Шаблоны

Каждый элемент управления имеет встроенное средство, определяющее способ его визуализации (как группу более фундаментальных элементов). Это средство называется шаблоном элемента управления (*control template*) и определяется с помощью блока XAML-разметки.

Каждый элемент управления WPF спроектирован как не имеющий внешнего вида, в том смысле, что его внешний вид может быть полностью переопределен.

Пример

```
<ControlTemplate ... >
   <mwt:ButtonChrome Name="Chrome" ... >
      <ContentPresenter Content="{TemplateBinding ContentControl.Content}" ... />
   </mwt:ButtonChrome>
   <ControlTemplate.Triggers>
      ...
   </ControlTemplate.Triggers>
</ControlTemplate>
```

Шаблоны

Класс ButtonChrome определяет стандартные визуальные элементы кнопки, в то время как ContentPresenter хранит все содержимое. Для построения совершенно новой кнопки понадобится лишь создать новый шаблон элемента управления. Вместо ButtonChrome можно было бы использовать что-то другое, возможно, собственный специальный элемент или элемент, рисующий фигуру.

Типы шаблонов

В мире WPF существует три типа шаблонов, и все они наследуются от базового класса FrameworkTemplate.

Наряду с шаблонами элементов управления (представленными классом ControlTemplate) есть шаблоны данных (классы DataTemplate и HierarchicalDataTemplate), а также более специализированный шаблон панели для ItemsControl(ItemsPanelTemplate).

Шаблоны данных используются для извлечения данных из объекта и отображения их в элементе управления содержимым либо в индивидуальных позициях списочного элемента. Шаблоны данных незаменимы в сценариях привязки данных. В некоторой степени шаблоны данных и шаблоны элементов управления пересекаются. Например, оба типа шаблонов позволяют вставлять дополнительные элементы, применять форматирование и т.д. Однако шаблоны данных служат для добавления элементов внутрь существующего элемента управления.

Типы шаблонов

Шаблоны панелей применяются для управления компоновкой позиций в списочном элементе управления (элементов, унаследованных от класса ItemsControl). Например, их можно использовать для создания окон списков, которые располагают свои элементы слева направо и затем сверху вниз (вместо стандартного размещения сверху вниз в один столбец).

Классы Chrome

Класс ButtonChrome определен в пространстве имен Microsoft.Windows.Themes, которое содержит относительно небольшой набор сходных классов, визуализирующих базовые детали Windows. Наряду с ButtonChrome он включает BulletChrome (для флажков и переключателей), ScrollChrome (для полос прокрутки), ListBoxChrome и SystemDropShadowChrome.

Классы Chrome

На чуть более высоком уровне находится пространство имен System.Windows.Controls.Primitives, содержащее множество базовых элементов, которые можно использовать независимо, но гораздо чаще помещать в оболочки более удобных элементов управления. К ним относятся ScrollBar, **ResizeGrip** (для изменения размеров окна), **Thumb** (перетаскиваемая кнопка на полосе прокрутки), **TickBar**(дополнительный набор засечек на ползунке) и т.д. По сути, System.Windows.Controls.Primitives представляет готовые ингредиенты, которые можно применять в самых разных элементах управления, и которые не слишком полезны сами по себе, в то время как Microsoft.Windows.Themes содержит низкоуровневую логику рисования для визуализации этих деталей.

Создание шаблонов

Чтобы применить специальный шаблон элемента управления, необходимо просто установить свойство Template элемента. Хотя можно определить встроенный шаблон (поместив дескриптор шаблона элемента внутрь дескриптора самого элемента управления), этот подход редко бывает оправдан. Дело в том, что шаблон почти всегда нужно использовать многократно, создавая обложки для нескольких экземпляров одного и того же элемента управления. Шаблон элемента управления должен быть определен как ресурс, на который можно будет ссылаться с помощью StaticResource.

Примемр

```
<Window.Resources>
        <ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
            <Border BorderBrush="Orange" BorderThickness="3" CornerRadius="2"
                    TextBlock.Foreground="White">
                <Border.Background>
                    <LinearGradientBrush>
                        <GradientStopCollection>
                            <GradientStop Offset="0" Color="LimeGreen"></GradientStop>
                            <GradientStop Offset="1" Color="LightBlue"></GradientStop>
                        </GradientStopCollection>
                    </LinearGradientBrush>
                </Border.Background>
                <ContentPresenter RecognizesAccessKey="True"></ContentPresenter>
            </Border>
        </ControlTemplate>
</Window.Resources>
...
<Button Margin="10" Width="110" Padding="5" Height="30" Template="{StaticResource ButtonTemplate}">It's TemplateButton</Button>
```

Триггеры, изменяющие свойства

```
<ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
            <Border BorderBrush="Orange" BorderThickness="2" CornerRadius="2" Name="border"
                    TextBlock.Foreground="White">
                <Border.Background>
                    <LinearGradientBrush>
                        <GradientStopCollection>
                            <GradientStop Offset="0" Color="LimeGreen"></GradientStop>
                            <GradientStop Offset="1" Color="LightBlue"></GradientStop>
                        </GradientStopCollection>
                    </LinearGradientBrush>
                </Border.Background>
                <ContentPresenter RecognizesAccessKey="True" Margin="{TemplateBinding Padding}"></ContentPresenter>
            </Border>
            <ControlTemplate.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName="border" Property="Background" Value="Red"></Setter>
                </Trigger>
                <Trigger Property="IsPressed" Value="True">
                    <Setter TargetName="border" Property="BorderBrush" Value="Yellow"></Setter>
                </Trigger>
            </ControlTemplate.Triggers>
</ControlTemplate>
```

Поддержка фокуса

```
<Grid>
            <Rectangle Name="FocusCue" Visibility="Hidden" Stroke="Black"
                               StrokeThickness="1.5" StrokeDashArray="1 2" SnapsToDevicePixels="True"></Rectangle>
                    <ContentPresenter RecognizesAccessKey="True" Margin="{TemplateBinding Padding}"></ContentPresenter>
                </Grid>
            </Border>
            <ControlTemplate.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName="border" Property="Background" Value="Red"></Setter>
                </Trigger>
                <Trigger Property="IsPressed" Value="True">
                    <Setter TargetName="border" Property="BorderBrush" Value="Yellow"></Setter>
                </Trigger>
                <Trigger Property="IsKeyboardFocused" Value="True">
                    <Setter TargetName="FocusCue" Property="Visibility" Value="Visible"></Setter>
                </Trigger>
```

Автоматическое применение шаблонов

```
<Style TargetType="{x:Type Button}">
   <Setter Property="Control.Template" Value="{StaticResource ButtonTemplate}"
</Style>
```

Страничная навигация

Чтобы создать страничное приложение в WPF, нужно перестать применять для пользовательских интерфейсов в качестве контейнера высшего уровня класс Window и вместо него переключиться на класс System.Windows.Controls.Page.

Пример

```
<Page x:Class="WpfApplication1.Page1"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Page1" WindowTitle="Page1">
    <StackPanel Margin="5">
        <Label Margin="5">Пример обычной страницы</Label>
        <Button Margin="5,0,5,0" Padding="5">OK</Button>
        <Button Padding="5" Margin="5">Close</Button>
    </StackPanel>
</Page>
```

Глобальный файл

```
<Application x:Class="WpfApplication1.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="Page1.xaml">
    <Application.Resources></Application.Resources>
</Application>
```

Гиперссылки

Наиболее простой способ позволить пользователю перемещаться с одной страницы на другую — это гиперссылки. В WPF гиперссылки являются не отдельными, а внутристрочными потоковыми элементами, которые обязательно должны размещаться внутри другого поддерживающего их элемента. (Причина такого проектного решения связана с тем, что гиперссылки и текст часто используются вперемешку.)

Пример

```
<TextBlock Margin="5">
      Это обычная страница. А это <Hyperlink NavigateUri="page2.xaml">ссылка на другую страницу!</Hyperlink>
</TextBlock>
```

NavigationWindow включает две заметные кнопки: "назад" и "вперед" (если только они не скрыты установкой свойства Page.ShowsNavigationUI в false). Щелкая на этих кнопках, пользователи могут перемещаться по навигационной последовательности на одну страницу назад или вперед.

Размещение страниц

Элемент NavigationWindow является удобным контейнером, но не единственным вариантом. Страницы также можно размещать и непосредственно внутри других окон или даже внутри других страниц. Это подразумевает возможность создания чрезвычайно гибкой системы, поскольку означает, что одну и ту же страницу можно использовать многократно разными способами в зависимости от типа приложения, которое требуется создать.

Чтобы вставить страницу внутрь окна, нужно воспользоваться классом Frame. Класс Frame представляет собой элемент управления содержимым, который может удерживать любой элемент, но особенно полезен именно в качестве контейнера для страницы. Он включает свойство под названием Source, которое указывает на отображаемую страницу XAML.

Пример

```
<Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <TextBlock TextWrapping="Wrap" Margin="5">
            Данная страница находится в оконном приложении
        </TextBlock>
        <Frame Grid.Column="1" Source="Page1.xaml" Margin="3"
               BorderBrush="LimeGreen" BorderThickness="2"></Frame>
</Grid>
```

Служба навигации

Можно сделать так, чтобы значения свойств Hyperlink.NavigateUri и Frame.Source устанавливались динамически. Однако самый гибкий и мощный подход предполагает использование службы навигации WPF. Получать доступ к этой службе навигации можно через контейнер, в котором размещена страница (например, объект Frame или NavigationWindow), но такой подход ограничивает страницы использованием в контейнере только этого типа. Поэтому лучше всего получать доступ к службе навигации через статический метод NavigationService.GetNavigationService()

Служба навигации

В классе NavigationService определен набор методов для работы с навигацией. Наиболее часто используется метод Navigate(), который позволяет переходить на страницу по ее

```
nav.Navigate(new System.Uri("Page1.xaml", UriKind.RelatlveOrAbsolute));
```

```
Page1 nextPage = new Page1();
nav.Navigate(nextPage);
```

События навигации

Класс NavigationService также предоставляет полезный набор событий, которые можно использовать для реагирования на навигацию. Наиболее распространенной причиной реагирования на навигацию является необходимость выполнения по ее завершении какой-то задачи. Поскольку навигация осуществляется асинхронным образом, возврат из метода Navigate() происходит до появления целевой страницы. В некоторых случаях разница во времени может быть значительной.

Процесс навигации в WPF описан ниже:

- Определяется местонахождение страницы.
- Извлекается информация о странице. (Если страница находится на удаленном сайте, тогда она на этом этапе загружается.)
- Устанавливается местонахождение всех необходимых странице и связанных с ней ресурсов (например, изображений) и выполняется их загрузка.
- Осуществляется синтаксический анализ страницы и генерируется дерево ее объектов. На этом этапе страница запускает события Initialized (если только она не восстанавливается из журнала) и Loaded.
- Страница визуализируется.
- Если URI включает фрагмент, WPF переходит сразу же к этому элементу.

События навигации

Navigating

Процесс навигации начинается. Это событие можно отменить и тем самым предотвратить выполнение навигации

Navigated

Процесс навигации начался, но целевая страница еще не извлечена

LoadCompleted

Страница прошла синтаксический анализ. Однако события Initialized и Loaded еще не были сгенерированы

События навигации

FragmentNavigation

Страница готовится к прокручиванию до целевого элемента. Это событие срабатывает только в случае, если используется URI с информацией о фрагменте

NavigationStopped

Процесс навигации был отменен с помощью метода StopLoading()

NavigationFailed

Процесс навигации не удался из-за того, что не получилось обнаружить или загрузить целевую страницу. Это событие можно использовать для нейтрализации исключения до того, как оно появится и превратится в необработанное исключение приложения. Для этого необходимо просто установить NavigationFailedEventArgs.Handled в true

События навигации

NavigationProgress

Процесс навигации идет полным ходом, и часть данных страницы уже загружена. Это событие вызывается периодически для предоставления информации о ходе навигации. Оно предоставляет информацию об объеме данных, которые уже загружены (NavigationProgressEventArgs.BytesRead), и общем объеме данных, которые требуется загрузить (NavigationProgressEventArgs.MaxBytes). Это событие запускается после каждого извлечения данных объемом 1 Кбайт