CHAPTER 3

第3章 常用控件

控件是在系统内部定义的用于和用户交互的基本单元。在所有的控件中，根据它们的使用及 Visual C++ 6.0 对其支持的情况，可以把控件分为 Windows 普通控件（如编辑框、列表框、组合框等）、MFC 扩展控件和 ActiveX 控件。ActiveX 控件可以理解成是一个 OLE（Object Linking and Embedding，对象连接与嵌入）组件，它既可用于 Windows 应用程序中，也可用于 Web 页面中。

本章重点介绍在 MFC 应用程序中经常使用的控件，主要有静态控件、按钮、编辑框、列表框、组合框、滚动条、进展条、旋转按钮控件、滑动条、日期时间控件、列表控件和树控件。

3.1 创建和使用控件

在 MFC 应用程序中使用控件不仅简化编程，还能完成常用的各种功能。为了更好地发挥控件作用，还必须理解和掌握控件的属性、消息以及创建和使用的方法。

3.1.1 控件的创建方法

控件的创建方式有以下两种：

一种是在对话框模板中用编辑器指定控件，也就是说，将对话框看作控件的父窗口。这样做的好处是显而易见的，因为当应用程序启动该对话框时，Windows 系统就会为对话框创建控件，而当对话框消失时，控件也随之清除。

另一种是编程方式，即调用 MFC 相应控件类的成员函数 Create 来创建，并在 Create 函数指定控件的父窗口指针。例如，下面的示例过程是使用编程方式来创建一个按钮。

① 在“E:\VISUAL C++程序”文件夹中，创建本章应用程序工作文件夹“第 3 章”。

② 启动 Visual C++ 6.0，选择“文件”→“新建”菜单命令，在弹出的“新建”对话框的“工程”选项卡中，选择 MFC AppWizard(exe)的项目类型。单击“位置”编辑框右侧的…按钮，在弹出的对话框中，将该应用程序的文件夹定位在“E:\VISUAL C++程序\第 3 章”，并在“工程名称”编辑框中输入应用程序名 Ex_Create。

③ 单击“确定”按钮进入下一步，从弹出的“步骤 1”对话框中，选择“基本对话框”

应用程序类型。单击“完成”按钮，出现一个信息对话框，显示出用户在前面几个步骤中做出的选择，单击“确定”按钮，系统开始创建，并回到了 Visual C++ 6.0 的主界面。这样，一个默认的基于对话框的应用程序项目 Ex_Create 就创建好了。

④ 将项目工作区切换到 ClassView 选项卡，展开 Ex_Create 所有的类结点，右击 CEx_CreateDlg 类名，弹出如图 3.1 所示的快捷菜单。从快捷菜单中选择 Add Member Variable（添加成员变量）命令，出现如图 3.2 所示的对话框，在 Variable Type（变量类型）编辑框中输入 CButton（MFC 按钮类），在 Variable Name（变量名称）编辑框中输入要定义的 CButton 类对象名 m_btnWnd。单击“确定”按钮。

注意：对象名通常以“m_”作为开头，表示“成员”(member)的意思。

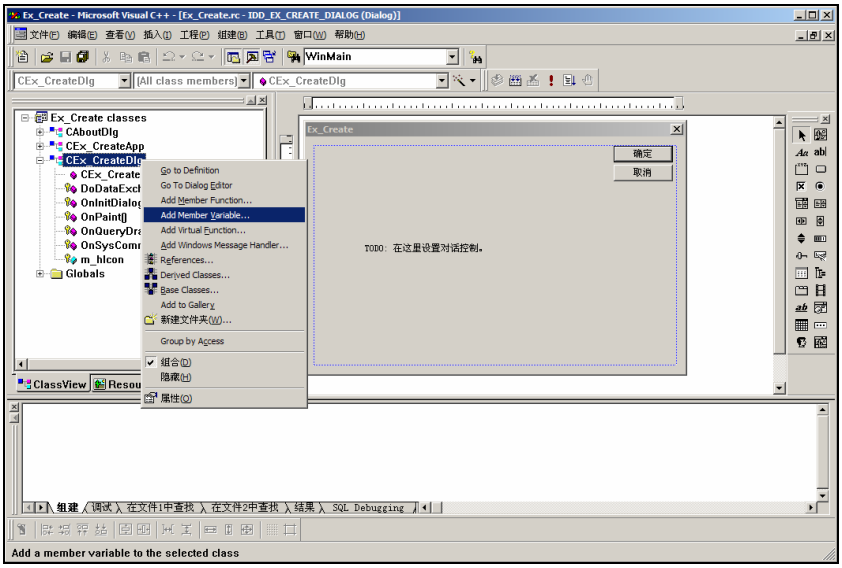

图 3.1　弹出的快捷菜单

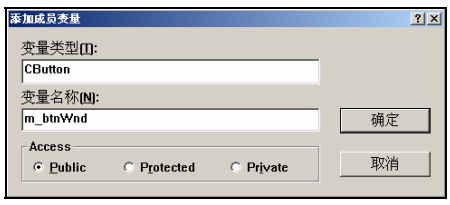

图 3.2　添加控件类对象成员

需要说明的是，在 MFC 中，每一种类型的控件都用相应的类来封装。如编辑框控件的类是 CEdit，按钮控件的类是 CButton，通过这些类创建的对象来访问其成员，从而实现控件的相关操作。

⑤ 在项目工作区窗口的 ClassView 页面中，将 CEx_CreateDlg 结点展开，双击 OnInitDialog 函数名，在该函数中添加下列代码（在 return true 语句前添加）：

```
BOOL CEx_CreateDlg::OnInitDialog()
{
```

```
    CDialog::OnInitDialog();
    …
    m_btnWnd.Create("你好", WS_CHILD | WS_VISIBLE
                    | BS_PUSHBUTTON | WS_TABSTOP,
                    CRect(20, 20, 120, 40), this, 201); // 创建
    CFont *font = this->GetFont();              // 获取对话框的字体
    m_btnWnd.SetFont(font);                     // 设置控件字体
    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

分析和说明：

- 前面曾说过，由于 OnInitDialog 函数在对话框初始化时被调用，因此将对话框中的一些初始化代码都添加在此函数中。
- 由于 Windows 操作系统使用的是图形界面，因此在 MFC 中，对于每种界面元素的几何大小和位置常使用 CPoint 类（点）、CSize 类（大小）和 CRect 类（矩形）来描述（以后还会讨论）。
- 代码中，CButton 类成员函数 Create 用来创建按钮控件，该函数第一个参数用来指定按钮的标题，第二个参数用来指定按钮控件的样式，其中 BS_PUSHBUTTON（以 BS_开头的）是按钮类封装的预定义样式，表示创建的是按键按钮。WS_CHILD（子窗口）、WS_VISIBLE（可见）、WS_TABSTOP（可用 Tab 键选择）等都是 CWnd 类封装的预定义窗口样式，它们都可以直接引用，当多个样式指定时，需要使用按位或运算符“|”来连接。第三个参数用来指定它在父窗口中的位置和大小，第四个参数用来指定父窗口指针，最后一个参数是指定该控件的标识值。
- 由于按钮是作为对话框的一个子窗口来创建的，因此 WS_CHILD 样式是必不可少的，且还要使用 WS_VISIBLE 使控件在创建后显示出来。

⑥ 编译并运行，结果如图 3.3 所示。

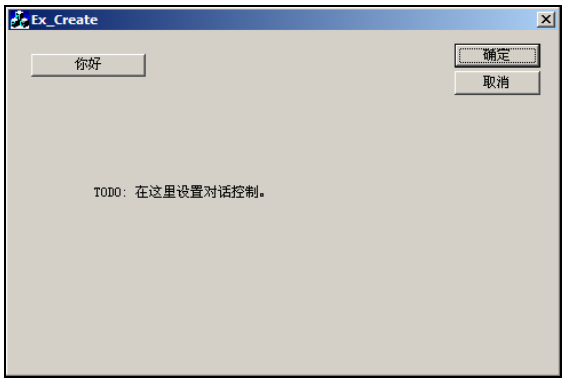

图 3.3　Ex_Create 运行结果

以上可以看出，控件编程创建方法是使用各自封装的类的 Create 成员来创建，它最大的优点就是能动态创建，但它涉及的编程内容比较复杂，且不能发挥对话框编辑器可视化的优点，故在一般情况下都采用第一种方法，即在对话框模板中用编辑器指定控件。

3.1.2 控件的消息及消息映射

应用程序创建一般控件或公共控件之后，当控件的状态发生改变（例如用户利用控件进行输入）时，控件就会向其父窗口发送消息，这个消息称为“通知消息”。对于每个消息，系统都会用一个 MSG 结构来记录，MSG 具有下列原型：

```
typedef struct tagMSG {      // msg
    HWND     hwnd;           // 接收到消息的窗口句柄
    UINT     message;        // 消息
    WPARAM   wParam;         // 消息的附加信息，它的含义取决于message
    LPARAM   lParam;         // 消息的附加信息，它的含义取决于message
    DWORD    time;           // 消息传送时的时间
    POINT    pt;             // 消息传送时，光标所在的屏幕坐标
} MSG;
```

对于一般控件来说，其通知消息通常是一条 WM_COMMAND 消息，这条消息的 wParam 参数的低位字中含有控件标识符，wParam 参数的高位字则为通知代码，lParam 参数则是指向控件的句柄。

而对于有些控件，其通知消息通常是一条 WM_NOTIFY 消息，这条消息的 wParam 参数是发送通知消息的控件的标识符，而 lParam 参数则是指向一个结构指针。

1．映射控件消息

不管是什么控件消息，一般都可以用 MFC ClassWizard 对它们加以映射。

① 将项目工作区窗口切换到 ResourseView，双击 Dialog 资源下的标识 IDD_EX_CREATE_DIALOG，打开 Ex_Create 项目的对话框资源模板。

② 选中“TODO: 在这里设置对话控制。”控件，按 Delete 键删除。从控件工具箱中拖放添加一个按钮控件，如图 3.4 所示，保留其默认属性。

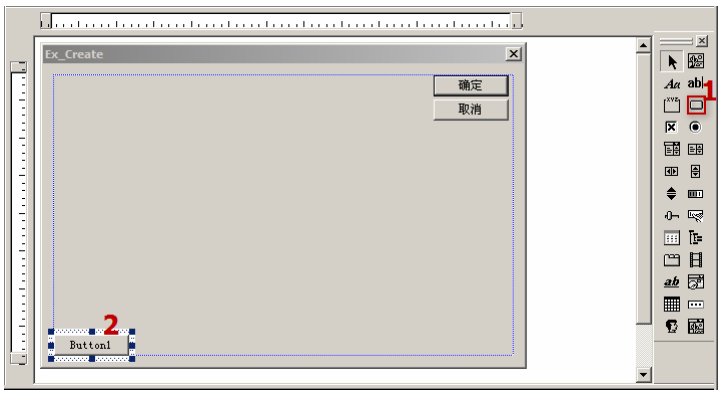

图 3.4 添加一个按钮

③ 按 Ctrl+W 键，打开 MFC ClassWizard 对话框，查看 Class name 列表中是否选择了 CEx_CreateDlg，在 IDs 列表中选择 IDC_BUTTON1，这是添加按钮后，系统自动为此按钮

设置的默认标识符，然后在 Messages 列表框中选择 BN_CLICKED 消息。

④ 单击 Add Function 按钮或双击 BN_CLICKED 消息，出现 Add Member Function 对话框，在这里可以输入成员函数的名称，系统默认的函数名为 OnButton1，如图 3.5 所示。单击 OK 按钮，BN_CLICKED 消息映射函数添加到 Member functions 列表中。

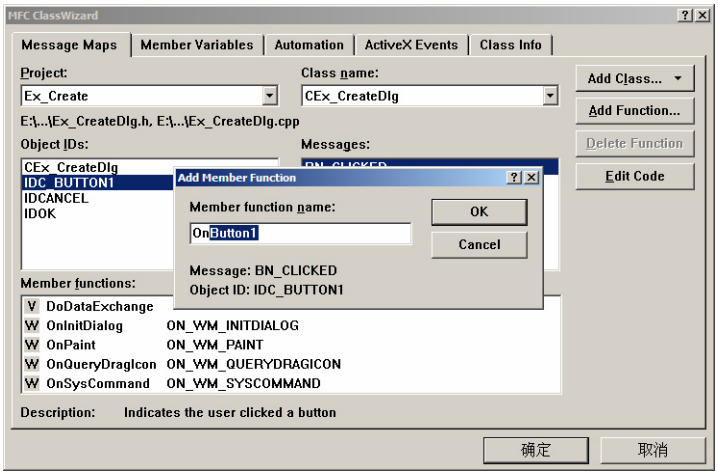

图 3.5　添加按钮消息映射函数

说明：

- 不同资源对象（控件、菜单命令等）所产生的消息是不相同的。例如，按钮控件 IDC_BUTTON1 的消息有两个：BN_CLICKED 和 BN_DOUBLECLICKED，分别表示当用户单击或双击该按钮时产生的消息。
- 一般不需要对对话框中的 OK（确定）与 Cancel（取消）按钮进行消息映射，因为系统已自动设置了这两个按钮的动作，当用户单击这两个按钮时都将自动关闭对话框，且 OK（确定）按钮动作还使得对话框数据有效。

⑤ 双击消息函数 OnButton1 或单击 Edit Code 按钮，退出 MFC ClassWizard 对话框，并转向文档窗口，定位到 CEx_CreateDlg::OnButton1 函数实现的源代码处，添加下列代码：

```
void CEx_CreateDlg::OnButton1()
{
    MessageBox(_T("你按下了\"Button1\"按钮！"));
}
```

⑥ 编译并运行，当单击 Buttonl 按钮时，就会执行 OnButton1 函数，弹出一个消息对话框，显示“你按下了‘Buttonl’按钮!”内容。

这就是按钮 BN_CLICKED 消息的映射过程，其他控件的消息也可以类似操作。

2．映射控件通用消息

上述的过程是映射一个控件的某一个消息，事实上也可以通过 WM_COMMAND 消息的映射来处理一个或多个控件的通用消息，如下面的过程：

① 按 Ctrl+W 键，打开 MFC ClassWizard 对话框，查看 Class name 下拉列表框中是否选择了 CEx_CreateDlg，在 IDs 列表中选择 CEx_CreateDlg，在 Messages 列表框中找到并

双击 OnCommand，这样 OnCommand 消息函数就添加好了，如图 3.6 所示。

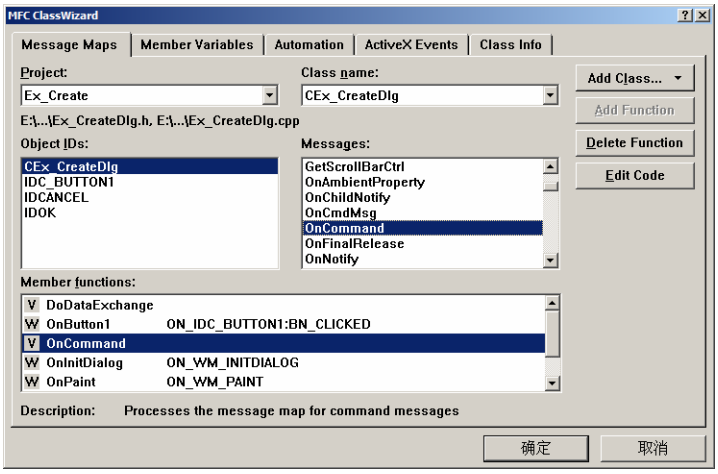

图 3.6　添加 OnCommand 消息函数

需要说明的是，由于 OnCommand 函数是一个用来处理 WM_COMMAND 消息的虚函数，因此这里添加的 OnCommand 函数事实上是一个在类中实际调用的函数，可称为“实例函数”。这样的映射操作，可以称为“对虚函数 OnCommand 的重载”。

② 双击消息函数 OnCommand 或单击 Edit Code 按钮，退出 MFC ClassWizard 对话框，并转向文档窗口，定位到 CEx_CreateDlg:: OnCommand 函数实现的源代码处，添加下列代码：

```
BOOL CEx_CreateDlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    WORD  nCode = HIWORD(wParam);           // 控件的通知消息
    WORD  nID   = LOWORD(wParam);           // 控件的ID号
    if ((nID == 201)&&(nCode == BN_CLICKED))
          MessageBox(_T("你按下了\"你好\"按钮！"));
    if ((nID == IDC_BUTTON1)&&(nCode == BN_CLICKED))
          MessageBox(_T("这是在OnCommand处理的结果！"));
    return CDialog::OnCommand(wParam, lParam);
}
```

③ 编译并运行。当单击图 3.3 所示的对话框中的“你好”按钮时，弹出消息对话框，显示“你按下了‘你好’按钮!”内容。

说明：

- 在 MFC 中，资源都是用其 ID 来标识的，而各资源的 ID 号本身就是数值，因此上述代码中，201 和 IDC_BUTTON1 都是程序中用来标识按钮控件的 ID，201 是前面创建控件时指定的 ID 值。
- 在上述编写的代码中，Buttonl 按钮的 BN_CLICKED 消息用不同的方式处理了两次，即同时存在两种函数 OnButton1 和 OnCommand，因此若单击 Buttonl 按钮，系统会先执行哪一个函数呢？测试的结果表明，系统首先执行 OnCommand 函数，

然后执行 OnButton1 代码。之所以还能执行 OnButton1 函数代码，是因为 OnCommand 函数的最后一句代码“return CDialog::OnCommand(wParam, lParam);”，它将控件的消息交由对话框其他函数处理。

- 由于用 Create 创建的控件无法用 MFC ClassWizard 直接映射其消息，因此上述方法弥补了 ClassWizard 的不足，使用时要特别注意。

3.1.3　控件类和控件对象

一旦创建控件后，有时就需要进行深入编程。使用控件之前需获得该控件的类对象指针或映射一个对象，然后通过该指针或对象来引用其成员函数进行操作。表 3.1 列出了 MFC 封装的常用控件类。

表 3.1　常用控件类

控件名称	MFC 类	功能描述
静态控件	CStatic	用来显示一些几乎固定不变的文字或图形
按钮	CButton	用来产生某些命令或改变某些选项，包括单选按钮、复选框和组框
编辑框	CEdit	用于完成文本和数字的输入与编辑
列表框	CListBox	显示一个列表，让用户从中选取一个或多个项
组合框	CComboBox	是一个列表框和编辑框组合的控件
滚动条	CScrollBar	通过滚动块在滚动条上的移动和滚动按钮来改变某些量
进展条	CProgressCtrl	用来表示一个操作的进度
滑动条	CSliderCtrl	通过滑动块的移动来改变某些量，并带有刻度指示
旋转按钮控件	CSpinButtonCtrl	带有一对反向箭头的按钮，单击这对按钮可增加或减少某个值
日期时间控件	CDateTimeCtrl	用于选择指定的日期和时间
图像列表	CImageList	一个具有相同大小的图标或位图的集合
标签控件	CTabCtrl	类似于一个笔记本的分隔器或一个文件柜上的标签，使用它可以将一个窗口或对话框的相同区域定义为多个页面

在 MFC 中，获取一个控件的类对象指针是通过 CWnd 类的成员函数 GetDlgItem 来实现的，它具有下列原型：

```
CWnd* GetDlgItem(int nID ) const;
void GetDlgItem( int nID,  HWND* phWnd) const;
```

在 C++中允许同一个类中出现同名的成员函数，只是这些同名函数的形参类型或形参个数各不相同（为叙述方便，这些同名函数从上到下依次称为第一版本、第二版本、……）。其中，nID 用来指定控件或子窗口的 ID 值，第一版本是直接通过函数来返回 CWnd 类指针，而第二版本是通过函数形参 phWnd 来返回其句柄指针。

需要说明的是，由于 CWnd 类是通用的窗口基类，因此想要调用实际的控件类及其基类成员，还必须将其进行类型的强制转换，例如：

```
CButton* pBtn = (CButton*)GetDlgItem(IDC_BUTTON1);
```

由于 GetDlgItem 获取的是类对象指针，因而它可以用到程序的任何地方，且可多次使

用，并可对同一个控件定义不同的对象指针，均可对指向的控件操作有效。事实上，在父窗口类，还可为控件或子窗口定义一个成员变量，通过它也能引用其成员函数进行操作。

控件的成员变量又称控件变量，在 MFC 中，控件变量分为两种类型，一是用于操作的控件对象；另一是用于存取的数据变量。它们都是与控件或子窗口进行绑定，但 MFC 只允许每种类型仅绑定一次。下面就来看一个示例。

① 创建一个默认的对话框应用程序 Ex_Member。

② 在打开的对话框资源模板中，删除“TODO: 在这里设置对话控制。”静态文本控件，将“确定”和“取消”按钮向对话框左边移动一段位置，然后将鼠标移至对话框资源模板右下角的实心蓝色方块处，拖动鼠标，将对话框资源模板的大小缩小一些。

③ 在对话框资源模板的左边添加一个编辑框控件和一个按钮控件，保留其默认属性，并将其布局得整齐一些，如图 3.7 所示。

④ 按 Ctrl+W 键，打开 MFC ClassWizard 对话框，并切换到 Member Variables 选项卡，查看 Class name 列表中是否选择了 CEx_MemberDlg，此时可以在 Control IDs 列表中看到刚才添加的控钮和编辑框的标识符 IDC_BUTTON1 和 IDC_EDIT1。

⑤ 在 Control IDs 列表中，选定按钮控件标识符 IDC_BUTTON1，双击或单击 Add Variable 按钮，弹出 Add Member Variable 对话框，如图 3.8 所示。

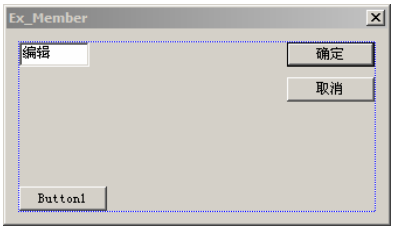

图 3.7　添加编辑框和按钮

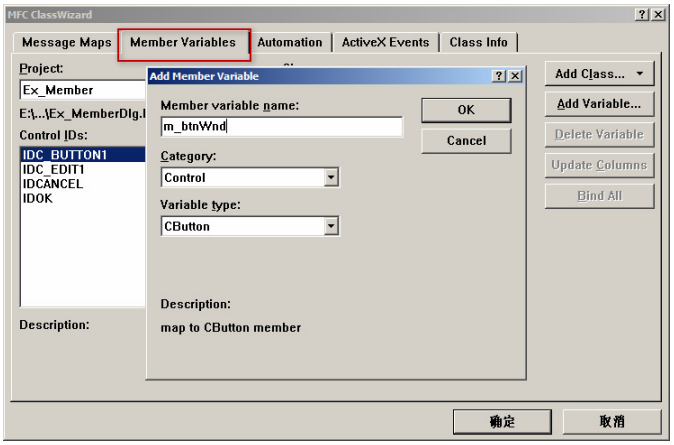

图 3.8　添加控件对象

⑥ 在 Member variable name 框中填写与控件相关联的成员变量 m_btnWnd，且使 Category（类别）项为 Control，单击 OK 按钮，又回到 MFC ClassWizard 对话框的 Member Variables 中，在 Control IDs 列表中出现刚才添加的 CButton 控件对象 m_btnWnd。这样，按钮控件 IDC_BUTTON1 的编程操作就可用与之绑定的对象 m_btnWnd 来操作。

⑦ 将 MFC ClassWizard 对话框切换到 Message Maps 选项卡，为 CEx_MemberDlg 添加 IDC_BUTTON1 的 BN_CLICKED 消息映射函数 OnButton1，并添加下列代码：

```
void CEx_MemberDlg::OnButton1()
{
    CString strEdit;                          // 定义一个字符串
    CEdit   *pEdit = (CEdit*)GetDlgItem( IDC_EDIT1);
```

```
        pEdit->GetWindowText( strEdit );    // 获取编辑框中的内容

        strEdit.TrimLeft();
        strEdit.TrimRight();

        if (strEdit.IsEmpty())
               m_btnWnd.SetWindowText(_T("Button1"));
        else
               m_btnWnd.SetWindowText(strEdit);
    }
```

由于 strEdit 是 CString 类对象，因而可以调用 CString 类的公有成员函数。其中，TrimLeft 和 TrimRight 函数不带参数时分别用来去除字符串左边或右边一些空格符、换行符、Tab 字符等字符，IsEmpty 用来判断字符串是否为空。

这样，当编辑框内容有除（空格）之外的实际字符的字符串时，用 SetWindowText 将其内容设定为按钮控件的标题。否则，按钮控件的标题为 Button1。

⑧ 编译并运行。当在编辑框中输入 Hello 后，单击 Button1 按钮，按钮的名称就变成了编辑框控件中的内容 Hello。

3.1.4　DDX 和 DDV

对于控件的数据变量，MFC 还提供了独特的 DDX 和 DDV 技术。DDX 将数据成员变量同对话类模板内的控件相联接，这样就使得数据在控件之间很容易传输。而 DDV 用于数据的校验，例如它能自动校验数据成员变量数值的范围，并发出相应的警告。

一旦某控件与一个数据变量相绑定后，就可以使用 CWnd::UpdateData 函数实现控件数据的输入和读取。UpdateData 函数只有一个参数，它为 TRUE 或 FALSE。当在程序中调用 UpdateData(FALSE)时，数据由控件绑定的成员变量向控件传输，当调用 UpdateData(TRUE)或不带参数的 UpdateData()时，数据从控件向相绑定的成员变量复制。

需要说明的是，数据变量的类型由被绑定的控件类型而定，例如对于编辑框来说，数值类型可以有 CString、int、UINT、long、DWORD、float、double、BYTE、short、BOOL 等。不过，任何时候传递的数据类型只能是一种。也就是说，一旦指定了数据类型，则在控件与变量传递交换的数据就不能是其他类型，否则无效。

下面来看一个示例，它是在 Ex_Member 项目基础上进行的：

① 按 Ctrl+W 键，打开 MFC ClassWizard 对话框，并切换到 Member Variables 选项卡，查看 Class name 列表中是否选择了 CEx_MemberDlg。

② 在 Control IDs 列表中，选定按钮控件标识符 IDC_EDIT1，双击或单击 Add Variable 按钮，弹出 Add Member Variable 对话框，将 Category（类别）选为默认的 Value（值），将 Variable Type 类型选为默认的 CString，在 Member variable name 框中填好与控件相关联的成员变量 m_strEdit，如图 3.9 所示。

③ 单击 OK 按钮，回到 MFC ClassWizard 对话框的 Member Variables 选项卡中，在

Control IDs 列表中出现刚才添加的编辑框控件变量 m_strEdit。选择后，将在 MFC ClassWizard 对话框下方出现 Maximum Characters 编辑框，从中可设定该变量允许的最大字符个数，这就是控件变量的 DDV 设置。填入 10 后，如图 3.10 所示，单击“确定”按钮，退出 MFC ClassWizard 对话框。

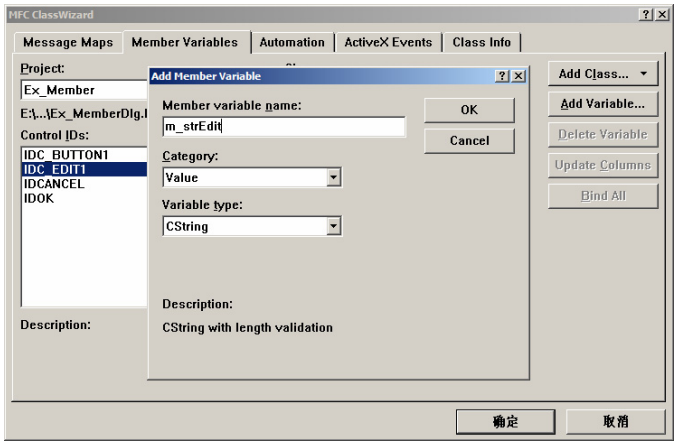

图 3.9　添加控件变量

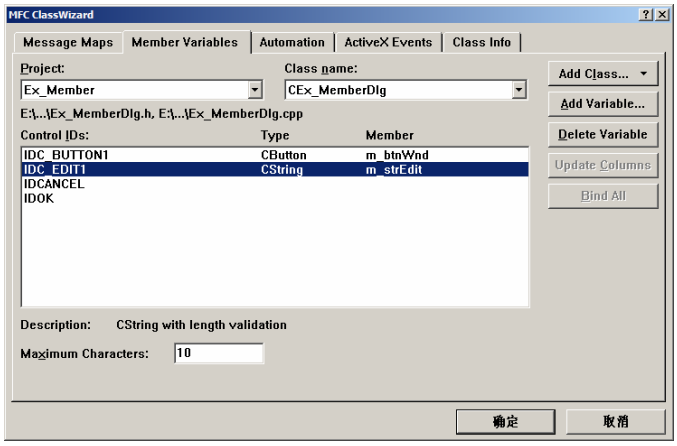

图 3.10　设置 m_strEdit 允许的最大字符个数

④ 将项目工作区切换到 ClassView 选项卡，展开 CEx_MemberDlg 类结点，双击 OnButton1 成员函数结点，定位到 CEx_MemberDlg::OnButton1 函数实现代码处，将代码修改如下：

```
void CEx_MemberDlg::OnButton1()
{
    UpdateData();                          // 将控件的内容存放到变量中
    // 没有参数，表示使用的是默认参数值TRUE
    m_strEdit.TrimLeft();
    m_strEdit.TrimRight();
    if (m_strEdit.IsEmpty())
```

```
            m_btnWnd.SetWindowText(_T("Button1"));
        else
            m_btnWnd.SetWindowText(m_strEdit);
    }
```

⑤ 编译并运行。当在编辑框中输入 Hello，单击 Buttonl 按钮后，OnButton1 函数中的 UpdateData 将编辑框中的内容保存到 m_strEdit 变量中，从而执行下一条语句后按钮的名称就变成了编辑框控件中的内容 Hello。若输入 Hello DDX/DDV，则当输入第 10 个字符后，再也输入不进去了，这就是 DDV 的作用。

3.2　静态控件和按钮

静态控件和按钮是 Windows 最基本的控件之一。

3.2.1　静态控件

一个静态控件是用来显示一个字符串、框、矩形、图标、位图或增强的图元文件。它可以被用来作为标签、框或用来分隔其他的控件。一个静态控件一般不接收用户输入，也不产生通知消息。

在对话框编辑器的控件工具栏中，属于静态控件的有静态文本（Aa）、组框（xyz）和静态图片（）三种。

3.2.2　按钮

在 Windows 中所用的按钮是用来实现一种开与关的输入，常见的按钮有三种类型：按键按钮、单选按钮和复选框按钮，如图 3.11 所示。

图 3.11　按钮的不同类型

1．不同按钮的作用

按键按钮通常可以立即产生某个动作，执行某个命令，因此也常被称为命令按钮。按键按钮有两种风格：标准按键按钮和默认按键按钮（或称默认按钮）。从外观上来说，默认按键按钮是在标准按键按钮的周围加上一个黑色边框（见图 3.11），这个黑色边框表示该按钮已接收到键盘的输入焦点，这样一来，用户只需按 Enter 键就能按下该按钮。一般来说，只把最常用的按键按钮设定为默认按键按钮，具体设定的方法是在按键按钮属性对话框的 Style 页面中选中“默认按钮”（Default Button）项。

单选按钮的外形是在文本前有一个圆圈，当它被选中时，单选按钮中就标上一个黑点，

它可分为一般和自动两种类型。在自动类型中，用户若选中同组按钮中的某个单选按钮，则其余的单选按钮的选中状态就会清除，保证了多个选项始终只有一个被选中。

复选框的外形是在文本前有一个空心方框，当它被选中时，复选框中就加上一个“✔”标记，通常复选框只有选中和未选中两种状态，若复选框前面有一个灰色是“✔”，则这样的复选框是三态复选框，如图 3.11 中的 Check2，它表示复选框的选择状态是“不确定”。

2．按钮的消息

按钮消息常见的只有两个：BN_CLICKED（单击按钮）和 BN_DOUBLE_CLICKED（双击按钮）。

3．按钮操作

最常用的按钮操作是设置或获取一个按钮或多个按钮的选中状态。封装按钮的 CButton 类中的成员函数 SetCheck 和 GetCheck 就是分别用来设置或获取指定按钮的选中状态，其原型如下：

```
void SetCheck( int nCheck );
int GetCheck( ) const;
```

其中，nCheck 和 GetCheck 函数返回的值可以是 0（不选中）、1（选中）和 2（不确定，仅用于三态按钮）。

若对于同组多个单选按钮的选中状态的设置或获取，则需要使用通用窗口类 CWnd 的成员函数 CheckRadioButton 和 GetCheckedRadioButton，它们的原型如下：

```
void CheckRadioButton( int nIDFirstButton, int nIDLastButton, int nIDCheck
Button );
int GetCheckedRadioButton( int nIDFirstButton, int nIDLastButton );
```

其中，nIDFirstButton 和 nIDLastButton 分别指定同组单选按钮的第一个和最后一个按钮 ID 值，nIDCheckButton 用来指定要设置选中状态的按钮 ID 值，函数 GetCheckedRadioButton 返回被选中的按钮 ID 值。

3.2.3 示例：制作问卷调查

问卷调查是日常生活中经常遇到的调查方式。例如，图 3.12 就是一个问卷调查对话框，它针对“上网”话题提出了三个问题，每个问题都有 4 个选项，除最后一个问题外，其余都是单项选择。本例用到了组框、静态文本、单选按钮、复选框等控件。实现时，需要通过 CheckRadioButton 函数来设置同组单选按钮的最初选中状态，通过 SetCheck 来设置指定复选框的选中状态，然后通过 GetCheckedRadioButton 和 GetCheck 来判断被选中的单选按钮和复选框，并通过 GetDlgItemText 或 GetWindowText 获取选中控件的窗口文本。

1．创建并设计对话框

① 创建一个默认的基于对话框应用程序 Ex_Research。系统会自动打开对话框编辑器并显示对话框资源模板。单击对话框编辑器工具栏上的“切换网格”按钮，显示对话框网格。打开属性对话框，将对话框标题改为“上网问卷调查”。

② 调整对话框的大小，删除对话框中间的“TODO: 在这里设置对话控制。”静态文本控件，将“确定”和“取消”按钮移至对话框的下方，并向对话框中添加组框（Group）控件，然后调整其大小和位置。

③ 右击添加的组框控件，从弹出的快捷菜单中选择“属性”命令，出现该控件的属性对话框，在“常规”选项卡中可以看到它的 ID 为默认的 IDC_STATIC。将其“标题”（Caption）属性内容由“Static”改成“你的年龄”。在组框控件的“样式”（Styles）属性中，“水平排列”属性用来指定文本在顶部的左边（Left）、居中（Center）还是右边（Right）。默认（Default）选项表示左对齐。

④ 在组框内添加 4 个单选按钮，默认的 ID 依次为 IDC_RADIO1、IDC_RADIO2、IDC_RADIO3 和 IDC_RADIO4。在其属性对话框中将 ID 属性内容分别改成 IDC_AGE_L18、IDC_AGE_18T27、IDC_AGE_28T38 和 IDC_AGE_M38，然后将其“标题”属性内容分别改成“< 18”、“18 - 27”、“28 - 38”和“> 38”，最后调整位置，结果如图 3.13 所示。

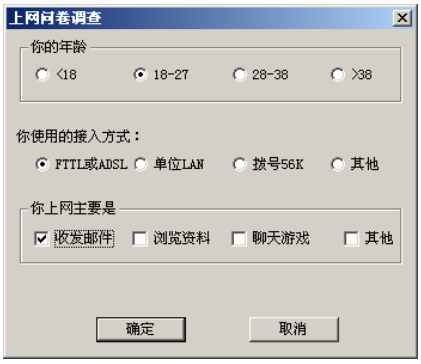

图 3.12　上网问卷调查对话框

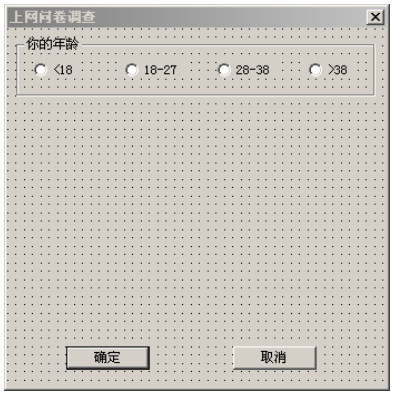

图 3.13　添加的组框和单选按钮

⑤ 接下来添加一个静态文本，标题设为“你使用的接入方式：”，然后在其下再添加 4 个单选按钮，标题分别是“FTTL 或 ADSL”、“单位 LAN”、“拨号 56K”和“其他”，并将相应的 ID 属性依次改成：IDC_CM_FTTL、IDC_CM_LAN、IDC_CM_56K 和 IDC_CM_OTHER。用对话框编辑器工具栏的按钮命令调整控件左右之间的间距，结果如图 3.14 所示。

⑥ 在对话框的下方，再添加一个组框控件，其标题为“你上网主要是”。然后添加 4 个复选框，其标题分别为“收发邮件”、“浏览资料”、“聊天游戏”和“其他”，ID 分别为 IDC_DO_POP、IDC_DO_READ、IDC_DO_GAME 和 IDC_DO_OTHER。结果如图 3.15 所示。

⑦ 单击工具栏上的测试对话框按钮。对话框测试后，可以发现：顺序添加的这 8 个单选按钮全部变成一组，也就是说，在这组中只有一个单选按钮被选中，这不符合本意。解决这个问题的最好的办法是将每一组中的第一个单选按钮的“组”属性选中。因此，分别将以上两个问题中的第一个单选按钮的“组”属性均选中。图 3.16 所示是对第二个问题设置的结果。

⑧ 单击对话框编辑器工具栏上的切换辅助线按钮，然后将对话框中的控件调整到辅

助线以内，并适当对其他控件进行调整。这样，整个问卷调查的对话框就设计好了，单击工具栏上的按钮测试对话框。

图 3.14　再添加单选框图

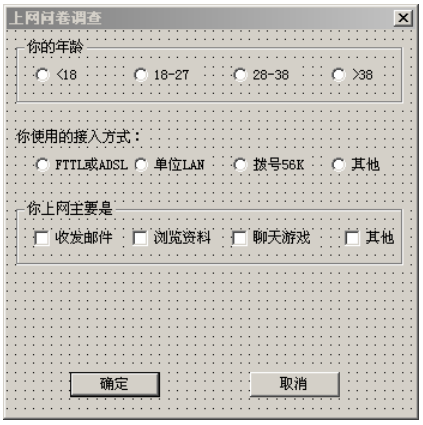

图 3.15　三个问题全部添加后的对话框

图 3.16　选中“组”属性

2．完善代码

① 将项目工作区切换到 ClassView 选项卡，展开 CEx_ResearchDlg 类的所有成员，双击 OnInitDialog 函数结点，将会在文档窗口中自动定位到该函数的实现代码处，在此函数添加下列初始化代码：

```
BOOL CEx_ResearchDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    …
    CheckRadioButton(IDC_AGE_L18, IDC_AGE_M38, IDC_AGE_18T27);
    CheckRadioButton(IDC_CM_FTTL, IDC_CM_OTHER, IDC_CM_FTTL);
    CButton* pBtn = (CButton*)GetDlgItem(IDC_DO_POP);
    pBtn->SetCheck(1);          // 使“收发邮件”复选框选中
    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

② 打开 MFC ClassWizard 对话框，在 CEx_ResearchDlg 类中添加 IDOK 按钮的 BN_CLICKED 消息映射，并添加下列代码：

```
void CEx_ResearchDlg::OnOK()
{
```

```
        CString str, strCtrl;   // 定义两个字符串变量，CString是操作字符串的MFC类
        // 获取第一个问题的用户选择
        str = "你的年龄：";
        UINT nID = GetCheckedRadioButton( IDC_AGE_L18, IDC_AGE_M38);
        GetDlgItemText(nID, strCtrl);   // 获取指定控件的标题文本
        str = str + strCtrl;
        // 获取第二个问题的用户选择
        str = str + "\n你使用的接入方式：";
        nID = GetCheckedRadioButton( IDC_CM_FTTL, IDC_CM_OTHER);
        GetDlgItemText(nID, strCtrl);   // 获取指定控件的标题文本
        str = str + strCtrl;
        // 获取第三个问题的用户选择
        str = str + "\n你上网主要是：\n";
        UINT nCheckIDs[4] = {IDC_DO_POP, IDC_DO_READ, IDC_DO_GAME, IDC_DO_
        OTHER};
        CButton* pBtn;
        for (int i=0; i<4; i++) {
              pBtn = (CButton*)GetDlgItem(nCheckIDs[i]);
              if ( pBtn->GetCheck() ) {
                  pBtn->GetWindowText( strCtrl );
                  str = str + strCtrl;
                  str = str + "  ";
            }
        }
        MessageBox( str );
        CDialog::OnOK();
    }
```

代码中，GetDlgItemText 是 CWnd 类成员函数，用来获得对话框（或其他窗口）中的指定控件的窗口文本。在单选按钮和复选框中，控件的窗口文本就是它们的标题属性内容。该函数有两个参数，第一个参数用来指定控件的标识，第二个参数是返回的窗口文本。后面的函数 GetWindowText 的作用与 GetDlgItemText 相同，也是获取窗口的文本内容。不过，GetWindowText 使用更加广泛，要注意这两个函数在使用上的不同。

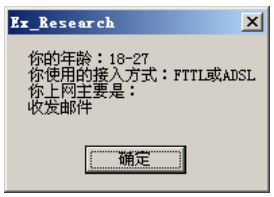

图 3.17　显示选择的内容

③ 编译并运行，出现“上网问卷调查”对话框，当回答问题后，按“确定”按钮，出现如图 3.17 所示的消息对话框，显示选择的结果内容。

3.3　编辑框和旋转按钮控件

编辑框（ab|）是一个让用户从键盘输入和编辑文本的矩形窗口，用户可以通过它很方便地输入各种文本、数字或者口令，也可使用它来编辑和修改简单的文本内容。

当编辑框被激活且具有输入焦点时，就会出现一个闪动的插入符（又可称为文本光标），表明当前插入点的位置。

3.3.1 编辑框的属性和通知消息

用对话框编辑器可以方便地设置编辑框的属性和风格，如图 3.18 所示。表 3.2 还列出其中各项的含义。

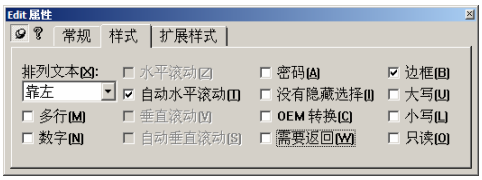

图 3.18 编辑框的属性对话框

表 3.2 编辑框的 Style 属性

项目	说明
排列文本（Align Text）	各行文本对齐方式：Left、Center、Right，默认为 Left
多行（Multiline）	选中时为多行编辑框，否则为单行编辑框
数字（Number）	选中时控件只能输入数字
水平滚动（Horizontal Scroll）	水平滚动，仅对多行编辑框有效
自动水平滚动（Auto HScroll）	当用户在行尾输入一个字符时，文本自动向右滚动
垂直滚动（Vertical Scroll）	垂直滚动，仅对多行编辑框有效
自动垂直滚动（Auto VScroll）	当用户在最后一行按 Enter 键时，文本自动向上滚动一页，仅对多行编辑框有效
密码（Password）	选中时，输入编辑框的字符都将显示为“*”，仅对单行编辑框有效
没有隐藏选择（No Hide Selection）	通常情况下，当编辑框失去键盘焦点时，被选择的文本仍然反色显示。选中时，则不具备此功能
OEM 转换（OEM Convert）	选中时，实现对特定字符集的字符转换
需要返回（Want Return）	选中时，用户按下 Enter 键，编辑框中就会插入一个回车符
边框（Border）	选中时，在控件的周围存在边框
大写（Uppercase）	选中时，输入在编辑框的字符全部转换成大写形式
小写（Lowercase）	选中时，输入在编辑框的字符全部转换成小写形式
只读（Read-Only）	选中时，防止用户输入或编辑文本

注意：多行编辑框具有简单文本编辑器的常用功能，例如它可以有滚动条，用户按 Enter 键另起一行以及文本的选定、复制、粘贴等常见操作。而单行编辑框功能较简单，它仅用于单行文本的显示和操作。

当编辑框的文本修改或者被滚动时，会向其父窗口发送一些消息，如表 3.3 所示。

表 3.3 编辑框的通知消息

通知消息	说明
EN_CHANGE	当编辑框中的文本已被修改，在新的文本显示之后发送此消息
EN_HSCROLL	当编辑框的水平滚动条被使用，在更新显示之前发送此消息
EN_KILLFOCUS	编辑框失去键盘输入焦点时发送此消息
EN_MAXTEXT	文本数目到达了限定值时发送此消息
EN_SETFOCUS	编辑框得到键盘输入焦点时发送此消息
EN_UPDATE	编辑框中的文本已被修改，新的文本显示之前发送此消息
EN_VSCROLL	当编辑框的垂直滚动条被使用，在更新显示之前发送此消息

3.3.2　编辑框的基本操作

由于编辑框的形式多样，用途各异，因此下面针对编辑框的不同用途，分别介绍一些常用操作，以实现一些基本功能。

1．口令设置

口令设置在编辑框中不同于一般的文本编辑框，用户输入的每个字符都被一个特殊的字符代替显示，这个特殊的字符称为口令字符。默认的口令字符是“*”，应用程序可以用成员函数 CEdit::SetPasswordChar 来定义自己的口令字符，其函数原型如下：

```
void SetPasswordChar( TCHAR ch );
```

其中，参数 ch 表示设定的口令字符；当 ch = 0 时，编辑框内将显示实际字符。

2．选择文本

当在编辑框中编辑文本时，往往需要选定文本作为整体进行各种编辑操作。用户可以用鼠标或键盘来选择文本。用鼠标来选择文本的操作方法是：在要选择的文本的一端按下鼠标左键并拖动鼠标，到另一端释放鼠标键。用键盘来选择文本的方法是：在按光标方向移动键的同时，按住 Shift 键。

在应用程序中也可以通过编程选择文本，这时需要通过调用成员函数 CEdit::SetSel 来实现。这个函数确定了编辑框内文本的选择范围，与该函数相对应的还有 CEdit::GetSel 和 CEdit::ReplaceSel，它们分别用来获取编辑框选择的开始和结束的位置以及替换被选择的文本。

3．设置编辑框的页面边距

设置编辑框的页面边距可以使文本在编辑框显示更具满意效果，这在多行编辑框中尤为重要，应用程序可通过调用成员函数 CEdit::SetMargins 来实现，这个函数的原型如下：

```
void SetMargins( UINT nLeft, UINT nRight );
```

其中，参数 nLeft 和 nRight 分别用来指定左、右边距的像素大小。

4．剪贴板操作

编辑框通过 CEdit 类的 Copy、Paste 和 Cut 成员函数来实现文本的复制、粘贴、剪切的操作，并还自动支持键盘快捷操作，其对应的快捷键分别为 Ctrl+C、Ctrl+V 和 Ctrl+X。若应用程序调用 CEdit::Undo 函数时，则还可撤销当前的操作，再调用一次该函数，则恢复刚才的操作。例如下面的代码：

```
if (m_Edit.CanUndo())   m_Edit.Undo();
```

5．获取多行编辑框文本

获取多行编辑框控件的文本有两种方法：

一种是使用 DDX/DDV，当将编辑框控件所关联的变量类型选定为 CString 后，则不管多行编辑框的文本有多少都可用此变量来保存，从而能简单地解决多行文本的读取。但这种方法不能单独获得多行编辑框中的某一行文本。

另一种方法是使用编辑框 CEdit 类的相关成员函数来获取文本。例如，下面的代码将显示编辑框中第二行的文本内容：

```
char str[100];
if (m_Edit.GetLineCount()>=2)        // 判断多行编辑框的文本是否有两行以上
{
      int nChars;
      nChars = m_Edit.LineLength(m_Edit.LineIndex(1));
      // 获取第二行文本的字符个数。0表示第一行，1表示第二行，依此类推
      // LineIndex用于将文本行转换成能被LineLength识别的索引
      m_Edit.GetLine(1,str,nChars); // 获取第二行文本
      str[nChars] = '\0';        MessageBox(str);
}
```

在上述代码中，由于调用 GetLine 获得某行文本内容时，并不能自动在文本后添加文本的结束符'\0'，因此需要首先获得某行文本的字符数，然后设置文本的结束符。

3.3.3 旋转按钮控件

“旋转按钮控件”（，也称为上下控件）是一对箭头按钮，用户单击它们来增加或减小某个值，比如一个滚动位置或显示在相应控件中的一个数字。

一个旋转按钮控件通常是与一个相伴的控件一起使用的，这个控件称为“伙伴窗口”。若相伴的控件的 Tab 键次序刚好在旋转按钮控件的前面，则这时的旋转按钮控件可以自动定位在它的伙伴窗口的旁边，看起来就像一个单一的控件。通常，将一个旋转按钮控件与一个编辑框一起使用，以提示用户进行数字输入。单击向上箭头使当前位置向最大值方向移动，而单击向下箭头使当前位置向最小值的方向移动，如图 3.19 所示。

默认时，旋转按钮控件的最小值是 100，最大值是 0。当用户单击向上箭头减少数值，而单击向下箭头则增加数值，这看起来就像颠倒一样，因此用户还需使用成员函数 CSpinButtonCtrl::SetRange 来改变最大值和最小值。但在使用时不要忘记在旋转按钮控件属性对话框中选中了“自动伙伴”（Auto Buddy），若选中“设置结伴整数”（Set Buddy Integer）属性，则伙伴窗口的数值自动改变。

1．旋转按钮控件常用的风格

旋转按钮控件有许多风格，它们都可以通过旋转按钮控件属性对话框进行设置，如图 3.20 所示。其中各项的含义见表 3.4 所示。

图 3.19　旋转按钮控件及其伙伴窗口

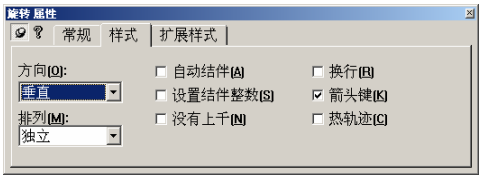

图 3.20　旋转按钮控件属性对话框

2．旋转按钮控件的基本操作

MFC 的 CSpinButtonCtrl 类提供了旋转按钮控件的各种操作函数，使用它们可以进行基数、范围、位置设置和获取等基本操作。

成员函数 SetBase 是用来设置其基数的，这个基数值决定了伙伴窗口显示的数字是十进制还是十六进制。如果成功则返回先前的基数值，如果给出的是一个无效的基数则返回一个非零值。函数的原型如下：

```
int SetBase( int nBase );
```

其中，参数 nBase 表示控件的新的基数，如 10 表示十进制，16 表示十六进制等。与此函数相对应的成员函数 GetBase 是获取旋转按钮控件的基数。

成员函数 SetPos 和 SetRange 分别用来设置旋转按钮控件的当前位置和范围，它们的函数原型如下：

```
int SetPos( int nPos );
void SetRange( int nLower, int nUpper );
```

其中，参数 nPos 表示控件的新位置，它必须在控件的上限和下限指定的范围之内。nLower 和 nUpper 表示控件的上限和下限。任何一个界限值都不能大于 0x7fff 或小于 −0x7fff。与这两个函数相对应的成员函数 GetPos 和 GetRange 分别用来获取旋转按钮控件的当前位置和范围。

3．旋转按钮控件的通知消息

旋转按钮控件的通知消息只有一个：UDN_DELTAPOS，它是在控件的当前数值将要改变时向其父窗口发送的。

表 3.4　旋转按钮控件的 Style 属性

项目	说明
方向（Orientation）	控件放置方向：Vertical（垂直）、Horizontal（水平）
排列（Alignment）	控件在伙伴窗口的位置安排：Unattached（不相干）、Right（右边）、Left（左边）
自动结伴（Auto Buddy）	选中此项，自动选择一个 Z-order 中的前一个窗口作为控件的伙伴窗口
自动结伴整数（Set Buddy Integer）	选中此项，使控件设置伙伴窗口数值，这个值可以是十进制或十六进制
没有上千（No Thousands）	选中此项，不在每隔三个十进制数字的地方加上千分隔符
换行（Wrap）	选中此项，当增加或减小的数值超出范围，则从最小值或最大值开始回绕
箭头键（Arrow Keys）	选中此项，当按下向上和向下方向键时，也能增加或减小
热轨迹（Hot Track）	选中此项，当光标移过控件时，突出显示控件的上下按钮

3.3.4　用对话框输入学生成绩示例

在一个简单的学生成绩结构中，常常有学生的姓名、学号以及三门成绩等内容。为了能够输入这些数据，需要设计一个对话框，如图 3.21 所示。本例将用到静态文本、编辑框、旋转按钮控件等控件。实现时，最关键的是如何将编辑框设置成旋转按钮控件的伙伴窗口。

1．添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl1SDI。

② 添加一个新的对话框资源，将 ID 号改为 IDD_INPUT，标题为“学生成绩输入”，将对话框字体改为“宋体，9 号”。将 OK 和 Cancel 按钮标题改为“确定”和“取消”。

③ 调整对话框的大小，将“确定”和“取消”按钮移至对话框的下方，然后显示对话框网格。向对话框添加如表 3.5 所示的控件，调整控件的位置，结果如图 3.22 所示。

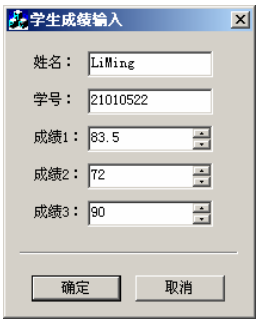

图 3.21　学生成绩输入对话框

图 3.22　设计的学生成绩输入对话框

表 3.5　学生成绩输入对话框添加的控件

添加的控件	ID 号	标题	其他属性
编辑框	IDC_EDIT_NAME	—	默认
编辑框	IDC_EDIT_NO	—	默认
编辑框	IDC_EDIT_S1	—	默认
旋转按钮控件	IDC_SPIN_S1	—	Auto buddy, Right 对齐
编辑框	IDC_EDIT_S2	—	默认
旋转按钮控件	IDC_SPIN_S2	—	Auto buddy, Set buddy integer, Right 对齐
编辑框	IDC_EDIT_S3	—	默认
旋转按钮控件	IDC_SPIN_S3	—	Auto buddy, Set buddy integer, Right 对齐

需要说明的是，由于控件的添加、布局和设置属性的方法以前已详细阐述过，为了节约篇幅，这里用表格形式列出所要添加的控件，并且因默认 ID 号的静态文本控件的“标题”属性内容可从对话框直接看出，因此一般不在表中列出，本书作此约定。

表格中 ID、标题和其他属性均是通过控件的属性对话框进行设置的，凡是“默认”属性均为保留属性对话框中的默认设置。

④ 选择“布局”→“Tab 次序”菜单命令，或按 Ctrl+D 键，此时每个控件的左上方都有一个数字，表明了当前 Tab 键次序，这个次序就是在对话框显示时按 Tab 键所选择控件的次序。

⑤ 单击对话框中的控件，重新设置控件的 Tab 键次序，以保证旋转按钮控件的 Tab 键次序在相对应的编辑框（伙伴窗口）之后，结果如图 3.23 所示，单击对话框或按 Enter 键结束 Tab Order 方式。

图 3.23　改变控件的 Tab 键次序

⑥ 双击对话框模板空白处，为该对话框模板创建一个对

话框类 CInputDlg。

2．完善 CInputDlg 类代码

① 在 MFC ClassWizard 的 Member Variables 选项卡中，确定 Class name 中是否已选择了 CInputDlg，选中所需的控件 ID 号，双击鼠标或单击 Add Variables 按钮。依次为表 3.6 控件增加成员变量。

② 在 MFC ClassWizard 的 Messsage Maps 选项卡中，为 CInputDlg 添加 WM_INITDIALOG 消息映射，并添加下列代码：

```
BOOL CInputDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_spinScore1.SetRange( 0, 100 );        // 设置旋转按钮控件范围
    m_spinScore2.SetRange( 0, 100 );
    m_spinScore3.SetRange( 0, 100 );
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

表 3.6　控件变量

控件 ID 号	变量类别	变量类型	变量名	范围和大小
IDC_EDIT_NAME	Value	CString	m_strName	20
IDC_EDIT_NO	Value	CString	m_strNO	20
IDC_EDIT_S1	Value	float	m_fScore1	0.0 ~ 100.0
IDC_SPIN_S1	Control	CSpinButtonCtrl	m_spinScore1	—
IDC_EDIT_S2	Value	float	m_fScore2	0.0 ~ 100.0
IDC_SPIN_S2	Control	CSpinButtonCtrl	m_spinScore2	—
IDC_EDIT_S3	Value	float	m_fScore3	0.0 ~ 100.0
IDC_SPIN_S3	Control	CSpinButtonCtrl	m_spinScore3	—

③ 用 MFC ClassWizard 为 CInputDlg 增加 IDC_SPIN_S1 控件的 UDN_DELTAPOS 消息映射，并添加下列代码：

```
void CInputDlg::OnDeltaposSpinS1(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_UPDOWN* pNMUpDown = (NM_UPDOWN*)pNMHDR;
    UpdateData(TRUE);       // 将控件的内容保存到变量中
    m_fScore1 += (float)pNMUpDown->iDelta * 0.5f;
    if (m_fScore1<0.0)
        m_fScore1 = 0.0f;
    if (m_fScore1>100.0)
        m_fScore1 = 100.0f;
    UpdateData(FALSE);      // 将变量的内容显示在控件中
    *pResult = 0;
}
```

在上述代码中，LPNMUPDOWN 是 NMUPDOWN 结构指针类型，NMUPDOWN 结构

用于反映旋转控件的当前位置(由成员 iPos 指定)和增量大小(由成员 iDelta 指定)。

3．调用对话框

① 打开 Ex_Ctrl1SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在其下添加一个菜单项“学生成绩输入(&I)”，ID 为 ID_TEST_INPUT。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_INPUT 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestInput()
{
      CInputDlg dlg;
      if (IDOK == dlg.DoModal())    {
            // 获取对话框数据
        CString str;
        str.Format("%s, %s, %4.1f, %4.1f, %4.1f",
               dlg.m_strName, dlg.m_strNO, dlg.m_fScore1, dlg.m_fScore2,
               dlg.m_fScore3 );
        AfxMessageBox(str);
    }
}
```

在上述代码中，if 语句判断用户是否单击对话框的“确定”按钮。Format 是 CString 类的一个经常使用的成员函数，它通过格式操作使任意类型的数据转换成一个字符串。该函数的第一个参数是带格式的字符串，其中的“%s”就是一个格式符，每一个格式符依次对应于该函数的后面参数表中的参数项。例如格式字符串中第一个%s 对应于 dlg.m_strName。CString 类的 Format 和 C 语言库函数 printf 十分相似。

③ 在文件 MainFrm.cpp 的前面添加 CInputDlg 类的头文件包含：

```
#include "Ex_Ctrl1SDI.h"
#include "MainFrm.h"
#include "InputDlg.h"
```

④ 编译并运行，在应用程序的菜单上，选择“测试”→“学生成绩输入”菜单项，将弹出如图 3.21 所示的对话框。单击成绩 1 的旋转按钮控件将以 0.5 增量来改变它的伙伴窗口的数值。而成绩 2 和成绩 3 的旋转按钮控件由于设置了“设置结伴整数”属性，因此它按默认增量 1 自动改变伙伴窗口的数值。

3.4 列表框

列表框（  ）是一个列有许多项目让用户选择的控件。它与单选按钮组或复选框组一样，可让用户在其中选择一个或多个项，但不同的是，列表框中项的数目是可灵活变化的，程序运行时可往列表框中添加或删除某些项。并且，当列表框中项的数目较多，而不能一

次全部显示时，还可以自动提供滚动条来让用户浏览其余的列表项。

3.4.1　列表框的风格和消息

按性质来分，列表框有单选、多选、扩展多选以及非选 4 种类型，如图 3.24 所示。默认样式下的单选列表框让用户一次只能选择一个项，多选列表框可让用户一次选择几个项，而扩展多选列表框允许用户用鼠标拖动或其他特殊组合键进行选择，非选列表框则不提供选择功能。

列表框还有一系列其他风格，用来定义列表框的外观及操作方式，这些风格可在如图 3.25 所示的列表框属性对话框中设置。表 3.7 列出样式（Style）各项的含义。

图 3.24　不同类型的列表框

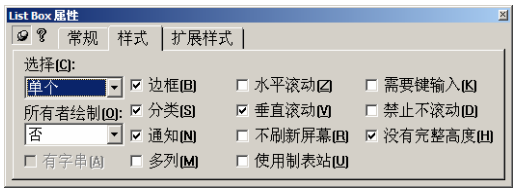

图 3.25　列表框的属性对话框

表 3.7　列表框的样式属性

项目	说明
选择（Selection）	指定列表框的类型：单选（Single）、多选（Multiple）、扩展多选（Extended）、不选（None）
所有者绘制（Owner Draw）	自画列表框，默认为 No
有字符串（Has Strings）	选中时，在自画列表框中的项目中含有字符串文本
边框（Border）	选中时，使列表框含有边框
排序（分类）（Sort）	选中时，列表框的项目按字母顺序排列
通知（Notify）	选中时，当用户对列表框操作，就会向父窗口发送通知消息
多列（Multi-Column）	选中时，指定一个具有水平滚动的多列列表框
水平滚动（Horizontal Scroll）	选中时，在列表框中创建一个水平滚动条
垂直滚动（Vertical Scroll）	选中时，在列表框中创建一个垂直滚动条
不刷新屏幕（No Redraw）	选中时，列表框发生变化后不会自动重画
使用制表站（位）（Use Tabstops）	选中时，允许使用停止位来调整列表项的水平位置
需要键输入（Want Key Input）	选中此项，当用户按键且列表框有输入焦点时，就会向列表框的父窗口发送相应消息
禁止不滚动（Disable No Scroll）	选中时，即使列表框的列表项能全部显示，垂直滚动条也会显示，但此时是禁用的（灰显）
没有完整高度（No Integral Height）	选中时，在创建列表框的过程中，系统会把用户指定的尺寸完全作为列表框的尺寸，而不管是否会有项目在列表框不能完全显示出来

当列表框中发生了某个动作，如用户双击选择了列表框中某一项时，列表框就会向其父窗口发送一条通知消息。常用的通知消息如表 3.8 所示。

表 3.8 列表框的通知消息

通知消息	说明
LBN_DBLCLK	用户双击列表框的某项字符串时发送此消息
LBN_KILLFOCUS	列表框失去键盘输入焦点时发送此消息
LBN_SELCANCEL	当前选择项被取消时发送此消息
LBN_SELCHANGE	列表框中的当前选择项将要改变时发送此消息
LBN_SETFOCUS	列表框获得键盘输入焦点时发送此消息

3.4.2 列表框的基本操作

当列表框创建之后，往往要添加、删除、改变或获取列表框中的列表项，这些操作都可以调用 MFC 封装 CListBox 类的成员函数来实现。需要注意的是：列表框的项除了用字符串来标识外，还常常通过索引来确定。索引表明项目在列表框中排列的位置，它是以 0 为基数的，即列表框中第一项的索引是 0，第二项的索引是 1，依次类推。

1．添加列表项

列表框创建时是一个空的列表，需要用户添加或插入一些列表项。CListBox 类成员函数 AddString 和 InsertString 分别用来向列表框增加列表项，其函数原型如下：

```
int AddString( LPCTSTR lpszItem );
int InsertString( int nIndex, LPCTSTR lpszItem );
```

其中，列表项的字符串文本由参数 pszItem 来指定。虽然两个函数成功调用时都将返回列表项在列表框的索引，错误时返回 LB_ERR，空间不够时，返回 LB_ERRSPACE。但 InsertString 函数不会对列表项进行排序，不管列表框控件是否具有“排序（分类）”（Sort）属性，只是将列表项插在指定索引的列表项之前，若 nIndex 等于-1，则列表项添加在列表框末尾。而 AddString 函数当列表框控件具有“排序（分类）”属性时会自动将添加的列表项进行排序。

上述两个函数只能将字符串增加到列表框中，但有时用户还会需要根据列表项使用其他数据。这时，就需要调用 CListBox 的 SetItemData 和 SetItemDataPtr，它们能使用户数据和某个列表项关联起来。

```
int SetItemData( int nIndex, DWORD dwItemData );
int SetItemDataPtr( int nIndex, void* pData );
```

其中，SetItemData 是将一个 32 位数与某列表项（由 nIndex 指定）关联起来，而 SetItemDataPtr 可以将用户的数组、结构体等大量的数据与列表项关联。若有错误产生时，两个函数都将返回 LB_ERR。

与上述函数相对应的两个函数 GetItemData 和 GetItemDataPtr 分别用来获取相关联的用户数据。

2．删除列表项

CListBox 类成员函数 DeleteString 和 ResetContent 分别用来删除指定的列表项和清除列表框所有项目。它们的函数原型如下：

```
int DeleteString( UINT nIndex );        // nIndex指定要删除的列表项的索引
void ResetContent( );
```

需要注意的是，若在添加列表项时使用 SetItemDataPtr 函数，不要忘记在进行删除操作时及时将关联数据所占的内存空间释放出来。

3．查找列表项

为了保证列表项不会重复地添加在列表框中，有时还需要对列表项进行查找。CListBox 类成员函数 FindString 和 FindStringExact 分别用来在列表框中查找所匹配的列表项。其中，FindStringExact 的查找精度最高。

```
int FindString( int nStartAfter, LPCTSTR lpszItem ) const;
int FindStringExact( int nIndexStart, LPCTSTR lpszFind ) const;
```

其中，lpszFind 和 lpszItem 指定要查找的列表项文本，nStartAfter 和 nIndexStart 指定查找的开始位置，若为-1，则从头至尾查找。查到后，这两个函数都将返回所匹配列表项的索引，否则返回 LB_ERR。

4．列表框的单项选择

当选中列表框中某个列表项，用户可以使用 CListBox::GetCurSel 来获取这个结果，与该函数相对应的 CListBox::SetCurSel 函数是用来设定某个列表项呈选中状态（高亮显示）。

```
int GetCurSel( ) const;          // 返回当前选择项的索引
int SetCurSel( int nSelect );
```

其中，nSelect 指定要设置的列表项索引，错误时这两个函数都将返回 LB_ERR。

若要获取某个列表项的字符串，可使用下列函数：

```
int GetText( int nIndex, LPTSTR lpszBuffer ) const;
void GetText( int nIndex, CString& rString ) const;
```

其中，nIndex 指定列表项索引，lpszBuffer 和 rString 是用来存放列表项文本。

5．列表框的多项选择

当在列表框的 Style 属性对话框中选中多选（Multiple）或扩展多选（Extended）类型后，就可以在列表框中进行多项选择。要想获得选中的多个选项，需要用 MFC ClassWizard 映射列表框控件的 LBN_SELCHANGE 消息，并添加类似下面的一些代码：

```
void CListBoxDlg::OnSelchangeList1()
{
     int nCount = m_list.GetSelCount(); // 获取用户选中的项数
     if (nCount == LB_ERR)  return;
     int *buffer = new int[nCount];     // 开辟缓冲区
     m_list.GetSelItems(nCount,buffer); // 将各个选项的索引号内容存放在缓冲区中
     CString allStr = NULL, str;
     for (int i=0; i<nCount; i++)
     {
          m_list.GetText(buffer[i], str); // 获得各个索引的项目文本
```

```
            allStr = allStr + "[" + str + "]";   // 处理项目文本
        }
        delete []buffer;                          // 释放内存
        // MessageBox(allStr);                    // 处理获得的文本
    }
```

3.4.3 创建并使用城市邮政编码对话框示例

在一组城市邮政编码中，城市名和邮政编码是一一对应的。为了能添加和删除城市邮政编码列表项，需要设计一个这样的对话框，如图 3.26 所示。单击“添加”按钮，将城市名和邮政编码添加到列表框中，为了使添加不重复，还要进行一些判断操作，单击列表框的城市名，将在编辑框中显示出城市名和邮政编码，单击“删除”按钮，删除当前的列表项。实现本例有两个要点：一是在添加时需要通过 FindString 或 FindStringExact 来判断添加的列表项是否重复，然后通过 SetItemData 将邮政编码（将它视为一个 32 位整数）与列表项关联起来；二是由于删除操作是针对当前选中的列表项的，如果当前没有选中的列表项则应通过 EnableWindow(FLASE)使“删除”按钮灰显，即不能单击它。

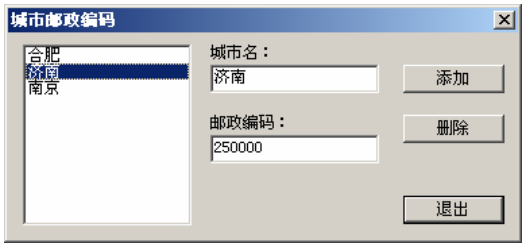

图 3.26 城市邮政编码

1．添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl2SDI。

② 向应用程序中添加一个对话框资源 IDD_CITYZIP，标题定为“城市邮政编码”，字体设为“宋体，9 号”，创建此对话框类为 CCityDlg。删除原来的 Cancel 按钮，将 OK 按钮标题改为“退出”

③ 打开对话框网格，参照图 3.27 的控件布局，用编辑器为对话框添加如表 3.9 所示的一些控件。

表 3.9 城市邮政编码对话框添加的控件

添加的控件	ID 号	标题	其他属性
列表框	IDC_LIST1	—	默认
编辑框（城市名）	IDC_EDIT_CITY	—	默认
编辑框(邮政编码)	IDC_EDIT_ZIP	—	默认
按钮（添加）	IDC_BUTTON_ADD	添加	默认
按钮（修改）	IDC_BUTTON_DEL	修改	默认

2．完善 CCityDlg 类代码

① 打开 ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CCityDlg，然后选中所需的控件 ID 号，双击鼠标或单击 Add Variables 按钮，依次为下列控件增加成员变量，如表 3.10 所示。

表 3.10　控件变量

控件 ID 号	变量类别	变量类型	变量名	范围和大小
IDC_LIST1	Control	CListBox	m_ListBox	—
IDC_EDIT_CITY	Value	CString	m_strCity	40
IDC_EDIT_ZIP	Value	DWORD	m_dwZipCode	100 000~999 999

② 将项目工作区切换到 ClassView 选项卡，右击 CCityDlg 类名，从弹出的快捷菜单中选择 Add Member Function 命令，弹出如图 3.27 所示的“添加成员函数”对话框，在“函数类型”（Function Type）文本框中输入 BOOL，在“函数描述”（Function Declaration）文本框中输入 IsValidate，单击“确定”按钮。

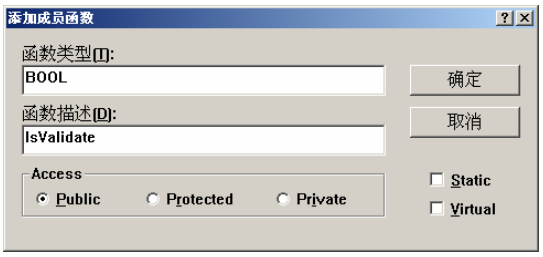

图 3.27　添加成员函数

③ 在 CCityDlg::IsValidate 函数中输入下列代码：

```
BOOL CCityDlg::IsValidate()
{
    UpdateData();
    m_strCity.TrimLeft();
    if (m_strCity.IsEmpty()) {
          MessageBox("城市名输入无效！"); return FALSE;
    }
    return TRUE;
}
```

IsValidate 函数的功能是判断城市名编辑框中的内容是否是有效的字符串。TrimLeft 是 CString 类的一个成员函数，用来去除字符串左边的空格。

④ 打开 MFC ClassWizard，切换到 Messsage Maps 选项卡，为对话框添加 WM_INITDIALOG 消息映射，并增加下列代码：

```
BOOL CCityDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
```

```
    m_dwZipCode = 100000;           // 设置初始的邮政编码
    UpdateData( FALSE );            // 将邮政编码显示在控件中
    GetDlgItem(IDC_BUTTON_DEL)->EnableWindow(FALSE); // 使“删除”按钮显灰
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

⑤ 打开 MFC ClassWizard，切换到 Messsage Maps 选项卡，为按钮 IDC_BUTTON_ADD 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CCityDlg::OnButtonAdd()
{
    if (!IsValidate()) return;
    int nIndex = m_ListBox.FindStringExact( -1, m_strCity );
    if (nIndex != LB_ERR ){
        MessageBox("该城市已添加！");       return;
    }
    nIndex = m_ListBox.AddString( m_strCity );
    m_ListBox.SetItemData( nIndex, m_dwZipCode );
}
```

⑥ 用 MFC ClassWizard 为按钮 IDC_BUTTON_DEL 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CCityDlg::OnButtonDel()
{
    int nIndex = m_ListBox.GetCurSel();
    if (nIndex != LB_ERR ){
        m_ListBox.DeleteString( nIndex );
    } else
        GetDlgItem(IDC_BUTTON_DEL)->EnableWindow( FALSE );
}
```

⑦ 用 MFC ClassWizard 为列表框 IDC_LIST1 添加 LBN_SELCHANGE（当前选择项发生变化产生的消息）的消息映射，并增加下列代码。这样，当单击列表框的城市名时，将会在编辑框中显示出城市名和邮政编码。

```
void CCityDlg::OnSelchangeList1()
{
    int nIndex = m_ListBox.GetCurSel();
    if (nIndex != LB_ERR ){
        m_ListBox.GetText( nIndex, m_strCity );
        m_dwZipCode = m_ListBox.GetItemData( nIndex );
        UpdateData( FALSE );    // 使当前列表项所关联的内容显示在控件上
        GetDlgItem(IDC_BUTTON_DEL)->EnableWindow( TRUE );
    }
}
```

3．调用对话框

① 打开 Ex_Ctrl2SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在其下添加一个菜单项“城市邮政编码(&C)”，ID 为 ID_TEST_CITY。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_CITY 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestCity()
{
    CCityDlg dlg;
    dlg.DoModal();
}
```

③ 在文件 MainFrm.cpp 的前面添加 CCityDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "CityDlg.h"
```

④ 编译运行后，在应用程序的菜单中，选择“测试”→“城市邮政编码”菜单项，将弹出如前图 3.26 所示的对话框。

3.5　组合框

作为用户输入的接口，前面的列表框和编辑框各有其优点。例如，列表框中可列出用户所需的各种可能的选项，这样一来，用户不需要记住这些项，只需进行选择操作即可，但用户却不能输入列表框中列表项之外的内容。虽然编辑框能够允许用户输入内容，但却没有列表框的选择操作。于是很自然地产生这样的想法：把常用的项列在列表框中以供选择，而同时提供编辑框，允许用户输入列表框中所没有的新项。组合框正是这样的一种控件，它结合列表框和编辑框的特点，取二者之长，从而完成较为复杂的输入功能。

3.5.1　组合框的风格类型和消息

按照组合框的主要风格特征，可把组合框分为三类：简单组合框、下拉式组合框、下拉式列表框，如图 3.28 所示。

简单组合框和下拉式组合框都包含有列表框和编辑框，但是简单组合框中的列表框不需要下拉，是直接显示出来的，而当用户单击下拉式组合框中的下拉按钮时，下拉的列表框才被显示出来。下拉式列表框虽然具有下拉式的列表，却没有文字编辑功能。

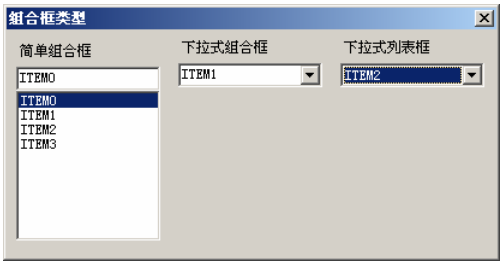

图 3.28　组合框的类型

组合框还有其他一些风格，这些风格可在如图 3.29 所示的组合框的属性对话框中设置。

其各项含义见表 3.11。

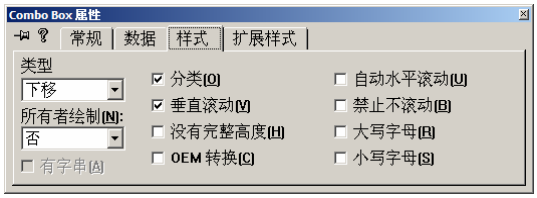

图 3.29　组合框的属性对话框

表 3.11　组合框的 Style 属性

项目	说明
类型（Type）	设置组合框的类型：Simple（简单）、Dropdown（下拉）、Drop List（下拉列表框）
所有者绘制（Owner Draw）	自画组合框，默认为 No
有字符串（Has Strings）	选中时，在自画组合框中的项目中含有字符串文本
排序（分类）（Sort）	选中时，组合框的项目按字母顺序排列
垂直滚动（Vertical Scroll）	选中时，在组合框中创建一个垂直滚动条
没有完整高度（No Integral Height）	选中时，在创建组合框的过程中，系统会把用户指定的尺寸完全作为组合框的尺寸，而不管是否会有项目在组合框中的列表中不能完全显示出来
OEM 转换（OEM Convert）	选中时，实现对特定字符集的字符转换
自动水平滚动（Auto HScroll）	当用户在行尾输入一个字符时，文本自动向右滚动
禁止不滚动（Disable No Scroll）	选中时，即使组合框的列表项能全部显示，垂直滚动条也会显示，但此时是禁用的（灰显）
大写字母（Uppercase）	选中时，输入在编辑框的字符全部转换成大写形式
小写字母（Lowercase）	选中时，输入在编辑框的字符全部转换成小写形式

需要说明的是，组合框属性对话框中，“数据”（Data）选项卡可以让用户直接输入组合框的数据项，每输入一条数据项后，按 Ctrl+Enter 键可继续输入下一条数据项。

在组合框的通知消息中，有的是列表框发出的，有的是编辑框发出的，如表 3.12 所示。

表 3.12　组合框的通知消息

通知消息	说明
CBN_CLOSEUP	当组合框的列表关闭时发送此消息
CBN_DBLCLK	用户双击组合框的某项字符串时发送此消息
CBN_DROPDOWN	当组合框的列表打开时发送此消息
CBN_EDITCHANGE	同编辑框的 EN_CHANGE 消息
CBN_EDITUPDATE	同编辑框的 EN_UPDATE 消息
CBN_SELENDCANCEL	当前选择项被取消时发送此消息
CBN_SELENDOK	当用户选择一个项并按下 Enter 键或单击下拉箭头（▾）隐藏列表框时发送此消息
CBN_KILLFOCUS	组合框失去键盘输入焦点时发送此消息
CBN_SELCHANGE	组合框中的当前选择项将要改变时发送此消息
CBN_SETFOCUS	组合框获得键盘输入焦点时发送此消息

3.5.2　组合框常见操作

组合框的操作大致分为两类，一类是对组合框中的列表框进行操作，另一类是对组合框中的编辑框进行操作。这些操作都可以调用 CComboBox 成员函数来实现，如表 3.13 所示。

表 3.13　CComboBox 类常用成员函数

成员函数	作用	说明
int AddString(LPCTSTR lpszString);	向组合框添加字符串	错误时返回 CB_ERR；空间不够时，返回 CB_ERRSPACE
int DeleteString(UINT nIndex);	删除指定的索引项	返回剩下的列表项总数，错误时返回 CB_ERR
int InsertString(int nIndex, LPCTSTR lpszString);	在指定的位置处插入字符串，若 nIndex=−1 时，向组合框尾部添加	返回插入字符串的索引，错误时返回 CB_ERR；空间不够时，返回 CB_ERRSPACEf
void ResetContent( );	删除组合框的全部项和编辑文本	
int FindString(int nStartAfter, LPCTSTR lpszString) const;	查找字符串	参数 1=搜索起始项的索引，–1 时从头开始，参数 2=被搜索字符串
int FindStringExact(int nIndexStart, LPCTSTR lpszFind) const；	精确查找字符串	返回匹配项的索引，错误时返回 CB_ERR
int SelectString(int nStartAfter, LPCTSTR lpszString);	选定指定字符串	返回选择项的索引，若当前选择项没有改变则返回 CB_ERR
int GetCurSel( ) const;	获得当前选择项的索引	当没有当前选择项时返回 CB_ERR
int SetCurSel(int nSelect);	设置当前选择项	参数为当前选择项的索引，−1 时，没有选择项。错误时返回 CB_ERR
int GetCount( ) const;	获取组合框的项数	错误时返回 CB_ERR
int SetDroppedWidth(UINT nWidth);	设置下拉组合框的最小像素宽度	成功时，返回新的组合框宽度，否则返回 CB_ERR
int SetItemData(int nIndex, DWORD dwItemData);	将一个 32 位值和指定列表项关联	错误时返回 CB_ERR
int SetItemDataPtr(int nIndex, void* pData);	将一个值的指针和指定列表项关联	错误时返回 CB_ERR
DWORD GetItemData(int nIndex) const;	获取和指定列表项关联的一个 32 位值	错误时返回 CB_ERR
void* GetItemDataPtr(int nIndex) const;	获取和指定列表项关联的一个值的指针	错误时返回−1
int GetLBText(int nIndex, LPTSTR lpszText);	获取指定项的字符串	返回字符串的长度，若每一个参数无效时返回 CB_ERR
void GetLBText(int nIndex, CString& rString);		
int GetLBTextLen(int nIndex) const；	获取指定项的字符串长度	若参数无效时返回 CB_ERR

由于组合框的一些编辑操作与编辑框 CEdit 的成员函数相似，如 GetEditSet、SetEditSel 等，因此这些成员函数没有在上述表中列出。

3.5.3 创建并使用城市邮政编码和区号对话框示例

在前面的示例中，只是简单地涉及城市名和邮政编码的对应关系。实际上，城市名还和区号一一对应，为此本例需要设计这样的对话框，如图 3.30 所示。

图 3.30 城市邮政编码和区号

单击“添加”按钮将城市名、邮政编码和区号添加到组合框中，在添加前同样需要进行重复性的判断。选择组合框中的城市名，将在编辑框中显示出邮政编码和区号，单击“修改”按钮，将以城市名作为组合框的查找关键字，找到后修改其邮政编码和区号内容。

实现本例最关键的技巧是如何使组合框中的项关联邮政编码和区号内容，这里先将邮政编码和区号变成一个字符串，中间用逗号分隔，然后通过 SetItemDataPtr 将字符串和组合框中的项相关联。由于 SetItemDataPtr 关联的是一个数据指针，因此需要用 new 运算符为要关联的数据分配内存，同时在对话框即将关闭时，需要用 delete 运算符来释放组合框中的项所关联的所有数据的内存空间。

1．添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl3SDI。

② 向应用程序中添加一个对话框资源 IDD_CITYZONE，标题定为“城市邮政编码和区号”，字体设为“宋体，9 号”，创建此对话框类为 CCityZoneDlg。删除原来的 Cancel 按钮，将 OK 按钮标题改为“退出”。

③ 打开对话框网格，参照图 3.31 所示的控件布局，为对话框添加如表 3.14 所示的一些控件。

表 3.14 城市邮政编码和区号对话框添加的控件

添加的控件	ID 号	标题	其他属性
组合框	IDC_COMBO1	—	默认
编辑框(邮政编码)	IDC_EDIT_ZIP	—	默认
编辑框(区号)	IDC_EDIT_ZONE	—	默认
按钮(添加)	IDC_BUTTON_ADD	添加	默认
按钮(修改)	IDC_BUTTON_CHANGE	修改	默认

需要说明的是，在组合框添加到对话框模板后，一定要单击组合框的下拉按钮（▼），然后调整出现的下拉框大小（见图 3.31），否则组合框可能因为下拉框太小而无法显示其下拉列表项。

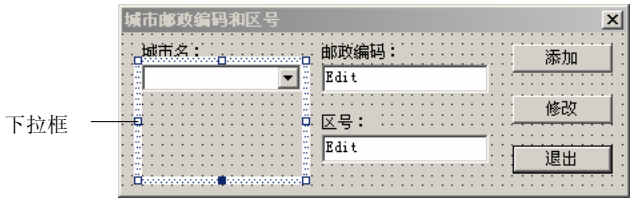

图 3.31　调整组合框的下拉框

2．完善 CCityZoneDlg 类代码

① 打开 MFC ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CCityZoneDlg，然后选中所需的控件 ID 号，双击鼠标或单击 Add Variables 按钮。依次为下列控件增加成员变量，如表 3.15 所示。

表 3.15　控件变量

控件 ID 号	变量类别	变量类型	变量名	范围和大小
IDC_COMBO1	Control	CComboBox	m_ComboBox	—
IDC_COMBO1	Value	CString	m_strCity	20
IDC_EDIT_ZONE	Value	CString	m_strZone	10
IDC_EDIT_ZIP	Value	CString	m_strZip	6

② 将项目工作区切换到 ClassView 选项卡，右击 CCityZoneDlg 类名，从弹出的快捷菜单中选择 Add Member Function 命令，弹出“添加成员函数”对话框，在“函数类型”框中输入“BOOL”，在“函数声明”框中输入“IsValidate”，单击“确定”按钮。

③ 在 CCityZoneDlg::IsValidate 函数中输入下列代码：

```
BOOL CCityZoneDlg::IsValidate()
{
    UpdateData();
    m_strCity.TrimLeft();
    if (m_strCity.IsEmpty()){
        MessageBox("城市名输入无效！");      return FALSE;
    }
    m_strZip.TrimLeft();
    if (m_strZip.IsEmpty()) {
        MessageBox("邮政编码输入无效！");    return FALSE;
    }
    m_strZone.TrimLeft();
    if (m_strZone.IsEmpty())    {
        MessageBox("区号输入无效！");        return FALSE;
    }
    return TRUE;
}
```

④ 打开 MFC ClassWizard，切换到 Messsage Maps 选项卡，为按钮 IDC_BUTTON_ADD 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CCityZoneDlg::OnButtonAdd()
{
    if (!IsValidate()) return;
    int nIndex = m_ComboBox.FindStringExact( -1, m_strCity );
    if (nIndex != CB_ERR ){
        MessageBox("该城市已添加！");     return;
    }
    CString strData;
    strData.Format("%s,%s", m_strZip, m_strZone);
    // 将邮政编码和区号合并为一个字符串
    m_ComboBox.SetItemDataPtr( nIndex, new CString(strData) );
}
```

⑤ 用 MFC ClassWizard 为按钮 IDC_BUTTON_CHANGE 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CCityZoneDlg::OnButtonChange()
{
    if (!IsValidate()) return;
    int nIndex = m_ComboBox.FindStringExact( -1, m_strCity );
    if (nIndex != CB_ERR ){
        delete (CString*)m_ComboBox.GetItemDataPtr( nIndex );
        CString strData;
        strData.Format("%s,%s", m_strZip, m_strZone);
        m_ComboBox.SetItemDataPtr( nIndex, new CString(strData) );
    }
}
```

⑥ 用 MFC ClassWizard 为组合框 IDC_COMBO1 添加 CBN_SELCHANGE（当前选择项发生改变时发出的消息）的消息映射，并增加下列代码：

```
void CCityZoneDlg::OnSelchangeCombo1()
{
    int nIndex = m_ComboBox.GetCurSel();
    if (nIndex != CB_ERR ){
        m_ComboBox.GetLBText( nIndex, m_strCity );
        CString strData;
        strData = *(CString*)m_ComboBox.GetItemDataPtr( nIndex );
        // 分解字符串
        int n = strData.Find(',');
        m_strZip = strData.Left( n );    // 前面的n个字符
        m_strZone = strData.Mid( n+1 ); // 从中间第n+1字符到末尾的字符串
        UpdateData( FALSE );
    }
}
```

⑦ 用 MFC ClassWizard 为对话框添加 WM_DESTROY 的消息映射，并增加下列代码：

```
void CCityZoneDlg::OnDestroy()  // 此消息是当对话框关闭时发送的
{
     for (int nIndex = m_ComboBox.GetCount()-1; nIndex>=0; nIndex--)
     {
        // 删除所有与列表项相关联的CString数据，并释放内存
        delete (CString *)m_ComboBox.GetItemDataPtr(nIndex);
     }
    CDialog::OnDestroy();
}
```

需要说明的是，当对话框从屏幕消失后，对话框被清除时发送 WM_DESTROY 消息。在此消息的映射函数中添加一些对象删除代码，以便在对话框清除前有效地释放内存空间。

3．调用对话框

① 打开 Ex_Ctrl3SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在其下添加一个菜单项“城市邮政编码和区号(&Z)”，ID 为 ID_TEST_CITYZONE。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_CITYZONE 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestCityzone()
{
     CCityZoneDlg dlg;
     dlg.DoModal();
}
```

③ 在文件 MainFrm.cpp 的前面添加 CCityZoneDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "CityZoneDlg.h"
```

④ 编译运行并测试。

3.6　进展条、滚动条和滑动条

进展条通常用来说明一个操作的进度，并在操作完成时从左到右填充进展条，这个过程可以让用户看到任务还有多少要完成。而滚动条和滑动条可以完成诸如定位之类的操作。

3.6.1　进展条

进展条（又称进程条）是一个如图 3.32 所示的控件。除了能表示一个过程进展情况外，使用进展条还可表示温度、水平面或类似的测量值。

1．进展条的风格

打开进展条的属性对话框，如图 3.33 所示，可以看到它的风格属性并不是很多。其中，“边框”（Border）用来指定进展条是否有边框，“垂直”（Vertical）用来指定进展条是水平还是垂直的，不选中该属性，表示进展条从左到右水平显示。“平滑”（Smooth）表示平滑地填充进展条，若不选中则表示将用块来填充，如图 3.32 所示。

图 3.32　进展条

图 3.33　进展条“样式”属性对话框

2．进展条的基本操作

进展条的基本操作有：设置其范围、当前位置、设置增量等。这些操作都是通过 CProgressCtrl 类的相关成员函数来实现的。

```
int SetPos( int nPos );
int GetPos();
```

这两个函数分别用来设置和获取进展条的当前位置。需要说明的是，这个当前位置是指在 SetRange 中的上限和下限范围之间的位置。

```
void SetRange( short nLower, short nUpper );
void SetRange32(int nLower, int nUpper );
void GetRange( int & nLower, int& nUpper );
```

它们分别用来设置和获取进展条范围的上限和下限。一旦设置后，还会重画此进展条来反映新的范围。成员函数 SetRange32 为进展条设置 32 位的范围。参数 nLower 和 nUpper 分别表示范围的下限(默认值为 0)和上限(默认值为 100)。

```
int SetStep( int nStep );
```

该函数用来设置进展条的步长并返回原来的步长，默认步长为 10。

```
int StepIt();
```

该函数将当前位置向前移动一个步长并重画进展条以反映新的位置。函数返回进展条上一次的位置。

3．使用进展条示例

该示例创建的对话框，内有一个进展条、一个静态文本和两个按钮，如图 3.34 所示。单击“继续”按钮，进展条向前进，单击“后退”按钮，进展条向后退，静态文本中还显示出进展条的百分比。实现本例最主要的是进展条的百分比显示，这里通过 CProgressCtrl 类的 GetPos 和 GetRange 来获

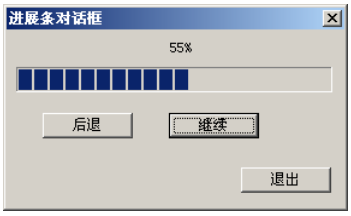

图 3.34　进展条对话框

取当前进展条当前位置和范围，并根据范围计算当前位置所处的百分比，然后将百分比转换成字符，显示在文本控件中。

（1）添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl4SDI。

② 向应用程序中添加一个对话框资源 IDD_PROGRESS，标题定为“进展条对话框”，字体设为“宋体，9 号”，创建此对话框类为 CProgressDlg。删除原来的 Cancel 按钮，将 OK 按钮标题改为“退出”。

③ 打开对话框网格，用编辑器为对话框添加如表 3.16 所示的一些控件，调整控件的位置，结果如图 3.34 所示。

表 3.16　进展条对话框添加的控件

添加的控件	ID 号	标题	其他属性
静态文本	IDC_STATIC_TEXT	默认	对齐设为 Center，其余默认
进展条	IDC_PROGRESS1	—	默认
按钮(后退)	IDC_BUTTON_BACK	—	默认
按钮(继续)	IDC_BUTTON_GOON	—	默认

（2）完善 CProgressDlg 类代码

① 打开 ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CProgressDlg，选中控件 ID 号 IDC_STATIC_TEXT，双击鼠标或单击 Add Variables 按钮，为其添加一个 CString 类型变量 m_strPercent。再为进展条 IDC_PROGRESS1 控件添加一个 CProgressCtrl 类变量 m_Progress。

② 为 CProgressDlg 类添加一个成员函数 UpdatePercentText，用来当进展条位置变化后更新静态文本控件显示的百分比。代码如下：

```
void CProgressDlg::UpdatePercentText()
{
    int nPos = m_Progress.GetPos();                    // 获取进展条当前位置
    int nLow, nUp;
    m_Progress.GetRange(nLow, nUp);                    // 获取进展条范围
    m_strPercent.Format("%4.0f%%",(float)nPos/(float)(nUp-nLow)*100.0);
    UpdateData(FALSE);
}
```

③ 用 MFC ClassWizard 为 CProgressDlg 类添加 WM_INITDIALOG 消息映射，并添加下列初始化代码：

```
BOOL CProgressDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_Progress.SetRange( 0, 100 );                     // 设置进展条范围
    m_Progress.SetStep(5);                             // 设置进展条步长
    m_Progress.SetPos(30);
```

```
    UpdatePercentText();
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

④ 用 MFC ClassWizard 为按钮 IDC_BUTTON_BACK 添加 BN_CLICKED 消息映射，并增加下列代码：

```
void CProgressDlg::OnButtonBack()
{
    int nPos = m_Progress.GetPos();            // 获取进展条当前位置
    int nLow, nUp;
    m_Progress.GetRange(nLow, nUp);            // 获取进展条范围
    nPos = nPos-5;
    if (nPos<nLow)
        nPos = nLow;
    m_Progress.SetPos( nPos );
    UpdatePercentText();
}
```

⑤ 用 MFC ClassWizard 为按钮 IDC_BUTTON_GOON 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CProgressDlg::OnButtonGoon()
{
    m_Progress.StepIt();
    UpdatePercentText();
}
```

（3）调用对话框

① 打开 Ex_Ctrl4SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在其下添加一个菜单项“进展条对话框(&S)”，ID 为 ID_TEST_PROGRESS。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_PROGRESS 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestProgress()
{
    CProgressDlg dlg;
    dlg.DoModal();
}
```

③ 在文件 MainFrm.cpp 的前面添加 CProgressDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "ProgressDlg.h"
```

④ 编译运行并测试。

3.6.2　滚动条

滚动条是一个独立的窗口，虽然它有直接的输入焦点，但却不能自动地滚动窗口内容，因此，它的使用受到一定的限制。

根据滚动条的走向，可分为垂直滚动条和水平滚动条两种类型。这两种类型滚动条的组成部分都是一样的，两端都有两个箭头按钮，中间有一个可沿滚动条方向移动的滚动块，如图 3.35 所示。

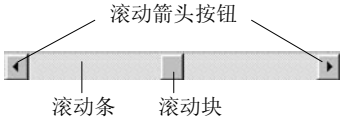

图 3.35　滚动条外观

1．滚动条的基本操作

滚动条的基本操作一般包括设置和获取滚动条的范围及滚动块的相应位置。

由于滚动条控件的默认滚动范围是 0 到 0，因此如果使用滚动条之前不设定其滚动范围，那么滚动条中的滚动块就滚动不起来。在 MFC 的 CScrollBar 类中，用函数 SetScrollRange 来设置滚动条的滚动范围，其原型如下：

```
SetScrollRange( int nMinPos, int nMaxPos, BOOL bRedraw = TRUE );
```

其中，nMinPos 和 nMaxPos 表示滚动位置的最小值和最大值。bRedraw 为重画标志，当为 TRUE 时，滚动条被重画。

在 CScrollBar 类中，设置滚动块位置操作是由 SetScrollPos 函数来完成的，其原型如下：

```
int SetScrollPos( int nPos, BOOL bRedraw = TRUE );
```

其中，nPos 表示滚动块的新位置，它必须是在滚动范围之内。

与 SetScrollRange 和 SetScrollPos 相对应的两个函数分别用来获取滚动条的当前范围以及当前滚动位置：

```
void GetScrollRange( LPINT lpMinPos, LPINT lpMaxPos ) ;
int GetScrollPos();
```

其中，LPINT 是整型指针类型，lpMinPos 和 lpMaxPos 分别用来返回滚动块最小和最大滚动位置。

2．WM_HSCROLL 或 WM_VSCROLL 消息

当对滚动条进行操作时，滚动条就会向父窗口发送 WM_HSCROLL 或 WM_VSCROLL 消息（分别对应于水平滚动条和垂直滚动条）。这些消息是通过 MFC ClassWizard 在其对话框（滚动条的父窗口）中进行映射的，并产生相应的消息映射函数 OnHScroll 和 OnVScroll，其原型如下：

```
afx_msg void OnHScroll( UINT nSBCode, UINT nPos, CScrollBar* pScrollBar );
afx_msg void OnVScroll( UINT nSBCode, UINT nPos, CScrollBar* pScrollBar );
```

其中，nPos 表示滚动块的当前位置，pScrollBar 表示由滚动条控件的指针，nSBCode 表示滚动条的通知消息。如图 3.36 表示，当鼠标单击滚动条的不同部位时，所产生的通知消息也不同。表 3.17 列出了各通知消息的含义。

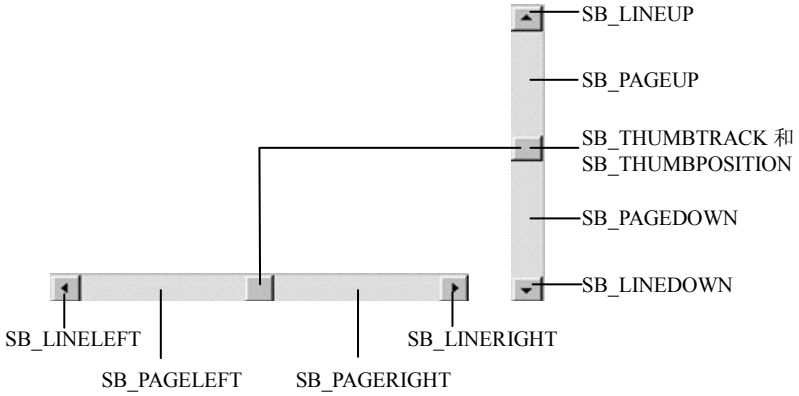

图 3.36　滚动条通知代码与位置的关系

表 3.17　滚动条的通知消息

通知消息	说明
SB_LEFT、SB_RIGHT	滚动到最左端或最右端时发送此消息
SB_TOP、SB_BOTTOM	滚动到最上端或最下端时发送此消息
SB_LINELEFT、SB_LINERIGHT	向左或右滚动一行(或一个单位)时发送此消息
SB_LINEUP、SB_LINEDOWN	向上或下滚动一行(或一个单位)时发送此消息
SB_PAGELEFT、SB_PAGERIGHT	向左或右滚动一页时发送此消息
SB_PAGEUP、SB_PAGEDOWN	向上或下滚动一页时发送此消息
SB_THUMBPOSITION	滚动到某绝对位置时发送此消息
SB_THUMBTRACK	拖动滚动块时发送此消息
SB_ENDSCROLL	滚动结束时发送此消息

3.6.3　滑动条

滑动条控件是由滑动块和可选的刻度线组成的。当用户用鼠标或方向键移动滑动块时，该控件发送通知消息来表明这些改变。

滑动条按照应用程序中指定的增量来移动。例如，如果用户指定此滑动条的范围为 5，则滑动块只能有 6 个位置，包括在滑动条控件最左边的一个位置和另外 5 个在此范围内每隔一个增量的位置。通常，这些位置都是由相应的刻度线来标识的，如图 3.37 所示。

1．滑动条的风格和消息

滑动条控件有许多风格，它们都可以通过滑动条控件的属性对话框进行设置，如图 3.38 所示。表 3.18 列出了该属性对话框的各项含义。

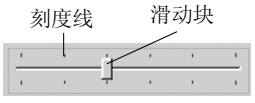

图 3.37　带刻度线的滑动条

图 3.38　滑动条属性对话框

表 3.18　滑动条控件的“样式”属性

项目	说明
方向（Orientation）	控件放置方向：Vertical（垂直）、Horizontal（水平，默认）
点（Point）	刻度线在滑动条控件中放置的位置：Both（两边都有）、Top/Left（水平滑动条的上边或垂直滑动条的左边，同时滑动块的尖头指向有刻度线的那一边）、Bottom/Right（水平滑动条的下边或垂直滑动条的右边，同时滑动块的尖头指向有刻度线的那一边）
打勾标记（Tick Marks）	选中此项，在滑动条控件上显示刻度线
自动打勾（Auto Ticks）	选中此项，滑动条控件上的每个增量位置处都有刻度线，并且增量大小自动根据其范围来确定
边框（Border）	选中此项，控件周围有边框
允许选择（Enable Selection）	选中此项，控件中供用户选择的数值范围高亮显示

滑动条的通知消息代码常见的有 TB_BOTTOM、TB_ENDTRACK、TB_LINEDOWN、TB_LINEUP、TB_PAGEDOWN、TB_PAGEUP、TB_THUMBPOSITION、TB_TOP 和 TB_THUMBTRACK 等。这些消息代码都来自于 WM_HSCROLL 或 WM_VSCROLL 消息，其具体含义同滚动条。

2．滑动条的基本操作

MFC 的 CSliderCtrl 类提供了滑动条控件的各种操作函数，这其中包括范围、位置设置和获取等。成员函数 SetPos 和 SetRange 分别用来设置滑动条的位置和范围，其原型如下：

```
void SetPos( int nPos );
void SetRange( int nMin, int nMax, BOOL bRedraw = FALSE );
```

其中，参数 nPos 表示新的滑动条位置。bMin 和 nMax 表示滑动条的最小和最大位置，bRedraw 表示重画标志，为 TRUE 时，滑动条被重画。与这两个函数相对应的成员函数 GetPos 和 GetRange 是分别用来获取滑动条的位置和范围的。

成员函数 SetTic 用来设置滑动条控件中的一个刻度线的位置。函数成功调用后返回非零值；否则返回 0。函数原型如下：

```
BOOL SetTic( int nTic );
```

其中，参数 nTic 表示刻度线的位置。

成员函数 SetTicFreq 用来设置显示在滑动条中的刻度线的疏密程度。其函数原型如下：

```
void SetTicFreq( int nFreq );
```

其中，参数 nFreq 表示刻度线的疏密程度。例如，如果参数被设置为 2，则在滑动条的范围中每两个增量显示一个刻度线。要使这个函数有效，必须在属性对话框中选中 Auto ticks 项。

成员函数 ClearTics 用来从滑动条控件中删除当前的刻度线。其函数原型如下：

```
void ClearTics( BOOL bRedraw = FALSE );
```

其中，参数 bRedraw 表示重画标志。若该参数为 TRUE，则在选择被清除后重画滑动条。

成员函数 SetSelection 用来设置一个滑动条控件中当前选择的开始和结束位置。其函数原型如下：

```
void SetSelection( int nMin, int nMax );
```

其中，参数 nMin、nMax 表示滑动条的开始和结束位置。

3.6.4 调整对话框背景颜色示例

设置对话框背景颜色有许多方法，这里采用最简单的也是最直接的方法，即通过映射 WM_CTLCOLOR（当子窗口将要绘制时发送的消息，以便能使用指定的颜色绘制控件）来达到改变背景颜色的目的。本例通过滚动条和两个滑动条来调整 Visual C++所使用的 RGB 颜色的三个分量：R（红色分量）、G（绿色分量）和 B（蓝色分量），如图 3.39 所示。

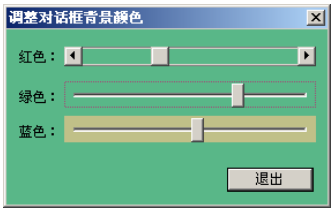

图 3.39 调整对话框背景颜色

1．添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl5SDI。

② 向应用程序中添加一个对话框资源 IDD_COLOR，标题定为“调整对话框背景颜色”，字体设为“宋体，9 号”，创建此对话框类为 CBkColorDlg。删除原来的 Cancel 按钮，将 OK 按钮标题改为“退出”。

③ 打开对话框网格，参照图 3.40 所示的控件布局，为对话框添加如表 3.19 所示的一些控件。

表 3.19 对话框添加的控件

添加的控件	ID 号	标题	其他属性
水平滚动条	IDC_SCROLLBAR_RED	—	默认
滑动条(绿色)	IDC_SLIDER_GREEN	—	默认
滑动条(蓝色)	IDC_SLIDER_BLUE	—	默认

2．完善 CBkColorDlg 类代码

① 打开 ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CBkColorDlg，选中所需的控件 ID 号，双击鼠标。依次为下列控件增加成员变量，如表 3.20 所示。

表 3.20 控件变量

控件 ID 号	变量类别	变量类型	变量名	范围和大小
IDC_SCROLLBAR_RED	Control	CScrollBar	m_scrollRed	—
IDC_SLIDER_GREEN	Control	CSliderCtrl	m_sliderGreen	—
IDC_SLIDER_GREEN	Value	int	m_nGreen	
IDC_SLIDER_BLUE	Control	CSliderCtrl	m_sliderBlue	—
IDC_SLIDER_BLUE	Value	int	m_nBlue	—

② 为 CBkColorDlg 类添加两个成员变量，一个是 int 型 m_nRedValue，用来设置颜色 RGB 中的红色分量；另一个是画刷 CBrush 类对象 m_Brush，用来设置对话框背景所需要的画刷。

③ 用 MFC ClassWizard 为 CBkColorDlg 类添加 WM_INITDIALOG 消息映射，并添加下列初始化代码：

```
BOOL CBkColorDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_scrollRed.SetScrollRange(0, 255);
    m_sliderBlue.SetRange(0, 255);
    m_sliderGreen.SetRange(0, 255);
    m_nBlue = m_nGreen = m_nRedValue = 192;
    UpdateData( FALSE );
    m_scrollRed.SetScrollPos(m_nRedValue);
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

④ 用 MFC ClassWizard 为 CBkColorDlg 类添加 WM_HSCROLL 消息映射，并添加下列代码：

```
void CBkColorDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    int nID = pScrollBar->GetDlgCtrlID();          // 获取对话框中控件ID号
    if (nID == IDC_SCROLLBAR_RED)      // 或是滚动条产生的水平滚动消息
    {
        switch(nSBCode) {
            case SB_LINELEFT:   m_nRedValue--;  // 单击滚动条左边箭头
                                break;
            case SB_LINERIGHT:  m_nRedValue++;  // 单击滚动条右边箭头
                                break;
            case SB_PAGELEFT:   m_nRedValue -= 10;
                                break;
            case SB_PAGERIGHT:  m_nRedValue += 10;
                                break;
            case SB_THUMBTRACK: m_nRedValue = nPos;
                                break;
        }
        if (m_nRedValue<0) m_nRedValue = 0;
        if (m_nRedValue>255) m_nRedValue = 255;
        m_scrollRed.SetScrollPos(m_nRedValue);
    }
    Invalidate();                       // 使对话框无效，强迫系统重绘对话框
    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

⑤ 用 MFC ClassWizard 为 CBkColorDlg 类添加 WM_CTLCOLOR 消息映射，并添加下列代码：

```
HBRUSH CBkColorDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    UpdateData(TRUE);
    COLORREF color = RGB(m_nRedValue, m_nGreen, m_nBlue);
    m_Brush.Detach();                    // 使画刷和对象分离
    m_Brush.CreateSolidBrush(color);     // 创建颜色画刷
    pDC->SetBkColor( color );            // 设置背景颜色
    return (HBRUSH)m_Brush;          // 返回画刷句柄，以便系统使此画刷绘制对话框
}
```

其中，COLORREF 是用来表示 RGB 颜色的一个 32 位的数据类型，它是 Visual C++中一种专门用来定义颜色的数据类型。画刷的详细用法以后还会讨论。

3．调用对话框

① 打开 Ex_Ctrl5SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在其下添加一个菜单项“调整对话框背景颜色(&O)”，ID 为 ID_TEST_COLOR。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_COLOR 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestColor()
{
     CBkColorDlg dlg;
     dlg.DoModal();
}
```

③ 在文件 MainFrm.cpp 的前面添加 CBkColorDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "BkColorDlg.h"
```

④ 编译运行并测试。

说明：

- 由于滚动条和滑动条等许多控件都能产生 WM_HSCROLL 或 WM_VSCROLL 消息，因此当它们处在同一方向（水平或垂直）时，就需要添加相应代码判断消息是谁产生的。
- 由于滚动条中间的滚动块在默认时是不会停止在用户操作的位置处的，因此需要调用 SetScrollPos 函数来进行相应位置的设定。

3.7 日期时间控件

在日期时间控件中，用户可调整显示的日期。默认时，用户可单击控件的右边的下拉

按钮，即可弹出日期控件可供用户选择日期，通过风格的改变还可在 DTP 控件内显示时间。图 3.40 所示是单击下拉按钮后显示的结果。

1．日期时间控件的风格和操作

日期时间有许多风格，这些风格用来定义日期时间控件的外观及操作方式，它们可以在日期时间控件属性对话框中进行设置，如图 3.41 所示。表 3.21 列出该属性对话框的各项含义。

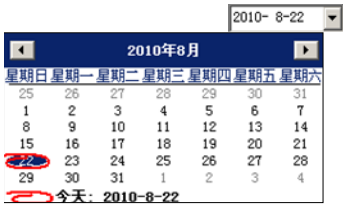

图 3.40　日期时间控件

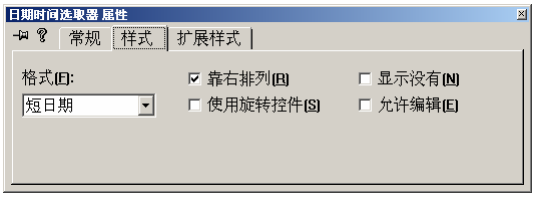

图 3.41　“日期时间选取器属性”对话框

一般来说，用户最关心的是如何设置和获取日期时间控件的日期或时间。CDateTimeCtrl 类的成员函数 SetTime 和 GetTime 可以满足这样的要求，它们最常用的函数原型如下：

```
BOOL SetTime( const CTime* pTimeNew );
DWORD GetTime( CTime& timeDest ) const;
```

其中，CTime 是 Visual C++用于时间操作的类。

表 3.21　日期时间控件的“样式”属性

项目	说明
格式（Format）	日期时间控件的格式有：Short Date（短格式）、Long Date（长格式）、Time（显示时间）
靠右排列（Right Align）	下拉月历右对齐控件
使用旋转控件（Use Spin Control）	选中此项，在控件的右边出现一个旋转按钮用来调整日期。否则，控件的右边是一个下拉按钮用来弹出月历
显示没有（Show None）	选中此项，日期前面显示一个复选框，当用户选中复选框时，方可输入或选择一个日期
允许编辑（Allow Edit）	选中此项，日期和时间允许编辑

2．创建并使用学生基本信息对话框示例

在学生信息管理系统中，往往需要设计一个学生基本信息对话框来添加和修改学生基本信息，如图 3.42 所示。下面就来看看这个示例。

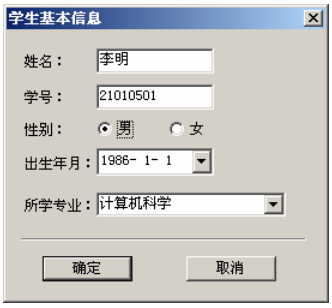

图 3.42　“学生基本信息”对话框

（1）添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl6SDI。

② 向应用程序中添加一个对话框资源 IDD_STUINFO，标题定为“学生基本信息”，字体设为“宋体，

9 号”，创建此对话框类为 CStuInfoDlg。

③ 将 OK 和 Cancel 按钮的标题改为“确定”和“取消”。

④ 打开对话框网格，参照图 3.42 所示的控件布局，为对话框添加如表 3.22 所示的一些控件。

表 3.22 “学生基本信息”对话框添加的控件

添加的控件	ID 号	标题	其他属性
编辑框(姓名)	IDC_EDIT_NAME	—	默认
编辑框(学号)	IDC_EDIT_NO	—	默认
单选按钮(男)	IDC_RADIO_MALE	男	默认
单选按钮(女)	IDC_RADIO_FEMALE	女	默认
日期时间控件(出生年月)	IDC_DATETIMEPICKER1	—	默认
组合框(专业)	IDC_COMBO_SPECIAL	—	默认

（2）完善 CStuInfoDlg 类代码

① 打开 ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CStuInfoDlg，选中所需的控件 ID 号，双击鼠标。依次为下列控件增加成员变量，如表 3.23 所示。

表 3.23 控件变量

控件 ID 号	变量类别	变量类型	变量名	范围和大小
IDC_EDIT_NAME	Value	CString	m_strName	–10
IDC_EDIT_NO	Value	CString	m_strNo	10
IDC_DATETIMEPICKER1	Value	CTime	m_tBirth	—
IDC_COMBO_SPECIAL	Control	CComboBox	m_comboSpecial	—
IDC_COMBO_SPECIAL	Value	CString	m_strSpecial	40

② 为 CStuInfoDlg 类添加一个 BOOL 型成员变量 m_bMale，并在 CStuInfoDlg 类设置该变量的初始值。如下面的代码：

```
CStuInfoDlg::CStuInfoDlg(CWnd* pParent /*=NULL*/)
            : CDialog(CStuInfoDlg::IDD, pParent)
{
      m_bMale = FALSE;
      //{{AFX_DATA_INIT(CStuInfoDlg)
      …
      //}}AFX_DATA_INIT
}
```

其中，“//{{AFX_DATA_INIT(CStuInfoDlg)”和“//}}AFX_DATA_INIT”之间的代码是控件变量的初始化代码，并由 MFC ClassWizard 自动管理。

③ 用 MFC ClassWizard 为 CStuInfoDlg 类添加 WM_INITDIALOG 消息映射，并添加下列初始化代码：

```
BOOL CStuInfoDlg::OnInitDialog()
{
```

```
    CDialog::OnInitDialog();
    // 设置单选按钮初始选中状态
    if (!m_bMale)
        CheckRadioButton(IDC_RADIO_MALE,IDC_RADIO_FEMALE,IDC_RADIO_FEMALE);
    else
        CheckRadioButton(IDC_RADIO_MALE,IDC_RADIO_FEMALE,IDC_RADIO_MALE);
    // 这里对专业组合框进行初始化
    m_comboSpecial.AddString( "机械工程及其自动化" );
    m_comboSpecial.AddString( "电气工程及其自动化" );
    m_comboSpecial.AddString( "计算机科学" );
    m_strSpecial = "计算机科学";
    m_tBirth = CTime(1986, 1, 1, 0, 0, 0);  // 对出生年月初始化
    UpdateData(FALSE);
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

④ 用 MFC ClassWizard 为单选按钮 IDC_RADIO_MALE 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CStuInfoDlg::OnRadioMale()
{
    m_bMale = TRUE;
}
```

⑤ 用 MFC ClassWizard 为单选按钮 IDC_RADIO_FEMALE 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CStuInfoDlg::OnRadioFemale()
{
    m_bMale = FALSE;
}
```

⑥ 用 MFC ClassWizard 为按钮 IDOK 添加 BN_CLICKED 的消息映射，并增加下列代码：

```
void CStuInfoDlg::OnOK()
{
    UpdateData();
    m_strName.TrimLeft();
    m_strNo.TrimLeft();
    if (m_strName.IsEmpty())
        MessageBox("必须要有姓名！");
    else if (m_strNo.IsEmpty())
        MessageBox("必须要有学号！");
    else
        CDialog::OnOK();
}
```

（3）调用对话框

① 打开 Ex_Ctrl6SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在其下添加一个菜单项“学生基本信息(&U)”，ID 为 ID_TEST_STUINFO。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_STUINFO 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestStuinfo()
{
     CStuInfoDlg dlg;
     if (IDOK == dlg.DoModal()) {
          CString strRes, strSex("女");
          if (dlg.m_bMale) strSex = "男";
          strRes.Format("姓名：%s，学号：%s，性别：%s，出生年月：%s，专业：%s",
              dlg.m_strName, dlg.m_strNo, strSex,
              dlg.m_tBirth.Format("%Y-%m-%d"), dlg.m_strSpecial);
          AfxMessageBox( strRes );
     }
}
```

其中，m_tBirth.Format 中的 Format 是 CTime 类的成员函数，用来将时间转换成字符串。%Y 表示四位数“年”，%m 表示两位数月份，%d 表示两位数日期。

③ 在文件 MainFrm.cpp 的前面添加 CStuInfoDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "StuInfoDlg.h"
```

④ 编译运行并测试。

3.8 列表控件和树控件

当每项内容包含多组信息时，就需要用列表控件来呈现。若项目之间还存在层次关系，则用树控件来表现最为合适。无论是列表控件还是树控件，它们还可与“图像列表”相关联，为各项目指定不同的图标或位图。

3.8.1 图像列表控件

图像列表控件常常用来有效地管理多个位图和图标，它是一系列相同大小的图像的集合，每一个图像均提供一个以 0 为基数的索引号。在 MFC 中，图像列表控件是使用 CImageList 类来创建、显示和管理图像的。

1．图像列表的创建

图像列表的创建不像其他控件，它不能通过对话框编辑器来创建。因此，创建一个图像列表首先要声明一个 CImageList 对象，然后调用 Create 函数。由于 Create 函数的重载很多，故这里给出最常用的一个原型：

```
BOOL Create( int cx, int cy, UINT nFlags, int nInitial, int nGrow );
```

其中，cx 和 cy 用来指定图像的像素大小；nFlags 表示要创建的图像类型，一般取其 ILC_COLOR 和 ILC_MASK(指定屏蔽图像)的组合，默认的 ILC_COLOR 为 ILC_COLOR4(16 色)，当然也可以是 ILC_COLOR8(256 色)、ILC_COLOR16(16 位色)等；nInitial 用来指定图像列表中最初的图像数目；nGrow 表示当图像列表的大小发生改变时图像可以增加的数目。

2．图像列表的基本操作

常见的图像列表的基本操作有增加、删除和绘制等，其相关成员函数如下：

```
int Add( CBitmap* pbmImage, CBitmap* pbmMask );
int Add( CBitmap* pbmImage, COLORREF crMask );
int Add( HICON hIcon );
```

此函数用来向一个图像列表添加一个图标或多个位图。成功时返回第一个新图像的索引号，否则返回–1。参数 pbmImage 表示包含图像的位图指针，pbmMask 表示包含屏蔽的位图指针，crMask 表示屏蔽色，hIcon 表示图标句柄。

```
BOOL Remove( int nImage );
```

该函数用来从图像列表中删除一个由 nImage 指定的图像，成功时返回非 0，否则返回 0。

```
BOOL Draw( CDC* pdc, int nImage, POINT pt, UINT nStyle );
```

该函数用来在由 pt 指定的位置绘制一个图像。参数 pdc 表示绘制的设备环境指针，nImage 表示要绘制的图像的索引号，nStyle 用来指定绘制图像时采用的方式。

```
HICON ExtractIcon( int nImage );
```

该函数用来将 nImage 指定的图像扩展为图标。

```
COLORREF SetBkColor( COLORREF cr );
```

该函数用来设置图像列表的背景色，它可以是 CLR_NONE。成功时返回先前的背景色，否则为 CLR_NONE。

3.8.2　列表控件

列表控件是一种极为有用的控件之一，它可以用 “大图标”、“小图标”、“列表视图”或“报表视图”4 种不同的方式来显示一组信息，如图 3.43 所示。

所谓大图标方式，是指列表中的所有项的上方均以大图标（32×32）形式出现，用户可将其拖动到列表视图窗口的任意位置。小图标方式是指列表中的所有项的左方均以小图标（16×16）形式出现，用户可将其拖动到列表视图窗口的任意位置。列表视图方式与图标方式不同，列表项被安排在某一列中，用户不能拖动它们。报表视图方式是指列表项出

现在各自的行上，而相关的信息出现在右边，最左边的列可以是标签或图标，接下来的列则是程序指定的列表项内容。报表视图方式中最引人注目的是它可以有标题头。

图 3.43　列表控件样式

1．列表控件的样式及其修改

列表控件的样式有两类，一类是一般样式，如表 3.24 所示。

表 3.24　列表控件的一般风格

风格	含义
LVS_ALIGNLEFT	在“大图标”或“小图标”显示方式中，所有列表项左对齐
LVS_ALIGNTOP	在“大图标”或“小图标”显示方式中，所有列表项被安排在控件的顶部
LVS_AUTOARRANGE	在“大图标”或“小图标”显示方式中，图标自动排列
LVS_EDITLABELS	允许用户编辑项目文本，但父窗口必须处理 LVN_ENDLABELEDIT 通知消息
LVS_ICON	“大图标”显示方式
LVS_LIST	“列表视图”显示方式
LVS_NOCOLUMNHEADER	在“报表视图”显示方式中，不显示其标题头
LVS_NOLABELWRAP	在“大图标”显示方式中，项目文本占满一行
LVS_NOSCROLL	禁用滚动条
LVS_NOSORTHEADER	当用户单击标题头时，不产生任何操作
LVS_OWNERDRAWFIXED	指明控件的拥有者，而不是 Windows 负责绘制该控件
LVS_REPORT	“报表视图”显示方式
LVS_SHAREIMAGELISTS	共享图像列表
LVS_SHOWSELALWAYS	一直显示被选择的部分
LVS_SINGLESEL	只允许单项选择，默认时是多项选择
LVS_SMALLICON	“小图标”显示方式
LVS_SORTASCENDING	按升序排列
LVS_SORTDESCENDING	按降序排列

另一类是 Visual C++在原有的基础上添加的扩展样式，如 LVS_EX_FULLROWSELECT，表示整行选择，但它仅用于“报表视图”显示方式中。类似的常用的还有：

LVS_EX_BORDERSELECT	用边框选择方式代替高亮显示列表项
LVS_EX_GRIDLINES	列表项各行显示线条（仅用于“报表视图”）

对于列表控件的一般风格的修改，可先调用 GetWindowLong 来获取当前风格，然后调用 SetWindowLong 重新设置新的风格。对于列表控件的扩展风格，可直接调用成员函数 CListCtrl::SetExtendedStyle 加以设置。

2．列表项的基本操作

列表控件类 CListCtrl 提供了许多用于列表项操作的成员函数，如列表项与列的添加和删除等，下面分别介绍。

① 函数 SetImageList 用来为列表控件设置一个关联的图像列表，其原型如下：

```
CImageList* SetImageList( CImageList* pImageList, int nImageList );
```

其中，nImageList 用来指定图像列表的类型，它可以是 LVSIL_NORMAL（大图标）、LVSIL_SMALL（小图标）和 LVSIL_STATE（表示状态的图像列表）。

② 函数 InsertItem 用来向列表控件中插入一个列表项。该函数成功时返回新列表项的索引号，否则返回–1。函数原型如下：

```
int InsertItem( const LVITEM* pItem );
int InsertItem( int nItem, LPCTSTR lpszItem );
int InsertItem( int nItem, LPCTSTR lpszItem, int nImage );
```

其中，nItem 用来指定要插入的列表项的索引号，lpszItem 表示列表项的文本标签，nImage 表示列表项图标在图像列表中的索引号；而 pItem 用来指定一个指向 LVITEM 结构的指针，其结构描述如下：

```
typedef struct _LVITEM
{
    UINT     mask;           // 指明哪些参数有效
    int      iItem;          // 列表项索引
    int      iSubItem;       // 子项索引
    UINT     state;          // 列表项状态
    UINT     stateMask;      // 指明state哪些位是有效的，-1全部有效
    LPTSTR   pszText;        // 列表项文本标签
    int      cchTextMax;     // 文本大小
    int      iImage;         // 在图像列表中列表项图标的索引号
    LPARAM   lParam;         // 32位值
    int      iIndent;        // 项目缩进数量，1个数量等于1个图标的像素宽度
} LVITEM, FAR *LPLVITEM;
```

其中，mask 最常用的值可以是：

```
LVIF_TEXT           pszText有效或必须赋值
LVIF_IMAGE          iImage有效或必须赋值
LVIF_INDENT         iIndent有效或必须赋值
```

③ 函数 DeleteItem 和 DeleteAllItems 分别用来删除指定的列表项和全部列表项，函数原型如下：

```
BOOL DeleteItem( int nItem );
BOOL DeleteAllItems( );
```

④ 函数 FindItem 用来查寻列表项，函数成功查找时返回列表项的索引号，否则返回–1。其原型如下：

```
int FindItem( LVFINDINFO* pFindInfo, int nStart = -1 ) const;
```

其中，nStart 表示开始查找的索引号，–1 表示从头开始。pFindInfo 表示要查找的信息，其结构描述如下：

```
typedef struct tagLVFINDINFO
{
    UINT        flags;          // 查找方式
    LPCTSTR     psz;            // 匹配的文本
    LPARAM      lParam;         // 匹配的值
    POINT       pt;             // 查找开始的位置坐标
    UINT        vkDirection;    // 查找方向，用虚拟方向键值表示
} LVFINDINFO, FAR* LPFINDINFO;
```

其中，flags 可以是下列值之一或组合：

```
LVFI_PARAM      查找内容由lParam指定
LVFI_PARTIAL    查找内容由psz指定，不精确查找
LVFI_STRING     查找内容由psz指定，精确查找
LVFI_WRAP       若没有匹配，再从头开始
LVFI_NEARESTXY  靠近pt位置查找，查找方向由vkDirection 确定
```

⑤ 函数 Arrange 用来按指定方式重新排列列表项，其原型如下：

```
BOOL Arrange( UINT nCode );
```

其中，nCode 用来指定排列方式，它可以是下列值之一：

```
LVA_ALIGNLEFT   左对齐
LVA_ALIGNTOP    上对齐
LVA_DEFAULT     默认方式
LVA_SNAPTOGRID  使所有的图标安排在最接近的网格位置
```

⑥ 函数 InsertColumn 用来向列表控件插入新的一列，函数成功调用后返回新的列的索引，否则返回–1。其原型如下：

```
int InsertColumn( int nCol, const LVCOLUMN* pColumn );
int InsertColumn( int nCol, LPCTSTR lpszColumnHeading, int nFormat =
LVCFMT_LEFT, int nWidth = -1, int nSubItem = -1 );
```

其中，nCol 用来指定新列的索引，lpszColumnHeading 用来指定列的标题文本，nFormat 用来指定列排列的方式，它可以是 LVCFMT_LEFT（左对齐）、LVCFMT_RIGHT（右对齐）和 LVCFMT_CENTER（居中对齐）；nWidth 用来指定列的像素宽度，–1 时表示宽度没有

设置；nSubItem 表示与列相关的子项索引，其为–1 时表示没有子项。pColumn 表示包含新列信息的 LVCOLUMN 结构地址，其结构描述如下：

```
typedef struct _LVCOLUMN {
    UINT     mask;       // 指明哪些参数有效
    int      fmt;        // 列的标题或子项文本格式
    int      cx;         // 列的像素宽度
    LPTSTR   pszText;    // 列的标题文本
    int      cchTextMax; // 列的标题文本大小
    int      iSubItem;   // 和列相关的子项索引
    int      iImage;     // 图像列表中的图像索引
    int      iOrder;     // 列的序号，最左边的列为0
} LVCOLUMN, FAR *LPLVCOLUMN;
```

其中，mask 可以是 0 或下列值之一或组合：

```
LVCF_FMT                fmt参数有效
LVCF_IMAGE              iImage参数有效
LVCF_ORDER              iOrder参数有效
LVCF_SUBITEM            iSubItem参数有效
LVCF_TEXT               pszText参数有效
LVCF_WIDTH              cx参数有效
```

fmt 可以是下列值之一：

```
LVCFMT_BITMAP_ON_RIGHT  位图出现在文本的右边，对于从图像列表中选取的图像无效
LVCFMT_CENTER           文本居中
LVCFMT_COL_HAS_IMAGES   列表头的图像是在图像列表中
LVCFMT_IMAGE            从图像列表中显示一个图像
LVCFMT_LEFT             文本左对齐
LVCFMT_RIGHT            文本右对齐
```

⑦ 函数 DeleteColumn 用来从列表控件中删除一个指定的列，其原型如下：

```
BOOL DeleteColumn( int nCol );
```

除了上述操作外，还有一些函数是用来设置或获取列表控件的相关属性的。例如 SetColumnWidth 用来设置指定列的像素宽度，GetItemCount 用来返回列表控件中的列表项个数等。它们的原型如下：

```
BOOL SetColumnWidth( int nCol, int cx );
int GetItemCount( );
```

其中，nCol 用来指定要设置的列的索引号，cx 用来指定列的像素宽度，它可以是 LVSCW_AUTOSIZE，表示自动调整宽度。

3．列表控件的消息

在列表视图中，可以用 MFC ClassWizard 映射的控件消息有公共控件消息（如

NM_DBLCLK）、标题头控件消息以及列表控件消息。常用的列表控件消息有：

LVN_BEGINDRAG	用户按住鼠标左键拖动列表项
LVN_BEGINLABELEDIT	用户对某列表项标签进行编辑
LVN_COLUMNCLICK	某列表项被单击
LVN_ENDLABELEDIT	用户对某列表项标签结束编辑
LVN_ITEMACTIVATE	用户激活某列表项
LVN_ITEMCHANGED	当前列表项已被改变
LVN_ITEMCHANGING	当前列表项即将改变
LVN_KEYDOWN	某键被按下

需要说明的是，在用 ClassWizard 处理上述这些消息时，其消息处理函数参数中往往会出现 NM_LISTVIEW 结构，其定义如下：

```
typedef struct tagNMLISTVIEW
{
    NMHDR       hdr;            // 包含通知消息的结构
    int         iItem;          // 列表项索引，没有为-1
    int         iSubItem;       // 子项索引，没有为0
    UINT        uNewState;      // 新的项目状态
    UINT        uOldState;      // 原来的项目状态
    UINT        uChanged;       // 项目属性更改标志
    POINT       ptAction;       // 事件发生的地点
    LPARAM      lParam;         // 用户定义的32位值
} NMLISTVIEW, FAR *LPNMLISTVIEW;
```

但对于 LVN_ITEMACTIVATE 来说，上述结构变成了 NMITEMACTIVATE，它在结构 NM_LISTVIEW 基础上增加了一个成员 UINT uKeyFlags，用来表示 Alt、Ctrl 和 Shift 键的按下状态，它的值可以是 LVKF_ALT、LVKF_CONTROL 和 LVKF_SHIFT。

4．学生基本信息的添加和显示示例

本例将在对话框中添加一个列表控件，用来显示学生基本信息，如图 3.44 所示。单击“添加”按钮，将弹出前面示例中创建的“学生基本信息”对话框，添加的信息出现在列表控件中，在添加之前仍需进行重复性判断，单击最上面的一组单选按钮，可将列表控件按不同方式显示列表信息。

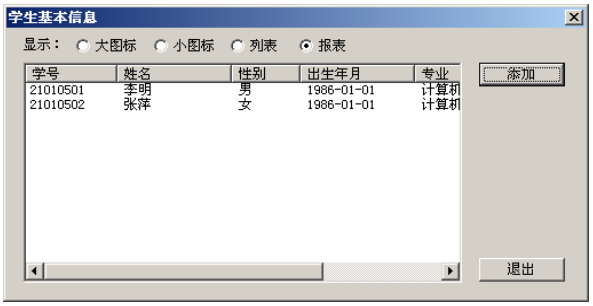

图 3.44　列表控件示例运行结果

5．添加并设计对话框

① 打开单文档应用程序项目 Ex_Ctrl6SDI。

② 向应用程序中添加一个对话框资源 IDD_LIST，标题定为“学生基本信息”，字体设为“宋体，9 号”，创建此对话框类为 CListDlg。

③ 删除 Cancel 按钮，将 OK 的标题改为“退出”。打开对话框网格，参照图 3.45 所示的控件布局，为对话框添加如表 3.25 所示的一些控件。

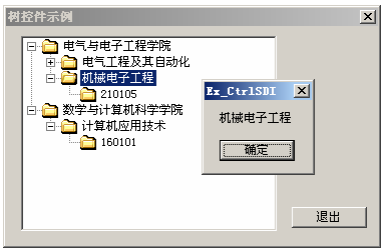

图 3.45　树控件示例运行结果

表 3.25　列表示例控件及属性

添加的控件	ID 号	标题	其他属性
单选按钮(姓名)	IDC_RADIO_MALE	大图标	默认
单选按钮(学号)	IDC_RADIO_MALE	小图标	默认
单选按钮(男)	IDC_RADIO_MALE	列表	默认
单选按钮(女)	IDC_RADIO_FEMALE	报表	默认
列表控件	IDC_LIST1	—	默认
按钮控件	IDC_BUTTON_ADD	添加	默认

6．完善 CListDlg 类代码

① 打开 ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CListDlg，选中 IDC_LIST1，双击鼠标或单击 Add Variables 按钮，为其添加一个控件变量 m_ListCtrl，类型为 CListCtrl。

② 为 CListDlg 添加成员函数 SetCtrlStyle(HWND hWnd, DWORD dwNewStyle)，用来设置列表控件的不同显示方式，其代码如下：

```
void CListDlg::SetCtrlStyle(HWND hWnd, DWORD dwNewStyle)
{
    DWORD   dwOldStyle;
    dwOldStyle = GetWindowLong(hWnd, GWL_STYLE);        // 获取当前风格
    if ((dwOldStyle&LVS_TYPEMASK) != dwNewStyle)  {
        dwOldStyle &= ~LVS_TYPEMASK;
        dwNewStyle |= dwOldStyle;
        SetWindowLong(hWnd, GWL_STYLE, dwNewStyle);   // 设置新风格
    }
}
```

代码中，HWND 是窗口句柄类型，LVS_TYPEMASK 用来表示指定风格是列表视图类型风格，即只有 LVS_ICON、LVS_LIST、LVS_REPORT 或 LVS_SMALLICON 风格有效。

③ 用 MFC ClassWizard 为 CListDlg 类添加 WM_INITDIALOG 消息映射，并添加下列初始化代码：

```
BOOL CListDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    // 设置单选按钮组初始选中状态
```

```
        CheckRadioButton(IDC_RADIO_LARGE, IDC_RADIO_REPORT, IDC_RADIO_REPORT);
        // 将列表控件设置为“报表”显示方式
        SetCtrlStyle( m_ListCtrl.m_hWnd, LVS_REPORT );
        // 创建列表控件的标题头
        CString strHeader[5]={ "学号", "姓名", "性别", "出生年月", "专业"};
        int nWidth[5] = {80, 100, 60, 100, 200};
        for (int nCol=0; nCol<5; nCol++)
              m_ListCtrl.InsertColumn(nCol,strHeader[nCol],LVCFMT_LEFT,
              nWidth[nCol]);
        return TRUE;  // return TRUE unless you set the focus to a control
    }
```

④ 用 MFC ClassWizard 依次为单选按钮 IDC_RADIO_LARGE、IDC_RADIO_SMALL、IDC_RADIO_LIST、IDC_RADIO_REPORT 添加 BN_CLICKED 消息映射，并增加下列代码：

```
    void CListDlg::OnRadioLarge()
    {
        SetCtrlStyle( m_ListCtrl.m_hWnd, LVS_ICON );
    }
    void CListDlg::OnRadioSmall()
    {
        SetCtrlStyle( m_ListCtrl.m_hWnd, LVS_SMALLICON );
    }
    void CListDlg::OnRadioList()
    {
        SetCtrlStyle( m_ListCtrl.m_hWnd, LVS_LIST );
    }
    void CListDlg::OnRadioReport()
    {
        SetCtrlStyle( m_ListCtrl.m_hWnd, LVS_REPORT );
    }
```

⑤ 用 MFC ClassWizard 为按钮 IDC_BUTTON_ADD 添加 BN_CLICKED 消息映射，并增加下列代码：

```
    void CListDlg::OnButtonAdd()
    {
        CStuInfoDlg dlg;
        if (IDOK != dlg.DoModal()) return;
        // 根据学号来判断学生基本信息是不是已经添加过
        LVFINDINFO info;
        info.flags = LVFI_PARTIAL|LVFI_STRING;
        info.psz = dlg.m_strNo;
        if (m_ListCtrl.FindItem( &info ) != -1) // 若找到
        {
              CString str;
              str.Format("学号为%s的学生基本信息已添加过！", dlg.m_strNo);
```

```
            MessageBox(str);      return;
        }
        // 添加学生基本信息
        int nIndex = m_ListCtrl.InsertItem(m_ListCtrl.GetItemCount(), dlg.m_
        strNo);
        m_ListCtrl.SetItemText( nIndex, 1, dlg.m_strName );
        if (dlg.m_bMale)
            m_ListCtrl.SetItemText( nIndex, 2, "男" );
        else
            m_ListCtrl.SetItemText( nIndex, 2, "女" );
        m_ListCtrl.SetItemText( nIndex, 3, dlg.m_tBirth.Format("%Y-%m-%d"));
        m_ListCtrl.SetItemText( nIndex, 4, dlg.m_strSpecial);
    }
```

⑥ 在文件 ListDlg.cpp 的前面添加 CStuInfoDlg 类的头文件包含：

```
#include "Ex_Ctrl6SDI.h"
#include "ListDlg.h"
#include "StuInfoDlg.h"
```

7．调用对话框

① 打开 Ex_Ctrl6SDI 单文档应用程序的菜单资源，在顶层菜单项“测试(&T)”中再添加一个菜单项“列表控件(&L)”，ID 为 ID_TEST_LIST。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_LIST 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestList()
{
    CListDlg dlg;
    dlg.DoModal();
}
```

③ 在文件 MainFrm.cpp 的前面添加 CListDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "ListDlg.h"
```

④ 编译运行并测试。

3.8.3　树控件

与列表控件不同的是，在树控件的初始状态下只显示少量的顶层信息，这样有利于用户决定树的哪一部分需要展开，且可看到结点之间的层次关系。每一个结点都可由一个文本和一个可选的位图图像组成，单击结点可展开或收缩该结点下的子结点。

树控件由父结点和子结点组成。位于某一结点之下的结点称为子结点，位于子结点之

上的结点称为该结点的父结点。位于树的顶层或根部的结点称为根结点。

1．树控件的风格

常见的树控件风格如表 3.26 所示，其修改方法与列表控件的一般风格修改方法相同。

表 3.26　树控件的一般风格（样式）

风格	含义
TVS_HASLINES	子结点与它们的父结点之间用线连接
TVS_LINESATROOT	用线连接子结点和根结点
TVS_HASBUTTONS	在每一个父结点的左边添加一个按钮“＋”和“－”
TVS_EDITLABELS	允许用户编辑结点的标签文本内容
TVS_SHOWSELALWAYS	当控件失去焦点时，被选择的结点仍然保持被选择
TVS_DISABLEDRAGDROP	该控件被禁止发送 TVN_BEGINDRAG 通知消息
TVS_NOTOOLTIPS	控件禁用工具提示
TVS_SINGLEEXPAND	当使用这个风格时，结点可展开收缩
TVS_CHECKBOXES	在每一结点的最左边有一个复选框
TVS_FULLROWSELECT	多行选择，不能用于 TVS_HASLINES 风格
TVS_INFOTIP	控件得到工具提示时发送 TVN_GETINFOTIP 通知消息
TVS_NONEVENHEIGHT	结点的高度值不一样。默认结点高度是一样的
TVS_NOSCROLL	不使用水平或垂直滚动条
TVS_TRACKSELECT	使用热点跟踪

2．树控件的常用操作

MFC 树控件类 CTreeCtrl 类提供了许多关于树控件操作的成员函数，如结点的添加和删除等。下面分别说明。

① 函数 InsertItem 用来向树控件插入一个新结点，操作成功后，函数返回新结点的句柄，否则返回 NULL。函数原型如下：

```
HTREEITEM InsertItem( UINT nMask, LPCTSTR lpszItem,int nImage, int
nSelectedImage, UINT nState, UINT nStateMask, LPARAM lParam,
    HTREEITEM hParent, HTREEITEM hInsertAfter );
HTREEITEM InsertItem( LPCTSTR lpszItem, HTREEITEM hParent = TVI_ROOT,
    HTREEITEM hInsertAfter = TVI_LAST );
HTREEITEM InsertItem( LPCTSTR lpszItem, int nImage, int nSelectedImage,
    HTREEITEM hParent = TVI_ROOT, HTREEITEM hInsertAfter = TVI_LAST );
```

其中，nMask 用来指定要设置的属性，lpszItem 用来指定结点的文本标签内容，nImage 用来指定该结点图标在图像列表中的索引号，nSelectedImage 表示该结点被选定时，其图标图像列表中的索引号，nState 表示该结点的当前状态，它可以是 TVIS_BOLD（加粗）、TVIS_EXPANDED（展开）和 TVIS_SELECTED（选中）等，nStateMask 用来指定哪些状态参数有效或必须设置，lParam 表示与该结点关联的一个 32 位值，hParent 用来指定要插入结点的父结点的句柄，hInsertAfter 用来指定新结点添加的位置，它可以是：

```
TVI_FIRST          插到开始位置
```

TVI_LAST	插到最后
TVI_SORT	插入后按字母重新排序

② 函数 DeleteItem 和 DeleteAllItems 分别用来删除指定的结点和全部的结点。它们的原型如下：

```
BOOL DeleteAllItems( );
BOOL DeleteItem( HTREEITEM hItem );
```

其中，hItem 用来指定要删除的结点的句柄。如果 hItem 的值是 TVI_ROOT，则所有的结点都被从此控件中删除。

③ 函数 Expand 用来展开或收缩指定父结点的所有子结点，其原型如下：

```
BOOL Expand( HTREEETEM hItem, UINT nCode );
```

其中，hItem 指定要被展开或收缩的结点的句柄，nCode 用来指定动作标志，它可以是：

TVE_COLLAPSE	收缩所有子结点
TVE_COLLAPSERESET	收缩并删除所有子结点
TVE_EXPAND	展开所有子结点
TVE_TOGGLE	如果当前是展开的则收缩，反之则展开

④ 函数 GetNextItem 用来获取下一个结点的句柄。它的原型如下：

```
HTREEITEM GetNextItem( HTREEITEM hItem, UINT nCode );
```

其中，hItem 指定参考结点的句柄，nCode 用来指定与 hItem 的关系标志，常见的标志有：

TVGN_CARET	返回当前选择结点的句柄
TVGN_CHILD	返回第一个子结点句柄，hItem必须为NULL
TVGN_NEXT	返回下一个兄弟结点(同一个树支上的结点)句柄
TVGN_PARENT	返回指定结点的父结点句柄
TVGN_PREVIOUS	返回上一个兄弟结点句柄
TVGN_ROOT	返回hItem父结点的第一个子结点句柄

⑤ 函数 HitTest 用来测试鼠标当前操作的位置位于哪一个结点中，并返回该结点句柄。它的原型如下：

```
HTREEITEM HitTest( CPoint pt, UINT* pFlags );
```

其中，pFlags 包含当前鼠标所在的位置标志，如下列常用定义：

TVHT_ONITEM	在结点上
TVHT_ONITEMBUTTON	在结点前面的按钮上
TVHT_ONITEMICON	在结点文本前面的图标上
TVHT_ONITEMLABEL	在结点文本上

除了上述操作外，还有其他常见操作，如表 3.27 所示。

表 3.27 CTreeCtrl 类其他常见操作

成员函数	说明
UINT GetCount( )；	获取树中结点的数目，若没有返回–1
BOOL ItemHasChildren(HTREEITEM hItem);	判断一个结点是否有子结点
HTREEITEM GetChildItem(HTREEITEM hItem);	获取由 hItem 指定的结点的子结点句柄
HTREEITEM GetParentItem(HTREEITEM hItem);	获取由 hItem 指定的结点的父结点句柄
HTREEITEM GetSelectedItem( );	获取当前被选择的结点
HTREEITEM GetRootItem( );	获取根结点句柄
CString GetItemText(HTREEITEM hItem) const;	返回由 hItem 指定的结点的文本
BOOL SetItemText(HTREEITEM hItem, LPCTSTR lpszItem);	设置由 hItem 指定的结点的文本
DWORD GetItemData(HTREEITEM hItem) const;	返回与指定结点关联的 32 位值
BOOL SetItemData(HTREEITEM hItem, DWORD dwData);	设置与指定结点关联的 32 位值
COLORREF SetBkColor(COLORREF clr);	设置控件的背景颜色
COLORREF SetTextColor (COLORREF clr);	设置控件的文本颜色
BOOL SelectItem(HTREEITEM hItem);	选中指定结点
BOOL SortChildren(HTREEITEM hItem);	用来将指定结点的所有子结点排序

3．树视图控件的消息

同列表控件相类似，树控件也可以用 MFC ClassWizard 映射公共控件消息和树控件消息。其中，常用的树控件消息有：

```
TVN_BEGINDRAG          开始拖放操作
TVN_BEGINLABELEDIT     开始编辑文本
TVN_BEGINRDRAG         按鼠标右键开始拖放操作
TVN_ENDLABELEDIT       文本编辑结束
TVN_ITEMEXPANDED       含有子结点的父结点已展开或收缩
TVN_ITEMEXPANDING      含有子结点的父结点将要展开或收缩
TVN_SELCHANGED         当前选择结点发生改变
TVN_SELCHANGING        当前选择结点将要发生改变
```

需要说明的是，在用 ClassWizard 处理上述这些消息时，其消息处理函数参数中往往会出现 NM_TREEVIEW 结构，其定义如下：

```
typedef struct tagNMTREEVIEW
{
    NMHDR       hdr;        // 含有通知代码的信息结构
    UINT        action;     // 通知方式标志
    TVITEM      itemOld;    // 原有结点的信息
    TVITEM      itemNew;    // 现在结点的信息
    POINT       ptDrag;     // 事件产生时，鼠标的位置
} NMTREEVIEW, FAR *LPNMTREEVIEW;
```

4．使用树控件示例

本例用树控件来显示院系、专业和班级信息，如图 3.45 所示。右击结点，将弹出一个

消息对话框显示出该结点的文本。

（1）添加并设计对话框

① 用 MFC AppWizard(exe)创建一个默认的单文档应用程序 Ex_Ctrl7SDI。

② 向应用程序中添加一个对话框资源 IDD_TREE，标题定为“树控件示例”，字体设为“宋体，9 号”，创建此对话框类为 CTreeDlg。

③ 删除 Cancel 按钮，将 OK 的标题改为“退出”。打开对话框网格，参照图 3.45 所示的控件布局，用编辑器为对话框添加一个树控件，取其默认的 ID 号 IDC_TREE1。打开该控件的属性对话框，按图 3.46 进行设置。

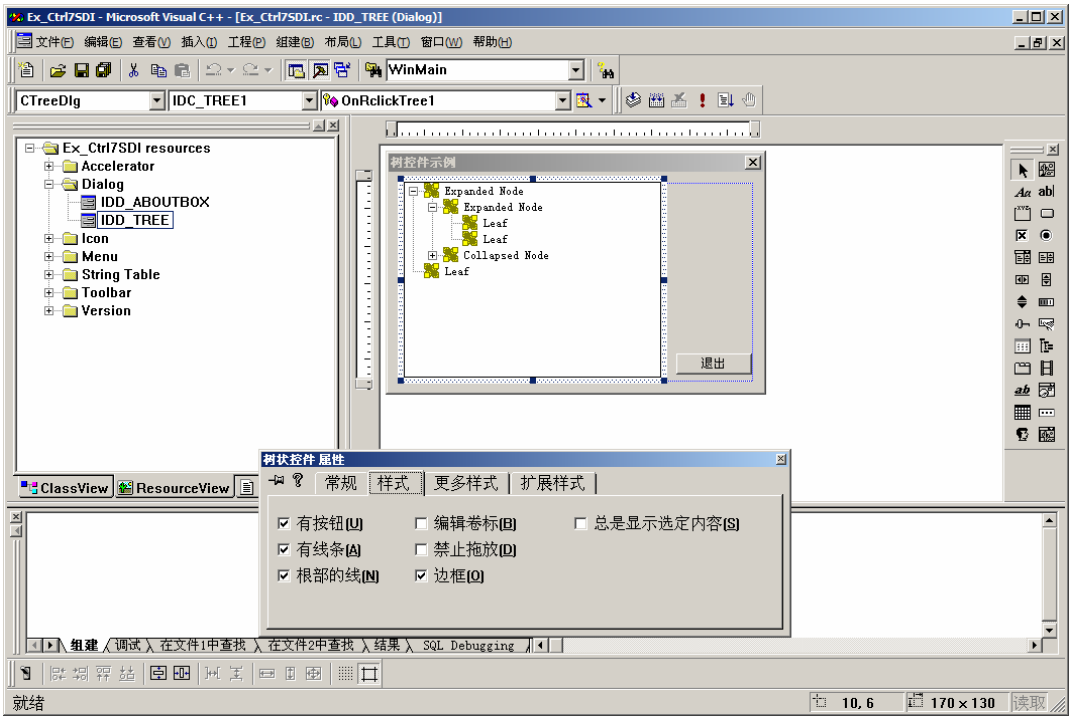

图 3.46　设置树控件样式

（2）完善 CTreeDlg 代码

① 打开 ClassWizard 的 Member Variables 选项卡，看看 Class name 是否是 CTreeDlg，选中 IDC_TREE1，双击鼠标或单击 Add Variables 按钮，为其添加一个控件变量 m_TreeCtrl，类型为 CTreeCtrl。

② 为 CTreeDlg 类添加一个图像列表类 CImageList 对象 m_ImageList。

③ 用 MFC ClassWizard 为 CTreeDlg 类添加 WM_INITDIALOG 消息映射，并添加下列初始化代码：

```
BOOL CTreeDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
```

```
    m_ImageList.Create(16, 16, ILC_COLOR8 | ILC_MASK, 2, 1);
    m_TreeCtrl.SetImageList( &m_ImageList, TVSIL_NORMAL );
    SHFILEINFO fi;                          // 定义一个文件信息结构变量
    SHGetFileInfo("C:\\Windows", 0, &fi, sizeof(SHFILEINFO),
        SHGFI_ICON | SHGFI_SMALLICON);  // 获取文件夹图标
    m_ImageList.Add( fi.hIcon );
    SHGetFileInfo("C:\\Windows", 0, &fi, sizeof(SHFILEINFO),
        SHGFI_ICON | SHGFI_SMALLICON | SHGFI_OPENICON);// 获取打开文件夹图标
    m_ImageList.Add( fi.hIcon );
    HTREEITEM hRoot, hParent;
    hRoot = m_TreeCtrl.InsertItem("电气与电子工程学院",0,1);
    hParent = m_TreeCtrl.InsertItem("电气工程及其自动化",hRoot);
    m_TreeCtrl.InsertItem("210101",0,1, hParent);
    hParent = m_TreeCtrl.InsertItem("机械电子工程",0,1, hRoot);
    m_TreeCtrl.InsertItem("210105",0,1, hParent);
    hRoot = m_TreeCtrl.InsertItem("数学与计算机科学学院",0,1);
    hParent = m_TreeCtrl.InsertItem("计算机应用技术",hRoot);
    m_TreeCtrl.InsertItem("160101",0,1, hParent);
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

其中，SHGetFileInfo 是一个全局函数，通过它可以获取文件或文件夹的图标。

④ 用 MFC ClassWizard 为树控件 IDC_TREE1 添加 NM_RCLICK（右击）消息映射，并增加下列代码：

```
void CTreeDlg::OnRclickTree1(NMHDR* pNMHDR, LRESULT* pResult)
{
    CPoint point;
    UINT uFlags;
    ::GetCursorPos( &point );              // 获取当前鼠标所在的屏幕坐标
    m_TreeCtrl.ScreenToClient( &point ); // 将屏幕坐标转换成树控件中的客户坐标
    HTREEITEM hSel = m_TreeCtrl.HitTest(point, &uFlags);
    // 测试鼠标点是否在一个结点项上，若是，选中该结点
    if ((hSel != NULL) && (TVHT_ONITEM & uFlags))
    {
            m_TreeCtrl.SelectItem(hSel);
            CString strItem = m_TreeCtrl.GetItemText( hSel );
            MessageBox( strItem );
    }
    *pResult = 0;
}
```

（3）调用对话框

① 打开 Ex_Ctrl7SDI 单文档应用程序的菜单资源，添加顶层菜单项“测试(&T)”，在

其下添加一个菜单项“树控件(&T)”，ID 为 ID_TEST_TREE。

② 用 MFC ClassWizard 为 CMainFrame 类添加菜单项 ID_TEST_TREE 的 COMMAND 消息映射，取默认的映射函数名，并添加下列代码：

```
void CMainFrame::OnTestTree()
{
    CTreeDlg dlg;
    dlg.DoModal();
}
```

③ 在文件 MainFrm.cpp 的前面添加 CTreeDlg 类的头文件包含：

```
#include "MainFrm.h"
#include "TreeDlg.h"
```

④ 编译运行并测试。

3.9 总结与提高

由于 Windows 常规的窗口总都是方方正正的矩形，因而这给了人们进行 Windows 界面设计的动力，控件也不除外，许多程序员开发并定制出许许多多形状各异、功能独特的控件。不过，若能熟练使用 MFC 中的控件并能设计出优秀的界面，就足以满足本课程所要的教学目标了。

在 MFC 中，控件是具有独立功能的人机交互的小窗口，这些小窗口除了可以使用自身成员外，还可使用其基类 CWnd 类的公有成员，因为几乎所有的控件类都是从 CWnd 类派生而来的。也正因为如此，当用控件类的 Create 创建控件时，除了自身的样式预定义标识外，还有窗口通用的样式预定义标识。

将对话框资源作为控件的界面模板（容器），控件的“创建”就变得“所见即所得”了，通过编辑器中的属性对话框可简单方便地设置控件的样式。不过，当对话框资源模板创建对话框类后，这些控件则只能以成员的形式出现在对话框类中。

用 MFC ClassWizard 可为控件在对话框类中创建两种类别的成员变量：一是控制类，即创建的是控件类对象，二是数据类，即创建的是控件数据变量。这两种类别的成员在对话框类中只能各有一个。控件类对象可以引用控件类及其基类的公有成员，从而实现控件的操作；而数据类变量则是与控件绑定在一起，当使用 UpdateData(TRUE)或 UpdateData() 时将控件上的数据存储到绑定的数据变量中，而当使用 UpdateData(FALSE)时，则是将绑定的数据变量的数值回填到控件中。

控件除了在对话框类中使用控件变量操作外，还可通过控件的消息映射来实现代码功能。不同控件的“通知消息”有所不同，总体可分为三类：一类是与界面相关的单击、选择与取消或展开与收缩等的命令消息；二是与输入焦点（Focus）相关的失去、得到等消息；三是与数据相关的更新、改变等消息。这些消息，系统都会用一个称为 MSG 结构的系统变量来记录，并可用 MFC ClassWizard 对话框对其进行映射。

需要说明的是，MFC 的控件不同于 VB（Visual Basic）中的控件。MFC 的控件更注重于控件的程序控制，而 VB 中的控件更注重于控件的界面设计。简单来说，VB 中的控件更“傻瓜”一些，这就使许多学习 MFC 的人似乎更欣赏 VB 的做法。事实上，MFC 的控件也可进行更深入的界面设计，它提供了两种层次不同的方法：一是使用“自画”（Owner Draw，所有者绘制）体系，二是跟踪消息。

“自画”体系是 MFC 中层次较高的定制控件功能和外观的方法，外观上可以通过重载 DrawItem 函数达到自画的目的。由于这种方法需要更多的代码，因而这里暂不作讨论，留待学透之后再来深究。

跟踪消息的方法倒是比较简单，在前面的“调整对话框背景颜色”示例中，用到了 WM_CTLCOLOR 消息。这个消息是当对话框及控件等在显示之前向父窗口发生的消息，通过跟踪这个消息，在 WM_CTLCOLOR 消息函数 OnCtlColor 返回之前，指定返回一个 HBRUSH，系统就会用它绘制控件，从而改变控件的背景颜色。当然，也可在此函数中添加设置控件文本的颜色、格式等的代码，从而改变控件的外观。例如，在示例 Ex_Ctrl6SDI 的基础上，为 CStuInfoDlg 类添加 WM_CTLCOLOR 消息映射，并添加下列代码：

```
HBRUSH CStuInfoDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
    if ( nCtlColor == CTLCOLOR_STATIC)
    {
            pDC->SetBkColor( RGB( 0, 0, 192 ) );
            pDC->SetTextColor( RGB( 0, 128, 128 ) );
    }
    return hbr;
}
```

编译运行后，在应用程序的菜单上，选择“测试”→“学生基本信息”菜单项，将弹出如图 3.47 所示的对话框。可以看出，对话框中所有静态控件（或与之相关）的文本的前景和背景颜色发生了改变。

图 3.47　改变控件颜色

类似地，也可通过（nCtlColor == CTLCOLOR_LISTBOX），使列表框的颜色得以改变；除此之外，还有对按钮（CTLCOLOR_BTN）、对话框（CTLCOLOR_DLG）、编辑框（CTLCOLOR_EDIT）、消息框（CTLCOLOR_MSGBOX）、滚动条（TLCOLOR_SCROLLBAR）、静态文本（CTLCOLOR_STATIC）等控件进行类似操作。

在代码中，SetBkColor、SetTextColor 都是环境设备类 CDC 的成员函数，分别用来设置环境设备的背景和文本颜色。RGB 是一个颜色宏，用来得到由颜色红（Red）、绿（Green）、蓝（Blue）各分量值（0～255）所构成的颜色。

可见在界面设计中，对话框是一种常用的模板，它包含了许多控件等界面元素。实际上，在文档应用程序中，除了对话框外还有菜单、工具栏和状态栏等界面模板，第 4 章将讨论。