第5章 对 话 框

对话框(dialog box)是 Windows 应用程序中一种常用的资源。它其实是一个“窗口”,是 Windows 程序与用户交互的一个手段,它的主要功能是输出信息和接收用户的输入。它可以只是一个简单的 OK 消息框,也可以是一个复杂的数据输入表单。在对话框内一般都有一些控件(control),对话框依靠这些控件与用户进行交互。

本章主要讲解对话框的原理和编程方法,下一章将详细介绍各种控件的使用。

5.1 对话框概述

Windows 应用程序虽然提供了菜单和工具栏等界面元素,但就用户交互输入功能而言,菜单和工具栏的功能是有限的。对话框除了用来显示提示信息(例如程序启动时显示版权和运行进度),主要用于接收用户的输入数据。在 MFC 中,对话框的功能被封装在 CDialog 类中,而 CDialog 类是 CWnd 类的派生类。作为窗口,对话框具有窗口的一切功能。对话框的一个典型应用是通过菜单命令或工具栏按钮打开一个对话框,当然也可以编写基于对话框的应用程序,将对话框作为一个程序的主界面。

5.1.1 对话框的类型

尽管不同对话框的外观、大小和对话框的控件千差万别,但从对话框的工作方式看,对话框可分为模态对话框和非模态对话框两种类型。

当一个模态对话框打开后,在其关闭之前,用户不能转向其他用户界面对象,而只能与该对话框进行交互。大家平时接触到的对话框大多数是模态对话框。例如,选择“文件”|“另存为”菜单命令后,“另存为”对话框被打开,用户不能再做其他工作,只有保存完文件或取消保存文件、关闭对话框窗口后才能做其他工作。

非模态对话框恰恰相反,当打开一个非模态对话框,对话框停留在屏幕上时,仍然允许用户与其他窗口进行交互。最典型的非模态对话框是在 Word 中使用的“查找和替换”对话框,打开该对话框后,可以交替地进行文档编辑和查找替换操作。

5.1.2 对话框的 CDialog 类

为了方便实现对话框功能,MFC 提供了一系列对话框类,并实现了对话框消息响应和处理机制。CDialog 类是对话框类中最重要的类,用户在程序中创建的对话框类一般都是 CDialog 类的派生类。CDialog 类还是其他所有对话框类的基类,其派生关系如图 5.1 所示。对话框类为程序员提供了管理对话框的编程接口。

CDialog 类从 CWnd 类派生而来，所以它继承了 CWnd 类的成员函数，具有 CWnd 类的基本功能，可以编写代码移动、显示或隐藏对话框，并能根据对话框的特点增加新的成员函数，扩展它的功能。表 5.1 列出了 CDialog 类中经常要使用的成员函数，在用户的 CDialog 类的派生类中可以直接调用。大部分的成员函数是虚函数，可以在用户的派生类中重新定义，以实现特定的目的。除了 CDialog 类成员函数，CWnd 和 CWinApp 类也提供了一些成员函数用于对话框的管理，常用的函数见表 5.1。

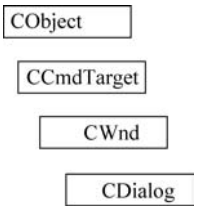

图 5.1 CDialog 类的派生关系

表 5.1 有关对话框的常用处理函数

成员函数	功能
CDialog::CDialog()	构造一个 CDialog 对象
CDialog::DoModal()	激活模态对话框，显示对话框窗口，直到对话框窗口关闭时返回
CDialog::Create()	根据对话框资源模板创建非模态对话框窗口并立即返回。如果对话框不是 Visible 属性，还需通过调用 CWnd::ShowWindow()函数显示非模态对话框窗口
CDialog::OnOK()	单击 OK 按钮时调用该函数，接收对话框输入数据，关闭对话框。默认实现时调用 EndDialog()函数关闭对话框和使 DoModal()返回值 IDOK
CDialog::OnCancel()	单击 Cancel 按钮或按 Esc 键时调用该函数，不接收对话框输入数据，关闭对话框。默认实现时调用 EndDialog()函数关闭对话框和使 DoModal()返回值 IDCANCEL
CDialog::OnInitDialog()	WM_INITDIALOG 的消息处理函数。在调用 DoModal()或 Create()函数时系统发送 WM_INITDIALOG 消息，在显示对话框前调用该函数进行初始化工作
CDialog::EndDialog()	关闭对话框窗口
CWnd::ShowWindow()	显示或隐藏对话框窗口
CWnd::DestroyWindow()	关闭并销毁非模态对话框
CWnd::UpdateData()	通过调用 DoDataExchange()设置或获取对话框控件的数据。单击 OK 按钮关闭模态对话框或默认 OnInitDialog()时该函数被自动调用来设置控件的初始值
CWnd::DoDataExchange()	被 UpdateData()调用以实现对话框的数据交换与校验，不能直接调用
CWnd::PostNcDestroy()	这个虚函数在窗口销毁时被自动调用
CWnd::SetActiveWindow()	激活窗口
CWnd::UpdateWindow()	更新客户区

5.1.3 对话框的组成

可以通过应用程序向导生成基于对话框的应用程序，在文档/视图结构应用程序中也大量使用对话框。在 Windows 中对话框是作为一种资源被使用，从 MFC 编程的角度看，对话框主要由以下两部分组成。

(1) 对话框模板资源：对话框模板资源定义了对话框的特性(例如大小、位置和风格)以及对话框中每个控件的类型和位置。

(2) 对话框类：对话框类用来实现对话框的功能。由于对话框行使的功能各不相同，

所以一般需要从 CDialog 类派生一个新类，以提供编程接口来管理相应的对话框。

因此在程序中要设计一个对话框，首先要设计一个对话框模板资源，然后设计一个基于该对话框模板资源的对话框类，最后声明对话框类的对象以便激活并打开对话框。

5.2 模态对话框

模态对话框是最常用的对话框。本节以模态对话框编程为例来介绍如何在程序中设计对话框和运行对话框，同时分析对话框的数据交换和校验机制。

5.2.1 设计对话框模板资源

可以使用对话框编辑器来创建包含不同控件的对话框模板资源。直接双击项目中的对话框资源打开对话框编辑器；或者使用 Visual Studio IDE 中的“添加资源”工具，选中 Dialog 后单击“新建”按钮打开对话框编辑器；或者在项目的“资源视图”窗口中找到 Dialog 资源，右击后选择“插入 Dialog”菜单命令来打开对话框编辑器。图 5.2 即为打开的对话框编辑器。

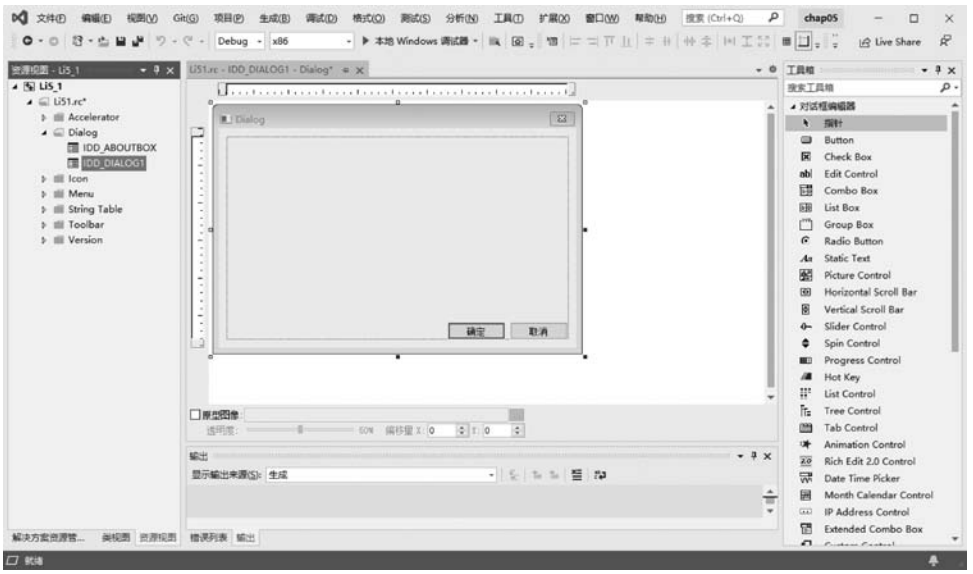

图 5.2 对话框编辑器

对话框编辑器会对项目的资源(RC)文件进行更新，使之包含新的对话框资源，并且该项目的 resource.h 文件也会被更新。在对话框编辑器中可以调整对话框显示时的大小和位置，从控件工具栏拖放各种类型的控件到对话框中，用控件布局工具栏调整控件的位置，测试对话框的外观和行为。另外，也可以直接用文本方式来编辑项目的资源文件，这需要用户掌握资源脚本的编写方法。

如果没有控件，对话框完不成具体功能，对话框与控件有着密不可分的关系。所以设计对话框模板资源有两个重要的内容，第一是从“工具箱”选择控件添加到对话框中，并调整其位置和大小；第二是在属性窗口中设置控件的“描述文字”、ID 以及其他属性。

1. 增加或删除控件

控件是独立的小部件，能够放置在一个对话框中，提供应用程序与用户交互的某种功能。控件的种类较多，图 5.3 中所示为 Visual Studio 2019 支持的部分标准控件，用户可以很方便地从“工具箱”中添加新的控件到对话框中。

在一个对话框资源中添加控件的操作十分方便，只需从图 5.3 所示的“工具箱”中选中要增加的控件，再将此控件拖动至对话框模板中的确定位置，松开鼠标按键即添加了一个控件。调整控件的位置和大小的操作与 Word 中对文本框的操作完全一样，这里不再赘述。

如果要删除已添加的控件，先单击对话框中的控件，再按 Delete 键即可。

在默认情况下，“工具箱”窗口总是打开的。如果没有打开，可以通过 IDE 的“视图”|“工具箱”菜单命令将其打开。用户可以通过设置“工具箱”窗口的停靠属性或者通过直接拖曳的方式来调整工具箱的位置。

2. 设置控件的属性

一个控件的相关属性决定了一个控件的可操作行为和显示效果，属性的设置是在与每个控件相对应的属性窗口中进行的。将光标指向对话框中需要设置属性的控件，右击该控件，在弹出的快捷菜单中选择“属性”菜单命令，打开控件的属性窗口。每一种控件的属性窗口中的内容都有所不同，与其特性相关，第 6 章将详细介绍一些常用控件的属性的含义。一个控件的典型的属性窗口如图 5.4 所示，其中的 ID 和“描述文字”的含义及设置方法与菜单项相同。

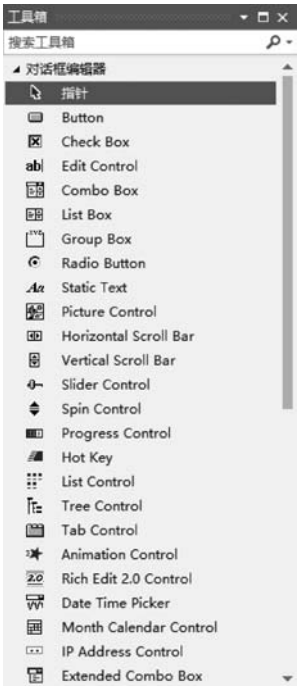

图 5.3　工具箱中的部分标准控件

图 5.4　控件的属性窗口

同样，对话框的属性是在对话框的属性窗口中设置的，在对话框的任意空白处右击，在弹出的快捷菜单中选择“属性”菜单命令，弹出如图 5.5 所示的对话框属性窗口。在“外观”|

“描述文字”中填写对话框的标题，单击“字体”|“字体(大小)”会弹出“字体”对话框，设置对话框中显示的字体格式。其他的属性设置请大家自行测试。

3. 测试对话框的运行效果

当设计好一个对话框模板资源后，还不能立即在应用程序中运行对话框，MFC 提供了“测试对话框”命令，使程序员在设计阶段就能够测试对话框的运行效果。

测试对话框的方法有下面 3 种。

(1) 选择“格式”|“测试对话框”菜单命令。

(2) 单击布局工具栏上的“测试对话框”按钮。

(3) 按 Ctrl+T 键。

通过测试对话框就可以评估对话框是否符合要求。如果发现了错误或不满意的地方，可以按 Esc 键退出测试对话框并重新修改对话框模板资源。

视频讲解

【例 5.1】 创建一个单文档的 MFC 应用程序 MyDialog，向应用程序中添加如图 5.6 所示的对话框模板资源，并设置控件的“描述文字”和 ID 属性。

图 5.5 对话框的属性窗口

图 5.6 模态对话框

其操作步骤如下：

(1) 使用“MFC 应用”项目模板创建一个项目名为 MyDialog 的单文档应用程序。

(2) 插入新的对话框模板。右击“解决方案资源管理器”窗口中的 MyDialog 项目，选择快捷菜单中的“添加”|“资源”菜单命令，打开“添加资源”窗口，选中其中的 Dialog 后单击“新建”按钮，插入一个新的对话框模板。Visual Studio IDE 提供的对话框模板创建了一个基本界面，包括一个“确定”按钮和一个“取消”按钮等。可以看到“资源视图”窗口的 Dialog 文件夹下增加了一个对话框资源 IDD_DIALOG1，如图 5.2 所示。

(3) 在新的对话框模板上添加控件。从工具箱中选取一个 Aa 静态控件，该控件作为输入框的提示文本。再从工具箱中选取一个 ab| 编辑框控件，该控件用来接受输入的数据。

(4) 按图 5.6 所示的布局调整控件的大小和位置。

(5) 设置控件的属性。选择静态控件的“描述文字”属性,将其值设置为“请输入边长”;选择编辑框控件,将其 ID 属性设置为 IDC_LENGTH。

(6) 设置对话框的属性。打开对话框的属性窗口,将其“描述文字”属性设置为“输入边长”。

(7) 测试对话框。单击布局工具栏上的“测试对话框”按钮,测试并重新修改对话框模板资源,直到满意为止。

5.2.2 设计对话框类

一旦完成了对话框的属性和外观设计,接下来就是设计其行为。设计对话框类主要包括以下几个方面:

第一,从 MFC 的 CDialog 中派生出一个类,用来负责对话框行为。

第二,利用 ClassWizard 把这个类和先前产生的对话框资源连接起来。通常这意味着必须声明某些函数,用于处理相应的对话框消息,并将对话框中的控件对应到类的成员变量上,这也就是所谓的对话框数据交换(Dialog Data eXchange,DDX)。如果用户对这些变量内容有任何“确认规则”,ClassWizard 也允许用户设定它,这就是所谓的数据验证(Dialog Data Validation,DDV)。

第三,对话框的初始化。

1. 创建对话框类

使用“类向导”工具可以十分方便地创建 MFC 窗口类的派生类,对话框类也不例外。

【例 5.2】 完善例 5.1 中的应用程序 MyDialog,给对话框资源添加相应的对话框类。


视频讲解

其操作步骤如下:

(1) 保存已创建的对话框资源。

(2) 确保新的对话框资源在对话框编辑器中处于打开状态,右击对话框的空白区域,在弹出的快捷菜单中选择“添加类”菜单命令,为新对话框添加一个与之相关联的类,如图 5.7 所示。

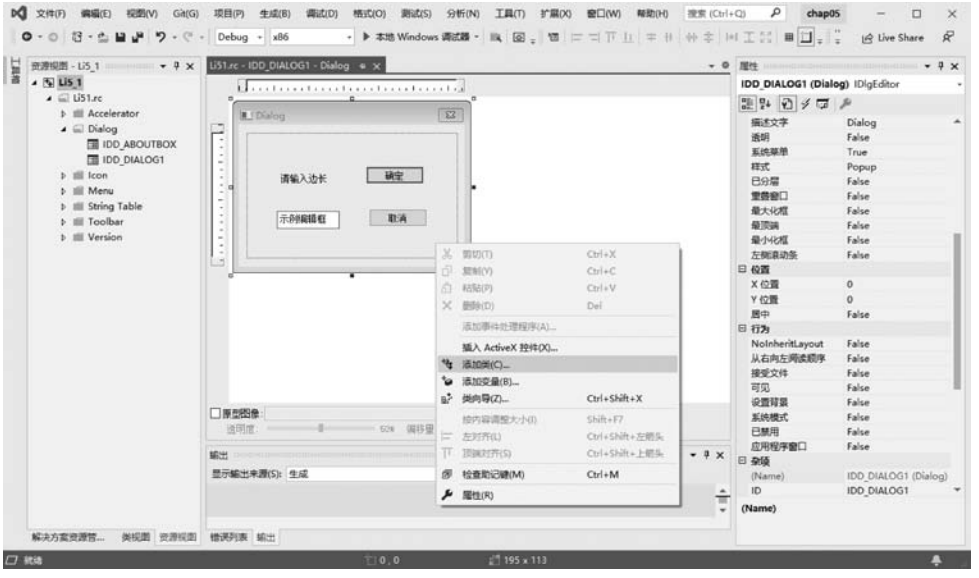

图 5.7 添加对话框关联类

(3) 在“添加 MFC 类”窗口中输入新类的相关信息。其中,类名为 CSquare,基类为 CDialog,头文件为 Square.h,源文件为 Square.cpp,对话框 ID 为 IDD_DIALOG1,如图 5.8 所示。

图 5.8 “添加 MFC 类”窗口

新类添加完成后,在 IDE 的“类视图”窗口中可以看到增加了一个新的类 CSquare,切换到“解决方案资源管理器”窗口,在项目的“头文件”和“源文件”文件夹中可以看到该类的头文件和实现文件,如图 5.9 所示。

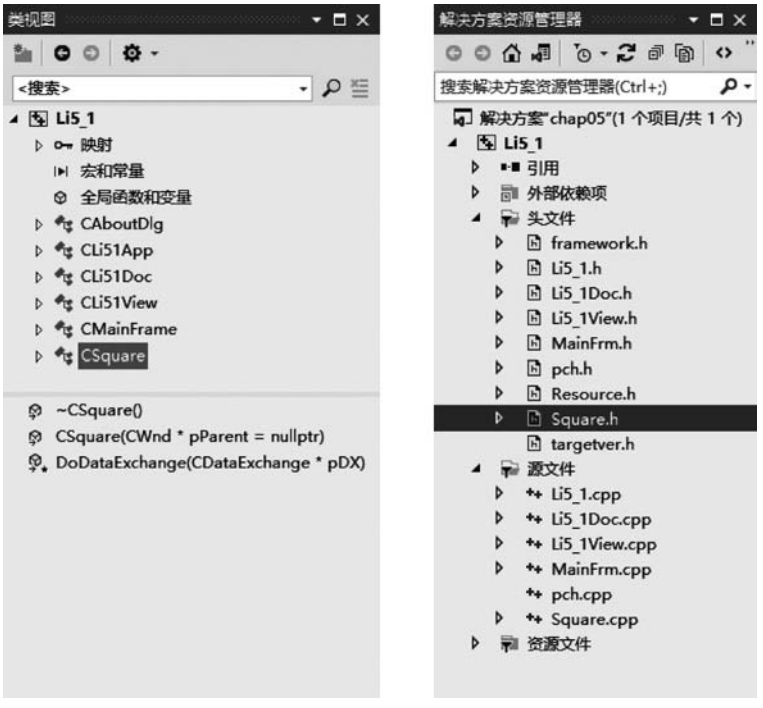

图 5.9 新增对话框类及文件

2. 创建对话框成员变量

在创建一个对话框类后，可以增加类的成员变量来操作对话框上的控件。可以为对话框资源上的每一个控件添加一个或多个对应的成员变量。类向导的“成员变量”选项卡主要用来为对话框类添加和删除与对话框控件相关联的成员变量，在编写对话框程序时经常和该选项卡“打交道”。打开“类向导”工具，选择项目中的对话框类，切换到“成员变量”选项卡，即可看到对话框类中控件对象的成员变量信息，如图 5.10 所示。

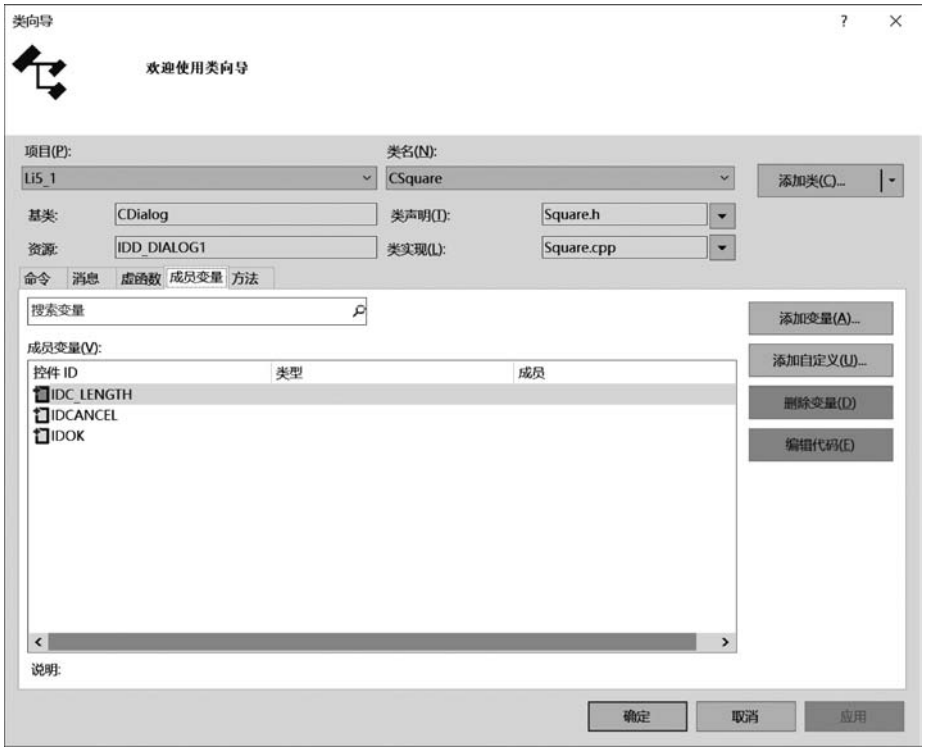

图 5.10　控件的成员变量

在类向导的“成员变量”选项卡中有一个标题为“成员变量”的列表框，第 1 列“控件 ID”表示控件的 ID，第 2 列“类型”表示变量的类型，第 3 列“成员”表示成员变量名。双击一个 ID 或选定 ID 后，单击“添加变量”按钮，将弹出“添加控制变量”对话框，如图 5.11 所示。“名

图 5.11　“添加控制变量”对话框

称”框用于输入成员变量名，类向导建议以“m_”作为成员变量名的前缀。不同控件所关联的成员变量的类型不一定相同，“类别”下拉列表框用于选择成员变量的类别。出于不同的操作目的，MFC 提供了两种类型的成员变量，如表 5.2 所示。用户可以为一个控件同时定义一个 Value 类型的变量和一个 Control 类型的变量。

表 5.2　对话框类的成员变量类型

类　　型	描　　述
Value	值类型的成员变量，用于控件的值的控制。它可以有多种数据类型，由所连接的控件类型决定。例如，EditBox 控件可以有 CString 型或 int 型；RadioButton 可以是 int 型
Control	控件类型的成员变量，实际上是该控件类的一个对象。在创建了一个控件对象之后，就可以通过该对象使用所属控件类的成员函数对控件进行操作，例如在程序运行时为 ComboBox 加入选择项，设置控件是否有效或可见等

添加操作结束后，回到“类向导”工具的“成员变量”选项卡，在列表框中显示出了目前已创建的成员变量及它们的类型。如果需要规定某个成员变量的有效性校验规则，只需要在添加变量时在如图 5.11 所示的“其他”窗口中设置即可。如果需要修改已添加的成员变量，必须先删除该变量，然后再进行添加操作。选中需要删除的成员变量，单击“删除变量”按钮即可删除该变量。

视频讲解

【例 5.3】 继续完善例 5.2 中的应用程序 MyDialog，在对话框类中添加与控件相关联的成员变量。

其操作步骤如下：

(1) 打开“类向导”工具，选择 CSquare 类下的“成员变量”选项卡。

(2) 添加与编辑框控件相关联的成员变量。双击“成员变量”列表中的编辑框 IDC_LENGTH，通过“添加控制变量”对话框添加成员变量。在“名称”框中填写变量名 m_length，在“类别”下拉列表框中选择 Value，在“变量类型”中输入数据类型 int。

(3) 给变量 m_length 添加校验规则。在“添加控制变量”窗口中选择“其他”或单击“下一步”按钮，打开“添加控制变量”的其他设置窗口，在这里输入 m_length 的最小值 10 和最大值 200。

(4) 单击“完成”按钮回到“类向导”工具窗口，依次单击“应用”和“确定”按钮关闭“类向导”工具。

3. 对话框的初始化

除了以上工作，对话框上的许多控件还需要进行初始化，从而使对话框被显示时这些控件具有相应类型的初值。对话框的初始化工作可以使用以下 3 种方法来进行。

1) 在构造函数中初始化

从 C++的观点看，在类的构造函数中应该初始化类的数据成员，但是在 MFC 应用程序中应尽量避免在构造函数中完成太多的工作，因为构造函数没有返回失败条件的方法，无法报告其中的失败信息。在构造函数中初始化主要是针对对话框的数据成员。

打开应用程序 MyDialog 中 CSquare 类的构造函数，可以看到“类向导”工具自动加入了成员变量的初始代码，代码如下：

```
//CSquare 类的构造函数
```

```
CSquare::CSquare(CWnd * pParent /* = nullptr */)
    : CDialog(IDD_DIALOG1, pParent)
    , m_length(0)
{
}
```

2) WM_CREATE 初始化

由于对话框也是窗口，它在窗口创建时也会收到 WM_CREATE 消息，这样就能在 WM_CREATE 的消息处理函数 OnCreate()中进行一些数据成员的初始化工作。但由于此时很多控件尚未建立，有些控件的初始化工作还无法完成。

若要响应 WM_CREATE 进行初始化工作，需要在创建对话框类时使用“类向导”工具对消息 WM_CREATE 进行映射，从而形成 OnCreate()函数后再编写代码。

3) WM_INITDIALOG 初始化

在收到 WM_INITDIALOG 消息时，对话框处于这样一种状态：首先，对话框框架已经建立起来；其次，各个控件也建立起来并放在适当的地方；最后，对话框还没有显示出来。这样就可以设置或优化对话框中各个控件的外观、大小尺寸、位置及其他内容。这是构造函数和 WM_CREATE 无法相比的，所以对于对话框的初始化工作通常都在响应该消息时进行。可以通过“类向导”工具映射该消息以得到其处理函数 OnInitDialog()，初始化工作可在该函数中进行。

5.2.3 运行对话框

在完成一个对话框资源和对话框类的设计之后，就可以在应用程序中运行该对话框。模态对话框的运行分两个步骤：首先创建一个对话框对象，然后调用 CDialog::DoModal()函数打开对话框。

DoModal()函数负责模态对话框的创建和撤销，可以根据其返回值是 IDOK 还是 IDCANCEL 来判断用户关闭对话框时按的是哪一个键。


视频讲解

【例 5.4】 完善例 5.3 中的应用程序 MyDialog，通过“对话框”|“模态对话框”菜单项打开上述标题为“输入边长”的对话框，并根据输入的边长画一个正方形。

其操作步骤如下：

(1) 在 CMyDialogView 视图类中添加一个类型为 int 的成员变量 m_vlength，用来在视图中接收并存储对话框类的成员变量 m_length 的值，并在构造函数 CMyDialogView()中将它初始化为 0。

```
CMyDialogView::CMyDialogView()
{
    m_vlength = 0;                                    //设置编辑框显示的初始值
}
```

(2) 使用菜单编辑器增加一个“对话框”主菜单，并在其中添加“模态对话框”菜单项，其 ID 为 ID_MODEDLG。

(3) 利用“类向导”工具在视图类中为 ID_MODEDLG 菜单项添加 COMMAND 消息的处理函数，在函数中添加如下代码。

```
void CMyDialogView::OnModedlg()
{
    CSquare dlg;                                    //定义一个对话框对象
    if(dlg.DoModal() == IDOK)                       //显示对话框
    {
        m_vlength = dlg.m_length;                   //接收并存储编辑框数据
        Invalidate();                               //刷新视图
    }
}
```

注意：为了简单，在上述代码中直接使用 dlg 对象访问其成员变量 m_length，所以要求 m_length 的访问权限必须是 public。

(4) 在成员函数 CMyDialogView::OnDraw()中添加绘制正方形的语句。

```
pDC->Rectangle(10,10,10 + m_vlength,10 + m_vlength);
```

(5) 在视图类实现文件 MyDialogView.cpp 的所有 include 语句之后加入包含对话框类头文件的语句。

```
#include "Square.h"
```

(6) 编译、链接并运行程序，选择“对话框”|“模态对话框”菜单命令，打开“输入边长”对话框。输入边长后单击“确定”按钮，程序将在客户区画出一个正方形。

5.2.4 对话框的数据交换和校验机制

在运行对话框时，对话框类的成员变量需要与控件交换数据，以完成输入和输出功能。CDialog 类通过调用其成员函数 DoDataExchange()实现对话框的数据交换和验证，而在 DoDataExchange()中使用了 MFC 提供的 CDataExchange 类，该类实现对话框类的成员变量与控件之间的数据交换(DDX)和数据验证(DDV)。DDX 将成员变量与对话框控件相连接，完成数据在成员变量和控件之间的交换。DDV 用于数据的校验，它能自动校验输入的数据(如字符串的长度或数值的范围)是否符合设计要求。DDX 和 DDV 适用于文本框、复选框、单选按钮、列表框和组合按钮。

在对话框中使用 DDX/DDV 非常简单，一般不需要编写 CDataExchange 类的代码，DoDataExchange()函数也由框架调用。用户只需通过“类向导”工具为对话框类添加与对话框控件相关联的成员变量即可。在应用程序 MyDialog 中可以找到下列函数：

```
void CSquare::DoDataExchange(CDataExchange * pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX, IDC_LENGTH, m_length);
    DDV_MinMaxInt(pDX, m_length, 10, 200);
}
```

可见“类向导”工具替用户完成了所有的工作。DoDataExchange()函数只有一个参数 CDataExchange * pDX。在该函数中调用 DDX 函数来完成数据交换，调用 DDV 函数来完成数据的有效性检查。当利用“类向导”工具为对话框类添加与对话框控件相关联的成员变量后，它会自动在该函数中添加代码。下列语句

```
DDX_Text(pDX, IDC_LENGTH, m_length);
DDV_MinMaxInt(pDX, m_length, 10, 200);
```

中第一句是 DDX 函数调用语句，表明 m_length 是一个 Value 值类别的成员变量，用于交换 IDC_LENGTH 控件中的内容；第二句是 DDV 函数调用语句，程序运行后，如果用户的输入数据超出 10～200 的范围，DDV 将显示如图 5.12 所示的提示信息对话框，提示用户有效的输入范围。

图 5.12　DDV 提示信息对话框

需要说明的是，虽然 DoDataExchange()函数实现了 DDX/DDV 功能，但用户不能直接调用 DoDataExchange()函数，它一般由成员函数 CWnd::UpdateData()调用。用户可以随时在需要进行数据交换的地方调用 UpdateData()函数。

UpdateData()函数只有一个 BOOL 类型的参数。当参数为 TRUE 时，MFC 通过调用 DoDataExchange()函数将数据从控件传递到相关联的成员变量；当参数为 FALSE 时，数据从成员变量传递到相关联的控件。

应用程序 MyDialog 表面上看没有调用 UpdateData()函数。但是在创建对话框时，DoModal()的任务包括装载对话框资源、调用 OnInitDialog()初始化对话框并将对话框显示在屏幕上。在默认的 CDialog::OnInitDialog()函数中调用了 UpdateData(FALSE)，这样数据成员的初值就反映到了相应的控件上。而单击系统默认生成的“确定”按钮，将调用 CDialog::OnOK()函数，CDialog::OnOK()函数中调用了 UpdateData(TRUE)，将控件中的数据传给成员变量。图 5.13 描绘了对话框的这种数据交换机制。

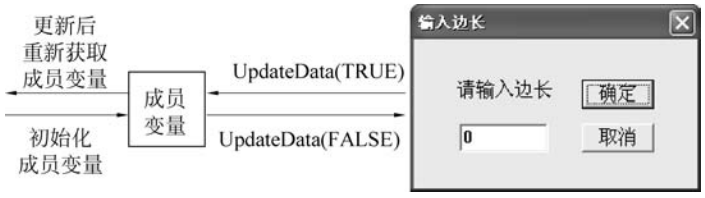

图 5.13　对话框的数据交换

由此看来，无论 MFC 将 DDX 技术如何复杂化，大家只需知道 DDX 如同一条双向通道，而方向控制开关就是 UpdateData()函数中的参数值是 TRUE 还是 FALSE。

5.3　非模态对话框

区别于模态对话框，非模态对话框弹出后，用户不需要关闭它就可以在非模态对话框和应用程序的其他窗口之间进行切换。

5.3.1　非模态对话框的特点

对于非模态对话框，使用对话框编辑器创建对话框资源以及使用“类向导”工具添加对话框类、成员变量和消息处理函数的方法与模态对话框完全一样，但创建和退出对话框的方

式存在着差异。

1. “可见”属性

在对话框的属性窗口的“行为”分组中有“可见”属性的设置，默认情况下，对话框模板不选择“可见”属性。模态对话框不需要设置该属性，而非模态对话框必须将该属性值设置为TRUE，否则对话框是不显示的。保险的办法是调用 CWnd::ShowWindow(SW_ SHOW)来显示对话框，而不管对话框是否具有可见风格。

2. 对话框窗口的创建方式

非模态对话框的创建与普通窗口的创建是一样的，通过调用 CWnd::Create()函数来创建对话框，这是非模态对话框的关键所在。Create()函数的原型有两种，最常用的为

```
BOOL Create(UINT nIDTemplate,CWnd * pParentWnd = NULL);
```

其中，nIDTemplate 是对话框资源模板的 ID 标识，pParentWnd 为对话框父窗口的指针。

由于 Create()函数不会启动新的消息循环，即对话框与应用程序共用一个消息循环，这样对话框就不会屏蔽用户对其他界面对象的访问。Create()函数与 DoModal()函数的不同之处是 Create()函数创建了对话框后立即返回，而 DoModal()函数要在对话框关闭后才会返回。

3. 对话框对象的创建方式

众所周知，在 MFC 程序中窗口对象的生存期应长于对应的窗口，也就是说不能在未关闭屏幕上窗口的情况下先把对应的窗口对象删除。由于在 Create()返回后不能确定对话框是否已关闭，这样也就无法确定对话框对象的生存期，所以不能以局部变量的形式创建非模态对话框的对象，只能用 new 操作符动态创建，并且在调用对话框类的窗口类内声明一个指向对话框类的指针变量，通过该指针访问对话框对象。

4. 窗口删除函数

非模态对话框必须调用 CWnd::DestoryWindow()来关闭对话框。模态对话框是调用 CDialog::EndDialog()关闭对话框。由于默认的对话框函数 OnOK()和 OnCancel()都是调用 EndDialog()关闭对话框的，该函数使对话框不可见但不删除对话框对象，所以非模态对话框类要定义自己的 OnOK()和 OnCancel()函数，调用 DestoryWindow()来关闭对话框。

5. 清理对话框对象的方式

与创建对象的方式——new 操作相对应，使用 delete 操作删除一个非模态对话框对象。由于当屏幕上的一个窗口被关闭后，框架会自动调用 CWnd::PostNcDestroy()函数，也可以编写程序代码，在这个函数中清理非模态对话框对象。

6. 必须有一个标志表明非模态对话框是否为打开的

因为在非模态对话框打开的情况下，用户有可能再次选择打开该对话框，这时不能再创建一个新的非模态对话框。程序根据标志来判断是打开一个新的对话框还是激活一个已打开的对话框。通常可以用拥有者窗口中的指向对话框对象的指针作为这种标志，当对话框关闭时给该指针赋 NULL 值，以表明对话框对象已经不存在了。

视频讲解

【例 5.5】 创建一个单文档的 MFC 应用程序 Li5_5，以非模态对话框的形式实现与应用程序 MyDialog 同样的功能。

其操作步骤如下：

(1) 使用“MFC 应用”项目模板创建一个项目名为 Li5_5 的单文档应用程序。

(2) 与应用程序 MyDialog 一样创建对话框资源和对话框类，并在类中添加与控件相关联的成员变量。

(3) 定义对话框指针。

① 在 CLi55View 视图类中增加 CSquare 指针类型的公有成员变量 m_pDlg，并初始化为 NULL。

② 在 Li5_5View.h 文件的头部预编译命令之前增加类的前向声明语句。

```
class CSquare;
class CLi5_5Doc;
```

类的前向声明语句的作用为：由于在 CLi5_5View 类中有一个 CSquare 类的指针和一个返回值为 CLi5_5Doc 指针的 GetDocument() 函数，所以必须保证 CSquare 类和 CLi5_5Doc 类的声明出现在 CLi5_5View 之前，否则会产生编译错误。

(4) 增加对话框类成员变量接收视图指针。

① 在 CSquare 类中增加 CLi5_5View 指针类型的公有成员变量 m_pParent，并在 CSquare 类的构造函数中添加对该变量进行初始化的语句。

```
m_pParent = (CLi55View * )pParent;
```

这样就可以在 CSquare 类的其他函数中利用这个指针向主窗口发送消息了。

② 在 Square.h 文件的头部预编译命令之前增加类的前向声明语句。

```
class CLi5_5View;
```

(5) 在 Li5_5View.cpp 文件头部所有的 include 语句之后添加如下 include 语句，将对话框类的头文件包含进来。

```
#include "Square.h"
```

(6) 增加菜单，显示对话框。

① 使用菜单编辑器增加一个“对话框”主菜单，并在其中添加“非模态对话框”菜单项，其 ID 为 ID_NOMODEDLG。

② 利用“类向导”工具在视图类中为 ID_NOMODEDLG 菜单项添加 COMMAND 消息的处理函数，在函数中添加以下代码。

```
void CLi55View::OnNomodedlg()
{
    if(m_pDlg)
       m_pDlg->SetActiveWindow();                    //激活非模态对话框
    else
    {
        //创建非模态对话框
        m_pDlg = new CSquare(this);                  //创建对话框对象
        m_pDlg->m_pParent = this;                    //对话框对象获取主窗口视图指针
        m_pDlg->Create(IDD_DIALOG1,this);            //创建对话框窗口
```

```
        m_pDlg->ShowWindow(SW_SHOW);                    //显示对话框
    }
}
```


(7) 在 Square.cpp 文件头部所有的 include 语句之后添加 include 语句,将视图类的头文件包含进来。

```
#include "Li5_5View.h"
```

(8) 利用“类向导”工具为对话框中的“确定”和“取消”按钮添加自己的处理函数,并添加代码。

```
void CSquare::OnOK()
{
    UpdateData(true);                                   //获取用户数据
    CClientDC dc(m_pParent);                            //创建指向主窗口视图的 CClientDC 对象
    dc.Rectangle(10,10,10+m_length,10+m_length);        //画正方形
}
void CSquare::OnCancel()
{
    if(m_pParent!=NULL)
    {
        m_pParent->m_pDlg=NULL;                         //表明对话框对象已经不存在了
        DestroyWindow();                                //删除对话框窗口
    }
}
```

(9) 利用“类向导”工具在对话框类 CSquare 中添加 PostNcDestroy 消息,并添加代码。

```
void CSquare::PostNcDestroy()
{
    CDialog::PostNcDestroy();
    delete this;                                        //删除对话框对象
}
```

(10) 编译、链接并运行程序,选择“对话框”|“非模态对话框”菜单命令,打开“输入边长”对话框,输入边长后单击“确定”按钮,运行结果如图 5.14 所示。

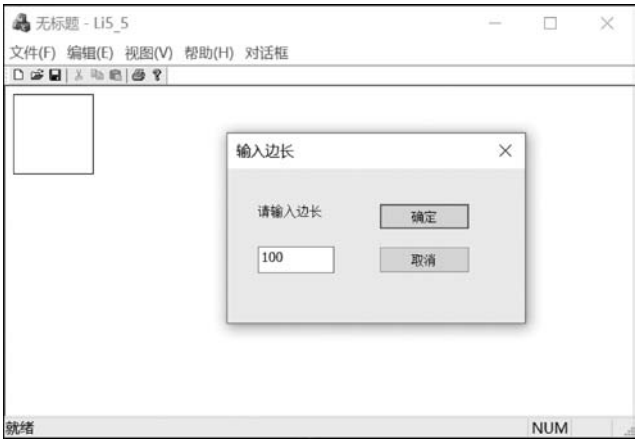

图 5.14　非模态对话框的运行结果

5.3.2 窗口对象的自动清除

一个 MFC 窗口对象包括两方面的内容：一是窗口对象封装的窗口，即地地道道的 Windows 窗口；二是窗口对象本身是一个 C++对象。为了区分，把前者简称为窗口，把后者称为窗口对象。如果要删除一个 MFC 窗口对象，应该先删除窗口对象封装的窗口，然后删除窗口对象本身。

删除窗口最直接的方法是调用 CWnd::DestroyWindow()或::DestroyWindow()，前者封装了后者的功能。前者不仅会调用后者，而且会使成员 m_hWnd 保存的 HWND 无效(NULL)。如果 DestroyWindow()删除的是一个父窗口或拥有者窗口，则该函数会先自动删除所有的子窗口或被拥有者，然后再删除父窗口或拥有者。一般情况下，在程序中不必直接调用 DestroyWindow()来删除窗口，因为 MFC 会自动调用 DestroyWindow()来删除窗口。例如，当用户退出应用程序时会产生 WM_CLOSE 消息，该消息会导致 MFC 自动调用 CWnd::DestroyWindow()来删除主框架窗口。

窗口对象本身的删除则根据对象创建方式的不同分为两种情况。在 MFC 编程中会使用大量的窗口对象，有些窗口对象以变量的形式嵌入在其他对象内或以局部变量的形式创建在堆栈上，有些则用 new 操作符创建在堆中。对于一个以变量形式创建的窗口对象，程序员不必关心它的删除问题，因为该对象的生存期总是有限的，若该对象是某个对象的成员变量，它会随着父对象的消失而消失；若该对象是一个局部变量，它会在函数返回时被清除。对于一个在堆中动态创建的窗口对象，其生存期却是任意长的。初学者在学习 C++编程时，对 new 操作符的使用往往不太踏实，因为用 new 在堆中创建对象就不能忘记用 delete 删除对象。读者在学习 MFC 编程时可能会产生这样的疑问：为什么有些程序用 new 创建了一个窗口对象，却未显式地用 delete 来删除它？问题的答案就是有些 MFC 窗口对象具有自动清除的功能。

如前面讲述非模态对话框时所提到的，当调用 CWnd::DestroyWindow()或::DestroyWindow()删除一个窗口时，被删除窗口的 PostNcDestroy()成员函数会被调用。默认的 PostNcDestroy()什么也不干，但有些 MFC 窗口类会覆盖该函数并在新版本的 PostNcDestroy()中调用 delete this 来删除对象，从而具有了自动清除的功能。此类窗口对象通常是用 new 操作符在堆中创建的，但程序员不必操心用 delete 操作符去删除它们，因为一旦调用 DestroyWindow()删除窗口，对应的窗口对象也会紧接着被删除。

不具有自动清除功能的窗口类如下，这些窗口对象通常是以变量的形式创建的，无自动清除功能。

- 所有标准的 Windows 控件类；
- 从 CWnd 类直接派生出来的子窗口对象(如用户定制的控件)；
- 切分窗口类 CSplitterWnd；
- 默认的控制条类(包括工具栏、状态栏和对话框)；
- 模态对话框类；

具有自动清除功能的窗口类如下，这些窗口对象通常是在堆中创建的。

- 主框架窗口类(直接或间接地从 CFrameWnd 类派生)；
- 视图类(直接或间接地从 CView 类派生)；

• 非模态对话框类。

读者在设计自己的派生窗口类时,可以根据窗口对象的创建方法来决定是否将窗口类设计成能够自动清除的。例如,对于一个非模态对话框,其对象是在堆中创建的,因此应该具有自动清除功能。

综上所述,对于 MFC 窗口类及其派生类,在程序中一般不必显式地删除窗口对象。也就是说,既不必调用 DestroyWindow()来删除窗口对象封装的窗口,也不必显式地用 delete 操作符来删除窗口对象本身。只要保证非自动清除的窗口对象是以变量的形式创建的,自动清除的窗口对象是在堆中创建的,MFC 的运行机制就可以保证窗口对象被彻底删除。

如果需要手工删除窗口对象,则应该先调用相应的函数(例如 CWnd::DestroyWindow())删除窗口,然后再删除窗口对象。对于以变量形式创建的窗口对象,窗口对象的删除是框架自动完成的。对于在堆中动态创建的非自动清除的窗口对象,必须在窗口被删除后显式地调用 delete 来删除对象(一般在拥有者或父窗口的析构函数中进行)。对于具有自动清除功能的窗口对象,只需调用 CWnd::DestroyWindow()即可删除窗口和窗口对象。注意,对于在堆中创建的窗口对象,不要在窗口还未关闭的情况下就用 delete 操作符来删除窗口对象。

在非模态对话框的 OnCancel()函数中可以不调用 CWnd::DestroyWindow(),取而代之的是调用 CWnd::ShowWindow(SW_HIDE)来隐藏对话框。在下次打开对话框时就不必调用 Create()成员函数了,只需调用 CWnd::ShowWindow(SW_SHOW)来显示对话框。这样做的好处在于对话框中的数据可以保存下来,供以后使用。由于拥有者窗口在被关闭时会调用 Window 删除每一个所属窗口,所以只要非模态对话框是自动清除的,就不必担心对话框对象的删除问题。

例 5.5 中设计的非模态对话框就具有自动清除功能,这里对部分函数做如下修改:

```
void CLi55View::OnNomodedlg()
{
    if(m_pDlg)
    {
         m_pDlg->SetActiveWindow();
         m_pDlg->ShowWindow(SW_SHOW);                    //新增显示对话框语句
    }
    else
    {
        m_pDlg = new CSquare(this);
        m_pDlg->m_pParent = this;
        m_pDlg->Create(IDD_DIALOG1,this);
        m_pDlg->ShowWindow(SW_SHOW);
    }
}
  void CSquare::OnCancel()
{
    if(m_pParent!= NULL)
    {
        CWnd::ShowWindow(SW_HIDE);                       //修改为隐藏对话框语句
    }
}
```

重新运行程序，发现可以保存上次运行时对话框中的数据。

5.4 属性页对话框

在设计较为复杂的对话框时经常会用到大量的控件，以至于在一个对话框中布置不了这些控件。用普通的对话框技术，这一问题很难解决。MFC 提供了对属性页对话框的支持，可以很好地解决上述问题。

属性页对话框实际上是一个包含多个子对话框的对话框，这些子对话框通常被称为页(Page)。每次只有一个页是可见的，在对话框的顶端有一行标签，用户通过单击这些标签可以切换到不同的页。显然属性页对话框可以容纳大量控件。例如，Visual C++的“类向导”窗口就是属性页对话框的典型应用。

为了支持属性页对话框，MFC 提供了 CPropertySheet 类和 CPropertyPage 类。前者代表属性页对话框，后者代表对话框中的某一页。CPropertySheet 类对象或其派生类对象中包含了若干 CPropertyPage 类对象。CPropertyPage 是 CDialog 类的派生类，而 CPropertySheet 是 CWnd 类的派生类。虽然 CPropertySheet 不是 CDialog 类的派生类，但使用 CPropertySheet 对象与使用 CDialog 对象的方法是类似的。

属性页对话框是一种特殊的对话框，和普通对话框相比，它们的设计与实现既有许多相似之处，又有一些不同的特点。下面通过一个实例来讲解设计一个属性页对话框的过程。

【例 5.6】 创建一个单文档的 MFC 应用程序 Li5_6，通过选择“对话框”|“属性页对话框”菜单命令打开如图 5.15 所示的对话框，当单击“确定”按钮后将在消息框中输出相关信息。

视频讲解

图 5.15 “属性页对话框”

其操作步骤如下：

(1) 使用“MFC 应用”项目模板创建一个项目名为 Li5_6 的单文档应用程序。

(2) 设计对话框资源。分别为各页创建对话框模板,每页的模板最好具有相同尺寸,如果尺寸不统一,则框架将根据尺寸最大的页来确定属性页对话框的大小。

① 创建如图 5.16 所示的对话框资源作为属性页的第一页,将该对话框的 ID 改为 IDD_PERSONAL,将“描述文字”改为“姓名”。

② 创建如图 5.17 所示的对话框资源作为属性页的第二页,将该对话框的 ID 改为 IDD_UNIT,将“描述文字”改为“工作单位”。

图 5.16 第一页属性页

图 5.17 第二页属性页

(3) 用“类向导”工具为每页创建新类,加入与控件对应的成员变量。

① 为属性页 IDD_PERSONAL 创建 CPropertyPage 类的派生类 CPersonalPage,并加入与编辑框对应的成员变量 m_name,其类型为 CString。

② 为属性页 IDD_ UNIT 创建 CPropertyPage 类的派生类 CUnitPage,并加入与编辑框对应的成员变量 m_work,其类型为 CString。

(4) 在 CLi5_6View 类的头文件开始处加入下列语句。

```
#include "PersonalPage.h"
#include "UnitPage.h"
```

(5) 增加菜单以打开属性页对话框。

① 使用菜单编辑器增加一个“对话框”主菜单,并在其中添加“属性页对话框”菜单项,其 ID 为 ID_PROPAGE。

② 利用“类向导”工具在视图类中为 ID_PROPAGE 菜单项添加 COMMAND 消息的处理函数,在函数中添加代码。

```
void CLi56View::OnPropage()
{
    CPropertySheet m_mysheet(L"属性页对话框");
    CPersonalPage PageFirst;
    CUnitPage PageSec;
    CString str = L"";
    m_mysheet.AddPage(&PageFirst);
    m_mysheet.AddPage(&PageSec);
    if(m_mysheet.DoModal() == IDOK)
    {
        str = str + PageFirst.m_name + "工作单位是" + PageSec.m_work;
        MessageBox(str);
    }
}
```

(6) 编译、链接并运行程序。选择“对话框”|“属性页对话框”菜单命令,弹出如图 5.15 所示的属性页对话框。将姓名改为“Mary”,将工作单位改为“清华大学出版社”,当单击“确定”按钮后,将在消息框中输出“Mary 工作单位是清华大学出版社”。

从上面的例子可以看出,对话框框架的创建过程与普通对话框基本相同,不同之处是还需将页对象加入 CPropertySheet 对象中。如果要创建的是模态对话框,应调用 CPropertySheet::DoModal()函数;如果想创建非模态对话框,则应该调用 CPropertySheet::Create()函数。

若从 CPropertySheet 类派生了一个新类,则应该将所有的页对象以成员变量的形式嵌入派生类中,并在派生类的构造函数中调用 CPropertySheet::AddPage()函数把各个页添加到对话框中。


视频讲解

【例 5.7】 使用 CPropertySheet 类的派生类对象创建与例 5.6 同样的属性页对话框。其操作步骤如下:

(1) 与例 5.6 一样完成前 3 步,应用程序的名称为 Li5_7。

(2) 创建 CPropertySheet 类的派生类 CProframeSheet。

(3) 在 CProframeSheet 类的头文件开始处加入以下语句。

```
#include "PersonalPage.h"
#include "UnitPage.h"
```

(4) 在 CLi5_7View.cpp 文件头部所有的 include 语句之后添加语句。

```
#include "ProframeSheet.h"
```

(5) 在 CProframeSheet.h 中加入语句。

```
CUnitPage m_unit;
CPersonalPage m_personal;
```

(6) 在 CProframeSheet 类的第一个参数字符串的构造函数中加入语句,以便把各个页添加到对话框中。

```
AddPage(&m_personal);
AddPage(&m_unit);
```

(7) 增加菜单,显示属性页对话框。

① 使用菜单编辑器增加一个“对话框”主菜单,并在其中添加“属性页对话框”菜单项,其 ID 为 ID_PROPAGE。

② 使用“类向导”工具在视图类中为 ID_PROPAGE 菜单项添加 COMMAND 消息的处理函数,在函数中添加代码。

```
void CLi57View::OnPropage()
{
    CProframeSheet m_mysheet(L"属性页对话框");
    CString str = L" ";
    if(m_mysheet.DoModal() == IDOK){
        Str = str + m_mysheet.m_personal.m_name + "工作单位是" + m_mysheet.m_unit.m_work;
    MessageBox(str);
}
```

(8) 编译、链接并运行程序,可以得到与例 5.6 相同的效果。

5.5 通用对话框

Windows 提供了一组标准用户界面的通用对话框,用户在程序中可以直接使用这些通用对话框,不必再设计对话框资源和对话框类,减少了大量的编程工作。为了在 MFC 应用程序中使用通用对话框,MFC 提供了封装这些通用对话框的类。表 5.3 列出了 MFC 中一些常用的通用对话框类,这些类都是从 CCommonDialog 类派生而来的,而 CCommonDialog 类又是 CDialog 类的派生类。

表 5.3 常用的通用对话框类

通用对话框类	说 明
CFileDialog	文件对话框,用于打开或保存文件
CColorDialog	"颜色"对话框,用于选择不同的颜色
CFontDialog	"字体"对话框,用于选择字体、字型、大小、效果及颜色
CPrintDialog	"打印"和"打印设置"对话框,用于打印和进行打印设置
CPageSetupDialog	"页面设置"对话框,用于设置和修改打印边距等
CFindReplaceDialog	"查找"和"替换"对话框,用于查找和替换文本字符串

在这些通用对话框中,"查找"和"替换"对话框是非模态对话框,其他对话框都是模态对话框,它们的使用方式由其所属类型来决定。下面分别介绍这些通用对话框类的构造函数及相应类的成员函数和数据成员及其使用。

5.5.1 CFileDialog 类

CFileDialog 类用于实现文件对话框,以支持文件的打开和保存操作。用户要打开或保存文件,就会和文件对话框"打交道",图 5.18 显示了一个标准的用于打开文件的"打开"对话框。使用"MFC 应用"项目模板建立的应用程序中自动加入了文件对话框,在"文件"菜单中选择"打开"或"另存为"菜单命令就会启动它们。

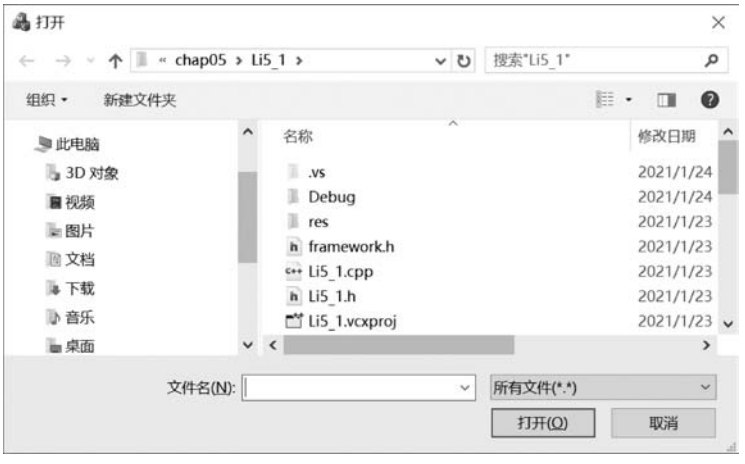

图 5.18 "打开"对话框

利用 CFileDialog 类使用文件对话框与一般模态对话框类似，首先声明一个 CFileDialog 类对象，这时会自动调用 CFileDialog 类的构造函数。该构造函数的原型为

```
CFileDialog(BOOL bOpenFileDialog,LPCTSTR lpszDefExt = NULL,LPCTSTR lpszFileName = NULL,DWORD
dwFlags = OFN_HIDEREADONLY
|OFN_OVERWRITEPROMPT,LPCTSTR lpszFilter = NULL,CWnd * pParentWnd = NULL)
```

参数 bOpenFileDialog 是一个标记位，其值如果为 TRUE，那么将构造“打开”对话框；如果为 FALSE，那么将构造“另存为”对话框。

参数 lpszDefExt 为默认的文件扩展名。如果用户没有在“文件名”编辑框中输入扩展名，则由 lpszDefExt 所指定的扩展名将自动附加在文件名后。

参数 lpszFileName 是出现在“文件名”编辑框中的初始文件名。如果该参数的值为 NULL，则不显示初始文件名。

参数 dwFlags 由一个或多个标志组成。其中，OFN_HIDEREADONLY 将隐藏只读文件，OFN_ALLOWMULTISELECT 将允许在选择时与 Shift 键或 Ctrl 键配合以选择多个文件。

参数 lpszFilter 指定文件过滤器，用于确定显示在文件列表中的文件类型。每个过滤器都是一个字符串对，第一个字符串描述过滤器，第二个字符串是用户过滤文件的扩展名，多个扩展名要用分号(;)作为分界符，两个字符串之间用管道符号(|)分隔。整个字符串以两个管道符号(||)结束，可以用 CString 对象值作为该参数的值。例如，以下字符串就是一个描述只在文件列表框中显示文本文件(*.txt)和 Word 文件(*.doc)的过滤器。

```
CFileDialog dlg(TRUE,"bmp","*.bmp", OFN_HIDEREADONLY
|OFN_ALLOWMULTISELECT,"文本文件(*.txt)|*.txt|Word 文件(*.doc)|*.doc||");
```

然后调用 CFileDialog::DoModal()来启动对话框。若调用 DoModal()函数打开对话框返回的是 IDOK，那么可以用表 5.4 列出的 CFileDialog 类的成员函数来获取与所选文件有关的信息。

表 5.4　CFileDialog 类的常用成员函数

类的成员函数	功 能 说 明
GetPathName()	获取当前所选文件的全路径
GetFileName()	获取当前所选文件的文件名
GetFileExt()	获取当前所选文件的扩展名
GetFileTitle()	获取当前所选文件的标题
GetNextPathName()	获取所选择的下一个文件的全路径
GetStartPosition()	得到文件列表的第一个元素的位置


视频讲解

【例 5.8】 创建一个单文档的 MFC 应用程序 Li5_8，利用文件对话框打开一个文本文件。

其操作步骤如下：

(1) 使用“MFC 应用”项目模板创建一个项目名为 Li5_8 的单文档应用程序。

(2) 使用“类向导”工具在视图类 CLi58View 中添加 WM_LBUTTONDOWN 消息，并添加代码。

```
void CLi58View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CString FilePathName;
    CFileDialog dlg(TRUE,L"txt",L"*.txt",
                OFN_HIDEREADONLY|OFN_ALLOWMULTISELECT,
        L"文本文件(*.txt)|*.txt|Word 文件(*.doc)|*.doc||");
if(dlg.DoModal() == IDOK)
        FilePathName = dlg.GetPathName();                          //得到文件路径
ShellExecute(NULL,L"open",FilePathName,NULL,NULL,SW_RESTORE);      //打开文件
    CView::OnLButtonDown(nFlags, point);
}
```

(3) 编译、链接并运行程序。在视图窗口中单击,运行结果如图 5.19 所示。

图 5.19　使用文件对话框

5.5.2　CColorDialog 类

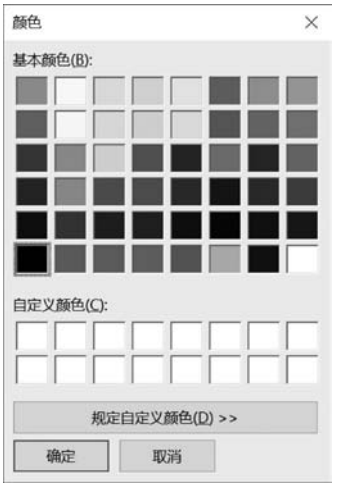

图 5.20　“颜色”对话框

CColorDialog 类用于实现“颜色”对话框,如图 5.20 所示。在 Windows 的画板程序中,如果用户双击“颜色面板”中的某种颜色,就会显示一个“颜色”对话框让用户选择颜色。

利用 CColorDialog 类使用“颜色”对话框也与一般模态对话框类似。首先构建一个 CColorDialog 对象,然后调用 CColorDialog::DoModal()来启动对话框。CColorDialog 类的构造函数的原型为

```
CColorDialog(COLORREF clrInit = 0, DWORD dwFlags = 0,
CWnd* pParentWnd = NULL);
```

其中,参数 clrInit 用来指定初始的颜色选择,dwFlags 用来

设置对话框，pParentWnd 用来指定对话框的父窗口或拥有者窗口。

根据 DoModal()返回的是 IDOK 还是 IDCANCEL 可以知道用户是否确认了对颜色的选择。若调用 DoModal()函数打开对话框返回的是 IDOK，调用 CColorDialog::GetColor()可以返回一个 COLORREF 类型的结果来表示在对话框中选择的颜色。COLORREF 是一个 32 位的值，用来说明一个 RGB 颜色。GetColor()返回的 COLORREF 的格式是 0x00bbggrr，即低位 3 个字节分别包含了蓝、绿、红 3 种颜色的强度。


视频讲解

【例 5.9】 创建一个单文档的 MFC 应用程序 Li5_9，利用“颜色”对话框选择颜色，并在视图区中画一个该颜色的矩形。

其操作步骤如下：

(1) 使用“MFC 应用”项目模板创建一个名为 Li5_9 的单文档应用程序。

(2) 给视图类 CLi59View 添加一个成员变量 m_cc，用来保存选择的颜色，其类型为 COLORREF。

(3) 使用“类向导”工具在视图类 CLi59View 中添加 WM_LBUTTONDOWN 消息，并添加代码。

```
void CLi59View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CColorDialog dlg;                          //构建一个 CColorDialog 对象
    if(dlg.DoModal() == IDOK)
    {
        CPen newpen, * oldpen;
        CClientDC dc(this);
        m_cc = dlg.GetColor();                 //得到在对话框中选择的颜色
        //用得到的颜色画矩形
        newpen.CreatePen(PS_SOLID,2,m_cc);
        oldpen = dc.SelectObject(&newpen);
        dc.Rectangle(100,100,200,200);
        dc.SelectObject(oldpen);
    }
    CView::OnLButtonDown(nFlags, point);
}
```

(4) 编译、链接并运行程序。在视图窗口中单击，弹出“颜色”对话框，选择颜色后单击“确定”按钮，在视图窗口中出现一个选定颜色的矩形，如图 5.21 所示。

5.5.3 CFontDialog 类

CFontDialog 类支持“字体”对话框，用来让用户选择字体。利用 CFontDialog 类使用“字体”对话框的过程是首先构建一个 CFontDialog 对象，然后调用 CFontDialog::DoModal()来启动对话框。

CFontDialog 类的构造函数的原型为

```
CFontDialog(LPLOGFONT lplfInitial = NULL,
DWORD dwFlags = CF_EFFECTS |CF_SCREENFONTS,
CDC * pdcPrinter = NULL, CWnd * pParentWnd = NULL);
```

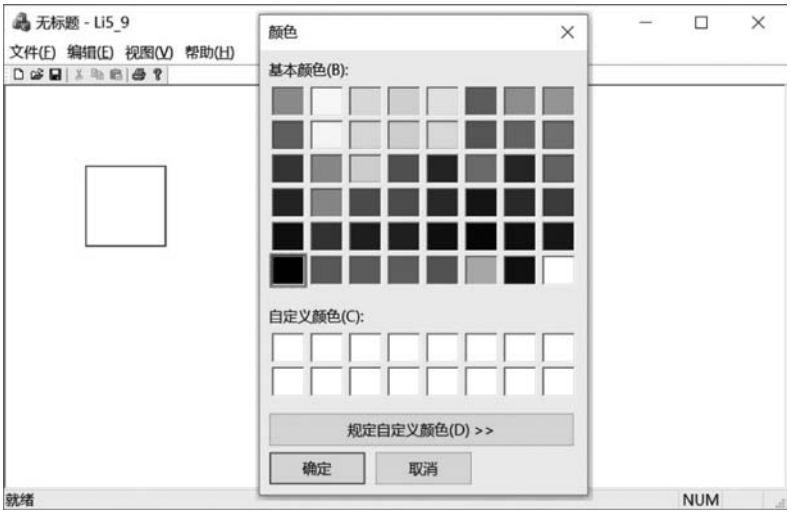

图 5.21　使用“颜色”对话框

其中，参数 lplfInitial 指向一个 LOGFONG 结构，用来初始化对话框中的字体设置；dwFlags 用于设置对话框；pdcPrinter 指向一个代表打印机的 CDC 对象，若设置该参数，则选择的字体就为打印机所用；pParentWnd 用于指定对话框的父窗口或拥有者窗口。

CFontDialog 类的数据成员 m_cf 用于自定义 CFontDialog 对象的结果。若 DoModal()返回 IDOK，那么可以用表 5.5 列出的 CFontDialog 类的成员函数来获得所选字体的信息。

表 5.5　CFontDialog 类的常用成员函数

类的成员函数	功　能
GetCurrentFont()	用来获得所选字体的属性。该函数有一个参数，该参数是指向 LOGFONT 结构的指针，函数将所选字体的各种属性写入这个 LOGFONT 结构中
GetFaceName()	返回一个包含所选字体名称的 CString 对象
GetStyleName()	返回一个包含所选字体风格的 CString 对象
GetSize()	返回所选字体的尺寸(以 10 个像素为单位)
GetColor()	返回一个含有所选字体颜色的 COLORREF 型值
GetWeight()	返回所选字体的权值
IsStrikeOut()	若用户选择了空心效果，返回 TRUE，否则返回 FALSE
IsUnderline()	若用户选择了下画线效果，返回 TRUE，否则返回 FALSE
IsBold()	若用户选择了黑体风格，返回 TRUE，否则返回 FALSE
IsItalic()	若用户选择了斜体风格，返回 TRUE，否则返回 FALSE


视频讲解

【例 5.10】 创建一个单文档的 MFC 应用程序 Li5_10，利用“字体”对话框选择字体的属性，并在视图区中以该属性显示文本信息。

其操作步骤如下：

(1) 使用“MFC 应用”项目模板创建一个项目名为 Li5_10 的单文档应用程序。

(2) 在视图类 CLi510View 中添加一个成员变量 m_cc，用来保存选择的颜色，其类型为 COLORREF。

(3) 在视图类 CLi510View 中添加一个成员变量 m_font，用来保存选择的字体，其类型

为 LOGFONT。

(4) 使用“类向导”工具在视图类 CLi510View 中添加 WM_LBUTTONDOWN 消息，并添加代码。

```
void CLi510View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CFontDialog dlg;
    if(dlg.DoModal() == IDOK)
        {
//得到在对话框中选择的字体
        memcpy(&m_cf,dlg.m_cf.lpLogFont,sizeof(LOGFONT));
        m_cc = dlg.GetColor();                          //得到在对话框中选择的颜色
        CFont font;
        font.CreateFontIndirect(&m_cf);                 //创建字体
        CClientDC dc(this);
        CFont * def_font = dc.SelectObject(&font);      //选择字体
        dc.SetTextColor(m_cc);                          //选择字体颜色
        dc.TextOut(5,5,"Hello C++6.0");
        dc.SelectObject(def_font);
        font.DeleteObject();
    }
    CView::OnLButtonDown(nFlags, point);
}
```

(5) 编译、链接并运行程序。在视图窗口中单击，弹出“字体”对话框，选择字体的相关属性后单击“确定”按钮，效果如图 5.22 所示。

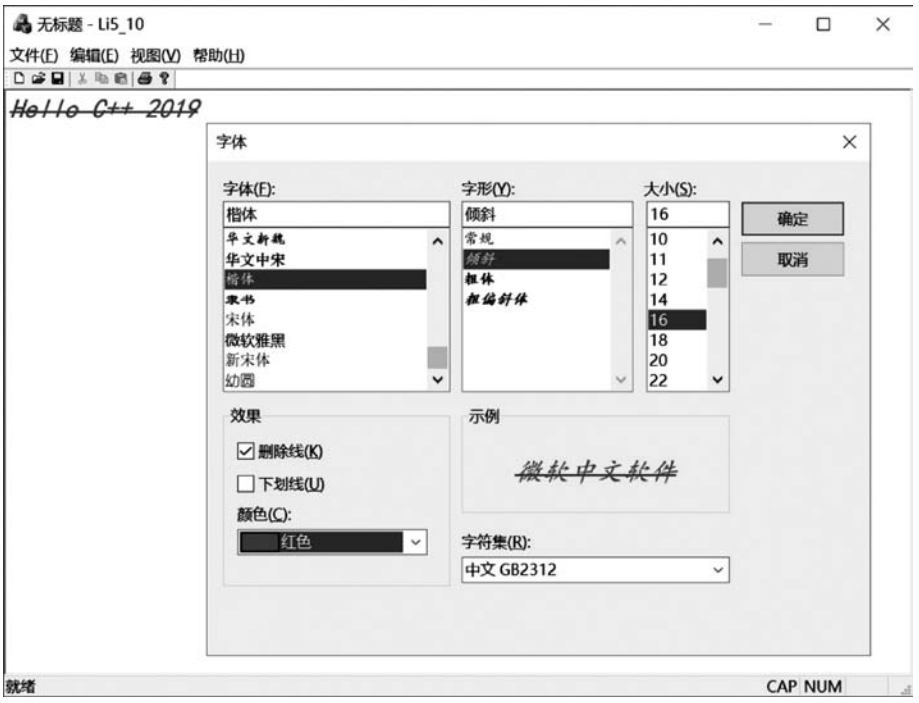

图 5.22　使用“字体”对话框

5.5.4 CPrintDialog 类和 CPageSetupDialog 类

CPrintDialog 类支持“打印”和“打印设置”对话框，用户通过这两个对话框可以进行与打印有关的设置。图 5.23 显示了一个“打印”对话框，图 5.24 显示了一个“打印设置”对话框。使用默认配置的“MFC 应用”项目模板建立的程序支持“打印”对话框和“打印设置”对话框，用户可以在“文件”菜单中启动它们。

图 5.23 “打印”对话框

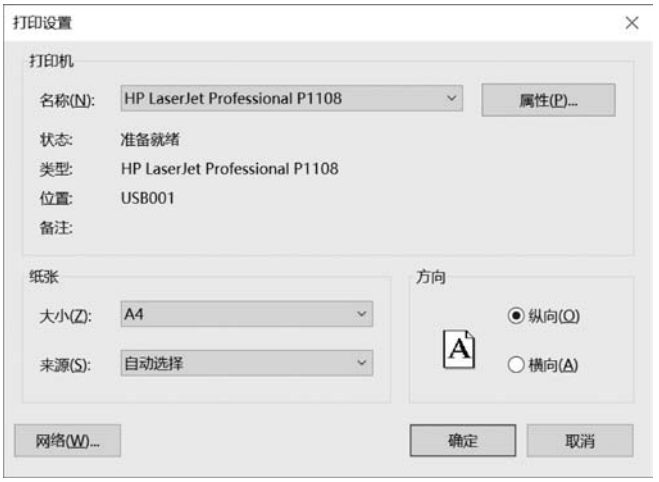

图 5.24 “打印设置”对话框

CPageSetupDialog 类封装了标准的“页面设置”对话框，使得用户可以设置和修改打印边距等。

使用这两种对话框的过程与使用前 3 种对话框类似，只是在成员函数上有所不同，在此不再详细介绍。CPrintDialog 类和 CPageSetupDialog 类的构造函数和成员函数请参阅 MSDN 文档。图 5.25 是使用 CPageSetupDialog 类打开的“页面设置”对话框。

图 5.25 “页面设置”对话框

5.5.5 CFindReplaceDialog 类

CFindReplaceDialog 类用于实现“查找”对话框和“替换”对话框。这两个对话框都是非模态对话框,用于在正文中查找和替换指定的字符串。图 5.26 显示了一个“查找”对话框,图 5.27 显示了一个“替换”对话框。

图 5.26 “查找”对话框

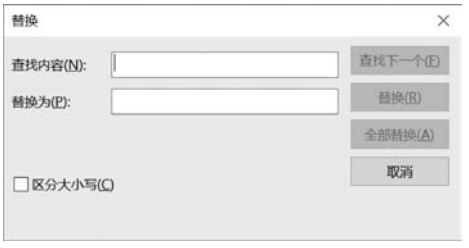

图 5.27 “替换”对话框

在 MFC 类库中用于生成文本编辑器(例如 Windows 附带的记事本)的 CEditView 类自动实现了“查找”对话框和“替换”对话框的功能。虽然“MFC 应用”项目模板并未提供相应的菜单命令,但可以在“编辑”菜单中加入“查找”和“替换”两项,令其 ID 分别为 ID_EDIT_FIND 和 ID_EDIT_REPLACE,则“查找”对话框和“替换”对话框的功能就可以实现了。一般很少直接用 CFindReplaceDialog 类来使用“查找”对话框和“替换”对话框。

视频讲解

5.6 应 用 实 例

5.6.1 实例简介

制作一个简单的计算器,实现加、减、乘、除、求倒数和平方根的混合运算,并能进行清屏

及倒退操作。

5.6.2 创建过程

1. 创建 MFC 应用程序框架

使用“MFC 应用”项目模板生成一个基于对话框的应用程序 Calculator，并将主窗口对话框的 Caption 改为 Calculator。

2. 编辑对话框

打开对话框编辑器，在对话框设计模板中添加如图 5.28 所示的控件，并对各个控件进行属性设置，如表 5.6 所示。

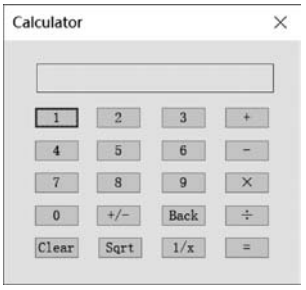

图 5.28 对话框控件的布局

在 Calculator 窗口中共包含 21 个控件，其中一个为编辑框，20 个为按钮。将编辑框设置为只读，不能接收输入；＋、－、×、÷为操作按钮；＋/－、Sqrt、1/x 分别为取负、求平方根及求倒数按钮；Back 为倒退按钮，用于删除错误的数字输入；Clear 为清除按钮，用于重新开始新的运算；单击“＝”按钮，则在编辑框中显示最后的计算结果。

表 5.6 控件 ID 和“描述文字”的设置

控件类型	ID	描述文字	控件类型	ID	描述文字
编辑框	IDC_EDIT_PUTOUT		按钮	IDC_ADD	＋
按钮	IDC_NUMBER1	1	按钮	IDC_SUBTRACT	－
按钮	IDC_NUMBER2	2	按钮	IDC_MULTIPLY	×
按钮	IDC_NUMBER3	3	按钮	IDC_DIVIDE	÷
按钮	IDC_NUMBER4	4	按钮	IDC_RESULT	＝
按钮	IDC_NUMBER5	5	按钮	IDC_MINUS	＋/－
按钮	IDC_NUMBER6	6	按钮	IDC_BACK	Back
按钮	IDC_NUMBER7	7	按钮	IDC_CLEAR	Clear
按钮	IDC_NUMBER8	8	按钮	IDC_SQRT	Sqrt
按钮	IDC_NUMBER9	9	按钮	IDC_RECIPROCAL	1/x
按钮	IDC_NUMBER0	0			

3. 添加成员变量

(1) 利用“类向导”工具为编辑框在对话框类 CCalculatorDlg 中添加 double 型成员变量 m_result。

(2) 为 CCalculatorDlg 类添加两个 int 型变量 m_OperationCount、m_NumberCount，一个 double 型数组 m_number[15]和一个 int 型数组 m_Operation[15]。变量 m_OperationCount 存放输入的加、减、乘、除 4 种运算符的顺序号，m_NumberCount 存放输入的操作数的顺序号；数组 m_number[15]存放输入的操作数，m_Operation[15]存放输入的操作符。

4. 添加消息映射及成员函数

(1) 手工加入 ON_COMMAND_RANGE 命令消息映射，处理分配给一系列相邻编号的命令 ID。

① 在 CalculatorDlg.h 头文件中声明消息映射函数。

```
afx_msg void OnNumberKey(UINT nID);
afx_msg void OnOperationKey(UINT nID);
```

其中，函数 OnNumberKey()用来响应数字按钮的单击操作，OnOperationKey()用来响应操作符按钮的单击操作。

② 在 CalculatorDlg.cpp 实现文件的消息映射表中加入 ON_COMMAND_RANGE 命令消息。

```
ON_COMMAND_RANGE(IDC_NUMBER1,IDC_NUMBER0,OnNumberKey)
ON_COMMAND_RANGE(IDC_MINUS,IDC_RESULT,OnOperationKey)
```

(2) 在 CalculatorDlg.cpp 实现文件中加入消息处理函数。

```
void CCalculatorDlg::OnNumberKey(UINT nID)
{//处理单击数字按钮操作,记录输入的操作数
     int n = 0;
     switch(nID)                    //根据单击的数字键 ID 记录输入数字
     {
       case IDC_NUMBER1:
            n = 1;
            break;
       case IDC_NUMBER2:
            n = 2;
            break;
       case IDC_NUMBER3:
            n = 3;
            break;
        case IDC_NUMBER4:
             n = 4;
             break;
        case IDC_NUMBER5:
             n = 5;
             break;
        case IDC_NUMBER6:
             n = 6;
             break;
        case IDC_NUMBER7:
             n = 7;
             break;
        case IDC_NUMBER8:
             n = 8;
             break;
        case IDC_NUMBER9:
             n = 9;
             break;
        case IDC_NUMBER0:
             n = 0;
             break;
    }
```

```
        //计算操作数
        m_number[m_NumberCount] = m_number[m_NumberCount] * 10 + n;
        m_result = m_number[m_NumberCount];
        UpdateData(false);              //在编辑框中显示操作数
}
void CCalculatorDlg::OnOperationKey(UINT nID)
{//处理单击操作符按钮操作,记录输入的操作符
    int i;
    switch(nID)                         //根据单击的操作键 ID 记录输入的加、减、乘、除操
    {                                   //作符,处理取负、求平方根、求倒数、倒退及清屏操作
        case IDC_ADD:
            m_Operation[m_OperationCount] = 1;
            break;
        case IDC_SUBTRACT:
            m_Operation[m_OperationCount] = 2;
            break;
        case IDC_MULTIPLY:
            m_Operation[m_OperationCount] = 3;
            break;
        case IDC_DIVIDE:
            m_Operation[m_OperationCount] = 4;
            break;
        case IDC_MINUS:                 //取负
            m_number[m_NumberCount] = -m_number[m_NumberCount];
            break;
        case IDC_SQRT:                  //求平方根
            m_number[m_NumberCount] = sqrt(m_number[m_NumberCount]);
            break;
        case IDC_RECIPROCAL:            //求倒数
            m_number[m_NumberCount] = (double)1/m_number[m_NumberCount];
            break;
        case IDC_BACK:                  //倒退
            m_number[m_NumberCount] = (int)m_number[m_NumberCount]/10;
            m_result = m_number[m_NumberCount];
            UpdateData(false);
            break;
        case IDC_CLEAR:                 //清屏
            for(i = 1;i < 11;i++)
            {
               m_number[i] = 0;
               m_Operation[m_OperationCount] = 999;
               m_NumberCount = 1;
               m_OperationCount = 1;
               m_result = 0;
               UpdateData(false);
            }
            break;
        case IDC_RESULT:                //计算最后结果
            cal();
            break;
    }
```

```
    if(m_Operation[m_OperationCount]< 5)
    {
        m_NumberCount++;
        m_OperationCount++;
    }
}
```

(3) 在对话框类 CalculatorDlg 中添加 void 型成员函数 cal(),并在 CalculatorDlg.cpp 实现文件前加上包含语句#include "math.h"。

```
void CCalculatorDlg::cal()
{
    for(int i = 1;i < 15;i++)
        switch(m_Operation[i])      //先处理乘、除运算
        {
          case 3:
                m_number[i] = m_number[i + 1] = m_number[i] * m_number[i + 1];
                break;
           case 4:
                m_number[i] = m_number[i + 1]
                            = (double)m_number[i]/m_number[i + 1];
                break;
        }
    m_result = m_number[1];
    for(i = 1;i < 15;i++)             //处理加、减运算,计算最后结果
        if(m_Operation[i] == 1)
            m_result = m_result + m_number[i + 1];
        else if(m_Operation[i] == 2)
            m_result = m_result - m_number[i + 1];
    UpdateData(false);                //在编辑框中显示最后结果
}
```

5. 成员变量的初始化

代码如下:

```
CCalculatorDlg::CCalculatorDlg(CWnd * pParent /* = NULL */)
    : CDialog(CCalculatorDlg::IDD, pParent)
{
    …
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_NumberCount = 1;
    m_OperationCount = 1;
    for(int i = 0;i < 15;i++)
    {
        m_number[i] = 0;
        m_Operation[i] = 999;
    }
}
```

编译、链接并运行程序。该应用程序未能实现乘、除运算的连续操作,请读者自行完善。

习　　题

1. 填空题

(1) 对话框的主要功能是_____和_____。

(2) 从对话框的工作方式看,对话框可分为_____和_____两种类型。

(3) 对话框主要由_____与_____两部分组成。

(4) 使用_____函数可以创建模态对话框,使用_____函数可以创建非模态对话框。

(5) 为了支持属性页对话框,MFC 提供了_____类和_____类。

2. 选择题

(1) 对话框的功能被封装在(　　)类中。

A. CWnd　　B. CDialog　　C. CObject　　D. CCmdTarget

(2) (　　)是非模态对话框。

A. “查找”对话框　　B. “字体”对话框

C. “打开”对话框　　D. “颜色”对话框

(3) 要将模态对话框在屏幕上显示需要用到(　　)函数。

A. Create()　　B. DoModal()　　C. OnOK()　　D. 构造

(4) 通常将对话框的初始化工作放在(　　)函数中进行。

A. OnOK()　　B. OnCancel()　　C. OnInitDialog()　　D. DoModal()

(5) 使用(　　)通用对话框类可以打开文件。

A. CFileDialog　　B. CColorDialog　　C. CPrintDialog　　D. CFontDialog

3. 简答题

(1) 简述创建和使用模态对话框的主要步骤。

(2) 如何向对话框模板资源添加控件? 如何添加与控件关联的成员变量?

(3) 什么是 DDX 和 DDV? 在编程时如何使用 MFC 提供的 DDX 功能?

(4) 简述创建属性页对话框的主要步骤。

4. 操作题

(1) 编写一个 SDI 应用程序,在执行某菜单命令时打开一个模态对话框,通过该对话框输入一对坐标值,单击 OK 按钮在视图区中的该坐标位置显示自己的姓名。

(2) 编写一个 SDI 应用程序,采用非模态对话框的方式完成与第(1)题同样的功能。

(3) 编写一个单文档的应用程序,为该应用程序添加两个按钮到工具栏中,单击第一个按钮,利用文件对话框打开一个.doc 文件; 单击第二个按钮,利用“颜色”对话框选择颜色,并在视图区中画一个该颜色的矩形。