第3章

XAML简介

Windows Phone 8 普通应用程序开发使用的是 Silverlight 的框架，Windows Phone 8 应用程序中的界面都是由 XAML 文件组成的，和这些 XAML 文件一一对应起来的是 XAML.CS 文件，这就是微软典型的 Code-Behind 模式的编程方式。XAML 文件的语法类似于 XML 和 HTML 的结合体，这是 Silverlight 程序特有的语法结构，本章将介绍有关 XAML 方面的知识和语法。

3.1 什么是 XAML

XAML 是一种声明性标记语言，如同应用于.NET Framework 编程模型一样，XAML 简化了为.NET Framework 应用程序创建 UI 的过程。在声明性 XAML 标记中可以创建可见的 UI 元素，然后使用代码隐藏文件(通过分部类定义与标记相连接)将 UI 定义与运行时逻辑相分离。XAML 直接以程序集中定义的一组特定后备类型表示对象的实例化，就如同其他的基于 XML 的标记语言一样，XAML 大体上也遵循 XML 的语法规则。例如，每个 XAML 元素包含一个名称以及一个或多个属性。在 XAML 中，每个属性都是和某个 Windows Phone Silverlight 类的属性相对应的，而且所有的元素名称都和 Windows Phone Silverlight 类库中定义的类名称相匹配。例如，<Button/>元素就和 System.Windows.Controls.Button 类对应。

XAML 是一个纯粹的标记语言，这也就意味着某个元素要实现一个事件的处理时，需要在该元素中通过特定的属性来指定相应的事件处理方法名，而真正的事件处理逻辑可以通过 C#或 VB.NET 语言实现，用户是无法通过 XAML 来编写相应的事件处理逻辑的。如果对 ASP.NET 技术比较了解的话，那么应该对代码后置这个概念不会陌生。对于一个 Windows Phone 程序来说，也可以像 ASP.NET 那样采用代码后置模型，将页面和相应的逻辑代码分别存放在不同的文件中，也可以以一种内联的方式将页面和逻辑代码都存放在同一个文件中。

XAML 开发人员应注意，声明一个 XAML 元素时，最好用 Name 属性为该元素指定一个名称，这样应用程序逻辑开发人员才可以通过代码来访问此元素。这是因为某种类型的

元素可能在 XAML 页面上声明多次，但是如果不显式地指明各个元素的 Name 属性，则无法区分哪个是想要操作的元素，也就无法通过 C＃或 VB.NET 来操作该元素和其中的属性。

下面是声明一个 XAML 元素必须遵循的 4 大原则：

(1) XAML 是大小写区分的，元素和属性的名称必须严格区分大小写，例如，对于 Button 元素来说，其在 XAML 中的声明应该为＜Button＞，而不是＜button＞；

(2) 所有的属性值，无论它是什么数据类型，都必须包含在双引号中；

(3) 所有的元素都必须是封闭的，也就是说，一个元素必须是自我封闭的，＜Button.../＞，或者是有一个起始标记和一个结束标记，例如＜Button＞...＜/Button＞；

(4) 最终的 XAML 文件也必须是合适的 XML 文档。

在 Silverlight 应用程序开发过程中，XAML 发挥着以下的作用：

(1) XAML 是用于声明 Silverlight UI 及该 UI 中元素的主要格式。通常，项目中至少有一个 XAML 文件表示应用程序中用于最初显示的 UI 页面。其他 XAML 文件可能声明其他用于导航 UI 或模式替换 UI 页面。另外一些 XAML 文件可以声明资源，如模板或其他可以重用或替换的应用程序元素。

(2) XAML 是用于声明样式和模板的格式，这些样式和模板应用于 Silverlight 控件和 UI 的逻辑基础。可以执行此操作来模板化现有控件，或作为为控件提供默认模板的控件作者来执行此操作。

(3) XAML 是用于为创建 Silverlight UI 和在不同设计器应用程序之间交换 UI 设计提供设计器支持的常见格式。最值得注意的是，Silverlight 应用程序的 XAML 可在 Expression Blend 产品与 Visual Studio 之间互换。

(4) Silverlight XAML 定义 UI 的可视外观，而关联的代码隐藏文件定义逻辑。可以对 UI 设计进行调整，而不必更改代码隐藏中的逻辑。就此作用而言，XAML 简化了负责主要可视化设计的人员与负责应用程序逻辑和信息设计的人员之间的工作交流。

(5) 由于支持可视化设计器和设计图面，因此，XAML 支持在早期开发阶段快速构造 UI 原型，并在整个开发过程中使设计的组成元素更可能保留为代码访问点，即使可视化设计发生了较大变化也不例外。

3.2 XAML 语法概述

编写 XAML 文件时，必须严格遵守 XAML 的语法，下面将介绍 XAML 的一些重要的语法。

3.2.1 XAML 命名空间

按照针对编程的广泛定义，命名空间确定如何解释引用编程实体的字符串标记。如果重复使用字符串标记，命名空间还可以解决多义性。命名空间概念的存在使得编程框架能

够区分用户声明的标记与框架声明的标记，并通过命名空间限定来消除可能的标记冲突，等等。XAML 命名空间是为 XML 语言提供此用途的命名空间概念。就 XAML 的常规作用及其面向 Silverlight 的应用程序而言，XAML 用于声明对象、这些对象的属性和对象-属性关系（表示为层次结构）。声明的对象由类型库提供支持，相关的库可以是以下任意一项：

（1）Silverlight 核心库（任何分布式运行时都提供）；

（2）分布式库，它们是在包中再分发的 Silverlight SDK 的一部分（可能带有应用程序库缓存选项）；

（3）表示应用程序中融入的和应用程序包再分发的第三方控件的定义的库；

（4）用户自己的库，这是用户通过 Silverlight 项目创建，用于容纳某些或所有应用程序的 Silverlight 用户代码的库；

（5）其他库，即用户在单独的项目中定义，通过 Silverlight 应用程序模型进行引用的库。

XAML 命名空间概念使用标记中提供的 XML 样式命名空间声明（xmlns），并将以 CLR 命名空间格式表示的后备类型信息和程序集信息与特定的 XAML 命名空间关联。这使得读取 XAML 文件的 XAML 处理器能够区分标记（markup）中的标记（token），并且在创建运行时对象表示形式时，该处理器能够从与该 XAML 命名空间关联的后备程序集中查找类型和成员。

XAML 文件几乎始终在其根元素中声明一个默认的 XAML 命名空间。默认 XAML 命名空间定义可以声明哪些元素，而无需通过前缀进一步进行限定。例如，用户声明一个元素＜Balloon/＞，则该元素 Balloon 应存在且在默认 XAML 命名空间中有效。相反，如果 Balloon 不在所定义的默认 XAML 命名空间中，则必须转而使用一个前缀来限定该引用。例如，＜party:Balloon/＞，该前缀指示此实体存在于与默认命名空间不同的 XAML 命名空间中，尤其是，用户已将某个 XAML 命名空间映射到前缀 party 以便于使用。

XAML 命名空间应用于声明它们的特定元素，同时应用于 XAML 结构中该元素所包含的任何元素。因此，XAML 命名空间几乎始终在根元素上声明，以充分利用此继承概念。

来自除 Silverlight 核心库之外的其他库的类型将要求用户使用前缀声明和映射 XAML 命名空间，然后才能从该库中引用类型。针对默认命名空间之外的其他 XAML 命名空间的 XAML 命名空间声明提供了 3 项信息：

（1）一个前缀，它定义用于在后续 XAML 标记中引用该 XAML 命名空间的标记（markup）标记（token）；

（2）在该 XAML 命名空间中定义元素的后备类型的程序集，XAML 处理器必须访问此程序集才能基于 XAML 声明创建对象；

（3）该程序集中的一个 CLR 命名空间。

SDK 库具有 CLR 特性，以便加载程序集的设计器可以建议使用特定的前缀。在 Visual Studio 中，对于已由某个项目引用的任何程序集，都可以使用自动完成功能从所引用的程序中读取 CLR 特性。这一 Visual Studio 功能要么将所有可能的 XAML 命名空间

显示为下拉列表,要么使用建议的前缀作为提示以帮助建议特定的映射选择。

在几乎每个 Silverlight XAML 文件中声明的一个特定的 XAML 命名空间是针对由 XAML 语言定义的元素的 XAML 命名空间。根据约定,XAML 语言 XAML 命名空间映射到前缀 x:。Silverlight 项目的默认项目和文件模板始终同时将默认的 XAML 命名空间(无前缀,只有 xmlns=)和 XAML 语言命名空间(映射到前缀 x:)定义为根元素的一部分。例如:

```
< phone:PhoneApplicationPage
  xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"
...>
```

“x:前缀”类型的命名空间包含多个将在 SilverlightXAML 中频繁使用的编程构造。下面列出了最常见的“x:前缀”类型的命名空间构造:

(1) x:Key: 为 ResourceDictionary 中的每个资源设置一个唯一键。

(2) x:Class: 指定为 XAML 页提供代码隐藏的类的 CLR 命名空间和类名称,并命名由标记编译器在 Silverlight 应用程序模型中创建的类。必须具有一个这样的类才能支持代码隐藏或支持初始化为 RootVisual。

(3) x:Name: 处理 XAML 中定义的对象元素后,为运行时代码中存在的实例指定运行时对象名称。对于不支持 FrameworkElement.Name 属性的情形,可以将 x:Name 用于元素命名方案。默认情况下,通过处理对象元素而创建的对象实例没有可供在代码中使用的唯一标识符或固有的对象引用。在代码中调用构造函数时,几乎总是使用构造函数结果为构造的实例设置一个变量,以便以后在代码中引用该实例。为了对通过标记定义创建的对象进行标准化访问,XAML 定义了 x:Name 属性。可以在任何对象元素上设置 x:Name 属性的值。在代码隐藏文件中,所选择的标识符等效于引用构造实例的实例变量。在任何方面,命名元素都像它们是对象实例一样工作(此名称只是引用该实例),并且代码隐藏文件可以引用该命名元素来处理应用程序内的运行时交互。

3.2.2 声明对象

一个 XAML 文件始终只有一个元素作为其根,该元素声明的一个对象将作为某些编程结构(如页面)的概念根,或者是应用程序的整个运行时定义的对象图。

根据 XAML 语法,可以通过 3 种方法在 XAML 中声明对象:

1) 直接使用对象元素语法

直接使用对象元素语法是使用开始标记和结束标记将对象实例化为 XML 格式的元素。可以使用此语法声明根对象或创建用于设置属性值的嵌套对象。

2) 间接使用属性语法

间接使用属性语法是使用内联字符串值声明对象。在概念上,这可能用于实例化除根之外的任何对象。可以使用此语法设置属性值。这是一个针对 XAML 处理器的间接操作,

因为必须有某个过程知道如何在了解正设置哪个属性、该属性的类型系统特性和所提供的字符串值的基础上创建新对象。通常，这表明相关类型或属性要么支持可处理字符串输入的类型转换器，要么 XAML 分析器支持进行本机转换。

3）使用标记扩展

这并不意味着始终可以选择使用任何语法以给定的 XAML 词汇创建对象。词汇中的某些对象只能使用对象元素语法创建。少量对象只能通过初始设置为属性值来创建。事实上，在 Silverlight 中，可以使用对象元素或属性语法创建的对象比较少。即使这两种语法格式都是可能的，也只有其中一种语法格式占主流或是最适合方案使用的格式。除了以等同于实例化对象的方式声明对象之外，XAML 中还提供了一些可用来引用现有对象的方法。这些对象可能在 XAML 的其他区域中定义，或者通过平台及其应用程序或编程模型的某种行为隐式存在。

若要使用对象元素语法声明对象，需要使用以下模式编写标记，其中，objectName 是要实例化的类型的名称。在本文档中，经常出现术语“对象元素用法”，这是用于用对象元素语法创建对象的特定标记的简称。

```
<objectName>
</objectName>
```

下面的示例是用于声明 Canvas 对象的对象元素用法。

```
<Canvas></Canvas>
```

许多 Silverlight 对象（例如 Canvas）可以包含其他对象。

```
<Canvas>
  <Rectangle>
  </Rectangle>
</Canvas>
```

为方便起见（且作为 XAML 与 XML 的一般关系的一部分），如果对象不包含其他对象，则可以使用一个自结束标记（而不是开始/结束标记对）来声明对象元素，如下面示例中的 ＜Rectangle /＞ 标记所示。

```
<Canvas>
  <Rectangle />
</Canvas>
```

在某些情况下，属性（Property）值并不只是语言基元（如字符串），此时可以使用属性（Attribute）语法来实例化设置该属性（Property）的对象，并初始化用于定义新对象的键属性（Property）。由于此行为绑定到属性（Property）设置，请参见后面有关如何使用属性（Attribute）语法在一个语法步骤中声明对象并设置其属性（Property）的信息。

3.2.3 设置属性

可以设置使用对象元素语法声明的对象的属性。可以通过多种方法使用 XAML 设置

属性：

(1) 使用属性语法；

(2) 使用属性元素语法；

(3) 使用内容元素语法；

(4) 使用集合语法(通常是隐式集合语法)。

对于对象声明，用于在 XAML 中设置对象属性的此方法列表并不表示可以使用这些方法中的任何一种来设置给定的属性。某些属性只支持其中一种方法，某些属性(Property)可能支持组合，例如，支持内容元素语法的属性(Property)可能还通过属性(Property)元素语法或备选属性(Attribute)语法支持更详细的格式。这取决于属性和属性使用的对象类型。可用 XAML 设置的每个属性的参考页的"XAML 用法"部分指出了使用 XAML 语法的可能性。Silverlight 中的对象还有一些无论使用任何方式都无法使用 XAML 设置的属性，只能使用代码来设置这些属性。

无论使用任何方式(包括 XAML 或代码)都无法设置只读属性，除非有其他机制适用。该机制可能是调用一个设置为属性的内部表示形式的构造函数重载、一个并非是严格意义上的属性访问器的帮助器方法或一个计算属性。计算属性依赖于其他可设置属性的值，以及服务或行为对该属性值的可能影响，而这些功能在依赖项属性系统中提供。

1. 使用属性(Attribute)语法设置属性(Property)

设置属性使用以下语法。其中 objectName 是要实例化的对象，propertyName 是要对该对象设置的属性的名称，propertyValue 是要设置的值。

```
<objectName propertyName = "propertyValue" .../>
```

或者

```
<objectName propertyName = "propertyValue">
...
</objectName>
```

使用上述任何一种语法都可以声明对象并设置该对象的属性。虽然第一个示例是标记中的单一元素，实际上这里有一些与 XAML 处理器如何分析此标记有关的分离步骤。首先，对象元素的存在表明必须实例化新的 objectName 对象。只有存在这样的实例后，才可以对它设置实例属性 propertyName。

下面的示例使用 4 个属性(Attribute)的属性(Attribute)语法来设置 Rectangle 对象的 Name、Width、Height 和 Fill 属性(Property)。

```
<Rectangle Name = "rectangle1" Width = "100" Height = "100" Fill = "Blue" />
```

如果清楚地了解 XAML 分析器如何解释此标记和定义对象树，则等效的代码可能类似以下伪代码：

```
Rectangle rectangle1 = new Rectangle();
rectangle1.Width = 100.0;
rectangle1.Height = 100.0;
```

```
rectangle1.Fill = new SolidColorBrush(Colors.Blue);
```

许多 Silverlight 属性可以使用属性元素语法来设置。若要使用属性元素语法，必须能够指定对象元素的新实例才能“填充”属性元素值。

若要使用属性元素语法，需要为要设置的属性创建 XML 元素。这些元素的形式为＜object.property＞。在标准的 XML 中，此元素只被视为在名称中有一个点的元素。但是使用 XAML 时，元素名称中的点将该元素标识为属性元素，且 property 是 object 的属性。

在下面的语法中，property 是要设置属性的名称，propertyValueAsObjectElement 是声明新对象的新对象元素，其值类型是该属性期望的值。

```
<object>
 <对象.属性>
propertyValueAsObjectElement
 </对象.属性>
</object>
```

下面的示例使用属性元素语法通过 SolidColorBrush 对象元素来设置 Rectangle 的填充。（在 SolidColorBrush 中，Color 使用属性语法来设置。）此 XAML 的呈现结果等同于前面使用属性语法设置 Fill 的 XAML 示例：

```
<Rectangle
  Name = "rectangle1"
  Width = "100"
  Height = "100"
>
  <Rectangle.Fill>
    <SolidColorBrush Color = "Blue"/>
  </Rectangle.Fill>
</Rectangle>
```

2. 使用 XAML 内容语法设置属性

一些 Silverlight 类型定义了一个启用 XAML 内容元素语法的属性。在 XAML 内容元素语法中，可以忽略该属性的属性元素，并可以通过提供所属类型的对象元素标记中的内容来设置该属性。该内容通常为一个或多个对象元素。这称为 XAML 内容语法。如果 XAML 内容语法可用，则 Silverlight 参考文档中针对该属性的“XAML 用法”部分将显示该语法。

例如，Border 的 Child 属性页显示了 XAML 内容语法（而非属性元素语法），以设置 Border 的单一对象 Child 值。下面的示例与这一用法类似：

```
<Border>
  <Button .../>
</Border>
```

如果声明为 XAML 内容属性的属性也支持“松散”对象模型（在此模型中，属性类型为 Object，或具体而言为类型 String），则可以使用 XAML 内容语法将纯字符串作为内容放入开始对象标记与结束对象标记之间。例如，TextBlock 的 Text 属性（Property）页显示了另一种 XAML 语法，该语法使用 XAML 内容语法（而不是属性（Attribute）语法）来为 Text

设置一个字符串值。下面的示例说明该用法并设置 TextBlock 的 Text 属性，而不显式指定 Text 属性。Text 使用将 XML 视为内容或“内部文本”的内容进行设置，而不是通过使用属性或声明对象元素来设置。

```
<TextBlock>Hello!</TextBlock>
```

3. 使用集合语法设置属性

在 XAML 中，有几个集合语法的变体。这看上去似乎允许“设置”只读集合属性。而实际上，XAML 允许的操作是向集合中添加项。实现 XAML 支持的 XAML 语言和 XAML 处理器依赖于后备集合类型中的约定来启用此语法。

通常，XAML 语法中不存在保留集合项的集合类型的属性（如索引器或 Items 属性）。对于集合而言，XAML 中的集合实际所需的未必是属性，而是方法——Add 方法。调用 Add 方法就是上述约定。当 XAML 处理器遇到 XAML 集合语法中的一个或多个对象元素时，首先通过使用其对象标记创建每个此类对象，然后通过调用集合的 Add 方法以声明顺序将每个新对象添加到包含集合中。

下面的示例演示了一个使用可构造集合类型的集合属性（可以定义实际的集合并将其实例化为 XAML 中的一个对象元素）：

```
<LinerGradientBrush>
  <LinearGradientBrush.GradientStops>
    <GradientStopCollection>
      <GradientStop Offset="0.0" Color="Red" />
      <GradientStop Offset="1.0" Color="Blue" />
    </GradientStopCollection>
  </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

不过，对于采用集合的 Silverlight 属性而言，XAML 分析器可根据集合所属的属性隐式知道集合的后备类型。因此，可以省略集合本身的对象元素，如下所示：

```
<LinearGradientBrush>
  <LinearGradientBrush.GradientStops>
    <GradientStop Offset="0.0" Color="Red" />
    <GradientStop Offset="1.0" Color="Blue" />
  </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

另外，有一些属性不但是集合属性，还标识为类的 XAML 内容属性。前面示例中以及许多其他 Silverlight 属性中使用的 GradientStops 属性就是这种情况。在这些语法中，也可以省略属性元素。这生成以下标记：

```
<LinearGradientBrush>
  <GradientStop Offset="0.0" Color="Red" />
  <GradientStop Offset="1.0" Color="Blue" />
</LinearGradientBrush>
```

在广泛用于控件合成的类（如面板、视图或项控件）中，此集合和内容语法的组合是最常

见的。例如，下面的示例演示将两个 UI 元素合成到一个 StackPanel 中的显式 XAML 以及可能的最简单 XAML：

```
<StackPanel>
  <StackPanel.Children>
    <!--UIElementCollection-->
    <TextBlock>Hello</TextBlock>
    <TextBlock>World</TextBlock>
    <!--/UIElementCollection-->
  </StackPanel.Children>
</StackPanel>
<tackPanel>
  <TextBlock>Hello</TextBlock>
  <TextBlock>World</TextBlock>
</StackPanel>
```

请注意显式语法中注释掉的 UIElementCollection。将其注释掉是因为即使将在对象树中创建相关集合，也无法在 XAML 中显式指定它。这是因为 UIElementCollection 不是可构造的类。在运行时对象树中获取的值是所属类中的一个默认初始化值，在初始化之后无法更改此值。在某些情况下，标记中会特意且显式包含集合类（例如，赋予集合一个 x:Name，以便可以在代码中更方便地引用该集合）。但是，注意不要显式声明由于其后备类型的特征而无法由 XAML 分析器构造的集合类。

4. 何时使用属性(Attribute)语法或属性(Property)元素语法来设置属性(property)

所有支持使用 XAML 设置的属性（Property）都支持用于直接值设置的属性(Attribute)语法或属性(Property)元素语法，但可能不会互换支持每种语法。某些属性支持上述两种语法，某些属性还支持其他语法选项（如前面所示的 Text 的内容元素语法）。属性支持的 XAML 语法的类型在某种程度上取决于该属性用作其属性类型的对象的类型。如果该属性(Property)类型为基元类型（如双精度、整型或字符串），则该属性(Property)始终支持属性(Attribute)语法。

下面的示例使用属性语法设置 Rectangle 的宽度。Width 属性(Property)支持属性(Attribute)语法，这是因为属性(Property)值是双精度值。

```
<Rectangle Width="100" />
```

如果可以通过对字符串进行类型转换来创建用于设置某属性(Property)的对象类型，也可以使用属性(Attribute)语法来设置该属性(Property)。对于基元，始终是这种情况。但是，某些其他对象类型也可以使用指定为属性值的字符串（而不是需要对象元素语法）来创建。此方法使用该特定属性或该属性类型通常所支持的基本类型转换。属性(Attribute)的字符串值经过分析后，字符串信息用于设置对新对象的初始化非常重要的属性(Property)。特定类型转换器还可能创建公共属性类型的不同子类，这取决于它处理字符串中的信息的独特方式。

下面的示例使用属性语法设置 Rectangle 的填充。当使用 SolidColorBrush 设置 Fill

属性(property)时,该属性(Property)支持属性(Attribute)语法。这是因为支持 Fill 属性(Property)的 Brush 抽象类型支持类型转换语法,该语法可以创建一个通过将属性(Attribute)指定的字符串作为其 Color 来初始化的 SolidColorBrush。

```
<Rectangle Width = "100" Height = "100" Fill = "Blue" />
```

如果用于设置某属性的对象支持对象元素语法,则可以使用属性元素语法来设置该属性。如果该对象支持对象元素语法,该属性也支持属性元素语法。下面的示例使用属性元素语法设置 Rectangle 的填充。当使用 SolidColorBrush 设置 Fill 属性时,该属性支持属性元素语法,这是因为 SolidColorBrush 支持对象元素语法并满足该属性的使用 Brush 类型设置其值的要求。(SolidColorBrush 也使用属性(Attribute)语法设置了其 Color 属性(Property),此 XAML 的呈现结果等同于前面使用属性语法设置 Fill 的 XAML 示例。)

```
<Rectangle Width = "100" Height = "100">
  <Rectangle.Fill>
    <SolidColorBrush Color = "Blue"/>
  </Rectangle.Fill>
</Rectangle>
```

3.2.4 标记扩展

标记扩展是一个在 Silverlight XAML 实现中广泛使用的 XAML 语言概念。在 XAML 属性语法中,花括号"{"和"}}"表示标记扩展用法。此用法指示 XAML 处理不要像通常那样将属性值视为文本字符串或者视为可直接转换为文本字符串的值。相反,分析器通常应调用支持该特定标记扩展的代码,该标记扩展可帮助从标记中构造对象树。

Silverlight 支持在其默认的 Silverlight XAML 命名空间下定义且其 XAML 分析器可以理解的以下标记扩展:

(1) 绑定:支持数据绑定,此绑定将延迟属性值,直至在数据上下文中解释此值。

(2) StaticResource:支持引用在 ResourceDictionary 中定义的资源值。

(3) TemplateBinding:支持 XAML 中可与模板化对象的代码属性交互的控件模板。

(4) RelativeSource:启用特定形式的模板绑定。

采用引用类型值(类型没有转换器)的属性需要属性元素语法(该语法始终创建新实例)或通过标记扩展的对象引用。Silverlight 标记扩展通常返回一个现有实例或将值延迟到运行时。通过使用标记扩展,每个可使用 XAML 设置的属性(Property)都可能在属性(Attribute)语法中设置。即使属性(Property)不支持对直接对象实例化使用属性(Attribute)语法,也可以使用属性(Attribute)语法为属性(Property)提供引用值;或者可以使特定行为能够符合用值类型或实时创建的引用类型填充 XAML 属性(Property)这一常规要求。

例如,下面的 XAML 使用属性(Attribute)语法设置 Border 的 Style 属性(Property)的值。Style 属性(Property)采用了 Style 类的实例,这是默认情况下无法使用属性(Attribute)语法字符串创建的引用类型。但在本例中,属性(Attribute)引用了特定的标记

扩展 StaticResource。当处理该标记扩展时，它返回对以前在资源字典中定义为键控资源的某个样式的引用。

```
<Canvas.Resources>
  <Style TargetType="Border" x:Key="PageBackground">
    <Setter Property="BorderBrush" Value="Blue"/>
    <Setter Property="BorderThickness" Value="5"/>
  </Style>
</Canvas.Resources>
...
<Border Style="{StaticResource PageBackground}">
  ...
</Border>
```

在许多情况下，可以使用标记扩展来提供一个作为对现有对象的引用的值，或者标记扩展可以提供一个可以以属性(Attribute)格式设置属性(Property)的对象。文本"{"值因为左花括号符号"{"是标记扩展序列的开始标记，所以必须使用转义符序列，以便指定以"{"开头的文本字符值。转义序列是"{}"。例如，若要指定作为单个左花括号的字符值，请将属性值指定为"{"。还可以在某些情况下使用替代引号(如""分隔的属性值内的')，以便将"{"值作为字符串提供。

3.2.5 事件

XAML 是用于对象及其属性的声明性语言，但它也可以包含用于将事件处理程序附加到标记中的对象的语法。接着，可以通过特定的技术(如 Silverlight)扩展 XAML 事件语法约定，这会通过编程模型集成 XAML 声明的事件。可以将相关事件的名称指定为处理该事件的对象的属性名称。对于属性值，可以指定在代码中定义的事件处理程序函数的名称。XAML 处理器使用此名称在加载的对象树中创建一个委托表示形式，并将指定的处理程序添加到内部处理程序列表中。

大多数基于 Silverlight 的应用程序都是由标记和代码隐藏源生成的。在一个项目中，XAML 被编写为.xaml 文件，而使用 CLR 语言(如 VisualBasic 或 C#)编写代码隐藏文件。编译 XAML 文件时，通过将一个命名空间和类指定为 XAML 页的根元素的 x:Class 属性来确定每个 XAML 页的 XAML 代码隐藏文件的位置。