

Visual C++ .NET 开发技术丛书

Visual C++ .NET

用户界面开发技术

李博轩 等编著

国防工业出版社

Visual C++ .NET

用户界面开发技术

李博轩 等编著



Visual C++ .NET 开发技术丛书

国防工业出版社

·北京·

05
10
02

内 容 简 介

本书通过大量实例深入浅出地介绍了 Visual C++ .NET 用户界面设计技术。全书共 10 章,主要内容包括:Windows 常用控件、菜单、工具栏、状态栏以及框架窗口等。

本书内容全面、深入,适合中高级读者、大专院校师生、企业技术开发人员学习参考,也适合各类培训班学员学习 Visual C++ .NET 用户界面设计技术。

图书在版编目(CIP)数据

Visual C++ .NET 用户界面开发技术/李博轩等编著.
北京:国防工业出版社,2002.9
(Visual C++ .NET 开发技术丛书)
ISBN 7-118-02887-8

I . V... II . 李... III . C 语言 - 程序设计
IV . TP312

中国版本图书馆 CIP 数据核字(2002)第 047898 号

国防工业出版社出版发行

(北京市海淀区紫竹院南路 23 号)

(邮政编码 100044)

北京奥隆印刷厂印刷

新华书店经售

*

开本 787×1092 1/16 印张 25½ 593 千字

2002 年 9 月第 1 版 2002 年 9 月北京第 1 次印刷

印数:1—4000 册 定价:37.00 元

(本书如有印装错误,我社负责调换)

前言

我们设计的程序是否能够吸引用户？是否总能寄希望于程序精彩的内涵？“酒好不怕巷子深”并不能在任何情况下适用。在一个软硬件技术不断发展的时代，对大多数并非计算机专家的用户来说，在他们面临多种选择时，怎样才能使自己的产品脱颖而出？这是一个对大多数程序，特别是商用程序来说相当重要的问题。

显然，别具一格的程序外观足以影响用户的选择。拥有友好而富于个性的用户界面的软件，其往往也具有更强大的功能，这也是经过事实证明的。另一方面，随着开发工具的不断发展，程序员所要做的工作越来越少，而这必然导致系统的许多细节被隐藏起来，能供程序员自由发挥的空间越来越小。其最直观的表现就是，通过 Visual C++ .NET 开发的应用程序具有极其相似的界面。如果你希望编写出富有个性化的界面，就必须付出更多的努力。

如果说漂亮的界面有利而无弊，那么我们怎么实现它呢——阅读本书就将为你提供最大的帮助。本书对 Windows 界面中最重要的组成元素，分别进行了介绍。对于每个元素，都通过大量经典实例深入浅出地向读者展示，如何使其具有更强大的功能和更漂亮的外观。但同时本书又绝非代码的堆积，而是试图通过对经典实例的分析，使读者理解界面设计的原则和方法，并进而对 Windows 高级编程技术有更深入的理解。

本书除封面署名作者外，武装、于佳音、朱石、周一兵、薛文涛、林茵茵、黄剑波、李义、王芳、沈鹏、刘树声、季洪飞、司马小凡、王江辉、廖晓筠、沈冰、汤春明、许颖、赵立峰、李国梁、胡建明、龚雪梅、黄君玲、吴强、郭士明等同志也为本书的出版付出了不同程度的劳动，在此一并表示感谢。

由于时间所限，书中错误和疏漏之处在所难免，敬请指正。

05172/01

目 录

第1章 用户界面设计基础	1
1.1 界面设计原则	1
1.1.1 保持界面的简单明了	2
1.1.2 保持界面的协调一致	2
1.1.3 设置恰当的界面布局	3
1.1.4 选择合适的界面颜色	3
1.1.5 选择易于理解的图片和图标	4
1.1.6 选择通用字体	4
1.1.7 隔离错误	4
1.1.8 提供用户辅助	5
1.1.9 其他原则	6
1.2 用户界面元素	7
1.2.1 菜单、控件和工具栏	7
1.2.2 窗口	8
本章小结	9
第2章 设计按钮控件	10
2.1 按钮设计基础	10
2.1.1 按钮控件概述	10
2.1.2 构造函数	11
2.1.3 操作函数	12
2.1.4 重载函数	15
2.1.5 CBitmapButton 类	17
2.2 改变按钮颜色	20
2.2.1 设计彩色按钮管理类	20
2.2.2 使用彩色按钮管理类	25
2.3 改变按钮形状	26
2.3.1 创建多边形按钮	26
2.3.2 创建圆形按钮	31
2.3.3 创建球形按钮	34
2.4 动态创建高级按钮	41
2.4.1 设计高级按钮管理类	41
2.4.2 动态创建	43
2.4.3 按钮绘制过程分析	45
2.4.4 使用高级按钮管理类	48
本章小结	51
第3章 设计编辑控件	52
3.1 编辑控件设计基础	52
3.1.1 编辑控件概述	52

	V
3.1.2 构造函数	53
3.1.3 属性操作函数	55
3.1.4 常规操作函数	58
3.1.5 剪贴板操作	62
3.2 CEditView 类	62
3.2.1 CEditView 类概述	63
3.2.2 构造函数	63
3.2.3 属性操作函数	63
3.2.4 常规操作函数	65
3.2.5 重载函数	65
3.3 改变控件的外观	67
3.3.1 能够保持“高亮”状态的编辑控件	67
3.3.2 鼠标敏感编辑控件	68
3.4 改变控件的编辑及显示方式	72
3.4.1 限制输入的数据类型	72
3.4.2 在位编辑	86
3.4.3 语法着色	89
本章小结	91
第 4 章 设计组合框控件	92
4.1 组合框控件设计基础	92
4.1.1 组合框控件概述	92
4.1.2 构造函数	94
4.1.3 常规操作函数	95
4.1.4 字符串操作函数	101
4.1.5 重载函数	103
4.2 改变组合框控件的行为	106
4.2.1 自动完成组合框控件	106
4.2.2 使用工具窗口替代列表框	113
4.2.3 鼠标敏感组合框控件	114
4.3 改变组合框控件选项形式	119
4.3.1 图标选择组合框控件	120
4.3.2 字体选择组合框控件	124
4.3.3 颜色选择组合框	138
4.4 增强列表框控件	142
4.4.1 CComboBoxEx 类概述	142
4.4.2 常用操作编程	146
本章小结	148
第 5 章 设计列表视图控件	149
5.1 列表视图控件设计基础	149

5.1.1 构造函数	149
5.1.2 属性操作函数	150
5.1.3 常规操作函数	165
5.1.4 重载函数	169
5.2 一个经典话题	170
5.3 动态改变列表视图的行高	179
5.4 改变列表视图控件的背景	183
5.4.1 改变背景颜色	183
5.4.2 使用位图背景	184
5.5 改善列表视图控件的交互方式	191
5.5.1 在列表视图控件中使用复选框	191
5.5.2 在位编辑子项	192
5.5.3 使用组合框控件	202
5.6 改变列表视图控件的标头显示	210
5.6.1 在标头中显示图像	210
5.6.2 在标头中使用图像列表	211
本章小结	216
第6章 设计树视图控件	217
6.1 树视图控件设计基础	217
6.1.1 树视图控件概述	217
6.1.2 构造函数	219
6.1.3 属性操作函数	219
6.1.4 常规操作函数	229
6.2 条目图像编程	234
6.2.1 设置条目图像	234
6.2.2 设置状态图像	235
6.2.3 使用覆盖图像	236
6.3 实现拖曳操作	237
6.3.1 处理拖曳	237
6.3.2 处理无意拖曳	240
6.3.3 使用 Esc 取消拖曳	241
6.3.4 处理拖曳操作中的滚动问题	242
6.3.5 在拖曳中保持条目等级	244
6.3.6 增强拖曳功能	246
6.4 树视图控件与工具提示	248
6.4.1 为条目图像添加工具提示	248
6.4.2 为条目添加工具提示	252
6.5 实现多重选择	253
6.6 改善条目形式和外观	259

6.6.1 鼠标敏感条目	259
6.6.2 为条目添加复选框	261
6.6.3 改变条目的字体和颜色	267
6.7 改善控件外观	271
6.7.1 改变控件背景颜色	272
6.7.2 使用位图背景	274
6.8 序列化树视图控件内容	278
本章小结	279
第 7 章 设计菜单	280
7.1 菜单编程基础	280
7.1.1 构造函数	280
7.1.2 初始化函数	281
7.1.3 菜单操作函数	283
7.1.4 菜单项操作函数	284
7.1.5 重载函数	292
7.2 使用标准菜单	292
7.3 使用快捷菜单	295
7.4 使用动态菜单	296
7.5 使用自绘制菜单	306
7.5.1 彩色菜单	306
7.5.2 图标菜单	310
本章小结	315
第 8 章 设计工具栏	316
8.1 工具栏设计基础	316
8.1.1 工具栏概述	316
8.1.2 构造函数	318
8.1.3 属性操作函数	320
8.2 使用标准工具栏	322
8.3 创建 IE 风格的工具栏	324
8.3.1 使工具栏具有“热敏”变色风格	324
8.3.2 在工具栏中显示文本	325
8.4 创建下拉菜单式工具栏按钮	326
8.5 在工具栏中使用控件	328
8.5.1 添加组合框控件	328
8.5.2 添加复选框控件	330
8.6 使用 16M 色位图创建工具栏	332
8.7 去除浮动工具栏中的系统菜单	333
8.8 排列多个工具栏	334
8.9 在对话框中使用工具栏和工具提示	335

8.9.1 创建工具栏	336
8.9.2 修改对话框尺寸	336
8.9.3 显示工具提示	337
8.10 在 MDI 应用程序中切换工具栏	339
本章小结	343
第 9 章 状态栏	344
9.1 状态栏编程基础	344
9.1.1 状态栏概述	344
9.1.2 构造函数	345
9.1.3 属性操作函数	346
9.1.4 重载函数	348
9.2 在状态栏中显示滚动效果的文本	348
9.3 在状态栏中输出时间	350
9.4 动态改变状态栏中的默认提示	352
9.5 在状态栏中使用控件	355
9.5.1 设计通用控件状态栏类	355
9.5.2 设计控件友元类	357
9.5.3 应用实例	361
9.5.4 使用自定义消息响应状态栏控件动作	363
9.5.5 使用注册窗口消息响应状态栏控件动作	363
本章小结	364
第 10 章 设计框架窗口	365
10.1 改变窗口效果	365
10.1.1 应用程序的默认图标	365
10.1.2 修改窗口的默认风格	366
10.1.3 改变窗口标题	368
10.1.4 改变窗口位置和排列	369
10.1.5 改变窗口形状	369
10.2 添加闪屏效果	372
10.3 添加窗口背景	373
10.4 创建不同形状的窗口	375
10.4.1 理解区域	375
10.4.2 简单实现技术	377
10.4.3 混合区域实现	382
10.4.4 综合实现	391
本章小结	400

第 1 章 用户界面设计基础

对于使用 Visual 系列编程的程序员来说，创建用户界面是一件非常容易的事，简单到只需将控件拖动到对话框中，或在代码中调用一两个函数，将自己绘制的位图设置为工具栏即可。然而正是这种简单性，使得创建出的应用程序越来越千篇一律，缺乏个性。关于这一点，读者可能也有所体会。在用户界面设计上多花一些心思不仅能美化程序、提高易用性，而且能增加对程序用户的吸引力（当然，从根本上说程序的功能才是最重要的因素）。

除非应用程序给自己使用，否则它的价值就要由别人决定。应用程序的界面对用户的影响很大——无论程序代码如何高效，功能如何强大，如果用户发现它太难于操控，那么这个程序就不会得到广泛的使用。要创建出优秀的用户界面，并不一定要拥有出色的艺术天赋——绝大多数界面设计的原则，与小学图画课上讲的完全一致。在那些课上介绍的组分、颜色等的设计原则，同样也适用于计算机屏幕，这与使用蜡笔在纸上画图没有什么不同。

在设计用户界面之前，最好先参考一下那些销路很好的软件界面，如果可能再参考一些销路并不很好的软件界面，这样就可以得到很多有用的信息。另外，自己在软件使用过程中，对于用户界面也可能有一些想法和期望。例如对于某个软件，你可能对它的一些地方很满意而对另一些则不太满意，那么就可以在自己的设计中加以改进和借鉴。当然，个人的看法不一定就适合大多数用户，因此还需要用户的确认。一般来讲，一个完善的应用程序都会为用户提供多样化的选择。这不仅提高了应用程序的可操作性，而且也提高了应用程序的吸引力。例如，IE 允许用户使用菜单命令、键盘命令或直接拖放来进行文件拷贝。在应用程序中，同一功能至少应该具备键盘和鼠标操作两种方式。

本章要点：

- ❖ 界面设计原则
- ❖ 用户界面元素

1.1 界面设计原则

对于大多数用户来说，他们并不太理解或关心应用程序的技术问题，而只是将其作为一种更简单的解决问题的工具。因此，好的应用程序界面应当隔离用户和底层技术，就像 MFC 类一样将底层功能包装在一个个的函数中。

用户界面设计一个总的原则就是用户至上。例如应用程序的设计是否能够吸引用户；

在没有帮助的情况下，用户是否能够比较容易地发现应用程序的不同功能；应该如何为用户提供在线帮助等。所有的这些都要求程序员考虑问题时，必须更多地从用户的角度出发。

1.1.1 保持界面的简单明了

除了 Windows 的常规支持外，在改进用户最佳体验的方面依然存在重要挑战。目前计算机的使用还相当复杂，并且其支持费用也很高。用户经常抱怨他们无法找到想要使用的功能，或不理解他们找到的功能是干什么的。

许多用户希望计算机易用、可预计、可靠和实用，希望计算机以他们熟悉的方式工作。因此，设计的简单明了是所有 Windows 应用程序设计都必须考虑的问题。但是界面的简单并不意味功能的简单，即使是简单的界面也可能需要大量的代码开发。当然，这种工作的回报和益处也是显而易见的，例如较少的培训和支持费用。

界面设计中最重要准则可能就是简单。简单即美，这也是艺术中的一个准则。如果应用程序的界面过于复杂，也会使用户望而却步。用户界面设计中经常出现的一个误区就是，试图以真实世界中的对象作为设计的模型。例如要设计一个用于完成保险表单的应用程序，很自然的反应就是创建一个实际表单的翻版作为界面。而这将会导致一些问题——纸的尺寸和形状与屏幕上的不同，完全照搬就会使界面元素限于文本框和复选框，而这并没有为用户带来任何方便。

最好的方法是设计自己的界面。通过逻辑关系创建不同的标签或链接表单，从而使用户无需滚动翻页即可得到全部信息。此外还可以使用列表控件为用户提供选择，这样能够减少用户的输入量，等等。只要多从用户的角度考虑，就会设计出相对简单易用的界面。

简化界面的另外一个有效方法，就是根据某个功能的使用频率决定其显示与否。例如 Word XP 中会智能化地根据某个按钮（控件）的使用频率，而决定其是否显示。有时提供默认值也会简化界面。例如，如果大多数用户都喜欢使用加粗的文本，那么将加粗风格设置为默认值，会比用户自己选择要方便一些，当然此默认值也必须是可修改的。

使用向导也能简化复杂的操作，这一点经常使用 Windows 软件的读者可能都深有体会。例如，Visual C++ .NET 中的应用程序向导，就大大简化了建立应用程序框架的过程。

要测试应用程序的简单性，其最好方法就是实际使用。如果在没有帮助的情况下，一个一般用户不能很快完成某个操作，那么就需要考虑重新设计了。

1.1.2 保持界面的协调一致

协调一致的用户界面会给人以舒适的感觉。相反，杂乱的用户界面不仅令使用者感觉不适，而且会使其低估应用程序的功能及其稳定性。

要保证用户界面的协调一致，需要注意以下几点：

- 为了保证视觉上的协调一致，需要在开发前进行大致规划。例如，控件的种类、尺寸以及字体等。预先制作一个模型将会有助于后续的开发。

- 在开发中尽量避免使用过多的控件。由于 Visual C++ .NET 提供了多种类型的控件，所以很容易使程序员产生各种类型控件都使用的想法，应该仔细选择最适合应用程序的一些控件。例如，列表框、组合框、列表视图控件和树视图控件都可用于显示一系列的信息，但是对于开发者来说，应该尽可能地选择单一风格的控件。

- 注意所用的控件是否合适。例如，使用只读编辑框显示文本，就不如使用静态文本控件更合适。

- 出于易用性的考虑，在应用程序的不同窗口中也需保证一定的协调一致。例如，在一个对话框中使用灰色背景，而在另一个对话框中使用白色背景，就会令整个应用程序给人以缺少内部联系的感觉。总之，选择什么风格就坚持什么风格，即便在某些情况下需要重新设计一些功能。

1.1.3 设置恰当的界面布局

应用程序的布局不仅影响其美观，而且对其本身的易用性也有着举足轻重的作用。这包括控件的位置、元素之间的协调性、空间的使用以及设计的简单明了。

1. 设置控件位置

在大多数界面设计中，并非所有的组成元素都具有同等的重要性，因此必须保证常用的重要元素处于最明显的位置。

绝大多数文字的书写顺序都是从左到右、从上到下，因此当用户观看屏幕时，会从左上角开始——最重要的元素应该放置在那里。例如，如果在屏幕上显示有关消费者信息的表单，则应该首先显示姓名字段；而诸如“确定”或“下一步”之类的按钮，则应该在屏幕右下方显示。这是因为，通常用户只有在完成了整个表单后，才会按下这些按钮。

另外，将信息、功能相近或关联的元素和控件分组也非常重要。分组排列一般是通过组框等控件完成的。

2. 调配空白空间

在应用程序的界面设计中，空白空间是指控件之间的未使用空间。如果过多的控件拥挤在一起，会增加寻找的时间，从而降低效率，所以适当地保留空白空间也很重要，它有助于改善程序的外观和加强对某些元素的强调。当然，也要注意不要造成过多的空间浪费。在设计时，需要综合考虑以确定最佳的分布。

使控件间距一致，并且保证控件垂直和水平对齐还会提高应用程序的易用性。这与报纸中文字的有序排列会方便阅读的道理一样。在 Visual C++ .NET 中提供了一些工具用于调整布局，当读者使用其资源编辑器时，会经常用这些空间调整界面外观。

1.1.4 选择合适的界面颜色

多彩的界面会增加应用程序的吸引力。由于现在使用的绝大多数显示器能够显示数百万种颜色，所以往往会导致颜色的过度使用。颜色的效果是千差万别的，而用户的喜好也会因人而异。因此在设计应用程序，特别是将广泛使用的应用程序时，需要注意不

同文化中颜色的差异。一般来说，在颜色的使用上应该尽量选择软色调和中性颜色。

当然，对颜色的选择也要考虑预期用户的情况。例如，在设计儿童软件时，亮红色、绿色和黄色是很好的选择；而对于金融或银行应用程序，则会产生反面影响。使用少量的亮色能够有效地突出重要区域。作为原则，在选择颜色时还应注意以下两点：

(1) 在应用程序中应该限制颜色的数量，而且应该保持前后一致。

(2) 应该尽可能使用标准 16 色调色板，这样会扩大应用程序的兼容性。

此外，色盲用户也是需要考虑入内的。

1.1.5 选择易于理解的图片和图标

在应用程序中使用图片和图标，也能提高应用程序对用户的吸引力。在设计应用程序时，应尽量使用简单的 16 色调色板图像，以提高应用程序的兼容性。

图片所传达的信息有时是文字所远远不及的，但其致命的弱点在于可能会导致误解。例如，工具栏中按钮上的不同图标，形象地表达了它的不同功能；但是当用户不能确定某个图标的功能时，就会产生反作用。在设计工具栏图标时，应该参考其他应用程序以得到已经建立的标准。例如绝大多数 Windows 应用程序，会将打开文件夹的图像作为“打开”命令的工具栏按钮图标。如果你的应用程序不遵循这一标准，就会给用户造成混乱。

此外还需要考虑文化的差异。例如美国应用程序经常将  作为“邮件”按钮的图标，而在其他国家中用户却并不将该图标看做邮筒，从而使对其功能的理解出现偏差。

1.1.6 选择通用字体

字体是用户界面的重要组成部分，因为它们通常用于与用户进行重要信息的交互。在选择字体时，必须保证它能在各种不同分辨率的显示器上都有良好的可读性。

除非计划将应用程序与字体一同发行，否则最好使用标准 Windows 字体，例如 Arial、New Times Roman 或 System 字体。因为如果用户的系统不包括指定的字体，则系统将使用其他字体，这就有可能造成实际的显示与预期的一致。例如，CoralDraw 10.0 英文版在 Windows 9x 下字体就无法完全显示，造成的后果就是文字不全。如果应用程序将在世界范围内发布，则需要为每种语言选择合适的字体。此外还需要考虑字体所占的空间，例如中文比英文所占的空间大 50%。这里还要旧话重提，一定要保持一致性，具有类似功能的文本要具有一致的字体。

1.1.7 隔离错误

在实际使用中，由于硬件的发展速度和软件的更新并不同步，所以软件和硬件并不能实现无缝结合。这也就是说，错误几乎是不可避免的，因此错误处理也就成了用户界面设计的一部分。当出现错误时，通常的方法是显示一个对话框，询问用户应该如何处理。而另一种不太常用的方法（但是更值得使用）就是计算机后台解决，而不是麻烦用户。总而言之，用户关心的是完成特定的任务，而不是技术细节。因此在设计用户界面

时, 需要考虑到可能出现的错误, 并确定哪些需要与用户交互处理, 而哪些应内部解决。

当在应用程序中设计错误信息对话框时, 首先要考虑这个信息对于用户是否有用; 其次是按钮所给出的选项意思是否清楚; 以及选项和消息是否适合, 等等。此外, 如果仅仅显示一个错误信息, 往往会给用户造成不好的印象。例如 Windows 著名的蓝屏现象。

并不是所有错误都要用户进行处理。对于在代码中就能够处理的错误, 一般无需通知用户, 或者以不中断当前进程的方式进行通知。关于这一点, Word 里面的“自动更正”功能就是一个很好的例子: 如果常用词出现错误, 则自动修改; 而如果不常用的词出现错误, 则以红色波浪线来表示, 以提醒用户修改。处理错误能够使用的技术有多种, 但哪个更适合则需要程序员自己作出决定。下面是一些通用的规则:

- 在“编辑”菜单中添加“撤消”命令。这要比打断用户当前的操作, 并显示对话框询问是否要撤消好一些。

- 在状态栏显示提示, 或使用闪烁的图标表示错误。这样用户可在方便的时候进行处理。

- 当错误很明显时, 直接改正它。例如当用户保存文件时, 如果当前驱动器已满, 则检查其他驱动器的空间, 如果有可用空间则保存文件, 同时在状态栏上显示信息, 告诉用户程序都做了什么。

- 并非所有的错误都需要立即解决, 这时可以采用将错误记录在日志文件中的方法, 供用户方便时浏览。例如在 Windows XP 中, 如果登录时网络连接出现问题, 则将问题记录在系统信息中, 用户可以通过系统信息管理器查询。

当应用程序出现错误时, 一般是通过对对话框与用户进行交互。对话框是应用程序界面的一部分, 它的设计对于应用程序的易用性非常重要。例如, 在保存文件到 C 盘出现错误时, 提示“C 盘中的一个扇区被损坏或不可访问。取消、重试或忽略?”之类的消息, 对于大多数用户来说毫无意义。这就好比点某一样菜却没有时, 服务员问你“可能饭菜卖完了或者厨房失火了。取消、重试还是忽略?”一样。所以我们应该以用户能够理解的方式来提示选择, 例如“在 C 盘中保存文件时发生错误。将文件保存到 A 盘, 还是不保存文件”。

1.1.8 提供用户辅助

无论界面多么优秀, 用户也还是会需要一些帮助。在应用程序中, 用户辅助模型通常包括在线帮助、帮助手册、工具提示、状态栏、“这是什么?”以及向导等。与应用程序其他部分的设计一样, 用户辅助模型的设计也应该在开发前进行。模型的内容则与应用程序的复杂程度以及预期的用户有关。

用户界面可用性检测的最好手段, 就是使用户自始至终地参与到设计过程中, 而不论所设计应用程序的使用范围如何。可用性检测的关键之一就是, 用户是否能够较为容易地了解如何使用应用程序的各种功能。例如在 Windows 3.1 中, 很少有用户知道使用 Alt+Tab 键能够在应用程序间进行切换, 因为几乎没有什么线索能帮助用户发现这一点。

要测试用户界面的可用性, 可以要求用户完成某个任务而不加任何解释(例如, 要

求用户从信函模板创建一个新文档)。如果他们不能完成,或中间经历了多次尝试,那么就需要重新考虑设计。

在线帮助是应用程序的重要组成部分——它通常是用户寻找问题答案的首选。在设计帮助系统时,要时刻牢记它的主要目的是回答问题。当创建主题和索引时,一定要从用户的角度出发。例如,语句“应该如何格式化页面?”要比语句“编辑、页面格式菜单”更容易使用户定位问题。

此外,一定要保证上下文敏感性,也就是说,当用户选择了“格式”然后按下 F1 键时,应该出现的是针对这个词的帮助,而不是出现帮助主题。印刷或直接放在光盘上的帮助手册,是另一种非常有用的工具,它们能提供简洁的帮助主题中难于传递的信息。如果没有帮助手册,那么至少也应该提供一份自述文件。

除以上外,工具提示、状态栏等也能够为用户提供很大的方便(使用 Visual C++ .NET 能够很轻易地为应用程序添加此类支持)。

1.1.9 其他原则

除了本节前述准则外,在设计应用程序时还需注意以下几点:

- 应用程序的安装步骤应尽可能少。
- 应用程序的安装不需要重启系统。
- 用户在使用应用程序前不一定非要阅读 **Readme**。
- 由用户生成的数据文件默认存储到“我的文档”文件夹中。
- 应用程序应避免向用户显示隐藏文件名。
- 应用程序不应在 **Program Files** 文件夹外创建文件夹。
- 应用程序不应将文件写入硬盘的根目录。
- 如果应用程序使用磁盘缓存,它还应注册到磁盘清理工具。
- 应用程序不应在“开始”菜单中包含 **Help**、**Readme** 和 **Uninstall** 文件项。
- 未经用户允许,应用程序不要在 **Windows** 桌面上安装图标。
- 如果应用程序在系统启动时运行,则不应显示闪屏和对话框。
- 应用程序不应使用任务栏显示状态、运行应用程序或工具、或查询属性,而只应在其中显示对于用户来说很重要的修改——例如,发现新硬件。
 - 应用程序能正确应用用户在“控制面板”/“显示属性”中做出的修改。
 - 应用程序应具备键盘访问性。
 - 如果用户增加默认字体尺寸,应用程序应该能正确工作。
 - 应用程序应尽可能地支持标准键盘快捷键集。
 - 应用程序的反安装处理不应遗留任何文件或注册表项,除非文件是由用户创建的。
 - 应用程序不应在用户界面文本中使用行话。只在确保用户能理解的情况下,才使用工业或技术术语。
- 当用户调整显示分辨率时,应用程序应能正确调整。

1.2 用户界面元素

Windows 操作系统的一个重要优势在于，它为所有应用程序提供了相同的界面。这样，一个有经验的用户能够很快掌握原先并没有使用过的 Windows 应用程序。菜单就是一个很好的例子：大多数 Windows 应用程序都遵循最左边是“文件”菜单，然后是“编辑”、“工具”等可选菜单，最右边是“帮助”菜单的规则。当然，也有人会有不同的想法（比如认为“文档”要比“文件”更合适，或“帮助”菜单应该排在最左边），并且也完全可以按照这些想法来设计。但是，这样就会给用户带来不必要的混乱，比如当用户在切换到另一个应用程序时，还必须调整一下使用方式，这样就降低了程序的可用性。

除菜单的排列顺序外，菜单命令的位置也很重要。例如，“保存”命令一般在“文件”菜单下，“拷贝”、“粘贴”命令一般在“编辑”菜单下等。总之，除非有充分的理由，否则不应该改变公认的 Windows 界面规则。使用已经存在的界面设计规则，不但能够使应用程序的可用性更好，而且会使自己的设计有个良好的开端（也就是说，用户界面的布局有个大概的框架）。

Windows 用户界面通常由窗口、控件、菜单和工具栏组成，如图 1-1 所示。实际上，从严格意义上说，控件、菜单和工具栏也是一种窗口，它们遵循与窗口一样的绘制和刷新模式。这一点始终贯穿在本书以后介绍的内容中。

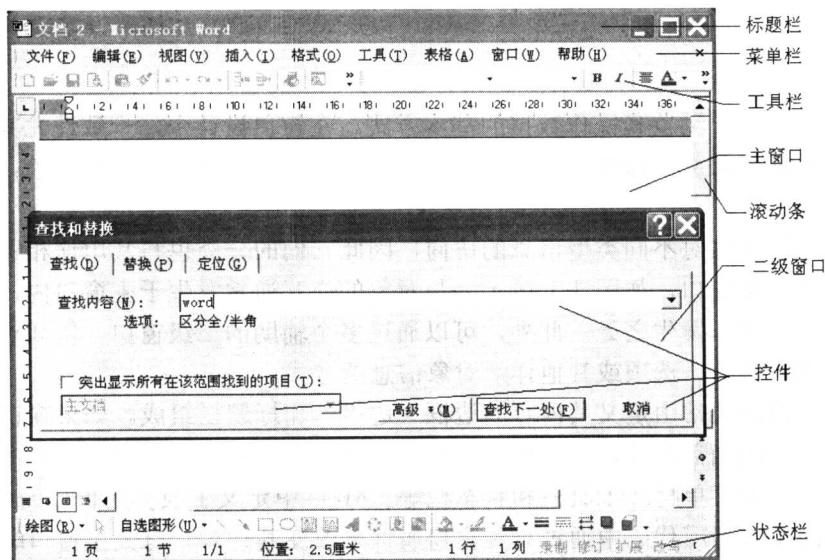


图 1-1 典型的 Windows 用户界面

1.2.1 菜单、控件和工具栏

Windows 提供了大量支持用户输入的组件。这些交互性组件允许用户执行命令和设

定值，另外，它们还为用户和自动化工具，提供了一致的结构和界面管理集。

1. 控件

控件是代表其他对象的属性或操作的图形化对象。每种控件都具有唯一的外观，以及为指定形式交互设计的操作。与大多数界面元素一样，控件可以为用户提供反馈。如果系统提供的控件已经具备了必要的功能，那么就尽量不要自己定制控件，因为这能使程序具有更大的兼容性和一致性。本书的第 2~6 章将通过一些最常用的控件，介绍控件外观的一些设计技巧。

2. 菜单

菜单用以显示用户可用的命令清单。由于菜单使得命令可见和可发现，因此可以最大限度地帮助用户记住和识别命令。菜单分为下拉式菜单、快捷菜单和级联菜单。有关菜单编程的知识请参见第 7 章。

3. 工具栏和状态栏

工具栏和状态栏是特殊的界面实体，它们可以像菜单一样管理一套控件。工具栏是包含一套控件的窗格，用以提供对特定命令或选项的快速访问。具有一组特定功能的工具栏还分别称为格式栏、工具箱或调色板等。本书第 8 章介绍了与工具栏设计相关的知识。

状态栏是窗口中的一个特殊区域（一般处于窗口底部），用以显示当前状态信息。状态栏中可包含按钮、编辑框等控件，但一般只利用它显示只读和不可交互的信息。本书第 9 章介绍了与状态栏设计相关的知识。

1.2.2 窗口

窗口提供了用户浏览和操作数据的基本方式。在窗口设计中，一致性尤其重要，因为这样用户不必学习新的操作规则，就能方便运用自己以往的软件使用经验，从而专注于完成自己的工作。

由于窗口能提供对不同类型信息的访问，因此它们的分类也与其用途相关。窗口可分为主窗口和二级窗口，如图 1-1 所示。与对象的交互通常发生于主窗口内，这也是大多数浏览和编辑动作发生之处。此外，可以通过多个辅助的二级窗口（例如对话框等）为用户提供指定参数、选项或其他详细对象信息的方式。

典型的主窗口通常由定义其内容的边框（边界）和标题栏组成。如果窗口的可浏览内容超过了窗口的当前尺寸，那么可以使用滚动条查看那些不可见的部分。主窗口中还包括其他元素，如菜单栏、工具栏和状态栏等。MFC 中定义了很多主窗口的外观特性，你可以在资源编辑器或代码中通过设置窗口属性来改变其外观。有关主窗口编程的内容请参见第 10 章。

大多数主窗口都需要一组二级窗口，以支持和辅助用户在主窗口中的动作。二级窗口与主窗口类似，但在某些行为方面存在差别。例如，二级窗口通常不会显示在任务栏。二级窗口中显示或获取的信息，通常都与主窗口中对象相关。在下文介绍控件设计时，会同时为读者介绍一些二级窗口的知识（如构造对话框）。