从零开始，手把手教会你用Visual C++编程

本书特色:

基础知识→核心技术→典型应用→综合练习→项目实践
107个典型实例、70个练习题、2个项目开发案例

超值、大容量DVD

17小时多媒体视频教学
本书源代码、本书教学PPT
赠送40小时相关知识多媒体视频

本书技术支持

QQ群：21948169
论坛：http://www.rzchina.net

•从零开始学编程•

从零开始学 Visual C++

DVD

时多媒体教学视频

◎ 刘雪洁 刘永纯 等编著

循序渐进・由浅入深 内容充实・层次清楚 实例丰富・步骤清晰 对比讲解・理解深刻 习题指导・巩固学习 案例精讲・深入剖析

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

从零开始学Visual C++

本书涵盖的内容

- C++基本语法
- 熟悉开发环境
- 对话框
- 单文档应用程序
- 切分窗口
- 文件的读写、查找、序列化
- DataGrid控件
- 五子棋游戏
- 面向对象程序设计
- 常用控件
- GDI图形编程
- 视图风格
- 多文档应用程序
- ADO访问数据库
- OpenGL三维编程
- 公交换乘软件

从零开始学编程系列

上架建议：Visual C++

ISBN 978-7-121-12247-7

9 787121 122477 >

定价：39.80元（含DVD光盘1张）

策划编辑：胡辛征
责任编辑：高洪霞
封面设计：李　玲

•从零开始学编程•

从零开始学 Visual C++

◎ 刘雪洁 刘永纯 等编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

要想学好 Visual C++开发，一本适合自己的入门书是非常重要的。本书充分考虑 Visual C++的难度，合理安排章节，由浅入深，通过生动的范例程序和详细的代码注释，带领读者掌握 Visual C++软件开发的技巧。

本书共分 16 章，由浅入深，循序渐进地介绍了 Visual C++编程的各个知识点。本书共分为 5 篇，内容包括 C++基本语法、面向对象程序设计、Visual C++开发环境、常用控件、对话框、GDI 图形编程、单文档应用程序、视图风格、切分窗口、多文档应用程序、文件编程、数据库编程、DataGrid 控件、OpenGL 三维编程、五子棋游戏、公交换乘软件等。

本书配 DVD 光盘 1 张，内容为本书的实例文件和作者专门为本书录制的全程多媒体语音教学视频。

本书内容全面，论述翔实，适合 Visual C++的初学者，也可作为大、中专院校师生的培训教材，对于 Visual C++爱好者，本书也有很大的参考价值。

图书在版编目（CIP）数据

从零开始学 Visual C++ / 刘雪洁等编著. —北京：电子工业出版社，2011.2
（从零开始学编程）
ISBN 978-7-121-12247-7

Ⅰ. ①从… Ⅱ. ①刘… Ⅲ. ①C 语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2010）第 220700 号

责任编辑：高洪霞
印　　刷：北京中新伟业印刷有限公司
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：20.25　字数：488 千字
印　　次：2011 年 2 月第 1 次印刷
印　　数：5000 册　定价：39.80 元（含 DVD 光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前言

C++有着极美好的未来，用它你能写出伟大的代码。

——Bjarne Stroustrup（C++之父）

Visual C++是 Windows 下的桌面软件开发利器，结合 C++语言的高效灵活和 MFC 框架的强大实用，开发者可以尽显才智，实现各类特定功能，如自己电脑桌面上的软件，大部分都是使用 Visual C++开发实现的。在学习 Visual C++的开发过程中，可以深入了解 Windows 程序的实现原理，为成为一名高级软件工程师打下坚实基础。

但 MFC 框架的复杂也是公认的，号称“最难学的开发框架”，事实上没有什么东西是好学的，Visual C++之难在于它将所有可用资源全部罗列出来，初学时会茫然无措，不知从何处下手，但一旦学成之后，就身怀绝技，从此迈入高手行列。就个人发展而言，无须追赶技术潮流，如今 IT 技术发展之快令人望尘莫及，追也是追不上的，最紧要的事是练好内功，以不变应万变。Visual C++可谓开发工具中的“少林七十二绝技”，是修炼内功最好的一门技术，即便以后工作中用不上，也能从中受益匪浅。言尽于此，还望读者静下心来，认真读完这本书，相信会大有收获。

本书的特点

不可否认，Visual C++确实有一定的难度，事实上，难学的东西太多，没有最难，只有更难，比如高等数学、大学物理，但总有人能够把这些东西学得很精通，会者不难，只要坚持学习并找到科学的学习方法，以各位的聪明才智，相信没有搞不定的东西。

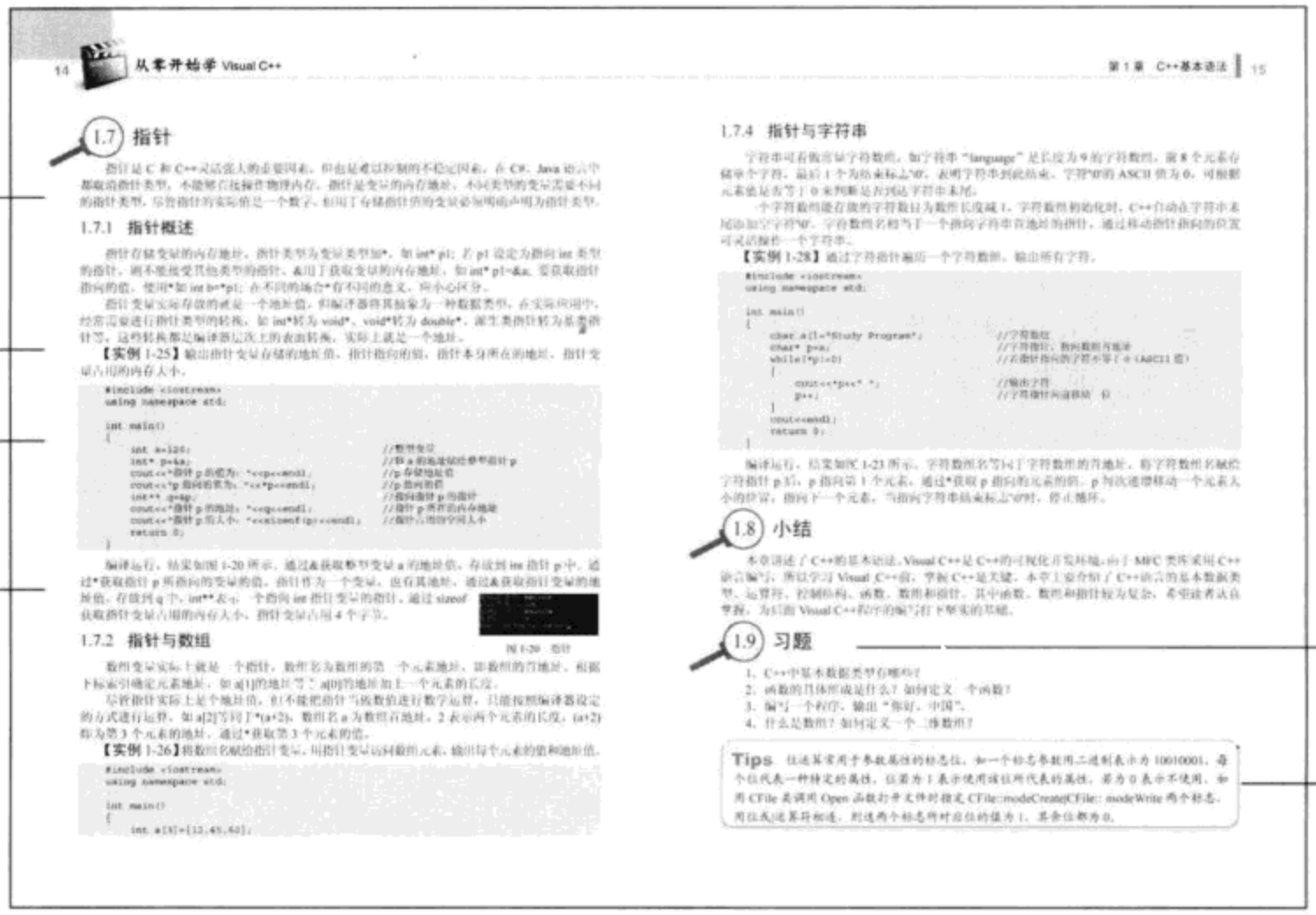
Visual C++之所以难学，最主要的原因在于没有好的教材，好的教材应该包含生动的文字描述信息和丰富实用的实例程序，不少教材给出大段代码后，却少有注释，或者提供的实例程序过于简单，没有实际意义，笔者深知读者需要什么，怎样讲解最利于读者学习，简而言之，一切为读者考虑。本书的主要特点如下：

- 讲解内容全面，涵盖了 Visual C++开发的各项基础知识，如对话框、单文档、多文档三大框架，以及文件、数据库等工作中常用的开发技术，集中篇幅深入讲解这些最实用的内容。
- 实例丰富，绝大部分章节以实例程序为核心，在实际应用中讲解要介绍的内容，如介绍树控件时，将树控件常用的添加、修改、删除、选中、勾选等功能集中到一个实例程序中，便于读者掌握最实用的技术。
- 代码注释丰富，对于实例代码，提供详细的代码注释，通过注释，读者可以了解每一句代码的意义，无须担心看不懂代码。
- 通俗易懂，将很多复杂的原理用通俗的话讲出来，原理就简单了，方便读者了解。

① 知识点介绍 准确、清晰是其显著特点，一般放在每一节开始位置，让零基础的读者了解相关概念，顺利入门。

② 实例 书中出现的完整实例，以章节顺序编号，便于检索和循序渐进地学习、实践，放在每节知识点介绍之后。

③ 实例代码 与实例编号对应，层次清楚、语句简洁、注释丰富，体现了代码优美的原则，有利于读者养成良好的代码编写习惯。



④ 运行结果 对实例给出运行结果和对应图示，帮助读者更直观地理解实例代码。

⑤ 习题 每章最后提供专门的测试习题，供读者检验所学知识是否牢固掌握。

⑥ 贴心的提示 为了便于读者阅读，全书还穿插着一些技巧、提示等小贴士，体例约定如下：

提示：通常是一些贴心的提醒，让读者加深印象或提供建议，或者解决问题的方法。

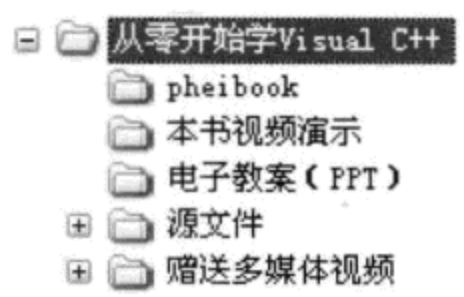
注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关信息。

警告：对操作不当或理解偏差将会造成的灾难性后果做警示，以加深读者印象。

经作者多年的培训和授课证明，以上讲解方式是最适合初学者学习的方式，读者按照这种方式，会非常轻松、顺利地掌握本书知识。

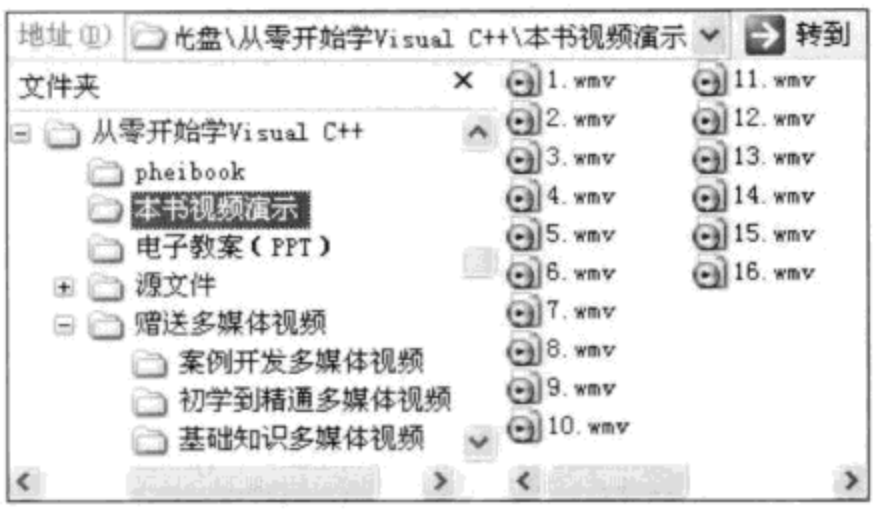
2. 实用超值的 DVD 光盘

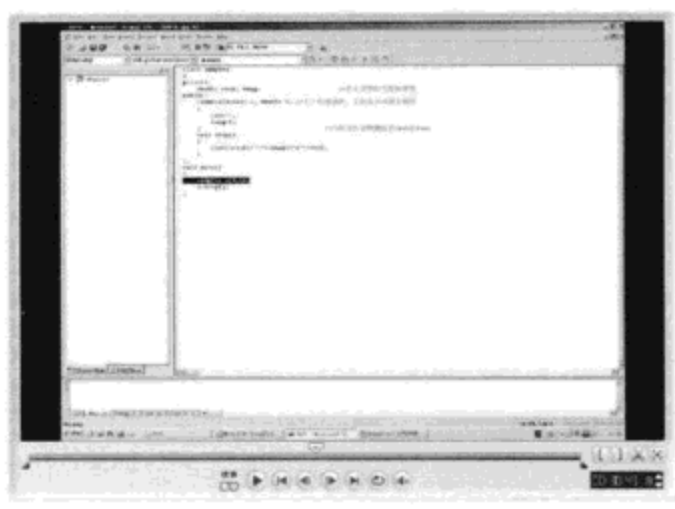
为了帮助读者比较直观地学习，本书附赠 DVD 光盘，内容包括多媒体视频、电子教案（PPT）和实例源代码等。



- **多媒体视频**

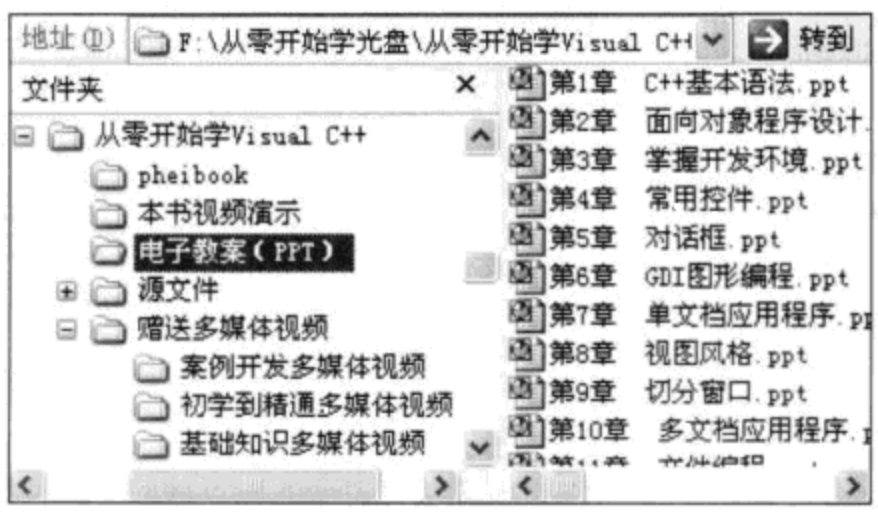
配有长达 17 小时手把手教学视频，讲解关键知识点界面操作和书中的一些综合练习题。作者亲自配音、演示，手把手教会读者使用。





- 电子教案（PPT）

本书可以作为高校相关课程的教材或课外辅导书，所以笔者特别为本书制作了电子教案（PPT），以方便老师教学使用。



3. 提供完善的技术支持

本书提供了论坛：http://www.rzchina.net，读者可以在上面提问交流。另外，论坛上还有一些小的教程、视频动画和各种技术文章，可帮助读者提高开发水平。

4. 丰富的额外素材下载

相关的开发素材文件，在 www.broadview.com.cn 提供下载。

适合阅读本书的读者

- 本书适合 Visual C++开发自学者；
- 软件开发人员；
- 大中专院校相关专业的学生。

本书作者

本书主要由吉林大学计算机科学与技术学院的刘雪洁和黑龙江省直属机关党校的刘永纯编写。其中，第 1~5 章和第 11~16 章主要由刘雪洁编写，第 6~10 章主要由刘永纯编写。其他参与本书编写的人员有昊燃、曾光、张双、朱照华、黄永湛、孟祥嵩、张贺军、李勇、关涛、王岩、李晓白、魏星、刘蕾。在此一并表示感谢！

目　录

第 1 篇　Visual C++基础入门

第 2 篇 可视化编程

第 3 篇　Visual C++的应用

第 4 篇 Visual C++编程

第 5 篇 案例篇

第 1 篇　Visual C++基础入门

第 1 章　C++基本语法

程序设计语言经历了四十多年的发展，从传统的汇编、C、C++、Visual Basic 到现今的 C#、Java，以及其他各类编程语言共有百余种，数目之多令人眼花缭乱。C++作为众多编程语言中的经典之作，于 20 世纪 80 年代由 AT&T 贝尔实验室的 Bjarne Stroustrup 设计而成。C++兼容 C 语言，并具备面向对象的能力，是一种适合编写大型软件的高级语言。Visual C++是 C++的可视化开发环境，其核心 MFC 类库采用 C++语言编写，C++的面向对象特性充分体现在 MFC 类库里，在学习 Visual C++之前，必须掌握基本的 C++语法。

1.1 了解 C++程序设计

C++是一门同时具备面向过程和对象特性的语言，学习 C++语言既可以了解底层 API 函数的使用方法、控制变量的内存生存周期，编写功能强大高效的算法，也可以从对象的角度组织各个模块，掌握面向对象的编程思想。在学习 C++之后能够对程序设计有深刻的体会，为进一步学习打下坚实基础。

1.1.1 学习 C++的好处

在软件技术快速更新的今天，初学者面临的第一个问题就是：我该学哪种语言。刚接触编程的初学者容易被 ASP.NET、Java、Visual Basic、PHP 等名字弄得晕头转向，诚然这是一个让开发人员疯狂的年代，曾经引以为豪的技术可能不久就被新技术替代，或者由于公司项目需要不得不从自己擅长的技术平台转向新平台。心中不免思考：究竟什么东西是一劳永逸的？答案是没有，没有一门技术可以解决所有问题，但有一样东西可以帮助你快速转入其他相关领域，那就是编程的思想。

所谓编程思想，即从程序的角度思考解决问题，了解程序代码的运行原理、变量的生存周期、窗口的诞生与销毁、消息的传递流程等。当深入了解这些底层原理后，再去学习任何一门开发技术，都会有居高临下之感，从而快速转型。

和 C#、Java 等托管代码不同，C++可以直接操作物理内存，自由控制变量的生存周期，学习 C++你可以了解程序运行时，哪些要占用内存、内存分配方式、变量内存布局、内存释放、内存泄漏等其他语言无法涉及的深层内容。

C++面向对象的特性，可以让你了解如何构造类、创建类对象、释放类对象、类继承机制、虚函数实现原理等，C#、Java 都是由 C++演变而来的，学习 C++可以掌握语言最核心的内容，其他语言则可触类旁通。事实证明，有 C++背景的程序员学习其他语言周期比较短，而且理解得更加深刻，记住，真正的程序员学习 C++！

1.1.2 最简单的 C++程序——hello world

【实例 1-1】新建一个控制台程序 hello world，输出一段文字。

(1)启动 VC 6.0，选择 File|New 命令，打开 New 窗口，在 Projects 选项卡里选择 Win32 Console Application 项（Win32 控制台应用程序），在 Project name 文本框中输入 hello world。在 Location 文本框里选择工程存放路径，如图 1-1 所示。

（2）单击 OK 按钮，打开 Win32 Console Application 窗口，选择 An empty project 项（空工程），单击 Finish 按钮，再单击 OK 按钮，完成工程的创建。

（3）选择 File|New 命令，打开 New 窗口，在 Files 选项卡里选择 C++ Source File 项（C++源文件），在 File 文本框中输入 hello world，如图 1-2 所示。

图 1-1　新建工程窗口

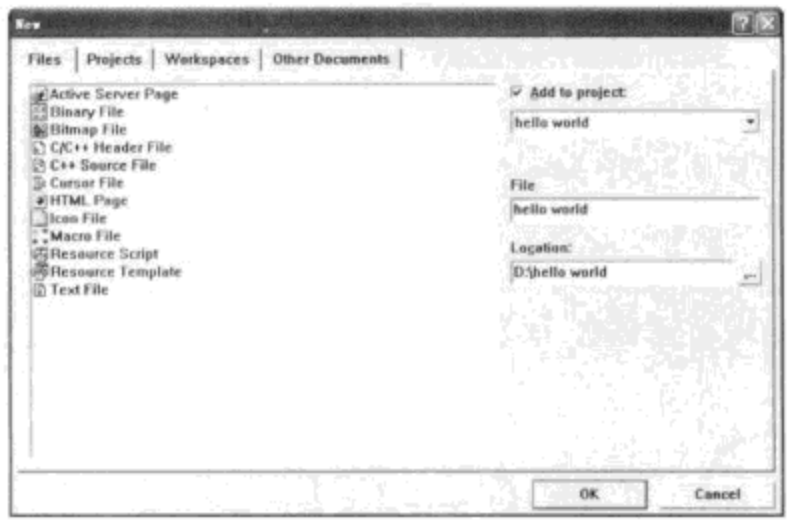


图 1-2　添加 C++源文件

（4）单击 OK 按钮后，在工作区窗口选择 FileView 标签，展开 Source Files 节点，双击添加的源文件 main.cpp，打开代码编辑窗口，输入以下代码：

```
#include <iostream>                    //包含输入/输出流头文件
using namespace std;                   //使用标准命名空间 std
int main()                             //主函数
{
    cout<<"hello world"<<endl;
    return 0;                          //返回 0
}
```

（5）按 F7 键或单击按钮，生成 EXE 可执行程序，单击按钮执行 EXE 程序，结果如图 1-3 所示。

图 1-3　输出 hello world

1.2 基本数据类型

C++是一种数据类型严格的语言，计算机根据数据类型分配对应大小的内存空间，C++内置有几种基本的数据类型，掌握这些基本类型后，可以通过结构体（struct）和类（class）创建自定义类型。

1.2.1 整型 int

int 类型用来存放整数（integer），一个 int 类型变量通常占用 4 个字节（byte），具体大小在不同平台上可能不同。计算机使用二进制方式存储数据，类似 10001000，一个 0 或 1 占用 1 位（bit），一个字节占用 8 位，int 类型变量占用 32 位，范围为-2^{31}~2^{31}。

修饰符 short 表示短整型，如 short int a=123; 占用 2 个字节，范围为-2^{15}~2^{15}，short 等同于 short int。修饰符 long 表示长整型，如 long int a=12345678; 占用 4 个字节，long 等同于 long int。

默认 int 类型有正负符号，使用修饰符 unsigned 表示无符号，如 unsigned int a=12345678; 范围为 0~2^{32}，无符号修饰符 unsigned 对 float、double、char 同样适用。

【实例 1-2】输出未赋值和赋值后的整型变量的值。

```
#include <iostream>
using namespace std;
int main()
```

```
{
    int a;
    cout<<"未赋初值 a="<<a<<endl;              //输出系统随机分配的值
    a=123;
    cout<<"赋值后   a="<<a<<endl;              //输出新值
    return 0;
}
```

编译运行，结果如图 1-4 所示。若未赋初值，系统随机分配一个初始值，为安全起见，建议声明变量同时赋初值，如 int a=0;

图 1-4　整型变量

1.2.2　浮点型 float、double

浮点型用来存放小数，float 表示单精度，占用 4 个字节，精度约为 7 位。double 表示双精度，占用 8 个字节，精度约为 15 位。可使用科学计数法，例如小数 0.0000123 可表示为 1.23e-5，e 代表指数值（exponent），e-5 即 10 的-5 次方。C++默认将浮点数值当做 double 类型，例如数字 1 默认为 int 类型，1.0 默认为 double 类型，若用 float 类型表示，可加上后缀 f 如 1.0f。

【实例 1-3】输出两个整数相除的值，以及小数与整数相除的值。

```
#include <iostream>
using namespace std;
int main()
{
    double d1=1/7;                             //两个整数相除
    double d2=1.0/7;                           //浮点数除以整数
    cout<<"1/7 = "<<d1<<endl;
    return 0;
}
```

编译运行，如图 1-5 所示。两个 int 类型相除得到的仍然是一个整型数，浮点数与整型数相除，整型数自动转换为浮点数，得到一个浮点值。

图 1-5　浮点运算

Tips　若期望得到两个 int 类型变量相除的浮点结果，应先将其中一个变量强制转换为浮点类型。

1.2.3　字符型 char

字符型 char 用来存放一个字符（character），占用 1 个字节，范围为-2^7~2^7，用两个单引号''包括一个字符，如字母、数字、符号等。每个 char 字符对应有一个 ASCII 值，将 char 类型转换为 int 类型，得到的是 char 字符的 ASCII 值，同理，将 int 型转为 char 类型，得到的是 ASCII 值对应的字符。

【实例 1-4】输出字符'0'对应的 ASCII 值，以及 ASCII 值 0 对应的字符。

```
#include <iostream>
using namespace std;
int main()
{
    char ch='0';
    int a=ch;
    cout<<"字符'0'对应的值："<<a<<endl;          //输出字符'0'对应的值
    ch=0;
    cout<<"值 0 对应的字符为："<<ch<<endl;       //输出值 0 对应的字符
    return 0;
```

```
}
```

编译运行，结果如图 1-6 所示。字符'0'对应的 ASCII 值为 48，ASCII 值 0 对应的字符为'\0'，'\0'是一个转义字符，常作为字符串结束标志，类似有'\n'表示换行，'\\'表示一个\。

图 1-6 字符类型

1.2.4 布尔型 bool

布尔型 bool 用于逻辑判断，只有两个值 true 和 false。布尔型和其他类型可以转换，0 代表 false，所有非 0 数值代表 true，将 bool 值转为整数时，true 为 1，false 为 0。

【实例 1-5】输出浮点值转换得到的 bool 值，以及 false 值。

```
#include <iostream>
using namespace std;
int main()
{
    double d=0.0001;
    bool b1=d;
    bool b2=false;
    cout<<"0.0001 -> "<<b1<<endl;                    //double 转为 bool
    cout<<"false  -> "<<b2<<endl;                    //输出 bool
    return 0;
}
```

编译运行，结果如图 1-7 所示。

图 1-7 bool 类型

1.2.5 宽字符型 wchar_t

宽字符型 wchar_t 用于表示世界上所有的字符，全世界有广泛的语种，如中文、韩文、日文、阿拉伯文等，char 类型只能表示 200 多个基本字符，因而出现了 ANSI 和 Unicode 两种字符集，ANSI 对应 char 单字节类型，Unicode 对应 wchar_t 双字节类型。Unicode 采用双字节编码可以表示世界上所有的字符，更具通用性。

【实例 1-6】输出字符'a'、'爱'对应的值，以及 char 和 wchar_t 占用的字节数。

```
#include <iostream>
using namespace std;
int main()
{
    wchar_t wch1='a';                                //英文字符'a'
    wchar_t wch2='爱';                               //中文字符'爱'
    cout<<wch1<<endl;
    cout<<wch2<<endl;
    cout<<"char "<<sizeof(char)<<endl;               //char 大小
    cout<<"wchar_t "<<sizeof(wchar_t)<<endl;         //wchar_t 大小
    return 0;
}
```

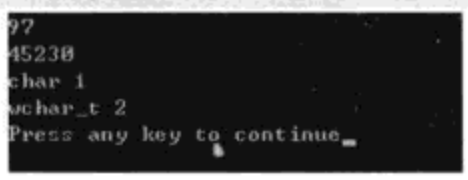


图 1-8 wchar_t 类型

编译运行，结果如图 1-8 所示。中文字符'爱'对应的值超出了 char 的范围，需要用 wchar_t 类型表示。运算符 sizeof 可以得到某一数据类型或变量占用的字节数，char 为 1 个字节，wchar_t 为 2 个字节。

1.3 运算符

C++内置了一些运算符（operator），使用这些运算符能够满足基本的计算需要，若要增强运算符的功能，扩展运算符的应用范围，可使用 C++的运算符重载功能，如加号（+）只能用于基

本类型之间的运算，若要实现两个字符串的相加，如“abc”+“def”得到“abcdef”，由于字符串不是基本数据类型，需要对字符串类重载加号运算符 operator +，以实现特定功能。

1.3.1　算术运算符

算术运算符用于常见的数学运算，如加（+）、减（-）、乘（*）、除（/）、求余(%)，其中求余运算要求为两个整数，即 int%int。若参与运算的两个数精度不同，低精度的数自动转换为相同的高精度数，如 double/int 运算，int 自动提升为 double 类型。优先级规则同一般的数学运算，可使用括号()。

【实例 1-7】输出两个整数加、减、乘、除、求余的结果。

```
#include <iostream>
using namespace std;
int main()
{
    int a1=1500;
    int a2=300;
    cout<<"相加："<<a1+a2<<endl;
    cout<<"相减："<<a1-a2<<endl;
    cout<<"相乘："<<a1*a2<<endl;
    cout<<"相除："<<a1/a2<<endl;
    cout<<"求余："<<a1%a2<<endl;
    return 0;
}
```

编译运行，结果如图 1-9 所示。

1.3.2　关系运算符

关系运算符用于比较两个变量的大小，如大于（>）、小于（<）、等于（==）、大于等于（>=）、小于等于（<=）、不等于（!=），关系运算结果为 true 或 false，常用于逻辑判断。

【实例 1-8】输出两个关系表达式的结果。

```
#include <iostream>
using namespace std;
int main()
{
    double d1=1.0/99999;                          //极小的浮点值
    cout<<d1<<"==0 结果："<<(d1==0)<<endl;        //是否等于 0
    cout<<d1<<"> 0 结果："<<(d1>0)<<endl;         //是否大于 0
    return 0;
}
```

编译运行，结果如图 1-10 所示。浮点值总不等于 0，应避免浮点值和 0 的相等比较。

图 1-9　算术运算

图 1-10　关系运算符

1.3.3　赋值运算符

赋值运算符（=）用于将运算符右边的值赋给左边的变量，两个连续的等号（==）表示等于关系，要小心区别。赋值运算符与算术运算符结合可简化代码，如 a=a+2; 可简化为 a+=2; 类似的有-=、*=、/=、%=。

赋值运算符两边的数据类型应一致，或右边的类型可自动转换为左边的类型，否则需要强制类型转换，如 double b=0.01; int a=(int)b;。double 类型不会自动转换为低精度的 int 类型，需要使用(int)进行强制类型转换。

一般情况下，低精度类型可以自动转换为高精度，高精度向低精度转换需要强制转换。强制转换在 C 和 C++中有所不同，C 语言在括号中放入要转换的类型，如 int a=(int)b; C++语言在括号中放入要转换的变量或表达式，如 int a=int(b); 由于 C++完全兼容 C，因此在 C++中两种方式都可以使用。

【实例 1-9】输出两整数、浮点数与整数相除的结果，以及*=自乘运算结果。

```
#include <iostream>
using namespace std;
int main()
{
    int a=7;                                    //赋初值
    double b1=a/22;                             //两整数相除
    double b2=(double)a/22;                     //强制转换为浮点数后相除
    cout<<"b1 "<<b1<<endl;
    cout<<"b2 "<<b2<<endl;
    b2*=100;                                    //自乘 100
    cout<<"b2*100 "<<b2<<endl;
    return 0;
}
```

编译运行，结果如图 1-11 所示。(double)a 强制将 a 的值转为 double 类型，在括号内写入要转换的类型，也可使用 double(a)形式进行转换。

```
b1 0
b2 0.318182
b2*100 31.8182
Press any key to continue
```

图 1-11 赋值运算符

1.3.4 自增自减运算符

自增（++）自减（--）运算符用于将变量增加或减去 1，如 a++、a--、++a、--a，运算符在变量后（a++、a--）先以 a 的当前值运算，再对 a 自增或自减。若在变量前（++a、--a）先对 a 自增或自减，以变化后的 a 参与计算。若为一个单独的表达式，如 a++; ，则在前在后没有区别。

【实例 1-10】输出自增、自减运算结果。

```
#include <iostream>
using namespace std;
int main()
{
    int a=0;
    int b=a++;                                  //将当前值 0 赋给 b，然后自增 1
    cout<<"a++ "<<b<<endl;
    cout<<"a   "<<a<<endl;
    int c=--a;                                  //先自减 1，然后将值赋给 c
    cout<<"--a "<<c<<endl;
    cout<<"a   "<<a<<endl;
    return 0;
}
```

编译运行，结果如图 1-12 所示。程序代码的可读性非常重要，使用 int b=a++; 这种形式的语句无形中提高了阅读的难度，把简单的事情复杂化，为提高程序的易读性，建议拆分为 int b=a; a++; 而 int c=--a; 可拆分为 int c=a-1; a--; 将 a--当做 a=a-1 的一种简写方式。

```
a++ 0
a   1
--a 0
a   0
Press any key to continue
```

图 1-12 自增自减运算符

1.3.5　逻辑运算符

逻辑运算符用于关系表达式的组合，如与（&&）、或（||）、非（!），常用于条件的逻辑判断。

- 要求多个条件同时成立，使用&&。若前面的表达式为假，则停止判断后面的表达式。
- 要求至少一个条件成立，使用||。若前面的表达式为真，则停止判断后面的表达式。
- 要求条件的对立面成立，使用!。

【实例 1-11】输出用逻辑运算符连接的多个关系表达式的结果。

```
#include <iostream>
using namespace std;
int main()
{
    bool b1=(12>=5 && 1<0);                    //与，要求全部为 true
    bool b2=(1/7==0 || -1>=0);                 //或，要求至少一个为 true
    bool b3=(!b1);                             //非，true 的对立面为 false
    cout<<"b1 "<<b1<<endl;
    cout<<"b2 "<<b2<<endl;
    cout<<"b3 "<<b3<<endl;
    return 0;
}
```

编译运行，结果如图 1-13 所示。若一个逻辑判断表达式中有多个逻辑运算符，建议使用括号提高可读性，尽量不要依赖逻辑运算符的优先级，代码是让人来读的，以易读为第一标准。

```
b1 0
b2 1
b3 1
Press any key to continue
```

图 1-13　逻辑运算符

1.3.6　位运算符

计算机存储数据的最小单位是位（bit），位运算符用于位级别的数据计算，包括位移运算符左移（<<）、右移（>>），位逻辑运算符与（&）、或（|）、异或（^）、非（~）。

位移运算符移动位数据的位置，例如十进制整数 10 用二进制表示为 1010。

- 将 10 右移 1 位（10>>1）后，变为 0101 即 5。
- 将 10 右移 2 位（10>>2）后，变为 0010 即 2。
- 将 10 左移 1 位（10<<1）后，变为 10100 即 20。

位逻辑运算符根据每个位的值进行逻辑运算，例如十进制整数 10 用二进制表示为 1010，十进制整数 2 用二进制表示为 0010。

- 与运算（&）：若对应位都为 1，结果为 1，否则为 0，10&2 结果为 0010。
- 或运算（|）：若对应位有一个为 1，结果为 1，否则为 0，10|2 结果为 1010。
- 异或运算（^）：若对应位一个为 1，一个为 0，结果为 1，否则为 0，10^2 结果为 1000。
- 非运算（~）：若对应位为 1，变为 0，若为 0，变为 1，~10 的结果为 0101。

【实例 1-12】输出左移、右移，以及与、或、异或、非的位运算结果。

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"10<<1  "<<(10<<1)<<endl;          //左移 1
    cout<<"10>>1  "<<(10>>1)<<endl;          //右移 1
    cout<<"10&2   "<<(10&2)<<endl;           //与
    cout<<"10|2   "<<(10|2)<<endl;           //或
    cout<<"10^2   "<<(10^2)<<endl;           //异或
    cout<<"~10    "<<(~10)<<endl;            //非
    return 0;
}
```

编译运行，结果如图 1-14 所示。左移一位相当于乘以 2，右移一位相当于除以 2。

Tips 位运算常用于参数属性的标志位，如一个标志参数用二进制表示为 10010001，每个位代表一种特定的属性，位若为 1 表示使用该位所代表的属性，若为 0 表示不使用，如用 CFile 类调用 Open 函数打开文件时指定 CFile::modeCreatelCFile:: modeWrite 两个标志，用位或运算符相连，则这两个标志所对应位的值为 1，其余位都为 0。

图 1-14 位运算符

1.4 控制结构

计算机最神奇的地方在于能够按照人指定的方式执行，且高效无误。比如有一组成绩数据，去除最高分和最低分，然后计算平均分，写好算法后，只要输入数据，程序会自动算出结果。算法步骤如下：

（1）将成绩按照高低排序，去除最高分和最低分。

（2）计算剩余成绩的总分数。

（3）总分数除以数目，得到平均分。

分析以上步骤，排序要判断两个成绩的高低，计算总成绩要重复地累加，因此程序需要一种机制来进行逻辑判断和循环计算，C++提供了 if/else 结构进行逻辑判断，while、for 结构进行循环计算，所有复杂的算法都是基于这几种控制结构而成的。

1.4.1 if/else 选择结构

if/else 选择结构是一种常见的结构，例如：如果（if）周末天气不错，我就出去玩，否则（else）我就在家学习。if 表达式判断条件语句是否为真，若为真执行 if 块内的语句，否则跳转到 else 块内执行。可以只有一个 if 语句，也可以有多个 else，也可以在 if/else 里面嵌套多层 if/else。

【实例 1-13】利用 if/else 判断语句输出结果。

```
#include <iostream>
using namespace std;
int main()
{
    double d1=40;
    if(d1>=80)                                    //如果大于等于 80
        cout<<d1<<" 良好"<<endl;
    else if(d1>=60)                               //小于 80，如果大于等于 60
        cout<<d1<<" 及格"<<endl;
    else                                          //小于 60
    {
        cout<<d1<<" 不及格"<<endl;                 //执行{}块内的语句
        cout<<"要努力了！ "<<endl;
    }
    return 0;
}
```

图 1-15 if/else 结构

编译运行，结果如图 1-15 所示。使用多个 if/else 控制语句时应注意代码的缩进和{}的使用，以便快速查找匹配的 if/else。若用{}包括多行语句，执行该块内的语句，否则执行到第一个分号处。

1.4.2　while 循环结构

重复性计算是计算机程序最实用的功能，随着计算机硬件计算性能的飞速提升，一秒钟能完成上亿次的运算，充分利用其计算能力可以大大提高工作效率。C++提供多种方式实现循环计算，如 while、do-while、for 循环。

while 循环有一个条件语句，在该条件为假之前，循环一直持续下去。若 while 循环一直持续下去，就会陷入死循环，造成程序瘫痪，需要在循环体里改变变量值，使条件语句在某时刻为假，从而退出循环。或者使用跳出循环体语句 break;。

使用 while 循环时，首先检查条件语句是否成立，若第一次就不成立，则不进入循环体。若要循环体至少执行一次，使用 do-while 循环，先执行一次循环体，然后判断条件语句是否成立。第一次循环之后，while 和 do-while 没有区别。

【实例 1-14】利用 while 循环计算整数 a 的素数组合因子，如 48 的素数组合因子为 2、2、2、2、3。

```
#include <iostream>
using namespace std;
int main()
{
    int a=48;
    int i=2;                          //测试因子（从 2 到 a）
    cout<<a<<"的因子: ";
    while(a!=1)                       //若 a 已除尽，退出循环
    {
        if(a%i==0)                    //若 a 能被 i 整除
        {
            a=a/i;                    //除去 i 后的值，若 a 为 1，则已除尽
            cout<<i<<" ";             //输出素数因子
        }
        else                          //若 a 不能被 i 整除，因子自增
            i++;
    }
    cout<<endl;
    return 0;
}
```

编译运行，结果如图 1-16 所示。i 作为测试因子，从 2 开始递增测试，若 a 能够整除 i，则将 a 变为整除后的值，同时输出因子 i。反复测试 i 直到 a 不能整除 i，如 a=48，i=2，a/i 后，a=24，输出 i，再次执行 a/i 后，a=12，输出 i，直到 a 不能整除 i 时，i 递增为 3，依此类推，直到 a 等于 1 时退出循环体。

图 1-16　while 循环

1.4.3　break 和 continue 语句

break 语句用于终止循环，例如要查找某用户名对应的密码，当查找到对应密码后即可停止循环。continue 语句用于终止本次循环，直接进入下一次循环，例如查找到的用户名不是想要找的用户名，就没必要继续查看对应的密码，直接查找下一个。若有多层嵌套循环，break 只退出该语句所在的循环体，continue 同理。

【实例 1-15】查找 2000～3000 范围内第一个可被 168 整除的数。

```
#include <iostream>
using namespace std;
int main()
{
    int i=2000;                       //从 2000 开始逐个测试
```

```
    while(i<=3000)                    //若 i 小于等于 3000，执行循环体
    {
        if(i%168==0)                  //若 i 能被 168 整除
        {
            cout<<i<<" ";             //输出 i
            break;                    //跳出循环体
        }
        i++;                          //i 自增
    }
    return 0;
}
```

编译运行，计算出 i 为 2016。从 2000 开始逐个数进行判断，若能被 168 整除，输出值并退出循环体，否则递增值，判断下一个值是否满足条件。若测试值大于 3000，则不再执行循环体。

1.4.4 for 循环结构

for 循环类似于 while 循环，不同之处在于 for 循环将初始化、条件判断、变量递增放在一起，如上一节的实例代码中，int i=2000; 为初始化，i<=3000 为条件判断，i++; 为变量递增，用 for 循环结构表示就是 for(int i=2000;i<=3000;i++)，括号内至少有两个分号，内容可以为空，如 for(; ;)，其他同 while 循环。

【实例 1-16】计算 1～99 的累加和。

```
#include <iostream>
using namespace std;
int main()
{
    int sum=0;                        //总和
    for(int i=1;i<=99;i++)            //从 1 到 99 逐个计算
    {
        sum+=i;                       //将 i 加到 sum 中
    }
    cout<<"1+2+…+99 = "<<sum<<endl;   //输出累加和
    return 0;
}
```

编译运行，计算结果为 4950。利用 for 循环初始化 i 为 1，循环条件为 i<=99，每次循环变量 i 递增 1，将 1～99 中的每个数都加到 sum 中，得到累加和。

Tips for 循环的( )中定义的变量 i，在标准 C++中规定是局部变量，仅在 for 循环内部可用。在 VC6.0 环境中 i 在 for 循环体之后仍可使用，相当于 int i=0; for(i=0;…){}，这是 VC6.0 的自身不规范造成的，Visual Studio 2005 等高级版本符合标准 C++规范，常用解决方法是把 int i=0; 提取出来作为一个单独的表达式，放到 for 循环之前，避免在不同环境下造成混乱。

1.4.5 switch 多选结构

switch 结构用于多种可能结果的分类操作，例如选择计算类型，若为加法，执行相加运算，若为除法，执行相除运算。若使用 if/else 结构也可，但不够简练直观。

switch()内的变量或表达式必须是整型值，或者能够自动转换为整型的类型。每种可能结果都是一个 case 子句，可执行不同的操作，case 子句后需要有 break 语句，否则将继续执行后面的 case 子句。可有一个默认的 default 子句，若所有 case 都不匹配，执行 default 子句。

【实例 1-17】利用 switch 语句根据运算符类型，执行不同的运算。

```
#include <iostream>
using namespace std;
int main()
{
    char ch='*';                                    //运算类型
    int a=120;
    int b=30;
    double result=0;                                //结果
    switch(ch)                                      //根据运算类型不同决定计算方式
    {
    case '+':                                       //若为加
        result=a+b;break;
    case '-':                                       //若为减
        result=a-b;break;
    case '*':                                       //若为乘
        result=a*b;break;
    case '/':                                       //若为除
        result=(double)a/b;break;
    }
    cout<<"运算类型 "<<ch<<" 结果 "<<result<<endl;
    return 0;
}
```

编译运行，结果如图 1-17 所示。若在 case 子句后不使用 break 语句，则继续执行后面的 case 子句，直到遇到 break 语句跳出 switch 语句块。

图 1-17　switch 结构

1.5 函数

在面向对象编程（OOP）出现之前，函数（function）是程序的基本组成单元，如 Windows API（Application Program Interface，应用程序编程接口）就是一组数量庞大的函数库，MFC（Microsoft Foundation Class，微软基本类库）类库将功能相关的 API 函数封装到一个类中，可以通过类成员函数方式调用，也可直接调用全局 API 函数，效果是一样的。

1.5.1 什么是函数

函数可看做是一组代码的集合，调用一个函数后，自动执行该函数内的代码。函数可以有多个输入参数和一个返回值，类似于数学公式 y=ax1+bx2+cx3+d，其中 x1、x2、x3 是输入参数，y 是返回值，在函数体内编写具体的算法步骤。

调用函数时，传递输入参数，执行完毕后函数返回一个值，若返回值类型为 void，则无返回值。使用 C++自带的函数库和已写好的函数模块可以大大提高开发效率，不用为某个功能的具体实现影响开发进度，在实际开发中，尽可能使用成熟稳定的函数代码。

1.5.2 定义函数

函数由函数名、参数列表、返回值、函数体组成，函数名应使用有意义的命名，根据名称即可大致了解其用途。参数值可有 0 个或多个，由参数类型和参数名组成，不同参数用逗号分隔。返回值决定函数返回类型，若为 void，表示无返回值，否则，需要用 return 语句返回一个同类型的值。

函数原型：返回值 函数名(类型 参数 1,类型 参数 2,……){ 函数体 }

例如：int GetMax(int a,int b){ if(a>b) return a; else return b; }

1.5.3 变量作用域

函数体内不同位置的变量具有不同的作用域，在函数体外的变量为全局变量（Global），全

局变量在函数内任何位置都有效。作用域可根据{}语句块判定，在块之前定义的变量在块内可用，在块之后定义的变量在块内不可用。

【实例 1-18】全局变量、局部变量的作用域。

```
#include <iostream>
using namespace std;

int global=10;                          //全局变量
int main()
{
    int outBlock=12;                    //块外变量
    if(outBlock>1)
    {
        int inBlock=20;                 //块内变量
    }
    return 0;
}
```

global 作为全局变量，在 main 函数内部任何位置都可用。outBlock 在 main 块内、if 块外，因此 outBlock 在 main 块外不可用，在 main 块内定义位置后所有地方都可用。inBlock 在 if 块内定义，仅在 if 块内可用。

1.5.4 使用函数

自定义的函数必须在定义位置之后才能使用，若在定义位置之前使用，需要做函数声明(declare)，函数声明只需表明函数名、返回值、参数类型，不需要参数名称。

函数若有参数，使用时需传递相同类型的参数值，用赋值运算符接收函数返回值。调用函数时，程序流程进入函数内部，执行完函数体后返回调用位置处。

【实例 1-19】创建自定义函数 sum，在主函数 main 中调用该函数，并输出返回值。

```
#include <iostream>
using namespace std;
int sum(int,int=10);                    //在定义位置之前调用，要做函数声明，第 2 个参数默认为 10
int main()
{
    cout<<sum(100,20)<<endl;            //调用自定义函数
    cout<<sum(100)<<endl;               //使用默认参数值
    return 0;
}
int sum(int a,int b)                    //函数定义
{
    return a+b;                         //返回值
}
```

自定义的 sum 函数的参数 a 和 b 是形式参数，在调用时将实际参数的值复制传入函数，由于传递实际参数的复制，在函数内部对参数值的任何改变都不会影响实际参数的值。

若想改变实际参数的值，可将函数的参数改为指针类型或引用类型，传入实际参数的内存地址，通过内存地址，函数内部的修改直接针对指定内存地址上的变量，这样函数内部的修改就可以影响实际参数的值。

函数的参数可以有默认值，在函数声明时指定默认值，如 int sum(int,int=10); 有默认值的参数必须在参数列表尾处。调用函数时有默认值的参数可不传入，如 sum(100)，第 1 个参数为 100，第 2 个参数为默认的 20，若传入两个参数值，则用传入第 2 个参数值替代默认值。默认参数在函数的参数较多时很有作用，免去每次都输入冗长的无关紧要的参数。

1.5.5　函数重载

函数重载（overload）可以使多个函数名称相同，但参数列表不同，例如函数 sum 计算两个数值的和，数值类型可能为整数、浮点数以及其他可以求和的类型，若分别使用不同的函数名，造成命名复杂且难以记忆，使用函数重载后只需记住一个函数名，调用时传入指定类型的参数即可。

【实例 1-20】重载 sum 函数，可用于整型、浮点型两种类型的求和计算。

```
#include <iostream>
using namespace std;

int sum(int a,int b){                                    //整型计算
    return a+b;
}
double sum(double a,double b){                           //浮点型计算
    return a+b;
}
int main()
{
    cout<<sum(100,20)<<endl;                             //调用整型版本
    cout<<sum(123.5,32.6)<<endl;                         //调用浮点版本
    return 0;
}
```

定义了两个版本的 sum 函数，分别用于整型、浮点型的求和计算。程序执行时根据参数类型自动选择匹配版本的函数，如 sum(100,20)调用整型版本，sum(123.5,32.6)调用浮点版本。

1.6　数组

数组（Array）是用于处理同种类型数据的集合，比如有 50 个人的成绩，可以存放到大小为 50 的整型数组中，便于进行排序、求平均值、最大最小值等操作。数组在定义时需要指定大小，以便分配固定大小的存储空间，在实际开发中，常使用动态大小的链式数据结构替代数组。

1.6.1　什么是数组

数组是内存上连续排列的一种数据结构，数组的起始地址是第一个元素所在的地址，通过下标索引可访问某个元素，索引从 0 开始编号，最后一个元素的索引值为数组长度减 1，若索引大于等于数组长度，会引发程序崩溃。

声明数组时需明确指定数组大小，若要使用变量指定数组大小，可用 new 运算符动态生成一个数组，在使用完数组后用 delete 运算符手动释放该数组。

1.6.2　一维数组

一维数组可看做一组连续排列的数据集合，数组占用内存大小等于数组长度乘以数组元素的大小，使用下标索引访问每个元素，如 a[i]代表第 i+1 个元素。定义数组时可进行初始化，若不对每个元素赋值，系统将分配一个随机值，用{}初始化数组可不指定大小。

【实例 1-21】定义一个一维数组，根据元素值进行升序排序后，获取数组的最大值、最小值、平均值。

```
#include <iostream>
using namespace std;

int main()
{
```

```
    int score[]={78,90,88,100,87};                          //定义并初始化一维数组
    int count=sizeof(score)/sizeof(int);                    //计算数组大小
    for(int i=0;i<count-1;i++)                              //数组按照从小到大的顺序排序
    {
        for(int j=i;j<count;j++)
        {
            if(score[i]>score[j])                           //若第 i 元素大于后面的元素，交换值
            {
                int temp=score[i];                          //将第 i 个元素的值暂时保存到 temp 中
                score[i]=score[j];                          //将第 j 个元素的值赋给第 i 个元素
                score[j]=temp;                              //将 temp 保存的值赋给第 j 个元素
            }
        }
    }
    int sum=0;                                              //总和
    cout<<"排序后：";
    for(int k=0;k<count;k++)                                //遍历数组，输出排序后的数组
    {
        sum+=score[k];                                      //数据累加
        cout<<score[k]<<" ";
    }
    cout<<endl<<endl;
    cout<<"最大值："<<score[count-1]<<endl;                 //最大值
    cout<<"最小值："<<score[0]<<endl;                       //最小值
    cout<<"平均值："<<(double)sum/count<<endl;              //平均值
    return 0;
}
```

```
排序后：78 87 88 90 100

最大值：100
最小值：78
平均值：88.6
Press any key to continue
```

图 1-18 一维数组

编译运行，结果如图 1-18 所示。定义数组时直接使用{}赋初值，这样无须在[]中输入数组的大小，自动根据{}中元素的数目决定数组的大小。sizeof(score)得到整个数组占用的内存大小，sizeof(int)得到一个 int 元素占用的大小，相除得到数组元素数目。

两个 for 循环用于对数组排序，外层 for 循环第 1 次循环得到最小值，用第 1 个元素的值与其后的所有元素比较，若第 1 个元素的值大于其后的某个元素，将较小的元素和第 1 个元素交换值，这样第 1 个元素的值始终是较小的，所有元素都比较结束后，第 1 个元素的值就是最小值。第 2 次循环用第 2 个元素的值与其后的所有元素比较，得到次小值，依此类推。

排序后将每个元素的值累加，得到所有元素的总和，除以元素数目得到平均值。由于排序方式为升序，第 1 个元素为最小值，最后一个元素为最大值。

1.6.3 二维数组

二维数组可看做一个 n 行 m 列的矩阵，使用方法类似于一维数组。通过下标访问每个元素，如 a[i][j]代表第 i 行 j 列的元素（索引从 0 开始编号）。

二维数组定义时可进行初始化，如 int a[][2]={{1,2},{3,4}}；内层的{1,2}为第一行的初始值，数组 a 的行数为内层{}的数目，可不指定大小，但必须指定数组 a 的列数，内层{}中元素的数目不能大于列数，若小于列数，剩余未赋值的元素默认为 0。

【实例 1-22】定义二维数组并初始化，获取行数和列数，输出所有元素的值。

```
#include <iostream>
using namespace std;

int main()
{
    int a[2][3]={{1,3,4},{7,5,6}};                          //定义二维数组并初始化
    int row=sizeof(a)/sizeof(a[0]);                         //行数
```

```
    int column=sizeof(a[0])/sizeof(int);                    //列数
    cout<<"a 行数: "<<row<<endl;
    cout<<"a 列数: "<<column<<endl;
    for(int i=0;i<row;i++)                                  //输出所有元素
    {
        for(int j=0;j<column;j++){
            cout<<a[i][j]<<" ";
        }
        cout<<endl;                                         //换行
    }
    cout<<endl;
    return 0;
}
```

编译运行，结果如图 1-19 所示。初始化时一个内层{}表示一行数据，sizeof(a)得到整个二维数组占用的内存大小，sizeof(a[0])得到一行元素占用的内存大小，sizeof(int)得到一个 int 元素的大小，用数组总大小除以一行大小得到行数，用一行大小除以每个元素大小得到列数。

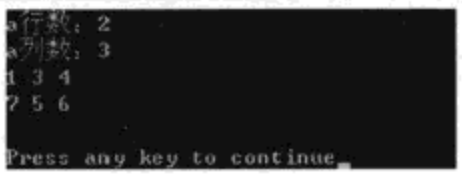


图 1-19　二维数组

外层的 for 循环遍历每一行，内层的 for 循环遍历当前行的每一列，两个 for 循环输出所有元素，当输出一行元素后，输出换行符，另起一行显示下一行元素。

1.6.4　动态数组

声明数组时必须指定固定的大小值，但有时候在运行前并不知道元素的总数目，需要在程序运行时确定元素数目，可使用 new 运算符在程序运行时动态生成一定大小的数组。

一般类型的变量都有作用域，程序运行时在栈（stack）中创建，当超出变量作用域后被自动释放。使用 new 运算符动态创建的变量在堆（heap）中创建，不会自动释放，必须使用 delete 运算符手动释放。若使用 new 动态创建后忘记用 delete 释放，会造成内存泄漏，这也是 C++被认为不安全的原因之一。

【实例 1-23】用 new 运算符动态创建一个数组，对元素赋值并输出后，用 delete 运算符释放该数组。

```
#include <iostream>
using namespace std;

int main()
{
    int cnt=10;
    int* a=new int[cnt];                                    //动态创建数组，数组大小为变量 cnt
    for(int i=0;i<10;i++)                                   //数组赋值
    {
        a[i]=i*10-i;
        cout<<a[i]<<" ";
    }
    delete [] a;                                            //释放动态创建的数组
    cout<<endl;
    return 0;
}
```

用 new 运算符动态创建一个 int 数组，大小为变量 cnt，返回一个 int 指针，指向刚分配的数组的首地址。数组名实际上等于数组第一个元素的地址，即 a 等同于&a[0]，所以数组变量和指针变量性质是相同的。

使用完数组后，要用 delete 手动释放，如 delete a; a 为要释放的变量的地址，若 a 指向的是一个数组，需要使用[]表明要释放的是数组，如 delete [] a; 否则只释放第一个元素。

1.6.5 数组排序

在一维数组中使用了一种排序方法，C++函数库自带一个排序函数 qsort，用于实现对数组的快速排序，函数格式如下：

```
void qsort(void *buf, size_t num, size_t size, int (*compare)(const void *, const void *))
```

参数：

- buf：数组首地址（数组名即可）。
- num：数组元素数目。
- size：每个元素的字节大小。
- compare：比较函数名，指定了函数原型。

size_t 是 unsigned int 类型的重定义，格式如下：

```
typedef unsigned int size_t;
```

typedef 将一种类型重命名为另一种名称，便于根据类型即可知道变量的作用，实际上还是一种数据类型，Windows API 有很多陌生的数据类型，都是使用 typedef 进行基本类型的重定义得到的。

int (*compare)(const void *, const void *)表示一个符合特定函数原型的指针，即传入的函数名的原型必须与该原型一致，qsort 函数根据该函数的返回值决定是否交换两个参数。

【实例 1-24】使用 qsort 函数对数组进行排序，输出排序后的元素。

```
#include <iostream>
using namespace std;

int compare(const void* a,const void* b) //排序比较函数，a 和 b 为比较的两个数据的指针
{
    int n1=*((int*)a);                   //指针 a 对应的值
    int n2=*((int*)b);                   //指针 b 对应的值
    if(n1<n2)                            //若参数 1 的值小于参数 2 的值，返回-1
        return -1;
    else if(n1==n2)                      //若相等，返回 0
        return 0;
    else                                 //若大于，返回 1
        return 1;
}

int main()
{
    int a[5]={46,43,90,10,40};                                  //定义并初始化数组
    qsort(a,sizeof(a)/sizeof(int),sizeof(int),compare);         //快速排序
    for(int i=0;i<sizeof(a)/sizeof(int);i++)                    //显示排序后的数组元素
    {
        cout<<a[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

传入 compare 函数的指针为 void 类型，需要转换为实际的 int 类型，利用*获取指针指向的值。函数 compare 的原型必须与 qsort 函数中的定义保持一致，对传入的两个值进行判断，compare 函数可决定在何种情况下返回何值。

若 compare 函数返回正值，qsort 函数会交换两个参数的值，如当参数 1 大于参数 2 时，返回正值，qsort 函数交换参数 1 和参数 2 的值，实现升序排序。

1.7 指针

指针是 C 和 C++灵活强大的重要因素，但也是难以控制的不稳定因素，在 C#、Java 语言中都取消指针类型，不能够直接操作物理内存。指针是变量的内存地址，不同类型的变量需要不同的指针类型，尽管指针的实际值是一个数字，但用于存储指针值的变量必须明确声明为指针类型。

1.7.1 指针概述

指针存储变量的内存地址，指针类型为变量类型加*，如 int* p1; 若 p1 设定为指向 int 类型的指针，则不能接受其他类型的指针。&用于获取变量的内存地址，如 int* p1=&a; 要获取指针指向的值，使用*如 int b=*p1; 在不同的场合*有不同的意义，应小心区分。

指针变量实际存放的就是一个地址值，但编译器将其抽象为一种数据类型，在实际应用中，经常需要进行指针类型的转换，如 int*转为 void*、void*转为 double*、派生类指针转为基类指针等，这些转换都是编译器层次上的表面转换，实际上就是一个地址。

【实例 1-25】输出指针变量存储的地址值，指针指向的值，指针本身所在的地址，指针变量占用的内存大小。

```
#include <iostream>
using namespace std;

int main()
{
    int a=120;                                  //整型变量
    int* p=&a;                                  //将 a 的地址赋给整型指针 p
    cout<<"指针 p 的值为："<<p<<endl;            //p 存储地址值
    cout<<"p 指向的值为："<<*p<<endl;            //p 指向的值
    int** q=&p;                                 //指向指针 p 的指针
    cout<<"指针 p 的地址："<<q<<endl;            //指针 p 所在的内存地址
    cout<<"指针 p 的大小："<<sizeof(p)<<endl;    //指针占用的空间大小
    return 0;
}
```

编译运行，结果如图 1-20 所示。通过&获取整型变量 a 的地址值，存放到 int 指针 p 中。通过*获取指针 p 所指向的变量的值。指针作为一个变量，也有其地址，通过&获取指针变量的地址值，存放到 q 中，int**表示一个指向 int 指针变量的指针。通过 sizeof 获取指针变量占用的内存大小，指针变量占用 4 个字节。

图 1-20　指针

1.7.2 指针与数组

数组变量实际上就是一个指针，数组名为数组的第一个元素地址，即数组的首地址。根据下标索引确定元素地址，如 a[1]的地址等于 a[0]的地址加上一个元素的长度。

尽管指针实际上是个地址值，但不能把指针当做数值进行数学运算，只能按照编译器设定的方式进行运算，如 a[2]等同于*(a+2)，数组名 a 为数组首地址，2 表示两个元素的长度，(a+2)即为第 3 个元素的地址，通过*获取第 3 个元素的值。

【实例 1-26】将数组名赋给指针变量，用指针变量访问数组元素，输出每个元素的值和地址值。

```
#include <iostream>
using namespace std;

int main()
{
    int a[3]={12,45,60};
```

```
    int* p=a;                                          //数组名赋给指针变量 p
    cout<<p[0]<<" "<<(long)&p[0]<<endl;                //输出第 1 个元素值及地址值
    cout<<p[1]<<" "<<(long)&p[1]<<endl;
    cout<<p[2]<<" "<<(long)&p[2]<<endl;
    return 0;
}
```

```
12 1245044
45 1245048
60 1245052
Press any key to continue
```

图 1-21　指针与数组

编译运行，结果如图 1-21 所示。将数组名 a 赋给指针变量 p 后，指针 p 同样可以使用下标访问数组元素，由此可见，数组名 a 等同于数组的首地址&a[0]。&p[0]得到数组第 1 个元素的地址，(long)强制将地址值转换为长整型数，默认为十六进制数。三个元素的内存地址是连续的，相邻元素相差 4 个字节，为一个元素的大小。

1.7.3　指针与函数

函数有多个参数，但只有一个返回值，若函数执行后有多个输出值，仅依赖函数的返回值是不够的。可以将部分参数作为输出值，实现输出多个值，如将变量 a、b 作为参数传给函数 f，即 f(a,b)，调用函数 f 后，a 和 b 存放的是输出值。但一般情况下，传递给函数 f 的是变量 a、b 的值拷贝，在函数 f 内部的任何修改对 a、b 本身都没有影响，在 C 语言中使用传入变量的指针的方法解决该问题。

将变量的指针值传递给函数，如 f(&a,&b)，传入函数 f 的是 a、b 的地址值的拷贝，在函数内部通过该地址值可以访问变量 a、b，从而修改其值，在函数 f 的内部操作过程中已经将外部变量 a、b 的值修改了。C++提供了引用方式可达到同样的效果，两者形式不同，但实质上都是通过传入变量的内存地址达到改变外部变量值的效果。

【实例 1-27】使用传递变量指针和引用两种方式，实现修改外部变量。

```
#include <iostream>
using namespace std;
//指针方式改变参数值
void CalcByPointer(int a,int b,int* pAddResult,int* pSubResult)
{
    *pAddResult=a+b;                                   //相加结果保存到指针参数所指向的变量中
    *pSubResult=a-b;
}
//引用方式改变参数值
void CalcByReference(int a,int b,int& AddResult,int& SubResult)
{
    AddResult=a+b;                                     //相加结果保存到引用参数中
    SubResult=a-b;
}
int main()
{
    int a=120,b=30;
    int c1,c2,d1,d2;
    CalcByPointer(a,b,&c1,&c2);                        //指针方式
    CalcByReference(a,b,d1,d2);                        //引用方式
    cout<<"指针方式: "<<c1<<","<<c2<<endl;             //输出结果
    cout<<"引用方式: "<<d1<<","<<d2<<endl;
    return 0;
}
```

编译运行，结果如图 1-22 所示。通过指针方式传入变量 c1、c2 的指针值后，pAddResult 指向要改变的变量 c1，pSubResult 指向要改变的变量 c2，在 CalcByPointer 函数内部执行 *pAddResult=a+b; 后，直接在内存上进行变量的修改，pAddResult 所指向变量 c1 的值已经改变。

引用相当于变量的别名，引用和变量占用同一块内存，改变引用的值等同于改变变量本身。引用定义的同时必须初始化，如 int& b=a; 表示 b 是变量 a 的一个引用，对 b 的任何操作等同于对 a 的直接操作。CalcByReference 函数中 AddResult 和 SubResult 两个参数是整型引用，将 d1、d2 传入函数，相当于 int& AddResult=d1; int& SubResult=d2; 。对 AddResult、SubResult 的修改相当于直接修改 d1、d2 的值。

图 1-22　指针与函数

1.7.4　指针与字符串

字符串可看做常量字符数组，如字符串“language”是长度为 9 的字符数组，前 8 个元素存储单个字符，最后 1 个为结束标志'\0'，表明字符串到此结束。字符'\0'的 ASCII 值为 0，可根据元素值是否等于 0 来判断是否到达字符串末尾。

一个字符数组能存放的字符数目为数组长度减 1，字符数组初始化时，C++自动在字符串末尾添加空字符'\0'。字符数组名相当于一个指向字符串首地址的指针，通过移动指针指向的位置可灵活操作一个字符串。

【实例 1-28】通过字符指针遍历一个字符数组，输出所有字符。

```
#include <iostream>
using namespace std;

int main()
{
    char a[]="Study Program";                    //字符数组
    char* p=a;                                   //字符指针，指向数组首地址
    while(*p!=0)                                 //若指针指向的字符不等于 0（ASCII 值）
    {
        cout<<*p<<" ";                           //输出字符
        p++;                                     //字符指针向前移动一位
    }
    cout<<endl;
    return 0;
}
```

编译运行，结果如图 1-23 所示。字符数组名等同于字符数组的首地址，将字符数组名赋给字符指针 p 后，p 指向第 1 个元素，通过*获取 p 指向的元素的值。p 每次递增移动一个元素大小的位置，指向下一个元素，当指向字符串结束标志'\0'时，停止循环。

图 1-23　指针与字符串

1.8　小结

本章讲述了 C++的基本语法。Visual C++是 C++的可视化开发环境，由于 MFC 类库采用 C++语言编写，所以学习 Visual C++前，掌握 C++是关键。本章主要介绍了 C++语言的基本数据类型、运算符、控制结构、函数、数组和指针。其中函数、数组和指针较为复杂，希望读者认真掌握，为后面 Visual C++程序的编写打下坚实的基础。

1.9　习题

1. C++中基本数据类型有哪些？
2. 函数的具体组成是什么？如何定义一个函数？
3. 编写一个程序，输出“你好，中国”。
4. 什么是数组？如何定义一个二维数组？
5. 编写一个程序，比较两个数的大小。

第 2 章　面向对象程序设计

在面向对象程序（OOP）设计思想出现之前，程序采用面向过程的设计方法，程序由数据和函数组成，函数是程序的基本组成单元。面向过程方法灵活性强，逻辑简单，但函数名众多容易造成混乱，且函数和数据分开存放，联系不够紧密，在大型软件开发中采用面向过程方式效率低下、维护困难。

面向对象编程的基本组成单元是类（class），C++语言作为 C 的扩展，提供了一种新的数据类型：类（class），由数据和处理数据的函数共同组成。从类的层次组织程序模块，设计类及所用的算法是面向对象编程的主要内容。

2.1 类和对象

类和对象是面向对象编程的核心，类和对象类似于建筑图纸和大楼的关系，类是对象的一种抽象，对象是类的一个实例。在面向对象世界中，一切事物都可以用对象表示，对象间通过消息进行交流，无须了解对象的内部实现，只要知道各个对象提供的功能接口就能实现各种高级功能。

2.1.1 类和对象的关系

类是一种自定义数据类型，类似于 C 中的结构体（struct），不同之处在于类中包括数据和函数，而结构体只有数据。类是对象的形式上的抽象，不是一个实际存在的事物，它规定有哪些数据成员，哪些成员函数及具体功能，是对具体事物的一种描述。对象是类的实现（implement），对象根据类的描述信息生成一个真实的事物，占用实际内存，对象生成时要调用类的构造函数（construct）进行初始化（initialize），对象释放（release）时要调用类的析构函数做必要的清理工作。

面向对象的三大特性是封装、继承、多态。封装表示将数据和函数封装到类中，用户通过类的方式调用，只需知道类成员的功能和使用方法，无须了解类的具体实现细节。

继承表示一个类可被另一个类重用，如类 A 可以对两个数相加计算，类 B 继承了类 A，同时类 B 实现了相减功能，那么类 B 就拥有相加和相减两个功能。

多态表示同一个成员函数在有继承关系的多个类中具有不同的功能实现，如类 B 继承了类 A，两个类都有 Calc 函数用于实现不同的功能，指针 p 若指向 A 对象，调用类 A 的 Calc 函数，若改为指向 B 对象，则调用类 B 的 Calc 函数。

2.1.2 定义类

使用 class 关键字定义一个类，可在类中添加成员变量（member variable）和函数。一般在类里声明函数，在类外实现函数以体现类的封装性，也可以声明和实现都在类中完成，适用于简短且频繁调用的成员函数。通过类名调用函数使用作用域解析操作符::，如 A::Calc(); ，通过对象名调用函数使用圆点操作符.，如 a.Calc(); 。

类成员有访问控制权限，分为公共成员（public）、保护成员（protected）、私有成员（private）三种。公共成员可在类外任意访问，保护成员只能在类中或派生类中访问，私有成员只能在本

类中访问。若没有设置访问控制符，默认为 private 成员。

【实例 2-1】定义一个类 Calc，用来对两个数进行相加、相减运算。

```
#include <iostream>
using namespace std;
class Calc                                          //定义类 Calc
{
private :                                           //私有成员，只能在类中访问
    int a;                                          //成员变量
    int b;
public:                                             //公有成员，可在类外访问
    void SetValue(int n1,int n2){                   //成员函数，设置成员变量值
        a=n1; b=n2;
    }
    int AddResult(){                                //相加运算
        return a+b;
    }
    int SubResult(){                                //相减运算
        return a-b;
    }
};
int main()
{
    Calc objCalc;                                   //创建类对象
    objCalc.SetValue(120,80);                       //调用公有成员函数，设置成员变量值
    cout<<"调用 AddResult : "<<objCalc.AddResult()<<endl;  //输出相加运算结果
    cout<<"调用 SubResult : "<<objCalc.SubResult()<<endl;  //输出相减运算结果
    return 0;
}
```

编译运行，结果如图 2-1 所示。用 class 关键字定义类 Calc，包括两个私有的成员变量 a、b，三个公有的成员函数，SetValue 函数用于设置两个变量的值，AddResult 函数用于求和，SubResult 函数用于求差。函数的声明和实现都在类中完成，私有变量只能在类中访问，在类外如 main 函数中无法访问。

使用类需要先创建一个类对象，如 Calc objCalc; 可将 Calc 看做一种新的数据类型，objCalc 是 Calc 类型的变量，通过圆点运算符.调用公有的成员。

图 2-1　定义类

类将变量和相关函数放在一起，提高了代码的聚合度，使用类时不需要了解类的具体实现，只需知道函数功能和使用方法，体现了面向对象的封装性。

2.1.3　构造函数

类不同于 C++的基本数据类型，类的结构可能非常复杂，包括大量的成员变量和函数，仅仅简单地分配一块足够大小的内存给类对象是不够的，还需要对类中的成员变量进行集中初始化。C++提供构造函数来完成类对象的初始化操作，构造函数先根据类的大小分配一块足够的内存用于存放类对象，再根据构造函数传入的参数和具体实现完成类对象的初始化。

构造函数名和类名相同，方便 C++编译器识别，若没有手动添加一个构造函数，C++自动为类生成一个没有参数的构造函数，仅仅完成内存的分配。构造函数没有返回值，可以重载多个构造函数，使用不同的参数列表，创建类对象时根据参数数目和类型自动调用匹配的构造函数。

【实例 2-2】定义一个类 Rect，使用不同的构造函数初始化矩形类对象，并计算矩形的面积。

```
#include <iostream>
using namespace std;
class Rect                                          //矩形类
```

```
{
private :                                                    //私有成员变量
    int x1,y1,x2,y2;                                         //左上角、右下角的 x、y 值
public:
    Rect(int leftTop[],int rightBottom[]){    //构造函数 1，参数为左上角、右下角坐标数组
        x1=leftTop[0];       y1=leftTop[1];
        x2=rightBottom[0];   y2=rightBottom[1];
    }
    Rect(int _x1,int _y1,int width,int height){ //构造函数 2，参数为左上角 x、y，宽度、高度
        x1=_x1;      y1=_y1;
        x2=_x1+width;    y2=_y1+height;
    }
    int Area();                                              //成员函数声明，计算矩形面积值
};
int Rect::Area()                                             //类体外实现
{
    return abs(x2-x1)*abs(y2-y1);                            //宽乘以高（取绝对值）
}
int main()
{
    int leftTop[]={12,15};                                   //整型数组
    int rightBottom[]={30,60};
    Rect r1(leftTop,rightBottom);                            //调用构造函数 1 初始化 r1
    Rect r2(12,15,75,60);                                    //调用构造函数 2 初始化 r2
    cout<<"r1 面积: "<<r1.Area()<<endl;                      //调用成员函数
    cout<<"r2 面积: "<<r2.Area()<<endl;
    return 0;
}
```

```
r1 面积: 810
r2 面积: 4500
Press any key to continue
```

图 2-2 构造函数

编译运行，结果如图 2-2 所示。手动添加两个构造函数后，C++不再提供默认的构造函数，构造函数 1 使用左上角、右下角的坐标初始化矩形，构造函数 2 使用左上角坐标、宽度、高度初始化矩形。Area 函数用于计算矩形面积，其中 abs 函数返回一个数的绝对值。

创建类对象时必须有匹配的构造函数，若调用的构造函数没有参数，则无须使用括号，如 Rect r;。若调用有参数的构造函数，在对象名后使用括号并传入参数，如 Rect r2(12,15,75,60);。构造函数仅在创建对象时调用，根据参数列表寻找匹配的构造函数。

2.1.4 析构函数

类在释放自身之前可能需要做一些清理工作，如类在使用过程中动态分配了内存空间，在类释放之前需要手动释放这些内存。析构函数用于完成类释放之前的一些清理工作，由 C++自动调用。

同构造函数，析构函数名为在类名前加~符号，如~Rect()，析构函数没有返回值也没有参数列表，每个类只有一个析构函数，若没有添加析构函数，C++提供默认的析构函数，但什么也不做。

【实例 2-3】定义一个字符串类 STR，在构造函数中动态创建字符数组，在析构函数里释放字符数组，并可根据索引获取某个字符。

```
#include <iostream>
using namespace std;
class STR                                                    //自定义的处理字符串类
{
private:
    char* pValue;                                            //字符指针，指向字符串
    int nCount;                                              //字符串长度，不包括末尾的'\0'
public:
    STR(char* str){                                          //构造函数
```

```
        nCount=strlen(str);                          //获取传入字符串的长度
        pValue=new char[nCount+1];                   //根据长度分配相应大小的内存
        strcpy(pValue,str);                          //将传入字符串的值复制到类成员变量中
        cout<<"动态创建字符数组"<<endl;
    }
    ~STR(){                                          //析构函数
        delete [] pValue;                            //释放动态创建的字符数组
        pValue=NULL;                                 //字符指针值设为 0
        cout<<"动态创建的字符数组被释放"<<endl;
    }
    char getValue(int nIndex){                       //获取单个字符值
        if(nIndex<0 || nIndex>=nCount)               //若索引参数超出了范围，返回'\0'
            return '\0';
        else
            return pValue[nIndex];                   //返回字符值
    }
};
int main()
{
    STR s1("heal the world");                        //创建类对象，同时调用构造函数初始化
    cout<<"第 6 个字符："<<s1.getValue(5)<<endl;          //调用成员函数，输出字符
    cout<<"第 20 个字符："<<s1.getValue(19)<<endl;        //索引不合理
    return 0;
}
```

编译运行，结果如图 2-3 所示。在构造函数中传入字符串，strlen 函数获取字符串的长度（不包括结束标志'\0'），动态创建一个字符数组，数组长度为字符串长度加 1（最后一位存放结束标志'\0'），strcpy 函数将字符串参数的值复制到字符数组中（包括结束标志'\0'）。

图 2-3　析构函数

strlen 函数获取字符串的长度，不包括结束标志'\0'，格式如下：

```
size_t strlen(char *str)
```

参数如下。

- str：指向字符串的字符指针。

返回值：字符串实际长度。

strcpy 函数将一个字符串的值复制到另一个字符串中，包括结束标志'\0'，格式如下：

```
char *strcpy(char *to, const char *from)
```

参数如下。

- to：接收字符串。
- from：被复制的字符串。

返回值：接收字符串 to。

getValue 函数根据索引获取某个字符，若索引小于 0 或超出范围，返回'\0'空字符。在对象 s1 超出作用域被释放之前，自动调用析构函数~STR，使用 delete 运算符释放字符数组，设置字符指针为 NULL，不指向任何位置，并输出一段文字表示调用了析构函数。

2.1.5　内联函数

程序代码中经常要调用函数，从调用点进入函数的内部，执行完毕后返回调用点，函数调用需要一定的开销。C 语言为节省函数调用带来的开销，常使用宏（macro）的方式模拟函数，如求和运算定义宏#define SUM(a,b) (a+b)，编译时用宏替换代码，减少函数调用的开销。

宏方式功能有限，不能定义数据类型，且容易出错，C++类提供了内联函数（inline）更好地解决这个问题，在类中声明成员函数时使用 inline 关键字表明为内联函数，调用时自动将函数

代码复制到函数调用处，这样既保留了函数的功能，又免去了调用开销。若成员函数的实现也在类体里完成，则默认为内联函数。

【实例 2-4】定义一个类 AB，声明一个内联函数。

```
#include <iostream>
using namespace std;
class AB
{
private:
    int a;
    int b;
public:
    inline int GetA();                          //inline 关键字声明为内联函数
    int GetB(){ return b; }                     //函数在类内实现，默认为内联函数
    void SetA(int _a){ a=_a; }                  //设置 a 的值
    void SetB(int _b){ b=_b; }                  //设置 b 的值
};
int AB::GetA(){                                 //内联函数，返回 a
    return a;
}
int main()
{
    AB ab;
    ab.SetA(100);
    ab.SetB(45);
    cout<<"a="<<ab.GetA()<<endl;                //内联函数，调用时直接将函数代码插入到该位置
    cout<<"b="<<ab.GetB()<<endl;
    return 0;
}
```

编译运行，结果如图 2-4 所示。函数 GetA 使用 inline 关键字声明为内联函数，调用时直接将函数代码复制到调用位置，函数 GetB 的实现在类体中完成，默认为内联函数。

Tips 若函数体简短且频繁调用，可声明为内联函数。若函数体复杂则不适合作为内联函数，此时相对于函数体的复杂运算，函数的调用开销无关紧要。

2.1.6 static 成员

图 2-4 内联函数

static（静态）关键字在不同场合具有不同的意义，也是常混淆用途的一个关键字（keyword）。在 C 语言中，static 有两层含义，若在函数外用 static 修饰全局变量和函数，表示具有文件作用域，只能在本文件中使用，不能在其他文件中使用。

【实例 2-5】新建一个.cpp 文件，使用 static 关键字修饰变量和函数，具体实现如下所示。

（1）选择 File|New 命令，添加一个 C++ Source File 名为 test，在 test.cpp 中输入以下代码，使用 static 关键字修饰全局变量和函数。

```
//test.cpp
static int sVar;                                //静态变量
static void sFun(){}                            //静态函数，仅在本文件中可用
int eVar;                                       //默认为所有文件可见
void eFun(){}
```

（2）在 main.cpp 中输入以下代码。调用外部定义的变量和函数。

```
#include <iostream>
```

```
using namespace std;
extern int eVar;                          //表明该变量已在其他文件中定义过
extern void eFun();                       //表明该函数已定义过
//extern void sFun();                     //静态函数，只在函数定义所在的文件中可用
int main()
{
    eFun();                               //调用外部定义的函数
    return 0;
}
```

extern（外部）关键字表明该变量或函数已经在其他文件中定义过，用于不同文件间的变量和函数传递，extern 修饰的变量不分配内存空间。若在文件 A 中定义了 int a; 在文件 B 中使用 extern int a; 即可使用 A 中定义的变量 a。函数默认为全局可见，在调用前包含函数声明所在的头文件即可。

若在函数内部使用 static 修饰局部变量，表示该变量永久存在，即使超出变量作用域后也不会被释放。变量第一次赋值时被初始化，其后保持值不变，直到被重新赋值。

【实例 2-6】在函数内部使用 static 局部变量，连续三次调用函数，测试变量的值。

```
#include <iostream>
using namespace std;
void display()                            //自定义函数
{
    static int num=1;                     //静态局部变量，永久存在
    for(int i=1;i<=num;i++)               //输出 1 到 num 间的值
        cout<<i<<" ";
    cout<<endl;
    num++;                                //静态局部变量递增
}
int main()
{
    display();                            //第 1 次调用，num 为 1
    display();                            //第 2 次调用，num 为 2
    display();                            //第 3 次调用，num 为 3
    return 0;
}
```

编译运行，结果如图 2-5 所示。第一次调用 display 函数时，静态局部变量 num 被初始化为 1，display 函数返回后 num 不会被释放，保持其值存在，第二次调用 display 函数时，不再执行初始化语句，num 为 2，第三次调用时 num 为 3。

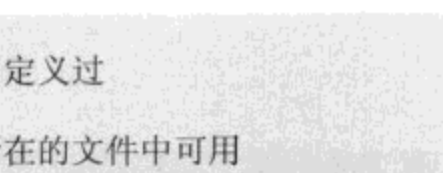


图 2-5　static 局部变量

在 C++类中，static 被赋予新的意义，用 static 修饰类成员变量，表示该变量不属于某个具体对象，所有对象共享同一份数据，都可以修改该变量的值，也可以直接用类名获取该变量。

【实例 2-7】在类中使用静态成员变量，在类外进行初始化，通过对象或类名访问静态成员变量。

```
#include <iostream>
using namespace std;
class STA
{
public:
    static int a;                         //静态成员变量，所有对象使用同一个值
};
int STA::a=0;                             //静态成员在类体外初始化
int main()
{
    STA s1,s2,s3;                         //创建类对象
    s1.a=1;                               //设置变量值
    cout<<"s2.a  "<<s2.a<<endl;           //查看值
```

```
    cout<<"STA.a "<<STA.a<<endl;                    //用类名获取变量值
    return 0;
}
```

```
s2.a  1
STA.a 1
Press any key to continue
```

图 2-6　static 成员变量

编译运行，结果如图 2-6 所示。static 成员变量必须在类外进行初始化，可使用对象名和类名访问静态成员变量，其中类名访问可使用 STA.a 和 STA::a 两种形式，static 成员变量适用于对所有类对象都一致的属性，或不需要创建对象，通过类名就能访问的变量。

用 static 修饰类成员函数，通过对象名和类名都可以调用静态成员函数。静态成员函数与具体类对象无关，在静态成员函数内部不能使用类的非 static 成员。

一般函数能够访问类对象的成员，在于 C++为成员函数传入一个 this 指针，指向调用函数的对象，这样被调用的函数就知道是哪个对象。静态成员函数没有 this 指针，不知道哪个对象在调用它，因此只能调用类的静态成员。

【实例 2-8】在类中定义 static 成员变量和函数，在 static 成员函数中使用 static 成员变量。

```
#include <iostream>
using namespace std;
class STA2
{
public:
    int b;                                          //普通成员变量
    static int a;                                   //静态成员变量
    static int GetA(){                              //静态成员函数，只能访问静态成员
        return a;
    }
    int GetB(){                                     //普通成员函数，有 this 指针指向调用的对象
        return this->b;
    }
};
int STA2::a=20;                                     //静态成员变量类外初始化
int main()
{
    STA2 s1;                                        //类对象
    s1.b=100;                                       //对公有成员赋值
    cout<<"静态函数  : "<<STA2::GetA()<<endl;       //用类名访问静态成员函数，使用::限定符
    cout<<"非静态函数: "<<s1.GetB()<<endl;          //用类对象访问普通成员函数
    return 0;
}
```

编译运行，结果如图 2-7 所示。普通成员函数有 this 指针指向调用对象，静态成员函数没有 this 指针，只能访问静态成员变量。静态成员函数适用于不需要创建类对象，使用类名就能访问的函数。

```
静态函数  : 20
非静态函数: 100
Press any key to continue
```

图 2-7　静态成员函数

2.1.7　const 成员

const（constant 常量）修饰符用于表示一个变量在初始化后不能再被修改，即只读（read only）。const 变量在定义时必须同时进行初始化，常用在函数参数中，表示该参数在函数内部不会被修改。

在传递类对象时，若采用传值方式，将类对象的拷贝传入函数，在类对象复杂的时候效率很低，一般传递类对象的引用，但若使用引用方式，类对象的值可能会被函数修改，这时使用 const 修饰符可限定类对象的值为只读，如 GetLength(const CString& str)。

若类成员函数使用 const 修饰符，表示该函数不能修改类成员变量的值。若类对象使用 const 修饰符，表示对象为只读，只能调用 const 成员函数。C++假定非 const 成员函数都会修改类成

员变量的值，为保证 const 对象不被修改，只能调用 const 成员函数。

使用 const 修饰指针类型时，若 const 在指针符号*之前，如 const int* p; 表示 p 指向的变量是只读的，不能通过指针 p 修改指向的变量的值。若 const 在*之后，如 int* const p; 表示 p 存储的指针值是只读的，不能修改指针 p 使其指向另一个对象。

【实例 2-9】在类中定义 const 类型的成员变量和函数，使用 const 类型的参数，创建 const 对象，用 const 对象调用 const 成员函数。

```
#include <iostream>
using namespace std;
class Constant
{
public:
    const int a;                                    //const 成员变量
    int b;
    Constant(int _a,int _b):a(_a){                  //构造函数，const 成员要用初始化列表
        b=_b;                                       //非 const 成员初始化
    }
    int Sum() const{                                //const 成员函数，不能修改成员变量值
        return a+b;
    }
    void SetB(const int _b){                        //const 参数在函数体内不能被修改
        this->b=_b;
    }
};
int main()
{
    Constant c1(20,30);                             //创建对象并初始化
    c1.SetB(50);                                    //非 const 对象调用非 const 成员函数
    cout<<"非 const 对象："<<c1.Sum()<<endl;         //非 const 对象调用 const 成员函数
    const Constant c2(20,30);                       //const 对象
    cout<<"const 对象：  "<<c2.Sum()<<endl;          //const 对象只能调用 const 成员函数
    return 0;
}
```

编译运行，结果如图 2-8 所示。const 成员变量 a 必须在构造函数的初始化列表中完成初始化，如:a(_a)，括号外的 a 为要初始化的变量，括号内的_a 为传入的参数，若有多个初始化变量，用逗号分隔，形式如下：

```
Constant(int _a,int _b,int _c):a(_a),b(_b),c(_c){}
```

Sum 函数为 const 成员函数，不能修改成员变量的值。SetB 函数的参数为 const 参数，不能在函数内部修改_b 的值。Constant 类对象 c1 可以调用非 const 成员函数和 const 成员函数，const 对象 c2 只能调用 const 成员函数。

图 2-8　const 成员

2.1.8　友元

类的私有成员一般情况下在类外不可访问，但有时候需要在类外访问，可使用 friend 关键字，在被访问类中声明为 friend 的类或函数可以访问其私有成员。

使用 friend 修饰的类、函数称为友元类、友元函数，友元不是类的成员，但能像类成员函数一样访问类的私有成员。当需要同时访问多个类的私有成员时，适宜使用友元。

【实例 2-10】定义三个类 FRI1、FRI2、Equal，其中 Equal 是 FRI1、FRI2 的友元类，可以访问两个类的私有成员，Equal 类的静态成员函数 equal 用于比较 FRI1、FRI2 两个类对象的私有成员变量值是否相同。

```
#include <iostream>
using namespace std;
class Equal;                                    //友元类，在定义前使用需要做类声明
class FRI1
{
private:
    int a;                                      //私有成员
public:
    friend class Equal;                         //类 Equal 是友元类，可以访问 FRI1 类的私有成员
    FRI1(int _a):a(_a){}                        //构造函数
};
class FRI2
{
private:
    int a;
public:
    friend class Equal;                         //类 Equal 是友元类，可以访问 FRI2 类的私有成员
    FRI2(int _a):a(_a){}
};
class Equal
{
public:                                         //静态函数，可直接用类名调用，比较两个类的值是否相等
    static bool equal(const FRI1& f1,const FRI2& f2){
        if(f1.a==f2.a)                          //若 FRI1 类对象的私有成员 a 等于 FRI2 类对象的私有成员
            return true;                        //返回 true
        else
            return false;
    }
};
int main()
{
    FRI1 f1(20);
    FRI2 f2(30);
    cout<<"f1 和 f2 的值是否相等："<<Equal::equal(f1,f2)<<endl;
    return 0;
}
```

图 2-9　友元类

编译运行，结果如图 2-9 所示。在两个类中添加 friend class Equal; 将类 Equal 作为 FRI1、FRI2 类的友元类，可以直接访问这两个类的私有成员。类 Equal 的定义在两个类定义之后，在使用前需要先做类声明，表明类 Equal 在其后有定义。

静态成员函数 equal 可以在函数内部访问 FRI1、FRI2 类的私有成员变量，比较两个类对象的成员变量值是否相等，若相等返回 true，否则返回 false。equal 的两个参数为类的 const 引用，调用时传入两个对象的引用，const 表示不能在函数内部修改类对象的值。调用静态成员函数 equal 时无须创建类对象，可直接用类名访问。

2.2 运算符重载

C++提供一些运算符用于完成基本的运算，如+可以计算两个整型或浮点类型数值的和，==可以判断两个值是否相等，但都只能用于预置的基本类型。若想用+得到两个字符串拼接后的新字符串，用==判断两个类是否包含相同的值，可使用 C++提供的运算符重载功能，类似函数重载，运算符可看做函数名，运算符两边的变量可看做函数的参数，调用时根据运算符的参数数目和类型自动选择匹配的版本。

2.2.1　了解运算符重载

运算符重载在定义时使用 operator 关键字，根据运算符操作数的数目分为一元重载和二元重载，如+、/、%为二元运算符，++、--为一元运算符，-当做减号为二元运算符，当做负号为一元运算符。

运算符可看做为函数，如 operator +类似 Add(STR a,int b); +两边的操作数类似参数 a、b，重载的运算符可作为类的成员，也可作为类的友元函数，还可作为普通函数。

若作为类成员，运算符左边的参数为类对象，右边可为其他类型的参数或为空。若作为友元函数，可访问类的私有成员，参数顺序不固定。若为普通函数，不可访问私有及保护成员，参数顺序不固定。

只能重载限定范围内的已存在的运算符，不可自己创建一个运算符，圆点（.）、作用域（::）、条件（?:）运算符不能被重载。不能改变操作符原有的操作数数目，也不能改变操作符的优先级。

运算符重载是为了提高程序的易用直观性，若重载后操作反而难懂、复杂，就失去了重载的意义，重载的运算符都可以用函数形式代替。

2.2.2　一元重载

一元重载只有一个操作数即类名，一般作为类成员，常用的一元运算符有自增、自减、求负运算符，其中自增、自减运算有前后之分，运算符在前表示先自增自减，然后再赋值，在后表示先赋值然后再自增自减，若为单独的表达式则没有区别。默认重载++、--为前缀版本，若表示后缀版本，添加一个 int 参数。

【实例 2-11】定义类 IntOper，重载++、--运算符，分为前缀、后缀两种版本，实现自定义类的一元运算符重载。

```
#include <iostream>
using namespace std;
class IntOper
{
private:
    int a;                                  //私有成员
public:
    void SetA(int _a){                      //设置 a 的值
        a=_a;
    }
    int GetA(){                             //获取 a 的值
        return a;
    }
    int operator ++(){                      //默认为前缀，先自增，后赋值
        a=a+1;
        return a;
    }
    int operator ++(int){                   //添加 int 参数，表示后缀，先赋值，后自增
        a=a+1;
        return a-1;
    }
    int operator --(){                      //默认为前缀，先自减，后赋值
        a=a-1;
        return a;
    }
    int operator --(int){                   //添加 int 参数，表示后缀，先赋值，后自减
        a=a-1;
        return a+1;
    }
```

```
};
int main()
{
    IntOper oper;
    oper.SetA(20);                                    //设置值
    cout<<"oper   : "<<oper.GetA()<<endl;             //获取当前值
    cout<<"++oper : "<<++oper<<endl;                  //前缀自增
    cout<<"oper++ : "<<oper++<<endl;                  //后缀自增
    cout<<"--oper : "<<--oper<<endl;                  //前缀自减
    cout<<"oper-- : "<<oper--<<endl;                  //后缀自减
    return 0;
}
```

编译运行，结果如图 2-10 所示。默认状态下重载的++、--为前缀版本，添加的 int 参数是个标志，用来表明要重载后缀版本。建议将自增自减作为一个单独的表达式，以免混淆。

图 2-10 一元重载

2.2.3 二元重载

二元重载有两个操作数，若作为类成员，只需一个参数，否则需要两个参数。参数的顺序代表操作数的位置，如 operator +(A a,int b){} 调用形式为 a+2; 不能为 2+a; 。若调换位置也可用，要再重载一次如 operator +(int b,A a){}。

【实例 2-12】定义类 IntArray，用来操作一个整型数组，重载[]运算符后，可通过在[]中输入下标索引得到某个元素的值，可设置某个元素的值、获取数组长度。

```
#include <iostream>
using namespace std;
class IntArray
{
private:
    int* pInt;                                        //整型指针，指向动态创建的数组
    int nCount;                                       //元素数目
public:
    IntArray(int count){                              //构造函数，根据传入参数创建相应大小的数组
        nCount=count;
        pInt=new int[nCount];
    }
    ~IntArray(){                                      //析构函数，释放动态创建的数组
        delete [] pInt;
        pInt=NULL;
    }
    int operator[](int index) const{                  //重载[]，参数为下标索引
        if(index<0 || index>=nCount)                  //若参数不合理，返回 0
            return 0;
        return pInt[index];                           //返回对应位置的元素值
    }
    void SetAt(int index,int value){                  //设置元素值，参数为索引和元素值
        if(index<0 || index>=nCount)                  //若索引不合理，直接返回
            return;
        pInt[index]=value;                            //设置对应位置的元素值
    }
    int GetLength(){                                  //获取元素数目
        return nCount;
    }
};
int main()
{
    IntArray arr(3);                                  //创建对象，元素数目为 3
    arr.SetAt(0,12);                                  //设置每个元素值
```

```
    arr.SetAt(1,45);
    arr.SetAt(2,80);
    for(int i=0;i<arr.GetLength();i++)                    //输出所有元素
    {
        cout<<"第"<<i+1<<"个元素："<<arr[i]<<endl;//调用重载的[]运算符，返回第 i 个元素值
    }
    cout<<endl;
    return 0;
}
```

图 2-11　二元重载

编译运行，结果如图 2-11 所示。创建类对象 arr 时，调用构造函数并传入元素数目 3，在构造函数中动态创建一个大小为 3 的整型数组，整型指针 pInt 指向该数组。SetAt 函数设置每个元素的值，参数 1 为元素索引，参数 2 为元素值，若索引小于 0 或超出数组范围，直接返回。GetLength 函数获取数组元素的数目。

通过重载运算符[]，自定义的类 IntArray 可通过在[]中传入元素索引，获取某个元素的值，若元素索引小于 0 或超出数组的范围，返回 0。使用[]运算符的效果类似于 GetAt(int index)函数，但形式上更为直观简洁。

2.3 继承性

面向对象编程的一个重要特性是继承性，类之间可以有继承关系，通过从一个父类派生出子类，子类在拥有父类功能的基础上，添加新增的功能，无须从零开始构建整个类，只需要补充新增的功能即可。C++允许多重继承，即从多个父类派生出一个子类，子类拥有多个父类的功能，多重继承虽功能强大，却复杂难以理解，在 C#、Java 中已取消了多重继承，应避免使用。

MFC 类库中绝大多数类都继承于 CObject 类，CObject 类提供了基本的服务，如运行时（runtime）的类型判断、对象的序列化存储等，所有从 CObject 类继承的类自动具有这些功能。

2.3.1 类的继承

父类称为基类，从基类继承的子类称为派生类，在类定义时指定要继承的基类，派生类自动获取基类的成员变量和函数，但不会获取基类的构造函数和析构函数。

创建派生类对象时，首先要初始化基类，可在派生类的构造函数里调用基类的构造函数，若没有显式调用基类的构造函数，自动调用基类的默认无参数版本构造函数。

【实例 2-13】定义基类 Person，派生类 Student 继承于 Person，在派生类的构造函数里调用基类的构造函数，并重定义基类的 display 函数。

```
#include <iostream>
#include <string>                                          //包含 string 类头文件
using namespace std;
class Person                                               //基类
{
protected:                                                 //保护成员，派生类中可访问
    int nYear;                                             //年龄
    string sex;                                            //性别
    string name;                                           //名称
public:
    Person(){}                                             //无参构造函数
    Person(int _year,string _sex,string _name){
        nYear=_year;                                       //有参构造函数，初始化成员变量
        sex=_sex;
        name=_name;
    }
```

```
    void display(){                                          //输出成员，基类版本
        cout<<"姓名："<<name<<endl;
        cout<<"年龄："<<nYear<<endl;
        cout<<"性别："<<sex<<endl;
    }
};
class Student:public Person                                  //派生类，公有继承于类 Person
{
protected:                                                   //保护成员
    string school;                                           //学校
    string number;                                           //学号
public:
    Student(int _year,string _sex,string _name,string _school,string _number)
        :Person(_year,_sex,_name){                           //调用基类的构造函数
        school=_school;                                      //初始化新增成员变量
        number=_number;
    }
    void display()                                           //输出所有成员，派生类版本，函数重定义
    {
        Person::display();                                   //调用基类版本函数
        cout<<"学校："<<school<<endl;                        //输出
        cout<<"学号："<<number<<endl;
    }
};
int main()
{
    Student s1(1985,"男","Luo","hpu","0216");                //创建类对象并初始化
    s1.display();                                            //调用派生类版本输出函数
    return 0;
}
```

编译运行，结果如图 2-12 所示。基类 Person 定义了三个保护成员变量，一个公有的 display 函数用于输出三个变量，并重载一个有参的构造函数用于初始化基类对象。派生类 Student 继承了基类 Person，自动拥有基类的三个变量和 display 函数，并在自定义的有参构造函数中，调用基类的有参构造函数初始化基类成员，同时初始化新增的两个变量。

图 2-12　类继承

string 类是 C++自带的可变字符串类，相对于字符数组，string 类更加简单稳定，有关字符串的操作可用 string 类完成，使用前要包含其头文件，在 MFC 环境下，使用自带的 CString 类替代 string 类。

创建一个 string 类对象后，可直接将一个字符串赋给 string 对象，形式如下：

```
string str1="beautiful ";                                    //将字符串赋给 string 类对象 str1
string str2("girl");                                         //在构造函数里传入初始字符串
string str3=str1+str2;                                       //str3 为 str1 和 str2 字符串的拼接值
for(int i=0;i<str3.size();i++)                               //遍历 str3 所有字符元素
  cout<<str3[i];                                             //输出第 i 个字符
```

string 类常用函数如下所示。

- c_str：返回 C 形式字符串指针（const char *）。
- compare：比较本字符串和目标字符串。
- empty：判断字符串是否为空。
- erase：删除指定位置开始的 *n* 个元素。
- find：查找字符或字符串出现的位置。
- size：获取字符串长度。
- substr：获取从指定位置开始的长度为 *n* 的子字符串。

在派生类中重定义了基类的 display 函数，不同于函数重载，重定义的函数具有相同的函数名、参数列表，类对象根据实际情况调用其对应版本的函数。

若派生类重定义了 display 函数，则 s1.display(); 调用派生类版本的 display 函数，若派生类未进行重定义，则 s1.display(); 调用基类版本的 display 函数。

若派生类重定义了 display 函数，要调用基类版本的 display 函数，可用 s1.Person:: display(); 或者在派生类函数内部使用 Person::display(); 限定调用基类版本的 display 函数。基类对象只能调用基类版本的 display 函数。

2.3.2　访问控制

类继承有三种方式：公有（public）、保护（protected）、私有（private）继承。基类的私有成员无论为何种继承方式，在派生类中都不可访问。

若为私有继承，基类的公有和保护成员在派生类中变为私有。若为保护继承，基类的公有和保护成员在派生类中变为保护成员。若为公有继承，基类的公有成员和保护成员在派生类中保持不变，一般情况下只使用 public 即公有继承方式。

2.3.3　调用流程

基类的构造函数和析构函数不会被派生类继承。若类 B 继承类 A，类 C 继承类 B，则创建一个类 C 对象时，先调用类 A 的构造函数，然后类 B，再调用类 C 的构造函数，若类 C 对象释放时，先调用类 C 的析构函数，然后类 B，最后调用类 A 的析构函数。即构造函数调用顺序为从基类到派生类，析构函数与构造函数顺序相反，从派生类到基类。

【实例 2-14】定义类 A、B、C，其中类 B 继承类 A，类 C 继承类 B，创建一个类 C 对象，测试类函数的调用流程。

```
#include <iostream>
using namespace std;
class A                                          //类A
{
public:
    A(){                                         //构造函数
        cout<<"构造 A"<<endl;
    }
    ~A(){                                        //析构函数
        cout<<"析构 A"<<endl;
    }
};
class B:public A                                 //类B派生自类A
{
public:
    B(){
        cout<<" 构造 B"<<endl;
    }
    ~B(){
        cout<<" 析构 B"<<endl;
    }
};
class C:public B                                 //类C派生自类B
{
public:
    C(){
        cout<<"  构造 C"<<endl;
    }
    ~C(){
```

```
        cout<<"  析构 C"<<endl;
    }
};

int main()
{
    C c1;                                    //类 C 对象，创建时调用构造函数，释放时调用析构函数
    return 0;
}
```

编译运行，结果如图 2-13 所示。创建类 C 对象 c1 时，调用构造函数进行初始化，按照从基类到派生类的顺序，先调用类 A 的构造函数初始化成员，再调用类 B、类 C 的构造函数初始化各个成员。

当类对象 c1 超出作用域被释放时，调用析构函数进行清理工作，按照从派生类到基类的顺序，先调用类 C 的析构函数，再调用类 B、类 A 的析构函数。

图 2-13　类调用顺序

2.4 多态性

用类指针去调用类成员函数时，根据指针的类型即可确定调用的函数版本。如类 B 继承类 A，类 B 重定义了类 A 的 Calc 函数，若有类 A 指针 p 调用 Calc 函数，调用了类 A 版本的 Calc 函数，若将类 A 指针 p 改为指向类 B 对象（指针类型没有变），再调用 Calc 函数，仍调用类 A 版本的 Calc 函数。重定义函数的调用仅与指针类型相关，与指针实际指向对象无关。

在类继承关系复杂的程序中，若在调用每个类的函数时，都要定义一个指向该类的指针，是不实际的。需要一个统一的调用方式，当指针指向某个对象时，就调用该对象所在的类版本函数，如 p 指向类 A 对象，调用类 A 版本的 Calc 函数，若改为指向类 B 对象，调用类 B 版本的 Calc 函数，调用方式始终为 p->Calc(); 这就是 C++的多态性，能够在程序运行时，根据指针实际所指对象调用对应版本的函数。

2.4.1 多态性的实现

多态性与函数重定义的区别在于：重定义函数的调用与指针类型相关，在程序编译时就确定了调用的函数版本，多态性的函数调用与指针实际所指对象相关，在程序运行时才确定调用的函数版本。

多态性通过一种晚期绑定的方式实现运行时的识别，实现晚期绑定可使用 virtual 关键字声明一个函数是虚函数，只需要在基类中声明函数为虚函数，在派生类中对该函数重写后，即可实现动态绑定。

【实例 2-15】定义类 A、类 B，其中类 B 继承类 A，在类 A 中声明 Calc 函数为虚函数，在类 B 中重写 Calc 函数，运行时根据指针 p 实际所指对象调用对应版本的 Calc 函数。

```
#include <iostream>
using namespace std;
class A                                      //基类
{
public:
    virtual void Calc(){                     //使用 virtual 关键字，表明为虚函数
        cout<<"类 A 版本的 Calc"<<endl;
    }
};
class B:public A                             //派生类
{
public:
    void Calc(){                             //重写虚函数
```

```
        cout<<"类 B 版本的 Calc"<<endl;
    }
};
int main()
{
    A a;                                    //类 A 对象
    B b;                                    //类 B 对象
    A* p=&a;                                //类 A 指针，指向 A 对象
    p->Calc();                              //调用类 A 版本的 Calc
    p=&b;                                   //指针改为指向类 B 对象
    p->Calc();                              //调用类 B 版本的 Calc
    return 0;
}
```

编译运行，结果如图 2-14 所示。通过 virtual 关键字声明 Calc 函数是一个虚函数，在派生类中重写的 Calc 也自动为一个虚函数，只要声明为虚函数，在实际调用时将根据指针实际所指对象，调用对应版本的函数，如 p 指向对象 a 时，调用类 A 版本的 Calc 函数，p 改为指向对象 b 时，调用类 B 版本的 Calc 函数。

图 2-14　多态性

若去掉 virtual 关键字，则两次执行 p->Calc(); 都将调用类 A 版本的 Calc 函数，对于函数重定义，调用的函数版本只跟指针类型相关。

当通过指针调用函数时，如 p->Calc(); ，由于不确定实际调用对象，需要使用多态性进行晚期绑定。若通过类对象调用函数，如 s1.display(); ，由于对象类型明确，不需要使用多态性。

2.4.2　virtual 虚函数

添加一个 virtual 关键字就可以实现动态绑定，在于编译器为每个包含虚函数的类生成一个虚函数表（virtual table），同时在类对象的内存空间的头部添加一个虚指针，指向生成的虚函数表。虚函数表存放该类所有的虚函数，若在派生类中重写了基类的虚函数，则派生类的虚函数表中存放的是重写后的虚函数。

当类指针指向基类对象时，通过虚指针获取基类虚表中的虚函数地址，即基类版本的函数。当类指针改为指向派生类对象时，通过虚指针获取派生类虚表中重写后的函数地址，即派生类版本的函数。过多的虚函数会影响程序的执行效率，尤其是继承关系复杂的类中，应根据情况决定是否将函数设为虚函数。

若基类的析构函数设为 virtual，派生类的所有析构函数自动成为虚析构函数。若有基类指针 p 指向动态创建的派生类对象 b，如 A* p=&b; ，由于析构函数是虚函数，使用 delete p; ，则先调用派生类的虚析构函数，再自动调用基类 A 的析构函数。若析构函数不是虚函数，使用 delete p; ，则仅调用基类的析构函数，不调用派生类的析构函数，建议将基类的析构函数声明为 virtual。

2.4.3　抽象类

在实际开发中，可能某些基类只是做个规范，形式上确定有哪些基本成员及其功能，并不涉及具体的实现。如基类 CMap 定义有虚函数 Draw()，作为所有图形类的绘制函数，基类下有派生类 CLine、CPoint、CRect 分别用于绘制线、点、矩形，具体的图形绘制方法在派生类的 Draw 函数里完成，而基类的 Draw 只是一个接口的规范，不需要具体实现，此时可设置 Draw 函数为纯虚函数。

纯虚函数只有函数声明，没有具体实现，如 virtual void Calc()=0; 包含纯虚函数的类称为抽象类（abstract class），抽象类不能用于创建类对象，主要用于对类功能的描述，类似于 C#、Java 中的接口（interface），其派生类必须重写所有纯虚函数。

【实例 2-16】定义类 A、类 B，其中类 B 继承类 A，在类 A 中定义纯虚函数 Calc，在类 B 中实现该纯虚函数。

```
#include <iostream>
using namespace std;
class A                                          //包含纯虚函数，为抽象类
{
public:
    virtual void Calc()=0;                       //纯虚函数，没有函数体，添加=0
};
class B:public A                                 //继承抽象类
{
public:
    void Calc(){                                 //实现抽象类所定义的纯虚函数
        cout<<"类 B 版本的 Calc"<<endl;
    }
};
int main()
{
    B b;                                         //创建类对象
    A* p=&b;                                     //基类指针，指向派生类对象
    p->Calc();                                   //多态性，调用派生类函数
    return 0;
}
```

编译运行，结果如图 2-15 所示。纯虚函数没有{}包括的函数体，且在()后面添加=0 标志，表明是纯虚函数。派生类必须实现抽象类定义的所有纯虚函数，抽象类不能用于创建对象，其他方面与普通类没有区别。

图 2-15 纯虚函数

2.5 模板

标准 C++库除了包含标准 C 函数库、IO 输入/输出类、string 字符串类之外，还提供一套强大的通用类和算法库，即标准模板库 STL（Standard Template Library），包括常见的数据结构和算法，如链表、栈、队列等。

得益于 C++的模板（Template）机制，STL 适用于任何数据类型，如链表 list 既可存放整型，也可存放 string 类型。模板也被称为泛型编程，忽略实际的数据类型，使用模板的函数和类可以适用于多种数据类型，从而节省工作量，提高代码重用性。

2.5.1 如何定义模板

在定义类或函数时使用模板可以忽略实际数据类型，调用时根据实际的数据类型，生成一个函数或类的具体定义。通过在函数或类的定义前添加 template<class T>，表明定义的函数或类使用了模板机制，class 是类型关键字，T 是一个未知的数据类型，调用时被实际类型替换。若有两个模板参数，形式为 template<class T1,class T2>，T1 和 T2 为两个未知的数据类型。

【实例 2-17】定义模板函数 Sum，用于不同数据类型的求和。

```
#include <iostream>
using namespace std;
template<class T>                                //模板声明，T 为参数类型
T Sum(T a,T b)                                   //模板函数，计算两个未知类型的和
{
    return a+b;
}

int main()
{
    int a=10,b=50;
    double c=15.2,d=45.3;
    char e='0',f='1';
    cout<<"int     "<<Sum(a,b)<<endl;            //整型参数
    cout<<"double  "<<Sum(c,d)<<endl;            //浮点型参数
```

```
    cout<<"char    "<<Sum(e,f)<<endl;      //字符型参数
    return 0;
}
```

编译运行，结果如图 2-16 所示。在函数 Sum 定义位置前使用 template<class T>，表明该函数是一个模板函数，在函数内部可用未知类型 T 进行相关运算。调用 Sum 函数时，用参数的实际类型替换 T，如 int、double、char，系统自动生成一个有实际类型的 Sum 函数用于求和。

图 2-16　函数模板

若有几个函数功能相同，仅仅数据类型不一致，可用一个模板函数替代，定义时使用未知类型，调用时再用实际类型替换未知类型。

2.5.2　模板类

模板类是泛型程序设计的基础，可以处理多种类型的数据，将通用功能集成到模板类中，可节省大量重复性工作。类似于函数模板，定义模板类之前使用 template<class T>做模板声明，在类中可用未知类型 T 进行相关处理。

template<class T>只能在其后的类或函数中使用一次，若要在多个类或函数中使用模板，需要各自在定义前声明一次。传入实际数据类型时，形式为 Calc<int> c1(12,35); 在<>中输入数据类型。

【实例 2-18】定义模板类 Calc，在类外实现模板成员函数，可用于计算不同数据类型的加减运算。

```
#include <iostream>
using namespace std;
template<class T>                              //模板声明
class Calc                                     //模板类
{
private:
    T a;                                       //成员变量，未知类型
    T b;
public:
    Calc(T _a,T _b):a(_a),b(_b){}              //构造函数
    T Add();                                   //成员函数
    T Sub();
};
template<class T>                              //模板函数
T Calc<T>::Add(){                              //类外实现
    return a+b;
}
template<class T>
T Calc<T>::Sub(){
    return a-b;
}
int main()
{
    Calc<int> c1(12,35);                       //模板类对象，指明对象类型为 int
    Calc<double> c2(35.8,15.3);                //指明对象类型为 double
    cout<<"int    "<<c1.Add()<<endl;
    cout<<"double "<<c2.Sub()<<endl;
    return 0;
}
```

编译运行，结果如图 2-17 所示。一个模板声明只针对其后的一个函数或类，若有多个模板函数或类，需要添加各自的模板声明。模板类使用时必须使用<>并传入数据类型，表明类具体的类型，如 Calc<T>::Add 和 Calc<int> c1。

图 2-17　模板类

2.5.3 标准模板库 STL

STL 是标准 C++库的一部分，包含链表（list）、向量（vector）、栈（stack）、队列（queue）等常见数据结构，STL 中的类都是模板类，适用于任意数据类型，也称为容器类。STL 使用迭代器（iterator）遍历容器中的元素，迭代器类似于指针，根据迭代器变量可查找到容器中的某个元素。

通过迭代器重载的运算符可操作迭代器，如++用于将迭代器指向下一个元素，--用于指向上一个元素，==和!=用于判断两个迭代器是否指向同一个元素，*用于获取迭代器指向元素的引用。

向量 vector 是一个长度可变数组，在运行时可改变长度，定义 vector 类型变量时，需指定存放的数据类型，索引编号从 0 开始，可通过[]传入索引读取或设置元素值。向量类提供的成员函数如下。

- push_back：在尾部添加一个元素。
- pop_back：删除尾部元素。
- insert：在指定位置插入元素。
- erase：删除指定位置元素。
- begin：返回第一个元素的迭代器。
- end：返回结束标志。
- at：返回指定位置元素。
- size：返回元素数目。
- clear：清空所有元素。

【实例 2-19】定义一个存放 string 类型的 vector 类对象，添加三个元素，输出数组大小，使用迭代器遍历数组，输出所有元素的值。

```
#include <iostream>
#include <vector>                                          //包含 vector 头文件
#include <string>                                          //包含 string 头文件
using namespace std;
int main()
{
    vector<string> v;                                      //存放 string 类型的 vector
    v.push_back("1.Hello");                                //尾部添加 string 字符串
    v.push_back("2.Vector");
    v.push_back("3.Test");
    cout<<"元素数目: "<<v.size()<<endl;                    //元素数目
    for(vector<string>::iterator p=v.begin();p!=v.end();p++)  //使用迭代器遍历向量
        cout<<*p<<endl;                                    //获取迭代器指向的值
    cout<<"第 1 个元素: "<<v[0]<<endl;                     //使用[]获取元素值
    return 0;
}
```

编译运行，结果如图 2-18 所示。创建 vector 类对象 v 时，要传入数据元素的类型 string，push_back 函数用于在尾部添加一个元素，size 函数获取数组元素的数目。

图 2-18 vector

不同的容器类有不同的迭代器，使用迭代器访问 vector 对象，要定义当前类型的迭代器，如 vector<string>::iterator，若数据类型为 int，则迭代器形式为 vector<int>::iterator。

begin 函数返回第一个元素的迭代器，end 返回结束标志，p++将迭代器指向下一个元素，使用 for 循环和迭代器遍历所有元素，*p 获取迭代器指向元素的引用。

链表 list 也可存储可变长度的元素，不同于 vector 中的元素连续存储，链表中的元素是随意存储的，通过元素中的指针成员确定前后元素的位置。相对于 vector，链表可以快速地插入和删除，但随机访问较慢，链表类提供的成员函数如下。

- sort：对链表进行排序。

- unique：删除链表中重复的元素。
- push_back：在尾部添加一个元素。
- push_front：在头部添加一个元素。
- pop_back：删除尾部元素。
- pop_front：删除第一个元素。

【实例 2-20】定义一个存放 int 类型的 list 类对象，添加并输出 10 个随机的元素值，再输出排序和去除重复值后的所有元素。

```
#include <iostream>
#include <list>                                    //包含 list 头文件
using namespace std;
int main()
{
    list<int> ls;                                  //存放 int 类型的 list
    cout<<"所有元素：  ";
    for(int i=1;i<=10;i++)                         //添加 10 个元素
    {
        int value=rand()%10;                       //获取 0～9 的随机值
        ls.push_back(value);                       //添加到 list 尾部
        cout<<value<<" ";                          //输出当前元素
    }
    cout<<endl<<"排序后：   ";
    ls.sort();                                     //list 排序，默认为升序
    list<int>::iterator p;                         //当前类型的迭代器
    for(p=ls.begin();p!=ls.end();p++)              //遍历 list
        cout<<*p<<" ";
    cout<<endl<<"去除重复元素：";
    ls.unique();                                   //list 去除重复项
    for(p=ls.begin();p!=ls.end();p++)
        cout<<*p<<" ";
    cout<<endl;
    return 0;
}
```

编译运行，结果如图 2-19 所示。创建 list 对象 ls 时，要传入实际的数据类型 int。push_back 函数在尾部添加一个元素。sort 函数执行排序，默认为升序。unique 函数删除重复项。

图 2-19　list

Tips　unique 函数只删除相邻元素的重复项，使用前应先调用 sort 函数排序。

rand 函数返回一个 int 类型的伪随机值，即每次启动程序产生的随机数都是一样的，若要生成不一样的随机数，可用 srand 函数设置随机种子，rand 函数根据不同的随机种子生成不同的随机数，一般将当前时间作为随机数种子，如 srand(time(NULL));。

使用迭代器 p 访问 list 类对象时，要定义当前类型的迭代器，如 list<int>::iterator p; begin 函数返回第一个元素的迭代器，end 函数返回结束标志，p++将迭代器指向下一个元素，使用 for 循环和迭代器遍历所有元素，*p 获取迭代器指向元素的引用。

2.6 异常处理

程序在执行过程中时常会发生一些异常（exception），如打开文件时找不到指定文件、分配内存时内存不足、除法运算中分母为 0、连接不上数据库等，有些异常可通过代码判断，但有些

是难以预测的。为保证程序运行时能够正确处理各种异常情况，不会直接崩溃退出，C++提供异常处理机制，可以获取运行时的异常情况，并提供错误处理方法，从而提高程序的健壮性。

2.6.1 处理程序异常

C++提供 try 和 catch 关键字处理异常情况，将正常执行的代码放入 try{}语句块中，catch{}块中放入处理异常的代码。若无异常发生，只执行 try 块中的语句，若发生了异常，停止执行 try 块中剩下的语句，直接跳入 catch 块中执行异常处理代码，使用 try 和 catch 可以将正常代码和异常代码区分开来，增强代码的可读性。

throw 关键字用于抛出异常，通常在函数 A 中抛出异常，函数 B 在调用 A 时捕获 A 抛出的异常。可抛出任意类型的一个值，catch 块捕获的类型必须和抛出的类型一致，才能捕获该异常，可以有多个 catch 语句块，用于捕获不同类型的异常。

【实例 2-21】定义函数 divide 抛出 int 类型的异常，在 main 函数中使用 try 和 catch 捕获该异常，并输出异常提示信息。

```
#include <iostream>
using namespace std;
double divide(double a,double b)          //可抛出异常的函数
{
    if(b==0)                              //若分母为 0，抛出整型异常
        throw 0;
    return a/b;
}
int main()
{
    double a=12.5;
    double b=0;
    double result;
    try{                                  //异常处理
        result=divide(a,b);               //调用函数
        cout<<"结果是"<<result;
    }
    catch(int e){                         //捕获函数 divide 抛出的异常，参数类型与抛出类型一致
        cout<<"分母不能为"<<e<<endl;      //异常处理语句
    }
    return 0;
}
```

图 2-20 捕获异常

编译运行，结果如图 2-20 所示。divide 函数用于计算两个浮点数的相除结果，若分母为 0，使用 throw 关键字抛出 int 类型的异常。main 函数中使用 try 和 catch 添加异常处理功能，在 try 块中放入正常代码。

当调用 divide 函数时，由于 b 为 0，导致 divide 函数抛出 int 类型异常，catch 块根据参数类型捕获对应类型的异常，如 catch(int e)只捕获 int 类型的异常，异常抛出后，停止 try 块中后面的语句，跳入对应的 catch 块中，输出一段提示信息，e 为 throw 抛出的值 0。

2.6.2 自定义异常类

throw 可抛出任意类型的异常，可自定义一个异常类，用以提示更多的错误信息。MFC 类库提供一系列异常类，用以抛出不同类型的异常，如内存不足抛出 CMemoryException 异常、文件错误抛出 CFileException 异常。

【实例 2-22】自定义一个异常类 Exception，类成员函数 GetMessage 获取错误信息，在 try

块中抛出 Exception 类异常，在 catch 块中捕获该异常。

```
#include <iostream>
#include <string>
using namespace std;
class Exception                                    //自定义异常类
{
private:
    string message;                                //错误提示信息
public:
    Exception(){                                   //构造函数
        message="自定义的异常类";
    }
    string GetMessage(){                           //获取错误信息
        return message;
    }
};
int main()
{
    try{
        int b=0;
        if(b==0)
            throw Exception();                     //抛出自定义类异常
    }
    catch (Exception e){                           //捕获抛出的自定义类异常
        cout<<e.GetMessage()<<endl;                //输出自定义类的错误信息
    }
    return 0;
}
```

编译运行，结果如图 2-21 所示。Exception 为自定义的异常类，在构造函数中初始化异常信息，GetMessage 函数获取异常信息。try 块中使用 throw 抛出一个 Exception 类型的异常，catch 块捕获对应类型的异常，e 为抛出的 Exception 类对象，调用 GetMessage 函数输出异常信息。

图 2-21　自定义异常类

2.7 小结

C++语言的优势是其具有面向对象的特性，并结合面向过程编程，最大程度发挥语言的特色。本章讲解了与类有关的知识，如三大特性：封装、继承和多态，分别列举实例，帮助读者真正理解其内涵。最后对模板和异常处理进行了详细介绍。希望读者通过本章的学习，充分掌握面向对象程序设计的精髓。

2.8 习题

1. 类和对象之间是什么关系？
2. 如何定义一个简单的类？
3. 什么是友元函数？什么时候会用到友元函数？
4. C++编程的重要特性是什么？
5. 如何捕获异常？

第3章　掌握开发环境

早期的软件开发基于命令行（command line）模式，调用各种命令完成代码的编译（compile）、链接（link）、运行（execute）及环境的配置（config）等，操作复杂且不直观。Visual C++提供可视化的开发界面，所有操作均可在开发环境中完成，极大地提高了软件开发的效率，“工欲善其事，必先利其器”，熟练操作开发平台是学习 Visual C++的第一步。

3.1 创建运行程序

在 Visual C++中创建一个桌面程序，有 Win32 API 和 MFC 两种方式。API 方式直接调用 Windows API 函数，完全从零开始创建应用程序，可从最底层了解 Windows 程序运行原理，但难度太大，不适合做桌面应用开发。

MFC 方式使用 Microsoft 构建的 MFC 框架，在基础框架上添加新增功能，相对于 API 方式，节省了大量的工作量。MFC 框架提供三种类型的程序：对话框（Dialog）、单文档（SDI）、多文档（MDI）。对话框是一个可以拖放控件的窗口，如 Windows 自带的计算器程序。单文档用于处理一种类型的文件，如记事本。多文档用于同时处理多种类型的文件，如 Word、Excel 等。

3.1.1 Win32 程序

Win32 程序利用 Windows API 函数创建桌面窗口，可以从最底层了解 Windows 程序的运行原理，掌握 Windows 的消息流程，对于学习 MFC 开发很有用处。

【实例 3-1】创建一个 Win32 程序，并在程序窗口上绘制图形。

（1）启动 VC，选择 File|New 命令，弹出 New 窗口，选择 Win32 Application 项，在 Project name 文本框里输入 API01，单击 OK 按钮，弹出 Win32 Application 窗口，如图 3-1 所示。

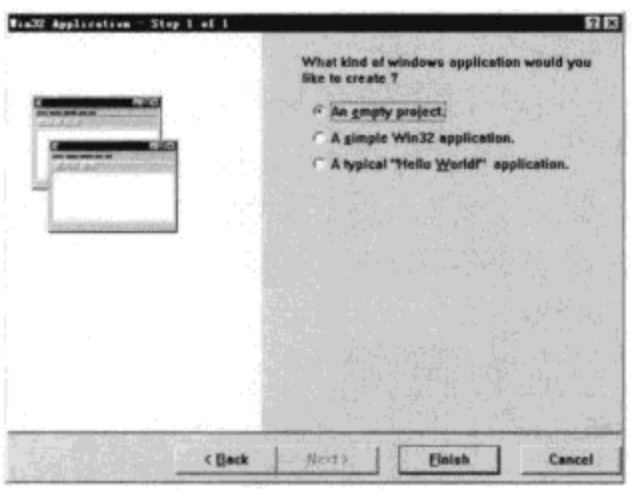


图 3-1　创建 Win32 程序

选择要创建的程序类型，有如下三种类型。

- An empty project：一个空工程，不包含任何文件。
- A simple Win32 application：一个简单的 Win32 程序，自动生成一个空的主函数。
- A typical “Hello World!” application：一个基本的 Win32 程序框架，包括窗口和菜单，运行后在窗口中显示一行文字。

（2）选择 A typical “Hello World!” application 项，单击 Finish 按钮，再单击 OK 按钮完成创建。

（3）单击工具栏上的按钮或按 F7 键，生成 EXE 可执行程序，单击按钮运行，如图 3-2 所示。

（4）在工作区窗口选择 ClassView 标签，展开 Globals 节点，如图 3-3 所示。

应用程序向导（Application Wizard）自动生成一个 Win32 程序的基本框架，在控制台程序中 main 函数是主函数，在 Windows 应用程序中 WinMain 函数是主函数，若 WinMain 函数执行完毕，程序也随之结束。

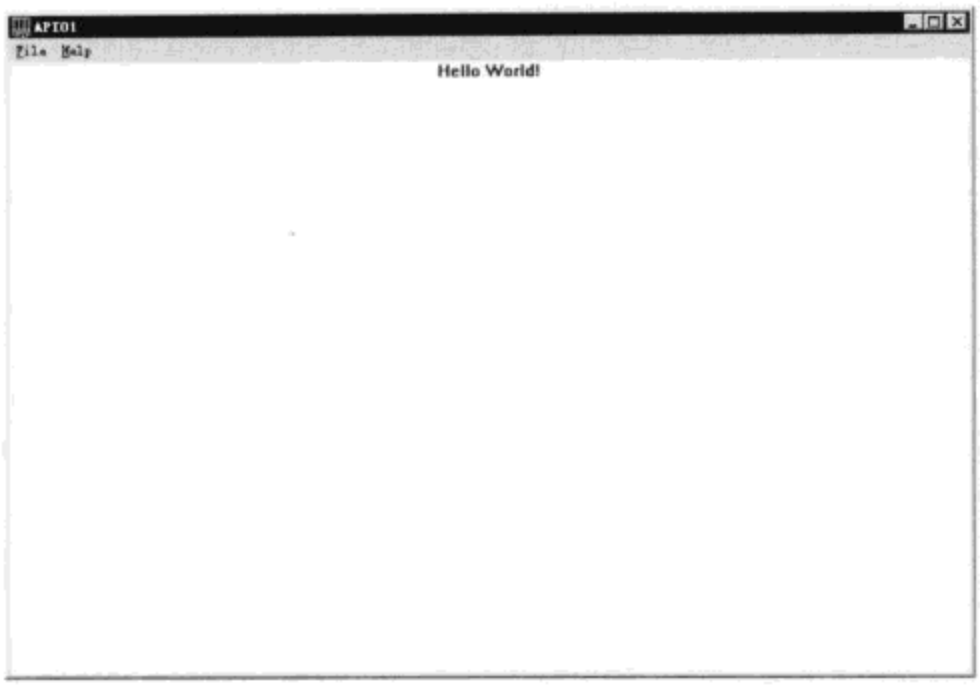


图 3-2　Win32 程序

图 3-3　类视图

WndProc 函数是窗口消息处理（process）函数，用于处理窗口收到的消息，如鼠标左键单击一下窗口，窗口收到 WM_LBUTTONDOWN 消息，按下键盘，窗口收到 WM_KEYDOWN 消息，WM（Windows Message）前缀表示窗口消息，WndProc 函数根据消息类型可执行不同的功能操作。

（5）在 ClassView 标签中双击 WndProc 项，定位到 WndProc 函数，在代码 case WM_ PAINT: 处，将 GetClientRect(hWnd, &rt); 和 EndPaint(hWnd, &ps); 之间的代码替换为如下代码：

```
    GetClientRect(hWnd, &rt);
    //开始添加代码
    DrawText(hdc, "Win32 程序", strlen("Win32 程序"), &rt, DT_CENTER);    //绘制文本
    Rectangle(hdc,20,20,300,300);                                          //绘制边框矩形
    Ellipse(hdc,20,20,300,300);                                            //绘制椭圆，圆脸
    Ellipse(hdc,80,85,120,100);                                            //椭圆，左眼
    Ellipse(hdc,200,85,240,100);                                           //椭圆，右眼
    Arc(hdc,150,140,170,160, 150,150,170,150);                             //绘制圆弧，鼻子
    Arc(hdc,120,180,200,240 , 120,220,200,220);                            //圆弧，嘴巴
    //添加结束
    EndPaint(hWnd, &ps);
```

（6）生成程序并运行，如图 3-4 所示。当窗口内容需要重新绘制时，会收到 WM_PAINT 消息，可在该消息的处理代码中添加相关的绘制函数。

若窗口 A 被窗口 B 遮挡，窗口 A 中被遮挡区域的内容会被 Windows 擦除，当窗口 B 移开后，窗口 A 中被遮挡区域的内容不会自动恢复，类似黑板上的字，被擦除后不会自动恢复到原样，只能重新写上字。同理，若要保持窗口上绘制的图形始终存在，需要将绘制函数放到 WM_PAINT 重绘消息的处理代码中，当窗口被遮挡后重新显示时，调用绘制函数重新绘制图形，从而保持图形始终存在。

图 3-4　绘制图形的窗口

DrawText、Rectangle、Ellipse、Arc 都是 Windows API 函数，分别用于绘制文本、矩形、椭圆、圆弧，当窗口被遮挡后重新显示时，调用这些函数重新绘制图形，恢复窗口原貌。

（7）双击 WinMain 项，定位到 WinMain 函数，在函数末尾找到如下代码：

```
    while (GetMessage(&msg, NULL, 0, 0))                                   //获取窗口收到的消息
    {
```

```
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
      TranslateMessage(&msg);                                    //转换消息
      DispatchMessage(&msg);                                     //分发消息
    }
  }
```

Windows 程序以消息为驱动，接收到一个消息后，窗口处理函数 WndProc 中若有对应的消息处理代码，执行该代码，否则执行默认的消息处理函数 DefWindowProc。

GetMessage 函数用于获取窗口收到的消息，存储到消息结构体变量 msg 中，在收到 WM_QUIT 退出消息之前，GetMessage 函数返回非零值，while 循环将一直持续下去，程序保持运行。若单击标题栏上的关闭按钮，程序收到 WM_QUIT 消息，GetMessage 返回 0，结束 while 循环，程序结束运行。

Tips 若要关闭当前工程，选择 File|Close Workspace 命令，弹出关闭提示窗口，单击 OK 按钮关闭工程。若要重新打开该工程，选择 File|Open Workspace 命令，弹出 Open Workspace 窗口，选择工程存放目录下的 dsw 工程文件，双击文件名或单击“打开”按钮，打开工程。

3.1.2 对话框程序

【实例 3-2】利用 MFC 应用程序向导，创建一个对话框程序。

（1）选择 File|New 命令，弹出 New 窗口，选择 MFC AppWizard(exe)项，在 Project name 文本框里输入 Dlg01，单击 OK 按钮，弹出 MFC AppWizard – Step 1 窗口，如图 3-5 所示。

（2）在程序类型中选择 Dialog based（基于对话框）项，单击 Next 按钮，弹出 MFC AppWizard – Step 2 of 4 窗口，如图 3-6 所示。

- What features：用于选择包含的要素，About Box 为“关于”对话框，用于显示版本版权信息，Context 用于添加 Help 帮助按钮，3D controls 设置控件具有 3D 效果。
- What other support：用于包含其他支持的功能，Automation 用于为程序添加自动化服务功能，ActiveX Controls 用于支持 ActiveX 控件。
- WOSA support：用于添加 Windows Sockets 网络开发功能。
- Please enter a title：用于设置对话框的标题。

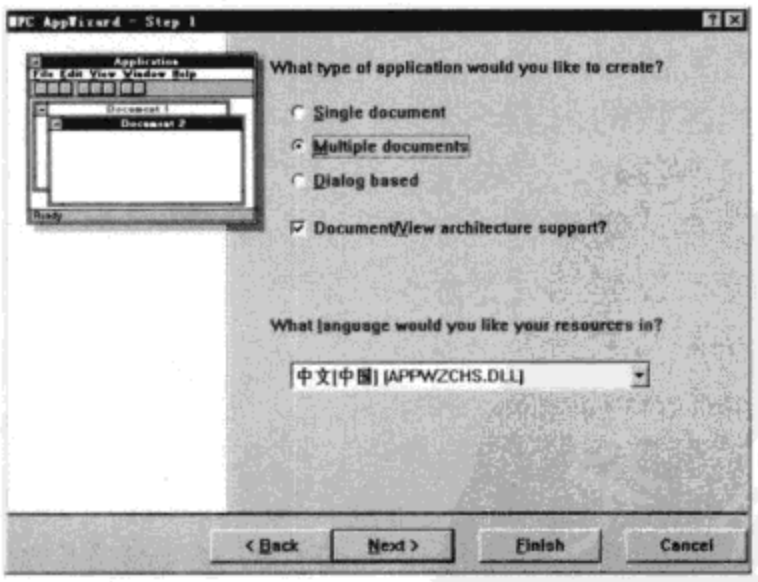


图 3-5 新建对话框程序

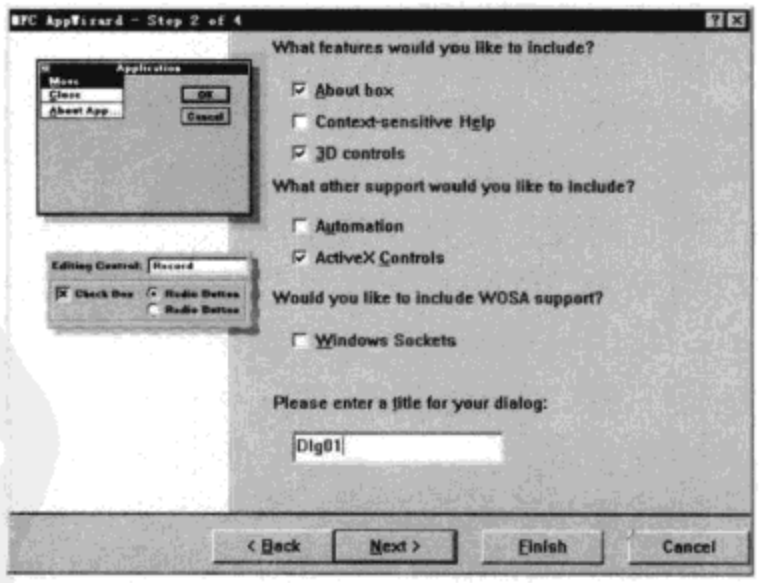


图 3-6 对话框向导步骤二

（3）保持默认设置，单击 Next 按钮，弹出 MFC AppWizard – Step 3 of 4 窗口，如图 3-7 所示。

- What style：设置程序的样式，默认为 MFC Standard。

- generate source file comments：设置是否为自动生成的代码添加注释（comment）。
- MFC library：设置 MFC 库的使用方式。

类库分为动态 DLL 和静态 LIB 两种，shared DLL 项使用 DLL 动态链接库，程序运行时需要 MFC 库的 DLL 文件支持，statically linked library 项使用 LIB 静态链接库，生成程序时将库包含到程序中，程序可独立运行。

动态 DLL 库作为一个单独的文件，随程序一块发布，程序运行时动态加载 DLL 文件，一般情况下 DLL 文件和程序放在同一个目录中，或 Windows 系统目录 system32 中。静态 LIB 库直接包含到程序中，程序可独立运行，移植性好，但体积较大。

（4）保持默认设置，单击 Next 按钮，弹出 MFC AppWizard – Step 4 of 4 窗口，如图 3-8 所示。

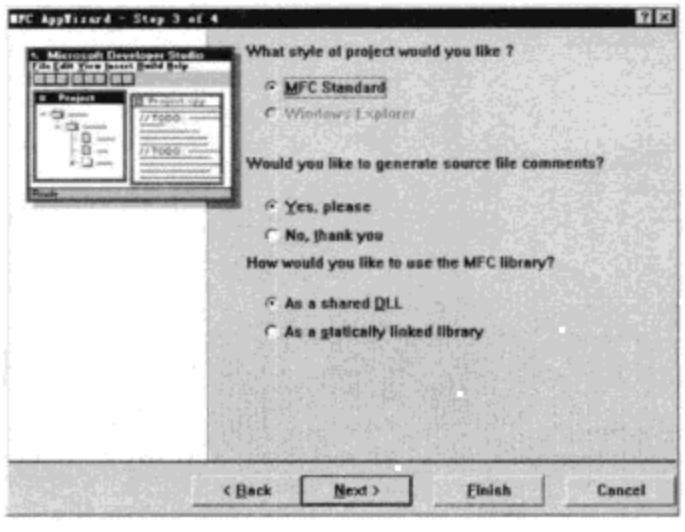


图 3-7　对话框向导步骤三

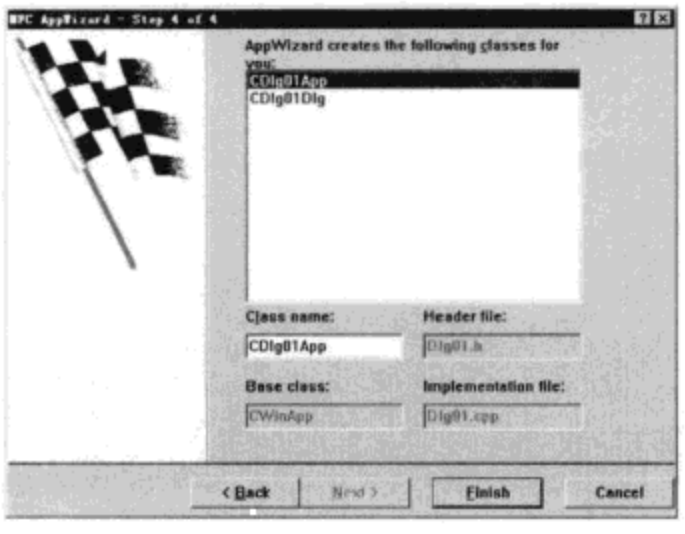


图 3-8　对话框向导步骤四

该窗口显示应用程序向导自动创建的类，Class name 文本框显示类名，Base class 文本框显示该类的基类名，Header file 文本框显示类所在的头文件（.h）名，Implementation file 文本框显示类所在的源文件（.cpp）名。

（5）保持默认设置，单击 Finish 按钮，弹出 New Project Information 窗口，显示工程的属性设置信息，单击 OK 按钮，完成工程的创建。

（6）生成程序并运行，如图 3-9 所示。

3.1.3　单文档程序

【实例 3-3】利用 MFC 应用程序向导，创建一个单文档应用程序，使用 HTML 视图显示网页文件。

（1）选择 FilelNew 命令，弹出 New 窗口，选择 MFC AppWizard(exe)项，在 Project name 文本框里输入 SDI01，单击 OK 按钮，弹出 MFC AppWizard – Step 1 窗口，如图 3-10 所示。

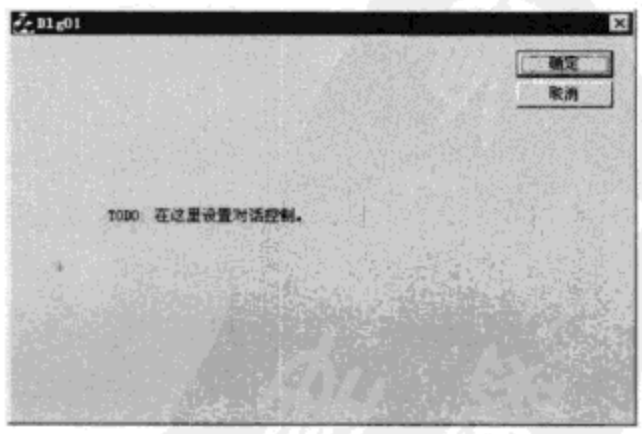


图 3-9　自动生成的对话框程序

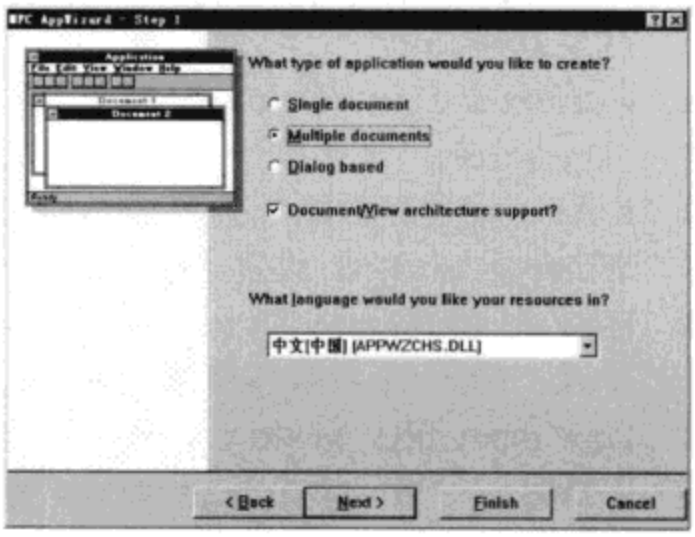


图 3-10　新建单文档程序

Document/View 表示使用文档/视图结构，该结构将数据读取与数据显示分离，MFC 提供 CDocument 类用于处理与数据有关的读写操作，CView 类用于数据的界面显示，选中该项后，向导自动生成自定义的文档和视图类。

（2）在程序类型中选择 Single document 项，选中 Document/View 复选框，单击 Next 按钮，弹出 MFC AppWizard – Step 2 of 6 窗口，如图 3-11 所示。

What database support 设置包含何种方式的数据库支持，一般选择 None。

（3）保持默认设置，单击 Next 按钮，弹出 MFC AppWizard – Step 3 of 6 窗口，如图 3-12 所示。

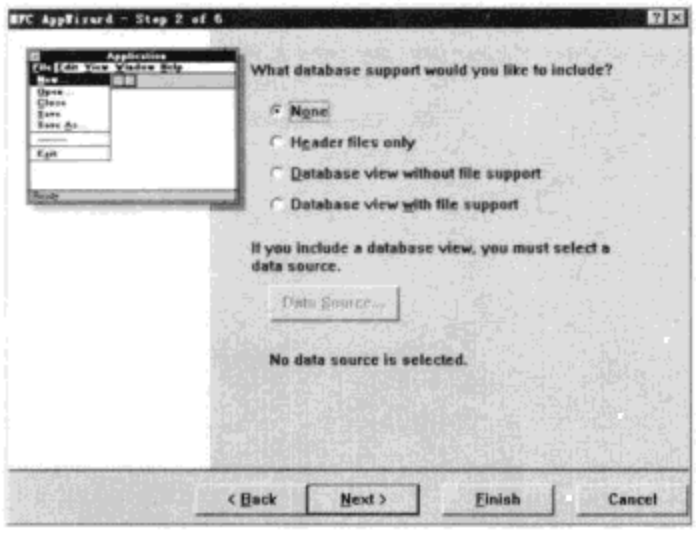


图 3-11　单文档向导步骤二

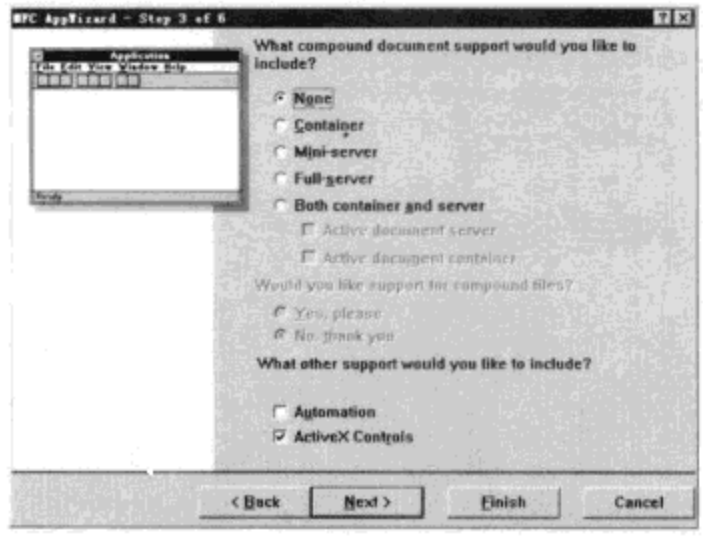


图 3-12　单文档向导步骤三

What compound document 设置包含何种类型的混合文档支持，常用于自动化服务器开发中，比如可以在 Word 中插入 Excel 表格并用 Excel 的功能操作表格数据，或者可以在浏览器中直接打开 Word 文档，不用手动打开 Word 程序。

利用 Microsoft 提供的 OLE（Object Linked and Embedded）对象链接和嵌入技术，无须手动打开软件，可直接在另外一个软件中使用其功能，如直接在 Word 中使用 Excel 的功能，调用者称为容器（Container），被调用者称为服务器（Server），若被调用者必须依赖容器才能运行称为微型服务器（Mini-server），若被调用者也能单独运行称为完全服务器（Full-server）。一个软件可以同时作为容器和服务器，如 Word 中可调用 Excel，也可被 Excel 调用。

（4）保持默认设置，单击 Next 按钮，弹出 MFC AppWizard – Step 4 of 6 窗口，如图 3-13 所示。

- Docking toolbar：设置是否添加停靠的工具栏。
- Initial status bar：设置是否添加初始化后的状态栏。
- Printing and print preview：设置是否添加打印和打印预览功能。
- Context-sensitive Help：设置是否添加帮助菜单项。
- 3D controls：设置控件是否具有 3D 效果。
- MAP：设置是否添加消息传递相关的 API 支持。
- Windows Sockets：设置是否添加 socket 网络开发支持。
- How do you want your toolbars to look：设置工具栏的样式，Normal 自然风格或 IE 风格。
- How many files would you like on your recent file：设置最近浏览文件的显示数目。

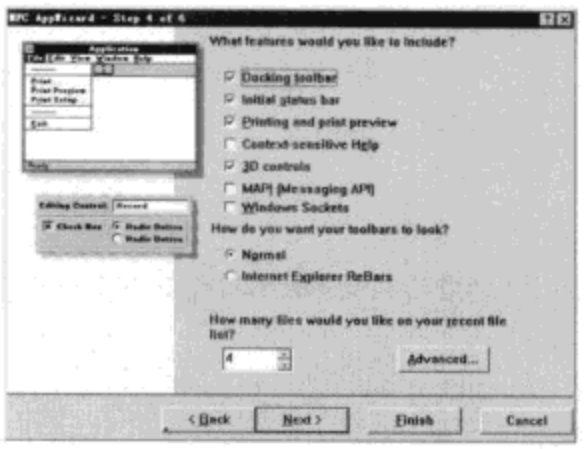


图 3-13　单文档向导步骤四

（5）保持默认设置，单击 Next 按钮，弹出 MFC AppWizard – Step 5 of 6 窗口，保持默认值，单击 Next 按钮，弹出 MFC AppWizard – Step 6 of 6 窗口，如图 3-14 所示。

自动生成的 View 视图类可以设置基类，默认为 CView 类。MFC 提供多种类型的视图以简化程

序开发，如 CEditView 可编辑文字，CFormView 可拖放控件，CHtmlView 可显示 Html 网页文件。

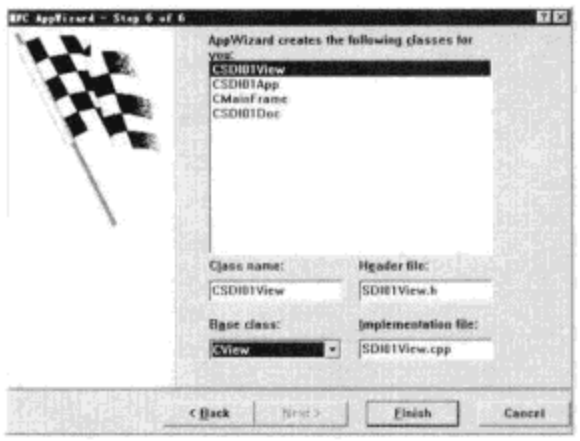


图 3-14　单文档向导步骤六

（6）在 Base class 组合框中选择 CHtmlView 项，单击 Finish 按钮，再单击 OK 按钮，完成工程的创建。

CHtmlView 可以用来显示 HTML 网页文件，HTML（Hypertext Markup Language）超文本标记语言用于制作 Web 页面，和 CSS 样式表及 JavaScript 脚本构成网站的前台，使用 CHtmlView 通过程序也可以浏览网页，关于 HTML 语言请阅读相关书籍。

（7）打开工程文件目录 SDI01，添加一张.jpg 格式图片，命名为 sdi.jpg。再新建一个文本文档，打开后添加如下 HTML 代码，保存修改后，文件重命名为 sdi.html，将图片和 HTML 文件都放在工程存放目录 SDI01 下。

```
<html>
<head><title>创建 SDI 工程</title></head>
<body>
<h2>游戏截图</h2>
<img src="sdi.jpg">
</body>
</html>
```

（8）在工作区窗口选择 ClassView 标签，双击 CSDI01View 类的 OnInitialUpdate 函数，定位到该函数，添加如下代码：

```
void CSDI01View::OnInitialUpdate()
{
    CHtmlView::OnInitialUpdate();
    CString strPage;
    ::GetCurrentDirectory(MAX_PATH,strPage.GetBuffer(MAX_PATH));  //获取当前工程目录
    strPage.ReleaseBuffer();
    Navigate2(_T(strPage+"\\sdi.html"),NULL,NULL);                //打开目录下的 HTML 文
件
}
```

OnInitialUpdate 函数用于视图的初始化显示，GetCurrentDirectory 函数获取当前工程所在目录，Navigate2 函数指定 HTML 视图显示的 HTML 文件的路径。

（9）生成程序并运行，如图 3-15 所示。

图 3-15　单文档 HTML 视图

3.1.4　多文档程序

【实例 3-4】利用 MFC 应用程序向导，创建一个多文档程序，在视图窗口绘制图形。

（1）选择 File|New 命令，弹出 New 窗口，选择 MFC

AppWizard(exe)项，在 Project name 文本框里输入 MDI01，单击 OK 按钮，弹出 MFC AppWizard – Step 1 窗口，保持默认选择 Multiple documents 项，其余步骤同单文档程序，单击 Finish 按钮完成工程的创建。

（2）在工作区窗口选择 ClassView 标签，双击 CMDI01View 类的 OnDraw 函数，定位到函数，添加如下代码：

```
void CMDI01View::OnDraw(CDC* pDC)
{
    CMDI01Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    pDC->Arc(10,10,160,160,80,10,80,160);                    //绘制圆弧，C
    pDC->MoveTo(80,80);  pDC->LineTo(180,80);
    pDC->MoveTo(200,80); pDC->LineTo(300,80);                //绘制两直线，- -
    pDC->MoveTo(130,30); pDC->LineTo(130,140);
    pDC->MoveTo(250,30); pDC->LineTo(250,140);               //绘制两直线，| |
}
```

Arc 函数用于绘制圆弧，MoveTo 函数设置直线起点，LineTo 函数根据起点和终点绘制直线。CDC 类是 MFC 提供的图形绘制类，利用该类可以设置画笔、画刷、字体，绘制点、线、多边形、文本等。OnDraw 函数用于视图类重绘操作，当窗口被遮挡后重新显示时，自动调用 OnDraw 函数完成图形的重新绘制。

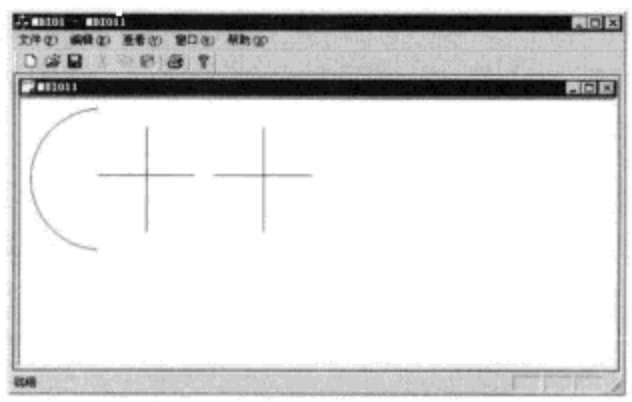


图 3-16　多文档程序

（3）生成程序并运行，如图 3-16 所示。

3.2 开发界面

Windows 以其人性化的操作界面广受欢迎，这一点在 Visual C++中也得到了充分体现，相对于 Turbo C++、DEV C++、GCC 等编译器，Visual C++功能强大、使用方便，是 Windows 下 C++软件开发的首选工具，熟悉开发界面，可以更好地发挥 Visual C++的优势。

3.2.1 菜单

菜单（menu）是 Windows 程序的基本元素，将功能相关的命令放在一个菜单下，便于用户选择使用，Visual C++的菜单项如图 3-17 所示。

（1）File 菜单用于新建、打开、保存、关闭文件等操作，包含的子菜单如图 3-18 所示。

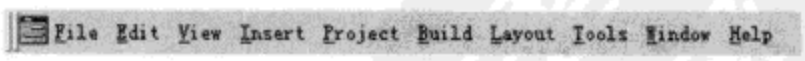


图 3-17　菜单项

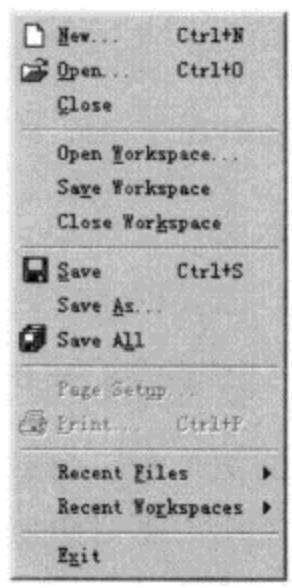


图 3-18　File 菜单

- New：新建一个工程或文件。
- Open：打开文件，如 C++源文件、资源文件等。
- Close：关闭当前窗口显示的文件。
- Open Workspace：打开一个工作空间文件（DSW）或工程文件（DSP），一个工作空间可包含多个工程。
- Save Workspace：保存工作空间。
- Close Workspace：关闭工作空间。
- Save：保存当前编辑的文件。
- Save As：另存当前编辑的文件。
- Save All：保存所有文件。
- Page Setup：页面打印设置。
- Print：打印页面。
- Recent Files：最近打开的文件。
- Recent Workspaces：最近打开的工作空间。
- Exit：退出开发环境。

（2）Exit 菜单用于界面和代码的编辑操作，包含的子菜单如图 3-19 所示。

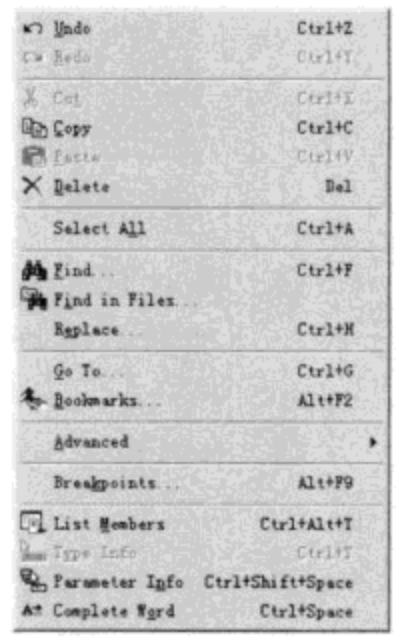


图 3-19　Edit 菜单

- Undo：撤销前一步操作，用于修正误操作，快捷键 Ctrl+Z。
- Redo：恢复前一步操作，用于恢复 Undo 撤销的操作，快捷键 Ctrl+Y。
- Cut：将选中项复制到剪贴板后并删除，快捷键 Ctrl+X。
- Copy：将选中项复制到剪贴板中，快捷键 Ctrl+C。
- Paste：将复制到剪贴板中的内容粘贴到指定位置处，快捷键 Ctrl+V。
- Delete：删除选中项，快捷键 Delete。
- Select All：选中当前所有项，快捷键 Ctrl+A。
- Find：在当前窗口中查找某个字符串，快捷键 Ctrl+F。
- Find in Files：在整个工程中查找某个字符串。
- Replace：用指定字符串替代目标字符串，可在整个文件或选中项范围内替换，快捷键 Ctrl+H。

（3）View 菜单用于子窗口的显示与隐藏，包含的子菜单如图 3-20 所示。

- ClassWizard：打开类向导窗口，便于对类进行操作，快捷键 Ctrl+W。
- Resouce Symbols：打开资源符号窗口，查看已有的资源 ID，可新建、更改资源 ID。
- Full Screen：当前编辑窗口切换到全屏模式。
- Workspace：显示工作区窗口。
- Output：显示输出窗口。
- Debug Windows：程序调试时要显示的监视窗口。
- Properties：打开当前选中项的属性窗口，快捷键 Alt+Enter。

（4）Insert 菜单用于在当前工程中插入新项，包含的子菜单如图 3-21 所示。

- New Class：添加一个新类。
- New Form：添加一个新的窗体。
- Resource：添加资源。

（5）Project 菜单用于设置工程属性、添加已有文件和控件等，包含的子菜单如图 3-22 所示。

- Set Active Project：设置当前活动工程，用于一个工作空间包含多个工程时。

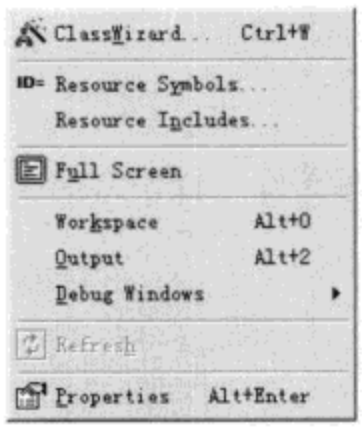


图 3-20 View 菜单

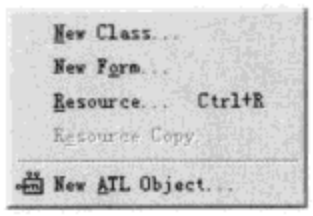


图 3-21 Insert 菜单

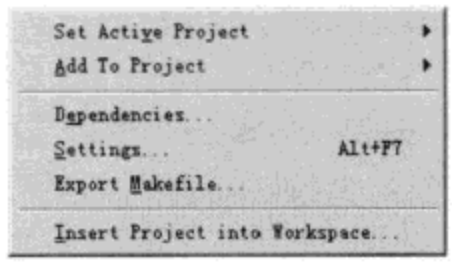


图 3-22 Project 菜单

- Add To Project：向当前工程添加文件或控件，其子菜单如图 3-23 所示。New 菜单添加新建项，Files 菜单添加已有项，Components and Controls 菜单添加已注册的 ActiveX 控件。
- Settings：设置工程属性，弹出的工程设置窗口如图 3-24 所示。

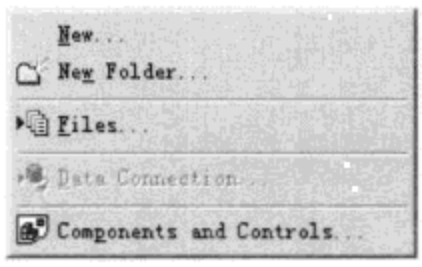


图 3-23 Add To Project 子菜单

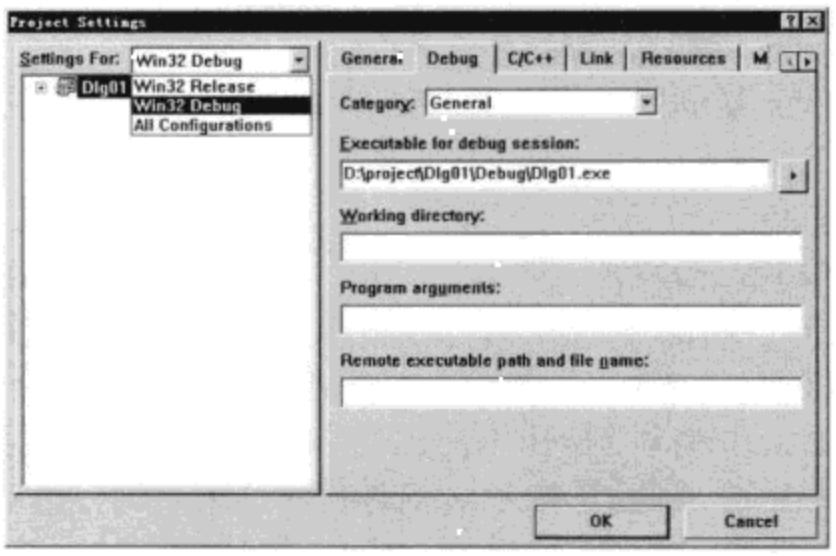


图 3-24 工程设置窗口

EXE 可执行程序分为 Debug（调试版）和 Release（发布版）两种版本，在程序开发尚未结束时使用 Debug 调试版本，可以设置断点调试、查找错误，该版本中包含有大量的测试信息。当程序测试完毕、正确无误后生成 Release 版本，该版本不能调试，移除了测试代码，相对于 Debug 版本运行速度更快。

（6）Build 菜单用于工程的编译、生成、调试等，包含的子菜单如图 3-25 所示。

- Compile：编译当前文件。
- Build：生成 EXE 程序。
- Rebuild All：所有文件重新生成。
- Batch Build：批量生成程序。
- Clean：清除生成目录下的文件。
- Start Debug：调试程序，包括断点调试、单步调试、进入函数内部等。
- Execute：执行 EXE 程序。

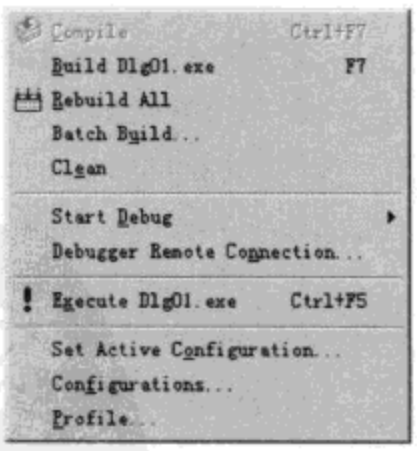


图 3-25 Build 菜单

（7）Tools 菜单提供一些辅助工具，选项设置等，包含的子菜单如图 3-26 所示。

- Source Browser：查看工程资源，需要生成 BSC 浏览文件。
- Register Control：注册 DLL 工程生成的 ActiveX 控件。
- Error Lookup：查看函数错误代码所代表的意义。

- ActiveX Control Test Container：控件测试容器。
- OLE/COM Object Viewer：查看 OLE/COM 对象。
- Options：选项的设置，弹出的设置窗口如图 3-27 所示。

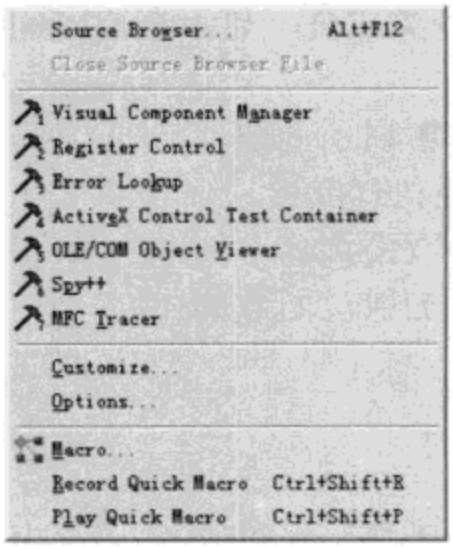


图 3-26　Tools 菜单

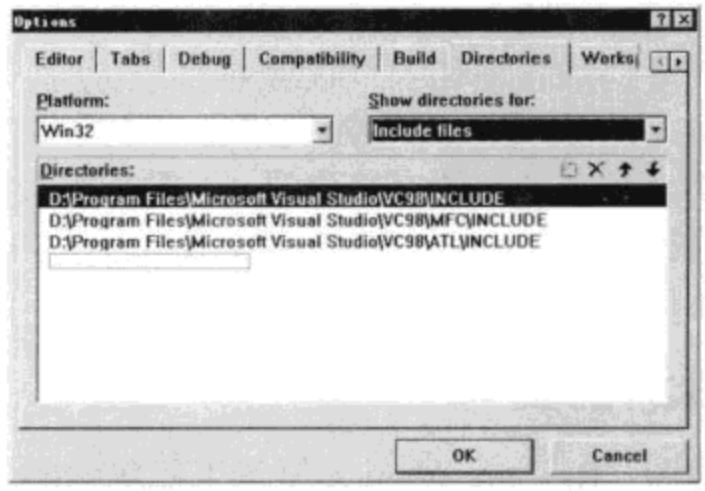


图 3-27　Options 窗口

设置窗口有多个选项卡，其中 Directories 选项卡用于设置工程相关文件所在的路径，Platform 组合框选择工程所在的平台，Show directories for 组合框选择目录类型，Visual C++根据设置关联的目录查找需要的文件，目录类型有如下几种。

- Executable files exe：程序的关联目录。
- Include files：头文件（.h）的关联目录。
- Library files：库文件（.lib）的关联目录。
- Source files：源文件（.cpp）的关联目录。

3.2.2　工具条

工具条常作为菜单项的快捷方式，在菜单中一般有其功能的对应项，Visual C++提供工具栏便于操作，在窗口非工作区右键单击鼠标，在弹出的快捷菜单中可选择要显示的工具栏。

Standard 工具栏提供基本操作，如图 3-28 所示。该工具栏功能依次是新建文本、打开文件、保存当前文件、保存所有文件，剪切、复制、粘贴，撤销、重做，工作区窗口、输出窗口、窗口列表，文件查找窗口。

图 3-28　Standard 工具栏

WizardBar 工具栏显示类和类成员列表，如图 3-29 所示。

图 3-29　WizardBar 工具栏

Build MiniBar 工具栏提供编译、生成程序等功能，如图 3-30 所示。该工具栏功能依次是编译、生成可执行程序、停止生成、执行程序、断点调试、添加断点。Build 工具栏除了具备 Build MiniBar 的所有功能，还可选择 Debug 和 Release 模式，如图 3-31 所示。

图 3-30　Build MiniBar 工具栏

图 3-31　Build 工具栏

3.2.3 类视图

类视图（ClassView）以类结构的方式显示整个工程，根节点为工程名称，如 Dlg01，根节点下的一级子节点为类名，如 CAboutDlg，若函数或变量不是类成员，显示在 Globals 节点下，表示全局成员，如图 3-32 所示。

不同类型的成员使用不同的图标加以区分，如红色菱形 ◆ 表示函数，绿色菱形 ◆ 表示变量，钥匙 表示类保护成员，锁 表示类私有成员。

双击类名可打开类的定义位置，一般为类所在的头文件，双击类名下的成员名称可打开函数或变量的定义位置。右键单击类名，可添加成员函数和变量，以及虚函数、消息处理函数等。

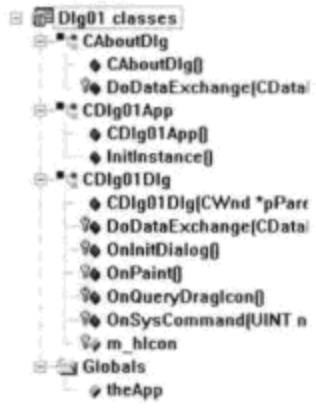


图 3-32 类视图

3.2.4 资源视图

资源视图（ResourceView）显示工程包含的各种资源，程序代码仅负责逻辑功能，程序的外观显示需要使用资源文件，包括 Accelerator 快捷键、Bitmap 位图、Cursor 鼠标光标、Dialog 对话框模板、Icon 图标、Menu 菜单、String Table 字符串表、Toolbar 工具栏、Version 版本信息，如图 3-33 所示。

可利用 Visual C++提供的编辑界面制作资源，也可直接从外部文件导入已有资源，生成程序时将资源文件包含到程序中，资源文件是程序组成的一部分。

3.2.5 文件视图

文件视图（FileView）显示工程包含的所有文件，Source Files 节点下显示所有的源文件，Header Files 节点下显示所有的头文件，Resouce Files 节点下显示所有的资源文件，ReadMe.txt 存放说明信息，如图 3-34 所示。

若要删除某个文件，选中文件名按 Delete 键，先从工程中移除该文件，再从硬盘上物理删除。若直接从硬盘上删除，会导致工程找不到指定文件。

图 3-33 资源视图

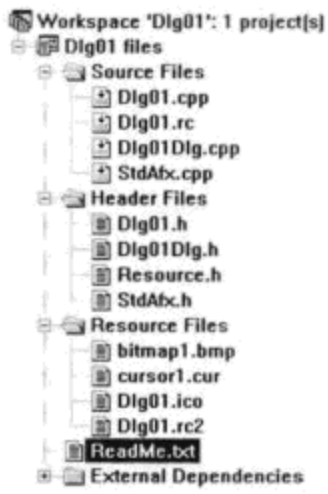


图 3-34 文件视图

3.2.6 类向导

类向导提供一个窗口集中操作类成员，可以方便地添加虚函数、消息处理函数、类成员变量等。选择 View|ClassWizard 命令，或按 Ctrl+W 键打开 MFC ClassWizard 窗口，如图 3-35 所示。

Message Maps（消息映射）选项卡用于为类添加虚函数和消息处理函数，Project 组合框选择工程名称，Class name 组合框选择类名，Object IDs 列表框选择要添加函数的类或控件，Messages 列表框选择要添加的虚函数或消息，以 WM_开头的为消息，其余为要重写的虚函数，Member

functions 列表框为已添加的成员函数，其中图标 V 代表虚函数，图标 W 代表消息处理函数。

在 Messages 列表框选择一项后，单击 Add Function 按钮，添加新的成员函数，单击 Edit Code 按钮直接进入代码编辑窗口，单击 OK 按钮保存并退出。

Member Variables（成员变量）选项卡用于为对话框类的控件添加变量，如图 3-36 所示。Control IDs 列表框显示控件 ID 和其对应的变量，每种控件都有其对应的控件类，对控件添加映射的变量后，可通过映射变量操作控件。

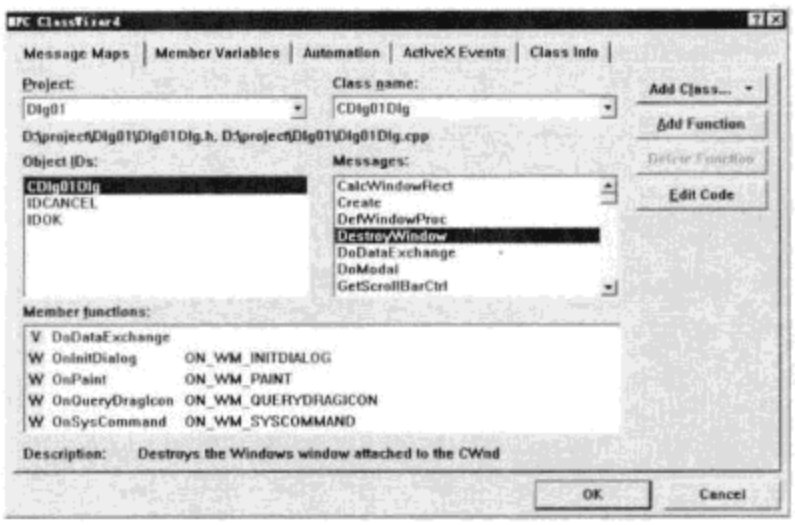


图 3-35　类向导窗口 Message Maps 选项卡

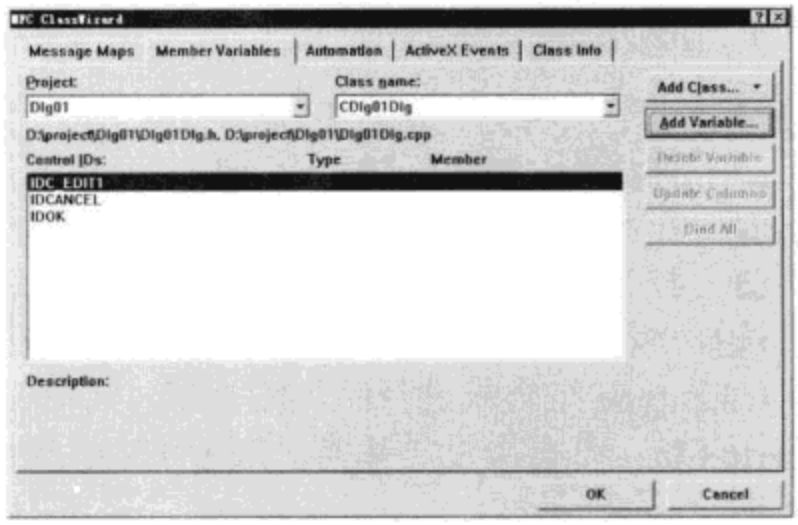


图 3-36　类向导 Member Variables 选项卡

选择要添加映射变量的控件 ID，双击项或单击 Add Variable 按钮，弹出 Add Member Variable 窗口，如图 3-37 所示。一个控件可以添加多个不同类型的变量，单击 Delete Variable 按钮删除已添加的变量。

Member variable name 文本框用于输入变量名称，一般以 m_开头表示成员变量。Category 组合框选择类别，分为 Value 值类型和 Control 控件类型，一些控件可以映射为一个简单类型的变量，如文本框可以添加一个 CString 类型变量，变量值为文本框中的字符串，也可添加一个 int 类型变量，变量值为文本框中的数字。

若在 Category 组合框选择 Control 项，Variable type 组合框显示控件对应的类。单击 OK 按钮完成添加并关闭窗口。添加完成员变量后，单击 MFC ClassWizard 窗口的 OK 按钮，保存添加并退出。

3.2.7　输出窗口

输出窗口主要用于显示生成结果，如图 3-38 所示。error 显示错误数目，有错误不能生成程序，warning 显示警告数目，表示部分代码不规范，一般不影响程序的生成。双击 error 项可定位到错误位置，若有多个错误，应从第一个错误开始检查。

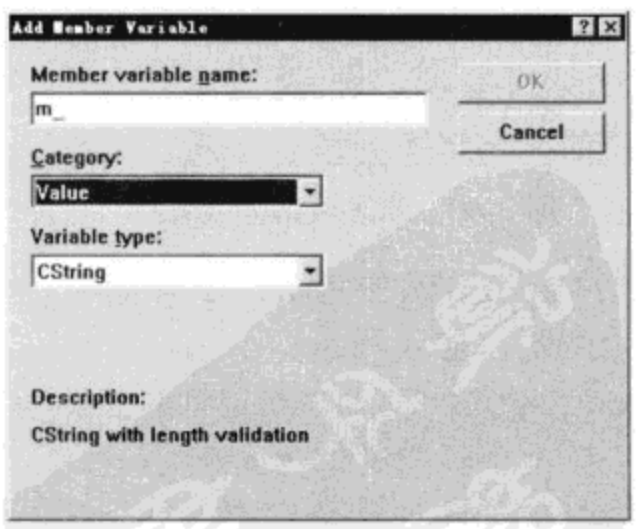


图 3-37　添加变量窗口

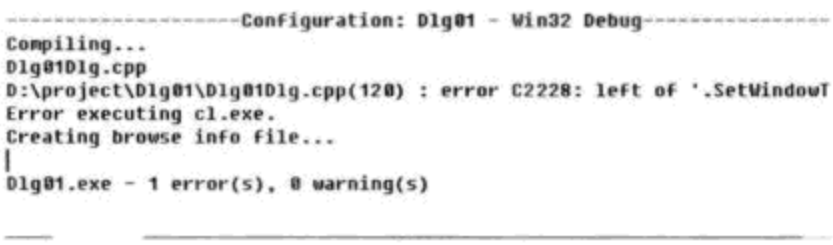


图 3-38　输出窗口

3.3 使用技巧

在实际开发中，经常需要进行创建一个自定义类或派生类、为类添加成员、添加资源文件等操作，若通过手工添加代码的方式完成，工作量大且容易出错，Visual C++提供一些窗口用以自动完成这些功能，掌握开发环境的使用技巧，可以节省工作量，提高开发效率。

3.3.1 添加类

通过以下三种方式可打开添加类窗口：

- 选择 Insert|New Class 命令。
- 在类视图中右键单击工程名，在弹出的快捷菜单中选择 New Class 命令。
- 打开类向导窗口，在 Message Maps 选项卡中单击 Add Class 按钮并选择 New 命令。

选择一种方式，打开 New Class 窗口，如图 3-39 所示。

Class type 组合框选择添加的类型，其中 MFC Class 从已有 MFC 类派生，扩展基类的功能，Generic Class 可自由设定基类，Form Class 从窗口类派生。Name 文本框设置类名，一般以大写 C 开头。

Base class 组合框选择基类。若在 Class type 组合框中选择 Form Class 项，Dialog ID 组合框选择对话框模板 ID。单击 OK 按钮完成创建并退出。

3.3.2 添加类成员函数

在类视图右键单击类名，在弹出的快捷菜单中选择 Add Member Function 命令，弹出 Add Member Function 窗口，如图 3-40 所示。Function Type 文本框用于输入函数返回类型，Function Declaration 文本框用于输入函数名和参数列表，Access 单选按钮用于选择函数访问权限，Static 复选框用于设置是否为静态成员函数，Virtual 复选框用于设置是否为虚函数，单击 OK 按钮完成添加并退出。

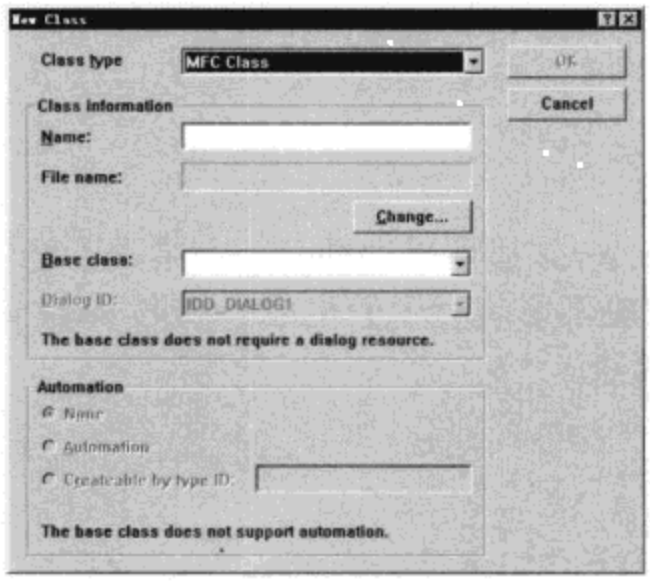


图 3-39 添加类窗口

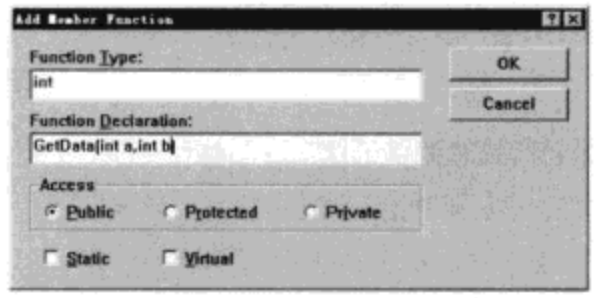


图 3-40 添加类成员函数

若手动添加类成员函数，需要在类的头文件(.h)和源文件(.cpp)里添加两次，步骤如下：

（1）双击类名可打开类的头文件，在类定义里添加函数声明，如下所示：

```
public:
  int GetData(int a,int b);
```

（2）双击类成员名可打开类的实现文件，在函数体外的空白处添加函数实现，如下所示：

```
int CDlg01Dlg::GetData(int a,int b)
```

```
{
  return a+b;
}
```

3.3.3　添加类成员变量

在类视图右键单击类名，在弹出的快捷菜单中选择 Add Member Variable 命令，弹出 Add Member Variable 窗口，如图 3-41 所示。Variable Type 文本框用于输入变量类型，Variable Name 文本框用于输入变量名称，一般以 m_开头表示类成员，Access 单选按钮用于选择变量访问权限，单击 OK 按钮完成添加并退出。

若手动添加成员变量，双击类名打开类的头文件，在类定义里添加成员变量。若要添加多个成员变量，手动添加类成员变量更快捷。

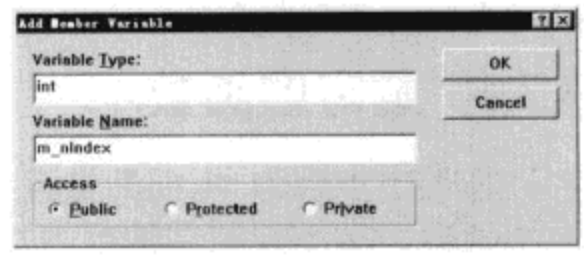


图 3-41　添加类成员变量

3.3.4　添加消息处理函数

从 MFC 类 CCmdTarget 派生的类都可接收消息，消息由操作系统发送给程序，程序可添加相应的消息处理函数用来响应消息。在 Windows 中用结构体 MSG 包装消息，格式如下：

```
typedef struct {
 HWND hwnd; UINT message; WPARAM wParam; LPARAM lParam; DWORD time; POINT pt;
 } MSG, *PMSG;
```

参数如下。

- hwnd：接收消息的窗口的句柄。
- message：消息类型，如左键按下为 WM_LBUTTONDOWN，鼠标移动为 WM_MOUSEMOVE。
- wParam：消息的附加参数，根据 message 类型决定，如键盘消息的按键值。
- lParam：消息的附加参数，根据 message 类型决定。
- time：消息发送时间。
- pt：消息发送时鼠标的位置。

句柄（handle）用来唯一标识一个 Windows 资源，窗口（HWND）、程序实例（HINSTANCE）、设备环境（HDC）、画刷（HBRUSH）、画笔（HPEN）、光标（HCURSOR）等都有句柄值，如根据窗口句柄值区分不同窗口，可将句柄作为资源的 ID 值，在 Windows 中句柄的部分定义如下：

```
typedef void* HANDLE;
typedef HANDLE HWND;
typedef HANDLE HINSTANCE;
```

句柄实际上就是一个 void 类型指针值，指向内存中资源文件的地址，Windows 将其抽象为一种数据类型，在使用时可将其当做一种可定位 Windows 资源的新数据类型。

UINT、WPARAM、LPARAM、DWORD 是 Windows 中重定义的数据类型，定义如下：

```
typedef unsigned int UINT;
typedef UINT WPARAM;
typedef LONG LPARAM;
typedef unsigned long DWORD;
```

使用 typedef 可重定义数据类型，便于根据类型了解变量的作用，或提高代码的可移植性，如 WPARAM 类型变量代表消息参数，HWND 代表窗口句柄，TCHAR 在不同环境中可替换为 char 或 wchar_t。

MFC 框架为 Windows 消息提供了默认的消息处理函数，也可添加自定义的消息处理函数，

在类视图右键单击类名，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，弹出 New Windows Message 窗口，如图 3-42 所示。

New Windows message/events 列表框用于选择消息类型，Existing message/event handlers 列表框为已添加的消息，Class or object to handle 列表用于选择添加消息处理函数的类或控件，单击 Add Handler 按钮可添加消息处理函数，单击 Add and Edit 按钮可添加消息函数并编辑代码，单击 Edit Existing 按钮可编辑已添加的消息处理函数，单击 OK 按钮完成添加并退出。

在类向导窗口中也可添加消息处理函数，按下Ctrl+W组合键打开类向导窗口，在Message Maps选项卡中，选择要添加的类或控件 ID、消息类型，单击 Add Function 按钮添加消息处理函数。

3.3.5 重写虚函数

基类中声明为 virtual 的函数可在派生类中重写，根据指针实际指向的对象确定调用哪个版本的函数。在类视图右键单击类名，在弹出的快捷菜单中选择 Add Virtual Function 命令，弹出 New Virtual Override 窗口，如图 3-43 所示。

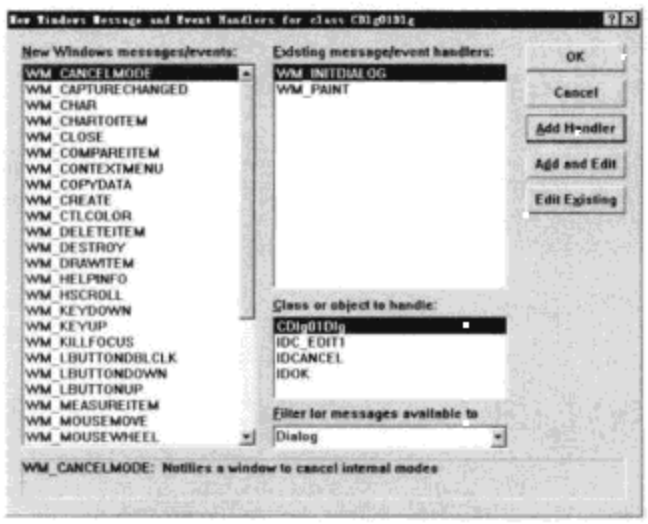


图 3-42 添加消息处理函数

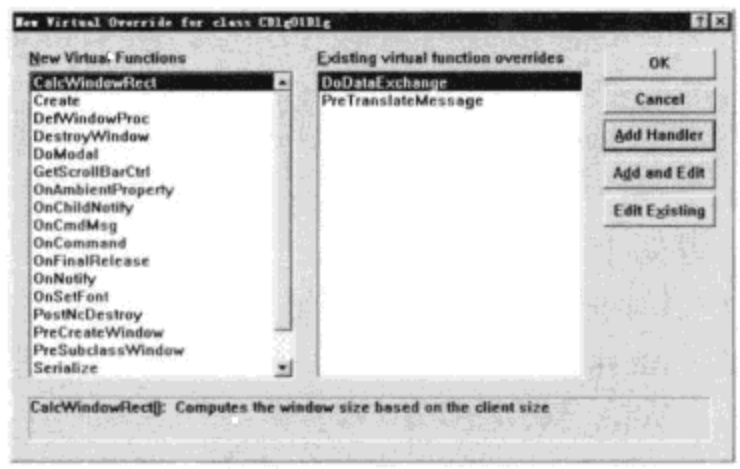


图 3-43 添加虚函数

New Virtual Functions 列表框中可选择要重写的虚函数，Existing virtual function overrides 列表框中显示已重写的虚函数，单击 Add Handler 按钮可添加虚函数，单击 Add and Edit 按钮可添加并编辑，单击 OK 按钮保存添加并退出。

3.3.6 添加资源

在资源视图右键单击一项，在弹出的快捷菜单中选择 Insert 命令，弹出 Insert Resource 窗口，如图 3-44 所示。Resource type 列表框用于选择资源类型，单击 New 按钮可添加新项，单击 Import 按钮可导入已有资源文件。

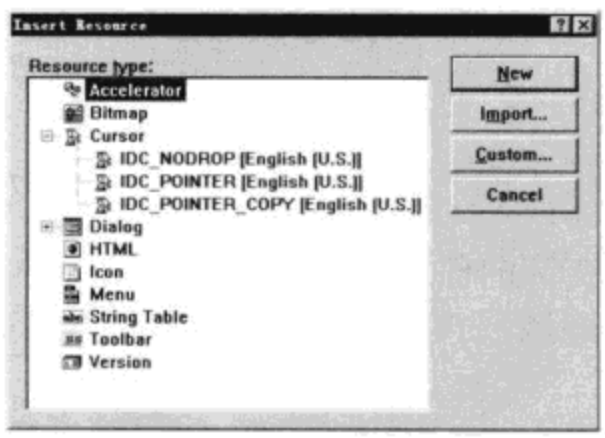


图 3-44 添加资源

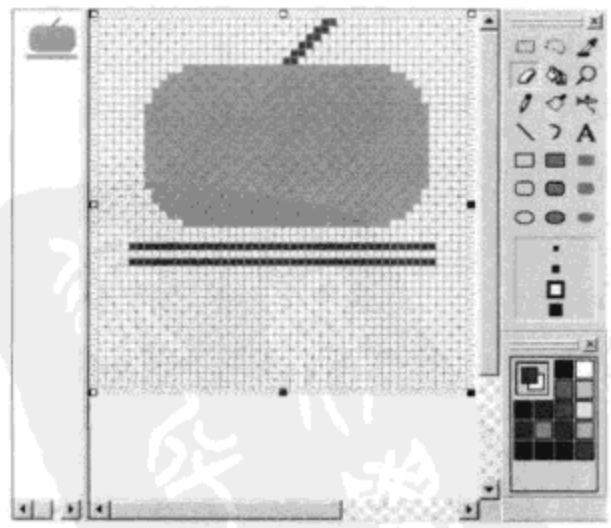


图 3-45 位图编辑界面

在资源视图双击 Bitmap 节点下的项，打开位图编辑界面，如图 3-45 所示。左边为效果图，中间为图形编辑窗口，右边为工具箱和调色板。工具箱中可以选择图形工具和图形尺寸，调色板中鼠标左击颜色设置前景色，鼠标右键单击颜色设置背景色。

3.3.7　添加已有文件和控件

若添加已有文件到工程，可先将文件复制到工程目录下，选择 Project|Add To Project|Files 命令，弹出 Insert Files into Project 窗口，如图 3-46 所示。选择要添加的一个或多个文件，单击 OK 按钮完成添加并退出。

若添加已注册的 ActiveX 控件到控件箱中，选择 Project|Add To Project|Components and Controls 命令，弹出 Components and Controls Gallery 窗口，可添加已注册的控件和 Visual C++ 自带的控件，双击 Registered ActiveX Controls 文件夹，显示所有已注册的控件，如图 3-47 所示。

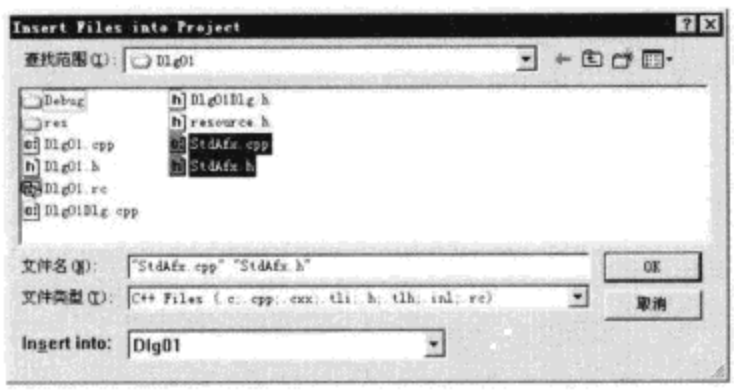


图 3-46　添加已有文件

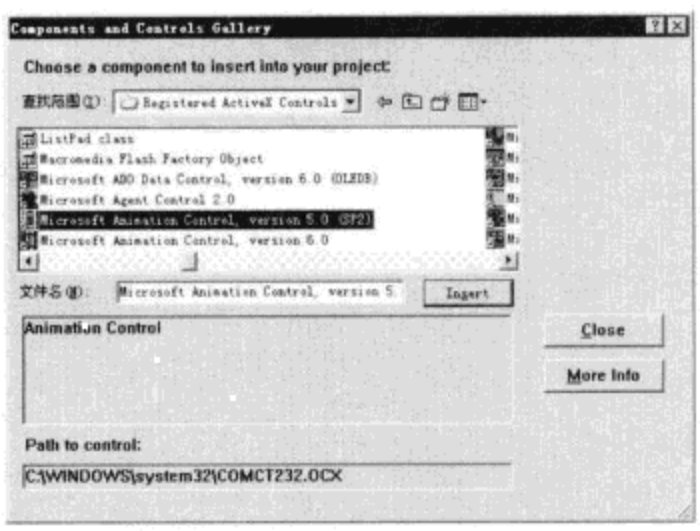


图 3-47　添加控件

选择要添加的控件，单击 Insert 按钮，再单击两次 OK 按钮完成添加。单击 More Info 按钮可查看控件的帮助信息。添加成功后，在控件工具箱中出现添加控件的图标，可将控件拖放到对话框模板上。

3.3.8　设置代码字体样式

Visual C++的代码编辑窗口默认使用 Fixedsys 字体，大小为 12，若不习惯该字体，可改变代码的字体样式。选择 Tools|Options 命令，弹出 Options 窗口，切换到 Format 选项卡，如图 3-48 所示。

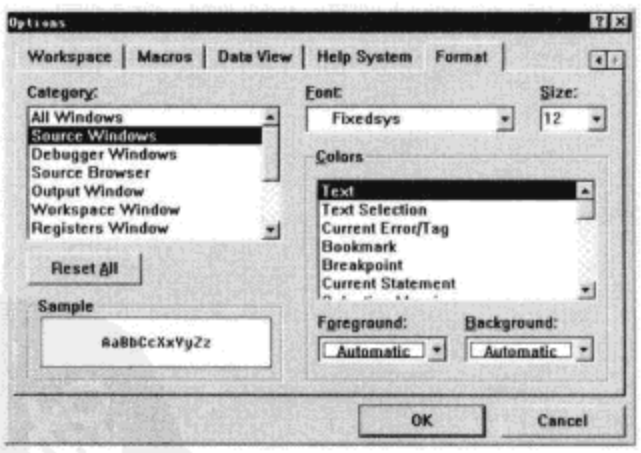


图 3-48　设置代码字体

Category 列表框用于选择设置样式的窗口，如 Source Windows 为代码编辑窗口，Font 组合框选择字体，Size 组合框选择字体大小，Colors 分组框用于设置不同类别代码的前景色和背景色。单击 OK 按钮保存修改并退出。

3.4 小结

从本章开始，读者开始接触 Visual C++开发环境。首先介绍了如何创建运行程序，读者在此不必着急，以后创建的工程多了，对于创建过程会慢慢熟悉的，随后详细介绍了开发界面，最后介绍了开发环境的使用技巧。不必死记硬背开发环境的操作方法，这是一个熟能生巧的过程。希望通过本章的学习，读者能创建各种类型的运行程序。

3.5 习题

1. 简要说出 Visual C++的各个系统菜单的主要功能。
2. 练习创建一个 Win32 程序。
3. 练习创建一个对话框程序。
4. 练习创建一个单文档程序。
5. 练习创建一个多文档程序。

第2篇　可视化编程

第4章　常用控件

在微软公司推出 Windows 操作系统之前，绝大多数计算机使用基于字符界面的系统，如著名的 MS-DOS 系统，它只有简单的输入/输出，且用户操作极其不便。Windows 操作系统面世后，其人性化的操作界面、丰富的鼠标键盘交互操作，赢得了用户的广泛好评，从而牢牢地占领桌面操作系统的市场。

对话框是一类常用的桌面软件，如计算器、扫雷等，典型的对话框由标题栏、系统按钮、窗体及窗体上的控件组成，用户通过鼠标、键盘交互操作可以实现特定的功能。

可以在多种环境下开发对话框软件，如 Visual Basic、Delphi、.NET、Visual C++等。使用 C++作为开发语言，就要掌握 Visual C++开发环境及其控件，在熟悉了一种环境下的开发后，就可以触类旁通，快速转入其他开发环境。

4.1 了解生成类

学习 Visual C++的第一道坎，就是面对 MFC Wizard 自动生成的代码时的茫然无措，骤然从熟悉的 C++语法到复杂陌生的代码框架，数不清的宏定义、数据类型、MFC 类、消息函数扑面而来，过眼即忘，不禁怀疑自己能否掌握 Visual C++。

Visual C++之所以强大，在于它对 Windows API 函数的浅层封装，如将绘图相关的 API 函数封装到 CDC 类中，将窗口相关的 API 函数封装到 CWnd 类中，开发人员可以在这个浅层框架基础上自由发挥想象与能力，最大限度地利用系统资源。但凡事有利也有弊，正是由于高度的灵活，造成学习曲线陡增，初学者犹如攀登近乎垂直的山壁，信心大减。

Visual C++之难在于此，除了必要的兴趣和坚持，学习方法也是很重要的因素。一般来说，多动手写代码、调试程序、多做学习笔记、动脑思考是有效的学习方式。本节简要介绍新建一个对话框工程所自动生成的类，便于读者对框架有个大致的了解。

【实例 4-1】创建一个基于对话框的工程，名为 Dlg041。

（1）选择 File|New 命令，打开 New 窗口，选择 Projects 标签，在左侧工程类型中选择 MFC AppWizard(exe)项，在右侧 Project name 文本框中输入工程名称 Dlg041，在 Location 文本框中选择工程文件所在的路径，如图 4-1 所示。

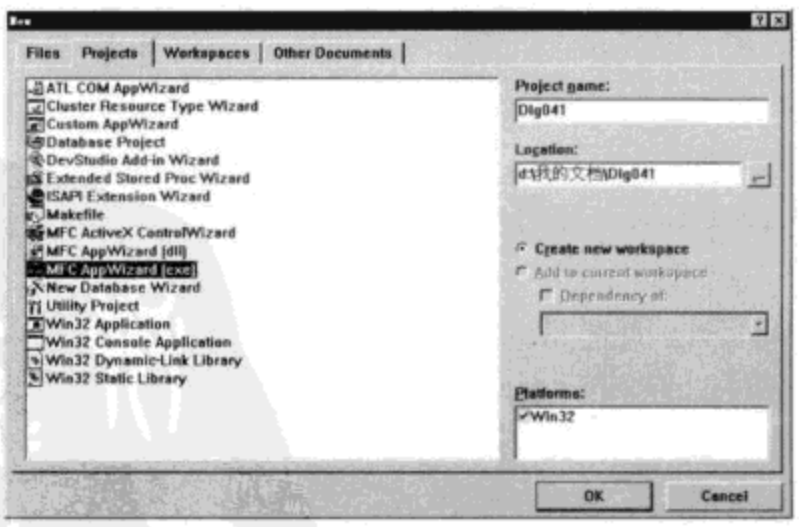


图 4-1　新建工程窗口

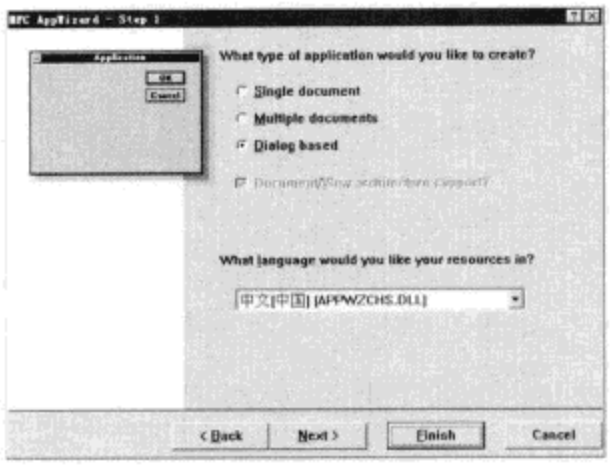


图 4-2　MFC 应用程序向导窗口

（2）单击 OK 按钮，弹出 MFC AppWizard – Step 1 窗口，在应用程序类型中选择 Dialog based 单选按钮，如图 4-2 所示。

（3）单击 Finish 按钮，弹出 New Project Information 窗口，单击 OK 按钮，完成对话框工程的创建。

（4）在工作区窗口选择 ClassView 标签，展开所有节点，如图 4-3 所示。

对话框工程的应用程序向导自动生成三个类：CAboutDlg 类、CDlg041App 类、CDlg041Dlg 类，以及一个全局对象 theApp。

（5）切换到 ResourceView 标签，展开所有节点，如图 4-4 所示。

图 4-3　对话框工程类视图

图 4-4　对话框工程资源视图

应用程序向导自动生成两个对话框模板，其中 IDD_ABOUTBOX 对应 CAboutDlg 类，一般用于显示版本和版权信息，如图 4-5 所示。IDD_DLG041_DIALOG 对应 CDlg041Dlg 类，是对话框程序的主窗口，如图 4-6 所示。

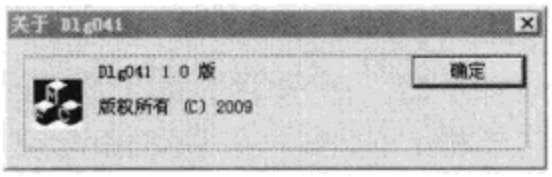


图 4-5　关于对话框模板

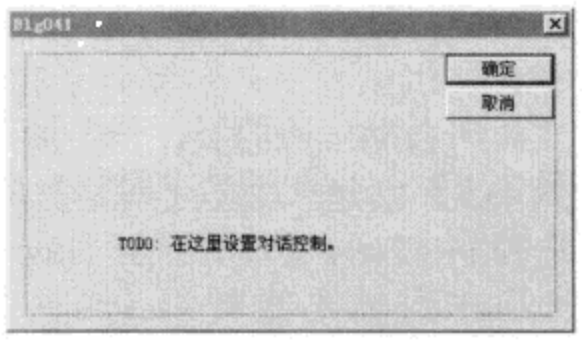


图 4-6　对话框主界面模板

资源视图中的对话框模板、图标、位图、加速键、字符串表等都属于资源文件，负责桌面软件的外观显示，是程序具有可视化界面的基础。程序代码负责软件的逻辑流程，它们相互配合完成桌面软件的显示和功能操作。

根据对话框模板 IDD_DLG041_DIALOG，创建对应的对话框类 CDlg041Dlg，当窗口打开和关闭时，内部流程如下：

（1）先创建一个 CDlg041Dlg 类对象，类对象根据对话框模板生成一个对话框实例。对话框实例作为图形资源也占用内存，在窗口显示期间，类对象和对话框实例同时存在。

（2）关闭窗口时，类对象销毁其对话框实例。

（3）当类对象超出其作用域后，自动被释放。

CDlg041Dlg 类负责对话框窗口的创建、初始化、显示、销毁，而对话框窗口的界面布局由对话框模板资源 IDD_DLG041_DIALOG 决定。同理，类 CAboutDlg 负责操作对话框模板

IDD_ABOUTBOX 生成的对话框窗口。

CDlg041App 类负责整个程序的启动和退出，一个程序可以有多个窗口，但只能有一个主窗口和程序实例。全局类对象 theApp 被创建时，自动调用 CDlg041App 类的构造函数，程序框架开始运行。

4.2 静态文本

对话框作为一个容器（container），可以拖放（drag）各种类型的控件到对话框容器中，熟练使用各种控件是制作对话框软件的基础技能。静态文本（Static Text）控件常作为文本提示信息，表明其他控件的作用。

4.2.1 设置属性

【实例 4-2】在 Dlg041 工程中添加静态文本控件，并使用代码设置控件显示内容。

（1）启动 Visual C++，选择 File|Open Workspace 命令，打开 Open Workspace 对话框，选择文件 Dlg041.dsw，双击文件名或单击“打开”按钮，打开工程 Dlg041。

（2）在工作区窗口切换到 ResourceView 标签，展开 Dialog 节点，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口，如图 4-6 所示。

（3）分别单击对话框窗口上已有的三个控件，按下 Delete 键，删除自动添加的控件，用鼠标拖动窗口边界调整窗口的大小。

（4）在控件工具箱窗口单击静态文本控件 Aa，拖放到对话框窗口上，如图 4-7 所示。

（5）选择静态文本控件，单击鼠标右键，在弹出的快捷菜单中选择 Properties 命令，打开 Text Properties 窗口，如图 4-8 所示。

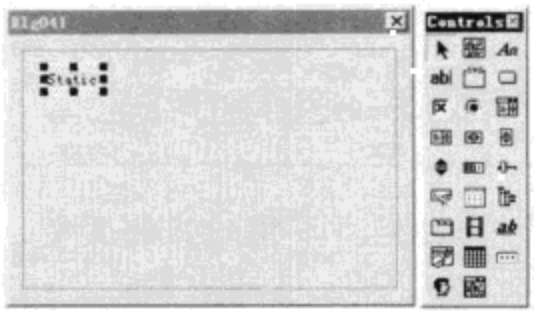


图 4-7　添加静态文本控件

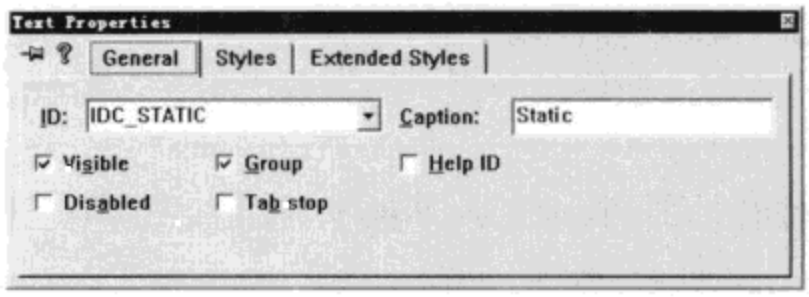


图 4-8　静态控件属性窗口

属性窗口有多个选项卡，General 选项卡可设置控件通用属性，ID 组合框设置控件的 ID，一个对话框窗口中每个控件的 ID 值是唯一的，一个 ID 标识对应一个数字，用以区分不同控件。Caption 编辑框设置控件显示的内容，默认文本为“Static”，Visible 复选框设置控件是否可见，Disabled 复选框设置控件是否可用，若选中则控件变为灰色不可用。

> **Tips**　单击左上角的图标变成，属性窗口可一直保持显示状态。

（6）设置 ID 组合框为 IDC_LABEL，在 Caption 编辑框里输入“静态文本控件测试”。切换到 Styles 选项卡，如图 4-9 所示。

Styles 选项卡可设置控件的外观样式，Align text 组合框设置文本对齐方式，Border 复选框设置是否显示边框，No wrap 复选框设置是否不换行。Extended Styles 选项卡可设置控件扩展样式。

（7）选中 Border 复选框，取消选中 No wrap 复选框，鼠标调整静态控件的大小使内容可全部显示。

（8）生成程序并运行，如图 4-10 所示。

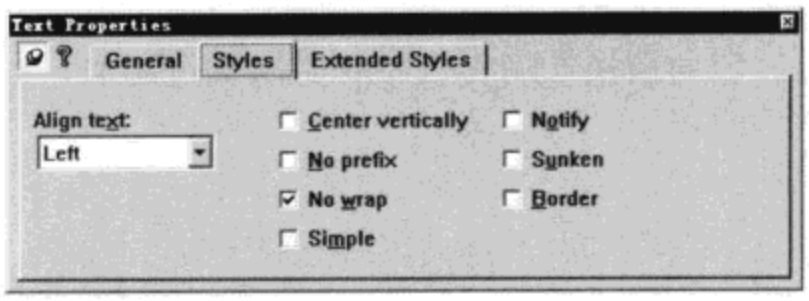


图 4-9　静态控件属性 Styles 窗口

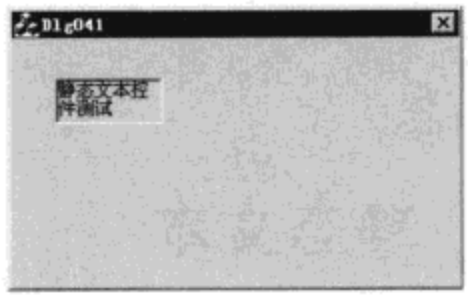


图 4-10　运行窗口

4.2.2　更新内容

在界面布局窗口可手动设置控件属性，也可使用代码设置控件属性，更具灵活性。对话框作为控件的父窗口，对应 CDialog 类，静态控件对应 CStatic 类，CDialog、CStatic 类都继承于 CWnd 窗口类。CWnd 类提供窗口的基本功能操作，如设置标题文本、是否可见、是否可用，CWnd 派生类根据自身需要添加新功能。

通过代码操作控件，首先要获取指向控件的 CWnd 窗口类指针，若设置基本属性，使用 CWnd 类就能完成，若设置控件类专有属性，需要将 CWnd 类指针强制转换为控件类指针。如静态文本控件需要强制转换为 CStatic 类指针，再调用 CStatic 类的成员函数。

若使用代码设置控件显示的文本，可先调用 CWnd::GetDlgItem 函数获取控件的 CWnd 类指针，再调用 CWnd::SetWindowText 函数设置控件的文本。

GetDlgItem 函数获取对话框内的子窗口或控件的窗口类指针，格式如下：

```
CWnd* CWnd::GetDlgItem( int nID ) const
```

参数如下。

- nID：控件或子窗口的 ID 值。

返回值：控件或子窗口的窗口类指针，若找不到该 ID 值对应项，返回 NULL。

Tips　在 Visual C++环境中 NULL 即为 0，表示一个指针值为空，不指向任何变量。

SetWindowText 函数设置窗口标题或控件文本内容，格式如下：

```
void CWnd::SetWindowText( LPCTSTR lpszString)
```

参数如下。

- lpszString：以 null 结尾的字符串，要显示的文本内容。

Tips　LPCTSTR 是 Win32 自定义数据类型，LP 表示长指针，C 表示 const 常量，T 表示 TCHAR，STR 表示字符串，等同于 const TCHAR*。在 Win32 系统中指针没有长短之分，LP 和 P 等同。由于字符集分为 ANSI 和 Unicode 两种，对应 char 和 wchar_t 两种字符类型，TCHAR 是个宏定义，它根据开发环境的设置不同自动替换为 char 或 wchar_t，使用 TCHAR 可以增强程序的可移植性。

（1）切换到 ClassView 标签，双击 CDlg041Dlg 类下的 OnInitDialog 项，定位到函数，在函数最后一句 return TRUE; 之前，添加如下代码：

```
GetDlgItem(IDC_LABEL)->SetWindowText("更新控件内容");
```

OnInitDialog 函数用于对话框的初始化，在对话框窗口创建完成、显示之前调用，可在该函数内添加控件或变量的初始化操作。GetDlgItem 函数获取 ID 值为 IDC_LABEL 的控件的窗口类指针，SetWindowText 函数设置控件显示的文本内容。

（2）生成程序并运行，如图 4-11 所示。

Tips　Visual C++自带的 CStatic 类功能有限，无法设置颜色、粗体、斜体、阴影等效果。若要扩展控件，可创建一个从 CStatic 类派生的类，在派生类的重绘函数 OnPaint 里实现显示效果，如图 4-12 所示。

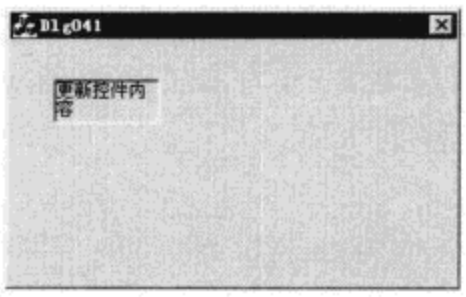


图 4-11　更新控件内容

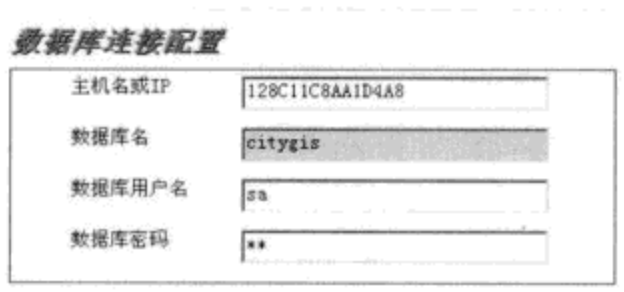


图 4-12　增强效果的静态控件

4.3 编辑框

编辑框用于显示和输入文字信息，如登录系统时，在编辑框中输入用户名和密码。编辑框可设置输入格式，如输入密码时用*替代真实字符，输入数据时控制其只能输入数字。

4.3.1 设置属性

【实例 4-3】在 Dlg041 工程中添加编辑框控件，设置控件属性，使用 CEdit 类和数据交换两种方式更新控件。使用 CEdit 类方式时，静态文本控件显示编辑框中的数值的倒数值。使用数据交换方式时，在编辑框中输入数值的同时，静态文本控件同步显示当前输入数值的倒数值。

① 打开工程 Dlg041，切换到资源视图，展开 Dialog 节点，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口，在控件窗口单击编辑框控件 abl，在对话框中单击放置控件，如图 4-13 所示。

② 右键单击添加的编辑框控件，在弹出的快捷菜单中选择 Properties 命令，打开 Edit Properties 对话框，切换到 Styles 选项卡，如图 4-14 所示。

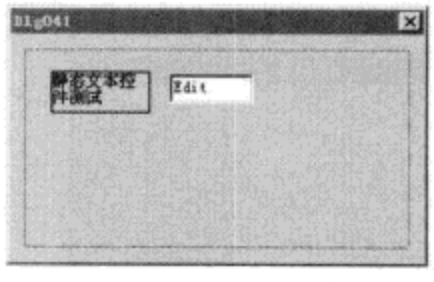


图 4-13　编辑框控件

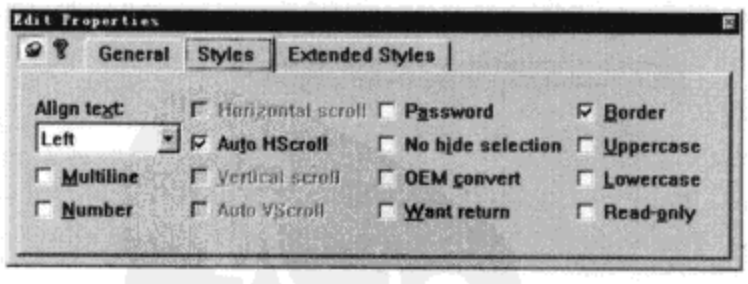


图 4-14　编辑框属性窗口

Align text 组合框设置控件内文本的对齐方式，Auto HScroll 复选框设置当文字长度超出控件显示范围时，是否在水平方向滚动显示，Password 复选框设置文本是否以密码形式显示，Multiline 复选框设置是否多行显示，Number 复选框设置是否限制输入内容为数字格式，Read-only 复选框设置是否只读。

③ 勾选 Number 复选框，限制输入内容为数字格式。

4.3.2 数据交换

在介绍数据交换之前，先使用编辑框对应的 CEdit 类操作编辑框。类似于静态文本控件，先获取控件的窗口类指针，强制转换为 CEdit 类指针，再调用 CEdit 类的成员函数实现特定功能。

① 切换到类视图，双击 CDlg041Dlg 类下的 OnInitDialog 项，定位到函数，在最后一句 return TRUE; 之前，添加如下代码：

```
CEdit* pEdit=(CEdit*)GetDlgItem(IDC_EDIT1);          //获取编辑框指针
pEdit->SetLimitText(6);                              //限制字符数目
pEdit->SetPasswordChar('#');                         //设置密码显示字符
pEdit->SetWindowText("123");                         //设置显示文本
```

GetDlgItem 函数获取 ID 值为 IDC_EDIT1 的控件的窗口类指针，强制转换为 CEdit 类指针，再调用 CEdit 类的成员函数。SetLimitText 函数限制输入的字节总数最多为 6，SetPasswordChar 函数设置密码显示字符为#，SetWindowText 函数设置编辑框内容为 123。

SetLimitText 函数限制编辑框输入的字节总数，格式如下：

```
void CEdit::SetLimitText(UNIT nMax)
```

参数如下。

❑ nMax：最多输入的字节总数，其中一个英文字符占用 1 个字节，一个汉字占用 2 个字节。

SetPasswordChar 函数设置编辑框显示的字符，用该字符替代所有可视字符，格式如下：

```
void CEdit::GetPasswordChar(TCHAR ch)
```

参数如下。

❑ ch：密码字符，若为 0，显示实际字符。

② 生成程序并运行，如图 4-15 所示。由于设置了密码字符，文本内容显示为三位密码，删除内容重新输入，可发现只能输入数字，且最多输入六位。

③ 接着上面的代码，添加如下代码：

```
CString strEdit;
pEdit->GetWindowText(strEdit);                       //获取编辑框的文本内容
double nEdit=atof(strEdit);                          //将字符串转为浮点数
double dValue=1.0/nEdit;                             //计算倒数值
strEdit.Format("%f",dValue);                         //将浮点值格式化为字符串
GetDlgItem(IDC_LABEL)->SetWindowText(strEdit);       //在静态控件中显示字符串格式的倒数
```

④ 重新生成程序并运行，如图 4-16 所示。静态文本控件显示编辑框中数值的倒数值。

CString 类是 MFC 中定义的字符串类，它类似于标准 C++库的 string 类，用于存储可变长度的字符串，所有与字符串相关的操作都可通过 CString 类完成。

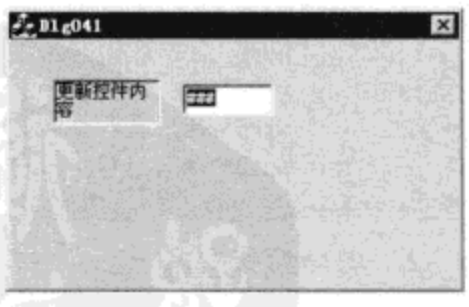


图 4-15 编辑框密码样式图

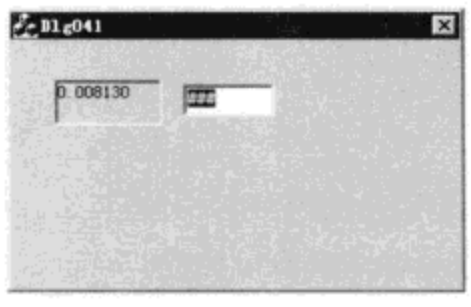


图 4-16 静态控件显示倒数值

GetWindowText 函数获取编辑框的文本内容，存入 CString 类对象 strEdit 中。atof 函数将字符串转为浮点数，Format 函数将浮点值格式化为字符串，SetWindowText 函数设置静态控件的文本内容。

GetWindowText 函数获取控件的文本内容或窗口的标题，格式如下：

```
void CWnd::GetWindowText( CString& rString ) const
```

参数如下。

- ❑ rString：输出参数，存储获取的文本内容。

atof 函数将字符串转换为浮点数，格式如下：

```
double atof( const char *str )
```

参数如下。

- ❑ str：要转换的字符串，必须以数字开头，如“32.12abc”。

返回值：转换后的双精度数值，如 32.12。

Tips atof 中 a 代表字符串，to 代表转换，f 代表浮点数，atoi 可将字符串转为整数（int），atol 可将字符串转为长整数（long）。

Format 函数将各种类型的值格式化为字符串，格式化方式类似于 printf 函数，格式如下：

```
void CString::Format( LPCTSTR lpszFormat, ... )
```

参数如下。

- ❑ lpszFormat：格式化字符串，如“name %s age %d”。
- ❑ ...：可变长度的参数列表，可传入多个参数。

Tips 一般情况下，函数的参数数目是固定的，若使用可变长度的参数列表，如 printf、scanf，可将函数最后一个参数设为...，在函数内部使用 va_start、va_arg、va_end 宏来获取传递的参数值，参见< stdarg.h >。

若编辑框为数字格式，使用 CEdit 类需要经历获取文本、转换为数值、处理数据、再转换为文本多个步骤。Visual C++环境提供一种相对简单的方式：数据交换（Dialog Data Exchange，DDX），为控件添加一个映射变量，当控件内容改变后，变量值同步更新，反之亦然。如将编辑框添加一个 int 类型的变量，当编辑框的值改变后，int 变量的值随之改变，无须格式转换，从而节省工作量。

Visual C++环境还提供一种验证控件输入是否合理的方法：数据验证（Dialog Data Validation，DDV），若控件关联的变量为数值，可限制最大最小值，若为字符串，可限制字符数目。当输入的内容不符合要求时，会弹出错误提示框直到输入正确。

⑤ 切换到资源视图，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口。右键单击编辑框，在弹出的快捷菜单中选择 ClassWizard 命令，弹出 MFC ClassWizard 窗口，切换到 Member Variables 选项卡，如图 4-17 所示。

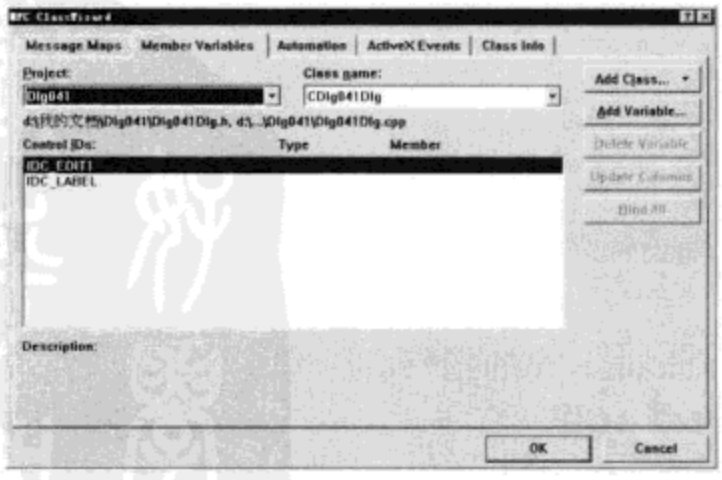


图 4-17　MFC ClassWizard 窗口

⑥ 在 Class name 组合框中选择 CDlg041Dlg 类，双击 IDC_EDIT1 项，弹出 Add Member Variable 窗口，如图 4-18 所示。

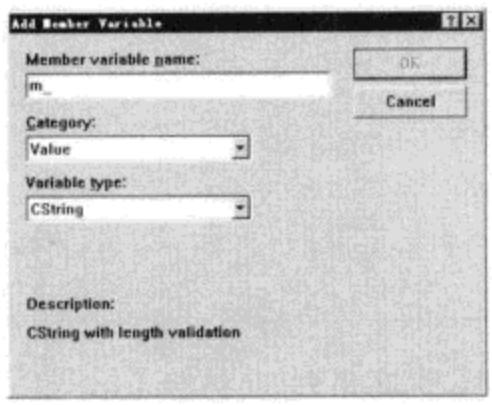


图 4-18 添加成员变量窗口

⑦ 在 Member variable name 编辑框中输入 m_edit1，Category 组合框选择 Value 项，Variable type 组合框选择 int 项，单击 OK 按钮保存并关闭窗口，MFC ClassWizard 窗口显示添加的变量，如图 4-19 所示。

⑧ 选择 IDC_EDIT1 项，在 Minimum Value 编辑框中输入 50，在 Maximum Value 编辑框中输入 500，单击 OK 按钮保存并关闭窗口。

编辑框控件添加一个 int 类型的变量 m_edit1，并设置变量的最小值为 50，最大值为 500。若编辑框中输入的数值不在设定的范围内，自动弹出错误信息提示窗口，直到输入合理的数值。

⑨ 切换到类视图，双击 CDlg041Dlg 类下的 DoDataExchange 项，定位到函数，如图 4-20 所示。

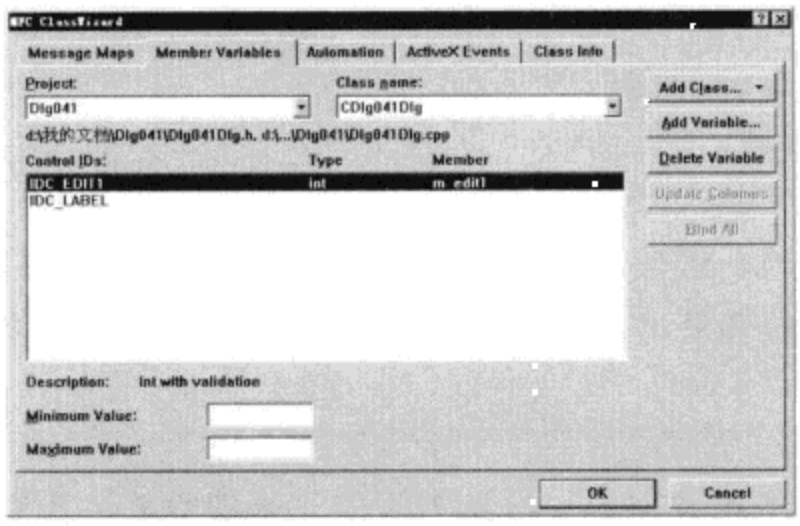


图 4-19 添加变量后的 MFC ClassWizard 窗口

```
void CDlg041Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlg041Dlg)
    DDX_Text(pDX, IDC_EDIT1, m_edit1);
    DDV_MinMaxInt(pDX, m_edit1, 50, 500);
    //}}AFX_DATA_MAP
}
```

图 4-20 DoDataExchange 函数

Tips 在自动生成的代码中，//{{和//}}包括的代码由开发环境自动插入，不能随意修改，否则无法完成代码的自动添加。

DoDataExchange 函数用于实现对话框的数据交换和数据验证，DDX_Text 函数负责编辑框和其映射变量之间的数据交换，格式如下：

```
void DDX_Text(CDataExchange* pDX,int nIDC,int& value)
```

参数如下。

- pDX：指向 CDataExchange 类的指针，用于实现数据交换。
- nIDC：控件的 ID 值。
- value：关联的整型变量引用。

DDV_MinMaxInt 函数用于验证编辑框的输入值是否在设定范围内，格式如下：

```
void DDV_MinMaxInt(CDataExchange* pDX,int value,int minVal,int maxVal)
```

参数如下。

- pDX：指向 CDataExchange 类的指针。
- value：关联的整型变量引用。
- minVal：设定的最小值。

❑ maxVal：设定的最大值。

利用 DDX 数据交换机制，编辑框 IDC_EDIT1 和整型变量 m_edit1 建立了映射关系，当其中一个发生改变后，另一个也随之改变。利用 DDV 数据验证机制，当编辑框的输入值不符合要求时，自动弹出提示框，直到输入符合要求的值。

⑩ 双击 CDlg041Dlg 类下的 OnInitDialog 项，注释掉手动添加的代码，并添加如下代码：

```
/*  注释以下代码
GetDlgItem(IDC_LABEL)->SetWindowText("更新控件内容");
CEdit* pEdit=(CEdit*)GetDlgItem(IDC_EDIT1);
pEdit->SetLimitText(6);
pEdit->SetPasswordChar('#');
pEdit->SetWindowText("123");
CString strEdit;
pEdit->GetWindowText(strEdit);
double nEdit=atof(strEdit);
double dValue=1.0/nEdit;
strEdit.Format("%f",dValue);
GetDlgItem(IDC_LABEL)->SetWindowText(strEdit); */
//新增代码，使用数据交换方式更新编辑框
m_edit1=160;                                    //设置关联变量值
UpdateData(FALSE);                              //更新编辑框控件
```

UpdateData 函数用于数据交换中及时更新控件和映射变量，格式如下：

```
BOOL CWnd::UpdateData( BOOL bSaveAndValidate = TRUE)
```

参数如下。

❑ bSaveAndValidate：若为 TRUE，获取控件数据，用控件的值更新变量。若为 FALSE，更新控件数据，用变量值更新控件。

⑪ 生成程序并运行，如图 4-21 所示。编辑框的值更新为 160。

当编辑框中的值改变时，触发编辑框的 EN_CHANGE 消息，可添加该消息的处理函数，在该函数中计算倒数值，并在静态控件中显示，实现同步更新。

⑫ 双击 CDlg041Dlg 类的 DoDataExchange 项，定位到函数，注释下面一句代码，去除编辑框的最大最小值限制，便于自由输入数值。

```
//DDV_MinMaxInt(pDX, m_edit1, 50, 500);  //注释该行代码
```

⑬ 切换到资源视图，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口，右键单击编辑框，在弹出的快捷菜单中选择 ClassWizard 命令，弹出 MFC ClassWizard 窗口，如图 4-22 所示。

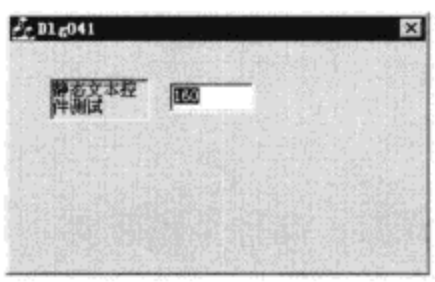


图 4-21　数据交换测试

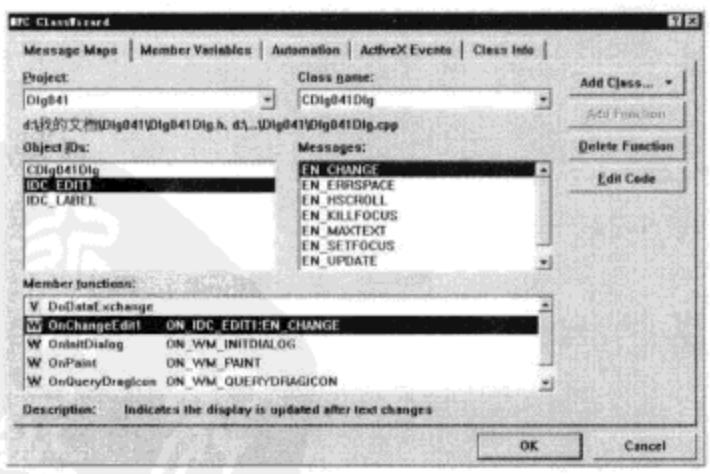


图 4-22　添加控件消息处理函数

⑭ 在 Class name 组合框中选择 CDlg041Dlg 项，Object IDs 列表框选择 IDC_EDIT1 项，Messages 列表框选择 EN_CHANGE 项，单击 Add Function 按钮，弹出 Add Member Function 窗口，单击 OK 按钮，再单击 Edit Code 按钮完成消息处理函数的添加，并定位到函数，添加如下

代码：

```
void CDlg041Dlg::OnChangeEdit1()
{
    UpdateData();                                          //更新映射变量的值
    double dValue=1.0/m_edit1;                             //计算倒数值
    CString strEdit;
    strEdit.Format("%f",dValue);                           //将计算结果格式化为字符串
    GetDlgItem(IDC_LABEL)->SetWindowText(strEdit);         //在静态控件中显示计算结果
}
```

当编辑框中输入值改变时，自动调用该函数。UpdateData 函数获取输入值，并更新 m_edit1 变量的值。dValue 为输入值的倒数值，Format 函数将倒数值格式化为字符串，存入 strEdit 中。

⑮ 生成程序并运行，如图 4-23 所示。在编辑框中输入任意数值，静态控件同步显示其倒数值。

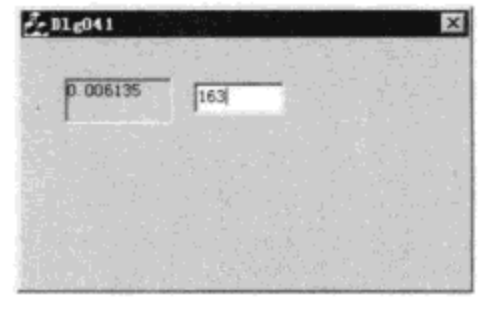


图 4-23　控件同步更新

4.4 按钮

按钮（button）是一个小的矩形窗口，可以设置标题文字，常用于执行一项操作，如保存操作结果、打开一个窗口等。按钮被单击时触发 BN_CLICKED 消息，可添加该消息的处理函数，当单击按钮时，自动调用该函数，实现特定的功能。

4.4.1 设置属性

【实例 4-4】在 Dlg041 工程中添加按钮控件，单击按钮后，对话框窗口每隔一定时间，随机移动一次窗口位置。

① 打开工程 Dlg041，切换到资源视图，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口，拖放按钮控件 到对话框模板中，如图 4-24 所示。

② 右键单击按钮控件，在弹出的快捷菜单中选择 Properties 命令，弹出 Push Button Properties 窗口，设置 Caption 编辑框为“移动”。切换到 Styles 选项卡，如图 4-25 所示。

Default button 复选框设置是否为默认按钮，一个对话框只有一个默认按钮，按下 Enter 键等同于单击默认按钮。Multiline 复选框设置是否可显示多行文字，换行符用‘\n’表示。

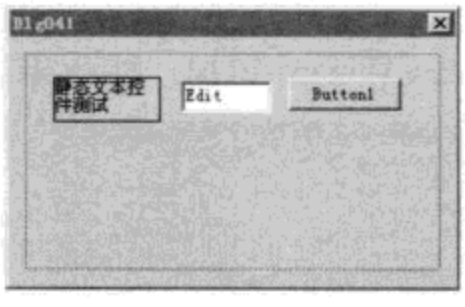


图 4-24　添加按钮控件

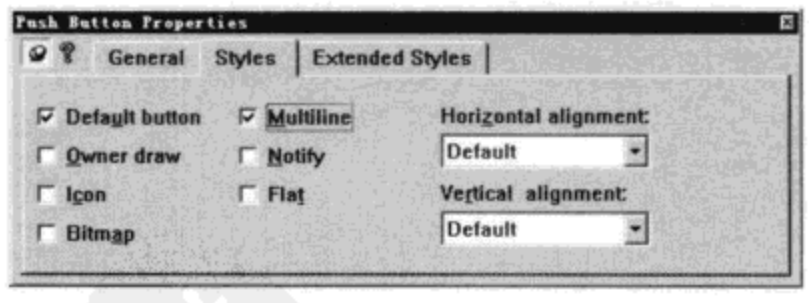


图 4-25　按钮控件属性窗口

4.4.2 消息响应

按钮一般响应左键单击 BN_CLICKED 事件，双击按钮控件即可添加单击事件响应函数。程序运行时，用鼠标左键单击按钮后，自动调用该函数。

（1）双击按钮控件，添加单击事件处理函数，在函数内添加如下代码：

```
void CDlg041Dlg::OnButton1()
{
    SetTimer(1,500,NULL);                                  //启动时钟 1，时间间隔为半秒
}
```

SetTimer 函数用于启动时钟定时器，定时器每隔一定时间，向程序发送一次 WM_TIMER 消息，格式如下：

```
UINT CWnd::SetTimer(UINT nIDEvent, UINT nElapse, void (CALLBACK EXPORT*lpfnTimer)
(HWND, UINT, UINT, DWORD) )
```

参数如下。

- nIDEvent：定时器标识符，不能为 0。
- nElapse：时间间隔，单位为毫秒（ms）。
- lpfnTimer：处理 WM_TIMER 消息的回调函数，若为 NULL，消息放入队列中。

返回值：新定时器的标识符，若失败则返回 0。

（2）在类视图右键单击 CDlg041Dlg 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，弹出 New Windows Message 窗口，选择 WM_TIMER 项，单击 Add and Edit 按钮添加 WM_TIMER 消息处理函数，添加如下代码：

```
void CDlg041Dlg::OnTimer(UINT nIDEvent)
{
    CDialog::OnTimer(nIDEvent);
    if(nIDEvent==1)                                    //若时钟 ID 为 1
    {
        CRect rcWnd;
        GetWindowRect(rcWnd);                          //获取窗口的矩形边界
        MoveWindow(rand()%800,rand()%600,rcWnd.Width(),rcWnd.Height());
                                                       //移动窗口到随机位置
    }
}
```

CRect 类是 MFC 中定义的矩形类，可以存放矩形的左上角和右下角坐标值，并提供一系列矩形操作函数。该类重载了 LPRECT 类型转换，在任何需要 LPRECT 参数的地方，可用 CRect 类对象替代。常用函数如下。

- Width：获取矩形的宽度。
- Height：获取矩形的高度。
- TopLeft：获取矩形左上角的坐标值。
- BottomRight：获取矩形右下角的坐标值。
- CenterPoint：获取矩形中心点的坐标值。
- PtInRect：判断某个点是否在矩形内。
- InflateRect：扩大矩形的宽度和高度。
- DeflateRect：缩小矩形的宽度和高度。

GetWindowRect 函数获取窗口的矩形边界，包括标题栏和边框，格式如下：

```
void CWnd::GetWindowRect(LPRECT lpRect) const
```

参数如下。

- lpRect：CRect 对象或指向 RECT 结构的指针，用于存放窗口的矩形边界。

MoveWindow 函数改变窗口的大小和位置，若为顶层窗口，相对于屏幕左上角移动，若为子窗口，相对于父窗口客户区的左上角移动，格式如下：

```
void CWnd::MoveWindow(int x,int y,int nWidth,int nHeight,BOOL bRepaint=TRUE)
```

参数如下。

- x：窗口左边界的新值。
- y：窗口上边界的新值。
- nWidth：窗口宽度的新值。

- nHeight：窗口高度的新值。
- bRepaint：是否重画窗口，默认为 TRUE。

启动时钟 1 后，程序每隔 0.5 秒收到一次 WM_TIMER 消息，自动调用 OnTimer 函数。在函数内部通过 GetWindowRect 函数获取窗口的矩形边界，存放到 rcWnd 中，调用 MoveWindow 函数移动窗口。rand 函数返回一个随机值，Width、Height 函数获取矩形边界的宽度和高度，新窗口的大小保持不变，位置是随机变化的。

（3）在类视图右键单击 CDlg041Dlg 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，弹出 New Windows Message 窗口，选择 WM_DESTROY 项，单击 Add and Edit 按钮，添加 WM_DESTROY 消息处理函数，添加如下代码：

```
void CDlg041Dlg::OnDestroy()
{
    CDialog::OnDestroy();
    KillTimer(1);                                    //销毁时钟 1
}
```

KillTimer 函数销毁 SetTimer 函数创建的时钟，时钟也是一种资源，用完后应释放，格式如下：

```
BOOL CWnd::KillTimer(int nIDEvent)
```

参数如下。

- nIDEvent：时钟 ID 值，在 SetTimer 函数中定义。

返回值：若成功返回非零值，若找不到指定 ID 的时钟则返回 0。

当窗口关闭时，自动调用 OnDestroy 函数，可在该函数中做销毁窗口前的一些清理工作。调用 KillTimer 函数销毁指定 ID 的时钟。

（4）生成程序并运行，如图 4-26 所示。单击“移动”按钮，窗口每隔 0.5 秒自动移动到随机位置，按 Esc 键可退出程序。

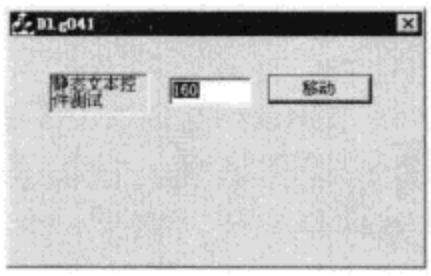


图 4-26　窗口移动

4.5 单选按钮

单选按钮（radio button）是一组互斥选择的按钮，同一个按钮组中一次只能选择一个单选按钮。同一对话框中若有多个单选按钮，默认属于同一个分组，可设置单选按钮的 Group 属性，设为多个分组，每个分组中只能选择一个单选按钮，不同分组间互不影响。

4.5.1 设置属性

【实例 4-5】在 Dlg041 工程中添加 4 个单选按钮控件，分为两组，第一组单选按钮更新静态控件的内容，第二组控制静态控件的显示状态。

（1）打开工程 Dlg041，切换到资源视图，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口，拖放四个单选按钮控件◉到对话框模板中，如图 4-27 所示。

若要调整对话框模板中的控件布局（layout），使用 Dialog 工具栏，如图 4-28 所示。按钮功能依次为：对话框测试，左对齐、右对齐、顶部对齐、底部对齐，垂直居中、水平居中，水平间距相等、垂直间距相等，宽度相同、高度相同、大小相同，显示格网、显示标尺。选择多个要对齐的控件，其中最后一个选择的控件作为对齐的标准。

（2）右键单击单选按钮控件，在弹出的快捷菜单中选择 Properties 命令，弹出 Radio Button Properties 窗口，依次设置 Caption 编辑框为“男”、“女”、“显示”、“隐藏”，依次设置 ID 编辑框为 IDC_RADIO_MAN、IDC_RADIO_WOMAN、IDC_RADIO_SHOW、IDC_RADIO_HIDE。

（3）选择 Layout|Tab Order 命令或按 Ctrl+D 快捷键，显示出所有控件的 Tab 键顺序，用鼠标依次单击控件，重新设定控件的 Tab 键顺序，如图 4-29 所示。

对话框某一时刻只有一个控件处于焦点（focus）状态，使用 Tab 键可切换焦点控件，处于活动状态的焦点控件可以接收键盘消息，在对话框设计时可设定控件的 Tab 键顺序，按下 Tab 键时根据顺序切换焦点。

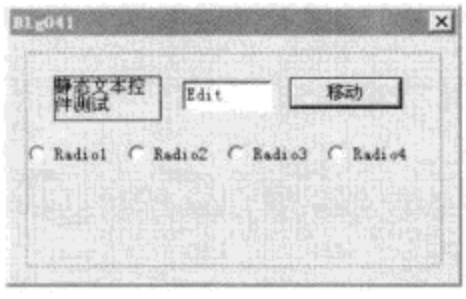


图 4-27　添加单选按钮

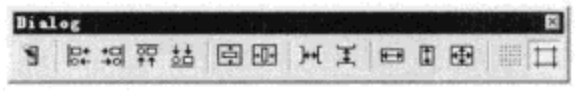


图 4-28　Dialog 工具栏

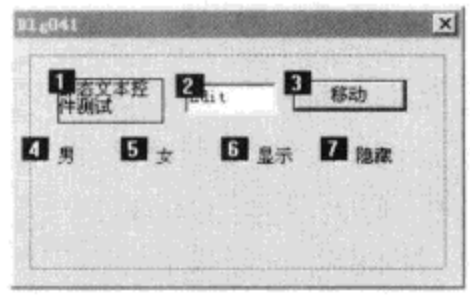


图 4-29　控件 Tab 键顺序

（4）右键单击“男”单选按钮，在弹出的快捷菜单中选择 Properties 命令，在 General 选项卡里勾选 Group 复选框，再勾选“显示”单选按钮的 Group 复选框。

默认状态下所有单选按钮为一组，只能选择其一。若要分为多组，根据 Tab 键的顺序，勾选每个分组的第一个单选按钮的 Group 复选框，则该分组包括其 Tab 键后的所有单选按钮，直到另一个 Group 出现。

如四个单选按钮的 Tab 键为 4、5、6、7，其中 4、6 所在的单选按钮勾选了 Group 复选框，则 4、5 所在的单选按钮构成分组 1，6、7 所在的单选按钮构成分组 2。

4.5.2　消息响应

单选按钮和按钮都对应 CButton 类，一般响应鼠标单击事件，双击单选按钮添加消息处理函数。程序运行时，用鼠标左键单击单选按钮后，自动调用该函数。

（1）分别双击四个单选按钮，添加四个消息处理函数，添加如下代码：

```
void CDlg041Dlg::OnRadioMan() {
    GetDlgItem(IDC_LABEL)->SetWindowText("男");          //设置文本内容
}
void CDlg041Dlg::OnRadioWoman() {
    GetDlgItem(IDC_LABEL)->SetWindowText("女");
}
void CDlg041Dlg::OnRadioShow() {
    GetDlgItem(IDC_LABEL)->ShowWindow(TRUE);             //设置显示状态
}
void CDlg041Dlg::OnRadioHide() {
    GetDlgItem(IDC_LABEL)->ShowWindow(FALSE);
}
```

ShowWindow 函数用于设置窗口的显示状态，格式如下：

```
BOOL CWnd::ShowWindow(int nCmdShow)
```

参数如下。

- nCmdShow：显示状态，若为 FALSE 或 SW_HIDE 隐藏窗口，若为 TRUE 或 SW_SHOWNORMAL 激活并显示窗口。

返回值：若窗口先前可见，则返回非零值，若窗口先前不可见，则返回 0。

（2）生成程序并运行，如图 4-30 所示。单击“男”或“女”单选按钮，改变静态控件文本内容，单击“显示”或“隐藏”单选按钮，改变静态控件显示状态。

4.6　复选按钮

不同于单选按钮，复选按钮（check box）互不影响，可以勾选零个或多个，常用于窗口的

属性设置。按钮、单选按钮、复选框都对应 CButton 类，通过 CButton 类的 GetCheck 函数可获取复选按钮的选中状态，实现特定功能。

4.6.1 设置属性

【实例 4-6】在 Dlg041 工程中添加两个复选框控件，第一个复选框控制“移动”按钮的可用状态，第二个控制 4 个单选按钮的可用状态。

① 打开工程 Dlg041，切换到资源视图，双击 IDD_DLG041_DIALOG 项，打开对话框设计窗口，拖放两个复选框控件到对话框模板中。

② 右键单击复选框控件，在弹出的快捷菜单中选择 Properties 命令，弹出 Check Box Properties 窗口，依次设置 Caption 编辑框为“按钮”、“单选按钮”，依次设置 ID 编辑框为 IDC_CHECK_BTN、IDC_CHECK_RADIO_BTN，如图 4-31 所示。

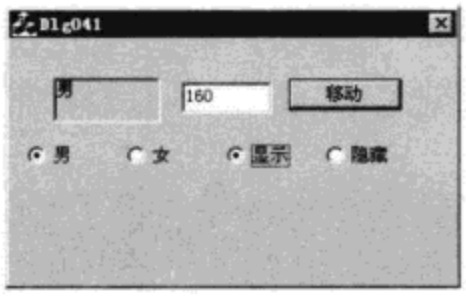


图 4-30 单选按钮消息响应

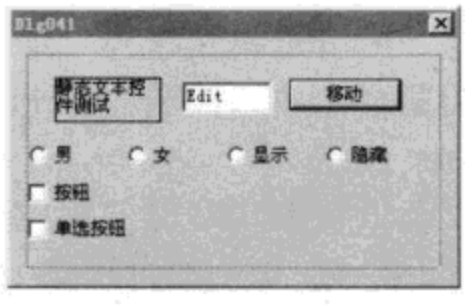


图 4-31 添加复选框按钮

4.6.2 消息响应

(1)在类视图双击 CDlg041Dlg 类下的 OnInitDialog 项，定位到函数，在最后一句 return TRUE; 之前，添加如下代码：

```
CButton* pBtn1=(CButton*)GetDlgItem(IDC_CHECK_BTN);    //获取复选框的 CButton 类指针
pBtn1->SetCheck(TRUE);                                  //选中复选框
CButton* pBtn2=(CButton*)GetDlgItem(IDC_CHECK_RADIO_BTN);
pBtn2->SetCheck(TRUE);
```

对话框初次显示前，调用 OnInitDialog 函数进行初始化，选中两个复选框。GetDlgItem 函数获取控件的 CWnd 类指针，并强制转换为 CButton 类指针。SetCheck 函数设置复选框的勾选状态，格式如下：

```
void CButton::SetCheck(int nCheck)
```

参数如下。

- nCheck 选中状态，0 为不选，1 为选中。

(2) 双击两个复选框，添加两个消息处理函数，添加如下代码：

```
void CDlg041Dlg::OnCheckBtn()
{
    CButton* pBtn1=(CButton*)GetDlgItem(IDC_CHECK_BTN);          //获取复选框的指针
    GetDlgItem(IDC_BUTTON1)->EnableWindow(pBtn1->GetCheck());    //设置复选框是否可用
}
void CDlg041Dlg::OnCheckRadioBtn()
{
    CButton* pBtn1=(CButton*)GetDlgItem(IDC_CHECK_RADIO_BTN);
    GetDlgItem(IDC_RADIO_MAN)->EnableWindow(pBtn1->GetCheck());
    GetDlgItem(IDC_RADIO_WOMAN)->EnableWindow(pBtn1->GetCheck());
    GetDlgItem(IDC_RADIO_SHOW)->EnableWindow(pBtn1->GetCheck());
    GetDlgItem(IDC_RADIO_HIDE)->EnableWindow(pBtn1->GetCheck());
}
```

EnableWindow 函数设置窗口或控件是否可用，不可用时变为灰色，格式如下：

```
BOOL CWnd::EnableWindow(BOOL bEnable = TRUE)
```

参数如下。

- bEnable：是否可用，默认为 TRUE 可用。

返回值：若窗口或控件先前不可用返回非零值，否则返回 0。

GetCheck 函数获取复选框的选中状态，格式如下：

```
int CButton::GetCheck() const
```

返回值：若选中返回 1，未选中返回 0。

（3）生成程序并运行，如图 4-32 所示。单击“按钮”复选框，控制“移动”按钮是否可用，单击“单选按钮”复选框，控制 4 个单选按钮是否可用。

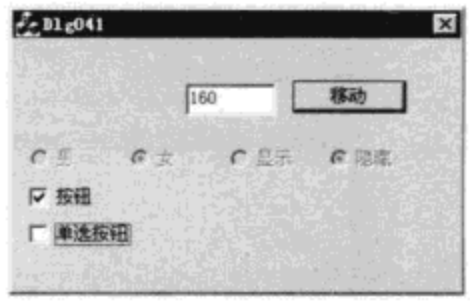


图 4-32　复选按钮响应

4.7 组合框

组合框（combo box）是一种可以编辑和选择项的控件，用键盘输入或从下拉列表框中选择一项，用户单击下拉箭头时弹出列表框，选择一项或者失去焦点后隐藏列表框。如 IE 浏览器的地址栏，既可以输入也可以选择，组合框由编辑框、按钮、列表框三种控件组合而成。

4.7.1 设置属性

【实例 4-7】新建一个对话框工程名为 Dlg042，实现两个组合框控件联动显示，第一个组合框选择项改变后，第二个组合框自动显示相关项，使用扩展组合框显示图像。

（1）新建一个对话框工程名为 Dlg042，在资源视图双击 IDD_DLG042_DIALOG 项，拖放两个组合框控件、两个静态文本控件到对话框模板中，如图 4-33 所示。单击组合框控件的箭头，设置下拉列表框的高度。

（2）依次设置两个静态文本控件属性窗口的 Caption 编辑框为“类别”、“分类”。右键单击“类别”对应的组合框，在弹出的快捷菜单中选择 Properties 命令，弹出 Combo Box Properties 窗口，在 General 选项卡里设置 ID 为 IDC_COMBO_CLASS1，切换到 Styles 选项卡，如图 4-34 所示。

若勾选 Sort 复选框，启用自动排序，组合框根据其项元素的字母顺序进行排序，若需要根据选中项的索引做判断，应取消选中该项。组合框有三种显示方式，Simple 类型始终显示列表框且可编辑选择，DropDown 类型可编辑选择，Drop List 类型只能选择不可编辑，若不需要用户输入，一般设置为 Drop List。

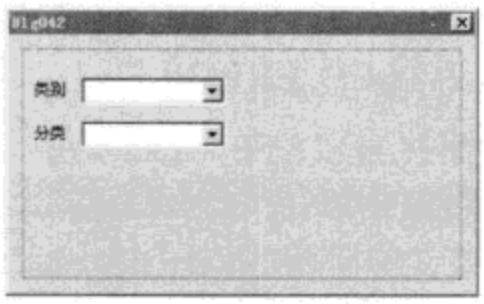


图 4-33　添加组合框控件

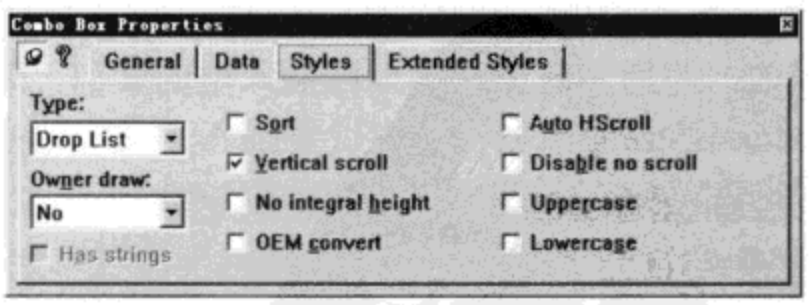


图 4-34　组合框 Styles 选项卡

（3）在 Styles 选项卡里，取消勾选 Sort 复选框，Type 组合框选择 Drop List 项。同理，设置“分类”对应的组合框的 ID 为 IDC_COMBO_CLASS2，取消勾选 Sort 复选框，选择 Drop List 类型。

（4）右键单击“类别”组合框，在弹出的快捷菜单中选择 ClassWizard 命令，弹出 MFC ClassWizard 窗口，切换到 Member Variables 选项卡，如图 4-35 所示。双击 IDC_COMBO_ CLASS1

项，弹出 Add Member Variable 窗口，在 Category 组合框中选择 Control 项，在 name 编辑框中输入 m_combo1，单击 OK 按钮保存并退出。同理，双击 IDC_COMBO_CLASS2 项，添加 CComboBox 类型的变量 m_combo2。单击 OK 按钮保存并退出。

组合框对应 CComboBox 类，该类提供一系列函数操作组合框。通过为组合框控件添加 CComboBox 类变量，可以通过类对象操作组合框控件。也可通过调用 GetDlgItem 函数获取控件的 CWnd 类指针，再强制转换为 CComboBox 类指针，但不如添加类变量方式方便。

（5）在类视图双击 CDlg042Dlg 类下的 DoDataExchange 项，定位到函数，如图 4-36 所示。

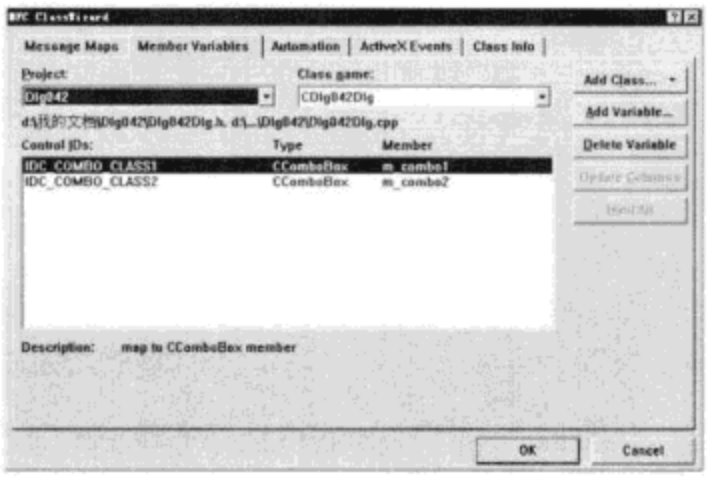


图 4-35　添加控件变量

```
void CDlg042Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlg042Dlg)
    DDX_Control(pDX, IDC_COMBO_CLASS2, m_combo2);
    DDX_Control(pDX, IDC_COMBO_CLASS1, m_combo1);
    //}}AFX_DATA_MAP
}
```

图 4-36　DoDataExchange 函数

DoDataExchange 函数实现了控件和变量的数据交换机制，当控件操作改变后，控件对应的变量也随之改变，反之亦然。DDX_Control 函数建立控件和类变量的映射关系，格式如下：

```
void DDX_Control(CDataExchange* pDX,int nIDC,CWnd& rControl)
```

参数如下。

- pDX：指向 CDataExchange 对象的指针。
- nIDC：控件的 ID 值。
- rControl：关联到控件的窗口类引用。

4.7.2　编辑项

CComboBox 类提供一系列函数操作组合框控件，可在运行时动态改变组合框的内容，常用的成员函数如下所示：

（1）AddString 函数用于在列表末尾添加一个字符串，格式如下：

```
int CComboBox::AddString(LPCTSTR lpszString)
```

参数如下。

- lpszString：要添加的字符串。

返回值：若插入成功，则返回插入项的索引，若失败，则返回 CB_ERROR 即-1。

（2）InsertString 函数在指定位置插入一个字符串，格式如下：

```
int CComboBox::InsertString(int nIndex, LPCTSTR lpszString)
```

参数如下。

- nIndex：插入位置的索引，若为-1，则相当于 AddString 函数。
- lpszString：要添加的字符串。

返回值：返回插入项的下标。

（3）DeleteString 函数在指定位置删除一个字符串，格式如下：

```
int CComboBox::DeleteString(UINT nIndex)
```

参数如下。

❑ nIndex：删除项的索引。

返回值：剩余字符串的数目，若 nIndex 超出索引范围，则返回 CB_ERROR。

（4）ResetContent 函数用于清空组合框中的内容，格式如下：

```
void CComboBox::ResetContent()
```

（5）GetCount 函数返回列表项的数目，格式如下：

```
int CComboBox::GetCount() const
```

返回值：列表项数目。

（6）GetCurSel 函数获取当前选中项的下标值，格式如下：

```
int CComboBox::GetCurSel() const
```

返回值：当前选中项的下标，若没有选中项，则返回 CB_ERROR。

（7）SetCurSel 函数设置当前选中项，格式如下：

```
int CComboBox::SetCurSel(int nSelect)
```

参数如下。

❑ nSelect：要选中项的下标，若为-1 则取消选中项。

返回值：选中项的下标，若 nSelect 超出索引范围，则返回 CB_ERROR。

（8）GetLBText 函数获取指定索引的字符串，格式如下：

```
void CComboBox::GetLBText(int nIndex, CString& rString) const
```

参数如下。

❑ nIndex：要获取字符串的索引。

❑ rString：CString 类对象引用，用于存放字符串。

4.7.3 消息响应

在操作组合框时，会触发一些消息，常用的消息有 CBN_SELCHANGE 和 CBN_CLOSEUP。当展开的列表框被收回后，触发 CBN_CLOSEUP 消息，当组合框选择项改变后，触发 CBN_SELCHANGE 消息，其中 CBN_表示 ComboBox Notify 通知消息，CLOSEUP 表示收回关闭，SELCHANGE 表示选择改变。可添加消息处理函数，当触发某消息时，自动调用对应函数。

（1）在类视图右键单击 CDlg042Dlg 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个类成员变量，在 Type 编辑框中输入 CStringArray，在 Name 编辑框中输入 m_arrayStr，如图 4-37 所示。单击 OK 按钮保存并退出。

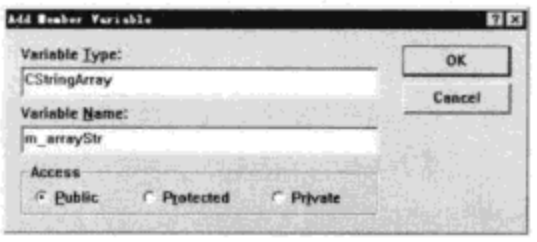


图 4-37　添加类成员变量

CStringArray 类是用于存放 CString 对象的可变数组，在运行时能任意添加删除元素，常用函数如下。

❑ Add：在数组尾部添加一个元素。

❑ InsertAt：在指定位置插入一个元素。

❑ RemoveAt：删除指定位置开始的一个或多个元素。

❑ SetAtGrow：设置指定位置元素的值，若索引参数超出范围，数组会自动增长。

❑ SetAt：设置指定位置元素的值，但不会自动增长。

❑ GetAt：获取指定位置元素的值。

❑ GetSize：获取数组元素的数目。

❑ SetSize：设置数组的大小，若确定数组大致长度，可一次分配相应的内存，避免每次添加元素都重新分配内存。

❑ RemoveAll：清空数组。

（2）双击 CDlg042Dlg 类下的 OnInitDialog 项，定位到函数，在最后一句 return TRUE; 前添加如下代码：

```
m_arrayStr.Add("音乐");                          //动态数组添加元素
m_arrayStr.Add("乐器");
m_arrayStr.Add("");
m_arrayStr.Add("通俗");
m_arrayStr.Add("摇滚");
m_arrayStr.Add("古典");
m_arrayStr.Add("");
m_arrayStr.Add("电吉他");
m_arrayStr.Add("贝斯");
m_arrayStr.Add("键盘");
m_arrayStr.Add("架子鼓");
for(int i=0;i<m_arrayStr.GetSize();i++)          //遍历所有元素
{
  CString str=m_arrayStr.GetAt(i);               //获取第 i 个元素的值
  if(str=="")                                    //若为空字符，则停止遍历
    break;
  m_combo1.AddString(str);                       //将第 i 个元素的值添加到组合框 1 中
}
```

Add 函数用于为动态数组添加元素，其中前 2 个元素作为“类别”，中间 3 个元素是“音乐”包含的分类，后面 4 个元素是“乐器”包含的分类，空字符元素""作为分隔标志。

GetSize 函数获取数组大小，GetAt 函数获取指定位置元素的值，AddString 函数在组合框尾部添加字符串。通过 for 循环遍历所有元素，将第一个空字符元素""前的所有元素添加到组合框 1 中。

（3）按 Ctrl+W 快捷键，打开类向导窗口，选择 Message Maps 选项卡，Object IDs 列表框选择 IDC_COMBO_CLASS1 项，Messages 列表框选择 CBN_SELCHANGE 项，单击 Add Function 按钮添加消息处理函数，如图 4-38 所示。

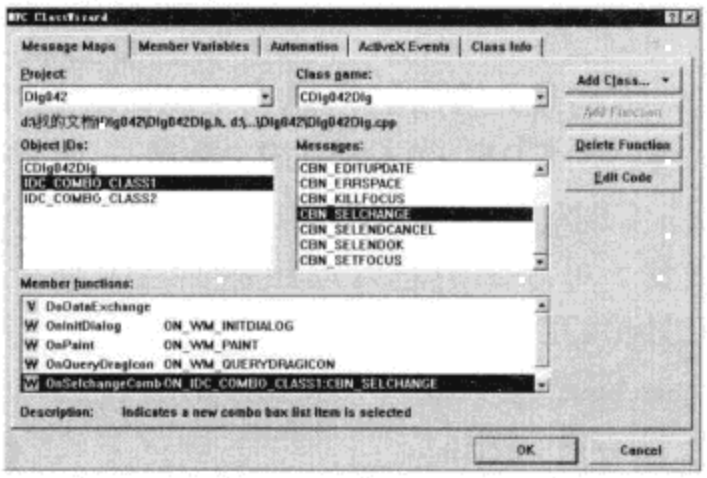


图 4-38　组合框添加消息处理函数

（4）单击 Edit Code 按钮编辑函数，添加如下代码：

```
void CDlg042Dlg::OnSelchangeComboClass1()
{
    m_combo2.ResetContent();                     //清空组合框 2 的内容
    int nSel=m_combo1.GetCurSel();               //获取组合框 1 选中项的索引(从 0 开始)
    int nFlag=-1;                                //标识符，记录空字符""的数目
    for(int i=0;i<m_arrayStr.GetSize();i++)      //遍历数组
    {
        if(m_arrayStr.GetAt(i)=="")              //若为空字符
        {
            nFlag++;                             //标识符加 1
            continue;                            //进入下次循环
        }
        if(nFlag==nSel)                          //若选中项的索引等于已读到的空字符数
```

```
            m_combo2.AddString(m_arrayStr.GetAt(i));  //将当前元素添加到组合框 2 中
        if(nFlag>nSel)                                //分类元素添加结束
            break;
    }
    m_combo2.SetCurSel(0);                            //选中第 1 项
}
```

ResetContent 函数清空组合框 2 的内容，GetCurSel 函数获取组合框 1 选中项的索引。根据组合框 1 的选中项，组合框 2 填充对应的元素，如选中第 1 项，组合框 2 填充数组中第 1 个空字符和第 2 个空字符之间的所有元素，依此类推。

nFlag 记录空字符""，即分隔标志的数目。利用 for 循环遍历数组所有元素，若当前元素为空字符，nFlag 加 1，并直接获取下一个元素。当 nFlag 等于 nSel 时，开始添加元素，当 nFlag 大于 nSel 时，停止添加，由于选中项索引 nSel 从 0 开始计数，将 nFlag 初值设为-1。SetCurSel 函数设置组合框 2 选中第一项。

（5）生成程序并运行，如图 4-39 所示。组合框 1 的选中项改变后，组合框 2 显示对应的分类，实现联动显示效果。

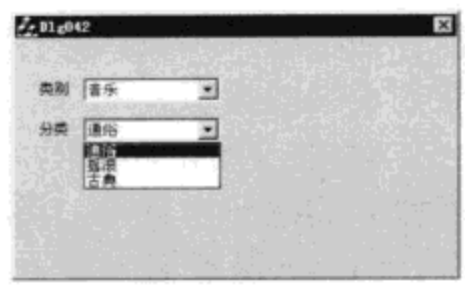


图 4-39　组合框联动

4.7.4　添加图像

若程序界面只有文字，显得有些单调，可在控件中添加图像，以增强美观性。扩展(Extended)组合框类 CComboBoxEx 支持图像列表，可为元素添加图标(Icon)或位图（Bitmap）。

（1）在资源视图展开 Dialog 节点，双击 IDD_DLG042_DIALOG 项，拖放扩展组合框控件到对话框模板中。按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_COMBOBOXEX1 项，添加 CComboBoxEx 类型的变量 m_comboEx1。

（2）在资源视图单击右键，在弹出的快捷菜单中选择 Insert 命令，弹出 Insert Resource 窗口，单击 Import 按钮，弹出 Import Resource 窗口，选择三个 ICO 图标文件，单击 Import 按钮导入图标文件，如图 4-40 所示。

（3）在 Icon 节点下，右键单击新添加的图标资源文件，在弹出的快捷菜单中选择 Properties 命令，弹出 Icon Properties 窗口，如图 4-41 所示。单击图钉按钮保持窗口显示，Preview 图片框可预览图标，ID 组合框依次设为 IDI_ICON_RED、IDI_ICON_GREEN、IDI_ICON_BLUE。

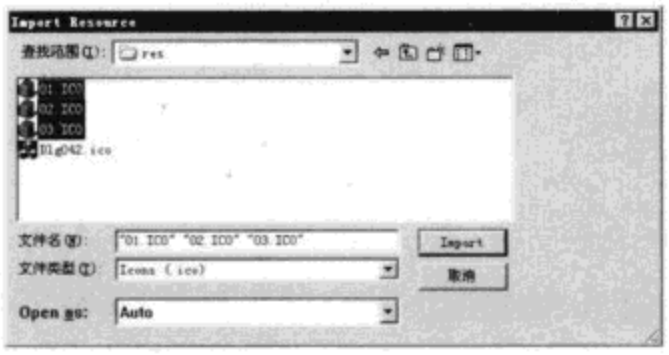


图 4-40　导入图标文件

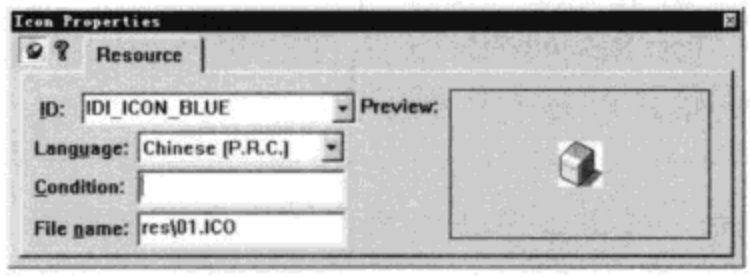


图 4-41　图标属性窗口

（4）在类视图右键单击 CDlg042Dlg 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加 CImageList 类型的变量 m_imgList，单击 OK 按钮保存并退出。

（5）在类视图双击 CDlg042Dlg 类下的 OnInitDialog 项，定位到函数，在 return TRUE; 前继续添加如下代码：

```
m_imgList.Create(32,32,TRUE,3,1);                          //创建图像列表,大小为 32*32
m_imgList.Add(AfxGetApp()->LoadIcon(IDI_ICON_RED));        //添加图标
m_imgList.Add(AfxGetApp()->LoadIcon(IDI_ICON_GREEN));
m_imgList.Add(AfxGetApp()->LoadIcon(IDI_ICON_BLUE));
m_comboEx1.SetImageList(&m_imgList);                       //设置控件使用的图像列表
```

```
COMBOBOXEXITEM comboItem;                                          //扩展组合框单项结构体
comboItem.mask=CBEIF_TEXT|CBEIF_IMAGE|CBEIF_SELECTEDIMAGE;         //设置结构体可用成员
LPSTR strText[3]={"红色","绿色","蓝色"};                            //元素文本
for(int j=0;j<3;j++)                                               //添加三个元素
{
  comboItem.iItem=j;                                              //该项在控件中的索引
  comboItem.iImage=j;                                             //该项图标的索引
  comboItem.iSelectedImage=j;                                     //该项被选中后图标的索引
  comboItem.pszText=strText[j];                                   //该项显示的文本
  m_comboEx1.InsertItem(&comboItem);                              //添加到扩展组合框中
}
```

CImageList 类用于存放一组相同大小的图像，可以存放图标和位图，控件和 CImageList 类结合使用显示图像。控件设置关联的 CImageList 类对象，并设置每个元素项所显示的图像在图像列表中的索引。

创建一个图像列表，需要先构造一个 CImageList 类对象，再调用 Create 函数创建图像列表，类似于创建一个窗口，要先构造一个 CWnd 或派生类对象，再调用 Create 函数创建一个窗口实例。Create 函数格式如下：

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

参数如下。

- cx：图像的宽度尺寸，以像素为单位，一般为 16 或 32。
- cy：图像的高度尺寸。
- nFlags：图像类型，若为图标可使用 TRUE，等同于 ILC_MASK。
- nInitial：图像的初始数目。
- nGrow：当需要增加存储空间时，一次增加的图像数目。

返回值：若成功返回非零值，否则返回 0。

图标作为一种资源，其句柄值（HICON）用来唯一指定一个图标。LoadIcon 函数用来载入资源文件中的图标资源，获取图标的句柄值，格式如下：

```
HICON CWinApp::LoadIcon(UINT nIDResource) const
HICON CWinApp::LoadIcon(LPCTSTR lpszResourceName) const
```

参数如下。

- nIDResource：图标资源的 ID。
- lpszResourceName：字符串格式的图标资源名称。

返回值：图标的句柄值。

Tips 若函数参数需要字符串格式的资源名称，使用 MAKEINTRESOURCE 函数可将整型的资源 ID 转换为字符串格式的名称。

AfxGetApp 函数获取当前程序的 CWinApp 类指针，App 类负责整个程序的启动和退出，一个程序只有一个 App 类，AfxGetApp 函数格式如下：

```
CWinApp* AfxGetApp()
```

返回值：当前程序的 CWinApp 类指针。

Tips 以 Afx 开头的函数都是全局函数，不属于任何类，如 AfxGetInstanceHandle 函数返回当前程序实例的句柄（HINSTANCE）。

创建图像列表后，Add 函数用于在图像列表中添加图像，格式如下：

```
int CImageList::Add(HICON hIcon)
```

参数如下。

- hIcon：图标的句柄。

返回值：新添加图标的索引。

图像列表创建完成后，SetImageList 函数设置扩展组合框所使用的图像列表，格式如下：

```
CImageList* CComboBoxEx::SetImageList(CImageList* pImageList)
```

参数如下。

- pImageList：CImageList 类对象指针。

返回值：先前使用的 CImageList 类对象指针，若无则返回 NULL。

COMBOBOXEXITEM 结构体包含扩展组合框单个项的所有信息，插入新项时传入该结构体的指针，格式如下：

```
typedef struct {
    UINT mask;                          //设置哪些成员可用
    INT_PTR iItem;                      //项在控件中的索引
    LPTSTR pszText;                     //显示文本
    int cchTextMax;                     //文本最大长度
    int iImage;                         //显示的图像在图像列表中的索引
    int iSelectedImage;                 //选中后显示的图像索引
    int iOverlay;
    int iIndent;
    LPARAM lParam;
} COMBOBOXEXITEM, *PCOMBOBOXEXITEM;
```

mask 设置结构体中哪些成员可用，如CBEIF_TEXT|CBEIF_IMAGE|CBEIF_ SELECTEDIMAGE 表示这三个成员的设置生效，其中 CBEIF_TEXT 的二进制格式为 00000001，CBEIF_IMAGE 为 00000010，CBEIF_SELECTEDIMAGE 为 00000100，位或运算符|执行位运算，只要两个数中有一个该位为 1，则结果为 1，函数根据该位是否为 1，决定是否启用该位所对应的成员。

InsertItem 函数用于在扩展组合框中插入新项，格式如下：

```
int CComboBoxEx::InsertItem(const COMBOBOXEXITEM* pCBItem)
```

参数如下。

- pCBItem：指向 COMBOBOXEXITEM 结构体的指针，const 限定该参数为只读。

返回值：若成功返回新项的下标，否则返回-1。

（6）生成程序并运行，如图 4-42 所示。在扩展组合框中添加三项，每项元素都使用一种图标。

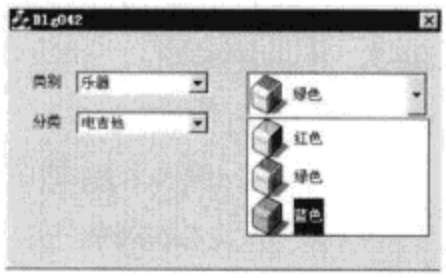


图 4-42　扩展组合框

4.8 列表框

组合框中弹出的下拉列表就是一个列表框控件，列表框（ListBox）可以一次显示、选择多项，当元素项数目超出显示范围时，自动出现垂直滚动条，滚动显示。项被选中后，以高亮状态显示，可根据选中项的值，实现相关操作。

4.8.1 设置属性

【实例 4-8】新建一个对话框工程名为 Dlg043，实现具有简易提示功能的编辑框，输入内容时弹出一个列表框，显示与输入内容匹配的项，当编辑框失去输入焦点或没有匹配项时自动隐藏列表框。

① 新建一个对话框工程名为 Dlg043，在资源视图双击 IDD_DLG043_DIALOG 项，拖放静态文本、编辑框、列表框控件到对话框模板中，如图 4-43 所示。

② 设置静态文本的 Caption 为“输入字符”，设置编辑框的 ID 为 IDC_EDIT_INPUT。右键单击列表框，在弹出的快捷菜单中选择 Properties 命令，弹出 List Box Properties 窗口，设置 ID 为 IDC_LIST_MAIN。切换到 Styles 选项卡，如图 4-44 所示。

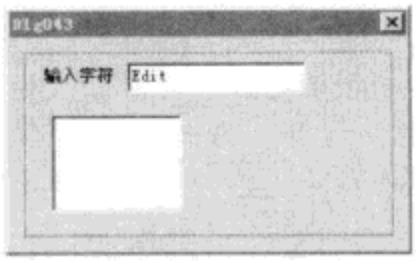


图 4-43 添加列表框控件

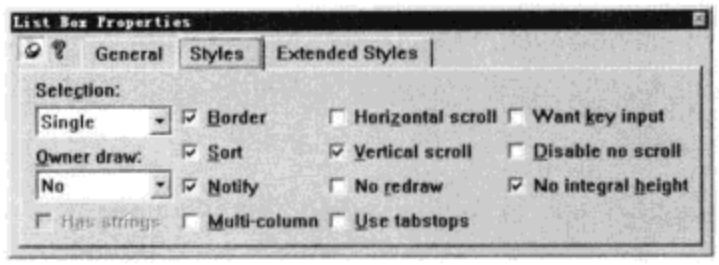


图 4-44 列表框 Styles 选项卡

Selection 组合框设置列表框的选择类型，其中 Single 表示单选，Multiple 表示多选。Border 复选框设置是否有边框，Sort 复选框设置是否自动排序。

③ 右键单击编辑框，在弹出的快捷菜单中选择 ClassWizard 命令，切换到 Member Variables 选项卡，双击 IDC_EDIT_INPUT 项，添加 CEdit 类型的变量 m_editInput，双击 IDC_LIST_MAIN 项，添加 CListBox 类型的变量 m_listMain，如图 4-45 所示。单击 OK 按钮保存并退出。

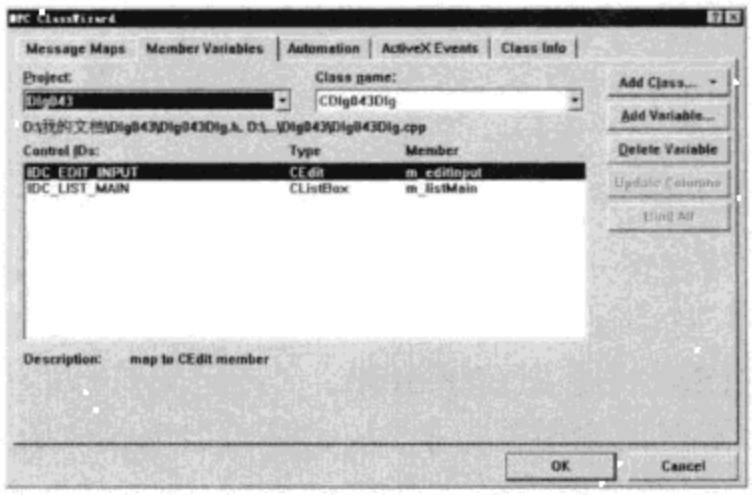


图 4-45 添加控件变量

4.8.2 编辑项

列表框对应 CListBox 类，操作函数类似于组合框，其中添加 AddString、删除 DeleteString、插入 InsertString、清空 ResetContent 函数可参考组合框中的介绍，不同之处在于列表框允许多选，相关函数如下：

（1）GetText 函数获取指定索引的字符串，格式如下：

```
void CListBox::GetText(int nIndex, CString& rString) const
```

参数如下。

- nIndex：指定项的索引。
- rString：CString 对象引用，存放字符串值。

（2）GetSelCount 函数获取多选状态下选择项数目，格式如下：

```
int CListBox::GetSelCount() const
```

返回值：选择项数目，若为单选列表框，则返回 LB_ERR。

（3）GetSelItems 函数获取选择项的索引集合，格式如下：

```
int CListBox::GetSelItems(int nMaxItems, LPINT rgIndex) const
```

参数如下。

- nMaxItems：索引集合的最大数目。
- rgIndex：指向整型数组的指针，存放索引集合。

返回值：实际存放的索引数目，若为单选列表框，则返回 LB_ERR。

4.8.3 消息响应

列表框在选择项改变时，触发 LBN_SELCHANGE 消息，可添加该消息的处理函数，当选择项改变时，自动调用该函数。若使用 SetCurSel 函数改变选择项，则不会触发该消息。

编辑框输入时触发 EN_CHANGE 消息，在该消息处理函数里显示列表框及匹配项。编辑框失去焦点时触发 EN_KILLFOCUS 消息，在该消息处理函数里隐藏列表框。

① 在类视图右键单击 CDlg043Dlg 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加 CStringArray 类型的变量 m_arrayStr。

② 双击 CDlg043Dlg 类下的 OnInitDialog 项，定位该函数，在 return TRUE; 前添加如下代码：

```
CString strArray[10]={"love","beyond","good","light","wonderful","program"
  ,"beautiful","guitar","geography","develop"};    //字符串数组
m_arrayStr.SetSize(10);                            //设置数组初始大小
for(int i=0;i<10;i++)                              //将字符串数组填充到 CStringArray 中
  m_arrayStr.SetAtGrow(i,strArray[i]);             //设置第 i 个元素的值
m_listMain.ShowWindow(FALSE);                      //隐藏列表框
```

SetSize 函数用于为动态数组一次分配固定大小的内存，若大致确定数组的大小，可使用 SetSize 一次分配足够内存，避免每次添加新元素都重新分配内存。

SetAtGrow 函数设置指定位置元素的值，若索引超出了范围，则会自动增加内存空间。ShowWindow 函数控制窗口的显示状态。利用 for 循环将字符串数组 strArray 的元素添加到动态数组 m_arrayStr 中。

> **Tips** 若使用 SetSize 函数分配内存空间后，应使用 SetAtGrow 函数设置每个元素的值，不可使用 Add 函数设置元素，Add 会在已分配的元素后重新添加新元素项，而不是从头开始设置元素的值。

③ 按 Ctrl+W 组合键打开类向导窗口，Object IDs 列表框选择 IDC_EDIT_INPUT 项，Messages 列表框选择 EN_KILLFOCUS 项，单击 Add Function 按钮添加消息处理函数，单击 Edit Code 按钮编辑函数，添加如下代码：

```
void CDlg043Dlg::OnKillfocusEditInput()
{
    m_listMain.ShowWindow(FALSE);                  //隐藏列表框
}
```

当编辑框失去输入焦点时，触发 EN_KILLFOCUS 消息，自动调用该函数，隐藏列表框。

④ 在类向导窗口中，选择 IDC_EDIT_INPUT 项，再添加 EN_CHANGE 消息处理函数，添加如下代码：

```
void CDlg043Dlg::OnChangeEditInput()
{
    CString strInput;
    m_editInput.GetWindowText(strInput);           //获取编辑框内容
    if(strInput.IsEmpty())                         //若编辑框为空，则隐藏列表框
    {
        m_listMain.ShowWindow(FALSE);
```

```
        return;
    }
    m_listMain.ResetContent();                                    //清空列表框
    for(int i=0;i<m_arrayStr.GetSize();i++)                       //遍历动态数组
    {
        CString strElement=m_arrayStr.GetAt(i);                   //获取第 i 个元素的值
        if(strElement.Find(strInput)!=-1)                         //若该项包含编辑框中的文本
            m_listMain.AddString(strElement);                     //将该项添加到列表框中
    }
    if(m_listMain.GetCount()>0)                                   //若列表框中元素数目大于 0
    {
        m_listMain.SetCurSel(0);                                  //选中第一项
        CRect rcEdit;
        m_editInput.GetWindowRect(rcEdit);                        //获取编辑框的屏幕坐标值
        ScreenToClient(rcEdit);                                   //将屏幕坐标转为客户区坐标
        m_listMain.MoveWindow(rcEdit.left,rcEdit.bottom+2,rcEdit.Width(),80);
                                                                  //将列表框移到编辑框下方
        m_listMain.ShowWindow(TRUE);                              //显示列表框
    }
    else                                                          //若没有匹配项，隐藏列表框
        m_listMain.ShowWindow(FALSE);
}
```

GetWindowText 函数获取编辑框中的内容，存放到 strInput 中。IsEmpty 函数判断字符串是否为空。若字符串为空，则调用 ShowWindow 函数隐藏列表框。

ResetContent 函数清空列表框已有项。利用 for 循环遍历数组 m_arrayStr 中的所有元素，若某个元素包含输入的字符串，则调用 AddString 函数将其添加到列表框中。

Find 函数用于搜索当前字符串是否包含子字符串，格式如下：

```
int CString::Find(LPCTSTR lpszSub) const
```

参数如下。

- lpszSub：子字符串。

返回值：若找到子字符串，则返回第一个字符的索引，否则返回-1。

GetCount 函数获取列表框元素数目。若数目大于 0，则调用 SetCurSel 函数选中第一项。GetWindowRect 函数获取窗口边界矩形的大小，格式如下：

```
void CWnd::GetWindowRect(LPRECT lpRect) const
```

参数如下。

- lpRect：CRect 类对象，用于存放窗口的矩形边界。

GetWindowRect 获取编辑框的边界矩形，存放到 rcEdit 中。该矩形边界相对于屏幕左上角而定，称为屏幕坐标（Screen），起点(0,0)位于屏幕左上角。若相对于父窗口的客户区的左上角而定，则称为客户区坐标（Client），起点(0,0)位于客户区左上角。

ScreenToClient 函数用于将屏幕坐标转换为客户区坐标，格式如下：

```
void CWnd::ScreenToClient(LPRECT lpRect) const
```

参数如下。

- lpRect：使用屏幕坐标的矩形边界，同时存放修改后的值。

> **Tips** 若将客户区坐标转换为屏幕坐标，使用 CWnd::ClientToScreen 函数。

MoveWindow 函数用于改变列表框的大小和位置，其中左边界与编辑框一致，上边界为编

- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

辑框底边界值加 2，宽度与编辑框一致，高度设为 80，用于将列表框移动到编辑框正下方，并显示列表框。若列表框匹配项数目为 0，调用 ShowWindow 传入 FALSE 隐藏列表框。

⑤ 生成程序并运行，如图 4-46 所示。当输入字符时，列表框自动显示匹配项，若没有匹配项或失去输入焦点，则隐藏列表框。

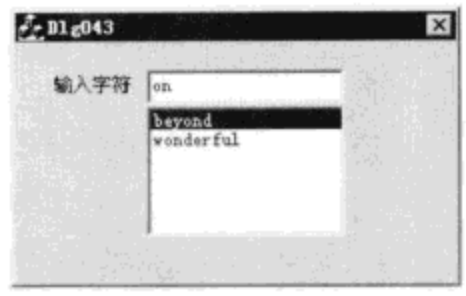


图 4-46　有提示功能的编辑框

4.9 进度条

进度条（progress）用来显示一个复杂冗长操作的当前进度。在执行一个费时操作时，用户通过进度条，可以知道需要多久才能执行完毕，从而提升用户体验效果，如 Windows 的文件传送框，显示当前传送进度，便于用户了解所需时间。

4.9.1 设置属性

【实例 4-9】新建一个对话框工程名为 Dlg044，在编辑框中输入倒计时秒数，单击“开始计时”按钮开始倒计时，进度条显示时间进度，进度完成后自动关闭程序。

① 新建一个对话框工程名为 Dlg044，在资源视图双击 IDD_DLG044_DIALOG 项，拖放静态文本、编辑框、按钮、进度条控件到对话框模板中，如图 4-47 所示。

② 设置静态控件的 Caption 为“倒计时(秒)”，设置编辑框的 ID 为 IDC_EDIT_ SECOND。设置按钮控件的 ID 为 IDC_BUTTON_BEGIN，Caption 为“开始计时”。

③ 右键单击进度条控件，在弹出的快捷菜单中选择 Properties 命令，弹出 Progress Properties 窗口，在 General 选项卡里设置 ID 为 IDC_PROGRESS_TIME。选择 Styles 选项卡，勾选 Smooth 复选框，如图 4-48 所示。

Border 复选框设置是否有边框，Vertical 复选框设置是否在垂直方向显示，Smooth 复选框设置是否平滑显示，默认为连续的方格。

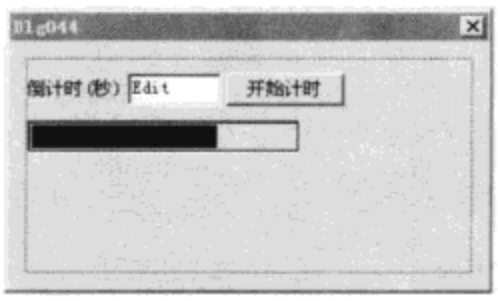


图 4-47　添加进度条控件

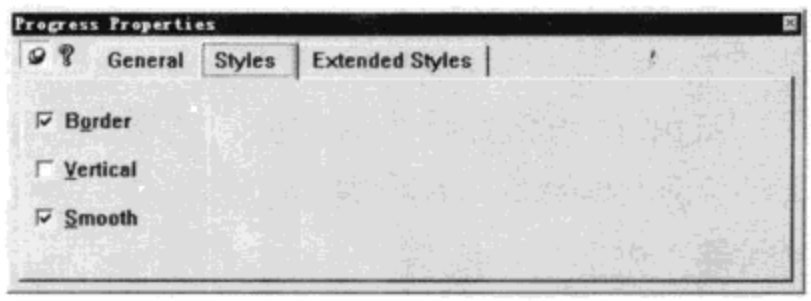


图 4-48　进度条 Styles 选项卡

④ 按 Ctrl+W 组合键，打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_EDIT_ SECOND 项，添加 int 类型的变量 m_nSecond，双击 IDC_PROGRESS_TIME 项，添加 CProgressCtrl 类型的变量 m_progCtrl，单击 OK 按钮保存并退出。

4.9.2 更新值

进度条对应 CProgressCtrl 类，有最大值、最小值及当前进度值，CProgressCtrl 类提供一系列函数操作进度条，相关函数如下：

（1）SetRange 函数设置最小值和最大值，格式如下：

```
void CProgressCtrl::SetRange(short nLower, short nUpper)
void CProgressCtrl::SetRange32(int nLower, int nUpper)
```

参数如下。

- nLower：最小值，默认为 0。

- nUpper：最大值，默认为 100。

（2）GetRange 函数获取进度条的范围，格式如下：

```
void CProgressCtrl::GetRange(int& nLower, int& nUpper)
```

参数如下。

- nLower：整型引用，存放最小值。
- nUpper：整型引用，存放最大值。

（3）SetPos 函数设置当前进度值，格式如下：

```
int CProgressCtrl::SetPos(int nPos)
```

参数如下。

- nPos：进度值，在最大值、最小值范围内。

返回值：先前的进度值。

（4）GetPos 函数获取当前进度值，格式如下：

```
int CProgressCtrl::GetPos()
```

返回值：当前进度值。

（5）OffsetPos 函数更新当前进度值，格式如下：

```
int CProgressCtrl::OffsetPos(int nPos)
```

参数如下。

- nPos：偏移量，在当前进度值的基础上变化。

返回值：先前的进度值。

（6）SetStep 函数设置单步增量，默认为 10，格式如下：

```
int CProgressCtrl::SetStep(int nStep)
```

参数如下。

- nStep：单步增量值，在 StepIt 函数中使用。

返回值：先前的单步增量。

（7）StepIt 函数根据单步增量值，递增当前进度值，格式如下：

```
int CProgressCtrl::StepIt()
```

返回值：先前的进度值。

要实现倒计时效果，可利用定时器计算时间，设定每隔 1 秒收到一次 WM_TIMER 消息，在消息处理函数中更新进度条的值。当进度条的值达到最大值时，调用 SendMessage 函数发送 WM_CLOSE 消息到当前程序，程序收到该消息后，自动调用相关函数关闭程序。

（1）在资源视图双击 IDD_DLG044_DIALOG 项，打开对话框模板，双击“开始计时”按钮，添加如下代码：

```
void CDlg044Dlg::OnButtonBegin()
{
    UpdateData(TRUE);                              //更新控件对应的变量的值
    if(m_nSecond==0)                               //若编辑框输入为 0，不做处理
        return;
    m_progCtrl.SetRange(0,m_nSecond);              //设置进度条范围
    m_progCtrl.SetStep(1);                         //设置单步增量
    SetTimer(1,1000,NULL);                         //启用时钟 1，每隔 1 秒发送一次 WM_TIMER 消息
    GetDlgItem(IDC_BUTTON_BEGIN)->EnableWindow(FALSE);          //设置按钮不可用
}
```

UpdateData 函数用来更新控件和变量的值，若参数为 TRUE 则更新变量，若为 FALSE 则更

新控件。SetRange 函数设置进度条的范围，最小值为 0，最大值为编辑框中输入的秒数。

SetStep 函数设置 StepIt 函数执行一次所增加的值，每隔 1 秒增加 1。SetTimer 函数启动时钟 1，程序每隔 1 秒收到一次 WM_TIMER 消息。EnableWindow 函数设置按钮不可用，避免多次单击造成混乱。

（2）在类视图右键单击 CDlg044Dlg 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，弹出 New Windows Message 窗口，选择 WM_TIMER 消息，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg044Dlg::OnTimer(UINT nIDEvent)
{
    CDialog::OnTimer(nIDEvent);
    m_progCtrl.StepIt();                          //单步递增
    if(m_progCtrl.GetPos()==m_nSecond)            //若进度条达到最大值
    {
        KillTimer(1);                             //销毁时钟 1
        SendMessage(WM_CLOSE);                    //发送 WM_CLOSE 消息到当前窗口
    }
}
```

程序每隔 1 秒收到一次 WM_TIMER 消息，自动调用该函数，其中 nIDEvent 为时钟的 ID 值。StepIt 函数执行单步递增，在 SetStep 函数中设置单步增量。

GetPos 函数获取当前进度值。若达到进度条的最大值，调用 KillTimer 函数销毁时钟 1，调用 SendMessage 函数发送 WM_CLOSE 消息到当前窗口，窗口收到消息后，调用默认消息处理函数关闭程序。

SendMessage 函数用于手动发送消息到窗口，格式如下：

```
LRESULT CWnd::SendMessage(UINT message,WPARAM wParam = 0,LPARAM lParam = 0)
```

参数如下。

- message：消息类型，如 WM_CLOSE。
- wParam：消息的附加参数，默认为 0。
- lParam：消息的附加参数。

返回值：消息处理结果。

Tips MFC 类库是对 Windows API 函数的浅层封装，将功能相关的函数封装到一个类中，其中很多函数都分为类成员版本和 API 版本，如 CWnd::SendMessage 为类成员版本，::SendMessage 为 API 版本，格式为 LRESULT SendMessage(HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam)；相对于类成员版本，多了一个 HWND 类型参数，而 CWnd 类对象本身已有 hWnd 值，无须在函数中再次传入。若使用 API 版本的函数，用::作用域限定符表明使用全局版本的函数。

（3）生成程序并运行，如图 4-49 所示。在编辑框中输入秒数，单击“开始计时”按钮后，进度条每隔 1 秒递增一次，同时按钮变灰不可用，当进度条达到最大值时，程序自动关闭。

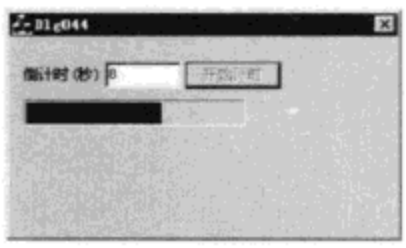


图 4-49　进度条倒计时

4.10 滑块

滑块（slider）控件用于滑动设置数值，由一个滑块和有刻度的矩形窗口组成，鼠标拖动滑块时，改变相关值，如 Windows Media Player 播放器的音量控制，拖动滑块改变音量大小。

4.10.1 设置属性

【实例 4-10】在工程 Dlg044 中添加滑块控件，通过拖动滑块改变编辑框的值。

① 打开工程 Dlg044，在资源视图中双击 IDD_DLG044_DIALOG 项，拖放滑块控件到对话框模板中，如图 4-50 所示。

② 右键单击滑块，在弹出的快捷菜单中选择 Properties 命令，弹出 Slider Properties 窗口，设置 ID 为 IDC_SLIDER_SECOND。选择 Styles 选项卡，如图 4-51 所示。

Orientation 组合框设置滑块水平还是垂直显示，Point 组合框设置滑块箭头指向，Tick marks 复选框设置是否显示标尺，Enable selection 复选框设置是否选择显示。

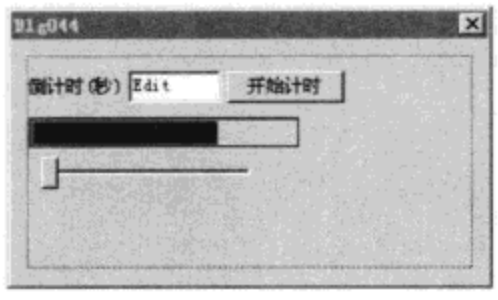


图 4-50 添加滑块控件

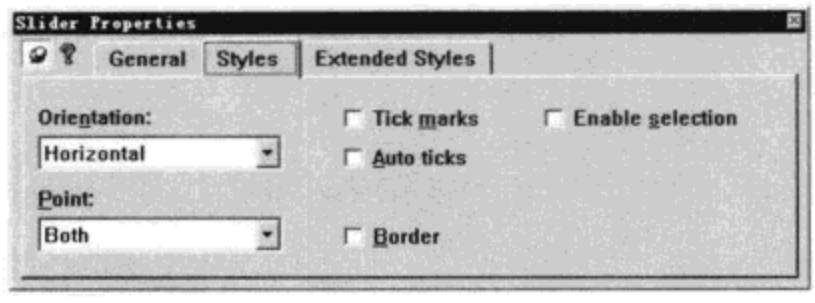


图 4-51 滑块 Styles 选项卡

③ 按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_SLIDER_SECOND 项，添加 CSliderCtrl 类型的变量 m_sliderCtrl，单击 OK 按钮保存并退出。

4.10.2 消息响应

不同于其他控件，滑块被拖动时，向父窗口发送 WM_HSCROLL 或 WM_VSCROLL 消息，在父窗口的滚动消息处理函数里执行相关操作。若滑块为水平显示，则发送 WM_HSCROLL 消息，若为垂直显示，则发送 WM_VSCROLL 消息。

① 在类视图双击 CDlg044Dlg 类下的 OnInitDialog 项，定位到函数，在 return TRUE; 前添加如下代码：

```
m_sliderCtrl.SetRange(0,30);                                    //设置滑块范围
```

SetRange 函数用于设置滑块的值范围，格式如下：

```
void CSliderCtrl::SetRange(int nMin,int nMax,BOOL bRedraw = FALSE)
```

参数如下。

- nMin：最小值。
- nMax：最大值。
- bRedraw：是否重绘，默认为 FALSE。

② 在类视图右键单击 CDlg044Dlg 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，选择 WM_HSCROLL 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg044Dlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    if(pScrollBar->GetDlgCtrlID()==IDC_SLIDER_SECOND)    //判断发出消息的控件是否为滑块
    {
        m_nSecond=m_sliderCtrl.GetPos();                 //获取滑块的值
        UpdateData(FALSE);                               //更新编辑框
    }
    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

当滑块水平滚动时，向父窗口发送 WM_HSCROLL 消息，在该消息处理函数中，判断是否

为滑块发出该消息。pScrollBar 参数包含发出滚动消息的控件的指针，调用 GetDlgCtrlID 函数获取控件的 ID 值，

若为滑块控件发出，则 GetPos 函数获取滑块的当前值，存放到编辑框的变量 m_nSecond 中，再调用 UpdateData 函数更新编辑框。

GetDlgCtrlID 函数获取窗口或控件的 ID 值，格式如下：

```
int CWnd::GetDlgCtrlID() const
```

返回值：CWnd 类对象所在的窗口或控件的 ID 值。

GetPos 函数获取滑块的当前值，格式如下：

```
int CSliderCtrl::GetPos() const
```

返回值：滑块的当前值。

③ 生成程序并运行，如图 4-52 所示。拖动滑块，编辑框中的值随之改变。

4.11 列表控件

列表视图有多种显示风格，如打开一个目录后，右键单击空白区弹出一个右键菜单，可以多种方式显示文件和文件夹，如图 4-53 所示。列表控件（ListControl）提供如下 4 种视图（View）。

- 图标视图（Icon）：显示全尺寸 32*32 图标，图标下方显示文本标签。
- 小图标视图（Small Icon）：显示 16*16 小图标，图标右边显示文本标签。
- 列表视图（List）：显示小图标，图标右边显示标签，按列排列，不能拖曳。
- 报表视图（Report）：一行显示一项，可以有多列附加信息，最左边的一列包括图标和标签。

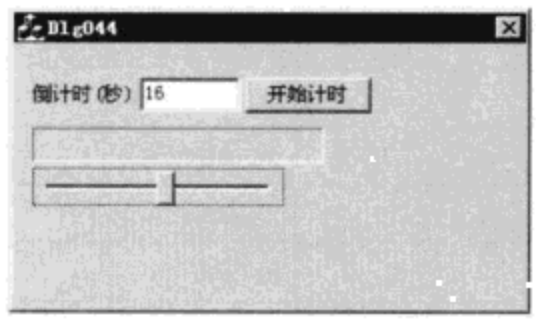


图 4-52　滑块更新编辑框值

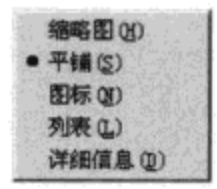


图 4-53　列表视图风格

4.11.1 设置属性

【实例 4-11】新建一个对话框工程名为 Dlg045，在列表控件中显示姓名、性别、年龄信息，并实现以下功能：

- 单击“添加”按钮，在列表控件中添加一项。
- 单击“更新”按钮，更新选中项的信息。
- 单击“删除”按钮，删除选中项。
- 列表控件选中一项后，下方的编辑框、组合框显示选中项的信息。

（1）新建一个对话框工程名为 Dlg045，在资源视图双击 IDD_DLG045_DIALOG 项，拖放三个静态文本、两个编辑框、三个按钮、一个组合框、一个列表控件到对话框模板中，如图 4-54 所示。

（2）设置三个静态文本的 Caption 依次为“姓名”、“性别”、“年龄”。

（3）设置“姓名”对应的编辑框的 ID 为 IDC_EDIT_NAME。设置“性别”对应的组合框的 ID 为 IDC_COMBO_SEX，在属性窗口的 Styles 选项卡里，Type 选择 Drop List 项，取消选中 Sort

复选框。设置“年龄”对应的编辑框的 ID 为 IDC_EDIT_AGE，在 Styles 选项卡里勾选 Number 复选框。

（4）设置三个按钮的 Caption 依次为“添加”、“更新”、“删除”，对应的 ID 依次为 IDC_BUTTON_ADD、IDC_BUTTON_UPDATE、IDC_BUTTON_DEL。

(5)右键单击列表控件，在弹出的快捷菜单中选择 Properties 命令，弹出 List Control Properties 窗口，设置 ID 为 IDC_LIST_PERSONS。选择 Styles 选项卡，如图 4-55 所示。

View 组合框设置视图风格，Sort 组合框设置是否排序，Single selection 复选框设置是否只能单选，Edit labels 复选框设置是否可以编辑项，Show selection always 设置是否持续显示选中项。

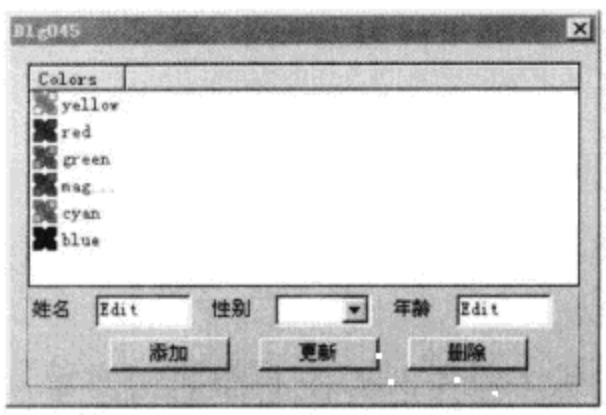


图 4-54　添加列表控件

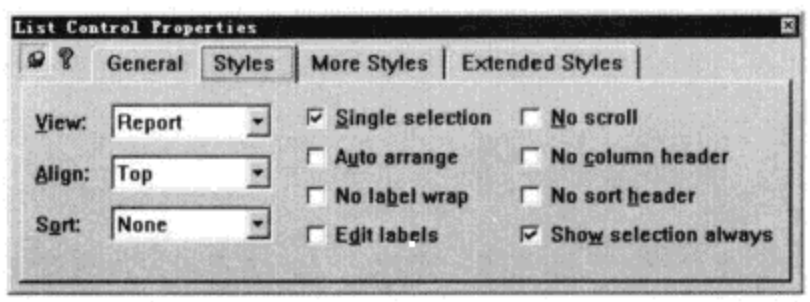


图 4-55　列表控件 Styles 选项卡

（6）View 组合框选择 Report 项，勾选 Single selection 和 Show selection always 复选框。

（7）按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，添加成员变量，如表 4-1 所示。单击 OK 按钮保存并退出。

表 4-1　控件映射变量

控件 ID	变量类型	变 量 名
IDC_COMBO_SEX	CComboBox	m_comboSex
IDC_EDIT_NAME	CString	m_strName
IDC_EDIT_AGE	int	m_nAge
IDC_LIST_PERSONS	CListCtrl	m_list

4.11.2　编辑项

列表控件对应 CListCtrl 类，该类提供一系列函数用于插入列、添加新项、删除项等操作，常用函数如下：

（1）InsertItem 函数用于插入新项，格式如下：

```
int CListCtrl::InsertItem(int nItem,LPCTSTR lpszItem)
```

参数如下。

- nItem：插入项在列表控件中的索引值。
- lpszItem：新项的字符串值。

返回值：新项的索引值。

（2）DeleteItem 函数用于删除一项，格式如下：

```
BOOL CListCtrl::DeleteItem(int nItem)
```

参数如下。

- nItem：删除项的索引。

返回值：若成功返回非零值，否则返回 0。

（3）DeleteAllItems 函数删除所有项，格式如下：

```
BOOL CListCtrl::DeleteAllItems()
```

返回值：若成功返回非零值，否则返回 0。

（4）InsertColumn 函数用于插入新列，格式如下：

```
int CListCtrl::InsertColumn(int nCol,LPCTSTR lpszColumnHeading,int nFormat = LVCFMT_
LEFT,int nWidth = -1,int nSubItem = -1)
```

参数如下。

- nCol：插入列的索引值。
- lpszColumnHeading：列的字符串名。
- nFormat：列内容对齐方式，默认为左对齐。
- nWidth：列的宽度。
- nSubItem：与该列关联的子项索引。

返回值：新列的索引值。

（5）DeleteColumn 函数用于删除一列，格式如下：

```
BOOL CListCtrl::DeleteColumn(int nCol)
```

参数如下。

- nCol：删除列的索引值。

返回值：若成功则返回非零值，否则返回 0。

（6）GetItemCount 函数获取项数目，格式如下：

```
int CListCtrl::GetItemCount()
```

返回值：所有项的数目。

（7）GetNextItem 函数获取与指定项有关系的项，格式如下：

```
int CListCtrl::GetNextItem(int nItem,int nFlags) const
```

参数如下。

- nItem：搜索项的索引，不搜索本项，若为-1 则从第一项开始搜索。
- nFlags：被搜索项与 nItem 指定项的关系，如 LVNI_SELECTED 表示指定项后的第一个选中项。

返回值：被搜索项的索引，若搜索失败则返回-1。

（8）GetItemText 函数获取某子项的文本值，格式如下：

```
CString CListCtrl::GetItemText(int nItem,int nSubItem) const
```

参数如下。

- nItem：项的索引值。
- nSubItem：子项的索引。

返回值：子项字符串文本。

（9）SetItemText 函数设置某子项的文本，格式如下：

```
BOOL CListCtrl::SetItemText(int nItem,int nSubItem,LPTSTR lpszText)
```

参数如下。

- nItem：项的索引值。
- nSubItem：子项的索引值。
- lpszText：子项的字符串文本。

返回值：若成功则返回非零值，否则返回 0。

（10）GetExtendedStyle 函数获取列表控件的已有扩展风格，格式如下：

```
DWORD CListCtrl::GetExtendedStyle()
```

返回值：已有的扩展风格组合。

（11）SetExtendedStyle 函数设置列表控件的扩展风格，格式如下：

```
DWORD CListCtrl::SetExtendedStyle(DWORD dwNewStyle)
```

参数如下。

- dwNewStyle：要使用的扩展风格组合。

返回值：先前的扩展风格组合。

4.11.3 消息响应

列表控件操作时会触发一些消息，如鼠标左键单击触发 NM_CLCIK 消息，右键单击触发 NM_RCLCIK 消息，添加相应的消息处理函数，执行特定操作。

（1）在类视图双击 CDlg045Dlg 类下的 OnInitDialog 项，定位该函数，在 return TRUE; 前添加如下代码：

```
m_comboSex.AddString("男");                                    //性别组合框添加项
m_comboSex.AddString("女");
m_comboSex.SetCurSel(0);                                       //选中第一项
m_list.InsertColumn(0,"姓名",LVCFMT_LEFT,90);                  //插入列
m_list.InsertColumn(1,"性别",LVCFMT_LEFT,90);
m_list.InsertColumn(2,"年龄",LVCFMT_LEFT,90);
m_list.SetExtendedStyle(m_list.GetExtendedStyle()|LVS_EX_FULLROWSELECT
|LVS_EX_GRIDLINES);                                            //设置扩展样式
```

AddString 函数在组合框中添加项，SetCurSel 设置选中第一项。InsertColumn 函数在列表控件中插入新列，如第 1 列名为“姓名”，左对齐，宽度为 90。

SetExtendedStyle 函数设置控件的扩展样式，GetExtendedStyle 函数获取已有的扩展样式，LVS_EX_FULLROWSELECT 扩展样式表示整行选中，LVS_EX_GRIDLINES 表示显示网格线，位或运算符|组合已有的和新添加的扩展样式。

（2）在资源视图双击 IDD_DLG045_DIALOG 项，打开对话框模板，双击“添加”按钮，添加如下代码：

```
void CDlg045Dlg::OnButtonAdd()
{
  UpdateData(TRUE);                                            //更新变量值
  if(m_strName.IsEmpty() || m_nAge==0)                         //若姓名为空或年龄为 0，直接返回
    return;
  CString strSex,strAge;
  m_comboSex.GetLBText(m_comboSex.GetCurSel(),strSex);         //获取性别选择项
  strAge.Format("%d",m_nAge);                                  //将整型年龄值格式化为字符串
  int nItem=m_list.InsertItem(m_list.GetItemCount(),m_strName);    //插入新项
  m_list.SetItemText(nItem,1,strSex);                                //设置子项值
  m_list.SetItemText(nItem,2,strAge);
}
```

UpdateData 函数更新控件映射的变量的值，若姓名为空或年龄为 0，直接返回。GetCurSel 函数获取组合框选中项的索引，GetLBText 函数根据选中项索引获取值，存入 strSex 中。Format 函数将整型年龄值格式化为字符串，存入 strAge 中。

GetItemCount 函数获取项数目，InsertItem 函数在尾部插入姓名项，返回插入项的索引。SetItemText 函数设置子项值，参数依次为项索引、子项的列索引、子项文本值。

Tips　在列表控件的 Report 视图下添加一项时，先调用 InsertItem 函数添加项的第 1 列的值，再调用 SetItemText 函数设置其他列的值。实际上只有第一列是列表控件的项，其余列作为该项的附加信息存在。

（3）右键单击列表控件，在弹出的快捷菜单中选择 Events 命令，选择 NM_CLICK 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg045Dlg::OnClickListPersons(NMHDR* pNMHDR, LRESULT* pResult)
{
    int nSel=m_list.GetNextItem(-1,LVNI_SELECTED);      //获取选中项索引
    if(nSel==-1)                                        //若没有选中项则返回
        return;
    m_strName=m_list.GetItemText(nSel,0);               //获取选中项第 1 列的值
    if(m_list.GetItemText(nSel,1)=="男")                //若为男，则组合框选择第一项
        m_comboSex.SetCurSel(0);
    else
        m_comboSex.SetCurSel(1);
    m_nAge=atoi(m_list.GetItemText(nSel,2));            //将第 2 列的年龄转为整型值
    UpdateData(FALSE);                                  //更新控件
    *pResult = 0;
}
```

鼠标左键单击列表控件时，触发 NM_CLICK 消息，自动调用该函数。GetNextItem 函数从头开始搜索选中项，返回选中项索引，若没有选中任何一项，直接返回。

GetItemText 函数获取选中项的对应列的值，第 1 列为姓名，第 2 列为性别，第 3 列为年龄。若性别为男，则 SetCurSel 函数设置组合框选中第一项，否则选中第二项。atoi 函数将字符串转换为整型值，UpdateData 函数根据变量值更新控件显示。

（4）打开对话框模板，双击“更新”按钮，添加如下代码：

```
void CDlg045Dlg::OnButtonUpdate()
{
  int nSel=m_list.GetNextItem(-1,LVNI_SELECTED);        //获取选中项索引
  if(nSel==-1)                                          //若没有选中项则返回
    return;
  UpdateData(TRUE);                                     //更新变量值
  if(m_strName.IsEmpty() || m_nAge==0)                  //若姓名为空或年龄为 0 则返回
    return;
  CString strSex,strAge;
  m_comboSex.GetLBText(m_comboSex.GetCurSel(),strSex);      //获取选择的性别
  strAge.Format("%d",m_nAge);                           //将整型年龄格式化为字符串格式
  m_list.SetItemText(nSel,0,m_strName);                 //更新选中项各列的值
  m_list.SetItemText(nSel,1,strSex);
  m_list.SetItemText(nSel,2,strAge);
}
```

GetNextItem 函数从第一项开始搜索选中项，返回选中项索引，存放到 nSel 中，若没有选中项则直接返回。若修改后的姓名为空，或年龄值为 0，则直接返回。用修改后的姓名、性别、年龄更新选中项的值。

（5）打开对话框模板，双击“删除”按钮，添加如下代码：

```
void CDlg045Dlg::OnButtonDel()
{
    int nSel=m_list.GetNextItem(-1,LVNI_SELECTED);          //获取选中项索引
    if(nSel==-1)
```

```
            return;
        m_list.DeleteItem(nSel);                                    //删除选中项
    }
```

GetNextItem 函数获取选中项索引，存放到 nSel 中，若没有选中项，则直接返回。DeleteItem 函数删除选中项。

（6）生成程序并运行，如图 4-56 所示。设置姓名、性别、年龄后，单击“添加”按钮添加一项。鼠标在列表控件选择一项后，下方三个控件同步显示选中项的值。修改控件中的值后，单击“更新”按钮，更新选中项的值。单击“删除”按钮移除选中项。

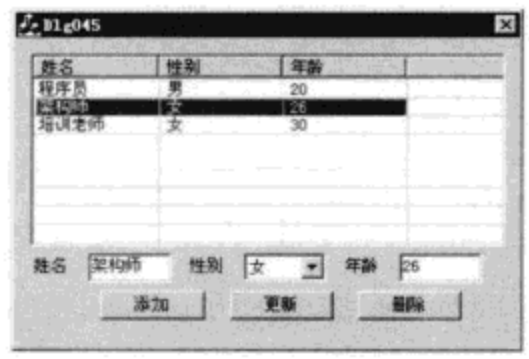


图 4-56 列表控件添加删除项

4.11.4 添加图像

在列表控件中添加图像，要先创建一个图像列表，为列表控件提供图像资源。创建图像列表，先调用 CImageList 类的构造函数创建一个类对象，再调用 Create 函数创建图像列表实例，设置图像的尺寸和颜色类型，调用 Add 函数添加位图或图标，具体代码参见组合框。图像列表创建完成后，调用 CListCtrl 类的 SetImageList 函数设置控件使用的图像列表。

在添加新项时指定该项的图像索引，调用 InsertItem 函数的重载版本，格式如下：

```
int CListCtrl ::InsertItem(int nItem, LPCTSTR lpszItem,int nImage)
```

参数如下。

- nItem：插入项的索引。
- lpszItem：项字符串值。
- nImage：使用图像的索引。

返回值：插入项的索引。

Tips 有些类成员函数有多个重载版本，介绍时仅列出常用的一个版本，要了解更多信息，请查阅 MSDN。

4.12 树控件

树（Tree）控件用来显示有层次关系的元素项，最顶部的项为根项（Root），同一级别的项为兄弟关系（Sibling），两个相邻级别中高级别的称为父项（Parent），低级别的称为子项（Child），类似书籍的目录结构，章、节、小节构成不同的层次关系。

4.12.1 设置属性

【实例 4-12】新建一个对话框工程名为 Dlg046，在树控件中显示添加的不同类别的文件，并实现以下功能：

- 单击“添加文件”按钮，在选中节点所在的类别下添加一项。
- 单击“重命名”按钮，修改选中的子节点的文本。
- 单击“删除文件”按钮，删除选中的子节点。
- 单击“已选择文件”按钮，显示所有勾选的子节点。
- 选择不同的子节点，在编辑框中显示节点名称。
- 根节点勾选状态改变后，所有子节点同步更新为相同的勾选状态。
- 直接在树控件中修改子节点的值。

（1）新建一个对话框工程名为 Dlg046，在资源视图展开 Dialog 节点，双击 IDD_DLG046_DIALOG 项，拖放静态文本、编辑框、4 个按钮、列表框、树控件到对话框模板中，如图 4-57 所示。

（2）设置静态文本的 Caption 为“当前文件名”，设置编辑框的 ID 为 IDC_EDIT_NAME，设置列表框的 ID 为 IDC_LIST_SELECT。

（3）设置 4 个按钮的 Caption 依次为“添加文件”、“ 重命名”、“ 删除文件”、“ 已选择文件”，ID 依次为 IDC_BUTTON_ADD、IDC_BUTTON_RENAME、IDC_BUTTON_DEL、IDC_BUTTON_SELECTED。

（4）右键单击树控件，在弹出的快捷菜单中选择 Properties 命令，弹出 Tree Control Properties 窗口，设置 ID 为 IDC_TREE_FILE。选择 Styles 选项卡，如图 4-58 所示。

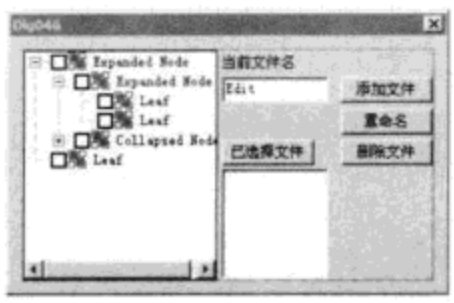


图 4-57　添加树控件

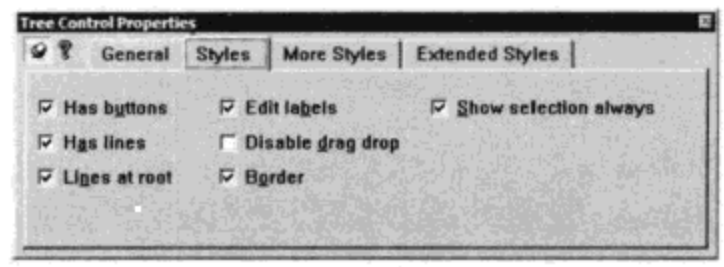


图 4-58　树控件的 Styles 选项卡

Has buttons 复选框设置是否有加减按钮，Has Lines 设置是否显示连接线，Lines at root 设置是否在根部显示连接线，Edit labels 设置是否可编辑节点，Disable drag drop 设置是否禁止拖放操作，Border 设置是否有边框，Show selection always 设置是否保持选中。

（5）勾选 Has buttons、Has Lines、Lines at root、Edit labels、Border、Show selection always 复选框。切换到 More Styles 选项卡，如图 4-59 所示。

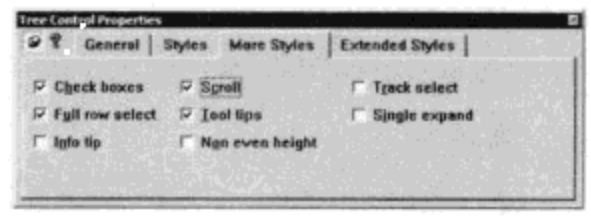


图 4-59　树控件的 More Styles 选项卡

Check boxes 设置是否有复选框，Full row select 设置是否整行选中，Scroll 设置是否有滚动条，Tool tips 设置是否显示提示信息，Single expand 设置是否只能展开一个节点。

（6）在 More Styles 选项卡中勾选 Check boxes、Full row select 复选框。

（7）按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_EDIT_ NAME 项添加 CString 类型的变量 m_strName，双击 IDC_LIST_SELECT 项添加 CListBox 类型的变量 m_list，双击 IDC_TREE_FILE 项添加 CTreeCtrl 类型的变量 m_tree，单击 OK 按钮保存并退出。

4.12.2　编辑项

树控件的项之间具有层次关系，用 HTREEITEM 句柄指定每个节点，添加节点时要指定父节点的句柄，若不指定父节点句柄，则作为根节点添加。树控件对应 CTreeCtrl 类，该类提供一系列函数用于添加、删除、更新节点，常用函数如下：

（1）InsertItem 函数用于添加新项，格式如下：

```
HTREEITEM CTreeCtrl::InsertItem(LPCTSTR lpszItem,int nImage,int nSelectedImage,
HTREEITEM hParent = TVI_ROOT,HTREEITEM hInsertAfter = TVI_LAST)
```

参数如下。

- lpszItem：项字符串文本。
- nImage：项的图像索引。
- nSelectedImage：项被选择后的图像索引。
- hParent：父节点的句柄，默认为根节点。

- hInsertAfter：新项之前的项的句柄值，默认为尾项。

返回值：新插入项的句柄。

（2）DeleteItem 函数用于删除指定项，格式如下：

```
BOOL CTreeCtrl::DeleteItem(HTREEITEM hItem)
```

参数如下。

- hItem：删除项的句柄。

返回值：若成功则返回非零值，否则返回 0。

（3）DeleteAllItems 函数删除所有项，格式如下：

```
BOOL CTreeCtrl::DeleteAllItems( )
```

返回值：若成功则返回非零值，否则返回 0。

（4）Expand 函数展开或收起子项列表，格式如下：

```
BOOL CTreeCtrl::Expand(HTREEETEM hItem, UINT nCode)
```

参数如下。

- hItem：要展开或收起列表的父项句柄。
- nCode：动作标志，如 TVE_EXPAND 为展开列表，TVE_COLLAPSE 为收起列表。

返回值：若成功返回非零值，否则返回 0。

（5）GetCount 函数获取项的数目，格式如下：

```
UINT CTreeCtrl::GetCount()
```

返回值：项数目。

（6）SetImageList 函数设置控件关联的图像列表，格式如下：

```
CImageList* CTreeCtrl::SetImageList(CImageList* pImageList, int nImageListType)
```

参数如下。

- pImageList：关联的图像列表的指针。
- nImageListType：图像列表的类型，如 TVSIL_NORMAL 包含选择和为选择的图像。

返回值：先前的图像列表的指针。

（7）GetNextItem 函数获取与参数项有关系的项，格式如下：

```
HTREEITEM CTreeCtrl::GetNextItem(HTREEITEM hItem, UINT nCode)
```

参数如下。

- hItem：搜索的起始项。
- nCode：与起始项的关系标志，如 TVGN_CHILD 为第一个子项，TVGN_NEXT 为下一个兄弟项。

返回值：被搜索项的句柄，否则返回 NULL。

（8）ItemHasChildren 函数判断一个项是否有子项，格式如下：

```
BOOL CTreeCtrl::ItemHasChildren(HTREEITEM hItem)
```

参数如下。

- hItem：判断项的句柄。

返回值：若有子项则返回非零值，否则返回 0。

（9）GetChildItem 函数获取参数项的子项句柄，格式如下：

```
HTREEITEM CTreeCtrl::GetChildItem(HTREEITEM hItem)
```

参数如下。

- hItem：父项句柄。

返回值：子项句柄，否则返回 NULL。

（10）GetNextSiblingItem 获取下一个兄弟项的句柄，格式如下：

```
HTREEITEM CTreeCtrl::GetNextSiblingItem(HTREEITM hItem)
```

参数如下。

- hItem：当前项句柄。

返回值：下一个兄弟项的句柄。

（11）GetParentItem 函数获取指定项的父项句柄，格式如下：

```
HTREEITEM CTreeCtrl::GetParentItem(HTREEITEM hItem)
```

参数如下。

- hItem：指定项句柄。

返回值：父项的句柄值，否则返回 NULL。

（12）GetSelectedItem 函数获取选中项的句柄，格式如下：

```
HTREEITEM CTreeCtrl::GetSelectedItem()
```

返回值：选中项的句柄。

（13）GetRootItem 获取根项的句柄，格式如下：

```
HTREEITEM CTreeCtrl::GetRootItem()
```

返回值：根项的句柄。

（14）GetItemText 函数获取指定项的文本，格式如下：

```
CString CTreeCtrl::GetItemText(HTREEITEM hItem) const
```

参数如下。

- hItem：指定项句柄。

返回值：项的文本。

（15）SetItemText 函数设置指定项的文本，格式如下：

```
BOOL CTreeCtrl::SetItemText(HTREEITEM hItem, LPCTSTR lpszItem)
```

参数如下。

- hItem：项句柄。
- lpszItem：新文本字符串。

返回值：若成功则返回非零值，否则返回 0。

（16）GetCheck 函数获取项的勾选状态，格式如下：

```
BOOL CTreeCtrl::GetCheck(HTREEITEM hItem) const
```

参数如下。

- hItem：项句柄。

返回值：若勾选则返回非零值，否则返回 0。

4.12.3　消息响应

树控件操作时会触发一些消息，如鼠标左键单击触发 NM_CLICK 消息，右键单击触发 NM_RCLICK 消息，选中项改变触发 TVN_SELCHANGED 消息，子项列表被展开或收起触发 TVN_ITEMEXPANDED 消息，开始编辑节点文本触发 TVN_BEGINLABELEDIT 消息，结束节点编辑触发 TVN_ENDLABELEDIT 消息，添加对应的消息处理函数，实现特定功能。

（1）在资源视图右键单击 Icon 项，在弹出的快捷菜单中选择 Import 命令，弹出 Import Resource 窗口，选择 res 文件夹下的 4 个 ICO 图标，单击 Import 按钮导入图标文件，如图 4-60 所示。

（2）右键单击添加的 ICO 图标资源，在弹出的快捷菜单中选择 Properties 命令，弹出 Icon Properties 窗口，设置 4 个图标资源的 ID 依次为 IDI_ICON1、IDI_ICON2、IDI_ICON3、IDI_ICON4。

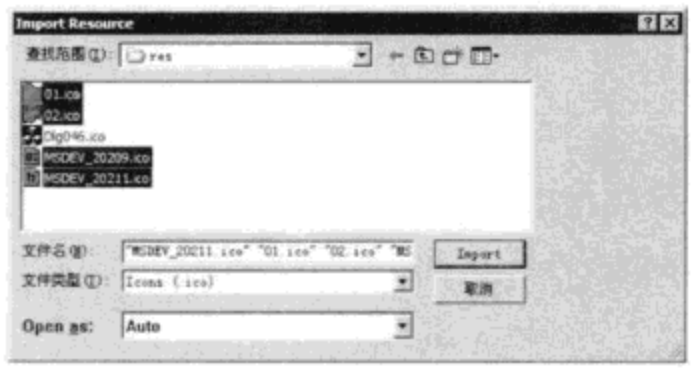


图 4-60　导入 ICO 文件

（3）在类视图双击 CDlg046Dlg 项，在类定义中添加如下成员变量：

```
public:
    CImageList m_img;                                    //图像列表
    HTREEITEM hSource;                                   //三个根节点的句柄
    HTREEITEM hHeader;
    HTREEITEM hResource;
```

CImageList 类对象加载图标资源，为树控件提供图像。HTREEITEM 存放三个根节点的句柄值，根据该值可以定位到根节点。

（4）在类视图双击 CDlg046Dlg 类下的 OnInitDialog 项，定位到函数，在 return TRUE; 前添加如下代码：

```
m_img.Create(16,16,ILC_MASK|ILC_COLOR32,4,1);        //创建图形列表，大小为 16*16
m_img.Add(AfxGetApp()->LoadIcon(IDI_ICON1));         //加载图标到图像列表中
m_img.Add(AfxGetApp()->LoadIcon(IDI_ICON2));
m_img.Add(AfxGetApp()->LoadIcon(IDI_ICON3));
m_img.Add(AfxGetApp()->LoadIcon(IDI_ICON4));
m_tree.SetImageList(&m_img,TVSIL_NORMAL);            //设置树控件关联的图像列表
hSource=m_tree.InsertItem("源文件",0,0);             //插入三个根节点，句柄值存入类变量中
hHeader=m_tree.InsertItem("头文件",0,0);
hResource=m_tree.InsertItem("资源文件",0,0);
```

Create 函数创建一个图像尺寸为 16*16、初始数目为 4、图像类型为 ILC_MASK|ILC_COLOR32 的图像列表，其中 ILC_COLOR32 表示使用 32 位颜色，ILC_MASK 表示使用掩码，掩码操作有两个图，一个为原始图，一个为掩码图，原始图根据掩码图的值去除部分区域，得到掩码后的图。

LoadIcon 函数获取图标资源的句柄，Add 函数将图标添加到图像列表中，SetImageList 函数设置树控件关联的图像列表。InsertItem 函数插入新项，参数依次为项文本、图标索引、选中后的图标索引，若不设置父节点句柄，则作为根节点添加。在树控件中添加三个根节点，用于表示三种文件类别，每个根节点下显示对应类别的文件。

（5）在资源视图双击 IDD_DLG046_DIALOG 项，打开对话框模板，双击“添加文件”按钮，添加如下代码：

```
void CDlg046Dlg::OnButtonAdd()
{
    UpdateData(TRUE);                                    //更新变量值
    HTREEITEM hSel=m_tree.GetSelectedItem();             //获取选中节点的句柄
    if(hSel==NULL || m_strName.IsEmpty())                //若没有选中项，或文件名为空，则直接返回
        return;
```

```
    if(hSel!=hSource && hSel!=hHeader && hSel!=hResource) //若选中项不是根节点
        hSel=m_tree.GetParentItem(hSel);                  //设为选中项的父节点
    int nIcon=0;                                          //图标索引
    if(hSel==hSource)                                     //若为源文件节点，则图标索引为 1
        nIcon=1;
    else if(hSel==hHeader)                                //若为头文件节点，则索引为 2
        nIcon=2;
    else if(hSel==hResource)                              //若为资源文件节点，则索引为 3
        nIcon=3;
    m_tree.InsertItem(m_strName,nIcon,nIcon,hSel);        //插入新项
    m_tree.Expand(hSel,TVE_EXPAND);                       //展开父节点
}
```

在树控件中选择要添加的文件类别的根节点或子节点，在编辑框中输入文件名，单击“添加文件”按钮，在节点所在类别下添加一项。

UpdateData 函数更新编辑框映射变量的值，GetSelectedItem 函数获取选中节点的句柄值，存放到 hSel 中，若没有选中节点或编辑框输入为空，则直接返回。

若选中节点不是根节点，则 GetParentItem 函数获取选中节点的根节点句柄，存放到 hSel 中。根据节点类型，使用不同的图标，调用 InsertItem 函数在 hSel 节点下插入一项，Expand 函数展开 hSel 节点下的子节点列表。

（6）右键单击树控件，在弹出的快捷菜单中选择 Events 命令，选择 TVN_SELCHANGED 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg046Dlg::OnSelchangedTreeFile(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
    HTREEITEM hSel=m_tree.GetSelectedItem();                    //选中节点的句柄值
    if(hSel!=hSource && hSel!=hHeader && hSel!=hResource)       //若不是根节点
        m_strName=m_tree.GetItemText(m_tree.GetSelectedItem()); //获取节点的文本值
    else
        m_strName="";                                           //若为根节点，则编辑框变量为空
    UpdateData(FALSE);                                          //更新编辑框
    *pResult = 0;
}
```

当树控件的选中节点改变后，触发 TVN_SELCHANGED 消息，自动调用该函数。GetSelectedItem 函数获取选中节点的句柄值，若不是根节点，则调用 GetItemText 函数获取节点的文本值，显示在编辑框中。

（7）打开对话框模板，双击“重命名”按钮，添加如下代码：

```
void CDlg046Dlg::OnButtonRename()
{
    HTREEITEM hSel=m_tree.GetSelectedItem();                    //选中节点的句柄值
    if(hSel==NULL)
        return;
    if(hSel!=hSource && hSel!=hHeader && hSel!=hResource) //若不是根节点
    {
        UpdateData(TRUE);                                       //更新变量值
        m_tree.SetItemText(hSel,m_strName);                     //用变量值更新节点
    }
}
```

选中节点后，编辑框同步显示节点的文本值，在编辑框中修改该值后，单击“重命名”按钮更新选中节点。若选中节点不是根节点，则 UpdateData 函数更新编辑框映射变量的值，SetItemText 函数更新选中节点的文本值。

（8）打开对话框模板，双击“删除文件”按钮，添加如下代码：

```
void CDlg046Dlg::OnButtonDel()
{
    HTREEITEM hSel=m_tree.GetSelectedItem();                          //获取选中节点的句柄值
    if(hSel==NULL)
        return;
    if(hSel!=hSource && hSel!=hHeader && hSel!=hResource)             //若不是根节点
        m_tree.DeleteItem(hSel);                                      //删除节点
}
```

选中一个节点，单击“删除文件”按钮，若不是根节点，则删除该节点。

（9）打开对话框模板，双击“已选择文件”按钮，添加如下代码：

```
void CDlg046Dlg::OnButtonSelected()
{
    m_list.ResetContent();                                            //清空列表框
    HTREEITEM hRoot=m_tree.GetRootItem();                             //根节点的句柄值
    while(hRoot!=NULL)                                                //遍历所有根节点
    {
        HTREEITEM hChild=m_tree.GetChildItem(hRoot);                  //第一个子节点的句柄值
        while(hChild!=NULL)                                           //遍历当前根节点下的所有子节点
        {
            if(m_tree.GetCheck(hChild))                               //若子节点已勾选
                m_list.AddString(m_tree.GetItemText(hChild));
                                                                      //将节点的文本值添加到列表框中
            hChild=m_tree.GetNextSiblingItem(hChild);//获取当前子节点的下一个兄弟节点
        }
        hRoot=m_tree.GetNextSiblingItem(hRoot);                       //获取当前根节点的下一个兄弟节点
    }
}
```

ResetContent 函数清空列表框已有项，GetRootItem 函数获取树控件的第一个根节点，存放到 hRoot 中。利用外层 while 循环遍历所有根节点，GetChildItem 函数获取第一个子节点的句柄值，利用内层 while 循环遍历当前根节点下的所有子节点。

GetCheck 函数获取节点的勾选状态，若已勾选，调用 AddString 函数将节点的文本值添加到列表框中。GetNextSiblingItem 函数获取当前节点的下一个兄弟项，如传入 hChild 获取下一个子节点，传入 hRoot 获取下一个根节点。

（10）右键单击树控件，在弹出的快捷菜单中选择 Events 命令，选择 TVN_ ENDLABELEDIT 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg046Dlg::OnEndlabeleditTreeFile(NMHDR* pNMHDR, LRESULT* pResult)
{
    TV_DISPINFO* pTVDispInfo = (TV_DISPINFO*)pNMHDR;
    HTREEITEM hItem=pTVDispInfo->item.hItem;                          //获取当前编辑节点的句柄值
    CString strInput=pTVDispInfo->item.pszText;                       //获取输入文本
    if(hItem!=hSource && hItem!=hHeader && hItem!=hResource && !strInput.IsEmpty())
        m_tree.SetItemText(hItem,strInput); //若不是根节点，且输入文本不为空，保存编辑结果
    *pResult = 0;
}
```

在树控件间断单击节点两次，节点进入编辑状态，当节点编辑结束时，触发 TVN_ENDLABELEDIT 消息，自动调用该函数，树控件不会自动保存编辑结果，可在该函数中添加保存功能。

TV_DISPINFO 结构体中存放编辑节点的各种信息，格式如下：

```
typedef struct tagNMTVDISPINFO {
    NMHDR hdr;                          //NMHDR 结构，存放通知消息
    TVITEM item;                        //编辑节点
} NMTVDISPINFO, *LPNMTVDISPINFO;
```

TVITEM 结构体存放树控件一个节点的所有信息，格式如下：

```
typedef struct tagTVITEM {
    UINT mask;                          //位标志，指定以下哪些成员可用
    HTREEITEM hItem;                    //节点的句柄值
    UINT state;                         //节点状态
    UINT stateMask;                     //state 的可用位
    LPTSTR pszText;                     //节点文本
    int cchTextMax;                     // pszText 指向的字符串长度
    int iImage;                         //图像索引
    int iSelectedImage;                 //节点选中后的图像索引
    int cChildren;                      //是否有子节点
    LPARAM lParam;                      //附加信息
} TVITEM, *LPTVITEM;
```

通过 TV_DISPINFO 结构体的 item 成员，可获取编辑节点。通过 TVITEM 结构体的 hItem 成员可获取编辑节点的句柄，存放到 hItem 中，pszText 成员可获取输入文本，存放到 strInput 中。若编辑节点不是根节点，且输入文本不为空，则调用 SetItemText 函数用输入文本替代原有值。

（11）右键单击树控件，在弹出的快捷菜单中选择 Events 命令，选择 NM_CLICK 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg046Dlg::OnClickTreeFile(NMHDR* pNMHDR, LRESULT* pResult)
{
    CPoint pt;
    UINT flag;
    ::GetCursorPos(&pt);                                    //当前鼠标的坐标位置
    ::ScreenToClient(m_tree.m_hWnd,&pt);                    //将屏幕坐标转换为客户坐标
    HTREEITEM hItem=m_tree.HitTest(pt,&flag);               //根据鼠标坐标获取单击节点的句柄
    if(hItem!=NULL && (flag&TVHT_ONITEMSTATEICON))          //若句柄不为空，且在复选框上单击
    {
        if(m_tree.ItemHasChildren(hItem))                   //若有子节点
        {
            HTREEITEM hChild=m_tree.GetChildItem(hItem);//获取第一个子节点的句柄
            while(hChild)                                    //遍历所有子节点
            {
                BOOL bCheck=m_tree.GetCheck(hItem);          //获取父节点的勾选状态
                m_tree.SetCheck(hChild,!bCheck);             //设置子节点同样的勾选状态
                hChild=m_tree.GetNextSiblingItem(hChild);//获取下一个子节点
            }
        }
    }
    *pResult = 0;
}
```

父节点的勾选状态改变后，所有子节点同步更新为相同的勾选状态。GetCursorPos 函数获取当前光标的坐标位置，相对于屏幕左上角，格式如下：

```
BOOL GetCursorPos(LPPOINT lpPoint)
```

参数如下。

- lpPoint：点坐标对象，存放坐标值。

返回值：若成功则返回非零值，否则返回 0。

ScreenToClient 函数将屏幕坐标值转换为树控件的客户区坐标值，转换后的坐标起点位于树控件客户区的左上角，其中 m_tree.m_hWnd 为树控件的句柄值，由于使用 API 版本的 ScreenToClient 函数，要传入窗口句柄参数。

HitTest 函数根据点坐标，获取该点所属节点的句柄值，格式如下：

```
HTREEITEM CTreeCtrl::HitTest(CPoint pt, UINT* pFlags)
```

参数如下。

- pt：点坐标值。
- pFlags：位标志，存放点坐标对应的节点具体对象，如复选框 TVHT_ONITEMSTATEICON、图标 TVHT_ONITEMICON、文本 TVHT_ONITEMLABEL 等。

返回值：点坐标所属的节点的句柄值，否则返回 NULL。

flag&TVHT_ONITEMSTATEICON 用于判断是否在复选框上单击，若不是在复选框上单击，该值为 0。若获取的节点句柄不为空，且在复选框上单击，ItemHasChildren 函数判断是否有子节点。

若有子节点，GetChildItem 函数获取第一个子节点，利用 while 循环遍历所有子节点，GetCheck 函数获取节点的勾选状态，并设置子节点为相同的勾选状态，GetNextSiblingItem 函数获取当前子节点的下一个兄弟项，即下一个子节点。

图 4-61　树控件编辑

（12）生成程序并运行，如图 4-61 所示。在左侧树控件中选择要添加的文件类别，在“当前文件名”编辑框中输入文件名称，单击“添加文件”按钮添加一项。

选择树控件的一项子节点，“当前文件名”编辑框显示节点的文本值。修改后，单击“重命名”按钮更新节点的文本值，也可直接在树控件上编辑节点。单击“删除文件”按钮删除选中的子节点项。

修改根节点项的勾选状态，则根节点下的所有子节点同步更新勾选状态。单击“已选择文件”按钮，在列表框中显示所有勾选的子节点项。

4.13 日期控件

日期控件可以用来选择和查看日期时间，它提供了统一的日期选择和读取方式，用户通过日期控件可以像查看日历一样，查看当前日期属于星期几，每个月的起始和结束日期等，无须手动输入年、月、日信息，方便用户操作。

4.13.1 设置属性

【实例 4-13】新建一个对话框工程名为 Dlg047，使用日期控件显示日期，当日期控件的选择日期改变后，对话框标题栏显示当前选择的日期。

（1）新建一个对话框工程名为 Dlg047，在资源视图展开 Dialog 节点，双击 IDD_DLG047_DIALOG 项，拖放静态文本、日期控件到对话框模板中，如图 4-62 所示。

（2）设置静态文本的 Caption 为“选择日期”。右键单击日期控件，在弹出的快捷菜单中选择 Properties 命令，弹出 Date Time Picker Properties 窗口，设置 ID 为 IDC_DT_CHOICE。选择 Styles 选项卡，如图 4-63 所示。

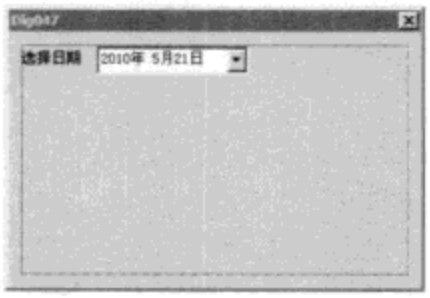


图 4-62　添加日期控件

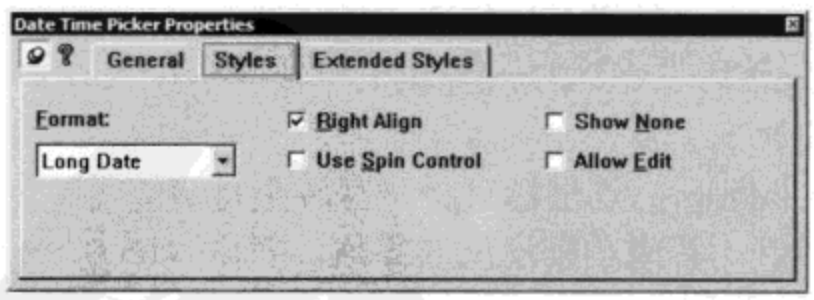


图 4-63　日期控件的 Styles 选项卡

Format 组合框设置控件显示格式，其中 Date 为日期格式，Time 为时间格式。Use Spin Control 复选框设置是否使用微调按钮。

（3）Format 组合框选择 Long Date 项，取消勾选 Use Spin Control 复选框。

(4)按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_DT_ CHOICE 项，添加 CDateTimeCtrl 类型的变量 m_dtCtrl，单击 OK 按钮保存并退出。

4.13.2　读取设置日期

（1）GetTime 函数获取日期控件当前显示的日期，格式如下：

```
DWORD CDateTimeCtrl::GetTime(CTime& timeDest) const
```

参数如下。

- timeDest：CTime 对象的引用，存放日期信息。

返回值：控件的日期状态。

（2）SetTime 函数设置日期控件显示的日期值，格式如下：

```
BOOL CDateTimeCtrl::SetTime(const CTime* pTimeNew)
```

参数如下。

- pTimeNew：指向 CTime 日期对象的指针。

返回值：若成功返回非零值，否则为 0。

4.13.3　日期响应

日期控件选择的日期值改变后，触发 DTN_DATETIMECHANGE 消息，添加该消息的处理函数，实现特定功能。

（1）在资源视图双击 IDD_DLG047_DIALOG 项，右键单击日期控件，在弹出的快捷菜单中选择 Events 命令，选择 DTN_DATETIMECHANGE 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg047Dlg::OnDatetimechangeDtChoice(NMHDR* pNMHDR, LRESULT* pResult)
{
    CTime time;
    m_dtCtrl.GetTime(time);                    //获取日期控件的值
    CString strTime;
    strTime.Format("%d/%d/%d",time.GetYear(),time.GetMonth(),time.GetDay());
    SetWindowText(strTime);                    //设置标题栏显示文本
    *pResult = 0;
}
```

CTime 类用于操作日期和时间信息，常用函数如下。

- GetCurrentTime：静态成员，获取当前系统日期的 CTime 对象。
- GetYear：获取年份。
- GetMonth：获取月份，范围为 1～12。
- GetDay：获取天数，范围为 1～31。
- GetHour：获取小时数，范围为 0～23。
- GetMinute：获取分钟数，范围为 0～59。
- GetSecond：获取秒数，范围为 0～59。
- GetDayOfWeek：获取当前日期的星期数，范围为 1～7，其中 1 为星期日，7 为星期六。

GetTime 函数获取控件的日期值，存放到 time 中。Format 函数将不同类型的值格式化为字符串，存放到 strTime 中。GetYear、GetMonth、GetDay 函数获取年、月、日的值。SetWindowText 函数设置对话框标题栏显示的文本。

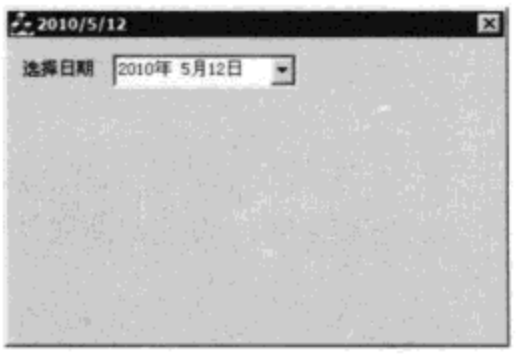


图 4-64　日期控件值改变

（2）生成程序并运行，如图 4-64 所示。日期控件选择值改变

后，标题栏自动更新为对应的日期。

4.14 高级控件

Visual C++自带的控件功能有限，只能满足基本的输入输出需求，可使用第三方开发的高级控件，实现更加丰富的功能，如 Windows Media Player 控件可用于播放视频文件，Flash 控件可用于显示 flash 动画，Web Browser 控件可用于显示 HTML 网页等，本节介绍 Windows Media Player、Flash 两个控件。

4.14.1 Windows Media Player 控件

Windows Media Player 是 Windows 自带的一款多媒体播放器，音频支持 MP3、WMA、WAV、MID 等格式，视频支持 AVI、WMV、MPEG 等格式，其核心功能由 Windows Media Player 控件完成，可以在程序中使用该控件，实现多媒体的播放功能。

【实例 4-14】在工程 Dlg047 中使用 Windows Media Player 控件，播放 AVI 视频文件。

（1）打开工程 Dlg047，选择 Project|Add To Project|Components and Controls 命令，弹出 Components and Controls Gallery 窗口，双击 Registered ActiveX Controls 文件夹，显示出所有已注册的 ActiveX 控件。单击一项，按下 w 键，定位到以 w 开头的控件，选择 Windows Media Player 项，单击 insert 按钮添加控件，如图 4-65 所示。

（2）在弹出的窗口中单击“确定”按钮，弹出 Confirm Classes 窗口，用于添加控件自动生成的类，单击 OK 按钮完成添加，再单击 Close 按钮关闭添加控件窗口。

（3）添加控件后，在控件工具箱里出现控件，拖放两个按钮控件、一个播放器控件到对话框模板 IDD_DLG047_DIALOG 上，如图 4-66 所示。

图 4-65 添加 Windows Media Player 控件

图 4-66 Windows Media Player 控件

（4）设置按钮的 Caption 依次为“打开”、“ 关闭”，ID 依次为 IDC_BUTTON_OPEN、IDC_BUTTON_CLOSE。设置播放器控件的 ID 为 IDC_OCX_PLAYER。

（5）按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_OCX_PLAYER 项，添加 CWMPPlayer4 类型的变量 m_player，单击 OK 按钮保存并退出。

（6）双击“打开”按钮，添加消息处理函数，先在函数所在文件的开头处添加一句#include "wmpcontrols.h"，包含 CWMPControls 类的头文件。在函数体内添加如下代码：

```
void CDlg047Dlg::OnButtonOpen()
{
    CFileDialog dlgFile(TRUE,"avi",NULL,NULL
        ,"AVI 视频文件 (*.avi)|*.avi||");                    //打开文件对话框
    if(dlgFile.DoModal()==IDOK)                              //显示对话框
    {
        CString strPath=dlgFile.GetPathName();               //选择文件的路径
        m_player.SetUrl(strPath);                            //设置播放器控件的文件路径
        CWMPControls wmpCtrl=m_player.GetControls();         //获取控件对象
```

```
        wmpCtrl.play();                                    //播放视频文件
    }
}
```

CFileDialog 类封装了 Windows 自带的文件对话框，提供了通用的文件打开和保存操作界面，关于 CFileDialog 类的具体介绍参见 11.1.2 节。

CFileDialog 类构造函数的第 1 个参数表示对话框类型，若为 TRUE 表示打开文件对话框，若为 FALSE 表示保存文件对话框，第 5 个参数为文件类型过滤字符串，用于只显示后缀名为 AVI 的文件，DoModal 函数以模态方式显示一个对话框窗口，模态窗口获取独占焦点，在关闭模态窗口前，不能将焦点切换到同一个程序的其他窗口中。若在打开文件对话框中，选择 AVI 文件后，单击“打开”按钮或直接双击文件名，DoModal 函数返回 IDOK，表示成功选择一个文件。

GetPathName 函数获取选择文件的路径，存放到 strPath 中。SetUrl 函数设置播放器控件的文件路径，GetControls 函数获取播放器控件的 CWMPControls 对象，CWMPControls 类提供一系列操作方法，如播放 play、暂停 pause、停止 stop 等，调用 play 函数开始播放 AVI 文件。

Tips 以#开头的语句都属于预编译指令，在编译之前执行。#include 指令包含所需的头文件，有#include ""和#include <>两种形式，区别在于文件的搜索方式。其中""方式先搜索当前工程所在的目录，再搜索系统目录，<>方式直接搜索系统目录。系统目录的具体路径可自由设置，选择 Tools|Options|命令，弹出 Options 窗口，切换到 Directories 选项卡，如图 4-67 所示。选择 Include files 项，列表框中显示的路径即为系统目录，可双击空白项添加新的系统目录。

（7）打开对话框模板，双击“关闭”按钮，添加如下代码：

```
void CDlg047Dlg::OnButtonClose()
{
    CWMPControls wmpCtrl=m_player.GetControls();
    wmpCtrl.stop();                                        //停止播放
}
```

（8）生成程序并运行，如图 4-68 所示。单击“打开”按钮，弹出打开文件对话框，选择一个 AVI 格式文件，播放器控件自动播放 AVI 文件，单击“关闭”按钮停止播放。

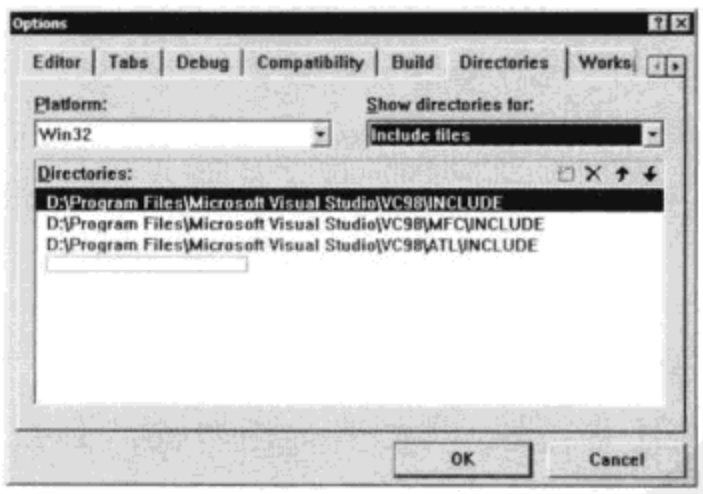


图 4-67　系统目录路径

图 4-68　文件播放

Tips 添加高级控件后，一般会在类视图中添加多个类，影响类视图操作。可在类视图单击右键，在弹出的快捷菜单中选择 New Folder 命令，输入新文件夹的名称如 WMP，单击 OK 按钮添加一个新文件夹。选中自动添加的多个类，用鼠标拖曳到新文件夹里，将相关的类放到一个目录下，简化界面。这只是形式上的组合，实际文件路径并未改变。

4.14.2 Flash 控件

Flash 动画是网页上广泛使用的一种动画技术，界面绚丽，且提供交互式操作功能。可在程序中使用 Flash 控件显示动画，增强界面的美观性。

【实例 4-15】新建一个对话框工程名为 Dlg048，使用 Flash 控件播放动画。

（1）新建一个对话框工程名为 Dlg048，选择 Project|Add To Project|Components and Controls 命令，弹出 Components and Controls Gallery 窗口，双击 Registered ActiveX Controls 文件夹，单击一项，按下 s 键，定位到以 s 开头的控件，选择 Shockwave Flash Object 项，单击 insert 按钮添加控件，操作方法同 Windows Media Player 控件。

图 4-69 添加 Flash 控件

（2）在控件工具箱里，拖放一个 Flash 控件到对话框模板 IDD_DLG048_DIALOG 上，如图 4-69 所示。

（3）按 Ctrl+W 组合键打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_ SHOCKWAVEFLASH1 项，添加 CShockwave Flash 类型的变量 m_flash，单击 OK 按钮保存并退出。

（4）在类视图双击 OnInitDialog 项，定位到函数，在 return TRUE; 前添加如下代码：

```
CString strDir;
GetCurrentDirectory(MAX_PATH,strDir.GetBuffer(MAX_PATH));      //获取当前工程所在目录
strDir.ReleaseBuffer();                                        //释放 CString 缓冲区
m_flash.SetMovie(strDir+"\\house.swf");                        //设置动画文件绝对路径
m_flash.SetPlaying(TRUE);                                      //播放动画
```

GetCurrentDirectory 函数获取当前程序所在的目录，格式如下：

```
DWORD GetCurrentDirectory(DWORD nBufferLength,LPTSTR lpBuffer)
```

参数如下。

- nBufferLength：存放路径的字符缓冲区的长度，MAX_PATH 是系统定义的路径字符串的最大长度，值为 260。
- lpBuffer：指向字符缓冲区的指针。

返回值：若成功则返回字符串实际长度，否则返回 0。

> **Tips** 若在 Visual C++开发环境里执行程序，调用 GetCurrentDirectory 函数得到 DSW 工程文件所在的目录，若直接双击 EXE 程序，调用 GetCurrentDirectory 函数得到 EXE 文件所在的目录。

GetBuffer 函数获取指向 CString 对象内部字符缓冲区的指针，返回 LPTSTR 指针，由于没有 const 修饰，可修改字符缓冲区的值，但在使用 CString 类的其他函数前，必须先调用 ReleaseBuffer 函数释放缓冲区。GetBuffer 函数格式如下：

```
LPTSTR CString::GetBuffer(int nMinBufLength)
```

参数如下。

- nMinBufLength：字符缓冲区的最小长度。

返回值：指向字符缓冲区的 LPTSTR 指针。

CShockwaveFlash 类提供一系列函数操作 Flash 控件，SetMovie 函数设置 SWF 动画文件的绝对路径，SetPlaying 函数设置是否播放动画文件。

（5）生成程序并运行，如图 4-70 所示。

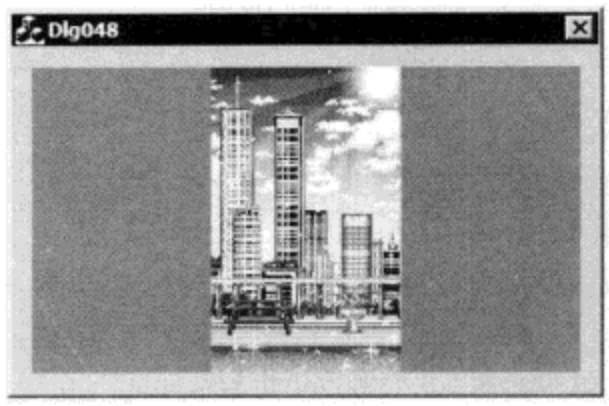


图 4-70　播放动画

4.15 小结

随着时代的发展，用户对于软件的人机交互性要求越来越高，控件是窗口的重要组成部分，掌握控件的使用方法尤为重要。本章主要介绍了 Visual C++开发环境中各种常用控件，对于每个控件的介绍都很细致。先介绍如何在 Visual C++环境中设置控件属性，再对控件进行编辑，设置控件的消息响应函数，实现特定的功能。

4.16 习题

1．应用程序类的作用是什么？
2．按钮控件包括哪几种，它们的功能和常用属性有哪些？
3．在 Visual C++中，如何改变按钮上显示的名称？
4．创建一个用户登录对话框，要求包括静态文本、编辑框和按钮等控件。

第5章　对话框

对话框常作为控件的父窗口，实现与用户的交互操作，如数据的输入显示、执行按钮命令等。对话框分为模态（modal）和非模态（modeless）两种，模态对话框显示后，获取独占状态，只有关闭模态对话框后，才能执行其他操作，如文件选择对话框。非模态对话框可以与其他窗口协同工作、切换焦点，如工具栏窗口。

一个对话框对象由对话框模板和 CDialog 派生类共同组成，其中对话框模板负责窗口的界面、控件的布局，CDialog 派生类负责对话框、控件的逻辑操作。一般先在资源视图添加对话框模板，完成控件布局、属性设置等操作，根据对话框模板创建一个对应的 CDialog 派生类，在派生类中添加消息处理函数，实现特定功能。

5.1 模态对话框

创建并显示一个模态对话框，步骤如下。

（1）添加对话框模板资源，在模板上拖放控件，设计界面布局。

（2）为对话框模板添加对应的 CDialog 派生类，用于操作该对话框。

（3）添加对话框初始化函数，初始化对话框及控件。

（4）调用 DoModal 函数，显示模态对话框，并获取返回值。

5.1.1 添加对话框资源

【实例 5-1】新建一个对话框工程名为 Dlg049，添加一个“个人信息”对话框，用于输入个人信息。在主窗口中单击“信息录入”按钮，打开模态形式的“个人信息”对话框，输入个人信息后，单击 OK 按钮关闭对话框，并将输入信息添加到主窗口的列表控件中。

（1）新建一个对话框工程名为 Dlg049，在资源视图右键单击 Dialog 项，在弹出的快捷菜单中选择 Insert Dialog 命令，添加一个对话框资源。右键单击对话框模板，在弹出的快捷菜单中选择 Properties 命令，弹出 Dialog Properties 窗口，如图 5-1 所示。

（2）设置对话框的 ID 为 IDD_DIALOG_INPUT，Caption 为“个人信息”。单击 Font 按钮，弹出 Select Dialog Font 窗口，设置 Font Name 为“宋体”，Font Size 为 9。

（3）拖放三个静态文本、两个编辑框、一个组合框到对话框模板 IDD_DIALOG_INPUT 上，如图 5-2 所示。

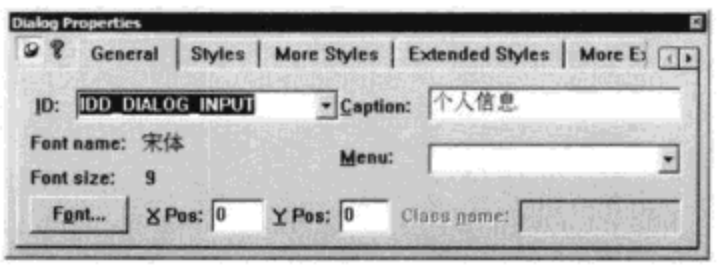


图 5-1　对话框 Properties 窗口

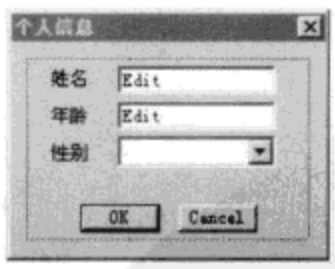


图 5-2　添加的对话框资源

（4）设置静态文本的 Caption 依次为“姓名”、“年龄”、“性别”，设置编辑框的 ID 依次为 IDC_EDIT_NAME、IDC_EDIT_AGE，并勾选“年龄”编辑框属性窗口的 Number 复选框。

（5）设置组合框的 ID 为 IDC_COMBO_SEX，在组合框属性窗口的 Styles 选项卡里，设置 Type 为 Drop List，取消勾选 Sort 复选框，单击组合框的箭头，调整下拉框的高度。

5.1.2　添加对话框类

创建对话框资源后，添加一个对应的对话框类操作该对话框，完成创建、初始化、消息响应、销毁等操作，如主对话框 IDD_DLG049_DIALOG 对应 CDlg049Dlg 类，关于对话框 IDD_ABOUTBOX 对应 CAboutDlg 类。

（1）在资源视图双击 IDD_DIALOG_INPUT 项，按 Ctrl+W 组合键打开类向导窗口，自动弹出 Adding a Class 窗口，单击 OK 按钮，打开 New Class 窗口，如图 5-3 所示。

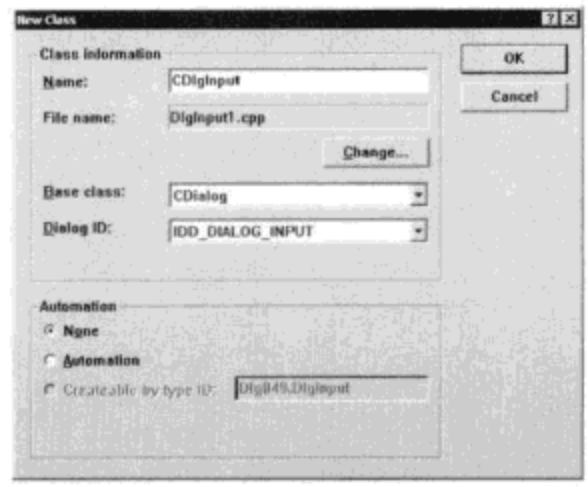


图 5-3　添加对话框类

在 Name 编辑框输入新类的名称，一般类名以 C 开头。Base class 组合框选择新添加类的基类，默认为 CDialog。Dialog ID 组合框选择对话框模板的 ID。

（2）在 Name 编辑框中输入 CDlgInput，单击 OK 按钮完成添加。

（3）在类向导窗口，选择 Member Variable 选项卡，Class name 组合框选择 CDlgInput 项，添加三个控件变量，如表 5-1 所示。

表 5-1　对话框控件映射变量

控件 ID	变量类型	变 量 名
IDC_COMBO_SEX	CComboBox	m_comboSex
IDC_EDIT_NAME	CString	m_strName
IDC_EDIT_AGE	int	m_nAge

5.1.3　初始化对话框

对话框在显示之前，可进行一些初始化工作，如组合框添加项、列表控件插入列等。对话框初始化时，触发 WM_INITDIALOG 消息，可添加该消息的处理函数，完成初始化操作。

（1）在类视图右键单击 CDlgInput 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，选择 WM_INITDIALOG 项，单击 Add and Edit 按钮，添加如下代码：

```
BOOL CDlgInput::OnInitDialog()
{
    CDialog::OnInitDialog();              //调用基类 OnInitDialog 函数
    m_comboSex.AddString("男");           //性别组合框添加项
    m_comboSex.AddString("女");
    m_comboSex.SetCurSel(0);              //组合框选中第一项
    return TRUE;
}
```

在 OnInitDialog 函数的 return TRUE; 前添加代码，完成控件的初始化操作。

Tips　与控件、窗口有关的初始化操作要在 OnInitDialog 函数里完成，与窗口无关的变量可在对话框类的构造函数和 OnInitDialog 函数里完成。程序执行到对话框类的构造函数时，窗口与控件尚未创建，不能在构造函数里进行控件的初始化。

5.1.4 显示模态对话框

若一个对话框窗口以模态形式显示，该窗口在关闭之前，一直处于焦点状态。模态窗口可以打开另一个模态窗口，但只能对顶层的模态窗口进行操作。

（1）在类视图右键单击 CDlgInput 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加 CString 类型的变量 m_strSex，保存选择的性别。

（2）在资源视图双击 IDD_DIALOG_INPUT 项，打开对话框模板，双击 OK 按钮，添加如下代码：

```
void CDlgInput::OnOK()
{
    UpdateData(TRUE);                                    //更新控件映射变量的值
    if(m_strName.IsEmpty() || m_nAge==0)                 //若名称为空或年龄为 0，弹出提示框
    {
        AfxMessageBox("请输入完整信息！");
        return;
    }
    m_comboSex.GetLBText(m_comboSex.GetCurSel(),m_strSex);    //获取选择的性别
    CDialog::OnOK();                                     //关闭对话框
}
```

UpdateData 函数更新控件或变量的值，若参数为 TRUE 用控件值更新变量，若为 FALSE 用变量值更新控件。AfxMessageBox 函数用于弹出一个消息提示框，格式如下：

```
int AfxMessageBox(LPCTSTR lpszText,UINT nType = MB_OK,UINT nIDHelp = 0)
```

参数如下。

- lpszText：提示信息字符串。
- nType：提示框的样式，可设置按钮和图标，如 MB_OK 为“确定”按钮，MB_OKCANCEL 为“确定”和“取消”按钮。
- nIDHelp：帮助信息的 ID。

返回值：若单击“确定”按钮则返回 IDOK，若单击“取消”按钮则返回 IDCANCEL。

GetCurSel 函数获取组合框选中项的索引，GetLBText 获取选择的性别，存放到 m_strSex 中。CDialog 类的 OnOK 函数关闭 DoModal 函数创建的模态对话框，并使 DoModal 函数返回 IDOK。

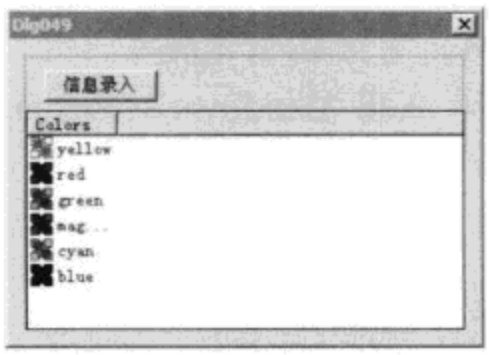


图 5-4 主对话框添加控件

（3）在资源视图双击 IDD_DLG049_DIALOG 项，拖放按钮、列表控件到主对话框模板上，如图 5-4 所示。

（4）设置按钮的 ID 为 IDC_BUTTON_INPUT，Caption 为“信息录入”。设置列表控件属性窗口 Styles 选项卡的 View 为 Report 项。打开类向导窗口，选择 Member Variable 选项卡，双击 IDC_LIST1 项，添加 CListCtrl 类型的变量 m_list。

（5）在类视图双击 CDlg049Dlg 类下的 OnInitDialog 项，定位到函数，在 return TRUE; 前添加如下代码：

```
m_list.InsertColumn(0,"姓名",LVCFMT_LEFT,80);                    //添加三列
m_list.InsertColumn(1,"年龄",LVCFMT_LEFT,50);
m_list.InsertColumn(2,"性别",LVCFMT_LEFT,50);
```

（6）在资源视图双击 IDD_DLG049_DIALOG 项，打开主对话框模板，双击“信息录入”按钮，添加消息处理函数。在函数所在文件的开头处，添加一句#include "DlgInput.h"，包含 CDlgInput 类的头文件。在函数体内添加如下代码：

```
void CDlg049Dlg::OnButtonInput()
```

```
{
    CDlgInput dlg;                                          //个人信息对话框类对象
    if(dlg.DoModal()==IDOK)                                 //调用模态对话框
    {
        CString strName=dlg.m_strName;                      //获取名称
        CString strSex=dlg.m_strSex;                        //获取性别
        int nAge=dlg.m_nAge;                                //获取年龄
        CString strAge;
        strAge.Format("%d",nAge);                           //将整型年龄格式化为字符串
        int nItem=m_list.InsertItem(m_list.GetItemCount(),strName);    //插入一项
        m_list.SetItemText(nItem,1,strAge);                 //设置插入项的值
        m_list.SetItemText(nItem,2,strSex);
    }
}
```

CDlgInput 类为添加的个人信息对话框类，dlg 为对话框类对象，DoModal 函数以模态形式显示对话框，直到模态对话框关闭该函数才返回，格式如下：

```
int CDialog::DoModal()
```

返回值：模态对话框的操作结果，若调用 CDialog::OnOK 函数关闭模态对话框，则返回 IDOK，若调用 CDialog::OnCancel 函数，则返回 IDCANCEL。

Tips 模态对话框使用 CDialog::EndDialog（int nResult）函数关闭对话框，OnOK 和 OnCancel 函数都通过调用 EndDialog 函数关闭对话框，不同之处在于 OnOK 函数调用 EndDialog(IDOK); OnCancel 函数调用 EndDialog(IDCANCEL); DoModal 函数返回 EndDialog 函数传入的参数值。

若 DoModal 函数返回 IDOK，表示通过单击 OK 按钮关闭对话框，需要保存输入信息。其中 dlg.m_strName 为输入的名称，dlg.m_strSex 为性别，dlg.m_nAge 为整型的年龄值，存放到三个变量中。

Format 函数将整型的年龄值格式化为字符串，存放到 strAge 中。InsertItem 函数在列表控件中添加一项，参数依次为插入位置的索引、新项的值。GetItemCount 函数获取项的数目，用于在尾部插入一项。SetItemText 函数设置添加项的其他列的值，参数依次为插入项的索引、列索引、文本值。

Tips DoModal 函数返回后，模态对话框窗口已经不存在，对话框类对象仍存在，只能获取非窗口类型的变量，不能获取控件变量如 dlg.m_comboSex。列表控件添加项时，第 1 列使用 InsertItem 函数插入，其余列使用 SetItemText 函数设置。

（7）生成程序并运行，单击“信息录入”按钮，打开“个人信息”模态对话框，如图 5-5 所示。输入姓名、年龄，选择性别，单击 OK 按钮完成添加并关闭对话框后，主对话框的列表控件显示添加的信息，如图 5-6 所示。

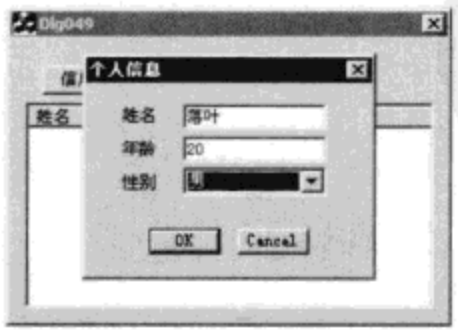


图 5-5　个人信息对话框

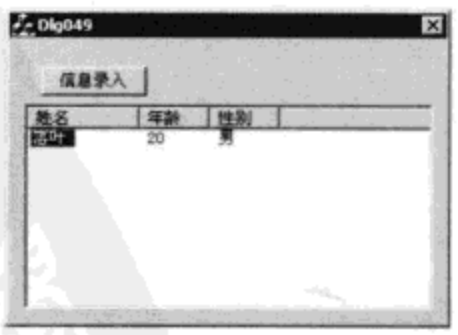


图 5-6　主对话框显示添加信息

5.2 非模态对话框

非模态对话框可以与其他窗口协同工作，在不同窗口间切换焦点。模态对话框调用 DoModal 函数完成窗口的创建、显示，非模态对话框调用 Create 函数完成创建，调用 ShowWindow 函数控制显示状态。

【实例 5-2】新建一个对话框工程名为 Dlg0410，添加一个对话框，以非模态窗口形式显示，在编辑框中同步显示主窗口中列表框选中项的值，且可更新值。

（1）新建一个对话框工程名为 Dlg0410，在资源视图右键单击 Dialog 项，在弹出的快捷菜单中选择 Insert Dialog 命令，添加一个对话框资源。右键单击对话框模板，在弹出的快捷菜单中选择 Properties 命令，弹出 Dialog Properties 窗口。

（2）设置对话框的 ID 为 IDD_DIALOG_VALUE，Caption 为“更新项”。单击 Font 按钮，设置 Font Name 为“宋体”，Font Size 为 9。

（3）拖放编辑框、按钮到对话框模板上，调整对话框大小，如图 5-7 所示。设置编辑框的 ID 为 IDC_EDIT_VALUE，设置按钮的 ID 为 IDC_BUTTON_ UPDATE，Caption 为“更新”。

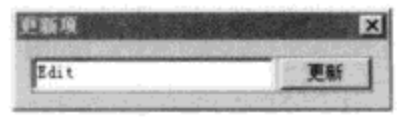


图 5-7 “更新项”对话框

（4）双击 IDD_DLG0410_DIALOG 项，拖放列表框、按钮到对话框模板上，如图 5-8 所示。设置按钮的 ID 为 IDC_BUTTON_ SHOW，Caption 为“编辑窗口”。取消勾选列表框属性窗口的 Sort 复选框。

图 5-8 主对话框窗口

（5）双击 IDD_DIALOG_VALUE 项，按 Ctrl+W 组合键打开类向导窗口，自动弹出 Adding a Class 窗口，单击 OK 按钮，打开 New Class 窗口，在 Name 编辑框中输入 CDlgValue，单击 OK 按钮保存。

（6）在类向导窗口中，选择 Member Variable 选项卡，在 Class name 组合框选择 CDlgValue 项，双击 IDC_EDIT_VALUE 项添加 CString 类型的变量 m_strValue。

（7）在 Class name 组合框中选择 CDlg0410Dlg 项，双击 IDC_LIST1 项，添加 CListBox 类型的变量 m_list，单击 OK 按钮保存并退出。

（8）在类视图双击 CDig0410Dlg 项，打开 CDlg0410Dlg 类的头文件，在类声明前添加一句#include "DlgValue.h" ，包含 CDlgValue 类的头文件。

（9）右键单击 CDlg0410Dlg 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加 CDlgValue 类型的变量 m_dlgValue，用来创建非模态对话框。

（10）在类视图双击 CDlg0410Dlg 类下的 OnInitDialog 项，定位到函数，在 return TRUE; 前添加如下代码：

```
m_dlgValue.Create(IDD_DIALOG_VALUE,this);          //创建非模态对话框
m_dlgValue.ShowWindow(FALSE);                      //隐藏窗口
m_list.AddString("礼物");                          //主对话框的列表框添加项
m_list.AddString("故乡");
m_list.AddString("旅行");
m_list.AddString("故事");
m_list.AddString("在路上");
m_list.AddString("曾经的你");
m_list.AddString("天鹅之旅");
```

Create 函数根据对话框模板资源，创建非模态对话框，格式如下：

```
BOOL CDialog::Create(UNIT nIDTemplate,CWnd* pParentWnd = NULL)
```

参数如下。

- ❑ nIDTemplate：对话框模板 ID。
- ❑ pParentWnd：对话框父窗口的指针，默认为 NULL 表示父窗口为主窗口。

返回值：若成功则返回非零值，否则为 0。

Tips 若通过在对话框模板上拖放控件的方式添加控件，则程序根据生成的 rc 资源文件，自动创建控件。若以代码方式创建窗口或控件，先调用构造函数创建一个窗口或控件类对象，再调用 Create 函数创建一个窗口或控件实例，并关联到类对象。

this 指针代表当前类对象，ShowWindow 函数传入 FALSE 隐藏创建的非模态对话框，AddString 函数在列表框尾部添加一项。

（11）在资源视图双击 IDD_DLG0410_DIALOG 项，打开主对话框模板，双击“编辑窗口”按钮，添加如下代码：

```
void CDlg0410Dlg::OnButtonShow()
{
    m_dlgValue.ShowWindow(TRUE);                          //显示“更新项”窗口
}
```

（12）打开主对话框模板，右键单击列表框，在弹出的快捷菜单中选择 Events 命令，选择 LBN_SELCHANGE 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlg0410Dlg::OnSelchangeList1()
{
    if(m_list.GetCurSel()==LB_ERR)                        //若没有选中，返回
        return;
    if(m_dlgValue.IsWindowVisible())                      //若子窗口已显示
    {
        m_list.GetText(m_list.GetCurSel(),m_dlgValue.m_strValue);//将选中项的值传递
给子窗口
        m_dlgValue.UpdateData(FALSE);                     //更新子窗口中的编辑框
    }
}
```

GetCurSel 函数获取列表框选中项的索引，若没有选中项直接返回。IsWindowVisible 函数判断子窗口是否显示，若显示则返回 TRUE，否则返回 FALSE。若子窗口已显示，GetText 函数获取列表框选中项的值，存放到子窗口的 m_strValue 中。UpdateData 函数更新子窗口的编辑框。

（13）在资源视图双击 IDD_DIALOG_VALUE 项，双击“更新”按钮，在函数所在文件的开头处添加一句#include "Dlg0410Dlg.h"，包含 CDlg0410Dlg 类的头文件。在函数体内添加如下代码：

```
void CDlgValue::OnButtonUpdate()
{
    UpdateData(TRUE);                                     //更新变量值
    CDlg0410Dlg* pParent=(CDlg0410Dlg*)GetParent();       //获取主对话框的指针
    int nSel=pParent->m_list.GetCurSel();                 //获取列表框选中项索引
    if(nSel!=LB_ERR && !m_strValue.IsEmpty())             //若选中且编辑框不为空
    {
        pParent->m_list.DeleteString(nSel);               //删除选中项
        pParent->m_list.InsertString(nSel,m_strValue);    //在选中位置插入新值
    }
}
```

UpdateData 函数更新控件映射变量的值，GetParent 函数获取父窗口的指针，格式如下：

```
CWnd* CWnd::GetParent() const
```

返回值：父窗口的指针，若没有父窗口则返回 NULL。

GetParent 函数返回 CWnd 类型的指针，要强制转换为 CDlg0410Dlg 类型的指针，以便调用类成员。若列表框有选中项，且编辑框输入不为空，先调用 DeleteString 函数删除选中项，再调用 InsertString 函数将新值插入删除位置，间接实现值的更新。

> **Tips** C++是强类型语言，虽然 GetParent 函数返回的 CWnd 类指针确实指向父窗口，但必须转换为 CDlg0410Dlg 类指针，才能调用类成员，否则编译器会报错。

（14）在类视图右键单击 CDlgValue 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，选择 WM_CLOSE 项，单击 Add and Edit 按钮，添加如下代码：

```
void CDlgValue::OnClose()
{
    ShowWindow(FALSE);                                    //隐藏窗口
}
```

当单击对话框标题栏的关闭按钮，或按下 Esc 键时，触发 WM_CLOSE 消息，可添加该消息的处理函数，执行特定操作。

EndDialog 函数用来终止一个模态对话框，若对话框为非模态，默认调用 EndDialog 函数会使对话框存在但不可见。要关闭一个非模态对话框，应调用 DestoryWindow 函数销毁窗口。默认状态下，单击非模态子窗口的关闭按钮或按 Esc 键，子窗口表面上消失，实际上仍存在。

在 OnClose 函数里移除自动添加的 CDialog::OnClose 函数，当单击关闭按钮时，不再调用基类的处理函数，仅调用 ShowWindow 函数隐藏窗口。当主窗口关闭时，自动销毁子窗口。

（15）生成程序并运行，单击“编辑窗口”按钮，显示非模态窗口，改变列表框的选中项，子窗口的编辑框同步显示选中项的值，如图 5-9 所示。在子窗口中修改编辑框的值，单击“更新”按钮后，主对话框中列表框选中项的值同步更新。

图 5-9 非模态对话框同步更新

5.3 小结

上一章介绍了控件的使用方法，本章主要介绍控件的载体——对话框。对话框可以分为两大类：模态对话框和非模态对话框。二者的区别在于当打开模态对话框后，它处于独占状态，只有在关闭模态对话框后，才能执行其他操作。而非模态对话框与其相反，可以与其他窗口协同工作、切换焦点。掌握二者的不同点，可以在开发过程中灵活使用对话框。

5.4 习题

1. 模态对话框与非模态对话框的区别有哪些？
2. 如何为对话框添加相应的对话框类？
3. 为控件添加事件的具体步骤是什么？
4. 简述对话框的创建流程。
5. 使用 Visual C++中的 MFC 创建基于对话框的 Windows 应用程序，编写一个简单的计算器。该程序可以进行加、减、乘、除运算。通过一组单选按钮，由用户选择进行相应的运算。

第3篇 Visual C++的应用

第6章 GDI图形编程

绘图是计算机软件常用的一个功能，如零件结构图、电子地图、流程图、地质剖面图等，常见的绘图软件能够满足基本的绘制需求，但在实际应用中经常需要开发专用的绘图程序，如根据地质数据自动生成地质剖面图，根据作物的历年产量数据生成趋势折线图等。Windows API提供一系列绘图函数，用于实现基本的图形绘制，MFC框架将图形绘制相关函数封装到CDC类中，在程序中调用CDC类提供的函数，实现图形绘制功能。

6.1 设备环境

设备环境（Device Context）相当于画布，可以在设备环境上进行图形绘制操作，设备可以是屏幕，也可以是打印机、绘图仪等。设备环境提供统一的图形绘制接口，屏蔽了不同设备的差异，提高了图形程序的通用性。

6.1.1 什么是设备环境

设备环境保存了绘图操作的属性设置，如当前使用的画笔、画刷、字体等，可通过CDC类提供的函数改变属性，设备环境的属性在改变之前持续有效，应在绘图完成后恢复原始属性。

设备环境也是一种资源，类似于窗口使用HWND句柄作为标识符，设备环境使用HDC句柄作为唯一标识符。CDC类提供了Windows中所有可用的绘图函数，但只是将类对象的HDC句柄成员传递给底层的Windows API函数。也可直接调用API函数，只需要多传入一个HDC参数。

窗口分为客户区和非客户区，客户区一般显示用户需要的信息，不包括标题栏、菜单栏、工具栏、状态栏、边框等区域，CWnd及其派生类通过GetDC函数获取客户区的设备环境，只能在客户区绘图，格式如下：

```
CDC* CWnd::GetDC()
```

返回值：若成功则返回CWnd客户区的设备环境指针，否则返回NULL。

通过GetWindowDC函数获取整个窗口的设备环境，可在窗口的任何位置绘图，格式如下：

```
CDC* CWnd::GetWindowDC()
```

返回值：若成功则返回窗口的设备环境指针，否则返回NULL。

在绘图完成后，应调用ReleaseDC函数释放设备环境，格式如下：

```
int CWnd::ReleaseDC(CDC* pDC)
```

参数如下。

- pDC：要释放的设备环境的指针。

返回值：若成功则返回非零值，否则为0。

6.1.2 设备环境分类

CDC是通用的设备环境类，提供了几个派生类，如CClientDC、CWindowDC、CPaintDC，还有一种内存设备环境，适宜在特定环境下使用。

CClientDC客户设备环境用于在客户区绘图，在其构造函数里传入CWnd指针，内部调用GetDC

函数获取窗口的客户区 DC，在析构函数里调用 ReleaseDC 函数释放获取的 DC，构造函数格式如下：

```
CClientDC::CClientDC(CWnd* pWnd)
```

参数如下。

- pWnd：要获取的客户区 DC 的窗口指针。

创建 CClientDC 类对象时只需传入 CWnd 指针，CClientDC 类自动实现客户区 DC 的获取与释放。与 CClientDC 对应的是 CWindowDC 窗口设备环境，CWindowDC 能够在窗口的任意位置绘图，构造函数同 CClientDC，唯一区别在于 CWindowDC 获取整个窗口的 DC。

CPaintDC 用来处理 WM_PAINT 消息，通常在 OnPaint 重绘函数里使用，在其构造函数里调用 CWnd::BeginPaint，获取用于绘图的 DC，在析构函数里调用 CWnd::EndPaint 结束绘图，构造函数格式如下：

```
CPaintDC::CPaintDC(CWnd* pWnd)
```

参数如下。

- pWnd：重绘窗口的指针。

BeginPaint 函数不仅获取绘图所需 DC，并将绘图相关信息存放到 PAINTSTRUCT 结构体中，结构体的一个成员 RECT rcPaint，指定需要重绘的矩形区域，当绘图量较大时，使用 rcPaint 只重绘需要绘制的区域，可以提高程序的执行效率。CPaintDC 的一个成员变量 PAINTSTRUCT m_ps，用于存放 BeginPaint 函数获取的 PAINTSTRUCT 信息。

内存设备环境是理论上的一种 DC，常用于绘图量较大的操作，使用内存 DC 先在后台内存中绘制完图像，替代屏幕上的直接绘制，再将内存中的图像直接复制到屏幕上，避免重绘造成的闪烁。如绘制图形需要 2 秒，复制到屏幕上需要 0.1 秒，若不使用内存 DC，绘制图形的 2 秒内屏幕会不停闪烁，影响视觉效果，若使用内存 DC，屏幕从空白状态直接变为绘制完成，减少了屏幕的闪烁。

一个内存 DC 对应一个窗口 DC，利用窗口 DC 调用 CreateCompatibleDC 函数，创建一个兼容的内存 DC，将绘图操作用内存 DC 实现，在后台完成图像的绘制，再调用 BitBlt 函数，将内存中的图像直接复制到当前屏幕上。

CreateCompatibleDC 函数用于创建一个与窗口 DC 兼容的内存 DC，格式如下：

```
BOOL CDC::CreateCompatibleDC(CDC* pDC)
```

参数如下。

- pDC：窗口 DC 的指针。

返回值：若成功则返回非零值，否则为 0。

BitBlt 函数用于将源 DC 的图像复制到当前 DC 中，格式如下：

```
BOOL CDC::BitBlt(int x,int y,int nWidth,int nHeight,CDC* pSrcDC,int xSrc,int
ySrc,DWORD dwRop)
```

参数如下。

- x：复制的矩形区域的左上角 *x* 值。
- y：复制的矩形区域的左上角 *y* 值。
- nWidth：复制的矩形区域的宽度。
- nHeight：复制的矩形区域的高度。
- pSrcDC：源设备环境。
- xSrc：源图像复制的起点 *x* 值。
- ySrc：源图像复制的起点 *y* 值。
- dwRop：图像操作效果。

返回值：若成功则返回非零值，否则为 0。

6.2 图形绘制

CDC 类提供一系列绘图函数，可以在设备环境上绘制各种图形，如点、直线、曲线、矩形、椭圆等。图形的绘制效果与当前 DC 的设置有关，若设置当前 DC 使用红色画笔，在使用新画笔之前，所有绘制操作都是用红色画笔，在图形绘制完成后，应恢复原始的 DC 状态。

6.2.1 点线

SetPixel 函数用于设置点的颜色值，格式如下：

```
COLORREF CDC::SetPixel(int x,int y,COLORREF crColor)
```

参数如下。

- x：点的 x 坐标值。
- y：点的 y 坐标值。
- crColor：点的颜色值。

返回值：实际显示的颜色值。

GetPixel 函数获取指定点的颜色值，格式如下：

```
COLORREF CDC::GetPixel(int x,int y) const
```

参数如下。

- x：点的 x 坐标值。
- y：点的 y 坐标值。

返回值：指定点的颜色值。

COLORREF 是表示颜色的 DWORD 类型的值，常用 RGB 宏设置，格式如下：

```
COLORREF RGB(BYTE byRed,BYTE byGreen,BYTE byBlue)
```

参数如下。

- byRed：红色值。
- byGreen：绿色值。
- byBlue：蓝色值。

返回值：COLORREF 类型的颜色值。

若要获取 COLORREF 颜色值中的红、绿、蓝各个分量的值，可使用如下三个宏：

```
BYTE GetRValue(DWORD rgb)
BYTE GetGValue(DWORD rgb)
BYTE GetBValue(DWORD rgb)
```

参数如下。

- rgb：COLORREF 或 DWORD 类型的颜色值。

返回值：分别返回红色分量、绿色分量、蓝色分量的值。

CDC 类提供一些基本的绘线函数，如直线、圆弧、折线、多段线、贝塞尔曲线等，其中部分绘制函数与设备环境画笔的当前位置有关，根据当前位置开始绘制，或者绘制完成后设置最后一个点为当前位置。MoveTo 函数用于设置画笔当前位置，格式如下：

```
CPoint CDC::MoveTo(int x,int y)
```

参数如下。

- x：新位置的 x 坐标。
- y：新位置的 y 坐标。

返回值：先前位置的点坐标。

LineTo 函数用于从当前位置开始绘制一条直线，但不包括参数指定的终点，并设置终点为当前位置，格式如下：

```
BOOL CDC::LineTo(int x,int y)
```

参数如下。

- x：终点的 x 坐标。
- y：终点的 y 坐标。

返回值：若直线已经绘制则返回非零值，否则为 0。

Arc 函数用于绘制一段圆弧，圆弧实际上是椭圆的一部分，根据椭圆的矩形边界和圆弧的起始点和结束点，获取从起始点到结束点的逆时针方向的一段圆弧，圆弧的实际起始点是起始点和椭圆中心的连线与椭圆的交点，实际结束点是结束点与椭圆中心的连线与椭圆的交点。格式如下：

```
BOOL CDC::Arc(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4)
```

参数如下。

- x1：矩形边界的左上角 x 坐标。
- y1：矩形边界的左上角 y 坐标。
- x2：矩形边界的右下角 x 坐标。
- y2：矩形边界的右下角 y 坐标。
- x3：起始点的 x 坐标，不一定在椭圆上。
- y3：起始点的 y 坐标，不一定在椭圆上。
- x4：结束点的 x 坐标，不一定在椭圆上。
- y4：结束点的 y 坐标，不一定在椭圆上。

返回值：若成功则返回非零值，否则为 0。

ArcTo 函数的功能类似 Arc 函数，区别在于 ArcTo 函数同时设置结束点为当前位置。

PolyLine 函数用于绘制折线，按照参数点数组的顺序，依次连接构成折线，该函数不使用也不更新当前位置，格式如下：

```
BOOL CDC::PolyLine(LPPOINT lpPoints,int nCount)
```

参数如下。

- lpPoints：点数组的起始地址。
- nCount：数组中参与绘制的点数目。

返回值：若成功则返回非零值，否则为 0。

PolyLineTo 函数类似 PolyLine 函数用于绘制折线，区别在于 PolyLineTo 函数使用当前位置，从当前位置到第 1 个点也绘制一段直线，同时设置终点为当前位置。

PolyPolyLine 函数用于多段线，即多条单独的折线，不使用也不更新当前位置，格式如下：

```
BOOL CDC::PolyPolyLine(const POINT* lpPoints,const DWORD* lpPolyPoints,int nCount)
```

参数如下。

- lpPoints：所有折线的点数组的起始地址。
- lpPolyPoints：每条折线的点数目构成的数组的起始地址，若有两条折线，则数组大小为 2，第一个元素的值为折线 1 的点数目，其他元素依次类推。
- nCount lpPolyPoints：指向的数组的大小。

返回值：若成功则返回非零值，否则为 0。

PolyBezier 函数用于绘制贝塞尔曲线，每条贝塞尔曲线需要 4 个点，其中第 1 个点为曲线的起始点，第 4 个点为曲线的结束点，第 2、3 个点为曲线的控制点，用于控制曲线的走向，但不

在曲线上。该函数可以绘制一条或多条贝塞尔曲线，第 1 条曲线需要 4 个点，其余曲线只需要 3 个点，将上一条曲线的终点作为起始点，前两个点作为控制点，第 3 个点作为终点。格式如下：

```
BOOL CDC::PolyBezier(const POINT* lpPoints,int nCount)
```

参数如下。

- lpPoints：包含曲线所需的起止点和控制点的点数组的起始地址。
- nCount：点数组中参与绘制的点数目，由于第 1 条曲线需要 4 个点，其余曲线需要 3 个点，其值应为 $3n+1$，n 为曲线的数目。

返回值：若成功则返回非零值，否则为 0。

PolyBezierTo 函数类似 PolyBezier 函数，区别在于 PolyBezierTo 函数将最后一条曲线的终点设为当前位置。

【实例 6-1】新建一个单文档程序名为 Gdi051，在客户区内利用 CDC 类提供的绘点、直线、圆弧、折线、多段线、贝塞尔曲线函数，绘制各种图形。

（1）启动 Visual C++6.0，选择 FilelNew 命令，弹出 New 窗口，选择 MFC AppWizard(exe) 项，在 Project name 编辑框里输入 Gdi051，单击 OK 按钮，弹出 MFC AppWizard – Step 1 窗口，在工程类型中选择 Single Documents 项，单击 Finish 按钮完成单文档工程的创建。

单文档工程自动生成 CGdi051App、CGdi051Doc、CGdi051View、CMainFrame 四个主要的类，其中 App 类负责程序的启动和退出，Doc 类负责文件的读取和保存，View 类负责数据的显示和编辑，Frame 类负责界面的布局，本实例需要绘制图形，应在 View 类中实现。

（2）在类视图双击 CGdi051View 类下的 OnDraw 项，在 View 类的重绘函数里添加如下代码：

```
void CGdi051View::OnDraw(CDC* pDC)
{
    CGdi051Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rcClient;
    GetClientRect(rcClient);                                    //获取窗口客户区矩形大小

    int i=0;
    for(i=rcClient.Width()/2-60;i<rcClient.Width()/2+60;i++)
    {
        for(int j=rcClient.Height()/2-60;j<rcClient.Height()/2+60;j++)
            pDC->SetPixel(i,j,RGB(i%255,i%255,i%255));//设置指定范围内每个点的像素值
    }
    int x1=rcClient.Width()/2-100;                              //要绘制的矩形的左上角点坐标
    int y1=rcClient.Height()/2-100;
    int x2=rcClient.Width()/2+100;                              //要绘制的矩形的右下角点坐标
    int y2=rcClient.Height()/2+100;
    pDC->MoveTo(x1,y1);                                         //设置当前位置
    pDC->LineTo(x1+200,y1);                                     //绘制直线，并将终点设为当前位置
    pDC->LineTo(x1+200,y1+200);
    pDC->LineTo(x1,y1+200);
    pDC->LineTo(x1,y1);                                         //连续绘制 4 条直线，构成一个矩形边界
    pDC->Arc(x1,y1,x2,y2,x1+200,y1+100,x1,y1+100);     //绘制圆弧
    CPoint pt[7];                                               //点数组
    for(i=0;i<7;i++)
    {
        pt[i].x=rand()%(rcClient.Width()/2);                    //元素的坐标值随机赋值
        pt[i].y=rand()%(rcClient.Height()/2);
    }
    pDC->Polyline(pt,7);                                        //绘制折线，点数为 7
    for(i=0;i<7;i++)
        pt[i].y+=200;                                           //将点数组的 y 坐标值加 200，整体往下移
```

```
    DWORD number[2];                                    //多段线每条折线的点数目构成的数组
    number[0]=3;                                        //第 1 条折线有 3 个点
    number[1]=4;                                        //第 2 条折线有 4 个点
    pDC->PolyPolyline(pt,number,2);                     //绘制多段线，分为 2 段

    for(i=0;i<7;i++)
        pt[i].x+=400;                                   //将点数组的 x 坐标值加 400，整体往右移
    pDC->PolyBezier(pt,7);                              //绘制贝塞尔曲线
}
```

OnDraw 函数是 View 类的重绘函数，当窗口被遮盖后重新显示时，触发 WM_PAINT 重绘消息调用 OnPaint 函数，CView 类在 OnPaint 函数里调用 OnDraw 函数实现视图窗口的重绘操作，应将绘图有关的操作放到 OnDraw 函数里完成，当窗口重新显示时，保持内容不变。

GetClientRect 函数获取窗口客户区的矩形大小，存放到 rcClient 中。Width 函数获取矩形的宽度，Height 函数获取矩形的高度，两个 for 循环遍历指定矩形区域内的所有点，调用 SetPixel 函数设置区域内每个点的像素值，绘制出一块有颜色填充的矩形区域。

MoveTo 函数设置当前位置，LineTo 函数用于从当前位置到终点绘制一条直线，并将终点设为当前位置，连续 4 次调用 LineTo 函数绘制出一个矩形的四个边。

Arc 函数绘制圆弧，前 4 个参数指定圆弧所在的椭圆的外切矩形的左上角和右下角的坐标，第 5、6 个参数指定起始点坐标，第 7、8 个参数指定结束点坐标，从起始点到结束点按逆时针顺序得到圆弧。

rand 函数获取一个随机的 int 值，Polyline 函数绘制折线，根据参数 1 点数组的元素顺序，依次连接构成一条折线，参数 2 指定参与绘制的点数目。

number 数组指定多段线每条折线的点数目，PolyPolyline 函数绘制多段线，根据参数 1 点数组的元素排列顺序和参数 2 每条折线的点数目，依次绘制每一条折线，折线间互不影响。

PolyBezier 函数绘制贝塞尔曲线，其中第 1 条曲线使用 4 个点，其余曲线使用 3 个点，如本例中 1、2、3、4 点用于绘制第 1 条曲线，4、5、6、7 用于绘制第 2 条曲线。

（3）生成程序并运行，如图 6-1 所示。由于点数组的值是随机生成的，因此每次显示的图形效果是不一样的。中间有填充效果的矩形由 SetPixel 函数循环调用完成绘制，包围填充矩形的矩形边界由 LineTo 函数连续 4 次调用完成绘制，填充矩形上方的一段圆弧由 Arc 函数完成绘制。左上方的折线由 Polyline 函数完成绘制，该折线由 7 个点依次相连构成。左下方的两段折线由 PolyPolyline 函数完成绘制，第 1 条折线有 4 个点，第 2 条折线有 3 个点，两条折线彼此分开。右下方的曲线由 PolyBezier 函数完成绘制，由两条相连的贝塞尔曲线构成。

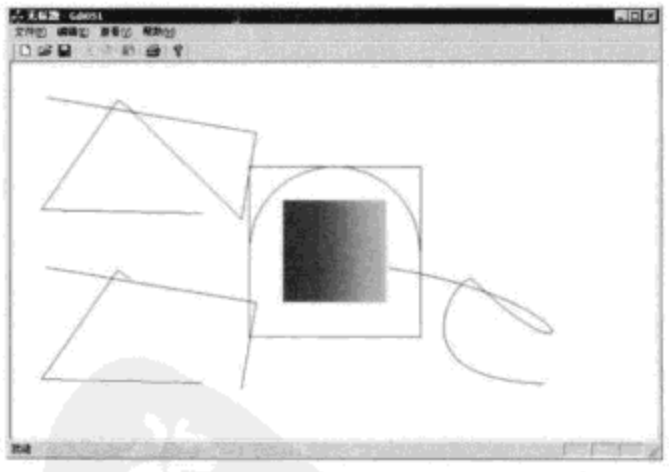


图 6-1　绘制点线图形

6.2.2　多边形

CDC 类提供一些基本的绘面函数，如直角矩形、圆角矩形、焦点矩形、椭圆、饼状图、多边形、多个多边形等，绘制面时使用当前画笔绘制边界轮廓，使用当前画刷填充内部区域。

Rectangle 函数用于绘制直角矩形，格式如下：

```
BOOL CDC::Rectangle(int x1,int y1,int x2,int y2)
BOOL CDC::Rectangle(LPCRECT lpRect)
```

参数如下。

- x1：矩形左上角 *x* 坐标

- y1：矩形左上角 *y* 坐标。
- x2：矩形右下角 *x* 坐标。
- y2：矩形右下角 *y* 坐标。
- lpRect Crect：矩形类对象。

返回值：若成功则返回非零值，否则为 0。

RoundRect 函数用于绘制圆角矩形，矩形四个角为圆弧形式，格式如下：

```
BOOL CDC::RoundRect(int x1,int y1,int x2,int y2,int x3,int y3)
BOOL CDC::RoundRect(LPCRECT lpRect,POINT point)
```

参数如下。

- x1：矩形左上角 *x* 坐标
- y1：矩形左上角 *y* 坐标。
- x2：矩形右下角 *x* 坐标。
- y2：矩形右下角 *y* 坐标。
- x3：圆角的椭圆宽度。
- y3：圆角的椭圆高度。
- lpRect Crect：矩形类对象。
- point：点对象，其中 *x* 值表示椭圆宽度，*y* 值表示椭圆高度。

返回值：若成功则返回非零值，否则为 0。

DrawFocusRect 函数绘制具有焦点风格的矩形，格式如下：

```
void CDC::DrawFocusRect(LPCRECT lpRect)
```

参数如下。

- lpRect：CRect 矩形类对象。

Ellipse 函数绘制用于绘制椭圆，椭圆包括圆形，格式如下：

```
BOOL CDC::Ellipse(int x1,int y1,int x2,int y2)
BOOL CDC::Ellipse(LPCRECT lpRect)
```

参数如下。

- x1：椭圆外切矩形的左上角 *x* 坐标
- y1：椭圆外切矩形的左上角 *y* 坐标。
- x2：椭圆外切矩形的右下角 *x* 坐标。
- y2：椭圆外切矩形的右下角 *y* 坐标。
- lpRect：椭圆外切矩形类对象。

返回值：若成功则返回非零值，否则为 0。

Pie 函数用于绘制饼形图，也称扇形图，是椭圆的一段圆弧和两端点与椭圆中心点连线构成的扇形区域，格式如下：

```
BOOL CDC::Pie(LPRECT lpRect,POINT ptStart,POINT ptEnd)
```

参数如下。

- lpRect：椭圆外切矩形类对象。
- ptStart：圆弧起点坐标。
- ptEnd：圆弧终点坐标。

返回值：若成功则返回非零值，否则为 0。

Polygon 函数用于绘制多边形，根据传入的点数组的元素顺序，首尾相连构成一个封闭的可填充的区域，格式如下：

```
BOOL CDC::Polygon(LPPOINT lpPoints,int nCount)
```

参数如下。

- lpPoints：点数组的起始地址。
- nCount：参与绘制的点数目。

返回值：若成功则返回非零值，否则为 0。

PolyPolygon 函数用于绘制两个及以上的多边形，格式如下：

```
BOOL CDC::PolyPolygon(LPPOINT lpPoints,LPINT lpPolyCount,int nCount)
```

参数如下。

- lpPoints：点数组的起始地址，包括所有多边形的点，按顺序排列。
- lpPolyCount：每个多边形拥有的点数构成的数组的起始地址。
- nCount lpPolyCount：数组的大小，要绘制的多边形的数目。

返回值：若成功则返回非零值，否则为 0。

【实例 6-2】新建一个单文档工程名为 Gdi052，利用 CDC 类提供的绘制矩形、椭圆、饼形图、多边形、多个多边形等函数，绘制各种图形。

（1）新建单文档工程 Gdi052，在类视图双击 CGdi052View 类下的 OnDraw 项，添加如下代码：

```
void CGdi052View::OnDraw(CDC* pDC)
{
    CGdi052Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rcClient;
    GetClientRect(rcClient);                                    //获取视图客户区矩形大小
    CPoint ptCenter=rcClient.CenterPoint();                     //客户区中心点坐标
    CRect rcRect(ptCenter.x-90,ptCenter.y-90,ptCenter.x+90,ptCenter.y+90);
    pDC->RoundRect(rcRect,CPoint(15,15));                       //绘制圆角矩形
    rcRect.DeflateRect(20,20);                                  //矩形 rcRect 缩小一定尺寸
    pDC->Rectangle(rcRect);                                     //绘制直角矩形
    rcRect.DeflateRect(20,20);                                  //矩形 rcRect 再缩小一定尺寸
    pDC->DrawFocusRect(rcRect);                                 //绘制焦点矩形
    pDC->Ellipse(rcRect);                                       //绘制椭圆
    rcRect.OffsetRect(0,-150);                                  //矩形 rcRect 上移
    CPoint start(rcRect.left,rcRect.CenterPoint().y+10);        //饼形图的圆弧起始点
    CPoint end(rcRect.right,rcRect.CenterPoint().y+10);         //饼形图的圆弧结束点
    pDC->Pie(rcRect,start,end);                                 //绘制饼形图
    CPoint pt[6];
    int i=0;
    for(i=0;i<6;i++)
    {
        pt[i].x=rand()%(rcClient.Width()/2);                    //点数组随机赋值
        pt[i].y=rand()%(rcClient.Height()/2);
    }
    pDC->Polygon(pt,6);                                         //绘制多边形
    for(i=0;i<6;i++)
        pt[i].x+=rcClient.Width()/2;                            //点数组整体右移
    int number[2];
    number[0]=3;                                                //第 1 个多边形有 3 个点
    number[1]=3;                                                //第 2 个多边形有 3 个点
    pDC->PolyPolygon(pt,number,2);                              //绘制多个多边形
}
```

GetClientRect 函数获取客户区的矩形大小，存放到 rcClient 中。CenterPoint 函数获取矩形的中心点坐标，格式如下：

```
CPoint CRect::CenterPoint() const
```

返回值：矩形的中心点坐标。

RoundRect 函数用于绘制圆角矩形，参数 2 指定圆角的宽度和高度。DeflateRect 函数减少矩形的宽度和高度，从四周向中心点移动实现缩放，格式如下：

```
void CRect::DeflateRect(int x,int y)
```

参数如下。

- x：矩形左右边界减少的值，矩形总宽度减少两倍的 x。
- y：矩形上下边界减少的值，矩形总高度减少两倍的 y。

Rectangle 函数用于绘制直角矩形，DrawFocusRect 函数用于绘制焦点矩形，Ellipse 函数用于绘制椭圆，OffsetRect 函数将矩形整体偏移一定位置，格式如下：

```
void CRect::OffsetRect(int x,int y)
```

参数如下。

- x：水平方向的偏移量，若向左移 x 为负值。
- y：垂直方向的偏移量，若向上移 y 为负值。

Pie 函数用于绘制饼形图，参数 1 指定外切矩形，参数 2、3 指定圆弧的起止点。Polygon 函数用于绘制多边形，参数 1 指定点数组，参数 2 指定数组大小，按照点顺序依次连接构成多边形。PolyPolygon 函数用于绘制多个多边形，根据所有点的坐标值和每个多边形的点数目，按顺序绘制各个多边形。

（2）生成程序并运行，如图 6-2 所示。由于点数组的值是随机生成的，因此每次显示的图形效果是不一样的。中间区域由外到内依次是圆角矩形、直角矩形、焦点矩形、椭圆，中间上方的饼形图由 Pie 函数完成绘制。左边的多边形由 Polygon 完成绘制，右边的两个多边形由 PolyPolygon 函数完成绘制。

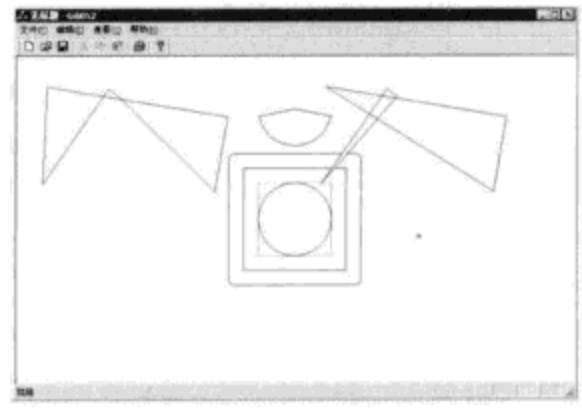


图 6-2　绘制多边形

6.2.3　文本

文本也是图形的一个重要组成元素，如地图上的地名标注、曲线图上的数据指标，需要在图形上绘制文本。CDC 类提供一些文本绘制函数实现文本的绘制及格式的控制，文本的格式在设置后，将持续有效直到重新设置，应根据需要设置对应的格式。

SetTextColor 函数用于设定文本的颜色，格式如下：

```
COLORREF CDC::SetTextColor(COLORREF crColor)
```

参数如下。

- crColor：文本颜色的 RGB 值。

返回值：先前的文本颜色值。

SetBkColor 函数用于设置当前背景色，可用于填充字符背景，格式如下：

```
COLORREF CDC::SetBkColor(COLORREF crColor)
```

参数如下。

- crColor：背景颜色的 RGB 值。

返回值：先前的背景色。

SetBkMode 函数用于设置背景模式，格式如下：

```
int CDC::SetBkMode(int nBkMode)
```

参数如下。

- nBkMode：背景模式，若为 OPAQUE 同时使用前景色和背景色，若为 TRANSPARENT

只使用前景色。

返回值：先前的背景模式。

TextOut 函数用于在指定位置开始绘制文本，格式如下：

```
BOOL CDC::TextOut(int x,int y,const CString& str)
```

参数如下。

- x：起点的 x 坐标值。
- y：起点的 y 坐标值。
- str：要绘制的文本。

返回值：若成功则返回非零值，否则为 0。

SetTextAlign 函数设置文本的对齐方式，默认为左对齐，格式如下：

```
UNIT CDC::SetTextAlign(UNIT nFlags)
```

参数如下。

- nFlags：文本对齐方式，如 TA_CENTER 表示文本的中部与起点对齐，TA_RIGHT 表示文本的右部与起点对齐。

返回值：先前的对齐方式。

ExtTextOut 函数用于在矩形范围内绘制文本，当文本超出矩形范围后可设置是否裁剪，格式如下：

```
BOOL CDC::ExtTextOut(int x,int y,UNIT nOptions,LPCRECT lpRect,const CString& str,LPINT
lpDxWidths)
```

参数如下。

- x：起点的 x 坐标值。
- y：起点的 y 坐标值。
- nOptions：矩形操作标志，如 ETO_CLIPPED 表示裁剪超出范围的文本，ETO_OPAQUE 表示使用背景色。
- lpRect：用于裁剪的矩形对象。
- str：要绘制的文本。
- lpDxWidths：字符间距数组的起始地址，默认为 NULL。

返回值：若成功则返回非零值，否则为 0。

DrawText 函数用于在矩形范围内绘制格式化的文本，可使用换行符\n 输出多行文字，格式如下：

```
int CDC::DrawText(const CString& str,LPRECT lpRect,UNIT nFormat)
```

参数如下。

- str：要绘制的文本。
- lpRect：控制范围的矩形对象。
- nFormat：格式化方法，如 DT_CENTER 表示文本水平居中，DT_LEFT 表示左对齐，DT_NOCLIP 表示当超出矩形范围后不剪切，DT_CALCRECT 表示矩形范围根据文本需要自动扩展。

返回值：若成功则返回文本高度。

【实例 6-3】新建一个单文档工程名为 Gdi053，根据 CDC 类提供的文本绘制和格式控制函数，绘制各种格式的文本。

（1）新建单文档工程 Gdi053，在类视图双击 CGdi053View 类下的 OnDraw 项，添加如下代码：

```
void CGdi053View::OnDraw(CDC* pDC)
{
```

```
    CGdi053Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    pDC->SetTextColor(RGB(0,0,255));                    //设置文本颜色
    pDC->SetBkColor(RGB(210,210,210));                  //设置文本背景颜色
    pDC->MoveTo(80,25);                                 //设置当前位置
    pDC->LineTo(80,200);                                //绘制一段直线
    pDC->TextOut(60,20,"Visual C++绘制文字");           //输出文本 1
    pDC->TextOut(80,50,"输出文字 TextOut");             //输出文本 2
    pDC->SetBkMode(TRANSPARENT);                        //设置背景模式为透明
    pDC->TextOut(80,70,"透明字体 TRANSPARENT");         //输出文本 3
    pDC->SetTextAlign(TA_CENTER);                       //设置对齐方式为居中
    pDC->TextOut(80,90,"居中显示 TA_CENTER");           //输出文本 4
    pDC->SetTextAlign(TA_LEFT);                         //设置对齐方式为左对齐
    CRect rcText(90,120,150,180);                       //裁剪矩形
    pDC->Rectangle(rcText);                             //绘制矩形
    pDC->ExtTextOut(80,120,0,rcText,"矩形裁剪文本",NULL);   //在矩形内输出文本 5，不裁剪
    pDC->ExtTextOut(80,160,ETO_CLIPPED,rcText,"矩形裁剪文本",NULL);
                                                        //在矩形内输出文本 6，裁剪
    CRect rcDraw(90,200,200,260);
    pDC->Rectangle(rcDraw);                             //绘制范围矩形，在矩形内绘制多行文本
    pDC->DrawText("在矩形内部\n  使用换行符\\n\n  显示多行文字",rcDraw,DT_LEFT);
}
```

SetTextColor 函数设置文本前景色，SetBkColor 函数设置背景色，MoveTo、LineTo 函数用于绘制一段直线。TextOut 函数根据起点位置和对齐方式绘制文本，SetBkMode 函数设置背景模式，若为 TRANSPARENT 则不使用文本的背景色。SetTextAlign 函数设置文本与起点的对齐方式，若为 TA_CENTER 则表示文本的中部与起点对齐。

Rectangle 函数用于绘制矩形，ExtTextOut 函数在矩形内输出文本，若使用 ETO_CLIPPED 标志，则裁剪超出矩形范围的文本。DrawText 函数用于在矩形范围内输出文本，超出范围的被裁剪掉，可使用换行符\n 输出多行文本。

（2）生成程序并运行，如图 6-3 所示。前两行文本同时使用前景色和背景色，第 3 行“透明字体”开始使用透明背景模式，只有前景色，第 4 行设置居中对齐，第 5 行未设置裁剪标志，没有被矩形裁剪，第 6 行使用裁剪标志，只保留矩形范围内的文本，第 7 行在矩形范围内使用换行符\n 绘制了多行文本，且为左对齐。

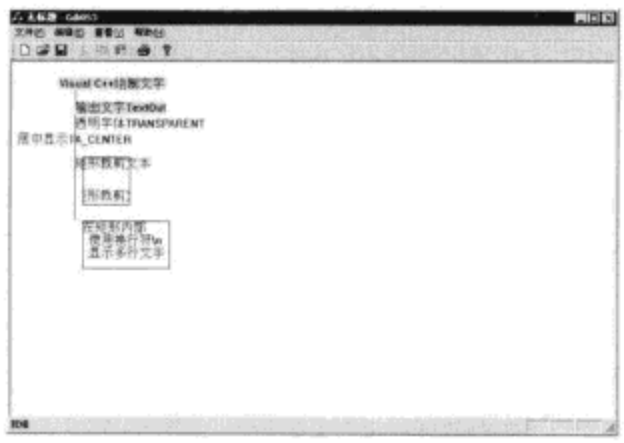


图 6-3　绘制文本

6.3 画笔

画笔是绘图的基本工具，常用于绘制线和面的边界，可通过设置画笔的线型、线宽、颜色，实现不同的绘制效果，也可使用系统自带的库存笔。一个设备环境同一时间只能有一种画笔，在使用完自定义的画笔后，应恢复原始画笔状态。

6.3.1 创建画笔

画笔也是一种系统资源，使用 HPEN 句柄唯一标识一个画笔，MFC 框架提供 CPen 类用来处理画笔的创建和释放。创建一个画笔有两种方式，第一种方式使用无参的构造函数，再调用 CreatePen 函数创建一个画笔实例，格式如下：

```
BOOL CPen::CreatePen(int nPenStyle,int nWidth,COLORREF cfColor)
```

参数如下。

- nPenStyle：画笔的风格，如 PS_SOLID 表示实线，PS_DASH 表示虚线，PS_DOT 表示点线。
- nWidth：画笔的宽度，其中虚线和点线的宽度不能超过 1，否则将变成实线。
- cfColor：画笔的 RGB 颜色值。

返回值：若成功则返回非零值，否则为 0。

第二种方式直接在构造函数中传入参数，创建画笔对象及其实例，参数列表同 CreatePen 函数，格式如下：

```
CPen::CPen(int nPenStyle,int nWidth,COLORREF crColor)
```

Windows 系统自带有一些库存笔，要使用库存笔，先使用无参构造函数定义一个 CPen 对象，再调用 CreateStockObject 函数创建一个库存笔，格式如下：

```
BOOL CGdiObject::CreateStockObject(int nIndex)
```

参数如下。

- nIndex：标准库存画笔，如 BLACK_PEN 为黑色画笔，WHITE_PEN 为白色画笔，NULL_PEN 为背景颜色画笔，相当于不绘制，在绘制多边形时，使用 NULL_PEN 作为画笔，可以不绘制边界。

返回值：若成功则返回非零值，否则为 0。

CGdiObject 类是各种基本 GDI 类如画笔、画刷、字体类的基类，提供了一些通用的功能，主要函数如下所示。

- GetSafeHandle：返回 GDI 对象的句柄值。
- FromHandle：根据 GDI 句柄值返回一个 CGdiObject 对象指针。
- Attach：将一个 GDI 句柄附加到一个 CGdiObject 对象上。
- Detach：解除并返回一个 CGdiObject 对象关联的 GDI 句柄。
- DeleteObject：释放 GDI 对象关联的句柄资源实例，若 GDI 对象正在使用则不能调用该函数。

一个设备环境一次只能使用一个画笔，若要使用某画笔，需要先调用 SelectObject 函数将其选入设备环境，同时保存旧画笔，当绘制结束后，再调用 SelectObject 函数将旧画笔重新选入设备环境，格式如下：

```
CPen* CDC::SelectObject(CPen* pPen)
```

参数如下。

- pPen：新画笔对象的指针。

返回值：旧画笔的指针。

画笔是一种系统资源，当使用完毕后，应释放占用的系统资源。当 CPen 对象被释放时，系统会自动调用 CGdiObject 基类的 DeleteObject 函数，释放其关联的 GDI 资源。但若用同一个 CPen 对象多次调用 CreatePen 函数创建多个画笔，系统仅释放对象关联的那个画笔，应在调用 CreatePen 函数创建另一个画笔之前，先调用 DeleteObject 函数释放当前关联的画笔资源。

6.3.2 使用画笔

在程序使用自定义画笔对象一般分为如下几个步骤：

（1）创建画笔对象，使用有参构造函数，或先使用无参构造函数，再调用 CreatePen 函数创建画笔。

（2）调用 SelectObject 函数将新画笔选入设备环境，同时保存旧画笔。

（3）利用新画笔绘制线或面的边界。

（4）再调用 SelectObject 函数将旧画笔选入设备环境，如有必要，调用 DeleteObject 函数释

放新画笔。

【实例 6-4】新建一个单文档工程名为 Gdi054，创建并使用画笔对象，用两种样式的画笔绘制两个矩形。

（1）新建单文档工程 Gdi054，在类视图右键单击 CGdi054View 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个 CPen 类型的成员变量 m_pen。

（2）在类视图双击 CGdi054View 类下的 OnDraw 项，添加如下代码：

```
void CGdi054View::OnDraw(CDC* pDC)
{
    CGdi054Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CPen* pOldPen=NULL;                                  //旧画笔指针
    if(m_pen.GetSafeHandle())                            //若画笔句柄实例存在
        m_pen.DeleteObject();                            //释放该画笔的 GDI 资源
    m_pen.CreatePen(PS_SOLID,3,RGB(0,0,255));            //创建画笔
    pOldPen=pDC->SelectObject(&m_pen);                   //将新画笔选入设备环境，同时保存旧画笔
    pDC->Rectangle(20,20,200,200);                       //用新画笔 1 绘制矩形
    pDC->SelectObject(pOldPen);                          //将旧画笔选入设备环境
    m_pen.DeleteObject();                                //释放新画笔的 GDI 资源
    m_pen.CreatePen(PS_DASH,1,RGB(255,0,0));             //再重新创建一个画笔
    pDC->SelectObject(&m_pen);                           //将新画笔 2 选入设备环境
    pDC->Rectangle(40,40,180,180);                       //用新画笔 2 绘制矩形
    pDC->SelectObject(pOldPen);                          //将旧画笔选入设备环境
}
```

GetSafeHandle 函数用于获取一个 GDI 对象关联的句柄值，格式如下：

```
HGDIOBJ CGdiObject::GetSafeHandle() const
```

返回值：GDI 对象关联的句柄值，若无关联句柄，则返回 NULL。

DeleteObject 函数用于释放 GDI 对象占用的 GDI 资源，有类成员和全局两种版本，格式如下：

```
BOOL CGdiObject::DeleteObject()
BOOL DeleteObject(HGDIOBJ hObject)
```

第一个为类成员版本，通过对象的方式调用。第二个为全局版本，传入 GDI 对象的句柄值，实际上 CGdiObject::DeleteObject 函数也是内部调用:: DeleteObject 函数实现 GDI 资源的释放。

由于 OnDraw 函数经常被调用，且 m_pen 对象持续有效，因此在调用 CreatePen 函数之前应判断画笔实例是否已经存在，若存在则调用 DeleteObject 函数释放该实例。

CreatePen 函数用于创建一个画笔实例，先创建一个实线、宽度为 3、蓝色的新画笔，调用 SelectObject 函数将新画笔选入当前设备环境，同时用 pOldPen 保存旧画笔的指针，一个设备环境某一时刻只能使用一个画笔。

Rectangle 函数用于绘制一个矩形，使用当前画笔绘制边界，使用当前画刷填充内部，若使用 NULL_PEN 样式的画笔则不绘制边界。在调用 DeleteObject 函数释放画笔之前，应停止使用该画笔，调用 SelectObject 函数将旧画笔选入设备环境，不能释放正在使用的画笔。

当用同一个 CPen 对象多次调用 CreatePen 函数创建多个画笔时，应在调用下一个 CreatePen 函数之前，先调用 DeleteObject 函数释放当前已有的 GDI 资源，否则会报错。

（3）生成程序并运行，如图 6-4 所示。外部的矩形使用实线、宽度为 3、蓝色的画笔，内部的矩形使用虚线、宽度为 1、红色的画笔。

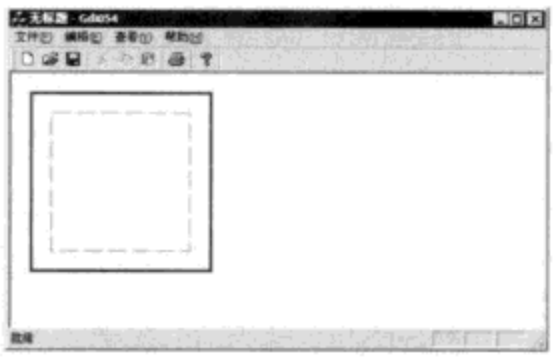


图 6-4　使用画笔

6.4 画刷

面的边界可用画笔绘制，内部区域需要使用画刷进行填充，与画笔类似，画刷可以设置填充的颜色，也可以使用图像作为填充图案，或使用库存的刷子。一个设备环境同一时间只能有一种画刷，在使用完自定义的画刷后，应恢复原始画刷状态。

6.4.1 创建画刷

画刷也是一种系统资源，使用 HBRUSH 句柄唯一标识一个画刷，MFC 框架提供 CBrush 类用于处理画刷的创建和释放。创建一个画刷有两种方式，第一种方式使用无参的构造函数，再调用一种创建函数创建一个画刷实例，有如下几种创建函数：

CreateSolidBrush 函数用于创建用一种颜色填充的画刷，格式如下：

```
BOOL CBrush::CreateSolidBrush(COLORREF crColor)
```

参数如下。

- crColor：画刷颜色的 RGB 值。

返回值：若成功则返回非零值，否则为 0。

CreateHatchBrush 函数用于创建阴影画刷，格式如下：

```
BOOL CBrush::CreateHatchBrush(int nIndex,COLORREF crColor)
```

参数如下。

- nIndex：阴影风格，如 HS_CROSS 表示水平线和垂直线相交的网格阴影，HS_BDIAGONAL 表示 45 度的从左到右的向下阴影，HS_DIAGCROSS 表示 45 度的网格阴影。
- crColor：画刷颜色的 RGB 值。

返回值：若成功则返回非零值，否则为 0。

CreatePatternBrush 函数用于将位图作为填充图案，格式如下：

```
BOOL CBrush::CreatePatternBrush(CBitmap* pBitmap)
```

参数如下。

- pBitmap：指向 CBitmap 对象的指针。

返回值：若成功则返回非零值，否则为 0。

CreateSysColorBrush 函数创建用系统颜色填充的画刷，格式如下：

```
BOOL CBrush::CreateSysColorBrush(int nIndex)
```

参数如下。

- nIndex：系统颜色的索引，以 COLOR_为前缀。

返回值：若成功则返回非零值，否则为 0。

颜色索引对应的颜色值可通过 GetSysColor 函数获取，格式如下：

```
DWORD GetSysColor(int nIndex)
```

参数如下。

- nIndex：系统颜色的索引，以 COLOR_为前缀。

返回值：对应的系统颜色值。

第二种方式直接在构造函数里传入参数，创建画刷及实例，有如下几种形式：

```
CBrush::CBrush(COLORREF crColor)
```

```
CBrush::CBrush(int nIndex,COLORREF crColor)
CBrush::CBrush(CBitmap* pBitmap)
```

第一种构造对应 CreateSolidBrush 函数，第二种构造对应 CreateHatchBrush 函数，第三种构造对应 CreatePatternBrush 函数。

Windows 系统自带有一些库存刷子，如表 6-1 所示。

表 6-1　Windows系统的库存刷子

WHITE_BRUSH	白色画刷	LTGRAY_BRUSH	浅灰色画刷
BLACK_BRUSH	黑色画刷	HOLLOW_BRUSH	空心画刷
GRAY_BRUSH	灰色画刷	NULL_BRUSH	空画刷
DKGRAY_BRUSH	深灰色画刷		

调用 CDC 类的 SelectStockObject 函数将库存刷子选入设备环境，格式如下：

```
CGdiObject* CDC::SelectStockObject(int nIndex)
```

参数如下。

- nIndex：库存对象的值。

返回值：先前的 GDI 对象的指针，实际为画笔、画刷、字体的一种。

类似于画笔对象，一个设备环境一次只能使用一个画刷，使用 SelectObject 函数将新画刷选入设备环境，同时保存旧画刷，绘制完成后，再调用 SelectObject 函数将旧画刷选入设备环境。当画刷对象被释放时，系统自动释放其关联的 GDI 资源，如有必要可调用 DeleteObject 手动释放 GDI 资源。若使用位图刷子，应在释放位图刷子对象后删除位图对象。

6.4.2　使用画刷

在程序使用自定义画刷对象一般分为如下几个步骤：

（1）创建画刷对象，使用有参构造函数，或先使用无参构造函数，再调用创建函数创建画刷。

（2）调用 SelectObject 函数将新画刷选入设备环境，同时保存旧画刷。

（3）利用新画刷填充多边形内部。

（4）再调用 SelectObject 函数将旧画刷选入设备环境，如有必要，调用 DeleteObject 函数释放新画刷。

【实例 6-5】新建一个单文档工程名为 Gdi055，创建并使用画刷对象，用位图画刷填充整个视图背景，用实心画刷填充椭圆。

（1）新建单文档工程 Gdi055，在类视图右键单击 CGdi055View 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加如下四个成员变量：

```
public:
    CBitmap m_bmp;          //位图对象
    CBrush m_br2;           //位图画刷
    CBrush m_br1;           //实心画刷
    CPen m_pen;             //画笔
```

CBitmap 类用于操作位图，位图也是一种系统资源，使用 HBITMAP 句柄唯一标识一个位图，创建一个位图对象需要两步，先调用 CBitmap 类的构造函数创建一个类对象，再调用初始化函数。

LoadBitmap 函数用于从程序的资源文件中加载位图来初始化对象，格式如下：

```
BOOL CBitmap::LoadBitmap(UINT nIDResource)
```

参数如下。

- nIDResource 位图资源的 ID。

返回值：若成功则返回非零值，否则为 0。

GetBimap 函数用于获取位图信息，存放到 BITMAP 结构体中，格式如下：

```
int CBitmap::GetBimap(BITMAP* pBitMap)
```

参数如下。

- pBitMap BITMAP：结构的指针，用于存放位图信息。

返回值：若成功则返回非零值，否则为 0。

BITMAP 结构存放了位图的所有信息，格式如下：

```
typedef struct tagBITMAP
{
    int    bmType;                  //位图类型
    int    bmWidth;                 //宽度
    int    bmHeight;                //高度
    int    bmWidthBytes;            //每个扫描行中字节的数目
    BYTE   bmPlanes;                //颜色平面的数目
    BYTE   bmBitsPixel;             //每个位平面的颜色位数
    LPVOID bmBits;                  //位值，指向单字节数组的指针
} BITMAP;
```

（2）在资源视图单击右键，在弹出的快捷菜单中选择 Import 命令，弹出 Import Resource 窗口，“文件类型”组合框选择“所有文件”项，选择一张 bmp 格式的图片，单击 Import 按钮导入位图文件，成功导入后资源视图出现 Bitmap 节点，节点下的位图 ID 默认为 IDB_BITMAP1。

（3）在类视图右键单击 CGdi055View 项，在弹出的快捷菜单中选择 Add Virtual Function 命令重写虚函数，选择 OnInitialUpdate 项，单击 Add and Edit 按钮，添加如下代码：

```
void CGdi055View::OnInitialUpdate()
{
    CView::OnInitialUpdate();
    // TODO: Add your specialized code here and/or call the base class
    m_pen.CreatePen(PS_SOLID,2,RGB(255,255,0));   //创建画笔
    m_br1.CreateSolidBrush(RGB(255,0,0));         //创建实心画刷
    m_bmp.LoadBitmap(IDB_BITMAP1);                //加载位图资源
    m_br2.CreatePatternBrush(&m_bmp);             //创建位图画刷
}
```

当视图第一次连接到文档但尚未显示时，框架调用 OnInitialUpdate 函数，可重写该函数，进行初始化操作，该函数功能类似于 CDialog 类的 OnInitDialog 函数。

CreatePen 函数创建一个实线、宽度为 2、黄色的画笔，CreateSolidBrush 函数创建一个红色的实心画刷，LoadBitmap 函数加载位图资源 IDB_BITMAP1，并关联到 m_bmp 对象，CreatePatternBrush 函数根据位图对象创建一个位图画刷。

（4）在类视图双击 CGdi055View 类下的 OnDraw 项，添加如下代码：

```
void CGdi055View::OnDraw(CDC* pDC)
{
    CGdi055Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CPen* pOldPen=NULL;                          //旧画笔指针
    CBrush* pOldBrush=NULL;                      //旧画刷指针
    CRect rcClient;
    GetClientRect(rcClient);                     //客户区的矩形大小
    pDC->FillRect(rcClient,&m_br2);              //用位图画刷填充客户区矩形
    pOldBrush=pDC->SelectObject(&m_br1);         //将画刷 1 选入设备环境，同时保存旧画刷
    pOldPen=(CPen*)pDC->SelectStockObject(NULL_PEN);
                                                 //将库存的空笔选入设备环境，保存旧画笔
```

```
    CRect rcRound(500,30,570,100);
    pDC->Ellipse(rcRound);                          //用画刷 1 和空笔绘制椭圆
    pDC->SelectObject(pOldBrush);                   //将旧画刷选入设备环境
    pDC->SelectObject(&m_pen);                      //将新画笔选入设备环境
    int cx=rcRound.CenterPoint().x;                 //绘制的椭圆的中心 x、y
    int cy=rcRound.CenterPoint().y;
    pDC->MoveTo(cx-60,cy);                          //绘制三段直线
    pDC->LineTo(cx-90,cy);
    pDC->MoveTo(cx-(int)30*sqrt(2),cy+(int)30*sqrt(2));
    pDC->LineTo(cx-(int)45*sqrt(2),cy+(int)45*sqrt(2));
    pDC->MoveTo(cx,cy+60);
    pDC->LineTo(cx,cy+90);
    pDC->SetTextColor(RGB(0,255,0));                //设置文本颜色
    pDC->SetBkMode(TRANSPARENT);                    //设置背景模式，透明
    pDC->TextOut(rcClient.CenterPoint().x,rcClient.top+20,"绿色草原");  //输出文本
    pDC->SelectObject(pOldPen);                     //将旧画笔选入设备环境
}
```

GetClientRect 函数获取客户区的矩形大小，FillRect 函数用指定画刷填充矩形，格式如下：

```
void CDC::FillRect(LPCRECT lpRect,CBrush* pBrush)
```

参数如下。

- lpRect：被填充的矩形对象。
- pBrush：用于填充的画刷的指针。

若用指定颜色填充矩形可用 FillSolidRect 函数，格式如下：

```
void CDC::FillSolidRect(LPCRECT lpRect,COLORREF clr)
```

参数如下。

- lpRect：被填充的矩形对象。
- clr：填充颜色的 RGB 值。

SelectStockObject 函数将系统自带的 NULL_PEN 选入设备环境，NULL_PEN 使用背景颜色绘图，相当于什么都不绘制，该函数返回 CGdiObject 类型的指针，需要强制转换为需要的 CPen 类型指针，存入 pOldPen 中。Ellipse 函数根据当前的画笔和画刷绘制椭圆，由于画笔为 NULL_PEN，不绘制边界。

cx、cy 记录绘制椭圆的中心点坐标，连续调用 MoveTo、LineTo 函数绘制三段直线，其中 sqrt 函数用于计算一个数的平方根，该函数在 cmath 头文件中声明，需要在 Gdi055View.cpp 文件的开头添加一句#include <cmath>，包含 cmath 头文件。

SetTextColor 函数设置文本的颜色为绿色，SetBkMode 函数设置背景模式为透明，TextOut 在屏幕中心输出一段文本。

（5）生成程序并运行，如图 6-5 所示。客户区背景由 FillRect 函数使用位图画刷填充完成，右上角的圆由 Ellipse 函数绘制完成，圆左下方的三根直线由 MoveTo、LineTo 函数绘制完成，中间上方的文本由 TextOut 函数绘制完成。

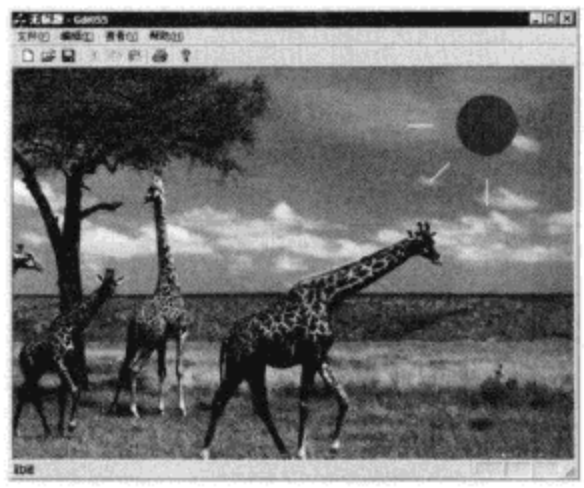


图 6-5　使用画刷

6.5 字体

在使用文本编辑器时，要经常设置不同段落的字体样式，如标题、章节、正文等需要用不同的字体样式加以区分。TextOut 等函数根据当前设备环境的字体样式输出文本，若未设置则使用默认的字体。同画笔、画刷，一个设备环境同一时间只能有一种字体，在使用完自定义的字体后，应恢复原始字体状态。

6.5.1 创建字体

字体也是一种系统资源，使用 HFONT 句柄唯一标识一个字体，MFC 框架提供 CFont 类用于处理字体的创建和释放。创建一个字体先调用构造函数生成一个 CFont 对象，再调用一种创建函数创建一个字体实例，有如下几种创建函数：

CreatePointFont 函数用于创建一个简单的字体，格式如下：

```
BOOL CFont::CreatePointFont(int nPointSize,LPCTSTR lpszFaceName,CDC* pDC = NULL)
```

参数如下。

- nPointSize：字体高度，单位是 0.1 点，如 360 表示 36 点。
- lpszFaceName：字体名称。
- pDC：CDC 对象的指针，自动将 nPointSize 指定的高度转为逻辑单位。

返回值：若成功则返回非零值，否则为 0。

CreateFont 函数用于创建一个复杂的字体，能够满足专业需求，格式如下：

```
BOOL CFont::CreateFont(int nHeight,int nWidth,int nEscapement,int nOrientation,int
nWeight,BYTE bItalic,BYTE bUnderline,BYTE cStrikeOut,BYTE nCharset,BYTE
nOutPrecision,BYTE nClipPrecision,BYTE nQuality,BYTE nPitchAndFamily,LPCTSTR
lpszFacename)
```

参数如下。

- nHeight：字体的高度，若为 0 则使用默认值。
- nWidth：字符平均宽度。
- nEscapement：文本行相对于 *X* 轴的倾斜角度，单位为 0.1 度，如 900 表示 90 度。
- nOrientation：字符本身相对于 *X* 轴的倾斜角度，单位为 0.1 度。
- nWeight：字体磅数，范围为 0～1000，以 FW_为前缀。
- bItalic：是否使用斜体。
- bUnderline：是否使用下画线。
- bStrikeOut：是否使用删除线。
- nCharSet：字体使用的字符集，常用 ANSI_CHARSET 标准字符集。
- nOutPrecision：字体输出精度，常用 OUT_DEFAULT_PRECIS，若使用 TrueType 字体，应设为 OUT_TT_PRECIS。
- nClipPrecision：字体裁剪精度，常用 CLIP_DEFAULT_PRECIS。
- nQuality：字体输出质量，有 DEFAULT_QUALITY、DRAFT_QUALITY、PROOF_QUALITY 三种。
- nPitchAndFamily：字体间距和属性两个标志，将多个标志参数用+或|联合起来。
- lpszFacename：为当前字体设定一个名称。

返回值：若成功返回非零值，否则为 0。

6.5.2 使用字体

在程序使用自定义字体对象一般分为如下几个步骤：

（1）创建字体对象，先使用构造函数，再调用创建函数创建字体。

（2）调用 SelectObject 函数将新字体选入设备环境，同时保存旧字体。

（3）利用新字体绘制文本。

（4）再调用 SelectObject 函数将旧字体选入设备环境，如有必要，调用 DeleteObject 函数释放新字体。

【实例 6-6】新建一个单文档工程名为 Gdi056，创建并使用字体对象，设置不同的字体样式，并在窗口中输出文本。

（1）新建单文档工程 Gdi056，在类视图右键单击 CGdi056View 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加如下三个成员变量：

```
public:
    CFont m_font1;   //三个字体
    CFont m_font2;
    CFont m_font3;
```

（2）在类视图双击 CGdi056View 类下的 OnDraw 项，添加如下代码：

```
void CGdi056View::OnDraw(CDC* pDC)
{
    CGdi056Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    if(!m_font1.GetSafeHandle())                          //若字体 1 实例不存在，则创建字体 1
        m_font1.CreatePointFont(360,"宋体",pDC);
    if(!m_font2.GetSafeHandle())                          //若字体 2 实例不存在，则创建字体 2
        m_font2.CreateFont(50,0,0,0,FW_BOLD,TRUE,TRUE,FALSE,ANSI_CHARSET,
            OUT_DEFAULT_PRECIS,OUT_DEFAULT_PRECIS,PROOF_QUALITY,
            DEFAULT_PITCH|FF_DONTCARE,"custom font2");
    if(!m_font3.GetSafeHandle())                          //若字体 3 实例不存在，则创建字体 3
    m_font3.CreateFont(30,0,2700,2700,FW_MEDIUM,FALSE,FALSE,FALSE,ANSI_CHARSET,
            OUT_DEFAULT_PRECIS,OUT_DEFAULT_PRECIS,PROOF_QUALITY,
            DEFAULT_PITCH|FF_DONTCARE,"custom font3");
    CFont* pOldFont=NULL;
    pOldFont=pDC->SelectObject(&m_font1);                 //将字体 1 选入设备环境，同时保存旧字体
    pDC->TextOut(20,20,"Visual C++开发");                 //用字体 1 输出文本
    pDC->SelectObject(&m_font2);                          //将字体 2 选入设备环境
    pDC->TextOut(40,80,"CObject -> CCmdTarget -> CWnd");
    pDC->SelectObject(&m_font3);                          //将字体 3 选入设备环境
    pDC->TextOut(160,180,"软件工程师");
    pDC->SelectObject(pOldFont);                          //将原有字体选入设备环境
}
```

GetSafeHandle 函数获取字体对象的 GDI 句柄，若句柄不存在，则调用 CreatePointFont 函数创建一个 36 点、宋体的字体 1，调用 CreateFont 函数创建字体 2 和字体 3，其中字体 2 使用斜体、下画线，字体 3 旋转 270 度。SelectObject 函数将字体选入设备环境，TextOut 函数根据当前的字体样式输出文本。

（3）生成程序并运行，如图 6-6 所示。

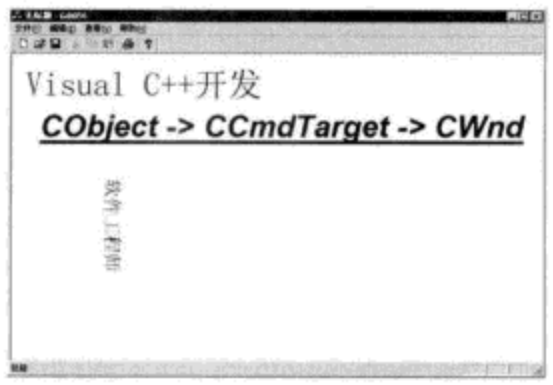


图 6-6　使用字体

6.6 映射模式

默认状态下绘图的单位是像素，像素的范围由屏幕分辨率决定，不同的屏幕分辨率下绘制的图形长度是不一致的，这会导致图形变形。设备环境提供了映射模式，可用实际长度如英寸、毫米替代像素作为单位，在不同的分辨率下，图形保持固定的长度。

6.6.1 了解映射模式

在映射模式中有逻辑单位（Logic）和设备单位（Device），逻辑单位是传入绘制函数的数值，其具体绘制长度与当前映射模式有关，设备单位是一个像素。

默认状态下映射模式为 MM_TEXT，即一个逻辑单位对应一个像素。设备环境提供多种映射模式，用于设置逻辑单位的具体值，所有可用的映射模式如表 6-2 所示。

表 6-2 各种映射模式中一个逻辑单位的值

MM_TEXT	一个像素	MM_HIMETRIC	0.01 毫米
MM_LOMETRIC	0.1 毫米	MM_HIENGLISH	0.001 英寸
MM_TWIPS	1/1440 英寸	MM_ISOTROPIC	自定义，X、Y 方向一致
MM_LOENGLISH	0.01 英寸	MM_ANISOTROPIC	自定义

若使用 MM_TEXT 映射模式，则在客户区窗口中，x 从左到右、y 从上到下递增。若使用其他映射模式，则 x 从左到右、y 从下到上递增，即 y 轴方向与 MM_TEXT 模式相反。如在 MM_TEXT 模式下，左上角点坐标为(0,0)，左下角点坐标可能为(0,444)，若改用 MM_HIENGLISH 模式，则左上角点坐标仍为(0,0)，但左下角点坐标已经变化为(0,–5463)。

SetMapMode 函数设置设备环境的映射模式，格式如下：

```
int CDC::SetMapMode(int nMapMode)
```

参数如下。

- nMapMode：新的映射模式，如 MM_HIENGLISH。

返回值：先前的映射模式。

DptoLP 函数根据映射模式，将设备坐标转换为逻辑坐标，格式如下：

```
void CDC::DptoLP(LPPOINT lpPoints, int nCount = 1) const
void CDC::DptoLP(LPRECT lpRect) const
void CDC::DptoLP(LPSIZE lpSize) const
```

参数如下。

- lpPoints：指向 POINT 结构的数组或 CPoint 对象的指针。
- nCount：数组中的点数目。
- lpRect：CRect 对象。
- lpSize：CSize 对象。

LPtoDP 函数根据当前映射模式，将逻辑坐标转换为设备坐标，格式类似于 DptoLP 函数。如某点的设备坐标值为(1,1)，调用 DptoLP 函数可得到该点的逻辑坐标值，代码如下：

```
CPoint pt(1,1);
pDC->SetMapMode(MM_LOMETRIC);
pDC->DPtoLP(&pt);
```

设备坐标点(1,1)在客户区窗口的左上角(0,0)的右下方位置，SetMapMode 函数设置设备环境的映射模式为 MM_LOMETRIC，一个逻辑单位相当于 0.1mm，若在绘制函数里传入 20，实际值为 20 乘以 0.1mm 即 2mm，不管屏幕分辨率多大，根据 2mm 实际长度完成绘制操作。

调用 DptoLP 函数进行转换时，步骤大致如下：

（1）获取(0,0)到(1,1)的 x、y 方向的实际长度（可用直尺量取）。

（2）根据映射模式，用实际长度（单位为毫米）除以 0.1，得到逻辑单位的长度值，如 x、y 方向的逻辑长度都为 3。

（3）由于在 MM_LOMETRIC 映射模式下，y 从下到上递增，客户区左上角的逻辑坐标为(0,0)，所以设备坐标点(1,1)的逻辑坐标为(3,–3)。

【实例 6-7】新建一个单文档工程名为 Gdi057，在三种映射模式下，分别绘制 x 方向上逻辑坐标值从 20 到 400 的水平直线。

（1）新建单文档工程 Gdi057，在类视图双击 CGdi057View 类下的 OnDraw 项，添加如下代码：

```
void CGdi057View::OnDraw(CDC* pDC)
{
    CGdi057Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    int y=80;                                          //水平直线的 y 值
    pDC->SetMapMode(MM_TEXT);                          //MM_TEXT 模式
    pDC->TextOut(20,y-40,"MM_TEXT    1pixel");         //输出文本
    pDC->MoveTo(20,y);                                 //设置起点，x 逻辑坐标为 20
    pDC->LineTo(400,y);                                //绘制直线，终点 x 逻辑坐标为 400
    pDC->SetMapMode(MM_LOMETRIC);                      // MM_LOMETRIC 模式
    CPoint ptStart(20,200);
    pDC->DPtoLP(&ptStart);                             //将设备坐标下的 200 转换为逻辑坐标
    y=ptStart.y;                                       //获取 y 逻辑坐标
    pDC->TextOut(20,y+200,"MM_LOMETRIC    0.1mm");//输出文本
    pDC->MoveTo(20,y);                                 //设置起点，x 逻辑坐标为 20
    pDC->LineTo(400,y);                                //终点 x 逻辑坐标为 400
    pDC->SetMapMode(MM_HIMETRIC);                      //MM_HIMETRIC 模式
    CPoint ptStart2(20,280);                           //将设备坐标下的 280 转换为逻辑坐标
    pDC->DPtoLP(&ptStart2);
    y=ptStart2.y;
    pDC->TextOut(20,y+1000,"MM_HIMETRIC    0.01mm");
    pDC->MoveTo(20,y);
    pDC->LineTo(400,y);
}
```

SetMapMode 函数设置当前的映射模式，TextOut 函数用于输出一行文本，MoveTo、LineTo 函数用于绘制一段直线。在不同的映射模式下，相同逻辑长度的直线的实际绘制长度是不同的。

在 MM_TEXT 模式下，一个逻辑单位相当于一个像素，当屏幕分辨率改变后，该模式下的实际直线长度是变化的，如将屏幕分辨率从 1024×768 调整到 1400×1050 后，用直尺测量直线长度，可发现分辨率增大后，直线长度变短。

在 MM_LOMETRIC 模式下，一个逻辑单位相当于 0.1mm，调用 DPtoLP 函数将点 ptStart 的设备坐标值转换为逻辑坐标值，其中 y 存放转换后的 *Y* 坐标值。该模式下的实际直线长度与屏幕分辨率无关，始终显示固定长度。

MM_HIMETRIC 模式类似 MM_LOMETRIC，在 MM_HIMETRIC 模式下，一个逻辑单位相当于 0.01mm，同样是 *x* 方向上逻辑坐标从 20 到 400 的水平直线，MM_HIMETRIC 模式下的直线长度只是 MM_LOMETRIC 模式直线长度的 1/10。

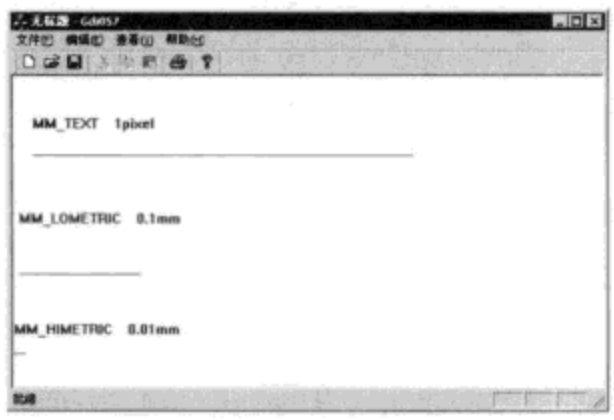


图 6-7　使用映射模式

（2）生成程序并运行，如图 6-7 所示。在三种映射模式下，逻辑长度相同的直线的实际长度是不同的，其中 MM_TEXT 模式下的直线长度与屏幕分辨率相关，另外两种模式在不同分辨率下的长度是固定的。

6.6.2　窗口和视口

在 GDI 绘图中，窗口（Window）和视口（Viewport）是两个不同的概念，窗口代表实际的图形画布，如一幅大尺寸的壁画，视口代表图形显示窗口，如程序的客户区窗口。视口可显示全部图形，也可只显示部分图形，视口有一定尺寸，图形也有一定尺寸，在视口中显示图形，视口和图形必定有一个比例映射关系。

设备环境提供的几种映射模式，如 MM_LOMETRIC、MM_HIMETRIC，视口和图形的比例是设定好的。若要自定义比例，可使用 MM_ISOTROPIC 和 MM_ANISOTROPIC 模式，两种模式的区别在于 MM_ISOTROPIC 在 x 方向和 y 方向上的比例相同，而 MM_ANISOTROPIC 在两个方向的比例可以不同。

使用 SetWindowExt 和 SetViewportExt 两个函数可以设置窗口和视口的比例，即逻辑坐标和设备坐标的比例，用两个函数设置的 x 方向和 y 方向的对应值相除，得到两个方向上的比例。

SetWindowExt 函数设置窗口的逻辑宽度和高度（逻辑单位），格式如下：

```
CSize CDC::SetWindowExt(int cx,int cy)
```

参数如下。

- cx：窗口的逻辑宽度。
- cy：窗口的逻辑高度。

返回值：先前的逻辑宽度和高度值。

SetViewportExt 函数设置视口的设备宽度和高度（设备单位），格式如下：

```
CSize CDC::SetViewportExt(int cx ,int cy)
```

参数如下。

- cx：视口的设备宽度。
- cy：视口的设备高度。

返回值：先前的设备宽度和高度值。

默认状态下视口的坐标原点位于左上角(0,0)，SetViewportOrg 函数可用于修改视口的坐标原点（设备单位），格式如下：

```
CPoint CDC::SetViewportOrg(int x,int y)
```

参数如下。

- x：新坐标原点的 x 值。
- y：新坐标原点的 y 值。

返回值：先前的坐标原点。

SetWindowOrg 函数用于修改窗口的坐标原点（逻辑单位），格式如下：

```
CPoint CDC::SetWindowOrg(int x,int y)
```

参数如下。

- x：新窗口原点的 x 值。
- y：新窗口原点的 y 值。

返回值：先前的窗口原点。

【实例 6-8】新建一个单文档工程名为 Gdi058，在 MM_ANISOTROPIC 模式下，设置窗口和视口的两种不同的比例，并和 MM_TEXT 模式做比较，分别改变视口和窗口的坐标原点，绘制多条直线对比显示。

（1）新建单文档工程 Gdi058，在类视图双击 CGdi058View 类下的 OnDraw 项，添加如下代码：

```
void CGdi058View::OnDraw(CDC* pDC)
{
    CGdi058Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    pDC->SetMapMode(MM_TEXT);                    //MM_TEXT 模式
    int x=20;
    int y=120;
    pDC->MoveTo(x,y);                            //绘制 x 从 20 到 400 的水平直线，记为线 1
```

```
    pDC->LineTo(400,y);
    CRect rcClient;
    GetClientRect(rcClient);                          //获取客户区矩形大小
    pDC->SetMapMode(MM_ANISOTROPIC);                  //MM_ANISOTROPIC 模式
    pDC->SetViewportExt(rcClient.right,rcClient.bottom);
                                                      //设置视口的设备宽度、高度
    pDC->SetWindowExt(rcClient.Width(),rcClient.Height()); //设置窗口的逻辑宽度、高度
    CPoint ptStart(20,160);
    pDC->DPtoLP(&ptStart);                            //将设备坐标转换为逻辑坐标
    x=ptStart.x;
    y=ptStart.y;
    pDC->MoveTo(x,y);                                 //在当前比例下绘制水平直线，记为线 2
    pDC->LineTo(400,y);
    pDC->SetViewportExt(rcClient.right/2,rcClient.bottom);
                                                      //重设视口的设备宽度、高度
    pDC->SetWindowExt(rcClient.Width(),rcClient.Height());
                                                      //重设窗口的逻辑宽度、高度
    CPoint ptStart2(20,200);
    pDC->DPtoLP(&ptStart2);                           //新比例下坐标转换
    x=ptStart2.x;
    y=ptStart2.y;
    pDC->MoveTo(x,y);                                 //新比例下绘制水平直线，记为线 3
    pDC->LineTo(400,y);
    pDC->SetMapMode(MM_TEXT);                         //切换为 MM_TEXT 模式
    pDC->SetViewportOrg(30,30);                       //设置视口坐标原点
    pDC->MoveTo(0,0);                                 //绘制从(0,0)到(30,30)的直线，记为线 4
    pDC->LineTo(30,30);
    pDC->SetViewportOrg(0,0);                         //重设视口的坐标原点
    pDC->MoveTo(0,0);                                 //绘制从(0,0)到(25,25)的直线，记为线 5
    pDC->LineTo(25,25);
    pDC->SetMapMode(MM_LOMETRIC);                     //切换为 MM_LOMETRIC 模式
    pDC->MoveTo(200,-30);                             //绘制直线，记为线 6
    pDC->LineTo(800,-150);
    pDC->SetWindowOrg(0,50);                          //设置窗口的坐标原点
    pDC->MoveTo(200,-30);                             //绘制逻辑值相同的直线，记为线 7
    pDC->LineTo(800,-150);
    pDC->SetWindowOrg(0,100);                         //重设窗口的坐标原点
    pDC->MoveTo(200,-30);                             //绘制逻辑值相同的直线，记为线 8
    pDC->LineTo(800,-150);
}
```

SetMapMode 函数设置当前使用的映射模式，在 MM_TEXT 模式下，绘制一段 x 从 20 到 400 的水平直线 1，用于和下面绘制的直线 2、3 做比较。

在 MM_ANISOTROPIC 模式下，x 方向和 y 方向的比例可以不同，调用 SetViewportExt 函数设置视口的设备宽度和高度，调用 SetWindowExt 函数设置窗口的逻辑宽度和高度，用逻辑宽度除以设备宽度得到 x 方向的比例，用逻辑高度除以设备高度得到 y 方向的比例。

由于 SetViewportExt 函数中的宽度 rcClient.right，等于 SetWindowExt 函数中的宽度 rcClient.Width()，SetViewportExt 函数中的高度 rcClient.bottom，等于 SetWindowExt 函数中的高度 rcClient.Height()，所以在 x 方向和 y 方向的比例都是 1：1，即一个逻辑单位等于一个像素，绘制效果等同于 MM_TEXT。

第二次调用 SetViewportExt、SetWindowExt 函数，重新设置比例时，由于 SetViewportExt 函数中的宽度为 rcClient.right/2，其他保持不变，在 x 方向上一个逻辑单位相当于 0.5 个像素，y 方向上保持不变。当调用 DPtoLP 函数，将点 ptStart2(20,200)转换为逻辑坐标后，ptStart2 的逻辑坐标为(40,200)。

在新比例下，绘制相同的逻辑坐标从(40,200)到(400,200)的水平直线，其对应的设备坐标为从(20,200)到(200,200)，实际绘制长度是直线 2 的一半。

调用 SetMapMode 函数重设当前模式为 MM_TEXT，SetViewportOrg 函数设置视口的新坐标原点为(30,30)，默认为客户区窗口的左上角(0,0)处。在新坐标原点下，MoveTo、LineTo 函数绘制从(0,0)到(30,30)的直线 4，若不修改坐标原点，直线 4 应该从左上角出发，修改坐标原点后，从(30,30)处开始绘制，相当于一条从(0,0)开始的直线，经过平移后，起点变为(30,30)。第二次调用 SetViewportOrg 函数，恢复坐标原点为左上角(0,0)，绘制从(0,0)到(25,25)的直线 5，这次直线从左上角开始出发。

Tips 关于 SetViewportOrg 函数修改视口坐标原点，可理解为先按函数中的参数值进行绘制，如从(0,0)到(30,30)，不管坐标原点在哪，从左上角(0,0)开始到(30,30)处绘制一条直线，再将直线从坐标原点(0,0)处整体平移到实际坐标原点处。SetViewportOrg 函数中的新坐标原点值，不受当前坐标原点影响。

在 MM_LOMETRIC 模式下，先在默认窗口坐标原点下，绘制直线 6，默认的逻辑坐标原点也在左上角(0,0)处，由于 y 从下往上递增，直线 6 的 y 值是负值。调用 SetWindowOrg 函数修改窗口的逻辑坐标原点后，类似 SetViewportOrg 函数，先根据参数值绘制直线，再根据逻辑坐标原点进行平移。通过对比直线 6、7、8，可发现相同的绘制操作，在不同的逻辑坐标原点下显示不同的图形。

Tips 视口和窗口是两个不同的窗口，有各自的坐标系，窗口中的图形通过逻辑坐标原点和设备坐标原点，映射到视口中，图形的实际尺寸根据当前的映射模式所确定。在实际应用中，应避免同时使用 SetViewportOrg 和 SetWindowOrg 函数设置坐标原点，否则会造成混乱，难以理解。

（2）生成程序并运行，如图 6-8 所示。直线 1 使用 MM_TEXT 模式，直线 2、3 使用 MM_ANISOTROPIC 模式并设置不同的比例，直线 4、5 使用 MM_TEXT 模式并调用 SetViewportOrg 函数修改设备坐标原点，直线 6、7、8 使用 MM_LOMETRIC 模式并调用 SetWindowOrg 函数修改逻辑坐标原点。

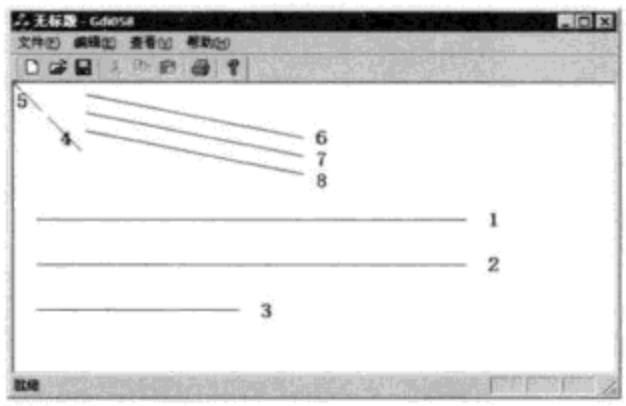


图 6-8 窗口和视口

6.7 小结

在 Windows API 中，提供了一系列绘图函数，用于实现基本图形的绘制。本章主要介绍了 GDI 图形的编程。先介绍了设备环境，接着介绍了基本图形的绘制，然后对于画笔和画刷的使用进行了详细的介绍。对于每一节，都配有具体的实例，让读者能充分理解本章知识。

6.8 习题

1. 什么是 Windows 图形设备接口 GDI?
2. 如何获取设备环境?
3. 如何创建画刷、画笔等 GDI 对象?
4. 如何创建字体?
5. 编写一个实例程序，实现绘制直线、曲线、圆形的功能。

第 7 章　单文档应用程序

单文档是一种常见的文件处理程序，如 Windows 自带的画图、记事本等。相对于对话框简单的拖放控件，单文档程序提供强大的文件读取显示功能，其核心为文档/视图结构，文档用于读取、保存文件中的数据，视图用于数据的显示、编辑，在该框架基础上，添加需要的功能，可快速实现强大的文件处理显示功能。

7.1 了解生成类

使用 MFC 应用程序向导创建一个单文档程序后，自动生成 5 个类，其中 CMainFrame、App、Doc、View 四个类是单文档程序的基础支架，CMainFrame 类负责程序窗口的框架显示，包括菜单、工具栏、状态栏等元素，App 类负责程序的启动和退出，Doc 类负责文件的读取和保存，View 类负责数据的显示、编辑，以及与用户的交互操作等，了解这几个基本类有助于快速掌握单文档程序的开发。

7.1.1 App 类

【实例 7-1】新建一个单文档工程名为 Sdi061，了解 AppWizard 自动生成的几个类。

类似于对话框中的 App 类，该类主要负责程序的启动和退出，一般程序级别的操作可在该类中完成，如限制只能运行一个程序实例，一个运行的程序为一个实例，可同时运行多个程序，即多个程序实例。

程序实例初始化时调用 InitInstance 函数，退出时调用 ExitInstance 函数，可为这两个函数添加相应功能。在 InitInstance 函数中，使用文档模板类将 Frame、Doc、View 和资源组合到一起，代码如下：

```
CSingleDocTemplate* pDocTemplate;          //文档模板
pDocTemplate = new CSingleDocTemplate(
  IDR_MAINFRAME,                           //资源 ID
  RUNTIME_CLASS(CSdi061Doc),               //文档类
  RUNTIME_CLASS(CMainFrame),               //框架类
  RUNTIME_CLASS(CSdi061View));             //视图类
AddDocTemplate(pDocTemplate);              //加入到文档模板列表
```

一个程序可以有多个文档模板，单文档程序只有一个文档模板，多文档程序可以有多个文档模板，CDocManager 类用于管理文档模板列表，CWinApp 类内置了 CDocManager 类成员，并提供三个函数用于添加、访问文档模板。

AddDocTemplate 函数用于添加一个文档模板，格式如下：

```
void CWinApp::AddDocTemplate(CDocTemplate* pTemplate)
```

参数如下。

- pTemplate CdocTemplate：类或派生类对象的指针。

GetFirstDocTemplatePosition 函数获取程序第一个文档模板的位置，格式如下：

```
POSITION CWinApp::GetFirstDocTemplatePosition() const
```

返回值：第一个文档模板的 POSITION 值。

GetNextDocTemplate 函数获取当前位置的文档模板，用于循环获取所有文档模板，格式如下：

```
CDocTemplate* CWinApp::GetNextDocTemplate(POSITION& pos) const
```

参数如下。

- pos：当前文档模板的 POSITION 值引用，调用该函数后，该值指向下一个位置。

返回值：当前位置的文档模板的指针。

> **Tips** 这三个函数也是通过内部调用 CDocManager 类的三个同名函数，完成文档模板的添加、访问。POSITION 类型实际上就是一个空结构体类型的指针，指向一个位置，常用于循环获取所有元素，类似于 STL 中的迭代器。

CSingleDocTemplate 类用于实现单文档界面，管理文档、视图、框架三个类，合作实现文档/视图结构，构造函数如下：

```
CSingleDocTemplate::CSingleDocTemplate(UINT nIDResource,CRuntimeClass* pDocClass,
CRuntimeClass * pFrameClass,CRuntimeClass* pViewClass)
```

参数如下。

- nIDResource：文档模板使用的资源 ID，可包括图标、菜单、快捷键、字符串等，不同类别的资源使用相同的 ID。
- pDocClass：文档类的 CRuntimeClass 对象指针。
- pFrameClass：框架类的 CRuntimeClass 对象指针。
- pViewClass：视图类的 CRuntimeClass 对象指针。

RUNTIME_CLASS 宏用于获取类的运行时结构信息，返回一个指向 CRuntimeClass 结构的指针。只有从 CObject 类派生，并使用 DECLARE_DYNAMIC、DECLARE_ DYNCREATE、DECLARE_SERIAL 其中一个宏的类，才能使用 RUNTIME_CLASS 宏。

CRuntimeClass 结构用于在运行时获取类的结构信息，并可动态创建一个类对象，如 m_lpszClassName 成员获取类名，CreateObject 函数用于运行时动态创建一个类对象，文档、视图、框架类都支持动态创建。

> **Tips** 在 C++中，结构体 struct 也可有成员函数，默认为公有成员。

CSingleDocTemplate 类将文档、视图、框架类组合起来，利用 RUNTIME_CLASS 宏在运行时动态创建三个类的对象，AddDocTemplate 函数将文档模板加入到列表中，在单文档程序中，文档模板列表的元素数目为 1。

7.1.2 Doc 类

文档类负责文件的读取和存储，一个文档可以与多个视图关联，如有一组数据，在视图 1 中用表格形式显示，在视图 2 中用折线图形式显示，CDocument 类提供两个函数用于循环访问其关联的视图类。

GetFirstViewPosition 函数获取与文档关联的视图列表的第一个视图，格式如下：

```
POSITION CDocument::GetFirstViewPosition() const
```

返回值：第一个视图的位置。

GetNextView 函数获取当前位置视图的指针，格式如下：

```
CView* CDocument::GetNextView(POSITION& rPosition) const
```

参数如下。

- rPosition：当前视图的 POSITION 值引用，调用该函数后，该值指向下一个位置。

Tips 默认情况下，单文档程序只有一个文档、一个视图，但可以通过切换视图方法，实现多视图显示。单文档只能有一个文档对象，且仅创建一次。

当文档内容更新后，文档所关联的视图也应同步更新，UpdateAllViews 函数用于刷新关联的所有视图，格式如下：

```
void CDocument::UpdateAllViews(CView* pSender,LPARAM lHint=0L,CObject* pHint=NULL)
```

参数如下。

- pSender：要更新的视图的指针，若为 NULL 则更新所有视图。
- lHint：包含修改的信息。
- pHint：指向存放修改信息对象的指针。

文档/视图的功能很强大，但也很复杂，MFC 框架在后台完成了大量工作，仅通过 AppWizard 自动生成的一些代码，是无法了解文档/视图如何实现的，有必要阅读 MFC 源代码，简要地了解工作流程，对于自身能力的提升非常有帮助。

Tips MFC 框架的源代码是公开的，MFC 的函数功能及调用流程，在源代码中得到完全体现，若不确定某个函数的内部实现，可打开源代码，找到函数定义处，阅读后便可柳暗花明。源代码路径为：...\Microsoft Visual Studio\VC98\MFC\SRC。

（1）打开 Visual C++ 安装目录，定位到 VC98\MFC\SRC 处，用鼠标右键单击 SRC 文件夹，在弹出的快捷菜单中选择“搜索”命令，弹出“搜索结果”窗口，在“文件中的一个字或词组”编辑框中输入 OpenDocumentFile，单击“搜索”按钮，如图 7-1 所示。

图 7-1　搜索源代码

搜索完毕后，出现多个 CPP 文件，说明这些文件中都调用了 OpenDocumentFile 函数，若不知道在哪个文件中查找，可逐个打开文件查看。

（2）用记事本打开 DOCSINGL.CPP 文件，按 Ctrl+F 组合键，打开“查找”窗口，输入 OpenDocumentFile 函数，定位到函数定义处，部分代码如下：

```
CDocument* CSingleDocTemplate::OpenDocumentFile(LPCTSTR lpszPathName,BOOL bMakeVisible)
{
    CDocument* pDocument = NULL;                    //文档
    CFrameWnd* pFrame = NULL;                       //框架
    BOOL bCreated = FALSE;                          //是否已创建
    BOOL bWasModified = FALSE;                      //文档是否被修改
    if (m_pOnlyDoc != NULL)                         //已经存在一个文档
    {
        pDocument = m_pOnlyDoc;
        if (!pDocument->SaveModified())             //若尚未保存修改，直接返回
```

```
            return NULL;
        pFrame = (CFrameWnd*)AfxGetMainWnd();          //主窗口
    }
    else                                               //创建一个新文档
    {
        pDocument = CreateNewDocument();
        bCreated = TRUE;
    }
    if (pFrame == NULL)                                //创建框架，作为主窗口
    {
        //使用 RUNTIME_CLASS 获取的运行时类结构信息，创建框架对象，同时创建视图
        pFrame = CreateNewFrame(pDocument, NULL);
    }
    if (lpszPathName == NULL)                          //若打开文件路径为空
    {
        SetDefaultTitle(pDocument);                    //设置默认标题
        if(!pDocument->OnNewDocument())                //创建新文档
            return NULL;
    }
    else                                               //打开一个存在的文件
    {
        if (!pDocument->OnOpenDocument(lpszPathName))
            return NULL;
        pDocument->SetPathName(lpszPathName);          //设置文件路径
    }
    CWinThread* pThread = AfxGetThread();              //获取程序的主线程对象
    if (bCreated && pThread->m_pMainWnd == NULL)
    {
        pThread->m_pMainWnd = pFrame;                  //pFrame 设为程序的主窗口
    }
    //初始化框架，激活视图，并发送 WM_INITIALUPDATE 消息
    // CView 类自动调用 OnInitialUpdate 函数完成视图的初始化更新
    InitialUpdateFrame(pFrame, pDocument, bMakeVisible);
    return pDocument;
}
```

在 App 类的 InitInstance 函数中，调用 ProcessShellCommand 函数，该函数根据传入的命名行参数，在函数内部自动 OpenDocumentFile 函数，具体实现请阅读源代码。

参数 lpszPathName 若为空创建一个新文件，否则打开一个文件。若程序已经存在一个文档，直接使用已有的文档、框架对象，无须重新创建，否则调用 CreateNewDocument 函数创建一个新的文档对象。

若框架对象不存在，根据 pDocument 文档对象，调用 CreateNewFrame 函数创建框架对象，同时创建视图，根据 RUNTIME_CLASS 宏获取的运行时类结构信息，运行时动态创建类对象。

若参数 lpszPathName 为空，则调用 SetDefaultTitle 函数设置默认的标题，调用 CDocument 类的 OnNewDocument 函数创建新文档。若不为空，则调用 CDocument 类的 OnOpenDocument 函数打开一个文件，再调用 SetPathName 函数记录打开文件的路径。

AfxGetThread 函数获取当前程序的主线程 CWinThread 类对象的指针，若创建成功，且当前主窗口为空，则设置 pFrame 为当前程序的主窗口。一个程序只能有一个主窗口，当主窗口关闭后，程序随之结束。

InitialUpdateFrame 函数用于初始化框架，激活视图，并发送 WM_INITIALUPDATE 消息，CView 类自动调用 OnInitialUpdate 函数完成视图的初始化更新。

应用程序向导自动创建的单文档程序，自带有“文件”菜单项，包括“新建”、“打开”、“保存”等子菜单，其中“新建”子菜单的 ID 为 ID_FILE_NEW，该 ID 由系统定义，函数调用流程如下所示：

```
    CWinApp::OnFileNew -> CDocManager::OnFileNew -> CDocTemplate::OpenDocumentFile ->
CDocument:: OnNewDocument
```

Tips 所谓的 MFC 框架流程，只是 Microsoft 定义的一套规则，由于代码本身就很灵活，函数的具体功能仅凭名称和文字介绍信息有时候是不够的，通过阅读源代码的方式，可以直接深入 MFC 框架，了解各个类和函数的具体实现，从使用 MFC 框架升级为掌握 MFC 框架，而这正是学习 MFC 的最大好处。

Tips MFC 定义了许多常用的命令 ID，并有关联的消息处理函数，如 ID_FILE_NEW 的消息处理函数为 CWinApp::OnFileNew。

“打开”子菜单的 ID 为 ID_FILE_OPEN，函数调用流程如下：

```
    CWinApp::OnFileOpen -> CDocManager::OnFileOpen -> CWinApp::OpenDocumentFile ->
CDocManager:: OpenDocumentFile -> CDocTemplate::OpenDocumentFile -> CDocument::OnOpen
Document
```

“保存”子菜单的 ID 为 ID_FILE_SAVE，“另存为”子菜单的 ID 为 ID_FILE_SAVE_AS，另外还有一个关闭 ID 为 ID_FILE_CLOSE，在 CDocument 类中已内置了消息处理函数，代码如下：

```
    BEGIN_MESSAGE_MAP(CDocument, CCmdTarget)
        //{{AFX_MSG_MAP(CDocument)
        ON_COMMAND(ID_FILE_CLOSE, OnFileClose)
        ON_COMMAND(ID_FILE_SAVE, OnFileSave)
        ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
        //}}AFX_MSG_MAP
    END_MESSAGE_MAP()
```

BEGIN_MESSAGE_MAP 和 END_MESSAGE_MAP 宏用于建立消息映射，BEGIN_MESSAGE_MAP 宏中的两个参数为当前类和基类名，中间为各种类型的消息映射，如 ON_COMMAND 宏用于命令 ID，参数为 ID、消息处理函数，ON_BN_CLICKED 宏用于按钮的单击事件，ON_NOTIFY 宏用于控件的通知消息。

Tips 通过类向导或其他方式添加消息处理函数时，MFC 在对应类的 BEGIN_MESSAGE_MAP 和 END_MESSAGE_MAP 宏之间，自动添加相应的消息映射，如单击 ID_FILE_SAVE 菜单时，自动调用 OnFileSave 函数。

“保存”子菜单的函数调用流程如下：

```
    CDocument::OnFileSave -> CDocument::DoFileSave -> CDocument::DoSave -> CDocument::
OnSaveDocument
```

“另存为”子菜单的函数调用流程如下：

```
    CDocument::OnFileSaveAs -> CDocument::DoSave -> CDocument::OnSaveDocument
```

“退出”ID 的函数调用流程如下：

```
    CDocument::OnFileClose -> CDocument::OnCloseDocument
```

通过以上的函数流程分析，拨开层层封装调用，定位到文档类的 4 个基本函数：OnNewDocument()、OnOpenDocument()、OnSaveDocument()、OnCloseDocument()。

OnNewDocument 函数用于新建一个文档，先调用 DeleteContents 函数删除文档数据，并清除文件路径，调用 SetModifiedFlag 函数设置修改标志为 FALSE。

DeleteContents 函数用于在新建、打开、关闭文档时，清空文档数据，默认为空操作，可在派生类中重写该函数，添加清除功能，格式如下：

```
void CDocument::DeleteContents()
```

SetModifiedFlag 函数用于设置文档是否被修改过，框架根据修改标志，提示是否需要保存，格式如下：

```
void CDocument::SetModifiedFlag(BOOL bModified=TRUE)
```

参数如下。

- bModified 修改标志，默认为 TRUE。

OnOpenDocument 函数根据传入的文件路径参数，打开一个文件。先调用 DeleteContents 函数删除文档数据，利用 CArchive 类装载文件，调用 Serialize 函数读取序列化文件后，关闭文件对象。

CArchive 类用于以二进制的形式保存和读取多个变量，变量既可以是基本类型，也可以是支持序列化的 CObject 派生类对象。Serialize 函数用于序列化对象的读取和保存，代码如下：

```
void CSdi061Doc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())        //若存储序列化对象
    {
        ar<<a1<<b1<<c1;
    }
    else                       //若读取序列化对象
    {
        ar>>a1>>b1>>c1;
    }
}
```

IsStoring 函数判断是否正在存储序列化对象，格式如下：

```
BOOL CArchive::IsStorng() const
```

返回值：若正在存储则返回非零值，否则返回 0。

IsLoading 函数判断是否正在读取序列化对象，与 IsStoring 函数相对应。若正在存储，则使用<<插入符将变量 a1、b1、c1 存入 ar 对象中，三个变量的值以二进制形式存入文件中。若正在读取，则使用>>提取符获取值，并分别存入变量 a1、b1、c1 中。

Tips 可根据<<和>>箭头的方向区分用途，如 ar<<a1<<b1<<c1；箭头指向 ar，表示要将三个变量存入 ar 中，而 ar>>a1>>b1>>c1；箭头指向三个变量，表示要将文件中的值存入三个变量中。使用序列化时，变量存储和读取的顺序要保持一致，否则会产生读取错误。

OnSaveDocument 函数在单击“保存”、“另存为”菜单，或退出程序时调用，若修改标志为 TRUE，在退出程序时会弹出提示对话框，询问是否需要保存。调用 Serialize 函数保存文件，并设置修改标志为 FALSE。

OnCloseDocument 函数在文档被关闭时调用，先销毁文档关联的所有视图，再调用 DeleteContents 函数删除文档数据，如有必要，调用 delete this；释放自身对象。

7.1.3 View 类

视图类用于显示文档数据，一个文档可以与多个视图关联，但一个视图只能关联一个文档，

多个视图可共用一个框架窗口，如 Word 中可在左边显示目录，右边显示正文，两个视图共用一个框架窗口。

当文档数据发生改变后，通过文档类的 UpdateAllViews 函数通知视图需要更新，在内部调用 View 类的 OnUpdate 函数，默认情况下，OnUpdate 函数在内部调用 Invalidate 函数，使整个视图无效并重绘。

Invalidate 函数设置整个 CWnd 窗口的客户区无效，格式如下：

```
void CWnd::Invalidate(BOOL bErase=TRUE)
```

参数如下。

- bErase：是否擦除客户区的背景，默认为 TRUE。

GetDocument 函数获取与视图关联的文档类对象的指针，格式如下：

```
CDocument* CView::GetDocument() const
```

返回值：文档类对象的指针。

AppWizard 自动生成的 CView 派生类重定义了 GetDocument 函数，且分为 Debug 和非 Debug 内联两种版本，Debug 版本在 cpp 文件中定义，代码如下：

```
CSdi061Doc* CSdi061View::GetDocument()
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSdi061Doc)));  //断言宏
    return (CSdi061Doc*)m_pDocument;                           //返回文档派生类指针
}
```

非调试版本使用 inline 关键字声明为内联函数，在头文件中定义，代码如下：

```
#ifndef _DEBUG
inline CSdi061Doc* CSdi061View::GetDocument()                  //内联函数
   { return (CSdi061Doc*)m_pDocument; }
#endif
```

ASSERT 断言宏用于判断其参数表达式是否为真，如 ASSERT(1==0)；会弹出一个错误窗口，并退出程序。类似的断言宏 VERIFY 具有同样的功能，区别在于 ASSERT 宏只能在 Debug 模式下可用，在 Release 版本中失效，而 VERIFY 宏在 Release 版本仍然被执行，但不会中断程序。

#ifndef 和#endif 是预编译指令，用于预编译判断，根据条件决定要编译哪些语句，#ifndef 判断是否没有定义某个宏，即 if no define，#endif 用于结束预编译判断。调试版本的程序使用_DEBUG 宏标志，预编译指令根据是否有_DEBUG 宏标志，决定编译哪些语句，如在 Debug 模式下，程序编译时跳过非 Debug 版本的 GetDocument 函数。

当视图窗口失效，或被其他窗口遮盖后重新显示时，需要重新绘制，Windows 发送 WM_PAINT 消息到视图窗口，自动调用 OnPaint()重绘函数，函数代码如下：

```
void CView::OnPaint()          //重绘函数
{
    CPaintDC dc(this);         //视图窗口的设备环境
    OnPrepareDC(&dc);          //绘制图形前设置属性
    OnDraw(&dc);               //派生类使用的重绘函数
}
```

OnPaint 函数内部先获取视图窗口的设备环境，用于在 DC 上绘制图形。OnPrepareDC 函数在实际绘制图形前调用，用于设置设备环境的属性。OnDraw 函数用于图形的重绘实现，格式如下：

```
virtual void OnDraw(CDC* pDC) = 0
```

参数如下。

- pDC：设备环境的指针。

CView 类中 OnDraw 函数被声明为纯虚函数，在派生类中必须重写该函数，否则派生类也成为抽象类，不能用于直接创建类对象。

Tips 不能在 OnDraw 函数中调用 Invalidate()及其他重绘函数，在 OnDraw()重绘函数里发出重绘消息，会造成死循环，程序陷入假死状态。

7.1.4 Frame 类

框架类用于程序窗口的界面显示，管理视图窗口、工具条、状态栏等元素，创建一个 Frame 框架窗口有如下三种方式：

- 直接使用 Create 函数创建。
- 直接使用 LoadFrame 函数创建。
- 使用文档模板类创建。

文档模板类也是通过内部调用 LoadFrame 函数创建框架窗口，LoadFrame 函数格式如下：

```
BOOL CFrameWnd::LoadFrame(UINT nIDResource,DWORD dwDefaultStyle = WS_OVERLAPPEDWIN
DOW|FWS_ ADDTOTITLE,CWnd*pParentWnd=NULL,CCreateContext* pContext=NULL)
```

参数如下。

- nIDResource：框架窗口使用的资源 ID。
- dwDefaultStyle：框架的窗口风格。
- pParentWnd：框架的父窗口指针。
- pContext：指向 CCreateContext 结构的指针。

Tips 类似于创建一个对话框，创建一个框架窗口需要两步，先构造一个类对象，再调用 LoadFrame 或 Create 函数创建框架实例，并关联到框架类对象。

CCreateContext 结构用于设定框架窗口关联的文档和视图，结构体定义如下：

```
struct CCreateContext
{
    CRuntimeClass* m_pNewViewClass;          //待创建的视图的 CRuntimeClass 指针
    CDocument* m_pCurrentDoc;                //用于关联新视图的已有文档
    CDocTemplate* m_pNewDocTemplate;         //用于创建多文档窗口的文档模板
    CView* m_pLastView;                      //待创建视图的先前视图
    CFrameWnd* m_pCurrentFrame;              //待创建框架的依赖框架
    CCreateContext();
};
```

在 Doc 类一节中，简要介绍了 OpenDocumentFile 函数的内部流程，其内部调用 CreateNewFrame 函数创建框架和视图，为理解框架内部如何创建，在 DOCTEMPL.CPP 文件找到函数定义，部分代码如下：

```
CFrameWnd* CDocTemplate::CreateNewFrame(CDocument* pDoc, CFrameWnd* pOther)
{
    CCreateContext context;                  //CCreateContext 结构对象
    context.m_pCurrentFrame = pOther;        //先前框架，默认为 NULL
    context.m_pCurrentDoc = pDoc;            //已有文档对象
    context.m_pNewViewClass = m_pViewClass;  //待创建的视图
    context.m_pNewDocTemplate = this;
    CFrameWnd* pFrame = (CFrameWnd*)m_pFrameClass->CreateObject();//运行时动态创建对象
```

```
        if(!pFrame->LoadFrame(m_nIDResource,WS_OVERLAPPEDWINDOW|FWS_ADDTOTITLE,NULL,
context))
            return NULL;                    //根据资源 ID、默认样式、context，创建框架实例
        return pFrame;
    }
```

定义一个 CCreateContext 结构的对象，pDoc 设为当前文档，m_pViewClass 为待创建的视图，根据 CRuntimeClass 运行时类信息和当前文档指针，动态创建一个视图。CreateObject 函数用于运行时动态创建类对象，格式如下：

```
static CObject* CRuntimeClass:: CreateObject(LPCSTR lpszClassName)
```

参数如下。

- lpszClassName：待创建类的名称。

返回值：指向新创建对象的 CObject 类指针。

CreateObject 函数动态创建框架类对象后，调用 LoadFrame 函数根据资源 ID、窗口样式、关联的文档视图类，创建框架窗口实例，并关联到框架类对象，创建完成后，返回创建的框架对象。

AppWizard 自动生成的 CMainFrame 类自带有 PreCreateWindow()和 OnCreate()两个函数，框架类调用 LoadFrame 函数时，在内部调用 Create 函数创建实例前，先调用 PreCreateWindow 函数进行窗口类的预设，可在该函数中修改窗口类的一些属性，函数代码如下：

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    //修改 cs 的属性值
    return TRUE;
}
```

参数为 CREATESTRUCT 结构体的引用，可修改 cs 的属性值，改变窗口的样式。CREATESTRUCT 结构体包含创建一个窗口所需的全部信息，定义如下：

```
typedef struct tagCREATESTRUCTA {
    LPVOID      lpCreateParams;         //传递给窗口的相关数据
    HINSTANCE   hInstance;              //窗口所在的模块的实例句柄
    HMENU       hMenu;                  //窗口使用的菜单句柄
    HWND        hwndParent;             //父窗口的句柄
    int         cy;                     //窗口的高度
    int         cx;                     //窗口的宽度
    int         y;                      //窗口的起点 y 坐标
    int         x;                      //窗口的起点 x 坐标
    LONG        style;                  //窗口的样式
    LPCSTR      lpszName;               //窗口的名称
    LPCSTR      lpszClass;              //窗口所在类的名称
    DWORD       dwExStyle;              //窗口使用的扩展样式
} CREATESTRUCTA, *LPCREATESTRUCTA;
```

从 LoadFrame 函数开始的内部调用流程如下。

（1）CFrameWnd::LoadFrame：加载资源，开始创建窗口框架和视图。

（2）CFrameWnd::PreCreateWindow：修改窗口属性。

（3）AfxRegisterWndClass：注册窗口类。

（3）CFrameWnd::Create：将相关参数传递给 Create 函数。

（4）CWnd::CreateEx：调用基类 CWnd 的创建函数。

（5）::CreateWindowEx：调用系统 API 函数，最终完成窗口创建。

Tips　PreCreateWindow 函数是虚函数，运行时根据实际框架类对象，调用对应版本的函数。

当框架调用 CreateWindowEx 函数创建窗口实例时，会发送 WM_CREATE 消息到新创建的窗口，自动调用 OnCreate 函数，用于完成窗口的初始化工作，如添加工具条、状态栏等，部分代码如下：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)          //调用基类版本函数
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))           //创建工具条
        return -1;
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))                 //创建状态栏
        return -1;
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);             //设置工具条的停靠方式
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);                          //工具条停靠到窗口
    return 0;
}
```

CToolBar 类对象 m_wndToolBar 调用 CreateEx 函数创建工具条，再调用 LoadToolBar 函数加载工具条资源。CStatusBar 类对象 m_wndStatusBar 调用 Create 函数创建状态栏，并调用 SetIndicators 函数设置指示器 ID，实现分栏显示。

m_wndToolBar.EnableDocking 函数设置工具条的可停靠方式，EnableDocking 函数设置框架的可停靠方式，DockControlBar 函数将工具条停靠到框架窗口。

7.1.5 类联系方式

在实际开发中，经常需要在自动生成的四个类间传递数据，在一个类中调用另一个类中的成员，框架提供了一些方法用于类之间的联系（以单文档程序 Sdi061 为例）。

（1）所有类中获取 App 类指针：

```
CWinApp* pApp=AfxGetApp();
CSdi061App* pMyApp=( CSdi061App*)pApp;
```

（2）所有类中获取 CMainFrame 类指针：

```
CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();
CMainFrame* pFrame=(CMainFrame*)(AfxGetApp()->m_pMainWnd);
```

（3）在框架（CMainFrame）中访问视图（View）：

```
CSdi061View* pView=(CSdi061View*)GetActiveView();
```

（4）在框架（CMainFrame）中访问文档（Doc）：

```
CSdi061Doc* pDoc=(CSdi061Doc*)GetActiveDocument();
```

（5）在视图（View）中访问文档（Doc）：

```
CSdi061Doc* pDoc = GetDocument();
```

（6）在文档（Doc）中访问关联的所有视图（View）：

```
POSITION pos = GetFirstViewPosition();
while (pos != NULL)
{
    CSdi061View* pView = (CSdi061View*)GetNextView(pos);
    //相关操作
}
```

7.2 菜单

菜单（Menu）是 Windows 程序的一个标准元素，每个菜单项可以包含弹出式子菜单，可为某个子菜单项添加消息处理函数，单击该项后，自动调用关联函数。可以禁用、勾选菜单项，AppWizard 自动生成的单文档程序自带有一个菜单资源，可修改该菜单资源，添加消息映射函数。

7.2.1 添加菜单资源

【实例 7-2】在工程 Sdi061 上添加菜单资源，并实现程序运行时，动态添加、移除、更新、禁用、勾选菜单项。

（1）打开工程 Sdi061，在资源视图上双击 Menu 节点下的 IDR_MAINFRAME 项，添加新菜单项，如图 7-2 所示。

（2）在空白菜单项单击鼠标右键，在弹出的快捷菜单中选择 Properties 命令，弹出“Menu Item Properties”窗口，如图 7-3 所示。

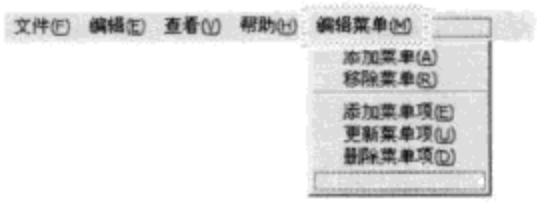


图 7-2　添加新菜单项

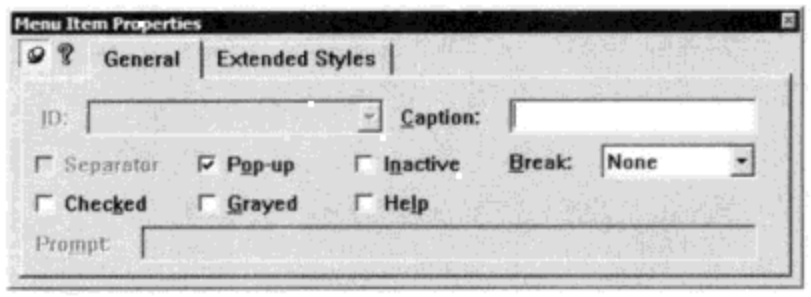


图 7-3　菜单项属性窗口

ID 组合框设置菜单项的 ID，Caption 编辑框设置显示的文本内容，Separator 复选框设置是否为分隔符，若勾选菜单项则变为分隔符样式。Pop-up 复选框设置是否为弹出菜单，若勾选，则无须设置 ID，弹出菜单可添加子菜单。

Checked 复选框设置是否显示勾选箭头，Grayed 复选框设置是否变灰不可用。Prompt 编辑框用于输入状态栏的提示信息，当鼠标移到该菜单项上时，在状态栏显示提示信息。

（3）新菜单的各项属性设置如表 7-1 所示。

表 7-1　新菜单项的属性设置

ID 值	Caption
无，勾选 Pop-up 复选框	编辑菜单（&M）
ID_MENU_ADD	添加菜单（&A）
ID_MENU_REMOVE	移除菜单（&R）
无，勾选 Separator 复选框	无
ID_MENU_ADD_ITEM	添加菜单项（&E）
ID_MENU_UPDATE_ITEM	更新菜单项（&U）
ID_MENU_DEL_ITEM	删除菜单项（&D）

Tips　要删除某个菜单项，选中后按 Delete 键删除。若要移动菜单项的位置，选中后用鼠标拖曳到指定位置。&符号用于添加快捷键，如&M 表示当菜单处于焦点时，按 M 键相当于单击该项，同一级别菜单不能设置相同的快捷键。

图 7-4　添加新菜单

（4）在资源视图上用鼠标右键单击 Menu 项，在弹出的快捷菜单中选择 Insert Menu 命令，添加一个新的菜单资源，在属性窗口设置新菜单的 ID 为 IDR_MENU_DRAW，如图 7-4 所示。

（5）添加菜单项，各项属性设置如表 7-2 所示。

表 7-2　新菜单项的属性设置

ID 值	Caption	Prompt 提示信息
无，勾选 Pop-up 复选框	绘制图形（&D）	无
ID_MENU_POINT	点（&P）	绘制点\n 绘制点元素
ID_MENU_LINE	折线（&L）	绘制折线\n 绘制连续折线
ID_MENU_REGION	多边形（&R）	绘制多边形\n 绘制多边形元素

Prompt 提示信息可分为两段，\n 前的文本在状态栏显示，\n 后的文本用于鼠标在工具条按钮上停留时显示，菜单项和工具按钮可使用同一个 ID，单击时实现同一个功能。

（6）选择 View|Resource Symbols 命令，弹出“Resource Symbols”窗口，单击“New”按钮，弹出“New Symbol”窗口，添加一个新的 ID 符号，如图 7-5 所示。

（7）在 Name 编辑框中输入 ID_MENU_TEST，单击“OK”按钮保存，关闭“Resource Symbols”窗口。

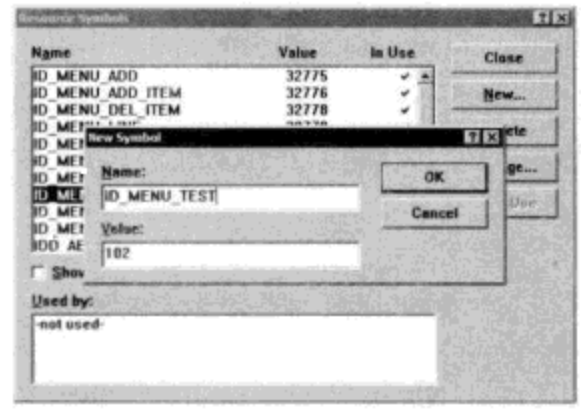


图 7-5　添加资源 ID 符号

Tips　“Resource Symbols”窗口显示已有的资源 ID 符号，可新建一个 ID 符号，系统自动分配一个未使用的 ID 值。所有 ID 符号在 resource.h 文件中定义，可直接在该文件中添加 ID。

7.2.2　更新菜单

菜单一般在框架窗口的顶部显示，可在程序运行时动态改变菜单项，通过 SetMenu 和 GetMenu 函数设置、获取窗口使用的菜单。

SetMenu 函数设置窗口使用的菜单，格式如下：

```
BOOL CWnd::SetMenu(CMenu* pMenu)
```

参数如下。

- pMenu：新菜单的 CMenu 类指针，若为 NULL，则当前菜单被清除。

返回值：若菜单改变则返回非零值，否则返回 0。

GetMenu 函数获取窗口的菜单指针，格式如下：

```
CMenu* CWnd::GetMenu() const
```

返回值：菜单的 CMenu 类指针，若无菜单则返回 NULL。

DrawMenuBar 函数用于重绘窗口的菜单，应在改变菜单项后调用，格式如下：

```
void CWnd::DrawMenuBar()
```

CMenu 类用于操作菜单资源，提供一系列方法如创建、添加、删除、更新菜单等，类似于窗口句柄 HWND，菜单使用 HMENU 句柄作为唯一标识符。创建一个菜单，要先构造一个 CMenu 类对象，再调用相关函数创建一个菜单实例，并关联到类对象。其常用函数如下所示：

（1）Attach 函数用于将已有的菜单句柄关联到类对象，格式如下：

```
BOOL CMenu::Attach(HMENU hMenu)
```

参数如下。

- hMenu：菜单的句柄值。

返回值：若成功则返回非零值，否则返回 0。

（2）Detach 函数用于从 CMenu 类对象中分离出菜单句柄实例，格式如下：

```
HMENU CMenu::Detach()
```

返回值：类对象关联的句柄实例。

（3）GetSafeHmenu 函数获取 CMenu 类对象关联的句柄值，格式如下：

```
HMENU CMenu::GetSafeHmenu() const
```

返回值：菜单对象的句柄值。

（4）CreatePopupMenu 函数用于创建一个空的弹出式菜单，格式如下：

```
BOOL CMenu::CreatePopupMenu()
```

返回值：若成功则返回非零值，否则返回 0。

（5）LoadMenu 函数用于从资源文件中加载菜单，并关联到 CMenu 类对象，格式如下：

```
BOOL CMenu::LoadMenu(UINT nIDResource)
```

参数如下。

- nIDResource：菜单资源的 ID。

返回值：若成功则返回非零值，否则返回 0。

（6）DestroyMenu 函数释放 CMenu 类对象关联的菜单实例，在析构函数中自动调用，格式如下：

```
BOOL CMenu::DestroyMenu()
```

返回值：若成功则返回非零值，否则返回 0。

（7）DeleteMenu 函数删除一个菜单项，以及其包括的所有子菜单，释放弹出式菜单占用的资源，格式如下：

```
BOOL CMenu::DeleteMenu(UINT nPosition, UINT nFlags)
```

参数如下。

- nPosition：要删除的菜单项，菜单项 ID 或位置索引。
- nFlags：指明 nPosition 的值类型，若为 MF_BYCOMMAND，nPosition 表示菜单 ID，若为 MF_BYPOSITION，nPosition 表示从 0 开始的位置索引。

返回值：若成功则返回非零值，否则返回 0。

（8）AppendMenu 函数用于在菜单尾部添加一个菜单项，格式如下：

```
BOOL CMenu::AppendMenu(UINT nFlags,UINT nIDNewItem=0,LPCTSTR lpszNewItem=NULL)
```

参数如下。

- nFlags：添加项的标志属性，如 MF_STRING 表示菜单项是一个字符串，MF_SEPARATOR 表示分隔符，MF_POPUP 表示添加项有弹出菜单。
- nIDNewItem：新项的 ID，若有弹出式菜单，则新项不需要 ID，该参数表示弹出菜单的句柄。
- lpszNewItem：默认为新项的文本值。

返回值：若成功则返回非零值，否则返回 0。

（9）GetMenuItemCount 函数获取顶层菜单或弹出菜单的项数，格式如下：

```
UINT CMenu::GetMenuItemCount() const
```

返回值：菜单项数。

（10）GetMenuString 函数获取菜单项的显示文本，格式如下：

```
int CMenu::GetMenuString(UINT nIDItem,CString& rString,UINT nFlags) const
```

参数如下。

- nIDItem：菜单项的 ID 或位置索引。
- rString：存放显示文本的值。
- nFlags：指明 nIDItem 的值类型，MF_BYCOMMAND 或 MF_BYPOSITION。

返回值：获取文本的实际字节数。

（11）GetSubMenu 函数获取指定菜单项下的弹出式菜单的指针，格式如下：

```
CMenu* CMenu::GetSubMenu(int nPos) const
```

参数如下。

- nPos：指定菜单项的位置索引。

返回值：若包含弹出式菜单，则返回弹出式菜单的 CMenu 类指针，否则返回 NULL。

（12）InsertMenu 函数用于在指定位置插入一个菜单项，格式如下：

```
BOOL CMenu::InsertMenu(UINT nPosition,UINT nFlags,UINT nIDNewItem=0,LPCTSTR
lpszNewItem=NULL)
```

参数如下。

- nPosition：插入位置所在项的 ID 或位置索引。
- nFlags：指明 nPosition 的值类型。
- nIDNewItem：新项的 ID，若有弹出式菜单，传入弹出菜单的句柄。
- lpszNewItem：新项的文本值。

返回值：若成功则返回非零值，否则返回 0。

（13）ModifyMenu 函数用于更新一个菜单项，格式如下：

```
BOOL CMenu::ModifyMenu(UINT nPosition,UINT nFlags,UINT nIDNewItem=0,LPCTSTR
lpszNewItem=NULL)
```

参数如下。

- nPosition：更新项的 ID 或位置索引。
- nFlags：指明 nPosition 的值类型，以及使用的新标志值，如 MF_CHECKED 表示勾选该项。
- nIDNewItem：更新项的新 ID。
- lpszNewItem：更新项的新文本。

返回值：若成功则返回非零值，否则返回 0。

（14）RemoveMenu 函数用于移除一个菜单项，但不销毁弹出菜单的句柄，可重用弹出菜单，格式如下：

```
BOOL CMenu::RemoveMenu(UINT nPosition,UINT nFlags)
```

参数如下。

- nPosition：移除项的 ID 或位置索引。
- nFlags：指明 nPosition 的值类型。

返回值：若成功则返回非零值，否则返回 0。

默认使用自动生成的 IDR_MAINFRAME 菜单资源，作为框架窗口的菜单，若要使用自定义的菜单资源作为窗口的菜单，步骤如下：

（1）构造一个 CMenu 类对象。

（2）调用 CMenu::LoadMenu 函数加载菜单资源，并将菜单实例句柄附加到类对象上。

（3）调用 CWnd::SetMenu 函数设置窗口菜单。

（4）调用 CMenu::Detach 函数分离类对象和菜单实例句柄，类对象只是负责为窗口提供菜单实例句柄，完成后，实例句柄交由窗口管理。当类对象超出作用域被析构时，不会顺带释放菜单实例。

（5）窗口销毁时，自动销毁使用的菜单实例。

具体操作步骤如下：

（1）在类视图用鼠标右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个 CMenu 类型的变量 m_newMenu。

（2）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CMainFrame 项，Object IDs 列表框选择 ID_MENU_ADD 项，Messages 列表框选择 Command 项，单击 Add Function 按钮添加菜单命令处理函数。用同样的方式，为 ID_MENU_REMOVE、ID_MENU_ADD_ITEM、ID_MENU_UPDATE_ITEM、ID_MENU_DEL_ITEM 添加函数。单击 OK 按钮保存并退出。

（3）在类视图双击 CMainFrame 类下的 OnCreate 项，定位到函数，在 return 0; 前添加如下代码：

```
    m_newMenu.LoadMenu(IDR_MENU_DRAW);          //加载菜单资源
```

（4）双击 OnMenuAdd 项，添加如下代码：

```
    void CMainFrame::OnMenuAdd()
    {
        CMenu* pMenu=GetMenu();                 //获取窗口菜单指针
        pMenu->AppendMenu(MF_POPUP|MF_STRING,(UINT)m_newMenu.GetSubMenu(0)->GetSafeH
menu()
            ,"绘制图形(&D)");                   //在顶层菜单栏尾部添加一个菜单项，且有弹出菜单
        DrawMenuBar();
    }
```

GetMenu 函数获取窗口使用的菜单的指针。AppendMenu 函数在顶层菜单尾部添加一项，参数 1 中的 MF_POPUP 表示有弹出菜单，MF_STRING 表示菜单项是一个字符串，参数 2 为弹出菜单的句柄。GetSubMenu 函数获取 IDR_MENU_DRAW 菜单的第 1 个弹出菜单，GetSafeHmenu 函数获取弹出菜单的实例句柄。DrawMenuBar 函数用于重绘窗口菜单。

Tips　当更新窗口菜单后，应调用 DrawMenuBar 函数重绘菜单，否则不能及时显示更新效果。

（5）在类视图用鼠标右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加一个返回值为 BOOL 的函数 IsMenuExist()，添加如下代码：

```
    BOOL CMainFrame::IsMenuExist()
    {
        CMenu* pMenu=GetMenu();                          //获取窗口菜单指针
        int menuCount=pMenu->GetMenuItemCount();         //获取顶层菜单项的数目
        CString strName;
        pMenu->GetMenuString(menuCount-1,strName,MF_BYPOSITION); //获取最后一个菜单的显示文本
        if(strName=="绘制图形(&D)")                      //若是指定值，则返回 TRUE
            return TRUE;
        else
            return FALSE;
    }
```

GetMenuItemCount 函数获取当前菜单对象包含的菜单项数目，不包括弹出菜单。GetMenuString 函数获取最后一个菜单的显示文本，参数 1 为菜单项的位置索引，参数 3 指定参

数 1 的类型，获取的文本存放到 strName 中。若 strName 等于新增菜单的显示文本，表明已添加菜单，则返回 TRUE，否则返回 FALSE。

（6）双击 OnMenuRemove 项，添加如下代码：

```
void CMainFrame::OnMenuRemove()
{
    if(IsMenuExist())          //若已添加菜单
    {
        GetMenu()->RemoveMenu(GetMenu()->GetMenuItemCount()-1,MF_BYPOSITION);
        DrawMenuBar();         //移除添加的菜单，并重绘窗口菜单
    }
}
```

RemoveMenu 函数移除最后一个菜单项，即已添加的菜单，参数 1 为删除项的位置索引，参数 2 指明参数 1 的类型为位置索引。

（7）双击 OnMenuAddItem 项，添加如下代码：

```
void CMainFrame::OnMenuAddItem()
{
    if(IsMenuExist())                                    //若已添加菜单
    {
        CMenu* mainMenu=GetMenu();                       //获取已添加菜单的弹出菜单
        CMenu* pSubMenu=mainMenu->GetSubMenu(mainMenu->GetMenuItemCount()-1);
        if(pSubMenu->GetMenuItemCount()<4)               //若尚未添加子菜单
        {
            pSubMenu->InsertMenu(0,MF_STRING|MF_BYPOSITION,ID_MENU_TEST,"test menu");
            DrawMenuBar();                               //在起始位置添加一个子菜单，并重绘窗口菜单
        }
    }
}
```

GetSubMenu 函数获取已添加菜单的弹出菜单，若弹出菜单的项数小于 4，表明尚未添加子菜单（初始有 3 项），InsertMenu 函数在起始位置插入一项，参数 1 为位置索引，参数 3 为新项的 ID，参数 4 为新项的文本。

（8）双击 OnMenuUpdateItem 项，添加如下代码：

```
void CMainFrame::OnMenuUpdateItem()
{
    if(IsMenuExist())                                    //若已添加菜单
    {
        CMenu* mainMenu=GetMenu();
        CMenu* pSubMenu=mainMenu->GetSubMenu(mainMenu->GetMenuItemCount()-1);
        if(pSubMenu->GetMenuItemCount()==4)              //若已添加子菜单
        {
            pSubMenu->ModifyMenu(ID_MENU_TEST,MF_STRING|MF_BYCOMMAND,
                ID_MENU_TEST,"测试菜单");                //修改子菜单的文本
            DrawMenuBar();                               //重绘窗口菜单
        }
    }
}
```

若弹出菜单的项数为 4，则表明已添加子菜单，ModifyMenu 函数更新菜单项，参数 1 为更新项的 ID，参数 2 中的 MF_BYCOMMAND 指明参数 1 的类型为菜单项 ID，MF_STRING 指明要修改文本，参数 3 为菜单项的新 ID，参数 4 为新的文本。

（9）双击 OnMenuDelItem 项，添加如下代码：

```
void CMainFrame::OnMenuDelItem()
{
    if(IsMenuExist())                                    //若已添加菜单
```

```
    {
        CMenu* mainMenu=GetMenu();
        CMenu* pSubMenu=mainMenu->GetSubMenu(mainMenu->GetMenuItemCount()-1);
        if(pSubMenu->GetMenuItemCount()==4)                    //若已添加子菜单
        {
            pSubMenu->DeleteMenu(ID_MENU_TEST,MF_BYCOMMAND);  //删除子菜单
            DrawMenuBar();
        }
    }
}
```

DeleteMenu 函数删除菜单项，参数 1 为菜单项 ID，参数 2 指明参数 1 的类型。

（10）生成程序并运行，如图 7-6 所示。“编辑菜单”下有 5 个子菜单，各自功能如下：

- 添加菜单（A）：在顶层菜单尾部添加一个“绘制图形（&D）”菜单。
- 移除菜单（R）：移除添加的“绘制图形（&D）”菜单。
- 添加菜单项（E）：在“绘制图形（&D）”菜单下添加一个“test menu”子菜单。
- 更新菜单项（U）：修改“test menu”子菜单的文本为“测试菜单”。
- 删除菜单项（D）：删除添加的子菜单。

图 7-6　更新窗口菜单

7.2.3　禁用和勾选菜单

菜单项可以被禁用和勾选，禁用时菜单项变为灰色不可用，勾选时菜单项前出现一个“√”，表示该项已选中。MFC 提供用户接口（User Interface）函数管理每个菜单项的状态，通过与菜单项关联的 CCmdUI 类，可以轻松设置菜单项的外观。

Enable 函数用于设置菜单项是否可用，格式如下：

```
void CCmdUI::Enable(BOOL bOn=TRUE)
```

参数如下。

- bOn：是否可用，TRUE 表示可用，FALSE 表示不可用。

SetCheck 函数设置菜单项是否勾选，格式如下：

```
void CCmdUI::SetCheck(int nCheck=1)
```

参数如下。

- nCheck：是否勾选，1 表示勾选，0 表示不勾选。

（1）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CMainFrame 项，Object IDs 列表框选择 ID_MENU_ADD 项，Messages 列表框选择 UPDATE_COMMAND_UI 项，单击“Add Function”按钮添加用户接口函数。用同样方式，为 ID_MENU_REMOVE 添加用户接口函数，单击“OK”按钮保存并退出。

UI 处理函数要选择 UPDATE_COMMAND_UI 消息，当单击菜单项，在弹出菜单显示之前调用该函数，函数自带一个 CCmdUI 类指针参数，可用类指针调用相关函数，设置菜单外观状态。

（2）在类视图双击 CMainFrame 类下的 OnUpdateMenuAdd 项，添加如下代码：

```
void CMainFrame::OnUpdateMenuAdd(CCmdUI* pCmdUI)
{
    if(IsMenuExist())                          //若已添加菜单
    {
        pCmdUI->SetCheck();                    //勾选
        pCmdUI->Enable(FALSE);                 //禁用
    }
    else
```

```
    {
        pCmdUI->SetCheck(FALSE);          //取消勾选
        pCmdUI->Enable(TRUE);             //可用
    }
}
```

在“添加菜单”的 UI 函数里，若已添加菜单，则勾选并禁用该菜单项，否则取消勾选并设为可用。SetCheck 函数设置勾选状态，Enable 函数设置是否可用。

（3）双击 OnUpdateMenuRemove 项，添加如下代码：

```
void CMainFrame::OnUpdateMenuRemove(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(IsMenuExist());
}
```

若 IsMenuExist 函数返回 TRUE，则表示已添加菜单，调用 Enable 函数设置该菜单项可用，否则禁用该项。

（4）生成程序并运行，如图 7-7 所示。单击“添加菜单”，添加“绘制图形”菜单后，“添加菜单”变为灰色不可用，并勾选，此时“移除菜单”可用。单击“移除菜单”后，切换可用状态。

图 7-7 菜单 UI 控制

7.2.4 右键菜单

右键菜单也称为环境（Context）菜单，可在屏幕任意位置显示，一般在窗口用鼠标右键单击鼠标时弹出。系统的默认消息处理函数 DefWindowProc，在处理 WM_RBUTTONUP 和 WM_NCRBUTTONUP 消息时，自动发送 WM_CONTEXTMENU 消息，可添加该消息的处理函数，实现右键菜单。

【实例 7-3】新建一个单文档工程名为 Sdi062，在视图窗口中使用右键菜单设置背景色。

图 7-8 右键菜单资源

（1）新建单文档工程 Sdi062，在资源视图用鼠标右键单击 Menu 项，在弹出的快捷菜单中选择 Insert Menu 命令，添加一个新的菜单资源，在属性窗口设置 ID 为 IDR_MENU_COLOR，添加菜单项，如图 7-8 所示。

（2）设置“红色”项的 ID 为 ID_MENU_RED，“绿色”项的 ID 为 ID_MENU_GREEN，“蓝色”项的 ID 为 ID_MENU_BLUE。

（3）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CSdi062View 项，Object IDs 列表框选择 ID_MENU_RED 项，Messages 列表框选择 COMMAND 项，单击“Add Function”按钮添加消息处理函数。用同样方式，为 ID_MENU_GREEN、ID_MENU_BLUE 添加函数，单击“OK”按钮保存并退出。

（4）在类视图双击 CSdi062View 项，在类定义中添加三个成员变量，代码如下：

```
public:
    CMenu m_menuPopup;                //弹出式菜单
    CMenu m_Menu;                     //右键菜单
    COLORREF m_clrFill;               //背景填充颜色
```

（5）选择 View|Resource Symbols 命令，弹出“Resource Symbols”窗口，单击“New”按钮，添加两个资源 ID 符号：ID_MENU_WHITE、ID_MENU_BLACK。

（6）在类视图用鼠标右键单击 CSdi062View 项，在弹出的快捷菜单中选择 Add Virtual Function 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CSdi062View::OnInitialUpdate()
{
```

```
    CView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class
    m_clrFill=RGB(255,255,255);                                    //初始颜色为白色
    m_menuPopup.CreatePopupMenu();                                 //创建弹出式菜单
    m_menuPopup.AppendMenu(MF_STRING,ID_MENU_WHITE,"白色");        //添加“白色”子菜单
    m_menuPopup.AppendMenu(MF_STRING,ID_MENU_BLACK,"黑色");        //添加“黑色”子菜单
    m_Menu.LoadMenu(IDR_MENU_COLOR);                               //加载菜单资源
    m_Menu.GetSubMenu(0)->AppendMenu(MF_STRING|MF_POPUP,(UINT)m_menuPopup.GetSaf
eHmenu(),"黑白色");                                                //将弹出式菜单添加到尾部
}
```

CreatePopupMenu 函数创建一个空的弹出式菜单，AppendMenu 函数在弹出菜单中添加两个新项。LoadMenu 函数加载菜单资源，AppendMenu 函数将弹出菜单添加到右键菜单的尾部，参数 1 中的 MF_POPUP 表示该项有弹出菜单，GetSafeHmenu 函数获取弹出菜单的句柄，并转换为 UINT 类型。

（7）在类视图双击 CSdi062View 类下的 OnMenuRed 项，添加如下代码：

```
void CSdi062View::OnMenuRed()
{
    m_clrFill=RGB(255,0,0);          //颜色设为红色
    Invalidate(TRUE);                //重绘视图窗口
}
```

（8）双击 OnMenuGreen 项，添加如下代码：

```
void CSdi062View::OnMenuGreen()
{
    m_clrFill=RGB(0,255,0);          //颜色设为绿色
    Invalidate(TRUE);
}
```

（9）双击 OnMenuBlue 项，添加如下代码：

```
void CSdi062View::OnMenuBlue()
{
    m_clrFill=RGB(0,0,255);          //颜色设为蓝色
    Invalidate(TRUE);
}
```

（10）在类视图用鼠标右键单击 CSdi062View 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_CONTEXTMENU 消息处理函数，添加如下代码：

```
void CSdi062View::OnContextMenu(CWnd* pWnd, CPoint point)
{
    m_Menu.GetSubMenu(0)->TrackPopupMenu(TPM_LEFTALIGN,point.x,point.y,this);
}
```

TrackPopupMenu 函数用于在指定位置显示弹出菜单，格式如下：

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

参数如下。

- nFlags：弹出菜单和参数 2、3 的对齐标志，如 TPM_LEFTALIGN 表示左对齐。
- x：菜单显示的水平位置。
- y：菜单显示的垂直位置。
- pWnd：当前窗口的指针。
- lpRect：矩形对象，鼠标在该矩形内单击，弹出菜单不会消失，若为 NULL，鼠标在菜单外部单击，弹出菜单消失。

返回值：若成功则返回非零值，否则返回 0。

当鼠标在视图窗口用鼠标右键单击时，系统默认的消息处理函数发出 WM_CONTEXTMENU 消息，自动调用 OnContextMenu 函数。TrackPopupMenu 函数在鼠标右键单击处显示环境菜单，参数 2、3 为鼠标所在位置，参数 1 表示环境菜单与鼠标位置左对齐，参数 4 为当前视图窗口的指针。

（11）双击 CSdi062View 类下的 OnDraw 项，添加如下代码：

```
void CSdi062View::OnDraw(CDC* pDC)
{
    CSdi062Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rcClient;
    GetClientRect(rcClient);        //获取客户区矩形
    pDC->FillSolidRect(rcClient,m_clrFill);   //用颜色填充矩形
}
```

（12）生成程序并运行，如图 7-9 所示。在视图窗口用鼠标右键单击鼠标，显示环境菜单，其中“红色”、“绿色”、“蓝色”是在资源视图中添加的菜单项，“黑白色”弹出式菜单是在视图的初始化函数中，通过代码动态添加的。单击“红色”、“绿色”、“蓝色”中的一项，视图背景自动切换为对应的颜色。

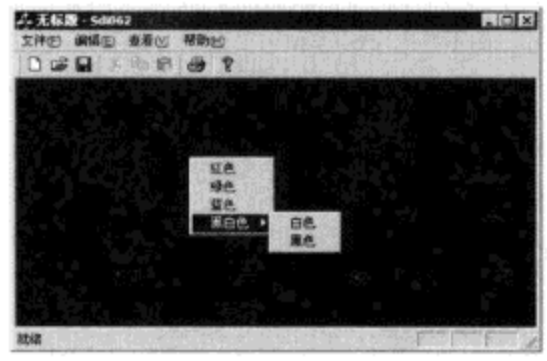


图 7-9 使用右键菜单

Tips 在 View 类中实现的弹出菜单，只能在客户区中显示。若要在非客户区显示环境菜单，可在框架类中添加相应功能。

在资源视图中添加的菜单项，可通过类向导添加消息处理函数，但用代码动态添加的菜单项，无法使用类向导，需要手动添加消息处理函数，步骤如下所示：

- 在头文件类定义中，添加函数声明。
- 在消息映射宏中，添加消息映射。
- 在 CPP 文件中，添加函数实现。

（13）在类视图双击 CSdi062View 项，在类定义中添加两个函数声明，代码如下：

```
protected:
    //{{AFX_MSG(CSdi062View)
    afx_msg void OnMenuRed();
    afx_msg void OnMenuGreen();
    afx_msg void OnMenuBlue();
    afx_msg void OnMenuWhite();          //手动添加的消息处理函数 OnMenuWhite
    afx_msg void OnMenuBlack();          //手动添加的消息处理函数 OnMenuBlack
    afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

//{{AFX_MSG 和//}}AFX_MSG 是框架自定义的标记，用于帮助类向导自动添加代码到该处。afx_msg 是一个宏定义，表明当前函数是一个消息处理函数。DECLARE_ MESSAGE_MAP 宏表明该类可以使用消息映射机制。

（14）在类视图双击 CSdi062View 类下的一项，打开 CPP 文件，在文件开头处找到 BEGIN_MESSAGE_MAP 消息映射宏，添加两个消息映射，代码如下：

```
BEGIN_MESSAGE_MAP(CSdi062View, CView)
    //{{AFX_MSG_MAP(CSdi062View)
```

```
    ON_COMMAND(ID_MENU_RED, OnMenuRed)
    ON_COMMAND(ID_MENU_GREEN, OnMenuGreen)
    ON_COMMAND(ID_MENU_BLUE, OnMenuBlue)
    ON_COMMAND(ID_MENU_BLACK, OnMenuBlack)              //手动添加的消息映射
    ON_COMMAND(ID_MENU_WHITE, OnMenuWhite)              //手动添加的消息映射
    ON_WM_CONTEXTMENU()
    //}}AFX_MSG_MAP
    //省略内容…
END_MESSAGE_MAP()
```

ON_COMMAND 宏用来指定处理菜单命令消息的函数，参数 1 为菜单 ID，参数 2 为处理该 ID 命令的消息处理函数。

（15）在 CPP 文件尾部手动添加如下两个函数，代码如下：

```
void CSdi062View::OnMenuBlack()                 //“黑色”菜单处理函数
{
    m_clrFill=RGB(0,0,0);                       //颜色设为黑色
    Invalidate(TRUE);                           //重绘视图窗口
}
void CSdi062View::OnMenuWhite()                 //“白色”菜单处理函数
{
    m_clrFill=RGB(2550,255,255);                //颜色设为白色
    Invalidate(TRUE);
}
```

通过在 3 个位置手动添加代码，在程序运行时动态添加的菜单项，也有相应的消息处理函数。类向导的功能虽强大，也有力所不及的时候，通过手工添加代码方式，可以更加灵活地控制程序。

（16）生成程序并运行，在视图窗口右键单击弹出环境菜单，单击“黑色”和“白色”菜单项，视图背景自动填充为黑色或白色。

7.3 工具栏

工具栏是 Windows 程序的标准元素，类似于菜单，单击一项后执行相应的命令，工具按钮和菜单项可共用一个 ID，响应同一个消息函数，常作为菜单的快捷方式。一个程序可以有多个工具栏，实现不同类别的功能，工具栏可在框架窗口停靠，如顶部、左边框、底部等位置，或作为一个浮动窗口显示。

7.3.1 添加工具栏资源

【实例 7-4】新建一个单文档工程名为 Sdi063，添加一个工具栏，实现鼠标交互式绘点、绘线功能。

（1）新建单文档工程 Sdi063，在资源视图用鼠标右键单击 Toolbar 项，在弹出的快捷菜单中选择 Insert Toolbar 命令，添加一个工具栏资源，设置 ID 为 IDR_TOOLBAR_DRAW，并添加两个按钮，如图 7-10 所示。

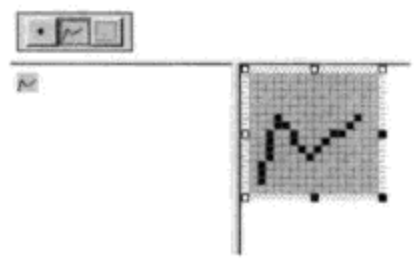


图 7-10　添加工具栏资源

顶部窗口显示已添加的工具按钮，左边显示预览图，右边为按钮编辑窗口，可使用工具箱和颜色板绘制按钮图形，也可直接从其他工程中复制。可在顶部窗口用鼠标移动按钮的位置，若要移除某个按钮，用鼠标拖曳到窗口外。

（2）双击按钮，弹出“Toolbar Button Properties”窗口，设置第 1 个按钮的 ID 为 ID_BUTTON_POINT，在 Prompt 编辑框输入“绘制点元素\n 绘点”，设置第 2 个按钮的 ID 为 ID_BUTTON_LINE，在 Prompt 编辑框输入“绘制线元素\n 绘线”。

Prompt 编辑框中的内容，\n 之前的在状态栏显示，\n 之后的当鼠标在工具按钮上停留时显示。

7.3.2 显示工具栏

创建并显示一个工具栏，步骤如下所示：

- 在资源视图，添加一个工具栏资源。
- 在框架类中，构造 CToolBar 类对象。
- 调用 Create 或 CreateEx 函数，创建一个工具栏实例，并关联到类对象。
- 调用 LoadToolBar 函数，加载工具条资源。
- 设置工具栏的停靠方式，以及框架窗口的可停靠方式。
- 调用框架类的 DockControlBar 函数，实现工具栏的停靠显示。

（1）在类视图用鼠标右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加 CToolBar 类型的变量 m_wndToolDraw。

（2）双击 CMainFrame 类下的 OnCreate 项，定位到函数，在 return 0；前添加如下代码：

```
if (!m_wndToolDraw.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
  | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
  !m_wndToolDraw.LoadToolBar(IDR_TOOLBAR_DRAW))                    //创建工具栏，加载工具栏资源
  return -1;
m_wndToolDraw.EnableDocking(CBRS_ALIGN_LEFT|CBRS_ALIGN_TOP); //设置停靠方式
DockControlBar(&m_wndToolDraw);                                    //停靠显示
```

CreateEx 函数用于创建一个有扩展风格的 Windows 工具栏，并关联到类对象，格式如下：

```
BOOL CToolBar::CreateEx(CWnd* pParentWnd,DWORD dwCtrlStyle=TBSTYLE_FLAT,
   DWORD dwStyle=WS_CHILD|WS_VISIBLE|CBRS_ALIGN_TOP,
   CRect rcBorders=CRect(0,0,0,0),UINT nID=AFX_IDW_TOOLBAR)
```

参数如下。

- pParentWnd：父窗口的指针，this 表示框架类对象。
- dwCtrlStyle：嵌入的 CToolBarCtrl 对象的风格，如 TBSTYLE_FLAT 表示平面风格。
- dwStyle：工具栏的风格，如 WS_CHILD 表示子窗口，WS_VISIBLE 表示可见，CBRS_ALIGN_TOP 表示顶部停靠。
- rcBorders：工具栏窗口边框矩形对象，默认没有边框。
- nID：工具条子窗口的 ID。

返回值：若成功则返回非零值，否则返回 0。

LoadToolBar 函数用于加载工具栏资源，格式如下：

```
BOOL CToolBar::LoadToolBar(UINT nIDResource)
```

参数如下。

- nIDResource：工具栏资源的 ID。

返回值：若成功则返回非零值，否则返回 0。

EnableDocking 函数设置控制条的停靠方式，格式如下：

```
void CControlBar::EnableDocking(DWORD dwStyle)
```

参数如下。

- dwStyle：停靠方式，如 CBRS_ALIGN_TOP 表示可停靠在框架窗口顶部，CBRS_ALIGN_ANY 表示可停靠在任意位置。

DockControlBar 函数根据可停靠方式，将工具栏停靠在框架窗口上，格式如下：

```
    void CFrameWnd::DockControlBar(CControlBar* pBar,UINT nDockBarID=0,LPCRECT lpRect=
NULL)
```

参数如下。

- pBar：停靠控制条的指针。
- nDockBarID：停靠位置，如 AFX_IDW_DOCKBAR_TOP 表示在顶部停靠，0 表示在任意位置停靠。
- lpRect：用屏幕坐标表示停靠位置。

CcontrolBar 类是所有控制条如工具栏、状态栏、对话栏类的基类，控制条可以包含子窗口，可在框架窗口停靠，框架类提供一系列函数操作控制条，常用函数如下所示：

ShowControlBar 函数设置是否显示控制条，格式如下：

```
    void CFrameWnd::ShowControlBar(CControlBar* pBar,BOOL bShow,BOOL bDelay)
```

参数如下。

- pBar：控制条的指针。
- bShow：是否显示，TRUE 表示显示，FALSE 表示隐藏。
- bDelay：是否延迟显示，FALSE 表示立即显示。

GetControlBar 函数根据控制条的 ID，获取控制条指针，格式如下：

```
    CContrdBar* CFrameWnd::GetControlBar(UINT nID)
```

参数如下。

- nID：控制条的 ID。

返回值：控制条的指针。

EnableDocking 函数设置框架的哪个位置可供停靠，格式如下：

```
    void CFrameWnd::EnableDocking(DWORD dwDockStyle)
```

参数如下。

- dwDockStyle：停靠方向，如 CBRS_ALIGN_TOP 表示顶部可停靠，CBRS_ALIGN ANY 表示任意位置都可停靠。

（3）生成程序并运行，在框架窗口顶部，出现新增的工具栏。

7.3.3　添加按钮处理函数

若尚未为按钮添加消息处理函数，则按钮变灰不可用，可使用类向导，为工具栏按钮添加消息处理函数，实现交互式绘点、绘线功能。

（1）在类视图双击 CSdi063View 项，在类定义中添加如下成员变量：

```
public:
    CString m_strPerLine;                    //每条折线上的所有点坐标
    CPoint m_ptEnd;                          //交互绘线的终点坐标
    CPoint m_ptStart;                        //交互绘线的起点坐标
    int m_nFlag;                             //标识符，若为 1 则表示绘点，2 表示绘线
    COLORREF m_clrFill;                      //画刷填充颜色
    COLORREF m_clrPen;                       //画笔颜色
    CBrush m_br;                             //画刷对象
    CPen m_pen;                              //画笔对象
    CStringArray m_listPoint;                //所有点坐标
    CStringArray m_listLine;                 //所有折线的坐标
```

（2）用鼠标右键单击 CSdi063View 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
    void CSdi063View::OnInitialUpdate()
```

```
{
    CView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class
    m_clrPen=RGB(80,120,210);                    //画笔颜色
    m_clrFill=RGB(220,10,10);                    //画刷填充颜色
    m_pen.CreatePen(PS_SOLID,3,m_clrPen);        //创建画笔实例
    m_br.CreateSolidBrush(m_clrFill);            //创建画刷实例
    m_nFlags=0;                                  //标识符初始为 0
    m_ptStart.x=m_ptStart.y=0;                   //设置起点和终点坐标值为 0
    m_ptEnd.x=m_ptEnd.y=0;
}
```

（3）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CSdi063View 项，Object IDs 列表框选择 ID_BUTTON_POINT 项，Messages 列表框选择 COMMAND 项，单击“Add Function”按钮添加消息处理函数。用同样的方式，为 ID_BUTTON_LINE 添加函数，单击“OK”按钮保存并退出。

（4）在类视图双击 OnButtonPoint 项，添加如下代码：

```
void CSdi063View::OnButtonPoint()
{
    m_nFlags=1;              //设置标识符为 1，鼠标函数根据此标志，绘点
}
```

（5）在类视图双击 OnButtonLine 项，添加如下代码：

```
void CSdi063View::OnButtonLine()
{
    m_nFlags=2;              //设置标识符为 2，鼠标函数根据此标志，绘线
}
```

（6）在类视图用鼠标右键单击 CSdi063View 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_LBUTTONDOWN、WM_MOUSEMOVE、WM_RBUTTONDOWN 三个消息的处理函数。

（7）在类视图双击 OnLButtonDown 项，添加如下代码：

```
void CSdi063View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CView::OnLButtonDown(nFlags, point);
    if(m_nFlags==1)                                  //若标识符为 1，绘点
    {
        CClientDC dc(this);                          //获取视图客户区窗口的设备环境
        //以当前鼠标位置绘制一个圆，代表点
        dc.Ellipse(point.x-3,point.y-3,point.x+3,point.y+3);

        CString strX,strY;
        strX.Format("%d",point.x);                   //将 x、y 坐标值格式化为字符串
        strY.Format("%d",point.y);
        m_listPoint.Add(strX+","+strY);              //当前点的坐标值存入动态数组中
    }
    else if(m_nFlags==2)                             //若标识符为 2，则绘线
    {
        if(m_ptStart.x==0 && m_ptEnd.x==0)           //若起止点的坐标值为 0，则表明尚未开始绘线
            m_strPerLine="";                         //清空存放折线坐标值的字符串
        m_ptStart=point;                             //设置起点为当前点
        m_ptEnd=point;                               //设置终点为当前点
        CString strX,strY;
        strX.Format("%d",m_ptStart.x);               //将 x、y 坐标值格式化为字符串
        strY.Format("%d",m_ptStart.y);
        m_strPerLine+=strX+","+strY+",";             //将当前点的坐标值添加到当前折线中
```

```
    }
}
```

当鼠标在视图窗口上按下左键时，触发 WM_LBUTTONDOWN 消息，自动调用该函数，参数 1 为按键标志值，如 MK_CONTROL 表示 Ctrl 键同时按下，MK_SHIFT 表示 Shift 键同时按下，参数 2 为按下时的坐标点值。

根据标识符 m_nFlags 的值执行不同的操作，若为 1，执行绘点操作。CClientDC dc(this); 获取当前窗口的客户区的设备环境，this 指向当前视图类对象，Ellipse 函数绘制一个圆形，代表当前点。

Format 函数将当前点的 x、y 值格式化为字符串，Add 函数将坐标值以字符串形式如“125,453”，存入点数组中，m_listPoint 是一个动态数组，数组每个元素存放一个点的 x、y 坐标值，以逗号分隔。

若 m_nFlags 值为 2，执行绘线操作。若 m_ptStart 和 m_ptEnd 的 x 值都为 0，表明尚未开始绘线，m_strPerLine 存放一条折线上所有点的坐标值。绘制折线时，鼠标左键按下一次，表示添加一个点，设置起点 m_ptStart 和终点 m_ptEnd 为当前点，从该点开始绘制。

Format 函数将当前点的 x、y 值格式化为字符串，并以逗号分隔，添加到 m_strPerLine 中。如一条折线上有两个点，则 m_strPerLine 的值类似“132,432,532,342”，其中“132,432”为第一个点的 x，y 坐标值，“532,342”为第二个点的 x，y 坐标值。

（8）在类视图双击 OnMouseMove 项，添加如下代码：

```
void CSdi063View::OnMouseMove(UINT nFlags, CPoint point)
{
    CView::OnMouseMove(nFlags, point);
    if(m_nFlags==2)                              //若标识符为 2，绘线
    {
        if(m_ptStart.x==0 && m_ptEnd.x==0)       //若起止点为 0，表明尚未开始绘线
            return;
        SetCapture();                            //捕捉鼠标控制权
        CClientDC dc(this);                      //视图客户区的设备环境
        dc.SetROP2(R2_NOT);                      //设置绘图模式为反转色
        dc.MoveTo(m_ptStart);                    //用反色从起点和上一个终点绘制直线
        dc.LineTo(m_ptEnd);
        m_ptEnd=point;                           //当前点作为终点
        dc.MoveTo(m_ptStart);                    //从起点和当前终点绘制直线
        dc.LineTo(m_ptEnd);
    }
}
```

当鼠标在视图窗口上移动时，触发 WM_MOUSEMOVE 消息，自动调用该函数，参数 2 为当前鼠标位置。若 m_nFlags 值为 2，则执行绘线操作，若起止点为 0，则表明尚未开始绘线，直接返回。

SetCapture 函数设置当前窗口获取鼠标的控制权，不管鼠标在什么位置，都由当前窗口处理鼠标消息。SetROP2 函数设置绘图模式，格式如下：

```
int CDC::SetROP2(int nDrawMode)
```

参数如下。

- nDrawMode：新的绘图模式，如 R2_NOT 表示使用当前屏幕颜色的反色，若屏幕颜色为黑色，则使用白色绘图。

返回值：先前的绘图模式。

设置绘图模式为反转色后，先用反色从起点到上一个终点，绘制一条直线，由于起点和上一个终点间已经用反色绘制一条直线，再次用反色绘制后，相当于没有绘制直线，实现在鼠标移动时，擦除先前绘制的直线。

从起点到鼠标当前点绘制一条直线，并将当前点设为终点，以便鼠标再次移动时，擦除本

次绘制的直线。在鼠标交互式绘图中，绘图模式是关键技术，也称为“橡皮条技术”。

（9）在类视图双击 OnRButtonDown 项，添加如下代码：

```
void CSdi063View::OnRButtonDown(UINT nFlags, CPoint point)
{
    CView::OnRButtonDown(nFlags, point);
    if(m_nFlags==2)                          //若标识符为 2，结束绘线
    {
        ReleaseCapture();                    //释放鼠标控制权
        CString strX,strY;
        strX.Format("%d",point.x);           //将当前点坐标格式化为字符串
        strY.Format("%d",point.y);
        m_strPerLine+=strX+","+strY;         //当前点坐标添加到当前折线中
        m_listLine.Add(m_strPerLine);        //将当前折线值添加到动态数组中
        m_ptStart.x=m_ptStart.y=0;           //设置起止点坐标为 0
        m_ptEnd.x=m_ptEnd.y=0;
        m_strPerLine="";                     //清空折线字符串
    }
}
```

当鼠标在视图窗口按下右键时，触发 WM_RBUTTONDOWN 消息，自动调用该函数。若 m_nFlags 值为 2，则结束绘线操作。ReleaseCapture 函数释放当前窗口的鼠标控制权，并将当前点坐标值加入折线坐标值 m_strPerLine 中，结束当前折线的绘制。

m_strPerLine 记录当前折线所有点的 x，y 坐标值，按照顺序依次存放，用逗号分隔。Add 将折线坐标值以字符串形式添加到动态数组中，m_listLine 存放所有折线的坐标值。设置起止点的坐标为 0，清空折线坐标字符串，用于下一次绘制折线。

（10）在类视图用鼠标右键单击 CSdi063View 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加返回类型为 int 的函数 GetSingleStringNum（CString strMulti, CString strSplit），添加如下代码：

```
int CSdi063View::GetSingleStringNum(CString strMulti, CString strSplit)
{
    if(!strMulti.IsEmpty())                                    //若拼接字符串不为空
    {
        int nSplitCount=strMulti.Replace(strSplit,strSplit);  //获取分隔符个数
        return nSplitCount+1;                                  //返回子字符串个数
    }
    return 0;
}
```

GetSingleStringNum 函数获取拼接字符串中元素的数目，参数 1 为拼接字符串，如“a,b,c,d”，参数 2 为分隔符，如“,”，返回值为元素数目，如 4。IsEmpty 函数判断字符串是否为空，若不为空，Replace 函数用分隔符替换分隔符，得到分隔符的数目，格式如下：

```
int CString::Replace(LPCTSTR lpszOld,LPCTSTR lpszNew)
```

参数如下。

- lpszOld：被替换的字符串。
- lpszNew：新字符串。

返回值：替换的数目。

（11）在类视图用鼠标右键单击 CSdi063View 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加返回类型为 CString 的函数 GetSingleString（CString strMulti, int nIndex, CString strSplit），添加如下代码：

```
CString CSdi063View::GetSingleString(CString strMulti, int nIndex, CString strSplit)
{
```

```
        CString strAim="";
        if(!strMulti.IsEmpty())                                    //若拼接字符串不为空
        {
            int nSplitCount=strMulti.Replace(strSplit,strSplit);  //获取分隔符个数
            //若序号大于总数目，或序号小于 1，或分隔符为 0，返回空值
            if(nIndex>nSplitCount+1 || nIndex<1 || nSplitCount==0)
                return "";

            if(nIndex==1)                                          //若序号为 1
            {
                int nBegin=0;
                if((nBegin=strMulti.Find(strSplit,nBegin))!=-1)    //查找第一个分隔符
                {
                    strAim=strMulti.Left(nBegin); //获取第一个分隔符左边的字符串
                }
            }
            else if(nIndex==nSplitCount+1)         //若为最后一个序号
            {
                int nBegin=0,nPlace=0;
                while((nBegin=strMulti.Find(strSplit,nBegin))!=-1)//查找最后一个分隔符位置
                    nPlace=nBegin++;
                if(nPlace)                                 //获取最后一个分隔符右边的字符串
                    strAim=strMulti.Right(strMulti.GetLength()-nPlace-strSplit.
GetLength());
            }
            else                                           //若为中间序号
            {
                int nBeginPlacePre=0,nSplitIndex=0,nBeginPlaceNext=0;
                //从前往后查找分隔符
                while((nBeginPlacePre=strMulti.Find(strSplit,nBeginPlacePre))!=-1)
                {
                    nSplitIndex++;                         //分隔符数目递增
                    if(nSplitIndex==nIndex-1)              //若到指定序号
                    {
                        //查找下一个分隔符位置
                        if((nBeginPlaceNext=strMulti.Find(strSplit,nBeginPlace
Pre+1))!=-1)
                        {
                            //获取两个分隔符之间的字符串
                            strAim=strMulti.Mid(nBeginPlacePre+strSplit.GetLength()
                              ,nBeginPlaceNext-nBeginPlacePre-strSplit.GetLength());
                            return strAim;
                        }
                    }
                    else                                   //若没到指定序号，继续查找
                        nBeginPlacePre++;
                }
            }
        }
        return strAim;                                     //返回子字符串值
    }
```

GetSingleString 函数获取拼接字符串某个字符串的值，参数 1 为拼接字符串，如“a,b,c,d”，参数 2 为字符串索引（从 1 开始），如 2，参数 3 为分隔符，如“,”，返回值为指定位置的字符串，如“b”。

获取指定位置的元素，算法如下所示：

- 若参数 nIndex 超出元素数目或小于 0，或找不到分隔符，则返回空字符串。
- 若 nIndex 为 1，则获取第一个分隔符左边的子字符串。
- 若 nIndex 为最后一个序号，则获取最后一个分隔符右边的子字符串。

❑ 若 nIndex 为中间序号，则获取第 nIndex-1 个的分隔符和第 nIndex 个分隔符间的子字符串。

（12）在类视图双击 OnDraw 项，添加如下代码：

```
void CSdi063View::OnDraw(CDC* pDC)
{
    CSdi063Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rcClient;
    GetClientRect(rcClient);                                    //获取客户区矩形
    CDC drawDC;
    drawDC.CreateCompatibleDC(pDC);                             //创建与当前 DC 兼容的内存 DC
    CBitmap drawBmp;                                            //创建与当前 DC 兼容的位图
    drawBmp.CreateCompatibleBitmap(pDC,rcClient.Width(),rcClient.Height());
    CPen* pOldPen=drawDC.SelectObject(&m_pen);                  //将画笔选入内存 DC
    CBrush* pOldBrush=drawDC.SelectObject(&m_br);               //将画刷选入内存 DC
    drawDC.SelectObject(&drawBmp);                              //将兼容位图选入内存 DC
    int i=0;
    int j=0;
    CString strSplit=",";                                                   //坐标值分隔符
    for(i=0;i<m_listPoint.GetSize();i++)                                    //遍历点数组
    {
        CString strElem=m_listPoint.GetAt(i);                               //获取第 i 个点
        CString strX=GetSingleString(strElem,1,strSplit);                   //获取 x 坐标值
        CString strY=GetSingleString(strElem,2,strSplit);                   //获取 y 坐标值
        int nX=atoi(strX);                                                  //字符串转为整型值
        int nY=atoi(strY);
        drawDC.Ellipse(nX-3,nY-3,nX+3,nY+3);                    //内存 DC 上绘制圆，代表点
    }
    for(i=0;i<m_listLine.GetSize();i++)                                     //遍历线数组
    {
        CString strElem=m_listLine.GetAt(i);                                //获取第 i 个折线
        int number=GetSingleStringNum(strElem,strSplit);                    //获取坐标值数目
        LPPOINT pt=new POINT[number/2];                                     //当前折线包含的所有点
        for(j=0;j<number;j+=2)                                              //获取每个点的坐标值
        {
            CString strX=GetSingleString(strElem,j+1,strSplit);//获取点的 x、y 坐标值
            CString strY=GetSingleString(strElem,j+2,strSplit);
            pt[j/2].x=atoi(strX);                                           //字符串转为整型值
            pt[j/2].y=atoi(strY);
        }
        drawDC.Polyline(pt,number/2);                                       //内存 DC 上绘制折线
        delete [] pt;                                                       //释放动态创建的数组
    }
    drawDC.SelectObject(pOldPen);                                           //恢复原有画笔、画刷
    drawDC.SelectObject(pOldBrush);
    //将内存 DC 上绘制的图形，直接复制到当前 DC 上
    pDC->BitBlt(0,0,rcClient.Width(),rcClient.Height(),&drawDC,0,0,SRCCOPY);
}
```

在 OnLButtonDown、OnMouseMove 函数中绘制的图形都是临时的，当窗口被最小化，重新打开时，会消失不见，需要在 OnDraw 函数里，重绘已绘制的图形，保持图形始终存在。

GetClientRect 函数获取视图客户区的矩形大小，存放到 rcClient 中。CreateCompatible DC 函数用于创建一个与当前 DC 兼容的内存 DC，可在内存设备环境 drawDC 中完成绘图操作，绘制完成后，直接将图形复制到当前 DC 上，以减少屏幕的闪烁。

CreateCompatibleBitmap 函数创建一个与当前 DC 兼容的位图，格式如下：

```
BOOL CBitmap::CreateCompatibleBitmap(CDC* pDC,int nWidth,int nHeight)
```

参数如下。

- pDC：兼容的设备环境指针。
- nWidth：位图的宽度。
- nHeight：位图的高度。

返回值：若成功则返回非零值，否则返回 0。

SelectObject 函数将画笔、画刷、位图选入内存 DC，并保存原有的画笔、画刷，在内存 DC 中的绘制操作需要使用位图，以便将绘图的图形复制到当前 DC 中。

GetSize 函数获取点数组的元素数目，利用 for 循环遍历点数组，strElem 存放每个点元素的坐标值，GetSingleString 函数根据分隔符和索引获取 x、y 的值，atoi 函数将字符串转换为整型值，Ellipse 函数根据坐标点绘制一个圆，代表点。

第二个 for 循环遍历线数组，strElem 存放每个折线的坐标值，GetSingleStringNum 函数获取所有坐标值的数目，其中两个值代表一个点，根据点数使用 new 动态创建一个点数组 pt，GetSingleString 函数获取当前折线上每个点的坐标值，存放到 pt 中，Polyline 函数根据点数组绘制折线，绘制完成后，使用 delete 释放动态创建的数组。

在内存 DC 上绘制完成后，使用 SelectObject 函数将旧画笔、旧画刷重新选入设备环境，BitBlt 函数将内存 DC 上绘制的图形复制到当前 DC 上。

（13）生成程序并运行，如图 7-11 所示。单击“绘点”按钮，在客户区绘制点，单击“绘线”按钮，在客户区按下左键开始绘制折线，左击一次添加一个点，用鼠标右键单击鼠标结束绘制。将窗口最小化，重新显示，图形仍存在。

7.4 状态栏

状态栏作为控制条的一种，一般停靠在程序窗口的底部，不可移动，常用来显示各种提示信息，如鼠标经过一个工具按钮或菜单项时，会在状态栏显示功能提示信息。不同于工具栏，状态栏只有一个，可修改系统自带的状态栏，显示特定信息。

7.4.1 设置分区

【实例 7-5】新建一个单文档工程名为 Sdi064，在状态栏同步显示鼠标坐标位置和颜色信息。

（1）新建单文档工程 Sdi064，在资源视图中双击 String Table 项，打开字符串表，如图 7-12 所示。

图 7-11　交互式绘制点线

ID_INDICATOR_EXT	59136	扩展名
ID_INDICATOR_CAPS	59137	大写
ID_INDICATOR_NUM	59138	数字
ID_INDICATOR_SCRL	59139	滚动
ID_INDICATOR_OVR	59140	改写
ID_INDICATOR_REC	59141	记录
ID_INDICATOR_MOUSE	59142	鼠标位置
ID_INDICATOR_COLOR	59143	当前颜色值
ID_VIEW_TOOLBAR	59392	显示或隐藏工具栏\n显隐工具栏
ID_VIEW_STATUS_BAR	59393	显示或隐藏状态栏\n显隐状态栏

图 7-12　字符串表

（2）定位到 ID_INDICATOR_CAPS 项，用鼠标右键单击，在弹出的快捷菜单中选择 New

String 命令，弹出 String Properties 窗口，在 ID 组合框中输入 ID_INDICATOR_MOUSE，在 Caption 编辑框中输入“鼠标位置”。用同样的方式，再添加一个字符串，ID 为 ID_INDICATOR_ COLOR，Caption 为“当前颜色值”。

状态栏可分为多个窗格，也称为“指示器”，系统自带的指示器有 3 个：大写、数字、滚动，当按下 Caps Lock 键时，指示器窗口显示“大写”，按下 Num Lock 键后，显示“数字”，按下 Scroll Lock 键后，显示“滚动”，这三个指示器由 AppWizard 自动创建。

（3）在类视图双击 CMainFrame 类下的 CMainFrame 项，打开 CPP 文件，在文件开头处找到 indicators 数组，添加两个字符串 ID，代码如下：

```
static UINT indicators[] =
{
    ID_SEPARATOR,
    ID_INDICATOR_MOUSE,        //新增
    ID_INDICATOR_COLOR,        //新增
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

indicators 数组包含状态栏使用的所有指示器，ID_SEPARATOR 表示分隔符，该项是可伸缩的，用来填充状态栏的剩余空间，常用来显示鼠标提示信息。所有指示器按照顺序，依次显示在状态栏中。

（4）在类视图双击 CMainFrame 类下的 OnCreate 项，定位到函数，找到如下代码：

```
/*创建状态栏窗口*/
//设置指示器
if   (!m_wndStatusBar.Create   (this)||!m_wndStatusBar.SetIndicators(indicators,
sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return -1;
}
```

创建一个状态栏窗口，步骤如下：

（1）构造一个 CStatusBar 类对象。

（2）调用 Create 或 CreateEx 函数，创建一个状态栏实例，并关联到类对象。

（3）调用 SetIndicators 函数，设置状态栏的指示器。

Create 函数用于创建一个状态栏实例，并关联到类对象，格式如下：

```
BOOL CStatusBar::Create(CWnd* pParentWnd,DWORD dwStyle=WS_CHILD | WS_VISIBLE |
CBRS_BOTTOM,UINT nID=AFX_IDW_STATUS_BAR)
```

参数如下。

- pParentWnd：父窗口的指针。
- dwStyle：状态条样式，如 CBRS_BOTTOM 表示在底部显示。
- nID：子窗口的 ID。

返回值：若成功则返回非零值，否则返回 0。

SetIndicators 函数设置状态栏使用的指示器数组，格式如下：

```
BOOL CStatusBar::SetIndicators(const UINT* lpIDArray,int nIDCount)
```

参数如下。

- lpIDArray：指示器 ID 数组。
- nIDCount：数组的元素数目，如 sizeof(indicators)/sizeof(UINT)，用总大小除以单个元

素大小，得到元素数目。

返回值：若成功则返回非零值，否则返回 0。

（5）在 OnCreate 函数的 return 0; 前，添加如下代码：

```
    int nPosIndex=m_wndStatusBar.CommandToIndex(ID_INDICATOR_MOUSE);  //获取窗口索引
    int nColorIndex=m_wndStatusBar.CommandToIndex(ID_INDICATOR_COLOR);
    m_wndStatusBar.SetPaneInfo(nPosIndex,ID_INDICATOR_MOUSE,SBPS_NORMAL,100); //设置窗
口宽度
    m_wndStatusBar.SetPaneInfo(nColorIndex,ID_INDICATOR_COLOR,SBPS_NORMAL,150);
```

CommandToIndex 函数获取指定 ID 的窗格的位置索引（从 0 开始），格式如下：

```
int CStatusBar::CommandToIndex(UINT nIDFind) const
```

参数如下。

- nIDFind：指示器的 ID。

返回值：指示器所在窗格的位置索引。

SetPaneInfo 函数用来设置窗格的 ID、样式、宽度值，格式如下：

```
void CStatusBar::SetPaneInfo(int nIndex,UINT nID,UINT nStyle,int cxWidth)
```

参数如下。

- nIndex：要修改的指示器窗格的位置索引。
- nID：新 ID 值。
- nStyle：新的样式，一般用 SBPS_NORMAL。
- cxWidth：新的宽度。

调用 CommandToIndex 函数，根据 ID 值获取位置索引，相对于直接使用索引，更安全，当状态栏改变后，仍然可用。SetPaneInfo 函数设置两个新增窗格的宽度。

类似于菜单和工具条，若没有添加消息处理函数，则默认状态下，菜单项和工具按钮不可用。状态栏指示器也需要添加用户接口 UI 函数，设置指示器窗口是否可用。但类向导功能有限，无法添加 UI 函数，需要手动添加相关代码。

（6）在类视图双击 CMainFrame 项，在类定义中添加 UI 函数声明，代码如下：

```
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnUpdatePos(CCmdUI* pCmdUI);          //新增函数声明
    afx_msg void OnUpdateColor(CCmdUI* pCmdUI);        //新增函数声明
```

（7）双击 CMainFrame 类下的 CMainFrame 项，在文件开头处找到 BEGIN_MESSAGE_ MAP，添加两个消息映射，代码如下：

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //省略…
    ON_WM_CREATE()
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_MOUSE,OnUpdatePos)       //新增函数映射
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_COLOR,OnUpdateColor)     //新增函数映射
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

ON_UPDATE_COMMAND_UI 宏用于 UI 函数的消息映射，参数 1 为指示器 ID，参数 2 为消息函数名，指定处理用户界面消息的函数。

（8）在当前 CPP 文件尾部，添加如下代码：

```
void CMainFrame::OnUpdatePos(CCmdUI* pCmdUI)       //鼠标位置窗格 UI 函数
{
    pCmdUI->Enable();                              //设置窗格可用
```

```
}

void CMainFrame::OnUpdateColor(CCmdUI* pCmdUI)      //当前颜色值窗格 UI 函数
{
    pCmdUI->Enable();                               //设置窗格可用
}
```

（9）生成程序并运行，如图 7-13 所示。在状态栏新增两个窗格，显示指示器 ID 对应的字符串。

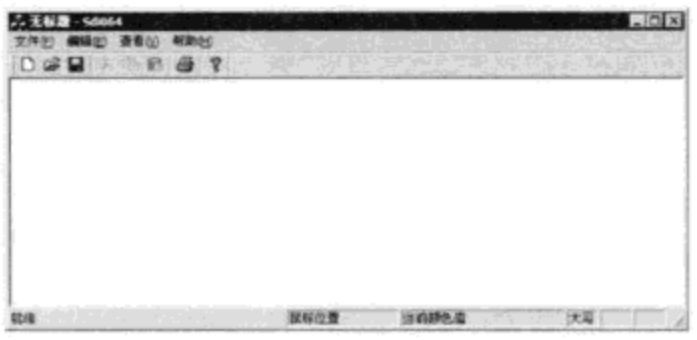


图 7-13 状态栏分区

7.4.2 更新内容

在程序运行时，可动态更新状态栏窗格显示的文本，用于显示各种提示信息，如当前鼠标位置、当前位置的颜色信息等。SetPaneText 函数设置窗格显示的文本，格式如下：

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

参数如下。

- nIndex：要更新的窗格的位置索引。
- lpszNewText：新文本值。
- bUpdate：是否立即更新，默认为 TRUE。

返回值：若成功则返回非零值，否则返回 0。

（1）在类视图用鼠标右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Member Function 项，添加返回类型为 void 的函数 SetStatusText(UINT nID, CString strText)，添加如下代码：

```
void CMainFrame::SetStatusText(UINT nID, CString strText)
{
    int nIndex=m_wndStatusBar.CommandToIndex(nID);          //获取指定 ID 窗格的位置索引
    m_wndStatusBar.SetPaneText(nIndex,strText);             //更新文本值
}
```

由于 m_wndStatusBar 变量为 protected 类型，无法在类外访问，因此添加一个 public 类型的成员函数，以便在类外更新内容。SetStatusText 函数用于更新指定 ID 窗格的文本，参数 1 为窗格 ID，参数 2 为新的文本值。CommandToIndex 函数根据指示器 ID 获取窗格的位置索引，SetPaneText 函数根据位置索引、新文本值，更新窗格的显示文本。

（2）在类视图双击 CSdi064View 类下的 OnDraw 项，添加如下代码：

```
void CSdi064View::OnDraw(CDC* pDC)
{
    CSdi064Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rcClient;
    GetClientRect(rcClient);                                    //获取客户区矩形
    rcClient.DeflateRect(rcClient.Width()/4,rcClient.Height()/4);//矩形对象尺寸缩小一半
    for(int i=rcClient.left;i<rcClient.right;i++)               //绘制图形
    {
        for(int j=rcClient.top;j<rcClient.bottom;j++)
            pDC->SetPixel(i,j,RGB(i,j,0));                      //设置各个点的像素值
    }
}
```

（3）在当前 CPP 文件的开头处，添加一句#include “MainFrm.h”，包含 CMainFrame 类的头文件。

（4）在类视图用鼠标右键单击 CSdi064View 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_MOUSEMOVE、WM_SETCURSOR 两个消息的处理函数。

（5）双击 CSdi064View 类下的 OnMouseMove 项，添加如下代码：

```
void CSdi064View::OnMouseMove(UINT nFlags, CPoint point)
{
    CString strPos;
    strPos.Format("X:%d,Y:%d",point.x,point.y);                    //将坐标值格式化为字符串

    CClientDC dc(this);                                            //获取视图客户区的 DC
    COLORREF color=dc.GetPixel(point);                             //获取当前点的颜色值
    CString strColor;
    strColor.Format("红:%d,绿:%d,蓝:%d",GetRValue(color),GetGValue(color)
        ,GetBValue(color));                    //获取 RGB 颜色值，格式化后存入 strColor 中

    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();               //获取主窗口指针
    pFrame->SetStatusText(ID_INDICATOR_MOUSE,strPos);              //显示当前坐标
    pFrame->SetStatusText(ID_INDICATOR_COLOR,strColor);            //显示颜色信息

    CView::OnMouseMove(nFlags, point);
}
```

Format 函数将鼠标坐标值格式化为字符串，存放到 strPos 中。GetPixel 函数获取当前位置的颜色值，利用 GetRValue、GetGValue、GetBValue 宏获取颜色的 RGB 值，并格式化为字符串，存放到 strColor 中。

AfxGetMainWnd 函数获取框架主窗口的指针，强制转换为 CMainFrame 类指针。SetStatusText 函数为自定义函数，用来设置指定 ID 窗格的文本值。当鼠标在视图窗口移动时，在状态栏中显示位置和颜色信息。

（6）双击 CSdi064View 类下的 OnSetCursor 项，添加如下代码：

```
BOOL CSdi064View::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    if(nHitTest==HTCLIENT)                                         //若鼠标在客户区中
        SetCursor(AfxGetApp()->LoadStandardCursor(IDC_CROSS));     //设置鼠标样式
    return TRUE;                                                   //停止处理
}
```

当鼠标进入视图窗口，且没有被捕获控制权时，自动调用该函数，设置鼠标样式。参数 1 为包含鼠标的窗口指针，参数 2 为击中测试区域代码，参数 3 为鼠标消息。

若鼠标在客户区中，调用 LoadStandardCursor 函数加载系统标准光标，格式如下：

```
HCURSOR CWinApp::LoadStandardCursor(LPCTSTR lpszCursorName) const
```

参数如下。

- lpszCursorName：标准的 Windows 光标标识符。

返回值：光标的句柄。

SetCursor 函数设置窗口的鼠标样式，格式如下：

```
HCURSOR SetCursor(HCURSOR hCursor)
```

参数如下。

- hCursor：新光标的句柄。

返回值：先前光标的句柄。

（7）生成程序并运行，如图 7-14 所示。在视图客户区绘制图形，移动鼠标，在状态栏中分别显示当前鼠标位置、鼠标所在位置的 RGB 颜色值，同时鼠标变成十字形。

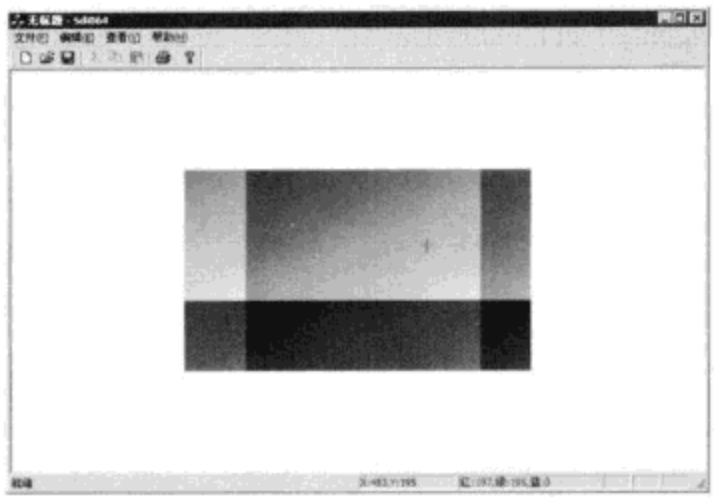


图 7-14　状态栏更新内容

7.5 对话栏

对话栏可看做一种可停靠的非模态窗口，是控制条的一种类型。可在对话栏中添加各种控件，为控件添加消息处理函数，相对于非模态窗口，不同之处在于对话栏不能为控件添加映射变量，只能通过控件 ID 获取窗口指针，再转换为控件类指针。对话栏可以像工具栏一样在框架窗口停靠。

7.5.1 添加对话栏资源

【实例 7-6】新建一个单文档工程名为 Sdi065，添加一个对话栏，用于设置视图中显示文本的字体、大小、颜色。

（1）新建单文档工程 Sdi065，在资源视图用鼠标右键单击，在弹出的快捷菜单中选择 Insert 命令，弹出“Insert Resource”窗口，选择 Dialog 节点下的 IDD_DIALOGBAR 项，单击“New”按钮添加一个对话栏资源，如图 7-15 所示。

（2）选择 Layout|Guide Settings 菜单命令，弹出“Guide Settings”窗口，选择 None 项，取消对话栏模板的标尺线，单击“OK”按钮保存并退出，如图 7-16 所示。

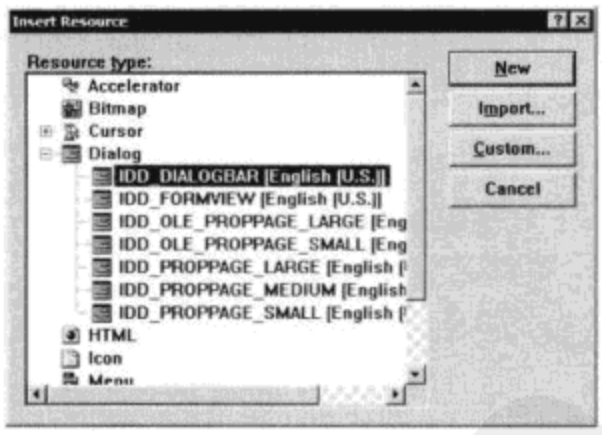


图 7-15　添加对话栏资源

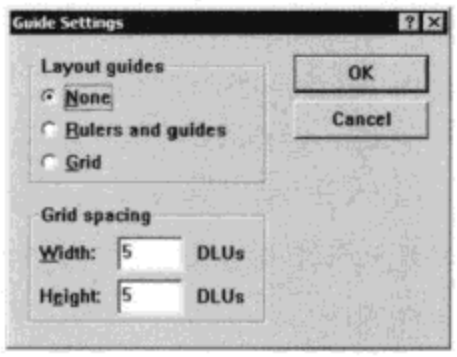


图 7-16　取消标尺线

（3）拖放两个组合框、一个按钮到对话栏模板上，如图 7-17 所示。

（4）设置第 1 个组合框的 ID 为 IDC_COMBO_FONT，Type 为 Drop List。设置第 2 个组合框的 ID 为 IDC_COMBO_SIZE，取消勾选 Sort 复选框。设置按钮的 ID 为 IDC_ BUTTON_COLOR，Caption 为空。

Tips　添加的对话栏默认使用 English (U.S.)语言，不能显示中文。若要显示中文，用鼠标右键单击 Dialog 节点下的 IDD_DIALOGBAR 项，在弹出的快捷菜单中选择 Properties 命令，弹出“Dialog Properties”窗口，如图 7-18 所示。在 Language 组合框中选择 Chinese (P.R.C.)项。

图 7-17　对话栏添加控件

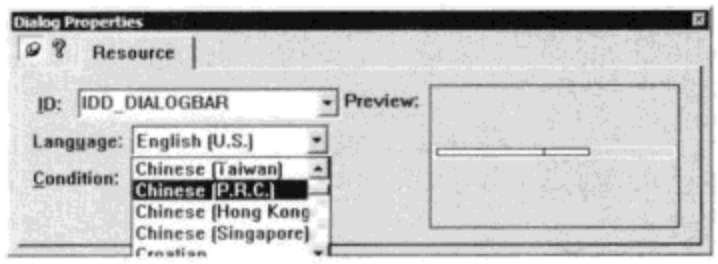


图 7-18　设置对话栏的语言

7.5.2　显示对话栏

创建并显示一个对话栏，步骤如下：

（1）添加一个对话栏资源。

（2）构造一个 CDialogBar 对象。

（3）调用 Create 函数，创建对话栏窗口实例。

（4）调用 EnableDocking 函数，设置对话栏停靠方式。

（5）调用 DockControlBar 函数，实现对话栏停靠。

在实例的具体实现如下：

（1）在类视图用鼠标右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个 CDialogBar 类型的变量 m_wndFontDlg。

（2）在类视图双击 CMainFrame 类下的 OnCreate 项，在 return 0; 前添加如下代码：

```
    if(!m_wndFontDlg.Create(this,IDD_DIALOGBAR,CBRS_TOP,AFX_IDW_CONTROLBAR_LAST-1))
      return -1;                                                    //创建对话栏实例
    m_wndFontDlg.EnableDocking(CBRS_ALIGN_TOP|CBRS_ALIGN_BOTTOM); //设置停靠方式
    DockControlBar(&m_wndFontDlg);                                 //停靠对话栏
```

Create 函数用于创建一个对话栏实例，格式如下：

```
BOOL CDialogBar::Create(CWnd* pParentWnd,LPCSTR lpszTemplateName,UNIT nStyle,UNIT nID)
```

参数如下。

- pParentWnd：父窗口的指针。
- lpszTemplateName：对话栏模板 ID。
- nStyle：停靠方式，如 CBRS_TOP 表示在顶部停靠。
- nID：对话栏的控制 ID。

返回值：若成功则返回非零值，否则返回 0。

AFX_IDW_CONTROLBAR_LAST 是系统自带的 ID，表示对话栏 ID 的最大值。EnableDocking 函数设置对话栏的停靠方式，可在顶部、底部停靠。DockControlBar 函数实现对话栏的停靠。

（3）生成程序并运行，在框架窗口顶部出现新增的对话栏窗口，可拖至窗口底部停靠。

7.5.3　添加控件处理函数

（1）在类视图双击 CSdi065View 项，在类定义中，添加成员变量，代码如下：

```
public:
    int m_nFontSize;                    //字体大小
    CString m_strFontName;              //字体名称
    COLORREF m_clrChoose;               //字体颜色
    CFont m_font;                       //字体对象
```

（2）在类视图用鼠标右键单击 CSdi065View，在弹出的快捷菜单中选择 Add Member Function

命令，添加返回类型为 void 的成员函数 SetBtnText（COLORREF m_clrChoose），添加如下代码：

```
void CSdi065View::SetBtnText(COLORREF m_clrChoose)
{
    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();  //获取主窗口指针
    CButton*    pBtnColor=(CButton*)(pFrame->m_wndFontDlg.GetDlgItem(IDC_BUTTON_
COLOR));
                                                      //获取按钮控件指针
    BYTE red=GetRValue(m_clrChoose);                  //获取颜色的 RGB 分量值
    BYTE green=GetGValue(m_clrChoose);
    BYTE blue=GetBValue(m_clrChoose);
    CString strColor;
    strColor.Format("R:%d G:%d B:%d",red,green,blue); //将 RGB 值格式化为字符串
    pBtnColor->SetWindowText(strColor);               //颜色字符串作为按钮文本
}
```

AfxGetMainWnd 函数获取主窗口指针，强制转换为 CMainFrame 类型。对话栏中的控件不能添加映射变量，只能通过 pFrame->m_wndFontDlg.GetDlgItem 方式，先获取主窗口中的对话栏对象，再调用 GetDlgItem 函数获取指定 ID 的窗口指针，并强制转换为控件类指针。

GetRValue、GetGValue、GetBValue 宏获取颜色的 RGB 分量值，Format 函数将 RGB 值格式化为字符串，SetWindowText 函数将颜色字符串设为按钮文本。

（3）在当前 CPP 文件的开头处，添加一句#include "MainFrm.h" ，包含 CMainFrame 类的头文件。

（4）在当前 CPP 文件的末尾空白处，添加如下代码：

```
int CALLBACK GetFontList(ENUMLOGFONT FAR* pEachLogFont,NEWTEXTMETRIC FAR* pText,
    int nType,LPARAM lParam)
{
    CComboBox* pComboFont=(CComboBox*)lParam;              //自定义值转换为控件指针
    CString strFontName=pEachLogFont->elfLogFont.lfFaceName;  //获取当前字体名称
    pComboFont->AddString(strFontName);                   //将字体名称添加到组合框中
    return TRUE;
}
```

CALLBACK 表明该函数是一个回调函数，回调函数是一个全局函数，由系统自动调用，必须符合指定的函数原型。GetFontList 函数用于获取当前字体的名称，其指定的函数原型如下所示：

```
typedef int (CALLBACK* OLDFONTENUMPROCA)(CONST LOGFONTA *, CONST TEXTMETRICA *, DWORD,
LPARAM);
```

参数 1 为当前字体的逻辑属性，参数 2 为物理字体信息，参数 3 为字体类型，参数 4 为传入的自定义值。lParam 是用户自定义值，传入时转换为 LPARAM 类型，在回调函数中再恢复为原始类型。

pEachLogFont->elfLogFont.lfFaceName 获取当前字体的名称，AddString 函数将字体名添加到组合框中，lParam 实际为组合框控件的指针，在内部转换为 CComboBox*类型。若返回 TRUE，则继续下一次枚举，否则停止枚举字体。

（5）用鼠标右键单击 CSdi065View 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CSdi065View::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class
    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();      //获取主窗口指针
```

```
        CComboBox*  pComboFont=(CComboBox*)(pFrame->m_wndFontDlg.GetDlgItem(IDC_COMBO
_FONT));
        //获取两个组合框的指针
        CComboBox*
pComboSize=(CComboBox*)(pFrame->m_wndFontDlg.GetDlgItem(IDC_COMBO_SIZE));
        CClientDC dc(this);        //获取视图客户区的 DC，并枚举所有字体，添加到字体组合框中
        EnumFontFamilies(dc.GetSafeHdc(),NULL,(FONTENUMPROC)GetFontList,(LPARAM)pCom
boFont);
        pComboFont->SetCurSel(pComboFont->GetCount()-1);        //选中最后一项
        CString strSize;
        for(int i=20;i<=400;i+=20)                              //字体大小组合框添加项
        {
            strSize.Format("%d",i);
            pComboSize->AddString(strSize);
        }
        pComboSize->SetCurSel(pComboSize->GetCount()/2);        //选择中间项
        m_clrChoose=RGB(20,20,200);                             //设置颜色初始值
        SetBtnText(m_clrChoose);                                //根据颜色值，设置按钮文本
        pComboFont->GetLBText(pComboFont->GetCurSel(),m_strFontName); //获取选择的字体名称
        pComboSize->GetLBText(pComboSize->GetCurSel(),strSize);    //获取选择的字体大小
        m_nFontSize=atoi(strSize);                                 //字符串转为整型值
        m_font.CreatePointFont(m_nFontSize,m_strFontName,&dc);     //创建字体实例
    }
```

EnumFontFamilies 是一个枚举函数，用于获取系统所有字体，格式如下：

```
    int  EnumFontFamilies(HDC  hdc,LPCTSTR  lpszFamily,FONTENUMPROC  lpEnumFontFamProc,
LPARAM lParam)
```

参数如下。

- hdc：设备环境的句柄值。
- lpszFamily：要获取的字体，若为 NULL，则获取所有字体。
- lpEnumFontFamProc：回调函数，处理获取的字体。
- lParam：附加信息，传入回调函数中。

返回值：回调函数的最后一个返回值。

EnumFontFamilies 函数枚举系统所有字体，每获取一个字体，自动调用一次 GetFontList 函数。最后一个参数（LPARAM）pComboFont 为组合框控件的指针值，转换为 LPARAM 类型，传入回调函数中，在回调函数中将字体名称添加到组合框中。SetCurSel 函数设置选中最后一项。

SetBtnText 函数根据颜色值，设置按钮的标题文本。GetLBText 函数获取选择的字体、大小，存放到 m_strFontName、strSize 中。atoi 函数将字符串类型的字体大小转换为整型值。CreatePointFont 函数根据字体名称、字体大小、当前 DC，创建一个简单的字体实例。

（6）双击 CSdi065View 类下的 OnDraw 项，添加如下代码：

```
    void CSdi065View::OnDraw(CDC* pDC)
    {
        CSdi065Doc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // TODO: add draw code for native data here
        CRect rcClient;
        GetClientRect(rcClient);                                //获取客户区矩形
        CString strText="\n 旅行\n\n"                           //输出文本
            "只有青山藏在白云间\n"
            "蝴蝶自由穿行在清涧\n"
            "看那晚霞盛开在天边\n"
            "有一群向西归鸟...";
        CFont* pOldFont=pDC->SelectObject(&m_font);             //将新字体选入 DC
        pDC->SetTextColor(m_clrChoose);                         //设置文本颜色
```

```
    pDC->DrawText(strText,rcClient,DT_CENTER);          //在客户区矩形中输出多行文本
    pDC->SelectObject(pOldFont);                        //恢复原有字体
}
```

若字符串过长，无法在一行中显示，可拆分为多行字符串，不需要使用加号连接。SelectObject 函数将新字体选入 DC，同时保存旧字体。SetTextColor 函数设置文本前景色。DrawText 函数在矩形中输出多行文本，DT_CENTER 表示文本居中对齐显示。

（7）在资源视图双击 IDD_DIALOGBAR 项，按 Ctrl+W 组合键打开类向导窗口，系统自动弹出“Add a class”窗口，单击“Cancel”按钮关闭窗口。选择 Message Maps 选项卡，Class name 组合框选择 CSdi065View 项，Object IDs 列表框选择 IDC_BUTTON_COLOR 项，Messages 列表框选择 BN_CLICKED 项，单击“Add Function”按钮添加消息处理函数。

（8）Object IDs 列表框选择 IDC_COMBO_FONT 项，Messages 列表框选择 CBN_SELCHANGE 项，单击“Add Function”按钮添加函数。用同样方式，为 IDC_COMBO_SIZE 添加函数，单击“OK”按钮保存并退出。

Tips 只有当对话栏模板窗口显示时，类向导窗口中才出现对话栏控件的 ID。

（9）在类视图双击 OnButtonColor 项，添加如下代码：

```
void CSdi065View::OnButtonColor()
{
    CColorDialog colorDlg(m_clrChoose,CC_ANYCOLOR,this);  //系统自带的颜色对话框
    if(colorDlg.DoModal()==IDOK)                          //显示模态颜色对话框
    {
        m_clrChoose=colorDlg.GetColor();                  //获取选择的颜色值
        SetBtnText(m_clrChoose);                          //设置按钮的文本
        Invalidate();                                     //刷新视图
    }
}
```

CColorDialog 是系统自带的颜色对话框，用于选择颜色值，构造函数格式如下：

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

参数如下。

- clrInit：初始颜色值。
- dwFlags：对话框外观标志，如 CC_ANYCOLOR 表示显示所有可用颜色。
- pParentWnd：父窗口的指针。

DoModal 函数以模态形式显示颜色对话框，若单击“OK”按钮，返回 IDOK，表示成功选择一个颜色值。GetColor 函数获取选择的颜色值，存放到 m_clrChoose 中。SetBtnText 函数设置按钮的显示文本。Invalidate 函数强制重绘视图窗口。

（10）在类视图双击 OnSelchangeComboFont 项，添加如下代码：

```
void CSdi065View::OnSelchangeComboFont()
{
    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();      //获取主窗口的指针
    CComboBox*
pComboFont=(CComboBox*)(pFrame->m_wndFontDlg.GetDlgItem(IDC_COMBO_FONT));
    pComboFont->GetLBText(pComboFont->GetCurSel(),m_strFontName);  //获取选择的字体名
    CClientDC dc(this);                                   //获取视图客户区的 DC
    m_font.DeleteObject();                                //释放已有字体实例
    m_font.CreatePointFont(m_nFontSize,m_strFontName,&dc);          //创建新字体
    Invalidate();                                                   //刷新视图
}
```

GetLBText 函数获取字体组合框的选择字体的名称。DeleteObject 函数释放 m_font 对象已有的字体实例，在用同一个对象创建另一个 GDI 实例前，应先释放已有实例。CreatePointFont 函数创建一个新的字体实例。

（11）在类视图双击 OnSelchangeComboSize 项，添加如下代码：

```
void CSdi065View::OnSelchangeComboSize()
{
    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();                    //获取主窗口指针
    CComboBox*      pComboSize=(CComboBox*)(pFrame->m_wndFontDlg.GetDlgItem(IDC_
COMBO_SIZE));
    CString strSize;
    pComboSize->GetLBText(pComboSize->GetCurSel(),strSize);    //获取选择的字体大小
    m_nFontSize=atoi(strSize);                                          //转换为整型值
    CClientDC dc(this);
    m_font.DeleteObject();                                              //释放已有字体实例
    m_font.CreatePointFont(m_nFontSize,m_strFontName,&dc);     //创建新字体
    Invalidate();
}
```

（12）生成程序并运行，如图 7-19 所示。在框架窗口顶部停靠有自定义的对话栏，第 1 个组合框显示系统所有字体，第 2 个组合框用于选择字体大小，单击按钮，弹出“颜色”对话框，如图 7-20 所示。当字体名称、字体大小、字体颜色其中一项改变后，视图中的文本样式随之改变。

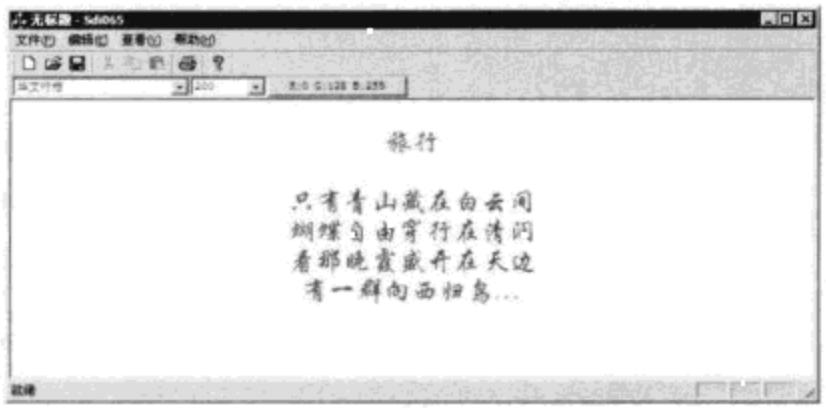


图 7-19　对话栏设置字体样式

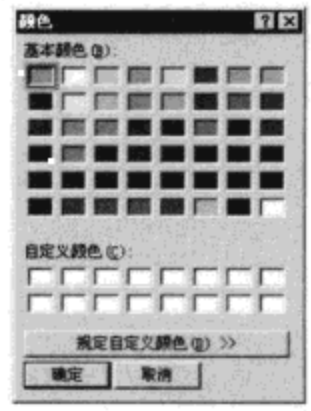


图 7-20　“颜色”对话框

7.6 文档视图

文档视图体现了一种思想：分离数据和显示，数据的存取和界面显示是两个单独的功能，便于维护和升级。Windows 本身并不支持文档/视图结构，MFC 通过构造一系列类，如文档类、视图类、框架类、文档模板类等，搭建了一个文档/视图框架，在该框架基础上，只需要添加部分功能，就可以开发出强大的文件处理程序。

7.6.1 文档类存取数据

【实例 7-7】新建一个单文档程序名为 Sdi066，在文档类中读取和保存一组号码，在视图类中显示和编辑号码。

（1）新建单文档工程 Sdi066，在资源视图双击 Menu 节点下的 IDR_MAINFRAME 项，打开菜单资源，单击“编辑”菜单，按下 Delete 键移除该菜单项，“文件”菜单下只保留“打开”、“保存”、“退出”三个子菜单，如图 7-21 所示。

（2）双击 Toolbar 节点下的 IDR_MAINFRAME 项，打开工具栏资源，只保留三个按钮图标，其余图标用鼠标拖曳到窗口外，如图 7-22 所示。

（3）在类视图用鼠标右键单击 CSdi066Doc 项，在弹出的快捷菜单中选择 Add Member

Variable 命令，添加一个 int 类型的数组变量 m_num[7]。

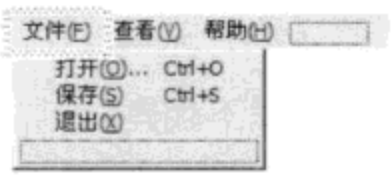


图 7-21 编辑菜单资源

图 7-22 编辑工具栏资源

（4）在类视图双击 CSdi066Doc 下的 CSdi066Doc 项，添加如下代码：

```
CSdi066Doc::CSdi066Doc()                    //文档类构造函数
{
    for(int i=0;i<7;i++)                    //初始化数组元素
        m_num[i]=0;
}
```

（5）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CSdi066Doc 项，Object IDs 列表框选择 ID_FILE_OPEN 项，Messages 列表框选择 COMMAND 项，单击“Add Function”按钮添加函数。用同样的方式，为 ID_FILE_SAVE 添加添加函数，单击“OK”按钮保存并退出。

（6）在类视图双击 CSdi066Doc 下的 OnFileSave 项，添加如下代码：

```
void CSdi066Doc::OnFileSave()
{
    CFileDialog dlgFile(FALSE,"ball","双色球号码",NULL,"ball 文件 (*.ball)|*.ball||");
                                                          //保存文件对话框
    if(dlgFile.DoModal()==IDOK)                           //若单击“OK”按钮
    {
        CString strPath=dlgFile.GetPathName();           //获取文件路径
        CStdioFile f(strPath,CFile::modeCreate|CFile::modeWrite); //构造文件类对象
        CString strTemp;
        for(int i=0;i<7;i++)                              //遍历数组元素
        {
            strTemp.Format("%d\n",m_num[i]);              //整型号码格式化为字符串
            f.WriteString(strTemp);                       //写入文件中
        }
    }
}
```

CFileDialog 封装了系统自带的文件对话框，其构造函数中参数 1 若为 FALSE，则用于保存文件，若为 TRUE，则用于打开文件，参数 2 为默认的扩展名，参数 3 为默认的文件名，参数 4 为对话框标志组合，参数 5 为文件类型过滤字符串，如"ball 文件 (*.ball)|*.ball||"中，”ball 文件 (*.ball)”为文件对话框显示的文件类型提示信息，”*.ball”为文件匹配符，只显示后缀名为 ball 的文件。

Tips 后缀名与文件内容无关，只要文件格式和内容没有改变，不论后缀名如何改变，用相应程序仍能打开文件，如将 Word 文件的原有后缀名.doc，修改为.jpg 或.rmvb 或直接删除后，只是不能用双击方式打开文件，可将文件拖放到 Word 软件中，仍能打开。后缀名的作用是将文件关联到一个程序，双击文件时，根据后缀名自动调用该程序打开文件，或在文件对话框中根据后缀名过滤文件。

DoModal 函数以模态形式显示文件对话框，若单击“OK”按钮或双击文件名，DoModal 函数返回 IDOK，表示成功选择一个文件。GetPathName 函数获取选择文件的路径，存放到 strPath 中。

CStdioFile 类用于对文本文件进行读/写操作，其构造函数格式如下：

```
CStdioFile::CStdioFile(LPCTSTR lpszFileName,UINT nOpenFlags)
```

参数如下。

- lpszFileName：文件路径，相对或绝对路径。
- nOpenFlags：文件操作标志，如 CFile::modeCreate 表示创建一个新文件，CFile::modeRead 表示以只读形式打开文件，CFile::modeReadWrite 表示以可读写形式打开文件。

WriteString 函数用于在文件中写入字符串，使用“\n”实现换行存储，格式如下：

```
void CStdioFile::WriteString(LPCTSTR lpsz)
```

参数如下。

- lpsz：要写入的字符串。

（7）在当前 CPP 文件的开头处，添加一句#include "Sdi066View.h"，包含 CSdi066View 类的头文件。

（8）在类视图双击 CSdi066Doc 下的 OnFileOpen 项，添加如下代码：

```
void CSdi066Doc::OnFileOpen()
{
    CFileDialog dlgFile(TRUE,"ball",NULL,NULL,"ball 文件 (*.ball)|*.ball||"); //打开文件对话框
    if(dlgFile.DoModal()==IDOK)                                    //若单击“OK”按钮
    {
        CString strPath=dlgFile.GetPathName();                     //获取文件路径
        CStdioFile f(strPath,CFile::modeRead);                     //构造文件类对象
        CString strTemp;
        for(int i=0;i<7;i++)                                       //遍历数组元素
        {
            f.ReadString(strTemp);                                 //读取一行数据
            m_num[i]=atoi(strTemp);                                //将字符串转为整型值
        }
        POSITION pos=GetFirstViewPosition();                       //获取第 1 个视图的位置
        CSdi066View* pView=(CSdi066View*)GetNextView(pos);         //获取视图指针
        pView->Invalidate();                                       //强制刷新视图窗口
    }
}
```

CFileDialog 类的构造函数的参数 1 若为 TRUE，用于打开文件。ReadString 函数用于从文本文件中读取一行文本，格式如下：

```
BOOL CStdioFile::ReadString(CString& rString)
```

参数如下。

- rString：存放当前行的文本字符串。

返回值：若读到文件末尾则返回 FALSE，否则返回 TRUE。

GetFirstViewPosition 函数获取文档关联的第 1 个视图的位置，存放到 pos 中，GetNextView 函数根据 pos 值获取视图指针，并将 pos 指向下一个视图。Invalidate 函数使视图窗口无效，强制视图重绘。

7.6.2　视图类显示数据

文档类提供 OnFileOpen 函数，用于读取文件，并将数据存放到整型数组中。视图类根据文档类提供的数据，以特定形式显示出来，且可编辑数据，根据需要调用文档类的 OnFileSave 函数，将数据保存到指定文件中。

（1）在类视图双击 CSdi066View 项，添加两个成员变量，代码如下：

```
public:
```

```
    CBrush m_brBlue;                                    //蓝色画刷
    CBrush m_brRed;                                     //红色画刷
```

（2）用鼠标右键单击 CSdi066View 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CSdi066View::OnInitialUpdate()
{
    CView::OnInitialUpdate();
    m_brRed.CreateSolidBrush(RGB(210,35,35));       //创建画刷实例
    m_brBlue.CreateSolidBrush(RGB(80,150,230));
}
```

（3）用鼠标右键单击 CSdi066View 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_LBUTTONDOWN 消息的处理函数，添加如下代码：

```
void CSdi066View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CView::OnLButtonDown(nFlags, point);
    CSdi066Doc* pDoc = GetDocument();               //获取文档类指针
    srand((unsigned)time(NULL));                    //设置当前时间为随机数种子
    for(int i=0;i<6;i++)                            //前 6 个元素随机赋值，范围为 1—33
        pDoc->m_num[i]=rand()%33+1;
    pDoc->m_num[6]=rand()%16+1;                     //第 7 个元素随机赋值，范围为 1—16
    Invalidate();                                   //强制重绘视图
}
```

GetDocument 函数获取文档类的指针，srand 函数设置随机数种子，利用当前时间作为种子，每次执行程序可生成不同的随机数序列。利用 for 循环，对数组的前 6 个元素随机赋值，rand 函数返回一个整型的随机值，用%取余后得到 0～32 范围内的值，再加 1 得到 1～33 范围内的值。

（4）双击 CSdi066View 类下的 OnDraw 项，添加如下代码：

```
void CSdi066View::OnDraw(CDC* pDC)
{
    CSdi066Doc* pDoc = GetDocument();                   //获取文档类指针
    ASSERT_VALID(pDoc);                                 //检验指针是否为空
    CRect rcClient;
    GetClientRect(rcClient);                            //获取视图客户区矩形
    CString strNum;
    CBrush* pOldBrush=pDC->SelectObject(&m_brRed); //将红色画刷选入 DC，并保存旧画刷
    CPen* pOldPen=(CPen*)pDC->SelectStockObject(NULL_PEN); //将空笔选入 DC，并保存旧画笔
    pDC->SetBkMode(TRANSPARENT);                        //设置文本背景透明
    pDC->TextOut(rcClient.CenterPoint().x-40,rcClient.top+40,"双色球摇号器");
                                                        //输出标题文本
    int nRadius=20;                                     //圆球的半径
    for(int i=0;i<7;i++)                                //绘制 7 个球
    {
        int num=pDoc->m_num[i];                         //获取第 i 个球的号码
        strNum.Format("%d",num);
        if(num<10)
            strNum="0"+strNum;                          //若号码小于 10，前面加个 0
        int midX=rcClient.CenterPoint().x+(i*3-9)*nRadius;
                                                        //计算第 i 个球的球心 x 坐标
        int midY=rcClient.top+100;                      //第 i 个球的球心 y 坐标
        if(i==6)                                        //若为第 7 个球，则使用蓝色画刷
            pDC->SelectObject(&m_brBlue);
        CRect rcEllipse(midX-nRadius,midY-nRadius,midX+nRadius,midY+nRadius);
                                                        //圆球的矩形边界
        pDC->Ellipse(rcEllipse);                        //绘制圆球
        pDC->TextOut(midX-8,midY-8,strNum);             //输出文本
```

```
        }
        pDC->SelectObject(pOldBrush);                    //恢复原有画刷
        pDC->SelectObject(pOldPen);                      //恢复原有字体
    }
```

ASSERT_VALID 宏用于检测指针是否为 NULL，仅在 Debug 版本中有效。SelectStockObject 函数选取系统自带的画笔，NULL_PEN 为空画笔。SetBkMode 函数设置输出文本的背景模式透明，TextOut 函数输出一行文本。绘制圆球时，使用空画笔，不绘制边界。

（5）生成程序并运行，如图 7-23 所示。初次显示所有号码均为“00”，在视图窗口上单击随机产生一组号码，单击“保存”菜单或按钮，可将当前号码保存到 ball 文件中，单击“打开”菜单或按钮，打开先前保存的 ball 文件，视图窗口显示保存的号码。

ball 文件为文本格式，可拖放到记事本程序中打开，一行保存一个号码，如图 7-24 所示。

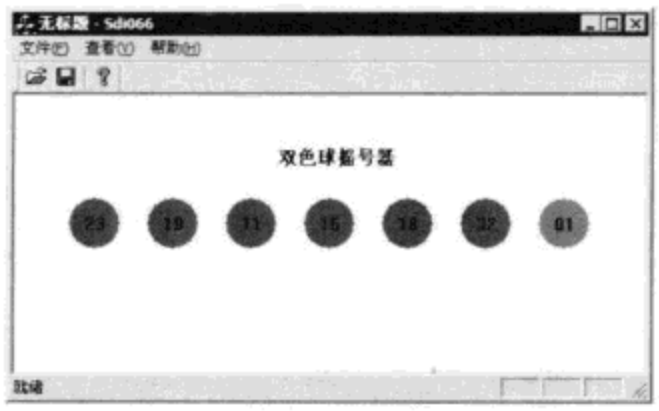


图 7-23　双色球摇号器

图 7-24　存储数据的 ball 文件

7.7 小结

本章主要介绍单文档应用程序相关内容。对于单文档应用程序的创建，在第 3 章中有详细的插图与解释。本章主要介绍单文档应用程序的内部结构，即应用程序类、文档类、视图类和主框架类。四者相互联系，组成一个完整的应用程序。然后介绍了对于菜单栏、工具栏、状态栏和对话栏的操作。最后讲解了文档类如何存取数据和视图类如何显示数据。

7.8 习题

1. 如何创建一个菜单？
2. 如何添加菜单的消息响应函数？
3. 如何创建一个工具栏？
4. 如何为工具栏按钮添加消息响应函数？
5. 编写一个程序，要求创建一个绘图菜单，并添加一个工具栏。

第 8 章　视图风格

MFC 提供多种视图风格，如编辑视图、列表视图、树视图、Html 视图等，这些视图通过内部封装一个对应的控件，用控件填充视图窗口，使用这些视图，可以轻松实现各种高级功能。也可手动创建一个 CView 派生类，在派生类的 OnCreate 函数里，动态创建一个控件，并用控件填充视图窗口，实现相同效果。

8.1 Edit 视图

Edit 视图相当于一个编辑框控件，提供文本编辑功能，可在 Edit 视图中输入、编辑文本，但只能显示单一字体样式，若要实现更加丰富的文本编辑功能，可使用 RichEdit 视图。

【实例 8-1】新建一个单文档工程 Sdi067，使用 Edit 视图，在视图窗口中显示 Sdi067View.cpp 文件的内容。

（1）新建单文档工程 Sdi067，在 MFC AppWizard 的最后一步，设置 CSdi067View 类的基类为 CEditView，如图 8-1 所示。

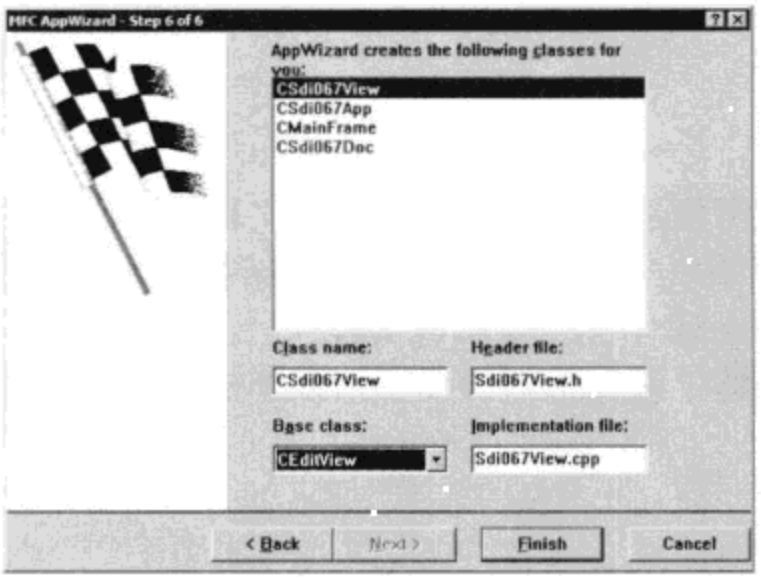


图 8-1　设置基类为 CEditView

（2）用鼠标右键单击 CSdi067View 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CSdi067View::OnInitialUpdate()
{
    CEditView::OnInitialUpdate();
    // TODO: Add your specialized code here and/or call the base class
    CStdioFile f;
    if(f.Open("Sdi067View.cpp",CFile::modeRead)) //以只读方式打开 CPP 文件
    {
        CString strLine,strText;
        while (f.ReadString(strLine))                    //读取一行文本，存入 strLine 中
            strText+=strLine+"\r\n";                     //累加到 strText，"\r\n"为换行符标记
        GetEditCtrl().SetWindowText(strText);            //设置 EditView 视图内容
    }
}
```

GetEditCtrl 函数获取 EditView 视图包含的 CEdit 对象的引用，格式如下：

```
CEdit& CEditView::GetEditCtrl() const
```

返回值：CEdit 对象的引用。

（3）生成程序并运行，其结果如图 8-2 所示。

图 8-2　EditView 视图

8.2 List 视图

List 视图有 4 种显示方式：列表 List、报表 Report、大图标 Icon、小图标 Small Icon，默认为大图标样式。GetListCtrl 函数获取列表视图包含的 CListCtrl 对象的引用，格式如下：

```
CListCtrl& CListView::GetListCtrl() const
```

返回值：CListCtrl 对象的引用。

【实例 8-2】新建一个单文档工程名为 Sdi068，使用 List 视图，并以报表样式显示数据。

（1）新建单文档工程 Sdi068，在 MFC AppWizard 的最后一步，设置 CSdi068View 类的基类为 CListView。

（2）在类视图用鼠标右键单击 CSdi068View 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个 CImageList 类型的变量 m_img。

（3）用鼠标右键单击 CSdi068View 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CSdi068View::OnInitialUpdate()
{
    CListView::OnInitialUpdate();
    // TODO: You may populate your ListView with items by directly accessing
    //  its list control through a call to GetListCtrl().
    m_img.Create(16,16,TRUE,2,1);                                    //创建图像列表，16*16
    m_img.Add(AfxGetApp()->LoadIcon(IDR_MAINFRAME));                 //加载图标资源
    m_img.Add(AfxGetApp()->LoadIcon(IDR_SDI068TYPE));
    CListCtrl& list=GetListCtrl();                                   //获取 CListCtrl 对象
    list.SetImageList(&m_img,LVSIL_SMALL);                           //设置控件使用的图像列表
    DWORD ctrlStyle=GetWindowLong(list.GetSafeHwnd(),GWL_STYLE);  //获取控件样式
    ctrlStyle|=LVS_REPORT;                                           //添加报表样式
    SetWindowLong(list.GetSafeHwnd(),GWL_STYLE,ctrlStyle);//设置控件新样式
    DWORD extStyle=list.GetExtendedStyle();                          //获取控件的扩展样式
    extStyle|=LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT;                 //添加两个扩展样式
    list.SetExtendedStyle(extStyle);                                 //设置新的扩展样式
    SetRedraw(TRUE);                                                 //重绘视图
    CRect rcClient;
```

- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

```
    GetClientRect(rcClient);
    list.InsertColumn(0,"营养元素",LVCFMT_LEFT,rcClient.Width()/2); //添加两列
    list.InsertColumn(1,"含量",LVCFMT_LEFT,rcClient.Width()/2);
    int nItem=list.InsertItem(0,"维生素 A",0);          //添加项
    list.SetItemText(nItem,1,"3000 单位");              //设置当前项其他列的值
    nItem=list.InsertItem(1,"维生素 C",1);
    list.SetItemText(nItem,1,"50 毫克");
}
```

Create 函数创建一个尺寸为 16*16、初始数目为 2 的图像列表，LoadIcon 函数加载图标资源，Add 函数将图标添加到图像列表中。GetListCtrl 函数获取 List 视图包含的控件对象，SetImageList 函数设置列表控件使用的图像列表，参数 2 表示使用小图标。

GetWindowLong 函数获取指定窗口的属性信息，格式如下：

```
LONG GetWindowLong(HWND hWnd,int nIndex)
```

参数如下。

- hWnd：指定窗口的句柄。
- nIndex：要查看的信息类别，如 GWL_STYLE 获取窗口的样式，GWL_HINSTANCE 获取程序的实例句柄，GWL_ID 获取窗口的 ID。

返回值：要查看信息的 32 位值。

先调用 GetWindowLong 函数获取控件的样式，存放到 ctrlStyle 中，使用|=位或运算符添加报表样式，再调用 SetWindowLong 重设控件的样式，SetWindowLong 函数格式如下：

```
LONG SetWindowLong(HWND hWnd,int nIndex,LONG dwNewLong)
```

参数如下。

- hWnd：窗口的句柄。
- nIndex：要设置的信息类别。
- dwNewLong：指定类别信息的新值。

返回值：指定类别信息的先前值。

GetExtendedStyle 函数获取控件的扩展样式，使用|=位或运算符添加网格线、整行选中两个扩展样式，SetExtendedStyle 函数重设控件的扩展样式。SetRedraw 函数设置视图窗口需要重绘，格式如下：

```
void CWnd::SetRedraw(BOOL bRedraw=TRUE)
```

参数如下。

- bRedraw：是否需要重绘，默认为 TRUE。

InsertColumn 函数在列表控件中插入列，InsertItem 函数添加一项，SetItemText 函数设置添加项的其他列的值，其中参数 1 为项的索引，参数 2 为列索引，参数 3 为文本值。

（4）生成程序并运行，其结果如图 8-3 所示。List 视图以报表形式显示数据，且每一项都有图标显示。

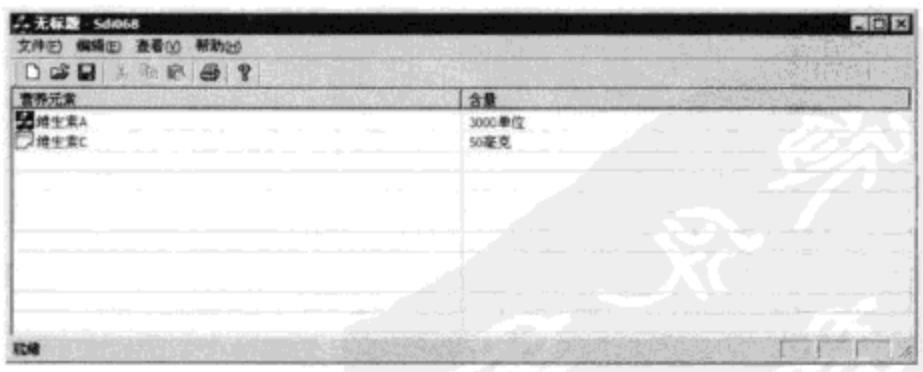


图 8-3　List 视图

8.3 Tree 视图

Tree 视图以层次关系显示所有元素，一个节点可以包含多个子节点，但一个子节点只能有一个父节点。GetTreeCtrl 函数获取 Tree 视图包含的 CTreeCtrl 对象的引用，格式如下：

```
CTreeCtrl& CTreeView::GetTreeCtrl() const
```

返回值：CTreeCtrl 对象的引用。

【实例 8-3】新建一个单文档工程名为 Sdi069，使用 Tree 视图，以树状形式显示元素。

（1）新建单文档工程 Sdi069，在 MFC AppWizard 的最后一步，设置 CSdi069View 类的基类为 CTreeView。

（2）在类视图用鼠标右键单击 CSdi069View 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个 CImageList 类型的变量 m_img。

（3）用鼠标右键单击 CSdi069View 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CSdi069View::OnInitialUpdate()
{
    CTreeView::OnInitialUpdate();
    // TODO: You may populate your TreeView with items by directly accessing
    //  its tree control through a call to GetTreeCtrl().
    m_img.Create(32,32,ILC_COLOR32|ILC_MASK,2,1);          //创建图像列表，32*32
    m_img.Add(AfxGetApp()->LoadStandardIcon(IDI_WINLOGO));//加载系统自带图标
    m_img.Add(AfxGetApp()->LoadStandardIcon(IDI_INFORMATION));
    CTreeCtrl& tree=GetTreeCtrl();                         //获取控件对象
    tree.SetImageList(&m_img,TVSIL_NORMAL);                //设置控件使用的图像列表
    tree.ModifyStyle(NULL,TVS_HASLINES|TVS_LINESATROOT|TVS_HASBUTTONS);
                                                           //修改样式
    SetRedraw();                                           //重绘 Tree 视图
    HTREEITEM h1=tree.InsertItem("桌面开发",0,0);          //添加两个根节点
    HTREEITEM h2=tree.InsertItem("Web 开发",0,0);
    tree.InsertItem("Visual C++",1,1,h1);                  //添加子节点
    tree.InsertItem("Visual C#",1,1,h1);
    tree.InsertItem("Java",1,1,h2);
    tree.InsertItem("Silverlight",1,1,h2);
}
```

Create 函数创建一个尺寸为 32*32，使用 32 位色和图像掩码，初始大小为 2 的图像列表。Add 函数添加两个系统标准图标，LoadStandardIcon 函数获取系统图标的句柄，LoadStandardIcon 函数格式如下：

```
HICON CWinApp::LoadStandardIcon(LPCTSTR lpszIconName) const
```

参数如下。

- lpszIconName：系统图标的 ID，如 IDI_APPLICATION 为应用程序图标。

返回值：图标的句柄。

GetTreeCtrl 函数获取树控件对象的引用，SetImageList 函数设置树控件使用的图像列表。ModifyStyle 函数用于修改窗口或控件的样式，格式如下：

```
BOOL CWnd::ModifyStyle(DWORD dwRemove,DWORD dwAdd,UINT nFlags = 0)
```

参数如下。

- dwRemove：要移除的样式，可使用|或运算符连接。
- dwAdd：要添加的样式。

❑ nFlags：传递给 SetWindowPos 函数的标志，默认为 0。

返回值：若成功则返回非零值，否则返回 0。

InsertItem 函数用于在树控件中插入项，参数 1 为项文本，参数 2 为图像索引，参数 3 为选中时的图像索引，参数 4 为父节点的句柄，默认为根节点。

（4）生成程序并运行，如图 8-4 所示。Tree 视图显示多个不同层次的节点，不同层次显示不同的图标。

图 8-4 Tree 视图

8.4 RichEdit 视图

RichEdit 视图提供强大的文本编辑功能，可实现类似 Word 软件的文字处理功能，不同的文本可使用不同的样式，且支持 OLE 对象嵌入。GetRichEditCtrl 函数获取 RichEdit 视图包含的 CRichEditCtrl 对象的引用，格式如下：

```
CRichEditCtrl& CRichEditView::GetRichEditCtrl() const
```

返回值：CRichEditCtrl 对象的引用。

【实例 8-4】新建一个单文档工程名为 Sdi0610，使用 RichEdit 视图，显示文件内容，且可设置不同文本的样式。

（1）新建单文档工程 Sdi0610，在 MFC AppWizard 的最后一步，设置 CSdi0610View 类的基类为 CRichEditView，单击“Finish”按钮，弹出提示窗口，单击“确定”按钮。

（2）在资源视图双击 Toolbar 节点下的 IDR_MAINFRAME 项，打开工具栏资源，添加一个图标，如图 8-5 所示。

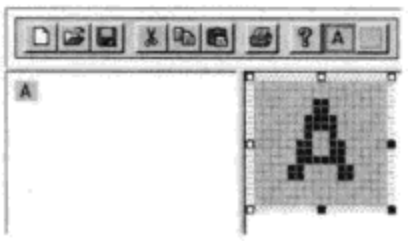


图 8-5 添加工具按钮

（3）双击按钮，设置 ID 为 ID_BUTTON_FONT，打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CSdi0610View 项，Object IDs 列表框选择 ID_BUTTON_ FONT 项，Messages 列表框选择 COMMAND 项，单击“Add Function”按钮添加函数，并添加如下代码：

```
void CSdi0610View::OnButtonFont()
{
    CRichEditCtrl& ctrl=GetRichEditCtrl();                    //获取 CRichEditCtrl 对象
    if(ctrl.GetSelectionType()==SEL_EMPTY)                    //若没有选中文本，直接返回
        return;
    CHARFORMAT currentFormat;                                 //字体样式结构体
    memset(&currentFormat,0,sizeof(CHARFORMAT));              //清空样式结构体
    ctrl.GetSelectionCharFormat(currentFormat);               //获取选中文本的字体样式
    LOGFONT logFont;                                          //字体结构体
    memset(&logFont,0,sizeof(CHARFORMAT));                    //清空字体结构体
    strcpy(logFont.lfFaceName,currentFormat.szFaceName);      //字体名称
    logFont.lfPitchAndFamily=currentFormat.bPitchAndFamily;   //字体间距和家族
    logFont.lfCharSet=currentFormat.bCharSet;                 //字符集
    logFont.lfHeight=currentFormat.yHeight/15;                //字体磅数
```

```
    logFont.lfStrikeOut=currentFormat.dwEffects & CFE_STRIKEOUT;  //是否删除线
    logFont.lfUnderline=currentFormat.dwEffects & CFE_UNDERLINE;  //是否下画线
    logFont.lfItalic=currentFormat.dwEffects & CFE_ITALIC;        //是否斜体
    if(currentFormat.dwEffects & CFE_BOLD)                        //是否粗体
        logFont.lfWeight=FW_BOLD;
    else
        logFont.lfWeight=FW_NORMAL;
    CFontDialog fontDialog(&logFont);                          //构造字体对话框对象
    fontDialog.m_cf.rgbColors=currentFormat.crTextColor;       //当前使用的颜色值
    if(fontDialog.DoModal()==IDOK)                             //以模态形式显示字体对话框
    {
        fontDialog.GetCharFormat(currentFormat);               //获取设置的字体样式
        ctrl.SetSelectionCharFormat(currentFormat);            //用字体样式设置选中文本
    }
}
```

GetRichEditCtrl 函数获取 CRichEditCtrl 对象的引用。GetSelectionType 函数获取控件中文本的选取类型，格式如下：

```
WORD CRichEditCtrl::GetSelectionType() const
```

返回值：选取类型，如 SEL_EMPTY 表示没有选取，SEL_TEXT 表示选中文本，SEL_OBJECT 表示选中至少一个 OLE 对象。

若没有选中文本，则直接返回。CHARFORMAT 结构包含 Rich Edit 控件文本的字体信息，格式如下：

```
typedef struct _charformat {
    UINT cbSize;                        //结构体的字节大小
    DWORD dwMask;                       //哪些成员可用
    DWORD dwEffects;                    //字体效果组合，如加粗、斜体等
    LONG yHeight;                       //字体高度
    LONG yOffset;                       //垂直高度偏移
    COLORREF crTextColor;               //文本颜色
    BYTE bCharSet;                      //字符集
    BYTE bPitchAndFamily;               //字体间距和家族
    TCHAR szFaceName[LF_FACESIZE];      //字体名称
} CHARFORMAT;
```

memset 函数用于将某段内存初始化为一个值，格式如下：

```
void *memset(void *dest,int c,size_t count)
```

参数如下。

- dest：指向起始地址的指针。
- c：初始值。
- count：要初始化内存的字节长度。

返回值：起始指针。

调用 memset 函数将 CHARFORMAT 结构体成员初始为 0。GetSelectionCharFormat 函数获取选中文本的字体样式，格式如下：

```
DWORD CRichEditCtrl::GetSelectionCharFormat(CHARFORMAT& cf) const
```

参数如下。

- cf：CHARFORMAT 结构体对象的引用，存放字体样式。

返回值：结构体的 dwMask 值。

LOGFONT 结构体包含逻辑字体的所有信息，可以直接用来创建一个字体对象，memset 函数初始结构体成员为 0。strcpy 函数用于将一个字符串的值复制到另一个字符串中，格式如下：

```
char *strcpy(char *strDestination,const char *strSource)
```

参数如下。

- strDestination：目的字符串的起始地址。
- strSource：被复制的字符串的起始地址。

返回值：目的字符串的起始地址。

CHARFORMAT 结构体的 dwEffects 成员存放字体的效果组合，使用&位与运算符获取是否包含某种效果，如 currentFormat.dwEffects & CFE_STRIKEOUT 用于判断是否包含删除线，若不包含则所有位都为 0，相当于 FALSE，若包含则至少有一位为 1，相当于 TRUE。

CFontDialog 类封装了系统自带的字体对话框，其构造函数格式如下：

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

参数如下。

- lplfInitial：指向 LOGFONT 结构体的指针，包含初始信息。
- dwFlags：字体标志组合。
- pdcPrinter：指向打印机 DC 的指针。
- pParentWnd：父窗口的指针。

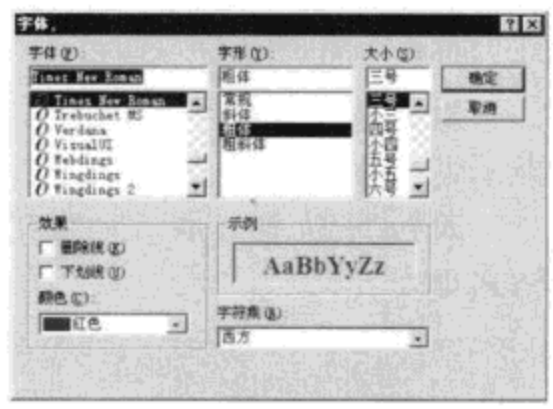


图 8-6 字体对话框

m_cf 成员是一个 CHOOSEFONT 结构体，用于定制 CFontDialog 类对象，其结构体成员 rgbColors 用于设置字体对话框的颜色值。DoModal 函数以模态形式显示字体对话框，若单击“确定”按钮则返回 IDOK，表示成功设置字体样式，如图 8-6 所示。

GetCharFormat 函数获取字体对话框设置的字体样式，格式如下：

```
void CFontDialog::GetCharFormat(CHARFORMAT& cf) const
```

参数如下。

- cf：CHARFORMAT 结构体对象的引用，存放设置的字体样式。

SetSelectionCharFormat 函数设置 Rich Edit 控件选中文本的字体样式，格式如下：

```
BOOL CRichEditCtrl::SetSelectionCharFormat(CHARFORMAT& cf)
```

参数如下。

- cf：使用的字体样式。

返回值：若成功返回非零值，否则返回 0。

（4）在类视图双击 CSdi0610View 类下的 OnInitialUpdate 项，添加如下代码：

```
void CSdi0610View::OnInitialUpdate()
{
    CRichEditView::OnInitialUpdate();
    // Set the printing margins (720 twips = 1/2 inch).
    SetMargins(CRect(720, 720, 720, 720));
    CStdioFile f;
    if(f.Open("Sdi0610View.cpp",CFile::modeRead))      //以只读方式打开文件
    {
        CString strLine,strText;
        while(f.ReadString(strLine))                   //读取所有行文本
            strText+=strLine+"\r\n";
        GetRichEditCtrl().SetWindowText(strText);      //在 RichEdit 视图显示文件内容
        GetRichEditCtrl().SetSel(100,300);             //选中部分文本
    }
```

```
}
```

SetSel 函数设置 Rich Edit 控件选中的文本，格式如下：

```
void CRichEditCtrl::SetSel(long nStartChar,long nEndChar)
```

参数如下。

- nStartChar：选中文本第 1 个字符的索引。
- nEndChar：选中文本最后 1 个字符的索引。

（5）生成程序并运行，如图 8-7 所示。在 RichEdit 视图中显示 CPP 文件的内容，并初始选中部分文本，单击“A”工具按钮，弹出字体对话框，设置字体、字号、颜色等样式。

图 8-7　RichEdit 视图

（6）选择“编辑”|“插入新对象”命令，弹出“插入对象”窗口，如图 8-8 所示。在“对象类型”列表框中选择“Microsoft PowerPoint 幻灯片”项，单击“确定”按钮，插入 OLE 对象，如图 8-9 所示。

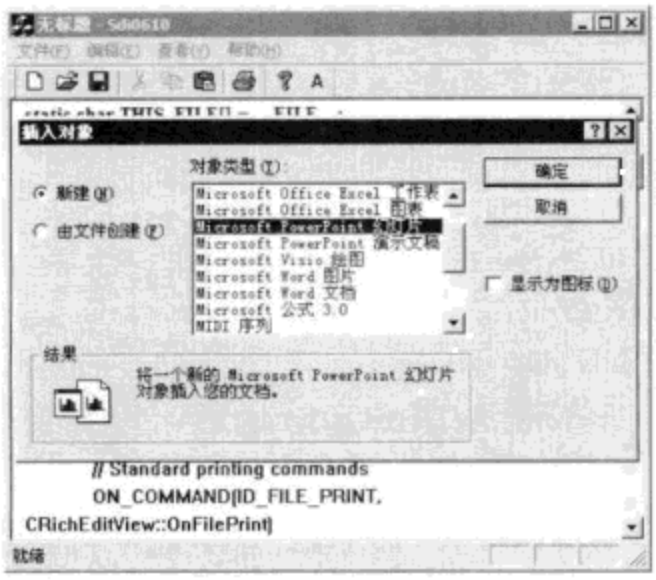


图 8-8　插入对象

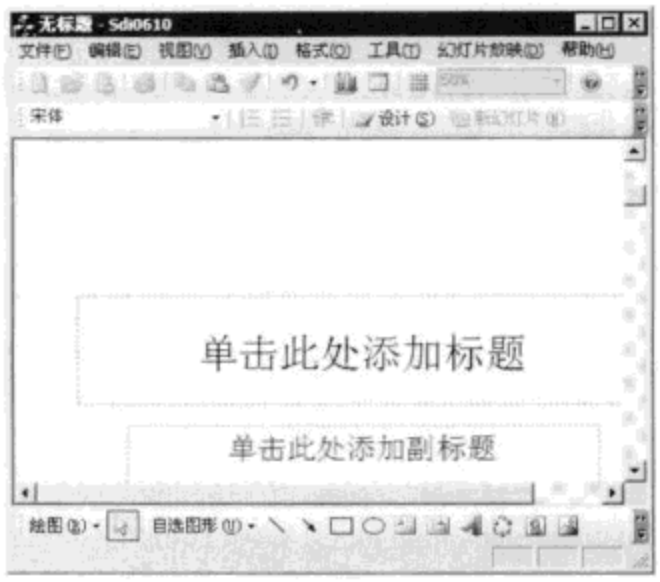


图 8-9　插入的幻灯片对象

RichEdit 视图支持插入 OLE 对象，如在视图中插入一个幻灯片对象后，自动在后台启动 PowerPoint 程序，无须手动打开 PowerPoint 程序，直接在当前程序中调用 PowerPoint 程序提供的嵌入编辑功能。

8.5 小结

MFC 提供了多种风格的视图，包括编辑视图、列表视图、树视图、Html 视图等。这些视图通过内部封装一个对应的控件，用控件填充视图窗口，使用这些视图，可以轻松地实现各种高级功能。本章分别介绍了常用的视图，并通过实例展示各种视图之间的不同。

8.6 习题

1. 视图风格分为哪几种？
2. Edit 视图有什么特性？
3. List 视图有几种显示方式？分别是什么？
4. RichEdit 视图和 Edit 视图有什么区别？
5. 理解四种视图方式的运用。

第9章　切分窗口

默认的单文档框架只有一个视图窗口，但有时候需要多个视图窗口协同操作，如Windows 的资源管理器，左边显示驱动器和目录列表，右边显示左边选中目录包含的所有文件。MFC 提供 CSplitterWnd 类用于切分框架窗口，实现多视图显示。

9.1 了解窗口切分

框架窗口可以被水平、垂直切分，切分后的每个窗格都有自己的视图类，视图类可相同也可不同，不同的窗格间使用切分条分隔，使用鼠标拖动切分条可改变窗格的大小。MFC 提供 CSplitterWnd 类用于实现框架窗口的切分，可分为动态、静态切分两种类型。

动态切分最多有两行两列，通常用于显示同一个视图的不同区域，如一个文档篇幅较长，可在多个窗格中显示该文档，不同窗格显示不同的段落，以便切换阅读。动态切分窗口，步骤如下所示：

（1）在 CMainFrame 类中添加一个 CSplitterWnd 类变量。

（2）在 CMainFrame 类中重写 OnCreateClient 虚函数。

（3）调用 CSplitterWnd::Create 函数，动态切分窗口。

（4）调用 CSplitterWnd::SetColumnInfo 和 CSplitterWnd::SetRowInfo 函数，设置窗格宽度和高度值。

OnCreateClient 函数在 OnCreate 函数之前被调用，用于切分框架窗口。Create 函数用于动态切分窗口，格式如下：

```
BOOL CSplitterWnd::Create(CWnd* pParentWnd,int nMaxRows,int nMaxCols,SIZE sizeMin,CCreateContext*pContext,DWORDdwStyle=WS_CHILD|WS_VISIBLE|WS_HSCROLL|WS_VSCROLL| SPLS_DYNAMIC_SPLIT,UINT nID = AFX_IDW_PANE_FIRST)
```

参数如下。

- pParentWnd：父窗口的指针。
- nMaxRows：最多行数，不能超过 2。
- nMaxCols：最多列数，不能超过 2。
- sizeMin：显示窗格的最小尺寸。
- pContext：指向 CCreateContext 对象的指针，保存有框架、视图、文档的运行时信息。
- dwStyle：切分窗口的样式。
- nID：子窗格的 ID。

返回值：若成功则返回非零值，否则返回 0。

静态切分最多有 16 行 16 列，通常用于显示不同的视图，如资源管理器中左边为树状视图，右边为列表视图，不同于动态切分，静态切分必须指定每个窗格所对应的视图，步骤如下所示：

（1）在 CMainFrame 类中添加一个 CSplitterWnd 类变量。

（2）在 CMainFrame 类中重写 OnCreateClient 虚函数。

（3）调用 CSplitterWnd:: CreateStatic 函数，静态切分窗口。

（4）调用 CSplitterWnd::CreateView 函数，指定每个窗格的视图。

（5）调用 CSplitterWnd::SetColumnInfo 和 CSplitterWnd::SetRowInfo 函数，设置窗格宽度和高度值。

（6）调用 CSplitterWnd::RecalcLayout 函数，重设窗口大小。

CreateStatic 函数用于静态切分窗口，格式如下：

```
BOOL CSplitterWnd::CreateStatic(CWnd* pParentWnd,int nRows,int nCols,DWORD dwStyle
= WS_CHILD | WS_VISIBLE,UINT nID = AFX_IDW_PANE_FIRST)
```

参数如下。

- pParentWnd：父窗口的指针。
- nRows：行数。
- nCols：列数。
- dwStyle：切分窗口的样式。
- nID：子窗格的 ID。

返回值：若成功则返回非零值，否则返回 0。

CreateView 函数用于设置某个窗格使用的视图，格式如下：

```
BOOL CSplitterWnd::CreateView(int row, int col, CRuntimeClass* pViewClass, SIZE
sizeInit, CCreateContext* pContext)
```

参数如下。

- row：指定窗格的行索引。
- col：列索引。
- pViewClass：视图的 CRuntimeClass 类指针。
- sizeInit：视图的初始尺寸。
- pContext：指向 CCreateContext 对象的指针。

返回值：若成功则返回非零值，否则返回 0。

SetColumnInfo 函数用于设置切分窗口的指定列的宽度值，格式如下：

```
void CSplitterWnd::SetColumnInfo(int col,int cxIdeal,int cxMin)
```

参数如下。

- col：列索引。
- cxIdeal：期望宽度值。
- cxMin：最小宽度值。

SetRowInfo 函数用于设置切分窗口的指定行的高度值，格式如下：

```
void CSplitterWnd::SetRowInfo(int row,int cyIdeal,int cyMin)
```

参数如下。

- row：行索引。
- cyIdeal：期望高度值。
- cyMin：最小高度值。

RecalcLayout 函数用于设置行列值后，重新显示切分窗口，格式如下：

```
void CSplitterWnd::RecalcLayout()
```

9.2 静态切分窗口

使用静态切分窗口功能，可在一个框架窗口中显示多种视图，实现各种丰富的功能，如 Word 软件中可在左边显示文档目录，右边显示正文，单击左边的一个章节，右边自动定位至该章节。

【实例 9-1】新建一个单文档工程名为 Sdi0611，将框架窗口切分为 1 行 2 列，左边为树视图，右边为 Edit 视图，单击树视图的一项后，Edit 视图显示选中项的文件内容。

（1）新建单文档工程 Sdi0611，在 MFC AppWizard 的最后一步，设置 CSdi0611View 类的基类为 CEditView。

（2）在类视图双击 CMainFrame 项，在类定义中添加如下成员变量：

```
public:
    CImageList m_img;                    //图像列表
    CTreeCtrl m_tree;                    //树控件
    CSplitterWnd m_split;                //切分条
```

（3）右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnCreateClient 函数，添加如下代码：

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
    m_split.CreateStatic(this,1,2);                          //静态切分窗口为 1 行 2 列
    m_tree.Create(WS_CHILD|WS_VISIBLE|TVS_HASLINES|TVS_HASBUTTONS|TVS_LINESATROOT,
        CRect(0,0,10,10),&m_split,m_split.IdFromRowCol(0,0)); //动态创建树控件，在窗格中显示
    m_tree.SetOwner(this);                                   //设置树控件的拥有者为框架窗口
    m_split.CreateView(0,1,RUNTIME_CLASS(CSdi0611View),CSize(0,0),pContext);
                                                             //设置窗格使用的视图
    CRect rc;
    GetClientRect(rc);                                       //获取客户区矩形
    m_split.SetColumnInfo(0,rc.Width()/4,0);                 //设置第 1 列的宽度
    m_split.SetColumnInfo(1,rc.Width()/4*3,0);               //设置第 2 列的宽度
    m_split.RecalcLayout();                                  //重新显示切分窗口

    return TRUE;                                             //直接返回，不调用基类版本
}
```

CreateStatic 函数用于将框架窗口切分为 1 行 2 列。Create 函数动态创建一个树控件实例，参数 1 为控件的样式，参数 2 为初始尺寸，参数 3 为父窗口的指针，参数 4 为控件的 ID，将 m_split 作为父窗口，控件 ID 设为 0 行 0 列子窗格的 ID，可创建非视图类的窗格，该窗格显示树控件。

IdFromRowCol 函数指定行列所在窗格的子窗口 ID，格式如下：

```
int CSplitterWnd::IdFromRowCol(int row, int col)
```

参数如下。

- row：窗格的行索引。
- col：窗格的列索引。

返回值：指定窗格的子窗口 ID。

SetOwner 函数设置窗口的拥有者窗口，拥有者可接收该窗口发出的消息，格式如下：

```
void CWnd::SetOwner(CWnd* pOwnerWnd)
```

参数如下。

- pOwnerWnd：窗口拥有者的指针。

CreateView 函数设置 0 行 1 列窗格使用 CSdi0611View 视图，RUNTIME_CLASS 宏获取类的运行时信息。SetColumnInfo 函数设置第 0 列宽度为窗口宽度的 1/4，第 1 列为窗口宽度的 3/4，RecalcLayout 函数用于设置行列值后，重新显示切分窗口。

（4）在当前 CPP 文件开头处，添加一句#include "Sdi0611View.h"，包含 CSdi0611View 类的头文件。在类视图双击 CSdi0611View 项，在类定义前添加一句#include "Sdi0611Doc.h"，包含 CSdi0611Doc 类的头文件。

（5）在类视图双击 CMainFrame 类下的 OnCreate 项，添加如下代码：

```
m_img.Create(32,32,ILC_COLOR32|ILC_MASK,2,1);                    //创建图像列表
m_img.Add(AfxGetApp()->LoadStandardIcon(IDI_APPLICATION));       //加载系统图标
m_img.Add(AfxGetApp()->LoadIcon(IDR_SDI061TYPE));
m_tree.SetImageList(&m_img,TVSIL_NORMAL);                        //设置树控件使用的图像列表
HTREEITEM h1=m_tree.InsertItem("Sdi0611",0,0);                   //插入根节点
m_tree.InsertItem("MainFrm.cpp",1,1,h1);                         //插入子节点
m_tree.InsertItem("MainFrm.h",1,1,h1);
m_tree.InsertItem("ReadMe.txt",1,1,h1);
m_tree.InsertItem("Resource.h",1,1,h1);
m_tree.InsertItem("Sdi0611.dsp",1,1,h1);
m_tree.InsertItem("Sdi0611.dsw",1,1,h1);
m_tree.Expand(h1,TVE_EXPAND);                                    //展开根节点
```

Create 函数创建图像列表实例，LoadStandardIcon 函数获取系统图标的句柄，LoadIcon 函数获取图标资源的句柄，Add 函数将图标添加到图像列表中。SetImageList 函数设置树控件使用的图标列表。

InsertItem 函数在树控件插入节点项，若不指定父节点，作为根节点插入。Expand 函数展开根节点下的所有子项。OnCreate 函数在 OnCreateClient 函数之后调用，在 OnCreateClient 函数中已创建树控件实例，可在 OnCreate 函数中为树控件添加项。

（6）右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_SIZE 消息的处理函数，添加如下代码：

```
void CMainFrame::OnSize(UINT nType, int cx, int cy)
{
    CFrameWnd::OnSize(nType, cx, cy);
    // TODO: Add your message handler code here
    if(IsWindowVisible())                                        //若框架窗口可见
    {
        m_split.SetColumnInfo(0,cx/4,0);                         //重设列宽
        m_split.SetColumnInfo(1,cx/4*3,0);
        m_split.RecalcLayout();
    }
}
```

当窗口大小改变后，触发 WM_SIZE 消息，自动调用该函数，参数格式如下。

- nType：窗口的变化类型，如 SIZE_MAXIMIZED 表示窗口被最大化。
- cx：客户区的新宽度。
- cy：客户区的新高度。

IsWindowVisible 函数判断窗口是否可见，若可见，则调用 SetColumnInfo 函数重设切分窗口的列宽，再调用 RecalcLayout 函数重新显示切分窗口。

（7）在资源视图双击 Toolbar 节点下的 IDR_MAINFRAME 项，打开工具栏资源，添加一个图标，如图 9-1 所示。

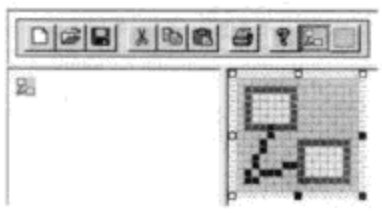


图 9-1　添加工具按钮

（8）双击添加的工具按钮，设置 ID 为 ID_BUTTON_SHOW，打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CMainFrame 项，Object IDs 列表框选择 ID_BUTTON_SHOW 项，Messages 列表框选择 COMMAND 项，单击 Add Function 按钮添加函数，并添加如下代码：

```
void CMainFrame::OnButtonShow()
{
    HTREEITEM hSel=m_tree.GetSelectedItem();                     //获取选中节点的句柄
    if(m_tree.ItemHasChildren(hSel))                             //若有子节点，则直接返回
        return;
    CString strSel=m_tree.GetItemText(hSel);                     //获取选中节点的文本值
    CStdioFile f;
```

```
    if(f.Open(strSel,CFile::modeRead))                          //以只读方式打开文件
    {
        CString strLine,strText;
        while(f.ReadString(strLine))                            //读取所有行文本
            strText+=strLine+"\r\n";
        CView* pView=(CView*)m_split.GetPane(0,1);              //获取 0 行 1 列窗格的指针
        if(pView->GetRuntimeClass()==RUNTIME_CLASS(CSdi0611View)) //若为指定视图
        {
            CSdi0611View* pEditView=(CSdi0611View*)pView;           //转换指针类型
            pEditView->GetEditCtrl().SetWindowText(strText); //设置 Edit 视图显示文本
        }
    }
}
```

GetSelectedItem 函数获取选中节点的句柄，ItemHasChildren 函数判断是否有子节点，若为根节点，直接返回。GetItemText 函数获取选中节点的文本值，Open 函数以只读方式打开指定文件，ReadString 函数读取一行文本，存放到 strLine 中。

GetPane 函数获取指定窗格的指针，格式如下：

```
CWnd* CSplitterWnd::GetPane(int row, int col)
```

参数如下。

- row：窗格的行索引。
- col：窗格的列索引。

返回值：窗格的 CWnd 指针，通常是一个 CView 派生类。

GetRuntimeClass 函数获取类的运行时信息，格式如下：

```
CRuntimeClass* CObject::GetRuntimeClass() const
```

返回值：类的 CRuntimeClass 结构的指针。

若 0 行 1 列窗格的所使用的视图类为 CSdi0611View 类，将 CView 类指针转换为 CSdi0611View 类指针，GetEditCtrl 函数获取 Edit 视图包含的 CEdit 对象，SetWindowText 函数将文件内容显示到 Edit 视图中。

（9）生成程序并运行，如图 9-2 所示。框架窗口被切分为 1 行 2 列，左边为树视图，右边为 Edit 视图，选择树视图一个子节点，单击添加的工具按钮，在 Edit 视图显示对应文件的内容。

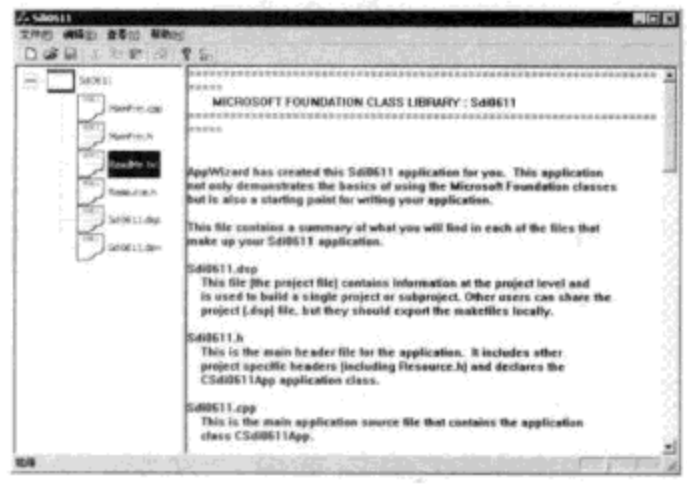


图 9-2　静态切分窗口

9.3 多视图切换

切分视图可将框架分为多个显示区域，但有时候需要以视图切换的方式显示，如 Word 中视图有“普通”、“Web 版式”、“页面”三种视图方式，选中一项后，视图窗口切换为对应样式。MFC 提供根据已有文档动态创建一个视图的方法，可随时切换至指定类型的视图，相对于切分视图，节省了宝贵的窗口资源。

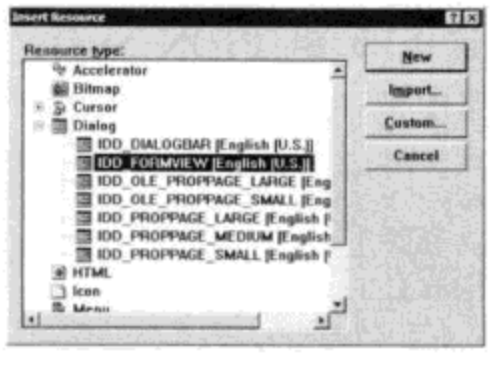


图 9-3　添加对话框资源

【实例 9-2】新建一个单文档工程名为 Sdi0612，在同一个框架窗口中，切换显示两种不同视图。

（1）新建单文档工程 Sdi0612，在资源视图单击右键，在弹出的快捷菜单中选择 Insert 命令，弹出 Insert Resource 窗口，选择 Dialog 节点下的 IDD_FORMVIEW 项，单击 New 按钮，添加两个对话框模板资源，如图 9-3 所示。

（2）设置两个对话框资源的 ID 分别为 IDD_FORMVIEW1、IDD_FORMVIEW2。

（3）在资源视图分别右键单击 IDD_FORMVIEW1、IDD_FORMVIEW2 项，在弹出的快捷菜单中选择 Properties 命令，设置 Language 为 Chinese (P.R.C.)。

（4）双击 IDD_FORMVIEW1 项，打开对话框模板，拖放编辑框、按钮控件到模板上，如图 9-4 所示。设置按钮的 Caption 为“打开文件”。

（5）双击 IDD_FORMVIEW2 项，打开对话框模板，拖放编辑框控件到模板上，如图 9-5 所示。右键单击编辑框，在弹出的快捷菜单中选择 Properties 命令，切换到 Styles 选项卡，勾选 Multiline、Vertical Scroll、Auto VScroll 复选框，取消勾选 Auto HScroll 勾选框。

（6）双击 IDD_FORMVIEW1 项，按 Ctrl+W 组合键，自动弹出 Adding a class 窗口，单击 OK 按钮，打开 New Class 窗口，如图 9-6 所示。

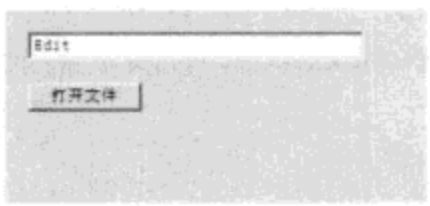


图 9-4　IDD_FORMVIEW1

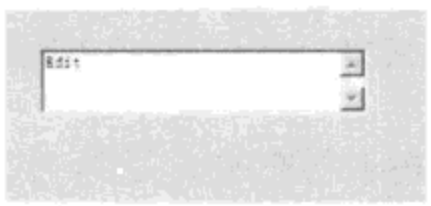


图 9-5　IDD_FORMVIEW2

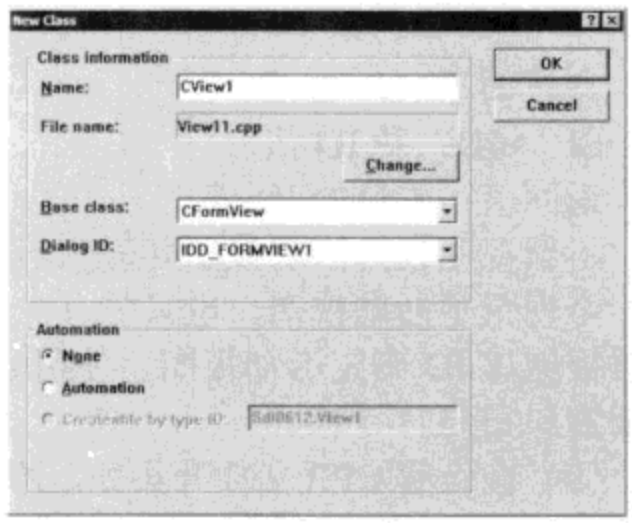


图 9-6　添加类

（7）Name 编辑框输入 CView1，Base class 组合框选择 CFormView 项，单击 OK 按钮添加类。用同样方式，为 IDD_FORMVIEW2 添加类 CView2。

Tips　CFormView 类派生自 CScrollView 类，可看做是对话框和视图的结合体，在资源视图中可拖放控件设计界面，运行时以 Form 视图形式显示。

（8）在类视图右键单击 CMainFrame 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加一个返回类型为 void 的函数 ChangeView(int nView)，添加如下代码：

```
void CMainFrame::ChangeView(int nView)
{
    CView *pNewView=NULL;                                    //新视图指针
    switch(nView)                                            //视图模板 ID
    {
    case IDD_FORMVIEW1:                                      //若为视图 1
        pNewView=(CView*)new CView1; break;                  //动态创建一个视图 1 对象
    case IDD_FORMVIEW2:
        pNewView=(CView*)new CView2; break;
    }
    CCreateContext context;
    CView *pOldView=GetActiveView();                         //获取当前活动视图
    context.m_pCurrentDoc=pOldView->GetDocument();           //获取当前文档
    pNewView->Create(NULL,NULL,WS_CHILD,CFrameWnd::rectDefault,
        this, nView, &context);                              //创建视图实例
    pNewView->OnInitialUpdate();                             //新视图初始化更新
    SetActiveView(pNewView);                                 //新视图设为活动视图
    pNewView->ShowWindow(SW_SHOW);                           //显示新视图
```

```
    pOldView->ShowWindow(SW_HIDE);                          //隐藏旧视图
    pNewView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);             //设置视图窗口 ID
    delete pOldView;                                        //删除旧视图
    RecalcLayout();                                         //重新显示框架窗口
}
```

参数 nView 为视图模板的 ID，若为视图 1，使用 new 动态创建视图 1 对象，存放到 pNewView 中。CCreateContext 结构存放框架、视图、文档信息，GetActiveView 函数获取当前活动视图的指针，调用 GetDocument 函数获取视图关联的文档类指针。Create 函数根据 CCreateContext 结构，创建一个视图窗口实例，父窗口为主框架窗口，OnInitialUpdate 函数用于初始化视图。

SetActiveView 函数设置新视图为当前活动视图，ShowWindow 函数设置窗口显示状态。SetDlgCtrlID 函数设置窗口或控件的新 ID，格式如下：

```
int CWnd::SetDlgCtrlID(int nID)
```

参数如下。

- nID：新 ID 值。

返回值：先前的 ID 值。

将新视图设为活动视图并显示后，还需要将新视图的 ID 设为 AFX_IDW_ PANE_FIRST，只有一个视图是活动视图，也只能有一个视图的 ID 是 AFX_IDW_PANE_ FIRST，RecalcLayout 函数根据该 ID 确定活动视图。使用 delete 释放动态创建的旧视图，RecalcLayout 函数重新显示框架窗口布局。

（9）在当前 CPP 文件开头处，添加视图类所在的头文件，代码如下：

```
#include "View1.h"
#include "View2.h"
```

（10）在资源视图双击 Toolbar 节点下的 IDR_MAINFRAME 项，打开工具栏资源，添加两个工具图标，如图 9-7 所示。

图 9-7 添加工具按钮

（11）双击添加的按钮，分别设置 ID 为 ID_BUTTON_ VIEW1、ID_BUTTON_VIEW2。打开类视图，为两个按钮添加 CMainFrame 类的 COMMAND 消息函数，添加如下代码：

```
void CMainFrame::OnButtonView1()
{
    ChangeView(IDD_FORMVIEW1);                              //切换至视图 1
}
void CMainFrame::OnButtonView2()
{
    ChangeView(IDD_FORMVIEW2);                              //切换至视图 2
}
```

（12）CMainFrame 类添加一个 CString 类型的成员变量 m_strInput。

（13）在资源视图双击 IDD_FORMVIEW1 项，双击“打开文件”按钮，添加如下代码：

```
void CView1::OnButton1()
{
    CString strInput;
    GetDlgItem(IDC_EDIT1)->GetWindowText(strInput);         //编辑框输入值
    if(strInput.IsEmpty())                                  //若为空，直接返回
        return;
    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();        //获取主窗口指针
    pFrame->m_strInput=strInput;                            //设置主窗口变量
    pFrame->ChangeView(IDD_FORMVIEW2);                      //切换至视图 2
}
```

（14）在当前 CPP 文件开头处，添加一句#include "MainFrm.h" ，包含 CMainFrame 类的头文件。

（15）在类视图右键单击 CView2 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_SIZE 消息的处理函数，添加如下代码：

```
void CView2::OnSize(UINT nType, int cx, int cy)
{
    CFormView::OnSize(nType, cx, cy);
    // TODO: Add your message handler code here
    CEdit* pEdit=(CEdit*)GetDlgItem(IDC_EDIT1);          //获取编辑框控件指针
    if(pEdit->GetSafeHwnd())                             //若控件句柄存在
        pEdit->MoveWindow(0,0,cx,cy);                    //填充整个视图
}
```

GetSafeHwnd 函数获取窗口或控件的句柄值，若窗口实例不存在，则返回 NULL。若控件实例存在，则调用 MoveWindow 函数设置编辑框大小，充满整个视图。

（16）右键单击 CView2 项，在弹出的快捷菜单中选择 Add Virtual Functions 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CView2::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    // TODO: Add your specialized code here and/or call the base class
    CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();     //获取主窗口指针
    if(pFrame->m_strInput.IsEmpty())                     //若文件名为空
        return;
    CStdioFile f;
    if(f.Open(pFrame->m_strInput,CFile::modeRead))       //以只读方式打开文件
    {
        CString strLine,strText;
        while(f.ReadString(strLine))
            strText+=strLine+"\r\n";
        GetDlgItem(IDC_EDIT1)->SetWindowText(strText);   //在编辑框中显示文件内容
    }
}
```

CMainFrame 类的 m_strInput 变量用于记录视图 1 中输入的文件名，视图 2 通过该变量获取视图 1 中输入的值，实现数据交换。CMainFrame 类在各个类中访问比较方便，可作为数据的中转站。

（17）在当前 CPP 文件开头处，添加一句#include "MainFrm.h" ，包含 CMainFrame 类的头文件。

（18）生成程序并运行，单击 1 按钮，切换至视图 1，如图 9-8 所示。在编辑框中输入文件名，如 readme.txt，单击“打开文件”按钮，自动切换到视图 2，显示文件内容，也可单击 2 按钮，切换至视图 2，如图 9-9 所示。

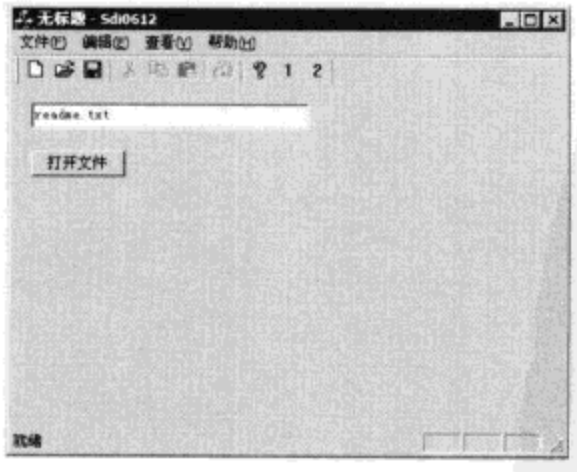


图 9-8　视图 1

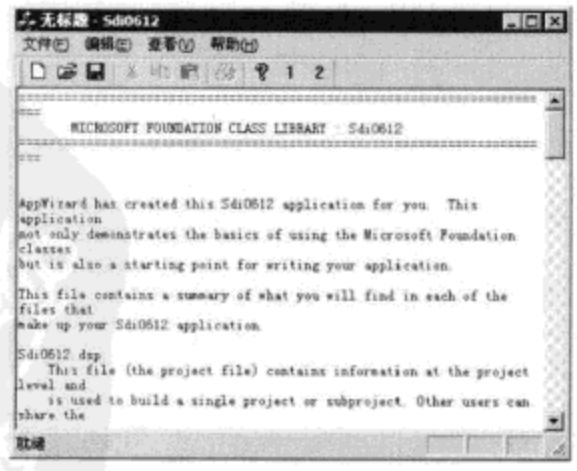


图 9-9　视图 2

9.4 小结

在学习了视图的特性之后，我们学习了对窗口进一步的操作——切分窗口。单文档应用程序默认提供了一个视图窗口，但是有时无法满足需求，需要切分窗口。切分窗口分为静态切分和动态切分。最后介绍了多视图切换，根据要求，可随时切换至指定类型的视图。

9.5 习题

1. 动态切分窗口的具体步骤是什么？
2. 静态切分窗口的具体步骤是什么？
3. 编写一个程序，使用动态切分窗口的方法，将窗口分为 2×2 的布局。
4. 编写一个程序，使用静态切分窗口的方法，将窗口分为 2×2 的布局。

第 10 章　多文档应用程序

多文档程序可在多个窗口打开多个文档，Visual C++开发环境就是一个典型的多文档程序，如图 10-1 所示。在实际开发中，根据需要决定采用单文档还是多文档模式，若需要同时显示不同类型的文档，应采用多文档模式，一般情况下，单文档程序已经可以满足功能需求。

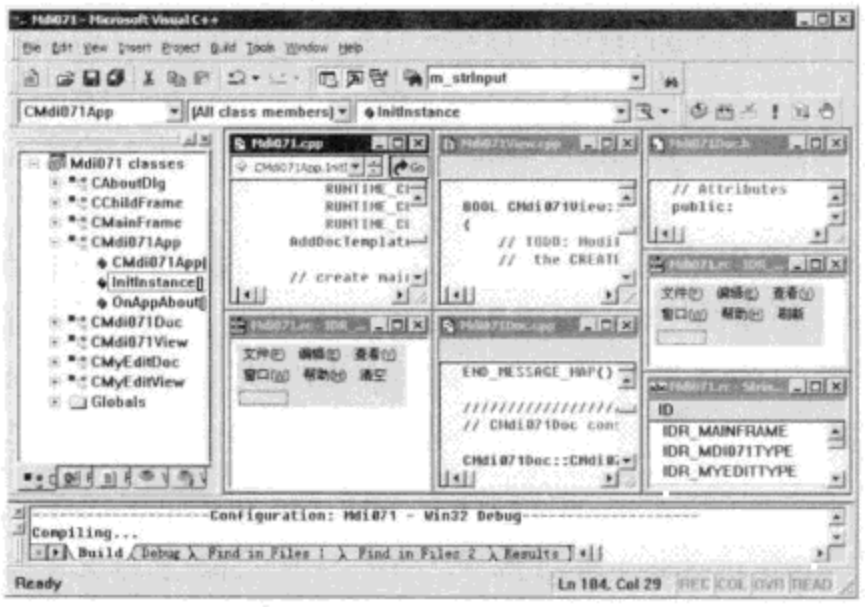


图 10-1　Visual C++开发环境

10.1 了解生成类

使用 MFC AppWizard 生成的多文档程序，自动生成 6 个类，其中 CMainFrame、CChildFrame、App、Doc、View 五个类是多文档程序的基础支架，CMainFrame 类负责程序主窗口的框架显示，包括子框架窗口、工具栏、状态栏等元素，CChildFrame 类负责多文档窗口的子框架窗口，其父窗口为主框架窗口，CChildFrame 类和 Doc、View 类共同构成一个文档模板组合。

【实例 10-1】新建一个多文档工程名为 Mdi071，了解 AppWizard 自动生成的几个类。

（1）新建多文档工程 Mdi071，在类视图双击 CMdi071App 类下的 InitInstance 项，部分代码如下：

```
CMultiDocTemplate* pDocTemplate;                    //多文档模板
pDocTemplate = new CMultiDocTemplate(               //构造文档模板对象
    IDR_MDI071TYPE,                                 //文档资源，包括菜单、图标、字符串
    RUNTIME_CLASS(CMdi071Doc),                      //文档类
    RUNTIME_CLASS(CChildFrame),                     //子框架窗口
    RUNTIME_CLASS(CMdi071View));                    //视图类
    AddDocTemplate(pDocTemplate);                   //添加文档模板
//创建多文档主框架窗口
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))          //创建窗口实例
    return FALSE;
m_pMainWnd = pMainFrame;                            //设为主窗口
//获取命令行参数
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
//处理命令行消息
if (!ProcessShellCommand(cmdInfo))
```

```
        return FALSE;
    //显示主窗口并更新
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
```

多文档程序可以有多个文档模板，CMultiDocTemplate 类定义了多文档程序的文档模板，构造函数格式如下：

```
    CMultiDocTemplate::CMultiDocTemplate(UINT  nIDResource,CRuntimeClass*  pDocClass,
CruntimeClass * pFrameClass,CRuntimeClass* pViewClass)
```

参数如下。

- nIDResource：当前文档类型使用的资源 ID。
- pDocClass：文档类的 CRuntimeClass 类指针。
- pFrameClass：框架类的 CRuntimeClass 类指针，可为 CMDIChildWnd 派生类。
- pViewClass：视图类的 CRuntimeClass 类指针。

AddDocTemplate 函数将模板添加到文档模板列表中，每调用一次该函数，添加一个新的文档模板。LoadFrame 函数根据资源 ID 创建框架窗口实例，并设为主窗口。ParseCommandLine 函数获取程序的命令行参数，存放到 CCommandLineInfo 类对象中，格式如下：

```
    void CWinApp::ParseCommandLine(CCommandLineInfo& rCmdInfo)
```

参数如下。

- rCmdInfo CCommandLineInfo 类对象的引用，存放命令行参数。

ProcessShellCommand 函数根据命令行参数，执行特定功能，格式如下：

```
    BOOL CWinApp::ProcessShellCommand(CCommandLineInfo& rCmdInfo)
```

参数如下。

- rCmdInfo CCommandLineInfo 类对象的引用。

返回值：若成功处理 Shell 命令，返回非零值，否则返回 0。

（2）在 MFC 源代码文件 APPUI2.CPP 中找到 ParseCommandLine 函数的具体实现，部分代码如下：

```
    BOOL CWinApp::ProcessShellCommand(CCommandLineInfo& rCmdInfo)
    {
        BOOL bResult = TRUE;                                    //返回结果
        switch (rCmdInfo.m_nShellCommand)                       //消息类型
        {
        case CCommandLineInfo::FileNew:                         //新建文档
                OnFileNew(); break;
        case CCommandLineInfo::FileOpen:                        //打开文档
            if (!OpenDocumentFile(rCmdInfo.m_strFileName))
                bResult = FALSE;
            break;

        return bResult;
    }
```

根据命令行参数的值，若 FileNew，调用 CWinApp 类的 OnFileNew 函数，内部再调用 CDocManager 类的 OnFileNew 函数。

（3）在 DOCMGR.CPP 中找到 CDocManager::OnFileNew 函数的实现，部分代码如下：

```
    void CDocManager::OnFileNew()
    {
        if (m_templateList.IsEmpty())                    //若文档模板列表为空，直接返回
            return;
```

```
    CDocTemplate* pTemplate = (CDocTemplate*)m_templateList.GetHead();
                                                        //获取第一个文档模板
    if (m_templateList.GetCount() > 1)                  //若文档模板列表数目大于 1
    {
        CNewTypeDlg dlg(&m_templateList);               //“新建”对话框，选择文档类型
        int nID = dlg.DoModal();                        //以模态形式显示对话框
        if (nID == IDOK)                                //若单击“确定”按钮
            pTemplate = dlg.m_pSelectedTemplate;        //获取选择的文档类型
        else
            return;
    }
    pTemplate->OpenDocumentFile(NULL); //调用 CMultiDocTemplate::OpenDocumentFile 函数
}
```

若 CDocManager 类对象的文档模板列表为空，直接返回。GetHead 函数获取第 1 个文档模板的指针，存放到 pTemplate 中。GetCount 函数获取列表的数目，若大于 1，则弹出“新建”模态对话框，用于选择新建的文档类型，如图 10-2 所示。

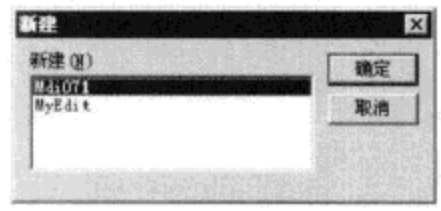


图 10-2　选择文档类型

若命令行参数为打开文档，则调用 CWinApp 类的 OpenDocumentFile 函数，则再调用 CDocManager 类的 OpenDocumentFile 函数，根据传入的参数，选择一个最合适的文档模板，再调用 CMultiDocTemplate 类的 OpenDocumentFile 函数，两种命令行参数最终都调用该函数。

（4）在 DOCMULTI.CPP 中找到 CMultiDocTemplate::OpenDocumentFile 函数的实现，部分代码如下：

```
    CDocument*      CMultiDocTemplate::OpenDocumentFile(LPCTSTR      lpszPathName,BOOL
bMakeVisible)
    {
        CDocument* pDocument = CreateNewDocument();             //创建新文档对象
        CFrameWnd* pFrame = CreateNewFrame(pDocument, NULL); //创建包含文档和视图的框架窗口
        if (lpszPathName == NULL)                               //若文件路径为空，新建文档
        {
            SetDefaultTitle(pDocument);                         //设置默认标题
            if (!bMakeVisible)
                pDocument->m_bEmbedded = TRUE;                  //设为 OLE 嵌入文档
            if (!pDocument->OnNewDocument())                    //新建文档
                return NULL;
            m_nUntitledCount++;                                 //计数器，用于设置文档窗口标题
        }
        else
        {
            CWaitCursor wait;                                   //等待光标
            if (!pDocument->OnOpenDocument(lpszPathName)) //打开已有文件
            {
                pFrame->DestroyWindow();
                return NULL;
            }
            pDocument->SetPathName(lpszPathName);               //保存文件路径
        }
        InitialUpdateFrame(pFrame, pDocument, bMakeVisible);  //初始化框架窗口
        return pDocument;
    }
```

每次调用该函数，都调用 CreateNewDocument 函数创建一个新的文档对象，每个子窗口都有自己独有的文档对象，不同子窗口之间的操作互不影响。CreateNewFrame 函数根据文档模板的运行时类信息，创建框架窗口，并关联文档和视图。

若 lpszPathName 文件路径为空，则创建新文档。SetDefaultTitle 函数设置默认的标题，m_nUntitledCount 记录已新建的文档数目，每创建一个新文档时自动累加，根据其值设置标题名，如 Mdi0711、Mdi0712、Mdi0713。OnNewDocument 函数用于新建一个文档。

若文件路径不为空，则打开一个已有文档。CWaitCursor 类对象用于显示等待光标，当类对象超出作用域后，自动恢复为原始光标。OnOpenDocument 函数用于打开已有文档，SetPathName 函数保存文件路径。InitialUpdateFrame 函数初始化框架，并向视图发送 WM_INITIALUPDATE 消息。

（5）若程序只有一个文档模板，自动创建一个子窗口，若有两个及以上的文档模板，弹出“新建”对话框，选择一种文档类型，创建一个子窗口，ProcessShellCommand 函数执行完毕，程序流程返回到 InitInstance 函数中。

ShowWindow 函数根据 m_nCmdShow 的值显示窗口，如 SW_SHOW 表示以当前大小和位置显示窗口，SW_HIDE 表示隐藏窗口，SW_SHOWMAXIMIZED 表示最大化显示。UpdateWindow 函数向窗口发送 WM_PAINT 消息，重绘框架窗口。

单文档每次调用 OnOpenDocument 函数时，先判断是否已存在文档对象，若存在，使用已有的文档对象。不同于单文档，多文档每次调用 OnOpenDocument 函数，都会创建一个新的文档对象。单文档程序自始自终只有一个 CDocument 类对象，而多文档每新建一个子窗口，都会创建一个 CDocument 类对象，并同时创建框架和视图对象。单文档使用 CMainFrame 作为文档模板的框架窗口，而多文档使用 CChildFrame 作为文档模板的子框架窗口，CMainFrame 是整个程序的框架窗口。

10.2 类联系方式

在实际应用中，经常需要在多个类之间交换数据，调用另一个类中的函数，框架提供一系列函数用于多个类之间的交互（以多文档工程 Mdi071 为例）。

（1）所有类中获取 App 类指针：

```
CWinApp* pApp=AfxGetApp();
CMdi071App* pMyApp=( CMdi071App*)pApp;
```

（2）所有类中获取 CMainFrame 类指针：

```
CMainFrame* pFrame=(CMainFrame*)AfxGetMainWnd();
CMainFrame* pFrame=(CMainFrame*)(AfxGetApp()->m_pMainWnd);
```

（3）获取当前活动的 MDI 子窗口指针：

```
CMainFrame* pMainFrame=(CMainFrame*)AfxGetMainWnd();
CChildFrame* pActiveChild=(CChildFrame*)pMainFrame->MDIGetActive();
```

多文档程序可以有多个 MDI 子窗口，但只有一个活动子窗口，MDIGetActive 函数获取当前活动的子窗口指针，格式如下：

```
CMDIChildWnd* CMDIFrameWnd::MDIGetActive(BOOL* pbMaximized = NULL) const
```

参数如下。

- pbMaximized：输出参数，若当前活动子窗口为最大化，则为 TRUE，否则为 FALSE。

返回值：活动子窗口的指针。

（4）获取活动视图、文档的指针：

```
CMainFrame* pMainFrame=(CMainFrame*)AfxGetMainWnd();
CChildFrame* pActiveChild=(CChildFrame*)pMainFrame->MDIGetActive();
CMdi071View* pActiveView=(CMdi071View*)pActiveChild->GetActiveView();
CMdi071Doc* pActiveDoc=(CMdi071Doc*)pActiveChild->GetActiveDocument();
```

先获取活动子窗口的指针，再调用 GetActiveView 函数获取活动视图，调用 GetActive Document 函数获取活动文档。

Tips GetActiveView、GetActiveDocument 函数返回的指针类型进行转换时，应先使用 GetRuntimeClass 函数获取运行时的类信息，根据类实际信息，再强制转换为对应类型。

10.3 多文档视图

多文档程序的优势在于能够同时显示多种类型的文档，自动生成的 MDI 程序只有一个文档模板，可手动添加视图类、文档类，实现多种类型文档的创建、显示、编辑。MFC 框架已经完成多文档程序的大部分工作，只需要在框架基础上，添加新增功能，即可实现强大的 MDI 程序。

10.3.1 添加文档模板

【实例 10-2】在多文档工程 Mdi071 基础上，添加视图、文档类，构建新的文档模板，实现两种类型文档的新建、显示、编辑操作。

（1）打开多文档工程 Mdi071，选择 Insert|New Class 命令，弹出 New Class 窗口，如图 10-3 所示。

（2）Class type 组合框选择 MFC Class 项，Name 编辑框输入 MyEdit，Base class 组合框选择 CDocument 项，单击 OK 按钮添加类。再添加一个类，Name 设为 MyEdit，Base class 选择 CEditView 项。

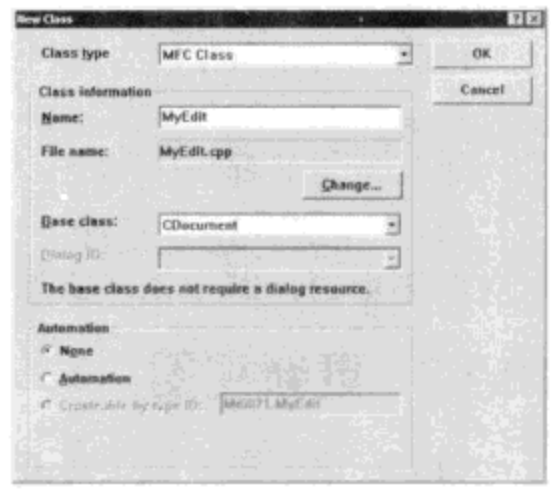


图 10-3　添加文档、视图类

（3）在资源视图选中 Menu 节点下的 IDR_MDI071TYPE 项，按 Ctrl+C、Ctrl+V 组合键，复制一个新菜单，修改 ID 为 IDR_MDI_MYEDIT，双击该项，打开菜单资源，如图 10-4 所示。

（4）添加一个菜单项，在属性窗口里，取消勾选 Pop-up 复选框，设置 ID 为 ID_MENU_CLEAR，设置 Caption 为“清空”。

（5）双击 IDR_MDI071TYPE 项，打开菜单资源，如图 10-5 所示。添加一个菜单项，取消勾选 Pop-up 复选框，设置 ID 为 ID_MENU_REFRESH，设置 Caption 为“刷新”。

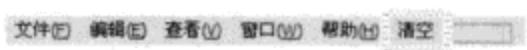


图 10-4　IDR_MDI_MYEDIT 菜单

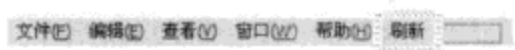


图 10-5　IDR_MDI071TYPE 菜单

（6）双击 String Table 节点下的 String Table 项，打开字符串资源，如图 10-6 所示。

ID	Value	Caption
IDR_MAINFRAME	128	Mdi071&MyEdit
IDR_MDI071TYPE	129	\nMdi071\nMdi071\n\n\nMdi071.Document\nMdi071 Document
IDR_MDI_MYEDIT	130	\nMyEdit\nMyEdit\n\n\nMyEdit.Document\nMyEdit Document

图 10-6　添加字符串资源

（7）双击 IDR_MAINFRAME 项，弹出 String Properties 窗口，设置 Caption 为 Mdi071&MyEdit。

（8）右键单击选择 New String 命令，添加一个字符串资源，设置 ID 为 IDR_MDI_MYEDIT，设置 Caption 为\nMyEdit\nMyEdit\n\n\nMyEdit.Document\nMyEdit Document。

添加的文档字符串有固定的格式，通过 6 个\n 分隔为 7 个子字符串，各个字符串的意义如下。

- 串 1：在 MDI 程序中是空格，IDR_MAINFRAME 项已设置窗口的标题。

- 串 2：默认文档名，自动添加数字加以区分。
- 串 3：默认文档标题，可设为空，不在已使用文档列表中显示。
- 串 4：文档类型及说明信息。
- 串 5：文件扩展名，如.doc。
- 串 6：资源管理器中的注册名，可将文件与程序关联起来。
- 串 7：OLE 对象名称。

（9）在类视图双击 CMdi071App 类下的 InitInstance 项，在 AddDocTemplate (pDocTemplate); 后面添加如下代码：

```
pDocTemplate=new CMultiDocTemplate(                    //新增文档模板
  IDR_MDI_MYEDIT,                                      //文档资源 ID
  RUNTIME_CLASS(CMyEditDoc),                           //自定义文档类
  RUNTIME_CLASS(CChildFrame),                          //子框架类
  RUNTIME_CLASS(CMyEditView));                         //自定义视图类
AddDocTemplate(pDocTemplate);                          //添加文档模板
```

根据文档资源 ID、文档类、框架类、视图类，构造新的 CMultiDocTemplate 对象，RUNTIME_CLASS 宏获取类的运行时信息，AddDocTemplate 函数将新建的文档模板添加到列表中。

（10）在当前 CPP 文件的开头处，包含 CMyEditDoc、CMyEditView 类的头文件，代码如下：

```
#include "MyEditDoc.h"
#include "MyEditView.h"
```

（11）生成程序并运行，自动弹出“新建”对话框，选择一项后，单击“确定”按钮，创建对应的子窗口，若单击“取消”按钮，则只显示主框架窗口。

10.3.2 更新视图

（1）在类视图右键单击 CMdi071Doc 项，在弹出的快捷菜单中选择 Add Member Variable 命令，添加一个 CPoint 类型的变量 m_pt[4]。

（2）双击 CMdi071Doc 类下的 CMdi071Doc 项，添加如下代码：

```
CMdi071Doc::CMdi071Doc()
{
    for(int i=0;i<sizeof(m_pt)/sizeof(CPoint);i++)     //遍历数组
    {
        m_pt[i].x=i*20;                                //设置坐标值
        m_pt[i].y=i*20;
    }
}
```

（3）双击 CMdi071View 类下的 OnDraw 项，添加如下代码：

```
void CMdi071View::OnDraw(CDC* pDC)
{
    CMdi071Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    int nCount=sizeof(pDoc->m_pt)/sizeof(CPoint);      //获取数组大小
    pDC->Polygon(pDoc->m_pt,nCount);                   //绘制多边形
}
```

（4）按 Ctrl+W 组合键打开类向导窗口，为 ID_MENU_REFRESH 添加 CMdi071View 类的 COMMAND 消息处理函数，添加如下代码：

```
void CMdi071View::OnMenuRefresh()
{
    CMdi071Doc* pDoc = GetDocument();
```

```
    int nCount=sizeof(pDoc->m_pt)/sizeof(CPoint);      //获取数组大小
    CRect rcClient;
    GetClientRect(rcClient);                           //获取视图客户区矩形
    srand((unsigned)time(NULL));                       //设置随机数种子
    for(int i=0;i<nCount;i++)                          //遍历坐标数组
    {
        pDoc->m_pt[i].x=rand()%rcClient.Width();       //随机设置坐标值
        pDoc->m_pt[i].y=rand()%rcClient.Height();
    }
    Invalidate();                                      //强制重绘视图
}
```

（5）按 Ctrl+W 组合键打开类向导窗口，为 ID_MENU_CLEAR 添加 CMyEditView 类的 COMMAND 消息处理函数，添加如下代码：

```
void CMyEditView::OnMenuClear()
{
    GetEditCtrl().SetWindowText("");                   //清空编辑框中的文本
}
```

（6）生成程序并运行，创建两个不同文档类型的子窗口，如图 10-7 所示。当活动子窗口改变后，菜单栏自动切换为对应的菜单资源，若没有活动子窗口，显示 IDR_MAINFRAME 对应的菜单资源。每个子窗口使用单独的文档数据，所有子窗口间互不影响。

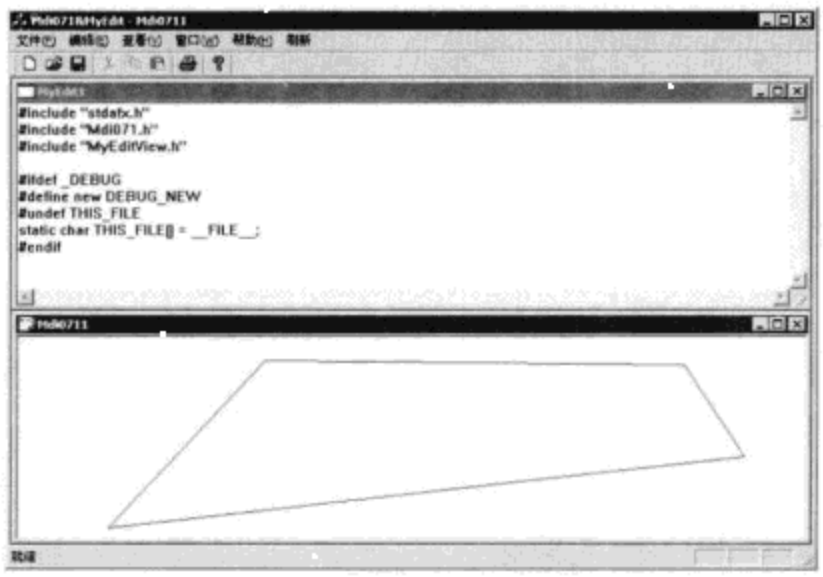


图 10-7　多文档视图

10.4 小结

前面介绍了单文档应用程序，在此介绍多文档应用程序。单文档是指在一个程序打开一个文档，而多文档程序可在一个程序中打开多种文档。本章首先通过实例，向读者介绍了多文档应用程序中的各个生成类，接着介绍了如何在各个生成类间交换数据。最后介绍了多文档视图的进一步操作与应用。

10.5 习题

1．多文档应用程序和单文档应用程序有什么不同？

2．在多文档应用程序中，类与类之间如何联系？

3．如何创建多文档视图的应用程序？

4．编写一个程序，创建一个多文档应用程序的文版编辑器，要求具有文本录入、编辑和删除的基本功能，并能打开当前已有的文本文件。

第4篇 Visual C++编程

第11章 文件编程

文件处理是日常办公最主要的操作，启动一个文件处理软件，如 Word、Visual Studio 2005、CAD、Photoshop 等，打开相关文件，在软件中读取文件内容，并可显示、编辑，处理完成后，保存修改或另存为另一个文件。在 Windows 资源管理器里，可以创建、复制、重命名、删除文件，搜索指定目录下的所有文件，获取文件的创建、修改、访问日期，以及文件大小等信息，学习文件编程，可以通过程序操作文件，实现特定功能。

11.1 文件类

MFC 提供 CFile 及其他几个类，用于操作文件，如打开、读取、写入、重命名、删除文件等，这些功能能够满足基本需求。若要实现更多的功能，如移动文件、复制文件、设置文件属性等操作，或与目录相关的操作，如创建、删除、设置当前目录，需要调用系统 API 函数。

11.1.1 文件格式

文件格式分为两种：明码格式和暗码格式。明码格式为文本文件，可用记事本或写字板等文字处理程序打开，能看到原始的文件内容，如 TXT、CPP 文件，MFC 提供 CStdioFile 类可以很方便地读取和写入文本文件，CStdioFile 类派生自 CFile 类，在 CFile 类功能的基础上，新增两个函数用于读写文本。

ReadString 函数用于读取一行文本，遇到换行符时停止读取，格式如下：

```
BOOL CStdioFile::ReadString(CString& rString)
```

参数如下。

- rString：存放当前行的字符串文本。

返回值：若到文件末尾则返回 FALSE，否则返回 TRUE。

WriteString 函数用于写入文本，可在文本中加入换行符“\n”，格式如下：

```
void CStdioFile::WriteString(LPCTSTR lpsz)
```

参数如下。

- lpsz：要写入的文本。

暗码格式为二进制文件，只能用专门软件打开，如 DOC 文件只能用 Word 软件打开，JPG 文件只能用图像软件打开，若拖放到记事本中打开，则显示一堆乱码。二进制文件有着固定的格式，程序根据文件格式以字节为单位进行读取，若不知道文件内部格式，是无法读取文件内容的。

一些软件提供两种格式的文件，一种不公开文件格式，只能用本软件打开，另一种公开格式，可编写程序读取文件中的数据，公开格式的文件常用于不同软件间的数据交换。如 AutoCAD 软件有 DWG 和 DXF 两种格式，DWG 不公开文件格式，只能在 CAD 软件中打开，而 DXF 公开其内部格式，根据格式可编写程序读取，获取文件数据后，再存储为另一种格式，从而实现数据交换，减少重复性工作。

相对于文本文件，二进制文件占用空间小，存取速度快，且数据保密，是常用的文件保存方法，但二进制文件读取难度大，容易出错，不如文本文件读取方便，在实际开发中应根据需

要，选择合适的存储方式。

MFC 提供 CFile 类用于读写各种文件，主要函数如下。

（1）Open 函数打开指定路径文件，格式如下：

```
BOOL CFile::Open(LPCTSTR lpszFileName, UINT nOpenFlags, CFileException* pError = NULL)
```

参数如下。

- lpszFileName：打开文件的路径，相对或绝对路径。
- nOpenFlags：打开标志，如 CFile::modeCreate 创建一个新文件，若文件已存在，则清空已有文件，CFile::modeRead 以只读方式打开文件，CFile::modeWrite 以只写方式打开文件。
- pError：文件异常类指针，存放异常信息。

返回值：若成功则返回非零值，否则返回 0。

（2）Close 函数用于关闭文件对象，类对象析构时自动调用该函数，格式如下：

```
void CFile::Close()
```

（3）Read 函数读取指定字节长度的数据，格式如下：

```
UINT CFile::Read(void* lpBuf,UINT nCount)
```

参数如下。

- lpBuf：指向缓冲区的指针，用于接收数据。
- nCount：要读取的字节数。

返回值：实际读取的字节数。

（4）Write 函数写入指定字节长度的数据到文件中，格式如下：

```
void CFile::Write(const void* lpBuf,UINT nCount)
```

参数如下。

- lpBuf：要写入数据的起始指针。
- nCount：写入的字节数。

（5）Flush 函数强制将缓冲区中的数据写入文件，格式如下：

```
void CFile::Flush()
```

（6）Seek 函数用于设置当前文件指针，读写函数根据文件指针执行操作，格式如下：

```
LONG CFile::Seek(LONG lOff,UINT nFrom)
```

参数如下。

- lOff：指针偏移的字节数。
- nFrom：开始偏移的位置，如 CFile::begin 表示从文件头开始向后偏移，CFile::current 表示从当前位置开始向后偏移，CFile::end 从文件尾开始向前偏移。

返回值：从文件头开始的新偏移量。

（7）GetPosition 函数获取当前文件指针的位置，格式如下：

```
DWORD CFile::GetPosition () const
```

返回值：文件指针的位置。

11.1.2　文件对话框

在日常办公中经常需要打开和保存文件，为统一操作界面，减少重复性开发工作，Windows 提供了标准的文件对话框，CFileDialog 类封装了文件对话框，提供一系列函数用于操作文件对话框，常用函数如下所示。

（1）CFileDialog 构造函数用于创建一个文件对话框对象，格式如下：

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

参数如下。

- bOpenFileDialog：标识符，若为 TRUE 则用于打开文件，若为 FALSE 则用于保存文件。
- lpszDefExt：默认文件后缀名。
- lpszFileName：默认文件名。
- dwFlags：标志组合，如 OFN_HIDEREADONLY 表示隐藏文件为只读，OFN_ALLOWMULTISELECT 表示可多选，OFN_ENABLESIZING 表示能调整对话框大小。
- lpszFilter：文件类型过滤字符串。
- pParentWnd：父窗口的指针。

Tips 使用文件对话框会改变程序当前目录的位置，如将文件保存到 E 盘后，GetCurrentDirectory 函数得到的路径是 E:\，若程序中其他位置需要使用 GetCurrentDirectory 函数获取当前目录，应在使用文件对话框后，调用 SetCurrentDirectory 函数恢复当前目录的位置。

（2）DoModal 函数以模态形式显示文件对话框，格式如下：

```
int CFileDialog::DoModal()
```

返回值：若单击“打开”或“保存”按钮，则返回 IDOK，否则返回 IDCANCEL。

（3）GetPathName 函数获取选择文件的绝对路径，格式如下：

```
CString CFileDialog::GetPathName() const
```

返回值：选择文件的完整路径，如 D:\123.txt。

（4）GetFileName 函数获取选择文件的文件名，格式如下：

```
CString CFileDialog::GetFileName() const
```

返回值：选择文件的文件名，如 123.txt。

（5）GetFileExt 函数获取文件的后缀名，格式如下：

```
CString CFileDialog::GetFileExt() const
```

返回值：选择文件的后缀名，如 txt。

（6）GetFileTitle 函数获取文件的标题，格式如下：

```
CString CFileDialog::GetFileTitle() const
```

返回值：选择文件的标题，如 123。

Tips 以上 4 个函数仅在单选文件时可用，若使用 OFN_ALLOWMULTISELECT 标志允许多选，且同时选择多个文件，这 4 个函数不能返回正确值，若只选择一个文件，可返回正确值。

（7）GetStartPosition 函数获取第 1 个选中文件的位置，用于多选文件，格式如下：

```
POSITION CFileDialog::GetStartPosition() const
```

返回值：第 1 个文件的 POSITION 值，若选择为空则返回 NULL。

（8）GetNextPathName 函数根据 POSITION 值获取文件路径，用于遍历选中文件，格式如下：

```
CString CFileDialog::GetNextPathName(POSITION& pos) const
```

参数如下。

- pos：当前文件的位置值，调用函数后，指向下一个文件。

返回值：当前位置的文件路径。

【实例 11-1】创建一个文件对话框对象，以模态对话框形式显示，显示所有选择文件的路径。

```
CString strFilter="jpg 图像(*.jpg)|*.jpg|bmp 图像(*.bmp)|*.bmp|所有文件(*.*)|*.*||";
                                                          //文件类型过滤字符串
CFileDialog fileDlg(TRUE,NULL,"未命名图像",
    OFN_FILEMUSTEXIST|OFN_ALLOWMULTISELECT|OFN_ENABLESIZING,  //标志组合
     (LPCTSTR)strFilter,NULL);
CStringArray arrayFile;
if(fileDlg.DoModal()==IDOK)                                  //显示文件对话框
{
    POSITION pos=fileDlg.GetStartPosition();                 //获取第 1 个选中文件的位置
    while(pos)                                               //遍历选中文件
    {
        CString strName=fileDlg.GetNextPathName(pos);        //获取当前位置的文件路径
        arrayFile.Add(strName);                              //添加到可变数组中
    }
}
CString temp;
for(int i=0;i<arrayFile.GetSize();i++)                       //遍历可变数组
    temp+=arrayFile.GetAt(i)+"\n";
AfxMessageBox(temp);                                         //弹出提示框
```

strFilter 为文件类型过滤字符串，第 1 个|分隔符前的为类型说明信息，显示给用户看，第 1 个|分隔符后的为类型匹配符，*匹配任意字符串，如*.jpg 匹配所有后缀名为 jpg 的文件，*.*匹配所有文件，不同文件类型间也用|分隔符，最后使用||作为结束标志。

CFileDialog 类的构造函数中，参数 1 为 TRUE 表示打开文件，参数 2 为 NULL 表示默认后缀名为空，参数 3 作为“文件名”编辑框的默认值，参数 4 为标志组合，依次为文件必须存在、允许多选、可调整对话框大小，参数 5 为文件过滤字符串，依次为 JPG 图像、BMP 图像、所有文件，当选中一项后，如 JPG 图像，文件对话框只显示所有后缀名为 JPG 的文件，参数 6 为父窗口的指针。

DoModal 函数显示模态对话框，在关闭文件对话框之前不能进行其他操作，若单击“打开”按钮或双击一个文件，则返回 IDOK，表示成功选择一个或多个文件，如图 11-1 所示。

在多选状态下，GetStartPosition 函数获取第 1 个文件的位置，存放到 pos 中，利用 while 循环，遍历所有选中文件，GetNextPathName 函数根据 pos 值获取文件路径，并将 pos 指向下一个文件。Add 函数将文件路径存放到可变数组 arrayFile 中，AfxMessageBox 函数弹出信息提示框，显示所有选择文件路径，如图 11-2 所示。

图 11-1　文件对话框打开多个文件

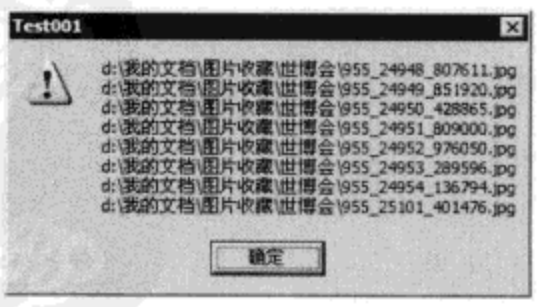


图 11-2　选择的多个文件路径

11.1.3 文件操作

常用的文件操作有新建、重命名、复制、移动、删除等，以及相关的目录操作。CFile 类仅提供新建、重命名、删除文件功能，更多与文件、目录相关的操作，需要调用系统 API 函数。

Rename 函数用于重命名文件，静态函数，可直接用类名调用，格式如下：

```
static void CFile::Rename(LPCTSTR lpszOldName,LPCTSTR lpszNewName)
```

参数如下。

- lpszOldName：原来的文件路径。
- lpszNewName：新路径。

Remove 函数用于移除一个文件，也是静态函数，格式如下：

```
static void CFile::Remove(LPCTSTR lpszFileName)
```

参数如下。

- lpszFileName：要移除文件的路径。

CopyFile 函数用于复制一个文件到其他位置，格式如下：

```
BOOL CopyFile(LPCTSTR lpExistingFileName,LPCTSTR lpNewFileName,BOOL bFailIfExists)
```

参数如下。

- lpExistingFileName：要复制文件的路径。
- lpNewFileName：新文件的路径。
- bFailIfExists：若为 TRUE，则当新文件已存在时，复制失败，若为 FALSE，则当新文件已存在时，覆盖已有文件，复制成功。

返回值：若成功则返回非零值，否则返回 0。

MoveFile 函数用于移动一个文件或目录，格式如下：

```
BOOL MoveFile(LPCTSTR lpExistingFileName,LPCTSTR lpNewFileName)
```

参数如下。

- lpExistingFileName：要移动的已存在的文件或目录。
- lpNewFileName：新文件名。在移动后，可重命名文件或目录。
- 返回值：若成功则返回非零值，否则返回 0。

Tips 若移动目录，则只能在同一个驱动器内进行，如不能将 D 盘的目录移动到 E 盘里。

CreateDirectory 函数用于创建一个新目录，格式如下：

```
BOOL CreateDirectory(LPCTSTR lpPathName,LPSECURITY_ATTRIBUTES lpSecurityAttributes)
```

参数如下。

- lpPathName：新目录的路径。
- lpSecurityAttributes：目录的安全属性，一般设为 NULL，使用默认属性。

返回值：若成功则返回 TRUE，若目录已存在或其他错误，则返回 FALSE。

RemoveDirectory 函数用于移除一个已存在的空目录，格式如下：

```
BOOL RemoveDirectory(LPCTSTR lpPathName)
```

参数如下。

- lpPathName：要移除目录的路径。

返回值：若成功则返回非零值，否则返回 0。

【实例 11-2】创建一个目录，在目录中创建一个空文件，并执行复制、重命名、移动、删除文件操作。

```
CString strDir="D:\\myDir";
BOOL bCreate=CreateDirectory(strDir,NULL);                    //创建一个新目录
if(bCreate)                                                   //若创建成功
{
    CFile f;                                                  //文件对象
    CString strFilePath=strDir+"\\myFile.dat";                //文件路径
    if(f.Open(strFilePath,                                    //创建并打开新文件
        CFile::modeWrite|CFile::modeCreate|CFile::modeNoTruncate))
        f.Close();                                            //关闭文件
    CString strCopyFile1=strDir+"\\myFile_Copy1.dat";         //复制后的文件路径
    CString strCopyFile2=strDir+"\\myFile_Copy2.dat";
    CString strCopyFile3=strDir+"\\myFile_Copy3.dat";
    CString strCopyFile4=strDir+"\\myFile_Copy4.dat";
    CopyFile(strFilePath,strCopyFile1,TRUE);                  //复制出 4 个文件
    CopyFile(strFilePath,strCopyFile2,TRUE);
    CopyFile(strFilePath,strCopyFile3,TRUE);
    CopyFile(strFilePath,strCopyFile4,TRUE);
    CString strNewName=strDir+"\\myFile_Copy2_Rename.dat";
    CFile::Rename(strCopyFile2,strNewName);                   //重命名拷贝 2
    int nIndex=strCopyFile3.ReverseFind('\\');                //反向查找\
    CString strMoveFileTitle=strCopyFile3.Right(strCopyFile3.GetLength()-nIndex-1);
                                                              //获取文件标题
    MoveFile(strCopyFile3,"D:\\"+strMoveFileTitle);           //将拷贝 3 移动到 D 盘根目录
    CFile::Remove(strCopyFile4);                              //移除拷贝 4
}
```

CreateDirectory 函数创建一个新目录，若目录已存在，则返回 FALSE，若创建成功，则在 D 盘根目录出现一个新文件夹 myDir。Open 函数在该目录内创建并打开一个新文件 myFile.dat，参数 2 中的 modeWrite 表示可写入，modeCreate 函数表示创建新文件，modeNoTruncate 与 modeCreate 组合使用，表示若要创建的文件已存在，其内容不变，若不使用 modeNoTruncate，modeCreate 将清空已存在文件的内容。再调用 Close 函数关闭文件对象，从而创建一个空文件。

CopyFile 函数将创建的空文件复制 4 份，参数 1 表示源文件，参数 2 表示复制路径，参数 3 为 TRUE 表示不覆盖已存在文件。CFile::Rename 函数将复制的文件 2 重命名为 myFile_Copy2_Rename.dat。调用 ReverseFind 函数反向查找最后一个“\”符号，Right 函数获取“\”后的子字符串，即文件名，MoveFile 函数将复制的文件 3 移动到 D 盘根目录。CFile::Remove 函数移除复制的文件 4。

执行代码后，在 D 盘根目录出现 myDir 文件夹，以及移动的文件 myFile_Copy3.dat，如图 11-3 所示。在 myDir 文件夹中，出现创建的文件 myFile.dat，以及复制的文件 myFile_Copy1.dat、重命名后的文件 myFile_Copy2_Rename.dat，如图 11-4 所示。

图 11-3　目录操作

图 11-4　文件操作

11.1.4 文件状态

硬盘上存放的文件具有一些状态信息，如创建、修改、访问时间，文件大小，文件路径，是否为只读、系统、存档文件，MFC 提供 CFileStatus 结构体用于存放文件状态信息，格式如下：

```
struct CFileStatus
{
    CTime m_ctime;                          //创建时间
    CTime m_mtime;                          //最后一次修改时间
    CTime m_atime;                          //最后一次访问时间
    LONG m_size;                            //文件大小，以字节为单位
    BYTE m_attribute;                       //属性集合
    TCHAR m_szFullName[_MAX_PATH];          //文件路径
};
```

GetStatus 函数获取文件的状态信息，静态函数，格式如下：

```
static BOOL CFile::GetStatus(LPCSTR lpszFileName, CFileStatus& rStatus)
```

参数如下。

- lpszFileName：文件路径。
- rStatus：CFileStatus 结构体的引用，存放状态信息。

返回值：若成功则返回 TRUE，否则返回 FALSE。

SetStatus 函数用于设置文件的状态信息，也为静态函数，格式如下：

```
static void CFile::SetStatus(LPCTSTR lpszFileName,const CFileStatus& status)
```

参数如下。

- lpszFileName：文件路径。
- status：要设置的状态信息。

【实例 11-3】获取并显示某个文件的状态信息。

```
    CFileStatus status;
    CFile::GetStatus("C:\\pagefile.sys",status);                    //获取文件的状态信息
    CString strInfo,strElem;
    strElem.Format("文件名称：%s\n",status.m_szFullName);            //文件路径
    strInfo+=strElem;
    strElem.Format("文件长度：%d MB\n",status.m_size/1024/1024);     //文件大小，转换为 MB
    strInfo+=strElem;
    CTime time=status.m_ctime;                                      //创建时间
    strElem.Format("创建时间：%d-%d-%d\n",time.GetYear(),time.GetMonth(),time.GetDay());
    strInfo+=strElem;
    time=status.m_mtime;                                            //最后一次修改时间
    strElem.Format("最后一次修改时间：%d-%d-%d\n",time.GetYear(),time.GetMonth(),time.
GetDay());
    strInfo+=strElem;
    time=status.m_atime;                                            //最后一次访问时间
    strElem.Format("最后一次访问时间：%d-%d-%d\n",time.GetYear(),time.GetMonth(),time.
GetDay());
    strInfo+=strElem;
    BYTE bAttr=status.m_attribute;                                  //属性信息
    if(bAttr & CFile::hidden)                                       //是否为隐藏文件
        strInfo+="隐藏文件\n";
    if(bAttr & CFile::system)                                       //是否为系统文件
        strInfo+="系统文件\n";
    if(bAttr & CFile::readOnly)                                     //是否为只读文件
        strInfo+="只读文件\n";
    if(bAttr & CFile::archive)                                      //是否为存档文件
```

```
    strInfo+="存档文件\n";
AfxMessageBox(strInfo);                                    //弹出信息提示框
```

GetStatus 函数获取文件的状态信息，存放到 status 中。pagefile.sys 是系统虚拟内存文件，默认为隐藏、系统文件。strElem 存放各项状态信息，并累加到 strInfo 中。

m_attribute 存放文件的属性信息，CFile 类提供枚举成员 Attribute 用来判断属性信息，格式如下：

```
enum Attribute {
    normal =    0x00,
    readOnly =  0x01,
    hidden =    0x02,
    system =    0x04,
    volume =    0x08,
    directory = 0x10,
    archive =   0x20
};
```

使用与运算符&可以判断是否包含某个属性，如 bAttr & CFile::hidden，若文件具有隐藏属性，则运算结果为 TRUE，否则为 FALSE。执行代码后，弹出 pagefile.sys 文件的状态信息，如图 11-5 所示。

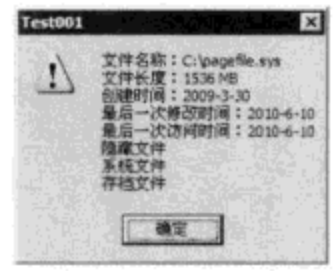


图 11-5　文件状态信息

11.1.5　读/写文本文件

文本文件可用任意文本处理程序打开，与文件的后缀名无关，如 dsw、dsp、cpp、h、txt、clw、rc 均属于文本文件，可拖放到记事本中打开。

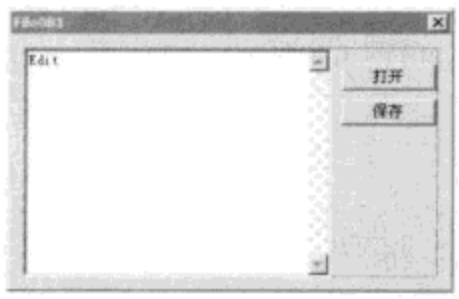


图 11-6　对话框模板

MFC 提供 CStdioFile 类专门用于处理文本文件，可以很方便地读取和写入文本，也可使用 CFile 类读写文本文件，但较为复杂。

【实例 11-4】新建一个对话框工程 File081，实现读取和写入文本文件。

（1）新建对话框工程 File081，在对话框模板上，拖放两个按钮、一个编辑框控件，如图 11-6 所示。

（2）设置两个按钮的 Caption 依次为“打开”、“保存”，ID 依次为 IDC_BUTTON_OPEN、IDC_BUTTON_SAVE。

（3）打开编辑框的 Styles 属性页，取消勾选 Auto HScroll 复选框，勾选 Multiline、Vertical scroll、Auto VScroll 复选框。

（4）双击两个按钮，添加单击处理函数，添加如下代码：

```
void CFile081Dlg::OnButtonOpen()                           //读取文件
{
    CStdioFile f;
    if(f.Open("D:\\myText.dat",CFile::modeRead))  //以只读方式打开文件
    {
        CString strLine,strText;
        while(f.ReadString(strLine))                       //循环读取所有行，并添加换行符
            strText+=strLine+"\r\n";
        GetDlgItem(IDC_EDIT1)->SetWindowText(strText);     //在编辑框中显示文件内容
    }
}
void CFile081Dlg::OnButtonSave()                           //保存文件
{
    CStdioFile f;
    if(f.Open("D:\\myText.dat",CFile::modeWrite|CFile::modeCreate))//创建并写入文件
    {
        CString strText;
        GetDlgItem(IDC_EDIT1)->GetWindowText(strText);     //获取编辑框中的内容
```

```
            f.WriteString(strText);                                    //写入文本文件中
        }
    }
```

读取文件时，Open 函数以只读方式打开 myText.dat 文件，若文件不存在，则返回 FALSE。若存在，利用 while 循环读取所有行，并在每一行文本后添加换行符\r\n，累加到 strText 中，并在编辑框中显示。

保存文件时，Open 函数创建并写入文件，modeCreate 表示创建新文件，若文件已存在，则清空已有文件的内容。GetWindowText 函数获取编辑框的内容，WriteString 函数将文本写入文件中。

Tips 字符串使用\n 换行，编辑框控件使用\r\n 换行。

（5）生成程序并运行，如图 11-7 所示。在编辑框中输入文本，单击“保存”按钮后，在 D 盘根目录出现新文件 myText.dat，用记事本打开，如图 11-8 所示。关闭再运行程序，单击“打开”按钮后，编辑框中显示保存的文件内容。

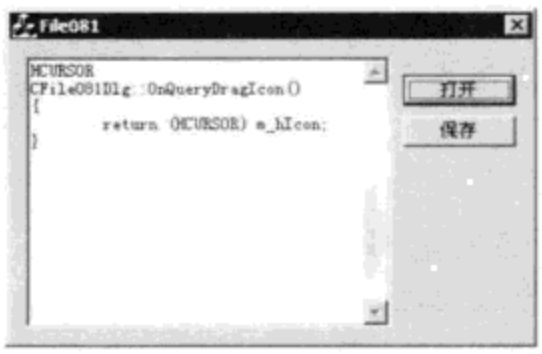


图 11-7 读写文本文件

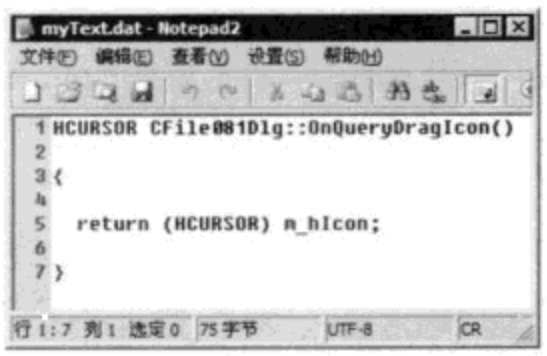


图 11-8 文本文件内容

11.1.6 读/写二进制文件

二进制文件以 0、1 形式存储数据，以字节为单位存取，若用记事本打开，则显示为乱码字符。每种二进制文件都有其固定的格式，若要读取某种二进制文件，则必须先了解其内部格式，如 Word 软件的 DOC 文件，在获取其内部格式前，是无法读取其文件内容的，JPG、BMP 等图形文件的内部格式是公开的，可根据内部格式，以字节为单位编程读取、显示。MFC 提供 CFile 类可用于读写二进制文件，相对于文本文件，二进制文件读写较为复杂，但数据保密，且体积较小、读取更快。

【实例 11-5】新建一个对话框工程 File082，显示个人信息，以二进制形式保存和读取信息。

（1）新建对话框工程 File082，在对话框模板上，拖放三个静态控件、三个编辑框、两个按钮控件，如图 11-9 所示。

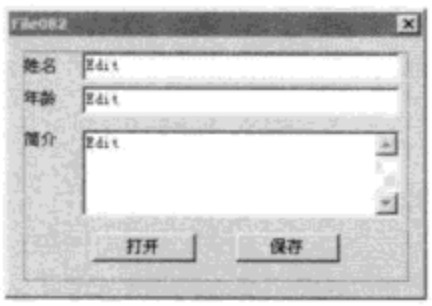


图 11-9 个人信息对话框

（2）设置三个静态控件的 Caption 依次为“姓名”、“年龄”、“简介”。设置三个编辑框的 ID 依次为 IDC_EDIT_NAME、IDC_EDIT_AGE、IDC_EDIT_CONTENT，其中“年龄”编辑框勾选 Number 复选框，“简介”编辑框取消勾选 Auto HScroll 复选框，勾选 Multiline、Vertical scroll、Auto VScroll 复选框。

（3）设置两个按钮的 Caption 依次为“打开”、“保存”，ID 依次为 IDC_BUTTON_OPEN、IDC_BUTTON_SAVE。

（4）按 Ctrl+W 组合键打开类向导，选择 Member Variable 选项卡，双击 IDC_EDIT_AGE 项添加 int 类型的变量 m_nAge，双击 IDC_EDIT_CONTENT 项添加 CString 类型的变量 m_strContent，双击 IDC_EDIT_NAME 项添加 CString 类型的变量 m_strName，单击 OK 按钮保存退出。

（5）在类视图右键单击 CFile082Dlg 项，选择 Add Member Functions 按钮，添加返回类型为

CString 的函数 Encode(CString str)，添加如下代码：

```
CString CFile082Dlg::Encode(CString str)
{
    if(str.IsEmpty())                                    //若字符串为空，则直接返回
        return "";
    CString strOut=str;                                  //输出字符串
    for(int i=0;i<str.GetLength();i++)                   //遍历 str 的所有字节
    {
        TCHAR ch=str.GetAt(i);                           //第 i 个字节的值
        ch+=1;                                           //字节值加 1
        strOut.SetAt(i,ch);                              //新值存入 strOut 中
    }
    return strOut;                                       //返回加密后的字符串
}
```

CFile 类使用二进制形式存储 int、double 等数值类型，而对于字符串类型仍使用文本形式，为实现数据的保密，可先对字符串进行加密处理，将加密后的字符串写入文件。加密的方式多种多样，本函数通过对字符串的每个字节加 1，实现简单的加密效果。

GetLength 函数获取字符串的字节长度，利用 for 循环遍历所有字节，GetAt 函数获取每个字节的值，加 1 处理后，SetAt 函数用处理后的字节值，设置输出字符串对应字节的值。

Tips　在字符串中，一个英文字符占用 1 个字节，一个汉字占用 2 个字节。

（6）再添加一个返回类型为 CString 的函数 Decode(CString str)，添加如下代码：

```
CString CFile082Dlg::Decode(CString str)
{
    if(str.IsEmpty())
        return "";
    CString strOut=str;
    for(int i=0;i<str.GetLength();i++)
    {
        TCHAR ch=str.GetAt(i);                           //第 i 个字节的值
        ch-=1;                                           //字节值减 1
        strOut.SetAt(i,ch);                              //新值存入 strOut 中
    }
    return strOut;                                       //返回解密后的字符串
}
```

在文件中存储的是加密后的字符串，打开文件时，需要将加密的字符串解密，得到原始的字符串。解密和加密是对应的，根据加密的方法进行逆操作，如 Encode 函数对各个字节加 1，则 Decode 函数对各个字节减 1，即可得到解密后的字符串。

（7）在资源视图，双击对话框模板上的“保存”按钮，添加如下代码：

```
void CFile082Dlg::OnButtonSave()
{
    UpdateData(TRUE);                                                    //更新控件变量值
    if(m_strName.IsEmpty() || m_nAge==0 || m_strContent.IsEmpty())//若输入为空，返回
        return;
    CString strFilter="个人信息文件(*.pinfo)|*.pinfo||";     //文件过滤字符串
    CFileDialog  fileDlg(FALSE,"pinfo","个人信息",OFN_FILEMUSTEXIST,(LPCTSTR)
strFilter,NULL);
    CString m_strPath;
    if(fileDlg.DoModal()==IDOK)                                          //弹出保存文件对话框
        m_strPath=fileDlg.GetPathName();                                 //获取保存路径
    else
```

```
        return;
    CFile f;
    if(f.Open(m_strPath,CFile::modeWrite|CFile::modeCreate))  //创建并写入保存文件
    {
        int nNameSize=m_strName.GetLength();                  //“名称”字符串的字节长度
        int nAgeSize=sizeof(m_nAge);                          //“年龄”整型值的字节长度
        int nContentSize=m_strContent.GetLength();            //“简介”字符串的字节长度
        f.Write(&nNameSize,sizeof(int));                      //写入“名称”的字节长度
        CString strNameEncode=Encode(m_strName);              //加密“名称”
        f.Write((LPCTSTR)strNameEncode,nNameSize);            //写入加密后的“名称”
        f.Write(&m_nAge,nAgeSize);                            //写入“年龄”整型值
        f.Write(&nContentSize,sizeof(int));                   //写入“简介”的字节长度
        CString strContentEncode=Encode(m_strContent);        //加密“简介”
        f.Write((LPCTSTR)strContentEncode,nContentSize);      //写入加密后的“简介”
    }
}
```

UpdateData 函数更新控件对应的变量值，若输入为空或年龄为 0，直接返回。strFilter 为文件过滤字符串，只显示后缀名为 pinfo 的文件，pinfo 是自定义的文件保存格式。DoModal 函数显示保存文件模态对话框，GetPathName 函数获取保存文件的路径。

nNameSize、nContentSize 为输入的“名称”、“简介”字符串的字节长度，需要在文件中先写入字符串的长度，以便读取时，确定要读取的字符串的字节长度。Write 函数先写入“名称”的字节长度值，参数 1 为写入变量的起始地址，参数 2 为写入的字节长度，sizeof(int)获取 int 变量的字节长度。

Encode 函数对“名称”、“简介”字符串进行加密处理。Write 函数将加密后的字符串写入文件，(LPCTSTR)获取 CString 对象的字符指针。

pinfo 文件的内部格式如下：

- “名称”字符串的长度，4 个字节。
- “名称”字符串，字节长度由前面的值决定。
- “年龄”值，4 个字节。
- “简介”字符串的长度，4 个字节。
- “简介”字符串，字节长度由前面的值决定。

（8）在资源视图，双击对话框模板上的“打开”按钮，添加如下代码：

```
void CFile082Dlg::OnButtonOpen()
{
    CString strFilter="个人信息文件(*.pinfo)|*.pinfo||";    //文件过滤字符串
    CFileDialog fileDlg(TRUE,"pinfo",NULL,OFN_FILEMUSTEXIST,(LPCTSTR)strFilter, NULL);
    CString m_strPath;
    if(fileDlg.DoModal()==IDOK)                              //显示打开文件对话框
        m_strPath=fileDlg.GetPathName();                     //获取打开文件路径
    else
        return;
    CFile f;
    if(f.Open(m_strPath,CFile::modeRead|CFile::shareDenyRead))//以只读方式打开文件
    {
        int nData;
        f.Read(&nData,sizeof(int));                          //读取一个 int 长度的值，存入 nData 中
        int nNameSize=nData;                                 //获取“名称”字符串的字节长度
        CString strNameEncode;
        LPTSTR pStrName=strNameEncode.GetBuffer(nNameSize);  //获取字符串缓冲区
        f.Read(pStrName,nNameSize);                          //读取 nNameSize 长度的值，存入缓冲区中
        strNameEncode.ReleaseBuffer();                       //释放缓冲区
        m_strName=Decode(strNameEncode);                     //获取解密后的“名称”
        f.Read(&nData,sizeof(int));                          //读取 int 长度的值
        m_nAge=nData;                                        //获取“年龄”值
        f.Read(&nData,sizeof(int));                          //读取 int 长度的值
        int nContentSize=nData;                              //获取“简介”字符串的字节长度
        CString strContentEncode;
        LPTSTR pStrContent=strContentEncode.GetBuffer(nContentSize);
```

```
            f.Read(pStrContent,nContentSize); //读取 nContentSize 长度的值，存入缓冲区中
            strContentEncode.ReleaseBuffer();     //释放缓冲区
            m_strContent=Decode(strContentEncode);          //获取解密后的“简介”
            UpdateData(FALSE);                              //更新控件
        }
    }
```

Read 函数读取一定字节长度的值，参数 1 为缓冲区的起始地址，存放读取的数据，参数 2 为读取的字节长度。GetBuffer 函数获取指定长度的字符缓冲区，返回的 LPTSTR 类型指针可以用来改变缓冲区内的值，将读取的“名称”、“简介”字符串，存入缓冲区中，缓冲区操作结束后，ReleaseBuffer 函数释放缓冲区。

Decode 函数用于对读取的字符串解密，文件中存放加密过的字符串，读取后需要解密。UpdateData 函数更新控件，显示文件中存放的解密后的数据。

（9）生成程序并运行，如图 11-10 所示。输入个人信息后，单击“保存”按钮，弹出保存文件对话框，保存为 pinfo 格式的二进制文件，用记事本打开文件，如图 11-11 所示。关闭再运行程序，单击“打开”按钮，弹出打开文件对话框，选择先前保存的 pinfo 格式文件，程序显示保存的信息。

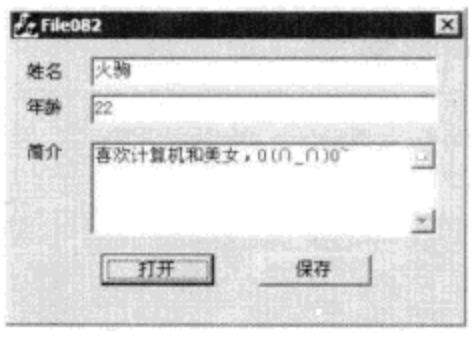


图 11-10　存取二进制文件

图 11-11　pinfo 文件内容

11.2 文件查找

文件查找是一项常用的电脑操作，随着电脑使用时间增长，硬盘中存放的文件随之增加，使用文件查找功能可以快速找到指定类型的文件，如可以搜索某个目录下的所有 Word 文件，或搜索文件内容中包含“CFileFind”字符串的所有文件。

MFC 提供 CFileFind 类可以实现快速文件查找，根据查找匹配符，如“*.doc”匹配所有 DOC 文件，“*.*”匹配所有文件，循环搜索指定目录下的所有匹配文件，并可获取文件名，判断文件是否为目录，常用函数如下所示：

（1）FindFile 函数用于开始查找一个文件，格式如下：

```
BOOL CFileFind::FindFile(LPCTSTR pstrName = NULL, DWORD dwUnused = 0)
```

参数如下。

- pstrName：查找的文件路径字符串，如”D:*.*”表示搜索 D 盘根目录下所有文件。
- dwUnused：保留参数，必须为 0。

返回值：若成功则返回非零值，否则返回 0。

（2）FindNextFile 函数用于查找下一个文件，获取文件信息前必须先调用该函数，格式如下：

```
BOOL CFileFind::FindNextFile()
```

返回值：若查找尚未结束，则返回非零值，若为最后一个文件，返回 0。

（3）GetFilePath 函数获取当前查找文件的全路径，格式如下：

```
CString CFileFind::GetFilePath() const
```

返回值：当前查找文件的完整路径，如 D:\123.dat。

（4）GetFileName 函数获取当前查找文件的文件名，格式如下：

```
CString CFileFind::GetFileName() const
```

返回值：当前查找文件的文件名，如 123.dat。

（5）GetFileTitle 函数获取文件标题，格式如下：

```
CString CFileFind::GetFileTitle() const
```

返回值：文件标题，如 123。

（6）GetRoot 函数获取文件路径的上一级目录，格式如下：

```
CString CFileFind::GetRoot() const
```

返回值：上一级目录，如文件路径为 D:\123.dat，返回 D:\。

（7）GetLength 函数获取文件长度，以字节为单位，格式如下：

```
DWORD CFileFind::GetLength() const
```

返回值：文件长度。

（8）IsDirectory 函数判断当前查找文件是否为目录，格式如下：

```
BOOL CFileFind::IsDirectory() const
```

返回值：若为目录则返回非零值，否则返回 0。

（9）Close 函数结束当前文件查找，结束后可直接调用 FindFile 函数开始下一次查找，无须创建新的 CFileFind 对象，格式如下：

```
void CFileFind::Close()
```

【实例 11-6】新建一个对话框工程名为 File083，在指定目录下查找指定后缀名的文件，在树控件中显示该目录下的所有匹配文件名，以及文件大小。

（1）新建对话框工程 File083，在对话框模板上，拖放静态文本、编辑框、按钮、树控件，如图 11-12 所示。

图 11-12　文件查找对话框

（2）设置静态控件的 Caption 为“文件类型”，设置按钮的 ID 为 IDC_BUTTON_ BROWSE，Caption 为“浏览目录”。在树控件的 Styles 选项卡里，勾选 Has buttons、Has lines、Lines at root 复选框。

（3）按 Ctrl+W 组合键打开类向导，选择 Member Variable 选项卡，双击 IDC_TREE1 项，添加 CTreeCtrl 类型的变量 m_tree。

（4）在类视图双击 CFile083Dlg 项，在类定义中添加两个成员变量，代码如下：

```
public:
    HICON m_hFileIcon;          //指定后缀名文件的关联图标的句柄
    CImageList m_img;           //图像列表
```

（5）在类视图双击 CFile083Dlg 类下的 OnInitDialog 项，在 return TRUE; 前添加如下代码：

```
m_hFileIcon=NULL;                                          //图标句柄初始为空
m_img.Create(16,16,ILC_COLOR16|ILC_MASK,2,1);              //创建图像列表，16*16
m_img.Add(AfxGetApp()->LoadStandardIcon(IDI_APPLICATION));//加载系统图标
m_tree.SetImageList(&m_img,TVSIL_NORMAL);                  //设置树控件使用的图像列表
```

（6）在资源视图，双击对话框模板上的“浏览目录”按钮，添加如下代码：

```
void CFile083Dlg::OnButtonBrowse()
{
    CString strFileExt;
    GetDlgItem(IDC_EDIT1)->GetWindowText(strFileExt);      //获取输入的后缀名，如 dll
    strFileExt.MakeLower();                                //后缀名小写
```

```
    if(strFileExt.IsEmpty())                                    //若输入为空，返回
        return;
    BROWSEINFO bInfo;                                           //浏览信息结构体
    TCHAR name[MAX_PATH];
    ZeroMemory(&bInfo,sizeof(BROWSEINFO));                      //清空结构体
    bInfo.hwndOwner=GetSafeHwnd();                              //父窗口句柄
    bInfo.pszDisplayName=name;                                  //接收选择的目录名的变量
    bInfo.lpszTitle="选择目录";                                  //提示标题
    bInfo.ulFlags=BIF_EDITBOX;                                  //窗口标志，有编辑框
    LPITEMIDLIST pItemList=SHBrowseForFolder(&bInfo);           //显示“浏览文件夹”窗口
    if(pItemList==NULL)                                         //若单击“取消”按钮，则返回 NULL
        return;
    CString strDir;
    SHGetPathFromIDList(pItemList,strDir.GetBuffer(MAX_PATH));//获取选择的目录路径
    strDir.ReleaseBuffer();                                     //释放字符缓冲区
    SHFILEINFO fileInfo;                                        //文件信息结构体
    SHGetFileInfo("."+strFileExt,0,&fileInfo,sizeof(fileInfo),//获取文件关联的图标
        SHGFI_ICON|SHGFI_SMALLICON|SHGFI_USEFILEATTRIBUTES);
    if(m_hFileIcon)                                             //若图标句柄变量不为空
    {
        if(m_img.GetImageCount()>1)                             //若图像列表有两个图标
            m_img.Remove(1);                                    //移除第 2 个图标
        DestroyIcon(m_hFileIcon);                               //释放先前获取的图标资源
        m_hFileIcon=NULL;                                       //设置图标句柄为空
    }
    m_hFileIcon=fileInfo.hIcon;                                 //获取新图标的句柄
    m_img.Add(m_hFileIcon);                                     //将新图标添加到图像列表

    m_tree.DeleteAllItems();                                    //清空树控件
    HTREEITEM hRoot=m_tree.InsertItem(strDir,0,0);              //将选择目录作为根节点插入
    CFileFind f;                                                //文件查找对象
    BOOL bFind=f.FindFile(strDir+"\\*."+strFileExt);            //查找选择目录下的所有匹配文件
    while(bFind)                                                //循环遍历匹配文件
    {
        bFind=f.FindNextFile();                                 //查找下一个匹配文件
        if(f.IsDirectory())                                     //若为目录，则直接进入下一次循环
            continue;
        CString strFileSize;
        strFileSize.Format("%d kb",f.GetLength()/1024);   //获取文件大小
        m_tree.InsertItem(f.GetFileTitle()+"  "+strFileSize,1,1,hRoot);
                                                                //将文件名和大小作为子节点插入
    }
    f.Close();                                                  //停止查找
    m_tree.Expand(hRoot,TVE_EXPAND);                            //展开根节点
}
```

GetWindowText 函数获取编辑框输入的后缀名，MakeLower 函数将输入字符串变为小写，文件名一般不区分大小写，编程比较时，应统一转换为小写。

BROWSEINFO 结构体包含“浏览文件夹”窗口所需要的信息，格式如下：

```
typedef struct _browseinfo {
    HWND hwndOwner;                 //父窗口句柄
    LPCITEMIDLIST pidlRoot;         //起始目录，若为 NULL，起始目录为桌面
    LPTSTR pszDisplayName;          //用户选择的目录名
    LPCTSTR lpszTitle;              //提示文本
    UINT ulFlags;                   //窗口标志
    BFFCALLBACK lpfn;               //回调函数
    LPARAM lParam;                  //传递给回调函数的参数
    int iImage;                     //选择目录的图像索引
} BROWSEINFO, *PBROWSEINFO, *LPBROWSEINFO;
```

ZeroMemory 函数设置指定长度的内存值为 0，GetSafeHwnd 函数获取当前窗口的句柄，窗口标志 BIF_EDITBOX 表示显示编辑框。SHBrowseForFolder 函数显示“浏览文件夹”窗口，用于选择一个目录，格式如下：

```
LPITEMIDLIST SHBrowseForFolder(LPBROWSEINFO lpbi)
```

参数如下。

- lpbi：指向 BROWSEINFO 结构体的指针。

返回值：指向 ITEMIDLIST 结构体的指针，存放选择目录的位置，若没有选择目录，则返回 NULL。

SHGetPathFromIDList 函数将 ITEMIDLIST 结构体转换为文件路径，格式如下：

```
BOOL SHGetPathFromIDList(LPCITEMIDLIST pidl,LPTSTR pszPath)
```

参数如下。

- pidl：指向 ITEMIDLIST 结构体的指针。
- pszPath：字符缓冲区的指针，存放文件路径，长度至少为 MAX_PATH。

返回值：若成功则返回 TRUE，否则返回 FALSE。

SHFILEINFO 结构包含某种类型文件的信息，格式如下：

```
typedef struct _SHFILEINFO {
    HICON hIcon;                          //文件关联的图标的句柄
    int iIcon;                            //图标在系统图像列表中的索引
    DWORD dwAttributes;                   //文件对象的属性信息
    TCHAR szDisplayName[MAX_PATH];        //文件显示的名称
    TCHAR szTypeName[80];                 //文件类型描述
} SHFILEINFO;
```

SHGetFileInfo 函数获取文件系统的对象信息，如文件、目录、驱动器，格式如下：

```
DWORD_PTR SHGetFileInfo(LPCTSTR pszPath,DWORD dwFileAttributes,SHFILEINFO *psfi,
UINT cbFileInfo,UINT uFlags)
```

参数如下。

- pszPath：文件类型
- dwFileAttributes：文件属性标志
- psfi：指向 SHFILEINFO 结构体的指针，接收获取的信息。
- cbFileInfo：SHFILEINFO 结构体的大小。
- uFlags：获取信息的标志。

返回值：uFlags 参数指定的值。

参数 1 指定文件类型，如“*.dll”，参数 3 指向接收信息的结构体，参数 5 表示要获取图标信息，调用该函数后，SHFILEINFO 结构体的 hIcon 成员存放文件关联图标的句柄值。

通过 SHGetFileInfo 函数获取的图标句柄，在使用结束后，需要调用 DestroyIcon 函数释放该图标资源，m_hFileIcon 存放获取的图标句柄，并在获取下一个图标前，先释放该图标。GetImageCount 函数获取图像列表的图像数目，Remove 函数移除指定位置的图像，Add 函数将新图标添加到图像列表中。

FindFile 函数开始查找指定目录下的匹配文件，如 D:*.dll 匹配 D 盘根目录下的所有 DLL 文件，若查找到文件，则返回 TRUE，存放到 bFind 中。利用 while 循环遍历所有匹配文件，FindNextFile 函数获取下一个匹配文件，若为最后一个文件，则返回 FALSE。

IsDirectory 函数判断是否为目录，若为目录，则直接进入下一次循环。GetLength 函数获取文件长度，GetFileTitle 函数获取文件标题，作为子节点插入到树控件中，并使用获取的文件关联图标。Close 函数关闭文件查找，Expand 函数展开根节点。

（7）在类视图右击 CFile083Dlg 项，选择 Add Windows Message Handler 命令，添加 WM_DESTROY 消息的处理函数，添加如下代码：

```
void CFile083Dlg::OnDestroy()
{
    CDialog::OnDestroy();
    // TODO: Add your message handler code here
    if(m_hFileIcon)                          //若图标句柄不为空
    {
        m_img.DeleteImageList();             //清空图像列表
        DestroyIcon(m_hFileIcon);            //释放图标资源
    }
}
```

当程序关闭时，自动调用该函数。若图标句柄不为空，则 DeleteImageList 函数清空图像列表，DestroyIcon 函数释放获取的图标句柄资源。

（8）生成程序并运行，如图 11-13 所示。在“文件类型”编辑框中输入后缀名，如 dll，单击“浏览目录”按钮，弹出“浏览文件夹”窗口，如选择 C:\WINDOWS\system32，如图 11-14 所示。单击“确定”按钮后，树控件显示该目录下的所有 DLL 文件，并显示 DLL 文件的图标，以及文件大小。

图 11-13　文件查找

图 11-14　选择目录

11.3 文件序列化

序列化是一种特殊的文件存取技术，在 CFile 类的功能基础上，将存取的元素看做一个个对象，要存储的数据看做数据流，如有对象 a、b、c，存储时按照 a、b、c 的顺序放入数据流中，一次性存入硬盘文件中，读取时按照同样的顺序，将读取的数据依次存入 a、b、c 中。

序列化技术的关键是创建可序列化的类，任何从 CObject 派生的类都可重写基类的 Serialize 函数，在函数内部实现数据成员的序列化存储。使用序列化技术，无须关心数据的实际字节长度，只需用同样的元素顺序执行存取操作，在实际开发中，根据需要决定采用序列化技术还是 CFile 类直接读取。

11.3.1 如何实现序列化

MFC 提供 CArchive 类用于实现序列化技术，该类在 CFile 类的基础上，实现数据的序列化存储，以简化文件的存取操作。CArchive 类不仅可以处理基本数据类型，还可处理支持序列化的类，CObject 类提供的基本服务包括支持序列化，可创建一个 CObject 派生类，并重写 Serialize 函数，实现类的序列化。

若派生类重写 Serialize 函数，必须在函数内部先调用基类版本的 Serialize 函数，同时在类声明中添加 DECLARE_SERIAL 宏，在类实现文件中添加 IMPLEMENT_SERIAL 宏，用于添加序列化支持。

CArchive 类提供一系列函数用于实现序列化技术，常用函数如下所示。

（1）CArchive 构造函数用于构造一个类对象，并指定操作的文件对象，格式如下：

```
CArchive::CArchive(CFile* pFile,UINT nMode,int nBufSize=4096,void* lpBuf=NULL )
```

参数如下。

- pFile：要操作文件的 CFile 类指针。
- nMode：操作标识符，如 CArchive::load 表示读取数据，CArchive::store 表示存储数据。
- nBufSize：接收数据的缓冲区大小，默认为 4096 字节。
- lpBuf：指向缓冲区的指针，若不指定，则自动分配和释放一个缓冲区。

（2）Close 函数将数据流中的数据传输到文件中，并关闭与文件对象的连接，格式如下：

```
void CArchive::Close()
```

（3）operator >>用于从文件中读取数据，存入指定变量中，格式如下：

```
CArchive& CArchive::operator >>(int& i)
friend CArchive& operator >>(CArchive& ar, CObject* pOb)
```

参数如下。

- i：接收数据的整型变量。
- ar：CArchive 对象引用。
- pOb：CObject 类及派生类对象的指针，接收数据。

返回值：可连续读取的 CArchive 对象引用。

Tips 通过重载>>运算符，读取指定长度的数据，如 ar>>i；读取 4 字节长度的值，存入 i 中，也可一次读入一个类对象，读取长度为类对象的总大小。除了 int 类型，operator >> 还支持 BYTE、WORD、LONG、DWORD、float、double 类型。

（4）operator <<用于将数据存入文件，格式如下：

```
friend CArchive& operator << (CArchive& ar, const CObject* pOb)
CArchive& CArchive::operator <<(int i)
```

参数如下。

- ar：CArchive 对象引用。
- pOb：CObject 类及派生类对象的指针，存储数据。
- i：要存储的数据。

返回值：可连续存储的 CArchive 对象引用。

（5）IsStoring 函数判断当前操作是否为存储，格式如下：

```
BOOL CArchive::IsStorng() const
```

返回值：若正在存储则返回非零值，否则返回 0。

（6）IsLoading 函数判断当前操作是否为读取，格式如下：

```
BOOL CArchive::IsLoading() const
```

返回值：若正在读取则返回非零值，否则返回 0。

11.3.2 创建可序列化类

创建一个可序列化类，步骤如下：

（1）新建一个 CObject 派生类。

（2）添加 DECLARE_SERIAL 和 IMPLEMENT_SERIAL 宏。

（3）重写 Serialize 虚函数。

（4）添加类成员函数。

【实例 11-7】新建一个单文档工程名为 File084，创建一个可序列化类 CMyCircle，在文档类中使用序列化技术存取类对象，在视图类中显示、编辑数据。

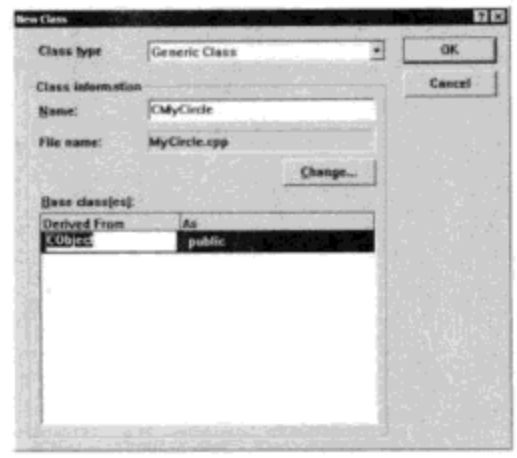


图 11-15　添加类

（1）新建单文档工程 File084，选择 Insert|New Class 命令，弹出 New Class 窗口，如图 11-15 所示。

（2）Class type 组合框选择 Generic Class 项，Name 编辑框输入 CMyCircle，Base class 编辑框输入 CObject，单击 OK 按钮，新建一个 CObject 派生类。

（3）在类视图双击 CMyCircle 项，修改类头文件，完整代码如下：

```
class CMyCircle : public CObject                    //基类为 CObject
{
    DECLARE_SERIAL(CMyCircle)                       //序列化声明宏
public:
    void Serialize(CArchive& myAr);                 //重写 Serialize 序列化函数
    CPoint m_ptCenter;                              //圆心坐标
    int m_nRadius;                                  //圆半径值
    COLORREF m_clr;                                 //圆弧颜色
    CMyCircle();                                    //默认构造函数
    CMyCircle(CPoint ptCenter);                     //重载的构造函数
    virtual ~CMyCircle();                           //虚析构函数
    void DrawShape(CDC* pDC);                       //绘圆函数
};
```

DECLARE_SERIAL 宏为要序列化的类生成必要的代码，参数为类名。序列化类必须有一个无参构造函数，从文件中读取数据时，可通过无参构造函数创建类对象，再调用 Serialize 函数初始化类成员。

手动添加的 Serialize 函数是个虚函数，在基类 CObject 类中有定义，但类向导不能为自定义类添加虚函数，需要手动添加，实际上和添加普通成员函数没有区别，只是函数原型要保持一致。

（4）双击 CMyCircle 类下的一项，修改类实现文件，完整代码如下：

```
IMPLEMENT_SERIAL(CMyCircle,CObject,1)                       //序列化实现宏
CMyCircle::CMyCircle()
{
}
CMyCircle::~CMyCircle()
{
}
CMyCircle::CMyCircle(CPoint ptCenter)                       //重载的构造函数，传入圆心坐标
{
    srand(unsigned(time(NULL)));                            //当前时间作为随机数种子
    m_ptCenter=ptCenter;                                    //设置圆心坐标
    m_clr=RGB(rand()%255,rand()%255,rand()%255);            //设置随机颜色值
    m_nRadius=rand()%50;                                    //设置随机半径值
}
void CMyCircle::DrawShape(CDC* pDC)                         //绘圆函数
{
    CPen pen(PS_SOLID,2,m_clr);                             //创建画笔对象，实线、宽度 2
    CPen* pOldPen=pDC->SelectObject(&pen);                  //新画笔选入 DC
    CBrush* pOldBrush=(CBrush*)pDC->SelectStockObject(NULL_BRUSH);     //选入空画刷
    pDC->Ellipse(m_ptCenter.x-m_nRadius,m_ptCenter.y-m_nRadius,        //绘制圆
        m_ptCenter.x+m_nRadius,m_ptCenter.y+m_nRadius);
```

```
    pDC->SelectObject(pOldPen);                          //恢复原有画笔、画刷
    pDC->SelectObject(pOldBrush);
}
void CMyCircle::Serialize(CArchive &myAr)                //重写的序列化函数
{
    CObject::Serialize(myAr);                            //调用基类版本的函数
    if(myAr.IsStoring())                                 //若正在存储
    {
        myAr<<m_ptCenter.x<<m_ptCenter.y;                //存入圆心的 x、y 坐标
        myAr<<m_nRadius;                                 //存入半径值
        myAr<<m_clr;                                     //存入颜色值
    }
    else                                                 //若正在读取
    {
        myAr>>m_ptCenter.x>>m_ptCenter.y;                //获取圆心的 x、y 值
        myAr>>m_nRadius;                                 //获取半径值
        myAr>>m_clr;                                     //获取颜色值
    }
}
```

IMPLEMENT_SERIAL 宏和 DECLARE_SERIAL 宏要联合使用，为派生类生成必要的代码，参数 1 为类名，参数 2 为基类名，参数 3 为版本号，表明当前存储数据的版本。

CMyCircle(CPoint ptCenter)函数是重载的构造函数，参数为圆心坐标，在该构造函数中，设置圆心坐标、圆半径、圆弧颜色的值，其中半径、圆弧颜色是随机值，使用当前时间作为随机数种子。

DrawShape 函数用于绘制圆，参数为设备环境的指针，使用实线、宽度为 2 的画笔绘制圆的边界线，使用空画刷填充圆，相当于不填充。绘制结束后，使用 SelectObject 函数恢复原有画笔、画刷。

Serialize 函数为重写的序列化函数，用于序列化类成员，参数为 CArchive 类对象的引用。先调用基类版本的 Serialize 函数，IsStoring 函数判断当前是否正在存储数据，若正在存储，则通过<<运算符将类成员数据存入 myAr 所关联的文件中，若正在读取，则通过>>获取文件中的数据，按照同样的顺序，存入类成员中。

11.3.3 序列化对象

（1）在类视图右键单击 CFile084Doc 项，选择 Add Member Variable 命令，添加一个 CObArray 类型的变量 m_arrayCircle。

（2）在类视图右键单击 CFile084View 项，选择 Add Windows Message Handler 命令，添加 WM_LBUTTONDOWN 消息的处理函数，添加如下代码：

```
void CFile084View::OnLButtonDown(UINT nFlags, CPoint point)
{
    CView::OnLButtonDown(nFlags, point);
    CFile084Doc* pDoc=GetDocument();          //获取文档类对象指针
    CMyCircle* pNewCircle=new CMyCircle(point);
                                              //根据当前坐标，动态创建一个 CMyCircle 类对象
    pDoc->m_arrayCircle.Add(pNewCircle); //将类对象指针添加到数组中
    Invalidate();                             //强制视图重绘
}
```

CObArray 类用于存储 CObject 类及派生类指针的可变数组，能存放任何 CObject 派生类的指针，可在程序运行时动态添加和删除元素，常用函数如下。

- GetSize：获取元素数目。
- SetSize：设置初始元素数目。
- RemoveAll：移除所有元素。
- GetAt：获取指定索引的 CObject 指针。

- SetAtGrow：设置指定索引的元素值，且可自动增长。
- Add：添加新元素。

GetDocument 函数获取文档类的指针，使用 new 动态创建一个 CMyCircle 类对象，调用有参构造函数，参数为当前鼠标坐标。Add 函数将动态创建对象的指针添加到指针数组中。

（3）在当前 CPP 文件的开头处，添加一句#include "MyCircle.h" ，包括 CMyCircle 类的头文件。

（4）双击 CFile084View 类下的 OnDraw 项，添加如下代码：

```
void CFile084View::OnDraw(CDC* pDC)
{
    CFile084Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    int nSize=pDoc->m_arrayCircle.GetSize(); //获取指针数组大小
    for(int i=0;i<nSize;i++)                 //遍历数组
    {
        CMyCircle* pCircle=(CMyCircle*)pDoc->m_arrayCircle.GetAt(i);
                                             //获取第 i 个指针
        pCircle->DrawShape(pDC);             //根据对象成员值，绘圆
    }
}
```

GetSize 函数获取指针数组的大小，利用 for 循环遍历所有元素，GetAt 函数获取第 i 个元素的指针值，并转为 CMyCircle 类指针，调用 DrawShape 函数在视图 DC 上绘圆。

（5）在类视图右键单击 CFile084Doc 项，选择 Add Member Function 命令，添加返回类型为 void 的函数 ReleaseCircles()，添加如下代码：

```
void CFile084Doc::ReleaseCircles()
{
    for(int i=0;i<m_arrayCircle.GetSize();i++)
    {
        CMyCircle* pCircle=(CMyCircle*)m_arrayCircle.GetAt(i);    //获取第 i 个指针
        delete pCircle;                                          //释放动态创建的类对象
    }
    m_arrayCircle.RemoveAll();                                   //清空指针数组
}
```

指针数组中存放的类对象，是使用 new 动态创建的，在必要的时候需要手动调用 delete，释放这些对象，以免造成内存泄漏，RemoveAll 函数清空指针数组。

（6）在当前 cpp 文件的开头处，添加一句#include "MyCircle.h" ，包括 CMyCircle 类的头文件。

（7）双击 CFile084Doc 类下的~CFile084Doc 项，添加如下代码：

```
CFile084Doc::~CFile084Doc()                  //文档类析构函数
{
    ReleaseCircles();                        //释放动态创建的类对象
}
```

当程序关闭时，自动调用文档类的析构函数，可在该函数中释放动态创建的类对象，也在其他函数中完成释放功能，只要程序关闭时，能自动调用该函数就行。

（8）双击 CFile084Doc 类下的 OnNewDocument 项，添加如下代码：

```
BOOL CFile084Doc::OnNewDocument()            //新建文档函数
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    ReleaseCircles();                        //释放已创建的对象
    return TRUE;
}
```

单击“新建”菜单或工具按钮时，表示要创建一个新文档，需要清空已有数据，可在该函数中释放已创建的对象。

（9）双击 CFile084Doc 类下的 Serialize 项，添加如下代码：

```
void CFile084Doc::Serialize(CArchive& ar)//序列化
{
    m_arrayCircle.Serialize(ar);          //指针数组整体序列化
}
```

CObArray 类可自动实现所有元素的序列化，只要元素对应的类支持序列化。存储时，将每个元素所在的类对象的值存入文件中，读取时，根据文件中包含的类对象数目，动态创建类对象，存放读取出来的数据。CObArray 类的 Serialize 函数内部流程如下：

```
void CObArray::Serialize(CArchive& ar)
{
    ASSERT_VALID(this);                   //测试指针是否为空
    CObject::Serialize(ar);               //调用基类版本函数
    if (ar.IsStoring())                   //若正在存储
    {
        ar.WriteCount(m_nSize);           //写入元素数目
        for (int i = 0; i < m_nSize; i++)     //遍历所有元素，依次写入文件
            ar << m_pData[i];
    }
    else                                  //若正在读取
    {
        DWORD nOldSize = ar.ReadCount();      //读取元素数目
        SetSize(nOldSize);                    //设置数组大小，动态创建类对象
        for (int i = 0; i < m_nSize; i++)
            ar >> m_pData[i];                 //读取文件数据，依次存入对象
    }
}
```

Tips CObArray 类的 Serialize 函数、CArchive 类的 WriteCount 和 ReadCount 函数，在 msdn 帮助文档中并没有列出，可见帮助文档并非包罗一切资料，阅读 MFC 源代码是有必要也很有用的。

（10）按 Ctrl+W 组合键打开类向导，选择 Message Maps 选项卡，为 ID_FILE_OPEN、ID_FILE_SAVE 菜单项，添加 CFile084Doc 类的 COMMAND 消息处理函数，添加如下代码：

```
void CFile084Doc::OnFileSave()                          //“保存”菜单项
{
    CString strFilter="cir 图形文件(*.cir)|*.cir||";    //文件过滤字符串
    CFileDialog  fileDlg(FALSE,"cir"," 未 命 名 图 像 ",OFN_ENABLESIZING,(LPCTSTR)
strFilter, NULL);
    if(fileDlg.DoModal()==IDOK)                         //显示保存文件对话框
    {
        CString strPath=fileDlg.GetPathName();    //获取文件路径
        CFile f;
        if(f.Open(strPath,CFile::modeWrite|CFile::modeCreate)) //创建并写入新文件
        {
            CArchive ar(&f,CArchive::store); //创建 CArchive 类对象，并关联到文件对象
            ar.m_pDocument=this;             //设置序列化的文档对象
            TRY                              //异常处理
            {
                CWaitCursor wait;            //等待光标
                Serialize(ar);               //调用文档类的序列化函数
                ar.Close();                  //写入文件，并断开连接
```

```
                f.Close();                                  //关闭文件
            }
            CATCH (CMemoryException, e)                     //若有内存异常
            {
                ar.Close();                                 //断开连接
                AfxMessageBox("文件存储异常");               //弹出信息提示框
            }
            END_CATCH                                       //结束异常处理
        }
    }
}

void CFile084Doc::OnFileOpen()                              //“打开”菜单项
{
    CString strFilter="cir 图形文件(*.cir)|*.cir||";    //文件过滤字符串
    CFileDialog fileDlg(TRUE,"cir",NULL,OFN_ENABLESIZING,(LPCTSTR)strFilter,NULL);
    if(fileDlg.DoModal()==IDOK)                                 //显示打开文件对话框
    {
        CString strPath=fileDlg.GetPathName();              //获取文件路径
        CFile f;
        if(f.Open(strPath,CFile::modeRead))                 //以只读方式打开文件
        {
            CArchive ar(&f,CArchive::load);                 //创建 CArchive 类对象，读取文件
            ar.m_pDocument=this;                            //设置序列化文档对象
            TRY
            {
                CWaitCursor wait;
                Serialize(ar);                              //调用文档类的序列化函数
                ar.Close();
                f.Close();
                CFrameWnd* pWnd=(CFrameWnd*)AfxGetMainWnd();  //获取主窗口指针
                pWnd->GetActiveView()->Invalidate();          //强制视图重绘
            }
            CATCH (CMemoryException, e)
            {
                ar.Close();
                AfxMessageBox("文件打开异常");
            }
            END_CATCH
        }
    }
}
```

strFilter 为文件过滤字符串，用来存储和读取 CIR 格式的文件，CIR 是自定义的文件后缀名。在文件对话框中设置文件路径后，Open 函数以写入或读取形式打开该文件，创建一个 CArchive 类对象，并关联到文件对象，根据当前操作类型，设置 CArchive 类对象的状态，若要保存文件，则使用 CArchive::store 标志，若要打开文件，则使用 CArchive::load 标志。

m_pDocument 属性用来设置序列化的文档对象，TRY、CATCH、END_CATCH 宏用来进行异常处理，TRY 块中为正常代码，CATCH 块中为异常处理代码，END_CATCH 结束异常处理。

CWaitCursor 类用来启用鼠标等待光标，当类对象超出作用域时，自动恢复原始光标。调用文档类的 Serialize 函数，序列化对象，根据操作类型，写入或读取文件内容。序列化结束后，Close 函数先将缓冲区的内容写入文件，并断开与文件对象的连接，再关闭文件对象。

若为打开文件，获取文件数据并存入文档类的成员变量后，AfxGetMainWnd 函数获取主窗口的指针，GetActiveView 函数获取活动视图的指针，Invalidate 函数强制视图重绘。

Tips 若不为“打开”、“保存”菜单项添加消息处理函数，这两个菜单项执行系统默认流程，也能弹出文件对话框，保存和打开文件，但保存的文件没有后缀名。通过添加消息处理函数，手动调用文件对话框，设置文件过滤字符串，可以为文件添加后缀名。也可在创建单文档程序时，在 AppWizard 的第 4 步，单击 Advanced 按钮，弹出 Advanced Options 窗口，如图 11-16 所示，在 File extension 编辑框输入文件后缀名，如 cir，创建完成后，单击“保存”菜单时，在文件对话框中文件有后缀名 cir，这种方式将文件信息写入注册表，保存文件时读取注册表中的信息，获取文件后缀名。可根据自身需要，采用其中一种方式。

图 11-16 设置程序文件的后缀名

（11）生成程序并运行，如图 11-17 所示。用鼠标在视图窗口单击，每单击一次，绘制一个圆，圆心为鼠标坐标，圆半径和颜色是随机值，单击“保存”菜单或按钮，弹出“另存为”对话框，存储为 CIR 格式的二进制文件，如图 11-18 所示。关闭再运行程序，单击“打开”菜单或按钮，弹出“打开”对话框，选择存储的 CIR 文件，视图窗口重新显示存储的图形。

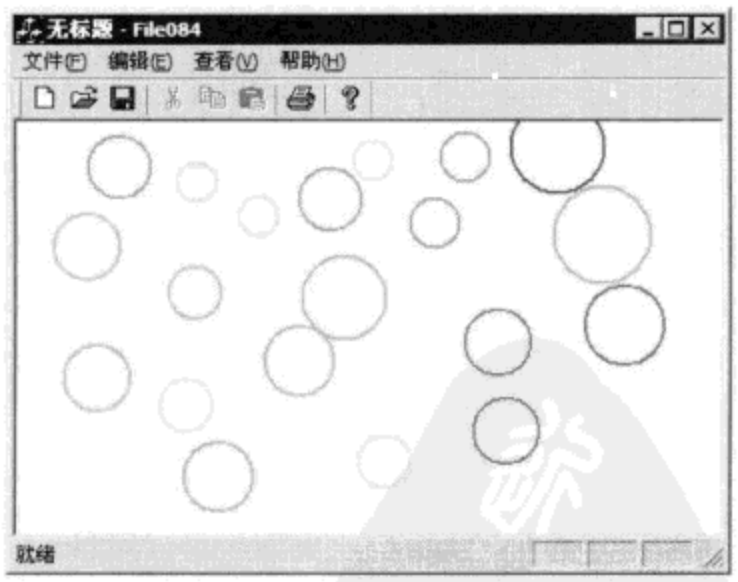


图 11-17 序列化对象

图 11-18 保存文件

11.4 小结

对于日常办公，最常用的就是文件操作。在 Visual C++中，通过 MFC 提供的 CFile 及其他几个类，可以操作文件，如打开、读取、写入、重命名、删除文件等。但若对文件进一步操作，

则需要调用 API 函数。本章主要介绍了文件类、文件查找和文件的序列化。对于文件编程，需要注意对于异常的处理。

11.5 习题

1. 如何理解文件与流？
2. 如何读写一个顺序文件？
3. 如何使用 Cfile 类来实现文件的读写？
4. 如何实现文件的序列化？
5. 编写一个程序，用于合并两个顺序文件的内容，并将合并结果保存在第一个文件中。

第 12 章　数据库编程

数据库是信息时代的核心工具，所有信息都需要存放到数据库中，以便读取和更新，如学校的学生学籍信息、网站的个人注册信息、论坛上网友发表的帖子信息等。数据库是软件的基础，数据库的设计合理与否直接关系到软件项目的开发进度，在实际项目开发中，数据库的设计是很重要的一个步骤，甚至占据大半的工作时间，数据库表结构确定后，一般情况下不能改变，若表结构发生改变，则程序通常需要做大量修改，甚至要修改软件界面，这会严重影响软件的开发进度。

12.1 了解数据库

使用数据库的程序通常分为两个部分，一个是数据库服务器，在服务器上安装数据库平台软件，存放程序所需的数据，另一个是客户端程序，是用户操作的主程序，通过网络连接服务器，可发送命令到数据库平台软件，执行数据读取或更新操作。

数据库服务器和客户端程序通常部署在不同的机器上，通过网络保持连接，若两个都安装在一台机器上，则该机器既是服务器也是客户端。常用的数据库平台软件有 Microsoft 的 SQL Server、Access、Excel，Oracle、MySQL、Sqlite 等，其中大型应用以 SQL Server、Oracle 为主，Access 可存储小规模数据，Excel 可用于前期的数据输入，完成后直接导入其他数据库，MySQL 可用于 Linux 平台，Sqlite 可用于嵌入式系统。在实际开发中，根据需要采用其中一种数据库平台。

12.1.1 安装 SQL Server 2000

Microsoft 公司在 2000 年发布 SQL Server 2000 软件，用于管理关系数据库，具有图形化的操作界面，以及强大的数据管理功能，是企业级项目开发中常用的数据库平台软件。

SQL Servers 2000 有多个版本，其中企业版（Enterprise Edition）只能安装在 Windows Server 2000 及以上的服务器操作系统中，若使用 Windows XP 操作系统，可安装个人版（Person Edition）或评估版（Evaluation Edition）。下面以评估版为例，安装 SQL Server 2000 软件。

（1）双击安装目录下的 autorun.exe，弹出安装界面，如图 12-1 所示。

（2）选择“安装 SQL Server 2000 组件”选项，弹出新窗口，如图 12-2 所示。

图 12-1　启动安装界面

图 12-2　安装组件

（3）选择“安装数据库服务器”选项，弹出一系列对话框，均保持默认选择，直到弹出“安装类型”对话框，如图 12-3 所示。

（4）根据需要选择“典型”、“最小”或“自定义”单选按钮，一般情况下保持默认选择。单击“浏览”按钮，可设置程序和自带数据库的存放路径，若 C 盘空间有限，可改存到 D 盘中。单击“下一步”按钮，弹出“服务账户”对话框，如图 12-4 所示。

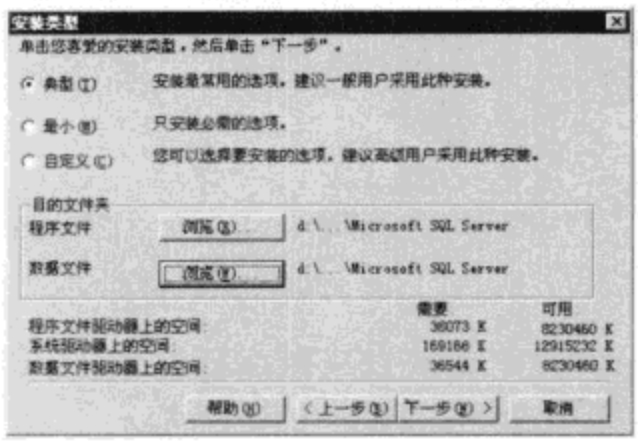


图 12-3　安装类型

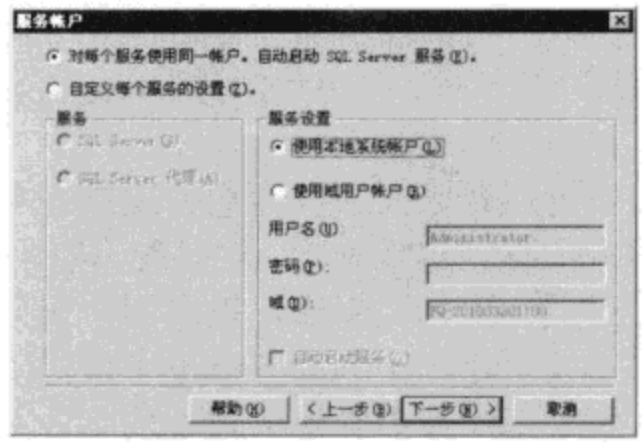


图 12-4　服务账户

（5）选择“使用本地系统账户”选项，单击“下一步”按钮，弹出“身份验证模式”对话框，如图 12-5 所示。

（6）选择“混合模式”单选按钮，在“输入密码”文本框输入密码，如 sa，并在“确认密码”文本框中输入相同的密码。单击“下一步”按钮，在新对话框中再单击“下一步”按钮，开始复制文件，如图 12-6 所示，安装完成后，自动关闭该窗口。

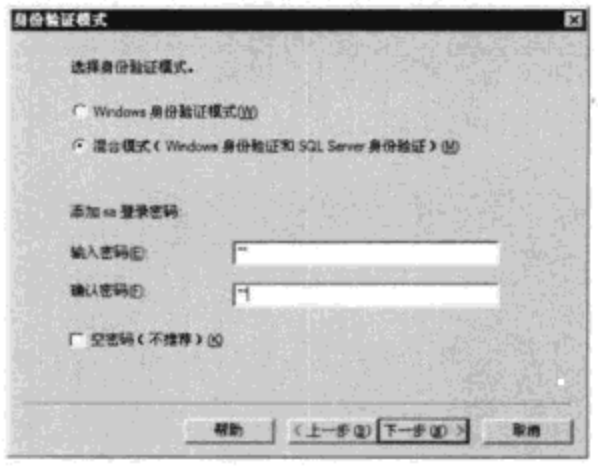


图 12-5　身份验证模式

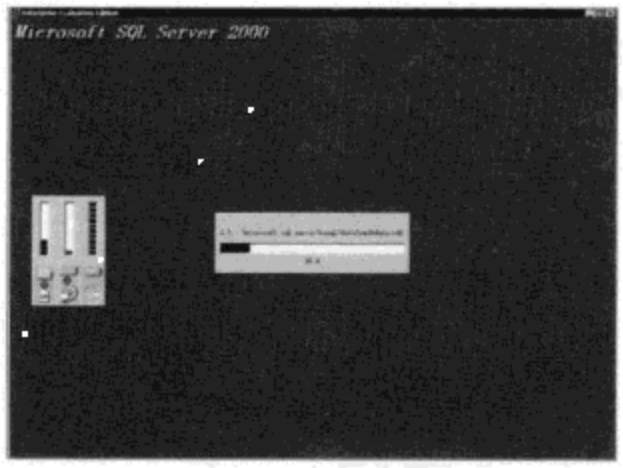


图 12-6　复制文件

12.1.2　企业管理器

打开企业管理器的具体步骤如下。

（1）选择“开始”|“程序”|“Microsoft SQL Server”命令，弹出已安装的项，如图 12-7 所示。

（2）选择“服务管理器”命令，弹出“SQL Servers 服务管理器”窗口，如图 12-8 所示。

图 12-7　软件菜单

图 12-8　服务管理器

“服务器”组合框显示可用的数据库服务器，一般为主机名，“服务”组合框显示可用的数

据服务。“开始/继续”按钮用于启动数据库，“暂停”按钮暂停数据库服务，“停止”按钮关闭数据库服务。“当启动 OS 时自动启动服务”若勾选，则每次启动电脑时，自动启动数据库服务，若不需要可取消勾选该项，以节省系统资源。

图 12-9　企业管理器

（3）单击“开始/继续”按钮，启动服务后，选择“企业管理器”菜单项，弹出“SQL Server Enterprise Manager”窗口，如图 12-9 所示。

企业管理器以图形界面的方式管理 SQL Server 软件，常用的操作有创建数据库、数据表、视图，导入导出数据，备份、还原、附加数据库，设置用户权限等。

本节以创建一个名为 Person 的数据库为例，介绍如何使用企业管理器创建数据库，添加数据表，该数据库在后续章节将要用到，关于企业管理器的更多知识请阅读相关书籍。

（4）右键单击“数据库”项，在弹出的快捷菜单中选择“新建数据库”命令，弹出“数据库属性”对话框，如图 12-10 所示。

（5）“名称”编辑框输入 Person，单击“确定”按钮，“数据库”节点下出现 Person 项。展开 Person 的子节点，右键单击“表”，在弹出的快捷菜单中选择“新建表”命令，弹出表设计窗口，如图 12-11 所示。

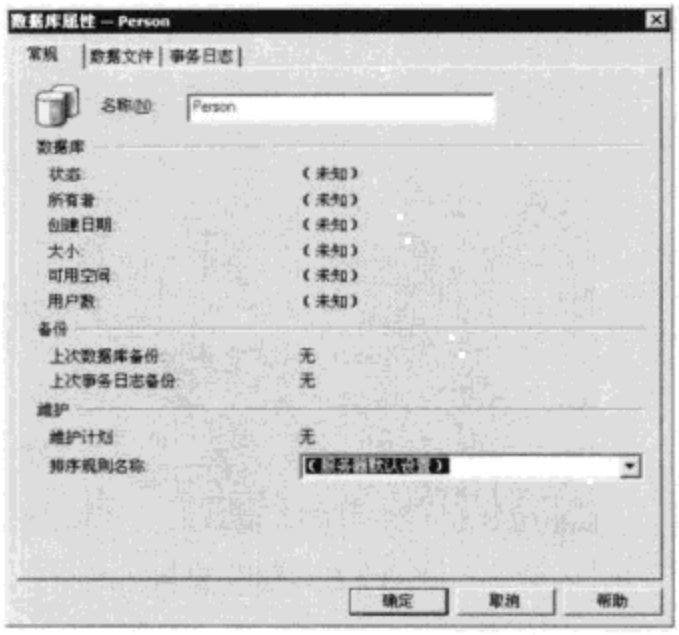


图 12-10　新建数据库

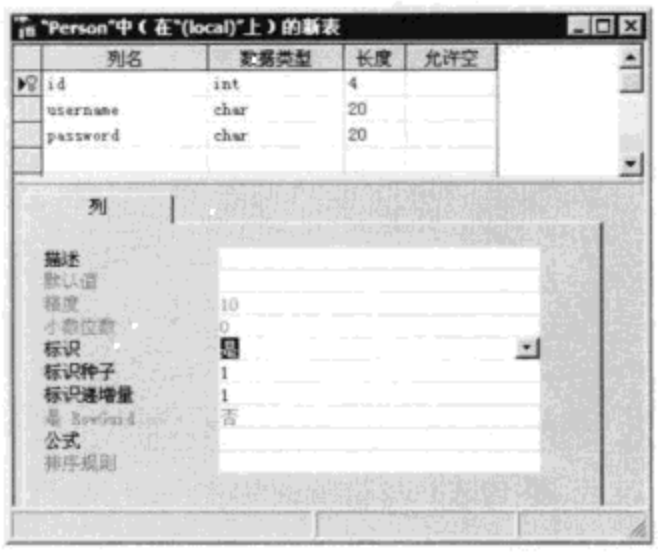


图 12-11　loginUser 表

“列名”设置字段名称，“数据类型”设置字段的类型，“长度”设置字段的长度，“允许空”设置字段是否允许为空。下方的“列”选项卡用于设置字段的更多属性，如“描述”用于解释字段含义，“标识”用于字段的自动增长，选择“是”后，该字段的值会自动增长。

（6）表结构格式如表 12-1 所示。

表 12-1　loginUser表结构

列　名	数据类型	长　度	允 许 空
id	int	4	否
username	char	20	否
password	char	20	否

（7）选中 id 行，“标识”组合框选择“是”，并单击 按钮设置 id 字段为主键。表结构设计完成后，单击“关闭”按钮，弹出保存提示框，单击“是”按钮，输入表名 loginUser，单击“确

定”按钮，完成添加新表。

（8）用同样方式再添加一个新表 friends，如图 12-12 所示。同样设置 id 字段为“标识”，主键。

（9）friends 表结构格式如表 12-2 所示。

图 12-12　friends 表

表 12-2　friends表结构

列　名	数据类型	长　度	允许空
id	int	4	否
name	char	20	否
sex	char	2	是
birthday	smalldatetime	4	是
mobile	char	20	是
qq	char	15	是
email	char	30	是
address	varchar	100	是

若要修改表结构，右键单击表名，在弹出的快捷菜单中选择“设计表”命令，弹出设计表窗口。若要查看表中已有的数据，则右键单击表名，在弹出的快捷菜单中选择“打开表”|“返回所有行”命令，查看已有数据，也可输入数据。

12.1.3　查询分析器

打开查询分析器的具体步骤如下：

（1）选择“开始”|“程序”|“Microsoft SQL Server”|“查询分析器”命令，弹出“连接到 SQL Server”窗口，如图 12-13 所示。

（2）选择任意一种连接方式，若选择“SQL Server 身份验证”，需要输入登录名和密码，如 sa，单击“确定”按钮，打开“查询”窗口，如图 12-14 所示。

图 12-13　连接到 SQL Server

图 12-14　查询分析器

查询分析器用于执行 SQL 命令，以图形界面查看查询结果，如执行一条 SQL 查询语句，在

工具栏的组合框选择 Northwind 数据库，在“查询”窗口中输入 select * from orders，单击▶按钮或按 F5 键，执行 SQL 命令，下方显示 orders 表的所有数据。

12.1.4 数据查询语言

数据库平台软件使用结构化查询语言（Structured Query Language，简称 SQL）管理数据库，包括以下三种类别。

- 数据定义语言：用来创建数据库、数据表、视图、索引等对象。
- 数据操作语言：用来读取、更新表中的数据。
- 数据控制语言：用来实现数据库的高级控制，如事务、并发控制等。

其中数据定义可在企业管理器里，通过图形界面形式完成，在实际开发中，数据操作是最常用的 SQL 命令，客户端程序向数据库服务器发送 SQL 命令，数据库接收到 SQL 命令后，自动执行相关操作，如返回指定数据，插入、更新、删除指定记录等。

数据查询是最常用的 SQL 命令，在 SQL 语言中，使用 select 语句获取数据表中的数据，格式如下：

```
select [distinct] 字段1,字段2 from 表1,表2 [where 表达式1 and 表达式2] [order by 字段名 ASC]
```

其中[]为可选项，distinct 关键字表示去除重复项，如 sex 字段有男、女两种值，但若读取多条记录，就有多个男、女值，可使用 distinct 关键字，得到唯一的两个值。

使用 Northwind 数据库，在查询分析器中输入下面的 SQL 语句，得到 Orders 表中 EmployeeID 字段的唯一值，执行结果如图 12-15 所示。

```
select distinct EmployeeID from orders
```

select 语句可同时读取多个表中的多个字段，如表 Orders 有字段 OrderID，表 Order Details 也有字段 OrderID，两个表通过 OrderID 字段关联，若获取两个表中的关联数据，SQL 语句如下所示：

```
select Orders.OrderID,ShipName,UnitPrice from Orders,[Order Details]
  where [Order Details].OrderID=Orders.OrderID
```

执行结果如图 12-16 所示。其中 ShipName 字段属于表 Orders，UnitPrice 字段属于表 Order Details，OrderID 属于两个表共有的关联字段，通过该字段，两个表中的记录建立对应关系。

	EmployeeID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

图 12-15 唯一值

	OrderID	ShipName	UnitPrice
1	10248	Vins et alcools Chevalier	14.0000
2	10248	Vins et alcools Chevalier	9.8000
3	10248	Vins et alcools Chevalier	34.8000
4	10249	Toms Spezialitäten	18.6000
5	10249	Toms Spezialitäten	42.4000
6	10250	Hanari Carnes	7.7000
7	10250	Hanari Carnes	42.4000
8	10250	Hanari Carnes	16.8000
9	10251	Victuailles en stock	16.8000
10	10251	Victuailles en stock	15.6000
11	10251	Victuailles en stock	16.8000
12	10252	Suprêmes délices	64.8000

图 12-16 多表查询

Tips SQL Server 内置有许多关键字，若表名或字段名与关键字同名，有时无法正确执行 SQL 命令，可通过[]将与关键字同名的表名或字段名包括起来，避免这种同名异常。

多表查询时，若两个表中有同名的字段，应通过“表名.字段名”方式加以区分，未重复的字段无须添加表名，*表示读取所有字段。

where 子句用于设置条件表达式，只有符合 where 表达式要求的记录，才放入读取结果中，若有多个条件表达式，则应使用 and 关键字连接，如获取 EmployeeID 为 5，ShipCity 为 London

的记录，则 SQL 语句如下：

```
select * from Orders where EmployeeID=5 and ShipCity='London'
```

执行结果如图 12-17 所示。

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia
1	10359	SEVES	5	1996-11-21 00:00:00.000	1996-12-19 00:00:00.000	1996-11-26 00:00:00.000	3
2	10869	SEVES	5	1998-02-04 00:00:00.000	1998-03-04 00:00:00.000	1998-02-09 00:00:00.000	1

图 12-17　条件查询

Tips　在条件表达式中，若字段为数值类型，用=或其他符号比较时，无须添加引号，如 EmployeeID=5。若字段为字符类型，比较时应添加引号，表明要比较的是个字符串，如 ShipCity='London'。

order by 根据指定字段，对读取结果进行排序，其中 ASC 表示升序（Add），DESC 表示降序（Decrease），可设置多个排序字段，根据字段顺序依次排序，如先按 EmployeeID 字段，后按 OrderID 字段升序排序，SQL 语句如下：

```
select * from orders order by EmployeeID,OrderID ASC
```

执行结果如图 12-18 所示。

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate
1	10258	ERNSH	1	1996-07-17 00:00:00.000	1996-08-14 00:00:00.000	1996-07-23 00:00:00.000
2	10270	WARTH	1	1996-08-01 00:00:00.000	1996-08-29 00:00:00.000	1996-08-02 00:00:00.000
3	10275	MAGAA	1	1996-08-07 00:00:00.000	1996-09-04 00:00:00.000	1996-08-09 00:00:00.000
4	10285	QUICK	1	1996-08-20 00:00:00.000	1996-09-17 00:00:00.000	1996-08-26 00:00:00.000
5	10292	TRADH	1	1996-08-28 00:00:00.000	1996-09-25 00:00:00.000	1996-09-02 00:00:00.000
6	10293	TORTU	1	1996-08-29 00:00:00.000	1996-09-26 00:00:00.000	1996-09-11 00:00:00.000
7	10304	TORTU	1	1996-09-12 00:00:00.000	1996-10-10 00:00:00.000	1996-09-17 00:00:00.000
8	10306	ROMEY	1	1996-09-16 00:00:00.000	1996-10-14 00:00:00.000	1996-09-23 00:00:00.000
9	10311	DUMON	1	1996-09-20 00:00:00.000	1996-10-04 00:00:00.000	1996-09-26 00:00:00.000
10	10314	RATTC	1	1996-09-25 00:00:00.000	1996-10-23 00:00:00.000	1996-10-04 00:00:00.000
11	10316	RATTC	1	1996-09-27 00:00:00.000	1996-10-25 00:00:00.000	1996-10-08 00:00:00.000
12	10325	KOENE	1	1996-10-09 00:00:00.000	1996-10-23 00:00:00.000	1996-10-14 00:00:00.000

图 12-18　升序排序

12.1.5　数据更新语言

数据更新包括记录的添加、更新、删除，添加记录使用 insert 语句，格式如下：

```
insert into 表名[(字段 1，字段 2)] values(字段值 1，字段值 2)
```

除了自动递增的标识字段，所有非空字段必须赋值，其他字段可部分赋值。若所有字段都赋值，可省去字段列表，直接在 values 中按顺序放入要添加的值。

使用 Person 数据库，在查询分析器中输入以下 SQL 语句，在 friends 表中添加一条记录。

```
insert into friends values('乐乐','女','1990-1-1','13212345678','10601548',
'lele@abc.com','china')
```

执行后显示“所影响的行数为 1 行”，表示成功插入一条记录。由于对所有字段都赋值，无须设置字段列表，自动按照字段顺序依次赋值。

也可只对部分字段赋值，其中非空字段必须赋值，输入以下 SQL 语句：

```
insert into friends(name,mobile,qq) values('火驹','13213141213','10161106');
select * from friends
```

执行结果如图 12-19 所示。只设置其中三个字段的值，其他字段自动设为默认值，一般为空。

	id	name	sex	birthday	mobile	qq	email	address
1	5	乐乐	女	1990-01-01 00:00:00	13212345678	10601548	lele@abc.com	china
2	6	火驹	NULL	NULL	13213141213	10161106	NULL	NULL

图 12-19　插入记录

更新记录使用 update 语句，格式如下：

```
update 表名 set 字段1=字段值1 [,字段2=字段值2] where 条件表达式
```

更新部分字段的值，不同字段间用逗号分隔，where 子句限定要修改的记录。若不设置 where 条件，则将修改所有记录。

> **Tips** 在实际开发中，应先使用部分样例数据做测试，直到程序测试结束，正确无误后，再使用真实数据，否则若一不小心忘记设置 where 条件，则所有记录都会被更新。

输入以下 SQL 语句，修改添加的记录。

```
update friends set sex='男',birthday='1900-1-5',mobile='01012345678' where name='火驹'
select * from friends
```

执行结果如图 12-20 所示。where 子句限定只修改 name 为“火驹”的记录，更新其中三个字段的值。

	id	name	sex	birthday	mobile	qq	email	address
1	5	乐乐	女	1990-01-01 00:00:00	13212345678	10601548	lele@abc.com	china
2	6	火驹	男	1900-01-05 00:00:00	01012345678	10161106	NULL	NULL

图 12-20 更新记录

删除记录使用 delete 语句，格式如下：

```
delete from 表名 where 条件表达式
```

根据 where 条件，删除符合条件的一条或多条记录，若不设置 where 条件，将删除表格所有记录。在使用 delete 语句时，一定要慎重。

输入以下 SQL 语句，删除一条记录。wher 子句限定只删除 name 为“火驹”的记录。

```
delete from friends where name='火驹'
select * from friends
```

执行结果如图 12-21 所示。

	id	name	sex	birthday	mobile	qq	email	address
1	5	乐乐	女	1990-01-01 00:00:00	13212345678	10601548	lele@abc.com	china

图 12-21 删除记录

12.1.6 ADO 数据库访问技术

Microsoft 提供多种数据库访问技术，如 ODBC、ADO、ADO.NET，其中 ODBC（Open Database Connectivity，开放数据库连接）提供一组访问数据库的 API 函数，可直接利用 API 实现数据库的访问和更新，但 ODBC 方式需要手动配置数据源，可移植性差。

ADO.NET 是.NET 环境下的数据库访问技术，基于.NET Framework 框架，以 XML 形式存放数据，是 ADO 技术的升级版，使用 ADO.NET 技术可以很轻松地实现各种数据库操作，且安全高效，但只能在托管代码环境下使用，如 C#、VB.NET、C++.NET，而 Visual C++使用原始的物理代码，难以使用 ADO.NET 数据库访问技术。

ADO（ActiveX Data Objects）基于 Microsoft 的 ActiveX 技术，可在多种语言环境下使用，为不同的应用程序提供一个通用的访问接口。相对于 ODBC 技术，ADO 可移植性好，可随意将程序从一台机器转移到另一台机器上，无须重新配置数据源。相对于 ADO.NET 技术，ADO 效率更高，访问速度更快，适用于海量数据的读取和写入。

ADO 包括 7 个对象，各个对象的功能如下。

- Connection 连接对象：用于与数据库建立、断开连接。
- Command 命令对象：向数据库发送 SQL 命令，执行读取或更新操作。
- Recordset 记录集对象：用于获取读取的数据，以及表结构属性信息。
- Field 字段对象：用于表示表中的字段信息。
- Parameter 参数对象：用于在 Command 命令中设置参数。
- Error 错误对象：用于获取数据库访问的错误信息。
- Property 属性对象：用于获取对象的属性信息。

12.2 ADO 封装类

使用 ADO 技术访问 SQL Server 数据库具有速度快、效率高的优点，但 ADO 较为底层，操作复杂，程序经常需要访问数据库。若直接使用 ADO 对象访问数据库，会给开发带来不便，增加额外负担。有必要对 ADO 对象进行封装，将重复性代码封装为类成员函数，使用时通过 ADO 封装类对象，传入几个参数，即可获取所需数据，无须关心 ADO 对象的打开或关闭、字段的读取等步骤，从而简化开发、提高效率。

12.2.1 类头文件定义

【实例 12-1】创建 ADO 封装类，将 ADO 常用功能封装为类成员函数。

```
#ifndef _ADOEx_CLASS
#define _ADOEx_CLASS
#pragma warning(disable:4146)     //屏蔽警告
//导入 ADO 库
#import "C:\Program Files\Common Files\System\ado\msado15.dll" no_namespace rename
("EOF","adoEOF")
class CADOEx                      //ADO 封装类
{
public:
    int GetSingleStringNum(CString strMulti,CString strSplit);//获取分隔字符串元素数目
    int GetIndexOfString(CString strSource,CString strFind,CString strSplit="->");
                                  //获取元素位置
    BOOL GetConnectState();                                    //获取连接状态
    long GetRecordColNum();                                    //获取字段列数
    long GetRecordCount();                                     //获取记录行数
    void CloseRecordset();                                     //关闭记录集
    BOOL ExecuteNotSelSQL(CString strNotSelSQL);               //执行非查询语句
    BOOL ExecuteSelSQL(CString strSelectSQL);                  //执行 select 语句
    //查询单字段
    BOOL  ExecuteSelSQL(CString  strSelectSQL,CString  strFieldName,CStringArray&
strResult);
    //查询多字段
    BOOL  ExecuteSelSQL(CString  strSelectSQL,CStringArray&  strFieldName,CString
-Array& strResult
      ,CString strSplit="*@#$*");
    CString GetSingleString(CString strMulti,int nIndex,CString strSplit="*@#$*");
                                                               //获取单个元素

    void DisConnect();                                         //关闭连接
    BOOL Connect(CString strLinkDB);                           //连接数据库
    CADOEx();                                                  //构造函数
    virtual ~CADOEx();                                         //析构函数
protected:
    BOOL bState;                  //连接状态
    _RecordsetPtr m_pRs;          //记录集对象
```

```
    _CommandPtr m_pCmd;                    //命令对象
    _ConnectionPtr m_pConn;                //连接对象
};

#endif
```

类所在的头文件通常都有#ifndef、#define、#endif 三个预编译指令，用于避免头文件的重复包含，如文件 A 包含文件 C，文件 B 也包含文件 C，若在另一个文件 D 中同时包含文件 A 和 B，则 D 中包含两份 C，这会造成 C 中的类的重复定义，编译器会报错。

#ifndef 先判断是否定义了宏_ADOEx_CLASS，若没有定义，#define 定义宏_ADOEx_CLASS，若已定义，编译时跳过#ifndef 和#endif 块内的代码，#endif 结束预编译判断指令。第一次包含头文件时，定义宏_ADOEx_CLASS，编译头文件，第二次包含同一个头文件时，已定义了宏_ADOEx_CLASS，不再编译该头文件，通过这种预编译判断方式，避免头文件的重复编译。

#pragma warning（disable:4146）用于屏蔽导入 ADO 库时，引发的警告信息。#pragma 用于以代码方式配置开发环境，如在文件开头处添加一句#pragma comment（lib, "opengl32.lib"），以代码方式添加库文件 opengl32.lib，这样可以提高程序的可移植性，无须手动配置开发环境。

#import 用于导入已有 DLL 库文件，使用 ADO 技术需要 msado15.dll 文件支持，一般在 C 盘中存放，该 DLL 文件仅在开发时使用，程序完成后不再需要该 DLL 文件。rename 用于修改库文件中的属性名称，如将属性 EOF 重命名为 adoEOF，避免同名属性造成的混乱。

Tips #pragma warning(disable:4146)和#import 语句是使用 ADO 对象的必备工作，ADO 开发较为复杂，但也有固定的套路，无须了解过多内部细节，按照固定的套路，能够正确、高效地读取和更新数据，完成程序所需功能即可。

_ConnectionPtr、_CommandPtr、_RecordsetPtr 分别是连接、命令、记录集对象的智能指针，通过指针动态创建、释放对象，可在构造函数中创建对象，在析构函数中关闭、释放对象。

12.2.2 数据库连接函数

ADO 封装类中函数众多，其中与数据库连接相关的函数有构造函数、析构函数、Connect 函数、DisConnect 函数，函数实现如下：

```
#include "ADOEx.h"
CADOEx::CADOEx()                                          //构造函数
{
    CoInitialize(NULL);                                   //COM 初始化
    bState=FALSE;                                         //连接状态为 false
    m_pConn.CreateInstance(__uuidof(Connection));         //创建 Connection 对象
    m_pRs.CreateInstance(__uuidof(Recordset));            //创建 Recordset 对象
    m_pCmd.CreateInstance(__uuidof(Command));             //创建 Command 对象
}
CADOEx::~CADOEx()                                         //析构函数
{
    if(m_pRs->State!=adStateClosed)                       //关闭记录集
        m_pRs->Close();
    m_pRs.Release();                                      //释放 Recordset 对象
    m_pRs=NULL;
    if(m_pConn->State!=adStateClosed)                     //关闭连接
        m_pConn->Close();
    m_pConn.Release();                                    //释放 Connection 对象
    m_pConn=NULL;
    m_pCmd.Release();                                     //释放 Command 对象
```

```
    m_pCmd=NULL;
}
BOOL CADOEx::Connect(CString strLinkDB)                          //连接数据库
{
    HRESULT hr;
    _bstr_t bstrConn=(_bstr_t)strLinkDB;                         //类型转换
    try
    {
        DisConnect();                                            //断开连接
        m_pConn->ConnectionTimeout=8;                            //连接超时时间
        hr=m_pConn->Open(bstrConn,"","",adModeUnknown);          //打开连接
        if(FAILED(hr))                                           //若连接失败
        {
            bState=FALSE;
            return FALSE;
        }
    }
    catch(_com_error& e)                                         //捕获异常
    {
        e.ErrorMessage();
        bState=FALSE;
        return FALSE;
    }
    bState=TRUE;
    return TRUE;
}
void CADOEx::DisConnect()                                        //关闭连接
{
    if(m_pConn->State!=adStateClosed)
        m_pConn->Close();
    bState=FALSE;
}
```

在类实现所在的 CPP 文件开头处，应先添加一句#include "ADOEx.h" ，包含头文件。在构造函数中，先调用 CoInitialize 函数初始化 COM 组件，ADO 基于 COM 技术开发，在使用前应先初始化 COM。

初始设置 bState 为 FALSE，表示未连接到数据库。CreateInstance 函数用于创建 ADO 对象实例，参数为对象的名称或 CLSID 值，如 Connection 对象的名称为"ADODB. Connection"，可通过__uuidof 关键字自动获取对象的 CLSID 值，如__uuidof（Connection）等同于"ADODB.Connection"。

Tips CLSID 是 COM 类的 ID 值，是 128 位的全球唯一值，可通过 Microsoft Visual Studio\Common\Tools 目录下的 GUIDGEN.EXE 工具自动生成，如图 12-22 所示。无须担心这个值会重复，单击 New GUID 按钮生成一个新的 ID 值。

当 ADO 封装类对象释放时，自动调用析构函数，通过 State 属性判断对象是否已关闭，若未关闭，则调用 Close 函数关闭 Connection 和 Recordset 对象，再调用 Release 函数释放动态创建的对象实例，释放实例后，最好同时将指针值设为 NULL，避免系统再次自动释放实例。

Connect 函数用于连接数据库，参数 strLinkDB 为连接字符串，不同的数据库有不同的连接字符串，但也仅限于连接字符串不同，读取更新数据的方法基本一致，因此该 ADO 封装类可适用于不同类型的数据库，如 Excel、Access、SQL Server 等，只

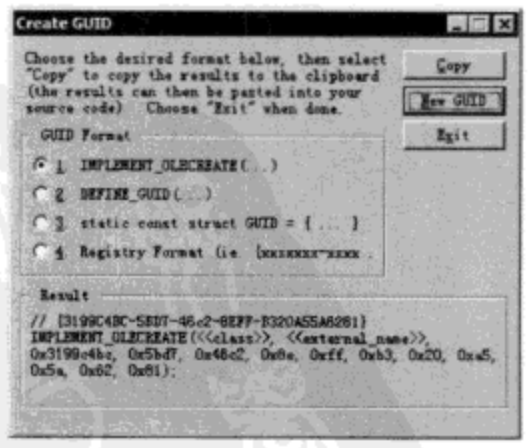


图 12-22　GUID 生成器

需使用不同的连接字符串。

HRESULT 实际就是 LONG 类型，主要用于函数返回结果，类似 LRESULT，LRESULT 主要用于消息函数的返回结果。_bstr_t 类用于存储 COM 字符串 BSTR 类型数据，类似于 CString 类，可将 CString 类字符串直接强制转换为_bstr_t 类字符串。

try、catch 结构用于异常控制，数据库操作异常时有发生，异常控制是必需的，否则稍有异常，程序就会崩溃。DisConnect 函数用于断开已有连接，ConnectionTimeout 属性设置超时时间，当超出指定时间后，若仍未连接上数据库，则停止连接尝试。Open 函数连接数据库，参数 1 为连接字符串，返回值存放到 hr 中。

FAILED 宏用于判断返回结果是否成功，宏格式如下：

```
#define FAILED(Status) ((HRESULT)(Status)<0)
```

若打开失败，则 FAILED 宏结果为 TRUE，设置 bState 为 FALSE，返回 FALSE。若打开时发生异常，也做同样处理。若打开成功，则设置 bState 为 TRUE，返回 TRUE。

12.2.3 SQL 命令函数

与 SQL 命令有关的函数有 4 个，其中 ExecuteNotSelSQL 函数用于执行非 select 语句，如 insert、update、delete 语句，ExecuteSelSQL 函数有三个重载版本，用于单字段和多字段的查询，函数实现如下：

```
BOOL CADOEx::ExecuteNotSelSQL(CString strNotSelSQL)   //执行更新操作
{
    try
    {
        CloseRecordset();                                    //关闭记录集
        _variant_t vResult;
        vResult.vt=VT_ERROR;
        vResult.scode=DISP_E_PARAMNOTFOUND;
        m_pCmd->ActiveConnection=m_pConn;                    //设置关联的 Connection 对象
        m_pCmd->CommandText=(_bstr_t)strNotSelSQL;           //SQL 命令
        m_pRs=m_pCmd->Execute(&vResult,&vResult,adCmdText);  //执行命令
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());                     //显示错误信息
        return FALSE;
    }
    return TRUE;
}
BOOL CADOEx::ExecuteSelSQL(CString strSelectSQL)             //执行查询语句
{
    try
    {
        CloseRecordset();                                    //关闭记录集
        m_pRs->CursorLocation=adUseClient;                   //设置游标为客户端
        m_pRs->Open(_variant_t(strSelectSQL),_variant_t((IDispatch*)m_pConn,true),
adOpenDynamic,adLockOptimistic,adCmdText);                   //打开数据集
        if(m_pRs->GetRecordCount()==0)                       //若返回记录数为 0
            return FALSE;
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());                     //显示错误信息
        return FALSE;
    }
```

```
        return TRUE;
    }
    //单字段查询
    BOOL CADOEx::ExecuteSelSQL(CString strSelectSQL,CString strFieldName, CStringArray
&strResult)
    {
        try
        {
            strResult.RemoveAll();                                    //清空结果数组
            if(ExecuteSelSQL(strSelectSQL)!=FALSE)                    //若 select 语句执行成功
            {
                strResult.SetSize(m_pRs->GetRecordCount());           //设置结果数组大小
                for(int i=0;i<m_pRs->GetRecordCount();i++)            //记录集遍历
                {
                    CString strValue;
                    _variant_t var=m_pRs->GetCollect((_bstr_t)strFieldName);
                                                                      //获取指定字段值
                    if(var.vt!=VT_NULL)                               //若字段值不为空
                        strValue=(char*)(_bstr_t)var;                 //转为字符串
                    else
                        strValue=" ";                                 //若为空，设为空字符串
                    strResult.SetAtGrow(i,strValue);                  //加入结果数组中
                    m_pRs->MoveNext();                                //记录移至下一行
                }
                return TRUE;
            }
        }
        catch(_com_error& e)
        {
            AfxMessageBox(e.ErrorMessage());                          //显示错误信息
            return FALSE;
        }
        return FALSE;
    }
    //多字段查询
    BOOL CADOEx::ExecuteSelSQL(CString strSelectSQL, CStringArray &strFieldName
    , CStringArray &strResult,CString strSplit)
    {
        try
        {
            strResult.RemoveAll();                                    //结果数组清空
            if(ExecuteSelSQL(strSelectSQL)!=FALSE)                    //若执行 sql 成功
            {
                strResult.SetSize(m_pRs->GetRecordCount());           //设置数组大小
                for(int i=0;i<m_pRs->GetRecordCount();i++)            //记录集遍历
                {
                    CString strValue;
                    for(int j=0;j<strFieldName.GetSize();j++)//要查询的字段遍历
                    {
                        _variant_t      var=m_pRs->GetCollect((_bstr_t)strFieldName.
GetAt(j));
                                                                      //获取字段值
                        if(var.vt!=VT_NULL)                           //若值不为空
                            strValue+=(char*)(_bstr_t)var;            //转为字符串
                        else
                            strValue+=" ";
                        if(j!=strFieldName.GetSize()-1)               //若当前字段不是最后一个
                            strValue+=strSplit;                       //字段值后加上分隔符
                    }
                    strResult.SetAtGrow(i,strValue);                  //添加至结果数组
                    m_pRs->MoveNext();                                //记录集移至下一行
```

```
            }
            return TRUE;
        }
    }
    catch(_com_error& e)
    {
        AfxMessageBox(e.ErrorMessage());                    //显示错误信息
        return FALSE;
    }
    return FALSE;
}
```

ExecuteNotSelSQL 函数用于执行非查询语句，参数 strNotSelSQL 为要执行的 SQL 命令字符串。CloseRecordset 函数关闭记录集对象，_variant_t 类用于存储 VARIANT 类型数据，VARIANT 类型可用于存放任意类型的数据，常作为通用数据类型。

Tips VARIANT 类型实际上是一个不同数据类型构成的 union 联合类型，其中 vt 属性存储真正的变量类型，VARIANT 变量存储的数据与开发语言无关，可在任意支持 OLE 的环境中使用。

ActiveConnection 属性设置 Command 命令使用的连接对象，CommandText 属性设置要执行的 SQL 命令。Execute 函数执行 SQL 命令，参数 3 指定 SQL 命令类型，如 adCmdText 表示单一的 SQL 语句，adCmdStoredProc 表示存储过程。

Tips 存储过程是 SQL Server 提供的一种数据访问技术，可将多条 SQL 语句组合起来，作为一个单独的单元一次执行，减少客户端和服务器的交流次数，提高执行效率。

ExecuteSelSQL（CString strSelectSQL）函数为另外两个同名函数服务，参数 strSelectSQL 为 select 查询语句。CursorLocation 属性设置游标为客户端，设置后，GetRecordCount 函数可获取记录集的记录行数，否则需要从头到尾遍历一遍后，才能获取记录数。

Open 函数打开记录集，参数 1 为 SQL 查询语句，参数 2 为当前 Connection 对象，参数 5 指定 SQL 语句类型。执行查询命令后，GetRecordCount 函数获取记录行数，若为 0 行，则表示读取失败，返回 FALSE。

第 2 个 ExecuteSelSQL 函数用于单字段查询，参数 1 为 SQL 命令，参数 2 为字段名，参数 3 为可变数组的引用，用于存放读取结果。RemoveAll 函数清空结果数组，先调用第 1 个 ExecuteSelSQL 函数，打开记录集。

GetRecordCount 函数获取记录集行数，SetSize 函数设置结果数组的大小，以便一次分配足够内存。利用 for 循环遍历记录集，GetCollect 函数获取当前行的指定字段的值，参数为字段名，字段值存放到 var 中，参数也可为字段索引，如只读取一个字段，参数可设为 0。

var 是_variant_t 类型，vt 属性可用于判断真实的数据类型，若 vt 属性为 VT_NULL，表明该字段值为空，若为空，则设为空字符串，若不为空，则通过(char*)(_bstr_t)var 形式，强制转换为 CString 类型。

SetAtGrow 函数设置结果数组指定位置的元素值，MoveNext 函数将记录集指针移动到下一行。通过 for 循环遍历所有行，指定字段的值都存放到 strResult 可变数组中。

Tips CStringArray 类不能作为函数返回类型，只能通过传递对象引用方式，作为输出参数，获取函数执行结果。

第 3 个 ExecuteSelSQL 函数用于多个字段的查询，参数 1 为 SQL 命令，参数 2 为字段名数组，参数 3 为结果数组，参数 4 为分隔符。外层 for 循环遍历所有行，内层 for 循环遍历字段数组，GetCollect 获取当前行的各个字段的值，存放到 var 中。

strValue 用于存放当前行所有字段的值，不同字段间添加分隔符，如读取 3 个字段，则 strValue 的值可能为“值 1*@#$*值 2*@#$*值 3”，其中*@#$*为默认的分隔符，将各个行获取的 strValue 值添加到 strResult 结果数组中。

Tips 在 MFC 环境下，要实现二维的 CString 可变数组是比较困难的，通过分隔符将所有列的字符串拼接为一个字符串，再存入一维可变数组中，使用时再拆分为多个字符串，是一种合理高效的方法。

12.2.4 相关辅助函数

多字段查询时，所有字段的值拼接为一个字符串存入一维数组中，使用时需要拆分为多个字符串。还有一些其他辅助函数，如获取字段数目、获取连接状态、关闭记录集等，函数实现如下：

```
void CADOEx::CloseRecordset()                          //关闭记录集
{
    if(m_pRs->State!=adStateClosed)                    //若记录集尚未关闭
        m_pRs->Close();
}
long CADOEx::GetRecordCount()                          //获取记录集行数
{
    if(m_pRs->State==adStateClosed)                    //若记录集关闭，返回 0
        return 0;

    m_pRs->MoveFirst();                                //移至首行
    while(!m_pRs->adoEOF)
        m_pRs->MoveNext();                             //从首行移至尾行
    long nCount=m_pRs->GetRecordCount();               //获取记录行数
    m_pRs->MoveFirst();                                //再移回首行
    return nCount;                                     //返回行数
}
long CADOEx::GetRecordColNum()                         //获取记录集列数
{
    if(m_pRs->State==adStateClosed)
        return 0;
    long nCount=m_pRs->Fields->Count;
    return nCount;                                     //返回字段数
}
BOOL CADOEx::GetConnectState()                         //获取连接状态
{
    return bState;
}
//获取多分隔符字符串中子字符串个数
int CADOEx::GetSingleStringNum(CString strMulti, CString strSplit)
{
    if(!strMulti.IsEmpty())                            //若不为空
    {
        int nSplitCount=strMulti.Replace(strSplit,strSplit);  //获取分隔符个数
        return nSplitCount+1;                          //返回子字符串个数
    }
    return 0;
}
```

```
//获取多分隔符字符串中某个子字符串
CString CADOEx::GetSingleString(CString strMulti, int nIndex, CString strSplit)
{
    CString strAim="";
    if(!strMulti.IsEmpty())                                          //若不为空
    {
        int nSplitCount=strMulti.Replace(strSplit,strSplit);  //获取分隔符个数
        if(nIndex>nSplitCount+1 || nIndex<1 || nSplitCount==0)
            return "";
        if(nIndex==1)                                                //若序号为 1
        {
            int nBegin=0;
            if((nBegin=strMulti.Find(strSplit,nBegin))!=-1)//查找第一个分隔符左边的字符串
                strAim=strMulti.Left(nBegin);
        }
        else if(nIndex==nSplitCount+1)                               //若序号为最后一个
        {
            int nBegin=0,nPlace=0;
            while((nBegin=strMulti.Find(strSplit,nBegin))!=-1)//查找最后一个分隔符位置
                nPlace=nBegin++;
            if(nPlace)                                               //获取最后一个分隔符右边的字符串

strAim=strMulti.Right(strMulti.GetLength()-nPlace-strSplit.GetLength());
        }
        else                                                         //若为中间序号
        {
            int nBeginPlacePre=0,nSplitIndex=0,nBeginPlaceNext=0;
            while((nBeginPlacePre=strMulti.Find(strSplit,nBeginPlacePre))!=-1)
                                                                     //从前往后查找分隔符
            {
                nSplitIndex++;                                       //分隔符数目递增
                if(nSplitIndex==nIndex-1)                            //若到指定序号
                {                                                    //查找下一个分隔符位置
                    if((nBeginPlaceNext=strMulti.Find(strSplit,nBeginPlacePre
+1))!=-1)
                    {
                        strAim=strMulti.Mid(nBeginPlacePre+strSplit.GetLength(),
                          nBeginPlaceNext-nBeginPlacePre-strSplit.GetLength());
                        return strAim;
                    }
                }
                else                                                 //若没到指定序号，继续查找
                    nBeginPlacePre++;
            }
        }
    }
    return strAim;                                                   //返回子字符串值
}
//获取指定字符串在拼接字符串中的位置
int CADOEx::GetIndexOfString(CString strSource, CString strFind, CString strSplit)
{
    CString strTemp;
    int nIndex;
    nIndex=strSource.Find(strSplit+strFind+strSplit); //查找形式类似 "->aa->"
    if(nIndex!=-1)                                                   //若找到
    {
        CString strLeft=strSource.Left(nIndex+strSplit.GetLength()+1);
                                                                     //获取字符串左边的分隔符数目
        int nCount=strLeft.Replace(strSplit,strSplit);
        return nCount+1;                                             //返回序号
```

```
    }
    else                                                    //若没找到 "->aa->"
    {
        nIndex=strSource.Find(strSplit+strFind);            //查找形式类似 "->aa"
        if(nIndex!=-1)                                      //若找到
        {                                                   //若确实处于最后面
if((nIndex+strSplit.GetLength()+strFind.GetLength())==strSource.GetLength())
            {
                int nCount=strSource.Replace(strSplit,strSplit);
                                                            //获取所有分隔符数目
                return nCount+1;                            //返回序号
            }
        }
        else                                                //若也没找到 "->aa"
        {
            nIndex=strSource.Find(strFind+strSplit); //查找形式类似 "aa->"
            if(nIndex!=-1)                                  //若找到
            {
                if(nIndex==0)                               //若确实处于最前面
                    return 1;                               //返回 1
                else
                {
                    if(strFind.GetLength()==strSource.GetLength())
                                                            //若目标字符串长度等于源
                        return 1;                           //返回 1
                    return 0;
                }
            }
            return 0;                                       //若没找到，返回 0
        }
    }
```

CloseRecordset 函数用于关闭记录集。GetRecordCount 函数获取记录集行数，MoveFirst 函数将记录指针移至首行，adoEOF 为记录行结束标志，利用 while 循环和 MoveNext 函数遍历所有行后，再调用 Recordset 对象的 GetRecordCount 函数获取行数。

Tips 若设置 CursorLocation 属性为 adUseClient，可直接调用 Recordset 对象的 GetRecordCount 函数获取记录行数。

GetRecordColNum 函数获取记录集的字段数目，通过 Fields 属性获取记录集的 Field 对象，再通过字段对象的 count 属性获取字段数目。GetConnectState 函数获取保护成员 bState 的值，表明当前的连接状态。

GetSingleStringNum 函数用于获取拼接字符串中的元素数目，参数 1 为拼接字符串，如“a->b->c”，参数 2 为分隔符，如“->”，返回值为 3。GetSingleString 函数获取拼接字符串中单个字符串的值，参数 1 为拼接字符串，如“a->b->c”，参数 2 为位置索引，如 2，参数 3 为分隔符，如“->”，返回值为“b”。这两个函数的具体介绍请参见“7.3 工具栏”一节。

GetIndexOfString 函数用于获取某个字符串在拼接字符串的位置，参数 1 为拼接字符串，如“a->b->c”，参数 2 为目标字符串，如“c”，参数 3 为分隔符，如“->”，则函数返回值为 3，函数内部算法如下所示：

- 先查找“->c->”形式，若找到，则获取“c”左边的分隔符“->”的数目，返回值为分隔符数目加 1。

- 若未找到，再查找“->c”形式，即目标字符串是最后一个元素，若找到且确实是最后一个元素，则返回值为所有分隔符数目加 1。
- 若仍未找到，再查找“c->”形式，即目标字符串为第一个元素，若找到且确实是第一个元素，则返回 1。若不是第一个元素，则判断目标字符串和拼接字符串长度相等，若相等，则说明是同一个字符串，则返回 1。
- 若三种情况都不存在，则返回 0。

Tips 该函数获取精确匹配结果，如参数 1 为“ab->df->dg”，参数 2 为“d”，参数 3 为“->”，则函数返回值为 0。所以在查找到匹配项，如“->d”后，还需要判断是否确实为最后一个元素，若不是则继续判断。

12.3 ADO 访问数据库

若直接使用 ADO 对象访问数据库，步骤如下：

（1）创建 Connection 对象，并连接到数据库。

（2）创建 Command 对象，执行 SQL 更新命令。

（3）创建 Recordset 对象，根据查询语句打开记录集，获取数据。

（4）关闭 Connection、Recordset 对象。

（5）释放 Connection、Command、Recordset 对象。

这种方式操作较为复杂，不便于快速开发，可使用自定义的 ADO 封装类，无须直接操作 ADO 对象，对象的创建在构造函数和析构函数里自动完成，相关功能都封装为类成员函数，传入简单的几个参数，就能执行 SQL 命令，获取所需数据。将封装类的定义保存为 ADOEx.h 文件，将封装类的函数实现代码保存为 ADOEx.cpp 文件，使用时将两个文件添加到工程中，即可利用封装类提供的强大功能。

12.3.1 连接数据库

【实例 12-2】新建一个单文档工程名为 Ado091，访问 SQL Server 中的 Person 数据库，实现添加、更新、删除、导出数据功能。

（1）新建单文档工程 Ado091，在最后一步设置 View 类的基类为 CListView。创建完成后，先将封装类所在的 ADOEx.h 和 ADOEx.cpp 文件复制到工程目录中，选择 ProjectlAdd To ProjectlFiles 命令，弹出 Insert Files into Project 窗口，如图 12-23 所示。

（2）选择 ADOEx.h 和 ADOEx.cpp 文件，单击 OK 按钮，将封装类添加到工程中。

（3）在类视图双击 Globals 节点下的 theApp 项，定位到全局变量 theApp 定义处，添加一句代码：

```
CAdo091App theApp;
CADOEx theAdo;                    //全局 CADOEx 对象
```

（4）在当前 CPP 文件开头处，添加一句#include "ADOEx.h" ，包括 ADO 封装类的头文件。

（5）在资源视图右键单击 Dialog，在弹出的快捷菜单中选择 Insert Dialog 命令，添加一个对话框模板，拖放两个静态文本、编辑框、按钮到模板上，如图 12-24 所示。

（6）设置静态文本的 Caption 依次为“用户名”、“密码”。编辑框 ID 依次为 IDC_EDIT_USERNAME、IDC_EDIT_PASSWORD，其中“密码”编辑框勾选 Password 复选框。按钮 Caption 依次为“登录”、“注册”，ID 依次为 IDC_BUTTON_LOGIN、IDC_BUTTON_REG。

设置对话框模板 ID 为 IDD_DIALOG_LOGIN，Caption 为“登录窗口”，字体设为“宋体”、9 号。

（7）按 Ctrl+W 组合键，添加一个新类名为 CDlg_Login，基类为 CDialog。选择 Member Variable 选项卡，双击 IDC_EDIT_USERNAME 项，添加 CString 变量 m_strUsername，双击 IDC_EDIT_PASSWORD 项，添加 CString 变量 m_strPassword。

（8）双击 Menu 节点下的 IDR_MAINFRAME 项，打开菜单资源，删除所有菜单项，并添加两个根菜单，如图 12-25 所示。

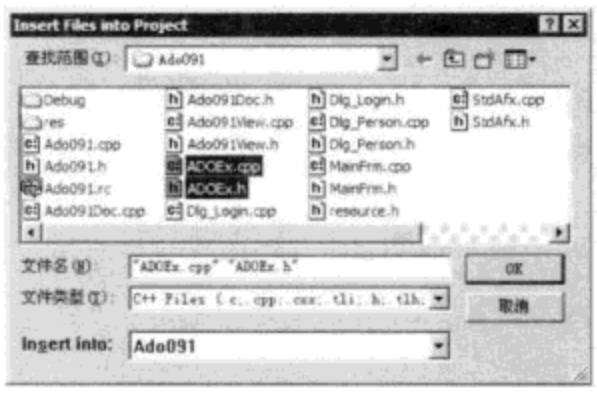


图 12-23　添加已有文件

图 12-24　登录窗口

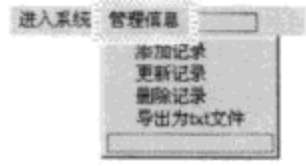


图 12-25　菜单项

（9）设置“进入系统”菜单项的 ID 为 ID_MENU_LOGIN，“添加记录”为 ID_MENUITEM_ADD，“更新记录”为 ID_MENUITEM_UPDATE，“删除记录”为 ID_MENUITEM_DELETE，“导出为 txt 文件”为 ID_MENUITEM_EXPORT。

（10）按 Ctrl+W 组合键打开类向导，选择 Message Maps 选项卡，Class name 组合框选择 CAdo091View 项，Object IDs 列表框依次选择以上 5 个菜单项的 ID，Messages 列表框依次选择 COMMAND、UPDATE_COMMAND_UI 项，添加 10 个消息处理函数，单击 OK 按钮保存。

（11）在类视图双击 CMainFrame 类下的 OnCreate 项，在 return 0; 前添加一句代码：

```
ShowControlBar(&m_wndToolBar,FALSE,FALSE);          //隐藏工具栏
```

（12）双击 CAdo091Doc 类下的 OnNewDocument 项，在 return TRUE; 前添加一句代码：

```
m_strTitle="个人信息管理";                          //设置文档标题
```

（13）右键单击 CAdo091View 类添加一个 BOOL 类型的变量 bState，再添加一个返回类型为 void 的函数 RefreshList()。

（14）双击 CAdo091View 类下的 OnInitialUpdate 项，添加如下代码：

```
void CAdo091View::OnInitialUpdate()
{
    CListView::OnInitialUpdate();
    // TODO: You may populate your ListView with items by directly accessing
    //  its list control through a call to GetListCtrl().
    CListCtrl& list=GetListCtrl();                                    //获取 CListCtrl 对象
    DWORD ctrlStyle=GetWindowLong(list.GetSafeHwnd(),GWL_STYLE);  //获取窗口样式
    ctrlStyle|=LVS_REPORT;                                            //使用报表样式
    SetWindowLong(list.GetSafeHwnd(),GWL_STYLE,ctrlStyle);     //设置新样式
    list.SetExtendedStyle(list.GetExtendedStyle()|LVS_EX_FULLROWSELECT
        |LVS_EX_GRIDLINES);                                           //设置扩展样式
    bState=FALSE;                                                     //登录标志，初始为 FALSE
    CString strConn="driver={SQL Server};Server=FQ-201003301100;Database=Person;"
        "uid=sa;pwd=sa";                                              //SQL Server 数据库连接字符串
    if(theAdo.Connect(strConn))                                       //连接 SQL Server 数据库
    {
        list.InsertColumn(0,"序号",LVCFMT_LEFT,50);    //若连接成功，则插入列
        list.InsertColumn(1,"姓名",LVCFMT_LEFT,80);
        list.InsertColumn(2,"性别",LVCFMT_LEFT,50);
        list.InsertColumn(3,"生日",LVCFMT_LEFT,80);
```

```
        list.InsertColumn(4,"手机号",LVCFMT_LEFT,80);
        list.InsertColumn(5,"QQ 号",LVCFMT_LEFT,80);
        list.InsertColumn(6,"邮箱",LVCFMT_LEFT,150);
        list.InsertColumn(7,"地址",LVCFMT_LEFT,300);
    }
    else
    {
        AfxMessageBox("连接数据库失败！");                    //若连接失败，则弹出提示信息
    }
}
```

GetListCtrl 函数获取 CListCtrl 对象的引用，GetWindowLong 函数获取窗口样式，SetWindowLong 函数设置列表视图使用报表样式，SetExtendedStyle 函数设置列表视图的扩展样式，使用网格和全行选中。

strConn 为数据库连接字符串，分为 4 部分，各部分作用如下。

- driver：数据驱动程序，不同的数据库有不同的字符串。
- Server 或 Data Source：数据库服务器名称或 IP 地址。
- Database 或 Initial Catalog：数据库名称。
- uid、pwd：登录 SQL Server 的用户名和密码。

Tips 不同的数据库有不同的连接字符串，其中 Server 可以用服务器主机名或 IP 地址，若用本机作为服务器，且没有连接网络，只能使用主机名，不能使用 IP 地址。

Connect 函数根据连接字符串连接 SQL Server 的 Person 数据库，若连接成功，则 InsertColumn 函数在列表视图中插入多个列，否则，AfxMessageBox 函数弹出错误信息提示框。

（15）在当前 cpp 文件开头处（#include 语句后），添加一句代码：

```
extern CADOEx theAdo;                                    //声明外部变量
```

若要使用在其他文件中定义的全局变量，需要使用 extern 关键字做外部声明，表明该变量确实已经在其他文件中定义过，声明后可在任意位置使用该变量。

（16）在当前 CPP 文件开头处，包含相关头文件，添加如下代码：

```
#include "Dlg_Login.h"                                   //包含“登录窗口”所在类的头文件
#include "ADOEx.h"                                       //包含 ADO 封装类的头文件
```

（17）双击 CAdo091View 类下的 OnMenuLogin 项，添加如下代码：

```
void CAdo091View::OnMenuLogin()
{
    CDlg_Login dlgLogin;                                 //对话框类对象
    if(dlgLogin.DoModal()==IDOK)                         //显示模态对话框，若返回 IDOK
    {
        bState=TRUE;                                     //登录标志设为 TRUE
        RefreshList();                                   //刷新列表
    }
    else
        bState=FALSE;
}
```

CDlg_Login 为“登录窗口”所在的类，用于注册和登录用户，DoModal 函数以模态形式显示窗口，若返回 IDOK，则表明成功登录，将登录标志 bState 设为 TRUE，同时获取数据库中的数据，在列表视图中显示。

（18）双击 CAdo091View 类下的 OnUpdateMenuLogin 项，添加如下代码：

```
void CAdo091View::OnUpdateMenuLogin(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(!bState);      //若登录标志为 TRUE，禁用该菜单
}
```

当登录成功时，无须再次执行“登录窗口”，应禁用该菜单项，避免重复登录，造成混乱。

（19）双击 CAdo091View 类下的 RefreshList 项，添加如下代码：

```
void CAdo091View::RefreshList()
{
    CString strSQL="select * from friends";   //查询语句，读取 friends 表中所有数据
    CString fields[]={"id","name","sex","birthday","mobile","qq","email","address"};
    CStringArray strField;                    //要查询的字段数组
    int i=0;
    for(;i<8;i++)                             //填充字段数组
        strField.Add(fields[i]);
    CStringArray strResult;
    if(theAdo.ExecuteSelSQL(strSQL,strField,strResult))   //执行多字段查询
    {
        CListCtrl& list=GetListCtrl();
        list.DeleteAllItems();                            //清空列表视图已有项

        for(i=0;i<strResult.GetSize();i++)                //遍历读取的所有记录
        {
            CString strID=theAdo.GetSingleString(strResult.GetAt(i),1);
                                                          //获取第 1 列的值
            int nItem=list.InsertItem(list.GetItemCount(),strID); //插入新项
            for(int j=1;j<strField.GetSize();j++)         //遍历剩余列
            {
                CString strItem=theAdo.GetSingleString(strResult.GetAt(i),j+1);
                strItem.TrimRight();                      //去除字符串右边的空格
                list.SetItemText(nItem,j,strItem);        //设置新项其他列的值
            }
        }
    }
}
```

当初次显示数据，或添加、更新、删除记录后，应重新读取数据库中的数据，刷新列表视图。strSQL 为查询语句，获取 friends 表中的所有数据，Add 函数将 fields 数组中的所有字符串添加到字段数组中。

ExecuteSelSQL 函数用于执行多字段查询，获取的数据存放到 strResult 中。DeleteAllItems 函数清空列表视图的已有项，外层 for 循环遍历所有记录，内层 for 循环遍历字段。GetSingleString 函数获取拼接字符串中单个字符串的值，先获取 id 字段的值，调用 InsertItem 函数插入新项，再获取其他字段的值，调用 SetItemText 函数设置其他列的值，TrimRight 函数用于清除字符串右边的空格。

Tips　数据库中的字段若为 char 类型，会自动在字段值后添加多个空格，影响操作，TrimRight 函数用于清除右边多余的空格，TrimLeft 函数用于清除左边多余的空格。

（20）在资源视图双击 IDD_DIALOG_LOGIN 项，双击对话框模板上的“登录”和“注册”按钮，添加如下代码：

```
void CDlg_Login::OnButtonLogin()                          //“登录”按钮
{
    UpdateData();                                         //更新控件变量的值
    if(m_strUsername.IsEmpty() || m_strPassword.IsEmpty())  //若输入为空，则返回
        return;
```

```
    if(theAdo.GetConnectState())                                    //若已连接数据库
    {
        CString strSQL="select * from loginUser where username='"+m_strUsername
            +"' and password='"+m_strPassword+"'";                  //查询该用户名和密码是否存在
        CStringArray strResult;
        if(theAdo.ExecuteSelSQL(strSQL,"username",strResult)) //单字段查询
        {
            if(strResult.GetSize()>0)                               //若存在
                CDialog::OnOK();                                    //关闭模态对话框，返回 IDOK
        }
        else
            MessageBox("登录失败！","提示信息");                    //若不存在，则弹出错误提示信息
    }
}
void CDlg_Login::OnButtonReg()                                      //“注册”按钮
{
    UpdateData();                                                   //更新控件变量的值
    if(m_strUsername.IsEmpty() || m_strPassword.IsEmpty())
        return;
    if(m_strUsername.GetLength()>20 || m_strPassword.GetLength()>20)
                                                                    //若输入字符数大于 20
    {
        MessageBox("输入内容超长，请重新输入","提示信息"); //弹出错误提示信息
        return;
    }

    if(theAdo.GetConnectState())                                    //若已连接数据库
    {
        CString strSQL="insert into loginUser values('"+m_strUsername
            +"','"+m_strPassword+"')";                              //插入一条记录
        if(theAdo.ExecuteNotSelSQL(strSQL))                         //执行插入命令
        {
            MessageBox("注册成功！","提示信息");                    //弹出成功提示信息
            m_strPassword="";                                       //清空“密码”编辑框
            UpdateData(FALSE);                                      //更新控件
        }
    }
}
```

（21）在当前 cpp 文件开头处，添加一句#include "ADOEx.h"，包括 CADOEx 类的头文件。在下方再添加一句 extern CADOEx theAdo;，声明外部全局变量 theAdo。

（22）生成程序并运行，如图 12-26 所示。单击“进入系统”菜单，弹出“登录窗口”对话框，输入用户名和密码，单击“注册”按钮，注册新用户，弹出“注册成功”信息。

图 12-26 登录窗口

Tips 运行程序时要保证已启动数据库服务，在桌面右下角若有托盘图标，且显示为绿色，表明数据库服务已正常启动。

（23）输入已注册的用户名和密码，单击“登录”按钮，在主窗口列表视图中显示 friends 表中的所有记录，如图 12-27 所示。

12.3.2　添加记录

（1）在资源视图右键单击 Dialog，在弹出的快捷菜单中选择 Insert Dialog 命令，添加一个对话框模板，拖放七个静态文本、五个编辑框、两个按钮、一个组合框、一个日期控件到模板上，如图 12-28 所示。

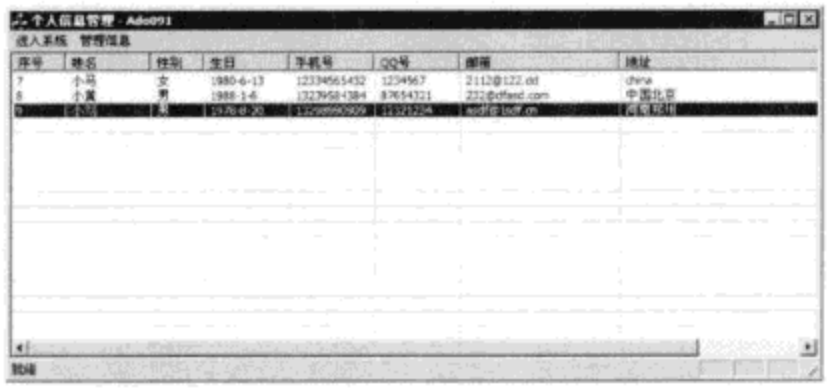


图 12-27　主窗口

图 12-28　编辑个人信息

（2）设置静态文件的 Caption 依次为“姓名”、“性别”、“生日”、“手机号”、“QQ 号”、“邮箱”、“地址”。设置编辑框的 ID 依次为 IDC_EDIT_NAME、IDC_EDIT_MOBILE、IDC_EDIT_QQ、IDC_EDIT_EMAIL、IDC_EDIT_ADDRESS，其中“手机号”和“QQ 号”编辑框勾选 Number 复选框。

（3）设置“性别”组合框 ID 为 IDC_COMBO_SEX，Type 为 Drop List，取消勾选 Sort 复选框。设置按钮的 Caption 依次为“确定”、“取消”，ID 依次为 IDC_BUTTON_OK、IDC_BUTTON_CANCEL。设置对话框的 Caption 为“编辑个人信息”，ID 为 IDD_DIALOG_PERSON，字体设为“宋体”、9 号。

（4）按 Ctrl+W 组合键，添加一个新类名为 CDlg_Person，基类为 CDialog。选择 Member Variable 选项卡，添加控件变量，如表 12-3 所示。

表 12-3 “编辑个人信息”对话框控件变量

控件 ID	类　型	变量名称
IDC_DATETIMEPICKER1	CDateTimeCtrl	m_dtBirthday
IDC_COMBO_SEX	CComboBox	m_comboSex
IDC_EDIT_ADDRESS	CString	m_strAddress
IDC_EDIT_EMAIL	CString	m_strEmail
IDC_EDIT_MOBILE	CString	m_strMobile
IDC_EDIT_NAME	CString	m_strName
IDC_EDIT_QQ	CString	m_strQQ

（5）在类视图双击 CDlg_Person 项，在类定义中添加如下变量，代码如下：

```
public:
    CString m_strID;                    //要更新项的 id
    CString m_strFlag;                  //标识符，区分添加、更新操作
    CString m_strBirthday;              //生日字符串
    CString m_strSex;                   //性别字符串
```

（6）右键单击 CDlg_Person 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_INITDIALOG 消息的处理函数，添加如下代码：

```
BOOL CDlg_Person::OnInitDialog()
{
```

```
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here
    m_comboSex.AddString("男");                          //“性别”组合框添加项
    m_comboSex.AddString("女");
    if(m_strSex=="男")                                   //若为“男”，选择第 1 项
        m_comboSex.SetCurSel(0);
    else
        m_comboSex.SetCurSel(1);

    if(!m_strBirthday.IsEmpty())                         //若生日字符串不为空
    {
        CString strYear=theAdo.GetSingleString(m_strBirthday,1,"-");  //获取年份
        CString strMonth=theAdo.GetSingleString(m_strBirthday,2,"-"); //获取月份
        CString strDay=theAdo.GetSingleString(m_strBirthday,3,"-");   //获取日期
        int nYear=atoi(strYear);                         //转换为整型值
        int nMonth=atoi(strMonth);
        int nDay=atoi(strDay);
        m_dtBirthday.SetTime(COleDateTime(nYear,nMonth,nDay,0,0,0));
                                                         //设置日期控件的值
    }

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

GetSingleString 函数获取日期字符串中各个字串的值，如“1900-1-1”，拆分为“1900”、“1”、“1”三个子串，atoi 函数将字符串转换为整型值。SetTime 函数用于设置日期控件的值，参数为 COleDateTime 类临时对象。

（7）在当前 cpp 文件开头处，添加一句#include "ADOEx.h" ，包括 CADOEx 类的头文件。在下方再添加一句 extern CADOEx theAdo; ，声明外部全局变量 theAdo。

（8）在资源视图双击 IDD_DIALOG_PERSON 项，双击对话框模板上的“确定”和“取消”按钮，添加如下代码：

```
void CDlg_Person::OnButtonOk()
{
    UpdateData();                                        //更新控件变量
    if(m_strName.IsEmpty() || m_strMobile.IsEmpty() || m_strQQ.IsEmpty() ||
        m_strEmail.IsEmpty() || m_strAddress.IsEmpty())
        return;                                          //若输入为空，则返回
    m_comboSex.GetLBText(m_comboSex.GetCurSel(),m_strSex);//获取选择的性别
    CTime timeBirthday;
    m_dtBirthday.GetTime(timeBirthday);                  //获取生日日期
    m_strBirthday.Format("%d-%d-%d",timeBirthday.GetYear(),timeBirthday.GetMonth(),
        timeBirthday.GetDay());                          //格式化为字符串
    CString strSQL="";
    if(m_strFlag=="insert")                              //若执行添加操作
    {
        strSQL="insert into friends values('"+m_strName+"','"+m_strSex+"','"+
            m_strBirthday+"','"+m_strMobile+"','"+m_strQQ+"','"+m_strEmail+"',
'"+
            m_strAddress+"')";                           //insert 语句
        if(theAdo.GetConnectState())
        {
            if(theAdo.ExecuteNotSelSQL(strSQL))          //执行 SQL 插入命令
            {
                MessageBox("添加记录成功","提示信息");     //弹出提示框
                CDialog::OnOK();                         //关闭模态对话框，返回 IDOK
            }
```

```
            }
        }
        else if(m_strFlag=="update")                                    //若执行更新操作
        {
            strSQL="update   friends   set   name='"+m_strName+"',sex='"+m_strSex+"',
birthday='"
                +m_strBirthday+"',mobile='"+m_strMobile+"',qq='"+m_strQQ+"',email='"
                +m_strEmail+"',address='"+m_strAddress+"' where id="+m_strID;
            if(theAdo.GetConnectState())                                //update 语句
            {
                if(theAdo.ExecuteNotSelSQL(strSQL))                     //执行 SQL 更新命名
                {
                    MessageBox("更新记录成功","提示信息");
                    CDialog::OnOK();                                    //关闭模态对话框，返回 IDOK
                }
            }
        }
    }
    void CDlg_Person::OnButtonCancel()                                  //“取消”按钮
    {
        CDialog::OnCancel();                                    //关闭模态对话框，返回 IDCANCEL
    }
```

GetLBText 函数获取“性别”组合框选中项的值，GetTime 函数获取选择的“生日”日期，Format 函数将日期格式化为“1900-1-1”形式。

m_strFlag 是标志符，用来判断执行插入还是更新操作。若为 insert，则 strSQL 为插入语句，在 friends 表中插入一条记录，ExecuteNotSelSQL 函数用于执行非 select 语句，若添加成功，则弹出成功提示框，并用 CDialog::OnOK 关闭模态对话框，DoModal 函数返回 IDOK。

若为 update 操作，strSQL 为更新语句，更新 friends 表中的一条记录，其中 m_strID 指定更新记录的 id，where 条件表达式用来限定符合条件的记录。“取消”按钮使用 CDialog::OnCancel 函数关闭模态对话框，DoModal 函数返回 IDCANCEL。

Tips　strSQL 是一个拼接字符串，拼接出一个能被 SQL Server 正确执行的 SQL 语句，若字段为 char 或 varchar 等字符串类型，要使用"将字符串包括起来，如"name='"+m_strName+"'"，传至 SQL Server 后实际上是 name='lili'。

（9）双击 CAdo091View 类下的 OnMenuitemAdd 项，添加如下代码：

```
void CAdo091View::OnMenuitemAdd()
{
    CDlg_Person dlgPerson;                          //“编辑个人信息”类对象
    dlgPerson.m_strFlag="insert";                   //设置标识符为 insert
    if(dlgPerson.DoModal()==IDOK)                   //弹出模态对话框，若返回 IDOK
        RefreshList();                              //刷新列表
}
```

（10）在当前 CPP 文件开头处，添加一句#include "Dlg_Person.h" ，包含 CDlg_Person 类的头文件。

（11）双击 CAdo091View 类下的 OnUpdateMenuitemAdd 项，添加如下代码：

```
void CAdo091View::OnUpdateMenuitemAdd(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(bState);                         //若已登录，菜单项可用
}
```

（12）生成程序并运行，单击“进入系统”菜单，输入用户名和密码，单击“登录”按钮进

入系统，显示所有信息。再单击“管理信息”|“添加记录”菜单，弹出“编辑个人信息”窗口，如图 12-29 所示。

（13）输入信息后，单击“确定”按钮，弹出“添加记录成功”提示信息，并在数据库中添加一条记录，如图 12-30 所示。

图 12-29 输入个人信息

图 12-30 添加记录

12.3.3 更新记录

（1）双击 CAdo091View 类下的 OnMenuitemUpdate 项，添加如下代码：

```
void CAdo091View::OnMenuitemUpdate()
{
    CDlg_Person dlgPerson;
    dlgPerson.m_strFlag="update";                          //设置标识符为 update
    CListCtrl& list=GetListCtrl();
    int nSel=list.GetNextItem(-1,LVNI_SELECTED);           //获取选中项索引
    if(nSel!=-1)
    {
        dlgPerson.m_strID=list.GetItemText(nSel,0);    //获取 id 值，指定要更新的记录
        dlgPerson.m_strName=list.GetItemText(nSel,1);//获取列值，初始化对话框
        dlgPerson.m_strSex=list.GetItemText(nSel,2);
        dlgPerson.m_strBirthday=list.GetItemText(nSel,3);
        dlgPerson.m_strMobile=list.GetItemText(nSel,4);
        dlgPerson.m_strQQ=list.GetItemText(nSel,5);
        dlgPerson.m_strEmail=list.GetItemText(nSel,6);
        dlgPerson.m_strAddress=list.GetItemText(nSel,7);
        if(dlgPerson.DoModal()==IDOK)                      //显示模态对话框，若返回 IDOK
            RefreshList();                                 //刷新列表
    }
}
```

（2）双击 CAdo091View 类下的 OnUpdateMenuitemUpdate 项，添加如下代码：

```
void CAdo091View::OnUpdateMenuitemUpdate(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(bState);                                //若未登录，则不可用
    CListCtrl& list=GetListCtrl();
    int nSel=list.GetNextItem(-1,LVNI_SELECTED);           //获取选中项
    if(nSel==-1)
        pCmdUI->Enable(FALSE);                             //若未选中一项，则不可用
    else
        pCmdUI->Enable();                                  //若有选中项，则可用
}
```

（3）生成程序并运行，登录后，在列表视图选择一项，选择“管理信息”|“更新记录”命令，弹出“编辑个人信息”窗口，如图 12-31 所示。编辑个人信息，单击“确定”按钮后，选中记录的信息被更新。

图 12-31　更新记录

12.3.4　删除记录

（1）双击 CAdo091View 类下的 OnMenuitemDelete 项，添加如下代码：

```
void CAdo091View::OnMenuitemDelete()
{
    CListCtrl& list=GetListCtrl();
    int nSel=list.GetNextItem(-1,LVNI_SELECTED);                    //获取选中项索引
    if(nSel!=-1)
    {
        CString strID=list.GetItemText(nSel,0);                     //获取 id 列的值
        CString strSQL="delete from friends where id="+strID; //delete 语句
        if(theAdo.GetConnectState())
        {
            if(theAdo.ExecuteNotSelSQL(strSQL))                     //执行 SQL 删除语句
            {
                MessageBox("删除成功","提示信息");                    //弹出成功删除信息
                RefreshList();                                      //刷新列表
            }
        }
    }
}
```

（2）双击 CAdo091View 类下的 OnUpdateMenuitemDelete，添加如下代码：

```
void CAdo091View::OnUpdateMenuitemDelete(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(bState);                                         //若未登录，则不可用
    CListCtrl& list=GetListCtrl();
    int nSel=list.GetNextItem(-1,LVNI_SELECTED);
    if(nSel==-1)
        pCmdUI->Enable(FALSE);                                      //若选中一项，则可用
    else
        pCmdUI->Enable();                                           //若未选中一项，则不可用
}
```

（3）生成程序并运行，登录后选择一项，选择“管理信息”|“删除记录”命令，删除选中项，并弹出成功提示信息。

12.3.5　导出记录

（1）双击 CAdo091View 类下的 OnMenuitemExport 项，添加如下代码：

```
void CAdo091View::OnMenuitemExport()
{
    CStdioFile f;
    if(f.Open("D:\\导出的个人信息.txt",CFile::modeCreate|CFile::modeWrite))
                                                                    //创建并写入文件
    {
```

```
        CListCtrl& list=GetListCtrl();
        for(int i=0;i<list.GetItemCount();i++)                    //获取列表视图所有行
        {
            f.WriteString("名称 : "+list.GetItemText(i,1));       //写入各列信息，并换行
            f.WriteString("\n性别 : "+list.GetItemText(i,2));
            f.WriteString("\n生日 : "+list.GetItemText(i,3));
            f.WriteString("\n手机号 : "+list.GetItemText(i,4));
            f.WriteString("\nQQ号 : "+list.GetItemText(i,5));
            f.WriteString("\n邮箱 : "+list.GetItemText(i,6));
            f.WriteString("\n地址 : "+list.GetItemText(i,7));
            f.WriteString("\n\n");                                //两个换行符
        }
        MessageBox("成功导出个人信息到D盘根目录！","提示信息");//成功导出提示信息
    }
}
```

（2）双击 CAdo091View 类下的 OnUpdateMenuitemExport 项，添加如下代码：

```
void CAdo091View::OnUpdateMenuitemExport(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(bState);                                       //若未登录，不可用
}
```

（3）生成程序并运行，登录后，选择“管理信息”|“导出为 txt 文件”命令，在 D 盘根目录生成“导出的个人信息.txt”文件，文件内容如图 12-32 所示。

图 12-32　导出记录

12.4 小结

本章主要介绍数据库编程。对于数据库编程，涉及 ADO、DAO、ODBC 等方式，本章主要介绍了最常用的 ADO 技术。先介绍了数据库的基本知识，如 SQL Server 2000 的安装、企业管理器和查询分析器，并对 SQL 语言进行简单介绍。然后提供一个 ADO 封装类，最后介绍如何借助 ADO 技术连接数据库，并进行数据的查询、添加、修改、删除和导出。

12.5 习题

1．简要说出 SQL 语言的基本语句。

2．数据查询语言包括哪几种？

3．数据更新语言包括哪几种？

4．使用 ADO 技术，如何连接数据库？

5．编写一个程序，创建一个登录窗口。其用户名和密码保存在数据表 uesr 中，每次认证都从数据库中读取进行比较。

第 13 章　DataGrid 控件

在数据库开发中，经常使用数据绑定控件，直接将读取的数据集绑定到控件后，控件自动显示所有数据，无须利用 for 循环逐行添加到控件中。使用数据绑定控件，可以简化程序代码，提高开发效率，在.NET 环境下大部分控件都支持数据绑定，因而做数据库系统开发效率很高。在 Visual C++环境中，常使用 Microsoft 提供的 DataGrid 控件，该控件支持数据绑定，具有强大的数据显示编辑功能。

上一章利用 ADO 封装类访问数据库，简化了开发代码，但也隐藏了 ADO 对象的操作流程，本章直接使用 ADO 的三个对象访问 Excel 表格数据，并利用 DataGrid 控件显示读取的数据。

13.1 添加 DataGrid 控件

【实例 13-1】新建一个对话框工程名为 Ado092，使用 ADO 对象直接读取 Excel 表格数据，利用 DataGrid 控件显示数据，实现添加、删除、计算功能。

（1）新建对话框工程 Ado092，选择 Project|Add To Project|Components and Controls 命令，双击 Registered ActiveX Controls 项，按下 M 键，选择 Microsoft DataGrid Control, Version 6.0 (OLEDB) 项。单击 Insert 按钮，添加 DataGrid 控件，如图 13-1 所示。

（2）单击工具箱的图标，拖放 DataGrid 控件到对话框模板上，并添加两个静态文本、两个编辑框、三个按钮控件到模板上，如图 13-2 所示。

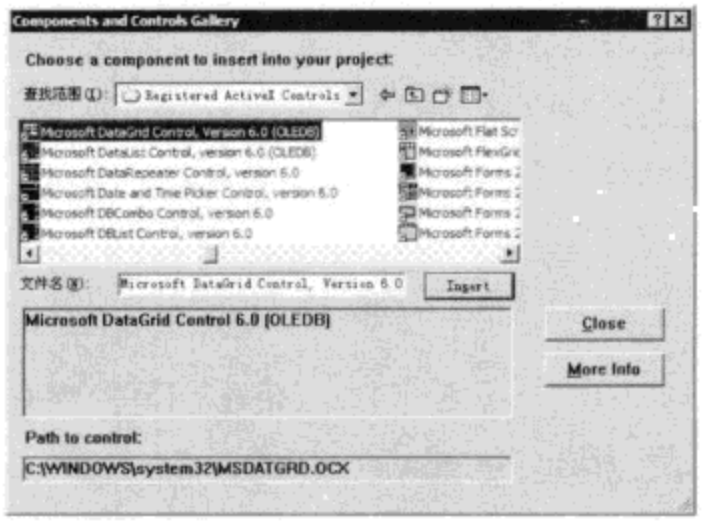


图 13-1　添加注册控件

图 13-2　添加 DataGrid 控件

（3）设置静态文本的 Caption 依次为“姓名”、“评分(逗号分隔)”。设置编辑框的 ID 依次为 IDC_EDIT_NAME、IDC_EDIT_SCORE。设置按钮的 Caption 依次为“添加选手”、“删除选手”、“计算名次”，ID 依次为 IDC_BUTTON_ADD、IDC_BUTTON_DELETE、IDC_BUTTON_CALC。

（4）按 Ctrl+W 组合键打开类向导，选择 Member Variable 选项卡，双击 IDC_DATAGRID1 项，添加 CDataGrid 类型的变量 m_grid，双击 IDC_EDIT_NAME 项，添加 CString 变量 m_strName，双击 IDC_EDIT_SCORE 项，添加 CString 变量 m_strScore。

（5）在类视图右键单击，在弹出的快捷菜单中选择 New Folder 命令，输入 DataGrid，单击 OK 按钮，在类视图添加一个目录，将添加 DataGrid 控件自动生成的 10 个类选中后，拖放到

DataGrid 目录下，以简化界面，如图 13-3 所示。

（6）在 D 盘根目录，新建一个 Excel 工作表，重命名为“分数.xls”。打开文件，将 Sheet1 重命名为“分数表”，并添加 8 个字段，依次为“姓名”、“评分 1”、“评分 2”、“评分 3”、“评分 4”、“评分 5”、“成绩”、“名次”，如图 13-4 所示。

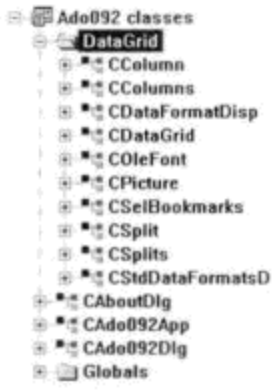


图 13-3 添加目录

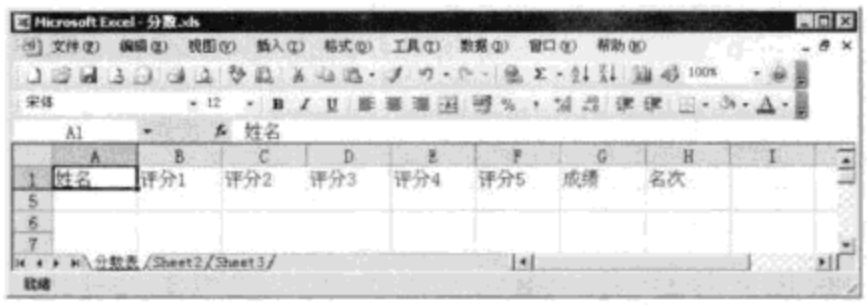


图 13-4 Excel 数据表

13.2 读取 Excel 数据表

实现读取 Excel 数据表的功能，具体步骤如下。

（1）在类视图双击 CAdo092Dlg 项，在类定义前添加如下代码：

```
    #pragma warning(disable:4146)                    //屏蔽警告
    #import "c:\program files\common files\system\ado\msado15.dll" no_namespace rename
("EOF","adoEOF")
```

（2）在类定义中添加三个成员变量，代码如下：

```
public:
    _RecordsetPtr m_pRecordset;          //Recordset 对象的智能指针
    _CommandPtr m_pCommand;              //Command 对象
    _ConnectionPtr m_pConnection;        //Connection 对象
```

（3）右键单击 CAdo092Dlg 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加一个返回类型为 void 的函数 RefreshDataGrid()，添加如下代码：

```
void CAdo092Dlg::RefreshDataGrid()
{
    try
    {
        if(m_pRecordset->State!=adStateClosed)                    //关闭记录集
            m_pRecordset->Close();
        m_pRecordset->Open("SELECT * FROM [分数表$] where 姓名 is not null"
                                                                  //读取表数据,

_variant_t((IDispatch*)m_pConnection,true),adOpenDynamic,adLockPessimistic
            ,adCmdText);
    }
    catch(_com_error e)                                           //若有异常发生
    {
        CString temp;
        temp.Format("打开[分数表$]失败，错误信息:%s",e.ErrorMessage()); //弹出提示信息
        AfxMessageBox(temp);
    }
    m_grid.SetRefDataSource(NULL);                                //先设置数据源为空
    m_grid.SetRefDataSource((LPUNKNOWN)m_pRecordset);             //绑定记录集
    m_grid.SetCaption("分数表");                                  //设置表头名称
    m_grid.SetAllowUpdate(FALSE);                                 //不允许编辑
```

```
    int widths[]={50,40,40,40,40,40,40,40};                    //各列宽度
    _variant_t vIndex;
    for(int i=0;i<sizeof(widths)/sizeof(int);i++)              //遍历所有列，设置列宽
    {
        vIndex=long(i);
        m_grid.GetColumns().GetItem(vIndex).SetWidth(widths[i]);
    }
}
```

State 属性判断记录是否关闭，若未关闭，Close 函数关闭记录集。Open 函数打开指定记录，参数 1 为 SQL 查询语句，读取“分数表”表中的数据，“where 姓名 is not null”限定“姓名”字段不能为空。

Tips　Excel 中的表格名称必须使用“[分数表$]”格式，即表名后添加$符号。

SetRefDataSource 函数设置 DataGrid 绑定的数据源，将读取得到的记录集作为 DataGrid 的数据源，则记录集中的所有数据自动显示在 DataGrid 控件中，无须利用 for 循环逐行添加。SetCaption 函数设置控件的表头名称，SetAllowUpdate 函数设置控件不能编辑更新。

_variant_t 类通常作为函数参数，用于接收不同类型的数据，由于_variant_t 类没有重载 int 类型转换，不能直接将 int 赋给_variant_t 类型变量，可先转换为 long 类型，_variant_t 重载了 long 类型转换。GetColumns 函数获取 DataGrid 的列集合，GetItem 函数获取指定列，SetWidth 函数设置列宽。

Tips　当首次显示数据，或数据发生了变化，如添加、删除，更新记录后，应调用该函数，刷新控件。

（4）双击 CAdo092Dlg 类下的 OnInitDialog 项，在 return TRUE; 前添加如下代码：

```
CoInitialize(NULL);                                            //初始化 COM
m_pConnection.CreateInstance("ADODB.Connection");              //创建 Connection 实例
try
{
  m_pConnection->ConnectionTimeout=8;                          //连接超时时间
  m_pConnection->PutCursorLocation(adUseClient);               //设置游标位置
  _bstr_t strConn="Provider=Microsoft.Jet.OLEDB.4.0;Data Source='D:\\分数.xls';"
    "Extended Properties=\"Excel 8.0;HDR=Yes;IMEX=0\"";        //Excel 表的连接字符串
  m_pConnection->Open(strConn,"","",adModeUnknown);            //连接 Excel 表
}
catch(_com_error e)
{
  AfxMessageBox("无法正常连接 Excel 表格：D:\\分数.xls");
  return FALSE;
}

try
{
  m_pRecordset.CreateInstance("ADODB.Recordset");          //创建 Recordset 实例
  m_pCommand.CreateInstance("ADODB.Command");              //创建 Command 实例
  m_pCommand->ActiveConnection=m_pConnection;          //设置Command关联的Connection对象
}
catch(_com_error e)
{
  return FALSE;
}

RefreshDataGrid();                                             //刷新 DataGrid
```

CoInitialize 函数用于初始化 COM 库，CreateInstance 函数创建 ADO 对象实例，参数为对象的 CLSID 字符串，如 Connection 对象为 ADODB.Connection，Command 对象为 ADODB.Command，也可用__uuidof(Connection)和__uuidof(Command)。

ConnectionTimeout 属性设置数据库连接超时时间，PutCursorLocation 函数设置使用客户区游标。strConn 为 Excel 数据表的连接字符串，其中 Data Source 为表格文件路径，其他为固定格式。Open 函数根据连接字符串连接 Excel 表。

若连接 Excel 表失败，则弹出错误提示信息，并返回 FALSE。若连接成功，则调用 RefreshDataGrid 函数读取表格数据，刷新 DataGrid 控件。

（5）在当前 CPP 文件开头处，包含两个类头文件，代码如下：

```
#include "columns.h"
#include "column.h"
```

（6）生成程序并运行，如图 13-5 所示。

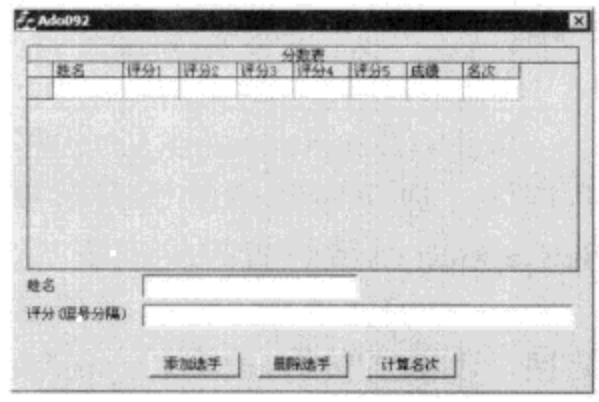


图 13-5 显示 Excel 表格数据

13.3 添加删除数据

（1）右键单击 CAdo092Dlg 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加一个返回类型为 void 的函数 GetScore(CString strScore[])，添加如下代码：

```
void CAdo092Dlg::GetScore(CString strScore[])
{
    int nIndex[4];                                    //4 个分隔符的索引
    int nStart=0,i=0;
    for(;i<4;i++)
    {
        nStart=m_strScore.Find(',',nStart);           //查找 4 个分隔符
        nIndex[i]=nStart;
        nStart++;                                     //查找的起始索引
    }

    strScore[0]=m_strScore.Left(nIndex[0]);           //第 1 个评分
    strScore[1]=m_strScore.Mid(nIndex[0]+1,nIndex[1]-nIndex[0]-1);//第 2、3、4 个评分
    strScore[2]=m_strScore.Mid(nIndex[1]+1,nIndex[2]-nIndex[1]-1);
    strScore[3]=m_strScore.Mid(nIndex[2]+1,nIndex[3]-nIndex[2]-1);
    strScore[4]=m_strScore.Mid(nIndex[3]+1);          //第 5 个评分
}
```

GetScore 函数获取编辑框中一次输入的 5 个评分值，输入时用逗号分隔，如“34,54,65,67,65”，参数 strScore 为 CString 数组，用于存放 5 个评分值。

nIndex 为整型数组，用于存放 4 个分隔符的索引。nStart 为查找的起始索引，Find 函数从起始索引开始查找逗号分隔符，查找到一个后，就存入 nIndex 数组中，并将起始索引加 1。

Left、Mid 函数根据分隔符的索引，分别获取 5 个评分值，存放到 strScore 中。Left 函数的参数为从最左边开始获取的字符数目，Mid 函数的参数 1 为起始索引，参数 2 为要读取的字符数目，若不指定参数 2，则获取所有剩余字符串。

Tips 由于 CString 类未提供拼接字符串的切割函数，只能逐个获取，或自己编写分隔函数。

（2）在资源视图双击 IDD_ADO092_DIALOG 项，打开对话框模板，双击“添加选手”按钮，添加如下代码：

```
void CAdo092Dlg::OnButtonAdd()
```

```
{
    UpdateData();                                                    //更新控件变量
    if(m_strName.IsEmpty() || m_strScore.IsEmpty())                  //若输入为空，则返回
        return;
    int nCount=m_strScore.Replace(",",",");                          //获取分隔符数目
    if(nCount!=4)                                                    //若分隔符数目不等于 4，则返回
    {
        MessageBox("请输入 5 组评分，用逗号分隔","提示信息");
        return;
    }

    CString strScore[5];
    GetScore(strScore);                                              //获取输入的 5 个评分

    CString strSQL="insert into [分数表$](姓名,评分 1,评分 2,评分 3,评分 4,评分 5)"
        " values('"+m_strName+"','"+strScore[0]+","+strScore[1]+","+strScore[2] +","
        +strScore[3]+","+strScore[4]+")";                            //SQL 插入语句
    m_pCommand->CommandText=(_bstr_t)strSQL;
    try
    {
        m_pCommand->Execute(NULL,NULL,adCmdText);                    //执行插入命令
        MessageBox("添加信息成功！","提示信息");
        RefreshDataGrid();                                           //刷新控件
    }
    catch (_com_error& e)
    {
        MessageBox("添加信息失败！","提示信息");
    }
}
```

Replace 函数用于字符串替换，用分隔符替换分隔符，即可得到分隔符的数目，若分隔符数目不为 4，则表明没有输入 5 组评分，弹出错误提示信息，并返回。GetScore 函数获取输入的 5 个评分，存入 strScore 数组中。

strSQL 为 SQL 插入命令的拼接字符串，添加姓名和 5 个评分，其中姓名使用"表明该字段为字符串格式。Execute 函数执行插入命令，若添加成功，弹出成功提示信息，并刷新控件。

（3）双击对话框模板上的“删除选手”按钮，添加如下代码：

```
void CAdo092Dlg::OnButtonDelete()
{
    CColumns cols=m_grid.GetColumns();                               //获取列集合
    CString strName=cols.GetItem(COleVariant(0L)).GetText();//获取选中项的第 1 列的值
    if(MessageBox("确定要删除选手 "+strName+" 吗？","删除提示",//弹出是否删除提示窗口
        MB_OKCANCEL)==IDCANCEL)                                      //若单击“取消”按钮，则返回
        return;

    CString strSQL="update [分数表$] set 姓名='',评分 1='',评分 2='',评分 3='',评分 4='',"
        "评分 5='',成绩='',名次='' where 姓名='"+strName+"'";        //SQL 更新命令
    m_pCommand->CommandText=(_bstr_t)strSQL;
    try
    {
        m_pCommand->Execute(NULL,NULL,adCmdText);                    //执行更新，清空指定记录
        MessageBox("删除选手成功！","提示信息");
        RefreshDataGrid();                                           //刷新控件
    }
    catch (_com_error& e)
    {
        MessageBox("删除选手失败！","提示信息");
    }
}
```

GetColumns 函数获取 DataGrid 的列集合，GetItem 函数获取第 1 列，GetText 函数获取第 1 列选中行的值，即选中项的姓名。MessageBox 函数弹出“删除提示”窗口，用于确认删除操作，避免误删，如图 13-6 所示。

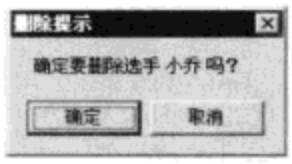


图 13-6 删除提示窗口

其中参数 MB_OKCANCEL 设置窗口的按钮样式，若单击“确定”按钮返回 IDOK，若单击“取消”按钮返回 IDCANCEL。strSQL 为 SQL 更新语句的拼接字符串，where 子句限定更新指定记录，Execute 函数执行更新命令，弹出成功提示窗口，并刷新控件。

Tips Excel 表不支持 delete 语句，只能通过清空记录值，删除记录，读取时只读取不为空的记录。

（4）打开对话框模板，右键单击 DataGrid，在弹出的快捷菜单中选择 Events 命令，添加 SelChange 事件的处理函数，添加如下代码：

```
void CAdo092Dlg::OnSelChangeDatagrid1(short FAR* Cancel)
{
    CColumns cols=m_grid.GetColumns();
    m_strName=cols.GetItem(COleVariant(0L)).GetText();     //获取选中项第 1 列的值
    CString strScore[5];
    m_strScore="";
    for(int i=0;i<5;i++)
    {
        int j=i+1;
        strScore[i]=cols.GetItem(COleVariant((long)j)).GetText(); //获取 5 个评分的值
        if(i==4)
            m_strScore+=strScore[i];                    //拼接为一个字符串，用逗号分隔
        else
            m_strScore+=strScore[i]+",";
    }
    UpdateData(FALSE);                                  //更新控件
}
```

当 DataGrid 选中项改变后，自动调用该函数。GetText 函数获取选中项的姓名，以及 5 个评分值，并将 5 个评分拼接为一个字符串，在“评分”编辑框中显示。

（5）生成程序并运行，如图 13-7 所示。“姓名”编辑框输入名称，“评分”编辑框输入 5 个评分，用逗号分隔，如“34,54,65,76,45”，单击“添加选手”按钮添加记录。选中一项后，在编辑框中显示选中项的值，单击“删除选手”按钮，删除选中项。

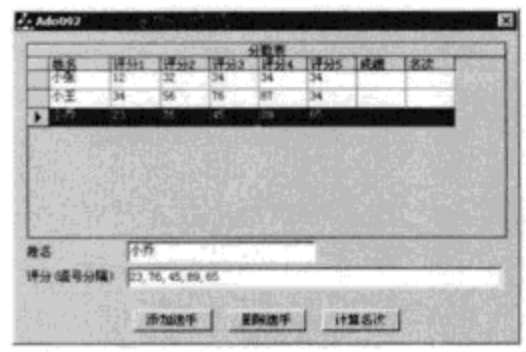


图 13-7 添加删除记录

13.4 计算并更新数据

（1）在资源视图双击 IDD_ADO092_DIALOG 项，打开对话框模板，双击“计算名次”按钮，添加如下代码：

```
void CAdo092Dlg::OnButtonCalc()
{
    int nRowCount=m_grid.GetApproxCount();              //获取行数目
    if(nRowCount<2)                                     //若行数小于 2，则返回
        return;
    int* dResult=new int[nRowCount];                    //动态数组，存放成绩
    int* dOrder=new int[nRowCount];                     //动态数组，存放名次
    int score[5];                                       //5 个评分
```

```
_variant_t varRow,varCol;                                    //行、列索引
for(int nRow=0;nRow<nRowCount;nRow++)                        //遍历所有行
{
    varRow=long(nRow+1);
    for(int nCol=1;nCol<m_grid.GetColumns().GetCount()-2;nCol++)  //遍历评分列
    {
        varCol=long(nCol);
        CColumn col=m_grid.GetColumns().GetItem(varCol);
        score[nCol-1]=atoi(col.CellText(varRow));//获取列值，并转换为整数
    }
    for(int i=0;i<4;i++)                                     //同一个选手的 5 个评分值，排序
    {
        for(int j=i+1;j<5;j++)
        {
            if(score[i]>score[j])
            {
                int nTemp=score[i];
                score[i]=score[j];
                score[j]=nTemp;
            }
        }
    }
    dResult[nRow]=(score[1]+score[2]+score[3])/3;  //计算去掉最大、最小值后的平均值
}
int i=0;
for(;i<nRowCount;i++)                                        //设置所有选手的名次初始为 1
    dOrder[i]=1;

for(i=0;i<nRowCount;i++)                                     //计算所有选手的名次
{
    int k=0;
    for(;k<nRowCount;k++)                                    //与其他所有选手比较
    {
        if(k==i)                                             //若是同一个选手，则不做比较
            continue;
        else if(dResult[i]<dResult[k])                       //若低于其中 1 个选手，则名次值加 1
            dOrder[i]++;
    }
}

for(i=0;i<nRowCount;i++)                                         //遍历所有选手，存入成绩和名次
{
    varRow=long(i+1);
    varCol=long(0);
    CColumn col=m_grid.GetColumns().GetItem(varCol);
    CString strName=col.CellText(varRow);                        //获取当前选手的姓名
    CString strScore,strOrder;
    strScore.Format("%d",dResult[i]);                            //将成绩、名次格式化为字符串
    strOrder.Format("%d",dOrder[i]);
    CString strSQL="update [分数表$] set 成绩="+strScore+",名次="+strOrder
        +" where 姓名='"+strName+"'";                            //更新选手的成绩和名次
    m_pCommand->CommandText=(_bstr_t)strSQL;
    try
    {
        m_pCommand->Execute(NULL,NULL,adCmdText);                //执行 SQL 更新命令
    }
    catch (_com_error& e)
    {
        MessageBox("计算名次失败！","提示信息");
    }
```

```
        }
        delete [] dResult;                                    //释放动态创建的数组
        delete [] dOrder;
        RefreshDataGrid();                                    //刷新控件
    }
```

GetApproxCount 函数获取 DataGrid 的记录行数，dResult、dOrder 为根据行数动态创建的数组，分别用于存放成绩、名次。score 数组存放每个选手的 5 个评分值，外层 for 循环遍历所有选手，内层第 1 个 for 循环用于获取选手的 5 个评分值，内层第 2 个 for 循环用于对评分进行排序。

GetCount 函数获取列总数，CellText 函数获取指定列的指定行的单元格值，atoi 函数将字符串转换为整型值，并存入 score 数组中。排序后，计算去除最大值、最小值后的平均值，存入 dResult 中。

Tips 由于列名也占用一行，CellText 函数的参数应为当前记录的行数加 1。

dOrder 数组元素初始设置为 1，即初始都是第 1 名，利用 for 循环和其他选手比较，若小于某个选手，则名次值递增，如成绩第二的选手初始名次为 1，只比第一名小，只执行一次递增，名次值为 2。

计算所有选手的成绩和名次后，逐个调用 SQL 更新语句，更新 Excel 表中的数据。使用 delete 释放动态创建的数组，RefreshDataGrid 函数刷新控件，显示最新数据。

（2）生成程序并运行，如图 13-8 所示。单击“计算名次”按钮，自动计算所有选手的成绩和名次，并在 DataGrid 控件中显示。

图 13-8　计算成绩和名次

13.5 小结

DataGrid 控件是数据库编程中常用的一个数据绑定控件，可以快速实现数据库软件的开发。本章主要介绍了 DataGrid 控件的添加方法，并结合 Excel 数据表的读取，在 DataGrid 控件中显示数据表的内容，并实现数据的添加删除操作，最后根据已有数据进行相关计算，并更新计算结果。

13.6 习题

1. 简述 DataGrid 控件的重要性。
2. 添加 DataGrid 控件的具体步骤有哪些？
3. 如何读取 DataGrid 控件中的数据？
4. 如何向 DataGrid 控件中添加数据？
5. 如何从 DataGrid 控件中删除数据？

第 14 章　OpenGL 三维编程

随着互联网技术的快速发展，计算机硬件设备的普及大众化，三维网络游戏成为现今软件业的最大热门，网络游戏具有炫丽的界面、丰富的剧情和任务、持续的角色升级、现实与虚拟的结合等特点，深受广大网民的追捧，而且网络游戏依赖游戏中提供的服务获利，无须担心软件的盗版问题，各大网游公司从中获利颇丰。

MFC 库只能实现二维的软件界面，若要进行三维软件开发，需要使用三维软件开发包。目前主要的三维图形库有 OpenGL（Open Graphic Library）和 Microsoft 的 DirectX，其中 OpenGL 最初由美国的 SGI 公司推出，是一个跨平台的功能强大的三维图形库，在各个行业软件都得到广泛应用，如机械、地质、地理信息系统、医学、网络游戏等领域。

DirectX 是 Microsoft 推出的三维开发包，目前也得到了广泛使用，OpenGL 虽三维图形功能强大，但仅限于图形显示，不具备键盘鼠标交互操作、视频、音频、文件等功能，目前网络游戏开发大多使用 DirectX 库。

相对于一般软件开发，三维开发具有一定的难度，需要开发者了解计算机图形学的相关知识，在深刻理解三维图形的基础上，利用三维图形库提供的 API 函数，实现三维图形效果。

14.1 了解 OpenGL

OpenGL 是一个功能非常强大的三维图形库，在各类三维图形软件中得到广泛使用，计算机显示硬件通常将其部分指令固化到硬件中，从而提高执行效率。学习 OpenGL 需要掌握相关的图形学知识，如坐标变换、投影、纹理等内容，建议阅读《OpenGL 编程指南（第四版）》，以了解更多内容。

14.1.1 OpenGL 三维绘图

在使用 GDI 提供的绘图函数时，只需传入 x、y 两个方向的坐标值，而在 OpenGL 三维环境下，每个点有三个坐标值：x、y、z，z 表示点的深度值。默认情况下，窗口中心点的 x、y 值为 0，x 从左往右递增，y 从下往上递增，窗口所在的平面 z 值为 0，z 从内往外递增，即窗口横向为 x 轴，窗口纵向为 y 轴，垂直于窗口为 z 轴。

类似于内存设备环境，OpenGL 先在内存中绘制三维图形，绘制完成再映射到二维窗口中显示。类似于设备环境使用 HDC 句柄作为唯一标识符，OpenGL 使用 HGLRC 句柄作为渲染环境（Render Context）的唯一标识符，可将 HGLRC 看做是连接 OpenGL 三维场景和窗口设备环境的桥梁，通过 HGLRC 将内存中绘制的三维图形映射到窗口 DC 中。

Windows 提供一些函数用于关联 HGLRC 和 HDC，这些函数都以 wgl 开头，表示 Windows 下专用的 OpenGL 函数，常用的几个函数如下。

（1）wglCreateContext 函数根据设备环境创建一个匹配的渲染环境，格式如下：

```
HGLRC wglCreateContext(HDC hdc)
```

参数如下。

- Hdc：窗口设备环境的句柄。

Tips 设备环境和渲染环境必须要使用相同的像素格式（PixelFormat）。

返回值：与 DC 匹配的渲染环境的句柄。

（2）wglMakeCurrent 函数设置当前线程使用的渲染环境，格式如下：

```
BOOL wglMakeCurrent(HDC hdc,HGLRC hglrc)
```

参数如下。

- hdc：图形显示窗口的设备环境句柄。
- hglrc：当前线程所使用的渲染环境。

返回值：若成功则返回 TRUE，否则返回 FALSE。

（3）wglGetCurrentContext 函数获取当前线程使用的渲染环境，格式如下：

```
HGLRC wglGetCurrentContext(VOID)
```

返回值：当前线程使用的 RC，若没有则返回 NULL。

（4）wglGetCurrentDC 函数获取当前线程使用的 RC 所对应的设备环境，格式如下：

```
HDC wglGetCurrentDC(VOID)
```

返回值：当前 RC 所对应的 DC 句柄，若没有 RC 则返回 NULL。

（5）wglDeleteContext 函数释放指定的渲染环境，格式如下：

```
BOOL wglDeleteContext(HGLRC hglrc)
```

参数如下。

- hglrc：要释放的 RC 的句柄。

返回值：若成功则返回 TRUE，否则返回 FALSE。

14.1.2 OpenGL 库文件

Visual C++ 6.0 自带有 OpenGL 的库文件，包括头文件、LIB 文件、DLL 文件，但其版本较低，若要使用高版本的 OpenGL 库，可将三种类型文件分别拷贝到对应位置。

（1）打开 Visual C++安装目录，如 D:\Program Files\Microsoft Visual Studio，再打开 VC98\Include\GL 目录，可发现有三个 OpenGL 的头文件，如图 14-1 所示。

图 14-1 头文件

其中 GL.H 包含 OpenGL 基本的宏定义、函数声明，总共有 200 多个函数，函数名以 gl 开头，包含函数功能名称、参数数目、数据类型、参数类别等信息，以便根据函数名即可知道功能和参数列表，如其中两个函数的定义如下：

```
WINGDIAPI void APIENTRY glVertex3d (GLdouble x, GLdouble y, GLdouble z);
WINGDIAPI void APIENTRY glVertex3dv (const GLdouble *v);
```

第一个函数 glVertex3d 的前缀为 gl，Vertex 表示一个顶点，3 表示有三个参数，d 表示数据类型为双精度浮点数，第二个函数 glVertex3dv 多一个后缀 v，表示参数为向量数组。

GLdouble 是对 double 类型的重定义，由于在不同平台上基本数据类型的长度可能不一致，

因此使用重定义后的类型可以提高代码的移植性，重定义的数据类型如下：

```
typedef unsigned int GLenum;
typedef unsigned char GLboolean;
typedef unsigned int GLbitfield;
typedef signed char GLbyte;
typedef short GLshort;
typedef int GLint;
typedef int GLsizei;
typedef unsigned char GLubyte;
typedef unsigned short GLushort;
typedef unsigned int GLuint;
typedef float GLfloat;
typedef float GLclampf;
typedef double GLdouble;
typedef double GLclampd;
typedef void GLvoid;
```

GLU.H 文件包含 OpenGL 的一些实用（utility）函数，函数名以 glu 开头，提供了许多强大实用的函数，是 OpenGL 库的一部分。

（2）打开 VC98\Lib 目录，找到 GLAUX.LIB、GLU32.LIB、OPENGL32.LIB 这三个 LIB 静态库文件，如图 14-2 所示。

（3）打开 C:\WINDOWS\system32 目录，找到 glu32.dll、opengl32.dll 这两个 DLL 动态链接库文件，如图 14-3 所示。

图 14-2　LIB 文件

图 14-3　DLL 文件

以上是 Visual C++ 6.0 自带的 OpenGL 库文件，若要使用高版本的库文件，可将 H、LIB、DLL 三类文件分别复制到对应的目录下。

14.2 MFC 框架下使用 OpenGL

一般情况可直接在 Win32 环境中使用 API 函数创建桌面窗口，并利用 OpenGL 函数绘制三维图形，但由于使用 API 做为框架需要掌握大量基础函数，难度较大，且不便于交互式操作。在 MFC 框架下调用 OpenGL 函数较为简单，只需添加少量代码就可实现三维效果，且便于实现鼠标、键盘的交互式操作。

14.2.1 创建 MFC 框架

【实例 14-1】新建一个单文档工程名为 MyGL，读取一组坐标数据，利用 OpenGL 函数绘制三维图形，并实现鼠标的交互式浏览。

图 14-4　编辑菜单

（1）新建单文档工程 MyGL，在资源视图双击 Menu 节点下的 IDR_MAINFRAME 项，打开菜单资源，只保留两个菜单项，如图 14-4 所示。

（2）在类视图双击 CMainFrame 类下的 OnCreate 项，在 return 0; 前添加一句代码：

```
ShowControlBar(&m_wndToolBar,FALSE,FALSE);
```

（3）在类视图双击 CMyGLView 项，在类定义前添加如下代码：

```
#include <gl/gl.h>                                          //包含头文件
#include <gl/glaux.h>
#include <gl/glu.h>
#pragma comment(lib,"opengl32.lib")                         //连接库文件
#pragma comment(lib,"glu32.lib")
#pragma comment(lib,"glaux.lib")
```

使用 OpenGL 函数前要包含相关的头文件，如<gl/gl.h>表示 gl 文件夹下的 gl.h 文件。#pragma comment 预编译指令用于设置当前程序的属性，定义如下：

```
#pragma comment(comment-type [,"commentstring"])
```

如 lib 表示设置程序要连接的 LIB 库文件，通过代码方式连接库文件，可提高代码的移植性，当程序在另一台机器上的 Visual C++上编译生成时，无须专门配置开发环境。opengl32.lib 表示库文件的名称，将库文件放到开发环境设置的库目录下，会自动搜索目录下的所有库文件。

（4）在 CMyGLView 类定义中添加如下成员变量：

```
public:
    HGLRC m_hRender;                                        //渲染环境
    float m_rotateX;                                        //x方向旋转量
    float m_rotateY;                                        //y方向旋转量
    CPoint m_ptMouse;                                       //鼠标位置
    BOOL m_bDown;                                           //鼠标左键是否按下
```

14.2.2 使用 OpenGL

（1）在类视图右键单击 CMyGLView，在弹出的快捷菜单中选择 Add Virtual Function 命令，重写 OnInitialUpdate 函数，添加如下代码：

```
void CMyGLView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class
    m_rotateX=0;                                            //旋转值为 0
    m_rotateY=0;
    m_bDown = FALSE;                                        //按下标志为 FALSE
    CDC * pDC=GetDC();                                      //获取视图设备环境
    HDC hDC=pDC->GetSafeHdc();                              //获取 DC 句柄
    PIXELFORMATDESCRIPTOR pfd={                             //设置像素格式
        sizeof(PIXELFORMATDESCRIPTOR),                      //pfd 结构的大小
            1,                                              //版本号
            PFD_DRAW_TO_WINDOW|                             //支持在窗口中绘图
            PFD_SUPPORT_OPENGL|                             //支持 OpenGL
            PFD_DOUBLEBUFFER,                               //双缓存模式
            PFD_TYPE_RGBA,                                  //RGBA 颜色模式
            24,                                             //24 位颜色深度
            0,0,0,0,0,0,                                    //忽略颜色位
            0,0,0,
            0,0,0,0,                                        //忽略累加位
            32,                                             //32 位深度缓存
            0,                                              //无模板缓存
            0,                                              //无辅助缓存
            PFD_MAIN_PLANE,                                 //主平面
            0,0,0,0
    };
    int nPixelFormat=::ChoosePixelFormat(hDC,&pfd);         //选择像素格式
    ::SetPixelFormat(hDC,nPixelFormat,&pfd);                //设置像素格式
```

```
    m_hRender=wglCreateContext(hDC);                    //创建 RC 渲染环境
    ReleaseDC(pDC);                                     //释放 DC
}
```

GetDC 函数获取当前视图客户区的设备环境，GetSafeHdc 函数获取设备环境的句柄值。结构体 PIXELFORMATDESCRIPTOR 用于描述像素格式，参数 1 为结构体的大小，参数 2 为版本号，参数 3 为标志组合，如 PFD_SUPPORT_OPENGL 表示支持 OpenGL，PFD_DOUBLEBUFFER 表示支持双缓存模式，参数 4 为像素类型，PFD_TYPE_RGBA 表示使用 RGBA 红、绿、蓝、透明度模式，参数 5 为颜色位数。

ChoosePixelFormat 函数根据像素格式的描述信息，为指定的设备环境寻找最相近的一个像素格式，格式如下：

```
int ChoosePixelFormat(HDC hdc,CONST PIXELFORMATDESCRIPTOR* ppfd)
```

参数如下。

- hdc：要查找的设备环境。
- ppfd：指向 PIXELFORMATDESCRIPTOR 结构体的指针。

返回值：最匹配的一个像素格式的索引，起始索引为 1。

SetPixelFormat 函数设置指定设备环境使用的像素格式，格式如下：

```
BOOL SetPixelFormat(HDC hdc,int iPixelFormat,CONST PIXELFORMATDESCRIPTOR* ppfd)
```

参数如下。

- hdc：要设置像素格式的设备环境的句柄。
- iPixelFormat：要使用的像素格式的索引。
- ppfd：指向 PIXELFORMATDESCRIPTOR 结构体的指针，包含像素格式的信息。

返回值：若成功则为 TRUE，否则为 FALSE。

wglCreateContext 函数根据当前 DC 创建匹配的渲染环境，ReleaseDC 函数释放获取的客户区 DC。获取视图客户区 DC 对应的 RC 后，OpenGL 绘图函数先在 RC 上绘制三维图形，再将图形映射到二维的 DC 上，从而实现三维效果。

（2）右键单击 CMyGLView，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_SIZE、WM_DESTROY 消息的处理函数。在类视图双击 OnSize 项，添加如下代码：

```
void CMyGLView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Add your message handler code here
    CRect rcClient;
    GetClientRect(&rcClient);                           //获取客户区矩形
    CDC* pDC=GetDC();                                   //获取客户区 DC
    wglMakeCurrent(pDC->m_hDC,m_hRender);               //设置当前 RC
    glViewport(0,0,rcClient.right,rcClient.bottom);     //设置视口大小
    wglMakeCurrent(NULL,NULL);                          //取消当前 RC
    ReleaseDC(pDC);                                     //释放 DC
}
```

当窗口大小改变时，需要重设渲染环境 RC 的视口大小，视口即三维图形映射到的二维窗口的大小。GetClientRect 函数获取客户区矩形大小，GetDC 函数获取客户区 DC，wglMakeCurrent 函数设置客户区 DC 所使用的 RC。

glViewport 函数设置渲染环境的视口大小，格式如下：

```
void glViewport(GLint x,GLint y,GLsizei width,GLsizei height)
```

参数如下。

- x：视口的左上角 x 坐标。
- y：视口的左上角 y 坐标。
- width：视口的宽度。
- height：视口的高度。

（3）在类视图双击 OnDestroy 项，添加如下代码：

```
void CMyGLView::OnDestroy()
{
    CView::OnDestroy();

    // TODO: Add your message handler code here
    wglMakeCurrent(NULL,NULL);                          //取消当前 RC 和 DC
    if(m_hRender)
        wglDeleteContext(m_hRender);                    //释放 RC
    m_hRender=NULL;                                     //设置 RC 句柄为空
}
```

当关闭窗口，退出程序时，需要释放创建的渲染环境。wglMakeCurrent 函数取消当前使用的 RC，若当前 RC 可用，则 wglDeleteContext 函数释放创建的 RC，并设置 RC 句柄为空。

14.2.3 读取坐标文件数据

（1）在工程文件所在目录下，新建一个 TXT 文本文件，重命名为“测试点数据.txt”，打开文件，输入一组数据，内容如下：

```
13,52,2
43,23,34
14,23,-35
33,98,45
72,94,23
```

一行有三个数值，用逗号分隔，分别表示点的 x、y、z 坐标值，每一行表示一个点。

（2）在类视图双击 CMyGLDoc 项，在 CMyGLDoc 类定义前添加如下代码：

```
#include <vector>                                       //包含 vector 类的头文件
struct ZPoint                                           //三维点结构体
{
    float x;                                            //x、y、z 坐标值
    float y;
    float z;
};
```

（3）在 CMyGLDoc 类定义中，添加一个成员变量，代码如下：

```
public:
    std::vector<ZPoint> m_Zpts;                         //点数组
```

（4）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，Class name 组合框选择 CMyGLDoc 项，添加 ID_FILE_OPEN 的 COMMAND 消息处理函数，添加如下代码：

```
void CMyGLDoc::OnFileOpen()
{
    CFileDialog dlgFile(TRUE,"txt",NULL,NULL,"文本文件 (*.txt)|*.txt||");
                                                        //打开文件对话框
    if(dlgFile.DoModal()==IDOK)                         //显示模态对话框
    {
        CString strPath=dlgFile.GetPathName();          //获取文件路径
        CStdioFile f(strPath,CFile::modeRead);          //读取文件
        CString strLine;
        CString p1,p2,p3;
        m_Zpts.clear();                                 //清空点数组
```

```
        while(f.ReadString(strLine))                                    //读取所有行
        {
            int nIndex1=strLine.Find(',');                              //第 1 个分隔符的索引
            int nIndex2=strLine.ReverseFind(',');                       //第 2 个分隔符的索引
            p1=strLine.Left(nIndex1);                                   //获取 3 个坐标值
            p2=strLine.Mid(nIndex1+1,nIndex2-nIndex1-1);
            p3=strLine.Mid(nIndex2+1);
            ZPoint zpt;
            zpt.x=atof(p1);                                             //将字符串转为浮点数
            zpt.y=atof(p2);                                             //存入点对象中
            zpt.z=atof(p3);
            m_Zpts.push_back(zpt);                                      //点对象添加到动态数组中
        }
    }
}
```

ReadString 函数读取文本文件的每一行的数据，存入 strLine 中。先获取两个分隔符的索引，再分别获取 x、y、z 三个坐标值，调用 atof 函数转为浮点数，存入点对象中，再将每一行所代表的点存入动态数组中。

14.2.4　绘制三维图形

（1）在类视图双击 CMyGLView 类下的 OnDraw 项，添加如下代码：

```
void CMyGLView::OnDraw(CDC* pDC)
{
    CMyGLDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    wglMakeCurrent(pDC->m_hDC,m_hRender);                               //设置当前 RC
    glClearColor(0.0,0.0,0.0,0.0);                                      //设置清除颜色缓冲的颜色
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);                   //清除缓冲位
    CRect rect;
    GetClientRect(&rect);                                               //获取客户区矩形
    int height=rect.Height();                                           //客户区高度
    int width=rect.Width();                                             //客户区宽度
    glMatrixMode(GL_PROJECTION);                                        //设置当前矩阵
    glLoadIdentity();                                                   //初始化矩阵
    glOrtho(-1,1,(-1)*height/width,height/width,-1,1);                  //设置投影参数
    glRotated(m_rotateX, 1.0, 0.0, 0.0);                                //绕 x 轴旋转图形
    glRotated(m_rotateY, 0.0, 1.0, 0.0);                                //绕 y 轴旋转图形
    float fx1,fy1,fz1;                                                  //三角形三个点的坐标
    float fx2,fy2,fz2;
    float fx3,fy3,fz3;
    glBegin(GL_TRIANGLES);                                              //开始绘制三角形
    int nZPtCount=pDoc->m_Zpts.size();                                  //文件中点数目
    float fRatio=100.0f;                                                //坐标值缩小倍数
    for(int i=0;i<nZPtCount;i++)                                        //遍历所有点
    {
        if(i>nZPtCount-3)                                               //若到倒数第 2 个点，停止循环
            break;
        fx1=pDoc->m_Zpts[i].x/fRatio;                                   //三角形第 1 个点的坐标值
        fy1=pDoc->m_Zpts[i].y/fRatio;
        fz1=pDoc->m_Zpts[i].z/fRatio;
        fx2=pDoc->m_Zpts[i+1].x/fRatio;                                 //第 2 点的坐标值
        fy2=pDoc->m_Zpts[i+1].y/fRatio;
        fz2=pDoc->m_Zpts[i+1].z/fRatio;
        fx3=pDoc->m_Zpts[i+2].x/fRatio;                                 //第 3 个点的坐标值
        fy3=pDoc->m_Zpts[i+2].y/fRatio;
        fz3=pDoc->m_Zpts[i+2].z/fRatio;
        glColor3f(1.0f,0.0f,0.0f);                                      //设置颜色 1
        glVertex3f(fx1,fy1,fz1);                                        //绘制三维点 1
        glColor3f(0.0f,1.0f,0.0f);                                      //设置颜色 2
        glVertex3f(fx2,fy2,fz2);                                        //绘制三维点 2
        glColor3f(0.0f,0.0f,1.0f);                                      //设置颜色 3
        glVertex3f(fx3,fy3,fz3);                                        //绘制三维点 3
```

```
    }
    glEnd();                                                    //结束绘制三角形
    glBegin(GL_LINES);                                          //开始绘制直线
        glColor3f(0.0f,1.0f,1.0f);                              //设置颜色
        glVertex3f(0.0f,0.0f,0.0f);                             //绘制三维点
        glColor3f(1.0f,1.0f,0.0f);
        glVertex3f(1.0f,1.0f,1.0f);
        glColor3f(0.0f,1.0f,1.0f);                              //绘制第 2 条直线
        glVertex3f(0.0f,0.0f,0.0f);
        glColor3f(1.0f,0.0f,1.0f);
        glVertex3f(-1.0f,-1.0f,1.0f);
        glColor3f(0.0f,1.0f,1.0f);                              //绘制第 3 条直线
        glVertex3f(0.0f,0.0f,0.0f);
        glColor3f(1.0f,0.0f,1.0f);
        glVertex3f(1.0f,-1.0f,1.0f);
        glColor3f(0.0f,1.0f,1.0f);                              //绘制第 4 条直线
        glVertex3f(0.0f,0.0f,0.0f);
        glColor3f(1.0f,0.0f,1.0f);
        glVertex3f(-1.0f,1.0f,1.0f);
    glEnd();                                                    //结束绘制直线
    SwapBuffers(pDC->m_hDC);                                    //交换缓存，显示图形
    wglMakeCurrent(pDC->m_hDC,NULL);                            //取消当前 RC
}
```

wglMakeCurrent 函数设置客户区 DC 所使用的 RC，glClearColor 函数设置清除颜色缓冲区的颜色值，格式如下：

```
void glClearColor(GLclampf red,GLclampf green,GLclampf blue,GLclampf alpha)
```

参数如下。

- red：颜色值的红色分量。
- green：颜色值的绿色分量。
- blue：颜色值的蓝色分量。
- alpha：颜色值的透明度，4 个参数的值都在 0～1 之间。

glClear 函数清除缓冲区数据到预设值，格式如下：

```
void glClear(GLbitfield mask)
```

参数如下。

- mask：要清除的缓冲区组合，如 GL_COLOR_BUFFER_BIT 表示颜色缓冲区，GL_DEPTH_BUFFER_BIT 表示深度缓冲区。

glMatrixMode 函数设置当前矩阵类型，格式如下：

```
void glMatrixMode(GLenum mode)
```

参数如下。

- mode：矩阵类型，如 GL_MODELVIEW 表示模型矩阵，GL_PROJECTION 表示投影矩阵，GL_TEXTURE 表示纹理矩阵。

Tips 模型矩阵相当于平行视图，以平行的视线观看所有物体，无论远近，看到的大小都保持一致。投影矩阵相当于透视图，这符合人的观察习惯，同样大小的物体，离得越远看起来越小。

glLoadIdentity 函数初始化当前矩阵为单位矩阵，即矩阵从左上角到右下角连线所经过位置值为 1，其余位置的值均为 0，格式如下：

```
void glLoadIdentity(void)
```

glOrtho 函数设置透视矩阵的 x、y、z 三个方向的值范围，格式如下：

```
void glOrtho(GLdouble left,GLdouble right,GLdouble bottom,GLdouble top,GLdouble
```

```
zNear,GLdouble zFar)
```

参数如下。

- left：左边界值。
- right：右边界值。
- bottom：底边界值。
- top：顶边界值。
- zNear：最近的 z 值。
- zFar：最远的 z 值。

左右边界值设为-1、1，表示 x 坐标的显示范围为-1~1，超出这个范围的值将不被显示。同理，y 坐标的范围(-1)*height/width 至 height/width，z 坐标的范围的为-1~1。

glRotated 函数用旋转矩阵乘以当前矩阵，实现图形的旋转，格式如下：

```
void glRotated(GLdouble angle,GLdouble x,GLdouble y,GLdouble z)
```

参数如下。

- angle：旋转的角度。
- x：旋转点的 x 坐标。
- y：旋转点的 y 坐标。
- z：旋转点的 z 坐标。

angle 表示图形旋转的角度，x、y、z 表示旋转点的坐标值，从起点到旋转点的连线为旋转轴，图形绕轴旋转指定角度。如(m_rotateX, 1.0, 0.0, 0.0)表示绕 x 轴旋转 m_rotateX。

glBegin 函数指定开始绘制的图形类型，格式如下：

```
void glBegin(GLenum mode)
```

参数如下。

- mode：图形类型，如 GL_POINTS 表示点，GL_LINES 表示线，GL_TRIANGLES 表示三角形，GL_QUADS 表示四边形。

Tips　所有绘制图形的函数都必须放到 glBegin 和 glEnd 之间：若绘点，每一个顶点单独绘制；若绘线，则每一对顶点绘制一条直线；若绘三角形，则每三个点绘制一个；若绘四边形，则每四个点绘制一个。

pDoc->m_Zpts.size 获取读取的点数目，利用 for 循环遍历所有点。这里采用相邻的三个点绘制一个三角形，当执行到倒数第 2 个点时，由于不够三个点，停止循环。fRatio 为缩小倍数，由于透视矩阵的各个方向的坐标范围为-1~1，因此所有点坐标的值都应缩小到该范围里。

glColor3f 函数设置当前使用的颜色值，格式如下：

```
void glColor3f(GLfloat red,GLfloat green,GLfloat blue)
```

参数如下。

- red：红色分量。
- green：绿色分量。
- blue：蓝色分量。

glVertex3f 函数指定一个顶点的坐标值，格式如下：

```
void glVertex3f(GLfloat x,GLfloat y,GLfloat z)
```

参数如下。

❑ x,y,z：点的坐标值。

glEnd 函数结束当前三角形的绘制。再次调用 glBegin 函数开始绘制直线，其中每两个顶点绘制一条直线。SwapBuffers 函数交换渲染环境和设备环境的缓冲值，将 RC 中的图形映射到 DC。wglMakeCurrent 函数清除当前使用的 RC，以便于其他 DC 使用该 RC。

（2）生成并运行程序，如图 14-5 所示。客户区显示绘制的 4 条三维直线，起点均为窗口中心点，终点分别为 4 个边角，根据起止点的颜色绘制渐变直线。

（3）选择“文件”|“打开”命令，读取“测试点数据”文件中的坐标值，绘制几个连续的三角形，如图 14-6 所示。

图 14-5 三维环境下的直线

图 14-6 读取文件数据，绘制三角形

14.2.5 鼠标交互式浏览

（1）右键单击 CMyGLView，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_LBUTTONDOWN、WM_MOUSEMOVE、WM_LBUTTONUP 消息的处理函数。

（2）在类视图双击 CMyGLView 类下的 OnLButtonDown 项，添加如下代码：

```
void CMyGLView::OnLButtonDown(UINT nFlags, CPoint point)
{
    m_bDown = TRUE;                                    //左键按下标志
    m_ptMouse = point;                                 //记录鼠标位置

    CView::OnLButtonDown(nFlags, point);
}
```

（3）双击 OnMouseMove 项，添加如下代码：

```
void CMyGLView::OnMouseMove(UINT nFlags, CPoint point)
{
    if(m_bDown)                                                    //若左键已按下
    {
        m_rotateY -= m_ptMouse.x-point.x;                          //绕 y 轴旋转角度
        m_rotateX -= m_ptMouse.y-point.y;                          //绕 x 轴旋转角度
        m_ptMouse = point;                                         //鼠标当前位置

        InvalidateRect(NULL,FALSE);                                //重绘视图窗口
    }
    CView::OnMouseMove(nFlags, point);
}
```

若已按下鼠标左键，且移动鼠标位置，则根据鼠标 x、y 方向的偏移量，计算旋转角度。如鼠标左右移动，表示要绕 y 轴旋转，根据 x 方向的偏移量修改旋转角度。InvalidateRect 函数重

绘视图窗口。

（4）双击 OnLButtonUp 项，添加如下代码：

```
void CMyGLView::OnLButtonUp(UINT nFlags, CPoint point)
{
    m_bDown = FALSE;                                    //按下标志为 FALSE

    CView::OnLButtonUp(nFlags, point);
}
```

（5）生成程序并运行，读取文件，用鼠标拖曳图形，实现交互式浏览，如图 14-7 所示。

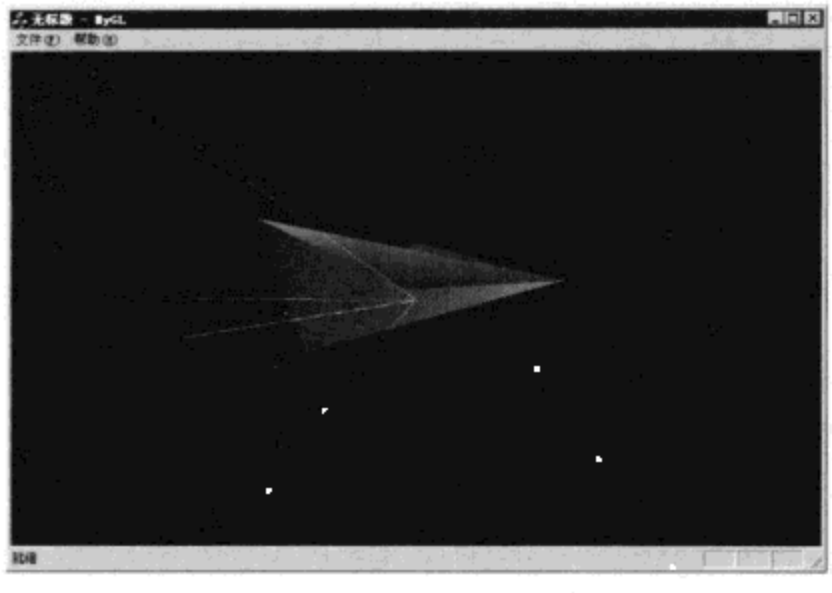


图 14-7　交互式浏览

14.3 小结

OpenGL 是开发三维软件时常用的一个图形库，具有高效、跨平台的特点，在各个行业中都得到广泛应用，通过 OpenGL 可以在程序中嵌入三维图形显示、浏览功能，提高程序的丰富性。本章主要介绍了 OpenGL 的组成、常用函数，以及如何在 MFC 框架下使用 OpenGL 库，并实现简单的三维图形显示、交互式浏览功能。

14.4 习题

1. OpenGL 图形库，具有什么特点。
2. 在 Visual C++中，OpenGL 三维编程常用的绘图函数有哪些？
3. 如何在 MFC 框架下使用 OpenGL 库？
4. 编写一个程序，实现交互式浏览功能。

第5篇　案例篇

第15章　五子棋游戏

五子棋是一类益智游戏，双方轮流下子，若有一方在横竖斜其中一个方向上有 5 个连续的棋子，则该方获胜。常见的五子棋游戏主要为人机对弈和人人对弈，人机对弈需要人工智能方面的知识，人人对弈需要利用网络传递信息，人工智能和网络编程都较为复杂，需要专门系统的学习，为减少开发难度，方便读者了解基本原理，本实例采用单机版的人人对弈模式，实现一个简单的五子棋游戏，读者可在该基础上扩展功能，实现人机对弈或网络版的人人对弈。

15.1 界面设计

【实例 15-1】新建一个对话框工程名为 MyChess，实现一个简单的五子棋游戏。

（1）在资源视图右键单击，在弹出的快捷菜单中选择 Insert 命令，弹出 Insert Resource 窗口，选择 Menu 项，单击 New 按钮，添加一个菜单资源，如图 15-1 所示。

（2）添加“玩家设置(&D)”菜单项，并设置 ID 为 ID_PLAYER_NAME。

（3）双击 IDD_MYCHESS_DIALOG，打开对话框模板资源，单击右键，在弹出的快捷菜单中选择 Properties 命令，设置 Caption 为“五子棋 小游戏”，Menu 组合框选择 IDR_MENU1 项，如图 15-2 所示。

图 15-1　添加菜单资源

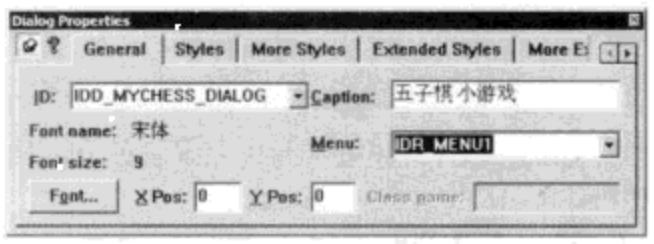


图 15-2　对话框窗口使用菜单

（4）右键单击 Dialog 节点，在弹出的快捷菜单中选择 Insert Dialog 命令，添加一个对话框模板，如图 15-3 所示。

（5）拖放两个静态文本、两个编辑框控件到模板上，设置静态文本的 Caption 依次为“玩家 A 姓名”、“玩家 B 姓名”。设置编辑框的 ID 依次为 IDC_EDIT_A、IDC_EDIT_B。设置 OK 按钮的 Caption 为“确定”，Cancel 按钮的 Caption 为“取消”。

（6）右键单击对话框模板，在弹出的快捷菜单中选择 Properties 命令，设置 ID 为 IDD_PLAYERNAME，Caption 为“输入玩家姓名”，字体使用“宋体”、9 号。

（7）按 Ctrl+W 组合键为对话框模板 IDD_PLAYERNAME 添加类 CPlayerName，基类为 CDialog。

（8）在资源视图右键单击，在弹出的快捷菜单中选择 Insert 命令，选择 Cursor 节点下的 IDC_POINTER 项，单击 New 按钮，添加一个光标资源，如图 15-4 所示。用同样方式，再添加一个光标资源。

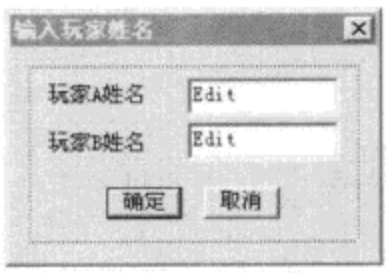


图 15-3　输入对话框

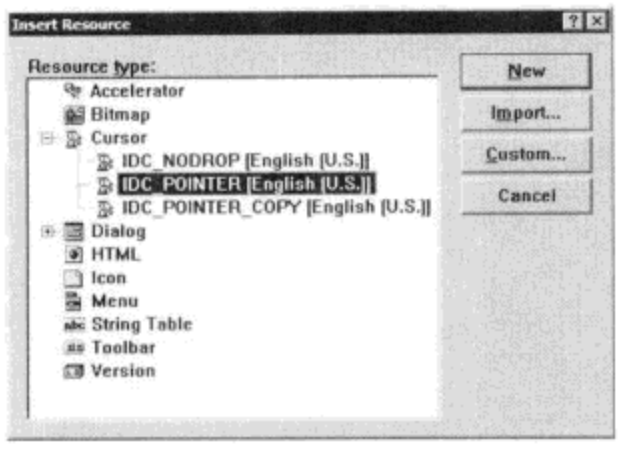


图 15-4　添加光标资源

（9）编辑光标资源，单击A工具，添加 A 和 B 两个字符，如图 15-5 和图 15-6 所示。

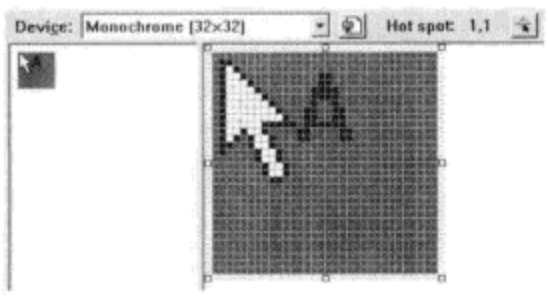


图 15-5　光标资源 1

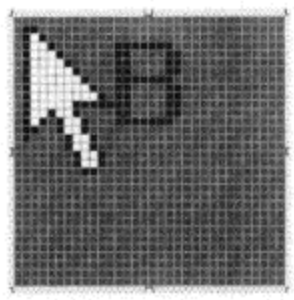


图 15-6　光标资源 2

Tips　单击 按钮，在编辑窗口中单击，可设置光标的热点位置，即光标哪个位置表示精确的单击热点。如 1,1 表示光标箭头位置为单击热点。

15.2 算法设计

本实例在对话框上绘制棋盘，分为 30 行 30 列，使用一个二维数组记录各个位置的值，若尚未下子则值为 0，若 A 已下子则值为 1，若 B 已下子则值为 2。每下一个棋子后，应判断是否有一方已经获胜，若获胜则弹出提示信息，并重开一局。

每种游戏都有其规则，五子棋的规则就是若有一方在横向、竖向、45 度倾斜方向有 5 个连续的棋子，则该方获胜。获取方是在下了最后一个棋子后胜出的，因此 5 个连续的棋子中肯定有 1 个是最后一个棋子，只需判断最后一个棋子的 8 个方向上是否有 5 个连续的棋子，若其中一个方向上有，则胜出，棋子要判断的方向如图 15-7 所示。

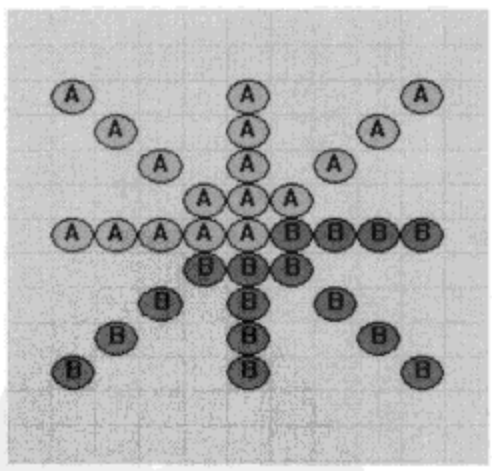


图 15-7　棋子方向

最中心的棋子 A 为最后一个棋子，可能位于 5 个相连棋子的两端，也可能位于中间，需要遍历图中标记为 A 的所有棋子，如水平方向上的 5 个 A，从最左边的第一个 A，到中间的第五个 A，依次查看其右边的 4 个棋子是否相同，若相同，则 A 获胜，否则继续查看左上方的 5 个 A、垂直方向的 5 个 A、右上方的 5 个 A。

在类视图右键单击 CMyChessDlg 项，在弹出的快捷菜单中选择 Add Member Function 命令，添加返回类型为 int 的函数 Beat(int xPos,int yPos,int curPerson)，添加如下代码：

```
int CMyChessDlg::Beat(int xPos, int yPos,int curPerson)
{
```

```
int nX,nY,i,j,k;
BOOL bFlag=TRUE;                                    //标识符，是否胜出
for(i=4;i>=0;i--)                                   //遍历水平方向上的 5 个棋子
{
    nX=xPos-i;                                      //起始棋子的 x、y 位置
    nY=yPos;
    if(nX>=0)
    {
        bFlag=TRUE;                                 //初始为 TRUE
        for(j=nX;j<nX+5;j++)                        //查看起始棋子右边的 4 个棋子
        {
            if(j>29 || hasChess[j][nY]!=curPerson)
                                                    //若棋子位置超出边界，或不是相同的棋子
            {
                bFlag=FALSE;                        //设为 FALSE，退出循环
                break;
            }
        }
        if(bFlag)                                   //若为 TRUE，则表明有 5 个连续的相同棋子
            return 1;
    }
}
for(i=4;i>=0;i--)                                   //遍历左上方的 5 个棋子
{
    nX=xPos-i;
    nY=yPos-i;
    if(nX>=0 && nY>=0)
    {
        bFlag=TRUE;
        for(j=nX,k=nY;j<nX+5;j++,k++)//查看起始棋子右下方的 4 个棋子
        {
            if(j>29 || k>29 || hasChess[j][k]!=curPerson)
            {
                bFlag=FALSE;
                break;
            }
        }
        if(bFlag)
            return 1;
    }
}
for(i=4;i>=0;i--)                                   //遍历垂直方向上的 5 个棋子
{
    nX=xPos;
    nY=yPos-i;
    if(nY>=0)
    {
        bFlag=TRUE;
        for(k=nY;k<nY+5;k++)                        //查看起始棋子下方的 4 个棋子
        {
            if(k>29 || hasChess[nX][k]!=curPerson)
            {
                bFlag=FALSE;
                break;
            }
        }
        if(bFlag)
            return 1;
    }
}
```

```
        for(i=4;i>=0;i--)                                    //遍历右上方的 5 个棋子
        {
            nX=xPos+i;
            nY=yPos-i;
            if(nX<=29 && nY>=0)
            {
                bFlag=TRUE;
                for(j=nX,k=nY;j>nX-5;j--,k++)                //查看起始棋子左下方的 4 个棋子
                {
                    if(j<0 || k>29 || hasChess[j][k]!=curPerson)
                    {
                        bFlag=FALSE;
                        break;
                    }
                }
                if(bFlag)
                    return 1;
            }
        }
        return 0;                                            //若 4 个方向上都没有 5 个连续的棋子，未获胜
    }
```

参数 1、2 为当前棋子的位置，参数 3 为当前棋子的值，若当前棋子获胜，返回 1，否则返回 0。nX、nY 为要判断的起始棋子的位置。bFlag 为获胜状态，在判断前初始为 TRUE，若棋子位置超出边界，或棋子值不一致，设为 FALSE 并退出 for 循环。

15.3 功能实现

五子棋程序的流程如下。

（1）调整窗口的边界大小。

（2）加载光标资源，创建画刷，初始化棋子数组。

（3）在客户区利用绘线函数，绘制棋盘。

（4）输入对弈双方的名称。

（5）鼠标单击棋盘，在指定位置放入棋子，并设置光标样式。

（6）判断当前下棋者是否已获胜，若获胜则弹出提示信息，并重开一局。

具体实现过程如下：

（1）在类视图双击 CMyChessDlg 项，添加如下成员变量：

```
public:
    CString name_b;              //玩家 B 的姓名
    CString name_a;              //玩家 A 的姓名
    HCURSOR hcur2;               //玩家 B 使用的光标
    HCURSOR hcur1;               //玩家 A 使用的光标
    CBrush br1;                  //A 使用的画刷
    CBrush br2;                  //B 使用的画刷
    int person;                  //当前下棋者
    int ystep;                   //棋盘格子的高度
    int xstep;                   //棋盘格式的宽度
    int hasChess[30][30];        //二维数组，记录各个位置的棋子值
```

（2）双击 CMyChessDlg 类下的 OnInitDialog 项，添加如下代码：

```
BOOL CMyChessDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    // Add "About..." menu item to system menu.
    // IDM_ABOUTBOX must be in the system command range.
```

```
ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
ASSERT(IDM_ABOUTBOX < 0xF000);
CMenu* pSysMenu = GetSystemMenu(FALSE);                    //获取系统菜单指针
if (pSysMenu != NULL)
{
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);                 //加载字符串资源
    if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);                //添加菜单项
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
}
// Set the icon for this dialog.  The framework does this automatically
//  when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);                // Set big icon
SetIcon(m_hIcon, FALSE);               // Set small icon
// TODO: Add extra initialization here
CRect rcClient;
CRect rcWindow;
GetWindowRect(rcWindow);                                   //获取窗口矩形
GetClientRect(rcClient);                                   //获取客户区矩形
int nBorderW=rcWindow.Width()-rcClient.Width();            //计算边框的宽度
int nBorderH=rcWindow.Height()-rcClient.Width();           //计算边框的高度
MoveWindow(80,80,750+nBorderW,750+nBorderH);               //设置窗口的大小
hcur1=AfxGetApp()->LoadCursor(IDC_CURSOR1);                //加载光标资源
hcur2=AfxGetApp()->LoadCursor(IDC_CURSOR2);
br1.CreateSolidBrush(RGB(192,192,0));                     //创建画刷
br2.CreateSolidBrush(RGB(0,192,192));
person=0;                                                  //下棋者初始为 0
for(int i=0;i<30;i++)
    for(int j=0;j<30;j++)
        hasChess[i][j]=0;                                  //数组元素初始为 0
return TRUE;  // return TRUE  unless you set the focus to a control
}
```

GetWindowRect 函数获取对话框窗口的矩形边界，GetClientRect 函数获取对话框客户区的矩形边界。客户区用于绘制棋盘，应先计算边框及菜单、标题栏占用的尺寸，再用期望的客户区大小加上边框尺寸，得到整个窗口的尺寸。MoveWindow 函数设置对话框窗口的大小和位置。

LoadCursor 函数用于加载光标资源，CreateSolidBrush 函数创建单色画刷。初始 person 为 0，person 指明当前下棋者，如玩家 A 下棋，person 值为 1，玩家 B 下棋，person 值为 2。hasChess 数组存放棋盘上各个位置的值，若没有下棋，则值为 0，若玩家 A 已下子，则值为 1，若玩家 B 已下子，则值为 2。

（3）双击 CMyChessDlg 类下的 OnPaint 项，添加如下代码：

```
void CMyChessDlg::OnPaint()
{
    CPaintDC dc(this);                                     //客户区窗口的设置环境
    if (IsIconic())                                        //若被最小化
    {
        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;
        // Draw the icon
```

```
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }

    CRect rc;
    GetClientRect(rc);                                          //客户区窗口的矩形边界
    xstep=rc.Width()/30;                                        //棋格的宽度
    ystep=rc.Height()/30;                                       //棋格的高度
    CPen penSolid(PS_SOLID,1,RGB(192,192,192));                 //实线画笔
    CPen* pOldPen=dc.SelectObject(&penSolid);                   //新画笔选入设备环境
    CBrush* pOldBrush=dc.SelectObject(&br1);                    //新画刷选入设备环境
    dc.SetBkMode(TRANSPARENT);                                  //文本设为透明
    for(int i=0;i<=rc.Width();i+=xstep)                         //绘制竖线
    {
        dc.MoveTo(i,0);
        dc.LineTo(i,rc.Height());
    }
    for(int j=0;j<=rc.Height();j+=ystep)                        //绘制水平线
    {
        dc.MoveTo(0,j);
        dc.LineTo(rc.Width(),j);
    }
    for(i=0;i<30;i++)                                           //绘制棋子
    for(j=0;j<30;j++)
    {
        if(hasChess[i][j]!=0)                                   //若棋子值不为 0
        {
            CRect rcChess(i*xstep,j*ystep,(i+1)*xstep,(j+1)*ystep);//棋格的矩形边界
            if(hasChess[i][j]==1)                               //若为玩家 A 的棋子
            {
                dc.SelectObject(&br1);                          //使用画刷 1
                dc.Ellipse(rcChess);                            //绘制椭圆
                dc.TextOut(rcChess.left+xstep/3*1,rcChess.top,"A");   //绘制文本 A
            }
            else if(hasChess[i][j]==2)                          //若为玩家 B 的棋子
            {
                dc.SelectObject(&br2);                          //使用画刷 2
                dc.Ellipse(rcChess);                            //绘制椭圆
                dc.TextOut(rcChess.left+xstep/3*1,rcChess.top,"B");   //绘制文本 B
            }
        }
    }
    dc.SelectObject(pOldPen);                                   //恢复原有画笔、画刷
    dc.SelectObject(pOldBrush);
}
```

CPaintDC dc(this)获取客户区窗口的设备环境，xstep、ystep 为棋格的宽度和高度值。前两个 for 循环用于绘制棋盘，第 1 个 for 循环绘制竖线，第 2 个 for 循环绘制横线，交织构成棋盘。

后两个 for 循环用于绘制棋子，遍历所有棋格，若已下子，根据棋子所属的玩家，使用不同的颜色绘制椭圆，并显示不同的字符。绘制完成后，恢复原有的画笔、画刷。

（4）双击 CPlayerName 项，添加两个成员变量：

```
public:
    CString nameB;                                              //玩家 B 的名称
    CString nameA;                                              //玩家 A 的名称
```

（5）在资源视图双击 IDD_PLAYERNAME 项，双击“确定”按钮，添加如下代码：

```
void CPlayerName::OnOK()
{
    GetDlgItem(IDC_EDIT_A)->GetWindowText(nameA);          //获取输入的名称
    GetDlgItem(IDC_EDIT_B)->GetWindowText(nameB);
    if(nameA.IsEmpty() || nameB.IsEmpty())                 //若输入为空，返回
        return;

    CDialog::OnOK();
}
```

图 15-8　菜单项添加消息函数

（6）按 Ctrl+W 组合键打开类向导窗口，选择 Message Maps 选项卡，为菜单项 ID_PLAYER_NAME 添加 CMyChessDlg 类的 COMMAND、UPDATE_COMMAND_UI 消息的两个处理函数，如图 15-8 所示。

（7）在类视图双击 CMyChessDlg 类下的 OnPlayerName 项，添加如下代码：

```
void CMyChessDlg::OnPlayerName()
{
    CPlayerName dlg;
    int result=dlg.DoModal();//显示模态对话框
            //若返回 IDOK

            //获取输入的玩家名称


            //获取已有的窗口标题
        SetWindowText(title+"  玩家 "+name_a+" vs "+name_b);    //设置新的标题
        person=1;                                  //下棋者为玩家 A
    }
}
```

```
    if(result==IDOK)
    {
        name_a=dlg.nameA;
        name_b=dlg.nameB;
        CString title;
        GetWindowText(title);
```

（8）在当前 CPP 文件开头处，添加一句#include "PlayerName.h" ，包含 CPlayerName 类的头文件。

（9）双击 CMyChessDlg 类下的 OnUpdatePlayerName 项，添加如下代码：

```
void CMyChessDlg::OnUpdatePlayerName(CCmdUI* pCmdUI)
{
    if(person==0)
        pCmdUI->Enable();                          //若尚未输入玩家姓名，可用
    else
        pCmdUI->Enable(FALSE);                     //若已输入，不可用
}
```

（10）在类视图右键单击 CMyChessDlg 项，在弹出的快捷菜单中选择 Add Windows Message Handler 命令，添加 WM_SETCURSOR、WM_LBUTTONDOWN 两个消息的处理函数，如图 15-9 所示。

（11）双击 CMyChessDlg 类下的 OnSetCursor 项，添加如下代码：

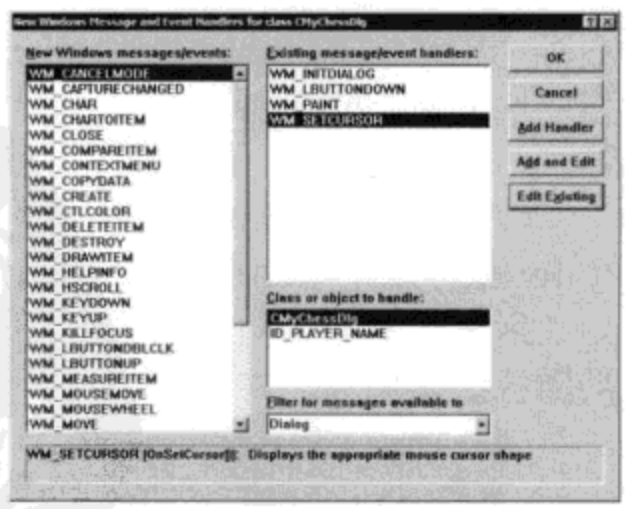


图 15-9　添加消息处理函数

```
    BOOL  CMyChessDlg::OnSetCursor(CWnd*  pWnd,  UINT
nHitTest, UINT message)
    {
        if(person==0) //若尚未输入玩家姓名，则使用默认光标
            return CDialog::OnSetCursor(pWnd, nHitTest,
message);
        if(nHitTest==HTCLIENT)
```

```
    {
        if(person==1)                                    //若下棋者为玩家 A，则使用光标 1
            SetCursor(hcur1);
        else if(person==2)                               //若下棋者为玩家 B，则使用光标 2
            SetCursor(hcur2);
    }
    return TRUE;
}
```

（12）双击 CMyChessDlg 类下的 OnLButtonDown 项，添加如下代码：

```
void CMyChessDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
    if(person==0)                                        //若未输入玩家姓名，则返回
        return;

    int xpos=point.x/xstep;                              //计算鼠标单击处的棋格位置
    int ypos=point.y/ystep;
    int curPerson;                                       //当前下棋者
    if(hasChess[xpos][ypos]==0)                          //若该位置尚未下子
    {
        CClientDC dc(this);                              //获取客户区 DC
        CRect rcChess(xpos*xstep,ypos*ystep,(xpos+1)*xstep,(ypos+1)*ystep);
        CBrush* pOldBrush=dc.SelectObject(&br1);
        dc.SetBkMode(TRANSPARENT);
        if(person==1)                                    //若为玩家 A
        {
            hasChess[xpos][ypos]=1;                      //记录棋子值
            pOldBrush=dc.SelectObject(&br1);             //使用画刷 1
            dc.Ellipse(rcChess);
            dc.TextOut(rcChess.left+xstep/3*1,rcChess.top,"A");
            person=2;                                    //设置下棋者为玩家 B
            curPerson=1;                                 //记录当前下棋者
        }
        else if(person==2)                               //若为玩家 B
        {
            hasChess[xpos][ypos]=2;
            pOldBrush=dc.SelectObject(&br2);
            dc.Ellipse(rcChess);
            dc.TextOut(rcChess.left+xstep/3*1,rcChess.top,"B");
            person=1;
            curPerson=2;
        }
        dc.SelectObject(pOldBrush);
    }
    BOOL isEnd=FALSE;                                    //是否有一方获胜
    CString winner="";                                   //获胜者名称
    if(Beat(xpos,ypos,curPerson)==1)                     //若当前下棋者已获胜
    {
        if(curPerson==1)
            winner=name_a;                               //设置获胜者名称
        else
            winner=name_b;
        isEnd=TRUE;                                      //设为 TRUE
    }
    if(isEnd==TRUE)                                      //若获胜
    {                                                    //弹出信息提示框
        MessageBox(winner+"获胜...再来一局吧！","胜负已定",MB_OK|MB_ICONINFORMATION);
        if(person==1)                                    // person 切换为获胜者
            person=2;
        else
            person=1;
        for(int i=0;i<30;i++)
            for(int j=0;j<30;j++)
                hasChess[i][j]=0;                        //清空数组
        Invalidate();                                    //重绘窗口
    }
    CDialog::OnLButtonDown(nFlags, point);
```

```
}
```

xpos、ypos 为鼠标单击处所在的棋格位置，若该位置尚未下子，根据当前下棋者，在棋盘上绘制椭圆和文本，并在数组中记录当前棋格的值。下棋后，调用 Beat 函数判断当前下棋者是否已获胜。若获胜，弹出消息提示框，并清空数组，开始新一局。

Tips 在 OnLButtonDown 函数中绘制的椭圆和文本是暂时的，当窗口重绘时会被清除掉，需要在重绘函数 OnPaint 中，根据二维数组记录的值，重新绘制棋盘和棋子。

（13）生成程序并运行，如图 15-10 所示。选择“游戏设置”|“玩家设置”命令，弹出“输入玩家姓名”窗口，输入姓名后，单击“确定”按钮，开始游戏。

（14）用鼠标在棋盘下棋，双方轮流下棋，可根据光标的样式判断该哪一方下棋，若有一方胜出，则弹出信息提示框，如图 15-11 所示。

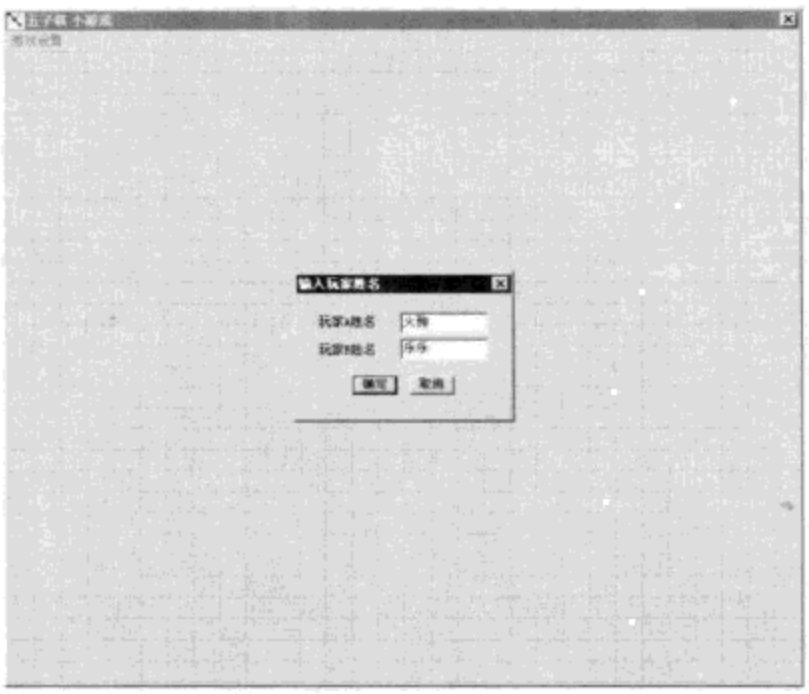


图 15-10　输入玩家姓名

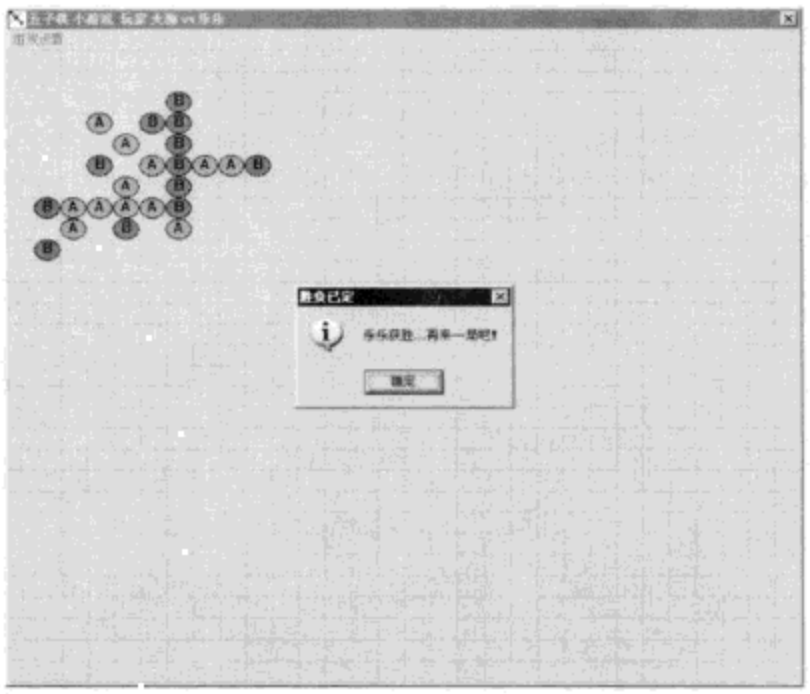


图 15-11　鼠标下棋，一方获胜

15.4 小结

五子棋游戏看似简单，却也包含巨大的智慧，要掌握好防御和进攻的方法。本章利用 MFC 提供的 GDI 图形编程类，实现棋盘、棋子的绘制，并设计获取算法，判断是否有一方已经获胜，利用鼠标事件实现交互式下棋，最终实现一个简单的五子棋游戏。

第 16 章　公交换乘软件

公交换乘是日常出行的一个重要组成部分，如北京市有上千路公交车，每路公交车都经过三四十个公交站点，如此庞大复杂的公交网络交织在一起，给日常出行带来一定困难，我们经常需要查询从一个站点到另一个站点怎么坐最近，是否有直达路线，若没有直达车，怎样换乘路线最短。

目前手机上网已发展较为成熟，很多网站也提供手机版的页面，使用手机上网可以很方便地查询到换乘路线，相信随着技术的发展，移动设备将逐步改善生活，帮助人们获取各种需要的信息。本实例实现一个桌面版的公交换乘软件，具有强大的输入提示功能，帮助读者了解如何实现公交换乘，在此基础上，可将软件移植到网页或移动设备中。

16.1 数据库设计

本实例需要从数据库中读取公交线路和公交站点的信息，在设计程序界面前应先设计数据库，数据库表设计好后，再根据表结构设计程序界面。

Tips　本数据库包含大量数据，下面的内容只是介绍如何创建数据库和表格，实际运行程序时，可直接将数据库文件附加到 SQL Server 中，否则需要手动输入大量数据。

（1）运行 SQL Server 的“服务管理器”，启动数据库服务。

（2）打开 SQL Server 的“企业管理器”，新建一个数据库名为 citygis，再新建一个数据表名为“公交线路表”，表结构如图 16-1 所示。

图 16-1　表结构

其中第 1 列“公交编号”用于存放公交车的号码，如 387、特 6 等，第 2 列“上行路线”用于存放公交车上行线经过的所有站点，第 3 列“下行路线”用于存放公交车下行线经过的所有站点。

Tips　一般情况下，公交车的上行、下行路线是对应一致的，但部分路线可能上下行经过的站点不一致，走不同的路线，因此需要将来回路线分别存储。

（3）设置表结构和表名后，右键单击“公交线路表”，在弹出的快捷菜单中选择“打开表”|“返回所有行”命令，显示表格数据，如图 16-2 所示。

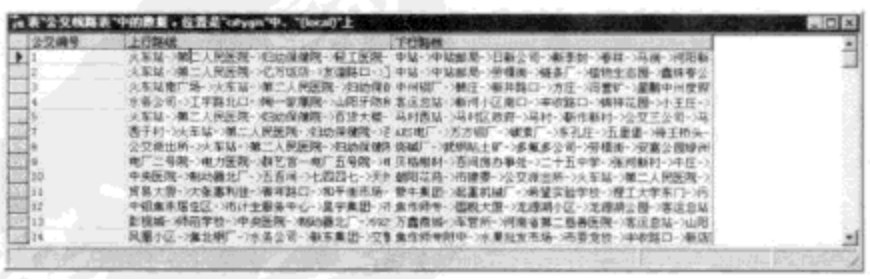


图 16-2　表数据

“上行路线”列显示该路线出发时经过的所有站点，其内容格式如下：

火车站南广场->火车站->第二人民医院->妇幼保健院->百货大楼->实验中学->东方宾馆->第一人民医院->亚细亚大酒店->第十七中学->体育馆->百货大楼焦东超市->凤凰小区->牛庄->陶瓷三厂->百间房乡政府->百间房->市戒毒所->小马村->马界村->田门村->李庄->星鹏中州度假酒店->冯营矿->方庄->新井路口->韩庄->中州铝厂

“下行路线”列显示该路线回去时经过的所有站点，其内容格式如下：

中州铝厂->韩庄->新井路口->方庄->冯营矿->星鹏中州度假酒店->李庄->田门村->马界村->小马村->市戒毒所->百间房->百间房乡政府->陶瓷三厂->牛庄->凤凰小区->百货大楼焦东超市->体育馆->第十七中学->亚细亚大酒店->第一人民医院->东方宾馆->实验中学->百货大楼->妇幼保健院->第二人民医院->火车站->火车站南广场

不同站点间用“->”分隔符隔开，分隔符可任意设置，只要能起到分隔字符串的效果就行。由于表数据量大，为减少工作量，可直接将已有数据库附加到 SQL Server 中，步骤如下所示：

（1）右键单击“数据库”，在弹出的快捷菜单中选择“所有任务”|“附加数据库”命令，弹出“附加数据库”窗口，如图 16-3 所示。

（2）单击…按钮，打开“浏览现有的文件”窗口，选择 MDF 文件，如图 16-4 所示。

图 16-3 附加数据库

图 16-4 浏览 MDF 文件

（3）单击“确定”按钮，添加 MDF 和 LDF 文件，再单击“确定”按钮，完成附加数据库。

Tips MDF 是数据文件，LDF 是日志文件，记录数据库操作步骤。

16.2 界面设计

【实例 16-1】新建一个对话框工程名为 BusLine，实现一个具有输入提示功能的公交换乘软件。

（1）新建对话框工程 BusLine，拖放六个静态文本、三个编辑框、组合框、按钮、列表控件到对话框模板上，如图 16-5 所示。

（2）设置静态文本的 Caption 依次为“输入站点信息”、“换乘方案”、“起始站点”、“结束站点”、“换乘次数”、“路线信息”。设置编辑框的 ID 依次为 IDC_EDIT_START、IDC_EDIT_END、IDC_EDIT_INFO，其中“路线信息”编辑框取消勾选 Auto HScroll 复选框，勾选 Multiline、Vertical scroll、Auto VScroll、Read-only 复选框。

（3）组合框 ID 设为 IDC_COMBO_COUNT，取消勾选 Sort 复选框，Type 设为 DropList。按钮的 ID 设为 IDC_BUTTON_OK，Caption 设为“公交换乘”。列表控件的 View 设为 Report，勾选 Single Selection、Show selection always 复选框。

（4）对话框模板的 ID 设为 IDD_BUSLINE_DIALOG，Caption 设为“公交换乘系统”。

（5）按 Ctrl+W 键打开类向导，选择 Member Variables 选项卡，添加控件变量，如图 16-6 所示。

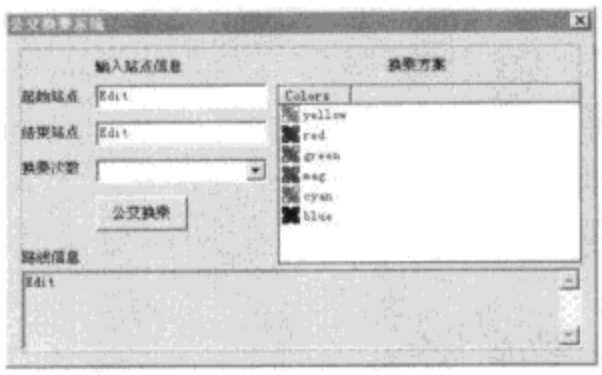


图 16-5　对话框模板

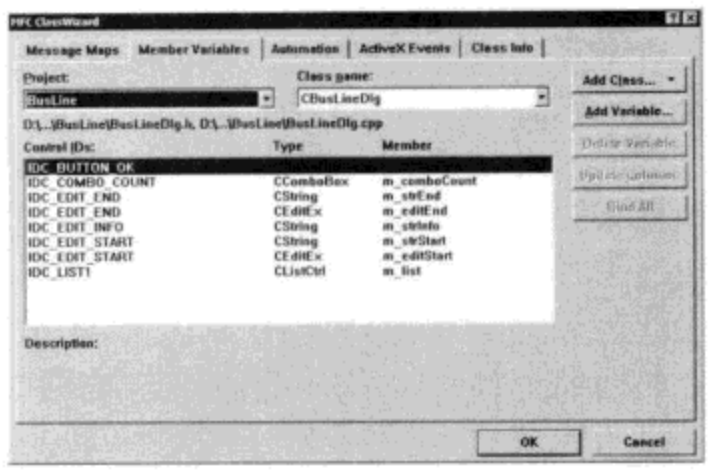


图 16-6　添加控件变量

Tips　一个控件可添加多个变量，如编辑框 IDC_EDIT_START 既有 CString 类型的变量，又有 CEditEx 类型的变量，其中 CEditEx 是自定义的 MFC 类，可先添加 CEdit 类的变量，再手动修改。

16.3 算法设计

常用的公交换乘有直达路线、一次换乘、两次换乘三种方式，每种方式对应一种算法，下面分别介绍三种方式各自采用的算法，其中起始站点为 A，结束站点为 B。

16.3.1 直达路线

直达路线为 A->B，从 A 乘坐一辆公交后，直接到达 B，算法步骤如下：

- 查询经过站点 A 的所有公交路线，并获取 A 在对应路线上的序号。
- 遍历经过 A 的路线上的所有站点，判断是否有站点 B。若没有 B，则查询下一条经过 A 的路线，若有 B，则获取 B 在该路线上的序号。
- 比较 A 和 B 的序号，若 A 序号小于 B，则该路线可用，否则查询下一条经过 A 的路线。

Tips　若 A 序号大于 B，则表明该路线先经过站点 B，后经过站点 A，因此该路线不可用，上下行路线分开存储，每条路线上的站点都是有序的。

16.3.2 一次换乘

一次换乘为 A->C->B，其中 C 为中转站，从 A 乘坐公交 1 到 C 后，再换乘公交 2 到达 B，算法步骤如下：

- 查询经过站点 A 的所有公交路线，并获取 A 在路线上的序号。
- 查询经过站点 B 的所有公交路线，并获取 B 在路线上的序号。
- 遍历经过 A 的路线上的 A 之后的所有站点，如站点 m。
- 遍历经过 B 的路线上的 B 之前的所有站点，如站点 n。
- 若有站点 m 和 n 是一个站点，则该站点是中转站，否则查询其他站点和路线。

16.3.3 两次换乘

两次换乘为 A->C->D->B，其中 C、D 为中转站，从 A 乘坐公交 1 到 C，换乘公交 2 到 D，再换乘公交 3 到达 B，算法步骤如下：

- 查询经过站点 A 的所有公交路线，并获取 A 在路线上的序号。
- 查询经过站点 B 的所有公交路线，并获取 B 在路线上的序号。
- 遍历经过 A 的路线上的 A 之后的所有站点，如站点 C。
- 遍历经过 B 的路线上的 B 之前的所有站点，如站点 D。
- 查询经过站点 C 的所有路线，如路线 X。
- 遍历经过 C 的路线 X 上的 C 之后的所有站点，如站点 P。
- 若有站点 P 和 D 是一个站点，则该站点 D 是中转站 2，站点 C 是中转站 1。

Tips 换乘时要注意去除相同的线路，如从站点 A 乘坐路线 1 到 C，再乘坐路线 1 到 B，这是没有意义的，应排除掉。

16.4 智能提示编辑框

在某些需要输入名称的操作中，由于名称不容易记忆且输入麻烦、容易出错，从而影响软件的使用效果。Windows 自带的编辑框功能有限，无法提供智能提示功能，因此有必要设计一个具有智能提示功能的输入编辑框，具体功能设计如下：

- 输入字符后，自动出现下拉提示框，提示框中的名称包含有编辑框中内容。
- 通过上下方向键，切换选中项，按下回车键，将提示框中的选中项显示在输入编辑框中。
- 提示框中的内容随编辑框的内容变化而动态改变。
- 若编辑框内容为空，或失去输入焦点，或没有满足的提示项时，提示框消失。

要扩展已有编辑框的功能，可创建一个 CEdit 派生类，在派生类中添加新功能。

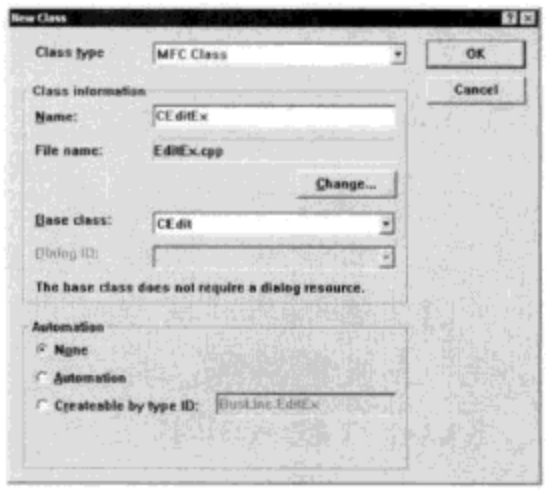


图 16-7 添加 CEdit 派生类

（1）选择 Insert|New Class 命令，如图 16-7 所示。

（2）Class type 组合框选择 MFC Class 项，Name 编辑框输入 CEditEx，Base class 组合框选择 CEdit 项，单击 OK 按钮完成添加。

（3）CEditEx 类的头文件代码如下：

```
#if !defined(AFX_EDITEX_H__7265F8B0_8D04_4F48_95ED_2094601E24AB__INCLUDED_)
#define AFX_EDITEX_H__7265F8B0_8D04_4F48_95ED_2094601E24AB__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// EditEx.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// 智能提示编辑框
class CEditEx : public CEdit
{
// Construction
```

```
public:
    CEditEx();                                                          //构造函数
// Attributes
public:
// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CEditEx)
    public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);                        //消息预处理函数
    //}}AFX_VIRTUAL
// Implementation
public:
    CFont m_font;                                                       //字体
    BOOL Initialize(CWnd* pParent,CRect rcEdit);                        //控件初始化
    void ShowList(BOOL bShow);                                          //是否显示提示框
    void SetData(CStringArray& strData,BOOL bStaticSource);             //设置数据源
    virtual ~CEditEx();                                                 //析构函数
    // Generated message map functions
protected:
    BOOL bShow;                                                         //显示标志
    CListBox m_list;                                                    //列表框
    //{{AFX_MSG(CEditEx)
    afx_msg void OnKillFocus(CWnd* pNewWnd);                            //失去焦点处理函数
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()                                               //消息映射宏声明
};
    ////////////////////////////////////////////////////////////////////////////
    //{{AFX_INSERT_LOCATION}}
    // Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
    #endif // !defined(AFX_EDITEX_H__7265F8B0_8D04_4F48_95ED_2094601E24AB__INCLUDED_)
```

（4）CEditEx 类的源文件代码如下：

```
    // EditEx.cpp : implementation file
    //
    #include "stdafx.h"
    #include "EditEx.h"
    #ifdef _DEBUG
    #define new DEBUG_NEW
    #undef THIS_FILE
    static char THIS_FILE[] = __FILE__;
    #endif
    /*************************************************************************/
    /*智能提示编辑框，为海量、难记的名称输入提供提示功能，提高输入效率*/
    /*************************************************************************/
    CEditEx::CEditEx()                          //构造函数
    {
        bShow=FALSE;                            //显示标志设为 false，不显示列表框
    }
    CEditEx::~CEditEx()                         //析构函数
    {
        if(m_list.GetSafeHwnd())
            m_list.DestroyWindow();             //销毁列表框控件
        if(m_font.GetSafeHandle())
            m_font.DeleteObject();              //释放字体资源
    }
    BEGIN_MESSAGE_MAP(CEditEx, CEdit)           //消息映射宏
        //{{AFX_MSG_MAP(CEditEx)
        ON_WM_KILLFOCUS()
```

```
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
// CEditEx message handlers
BOOL CEditEx::Initialize(CWnd *pParent,CRect rcEdit)  //控件初始化
{
    m_list.Create(WS_BORDER|WS_VSCROLL|WS_HSCROLL|LBS_SORT,CRect(0,0,10,10),pPar
ent,1235);   //动态创建列表控件，参数1为样式，依次为边框、垂直滚动条、水平滚动条、排序
    //参数3为父窗口指针，参数4为控件ID
    m_font.CreatePointFont(90,"宋体");                    //创建字体
    m_list.SetFont(&m_font);                              //设置列表框字体
    CRect ListRect;                                       //列表框位置
    pParent->ScreenToClient(rcEdit);                      //将编辑框的屏幕坐标转换为客户坐标
    ListRect.top=rcEdit.bottom;                           //设置列表框的上、左、右、下位置
    ListRect.left=rcEdit.left;
    ListRect.right=rcEdit.left+150;
    ListRect.bottom=rcEdit.bottom+160;
    m_list.MoveWindow(ListRect);                          //调整列表框位置

    return TRUE;
}
void CEditEx::SetData(CStringArray &strData,BOOL bStaticSource)//设置提示框数据源
{
    CString strEdit;
    GetWindowText(strEdit);                               //获取编辑框输入内容
    if(strEdit.IsEmpty())                                 //若输入为空
    {
        ShowList(FALSE);                                  //不显示提示框
        return;                                           //返回
    }
    m_list.ResetContent();                                //清空提示框

    if(bStaticSource)                                     //若为静态数据源
    {
        for(int i=0;i<strData.GetSize();i++)              //遍历数据源
        {
            CString strElem=strData.GetAt(i);
            if(strElem.Find(strEdit)!=-1)                 //若某一项包含输入的内容
                m_list.AddString(strElem);                //将该项添加至提示框
        }
        if(m_list.GetCount()==0)                          //若提示框数目为0，不显示
            ShowList(FALSE);
        else
            ShowList(TRUE);                               //显示提示框
    }
    else                                                  //若为动态数据源
    {
        if(strData.GetSize()==0)                          //若数据源为空
        {
            ShowList(FALSE);                              //不显示，返回
            return;
        }
        for(int i=0;i<strData.GetSize();i++)
            m_list.AddString(strData.GetAt(i));           //将数据源的所有项添加至提示框
        ShowList(TRUE);                                   //显示提示框
    }
}
void CEditEx::ShowList(BOOL bShow)                        //是否显示提示框
{
    this->bShow=bShow;
```

```
    m_list.ShowWindow(bShow);                                    //提示框是否显示
}
void CEditEx::OnKillFocus(CWnd* pNewWnd)                         //控件失去焦点处理函数
{
    CEdit::OnKillFocus(pNewWnd);

    // TODO: Add your message handler code here
    if(bShow)
        ShowList(FALSE);                                         //若提示框显示，则设为不显示
}
#define VisibleRowCount 14                                       //定义一次可见的提示项数目

BOOL CEditEx::PreTranslateMessage(MSG* pMsg)                     //消息预处理函数
{
    if(bShow)                                                    //若提示框已显示
    {
        int nSelect=m_list.GetCurSel();                          //获取当前选择项
        int nShowIndex=m_list.GetTopIndex();                     //获取当前可见的首项索引
                                                                 //若为键盘消息，向下方向键
        if(pMsg->message==WM_KEYDOWN && pMsg->wParam==VK_DOWN)
        {
            //若当前选择项小于可见的最小项索引，或大于可见范围
            if(nSelect<nShowIndex || nSelect>=nShowIndex+VisibleRowCount)
            {
                m_list.SetCurSel(nShowIndex);                    //设置当前选择项为可见的最小项
            }
            else                                                 //若当前项可见
            {
                if(nSelect==LB_ERR || nSelect==m_list.GetCount()-1)
                                                                 //若为最后一项，则移至首项
                    m_list.SetCurSel(0);
                else                                             //若为中间项
                {
                    m_list.SetCurSel(nSelect+1);                 //当前选择项下移
                }
            }
            return TRUE;                                         //返回
        }
        if(pMsg->message==WM_KEYDOWN && pMsg->wParam==VK_UP)  //若为向上方向键
        {
            if(nSelect!=LB_ERR  &&  (nSelect<nShowIndex  ||  nSelect>=nShowIndex+
VisibleRowCount))
            {
                m_list.SetCurSel(nShowIndex+VisibleRowCount-1);
            }
            else
            {
                if(nSelect==LB_ERR || nSelect==0)                //若为第 0 项，则移至尾项
                    m_list.SetCurSel(m_list.GetCount()-1);
                else
                {
                    m_list.SetCurSel(nSelect-1);                 //当前选择项上移
                }
            }
            return TRUE;                                         //返回
        }
        if(pMsg->message==WM_KEYDOWN && pMsg->wParam==13) //若为回车键
        {
            if(nSelect!=LB_ERR)                                  //若有选中项
            {
```

```
                CString strSelect;
                m_list.GetText(nSelect,strSelect);          //获取选中项的值
                SetWindowText(strSelect);                   //设置为编辑框的值
                ShowList(FALSE);                            //隐藏提示框
                return TRUE;                                //返回
            }
        }
    }

    return CEdit::PreTranslateMessage(pMsg);
}
```

Initialize 函数用于创建列表框，并设置列表框的位置，参数 1 为编辑框父窗口的指针值，参数 2 为编辑框的屏幕坐标值。Create 函数动态创建一个列表框控件，CreatePointFont 函数创建一个简单的字体，SetFont 函数设置列表框使用的字体对象。

ScreenToClient 函数将编辑框的屏幕坐标转换为父窗口下的客户坐标，ListRect 为列表框的矩形边界，其顶部与编辑框的底部对齐，左边界也与编辑框对齐，宽度为 150，高度为 160。MoveWindow 函数设置列表框的窗口位置，相对于父窗口而定。

SetData 函数设置列表框的数据源，参数 1 为可变数组，参数 2 为标识符，表明是否为静态数据源，若为静态数据源，表明数据源固定不变，始终是那些数据，当编辑框中输入内容，只显示匹配的数据项，若为动态数据源，表明数据源中的所有项都是匹配项，可全部在列表框中显示。本实例中所有公交站点构成的数据是固定不变的，因此使用静态数据源。

GetWindowText 函数获取编辑框中输入的内容，若为空，调用 ShowList 函数隐藏列表框，并直接返回。ResetContent 函数清空列表框。若为静态数据源，利用 for 循环遍历数据源，Find 函数判断每一项是否为输入值的匹配项，调用 AddString 函数将匹配项添加到列表框中。若列表框的项数为 0，隐藏列表框，否则显示列表框。

若为动态数据源，调用 GetSize 函数获取数组的长度，若为 0，不显示列表框。利用 for 循环将数据源中的所有项添加到列表框中，调用 ShowList 函数显示列表框，相对于静态数据源，动态数据源无须判断是否匹配，可直接将所有项添加到列表框中。

ShowList 函数设置列表框的显示状态，参数为显示标志，若为 TRUE 显示，否则隐藏。当编辑框失去输入焦点时，自动调用 OnKillFocus 函数，此时应调用 ShowList 函数隐藏编辑框。

PreTranslateMessage 函数用于预处理消息，在消息被传送至对应的处理函数之前，可拦截该消息，实现特定功能，参数 pMsg 为消息结构体的指针，包含消息所需的所有信息。若列表框已显示，调用 GetCurSel 函数获取列表框选中项的索引，存放到 nSelect 中。GetTopIndex 函数获取列表框当前可见区域的首项索引。

pMsg->message 获取消息的类型，如鼠标左键按下、键盘按下、键盘抬起、左键双击等，pMsg->wParam 获取消息的具体值，如到底按下了哪个键。WM_KEYDOWN 表示键盘按下消息，VK_DOWN 表示↓向下方向键，若按下↓键，且选择项不在可见区域范围，调用 SetCurSel 函数设置选中项为可见区域的首项。若在可见范围里，且尚未选择一项或选择最后一项，调用 SetCurSel 函数选择第一项。return TRUE;直接返回，表示该消息到此结束，停止进一步的处理。↑键的处理流程类似↓键。

若按下 Enter 键，则表示选择一项，Enter 键的虚拟键码为 VK_RETURN，值为 13。GetText 函数获取选中项的文本，SetWindowText 函数设置编辑框的文本，ShowList 函数隐藏列表框，

16.5 功能实现

公交换乘软件的系统流程如下所示：

- 连接数据库，读取表数据。
- 获取公交线路信息和站点信息。

- 输入站点名称弹出智能提示框。
- 选择换乘方式，计算换乘方案，在列表控件中显示。
- 选择一项方案，显示详细信息。

（1）在工程使用 ADO 封装类，将类的头文件和源文件添加到工程中，如图 16-8 所示。

图 16-8　添加 ADO 封装类

（2）在类视图双击 CBusLineDlg 项，在文件开头处添加如下代码，包括类头文件。

```
#include "ADOEx.h"                          //ADO 封装类
#include "EditEx.h"                         //提示编辑框
#include <list>                             //STL 链表类
```

（3）修改两个站点编辑框映射的变量使用 CEditEx 扩展类，代码如下：

```
// Dialog Data
    //{{AFX_DATA(CBusLineDlg)
    enum { IDD = IDD_BUSLINE_DIALOG };
    CEditEx m_editEnd;                              //修改 CEdit 为 CEditEx
    CEditEx m_editStart;                            //修改 CEdit 为 CEditEx
    CListCtrl    m_list;
    CComboBox    m_comboCount;
    CString m_strInfo;
    CString m_strEnd;
    CString m_strStart;
    //}}AFX_DATA
```

（4）在类定义添加如下成员变量：

```
public:
    std::list<CString> m_listStop;              //站点列表
    CStringArray m_arrayStop;                   //站点列表数组，列表框的静态数据源
    CADOEx m_ado;                               //ADO 封装类对象，操作数据库
    BOOL bStartFirstLoad;                       //首次加载标志，用于只需操作一次的判断
    BOOL bEndFirstLoad;                         //首次加载标志
    CStringArray m_arrayLine;                   //路线信息数组
    CStringArray m_arrayNumber;                 //路线编号数组
```

Tips　std 为 C++库的标准命名空间，使用命名空间可区分同名的类。

（5）双击 CBusLineDlg 类的 OnInitDialog 项，在 return TRUE; 前添加如下代码：

```
m_list.InsertColumn(0,"序号",LVCFMT_LEFT,45);                   //插入列
m_list.InsertColumn(1,"换乘方案",LVCFMT_LEFT,300);              //设置扩展样式
m_list.SetExtendedStyle(m_list.GetExtendedStyle()|LVS_EX_GRIDLINES|LVS_EX_FULLROWSELECT);
m_comboCount.AddString("直达线路");                             //组合框添加项
m_comboCount.AddString("一次换乘");
m_comboCount.AddString("两次换乘");
m_comboCount.SetCurSel(0);                                      //选中第一项
bStartFirstLoad=TRUE;                                           //首次加载标志为 TRUE
bEndFirstLoad=TRUE;
                                                                //SQL Server 连接字符串
CString strConn="driver={SQL Server};Server=FQ-201003301100;Database=citygis;"
  "uid=sa;pwd=sa";
if(!m_ado.Connect(strConn))                                     //连接 citygis 数据库
{
  MessageBox("连接 SQL Server 数据库 citygis 失败，请确认已启动数据库服务",
    "提示信息",MB_OK|MB_ICONERROR);                             //若连接失败，弹出提示框
}
if(m_ado.GetConnectState())                                     //若已连接
{
```

```
CString strSQL="select * from 公交线路表";                          //读取 SQL 语句
CStringArray fields,strResult;
fields.Add("公交编号");                                             //要读取的多个字段
fields.Add("上行路线");
fields.Add("下行路线");

m_ado.ExecuteSelSQL(strSQL,fields,strResult);                       //读取多个字段的数据
m_arrayLine.SetSize(strResult.GetSize()*2);                         //设置数组的初始大小
m_arrayNumber.SetSize(strResult.GetSize()*2);
CString busIndex,upLine,downLine;
for(int i=0;i<strResult.GetSize();i++)                              //遍历所有路线
{
  busIndex=m_ado.GetSingleString(strResult.GetAt(i),1);            //获取路线编号
  upLine=m_ado.GetSingleString(strResult.GetAt(i),2);               //获取上行路线
  downLine=m_ado.GetSingleString(strResult.GetAt(i),3);             //获取下行路线
  m_arrayLine.SetAtGrow(i*2,upLine);                                //路线信息存入动态数组中
  m_arrayLine.SetAtGrow(i*2+1,downLine);
  m_arrayNumber.SetAtGrow(i*2,"L"+busIndex+"u");                    //路线编号存入动态数组中
  m_arrayNumber.SetAtGrow(i*2+1,"L"+busIndex+"d");
  int nUpCount=m_ado.GetSingleStringNum(upLine,"->");               //获取上行路线的站点数目
  int nDownCount=m_ado.GetSingleStringNum(downLine,"->");           //获取下行路线的站点数目
  int j=0;
  for(j=0;j<nUpCount;j++)                                           //遍历站点
  {
    CString strStop=m_ado.GetSingleString(upLine,j+1,"->");  //获取每一个站点的名称
    m_listStop.push_back(strStop);                                  //添加到链表中
  }
  for(j=0;j<nDownCount;j++)
  {
    CString strStop=m_ado.GetSingleString(downLine,j+1,"->");
    m_listStop.push_back(strStop);
  }

  m_listStop.sort();                                                //链表排序
  m_listStop.unique();                                              //去除链表中的重复项
}
m_arrayStop.SetSize(m_listStop.size());                             //设置站点数组的初始大小
std::list<CString>::iterator iter;                                  //链表的迭代器
int nCount=0;
for(iter=m_listStop.begin();iter!=m_listStop.end();iter++)   //遍历链表
{
  m_arrayStop.SetAtGrow(nCount++,*iter);                    //将链表中的元素添加到数组中
}
}
```

strConn 为数据库连接字符串，Connect 函数连接数据库，若连接失败，弹出错误提示框。strSQL 为 SQL 查询语句，获取“公交线路表”中的所有数据，fields 为要读取的字段数组，ExecuteSelSQL 函数执行多字段读取，读取结果存放到 strResult 中。

SetSize 函数设置动态数组的初始大小，若已知数组所需大小，可设置初始大小，一次分配足够空间，提高效率。由于上下行路线分别作为一条单独的路线，所以 m_arrayLine、m_arrayNumber 的数组大小是 strResult 的两倍，其中 m_arrayLine 存放路线的站点信息，m_arrayNumber 存放路线的编号。

GetSingleString 函数获取拼接字符串中单个字符串的值，参数 1 为拼接字符串，参数 2 为单个字符串的位置。SetAtGrow 函数设置动态数组的元素值，依次存放上、下行路线的信息。

Tips 路线编号采用“L2u”、“L2d”的形式存储，其中 L 表示路线，2 为公交编号，u 和 d 分别代表上行 up、下行 down。

GetSingleStringNum 函数获取拼接字符串中单个字符串的数目，即分隔符的数目加 1。GetSingleString 函数获取各个站点的名称，存放到链表 m_listStop 中。sort 函数对链表进行排序，unique 函数去除链表中的重复项。

Tips　只有对链表 list 调用 sort 函数排序后，才能调用 unique 函数去除重复项。

动态数组 m_arrayStop 用于存放处理后的所有站点名称，作为列表框的数据源。iter 为 m_listStop 的迭代器，begin 函数获取迭代器的起始位置，end 获取结束位置，利用 for 循环遍历链表，将链表中存储的站点名称添加到动态数组 m_arrayStop 中。

（6）在资源视图双击 IDD_BUSLINE_DIALOG 项，打开对话框模板，双击“起始站点”和“结束站点”编辑框，添加编辑框的 EN_CHANGE 消息处理函数，添加如下代码：

```
void CBusLineDlg::OnChangeEditStart()                    //编辑框输入内容改变事件处理函数
{
    if(bStartFirstLoad)                                  //若为第一次改变
    {
        CRect EditRect;
        m_editStart.GetWindowRect(EditRect);             //获取编辑框的屏幕坐标

        m_editStart.Initialize(this,EditRect);           //初始化编辑框
        m_editStart.SetData(m_arrayStop,TRUE);           //设置数据源

        bStartFirstLoad=FALSE;                           //初次加载标志设为 FALSE
    }
    else                                                 //不是第一次加载，直接设置数据源
        m_editStart.SetData(m_arrayStop,TRUE);
}
void CBusLineDlg::OnChangeEditEnd()
{
    if(bEndFirstLoad)
    {
        CRect EditRect;
        m_editEnd.GetWindowRect(EditRect);
        m_editEnd.Initialize(this,EditRect);
        m_editEnd.SetData(m_arrayStop,TRUE);
        bEndFirstLoad=FALSE;
    }
    else
        m_editEnd.SetData(m_arrayStop,TRUE);
}
```

当编辑框输入内容改变时，自动调用该函数。bStartFirstLoad、bEndFirstLoad 用于判断是否为初次调用该函数，若为第一次调用，GetWindowRect 函数获取编辑框的屏幕坐标值，Initialize 函数初始化编辑框，参数为编辑框的父窗口指针、编辑框的屏幕坐标，在内部创建列表框，并设置列表框的窗口位置。SetData 函数设置编辑框的数据源，参数 1 为数据源所在的数组，参数 2 表示静态数据源。

（7）在类视图双击 CBusLineDlg 项，添加四个成员函数，代码如下：

```
public:
    static UINT ThreadProc(LPVOID pParam);    //线程函数
    void TwoTime();                           //两次换乘函数
    void OneTime();                           //一次换乘函数
    void ZeroTime();                          //直达函数
```

（8）在资源视图双击 IDD_BUSLINE_DIALOG 项，打开对话框模板，双击“公交换乘”按钮，添加如下代码：

```
void CBusLineDlg::OnButtonOk()
```

```
    {
        UpdateData();                                          //更新控件变量
        if(m_strStart.IsEmpty() || m_strEnd.IsEmpty())         //若输入为空，则直接返回
            return;
        int nSel=m_comboCount.GetCurSel();                     //获取换乘次数
        if(nSel==0)
            ZeroTime();                                        //若选择直达，则调用 ZeroTime 函数
        else if(nSel==1)
            OneTime();                                         //若选择一次换乘，则调用 OneTime 函数
        else
            TwoTime();                                         //若选择两次换乘，则调用 TwoTime 函数
    }
```

（9）在当前文件末尾添加 ZeroTime 函数，代码如下：

```
    void CBusLineDlg::ZeroTime()
    {
        m_list.DeleteAllItems();                               //清空列表控件
        CString strBusEnable,strTemp;
        int nBusEnable=0;                                      //结果数目
        for(int nCount=0;nCount<m_arrayLine.GetSize();nCount++)    //公交线路循环遍历
        {   //查询起点在第 nCount 条线路上的序号，若没有该站点，返回 0
            int     nStartStopIndex=m_ado.GetIndexOfString(m_arrayLine.GetAt(nCount),
m_strStart);
            if(nStartStopIndex==0)                             //若没找到起点，则查询下一条线路
                continue;
            //查询终点在第 nCount 条线路上的序号，若没有该站点，返回 0
            int nEndStopIndex=m_ado.GetIndexOfString(m_arrayLine.GetAt(nCount),m_ strEnd);
            if(nEndStopIndex==0)                               //若没找到终点，则查询下一条线路
                continue;
            if(nStartStopIndex<=nEndStopIndex)                 //若起点序号小于终点序号
            {
                nBusEnable++;                                  //结果数目递增
                strTemp.Format("%d",nBusEnable);

                strBusEnable=m_arrayNumber.GetAt(nCount);      //获取线路名称

                int nItem=m_list.InsertItem(m_list.GetItemCount(),strTemp);
                                                               //向控件插入新项，第 0 列为序号
                m_list.SetItemText(nItem,1,strBusEnable);      //设置第 1 列值为线路名称
            }
        }
        if(nBusEnable==0)                                      //若结果数目为 0
        {
            m_strInfo="\t 起点和终点没有直达的公交车，请尝试一次换乘...";//显示提示信息
            UpdateData(FALSE);
        }
    }
```

（10）在当前文件末尾添加 OneTime 函数，代码如下：

```
    void CBusLineDlg::OneTime()
    {
        GetDlgItem(IDC_BUTTON_OK)->SetWindowText("请稍等…");
        GetDlgItem(IDC_BUTTON_OK)->EnableWindow(FALSE);    //禁用公交换乘按钮
        m_list.DeleteAllItems();                           //清空列表
        CString strBusEnable,strTemp;
        int nBusEnable=0;
        for(int nBusA=0;nBusA<m_arrayLine.GetSize();nBusA++)  //线路循环遍历
        {   //查询起点序号
            int     nStartStopIndex=m_ado.GetIndexOfString(m_arrayLine.GetAt(nBusA),
m_strStart);
```

```
            if(nStartStopIndex==0)                              //若没找到起点，则跳入下一次循环
                continue;
            //对 nBusA 线路上起点后的站点进行循环遍历
            for(int nStopMid = nStartStopIndex + 1 ; nStopMid <= m_ado.GetSingleStringNum
( m_arrayLine.GetAt ( nBusA ) , "->" ) ; nStopMid++)
            {   //获取站点名称
                CString
strMidStop=m_ado.GetSingleString(m_arrayLine.GetAt(nBusA),nStopMid,"->");
                for(int nBusB=0;nBusB<m_arrayLine.GetSize();nBusB++)//线路第 2 次循环遍历
                {   //查询终点序号
                    int        nEndStopIndex=m_ado.GetIndexOfString(m_arrayLine.GetAt
(nBusB),m_strEnd);
                    if(nEndStopIndex==0)      //若没找到终点，则进入下一次循环 continue;
                    //若在该线路找到终点，再查找中间站点
                    int  nMidStopIndex=m_ado.GetIndexOfString(m_arrayLine.GetAt(nBusB),
strMidStop);
                    if(nMidStopIndex==0)      //若没有找到中间点，则进入下一次循环
                        continue;
                    if(nMidStopIndex<nEndStopIndex) //若找到中间点，且中间点序号小于终点序号
                    {   //获取路线 A 的名称，类似"L1d"
                        CString strBusAIndex=m_arrayNumber.GetAt(nBusA);
                        CString strBusBIndex=m_arrayNumber.GetAt(nBusB);//获取路线 B 的名称
                        //检测 A 和 B 是否为同一条路线
                        CString strTempA,strTempB;
                        strTempA=strBusAIndex;
                        strTempB=strBusBIndex;
                        strTempA.Delete(strTempA.GetLength()-1);
                                                            //清除最后一位，类似 L1
                        strTempB.Delete(strTempB.GetLength()-1);
                        if(strTempA==strTempB)    //若路线 A 和路线 B 为同一路线，则不符合要求
                            continue;
                        nBusEnable++;             //结果数目递增
                        strTemp.Format("%d",nBusEnable);
                        strBusEnable=strBusAIndex+"->"+strMidStop+"->"+strBusBIndex;
                                                  //拼接换乘方案
                        int nItem=m_list.InsertItem(m_list.GetItemCount(),strTemp);
                                                  //列表添加新项
                        m_list.SetItemText(nItem,1,strBusEnable);
                    }
                }
            }
        }
        if(nBusEnable==0)                                   //若没有结果
        {
            m_strInfo="\t 起点和终点之间没有可以一次换乘的路线...";
            UpdateData(FALSE);
        }
        GetDlgItem(IDC_BUTTON_OK)->SetWindowText("公交换乘");   //恢复按钮标题
        GetDlgItem(IDC_BUTTON_OK)->EnableWindow(TRUE);          //按钮恢复可用
    }
```

（11）在当前文件末尾添加 TwoTime 函数，代码如下：

```
void CBusLineDlg::TwoTime()
{
    AfxBeginThread(ThreadProc,(void*)this);                 //开辟新线程
}
```

AfxBeginThread 函数用来启动一个线程，参数 1 为线程函数，参数 2 为传递给线程函数的参数值。

Tips 两次换乘计算量大，耗时长，需要开辟新线程，否则会出现假死现象。多线程是一种常用的技术，多用于一些耗时的操作，如一些聊天软件，可以边视频，边发信息。一个运行的程序称为一个进程，一个进程至少包含一个线程，也称为主线程，若只有一个主线程，则该程序的执行流程是单一顺序执行的，若执行到了某一处复杂耗时的操作，一时无法完成时，程序会陷入假死状态，此时对程序做什么操作都没有反应。多线程技术可以将某段耗时的操作提取出来，放入一个函数中，并开辟一个新线程用来执行该函数，程序可以同时运行多个线程，互不影响，当某个子线程所在的函数执行结束后，自动结束该线程。

（12）在当前文件末尾添加 ThreadProc 函数，代码如下：

```
UINT CBusLineDlg::ThreadProc(LPVOID pParam)
{
    CBusLineDlg* pDlg=(CBusLineDlg*)pParam;                          //获取类指针
    CString strResult;
    pDlg->GetDlgItem(IDC_BUTTON_OK)->SetWindowText("请稍等…");//设置按钮标题
    pDlg->GetDlgItem(IDC_BUTTON_OK)->EnableWindow(FALSE);            //使按钮不可用
    pDlg->m_list.DeleteAllItems();                                   //清空列表
    CString strBusEnable,strTemp;
    int nBusEnable=0;                                                //结果数目
    int nBusA=0;                                                     //路线 1 序号
    int nBusB=0;                                                     //路线 2 序号
    int nBusMid=0;                                                   //路线 3 序号
    for(nBusA=0;nBusA<pDlg->m_arrayLine.GetSize();nBusA++)           //路线循环遍历
    {   //查询起点，若没有找到，则结束本次循环
        int nStartStopIndex = pDlg->m_ado.GetIndexOfString ( pDlg->m_arrayLine.
GetAt(nBusA ) , pDlg->m_strStart );
        if(nStartStopIndex==0)
            continue;
        for(nBusB=0;nBusB<pDlg->m_arrayLine.GetSize();nBusB++)    //路线 2 次循环遍历
        {
            if(nBusB==nBusA)      //若两路线序号相同，则结束本次循环
                continue;
            //查找终点，若没找到，则结束本次循环
            int nEndStopIndex = pDlg->m_ado.GetIndexOfString ( pDlg->m_arrayLine.
GetAt (nBusB) , pDlg->m_strEnd );
            if(nEndStopIndex==0)
                continue;
            int nMidOneIndex=nStartStopIndex+1;                  //起点后的站点循环遍历
            for(  ;  nMidOneIndex  <=  pDlg->m_ado.GetSingleStringNum  (  pDlg->
m_arrayLine. GetAt(nBusA) , "->" ) ; nMidOneIndex++ )
            {
                //获取站点名称，若中间点 1 为终点，则结束本次循环
                CString strMidOneStop = pDlg->m_ado.GetSingleString ( pDlg->m_
arrayLine. GetAt ( nBusA) , nMidOneIndex , "->" );
                if(strMidOneStop==pDlg->m_strEnd)
                    continue;
                //路线 3 次循环遍历
                for(nBusMid=0;nBusMid<pDlg->m_arrayLine.GetSize();nBusMid++)
                {   //若路线 2 与路线 1 和 3 相同，则结束本次循环
                    if(nBusMid==nBusA || nBusMid==nBusB)
                        continue;
                    //在路线 2 上查找中间点 1
                    int nStopMidOneIndex = pDlg->m_ado.GetIndexOfString ( pDlg ->
m_arrayLine. GetAt( nBusMid ) , strMidOneStop);
                    if(nStopMidOneIndex==0)
```

```
                        continue;
                    //中间点 1 后的站点循环遍历
                    int nStopMidTwoIndex=nStopMidOneIndex+1;
                    for( ; nStopMidTwoIndex <= pDlg->m_ado.GetSingleStringNum ( pDlg
-> m_arrayLine.GetAt ( nBusMid ) , "->" ) ; nStopMidTwoIndex++ )
                    {
                        //获取中间点 2 名称
                        CString strStopMidTwo = pDlg -> m_ado.GetSingleString ( pDlg
-> m_arrayLine . GetAt ( nBusMid ) , nStopMidTwoIndex,"->");
                        if(strStopMidTwo==pDlg->m_strStart   ||   strStopMidTwo==
pDlg->m_ strEnd)
                            continue;
                        //在路线 3 上查找中间点 2
                        int nStopMidTwoIndexInBusB = pDlg -> m_ado.GetIndex OfString
( pDlg-> m_arrayLine.GetAt (nBusB) , strStopMidTwo );
                        if(nStopMidTwoIndexInBusB==0)
                            continue;
                        //若中间点 2 的序号小于终点
                        if(nStopMidTwoIndexInBusB<nEndStopIndex)
                        {   //获取路线 1，2，3 的名称
                            CString strBusA=pDlg->m_arrayNumber.GetAt(nBusA);
                            CString strBusMid=pDlg->m_arrayNumber.GetAt(nBusMid);
                            CString strBusB=pDlg->m_arrayNumber.GetAt(nBusB);

                            CString strTempA,strTempB,strTempC;
                            strTempA=strBusA;
                            strTempB=strBusB;
                            strTempC=strBusMid;
                            strTempA.Delete(strTempA.GetLength()-1);
                        //去掉路线的最后一位
                            strTempB.Delete(strTempB.GetLength()-1);
                            strTempC.Delete(strTempC.GetLength()-1);
                            //若 3 条路线任意两条为同一路线，结束本次循环，即不符合要求
                            if(strTempA==strTempB   ||   strTempA==strTempC   ||
strTempB== strTempC)
                                continue;
                            nBusEnable++;                       //结果数目递增
                            strTemp.Format("%d",nBusEnable);
                            strBusEnable = strBusA + "->" + strMidOneStop + "->" +
strBusMid
                                + "->" + strStopMidTwo +"->" + strBusB;
                                                                //拼接方案结果
                            int      nItem=pDlg->m_list.InsertItem(pDlg->m_list.
GetItemCount (),strTemp);
                            pDlg->m_list.SetItemText(nItem,1,strBusEnable);
                        }
                    }
                }
            }
        }
        if(nBusEnable==0)                                       //若没有结果
        {
            pDlg->m_strInfo="\t 起点和终点没有二次换乘的公交车...";     //提示信息
            pDlg->UpdateData(FALSE);
        }
        pDlg->GetDlgItem(IDC_BUTTON_OK)->SetWindowText("公交换乘");//恢复按钮标题
        pDlg->GetDlgItem(IDC_BUTTON_OK)->EnableWindow(TRUE);    //恢复按钮可用
        return 1;
    }
```

ThreadProc 为线程函数，有固定的函数原型，函数原型如下：

```
UINT __cdecl MyControllingFunction( LPVOID pParam );
```

参数 pParam 为 AfxBeginThread 函数传递的值，在 ThreadProc 函数内部转换为 CBusLineDlg* 指针类型。由于 ThreadProc 为 static 静态成员函数，没有 this 指针，无法访问非静态成员变量，需要利用 pParam 传递的类指针获取变量的值。

（13）在资源视图双击 IDD_BUSLINE_DIALOG 项，打开对话框模板，双击“换乘方案”列表控件，添加如下代码：

```
void CBusLineDlg::OnClickList1(NMHDR* pNMHDR, LRESULT* pResult)
{
    CString strSelect;
    int nSelected=-1;
    nSelected=m_list.GetNextItem(nSelected,LVNI_SELECTED);     //获取选中项
    if(nSelected==-1)
        return;
    CString strResult;
    CString strBusResult=m_list.GetItemText(nSelected,1);      //获取选中项第 2 列内容
    if(strBusResult.Find("u")!=-1)                             //若路线名称中包含 u
        strBusResult.Replace("u","路车上行");                  //将字母替换为汉字
    if(strBusResult.Find("d")!=-1)                             //若包含 d
        strBusResult.Replace("d","路车下行");                  //替换为汉字
    strBusResult.Replace("L","");                              //清除字符 L
    strBusResult=m_strStart+"->"+strBusResult;
    strBusResult+="->"+m_strEnd;                               //拼接换乘方案
    strSelect+=m_strStart+"->"+m_strEnd;
    strSelect+="\r\n\t 序号: "+m_list.GetItemText(nSelected,0);
    strSelect+="\r\n\t 换乘方案: \r\n\t\t"+strBusResult;
    m_strInfo=strSelect;
    UpdateData(FALSE);
    *pResult = 0;
}
```

（14）启动数据库服务，生成程序并运行，在编辑框输入信息时，会弹出提示框，显示匹配的站点名称，如图 16-9 所示。按↑和↓键切换选择项，按 Enter 键输入选中项。

图 16-9 输入提示框

（15）单击“公交换乘”按钮，在“换乘方案”列表控件中显示可用的换乘方案，直达路线如图 16-10 所示，一次换乘如图 16-11 所示，两次换乘如图 16-12 所示。

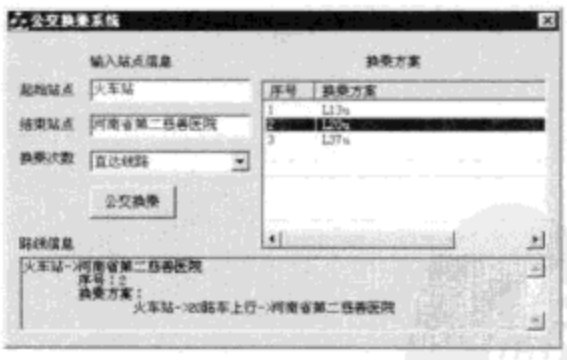


图 16-10 直达路线

图 16-11 一次换乘

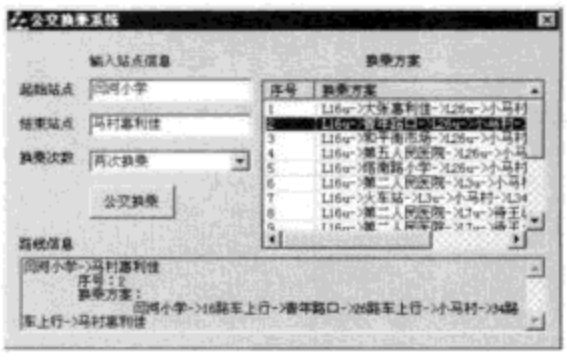


图 16-12 两次换乘

16.6 小结

公交换乘是日常生活的一个重要组成部分，快速找到换乘路线能够节省大量的时间，改善广大人民的生活。本章利用多种技术实现一个功能强大的公交换乘软件，如利用 ADO 技术访问数据库，获取公交路线、站点信息，设计直达、一次换乘、两次换乘算法，获取换乘路线，提供一个具有输入提示的智能编辑框，方便用户的输入，最终实现一个公交换乘软件。

附录 A　Win32 API 开发

Win32 API 是 MFC 开发的基础，其提供了 Windows 系统下所有可用的函数，使用这些函数能够最大程度上发挥 Windows 系统的功能，实现所需的功能，但也是难度最大的一种开发方式。一些书籍将 API 开发放在第 1 章，以便于读者在了解底层 Windows 窗口实现原理的基础上，学习 MFC 开发，本书考虑到 API 开发难度较大，对于初学者来说难以理解，故先讲解 MFC 开发，等读者对 Win32 程序开发有一定感性认识后，再介绍 API 开发流程，不至于一下子接触到那么多陌生的函数而不知所措，降低学习的信心。

【实例 A-1】：以第 3 章创建的 Win32 工程 API01 为例，介绍 Win32 API 开发流程。

（1）打开 Win32 工程 API01，在类视图双击任意一项，打开 API01.cpp 文件，文件开头的部分代码如下：

```
// API01.cpp : Defines the entry point for the application.
//

#include "stdafx.h"                                          //包含头文件
#include "resource.h"

#define MAX_LOADSTRING 100                                   //宏定义，字符串最大长度

//全局变量:
HINSTANCE hInst;                                             //程序实例句柄
TCHAR szTitle[MAX_LOADSTRING];                               //窗口标题栏文本
TCHAR szWindowClass[MAX_LOADSTRING];                         //窗口类名

//函数声明
ATOM             MyRegisterClass(HINSTANCE hInstance);       //注册窗口类函数
BOOL             InitInstance(HINSTANCE, int);               //程序实例初始化函数
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);        //主窗口消息处理函数
LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);          //关于窗口的消息处理函数
```

hInst 为程序实例的句柄，一个运行的程序代表一个实例，若一个程序同时运行多个，如多个记事本程序同时运行，则有多个程序实例。TCHAR 是一个字符宏定义，代表 char 或 wchar_t，在不同的环境下能自动替换为 char 或 wchar_t，从而提高代码的移植性，如 Windows Mobile 系统使用 Unicode 字符集，只能使用 wchar_t 类型，若使用 TCHAR 则无须修改原有代码，可直接在 Windows Mobile 环境下使用。TCHAR 宏定义的部分代码如下所示：

```
#ifdef  UNICODE                                              //若使用 Unicode 字符
  typedef WCHAR TCHAR, *PTCHAR;                              //TCHAR 代表 WCHAR，即 wchar_t
  #define __TEXT(quote) L##quote                             //__TEXT 宏也可用于字符自动切换
#else                                                        //若使用 ANSI 字符
  typedef char TCHAR, *PTCHAR;                               //TCHAR 代表 char
#endif
```

Tips　在 Unicode 环境下，常量字符串必须添加 TEXT 或 L 标记宏，如 TEXT("lgh")、L"lgh"。其中 TEXT 等同于__TEXT 宏。在 MFC 类库中，所有字符串函数都有两种版本，一种用于 char 类型，一种用于 wchar_t 类型，并提供一个通用函数可自动替换为对应版本，如 MessageBox 函数实际上是一个宏定义，其宏定义如下所示：

```
#ifdef UNICODE                                  //若使用 Unicode 字符
#define MessageBox  MessageBoxW                 // MessageBox 代表 MessageBoxW 函数
#else                                           //若使用 ANSI 字符
#define MessageBox  MessageBoxA                 // MessageBox 代表 MessageBoxA 函数
#endif
```

在函数定义前使用该函数需要做函数声明，表明该函数确实存在，定义在后面位置。ATOM 实际是 WORD 类型，LRESULT 实际是 LONG 类型，常作为消息函数的返回值。CALLBACK 表示该函数是一个回调函数，回调函数由系统自动调用，无须在程序中手动调用。WPARAM 和 LPARAM 常作为消息的两个附加值的类型，便于根据变量类型即可知道变量的作用，定义如下：

```
typedef UINT WPARAM;
typedef LONG LPARAM;
typedef LONG LRESULT;
```

（2）WinMain 函数的代码如下：

```
int APIENTRY WinMain(HINSTANCE hInstance,                    //程序实例句柄
                     HINSTANCE hPrevInstance,                //在 Win32 中无效
                     LPSTR     lpCmdLine,                    //命令行参数
                     int       nCmdShow)                     //窗口状态
{
    // TODO: Place code here.
    MSG msg;                                                 //消息结构体
    HACCEL hAccelTable;                                      //加速键资源句柄

    //初始化字符串变量
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING); //加载字符资源
    LoadString(hInstance, IDC_API01, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);                                  //注册窗口类

    //应用程序初始化:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_API01); //加载快捷键资源

    //消息循环:
    while (GetMessage(&msg, NULL, 0, 0))                         //获取窗口消息
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  //处理快捷键消息
        {
            TranslateMessage(&msg);                              //翻译消息
            DispatchMessage(&msg);                               //分发消息
        }
    }

    return msg.wParam;                                           //返回值
}
```

在控制台环境下，Main 函数是程序的主函数，在 Win32 桌面软件环境下，WinMain 函数是程序的主函数，程序启动时首先执行该函数，该函数执行结束后，程序也随之结束。APIENTRY 宏指定函数的调用方式，参数 1 的 hInstance 为程序实例的句柄，该句柄值由系统自动创建，传递给函数，参数 3 的 lpCmdLine 为传递给程序的命令行参数，参数 4 的 nCmdShow 为窗口的显示状态，如最大化、隐藏等。

MSG 结构体用于存放 Windows 消息包含的信息，HACCEL 为快捷键资源的句柄，快捷键

也是一种资源，通过组合键可以替代鼠标点击。LoadString 函数用于加载字符串资源到变量中，函数格式如下：

```
int LoadString(HINSTANCE hInstance,UINT uID,LPTSTR lpBuffer,int nBufferMax)
```

参数如下。

- hInstance：程序实例句柄。
- uID：字符串资源的 ID。
- lpBuffer：字符缓冲区的地址。
- nBufferMax：字符缓冲区的大小。

返回值：实际拷贝的字符数目。

LoadAccelerators 函数用于加载快捷键资源，格式如下：

```
HACCEL LoadAccelerators(HINSTANCE hInstance,LPCTSTR lpTableName)
```

参数如下。

- hInstance：程序实例句柄。
- lpTableName：快捷键资源的 ID。

返回值：快捷键资源的句柄。

GetMessage 函数获取窗口接收到的消息，存放到 msg 变量中，格式如下：

```
BOOL GetMessage(LPMSG lpMsg,HWND hWnd,UINT wMsgFilterMin,UINT wMsgFilterMax)
```

参数如下。

- lpMsg：指向消息结构体的指针，存放获取的消息。
- hWnd：接收消息的窗口句柄，若为 NULL 接收该线程所有窗口的消息。
- wMsgFilterMin：接收消息的最小值。
- wMsgFilterMax：接收消息的最大值，若最小最大值都为 0，则接收所有消息。

返回值：若接收到 WM_QUIT 消息，返回 0，若为其他消息，返回非零值，若有异常发生，返回-1。

利用 while 循环持续处理接收到的消息，若没有收到 WM_QUIT 消息，则 while 循环将一直持续下去，程序保持运行，若收到 WM_QUIT 消息，则 while 循环结束，程序也随之结束。

TranslateAccelerator 函数用于处理快捷键消息的转换，TranslateMessage 函数翻译消息包含的信息，DispatchMessage 函数将消息传递给窗口消息处理函数，执行相关功能。

（3）MyRegisterClass 函数代码如下：

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;                                                    //窗口类

    wcex.cbSize = sizeof(WNDCLASSEX);                                   //类大小

    wcex.style          = CS_HREDRAW | CS_VREDRAW;                      //窗口样式
    wcex.lpfnWndProc = (WNDPROC)WndProc;                                //窗口消息处理函数
    wcex.cbClsExtra     = 0;                                            //类扩展信息
    wcex.cbWndExtra     = 0;                                            //窗口扩展信息
    wcex.hInstance      = hInstance;                                    //窗口所在的实例句柄
    wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_API01); //窗口的图标句柄
    wcex.hCursor     = LoadCursor(NULL, IDC_ARROW);                     //窗口的光标句柄
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);                     //窗口的背景画刷
    wcex.lpszMenuName   = (LPCSTR)IDC_API01;                            //窗口的菜单资源 ID
    wcex.lpszClassName  = szWindowClass;                                //窗口类的名称
    wcex.hIconSm     = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL); //窗口的小图标
```

```
    return RegisterClassEx(&wcex);                                    //注册窗口类
}
```

WNDCLASSEX 为窗口类结构体，包含窗口的所有信息，其中 cbSize 指定结构体的大小，style 指定窗口的样式，CS_HREDRAW 表示当窗口水平方向改变时重绘窗口，CS_VREDRAW 表示当窗口垂直方向改变时重绘窗口。lpfnWndProc 指定窗口消息的处理函数，当窗口收到消息时，如鼠标左键按下、键盘按下等，系统自动调用一个回调函数处理该消息。其中该函数的原型如下所示：

```
typedef LRESULT (CALLBACK* WNDPROC)(HWND, UINT, WPARAM, LPARAM);
```

lpszClassName 为创建的窗口类命一个名，RegisterClassEx 函数用于注册一个窗口类，只有注册后，才能调用 CreateWindow 函数根据窗口类创建一个窗口实例。

（4）InitInstance 函数的代码如下：

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;
   hInst = hInstance;                                  //存储程序实例句柄
                                                       //创建窗口实例
   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);                         //显示窗口
   UpdateWindow(hWnd);                                 //更新窗口

   return TRUE;
}
```

CreateWindow 函数用于根据窗口类，创建一个窗口实例，格式如下：

```
HWND CreateWindow(
    LPCTSTR lpClassName,                               //窗口类名称
    LPCTSTR lpWindowName,                              //窗口标题名称
    DWORD dwStyle,                                     //窗口样式
    int x,                                             //左上角 x 值
    int y,                                             //左上角 y 值
    int nWidth,                                        //窗口宽度
    int nHeight,                                       //窗口高度
    HWND hWndParent,                                   //父窗口的句柄
    HMENU hMenu,                                       //菜单句柄
    HINSTANCE hInstance,                               //程序实例句柄
    LPVOID lpParam                                     //传递给 CREATESTRUCT 的自定义值
);
```

szWindowClass 为窗口类名，szTitle 为窗口标题名称，WS_OVERLAPPEDWINDOW 是多个窗口标志的组合，定义如下所示：

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME
| WS_MINIMIZEBOX | WS_MAXIMIZEBOX)
```

CreateWindow 函数创建窗口实例后，返回窗口的句柄值，存放到 hWnd 中。ShowWindow 函数根据 nCmdShow 的值显示窗口，UpdateWindow 函数重绘窗口。

（5）WndProc 函数的代码如下：

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
int wmId, wmEvent;
PAINTSTRUCT ps;                                    //重绘结构体
HDC hdc;                                           //设备环境句柄
TCHAR szHello[MAX_LOADSTRING];
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);     //加载字符串资源

switch (message)                                   //判断消息类型
{
    case WM_COMMAND:                               //若为命令消息
        wmId    = LOWORD(wParam);                  //获取消息附加值
        wmEvent = HIWORD(wParam);
        //判断命名消息的 ID:
        switch (wmId)
        {
            case IDM_ABOUT:                        //若 ID 为 IDM_ABOUT，则显示模态对话框
               DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
               break;
            case IDM_EXIT:                         //若 ID 为 IDM_EXIT，则释放窗口
               DestroyWindow(hWnd);
               break;
            default:                               //默认处理函数
               return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
    case WM_PAINT:                                 //重绘消息
        hdc = BeginPaint(hWnd, &ps);               //获取设备环境
        // TODO: Add any drawing code here...
        RECT rt;
        GetClientRect(hWnd, &rt);                  //客户区矩形
        DrawText(hdc, "Win32 程序", strlen("Win32 程序"), &rt, DT_CENTER);
                                                   //绘制文本
        Rectangle(hdc,20,20,300,300);              //矩形
        Ellipse(hdc,20,20,300,300);                //椭圆
        Ellipse(hdc,80,85,120,100);
        Ellipse(hdc,200,85,240,100);
        Arc(hdc,150,140,170,160, 150,150,170,150);     //绘制圆弧
        Arc(hdc,120,180,200,240 , 120,220,200,220);
        EndPaint(hWnd, &ps);                       //结束绘制
        break;
    case WM_DESTROY:                               //窗口销毁消息
        PostQuitMessage(0);                        //发送 WM_QUIT 消息
        break;

    default:                                       //默认的消息处理函数
        return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

WndProc 是回调函数，由系统自动调用。参数 1 的 hWnd 为窗口句柄，参数 2 的 message 为消息类型，参数 3 和 4 的 wParam、lParam 为消息的详细信息。

PAINTSTRUCT 结构体为存放重绘窗口的信息，HDC 为设备环境的句柄值。switch 判断消息的类型，若为 WM_COMMAND 命令消息，该消息通常由菜单、工具按钮发出，wParam 和 lParam 的值在不同的消息类型里有不同的意义，LOWORD 和 HIWORD 获取 wParam 里低位和高位的值，分别代表不同的意义，视消息类型而定。

如 WM_COMMAND 消息里，LOWORD(wParam)获取发出消息的 ID 值，switch 判断 ID 值，若为 IDM_ABOUT，调用 DialogBox 函数显示模态对话框，函数格式如下：

```
INT_PTR DialogBox(HINSTANCE hInstance,LPCTSTR lpTemplate,HWND hWndParent , DLGPROC
lpDialogFunc )
```

参数如下。

- hInstance：程序实例的句柄
- lpTemplate：对话框模板 ID。
- hWndParent：父窗口的句柄。
- lpDialogFunc：窗口消息的处理函数。

返回值：传递给 EndDialog 函数的参数值。

若为 IDM_EXIT，则调用 DestroyWindow 函数释放窗口资源，并向窗口发送 WM_DESTROY 消息。WM_PAINT 表示窗口重绘消息，BeginPaint 函数获取窗口的设备环境，即窗口的画布，用于在窗口上绘制图形，GetClientRect 函数获取窗口客户区的矩形大小，DrawText 函数绘制文本，Rectangle 函数绘制矩形，Ellipse 函数绘制椭圆，Arc 函数绘制圆弧，EndPaint 函数结束窗口的绘制，释放设备环境。

WM_DESTROY 表示窗口释放消息，PostQuitMessage 函数向窗口发送 WM_QUIT 消息，WinMain 函数收到该消息后，停止 while 循环，结束程序。DefWindowProc 是系统自定义的默认消息处理函数，可用来处理各种消息。

（6）About 函数的代码如下：

```
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)                                    //消息类型
    {
        case WM_INITDIALOG:                             //对话框初始化消息
            return TRUE;

        case WM_COMMAND:                                //命令消息
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam)); //关闭模态对话框
                return TRUE;
            }
            break;
    }
    return FALSE;
}
```

About 函数用于子窗口的消息，WM_INITDIALOG 为对话框的初始化消息，WM_COMMAND 为控件的命令消息，若点击 IDOK 或 IDCANCEL 按钮，调用 EndDialog 函数关闭模态对话框。

Tips 模态对话框必须使用 EndDialog 函数关闭，其中 EndDialog 的第 2 个参数值为 DialogBox 函数的返回值。

附录 B　程序调试技巧

一个功能强大、安全健壮的程序在开发过程中需要反复调试，调试是软件开发不可缺少的一步，尤其是 MFC 这种异常复杂的框架，需要进入各个函数内部，查看函数的功能实现，以及变量的变化状态，从而掌握程序的流程。在单线程程序中，程序的代码是一行行执行的，调用某函数时，流程进入函数内部，执行结束后，跳出函数体，回到函数调用处。

Visual C++提供一个 Debug 工具条专门用于程序调试，如图 B-1 所示。

其中 表示进入函数内部， 表示执行下一步，但不进入函数内部， 表示跳出函数体，回到函数调用处。在实际调试过程中，点击按钮比较麻烦，常使用快捷键代替，如 F9 键设置断点，F10 键单步执行，F11 键进入函数内部，F5 键断点调试。以第 3 章创建的 Win32 工程 API01 为例，介绍程序的调试技巧。

（1）打开 Win32 工程 API01，在如下位置所在行点击鼠标，按下 F9 键，添加 4 个断点，如图 B-2，B-3，B-4 所示。红色的圆点表示一个断点。

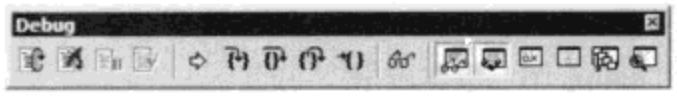


图 B-1　Debug 工具条

```
case IDM_ABOUT:
    DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
    break;
case IDM_EXIT:
    DestroyWindow(hWnd);
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
```

图 B-2　断点 1、2

```
case WM_DESTROY:
    PostQuitMessage(0);
    break;
```

图 B-3　断点 3

```
case WM_COMMAND:
    if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
    {
        EndDialog(hDlg, LOWORD(wParam));
        return TRUE;
```

图 B-4　断点 4

（2）按 F10 键单步调试，启动程序，并在 WinMain 函数处停住，如图 B-5 所示。

WinMain 函数是 Win32 程序的主函数，按 F10 键执行单步调试后，启动程序，并在程序的第一个调用位置处停下。黄色箭头表示当前流程所在位置。

（3）继续按 F10 键，一条一条语句的向下执行，在代码下方会显示两个变量监测窗口，可以查看变量的值，如图 B-6 所示。

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    // TODO: Place code here.
    MSG msg;
    HACCEL hAccelTable;
```

图 B-5　程序启动点 WinMain 函数

```
// Initialize global strings
LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadString(hInstance, IDC_API01, szWindowClass, MAX_LOADSTRING);
MyRegisterClass(hInstance);
```

Context: WinMain(HINSTANCE__ *, HINSTANCE__ *

Name	Value
hAccelTable	0xcccccccc
hInstance	0x00400000
msg	{msg=0xcccccccc wp=0xcccccccc lp=0xcccccccc}
szTitle	0x00423628 "API01"
szWindowClass	0x004235c4 ""

Name	Value
szTitle	0x00423628 "API01"
nCmdShow	1

图 B-6　变量查看窗口

左边的 Watch 窗口用来查看变量的值，当流程执行到某句代码时，自动显示相关变量的值，如 hAccelTable 的值为 0xcccccccc，szTitle 的值为 API01，由于尚未为 szWindowClass 赋值，szWindowClass 的值为空。每个变量都可以展开，如结构体 msg 查看各个成员的值。

右边的 Variable 窗口用来查看指定变量的值，在 Name 列输入变量的名称，如 nCmdShow，Value 列自动对应变量的值，如 1。也可直接将鼠标放到变量名称上，自动显示该变量的值或地

- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

址，若地址为 0x00000000，表示该变量尚未创建，若此时使用该变量，程序会报错。

（4）按 F10 键，当箭头在 MyRegisterClass 函数前时，按 F11 键，进入函数内部，如图 B-7 所示。

（5）按 Shift+F11 组合键，跳出函数体，回到函数调用处，如图 B-8 所示。

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
```

图 B-7　进入函数内部

```
    LoadString(hInstance, IDC_API01, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
```

图 B-8　跳出函数体

（6）按 F5 键，跳至下一个断点，若要执行的代码没有添加断点，显示程序界面，如图 B-9 所示。

在程序的操作过程中，若要执行的某行代码包含有断点，则进入代码界面，在断点处停下。

（7）选择 Help|About 命令，在要执行的代码的断点处停下，如图 B-10 所示。

图 B-9　程序界面

```
        switch (wmId)
        {
            case IDM_ABOUT:
                DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
```

图 B-10　命令断点

（8）按 F5 键，跳过该断点，调用 DialogBox 函数显示子对话框，如图 B-11 所示。

（9）点击 OK 按钮，关闭子对话框，在断点处再次停下，如图 B-12 所示。

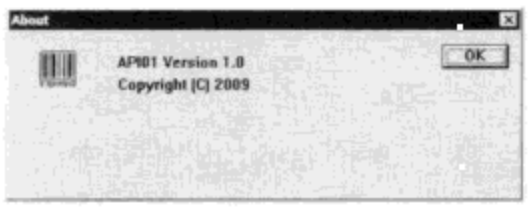


图 B-11　子对话框

```
    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return TRUE;
        }
        break;
```

图 B-12　关闭断点

（10）按 F5 键跳过该断点，调用 EndDialog 函数关闭子对话框。

（11）选择 File|Exit 命令，在调用代码的断点处停下，如图 B-13 所示。

（12）按 F5 键跳过该断点，调用 DestroyWindow 函数销毁窗口，并发出 WM_DESTROY 消息，在下一个调用断点处再次停下，如图 B-14 所示。

```
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
```

图 B-13　退出断点

```
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
```

图 B-14　销毁断点

（13）按 F5 键，调用 PostQuitMessage 函数发送 WM_QUIT 消息，退出消息循环，结束程序运行。

Tips　程序调试是一项开发必备的技术，或者说，好程序都是调出来的，通过程序调试，能掌握程序每一步的运行状态，及早找到问题所在。有时候程序会报错，但一时很难确定哪个地方出了问题，这时可在可能错误的地方添加多个断点，逐个断点调试运行，在调试过程中可确定错误代码的位置，找到有问题的代码，修改程序的 Bug。