pg1.htm

3. Overview of routed events in WPF, http://joshsmithonwpf.wordpress.com/2007/06/22/overview-of-routed-events-in-wpf/

4. Introduction to the WPF Command Framework, http://www.devx.com/DevX/Article/37893/0/page/3

5. Commanding Overview, http://msdn.microsoft.com/en-us/library/ms752308.aspx

6. Understanding Routed Events and Commands In WPF, http://msdn.microsoft.com/en-us/magazine/cc785480.aspx