项目一　WPF 基础——“Hello World!”程序制作

本项目将隆重地向学习者介绍 WPF 技术的定义，WPF 技术的产生、发展和现状，WPF 技术的主要特点和应用场合；同时还重点介绍 WPF 技术中最重要的内容 XAML（可扩展应用程序标记语言）。让学习者对正在学习的 WPF 技术有个清晰和正确的认识，并在学习中应用所学完成第一个 WPF 应用程序——“Hello World!”程序，它的运行效果如图 1-1 所示。

图 1-1　Hello World!程序运行效果

1. 了解 WPF 的产生、发展和现状。
2. 掌握常用 XAML 标记及其属性。
3. 熟悉 Visual Studio 2012 集成开发环境。

1. 会安装 Visual Studio 2012 开发工具。
2. 会编写常见 XAML 标记。
3. 会配置常见 XAML 标记的属性。
4. 会使用 Visual Studio 2012 创建、运行、调试 WPF 程序。

任务 1.1　搭建 WPF 开发环境

【任务描述】

本任务将介绍 WPF 技术的定义，WPF 技术的产生、发展和现状，WPF 技术的主要特点

和应用场合；再介绍 WPF 技术的开发工具有哪些、如何获取和各自的特点；然后选择当前主流的开发工具，介绍其软硬件环境需求，并完成安装和配置。

【知识准备】

1.1.1 WPF 是什么

WPF 是 Windows Presentation Foundation 的首字母缩写，中文译为“Windows 呈现基础”，因为与“我佩服”拼音首字母组合一样，国内有人调侃地称之为“我佩服”。WPF 由 .NET Framework 3.0 开始引入，与 Windows Communication Foundation（WCF，是由微软开发的一组数据通信的应用程序开发接口）及 Windows Workflow Foundation（WWF，是一项微软技术，用于定义、运行和管理工作流程）一并作为新一代 Windows 操作系统以及.NET 框架的三个重大应用程序开发类库。

WPF 是微软的新一代图形系统，为用户界面、2D/3D 图形、文档和媒体提供了统一的描述和操作方法。基于 DirectX 技术的 WPF 不仅带来了前所未有的 3D 界面，而且其图形向量渲染引擎也大大改进了传统的 2D 界面，比如从 Vista 操作系统开始的 Windows 中的半透明效果的窗体等都得益于 WPF。程序员在 WPF 的帮助下，要开发出媲美 Mac 程序的炫酷界面已不再是遥不可及的奢望。WPF 相对于 Windows 客户端的开发来说，向前跨出了巨大的一步，它提供了超丰富的.NET UI 框架，集成了矢量图形、丰富的流动文字支持（Flow Text Support）、3D 视觉效果和强大无比的控件模型框架。

WPF 是 Windows 操作系统中的一次重大变革，与早期的 GDI+/GDI 不同，WPF 是基于 DirectX 引擎的，支持 GPU 硬件加速，在不支持硬件加速时也可以使用软件绘制，提高使用者的体验，能自动识别显示器分辨率并进行缩放。WPF 统一了 Windows 创建、显示和操作文档、媒体和用户界面（UI）的方式，使开发人员和设计人员可以创建更好的视觉效果、不同的用户体验。Windows Presentation Foundation 发布后，Windows XP、Windows Server 2003 和以后所有的 Windows 操作系统版本都可以使用它。

WPF 的核心是一个与分辨率无关并且基于向量的呈现引擎，旨在利用现代图形硬件的优势。WPF 通过一整套应用程序开发功能扩展了这个核心，这些功能包括可扩展应用程序标记语言（XAML）、控件、数据绑定、布局、二维和三维图形、动画、样式、模板、文档、媒体、文本和版式。WPF 包含在 Microsoft .NET Framework 中，使您能够生成融入了 .NET Framework 类库的其他元素的应用程序。

1.1.2 WPF 的特点

（1）矢量图的超强支持。

WPF 兼容支持 2D 绘图，比如矩形、自定义路径、位图等；文字显示的增强、XPS 和消锯齿；强大的三维支持，包括 3D 控件及事件；与 2D 及视频合并打造更立体的效果；渐变、使用高精确的（ARGP）颜色，支持浮点类型的像素坐标。这些都远超 GDI+的功能。

（2）灵活、易扩展的动画机制。

.NET Framework 3.0 及更高版本类库提供了强大的基类，只需继承即可实现自定义程序使用绘制；接口设计非常直观，完全面向对象的对象模型，使用对象描述语言 XAML，使用开发工具的可视化编辑。WPF 可以使用任何一种.NET 编程语言（C#、VB.NET 等开发语言）进

行开发。XAML 主要针对界面的可视化控件描述，其后台处理程序为.cs 或.vb 文件，并最后将编译为 CLR 中间运行语言。

（3）WPF 为 Windows 客户端应用程序开发提供了更多的编程增强功能。

一个明显的增强功能就是使用标记和代码隐藏开发应用程序的功能（类似于 ASP.NET 动态网站程序开发）。通常使用可扩展应用程序标记语言（XAML）标记实现应用程序的外观，而使用托管编程语言（代码隐藏）实现其行为。这种外观和行为的分离具有以下优点：

- 降低了开发和维护成本，因为外观特定的标记并没有与行为特定的代码紧密耦合。
- 开发效率更高，因为设计人员可以在开发人员实现应用程序行为的同时实现应用程序的外观。
- 可以使用多种设计工具实现和共享 XAML 标记，以满足应用程序开发参与者的要求——Microsoft Expression Blend 提供了适合设计人员的体验，而 Visual Studio 2012（或其他版本）针对开发人员。
- WPF 应用程序的全球化和本地化得以大大简化。

1.1.3　WPF 的组成结构

WPF 由两个主要部分组成：引擎和编程框架，如图 1-2 所示。

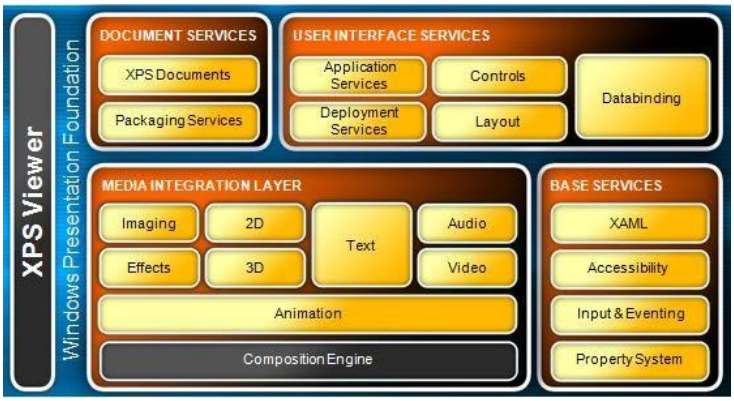


图 1-2　WPF 的组成结构

1. WPF 引擎

WPF 引擎统一了开发人员和设计人员体验文档、媒体和 UI 的方式，为基于浏览器的体验、基于窗体的应用程序、图形、视频、音频和文档提供了一个单一的运行时库。WPF 使得应用程序不仅能够充分利用现代计算机中现有的图形硬件的全部功能，而且能够利用硬件将来的进步。例如，WPF 基于矢量的呈现引擎使应用程序可以灵活地利用高 DPI 监视器，而无需开发人员或用户进行额外的工作。同样，当 WPF 检测到支持硬件加速的视频卡时，它将利用硬件加速功能。

2. WPF 编程框架

WPF 框架为媒体、用户界面设计和文档提供的解决方案远远超过开发人员现在所拥有的。它的设计考虑了可扩展性，使开发人员可以完全在 WPF 引擎的基础上创建自己的控件，也可以通过对现有的 WPF 控件进行再分类来创建自己的控件。WPF 编程框架的核心是用于形状、文档、图像、视频、动画、三维以及用于放置控件和内容的面板的一系列控件。这些“自有控

件”为开发下一代用户体验提供了构造块。

微软在引入 WPF 的同时，还引入了 XAML，这是一种公开表示 Windows 应用程序用户界面的标记语言，可使开发人员和设计人员用来构建和重用 UI 的工具更加丰富。对于 Web 开发人员，XAML 提供了熟悉的 UI 说明模式。XAML 还使 UI 设计从基础代码中分离出来，从而使开发人员和设计人员之间的合作更加紧密。

1.1.4 WPF 和 Silverlight 的关系

1. 什么是 Silverlight

Microsoft Silverlight 的中文名为“微软银光”，它是一个跨浏览器的、跨平台的插件，为网络带来下一代基于.NET Framework 的媒体体验和丰富的交互式应用程序。Silverlight 提供灵活的编程模型，并可以很方便地集成到现有的网络应用程序中。Silverlight 可以对运行在 Mac 或 Windows 上的主流浏览器提供高质量视频信息的快速、低成本的传递。借助该技术，用户将拥有内容丰富、视觉效果绚丽的交互式体验，而且无论是在浏览器内还是在桌面操作系统（如 Windows 和 Apple Macintosh）中，您都可以获得这种一致的体验。

对于互联网用户来说，Silverlight 是一个安装简单的浏览器插件程序。用户只要安装了这个插件程序，就可以在 Windows 和 Macintosh 上的多种浏览器中运行相应版本的 Silverlight 应用程序，享受视频分享、在线游戏、广告动画、交互丰富的网络服务等。

对于开发设计人员而言，Silverlight 是一种融合了微软的多种技术的 Web 呈现技术。它提供了一套开发框架，并通过使用基于向量的图像图层技术支持任何尺寸图像的无缝整合，对基于 ASP.NET、AJAX 在内的 Web 开发环境实现了无缝连接。Silverlight 使开发设计人员能够更好地协作，有效地创造出能在 Windows 和 Macintosh 上的多种浏览器中运行的内容丰富、界面绚丽的 Web 应用程序——Silverlight 应用程序。

简而言之，Silverlight 是一个跨浏览器、跨平台的插件，为网络带来下一代基于.NET 媒体体验和丰富的交互式应用程序。对运行在 Macintosh 和 Windows 上的主流浏览器，Silverlight 提供了统一而丰富的用户体验，通过 Silverlight 这个小小的浏览器插件，视频、交互性内容，以及其他应用能完好地融合在一起。

2. Silverlight 的特点

Silverlight 是基于浏览器插件的，在浏览器中运行，服务器端不需要部署任何环境，其交互式及动画等网页功能比较突出。但是它也类似于 Flash 应用客户端需要 Flash Player 一样，Silverlight 应用客户端也要安装相应的支持库才能显示。

3. Silverlight 和 WPF 的关系

Silverlight 作为 WPF 的一个轻量级的精简版本，曾经叫做 WPF/E（Windows Presentation Foundation/Everywhere）。其中 Everywhere 指的是跨平台的意思，使得在每个操作系统中可以运行 WPF。

因为跨平台特性，所以使用的是插件技术。为了网站设计的安全性，不能将 WPF 的全部能力和权限都提供给 Silverlight，因此就提取了一个精简的.NET Runtime Library 到了 WPF/E 中来执行 XAML 文件，去除了文件操作、Windows API、3D 控件、视频加速等类库方法。所以从核心本质上分开，说两者的关系更像是兄弟关系，或者说 Silverlight 是 WPF 的子集。当然随着 Silverlight 的发展，微软公司又结合其需求增加了很多新的个性功能特征。

【任务分析】

完成前面的知识准备，现在来对部署 WPF 开发环境任务进行分析。

WPF 从一开始就被微软公司设计为设计和开发分离形态，其专业的动画和美工设计部分使用 Blend 工具，普通界面和代码设计部分使用 Visual Studio。两个工具可以相互合作完成一个综合任务制作。

因此，部署 WPF 的开发环境我们需要安装至少两个工具软件：Microsoft Expression Blend 和 Microsoft Visual Studio。

1. Microsoft Expression Blend。

Microsoft Expression Blend 作为一款功能齐全的专业设计工具，可用来针对基于 Microsoft Windows 和基于 Microsoft Silverlight 1.0 的应用程序制作精美复杂的用户界面。Expression Blend 可让设计人员集中精力从事创作，而让开发人员集中精力从事编程工作，其开发界面如图 1-3 所示。

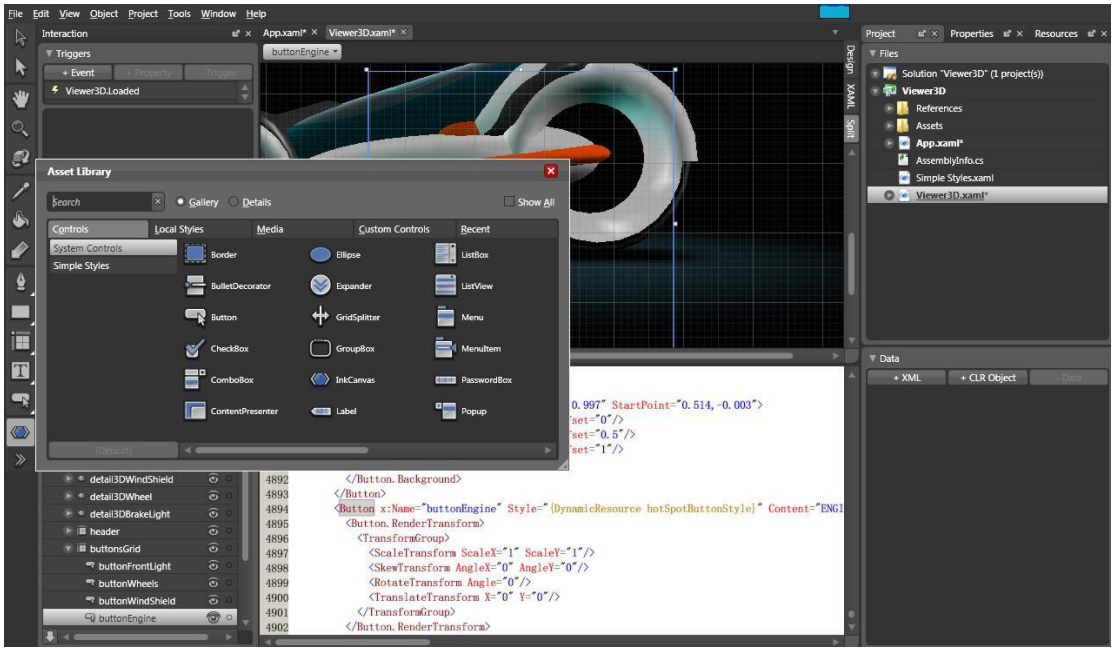


图 1-3　Blend 主界面

从 2012 版开始 Blend 就作为独立工具和 Visual Studio 一起提供，也就是说安装 Visual Studio 2012 版及更新的版本时就可以同时获得 Blend 和 Visual Studio 两个工具。

2. Microsoft Visual Studio

Microsoft Visual Studio（VS）是美国微软公司的开发工具包系列产品。VS 是一个基本完整的开发工具集，它包括了整个软件生命周期中所需要的大部分工具，如 UML 工具、代码管控工具、集成开发环境（IDE）等。所写的目标代码适用于微软支持的所有平台，包括 Microsoft Windows、Windows Mobile、Windows CE、.NET Framework、.NET Compact Framework、Microsoft Silverlight 和 Windows Phone。

Visual Studio 是目前最流行的 Windows 平台应用程序的集成开发环境，最新版本为 Visual

Studio 2013 版，基于.NET Framework 4.5.1。基于软件体积、性能及其对软硬件环境的需求综合考虑，本教材选用目前最为主流应用的 Visual Studio 2012 版进行介绍，后续的软件开发环境默认都是 Visual Studio 2012 版。

【任务实施】

1. 获得 Visual Studio 开发工具安装程序

安装程序的获得可以从专门的正版软件商店购买，也可以选择使用免费体验版或者是完全免费的部分版本。前者花费不菲，但可以获得完整的功能和支持服务；后者直接从网络获得，无需付费，但是使用时间或者功能有限制。

2. Visual Studio 的安装

这里以功能最完整的 Visual Studio Ultimate 2012 为例进行介绍。

（1）准备安装的软硬件环境。

目标计算机硬件配置至少应该是双核处理器，内存 1GB 以上，有 9 个 GB 左右的空余硬盘存储空间（见后面的安装提示）；软件方面操作系统应该是 Windows 7 及以上操作系统。如果是为开发选择新计算机时 CPU 和内存是最重要的部分，建议高配。

（2）安装过程。

解压安装包，单击 vs_ultimate.exe 运行，如图 1-4 所示。

图 1-4　Visual Stuido 2012 安装初始界面

从安装界面中就可以看出这个版本自带了 Blend 工具（with Blend）。可以结合计算机硬盘分区空间选择安装路径，勾选“我同意许可条款和条件”复选项，第二个选项可以不勾选，单击“下一步”按钮，如图 1-5 所示。

进入安装选择功能，依据自己的需求进行选择，建议对需求不明确时全部选择，然后单击“安装”按钮，出现图 1-6 所示的安装进度条。

和前面其他版本 Visual Studio 最大的区别就在这里，Visual Studio 2012 进入安装界面后有漫长的等待，直到图 1-7 所示的安装完成界面。

图 1-5　VS 2012 安装项目界面

图 1-6　VS 2012 安装进度条

图 1-7　VS 安装完成提示界面

安装成功，就可以启动了。可以直接从安装结束界面上选择启动，也可以进入“开始”菜单选择 Visual Studio 2012 来进行启动。无论哪种方式，首次启动后都会要求进行注册激活。激活界面如图 1-8 所示。

图 1-8　VS 2012 激活界面

输入正确的产品密钥后，单击“下一步”按钮完成注册激活，就可以使用了。单击激活界面中的“启动”或“所有程序”菜单中的 VS 2012 后都可以打开 VS 2012IDE。从 Visual Studio 2003 开始，首次进入开发环境前都会出现如图 1-9 所示的对话框，提示用户选择默认的开发环境，一旦选择后，后面再使用时都会默认使用该开发环境。

图 1-9　VS 2012 默认语言环境设置

这里选择“Visual C#开发设置”，然后单击“启动 Visual Studio”进入开发工具 IDE 环境。至此，WPF 的开发环境就搭建完毕了。

【任务小结】

1．本任务介绍了 WPF 的概念、特点和相关名词术语。学习者应该对即将学习的对象有个清晰正确的认识。

2．本任务介绍了 WPF 的开发工具及其获得和安装的过程。学习者需要结合自己的条件进行选择和部署。

任务 1.2　设计简单 XAML 程序

【任务描述】

前面已经介绍了 WPF 中创造性的技术内容 XAML，正是因为它的存在使得 WPF 程序的设计和开发可以分离，让不同的设计者能分工合作，最大限度地发挥特长，创造出美轮美奂的 WPF 应用程序。本任务中将引领学习者了解并熟悉 XAML，掌握其工作原理和基本的语法特点，并利用所学的 XAML 知识在 Visual Studio 2012 中完成第一个 WPF 程序——“Hello World!”。

【知识准备】

1.2.1　XAML 是什么

XAML（Extensible Application Markup Language）即可扩展应用程序标记语言，它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。我们可以在声明性 XAML 标记中创建可见的 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。这与大多数其他标记语言不同，后者通常是与后备类型系统没有此类直接关系的解释语言。XAML 实现了一个工作流，通过此工作流，各方可以采用不同的工具来处理应用程序的 UI 和逻辑。

尽管 XAML 是一种可以应用于不同问题领域的技术，但主要用于构造 WPF 用户界面。简单地说，XAML 文档定义了在 WPF 应用程序中组成窗口的面板、按钮以及各种控件的布局。对图形设计人员来说，不会手动编写 XAML，而应该使用图形设计工具 Blend；而对开发人员来说使用的则是 Visual Studio。尽管这两个工具完全不同，但是它们在生成 XAML 时本质上是相同的，所以软件设计团队可以使用 Visual Studio 创建一个基本用户界面，然后将该界面移交给专业的界面设计团队，让他们使用 Blend 来美化该界面。实际上具备将设计人员和开发人员工作流程集成起来的能力是微软力推 XAML 的主要原因之一。

以文本表示时，XAML 文件是通常具有.xaml 扩展名的 XML 文件。可通过任何 XML 编码对文件进行编码，但通常编码为 UTF-8。

下面的示例演示如何创建作为 UI 一部分的按钮。此示例的目的仅在于供我们初步了解 XAML 是如何表示常用 UI 编程形式的（它不是一个完整的示例）。

```
<StackPanel>
  <Button Content="Click Me"/>
</StackPanel>
```

XAML 是一种由 XML 派生而来的语言，所以很多 XML 中的概念在 XAML 中是通用的。比如使用标签声明一个元素（每个元素对应内存中的一个对象）时，需要使用起始标签<Tag>和终止标签</Tag>，夹在两者之间的 XAML 代码表示是隶属于这个标签的内容。如果没有下级隶属元素（对象），则这个称为空标签，可以写为<Tag/>（自结束标记）。

尽管 XAML 是微软专门用来制作 UI 的工具，但是在 WPF 中它并不是唯一的 UI 实现方法。如图 1-10 所示，在 WPF 中 XAML 和 C#都能通过不同的方法实现 UI。

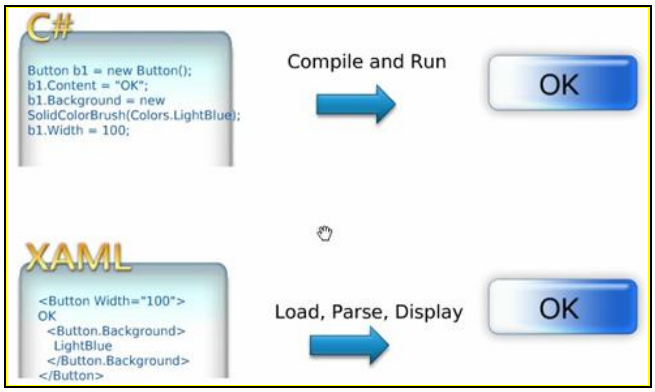


图 1-10　XAML 和 C#各自的 UI 实现

注意：XAML 对于 WPF 不是必需的，理解这一点很重要。Visual Studio 也可以使用 Windows 窗体方法，通过语句代码来构造 WPF 窗口。但是如果这样的话，窗口将被限制在 Visual Studio 开发环境之内，并且只能由开发人员使用。

1.2.2　XAML 语法基础

XAML 是一种由 XML 派生而来的语言，所以很多 XML 中的概念在 XAML 中是通用的。除了 XML 的特性外，XAML 更是具备大量的专有特性。

1. 命名空间

XAML 命名空间的概念和 C#代码中的 Using、VB.NET 代码中的 Import 类似，它为对象元素的实例化提供引用类库声明，避免使用冗长的完全限定名。

编程框架能够区分用户声明的标记和框架声明的标记，并通过命名空间限定来消除可能的标记冲突。例如一个新建的 WPF 窗体的定义代码：

```
<Window x:Class="HelloWorld.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WPF 案例" Height="300" Width="300">
    <Grid >
    </Grid>
</Window>
```

命名空间的语法格式如下：

```
xmlns[:可选的命名空间前缀]="名称空间"
```

xmlns 后可以跟一个可选的映射前缀，之间用冒号分隔，如果没有写可选映射前缀，就意味着所有来自这个命名空间的标签都不用加前缀，这个没有映射前缀的命名空间称为“默认命名空间”，默认命名空间只能有一个。默认命名空间无需定义命名空间前缀名，即意味着所有

来自于这个命名空间的标签前都不用加前缀。默认命名空间只能有一个，而且应该选择其中元素被最频繁使用的命名空间来充当。命名空间格式及功能如图 1-11 所示。

图 1-11　命名空间格式

前面的例子中，Window 和 Grid 都属于 xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation 声明的默认命名空间，而 Class 特征来自于 xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml 声明的命名空间，如果给 xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation 声明的命名空间加上一个前缀，例如“n”，那么代码必须修改成这样：

```
<n:Window x:Class=" HelloWorld.MainWindow"
          xmlns:n="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
          Title="MainWindow" Height="300" Width="300">
    <n:Grid>
    </n:Grid>
</n:Window>
```

在 WPF 窗体自动生成的代码中，自动引用进来的两个命名空间非常重要。其中默认命名空间映射的是 http://schemas.microsoft.com/winfx/2006/xaml/presentation，它代表了如下的命名空间：

```
. System.Windows
. System.Windows.Automation
. System.Windows.Controls
. System.Windows.Controls.Primitives
. System.Windows.Data
. System.Windows.Documents
. System.Windows.Forms.Integration
. System.Windows.Ink
. System.Windows.Input
. System.Windows.Media
. System.Windows.Media.Animation
. System.Windows.Media.Effects
. System.Windows.Media.Imaging
. System.Windows.Media.Media3D
. System.Windows.Media.TextFormatting
. System.Windows.Navigation
. System.Windows.Shapes
```

也就是说，在 XAML 中我们可以直接使用这些命名空间中的类型，而不需要使用前缀（完全限定名）。

而 x 命名空间映射的是 http://schemas.microsoft.com/winfx/2006/xaml，它包含的类均与解析 XAML 语言相关，所以也称之为“XAML 命名空间”。

与 C#语言一样，XAML 也有自己的编译器。XAML 语言被解析并编译，最终形成微软中间语言保存在程序集中。在解析和编译 XAML 的过程中，我们经常要告诉编译器一些重要的信息，如 XAML 编译的结果应该和哪个 C#代码编译的结果合并、使用 XAML 声明的元素是 public 还是 private 访问级别等。这些让程序员能够与 XAML 编译器沟通的工具就存在 x:命名空间中。它包含的用法非常多，可以分为 Attribute、标签扩展、XAML 指令元素三个种类，每个类别都有一些特别的用法。

下面对其中常用的几种形式进行介绍。

（1）x:Key。

在 XAML 文件中，我们可以把需要多次使用的内容提取出来放在 Resource Dictionary（资源字典）中，需要使用的时候就用这个资源的 Key 将这个资源检索出来。

在 WPF 中，几乎每个元素都有自己的 Resource 属性，这个属性就是 key-value 的集合。只要把元素放进这个集合里，这个元素就成了资源字典中的一个条目。当然，为了能检索到这个条件，就必须为它添加 x:Key。x:Key 的作用就是为资源贴上用于检索的索引，它为资源字典中的每个资源设置一个唯一键。注意，在带有 x:Key 的 XAML 中指定的值总是被作为字符串处理的，除非使用标记扩展，否则它不会尝试使用类型转换。

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Color x:Key="1" A="255" R="255" G="255" B="255"/>
    <Color x:Key="2" A="0" R="0" G="0" B="0"/>
</ResourceDictionary>
```

（2）x:Class。

这个 Attribute 是告诉 XAML 编译器将 XAML 编译器编译的结果和后台编译结果的哪一个类进行合并。以窗体类为例，XAML 中：

```
<Window x:Class=" HelloWorld.MainWindow"…>
</Window>
```

而在 CS 代码中：

```
namespace HelloWorld
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

同样的类名 MainWindow 前使用了 partial 关键字，这样由 XAML 解析成的类和 C#代码文件里定义的部分就合二为一了。正是这种机制我们才可以把类的逻辑代码留在.CS 文件里，用 C#语言来实现，而把那些与声明及布局 UI 元素相关的代码分离出去，实现 UI 与逻辑分离。而后者使用 XAML 和 XAML 编辑工具就能轻松搞定。

（3）x:Name。

它表示处理 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。在实际项目中，控件元素和资源的命名规则是只在需要的时候对控件和资源进行命名操作，绝大多数 XAML 元素都没有明确命名，因为它们不需要被其他代码所访问，其存在仅仅是提供静态作用。这样的好处有以下几点：

- 减小 XAML 文件或应用程序尺寸，加快 InitializeComponent 初始化调用速度。
- 易于项目维护。

但是对于需要被代码访问的控件元素或者资源就需要明确命名，其使用形式如下：

```
<object x:Name="XAMLNameValue".../>
```

例如没有被命名的标签：

```
<Label Content="Hello World!" HorizontalAlignment="Left" VerticalAlignment="Top" RenderTransform-
Origin="6.542,8.3" Margin="164,71,0,0"/>
```

被明确命名的标签：

```
<Label x:Name="lbTips" Content="Hello World!" HorizontalAlignment="Left" VerticalAlignment="Top"
RenderTransformOrigin="6.542,8.3" Margin="164,71,0,0"/>
```

每个已定义的 x:Name 在一个 XAML 名称范围中必须是唯一的。通常，XAML 名称范围在所加载页面的根元素级别定义，其中包含单个 XAML 页面中该元素下面的所有元素。

（4）x:Type。

一般情况下，我们在编程中操作数据类型的实例或者实例的引用，但有的时候我们也需要用到数据类型本身。

x:Type 顾名思义应该是一个数据类型的名称。当想在 XAML 中表达某一数据类型时就需要用到 x:Type 标记扩展。比如某个类的一个属性，它的值要求的是一个数据类型，当我们在 XAML 中为这个属性赋值时就需要用到 x:Type。如下案例中设置了按钮的通用样式，当前窗体中所有的按钮默认都将自动获得该样式。

```
<Window.Resources>
        <Style x:Key="{x:Type Button}" TargetType="{x:Type Button}">
            <Setter Property="Width" Value="30"></Setter>
            <Setter Property="Background" Value="black"></Setter>
        </Style>
</Window.Resources>
```

（5）x:Null。

在 XAML 里面表示空值就是 x:Null。多数时候我们不需要为属性赋一个 Null 值，但如果一个属性开始就有默认值而我们又不需要这个默认值则需要用到 Null 值了。在 WPF 中，Style 是按照一个特定的审美规格设置控件的各个属性，程序员可以为控件逐个设置 style，也可以指定一个 style 目标控件类型，一旦指定了目标类型，所有的这类控件都将使用这个 style——除非你显式地将某个实例的 Style 设置为 Null。

如 x:Type 案例中对所有按钮设置了通用的样式，如果某个按钮不要该样式，则可以用 x:Null。

```
<Button Content="个性" Height="23"    Style="{x:Null}"/>
```

（6）x:Code。

x:Code 的作用是可以在 XAML 文档中编写后置的 C#后台逻辑代码，当然这样写的问题是破坏了 XAML 和 CS 文件的分工，使得代码不容易维护，不易调试。

WPF 中对它的使用有严格的限制，同时并不建议使用。

2. XAML 对象

因为 XAML 是用来在 UI 上绘制控件的，而控件本身就是面向对象的产物，所以一个 XAML 标签就意味着一个对象。在 XAML 中，对象和对象之间的层次关系要么是并列，要么是包含，全部都体现在标签的关系上。

XAML 中使用开始标记和结束标记将对象实例化为 XML 格式的元素：

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

对象中可以包含其他对象，如：

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

若一个对象中不包含其他对象，可以使用一个自结束标记来声明对象：

```
  <Rectangle/>
```

3. XAML 对象属性

属性是面向对象的术语。在使用面向对象思想编程时，常常需要对客观事务进行抽象，再把抽象出来的结果封装成类，类中用来描述事物状态的成员就是属性。比如对一个车辆对象，就有长度、宽度、高度、重量和速度等数据来描述，这些都是它的属性。

在 XAML 中对于属性有两种表述，针对标签而言时属性叫 Atrribute，针对对象而言时属性叫 Property，它们不是一个层面上的内容。而且标签的 Atrribute 与对象的 Property 也不是完全映射的，往往是一个标签所具有的 Atrribute 多于它所代表的对象的 Property。

在 XAML 中也使用属性对 XAML 元素特征进行描述，并且属性不允许在 XAML 中重复设置多次，但允许在托管代码中改变元素的属性值。

在 XAML 中有多种属性设置方法。

（1）使用属性语法。

只有实例化对象才可以设置实例属性，格式如下：

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

或者

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

每个属性对应一个属性值，属性值类型必须与属性匹配，一个标记中可以设置对象的多个属性，例如：

```
<Canvas Width="150" Height="150" Background="Red">
</Canvas>
```

（2）使用属性元素语法。

某些属性可以使用属性元素语法来设置，格式如下：

```
<object>
    <object.property>
        <!--元素属性值-->
    </object.property>
  </object>
```

富创建是 WPF 的亮点之一，我们可以用 Button 来演示，可以把任意内容放在 Button 里

面，不仅限于文本，例如下面在 Button 中嵌入了一个简单的方形来做一个视频播放器的停止按钮。Button 的 Content 属性是 System.Object 类型的，因此它很容易被设置到 40×40 的 Rectangle 对象。但如何才能在 XAML 中用属性特性语法做相同的事呢？你该为 Content 属性设置哪种字串才能完成 C#中声明的 Rectangle 功能呢？没有这样的字串，但 XAML 提供了一种替代的语法来设置复杂的属性值，即属性元素，如下：

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Button.Content>
    <Rectangle Height="40" Width="40" Fill="Black"/>
</Button.Content>
</Button>
```

Content 属性被设置为一个 XML 元素而不是 XML 特性，Button.Content 中的句点用于区分对象元素（Object Element）与属性元素（Property Element）。它们总会以“类型名.属性名”（TypeName.PropertyName）的形式出现，总会包含在“类型名”对象元素中，但它们没有属于自己的特性。

属性元素语法也可以用于简单的属性值。下面的 Button 使用特性设置了两个属性，它们是 Content 和 Background：

```
<Button   Content="OK"   Background="White"/>
```

（3）使用内容元素语法。

某些元素的属性支持内容元素语法，允许忽略元素的名称，实例对象会根据 XAML 元素中的第一个标记值来设置属性。对于大量的格式化文本，使用内容元素语法更加灵活，属性标记之间可以插入大量的文本内容。

```
<TextBlock Width="200" TextWrapping="Wrap">
Windows 8 是微软即将推出的最新 Windows 操作系统。Windows 8 支持个人电脑（Intel 平台系统）及平面电脑（Intel 平台系统或 ARM 平台系统）。Windows 8 大幅改变以往的操作逻辑，提供更佳的屏幕触控支持。
</TextBlock>
```

（4）使用集合语法。

元素支持一个属性元素的集合时才能使用集合语法进行属性设置。对于这类属性可以使用托管代码的 Add 方法来增加更多的集合元素，它的本质是向对象的集合中添加属性项。

```
<Rectangle Width="200" Height="150">
    <Rectangle.Fill>
        <LinearGradientBrush>
            <GradientStopCollection>
                <GradientStop Offset="0.0" Color="Gold"/>
                <GradientStop Offset="1.0" Color="Green"/>
            </GradientStopCollection>
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

4. XAML 的特殊属性

对 XAML 对象，除了一些常见属性外，还有一些特殊属性。

（1）附加属性。

附加属性作用于支持附加属性的元素，它是由支持附加属性的父元素产生作用，支持附加属性的元素会继承所在的父元素的属性。

附加属性的格式：AttachedPropertyProvider.PropertyName。

例如：

```
<Canvas>
<Rectangle  Canvas.Left="50" Canvas.Top="50"  Width="200"  Height="150" RadiusX="10"
RadiusY="10"  Fill="Gold"/>
</Canvas>
```

（2）依赖属性。

依赖属性和 CLR 属性类似，提供一个实例级私有字段的访问封装，通过 GetValue 和 SetValue 访问器实现属性的读写操作。最重要的一个特点是属性值依赖于一个或者多个数据源，提供这些数据源的方式也可以不同。由于依赖多数据源的缘故，因此称之为依赖属性。

依赖属性可以通过多种不同类型的数据源进行赋值，不同赋值顺序影响属性值的改变，其优先级如图 1-12 所示。

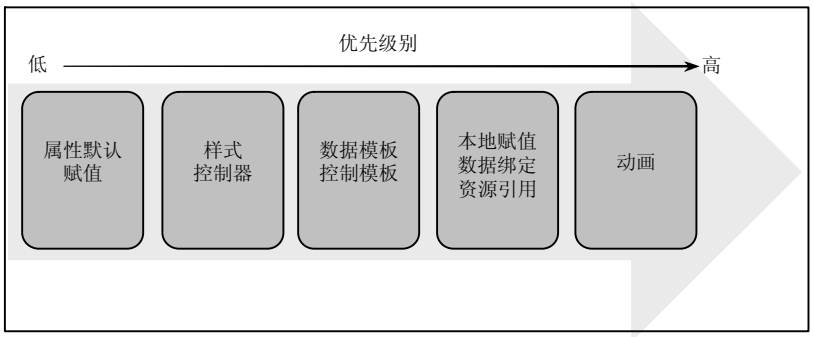


图 1-12　属性依赖优先级

例如：

```
<Page.Resources>
<Style x:Key="ButtonStyle" TargetType="Button">
   <Setter Property="Foreground" Value="Red"/>
   <Setter Property="FontSize" Value="24"/>
</Style>
</Page.Resources>

<Button  Content="依赖属性测试"  Style="{StaticResource  ButtonStyle}"  Width="240"/>
```

在案例中按钮使用已经定义好的样式，即按钮的文字前景色为红色，字号为 24 号。

```
<Button Content="依赖属性测试"  Style="{StaticResource  ButtonStyle}"  Width="240" Foreground
="Yellow"  FontFamily="14"/>
```

在该案例中根据依赖属性优先级，本地属性的赋值具有更高优先级，因此其样式属性赋值就没有效果了，按钮的文字前景色为黄色，字号是 14 号。

5. XAML 在 Windows 8 Metro 中的新特性

微软 Windows 8 系统不仅适用于 PC，而且适用于平板电脑。针对平板电脑的特性，XAML 也增加了新的事件处理特性，主要体现在：

（1）继承传统事件处理机制，XAML 将控制按钮单击事件。

（2）监听列表控件选项事件。

（3）监听应用激活和暂停事件。

（4）触控事件处理，包括指针处理、手势处理和控制操作事件等。

常用触控事件列表

Pointers	Gestures	Manipulation
Pressed	Tapped	Starting
Released	RightTapped	Started
Moved	DoubleTapped	Delta
Canceled	Holding	Completed
CaptureLost		InteriaStarting
Entered		
Exited		

1.2.3　WPF 中的树

在许多技术中，元素和组件都按树结构的形式组织，在这种结构中，开发人员可以直接操作树中的对象节点来影响应用程序的呈现或行为。WPF 也使用了若干树结构形式来定义程序元素之间的关系。多数情况下，WPF 开发人员可以用代码创建应用程序，也可以用 XAML 定义应用程序的组成部分。与此同时，他们在概念层面上考虑对象树形式，但却要调用具体的 API 或使用特定的标记来实现对象树，而不是像在 XML DOM 中那样，使用某些常规对象树操作 API。WPF 公开两个提供树形式视图的帮助器类：LogicalTreeHelper（逻辑树）和 VisualTreeHelper（可视化树），它们有助于理解某些关键 WPF 功能的行为。

1. 逻辑树

XAML 天生就是用来呈现用户界面的，这是由于它具有层次化的特性。在 WPF 中，用户界面由一个对象树构建而成，这棵树叫做逻辑树。逻辑树始终存在于 WPF 的 UI 中，不管 UI 是用 XAML 编写还是用代码编写。WPF 的每个方面（属性、事件、资源等）都是依赖于逻辑树的。WPF 逻辑树的理解对深入开发至关重要。下面就把逻辑树这个概念比较容易被误解的地方加以说明：

（1）WPF 逻辑树并不只存在于使用 AML 构建的对象中，使用程序代码构建的对象同样存在逻辑树。XAML 是专门用于 WPF 编程的新 API，就像当初 C#是专门用于 .NET 开发的 API 一样。因此，只要“界面对象”被创建出来，我们就可以为其绘制逻辑树。

（2）WPF 逻辑树描述的是“界面对象”的构建过程，而不是“界面对象”的结构。逻辑树是由“界面对象”及其所包含的对象共同构成的，这些被包含的对象是在创建“界面对象”时被添加到该“界面对象”的。

（3）WPF 逻辑树是由“界面对象”及其“内容属性”构成的，他们之间是树结构中的“父节点”与“子节点”。“子节点”还可以继续展开直至“子节点”不再包含“内容属性”，那么就可以说这个“子节点”是逻辑树中的一个“叶子节点”。

（4）没有值的“内容属性”不会出现在逻辑树中，只有具有属性值的“内容属性”才是逻辑树的一个节点。

（5）在逻辑树中，看不到来自所应用模板的可视化对象。

2. 可视化树

可视树可以说是逻辑树的扩展，它是把逻辑树中的每个节点全部打散，然后放到可视组

件中，这样形成的一棵树叫做可视树。在可视树中出现的节点有些在逻辑树中是没有的。逻辑树的节点对我们而言基本是一个黑盒。而可视树不同，它暴露了视觉的实现细节。并不是所有的逻辑树节点都可以扩展为可视树节点。只有从 System.Windows.Media.Visual 和 System.Windows.Media.Visual3D 继承的元素才能被可视树包含。其他的元素不能包含是因为它们本身没有自己的提交（Rendering）行为。

3．两种树的观察和比较

例如有如下 XAML：

```
<Window>
    <Grid>
        <Label Content="Label" />
        <Button Content="Button" />
    </Grid>
</Window>
```

则它们的整个树结构如图 1-13 所示。

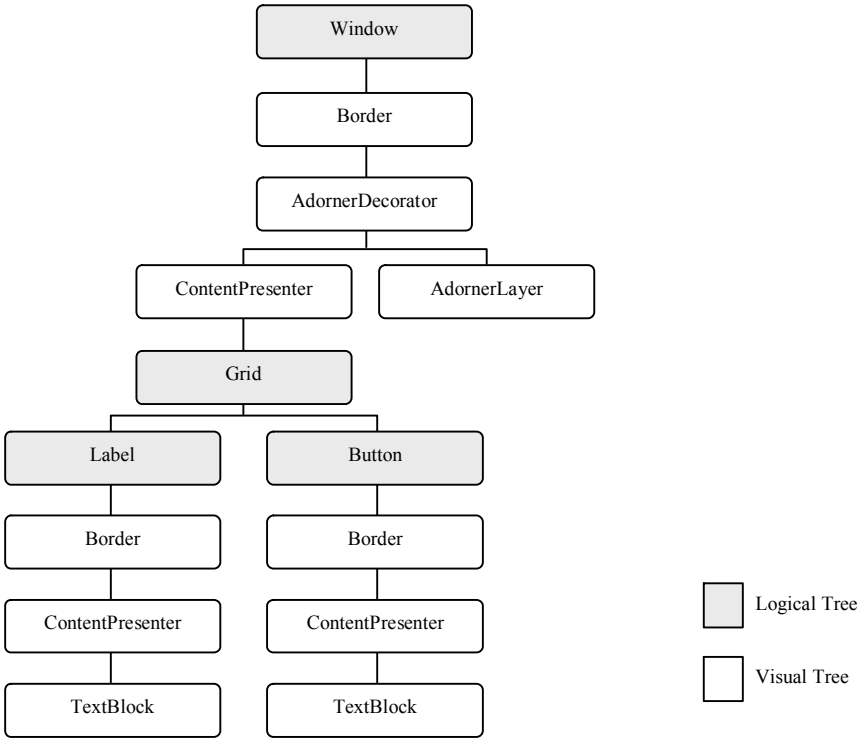


图 1-13　WPF 窗体的逻辑树与可视化树

实际应用中可以借助 WPF Inspector 工具（如图 1-14 所示）进行两种树结构的查看。

例如对如下 XAML：

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Button Content="Hello World" Width="200" Height="100"/>
</Window>
```

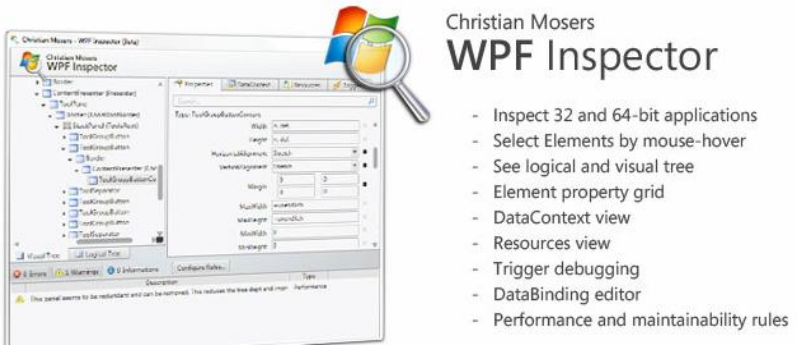
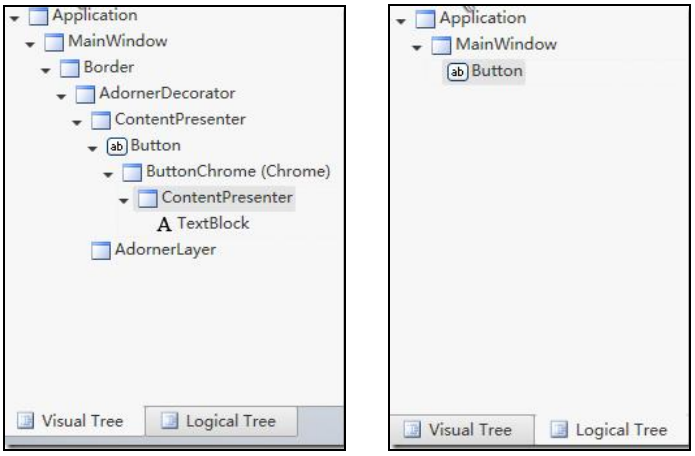


图 1-14　WPF Inspector 工具

运行结果如图 1-15 所示。

图 1-15　案例运行效果

使用 WPF Inspector 工具观测结果如图 1-16 所示。



可视化树　　逻辑树

图 1-16　观测结果

我们看出以下特性：

（1）WPF 启动程序的根元素均为 Application。

（2）逻辑树与 XAML 的布局结构是相同的。

（3）可视化树是根据控件的模板来呈现的，我们很难猜测可视化树的结构，因为控件还可以自定义模板。

【任务分析】

完成前面的知识准备，我们现在来对“Hello World!”程序任务进行分析。

要实现 Hello World 信息的展示，结合 XAML 知识，可以在窗体上放入标签的 XAML，并调整其属性，让其呈现在窗体中间。

对于标签来说，要实现居中显示，可以有两个思路：一个是针对固定大小窗体，主要是对位置的相关属性赋值；另一个是对大小变化的窗体，主要是对标签在窗体中的相对位置赋值。后者具有更高的灵活性和自适应性，我们选择这个方式实现。

要实现对其属性的调整可以有多种方式：

- 使用属性语法直接在 XAML 属性上设定值。
- 使用属性元素语法嵌入属性的设置。

对于位置设定这样的简单属性推荐使用属性语法。

【任务实施】

1. 新建 WPF 项目

打开 Visual Studio 2012，单击“文件”→“新建”→“项目”命令，如图 1-17 所示。

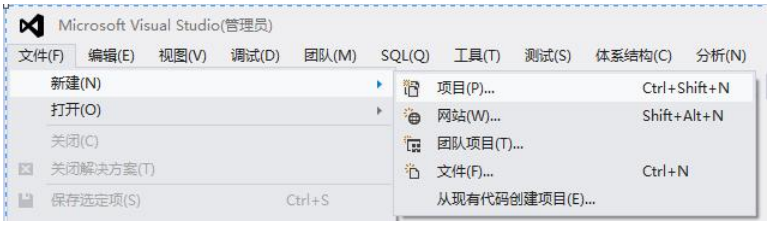


图 1-17　新建项目

弹出如图 1-18 所示的“新建项目”对话框。

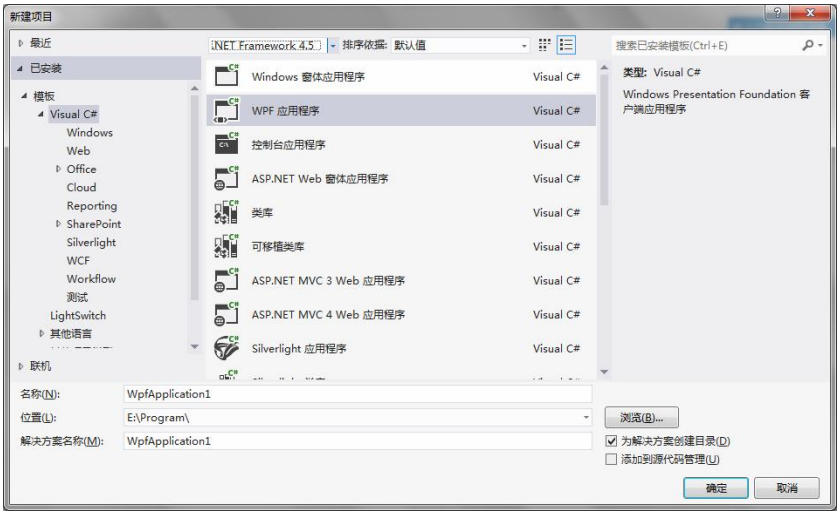


图 1-18　“新建项目”对话框

在该对话框中，左边默认选中 Visual C#，右边选择“WPF 应用程序”。如果栏目中没有“WPF 应用程序”项目，则可能是上面的.NET Framework 版本过低，只有选择 3.0 或更高版本的才有。同样道理也可以为 WPF 程序设定发布的目标.NET Framework 版本。

输入好名称，选择好保存位置，单击“确定”按钮，一个 WPF 程序就创建好了。

2. 制作“Hello World!”显示

因为要显示到窗体中央，常见方法有以下两种：

（1）将标签控件坐标修改为中央。

直接在窗体中加入标签的 XAML，并修改其属性，将其呈现到窗体中央。窗体布局效果如图 1-19 所示。

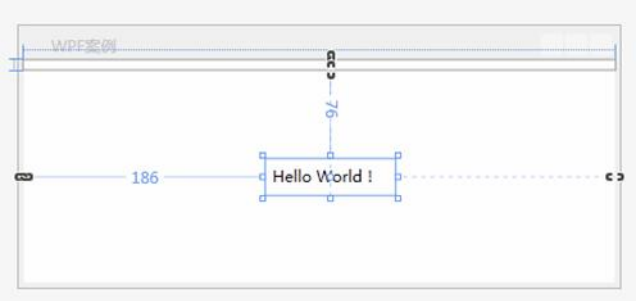


图 1-19　窗体控件直接定位布局图

窗体 XAML 如下：

```
<Window x:Class="HelloWorld.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WPF 案例" Height="200" Width="462">
    <Grid>
        <Label Content="Hello World！" Margin="186,76 " Height="28" Width="99"/>
    </Grid>
</Window>
```

这种方式制作简单，但是属于传统 Windows 程序制作思维，只适合于窗体控件很少的时候。

（2）对窗体 Grid 进行分割布局，然后让标签位置选择中心布局单元格。

选中窗体后，鼠标移动到窗体的上下左右边都会出现黄色小箭头，单击即可产生一个窗体布局切割线，已经产生的切割线可以删除，也可以拖放来重新定位，将窗体切割成奇数行和奇数列，让中心单元格就呈现在中心位置。然后添加标签 XAML，配置其基本属性和布局位置属性。

窗体布局效果图如图 1-20 所示。

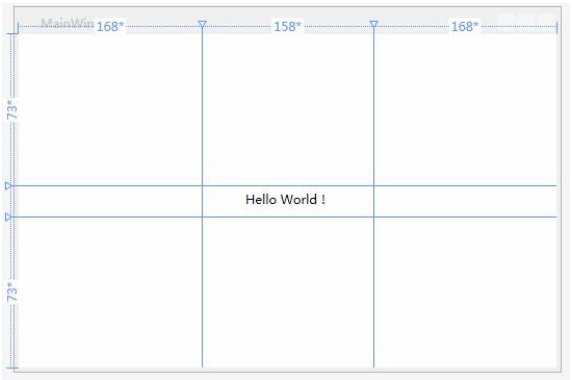


图 1-20　使用 Grid 布局方式效果

窗体 XAML 如下：

```
<Window x:Class=" HelloWorld.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">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="73*"/>
            <RowDefinition Height="15*"/>
            <RowDefinition Height="72*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="168*"/>
            <ColumnDefinition Width="158*"/>
            <ColumnDefinition Width="168*"/>
        </Grid.ColumnDefinitions>
        <Label Content="Hello World！" HorizontalAlignment="Center" VerticalAlignment="Top"
            Grid.Column="1" Grid.Row="1"/>
    </Grid>
</Window>
```

在后面的程序设计中，我们都推荐大家使用窗体 Grid 进行布局（详细布局方式见项目二），该方式布局程序具有一定的自适应性，容易理解接受，精心设计后能够达到类似网页般自由放缩的效果。

无论使用哪种方式制作的程序，制作完毕后，单击工具栏中的“启动”按钮（或者按快捷键 F5），都可看到图 1-1 所示的效果。

【任务小结】

1．本任务介绍了 XAML 的特点、特性、属性和事件。通过这些介绍大家需要体会为什么 WPF 可以做到 UI 和代码设计分离，甚至不同的工具做的事情还可以整合。

2．本任务演示了最简单的 WPF 程序的创建和制作过程。尽管要求简单，但也提供了两种思路来解决，给初学者一些参考。没有固定的方法，条条道路通罗马。

本项目通过两个任务分别对 WPF 的常识和 XAML 基础进行了介绍，一方面让学习者循序渐进地理清了 WPF 是什么、有哪些特点、如何开发、如何分工合作等情况，又对 WPF 的新推技术 XAML 有了初步掌握，让学习者对 WPF 的运行模式有了更多体会；另一方面又通过任务的布置、知识学习、任务分析和任务实施培训了学习者的项目实施方法和能力。

1．在个人计算机或虚拟机上完成 Visual Studio 2012 版的安装。

2．了解 Visual Studio 2012 集成开发环境，完成简单“HelloWorld!”程序的项目创建、编写、调试。

3．查询资料，学习使用 Blend for Visual Studio 2012，完成一些简单的设计。