如果 nID 值等于 ID_SYSMENU_ABOUT,则意味着 About MyApp 被选中。如果 nID 等于别的,则必须调用基本类的 OnSysCommand 处理程序,否则系统菜单(和程序其他部分)就会不起作用了。在检测传递给 OnSysCommand 的 nID 值之前,一定要将它和 0xFFF0 相与清除 Windows 添加的所有位。

也可以使用 OnSysCommand 修改 Windows 放在系统菜单中的菜单项。下面的消息处理程序使系统菜单的 Close 命令在框架窗口中失效:

```
void CMainWindow::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) != SC_CLOSE)
        CFrameWnd::OnSysCommand(nID, lParam);
}
```

只要 nID 值代表的不是 Close 菜单项,则这个版本的 OnSysCommand 检测 nID 并将消息传递给 CFrameWnd。用 OnSysCommand 处理程序使 Close 失效的可选方法包括调用 CMenu::EnableMenuItem使菜单项失效,或调用 CMenu::DeleteMenu 将菜单项整个删除,如下所示:

```
CMenu* pSystemMenu = GetSystemMenu(FALSE);
pSystemMenu->EnableMenuItem(SC_CLOSE,    // Disable it.
    MF_BYCOMMAND|MF_DISABLED);
pSystemMenu->DeleteMenu(SC_CLOSE, MF_BYCOMMAND);  // Delete it.
```

Close 和其他系统菜单项的命令 ID 列在 OnSysCommand 的文档中。

4.3.4 自制菜单

显示字符串的菜单在大部分应用程序中都很容易实现,但是有些菜单强烈要求显示图片。举个例子如包含 Cyan 和 Magenta 命令的 Color 菜单。许多用户不知道青色是蓝色和绿色一半对一半的混合,或者品红是红色和蓝色等比例的混合。但是如果菜单给出的是样品色而不是字符串,那么菜单项的含义就一清二楚了。图形菜单的创建要比文字菜单麻烦些,但是效果也更好。

建立图形菜单最简单的方法是创建描述菜单项的位图,并在调用 CMenu::AppendMenu 时使用它们。MFC 用类 CBitmap 代表位图图像,而 AppendMenu 的一种形式接受指向 CBitmap 对象的指针,这样对象的图像就变成了菜单项。一旦 CBitmap 对象挂接到菜单上,则菜单显示时 Windows 就会显示位图。使用位图的缺点是位图尺寸固定,很难随屏幕度量的变化而改变。

菜单中用图形替代文字的更灵活的方法是使用自制菜单项。在包含自制菜单项的菜单显示时,Windows 发送给菜单所有者(即挂接菜单的窗口)一个 WM_DRAWITEM 消息,告诉说"该画菜单项了,请把它画在这里"。Windows 甚至还提供一个画图用的设备描述表。WM_DRAWITEM 处理程序会显示一个位图,或用 GDI 函数将菜单项画在给定的位置。在包

含自制菜单项的菜单首次显示之前，Windows 给菜单所有者发送一个 WM_MEASUREITEM 消息询问菜单项的尺寸。如果子菜单包含，比如说，5 个自制菜单项，则在子菜单首次显示时，挂接菜单的窗口就会收到 5 个 WM_MEASUREITEM 消息和 5 个 WM_DRAWITEM 消息。此后每次显示子菜单，窗口就只收到 5 个 WM_DRAWITEM 消息而不再有 WM_MEASUREITEM 消息了。

建立自制菜单的第 1 步是用标签 MF_OWNERDRAW 粘贴全部自制菜单项。不幸的是，除非 MENU 模板通过手工编程转变为 MENUEX 资源，否则 MF_OWNERDRAW 不能在模板中指定，而且至少 Visual C++ 资源编辑器不支持自制菜单。因此创建 MFC 应用程序中 MF_OWNERDRAW 项最好的办法是用 CMenu::ModifyMenu 编程实现传统菜单项到自制菜单项的转化。

第 2 步是添加响应 WM_MEASUREITEM 消息用的 OnMeasureItem 处理程序和相关联的消息-位图映射项。OnMeasureItem 的原型如下：

```
afx_msg void OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT lpmis)
```

nIDCtl 包含消息所属的控件的控件 ID，但与自制菜单无关。（WM_MEASUREITEM 消息既用于自制控件也用在自制菜单。创建控件调用 OnMeasureItem 时，nIDCtl 确定控件。）lpmis 指向类型为 MEASUREITEMSTRUCT 的结构，该结构有如下形式：

```
typedef struct tagMEASUREITEMSTRUCT {
    UINT    CtlType;
    UINT    CtlID;
    UINT    itemID;
    UINT    itemWidth;
    UINT    itemHeight;
    DWORD    itemData;
} MEASUREITEMSTRUCT;
```

OnMeasureItem 的工作是填充 itemWidth 和 itemHeight 字段，通知 Windows 该菜单项用像素表示的横向和纵向尺寸。OnMeasureItem 处理程序可简化为：

```
lpmis->itemWidth = 64;
lpmis->itemHeight = 16;
```

为减小不同视频分辨率的影响，较好的办法是将自制菜单中的各菜单项的尺寸建立在一个标准上，比如::GetSystemMetrics 返回的 SM_CYMENU 值：

```
lpmis->itemWidth = ::GetSystemMetrics(SM_CYMENU) * 4;
lpmis->itemHeight = ::GetSystemMetrics(SM_CYMENU);
```

SM_CYMENU 是系统给顶层菜单画的菜单栏的高度。通过在该值基础上确定自制菜单项的高和宽，您就能保证自制菜单项与 Windows 画的菜单项的尺寸比例大致协调了。

如果消息属于自制菜单则 MEASUREITEMSTRUCT 结构的 CtlType 字段设为 ODT_MENU;如果窗口包含自制控件和菜单项,则 MEASUREITEMSTRUCT 结构的 CtlType 字段用来区分不同的自制 UI 元素。CtlID 和 itemData 不能用于菜单,但是 itemID 包含菜单项 ID。如果应用程序创建的自制菜单项具有不同的高度和宽度,则可用该字段确定要求调用 OnMeasureItem 的菜单项。

建立自制菜单项的第 3 步,也是最后一步,是给消息 WM_DRAWITEM 提供一个 OnDrawItem 处理程序。画图实际上是在 OnDrawItem 中完成的。函数的原型如下:

```
afx_msg void OnDrawItem (int nIDCtl, LPDRAWITEMSTRUCT lpdis)
```

对于自制菜单项,又一次没有定义 nIDCtl。lpdis 指向 DRAWITEMSTRUCT 结构,它包含下列成员:

```
typedef struct tagDRAWITEMSTRUCT {
    UINT    CtlType;
    UINT    CtlID;
    UINT    itemID;
    UINT    itemAction;
    UINT    itemState;
    HWND    hwndItem;
    HDC     hDC;
    RECT    rcItem;
    DWORD    itemData;
} DRAWITEMSTRUCT;
```

同在 MEASUREITEMSTRUCT 中一样,如果消息属于自制菜单项,CtlType 设置为 ODT_MENU,itemID 保存菜单项 ID,而不使用 CtlID 和 itemData。hDC 保存菜单项所在的设备描述表的句柄,而 rcItem 是一个 RECT 结构,包含菜单项所在矩形的坐标。由 rcItem 描述的矩形尺寸是建立在响应该菜单项的 WM_MEASUREITEM 消息时提供给 Windows 的尺寸的基础上。Windows 并不把所画的图形剪贴到矩形中,而是依据程序代码正常工作并限制在 rcItem 定义的范围内。hwndItem 保存包含该菜单项菜单的句柄。由于其他字段提供了所需的全部或大部分信息,所以该值并不常用。

DRAWITEMSTRUCT 的 itemAction 和 itemState 字段描述必要的画图动作以及当前菜单项的状态——选中或未选中,有效或无效等等。对于自制菜单项,itemAction 包含下面两个值中的一个:ODA_DRAWENTIRE 表示要画出整个菜单项,而 ODA_SELECT 意味着当菜单项加亮显示或没有加亮显示时,可以有选择地重画部分菜单项。当加亮显示条从一个自制菜单项移到另一个上时,菜单的所有者接收到失去加亮条菜单项的不含 ODA_SELECT 标志的 WM_DRAWITEM 消息,并同时接收到另一个得到加亮条菜单项的包含 ODA_SELECT 标志的 WM_DRAWITEM 消息。包含自制菜单的程序往往忽略 itemAction 中的值。无论 itemAction 的值如何,程序都重画整个菜单项,并由 itemState 的值决定菜单项是否要加亮显示或不加亮显示。

itemState 包含零或表 4-5 中的多个标志位，确定菜单项的当前状态。

表 4-5 itemState 中的标志位

值	含 义
ODS_CHECKED	当前菜单项复选选中
ODS_DISABLED	当前菜单项无效
ODS_GRAYED	当前菜单项灰化
ODS_SELECTED	当前菜单项选中

状态信息非常重要，因为它决定应该如何画菜单项。需要检查哪些标志位取决于菜单项允许有几种状态。始终要检查有无 ODS_SELECTED 标志，如果该标志设定，则加亮显示该菜单项。如果应用程序包含复选或取消复选自制菜单项的代码，则应检查有无 ODS_CHECKED，如果该标志设定，则在菜单项旁边画一个复选标记。同样地，如果菜单项还有有效和无效状态，则检查有无 ODS_DISABLED 标志，并相应地画出菜单项。默认情况下，如果您给出的既不是 ON_COMMAND 处理程序也不是 ON_UPDATE_COMMAND_UI 处理程序，则 MFC 使该菜单项无效。因此即使应用程序没有明显地使菜单项无效，菜单项也是有可能变成无效的。如果将 CFrameWnd::m_bAutoMenuEnable 设置成 FALSE，MFC 对于框架窗口的这个功能也就失效了。

另一种使用自制菜单的可选方法是将菜单挂接到 CMenu 对象，并重载 CMenu 的虚函数 MeasureItem 和 DrawItem 执行画图动作。这个技巧对创建自包含菜单的对象很有用处，这种对象不依赖它们的所有者绘制菜单，自己就可以完成菜单绘制工作。然而，如果菜单从资源文件加载进来并挂接到窗口时，没有使用 CMenu 对象作为中介，那么由包含菜单的窗口画各个菜单项也是容易的。我们修改 Shapes 应用程序，让它包含自制的 Color 菜单时，就要用到这种方法。

OnMenuChar 处理

使用自制菜单的弊病是 Windows 没有为此提供键盘快捷键，比如 Alt-C-R 对应 Color-Red。即使在使用 ModifyMenu 将菜单项转变成 MF_OWNERDRAW 之前把菜单项正文定义为"&Red"，Alt-C-R 也不再有效。Alt-C 仍然会显示 Color 菜单，但是 R 键是毫无作用的。

Windows 从 WM_MENUCHAR 消息角度提供了一种解决方法。菜单显示并按下一个与菜单项无关的键时，窗口接收到一个 WM_MENUCHAR 消息。处理 WM_MENUCHAR 消息时，可添加与各自制菜单项对应的键盘快捷键。MFC 的 CWnd::OnMenuChar 函数原型如下：

```
afx_msg LRESULT OnMenuChar(UINT nChar, UINT nFlags, CMenu* pMenu)
```

调用 OnMenuChar 时，nChar 保存被按键的 ANSI 或 Unicode 字符代码；如果消息所属的菜单为子菜单，则 nFlags 保存 MF_POPUP 标志；而 pMenu 确定菜单自身。保存在 pMenu 中的指针

可能是主结构创建的临时指针,并不能被保存下来以备后用。

由 OnMenuChar 返回的值指示 Windows 如何响应击键。返回值的高位字要设置成下列值中的一个:

- 0 如果 Windows 应该忽略击键
- 1 如果 Windows 应该关闭菜单
- 2 如果 Windows 应该选择菜单中显示的某一菜单项

如果返回值的高位字是 2,则低位字就应该保存相应菜单项的 ID。Windows 为设置 LRESULT 值的高位字和低位字提供了 MAKELRESULT 宏。下面的语句把 LRESULT 值的高位字设置为 2,把低位字设置为 ID_COLOR_RED:

```
LRESULT lResult = MAKELRESULT(ID_COLOR_RED, 2);
```

当然,您可以总使用键盘加速键,而不用键盘快捷键。键盘加速键对于自制菜单项也很好用。但是因为有 WM_MENUCHAR 消息,您就也能为应用程序提供传统的键盘快捷键了。

4.3.5 层叠菜单

单击任务栏中的 Start 按钮时,就会出现一个弹出式菜单,上面列着各种各样的选择项:开始运行应用程序、打开文档、改变系统设置等等。某些菜单项旁边还有箭头,表示单击该项会触发另一个菜单。在某些场合下,这些菜单嵌套在几层菜单里。例如:单击 Start-Programs-Accessories-Games,Games 菜单是屏幕上一系列层叠菜单的第 4 层。这种层叠的菜单结构允许分级组织 Start 菜单中的各菜单项,并避免多个菜单杂乱放置,使得菜单在实际中无法操作。

层叠菜单不是操作系统的唯一属性,应用程序也能使用这种结构。创建层叠菜单很简单,正如把菜单项插入菜单,层叠菜单也是把一个菜单插入另一个菜单。Windows 做这些细致的工作,包括在菜单项名字的旁边画一个箭头,以及光标停在该项上时,不用单击层叠菜单便显示出来。如果 Shape 菜单是嵌套在 Options 菜单中的,应该这样定义 Shapes 的顶层菜单。

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",          ID_APP_EXIT
    END
    POPUP "&Options"
    BEGIN
        POPUP "&Shape"
        BEGIN
            MENUITEM "&Circle\tF7",      ID_SHAPE_CIRCLE
            MENUITEM "&Triangle\tF8",    ID_SHAPE_TRIANGLE
```

```
            MENUITEM "&Square\tF9",     ID_SHAPE_SQUARE
        END
        MENUITEM "&Color_",       ID_OPTIONS_COLOR
        MENUITEM "Si&ze_",        ID_OPTIONS_SIZE
    END
END
```

图 4-12 显示了所得菜单的外观。在 Options 菜单中选中 Shape 会显示一个层叠菜单。此外，程序其他部分的运行情况同以前一样，与 Shape 菜单中各菜单项有关的命令和更新处理程序都不用改变。

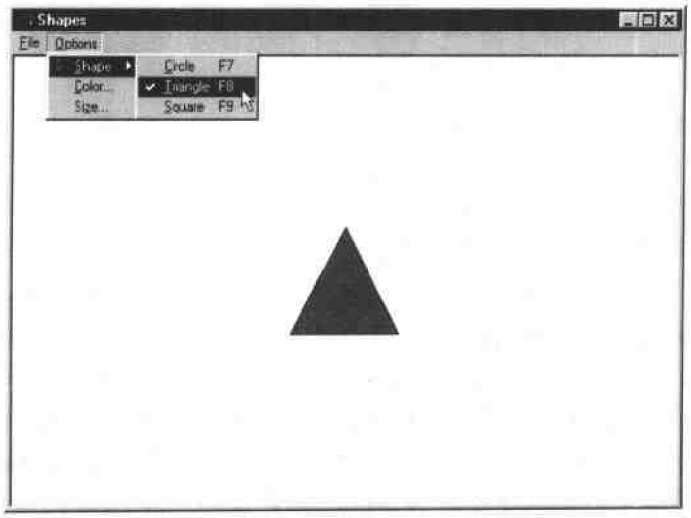

图 4-12　层叠菜单

不必通过手工编程编辑菜单资源创建层叠菜单。在 Visual C++ 菜单编辑器中通过选中 Menu Item Properties 对话框中的 Pop-up 复选框，您就能创建一个被嵌套的菜单。如图 4-13 所示。

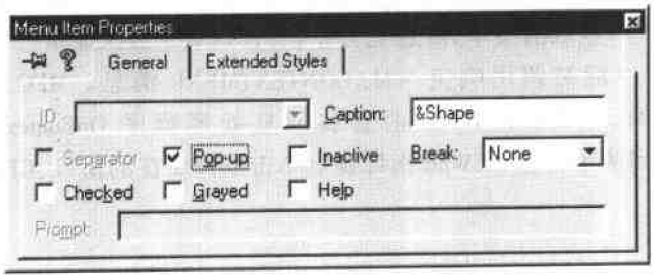

图 4-13　创建一个被嵌套的菜单

4.3.6 上下文菜单

Windows 大量使用右击上下文菜单,方便操作由外壳程序显示的对象。例如:在桌面上右击"我的电脑"图标,显示一个右击上下文菜单,上面简洁地列出了能在"我的电脑"中执行的动作:资源管理器、重命名、映射网络驱动器等等。右击桌面会产生完全不同的上下文菜单。开发人员应该在应用程序中建立上下文菜单,以便与外壳程序保持一致并补充面向对象的 UI(用户界面)的式样。在单击鼠标右键后,但还未处理就此引发的右击消息时通过向应用程序发送一个 WM_CONTEXTMENU 消息,Windows 很容易实现上下文菜单的创建。

上下文菜单和子菜单一样都不是挂接在顶层菜单上的。MFC 的 CMenu::TrackPopupMenu 函数显示一个上下文菜单。函数原型如下:

```
BOOL TrackPopupMenu (UINT nFlags, int x, int y, CWnd * pWnd,
    LPCRECT lpRect = NULL)
```

x 和 y 确定该菜单在屏幕上的显示位置(基于屏幕坐标)。nFlags 包含的标志位确定菜单相对于 x 值水平对齐,以及哪个鼠标键(或按钮)可用来选中菜单中的菜单项。对齐标志 TPM_LEFTALIGN、TPM_CENTERALIGN 和 TPM_RIGHTALIGN 告诉 Windows x 分别确定了菜单左边界、中心和右边界的位置,而 TPM_LEFTBUTTON 和 TPM_RIGHTBUTTON 标志确定鼠标左键或右键是否能进行菜单项选择。只能给定一个对齐标志,但是对于两个鼠标按钮标志则可以使用任何一个或同时使用。pWnd 确定窗口能接收到菜单中各种操作引发的消息,而 lpRect 指向一个 CRect 对象或 RECT 结构,它们包含一个矩形的屏幕坐标,在这个矩形中单击鼠标不会取消菜单。如果 lpRect 为 NULL,则在菜单外单击鼠标会取消菜单。假设 pMenu 是一个 CMenu 指针,指向一个子菜单,语句

```
pMenu -> TrackPopupMenu (TPM_LEFTALIGN|TPM_LEFTBUTTON |
    TPM_RIGHTBUTTON, 32, 64, AfxGetMainWnd ());
```

显示菜单,从屏幕左上角看,其左上角在右面 32 个像素、下面 64 个像素的位置。用户可以用鼠标左键或右键在菜单中选择。菜单显示时,应用程序的主窗口还同菜单是顶层菜单一部分那样接收消息。一旦取消菜单,消息就会停止,直到菜单再次显示。

TrackPopupMenu 一般被调用响应 WM_CONTEXTMENU 消息。MFC 的 ON_WM_CONTEXTMENU 宏把 WM_CONTEXTMENU 消息和消息处理程序 OnContextMenu 对应起来。OnContextMenu 接收到两个参数:CWnd 指针确定单击事件所在的窗口、CPoint 包含光标在屏幕上的坐标:

```
afx_msg void OnContextMenu (CWnd * pWnd, CPoint point)
```

如果有必要,可调用 CWnd::ScreenToClient 把 point 中的屏幕坐标转换为客户区坐标。由于鼠标消息随着光标来到某个窗口,所以 OnContextMenu 会接收到指向这个窗口的指针。

这似乎有点古怪，但这是有原因的。和其他消息不同，如果右击事件发生在子窗口（例如：一个下压式按钮控件），WM_CONTEXTMENU 消息按优先级选择，子窗口不处理这项消息。因此，如果窗口包含子窗口，它会接收到 WM_CONTEXTMENU 消息，其中 pWnd 包含指向一个子窗口的指针。

如果 OnContextMenu 处理程序查询了 pWnd 或 point，并决定不处理该消息，那么调用基本类的 OnContextMenu 处理程序就相当重要。否则，WM_CONTEXTMENU 消息就不会按优先级工作。更糟的是：右击窗口标题栏就不再显示系统菜单了。如果单击事件发生在窗口的上半部分，下面的 OnContextMenu 处理程序就显示 pContextMenu 引用的上下文菜单；如果单击事件发生在别处，处理程序便把它传递给基本类：

```
void CChildView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    CPoint pos = point;
    ScreenToClient(&pos);

    CRect rect;
    GetClientRect(&rect);
    rect.bottom /= 2; // Divide the height by 2.

    if (rect.PtInRect(pos)) {
        pContextMenu->TrackPopupMenu(TPM_LEFTALIGN |
            TPM_LEFTBUTTON|TPM_RIGHTBUTTON, point.x, point.y,
            AfxGetMainWnd());
        return;
    }
    CWnd::OnContextMenu(pWnd, point);
}
```

在基于视图的应用程序如 Shapes 中，WM_CONTEXTMENU 处理程序总是放在视图类中，因为响应右击事件并显示的对象都在那里。

如果要显示上下文菜单，怎样才能得到指向它的指针呢？一种方法是构造一个 CMenu 对象，并用它的成员函数创建菜单。另一种方法是按照顶层菜单的加载方式从资源文件中加载菜单。下面的菜单模板定义了一个包含单个子菜单的菜单：

```
IDR_CONTEXTMENU MENU
BEGIN
    POPUP ""
    BEGIN
        MENUITEM "&Copy", ID_CONTEXT_COPY
        MENUITEM "&Rename", ID_CONTEXT_RENAME
        MENUITEM "&Delete", ID_CONTEXT_DELETE
    END
END
```

下面的语句将菜单加载到 CMenu 对象,并以上下文菜单形式显示:

```
CMenu menu;
menu.LoadMenu (IDR_CONTEXTMENU);
CMenu * pContextMenu = menu.GetSubMenu (0);
pContextMenu -> TrackPopupMenu (TPM_LEFTALIGN |
    TPM_LEFTBUTTON|TPM_RIGHTBUTTON, point.x, point.y,
    AfxGetMainWnd ());
```

如果应用程序包含几个上下文菜单,可以把它们分别定义为独立的子菜单 IDR_CONTEXTMENU,并通过改变传递给 GetSubMenu 的索引检索 CMenu 指针。或者也可以把每个上下文菜单定义成独立的菜单资源。在任何情况下,只要把上下文菜单挂接到栈中的 CMenu 对象,则在对象失效时,菜单也会被清除。TrackPopupMenu 返回后,菜单不再使用,所以删除它可以释放部分内存,供其他场合使用。

TPM_RETURNCMD 标志

如何处理上下文菜单命令?同传统菜单命令处理方法一样:通过编写命令处理程序。也可以给上下文菜单中的命令编写更新处理程序。实际上,给一个传统菜单中的命令和上下文菜单中的命令分配同一个命令 ID 是完全合法的,并且还可以让它们使用同一个命令处理程序(如果您需要,同一个更新处理程序也行)。

有时,还需要得到 TrackPopupMenu 的返回值,确定选中了哪个菜单项,以便在现场处理命令,而不是把它委派给处理程序。这就是 TPM_RETURNCMD 的由来。如果传送的第一个参数中有 TPM_RETURNCMD 标志,则 TrackPopupMenu 返回菜单中被选中菜单项的命令 ID。A 0 返回值意味着菜单在菜单项没有被选中的状态下被消除。假定 pContextMenu 指代前面示例中用到的上下文菜单,下面的语句说明如何显示菜单并在用户选中时立刻响应:

```
int nCmd = (int) pContextMenu -> TrackPopupMenu (TPM_LEFTALIGN |
    TPM_LEFTBUTTON|TPM_RIGHTBUTTON|TPM_RETURNCMD,
    point.x, point.y, AfxGetMainWnd ());

switch (nCmd) {
case ID_CONTEXT_COPY:
    // Copy the object.
    break;
case ID_CONTEXT_RENAME:
    // Rename the object.
    break;
case ID_CONTEXT_DELETE:
    // Delete the object.
    break;
}
```

这样显示的菜单在某菜单项选中时还是要引发 WM_COMMAND 消息。一般情况下这不会有什么问题。因为如果该菜单项没有相应的命令处理程序，消息会传递给 Windows，而不会有别的影响。但是有时您需要禁止这些消息，例如您对传统菜单中某项和上下文菜单中某项使用了同一个 ID，但是又希望两菜单项的反应各不相同。为此，只要在调用 TrackPopupMenu 时包含一个 TPM_NONOTIFY 标志即可。

不要忘记：在默认方式下 MFC 使没有命令和更新处理程序的菜单项失效。因此如果使用 TPM_RETURNCMD 标志，您就知道在框架窗口中将 m_bAutoMenuEnable 设置为 FALSE 的必要性了。

4.4　COLORS 应用程序

让我们编写一个包含自制菜单和上下文菜单的应用程序结束本章吧。Colors 是 Shapes 应用程序的加速版，它提供了一个自制 Color 菜单和一个上下文菜单，用户从中可以选择形状和颜色。上下文菜单中的菜单项在功能上与 Shape 和 Color 菜单中的菜单项完全相同，甚至共享同一个命令和更新处理程序。如果用户用鼠标右键单击窗口中间的形状，则有上下文菜单显示，如图 4-14 所示。

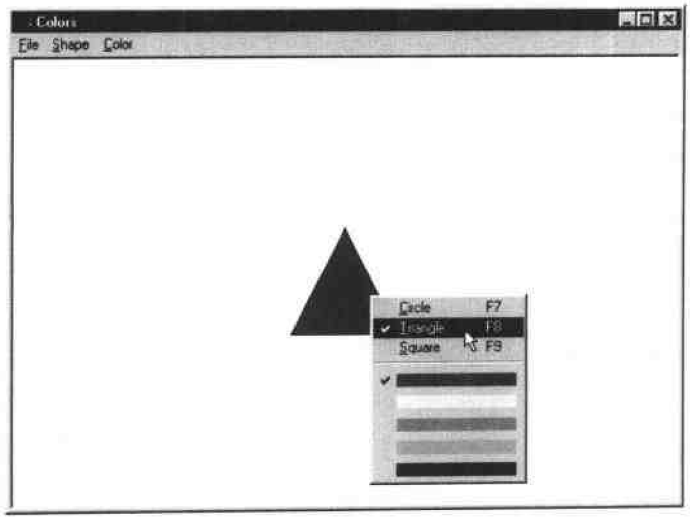

图 4-14　Colors 窗口

Colors 的源代码见图 4-15。生成这些源代码时，我先用 AppWizard 创建一个新项目，然后按照编写 Shapes 应用程序的步骤编写 Colors——实现 OnPaint，添加 Shape 菜单，编写命令和更新处理程序等等。最后，再添加 Color 菜单。即使菜单项被分配有“&Red”和“&Blue”文本字符串，但是由于菜单是自制，所以这些字符串不显示。InitInstance 中含有将 Color 菜单

中菜单项转化为自制菜单项的代码:

```
CMenu * pMenu = pFrame -> GetMenu ();
ASSERT (pMenu ! = NULL);

for (int i = 0; i < 5; i ++ )
    pMenu -> ModifyMenu (ID_COLOR_RED + i, MF_OWNERDRAW,
        ID_COLOR_RED + i);
```

第一个语句初始化 pMenu,使它的指针指向代表主菜单的 CMenu 对象。然后连续调用 ModifyMenu5 次,使 Color 菜单中的菜单项获得 MF_OWNERDRAW 标志。

Colors.h

```
// Colors.h : main header file for the COLORS application
//

#if !defined(AFX_COLORS_H__1B036BE8_5C6F_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_COLORS_H__1B036BE8_5C6F_11D2_8E53_006008A82731__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif //_MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"    // main symbols

/////////////////////////////////////////////////////////////////////////////
// CColorsApp:
// See Colors.cpp for the implementation of this class
//

class CColorsApp : public CWinApp
{
public:
    CColorsApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CColorsApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

public:
```

```
    //{{AFX_MSG(CColorsApp)
    afx_msg void OnAppAbout();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(AFX_COLORS_H__1B036BE8_5C6F_11D2_8E53_006008A82731__INCLUDED_)
```

Colors.cpp

```
// Colors.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Colors.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CColorsApp

BEGIN_MESSAGE_MAP(CColorsApp, CWinApp)
    //{{AFX_MSG_MAP(CColorsApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CColorsApp construction

CColorsApp::CColorsApp()
{
}
/////////////////////////////////////////////////////////////////////////////
// The one and only CColorsApp object

CColorsApp theApp;
```

```
/////////////////////////////////////////////////////////////////////////////
// CColorsApp initialization

BOOL CColorsApp::InitInstance()
{
    // Standard initialization

    // Change the registry key under which our settings are stored.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    CMainFrame* pFrame = new CMainFrame;
    m_pMainWnd = pFrame;

    // create and load the frame with its resources

    pFrame->LoadFrame(IDR_MAINFRAME,
        WS_OVERLAPPEDWINDOW|FWS_ADDTOTITLE, NULL,
        NULL);

    pFrame->ShowWindow(SW_SHOW);
    pFrame->UpdateWindow();

    //
    // Convert the items in the Color menu to owner-draw.
    //
    CMenu* pMenu = pFrame->GetMenu();
    ASSERT(pMenu != NULL);

    for (int i = 0; i < 5; i++)
        pMenu->ModifyMenu(ID_COLOR_RED + i, MF_OWNERDRAW,
            ID_COLOR_RED + i);

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CColorsApp message handlers

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
```

```
    //}}AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
// App command to run the dialog
void CColorsApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CColorsApp message handlers
```

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__1B036BEC_5C6F_11D2_8E53_006008A82731__INCLUDED_)
```

```
#define AFX_MAINFRM_H__1B036BEC_5C6F_11D2_8E53_006008A82731__INCLUDED

#if _MSC_VER > 1000
#pragma once
#endif //_MSC_VER > 1000

#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
    CMainFrame();
protected:
    DECLARE_DYNAMIC(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
        AFX_CMDHANDLERINFO* pHandlerInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
    CChildView    m_wndView;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg void OnSetFocus(CWnd *pOldWnd);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
    afx_msg void OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT lpmis);
    afx_msg void OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpdis);
    DECLARE_MESSAGE_MAP()
};
```

```
//////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
// !defined(AFX_MAINFRM_H__1B036BEC_5C6F_11D2_8E53_006008A82731__INCLUDED_)
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Colors.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_SETFOCUS()
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
    ON_WM_MEASUREITEM()
    ON_WM_DRAWITEM()
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
```

```
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    cs.lpszClass = AfxRegisterWndClass(0);
    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}
void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnSetFocus(CWnd* pOldWnd)
{
    // forward focus to the view window
    m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
    AFX_CMDHANDLERINFO* pHandlerInfo)
{
    // let the view have first crack at the command
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    // otherwise, do default handling
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
        CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL))
    {
```

```
        TRACE0("Failed to create view window\n");
        return -1;
    }
    return 0;
}

void CMainFrame::OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT lpmis)
{
    lpmis->itemWidth = ::GetSystemMetrics(SM_CYMENU) * 4;
    lpmis->itemHeight = ::GetSystemMetrics(SM_CYMENU);
}
void CMainFrame::OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpdis)
{
    BITMAP bm;
    CBitmap bitmap;
    bitmap.LoadOEMBitmap(OBM_CHECK);
    bitmap.GetObject(sizeof(bm), &bm);

    CDC dc;
    dc.Attach(lpdis->hDC);

    CBrush* pBrush = new CBrush(::GetSysColor((lpdis->itemState &
        ODS_SELECTED) ? COLOR_HIGHLIGHT : COLOR_MENU));
    dc.FrameRect(&(lpdis->rcItem), pBrush);
    delete pBrush;

    if (lpdis->itemState & ODS_CHECKED) {
        CDC dcMem;
        dcMem.CreateCompatibleDC(&dc);
        CBitmap* pOldBitmap = dcMem.SelectObject(&bitmap);

        dc.BitBlt(lpdis->rcItem.left + 4, lpdis->rcItem.top +
            (((lpdis->rcItem.bottom - lpdis->rcItem.top) -
            bm.bmHeight) / 2), bm.bmWidth, bm.bmHeight, &dcMem,
            0, 0, SRCCOPY);

        dcMem.SelectObject(pOldBitmap);
    }

    UINT itemID = lpdis->itemID & 0xFFFF; // Fix for Win95 bug.
    pBrush = new CBrush(m_wndView.m_clrColors[itemID - ID_COLOR_RED]);
    CRect rect = lpdis->rcItem;
    rect.DeflateRect(6, 4);
    rect.left += bm.bmWidth;
    dc.FillRect(rect, pBrush);
    delete pBrush;

    dc.Detach();
```

ChildView.h

```
// ChildView.h : interface of the CChildView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDVIEW_H__1B036BEE_5C6F_11D2_8E53_006008A82731__INCLUDED_)
#define AFX_CHILDVIEW_H__1B036BEE_5C6F_11D2_8E53_006008A82731__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CChildView window

class CChildView : public CWnd
{
// Construction
public:
    CChildView();

// Attributes
public:
    static const COLORREF m_clrColors[5];

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildView)
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildView();

    // Generated message map functions
protected:
    int m_nColor;
    int m_nShape;
    //{{AFX_MSG(CChildView)
    afx_msg void OnPaint();
    afx_msg void OnShapeCircle();
    afx_msg void OnShapeTriangle();
    afx_msg void OnShapeSquare();
```

```
    afx_msg void OnUpdateShapeCircle(CCmdUI * pCmdUI);
    afx_msg void OnUpdateShapeTriangle(CCmdUI * pCmdUI);
    afx_msg void OnUpdateShapeSquare(CCmdUI * pCmdUI);
    afx_msg void OnContextMenu(CWnd * pWnd, CPoint point);
    //}}AFX_MSG
    afx_msg void OnColor(UINT nID);
    afx_msg void OnUpdateColor(CCmdUI * pCmdUI);
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif
//!defined(AFX_CHILDVIEW_H__1B036BEE_5C6F_11D2_8E53_006008A82731__INCLUDED_)
```

ChildView.cpp

```
// ChildView.cpp : implementation of the CChildView class
//

#include "stdafx.h"
#include "Colors.h"
#include "ChildView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildView

CChildView::CChildView()
{
    m_nShape = 1; // Triangle
    m_nColor = 0; // Red
}

CChildView::~CChildView()
{
}

BEGIN_MESSAGE_MAP(CChildView,CWnd)
    //{{AFX_MSG_MAP(CChildView)
    ON_WM_PAINT()
```

```
    ON_COMMAND(ID_SHAPE_CIRCLE, OnShapeCircle)
    ON_COMMAND(ID_SHAPE_TRIANGLE, OnShapeTriangle)
    ON_COMMAND(ID_SHAPE_SQUARE, OnShapeSquare)
    ON_UPDATE_COMMAND_UI(ID_SHAPE_CIRCLE, OnUpdateShapeCircle)
    ON_UPDATE_COMMAND_UI(ID_SHAPE_TRIANGLE, OnUpdateShapeTriangle)
    ON_UPDATE_COMMAND_UI(ID_SHAPE_SQUARE, OnUpdateShapeSquare)
    ON_WM_CONTEXTMENU()
    //}}AFX_MSG_MAP
    ON_COMMAND_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnColor)
    ON_UPDATE_COMMAND_UI_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnUpdateColor)
END_MESSAGE_MAP()

const COLORREF CChildView::m_clrColors[5] = {
    RGB ( 255,   0,   0), // Red
    RGB ( 255, 255,   0), // Yellow
    RGB (   0, 255,   0), // Green
    RGB (   0, 255, 255), // Cyan
    RGB (   0,   0, 255)  // Blue
};

/////////////////////////////////////////////////////////////////////////////
// CChildView message handlers

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle |= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
        ::LoadCursor(NULL, IDC_ARROW), HBRUSH(COLOR_WINDOW+1), NULL);

    return TRUE;
}

void CChildView::OnPaint()
{
    CPoint points[3];
    CPaintDC dc(this);

    CRect rcClient;
    GetClientRect (&rcClient);
    int cx = rcClient.Width () / 2;
    int cy = rcClient.Height () / 2;
    CRect rcShape (cx - 45, cy - 45, cx + 45, cy + 45);

    CBrush brush (m_clrColors[m_nColor]);
    CBrush* pOldBrush = dc.SelectObject (&brush);
```

```
    switch (m_nShape) {
    case 0: // Circle
        dc.Ellipse (rcShape);
        break;
    case 1: // Triangle
        points[0].x = cx - 45;
        points[0].y = cy + 45;
        points[1].x = cx;
        points[1].y = cy - 45;
        points[2].x = cx + 45;
        points[2].y = cy + 45;
        dc.Polygon (points, 3);
        break;
    case 2: // Square
        dc.Rectangle (rcShape);
        break;
    }
    dc.SelectObject (pOldBrush);
}

void CChildView::OnShapeCircle()
{
    m_nShape = 0;
    Invalidate ();
}

void CChildView::OnShapeTriangle()
{
    m_nShape = 1;
    Invalidate ();
}

void CChildView::OnShapeSquare()
{
    m_nShape = 2;
    Invalidate ();
}
void CChildView::OnUpdateShapeCircle(CCmdUI * pCmdUI)
{
    pCmdUI -> SetCheck (m_nShape == 0);
}

void CChildView::OnUpdateShapeTriangle(CCmdUI * pCmdUI)
{
    pCmdUI -> SetCheck (m_nShape == 1);
}
```

```
void CChildView::OnUpdateShapeSquare(CCmdUI * pCmdUI)
{
    pCmdUI -> SetCheck (m_nShape == 2);
}

void CChildView::OnColor (UINT nID)
{
    m_nColor = nID - ID_COLOR_RED;
    invalidate ();
}

void CChildView::OnUpdateColor (CCmdUI * pCmdUI)
{
    pCmdUI -> SetCheck ((int) pCmdUI -> m_nID - ID_COLOR_RED == m_nColor);
}

void CChildView::OnContextMenu(CWnd * pWnd, CPoint point)
{
    CRect rcClient;
    GetClientRect (&rcClient);

    int cx = rcClient.Width () / 2;
    int cy = rcClient.Height () / 2;
    CRect rcShape (cx - 45, cy - 45, cx + 45, cy + 45);

    CPoint pos = point;
    ScreenToClient (&pos);

    CPoint points[3];
    BOOL bShapeClicked = FALSE;
    int dx, dy;

    //
    // Hit test the shape.
    //
    switch (m_nShape) {
    case 0: // Circle
        dx = pos.x - cx;
        dy = pos.y - cy;
        if ((dx * dx) + (dy * dy) <= (45 * 45))
            bShapeClicked = TRUE;
        break;

    case 1: // Triangle
        if (rcShape.PtInRect (pos)) {
            dx = min (pos.x - rcShape.left, rcShape.right - pos.x);
            if ((rcShape.bottom - pos.y) < (2 * dx))
                bShapeClicked = TRUE;
        }
```

```
            break;

        case 2: // Square
            if (rcShape.PtInRect (pcs))
                bShapeClicked = TRUE;
            break;
        }

        //
        // Display a context menu if the shape was clicked.
        //
        if (bShapeClicked) {
            CMenu menu;
            menu.LoadMenu (IDR_CONTEXTMENU);
            CMenu * pContextMenu = menu.GetSubMenu (0);

            for (int i = 0; i < 5; i++)
                pContextMenu -> ModifyMenu (ID_COLOR_RED + i,
                    MF_BYCOMMAND|MF_OWNERDRAW, ID_COLOR_RED + i);

            pContextMenu -> TrackPopupMenu (TPM_LEFTALIGN|TPM_LEFTBUTTON |
                TPM_RIGHTBUTTON, point.x, point.y, AfxGetMainWnd ());
            return;
        }

        //
        // Call the base class if the shape was not clicked.
        //
        CWnd::OnContextMenu (pWnd, point);
    }
```

Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Colors.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_COLORSTYPE                  129
#define IDR_CONTEXTMENU                 130
#define ID_SHAPE_CIRCLE                 32771
#define ID_SHAPE_TRIANGLE               32772
#define ID_SHAPE_SQUARE                 32773
#define ID_COLOR_RED                    32774
#define ID_COLOR_YELLOW                 32775
#define ID_COLOR_GREEN                  32776
#define ID_COLOR_CYAN                   32777
```

```
#define ID_COLOR_BLUE                32778

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE  131
#define _APS_NEXT_COMMAND_VALUE   32779
#define _APS_NEXT_CONTROL_VALUE   1000
#define _APS_NEXT_SYMED_VALUE     101
#endif
#endif
```

Colors.rc

```
//Microsoft Developer Studio generated resource script.

//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END
```

```
2 TEXTINCLUDE DISCARDABLE
BEGIN
    "# inc_ude ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "# define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "# define _AFX_NO_OLE_RESOURCES\r\n"
    "# define _AFX_NO_TRACKER_RESOURCES\r\n"
    "# define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "# if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "# ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "# pragma code_page(1252)\r\n"
    "# endif //_WIN32\r\n"
    "# include ""res\\Colors.rc2"
        "  // non-Microsoft Visual C++ edited resources\r\n"
    "# include ""afxres.rc""     // Standard components\r\n"
    "# endif\r\n"
    "\0"
END

# endif    // APSTUDIO_INVOKED

////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\Colors.ico"

////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Shape"
```

```
    BEGIN
        MENUITEM "&Circle\tF7",         ID_SHAPE_CIRCLE
        MENUITEM "&Triangle\tF8",       ID_SHAPE_TRIANGLE
        MENUITEM "&Square\tF9",         ID_SHAPE_SQUARE
    END
    POPUP "&Color"
    BEGIN
        MENUITEM "&Red",                ID_COLOR_RED
        MENUITEM "&Yellow",             ID_COLOR_YELLOW
        MENUITEM "&Green",              ID_COLOR_GREEN
        MENUITEM "&Cyan",               ID_COLOR_CYAN
        MENUITEM "&Blue",               ID_COLOR_BLUE
    END
END

IDR_CONTEXTMENU MENU DISCARDABLE
BEGIN
    POPUP "Top"
    BEGIN
        MENUITEM "&Circle\tF7",         ID_SHAPE_CIRCLE
        MENUITEM "&Triangle\tF8",       ID_SHAPE_TRIANGLE
        MENUITEM "&Square\tF9",         ID_SHAPE_SQUARE
        MENUITEM SEPARATOR
        MENUITEM "&Red",                ID_COLOR_ RED
        MENUITEM "&Yellow",             ID_COLOR_YELLOW
        MENUITEM "&Green",              ID_COLOR_GREEN
        MENUITEM "&Cyan",               ID_COLOR_CYAN
        MENUITEM "&Blue",               ID_COLOR_BLUE
    END
END

////////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "B",        ID_COLOR_BLUE,      VIRTKEY, CONTROL, NOINVERT
    "C",        ID_COLOR_CYAN,      VIRTKEY, CONTROL, NOINVERT
    "G",        ID_COLOR_GREEN,     VIRTKEY, CONTROL, NOINVERT
    "R",        ID_COLOR_RED,       VIRTKEY, CONTROL, NOINVERT
    VK_F7,      ID_SHAPE_CIRCLE,    VIRTKEY, NOINVERT
    VK_F8,      ID_SHAPE_TRIANGLE,  VIRTKEY, NOINVERT
    VK_F9,      ID_SHAPE_SQUARE,    VIRTKEY, NOINVERT
    "Y",        ID_COLOR_YELLOW,    VIRTKEY, CONTROL, NOINVERT
END
```

```
//////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 55
STYLE DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "About Colors"
FONT 8, "MS Sans Serif"
BEGIN
    ICON        IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT       "Colors Version 1.0",IDC_STATIC,40,10,119,8,SS_NOPREFIX
    LTEXT       "Copyright (c) 1998",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "OK",IDOK,178,7,50,14,WS_GROUP
END

#ifndef _MAC
//////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
FILEVERSION 1,0,0,1
PRODUCTVERSION 1,0,0,1
FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
FILEFLAGS 0x1L
#else
FILEFLAGS 0x0L
#endif
FILEOS 0x4L
FILETYPE 0x1L
FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "Colors MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "Colors\0"
            VALUE "LegalCopyright", "Copyright (c) 1998\0"
            VALUE "LegalTrademarks", "\0"
            VALUE "OriginalFilename", "Colors.EXE\0"
            VALUE "ProductName", "Colors Application\0"
```

```
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !MAC
/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END
END
#endif    // APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "Colors"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "Colors"
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
```

```
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT      "Display program information, version number and copyright\
nAbout"
    ID_APP_EXIT      "Quit the application; prompts to save documents\nExit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE      "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE      "Switch back to the previous window pane\nPrevious Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_SPLIT         "Split the active window into panes\nSplit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the selection and put it on the Clipboard\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the Clipboard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE         "Replace specific text with different text\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
```

```
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to save the docu-
ments"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////

#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL)|| defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\Colors.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#endif

/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

图 4-15 Colors 程序

因为框架窗口是菜单的所有者,因此它接收自制菜单项产生的消息 WM_MEASUREITEM 和 WM_DRAWITEM。于是消息处理程序在框架窗口类中。CMainFrame::OnMeasureItem 包含两个语句:一个确定各菜单项的高度(由::GetSystemMetrics 返回的值 SM_CYMENU),另

一个则确定宽度(SM_CYMENU * 4)。由于 CMainFrame::OnDrawItem 完成实际画图工作,所以显得更复杂些。在基础工作之后,包括创建一会儿将介绍到的 CBitmap 对象之后,OnDrawItem构造了一个空的 CDC 对象并通过调用 CDC::Attach 把 DRAWITEMSTRUCT 结构中提供的设备描述表句柄挂接到该对象上:

```
CDC dc;
dc.Attach (lpdis -> hDC);
```

这个语句将 dc 转化为一个有效的设备描述表对象,该对象封装了 Windows 提供的设备描述表。它返回 Windows 时的状态必须与被接收时的一致。在 OnDrawItem 结束之前必须清除选入设备描述表的对象,并消除对设备描述表状态所作的改变(例如:背景模式或文字颜色)。

第二步,OnDrawItem 创建了一个画刷。根据 itemState 字段中是否设置了 ODS_SELECTED 位,画刷的颜色或是 COLOR_MENU,或是 COLOR_HIGHLIGHT。而后通过指向画刷的指针调用 CDC::FrameRect,该画刷画一个矩形作为菜单项的轮廓:

```
CBrush * pBrush = new CBrush (::GetSysColor ((lpdis -> itemState &
    ODS_SELECTED) ? COLOR_HIGHLIGHT : COLOR_MENU));
dc.FrameRect (&(lpdis -> rcItem), pBrush);
delete pBrush;
```

COLOR_MENU 是默认的菜单背景色;COLOR_HIGHLIGHT 是菜单加亮条的颜色。CDC::FrameRect 用指定的画刷画一个线宽为 1 个像素的矩形。如果没有选中菜单项,上面的代码用背景色在菜单项周围画一个矩形;如果选中,则用加亮色。在下拉 Color 菜单并在其中上下移动鼠标时您就会看到这个矩形。如果 ODS_SELECTED 位清零,则用背景色画矩形;这样,当加亮条从一个菜单项移到另一个菜单项时,前一个选中矩形就会消失。

OnDrawItem 的下一个任务是:如果 ODS_SELECTED 位被设定,则在该菜单项旁边画一个复选标记。不幸的是,在使用自制菜单时,画复选标记是个需要特别注意的细节。更不幸的是,MFC 和 Windows API 都没有可以简化画复选标记的 DrawCheckMark 函数。可选的方法是创建一个描述复选标记的位图并调用 CDC::BitBlt 把复选标记"位块传送"(blit)到屏幕上。"位块传送"将在第 15 章详细介绍。但是尽管没有背景设置,OnDrawItem 代码还是比较容易理解的:

```
CDC dcMem;
dcMem.CreateCompatibleDC (&dc);
CBitmap * pOldBitmap = dcMem.SelectObject (&bitmap);

dc.BitBlt (lpdis -> rcItem.left + 4, lpdis -> rcItem.top +
    (((lpdis -> rcItem.bottom - lpdis -> rcItem.top) -
    bm.bmHeight) / 2), bm.bmWidth, bm.bmHeight, &dcMem,
    0, 0, SRCCOPY);

dcMem.SelectObject (pOldBitmap);
```

dcMem 代表内存设备描述表(DC)——内存中的一个虚拟显示平面,它同屏幕或其他输出设备一样,用户可以在这个虚拟显示平面上绘图。CreateCompatibleDC 创建一个内存 DC。Windows 不允许直接把位图"位块传送"(blit)到显示表面,因此必须先把位图选入内存 DC 并把它复制到屏幕 DC 中。在这个示例中,BitBlt 将内存 DC 中的位图复制到屏幕 DC 中,由 lpdis -> rcItem 描述的矩形的左端。当 BitBlt 返回时,在内存 DC 中清除位图,以便在 dcMem 失效时清除内存 DC。

位图从哪来? OnDrawItem 中的前 4 个语句创建了一个空的 CBitmap 对象,并把它初始化设置为 Windows 用来画菜单复选标记的位图,而后把位图信息(包括宽和高)复制到 BITMAP 结构:

```
BITMAP bm;
CBitmap bitmap;
bitmap.LoadOEMBitmap (OBM_CHECK);
bitmap.GetObject (sizeof (bm), &bm);
```

OBM_CHECK 是位图 ID,CBitmap::LoadOEMBitmap 将位图复制到 CBitmap 对象。CBitmap::GetObject 把位图信息复制到 BITMAP 结构,在调用 BitBlt 时要用到存储在结构中 bmWidth 和 bmHeight 字段的宽度和高度。OnDrawItem 结束时还要用到 bmWidth,这次是把每个颜色样本左端缩进一个复选标记的宽度。识别 OBM_CHECK 时,语句

```
#define OEMRESOURCE
```

必须在包含 Afxwin.h 的语句的前面。Colors 应用程序中,该 #define 在 StdAfx.h 中。

在画出或清除选中矩形并在 ODS_CHECKED 位设定画出复选标记后,OnDrawItem 画出有颜色的矩形表示菜单项本身。为此,OnDrawItem 创建了一个实心画笔,通过传送来的 DRAWITEMSTRUCT 中的 rcItem 结构创建了一个 CRect 对象,而后将矩形缩短几个像素,并调用 CDC::FillRect 绘制出该矩形:

```
UINT itemID = lpdis -> itemID & 0xFFFF; // Fix for Win95 bug.
pBrush = new CBrush (m_wndView.m_clrColors[itemID - ID_COLOR_RED]);
CRect rect = lpdis -> rcItem;
rect.DeflateRect (6, 4);
rect.left += bm.bmWidth;
dc.FillRect (rect, pBrush);
delete pBrush;
```

CDC::FillRect 是另一个 CDC 矩形函数。它用一个指定的画刷而不是用选入设备描述表的画刷填充矩形内部,并且不用当前画笔勾画这个矩形。运用 FillRect 而不是 Rectangle 时,不需要把画刷和画笔选入设备描述表并在结束时又将它们清除。传递给 FillRect 的画刷颜色是由 lpdis -> itemID 中的菜单项 ID 减去 ID_COLOR_RED 并把结果作为视图对象的

m_clrColors数组中的索引确定的。

下面谈谈 lpdis -> itemID：可以看到上段代码中该菜单项 ID 和 0xFFFF 相与。这样做可以避免在 Windows 95 中出现错误。如果您分配给自制菜单项的 ID 等于或超过 0x8000，当在 USER 的 16 位和 32 位部分间传送 ID 时，Windows 95 会不知情地把 ID 值的位扩展。结果是什么？结果是：命令 ID 0x8000 会变成 0xFFFF8000，0x8001 会变成 0xFFFF8001 等等这些变化，而 OnDrawItem 只有屏蔽掉前 16 位才能辨识出它自己的命令 ID。使用低于 0x8000 的 ID 值则能避免这个问题，只要在高位减 1 即可。但是很凑巧，由 Visual C++ 获取命令 ID 时，它所用的值都大于 0x8000，不必手工编程改变 ID，这里采用去除多余位的方法。这个问题在 Windows NT 中不存在，而在 Windows 98 中已得到解决。

OnDrawItem 最后的工作是把 dc 和来源于 DRAWITEMSTRUCT 的设备描述表句柄断开。这一步非常重要。因为在 OnDrawItem 结束时，它可以防止 dc 的析构函数删除设备描述表。正常情况下，在消息处理程序返回时您希望删除设备描述表，但是由于这个设备描述表是从 Windows 那儿挪用过来的，所以只有 Windows 可以删除它。CDC::Detach 将 CDC 对象和它的设备描述表分离，保证对象的正常失效。

4.4.1　上下文菜单

Colors 的上下文菜单来自于菜单资源 IDR_CONTEXTMENU。菜单资源在 Colors.rc 中是这样定义的：

```
IDR_CONTEXTMENU MENU DISCARDABLE
BEGIN
    POPUP "Top"
    BEGIN
        MENUITEM "&Circle\tF7",     ID_SHAPE_CIRCLE
        MENUITEM "&Triangle\tF8",   ID_SHAPE_TRIANGLE
        MENUITEM "&Square\tF9",     ID_SHAPE_SQUARE
        MENUITEM SEPARATOR
        MENUITEM "&Red",            ID_COLOR_RED
        MENUITEM "&Yellow",         ID_COLOR_YELLOW
        MENUITEM "&Green",          ID_COLOR_GREEN
        MENUITEM "&Cyan",           ID_COLOR_CYAN
        MENUITEM "&Blue",           ID_COLOR_BLUE
    END
END
```

通过用 Visual C++ 的 Insert-Resource 命令在应用程序中插入一个新菜单资源，我创建了上面这个菜单，并通过菜单编辑器给该菜单添加菜单项。

鼠标右键在视图中单击时，上下文菜单被加载并由 CChildView::OnContextMenu 显示。在加载菜单之前，OnContextMenu 命中测试窗口中的形状，如果单击事件发生在形状之外，则

向基本类发送 WM_CONTEXTMENU 消息。如果它确认单击事件发生在圆、三角形或正方形上,则加载菜单并在调用 TrackPopupMenu 之前将其中的菜单项转换为自制菜单项。

```
if (bShapeClicked)
{
    CMenu menu;
    menu.LoadMenu (IDR_CONTEXTMENU);
    CMenu* pContextMenu = menu.GetSubMenu (0);

    for (int i = 0; i < 5; i++)
        pContextMenu->ModifyMenu (ID_COLOR_RED + i,
            MF_BYCOMMAND|MF_OWNERDRAW, ID_COLOR_RED + i);

    pContextMenu->TrackPopupMenu (TPM_LEFTALIGN|TPM_LEFTBUTTON|
        TPM_RIGHTBUTTON, point.x, point.y, AfxGetMainWnd ());
    return;
}
```

每次加载菜单时都要执行这一转换。因为 menu 失效时,菜单被清除,并且对它所做的修改也会丢失。

Color 菜单和上下文菜单中的颜色根据 CChildView 消息映射表中的下面几项分别联接到命令处理程序 OnColor 和更新处理程序 OnUpdateColor:

```
ON_COMMAND_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnColor)
ON_UPDATE_COMMAND_UI_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnUpdateColor)
```

由于 ClassWizard 既不支持 ON_COMMAND_RANGE 也不支持 ON_UPDATE_COMMAND_UI_RANGE,我只好通过手工编程将这几项添加到资源代码中。MFC 编程人员为什么不能变得如此依赖生成代码的向导,其中 ClassWizard 不能支持这些宏就是一个非常重要的原因。向导虽然有用,但它们只支持一部分 MFC 的功能。我可能用 ClassWizard 给每个命令编写过独立的命令和更新处理程序,但是用手工编程的方法能更加有效地把 RANGE 宏写入消息映射表,因为原来需要 10 个独立的命令和更新处理程序,现在可以减少到 2 个。注意,通过手工编程添加到消息映射表中的几项要保证在 AppWizard 生成的 AFX_MSG_MAP 备注之外。消息映射表位于 ClassWizard 的备注之间。

如果要这些 RANGE 宏起作用,必须给 Color 菜单中菜单项分配连续的命令 ID,并由 ID_COLOR_RED 和 ID_COLOR_BLUE 分别作为最低和最高界限。为保证这些条件得到满足,您应该在菜单编辑器中创建菜单项时显式指定命令 ID,或在生成后编辑它们。可以在创建菜单项或通过在 Menu Item Properties 对话框(如图 4-16 所示)中输入的命令 ID 之后附加一个"= value"来编辑菜单项时指定一个数字的命令 ID。或者编辑 Resource.h 也行。对于 ID_COLOR_RED 和 ID_COLOR_BLUE 间的值我采用 32 774 到 32 778。

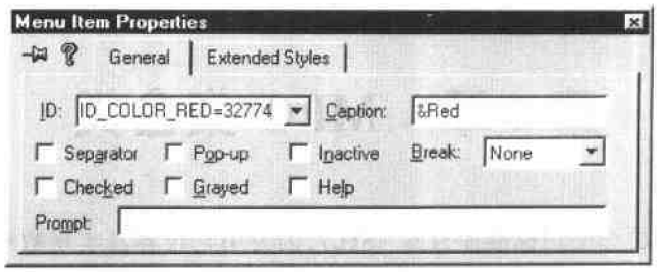

图 4-16　给菜单项 ID 分配数字值

4.4.2　试一试

有一个练习您可以自己试一试。到 ResourceView 并编辑图标资源 IDR_MAINFRAME。该资源是由 AppWizard 创建的，定义了应用程序的图标。图标包含两个图像：一个大的 (32 × 32)，一个小的 (16 × 16)。在应用程序发布以前应该编辑两个图像，使它们成为应用程序独特的图标。通过在图标编辑器的 Device 下拉表中选择 Standard 或 Small 您可以选出需要编辑的图像。如果打开包含 Colors.exe 的文件夹并选中 Large Icons 作为视图类，则大图标显示在操作系统外壳中。如果选中 Small Icons、List 或 Details，则显示小图标。由于 CFrameWnd::LoadFrame 中的部分代码加载了图标资源并把它挂接到窗口，所以小图标还出现在框架窗口的标题栏中。

第 5 章　MFC 集合类

许多 C++ 程序员都使用标准模板库 (STL),因为用它很容易实现数组、链接列表、映射以及其他容器。STL 语言中"容器"指的是保存数据集合的对象。但是在有 STL 之前,已经有 MFC 了。在称为"MFC 集合类"的一系列类中,MFC 提供了自己的数组、链接列表以及映射的实现途径。虽然在 MFC 应用程序中使用 STL 类非常安全,但许多 MFC 程序员还是更喜欢用 MFC 的集合类,一方面原因是他们更熟悉 MFC,另一方面原因是不愿意链接两个独立的类库而增加应用程序的 EXE 的尺寸。

有 MFC 集合类的帮助,您根本不必从头编写一个链接列表。本章将介绍 MFC 集合类并深入说明它们的使用和操作。

5.1　数组

C 和 C++ 的一个最大缺陷是数组不进行边界检查。看一下下面的代码,它反映了 C 和 C++ 应用程序中最常见的一种错误:

```
int array[10];
for (int i = 0; i <= 10; i ++ )
  array[i] = i + 1;
```

此代码出错是由于 for 循环中的最后一次迭代赋值超出了数组的范围。在运行时会产生非法存取错误。

C++ 程序员经常通过编写数组类并在内部进行边界检查来解决此问题。下面给出的数组类具有 Get 和 Set 函数,用来检查传递给它们的下标,如果传递来的下标无效就进行断言处理:

```
class CArray
{
protected:
  int m_nSize; // Number of elements in the array.
  int * m_pData; // Where the array's elements are stored.

public:
  CArray (int nSize)
```

```
    {
     m_nSize = nSize;
     m_pData = new int[nSize];
    }
    ~CArray()
    {
     m_nSize = 0;
     if (m_pData != NULL) {
        delete[] m_pData;
        m_pData = NULL;
     }
    }
    int Get(int nIndex)
    {
     assert(nIndex >= 0 && nIndex < m_nSize);
     return m_pData[nIndex];
    }
    void Set(int nIndex, int nVal)
    {
     assert(nIndex >= 0 && nIndex < m_nSize);
     m_pData[nIndex] = nVal;
    }
};
```

用这个简单的类作为整型数组的容器，下列代码在最后一次调用 Set 时产生断言提示：

```
CArray array(10);
for(int i = 0; i <= 10; i++)
  array.Set(i, i + 1); // Asserts when i == 10.
```

这样就会避免非法存取错误的发生。

5.1.1　MFC 数组类

您不必亲自编写数组类，MFC 已经提供了各种各样的数组。首先是一般的 CArray 类，它实际上是一个模板类，利用它可以创建任何数据类型的类型安全数组。在头文件 Afxtempl.h 中定义了 CArray。其次是非模板化的数组类，分别为保存特定类型的数据而设计。这些类在 Afxcoll.h 中定义。表 5-1 中列出了非模板化的 MCF 数组类以及它们所保存的数据类型。

表 5-1 特定类型的 MFC 数组类

类名	数据类型
CByteArray	8 位字节 (BYTE)
CWordArray	16 位字节 (WORD)
CDWordArray	32 位双字 (DWORD)
CUIntArray	无符号整型 (UINT)
CStringArray	CString
CPtrArray	void 指针
CObArray	CObject 指针

只要学会使用这些数组类中的一种,也就学会使用其他数组类了,因为它们共享公用的一组成员函数。下例声明了一个包含 10 个 UINT 的数组并用数字 1 到 10 对它进行了初始化:

```
CUIntArray array;
array.SetSize(10);
for (int i = 0; i < 10; i++)
  array[i] = i + 1;
```

可以采用相同的方法来声明一个 CStrings 数组并用整数 1 到 10 的文本表示来初始化它:

```
CStringArray array;
array.SetSize(10);
for (int i = 0; i < 10; i++) {
  CString string;
  string.Format(_T("%d"), i);
  array[i] = string;
}
```

在两个例子中,都是用 SetSize 来指定数组包含 10 个元素;重载"[]"运算符调用数组的 SetAt 函数,该函数将值复制到数组中指定位置处的元素中;如果数组边界非法,程序将执行断言处理。边界检查内置在 SetAt 代码中:

```
ASSERT(nIndex >= 0 && nIndex < m_nSize);
```

在 MFC 源程序文件 Afxcoll.inl 中您可以看到此代码。

可以使用 InsertAt 函数在不覆盖已有数组项的情况下给数组插入元素项。与 SetAt 不同,它只是给已存在的数组元素赋值,InsertAt 还要给新的元素分配空间,通过把数组中插入点上方的元素向上移动来完成。下列语句用数字 1 到 4 和 6 到 10 初始化一个数组并在数字 4 和 6 之间插入 5:

```
CUIntArray array;
```

```
array.SetSize (9);
for (int i = 0; i < 4; i ++)
  array[i] = i + 1;
for (i = 4; i < 9; i ++)
  array[i] = i + 2;
array.InsertAt (4, 5); // Insert a 5 at index 4.
```

还可以给 InsertAt 传递第三个参数指定元素项被插入的次数，或是在第二个参数中传递指向另一个数组对象的指针来插入整个数组。注意在本例中数组的大小为 9 个元素而不是 10 个，而在调用 InsertAt 时却没有执行断言处理。这是因为 InsertAt 是那些便于使用的函数之一，它们在新的元素项添加到数组中时自动增大数组尺寸。动态调整数组大小将在下一节讨论。

使用标准数组寻址语法可以在 MFC 数组中检索所要的值。下例将读取先前例子中写入 CUIntArray 中的 UINT：

```
for (int i = 0; i < 10; i ++)
   UINT nVal = array[i];
```

使用此方法，[]运算符将调用数组的 GetAt 函数，该函数从数组中的指定位置取回一个值(当然要进行边界检查)。如果愿意您可以直接调用 GetAt 而不用通过[]运算符。

要确定数组包含元素的个数，可以调用数组的 GetSize 函数。还可以调用 GetUpperBound 返回数组的上界下标，因为下标从 0 开始，所以其值为数组元素总数减 1。

MFC 的数组类为从数组中删除元素提供了两个函数：RemoveAt 和 RemoveAll。RemoveAt 从数组中删除一个以上的元素项并将被删除元素项上边的所有元素项向下移动。RemoveAll清空整个数组。两个函数都将调整数组的上界从而反映出被删除的元素项个数，说明如下：

```
// Add 10 items.
CUIntArray array;
array.SetSize (10);
for (int i = 0; i < 10; i ++)
   array[i] = i + 1;

// Remove the item at index 0.
array.RemoveAt (0);
TRACE (_T ("Count = %d\n"), array.GetSize ()); // 9 left.

// Remove items 0, 1, and 2.
array.RemoveAt (0, 3);
TRACE (_T ("Count = %d\n"), array.GetSize ()); // 6 left.

// Empty the array.
```

```
array.RemoveAll ();
TRACE ( _T ("Count = %d\n"), array.GetSize ()); // 0 left.
```

Remove 函数删除元素,但是如果元素是指针时它并不删除指针所指的对象。如果数组是 CPtrArray 或 CObArray 类型的,要清空数组并删除被删除指针所指的对象,就不应该写成:

```
array.RemoveAll ();
```

要写成:

```
int nSize = array.GetSize ();
for (int i = 0; i < nSize; i ++ )
   delete array[i];
array.RemoveAll ();
```

如果对地址保存在指针数组中的对象删除失败,就会导致内存漏损。对保存指针的 MFC 列表和映射也是这样。

5.1.2 动态调整数组大小

除了可以边界检查外,MFC 数组类还支持动态调整大小。由于为保存数组元素而分配的内存可以根据元素的添加或删除而增大或缩小,所以没必要事先预见动态调整尺寸的数组具有多少元素。

一种动态增大 MFC 数组的方法是调用 SetSize。可以在任何需要的时候调用 SetSize 来分配额外的内存。假设开始时给数组设置了 10 个元素项,后来却发现需要 20 个。这时只要第二次调用 SetSize 给额外的项分配空间即可:

```
// Add 10 items.
CUIntArray array;
array.SetSize (10);
for(int i = 0; i < 10; i ++ )
  array[i] = i + 1;
    .
    .
    .
// Add 10 more.
array.SetSize (20);
for(i = 10; i < 20; i ++ )
  array[i] = i + 1;
```

用此方法调整数组大小时,原来的项仍旧保持它们的值。因此,在调用 SetSize 之后只有新项需要明确的初始化。

另一种增大数组的方法是调用 SetAtGrow 而不是 SetAt 来添加元素项。例如:下列代码试图用 SetAt 给 UINT 数组添加 10 个元素项:

```
CUIntArray array;
for(int i = 0; i < 10; i ++ )
  array.SetAt (i, i + 1);
```

此程序会在第一次调用 SetAt 时就执行断言处理。为什么？因为数组大小为零时(注意没有调用 SetSize),SetAt 不会自动增大数组来容纳新的元素。但是,将 SetAt 更改为 SetAtGrow 后,程序将顺利执行:

```
CUIntArray array;
for (int i = 0; i < 10; i ++ )
  array.SetAtGrow (i, i + 1);
```

与 SetAt 不同,SetAtGrow 会在必要时自动增大数组的内存分配空间。Add 函数也是这样,它将元素项添加到数组的末尾。下一个例子的功能与上一个的相同,只是使用了 Add 而不是 SetAtGrow 来给数组添加元素:

```
CUIntArray array;
for (int i = 0; i < 10; i ++ )
  array.Add (i + 1);
```

其他可以自动增大数组来容纳新的元素项的函数还包括: InsertAt、Append(将一个数组附加给另一个数组)以及 Copy,顾名思义,它将一个数组复制到另一个数组中。

MFC 增大数组是通过分配新的内存缓冲区并将元素项从旧缓冲区中复制过去来实现的。如果由于内存不足使数组增大操作失败,则 MFC 会产生异常事件。为了在错误产生时捕获它们,要在 try 模块中封存扩大数组的调用命令,同时添加一个处理 CmemoryExceptions 的 catch 处理程序:

```
try {
  CUIntArray array;
  array.SetSize (1000); // Might throw a CMemoryException.
    .
    .
    .
}
catch (CMemoryException * e) {
    AfxMessageBox (_T ("Error: insufficient memory"));
    e -> Delete (); // Delete the exception object.
}
```

catch 处理程序显示一个出错消息,警告用户系统内存不足。在现实世界中,要想成功地跳出这种内存不足的状态还需要更进一步的处理。

由于每当数组尺寸增加时都要分配新的内存,所以太频繁地增大数组会对操作产生不好的影响并有可能导致产生内存碎片。考虑一下如下程序片段:

```
CUIntArray array;
for (int i = 0; i < 100000; i++)
  array.Add (i + 1);
```

这些语句看上去非常正确,但它们效率却不高,要申请分配成千上万个独立的内存。这也正是 MFC 让您在 SetSize 中可选的第二个参数内指定"增加量"的原因。下列代码更有效地初始化了一个数组,它告诉 MFC 在需要申请更多的内存时为 10 000 个新的 UINT 分配空间:

```
CUIntArray array;
array.SetSize (0, 10000);
for (int i = 0; i < 100000; i++)
  array.Add (i + 1);
```

当然,要是预先能给 100 000 个元素项分配空间,那么程序的效率会更高一些。但更经常的是不可能事先预见到数组要保存元素的数量。如果能预计到要给数组增加许多元素却不能确定到底会需要多少空间,那么指定大的增加量是有益的。

如果您没有指定增加量,MFC 会通过基于数组尺寸得到的简单公式为您选择一个值。数组越大,增加量也越大。如果指定数组尺寸为零,并且根本没有调用 SetSize,那么默认增加量为 4 项。在上一段的两个例子中,第一个的 for 循环对内存的分配和再分配不少于 25 000 次。而将增加量设置为 10 000 就使得内存分配数减少到了 10 次。

同样一个用来增大数组的 SetSize 函数也可以用来减少数组元素。但是,当它减小数组时,SetSize 并不会自动缩小保存数组数据的缓冲区。在调用数组的 FreeExtra 函数之前不会释放内存,说明如下:

```
array.SetSize (50);    // Allocate room for 50 elements.
array.SetSize (30);    // Shrink the array size to 30 elements.
array.FreeExtra ();    // Shrink the buffer to fit exactly 30 elements.
```

如果想将数组缩小到可以保存剩下元素所必须的最小尺寸,还应该在调用了 RemoveAt 和 RemoveAll 之后再调用 FreeExtra。

5.1.3 用 CArray 创建类型安全数组类

CUIntArray、CStringArray 以及其他 MFC 数组类都是针对特定数据类型的。如果假设需要一个其他数据类型的数组,例如:CPoint 对象,由于不存在 CPointArray 类,所以必须从 MFC 的 CArray 类中自己创建了。CArray 是一个模板类,用它可以为任意的数据类型创建类型安全数组类。

为明了起见,下列给出的程序示例中声明了一个 CPoint 对象的类型安全数组,并对类进行了实例化,然后用描述线段的 CPoints 数组将其初始化:

```
CArray < CPoint, CPoint& > array;
```

```
// Populate the array, growing it as needed.
for (int i = 0; i < 10; i++)
  array.SetAtGrow (i, CPoint (i * 10, 0));

// Enumerate the items in the array.
int nCount = array.GetSize ();
for (i = 0; i < nCount; i++) {
   CPoint point = array[i];
   TRACE (_T ("x = %d, y = %d\n"), point.x, point.y);
}
```

CArray 模板中的第一个参数指定了保存在数组中的数据类型，第二个参数指定类型在参数列表中的表示方法。可以使用 CPoints 来取代 CPoint 引用，但是在元素项的尺寸超出指针的尺寸时，使用引用会更有效。

在 CArray 模板参数中可以使用任何种类的数据，甚至是自己创建的类。下例中声明了一个代表三维点的类并用 10 个类实例填充了数组：

```
class CPoint3D
{
public:
  CPoint3D ()
  {
    x = y = z = 0;
  }
  CPoint3D (int xPos, int yPos, int zPos)
  {
    x = xPos;
    y = yPos;
    z = zPos;
  }
  int x, y, z;
};

CArray < CPoint3D, CPoint3D& > array;

// Populate the array, growing it as needed.
for (int i = 0; i < 10; i++)
  array.SetAtGrow (i, CPoint3D (i * 10, 0, 0));

// Enumerate the items in the array.
int nCount = array.GetSize ();
for (i = 0; i < nCount; i++) {
    CPoint3D point = array[i];
    TRACE (_T ("x = %d, y = %d, z = %d\n"), point.x, point.y, point.z);
}
```

使用 CArray 和其他基于模板的 MFC 集合类工作的时候,在创建的类中包含默认的构造函数很重要,因为 MFC 在类似 InsertAt 这样的函数被调用时会使用类的默认构造函数来创建新的元素项。

有了可以随意处理的 CArray,如果愿意的话,您可以不使用像 CUIntArray 这样的老式(非模板) MFC 数组类而只使用模板。下列 typedef 声明了一个 CUIntArray 数据类型,功能与 MFC 的 CUIntArray 等价:

```
typedef CArray < UINT, UINT > CUIntArray;
```

最终,选择哪个 CUIntArray 类取决于您。但是 MFC 资料中却建议尽可能地使用模板类,因为这样做可以与现代 C++ 程序设计惯例保持一致。

5.2 列表

InsertAt 和 RemoveAt 函数使得给数组添加和删除元素项非常简便。但这种插入和删除的简便方法也是有代价的:如果在数组的中间插入或删除元素,数组高端元素就会在内存中向上或向下移动。在用此方法处理大型的数组时,这种操作付出的代价是十分昂贵的。

要维护一个支持快速插入和删除的有序列表,传统的方法是使用链表。"链表"是包含指向其他列表项指针的列表项集合。在单链表中,每个列表项都包含一个指向列表中下一个列表项的指针。单链表中向前移动操作很快,因为要移动到下一个列表项处只要从当前列表项中提取该列表项的地址即可。要想支持快速向前向后移动,许多列表采取了双向链接,每个列表项都包含指向列表中前一个和后一个列表项的指针。给定列表中第一个列表项(列表头)的地址,用下列程序就可以很容易地实现列表项的枚举操作:

```
item* pItem = GetHead();
while (pItem != NULL)
  pItem = pItem->pNextItem;
```

反过来,给定列表中最后一项的地址(列表尾),双向链表可以以相反的顺序移动,如下:

```
item* pItem = GetTail();
while (pItem != NULL)
  pItem = pItem->pPrevItem;
```

这些例子假定列表不是循环列表,也就是说最后一个列表项中的 pNextItem 指针和第一个列表项中的 pPrevItem 指针都等于 NULL。一些链表将第一项和最后一项连结起来就形成了列表项的循环链。

链表是如何解决列表项快速插入和删除的呢?在链表中,将列表项插入到列表中并不需要在内存中移动任何列表项,只要求重新调整插入点前后的列表项指针来实现引用新的

列表项。删除列表项的工作量与插入的相同，只要修改两个指针即可。同样是在列表中间插入列表项，与那种需要使用 memcpy，可能会涉及到成千上万个列表项为一个新的列表项让位的方法相比，这种插入的方式优点很明显。

几乎所有的程序员在他们的职业生涯中都实现过链表。可能每个人都需要做一次，但绝对没有必要做第二次。幸运的是，许多类库，包括 MFC 在内，都提供了链表的封装实现。作为 MFC 程序员，您可以放心永远不必从零开始编写链表了。

5.2.1　MFC 列表类

MFC 的模板类 CList 实现了一般的链表，用它可以自定义处理任何数据类型。MFC 还提供了表 5-2 中列出的处理特定数据类型的非模板列表类。这些类主要用于与 MFC 旧版本兼容，在现代 MFC 应用程序中并不经常使用。

表 5-2　特定类型的 MFC 列表类

类名	数据类型
CObList	CObject 指针
CPtrList	void 指针
CStringList	CString

MFC 列表是双向链接的，便于前后移动操作。列表中的位置由抽象数值 POSITION 标识。对于列表，POSITION 实际上是指向 CNode 数据结构的指针，该结构代表了列表中的列表项。CNode 包含三个字段：一个指向列表中下一个 CNode 结构的指针、一个指向上一个 CNode 结构的指针以及一个指向列表项数据的指针。无论是在列表头还是列表尾，或是在 POSITION 指定的任何位置，插入操作都是快速高效的。还可以对列表进行查询操作，但是由于查询涉及到顺序遍历列表并逐个检查列表项，所以要是列表很大的话会占用很多时间。

我将借助 CStringList 来说明列表类的使用方法，但要注意这里所讲的许多原则对于其他列表类也是适用的。在下例中创建了一个 CStringList 对象并给它添加了 10 个字符串：

```
// Schools of the Southeastern Conference
const TCHAR szSchools[][20] = {
  _T ("Alabama"),
  _T ("Arkansas"),
  _T ("Florida"),
  _T ("Georgia"),
  _T ("Kentucky"),
  _T ("Mississippi"),
  _T ("Mississippi State"),
```

```
    _T ("South Carolina"),
    _T ("Tennessee"),
    _T ("Vanderbilt"),
};

CStringList list;
for (int i = 0; i < 10; i++)
  list.AddTail (szSchools[i]);
```

AddTail 函数在列表结尾处添加了一个列表项(或是另一个链接列表中所有列表项)。要给列表头添加列表项,可以使用 AddHead 函数。在列表头或尾删除列表项同样简单,只要调用 RemoveHead 或 RemoveTail 即可。RemoveAll 函数一下删除所有的列表项。

每次给 CStringList 添加一个字符串时,MFC 都会将字符串复制给 CString 并在相应的 CNode 结构中保存它。因此,用来初始化列表的字符串超出创建列表时设定的范围是完全可以接受的。

一旦列表创建成功,就可以使用 GetNext 和 GetPrev 函数通过迭代在列表中前后移动了。两个函数都接受表示列表中当前位置的 POSITION 值并返回该位置处的列表项。两者都要更新 POSITION 值来引用下一个或上一个列表项。可以使用 GetHeadPosition 或 GetTailPosition 来检索列表中列表头或列表尾的 POSITION。下列语句从头至尾列举了列表中的列表项,并通过 MFC 的 TRACE 宏将从列表中获得的字符串输出到了调试输出窗口:

```
POSITION pos = list.GetHeadPosition ();
while (pos != NULL) {
  CString string = list.GetNext (pos);
  TRACE (_T ("%s\"), string);
}
Walking the list backward is equally simple:

POSITION pos = list.GetTailPosition ();
while (pos != NULL) {
  CString string = list.GetPrev (pos);
  TRACE (_T ("%s\n"), string);
}
```

如果只希望得到列表的头或尾列表项,可以使用列表的 GetHead 或 GetTail 函数。由于位置已经隐含在调用中了,所以它们都不需要输入 POSITION 值。

如果给定标识特定列表项的 POSITION 值 pos,就可以使用列表的 At 函数来检索、修改、或删除它:

```
CString string = list.GetAt (pos);    // Retrieve the item.
list.SetAt (pos, _T ("Florida State")); // Change it.
```

```
list.RemoveAt (pos);    // Delete it.
```

还可以使用 InsertBefore 或 InsertAfter 在列表中插入列表项：

```
list.InsertBefore (pos,  _T ("Florida State"));   // Insert at pos.
list.InsertAfter (pos, _T ("Florida State"));   // Insert after pos.
```

链表的特性决定了这样进行插入和删除操作效率高。

MFC 的列表类包含两个成员函数，可以用来执行查找操作。FindIndex 接受从 0 开始的索引号并返回列表中相应位置处的列表项 POSITION。Find 查找与指定输入匹配的列表项并返回它的 POSITION。对于字符串列表，Find 比较字符串。对指针列表，它比较指针，但并不寻找和比较指针所指的列表项。要在字符串列表中查找“Tennessee”，只需要调用一个函数：

```
POSITION pos = list.Find (_T ("Tennessee"));
```

在默认状态下，Find 从头至尾查找列表。如果愿意，可以在函数第二个可选的参数中指定查找的起始点。但要注意，如果要找的列表项在起点 POSITION 的前面，Find 就不会找到它了，因为它不会返回到列表的开始进行查找。

可以用 GetCount 函数了解列表中元素的个数。如果 GetCount 返回 0，说明列表是空的。而检测空列表的快捷方式是调用 IsEmpty。

5.2.2　用 CList 创建类型安全列表类

可以利用 MFC 的 CList 类为所选的任何数据类型创建类型安全列表类。下面给出一个 CPoint 对象的链接列表：

```
CList < CPoint, CPoint& > list;

// Populate the list.
for (int i = 0; i < 10; i ++ )
   list.AddTail (CPoint (i * 10, 0));

// Enumerate the items in the list.
POSITION pos = list.GetHeadPosition ();
while (pos != NULL) {
   CPoint point = list.GetNext (pos);
   TRACE (_T ("x = %d, y = %d\n"), point.x, point.y);
}
```

与 CArray 相同，第一个模板参数指定了数据类型（CPoint 对象），第二个参数指出参数列表中列表项的传送方式（通过引用）。

如果在 CList 中使用了类而不是原始数据类型并且调用列表的 Find 函数,除非下列条件之一成立,否则程序不会得到编译:

- 类具有重载了的 == 运算符,执行与相似对象的比较。
- 用特殊类型的版本覆盖了模板函数 CompareElements,执行对两个类实例的比较。

第一种方法(重载 == 运算符)更常用,在 MFC 类如 CPoint 和 CString 中已经为您实现了。如果自己亲自编写一个类,就必须进行运算符重载。下面给出一种修改后的 CPoint3D 实现,为与 CList::Find 兼容而重载了比较运算符:

```
class CPoint3D
{
public:
  CPoint3D()
  {
    x = y = z = 0;
  }
  CPoint3D(int xPos, int yPos, int zPos)
  {
    x = xPos;
    y = yPos;
    z = zPos;
  }
  operator== (CPoint3D point) const
  {
    return (x == point.x && y == point.y && z == point.z);
  }
  int x, y, z;
};
```

除了重载比较运算符外,还可以覆盖全局 CompareElements 函数,如下:

```
class CPoint3D
{
public:
  CPoint3D()
  {
    x = y = z = 0;
  }
  CPoint3D(int xPos, int yPos, int zPos)
  {
    x = xPos;
    y = yPos;
```

```
    z = zPos;
  }
  // Note: No operator--
  int x, y, z;
};

BOOL AFXAPI CompareElements(const CPoint3D* p1, const CPoint3D* p2)
{
   return (p1->x == p2->x && p1->y == p2->y && p1->z == p2->z);
}
```

覆盖 CompareElements 消除了对重载运算符的需要，这是因为默认的 CompareElements 实现在被 CList::Find 调用时，将使用比较运算符来比较列表项。如果您覆盖了 CompareElements，而且在覆盖函数中也不会使用==，那也就不需要重载==运算符了。

5.3　映射表

在所有 MFC 集合类中，映射表应该说是最有意思的了。“映射表”，也称为“词典”，是一种表格，其中一种项目是另一种项目的关键字。一个简单的映射表例子是美国 50 个州的列表，用每个州的两个字母缩写作为关键字。例如：给出一个 CA，通过简单函数调用就可以检索出相应的州(California)。设计映射表的主要目的就是给定一个关键字，可以很快地在表中找到相应的项目，通常只查找一次。如果查找操作是首要工作，那么映射表就是大量数据的理想容器。MFC 使用映射表来实现句柄映射表(将 HWNDs 与 CWnds、HPENs 与 Cpens 等等联系起来的表)以及其他内部数据结构。它还使映射表类成为公用类，以便您可以创建自己的映射表类。

5.3.1　MFC 映射表类

除了基于模板的映射表类 CMap 以外，它可以被用来处理特殊的数据类型，MFC 还提供了表 5-3 中列出的特定类型的(不是基于模板的)映射表类。每个类都包含如下成员函数：添加和删除项目、通过关键字检索项目以及枚举映射表中所有项目。

表 5-3　特定类型的 MFC 映射表类

类名	说　明
CMapWordToPtr	保存 void 指针，关键字为 WORD
CMapPtrToWord	保存 WORD，关键字为 void 指针
CMapPtrToPtr	保存 void 指针，关键字为其他 void 指针
CMapWordToOb	保存 CObject 指针，关键字为 WORD

续表

类名	说　明
CMapStringToOb	保存 CObject 指针,关键字为字符串
CMapStringToPtr	保存 void 指针,关键字为字符串
CMapStringToString	保存字符串,关键字为其他字符串

为说明使用映射表的语法规则,下面我们使用 CMapStringToString 创建一个简单的包含一周各天名称的英法词典。下列语句生成一个映射表:

```
CMapStringToString map;
map[_T("Sunday")]     = _T("Dimanche");
map[_T("Monday")]     = _T("Lundi");
map[_T("Tuesday")]    = _T("Mardi");
map[_T("Wednesday")]  = _T("Mercredi");
map[_T("Thursday")]   = _T("Jeudi");
map[_T("Friday")]     = _T("Vendredi");
map[_T("Saturday")]   = _T("Samedi");
```

在本例中,保存在映射表中的项目是法语一周内各天的名称。而每个项目的关键字是对应英语名称的字符串。[]运算符将项目和它的关键字插入映射表。因为 CMapStringToString 在 CString 对象中保存关键字和项目,所以插入项目就是将项目文本和关键字文本复制给 CStrings。

对于这样初始化了的映射表,现在要检索星期四的法语名称。您可以通过调用映射表的 Lookup 函数并指定关键字来进行查找:

```
CString string;
if (map.Lookup (_T("Thursday"), string))
  TRACE (_T("Thursday in English - %s in French\n"), string);
```

从 Lookup 返回非零值说明成功地检索到项目了。返回为零表示该项目不存在,也就是说,没有项目的关键字是 Lookup 的第一个参数指定的关键字。

可以使用 RemoveKey 和 RemoveAll 从映射表中删除项目。GetCount 返回映射表中项目的数量,IsEmpty 检查映射表是否包含任何项目。使用 GetStartPosition 和 GetNextAssoc 可以逐项枚举映射表中的内容:

```
POSITION pos = map.GetStartPosition();
while (pos != NULL) {
  CString strKey, strItem;
  map.GetNextAssoc (pos, strKey, strItem);
  TRACE (_T("Key = %s, Item = %s\n"), strKey, strItem);
```

```
}
```

运行以上给出的 CMapStringToString 对象，程序会产生以下输出：

```
Key = Tuesday, Item = Mardi
Key = Saturday, Item = Samedi
Key = Wednesday, Item = Mercredi
Key = Thursday, Item = Jeudi
Key = Friday, Item = Vendredi
Key = Monday, Item = Lundi
Key = Sunday, Item = Dimanche
```

如上所示，项目并没有必要按加入时的顺序保存。这是针对尽可能快地检索项目而不是维持项目顺序设计映射表的必然结果。映射表的体系结构将在下一节介绍。

偶尔，如果您在给映射表插入项目时，所插入的项目具有与以前插入项目相同的关键字，那么新的项目将会取代旧的。MFC 映射表不可能包含由同一个关键字标识的两个以上的项目。

5.3.2　映射表工作方式

如果查找操作不迅速，那么就不能说映射表是非常出色。最大化操作效率的关键是最小化查找中要检查项目的数量。顺序查找是最差的一种方式，如果映射表具有 n 个项目，它所要求的最大查找数就是 n。二分法查找较好，但要求具有有序项目。最好的查找算法是无论给定多少项目，不进行其他任何查询就可以直接找到要找的项目。听上去不太可能？完全可能。如果映射表设置得当，MFC 的 Lookup 函数通过只要一次查询就可以找到任何项目。事实上很少需要两次或三次的查询。原因如下。

在映射表生成后不久（通常在添加第一个项目时或之前），会为一个散列表分配内存空间，该表实际上是一个指向 CAssoc 结构指针的数组。MFC 使用 CAssoc 结构来代表给映射表加入的项目（和关键字）。对于 CmapStringToString 如下定义 Cassoc：

```
struct CAssoc
{
  CAssoc* pNext;
  UINT nHashValue;
  CString key;
  CString value;
};
```

无论何时项目被加入映射表，就会创建一个新的 CAssoc 结构，根据项目的关键字计算散列值，指向 CAssoc 结构的指针被复制到索引号为 i 的散列表中，其中 i 由以下公式计算得到：

```
i = nHashValue % nHashTableSize
```

nHashValue 是从关键字计算来的散列值;nHashTableSize 是散列表中的元素个数。默认散列表大小为 17 个输入项,很快我将讲述修改其尺寸的方法和原因。如果偶尔索引号 i 处的元素已经包含了一个 CAssoc 指针,MFC 就会创建一个 CAssoc 结构的单向链接列表。列表中第一个 CAssoc 结构的地址保存在散列表中。第二个 CAssoc 结构的地址保存在第一个 CAssoc 结构的 pNext 字段,如此等等。图 5-1 说明了添加 10 个项目之后散列表的样子。在本例中,五个项目的地址被单独保存在了散列表中,而其他五个被分开保存在两个链接列表中,列表长度分别为 2 和 3。

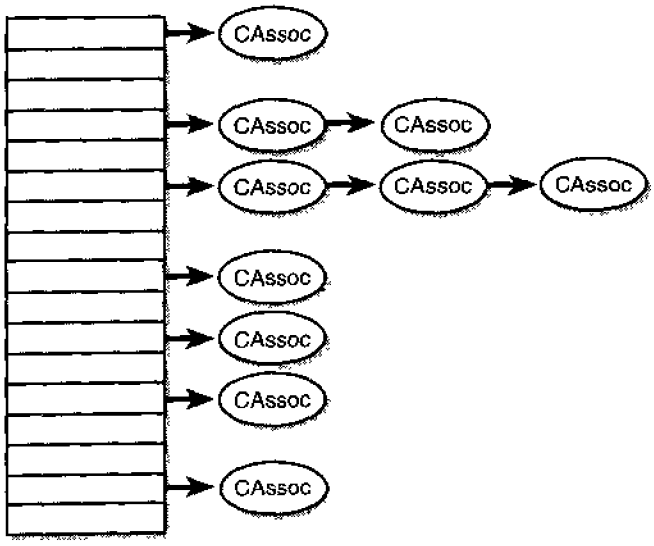

图 5-1 包含单独项目和链接列表组合的散列表

当调用映射表的 Lookup 函数时,MFC 根据输入关键字计算散列值,使用上段给出的公式将散列值转换为散列表中的索引号,并从散列表的相应位置检索得到 CAssoc 指针。在理想情况下,在要找的位置处只有一个 CAssoc 指针,而不是 CAssoc 指针链接列表。在这种情况下,只要在映射表中进行一次查找就可以找到项目,它的值可以由 CAssoc 对象得到。如果从散列表中得到的 CAssoc 指针是链接列表表头,MFC 就会查找该列表直到找到要找的关键字。一个适当创建的映射表在它的 CAssoc 结构列表中不会具有超过两个或三个的项目元素,这就意味着查找过程不会要求检查多余两个或三个项目。

5.3.3 提高查找效率

执行查找的效率取决于两个因素:

- 散列表的大小
- 散列算法根据任意(而且可能很相似)输入的关键字生成唯一散列值的能力

散列表尺寸很重要。如果映射表包含 1 000 个项目而散列表只有 17 个存放 CAssoc 指针的位置,那么最好的情况就是散列表中的每一项都存放具有 58 或 59 个 CAssoc 结构的链接列表中第一个 CAssoc 结构的地址。这种处理就会在很大程度上妨碍查找操作。散列算法也很重要,因为无论散列表可以保存多少 CAssoc 指针,如果散列算法只能生成一点儿不同的散列值(所以只有一点儿散列表索引号),查找操作效率也不会高的。

提高查找效率的最好办法就是使散列表尽可能地大从而最小化冲突发生的次数。在不同的输入关键字生成相同的散列表索引号时就会发生冲突。Microsoft 建议将散列表的尺寸设置为比映射表所包含项目的总数大百分之十到二十,这样可以合理地平衡内存消耗和查找效率。要指定散列表尺寸,可以调用映射表的 InitHashTable 函数:

```
// Assume the map will hold about 1,000 items.
map.InitHashTable (1200); // 1200 = 1000 + 20 percent
```

从统计上讲,对散列表尺寸使用质数会更好地减少冲突。所以,对于 1 000 个项目,初始化一个散列表最好按如下方式调用 InitHashTable:

```
map.InitHashTable (1201);
```

应该在给映射表添加任何项目之前调用 InitHashTable。在映射表已经包含了一个以上项目时试图调整散列表的大小会导致断言错误。

虽然 MFC 使用的用来生成散列值的算法在大多数情况下已经够用,但您还是可以用自己的算法替换它。要转换一个输入关键字,MFC 会调用名为 HashKey 的全局模板函数。对于大多数数据类型,HashKey 的实现如下:

```
AFX_INLINE UINT AFXAPI HashKey(ARG_KEY key)
{
  // default identity hash  works for most primitive values
  return ((UINT)(void*)(DWORD)key) >> 4;
}
```

对于字符串,它的实现如下:

```
UINT AFXAPI HashKey(LPCWSTR key) // Unicode strings
{
  UINT nHash = 0;
  while (*key)
    nHash = (nHash<<5) + nHash + *key++;
  return nHash;
}
```

```
UINT AFXAPI HashKey(LPCSTR key) // ANSI strings
{
  UINT nHash = 0;
  while (*key)
    nHash = (nHash<<5) + nHash + *key++;
  return nHash;
}
```

要想对特殊的数据类型实现自己的算法,只要编写特定类型 HashKey 函数即可。您可以将上面给出的字符串 HashKey 作为模型使用。

5.3.4 用 CMap 创建类型安全映射表类

正如您所猜测的,可以使用 MFC 的 CMap 模板类来为那些特定类型映射表类不支持的数据类型创建映射表。下例创建了一个 CPoint 对象的集合,关键字为 CString 并执行查找操作:

```
CMap<CString, CString&, CPoint, CPoint&> map;
map[CString(_T("Vertex1"))] = CPoint(0, 0);
map[CString(_T("Vertex2"))] = CPoint(100, 0);
map[CString(_T("Vertex3"))] = CPoint(100, 100);
map[CString(_T("Vertex4"))] = CPoint(0, 100);

CPoint point;
if (map.Lookup(CString(_T("Vertex3")), point))
  TRACE(_T("Vertex 3 = (%d, %d)\n"), point.x, point.y);
```

因为是用 CString 作为关键字,所以除非将 HashKey 覆盖实现把 CString 转换成散列值功能,否则此程序不会得到编译。下列是可能的一种实现:

```
UINT AFXAPI HashKey(CString& string)
{
  LPCTSTR key = (LPCTSTR) string;
  UINT nHash = 0;
  while (*key)
    nHash = (nHash<<5) + nHash + *key++;
  return nHash;
}
```

在把 CString 引用转换为常规字符串指针之后,此程序就可以像 MFC 的 LPCSTR/LPCWSTR HashKey 函数那样将字符串转换为散列值了。

如同 CList 类的 Find 函数一样,CMap::Lookup 使用 CompareElements 模板函数来比较元素。由于 CompareElements 使用了 == 运算符执行比较,所以对于原始数据类型和重载了 == 运算符的类其默认实现就很好用。但是,如果使用自己设计的类作为映射表中的关键字,就

必须在这些类中重载 == 运算符或者对单独的数据类型覆盖 CompareElements。可以参考本章前面"用 CList 创建类型安全列表类"中实现这两种方法的例子。

5.4　类型指针类

名字中带有 Ptr 和 Ob 的 MFC 集合类("Ptr"和"Ob"类)可以方便地实现保存一般(void)指针的容器和保存指向 MFC 对象(由 CObject 派生类创建的对象)指针的容器。使用 Ptr 和 Ob 类的问题出在它们太一般了。通常要求许多强制类型转换,这对于许多 C++ 程序员而言是令人生厌的,而且也是糟糕的编程风格。

MFC 的"类型指针类"是一组三个模板类,用来以安全的方式处理指针集合,它为保存指针而不危害类型安全问题提供了一种简便的解决办法。在表 5-4 中列出了类型安全指针类。

表 5-4　指针的集合类

类名	说　明
CTypedPtrArray	管理指针数组
CTypedPtrList	管理指针链接列表
CTypedPtrMap	管理使用指针作为项目或关键字的映射表

假设您编写了一个绘图程序,并且创建了一个名为 CLine 的类来代表屏幕上绘制的线段。每次用户绘制一条线就创建一个新的 CLine 对象。您需要一个地方来保存 CLine 指针,而且由于希望能够在集合的任何位置添加和删除指针都不会造成操作冲突,所以您决定使用链接列表。因为是从 CObject 派生的 CLine,所以 CObList 好像是个自然的选择。

CObList 可以完成任务,但是每次从列表中检索到一个 CLine 指针,都必须将它强制转换为 CLine*,因为 CObList 返回的是 CObject 指针。CTypedPtrList 是一个很好的选择,它不需要类型强制转换。下列程序代码说明了这一点:

```
CTypedPtrList<CObList, CLine*> list;

// Populate the list.
for (int i=0; i<10; i++) {
  int x = i * 10;
  CLine* pLine = new CLine(x, 0, x, 100);
  list.AddTail(pLine);
}

// Enumerate the items in the list.
POSITION pos = list.GetHeadPosition();
while (pos != NULL)
  CLine* pLine = list.GetNext(pos); // No casting!
```

当您使用 GetNext 检索一个 CLine 指针时,得到的就是一个 CLine 指针而不需要强制转换。这就是类型安全。

CTypedPtrList 和其他类型安全指针类要从第一个模板参数指定的类中实现派生。在派生类内部,类型安全成员函数封装了基类中相应的成员函数。可以在基类或派生类中调用任意个函数,但是如果它们重叠,通常就要使用函数的类型安全实现版本。一般情况下,对于其保存的指针是指向从 CObject 派生来的对象的集合,要使用 Ob 类作为基类;而对于其保存的指针是指向其他类型的对象的集合,就要使用 Ptr 类作为基类。

所有保存指针的 MFC 集合类,它们从数组、列表或映射表删除指针,但决不会删除指针所指的项目。因此,在清空一个 CLine 指针列表之前,可能会觉得也有必要删除 CLines:

```
POSITION pos = list.GetHeadPosition();
while (pos != NULL)
  delete list.GetNext (pos);
list.RemoveAll ();
```

记住:如果您不删除 CLines,没有人会为您删除。不要以为集合类会为您干这种事。

第6章　文件I/O和串行化

文件输入和输出(I/O)服务是所有操作系统的主要工作。不必惊奇，Microsoft Windows提供了各种API函数用来读、写和操作磁盘文件。MFC将这些函数和CFile类融合在面向对象的模型里。其中CFile类允许把文件当作对象，并用CFile成员函数，如Read和Write，对它们进行操作。CFile具有MFC编程人员实现低级文件I/O所需要的所有工具。

编写文件I/O代码最主要的原因是为了支持文档的存储和加载。尽管用CFile对象实现磁盘文档的读写并没有错，但是大部分MFC应用程序不会这么做，而是用CArchive对象。通过MFC实现巧妙的运算符重载，大部分数据都可以串行化为CArchive，即作为一个字节流输出，或从一个CArchive并行化为初始状态。这在句法实现上也非常简单。另一方面，如果CArchive对象挂接在CFile对象上，被串行化为该CArchive的数据就以透明方式写到磁盘上。此后，您还可以把与此文件有关的Carchive并行化，最终重建以这种方式存档的数据。

通过将文档串行化或CArchive并行化可以实现文档的存贮和加载。这是MFC文档/视图体系结构的基础构件之一。尽管目前对CArchive的了解还只限于有限的几种用法，但是可以确信：在第9章我们开始编写文档/视图应用程序时，Carchive的其他方法会使它更加容易操作。

6.1　CFile类

CFile是比较简单的类，它封装了Win32 API用来处理文件I/O的那部分。在多于25个的成员函数中有用来打开和关闭文件的函数，读和写文件数据的函数，删除和重命名文件的函数，还有检索文件信息的函数。它的公用数据成员之一，m_hFile，保存着与CFile对象相关联的文件的句柄。一个受保护的CString数据成员，名为m_strFileName，保存着文件的名称。成员函数GetFilePath、GetFileName和GetFileTitle可以用来提取整个文件名或文件名的一部分。例如：如果完整的文件名和路径名为C:\Personal\File.txt，GetFilePath返回整个字符串，GetFileName返回"File.txt"，而GetFileTitle返回"File."。

但是如果细讲这些函数，就等于忽略了对编程人员来说CFile拥有的重要功能，即用来读写磁盘文件的函数。下面几部分简要地介绍了CFile的使用方法，以及错误发生时CFile的奇特通知方式。(提示：如果您从没有用过C++异常处理，现在该是改进手工编程方式的时候了。)

6.1.1 打开、关闭和创建文件

用 CFile 打开文件有两种方法。第一种方法是构造一个没有初始化的 CFile 对象并调用 CFile::Open。下面的代码段就利用这种方法打开了一个具有读/写访问权的文件,File.txt。因为函数第一个参数中没有给出路径名,如果该文件不在当前目录下,Open 就会失败。

```
CFile file;
file.Open (_T ("File.txt"), CFile::modeReadWrite);
```

CFile::Open 返回一个 BOOL 值,表示是否成功打开文件。下面的示例利用该返回值确定文件是否成功打开:

```
CFile file;
if (file.Open (_T ("File.txt"), CFile::modeReadWrite)) {
    // It worked!
      .
      .
      .
}
```

返回值非零意味着文件打开了,零意味着文件没有打开。如果 CFile::Open 返回零,并且您想知道调用失败的原因,则创建一个 CFileException 对象并把它的地址传送到 Open 的第三个参数中。

```
CFile file;
CFileException e;
if (file.Open (_T ("File.txt"), CFile::modeReadWrite, &e)) {
    // It worked!
      .
      .
      .
}
else {
    // Open failed. Tell the user why.
    e.ReportError ();
}
```

如果 Open 失败,则它用描述失败本质的信息将 CFileException 对象初始化。ReportError 在该信息的基础上显示一条错误消息。通过检查 CFileException 的公用数据成员 m_cause,您可以找到导致失败的原因。CFileException 的文档资料包含一个完整的错误代码列表。

第二种方法是用 CFile 的构造函数打开文件。不必构造一个空的 CFile 对象并调用 Open,您可以这样创建一个 CFile 对象,并用一个语句打开文件:

```
CFile file (_T ("File.txt"), CFile::modeReadWrite);
```

如果文件不能打开，CFile 的构造函数会引发一个 CFileException。因此，利用 CFile::CFile 打开文件的代码通常使用 try 和 catch 块来俘获错误：

```
try {
    CFile file (_T ("File.txt"), CFile::modeReadWrite);
      .
      .
      .
}
catch (CFileException * e) {
    // Something went wrong.
    e -> ReportError ();
    e -> Delete ();
}
```

是否删除 MFC 发送给您的 CFileException 对象，决定权在您。这就是在处理异常后该示例调用 Delete 删除异常对象的原因。不想调用 Delete 的唯一场合是您要用 throw 重新发送异常，但这种情况很少见。

如果需要创建一个新文件，而不是打开一个现存文件，则要在 CFile::Open 或 CFile 构造函数的第二个参数中包含一个 CFile::modeCreate 标志：

```
CFile file (_T ("File.txt"), CFile::modeReadWrite|CFile::modeCreate);
```

如果用这种方法创建的文件已存在，则截去它的长度到 0。如果要创建一个不存在的文件，或要在文件存在但没有被截去时打开该文件，则也要包含一个 CFile::modeNoTruncate 标志：

```
CFile file (_T ("File.txt"), CFile::modeReadWrite|CFile::modeCreate|
    CFile::modeNoTruncate);
```

按这种方式打开文件基本上总是成功的，因为如果该文件还不存在，它能自动生成。

在默认方式下，用 CFile::Open 或 CFile::CFile 打开文件会获得该文件的独占访问权，也就是说，其他人不能再打开该文件。如果有必要，在打开文件时可以指定共享模式，明确地允许其他人访问该文件。表 6-1 中是可选的 4 种共享模式。

表 6-1　4 种共享模式

共享模式	说　明
CFile::shareDenyNone	非独占访问权式打开文件
CFile::shareDenyRead	禁止读访问权
CFile::shareDenyWrite	禁止写访问权
CFile::shareExclusive	禁止读写访问权(默认值)

另外，还可以指定表 6-2 中的三种读/写访问权之一。

表 6-2 3 种读/写访问权

访问权模式	说 明
CFile::modeReadWrite	请求读写访问权
CFile::modeRead	只请求读访问权
CFile::modeWrite	只请求写访问权

这些选项的常见用法是允许任一客户读取文件,但禁止往文件上写:

```
CFile file (_T ("File.txt"), CFile::modeRead|CFile::shareDenyWrite);
```

如果执行上面的语句时该文件已经打开,则这次调用失败,并且 CFile 会发送一个 CFileException,其 m_cause 等于 CFileException::sharingViolation。

关闭打开的文件有用两种方式。如果要显式关闭文件,则对相应的 CFile 对象调用 CFile::Close:

```
file.Close ();
```

如果您喜欢,可以用 CFile 的析构函数关闭文件。如果文件还没有关闭,类的析构函数则调用 Close。这就是说,在堆上创建的 CFile 对象在失效后会自动关闭。在下面的示例中,当程序执行到 try 块结尾的花括号时,文件关闭。

```
try {
    CFile file (_T ("File.txt"), CFile::modeReadWrite);
      .
      .
      .
    // CFile::~CFile closes the file.
}
```

有时编程人员显式调用 Close 的原因是:关闭当前处于打开状态的文件,以便用同一个 CFile 对象打开另一个文件。

6.1.2 读和写

可以用 CFile::Read 读一个具有读访问权的打开文件。可以用 CFile::Write 写一个具有写访问权的打开文件。下面的示例分配了一个 4KB 的文件 I/O 缓冲区并一次读取文件 4KB 内容。为了使程序清晰,省略错误检查。

```
BYTE buffer[0x1000];
CFile file (_T ("File.txt"), CFile::modeRead);
DWORD dwBytesRemaining = file.GetLength ();

while (dwBytesRemaining) {
```

```
    UINT nBytesRead = file.Read(buffer, sizeof(buffer));
    dwBytesRemaining -= nBytesRead;
}
```

未读字节数保存在 dwBytesRemaining 中，它的初始值为 CFile::GetLength 返回的文件尺寸。每次调用 Read 后，dwBytesRemaining 都要减去从文件读取的字节数(nBytesRead)。执行 while 循环直到 dwBytesRemaining 变成 0。

下面的示例是在前段代码的基础上发展的，它调用 ::CharLowerBuff 将从文件中读取的所有大写字符转化成小写，并调用 CFile::Write 把转化后的正文写回文件。为了使程序清晰，再次省略错误检查。

```
BYTE buffer[0x1000];
CFile file(_T("File.txt"), CFile::modeReadWrite);
DWORD dwBytesRemaining = file.GetLength();

while(dwBytesRemaining) {
    DWORD dwPosition = file.GetPosition();
    UINT nBytesRead = file.Read(buffer, sizeof(buffer));
    ::CharLowerBuff((LPTSTR)buffer, nBytesRead);
    file.Seek(dwPosition, CFile::begin);
    file.Write(buffer, nBytesRead);
    dwBytesRemaining -= nBytesRead;
}
```

该示例调用 CFile 函数 GetPosition 和 Seek 操作文件指针——文件中的偏移值，即执行下一个读或写的位置——使修改后的数据覆盖原文件。Seek 用第二个参数确定第一个参数中的字节偏移值是相对于文件起始位置 (CFile::begin)、结束位置 (CFile::end)还是当前位置(CFile::current)。如果要快速定位到文件的开始位置或结束位置，可以调用 CFile::SeekToBegin 或 CFile::SeekToEnd。

如果在文件 I/O 过程中有错误发生，Read、Write 和其他 CFile 函数就会发送一个 CFileException。CFileException::m_cause 告诉您引发错误的原因。例如：试图往已满的磁盘上写文件会引发 CFileException，其 m_cause 等于 CFileException::diskFull。试图在文件范围之外读取数据会引发 CFileException，其 m_cause 等于 CFileException::endOfFile。下面就是将小写正文转化为大写的例程，其中还包括检查代码：

```
BYTE buffer[0x1000];
try {
    CFile file(_T("File.txt"), CFile::modeReadWrite);
    DWORD dwBytesRemaining = file.GetLength();
    while(dwBytesRemaining) {
        DWORD dwPosition = file.GetPosition();
        UINT nBytesRead = file.Read(buffer, sizeof(buffer));
```

```
            ::CharLowerBuff ((LPTSTR)buffer, nBytesRead);
            file.Seek (dwPosition, CFile::begin);
            file.Write (buffer, nBytesRead);
            dwBytesRemaining -= nBytesRead;
        }
    }
    catch (CFileException* e) {
        e->ReportError ();
        e->Delete ();
    }
```

如果您不捕获 CFile 成员函数发送的异常,MFC 会替您捕获它们。MFC 给未处理的异常配有默认处理程序,该程序调用 ReportError 显示一条描述错误的消息。然而,一般情况下,最好捕获文件 I/O 异常,防止代码的关键部分被遗漏。

6.1.3 CFile 派生类

CFile 是整个 MFC 类家族的根类。其家族成员及彼此间的关系见图 6-1。

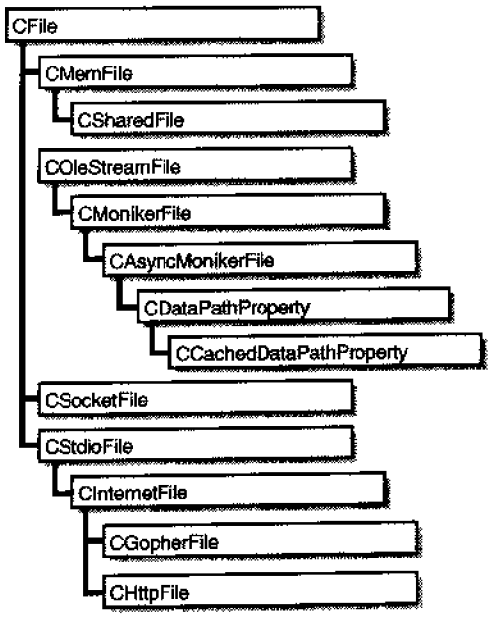

图 6-1 CFile 家族

CFile 家族某些成员的作用仅在于给非文件媒体提供文件式界面。例如:CMemFile 和 CSharedFile 允许内存块可以像文件那样读和写。在第 19 章将介绍到的 MFC 函数,COleDataObject::GetFileData,根据这个便利的抽象,可以允许 OLE 放下剪贴板的目标和用户,从

而调用 CFile::Read 检索内存中的数据信息。CSocketFile 对 TCP/IP 套接字进行了类似的抽象。MFC 编程人员有时把 CSocketFile 对象放在 CSocket 对象和 CArchive 对象之间，这样就可以用 C++ 的插入和提取运算符对打开的套接字进行读写了。COleStreamFile 使流对象，即表示字节流的 COM 对象，看上去像一个普通文件。对于支持对象链接和嵌入（OLE）的 MFC 应用程序，这种方法非常重要。

CStdioFile 将编程接口简化为文本文件。它在由 CFile 继承来的类中只添加了两个成员函数：一个用来读取正文行的 ReadString 函数，一个用来输出正文行的 WriteString 函数。对 CStdioFile 来说，一行正文就是由回车符和换行符（0x0D 和 0x0A）定界的字符串。ReadString 读取当前文件位置到下一个回车符间的所有数据，可以包含或不包含回车符。WriteString 输出正文字符串，并还在文件中写一个回车符和换行符。下面的代码段打开一个文本文件 File.txt，并将它的内容转放在调试输出窗口：

```
try {
    CString string;
    CStdioFile file (_T ("File.txt"), CFile::modeRead);
    while (file.ReadString (string))
        TRACE (_T ("%s\n"), string);
}
catch (CFileException * e) {
     e -> ReportError ();
     e -> Delete ();
}
```

同 Read 和 Write 一样，如果有错误发生，使得 ReadString 和 WriteString 无法执行任务，则这两个函数引发异常。

6.1.4　枚举文件和文件夹

CFile 包含一对静态的成员函数，Rename 和 Remove。可以用这两个函数重命名和删除文件。但是，它不包含用来枚举文件和文件夹的函数。因此，您只好求助于 Windows API。

枚举文件和文件夹的关键在于一对 API 函数，::FindFirstFile 和 ::FindNextFile。如果给定一个绝对或相对文件名（例如："C:*.*"或"*.*"），::FindFirstFile 打开一个"查找句柄"，并把它返回给调用者。::FindNextFile 利用该句柄枚举文件系统对象。常见的方法是：枚举一开始，先调用::FindFirstFile，然后反复调用::FindNextFile 直到枚举结束。每次成功地调用::FindFirstFile 或::FindNextFile（也就是说，调用::FindFirstFile 时，返回值是 INVALID_HANDLE_VALUE 外的任意值；或者调用::FindNextFile 时，返回值是个非 NULL 值）都会在 WIN32_FIND_DATA 结构中填充文件或目录信息。WIN32_FIND_DATA 是这样用 ANSI 代码定义的：

```
typedef struct_WIN32_FIND_DATAA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    DWORD dwReserved0;
    DWORD dwReserved1;
    CHAR cFileName[MAX_PATH];
    CHAR cAlternateFileName[14];
    } WIN32_FIND_DATAA;
typedef WIN32_FIND_DATAA WIN32_FIND_DATA;
```

如果要确定由 WIN32_FIND_DATA 结构表示的这一项是文件还是目录,检测 dwFileAttributes 字段的 FILE_ATTRIBUTE_DIRECTORY 标志:

```
if (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
      // It's a directory.
}
else {
    // It's a file.
}
```

cFileName 和 cAlternateFileName 字段保留着文件或目录名。cFileName 包含长的文件名,cAlternateFileName 包含短的文件名。当枚举完成时,您应该关闭由::FindFirstFile 和::FindClose 返回的任一句柄。

作为示范,下面的例程枚举了当前目录下的所有文件,并把它们的文件名写到调试输出窗口:

```
WIN32_FIND_DATA fd;
HANDLE hFind = ::FindFirstFile (_T ("*.*"), &fd);

if (hFind != INVALID_HANDLE_VALUE) {
     do {
        if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
              TRACE (_T ("%s\n"), fd.cFileName);
     } while (::FindNextFile (hFind, &fd));
     ::FindClose (hFind);
}
```

如果要枚举当前目录下的所有子目录,则需要稍微改动一下:

```
WIN32_FIND_DATA fd;
HANDLE hFind = ::FindFirstFile (_T ("*.*"), &fd);
```

```
if (hFind != INVALID_HANDLE_VALUE) {
      do {
          if (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
              TRACE (_T ("%s\n"), fd.cFileName);
      } while (::FindNextFile (hFind, &fd));
      ::FindClose (hFind);
}
```

更有趣的问题是：如何把给定目录“和它的子目录”下的所有目录枚举出来。下面的函数枚举出当前目录和其子目录中的所有目录，并把每个目录的名称写到调试输出窗口。秘诀是什么？一旦遇到目录，EnumerateFolders 便进入该目录并递归调用自身。

```
void EnumerateFolders ()
{
   WIN32_FIND_DATA fd;
   HANDLE hFind = ::FindFirstFile (_T ("*.*"), &fd);

   if (hFind != INVALID_HANDLE_VALUE) {
         do {
             if (fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
                 CString name = fd.cFileName;
                 if (name != _T (".") && name != _T (".."))) {
                     TRACE (_T ("%s\n"), fd.cFileName);
                     ::SetCurrentDirectory (fd.cFileName);
                     EnumerateFolders ();
                     ::SetCurrentDirectory (_T (".."));
                 }
             }
         } while (::FindNextFile (hFind, &fd));
         ::FindClose (hFind);
   }
}
```

如果要使用该函数，先导航到枚举起始时所在的目录，然后调用 EnumerateFolders。下面的语句枚举了驱动器 C 中的所有目录：

```
::SetCurrentDirectory (_T ("C:\\"));
EnumerateFolders ();
```

在第 10 章我们将用类似的技巧给树图填充，图中各项代表某驱动器上的所有文件夹。

6.2　串行化和 CArchive 类

尽管 MFC 的 CFile 类极大地简化了文件数据的读写，但是大部分 MFC 应用程序并不直

接和 CFile 对象发生相互作用。实际上,它们借助于 CArchive 对象完成读写工作,而 CArchive 对象又转而利用 CFile 函数实现文件 I/O。MFC 重载 << 和 >> 运算符,这两个运算符和 CArchive 一起简化了串行化和并行化过程。串行化和并行化的根本目的在于把应用程序持久性数据保存到磁盘上或再从磁盘读回需要的数据。

串行化是 MFC 编程中的一个重要概念,因为在文档/视图应用程序中打开并保存文档是 MFC 的基本功能。同将在第 9 章中学到的一样,在使用文档/视图应用程序时,如果在应用程序的 File 菜单中选中 Open 或 Save,MFC 就会打开文件进行读或写,并传递给应用程序一个指向 CArchive 对象的引用。接着,应用程序又将持久性数据串行化为档案,或把档案并行化为数据,这样就把一个完整的文档保存在磁盘上或重新把文档读取出来了。如果文档的持久性数据完全由基本数据类型或可串行化对象组成,那么通常只需几行代码就可以实现串行化。与此呈鲜明对照的是:如果应用程序向用户询问文件名,打开文件并自己完成文件 I/O,则需要成千上万行语句。

6.2.1 串行化基础

假定一个 CFile 对象,名为 file,代表一个打开的文件,该文件具有写访问权,并且您想在文件上写一对整数,名为 a 和 b。为了实现这个要求,一种方法是对每一个整数都调用 CFile::Write:

```
file.Write (&a, sizeof (a));
file.Write (&b, sizeof (b));
```

另一种方法是创建一个 CArchive 对象,并把它与该 CFile 对象关联起来,然后运用 << 运算符把整数串行化到档案中:

```
CArchive ar (&file, CArchive::store);
ar << a << b;
```

CArchive 对象也可以用来读取数据。假定 file 再次代表一个打开的文件,并且该文件具有读访问权,下面的小代码段将 CArchive 对象挂接到文件上,并从文件中读取整数,或将整数"并行化":

```
CArchive ar (&file, CArchive::load);
ar >> a >> b;
```

MFC 允许多种基本数据类型以这种方式串行化,包括 BYTE、WORD、LONG、DWORD、float、double、int、unsigned int、short 和 char。

MFC 还重载 << 和 >> 运算符,以便串行化或并行化 CString 和其他某些由 MFC 类表示的非基本数据类型。如果 string 是一个 CString 对象,ar 是一个 CArchive 对象,可以按如下方式把字符串写入档案:

```
ar << string;
```

将上面的运算符掉个方向，就可以从档案中读取字符串了。

```
ar >> string;
```

可以用这种方式串行化的类包括：CString、CTime、CTimeSpan、COleVariant、COleCurrency、COleDateTime、COleDateTimeSpan、CSize、CPoint 和 CRect。类型为 SIZE、POINT 和 RECT 的结构也可以串行化。

或许 MFC 串行化机制最强大的一面是：您能够创建自己的可串行化类，使它们与 CArchive 的插入和提取运算符一起工作，并且为了使这些类工作，您也不必自己重载任何运算符。什么原因呢？因为 MFC 为指向 CObject 派生类的实例的指针重载了 << 和 >> 运算符。

作为示范，假定您已经编写了一个绘图程序，它给出用户用 CLine 类实例画的线。再假定 CLine 是直接或间接由 CObject 派生来的可串行化类。如果 pLines 是 CLine 指针数组，nCount 是一个整型数，保存数组中指针的个数，而 ar 是一个 CArchive 对象，您可以按下面的方式将每个 CLine 存档，并同时对 CLines 进行记数：

```
ar << nCount;
for (int i = 0; i < nCount; i++)
    ar << pLines[i];
```

相反地，也可以根据档案中的信息重新创建 CLines，并用下面的语句将 pLines 初始化为 CLine 指针：

```
ar >> nCount;
for (int i = 0; i < nCount; i++)
    ar >> pLines[i];
```

怎样编写可串行化类，如 CLine 呢？这很容易，下节就会介绍到。

如果数据串行化或并行化时有错误发生，MFC 会发送一个异常。异常的类型取决于错误的性质。如果由于内存不足，串行化请求失败（例如：如果内存太少，不足以创建一个正在并行化的对象的实例），MFC 会发送一个 CMemoryException。如果由于文件 I/O 出错，请求失败，则 MFC 发送一个 CFileException。如果发生了其他错误，MFC 会发送一个 CArchiveException。如果您喜欢，您可以给这些类型的异常提供“捕捉”处理程序，制定错误发生时自己特有的处理方法。

6.2.2　编写可串行化类

如果一个对象支持串行化，那么它一定是可串行化类的实例。您可以按照以下五个步骤编写可串行化类：

1. 直接或间接得到 CObject 的派生类。

2. 在类的说明中写入 MFC 的 DECLARE_SERIAL 宏。DECLARE_SERIAL 只接收一个参数：类名。
3. 重载基本类的 Serialize 函数，并串行化派生类的数据成员。
4. 如果派生类没有默认的构造函数(该函数没有参数)，则添加一个。因为对象并行化时，MFC 要用默认构造函数在浮动标签上创建对象，并用从档案取回的值设置对象数据成员的初始值，所以这一步是非常必要的。
5. 在类的实现中写入 MFC 的 IMPLEMENT_SERIAL 宏。IMPLEMENT_SERIAL 宏接受三个参数：类名，基本类名和模式号。“模式号”是一个整型值，等于版本号。只要修改了类的串行化数据格式，模式号也要随之改变。至于如何分配可串行化类的版本号将在下节介绍。

假定您编写了一个简单的类来代表线，名为 CLine。该类有两个 CPoint 数据成员，它们存储着线的两个端点，并且您想添加串行化支持。最初，类的声明是这样的：

```
class CLine
{
protected:
     CPoint m_ptFrom;
     CPoint m_ptTo;
public:
    CLine (CPoint from, CPoint to) { m_ptFrom = from; m_ptTo = to; }
};
```

做到使该类可串行化非常容易。下面是添加串行化支持后的程序段：

```
class CLine : public CObject
{
DECLARE_SERIAL (CLine)

protected:
     CPoint m_ptFrom;
     CPoint m_ptTo;

public:
     CLine () { } // Required!
     CLine (CPoint from, CPoint to) { m_ptFrom = from; m_ptTo = to; }
     void Serialize (CArchive& ar);
};
The Serialize function looks like this:

void CLine::Serialize (CArchive& ar)
{
   CObject::Serialize (ar);
   if (ar.IsStoring ())
```

```
        ar << m_ptFrom << m_ptTo;
    else // Loading, not storing
        ar >> m_ptFrom >> m_ptTo;
}
```

在实现类的过程中出现语句

```
IMPLEMENT_SERIAL (CLine, CObject, 1)
```

经过这些修改，类就可以串行化了。目前版本号为 1，如果后来又给 CLine 添加了一个持久性数据成员，则要把版本号增加到 2，这样主结构就能根据程序的不同版本区别串行化到磁盘的 CLine 对象了。否则，磁盘上版本为 1 的 CLine 就可能被读入内存中版本为 2 的 CLine，从而可能造成严重损失。

该类实例被要求串行化或并行化时，MFC 调用实例的 CLine::Serialize 函数。在将自己的数据成员串行化之前，CLine::Serialize 调用 CObject::Serialize 串行化基本类的数据成员。在这个示例中，基本类的 Serialize 函数并不起作用，但是如果您编写的类是间接由 CObject 派生来的，那么情况就可能不同了。基本类调用返回之后，CLine::Serialize 调用 CArchive::IsStoring 决定数据流的方向。如果返回值为非 0 值，则数据串行化到档案中；如果为 0，则数据被串行化读出。CLine::Serialize 根据该返回值决定是用 << 运算符向档案中写数据呢，还是用 >> 运算符从档案中读取数据。

6.2.3　给可串行化类分配版本号：可配置版本模式

编写可串行化类时，MFC 用您指定的模式号制定一个粗略的版本控制方式。在向档案写数据时，MFC 用模式号标记该类的实例；而在读回数据时，MFC 将档案中记录的模式号和应用程序中使用着的该类对象的模式号做比较，如果两模式号不匹配，则 MFC 发送一个 CArchiveException，其 m_cause 等于 CArchiveException::badSchema。没有得到处理的该类异常会促使 MFC 显示一个消息框，提示“文件格式不对”。如果每次修改对象的串行化存贮格式时都能做到增加模式号，那么就不怕这种无心的操作—试图把磁盘中存的老版本对象读入内存里的新版本对象了。

有一个问题经常会突然在使用了可串行化类的应用程序中出现，这就是向下兼容性。换句话说，就是如何并行化在老版本应用程序中创建的对象。如果对象的持久存贮格式随应用程序版本的更新发生了变化，这时您可能希望新版本应用程序对两种格式都能读。但是一旦 MFC 发现不配套的模式号，它将发送异常。鉴于 MFC 的结构特点，最好按照 MFC 的方式处理异常并中止串行化过程。

可视化模式也就此产生了。可视化模式只是包含 VERSIONABLE_SCHEMA 标志的模式号。标志告诉 MFC 应用程序针对某一类能够处理多种串行化的数据格式。这种模式禁止 CArchiveException，并允许应用程序对不同的模式号有判断地响应。使用了可视化模式的应

用程序可以提供用户希望的向下兼容性。

如果要编写一个只有 MFC 可视化模式支持的可串行化类,一般需要两步:

1. 将 IMPLEMENT_SERIAL 宏中的模式号与值 VERSIONABLE_SCHEMA 相或。
2. 如果从档案加载对象时需要调用 CArchive::GetObjectSchema,则要修改类的 Serialize 函数,并相应地调整其并行化例程。GetObjectSchema 返回要进行并行化对象的模式号。

调用 GetObjectSchema 时要注意几个规则。首先,只有对象在被并行化时才能调用。其次,必须在读取档案对象数据之前调用。再者,它只能被调用一次。如果 GetObjectSchema 在调用 Serialize 前后被调用了两次,则返回 -1。

假定在应用程序的第 2 版本中,您要修改 CLine 类,想添加一个成员变量,用来保存线的颜色。下面是修改后的类的声明:

```
class CLine : public CObject
{
DECLARE_SERIAL (CLine)

protected:
     CPoint m_ptFrom;
     CPoint m_ptTo;
     COLORREF m_clrLine; // Line color (new in version 2)

public:
    CLine () {}
    CLine (CPoint from, CPoint to, COLORREF color)
        { m_ptFrom = from; m_ptTo = to; m_clrLine = color; }
    void Serialize (CArchive& ar);
};
```

因为线的颜色是持久性属性(也就是说,保存到档案中的红线在读出时依旧是红的。),所以您想修改 CLine::Serialize,使它在串行化 m_ptFrom 和 m_ptTo.to 之外还能串行化m_clrLine。这意味着要把 CLine 的模式号增加到 2。使用原类时按以下方式调用 MFC 的 IMPLEMENT_SERIAL宏:

```
IMPLEMENT_SERIAL (CLine, CObject, 1)
```

但是在修改后的类中,应该这样调用 IMPLEMENT_SERIAL :

```
IMPLEMENT_SERIAL (CLine, CObject, 2|VERSIONABLE_SCHEMA)
```

更新后的程序在读取 CLine 对象时,如果对象的模式号是 1,MFC 也不会发送 CArchive 异常,因为模式号中有 VERSIONABLE_SCHEMA 标志。但是它会了解到:由于模式号从 1 变为 2,两个模式实际上是不同的。

现在工作只做了一半。最后一步是修改 CLine::Serialize，使它根据 GetObjectSchema 不同的返回值并行化 CLine。原 Serialize 函数如下：

```
void CLine::Serialize (CArchive& ar)
{
   CObject::Serialize (ar);
   if (ar.IsStoring ())
         ar << m_ptFrom << m_ptTo;
   else // Loading, not storing
        ar >> m_ptFrom >> m_ptTo;
}
```

新函数如下：

```
void CLine::Serialize (CArchive& ar)
{
   CObject::Serialize (ar);
   if (ar.IsStoring ())
         ar << m_ptFrom << m_ptTo << m_clrLine;
   else {
         UINT nSchema = ar.GetObjectSchema ();
         switch (nSchema) {
         case 1: // Version 1 CLine
              ar >> m_ptFrom >> m_ptTo;
              m_clrLine = RGB (0, 0, 0); // Default color
              break;
         case 2: // Version 2 CLine
              ar >> m_ptFrom >> m_ptTo >> m_clrLine;
              break;
         default: // Unknown version
              AfxThrowArchiveException (CArchiveException::badSchema);
              break;
         }
   }
}
```

明白它是怎样工作的了吗？CLine 对象写到档案上时，它的格式总是 CLine 的第 2 个版本。但是读取 CLine 时，根据 GetObjectSchema 返回值的不同，它又被当作 CLine 版本 1 或版本 2 读回。如果模式号为 1，则对象按老方式读取，并把 m_clrLine 设置为默认值。如果模式号为 2，则对象所有数据成员，包括 m_clrLine，都要从档案中读取出来。其他模式号会导致 CArchiveException，表示不能识别版本号（如果发生异常，可能是因为程序错了或档案毁坏了。）如果将来还要修改 CLine，则要把模式号增加到 3 并给新模式添加一个 case 程序段。

6.2.4　串行化工作过程

看看实际中数据进出档案的串行化和并行化过程,可以帮助您进一步了解 MFC 的运作过程及其体系结构。MFC 通过直接把原始数据类型如 int 和 DWORD 复制到档案中从而实现了该类型数据的串行化。为便于理解,下列引自 MFC 源代码文件 Arccore.cpp 中的程序,说明了对于 DWORD,CArchive 中插入运算符的使用方法:

```
CArchive& CArchive::operator << (DWORD dw)
{
    if (m_lpBufCur + sizeof(DWORD) > m_lpBufMax)
        Flush();

    if (!(m_nMode & bNoByteSwap))
        _AfxByteSwap(dw, m_lpBufCur);
    else
        *(DWORD*)m_lpBufCur = dw;

    m_lpBufCur += sizeof(DWORD);
    return *this;
}
```

为了提高执行效率,CArchive 对象把接收到的数据保存在内部缓冲区。m_lpBufCur 指向该缓冲区中数据的当前位置。如果缓冲区太满而不能再保存一个 DWORD,它就会在 DWORD 复制给它之前被清空。对于挂接在 CFile 上的 CArchive 对象,CArchive::Flush 会将当前缓冲区中的内容写入文件。

CString、CRect 以及其他从 MFC 类中生成的非原始数据类型以不同的方式串行化。例如:对于串行化 CString,先输出字符数后输出字符本身。可以用 CArchive::Write 执行写入操作。下列从 Arccore.cpp 中节选的程序段说明了一个包含不足 255 个字符的 CString 是如何被串行化的:

```
CArchive& AFXAPI operator << (CArchive& ar, const CString& string)
{
    .
    .
    .
   if (string.GetData()->nDataLength < 255)
   {
       ar << (BYTE)string.GetData()->nDataLength;
   }
    .
    .
    .
   ar.Write(string.m_pchData,
```

```
        string.GetData()->nDataLength * sizeof(TCHAR));
    return ar;
}
```

CArchive::Write 将指定量的数据复制到档案的内部缓冲区中，并在必要的时候为防止溢出而清空缓冲区。偶尔，如果用 << 运算符串行化给档案的 CString 包含 Unicode 字符，MFC 就会在字符数之前给档案写入一个特殊的 3 位标记。这就使得 MFC 可以标识串行化的字符串字符的类型，以便在字符串从档案中并行化时在必要的情况下将字符转换为用户希望的格式。换句话说，由 Unicode 应用程序串行化一个字符串而让 ANSI 应用程序并行化它，或者相反，都是完全可以的。

在给档案串行化 CObject 指针时可以看到更有趣的情况。下面是 Afx.inl 中相关的代码：

```
_AFX_INLINE CArchive& AFXAPI operator<<(CArchive& ar,
    const CObject* pOb)
    { ar.WriteObject(pOb); return ar; }
```

您可以看到，<< 运算符调用了 CArchive::WriteObject 并给它传递了出现在插入符右边的指针，例如，

```
ar << pLine;
```

中的 pLine。WriteObject 最终会调用对象的 Serialize 函数来串行化对象的数据成员，但在此之前它要给档案写入附加信息，用来标识所创建对象的类。

例如：假设要串行化的对象是 CLine 的实例。第一次给档案串行化 CLine 时，WriteObject 将在档案中插入一个“新类标记”，是 16 位整型数，其值为 -1 或 0xFFFF；接着是对象的 16 位模式编号，一个 16 位值表示类名的字符数；最后才是类名自身。WriteObject 然后调用 CLine 的 Serialize 函数来串行化 CLine 的数据成员。

如果给档案写第二个 CLine，WriteObject 操作就不同了。在给档案写入新类标记时，WriteObject 将给内存中的数据库（实际上是 CMapPtrToPtr 的实例）添加一个类名并给该类分配一个唯一的标识符（实际上是数据库的索引号）。如果以前没有其他类被写入档案中，那么第一个写到磁盘上的 CLine 会得到索引号 1。在请求给档案写入第二个 CLine 时，WriteObject 先检查数据库，看看是否有 CLine 记录，为了不给档案写入多余的信息，它会写入一个由带有“旧类标记”（0x8000）的类索引号 ORed 组成的 16 位值。然后像以前一样调用 CLine 的 Serialize 函数。这样写入档案中的类的第一个实例由新类标记、模式编号以及类名标记；后来的实例将由一个 16 位值标记，其中低 15 位标识上此记录的模式编号和类名。

图 6-2 给出了档案的十六进制转储描述，其中档案包含两个已串行化版本 1 的 CLine。用下列程序片段将 CLine 写入档案中：

```
// Create two CLines and initialize an array of pointers.
CLine line1 (CPoint (0, 0), CPoint (50, 50));
```

```
CLine line2 (CPoint (50, 50), CPoint (100, 0));
CLine * pLines[2] = { &line1, &line2 };
int nCount = 2;

// Serialize the CLines and the CLine count.
ar << nCount;
for (int i = 0; i < nCount; i ++ )
    ar << pLines[i];
```

分解十六进制转储使得每行都代表档案的一个组成成分。为便于说明我给每行都编了号。Line 1 包含以下语句执行时写入档案中的对象数(2)。

```
ar << nCount;
```

Line 2 包含由 WriteObject 写入的定义 CLine 类的信息。第一个 16 位值是新类标记;第二个是类的模式编号(1);第三个保存了类名的长度(5)。第二行中的最后 5 位保存着类名("CLine")。

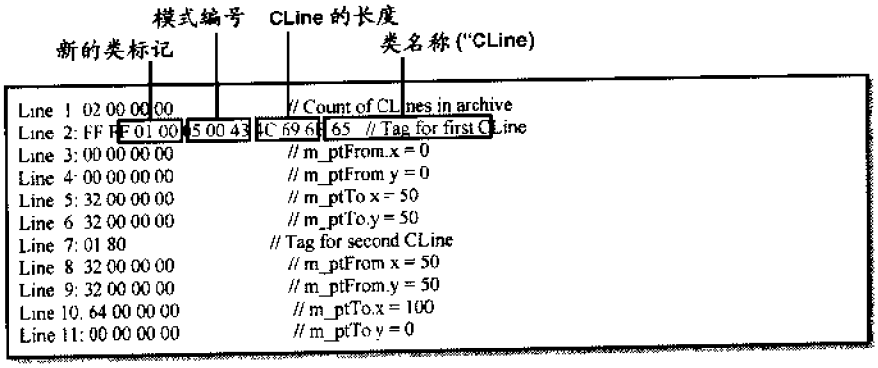

图 6-2　包含两个 CLine 的档案的十六进制转储

紧跟着类信息的第 3 到第 6 行是第一个串行化 CLine 的四个 32 位值,它们按顺序指定了 CLine 的 m_ptFrom 数据成员的 x 值,y 值以及 m_ptTo 的 x 值和 y 值。与此相似第 8 到第 11 行是有关第二个 CLine 的信息,而第 7 行是一个 16 位标记,标识了以后串行化 CLine 的数据。CLine 类的索引号是 1,因为它是第一个被加入档案的。16 位值 0x8001 是带有旧类标记的类索引号 ORed。

如果到此为止就好办了。将数据写入档案的过程并不是很难理解。但问题是将 CLines 从档案中读出时情况又是怎样的呢?

假定用下列程序并行化 CLine :

```
int nCount;
ar >> nCount;
CLine * pLines = new CLine[nCount];
```

```
for (int i = 0; i < nCount; i ++ )
    ar >> pLines[i];
```

在语句

```
ar >> nCount;
```

被执行时，CArchive 进入档案中检索 4 个字节，并将它们复制给 nCount。这样就为从档案中检索 CLine 的 for 循环作了准备。每次语句

```
ar >> pLines[i];
```

执行时，>> 运算符都将调用 CArchive::ReadObject 并传递一个 NULL 指针。下面是 Afx.inl 中的相关程序代码：

```
_AFX_INLINE CArchive& AFXAPI operator >> (CArchive& ar, CObject * & pOb)
    { pOb = ar.ReadObject(NULL); return ar; }
_AFX_INLINE CArchive& AFXAPI operator >> (CArchive& ar,
    const CObject * & pOb)
    { pOb = ar.ReadObject(NULL); return ar; }
```

ReadObject 调用另一个 CArchive 函数 ReadClass 来确定即将并行化的对象种类。在第一次循环过程中，ReadClass 从档案中读出一个字，由于是新类标记，它继续从档案中读出模式编号和类名。然后 ReadClass 比较从档案中得到的模式编号和保存在 CRuntimeClass 结构中的模式编号，该结构与检索得到的类关联。（DECLARE_SERIAL 和 IMPLEMENT_SERIAL 宏创建一个静态 CRuntimeClass 结构，其中包含有关类的重要信息，包括类名和模式编号。MFC 维持着一个 CRuntimeClass 结构的链接列表，可以用来查找到特定类的运行时信息。）如果模式编号相同，ReadClass 就给 ReadObject 返回 CRuntimeClass 指针，接下来 ReadObject 将调用通过 CRuntimeClass 指针 CreateObject 来创建一个类的新实例，然后调用对象的 Serialize 函数从档案中给对象数据成员加载数据。由 ReadClass 返回的指向新的类实例的指针被保存在调用者指定的地方，本例中是 pLines[i]的地址处。

在从档案中读出类信息过程中，ReadObject 也像 WriteObject 那样在内存中创建了一个数据库。在从档案中读取第二个 CLine 时，它前面的 0x8001 标记告诉 ReadClass 可以从数据库中获得 ReadObject 要求的 CRuntimeClass 指针。

如果一切顺利的话，串行化过程基本就是这样。我跳过了许多细节内容，如：MFC 执行的多种出错检查，对 NULL 对象指针的特殊处理，以及同一个对象的多个引用等。

如果从档案中读出的模式编号与保存在相应的 CRuntimeClass 中的模式编号不匹配怎么办？输入不同版本的模式。MFC 首先检查保存在 CRuntimeClass 内模式编号中的 VERSIONABLE_SCHEMA 标志。如果标志不存在，MFC 产生 CArchiveException。到此为止，串行

化过程就完了。这时除了显示一个出错消息以外无事可做,如果您自己不处理异常事件的话,MFC 就会代您做。但是如果 VERSIONABLE_SCHEMA 标志存在,MFC 就会跳过 AfxThrowArchiveException,将模式编号保存在应用程序通过调用 GetObjectSchema 可以检索到的地方。这就说明了为什么 VERSIONABLE_SCHEMA 和 GetObjectSchema 是对可串行化类进行成功版本编号的关键。

6.2.5 串行化 CObject

在结束本章之前,我对有关串行化 CObject 还有些建议。MFC 为 CObject 指针重载了 CArchive 的插入和提取运算符,但对 CObject 没有重载。这就意味着下面的语句有效:

```
CLine* pLine = new CLine(CPoint(0, 0), CPoint(100, 50));
ar << pLine;
```

而以下语句无效:

```
CLine line(CPoint(0, 0), CPoint(100, 50));
ar << line;
```

也就是说,CObjects 可以用指针而不能用值串行化。通常这并不是问题,但如果您要编写可串行化的类而该类又使用其他可串行化的类作为内部数据成员时,串行化这些数据成员就出现麻烦了。

通过值而不是指针串行化 CObject 的一种方法是按下列程序代码进行串行化和并行化:

```
// Serialize.
CLine line(CPoint(0, 0), CPoint(100, 50));
ar << &line;

// Deserialize.
CLine* pLine;
ar >> pLine;
CLine line = *pLine; // Assumes CLine has a copy constructor.
delete pLine;
```

但是更通用的方法是直接调用其他类的 Serialize 函数,如下:

```
// Serialize.
CLine line(CPoint(0, 0), CPoint(100, 50));
line.Serialize(ar);

// Deserialize.
CLine line;
line.Serialize(ar);
```

虽然直接调用 Serialize 完全合法，但您应该意识到这样做就意味着对于要串行化的对象没有可编版本号的模式。在使用 << 运算符串行化一个对象指针时，MFC 给档案写入对象的模式编号，而如果直接调用 Serialize，就不会这样。如果调用 GetObjectSchema 来检索没有记录模式的对象的模式编号，它会返回 -1 并且并行化处理的结果将依赖于 Serialize 处理意外模式编号的能力。

第7章 控 件

在几乎所有成功的 Microsoft Windows 应用程序中都可以找到一种组成成份，那就是控件。控件是一种特殊的窗口，用来将信息传送给用户或获取用户输入。大多数控件出现在对话框中，但它们在顶层窗口和其他非对话框窗口也可以工作。按钮就是控件的一个例子，还有编辑控件——用来输入和编辑文本的矩形。

控件减少了 Windows 程序设计中乏味的工作并通过集中使用通用用户界面元素促进了用户界面的一致性。控件是预先包装好了的对象，带有它自己完整的窗口处理过程。一个使用按钮控件的应用程序不必去在屏幕上画一个按钮而且也不必在按钮被单击时去处理鼠标消息。相反，它用一个简单的函数调用来生成按钮并在按钮被按下时接收通知。控件的 WM_PAINT 处理程序在屏幕上绘制按钮，控件中的其他消息处理程序将用户输入转换为通知消息。

由于它们和其他窗口所具有的父子关系，控件有时被称为子窗口控件。与没有父亲的顶层窗口不同，控件是属于其他窗口的子窗口。在父亲窗口移动时子窗口也移动，父亲窗口被销毁时它也被自动销毁，它被剖在了父亲窗口的矩形框上。当控件传送一个表明有输入事件发生的通知消息时，它的父亲就是消息的接收者。

Windows 的当前版本拥有 20 多种控件。其中 6 种，我们称为传统控件，在 Windows 第一版本时已经出现在 User.exe 中得到实现。其他控件，一起被称为通用控件，与 Windows 相比还较新（大多数首先出现在 Windows 95 中），在 Comctl32.dll 中得到实现。本章将介绍传统控件以及封装它们的 MFC 类。通用控件将在第 16 章讨论。

7.1 传统控件

Windows 通过注册 6 个预定义的 WNDCLASS 使控件所属的应用程序可以使用它们。表 7-1 给出了控件类型，它们的 WNDCLASS，以及相应的 MFC 类。

表 7-1 传 统 控 件

控件类型	WNDCLASS	MFC 类
按钮	"BUTTON"	CButton
列表框	"LISTBOX"	CListBox
编辑控件	"EDIT"	CEdit

续表

控件类型	WNDCLASS	MFC 类
组合框	"COMBOBOX"	CComboBox
滚动条	"SCROLLBAR"	CScrollBar
静态控件	"STATIC"	CStatic

初始化一个 MFC 控件类并调用所生成对象的 Create 函数就可以创建一个控件。如果 m_wndPushButton 是 CButton 对象，语句

```
m_wndPushButton.Create( _T("Start"), WS_CHILD WS_VISIBLE |
    BS_PUSHBUTTON, rect, this, IDC_BUTTON);
```

将生成一个标记为"Start"的按钮控件。第一个参数是在按钮表面显示的文本。第二个参数是按钮样式，是常规（WS_）窗口样式和按钮特有的窗口样式的组合。WS_CHILD、WS_VISIBLE 以及 BS_PUSHBUTTON 创建的按钮控件是第四个参数标识的窗口的孩子，在屏幕上可见。（如果从窗口样式中省略 WS_VISIBLE，那么直到调用 ShowWindow 之前控件是不可见的。）rect 是一个 RECT 结构或 CRect 对象用来指定控件以像素为单位相对于父亲客户区左上角的尺寸和位置。this 标识父窗口，IDC_BUTTON 是一个整数值用来标识控件自身。这个值也称为子窗口 ID 或控件 ID。对给定窗口中每个创建的控件分配唯一的 ID 是非常重要的，这样才能把控件的通知消息与父窗口类中的成员函数对应。

用 Create 创建的列表框和编辑控件具有"扁平"的外观。要想赋予它们大多数用户所熟悉的立体式样（图 7-1），就必须使用 CreateEx 而不是 Create 来创建它们，并在函数的第一个参数所指定的扩展样式中包含 WS_EX_CLIENTEDGE 标志。如果 m_wndListBox 是 CListBox 对象，下列语句将创建一个立体的列表框，它的父亲是 this 指针标识的窗口：

```
m_wndListBox.CreateEx(WS_EX_CLIENTEDGE, _T("LISTBOX"), NULL,
    WS_CHILD | WS_VISIBLE | LBS_STANDARD, rect, this, IDC_LISTBOX);
```

另外，还可以从 CListBox 派生自己的类，覆盖派生类中的 PreCreateWindow，并在 PreCreateWindow 中使用 WS_EX_CLIENTEDGE 窗口样式，如下所示：

```
BOOL CMyListBox::PreCreateWindow(CREATESTRUCT& cs)
{
    if(!CListBox::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle |= WS_EX_CLIENTEDGE;
        .
        .
        .
```

```
    return TRUE;
}
```

像这样使用 PreCreateWindow,则无论 CMyListBox 对象是如何创建的,它都会有立体外观。

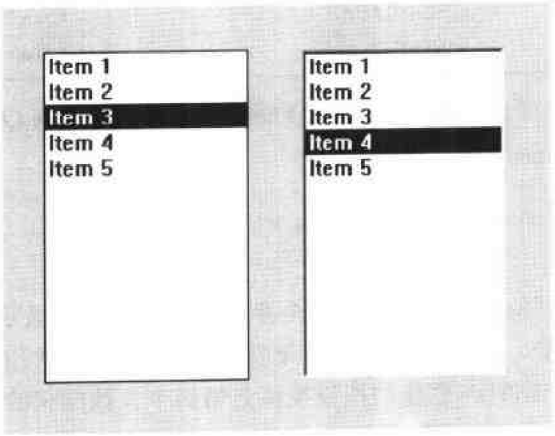

图 7-1 列表框的平面边(左)和凿刻边(右)外观

控件以 WM_COMMAND 消息的形式给它的父亲发送通知。虽然控件类型不同所发送的通知种类也不同,但是在任何时候,信息都被编码存入了消息的 wParam 和 lParam 参数,被用来标识发送消息的控件以及激励消息的操作。例如:在按钮被单击后发送的 WM_COMMAND 消息中,在 wParam 的高 16 位保存着通知代码 BN_CLICKED,wParam 的低 16 位保存着控件 ID,在 lParam 中保存着控件窗口句柄。

大多数应用程序不处理原始的 WM_COMMAND 消息,而是使用消息映射将控件通知与类成员函数联系在一起。例如:下列消息映射表输入项将控件 ID 为 IDC_BUTTON 按钮的单击通知与成员函数 OnButtonClicked 进行了映射:

```
ON_BN_CLICKED (IDC_BUTTON, OnButtonClicked)
```

ON_BN_CLICKED 是 MFC 提供的几个与控件有关的消息映射宏之一。例如:对于编辑控件有 ON_EN 宏,对于列表框控件有 ON_LBN 宏。还有更一般的 ON_CONTROL 宏,它处理所有通知和所有控件类型,以及 ON_CONTROL_RANGE 宏,将两个以上控件发出的同样的通知映射给共用的通知处理程序。

控件给它们的父窗口发送消息,但父窗口给控件发送消息的时候也不少。例如:通过发送带有 wParam 等于 BST_CHECKED 的 BM_SETCHECK 消息在复选框控件中设置复选标记。通过在交换 BM_SETCHECK 和其他控件消息的控件类中创建成员函数,MFC 简化了基于消息的控件接口。例如:语句

```
m_wndCheckBox.SetCheck(BST_CHECKED);
```

将复选标记放在了复选框中，该复选框由名为 m_wndCheckBox 的 CButton 对象代表。

由于控件是窗口，对于控件程序设计，一些控件类从 CWnd 继承来的成员函数很有用。例如：同样一个在窗口的标题栏内修改文本的 SetWindowText 函数也可以在编辑控件中插入文本。其他有用的 CWnd 函数还包括 GetWindowText，它可以从控件中检索文本；EnableWindow，可以使控件有效或无效；SetFont 可以修改控件的字体。如果您想对控件进行一些处理而又在控件类中找不到合适的成员函数，不妨查看一下 CWnd 的成员函数列表。可能会在那里找到您想要的东西。

7.1.1　CButton 类

CButton 代表基于"BUTTON"WNDCLASS 的按钮控件。按钮有 4 种类型：按钮、复选框、单选按钮以及组框。所有 4 种按钮类型如图 7-2 所示。

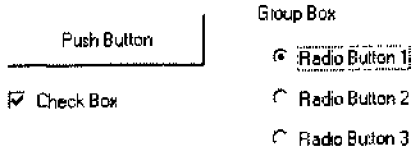

图 7-2　4 种类型的按钮控件

在创建按钮控件时，只要在按钮的窗口样式中设置表 7-2 中的标识之一，就可以确定想要创建的按钮形式。

表 7-2　按钮的窗口样式中包含的标识

样式	说　明
BS_PUSHBUTTON	创建一个标准按钮控件
BS_DEFPUSHBUTTON	创建默认按钮；用在对话框中指定回车键按下时相当于被单击的按钮
BS_CHECKBOX	创建复选框控件
BS_AUTOCHECKBOX	创建被单击时可以选中和不选中自己的复选框控件
BS_3STATE	创建 3 种状态的复选框控件
BS_AUTO3STATE	创建一个 3 状态复选框，它被单击时 3 种状态循环-选中、未选中、不确定
BS_RADIOBUTTON	创建单选按钮控件

续表

样式	说　明
BS_AUTORADIOBUTTON	创建一个单选按钮控件,它被单击时选中自己而取消对同组中别的单选按钮的选择
BS_GROUPBOX	创建组框控件

另外,可以在窗口样式中用表7-3中的一个或多个值进行"OR"运算来控制按钮表面文本的对齐方式。

表7-3 窗口样式用的值

样式	说　明
BS_LEFTTEXT	将伴随单选按钮或复选框的文本从按钮右边(默认)向左边移动
BS_RIGHTBUTTON	与BS_LEFTTEXT相同
BS_LEFT	左对齐控件矩形中的按钮文本
BS_CENTER	在控件矩形中将文本对中
BS_RIGHT	右对齐控件矩形中的按钮文本
BS_TOP	将按钮文本置于控件矩形的顶部
BS_VCENTER	将按钮文本置于控件矩形垂直方向的中间
BS_BOTTOM	将按钮文本置于控件矩形的底部
BS_MULTILINE	允许一行放不下的文本分成两行或多行

还有其他样式的按钮,但大多数都很少用到。例如:BS_NOTIFY编制一个按钮发送BN_DOUBLECLICKED、BN_KILLFOCUS和BN_SETFOCUS通知。BS_OWNERDRAW创建一个由所有者绘制的按钮——它的外观由按钮的父亲而不是自己维护。由所有者绘制的按钮主要已经被位图按钮和图标按钮替代了。在本章的稍后部分将学到位图按钮和图标按钮。

按钮

按钮是用BS_PUSHBUTTON样式创建的按钮控件。被单击时,按钮给它的父亲发送一个封装在WM_COMMAND消息中的BN_CLICKED通知。如果没有BS_NOTIFY按钮样式,按钮不会发送其他样式的通知。

MFC的ON_BN_CLICKED宏将BN_CLICKED通知与父窗口类中的成员函数联系在一起。消息映射输入项。

```
ON_BN_CLICKED(IDC_BUTTON, OnButtonClicked)
```

连接了OnButtonClicked和控件ID为IDC_BUTTON的按钮的单击事件。一个小型OnButtonClicked的应用如下:

```
void CMainWindow::OnButtonClicked()
{
    MessageBox(_T("I've been clicked!"));
}
```

如同对菜单项的命令处理程序，BN_CLICKED处理程序也不接受参数和返回任何值。

复选框

复选框是用样式BS_CHECKBOX、BS_AUTOCHECKBOX、BS_3STATE或BS_AUTO3STATE创建的按钮。BS_CHECKBOX和BS_AUTOCHECKBOX样式的复选框有两个状态：选中和未选中。复选框可以用CButton::SetCheck来选中和取消选中：

```
m_wndCheckBox.SetCheck(BST_CHECKED); // Check
m_wndCheckBox.SetCheck(BST_UNCHECKED); // Uncheck
```

要确定复选框是否被选中可以使用CButton::GetCheck。返回值为BST_CHECKED说明已经选中，否则为BST_UNCHECKED。

像按钮一样，复选框在被单击时给它的父窗口发送BN_CLICKED通知。在BS_AUTOCHECKBOX样式的复选框中的复选标记自动切换开关状态来响应按钮的单击。而在BS_CHECKBOX样式复选框中的复选标记却不是这样。因此，很少使用BS_CHECKBOX样式的复选框，除非为它们编写BN_CLICKED处理程序。下列BN_CLICKED处理程序将m_wndCheckBox的复选标记在开和关状态之间切换：

```
void CMainWindow::OnCheckBoxClicked()
{
    m_wndCheckBox.SetCheck(m_wndCheckBox.GetCheck() ==
        BST_CHECKED ? BST_UNCHECKED : BST_CHECKED);
}
```

用BS_3STATE和BS_AUTO3STATE按钮样式创建的复选框具有选中和未选中以外的第3种状态。第3种状态称为不确定状态，在用户单击当前被选中的BS_AUTO3STATE样式复选框时或在用BST_INDETERMINATE参数调用SetCheck时出现此状态：

```
m_wndCheckBox.SetCheck(BST_INDETERMINATE);
```

不确定复选框包含一个灰化了的复选标记。不确定状态用来说明某些事情既不完全肯定也不完全否定。例如：在一个字处理程序中，当用户混合使用常规和粗体文本时，标有“粗体”的复选框可能处于不确定状态。

单选按钮

单选按钮是使用样式BS_RADIOBUTTON或BS_AUTORADIOBUTTON的按钮控件。单

选按钮通常成组使用,每个单选按钮都代表一组相互排斥的选项中的一个。被单击时,BS_AUTORADIOBUTTON型单选按钮选中自己并撤消对同一组中其他按钮的选择。但是,如果选用 BS_RADIOBUTTON 样式,您就要自己使用 CButton::SetCheck 来进行这些选中和撤消选中处理了。

像按钮和复选框一样,单选按钮给它们的父窗口发送 BN_CLICKED 通知。下列 BN_CLICKED处理程序将选中 m_wndRadioButton1 单选按钮并撤消对同一组中的其他按钮的选择:

```
void CMainWindow::OnRadioButton1Clicked()
{
    m_wndRadioButton1.SetCheck(BST_CHECKED);
    m_wndRadioButton2.SetCheck(BST_UNCHECKED);
    m_wndRadioButton3.SetCheck(BST_UNCHECKED);
    m_wndRadioButton4.SetCheck(BST_UNCHECKED);
}
```

撤消对其他按钮的选择可以维护选项的排它性。虽然对于 BS_AUTORADIOBUTTON 单选按钮 BN_CLICKED 处理程序并不是必须的,但是如果愿意仍然可以提供它来响应当按钮被单击时单选按钮的状态变化。

要使 BS_AUTORADIOBUTTON 单选按钮恰当地撤消对一组中其他按钮的选择,就必须将按钮分组,以便 Windows 知道某个按钮与一组按钮的所属关系。按照下列步骤,可以创建一组 BS_AUTORADIOBUTTON 单选按钮:

1. 在应用程序代码中一个接一个按顺序创建按钮,它们之间不可以插入其他控件。
2. 要标记一组的开始,可以将 WS_GROUP 样式赋给创建的第一个单选按钮。
3. 如果在最后一个单选按钮创建之后要创建其他控件,就要把 WS_GROUP 样式赋给创建的第一个其他控件。这就隐含地说明上一个控件(最后一个单选按钮)是一组中的最后一个。如果在单选按钮之后没有其他控件了,但在窗口中还有别的控件,就要将第一个控件用 WS_GROUP 作标记以防止单选按钮受其干扰。

下面的例子说明如何将 4 个 BS_AUTORADIOBUTTON 单选按钮创建为一组,三个创建为另一组,并在两组间设置一个复选框控件:

```
m_wndRadioButton1.Create(_T("COM1"), WS_CHILD|WS_VISIBLE |
    WS_GROUP|BS_AUTORADIOBUTTON, rect1, this, IDC_COM1);
m_wndRadioButton2.Create(_T("COM2"), WS_CHILD|WS_VISIBLE |
    BS_AUTORADIOBUTTON, rect2, this, IDC_COM2);
m_wndRadioButton3.Create(_T("COM3"), WS_CHILD|WS_VISIBLE |
    BS_AUTORADIOBUTTON, rect3, this, IDC_COM3);
m_wndRadioButton4.Create(_T("COM4"), WS_CHILD|WS_VISIBLE |
    BS_AUTORADIOBUTTON, rect4, this, IDC_COM4);
m_wndRadioButton1.SetCheck(BST_CHECKED);
```

```
m_wndCheckBox.Create(_T("Save settings on exit"),
    WS_CHILD|WS_VISIBLE|WS_GROUP|BS_AUTOCHECKBOX,
    rectCheckBox, this, IDC_SAVESETTINGS);

m_wndRadioButton5.Create(_T("9600"), WS_CHILD|WS_VISIBLE|
    WS_GROUP|BS_AUTORADIOBUTTON, rect5, this, IDC_9600);
m_wndRadioButton6.Create(_T("14400"), WS_CHILD|WS_VISIBLE|
    BS_AUTORADIOBUTTON, rect6, this, IDC_14400);
m_wndRadioButton7.Create(_T("28800"), WS_CHILD|WS_VISIBLE|
    BS_AUTORADIOBUTTON, rect7, this, IDC_28800);
m_wndRadioButton5.SetCheck(BST_CHECKED);
```

由于使用了 BS_AUTORADIOBUTTON 样式和用 WS_GROUP 标识位进行了逻辑分组，只要选中前四个单选按钮之一就会自动地取消对其他三个的选择，只要选中第二组中任一个单选按钮就可以撤消对另外两个的选择。

出于形式上的考虑，上述程序调用 SetCheck 来复选各组中的按钮。在一组单选按钮中，即使用户没有提供输入也应该总有一个被选中。在默认状态下单选按钮处于未选中状态，因此您应该挑起初始化它们的责任。

组框

组框是使用了 BS_GROUPBOX 样式的按钮控件，与其他按钮控件不同，组框从不接受输入也不给父窗口发送通知。

组框的唯一作用是给控件组绘制可见的轮廓。将控件组圈在组框中可以使用户明确哪些控件在一起。组框与控件的逻辑分组毫无关系，因此不要因为简单地将一系列单选按钮用组框围起来就以为它们属于一组。

7.1.2 CListBox 类

MFC 的 CListBox 类封装了列表框控件，显示包含叫做项的文本字符串列表。列表框有选择性地将加入其中的项目分类，它还具有滚动功能使列表框中可以显示的项目数量不受列表框窗口物理尺寸的限制。

列表框对于提供列表信息并允许用户从列表中选择项目非常有用。当项被单击或双击时，大多数列表框（理论上讲是在窗口样式中带有 LBS_NOTIFY 的列表框）将用 WM_COMMAND 消息通知其父窗口。MFC 提供了 ON_LBN 消息映射宏从而简化了对这些消息的处理，可以使用宏将列表框的通知传递给父窗口类中的处理函数。

标准的列表框在垂直列中显示文本字符串，只允许一次选择一个项目。当前选择的项目用系统颜色 COLOR_HIGHLIGHT 加亮显示。Windows 支持许多标准列表框的变体，其中包括多选列表框、多列列表框以及显示图形而不是文本的自制列表框。

创建列表框

以下语句从名为 m_wndListBox 的 CListBox 对象中创建一个标准列表框：

```
m_wndListBox.Create (WS_CHILD|WS_VISIBLE|LBS_STANDARD,
    rect, this, IDC_LISTBOX);
```

LBS_STANDARD 组合了 WS_BORDER、WS_VSCROLL、LBS_NOTIFY 以及 LBS_SORT 样式，用它创建的列表框具有边框和垂直滚动条，可以在选项被更改或项目被双击时通知其父窗口，还可以按字母顺序排列加入的字符串。在默认状态下，滚动条只有在列表框中的项目数量超出可显示的数量时才出现。要使滚动条在任何时候都可见，必须包含 LBS_DISABLENOSCROLL 样式。除非使用了 WS_VSCROLL 或 LBS_STANDARD 样式，否则列表框不具有垂直滚动条。同样，如果没使用 WS_BORDER 或 LBS_STANDARD 样式也不会有边框。如果创建的列表框包围了父窗口的整个客户区，您也可以省略边框。用来自定义列表框外观和功能的样式总结在下表中给出。

列表框自身带有键盘接口。在单选列表框具有输入焦点后，就可以用向上箭头、向下箭头、Page Up、Page Down、Home 以及 End 键来移动加亮条来标识当前的选项。另外，按下字符键可以选中下一个以该字符开头的选项。键盘输入也可以在多选列表框中工作，不过是有焦点的矩形位置变化而不是选项更改。按下空格键可以用多选列表框中的焦点来切换项目的选中状态。

可以通过使用 LBS_WANTKEYBOARDINPUT 样式并处理 WM_VKEYTOITEM 和 WM_CHARTOITEM 消息来自定义列表框的键盘接口(见表 7-4)。MFC 应用程序可以使用 ON_WM_VKEYTOITEM 和 ON_WM_CHARTOITEM 宏将这些消息映射给 OnVKeyToItem 和 OnCharToItem 处理程序。派生的列表框类自己可以通过覆盖虚拟 CListBox::VKeyToItem 和 CListBox::CharToItem 函数来处理这些消息。这种功能的一个用途是创建包含自身的列表框，它通过删除当前选中的项来响应按下 Ctrl-D 事件。

表 7-4 列表框样式

样式	说 明
LBS_STANDARD	创建"标准"列表框，具有边框和垂直滚动条，在选项被更改或项目被双击时通知其父窗口，并按字母顺序排列项目
LBS_SORT	将加入列表框中的项目排序
LBS_NOSEL	创建项目只能查看而不能选择的列表框
LBS_NOTIFY	创建在选项被更改或项目被双击时通知父窗口的列表框
LBS_DISABLENOSCROLL	在不需要时取掉列表框的滚动条。如果没有使用此样式，不需要的滚动条会被隐藏而不是被拿掉

续表

样式	说　明
LBS_MULTIPLESEL	创建多选列表框
LBS_EXTENDEDSEL	给多选列表框添加扩展选项支持
LBS_MULTICOLUMN	创建多列列表框
LBS_OWNERDRAWVARIABLE	创建自制列表框，其中项的高度不同
LBS_OWNERDRAWFIXED	创建自制列表框，其中项的高度相同
LBS_USETABSTOPS	配置列表框展开项文本中的制表符
LBS_NOREDRAW	创建项目被添加或删除后不进行重画自身的列表框
LBS_HASSTRINGS	创建“记得”所添加字符串的列表框。常规列表框在默认时具有此样式，而自制列表框则没有
LBS_WANTKEYBOARDINPUT	创建当键按下时给父窗口发送 WM_VKEYTOITEM 或 WM_CHARTOITEM 消息的列表框。此样式用来自定义列表框对键盘输入的响应
LBS_NOINTEGRALHEIGHT	允许列表框具有任意高度。默认状态下，Windows 将列表框的高度设置为项目高度的倍数以防止项目的部分被省略

由于 Windows 在列表框中使用的字体是按比例调间距的，因此不可能靠眼睛通过用空格字符分开的方法将列表框中的信息排成一列。一种创建柱状列表框显示的方法是使用 SetFont 函数在列表框中应用固定调距字体。一种较好的办法是赋给列表框 LBS_USETABSTOPS 样式并用制表符将信息列分开。LBS_USETABSTOPS 样式的列表框象字处理器一样处理制表符，在遇到制表符时会自动前进到下一个制表位。默认时，制表位平均间隔 8 个字符宽。可以用 CListBox::SetTabStops 函数来修改制表位设置。SetTabStops 以对话框单位测量距离。一个对话框单位大约等于系统字体一个字符宽度的四分之一。语句

```
m_wndListBox.SetTabStops (64);

sets the space between tab stops to 64 dialog units, and

int nTabStops[] =  32, 48, 64, 128  ;
m_wndListBox.SetTabStops (4, nTabStops);
```

从左边开始将制表位设置在 32、48、64、和 128 对话框单位的地方。

默认时，每当项目被添加或删除时列表框都要重画自身。通常这也是您希望的，但当您以很快的方式添加成百上千个项目时，重复绘制会产生难看的闪烁效果并减慢插入过程。可以用 LBS_NOREDRAW 来创建不进行自动重画的列表框。这种列表框只有在客户区无效时才进行重画。

除了使用 LBS_NOREDRAW 以外还可以在开始大量的插入之前关闭重画功能并在最后一项插入之后恢复。通过给列表框发送 WM_SETREDRAW 消息可以自动地关闭并恢复重

画功能,如下:

```
m_wndListBox.SendMessage (WM_SETREDRAW, FALSE, 0); // Disable redraws.
    .
    .
    .
m_wndListBox.SendMessage (WM_SETREDRAW, TRUE, 0); // Enable redraws.
```

列表框会在 WM_SETREDRAW 消息使重画功能恢复时自动重画,因此没必要跟随对 Invalidate的调用。

除非用 LBS_MULTIPLESEL 样式创建列表框,否则每次只能选择一个项目。在单选列表框中,单击未选中的项目就会在选中该项目的同时撤消对其他项目的选择。在多选列表框中,却可以选择任意多个项目。大多数多选列表框也可以用 LBS_EXTENDEDSEL 样式来创建,这种列表框允许扩充选项。在扩充选项列表框中,用户可以单击来选中第一个项目然后按下 Ctrl 键同时单击来选中以后的其他项目。另外,用户还可以选择一个范围内邻近的所有项目,先单击选中范围内第一个项目然后按下 Shift 键并单击范围内最后一个项目即可。Ctrl 和 Shift 键可以组合使用来选中多个项目和多个范围,这样就为选择任意的项目组合提供了一种方便的接口。

LBS_MULTICOLUMN 样式创建多列列表框。多列列表框通常使用 WS_HSCROLL 样式来创建,以便不能同时显示所有项目时可以将其中内容作水平滚动。(多列列表框不可以垂直滚动。)可以用 CListBox::SetColumnWidth 函数来调整列的宽度。通常,列的宽度应该基于使用列表框字体字符的平均宽度。默认列宽度足够显示具有默认列表框字体的 16 个字符,因此如果想插入更长的字符串,就必须扩充列的宽度防止列之间重叠。

添加和删除项

直到给它添加项目以前,列表框是空的。项目可以用 CListBox::AddString 和 CListBox::InsertString 添加。语句

```
m_wndListBox.AddString (string);
```

给列表框添加了一个名为 string 的 CString 对象。如果列表框包含 LBS_SORT 样式,就会根据字典中的顺序来放置字符串,否则就将字符串添加在列的末端。InsertString 将项目添加到列表框中由 0 开始的索引号指定的地方。语句

```
m_wndListBox.InsertString (3, string);
```

将 string 插入列表框并放在第四条位置上。LBS_SORT 对用 InsertString 添加的项目不起作用。

AddString 和 InsertString 都返回基于 0 的索引号以指定字符串在列表框中的位置。如果任一个函数操作失败,都返回 LB_ERRSPACE 说明列表框已满或返回 LB_ERR 说明由于其

他原因插入失败。在 32 位 Windows 系统中很少能遇到返回 LB_ERRSPACE 的情况，这是因为列表框的储存能力只受到可用内存的限制。CListBox::GetCount 返回列表框中项目的数量。

CListBox::DeleteString 从列表框中删除一个项目。仅接受一个参数：要删除项目的索引号。返回列表框中剩余项目的数量。要一下把列表框中的项目都删除，使用 CListBox::ResetContent。

如果愿意，可以使用 CListBox::SetItemDataPtr 或 CListBox::SetItemData 将列表框中的项目与 32 位指针或 DWORD 关联在一起。与项目关联的指针或 DWORD 可以用 CListBox::GetItemDataPtr 或 CListBox::GetItemData 来检索。这种功能的一种用途就是将附加数据与列表框中的项目关联。例如：可以把包含地址和电话号码的数据结构与包含人名的列表框项目关联起来。由于 GetItemDataPtr 返回指向 void 数据类型的指针，所以需要强制转换它返回的指针。

另一个程序员用来将附加数据（特别是基于文本的数据）和列表框项目关联的技巧是创建 LBS_USETABSTOPS 样式的列表框，将制表位放置在超出列表框右边框的位置，并将包含制表符以及附加数据（跟随在制表符后）的字符串附加在列表框项目上。制表符右边的文本是不可见的，而 CListBox::GetText 会返回列表框项目的完整文本，当然也包括附加的文本。

查找与检索项目

CListBox 类也包括可以用来获取和设置当前选项以及查找和检索项目的成员函数。CListBox::GetCurSel 返回当前被选中项目基于 0 的索引号。返回值为 LB_ERR 意味着什么也没选。GetCurSel 通常在通知选项被更改或项目被双击后调用。程序可以用 SetCurSel 函数来设置当前选项。将 -1 传递给 SetCurSel 取消对任何项目的选择，使突出显示当前选中项的加亮条从列表框中消失。可以使用 CListBox::GetSel 来确定某个特定的项是否被选中。

SetCurSel 用索引号区分项目，但也可以用项目的内容来选择它。CListBox::SelectString 在单选列表框中查找以指定文本字符串开头的项目并在找到匹配项目后选中它。语句

```
m_wndListBox.SelectString(-1, _T("Times"));
```

从列表框中的第一个项目开始查找并在找到第一个以“Times”开头的项目后加亮显示它，例如“Times New Roman”或“Times Roman”。查找不区分大小写。SelectString 的第一个参数指定查找开始处前一个项目的索引号，-1 指出从 0 号项目开始。如果查找从任意地方开始的，必要时会最终返回第一个项目以保证所有项目都被查询过。

要查找列表框中特定的项目而又不修改选项，可以使用 CListBox::FindString 或 CListBox::FindStringExact。FindString 根据列表框的内容进行字符串查找并在找到第一个匹配的项目或以指定字符串开头的项目之后返回它的索引号。返回值如果是 LB_ERR 则说明找不到匹配项目。FindStringExact 也执行同样的操作但是仅仅在项目文本与查找文本完全匹配

时才报告。一旦得到了项目的索引号,就可以用 CListBox::GetText 来检索项目的文本。下列语句在列表框中查询当前选中的项目并将其文本复制到名为 string 的 CString 中:

```
CString string;
int nIndex = m_wndListBox.GetCurSel ();
if (nIndex != LB_ERR)
   m_wndListBox.GetText (nIndex, string);
```

另一种 GetText 的可选形式是接受指向字符数组的指针而不是 CString 引用。在调用数组类型的 GetText 之前可以使用 CListBox::GetTextLen 来确定需要的数组大小。

在多选列表框中对选项的处理与在单选列表框中不同。特别是,GetCurSel、SetCurSel 和 SelectString 函数对多选列表框不再适用。项目是用 SetSel 和 SelItemRange 函数来选中和取消选中的。下列语句选中了项目 0、5、6、7、8 以及 9 并同时取消了对项目 3 的选择:

```
m_wndListBox.SetSel (0);
m_wndListBox.SelItemRange (TRUE, 5, 9);
m_wndListBox.SetSel (3, FALSE);
```

CListBox 还提供了 GetSelCount 函数用来获取选中项目的个数,以及 GetSelItems 函数用来检索所选项目的索引号。在多选列表框中,点矩形代表具有焦点的项目可以被移动而不会修改当前选项。焦点矩形可以用 SetCaretIndex 和 GetCaretIndex 来移动和查询。大多数其他列表框函数,包括 GetText、GetTextLen、FindString 以及 FindStringExact,对多选列表框的操作和对单选列表框的相同。

列表框通知

列表框通过 WM_COMMAND 消息给父窗口发送通知。在 MFC 应用程序中,列表框通知通过 ON_LBN 消息映射表输入项映射给了类成员函数。表 7-5 中列出了 6 种通知类型以及它们相应的 ON_LBN 宏。只有用 LBS_NOTIFY 或 LBS_STANDARD 样式创建的列表框才发送 LBN_DBLCLK、LBN_SELCHANGE 和 LBN_SELCANCEL 通知。其他通知则与列表框样式无关。

表 7-5 列表框通知

通知	发送条件	消息映射宏	要求使用 LBS_NOTIFY 吗?
LBN_SETFOCUS	列表框获得输入焦点	ON_LBN_SETFOCUS	否
LBN_KILLFOCUS	列表框失去输入焦点	ON_LBN_KILLFOCUS	否
LBN_ERRSPACE	内存不足操作失败	ON_LBN_ERRSPACE	否
LBN_DBLCLK	项目被双击	ON_LBN_DBLCLK	是
LBN_SELCHANGE	选项被修改	ON_LBN_SELCHANGE	是
LBN_SELCANCEL	选项被取消	ON_LBN_SELCANCEL	是

程序员使用最多的两个列表框通知是 LBN_DBLCLK 和 LBN_SELCHANGE。LBN_DBLCLK 在列表框项目被双击时发送。要获得单选列表框中被双击项目的索引号，可以使用 CListBox::GetCurSel。下列程序代码片段在消息框中显示项目：

```
// In CMainWindow's message map
ON_LBN_DBLCLK (IDC_LISTBOX, OnItemDoubleClicked)
    .
    .
    .
void CMainWindow::OnItemDoubleClicked ()
{
    CString string;
    int nIndex = m_wndListBox.GetCurSel ();
    m_wndListBox.GetText (nIndex, string);
    MessageBox (string);
}
```

对于多选列表框，要用 GetCaretIndex 而不是 GetCurSel 来确定哪个项目被双击了。

在用户修改选项，而不是被程序修改时，列表框发送 LBN_SELCHANGE 通知。单选列表框在鼠标单击或击键造成选项移动时发送 LBN_SELCHANGE 通知。多选列表框在以下情况发送 LBN_SELCHANGE 通知：项目被单击、用空格键切换项目的所选状态以及焦点矩形移动时。

7.1.3　CStatic 类

CStatic，代表从“STATIC”WNDCLASS 创建的静态控件，是 MFC 控件类中最简单的一种。至少过去是这样：Windows 95 给静态控件添加了许多新功能，现在在复杂性方面 CStatic 可以和 CButton 和其他一些控件类匹敌了。

静态控件有三种类型：文本、矩形和图像。静态文本经常用来标记其他控件。下列语句创建一个静态文本控件显示字符串“Name”：

```
m_wndStatic.Create (_T ("Name"), WS_CHILD|WS_VISIBLE|SS_LEFT,
    rect, this, IDC_STATIC);
```

SS_LEFT 创建一个文本左对齐的静态文本控件。如果控件文本太长在一行放不下，它就会换行。要防止换行可以使用 SS_LEFTNOWORDWRAP 而不是 SS_LEFT。用 SS_CENTER 或 SS_RIGHT 替换 SS_LEFT 或 SS_LEFTNOWORDWRAP 可以将文本在水平方向中间对齐或右对齐。另一种可选的样式是很少用到的 SS_SIMPLE，它的作用与 SS_LEFT 相似但是用它创建的控件文本不能用 CWnd::SetWindowText 修改。

默认时，赋给静态文本控件的文本沿控件矩形的上边对齐。要想将文本在控件矩形垂

直方向上居中,可以在控件样式中"OR"运算 SS_CENTERIMAGE 标志。还可以通过使用 SS_SUNKEN 围绕静态控件画一个下陷边缘。

静态控件的第 2 个用途是画矩形。静态样式指定了所画矩形的类型。表 7-6 列出供选用的样式。

表 7-6 可选用的样式

样式	说 明
SS_BLACKFRAME	用系统颜色 COLOR_WINDOWFRAME (默认为黑色)画空心矩形
SS_BLACKRECT	用系统颜色 COLOR_WINDOWFRAME (默认为黑色)画实体矩形
SS_ETCHEDFRAME	带有蚀刻框的空心矩形
SS_ETCHEDHORZ	顶边和底边是蚀刻线的空心矩形
SS_ETCHEDVERT	左右边是蚀刻线的空心矩形
SS_GRAYFRAME	用系统颜色 COLOR_BACKGROUND (默认为灰色)画空心矩形
SS_GRAYRECT	用系统颜色 COLOR_BACKGROUND (默认为灰色)画空心矩形
SS_WHITEFRAME	用系统颜色 COLOR_BACKGROUND (默认为白色)画空心矩形
SS_WHITERECT	用系统颜色 COLOR_BACKGROUND (默认为白色)画空心矩形

语句

```
m_wndStatic.Create (_T (""), WS_CHILD|WS_VISIBLE|SS_ETCHEDFRAME,
    rect, this, IDC_STATIC);
```

创建一个类似于组框的静态控件。为取得最佳效果,应该在与默认对话框颜色(系统颜色为 COLOR_3DFACE)相同的表面绘制蚀刻矩形。即使在调用 Create 时指定了非零文本字符串,静态矩形控件也不会显示文本。

静态控件第 3 个用途是显示由位图、图标、光标或 GDI 元文件形成的图像。静态图像控件使用表 7-7 中列出的样式。

表 7-7 静态图像控件使用的样式

样式	说 明
SS_BITMAP	显示位图的静态控件
SS_ENHMETAFILE	显示元文件的静态控件
SS_ICON	显示图标或光标的静态控件

创建图像控件之后,要调用 SetBitmap、SetEnhMetaFile、SetIcon 或 SetCursor 函数将它与位图、元文件、图标或光标关联起来。语句

```
m_wndStatic.Create (_T (""), WS_CHILD|WS_VISIBLE|SS_ICON,
    rect, this, IDC_STATIC);
```

```
m_wndStatic.SetIcon(hIcon);
```

创建一个显示图标的静态控件并将句柄为 hIcon 的图标赋给它。默认时，图标图像位于控件的左上角，如果图标大于控件矩形，矩形会自动扩大使图像不会被剪去一部分。要将图像放在矩形的中间，可以在控件样式中加入“或”运算 SS_CENTERIMAGE。如果控件矩形很小而不能显示整个图像，SS_CENTERIMAGE 同时也阻止了系统自动缩放它，所以如果使用 SS_CENTERIMAGE，就要确保控件矩形足够得大来显示图像。对于 SS_ENHMETAFILE 样式的控件缩放不是问题，因为元文件可以缩放来与控件尺寸匹配。要获得漂亮的效果，可以在控件样式中使用 SS_SUNKEN 给图像控件添加一个下陷边框。

默认时，静态控件不给其父窗口发送通知。但是用 SS_NOTIFY 样式创建的静态控件可以发送表 7-8 中给出的 4 种类型通知。

表 7-8　静态控件通知

通知	发送条件	消息映射宏
STN_CLICKED	控件被单击	ON_STN_CLICKED
STN_DBLCLK	控件被双击	ON_STN_DBLCLK
STN_DISABLE	控件失效	ON_STN_DISABLE
STN_ENABLE	控件有效	ON_STN_ENABLE

STN_CLICKED 和 STN_DBLCLK 通知允许创建响应鼠标单击的静态控件。语句

```
// In CMainWindow's message map
ON_STN_CLICKED(IDC_STATIC, OnClicked)
        .
        .
        .
// In CMainWindow::OnCreate
m_wndStatic.Create(_T("Click me"), WS_CHILD|WS_VISIBLE|
    SS_CENTER|SS_CENTERIMAGE|SS_NOTIFY|SS_SUNKEN, rect,
    this,IDC_STATIC);
            .
            .
            .
void CMainWindow::OnClicked()
{
    m_wndStatic.PostMessage(WM_CLOSE, 0, 0);
}
```

创建了一个静态控件，它在下陷矩形的中间显示“Click me”，而当被单击时显示内容从屏幕上消失。如果静态控件没有 SS_NOTIFY 样式，由于控件窗口的处理过程在响应 WM_NCHITTEST 消息时返回 HTTRANSPARENT，所以鼠标消息就会到达下层窗口。

7.1.4 FontView 应用程序

我们把目前学到的有关按钮、列表框以及静态控件的知识用到一个应用程序中。图 7-3 所示的 FontView 程序在列表框中显示了安装在宿主 PC 机上的所有字体的名字列表。当某种字体被选中后,会在窗口底部的组框中显示示例。示例文本是静态控件,因此 FontView 显示字体示例就是要调用控件的 SetFont 函数。如果复选框"Show TrueType Fonts Only"被选中,则非 TrueType 类型的字体就会从列表中排除。除了说明如何使用按钮、复选框、列表框、组框以及静态控件以外,FontView 还演示了一种非常重要的 MFC 编程技巧,即用 C++ 成员函数作为回调函数。现在您可能还不太理解术语回调函数的意思,但很快就能学会了。

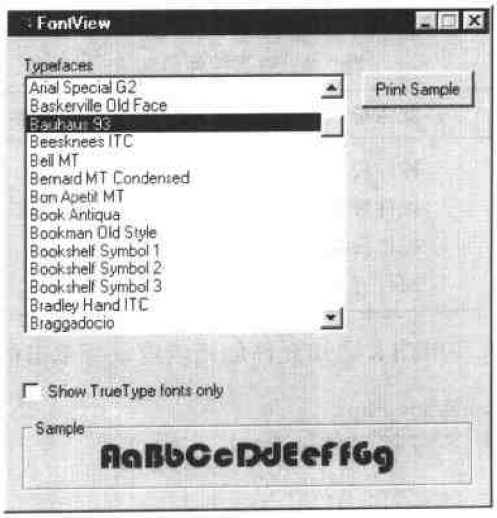

图 7-3 FontView 程序的窗口

在图 7-4 中给出了 FontView 的程序源代码。在 CMainWindow::OnCreate 中控件被一个接一个地创建。除了显示字体示例的静态控件以外所有控件都使用 8 点阵 MS Sans Serif 字体。没有使用原像素来计算控件的尺寸和位置,FontView 使用的是基于 8 点阵 MS Sans Serif 字符的宽度和高度来计算距离,这样就实现了对显示设备物理分辨率的独立性。通过把字体选入设备描述体并用指向 TEXTMETRIC 结构的指针来调用 CDC::GetTextMetrics,就可以测得字符的高度和宽度:

```
CFont * pOldFont = dc.SelectObject (&m_fontMain);
TEXTMETRIC tm;
dc.GetTextMetrics (&tm);
m_cxChar = tm.tmAveCharWidth;
m_cyChar = tm.tmHeight + tm.tmExternalLeading;
```

在返回值中，结构的 tmAveCharWidth 字段保存着平均字符宽度。（在按比例调整间距字体中实际的字符宽度各不相同。）将 tmHeight 和 tmExternalLeading 字段求和就得到一行文本的高度，其中包括行间距。

FontView.h

```
class CMyApp : public CWinApp
{
public:
    virtual BOOL InitInstance ();
};

class CMainWindow : public CWnd
{
protected:
    int m_cxChar;
    int m_cyChar;

    CFont m_fontMain;
    CFont m_fontSample;

    CStatic m_wndLBTitle;
    CListBox m_wndListBox;
    CButton m_wndCheckBox;
    CButton m_wndGroupBox;
    CStatic m_wndSampleText;
    CButton m_wndPushButton;

    void FillListBox ();

public:
    CMainWindow ();

    static int CALLBACK EnumFontFamProc (ENUMLOGFONT * lpelf,
        NEWTEXTMETRIC * lpntm, int nFontType, LPARAM lParam);

protected:
    virtual void PostNcDestroy ();

    afx_msg int OnCreate (LPCREATESTRUCT lpcs);
    afx_msg void OnPushButtonClicked ();
    afx_msg void OnCheckBoxClicked ();
    afx_msg void OnSelChange ();

    DECLARE_MESSAGE_MAP ()
};
```

FontView.cpp

```
#include <afxwin.h>
#include "FontView.h"

#define IDC_PRINT    100
#define IDC_CHECKBOX    101
#define IDC_LISTBOX    102
#define IDC_SAMPLE    103

CMyApp myApp;

//////////////////////////////////////////////////////////////////////////
// CMyApp member functions

BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow;
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

//////////////////////////////////////////////////////////////////////////
// CMainWindow message map and member functions

BEGIN_MESSAGE_MAP (CMainWindow, CWnd)
    ON_WM_CREATE ()
    ON_BN_CLICKED (IDC_PRINT, OnPushButtonClicked)
    ON_BN_CLICKED (IDC_CHECKBOX, OnCheckBoxClicked)
    ON_LBN_SELCHANGE (IDC_LISTBOX, OnSelChange)
END_MESSAGE_MAP ()

CMainWindow::CMainWindow ()
{
    CString strWndClass = AfxRegisterWndClass (
        0,
        myApp.LoadStandardCursor (IDC_ARROW),
        (HBRUSH) (COLOR_3DFACE + 1),
        myApp.LoadStandardIcon (IDI_WINLOGO)
    );
    CreateEx (0, strWndClass, _T ("FontView"),
        WS_OVERLAPPED|WS_SYSMENU|WS_CAPTION|WS_MINIMIZEBOX,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, NULL);

    CRect rect (0, 0, m_cxChar * 68, m_cyChar * 26);
```

```
    CalcWindowRect (&rect);

    SetWindowPos (NULL, 0, 0, rect.Width (), rect.Height (),
        SWP_NOZORDER|SWP_NOMOVE|SWP_NOREDRAW);
}

int CMainWindow::OnCreate (LPCREATESTRUCT lpcs)
{
    if (CWnd::OnCreate (lpcs) == -1)
        return -1;

    //
    // Create an 8-point MS Sans Serif font to use in the controls.
    //
    m_fontMain.CreatePointFont (80, _T ("MS Sans Serif"));

    //
    // Compute the average width and height of a character in the font.
    //
    CClientDC dc (this);
    CFont* pOldFont = dc.SelectObject (&m_fontMain);
    TEXTMETRIC tm;
    dc.GetTextMetrics (&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight + tm.tmExternalLeading;
    dc.SelectObject (pOldFont);

    //
    // Create the controls that will appear in the FontView window.
    //
    CRect rect (m_cxChar * 2, m_cyChar, m_cxChar * 48, m_cyChar * 2);
    m_wndLBTitle.Create (_T ("Typefaces"), WS_CHILD|WS_VISIBLE|SS_LEFT,
        rect, this);

    rect.SetRect (m_cxChar * 2, m_cyChar * 2, m_cxChar * 48,
        m_cyChar * 18);
    m_wndListBox.CreateEx (WS_EX_CLIENTEDGE, _T ("listbox"), NULL,
        WS_CHILD|WS_VISIBLE|LBS_STANDARD, rect, this, IDC_LISTBOX);

    rect.SetRect (m_cxChar * 2, m_cyChar * 19, m_cxChar * 48,
        m_cyChar * 20);
    m_wndCheckBox.Create (_T ("Show TrueType fonts only"), WS_CHILD |
        WS_VISIBLE|BS_AUTOCHECKBOX, rect, this, IDC_CHECKBOX);

    rect.SetRect (m_cxChar * 2, m_cyChar * 21, m_cxChar * 66,
        m_cyChar * 25);
    m_wndGroupBox.Create (_T ("Sample"), WS_CHILD|WS_VISIBLE|BS_GROUPBOX,
        rect, this, (UINT) -1);
```

```
    rect.SetRect (m_cxChar * 4, m_cyChar * 22, m_cxChar * 64,
        (m_cyChar * 99) / 4);
    m_wndSampleText.Create (_T (""), WS_CHILD|WS_VISIBLE|SS_CENTER, rect,
        this, IDC_SAMPLE);

    rect.SetRect (m_cxChar * 50, m_cyChar * 2, m_cxChar * 66,
        m_cyChar * 4);
    m_wndPushButton.Create (_T ("Print Sample"), WS_CHILD|WS_VISIBLE |
        WS_DISABLED|BS_PUSHBUTTON, rect, this, IDC_PRINT);

    //
    // Set each control's font to 8 - point MS Sans Serif.
    //
    m_wndLBTitle.SetFont (&m_fontMain, FALSE);
    m_wndListBox.SetFont (&m_fontMain, FALSE);
    m_wndCheckBox.SetFont (&m_fontMain, FALSE);
    m_wndGroupBox.SetFont (&m_fontMain, FALSE);
    m_wndPushButton.SetFont (&m_fontMain, FALSE);

    //
    // Fill the list box with typeface names and return.
    //
    FillListBox ();
    return 0;
}

void CMainWindow::PostNcDestroy ()
{
    delete this;
}

void CMainWindow::OnPushButtonClicked ()
{
    MessageBox (_T ("This feature is currently unimplemented. Sorry!"),
        _T ("Error"), MB_ICONINFORMATION|MB_OK);
}

void CMainWindow::OnCheckBoxClicked ()
{
    FillListBox ();
    OnSelChange ();
}
void CMainWindow::OnSelChange ()
{
    int nIndex = m_wndListBox.GetCurSel ();

    if (nIndex == LB_ERR) {
        m_wndPushButton.EnableWindow (FALSE);
```

```
        m_wndSampleText.SetWindowText (_T (""));
    }
    else {
        m_wndPushButton.EnableWindow (TRUE);
        if ((HFONT) m_fontSample != NULL)
            m_fontSample.DeleteObject ();

        CString strFaceName;
        m_wndListBox.GetText (nIndex, strFaceName);

        m_fontSample.CreateFont ( - m_cyChar * 2, 0, 0, 0, FW_NORMAL,
            0, 0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
            CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH |
            FF_DONTCARE, strFaceName);

        m_wndSampleText.SetFont (&m_fontSample);
        m_wndSampleText.SetWindowText (_T ("AaBbCcDdEeFfGg"));
    }
}

void CMainWindow::FillListBox ()
{
    m_wndListBox.ResetContent ();

    CClientDC dc (this);
    ::EnumFontFamilies ((HDC) dc, NULL, (FONTENUMPROC) EnumFontFamProc,
        (LPARAM) this);
}

int CALLBACK CMainWindow::EnumFontFamProc (ENUMLOGFONT * lpelf,
    NEWTEXTMETRIC * lpntm, int nFontType, LPARAM lParam)
{
    CMainWindow * pWnd = (CMainWindow *) lParam;

    if ((pWnd -> m_wndCheckBox.GetCheck () == BST_UNCHECKED) ||
        (nFontType & TRUETYPE_FONTTYPE))
        pWnd -> m_wndListBox.AddString (lpelf -> elfLogFont.lfFaceName);
    return 1;
}
```

图 7-4　FontView 应用程序

CMainWindow 处理三种类型的控件通知：从按钮产生的 BN_CLICKED 通知、从复选框产生的 BN_CLICKED 通知和从列表框产生的 LBN_SELCHANGE 通知。相应的消息映射表输入项如下：

```
ON_BN_CLICKED (IDC_PRINT, OnPushButtonClicked)
ON_BN_CLICKED (IDC_CHECKBOX, OnCheckBoxClicked)
```