第 2 章

菜单、工具栏和状态栏的设计

在 Windows 应用程序中，菜单、工具栏、状态栏等内容都是不可缺少的界面元素。菜单是一系列可视的命令列表，用户能够选中其中的菜单项（命令）并执行相应任务。工具栏提供图形按钮，实现快捷操作，用户可以通过工具栏执行最常用的命令，增强方便程度。状态栏可以显示动态的提示信息，便于用户的一些操作。

2.1 设计菜单

菜单为用户控制程序提供了一套分级选项，无论是标准菜单及命令，还是热键或弹出式菜单，以及为菜单或其命令定义加速键和状态条提示，都可用菜单编辑器来完成；除此之外，菜单项作为一个普通的对象也可在编辑时进行移动、复制、删除等操作。图 2.1 是一个典型的菜单实例，图中包含了标准菜单命令、快捷键、加速键、子菜单、核对符等。这些都是编写程序时经常遇到的。下面就介绍如何创建和编辑菜单项。

图 2.1 典型菜单实例

2.1.1 用编辑器设计菜单

当用户使用 AppWizard 创建 SDI 或 MDI 应用程序时，系统将为用户自动生成默认的菜单栏。用户需要做的工作仅仅是打开菜单编辑器，进行必要的修改，再编写菜单选项相应的消息处理函数即可。当然也可在菜单编辑器中创建新的菜单或创建新的菜单资源，如快捷菜单等。

无论是编辑已有菜单资源还是创建新的菜单资源，首先应当进入菜单资源编辑器。

［例 2.1］用编辑器设计一个子菜单

在菜单编辑器的“文件”下拉菜单的某个位置加一个菜单，其作用是单击它后，能在屏幕上显示一行字。下面介绍其编程步骤。

（1）建一个 SDI 单文档应用程序名为：显示一行字。

（2）用编辑器设计菜单：在项目工作区打开 ResourceView 选项卡，打开 Menu 文件夹，双击 IDR_MAINFRAME，右边出现菜单编辑器，如图 2.2 所示，打开“文件”下拉菜单，如图 2.2 所示，双击最后的空白菜单，弹出“菜单项属性”对话框，在 Caption 处写菜单名：显示一行字（&C），在 ID 处写 ID_FILE_XS，在 Prompt（注释）栏中写：点击新建菜单项，在窗口显示一行字，然后关闭该对话框。

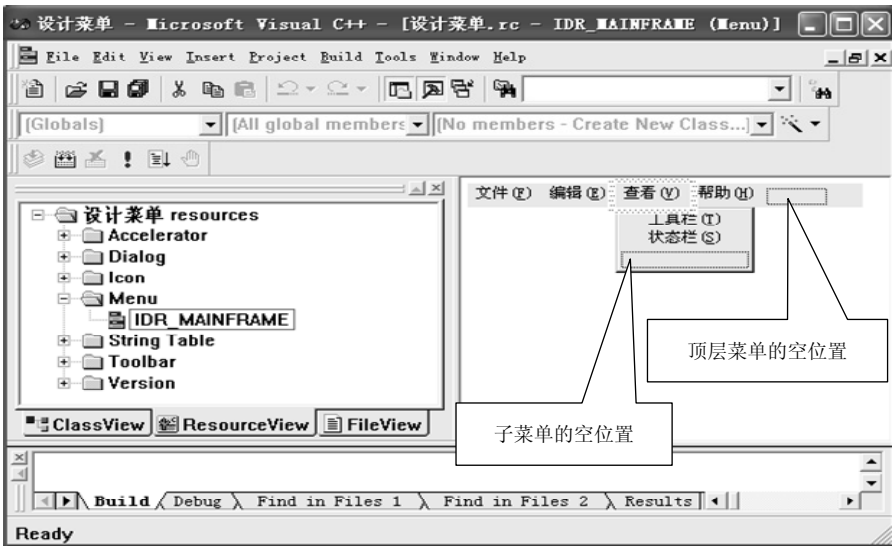

图 2.2 菜单资源编辑器

（3）在文档的头文件 CMyDoc.h 的 public 下添加：

```
CString str;   //定义字符串变量
```

在文档的实现文件 CMyDoc.cpp 的构造函数里添加：

```
str="  ";   //将字符串变量赋初值为"空"
```

（4）将菜单的标识 ID_FILE_XS 映射到视图类 CMyView 中。

在 Message Maps 选项卡的 Class name 下拉列表框中选择 CMyView 选项，在 Object IDs 下拉列表框中选择 ID_ FILE_XS 选项，在 Messages 列表框中选择 COMMAND 选项，依次单击 Add Function->OK->Edit Code 按钮。在 OnFileXS()映射函数中添加代码：

```
void CMyView::OnFileXS()
{  CMyDoc *pDoc=GetDocument();  //获得文档类指针
   ASSERT_VALID(pDoc);          //检查 pDoc 指针是否有效
    pDoc->str="你成功的在 File 菜单下，建立了一个显示一行字菜单";
   Invalidate();                //去强制执行 OnDraw()函数
}
```

（5）再在视图实现文件 CMyView.cpp 里的 OnDraw（CDC *Pdc）函数里添加代码：

```
CMyView::OnDraw(CDC *Pdc)
{  pDC->TextOut(100,100,pDoc->str);  } //在窗口的 x=100,y=100 坐标处输出字符串
```

（6）编译运行：

1）在出现的文档窗口上，选择“文件”→“显示一行字”命令，在屏幕上显示程序中写的一行字；

2）打开“文件”下拉菜单，直接按 C 键，也出现这行字；

3）将鼠标放在“显示一行字”这个菜单上，下面状态栏会出现在程序中写的注释：“点击新建菜单项，在窗口显示一行字”。

说明：

① TextOut（100，100，pDoc->str）是 CDC 类的输出函数，100，100 是 x，y 坐标，pDoc->str 是输出 str 内容。

② 改变菜单位置：鼠标左键选中该菜单不放，拖到你想要加的位置即可。

③ 想在哪个位置加菜单：可选中后面的那个菜单，按 Insert 键即可。

④“显示一行字（&C）”：其中“&”用于将其后面的字符作为该菜单项的助记符，也就是它后面的字符成了快捷键字符。打开这个菜单，直接按这个助记符键，菜单命令即被执行。

注意：快捷键字符 C 不能与同一级快捷键字符重复，例如：若写“显示一行字（&X）”，则与下面的“退出（&X）”重复，系统无法判别是哪个 X，则不能执行。

图 2.3 中的小对话框（Menu Item Properties）是双击“文件”下的空白菜单出现的菜单属性对话框，用它建立了“显示一行字（&C）”菜单，菜单 General 选项卡的各项含义如表 2.1 所示。

图 2.3　菜单编辑器和修改菜单项属性的结果

表 2.1　菜单 General 选项卡的各项含义

项　　目	含　　义
ID	菜单的资源 ID 标识符
Caption（标题）	用于标识菜单项显示文本，助记符字母前面须有一个&符号，这个字母与 Alt 构成组合键

续表

项　目	含　义
Separator（分隔符）	选中此复选框时，菜单项是一个分隔符或一条水平线
Checked（选中的）	选中此复选框时，菜单项文本前显示一个选中标记
Pop_up（弹出）	选中此复选框时，菜单项含有一个弹出式子菜单
Grayed（变灰）	选中此复选框时，菜单项显示是灰色的，用户不能选用
Inactive（非激活）	选中此复选框时，菜单项没有被激活，用户不能选用
Help（帮助）	选中此复选框时，菜单项在程序运行时被放在顶层菜单的最右端
Break（暂停）	当为 Column 时，对于顶层菜单项来说，被放置在另外一行上，而对于弹出式子菜单的菜单项来说，则被放置在另外一列上，当为 Bar 时，与 Column 相同，只不过对于弹出式子菜单来说，它还在新列与原来的列之间增加一条竖直线，注意：这些效果只能在程序运行后才能看到
Prompt（提示）	用于指明光标移至该菜单项时，在状态栏上显示的提示信息

[例 2.2] 用编辑器设计一个顶层菜单

在顶层菜单栏里建立一个菜单项，并在其下面建立带有子菜单的菜单项，使子菜单具有加速键、变灰和核对符，又使每个子菜单都能显示信息。下面是其详细步骤。

（1）创建一个单文档的应用程序（或用例 2.1 程序），名为：山东旅游。

（2）建立菜单，包括以下几步。

1）打开 ResourceView 选项卡，打开 Menu 文件夹，双击 IDR_MAINFRAME，右边出现菜单编辑器，左键选中顶层最后的空白菜单不放，将其拖到“帮助”的前面，松开鼠标（或打开“帮助”菜单，按 Insert 键）。双击这个空白菜单，弹出“菜单属性对话框”，在 Caption 处写：山东旅游（&S），Pop_up 处于选中状态（屏蔽 ID），退出。

2）双击下面出现的空白菜单，弹出“菜单属性对话框”，选中 Pop_up（屏蔽 ID），Capton 处写：烟台（&Y）。

3）右边出现空白子菜单，双击它，弹出“菜单属性对话框”，在 ID 处写：ID_SD_YT_PL，在 Capton 处写：蓬莱 Ctrl +F5（注：Ctrl+F5 是加速键标识），在注释栏 prompt 处写：蓬莱仙境。

4）双击“蓬莱”下面的子菜单，弹出“菜单属性对话框”，在 ID 处写：ID_SD_YT_NS，在 Capton 处写：南山（&N），在 prompt 处写：南山大佛。

5）双击“烟台”下面的空白菜单，弹出“菜单属性”对话框，选中 Pop up（屏蔽 ID），在 Capton 处写：青岛（&Q）。

6）右边出现空白子菜单，双击它，弹出“菜单属性对话框”，在 ID 处写：ID_SD_QD_LS，在 Capton 处写：崂山 Ctrl +F6，在 Prompt 处写：崂山道士。

7）双击“青岛”下面的空白菜单，弹出“菜单属性对话框”，选中 Pop up（屏蔽 ID），在 Capton 处写：泰安（&T）。

8）右边出现空白子菜单，双击它，弹出“菜单属性”对话框，在 ID 处写：ID_SD_TA_TS，在 Capton 处写：泰山 Ctrl +F7，在 Prompt 处写：泰山日出。

9）双击“泰安”下面的空白菜单，弹出“菜单属性对话框”，选中 Pop up（屏蔽 ID），

在 Capton 处写：济南（&J）。

10）右边出现空白子菜单，双击它，弹出“菜单属性对话框”，在 ID 处写：ID_SD_JN_BTQ，在 Capton 处写：趵突泉 Ctrl +F8，在 Prompt 处写：天下第一泉。

（3）填加加速键表有以下几步。

1）打开项目工作区的 ResourceView（资源界面），打开 Accelerator 文件夹，双击 IDR_MAINFRAME，出现加速键表，双击最下面的空白格，如图 2.4 所示，弹出加速键属性对话框 Acel Properties，在 ID 下拉列表框选择 ID_SD_YT_PL 选项，在 key 下拉列表框选择 Vk_F5 选项（或置好 ID 后，单击 Next Key Typed 按钮，弹出一个小对话框后，再按 Ctrl+F5 组合键也可）如图 2.5 所示，这样就为“蓬莱”菜单置好了加速键。Accel Properties 对话框的各项含义如表 2.2 所示。

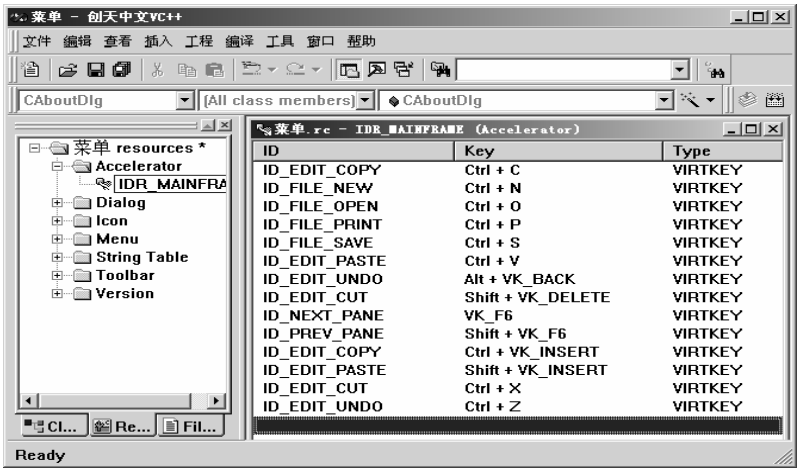

图 2.4　加速键资源列表

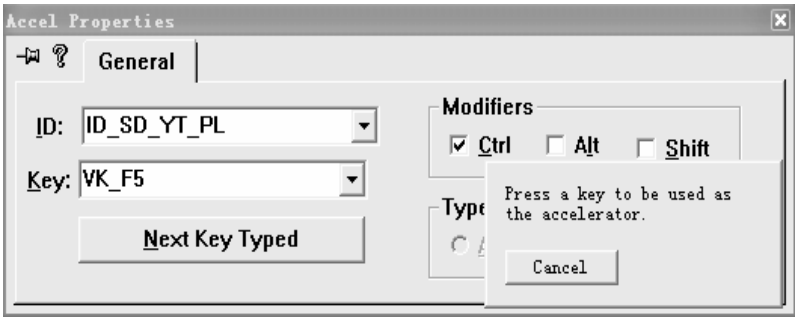

图 2.5　Accel Properties（加速键属性）对话框

2）再双击最下面的空白格，按上步的方法，分别将青岛崂山（ID_SD_QD_LS）、泰安泰山（ID_SD_TA_TS），济南趵突泉（ID_SD_JN_BTQ）菜单分别置好 VK_F6 ，VK_F7，VK_F8 的加速键。

注意：图 2.5 中 Modifiers 处，选中 Ctrl 复选框说明是 Ctrl 键，选中 Alt 复选框说明是 Alt 键，选中 Shift 复选框说明是 Shift 键，小对话框“Press a key to be used as the accelerator”是单击 Next Key Typed 按钮弹出来的。

表 2.2 Accel Properties 对话框的各项含义

项　目	含　义
ID	指定资源 ID 号的列表项，为了能和菜单联用，通常选择某菜单项的 ID 号
Modifiers	用于确定 Ctrl、Alt、Shift 是否是构成加速键的组成部分
Type	用于确定该加速键的值是虚拟键（VirKey），还是 ASCII
Key	是指启动加速键的键盘按键
Next Key Type	单击此按钮后，用户操作的任何按键将成为此加速键的键值

（4）使菜单变灰（不被激活，不起作用）：在 Message Maps 选项卡的 Class name 下拉列表框中选择 CMyView 选项，在 Object IDs 中找到想要变灰的菜单，这里选择 ID_SD_TA_TS（泰山）选项，在 Messages 中选择 UPDATE COMMAND UI（命令属性），依次单击 Add Function→OK→Edit Code 按钮，在该函数里写：

```
void CMyView::OnSdTaTs()
{  pCmdUI→Enable(false);   }
```

说明：

pCmdUI 是 CCmdUI 类的指针对象；CCmdUI 类只是被用在一个 CCmdTarget 派生类中的 ON_UPDATE_COMMAND_UI 处理程序中；当用户打开一个菜单时，每个菜单项都需要知道它应该被显示为可用还是禁用；当菜单被打开时，（框架）会寻找并调用各个 ON_UPDATE_COMMAND_UI 处理程序，每个处理程序都调用 CCmdUI 类中像 Enable 和 Check 这样的成员函数，然后框架将（按照其合适的方式）显示各个菜单项。

（5）核对菜单项，使这个菜单名字的前面加个“√”号。

在 Message Maps 选项卡的 Class name 下拉列表框中选择 CMyView，在 Object IDs 中找到你想要核对的菜单，这里选择 ID_SD_YD_NS（南山），在右边 Messages 里选择 UPDATE COMMAND UI，依次单击 AddFoution→OK→Edit Code 按钮，在该函数里写：

```
void CMyView::OnUpdateSdYtNs(CCmdUI* pCmdUI)
{  pCmdUI→Enable(true);//这里如果是Enable(false)则该菜单变灰
   pCmdUI→SetCheck(1);//设置核对符，如果括弧里写0是删除核对符
}
```

（6）菜单命令响应，包括以下几步。

1）在 View.h 里的 public:下定义变量：CString str；

在 View.cpp 的构造函数里将变量值赋空：str=“ ”；

2）将烟台的子菜单“蓬莱”的 ID 标识符 ID_SD_YT_PL 映射到视图类 View 里：在 CLassName 中选择 View（视图类），在 Object IDs 中选择 ID_SD_YT_PL，在 Messages 中选择选中 COMMAND，依次单击 Add Function→Edit Code 按钮。添加代码：

```
void CMyView::OnSdYtPl()
{  str="蓬莱仙境";
   Invalidate();
}
```

3）将烟台的子菜单“南山”的 ID_SD_YT_NS 映射到视图类 View 里，并添加代码：

```
void CMyView::OnSdYtNs()
{   str="南山大佛";
    Invalidate();
}
```

4）分别将青岛的子菜单“崂山”ID_SD_QD_LS、泰安的子菜单“泰山”ID_SD_TA_TS、济南的子菜单“趵突泉”ID_SD_JN_BTQ 映射到视图类中，并添加代码：

```
void CMyView::OnSdQdLs() //青岛崂山
{   str="崂山道士";
    Invalidate();
}
void CMyView::OnSdTaTs() //泰安泰山
{   str="泰山日出";
    Invalidate();
}
void CMyView::OnSdJnBtq()//济南趵突泉
{   str="天下第一泉";
    Invalidate();
}
```

5）在视图类实现文件 CMyView.cpp 的 OnDraw()函数里添加代码：

```
void CMyView::OnDraw(CDC* pDC)
{   CMyDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(50,50,str);
}
```

（7）编译运行，结果如图 2.1 所示。

1）打开“山东旅游”→“烟台”→“南山”子菜单，见其前面有“√”号。

2）打开“山东旅游”菜单，分别选择“蓬莱”、“南山”、“崂山”、“趵突泉”命令便出现各自的信息。而“泰山”是灰色的，不能显示信息。

3）程序运行后，出项空白窗口，按 Ctrl+F5、Ctrl+F6、Ctrl+F8 组合键，分别在窗口中显示“蓬莱仙境”，“崂山道士”，“天下第一泉”。Ctrl+F7 组合键变灰不能显示信息。

4）打开“山东旅游”菜单，按相应的助机记符（烟台）Y、（青岛）Q、（泰山）T、（济南）J 键，便出现各自的子菜单，将鼠标放在子菜单“南山”上，按助记符 N，便在文档窗口上显示出：南山大佛。

说明：

void Invalidate（BOOL bErase = TRUE）；该函数的作用是使整个窗口客户区无效。窗口的客户区无效意味着需要重绘，例如，如果一个被其他窗口遮住的窗口变成了前台窗口，那么原来被遮住的部分就是无效的，需要重绘。这时 Windows 会在应用程序的消息队列中放置 WM_PAINT 消息。MFC 为窗口类提供了

WM_PAINT 的消息处理函数 OnPaint，OnPaint 负责重绘窗口。

视图类有一些例外，在视图类的 OnPaint 函数中调用了 OnDraw 函数，实际的重绘工作由 OnDraw 来完成。参数 bErase 为 TRUE 时，重绘区域内的背景将被擦除，否则，背景将保持不变。

它和 UpdateWindow( )区别在于：

UpdateWindow( )的作用是使窗口立即重绘；调用 Invalidate 等函数后窗口不会立即重绘，这是由于 WM_PAINT 消息的优先级很低，它需要等消息队列中的其他消息发送完后才能被处理；调用 UpdateWindow 函数可使 WM_PAINT 被直接发送到目标窗口，从而导致窗口立即重绘。

2.1.2 菜单的编程控制

在交互式软件设计中，菜单有时会随着用户操作的改变而改变，这时的菜单就需要在程序中进行控制。MFC 提供的菜单类 CMenu 可在程序运行时，处理有关菜单的操作，如：创建菜单、装入菜单、删除菜单、获取菜单或设置菜单的状态等。下面给出了 CMenu 类的常用成员函数。

1. 创建菜单

CMenu 类的 CreateMenu()和 CreatePopupMenu()函数分别用于创建一个菜单或子菜单框架，它们的原型是：

```
BOOL CreateMenu();  产生一个空菜单
BOOL CreatePopupMenu();产生一个空的弹出式子菜单
```

2. 装入菜单

将菜单从资源装入应用程序中，需要调用 Cmenu 类成员函数 LoadMenu 或者用 SetMenu 对应用程序菜单进行重新设置。

```
BOOL LoadMenu(LPCTSTR lpszResourceName);
BOOL LoadMenu(UINT nIDResource);
```

参数：

lpszResourceName 表示菜单资源名称；

nIDResource 表示菜单资源 ID 标识号。

3. 添加菜单项

当菜单创建后，用户可以调用 AppendMenu 或 InsertMenu 函数来添加一些菜单项。但每次添加时，AppendMenu 是将菜单项添加在菜单的末尾处，而 InsertMenu 在菜单的指定位置处插入菜单项，并将后面的菜单项依次下移。

```
BOOL AppendMenu(UINT nFlags,UINT nIDNewItem=0,
                                                LPCTSTR lpszNewItem=NULL);
BOOL AppendMenu(UINT nFlags,UINT nIDNewItem,const CBitmap *pBmp);
BOOL InSertMenu(UINT nPosition,UINT nFlags,UINT nIDNewItem=0,
                                                LPCTSTR lpszNewItem=NULL);
BOOL InSertMenu(UINT nPosition,UINT nFlags,UINT nIDNewItem,
                                                const CBitmap *pBmp);
```

参数：

nIDNewItem 表示新菜单项的资源 ID 号；

lpszNewItem 表示新菜单项的内容；

pBmp 用于菜单项的位图指针；

nPosition 表示新菜单项要插入的菜单项位置；

nFlags 表示要增加的新菜单项的状态信息，其含义如表 2.3 所示。

表 2.3 nFlags 的值及其对其他参数的影响

nFlage 值	含义	nPosition 值	nIDNewItem 值	lpszNewItem
MF_BYCOMMAND	菜单项以 ID 标识符来标识	菜单项资源 ID		
MF_BYPOSITION	菜单项以位置来标识	菜单项的位置		
MF_POPUP	菜单项有弹出式子菜单		弹出式菜单句柄	
MF_SEPARATOR	分割线		忽略	忽略
MF_OWNERDRAW	自画菜单项			自画所需的数据
MF_STRING	字符串标志			字符串指针
MF_CHECKED	设置菜单项的选中标记			
MF_UNCHECKED	取消菜单项的选中标记			
MF_DISABLED	禁用菜单项			
MF_ENABLED	允许使用菜单项			
MF_GRAYED	菜单项灰显			

注意：

（1）当 nFlags 为 MF_BYPOSITION 时，nPosition 表示新菜单项要插入的具体位置，为 0 时表示第 1 个菜单项，为–1 时，将菜单项添加至菜单的末尾处；

（2）在 nFlags 的标志中，可以用“|”（按位或）来组合，例如 MF_CHECKED|MF_STRING 等。但有些组合是不允许的，例如 MF_DISABLED、MF_ENABLED 和 MF_GRAYED，MF_STRING、MF_OWNERDRAW、MF_SEPARATOR 和位图，MF_CHECKED 和 MF_UNCHECKED 都不能组合在一起；

（3）当菜单项增加、改变或删除后，不管菜单依附的窗口是否改变，都应调用 CWnd::DrawMenuBar 来更新菜单。

4. 删除菜单项

调用 DeleteMenu 函数可将指定的菜单项删除。函数 DeleteMenu 的原型如下：

```
BOOL DeleteMenu(UINT nPosition,UINT nFlags);
```

参数：

nPosition 表示要删除的菜单项位置，它由 nFlags 进行说明；

当 Flags 为 MF_BYCOMMAND 时，nPosition 表示菜单项的 ID 标识符；

当 Flags 为 MF_BYPOSITION 时，nPosition 表示菜单项的位置（第一个菜单项为 0）。

5. 获取菜单项

以下三个 CMenu 成员函数分别获得菜单的项数、菜单项的 ID 标识符以及弹出式子菜单的句柄。

```
UINT GetMenuItemCount()const;获得菜单项的项数，调用失败返回-1
UINT GetMenuItemID(int nPos)const;获得由 nPos 指定菜单项位置（以 0 为基数）的菜单项的标识号，若 nPos 是 SEPARATOR（分隔符）则返回-1
CMenu *GetSubMenu(int nPos)const;获得指定菜单的弹出式菜单的菜单句柄，该弹出式菜单位置由参数 nPos 指定，开始位置为 0，若选单不存在，则创建一个临时菜单指针
```

[例 2.3] 用编写程序的方法添加并处理一个新的菜单项

（1）创建一个单文档（SDI）应用程序（或用例 2.2 程序），名为：添加菜单项。

（2）在程序窗口中选择 View→ResourceSymbols 命令，弹出如图 2.6 所示的对话框。

（3）单击 New 按钮，弹出 New Symbol 对话框，在 Name（名字）文本框中输入一个新的标识符 ID_NEW_MENUITEM。在 Value（值）文本框中，输入该 ID 的值，如图 2.7 所示。系统要求用户定义的 ID 值应大于 15（0X000F）而小于 61440（0XF000）。这里保留默认的 ID 值 101，单击 OK 按钮。

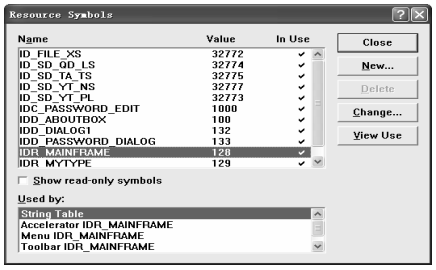

图 2.6 “资源符号”对话框

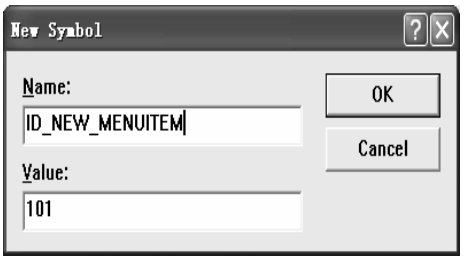

图 2.7 New Symbol 对话框

（4）关闭“资源符号”对话框，在 CMainFrame::OnCreate 函数中添加下列代码，该函数在框架窗口创建时自动调用。

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{  ……
   CMenu *pSysMenu=GetMenu();//获得主菜单句柄指针
   CMenu *pSubMenu=pSysMenu->GetSubMenu(1);//获得第二个子菜单指针
   CString StrMenuItem("新的菜单项");//字符串对象
   pSubMenu→AppendMenu(MF_SEPARATOR);//增加一水平分割线（如表 2.3 所示）
//在子菜单末尾增加一菜单项，允许使用 ON_UPDATE_COMMAND_UI 或 ON_COMMAND 的菜单
//项下面 MF_STRING 是字符串标志，ID_NEW_MENUITEM 是新的菜单标识符，StrMenuItem 是新菜单
//项内容
   pSubMenu->AppendMenu(MF_STRING,ID_NEW_MENUITEM,StrMenuItem);
   m_bAutoMenuEnable=FALSE;//关闭系统自动更新菜单状态，见下面的说明③
   pSysMenu->EnableMenuItem(ID_NEW_MENUITEM,MF_BYCOMMAND|
                                          MF_ENABLED);//激活菜单项
```

```
DrawMenuBar();//更新菜单
    return 0;
}
```

（5）编译运行，程序添加的菜单如图 2.8 所示，但此时只是将菜单项加上了，命令却无反映。

（6）使用 ClassWizard 在 CMainFrame 主框架类中添加 OnCommand 消息函数的重载，并检测用户菜单的 nID 参数：在 Class name 中选择 CMainFrame，在 Messages 处找到 OnCommand 消息，将其映射到 CMainFrame 里并添加代码：

```
BOOL CMainFrame::OnCommand(WPARAM wParam,LPARAM lParam)
{   //参数 wParam 的低字节表示菜单、控件、加速键的命令 ID
    if(LOWORD(wParam)==ID_NEW_MENUITEM)
    MessageBox("你选中了新的选单项");
}
```

（7）编译运行并测试。选择“编辑”→“新的选单项”命令；弹出如图 2.9 所示的对话框，显示“你选中了新的选单项”。

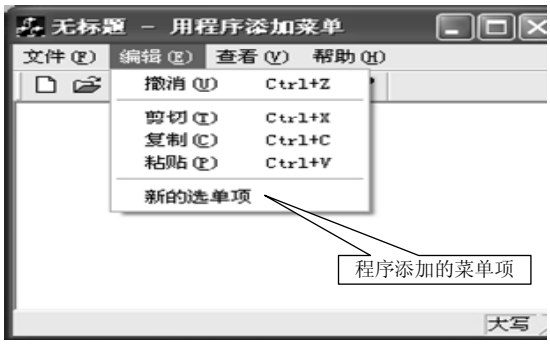

图 2.8　程序添加的菜单项

图 2.9　菜单命令执行结果

说明：

① WPARAM wParam，MPARAM lParam 参数，见第 1 章例 1.20 的说明。

② LOWORD 是获取其参数中的整型值位数。

下面是用 C++语言编写的程序，可以看到 LOWORD 是取 wParam 参数的右边的 4 位数，即菜单的 ID 值（是用资源符号创建菜单的 Value 框中写的 101）。

```
#include <windows.h>
// #include <stdio.h>
#include <iostream.h>
int main()
{   int iInWord = 101;
    cout<<hex<<iInWord<<endl;
    cout<<oct<<iInWord<<endl;
    cout<<dec<<iInWord<<endl;
```

```
    WORD usOutWord = LOWORD(iInWord);
    cout<<hex<<usOutWord<<endl;
    cout<<oct<<usOutWord<<endl;
    cout<<dec<<usOutWord<<endl;
    return 0;
}
```

结果：iInWord 是 65 （16 进制），145（8 进制），101（10 进制）

usOutWord 是 65 （16 进制），145（8 进制），101（10 进制）。

③ 资源符号：能对应用程序中的资源标识符进行管理。由于程序中要添加的菜单项需要一个标识值，最好用一个标识符来代替这个值，因此这里通过“资源符号”对话框来创建一个新的标识符。

④ m_bAutoMenuEnable：当此成员是可用的（默认），用户下拉一个菜单时，没有 ON_COMMAND 或 ON_UPDATE_COMMAND_UI 处理程序的菜单项目将被自动设置为无效。具有 ON_COMMAND 处理程序而无 ON_UPDATE_COMMAND_UI 处理程序的菜单项目将被自动设置为可用。当数据成员被设置时，菜单条与工具条按钮按照一样的方式被使用。此数据成员简化了基于当前选择的最优命令的实现，并减少为可用与无效菜单项目编写 ON_UPDATE_COMMAND_UI 处理程序的应用需求。

2.1.3 使用快捷菜单

快捷菜单是一种浮动弹出式菜单，它是一种新的用户界面设计风格，当用户右击时，就会相应地弹出一个浮动菜单，其中提供了几个与当前选择内容相关的选项。用资源编辑器和 MFC 库的 CMenu::TrackPopupMenu()函数可以很容易地创建这样的菜单，其原形为：

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

该函数用于显示一个浮动的弹出式菜单，其位置由各参数决定。

参数：

nFlags 表示菜单在屏幕显示的位置以及鼠标按钮标志，如表 2.4 所示；

x，y 表示菜单的水平坐标和菜单的顶端的垂直坐标；

pWnd 表示弹出菜单的窗口，此窗口将收到菜单的全部 WM_COMMAND 消息；

lpRect 是一个 RECT 结构或 CRect 对象指针，它表示一个矩形区域，用户单击这个区域时，弹出菜单不消失。当 lpRect 为 NULL 时，若用户在菜单外面单击，菜单立刻消失。

表 2.4 nFlags 的值及其对其他参数的影响

nFlags	含 义
TPM_CENTERALIGN	屏幕位置标志，表示菜单的水平中心位置由 x 坐标确定
TPM_LEFTALIGN	屏幕位置标志，表示菜单的左边位置由 x 坐标确定
TPM_RIGHTALIGN	屏幕位置标志，表示菜单的右边位置有 x 坐标确定
TPM_LEFTBUTTON	屏幕位置标志，表示当用户单击时弹出菜单
TPM_RIGHTBUTTON	屏幕位置标志，表示当用户右击时弹出菜单

［例 2.4］使用快捷菜单（截取“文件”下拉菜单中的选项，以快捷方式弹出）

（1）创建一个单文档的应用程序，名为：快捷菜单。

（2）用 MFC ClassWizard 在视图 CMyView 类添加 WM_CONTEXTMENU 消息映射，并添加代码：

```
void CMyView::OnContextMenu(CWnd* pWnd, CPoint point)
{  //获得主窗口指针
   CMainFrame *pFrame=(CMainFrame*)AfxGetApp()→m_pMainWnd;
   CMenu *pSysMenu=pFrame->GetMenu();  //获得程序窗口菜单指针
   int nCount=pSysMenu->GetMenuItemCount();//获得顶层菜单个数
   int nSubMenuPos=-1;  //给特征变量送个标记，下面要用
   for(int i=0;i<nCount;i++)  //查找“文件”菜单
   { CString str;
//将指定菜单项的标签复制到指定的缓冲区。如设置的是 MF_BYPOSITION（如表 2.3 所示）,这个参
//数就代表菜单条目在菜单中的位置（第一个菜单条目的位置为零）
   pSysMenu→GetMenuString(i,str,MF_BYPOSITION);
     if(str.Left(4)=="文件")
        {  nSubMenuPos=i; //将菜单的条目号送给特征变量
           break;
        }
   }
      if(nSubMenuPos<0) return;  //没有找到返回
      pSysMenu->GetSubMenu(nSubMenuPos)      //获得"文件"下面的子菜单
          ->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON,  //如表 2.4 所示
          point.x,point.y,this);//在屏幕的任意地方显示一个弹出式菜单
}
```

（3）在视图的执行文件“快捷菜单 View.cpp”文件头部加：#include “MainFrm.h”。

（4）编译运行，在应用程序窗口的客户区中右击，会弹出快捷菜单（将原 File 下拉的菜单弹出来），如图 2.10 所示。

图 2.10　例 2.4 快捷菜单显示结果

说明：

① WM_CONTEXTMENU：鼠标右键按下时发送的消息。

② GetMenuString（i，str，MF_BYPOSITION）；将指定菜单项的标签复制到指定的缓冲区。如果其最后的参数为 MF_BYCOMMAND，则其他参数指定菜单项标识符。如果其最后的参数为 MF_BYPOSITION，则其他参数指定菜单项位置。如果其最后的参数为：

MF_BYPOSITION，这个参数就代表菜单条目在菜单中的位置（第一个菜单条目的位置为零）。

③ AfxGetApp () 函数：是获取应用程序实例指针；GetMainWnd()函数：是获取主窗口对象指针。这两个函数可

以合成一个：afxgetmainwnd();。通常把一些重要的工程一开始就需要初始化的并且在其他地方类中都要用到的变量或函数定义在 C***App 类中，然后通过此函数获得这些变量或函数。

由于菜单、工具栏、状态栏是由主框架类 CMainFrame 来控制的，虽在视图类可以添加快捷菜单消息映射，但若要在视图类中访问应用程序的主框架窗口的系统菜单，则必须通过 AfxGetApp 来获取主框架类对象指针后才能获取相应的菜单。AfxGetApp 是 CWinApp 类的一个成员函数，该函数可在应用程序项目中的任何类中使用，用于获取应用程序中的 CWinApp 类对象指针。

[例 2.5] 使用快捷菜单（以快捷方式弹出自己设计的菜单项）

（1）创建一个单文档应用程序，名为：建立快捷菜单。

（2）选择 Insert→Resource 命令，选择 Menu，单击 New 按钮，便在 Menu 资源下出现一个新菜单资源（默认的 ID 号为 IDR_MENU1），将此 ID 号改为：IDR_MYFLOATMENU。

（3）双击右边出现的空白菜单项，弹出 Menu Item Properties 对话框，在对话框中选中 Pop up，在 Caption 处写：弹出快捷菜单，关闭对话框，打开下面的子菜单，依次添加如表 2.5 所示的子菜单项。

表 2.5 添加的子菜单项

菜单 ID	标　题	属　性
ID_MENU_SCOREIN	成绩输入（&S）	默认
ID_MENU_SCOREPRINT	成绩打印（&P）	默认
ID_SEPARATRO		选中 Separator
…	其他（&Q）	选中 Pop_up，其余默认

（4）将 ID_MENU_SCOREIN 的 COMMAND 消息映射到主框架类 MainFrame 中（如果要弹出提问对话框，选择 Select a new class，弹出 Select Class 对话框，选择对话框上的 CMainFrame→Select，回到 MFC ClassWizard，接着加如上 ID 的 COMMAND 消息）。

在 Classname 下拉列表框中选择 CMainFrame，将上面的 ID 分别加 COMMAND 消息。这里仅添加 ID_MENU_SCOREIN 的 COMMAND 消息，并添加如下代码：

```
void CMainFrame::OnMenuScorein()
{  AfxMessageBox("现在就输入成绩吗?");  }
```

（5）在 CMainFrame 类加入 WM_CONTEXTMENU 消息处理函数，添加代码：

```
void CMainFrame::OnContextMenu(CWnd *pWnd,CPoint point)
{  CMenu menu;
   menu.LoadMenu(IDR_MYFLOATMENU);//刚才加上的菜单资源
   menu.GetSubMenu(0)->TrackPopupMenu(TPM_LEFTALIGN|
                                TPM_RIGHTBUTTON,point.x,point.y,this);
}
```

（6）运行并测试，在出现的应用程序窗口中右击，会弹出创建的快捷菜单，如图 2.11 所示。再选择“成绩输入”命令，会弹出写有“现在就输入成绩吗?”的对话框。

2.2 工具栏

工具栏是一系列工具按钮的组合，借助它们可以提高用户的工作效率。Visual C++6.0 系统保存了每个工具栏相应的位图，其中包括所有按钮的图像，而所有的按钮图像具有相同的尺寸（15 像素高，16 像素宽），它们在位图中的排列次序与屏幕上的按钮在工具栏上的次序相同。

图 2.11 例 2.5 快捷菜单显示结果

2.2.1 使用工具栏编辑器

在项目工作区中打开 ResourceView 选项卡，双击 Toolbar 项目中的 IDR_MAINFRAME，则工具栏编辑器出现在主界面的右边，如图 2.12 所示。

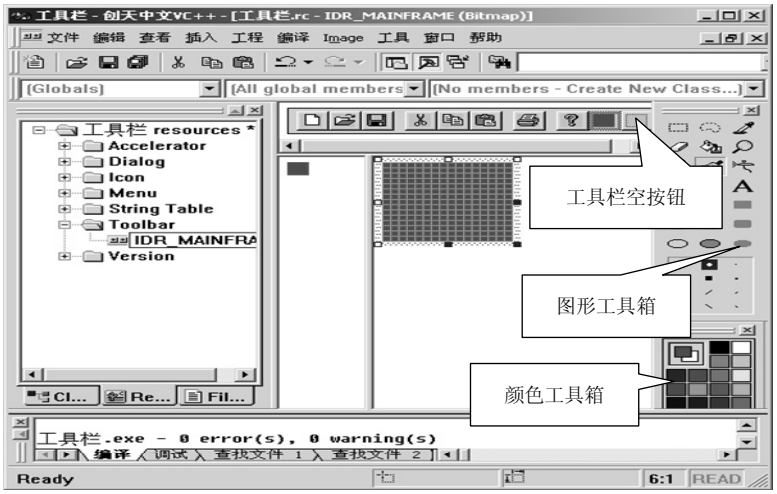

图 2.12 工具栏编辑器

［例 2.6］工具栏按钮和菜单联用

（1）创建一个 SDI 单文档应用程序，名为：工具栏。

（2）在 ResourceView 页打开 Toobar 文件夹，单击 IDR_MAINFRAME，则工具栏编辑器出现在主界面的右边，单击最后一个空的工具栏按钮（创建完该工具栏按钮后，其后面又会自动出现一个新的空工具栏按钮），单击颜色工具箱的红色，再点图形工具箱的画刷，之后将鼠标移到下面较大的方块图上，来回移动鼠标将这个按钮涂红色后，如图 2.12 所示，再双击上面这个按钮，弹出属性对话框，如图 2.13 所示，在 ID 处写：ID_TOOLBAR，在“M 提示”（注释）框内输入：建立新文档\n 新建。

（3）将 ID_TOOLBAR 映射到 View 视图类中，并添加代码：

```
void CMainFrame::OnToolbar()
```

```
{   MessageBox("工具栏显示");  }
```

（4）编译运行鼠标放在刚创建的工具栏上出现“新建”框，而一会儿便消失。在状态栏上出现“建立新文档”字样。再单击这个按钮，在文档窗口出现小对话框，上面写的是“工具栏显示”。

图 2.13 工具栏按钮属性对话框

（5）创建一个菜单项，在 ResourceView 页打开 Menu，双击 IDR_MAINFRAME，再双击右边出现的空白菜单，在标题处写：和工具栏联用，在 ID 处写（和工具栏的 ID 一样）：ID_TOOLBAR，运行它，则点这个菜单和点刚才创建的工具栏的显示结果是一样的（这就是工具栏和菜单相结合）。鼠标左键按住这个菜单不放，状态栏出现“建立新文档”，放开鼠标，状态栏的“建立新文档”没了，但在文档窗口出现一个小对话框，和点刚才建的工具栏按钮一样，上面也写着“工具栏显示”。

说明：

1）移动一个工具栏按钮：用鼠标左键按住这个按钮，拖动至相应位置即可。

2）删除一个工具栏按钮：用鼠标左键按住这个按钮，拖动它离开工具栏。

3）复制一个工具栏按钮：若在移动一个按钮的同时，按下 Ctrl 键，则在新位置复制一个按钮。

4）在工具栏中插入空格有以下几种情况。

① 如果工具栏按钮前没有空格，拖动该按钮向右移动并当覆盖相邻按钮的一半以上时，释放鼠标，则此按钮前出现空格。

② 如果工具栏按钮前有空格而按钮后没空格：拖动该按钮向左移动并当按钮的左边界接触到前面按钮时，释放鼠标键，则此按钮后出现空格。

③ 如果工具栏按钮前后均有空格，拖动该按钮向右移动并当接触到相邻按钮时，则此按钮前的空格保留，按钮后的空格消失。相反，拖动该按钮向左移动并当接触到前一个相邻按钮时，则此按钮前面的空格消失，后面的空格保留。

5）工具栏按钮属性的设置：双击某按钮弹出“工具栏按钮属性”对话框，如图 2.13 所示。“工具栏按钮属性”对话框中的各项含义如表 2.6 所示。

表 2.6 工具栏按钮属性对话框中的各项含义

项 目	含 义
ID	工具栏按钮的标识符，可以输入自己的标识符名称，也可从 ID 的下拉列表中选取标识符名称
Width（宽）	工具栏按钮的像素宽度
Height（高）	工具栏按钮的像素高度
Prompt（提示）	工具栏按钮提示文本：若为“建立新文档\n 新建”，则表示将鼠标指向该按钮时，在状态栏中显示“建立新文档”，而在弹出的提示信息中出现“新建”字样。“\n”是它们的分割转义符

2.2.2 多个工具栏的使用

实际应用中，常常需要多个工具栏，下面就讨论多个工具栏的创建、显示和隐藏，以及多个工具栏和菜单之间的联动操作等。

［例 2.7］使用多个工具栏

（1）添加并更改应用程序菜单

1）创建一个单文档应用程序，名为：多个工具栏。

2）打开 Resource 页，在资源类型中选中 Menu（或按 Ctrl+R 组合键），单击 New 按钮，便在右边出现一个空菜单，系统给的默认 ID 为 IDR_MENU1，如图 2.14 所示。

3）在 Menu 资源的 ID_MENU1 上右击，从弹出的快捷菜单中选择 Properties 命令，弹出如图 2.15 所示的 Menu Properties 对话框，在这里可以重新指定菜单资源 ID，设置菜单资源的语言和条件。这个条件用于决定菜单资源包含到哪个环境中，例如当指定条件为_DEBUG，则菜单资源只存于 Debug 编译环境中。

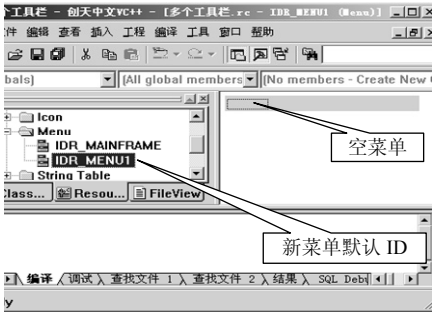

图 2.14 添加菜单资源后开发环境

图 2.15 Menu Properties 对话框

4）为菜单 ID_MENU1 添加一个顶层弹出菜单项：测试（&T），并在该菜单下添加一个子菜单：返回（&R），ID 设为 ID_TEST_RETURN，如图 2.16 所示。注意：“测试（&T）”中的符号“&”用于指定后面的字符（T）是一个助记符。

5）打开此程序的菜单资源 Menu，双击 IDR_MAINFRAME，在“查看”菜单的最后添加一个子菜单项：显示测试菜单（&M），将 ID 设为：ID_VIEW_TEST，如图 2.17 所示。

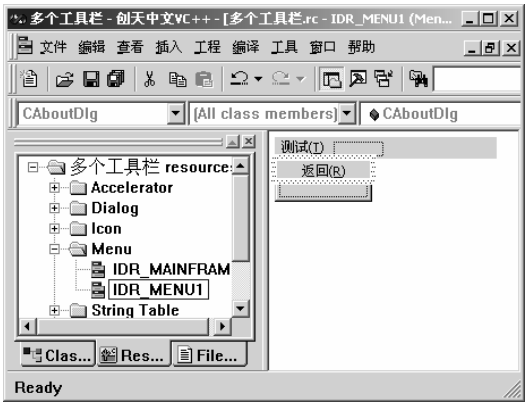

图 2.16 设计新的菜单资源

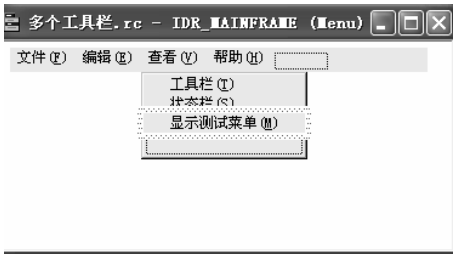

图 2.17 在“查看”下面建一个菜单项

6）在 CMainFrame 类头文件 MainFrm.h 中的 public 下添加成员变量：

```
CMenu m_NewMenu;
```

7)选择 View→ClassWizard(或按 Ctrl+W 组合键)命令，在弹出的对话框中切换到 Message Maps 选项卡，从“Classname”列表中选择 CMainFrame，分别为菜单项 ID_VIEW_TEST 和 ID_TEST_RETURN 添加 COMMAND 消息映射，使用默认的消息映射函数名，并添加代码：

```
void CMainFrame::OnViewTest()
{  m_NewMenu.Detach();   //使菜单对象和菜单句柄分离
   m_NewMenu.LoadMenu(IDR_MENU1); //将菜单从资源装入应用程序中
   SetMenu(NULL);        //清除应用程序菜单
   SetMenu(&m_NewMenu);  //设置应用程序菜单
}
void CMainFrame::OnTestReturn()
{  m_NewMenu.Detach(); //使菜单对象和菜单句柄分离
   m_NewMenu.LoadMenu(IDR_MAINFRAME);
   SetMenu(NULL);
   SetMenu(&m_NewMenu);
}
```

说明：

代码中，LoadMenu 和 Detach 都是 CMenu 类成员函数，LoadMenu 用于装载菜单资源，而 Detach 是使菜单对象与菜单句柄分离，在调用 LoadMenu 后，菜单对象 m_NewMenu 就拥有一个菜单句炳，当再次调用 LoadMenu 时，由于菜单对象的句柄已经创建，因而会发生运行错误，但当菜单对象与菜单句柄分离后，就可以再次创建菜单了。SetMenu 是 CWnd 类的一个成员函数，用于设置应用程序的菜单。

8）编译运行：单击菜单进行查看，显示测试菜单，菜单上面出现“测试”字样，选择“测试”→“返回”命令，又回到原菜单。

（2）添加并设计工具栏按钮

1）在 ResourceView 页打开 resources，再打开 Toolbar，双击 IDR_MAINFRAME，显示出工具栏编辑器，利用工具栏编辑器设计两个工具栏按钮，设计结果如图 2.18 所示。

图 2.18 设计的两个工具栏按钮

2）双击设计的第 1 个工具栏按钮，弹出该工具栏按钮的属性对话框，将该工具栏按钮的 ID 号设为：ID_VIEW_TEST，在提示框内输入：“显示测试菜单\n 显示测试菜单”。

3）再双击设计的第 2 个工具按钮，弹出该工具栏按钮的属性对话框，将该工具栏按钮的 ID 号设为 ID_TEST_RETURN，在提示框内输入：“返回应用程序主菜单\n 返回主菜单”。

4）再编译运行后，将鼠标移至第 1 个工具栏按钮处，这时在状态栏上显示出“显示测试菜单”，稍等片刻后，还会出现一个小窗口，也显

图 2.19 显示测试菜单

示“显示测试菜单”字样。再将鼠标移至第 2 个工具栏按钮处，在状态栏上显示“返回应用程序主菜单”，稍等片刻后，又会出现一个小窗口，显示“返回主菜单”，如图 2.19 所示。先单击第 1 个按钮，再单击第 2 个按钮，会各自执行和菜单一样的命令。

（3）添加工具栏

1）在项目工作区的 ResourceView 页面中，展开 Toolbar（工具栏）资源，用鼠标单击 IDR_MAINFRAME 不松开，然后按下 Ctrl 键，移动鼠标将 ID_MAINFRAME 拖到 Toolbar 资源名称上，松开鼠标和 Ctrl 键，这样就复制了工具栏默认资源：ID_MAINFRAME，复制后的资源标识，系统自动设为：IDR_MAINFRAME1。

2）右击工具栏资源 IDR_MAINFRAME1，从弹出的“工具栏属性”对话框中选择 Properties 命令，将 ID 设为 IDR_TOOLBAR1，如图 2.20 所示。

3）双击 IDR_TOOLBAR1，打开工具栏资源，删除（鼠标按住原有按钮不放，将其拖出工具栏即可）不要的工具栏按钮，如图 2.21 所示。

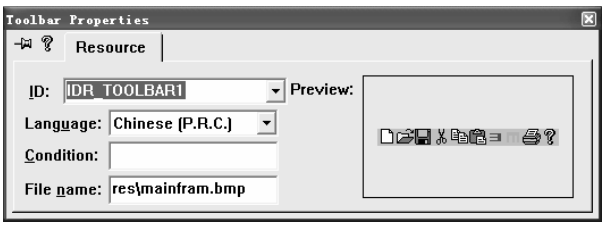

图 2.20 “工具栏属性”对话框

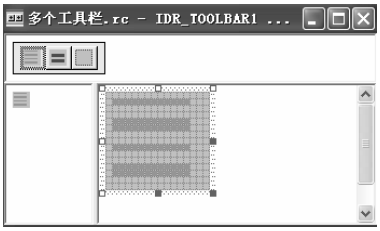

图 2.21 删除不要的工具按钮

4）在 CMainFrame 类头文件 CMainFrm.h 的 public 下添加成员变量：

```
CToolBar m_wndTestBar; // CToolBar 类封装了工具栏的操作
```

5）在 CMainFrame::OnCreate 函数中添加如下工具栏创建代码：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{ if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
   return -1;
   int nRes = m_wndTestBar.CreateEx(this,TBSTYLE_FLAT,
            WS_CHILD|WS_VISIBLE|CBRS_TOP|CBRS_GRIPPER|CBRS_TOOLTIPS|
                CBRS_FLYBY|CBRS_SIZE_DYNAMIC,CRect(0,0,0,0),
                        AFX_IDW_TOOLBAR+10);
   if(!nRes||!m_wndTestBar.LoadToolBar(IDR_TOOLBAR1))
   { TRACE0("Failed to create toolbar\n");
     return -1;
   }
```

```
    ……
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    m_wndTestBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
    DockControlBar(&m_wndTestBar);
    return 0;
}
```

6）编译运行，结果如图 2.22 所示，单击原工具栏新加的两个按钮和复制的两个工具栏钮，都执行和菜单一样的命令。

图 2.22 运行结果

说明：

① CreateEx(this，TBSTYLE_FLAT,WS_CHILD|WS_VISIBLE|CBRS_TOP|CBRS_GRIPPER|CBRSTOOLTIPS|CBRS_FLYBY|CBRS_SIZE_DYNAMIC,CRect(0,0,0,0),AFX_IDW_TOOLBAR+10)；创建工具栏

第一个参数用于指定工具栏所在的父窗口指针（this 表示当前的 CMainFrame 类窗口指针）。

第二个参数用于指定工具按钮的风格（TBSTYLE_FLAT 表示工具按钮是“平面”的）。

第三个参数用于指针工具栏的风格，由于这里的工具栏是 CMainFrame 的子窗口，因此需要指定 WS_CHILD|WS_VISIBLE（见第 4 章表 4.1：创建子窗口|窗口最初是可见的）。

CBRS_TOP	将工具栏放在边框窗口的顶部
CBRS_BOTTOM	将工具栏放在边框窗口的底部
CBRS_GRIPPER	表示工具栏前面有一个“把手”
CBRS_SIZE_DYNAMIC	表示工具栏在浮动时，其大小是可以动态改变的
CBRS_NOALIGN	边框窗口改变大小时，工具栏不重定位
CBRS_TOOLTIPS	使工具栏提示有效
CBRS_SIZE_FIXED	工具栏尺寸固定
CBRS_FLOATIONG	工具栏是浮动的
CBRS_FLYBY	在状态栏中显示按钮的有关信息
CBRS_HIDE_INPLACE	不显示工具栏

第四个参数用于指定工具栏四周的边框大小，一般都为 0，如：CRect（0，0，0，0）。

最后一个参数用于指定工具栏这个子窗口的标识 ID（与工具栏资源标识不同）。

② if 语句中的 LoadToolBar 函数用于装载工具栏资源。若 CreateEx 或 LoadToolBar 的返回值为 0，既调用不成功，则显示诊断信息“Failed to create toolbar”。

TRACEO 是一个用于程序调试的跟踪宏 OnCreate 函数返回-1 时，主框架窗口被清除。

③ 应用程序中的工具栏一般具有停靠或浮动性：

m_wndTestBar.EnableDocking 使得 m_wndTestBar 对象可以停靠。

CBRS_ALIGN_ANY 表示可以停靠在窗口的任一边。

EnableDocking（CBRS_ALIGN_ANY）调用的是 CFrameWnd 类的成员函数，用于让工具栏或其他控制条在主框架窗口可以进行停靠操作。

DockControlBar 也是 CFrameWnd 类的成员函数，用于将指定的工具栏或其他控制条进行停靠。

④ 代码中的 AFX_IDW_TOOLBAR 是系统内部的工具栏子窗口标识，并将：

AFX_IDW_TOOLBAR+1 的值表示默认的状态栏子窗口标识。如果在创建新的工具栏时没有指定相应的子窗口标识，则会使用默认的 AFX_IDM_TOOLBAR。这样，当选择“查看”菜单中的“工具栏”命令时，显示或隐藏的工具栏不是原来的工具栏，而是新添加的工具栏。为此，需要重新指定工具栏子窗口的标识，并使其值等于 AFX_IDW_TOOLBAR+10。

（4）完善程序代码

1）调用 CFrameWnd 类的成员函数 ShowControlBar，使程序一开始运行时，将工具栏 IDR_TOOLBAR1 隐藏起来。

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{      … …
  ShowControlBar(&m_wndTestBar,FALSE,FALSE);//关闭测试工具栏
  return 0;
}
```

说明：

ShowControlBar 函数有 3 个参数，第 1 个参数用于指定要操作的工具栏或状态栏指针，第 2 个参数是一个布尔型，当为 TRUE 时表示显示，否则表示隐藏，第 3 个参数用于表示是否延迟显示或隐藏，当为 FALSE 时表示立即显示或隐藏。

2）在 CMainFrame::OnViewTest 和 CMainFrame::OnTestReturn 函数中添加下列代码：

```
void CMainFrame::OnViewTest()
{  … …
   ShowControlBar(&m_wndTestBar,TRUE,FALSE); //显示测试工具栏
   ShowControlBar(&m_wndToolBar,FALSE,FALSE);//关闭主工具栏
}
void CMainFrame::OnTestReturn()
{  ShowControlBar(&m_wndTestBar,FALSE,FALSE);//关闭测试工具栏
   ShowControlBar(&m_wndToolBar,TRUE,FALSE); //显示主工具栏
}
```

3）编译运行：结果如图 2.23 和图 2.24 所示。

图 2.23　程序运行的结果

图 2.24　单击前面的工具按钮后结果

2.3　状态栏

状态栏是一条水平长条，位于应用程序的主窗口的底部，它可以分割成几个窗格，用来显示多组信息。应用程序往往需要把当前的状态信息或提示信息告诉用户，虽然其他窗口（如窗口的标题栏上、提示窗口等）也可显示文本，但它们的功能比较有限，而状态栏能很好地满足应用程序显示信息的需求。

2.3.1　状态栏的定义

在 MFC AppWizard 创建的 SDI 或 MDI 应用程序框架的 MainFrm.cpp 文件中有一个静态数组 indicators 数组，它被 MFC 用做状态栏的定义，如图 2.25 所示。

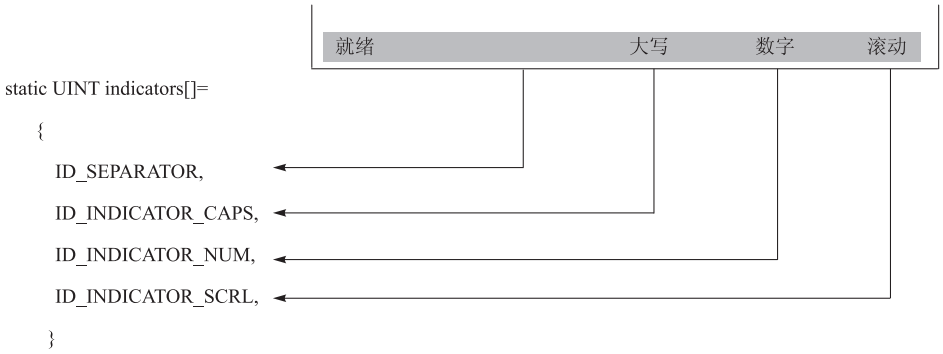

图 2.25　indicators 数组元素与标准状态栏窗口的关系

这个数组中的元素是一些标识常量或是字串资源的 ID 标识符。默认的 indicators 数组包含了以下 4 个元素。

ID_SEPARATOR：是用于标识信息行窗格的，菜单项或工具栏按钮的许多信息都在这个信息行窗格中显示。

ID_INDICATOR_CAPS：是用于标识指示器窗格显示出 CapsLock 键的状态（大写），CapsLock 键就是在键盘左边的那个控制大小写转换的键。

ID_INDICATOR_NUM：是用于标识指示器窗格显示出 NumLock 键状态（数字）。

ID_INDICATOR_SCRL：是用于标识指示器窗格显示出 ScrollLock 键的状态（滚动）。

2.3.2 状态栏的常用操作

Visual C++6.0 中可以方便地对状态栏进行操作，如增加窗格、减少窗格、在状态栏中显示文本、改变状态栏的风格大小等，并且 MFC 的 CStatusBar 类封装了状态栏的大部分操作。

1. 增加和减少窗格

状态栏中的窗格可以分为信息行窗格和指示器窗格两类。

（1）若在状态栏中增加一个信息行窗格，则只需要在 indicators 数组的适当位置增加一个 ID_SEPARATOR 标识即可。

（2）若在状态栏中增加一个用户指示器窗格，则在 indicators 数组中的适当位置增加一个在字符串表中定义过的资源 ID，其字符串的长度表示用户指示器窗格的大小。

（3）若状态栏减少一个窗格，其操作与增加相类似，只需减少 indicators 数组元素即可。

2. 在状态栏上显示文本

（1）调用 CWnd::SetWindowText 更新信息行窗格（或窗格 0）中的文本。由于状态栏也是一种窗口，故在使用时可直接调用。若状态栏变量为 m_wndStatusBar，则显示为 m_wndStatusBar。SetWindowText（“消息”）语句将在信息行窗格（或窗格 0）内显示“消息”字样。

（2）手动处理状态栏的 ON_UPDATE_COMMAND_UI 更新消息，并在处理函数中调用 CCmdUI::SetText 函数。

（3）调用 CStatusBar::SetPaneText 函数更新任何窗格（包括信息行窗格）中的文本，此函数原型描述如下：

```
BOOL SetPaneText(int nIndex,LPCTSTR lpszNewText,BOOL bUpdate=TRUE);
```

参数：

① nIndex 是表示设置的窗格索引（第 1 个窗格的索引为 0）；

② lpszNewText 表示要显示的字符串，若 bUpdate 为 TRUE，则系统自动更新显示的结果。下面用两种方法在状态栏中显示鼠标在客户区的位置。

［例 2.8］在状态栏的最右边两个窗格中显示出当前鼠标在窗口客户区的位置

（1）创建一个单文档应用程序（或用上文中“多个工具栏”程序），名为：状态栏。

（2）将项目工作区切换到 ClassView 选项卡，展开 CMainFrame 所有项，双击 CMainFrame() 函数，在文档窗口中出现该函数的定义，在它的前面是状态栏数组的定义。

（3）将状态栏 indicators 数组的定义改为下列代码：

```
static UINT indicators[] =
{   ID_SEPARATOR,          // status line indicator
    ID_SEPARATOR,
 };
```

（4）将鼠标移动消息 WM_MOUSEMOVE 映射到视图类“状态栏 View”中。

由于鼠标移动消息 WM_MOUSEMOVE 在 CMainFrame 类映射后不起作用，因此只能映射到视图“状态栏 View”类中。但是，这样一来，就需要更多的代码，因为状态栏对象 m_wndStatusBar 是 CMainFrame 类定义的成员变量，因而需要在视图类“状态栏 View”中添

加访问 CMainFrame 类的代码。

```
void CNnView::OnMouseMove(UINT nFlags, CPoint point)
{   CString str;
    //获得主窗口指针
     CMainFrame *pFrame=(CMainFrame*)AfxGetApp()->m_pMainWnd;
   //获得主窗口中的状态栏指针
    CStatusBar *pStatus=&pFrame->m_wndStatusBar;
    if(pStatus)
     {  str.Format("X=%d,Y=%d",point.x,point.y);//格式化文本
        //这里"1"是更新第 2 个窗格的文本，而"0"是第 1 个窗格
        pStatus->SetPaneText(1,str);
      }
}
```

（5）将 MainFrm.h 文件中的受保护变量 m_wndStatusBar 改为公共变量。

（6）在“状态栏 View.cpp”视图类执行文件的头部写：#include “MainFrm.h”。

（7）编译运行，结果如图 2.26 所示。

3. 改变状态栏的风格

在 MFC 的 CStatusBar 类中，有两个成员函数可以改变状态栏风格，它们是：

```
void SetPaneInfo(int nIndex,UINT nID,UINT nStyle,int cxWidth);
viod SetPaneStyle(int nIndex,UINT nStyle);
```

参数：

nIndex 表示要设置的状态栏窗格的索引；

nID 用于为状态栏窗格指定新的 ID；

cxWidth 表示窗格的像素宽度；

nStyle 表示窗格的风格类型，用于指定窗格的外观，例如 SBPS_POPOUT 表示窗格是凸起来的，状态栏窗格的风格类型如表 2.7 所示。

表 2.7　状态栏窗格的风格类型

风格类型	含　义
SBPS_NOBORDERS	窗口周围没有 3D 边框
SBPS_POPOUT	反显边界以使文字“凸出来”
SBPS_DISABLED	禁用窗格，不显示文本
SBPS_STRETCH	拉伸窗格，并填充窗格不用的空白空间。但状态栏只能有一个窗格具有这种风格
SBPS_NORMAL	普通风格，它没有“拉伸”，“3D 边框”或“凸出来”等特性

（8）在上面的 OnMouseMove（UINT nFlags，CPoint point）函数里添加：

```
void CNnView::OnMouseMove(UINT nFlags, CPoint point)
  {      … …
  //下面函数第一个参数表示状态栏窗格索引，第二个参数如表 2.7 所示
```

```
    pStatus->SetPaneStyle(1,SBPS_POPOUT);
    str.Format("X=%d,Y=%d",point.x,point.y); //格式化文本
     pStatus->SetPaneText(1,str);              //更新第 2 个窗格的文本
}
```

编译运行，见状态栏的第 2 个窗格凸起来了，如图 2.27 所示。

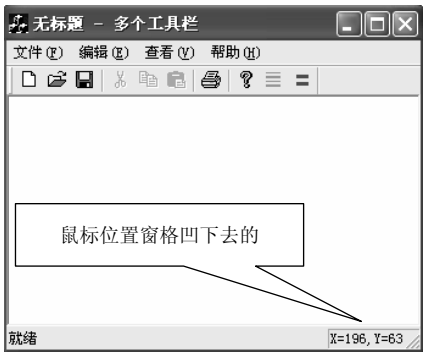

图 2.26 鼠标的位置显示在状态栏上

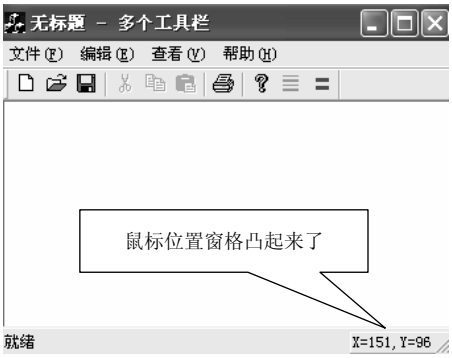

图 2.27 改变状态栏的风格

[例 2.9] 用与例 2.8 不同的方法，在状态栏的最右边两个窗格中显示出当前鼠标在窗口客户区的位置

（1）创建一个单文档的应用程序，名为：状态栏风格。

（2）将项目工作区窗口切换到 ResourceView 选项卡，双击 String Table 项的 String Table 图标，则在主界面的右边出现字符串编辑器。在字符串列表的最后一行的空项上双击，弹出一个对话框，如图 2.28 所示。

图 2.28 “字符串属性”对话框

（3）在该对话框中，用户可以指定相应的 ID 和字符串值，这里加入两个 ID 和其字符串资源，即：ID_LEFT（在 Caption 处写：X=999）和 ID_RIGHT（在 Caption 处写：Y=999），其字符的多少决定窗格的大小，其结果如图 2.29 所示。

（4）打开 MainFrm.cpp 文件，将原先的 indicators 数组修改如下：

```
static UINT indicators[] =
{  ID_SEPARATOR,          //第一个信息行窗格
   ID_SEPARATOR,          //第二个信息行窗格
   ID_LEFT,               //第三个窗格
```

```
    ID_RIGHT,                   //第四个窗格
};
```

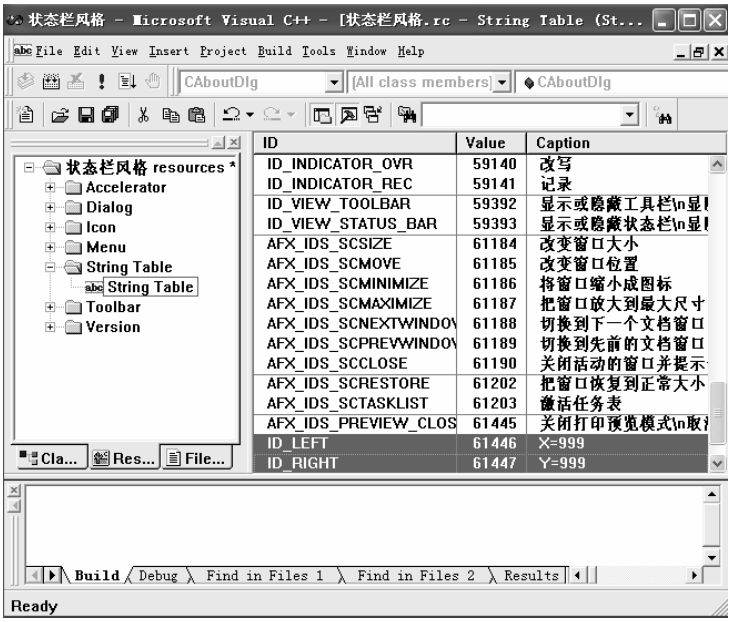

图 2.29 添加的字符串资源

（5）由于 ClassWizard 不能组织相应的命令更新消息的映射，用户必须手工添加消息处理函数原型。打开 CMyView.h 文件，在 AFX_MSG 内增加消息处理语句，ClassWizard 以后允许用户访问和编辑该代码。

```
protected:
//{{AFX_MSG(CQqView)
 // NOTE - the ClassWizard will add and remove member functions here.
     afx_msg void OnUpdateXY(CCmdUI *pCmdUI);
 //    DO NOT EDIT what you see in these blocks of generated code !
//}}AFX_MSG
```

说明：

CCmdUI 没有基类，和 CCmdTarget 派生类的 ON_UPDATE_COMMAND_UI 句柄一起使用，当用户选择一个菜单，每个菜单项需要知道自己是显示成可以使用的还是不可以使用的，菜单项通过执行 ON_UPDATE_COMMAND_UI 局柄来实现它；

当菜单被选择时，框架寻找并且唤醒 ON_UPDATE_COMMAND_UI 句柄，每一个句柄调用 CCmdUI 的一个功能（如 Enable and Check），然后框架就相应的现实每一个菜单项，一个菜单项不用改变实现 ON_UPDATE_COMMAND_UI 的代码就能够用按钮或快捷键代替。

（6）打开 View.cpp 文件，在其消息映射入口处添加消息映射宏函数：

```
BEGIN_MESSAGE_MAP(CQqView, CView)
  //{{AFX_MSG_MAP(CQqView)
    // NOTE - the ClassWizard will add and remove mapping macros here.
```

```
    ON_UPDATE_COMMAND_UI(ID_LEFT,OnUpdateXY)
    ON_UPDATE_COMMAND_UI(ID_RIGHT,OnUpdateXY)
  //    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG_MAP
```

（7）在视图类 CMyView.cpp 文件的末尾，增加修改状态栏指示器的消息映射函数代码，当状态栏的窗格需要更新时，应用程序框架自动调用此函数（全用手写）。

```
void CMyView::OnUpdateXY(CCmdUI *pCmdUI)
  {  pCmdUI->Enable(TRUE);  }  //使窗格文本能被更新
```

说明：

CCmdUI 类对象方法（成员函数也叫对象方法），如表 2.8 所示。

表 2.8　CCmdUI 类对象方法

对象方法	作　用
ContinueRouting()	告诉命令发送机构沿着 handlers 键继续发送当前的消息
Enable()	为该命令激活或关闭用户界面项
SetCheck()	为该命令设置用户界面项的核对状态
SetRadio()	类似 SetCheck 成员函数，但通过单选组操作
SetText()	为这个命令设置用于用户界面的文本

（8）用 ClassWizard 在视图类 CMyView 中加入 WM_MOUSEMOVE（鼠标移动）消息处理函数，并添加下列代码。该函数先获得状态栏对象的指针，然后调用 SetPaneText 函数更新第 3 和第 4 窗格中的文本。

```
void CMyView::OnMouseMove(UINT nFlags,CPoint point)
  { CString str;
    CMainFrame *pFrame=(CMainFrame*)AfxGetApp()->m_pMainWnd; //获得主窗口指针

    CStatusBar *pStatus=&pFrame->m_wndStatusBar; //获得主窗口中的状态栏指针
    if(pStatus)
      {  str.Format("X=%d",point.x);//格式化文本
         pStatus->SetPaneText(2,str);//更新第三个窗格的文本
         str.Format("Y=%d",point.y);
         pStatus->SetPaneText(3,str);//更新第四个窗格的文本
       }
  }
```

（9）将 MainFrm.h 文件中的受保护变量 m_wndStatusBar 变成公共变量。

（10）在视图 View.cpp 文件的开始处增加语句：#include “MainFrm.h”。

（11）编译运行，如图 2.30 所示，鼠标位置就在状态栏里显示出来了。

（12）改变状态栏的风格：将例 2.8 中的鼠标移动函数 OnMouseMove（UINT nFlags，CPoint point）代码修改为：

```
void CQqView::OnMouseMove(UINT nFlags, CPoint point)
{  CString str;
   CMainFrame *pFrame=(CMainFrame*)AfxGetApp()->m_pMainWnd;//获得主窗口指针
   CStatusBar *pStatus=&pFrame->m_wndStatusBar; //获得主窗口中的状态栏指针
     if(pStatus)
      {  pStatus->SetPaneStyle(2,SBPS_POPOUT);//第 2 个栏
         pStatus->SetPaneStyle(3,SBPS_POPOUT);//第 3 个栏
         str.Format("X=%d",point.x);          //格式化文本
         ……
      }
            CView::OnMouseMove(nFlags, point);
}
```

（13）编译运行，如图 2.31 所示，第 2 和第 3 窗格凸起来了。

图 2.30 鼠标的位置显示在状态栏上

图 2.31 设置状态栏的风格

2.4 交互对象的动态更新

用户交互对象是指可由用户操作而产生命令消息的对象，如：菜单项、工具条中的按钮和加速键，每个用户交互对象都有一个唯一的 ID 号，在发送消息时，该 ID 号被包含在 WM_COMMAND 消息中。特别是：菜单项可以有灰色显示，选中和未选中三种状态，而工具栏按钮则可以有禁止和选中状态等。

为了能使用户交互对象动态更新，MFC 是通过 ClassWizard 直接映射交互对象的更新命令消息来实现的。它自动将用户交互对象的 ID 标识符与 ON_UPDATE_COMMAND_UI 宏相连接并产生处理更新消息的相应函数。为说明交互对象的动态更新，看例 2.10。

［例 2.10］交互对象的动态更新

（1）创建一个单文档的应用程序，名为：动态更新。

（2）在 ResourceView 页中打开 Toolbar，选中 IDR_MAINFRAME，按住 Ctrl 键，移动鼠标将 IDR_MAINFRAME 拖到 Toolbar 资源名上，这样就复制了工具栏 IDR_MAINFRAME，复制后的资源标识，系统自动设为 IDR_MAINFRAME1。

（3）右击 IDR_MAINFRAME1，从弹出的快捷菜单中选择 Properties 命令，在弹出的属性对话框中将 ID 改为 IDR_NEWBAR。

（4）删除几个 IDR_NEWBAR 上的工具按钮，以与 IDR_MAINFRAME 有区别。

（5）在主框架 CMainFrame 类的头文件 MainFrm.h 的 protected 下，声明变量：

```
CToolBar m_wndNewBar;
BOOL m_bNewBar;
```

（6）在 CMainFrame::OnCreate 中添加下列代码：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
 {    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
     if(!m_wndNewBar.CreateEx(this,TBSTYLE_FLAT,WS_CHILD|WS_VISIBLE|
     CBRS_TOP|CBRS_GRIPPER|CBRS_TOOLTIPS|CBRS_FLYBY|
       CBRS_SIZE_DYNAMIC,CRect(0,0,0,0),AFX_IDW_TOOLBAR+10)||
         !m_wndNewBar.LoadToolBar(IDR_NEWBAR))
    {  TRACE0("Failed to ctreate newbar\n");
       return -1;             //fail to create
    }
      … …
      m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
      m_wndNewBar.EnableDocking(CBRS_ALIGN_ANY);
      EnableDocking(CBRS_ALIGN_ANY);
      DockControlBar(&m_wndToolBar);
      DockControlBar(&m_wndNewBar);
      return 0;
  }
```

（7）打开菜单资源 Menu，双击 IDR_MAINFRAME，在“查看”菜单下添加一个菜单项，名为：新工具栏（&N），ID 标识符设定为 ID_VIEW_NEWBAR。

（8）用 MFC ClassWizard 在 CMainFrame 类中添加菜单 ID_VIEW_NEWBAR 的 COMMAND 和 UPDATE_COMMAND_UI 两个消息映射，并在映射函数中添加下列代码：

```
void CMainFrame::OnViewNewbar()
 {    m_bNewBar=!m_bNewBar;
      ShowControlBar(&m_wndNewBar,m_bNewBar,FALSE);//显示或隐藏工具栏
 }
void CMainFrame::OnUpdateViewNewbar(CCmdUI* pCmdUI)
 {  m_bNewBar=m_wndNewBar.IsWindowVisible();
    //pCmdUI->SetRadio(m_bNewBar);   //选中用(•)
    pCmdUI->SetCheck(m_bNewBar);     //选中(√)
 }
```

说明：

① 代码中，OnUpdateViewNewbar 是 ID_VIEW_NEWBAR 的更新命令消息的消息映射函数。该函数只有 1 个参数，它是指向 CCmdUI 对象的指针。CCmdUI 类仅用于 ON_UPDATE_COMMAND_UI 消息映射函

数，它的成员函数将对菜单项、工具栏按钮等用户交互对象起作用，如表 2.9 所示。

② 调用 Create 时，还可以指定工具栏的风格，默认风格是：

WS_CHILD|WS_VISIBLE|CBRS_TOP 创建子窗口|窗口最初是可见的（见第 4 章表 4.1）。

③ 以 WS_为开头表示窗口风格，以 SW_开头表示窗口状态的改变（见第 4 章表 4.3）。

（9）编译运行：打开“查看”菜单，可以看到“新工具栏”菜单前面有一个“√”，如图 2.32 所示。选择“新工具栏”命令，则新创建的工具栏不见了，而“新工具栏”菜单前面的标记√没有了。

若将代码中 SetCheck 改为 SetRadio，则“√”变成了“•”，这就是交互对象的更新效果。

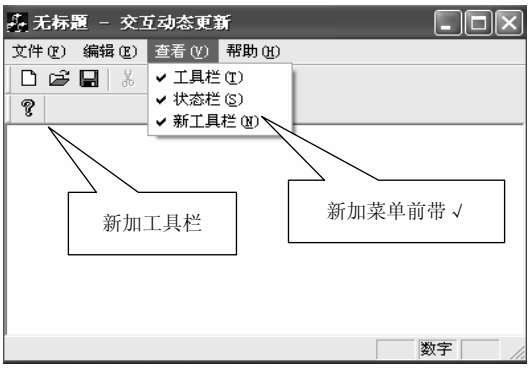

图 2.32　查看下拉菜单

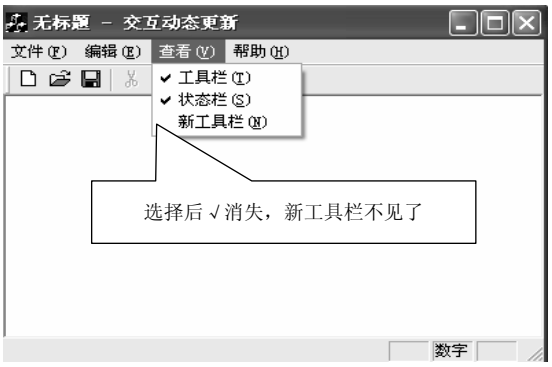

图 2.33　选择“新工具栏”命令

表 2.9　CComdUI 类的成员函数对用户交互对象的作用

用户交互对象	Enable	SetCheck	SetRadio	SetText
菜单项	允许或禁用	选中（√）或未选中	选定用（•）	设置菜单文本
工具栏按钮	允许或禁用	选定、未选定或不确定	同 SetCheck	无效
状态栏窗格（PANE）	使文本可见或不可见	边框外凸或正常	同 SetCheck	设置窗格文本
CDialogBar 中的按钮	允许或禁用	选中或未选中	同 SetCheck	设置按钮文本
CDialogBar 中的控件	允许或禁用	无效	无效	设置窗口文本

2.5　章后实训

实训 1　通用菜单

（1）创建一个单文档应用程序，名为：通用菜单。

（2）建立菜单：在顶层菜单“帮助”的后面连续建立名为：图形颜色（&C），图形边宽

（&W），可选图形（&G），属性选为：√Pop Up 的 3 个菜单，如图 2.34 所示。

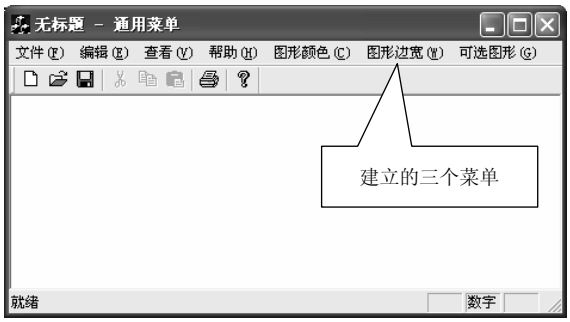

图 2.34 建立的三个顶层菜单

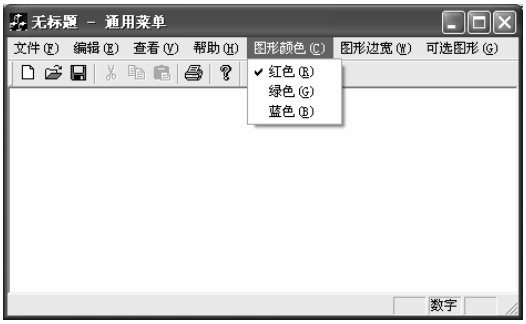

图 2.35 “图形颜色”下建的三个子菜单

（3）按表 2.10 在“图形颜色”菜单下面连续建立三个子菜单，如图 2.35 所示。

表 2.10 “图形颜色”的子菜单

ID	标题（Caption）	提示（Prompt）
ID_COLOR_RED	红色（&R）	你选择了红色
ID_COLOR_GREEN	绿色（&G）	你选择了绿色
ID_COLOR_BLUE	蓝色（&B）	你选择了蓝色

按表 2.11 在“图形边宽”菜单下面连续建立三个子菜单。

表 2.11 “图形边宽”的子菜单

ID	标题（Caption）	提示（Prompt）
ID_LINE_SINGE	单线宽（&S）	你选择了单线宽
ID_LINE_THREE	三线宽（&T）	你选择了三线宽
ID_LINE_FIVE	五线宽（&F）	你选择了五线宽

按表 2.12 在“可选图形”菜单下面连续建立三个子菜单。

表 2.12 “可选图形”的子菜单

ID	标题（Caption）	提示（Prompt）
ID_GRAPH_LINE	直线（&L）	你选择了直线
ID_GRAPH_CIRCLE	椭圆（&C）	你选择了椭圆
ID_GRAPH_RECTANGLE	矩形（&R）	你选择了矩形

（4）在主框架 CmainFrame.cpp 加入各菜单 COMMAND（命令消息）和 UPDATE_COMMAND_UI（更新消息）处理函数：

1）加入“图形颜色”菜单下面三个子菜单：

ID_COLOR_RED，ID_COLOR_GREEN，ID_COLOR_BLUE 的 COMMAND 命令消息和 UPDATE_COMMAND_UI 更新命令消息。

2）加入“图形边宽”菜单下面三个子菜单：

ID_LINE_SINGLE，ID_LINE_THREE，ID_LINE_FIVE 的 COMMAND 命令消息和 UPDATE_COMMAND_UI 更新命令消息。

3）加入“可选图形”菜单下面三个子菜单：

ID_GRAPH_LINE，ID_GRAPH_CIRCLE，ID_GRAPH_RECTANGLE 的 COMMAND 命令消息和 UPDATE_COMMAND_UI 更新命令消息。

（5）在 CmainFrame 类添加成员变量：在项目工作区的 ClassView 选项卡中右击 CmainFram，在弹出的快捷菜单中选择 Add Member Variable 命令，弹出 Add Member Variable 对话框，在 Variable Type 处写：COLORREF，在 Variable Name 处写：m_Color。接着按此方法添加：int m_Thickness 、UINT m_Graph 、UINT m_TagColor 三个变量。

（6）在 CmainFrame.cpp 的构造函数中，为以上添加的 4 个成员变量赋初值：

```
CMainFrame::CMainFrame()
{ m_Color=RGB(255,0,0);//初始颜色为红
  m_TagColor=ID_COLOR_RED;//初始颜色 ID 标识符为红
  m_Graph=ID_GRAPH_LINE;//初始图形的 ID 标识符为直线
  int m_Thickness=1;    //画图形时线的宽度
}
```

（7）创建浮动菜单：选择 Insert->Resource 命令，弹出 Insert Resource 对话框，选中“Menu”，单击 New 按钮，如图 2.36 所示，双击右边的空白菜单，弹出 Menu Item Properties 对话框，在 General 选项卡中选中“√Pop_Up”，“Caption”（标题）处写：浮动菜单。

（8）将该菜单项的标识符“ID_MENU1”改为：IDR_PopUpMenu。

（9）双击“Menu”下的标识符：IDR_MAINFRAME，右击“图形颜色”菜单，选择 Cope 命令。再双击“Menu”下的 IDR_PopUpMenu 菜单，右边出现“浮动菜单”，在“浮动菜单”下面空白子菜单上右击，在弹出的快捷菜单中选择 Paste 命令。用相同的方法将“图形边宽”和“可选图形”复制到“浮动菜单”的下一个子菜单下，如图 2.37 所示。

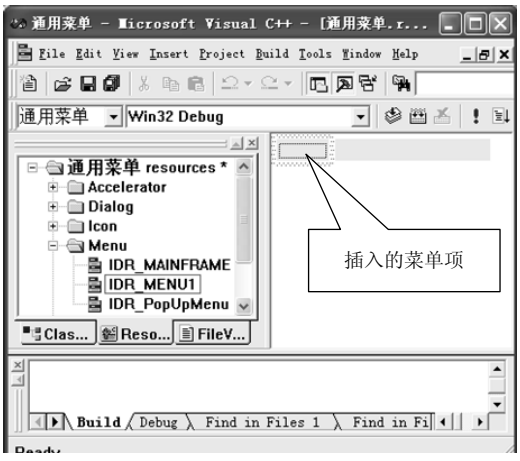

图 2.36 插入的菜单项

图 2.37 将加入的菜单复制到浮动菜单下

（10）为第（4）步在 CmainFrame 类添加的各个菜单命令 COMMAND 消息和 UPDATE_COMMAND_UI 更新命令消息处理函数加代码：

```
//选择"图形颜色"→"红色"命令后的消息处理函数
void CMainFrame::OnColorRed()
{      m_Color=RGB(255,0,0);//表示选择了红色
       m_TagColor=ID_COLOR_RED;//设置颜色标志为 ID_COLOR_RED
}
//选择"图形颜色"→"红色"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateColorRed(CCmdUI* pCmdUI)
{//判颜色标识是否等于该命令对应的标识符 ID_COLOR_RED,若相等则在该菜单命令左边显示"√"
   pCmdUI→SetCheck(m_TagColor==ID_COLOR_RED?1:0);
}
//选择"图形颜色"→"绿色"命令后的消息处理函数
void CMainFrame::OnColorGreen()
{       m_Color=RGB(0,255,0); //表示选择了绿色
        m_TagColor=ID_COLOR_GREEN; //设置颜色标志为 ID_COLOR_GREEN
}
//选择"图形颜色"→"绿色"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateColorGreen(CCmdUI* pCmdUI)
{//判颜色标识是否等于该命令对应标识符 ID_COLOR_GREEN,若相等则在该菜单命令左边显示"√"
   pCmdUI->SetCheck(m_TagColor==ID_COLOR_GREEN?1:0);
}
//选择"图形颜色"→"蓝色"命令后的消息处理函数
void CMainFrame::OnColorBlue()
{      m_Color=RGB(0,0,255); //表示选择了蓝色
       m_TagColor=ID_COLOR_BLUE; //设置颜色标志为 ID_COLOR_BLUE
}
//选择"图形颜色"→"蓝色"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateColorBlue(CCmdUI* pCmdUI)
{//判颜色标识是否等于该命令对应标识符 ID_COLOR_BLUE,若相等则在该菜单命令左边显示"√"
   pCmdUI->SetCheck(m_TagColor==ID_COLOR_BLUE?1:0);
}
//选择"图形边宽"→"单线宽"命令后的消息处理函数
void CMainFrame::OnLineSingle()
{      m_Thickness=1;//设置为单线宽
}
//选择"图形边宽"→"单线宽"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateLineSingle(CCmdUI* pCmdUI)
{ //判断图形边宽变量的值是否等于 1，若相等则在该菜单命令左边显示"√"
```

```
    pCmdUI->SetCheck(m_Thickness==1?1:0);
}
//选择"图形边宽"→"三线宽"命令后的消息处理函数
void CMainFrame::OnLineThree()
{      m_Thickness=3; //设置为三线宽     }
//选择"图形边宽"→"三线宽"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateLineThree(CCmdUI* pCmdUI)
{ //判断图形边宽变量的值是否等于 3，若相等则在该菜单命令左边显示"√"
    pCmdUI->SetCheck(m_Thickness==3?1:0);
}
//选择"图形边宽"→"五线宽"命令后的消息处理函数
void CMainFrame::OnLineFive()
{      m_Thickness=5; //设置为五线宽      }
//选择"图形边宽"→"五线宽"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateLineFive(CCmdUI* pCmdUI)
{//判断图形边宽变量的值是否等于 5，若相等则在该菜单命令左边显示"√"
    pCmdUI->SetCheck(m_Thickness==5?1:0);
}
//选择"可选图形"→"直线"命令后的消息处理函数
void CMainFrame::OnGraphLine()
{   m_Graph=ID_GRAPH_LINE;//设置图形标识为 ID_GRAPH_LINE
    CPen * oPen, nPen;
  //创建画笔，颜色由变量 m_Color 决定，宽度由变量 m_Thickness 决定
    nPen.CreatePen(PS_SOLID,m_Thickness,m_Color);
    CClientDC cdc(this);
    CRect rect;
    GetClientRect(&rect);
 cdc.FillRect(&rect,CBrush::FromHandle((HBRUSH)GetStockObject(LTGRAY_BRUSH)));
 oPen=cdc.SelectObject(&nPen);
 cdc.MoveTo(100,310);
 cdc.LineTo(500,310);
 cdc.SelectObject(oPen);
}
//选择"可选图形"→"直线"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateGraphLine(CCmdUI* pCmdUI)
{//判图形标识是否等于该命令对应标识符 ID_GRAPH_LINE,若相等则在该菜单命令左边显示"√"
    pCmdUI->SetCheck(m_Graph==ID_GRAPH_LINE?1:0);
}
```

```
//选择"可选图形"→"椭圆"命令后的消息处理函数
void CMainFrame::OnGraphCircle()
{    m_Graph=ID_GRAPH_CIRCLE;//设置图形标识为 ID_GRAPH_CIRCLE
     CPen * oPen, nPen;
//创建画笔，颜色由变量 m_Color 决定，宽度由变量 m_Thickness 决定
     nPen.CreatePen(PS_SOLID,m_Thickness,m_Color);
     CClientDC cdc(this);
     CRect rect;
     GetClientRect(&rect);
 cdc.FillRect(&rect,CBrush::FromHandle((HBRUSH)GetStockObject(LTGRAY_BRUSH)));
 oPen=cdc.SelectObject(&nPen);
  cdc.Ellipse(100,100,200,200);
  cdc.SelectObject(oPen);
}
//选择"图形颜色"→"椭圆"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateGraphCircle(CCmdUI* pCmdUI)
{//判图形标识是否等于该命令对应标识符 ID_GRAPH_CIRCLE,若相等则在该菜单命令左边显示"√"
   pCmdUI->SetCheck(m_Graph==ID_GRAPH_CIRCLE?1:0);
}
//选择"可选图形"→"矩形"命令后的消息处理函数
void CMainFrame::OnGraphRectangle()
{  m_Graph=ID_GRAPH_RECTANGLE; //设置图形标识为 ID_GRAPH_RECTANGLE
     CPen * oPen, nPen;
//创建画笔，颜色由变量 m_Color 决定，宽度由变量 m_Thickness 决定
     nPen.CreatePen(PS_SOLID,m_Thickness,m_Color);
     CClientDC cdc(this);
     CRect rect;
     GetClientRect(&rect);
 cdc.FillRect(&rect,CBrush::FromHandle((HBRUSH)GetStockObject(LTGRAY_BRUSH)));
     oPen=cdc.SelectObject(&nPen);
     cdc.Rectangle(350,100,450,200);
     cdc.SelectObject(oPen);
}
//选择"可选图形"→"矩形"命令后，可引发该菜单命令的更新消息处理函数
void CMainFrame::OnUpdateGraphRectangle(CCmdUI* pCmdUI)
{//判图形标识是否等该命令对应标识符ID_GRAPH_RECTANGLE,若等则在该菜单命令左边显示"√"
   pCmdUI->SetCheck(m_Graph==ID_GRAPH_RECTANGLE?1:0);
}
```

（11）在视图类（CMyView）中加入 WM_RBUTTONDOWN（鼠标右键按下）的消息处理函数，并添加代码：

```
//在视图窗口中右击时的消息处理函数的定义
void CMyView::OnRButtonDown(UINT nFlags, CPoint point)
{   CMenu popMenu; //声明菜单类对象
//LoadMenu 成员函数从一个执行文件中获取一个菜单资源，并把它分配给 CMenu 对象，该操作若成
//功则继续，否则抛出异常
      if(!popMenu.LoadMenu(IDR_PopUpMenu))
      ::AfxThrowResourceException();
//GetSubMenu 成员函数返回一个指向 CMenu 对象的指针，若给定位置不存在浮动菜单，返回 NULL
      CMenu * pPopUpMenu=popMenu.GetSubMenu(0);
  //若不存在浮动菜单，将抛出异常
      if(pPopUpMenu==NULL)
      ::AfxThrowResourceException();
    this->ClientToScreen(&point);
//TrackPopupMenu 成员函数在确定位置显示悬浮式弹出菜单，并跟踪此菜单的项目选择过程
    pPopUpMenu->TrackPopupMenu(TPM_CENTERALIGN|TPM_RIGHTBUTTON,
    point.x,point.y,::AfxGetMainWnd());
    CView::OnRButtonDown(nFlags, point);
}
```

（12）编译运行，如图 2.38 和图 2.39 所示。在显示图形时，视图窗口背景色被设置为灰色。在视图窗口中右击，则出现浮动菜单，其实现的结果与顶层菜单是一样的。

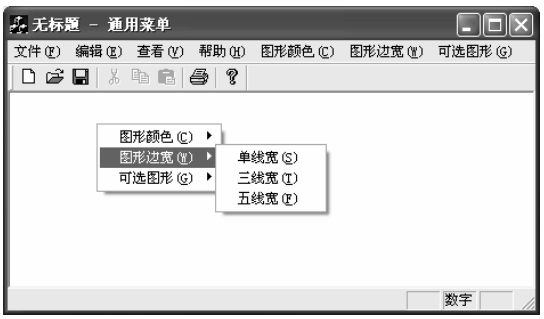

图 2.38 在窗口中右击出现浮动菜单

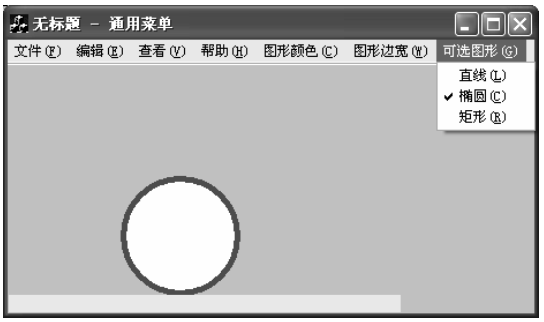

图 2.39 选择“可选图形”→“椭圆”

实训 2 多信息状态栏

（1）打开上面的“实训 1—通用菜单”程序，在该程序基础上继续操作。

（2）选择 View→Resource Symbols（资源符号）命令，弹出 Resource Symbols 对话框，单击 New 按钮，弹出 New Symbol 对话框，在 Name 文本框中输入：ID_STATUSPART1→系统自动在 Value（值）文本框中加入一个值，如图 2.40 所示。按同样方法，依次加入资源符号名为：ID_STATUSPART2、ID_STATUSPART3 和 IDS_STATUS_TIMER。

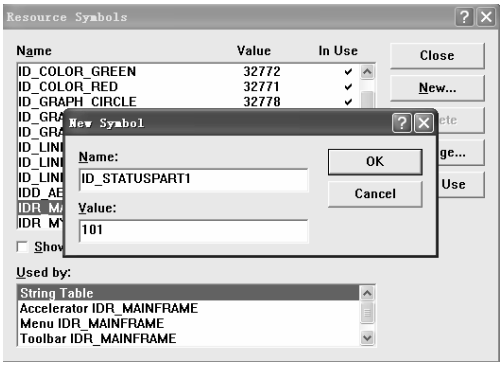

图 2.40　添加资源符号

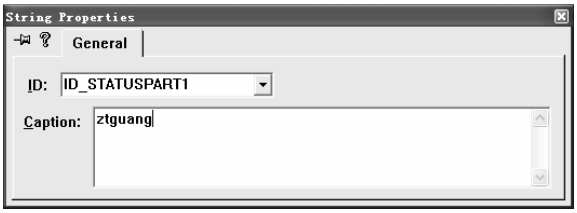

图 2.41　添加串属性表

（3）在项目工作区的 Resource View 选项卡中展开 String Table，双击标识符 String Table，打开字符串编辑窗口，双击最后一个空白行，弹出 String Properties 对话框，在 ID 文本框中输入：ID_STATUSPART1，在 Caption 文本框中输入：ztguang，如图 2.41 所示。用同样的办法，依次添加表 2.13 所列出的串属性。

表 2.13　串属性表

Control IDs	标　　题
ID_STATUSPART2	ztguang
ID_STATUSPART3	ztguang
IDS_STATUS_TIMER	00:00

（4）在主框架头文件 CMainFrm.h 的 public 下定义变量和声明函数。

```
public:
 CString m_StatusPart1;   //存放颜色红、绿、蓝
 CString info1;          //在状态栏显示红、绿、蓝
 CString m_StatusPart2;  //存放单线宽、三线宽、五线宽
 CString info2;          //在状态栏显示单线宽、三线宽、五线宽
 CString m_StatusPart3;  //存放图形直线、椭圆、矩形
 CString info3;         //在状态栏显示直线、椭圆、矩形
 UINT m_nIDTimer; //每秒钟发送一个消息到应用程序的消息队列，该变量存放时间消息队列
CStatusBar m_StatusBar; //声明类 CStatusBar 对象
 //添加 TimerProc()函数的声明
 static void CALLBACK TimerProc(HWND hwnd, UINT uMsg, UINT uIDEvent,DWORD
dwTime);
//添加 ShowStatusBarInfo()函数的声明
void ShowStatusBarInfo();   //显示状态条信息
//在消息映射处加入新的消息映射函数声明
protected:
 //{{AFX_MSG(CMainFrame)
```

```
afx_msg void OnUpdateTime(CCmdUI *pCmdUI);//声明状态条时间显示函数
//}}AFX_MSG
```

（5）在主框架执行文件 CMainFrm.cpp 中添加代码：

1）消息映射开始处添加代码：

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
 //{{AFX_MSG_MAP(CMainFrame)
ON_UPDATE_COMMAND_UI(IDS_STATUS_TIMER,OnUpdateTime)//状态条时间显示
 //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

2）修改状态栏内容的代码如下：

```
static UINT indicators[] =
{ ID_SEPARATOR,//在状态条左边生成空的状态栏目,用于菜单命令和工具条命令提示信息的显示
  ID_STATUSPART1,//显示选择的图形颜色
  ID_STATUSPART2,//显示选择的图形边宽
  ID_STATUSPART3,//显示选择的图形
  ID_INDICATOR_CAPS,//显示键盘键 Caps Lock 键的状态
  ID_INDICATOR_NUM,//显示键盘键 NumLock 键的状态
  ID_INDICATOR_SCRL,//显示键盘键 ScrollLock 键的状态
  IDS_STATUS_TIMER,//显示系统时间
};
```

3）在构造函数中为变量赋初值的代码如下：

```
CMainFrame::CMainFrame()
{     m_Color=RGB(0,0,0);//m_Color=RGB(255,0,0);//颜色
      m_Thickness=0;//m_Thickness=1;//线宽
      m_Graph=0;//m_Graph=ID_GRAPH_LINE;//图形 ID
      m_TagColor=0;//m_TagColor=ID_COLOR_RED;//颜色 ID
      m_StatusPart1="ztguang";//颜色栏初值
      m_StatusPart2="ztguang";//线宽栏初值
      m_StatusPart3="ztguang";//图形栏初值
}
```

4）在析构函数中加清除（应用程序消息队列）函数：

```
CMainFrame::~CMainFrame()
{  ::KillTimer(NULL,m_nIDTimer);// 清空时间消息队列      }
```

（6）修改主框架类（CMainFrame）OnCreate（LPCREATESTRUCT lpCreateStruct）消息处理函数：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{     if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
//Create()建立一状态条子窗口，并将它和 CstatausBar 对象联系在一起，同时按默认
```

```
//值设定状态栏的字模和高度
 if (!m_StatusBar.Create(this) ||
//SetIndicators()按照数组 indicators 中的对应元素的值设定标识符 ID 值，加载由每
//个 ID 所指定的字符串资源，并把字符串设置为标识符的文字
      !m_StatusBar.SetIndicators(indicators,
                        sizeof(indicators)/sizeof(UINT)))
 {    TRACE0("Failed to create status bar\n");
      return -1;      // fail to create
 }
//发送消息，以便更新状态条时间，该语句每秒钟发送一个消息到应用程序的消息队列，当应用程序
//清空其消息队列时，时间显示将被更新
 m_nIDTimer = ::SetTimer(NULL, 0, 1000, TimerProc);
//可将原代码全注释掉
   return();
}
```

（7）为主框架执行文件 CMainFrm.cpp 中的各个菜单命令函数添加代码：

```
void CMainFrame::OnColorRed()
{//当选择"图形颜色"→"红色"命令后，就执行该消息执行函数，首先设置变量的值，
//然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
m_StatusPart1="红色";
ShowStatusBarInfo();
m_Color=RGB(255,0,0);//颜色
m_TagColor=ID_COLOR_RED;//颜色的 ID
}
void CMainFrame::OnColorGreen()
{ //当选择"图形颜色"→"绿色"命令后，就执行该消息执行函数，首先设置变量的值，
//然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
  m_StatusPart1="绿色";
 ShowStatusBarInfo();
 m_Color=RGB(0,255,0);
 m_TagColor=ID_COLOR_GREEN;
}
void CMainFrame::OnColorBlue()
{ //当选择"图形颜色"→"蓝色"命令后，就执行该消息执行函数，首先设置变量的值，
  //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
 m_StatusPart1="蓝色";
 ShowStatusBarInfo();
 m_Color=RGB(0,0,255);
 m_TagColor=ID_COLOR_BLUE;
```

```
}
void CMainFrame::OnGraphLine()
{ //当选择"可选图形"→"直线"命令后，就会执行该消息执行函数，首先设置变量的值，
  //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
 m_StatusPart3="直线";
 ShowStatusBarInfo();
 m_Graph=ID_GRAPH_LINE;
 CPen * oPen, nPen;
 nPen.CreatePen(PS_SOLID,m_Thickness,m_Color);
 CClientDC cdc(this);
 CRect rect;
 GetClientRect(&rect);
cdc.FillRect(&rect,CBrush::FromHandle((HBRUSH)GetStockObject(LTGRAY_BRUSH)));
 oPen=cdc.SelectObject(&nPen);
 cdc.MoveTo(100,310);
 cdc.LineTo(500,310);
 cdc.SelectObject(oPen);
}
void CMainFrame::OnGraphCircle()
{  //当选择"可选图形"→"椭圆"命令后，就会执行该消息执行函数，首先设置变量的值，
   //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
   m_StatusPart3="椭圆";
   ShowStatusBarInfo();
 m_Graph=ID_GRAPH_CIRCLE;
 CPen * oPen, nPen;
 nPen.CreatePen(PS_SOLID,m_Thickness,m_Color);
 CClientDC cdc(this);
 CRect rect;
 GetClientRect(&rect);
cdc.FillRect(&rect,CBrush::FromHandle((HBRUSH)GetStockObject(LTGRAY_BRUSH)));
 oPen=cdc.SelectObject(&nPen);
 cdc.Ellipse(100,100,200,200);
 cdc.SelectObject(oPen);
}
void CMainFrame::OnGraphRectangle()
{ //当选择"可选图形"→"矩形"命令后，就会执行该消息执行函数，首先设置变量的值，
  //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
  m_StatusPart3="矩形";
 ShowStatusBarInfo();
```

```
 m_Graph=ID_GRAPH_RECTANGLE;
 CPen * oPen, nPen;
 nPen.CreatePen(PS_SOLID,m_Thickness,m_Color);
 CClientDC cdc(this);
 CRect rect;
 GetClientRect(&rect);
cdc.FillRect(&rect,CBrush::FromHandle((HBRUSH)GetStockObject(LTGRAY_BRUSH)));
 oPen=cdc.SelectObject(&nPen);
 cdc.Rectangle(350,100,450,200);
 cdc.SelectObject(oPen);
}
void CMainFrame::OnLineSingle()
{//当选择"可选图形"→"单线宽"命令后，就会执行该消息执行函数，首先设置变量的值，
 //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
 m_StatusPart2="单线宽";
 ShowStatusBarInfo();
 m_Thickness=1;
}
void CMainFrame::OnLineThree()
{ //当选择"可选图形"→"三线宽"命令后，就会执行该消息执行函数，首先设置变量的值，
  //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
  m_StatusPart2="三线宽";
  ShowStatusBarInfo();
  m_Thickness=3;
}
void CMainFrame::OnLineFive()
{ //当选择"可选图形"→"五线宽"命令后，就会执行该消息执行函数，首先设置变量的值，
  //然后调用 ShowStatusBarInfo()函数，在状态栏相应的位置显示刚设置的值
   m_StatusPart2="五线宽";
   ShowStatusBarInfo();
   m_Thickness=5;
}
```

（8）在主框架执行文件 CMainFrame.cpp 中，加入自定义函数及代码（全用手写）：

```
// ShowStatusBarInfo()显示状态栏内容函数：将 CString 类对象 m_StatusPart 信息格式化
//后放入 CString 对象 info 中,调用 SetPaneText()函数,用 info 的内容设置状态条中由第一
//个参数指定的位置
void CMainFrame::ShowStatusBarInfo()
{ info1.Format("%s",m_StatusPart1);// m_StatusPart1 颜色
  m_StatusBar.SetPaneText(m_StatusBar.CommandToIndex(ID_STATUSPART1),info1);
```

```
  info2.Format("%s",m_StatusPart2);// m_StatusPart2 线宽
  m_StatusBar.SetPaneText(m_StatusBar.CommandToIndex(ID_STATUSPART2),info2);
  info3.Format("%s",m_StatusPart3);// m_StatusPart3 图形
  m_StatusBar.SetPaneText(m_StatusBar.CommandToIndex(ID_STATUSPART3),info3);
}
void CMainFrame::OnUpdateTime(CCmdUI *pCmdUI) //状态条时间显示函数
{   CTime timer = CTime::GetCurrentTime();
    char szTimer[6];
    int mHour=timer.GetHour();
    int mMinute=timer.GetMinute();
     //如要按 12 小时制显示，请去掉下面两条语句的注释
    //if (nHour > 12)
    //nHour = nHour - 12;
    wsprintf(szTimer, "%i:%02i", mHour, mMinute);
    //把时间写到 Pane
    m_StatusBar.SetPaneText(m_StatusBar.CommandToIndex
                            (IDS_STATUS_TIMER), LPCSTR(szTimer));
                                    pCmdUI->Enable();
}
  void CALLBACK CMainFrame::TimerProc(HWND hwnd, UINT uMsg,
                                      UINT uIDEvent,DWORD dwTime)
    {   CMainFrame *pMainWnd = (CMainFrame *)AfxGetApp()->m_pMainWnd;
        ASSERT(uIDEvent == pMainWnd->m_nIDTimer);
        CCmdUI cui;
        cui.m_nID = IDS_STATUS_TIMER;
        cui.m_nIndex = 4;
        cui.m_pMenu = NULL;
        cui.m_pOther = &pMainWnd->m_StatusBar;
        pMainWnd->OnUpdateTime(&cui);//调用 OnUpdateTime()函数,更新时间显示
     }
```

（9）编译运行，如图 2.42～图 2.44 所示。

图 2.42 状态栏的初始状态

图 2.43 选择菜单后的状态栏

实训 3　自定义工具条

图 2.44　选择“可选图形”的“矩形”

自定义工具条，工具条由 3 种颜色按钮组成，还能使其放在窗口中的任何位置，并能随时关闭。

（1）创建一个单文档的应用程序，名为：浮动工具条。

（2）添加工具条资源：选择 Insert->Resource 菜单，选中 Toolbar 单击 New 按钮，将新工具条 ID 改为：IDR_COLORTOOLBAR。

（3）单击出现的第 1 个工具条按钮，用画刷将其涂成红色，接着将出现的第 2 个按钮涂成绿色，第 3 个按钮涂成蓝色。

（4）将 3 个工具条按钮的 ID 分别改为：ID_BUTTONRED，ID_BUTTONGREEN，ID_BUTTONBLUE 并将它们的 COMMAND 消息分别映射到主框架类 CMainFrame 中。

（5）创建显示/隐藏工具条、显示/隐藏状态栏和显示新建工具条的菜单：打开 Menu 下的 IDR_MAINFRAME，在“帮助”菜单的前面建一个菜单：观察（&V），选中 Pop up，双击下面的子菜单，ID 处写：ID_VIEW_TOOLBAR，Caption 处写：工具条（&T），Prompt 处写：显示或隐藏工具栏\n 显隐工具栏。

设置下一个子菜单，ID 处写：ID_VIEW_STATUS_BAR，Caption 处写：状态栏（&S），Prompt 处写：显示或隐藏状态栏\n 显隐状态栏。

再设置下一个子菜单，ID 处写：ID_VIEWCOLORTOOLBAR，Caption 处写：新建颜色工具条（&N）。

（6）在主框架类 CMainFrame.h 头文件的 public 下添加成员变量：

```
CToolBar *m_pColorToolbar;
```

（7）在主框架类 CMainFrame.cpp 文件的构造函数中置初值：

```
CMainFrame::CMainFrame()
:m_pColorToolbar(0)
{     }
```

（8）用 ClassWizard 在主框架类中创建自定义菜单命令 ID_VIEWCOLORTOOLBAR 的消息映射，并添加下列代码：

```
void CMainFrame::OnViewcolortoolbar()
{   if(0==m_pColorToolbar)
    {  m_pColorToolbar=new CToolBar;
       if(0==m_pColorToolbar->Create(this))
        {  MessageBox("创建失败");
          return ;
        }
       m_pColorToolbar->LoadToolBar(IDR_COLORTOOLBAR);//获得工具条资源
       //设置工具条出现时的位置，这里是可将工具条放在任一侧
```

```
        m_pColorToolbar->EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(m_pColorToolbar);
    }
    else
     if(m_pColorToolbar->IsWindowVisible()==TRUE)//该函数是获得给定窗口的可视状态
     ShowControlBar(m_pColorToolbar,FALSE,FALSE);
     else
     ShowControlBar(m_pColorToolbar,TRUE,TRUE);
}
```

（9）编译运行，打开“观察”菜单，如图 2.45 所示，选择“新建颜色工具条”命令，出现如图 2.46 所示，可见“新建颜色工具条”菜单前面的“√”不见了（如图 2.45 所示）。将工具条拖到窗口中，如图 2.47 所示，选择“工具栏”命令，“工具栏”前的“√”不见了，工具条也不见了，如图 2.48 所示，选择“状态栏”命令和单击拖到窗口的工具条右上角上的“x”情况如何，读者可试试看。

注意：新建的工具条在左上角，鼠标要放在工具条按钮的边缘上，按下左键才能拖动，否则若将鼠标放在工具条按钮上是拖不动的。

图 2.45 打开“观察”菜单

图 2.46 选择“新建颜色工具条”命令

图 2.47 将工具条拖到窗口中

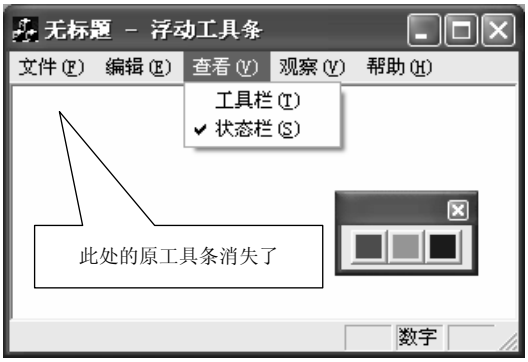

图 2.48 选择“工具栏”命令

说明：

如果在一个框架窗口中存在多个标准或浮动工具条时，需要利用函数

void DockControlBar（CControlBar ＊pBar，UINT nDockBarID=0，LPCRECT lpRect= NULL）来确定要控制停靠位置的工具条，它也是 CFrameWnd 类的成员函数。

参数：

pBar 用来指向被控制停靠位置的工具条对象指针。

nDockBarID 用来确定工具条停靠在框架窗口的哪条边上，其控制风格的具体取值为：

AFX_IDW_DOCKBAR_TOP	停靠在框架窗口的顶部；
AFX_IDW_DOCKBAR_BOTTOM	停靠在框架窗口的底部；
AFX_IDW_DOCKBAR_LEFT	停靠在框架窗口的左边；
AFX_IDW_DOCKBAR_RIGHT	停靠在框架窗口的右边。

当参数 nDockBarID 的取值为 0 时，则工具条可以停靠在框架窗口中的任何一个可停靠的边上，其默认的初始位置为窗口顶部。

应用程序中的工具栏的停靠或浮动性有：

CBRS_ALIGN_TOP	允许停靠客户区顶部；
CBRS_ALIGN_BOTTON	允许停靠客户区底部；
CBRS_ALIGN_LEFT	允许停靠客户区左侧；
CBRS_ALIGN_RIGHT	允许停靠客户区右侧；
CBRS_ALIGN_ANY	允许停靠客户区任意一侧；
CBRS_FLOAT_MULTI	允许多个控制条在一个单一的小框架窗口中浮动。