信息科学与技术丛书 程序设计系

张晗雨 编著

WPF 全视角分析

- 软件界面的基本组成——控件
- 控制控件的位置——布局与变换
- 数据与界面的联动——数据绑定
- 外观控制——模板、样式、皮肤、主题
- 定制外观——2D 图像支持
- 属性驱动的基础——属性系统

机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书
程序设计系列

WPF 全视角分析

张晗雨　编著

机械工业出版社

这是一本讲解 WPF（Windows Presentation Foundation）的使用、架构、实现逻辑的书。

本书按照 WPF 的各种功能由浅入深地进行讲解。不同于众多外文书籍的是：在本书的阅读过程中，读者将看到 WPF 如何组织其中包含的各个类，类层次结构中各个类所提供的功能以及各种功能的实际内部实现等多方面的内容。阅读本书后，读者能够真正掌握 WPF 各种功能的使用方法。

希望读者能够通过本书的学习清晰地认识到 WPF 所提供的各种功能之间的联系，并在此基础上熟练、灵活地掌握这些功能的用法。

书中的代码可在 http://www.cmpbook.com 下载。

图书在版编目（CIP）数据

WPF 全视角分析/张晗雨编著．—北京：机械工业出版社，2008.12
（信息科学与技术丛书·程序设计系列）
ISBN 978-7-111-25785-1

Ⅰ．W…　Ⅱ．张…　Ⅲ．主页制作－程序设计　Ⅳ．TP393.092

中国版本图书馆 CIP 数据核字（2008）第 200876 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：车　忱
责任编辑：车　忱
责任印制：洪汉军

北京振兴源印务有限公司印刷厂印刷

2009 年 1 月·第 1 版第 1 次印刷
184mm×260mm·30 印张·743 千字
0001－4000 册
标准书号：ISBN 978-7-111-25785-1
定价：50.00 元

凡购本图书，如有缺页、倒页、脱页，由本社发行部调换
销售服务热线电话：（010）68326294　68993821
购书热线电话：（010）88379639　88379641　88379643
编辑热线电话：（010）88379753　88379739
封面无防伪标均为盗版

出版说明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术、新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书，来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新型技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术受到不断挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机在社会生活中日益普及，随着因特网延伸到人类世界的层层面面，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用，所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们对了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络、工程应用等内容，注重理论与实践相结合，内容实用，层次分明，语言流畅，是信息科学与技术领域专业人员不可或缺的图书。

现今，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设作出贡献。

机械工业出版社

前 言

我在2006年10月去微软亚洲工程院上海分院面试的时候，考官问我曾经参与过的最喜欢的项目是什么。由于当时参与的项目涉及公司机密，因此我说出当时正在开发的一个个人项目。在该项目中，我试图用XML语言加关键词分析的方法描述软件界面，同时以脚本语言为基础构建各种控件的触发逻辑，并用自己编写的分析引擎对软件界面进行分析和构建。考官听了我的介绍，只是向我笑着推荐了WPF（Windows Presentation Foundation）。当我接触WPF后，我立即放弃了原项目，投入到对WPF的研究中。

在学习过程中，我一直在阅读与该技术有关的外文书籍。但是由于外文书籍更强调应用而非逻辑上的联系，因此我在对该类库的研究过程中常常有一种无法理清脉络的感觉。这种感觉困扰我很久。有感于市场上这方面书籍的匮乏，我在2007年6月决定编写一本适合国人阅读的书籍。

经过一年的编写、六次修订稿件，我终于完成了对该书的编写。

本书的组织

本书是按照由浅入深的方式组织知识的讲解和配套代码的。前十二章致力于对WPF的使用进行讲解，并在讲解过程中适当插入对实现本质的说明。后四章叙述对该技术的理解方法，主要将WPF的各部分功能进行串联，使读者在阅读了这四章后对该技术有深入的了解。

本书适合的读者

本书在讲解WPF各个功能时，致力于对其内部实现逻辑以及功能之间的联系的挖掘。因此无论是刚接触该技术的新人还是较熟悉该技术的人，阅读本书都将对其理解和应用WPF有所帮助。

代码样例

本书的各个章节基本都提供了帮助理解的代码样例，这些样例用“代码AppX.X”标明。为了便于刚接触这门技术的读者学习，在前面几章中，本书尽可能将更多的样例添加到与书相对应的代码中，以方便读者调试和理解。

本书的源代码一共提供了两个版本，分别对应支持WPF的两种Visual Studio版本：Visual Studio 2005和Visual Studio 2008。读者可在http://www.cmpbook.com下载代码。

张晗雨

试读结束：需要全本请在线购买： www.ertongbook.com

目　录

出版说明
前言
第 1 章　初识 WPF …… 1
1.1　GDI 与 WPF …… 1
1.2　WPF 架构 …… 5
1.3　WPF 的特点 …… 7
第 2 章　界面声明脚本——XAML …… 9
2.1　环境设置 …… 9
2.2　通过 Visual Studio 创建工程 …… 10
2.3　XML 简介 …… 11
2.4　XAML 简介 …… 12
2.4.1　XAML 是 WPF 中的声明式语言 …… 13
2.4.2　XAML 对名字空间的使用 …… 15
2.4.3　XAML 中的各个关键字 …… 20
2.4.4　XAML 对属性的设置 …… 26
2.4.5　关联属性和附加属性简介 …… 28
2.4.6　XAML 扩展标记 …… 29
第 3 章　软件界面的基本组成——控件 …… 31
3.1　控件类的派生结构 …… 35
3.1.1　基类——Object 类 …… 35
3.1.2　界面元素的单线程约束——DispatcherObject 类 …… 35
3.1.3　参与属性系统——DependencyObject 类 …… 36
3.1.4　界面外观组成——Visual 类 …… 38
3.1.5　界面元素的基本实现——UIElement 类 …… 39
3.1.6　界面元素的具体实现——FrameworkElement 类 …… 45
3.1.7　控件类基类——Control 类 …… 51
3.2　WPF 的内建控件 …… 53
3.2.1　控件组合基础——单条目控件 …… 53
3.2.2　项目集合的显示——多条目控件 …… 71
3.2.3　小空间显示大元素——范围控件 …… 85
3.2.4　遗漏了什么重要控件吗——其他控件 …… 87
第 4 章　控制控件的位置——布局与变换 …… 97
4.1　位置控制属性 …… 97
4.1.1　控件该有多大——尺寸控制属性 …… 97
4.1.2　应该占据什么位置呢——对齐控制属性 …… 104

4.1.3 让界面元素变形——使用变换 …… 108
4.1.4 设置冲突怎么办——布局属性优先级 …… 114
4.2 使用面板控制布局 …… 115
4.2.1 经典布局方式——Canvas …… 116
4.2.2 单向排列面板——StackPanel …… 117
4.2.3 逐行显示面板——WrapPanel …… 118
4.2.4 类网页布局面板——DockPanel …… 120
4.2.5 这种面板就是添格子——Grid …… 121
4.2.6 选择最合适的布局——五种面板的比较 …… 126
4.2.7 还有别的么——其他布局方式 …… 127
4.2.8 它们也能控制布局——具有布局功能的类 …… 129
4.2.9 不能完全显示怎么办——溢出处理 …… 132
第5章 用规律替代个体——使用资源 …… 137
5.1 二进制资源 …… 137
5.1.1 二进制资源的添加 …… 138
5.1.2 二进制资源的访问 …… 139
5.1.3 全球化和本地化操作 …… 143
5.2 逻辑资源 …… 149
5.2.1 在 WPF 工程里添加资源 …… 149
5.2.2 访问资源 …… 152
5.2.3 使用动态资源 …… 155
第6章 数据与界面的联动——数据绑定 …… 158
6.1 数据绑定简介 …… 158
6.1.1 数据绑定的代表——Binding 类 …… 158
6.1.2 在后台操作绑定——BindingOperation 类 …… 159
6.2 在 XAML 中使用数据绑定 …… 160
6.2.1 最简单的情况——绑定关联属性 …… 160
6.2.2 让绑定的使用更普遍——绑定普通属性 …… 163
6.2.3 并不常见的情况——绑定整个元素 …… 165
6.2.4 处理大量数据——绑定数据集合 …… 165
6.2.5 更灵活的数据记录方法——Data Providers …… 168
6.3 高级话题 …… 174
6.3.1 到底是谁更新谁——绑定方式 …… 174
6.3.2 数据更新的时机——数据更新规则 …… 175
6.3.3 绑定中的复杂逻辑——绑定多个数据源 …… 177
第7章 外观控制——模板、样式、皮肤、主题 …… 180
7.1 基础知识 …… 180
7.1.1 界面声明中的名字空间——元素名称作用范围 …… 180
7.1.2 简单逻辑的表示——触发器 …… 181

7.2 模板 …… 187
7.2.1 模板的抽象——FrameworkTemplate 类 …… 188
7.2.2 数据外观定义——数据模板 …… 188
7.2.3 控件外观定义——控件模板 …… 191
7.2.4 多条目控件的特殊模板——ItemsPanelTemplate …… 194
7.3 样式 …… 194
7.3.1 从继承结构的分析开始——样式的简单使用 …… 196
7.3.2 样式也可以派生——Style 类的继承与覆盖 …… 198
7.3.3 样式的自动使用——设置和获得控件的默认样式 …… 198
7.3.4 样式、模板和触发器之间的联系 …… 200
7.4 皮肤 …… 202
7.5 主题 …… 206
第 8 章 定制外观——2D 图像支持 …… 209
8.1 WPF 中的图像容器——Image 类 …… 209
8.2 轻量级图像元素——Drawing 类 …… 210
8.2.1 图形类基类——GeometryDrawing …… 214
8.2.2 简单的 Geometry 类派生类 …… 215
8.2.3 非常用图形的表示——PathGeometry 类 …… 215
8.2.4 多个图形的合并——GeometryGroup …… 218
8.2.5 图形的计算——CombinedGeometry …… 220
8.2.6 简化图形表示——StreamGeometry …… 221
8.3 操作底层实现进行绘制——Visual 类 …… 222
8.4 高级图像类——Shape …… 226
8.4.1 线的表示——Line …… 227
8.4.2 折线的表示——Polyline …… 228
8.4.3 多边形的表示——Polygon …… 229
8.4.4 矩形的表示——Rectangle …… 229
8.4.5 椭圆形的表示——Ellipse …… 230
8.4.6 通用图形表示类——Path …… 230
8.5 2D 图形相关知识 …… 230
8.5.1 颜色的表示——Color …… 230
8.5.2 透明度控制 …… 231
8.5.3 点击测试 …… 232
8.5.4 画刷的表示——Brush …… 235
8.5.5 画笔的表示——Pen …… 243
8.5.6 2D 特效——BitmapEffect …… 245
第 9 章 创建真实的世界——WPF 的 3D 支持 …… 249
9.1 3D 基础知识 …… 249
9.1.1 位置的表示——坐标系 …… 249

9.1.2 相对位置的表示——模型坐标 …… 251
9.1.3 观察者的表示——相机 …… 252
9.1.4 只显示能看到的——裁剪 …… 253
9.2 使用 WPF 创建最简单的 3D 程序 …… 253
9.2.1 3D 程序示例 …… 253
9.2.2 程序中的观察者——相机类 …… 256
9.2.3 确定显示效果——材质类 …… 258
9.2.4 虚拟世界中的光——光源类 …… 259
9.3 3D 变换 …… 261
9.3.1 位移变换——TranslateTransform3D …… 261
9.3.2 缩放变换——ScaleTransform3D …… 261
9.3.3 旋转变换——RotateTransform3D …… 262
9.3.4 组合变换——Transform3DGroup …… 262
9.4 3D 物体表示 …… 262
9.5 3D 物体的绘制 …… 265
第 10 章 软件界面中的多媒体——动画和音乐 …… 268
10.1 WPF 中的动画 …… 268
10.1.1 基础知识 …… 269
10.1.2 在 XAML 中使用动画类 …… 277
10.1.3 动画类的使用及示例 …… 288
10.2 WPF 中的音频和视频 …… 292
10.2.1 SoundPlayer 类 …… 293
10.2.2 MediaPlayer 类 …… 293
第 11 章 文字信息的展示——WPF 的文本功能 …… 297
11.1 流文本显示类——FlowDocument …… 297
11.1.1 FlowDocument 类的继承结构 …… 298
11.1.2 流文本可以使用的各个元素 …… 302
11.1.3 流文本的显示 …… 313
11.2 WPF 打印功能简介 …… 317
11.2.1 打印系统简介 …… 318
11.2.2 使用 WPF 打印功能 …… 319
11.2.3 打印机管理 …… 321
第 12 章 完善 WPF 项目——应用相关 …… 323
12.1 WPF 项目的组成 …… 323
12.1.1 WPF 项目的创建 …… 323
12.1.2 WPF 工程的种类以及项目文件组成 …… 325
12.1.3 Application 类 …… 329
12.1.4 Environment 类 …… 333
12.1.5 WindowsFormsApplicationBase 类 …… 334

12.1.6 ApplicationSettingsBase 类 …… 336
12.1.7 Window 类 …… 338
12.1.8 NavigationWindow 类及其相关类 …… 347
12.2 WPF 中的对话框 …… 354
12.2.1 通用对话框 …… 354
12.2.2 消息框 …… 355
12.2.3 TaskDialog …… 356
第 13 章 思考——深入了解 WPF …… 364
13.1 WPF 的架构 …… 364
13.1.1 界面元素系统 …… 365
13.1.2 视觉系统 …… 368
13.1.3 文字系统 …… 369
13.1.4 输入系统 …… 369
13.1.5 属性系统 …… 370
13.1.6 消息交换层 …… 371
13.1.7 功能实现部分 …… 371
13.2 WPF 思想 …… 371
13.2.1 XAML …… 371
13.2.2 变换 …… 373
13.2.3 布局 …… 374
13.2.4 数据绑定 …… 375
13.2.5 模板及样式 …… 376
13.2.6 WPF 中的图像 …… 377
13.2.7 动画 …… 378
13.3 WPF 中的性能问题 …… 378
13.3.1 从软件整体考虑性能 …… 378
13.3.2 使用硬件加速 …… 379
13.3.3 合理使用图形图像 …… 380
13.3.4 动画中的性能考虑 …… 384
13.3.5 使用最合适的布局 …… 384
13.3.6 绑定中的性能问题 …… 385
13.3.7 合理使用资源 …… 386
13.3.8 类实现相关 …… 386
第 14 章 属性驱动的基础——属性系统 …… 388
14.1 关联属性实现 …… 388
14.1.1 CLR 属性接口 …… 388
14.1.2 关联属性 …… 389
14.1.3 关联属性的实现 …… 390
14.2 元数据 …… 392

14.2.1 元数据简介 …… 393
14.2.2 默认值与属性继承 …… 395
14.2.3 属性更改及刷新的回调函数 …… 396
14.2.4 FrameworkPropertyMetadata …… 397
14.2.5 对元数据进行更改 …… 397
14.3 附加属性 …… 402
14.4 只读属性 …… 407
14.5 集合类型的关联属性实现 …… 408
14.6 属性优先级 …… 410
第 15 章 WPF 中的交互手段——了解 WPF 事件内部机制 …… 411
15.1 WPF 中的线程模型 …… 411
15.1.1 消息循环和 DispatcherObject …… 412
15.1.2 Dispatcher …… 413
15.2 WPF 中的事件 …… 417
15.2.1 路由事件 …… 417
15.2.2 附加事件 …… 420
15.2.3 自定义事件的实现 …… 421
15.2.4 RoutedEventArgs …… 424
15.3 WPF 对命令的支持 …… 426
15.3.1 WPF 中的内建命令 …… 427
15.3.2 ICommand 接口及其实现类 …… 428
15.3.3 ICommandSource 接口 …… 434
15.4 与 Win32 机制交互 …… 434
15.4.1 WPF 与 Win32 程序的交互方法 …… 435
15.4.2 在 WPF 中使用其他类库控件 …… 437
15.4.3 在 Win32 中使用 WPF 控件 …… 441
15.4.4 完成交互功能的类与接口 …… 443
第 16 章 扩展 WPF 功能——自定义 WPF 类 …… 446
16.1 功能类派生 …… 446
16.1.1 更改通知的实现——INotifyPropertyChanged 接口 …… 446
16.1.2 自定义转换器——IValueConverter 接口及 TypeConverter 类 …… 448
16.1.3 数据模板选择逻辑——DataTemplateSelector 类 …… 452
16.1.4 可复用界面元素基类——Freezable 类 …… 453
16.1.5 对界面元素进行修饰——从 Adorner 类派生 …… 453
16.2 自定义界面元素 …… 454
16.2.1 自定义控件 …… 454
16.2.2 自定义面板 …… 462
16.2.3 自定义动画类 …… 464

试读结束：需要全本请在线购买： www.ertongbook.com

第1章 初识 WPF

在介绍 WPF 之前，请读者看一款用 WPF 创建的软件 Roxio Central 的界面。其效果如图 1-1 所示。

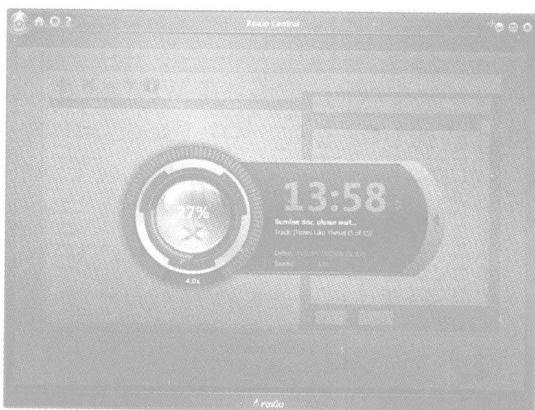

图 1-1 Roxio Central 的界面

Roxio Central 是微软出品的。它是一款可以在 Windows Vista 操作系统中运行的媒体刻录程序。与以往的基于 GDI 进行界面开发的程序不同，该软件的界面是通过 WPF（Windows Presentation Foundation）系统进行绘制的。

WPF 是为.NET 框架设计的全新的软件界面显示系统。为了能让软件开发人员方便地使用该系统所提供的各项功能，微软又提供了一个 WPF 编程类库，并希望借此将 WPF 定义成 Vista 操作系统程序的标准界面编程方式。在通常情况下，本书中所提到的 WPF 指的都是该类库。

WPF 的设计实际上借鉴了许多界面编程技术中的优秀设计理念，如网页技术中对布局和动画的支持。并且它的实现也使用了当前的多种热点技术，如利用 GPU 对软件界面渲染进行加速。而且在编程过程中，该类库可以将用户界面与业务逻辑完美地分离。可以说，无论对于软件的最终用户、软件界面设计人员还是软件架构的设计者，WPF 都将提供绝佳的用户体验。

也由于该类库中所引入的各种概念或多或少地借鉴了其他界面技术，因此在本书对 WPF 的介绍中，读者将常常看到 WPF 与其他编程技术的比较。首先，请读者先来看看 WPF 与以往的通用编程方式 GDI 之间的比较。

1.1 GDI 与 WPF

纵观 Windows 的可视化界面发展史，读者可以看到当前所使用的操作系统界面技术大部

分是从 Windows 1.0 建立和发展而来的。该技术的名称为 Graphics Device Interface，其缩写就是大名鼎鼎的 GDI。GDI 的主要任务是完成程序绘图功能与系统之间的信息交换，并完成 Windows 程序的图形输出。由于用户日常使用的程序中越来越多地采用了非典型界面，因此 Windows GDI 图形库的程序界面开发能力在某些程序的表现力需求面前显得越来越捉襟见肘。虽然说 GDI 在.net 框架中已经被微软进行了扩充，但是对于一个已经应用了十余年的图形界面技术而言，这种扩充并不能完全满足当前软件开发中对软件界面的越来越复杂的需求。因此发布一个新的程序界面开发技术势在必行。

实际上在当前版本的 WPF 中，用户可以使用的功能基本可以通过 GDI 完成。那为什么还要学习 WPF 呢？答案很简单：为了完成同样界面时能达到更好的效率以及效果，也为了与下一代操作系统兼容。

从操作系统的图形界面发展轨迹来看，在 Vista 之前的各个操作系统使用的界面绘制技术实际上是非常成功的，但同时存在着致命的缺陷。在 WPF 之前出现的界面绘制技术在根本上都是基于 Windows 组件 User32 和 GDI 之上。其中 User32 组件提供各个控件的外观绘制方式，而 GDI 提供的则是最基本的绘制功能。这两个组件构成了一个按需渲染的绘制系统。这里提到的按需渲染的定义是：当软件界面需要被绘制的时候，渲染系统将通过调用用户定义的代码对需要渲染的场景（DC，即 Device Context）进行绘制。该系统的好处是：由于对用户界面的绘制可以在系统发出绘制请求后立即被执行，因此它的绘制响应速度十分快，并且可以通过当前程序状态决定需要绘制的内容，不需要对界面元素的状态属性进行记录，节省了内存。但是该系统的缺点也同样明显：在对界面进行绘制的代码运行时间非常长的情况下，用户界面常常会失去响应，并最终会因为该段代码的运行影响了程序其他部分的绘制而使其他界面元素不再显示。由于 MFC 和 WinForms 都是通过 User32 和 GDI 来提供绘制功能，因此由这两个类库所实现的程序跳不出由于这两个组件的功能局限所导致的界面缺陷。

WPF 所提供的绘制功能则完全绕开了这种界面缺陷。这是因为通常的 WPF 程序都包含两个线程：界面控制线程和渲染线程。其中对用户输入进行响应并执行业务逻辑的线程都是界面控制线程，而对界面元素的绘制则是在通常不被用户访问到的渲染线程中完成的。当程序执行一个非常繁琐的任务导致任务执行函数不能正常返回时，渲染线程对界面的绘制并不会停止。因此从用户看来，程序的界面仍然可以进行交互。这种双线程渲染系统与原有绘制系统唯一相同的地方是：WPF 依然需要通过 User32 获得一些基础服务。这种双线程程序模型将在本书对 WPF 线程模型的介绍中涉及。

同时，GDI 绘制系统是基于剪切功能的系统。该系统中，需要被绘制的元素都有一个边缘矩形。在绘制时，该界面元素的超过该矩形的部分将被自动剪切。这样做的好处是：代码的编写并不需要知道过多的其他界面元素的信息。也正是因为如此，当一个界面元素需要进行绘制的时候，软件开发人员并不需要考虑其他与它交叠的界面元素的颜色，尽管他可以通过 CS_PARENTDC 获得 GDI 对交叠界面功能的一定程度的非原生支持。当然，GDI 绘制系统在 Vista 操作系统中并不是全无用处。无论是 Vista 还是 XP 系统，抑或无论是 WPF 程序还是普通的 GDI 程序，窗口都是通过基于剪切功能的 Win32 窗口与系统进行交互的，而这些交互通常也是由 User32 提供的，如鼠标点击、键盘输入等。

WPF 则是一个基于保持模式的绘制系统。并且该系统在保持模式之上提供了调色机制完成了对交叠界面元素绘制的原生支持。对于保持模式这一名词，读者可以简单地认为 WPF 将

每个界面元素的信息保留在系统中，并在绘制时根据这些在系统中保存的属性值对界面进行绘制。在对用户界面进行绘制的过程中，WPF会将上一个元素在程序界面中的绘制结果作为参与下一个界面元素绘制计算的参数。也就是说，如果一个按钮控件在绘制时将一个之前已经绘制的矩形的一部分覆盖了，并且这个按钮的透明度不为1，那么该按钮控件与矩形的公共部分的各个像素的颜色将不仅仅由按钮控件本身的颜色所决定，而是由按钮控件、矩形以及其他的在该点上存在的界面元素的颜色和透明度共同决定。图1-2展示的就是WPF中的控件叠加效果。

图1-2　WPF中矩形和按钮控件的叠加

相较于以往的以User32为基础的界面技术而言，WPF的另一个特点便是使用了矢量对界面元素进行绘制。下面的代码App1.1在WPF工程中声明了一个长度为0.7，宽度为0.3的按钮。该按钮中所显示的文字为“OK”。至于该段代码的意义，本书将在后面对它们进行讲解，读者不必理会。

```
代码 App1.1
<Window ...>
  <Canvas>
    <Rectangle Opacity="0.5" Canvas.Left="0" Canvas.Top="0" Width="0.7" Height="0.3" Fill="Blue"/>
  </Canvas>
</Window>
```

既然它的长度只为0.7，宽度只为0.3，都不足一个像素，根据读者以往对Windows程序界面绘制的理解，该矩形在运行时根本就不会被绘制。但是实际上，当用户在Vista系统下使用放大镜对该程序用户区的左上角进行观察的时候，他会发现那里赫然地绘制着一个蓝色矩形，如图1-3所示。

实际上，在WPF程序中，任何界面元素在理论上都是在无限分辨率下处理并显示的。因此在程序中，对界面元素的绘制实际上与屏幕分辨率无关。这是WPF的一个特性，本书将在后面对这个知识点进行介绍。这也是图1-3中所显示的图样中蓝色矩形还可以被显示的原因。

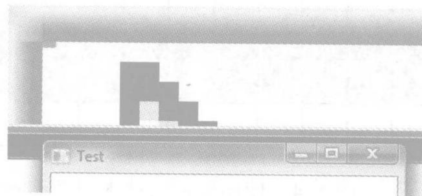

图1-3　WPF程序中不到一个像素的矩形

以往的GDI程序无法对这些特别小的细节进行显示的原因就在于屏幕的分辨率过低。通常情况下，程序所需要显示的图像的边缘并不完全与屏幕中各个显示单元完全重合。迄今为止，程序员们通常使用两种方法解决这个问题。一种是已经出现很长时间的抗锯齿（Anti-aliasing）技术，该技术通过在一个较高分辨率下计算界面元素的各个图像单元中的颜色并在渲染时混合各个图像单元，完成对每个像素的绘制。另外一种方法则是ClearType，它与抗锯齿技术非常相似，通过计算存在于各个像素中的各个界面元素的颜色所占比重完成对屏幕上一个像素的颜色的计算。但是由于缺少硬件驱动的支持，WPF编写的3D程序只有在Vista下可以使用该技术。

为什么ClearType技术在Vista和XP下会有不同的支持程度呢？这是因为Vista使用的是新一代驱动模型：WDDM。WDDM是Windows Display Driver Model的缩写。相较于上一代

驱动模型XPDM（Windows XP Display Driver Model），WDDM很大程度上支持了对GPU的使用，包括对GPU指令进行控制，以及将显存按照内存格式进行分页等。

通过上面介绍的各种处理技术，WPF就可以让它所定义的界面拥有分辨率无关的特性。在WPF中，界面元素的尺寸单位都是由硬件无关长度单位（device-independent units）所决定的。在通常情况下，它的长度是1in的1/96。如果软件开发人员希望一个界面元素的实际显示长度是1in，那么他就需要将该界面元素的长度声明为96。对于一个屏幕分辨率是1024×768像素的13.3in笔记本电脑而言，它的屏幕的DPI（Dot Per Inch）则为：$(1024^2+768^2)^{1/2}/13.3\approx96$。这也是该笔记本的默认配置，其中WPF所使用的硬件无关长度单位的长度则为DPI的倒数。如果用户将该笔记本的屏幕分辨率设置为800×600像素而没有对DPI设置进行手动更改，那么新的DPI则为$(800^2+600^2)^{1/2}/13.3\approx75$。此时这个界面元素的长度将在屏幕上显示为96/75 = 1.28in。需要读者注意的是，虽然DPI与屏幕的像素个数在默认情况下是相互对应的表示量，但这并不表明DPI与一英寸长度所包含的像素个数匹配。因此在800×600像素的屏幕分辨率下用户仍然可以将DPI手动更改为96，使得上面的界面元素在800×600像素的分辨率下的长度为1in。

这两种技术还具有同一个问题：通过它们绘制的图像看起来都是模糊的。比如WPF程序在白色屏幕上绘制了从点（17.4，20）到（17.4，30）的宽度为1的直线时，在用户的屏幕上将有可能显示成一段宽度为2的灰色直线。将该界面放大后一部分直线的效果可能如图1-4所示。

为了解决这个问题，WPF引入了一种新的技术：pixel snapping。通过它，WPF可以将直线及曲线精确映射到各个像素中。在使用pixel snapping技术的情况下，上面的直线绘制效果将如图1-5所示。

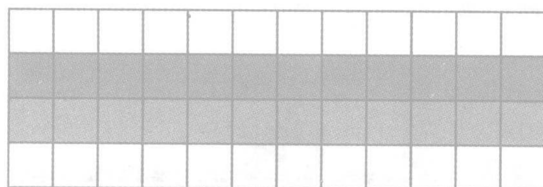

图1-4　使用抗锯齿技术以及ClearType技术绘制的直线效果

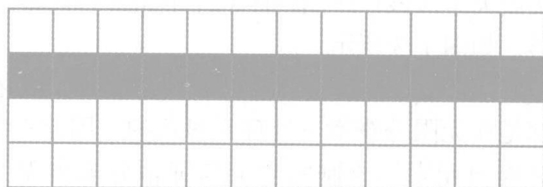

图1-5　使用pixel snapping技术绘制的直线效果

在默认情况下，该功能是关闭的。当需要某个界面元素使用该技术的时候，软件开发人员只需要将该类的SnapsToDevicePixels属性设置为true。

通常情况下，多种多样的界面特效也是软件界面技术中必不可少的一部分。其功能的完整性以及易用性往往成为了决定该界面技术是否成功的最主要原因。因此WPF专门提供了一部分功能完成对图像特效的支持。通过它，软件开发人员可以很容易地创建具有特殊效果的控件，更可以自己定义一些特殊效果，如图1-6所示。

或许读者已经对这些特效非常好奇。不必着急，相信读者在通读此书后就能完全了解这些特效是如何创建并使用的。

图1-6　WPF内建特效与自定义特效

1.2 WPF 架构

其实多年来，软件业已经出现了多个对图形进行显示的类库。20 世纪 90 年代，Silicon Graphics 出品的 OpenGL 迅速占领了对 2D 和 3D 图形进行处理的 CAD 软件、游戏等领域。1995 年，微软公司也推出了基于 Windows 的 DirectX 库。该库以及它的后续版本提供了高性能的图形处理功能，并同时提供了高性能的输入、音频功能。而从 DirectX 9 开始，微软就用托管代码对 DirectX 进行了包装，以使它可以应用到基于.NET 架构的程序开发中。

当托管代码可以使用 DirectX 的时候，微软即意识到，可以通过在界面显示过程中使用 DirectX，跳出软件界面实现过程中的 GDI 对软件界面样式的限制。于是，WPF 作为一种解决方案从微软公司诞生了。

仅仅从 WPF 的代码结构上来看，托管代码和非托管代码分居 WPF 的表层和底层。一般来说，托管代码向用户提供了构建 WPF 所需要的各种功能，如布局、绑定等。使用该部分代码的通常是界面管理线程。而非托管代码部分常常被称为媒体整合层（MIL，Media Integration Layer）。其主要功能集中在 WPF 的内部实现，如绘制以及元素合成功能等。使用该部分代码的通常是不可被用户接触的渲染线程。也就是说，剔除所有外部关联的 WPF 内部结构如图 1-7 所示。

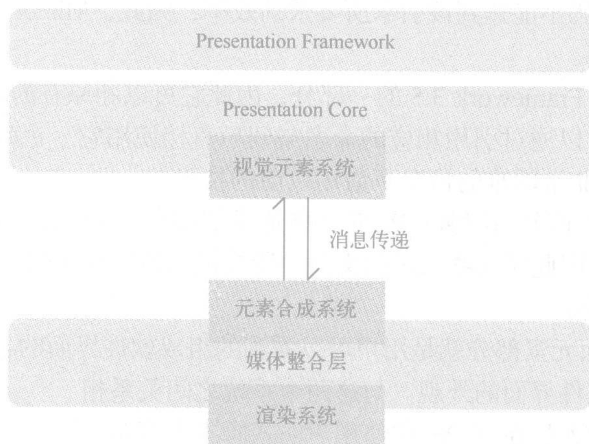

图 1-7 WPF 中的代码结构

可以看到，WPF 的托管代码部分包含一个重要的子系统：视觉元素系统。该系统是 WPF 的托管代码与非托管代码沟通的桥梁。它的内部常常保存 WPF 程序需要显示的视觉树（Visual Tree）结构。该树结构中的每个节点都存放着绘制该节点所需要的指令。而非托管代码部分的媒体整合层里也包含渲染及元素合成两个子系统。读者已经知道，WPF 的渲染功能实际上是通过 DirectX 来完成的。因此，渲染系统的功能仅仅是将 WPF 对界面的视觉树表示转化为 DirectX 函数调用，同时渲染系统中记录的数据也将转化为 DirectX 表示。另一个子系统则是与 WPF 上层的托管代码组件进行交互的元素合成系统。该系统对界面上的各个元素进行记录以待渲染线程使用。在该结构中存在两种表示界面组成的树型结构的原因是：在视觉元素系统中存在的视觉树是存在于界面管理线程中的，而相应的在元素合成系统中记录的元

素合成树状表示则存在于渲染线程中。

当然，WPF 中的各个组件并不是独立存在的。图 1-8 罗列出 WPF 组件与 Windows 组件混合之后的组件结构图，使读者对 WPF 各个组件在 Windows 中的位置有一个大概的了解。

在图 1-8 中，PresentationFramework，PresentationCore 和 Milcore 就是前面所介绍的 WPF 各子系统。组件 Milcore 是一个和 DirectX 进行交互的非托管组件，它负责对 WPF 界面元素进行渲染。设置这个组件的目的就是为了利用非托管代码的高效性这一优点。通过它，构建于其上的应用程序能更安全地和 DirectX 交互。该组件也是 WPF 以及 Vista 操作系统的一部分，比如，Vista 的桌面管理功能（DWM，Desktop Window Manager）也是通过 Milcore 来完成桌面绘制功能的。另一个组件 PresentationCore 表示的是 WPF 实现中的一个较低级别。该组件提供了一组基本服务，比如事件处理、输入、布局等，而这些功能的具体实现包含在动态链接库集合 mscorlib.dll，WindowsBase.dll 和 PresentationCore.dll 中。在组件 PresentaionCore 提供的基础功能上，组件 PresentationFramework 实现了 WPF 中的各种外观，比如按钮控件的实现、图形效果的实现等。该组件包含于动态链接库 PresentationFramework.dll 中。Milcore 组件中一个非常重要的功能就是上面介绍的元素合成系统。由于 CLR 代码不能达到该引擎所要求的效率，因此，Milcore 使用了一些非托管代码实现以获得执行效率。

图 1-8 Windows 系统架构

由于 WPF 是.NET Framework 3.5 的一部分，因此它可以和原有的.net 框架中的各个类进行完美的交互，甚至可以通过引用相应的命名空间后直接使用它。也就是说，完全可以在使用 WPF 实现一些功能非常困难的情况下借用以前的技术。

在通常情况下，程序员所接触的 WPF 功能通常都存在于 PresentationFramework.dll 以及其他托管代码组件中。因此对这些二进制文件中所提供的各种功能的分析十分必要。WPF 的子功能组成如图 1-9 所示。

在图 1-9 中，界面元素部分就是用户可以看到的组成软件界面的各个控件以及图案。而视觉系统则用来描绘软件界面的外观。与这两个系统之间关系相对应的是，当软件开发人员在 WPF 中使用声明式脚本语言对软件的界面元素进行描绘时，界面元素系统与视觉系统中将分别存在着逻辑树和视觉树。逻辑树用来表示各个界面元素之间的包含关系，而视觉树则用来表示用户所看到的界面中实际的各个视觉元素的对应关系。由于 XAML 中声明的界面元素常常是由多个其他的更基本的界面元素构成的，因此视觉树较逻辑树所描绘的界面元素之间的关系更为细致。除了这两个系统外，文字系统用来为 WPF 提供文字支持，输入系统则用来为 WPF 提供输入支持。属性系统是 WPF 中的一个重要概念。本书将在后面对它进行详细的讲解。在这里读者只需要知道该系统提供了许多关于属性的高级功能。WPF 中的各个子系统将在消息交换层中交换消息，并通过功能实现部分完成各个消息所对应的功能。

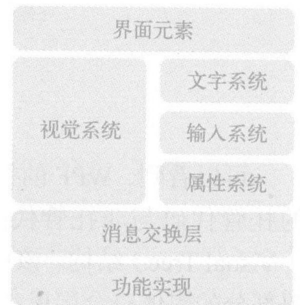

图 1-9 WPF 的子功能组成

就以上面的显示十分微小矩形的程序为例，WPF 将采用下面的顺序对该程序的界面进行

试读结束：需要全本请在线购买： www.ertongbook.com