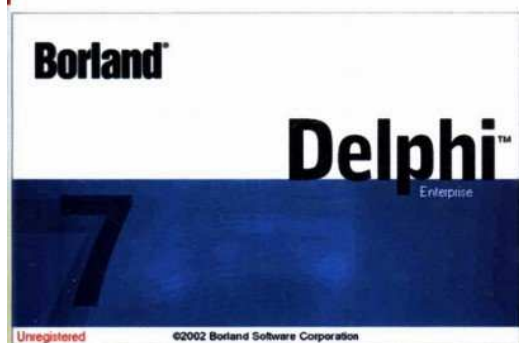


零点起航



—— 计算机编程系列教材

Delphi 7



基础教程

◎零点工作室 宋一兵 赵景波 李春艳 等编著



机械工业出版社
CHINA MACHINE PRESS

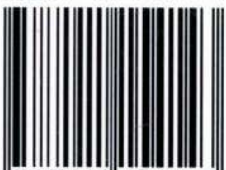
零点起航

零点起航 —— 计算机编程系列教材

- 《Visual Basic.NET基础教程》
- 《Visual C++.NET基础教程》
- 《PowerBuilder 9.0基础教程》
- 《Java 2基础教程》
- 《G++Builder 6.0基础教程》
- 《Delphi 7基础教程》
- 《Visual FoxPro 7.0基础教程》



ISBN 7-111-15414-2



9 787111 154143 >

定价: 29.50 元

● ISBN 7-111-15414-2/TP·4020(课)

地址: 北京市百万庄大街22号 邮政编码: 100037

联系电话: (010) 68326294

网址: <http://www.cmpbook.com>

E-mail: online@cmpbook.com

前 言

Delphi 7 是 Inprise 公司（原 Borland 公司）推出的一个面向对象的快速应用程序开发工具，它以可视化的开发环境、简洁明快的编程语言、功能强大的组件、优化的源代码编译器、可扩展的数据库访问引擎、稳定性和高效性等特点，成为一个成熟的功能全面的产品，赢得了越来越多的用户。

Delphi 7 继承了 Pascal 语言的严谨结构和优雅风格，以开放式的环境，完全地支持面向对象程序设计等诸多特性，成为 Windows 环境下首选的开发工具。

本书主要介绍了开发 Delphi 应用程序的多种基本要素和必备知识，包括 Delphi 的可视化开发环境、Object Pascal 语言、面向对象程序设计的基本概念、窗体和各种功能的组件、MDI、调试技术与异常处理、图形图像处理、多媒体应用、数据库开发和 Internet 编程等。

本书从基础入手，深入浅出，在功能讲解的同时，配合大量的实例，力求使读者能快速、轻松地学会用 Delphi 编程，能充分理解面向对象程序设计的内涵，能用本书提供的实例解决实际问题。

根据应用性人才的培养目标，本书注重理论与实践相结合，注重基础知识的理解与基本技能的培养。在理论“必需、够用为度”的前提下，突出实用性，突出实践性环节。强调“边学边做”，使读者每学习一点儿知识，就能够通过具体的编程练习得到锻炼，同时也加深了对内容的理解。

本书是 Delphi 7 程序设计的入门书，特别适合于高校教师、学生和初级程序设计人员使用。读者使用本书不需要预先具有任何编程经验，但是如果对 Pascal、C、C++ 或者 Visual Basic 等有所了解，那么将会对本书的学习非常有益。

本书 1~4 章由李春艳老师编写，8~12 章由赵景波老师编写，其余内容由宋一兵编写并负责全书统稿。本书在编写过程中，得到了许多同事和朋友的支持，隋成城、张书钦、张宪海、汪学清、刘丽莉、王臣业、林新志、杨兴河、张忠林、李忠伟等参与了内容的编写和程序的调试工作，在此一并表示感谢。

由于时间匆忙、水平有限，书中难免存在错误和疏漏之处，恳请读者批评指正。

书中实例程序的源码可以从零点工作室网站（<http://www.zerobook.net>）获得。

编者

2004 年 6 月



系列教材序言

随着计算机技术的飞速发展,各种应用件也在社会和领域得到了广泛应用,软件的设计和开发也成为日常工作中的一部分。对于从事计算机方面工作的人员,掌握一种或几种编程工具是其必须具备的专业技能,也是胜任工作的基本条件。同时,很多高校学生、电脑爱好者也希望自己能掌握基本的软件设计方法,以满足未来工作和学习的需要。

零点起航——计算机编程系列教材选择了目前常用的编程软件,涵盖了应用程序设计、数据库开发和网络编程开发工具,着眼于大专院校教师、学生和工程技术人员的自学和培训。整套教材由7本书组成:

- (1)《Visual Basic.NET 基础教程》
- (2)《Visual C++.NET 基础教程》
- (3)《PowerBuilder 9.0 基础教程》
- (4)《Java 2 基础教程》
- (5)《C++Builder 6.0 基础教程》
- (6)《Delphi 基础教程》
- (7)《Visual FoxPro 7.0 基础教程》

整套教材本着从零末始的思想,从基础培训的角度入手,在内容的选限和章节的设置上充分考虑了初学者的实际需要,力求简明清晰、通俗易懂。在详细讲解软件功能和用法的同时,引导读者练习一些针对性、耐用性很强的程序实例,以加深内容的理解。在每章的最后,都附带了一些习题,通过对这些习题的思考和练习,读者可以对该章所学内容有更深刻的认识。该套教材的作者都是长期从事计算机教学和软件设计的高校教师和专业人员,具有丰富的理论和实践经验,在写作过程中融入了多年的经验和体会,为初学者提出了许多有益的建议。

零点起航——计算机编程系列教材中的各教材自成体系,读者可以根据自己的实际需要选择。

希望这套教材对您的学习、工作和生活有所帮助。

零点工作室网站: <http://www.zerobook.net>

主编邮箱: guandianzhu@qdcnc.com

零点工作室

2004年6月

PDG

目 录

系列教材序言

前言

第 1 章 Delphi 基础知识	1	第 5 章 基本窗体设计	81
1.1 Delphi 的特点	1	5.1 Form (窗体) 组件	81
1.2 Delphi 7 的集成开发环境	2	5.2 向窗体中添加组件	86
1.3 实例——Hello, world!	5	5.3 单文档窗体	89
1.4 小结	9	5.4 多文档窗体	91
第 2 章 Object Pascal 语言基础	10	5.5 小结	92
2.1 Delphi 的编程风格	10	第 6 章 基本组件的应用	93
2.2 标识符	12	6.1 组件的概念	93
2.3 数据类型	14	6.2 文本输入组件	96
2.4 赋值语句	21	6.3 按钮及分类组件	103
2.5 过程与函数	23	6.4 列表组件	110
2.6 控制语句	25	6.5 滚动组件	121
2.7 实例——随机加减法测试	29	6.6 表格组件	125
2.8 小结	35	6.7 日期和时间组件	127
第 3 章 面向对象编程	36	6.8 多选项卡组件	130
3.1 OOP 基本概念	36	6.9 小结	133
3.2 类的基本概念	37	第 7 章 用户界面设计	135
3.3 类的封装	40	7.1 菜单设计	135
3.4 类的继承性	47	7.2 工具栏与状态栏	144
3.5 类的多态性	49	7.3 信息对话框	148
3.6 异常处理	52	7.4 对话框组件	152
3.7 小结	56	7.5 实例——文档编辑器	164
第 4 章 应用程序开发框架	57	7.6 小结	170
4.1 Delphi 的文件结构	57	第 8 章 图形图像技术	171
4.2 单元文件的内部结构	61	8.1 画布技术	171
4.3 项目管理	64	8.2 图形图像的种类和组件	180
4.4 应用程序类 TApplication	68	8.3 图形图像编程应用	183
4.5 项目的编译和调试	71	8.4 使用鼠标绘制图形	196
4.6 小结	80	8.5 小结	198

第9章 多媒体技术	199	11.2 数据控制组件	269
9.1 多媒体基础知识	199	11.3 人事管理系统开发	278
9.2 多媒体组件	200	11.4 小结	294
9.3 多媒体编程	214	第12章 报表和图表设计	295
9.4 小结	220	12.1 数据报表的设计	295
第10章 数据库基础	221	12.2 图表设计	299
10.1 数据库简介	221	12.3 小结	304
10.2 SQL 结构化查询语言	223	第13章 网络编程技术	305
10.3 Delphi 访问数据库的机制	227	13.1 网络基础知识	305
10.4 Delphi 的数据库管理工具	232	13.2 TCP/IP 编程	306
10.5 建立第一个数据库应用程序	247	13.3 使用网络函数编程	322
10.6 小结	248	13.4 小结	336
第11章 开发数据库应用程序	249	参考文献	337
11.1 数据集组件	249		



第 1 章 Delphi 基础知识

作为一款优秀的应用程序开发工具，Delphi 受到了广大程序设计人员的青睐，在各个领域得到了广泛应用。Delphi 7 是 Borland/Inprise 公司的主要作品之一，是基于对象 Pascal 语言的 RAD (Rapid Application Development, 快速应用程序开发) 工具，具有良好的可视化应用程序开发环境和强大的可扩展数据库功能。利用 Delphi 7 提供的 VCL (Visual Component Library, 可视化组件库) 进行编程，可以快速、高效地开发出基于 Windows 环境的各类应用程序。

1.1 Delphi 的特点

人们如此形容 Delphi：真正的程序员用 C++，聪明的程序员用 Delphi。与其他编程软件相比，特别是在数据库和网络应用方面，Delphi 具有简便易用、灵活高效的特点。

1. 强大的可视化编程环境

Delphi 的集成开发环境设计得非常简洁、明快，整个屏幕的各种窗口都经过精心安排，分布紧凑合理；而且用户可以定制桌面设置，建立一个符合自己风格的用户界面。

Delphi 7 具有非常优秀的窗体设计器，它是建立在一个真正面向对象的框架结构基础之上。这样，对基类所作的改变都将会传递给所有的派生类。在窗体设计器中进行设计时，Delphi 7 会自动在后台为窗体中的组件生成代码。

Delphi 7 的编辑器和其他工具的编辑器基本功能相差不多，但其 Code Insight 技术却省去了许多输入工作的麻烦，避免了记忆组件种类繁多的方法和属性。而且系统还能够自动对用户输入的函数和过程的参数列表给出提示。

Delphi 7 的调试器功能可与 Visual C++ 相媲美，除了能在编辑器中通过设置断点和监控点等来调试程序外，还具备了许多先进的功能，如远程调试、过程关联、DLL 和包调试、自动本地监控以及 CPU 窗口等。

2. 高效的编译器

开发软件通常要经过修改源代码、编译、测试、再修改、再编译、再测试等操作，这些操作形成了一个开发循环，所以快速的编译器可以大大降低开发周期。如果读者用过 Visual C++ 或者 C++ Builder，对其比较慢的编译速度可能会有较深的印象。

Pascal 编译器以其编译速度快的特点而闻名，而 Delphi 正是建立在这种高效编译器的基础之上，是针对 Windows 的最快的高级语言代码编译器。即便是 C++ 编译器近年来取得了很大的进步，增加了链接和各种缓存策略，其编译速度仍远较 Delphi 低。

Delphi 和 C++ Builder 共享同一种编译器后端, 因此生成的代码也十分精练高效, 使 Delphi 程序的运行速度更快。

3. 丰富的 VCL

VCL 是 Delphi 7 最重要的组成部分。VCL 包含丰富的不同种类的组件, 大大方便了软件的开发工作。读者还可以通过网络获得很多其他类型的组件, 并将其添加到组件面板上, 如同 Delphi 7 自带的组件一样使用。

在程序设计时操作组件、创建组件、使用面向对象技术继承其他组件的行为和能力, 是决定 Delphi 程序开发效率的关键因素。

1.2 Delphi 7 的集成开发环境

Delphi 7 是目前的最新版本, 它所提供的可视化集成开发环境 (IDE) 使程序员能够轻松设计出图形化的应用程序。

启动 Delphi 7 后, 其集成开发环境如图 1-1 所示, 主要包括菜单栏、工具栏、组件面板、窗体设计器、代码编辑器、对象监视器、对象树形查看器等几种。

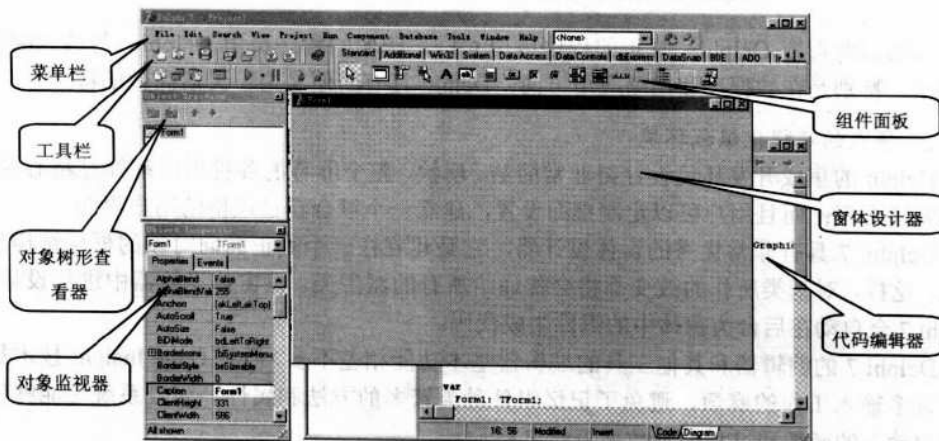


图 1-1 Delphi 7 的集成开发环境

1. 菜单栏

菜单栏提供了 Delphi 集成开发环境的所有功能。菜单栏除包含标准 Windows 程序通常所具有的【File】、【Edit】、【Help】等菜单外, 还有一些与 Delphi 7 项目、运行、辅助工具和环境配置等相关菜单。其中的菜单命令将在后面的学习中结合具体应用来讲解。

2. 工具栏

工具栏中包含了一些常用操作的快捷菜单, 各个按钮的功能如图 1-2 所示。鼠标指针在某个工具按钮上停留片刻就会出现 Tooltip (工具提示)。

工具栏中的快捷按钮可以由用户按自己的需要自由添加或删除。工具栏也可以进行个性化设置。选择【View】/【Toolbars】/【Customize】菜单命令, 会出现如图 1-3 所示的对

话框。通过该对话框，用户可以自定义工具栏中的栏目。

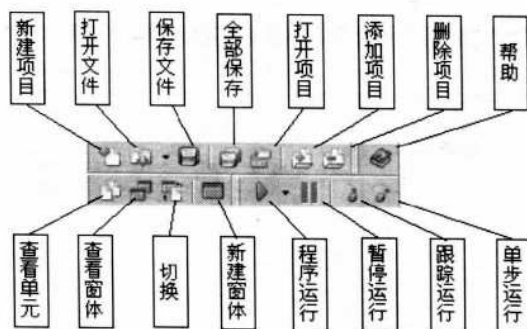


图 1-2 工具栏中按钮的功能

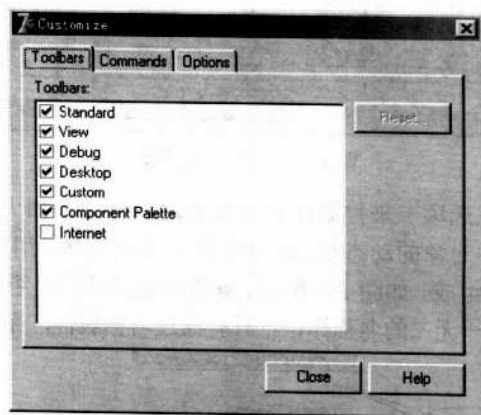


图 1-3 自定义工具栏栏目

3. 组件面板

组件面板是一个以选项卡形式显示的工具栏，其中包含了 Delphi 7 自带的 VCL 组件和用户安装的 ActiveX 组件，主要有【Standard】、【Additional】、【Win32】等选项卡，每个选项卡又包括若干以图标形式表现的组件。组件是建立一个应用程序最常用的元素。

用户可以自己定制组件栏的页面和各页面所包含的组件。选择【Tools】/【Environment Options】菜单命令，打开系统的环境设置对话框，其【Palette】页面如图 1-4 所示，页面左边为组件面板的各个选项卡，右边是该选项卡包含的组件。通过对话框中的几个按钮，可以添加和删除选项卡，也可以对选项卡进行更名操作。除了隐藏选项卡中的组件，还可以添加组件到指定的选项卡，如果是其他选项卡中的组件，可以简单地通过鼠标的拖放操作来完成。如果是从组件库中选择组件，就需要先构造和安装组件包。

4. 【Object Inspector】(对象监视器)

对象监视器是用来描述组件及窗体对象的属性特征和行为事件的。它是应用程序设计

过程中最重要的一个工具。其功能主要有以下两方面，一是修改窗体及其包含的对象的属性，二是通过自动代码生成机制使对象响应相应的事件。

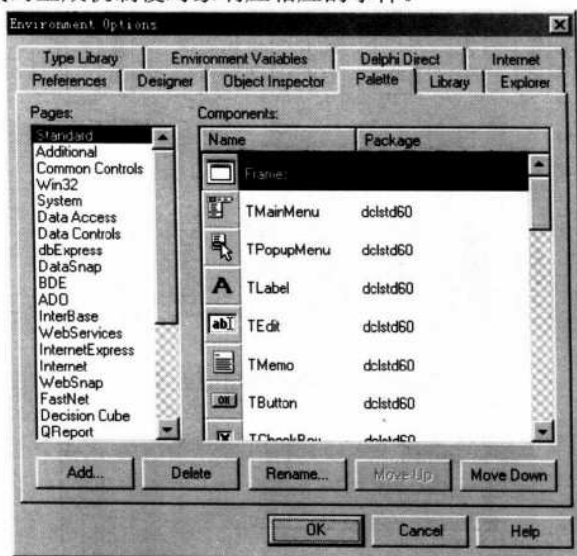


图 1-4 系统环境设置

对象监视器通过属性选项卡来控制组件对象的静态行为，如大小、色彩和名称等；通过事件选项卡来控制组件对象的动态行为，如组件打开或关闭、用户在组件上单击鼠标等。

对象监视器由 3 部分组成，如图 1-5 所示。属性中最重要的是组件的【Name】（名称）属性，每个对象都有自己独一无二的名称标识。只有通过名称属性，系统才能识别不同的对象。

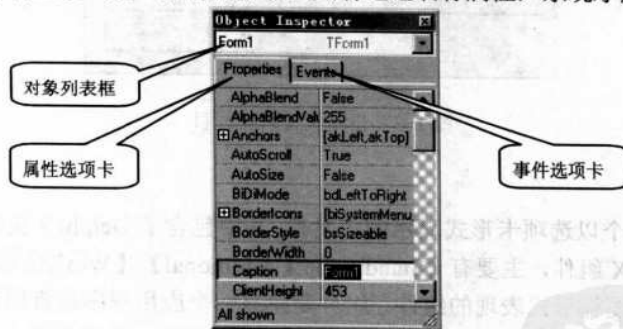


图 1-5 对象监视器窗口

5. 窗体设计器

窗体是放置 VCL 中各类组件对象的地方。通过所见即所得的窗体设计器，程序员可以在窗体中绘制各式各样的程序界面，用户通过界面操作，就可与应用程序交互。

窗体设计的主要步骤就是向窗体中添加可以完成应用程序任务的组件；更改组件的属性使之满足程序的设计要求；将组件的事件与程序代码相联系，用代码实现组件的活动。

6. 【Object Treeview】(对象树形查看器)

对象属性查看器能够以目录树的形式显示整个工程中使用到的对象。在这里选择对象,就如同在窗体设计器中选择对象一样。

7. 代码编辑器

代码编辑器又称单元窗口,是编写程序代码的地方,如图 1-6 所示。它包括两个密切相关的部分:单元管理窗口显示的是程序的结构组成,包括类定义、变量和常量的定义部分以及引用声明部分;代码编辑窗口是用来编辑源程序的地方。

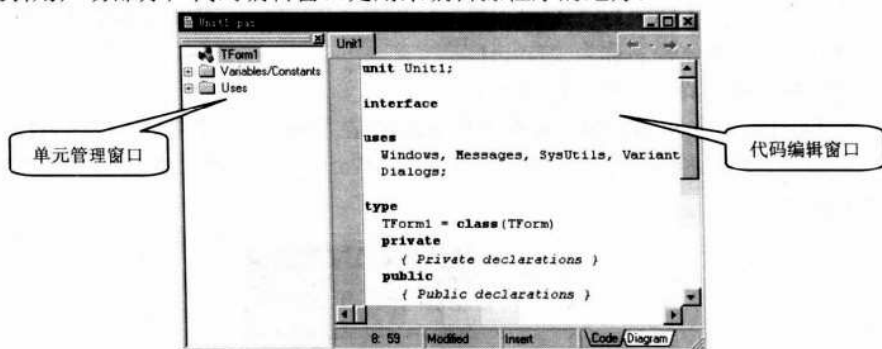


图 1-6 代码编辑器窗口

代码编辑器一般隐藏在窗体设计器下面,可以通过以下几种方法显示它:

- 选择【View】/【Code Explorer】菜单命令或【View】/【Toggle Form/Unit】菜单命令;

- 单击快捷工具栏中的【Form/Unit】转换按钮;
- 在集成开发环境中用鼠标单击代码编辑器窗口。

在窗口内单击鼠标右键会弹出快捷菜单,其中包含了代码编辑和调试的一些命令。

1.3 实例——Hello, world!

Delphi 7 应用程序开发的基本步骤为:

- 建立窗体;
- 在窗体上添加组件;
- 检查和设定对象属性;
- 编写响应事件处理程序;
- 保存文件;
- 编译、运行程序。

下面以一个简单的实例来说明 Delphi 程序的结构。

【例 1-1】Hello, world!

程序效果: 程序中包含两个按钮,单击“问候”按钮,会出现“Hello, world!”的文字;单击“退出”按钮,就结束程序。画面如图 1-7 所示。

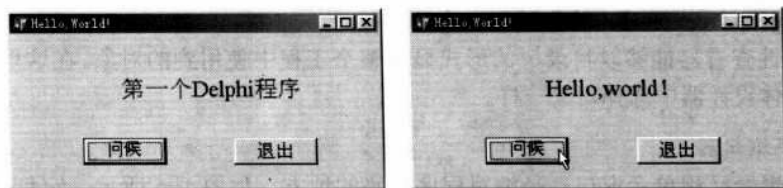


图 1-7 第一个 Delphi 程序

(1) 选择【File】/【New】/【Application】菜单命令，建立一个新的应用程序。此时，Delphi 自动生成一个窗体，名称为“Form1”。

(2) 拖动窗体的边界，调整窗体的大小。

(3) 在【Object Inspector】中，可见当前显示的是“Form1”对象的属性和方法。单击【Caption】属性，删除原来的文字，输入“Hello, World!”，如图 1-8 所示。修改窗体标题栏中的文字内容。



图 1-8 修改窗体标题栏中的文字内容

(4) 在组件面板的【Standard】选项卡中，选择“Button”组件；然后在窗体中，按住鼠标，拖出一个虚线方框；松开鼠标，则按钮出现，可见按钮上的文字为“Button1”，如图 1-9 所示。

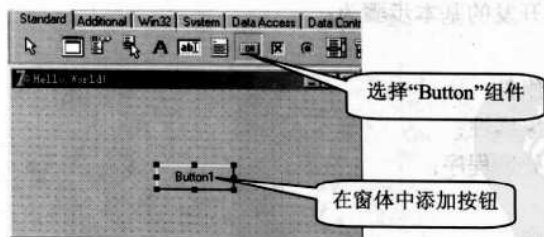


图 1-9 在窗体中添加按钮

提示：在“Button”组件上双击鼠标，会直接在窗体中建立一个组件对象。

(5) 选择窗体中的按钮，在【Object Inspector】中，可见当前显示的是“Button1”对

象的属性和方法。单击【Caption】属性，删除原来的文字，输入“问候”，修改按钮上显示的文字。


(6) 在【Font】属性后面单击按钮，会出现一个设置字体的对话框。在这里设置按钮上显示的文字样式，如图 1-10 所示。



图 1-10 设置按钮文字和文字样式

(7) 同理，再在窗体中添加一个按钮，并修改按钮文字为“退出”，字体样式同“问候”按钮。

(8) 在组件面板的【Standard】选项卡中，选择“Label”（标签）组件，然后在窗体中建立一个标签对象；修改对象的文字为“第一个 Delphi 程序”，并设置文字字体和颜色，如图 1-11 所示。

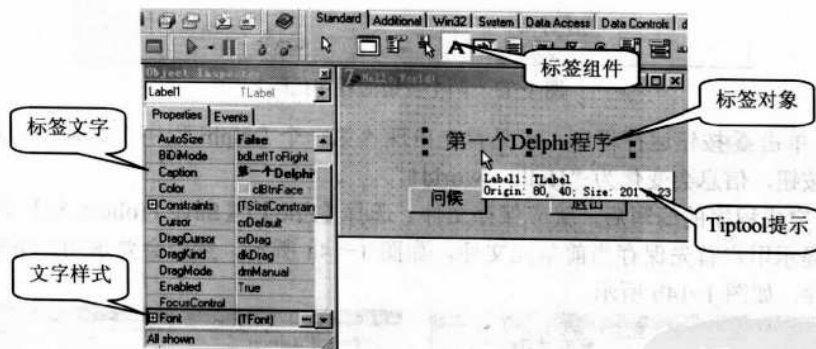


图 1-11 建立标签对象并修改其属性

(9) 在窗体上双击“问候”按钮对象，会显示出代码编辑窗口，如图 1-12 所示。可见当前已经有一些程序代码，这是系统自动创建的程序结构代码；而且，虽然前面修改了按钮的文字，但是并没有改变按钮的名称，所以按钮名称仍为默认的“Button1”。

提示：在【Object Inspector】窗口选择【Event】选项卡，双击“OnClick”事件右侧区域，也会打开代码编辑窗口；而且这里有更多的事件可供选择。

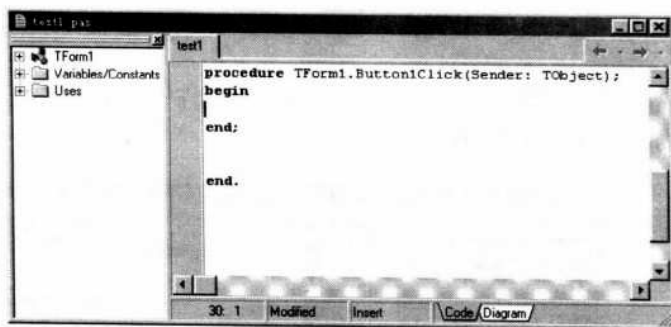


图 1-12 代码编辑窗口

(10) 在当前光标位置输入如下代码，定义单击按钮后显示一句问候：

```
Label1.Caption:='Hello,World!'
```

(11) 同理，双击“退出”按钮对象，在代码编辑窗口输入代码“Close”，如图 1-13 所示。定义单击按钮退出程序。

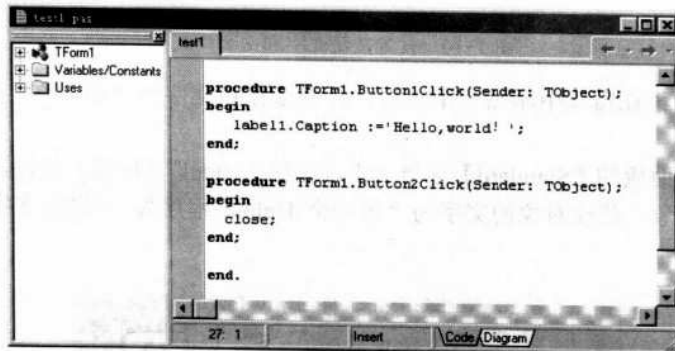



图 1-13 两个按钮的程序代码

(12) 单击  按钮运行程序，可见首先出现“第一个 Delphi 程序”这样的信息；单击“问候”按钮，信息会变化为“Hello, world!”。

(13) 完成程序的编写后，就应保存文件。选择【File】/【Save Project As】菜单命令，Delphi 会提示用户首先保存当前单元文件，如图 1-14a 所示。然后会要求用户继续保存当前工程文件，如图 1-14b 所示。

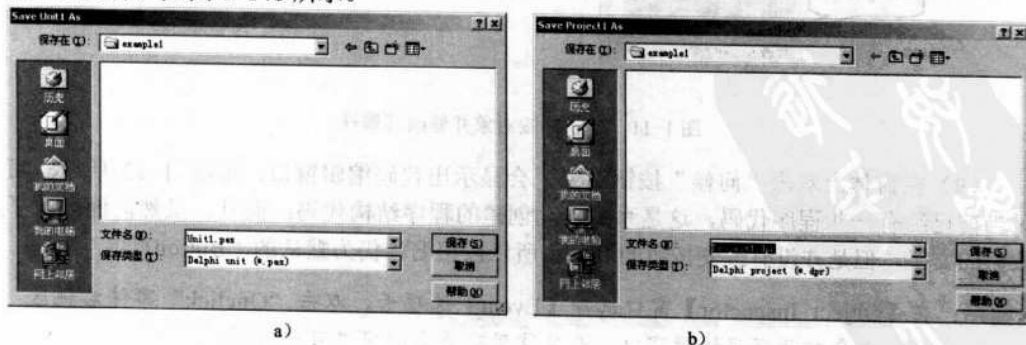


图 1-14 保存单元文件和工程文件

提示：工程文件和单元文件的名称不能一样，否则会出现错误警告。

程序最终代码如下。为了与系统自动建立的代码相区别，这里将我们自己添加的代码用带底纹的斜体文字表示出来。

```
unit test1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Hello, world!'; //添加的代码
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  close; //添加的代码
end;
end.
```

1.4 小结

本章主要介绍了 Delphi 7 的基本知识，使读者对 Delphi 7 有了一个大致的了解。通过一个简单的实例，使大家对 Delphi 的简单易用有了一个感性的认识。集成开发环境是 Delphi 的操作界面，认识和了解 IDE 的组成及各部分的作用是进行程序设计的基础。对于后面的

第2章 Object Pascal 语言基础

Object Pascal 是在 Pascal 的基础上发展起来的，它不仅继承了 Pascal 语言语法结构严谨、编译代码效率高等优点，而且使程序更加易学易用；同时，使用编译器创建的应用程序只生成单个可执行文件（.EXE）。正是这些特点，使得 Object Pascal 很适合作为基础的软件开发语言，从而成为 Delphi 这种先进开发工具的编程语言。

本章将讨论 Object Pascal 的基本知识，并讲解如何在事件处理过程和其他应用程序中，使用它来编制程序代码。讲解的重点在于 Object Pascal 的语法基础，而不是 Pascal 语言的一切细节。

2.1 Delphi 的编程风格

所谓编程风格，就是指程序员常用的编程习惯和文本样式。Pascal 程序具有良好的可读性，但是通过遵循一定的编程风格，会使程序更加便于理解、便于维护。

2.1.1 注释语句

Object Pascal 支持三种类型的注释：

- 花括号注释：组合符号“{”和“}”的成对使用表示它们之间的内容是注释部分。
- 圆括号+星号注释：组合符号“（*”和“*）”的成对使用表示它们之间的内容是注释部分。上面两种注释样式又称为块注释。
- c++风格的双斜杠注释：符号“//”的单独使用表示后面的内容是注释部分。这种注释样式又成为行注释。

请看下面的例子：

```
{ 花括号注释 }  
(* 圆括号+星号注释 *)  
// C++风格的注释
```

前两种注释在本质上是相同的，编译器把处于限定符头和限定符尾中间的内容当作注释。花括号和“圆括号+星号”相比，较适合在大段注释时使用。如果在“{”或“（*”后面是一个“\$”符号时，表示该句为一个编译器指令，与普通的注释不同，通常用来对编译过程进行设置。

对于 C++风格的注释来说，双斜杠后面到行尾的内容被认为是注释。比较适用于单行和少量几行注释的情况。

虽然不同类型的注释在语法上是合法的，但不建议这样做。另外，相同类型的注释不能嵌套使用。下面是一些注释语句嵌套的例子：

```

{ (*这是合法的*) }
{* {这是合法的}* }
{ (*这是非法的*) *}
{{这是非法的}}

```

2.1.2 对象命名

在 Pascal 中,经常需要对变量、函数、过程等用户自定义的对象进行命名。Pascal 对于字母的大小写是不敏感的,比如变量 MyName、Myname、myname 都是指同一个变量。

为了使程序中的对象更容易理解,增加程序的可读性,建议大家养成良好的命名习惯:

- 用意义明确的英文单词和词组命名。
- 单词首字母大写。
- 若需要用多个单词命名一个对象,则单词之间不能插入空格,且每一个单词的首字母均大写(这种方式也被称为驼峰格式)。

例如:

```

asdfas           //没有意义
myclassname      //可读性差
MyClassName      //良好的命名方式

```

注意:名称的第一个字母尽量不要用 T、F、I 这三个字母,这是英文 T 开头的命名一般百表一个类,而 F 往往用于类中私有成员的命名, I 则表示此命名为接口 (Interface)。

2.1.3 代码缩进

当程序较长时,代码按照某种格式排列,会使程序更加便于阅读和理解。一般采用的都是代码缩进的方式,即将程序代码按照所处的级别不同在代码行前面添加空格。通常,使代码的每一级向内缩进 2 格,也就是说,将第 1 层的代码缩进两个空格,第 2 层的代码缩进 4 个空格...

下面的代码说明了这种代码缩进的情况。

```

Procedure SetEmployeeSalary(name:String;salary:Shortint)
Var
    Temp: String;
    NewSalary: Shortint;
Begin
    Temp:=name;
    ...
    If temp='Tom' then
    begin
        NewSalary:=salary+1000;
        ...
    end
end

```



```
end;
end.
```

2.2 标识符

Object Pascal 语言使用的是 ASCII 码字符集, 包括字母 A~Z、a~z、数字 0~9 和其他的一些标准字符。

在对象 Pascal 语言中, 标识符 (Identifiers) 用来标志常量、变量、类型、属性、对象、过程、函数、程序、单元、库和组件库等。标识符可以由任意长的一个不带空格的字符串组成, 但是只有前面的 255 个字符有效。标识符的第一个字符必须是字母或下划线, 其余字符可以是字母、数字或下划线。一般标识符可以由一个或多个具有适当意义的英文单词组成。

下面的单个字符作为特殊符号存在:

```
# $ % ^ \ ( ) + , ? . / : ; < = > @ [ ] ^ { }
```

另外, 有些特殊符号使用的是成对的字符, 具体有:

```
(* (. *) .) .. // := <= >= <>
```

Pascal 语言对于字母的大小写不敏感。当然, 字符串中的大小写字母是需要分清的。在编写程序的时候要保持一定的风格, 不要有时候全部大写标识符, 另一些时候又实行大小写混写。一般都是将标识符中每个英文单词的首位字母大写, 其余字母小写。

如下的标识符为正确命名的标识符:

```
myvar, I, MySource, Name10
```

如下的标识符为错误命名的标识符:

```
1myvar      //数字不能放到最前面
end         //系统标识符
```

2.2.1 变量

变量是程序中数据的临时存放容器, 是程序设计最基本、最重要的元素。变量是程序代码中代表一个内存地址的标识符, 而此地址内存所记录的内容在程序代码执行时可以被改变。

1. 变量的声明

在使用变量前必须对它进行说明, 即对它进行命名, 并说明它的类型。在所有变量说明以前加上保留字 var。变量说明左边是变量的名称, 右边则是该变量的类型, 中间用“:”隔开。变量声明最基本的语法是:

```
var VariableName:type;
```

如可以使用下面的方法声明变量:

```
Var
    Value, Sum :Integer;    //定义了两个整型变量
    Line :String;          //定义了一个字符型变量
```


2. 局部变量和全局变量

从作用范围来讲，变量分为局部变量和全局变量。在过程或函数内部进行声明的变量通常被称为局部变量，否则称之为全局变量。他们本质上的不同在于，局部变量仅仅在程序中它们被声明的函数或过程内部才有效，在外部无法调用；全局变量可以被声明该变量的 implementation 部分内所有函数或过程使用。

全局变量一般在 implementation 后面声明，它可以在声明时直接被初始化为一个值。其语法为：

```
var VariableName:type=constantExpression;
```

这里 constantExpression 是任何值为 type 类型的常量表达式。而局部变量在声明时不能进行初始化。但是在同时声明多个全局变量时，也不能对变量进行初始化。

在没有初始化的情况下，所有的全局变量会自动初始化为 0，而局部变量则根据变量类型的不同而不同。

不管是局部变量还是全局变量，当声明后，系统都会给它分配一个内存空间。当变量不再使用时，应当释放所占用的空间。不同的是，局部变量处于函数或过程内部，当函数或过程执行完毕后，程序会自动释放变量占用的空间；但是对于全局变量，它会一直占用系统资源直到程序被关闭。

所以，在程序设计时，应尽量避免使用全局变量。

2.2.2 常量

常量是程序执行过程中不发生变化的值，可以这样理解：常量就是不能修改的变量。例如：

```
const
    Pi =3.1415926;
    selectNo=2003;
    MyName="Smith";
    MaxNum=200;
```

像变量一样，常量也有类型。不同的是，常量的类型就是常量说明中其所代表的值的类型。上面的 4 个常量的类型分别是 real 型、整型、字符串型和整型。常量用“=”表示两边的值是相等的。

使用常量可以带来几个好处：

- 可以更可靠地进行修改。如果程序中多个地方使用到了常量 MaxNum 的话，只需要在常量的定义处将数值改变，就可以将程序中的数值进行调整。
- 增强了程序的可读性。

2.2.3 预定义类型

Object Pascal 有多个预定义的数据类型，其中包含整型、实型、布尔型、字符型、指针型、字符串型等基本数据类型。各种数据类型的定义如下。

- 整型 (Integer)：范围是-32768~32767，占 2 字节的内存；Shortint 从-128~127，1 字节；Longint 从-2147443648~2147483647，4 字节；Byte 从 0~255，1 字节；Word 从

0~65535, 2 字节。它们都是没有小数部分的数字。

- 实型 (Single): 可以包含 7 到 8 位有效小数部分, 占用 4 字节的内存; Double 类可以包含 15 到 16 位有效小数部分, 8 字节; Extended 类型包含 19 到 20 位有效小数部分, 10 字节; Comp 可以包含 19 到 20 位有效小数部分, 8 字节。

- 布尔型 (Boolean): 只包含 true 或 False 两个值, 占用 1 字节内存。

- 字符型 (Char): 一个 ASCII 字符; 字符串类型 String 一串最长可达 255 个 ASCII 字符。

- 指针型 (Pointer): 可以指向任何特定类型。

- 字符串型 (Pchar): 是一个指向以零结尾的字符串的指针。

2.2.4 保留字和指令字

对象 Pascal 语言中定义了 65 个保留字, 它们具有特殊的含义, 不可以用来作为标识符、变量或常量, 读者在帮助文件中查找【Reserved Words】, 就可以知道这些关键字及其用法。

对象 Pascal 语言中还定义了 39 个指令字, 它们也具有特殊的含义。但是当用户重新定义了这些指令字后, 在作用域内它们就失去了原来的意义了。读者在帮助文件中查找【Directives】, 就可以知道这些关键字及它们的用法。

2.3 数据类型

Object Pascal 的最大特点是, 它的数据类型特别严谨, 这表示传递给过程或函数的实参必须和定义过程或函数时的形参的类型相同。不会在 Pascal 中看到一些著名编译器例如 C 编译器所提示的可疑的指针转换等编译警告信息。这是因为 object Pascal 编译器不允许用一种类型的指针去调用形参为另一种类型的函数 (无类型的指针除外)。

Object Pascal 语言提供的数据类型非常丰富。有简单数据类型 (Simple), 字符串数据类型 (String), 结构数据类型 (Struct), 指针数据类型 (Pointer), 函数和过程数据类型 (Procedural), 变体数据类型 (Variant) 等。

2.3.1 简单数据类型 (Simple)

简单数据类型包括有序数据类型 (ordinal) 和实数数据类型 (real)。其中, 有序数据类型又包括整数类型, 字符类型, 布尔类型, 枚举类型和子界类型等。

1. 整数类型

整数类型是最简单、最常用的一类数据, 其包含的数据类型和取值范围如表 2-1 所示。

不同的整数类型, 其占用的内存长度不同, 在定义整数类型时, 应根据数值的大小范围来确定它的类型, 以免造成内存浪费。例如要定义一个数值在 100 以内的整数, 使用 Shortint 类型就足够了。

整数类型的运算使用了 4 种精度的操作数: 8 位、16 位、32 位和 64 位。其云端遵循

下面的规则：对于二元运算符（每个运算符需要两个操作数），在进行操作之前都是将操作

表 2-1 Object Pascal 语言中的整数类型

整 数 类 型	长 度	数 值 范 围
Integer	signed 32 bit	-2147483648 ~ 2147483647
Cardinal	unsigned 32 bit	0 ~ 4294967295
Shortint	signed 8 bit	-128 ~ 127
Smallint	signed 16 bit	-32768 ~ 32767
Longint	signed 32 bit	-2147483648 ~ 2147483647
int64	signed 64 bit	$-2^{63} \sim 2^{63}-1$
Byte	unsigned 8 bit	0 ~ 255
Word	unsigned 16 bit	0 ~ 65535
Longword	unsigned 32 bit	0 ~ 4294967295

数转换成相同的类型。所谓相同的类型是指包含这两个操作数类型所有可能取值的最小预定义类型。例如，Smallint 和 Byte 的共同类型是 Smallint，因为 Smallint 是有符号的 16 位，而 Byte 是无符号的 8 位。如果定义两个变量分别是 Integer、Int64，那么在它们计算时，只有把 Integer 转换成 Int64，这样才能保证计算的精度。

2. 字符类型

Delphi 可以使用三种类型的字符变量：

- **AnsiChar**：标准的 1 字节的 ANSI 字符，能够存储 256 个不同的字符，其对应的整数范围为 0~255。

- **WideChar**：2 字节的 Unicode 字符。Unicode 字符集的前 256 个字符与 ANSI 字符集相同。

- **Char**：相当于 WideChar。

需要注意的是，一个字符在长度上并不表示一个字节，所以不能在应用程序中对字符长度进行硬编码，而应该使用 `Sizeof()` 函数。

3. 布尔类型

布尔类型 (Boolean) 包括四种：Boolean (通用类型)，ByteBool，WordBool 和 LongBool。其中 Boolean 和 ByteBool 为单字节，WordBool 为双字节，LongBool 为四字节。

Object Pascal 预定义了两个常量标识符 false 和 true。可以将 Boolean 类型的数据赋值为 false 和 true，对应的数值为 0 和 1。ByteBool，WordBool，LongBool 类型对应的数据为 0 时，可以认为是 false；不为 0 时，可以认为是 true。

4. 枚举类型

枚举类型 (Enumerated) 是由一组有序的标识符组成的，说明列出了所有这种类型可以包括的值，如下面的例子。

```
type
```

```
  TDays=(Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);
```

可以定义上述枚举类型的变量:

```
Var
    DayofWeek :TDays
```

在枚举型中, 括号里的每一个值都有一个由说明它的位置决定的整形值。例如 Sunday 有整形值 0, Monday 有整形值 1 等。用户可以把 DayofWeek 说明为一个整形变量, 并将一星期的每一天赋一个整型值以达到相同的效果, 但用枚举型会使得程序可读性好, 编写容易。当在枚举型中列出值时, 同时说明了这个值是一个标识符。例如程序中如果已经含有 TDays 类型且说明了 DayofWeek 变量, 则程序中便不能再使用 Monday 变量, 因为它已经被说明为标识符了。

5. 子界类型

子界类型 (Subrange) 是下列这些类型中某范围内的值: 整型、布尔型、字符型或枚举型。在用户想限制一个变量的取值范围时, 子界型是非常有用的。

```
type
    Thurs=0..23
    TValidLetter='A'..'F'
    TDays=(Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);
    TWorkDay=Monday..Friday; {一个 TDays 型的子界}
```

子界型限定了变量的可能取值范围。当范围检查打开时 (在库单元的 implementation 后面有 {\$R*.DFM} 字样表示范围检查打开, 否则可以在【Options】/【Project】/【Compiler Options】对话框中选择【Range Checking】属性来打开范围检查), 如果变量取到子界以外的值, 会出现一个范围检查错误。

6. 浮点数据

Object Pascal 语言中的浮点类型是全体带符号的实数的总称, 它用来表示不同格式的实数数据。浮点数的值域几乎是无限的, 参与的运算几乎都是近似计算。

浮点类型有几种基本类型, 其中 Real 是浮点类型的通用类型, 它等同于 Double, 但是一般情况下用 Double 定义实数数据。浮点数据的类型如表 2-2 所示。

表 2-2 Object Pascal 语言中的实数数据类型

实数类型	范 围	有效位数	字 节 数
Real48	$2.9 \times 10^{-9} \sim 1.7 \times 10^{38}$	11	6
Single	$1.5 \times 10^{-45} \sim 3.4 \times 10^{38}$	7	4
Double	$5 \times 10^{-324} \sim 1.7 \times 10^{308}$	15	8
Extended	$3.6 \times 10^{-4951} \sim 1.1 \times 10^{4932}$	19	10
Comp	$-263+1 \sim 263-1$	19	8
Currency	$-922337203685477.5808 \sim 922337203685477.5807$	19	8

2.3.2 字符串类型 (string)

字符串是代表一组字符的变量类型, 每一种语言都有自己的字符串类型的存储和使用

方法。在 Object Pascal 中，通常用一对单引号来把字符串括起来，例如'A String'。

Pascal 类型有下列几种不同的字符串类型来满足程序的要求：

- **AnsiString**: 这是 Pascal 默认的字符串类型，它由 AnsiChar 字符组成，其长度没有限制，同时与 null 结束的字符串相兼容。
 - **shortstring**: 保留该类型是为了向下兼容旧版本的 Delphi，它的长度限制在 255 个字符内。
 - **Widestring**: 功能上类似于 AnsiString，但它是由 WideChar 字符组成的。
- 默认情况下，如果用如下的代码来定义字符串，编译器认为是 AnsiString 字符串：

```
Var
    str:string;           //编译器认为 Str 的类型是 AnsiString
```

可以用编译开关 \$H 来将 string 类型定义为 ShortString。当 \$H 编译开关的值为负时，string 变量是 ShortString 类型；当 \$H 编译开关的值为正时（默认情况），字符串变量是 AnsiString 类型。如下面的程序：

```
Var
    {$H-}
    E1:string;           //E1 是 Shortstring 类型
    {$H+}
    S2:string;           //S2 是 AnsiString 类型
```

使用 \$H 规则的一个例外是，如果在定义时特地指定了长度（最大在 255 个字符内），那么就总是 ShortString，例如：

```
S:string[63];           //63 个字符的 Shortstring 字符串
```

Delphi 中有丰富的运算符、过程和函数来处理字符串型的变量运算。下面的程序说明了字符串的运算：

1. 使用 “+” 运算符

```
Var
    s1, s2:string;
begin
    s1:='Hello';
    s2:='World';
    s1:=s1+s2;           //'HelloWorld'
end.
```

2. 用 concat() 函数

```
var
    s1, s2:string;
begin
    s1:='Hello';
    s2:='world';
    s1:=Concat(s1,s2);   //'HelloWorld'
```

```
end.
```

2.3.3 结构类型 (Struct)

结构数据类型包括集合、数组、记录、文件、类、类引用、接口等类型。

1. 集合类型 (Set)

集合类型是一群相同类型元素的组合，这些类型必须是有限类型，如整型、布尔型、字符型、枚举型和子界型。在检查一个值是否属于一个特定集合时，集合类型非常有用。

集合类型的定义方法：set of BaseType。例如：

```
type
    TInt = 0..255;           //定义 BaseType 为有序类型
    Tl=set of TInt;         //定义集合类型
    Tdate=set of (Wed, Mon, Thu, sun, Sat);
    TChar=set of ('a', 'b', 'C');
```

Object Pascal 提供了一些用于集合的运算符，用这些运算符可以判断集合与集合之间的关系，对集合增删元素，或对集合进行求交集运算等。

2. 数组类型 (Array)

数组类型是某种数据类型的有序组合，其中每一个元素的值由其相对位置来指定，可以在数组的某个位置上放置数据，并在需要时使用这些数据。下面的类型声明了一个 Double 型的数组变量：

```
Var
    TInt :array[1..10] of Integer;
```

它表示 TInt 指向一个含有 10 个 Integer 型元素的数据串列，代表每一个元素的是 1 到 10 之间的数字，称为索引。Tint 包含 10 个变量，Tint[1]表示第一个变量。

也可以把数组定义成某种类型：

```
type
    Tdou=array[1..10] of Double ;
```

则变量说明改为：

```
Var
    Tdou :Dou ;
```

用户可通过给数组赋值等方法来使用数组。下面语句将 0.0 赋给 Tdou 数组中所有元素：

```
for i :=1 to 10 do
    Tdou[i]:=0.0;
```

数组也可以是多维的，下面的类型定义了一个 20 行、20 列的数组。

```
type
    Ttable = array[1..20, 1..20] of Double;
var
    table1 :Ttable;
```

想将这一表格的所有数据初始化为 0.0，可以使用 for 循环：

```

Var
    Col, Row: Integer;
    ... ..
    for Col:=1 to 25 do
        for Row:=1 to 25 do
            Table1[Col, Row]:=0.0 ;

```

3. 记录类型 (Record)

记录是程序可以成组访问的一群数据的集合。记录类型中下面的例程说明了一个记录类型的用法:

```

type
    TEmployee=record
        Name :string [20];
        YearHired:1990..2000;
        Salary :Double;
        Position :string {20};
    end;

```

记录包含可以保存数据的域, 每一个域有一个数据类型。上文的记录 TEmployee 类型就含有四个域。可以用以下的方式说明记录型的变量:

```

Var
    NewEmployee, promotedEmployee :TEmployee ;

```

用如下的方法可以访问记录的值域:

```
NewEmployee.Salary :=1000;
```

编写如下的语句可以给整个记录赋值:

```

with promotedEmployee do
begin
    Name:='wang Hong';
    YearHired :=1993;
    Salary :=2000.00;
    position:='editor';
end;

```

在程序里可以将记录当成单一实体来操作:

```
promptEmployee:=NewEmployee;
```

4. 指针类型 (Pointer)

指针类型的变量指向内存空间的地址。定义形式如下:

```
type pointerName=^typt;
```

下面的例子对指针的定义和使用进行了说明。

```

program project1;
{$APPTYPE CONSOLE }

```

```

type
    pint = ^integer;           //定义指针类型
var
    a : integer;
    b : integer;
    pt : pint;                 //整形指针
    p : pointer;              //无类型指针
begin
    a := 2;
    b := 3;
    pt := b;                   //整形指针指向整形数据
    writeln('b=', pt^);       //输出' '中的字符串以及后面的变量值
    p := @a;                   //无类型指针指向整形数据
    //b := P^;                 //不可以直接赋值给变量
    b := integer(P^);          //无类型指针指向的内容给整型变量赋值
    writeln('b=', b);
    pt := P;                   //指针间赋值
    writeln('pt^:', pt^);
    readln;
end .

```

运行结果如下:

```

b=3
b=2
pt^:2

```

例子中定义了一个无类型的指针 `p`，它可以指向任何类型的数据，但使用过程中要注意类型转换，不可以直接将地址中的内容直接赋值给其他类型变量。

将 `@` 运算符放在变量前面，将获取变量的地址，并可以把地址赋值为同样数据类型的指针。把 `^` 运算符放在一个数据类型的前面，可以定义该类型的一个指针类型；如果放在一个指针的后面，可以获取该指针指向的地址空间的内容。

5. 变体类型 (Variant)

从 Delphi 2.0 开始引进了一个功能强大的数据类型，称为变体类型 (Variant)，主要是为了支持 OLE 自动化操作。Delphi 3.0 引进了一个新的被称为 OLE Variant 类型，它跟 Variant 基本一致，但是它只能表达与 OLE 自动化操作相兼容的数据类型。

有时候变量的类型在编译期间是不确定的，而 Variant 能够在运行期间动态地改变类型，这就是引入 Variant 类型的目的。

Variant 能支持所有简单的数据类型，例如整型、浮点型、字符串型、布尔型、日期和时间、货币以及 OLE 自动化对象等。注意 Variant 不能表达 Object Pascal 对象。Variant 可以表达不均匀的数组（数组的长度是可变的，它的数据元素能表达前面介绍过的任何一种

类型，也可包括另一个 Variant 数组)。

6. 强制类型转换和类型约定

强制类型转换是一种技术，通过它能使编译器把一种类型的变量当作另一种类型。由于 Pascal 有定义新类型的功能，因此编译器在调用一个函数时对形参和实参类型匹配的检查是严格的。因此为了能通过编译检查，经常需要把一个变量的类型转换为另一个变量的类型。例如，假定要把一个字符类型的值赋给一个 Byte 类型的变量，使用如下的代码，将出现错误。

```
var
  c:char;
  b :byte;
begin
  c := 's';
  b :=c;                      //编译器要提示错误
end.
```

而在下面的代码中，强制类型转换把 c 转换成 byte 类型，事实上强制类型转换是告诉编译器“我知道我正在干什么”，并要把一种类型转换为另一种类型：

```
var
  c :char;
  b :byte;
begin
  c := 's';
  b :=byte (c);              //编译器不报错
end.
```

只有当两个变量的数据长度一样时，才能对变量进行强制类型转换。例如，不能把一个 Double 强制类型转换成 Integer。为了把一个浮点数类型转换为一个整数，要用 Trunc() 或 Round() 函数。为了把整型转换成一个浮点数类型的值，用下列的赋值语句：

```
floatVar:=intVar;
```

2.4 赋值语句

在事件处理过程中，最常用的操作就是把一个新值赋给一个属性或变量。在设计用户界面时，可以使用【Object Inspector】来改变属性，但有时需要在程序设计时改变属性的值，而且有些属性只能在执行时改变。进行这些改变，就必须使用赋值语句。

下面的代码定义了一个 OnClick 事件。当单击按钮后，赋值语句将编辑框对象 Edit1 的 Color 属性设置为 clRed。

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  Edit1.color :=clRed;
```

end;

[例 2-1] 改变编辑框的色彩

- (1) 在 Delphi 7 中新建一个应用程序，将窗体调整为一个较小的尺寸。
- (2) 在窗体中添加一个 Button 对象和一个 Edit 对象。设置 Button 对象上显示的文字为“改变色彩”，如图 2-1 所示
- (3) 双击按钮对象，打开单元文件，修改按钮的“鼠标单击”事件代码如图 2-2 所示。

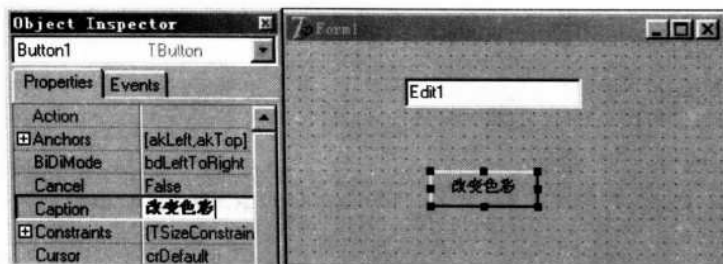


图 2-1 添加对象并修改文字

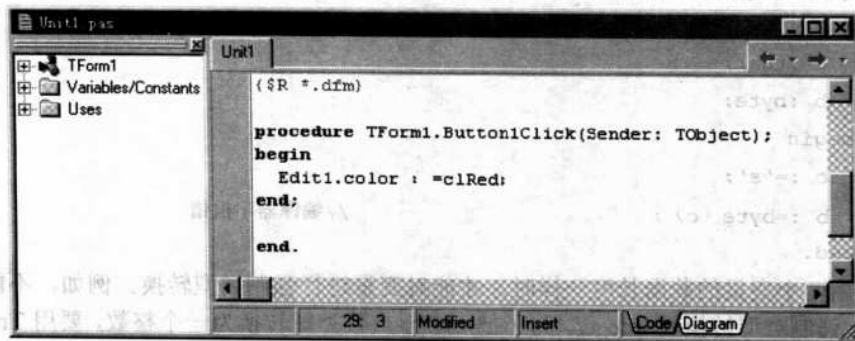


图 2-2 修改按钮的“鼠标单击”事件代码

提示：在程序中输入的符号一定要使用英文状态下的符号。

- (4) 执行程序。当单击按钮后，赋值语句被执行，编辑框变成红色，如图 2-3 所示。

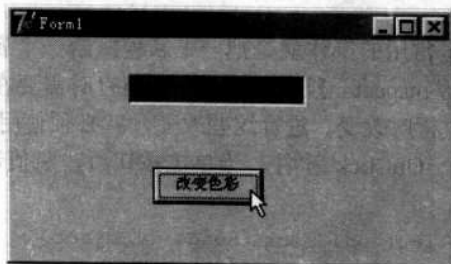


图 2-3 单击按钮，编辑框变成红色

在语句中, 部件的名称在属性前, 中间用“.”表示属性的所属关系。这样就准确地指定了要将 clRed 值赋给哪一部件的哪一属性。赋值号为“:=”, 不论给属性还是给变量赋值, 都是将右边的值赋给左边的属性或变量。

当将一个属性值、变量、常量或文本数据赋给属性或变量时, 所赋值的类型和接受此值的属性或变量的类型应相同或兼容。一个属性或变量的类型定义了此属性或变量的可能值集合, 也定义了程序代码可以执行的运算。在前边的例程中, 编辑框部件的 Color 属性和 clRed 的类型都是 TColor。可以在在线帮助中找到一个属性的类型; 另外一种方法是在【Object Inspector】中选定该属性值段, 并按下 F1 键, 则类型将在属性说明的结尾处列出。例如 Color 属性栏会列出下边的语句:

```
property Color :Tcolor;
```

2.5 过程与函数

在计算机的程序设计中, 对于较简单、规模较小的问题, 可直接编写程序。但对于较复杂的问题, 如果不精心策划, 一上来就动手写程序, 就可能随着语句数的增多和情况趋于复杂化而陷入困境。较好的方法是先考虑一下如何将这一复杂问题划分成若干个目标较为明确而且易于处理的子问题, 然后再就每一个子问题进行分析, 确定是否还需继续划分或者已能借助一些语句来实现。这样通过分而治之, 逐个击破, 再合而为一, 便解决了复杂的程序问题。为了在技术上能够实现上述想法, Pascal 提出了函数和过程的概念。

函数和过程是 Pascal 语言的两种子程序, 它们可以随时被调用。函数在执行时将返回一个值, 而过程执行时不返回任何值。调用函数时得到一个返回值, 而调用过程仅仅为了执行某种操作。调用函数时直接通过函数名称就可以返回一个值; 而调用过程不能通过过程名称带回结果。函数只能出现在表达式中, 不能单独使用; 而过程不能出现在表达式中, 仅能以某个语句出现。

2.5.1 函数的声明和使用

函数声明的语法格式为:

```
Function <函数名> (<形式参数表>):<类型>; //函数头
begin                                     //函数体
    <语句>;
    .....
    <语句>;
end;
```

函数声明分为函数头和函数体两个部分。函数头的 Function 是 Pascal 语言的关键字; 函数名指明了所定义的函数的名称; 形式参数表说明了函数的自变量及其类型, 同一类型的不同参数通过逗号分隔, 不同类型的参数需要使用分号分隔; 形式参数表后面给出了函数的返回类型; 函数头以分号结尾。函数体被包围在关键字 begin 和 end 内, 它给出了函数的实现代码。在函数体中必须有一个赋值语句, 将函数的返回值赋给函数名。

在 C 语言中, 返回值是通过 Return 语句实现的, 而在 Delphi 中, 返回函数值有两种方式:

- 返回值由函数名返回。
- 返回值由 Result 传回。

Result 变量是 Delphi 自动在函数内部隐含的一个局部变量, 这个变量的类型跟函数的返回行相同, 它实际上是函数名的一个别名。

在上述两种返回函数值的方法中, 推荐使用第二种方式, 因为使用 Result 变量来返回结果可以避免无意的递归调用同一个函数:

下面这个函数用来求出两个变量的和:

```
Function Total (x, y:real):real;
Begin
  Total:=x+y ;           //返回值由函数名返回
  //result:=x+y;         //返回值由 Result 传回
End;
```

函数调用的语法格式为:

<函数名> (<实参数>)

实参表中的各个参数必须通过逗号分隔开来, 其顺序必须和函数说明的顺序相对应, 而且个数相同、类型相符。

下面举一个简单的数值求和的例子。通过函数调用, 实现功能要求。

[例 2-2] 数值求和

(1) 在 Delphi 7 中新建一个应用程序, 调整窗体为较小的尺寸。

(2) 在窗体中添加一个 Button 对象和一个 Label 对象。设置 Button 对象上显示的文字为“数值求和”, 如图 2-4 所示。

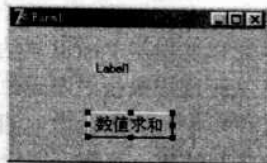


图 2-4 添加 Button 对象和 Label 对象

(3) 双击按钮, 打开代码编辑器, 输入如图 2-5 所示的代码, 定义在单击按钮事件中, 对给定的两个值 x, y 求和。当然, 现在函数 total() 还没有定义。

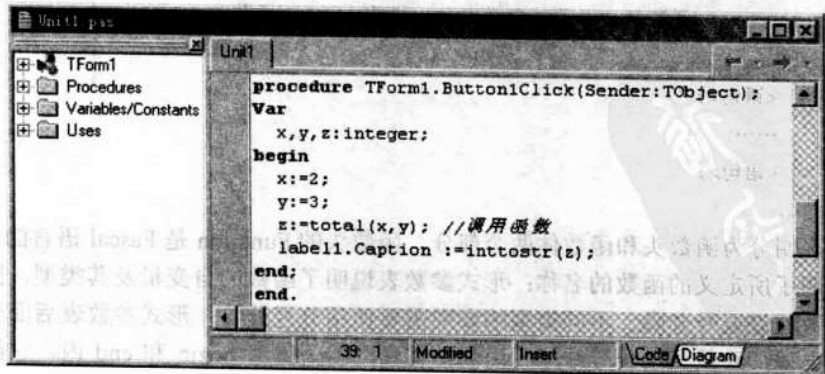


图 2-5 定义在单击按钮事件

(4) 在代码编辑器中添加函数的定义, 如图 2-6 所示。

注意: 函数的定义必须放在按钮单击事件之前, 因为后者调用了函数。

(5) 执行程序。单击按钮, 可见标签对象显示出求和得到的数值。

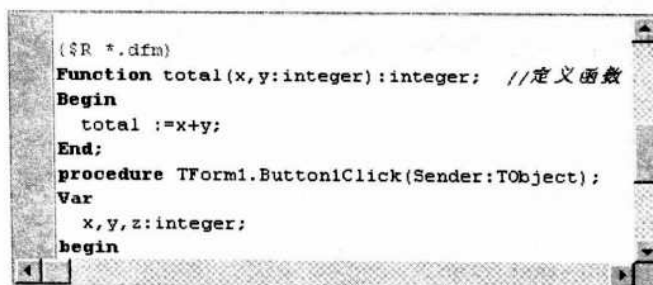


图 2-6 添加函数定义

2.5.2 过程的声明和使用

过程声明的语法格式为:

```
proceduce<过程名>(<形式参数表>)           //过程头
begin                                       //过程体
    <语句>
    .....
    <语句>
end
```

过程的声明和函数类似, 不同之处在于以下几点:

- 过程的关键字为 **Proceduce**。
- 在函数声明中, 必须指明函数的返回类型; 而过程不需要返回任何类型的值, 所以无需声明返回类型。
- 在函数体中, 必须将函数的返回类型赋给函数名, 而在过程中则不存在此语句。

过程调用的语法格式为:

```
<过程名>(<实参表>)
```

实参表中的各个参数必须通过逗号隔开, 其顺序必须和函数说明的顺序相对应, 而且个数相同、类型相符。

2.6 控制语句

Object Pascal 的控制语句包括条件语句、判断语句、循环语句等。

2.6.1 判断语句

判断语句一般有 **if** 和 **case** 两种, 其中前者使用的比较多。

1. IF 语句

if 计算一个表达式, 并根据计算结果决定程序流程。if 保留字后跟随一个生成 Boolean 值 True 或 False 的表达式。一般用 “=” 作为关系运算符, 比较后产生一个布尔型值。并根据条件是否成立来决定下面的执行。当表达式为 True 时, 执行 then 后的语句。否则执行 else 后的语句, if 语句也可以不含 else 部分; 表达式为 False 时自动跳到下一行程序。

下面是一个使用 IF 语句的程序结构示例。

```
if i<>0 then
begin
    result:=i/j;
    count:=count+1;
end
else if count=last then
    done:=true
else
    exit;
```

if 语句可以嵌套, 当使用复合语句表达时, 复合语句前后需加上 begin...end。else 保留字前不能加 “:”。当有多个 if 语句时, 编译器会将 else 语句视为与其最靠近的 if 语句相对应。必要时, 须使用 begin...end 保留字来强迫 else 部分属于某一级的 if 语句。

下面是嵌套型的 if 语句的示例。

```
if J<>0 then
begin
    if J>0 then
    begin
        Result:=I/J-1;
    end
    else
        Result:=I/J+1;
    count:=count +1;
end
else if Count =Last then
    Done :=True
else
    Exit;
```

2. CASE 语句

case 语句适用于被判断的变量或属性是整型、字符型、枚举型或子界型时 (Longint 除外)。用 case 语句进行逻辑跳转比编写复杂的 if 语句容易阅读, 而且程序代码速度较快。

case 语句的程序原型如下:

```

case selectorExpression of
    caseList1:statement1;
    ...
    caseListn:statementn;
else
    statements;
end

```

2.6.2 循环语句

Object Pascal 的循环语句有 3 种: repeat、while 和 for 语句。

1. repeat 语句

repeat 语句会重复执行一行或一段语句直到某一状态为真。语句以 repeat 开始, 以 until 结束, 其后跟随被判断的布尔表达式。其结构原型如下:

```

repeat
    statements                //执行语句
until condition              //执行循环的条件

```

下面的一段代码用于输出整数 1~10 的数字。

```

i:=0;
repeat
    i :=i+1;
    Writeln(i);
until i=10;

```

布尔表达式 $i=10$ (注意, 与其他语言不同的是, “=” 是关系运算符, 而不能进行赋值操作) 直到 repeat...until 程序段的结尾才会被计算, 这意味着 repeat 语句至少会被执行一次。

2. while 语句

while 语句和 repeat 语句的不同之处是, 它的布尔表达式在循环的开头进行判断。while 保留字后面必须跟一个布尔表达式。如果该表达式的结果为真, 循环被执行; 否则会退出循环, 执行 while 语句后面的程序。其结构原型如下:

```

while condition do          //condition 为循环条件
begin
    statements              // 执行语句
end;

```

下面的例程达到和上面的 repeat 例程同样的效果。

```

i:=0;
while i<10 do
begin
    i :=i+1;

```

```
writeln(i);
end;
```

3. for 语句

for 语句的程序代码会执行一定的次数。它需要一个循环变量来控制循环次数。该变量可以是整型、布尔型、字符型、枚举型或子界型。其原型如下：

```
for condition do           //执行条件
begin
    statements              //执行语句
end;
```

下面的程序段会显示 1~5 的数字，i 为控制变量。

```
var
i:integer;
for i:=1 to 5 do
    writeln(i);
```

以上介绍的三种循环语句分别适用于不同的应用场合。如果知道循环要执行的次数，可以使用 for 循环语句，for 循环执行速度快，效率较高；如果不知道要执行的次数，但至少会执行一次的话，选用 repeat ... until 语句比较合适；当认为程序可能一次都不执行的话，最好选用 while...do 语句。

2.6.3 其他控制语句和过程

除了前面这些条件和循环语句外，还有一些用于控制程序执行的语句。

1. break 过程

在 while、for 或 repeat 循环中调用 break()，使得程序的执行流程立即跳出循环；在循环中当某种条件满足时需要立即跳出循环，这时调用 break()。

下面的代码演示了在 30 次循环后跳出循环结构。

```
var
i:integer;
begin
    for i:=1 to 100 do
    begin
        if i=30 then break;
    end;
end;
```

2. continue 过程

如果想跳过循环中部分代码重新开始下一次循环，就调用 continue() 过程。注意下面的例子在执行第二次循环时，continue() 后的代码不执行。

```
var
i:integer;
```



```

begin
  for i:=1 to 3 do
  begin
    writeln(i, 'before continue');
    if i=2 then continue;
    writeln(i, 'after continue');
  end;
end;

```

3. with 语句

使用记录类型的变量时，可以通过 with 语句指定一些语句都是针对某一个变量来说的，这样可以简化代码的输入量。with 语句的形式如下：

```
with obj do statement
```

下面的代码为一个结构型变量赋值。

```

type
  TEmployee = record
    Name :string[20];
    YearHired:1990..2000;
    Salary :Double;
    Position:string[20];
  end;

var
  employee:TEmployee
  ... ..
with employee do
begin
  Name := 'wang';
  YearHired:=2001;
  salary:=10000;
  position:='Technic Department';

```

2.7 实例 —— 随机加减法测试

下面我们用一个简单的实例来说明如何利用基本的 Pascal 语言来设计 Delphi 程序。

[例 2-3] 随机加减法

实例的效果是要随机产生一个加减法的算式；输入答案，按下 **Enter** 键后，能够根据答案是否正确自动给出相应的提示对话框。如图 2-7 所示。

(1) 创建一个新的应用程序；调整窗体为较小的尺寸。

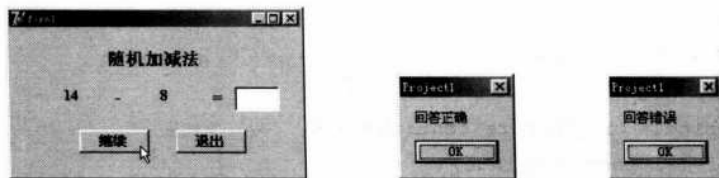


图 2-7 随机加减法

(2) 从【Standard】选项卡中拖动【Label】组件、【Button】组件和【Edit】组件到窗体上，创建如图 2-8 所示的内容对象。

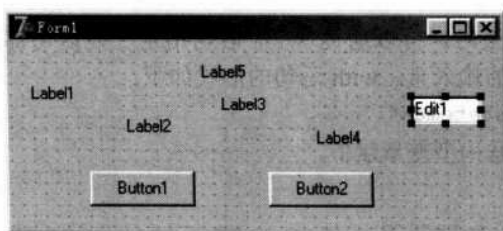


图 2-8 拖动组件到窗体上

(3) 选择【View】/【Alignment Palette】菜单命令，打开【Align】(对齐)面板，利用它调整各个组件对象的位置，如图 2-9 所示。使它们符合程序画面效果要求。

(4) 设置各个组件对象显示的文字、字体和大小等属性，如图 2-10 所示。其中“Label4”组件对象的【Caption】属性就是“=”。

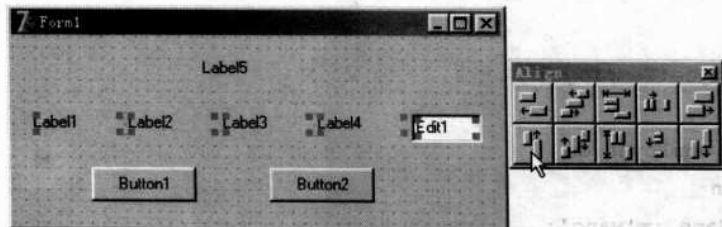


图 2-9 调整各个组件对象的位置

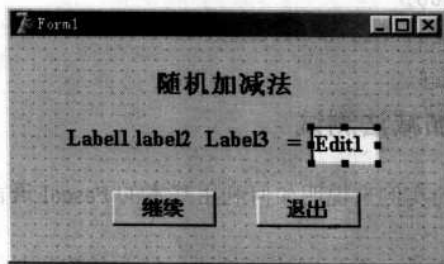


图 2-10 设置各个组件对象的显示属性

(5) 在窗体上双击，切换到代码编辑器，光标会自动定位于“TForm1.FormCreate();”

事件；该事件在窗体产生时发生。添加如下所示程序代码，其中函数 `inttostr()` 是将一个整型数转换为字符串值。

```
procedure TForm1.FormCreate(Sender: TObject); // “窗体创建” 事件
var
    m,n,k:integer; //定义整型变量
begin
    randomize; //初始化随机数发生器
    m:=random(50); //为变量 m 赋一个 50 以内的随机值
    n:=random(50); //为变量 n 赋一个 50 以内的随机值
    k:=random(2); //为变量 k 赋一个 2 以内的随机值
    label1.Caption :=inttostr(m); //在 Label1 中显示变量 m 的值
    label3.Caption :=inttostr(n); //在 Label3 中显示变量 n 的值
    if k=1 then //如果 k 的值等于 1
        label2.Caption := '+' //在 Label1 中显示 “+”
    else //如果 k 的值不等于 1
        label2.Caption := '-'; //在 Label1 中显示 “-”
    end;
end;
```

(6) 在窗体中选择 “Edit1” 组件对象，从【Object Inspector】中选择【Events】选项卡中的【OnKeyPress】(键盘按键按下) 事件，鼠标双击，切换到代码编辑器，输入如下所示程序代码。

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
    answer:integer; //定义变量 answer
begin
    if key=chr(VK_RETURN) then //如果按下的键是 Enter 键
    begin
        if label2.Caption='+' then //如果是加法，就对两个对象的值求和
            answer:=strtoint(label1.Caption)+strtoint(label3.Caption)
        else //如果不是加法，就对两个对象的值求差
            answer:=strtoint(label1.Caption)-strtoint(label3.Caption);
        if answer=strtoint(edit1.Text) then //如果答案与 Edit1 对象中的内容相同
        begin
            showmessage('回答正确'); //显示信息框 “回答正确”
            button1.SetFocus ; //将焦点定位于按钮 Button1 上
        end
        else //如果答案与 Edit1 对象中的内容不相同
        begin
            showmessage('回答错误'); //显示信息框 “回答错误”
            edit1.SetFocus ; //将焦点定位于按钮 Edit 上
        end
    end
end;
```

```

end;
edit1.SelectAll ;           //选择 Edit1 中的内容
end;
end;

```

(7) 在窗体中选择“Button2”组件对象，双击鼠标，切换到代码编辑器，再通过鼠标输入结束程序运行的命令“close”，如图 2-11 所示。

(8) 在窗体中选择“Button1”组件对象，从【Object Inspector】中选择【Events】选项卡中的【OnClick】（鼠标单击）事件；然后利用下拉按钮选择“FormCreate”事件，如图 2-12 所示。这样，按钮的【OnClick】（鼠标单击）事件就能够共享窗体【OnCreate】事件的代码，从而具有相同的功能。

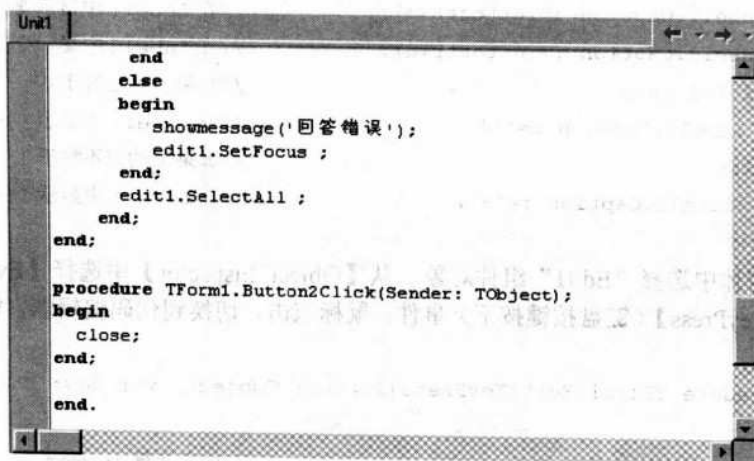


图 2-11 “Button2” 组件对象单击事件中的语句

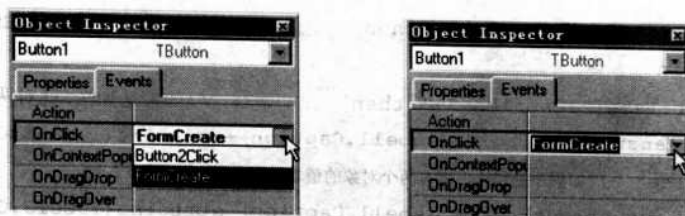



图 2-12 为“Button1” 组件对象选择共享代码

(9) 单击  按钮，经过编译后，程序就能够随机产生加减法数学题了。

(10) 完整的程序代码如下所示，请大家参考。

```

unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,

```

```

Forms,
  Dialogs, StdCtrls, qt;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Label5: TLabel;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  m, n, k: integer;
begin
  randomize;
  m:=random(50);
  n:=random(50);
  k:=random(2);
  label1.Caption :=inttostr(m);


```

```
label3.Caption :=inttostr(n);
if k=1 then
    label2.Caption := '+'
else
    label2.Caption := '-';
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
    answer:integer;
begin
    if key=chr(VK_RETURN) then
    begin
        if label2.Caption='+' then
            answer:=strtoint(label1.Caption)+strtoint(label3.Caption)
        else
            answer:=strtoint(label1.Caption)-strtoint(label3.Caption);
        if answer=strtoint(edit1.Text) then
        begin
            showmessage('回答正确');
            button1.SetFocus ;
        end
        else
        begin
            showmessage('回答错误');
            edit1.SetFocus ;
        end;
        edit1.SelectAll ;
    end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    close;
end;

end.
```



2.8 小结

本章主要介绍了 Object Pascal 语言的基本语法和语义，其中包括变量、运算符、数据类型、函数、过程等，这些知识是进行 Delphi 程序设计的基础。最后通过一个实例程序的练习，使大家了解了在 Delphi 中如何定义变量、如何使用控制语句和函数，以及如何的代码共享等知识。虽然有些方法和属性大家目前可能还不太了解，但是建立一个感性的认识，对于更好地理解本章内容，是很有好处的。

资源分享
PDG

第3章 面向对象编程

从20世纪90年代至今，面向对象编程（Object Oriented Programming, OOP）成为编程语言的最主要特征。OOP重视软件的代码重用，能够更好地模拟现实世界环境，被公认为是自上而下编程的理想选择。Borland公司在Pascal中添加了面向对象的特性，构成了Delphi的程序设计语言——面向对象的Pascal语言（Object Pascal）。

虽然Delphi是完全基于OOP的，但是一个不了解OOP的程序员也能够使用Delphi编写程序，因为Delphi会自动完成许多工作。但是，了解语言及其实现细节对于掌握Delphi是很有帮助的。

3.1 OOP 基本概念

面向对象编程是当前流行的编程技术，它解决了传统面向过程编程带来的许多问题。面向过程编程所有的功能都包含在几个代码模块中（常常是一个代码模块）。而使用OOP技术，常常要使用许多代码模块，每个模块都提供特定的功能，每个模块都是独立的。这种模块化编程方法提供了非常大的多样性，大大增加了代码的重用机会。

例如，我们可以将计算机程序考虑为一个房子。如果使用面向过程编程，这栋房子就是一个单元。如果你想为房子换个门窗，就必须替换整个单元，重新建造一栋房屋。如果使用OOP技术，就可以在建造时将房屋设计成一个个独立的模块（对象）。如果需要换玻璃，只需要选择门窗，调换玻璃就可以；如果需要改变风格，只需要重新调整房屋的颜色和布局就可以。这就是OOP编程的优势。

可以说，对象是OOP技术的关键，那么，什么是对象呢？

对象是OOP应用程序的一个重要组成部件。这个组成部件封装了部分应用程序，这部分应用程序可以是几个过程、数据或更抽象的实体。

简单来说，对象非常类似于前面介绍的记录类型的变量，但与记录类型不同的是，对象不但能够包含数据成员（字段），还能够包含函数和过程（方法），这些数据和方法被统称为类的成员。

Delphi中的对象是从类型中创建的，就像前面的变量一样。对象的类型在OOP中有一个特殊的名称：类。对象与类是OOP中极其重要的两个概念，要注意，类和对象是两个完全不同的东西，它们之间的关系就像类型与变量的关系。

对象中的几个基本概念：

- 字段：字段就是对象中的数据成员，它可以是一组任何Delphi支持的类型变量的集合，比如integer、string、数组等。字段用来标志对象的状态，因为同一个类的不同对象

可能在属性和字段中存储了不同的值。在这一点上它同记录类型的变量是一致的。

- 属性：属性和字段一样，都可以访问对象中包含的数据。字段和属性都可以输入，所以可以把信息存储在字段和属性中。同时，属性与字段又是有差异的，属性不能直接访问数据。有些属性是只读的，只能查看它们的值，而不能改变它们。

- 方法：方法是用于表示对象中的过程或函数，它用于访问对象的功能。与字段和属性一样，方法可以是共有的或私有的，从而根据需要限制外部的访问。

- 对象的生命周期：每个对象都有一个明确定义的生命周期，即从使用类定义开始一直到删除它为止。

实际上，Delphi 中几乎所有东西都是对象，尽管有时它们处在 Delphi 完美的封装下而无法表现出来。对象无处不在。

3.2 类的基本概念

类是一种用户定义的数据类型，它有自己的说明（表达）和操作（行为），类中含有内部数据和过程，或函数形式的对象方法，通常用来描述一些非常相似的对象所具有的共同特征和行为。

对象是类的实例，换句话说，是由类定义的数据类型的变量。对象是实体，当程序运行时，对象为它们的内部表达占用一些内存。对象与类的关系就像变量与类型的关系。

3.2.1 类的声明

类由封装在一起的数据和方法构成。封装是指对类中数据的访问会受到一定限制，要通过一定的方法才能访问数据。其优点是可以减少因直接访问数据而造成的错误。

从外部来看，类就像一个部分可见的黑匣子。可见部分称为接口，通过这个接口可以访问类中不可见的内部数据部分。Delphi 提供了一个关键字 `Class`，可以利用它定义一个类的对象。

其语法格式如下：

```
Type
Newclass = class
...
End;
```

其中，`Newclass` 是类的名字。

但是这个类除了说明一个类名外没有其他任何内容，真正有用的类还必须包括数据成员和方法。下面我们就在类中定义一些数据成员：

```
Type
Tmember=class
Name: String;
Age :Integer;
Male:Boolean;
End;
```

提示：Borland 公司的类库中有一个规定，使用 `T` 来作为每个类的前缀。这仅仅是一个约定，就像微软的 `WC` 类库中用 `C` 字母作为类的前缀一样，所有的类使用同一前缀有助于使代码易于理解。

在上面类 Tmember 中, 定义了 3 个属性字段, 这 3 个字段的类型各不相同。如果只看这 3 个字段的话, 类与记录类型有些相似 (当然记录使用的关键字是 Record)。例如, 可以说明对象, 然后用标准表达式来访问它的 3 个元素:

```
var
  Aman:Tmember;
begin
  Aman.Name:='宋一兵';
  Aman.Age:=28;
  Aman.Male:=true;
end;
```

但是, 类的特殊之处在于可以将方法与数据封装在一起。下面再把一些对象方法 (函数或过程) 加入到类的操作中:

```
Type
Tmember=class
Name: String;
Age :Integer;
Male:Boolean;
Procedure SetID(vname:String;vage:Integer;vmale:Boolean); //过程成员
Function IsAdult:Boolean; //函数成员
End;
```

在类中定义了方法, 就要加以实现。类的方法的说明和定义与普通的函数和过程相似, 唯一的区别是要在函数名和过程名的前面加类名和句点, 就像 C++ 成员函数前加双冒号一样。例如, 下面的代码定义了类 Tmember 中的过程成员和函数成员。

```
Procedure SetID(vname:String;vage:Integer;vmale:Boolean);
Begin
Name:=vname;
Age:=vage;
Male:=vmale;
End;
Function IsAdult:Boolean;
Begin
if (Age>=18) then
  IsAdult:=true
else IsAdult =false;
End
```

3.2.2 类的对象

仅仅依靠上述代码并不能进行工作, 如果应用到程序中, 还要创建类的对象。对象是类的实例, 是计算机系统内存中动态分配的一块内存空间, 其结构由类定义。在本书中, 对象可理解为可视化组件, 如按钮、标签和表等。对象使用构造函数创建, 使用析构函数删除。类的每一个对象都有独立的字段拷贝, 但是所有的对象都共享相同的方法。

1. 构造函数

在大多数 OOP 语言中, 通过声明类的变量来创建类的实例, 即实际的类的对象。在 Delphi 中, 类的对象并不真正保存类的数据, 它只是一个引用, 指向类数据在内存中的实际位置。

定义类对象的方法我们在前面已经应用过了，即：

```
Aman:Tmember;
```

但是这样的定义并没有真正创建类的对象，而只是它的一个指针位置。对于用户自定义的类对象，必须手工创建，而控件的实例则可以由 Delphi 自动建立。

要创建类的对象，可以调用它的 Create 方法为对象分配内存和进行数据初始化。要注意的是，Create 作用于类而不是类对象，如下面的代码：

```
Var
  Aman:Tmember;
Begin
  Aman.Create;
...
End;
```

可以使用 Create 方法为类分配内存。然而在使用对象时，常常需要对对象作初始化，可以在每次调用 Create 后紧接着进行，而更好的方法是定义自己的构造函数。可以用关键字 constructor 为类声明构造函数。

```
Type
Tmember=class
Name: String;
Age :Integer;
Male:Boolean;
Constructor Create(); //构造函数
Procedure SetID(vname:String;vage:Integer;vmale:Boolean); //过程成员
Function IsAdult:Boolean; //函数成员
End;
```

类构造函数的实现：

```
Constructor Create();
Begin
  Name:= 'myself';
  Age:=20;
  Male:=false;
End;
```

这样，就可以使用下面的方法为对象 Aman 分配内存并初始化：

```
Aman:=Tmember.Create();
```

构造函数是一种特殊的过程，它一般是由类而不是类的对象调用。当一个类调用它时，Delphi 会自动为对象分配内存。另外，也可以使类对象调用构造函数，但这时不进行任何内存分配工作。因此，一定不要让没有初始化的类对象使用析构函数，只有当类使用构造函数时才真正进行内存分配。

但是上面这段程序实际上有一个漏洞，因为我们在程序运行结束后并没有释放刚才创建的对象。要释放对象，就需要使用析构函数。

2. 析构函数

一旦建立了对象，就可以正常使用它们了，对类对象的使用与一般变量没有什么区别。然而，对于对象，还需要在使用后释放它们。

这可以通过 Free 方法来实现。Free 是 TObject 的另一个对象方法。同样的，与 constructor 对应，还可以用关键字 destructor 为类声明析构函数，它用于在类被消除前完成类占用的

资源释放工作。应用程序应当调用 Free 方法，而不是直接调用析构函数。只有在对象存在时，或者说对象不是 nil 时才可以调用析构函数。而另一方面，Free 不会自动将对象设置为 nil，这是用户自己的工作。

下面是一个创建对象、使用对象、处理对象的代码和释放对象的综合代码示例。

```
Var
Aman:Tmember;
begin
Amain.Create;                //创建对象
Aman.setID('宋一兵',28,true); //设置对象属性
if Aman.IsAdult then         //调用函数成员
    ShowMessage ('An adult.');
```

```
Adult.Free;
end;
```

大家可能不明白从哪里跑出来了 Create 和 Free 方法，其实这两个方法是类 Tobject 中的方法，因为 Delphi 中定义每个类都是 Tobject 的派生类（不一定是直接派生类），所以每一个类都继承了这两个方法。

其实我们前面定义的类就是 Tobject 类的直接派生类，因为：

```
type
Tmember=Class;
```

等价于

```
type
Tmember=Class(Tobject); //关键字后面用括号表示的是类 Tmember
                          的父类
```

与其他构造类型不同，类只能在程序或者单元文件最外层的 Type 语句声明部分出现，所以在变量声明部分或者过程、函数中不能声明类。

3.3 类的封装

一个类中会有多个数据成员方法，而对于良好的面向对象程序设计，类的数据应当被封装，只应在类中使用它。当访问类的内部数据时应通过类定义的接口进行。使用对象方法操作类数据有利于减少产生错误的机会，也有利于程序的修改。

为了便于理解，可以把类想象成一个封闭的小盒子，它上面只有一小部分可见，称为类接口，它允许类或其他程序部分访问和使用这些区域，而类的大部分内容则是不可见的。用户很少知道类有哪些隐含数据，也不能直接访问它们。这种 OOP（面向对象程序设计）的设计思想对于程序的安全和设计都是很有利的。

Object Pascal 使用了五个存取控制符：public、private、protected、published 和 automated。下面分别介绍各说明符的使用范围。

3.3.1 Public 类型

关键字 public 指出字段和方法是公有的，也就是说，它可能被类外部的程序访问。例如我们定义了一个包含公有字段和方法的类 TClerk。

```
type
```

```

Tclerk=Class
public
Name :String;
Constructor Create;
procedure SetName(One:String);
function GetName:String;
end ;

```

在这个类中，定义了公有字段 Name、方法 SetName 和函数 GetName。注意，关键字 public 之后的声明都是公有类型，直到遇到另一种存取控制符为止。另外，我们在这里省略了构造函数、过程 SetName 和函数 GetName 的具体实现。

对于公有类型的字段和方法，我们可以直接在程序中对它们进行访问。下面的代码说明了对公有类型的调用方法。

```

Var
NewClerk : Tclerk;
temp : String;
begin
NewClerk:=Tclerk.Create ;           //创建对象
NewClerk.SetName('Peter');         //调用公有方法
temp :=NewClerk.Name;               //直接获得类的属性值
end;

```

3.3.2 private 类型

关键字 private 指出字段和方法是私有的，私有的成员不能被类所在的单元以外的程序访问，也就是说，只有在包含这个类的单元中，才可以对定义为私有的字段和方法进行访问。例如我们在一个单元中定义了类 Tclerk:

```

type
Tclerk=Class
private
Name :String;
procedure SetName(One:String);
public
Constructor Create;
function GetName:String;
end ;

```

在类 Tclerk 中定义了私有字段 Name 和方法 SetName，公有构造函数 Create 和函数 GetName。

在同一个单元中，也就是说，在定义类的单元中，按照 3.2.1 节中的代码对私有类型的字段和方法进行调用是不会出错误的，但是在不同的单元中就不能使用该代码对类进行调用了。下面的代码说明了在不同单元中对类的私有类型的调用方法。

```

Var
NewClerk : Tclerk;
temp : String;
begin
NewClerk:=Tclerk.Create ;           //调用类的构造函数创建对象
// NewClerk.SetName;                //直接调用私有方法会出现错误
//temp:=NewClerk.Name;              //直接获得类的属性值会出错

```

```
temp:=NewClerk.GetName           //应通过公有方法获得属性值
end;
```

3.3.3 protected 类型

关键字 **protected** 定义保护字段和方法。保护成员可以被该类的所有派生类访问，并且成为了派生类的私有成员。注意，使用派生类的用户无法访问保护成员。

3.3.4 published 类型

关键字 **published** 声明的成员是发行类型的成员。从成员的可见性来说，发行成员是最高的。公有成员在运行期间是可以访问的，而发行成员在设计期间和运行期间都可以访问。

published 类型一般用在组件类的声明中。在这些类中，**published** 通常用来定义属性（不是方法和字段，而是组件类特有的属性）。这样，当组件被放到窗体上时，这些属性才能在对象编辑器中看到。

3.3.5 automated 类型

关键字 **automated** 与 **public** 基本相同。唯一的区别是声明为 **automated** 的属性和方法会生成自动类型信息。自动类型信息使得创建 OLE 自动化服务器成为可能。

3.3.6 类的封装应用练习

一般在程序中常用的类型就是 **Public** 和 **Private** 两种类型。下面我们用一个简单的程序实例来说明这两种类型的使用。

【例 3-1】类的应用练习

程序效果：设计两个窗体，在 Form1 对应的单元文件中定义一个类，该类具有 **Public** 和 **Private** 两种类型的方法，在两个窗体中分别调用该类的方法。程序基本画面如图 3-1 所示。

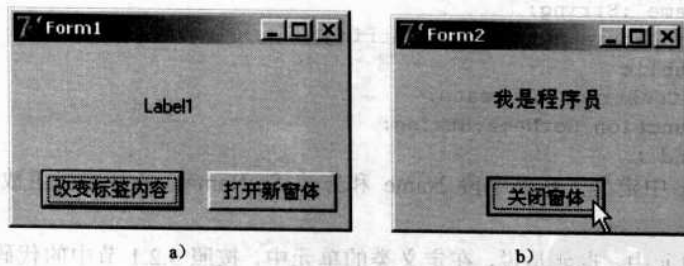




图 3-1 类的应用练习

(1) 在 Delphi7 中创建一个新的应用程序。

(2) 从【Standard】选项卡中拖动标签  和按钮  组件到窗体中，修改按钮上的文字，创建如图 3-1a 所示的窗体界面。

(3) 切换到代码编辑器“Unit1.pas”窗口，在【Interface】部分输入如图 3-2 所示程序代码，定义一个类 Tclerk，该类包含有私有的属性 Name 和过程 SetName，公有的构造函数 create 和函数 GetName。



图 3-2 定义一个类 Tclerk

(4) 在【implementation】部分实现 Tclerk 类的方法。输入如图 3-3 所示程序代码，创建类的构造函数 create、私有过程 SetName 和公有函数 GetName。

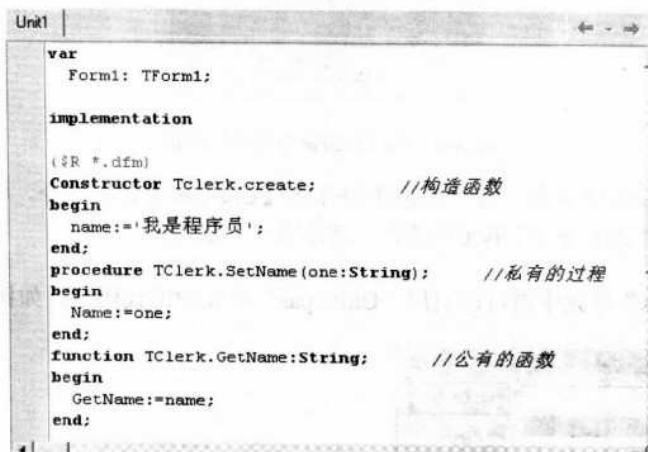


图 3-3 在【implementation】部分实现 Tclerk 类的方法

(5) 在窗体上双击第一个按钮，会自动切换到代码编辑器窗口，并定位于过程 Button1Click 中；输入如图 3-4 所示代码，创建类的对象并调用其方法。

代码说明：

- 定义一个 Tclerk 类型的对象 NewClerk 和一个 String 类型的变量 temp;
- 创建对象;
- 调用私有方法 SetName 设置 Name 属性;
- 直接获得类的属性值，并保存于变量 temp 中;
- 在 Label1 组件对象上显示变量的内容。

```

end;

procedure TForm1.Button1Click(Sender: TObject);
Var
  NewClerk : Tclerk;
  temp : String;
begin
  NewClerk:=Tclerk.Create ;           //创建对象
  NewClerk.SetName('程序员Peter');   //调用私有方法
  temp :=NewClerk.Name;               //直接获得类的属性值
  label1.Caption:=temp;
end;

```

图 3-4 创建类的对象并调用其方法

(6) 选择【File】/【New】/【Form】菜单命令，创建一个新的窗体；在其中添加一个【Label】对象和一个【Button】对象，并定义对象显示的文字如图 3-5 所示。



图 3-5 创建新的窗体并添加对象

(7) 选择【Form1】窗体，然后再选择【File】/【Use Unit】菜单命令，从中选择“Unit2”单元。这样，在“Unit1.pas”单元中就会出现一条语句：

```
uses Unit2;
```

在“Unit1.pas”单元中就可以引用“Unit2.pas”单元中的代码了。如图 3-6 所示。

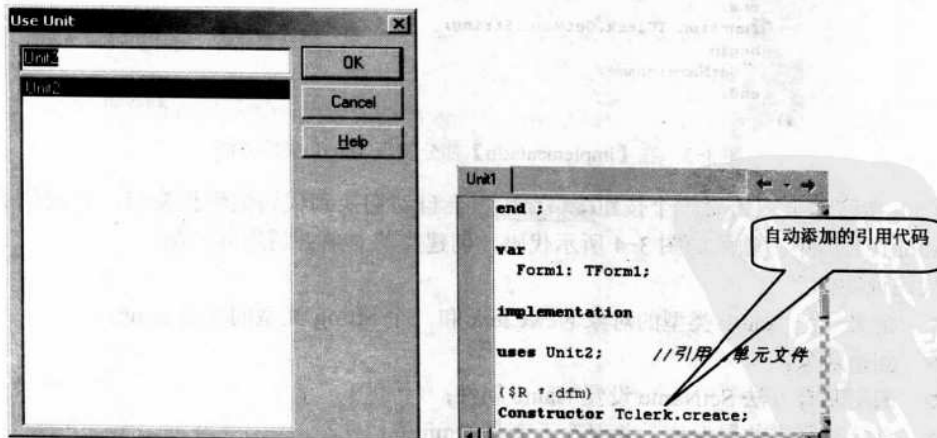


图 3-6 在“Unit1.pas”单元中引用“Unit2.pas”

(8) 同样, 在“Unit2.pas”单元中引用“Unit1.pas”单元。

(9) 在【Form2】的窗体上双击鼠标, 会切换到代码编辑器【Unit2】窗口, 并自动定位于 Form2.FormCreate 事件中。输入如图 3-7 所示代码, 创建 Tclerk 类型的对象, 并获取对象属性。

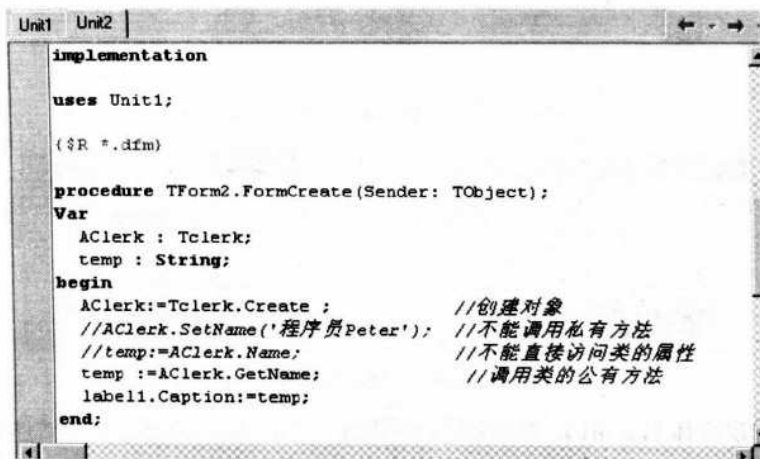


图 3-7 在“Unit2”中创建 Tclerk 类型的对象并获取对象属性

提示: 从程序代码中可以看到, 这段代码与 Form1 的 Button1Click 事件中的代码几乎完全相同, 但是在这个单元中对类的私有方法和类的属性的直接访问却是不允许的。

(10) 选择 Form1 窗体, 双击第二个按钮: **打开新窗体**, 在其 Button2Click 事件中输入如图 3-8 所示代码, 以显示 Form2 窗体。

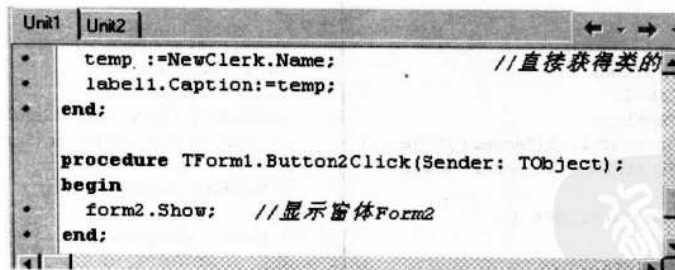


图 3-8 使用: **打开新窗体** 按钮显示 Form2 窗体

(11) 选择 Form2 窗体, 双击按钮: **关闭窗体**, 在其 Button1Click 事件中输入如图 3-9 所示代码, 以关闭当前窗体 (Form2)。

(12) 现在程序代码已经编写完成。单击 按钮, 对程序进行编译和运行。如图 3-10 所示。

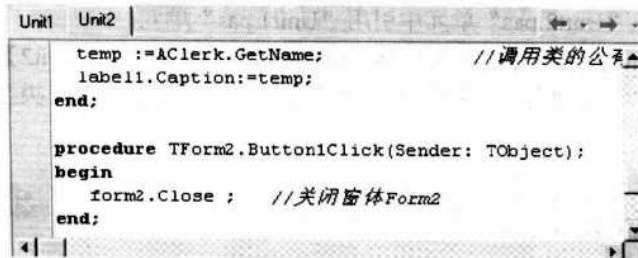


图 3-9 使用“关闭窗体”按钮关闭 Form2 窗体

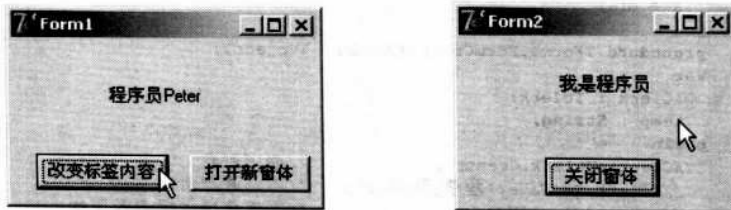


图 3-10 程序运行情况

首先会出现窗体【Form1】，单击“改变标签内容”按钮，则标签的内容改变；单击“打开新窗体”按钮，则打开窗体【Form2】，其中标签显示的内容为类的对象创建时的初始内容；单击“关闭窗体”按钮，则关闭窗体【Form2】。

(13) 程序完成，完整的程序代码如下：

<pre> unit Unit1; interface uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls; type TForm1 = class(TForm) Button1: TButton; Label1: TLabel; Button2: TButton; procedure Button1Click(Sender: TObject); procedure Button2Click(Sender: TObject); private { Private declarations } public { Public declarations } end; type TClerk=Class private Name :String; procedure SetName(One:String); public Constructor create; function GetName:String; </pre>	<pre> unit Unit2; interface uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls; type TForm2 = class(TForm) Button1: TButton; Label1: TLabel; procedure FormCreate(Sender: TObject); procedure Button1Click(Sender: TObject); private { Private declarations } public { Public declarations } end; var Form2: TForm2; implementation uses Unit1; {\$R *.dfm} </pre>
---	--

(续)

<pre> end ; var Form1: TForm1; implementation uses Unit2; //引用的单元文件 {\$R *.dfm} Constructor Tclerk.create; begin name:='我是程序员'; end; procedure Tclerk.SetName(one:String); begin Name:=one; end; function Tclerk.GetName:String; begin GetName:=name; end; procedure TForm1.Button1Click(Sender: TObject); Var NewClerk : Tclerk; temp : String; begin NewClerk:=Tclerk.Create ; //创建对象 NewClerk.SetName('程序员 Peter'); //调用私有方法 temp :=NewClerk.Name; //直接获得类的属性值 labell.Caption:=temp; end; procedure TForm1.Button2Click(Sender: TObject); begin form2.Show; //显示窗体 Form2 end; end. </pre>	<pre> procedure TForm2.FormCreate(Sender: TObject); Var AClerk : Tclerk; temp : String; begin AClerk:=Tclerk.Create ; //创建对象 //AClerk.SetName('程序员 Peter'); //不能调用私有方法 //temp:=AClerk.Name; //不能直接访问类的属性 temp :=AClerk.GetName; //调用类的公有方法 labell.Caption:=temp; end; procedure TForm2.Button1Click(Sender: TObject); begin form2.Close ; //关闭窗体 Form2 end; end. </pre>
--	---

这个实例的目的是为了说明类的封装类型对于其使用的影响。正如我们在前面指出的那样。

在同一个单元中，也就是说，在定义类的单元中，对私有类型的字段和方法进行访问和调用是不受限制的；但是在不同的单元中就不能使用对类的私有类型进行访问了。

3.4 类的继承性

在编程过程中，常常需要创建与现有的类稍有不同版本。这一种方法是对现有的类进行改动，但如果类已经被其他单元使用过，这样就会给其他单元造成问题。还有一种方法是对原型定义进行复制，将它定义成一个新类。这是常用的一种方法，一般讲也是可以

的,但这样做也有问题,比如原先的类中有已经发现的错误,则复制类的同时也把错误复制了一遍,当发现错误后需要在两个类中修改。而且,这样定义的两个类是完全不同的两种数据结构,根本无法充分利用 OOP 的强大功能。

为解决这个问题,所有面向对象的编程语言都允许用户从现有的一个类出发定义一个新类。这项技术叫做继承,继承是面向对象语言(如 Delphi)最重要的功能之一,也是其广为流行的原因之一。继承性的突出特点之一是代码的可重用性。

继承性的含义直接而明显,它是指把一个新的类定义为已存在对象的后代:新对象继承了旧类的大部分内容,在新对象中添加新内容以前,父类的每一个字段和方法都已存在于子类中,父类是创建子类的基石。它能使子类利用父类的属性,继承父类的字段、属性、方法和事件。子类除了具备父类的属性外,还可以给继承来的属性添加新的成员。图 3-11 说明了一种生物的继承关系。每一个后代都具有其祖先的特性,即继承了祖先的特性。



图 3-11 生物之间的继承关系

将这种关系用程序代码写出来,就是类的继承关系。

```

Type
TAnimal=Class(TObject)
private
age:Integer;
eat:Boolean;
sleep:Boolean;
walk:Boolean;
public
function GetAnimalType:Boolean;
end;
  
```

在上面的程序代码中,定义了一个 TAnimal 类,它的父类是 TObject,这是一个关于动物特性的说明,包括 4 个字段和 1 个方法。接下来定义另一个类 TMammal,由于它和类 TAnimal 有很多相同之处,因此不必重新定义一个类,只需要继承类 TAnimal 中的字段和方法的定义。下面是类 TMammal 的定义:

```

type
TMammal=Class(TAnimal)           //继承了类 TAnimal
private
    milk:Boolean;
    hair:Boolean;
end;
  
```

继承性虽然有很多的优点,但有时候也会遇到一些问题。例如,如果我们不需要使用

父类中的某些字段，怎么办呢？实话说，没有办法，你只能不去用它们，就像它们不存在一样。不过，如果你觉得父类中的某个方法不能适应你的要求，要修改这个方法，那么是完全可以的。你只要在子类中把要修改的方法重新定义一遍就可以了。这种方法的重新定义就叫做覆盖。

3.5 类的多态性

多态性是在对象体系中把设想和现实分开的手段，如果说继承性是系统的布局手段，多态性就是实现其功能的方法。多态性意味着某种动作可以由特定的方式来实现，这取决于执行该动作的对象。多态性允许以类似的方式处理类中类似的对象。应用程序根据特定的任务分解成许多对象，多态性把父类设计时的一些设想，如新对象的创建、对象在屏幕上的重显、属性的定义等，留给知道该如何具体处理它们的对象去实现。

函数和过程通常是静态的，也就是说，方法的调用在编译和链接过程中就确定了。但是对于类来说，存在一种多态性，就是在运行时才确定调用哪个方法，方法的调用取决于对象的类型。我们看下面这个示例。

```

type
  TShape=Class                                //定义了一个 TShape 类
  ...
  procedure Paint(X,Y :Integer);Virtual;
  end;
  TCircle=Class(TShape)                      //继承了 TShape 类
  ...
  procedure Paint;Override;                  //重载了 Paint 方法
  end ;
  TRectangle=Class (Tshape)                  //继承了 TShape 类
  ...
  procedure Paint;Override;                  //重载了 Paint 方法
  end;
  ...
  Var
  MyShape : TShape;
  x,y:Integer;
  begin
  x:=50;
  y:=50;
  MyShape:=TCircle.Create;                  //创建 Tcircle 对象
  MyShape.Paint(x,y);                      //调用 Tcilcle 对象的 Paint 方法
  MyShape.Free ;                          //释放对象
  MyShape:=TRectangle.Create;               //创建 Trectangle 对象
  MyShape.Paint (x,y);                     //调用 Trectangle 对象的 Paint 方法
  MyShape.Free;                            //释放对象
  end.

```

在这个例子中，首先定义了类 TShape 用来表示基本的几何图形。这个类中有一个方法 Paint 被定义成了虚拟方法（用关键字 Virtual），这个方法完成图形绘制工作。然后定义了两个子类 TRectangle 和 TCircle，在这两个类中分别重载了 Paint 方法（用关键字

Override)。

在主程序代码部分，定义了一个变量 `MyShape`，这个变量依次被赋予 `TRectangle` 和 `TCircle` 对象。当程序调用方法 `Paint` 时，究竟调用的是那一个类的 `Paint` 呢？是基类还是派生类？很显然，在编译时这是无法确定的，而需要编译器在运行时根据调用这个虚拟方法的对象实例来确定。在上面的例子中，当调用方法 `Paint` 的对象是 `TRectangle` 时，就调用类 `TRectangle` 的 `Paint` 方法；当调用方法 `Paint` 的对象是 `TCircle` 时，就调用类 `TCircle` 的 `Paint` 方法。

注意：重载的方法必须与基类中的同名虚拟方法在参数个数、参数顺序和数据类型上完全相同。如果是函数的话，返回类型也必须相同。

多态性主要是针对方法的。虽然前面讨论了多态性的概念，但是对于与多态性密切相关的虚拟方法还没有讨论。另外，Delphi 的类中还有多种其它方法，它们是静态方法、动态方法、抽象方法。下面就一一进行论述。

3.5.1 静态方法

除非特别指明，否则声明的方法都是静态方法。静态方法与一般的过程或者函数没有区别。编译器能够确定静态方法的确切地址，并且在编译期间就可以链接它。

静态方法的主要好处是启动速度非常快，因为编译器可以确定方法的确切地址。与之相反，虚拟方法和动态方法在运行时使用间接的方式查找方法的地址，这会花费比较长的时间。

在被派生类继承时静态方法不会改变。如果一个类中包含了一个静态方法，那么它的派生类将在相同的地址共享相同的方法。这就意味着不能重载静态方法，一个静态方法总是完成一模一样的工作，不管是哪个类调用了它。如果在派生类中声明了一个与基类中的静态方法同名的方法，那么新的方法将代替旧的方法。这种新方法代替旧方法的过程就是前面介绍过的覆盖。

在如下所示的代码段中，第一个组件声明了两个静态方法。第二个组件声明了两个同名的静态方法，它们替换了从第一个组件中继承的方法。

```
type
Tvehicle=class (TObject)
procedure move;
procedure wheel;
end;
Tcar=class (Tvehicle)
procedure Move;
function wheel(number:Integer): Boolean;
end ;
```

//重载了 move 方法
//没有重载函数 wheel

3.5.2 虚拟方法

虚拟方法使用了一种比静态方法更加复杂、更加灵活的机制。虚拟方法可以被派生类修改和扩展，但是在父类中仍然可以被调用。虚拟方法的地址不是在编译期间确定的，而是在运行期间由定义方法的对象来寻找地址的。

要定义虚拟方法，可以在方法声明的后面增加指令 `Virtual`。指令 `Virtual` 将在对象的虚

拟方法表（VMT）中创建一个表项。VMT 保存着对象中所有虚拟方法的地址。

当你从一个已经存在的类中派生一个新类时，新类就会建立自己的 VMT，它包括父类的 VMT 的所有表项加上在新类中定义的虚拟方法。

3.5.3 重载方法

重载一个方法的意思是扩展或者优化它，而不是替代它。一个派生类可以重载任何一个它继承的虚拟方法。

要在子类中重载某方法，可以在该方法声明的结尾增加一个 `Override` 指令。注意，有三种情况重载一个方法可能会引起编译错误：

- 该方法在父类中不存在。
- 父类中的该方法是静态的。
- 声明不是完全相同（参数的类型和个数不同）。

3.5.4 动态方法

动态方法与虚拟的方法只有一点轻微的不同。因为动态方法在对象的虚拟方法表中没有表项，因此使用它们可以减少内存的消耗量。但是启动动态的方法有时候比一般的虚拟方法还要慢。如果一个方法需要经常调用，或者执行时间是至关重要的，那么应该把它声明为虚拟类型而不是动态类型。

对象必须存储它们的动态方法的地址。但是与从虚拟方法表中接收表项不同，动态方法清单只包含被特殊的类引入或重载的方法（虚拟表与之相反，包含对象中所有的虚拟方法）。

要使方法动态，可以在方法声明后增加指令 `Dynamic`。

3.5.5 抽象类成员

当一个方法被声明为抽象类时，在派生类中你必须重新声明和实现它。Delphi 不能创建包含抽象成员类实例。例如，下面就是一个抽象类。

```
type
  TAbstractClass=Class
    procedure Method1; Virtual ; Abstract;
  end ;
```

在类 `TAbstractClass` 中，我们声明了一个方法，因此这个类也就成为了抽象类。

注意：在声明抽象方法时，`Virtual` 或 `Dynamic` 指令必须出现在 `Abstract` 指令之前。

当需要从 `TAbstractClass` 派生一个新类时，必须实现这个抽象的方法，否则这个新类也是抽象类，也无法实例化，如下所示。

```
type
  TNewClass=Class (TAbstractClass)
  ...
  procedure Method1;Override;
  end;
  ...
  procedure TNewClass.Method1;
  begin
  ...
```

```
end;
```

像这样定义的类 TnewClass 就可以被实例化了。

3.6 异常处理

异常处理是一种处理错误的手段,使得应用程序能够从致命的错误中很好地恢复。在早期的 Delphi 中,异常是由 Object Pascal 语言来处理的。从 Delphi 2.0 开始,异常成为 Win32 API 的一部分。用 Object Pascal 来处理异常比较简单了,因为异常中包含了错误的位置和特征信息。这使得异常的使用和实现与普通的类一样。当一个错误或其他一些事件中中止了程序的正常运行,系统就会抛出一个异常。

Delphi 中包含了一些预定义的通用的程序错误异常,例如内存不足、被零除、数字上溢和下溢以及文件的输入输出错误,可以定义自己的异常类来适应程序的需要。通过 Delphi 的异常处理机制,可以捕获这个异常并进行处理。

异常实际上是一些对象,可以是任何类的一个实例。但是通常是自己定义的一个从 Exception 类派生出的异常类,定义方法与普通类的定义方法基本一致。Exception 类是在 SysUtils 单元中定义的。如果一个程序的 uses 语句中包含了 SysUtils 单元,发生运行错误时就会抛出一个异常。

可以利用类的继承性将一组异常组合成一个系列。比如,在 SysUtils 单元中就定义了有关数学方面的一组异常类。有时在异常类中还定义一些字段、属性和方法,通过它们可以指示一些错误信息。

3.6.1 异常控制语句 try...except

在 try...except 语句中可以进行抛出异常和处理异常的工作。try...except 的一般形式如下:

```
try
    语句 1;
...
语句 n;
except
    on 异常情况 1 do 处理语句 1;
    on 异常情况 2 do 处理语句 2;
    ...
    on 异常情况 N do 处理语句 N;
else
    处理语句;
end.
```

对于有些操作,在异常处理部分要进行,在正常情况下也要进行。例如,在正常情况下,使用完文件之后要关闭文件。如果在对文件操作的过程中出现了异常,也需要关闭已经打开的文件。这时,就可以把关闭文件的过程放在 try...finally 语句的 finally 部分,不管 try 部分的操作是否正常,都要进行 finally 部分的操作。

通常 try...finally 语句的形式如下:

```
try
```



```

    语句块 1
finally
    语句块 2
end.

```

可以看到, try...finally 语句的用法与 try...except 语句的用法很相似。“语句块 1”可为简单语句,也可以为复合语句。如果在“语句块 1”中抛出了异常,程序立即转到 finally 部分;如果在“语句块 1”中执行了 Exit、Break 或 Continue 过程而导致程序的控制离开“语句块 1”部分时,程序也会跳转到 finally 部分;如果在 try 部分正常执行完毕,接着执行的还是 finally 部分。

下面的例子介绍了在文件输入/输出时,怎样用异常处理。可以区分与 try...except 语句和 try...finally 语句的用法。

```

program project1
uses Classes, Dialogs;
{$APPTYPE CONSOLE}
Var
F:TextFile;
S:String;
begin
AssignFile(F, 'Fl.TXT');
try
    reset(F);
    try
        readln(F, S);
    finally
        CloseFile(F);
    end;
except
    on EInOutError do
        ShowMessage('Error in FileIO ');
end;
end.

```

在内层的 try...finally 代码块用来确保文件总是关闭的,而不管是不是发生了异常。这段代码的执行过程是:先执行 try 与 finally 之间的代码;如果执行完毕或出现异常,就执行 finally 与 end 之间的代码;如果确实有异常发生,就跳到外层的异常处理块。这样,即使出现异常,文件也总是关闭的,并且异常总能得到处理。

在 try...finally 模块中,finally 后面的语句不管有没有异常都会被执行。因此,finally 后面的语句不能以发生异常为前提。另外,由于 finally 后面的语句并没有处理异常,因此,异常被传递到下一层的异常处理模块。

外层的 try...except 块用于处理程序中发生的异常。在 finally 中关闭文件后,except 块显示一个信息,告诉用户发生了 I/O 错误。

这种异常处理机制比传统的错误处理方式优越,它使得错误检测代码从错误纠正代码中分离出来。这是一件好事情,它会使程序更可读,它使得用户能集中处理程序的其他代码部分。

使用 try...finally 代码块但不捕捉特定种类的异常是有一定意义的。当代码中使用

try...finally 块的时候,意味着程序并不关心是否发生异常,而只是想最终总是能进行某项任务。finally 块最适合于释放先前分配的资源(例如文件或 Windows 资源),因为它总是执行的,即使发生了错误。不过,很多情况下,可能需要对特定的异常做特定的处理,这时候就要用 try...except。

下面是用 try...except 块来捕捉特定的异常的例子:

```

program project1;
{$APPTYPE CONSOLE }
Var
R1, R2:double;
begin
while true do
begin
try
write {'Enter a real number :'};
readln (R1);
write {'Enter another real number :'};
readln (R2);
writeln('The first number divided by the Second is:', (R1/R2):5:3);
except
on EInOutError do
showMessage{'It is not a valid number !'};
On EZeroDivide do
ShowMessage ('Can not divide by zero !');
end;
end;
end;
end .

```

尽管在 try...except 块中可以捕捉特定的异常,也可以用 try...except...else 结构来捕捉其他异常。当使用 try...except...else 结构的时候,应当明白 else 部分会捕捉所有的异常,包括那些程序员并没有预料到的异常,例如内存不足或其他运行期异常。因此,使用 else 部分要小心,能不用则不用。当进入不确定的异常处理过程中时,应当一直触发这个异常。

3.6.2 raise 语句

使用 raise 语句调用一个异常类的构造函数,并抛出一个异常。例如:

```
raise EInOutError.Create
```

通常,raise 语句的形式如下:

```
raise Object at address
```

其中 Object 和 at address 是可选项, address 通常是一个指向过程或函数的指针。一个抛出的异常在处理过后自动地被删除,一般不去主动地删除一个异常对象。

3.6.3 异常类

异常是一种特殊的对象实例,它在异常发生时才实例化,在异常被处理后自动删除。异常对象的基类被称为 Exception。

在异常对象中最重要的元素是 Message 属性,它是一个字符串,提供了对异常的解释,由 Message 所提供的信息是根据产生的异常来决定的。

如果定义自己的异常对象,一定要从一个已知的异常对象,例如 Exception 或它的派

生类派生出来, 因为这样, 通用的异常处理过程才能捕捉到这个异常。

当程序员在 `except` 块中处理一个特定的异常时, 可能会捕捉到该异常的派生异常。例如, `EMathError` 是所有与数学有关的异常 (例如 `EZeroDivide`、`EOverflow`) 的祖先。凡是没有显式地处理的异常最终将被传送到 Delphi 运行期库中的默认处理过程并在此得到处理。

3.6.4 添加了异常处理的程序实例

在第2章中, 我们曾练习了一个“随机加减法测试”的程序。但是在该程序中, 并没有对用户输入的数值是否合法进行检测。一旦用户输入包含字母的内容, 程序就会出现错误。为了检测这种情况, 我们应当为程序添加异常处理。

为了节约篇幅, 这里就不将全部程序代码罗列, 仅将添加了异常处理语句的 `Edit1` 组件的 `KeyPress` 事件的代码列出。


[例 3-2] 添加了异常处理的随机加减法程序

(1) 打开该“随机加减法测试”程序。

(2) 切换到代码编辑器窗口, 在 `TForm1.Edit1KeyPress()` 事件中添加如下阴影部分所示代码。

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
  answer: integer;
begin
  if key=chr(VK_RETURN) then
  try
    //这是正常执行语句块
    begin
      if label2.Caption='+' then
        answer:=strtoint(label1.Caption)+strtoint(label3.Caption)
      else
        answer:=strtoint(label1.Caption)-strtoint(label3.Caption);
      if answer=strtoint(edit1.Text) then
      begin
        showmessage('回答正确');
        button1.SetFocus ;
      end
      else
      begin
        showmessage('回答错误');
        edit1.SetFocus ;
      end;
      edit1.SelectAll ;
    end;
    //正常执行语句块结束
  except
    //异常处理语句块开始
    begin
      showmessage('请输入数字');
      //显示提示信息
      edit1.SetFocus ;
      //将焦点继续定位于 Edit1 组件
    end;
  end;
```

end;

(3) 代码添加完毕后, 单击  按钮, 对程序进行编译和运行。在答案栏输入数字和字母, 如图 3-12 所示。

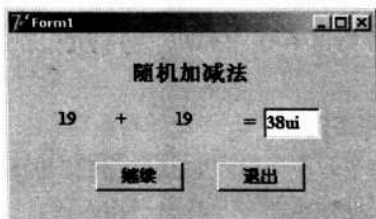




图 3-12 运行程序, 输入字母和数字

(4) 按下 **Enter** 键后, 会出现如图 3-13 所示错误提示框, 说明输入的内容不是一个有效的整型值。

(5) 单击  按钮, 对话框消失, 程序会进入一种暂停状态; 再次单击  按钮, 会出现如图 3-14 所示信息对话框, 显示我们设定的提示信息。

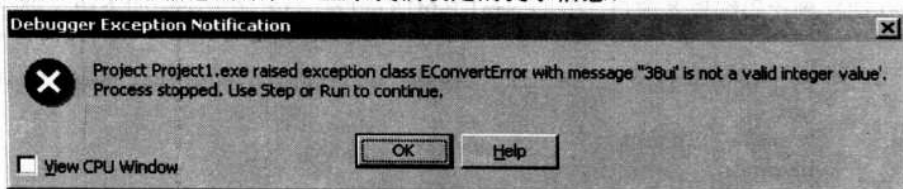


图 3-13 编译运行时出现的错误提示框



图 3-14 信息对话框

提示: 图 3-13 所示的对话框是在调试状态下出现的错误提示。当直接在 Windows 下运行程序的可执行文件 (.exe) 时, 就不会再出现这个提示框了; 而是会直接出现图 3-14 所示的对话框。

3.7 小结

类是面向对象编程语言中最为重要和基本的概念。本章在上一章的基础上进一步介绍了类的概念及一些相关知识, 并通过实例说明了类的定义和使用方法。当然, 作为基础教程, 这里并没有用更多的篇幅讨论这方面的知识, 如果读者希望多了解一些面向对象编程的内容, 请参考其他相关书籍。

第4章 应用程序开发框架

Delphi 开发的应用程序是以项目的形式来组织的，每一个项目包含了编译后的目标应用程序所需要的所有文件。了解各个文件的作用，认识文件的内容结构，对于进行程序开发是至关重要的。利用项目管理器对项目进行管理，了解应用程序类的常见属性和方法，掌握程序编译、调试的基本方法和技巧，也是进行 Delphi 程序设计所必须掌握的基础知识。

4.1 Delphi 的文件结构

Delphi 是以项目为中心的开发产品。当我们创建一个新的应用程序时，Delphi 会自动创建一个项目文件。这意味着每个应用程序都是一个项目，由一个或多个文件以及项目文件组成。组成项目的几种文件包括：源代码、窗体、编译过的单元、配置、选项、包以及备份文件等。下面介绍各个文件的结构和作用。

4.1.1 项目文件 (.dpr)

项目文件默认的名称是“Project1”，扩展名为“.dpr”（Delphi Project 的缩写）。项目文件本身也是含有 Object Pascal 的源代码的文件。当应用程序启动时，首先执行项目文件对应的代码，通过这些代码完成系统总流程的控制。

创建一个新的应用程序，然后选择【Project】/【View Source】菜单命令，可以看到项目文件“Project1.dpr”的源代码，如图 4-1 所示。

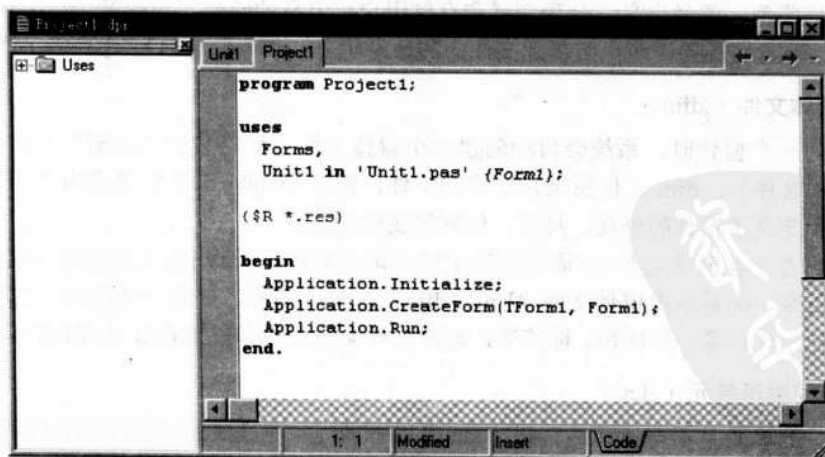


图 4-1 项目文件的源代码

上面列出的代码都是由 Delphi 自动添加的, 其中:

【program】是一个关键字, 说明当前文件为项目文件, 同时也表明了项目编译后的可执行文件的名称。

【uses】子句后部是逗号分隔的列表, 包含了项目文件中引用的其他文件。被引用的文件可以是系统创建的单元文件, 也可以是用户创建的单元文件。

【\$R 语句】是编译器指令, 告诉编译器去链接一个资源文件。编译器指令{\$R *.res}指示 Delphi 在与项目同名、扩展名为*.res 的文件中, 查找 Windows 资源信息。

【begin】与【end】语句定义了程序的开始与结束, 典型的 Delphi 应用程序以 Application.Initialize (对应用程序进行初始化) 开始, 以 Application.Run (运行应用程序) 结束。

一般情况下, Delphi 7 能够自动维护项目文件而不需要对其进行编辑, 许多时候可能完成了整个应用程序的设计工作而无需知道项目文件的存在。但是了解了项目文件, 可以使我們更好地进行程序设计, 可以通过向工程文件中添加代码来使程序在主窗体启动前执行一些特定的操作, 例如打开一个启动画面、初始化数据库链接等。

修改.dpr 文件中的代码是可以的, 但通常这并不必要。除非确实有修改它的原因, 否则最好还是让 Delphi 来管理项目源文件。

4.1.2 单元文件 (.pas)

Delphi 应用程序的源代码一般都保存在单元文件中, 单元文件的扩展名为“.pas”。通常每个项目至少有一个单元。Delphi 7 通常包含 3 种类型的单元文件:

(1) 窗体和数据模块 (DataModule) 组件等对应的单元文件。这是最常见的单元文件, 这些文件由 Delphi 7 在创建窗体等组件时自动创建。例如, 每一个窗体对应一个单元文件, 一般用来保存窗体的事件处理程序。

(2) 创建组件的单元文件。一般用来保存窗体的事件处理程序。

(3) 通用的单元文件。由程序员手动创建的单元文件, 一般用来声明应用程序中需要访问的数据类型、常量定义、全局变量和存储用户自定义的函数、过程等。

在 4.2 节我们还要详细说明单元文件的内部结构。

4.1.3 窗体文件 (.dfm)

当创建一个窗体时, 系统会自动创建一个窗体文件 (扩展名为“.dfm”) 和一个单元文件 (.pas 文件)。dfm 文件原来是二进制文件, 但在 Delphi 5 之后已成为脚本化的文本文件, 其中定义了窗体的外观、尺寸、位置等属性信息。

如果想看一看窗体文件, 只需在窗体上单击鼠标右键, 从弹出的快捷菜单中选择【View as Text】命令, 则显示出窗体文件“Unit1.dfm”的资源脚本, 如图 4-2 所示。如果窗体中还包含其他组件对象, 如按钮、标签等, 则这些对象的大小和位置也会出现在窗体文件中。

4.1.4 中间编译单元 (.dcu)

编译过的单元是不可执行的, 以“.dcu”为扩展名。在建立应用程序的链接阶段, 所有的.dcu 文件链接起来成为可执行程序。这些文件是可以删除的, 但最好还是让 Delphi 来管理这些文件。

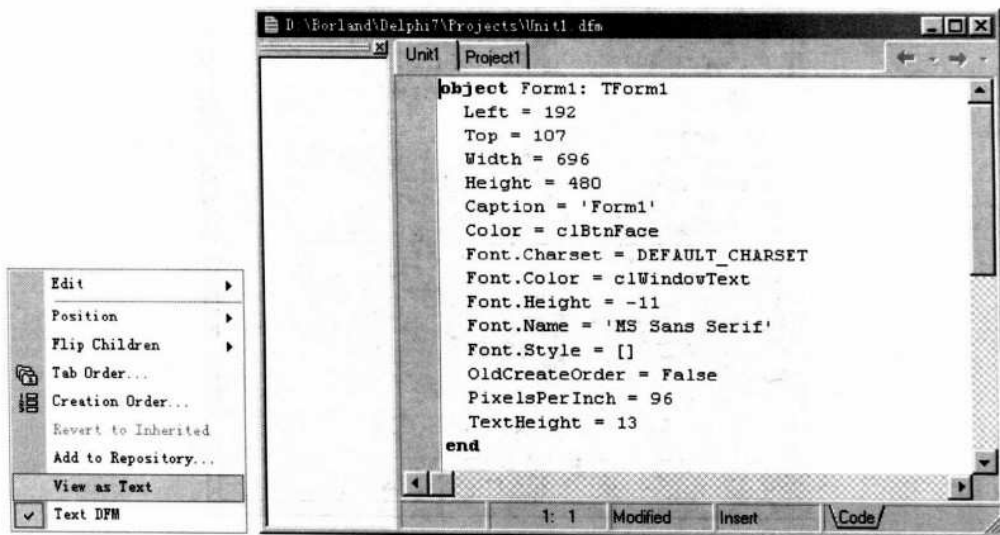


图 4-2 窗体文件

如果只分发 DCU 文件, 编译过的代码在 Delphi 将来发布的版本中可能是无效的。如果不升级, 程序员就无法继续使用这些 DCU 文件。如果出售的是工具软件, 可以考虑公开源代码并在许可协议中规定可接受的使用方式。

建立应用程序时, Delphi 会把源代码文件与编译过的单元进行比较。如果源代码没有被修改过, Delphi 就不需要重新编译源文件。如果希望其他的开发者使用这种产品来建立应用程序, 可以只发布 .dcu 文件, 而不发布源文件。以这种方式, 其他的开发者可以使用该代码, 而无须确切知道代码是如何编写的。发布 .dcu 文件确实是一种方法, 这样就可以传播私有代码而无须将其公开。

4.1.5 备份文件 (.~pas)

无论何时, 只要改动了一个文件并且进行了保存, Delphi 都会对该文件已存在的版本进行备份。这样, 就拥有该文件的最新版本以及一个稍早一点的版本。对备份文件的命名惯例是: 它的名字与原文件几乎相同, 但在 '.' 与扩展名的第一个字母之间插入了一个 ~ (波浪线)。例如, test1.pas 就变成了 test1.~pas。

由于备份文件不会以其他方式修改, 因此要恢复备份, 只要在资源管理器中重命名备份文件, 去掉其名字中的波浪线即可。如果开始开发后一直使用某种版本控制产品, 在更新文件的已归档版本时经常保存, 那么不会丢掉任何修改。

删除备份文件不会有什么问题, 系统默认情况下会进行文件备份, 如果想取消文件备份, 可以通过选择【Tools】/【Editor Options】菜单命令, 在出现的【Editor Properties】对话框中, 选择【Display】选项卡, 取消对【Create backup file】属性的选择。如图 4-3 所示。

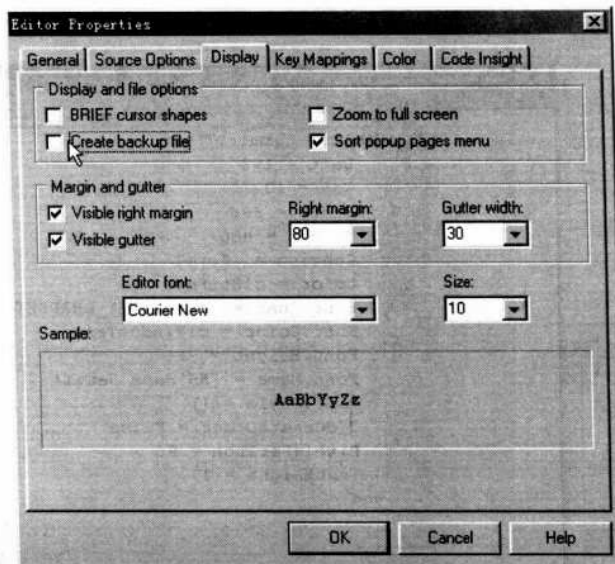


图 4-3 【Editor Properties】对话框

4.1.6 资源文件 (.res)

资源文件是用于保存应用程序的图标、应用程序信息以及其他资源信息，扩展名为“.res”。资源文件是二进制数据，在编译时，编译器会自动将这些资源链接到应用程序的可执行文件中。要把资源加入到应用程序中，首先要创建一个单独的资源文件（图像文件、图标文件、光标文件等），然后将这个文件链接到项目中去。

4.1.7 应用程序文件 (.dll, .exe, .ocx)

读者可能已经熟悉了几种基本类型的应用程序文件，Delphi 都能够创建所有种类的应用程序文件，包括动态链接库 (.dll)、可执行文件 (.exe) 和 ActiveX 组件 (.ocx，因为以前 ActiveX 支持的是 OLE 命名规范)。

其中每一种文件都代表一种终端产品，又经过编译和链接的代码组成。可执行应用程序是单独运行的程序或进程外服务器。动态链接库代表资源文件或进程内服务器，而 ActiveX 组件是用于建立其他程序的支持性组件。

4.1.8 配置与选项文件 (.dof)

当改动【Project Options】设置时，Delphi 把改动存储在一个具有 .dof 扩展名的文件中。当所做的改动影响到了应用程序的编译方式时，这些改动将以文本形式存储到 .cfg 文件中或配置文件中。

实际上分别生成了几个文件来存储配置、选项和其他类型的信息。如果在【Help Topics】对话框的【Find】属性页中查找【Generated Files】，即可得到关于存储项目信息的所有生成文件的信息。大部分情况下，Delphi 自动管理这些文件。请勿删除它们。

提示：无须记住所有的文件及其作用，只需记住一条规则：只能删除扩展名中带有 -

(波浪线)的文件、.dcu 文件或者不需要的文件。

4.2 单元文件的内部结构

经常使用的最重要的文件是源代码文件,称为单元。没有了单元,窗体文件只是一种画出图像的复杂方法。理解单元的各种不同方面是很重要的。单元不必有相关的窗体,反之则不然。

在 Pascal 中,单元是模块化编程的基础。Delphi 引入的类是建立在单元的概念基础之上的。在 Delphi 应用程序中,每个窗体一般都有与之相对应的单元。向项目中添加新窗体时,实际上就是添加了新单元。单元不仅能定义窗体,还能定义并使用各种例程。

4.2.1 认识单元文件

单元的概念很简单,每个单元都有一个名称(在每个项目中是唯一的,并与它的文件名相对应)用于标识单元文件;一个接口部分(Interface)用于声明对其他单元该部分是可以访问的;一个实现部分(Implementation),包括了代码实现部分和隐含部分。另外,有的单元还有一个初始化部分(Initialization),用于执行初始化功能,还有一个结束部分(Finalization)执行程序终止功能,它们都是可选部分。

带有全部可能部分的单元结构如下:

```

unit Unit1                                     //单元文件名

Interface                                     //接口部分
uses                                           //需引用的其他单元名
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs;

type                                           //定义类
TForm1 = class (TForm)
private
  { Private declarations }
Public
  { Public declarations }
end;

var                                           //定义全局变量
  TForm1: TForm1;

implementation                               //实现部分
uses unit2                                   //定义引用的单元

{$R *.dfm}                                   //定义资源文件
procedure TForm1.Button1Click(Sender: TObject);
//实现具体的事件操作
begin
...
end;
```

```

        initialization                //初始化部分（不一定需要）
    ... ..
        finalization                //结束部分（不一定需要）
    ... ..
    end.

```

【Interface】中的 `uses` 子句指出了本单元中引用了哪些系统预先定义的单元文件，被引用的单元之间以逗号分隔。这一部分一般是由 Delphi 自动维护的，在窗体文件产生时就产生了这部分代码。但是如果要引用一些特别的单元，那么就需要手动将其添加到 `Uses` 子句中。

在【Implementation】中的 `uses` 子句定义了程序实现部分引用的单元文件，这些单元文件是在用户当前项目中创建的。例如在第3章的【例3-1】中，我们就创建了两个窗体，并且相互建立了引用关系。

4.2.2 单元的各个部分

单元包括单元名和接口部分，接口部分中包含了类型声明、变量声明，如果需要还会有常数。单元的下半部分，即 `implementation` 之后，是实现部分，该部分可包含类型声明、变量声明、常数和过程。通常在实现部分会看到代码。

如果单元与数据模块或窗体关联，紧接关键字 `implementation` 之后是 `$R` 编译器指令，用于查找资源。关键字 `end.` 标志了文件的结尾。可以选择加入关键字 `initialization` 和 `finalization`，但它们不会被自动添加。`initialization` 代码在单元中的所有其他代码运行前运行，`finalization` 代码在单元中所有其他代码运行后运行。单元装载到内存后，马上运行 `initialization` 部分，而 `finalization` 代码刚好在单元被卸载之前运行。

1. Unit 语句

Unit 语句中包含了文件的名字。除了 Windows 文件系统存储文件时需要文件名以外，还可以把单元名作为定义命名空间的机制。例如，有一个单元名为 `test`，其中有个过程名为 `mytest`，另一个单元名为 `systest`，其中也有一个名为 `mytest` 的过程；为区分这两个过程，在调用 `mytest` 时，可以把去掉扩展名的单元名作为前缀。

这样，在解析 `test.mytest` 调用时，编译器将找到 `test` 单元中的 `mytest`；而另一个调用 `systest.mytest` 将对应到 `systest` 单元中的过程。unit 语句的形式是“unit 文件名”；其中文件名是在保存文件时由 Delphi 管理的，不包括 `.pas` 扩展名。

2. 接口部分

可以认为单元分为两部分。上半部为接口部分，起始于包含关键字 `interface` 的那一行，结束于关键字 `implementation` 之前，其余为第二部分。在最简单的意义上，这两部分的作用是互补的。上半部，或接口部分，描述了应用程序的其余部分在该单元中可以访问哪些东西。下半部，即实现部分，通常是编写运行代码之处。

最重要的是要记住：接口部分没有运行代码，但包含了其他单元可以访问的类型、常数和变量等。它也描述了该单元中可调用的过程和可使用的数据。

3. 实现部分

实现部分是编写运行代码之处，也可以包含类型、变量和常数。定义于接口部分的变

量、类型和常量可以在单元外使用，与此相反，在实现部分定义的只能在单元内部使用。

过程或函数的声明是不包含代码体的语句，即不包含 `begin` 和 `end` 语句及二者之间的代码。过程性声明放在接口部分。

过程或函数的定义包含声明部分和函数体，即实际的代码。过程性的定义放在实现部分。

另外，定义在实现部分的过程和函数，如果在接口部分没有相应的声明，则只能在单元内部使用。如果希望其他单元可以访问过程和函数，则要将其声明放在接口部分而将定义放在实现部分。

4. 定义 Uses 子句

Uses 子句指示编译器添加在列出的各个单元中找到的代码，可将该语句在接口部分和实现部分各放一个。如果用到 Uses 子句，它将紧跟在 `interface` 或 `implementation` 关键字之后。

如果单元 Unit1 需要单元 Unit2 的代码，Unit2 也需要 Unit1 的代码，则在 Unit1 和 Unit2 的 Uses 子句中相互列出对方。但 Unit1 和 Unit2 不能在同一部分相互引用。例如，在接口部分 Unit1 可引用 Unit2，Unit2 也可引用 Unit1；但 Unit1 和 Unit2 不能在接口部分相互引用，这是个循环引用错误。而在实现部分两个单元可以相互引用。

注意：所有的单元都隐式引用了 `system.pas` 单元，该单元无法显式引用。

如果改变了 Unit1 接口部分的 Uses 子句，而 Unit2 引用了 Unit1，那么 Unit1 和 Unit2 可能会被重新编译。如果 Unit1 在接口部分引用了 Unit2，而改变了 Unit2 实现部分的 Uses 子句，则 Unit2 必须重新编译，而 Unit1 则无需如此。

5. Type 子句

接口部分和实现部分都可能 Type 子句。按惯例，大多数 Type 子句都位于接口部分。类型声明可以定义集合、数组、记录和类等。在关键字 `Type` 之后可引入新的类型。

6. 变量部分

接口部分和实现部分均可包含 Var 子句。当要定义其他单元可访问的变量时，请把这些变量放在接口的变量部分，然后在使用变量单元的 Uses 子句中包含该单元即可。接口部分定义的变量可认为是全局变量，请谨慎使用。由于无法确保全局变量不被其他人误用，所以应适当加入说明以防止误解。

实现部分定义的变量只能在所定义的单元内访问。这种变量被称为本地变量，在单元内可随意访问，但不能被使用该单元代码的其他单元所引用。

7. 资源声明

在“test1Pro.dpr”文件的代码中可看到 `{$R *.res}` 编译器指令，文中提到过，该指令指示编译器包含与该单元同名的 `.res` 文件。`$R` 指令通常只出现在具有窗体的单元中，它们也可能是开发者因为某种原因添加的。

8. Initialization 部分的使用

单元的 initialization 部分的代码将在单元中任何其他代码运行前运行。在 initialization

与 finalization 或 end 关键字之间的代码，将在单元向内存加载时运行。如果要使用全局变量或本地变量，可在 initialization 关键字后进行初始化。

9. Finalization 部分的使用

如果单元中有 initialization 部分，也可以使用 finalization 部分。可在 finalization 部分运行清除代码，并释放在 initialization 部分分配给对象的内存。finalization 部分由关键字 finalization 开始直到关键字 end（文件结尾）。finalization 部分与对应的 initialization 部分按相反的顺序运行。例如，Unit1，Unit2，Unit3 按顺序装载入内存，则其 finalization 部分将按 Unit3，Unit2，Unit1 的顺序运行。

单元的接口部分可以声明许多不同的成员，包括函数、过程、全局变量和类型说明等。Delphi 每次在建立新窗体时都进行同样的工作。单元唯一的元素是窗体数据类型的定义 TForm1 和该类型的一个全局变量 Form1。单元中仍可以用普通的单元、函数和过程，和可以包含未引用的表单和其他成员的类。创建一个单元与创建表单之间没有关系，可以使用菜单【File】/【New】，选择【New】标签页下的 Unit 即可。

4.3 项目管理

由于 Delphi 项目包含了多个文件，其开发和设计都是比较复杂的工作，因此通过良好的组织和对开发过程的优化，可以大大加快应用程序的开发进度。一般对项目的管理主要是通过【Project Manager】（项目管理器）和【Project Options】（项目选项）来实现的。

4.3.1 Project Manager（项目管理器）

（1）创建一个新的应用程序。系统会自动创建一个【Form1】窗体和一个【Unit1】单元文件。

（2）选择【File】/【New】/【Form】菜单命令，再创建一个【Form2】窗体及【Unit2】单元文件。

（3）选择【View】/【Project Manager】菜单命令，可以弹出【Project Manager】对话框，如图 4-4 所示。

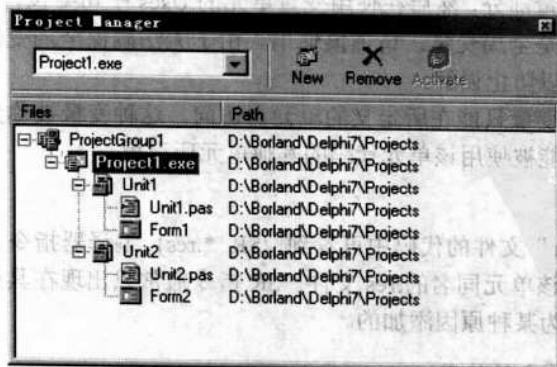


图 4-4 【Project Manager】对话框

可见,当前项目名称为“Project1.exe”,其中包含了两个 Unit,每个 Unit 中又包含了一个单元文件和一个窗体。

通过【Project Manager】对话框的目录树结构和弹出式菜单的常用项目管理命令,能够方便地管理项目。例如,可以向项目组中添加新的项目、或者删除不需要的项目等。对于多窗体应用程序项目的管理非常方便。

4.3.2 【Project Options】(项目选项)

选择主菜单中的【Project】/【Options】菜单命令,可以出现【Project Options for Project1.exe】对话框,利用它可以配置当前项目的一些常用参数,如版本信息、项目路径信息、编译信息以及图标等。如图 4-5 所示。

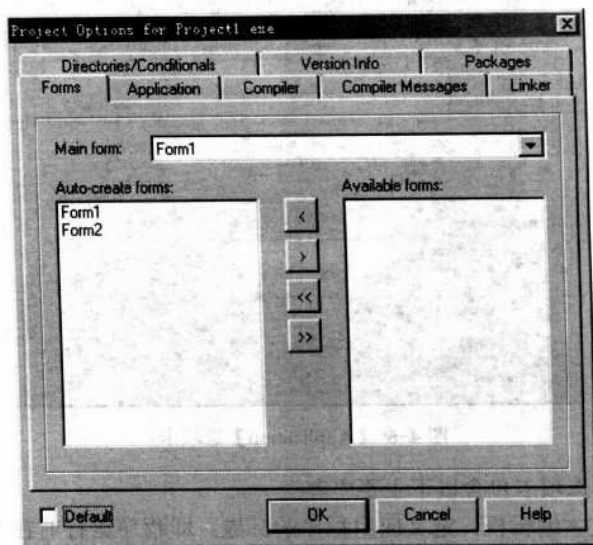


图 4-5 【Project Options for Project1.exe】对话框

1. 【Forms】选项卡

【Forms】选项卡主要用来设置应用程序的主窗体以及选择哪些窗体被系统自动创建、哪些窗体需要用户自己创建。









【Forms】选项卡包含以下主要内容:

(1) 【Main form】组合列表框:显示当前项目中的主窗体。如果当前项目中有多个窗体,可以通过鼠标单击下拉▼显示这些窗体,后选择其中的一个窗体为项目的主窗体。

(2) 【Auto-create forms】列表框:显示 Delphi7 在项目文件中能够自动创建的窗体列表。这些窗体的顺序也是有规律的,在程序运行时按照列表框中从上到下的次序被创建,而且列表中的第一个窗体将被设为主窗体。可以通过鼠标拖曳的方式调整这些窗体在列表框中的次序以改变其被创建的次序。需要注意的是,当某窗体一旦被指定为主窗体后将变为自动创建。

(3) 【Available forms】列表框:显示项目中拥有的 Delphi 7 未曾在项目文件中添加

自动创建代码的窗体列表。这些窗体需要程序员通过编制代码创建。

(4) 调整按钮 、、、：使窗体在【Auto-create forms】列表框和【Available forms】列表框之间切换，即调整窗体是否被自动创建。鼠标单击  按钮将选定的窗体由手动创建变为自动创建；鼠标单击  按钮将所有的窗体由非自动创建变为自动创建。同样，鼠标单击  按钮或  按钮能够将一个或所有的窗体由自动创建变为非自动创建。

2. 【Application】选项卡

【Application】选项卡是【Project Options】对话框中使用频率最高的选项卡，主要用来设置应用程序的标题、帮助文件、图标等。【Application】选项卡如图 4-6 所示。

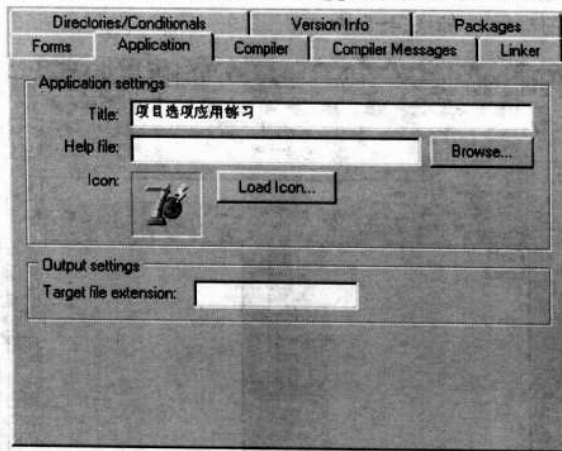




图 4-6 【Application】选项卡

【Application】选项卡包含以下主要内容：

(1) 【Title】文本编辑框：定义应用程序的标题，即程序运行时在 Windows 任务栏显示的程序名称。标题的最大长度为 255 个字符。

(2) 【Help file】文本编辑框：设置项目关联的帮助文件 (*.hlp) 的位置。用于在应用程序运行时提供联机帮助。单击  按钮会弹出寻找文件对话框，用于定位帮助文件。

(3) 【Icon】显示框：显示应用程序的图标。利用  按钮可以寻找和定位图标文件。

(4) 【Target file extension】文本编辑框：设置可执行文件的扩展名。

(5) 【Default】复选框：将当前选项卡的数据设为默认值。

3. 【Version Info】选项卡

【Version Info】选项卡如图 4-7 所示，主要用来设置项目的版本信息等。其中主要包含以下内容：

(1) 【Include version information in project】复选框：选中此项将在应用程序编译时加入版本信息。利用 Windows 系统的资源管理器，选择当前文件编译得到的可执行文件 (*.exe)，单击鼠标右键，在打开的弹出式菜单中选择【属性】菜单命令，就会显示软件

的版本信息，如图 4-8 所示。

(2) **【Module Version number】**：用于调整几个层次的版本序号。一般说来，**【Major】**是主版本号，**【Minor】**是次版本号，**【Release】**是发行版本号，**【Build】**是内部测试版本号。版本号的取值范围为“0~65535”。

(3) **【Auto-increment build number】**复选框：选中此项后，系统自动在每一次编译后使**【Build】**版本号增加 1。



图 4-7 【Version Info】选项卡

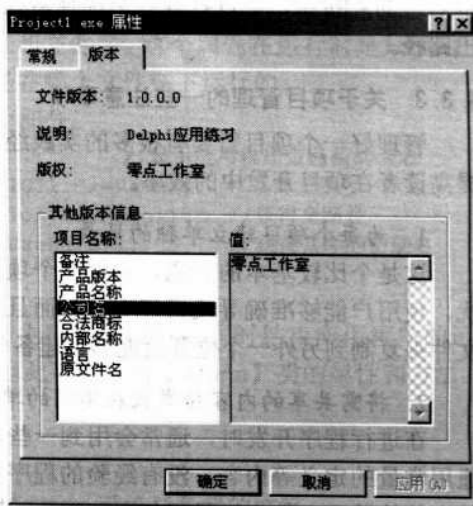


图 4-8 软件的版本信息

(4) **【Directories/Conditionals】**选项卡：**【Directories/Conditionals】**选项卡如图 4-9 所示，主要用来设置项目中相关联文件的位置等信息。

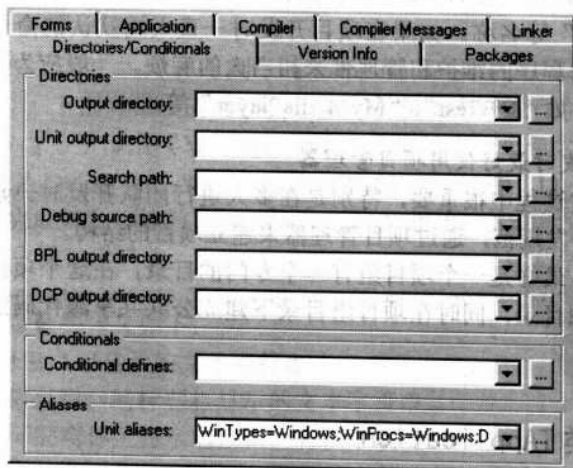


图 4-9 【Directories/Conditionals】选项卡


【Directories/Conditionals】选项卡包含以下主要内容：

【Output directory】：指定编译后的文件和可执行文件的保存位置。

【Unit output directory】：指定编译后的单元文件（.dcu）的保存位置。

【Search path】：指定寻找“.dcu”文件的路径。

【Debug source path】：设定 Debug 来源文件的路径。

当以上编辑框的内容为空白时，这些路径与存放单元文件的路径相同。鼠标单击编辑框右边的  按钮，可以打开一个选择路径的对话框。通过这个对话框，可以选择需要的输出路径。

4.3.3 关于项目管理的一些注意事项

管理好一个项目需要有很多的实践经验。下面提出一些项目管理方面的建议，有助于提高读者在项目开发中的效率。

1. 为每个项目建立单独的目录

这是个比较基本的问题，为每一个项目建立一个单独的目录可以使项目之间不互相干扰，使用户能够准确寻找目标文件，而且对项目文件备份也非常方便（直接将项目对应的文件夹复制到另外一个位置就能够创建备份）。

2. 将需共享的内容分类放在不同的单元文件中

在进行程序开发时，通常会用到一些需共享的内容，如全局变量、共享函数或过程、通用常量的定义等内容，没有经验的程序员常常将这些内容放置在第一次使用的地方（如某窗体的单元文件中）。最好把那些需要被其他单元或应用程序共事的内容放到一个单独的单元文件中。这样程序的结构化比较好，而且也使程序员不需要记忆太多的放置共享内容的那些单元文件，管理起来非常方便。

3. 养成良好的命名习惯

Delphi 7 在默认的情况下，对项目文件、窗体、单元、按钮等都采用“Project1”、“Form1”、“Unit1”、“Button1”等名称。如果在项目开发时一直采用默认的命名方式，那显然不是一个好主意，这会为程序的维护和管理带来相当大的麻烦。一般应为项目和内容定义一个有明确意义的名称，如“DBTest”、“MyMediaPlayer”等。

4. 多人合作开发时最好使用项目管理器

项目管理在开发过程中很重要，特别是在多人进行团队开发时。建议在整个小组投入开发以前建立一个开发规范，通过项目管理器来建立项目的结构。

对于文件存放，最好是一个项目组有一个专门的目录，在这个项目组目录下对各个项目再建立自己的单独目录，同时在项目组目录下建立公共共享源代码的目录和共享文档的目录。

4.4 应用程序类 TApplication

在前面的项目文件中已经见到了一个全局变量 Application，其类型为 TApplication。在

任何基于窗体的 Delphi7 应用程序中均有这个全局变量。TApplication 封装了一些很有用的属性和方法,使应用程序能够在 Windows 环境下良好运行。

在一般情况下,不必关心全局变量 Application 的作用情况。不过使用 TApplication 类提供的一些属性和方法能够给程序开发带来很大方便。

4.4.1 TApplication 的常见属性

1. 【ExeName】属性

【ExeName】属性能够返回应用程序的全路径和文件名。这个属性在运行时是只读的,不能修改。应用程序可以通过这个属性告诉用户是在哪个文件夹下运行的。

通过下面的代码能够访问应用程序名称的各个部分。

```
showMessage(Application.ExeName);           //显示应用程序带路径的全名
ShowMessage(ExtractFileName(Application.ExeName)); //获得文件名
ShowMessage(ExtractFilePath(Application.ExeName)); //获得全路径
ShowMessage(ExtractFileExt(Application.ExeName)); //获得文件的扩展名
```

2. 【MainForm】属性

【MainForm】属性指明应用程序的主窗体(主窗体关闭后将同时结束应用程序的运行)。【MainForm】属性的类型为【Tform】类,因此可以使用【Tform】类的属性和方法。

3. 其他属性

- 【Icon】: 设置应用程序最小化时的图标。
- 【Title】: 设置应用程序运行时,显示在 Windows 系统任务栏的文字。
- 【Active】: 显示应用程序是否为活动窗口(具有输入焦点的窗口)。
- 【Handle】: Windows 系统中的窗口句柄。
- 【HelpFile】: 指定帮助文件的文件名。
- 【ShowHint】: 是否显示提示。

4.4.2 TApplication 的常见方法

在前面项目文件的学习中,曾用到了【TApplication】类的【Initialize】、【Run】、【CreateForm】等方法。下面对常用的【TApplication】类的方法进行介绍。

(1) 【MessageBox】方法:调用【MessageBox】方法将弹出一个包含提示信息的对话框。【MessageBox】方法是 Delphi 7 对 Windows 的 API (应用程序接口) 函数 MessageBox 的封装,同前面讲过的完成类似功能的 ShowMessage 函数相比,其功能强大得多。

【MessageBox】方法的定义为:

```
function MessageBox (const Text,caption:PChar;Flags:longint =MB_OK ):Integer;
```

这个方法的各个参数的含义如下:

- Text: 在信息窗口中显示的文本字符串。
- Caption: 在信息窗口的标题栏显示的文本字符串。
- Flags: 在对话框窗体上显示的按钮。有以下取值:
 - MB_ABORTRETRYIGNORE: 对话框包含“终止”、“重试”、“忽略”3个按钮。
 - MB_OK: 对话框包含“确定”按钮。

MB_OKCANCEL: 对话框包含“确定”、“取消”按钮。

MB_RETRYCANCEL: 对话框包含“重试”、“取消”按钮。

MB_YESNO: 对话框包含“是”、“否”按钮。

MB_YESNOCANCEL: 对话框包含“是”、“否”、“取消”3个按钮。

● 返回值: 对应于各个按钮的操作, 共有7个返回值: IDOK、IDCANCEL、IDABORT、IDRETRY、IDIGNORE、IDYES 和 IDNO。

(2) 【CreateForm】方法: 【CreateForm】方法用于创建一个窗体的实例。一般不需要调用这个方法来创建窗体, Delphi 7 会自动在项目文件中添加创建窗体的代码。例如:

```
Application.CreateForm(TForm1, Form1);
```

自动被创建的窗体将出现在【Project Options】/【Forms】选项卡的【Auto-create forms】列表框中, 而出现在【Available forms】列表框中的窗体, 可以在程序的任何一个地方调用【CreateForm】方法来创建。其实, 【CreateForm】方法相当于窗体的【Create】方法, 但是【CreateForm】方法会在未创建主窗体时将当前创建的窗体设为主窗体。一般情况下不要调用【CreateForm】方法, 而调用窗体本身的【Create】方法。

【CreateForm】方法的声明如下:

```
procedure CreateForm(Formclass:TFormclass; Var Reference );
```

第1个参数 FormClass 用于指定窗体的类, 第2个参数返回创建的窗体的实例。

(3) 其他方法

● 【RUN】: 运行程序。

● 【Initialize】: 对程序进行初始化。

● 【Minimize】: 将应用程序的主窗体最小化。

● 【Restore】: 恢复应用程序的主窗体为最小化或最大化之前的状态。

下面我们通过一个简单的实例说明 TApplication 类的 MessageBox 方法的应用。

【例 4-1】 MessageBox 方法示例

窗体上有一个标签和一个按钮; 单击按钮, 会出现一个对话框, 其中包含了3个按钮; 任意按下一个按钮, 会在窗体上显示按下的按钮名称。画面如图4-10所示。

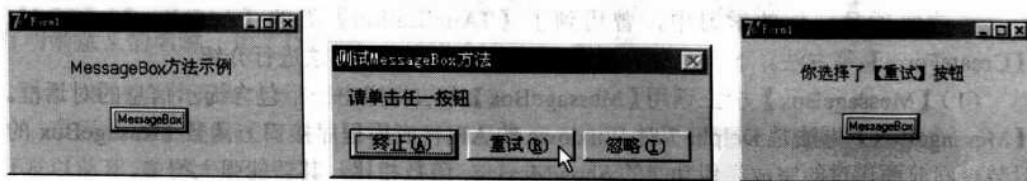


图 4-10 MessageBox 方法示例

(4) 创建一个新的应用程序。适当调整当前窗体的大小。

(5) 在组件面板的【Standard】选项卡中, 双击“Button”组件, 将其添加到窗体中; 利用【Object Inspector】面板设置按钮对象的【Caption】属性的内容为“MessageBox”。

(6) 再双击“Label”组件, 将其也添加到窗体中, 同理设置其【Caption】属性的内容及文字样式。

(7) 双击按钮对象, 进入代码编辑器窗口, 输入如下所示程序代码:

```

procedure TForm1.Button1Click(Sender: TObject);
var
  buttonvalue:integer;
begin
  buttonvalue:=Application.MessageBox(' 请单击任一按钮 ',' 测试
  MessageBox 方法',MB_ABORTRETRYIGNORE);
  case buttonvalue of
    IDABORT:label1.Caption:='你选择了【终止】按钮';
    IDRETRY:label1.Caption:='你选择了【重试】按钮';
    IDIGNORE:label1.Caption:='你选择了【忽略】按钮';
  end;
end;

```

(8) 运行程序,可见单击“MessageBox”对话框中的按钮,就会在窗体中显示对应的按钮名称。


4.5 项目的编译和调试

在应用程序的界面设计和代码编制完成之后,接下来的工作就是对应用程序进行编译和运行,通过编译能够产生应用程序的可执行文件。如果程序中存在与设计思想不符或程序自身代码编制不当等方面的问题,则可通过调试查出问题的所在。

4.5.1 应用程序的编译

首先应该明确一点,项目在编译成目标文件(通常是可执行文件)后才能运行。在 Delphi 7 中,可以不用关心程序是怎样编译且可以直接选择运行程序,这是因为系统在运行前能够自动编译项目。但是,掌握一些 Delphi 的基本编译知识对于一个程序员来说是非常必要的。

在 Delphi 7 中,用户可以使用以下几种编译方式:

(1) 使用工具栏的  按钮或通过选择主菜单下的【Run】/【Run】命令:在这种方式下,Delphi 7 将重新编译自上次编译以后存在的改动源代码,并运行编译后的目标程序。这种方式是用户使用最多的方式。该命令的快捷键是 **F9**。


(2) 通过选择主菜单下的【Project】/【Syntax check】命令:使用本命令将对当前项目中所有的单元文件重新进行语法检查,不进行编译和链接。这种方式下花费的时间较少。

(3) 通过选择主菜单下的【Project】/【Compile】命令:使用本命令将对当前项目中自上次编译以后存在改动的源代码进行重新编译,并生成可执行文件(.exe)、动态链接库文件(dll)、资源文件(.res)等目标文件。

(4) 通过选择主菜单下的【Project】/【Build】命令:使用本命令将对当前项目中所有的源代码进行重新编译(不管上次编译后是否进行过改动),并生成可执行文件(.exe)、动态链接库文件(.dll)、资源文件(.res)等目标文件。当工程中包含的单元文件较多时,使用这个命令将需要花费较长的时间。

不管使用上面的哪种编译方式,如果用户的源程序中存在语法错误,代码编译器将显示错误所在的位置和原因等信息,并且停止可能要进行的链接操作。

如果我们在【例 4-1】的程序代码中任意添加一些内容,如在按钮单击事件的变量定义

部分加入“test;”，然后单击  按钮进行编译，就会出现如图 4-11 所示的编译错误信息。错误信息提示在单元文件“Unit1.pas”的第 30 行，没有发现需要的界定符。

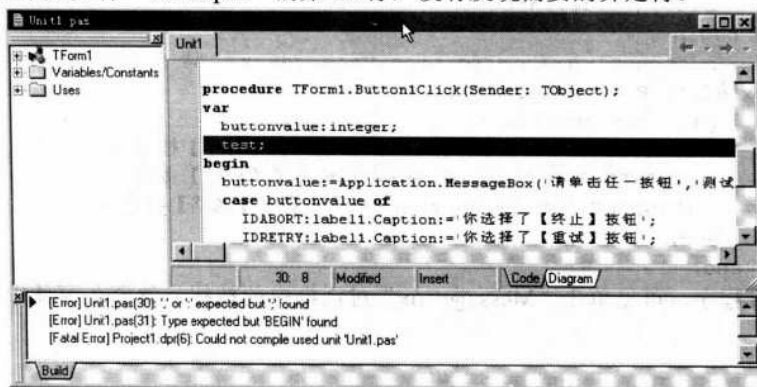


图 4-11 编译错误信息

鼠标双击出错信息提示，代码编辑器将打开该错误所在的单元文件并显示错误所在行。在默认情况下，Delphi 将不会产生编译和链接完毕的提示信息。如果要查看有关的编译和链接信息，可以通过下面的方法来完成。

【例 4-2】显示编译信息

- (1) 继续使用【例 4-1】的程序，将前面添加的“test;”删除；然后使用【Project】/【Compile All Projects】命令来编译程序，会发现没有任何编译信息出现。
- (2) 选择主菜单下的【Tools】/【Environment Options】命令，打开【Environment Options】对话框。
- (3) 选择【Preferences】选项卡，从【Compiling and running】栏中选中【Show compiler progress】复选框，如图 4-12 所示。

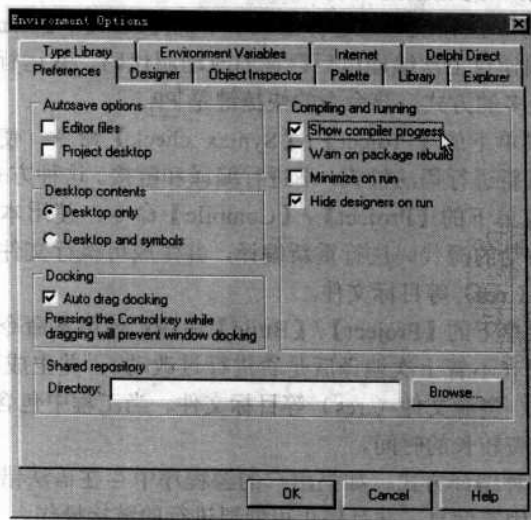


图 4-12 【Environment Options】对话框

(4) 单击  按钮, 关闭对话框。

(5) 再次使用【Project】/【Compile All Projects】命令来编译程序, 可以看到, 编译完成后, 会出现一个【Compiling】对话框, 说明一些基本的编译信息。如图 4-13 所示。

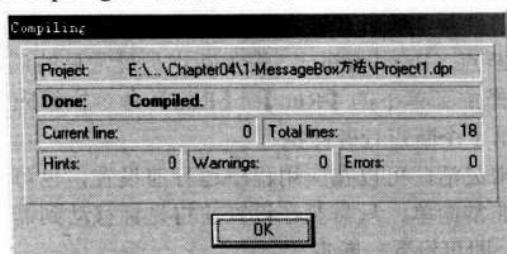


图 4-13 【Compiling】对话框

4.5.2 程序中常见错误

程序员在应用程序的开发过程中, 难免会遇到各种各样的错误。常见的错误可以分为以下几类。

- 语法错误: 是指在程序编制过程中, 代码的书写与编程语言的语法规则不符。例如没有定义的变量、函数或过程的参数不对、begin 和 end 不匹配等。这是程序编制中最常见的错误, 通常是因为程序员对编程语言不熟悉或者粗心而造成的。
- 逻辑错误: 是指程序的运行结果与程序员的设计思想不符。这种错误通常是编程人员在程序的结构设计上考虑不周, 而程序在语法上没有错误。例如程序中的死循环、测试条件设置错误等。
- 运行错误: 是指和程序的运行环境相关的错误。例如数据库相关的应用程序运行时 DBMS (数据库管理系统) 没有运行而无法连接到数据库、程序运行所需的附加的其他 DLL 没有运行等。

程序编制中遇到错误是很正常的事情, 遇到错误时不要着急, 要有耐心找到错误的所在。语法错误能够在编译时通过编译器检查出来, 而运行错误则需通过对程序的运行进行调试。Delphi 7 的集成开发环境也提供了很多与调试相关的命令和工具, 对用户迅速找出问题提供了很大帮助。

4.5.3 应用程序调试

一般来说, 应用程序的调试需要较多的专业知识。在 Delphi 7 中, 提供了强大实用的调试工具, 从而使常规的调试工作变得非常简单。

通过 Delphi7 进行调试来查找错误通常可以按以下步骤进行:

- 使程序的运行在出现错误的相关代码区域暂停。
- 使程序单步运行或执行到指定代码位置, 观察相关变量的数值变化情况。
- 根据程序段的设计目的判断在指定代码段中哪些变量值的变化有异常。
- 根据变量值的变化异常情况判断代码出错的地方。
- 修改错误代码并运行程序, 检查修改后的程序运行状况。

下面介绍调试中的一些细节。

1. 运行程序并在指定位置暂停程序的执行

在 Delphi 7 中, 用户可以通过选择主菜单下的【Run】/【Program Pause】命令直接暂停正在执行的应用程序, 这是无条件暂停程序执行, 在程序调试中运用并不普遍。一般我们需要使程序按照一定条件暂停执行, 这主要有以下两种方式。

(1) 在程序中设置断点 (Breakpoint): 设置断点是最常见的调试手段, 当在应用程序中设置断点后, 如果选择主菜单下的【Run】/【Run】命令执行应用程序, 则应用程序在程序执行过程中碰到的第 1 个断点处暂停执行。

设置断点的方式非常灵活, 可以在应用程序运行前设置, 也可以在调试应用程序时设置。断点的设置方法也非常简单, 只需要在设置前将光标移动到需要设置断点的代码行, 然后选择以下操作之一, 即可设置一断点。

- 通过主菜单下的【Run】/【Add Breakpoint】/【Source Breakpoint】命令, 打开【Add Source Breakpoint】对话框。选择需要的行号, 就可以在相应的行设置断点。

- 鼠标单击代码窗口左侧的行标识区域。

- 按下键盘上的 **F5** 键。

添加完断点后, 将在行标识区域显示一个红色圆点, 并将整行以红色显示, 如图 4-14 所示。

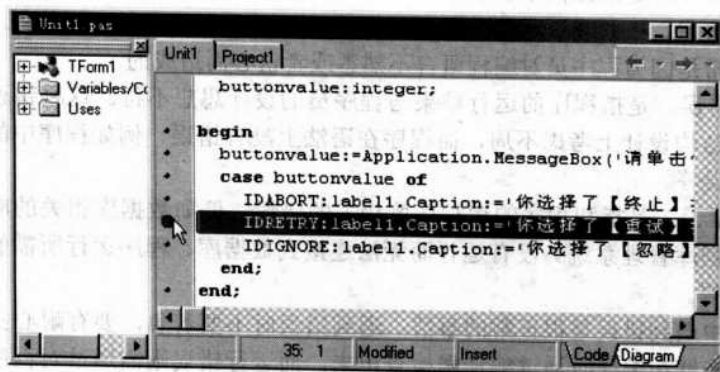


图 4-14 设置断点

删除断点的方法同添加断点的方法基本相同, 可以在光标移动到断点所在行后, 按下键盘上的 **F5** 键或用鼠标单击行标识区域的红色圆点。

(2) 通过主菜单下的【Run】/【Run to Cursor】命令: 除了使用断点外, 用户也可以通过选择主菜单下的【Run】/【Run to Cursor】命令 (或按下键盘上的 **F4** 键), 将应用程序运行并在光标所在位置暂停。使用这种方式以前, 用户需要事先将光标移动到需要暂停的代码行前面。

2. 调试运行程序

当程序在错误可能出现的代码行之前暂停运行后, 下一步需要做的是调试程序, 使其按照一定的方式运行。常见的调试运行方式有以下几种:

- 单步运行: 单步运行是指调试器每一次执行一条程序的代码, 如果运行到函数或者过程调用时, 则直接执行完整个函数或过程调用而不进入函数或过程的内部。选择主菜

单下的【Run】/【Step Over】命令，或按下键盘上的[F8]键执行单步运行操作。

- 跟踪运行：跟踪运行与单步运行基本上差不多，也是每一次执行一条程序的代码，不同的是，如果运行到函数或者过程调用时，调试器会进入函数或过程的内部执行。选择主菜单下的【Run】/【Trace Into】命令，或按下键盘上的[F7]键执行跟踪运行操作。

- 运行到函数或过程的结束处：这种方式是指调试器直接运行完当前的函数或过程，返回到调用该函数或过程的代码处。选择主菜单下的【Run】/【Run Until Return】命令，或按下键盘上的[Shift]+[F8]键执行操作。

在调试结束后，用户可以通过选择主菜单下的【Run】/【Program Reset】命令结束应用程序的执行，返回应用程序的设计状态。

3. 查看应用程序的运行状态

调试应用程序的最主要目的就是发现应用程序中存在的问题。而在调试期发现错误的主要手段就是通过检查应用程序运行过程中变量、表达式的数值是否与设计目标相符。

一般常用 Watch 窗口监测变量值的变化。可以通过以下方式之一将变量添加到 Watch 窗口：

- 在代码编辑器中单击鼠标右键需要添加到 Watch 窗口的变量，从弹出的快捷菜单中选择【Debug】/【Add Watch at Cursor】命令。
- 将光标移至需添加到 Watch 窗口的变量，按下键盘上的[Ctrl]+[F5]键。
- 将光标移至需添加到 Watch 窗口的变量，选择主菜单下的【Run】/【Add Watch】命令。

执行上述某一命令后，就会出现 Watch 窗口，并显示添加的变量。

4.5.4 “Code Insight”技术

为了方便程序员的编程工作，Delphi 7 在代码编辑器中采用了“Code Insight”技术来简化代码的输入，它主要包括如下几个方面：

1. Code compilation

在代码编辑器中，当用户在一个对象的名称后输入一个“.”以后，Delphi 将自动弹出一个提示窗口，用以显示与对象相关联的属性、方法和事件，如图 4-15 所示。

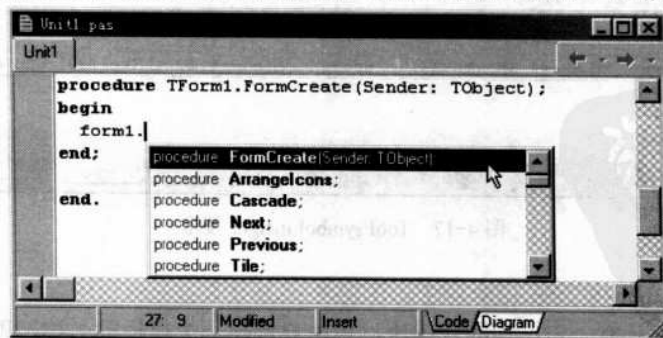


图 4-15 Code completion 技术

用户可以通过键盘的方向键和鼠标从提示窗口中选择需要的条目，然后按下空格键、

Enter 键或鼠标双击，将其添加到代码中。如果知道属性、方法或事件的第一个字母，按下相应的键，提示窗口就会罗列出以该字母打头的条目。

2. Code parameters

提供参数列表的动态提示功能。当用户输入一个函数或过程名称时，代码编辑器将自动产生一个参数提示窗口，显示该函数或过程需调用的各个参数类型和顺序的说明。当用户输入指定的参数时，该参数的说明信息也变为加粗的文字。如图 4-16 所示。

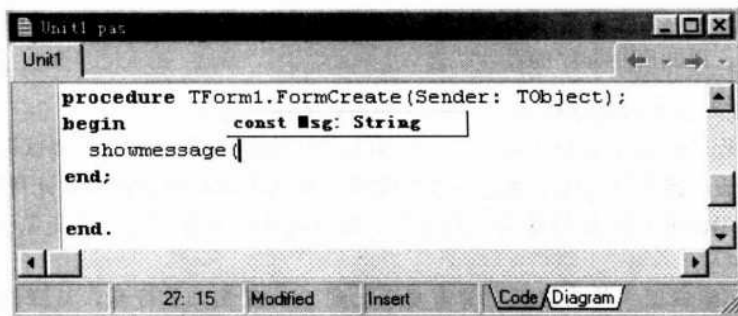


图 4-16 Code parameters 技术

3. Tooltip expression evaluation

程序调试运行时，如果将程序暂停，然后将光标移动到某个变量上，则代码编辑器能够显示一个信息框，说明该变量的当前值。

4. Tool symbol insight

在代码编辑器中，当用户将光标移动到一个标识符后，将会显示一个信息框，说明该标识符的信息。如图 4-17 所示。

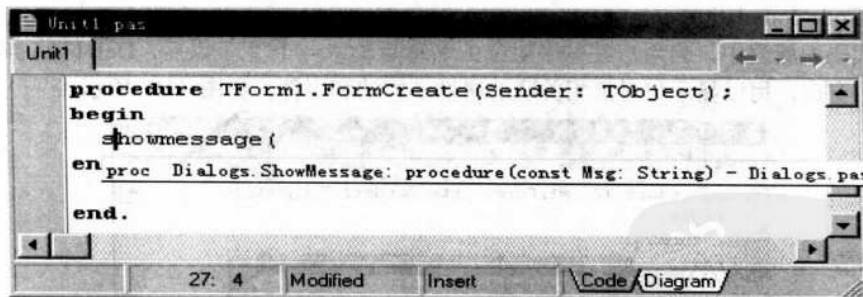


图 4-17 Tool symbol insight 技术

5. Code template

代码模板是指具有一定结构的代码块，当用户选择一个模板的名称，就可以将设定好的代码添加到代码编辑器中。在代码编辑器中，按下 **Ctrl+J** 键就可以打开代码模板的提示窗口，如图 4-18 所示。

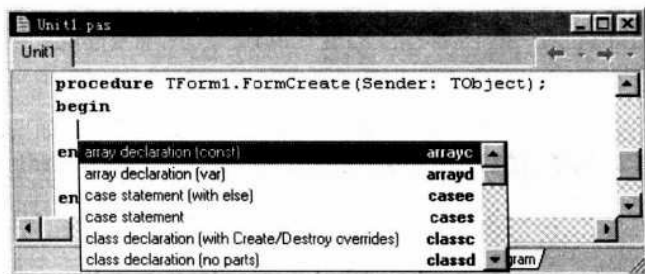


图 4-18 Code template

4.5.5 编译和调试示例

下面我们通过一个简单的实例说明如何对程序进行调试。

[例 4-3] 整数求和

在文本框中输入某一整数，然后单击窗体上的一个按钮，能够在标签对象上显示从 1 到该整数之间所有整数之和。程序画面如图 14-19 所示。



图 4-19 整数求和

- (1) 创建一个新的应用程序。适当缩小窗体的大小。
- (2) 在组件面板的【Standard】选项卡中，双击“Button”组件，将其添加到窗体中；利用【Object Inspector】面板设置按钮对象的【Caption】属性的内容为“求和”，设置字体大小为“12”。
- (3) 再在窗体中添加两个“Label”组件，设置“Label2”对象的【Caption】属性的内容为“请输入”。
- (4) 再添加一个“Edit”组件。布置各组件位置如图 14-16 所示。
- (5) 双击“求和”按钮，切换到代码编辑器窗口。输入如下所示程序代码，对整数进行求和。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,k:integer;
  sum:integer;
begin
  sum:=0; //初始化变量 sum
  k:=strtoint(edit1.Text); //将 Edit1 中的内容转化为数值，赋给变量 k
  for i := 1 to k do //循环语句，从 1 循环到 k
  begin
```

```

sum:=sum+i;           //将变量 sum 增加 i
label1.Caption:=inttostr(sum); //将变量 sum 的值显示到标签中
end;
end;

```

(6) 选择某一行，在行标识区单击鼠标，则该行被设置为断点，如图 4-20 所示。

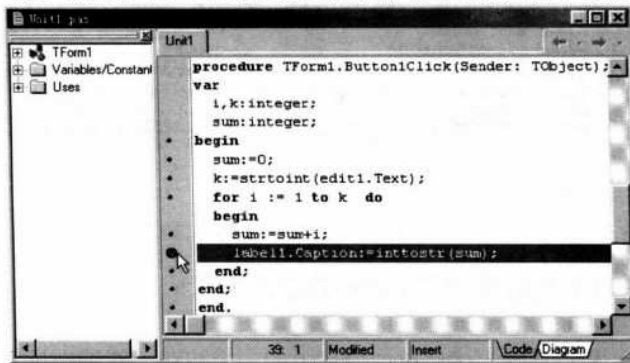


图 4-20 设置断点

(7) 将光标指向变量 sum，然后按下 **Ctrl+F5** 键，将 sum 变量添加到 Watch 窗口。

(8) 同样，将 i 变量也添加到 Watch 窗口，如图 4-21 所示。

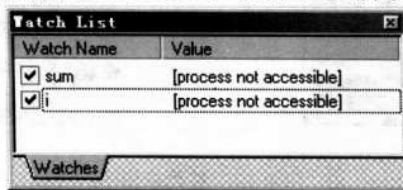


图 4-21 将变量添加到 Watch 窗口

(9) 为了方便程序的调试和观察，我们也可以拖动 Watch 窗口，将它加入到代码编辑器窗口，如图 4-22 所示。

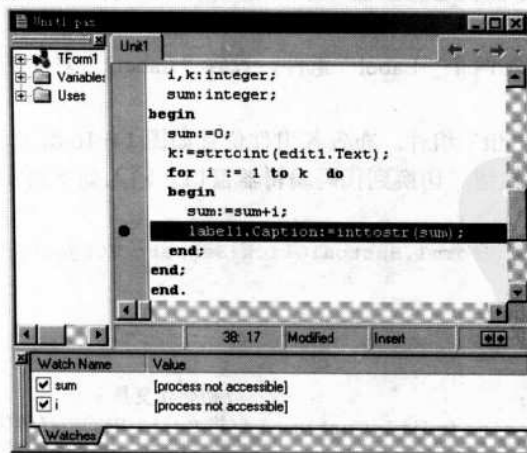



图 4-22 将 Watch 窗口加入到代码编辑器窗口

(10) 单击  按钮对程序进行编译和运行。在文本框中输入一个数值，如 100，然后单击“求和”按钮，则程序会自动停留在断点处，同时在 Watch 窗口会显示变量 sum 和 i 的当前值，如图 4-19 所示。

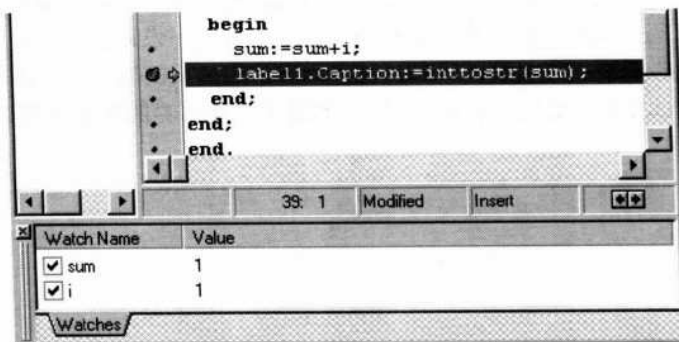


图 4-23 变量 sum 和 i 的当前值

(11) 按下 **F8** 键，单步调试程序，可见程序会执行到下一行代码，一个蓝色标记会显示当前执行的代码。

(12) 反复按下 **F8** 键，程序会反复执行求和代码；Watch 窗口中的变量值也会不断发生变化，如图 4-20 所示。

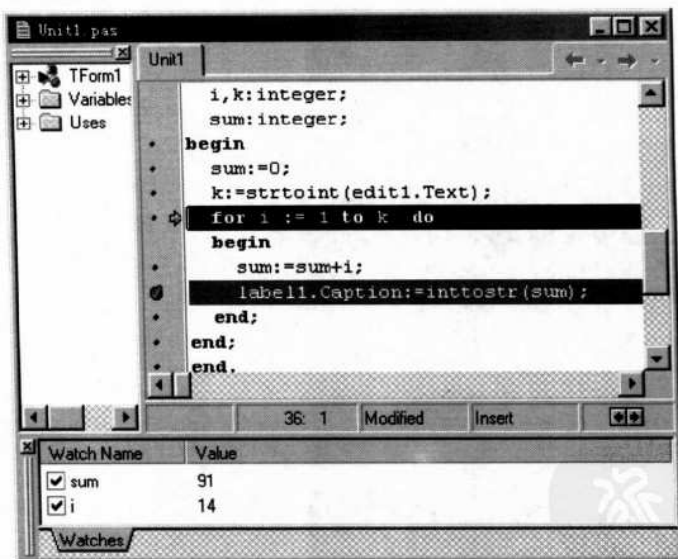



图 4-24 Watch 窗口中的变量值会不断发生变化

提示：将光标指向程序代码中的变量 sum 和 i，也会显示出变量当前的值。

(13) 如果不想使用断点，只需要在行标识区单击断点标志，就能够取消断点。再单击  按钮，就能够完整地运行程序了。

4.6 小结

Delphi7 的文件结构，特别是单元文件的内部结构，对于进行 Delphi 程序设计是至关重要的，只有理解了这些文件的关系和单元内部代码的组织关系，在编写程序时才能够知道该如何添加代码。同时，程序的编译和调试对于完成程序设计也是非常重要的。没有谁能够一丝不差地设计出一个复杂的项目，任何程序都需要反复地调试。所以，良好的编程习惯和正确的调试技巧，是每一个程序设计人员都应注重培养的基本素质。



第5章 基本窗体设计

虽然在前面的章节中我们已经接触过窗体，但是并没有详细讲述过窗体的概念。其实，窗体也是一个组件，有自己的属性、方法和事件，通过设计代码，我们可以有效地控制窗体的操作。窗体是进行 Delphi 程序设计的基础，所有的组件都要在窗体上进行布置和操作。另外，利用窗体的显示和隐藏，我们还可以轻松设计多文档窗体。

5.1 Form（窗体）组件

在 Delphi 中，窗体组件是一个特殊的组件，所有其他组件都要放到该组件中，它就像一个容器一样能够容纳别的组件。如同任何其他组件一样，窗体组件也有自己的属性、方法和事件。

5.1.1 窗体的属性

在 Delphi 7 中新建一个应用程序，系统都会自动为其创建一个窗体，并使用默认的标题“Form1”。利用【Object Inspector】可以看到窗体对象的各种属性，这些属性确定了窗体的外部形态和特征。

窗体的主要属性有：

- **【Align】**：设置窗体在屏幕上的排列位置，其中“alClient”选项是使窗体占整个屏幕，其余选项的含义比较好理解。
- **【Active】**：指示当前窗口是否为活动窗口，即具有输入焦点。
- **【AutoScroll】**：控制是否在内容不能完全显示时出现滚动条，以及滚动条出现的位置是水平或垂直。
- **【AlphaBlend】**：设置窗体是否可以透明显示，这是 Delphi 7 的新增功能，但是该功能只能在 Windows2000 以后的操作系统中体现。
- **【AlphaBlendValue】**：控制窗体的透明度，0 为完全透明，255 为完全不透明。
- **【BorderIcons】**：设定标题栏的系统按钮的样式，可以定义显示系统菜单、最大化按钮、最小化按钮和帮助按钮。
- **【BorderStyle】**：设置窗体边框和标题栏的显示风格。此属性的设置将影响【BorderIcons】属性的显示。
- **【FormStyle】**：设置窗体的风格。其中“fsMDIChild”/“fsMDIForm”选项定义了当前窗体是 MDI（多文档）窗体的子窗体还是父窗体。
- **【Visible】**：设置窗体是否可见。
- **【Icon】**：指定窗体的标题栏图标。

● **【WindowState】**: 设置窗体如何在屏幕上显示, 可以有正常、最小化、最大化几种方式。

窗体的属性可以在 **【Object Inspector】** 中直接设置, 也可以在运行时动态调整。下面用一个简单的实例来说明窗体属性的设置。

[例 5-1] 改变窗体的属性

效果: 使用按钮改变窗体的大小、色彩、标题等属性, 并可以利用滚动条来调整窗体的透明度属性。实例效果如图 5-1 所示。

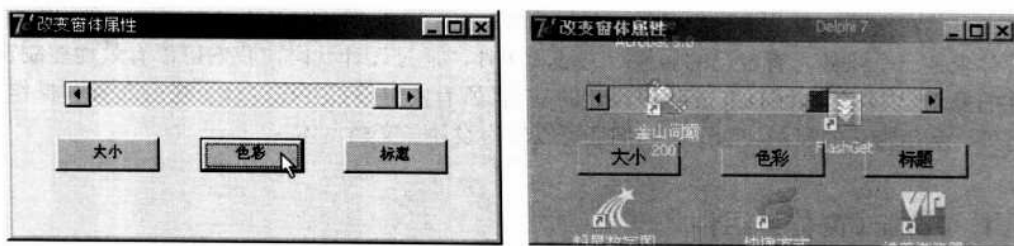


图 5-1 动态改变窗体的属性

- (1) 在 Delphi 7 中新建一个应用程序。适当调整窗体为较小的尺寸。
- (2) 向窗体中添加 3 个 Button 组件和一个 ScrollBar 组件。这两种组件都位于 **【Standard】** 组件选项卡中。
- (3) 在 **【Object Inspector】** 中修改各个按钮对象的文字 (**【Caption】** 属性)。
- (4) 双击第一个按钮对象“大小”, 则切换到其代码编辑器, 并且光标将自动定位到 OnClick 事件代码中。向其中添加如下斜体表示的代码, 调整窗体的大小。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    form1.Width := 500;
    form1.Height := 200;
end;
```

提示: 在输入“form1.”后, 系统会自动出现 form1 中所有的属性、事件和方法的列表, 这就是 Delphi 的 Code Insight 技术。只需要用选择需要的属性, 双击鼠标或按下空格键、回车键, 该属性就会自动进入到代码中。

- (5) 同理, 在第二个按钮对象“色彩”中, 添加如下斜体表示的代码, 设置窗体为黄色。

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    form1.Color := clYellow;
end;
```

- (6) 在第三个按钮对象 **【标题】** 中添加如下斜体表示的代码, 设置窗体的标题为“改变窗体属性”。

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    form1.Caption := '改变窗体属性';
```

end;

(7) 双击滚动条, 在代码编辑器中, 光标自动停留在 ScrollBar 的 OnChange 响应事件的代码中。向其中如下斜体表示的代码, 定义窗体的透明度随滚动条的位置值变化而变化。

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
  form1.AlphaBlend :=true;
  form1.AlphaBlendValue:=scrollbar1.Position;
end;
```

(8) 执行程序, 可见利用按钮可以改变窗体的大小、色彩和标题等属性; 调节滚动条, 可以改变窗体的透明度。前面添加的全部代码如图 5-2 所示。

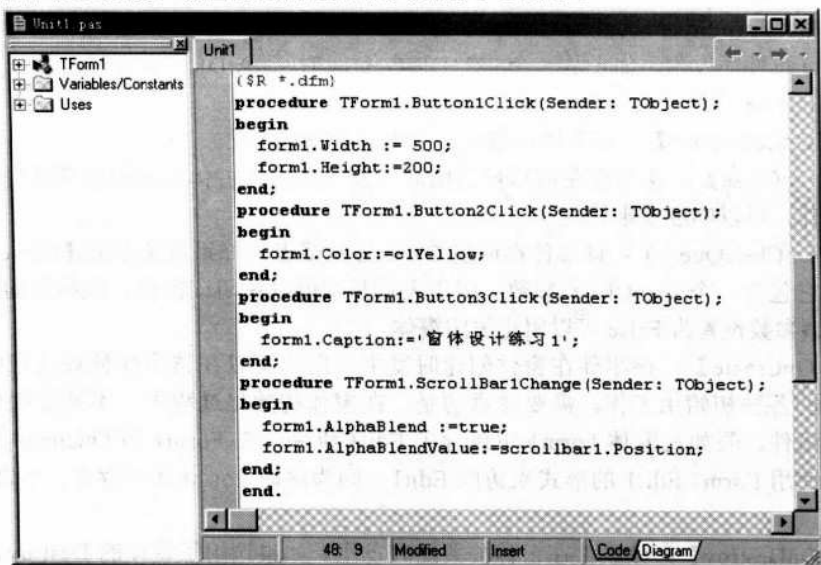


图 5-2 需要添加的全部代码

注意: 透明度的调整只能在 Windows2000 以上的系统中才能体现出来。在 Windows 98 下无法表现。

5.1.2 窗体的方法

方法是一个组件的动态属性, 标志了一个组件所具有的功能和操作, 通过调用窗体的方法可以实现对窗体的控制, 比如关闭窗体、隐藏窗体等。

窗体组件有许多方法, 这里仅列出一些最常用的方法。

- **【Close】**: 执行该方法将关闭窗体, 结束程序的运行。
- **【Hide】**: 该方法将窗体隐藏起来, 使其处于不可见状态。
- **【Release】**: 调用该方法将释放窗体以及相关组件占用的内存空间, 它和 Free 方法类似, 只是该方法要等到窗体的所有事件句柄或窗体中各组件的事件句柄全部执行完毕后, 才释放窗体。

● **【CloseQuery】**：该方法作为窗体的 Close 方法的一部分被调用，在 Close 方法中通过调用它来确定该窗体是否能够关闭。通常使用 OnCloseQuery 事件句柄询问用户是否允许用户关闭一个窗体，或者在窗体关闭之前保存窗体中的内容。

5.1.3 窗体的事件

所谓事件，不仅包括用户对程序中对象的操作，如鼠标单击、键盘按键等，还包括对象状态的某种改变，如对象激活或隐藏、打开或关闭等。为了能够处理这些事件并进行相应的操作，需要系统预先定义各种事件，然后由程序员确定如何处理这些事件。

常用的窗体事件主要有：

● **【OnActivate】**：该事件在应用程序窗体成为当前活动窗体时发生。对于 MDI 子窗体（窗体的 FormStyle 属性为 fsMDIChild），只有在各个子窗体之间切换时，窗体的 OnActivate 事件才发生，如果从一个非 MDI 子窗体切换到 MDI 子窗体时，只有 MDI 父窗体的 OnActivate 事件被激发。

● **【OnCanResize】**：该事件在窗体的大小被改变时触发。

● **【OnClose】**：该事件在窗体被关闭时触发。如果在窗体关闭时需要处理一些必须完成的工作，可以响应该事件。

● **【OnCloseQuery】**：该事件在调用 Close 方法或者选择系统菜单的**【Close】**菜单项时发生。它包含一个 CanClose 参数，用于表示是否可以关闭该窗体，该参数的默认值为 True，将该参数设置为 False 可以阻止关闭窗体。

● **【OnCreate】**：该事件在窗体创建时发生。用户可以在该事件处理过程中进行与窗体相关的各种初始化工作。需要注意的是，在窗体初始化过程中，不要使用全名引用窗体中的组件。假如在窗体 Form1 中包含了 Edit1 组件，在 Form1 的 Oncreate 事件句柄中，不能使用 Form1.Edit1 的形式来访问 Edit1，因为这时 Form1 还不存在，Edit1 还未被建立。

● **【OnDestroy】**：该事件在窗体被关闭时发生。如果调用了窗体的 Destroy、Free 或者 Release 方法，窗体及子窗体将被释放。

● **【OnPaint】**：该事件在 Windows 系统要求重新绘制窗体时发生。在该事件句柄中，通过在窗体的 Canvas 属性上绘制来显示窗体的客户区信息。

● **【OnResize】**：该事件在用户或者程序调整窗体的尺寸时发生。

● **【OnShow】**：当窗体被显示（如将 Visible 属性设置为 True）时发生。

● **【OnKeyDown】**：键盘按下时产生此事件。

● **【OnKeyPress】**：当窗体获得键盘输入焦点，且用户敲击键盘时，产生 OnKeyPress 事件。

● **【OnKeyUp】**：键盘放开时产生此事件。

● **【OnClick】**：鼠标单击事件。

● **【OnDblClick】**：鼠标双击事件。

● **【OnDrag|Drop】**：当鼠标在窗体中进行拖放时产生该事件。

● **【OnDragOver】**：当鼠标在窗体中拖过时产生该事件。

● **【OnMouseDown】**：鼠标按下时产生事件。

- **【OnMouseMove】**：鼠标在窗体中移动时产生该事件。
- **【OnMouseUp】**：鼠标被释放时产生该事件。

提示：当窗体建立后，并且 Visible 属性为 True 时，OnCreate、OnShow、OnActivate 和 OnPaint 事件将依次发生。

下面用一个简单的实例来说明窗体事件的使用法。

【例 5-2】检测鼠标和按键

效果：程序能够检测用户单击鼠标还是双击鼠标，以及按下了键盘上的哪一个按键，并且在标签对象上显示出来。效果如图 5-3 所示。

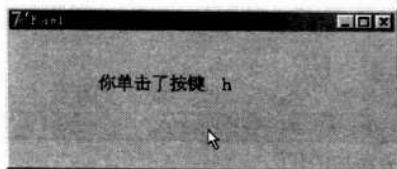


图 5-3 检测鼠标和按键

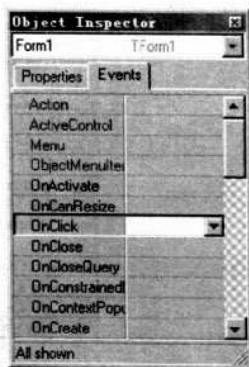


图 5-4 窗体的事件窗口

- (1) 在 Delphi 7 中新建一个应用程序。适当调整窗体为较小的尺寸。
- (2) 向窗体中添加 1 个 Label 组件。
- (3) 在 **【Object Inspector】** 中选择当前窗体，单击 **【Events】** 标签，切换到事件窗口，如图 5-4 所示。
- (4) 在 **【OnClick】** 事件后面的输入栏双击鼠标，则出现代码编辑器，并自动定位于窗体的 OnClick 事件代码中。向其中添加如下斜体表示的代码，使标签对象显示当前鼠标的单击事件。

```
procedure TForm1.FormClick(Sender: TObject);
begin
    label1.Caption := '你 单击 了鼠标...';
end;
```

提示：在添加代码后，可见 **【Object Inspector】** 中，**【OnClick】** 事件后出现了过程的名 称。

- (5) 同理，在 **【OnDblClick】** 事件中添加如下斜体表示的代码，使标签对象显示当前鼠标的双击事件。

```
procedure TForm1.FormDblClick(Sender: TObject);
begin
    label1.Caption := '你 双击 了鼠标...';
end;
```

(6) 在【OnKeyPress】事件中添加如下斜体表示的代码，使标签对象显示当前按下的按键名称。

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
    label1.Caption := '你单击了按键 ' + Key;
end;
```

(7) 执行程序，当对窗体进行操作或按下按键时，就会出现相应的提示信息。

(8) 前面步骤中添加的全部代码如图 5-5 所示。

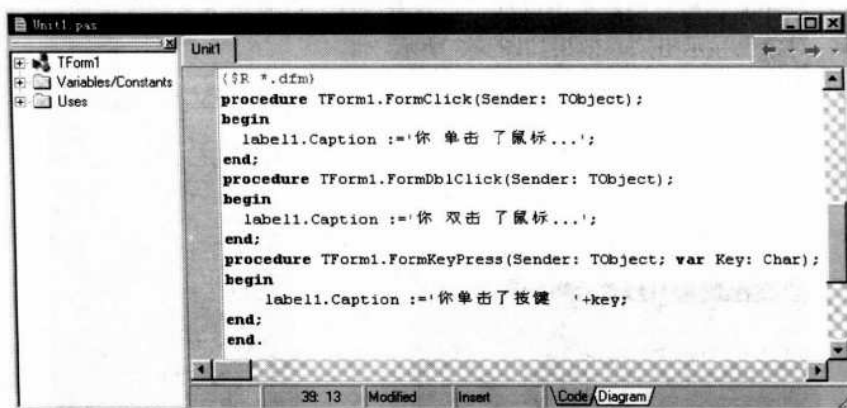


图 5-5 需要添加的代码

5.2 向窗体中添加组件

Delphi 的组件就像一块块的积木，经过适当的组合后，就能够完成特定的功能。可视化编程的基本方法就是添加组件，然后对组件对象进行编程。

虽然在前面的章节中已经练习过向窗体中添加组件，但是这里我们还要通过一些比较细致的讲解来说明如何对组件进行调整和编辑。

5.2.1 向窗体中添加组件

向窗体中添加组件一般有以下几种方法：

- 在组件面板中单击需要添加的组件，使其处于选择状态（凹陷下去），然后用鼠标在窗体上单击。

- 在组件面板上双击需要添加的组件，则组件自动放置在当前窗体的中央位置。

- 选择主菜单下的【View】/【Component List】命令，会出现如图 5-6 所示的【Components】对话框，选择指定的组件对象后，单击 **Add to form** 按钮，将其添加到当前窗体。


组件添加完毕后，单击组件面板最左侧的  按钮，则脱离组件添加状态，进入鼠标操作状态。



图 5-6 【Components】对话框

5.2.2 调整组件对象布局

1. 组件对象的选择

组件对象的选择一般有以下方式：

- 鼠标单击组件对象，这是最常用的选取方式。
- 通过【Object Inspector】的下拉选择框选择组件对象，如图 5-7a 所示。这种方式对于一些不容易被选中（如太小或被遮挡）的组件对象比较有效。
- 利用对象属性查看器来选择，如图 5-7b 所示。

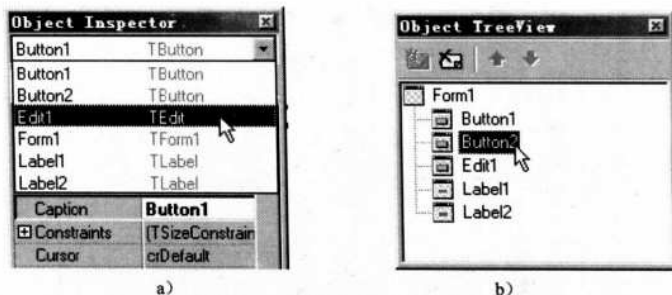


图 5-7 选择组件对象的方式

2. 对象的对齐

当窗体中有多个组件对象时，为了调整对象的布局，可能需要将对象对齐。一般有以下几种方式来调整对象的位置：

(1) 直接拖曳对象：这是一种最常用的对象位置调整方式。选择对象后，用鼠标拖曳对象到合适的位置就可以了。

(2) 使用对齐工具箱：如果来实现多个对象的精确对齐，使用鼠标直接调整就不方便了。选择多个对象，然后选择【View】/【Alignment Palette】菜单命令，会出现如图 5-8 所示的对齐工具箱；利用其中的工具可以方便地实现多个对象的精确对齐。

(3) 使用对齐对话框：对齐对话框同对齐工具箱的功能基本相同，使用起来也比较方便。在窗体上选择了需要调整的组件对象后，选择主菜单的【Edit】/【Align】命令，将弹出一个【Alignment】对话框，如图 5-9 所示。利用该对话框就能够方便地调整组件对象的相互位置。

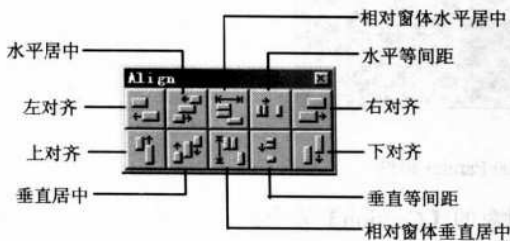


图 5-8 对齐工具箱

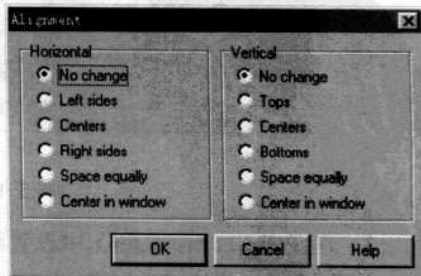


图 5-9 对齐对话框

5.2.3 添加组件对象实例

下面我们用一个简单的实例来说明如何在窗体中添加组件对象。虽然我们尚不清楚这些组件的用法，但这里练习的目的是添加组件和调整布局，所以大家只需要跟随步骤进行练习就可以了。

【例 5-3】添加组件对象

在窗体中任意添加一些组件，并调整各个组件对象的大小和位置。程序的画面如图 5-10 所示。

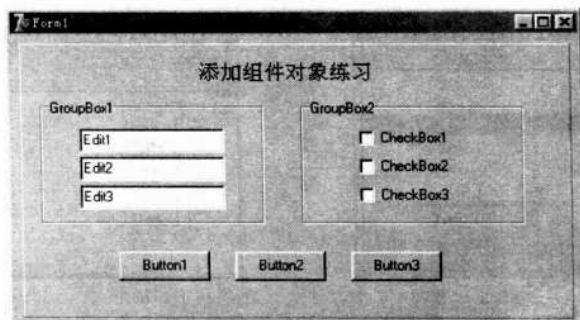



图 5-10 添加组件对象练习

(1) 创建一个新的应用程序，适当缩小窗体的大小。

(2) 从【Standard】选项卡中，选择  (Pannel) 组件（倒数第 2 个组件）；然后在窗体上单击鼠标，就可以添加一个组件对象，如图 5-11 所示。

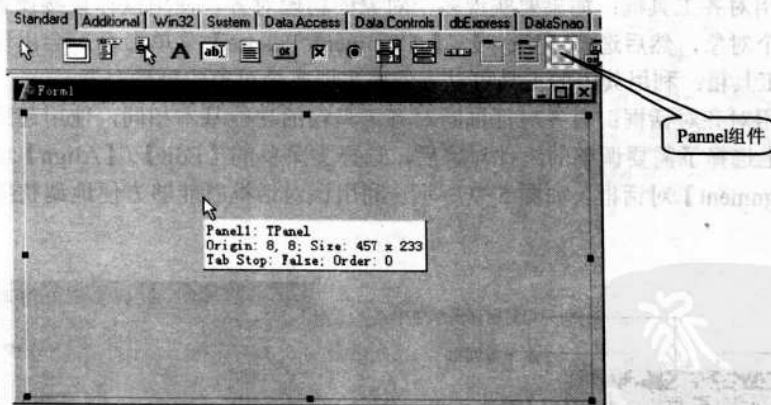



图 5-11 添加 Pannel 组件

(3) 在【Object Inspector】中设置组件对象的【Caption】为空。

(4) 继续添加其他组件对象，如图 5-12 所示，其中  (GroupBox) 组件位于【Standard】选项卡的倒数第 4 个位置。

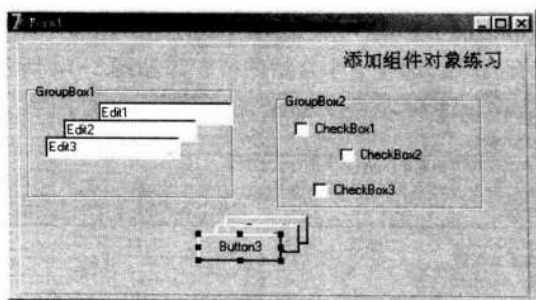


图 5-12 添加其他组件对象

(5) 选择各个组件对象，利用鼠标拖曳或对齐工具箱调整对象位置，使之符合程序画面要求。

5.3 单文档窗体

单文档窗体 (SDI, Single Document Interface) 是文档处理软件经常使用的方式。Windows 的“记事本”工具，只能打开一个窗口对文档进行编辑，这就是典型的单文档窗体。

单文档程序界面设计比较简单。每次新建的应用程序，如果不更改【Formstyle】属性，默认的都是单文档程序。

下面来利用新文件向导来制作单文档窗体程序。

【例 5-4】单文档窗体程序

(1) 在 Delphi 7 中，单击【File】/【New】/【Others】菜单命令，出现新项目对话框。单击“Projects”标签，打开【Projects】选项页，如图 5-13 所示。

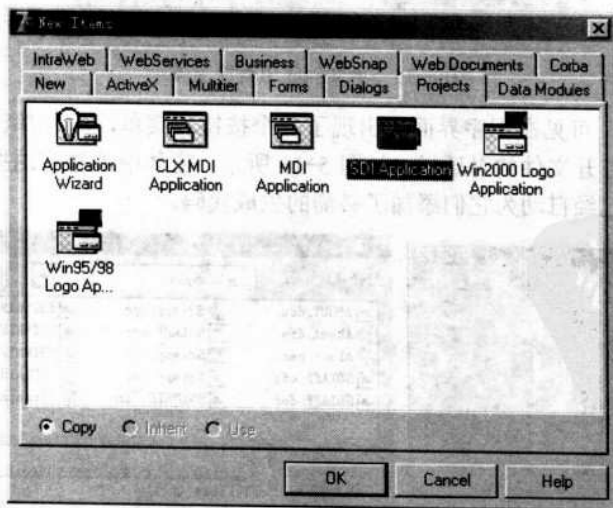


图 5-13 新项目对话框

(2) 选择【SDI Application】项，单击 按钮，出现一个【Select Directory】对话框，要求选择文件保存目录。选择一个合适的位置，如图 5-14 所示。



图 5-14 选择文件保存位置

(3) 确定后，出现一个单文档应用程序窗体，如图 5-15 所示。其中包含了单文档窗体所必须的各种元素，这些都是由系统自动生成的。

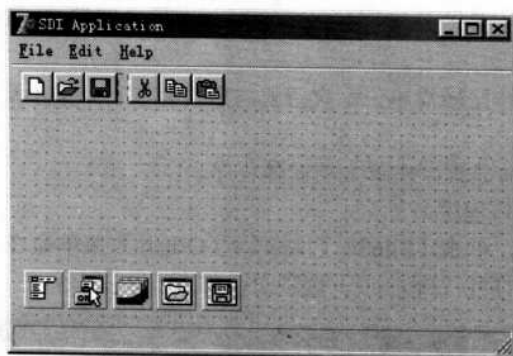


图 5-15 单文档应用程序窗体

(4) 执行程序，可见在程序界面上出现了几个按钮和菜单，而且都已经具备了基本的功能，如能够显示打开文件的对话框，如图 5-16 所示。窗体中能够有这些按钮、菜单和对话框，是因为系统已经自动为它们添加了必需的生成代码。

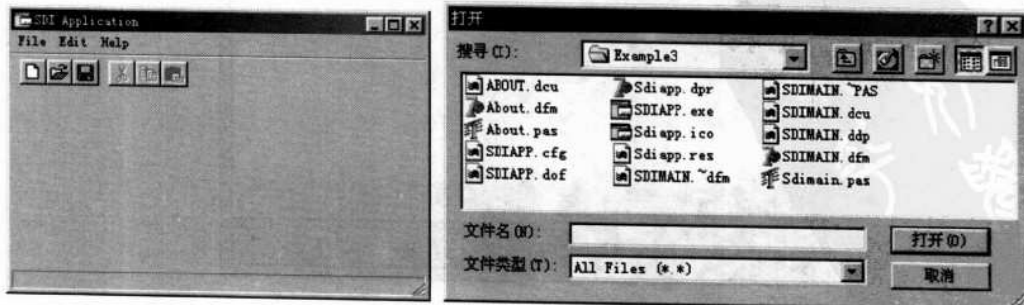


图 5-16 应用程序运行界面

(5) 打开代码编辑器, 可以看到其中已经有许多程序代码, 如图 5-17 所示。但是系统并没有建立处理具体操作的代码, 这需要用户自己完成。在后面的章节中, 我们将学习如何处理用户的操作。

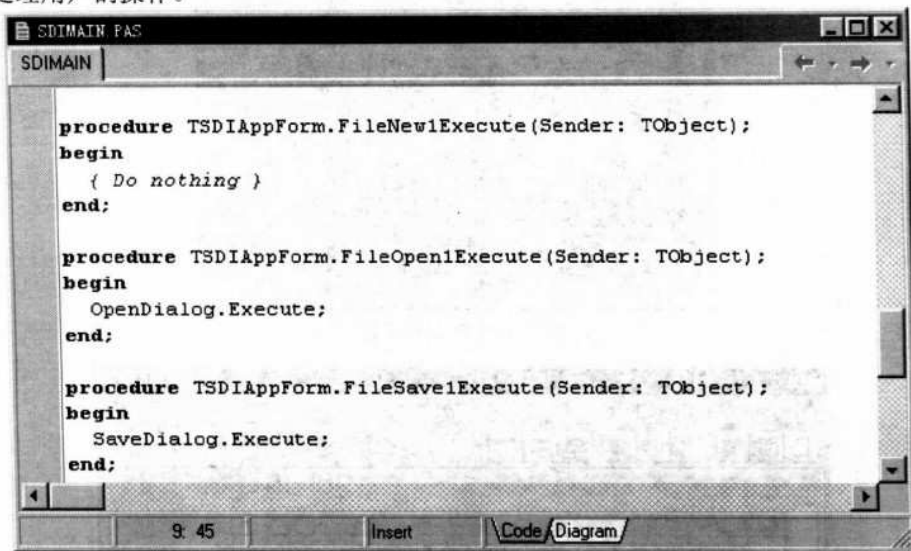


图 5-17 系统自动生成的代码

5.4 多文档窗体

相对于单文档窗体, 多文档窗体 (MDI, Multiple Document Interface) 的功能更为强大和复杂一些。如 Microsoft Office 的 Word 软件, 能够同时打开多个窗口编辑文档, 这就是典型的多文档窗体。

Form (窗体) 对象的【Formstyle】(窗体类型) 属性决定了窗体是单文档还是多文档。该属性有 4 个选项:

- FsNormal: 普通窗口, 即单文档窗口。这种窗体既不是父窗体, 也不是子窗体, 和其他窗体的地位是平等的。
- FsMDIChild: 设置为多文档应用程序的子窗体, 供父窗体调用。
- FsMDIForm: 设置为多文档应用程序的父窗体, 调用子窗体。
- FsStayOnTop: 设置窗体始终处于应用程序或其他窗口的最上面。

[例 5-5] 利用模板建立多文档窗体

(1) 同单文档窗体设计方法相似, 在【New Items】对话框的【Projects】选项页中, 选择“MDI Application”项, 确定并保存文件后, 会出现如图 5-18 所示的多文档应用程序窗体。


(2) 执行程序, 可见在程序界面上出现了更多的按钮和菜单, 而且都已经具备了基本的功能。单击  按钮, 能够打开多个空白窗体, 并可以在窗体中任意输入内容, 如图 5-19 所示。



图 5-18 多文档窗体

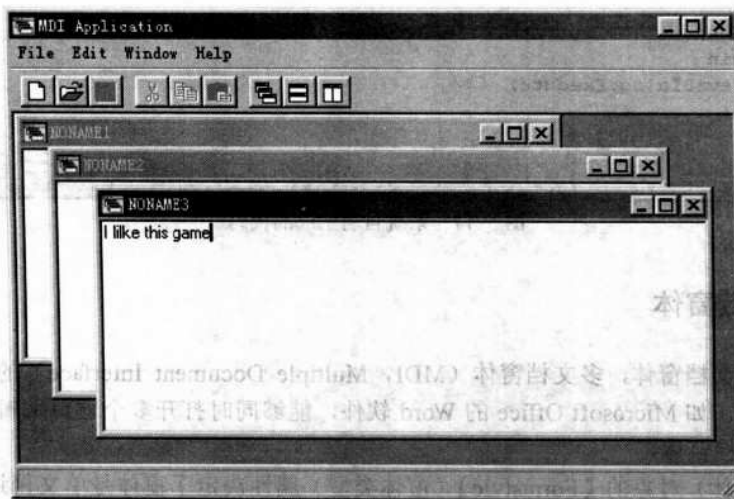


图 5-19 创建多个空白窗体

5.5 小结

窗体是 Delphi 程序设计的基础，是装载其他组件的容器。本章我们简单说明了窗体的属性、事件和方法，并学习了如何向窗体中添加组件、如何利用模板创建单文档和多文档窗体。这些知识比较简单，但却是我们后面学习使用其他组件、编写数据库应用程序的必备基础。

第6章 基本组件的应用

在面向对象的程序设计方法中，组件技术占据了重要的地位。Delphi 是一个完全集成的可视化编程环境，其最大的优点就在于提供了丰富的组件。用户通过拖放就能够设计出功能完善的应用程序。本章将集中讨论 Delphi 中部分常用组件的使用方法，其中包括文本输入组件、按钮组件、列表组件、信息反馈组件、表格显示组件等。

6.1 组件的概念

Delphi 的组件分为可视组件和非可视两类。可视组件在程序运行时是可以看见的，主要用于与用户进行交互，即界面的处理，如按钮、标签和文本框等。相对而言，非可视组件在运行时是不可见的，主要用于后台处理，如许多与数据库相关的组件、Timer（定时器）组件等。

组件是真正面向对象意义上的对象。Delphi 的组件封装了一些数据集和数据访问的过程与函数，从祖先类中继承了数据和行为。尽管每个组件有其特殊性，所有组件都共享从它们的共同祖先 Tcomponent 继承来的某些属性。Tcomponent 定义了组件用于 Delphi 环境所必需的最小属性集。

组件可以看作是由 3 个基本部分组成：属性（状态信息）、方法（动作信息）、事件（反馈信息），它是以一种标准的可重用的方式封装了应用元素。

6.1.1 Delphi 组件的属性（Properties）

所有组件都有其内置的属性、方法和事件，其中一些是从祖先类中继承而来。这意味着和其他组件共享这些元素，这些元素称为公共元素。例如，所有组件都继承了 Height 属性，它表示组件的垂直大小。因此，Height 是所有组件的公共属性。

每种组件除公共属性外，还引入了自己特有的属性，称为关键属性。例如，复选框组件有一个“Checked”属性，表示该复选框的状态（是否选中），该属性是复选框所特有的。所以，大家在学习组件时，在理解公共属性的基础上，要这种掌握每个组件的关键属性。

下面介绍一般组件所具有的公共属性：

1. 【Name】属性

【Name】是所有组件的公共属性。Name 属性是组件对象的标识名，所以它必须符合 Object Pascal 标识符的规定。在一个单元中由【Name】属性给出的对象标识名必须唯一，可以采用系统默认名称，也可以在一开始就取一个有意义的名字，这不仅增加代码的可读性，而且也避免以后潜在的名称冲突。值得注意的是，你只能在对象监视器中修改【Name】属性。一旦改变了【Name】属性，Delphi 会在自动生成的代码里做相应的改动。

2. 位置和大小属性

有 4 个属性定义了组件在窗体上的位置与大小，它们是：

- **【Height】**：组件垂直尺寸（尺寸单位是屏幕的像素值）。
- **【Width】**：组件水平尺寸。
- **【Left】**：组件的水平坐标（以窗体的左边缘为准，单位是像素）。
- **【Top】**：组件的垂直坐标（以窗体的上边缘为准，单位是像素）。

3. 显示属性

以下 4 个属性决定了组件的大致外观：

- **【BorderStyle】**：指定组件是否有边框。
- **【Color】**：定义组件的颜色。
- **【Ctrl3D】**：指定组件是否具有三维外观效果。
- **【Caption】**：在窗体上为用户识别组件对象而标注的名字（字符串）。
- **【Font】**：设置组件上的文字特性。

4. 父属性

为了在应用中保持一致的外观，可以让组件根据其容器决定显示属性。所谓容器就是可以包含其它组件的组件，窗体就是所有摆放在它上面的组件的容器。对象属性中，Parent 一词就是指包含本对象的直接容器。一般组件有几个父属性，如 ParentColor、ParentCtrl3D、ParentFont 等，如果将它们设置成 True，则组件的颜色、字体等就取容器组件的相应属性值。当然你也可以为对象设置自己的外观属性值，而此时，父属性值会变为 False。

5. 导航属性

所谓“导航”就是在不同的对象之间进行切换，使得被操作的对象获得焦点。

与导航有关的属性有下面几个：

- **【Caption】**：在 Caption 的字符串中，如果在一个字符的前面插入一个“&”字符，在程序运行时，该字符下面会出现下划线，这类字符称之为加速字符。用户可以按住 Alt 键并输入加速字符，来选择组件或菜单项。

- **【TabOrder】**：组件在其容器里的 Tab 顺序号，该顺序号为 0、1、2...等。这就是按 Tab 键时光标停靠的顺序。

- **【TabStop】**：该属性是一个逻辑值，它可以控制用户能否用 Tab 键跳到一个组件。若属性值为 True，则该组件在 Tab 顺序中。

6.1.2 组件的事件 (Events)

当发生一特定事件（如鼠标单击、鼠标移动、组件创建等）时，应用程序就会执行相应的事件处理代码。通过【Object Inspector】的【Events】选项卡，能够为组件对象创建对多个事件的响应代码，如图 6-1 所示。

组件之间有许多事件是相同或相似的，其中常用事件如表 6-1 所示。

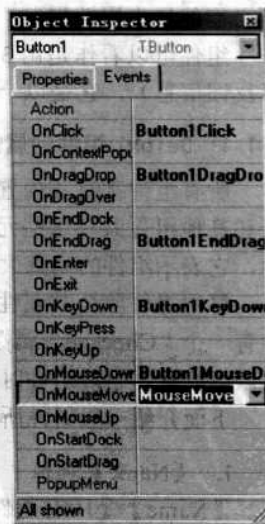


图 6-1 【Object Inspector】的【Events】选项卡

表 6-1 组件常用事件

事件类型	事件名称	事件说明
与鼠标操作相关的事件	OnClick	鼠标单击
	OnDblClick	鼠标双击
	OnMouseDown	鼠标左键或右键按下
	OnMouseMove	鼠标移动
	OnMouseUp	鼠标左键或右键松开
与键盘操作相关的事件	OnKeyDown	敲击键盘上的按键
	OnKeyPress	按下键盘上的按键
	OnKeyUp	松开已按下的按键
与拖曳操作相关的事件	OnStartDrag	开始拖曳对象
	OnDragOver	拖曳对象经过对象
	OnDragDrop	结束拖曳对象
	OnStartDock	开始进行停靠方式的拖曳
	OnEndDock	结束停靠方式的拖曳
焦点相关事件	OnEnter	获得焦点
	OnExit	失去焦点

6.1.3 组件的方法 (Methods)

方法是属于一个给定对象的过程和函数，方法反映的是对象的行为（某种处理机制）而不是数据。组件的方法反映了组件所具有的功能，通过调用组件的方法，可以使组件完成一定的功能。一般组件都提供了许多方法用于组件操控和通信。

6.1.4 Delphi 7 的组件库

Delphi 7 的可视化组件库 (VCL) 提供了丰富的组件，被分门别类地布置在 27 个选项卡中。由于某些组件可能很少用到，所以系统提供了自定义组件面板的机制让设计者根据需要定制组件面板。选择系统菜单中的【Component】/【Configure Palette】命令，就能够打开面板属性对话框，如图 6-2 所示。利用它就能够定制组件面板。



图 6-2 利用面板属性对话框定制组件面板

对于常用组件，按照组件功能可以分为几种基本类型，如表 6-2 所示。

表 6-2 常用组件的基本类型

组件类型	常用组件
文本输入类	Edit、LabelEdit、MaskEdit、Memo、RichEdit
选项按钮类	CheckBox、RadioButton、GroupBox、RadioGroup
列表类	Listbox、ComboBox、CheckListBox、ListView、TreeView
范围类	ScrollBar、TrackBar、ProgressBar、UpDown、PageScroller
按钮类	Button、BitBtn、SpeedButton
工具栏类	ControlBar、ToolBar、CoolBar
菜单类	MainMenu、PopupMenu
图形图像类	Bevel、Shape、Image、ImageList、PaintBox 等
系统类	HotKey、DateTimePicker、MonthCalendar、Timer、MediaPlayer 等
对话框类	OpenDialog、SaveDialog、FontDialog、ColorDialog、PrintDialog、FindDialog 等

Delphi 7 的联机帮助系统对于组件的使用是很有帮助的。单击某个选项卡，然后按 **F1** 键，就能够获得该选项卡中所有组件的简单描述；若选择某一组件，然后按 **F1** 键，就能够获得该组件的帮助信息。

下面我们通过实例，说明一些常用组件的具体用法。

6.2 文本输入组件

在应用程序的使用中，经常需要用户输入一些信息，或者以一定的格式将文本显示出来，这就需要使用 Label 类组件或 Edit 类组件。二者的区别主要在于后者既能够实现显示，也能够接受输入。下面我们介绍几个常用的 Edit 类组件。

6.2.1 Edit 组件

Edit 组件用来响应用户的输入，可以向其中输入信息。输入的内容以字符串的形式保存。

1. Edit 组件的主要属性

- **【Text】**：保存文本框中的字符串。
- **【Enabled】**：控制组件是否响应键盘、鼠标等操作。
- **【PasswordChar】**：控制文本框中是否显示用户输入的字符。默认值为 #0，标识显示字符。如果用户用一个非零字符代替，如“*”，则用户在文本框中输入的字符都是以该字符显示，隐藏实际输入的字符。这样可以起到对输入信息的保护作用。

- **【AutoSelected】**：设定当程序启动时，组件中的字符是否处于选中状态。当属性值为 True 时，字符处于选择状态，如图 6-3 所示。



图 6-3 组件中的字符自动处于选中状态

2. Edit 组件的主要事件

- **【OnKeyPress】**: 当键盘上的键按下时, 事件被触发。
- **【OnChange】**: 当组件中的内容改变时, 触发该事件。

3. Edit 组件的主要方法

- **【Clear】**: 清除文本框中的文字内容。
- **【CopyToClipboard】**: 将文本框中选中的内容复制到系统剪贴板中。
- **【CutToClipboard】**: 将文本框中选中的内容剪切到剪贴板中。
- **【SelectAll】**: 选中文本框中所有的内容。

下面我们用一个简单的示例说明 Edit 组件的使用。

[例 6-1] Edit 组件的使用

画面上有两个 Edit 组件, 在上面的文本框中输入数字, 下面的文本框中会自动与上面文本框中的内容相同; 若按下非数字的键, 就会出现错误提示。程序效果如图 6-4 所示。

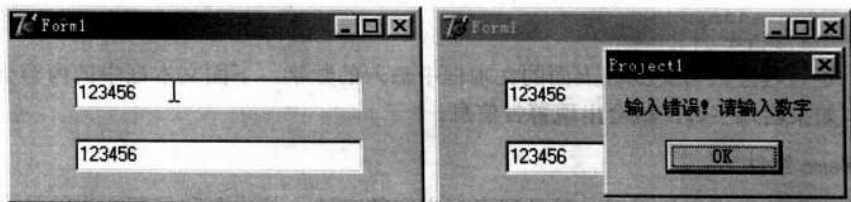


图 6-4 Edit 组件的使用

(1) 在窗体上创建两个 Edit 组件对象, 其中上面的 Edit 组件名称为“Edit1”, 下面的组件名称为“Edit2”。

(2) 选择“Edit1”组件, 在**【Object Inspector】**中, 打开**【Events】**选项卡, 找到**【OnChange】**事件, 如图 6-5 所示。

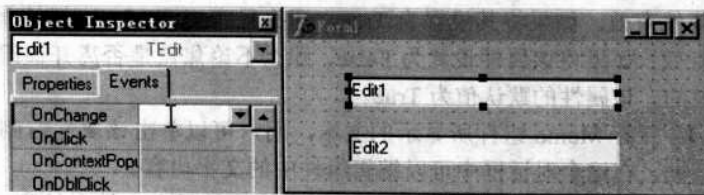


图 6-5 【OnChange】事件

(3) 在事件的输入栏中双击鼠标, 会打开代码编辑器, 并自动定位于 Edit1Change 过程中。输入如图 6-6 所示代码, 定义当“Edit1”中的内容发生变化时, “Edit2”中的内容就等于“Edit1”的内容。

(4) 同样, 在**【OnKeyPress】**事件中双击鼠标, 打开代码编辑器, 在 Edit1KeyPress 过程中输入如下代码, 对用户输入进行限制和警告。

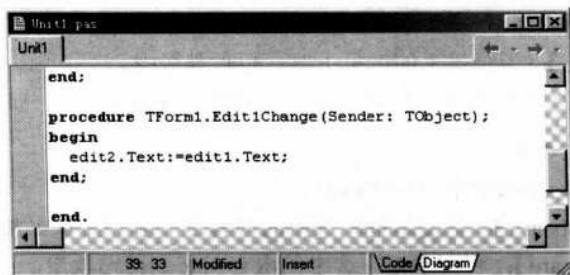


图 6-6 当“Edit1”中的内容发生变化时，“Edit2”中的内容就等于 Edit1 的内容

```

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if not(key in ['0'..'9']) then           //如果输入的字符不在 0-9 之间
    begin
        key:=chr(0);                        //将输入的字符置空，相当于没有输入
        showmessage('输入错误！请输入数字'); //显示提示信息
    end;
end;


```

(5) 运行程序，可见到在上面的文本框中输入的数字，下面文本框中的内容会随之发生变化；如果输入字母，就会出现警告信息。

6.2.2 Memo 组件

Memo 组件为用户提供了一种处理多行文本的方法，其读取和保存文本的最大长度可达 255KB。

1. Memo 组件的属性

- **【BorderStyle】**: 决定组件是单线边界还是无边界。
- **【Enabled】**: 决定组件中的内容是否可以修改。默认为 True，表示可以修改。
- **【SelLength】**: 返回在 Memo 框中被选择的文本的长度。
- **【HideSelection】**: 决定当输入焦点从 Memo 组件转到另一个组件时，在组件中被选中的文本是否依然有效。如果将其设置为 True，那么只有当该 Memo 组件重新为焦点时，选择的文本才有效；如果将该属性设置为 False，那么不论焦点是否离开该组件，被选择的文本一直是有效的。该属性的默认值为 True。
- **【Lines】**: 保存 Memo 组件所显示的文本，用户可以单击该属性右侧的  按钮，将会弹出一个对话框，在这个对话框中可以编辑组件中的文字内容。
- **【MaxLength】**: 指定用户可向该 Memo 框中输入的最大字符串长度。默认值为 0，标识向 Memo 组件中输入的字符数不受限制。
- **【Modified】**: 指出 Memo 框中的内容自建立以来或者 Modified 属性最后一次被设置为 False 之后是否被修改。
- **【ReadOnly】**: 决定是否可以对 Memo 框中的内容进行修改。默认值为 False，表示用户可以修改。
- **【ScrollBars】**: 控制 Memo 组件是否有滚动条。它的默认值是没有滚动条。另外还

可以设置为仅有水平滚动条、仅有竖直滚动条、既有水平又有竖直滚动条。

- **【SelStart】**: 确定在 Memo 框中被选择的文本的开始部分。
- **【SelText】**: 指出的字符串就是要从 Memo 框中选择的文字。
- **【WantReturns】**: 决定用户向 Memo 组件中输入文本时是否产生回车符号(换行符)。

如果将它设置为 True, 那么在 Memo 组件输入数据的过程中, 用户按了 **Enter** 键之后, 在文本中将产生一个换行符号。

2. Memo 组件的方法

- **【Canfocus】**: 用来测试 Memo 组件是否可以接收焦点。如果其值为 False, 则组件不能接收焦点, 反之可以接收焦点。
- **【Clear】**: 删除在 Memo 组件中的所有文字。
- **【ClearSelection】**: 删除 Memo 框中被选中的文本。如果在 Memo 组件中没有被选择的内容, 那么该方法将不删除任何东西。
- **【CopyToClipboard】**: 该方法将 Memo 框中被选择的内容复制到剪切板中。
- **【CutToClipboard】**: 该方法将 Memo 框中被选择的内容复制到剪切板中, 并且将 Memo 组件中被选择的内容删除。
- **【Focused】**: 该方法用来确定 Memo 组件有无焦点, 如果有焦点说明它是窗体上的 **ActiveControl** 组件。
- **【PasteFromClipboard】**: 该方法将剪切板上的内容粘贴到 Memo 框中, 并插入到 Memo 框中光标当前所在的位置。
- **【SelectAll】**: 该方法将 Memo 框中的所有文本都选中。
- **【SetFocus】**: 该方法将 Memo 组件设置为焦点。

3. Memo 组件的事件

- **【OnChange】**: 该事件指定 Memo 组件的内容被改变时所需的过程。当 Memo 框中的数据被修改时, 该事件发生。
 - **【OnExit】**: 当输入的焦点从 Memo 组件移动到另一个组件时, 该事件发生。
 - **【OnKeyDown】**: 当 Memo 组件成为当前焦点后, 不管按下哪一个键, 该事件都将发生。
- 下面我们在【例 6-1】的基础上, 添加 Memo 组件进行一个小的练习。

【例 6-2】Memo 组件使用练习

在上面的 Edit 框中输入数字, 内容会自动与下面的 Edit 框内容合并出现在 Memo 组件中; 也可以直接在 Memo 中输入内容, 但是当操作的焦点离开 Memo 组件后, 组件的颜色就会变为绿色, 同时下面的 Edit 框的内容显示为“焦点转换”。程序画面如图 6-7 所示。

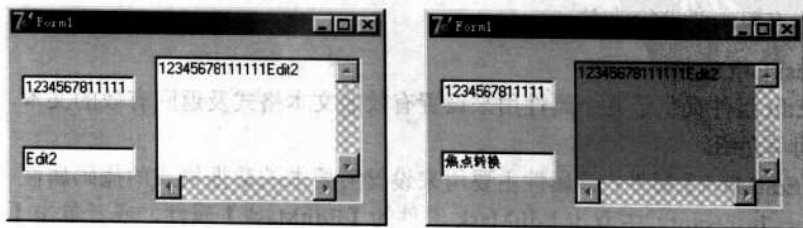



图 6-7 Memo 组件使用练习

- (1) 打开【例 6-1】的程序，调整两个 Edit 组件的位置和大小。
- (2) 在【Standard】选项卡中，单击  按钮，在窗体上添加 Memo 组件，并适当调整组件的位置和大小。
- (3) 选择上面的 Edit 组件，在【Object Inspector】中，选择其【OnChange】事件，双击进入代码编辑器，修改事件代码如图 6-8 所示。这样，Memo1 的内容就是 Edit1 与 Edit2 内容的合并。

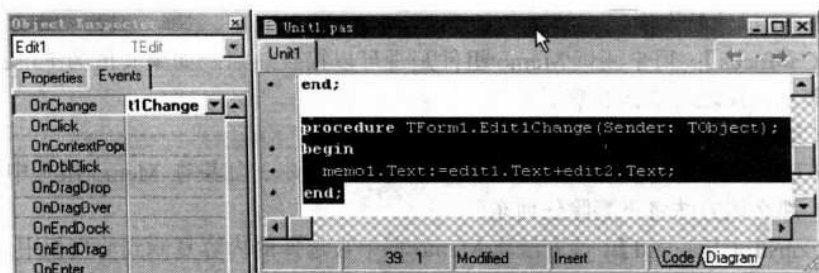


图 6-8 修改 Edit1 组件的【OnChange】事件的代码

- (4) 选择 Memo 组件，在【Object Inspector】中，选择其【OnExit】事件，双击进入代码编辑器，输入如图 6-9 所示代码。设置当焦点离开时，改变 Edit2 的内容，并将 Memo1 的颜色修改为绿色。

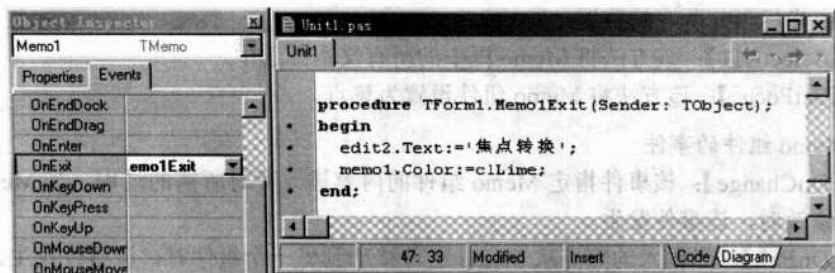


图 6-9 编写 Memo 组件的【OnExit】事件代码


- (5) 运行程序，可以看到，在 Edit1 中输入内容，会自动出现在 Memo 组件中；焦点离开 Memo 组件，Edit2 的内容和 Memo 的色彩就会发生变化。

6.2.3 MaskEdit 组件

掩码编辑框 (MaskEdit) 组件主要用来设计一些复杂的输入格式，可以用在对输入的类型、格式有限制要求的地方。

1. MaskEdit 组件的属性

MaskEdit 组件提供了几个属性用来设置有效的文本格式及返回有效的文本，下边对其属性进行简单介绍。

- (1) 【EditMask】属性：该属性主要用来设置所要求的数据格式的掩码属性。在对象查看器 (Object Inspector) 中双击 EditMask 组件的【EditMask】属性，或者单击【EditMask】属性右方的  按钮，则会打开【格式编辑器】 (Input Mask Editor) 对话框，如图 6-10 所

示。利用该对话框可以对掩码样式进行定义。

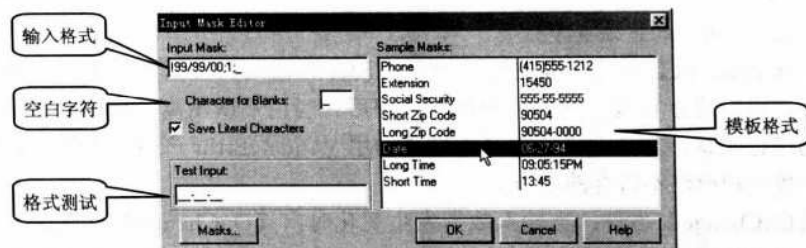


图 6-10 定义掩码样式

左上角的【Input Mask】(输入掩码)编辑框是用来输入所要的掩码格式的,这种格式可以分为3个部分,每两个部分用“;”分隔开,例如“!0000/99/99;1;_”,格式的含义如下。

输入格式的第一部分是掩码本身,也就是数据的输入格式,用一些特殊的格式符类来表示应输入的字符类型以及格式,例如“!0000/99/99”。其中:

- 1 (L): 1 表示该位置只可能是一个字母,可用光标键跳过,但不能是其他类型的字符; L 表示该位置必须有一个字母。
- a (A): a 表示该位置只可能是一个字母或者数字; A 表示该位置必须有一个字母或者数字。
- c (C): c 表示该位置只可能是一个字符; C 表示该位置必须有一个字符。
- 9 (0): 9 表示该位置只可能是一个数字; 0 表示该位置必须有一个数字。
- < (>): < 表示随后的字母均以小写显示,直到遇到“>”格式符或者“<”格式符; > 则与 < 正好相反。
- <>: 表示不做大小写的转换,以输入时的形式为准。
- \: 该格式符之后的那个掩码格式符将被作为数据中的普通字符对待,即作为数据中的一部分。
- #: 该位置可能是一个数字或者正负号。
- !: 表示输入数据前的空格部分将不被保存在数据中,无!表示数据后的空格不被保存,!只能放在掩码格式的第一个字符处。
- ; (/): 标准的分隔符,可以作为数据的一部分保存起来,其中“/”在显示的时候是“-”形式。

在输入格式的第二部分只能是1或者0。如果为1,掩码格式中非用户输入的数据和标准分隔符等作为数据的一部分进行保存;如果为0则不进行保存。

输入格式的第三部分用于表示数据中的空格用什么字符代替。

除了标准格式外,我们还可以用一些需要的字符来对内容进行分隔。例如,要在掩码编辑框中输入如“2004年05月18日”的数据格式,可以将掩码格式设置为“!0000年99月99日;1;_”。

(2) 【IsMasked】属性:该属性用来指定一个掩码编辑组件的掩码是否有效。通过读取 IsMasked 属性的值,来判定组件的掩码属性是否有效,若为真,则数据输入中掩码有效:

若为假，则在数据输入过程中，对组件的输入不进行有效性检查。

2. MaskEdit 组件的方法和事件

(1) **【ValidateEdit】**方法：根据当前掩码的设置对输入的数据进行确认。该方法是一个虚拟方法，可以进行重写。它在用户每次击键后和当用户试图离开掩码编辑框时对输入数据和掩码进行确认。如果两者不符合，则会调用 **ValidateError** 产生一个异常，并把光标移到第一个数据和掩码不符合的地方。

(2) **【OnChange】**事件：当输入数据发生变化时，**【OnChange】**事件触发，可以在这个事件句柄中编写对此事件的处理代码，这个事件给用户提供了第一次对数据修改进行处理的机会。

下面我们使用 MaskEdit 组件来进行一个练习。

【例 6-3】MaskEdit 组件的应用练习

要求用户在掩码编辑框中输入日期，格式是“××××年××月××日”；输入完毕后，按下 **Enter** 键，会将输入的日期显示在一个标签组件中。如果输入的日期格式不符合格式要求，就会出现错误信息。程序画面效果如图 6-11 所示。

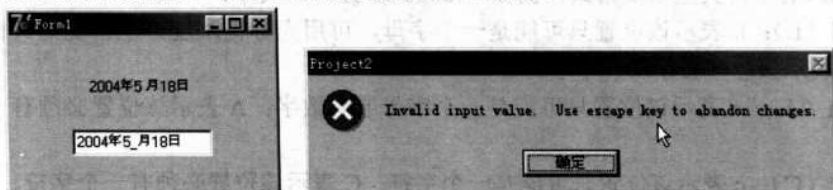


图 6-11 MaskEdit 组件的应用练习

(1) 在窗体中添加一个 MaskEdit 组件和一个 Label 组件。

(2) 选择 MaskEdit 组件，然后在 **【Object Inspector】** 窗口中，单击 **【EditMask】** 属性右方的 按钮，打开 **【Input Mask Editor】** 对话框。

(3) 在 **【Input Mask Editor】** 对话框的 **【Input Mask】** 栏，输入如图 6-12 所示输入格式，定义输入格式为“__年__月__日”。

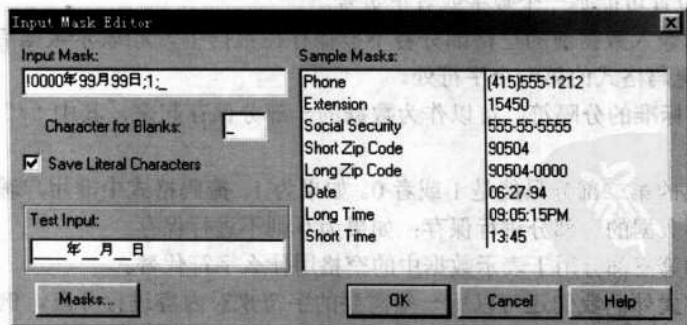


图 6-12 定义输入格式为“__年__月__日”

(4) 单击 **OK** 按钮，关闭对话框。

(5) 在 **【Object Inspector】** 窗口中，打开 **【Events】** 面板，在 **【OnKeyPress】** 事件中双

击，进入代码编辑器，在其中输入如下所示代码。

```
procedure TForm1.MaskEdit1KeyPress(Sender: TObject; var Key: Char);
begin
    if key=chr(VK_RETURN) then //如果按下 Enter 键
        Label1.Caption:= MaskEdit1.Text; //将 MaskEdit1 的内容显示在 Label1 中
    end;
```

(6) 运行程序。按照格式要求输入日期后，按下 Enter 键，输入日期就会显示在 Label 组件中；若输入的格式错误，就会出现错误提示。这时只需要关闭对话框，然后单击 Esc 键，就能够回到初始的输入状态。

6.3 按钮及分类组件

按钮组件为用户提供用来选择执行命令的组件（通常也称为命令按钮）。按钮在 Windows 程序中应用非常广泛。它可以放在应用程序的任何地方，在触发后，产生用户希望完成的工作，用户可以利用这些按钮来执行一些特定的操作。本章将讲述 Delphi 提供的各种按钮组件的使用方法。

6.3.1 Button 组件

Button 组件是最常用的按钮组件，几乎任何程序都离不开它。下面讲述了该组件的常用属性和方法。

1. Button 组件的主要属性

(1) **【Cancel】**属性：该属性决定按钮是否为【Cancel】按钮。当为 True 时，那么无论用户什么时候按 Esc 键，按钮的【OnClick】事件都将发生。该属性默认值为 True。

(2) **【Default】**属性：该属性指出按钮是否为默认按钮。如果将它设置为 True，那么无论用户什么时候按 Enter 键，按钮的【OnClick】事件都将发生。当任何一个按钮有焦点之后，它将成为暂时的默认按钮。

(3) **【Caption】**属性：该属性标识按钮的标题。它是一个文本字符串，用户可以根据其用途定义相应的名字，还可以调整字体的大小、颜色等属性。

2. Button 组件的主要事件

(1) **【OnClick】**：当用户单击鼠标上的按钮后，触发该事件。同时，出现以下情况时，【OnClick】事件也将被触发：

- 当按钮有一个焦点，且用户按下了空格键；
- 当窗体有一个默认按钮（Default 属性所指定），而且用户按下了 Enter 键；
- 当窗体有一个【Cancel】按钮（Cancel 属性所指定），而且用户按下了 Esc 键。

(2) **【OnEnter】**：当按钮成为当前按钮时，该事件发生。所谓当前按钮，即是按钮上带有虚线方框的按钮。

(3) **【OnKeyDown】**：当按钮成为当前焦点之后，不管按下哪个按钮，该事件都发生。

3. Button 组件的主要方法

(1) **【Focused】**：该方法用来确定按钮有无焦点，如果有，说明它是窗体上的

ActiveControl 组件。

(2) **【Hide】**: 该方法可以设置按钮的 Visible 属性, 使它为可视或者不可视。

(3) **【SetFocus】**: 该方法用来设置按钮成为当前焦点。

(4) **【Show】**: 该方法将按钮的 Visible 属性设置为 True, 使其成为可视组件。

Button 按钮的使用比较简单, 我们在前面的练习中也都有过介绍, 所以这里就不再举例说明了。

6.3.2 BitBtn 组件

BitBtn 组件位于**【Additional】**组件面板中, 它继承自 Button 组件, 拥有 Button 组件的所有属性和方法, 另外还包括一些自己独特的属性, 这里列举如下:

(1) **【Glyph】**属性: 该属性中可以放置图形。可以自己制作位图按钮上的图形, 也可以采用 Delphi 指定的一组默认图形。如果自己制作图形, 可以将位图保存在.bmp 文件中, 在设计阶段就可以通过 Glyph 属性将图形显示在位图按钮上。采用 Delphi 指定的一组默认图形, 可以通过**【Kind】**属性来设置。具体取值可以为 bkCustom、bkOK、bkCancel、bkHelp、bkYes、bkNo、bkClosebkAbort、bkRetry、bkIgnore 和 bkAll 等, 它们分别具有明显的意义。对于自己制作图形的位图按钮, **【Kind】**属性要设置为 bkCustom。

(2) **【NumGlyphs】**属性: 该属性指明该位图按钮使用图形的个数, 可以设置为 1~4 之间的一个整数。如果 Kind 属性不为 bkCustom, 也就是使用 Delphi 指定的一组默认图形时, NumGlyphs 自动设置为 2。对于自己制作图形的位图按钮, 如果只有一个位图, 就将 NumGlyphs 的值设置为 1, 如果设置为 2, 在按钮上只能显示一半的图形。在位图按钮具有多个图形的时候, 位图按钮通常显示的是第 1 个图形; 当按钮被按下后, 显示第 3 个图形; 当按钮失效后 (Enabled 属性为 False), 显示第 2 个图形。

位图按钮包含有几个图形, 可以自己设定, 然后 Delphi 自动将指定的图形在水平方向上进行等分, 并在不同的情况下显示不同的图形。

6.3.3 SpeedButton 组件

SpeedButton 组件也位于**【Additional】**组件面板中, 通常称为快速按钮, 它在 Windows 应用程序中被广泛采用。它常和 TPanel 控件一起使用, 用户通过它可以快速建立工具栏和工具面板的任务。该组件可以作为一个组在一起使用, 它和位图按钮一样, 在按钮上面可以显示图像, 但是它的图像必须是 Windows 位图文件及扩展名为 bmp 的文件。

该组件具有几个重要的属性。

(1) **【AllowAllup】**属性: 该属性决定在一组中的快速按钮是否可以一个都不选择。如果将它设置为 True, 那么在同一组中的快速按钮可以一个都不选, 但可以出现; 如果将它的属性设置为 False, 则必须选择同一组中的快速按钮之一, 即处于被按下状态。它的默认值为 False。




(2) **【Down】**属性: 该属性决定快速按钮在选择或者未被选择时的显示状态。如果将它设置为 True, 那么快速按钮最初以按下状态显示; 如果将它设置为 False, 则快速按钮以通常状态**【Up】**显示。这种状态也是未选择状态。

(3) **【GroupIndex】**属性: 该属性决定哪些快速按钮作为一组在一起工作。默认时, 它的值为 0, 这说明该组件不属于某个组。具有相同 GroupIndex 属性值, 且该值不为 0 的

快速按钮将作为同一组，当用户单击其中之一时，该按钮一直保持它的被按下（Down）状态，直到单击同一组的另一个组件为止。

下面我们以一个简单的例子来说明 BitBtn 组件和 SpeedButton 组件的应用。

【例 6-4】按钮组件应用练习

窗体上有三种类型的按钮，单击  按钮或者  按钮中的一个，会使另一个失效；单击  按钮，则使上面的两个按钮都有效。程序画面如图 6-13 所示。

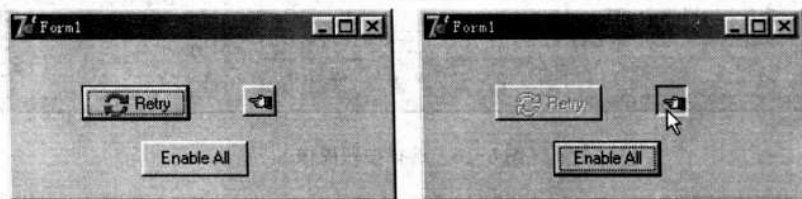


图 6-13 按钮组件应用练习

- (1) 在窗体上创建一个 BitBtn 组件、一个 SpeedButton 组件和一个 Button 组件。
- (2) 选择窗体上的 BitBtn 组件，在【Object Inspector】窗口，选择【Kind】属性右侧的下拉按钮，从弹出的下拉框中选择“bkRetry”项，如图 6-14 所示。则 BitBtn 按钮上显示出“Retry”字样。
- (3) 选择窗体上的 SpeedButton 组件，在【Object Inspector】窗口，双击【Glyph】属性，会打开一个【Picture Editor】对话框，如图 6-15 所示。

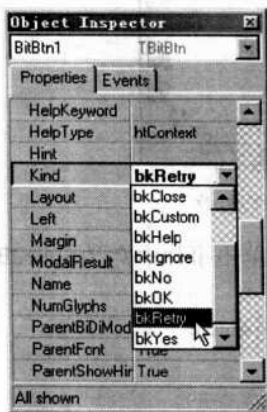


图 6-14 定义 BitBtn 组件的【Kind】属性为“bkRetry”

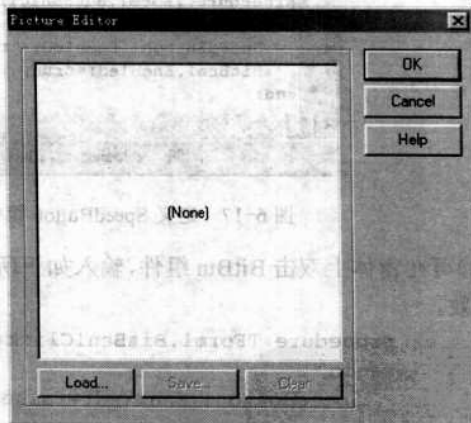



图 6-15 【Picture Editor】对话框

- (4) 单击  按钮，则打开【Load Picture】对话框，选择一种按钮图像，则对话框的右侧会显示出图像的样式，如图 6-16 所示。

提示：在 Delphi 7 的安装目录下，有“Borland Shared/Images/Buttons”文件夹，其中提供了 Windows 常用按钮图像。

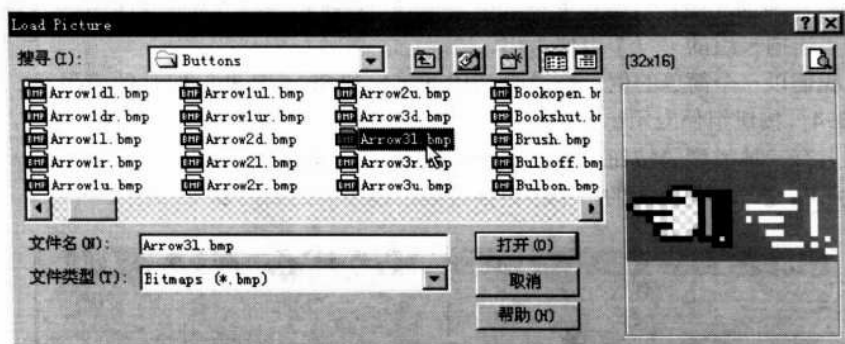


图 6-16 选择按钮图像

(5) 单击 **打开(O)** 按钮, 则选择的图像出现在 **【Picture Editor】** 对话框中; 再单击 **OK** 按钮, 则图像出现在 SpeedButton 按钮上。

(6) 选择窗体上的 Button 组件, 在 **【Object Inspector】** 窗口, 定义其 **【Caption】** 属性为 “Enable All”。

(7) 双击 Button 组件, 在代码编辑器中, 输入如图 6-17 所示代码, 在 Button 按钮的单击事件中, 定义 SpeedButton 组件和 BitBtn 组件均有效。

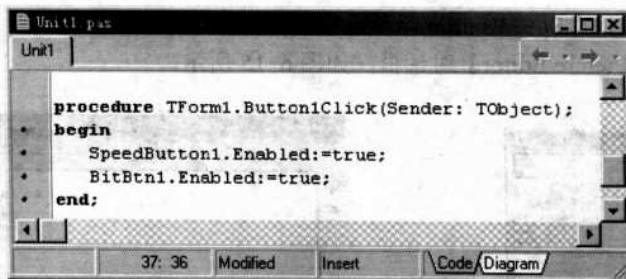


图 6-17 定义 SpeedButton 组件和 BitBtn 组件均有效

(8) 再在窗体上双击 BitBtn 组件, 输入如下所示代码, 在其单击事件中定义 SpeedButton 组件无效。

```

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    SpeedButton1.Enabled:=false;
end;
  
```

(9) 同样, 在窗体上双击 SpeedButton 组件, 输入如下代码, 在其单击事件中定义 BitBtn 组件无效。

```

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    BitBtn1.Enabled:=false;
end;
  
```

(10) 运行程序, 可以看到 3 个按钮之间的相互控制关系, 以及按钮上图像在有效和

无效两种状态下的变化。

6.3.4 CheckBox 组件

该组件也称为复选框，用户可以利用它来进行“是/否”或“真/假”选择。可在一个选择组框中一次作出多个选择。用户可任选一个将其设定为【Checked】（已被选择），也可以将其设定为 UnChecked（未被选择）。

(1) 【AllowGrayed】属性：该属性用来控制是否允许灰色选择。通常复选框有两种状态，即选择与不选择，分别用打上叉与空来代表，但是 Delphi 允许复选框有一个称为第三种状态的中间状态，即在复选框中显示一个灰色标记。该属性默认值为 False，这时复选框只有两种状态，但当该属性为 True 时，复选框有 3 种状态。

(2) 【Checked】属性：该属性用来确定复选框是否处于选中状态，它的默认值为 False，也就是未选中状态。当其值被设置为 True

(3) 【State】属性：该属性用来检测复选框处于什么状态。它有如下 3 种取值：

- cbUnchecked：处于未被选中状态。
- cbChecked：处于选中状态。
- cbGrayed：处于灰色状态。

6.3.5 RadioButton 组件

该组件通常被称为单选按钮，单选按钮作为一组来工作，它为用户提供了相互排斥的一个选项组，无论何时选项组中最多只有一个选项被选择，如果在选择过程中又选择了其他按钮，那么先前被选择的按钮会变成未选择。

该组件的主要属性如下：

(1) 【Checked】属性：该属性指出单选按钮是否被选中。如果该属性为 True，则单选按钮的框中出现一个圆点，表示选中。默认为 False，表示不选中。

(2) 【Visible】属性：该属性决定该单选按钮在窗体上是否可视。默认值为 True，即可视。

在实现单选按钮的分组时，可以有两种方法。

- 利用分组框组件 (GroupBox) 和单选按钮组件 (RadioButton) 实现。首先向窗体中添加分组框，然后向分组框中添加单选按钮。这样，同一个分组框中的单选按钮就自动成为一组。可以通过 RadioButton 组件的【Checked】属性来确定哪一个单选按钮被选中。

- 利用单选按钮分组框 (RadioGroup) 实现。可以通过【ItemIndex】属性来确定选中哪一个单选按钮。

下面我们通过一个例子来练习 CheckBox 组件和 RadioButton 组件的使用。

【例 6-5】选择组件应用练习

程序用于显示进货情况。当用户选择产地或货物时，下面的编辑框中将自动出现相应的信息。每次选择的货物可以是一个或多个，但产地只能有一个。程序效果如图 6-18 所示。

- (1) 创建一个新的应用程序；设置窗体 (Form1) 的【Caption】属性为“进货情况”。
- (2) 按照程序效果要求在窗体上添加各个组件，如图 6-19 所示。

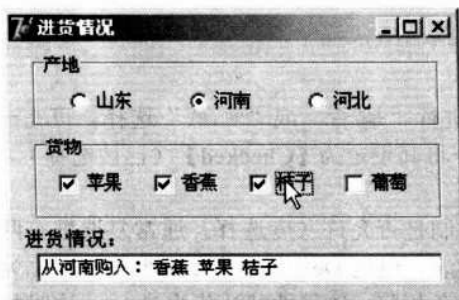


图 6-18 进货情况表单



图 6-19 在窗体上添加组件

窗体中各组件的属性如表 6-3 所示。

表 6-3 窗体中各组件的属性

组 件	【Name】属性	【Caption】属性
RadioButton	RadioButton1	山东
RadioButton	RadioButton2	河南
RadioButton	RadioButton3	河北
CheckBox	CheckBox1	苹果
CheckBox	CheckBox2	香蕉
CheckBox	CheckBox3	桔子
CheckBox	CheckBox4	葡萄
Label	Label1	进货情况
Edit	Edit1	

(3) 整个程序的代码清单如下所示:

```
//为节约篇幅,前面由系统自动创建的代码就不再罗列
implementation
{$R *.dfm}
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    edit1.Text:='';
    edit1.Text:='从'+radiobutton1.Caption+'购入: ';
end;
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
    edit1.Text:='';
    edit1.Text:='从'+radiobutton2.Caption+'购入: ';
end;
procedure TForm1.RadioButton3Click(Sender: TObject);
begin
    edit1.Text:='';
    edit1.Text:='从'+radiobutton3.Caption+'购入: ';
end;
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    //如果编辑框中无内容或复选框未变为选中状态则不向下执行代码
```



```

if(edit1.Text='') or (checkbox1.Checked=false) then
    exit;
edit1.Text:=edit1.Text+' '+checkbox1.Caption+' ';
end;
procedure TForm1.CheckBox2Click(Sender: TObject);
begin
    if(edit1.Text='') or (checkbox2.Checked=false) then
        exit;
    edit1.Text:=edit1.Text+' '+checkbox2.Caption+' ';
end;
procedure TForm1.CheckBox3Click(Sender: TObject);
begin
    if(edit1.Text='') or (checkbox3.Checked=false) then
        exit;
    edit1.Text:=edit1.Text+' '+checkbox3.Caption+' ';
end;
procedure TForm1.CheckBox4Click(Sender: TObject);
begin
    if(edit1.Text='') or (checkbox2.Checked=false) then
        exit;
    edit1.Text:=edit1.Text+' '+checkbox2.Caption+' ';
end;
end.

```

(4) 运行程序。选择不同的产地和货物, 就能够在下面的编辑框中显示相应的结果。

但是也许大家很快就会发现两个问题: 第一, 如果我们对一个货物选项反复选择, 其内容就会重复出现在编辑框中; 第二, 即便取消了对某个货物的选择, 该货物还是会显示在编辑框中。如图 6-20 所示。

要解决这个问题, 需要对编辑框中的字符串内容进行搜索并判断某项货物是否会显示。这就需要用到字符串处理函数。下面我们以后一个货物“葡萄”为例来解决这个问题。

(5) 修改“CheckBox4”组件的“Click”事件中的代码如下:

```

procedure TForm1.CheckBox4Click(Sender: TObject);
var
    charpos:integer; //定义变量, 记录字符在字符串中出现的位置
    tempString:string; //定义变量, 记录编辑框中的字符串内容
begin
    tempString:=edit1.Text; //用变量保存编辑框中的字符串内容
    if(edit1.Text='') then exit; //如果编辑框中无内容, 说明没有选择产地
    charpos:=pos('葡萄', edit1.Text); //记录字符“葡萄”在编辑框中出现的位置
    if(checkbox4.Checked) then //如果 checkbox4 被选择
        if(charpos=0) then //如果字符位置为 0, 说明没有这个字符
            edit1.Text:=edit1.Text+' '+checkbox4.Caption+' '; //添加货物内容

```

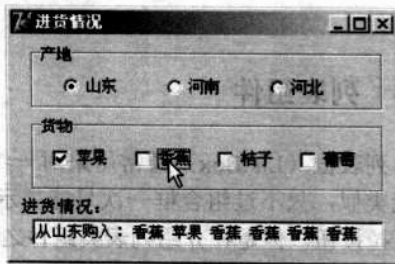


图 6-20 货物会反重复出现在编辑框中

```

if (checkbox4.Checked=false) then //如果取消对 checkbox4 的选择
if(charpos<>0) then           //如果字符位置为 0
begin
delete(tempString,charpos,length('葡萄'));
//删除货物内容字符
edit1.Text:=tempString;      //将临时字符串变量中的内容重新赋给编辑框
end;
end;

```

(6) 同理, 修改其他几个 CheckBox 组件进行的代码。

(7) 重新运行程序, 可以看到, 货物名称只能在编辑框中出现一次; 而且, 如果取消对某个货物的选择, 该货物名称也会消失。如图 6-21 所示。

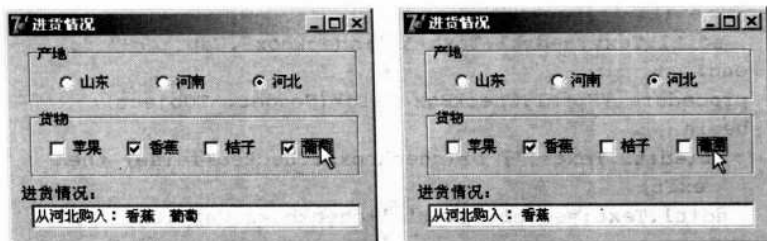


图 6-21 控制货物的选择和取消

6.4 列表组件

列表框 (ListBox) 通常用来对一组信息进行列表, 列表框和组合框 (ComboBox) 的功能类似, 只不过组合框一次只能显示一条信息, 要浏览全部信息, 需要按下组合框旁边的下拉按钮。但是在介绍列表类组件之前, 有必要介绍一下 TString 类。

6.4.1 TString 类

TStrings 是一个抽象类, 它并没有真正实现操纵字符串的代码, 只是定义了一系列方法。TStrings 的派生类组件实现了实际上的字符串操作方法, 用户可以操纵属于组件 (如下面讲到的 TComboBox、TListBox 等组件) 的字符串列表。

TComboBox.Items、TListBox.Items 等属性都是 TString 类型。如果希望直接使用这些功能而不依赖于具体组件的话, 就可以使用 TString 的方法。

TStrings 常用方法有:


- Add: 将一个字符串加到字符串列表中, 并返回字符串在列表中的位置。
- Assign: 为字符串列表赋值。
- Clear: 从列表中删除所有字符串。
- Delete: 删除位于列表某一位置上的字符串。
- Exchange: 交换列表中某两个字符串的位置。
- IndexOf: 获得字符串在列表中的位置。

- **Insert**: 把字符串插入到列表中的指定位置。
- **Move**: 把列表中某个位置的字符串移动到新的位置。
- **LoadFromFile**: 从文本文件中读取字符串列表。
- **SaveToFile**: 把字符串列表保存到文本文件中。

6.4.2 ListBox 组件

该组件通常也被称为列表框。它用来显示可滚动的项目列表，在项目列表中用户可以选择一个或者多个项目，但是不能直接对这些项目进行修改。在列表框中的项目列表是 **Items** 属性的值，可使用方法对列表框中的项目进行增加、插入和删除操作。

1. ListBox 组件的属性

- (1) **【Columns】**: 该属性用来控制列表框中的项目显示成几列。
- (2) **【Height】**: 该属性决定列表框在垂直方向的高度。
- (3) **【ItemHeight】**: 当列表框的 **【Style】** 属性为 “**lbOwnerDrawFixed**” 时，该属性指出列表框项目的高度，当 **【Style】** 属性为其他值时，它将被忽略。
- (4) **【ItemIndex】**: 该属性指出在列表框中哪个项目被选择。
- (5) **【Items】**: 该属性指出在列表框中显示的所有字符串。当按下 **【Items】** 属性右边的  按钮之后，会弹出 **【列表框编辑】** 窗口，用户可在这个编辑窗口中输入将要在列表框中显示项目的默认值。也可以在程序执行过程中通过调用方法动态地向列表框中添加数据。
- (6) **【MultiSelect】**: 该属性决定用户是否可以从列表框中一次做出多项选择。如果将它设置为 **True**，那么程序执行时用户可以选择多项。它的默认值为 **False**。
- (7) **【Selected】**: 该属性检测在列表框中特定的项目是否被选择，入口参数 **X** 是将要访问的项目在列表框中的位置，第一个项目的位置为 0，依次为 1、2、3…。如果指定的项目被用户选择，则该值为 **True**，反之为 **False**。
- (8) **【Sorted】**: 该属性决定是否自动地为用户进行排序。如果其值为 **True**，那么列表框中的数据按照字母的顺序进行排序。



2. ListBox 组件的方法

- (1) **【Clear】**: 该方法清除在列表框中的所有项目。
- (2) **【ScreenToClient】**: 该方法返回列表框在屏幕上的当前位置。

3. ListBox 组件的事件

- (1) **【OnDrawItem】**: 当列表框中的内容重新显示时，该事件发生。比如，当用户选择了列表框中的项目，系统需对选择的项加重显示，此时就触发 **【OnDrawItem】** 事件。但是该事件仅当 **【Style】** 属性值为 “**lbOwnerDrawfixed**” 或者 “**lbOwnerVariable**” 时才会发生。
- (2) **【OnEnter】**: 当 ListBox 组件成为当前列表框时，该事件发生。
- (3) **【OnMeasureItem】**: 不管何时应用程序需重新显示在列表框中的项目时，该事件发生。在事件发生后，它将测出需重画区域的大小参数传给 **【OnDrawItem】** 事件，由该事件重新显示给定的区域。

【例 6-6】列表框应用练习

在“候选人员”列表框中选择某一项，然后单击  按钮，则选定人员被添加到右边的“选择人员”列表框中，同时，该项目被从“候选人员”列表框中移除；同样，使用  按钮可以实现相反功能。程序效果如图 6-22 所示。

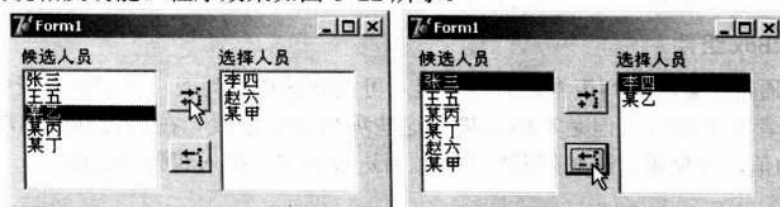



图 6-22 选择人员

(1) 创建一个新的应用程序。

(2) 按照程序效果要求在窗体上添加各个组件，其中两个按钮使用的是【Additional】选项卡中的【BitBtn】组件。

(3) 为两个【BitBtn】组件按钮添加图形。方法是：选择组件的【Glyph】属性，通过对话框加载一个按钮位图图像。

提示：这些图像一般位于“C:\Program Files\Common Files\Borland Shared\Images\Buttons”中。

(4) 选择 ListBox1 组件，单击【Items】属性的  按钮，会弹出【String List Editor】对话框，在其中创建一些人员列表，如图 6-23 所示。

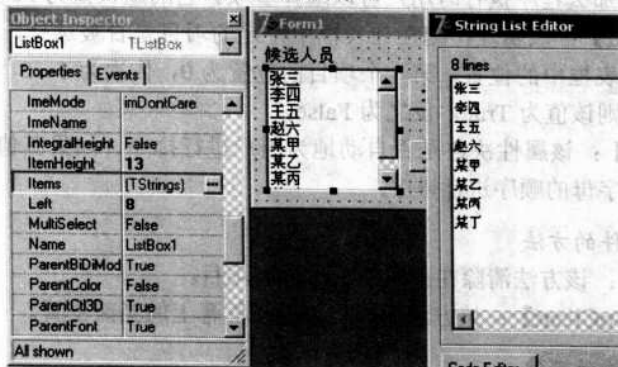



图 6-23 为 ListBox1 组件创建项目列表

(5) 单击  按钮，关闭对话框。

(6) 为 BitBtn1 按钮和 BitBtn2 按钮的 Click 事件创建代码如下：

```
... //为节约篇幅，前面由系统自动创建的代码就不再罗列
implementation
{$R *.dfm}
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    if(listbox1.ItemIndex=-1) then //如果没有项目被选择
```

```

        messagedlg('请选择需要移动的项目',mtcustom,[mbok],0)
    else
        begin
            //添加到另一个列表框
            listbox2.items.Add(listbox1.Items.Strings[listbox1.ItemIndex]
        );
            //在原有列表框中删除该项目
            listbox1.items.Delete(listbox1.ItemIndex);
            //如果此时原有列表框中还有项目,移动焦点到第一个项目上来(索引为0)
            if(listbox1.Count<>0) then
                listbox1.Selected[0]:=true;
            end;
        end;
    procedure TForm1.BitBtn2Click(Sender: TObject);
    begin
        if(listbox2.ItemIndex=-1) then
            messagedlg('请选择需要移动的项目',mtcustom,[mbok],0)
        else
            begin
                listbox1.items.Add(listbox2.Items.Strings[listbox2.ItemIndex]);
                listbox2.items.Delete(listbox2.ItemIndex);
                if(listbox2.Count<>0) then
                    listbox2.Selected[0]:=true;
                end;
            end;
        end;
    end.

```

(7) 运行程序,就可以通过两个按钮实现人员的选择和取消了。

6.4.3 ComboBox 组件

该组件称为组合框,它是设计 Windows 应用程序时使用较多的组件。它汇集了列表框和编辑框的功能,使用户可从下拉列表中选择数据或者直接向组合框中输入数据。

该组件的主要属性如下。

(1) **【Style】**: 该属性用来改变组合框的类型。组合框有 5 种类型:

- **csDropDomp**: 在组合框的编辑框中可以输入,同时下拉式列表框是一组具有相等高度的字符串。
- **csDropDownList**: 组合框的编辑框为只读的,而且编辑框中的内容只能从列表框中选择。
- **csOwnerDrawFixed**: 组合框由一个只读编辑框和条目高度相等的列表框组成,条目的高度由属性 **ItemHeight** 确定。
- **csOwnerDrawVariable**: 组合框由一个只读编辑框和列表框组成,条目的高度可以不同。
- **csSimple**: 只有一个编辑框显示,可以在编辑框中输入字符串,也可以通过上、下箭头键改变编辑框中的内容。


(2) **【Enabled】**: 该属性确定组合框是否处于激活状态。如果为 **False**,则处于非激活状态,程序运行后,组合框的区域呈现为灰色。

(3) **【ItemHeight】**: 该属性用来指定组合框每一行的高度。注意,使用该属性时,

【Style】属性必须为“csOwnerDrawFixed”。

(4) 【ItemIndex】：该属性指定组合框中被选择的项目。

(5) 【SelStart】：该属性确定组合框中所选文本的开始位置。

(6) 【Items】：该属性包含出现在组合框下拉列表中的所有字符串。单击该属性右侧的按钮后，显示【String List Editor】对话框，在其中可以输入需要在下拉列表中显示的文本。

下面以一个实例说明 ItemBox 和 ComboBox 组件的使用方法。

【例 6-7】下拉组件应用练习

通过一个组合框选择需要了解的班级的各种情况，在列表框中会显示具体的内容。程序效果如图 6-24 所示。

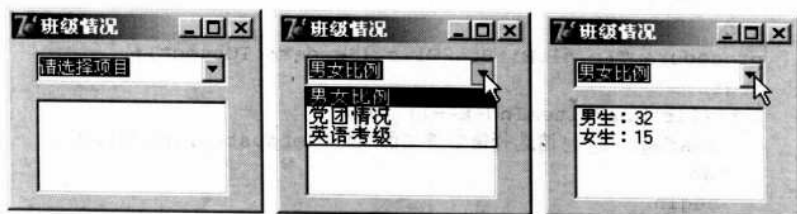


图 6-24 班级情况

(1) 创建一个新的应用程序，适当调整窗体大小；设置窗体的【Caption】属性为“班级情况”。

(2) 在窗体上添加一个组合框和一个列表框。

(3) 选择 ComboBox 组件，在其【Items】属性中，添加 3 个项目“男女比例”、“党团情况”、“英语考级”。

(4) 在【Object Inspector】中，选择 ComboBox1 组件，打开【Events】选项卡，双击“OnChange”事件，在代码编辑器中输入如下所示代码：

```
procedure TForm1.ComboBox1Change(Sender: TObject);
var
  selstr:string;
  selline:integer;
begin
  selstr:=combobox1.Items[combobox1.ItemIndex]; //获得当前选项的内容
  if(selstr='') then exit;
  //如果用户没有选择项目，就不执行下面的代码
  selline:=combobox1.Items.IndexOf(selstr); //获得当前选项的位置值
  //判断当前选择的是哪个项目
  case selline of
    0: //选择了第1项
    begin
      listbox1.Items.Clear; //先清空列表框中的内容
      listbox1.Items.Add('男生: 32'); //为列表框添加内容
      listbox1.Items.Add('女生: 15');
    end;
  end;
end;
```

```

begin
    listbox1.Items.Clear;
    listbox1.Items.Add('团员: 33');
    listbox1.Items.Add('党员: 7');
end;
2:
begin
    listbox1.Items.Clear;
    listbox1.Items.Add('四级通过: 41');
    listbox1.Items.Add('六级通过: 12');
end;
end;
end;

```

提示: 其余部分的内容是由系统自动产生。

(5) 运行程序。当用户选择了某个项目后, 相应的内容就会出现在列表框中。

6.4.4 TreeView 组件

树状视图通常用于表示层次结构比较明显的信息。例如用于表示一个单位的机构组成、人事上的管理关系、磁盘上的目录和文件列表等。最典型的应用莫过于 Windows 系统中的资源管理器。

1. TreeView 组件的属性

(1) **【AutoExpand】**: 决定树状视图是不是自动展开。如果取值为真, 则自动展开各个子节点。

(2) **【Images】**: 决定哪个图像列表组件 (ImageList 组件) 和树状视图组件相连接。在树状视图组件中, 通常每个节点的左边都会有一个与之相关联的图标。例如, 在 Windows 系统的资源管理器中, 桌面、我的电脑、软盘、硬盘等节点的图标就是各不相同的。为了给每一个节点附上一个图标, 首先要做的就是将一个 ImageList 组件放到窗体上, 并建立一个由多个图像构成的列表, 然后把树状视图组件的 **【Images】** 属性指向这个 ImageList 组件, 这样就可以通过节点的 **hageIndex** 属性指定在图像列表中与该节点相关联的图标的序号了。

(3) **【Items】**: 列出树状视图组件中的节点。树状视图组件所有节点是一个 TTreeNode 对象, 而每个节点又是一个 TTreeNode 对象。节点可以通过 **【Items】** 与节点的索引来访问。例如, 要访问树状视图组件中第三个节点, 可以使用以下代码:

```
MyTreeNode:=Treeview1.Items[2];
```

在设计阶段, 可以通过单击 **【Items】** 属性中右方的 **【...】** 按钮, 打开树状视图项目编辑器 (TreeView Items Editor) 来建立树状视图组件中的节点。

2. TreeView 组件的方法和事件

(1) **【ClearSelection】**: 该方法对选择的节点清除选择标记。其声明原型为:

```
procedure Clearselection(keepprimary:Boolean=False); virtual;
```

如果 keepPrimary 参数为真, 则表示清除选择标记的时候保留第一个选择的节点。

(2) **【GetHitTestInfoAt】**: 该方法返回指定点与树状视图的关系。其声明原型为:


```
function GetHitTestInfoAt(X,Y:Integer):ThitTests;
```

这个函数可能的返回值以及返回值的含义如下:

- htAbove: 该点在树状视图客户区的上方;
- htBelow: 该点在树状视图客户区的下方;
- htNowhere: 该点在树状视图客户区的里面,但在最后一个节点的下面;
- htOnItem: 该点在树状视图某个节点的位图或者标签上;
- htOnButton: 该点在树状视图某个节点的按钮上;
- htOnIcon: 该点在树状视图某个节点的位图上;
- htOnIndent: 该点在树状视图某个节点的缩进线上;
- htOnLabel: 该点在树状视图某个节点的标签上;
- htOnRight: 该点在树状视图某个节点的右边;
- htOnStateIcon: 该点在树状视图某个节点的状态图标上;
- htToLeft: 该点在树状视图客户区的左方;
- htToRight: 该点在树状视图客户区的右方。

(3) **【OnChange】**: 当选择的节点发生变化时该事件发生。

(4) **【OnCollapsing】**: 当一个节点正在折叠时该事件发生。

(5) **【OnExpanding】**: 当一个节点正在发生扩展时该事件发生。

在运行阶段,通过使用 TTreeNode 对象的 Add、AddChild、AddChildFirst、AddChildObject、AddChildObjectFirst、AddFirst、AddNode、AddObject、AddObjectFirst 和 Insert、InsertNode、InsertObject、Delete 等方法来增加、插入和删除节点。

以下代码显示了单击一个节点时就删除此节点:


```
procedure TForm1.Treeview1MouseDown (Sender :TObjectF Button;
TmouseButton; Shift:TshiftState; X,Y:Integer);
Var
HT: ThitTests; //声明一个点击测试变量
begin
HT:=GetHitTestInfoAt(X,Y); //测试当前点击处的信息
if(htOnItem in HT) then //如果点击处在项目、文字或者图标上就删除当前节点
Items.Delete(GetNodeAt(X,Y));
end;
```

下面举例说明如何设计一个树状视图,并显示鼠标单击点的位置属性。





【例 6-8】TreeView 组件应用练习

窗体右侧是一个 TreeView 窗口,左侧是一个 ListBox 窗口。用鼠标在树状视图上操作,单击点的位置信息会显示在列表框中。程序效果如图 6-25 所示。

(1) 创建一个新的应用程序,适当调整窗体的大小。

(2) 选定 TreeView 组件,选择其 **【Items】** 属性,单击  按钮,打开 **【Tree View Items Editor】** 对话框,如图 6-26 所示。

下面是对窗口的简单说明:

- 利用 、、、 按钮可以创建节点、创建子节点、删除节点或载入其他文件中已经存在的节点。

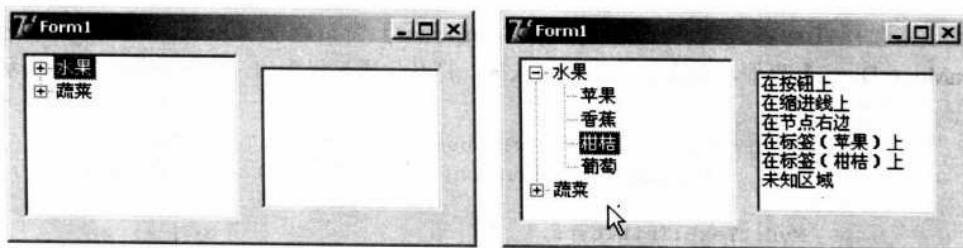


图 6-25 显示鼠标单击点的位置属性

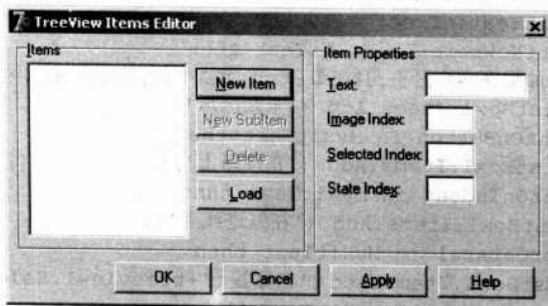


图 6-26 【Tree View Items Editor】对话框

- 在 Text 编辑框中输入项目文本标题，此时对话框左侧的 Items 列表框中将会出现该项目。
- 如果没有选中项目时，要在项目左边显示表示项目未选中的图标，则在【Image Index】编辑框中输入图标索引号；若不要图标，将其设置为-1（默认值）。
- 如果选中项目时，要在项目左边显示表示项目选中的图标，在【Selected Index】编辑框中输入图标索引号；若不要图标，将其设置为-1（默认值）。
- 如果要在项目左边显示其他图标，在 State Index 编辑框中输入图标索引号；若不要图标，将其设置为-1（默认值）。

注意：当删除一个项目时，其子项目会同时被删除。

(3) 根据程序要求创建项目树的内容，如图 6-27 所示。

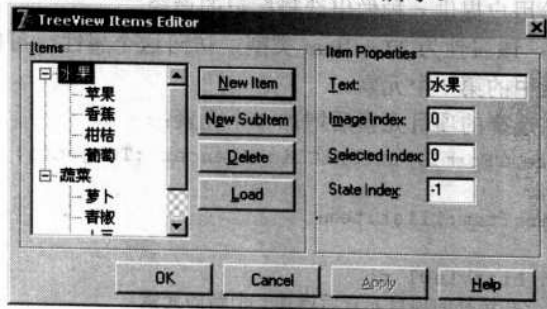


图 6-27 创建项目树

(4) 关闭对话框。

(5) 选择 TreeView 组件, 在【Object Inspector】窗口, 从【Events】标签页中选择【OnMouseDown】事件, 双击其右侧的输入框, 打开代码编辑窗口, 在其中输入如下所示代码:

```
procedure TForm1.TreeView1MouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
Var
    MyHitTest: THitTests;
Begin
    //返回点击位置与树状视图的关系
    MyHitTest:=TreeView1.GetHitTestInfoAt(x,y);
    if htNoWhere in MyHitTest then
        //如果在客户区里,且在最后一个节点下面,则显示未知区域
        ListBox1.Items.Add ('未知区域');
    if htOnButton in MyHitTest then
        ListBox1.Items.Add ('在按钮上');
    if htOnIndent in MyHitTest then
        ListBox1.Items.Add ('在缩进线上');
    if htOnLabel in MyHitTest then
        ListBox1.Items.Add ('在标签('+treeview1.Selected.Text+')上');
    if htonRight in MyHitTest then
        ListBox1.Items.Add ('在节点右边');
end;
```

(6) 运行程序, 可以看到, 单击点的位置信息会显示在列表框中。

6.4.5 ListView 组件

ListView 组件和 TreeView 组件不同, 它并不是将项目元素以树型视图显示, 而是以图标加上文件夹或者文件名的形式显示。最典型的 ListView 组件应用就是 Windows 系统中资源管理器右边的文件显示区域, 双击带图标的文件夹, 就会显示其下一级的内容。

1. ListView 组件的属性

ListView 组件的属性和 TreeView 组件有很大一部分是相同的, 例如 Items、StateImages 等, 另外它还有其他一些属性。ListView 组件常用的属性主要有以下这些:

(1) 【Checkboxes】: 该属性决定在显示列表视图中的项目元素时, 是否在项目元素的前面显示复选框, 给用户提供一种做出选择标记的途径。

(2) 【Selected】: 该属性为 TListItem 类型, 访问这个属性, 可以返回当前列表视图组件中选择的项目元素中的第一个元素。

下面的代码用来在选择的元素后增加一个元素:

```
procedure TForm1.Button1Click(Sender :TObject);
Var
    InsertItem:TListItem;
begin
    with Listview1 do
        //在选择的元素后增加一个新的元素
        InsertItem:=Items.Insert (Selected.Index);
        InsertItem.Caption:=Edit1.Text; //设置新的元素的标题为编辑组件里边的文本
end;
```

(3) **【Viewstyle】**: 该属性用来设置列表视图组件的显示模式。它的取值和含义如下:

- vsIcon: 列表以大图标显示, 可以进行拖放操作。
- vsSmallIcon: 列表以小图标显示, 可以进行拖放操作。
- vsList: 以简单列表的方式显示, 不能进行拖放操作。

2. ListView 组件的方法和事件

(1) **【Clear】**: 从当前的列表视图中删除所有的项目元素。

(2) **【OnColumnClick】**: 当用户用鼠标单击列表视图组件 (以报表方式显示时) 的表头按钮时, 此事件发生。可以利用此事件对每一列元素进行排序或者其他操作, 例如下面的代码将元素按照字母顺序进行排序。

```
procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
begin
    (Sender as TCustomListView).AlphaSort; //按字母顺序进行排序
end;
```

(3) **【OnColumnRightClick】**: 当用户用鼠标右键单击列表视图组件 (以报表方式显示时) 的表头按钮时, 此事件发生。可以在此时弹出菜单以进行其他操作。

6.4.6 ImageList 组件

Windows 系统中大量使用了图标, 为了对程序中的图标进行管理使用, 引入了 ImageList (图像列表组件)。ImageList 组件主要用来建立一组具有相同尺寸的图像的列表, 其中的每一个图像都有自己的索引, 通常情况下, ImageList 组件与 TreeView 组件和 ListView 组件配合使用。

ImageList 组件主要的属性、方法和事件有:

(1) **【BkColor】**、**【BlendColor】**: 这是两个 Tcolor 类型的属性, 用来设置绘制图像列表中的图像时的背景颜色和前景颜色。BkColor 是背景颜色属性, BlendColor 是前景颜色属性。

(2) **【AddIcon】**: 该方法用于把一个图标加入到图像列表组件中。

它与 Add 方法类似, 函数的声明原型为:

```
function AddIcon(Image: TIcon): Integer;
```

其中的 Image 参数为图标对象。返回值的定义和 Add 方法类似。

(3) **【Clear】**: 该方法用来清除图像列表组件中所有的图像。

(4) **【Delete】**: 该方法用来从图像列表中删除某个具体的图像。

ImageList 组件主要用来和其他组件配合使用, 作为一个图像管理的组件来使用, 因此它提供的事件比较少, 只有 **【OnChange】** 事件一个, 当图像列表组件发生变化时此事件即发生。

下面我们利用 ImageList 组件为【例 6-8】中的 TreeView 组件添加图标。

【例 6-9】ImageList 组件应用练习

为 TreeView 组件中的项目树的每个项目分别添加图标。程序效果如图 6-28 所示。

(1) 打开【例 6-8】的程序文件。

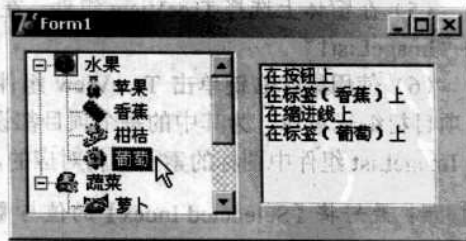


图 6-28 为 TreeView 组件添加图标

- (2) 向窗体中添加一个 ImageList 组件。
- (3) 双击该 ImageList 组件，打开一个图像列表对话框，如图 6-29 所示。

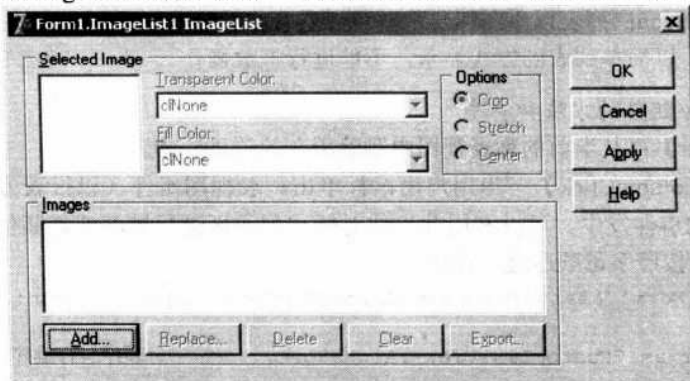


图 6-29 图像列表对话框

- (4) 单击 **Add...** 按钮，可以向组件中加入图像。Delphi 7 系统提供的图像资源一般位于 “C:\Program Files\Common Files\Boland Shared\Images” 文件夹中。这里我们引用了 “\Icons” 子文件夹中的几个小图标，如图 6-30 所示。

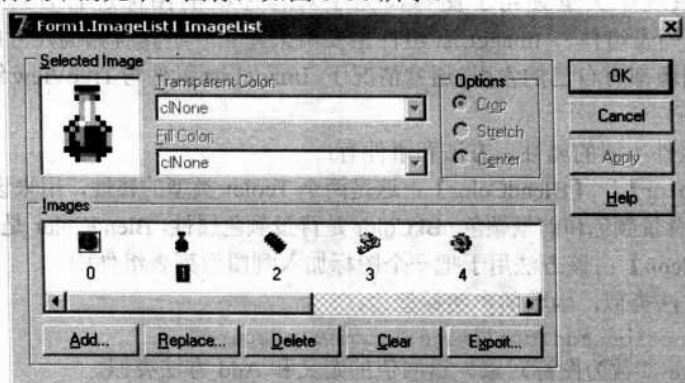


图 6-30 向组件中加入图像

提示：图标下面的值是图标对应的索引值，是系统自动添加的。

- (5) 在窗体上选择 TreeView 组件，在【Object Inspector】中，设置其【Images】属性为 “ImageList1”。

- (6) 使用鼠标右键单击 TreeView 组件，从快捷菜单中选择【Items Editor】命令，打开项目树编辑窗口，为其中的每个项目都设定【Image Index】（图像索引）值，这个值是与 ImageList 组件中图标的索引值相对应的，如图 6-31 所示。

提示：最好将【Selected Index】的值与【Image Index】相同，这样当用户选择项目时，项目的图标不会发生变化。当然，如果希望图标有一个变化，就应当选择不同的值。

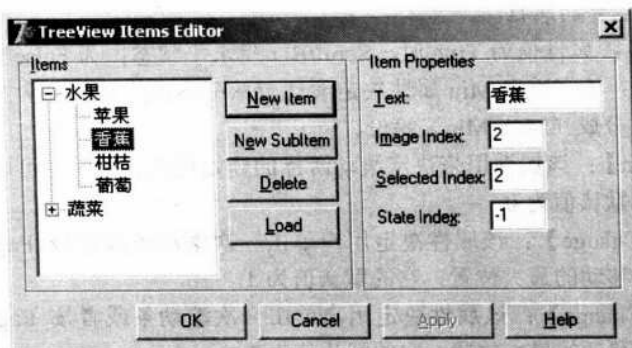


图 6-31 为每个项目都设定图像索引值

(7) 关闭对话框。运行程序，可以看到，每个项目都被添加了一个图标。

6.5 滚动组件

滚动组件包括 ScrollBar、TrackBar、UpDown、ProgressBar 和 ScrollBox 等组件，如图 6-32 所示。

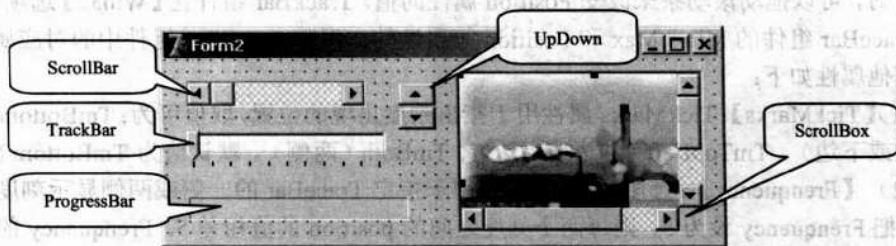


图 6-32 滚动组件

它们的具体用途如下：

- ScrollBar 组件为滚动条组件。如果一个窗口中的内容一次显示不完，可以通过滚动条浏览不同的内容。
- TrackBar 组件可以从一个连续的范围内选择一个数值。
- UpDown 组件中通过鼠标点击上下箭头，可以改变相关联的组件（如 Edit 组件）中的数值。
- ProgressBar 组件能够利用进度条动态地显示数据变化情况。
- ScrollBox 组件可以作为一个容器，使其中的内容在一个滚动的区域内显示。

6.5.1 ScrollBar 组件

ScrollBar 组件是最基本的滚动组件，它可以为没有滚动条的组件加上滚动控制，也可以让一个 ScrollBar 组件控制多个组件的滚动。ScrollBar 在【Standard】选项卡上。

1. ScrollBar 组件的属性

- (1) **【Kind】**: 属性值为 True 时, ScrollBar 呈水平状态; 为 False 时, 呈垂直状态。
- (2) **【Max】**: 该属性和 Min 属性决定滚动游标可移动的数值 (从 Min 到 Max), 它相当于自将滚动条分成 (Max-Min) 等份。
- (3) **【Position】**: 该属性用来决定滚动游标的开始位置。用户也可以改变其值以变更游标的位置。它的默认值为 0。
- (4) **【LargeChange】**: 该属性决定用户单击一次滚动条或者按 PageUP、PageDown 键时, 滚动游标能移动的最大位置。它的默认值为 1。
- (5) **【SmallChange】**: 该属性决定用户单击一次滚动条或者按 PageUP、PageDown 键时, 滚动游标能移动的最小位置。它的默认值为 1。

2. ScrollBar 组件的方法和事件

- (1) **【SetParams】**: 该方法可以完成滚动条的 Max、Min 和 Position 参数的设置, 利用它可以在运行中动态地调整滚动游标的位置。其方法原形如下:

```
procedure setparams(APosition,Min,APfax:Integer);
```

- (2) **【OnScroll】**: 当用户使用滚动条进行滚动时, 该事件发生。

6.5.2 TrackBar 组件

TrackBar 组件与 ScrollBar 组件相似, 它也有一个滚动条, 且两侧还可以显示刻度。程序运行时, 可以拖动滚动条来改变 Position 属性的值。TrackBar 组件在【Win32】选项卡中。

TraceBar 组件的 Min、Max 和 Position 等属性的作用与 ScroHBar 组件中的对应属性相同, 其他属性如下:

- (1) **【TickMarks】**: TickMarks 属性用于指出刻度出现的位置, 取值可为: TmBottomRight (右边或下边)、TmTopLeft (左边或上边)、TmBoth (两侧)。默认值为 TmBottomRight。
- (2) **【Frenquency】**: Frenquency 属性用于指定 TraceBar 的一侧或两侧显示刻度的间隔, 如把 Frenquency 设为 5, 则每两个刻度之间的 position 的值相差 5。Frenquency 的默认值为 1。
- (3) **【LineSize】**: LineSize 属性用于指出按下 Pageup 和 PageDown 键时 Position 值的变化量。LineSize 属性的默认值为 1。
- (4) **【Orientation】**: Orientation 属性用于指出 TraceBar 的摆放方向, 如果设置为 trHorizontal, 则 TraceBar 为水平方向; 如果设置为 trVertical, 则为垂直方向。Orientation 属性的默认值为 trHorizontal。
- (5) **【SelStart】**和**【SelEnd】**: SelStart 和 SelEnd 属性分别定义 TrackBar 中的选择区域。在 TrackBar 选择区域以深色显示, 同时刻度中出现特殊的标记。
- (6) **【SliderVisible】**: SliderVisible 属性用于决定滚动条是否可见。如果设置为 False 则滚动条不可见。SliderVisible 属性的默认值为 True。

TrackBar 组件的主要事件为 OnChange, 其用法与 ScrollBar 组件的相应事件用法相同。

6.5.3 UpDown 组件

UpDown 组件使用向上、向下的按钮来设置连续的整数。例如将 UpDown 与一个 Edit

组件相关联, 相当于一个 SpinEdit 组件的功能。程序运行时, 当用户单击向上、向下按钮时, Edit 中的值是连续变化的。UpDown 组件位于【Win32】选项卡中。

1. UpDown 组件的主要属性

UpDown 组件的 Min、Max 和 Position 等属性的作用与 ScrollBar 组件中的对应属性相同。其他属性如下:

(1) **【Associate】**: 该属性是 TWinControl 类型, 它用于指定 UpDown 组件依附的组件类, 运行时 UpDown 组件将自动出现在 Associate 属性指定的左侧或右侧 (由 AlignButton 决定), 且高度与其相同。

例如, Associate 属性被设置为一个 Edit 组件, 当 UpDown 组件的 Position 属性值变化时, 它会自动转化为字符串显示在 Edit 组件中。

(2) **【AlignButton】**: 该属性用于控制 UpDown 组件与其所依附的组件的相对位置。如果设置为 udLeft, 则 UpDown 出现在其所依附组件的左侧; 如果设置为 udRight, 则出现在所依附组件的右侧。AlignButton 属性的默认值为 udRight。

(3) **【Increment】**: 该属性用于控制当单击按钮时, Position 增减的数量。Increment 属性的默认值为 1。

(4) **【Orientation】**: 该属性用于决定 UpDown 组件两个按钮的指向。如果设置为 udHorizontal, 则两个按钮按水平排列, 箭头分别指向左右方向; 如果设置为 udVertical, 则按钮按垂直排列, 箭头分别指向上下方向。Orientation 属性默认值为 udVertical。

(5) **【Thousands】**: 该属性是一个布尔类型的属性。如果设置为 True, 则当 Position 大于或等于 1000 时, 将每 3 位插入一个逗号分隔符。Thousands 属性的默认值为 True。

(6) **【Wrap】**: 该属性也是一个布尔类型的属性。如果设置为 True, 则 Position 将在 Min 和 Max 之间循环变化, 即当 Position 为 Min 时, 再单击向下按钮则 Position 将变为最大值; 当 Position 为 Max 时, 单击向上按钮将使 Position 值变为最小值。如果 Wrap 设置为 False, 则当 Position 为 Min 和 Max, 分别单击向下和向上按钮时 Position 没有变化。Wrap 属性的默认值为 False。

(7) **【ArrowKeys】**: 该属性用于设定单击上、下方向键时是否自动增加和减少 Position 的值。ArrowKeys 属性默认值为 True。

2. UpDown 组件的主要事件

(1) **【OnChanging】**和**【OnChangingEx】**: 单击 UpDown 按钮时, 先触发 OnChanging 事件, 再触发 OnChangingEx 事件。OnChanging 事件的声明为:

```
procedure Changing(Sender:TObject;var AllowChange:Boolean);
```

其中, 当设置 AllowChange 参数为 False 时, Position 的值不变化。AllowChange 的默认值为 True。

OnChangingEx 事件的声明为:

```
procedure ChangingEx(Sender:TObject; var AllowChange:Boolean;
    NewValue:Smallint; Direction:TUpDownDirection);
```

其中, AllowChange 参数同上, NewValue 参数是设置的新值, Direction 参数识别向上或向下的按钮。

(2) **【OnClick】**: 单击 UpDown 按钮时, 上述两个事件执行完后, 再触发 OnClick 事

件。该事件的声明为:

```
procedure Click(Sender:TObject; Button:TUDBtnType);
```

其中当 Button 参数取值为 btNext 时,表示单击的是向上按钮,取值为 btPrev 时,则表示单击的是向下按钮。

6.5.4 ProgressBar 组件

ProgressBar 组件能够动态地显示数据值的变化,如工作完成的进度等。其特殊的属性和方法如下:

(1) **【Smooth】**: 属性,决定进度条是以连续的色条来表示数据的变化还是以一个个分离的色块来表示。

(2) **【Step】**: 属性,定义了进度条前进的步长。

(3) **【StepIt】**: 方法,使进度条每次前进一个步长,该步长值由 **【Step】** 属性定义。

(4) **【StepBy】**: 方法,使进度条按照函数参数定义的步长来变化。

下面通过一个简单的程序说明这几个组件的用法。

[例 6-10] 滚动组件的使用

利用 ScrollBar、TrackBar 和 UpDown 组件来调整文本框中的数值,同时这种变化会体现在 ProgressBar 中。程序效果如图 6-33 所示。

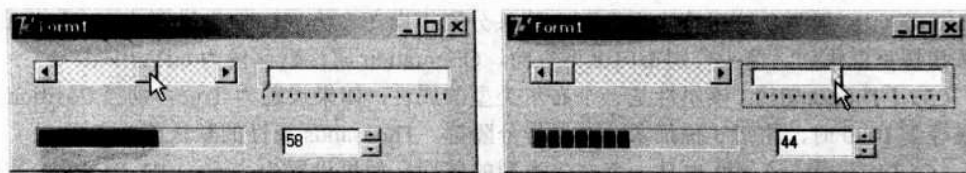


图 6-33 滚动组件的使用

(1) 创建一个新的应用程序,适当调整窗体的大小。

(2) 在窗体上添加 ScrollBar、TrackBar、UpDown 和 ProgressBar 组件,以及一个 Edit 组件。

(3) 设置 UpDown 组件的 **【Associate】** 属性为 "Edit1",即选择文本框作为 UpDown 组件的依附对象。可见,UpDown 组件会自动附着在文本框的右侧。

(4) 为 ScrollBar、TrackBar 和 Edit 组件的 OnChange 事件添加代码,如下所示:

```
... //为节约篇幅,前面由系统自动生成的代码被省略
implementation
{$R *.dfm}

// ScrollBar 的 OnChange 事件
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
  progressbar1.Smooth:=true; //设置进度条为连续色条方式
  //使编辑框显示 ScrollBar 的位置值
  edit1.Text:=inttostr(scrollbar1.Position);
end;
```



```

// TrackBar 的 OnChange 事件
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
    progressbar1.Smooth:=false;           //设置进度条为分离色块方式
    edit1.Text:=inttostr(trackbar1.Position);
end;

//Edit 组件的 OnChange 事件
procedure TForm1.Edit1Change(Sender: TObject);
begin
    //利用进度条表示编辑框中的数值情况
    progressbar1.Position:=strtoint(edit1.Text);
end;
end.

```

提示：由于 UpDown 组件已经与 Edit 组件绑定在一起，所以不需要为 UpDown 组件设置任何代码，直接就可以调整 Edit 组件中的数值。

(5) 运行程序，我们可以利用 ScrollBar、TrackBar 和 UpDown 组件来调整文本框中的数值，同时这种变化会体现在 ProgressBar 中。

6.5.5 ScrollBox

ScrollBox 组件提供一个可以滚动的区域，在该区域中放置其他组件，当组件的边界超过区域的边界时，ScrollBox 组件会自动产生滚动条。如 Image 组件就没有滚动条，可以在窗体上添加次 ScrollBox 组件，再添加 Image 组件放在 ScrollBox 中，产生滚动效果。

6.6 表格组件

Delphi 7 有一类表格组件，即网状图形表格，包括【Additional】选项卡上的 StringGrid 组件、DrawGrid 组件和【Samples】选项卡上的 ColorGrid 组件，以及【Data Controls】选项卡上的 DBGrid 组件。DBGrid 组件用于显示数据库中的数据内容，其用法将在后面数据库部分讲解。

6.6.1 StringGrid 组件

StringGrid 组件可以将字符串列成表格，而且其本身可存储数据。就本质而言，StringGrid 组件对应的类型 TStringGrid 是 TDrawGrid 类的一个子类。

StringGrid 组件常用属性如下：

- **【ColCount】、【RowCount】**：表格的列数、行数。
- **【FixedCols】、【FixedRows】**：表格的固定列数、行数。固定的行列常用作标题。
- **【FixedColor】**：表格固定行列的颜色。
- **【GridLineWidth】**：表格各单元格分隔线的线宽。
- **【Options】**：有 15 个子属性，用以在运行时操作表格。如 goColSizing 和 goRowSizing 子属性定义是否可以调整行列尺寸。goRowMoving 和 goColumnMoving 子属性定义是否允许用户将整行或整列移动到新位置。

下面我们用一个简单的程序来说明如何使用 StringGrid 组件。

[例 6-11] StringGrid 组件应用练习

用一个表格显示中文区位表中的字符。程序效果如图 6-34 所示。



图 6-34 用表格显示中文区位表

(1) 创建一个新的应用程序。

(2) 向窗体中添加一个 StringGrid 组件，设置组件的【Align】属性为“alClient”。这样，表格就会占据整个窗体。

(3) 在窗体的 OnCreate 事件中添加如下代码：

```
... //为节约篇幅，前面由系统自动生成的代码被省略
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var
  i, j: integer;
begin
  StringGrid1.ColCount:=127;           //设置表格 127 列
  StringGrid1.RowCount:=90;           //设置表格 90 行
  for i:=0 to StringGrid1.ColCount-1 do
  begin
    StringGrid1.Cells[i,0]:= inttostr(i); //为表格第一列设置序号
    //定义各列的宽度
    StringGrid1.ColWidths[i]:=StringGrid1.Canvas.TextWidth('宽度');
  end;
  for j:=0 to StringGrid1.RowCount-1 do
  begin
    StringGrid1.Cells[0,j]:=inttostr(j); //为表格第一行设置序号
  end;
  for j:=1 to StringGrid1.RowCount do
  for i:=1 to StringGrid1.ColCount do
    //在各单元格中显示区位表的字符
    StringGrid1.Cells[i,j]:=Chr(j+160)+Chr(i+160);
  end;
end;
```

end.

提示：代码中用到的数值，是根据区位表的特点来设置的。读者不需刻意琢磨，只需理解对 StringGrid 组件的使用方法。

(4) 运行程序，可见区位表的内容会准确地显示在表格中。

6.6.2 DrawGrid 组件

DrawGrid 组件既能够显示字符串，又能够显示图形。它与 StringGrid 组件基本相似，但是 DrawGrid 组件不能自己开辟 Cells 数组来存储各单元格数据的值。

DrawGrid 组件有一个 Boolean 型的 DefaultDrawing 属性，当该属性为 True 时，绘制表格的同时将绘制单元格；当该属性为 False 时，单元格需要另外进行绘制。

DrawGrid 组件常用事件及方法如下：

- **【CellRect】**：方法，用于指定单元格。
- **【Canvas.TextOut】**：方法，用于在指定单元格输出文本。
- **【Canvas.DrawFocusRect】**：方法，用来绘制聚焦框。
- **【OnGetEditText】**：事件，用来获取鼠标所在单元格的文本。

6.6.3 ColorGrid 组件

ColorGrid 组件是一个颜色表格组件，常用于制作调色板。其常用属性有：

- **【ForegroundIndex】**：属性，前景色，用鼠标左键指定。
- **【BackgroundIndex】**：属性，背景色，用鼠标右键指定。
- **【GridOrdering】**：属性，指定表格的行数和列数。

6.7 日期和时间组件

Delphi 7 中提供了 3 个图形化的日历组件：位于【Win32】选项卡上的 DateTimePicker 组件和 MonthCalender 组件、位于【Samples】选项卡上的 Calender 组件。另外，我们还要介绍一下 Timer 组件。

6.7.1 DateTimePicker 组件

DateTimePicker 组件用于处理 DDateTime 组件类型的日期和时间数据，如图 6-35 所示。



图 6-35 DateTimePicker 组件

- **【Date】**: 组件的日期值, 如 2004-6-1。
- **【Time】**: 组件的时间值, 如 10:10:10。
- **【Kind】**: 组件的显示类型。当 Kind 值为 dtkDate 时, DateTimePicker 显示日期, 当 Kind 值为 dtkTime 时, DateTimePicker 显示时间。默认值为 dtkDate。

主要属性有:

- **【DateFormat】**: 日期的显示格式。取值为 dfLong, 则以四位年份的长日期格式表示, 如“2004 年 6 月 20 日”; 取值为 dfShort, 则使用为两位年份的短日期格式, 如“04-6-20”、“16: 08: 16”等。

- **【DateMode】**: 日期的显示形式。取值为 dmAutomatic, 则日期为当前日期, 不可修改; 若取值为 dmManual, 则可以选择不同的日期。

- **【MaxDate】与【MinDate】**: 日期变化的最大与最小值。

要读取 DateTimePicker 组件的日期与事件, 直接读取 DateTimePicker 组件的 Date 属性和 Time 属性即可。下面是两个常用的日期转换函数:

- DateToStr 函数: 将日期 Date 转换为字符串。
- TimeToStr 函数: 将 Time 转换为字符串。

6.7.2 MonthCalendar 组件

MonthCalendar 以月历方式显示日期, 如图 6-36 所示。可见, MonthCalendar 组件相当于 DateTimePicker 组件的日期部分。

主要属性有:

- **【MultiSelect】与【EndDate】**: 这两个属性实现 MonthCalendar 的多选功能。当 MultiSelect 设为 True 运行时, 按住 shift 键可选择连续多个日期值, 起始日期在 Date 属性中, 结束日期在 EndDate 属性中。MultiSelect 的默认值为 False。

- **【ShowToday】与【ShowTodayCircle】**: 这两个属性与组件的 Today 值的显示方式有关。当 ShowToday 为 False 时, 不特别标注 Today 的日期; 当 ShowTodayCircle 为 False 时, Today 值没有红色图标标注。两者默认值均为 True。

- **【FirstDayOfWeek】**: 决定 MonthCalendar 表格以星期几开始排列。默认值为 dowLocaleDefault, 表格从星期日开始。当 FirstDayOfWeek 取 dowMonday, 表格以星期一开始, 依次类推。

- **【WeekNumbers】**: 当 WeekNumbers 为 True 时, 显示周数列表。WeekNumbers 的默认值为 False。



图 6-36 MonthCalendar 组件



图 6-37 Calendar 组件

6.7.3 Calendar 组件

Calendar 能够以星期表格方式显示日期, 如图 6-37 所示。但是它无法直观地给出当前年月的说明。

主要属性有:

- **【GridLineWidth】**: 属性, 表示网格线的宽度。
- **【StartOfWeek】**: StartOfWeek 决定 Calendar 表格以星期几开始排列。默认为 0, 表格从星期日开始。当 StartOfWeek 为 1 时, 表格从星期一开始, 依次类推。
- **【UseCurrentDate】**: 属性, 决定 Calendar 的时间。当取值为 True 时, Calendar 组件采用操作系统的系统时间; 当取值为 False 时, Calendar 的时间由用户决定。

用户在程序设计时, 指定号 Calendar 的属性后, 在运行时就可以通过 Calendar 组件来读取时间, 如通过 Year、Month、Day 属性来读取年、月、日。

6.7.4 Timer 组件

Timer (定时器) 是一个非可视化组件, 它能够定时地触发 OnTimer 事件, 从而可以完成模拟时钟、系统延时等工作。Timer 组件位于 **【System】** 选项卡上。

Timer 的主要属性和事件有:

- **【Enabled】**: 该属性控制定时器是否打开。当 Enabled 值为 True 时, 定时器开始工作; 值为 False 时, 定时器停止工作。Enabled 的默认值为 True。
- **【Interval】**: 该属性控制 Timer 触发事件的时间间隔, 单位是毫秒。将 Interval 为 0, 相当于关闭定时器。Interval 的默认值为 1000, 即 1 秒。

Interval 组件只有一个事件 OnTimer, 当 Timer 打开时, 每经过 Interval 指定的时间就会触发 OnTimer 事件, 并自动执行 OnTimer 事件中的程序。

下面我们用一个实例说明 Timer 组件的用法。

[例 6-12] 滚动的字幕

利用 Timer 组件使 Edit 组件中的文本不断循环移位, 产生滚动字幕效果; 使用 SpinEdit 组件来控制字幕的移动速度。程序效果如图 6-38 所示。

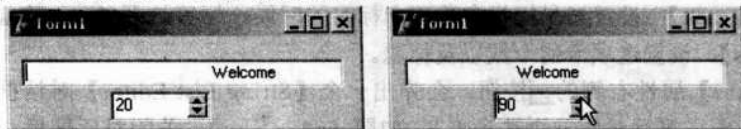


图 6-38 滚动的字幕

- (1) 创建一个新的应用程序。
- (2) 在窗体上放置一个 Edit 组件, 设置其 **【Text】** 属性为 “Welcome”。
- (3) 再添加一个 Timer 组件和一个 SpinEdit 组件, 如图 6-39 所示。

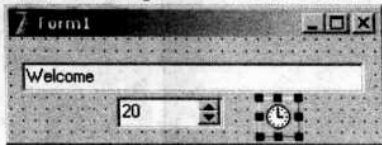


图 6-39 添加组件

- (4) 设置 SpinEdit 组件的 **【Value】** 属性为 20, **【Increment】** 属性为 10。
- (5) 设置 Timer 组件的 **【Interval】** 属性为 100。
- (6) 在 Timer 组件的 OnTimer 事件中输入如下代码, 使 Edit1 中的文本字符循环移位。

```

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    edit1.Text:=copy(edit1.Text,length(edit1.Text)-1,1)
                +copy(edit1.Text,1,length(edit1.Text)-1);
end;

```

(7) 在 SpinEdit 组件的 OnChange 事件中添加如下代码, 设置当 SpinEdit 的值改变时, 新值将作为 Timer 组件新的时间间隔。

```

procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
    timer1.Interval:=spinedit1.Value;
end;

```

(8) 运行程序, 可见文本在滚动。

但是文本只在自身长度范围内滚动, 这是由于程序代码定义了仅复制字符长度的内容。一个简单的解决方法是, 在 Edit1 的文本字符后面添加多个空格, 这样就能够看到字幕的滚动效果了。

6.8 多选项卡组件

多选项卡组件可以在有限的空间中尽可能多地存放信息, 而且便于对信息进行分类管理。例如常见的属性面板或选项面板, 往往都是采用多选项卡的样式。

6.8.1 TabControl 组件

TabControl 组件能够给窗体添加标签供用户选择, 它适合在外观上不变而内容发生变化的多页应用程序中使用。TabControl 组件位于【Win32】选项卡上。

主要属性有:

- **【HotTrack】**: 设置当鼠标指向选项卡标签的时候, 标签是否自动加亮显示。
- **【Tabs】**: 设置选项卡的个数以及标题。

单击**【Tabs】**属性右侧的按钮, 会弹出一个**【String List Editor】**对话框, 在其中输入选项卡的标题(每行对应一个选项卡), 如图 6-40a 所示。关闭对话框后, 就可以看到 TabControl 组件显示了多个选项卡, 如图 6-40b 所示。

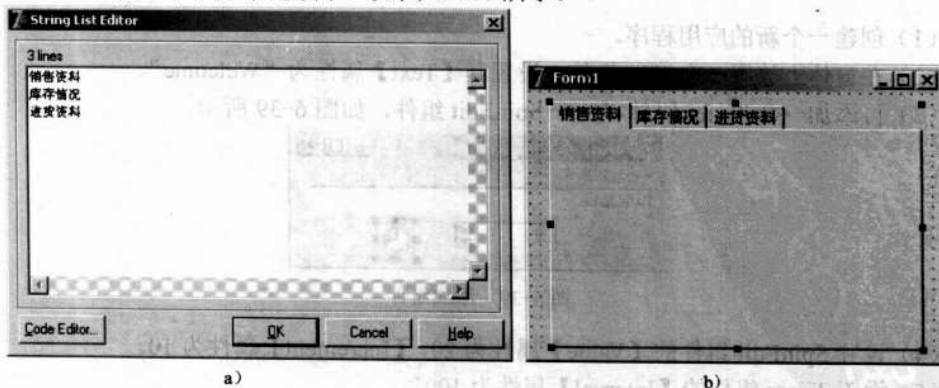


图 6-40 TabControl 组件

TabControl 组件有两个比较重要的事件: OnChange 和 OnChanging:

- 每当一个选项卡被选择时就会发生 OnChange 事件, 可以根据 TabIndex 属性来判断哪一个选项卡被选中。
- OnChanging 事件正好发生在选项卡被选中之前, 也就是正好在一个选项卡切换到另一个选项卡之前。这个事件使用户可以在选项卡切换之前采取某些行动, 例如在某些条件没有设置之前不能离开该选项卡。

6.8.2 PageControl 组件

和 TabControl 组件相比, PageControl 组件功能更加强大, 它的每个选项卡都是一个 TTabSheet 组件, 是一个容器类组件, 可以在它上面放置各种组件。

注意 PageControl 的选中状态的不同:

- 单击 PageControl 标题时, 选中的是 PageControl 组件对象;
- 单击页面时, 选中的是 TabSheet 对象。

选中什么对象, “Object Inspector” 中显示的就是该对象的属性。PageControl 与 TabSheet 分别有各自不同的属性。

PageControl 的主要属性:

- **【ActivePage】**: 设置 PageControl 组件的当前页。当用户切换页面时, ActivePage 属性的值会相应改变。
- **【Multiline】**: 值为 true 时, PageControl 组件呈现多行风格。
- **【Images】**: Images 属性表示与 PageControl 相关联的 ImageList 组件的名称。选定之后, PageControl 就可使用 ImageList 组件中的图像。
- **【Pages】**: PageControl 所有页面记录在 Pages 属性中。Pages 是运行时的属性, 即设计时在 “Object Inspector” 中不可见, 而运行却能在程序中引用。Pages 是一个数组, 其元素类型为 TTabSheet, 元素个数放在 PageCount 中, Pages[0] 是第一页, PPages[PageCount-1] 是最后一页。
- **【ActivePageIndex】**: 当前活动页的 Pages 属性的下标, 从 0 开始计数。ActivePageIndex 也是运行时的属性。

下面我们用一个实例来说明日期类组件和多选项卡组件的使用。

【例 6-13】多选项卡日历

窗体上有一个多选项卡组件, 对不同页面上日期的选择能够保持同步, 而且下面的文本框能够显示当前选择的选项卡页面标题。

程序效果如图 6-41 所示。

(1) 创建一个新的应用程序。

(2) 向窗体中添加一个 ImageList 组件, 双击组件, 打开图像列表对话框, 在其中添加两个图标, 如图 6-42 所示。这两个图标都来自于 “C:\Program Files\Common Files\Boland Shared\Images\button” 目录。

(3) 在窗体上添加一个 PageControl 组件。在组件上单击鼠标右键, 从快捷菜单中选择 **【New Page】** 命令, 为组件添加两个选项卡 TabSheet1 和 TabSheet2, 并分别定义选项卡的 **【Caption】** 属性为 “星期” 和 “月历”。

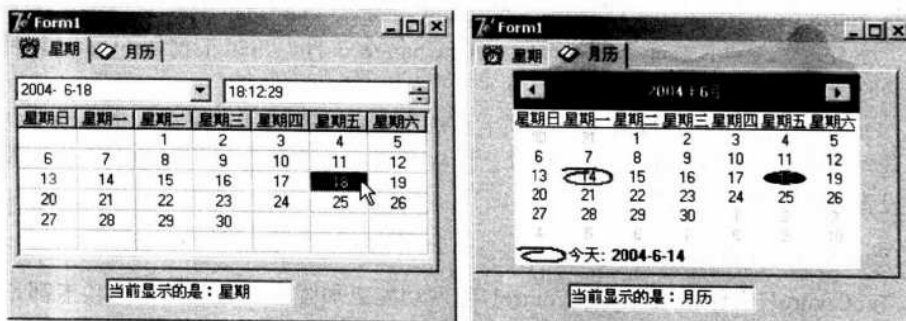


图 6-41 多选项卡日历

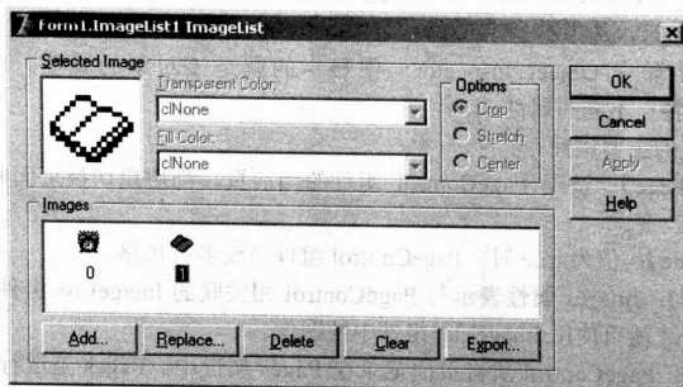


图 6-42 在 ImageList 组件中添加图标

(4) 设置 PageControl 组件的【Image】属性为“ImageList1”，则 PageControl 组件的各个选项卡就分别有一个图标显示了。

(5) 在窗体上添加一个 Edit 组件。

(6) 选择 TabSheet1 对象，在其中添加两个 DateTimePicker 组件和一个 Calendar 组件，并设置第二个 DateTimePicker 组件的【Kind】属性为“dtkTime”。这样，第一个 DateTimePicker 组件显示的是日期，而第二个 DateTimePicker 组件显示的是时间。

(7) 选择 TabSheet2 对象，在其中添加一个 MonthCalendar 组件。

(8) 选择窗体 Form1，在其 OnCreate 事件中添加如下代码，定义初始状态下，日历组件显示的是系统当前日期和时间。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    DateTimePicker1.Date:=Date;
    DateTimePicker2.Time:=Time;
    MonthCalendar1.Date:=Date;
end;
```

(9) 在 PageControl 的 OnChange 事件中添加如下代码，定义当用户切换页时，将当前

页的标签显示在 Edit1 组件中。

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
    edit1.Text:='当前显示的是: '+PageControl1.ActivePage.Caption;
end;
```

(10) 为使三个日期类组件所显示的日期一致, 在各个组件的 OnChange 事件中添加如下代码:

```
// DateTimePicker1 组件发生改变
procedure TForm1.DateTimePicker1Change(Sender: TObject);
var
    y,m,d:word;
begin
    DecodeDate(DateTimePicker1.Date,y,m,d);
    Calendar1.Day:=d;
end;

// Calendar1 组件发生变化
procedure TForm1.Calendar1Change(Sender: TObject);
var
    y,m,d:word;
begin
    DecodeDate(DateTimePicker1.Date,y,m,d);
    d:=Calendar1.Day;
    dateTimePicker1.Date:=EncodeDate(y,m,d);
end;

//用户切换页面
procedure TForm1.PageControl1Changing(Sender: TObject;
    var AllowChange: Boolean);
var
    y,m,d:word;
begin
    case pageControl1.ActivePageIndex of
        0:MonthCalendar1.Date:=DateTimePicker1.Date;
        1:begin
            DateTimePicker1.Date:=MonthCalendar1.Date;
            decodeDate(DateTimePicker1.Date,y,m,d);
            Calendar1.Day:=d;
        end;
    end;
end;
```

(11) 运行程序, 页面日期的改变会保持同步。

6.9 小结

本章介绍了一些常用组件的属性、方法和事件, 并通过一些具体的实例说明了组件的用法。总的说来, Delphi 7 的组件可分为四种基本类型:

- 命令型组件——用于激活动作、发出命令。如按钮等。
- 显示型组件——用于显示数据。如标签、编辑框、列表框、组合框等。
- 选择型组件——用于列出选项供选择。如单选按钮、复选框等。
- 装饰型组件——用于装饰需求的组件。如位图按钮、图像列表等。

Delphi 7 为用户提供了大量组件，这里不可能一一讲解，大家可以在联机文档的帮助下慢慢学习和理解。



第7章 用户界面设计

界面是应用程序与用户进行交流的窗口，是每个应用程序必备的部分。友好的界面是构成一个优秀软件的基本要素，通过各种窗体或对话框，用户能够轻松地与程序进行交互。本章讲述了如何创建不同形式的界面，制作各种菜单、工具栏和对话框，并通过设计一个文档编辑器程序，使大家对于如何利用 Delphi 7 进行界面设计有一个基本的认识。

7.1 菜单设计

菜单是 Windows 应用程序最常用的交互方式。菜单是一些分类的命令列表，它一般分为下拉式菜单和弹出式快捷菜单。下拉式菜单的主菜单栏一般位于应用程序窗体的标题栏之下；而快捷菜单一般是通过单击鼠标右键弹出，可以出现在程序的任何地方。在 Delphi 7 系统中就可以看到这两种类型的菜单。

菜单中的每一项命令被称为菜单项。在 Delphi 7 中菜单项都是 `TmenuItem` 类的一个对象，但是该组件并不出现在运行时的窗体中。创建菜单可以通过菜单编辑器来实现。Delphi 7 提供了可视化的组件 `TMainMenu` 和 `PopupMenu` 来创建下拉式菜单和弹出式快捷菜单。另外，也可以通过程序代码实现动态菜单设计。

在设计菜单过程中，要注意适当分类，将相关的编辑或操作命令组织到一个菜单中；对于使用频率较高的命令，可以放到快捷菜单中来实现。

另外，许多菜单都设有加速和热键。加速键就是用户可以通过按住 **Alt** 键和菜单项中带有下划线的字母来打开菜单，如使用 **Alt+F** 来打开【文件】菜单；而热键，又称快捷键，是使用一个功能键加某一个字母键的组合来直接执行该菜单命令，如使用 **Ctrl+N** 来新建一个文件。


7.1.1 下拉式菜单

在 Delphi 中，一般使用菜单设计器来设计下拉式菜单。下面用一个简单的例子来说明下拉式菜单设计方法。

【例 7-1】设计下拉式菜单

程序效果：设计一个简单的下拉式菜单，各菜单都有加速键，某些菜单项还有子菜单；单击菜单命令，会在一个标签对象中显示出选择的命令。程序画面效果如图 7-1 所示。

(1) 在 Delphi 7 中新建一个应用程序，适当调整窗体的大小。

(2) 从组件面板的【Standard】选项卡中双击  组件，则该组件在窗体中建立了一个 `MainMenu` 对象，如图 7-2 所示。

(3) 双击窗体中的 MainMenu 对象，能够打开一个菜单设计窗口，如图 7-3 所示。此时窗口已经显示了第一个空白菜单项。只需要输入标题就可以建立菜单。

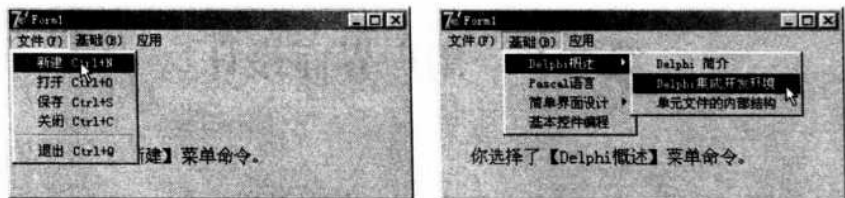


图 7-1 下拉式菜单

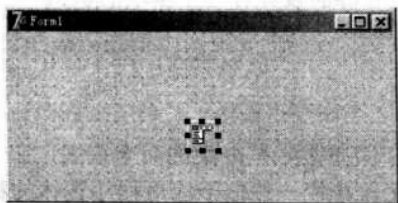


图 7-2 在窗体中建立了一个 MainMenu 对象

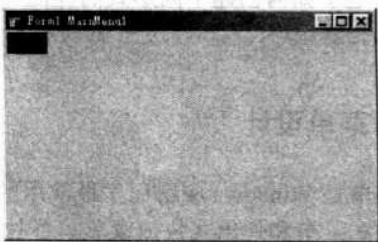


图 7-3 菜单设计器

(4) 在【Object Inspector】窗口的【Caption】栏输入当前菜单的标题，如“文件(&F)”，然后按下 **Enter** 键，菜单设计器就会自动建立一个【File】菜单，并将输入焦点移动到下一个菜单位置，允许继续建立其他菜单。如图 7-4 所示。

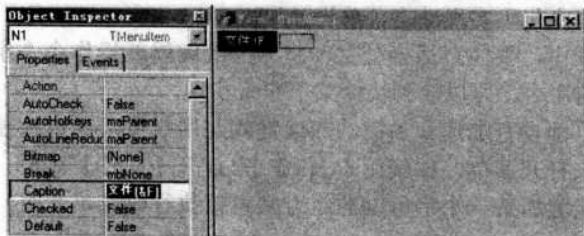


图 7-4 建立第一个菜单

提示：为菜单定义了加速键，就可以使用 **Alt + F** 来访问该菜单。

(5) 单击菜单项【File】，会进入其二级子菜单的编辑状态，而且【Object Inspector】窗口会自动显示新菜单项的属性；直接输入菜单项的标题，就能够建立一个子菜单项，如图 7-5 所示。

提示：在【Caption】中输入一个“-”号就能够产生菜单的分隔线，这种分隔线一般用于菜单项的分组。

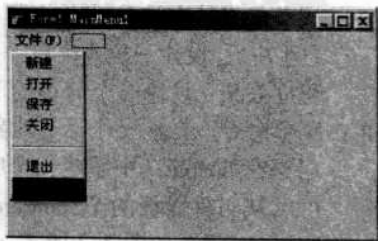


图 7-5 建立子菜单

(6) 这时, 从【Object TreeView】窗口可以看到当前建立的各种菜单对象, 如图 7-6 所示。这里按照菜单和菜单项的归属关系以树形结构列出了各个对象, 显示了对对象的标题文字 (Caption) 和名称 (Name)。

(7) 为菜单项设置热键: 选择【新建】菜单项, 在【Object Inspector】窗口, 选择【ShortCut】属性; 单击其右侧的下拉按钮, 能够打开一个列表, 其中罗列了许多热键组合, 如图 7-7 所示。



图 7-6 【Object TreeView】窗口



图 7-7 为菜单项设置热键

提示: 如果不定义菜单项的【Name】(名称) 属性, 系统会自动以“N”加顺序的数字来定义, 如 N1、N2、N3...等。

(8) 选择“Ctrl+N”热键组合, 就为【新建】菜单项定义了热键。

(9) 同理, 再为其他几个菜单项添加热键, 如图 7-8 所示。

(10) 再按照程序效果要求创建【基础】、【应用】两个菜单, 并为它们添加必要的菜单项。

(11) 选择【基础】菜单的【Delphi 概述】菜单项, 单击鼠标右键, 出现一个快捷菜单, 如图 7-9 所示。

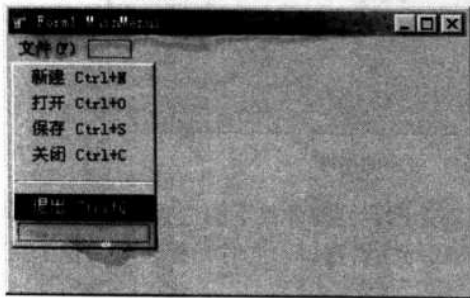


图 7-8 为各菜单项添加热键

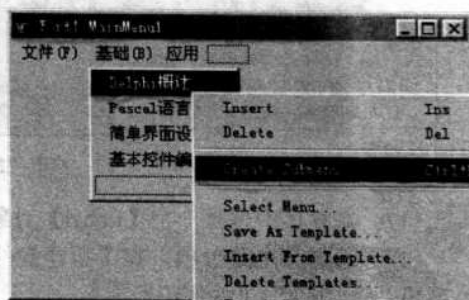


图 7-9 菜单设计器的快捷菜单

(12) 从快捷菜单中选择 Create Submenu 命令, 就会在【Delphi 概述】菜单项旁边出现一个带有空白菜单项的二级子菜单, 如图 7-10 所示。

(13) 子菜单项的建立方法同上, 不再赘述。使用同样的方法建立其他二级子菜单。

(14) 关闭菜单设计器, 可以看到现在窗体上已经有了一排菜单, 如图 7-11 所示。

(15) 现在执行程序, 可见菜单已经可以使用。但是, 大家也会发现, 菜单对象自动为每一个没有定义加速键的菜单项添加了不同的加速键, 如图 7-12 所示。

(16) 关闭程序。在【Object Inspector】窗口中选择“MainMenu1”对象，设置其【AutoHotkeys】属性为“maManual”，如图 7-13 所示。这样，菜单对象就不会自动为菜单项添加加速键了。

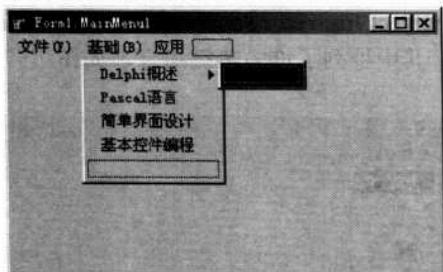


图 7-10 建立二级子菜单

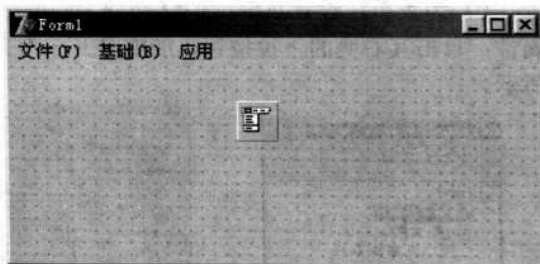


图 7-11 菜单已经在窗体中产生

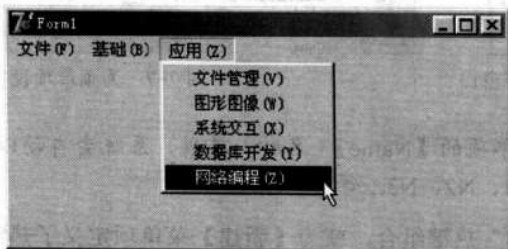


图 7-12 菜单对象自动添加了加速键



图 7-13 修改“MainMenu1”对象的属性

(17) 在窗体中建立一个【Label】对象，调整对象位置到窗体靠下的位置。

(18) 选择某个菜单项，如【新建】，则会打开代码编辑窗口，同时光标自动定位于菜单项的“OnClick”事件代码段内。

(19) 在其中输入如图 7-14 所示的代码，定义单击菜单项后，改变标签对象 Label1 显示的文字内容。

(20) 同理，为其他菜单项也定义单击事件的代码内容，使标签对象显示相应的文字内容。

(21) 在菜单项【退出】的单击事件中，输入代码“close;”，以结束程序。如图 7-15 所示。

(22) 执行程序，可见当选择某个菜单项后，标签对象就会显示相应的命令内容；使

用热键也能够达到相同的操作效果。



图 7-14 单击菜单项后的代码内容

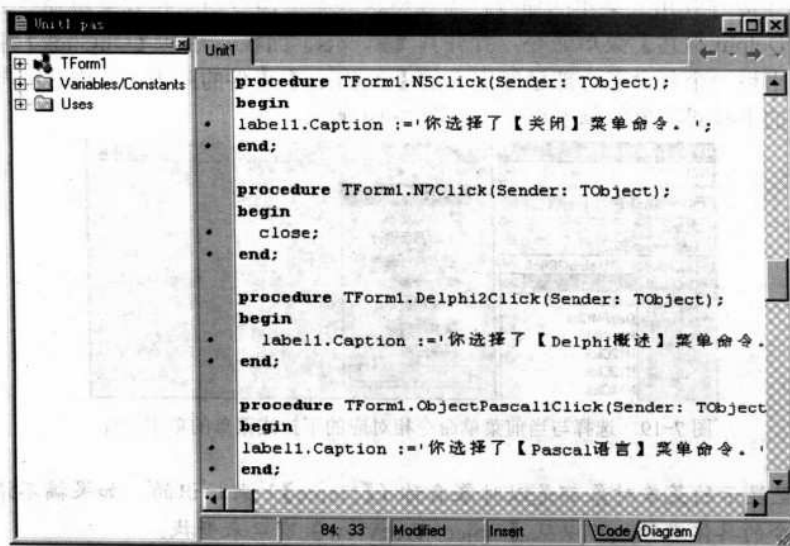


图 7-15 为其他菜单项定义单击事件的代码内容

7.1.2 弹出式快捷菜单

弹出式快捷菜单与下拉式菜单的设计方法基本相同，只有组件对象与弹出式菜单链接的问题需要注意。下面继续利用前面的实例来设计一个弹出式菜单。

【例 7-2】简单的弹出式菜单

程序效果：在窗体中单击鼠标右键，会弹出一个快捷菜单；选择其中任何一个菜单项，也具有同下拉式菜单相同的功能。程序画面如图 7-16 所示。

(1) 在窗体中添加一个 PopupMenu 组件；双击该组件，打开菜单设计器，如图 7-17 所示。可见弹出式菜单设计器与下拉式菜单设计器基本相同。

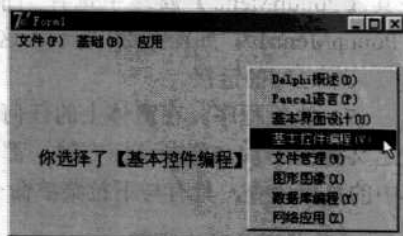


图 7-16 弹出式菜单

(2) 向弹出式菜单中添加菜单项的方法与下拉式菜单完全相同。把一些常用的菜单命

令组织到快捷菜单中,如图 7-18 所示。

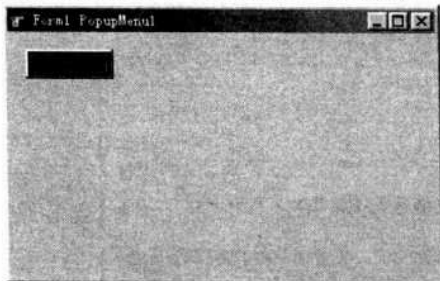


图 7-17 弹出式菜单编辑器

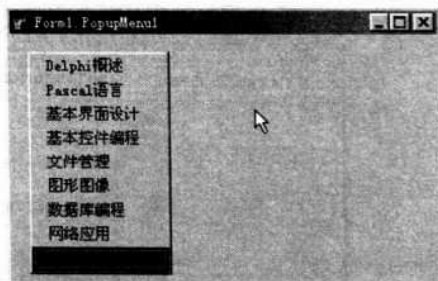


图 7-18 创建弹出式菜单

(3) 为了处理弹出式菜单的命令,需要将这些菜单命令与下拉式菜单命令之间建立联系。选择【Delphi 概述】菜单命令,打开其【Events】面板,在其【OnClick】事件的输入栏单击,会出现一个包含了全部已经定义的【OnClick】事件的列表。从中选择与当前菜单命令相对应的下拉式菜单的单击事件,如图 7-19 所示。

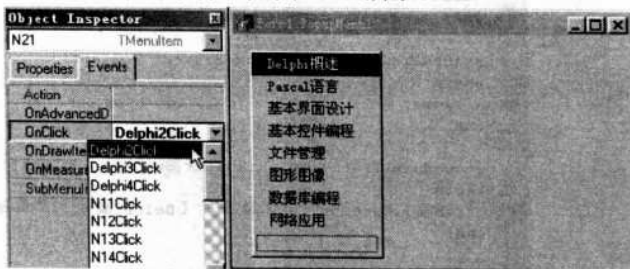


图 7-19 选择与当前菜单命令相对应的下拉式菜单的单击事件

提示: 这里显示的菜单对象都是以对象名称(【Name】)来标识的。如果搞不清楚菜单命令的具体名称,可以从【Object TreeView】窗口来查找。

(4) 同理,定义其他弹出式菜单项的单击事件。

(5) 关闭菜单编辑器。现在执行程序,你会发现无法打开弹出式菜单。这是因为还没有定义弹出式菜单与哪个对象链接。

(6) 在【Object Inspector】窗口中选择当前窗体对象“Form1”,在其【PopupMenu】属性中选择当前窗体中的弹出式菜单对象“PopupMenu1”,如图 7-20 所示。这样,就建立了当前窗体与弹出式菜单之间的链接。

(7) 执行程序。在窗体上的任何地方(菜单栏除外,因为没有定义链接关系)单击鼠标右键,都会弹出一个快捷菜单;选择其中的菜单命令,具有与下拉菜单命令同样的功能。

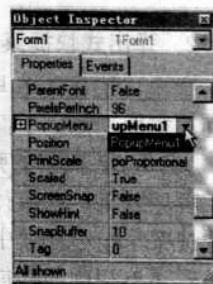


图 7-20 建立当前窗体与弹出式菜单之间的链接

7.1.3 菜单项的动态调整

在有些情况下,程序需要在运行时动态改变菜单的状态,如控制菜单项是否有效、设置选中标记、添减菜单项等。这些需求往往不能完全依靠属性的设置来完成,而需要利用

程序代码来实现。下面用一个实例来说明这种动态调整方法。这个实例涉及的内容较多，需要分几个步骤来完成。

【例 7-3】动态调整菜单项

1. 菜单项有效性控制和复选标记

程序效果：建立一个带有下拉式菜单和弹出式菜单的程序，弹出式菜单中的命令可以控制下拉菜单中的某个菜单项是否有效。同时，在弹出式菜单中选择某个命令后，该命令前面应有一个复选标记，如图 7-21 所示。

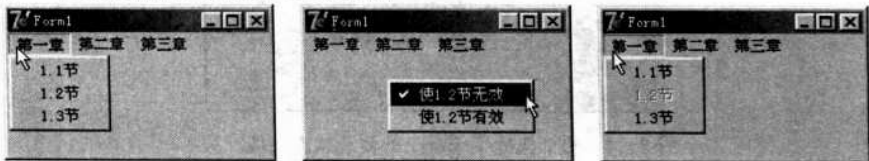


图 7-21 菜单项有效性控制和复选标记

- (1) 在 Delphi 7 中新建一个应用程序。在窗体中添加一个 MainMenu 菜单组件和 PopupMenu 菜单组件。
- (2) 建立如程序要求的下拉菜单和弹出式快捷菜单。
- (3) 设置弹出式快捷菜单各个菜单项的【AutoCheck】属性为“True”，如图 7-22 所示。

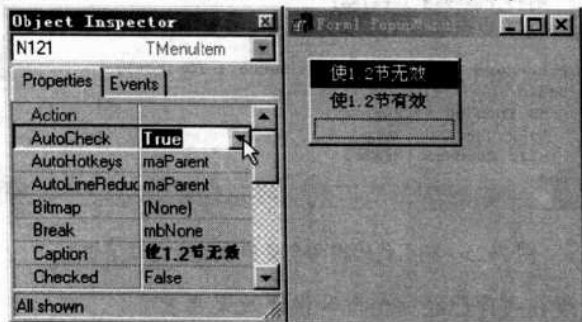


图 7-22 设置弹出式快捷菜单菜单项的【AutoCheck】属性

- (4) 双击快捷菜单的菜单项，在代码编辑器中输入如图 7-23 所示的程序代码，定义一个菜单命令是使菜单对象 N3（菜单项“1.2 节”）无效；另一个菜单命令是使 N3 有效。

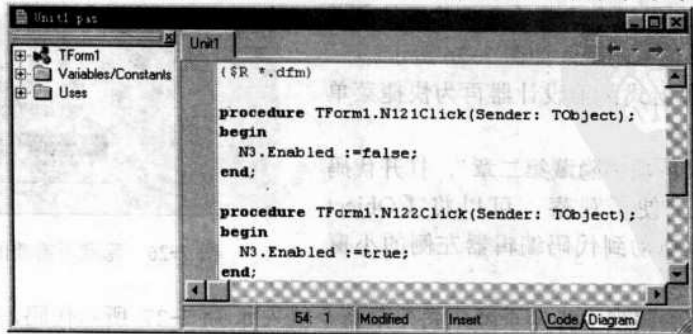


图 7-23 定义快捷菜单命令的代码

提示：当设置某个对象的【Enable】属性为“False”时，该对象就处于无效状态，在程序中表现为灰色无效状态。

(5) 执行程序，可见利用快捷菜单命令已经能够控制下拉菜单的“1.2 节”菜单项是否有效。但是，两个快捷菜单命令的复选状态却有问题，如图 7-24 所示。



图 7-24 两个快捷菜单命令的复选状态出现冲突

(6) 为了解决这个问题，还需要在程序代码中输入一些语句。打开代码编辑器，在这两个命令的单击事件中，再输入如图 7-25 所示内容，定义将两个菜单项的复选标记相反。

```
procedure TForm1.N121Click(Sender: TObject);
begin
  N3.Enabled :=false;
  N121.Checked :=true;
  N122.Checked :=false;
end;

procedure TForm1.N122Click(Sender: TObject);
begin
  N3.Enabled :=true;
  N122.Checked :=true;
  N121.Checked :=false;
end;
```

图 7-25 利用代码将两个菜单项的复选标记相反

(7) 执行程序。现在程序已经完全符合设计效果要求了。

有些情况下，需要将一些暂时不用或不应当出现的菜单（菜单项）隐藏起来，就好像根本没有这些菜单一样。

2. 隐藏不需要的菜单项

程序效果：利用弹出式菜单命令将下拉菜单中的“第二章”菜单隐藏，如图 7-26 所示。

(1) 利用弹出式菜单设计器再为快捷菜单添加两个菜单命令。

(2) 双击菜单项“隐藏第二章”，打开代码编辑窗口。为了便于观察，可以将【Object TreeView】窗口拖动到代码编辑器左侧的小窗口。

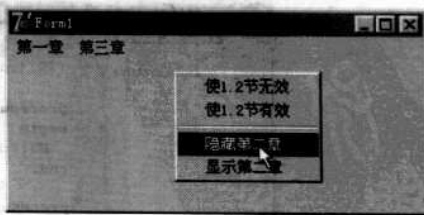


图 7-26 隐藏不需要的菜单项

(3) 在代码编辑器中为两个弹出式菜单命令输入如图 7-27 所示代码，分别用于定义对象 N5（下拉菜单“第二章”）是否可见。

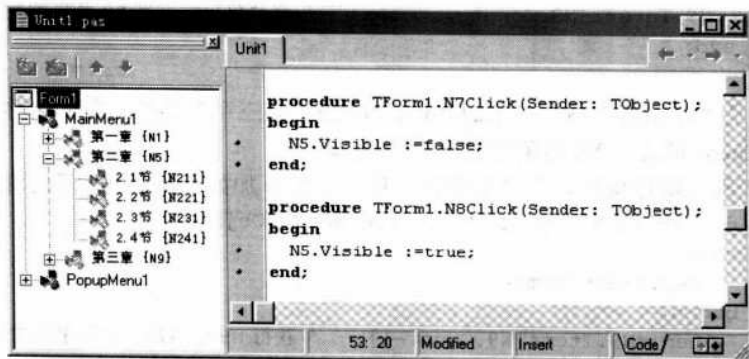


图 7-27 定义对象是否可见

提示：对象的【Visible】属性决定了对象是否可见，取值为“True”，可见；否则，为不可见。

(4) 执行程序，选择弹出式菜单的这两个命令，可以控制下拉菜单“第二章”是否可见。

有时候，需要在程序运行时动态添加或减少菜单和菜单项，而且这种菜单项的内容可能并不确定，这时就应使用菜单项的【Insert】方法和【Remove】方法了。

Delphi 中用来定义菜单项和菜单的类 TmenuItem 的每个对象都包括一个菜单项列表，这些选项的结构都采用递归形式。其中存储菜单项列表的菜单结构的属性是【items】，存储子菜单项树木的属性是【Count】。通过它们可以简单地添加和删除菜单项。

3. 添减菜单项

程序效果：使用一个文本编辑对象输入一个标题，然后使用快捷菜单命令添加一个具有该标题文字的菜单项；也可以利用菜单命令删除菜单项。如图 7-28 所示。

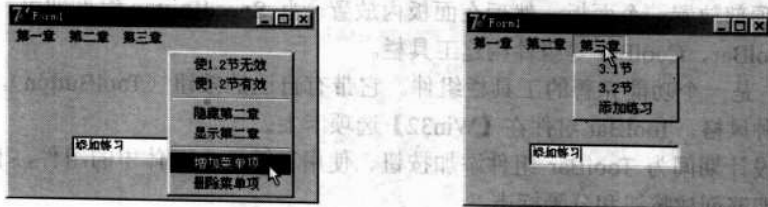


图 7-28 动态添减菜单项

- (1) 首先从【Standard】选项卡中向窗体添加一个 Edit 对象。
- (2) 按照程序效果要求，为弹出式菜单再添加两个菜单项。
- (3) 在代码编辑器窗口，为“增加菜单项”命令添加如下斜体文字表示的程序代码。

```
procedure TForm1.N11Click(Sender: TObject);
var
  item: TMenuItem; //定义一个菜单项类型的变量
begin
  item := TMenuItem.Create(self); //生成菜单项对象
  item.Caption := edit1.Text; //定义菜单项的标题为 Edit1 对象中的文字
```

```
N9.Insert(N9.Count, item); //将菜单项插入到 N9 菜单的最后位置
end;
```

其中:

- N11 是“增加菜单项”对象的名称, N9 是“第三章”菜单对象的名称。
- N9.Count 记录了 N9 对象中有多少个菜单项。

(4) 在代码编辑器窗口, 为“删除菜单项”命令添加如下斜体文字表示的程序代码。

```
procedure TForm1.N12Click(Sender: TObject);
var
  item:TMenuItem;
begin
  item:=N9.Items[N9.Count-1]; //获得 N9 中最后一个菜单项的位置值
  N9.Remove(item);           //删除指定位置的菜单项
end;
```

(5) 执行程序, 可以随意在文本编辑框中输入内容; 然后使用快捷菜单中的命令, 就能够反复增加或删除菜单项。

7.2 工具栏与状态栏

Windows 的窗口, 顶部大多有一个工具栏, 底部有一个状态栏。工具栏通常包含一些小按钮、组合框、编辑框等组件。这些组件一般能够实现与系统菜单相同的功能, 而且更加便捷。状态栏也有很多用处, 最常用的是显示用户当前所选菜单项的有关信息。除此之外, 还可以显示程序状态的信息。例如, 可以在图形应用程序中显示鼠标的位置, 在字处理器中显示文本的当前行号, 显示加锁状态, 时间与日期等等。

7.2.1 工具栏

Delphi 中设计工具栏的方法很多, 早期的工具栏是使用面板构建的, 功能简单, 在窗体的客户区顶部放置一个面板, 然后在面板内放置一些 SpeedButton 组件即可。Delphi 4.0 以后使用 ToolBar、CoolBar 等组件构建工具栏。

ToolBar 是一个功能完善的工具栏组件。它带有自己的按钮 (ToolButton), 并可将按钮设计成多种风格。ToolBar 组件在【Win32】选项卡上。

可以在设计期间为 ToolBar 组件添加按钮。使用右键单击窗体中的组件, 利用弹出的快捷菜单就能够创建按钮和分隔标志。

下面我们通过一个简单的实例说明如果使用 ToolBar 组件。

【例 7-4】文档编辑器工具栏

在窗体上创建一排类似 MS Office 标准按钮的工具栏, 并添加相应的图形按钮。程序效果如图 7-29 所示。

(1) 创建一个新的应用程序, 并适当调整窗体的大小。

(2) 如果你已经将 Delphi 7 安装到默认的目录, 请打开 “C:\Program Files\Borland\

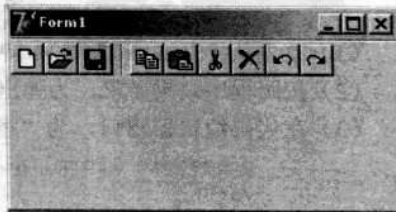


图 7-29 文档编辑器工具栏

Delphi7\Source\Vcl\ActnRes.pas”文件,【StandardsActions】窗口将被打开。

提示:这是一个 Delphi 7 的资源文件,其中的图像列表组件已经为标准指令指定了图像索引号。

(3) 选择 ImageList1 组件并复制它,将它粘贴到当前程序窗体中,如图 7-30 所示。

(4) 关闭【StandardsActions】窗口。双击 ImageList1 组件,从打开的图像列表对话框中,你可以看到所有可能使用到的图像,如图 7-31 所示。



图 7-30 标准的图像资源

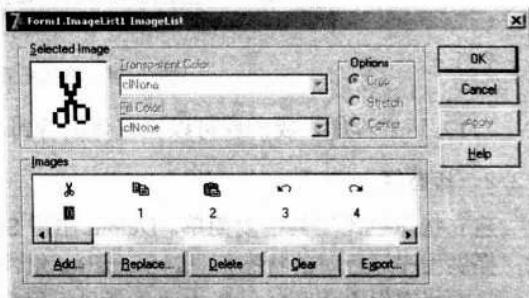


图 7-31 将 ImageList1 组件复制并粘贴到当前程序窗体中

当然我们也可以利用 **Add...** 按钮直接添加各种图像,但是要找到这些标准的按钮图像并不容易。

(5) 关闭对话框。再在窗体上添加一个 ToolBar 组件。

(6) 设置 ToolBar 组件的【Images】属性为“ImageList1”,如图 7-32 所示。

(7) 用鼠标右键单击 ToolBar 组件,从弹出的快捷菜单中选择【New Button】命令,在工具栏上添加一个按钮。

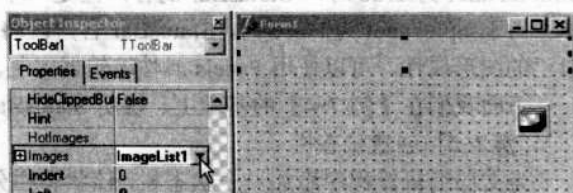


图 7-32 设置 ToolBar 组件的【Images】属性

(8) 选择该按钮,设置其【ImageIndex】属性值为 6,如图 7-33 所示。该值对应于图像列表中一个标准的“新建”文件图标。这样,按钮上面显示的图标就是这个“新建”文件图标。



图 7-33 设置按钮【ImageIndex】属性值为 6

(9) 同理,再在 ToolBar 上添加几个按钮,并分别设置适当的图标。

(10) 对于第 4 个按钮, 需要设置其 **【Style】** 属性为 “tbsDivider”, 如图 7-34 所示。这样, 按钮就表现为一个分隔线。

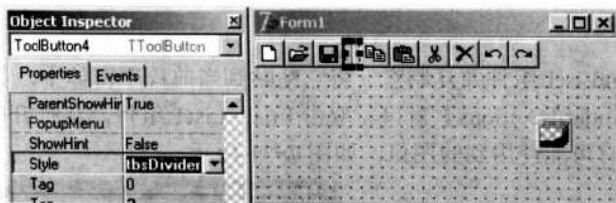


图 7-34 将按钮定义成分隔线

(11) 运行程序, 我们就可以在窗体上看到一排类似 MS Office 标准按钮的工具栏。

7.2.2 状态栏

建立状态栏比工具栏要简单。Delphi 中用 **StatusBar** 组件, 在窗体的下边显示一个状态栏, 提供程序运行时的状态信息。StatusBar 组件位于 **【Win32】** 选项卡上。

状态栏有很多用处, 一般主要用于显示用户操作或程序运行的有关信息。例如, 可以在图形应用程序中显示鼠标的位置, 在字处理器中显示文本的当前行号, 显示加锁状态, 时间与日期等等。

StatusBar 组件的结构有两种: 单面板与多面板, 由 **SimplePanel** 属性决定。当 SimplePanel 为 True 时, StatusBar 的客户区是一个单面板, 显示 SimpleText 属性中的字符串; 当 SimplePanel 为 False 时, StatusBar 客户区将被分割为多个子面板, 每个子面板都有自己的图形属性, 可以使用 Panels 属性编辑器定制。

下面我们在【例 7-4】的基础上, 说明 StatusBar 组件的使用方法。

【例 7-5】创建状态栏

为文档编辑器程序添加一个状态栏, 该状态栏分为 4 个部分, 能够分别显示光标位置、编辑状态、文件名称、日期时间等不同的信息。程序效果如图 7-35 所示。

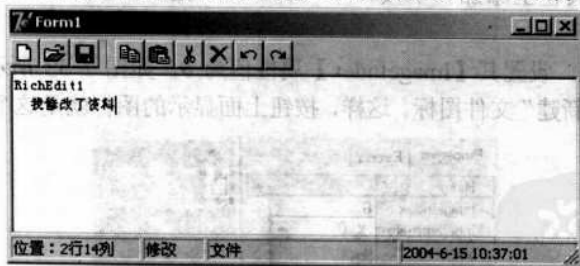


图 7-35 创建状态栏

(1) 打开【例 7-4】的工程文件。

(2) 在窗体上添加一个 StatusBar 组件。将它的 SimplePanel 属性修改为 “False”。双击状态栏, 在 “Panels 属性编辑器” 中加入 4 个子项, 其 **【Text】** 属性分别为 “位置”、“状态”、“文件”、“时间”, **【Width】** 属性分别为 100、50、150、150, 如图 7-36a 所示。设置完成后的状态栏, 如图 7-36b 所示。

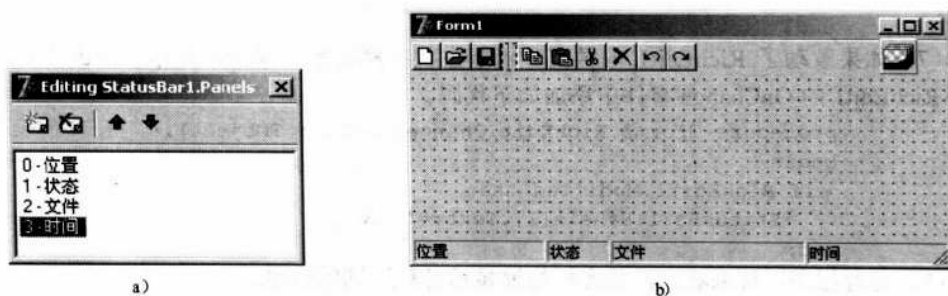


图 7-36 设置状态栏属性

(3) 在窗体上添加一个 Timer 组件。

(4) 在 Form1 的 OnCreate 事件和 Timer1 的 OnTimer 事件中添加如下代码，将系统日期和时间显示在状态栏的第四个子项上。

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    StatusBar1.Panels[3].Text:=DateTimeToStr(Now);
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    StatusBar1.Panels[3].Text:=DateTimeToStr(Now);
end;
```

其中, Now 是系统定义的 DateTime 类型的函数, 返回当前的日期和时间。DateTimeToStr 函数将 DateTime 类型的日期和时间转换成字符串。

(5) 在窗体上再添加一个 RichEdit 组件, 设置其【Align】属性为“alClient”, 这样, RichEdit 组件的窗口将充满整个窗体, 如图 7-37 所示。

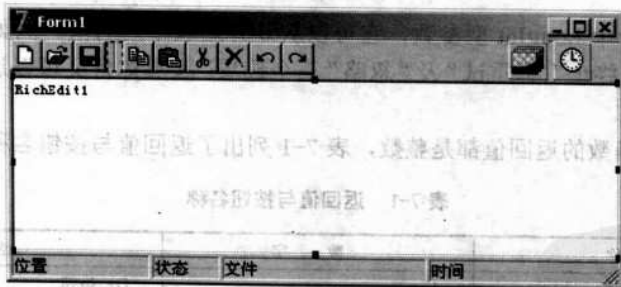


图 7-37 在窗体上添加一个 RichEdit 组件

(6) 在 RichEdit 组件的 OnSelectionChange 事件中添加如下代码, 将编辑窗口中光标的当前位置显示在状态栏的第一区域中。

```
procedure TForm1.RichEdit1SelectionChange(Sender: TObject);
begin
    StatusBar1.Panels[0].Text:='位置: '
        +inttostr(Richedit1.CaretPos.y+1) + '行'
        +inttostr(Richedit1.CaretPos.x) + '列';
```

end;

(7) 如果改动了 RichEdit1 的数据, 希望在状态栏的第二区域中显示“修改”标志, 需在 RichEdit1 的 OnChange 事件中添加如下代码。

```
procedure TForm1.RichEdit1Change(Sender: TObject);
begin
    if RichEdit1.Modified then
        StatusBar1.Panels[1].Text:='修改';
end;
```

(8) 运行程序, 可以看到, 状态栏能够显示各种不同的信息。

7.3 信息对话框

信息对话框是动态提示用户的主要手段。在许多应用程序中, 都可以看到它们的身影。例如在文档编辑器中, 会遇到提示用户保存文档的信息对话框。

在 Delphi 7 中, 信息对话框的创建由系统提供的函数来实现。下面我们分别简单说明一下。

7.3.1 MessageBox 函数

MessageBox 函数是 Windows API 函数, Delphi 7 能够直接使用它。函数格式为:

```
Function MessageBox(const
Text,Caption:Pchar;Flags:Longint=MB_OK):Integer;
```

其中:

- Text 参数是一个长度可以超过 255 等字符串, 它显示在对话框的中部, 能够自动环绕。
- Caption 参数类型与 Text 相同, 它只是作为对话框的标题, 也可以超过 255 个字符, 但不能环绕。
- Flags 是一个 LongInt 型参数, 它可以从 0 开始取不同的值。例如 0 代表一个“确定”选项, 2 代表“终止”、“重试”及“忽略”三个选项。其具体含义可以通过 Windows API 的帮助文件取得。

MessageBox 函数的返回值都是整数, 表 7-1 列出了返回值与按钮名称的对应关系。

表 7-1 返回值与按钮名称

按钮名称	数 字	含 义
IDOK	1	OK 按钮
IDCANCEL	2	Cancel 按钮
IDABORT	3	Abort 按钮
IDRETRY	4	Retry 按钮
IDIGNORE	5	Ignore 按钮
IDYES	6	Yes 按钮
IDNO	7	No 按钮

下面用一个简单的实例说明信息对话框的应用。

[例 7-6] MessageBox 函数应用练习

单击窗体上的按钮，就会产生一个信息框。程序画面如图 7-38 所示。



图 7-38 MessageBox 对话框

- (1) 新建一个应用程序，适当缩小窗体大小。
- (2) 在窗体上添加一个 Button 组件，定义其【Caption】属性为“MessageBox”。
- (3) 在 Button 组件的 OnClick 事件中添加如下代码：

```
Procedure TForm1.Form1Create(Sender:TObject);
Begin
    With Application do
        MessageBox('法国 2:1 英格兰','2004 欧洲杯战况',MB_OK);
End;
```

- (4) 运行程序，单击窗体上的按钮，就会出现一个信息对话框。

7.3.2 MessageDlg 函数

MessageDlg 函数的语法格式为：

```
Function MessageDlg(const Msg:string; DlgType:TmsgDlgType; Buttons:TmsgDlgButtons;
HelpCtx:Longint):Word;
```

其中：

- Msg 表示对话框的提示信息，可以是任意长度和内容。
- DlgType 表示该对话框的目的，比如警告、询问等。TmsgDlgType 参数取值和含义如表 7-2 所示。

表 7-2 TmsgDlgType 参数

取 值	含 义
mtWarning	信息对话框含有一个黄色的惊叹号
mtError	信息对话框含有一个红色的停止符号
mtInformation	信息对话框含有一个蓝色的“i”符号
mtConfirmation	信息对话框含有一个绿色的问号
mtCustom	信息对话框没有位图信息，但对话框的名字不再是 Msg，而是正在被使用的文件名

- Buttons 控制在信息对话框中出现的按钮名称，其参数如表 7-3 所示。

表 7-3 TmsgDlgButtons 参数表

取 值	含 义
mbYes	按钮 Yes 将出现在对话框上
mbNo	按钮 No 将出现在对话框上
mbOk	按钮 OK 将出现在对话框上
mbCancel	按钮 Cancel 将出现在对话框上

(续)

取 值	含 义
mbAbort	按钮 Abort 将出现在对话框上
mbRetry	按钮 Retry 将出现在对话框上
mbIgnore	按钮将出现在对话框上
mbAll	按钮 Ignore 将出现在对话框上
mbNoToAll	按钮 NoToAll 将出现在对话框上
mbYesToAll	按钮 YesToAll 将出现在对话框上
mbHelp	按钮 Help 将出现在对话框上
mbYesNoCancel	按钮 Yes、No、Cancel 将出现在对话框上
mbYesNoAllCancel	按钮 Yes、YesToAll、No、NoToAll、Cancel 将出现在对话框上
mbOKCancel	按钮 OK、Cancel 将出现在对话框上
mbAbortRetryIgnore	按钮 Abort、Retry、Ignore 将出现在对话框上
mbAbortIgnore	按钮 Abort、Ignore 将出现在对话框上

下面用一个简单的实例说明 MessageDlg 的应用。

【例 7-7】 MessageDlg 函数应用练习

单击窗体上的按钮，就会产生带有图标的信息框，选择 Yes 按钮后，还会出现进一步的提示信息。程序画面如图 7-39 所示。

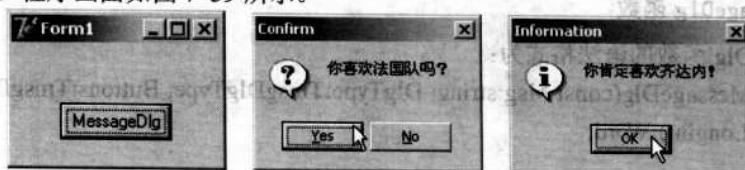


图 7-39 MessageBox 对话框

- (1) 新建一个应用程序，适当缩小窗体大小。
- (2) 在窗体上添加一个 Button 组件，定义其【Caption】属性为“MessageDlg”。
- (3) 在 Button 组件的 OnClick 事件中添加如下代码：

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    if messageDlg('你喜欢法国队吗?', mtConfirmation, [mbYes, mbNo], 0)
        = mrYes then
    begin
        messageDlg('你肯定喜欢齐达内!', mtInformation, [mbOK], 0);
        close;
    end;
end;

```

- (4) 运行程序，单击窗体上的按钮，就会出现信息对话框。

7.3.3 MessageDlgPos 函数

MessageDlgPos 函数与 MessageDlg 函数非常相似，只是 MessageDlgPos 函数能够控制信息对话框出现的位置。函数的语法格式为：

```

Function MessageDlgPos(const Msg:string; DlgType:TmsgDlgType;
    Buttons:TmsgDlgButtons; HelpCtx:Longint; X,Y:Integer):Word;

```

其中：参数 X、Y 表示信息对话框出现的位置

下面的代码能够利用按钮单击事件在指定的位置显示一个对话框：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    MessageDlgPos('Football!', mtconfirmation, mbYesNoCancel, 0, 200, 200);
end;
```

7.3.4 ShowMessage 函数

ShowMessage 函数是最简单的信息对话框，它只有一个参数。函数的语法格式为：

```
Procedure ShowMessage (const MSG:string);
```

该函数在被调用执行时，将弹出只有一个 OK 按钮的对话框。

7.3.5 InputBox 函数

InputBox 函数是一个既有按钮，又有文本输入框的信息对话框，因而使用起来更加灵活。函数的语法格式为：

```
Function InputBox(const Acaption, Aprompt, AdefaultMSG:string):string
```

其中：

- Acaption 表示信息对话框的名字。
- Aprompt 表示输入文本的提示信息。
- AdefaultMsg 是文本输入内容的默认值。

[例 7-8] InputBox 函数应用练习

单击窗体上的按钮，会出现一个输入文本对话框；任意输入信息，单击【OK】按钮关闭窗口，输入的内容就会出现在窗体上的文本框中。程序效果如图 7-40 所示。

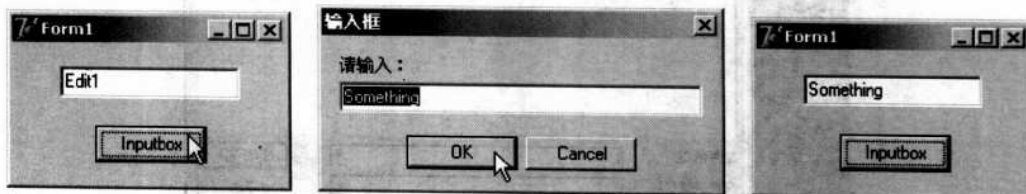


图 7-40 InputBox 函数应用练习

- (1) 新建一个应用程序，适当缩小窗体大小。
- (2) 在窗体上添加一个 Button 组件和一个 Edit 组件，定义 Button 组件的【Caption】属性为“InputBox”。
- (3) 在 Button 组件的 OnClick 事件中添加如下代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var
    inputstring:string;
begin
    inputstring:=inputbox('输入框','请输入: ','Something');
    edit1.Text:=inputstring;
end;
```

(4) 运行程序，我们就可以方便地使用输入框修改 Edit1 的内容了。

7.4 对话框组件

Delphi 7 提供的对话框组件包括 Open、Save、Font、Color 等，它们基本上都位于组件库的【Dialogs】选项卡上。所有的对话框组件都是不可见组件，当程序运行时也不会自动显示，而需要用户编写一些代码来显示（如使用菜单或按钮的 OnClick 事件处理过程）。每一个对话框的显示方式均相同，即都使用 Execute 方法。

例如，可以使用下面的代码来显示一个【打开文件】对话框：

```
OpenDialog1.Execute
```

Execute 方法的返回值为布尔类型。当用户单击对话框中的【OK】按钮时，返回 True，否则返回 False。用户可以根据返回值的不同而执行不同的事件处理过程。

对话框组件所提供的对话框大多是模态对话框，即在对话框被关闭之前，无法在同一应用程序的其他地方进行工作。

7.4.1 OpenFileDialog 对话框

OpenDialog（打开）对话框是用来打开文件的对话框窗口，如图 7-41 所示。

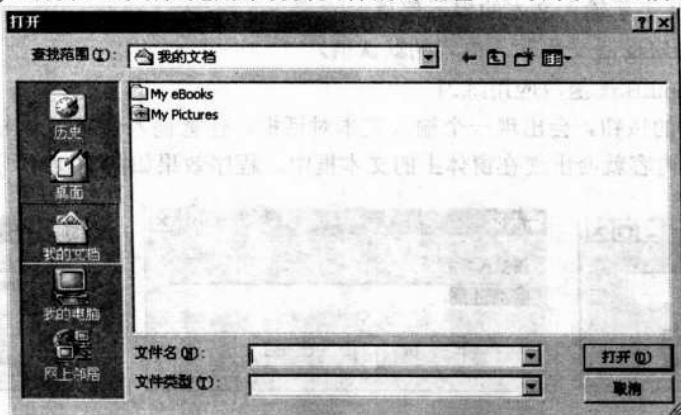


图 7-41 【打开】对话框

常用属性有：

- **【DefaultExt】**：用于指定一个默认的扩展名。主要用于文件保存时作为默认值。
- **【Filename】**：指定一个文件名。在【打开】对话框中，这个文件名出现在对话框的“文件名”栏。
- **【Filter】**：文件过滤器，使对话框只列出特定类型的文件。
- **【FilterIndex】**：如果有多个过滤器，这个属性用于指定一个作为默认过滤器。
- **【InitialDir】**：为对话框指定打开目录。默认情况下，对话框打开时，文件列表中显示的是当前目录的文件。
- **【Title】**：对话框的标题。


另外，OpenDialog 对话框还有一组 Options 属性，如表 7-4 所示。

表 7-4 OpenFileDialog 对话框 Options 属性的取值及含义

属性值	含 义
ofAllowMultiSelect	允许在文件名列表中选择多个文件
ofCreatePrompt	当用户在文件编辑框中输入一个不存在的文件名, 并选择 OK 按钮时, 会出现一个消息框, 提示用户此文件不存在, 并询问是否以此文件名创建一新文件
ofExiengronDifferent	从对话框中返回的文件扩展名将不同于默认扩展名。保存在 DefaultExt 属性中
ofFileMustExist	当用户在文件编辑框中输入一个不存在的文件名, 并选择了 OK 按钮, 则会出现一消息框提示用户文件不存在, 并询问是否输入了正确的路径和文件名
ofNoChangeDir	当前目录经设置成对话框第一次出现的目录, 并忽略任何目录改变
ofOverWritePrompt	当用户视图保存一个已经存在的文件时, 将出现一个消息框, 提示用户此文件已存在, 并询问是否覆盖
ofPathMustExit	用户在文件名编辑框中只能输入有效途径, 否则出现消息框, 提示用户路径无效
onShowHelp	对话框中将出现“帮助”按钮

下面, 我们在【例 7-5】的基础上扩充程序, 添加打开文件的功能。

【例 7-9】设计【打开】对话框

使用工具栏上的按钮, 能够打开一个【打开】对话框, 如图 7-42a 所示; 选择一个 TXT 格式的文件, 确定后, 文件中的内容被显示在 RichEdit1 窗口, 同时, 窗口底部的状态栏中显示出文件的路径和名称。如图 7-42b 所示。

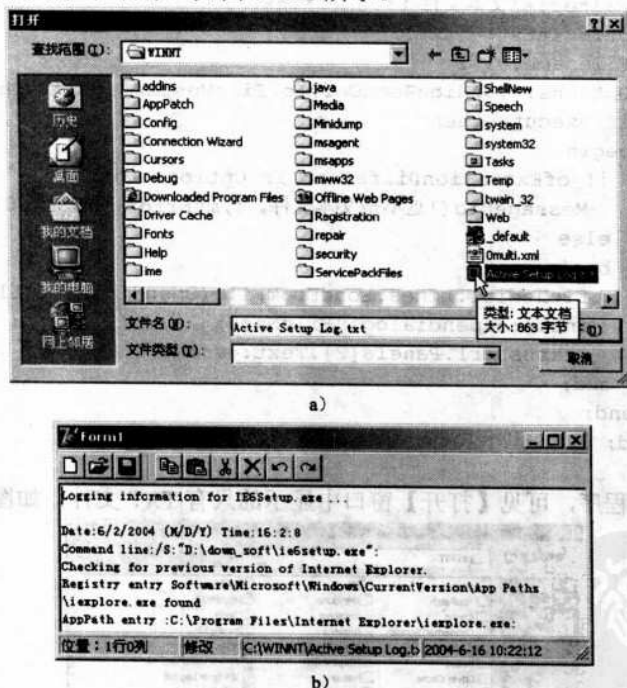




图 7-42 使用【打开】对话框打开文件

- (1) 打开【例 7-5】实例的工程项目文件。
- (2) 在窗体上添加一个 OpenFileDialog 组件。
- (3) 从工具栏上选择按钮, 双击进入代码编辑器, 在按钮的 OnClick 事件中添加

如下所示的代码:

```
procedure TForm1.ToolButton2Click(Sender: TObject);
begin
  if opendialog1.Execute then
  begin
    richedit1.Lines.LoadFromFile(Opendialog1.FileName);
    fname:=opendialog1.FileName;
    statusbar1.Panels[2].Text:=fname;
  end;
end;
```

(4) 运行程序, 就可以利用对话框来打开文件了。

但是现在这个程序还有一些问题, 因为一旦我们选择了打开一个非 TXT 格式的文件, 程序就会出错。为了解决这个问题, 我们需要使用过滤器 Filter。

(5) 修改  按钮的 OnClick 事件中的代码如下:

```
procedure TForm1.ToolButton2Click(Sender: TObject);
begin
  with opendialog1 do
  begin
    Filter:='文本文件 (*.txt|*.txt)';
    defaultExt:='txt';
    Filename:='';
    Options:=[ofHideReadOnly, ofFileMustExist, ofPathMustExist];
    if execute then
    begin
      if ofExtensionDifferent in Options then
        MessageDlg('这不是文本文件。', mtError, [mbOK], 0)
      else
      begin
        richedit1.Lines.LoadFromFile(Opendialog1.FileName);
        fname:=opendialog1.FileName;
        statusbar1.Panels[2].Text:=fname;
      end;
    end;
  end;
end;
```

(6) 再次运行程序, 可见【打开】窗口中显示的只有 TXT 文件, 如图 7-43 所示。




图 7-43 【打开】窗口中显示的只有 TXT 文件

7.4.2 SaveDialog 组件

SaveDialog（保存）组件的对话框界面和 OpenFileDialog 组件的对话框几乎完全一样。所以这里不再详细介绍。

我们直接使用【例 7-9】中的程序来说明 SaveDialog 组件的使用。

【例 7-10】 SaveDialog 组件的使用

使用编辑器中的  按钮，能够将文档中的内容保存为一个 TXT 格式的文件，同时，状态栏的信息也会发生改变。如图 7-44 所示。

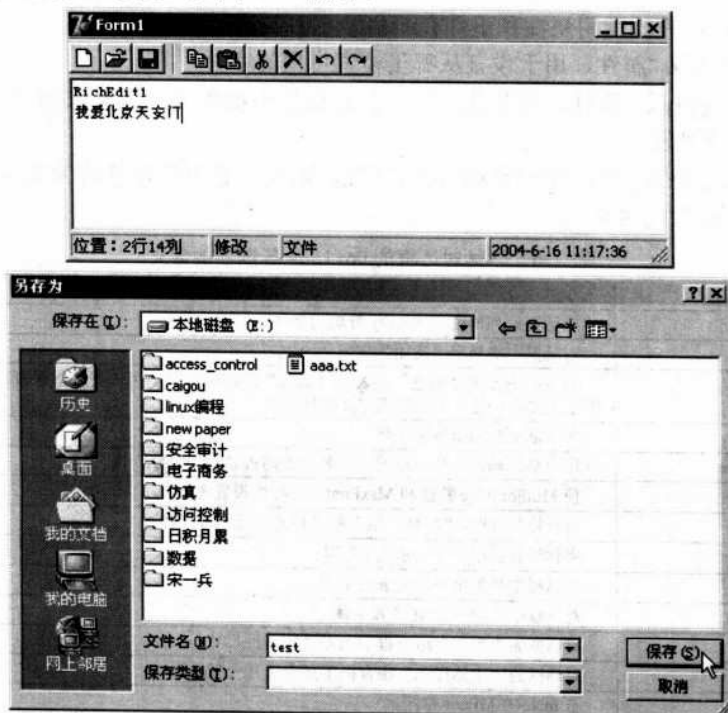




图 7-44 使用【另存为】对话框保存文件

- (1) 打开【例 7-9】实例的工程项目文件。
- (2) 在窗体上添加一个 SaveDialog 组件。
- (3) 从工具栏上选择  按钮，双击进入代码编辑器，在  按钮的 OnClick 事件中添加如下所示的代码：

```
procedure TForm1.ToolButton3Click(Sender: TObject);
begin
  with SaveDialog1 do
    if Execute then
    begin
      DefaultExt:='txt'; //定义缺省扩展名
      RichEdit1.Lines.SaveToFile(Filename); //保存文件
      RichEdit1.Modified:=False; //定义保存状态
    end;
end;
```

```

fname:=SaveDialog1.FileName;
StatusBar1.Panels[2].Text:=fname; //显示保存后的文件名
end;
end;

```

(4) 运行程序, 就可以利用对话框来保存文件, 同时, 状态栏的信息也会发生改变。

7.4.3 FontDialog 组件

FontDialog (字体) 组件在应用程序中产生【字体】对话框, 用户可以在对话框进行字体选择和属性设置, 相关信息存储在组件的【Font】属性中。

FontDialog 组件的常用属性和事件有:

- **【Device】**: 属性, 用于设置从哪里检索可用的字体。
- **【OnApply】**: 事件, 是否在【字体】对话框中出现【应用】按钮, 当用户按下该按钮时, 事件被触发。


另外, FontDialog 组件有一组 Options 属性, 默认时它们的取值均为 False, 表 7-5 为它们取值为 True 时的含义。

表 7-5 字体对话框的 Options 各选项的含义

取 值	含 义
fdAnsiOnly	对话框只列出使用 Ansi 字符集的字
fdApplyButton	对话框中将显示“应用”按钮
fdEffects	对话框中显示“颜色”列表和“效果”复选框; 用户可以使用效果复选框定义 Strikout 下划线文本; 使用颜色列表定义字体颜色
fdFixedPitchOnly	对话框中只列出等宽字体
fdForceFontExise	用户必须输入一个合法的字体名, 否则将显示一个警告框
fdLimitSize	使 MinFontSize 特性和 MaxFontSize 特性设置有效
fdNoFaceSel	对话框打开时, “字体”组合框不预先选择一种字体
fdNoOEMFont	字体组合框中将不显示向量字体
fdScalableOnly	对话框中只列出可以缩放
fdNoSimulations	对话框中不列出 GDI 仿真字体
fdNoSizeSel	对话框的“大小”组合框不预先选定一种尺寸
fdNoStyleSel	对话框的“字体样式”组合框不预先选择一种风格
fdNoVectorFonts	与 fdNoOEMFont 相同
fdShowHelp	对话框显示“帮助”按钮
fdTrueTypeOnly	对话框中只列出 TrueType 字体
fdWysiwyg	对话框中只列出所见即所得的字体

下面我们在【例 7-10】的基础上为文本编辑器添加字体对话框。

【例 7-11】添加字体对话框

为文本编辑器添加一个字体按钮: 从文本编辑窗口中选择一段文本后, 单击该按钮, 会弹出一个【字体】对话框, 如图 7-45a 所示; 选择某一字体后, 编辑窗口中选中的文本就会呈现为该字体类型, 如图 7-45b 所示。

(1) 打开【例 7-10】实例的工程项目文件。

(2) 我们首先需要在工具栏上添加一个字体按钮: 使用鼠标右键单击工具栏, 从弹出的快捷菜单中选择“New Button”命令, 在工具栏上添加了一个按钮; 在设置该按钮的【ImageIndex】属性值为 10, 这样, 按钮上就显示了一个合适的图标。如图 7-46 所示。

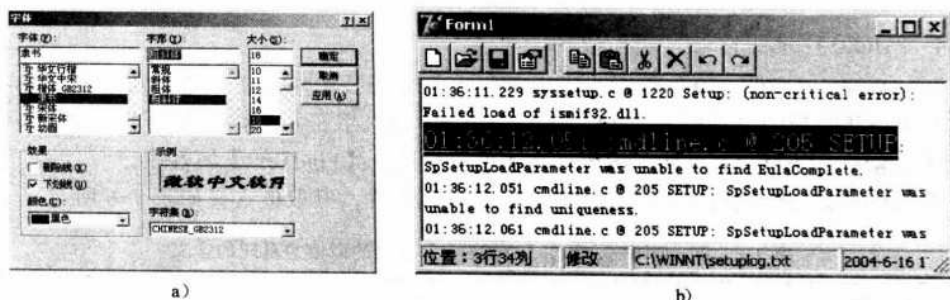


图 7-45 为编辑器添加字体对话框

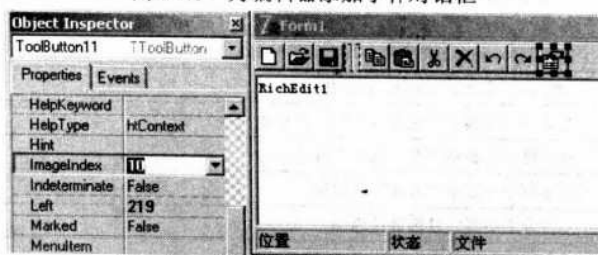


图 7-46 在工具栏上添加一个字体按钮

- (3) 将 按钮拖动到工具栏上比较靠前面的位置，与保存按钮放置在一起。
- (4) 在窗体上添加一个 FontDialog 组件。
- (5) 从工具栏上选择 按钮，双击进入代码编辑器，在 按钮的 OnClick 事件中添加如下所示的代码：

```

procedure TForm1.ToolButton1Click(Sender: TObject);
begin
    if RichEdit1.SelLength>0 then
    begin
        FontDialog1.Font.Assign(RichEdit1.SelAttributes);
        if FontDialog1.Execute then
            RichEdit1.SelAttributes.Assign(FontDialog1.Font);
        end
    else
        ShowMessage('请先选中一段文本。');
    end;
end;

```

- (6) 运行程序，就可以利用对话框来设置文本字体。如果没有选择文本对象，就会出现提示信息，要求首先选择一段文本。如图 7-47 所示。



图 7-47 没有选择文本对象，就会出现提示信息

7.4.4 FindDialog 组件

FindDialog (查找) 组件用于为应用程序提供【查找】对话框, 用户可以利用它在文本文件中查找匹配的字符串。如果用户在文本框中输入字符并选择【查找下一个】按钮, 对话框将触发【OnFind】事件, 需要查找的字符被放到【FindText】属性中。

【Options】属性决定了 FindDialog 组件的查找属性, 其取值含义如表 7-6 所示。

表 7-6 FindDialog 组件【Options】属性的取值为真时的含义

取 值	含 义
frDown	对话框中出现 Down 按钮, 查找方向向下。如果取值为 False, Up 按钮将被选中, 查找方向向上
frDisableUpDown	Up 和 Down 按钮将变灰, 用户不能选择
frFindNext	应用程序查找 FindNext 字符串
frHideMatchCase	对话框中不显示“区分大小写”复选框
frHideWholeWord	对话框中不显示“全字匹配”复选框
frHideUpDown	对话框中不显示“方向”分组框
frMatchCase	“区分大小写”复选框被选择
frWholeWord	“全字匹配”复选框被选中
frReplace	将 ReplaceText 属性中的字符串替换 FindText 属性中的字符串
frReplaceAll	将 ReplaceText 属性中的字符串替换所有 FindText 中的字符串
frDisableMatchCase	“区分大小写”复选框变灰
frDisableWholeWord	“全字匹配”复选框变灰

下面我们在【例 7-11】的基础上为文本编辑器添加查找对话框。

【例 7-12】为文本编辑器添加查找对话框

利用 FindDialog 组件为文本编辑器程序添加一个【查找】对话框, 并可以利用它在文本文件中查找字符。程序效果如图 7-48 所示。

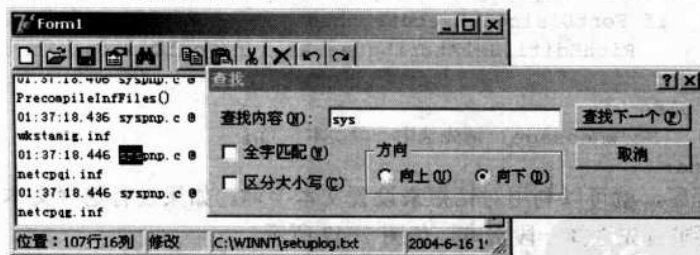


图 7-48 为文本编辑器添加查找对话框

- (1) 打开【例 7-11】实例的工程文件。
- (2) 首先需要在工具栏上添加一个查找按钮: 使用鼠标右键单击工具栏, 从弹出的快捷菜单中选择“New Button”命令, 在工具栏上添加了一个按钮; 在设置该按钮的【ImageIndex】属性值为 34, 这样, 按钮上就显示了一个望远镜图标。
- (3) 将按钮拖动到工具栏上比较靠前面的位置, 与字体按钮放置在一起。
- (4) 在窗体上添加一个 FindDialog 组件, 如图 7-49 所示。

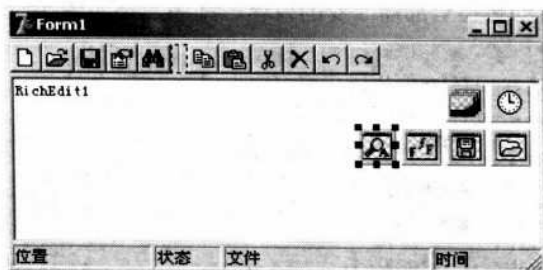




图 7-49 添加 FindDialog 组件

(5) 从工具栏上选择  按钮，双击进入代码编辑器，在  按钮的 OnClick 事件中添加如下所示的代码：

```
procedure TForm1.ToolButton12Click(Sender: TObject);
begin
    FindDialog1.Execute; //打开查找对话框
end;
```

(6) 在 FindDialog1 组件的 OnFind 事件中添加如下代码：

```
procedure TForm1.FindDialog1Find(Sender: TObject);
var
    FoundPos, InitPos : integer;
begin
    InitPos := RichEdit1.SelStart + RichEdit1.SelLength;
    FoundPos := Pos(FindDialog1.FindText, Copy(RichEdit1.Text,
        InitPos + 1, Length(RichEdit1.Text) - InitPos));
    if FoundPos > 0 then
    begin
        RichEdit1.SetFocus;
        RichEdit1.SelStart := InitPos + FoundPos - 1;
        RichEdit1.SelLength := Length(FindDialog1.FindText);
    end
    else
        MessageDlg('没有找到要查找的文本', mtInformation, [mbOK], 0);
end;
```


(7) 运行程序，就可以利用【查找】对话框从文本文件中查找字符串，并且文本编辑窗口中的光标会自动定位到符合查找要求的字符处。

7.4.5 ReplaceDialog 组件

ReplaceDialog 组件可以为应用程序提供【替换】对话框，完成对文本字符串的查找和替换。当用户在对对话框中输入要查找的字符串并按下【FindNext】按钮时，触发【OnFind】事件；当用户选择【Replace】或【ReplaceAll】按钮时，触发【OnReplace】事件。

下面我们继续以文本编辑器程序为例，说明如何使用 ReplaceDialog 组件。

【例 7-13】为文本编辑器添加替换对话框

在文本编辑框中添加一个替换按钮 ，单击按钮，会出现一个【替换】对话框；输入要替换的内容，单击【替换】按钮，就能够将文档中的相应内容替换掉；如果单击了【全

部替换】按钮，则会自动替换全部需改变的内容。程序效果如图 7-50 所示。

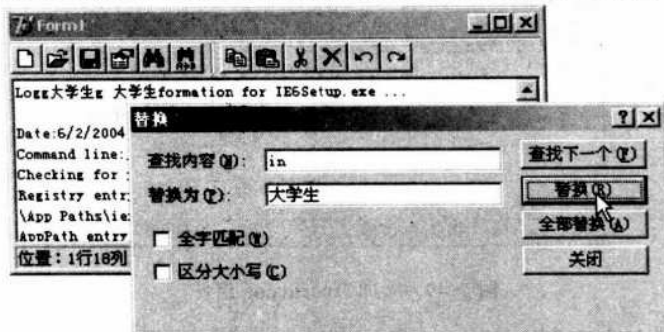




图 7-50 替换文本内容

(1) 打开【例 7-12】实例的工程项目文件。



(2) 首先需要在工具栏上添加一个替换按钮，设置按钮的【ImageIndex】属性值为 32。这样，按钮上就显示了一个标识为替换内容的图标。

(3) 将  按钮拖动到工具栏上比较靠前面的位置，与  按钮放置在一起。

(4) 在窗体上添加一个 ReplaceDialog 组件，如图 7-51 所示。



图 7-51 添加 ReplaceDialog 组件

(5) 从工具栏上选择  按钮，双击进入代码编辑器，在  按钮的 OnClick 事件中添加如下所示的代码：

```
procedure TForm1.ToolButton13Click(Sender: TObject);
begin
    ReplaceDialog1.Execute;
end;
```

(6) 在 ReplaceDialog1 组件的 On Replace 事件中添加如下代码：

```
procedure TForm1.ReplaceDialog1Replace(Sender: TObject);
var
    i, n: integer;
    s, find, replace: string;
begin
    s := RichEdit1.Text; //源串
    find := ReplaceDialog1.FindText; //查找串
    n := Length(find);
    replace := ReplaceDialog1.ReplaceText; //取代串
    i := Pos(find, s); //定位子串
    RichEdit1.SelStart := i-1;
    RichEdit1.SelLength := n;
    RichEdit1.SelText := replace; //取代子串
    if frReplaceAll in ReplaceDialog1.Options then //全部替换
```

```

repeat
  s:=RichEdit1.Text;
  i:=pos(find,s);
  if i<>0 then
    RichEdit1.Text:=Copy(s,1,i-1)+Copy(s,i+n,length(s)
      -(i+n)+1);
  Until i=0;
end;

```

(7) 运行程序, 就可以利用【替换】对话框从文本文件中替换掉查找到的字符串。

程序中使用 Copy 函数的一条语句, 是以子串 find 在源串 s 中首字符的位置 i 为界, 将源串 s 分成三部分, 如图 7-52 所示, 要取代的是其中第二段。

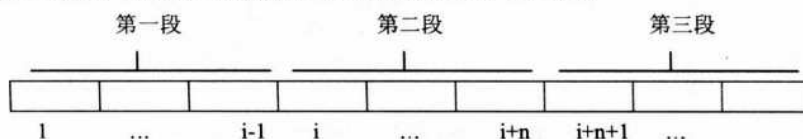


图 7-52 Copy 函数在代码中的应用说明

7.4.6 ColorDialog 组件


ColorDialog 组件用于显示标准的 Windows 颜色对话框。该组件的【Options】属性的取值情况如表 7-7 所示。

表 7-7 ColorDialog 组件【Options】属性各项取值为真时的含义

取 值	含 义
cdFullOpen	对话框打开时同时打开自定义颜色部分
cdPreventFullOpen	对话框中不允许自定义颜色
cdShowHelp	对话框中将显示“帮助”按钮
cdSolidColor	让 Windows 使用与所选颜色最接近的基本颜色
cdAnyColor	允许用户选择非基本颜色

下面我们继续以文本编辑器程序为例, 说明如何使用 ColorDialog 组件。

【例 7-14】为文本编辑器添加颜色对话框

为文本编辑器添加一个色彩按钮, 单击它可以打开【颜色】对话框, 并为编辑窗口中选择的文字设置不同的色彩, 如图 7-53 所示。

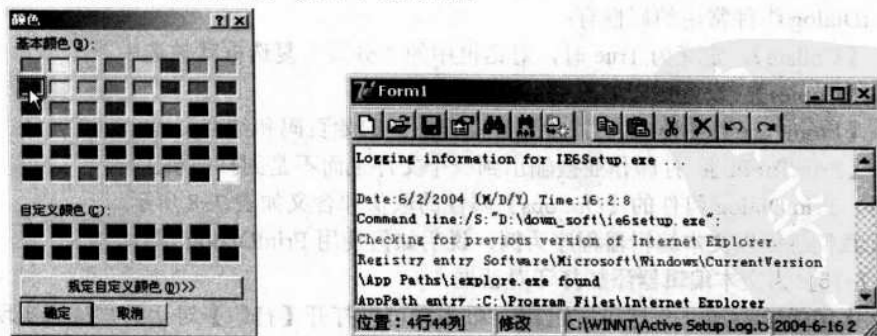





图 7-53 为文本编辑器添加颜色对话框

- (1) 打开【例 7-13】实例的工程项目文件。
- (2) 在工具栏上添加一个色彩按钮, 设置按钮的【ImageIndex】属性值为 42。这样, 按钮上就显示了一个图标。
- (3) 将按钮拖动到工具栏上比较靠前面的位置, 与按钮放置在一起。
- (4) 在窗体上添加一个 ColorDialog 组件, 如图 7-54 所示。

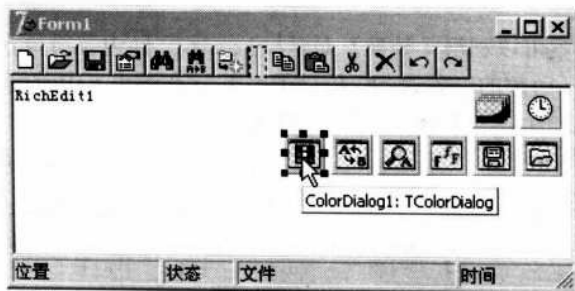




图 7-54 在窗体上添加一个 ColorDialog 组件

- (5) 从工具栏上选择按钮, 双击进入代码编辑器, 在按钮的 OnClick 事件中添加如下所示的代码:

```
procedure TForm1.ToolButton14Click(Sender: TObject);
begin
  with ColorDialog1 do
  begin
    Color:=RichEdit1.SelAttributes.Color;    //对话框显示选中文本的颜色
    if Execute then
      RichEdit1.SelAttributes.Color:=Color; //设置选中文本的颜色
  end;
end;
```

- (6) 运行程序, 就可以利用【颜色】对话框为文档中选中的文本设置颜色了。

7.4.7 PrintDialog 组件

PrintDialog 组件可以显示一个标准的打印对话框。用户可以利用对话框选择使用哪台打印机、设置打印机的属性、选择页码范围、设置打印份数等。

PrintDialog 组件常用的属性有:

- **【Collate】**: 定义为 True 时, 对话框中的“分页”复选框就被选中。
- **【Copies】**: 设置打印的份数。
- **【FromPage】**、**【ToPage】**: 设置打印范围的起始页码和结束页码。
- **【PrintToFile】**: 打印作业被输出到一个文件上而不是到打印机上。

另外, PrintDialog 组件的【Options】属性的取值和含义如表 7-8 所示。

下面我们继续以文本编辑器程序为例, 说明如何使用 PrintDialog 组件。

【例 7-15】为文本编辑器添加打印对话框


为文本编辑器添加一个打印按钮, 单击它可以打开【打印】对话框, 从而实现文件的打印, 如图 7-55 所示。

表 7-8 PrintDialog 组件【Options】属性的取值和含义

取 值	含 义
poPageNums	用户可以选择页的范围
poPrintToFile	对话框中将出现“打印到文件”复选框
poSelection	用户可以只打印文档中选中的部分
poWarning	如果没有安装打印机，将显示一个警告框
poHelp	对话框将显示一个“帮助”按钮
poDisablePrintToFile	对话框上的“打印到文件”复选框将变灰

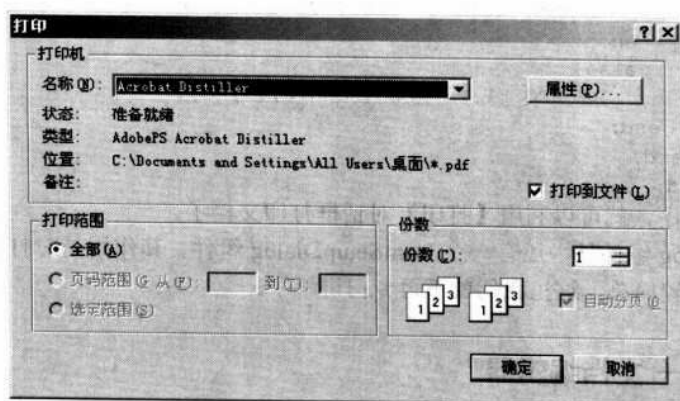





图 7-55 标准的【打印】对话框

- (1) 打开【例 7-14】实例的工程项目文件。
- (2) 在工具栏上添加一个色彩按钮, 设置按钮的【ImageIndex】属性值为 14。这样, 按钮上就显示了一个图标。
- (3) 将按钮拖动到工具栏上比较靠前面的位置, 与按钮放置在一起。
- (4) 在窗体上添加一个 PrintDialog 组件, 如图 7-56 所示。

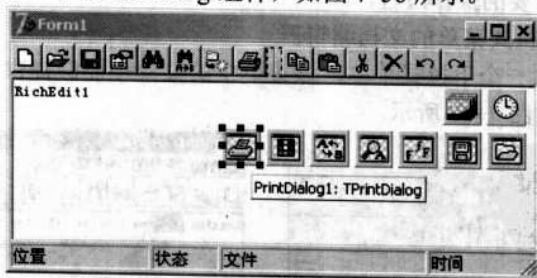




图 7-56 在窗体上添加一个 PrintDialog 组件

- (5) 从工具栏上选择按钮, 双击进入代码编辑器, 在按钮的 OnClick 事件中添加如下所示的代码:

```
procedure TForm1.ToolButton15Click(Sender: TObject);
begin
```

```

with PrintDialog1 do
begin
  Options:=[poPrintToFile];
  PrintToFile:=True;
  if Execute then
  begin
    if PrintToFile then
    begin
      SaveDialog1.Title:='打印到文件: ';
      if SaveDialog1.Execute then

        RichEdit1.Lines.SaveToFile(SaveDialog1.FileName+'.prt');
    end
    else
      RichEdit1.PaintTo(Handle,10,10);
    end;
  end;
end;

```

(6) 运行程序, 就可以利用【打印】对话框打印文档了。

除 PrintDialog 组件外, 还有一个 PrintSetupDialog 组件, 其作用是对打印机进行设置。该组件的用法比较简单, 这里我们就不再专门讲解了。

7.5 实例 —— 文档编辑器

下面, 我们将前面练习的文档编辑器实例扩充, 使它成为一个功能完整、符合 Windows 程序风格的文档编辑器。为了实现这个目标, 我们需要为文档编辑器添加下拉式菜单、弹出式菜单等, 并为菜单项添加 Action 行为。

7.5.1 添加下拉式菜单

下面我们首先来为文档编辑器添加下拉式菜单。下拉式菜单的设计我们在 7.1 节已经学习了, 这里仅讲述主要的操作过程。


[例 7-16] 添加下拉式菜单的文档编辑器

为文档编辑器添加一个下拉式菜单, 该菜单中的菜单项能够提供我们前面为文档编辑器设计的所有功能。如图 7-57 所示。



图 7-57 添加下拉式菜单的文档编辑器

(1) 打开【例 7-15】实例的工程项目文件。

(2) 程序工具栏上的按钮，有些已经添加了操作代码，有些还没有。对于后面几个与编辑操作相关的按钮，我们暂且不考虑其代码，但是对于第一个  按钮，我们还是要为它添加如下代码，以实现“新建”文件的功能。

```
procedure TForm1.ToolButton1Click(Sender: TObject);
begin
    RichEdit1.Clear;
end;
```

(3) 在窗体中添加一个 MainMenu 组件，并设置其【Images】属性为“ImageList1”，这样就可以为菜单项添加图标了。

(4) 双击 MainMenu 组件，打开菜单设计器，在其中创建【文件】、【编辑】、【样式】三个菜单，并添加相应的菜单项，如图 7-58 所示。



图 7-58 创建菜单及菜单项

(5) 为各个菜单项指定对应的 OnClick 事件。由于我们已经实现了许多功能，所以从下拉列表中就能够找到对应的事件，如图 7-59 所示。对于部分没有对应功能的菜单项，如复制、粘贴、加粗等，暂时先不管它。

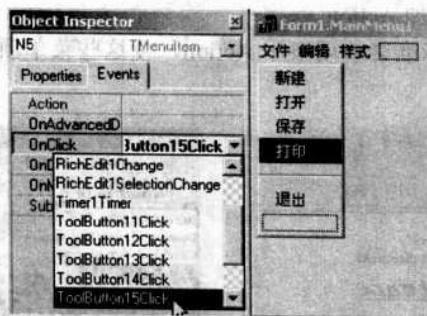


图 7-59 为各个菜单项指定对应的 OnClick 事件

注意：由于我们在前面设计工具栏按钮时，没有根据按钮功能定义其名称，这就使得这里选择对应事件比较困难，需要一一查找。从这个麻烦中我们应当看到，为对象定义含义明确的名称是非常有意义的。

(6) 由于“退出”菜单项没有已经实现的对应项，所以我们要为它定义 OnClick 事件。单击“退出”菜单项，在其 OnClick 事件添加代码如下：

```
procedure TForm1.N7Click(Sender: TObject);
begin
    Exit;
end;
```

(7) 运行程序，可见文档编辑器的下拉菜单已经能够完成工具栏上按钮所提供的功能。

7.5.2 Action 行为

在【编辑】和【样式】菜单中，包含了一些标准的 Windows 菜单项，如剪切、复制、粘贴等，除了常规的编程方法外，还有一个更为简便的方法，就是使用 ActionList 行为列表组件。ActionList 组件在【Standard】选项卡上。

ActionList 行为列表组件中包含了许多行为 (Action)。行为既说明了所执行的操作，又确定了所有与行为有关的元素状态。行为对象有自己的名称，而且拥有用于链接组件的其他属性，包括标题 (Caption)、图像表示 (ImageIndex)、状态 (Checked、Enabled 与 Visible) 等属性。每个行为对象都通过 ActionLink 对象与一个或多个客户对象连接。多个组件可以共享一个行为对象，这由它们的 Action 属性来说明。

与行为相连的客户组件通常是菜单项和各种类型的按钮，如设计剪切、复制、粘贴等这一组菜单项或按钮，使用时不仅要写相关操作的语句，还要确定这组菜单项之间的状态，当用户没有选中一段文字时，剪切、复制的菜单项应是不可执行状态等。一般情况下，将剪切、复制、粘贴设计成一组通用的行为，供菜单项或按钮使用。

行为对象由行为列表组件 ActionList 管理，对行为的操作在 ActionList 编辑器中进行。双击 ActionList 组件，就能够进入 ActionList 编辑器。

下面我们继续前面的练习，通过为几个菜单项添加 Action 来说明 ActionList 组件的用法。

[例 7-17] 为菜单项添加 Action

为下拉菜单中的部分标准菜单项添加 Action，使这些菜单项能够完成相应的功能。如图 7-60 所示。

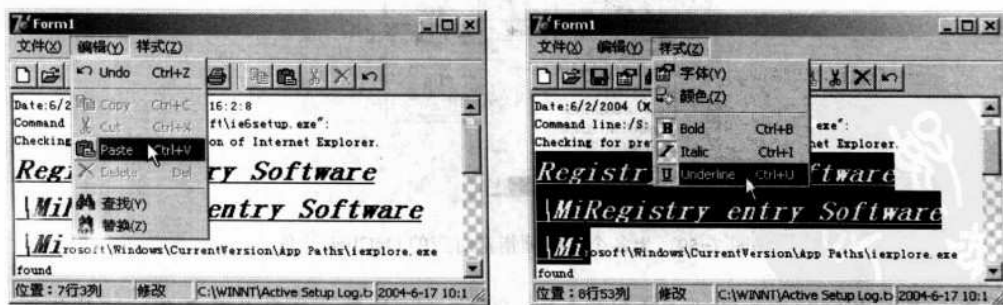


图 7-60 为菜单项添加 Action

(1) 向窗体中拖入一个 ActionList 组件，如图 7-61 所示。

(2) 双击该 ActionList 组件，打开 ActionList 编辑器，如图 7-62 所示。

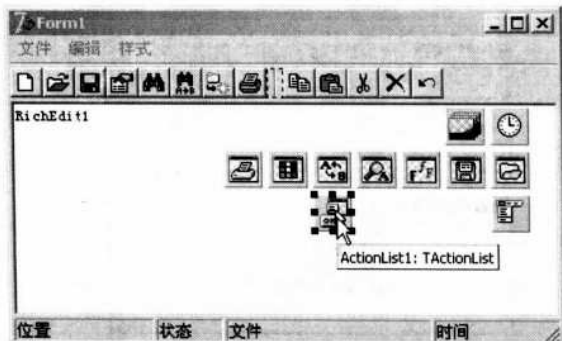


图 7-61 向窗体中拖入一个 ActionList 组件

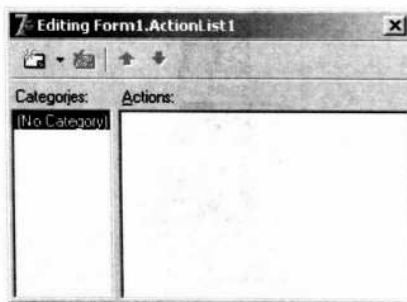


图 7-62 ActionList 编辑器

(3) 在 ActionList 编辑器中单击鼠标右键，从弹出的快捷菜单中选择“New Standard Adction”命令，会出现【Standard Action Classes】对话框，其中以分类方式罗列了系统提供的标准 Action 命令。如图 7-63 所示。

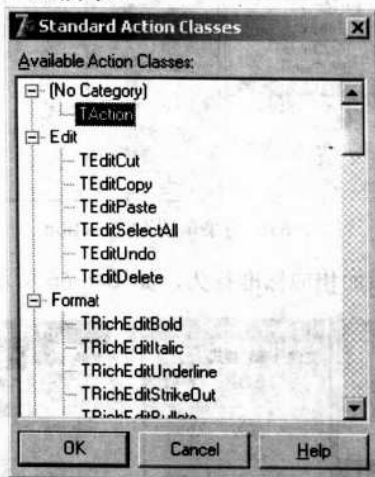


图 7-63 系统提供的标准 Action 命令

提示：所谓“系统定义的标准行为”是指在系统中注册的已有行为，其功能已经被内建，可以直接在程序中调用。

(4) 选择其中需要的一些 Action 命令，然后单击 **OK** 按钮，则选中的命令会出现在 ActionList 编辑器中，如图 7-64 所示。

(5) 关闭 ActionList 编辑器。在窗体上双击 MainMenu1 组件，打开菜单设计器，选择【编辑】菜单中的菜单项【撤销】，在【Object Inspector】窗口设置其【Action】属性为“EditUndo1”，如图 7-65 所示。

(6) 设置完成后，可见菜单项的名称已经被自动修改为系统默认的名称“Undo”，并且被添加了快捷键。

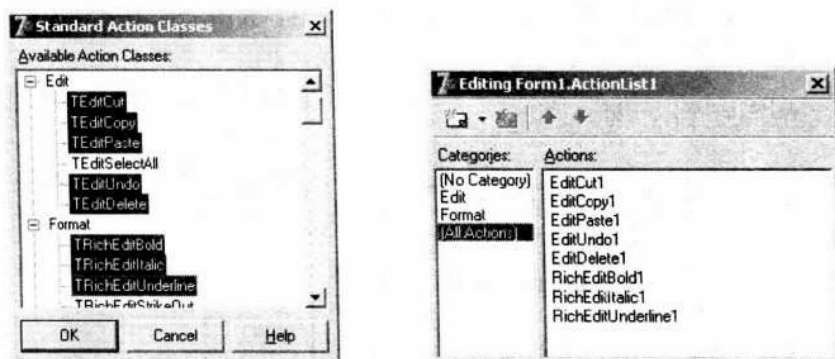


图 7-64 选中的命令会出现在 ActionList 编辑器中



图 7-65 为菜单项设置 Action

(7) 同理为其他菜单项添加相应标准行为，如图 7-66 所示。



图 7-66 为各菜单项添加相应标准行为

提示：可以根据需要修改这些菜单项的名称。

(8) 运行程序，可见现在所有的菜单项，包括添加 Action 命令的菜单项，都已经能够提供相应的功能。

7.5.3 添加弹出菜单

在文档编辑器中，一般都提供了快捷菜单，所以我们现在也来为程序添加弹出菜单。

【例 7-18】为文档编辑器添加弹出菜单

使用鼠标右键，能够打开弹出式快捷菜单，菜单中提供了常用的编辑和样式命令，如图 7-67 所示。

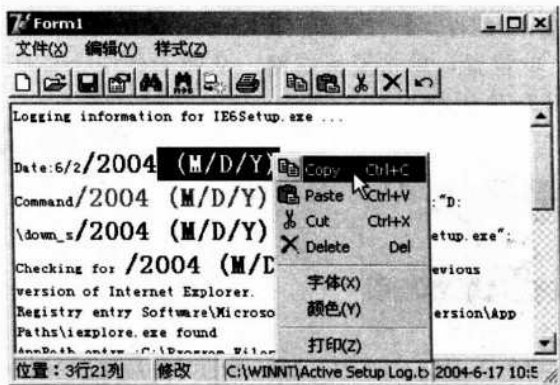


图 7-67 弹出式快捷菜单

- (1) 在窗体上添加一个 PopMenu 组件。
- (2) 双击该组件，打开菜单设计器，在其中添加各个菜单项，并为菜单项设置相应的 Action 或 OnClick 事件，如图 7-68 所示。



图 7-68

- (3) 运行程序，就可以利用快捷菜单来对文档进行编辑了。

7.5.4 菜单模板

1. 菜单模板

系统用菜单模板来保存已定义的菜单。用户既可以添加系统定制的菜单模板，也可以将自定义的菜单以模板形式保存。

(1) 添加系统设计菜单：在应添加菜单处，单击鼠标右键，执行弹出菜单中的“Insert From Template”命令，将系统设计好的菜单添加进来，如 File、Edit、Window、Help 等。

(2) 将自定义菜单存为模板：单击鼠标右键，执行弹出菜单中的“Save As Template”命令，将自己定义的菜单存为模板，供下次再使用。同样可以用“Delete Template”命令删除菜单模板。

下面我们利用系统菜单模板为文档编辑器提供一个“帮助”菜单。

[例 7-19] 添加“帮助”菜单

使用系统菜单模板为文档编辑器提供一个“帮助”菜单。如图 7-69 所示。

(1) 双击 MainMenu1 组件，打开菜单设计器。

(2) 选择一个菜单位置，单击鼠标右键，从弹出的快捷菜单中选择“Insert From Template”命令，如图 7-70 所示。

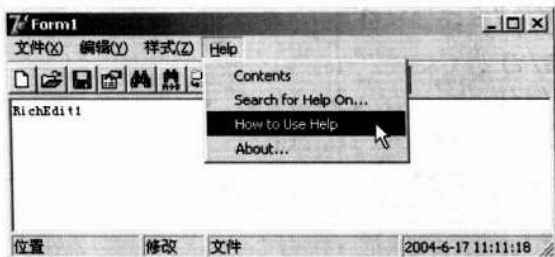
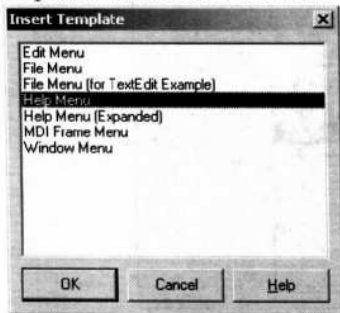


图 7-69 添加“帮助”菜单



图 7-70 选择菜单命令

(3) 再从出现的【Insert Template】对话框中选择“Help Menu”命令，如图 7-71a 所示，则 Help 菜单出现在菜单设计器中，如图 7-71b 所示。



a)



b)

图 7-71 Help 菜单出现在菜单设计器中

(4) 运行程序，可见程序中出现了 Help 菜单。

7.6 小结

几乎所有的 Windows 应用程序都借助于对话框和菜单与用户进行交互。对话框本质是一种窗口，不但可以接受消息，还能移动和关闭。一般在 Delphi 7 应用程序中可以使用三种对话框：由 Delphi 提供的对话框组件、由 Delphi 预定义的标准对话框函数、用户自己设计的对话框。其中前两种是最常用的对话框创建方式。

本章详细讲解了 Delphi 7 中菜单和对话框的设计方法，这些知识使我们能够设计出合理的应用程序界面，并向用户提供良好的交互反馈。

第8章 图形图像技术

在 Delphi 7 中,专门定义了一组对象和控件用以绘制图形,完成一些简单的图像功能。利用这些对象、控件的方法,可以方便地绘制各种常用图形;通过设置它们的属性,能得到不同风格的图形。另外,通过对鼠标事件的定义,可以方便地设计图形绘制程序。

8.1 画布技术

在 Delphi 中为编程人员定义了一个灵活的作图方式,即 Canvas 类(也就是画布),它可以把某些组件的表面当作一张画布,允许编程者利用有关命令在其上面随意作图。在 Delphi 中有许多组件支持画布类。

8.1.1 画布(Canvas)

Delphi 7 中的绘图操作都是在画布 Canvas 中进行的。Canvas 对象功能强大,它是 Windows 图形设备接口(GDI)的具体化,它为窗体中绘制图形对象提供了与设备独立的接口,用户可以很方便地在画布上绘制直线、矩形、椭圆、多边形等基本图形,还可以显示、拷贝位图。Canvas 对象本身并不是一个组件,不能单独使用,但很多组件都提供了 Canvas 属性,下面介绍一下 Canvas 对象的方法、属性和事件。

1. Canvas 对象方法

(1) Arc: 该方法的格式为:

Arc(x1,y1,x2,y3,x4,y4:Integer);

Arc 方法在椭圆上画一段弧,椭圆由(x1,y1),(x2,y2)两点所确定的椭圆所决定。弧的起点是椭圆圆周和椭圆中心与(x3,y3)连线的交点。弧的终点是椭圆圆周和椭圆中心与(x4,y4)连线的交点,以逆时针方向画弧。

【例 8-1】Arc 方法的使用。

效果:在窗口中绘制一个圆弧。执行结果如图 8-1 所示。

1) 新建一应用程序工程。在窗体上放置一命令按钮,该命令按钮的【Caption】属性为“画圆弧”。

2) 在命令按钮组件的 OnClick 事件中添加如下代码:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R:TRect;
```

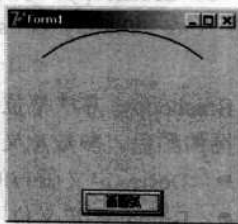


图 8-1 绘制一个圆弧


```

begin
    R:=GetClientRect;           //得到当前窗口的边界
    Canvas.Arc(R.Left,R.Top,R.Right,R.Bottom,R.Right,R.Top,R.Left,R.Top);
end;

```

3) 编译并运行程序, 窗体上出现一个圆弧。

(2) Chord: 该方法的格式为:

```
Chord(x1,y1,x2,y2,x3,y3,x4,y4:Integer);
```

Chord 方法连接椭圆上的两点, 椭圆由(x1,y1), (x2,y2) 两点所确定的矩形决定, (x3,y3) 是始点, (x4,y4) 是终点。

[例 8-2] Chord 方法的使用。

效果: 在窗口中绘制一个弓形。执行效果如图 8-2 所示。

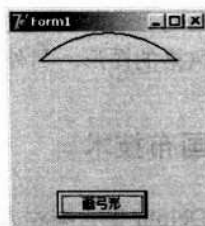


图 8-2 绘制一个弓形

1) 新建一个应用程序工程。在窗体上放置一命令按钮, 该命令按钮的【Caption】属性为“Button1”。

2) 在命令按钮组件的 OnClick 事件中添加如下代码:

```

procedure TForm1.Button1Click(Sender: TObject);
var
    R:TRect;
begin
    R:=GetClientRect;           //得到当前窗口的边界
    Canvas.Chord(R.Left,R.Top,R.Right,R.Bottom,R.Right,R.Top,R.Left,
        R.Top);
end;

```

3) 编译并运行程序。在窗体中点击按钮, 会出现一个弓形。

(3) Brushcopy: 该方法的格式为:

```

Brushcopy(const Dest:Trect;Bitmap:Tbitmap;const Source Trect;
    Color:Tcolor);

```

Brushcopy 方法把位图的一部分复制到画布的某个矩形区域, 并用画笔的当前颜色替换位图的颜色。参数意义如下:

- Dest: 定义画布的一个矩形区域, 该矩形用以填充位图。
- Bitmap: 定义位图。
- Source: 定义位图中的矩形区域, 该区域上的位图将被复制。
- Color: 定义画笔中用以替换位图的颜色。

[例 8-3] Brushcopy 方法的使用。

效果: 在窗体上放置一个图片, 点击按钮, 在窗体上显示该图片的一部分。

1) 新建一个应用程序工程。向窗体中添加一个 Button 组件, 该组件的【Caption】属

性为“图形复制”。再添加一个 Image 组件, 在该组件的【Picture】属性中添加一张图片。窗体设计界面如图 8-3 所示。



图 8-3 窗体设计界面

2) 在 Button1 组件的 OnClick 事件中添加如下代码:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Bitmap:Tbitmap;
    MyRect:Trect;
begin
    MyRect:=Rect(40,40,150,150);
    Bitmap:=Tbitmap.Create;
    Bitmap.LoadFromFile('风景.bmp');
    Form1.Canvas.BrushCopy(MyRect,Bitmap,MyRect,clBlack);
    Bitmap.Free;
end;
```

3) 运行程序。单击【图形复制】按钮, 在窗体上能够显示该图片的一部分, 程序的执行结果如图 8-4 所示。

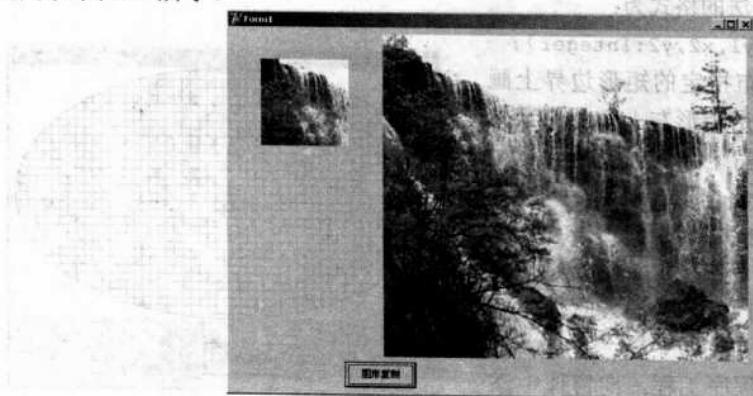


图 8-4 程序的运行结果

(4) CopyRect: 该方法的格式为:

```
CopyRect(Dest:Trect;Canvas:Tcanvas;SourceTRect);
```

此方法从另一个画布对象上复制部分图像到该画布。参数意义如下:

- Dest: 表示目标画布上将接受复制图像的矩形区域。
- Canvas: 表示源画布。
- Source: 源画布上要复制的图像区域。

(5) Draw: 该方法的格式为:

```
Draw(x,y: Integer; Graphic: Tgraphic);
```

此方法在画布给定的像素点坐标 (x, y) 处画 Graphic 所给的图像, 该图像可以是位图、图标或元位图。

例如下面的代码说明该方法是如何使用的。

```
var
  Bitmap:Tbitmap;
begin
  Bitmap:=Tbitmap.Create;
  try
    With Bitmap do begin
      LoadFromFile('风景.bmp');
      Transparent:=True;
      TransparentColor:=Bitmap.Canvas.Brush.Color;
      Form1.Canvas.Draw(10,10,Bitmap);
      TransparentMode:=tmAuto;
      Form1.Canvas.Draw(60,60,Bitmap);
    end;
  finally;
    Bitmap.Free;
  end;
```

(6) Ellipse: 该方法的格式为:

```
Ellipse(x1,y1,x2,y2:Integer);
```

Ellipse 方法在画布指定的矩形边界上画一个椭圆, (x1, y1) 是矩形左上角的像素坐标, (x2, y2) 是矩形右下角的像素坐标。如果矩形形成一个区域, 将出现一个椭圆。

【例 8-4】 Ellipse 方法的使用。

效果: 在窗体上画一个由 Image 指定矩形边界画一个内部填充十字交叉图案的蓝色椭圆。效果如图 8-5 所示。

1) 新建一个应用程序工程。向窗体中添加一个 Button 组件, 该组件的【Caption】属

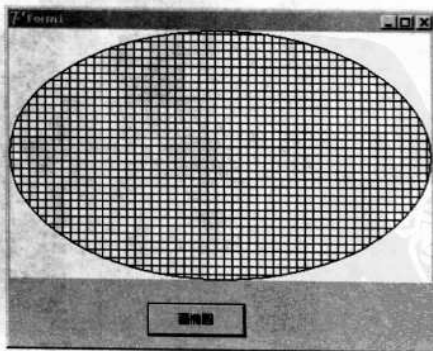


图 8-5 执行效果

性为“画椭圆”。

2) 再添加一个 Image 组件。

3) 在 Button1 组件的 Onclick 事件中添加如下代码:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    with Image1 do begin
        Canvas.Brush.Color:=clBlue;
        Canvas.Brush.Style:=bsCross;
        Canvas.Ellipse(0,0,Image1.Width,Image1.Height);
    end;
end;
```

4) 运行程序。单击【画椭圆】命令按钮,在窗体中会出现一个内部填充十字交叉图案的蓝色椭圆。

(7) LineTo: 该方法的格式为:

```
LineTo(x,y:Integer);
```

LineTo 方法从当前位置画一条线至 (x, y) 所指定的位置,并把笔的位置移至 (x, y)。

(8) MoveTo: 该方法的格式为:

```
MoveTo (x,y:Integer);
```

MoveTo 方法将笔的当前位置设置到点 (x, y) 处,笔的当前位置在 PenPos 属性中,改变笔的当前位置使用 MoveTo 方法,不要设法改变 PenPos 的值。

下面代码说明了如何使用 LineTo 和 MoveTo 方法。

```
procedure TForm1.FormMouseMove(Sender: TObject;Shift:TshiftState;
    x,y:Integer);
begin
    Canvas.FillRect(ClientRect);
    Canvas.MoveTo(10,10);
    Canvas.LineTo(x,y);
End;
```

(9) Pie: 该方法的格式为:

```
Pie(x1,y1,x2,y2,x3,y3,x4,y4:Longint);
```

Pie 方法用来绘制一个饼状图,它是椭圆的一部分,椭圆由点 (x1, y1), (x2, y2) 所指定的矩形所决定,绘制的那部分由椭圆中心到 (x3, y3), (x4, y4) 两点的两条辐射线所决定。

(10) Polygon: 该方法的格式为:

```
Polygon (Points:array of TPoint);
```

Polygon 方法在画布上绘制一系列的点,各点依次连成线,最后将首尾两点相接形成一个区域,并用当前笔刷填充此区域。

(11) Polyline: 该方法的格式为:

```
Polyline (Points:array of TPoint);
```

Polyline 方法在画布上用当前画笔绘制一系列的点，各点依次连成线。

下面代码说明了如何使用 Polyline 方法。

```
Procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with Sender as TpaintBox do
  begin
    Canvas.Pen.Color:=ClBlue;
    Canvas.Polyline([Point(50,20),Point(30,70),Point(80,40),
      Point(20,40),Point(70,70),Point(50,20)]);
  end;
end;
```

(12) StretchDraw: 该方法的格式为:

```
StretchDraw (Const Rect:Trect;Graphic:Tgraphic);
```

StretchDraw 方法在 Rect 参数指定的矩形内画一图像。图像延伸改变大小以适应矩形。

(13) Rectangle: 该方法的格式为:

```
Rectangle (x1,y1,x2,y2:Integer);
```

Rectangle 方法在画布上用当前画刷绘制矩形，(x1, y1) 是矩形的左上角，(x2, y2) 是矩形的右下角。

(14) DrawFocusRect: 该方法的格式为:

```
DrawFocusRect (Const Rect:Trect);
```

DrawFocusRect 方法绘制一矩形以指示此矩形获得焦点。此方法是异或 (XOR) 函数，第二次调用时原有矩形将消失。DrawFocusRect 绘制的矩形不能滚动。要实现滚动功能则先调用此方式使矩形消失，待滚动过后重新绘制。

2. Canvas 对象的属性

Canvas 对象还包含多种属性，供用户设置，说明如下:

(1) Brush: 这是只读属性，该属性用来返回画布的使用刷子，用户可以通过它设置填充的颜色或图案。

(2) CanvasOrientation: 声明:

```
TCanvasOrientation=(coLeftToRight,coRightToLeft);
CanvasOrientation: TCanvasOrientation;
```

这是枚举型只读属性，该属性用来设置画布的坐标。CoLeftToRight 表示从左到右的坐标，画布的坐标原点在左上角。coRightToLeft 表示从右到左的坐标，画布的坐标原点在右上角。

(3) ClipRect: 这是只读属性，该属性返回画布的矩形区域，超过这个区域的画布边缘将会被裁减掉。

(4) CopyMode: 该属性决定在调用 Canvas 的 CopyRect 方法时如何进行位组合，此属性用来设置拷贝图像的方式，其默认值是 cmSrcCopy，拷贝图像将会覆盖画布上的原有图像。画布的属性值还有: cmBlackness, cmDstInvert, cmMergeCopy, cmMergePaint,

cmNotSrcCopy, cmNotSrcErase, cmPatCopy, cmPatInvert, cmPatPaint, cmSrcAnd, cmSrcCopy, cmSrcErase, cmSrcInvert, cmSrcPaint, cmWhiteness。通过改变 CopyMode 的这些属性值, 可以取得一些各种特殊效果。

(5) Font: 用于设置文字的字体、颜色、大小等特征。对 Font 属性的子属性赋值, 可以选择调用 Canvas 的 TextOut 和 TextRect 方法输出文本时的文字字体样式。该 Font 对象与窗体的 Font 属性无关, 在绘制文本时应先初始化 Canvas.Font 属性。设置 Font 属性指派具体的 TFont object 而不是取代当前的 TFont object。

(6) Handle: 对于那些需要设备环境句柄的 GDI 函数, 可以把 Canvas.Handle 传递给它们的 HDC 参数。这使得调用 Canvas 中没有包括的 GDI 函数成为可能。

(7) LockCount: 此属性用来返回程序中为防止线程冲突而锁定画布的次数。每当 Lock 方法锁定一次, LockCount 属性值就累加一次, 反之在调用 Unlock 方法解锁时 LockCount 属性会递减。当 LockCount 属性为 0 时, 线程就可以对画布操作了。

(8) Pen: 此属性是一个 Tpen 对象, 用来设置画布中使用的画笔特性。

(9) PenPos: 此属性是一个 Tpoint 对象, 用于返回画布的当前位置。这个值是只读的, 如果用户要改变画笔的当前位置则应该使用 MoveTo 方法。

(10) Pixels: 该属性是一个二维数组, 用于存放 Canvas 的每个像素对应的颜色值。我们只有在必须访问单个像素的颜色时才使用 Pixels 数值。

3. Canvas 对象的常用事件

(1) OnChange: 这个事件在画布内容被改变的时候触发。

(2) OnChanging: 这个事件在画布内容正在被改变的时候触发。此事件与 OnChange 事件在事件触发顺序上有一定的先后次序, 当用户改变画布内容时这两个事件发生先后如下:

- OnChanging 事件触发。
- 画布中的内容被改变。
- OnChange 事件发生。

8.1.2 画刷 (Brush)

Brush 拥有一个画刷句柄 (Hbrush), 用于指定填充画布上的区域和图形时所使用的颜色和图案。该对象在 Graphics 单元中声明。下面介绍 Brush 对象的主要属性、方法和事件。

1. Brush 对象的属性

我们在前面的【例 8-4】中已经使用过 Brush 对象的一些属性和方法了, 这里再来详细介绍一下:

- Color 属性: 用来指定刷子的颜色。
- Style 属性: 用来指定填充图形所使用的图案。图 8-6 说明了 Style 不同取值情况下的填

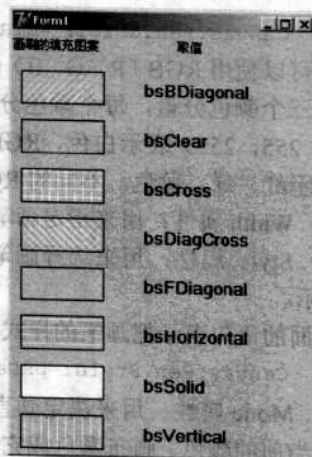


图 8-6 Style 不同取值情况下的填充效果

充效果。

- **Bitmap** 属性: 用来指定一个 8×8 的位图, 画布的刷子以它为图案重复绘制填满图形的内部区域。

默认情况下, 刷子颜色为 `clWhite`, 填充图案为 `bsSolid`, 没有位图。可以改变颜色和填充图案, 以使用不同的图案来填充区域。

2. Brush 对象的方法

- **Assign** 方法: 在一个 Brush 对象中向另一个对象赋值。
- **Create** 方法: Brush 的构造函数, 用来创建一个 Brush 对象实例。
- **Destroy** 方法: Brush 的析构函数, 用于释放一个 Brush 对象实例。
- **OwnHandle** 方法: 用来确保对更基本的 QBrush 类实例的权限。当需要接受 QBrush handle 使用或销毁权限时用到这个过程。
- **ReleaseHandle** 函数: 用来把 Brush 从 QBrush handle 中分离出来, 当需要把 TBrush handle 给一段程序或一个类时, 用到这个函数。

3. Brush 对象的事件

OnChange 事件在 Brush 对象中的图像被改变的时候触发。

8.1.3 画笔 (Pen)

画笔可以在画布上画线条, 它指明了线条的颜色和式样以及几何形状的轮廓线属性。Delphi 中可以通过 `Canvas.Pen` 属性来访问画笔。

1. Pen 对象的属性

画笔可以用 `Color`、`Width`、`Style` 和 `Mode` 属性来描述它的状态。

- **Color** 属性: 用来指定画笔的颜色。可以使用 Delphi 提供的预定义的颜色常量, 例如常量 `clRed` 和 `clYellow` 分别对应着红色和黄色。下面的代码把画笔的颜色设为红色:

```
Canvas.Pen.Color := clRed;
```

还可以使用 `RGB (R, G, B)` 函数表示颜色, 其中 R、G 和 B 分别表示这种颜色的红、绿、蓝三个颜色分量, 每个颜色分量有 255 个值, 例如 `RGB (0, 0, 0)` 表示黑色, `RGB (255, 255, 255)` 表示白色, `RGB (255, 0, 0)`、`RGB (0, 255, 0)`、`RGB (0, 0, 255)` 分别表示红、绿、蓝色。不同的 RGB 分量组合可以有不同的颜色。

- **Width** 属性: 用来指定画笔的笔宽, 即直线和轮廓线的宽度是多少像素。
- **Style** 属性: 用来选择画笔的线段类型。Style 属性定义了线段的各种类型, 如表 8-1 所示。

下面的语句表示把画笔的样式设置为点划线:

```
Canvas.Pen.Style := psDashDot;
```

- **Mode** 属性: 用来决定画直线和轮廓线的颜色, 它是一个 `Tpenmode` 的集合属性, 可结合当前的颜色、画布颜色或它们反向值对线段的颜色重新定义, 但不改变 `Color` 属性。Mode 属性的取值如表 8-2 所示。

表 8-1 Style 属性的取值以及含义

取 值	含 义
psSolid	画固定线段
psDash	画由下划线组成的线段
psDot	画由点组成的线段
psDashDot	画点划线
psDashDotDot	画双点划线
psClear	画看不见的线段
psInsideFrame	画边界的矩形线框

表 8-2 Mode 属性的取值以及含义

取 值	含 义
pmBlack	黑色
pmWhite	白色
pmNop	不变
pmNot	把画布颜色反向
pmCopy	使用 Color 属性中的颜色
pmNotCopy	画笔颜色的反向值
pmMergePenNot	画笔的颜色与画布颜色反向值的操作
pmMaskPenNot	画笔和画布反向的共同颜色值的操作
pmMergeNotPen	画布颜色与画笔颜色反向值的操作
pmMaskNotPen	画布颜色与画笔反向值共同颜色的操作
pmMerge	画笔颜色和画布颜色的操作
pmNot Merge	PmMerge 的反向值
pmMask	画布和画笔共同颜色的操作
pmNotMask	PmMask 的反向值
pmXor	对画布或者画笔进行抑或操作
pmNotXor	pmXor 的反向值

2. Pen 对象的方法

● **Assign** 过程：这个过程用于把指定的对象拷贝到目标对象中。如果指定的对象是 Pen 画笔对象，Assign 过程把源对象中的 Color 属性、Width 属性、Style 属性、Mode 属性赋给目标对象。如果指定的对象不是 Pen 画笔对象，则 Assign 过程将会执行继承的方法，像调用 AssignTo 方法一样把对象的属性拷贝到目标 Pen 对象中。

● **Create**：这是 Pen 对象的构造函数，用于建立一个 Pen 对象实例。用户在应用中不可以直接调用这个函数来创建对象实例，因为当对象建立的时候，Delphi 7 会为画布对象自动建立一个 Pen 对象。Create 函数在建立 Pen 对象实例时会为对象实例分配资源并把对象的 Mode 属性设为 pmCopy。

● **Destroy** 函数：这个方法是 Pen 对象的析构函数，用于释放一个 Pen 对象实例。不要直接调用 Destroy 函数来析构一个 Pen 对象实例，Delphi 7 推荐用户使用 Free 方法。因为 Free 方法在析构一个 Pen 对象实例的同时还释放为其分配的资源。

3. Pen 对象的事件

OnChange 事件，在 Pen 对象中的图像被改变的时候触发。

8.1.4 颜色 (Color)

Color 类型定义一个对象的颜色。很多组件的颜色属性就是 Color 类型，在 Graphic 单元中 Color 定义如下：

```
Tcolor = -(COLOR_ENDCOLORS+1) .. $0FFFFFFF;
```

这是一个 32 位二进制数据。Graphic 单元中还定义了一些常用的颜色常量，这些常量或直接映射成系统调色板中最相近的颜色，或映射成 Windows 控制面板中颜色部分的系统视频颜色。

直接映射成系统调色板中的颜色有：

clAqua、clBlack、clBlue...

映射程序用 4 字节的二进制码来定义颜色。低 3 位字节代表 RGB 相应的颜色，如 \$00FF0000 表示纯蓝，\$0000FF00 表示纯绿，\$000000FF 表示纯红，\$00000000 表示黑色，\$00FFFFFF 表示白色。如果最高位字节是 \$00，则表示系统调色板中最相近的颜色；最高位字节是 \$01，则表示用当前调色板中最相近的颜色匹配；如果最高位字节是 \$02，则用当前设备描述中逻辑调色板次相近的颜色相匹配。

8.2 图形图像的种类和组件

在描述信息时，一幅好的图形图像会胜过千言万语，因而图形图像是使用最广泛的一种媒体，也是目前研究最深入的一种媒体。Delphi 中提供了许多功能强大的类和组件，使用它们能方便地开发出与图形图像有关的应用程序。

8.2.1 与图形图像有关的类

Delphi 中与图形图像有关的类有下面三个：

1. Graphic

Tgraphic 是 TBitmap、TIcon 和 TMetafile 类的基类。如果知道图像的具体类型（如位图、图标、元文件），则应将图像储存在相应类型的对象中（如 TBitmap、TIcon、TMetafile），否则应该使用可储存任何图像类型的 TPicture 的对象。

2. TPicture

TPicture 的对象可以保存位图、图标或元文件。Graphic 属性中包括图相的类型；图像的高度和宽度通过 Height 和 Width 属性描述；调用 LoadFromFile 方法。可以从文件中装载一幅图像，例如下面的代码：


```
Procedure TForm1.FormCreate(Sender:TObject);
begin
    BitBtn1.Glyph.LoadFromFile('风景.BMP');
end;
```

要保存一个位图,可使用 SaveToFile 方法;要把图像复制到剪切板,可以调用 Tclipboard 对象的 Assign 方法。

3. Tbitmap

位图图像能包含一幅位图图像,有 HBITMAP, HPALETTE 句柄,可自动管理调色板。位图对象有画布属性,可用于绘制图形。位图的 Palette 属性用来控制位图的颜色映射,它包括 256 种可显示的颜色。如果应用程序用前景色绘制位图,Palette 属性的颜色将被加入 Windows 系统调色板,其他颜色被映射到系统调色板已存在的颜色。如果应用程序用自己的颜色绘制位图,而其他程序已占有系统调色板,位图的颜色将被映射到系统调色板中。如果 Monochrome 属性设置成假,位图将显示成彩色,反之显示成黑白色。调用 Draw 和 StretchDraw 方法,可在画布上绘制位图。

8.2.2 Image (图像) 组件

Image 组件  是一个容器组件,位于 Additional 选项卡上,它在应用程序窗体窗口中提供了一个矩形区域,用于显示和输出位图、图标、图元文件或用户自定义的图形文件。在应用程序运行时它是不可见的。它主要具有以下这些属性:

- Align 属性: 该属性 Image 组件在窗体中的位置。
- AutoSize 属性: 该属性是布尔类型, AutoSize 属性被设置为 True 时, Image 组件的显示区域可以随着调入图像的大小发生变化,也就是说自动调整显示区域来显示整幅图像。
- Canvas 属性: 该属性指出了 TImage 进行绘图操作的平面。
- Center 属性: 该属性用来确定图像是否显示在组件的中央。
- Picture 属性: 该属性用来在 Image 组件的显示区域中显示图像。

可以在程序设计阶段指定图像: 在对象查看器 (Object Inspector) 中设置 Image 组件的【Picture】属性,即单击【Picture】属性右边的省略号,在出现的【Picture Editor】对话框中单击【Load】按钮装入要显示的图像。如图 8-7 所示。

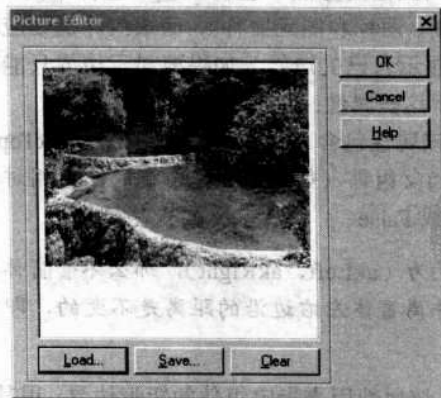


图 8-7 Picture Editor 对话框

也可以在程序运行阶段调入一幅图像,这要用到 Picture 的 LoadFromFile () 方法。例

如需要调入一位图，可使用下面的语句：


```
Image1.Picture.LoadFromFile('C:风景1.bmp');
```

当然，LoadFromFile() 方法能够使用变量作为参数来动态地调入图像。

- **Stretch 属性**：该属性是布尔类型，Stretch 属性被设置为 True 时，Image 组件将图像自动地放大或缩小以适应 Image 组件的显示区域。

- **Transparent 属性**：该属性是布尔类型，Transparent 属性被设置为 True 时，图像以透明的方式显示。

8.2.3 Shape (形状) 组件

Shape 组件 ，位于 Additional 选项卡上，用于在窗体中绘制几何图形，如椭圆、矩形和圆角矩形等。它主要具有以下这些属性：


- **Shape 属性**：该属性用来选择需要绘制的几何图形，它的取值如表 8-3 所示。

表 8-3 Shape 属性的取值以及含义

取 值	含 义
stCircle	圆形
stEllipse	椭圆形
stRectangle	矩形
stRoundRect	圆角矩形
stRoundSquare	圆角正方形
stSquare	正方形

- **Brush 属性**：该属性用来选择在几何图形中填充的样式。
- **Pen 属性**：该属性用来选择几何图形所使用的线型。

8.2.4 PaintBox (工具箱) 组件

PaintBox 组件 ，位于 System 选项卡上，是一个简单的画板，它为应用程序提供了可在窗体的特定矩形区域内画图的方法。它和 Image 组件不同。Image 组件是显示已经保存在文件中的图像（如位图、图标和图元文件），而该组件需要应用程序直接在窗体上进行绘制。PaintBox 组件不能显示自己的边界，如果要显示边界只能将它放在别的组件当中。PaintBox 组件主要具有以下这些属性：

(1) **Anchors 属性**：该属性包含 4 个子属性 (akLeft、akTop、akRight 和 akBottom)，用来指定在运行过程中，当父组件（如窗体）是可调时，它相对于父组件边沿的位置。每个子属性只能设置为 True 或 False。

注意：假如将该属性设置为 (akLeft、akRight)，那么不管窗体组件在左右方向的宽度如何调整，该组件离窗体左右边沿的距离是不变的，即它跟着窗体组件的宽度变大或者变小。

(2) **BoundsRect 属性**：该属性用来指定组件的矩形边界，用其父组件的坐标系来表示。

[例 8-5] BoundsRect 属性的使用。

效果：利用命令按钮演示 BoundsRect 属性。执行界面，如图 8-8a 所示。点击按钮，按钮被拉长变细，如图 8-8b 所示。

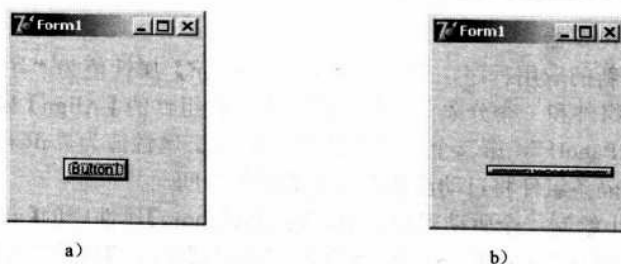


图 8-8 按钮变形效果

程序代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyRect:TRect;
begin
  MyRect:=ActiveControl.BoundsRect;
  MyRect.Right:=MyRect.Left+2*(MyRect.Right-MyRect.Left);
  MyRect.Bottom:=MyRect.Top+(MyRect.Bottom-MyRect.Top) div 2;
  ActiveControl.BoundsRect:=MyRect;
end;
```

(3) Canvas 属性：该属性仅在运行时有效。使用该属性可以在组件表面的特定区域进行绘图，它提供了完成绘图的所有属性和方法。

(4) Color 属性：该属性可用于除颜色对话框以外的所有组件或对象。该属性定义了组件的背景颜色。如果该组件的 ParentColor 属性为 True，则改变其父组件的 Color 属性将自动影响该组件的 Color 属性。当该组件的 Color 属性不同于其父组件时，其 ParentColor 属性将自动设置为 False。

(5) Constraints 属性：该属性用来指定组件的大小。它的取值以及含义如表 8-4 所示。

表 8-4 Constraints 属性的取值以及含义

取值	含义
MaxHeight	指定组件的最大高度
MaxWidth	指定组件的最大宽度
MinHeight	指定组件的最小高度
MinWidth	指定组件的最小宽度

8.3 图形图像编程应用

在本节中将通过几个实际例子说明如何使用 Delphi 7 的 Canvas 画布对象和图形图像组件进行编程。从中读者可以感受到 Delphi 7 的强大功能。

8.3.1 图片浏览器的设计

将以一个图片浏览器为例，讲解 Delphi 7 中提供的图形图像组件的应用。

[例 8-6] 图片浏览器

(1) 创建一个新的应用程序，设定窗体的【Caption】属性值为“图片浏览器”。在窗体上绘制两个面板组件和一个分隔条组件，第一个面板组件的【Align】属性值为“alLeft”，【Name】属性为“Panel1”，第二个面板组件的【Align】属性值为“alClient”，【Name】属性为“Panel2”，分隔条组件将自动出现在两个面板的中间。

(2) 在 Panel1 上绘制一个驱动器组合框，设定【Anchors】属性中的【akLeft】、【akRight】、【akTop】属性值为“True”，【akBottom】属性值为“False”，【Name】属性为“DriveComboBox”。

(3) 在 Panel1 上绘制一个目录列表框，设定【Anchors】属性中的【akLeft】、【akRight】、【akTop】、【akBottom】属性值为“True”，【Name】属性为“DirectoryListBox”。

(4) 设定 DriveComboBox 组件的【DirList】属性值为“DirectoryListBox”，把驱动器列表组件和目录列表组件关联起来。

(5) 在 Panel2 上绘制一个面板，设定面板的【Align】属性值为“alTop”，【Name】属性为“Panel3”。

(6) 在 Panel3 上绘制一个文件类型组合框，设定【Anchors】属性中的【akLeft】、【akRight】、【akTop】属性值为“True”，【akBottom】属性值为“False”，对于 Filter 属性如图 8-9 所示进行设定。其中所有图片所对应的值为“*.BMP; *.ICO; *.WMF; *.EMF; *.JPG; *.JPEG”，设定【Name】属性为“FilterComboBox”。

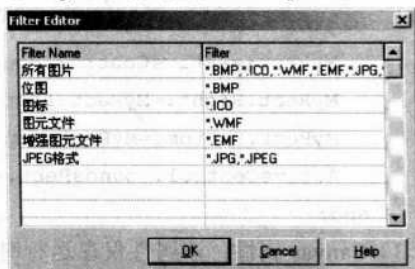


图 8-9 设定文件类型

(7) 在文件类型组合框的右侧绘制一个复选框，设定【Anchors】属性中【akRight】、【akTop】属性值为“True”，【akLeft】、【akBottom】属性值为“False”。【Caption】属性值为“自动缩放”，【Name】属性值为“CheckBoxStretch”。

(8) 在复选框的右侧绘制一个命令按钮，设定【Anchors】属性中【akRight】、【akTop】属性值为“True”，【akLeft】、【akBottom】属性值为“False”。【Caption】属性值为“全屏查看”，【Name】属性值为“ButtonFull”。

(9) 在 Panel3 的下侧绘制一个文件列表框，设定【Anchors】属性中【akRight】、【akTop】、【akLeft】、【akBottom】属性值为“True”，【Name】属性值为“FileListBox”。设定 DirectoryListBox 的【FileList】属性值为“FileListBox”。FilterComboBox 的【FileList】属性值为“FileListBox”。在 Panel2 上绘制一个分隔条，设定【Align】属性值为“alTop”。

(10) 在 Panel2 上绘制一个图像组件。设定 Align 属性值为“alClient”，Name 属性值为“Image”。对象之间的包含关系，如图 8-10 所示。

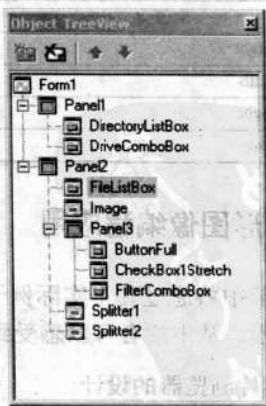


图 8-10 对象之间的包含关系

(11) 当前窗体的设计效果如图 8-11 所示。

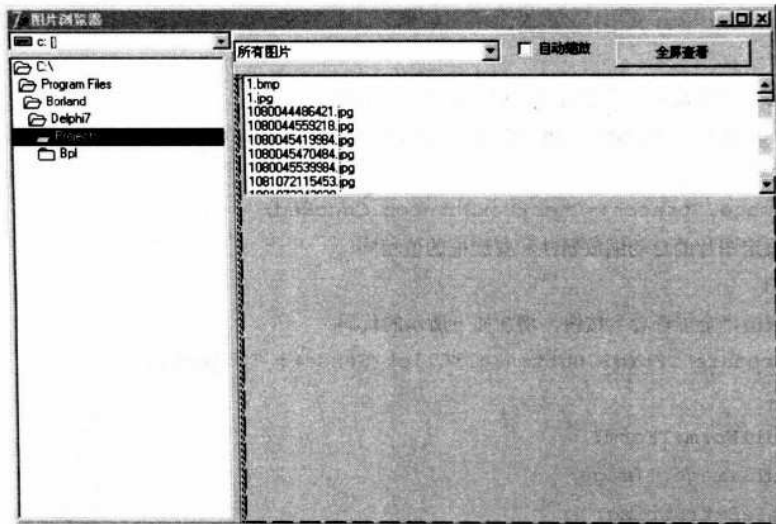


图 8-11 窗体的设计效果

(12) 为窗体各组件的操作添加程序代码，以实现相应的功能：

```

... .. //为节约篇幅，省略了系统自动创建的代码
implementation
{$R *.dfm}
//FileListBox 的 OnChange 事件的响应过程，代码如下：
procedure TForm1.FileListBoxChange(Sender: TObject);
var
    FileExt:String;
begin
    //取选中文件名的扩展名，并转化为大写字母
    FileExt:=UpperCase(ExtractFileExt(FileListBox.FileName));
    //根据扩展名，判断是否可以处理的文件类型
    If (FileExt='.BMP')or(FileExt='.ICO') or (FileExt='.WMF') or
        (FileExt='.EMF') or (FileExt='.JPG') or (FileExt='.JPEG') then
        begin
            //扩展名符合要求，把文件读入图像组件中
            Image.Picture.LoadFromFile(FileListBox.FileName);
            //在窗体的标题中显示文件名以及文件的尺寸
            Self.Caption:='图片浏览器-'+ ExtractFileName(FileListBox.FileName)
                + Format('%dX%d',[Image.Picture.Width,Image.Picture.Height]);
            if FileExt='.ICO' //判断文件的扩展名是否为 ICO (图标文件) then
                Self.Icon:=Image.Picture.Icon;
        end
    end;
end;

```



```

//是图标文件，设定窗体的图标为选中的图标
    end;
end;
//双击“自动缩放”复选框，增加如下所示的代码：
procedure TForm1.CheckBoxStretchClick(Sender: TObject);
begin
    Image.Stretch:=CheckBoxStretch.Checked;
//设定图片的自动缩放属性和复选框的值相同
end;
//双击“全屏查看”按钮，增加如下所示的代码：
procedure TForm1.ButtonFullClick(Sender: TObject);
var
    FullForm:TForm;
    FullImage:TImage;
    FileExt:String;
begin
    FileExt:=UpperCase(ExtractFileExt(FileListBox.FileName));
//获取扩展名
//判断是否为可以显示的图片类型
    if (FileExt='.BMP')or(FileExt='.ICO')or(FileExt='.WMF')or
        (FileExt='.EMF')or(FileExt='.JPG')or(FileExt='.JPEG') then
    begin
        FullForm:=TForm.Create(Self); //创建一个窗体对象
        FullImage:=TImage.Create(FullForm);
//在创建时，设定窗体对象的父对象为当前窗体，这
//样在释放当前窗体时将自动释放这个窗体，然后
//在新的窗体中创建一个图像对象
        FullForm.Caption:=Self.Caption;
//设定新窗体的标题和当前窗体的标题相同，也就
//是文件名和文件的尺寸
        FullImage.Parent:=FullForm; //设定图像对象的父对象为新窗体
        FullImage.Align:=alClient; //设定图像对象的对齐属性值为客户区，也就是充
//满全部的新窗体范围
        FullImage.Picture.LoadFromFile(FileListBox.FileName);
//让图像对象读取图像文件
        FullForm.WindowState:=wsMaximized; //设定新窗体为全屏
        FullForm.Visible:=true; //显示新窗体
    end;
end;

```

end.

(13) 运行程序, 此时鼠标可以拖动分隔条改变窗体的布局, 并浏览图片。

使用驱动器组合框, 文件夹列表框和文件类型组合框可以选择目录和文件类型, 由于这几个组件之间设置了关联, 因此不需要进行编码。程序的功能如下说明:

- 选择一个包含图片的目录, 结果如图 8-12 所示。

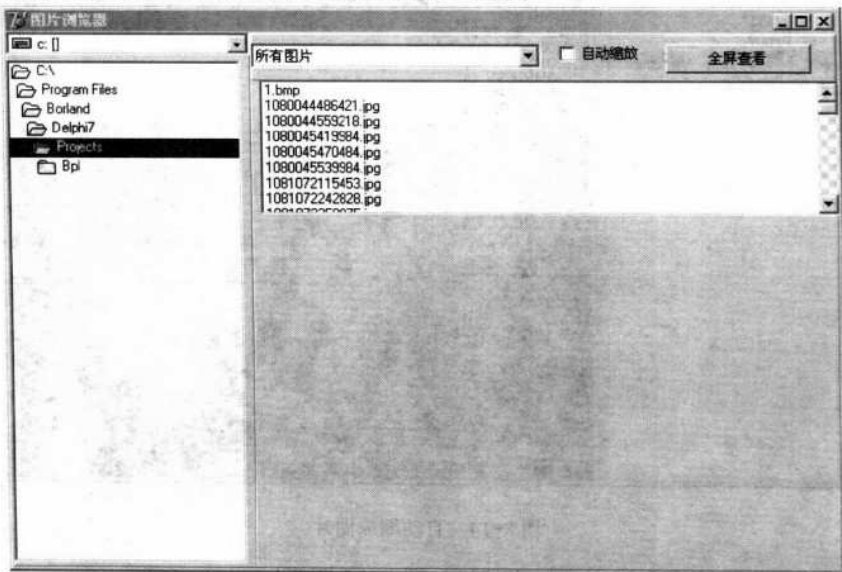


图 8-12 包含图片的目录

- 选择图片 1.bmp, 执行结果如图 8-13 所示。



图 8-13 显示图片

● 选择【自动缩放】复选框后，复选框的值变为真，所以图像组件设置为自动缩放，运行结果如图 8-14 所示。



图 8-14 自动缩放图片

- 单击【全屏查看】按钮，运行结果如图 8-15 所示。
- 选择另一个图片，单击【全屏查看】按钮，运行结果如图 8-16 所示(自由调整窗体)。



图 8-15 全屏查看图片

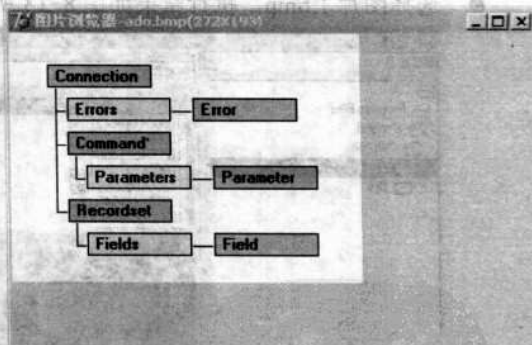


图 8-16 自由调整窗体的尺寸

8.3.2 绘制余弦曲线

如何使用函数绘制余弦曲线呢？我们可以利用 Canvas 对象的 Pixels 方法，通过对余弦曲线的算法可以实现对余弦曲线的绘制。

下面的实例说明如何实现余弦曲线的绘制。

【例 8-7】实现余弦图线的绘制

效果：在窗体中显示余弦曲线的绘制。执行效果如图 8-17 所示。

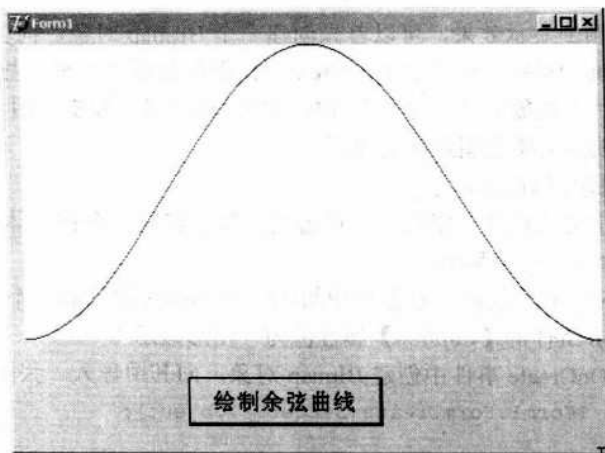


图 8-17 程序的执行效果

- (1) 新建一个应用程序工程，在窗体中加入一个 Button 组件和一个 Image 组件。
- (2) 设置 Button 组件的【Caption】属性值为“绘制余弦曲线”。
- (3) 双击 Button 组件，OnClick 事件代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,l:Integer;
  y,a:Double;
begin
  Image1.Picture.Bitmap:=TBitmap.Create;           //生成图片对象
  Image1.Picture.Bitmap.Width:=Image1.Width;
  Image1.Picture.Bitmap.Height:=Image1.Height;
  l:=Image1.Picture.Bitmap.Width;
  for x:=0 to l do                                //开始绘制图形
  begin
    a:=(x/l)*2*Pi;
    y:=cos(a);
    y:=y*(Image1.Picture.Bitmap.Height/2);
    y:=y+(Image1.Picture.Bitmap.Height/2);
    Image1.Picture.Bitmap.Canvas.Brush.Style:=bsSolid;
    Image1.Picture.Bitmap.Canvas.Pixels[Trunc(x),Trunc(y)]:=clred;
  end;
end;
```

- (4) 运行程序。单击【绘制余弦曲线】按钮，在窗体中会绘制一条红色的余弦曲线。

8.3.3 图片颠倒显示

要实现图片的颠倒显示效果,可以首先创建一个 `Bitmap` 对象,向其中导入一张图片,然后把该图片放到剪切板中,再通过 `PaintBox` 组件把剪切板中的图片显示出来,在显示的过程中可以指定图片的摆放方式,通过对摆放方式的设置就可以实现图片的翻转。

下面的实例说明如何实现图片颠倒显示。

[例 8-8] 实现图片颠倒显示

效果:在窗体中首先放置一张位图,单击图形显示按钮,会在窗体中显示该图片的翻转图片。执行效果如图 8-18 所示。

- (1) 新建一个应用程序工程,在窗体中加入一个 `Button` 组件和两个 `PaintBox` 组件。
- (2) 设置 `Button` 组件的【Caption】属性值为“图形显示”。
- (3) 在窗体的 `OnCreate` 事件中创建 `Bitmap` 对象,向其中导入一张图片。代码如下:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  b:TBitmap;
begin
  b:=TBitmap.Create;
  b.LoadFromFile('1.bmp');
  PaintBox1.Width:=b.Width;
  PaintBox1.Height:=b.Height;
  PaintBox2.Width:=b.Width;
  PaintBox2.Height:=b.Height;
end;
```



图 8-18 程序的执行效果

- (4) 在 `Button` 组件的 `OnClick` 事件中添加代码,向两个 `PaintBox` 组件添加图片。代码如下:

```

procedure TForm1.Button1Click(Sender: TObject);
var
    b:TBitmap;
begin
    PaintBox1.Canvas.Draw(0,0,b);
    b.Canvas.CopyRect(Rect(PaintBox2.Width,PaintBox2.Height,0,0),
        b.Canvas,Rect(0,0,b.Width,b.Height));
    PaintBox2.Canvas.CopyRect(Rect(PaintBox2.Width,0,0,PaintBox2.Height),
        b.Canvas,Rect(0,0,b.Width,b.Height));
end;

```

- (5) 在 PaintBox2 组件的 OnClick 事件中添加代码, 实现图片翻转, 代码如下:

```

procedure TForm1.PaintBox2Click(Sender: TObject);
var
    b:TBitmap;
begin
    b.Canvas.CopyRect(Rect(PaintBox2.Width,PaintBox2.Height,0,0),
        b.Canvas,Rect(0,0,b.Width,b.Height));
    PaintBox2.Canvas.CopyRect(Rect(PaintBox2.Width,0,0,PaintBox2.Height),
        b.Canvas,Rect(0,0,b.Width,b.Height));
end;

```

- (6) 响应窗体的 Destroy 事件, 释放变量, 代码如下:

```

procedure TForm1.FormDestroy(Sender: TObject);
begin
    b.Free;
end;

```

- (7) 运行程序。单击图形显示按钮, 在窗体中显示该图片翻转的图像。

8.3.4 图像的特殊显示效果

除了按常规方式显示图像外, 我们还可以采用特殊的方式显示图像, 例如马赛克方式、从中心展开、向两边展开、向中心缩进、上拉和下拉等。实现特殊方式显示图像, 主要是按照一定规律部分拷贝图像到要显示的位图上。

采用特殊的方式显示图像需要对图像进行拷贝, Delphi 7 的 Canvas 对象提供了拷贝位图的功能。Canvas 对象中与拷贝有关的是拷贝模式 CopyMode 属性和矩形拷贝 CopyRect 方法。

下面实例使用 CopyRect 方法实现一个从左到右拉动显示图像的特技效果。

[例 8-9] 图像拉幕效果

- (1) 创建一个应用程序工程, 在窗体中添加一个 Button 组件。

(2) 设置 Button 组件的 Caption 属性为“图像特技效果”。

(3) Button 组件对象的 OnClick 事件的响应代码如下:

```

procedure TForm1.Button1Click(Sender: TObject);
const
    Step=200;                //决定拉动速度
    x0=20;
    y0=20;                   //图像在窗体中显示时以(x0,y0)
                             //作为左上顶点坐标
var
    Bitmap:TBitmap;
    Midx:Integer;            //用于存放当前步骤所拷贝位图的
                             //宽的中间变量
    RatioX:Real;             //每步位图宽度增加量
    I:Integer;
    Rect1,Rect2:TRect;       //源、目标矩形区域
begin
    Bitmap:=TBitmap.Create;
    Bitmap.LoadFromFile('1.bmp');    //装入系统中存在的一个位图文件
    RatioX:=Bitmap.Width/Step;
    for i:=0 to Step do
    begin
        Midx:=Round(RatioX*I);
        With Rect1 do            //源矩形区域(在源图像中)
        begin
            Left:=Bitmap.Width-Midx;
            Top:=0; Right:=Bitmap.Width; Bottom:=Bitmap.Height;
        end;
        with Rect2 do            //目的矩形区域(在窗体中)
        begin
            Left:=x0;
            Top:=y0;
            Right:=x0+Midx;
            Bottom:=y0+Bitmap.Height;
        end;
        Canvas.CopyRect(Rect2,Bitmap.Canvas,Rect1); //拷贝位图
    end;
    Bitmap.Free;                //释放位图资源
end;
end.

```


(4) 运行程序。当单击 Button 组件后, 位图在窗体中显示时从左侧出现并逐渐向右侧移动, 好像位图被从左向右拉动一样, 拉动结束后位图左上角坐标为 (x0, y0)。执行结果如图 8-19 所示。

8.3.5 图像格式转换器

保存图像有多种格式供选择, 其中用得最多的是 BMP 文件格式和 JPEG 文件格式。BMP 是 Windows 的标准图像格式, 而 JPEG 是 Internet 中最常用的图像格式, Internet 中不使用 BMP 图像。

我们设计图像格式转换器, 能够浏览 BMP 图像和 JPEG 图像, 还可以实现两种图像格式的相互转换。

[例 8-10] 图像格式转换器

(1) 新建一个应用程序工程。在新的窗体中添加两个 Button 组件, Button1 组件的【Caption】属性值为“浏览图像”, Button2 组件的【Caption】属性值为“格式转换”。

(2) 在新的窗体中再添加一个 OpenFileDialog1 (文件打开对话框) 组件、一个 SaveDialog1 (文件保存对话框) 组件、一个 Image1 组件、一个 Panel1 组件, Image1 组件放在 Panel1 组件的上面, 调整 Image1 组件的大小, 让它添满整个 Panel1 的区域。设计界面如图 8-20 所示。

(3) 由于指定图像后才能进行格式转换, 因而在执行图像之前【格式转换】按钮是不可见的, 在对象查看器中将 Button2 的 Enabled 属性设置为 False 来实现。

图像格式转换器只浏览和转换 BMP 格式与 JPEG 格式的图像文件。在打开和保存文件时, 文件对话框就只需要列出这两类文件, 通过对组件的 Filter 的设置可以控制文件类型。

(4) 在对象查看器中选择 OpenFileDialog1 或 SaveDialog1 的 Filter 属性, 打开【Filter Editor】对话框, 将 OpenFileDialog1 的【Filter Editor】对话框按图 8-21 进行设置, 将 SaveDialog1 的【Filter Editor】对话框按图 8-22 进行设置。

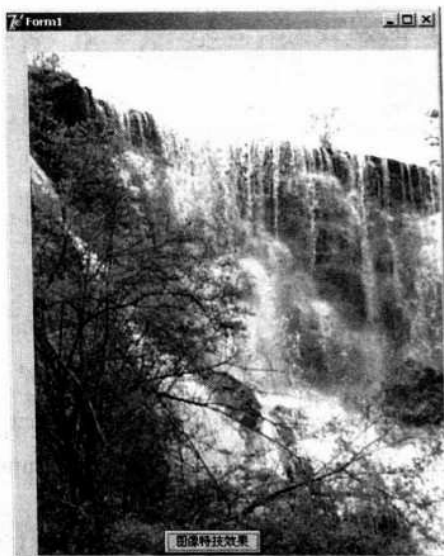


图 8-19 图像拉动效果

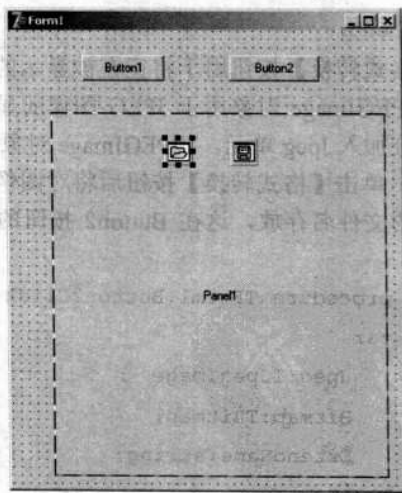


图 8-20 图像格式转换器界面

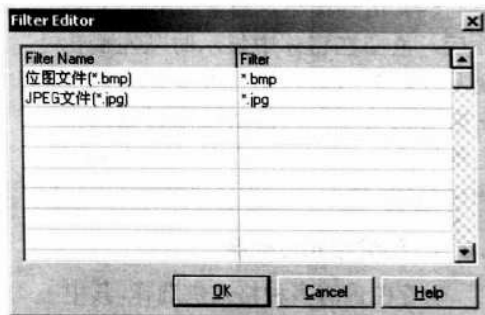
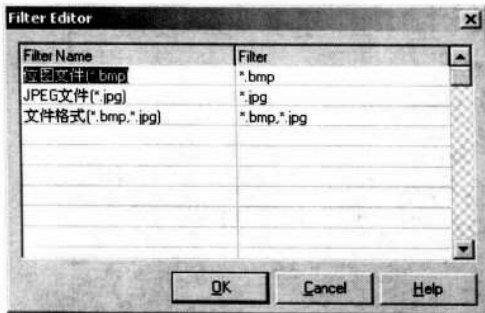


图 8-21 OpenDialog1 的 Filter Editor 对话框设置 图 8-22 SaveDialog1 的 Filter Editor 对话框设置

(5) **【浏览图像】**按钮用于弹出文件打开对话框，并将在文件打开对话框中选定的图像文件显示在 Image1 组件对象中，这在 Button1 按钮的 OnClick 事件的处理过程中实现，程序代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if OpenDialog1.Execute then //如果文件打开对话框被成功打开
    begin
        Image1.Picture.LoadFromFile(OpenDialog1.FileName);
        //将被选中的图像文件在 Image1 组件的显示区域中显示
        Button2.Enabled:=True; //将“格式转换”按钮设置为可用
    end;
end;
```

【格式转换】按钮用于将正在被显示的图像文件以另一种格式保存。在 Delphi 7 中有一个 TJPEGImage 对象用于 JPEG 图像的显示和处理，在使用这个对象时要在程序的 uses 语句部分加入 Jpeg 单元。TJPEGImage 对象的使用与 TBitmap 类似。

(6) 单击 **【格式转换】**按钮后将对图像文件格式进行转换，并将它以文件保存对话框中指定的文件名存放，这在 Button2 按钮的 OnClick 事件的处理过程中实现，程序代码如下：

```
procedure TForm1.Button2Click(Sender: TObject);
var
    Jpeg:TJpegImage ;           //TJpegImage
    Bitmap:TBitmap;             //TBitmap 对象
    ExtendName:string;          //用于保存文件的扩展名'.bmp'或'.jpg'
begin
    ExtendName:=ExtractFileExt(OpenDialog1.FileName);
    //得到当前正在显示的图像文件的扩展名
    if ExtendName='.bmp' then
```

```

begin
    Jpeg:=TJPEGImage.Create;
    Jpeg.assign(Image1.Picture.Bitmap);
    SaveDialog1.DefaultExt:='.jpg';
    if SaveDialog1.Execute then
        Jpeg.savetofile(SaveDialog1.FileName);
        Jpeg.free;
    end
    else
        begin
            Bitmap:=TBitmap.Create;
            Bitmap.assign(Image1.Picture.Graphic);
            SaveDialog1.DefaultExt:='.bmp';
            if SaveDialog1.Execute then
                Bitmap.savetofile(SaveDialog1.FileName);
                Bitmap.free;
            end;
            Button2.Enabled:=False;
        end;
end;

```

(7) 程序的运行界面如图 8-23 所示。

程序基本操作如下:

- 单击【浏览图像】按钮将弹出文件打开对话框, 可以浏览 BMP 格式和 JPEG 格式的图像文件。如图 8-24 所示。



图 8-23 运行界面



图 8-24 打开文件对话框

- 单击【格式转换】按钮可以打开文件保存对话框, 将正在浏览的图像以另一种文

件格式按指定的文件名存放。如图 8-25 所示。



图 8-25 文件保存对话框

8.4 使用鼠标绘制图形

图形通常是应用程序在执行时由用户手工动态绘制，此时鼠标是首选的绘图工具，应用程序利用鼠标位置的变化来绘制各种不同的图形。

鼠标有三个动作：按下鼠标按钮、移动鼠标和释放鼠标按钮。这三个动作执行时会分别触发三个不同的事件，它们是 OnMouseDown（按下鼠标按钮）、OnMouseMove（移动鼠标）、OnMouseUp（释放鼠标按钮）。在 Delphi 7 自动生成的鼠标事件处理过程的框架中，过程具有 5 个参数，例如窗体中 Image1 组件对象的 OnMouseDown 事件处理过程的框架如下：

```
procedure TForm1.Image1MouseDown(Sender:TObject;Button:TMouseButton;
    Shift:TShiftState;x,y:Integer);
begin
    ..... //添加必要的语句
end;
```

使用这些参数可以得到鼠标的一些消息。参数及其功能说明如表 8-5 所示。

表 8-5 鼠标事件的五个参数说明

参 数	说 明
Sender	探测鼠标动作的对象
Button	涉及的鼠标按钮：左键、中键还是右键
Shift	鼠标动作时，Alt、Ctrl 和 Shift 按钮的状态
x, y	事件发生时鼠标的坐标

下面我们通过鼠标画直线来展示使用鼠标绘制图形的方法。

【例 8-11】利用鼠标以橡皮筋方式画直线

效果：在鼠标按键释放之前，所画直线像一条橡皮筋随着鼠标的移动而伸缩。鼠标按键释放后所画直线即为最终结果，中间过程都被屏蔽掉了。执行效果如图 8-26 所示。

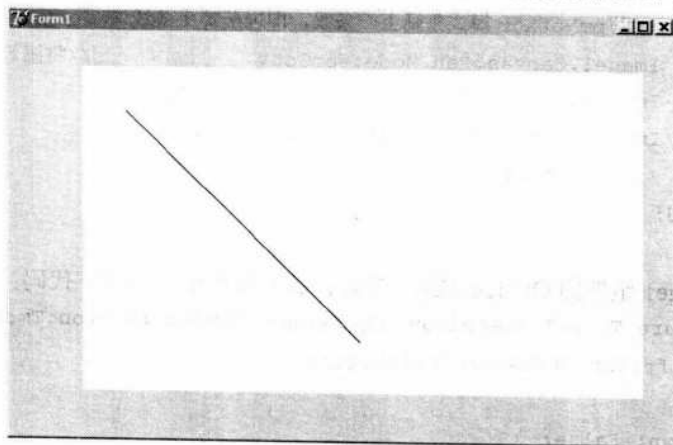


图 8-26 利用鼠标以橡皮筋方式画直线

(1) 新建一个应用程序工程。在窗体中放置一个 Image 组件，该组件对象的默认名为“Image1”。

(2) 在单元文件的 interface 的 var 中增加一个变量，该变量用于记录鼠标移动过程中所画的上一条线的终点坐标。

```
prior:Tpoint;
```

(3) 在 Image1 组件的 OnMouseDown (按下鼠标按钮) 事件中添加如下代码：

```
procedure TForm1.Image1MouseDown(Sender:TObject; Button:TmouseButton;
  Shift:TshiftState; x,y:Integer);
begin
  IsDown:=True;
  Image1.Canvas.MoveTo(x,y);
  origin:=Point(x,y);           //鼠标按下时的坐标
  prior:=Point(x,y);           //初始化上条线的坐标，与 origin
                                //的值相同
end;
```

(4) 在 Image1 组件的 OnMouseMove (鼠标移动) 事件中添加如下代码：

```
procedure TForm1.Image1MouseMove(Sender:TObject; Shift:TshiftState;
  x,y:Integer);
begin
  if IsDown then                //只有鼠标的按键仍处于按下状态
                                //才进行
  begin
    Image1.Canvas.Pen.Mode:=pmNotXor //将画笔模式设置为 pmNotXor
```

```

Image1.Canvas.MoveTo(origin.x,origin.y);
Image1.Canvas.LineTo(prior.x, prior.y);
//以 pmNotXor 模式重画上一条线,即屏蔽掉上条线
Image1.Canvas.Pen.Mode:=pmCopy           //将画笔还原为正常模式
Image1.Canvas.MoveTo(origin.x,origin.y);
Image1.Canvas.LineTo(prior.x, prior.y);
//画最终的直线
end;
end;

```

(5) 在 Image1 组件的 OnMouseUp (鼠标释放) 事件中添加如下代码:

```

procedure TForm1.Image1MouseUp(Sender:TObject;Button:TMouseButton;
    Shift:TShiftState;x,y:Integer);
begin
    IsDown:=False;
end;

```

(6) 运行程序,在窗体中会绘制一条直线,中间过程被屏蔽,看不到了。

注意: 由于中间过程不需要显示,随着鼠标的移动,在画下一条直线时,应将画笔设置为 pmNotXor 方式,把上一条直线重画一次,即以屏幕颜色将上一条直线重画,这样中间过程就被屏蔽掉了。

上面是通过鼠标以橡皮筋方式手工画直线,使用鼠标以橡皮筋方式画其他图形的方法是一样的,即将画笔模式设置为 pmNotXor 将中间过程屏蔽掉。

8.5 小结

本章讲述 Delphi 7 的图形图像编程的基本知识,使读者对 Delphi 7 有初步的了解,领会 Delphi 7 的图形图像编程的技巧。

几乎在每一个可视化组件中都包含一个 Canvas 对象,通过调用该对象的方法可以实现对图形图像的各种操作。通过使用 Brush 对象可以设定不同的填充样式,利用 Pen 对象可设定不同的画笔风格。通过使用 Image 组件、PaintBox 组件可导入图像文件,利用 Shape 组件可在窗体中绘制几何图形。



第9章 多媒体技术

多媒体技术带来了 20 世纪 90 年代的一场革命，它是当今计算机产业发展的一个新领域。多媒体技术使计算机具有综合处理文字、图形、图像、声音和视频的能力，它以形象丰富的信息和方便的交互性，为计算机进入人类生活和生产的各个领域提供了极大的方便，同时也给人们的工作、生活和娱乐带来了深刻的变化。

媒体在计算机领域中有两种含义：一种是指用于存储信息的介质，如磁盘、磁带和光盘等；另一种是表示信息的载体，如文字、数字、图形、图像、声音、动画和视频等。多媒体技术中媒体指的是后者。

9.1 多媒体基础知识

字、数字、图形、图像、声音、动画和视频等这些多媒体信息在 Windows 系统中都是以文件的形式存放的，不同的媒体有不同的文件存放格式。下面是一些常用的与多媒体有关的文件格式。

9.1.1 图形/图像文件

1. BMP 文件

BMP 即位图文件，是 Windows 的标准图像文件格式。它以二进制位的形式存储图像。位图有两种类型：与设备有关的位图（DDB）和与设备无关的位图（DIB）。两者的区别是：

DDB 使用系统提供的调色板，而 DIB 使用自己的调色板（即文件中自带的调色板）。通常，DIB 更灵活更简单，但有时会影响性能。DDB 的速度更快，但不方便。

2. JPEG 文件

JPEG（Joint Photographic Experts Group）文件格式提供了一种存储深度位像素的有效方法，它的压缩比相当高。作为图像文件格式，目前它在 Internet 上得到了广泛的应用。与其他的图像文件格式相比，JPEG 的最大区别是它采用有损压缩。无损压缩指被压缩的信息解压后能准确再现压缩前的信息，而有损压缩则通过牺牲一部分的信息以达到较高的压缩比，但信息的这点失真对所获得的高压缩比来说是微不足道的。

9.1.2 音频文件

1. WAV 文件

WAV 文件即波形文件，是 Windows 中标准的声音文件格式。波形音频是多媒体计算

机中获得声音最直接最简便的方式。WAV 文件也是二进制格式的,类似于数字声音。WAV 文件的最大优点是它符合工业标准,到处都在使用。缺点是:WAV 文件的体积较大,会占用大量的存储空间。

2. MIDI 文件

MIDI (Musical Instrument Digital Interface) 乐器数字接口是一种连接电子音乐设备的协议,它包括两个部分:一是 MIDI 硬件接口标准,二是 MIDI 数据格式。与其他的音频功能(如 WAV)相比,MIDI 的特点是它处理音乐的模式不同,不是将声音进行编码而是将 MIDI 音乐设备上产生的每一个活动记录下来,记录下来的信息能在任何一台 MIDI 合成器上重复原来的演奏。相同的音频信息,采用 MIDI 文件存放时的容量要比采用 WAV 文件小得多。但是 MIDI 只能记录标准所规定的有限种乐器的组合,且 MIDI 的声音质量过多的依赖于 MIDI 硬件,难以产生真实的演奏效果。

3. AVI 文件

AVI (Audio-Video Interleave) 是一种同时存储音频和视频信息的常用文件格式。在这种文件中,音频、视频单独存放,文件的每一帧都有音频和视频信息,在播放时被实时地送往音频和视频硬件。

9.1.3 计算机的硬件配置

实现把声音、影像和音乐等多媒体特征添加到 Delphi 应用程序中,需要用到三种类型的文件:

- 含有 AVI 扩展名的文件生成影像。
- 含有 MID 扩展名的文件以乐器数字接口(即 MIDI)格式生成音乐。
- 含有 WAV 扩展名的文件可以通过 Microsoft 的 WAVE 技术录制声音。


这些文件通常都占有较大存储空间。例如,一个历时一分钟的影片,生成的 AVI 文件约需要 5M~10MB 的硬盘空间。WAVE 文件通常也比较大,典型的比例是一分钟的声音对应 1MB 的磁盘空间。与 WAVE 和 AVI 文件不同,MIDI 文件可以被很大程度地压缩,而仍能提供上乘质量的音响效果。Windows95/98 中有 MIDI 播放工具,使从 MIDI 文件播放声音变得更加轻松简便。

要实现多媒体功能,计算机需要配备有声卡和 CD-ROM。此外,还需要音箱、耳机、或者一个放大器。也可以直接把声卡的输出插座接到立体声系统上。所以需要 CD-ROM 光盘是因为它提供了存储大量信息的空间。而声卡中的集成电路可以实现高品质的声音的复制与再现,一般来说,要实现多媒体技术,计算机需要配置 16 位声卡、高速光驱,并配有一个较高质量的扬声器。

9.2 多媒体组件

与其他的程序设计语言如 Visual C++, Visual Basic 等相比,使用 Delphi 7 开发多媒体程序要容易得多。因为 Delphi 7 提供了多媒体组件,它使得我们不必与复杂的 Windows MCI 接口打交道就可以开发出功能强大的多媒体应用程序。

9.2.1 Animate (动画) 组件

使用 Delphi 7 组件面板上【Win32】选项卡上的 Animate 组件，很容易为程序添加动画。它可以播放任何无声的 AVI 剪辑文件（如果使用这个组件播放有声的 AVI 文件，Delphi 将提示错误），从而在窗口中提供动画显示。更主要的是，我们可以用该组件显示 Windows 提供的标准 AVI 动画文件，来美化程序界面。

使用 Animate 组件很简单，只需对放置在窗体上 Animate 组件的属性进行设置，Animate 组件在窗体中给出一个用来显示动画的窗体区域。

下面我们看看 Animate 组件的主要属性。

1. Animate 组件对象的主要属性

- **Active 属性**：该属性设置为 True 时，可以播放动画。在把 Active 设置为 True 之前，应先把所选的 Windows 提供的标准 AVI 动画的名字赋给 CommonAVI 属性，或者是把一般的 AVI 视频剪辑文件的文件名输入到 FileName 属性中。把 CommonAVI 设置为 aviNone，或把 FileName 属性清空，都将使 Active 的值变为 False。在程序运行时，可以通过检查 Active 属性来确定动画是否被播放。

- **AutoSize 属性**：该属性设置为 True 时，可以使程序根据播放的 AVI 动画文件画面的大小，自动调整 Animate 显示窗体区域的大小。

- **Center 属性**：该属性设置为 True 时，动画出现在 Animate 显示区域的中心。Center 属性设置为 False 时，动画出现在 Animate 显示区域的左上角。

- **CommonAVI 属性**：该属性用于选中一个标准的 WindowsAVI 动画文件。CommonAVI 属性的取值以及含义如表 9-1 所示。

表 9-1 CommonAVI 属性的取值以及含义

取值	含 义
aviNone	无 AVI 动画
aviFindFolder	查找文件夹的 AVI 动画
aviFindFile	查找文件的 AVI 动画
aviFindComputer	查找计算机的 AVI 动画
aviCopyFiles	多个文件拷贝的 AVI 动画
aviCopyFile	单个文件拷贝的 AVI 动画
aviRecycleFile	将文件放入回收站的 AVI 动画
aviEmptyRecycle	清空回收站的 AVI 动画
aviDeleteFile	删除文件的 AVI 动画

- **FileName 属性**：该属性用于指明需要显示的纯视频图像（没有声音）的 AVI 动画文件的文件名。它的值可以在程序设计阶段指定，也可以在程序运行时动态赋值。一旦给 FileName 赋值，CommonAVI 属性的值会自动设置为 aviNone。用户不可能同时选择 Windows AVI 动画和一个动画文件，即要么使用 FileName 属性要么使用 CommonAVI 属性。使用 FileName 时，文件必须存在，而且必须是一个没有声音的 AVI 动画。

- **FrameCount 属性**：该属性给出 AVI 动画剪辑中所包含的帧数，这个值是只读的。当 CommonAVI 或 FileName 被赋值时，FrameCount 将自动被更新。

- Repetitions 属性: 当 Active 属性被设置为 True 时, Repetitions 的值代表重复播放动画剪辑的次数; 当该属性被设置为 0 时, 将一直播放动画, 直到 Active 属性被设置为 False 或调用了 Animate.Stop 方法为止。

- StartFrame 属性: 该属性是一个帧号, 动画将从这一帧开始播放。动画的第一帧编号为 1。如果用户程序调用了 play 方法, 就不需要设置这个属性。

- StoptFrame 属性: 该属性也是一个帧号, 动画播放到这一帧就停止。帧数可以由只读属性 FrameCount 中获得。如果用户程序调用了 play 方法, 就不需要设置这个属性。

- Timers 属性: 该属性值为 False 时 (默认值), 动画显示将以分开的线程运行。当该属性值为 True 时, 动画显示将由系统定时器控制。通常可以让 Timers 属性保持其默认值, 但如果需要使动画与其他事件同步, 就应该把 Timers 属性改为 True, 可使用 Animate 组件的 OnStart 和 OnStop 事件来控制同步活动。

- Transpatent 属性: 该属性值为 True 时, 就把父对象的颜色作为动画的背景色显示。在窗口中显示动画时, 通常都使用这一设置。如果 Transpatent 的值为 False 时, 那么动画背景色将取自 AVI 动画。

2. Animate 组件对象的主要方法

- Create 方法: 该方法可用来创建和初始化动画组件的一些属性, 它常用于实时状态下。当在设计阶段放置一个 Animate 组件在窗体上时, 该方法会自动被调用。

- Play 方法: Play 方法按照 AVI 剪辑文件顺序显示图片。该方法能按照 Repetitions 设定的重复次数进行显示。FromFrame 为第一个显示的图片框, ToFrame 为最后一个要显示的图片框。ToFrame 值大于或等于 FromFrame 的值, 且小于 FrameCount 的值。

- Reset 方法: 该方法将清除 StartFrame 和 StopFrame 的设定值, 并使用它们的默认值。Reset 方法关闭和响应动画控制组件的动作, 并激活 OnClose 和 OnOpen 事件。

- Seek 方法: 该方法显示在图片序列中的一个特定的图片。

3. Animate 组件对象的主要事件

- OnClose 事件: 该事件可以处理当 Open 属性变为 False 时的一些事情。当剪辑文件停止播放, 但动画控制组件还没有关闭时, 则可使用 OnStop 事件。

- OnStop 事件: 该事件在当 AVI 剪辑文件显示完成后被激活。在 OnStop 事件中可以处理一些特定的动作, 例如可以进行其他组件的属性的设定等。

- OnStart 事件: 当 Animate 组件开始播放图片时, OnStart 事件发生。它可用于在当 Open 属性由 False 变为 True 时处理一些特定的动作。

- OnOpen 事件: 当 Animate 组件打开时, OnOpen 事件发生。它可用于在当 Open 属性由 False 变为 True 时处理一些特定的动作。

4. Animate 组件的使用

下面我们设计一个程序, 说明 Animate 组件的使用。

[例 9-1] 使用 Animate 组件浏览 AVI 动画剪辑

效果: 设置 8 种标准的 AVI 动画单选按钮组, 选择 Windows 提供的 8 种标准的 AVI 动画中的任意一个单选按钮, 在屏幕显示改动画。执行效果如图 9-1 所示。

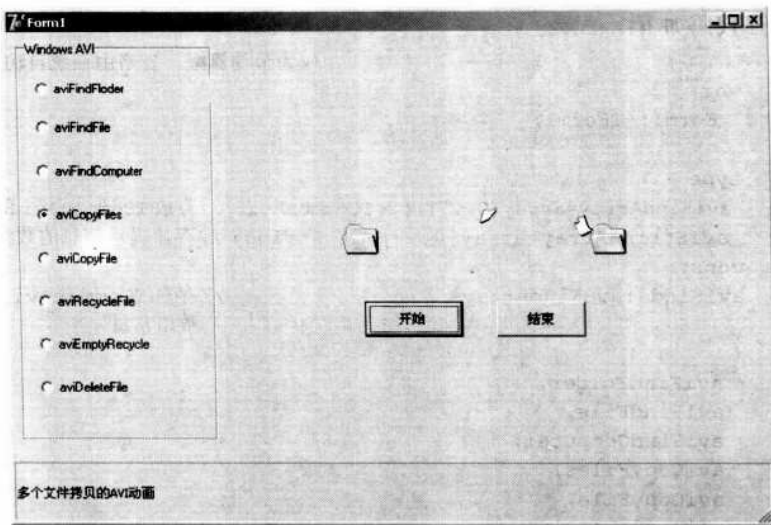


图 9-1 使用 Animate 组件浏览 AVI 动画剪辑

(1) 新建一个应用程序工程。在窗体中添加一个 **Animate1** 组件，用来显示指定的 Windows 标准 AVI 动画剪辑。

(2) 在窗体中添加一个开始播放动画的命令按钮 **Button1** 组件，该组件的【Caption】属性值为“开始”。

(3) 在窗体中添加一个结束播放动画的命令按钮 **Button2** 组件，该组件的【Caption】属性值为“结束”。

(4) 在窗体中添加一个状态栏 **StatusBar1** 组件，显示当前播放内容的状态。

(5) 在窗体中添加一个 **RadioGroup1** 组件，在该组件里加入 8 个 AVI 动画剪辑单选按钮。程序设计界面如图 9-2 所示。

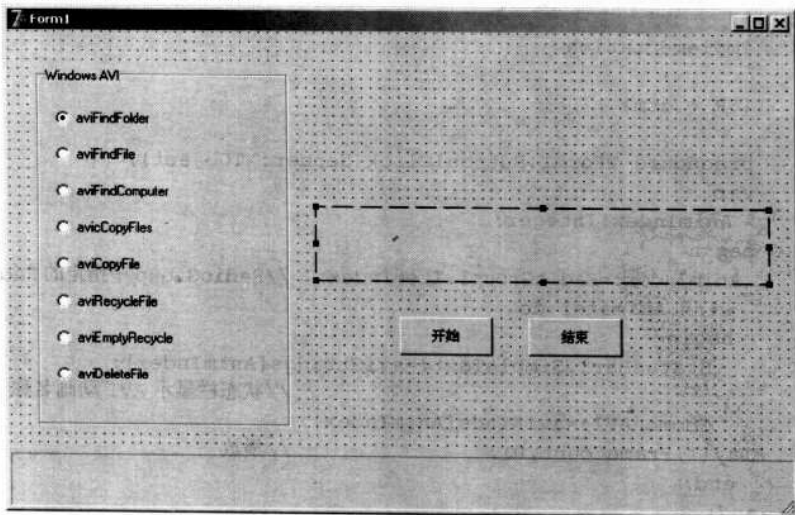


图 9-2 程序设计界面

(6) 程序代码如下:

```

... .. //为节约篇幅,省略由系统自动产生的代码
var
    Form1: TForm1;

type
    aviKindArray=array[0..7] of TCommonAvi; //存放 TCommonAvi 的数组类型
    aviStringArray=array[0..7] of String; //存放提示信息的数组类型
const
    aviKinds:aviKindArray= //存有 TCommonAvi 中常量成员的
                           数组常量
    (
        aviFindFolder,
        aviFindFile,
        aviFindComputer,
        aviCopyFiles,
        aviCopyFile,
        aviRecycleFile,
        aviEmptyRecycle,
        aviDeleteFile
    );
    aviStrings:aviStringArray= //存有提示信息的数组常量
    (
        '查找文件夹的 AVI 动画',
        '查找文件的 AVI 动画',
        '查找计算机的 AVI 动画',
        '多个文件拷贝的 AVI 动画',
        '单个文件拷贝的 AVI 动画',
        '将文件放入回收站的 AVI 动画',
        '清空回收站的 AVI 动画',
        '删除文件的 AVI 动画'
    );
implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
    AnimIndex:Integer;
begin
    AnimIndex:=RadioGroup1.ItemIndex; //RadioGroup 中指定的 Radio 的编号
    with Animatel do
    begin
        StatusBar1.SimpleText:=aviStrings[AnimIndex];
                                                //状态栏显示 AVI 动画名称
        CommonAVI:=aviKinds[AnimIndex];
        Play(1,FrameCount,0); //播放
    end;
end;
procedure TForm1.Button2Click(Sender: TObject);

```


```

begin
    Animatel.Stop;           //停止播放
    StatusBar1.SimpleText:='动画结束!';
end;
end.

```

(7) 运行程序, 点击图中的【aviCopyFiles】按钮, 会出现多个文件拷贝的动画。

9.2.2 MediaPlayer (媒体播放器) 组件

MediaPlayer 组件 , 位于【System】选项卡上, 是一个功能强大的音、视频媒体播放组件。它提供了类似 Windows 中的媒体控制接口 (MCI-Media Control Interface), 只要在 Windows 中 (位于控制面板的多媒体图标内) 完整安装了媒体控制设备驱动程序, 就可利用该组件来播放媒体文件。

1. MediaPlayer 组件的按钮

MediaPlayer 组件为用户提供了一组可视化按钮, 如图 9-3 所示。

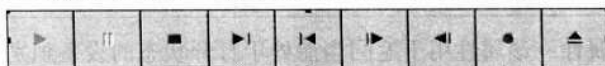


图 9-3 MediaPlayer 组件的可视化按钮

每个按钮代表了一种操作, 每个按钮对应一个特定的值和方法, 当某个按钮 OnClick 事件发生时, 就调用并执行相应的方法。MediaPlayer 组件上九个按钮的具体功能含义, 按照从左到右顺序, 如表 9-2 所示。

表 9-2 MediaPlayer 组件对象按钮功能

按钮名称	按钮值	执行操作	对应方法
Play	btPlay	开始播放	Play
Pause	bt Pause	暂停播放或录制, 若目前正处于暂停状态, 则继续播放或录制	Pause
Stop	bt Stop	停止播放或录制	Stop
Next	bt Next	跳到下一个轨道, 若媒体不使用轨道, 则跳到媒体最后	Next
Prev	bt Prev	跳到前一个轨道, 若媒体不使用轨道, 则跳到媒体的开始位置	Previous
Step	bt Step	前进几帧	Step
Back	bt Back	后退几帧	Back
Record	bt Record	开始录制	Start Recording
Eject	bt Eject	退出	Eject

2. MediaPlayer 组件的主要属性

- **AutoEnable** 属性: 该属性设置为 True 时, MediaPlayer 组件在运行过程中能够自动—控制哪些按钮当前可用, 哪些按钮当前不可用 (不可用的按钮以灰色显示)。判断的依据是 Mode 特性返回的设备状态以及 DeviceType 特性设置的设备类型。

- **AutoOpen** 属性: 该属性设置为 True 时, MediaPlayer 组件自动打开 DeviceType 属性指定的多媒体设备。如果 AutoOpen 属性设置为 False 时, 就必须调用 Open 方法打开设备。

● **AutoRewind** 属性: 该属性用于设置是否具有自动返回功能。如果 **AutoRewind** 属性设置为 **True** 时, 当需要播放或录制时将自动返回到媒体文件的开始处。否则 **AutoRewind** 属性设置为 **False** 时, 只有当用户按下 **Prev** 或在程序中调用了 **Previous** 方法后才能返回到媒体文件的开始处。

● **Capabilities** 属性: 该属性用于返回当前设备的可操作状态。其取值以及含义如表 9-3 所示。

表 9-3 Capabilities 属性的取值以及含义

取 值	含 义
MpCanEject	可以弹出
MpCanPlay	可以播放
MpCanRecord	可以记录
MpCanStep	可以单步返回或快进
MpUseWindows	可以使用窗口来显示输出

● **DeviceID** 属性: 该属性用于返回当前处于打开状态的设备识别号。如果当前没有处于打开状态的设备, 那么这个属性值就为 0。

● **DeviceType** 属性: 该属性用于指定设备类型。该属性的默认值是 **dtAutoSelect**, 表示自动识别设备类型, 如果设置为其他值时, 将指定某种特定的设备类型。该属性取值如表 9-4 所示。

当 **DeviceType** 属性设置为 **dtAutoSelect** 时, 将根据 **FileName** 属性指定的媒体文件的扩展名来判断设备类型, 每种扩展名对应着一种设备类型。

表 9-4 DeviceType 属性的取值以及含义

取值	含义
dtAutoSelect	自动检测设备类型
dtAviVideo	播放 AVI 视频文件
dtCDAudio	CD 唱盘
dtDat	数字音频磁带
dtDigitalVideo	AVI、MPG、MOV 文件
dtMMMove	MM 电影
dtScanner	图像扫描设备
dtSequencer	MIDI 文件
dtVCR	WAV 文件
dtOverlay	覆盖设备
dtVideodisc	影碟设备
dtWaveAudio	WAV 文件
dtOther	其他设备

● **Display** 属性: 该属性用于指定一个窗口作为媒体文件的输出窗口。默认值是 **Nil**, 此时 **MediaPlayer** 组件自己开一个窗口显示输出。

● **DisplayRect** 属性: 如果 **Display** 属性指定一个媒体文件的输出窗口。那么

DisplayRect 属性可用于在输出窗口上设置一个矩形区域作为输出范围。

- EnabledButtons 属性：该属性用于设置哪些按钮当前可用，哪些按钮当前不可用，不可用的按钮以灰色显示。如果 AutoEnable 属性设置为 True 时，这个属性无效。
- EndPos 属性：该属性和 StartPos 属性一起用于设置媒体文件的起始位置和终止位置，MediaPlayer 组件只能部分设置这个区间的内容。
- Error 属性：当媒体设备出错时，该属性返回错误代码。如果该属性返回值为 0，表示设备没有出错。
- ErrorMessage 属性：当媒体设备出错时，该属性返回错误的详细信息。
- FileName 属性：该属性用于指定要播放的媒体文件名。
- Frames 属性：该属性用于设置快进或返回时的步进幅度，即每次移动占整个媒体文件长度的百分比。该属性的默认值是 10，表示每次移动整个媒体文件长度的 10%。
- Length 属性：该属性的返回值是媒体文件的长度，通常以时间表示，格式由 TimeFormat 属性指定。该属性是只读的。
- Mode 属性：该属性返回媒体设备的当前状态，其取值如表 9-5 所示。

表 9-5 Mode 属性的取值以及含义

取 值	含 义
mpNotReady	没有准备好
mpStopped	已停止播放
mpPlaying	正在播放
mpRecording	正在记录
mpSeeking	正在搜索
mpPaused	暂停状态
mpOpen	已打开

- Notify 属性：该属性如果设置为 True 时，设置后的第一个媒体控制方法执行结束时，将触发 OnNotify 事件。否则设置为 False 时，媒体控制方法执行结束时不会触发 OnNotify 事件。
- NotifyValue 属性：该属性返回媒体控制方法的执行情况，其取值如表 9-6 所示。

表 9-6 NotifyValue 属性的取值以及含义

取 值	含 义
nvSuccessful	操作命令成功执行
nvSuperseded	操作命令被其他命令取代
nvAborted	操作命令被用户废止
nvFailure	操作命令失败

下面的代码说明了 Notify 属性和 NotifyValue 属性的使用。当应用程序播放声音文件失败时，显示对话框提示执行出错。

```
begin
    with MediaPlayer1 do
        begin
```

```

Visible:=False;
Notify:=True;
DeviceType:=dtCDAudio;
Open;
Play;
if NotifyValue<>nvSuccessful then
  MessageDlg('播放 CD 音乐失败!', mtError, [mbOK], 0);
end;
end;

```

● **Position** 属性：该属性用于指定 MediaPlayer 组件的当前位置。该属性通常以时间表示，格式由 TimeFormat 特性指定。该属性的默认值是媒体的开始位置；如果媒体支持多轨道，那么默认值是第一个轨道的开始位置。

● **Shareable** 属性：该属性用于确定多媒体设备是否能够共享。如果该属性的值设置为 True，则多个组件和应用程序可以共享指定的设备。如果该属性的值设置为 False，则其他组件和应用程序不能共享指定的媒体设备。

● **Start** 属性：该属性用于指定媒体播放时的起始位置。对于不使用轨道的媒体设备，该属性是指媒体的开始位置，而对于使用轨道的媒体设备，该属性是指媒体第一个轨道的开始位置。该属性通常以时间表示，格式由 TimeFormat 属性指定。该属性在设计时是不可见的，当调用 Open 方法打开媒体设备时，该属性才被正式定义。

● **StartPos** 属性：该属性用于指定当前媒体开始播放或录制时的起始点。StartPos 属性由时间表示，格式由 TimeFormat 属性指定。当该属性设置了有效值后，只对下一次调用 Play 和 StartRecording 方法起作用。在 play 和 StartRecording 方法调用之前，该属性的值不会影响媒体的当前位置。该属性与 EndPos 属性共同决定了媒体播放和录制时的起点位置和终点位置。

● **TimeFormat** 属性：该属性用于指定模式时间的格式。描述时间的格式是由 4 个字节的整数组成，对于不同的时间格式，四个字节有不同的含义，其取值如表 9-7 所示。

表 9-7 TimeFormat 属性的取值以及含义

取 值	含 义
TfMilliSeconds	MilliSeconds
TfHMS	时、分、秒、未用
TfMSF	分、秒、Frames、未用
TfFrames	Frames
TfSMPTE24	24-Frames: 时、分、秒、Frames
TfSMPTE25	25-Frames: 时、分、秒、Frames
TfSMPTE30	30-Frames: 时、分、秒、Frames
Tf SMPTE30Drop	30-Drop-Frames: 时、分、秒、Frames
TfBytes	Bytes
TfSamples	Samples
TfTMSF	时、分、秒、Frames

● **TrackLength** 属性：该属性用于表示指定轨道的长度。该属性以时间表示，格式由 TimeFormat 确定。某个轨道的时间长度，由该属性加 Integer 类型的索引指定，例如第 2 个

轨道的时间长度, 可用 `TrackLength[2]` 表示。

- **TrackPosition** 属性: 该属性用于表示指定轨道的开始位置。该属性以时间表示, 格式由 `TimeFormat` 确定。某个轨道的开始位置由该属性加 `Integer` 类型的索引指定, 例如第 1 个轨道的开始位置, 可用 `TrackPosition [1]` 表示。

- **Tracks** 属性: 该属性用于表示媒体可播放的轨道数目。

- **VisibleButtons** 属性: 该属性用于指定 `MediaPlayer` 组件组合按钮中, 哪些按钮可视, 哪些按钮不可视。

- **Wait** 属性: 该属性决定是否在媒体控制方法执行结束后, 将控制权移交给应用程序。如果 `Wait` 属性设置为 `True` 时, 须等待媒体控制方法执行结束后, 才能将控制权移交给应用程序; 否则, 不用等待媒体控制方法执行结束。

3. MediaPlayer 组件的主要方法

- **AutoButtonSet** 方法: 该方法能够指定 `MediaPlayer` 组件的按钮可用或不可用。当 `AutoEnable` 属性设置为 `True` 时, 该方法可根据媒体设备的状态自动控制 `MediaPlayer` 组件的按钮处于可用或不可用状态。

- **Back** 方法: 该方法使 `MediaPlayer` 组件后退几个 `Frames`。在程序运行中按下 `MediaPlayer` 组件的 `Back` 按钮, 将调用该方法。

- **Click** 方法: 该方法决定当 `OnClick` 事件被触发时, 执行什么操作。在默认状态下, 当 `OnClick` 事件触发时, 该方法不执行任何操作, 仅仅是调用一个事件处理句柄, 连接到 `MediaPlayer` 组件的 `OnClick` 事件处理程序中。该方法可根据用户需要重新加载事件处理程序。

- **Close** 方法: 该方法关闭已打开的多媒体播放设备。当应用程序中止时, 自动调用该方法。

- **Create** 函数: 这是一个动态的构造函数, 该函数可在运行时建立一个 `TMediaPlayer` 对象, 并对其进行初始化。对于 `MediaPlayer` 组件既可以在设计时静态建立, 也可以在运行时通过调用 `Create` 方法动态建立。通常重新加载的构造函数要通过关键字 `Inherited` 来执行一个继承来的构造函数。继承的构造函数主要做以下工作: 首先为 `MediaPlayer` 组件分配一个内存空间, 然后加载位图文件到按钮的 `Glyph` 属性上, 并对 `MediaPlayer` 组件进行初始化处理, 包括设置 `AutoEnable`、`AutoRewind`、`Colored`、`Enabled` 和 `Visible` 属性的值为 `True`; 设置 `AutoOpen` 的属性值为 `False`; 设置 `DeviceType` 的属性值为 `dtAutoSelect`。

- **Destroy** 函数: 这是一个动态的析构函数, 该函数用于撤消 `MediaPlayer` 组件对象。应用程序不能直接调用该函数, 应先调用 `Free` 方法检查 `MediaPlayer` 组件对象是否空闲, 如果空闲就释放其占用的内存空间。

注意: 该函数执行前, 必须确保媒体设备已经关闭。

- **DoNotify** 方法: 该方法决定在 `OnNotify` 事件触发时, 执行什么操作。该过程可以根据用户需要重新加载事件处理程序, 添加在继承的事件处理句柄之后。在默认状态下, 当 `OnNotify` 事件触发时, 该方法不执行任何操作, 仅仅是调用一个事件处理程序, 连接到 `MediaPlayer` 组件的 `OnNotify` 事件处理程序中。

- **Eject 方法**: 该方法用于打开多媒体设备仓门。删除已安装的媒体文件。在程序运行中按下 MediaPlayer 组件的 Eject 按钮, 将调用该方法。
- **Next 方法**: 该方法用于将 MediaPlayer 组件移动到下一个轨道的开始位置。如果调用该方法时, 媒体的当前位置是最后一个轨道, 那么 MediaPlayer 组件将定位于当前位置。如果多媒体设备不使用轨道, 该方法将定位于媒体的最后位置。在程序运行中按下 MediaPlayer 组件的 Next 按钮, 将调用该方法。
- **Open 方法**: 该方法用于打开多媒体设备。在调用该方法打开多媒体设备之前, 必须在 DeviceType 属性中指定设备类型。
- **Pause 方法**: 该方法用于暂停多媒体设备的播放或录制。如果在设备暂停状态调用该方法, 则设备将调用 resume 方法恢复播放或录制。在程序运行中按下 MediaPlayer 组件的 Pause 按钮, 将调用该方法。
- **PauseOnly 方法**: 该方法用于暂停多媒体设备的播放或录制。如果在设备暂停状态调用该方法, 设备将继续保持暂停状态。
- **Play 方法**: 该方法用于在打开的多媒体设备中播放已安装的媒体。如果设置了 StartPos 属性, 那么该方法从 StartPos 指定的位置开始播放; 否则从 Position 属性指定的当前位置开始播放。同样, 如果设置了 EndPos 属性, 那么该方法播放到 EndPos 指定的位置结束; 否则一直播放到媒体结束。如果设置 AutoRewind 属性为 true, 那么调用该过程时, 无论当前位置在何处, 都将从头开始播放。在程序运行中按下 MediaPlayer 组件的 Play 按钮, 将调用该方法。
- **PostClick 方法**: 该方法决定当 OnPostClick 事件被触发时, 执行什么操作。该方法是 OnPostClick 事件的一个可执行方法。该方法可以根据用户需要重新加载事件处理程序。在默认状态下, 当 OnPostClick 事件被触发时, 该方法不执行任何操作, 仅仅是调用一个事件处理句柄, 连接到 MediaPlayer 组件的 OnPostClick 事件处理程序中。
- **Previous 方法**: 该方法使 MediaPlayer 组件退回。如果当前位置是一个轨道的开始位置, 那么调用该方法可使媒体退回到上一个轨道的开始位置; 如果当前位置不是一个轨道的开始位置, 那么退回到当前轨道的开始位置; 如果媒体设备不使用轨道, 那么退回到 Start 属性指定的媒体开始位置。在程序运行中按下 MediaPlayer 组件的 Previous 按钮, 将调用该方法。
- **Resume 方法**: 该方法是使目前处于暂停状态的媒体设备恢复播放或录制。在程序运行过程中, 当媒体设备处于暂停状态时, 按下 MediaPlayer 组件的 Resume 按钮, 将调用该方法。
- **Rewind 方法**: 该方法用于移动当前位置到 Start 属性指定的媒体开始位置。
- **Save 方法**: 该方法用于把当前已加载的媒体存储到 FileName 属性指定的文件中。
- **StartRecording 方法**: 该方法从当前位置或 StartPos 属性指定的位置开始录制。在程序运行中按下 MediaPlayer 组件的 Record 按钮, 将调用该方法。
- **Step 方法**: 该方法在媒体中向前 (播放方向) 移动数个 Frames。在程序运行中按下 MediaPlayer 组件的 Step 按钮, 将调用该方法。
- **Stop 方法**: 该方法中止媒体播放或录制。在程序运行中按下 MediaPlayer 组件的 Stop 按钮, 将调用该方法。

4. MediaPlayer 组件的主要事件

● **OnClick 事件**: 当用户单击了 MediaPlayer 组件上的任一按钮, 或者当 MediaPlayer 组件按钮获得焦点时按下空格键, 都将会触发该事件。

● **OnNotify 事件**: 如果 Notify 属性设置为 True, 当一个媒体控制方法 (如 Back、Close、Eject、Next、Open、Pause、PauseOnly、Play、Previous、Resume、StartRecording、Step 或 Stop) 执行结束时, 触发该事件。一个 OnNotify 事件触发后, 必须重新设置 Notify 属性, 以便触发下一个 OnNotify 事件。

● **OnPostClick 事件**: 当 OnClick 事件处理程序被调用以后, 触发该事件。如果 Wait 属性设置为 True, 当 MediaPlayer 组件控制按钮被单击时, 只有在 OnClick 事件处理程序执行结束后, 才能调用该事件处理程序; 如果 Wait 设置为 False, 应用程序可以在 OnClick 事件处理程序执行结束前, 取得控制权。

● **OnEnter 事件**: 这个是从 WinControl 继承来的事件, 当一个组件接收到一个输入焦点时, 触发该事件。

● **OnExit 事件**: 这个也是从 WinControl 继承来的事件, 当输入焦点从一个组件移动到另一个组件时, 触发该事件。

5. MediaPlayer 组件的使用

【例 9-2】利用 MediaPlayer 组件, 实现音/视频媒体的播放。

效果: 设计的这个媒体播放器可以播放 WAV 文件和 MIDI 格式的音频文件以及 AVI 格式的视频文件。执行效果如图 9-4 所示。



图 9-4 程序执行效果

(1) 创建一个应用程序工程, 在空白窗体中放置一个 MediaPlayer1 组件用于播放音频和视频。

(2) 由于我们不需要 MediaPlayer 组件提供的所有按钮功能, 在窗体中通过 5 个按钮实现播放媒体的常用功能, 所以 MediaPlayer1 组件的【Visible】属性设置为“False” (不可见)。

(3) 窗体中 5 个命令按钮的属性设置如下表 9-8 所示。

表 9-8 5 个命令按钮属性设置

组件	Name	Caption
Button1	OpenButton	打开
Button2	PlayButton	播放
Button3	PauseButton	暂停
Button4	RewindButton	重播
Button5	OpenButton	离开

(4) 在窗体中还有一个文件打开对话框用于选择播放的媒体文件。在 `OpenDialog1` 中设置过滤属性, 如图 9-5 所示。

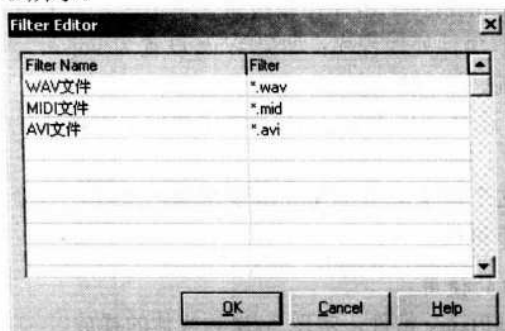


图 9-5 Filter Editor 的设置

(5) 在窗体中添加一个 `Panel` 面板用于作为播放视频的屏幕。程序设计界面如图 9-6 所示。

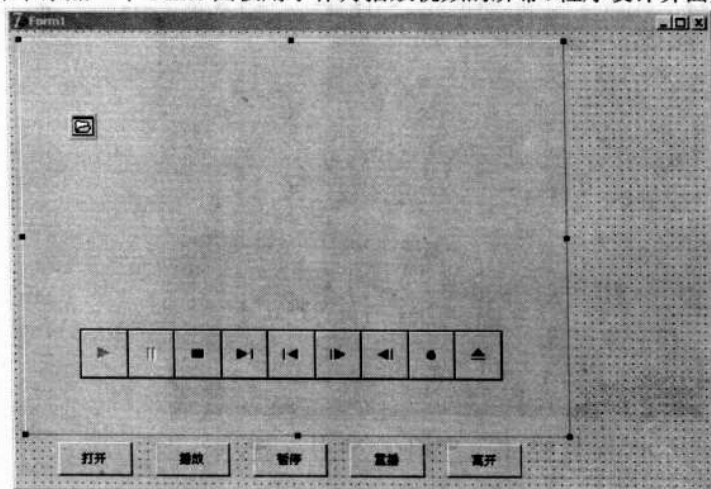


图 9-6 媒体播放器设计界面

【打开】按钮的功能包括: 在文件打开对话框中选择需要播放的媒体文件; 将媒体播放组件设置为打开状态, 并将播放按钮、暂停按钮和重播按钮设置为可用。

(6) 【打开】按钮的 `OnClick` 事件代码如下:

```
procedure TForm1.OpenButtonClick(Sender: TObject);
begin
```

```

if OpenFileDialog1.Execute then
begin
    MediaPlayer1.Close;
    MediaPlayer1.FileName:=OpenDialog1.FileName;
    MediaPlayer1.Open; // 将 MediaPlayer 组件对象设置为打开状态
    if ExtractFileExt(OpenDialog1.FileName)='*.avi' then
    begin
        MediaPlayer1.Display:=Panel1; // 将 Panel1 作为播放视频的屏幕

        MediaPlayer1.DisplayRect:=Rect(0,0,Panel1.Width,Panel1.Height);
        // 让视频填满整个屏幕播放
    end;
    PlayButton.Enabled:=True; // 播放按钮可用
    PauseButton.Enabled:=False; // 暂停按钮不可用
    RewindButton.Enabled:=False; // 重播按钮不可用
end;
end;

```

【播放】按钮功能包括首先判断当前媒体播放设备的播放状态，如果为暂停状态，就调用 resume 方法继续播放，否则调用 Play 方法重新开始播放。最后设置暂停按钮和重播按钮为可用。

(7) 【播放】按钮的 OnClick 事件代码如下：

```

procedure TForm1.PlayButtonClick(Sender: TObject);
begin
    if MediaPlayer1.Mode=mpPaused then
    begin
        MediaPlayer1.Resume;
    end
    else
    begin
        MediaPlayer1.Play;
        RewindButton.Enabled:=True;
        PauseButton.Enabled:=True;
    end;
end;

```

【暂停】按钮功能用于将 MediaPlayer 组件对象设置为暂停状态，同时将暂停按钮和重播按钮设置为不可用。

(8) 【暂停】按钮的 OnClick 事件代码如下：

```

procedure TForm1.PauseButtonClick(Sender: TObject);
begin
    MediaPlayer1.Pause;
    PauseButton.Enabled:=False;
    RewindButton.Enabled:=False;
end;

```

【重播】按钮功能用于将 MediaPlayer 组件对象设置为重播状态，并调用 Play 方法播放。

(9) 【重播】按钮的 OnClick 事件代码如下：

```

procedure TForm1.RewindButtonClick(Sender: TObject);
begin
    MediaPlayer1.Rewind;
end;

```



```
MediaPlayer1.Play;  
end;
```

【离开】按钮功能用于将 MediaPlayer 组件对象设置为关闭状态，并关闭应用程序窗体。

(10) 【离开】按钮的 OnClick 事件代码如下：

```
procedure TForm1.CloseButtonClick(Sender: TObject);  
begin  
  MediaPlayer1.Close;  
  Close;  
end;
```

(11) 由于 MediaPlayer 组件对象的【DeviceType】属性的默认设置为“dtAutoSelect”（设备自动选择），应用程序将根据用户选择播放的媒体文件的类型自动选择播放设备。

(12) 运行程序。首先点击【打开】按钮，打开 AVI 格式的文件，然后再点击【播放】按钮，在窗体中显示该 AVI 视频文件。

9.3 多媒体编程

通过对 MediaPlayer 组件的使用，领会多媒体编程的技巧。本节将介绍如何实现多媒体设计。

9.3.1 用 MediaPlayer 播放 WAVE 声音文件

播放 wave 声音文件是最基本、最普通的多媒体操作之一。一段完整的播放 wave 声音文件的程序代码如下：

```
MediaPlayer1.Wait:=True;  
MediaPlayer1.FileName:='声音.wav';  
MediaPlayer1.Open;  
MediaPlayer1.Play;
```

在上面这段代码中，首先对 MediaPlayer1 的 Wait 属性进行了设置。Wait 属性决定是否当一个媒体控制方法完成后才将控制返回应用程序。当其值设置为 True 时，将等到此设置后的一个媒体控制方法完成后再将控制权交回给程序，如果设置为 False 时，则程序不等后面一个媒体控制方法执行完就将继续执行下面的程序代码。

通过下面的程序代码可以说明 Wait 属性的重要性。

```
MediaPlayer1.FileName:='声音 1.wav';  
MediaPlayer1.Open;  
MediaPlayer1.Wait:=True;  
MediaPlayer1.Play;  
MediaPlayer1.FileName:='声音 2.wav';  
MediaPlayer1.Wait:=True;  
MediaPlayer1.Play;
```

在上面这段程序代码中，由于有 MediaPlayer1.Wait:=True; 这句代码，程序将在播放完声音 1.wav 声音文件后再播放声音 2.wav 声音文件，直到两个声音文件都播放完后，程序才可以继续执行后面的代码。如果这里没有设置 Wait 属性为 True，则程序在刚开始播放声音 1.wav 声音文件后立即中止播放并开始播放声音 2.wav 声音文件，如果后面的

代码中又对 MediaPlayer1 的媒体控制方法进行调用,那么声音 2.wav 声音文件的播放也将会被中止。

要播放 wave 文件的一部分,可以在播放前设置 StartPos 和 EndPos。下面程序代码打开一个 wave 文件,播放其中从第 1 秒到第 3 秒的一段声音。

```
MediaPlayer1.FileName:='声音.wav';
MediaPlayer1.Open;
MediaPlayer1.StartPos:=1000
MediaPlayer1.EndPos:=1000
MediaPlayer1.Play;
```

这里,StartPos 和 EndPos 的值按毫秒为单位设置,这是声音设备 (wave audio devices) 的默认值。

注意: 如果设置 StartPos 或 EndPos 为非法值,则 WAVE 文件不能播放。非法值包括 StartPos 大于 EndPos 或 EndPos 大于媒体长度。

设置声音设备的输出音量相对来说比较简单,需要通过 Windows API 函数来实现。waveOutGetVolume 和 waveOutSetVolume 函数可以分别读取音量和设置音量。在这两个函数中音量的值被存储为整数。其中 2 进制高位字规定右声道的音量设置,低位字代表左声道的音量设置。如果设备没有区分左右声道的能力,则低位字用于设置音量,而高位字被忽略。0 值表示没有音量,十六进制数 \$FFFF 表示全音量。下面的代码设置左、右声道都为 50% 音量:

```
waveOutSetVolume(0,$80008000);
```

而下例设置音量为全音量:

```
waveOutSetVolume(0,$FFFFFFFF);
```

在上面两个例子中第一个参数为声音播放设备号,一般情况下 wave 设备号为 0。如果需要进一步对设备号进行设置,可以参考 waveOutSetVolume 函数的语法。

9.3.2 用 MediaPlayer 播放 MIDI 声音文件

播放 MIDI 声音文件十分简单。所需要做的工作就是:设置 MediaPlayer 的【FileName】属性为 MIDI 文件名,然后调用 Play 方法。MIDI 文件的后缀为 .mid 或 .rmi。

多数声卡不能同时播放两个 wave 文件,同样也不能同时播放两个 MIDI 文件。然而,却能同时播放一个 wave 文件和一个 MIDI 文件。要做到这一点,可以采用两个 MediaPlayer 组件。

MIDI 文件常常用作背景音乐,这种情况需要对 MIDI 文件反复播放。要做到这一点,可以利用 MediaPlayer 的 OnNotify 事件。首先,需要设置 MediaPlayer1 的 Notify 属性为 true,以确保产生一个 OnNotify 事件。

```
MediaPlayer1.Notify:=True;
```

然后需要在这个 OnNotify 事件中加入代码以使该 MIDI 文件成功播放完后再次进行播放,OnNotify 的事件代码如下面的例子所示。

下面的代码实现 MIDI 音乐的连续播放。

```
Procedure TForm1.MediaPlayer1Notify(Sender:TObject);
begin
  with MediaPlayer1 do
```

```

    if NotifyValue=nvSuccessful then begin
        Position:=0;
        Play;
        Notify:=True;
    end;
end;

```

在上面的例子里，首先检查 NotifyValue 属性的值是否为 nvSuccessful。如果是，表明该 MIDI 声音文件已经成功播放完，可以设置文件返回起始位置，并调用 Play 方法再次播放该文件，从而实现不断循环往复地播放该背景音乐文件。

注意：如果 MediaPlayer 媒体播放组件的 AutoRewind 属性值被设为 True，文件在播放完后会自动返回起始位置而不再需要 Position:=0。

9.3.3 MediaPlayer 的 OnClick 事件编程

前面已经介绍 MediaPlayer 有 9 个控制按钮。当用户操作这些控制按钮时会有相应的事件发生。通过对这些事件句柄编程，可以获得对 MCI 多媒体能力的全面控制。这里着重介绍 MediaPlayer 的 OnClick 事件的使用例子。当用户鼠标在一个有效控制按钮上单击时产生 OnClick 事件。当 MediaPlayer 正被激活、用户按下空格键时也产生 OnClick 事件。Delphi 7 中关于 OnClick 事件属性的类型定义如下：

```

type
    TMPBtnType=(btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject);
    EMPNotify=procedure (Sender:TObject; Button:TMPBtnType; var
        DoDefault:Boolean) of object;
    Property OnClick: EMPNotify;

```

OnClick 属性（事件也是属性）的类型为 EMPNotify。EMPNotify 类型是一个方法指针，每当一个 OnClick 事件在 MediaPlayer 组件上发生时被调用。过程的 button 参数应为下列值之一：btPlay, btPause, btStop, btNext, btPrev, btStep, btBack, btRecord, btEject。

DoDefault 的默认值为 true。如果 DoDefault 为 True，则 MediaPlayer 组件调用的方法对应于控制按钮。例如，如果用户单击 Play 按钮（值为 btPlay），则 Play 方法被调用。如果 DoDefault 为 False，则程序员必须编写 OnClick 事件句柄代码，以使得在按下某个控制按钮时执行特定动作。

下面介绍一个 OnClick 事件编程的例子。

[例 9-3] MediaPlayer 的 OnClick 事件编程

效果：在窗体上放置一个 Label1 和一个 MediaPlayer，当用户单击 MediaPlayer 的一个控制按钮时，标签 Label1 中指示哪个按钮被单击。执行效果如图 9-7、9-8、9-9 所示。

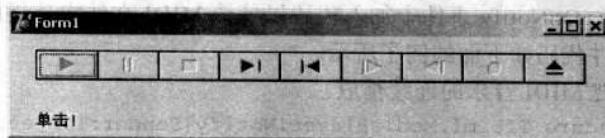


图 9-7 程序运行，标签显示“单击！”

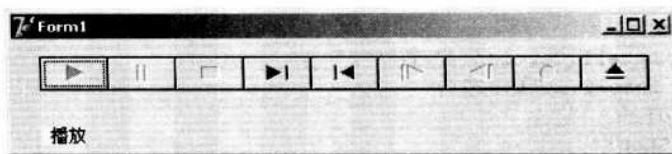


图 9-8 单击 Play 按钮, 标签显示“播放”

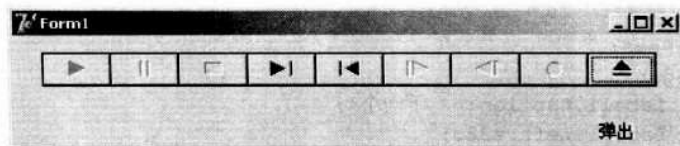


图 9-9 单击 Eject 按钮, 标签显示“弹出”

程序代码如下:

```

... .. //为节约篇幅, 省略由系统自动产生的代码
var
    Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    MediaPlayer1.DeviceType:=dtCDAudio;
    MediaPlayer1.Open;
    MediaPlayer1.Left:=20;
    MediaPlayer1.Top:=12;
    Label1.Top:=64;
    Label1.Left:=20; //标签的位置设置在第一个按钮下面
    Label1.Color:=clYellow;
    Label1.Font.Name:='宋体';
    Label1.Font.Size:=10;
    Label1.Caption:='单击!';
end;

procedure TForm1.MediaPlayer1Click(Sender: TObject; Button:
TMFBtnType;
    var DoDefault: Boolean);
begin
    case Button of
    btPlay:
    begin
        Label1.Caption:=' 播放';
        Label1.Left:=20;
    end;
    btPause:

```

```
begin
    Label1.Caption:=' 暂停';
    Label1.Left:=224;
end;
btStop:
begin
    Label1.Caption:=' 停止';
    Label1.Left:=252;
end;
btNext:
begin
    Label1.Caption:=' 下一个';
    Label1.Left:=280;
end;
btPrev:
begin
    Label1.Caption:=' 前一个';
    Label1.Left:=308;
end;
btStep:
begin
    Label1.Caption:=' 步进';
    Label1.Left:=336;
end;
btBack:
begin
    Label1.Caption:=' 返回';
    Label1.Left:=364;
end;
btRecord:
begin
    Label1.Caption:=' 录制';
    Label1.Left:=392;
end;
btEject:
begin
    Label1.Caption:='弹出';
    Label1.Left:=420;
end;
end;
end;
end.
```

程序运行时的结果, 可以看到, 标签显示的位置正在被操作的控制按钮下面, 这是因为程序中用 `case` 语句根据控制按钮的值来设置 `Label1` 的内容和左边沿的位置。

9.3.4 控制光驱

要控制光驱的弹出和关闭状态, 可以使用 `MMSystem` 单元的 `mciSendString` 函数来实现。该函数的原型如下:

```
MCIERROR mciSendString(
```

```

LPCTSTR lpszCommand,
LPTSTR lpszReturnString,
UNIT cchReturn,
HANDLE hwndCallback
);

```

下面以一个例子说明实现方法。

【例 9-4】实现光驱控制

效果：点击图中的弹出按钮，会弹出光驱，实现对光驱控制。执行效果如图 9-10 所示。

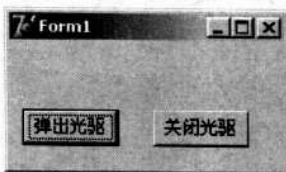


图 9-10 程序的执行效果

(1) 新建一个应用程序工程，向其中添加两个 Button 组件。

(2) 两个 Button 组件的 OnClick 事件程序代码如下：

```

procedure TForm1.Button1Click(Sender:TObject);
begin
  MMSystem.mciSendString('Set cdaudio door open wait', nil, 0, handle);
end;
procedure TForm1.Button2Click(Sender:TObject);
begin
  MMSystem.mciSendString('Set cdaudio door Closed wait', nil, 0,
handle);
end;

```

(3) 运行程序。单击第一个按钮，将会弹出光驱。单击第二个按钮，将会关闭光驱。

9.3.5 显示声音控制属性窗口

要显示声音控制属性窗口，可以使用 WinExec 函数，该函数的原型如下：

```

UNIT WinExec(
  LPCSTR lpCmdLine,
  UINT uCmdShow
);

```

下面举例说明如何显示声音控制属性。

【例 9-5】实现声音控制属性窗口显示

效果：通过使用 WinExec 函数，实现声音控制属性窗口显示。执行效果如图 9-11 所示。

(1) 新建一个应用程序工程。

(2) 在窗体的 OnCreate 事件中添加如下代码：

```

procedure TForm1.FormCreate(Sender:TObject)
begin
  winexec('rundll32.exe shell32.dll,Control_RunDLL access.cpl,,2',
  SW_SHOWNORMAL);
end.

```

(3) 运行程序。将会出现声音设置窗口。

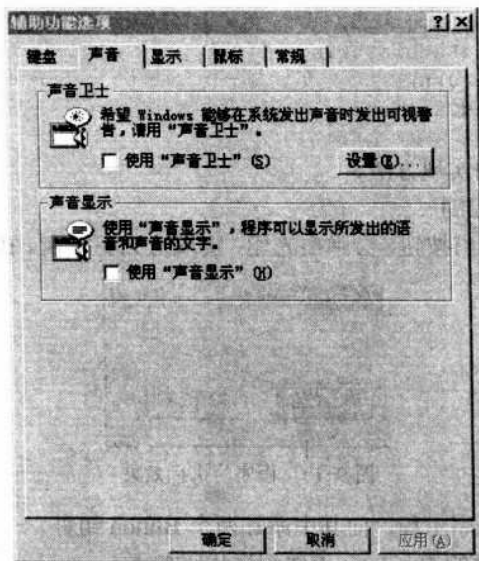


图 9-11 声音设置窗口

9.4 小结

Delphi 7 作为一种快速开发工具, 有着其他编程工具所没有的方便和快捷。当今的计算机是一个融合了声音、图像、视频等多种媒体的设备, 多媒体的应用也越来越广泛。Delphi 7 为了使用户快速地开发多媒体应用程序, 对 Windows 的底层多媒体操作进行了封装。它提供的 MediaPlayer 类型组件, 支持 CD 唱片、AVI 影片、MIDI 音乐序列文件播放以及 VCR 文件、WAVE 音效文件录放、MP3 文件播放等功能, 使用户能轻而易举地开发多媒体应用程序。

本章详细介绍了 Animate 和 MediaPlayer 两个多媒体组件的属性、方法和事件, 而且通过几个实例程序说明该组件的用法以及和其他组件配合使用所能达到的多媒体效果。通过阅读这些实例程序, 可以加深读者对 Animate 和 MediaPlayer 两个多媒体组件更进一步的理 解, 从而随心所欲地开发 Windows 环境下的各种多媒体应用程序。

第 10 章 数据库基础

在开发数据库应用程序之前，对开发数据库的基本概念应当了解，对数据库的结构、开发数据库应用程序的步骤、开发体系以及方法都应当有清晰的认识。本章在简单介绍了数据库的基本知识的基础上详细介绍了 Delphi 7 的数据库应用程序、Delphi 7 数据库工具的应用以及如何简单数据库开发。

10.1 数据库简介

在当今的信息时代，我们每天都要与各种各样的信息打交道。在计算机中实现对信息的存储、处理、查询等操作都离不开数据库技术的支持。

10.1.1 数据库的基本概念

1. 数据库

数据库简单的定义是：由蕴含着一定的意义的数据，一些按照一定的规律组织起来所组成的数据集合。在数据库中除了用一些作为外部信息的数据之外，还有一些内部信息数据。这些数据定义了数据库的用户及其相应的权限，数据库表单的定义等等，通常把存放这些数据的地方叫做数据字典。数据字典是由数据库系统自行创建并自动维护的，它实际上也是数据库的一组表和视图，与其他的表单和视图并没有物理结构上的区别，唯一不同的是它的内容。

2. 关系数据库

关系数据库的基本数据结构就是二维表，每一张二维表对应着一种联系。表的每一行称为记录（也叫元组）；表的每一列称为属性，我们也可以叫做字段；域就是属性的取值范围。

主码是对于这张表的惟一标识—即一个列或几个列的组合，主码最显著的特点就是在任何给定的时间，没有两个主码包含相同的值，这个称作主码的惟一性原则。同时主码中每一个属性都不能被去掉，而同时仍能够保持主码的惟一性，这个我们称作主码的最小性原则。

关系型数据库系统具有很多优点：

- 关系数据库有深厚的理论基础，它是基于关系代数和关系理论的模型。
- 以二维表的形式表示数据。
- 表与表之间的联系不是硬编码的。
- 不需要用户了解它在计算机中的物理存储形式。

- 用系统表来提供其本身的内容和结构。
- 可以通过 SQL 语言来操纵。SQL 语言是专门用于操作这种模型的语言。
- 支持空值的概念。

正是由于关系数据库模型的诸多优点,使得它成为当今数据库的主流模型。

3. 数据库管理系统 (DBMS)

数据库管理系统是一个用来管理数据库的软件包,是数据库能够正常工作的核心。它对于数据库就如同操作系统对于硬盘,对数据库的所有操作,包括创建各种数据库的数据类型、表单、视图、存储过程,以及其他的数据库应用程序对于数据库中数据的读取和修改,都是经由数据库管理系统完成的。而我们前面所说的数据库应用程序所接触的只是数据库的接口,这个接口也就是数据库管理系统的接口函数,当数据库应用程序把对于数据库数据的操作指令通过数据库管理系统的接口函数发送给数据库管理系统后的一切工作都只是数据库管理系统的了,数据库应用程序所要做的就只是等候数据库管理系统把它所需要的数据给它,然后进行加工处理。

10.1.2 数据库设计过程

1. 数据库的建立

创建一个数据库的过程有以下几个步骤:

- (1) 确定数据库的使用范围。
- (2) 确定支持数据库所需要的字段。
- (3) 将字段划分成一些合理的数据表格。
- (4) 确定数据表格之间的关联。

创建一个工程时,首先应当全面地分析工程的特点,根据工程的需要确定要建立的数据库,应当使数据库的内容既能达到工程的要求,同时内容上尽可能地清晰简练。在确定数据库的需求后,要将这些需求划分成合理的数据表格。所谓合理的数据表格,通常要满足以下几点:

- 数据表格中的字段所描述的内容有一定的联系。
- 数据表格中至少要有一个字段的记录是不重复的。
- 一个数据表格与数据库其他的数据表格中至少一个能够关联,
- 一个数据表格与数据库其他的同一数据表格不要有多对多的关联。

2. 数据表格的结构

在划分了合理的数据表格之后,就可以建立数据表格的结构。在为字段命名时,应使字段名能够反映字段的内容。字段的数据类型及数据宽度的选择要合理,既要满足使用要求,又要少占用内存。在数据表格结构中需要一个关键字段,数据表格中的数据就是按主关键字段的顺序存放的,而且利用主关键字段能够高效地与其他数据表格建立关联。索引也是数据表格常用的,在数据库中,利用索引可以加快访问速度。在表格中每一行称为记录。

10.1.3 数据库应用程序的开发步骤

数据库应用程序开发的目的是建立一个满足用户长期需求的产品,在开发的初期要分

析用户的需求，程序开发的几个步骤如下：

- 初步设计
- 功能实现
- 运行和维护程序

1. 初步设计

设计阶段要根据用户的需求，定义数据库和应用程序的功能，确定用户的需求功能哪些在设计阶段实现，哪些在程序中实现。

2. 功能实现

将客户需求功能分成几个合理的功能块，分别进行程序设计、调试。常见的划分方法上分成四个功能块：

- 信息处理
- 数据库管理
- 系统维护
- 辅助功能

信息处理是建立数据库应用程序的目的。设计数据库应用程序的目的是为客户提供所需要的信息服务，辅助管理工作，提高工作效率和水平。信息处理最基本的功能包括各类信息查询，统计报表等功能，对于特定的应用程序还可以有特定的功能。数据库管理的主要功能是负责数据库的更新、修改等。一个特定的数据库管理操作要由它的用户的权限决定，这个权限要由有权的用户指定。系统维护的功能是保证数据库应用程序运行的可靠性和安全性，一般包括用户管理，口令设置，各类系统变量和数据字典的维护等。

3. 运行和维护程序

一个应用系统性能的优劣要由用户的使用来做出判断。用户在使用应用程序的过程中会对应用程序提出一些建议和要求，根据用户的建议和要求对数据库应用程序进行适当的修改和完善，从而提高程序的性能。

10.2 SQL 结构化查询语言

SQL 语言作为关系数据库管理系统中的一种通用的结构查询语言，已经被众多的数据库管理系统所采用，如 ORACLE、Sybase、Informix 等数据库管理系统，它们都支持 SQL 语言。Delphi 7 与使用 SQL 语言的数据库管理系统兼容，在使用 Delphi 7 开发数据库应用程序时，我们可以使用 SQL 语言编程，支持 SQL 编程是 Delphi 7 的一个重要特征，这也是体现 Delphi 7 作为一个强大的数据库应用开发工具的一个重要标志。

10.2.1 SQL 语言的发展

1970 年，美国 IBM 研究中心的 E.F.Codd 连续发表了多篇论文，提出了关系数据库模型。1972 年 IBM 开始研制实验型的关系数据库管理系统，当时配置的查询语言称为 SQUARE (Specifying Queries As Relation Expression) 语言。在这种语言中使用了较多的数学符号，并采用了英语单词表示结构式的语法规则，看起来很像英语句子，用户比较欢迎

这种形式的语言。后来 SQUARE 简称为 SQL 即“结构化查询语言”。1986 年 10 月, 美国 ANSI 采用 SQL 作为关系数据库管理系统的标准语言 (ANSI X3.15—1986), 后来被国际标准化组织 (ISO) 采纳为国际标准。目前 SQL 语言被广泛地使用, 这说明它具有强大的生命力, 它使所有数据库用户包括程序员、DBA 管理员和终端用户都受益非浅。

SQL 语言具有如下优点:

- SQL 语言是所有关系数据库的公共语言。
- SQL 是非过程化查询语言。

过程化语言是指像 PASCAL, BASIC, C 这样的语言, 程序员在使用这些语言的时候, 需要把要执行的任务编写成一系列的模块, 每个模块完成任务的一个部分。SQL 语言则不同, 它是非过程化的, 也就是说, 不需说明事情怎么做, 只要描述事件是什么。例如下面的 SQL 语言代码:

```
SELECT Age, Sex, Name
FROM IndividualInformation
WHERE Name="DengKe"
```

你只要在 SELECT 后写明你需要什么, 然后在 FROM 后写从哪里找数据, 以及在 WHERE 后写明你要找的数据符合什么条件就可以了, 剩下的工作由 SQL 完成。

- SQL 使用查询优化器, 可以为查询配置一种最快速存储的方式。
- SQL 提供了简单而丰富的命令。

它的命令包括: 查询命令、数据更新命令、表更新命令、数据和数据对象的存储命令、保持完整性命令等。

有两种方式使用 SQL 语言: 一种是在终端交互方式下使用, 称为交互式 SQL 语言; 另一种是嵌入在高级语言编写的程序中使用, 称为嵌入式 SQL 语言, 这些高级语言称为宿主语言, 可以是 C、ADA、PASCAL、COBOL 或 P/L 等。

10.2.2 SQL 的基本查询功能

SELECT 语句是使用最多的 SQL 语句, 它完成的是数据库的查询功能, SELECT 语句的结构看起来很直观。SQL SELECT 语句, 从数据表中选择出符合条件的记录。例如

```
SELECT Name, Age FROM Employees WHERE Age>20;
```

Distinct 语句的作用是对某个表所选择的字段数据, 忽略重复的情况, 也就是说, 针对某个字段查询出来的记录结果是惟一的。例如,

```
Select Distinct(ID) FROM Employees where Age>20;
```

1. TOP 和 ORDER BY 语句

TOP 语句实现从第一条或最后一条开始 (利用 ORDER BY 条件子句), 返回特定记录的数据的功能。

例如, 需要查询在 2004 年毕业, 班级前 25 名的学生姓名数据时, 可以输入这样的语句:

```
SELECT TOP 25 学生姓名 FROM 学生表 WHERE 毕业年份=2004 ORDER BY 毕业成绩平均分 DESC ;
```

如果没有加上 ORDER BY 条件的话, 所得到的数据, 将会是随机的数据。此外, 在 TOP 语句之后, 除了可以加上数字以外, 还可以利用保留字 PERCENT 来查询。排序参数:

ASC 递增顺序排列, 默认值。DESC 递减顺序排列。例如下面代码实现在职员表中查询职员, 依据出生的先后次序排列输出:

```
SELECT LastName, FirstName FROM Employees ORDER BY LastName ASC;
```

2. IN 条件子句

指定要查询哪一个外部数据库的表。例如, Microsoft Jet database

```
SELECT 顾客编号 FROM 顾客表 IN CUSTOMER.MDB WHERE 顾客编号 LIKE "A*";
```

其中 CUSTOMER.MDBO 为 Microsoft Jet database 的数据库名称, 其中包含了顾客表。

3. HAVING 条件子句

指定一特定的分组记录, 并满足 HAVING 所指定的条件或状态, 但条件是针对分组的条件设置。例如,

```
SELECT 分类编号, Sum(库存数量) FROM 产品表 GROUP BY 分类编号 HAVING Sum(库存数量)>100 AND 产品名称 LIKE "*纸";
```

4. GROUP BY 条件子句

依据指定的字段, 将具有相同数值的记录合并成一条。例如,

```
SELECT 姓名, Count(姓名) AS 职员姓名 FROM 职员表 WHERE 部门名称='业务部' GROUP BY 姓名;
```

5. BETWEEN...AND 运算符

决定某一数值是否介于特定的范围之内。如果要从职员表查询出所有年龄介于 25~30 岁的员工, 可以利用下面的程序代码:

```
SELECT 姓名, 年龄 BETWEEN 25 AND 30 FROM 职员表;
```

6. LIKE 操作数

用来将一字符串与另一特定字符串样式比较, 并将符合该字符串样式的记录过滤出来。

例如, 要查询出所有以“李”为首的姓氏, 可以利用下面的语句:

```
Like "李*";
```

10.2.3 SQL 的其他应用

1. SQL 数字函数

(1) AVG: 算数平均数。例如, 如果要计算职员身高超过 165 厘米的职员平均身高, 可以利用下面的 SQL 语句完成。

```
SELECT Avg AS 平均身高 FROM 职员表 WHERE 身高>165;
```

(2) COUNT: 计算记录条数。如果要统计出业务部门的职员人数, 并查询出职员的姓名, 可以利用下面的程序代码:

```
SELECT Count(姓名) AS 职员姓名 FROM 职员表 WHERE 部门名称='业务部';
```

(3) FIRST 与 LAST: 返回某字段的第一条数据与最后一条数据。

(4) MAX 与 MIN: 返回某字段的最大值与最小值。

(5) SUM: 返回某特定字段或是运算的总和数值。

2. 多层 SQL 查询

顾名思义, 多层的 SQL 查询含义在于: 在一个 SQL 语句中可以包含另一个 SQL 查询语句, 形成内部嵌套的查询类型。

SELECT 语句构成的多层 SQL 查询, 必须用()将该语句括起来。

例如, 我们先从订单表当中, 查询出所有单位, 再将产品表中的单位与订单表的一一对比, 查询出所有高于订单表的单位价格的记录。

```
SELECT * FROM 产品表 WHERE 单位价格>ANY (SELECT 单位价格 FROM 订单表 WHERE
折扣>=.25)
```

3. SQL 与数据库的维护

(1) 表的建立 CREATE TABLE 语句: 我们可以利用这个命令, 来建立一个全新的表, 但前提则是数据库必须已经存在。例如, 建立一个拥有职员姓名与部门字段的表。

```
CREATE TABLE 职员表 (姓名 TEST, 部门 TEST, 职员编号 INTEGER CONSTRAINT
职员字段 索引 PRIMARY KEY)
```

在这一实例中, 我们建立了一个表名称为“职员表”, 并且定义了该表的主键值, 以限制数据不能重复输入。

(2) 表索引的建立 CREATE INDEX 语句: 这个命令主要是对一个已经存在的表格建立索引, 例如, 在职员表中建立一个索引。

```
CREATE INDEX 新索引名称 ON 职员表(姓名部门)
```

(3) 表的删除 DELETE 语句: 我们可以利用 DELETE 语句, 将表中的记录删除。例如, 要将职员表中姓名为“李四”的记录删除, 可以利用下面的 SQL 语句来完成。

```
DELETE * FROM 职员表 WHERE 姓名= '李四';
```

注意: 记录被删除后, 无法再复原, 所以条件设置要正确。

(4) SELECT...INTO 语句: 我们可以通过这个命令, 利用已存在的表格查询, 来建立一个新表。例如, 可以通过下面的 SQL 语句, 来建立一个新的“训练名册”表。

```
SELECT 职员表.姓名, 职员表.部门 INTO 训练名册 FROM 职员表 WHERE 职称= '新进人员';
```

(5) INNER JOIN 操作数: 当某一个共同的字段数据相等时, 将两个表的记录加以组合。例如, 把分类表与产品表作组合, 可参考下面的 SQL 语句。

```
SELECT 分类名称, 产品名称 FROM 分类表 INNER JOIN 产品表
ON 分类表.分类编号=
产品表.分类编号;
```

(6) UNION 操作数: 我们可以通过 UNION 操作数来建立连接的查询条件, UNION 操作数可以将两个以上的表或是查询的结果组合起来。例如, 利用 SQL 语句, 将订单数量超过 100 的顾客表记录与新客户表作 UNION 的操作。

```
TABLE 新客户表 UNION ALL SELECT * FROM 顾客表 WHERE 订单数量>100;
```

(7) ALTER 语句: 在一个表建立后, 利用 ALTER 语句, 我们可以去修改表的字段设计。例如, 在职员表中新建一个“薪水”字段。

```
ALTER TABLE 职员表 ADD COLUMN 薪水 CURRENCY;
```

再比如, 在职员表中删除一个“薪水”字段。

```
ALTER TABLE 职员表 DROP COLUMN 薪水;
```

(8) DROP 语句: 针对所指定的表或字段加以删除, 或是把索引删除。例如, 从职员表中, 删除编号索引。

```
DROP INDEX MyIndex ON Employees;
```

再比如, 从数据库中, 删除整个表。

DROP TABLE 职员表;

(9) INSERT INTO 语句: 新建一条数据到表当中。例如, 在客户数据表中, 从新的表插入数据。

INSERT INTO 客户数据表 SELECT 新客户数据表.*FROM 新客户数据表;

(10) UPDATE 语句: 建立一个 UPDATE 的查询, 通过条件的限制来修改特定的数据。例如, 要把订单表中的订单数量修改成 1.1 倍, 运费为 1.3 倍, 可利用下面的 SQL 语句来完成。

UPDATE 订单表 SET 订单数量=订单数量*1.1, 运费=运费*1.3WHERE 运达地点= '上海';

10.3 Delphi 访问数据库的机制

在 Delphi 7 中构建数据库系统建立连接对数据库进行访问的机制主要有 ODBC、BDE、ADO 和 dbExpress, 下面进行简要的介绍。

10.3.1 ODBC

ODBC 使应用程序可以通过统一的接口访问各种数据库管理系统, 而不必依赖于某个具体的数据库管理系统。

1. ODBC 体系结构

ODBC 体系结构由应用程序 (Application)、驱动程序管理器 (Driver Manager)、驱动程序 (Driver) 和数据源 (Data Source) 等部件组成。如图 10-1。

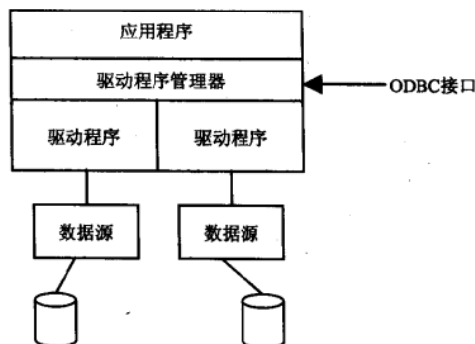


图 10-1 ODBC 体系结构

(1) 应用程序: 应用程序执行并调用 ODBC 函数来提交 SQL 语句并接收 SQL 的执行结果。应用程序可用 ODBC 接口执行以下任务:

- 请求与数据源的链接或会话。
- 发送 SQL 请求到数据源。
- 为 SQL 请求的结果定义存储区域和数据格式。
- 请求结果。
- 处理错误。

- 如有必要, 报告结果给用户。
- 为交互控制请求 commit 或 rollback 操作。
- 终止与数据源链接。

(2) 驱动程序管理器: 驱动程序管理器为应用程序装入驱动程序。除了装入驱动程序, 驱动程序管理器还执行以下任务:

- 使用 ODBC.INI 文件把数据源名映射成特定的驱动程序动态链接库。
- 处理 ODBC 初始化调用。
- 为每个驱动程序提供指向 ODBC 函数的入口指针。
- 为 ODBC 调用提供参数确认和顺序确认。

(3) 驱动程序: 每个驱动程序是一个动态链接库 (DLL), 实现 ODBC 函数调用并与数据源交互, 每个驱动程序执行以下任务:

- 建立与数据源的链接。
- 分发请求给数据源。
- 翻译数据格式。
- 返回结果给应用程序。
- 把错误格式化成标准错误代码并返回给应用程序。
- 如有必要, 声明并处理临时表 (cursor)。
- 如果数据源需要明确的事务初始化, 则初始化事务。

驱动程序的类型有两种, 即单层 (single-tier) 实现和多层 (multiple-tier) 实现。单层实现是指驱动程序不单要执行 ODBC 调用, 还要处理 SQL 语句, 最典型的例子就是对 xBase 文件的访问。而多层实现指驱动程序只处理 ODBC 调用, 而 SQL 语句直接发给数据源去处理, 例如对 Oracle 的访问就可采用多层访问, 因为后台一个个运行着的 Oracle DBMS 用于处理 SQL 语句, 这样执行的效率就会大大提高, 当然, Oracle 的访问也可采用单层访问, 由驱动程序处理 SQL 语句, 访问的信息量就要大大增加, 而且 Oracle DBMS 也没有发挥作用, 其执行效率会降低 10 倍。当然对于 xBase 这种没有 DBMS 在后台运行的数据库, 就只有单层实现。

(4) 数据源: 数据源由用户要访问的数据以及与之相关的操作系统、DBMS 和网络平台组成。例如, 用户可以通过 Novell 网来访问运行于 OS/2 操作系统上的 Oracle DBMS。

2. 连接 ODBC 数据库

下面通过一个例子讲述如何实现 ODBC 数据库的连接。

(1) 打开 Windows2000 的控制面板, 双击“数据源 (ODBC)”, 打开【ODBC 数据源管理器】对话框, 如图 10-2 所示。

(2) 选择【系统 DSN】选项卡, 单击【Add】按钮, 就会打开创建新数据源对话框, 如图 10-3 所示。

(3) 从驱动程序列表中选择“Microsoft Access Driver (*.mdb)”, 单击【完成】按钮, 则会打开【ODBC Microsoft Access 安装】对话框。选择数据库文件, 并定义数据源名称, 如图 10-4 所示。

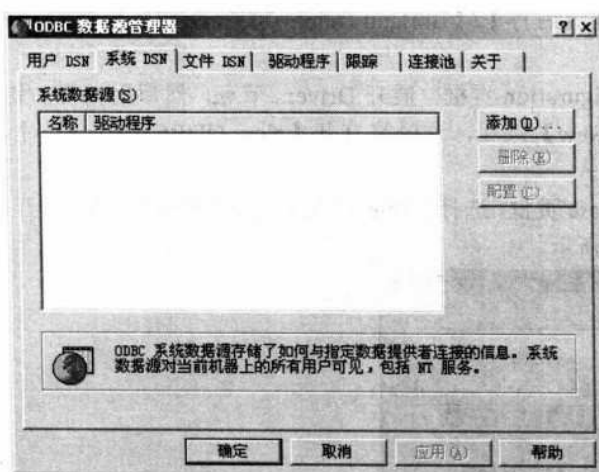


图 10-2 【ODBC 数据源管理器】对话框



图 10-3 创建新数据源对话框

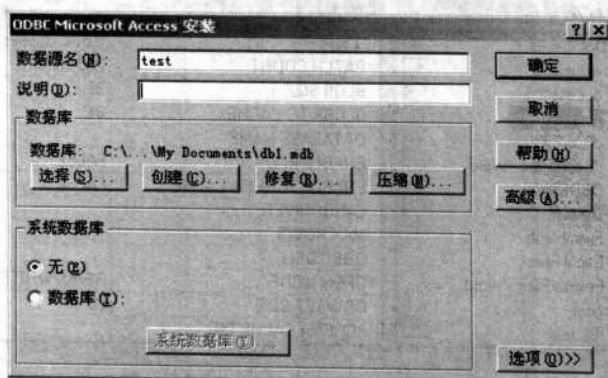


图 10-4 【ODBC Microsoft Access 安装】对话框

(4) 从【开始】/【程序】/【Borland Delphi 7】/【BDE Administrator】命令，打开 BDE 管理程序。

(5) 选择 Configuration 页框，展开 Drivers 节点，然后用鼠标右键单击 ODBC，从弹出的菜单中选择【New】选项，这时将打开【New ODBC Driver】对话框，如图 10-5 所示。

(6) 展开 Database 页框，选择【Object】菜单上的【New】命令，打开 New Database Alias 对话框，如图 10-6 所示。



图 10-5 New ODBC Driver 对话框

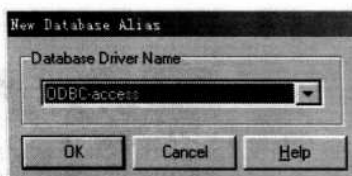


图 10-6 New Database Alias 对话框

(7) 选择“ODBC-access”作为数据库驱动程序，单击【OK】按钮。输入完整的别名 accesstry，如图 10-7 所示。最后，选择【Object】菜单上的【Apply】命令，将新建的别名保存到 BDE 配置文件中。

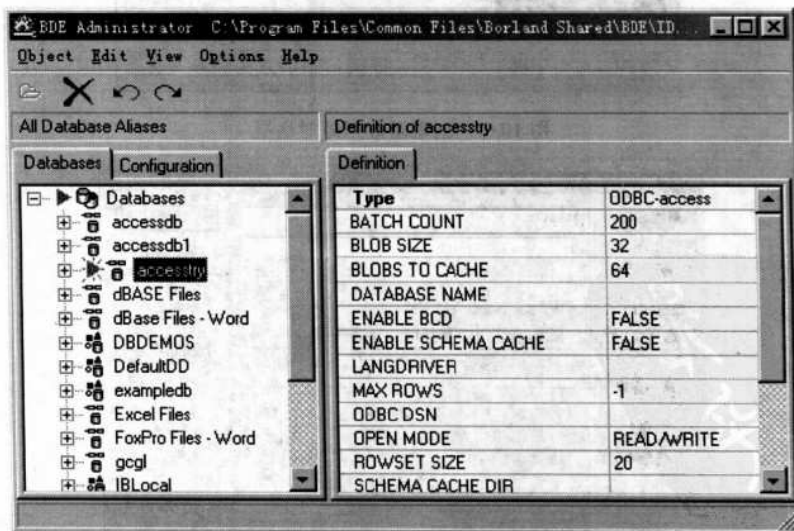


图 10-7 新的别名

10.3.2 BDE

在 Delphi5.0 前访问数据库主要是靠 Borland 公司自己开发的数据库引擎 (Borland Database Engine, 简称为 BDE)。它通过别名机制实现了建立与管理各种数据库的连接, 包括本地数据库 Paradox、dBASE、FoxPro 和 Access, 远程数据库如 Oracle、DB2、MS SQL SERVER、Sybase 等。

BDE 提供了访问各种数据库的 API 应用程序接口函数, 并且提供了基于 BDE 技术的 VCL 组件, 实现底层的调用, 另外 Delphi 基于 BDE 提供了许多简便的数据库工具, 用来直接访问数据库, 检测连接, 调试数据库应用程序的执行, 用户可以方便地建立别名, 定义和配置和各种数据库的连接, 包括数据库的物理位置、数据库的驱动程序、语言的驱动、用户登录等。另外, 利用别名大大增加了数据库应用程序的可移植性, 通过使用不同的别名来实现和多个数据库的连接。

BDE 为开发数据库应用程序提供了一致的接口, 支持以相同的方式访问本地数据库和远程数据服务器的数据, 使得程序设计的数据应用程序的接口和连接数据库的部分分开, 便于程序的开发和移植。

BDE 在 Delphi 访问数据库的机制中最为成熟和可靠的, 也是以前 Delphi 程序员使用最多一种机制。Borland 公司对 BDE 的数据连接进行了优化, 相对 ODBC 技术而言, 它在执行效率上显得更为优越, 它是连接 Paradox 和 dBase 数据表的最好的方式。

10.3.3 ADO

ADO (Active Data Objects) 是微软提供的一项技术。通过 ADO, 能让用户快速访问关系型或非关系型数据库以及 E-Mail 和文件系统。与 ODBC 一样, ADO 已经成为访问数据库的新的标准接口。

Delphi 对 ADO 的支持, 是为了让用户能迅速实现对终端用户用来做商业决策的数据的一致性访问, 结合 Delphi 本身的开放式数据组件结构, 用户可以很快地建立应用程序, 用来把自己的商业数据通过 Internet 发送给客户、最终用户以及整个销售环节。

ADO 是 Microsoft 用与连接各种数据源的高层界面, 这个技术的应用层界面是 OLE DB。用 OLE DB 可以快速存取任何数据源, 包括有关和无关的数据库, E-Mail 文件系统, 文本和图形, 以及定制的商业对象。

Delphi 7 较以前的版本, 进行了完善, 提供了 dbGO 技术, 使得 Delphi 7 对 ADO 的支持更为稳定可靠。ADO 是一个重量级的数据访问机制, 获得了广泛的支持, 在 Delphi 7 中使用也很简单。如果开发的数据库系统需要一定的可扩展性, 还是考虑用 ADO。

10.3.4 dbExpress

Delphi 7 是 Borland 在开发基于 Linux 操作系统下的 Delphi (Kylix) 的同时开发的, 为了跨平台的设计的需要, 把 Kylix 使用的数据库连接技术 dbExpress 引入到 Delphi 中来, 提供了基于 dbExpress 的 VCL 组件。DbExpress 核心是提供快速获取数据库信息的驱动, 包括整个简洁的界面, 支持跨平台的开发。基于 dbExpress 开发的程序可以作为独立的执行文件或是调用相关的 dbExpress 驱动。DbExpress 在 Kylix 中是 1.0 的版本, Borland 又修正和改善了 Kylix 中的 dbExpress, 在 Delphi 7 中则是 1.x 的版本, 现在 dbExpress 的执行

速度却已经和开发多年的 BDE 有着几乎一样的表现。

该机制最为突出的特点就是轻型和快速，不过获取的数据集为单向数据集，不支持数据更新，适合于快速获取数据生成数据报表、Web 页面等。

除了以上这些机制外，在 Delphi 7 中实现和数据库的连接还可以通过第三组件实现。另外，Delphi 7 还专门为 InterBase 提供了 IBX 组件，支持直接对 InterBase 数据库的程序设计。

10.4 Delphi 的数据库管理工具

开发一个 Delphi 7 数据库应用程序时，除了要了解 Delphi 集成开发环境中数据库组件的使用外，还要了解 Delphi 7 提供的一些开发工具。因为熟练使用 Delphi 7 提供的这些开发工具，会极大地提高我们的工作效率。

Delphi 7 提供了几个使用数据库的工具，可让用户直接修改数据库的数据，或是执行一些指令。这些关于数据库的使用工具有：

- BDE Administrator (数据库引擎)
- Database Desktop (数据库工作平台)
- SQL Explorer (数据库资源管理器)
- Data Pump (数据转移工具)
- Data dictionary (数据字典)
- SQL Monitor (SQL 监视器)

10.4.1 BDE Administrator

BDE Administrator 是数据库引擎管理工具。BDE 即 Borland Database Engine 的缩写。BDE Administrator 主要有如下功能：

- 帮助我们配置 BDE 数据库引擎。
- 可以用它来配置 STANDARD (Paradox、dBASE、Foxpro、ASCII Text) SQL、Access 和 ODBC 的驱动程序以及创建和删除驱动程序。
- 通过它我们可以管理数据库的别名。

从“Borland Delphi 7”程序组中选择“BDE Administrator”菜单项，能够打开【BDE Administrator】对话框，如图 10-8 所示。

BDE Administrator 的窗体由 3 个部分组成，上部是菜单栏；左边的窗口是驱动程序和系统中所有别名的列表，选中项的详细信息在右边窗口中显示。

在工具栏上，有 4 个速度按钮，最左边的是打开当前选中的数据库；第二个是删除当前选中的数据库；第三个用来取消上一步操作；最后一个确定所做的操作。

左边的窗口又分为【Databases】和【Configuration】两个标签页。【Databases】页列出了所有的别名，我们可以在其中创建、修改和删除所选的别名。

在【Configuration】标签页显示的是数据库引擎和 BDE 系统配置，如图 10-9 所示。【Configuration】页又有 Drives 和 System 两个子项。Drivers 子项列出了本地和 ODBC 的驱动程序，我们可以对它进行配置；System 子项是有关系统的配置，它下面有两个子节点，

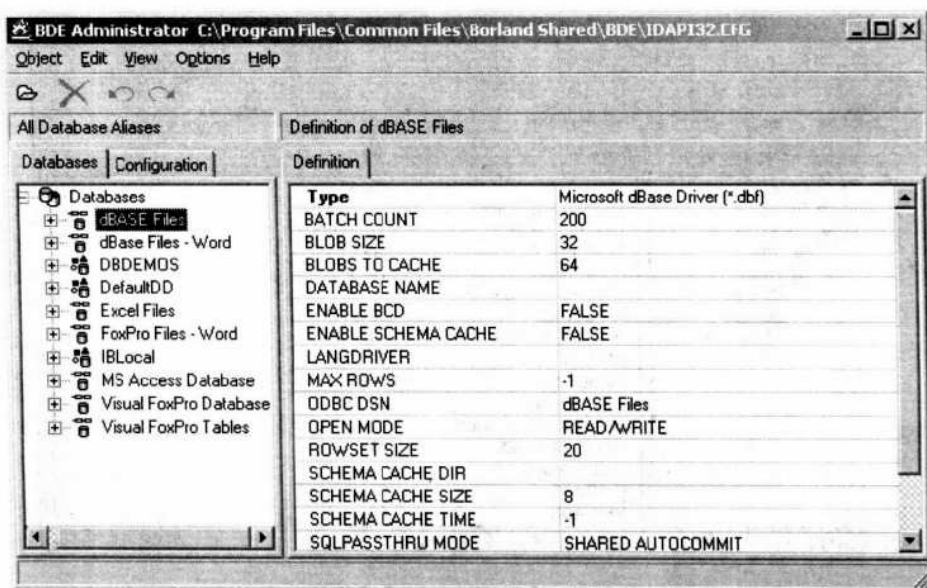


图 10-8 BDE Administrator 窗口

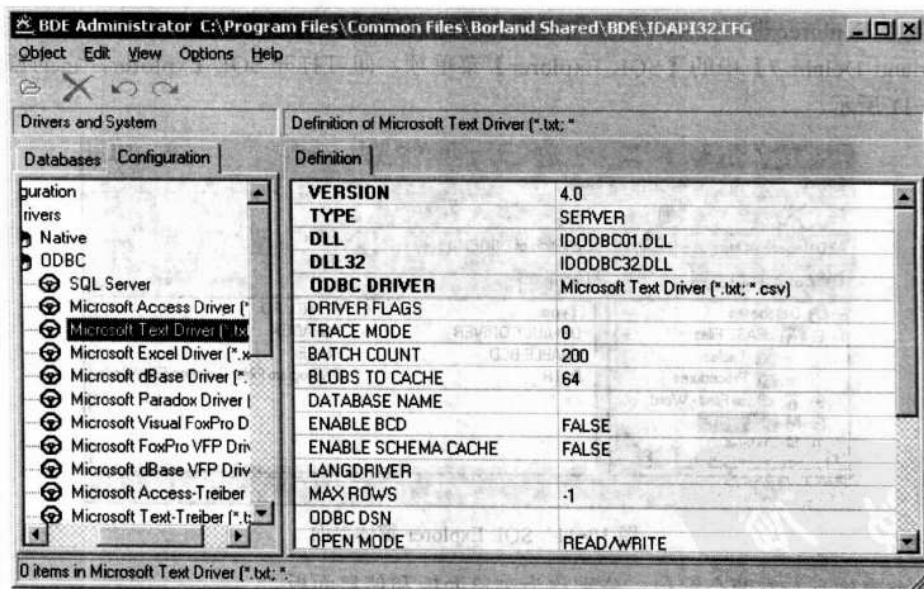


图 10-9 BDE 系统配置页

一个是 INIT，另一个是 Format 节点。INIT 用于设置 BDE 应用程序启动时的默认参数，这些参数存储在 Windows 的注册表中。Formats 节点的 3 个子节点 Date、Time 和 Number 分别用于设置日期、时间和数字的格式。如图 10-10 所示。

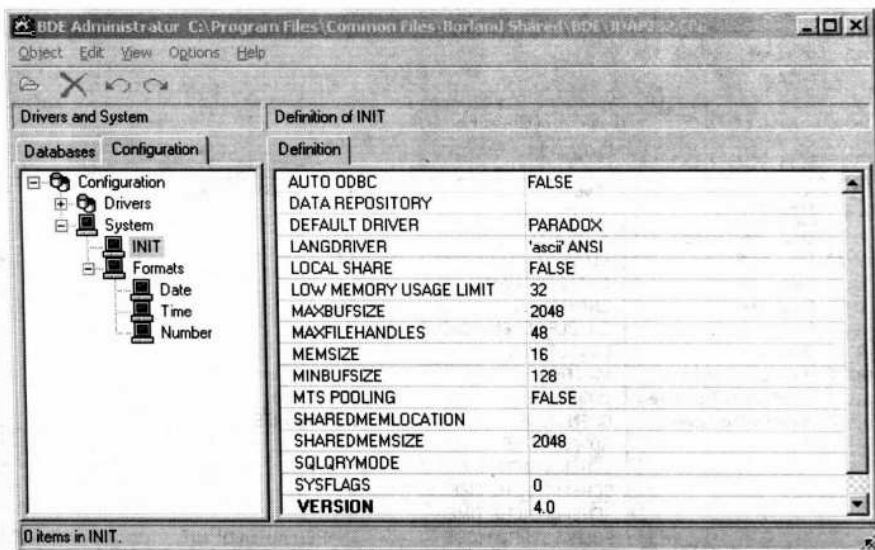


图 10-10 【Configuration】页选项

10.4.2 SQL Explorer

SQL Explorer 也叫数据库资源管理器，单击【开始】里的【程序】选项，单击其中的【Borland Delphi 7】中的【SQL Explorer】菜单项，即可打开 SQL Explorer，它的窗体如图 10-11 所示。

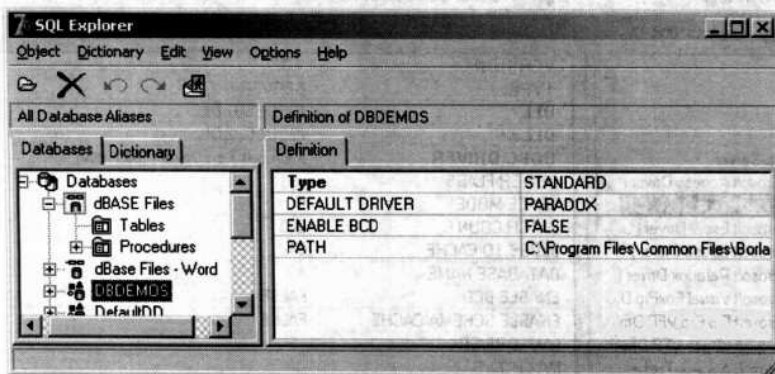


图 10-11 SQL Explorer 窗体界面

这个窗体有左右两个窗口，分别称为浏览面板和信息面板。

1. 浏览面板

在主窗口的左边部分是浏览面板，用来浏览数据库的信息，它显示了数据库的立体结构，在浏览面板中可以创建新的数据库别名和查看数据库。它有两个标签页：Databases 页和 Dictionary 页，Databases 页显示的是 BDE 的数据库；Dictionary 页显示数据字典的属性集。

数据库别名的作用是便于在数据库应用程序中对数据库进行修改。使用浏览面板可以创建新的数据库别名。在浏览面板中单击鼠标右键，从弹出菜单中选择 New 选项，出现如图 10-12 所示的对话框，选择数据库驱动类型。

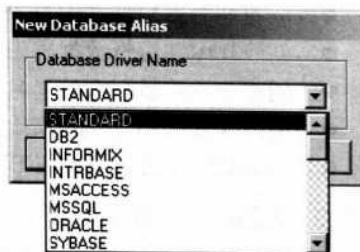


图 10-12 选择数据库驱动类型

2. 信息面板

信息面板中将显示浏览面板中所选的工程的信息。信息面板有几个不同的页面，根据选择的不同数据库的类型显示不同的页面。如图 10-13 所示。

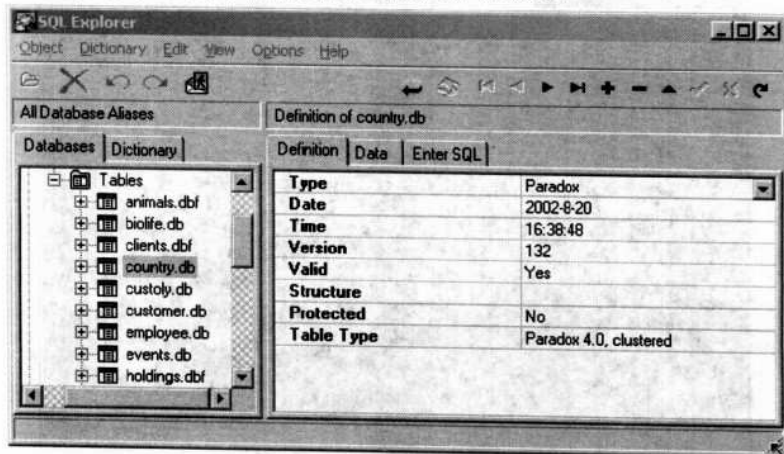


图 10-13 信息面板的信息

(1) **【Definition】** 页面：显示浏览面板中所选中的工程的性质。选择的工程不同，显示的内容也不同。选择数据库别名显示的是数据库与 BDE 配置信息，选择数据库中的数据表格显示的是数据表格类型、数据表格的建立日期等，选择数据表格的字段显示的是字段的结构信息。

(2) **【Data】** 页面：可以查看选中的数据表格的数据，并可以修改选中的数据，同时还提供了窗口来显示备注和图片信息。

(3) **【Text】** 页面：只有在 C/S 数据库管理器中有，它能够向数据库查询选中的数据的信息，数据信息转化为 SQL 语句显示出来，这个页面用于 C/S 类型的数据库应用程序。

(4) **【Enter SQL】** 页面：只有在专业版和 C/S 的 Delphi 中有，通过 Enter SQL 页面可以对所选的数据库执行 SQL 语句。

3. SQL Explorer 的功能

SQL Explorer 主要用于对数据库进行集中管理。使用这个工具我们可以完成下面这些功能：

- 浏览和编辑数据库的对象。
- 创建视图和编辑数据表中的数据。
- 创建和维护数据库的别名。
- 输入 SQL 语句查询特定的信息。
- 引导数据库管理员配置 BDE。

10.4.3 Database Desktop

Database Desktop (数据库工作平台) 可以创建、查看和修改数据表的结构。有两种方法可以打开它, 一是像打开 BDE Administrator 那样打开它, 只是在工具选项里选择 **【Database Desktop】**; 另一种方法是在 Delphi 的集成开发环境的主菜单的 **【Tools】** 菜单项下选择 **【Database Desktop】**, 这样也可打开它。Database Desktop 窗体如图 10-14 所示。

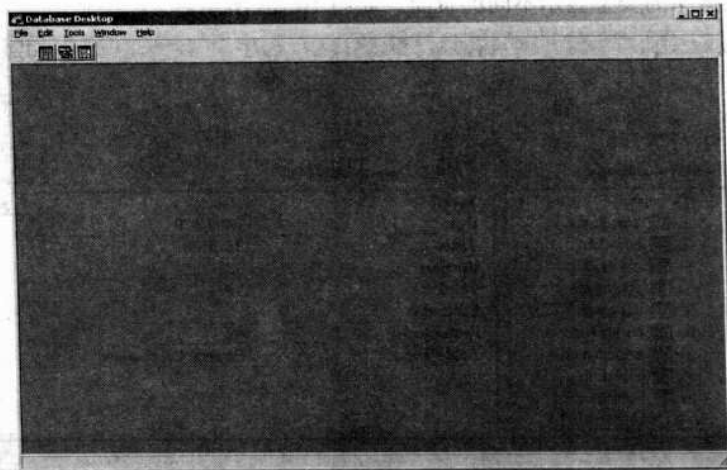


图 10-14 Database Desktop 窗体

Database Desktop 窗体中的主菜单和工具栏是动态显示的。它根据当前用户所操作的对象动态地显示相应的菜单项。要显示全部的菜单可单击主菜单的 **【Edit】** 菜单下的 **【References 命令】**, 在弹出的窗口中选择 **【Toolbars】** 页, 将 **【Global】** 选项打勾后单击 **【确定】** 按钮即可。如图 10-15 所示。

Database Desktop 有 4 个对象, 它们分别是表 Table、查询 Query、SQL 文件和临时文件。Database Desktop 支持查看和创建 Paradox、Dbase、Interbase 等多种类型的表。

要创建一个表可单击主菜单的 **【File】** 中的 **【New】** 的 **【Table】** 选项, 在弹出的菜单中选择要创建的表的类型, 系统就会为你自动创建一个该类型的表。

查询对象用来对表中的数据进行查询, 可以是单表查询也可以是多表查询。要创建一个查询对象需要在 **【New】** 菜单下选择 **【QBE Query】** 菜单项, 在弹出的对话框中选择要查询的文件就可以对这些表的数据进行查询。

SQL 文件就是用 SQL 语言所写的查询代码, 这些查询代码可通过 BDE 查询本地的数据表, 也可通过 SQL Links 查询远程数据表。要创建一个 SQL 文件可在 **【New】** 菜单下选择 **【SQL File】** 菜单项。

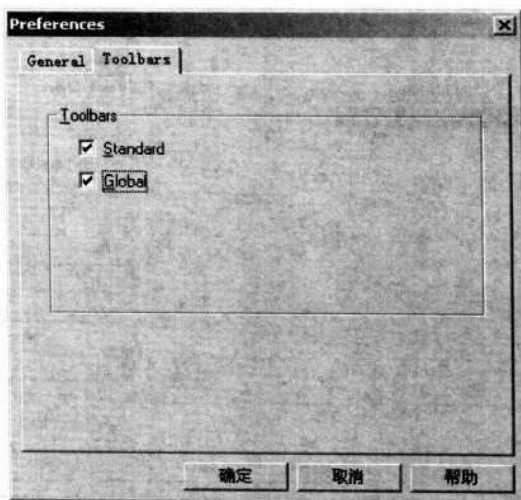


图 10-15 References 命令窗口

临时文件是在使用 Database Desktop 的时候自动生成的，可以对它们进行编辑。临时文件存放在私有目录下，改变了私有目录后，这些临时文件将自动删除。临时表中的数据有时候是很重要的，当表的结构被修改后，有些数据不能适应新的结构，它们被存放到临时表中，可以修改临时表中的记录并把它们添加到新表中。

Database Desktop 有两个目录：一个是被称为 Working Directory 的工作目录，它是在 Database Desktop 打开和保存文件时的默认路径。如图 10-16 所示。

另一个是被称为 Private Directory 的私有目录，它用来存放临时的表和查询的结果。如图 10-17 所示。

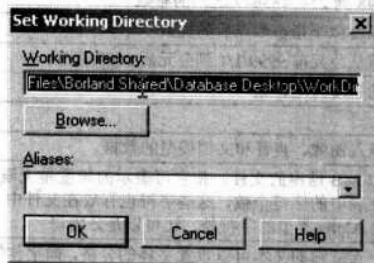


图 10-16 工作目录

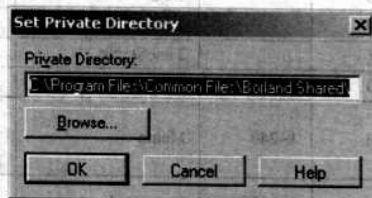


图 10-17 私有目录

这两个目录可通过 File 菜单下的【Working Directory】或【Private Directory】菜单项来设置。

10.4.3.1 创建数据表

下面我们通过一个具体地创建一个数据表来说明 Database Desktop 的用法。

【例 10-1】使用 Database Desktop 创建一个数据表

(1) 选择【File】/【New】/【New】菜单命令，在弹出的窗口中选择“Paradox7”，单击【确定】按钮，创建一个 Paradox7 类型的表，如图 10-18 所示。

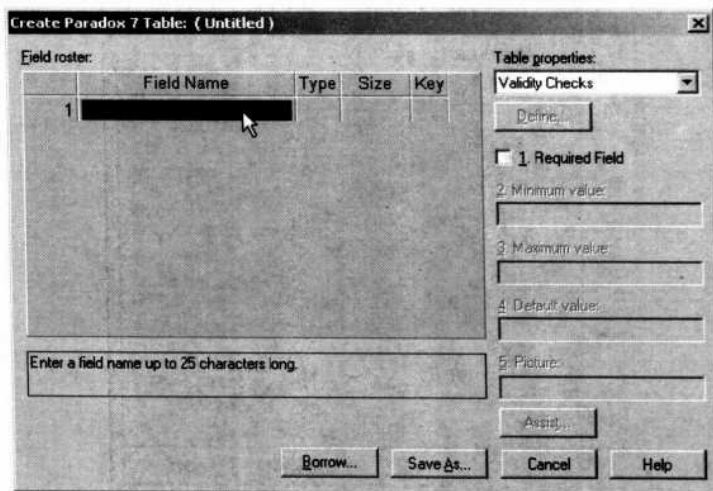


图 10-18 Paradox7 类型生成数据表框架

(2) 在表的【Field Name】下面填写表的字段名，在【Type】下面单击鼠标右键，在弹出的菜单中选择该字段的类型，Paradox 表字段的类型如表 10-1 所示。

表 10-1 Paradox 表字段的类型

符 号	长 度	类 型	说 明
A	1~255	Alpha	可以是字符、数字、可打印的字符以及特殊符号，如#、@、%、&等
N		Number	可以表示 15 位的数，其中包含两位数
\$		Money	表示货币金额，带有一个钱币符号和两位小数
S		Short	表示从 32767~+32767 之间的数，所需空间比 Number 要少
L		LongInteger	可表示从-2147483648 到 2147483647 之间的整数
#	0~32	BCD	计算精度高，它的长度似乎指小数点后的位数
D		Date	用来表示日期，范围从公元前 9999/1/1 到公元后 9999/12/31
T		Time	用来表示时间，它的精度为毫秒级
@		TimeStamp	时间和日期综合
O	0~240	OLE	在这个字段中可以嵌入图像、声音和文档类型的数据
M	1~240	Memo	系统给它分配一个以.MB 结尾的文件，使它可表示的长度是无限的。它的长度表示要存放在表中的字符个数，这些字符也存放在文件中。它的文本字符构成除了 Alpha 类型的字符外，还可以是分隔符
F Memo	0~240	Formatted	与 Memo 类型类似，支持它的文本可以设置字体的分格，如字体的大小、颜色等
G	0~240	Graphic	这种类型可以存放.BMP、.PCX、.TIF、.GIF、.EPS 类型的图像，这些图像最终都被转化成.BMP 图像
L		Logical	表达是真或假逻辑
B	0~240	Binary	以二进制的形式存放数据，它的值存放在外部文件中
Y	1~255	Bytes	与 Binary 字段类似，可以存放如条码等数据，它可以设定长度，它的数据存放在表中
±		Autoincrement	自动增量

提示：在输入字段类型的时候，只需输入字段类型的第一个字符就可以了。

(3) 在 Size 的下面填写该字段的长度。

(4) 定义表的关键字段。将某字段指定为关键字段可在该字段的【Key】项中双击鼠标, 会出现一个星型符号, 表示这个字段是关键字段。一个表可以定义多个关键字段。如图 10-19 所示。

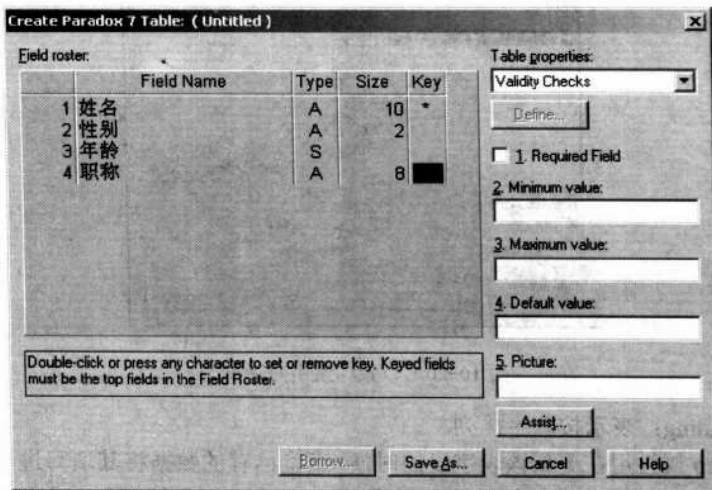


图 10-19 Paradox7 类型建立表框架

提示: 要删除该关键字段只需再次双击鼠标

(5) 字段的有效性定义。在 Table Properties 的下拉列表框中选择【Validity Checks】, 并在其下的几项中填写该字段的有效性数据。

如果选择【Required Filed】属性, 就表明这个字段不能为空。【Minimum value】文本框和【Maximum value】文本框分别规定了输入字段的最大值和最小值。【Picture】定义该字段的掩码格式。

副索引用于对临时表排序, 在它排序的时候可以不理会所定义的关键字段。在 Paradox 中除了 Memo, Formatted Memo, Binary, OLE, Graphic, Logical, Bytes 等类型不能建立副索引外, 其他字段都可以建立副索引。副索引可以加快表的查询与定位的速度, 但这个副索引必须是对大小写敏感和能自动维护的单字段。

(6) 建立副索引的方法是在 Table Properties 的下拉列表框中选择 Secondary Indexs, 然后单击 Define 按钮, 如图 10-20 所示。在弹出的窗口中将要定义的副索引从左边的列表框中添加到右边的列表框中即可。创建好后可按【OK】按钮确认; 要想放弃建立的副索引, 可按【Erase】按钮。

通过按 Change order 旁边的两个按钮可以改变副索引的顺序。Database Desktop 将按此顺序对记录依次排列。窗口下面四个复选框的含义如下:

- Unique: 表示该字段作为索引字段时不能有相同的记录。
- Case Sensitive: 表示排序时大写字符排在小写字符之前。
- Maintained: 表示要求 Database Desktop 在表被修改了之后, 自动维护表的索引文件。

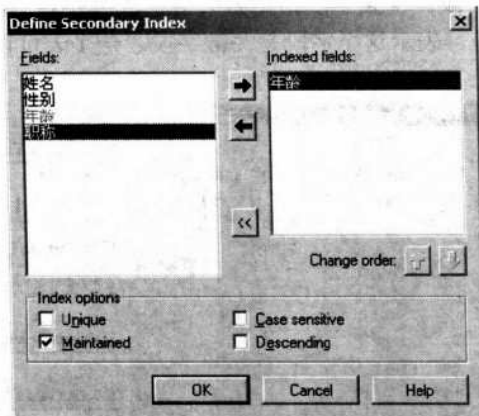


图 10-20 设置副索引对话框

- **Descending:** 表示按降序排列。

表的 **Lookup** 属性用在更新该表字段的时候, 查询已有的数据将其填写进当前的记录中。

(7) 定义表的 **Lookup** 属性的方法是在【Table Properties】的下拉列表框中选择【Table Lookup】然后单击【Define】按钮(单击 **Modify** 按钮可以对已定义的内容进行修改), 如图 10-21 所示, 在窗口的左边选择要查询的字段, 在窗口的右边选择要查询的表。这几项可以自由组合, 产生相应的含义。

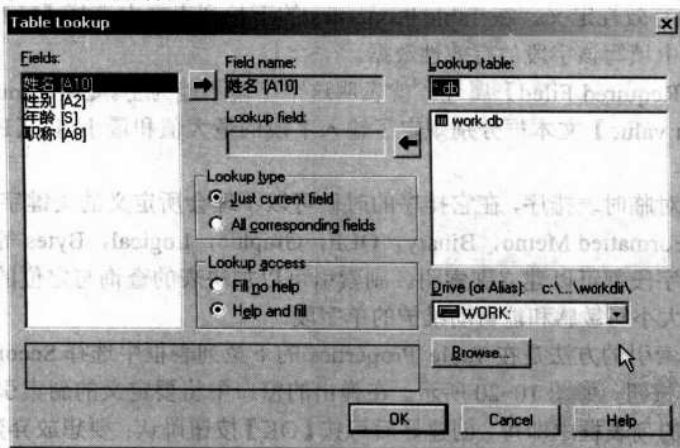


图 10-21 Lookup 对话框

窗口中的两组单选框中选项的含义如下:

- **Just current field:** 在被查询的表中可能有几项与该表的字段相对应, 选择此项时填写当前的字段
- **All corresponding field:** 选择此项时将所有满足条件(字段的长度和类型相同)的字段都填写。
- **Fill no help:** 在填写时不能按“Ctrl+Space”打开被查询的表。

- **Help and fill:** 在填写时可以按“Ctrl+Space”打开被查询的表。

(8) 表的完整性定义。一个表如果定义了完整性校验后, 在对该表进行更新数据时将判断此数据是否已在父表(被查的表)中被定义, 如果没有被定义将不允许更新。表完整性定义方法是在【Table Properties】的下拉列表框中选择【Referential Integrity】, 然后单击【Define】按钮, 如图 10-22 所示。弹出的窗口中左边是子字段(即要更新的字段), 右边是父表, 单击左边的子字段和右边的表将其连接起来。

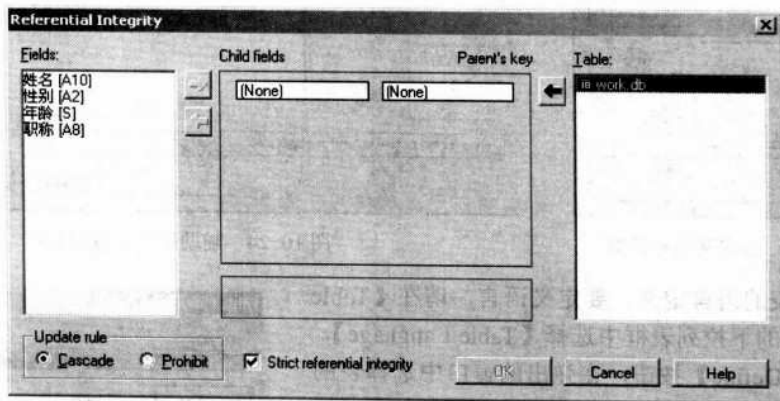


图 10-22 完整性对话框

在窗口的下面有一个单选框, 它是用来定义更新规则的, 其中各项的含义如下:

- **Cascade:** 此项被选中时, 当父表中的记录项被更新, 子表中相应的项也会被更新。
- **Prohibit:** 此项被选中时, 如果子表中还存在父表中的记录项, 父表就不能删除此记录项。

注意: 进行完整性定义的字类型和长度必须相匹配, 并且当前表与定义了主索引的父表只能被放在同一个目录中。

表的安全性定义: Database Desktop 有两种类型的密码, 一种是用于控制整个表的主密码, 另一种是辅助密码, 它可以为某个字段或多个字段设置密码。只有设置了主密码才能设置辅助密码。

(9) 在【Table Properties】的下拉列表框中选择【Password Security】, 然后单击【Define】按钮, 在弹出的窗口输入用户的密码, 完成后单击【OK】按钮即可, 如图 10-23 所示。

(10) 要设置辅助密码可单击【Auxiliary Password】按钮, 在弹出的窗口中为字段设置密码: 首先单击【New】按钮, 然后在右边的单选框中选择一种权力, 最后单击【Add】按钮将密码添加到左边的列表框中, 如图 10-24 所示。

一个密码只能设置一种权力。设置好的密码可单击【Change】按钮进行修改或单击【Delete】按钮将其删除。最后单击【OK】按钮保存这些密码。以后要访问这些表或是记录就得输入相应密码。要删除保存后的密码可单击主菜单【Tool】选项的【Password】菜单项, 在弹出的对话框中进行删除。

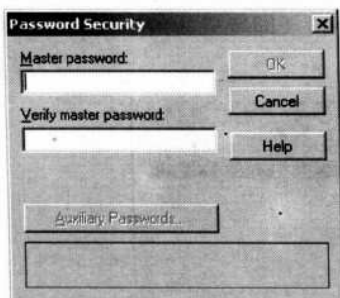


图 10-23 设定密码对话框

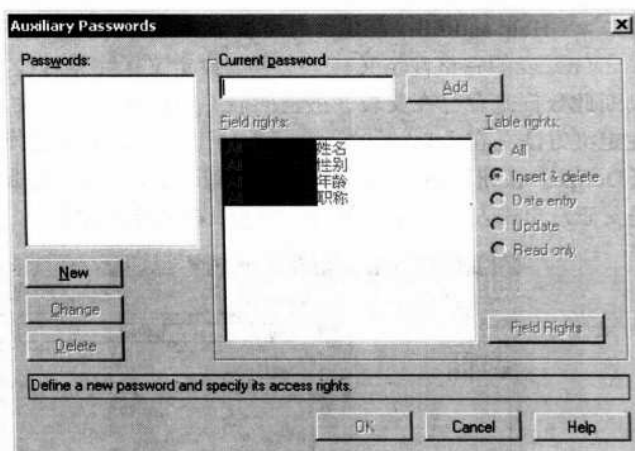


图 10-24 辅助密码设置对话框

(11) 表的语言定义。要定义语言，请在【Table Properties】的下拉列表框中选择【Table Language】，然后单击【Define】按钮，在弹出的窗口中选择表的语言，如图 10-25 所示。

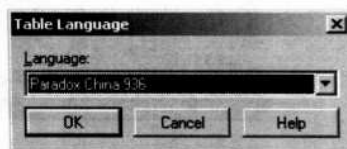


图 10-25 选择语言驱动程序对话框

(12) 最后单击【Save】或【Save As】按钮，将创建的表保存。

(13) 我们重新打开该表，单击工具栏上的最后一个按钮，即【Edit Data】按钮，可以为表添加数据，如图 10-26 所示。

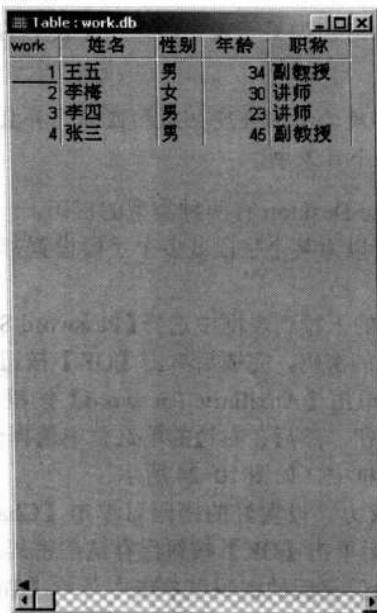


图 10-26 添加数据对话框

(14) 修改表的别名。单击主菜单【Tools】下的【Alias Manager】菜单项，将打开【Alias Manager】对话框，如图 10-27 所示。

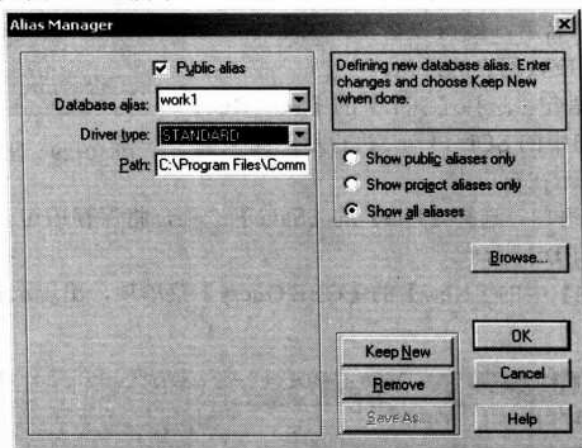


图 10-27 Alias Manager 对话框

在对话框中单击【New】按钮；在【Database Alias】中选择或输入别名；在【Driver Type】中选择驱动程序的类型；在【Path】中输入路径（也可单击【Browse】选择路径）；最后单击【Keep New】按钮保存设置。

到此为止，一个完整的表创建完毕。

10.4.3.2 表的查询使用

下面具体介绍一下 Database Desktop 如何使用查询。

【例 10-2】使用 SQL 查询

(1) 新建查询。选择【FILE】中的【New】的【SQL File】菜单项，打开一个新的 SQL 语句窗口，如图 10-28 所示。可以直接在 SQL 窗口中输入语句，此时 Database Desktop 的主菜单中将增加【Search】和【SQL】两项，同时增加了几个快速按钮来完成查询功能。

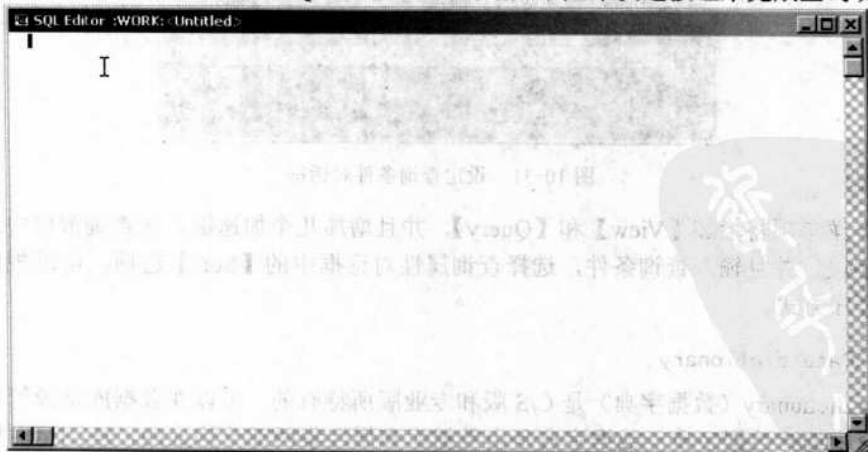


图 10-28 SQL 语句窗口

(2) 设定别名。选择【SQL】菜单下的【Aliasese】选项，将出现如图 10-29 所示的设置数据库别名对话框，设置了数据库别名后，就可以在文本编辑框中输入操作数据库别名相应数据表的语句。

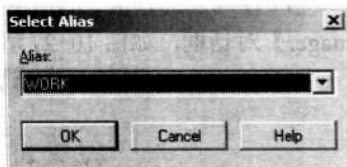


图 10-29 设定查询语句的别名

(3) 执行 SQL 语句。选择【SQL】的【SQL Run】选项，执行 SQL 窗口中的 SQL 语句，执行后的结果会以数据表格的形式显示在窗口中。

(4) 保存 SQL 语句。选择【File】的【Save】选项，将保存语句。

[例 10-3] 使用 BDE 查询

(1) 选择【FILE】中的【New】的【QBE Query】菜单项，出现选择数据文件对话框，如图 10-30 所示。

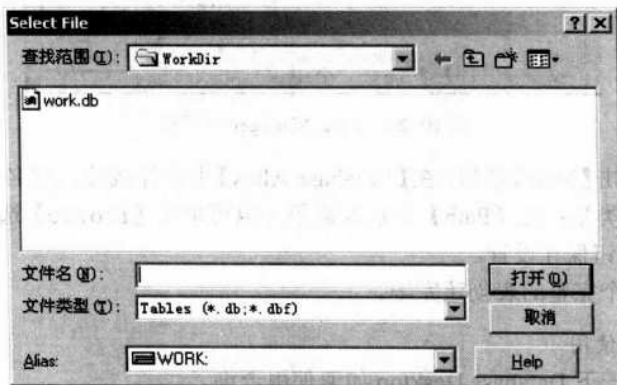


图 10-30 选择数据库文件

(2) 选定数据文件后，出现如图 10-31 所示设定查询条件对话框。

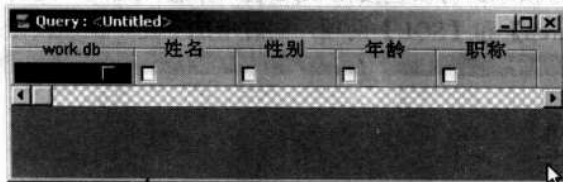


图 10-31 设定查询条件对话框

(3) 菜单项将增加【View】和【Query】，并且增加几个加速键，在查询窗口中选择要显示的字段，并且输入查询条件，选择查询属性对话框中的【Sort】选项，可以规定查询结果的排序方式。

10.4.4 Data dictionary

Data dictionary (数据字典) 是 C/S 版和专业版所特有的，可以在数据库资源管理器的浏览面板中选择数据字典，数据字典是由数据库和属性集两部分构成的，在浏览面板中选择【Dictionary】页面，就可以访问数据字典。

10.4.5 Dataump

Dataump 是 Delphi 7 中数据库之间转移的工具, 在 Delphi 7 程序中选择 Dataump 工具, 按照如下的步骤完成数据转移:

- (1) 为源数据库选择一个数据库别名, 如图 10-32 所示。
- (2) 单击【Next】按钮, 选择目标数据库别名, 如图 10-33 所示。
- (3) 单击【Next】按钮, 选择要从源数据库转移的数据表, 如图 10-34 所示。
- (4) 单击【Next】按钮, 将出现一个窗口, 用于修改被转移的数据表, 如图 10-35 所示。
- (5) 修改数据表的结构, 以达到和目标数据库一致。
- (6) 修改完成后, 单击【Upsize】按钮, 向目标数据库转移数据。

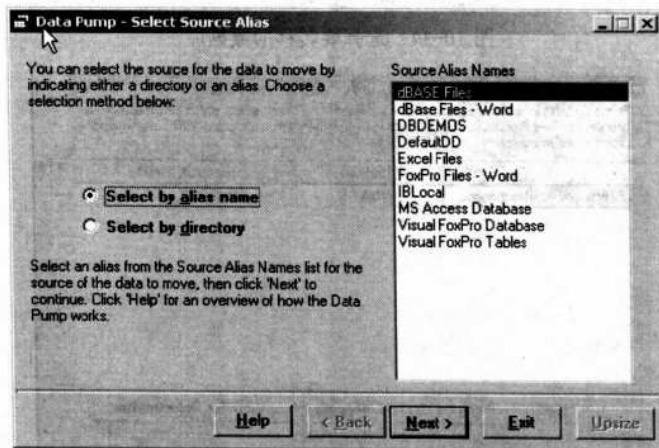


图 10-32 选择源数据库的数据库别名

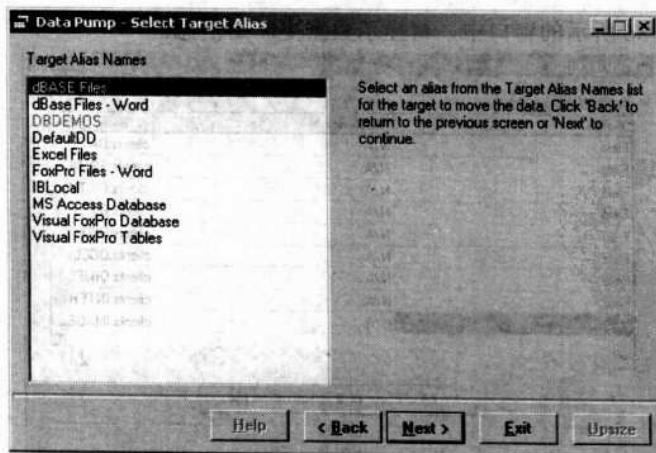


图 10-33 选择数据库别名

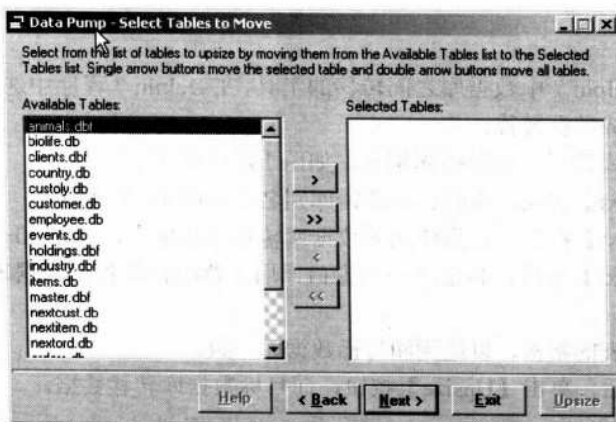


图 10-34 选择要转移的数据库

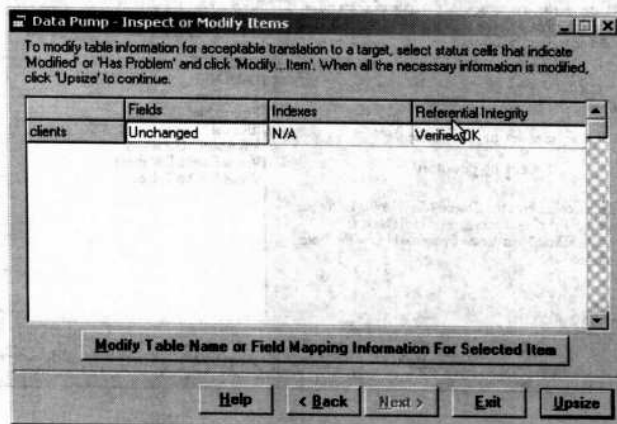


图 10-35 修改表

(7) 在图 10-36 所示的窗口中，将显示数据更新的信息。

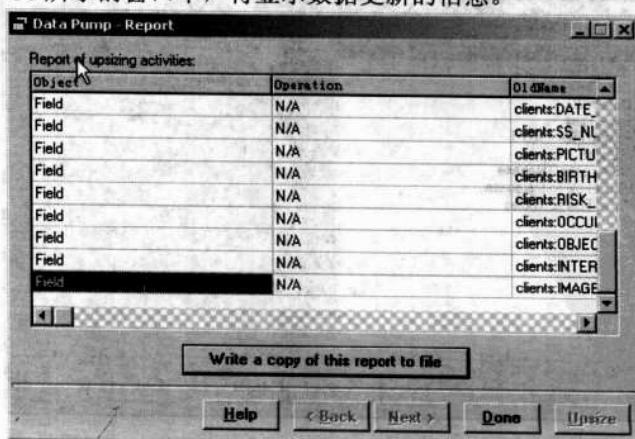


图 10-36 转移数据表的信息

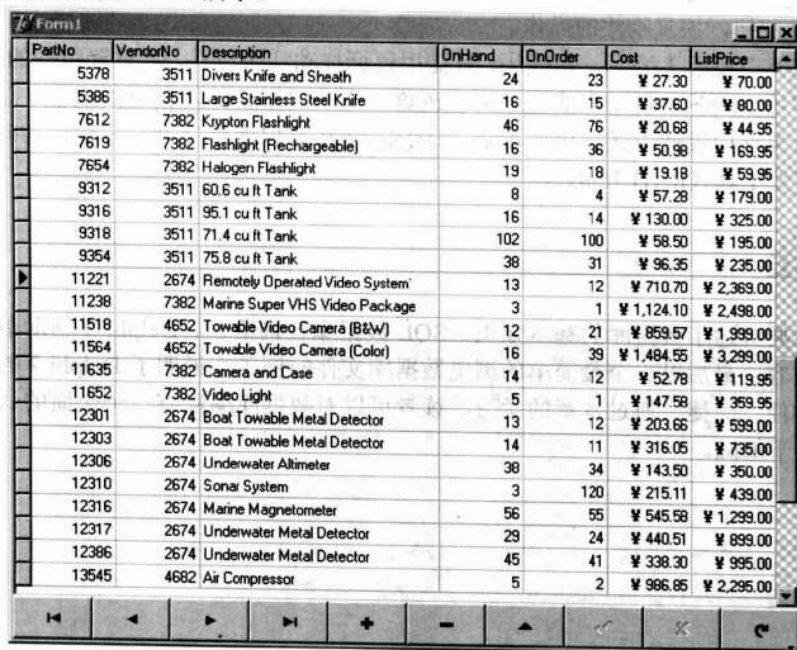
(8) 单击【Done】按钮，完成数据的转移。

10.5 建立第一个数据库应用程序

为了见识一下利用 Delphi 7 来撰写数据库应用程序的威力，我们先来实际建立一个简单的应用程序，验证一下到底有多强。

【例 10-4】建立一个简单数据库应用程序。

效果：在窗体中显示一个包含全部数据的完整数据库，通过窗体中按钮，实现数据的浏览。执行效果如图 10-37 所示。



PartNo	VendorNo	Description	OnHand	OnOrder	Cost	ListPrice
5378	3511	Divers Knife and Sheath	24	23	¥ 27.30	¥ 70.00
5386	3511	Large Stainless Steel Knife	16	15	¥ 37.60	¥ 80.00
7612	7382	Krypton Flashlight	46	76	¥ 20.68	¥ 44.95
7619	7382	Flashlight (Rechargeable)	16	36	¥ 50.98	¥ 169.95
7654	7382	Halogen Flashlight	19	18	¥ 19.18	¥ 59.95
9312	3511	60.6 cu ft Tank	8	4	¥ 57.28	¥ 179.00
9316	3511	95.1 cu ft Tank	16	14	¥ 130.00	¥ 325.00
9318	3511	71.4 cu ft Tank	102	100	¥ 58.50	¥ 195.00
9354	3511	75.8 cu ft Tank	38	31	¥ 96.35	¥ 235.00
11221	2674	Remotely Operated Video System	13	12	¥ 710.70	¥ 2,369.00
11238	7382	Marine Super VHS Video Package	3	1	¥ 1,124.10	¥ 2,498.00
11518	4652	Towable Video Camera (B&W)	12	21	¥ 859.57	¥ 1,999.00
11564	4652	Towable Video Camera (Color)	16	39	¥ 1,484.55	¥ 3,299.00
11635	7382	Camera and Case	14	12	¥ 52.78	¥ 119.95
11652	7382	Video Light	5	1	¥ 147.58	¥ 359.95
12301	2674	Boat Towable Metal Detector	13	12	¥ 203.66	¥ 599.00
12303	2674	Boat Towable Metal Detector	14	11	¥ 316.05	¥ 735.00
12306	2674	Underwater Altimeter	38	34	¥ 143.50	¥ 350.00
12310	2674	Sonar System	3	120	¥ 215.11	¥ 439.00
12316	2674	Marine Magnetometer	56	55	¥ 545.58	¥ 1,299.00
12317	2674	Underwater Metal Detector	29	24	¥ 440.51	¥ 899.00
12386	2674	Underwater Metal Detector	45	41	¥ 338.30	¥ 995.00
13545	4682	Air Compressor	5	2	¥ 986.85	¥ 2,295.00

图 10-37 执行效果

(1) 新建一应用程序工程。在窗体上放置 Table 组件，设置该组件的【DatabaseName】属性为“DBDEMOS”，【TableName】属性为“parts.db”，【Active】属性为“True”。











(2) 再选取一个 DataSource 组件加入到窗体中，并将其【DataSet】属性设置为“Table1”。

(3) 选择一个 DBGrid 组件放置到窗体中，设置它的【DataSource】属性为“DataSource1”，【Align】属性为“alClient”。

(4) 最后将 DBNavigator 组件加入到窗体中，设置它的【DataSource】属性为“DataSource1”，【Align】属性为“alBottom”。

(5) 到此为止，所有程序设计工作完毕，选择【Run】菜单中的【Run】命令（或 F9）运行程序，全部的数据都能够显示出来。

可以用光标键来浏览各个记录以及记录中的各个字段，或用下面的 DBNavigator 组件的按钮来浏览各个记录。

-  按钮, 光标定位到第一个记录;
-  按钮, 光标向上移动一个记录;
-  按钮, 光标向下移动一个记录;
-  按钮, 光标定位到最后一个记录;
-  按钮, 可以在当前增加一个空白记录;
-  按钮, 删除当前的记录;
-  按钮, 停止光标定位操作;
-  按钮, 确认修改;
-  按钮, 取消修改;
-  按钮, 恢复刚才的操作。

该程序不需要手工编写一句代码, 只需用鼠标拖动几个组件, 再设置组件的几个属性, 其余处理都由 Delphi 7 自动完成, 就可以浏览一个数据库文件, 甚至修改它, 这就是所谓可视化的对象导向设计方式。由此可知, Delphi 7 在数据库程序设计方面, 的确称得上是一套快速开发工具 (RAD Tools)。

10.6 小结

本章主要讲述了数据库的基本知识, SQL 数据库查询语言、Delphi 7 辅助数据库工具等方面的内容, 最后以一个最简单的浏览数据库文件的例子, 说明了 Delphi 7 的数据库开发非常的方便、简捷。通过本章的学习, 读者可以对数据库编程有一个全面的认识, 为下一章学习打下基础。



第 11 章 开发数据库应用程序

不写一行程序也可进行数据库的程序设计,这充分体现了 Delphi 7 的可视化的优点所在。但现实问题的复杂多样,使得程序的开发变得非常困难,以至于每个开发者必须既是思想家又是艺术家,同时具有为用户着想的极大热情。为避免各种意外情况的出现,程序必须具有极好的柔性,同时具有极好的刚性。因此,从这一点来看,程序设计的难度增加了,然而 Delphi 7 提供的开发工具使得设计者不再为一个界面花费心血,将注意力集中到程序的安全、友好和速度上来。

编写数据库应用程序,就是利用数据库组件完成预定的功能。本章将介绍数据集组件、数据控制组件以及报表组件等。


11.1 数据集组件

Delphi 7 使用面向对象的组件来生成数据库应用程序。数据集组件均被看做对象,它们具有属性,程序员可以对这些属性进行设置,在程序运行过程中也可对这些属性进行读写和修改。

开发数据库的应用程序必须与数据库之间建立联系,组件板上的数据访问组件(Data Access),又叫数据存取组件,提供了这种联系方法。它们通过使用数据库引擎访问数据库,提供了用户界面与数据库数据信息之间的联系。这样,应用程序开发者可以方便地通过设计用户界面来与数据库信息进行交互,从而使应用程序开发者更多地注意用户界面的设计,而不是如何建立与数据库之间的联系。

在数据访问组件中,Table、Query、StoreProc 三个组件是用来联系应用程序和数据库信息的,而 DataSource 组件是用来联系数据库信息和数据控制组件的。简单地说,就是数据访问组件允许应用程序通过 BDE 访问数据库,将数据库信息传递给用户接口,并通过 BDE 将用户接口的信息反馈给数据库,完成用户与数据库之间的信息交互。

11.1.1 Table (表) 组件

Table 组件 , 位于【BDE】选项卡上,由 BDE 从一个物理数据库表文件中取出数据,并通过 DataSource 组件把数据提供给一个或多个数据控制组件;另一方面,Table 还把用户通过数据控制组件输入的数据通过 BDE 发送给物理数据库。

11.1.1.1 Table 组件的主要属性

1. DatabaseName 属性

字符串类型,数据表组件通过此属性指明需要访问的数据库。该数据库可以是 BDE 定

义的数据库别名, 如 DBDEMOS; 也可以是数据库文件, 如 Paradox 和 dBASE 等文件做的路径; 还可以是由 DataBase 组件定义的数据库名。应用最多的还是用 BDE 定义的数据库别名。因为这样可以避免在应用程序所操作的数据库发生变化时西怪整个应用程序, 而只需要重新设置一下 BDE 定义的数据库别名即可。例如下面的代码就说明了该属性的使用。

```
Table1.Active:=False; //更改数据集前, 表必须先关闭
try
    Table1.DatabaseName:='work1'; //试用数据库别名
    Table1.Active:=True;
except
    on EdatabaseError do
        //如果试用失败, 改用数据库文件的路径名
        Table1.DatabaseName:=C:\\work\\database';
        Table1.Active:=True;
end;
```

2. TableName 属性

字符串类型, 它指明了被访问数据库表的名称。它和 DataBaseName 一样, 是在设计阶段给定的, 在 DataBaseName 设定后给出。一般在 DataBaseName 设定后, 如果有符合将打开表的文件格式的, 会自动在 TableName 的属性框中生成一个下拉列表, 包含所有符合该文件格式的数据库文件

注意: 设定 TableName 属性值时, Active 属性值一定要设置为 False。

3. Active 属性

布尔类型, 它表明了数据表组件是否与数据库建立连接。如果 Active 属性值为 True, 则表明数据集已经和数据库建立连接, 可以使用数据表组件访问数据库表; 否则表明数据集处于关闭状态, 不能通过数据表组件访问数据库的表。

注意: 用 Open 和 Close 方法对数据库文件操作的效果与改变 Active 属性的效果是一样的。调用 Open 方法会将 Active 设置为 True; 调用 Close 方法会将 Active 设置为 False。

以上 3 种属性在创建 Table 组件时是必不可少的, 也是最基本的。这些属性值的设置可以通过对象观察器, 也可以在程序代码中实现。

4. TableType 属性

该属性用来说明当前应用程序所操作的数据库表的类型。该属性取值如表 11-1 所示。

表 11-1 TableType 属性的取值以及含义

取 值	含 义
ttDefault	数据库表类型由文件的扩展名决定
ttParadox	数据库表类型是 Paradox 表
ttASCII	数据库表类型是 ASCII 表
ttDBase	数据库表类型是 DBase 表
ttFoxPro	数据库表类型是 FoxPro 表

5. ReadOnly 属性

布尔类型,通过 ReadOnly 属性可以防止数据库表中的记录数据被修改。如果 ReadOnly 属性值设置为 True,则表示数据库表的数据不能被修改。

6. Bof 和 Eof 属性

Bof (Beginning of file) 属性值为布尔类型,表示当前记录指针所处的位置为数据库表的第一个记录。

Eof (end of file) 属性值为布尔类型,表示当前记录指针所处的位置为数据库表的最后一个记录。

如果 Bof 和 Eof 都为 True 的话,则数据库表为空。

7. CanModify 和 Modified 属性

CanModify 属性为布尔类型,如果该属性值为 True,表明此数据库表可以被修改。反之,说明此数据库表上只读的。CanModify 属性在应用程序打开一张数据库表时被自动设置。如果该表组件的 ReadOnly 属性为 True,那么 CanModify 属性将被设置为 False。

Modified 属性的值为布尔类型,如果值为 True,则此数据库表已经被修改。反之,如果 Modified 属性值为 False,此数据表没有别修改。

8. Exclusive 属性

布尔类型,Exclusive 属性用来设置表的打开方式。如果 Exclusive 属性被设置为 True,则数据库以独占方式打开。当一个应用程序以独占方式打开一个数据库表后,其他应用程序将不能打开此表。

下面的程序代码实现对数据库进行独占式访问。

```
procedure TForm1.FormCreate(Sender:TObject);
begin
    if Table1.Active:=True then
        Table1.Active:=False;
    repeat
    try
        Table1.Exclusive:=True;
        Table1.Active:=True;
        Break;
    except
        on EdatabaseError do
            if Application.MessageBox('Could not open Table1 exclusively-Try
            again?','Open
            Error',MB_OKCANCEL+MB_DEFBUTTON1)<>IDOK then
                raise;
    end;
end;
```

11.1.1.2 Table 组件的主要方法

1. 数据库表的打开和关闭

(1) Open 方法: 用于打开一个数据库,并把数据库设置成为编辑方式。

(2) Close 方法: 用于关闭数据表,数据表组件的 Active 属性设置为 False。

2. 浏览数据库中的数据

(1) First 方法: 调用 First 方法将记录指针移动至数据库表的第一条记录处, 并使之成为当前记录, 同时将 Bof 属性值设置为 True。

(2) Last 方法: Last 方法与 First 方法正好相反, 它将记录指针移动至数据库表的最后一条记录处, 并使之成为当前记录。它是将 Eof 属性值设置为 True。

(3) MoveBy 方法: 调用 MoveBy 方法将记录指针指向数据库表中与当前记录相关的一条记录处, 它的格式如下:

```
function MoveBy(Distance:Integer):Integer;
```

其中距离 (Distance) 是指需要移动的记录数目。距离是正值表明在数据库表中向前移动, 负值表明在数据库表中向后移动。例如, 在数据库表中往回移动 8 条记录表示为:

```
MoveBy (-8);
```

函数返回实际移动的记录数目, 一般情况下, 返回值等于需要移动的记录数目, 但在移动的记录超出了文件的开头或结尾的情况下, 返回值要小于需要移动的记录数目, 而等于移动至文件的开头或结尾所需要移动的记录数目。

(4) Next 方法: 调用 Next 方法将记录指针后移一条记录, 并使之成为当前记录。如果记录指针指向了数据库表的最后一条记录, 将设置 Eof 属性为 True。

(5) Prior 方法: 调用 Prior 方法将记录指针前移一条记录, 并使之成为当前记录。如果记录指针指向了数据库表的第一条记录, 将设置 Bof 属性为 True。

下面是一个经常会用到的用于浏览数据所有记录的程序代码:

```
Table1.DisableControls; //断开 Table1 与其他数据控制组件的联系
Table1.First;           //将当前记录指针移到表中的第一条记录
While not Table1.EOF do //遍历表中的记录, 直到最后一条记录
begin
    ...对当前记录的处理... //对当前的记录进行处理
    Table1.Next;             //当前记录指针下移一条记录
end;
Table1.EnableControls; //恢复 Table1 与其他数据控制组件的联系
```

3. 编辑和修改数据库中的数据

(1) Append 方法: 调用 Append 方法将在数据库表的末尾添加一个新的空记录并将这个新的记录设置为当前记录, 记录指针指向此记录。在调用 Append 方法后, 应用程序允许用户向记录的字段中输入数据, 并且能够通过调用 Post 方法将这些更改提交给数据库。例如:

```
procedure TForm1.Button1Click(Sender:TObject)
begin
    Table1.Append;
    Table1.FieldValues('姓名'):=Edit1.text;
    Table1.FieldValues('年龄'):=StrToInt(Edit2.text);
    Table1.Post;
end;
```

(2) Delete 方法: 调用 Delete 方法用来将当前记录从数据库中删除。如果该数据库表处于激活状态, Delete 方法将产生一个异常事件。

(3) Edit 方法: 调用 Edit 方法用来编辑数据库表中的当前记录, Edit 方法决定了数据

库表的当前状态。

(4) Insert 方法: 调用 Insert 方法将在数据库表中插入一个新的空记录并将新记录作为当前记录。

在调用 Insert 方法之后, 用户向记录的字段中输入数据, 通过调用 Post 方法将输入的数据写回数据库。

(5) InsertRecord 方法: 调用 InsertRecord 方法用来在数据库表中建立一个新的记录, 它与 Insert 方法类似, 只不过 InsertRecord 方法直接将新记录的字段值以参数形式传给数据库, 且不需要调用 Post 方法。例如:

```
Table1.InsertRecord([Edit1.Text, Edit2.Text, Edit3.Text, Edit4.Text]);
```

如果插入记录中字段的数目小于 Table1 中的字段数目, InsertRecord 方法会将缺少的字段值处理成 Null。

(6) Post 方法: Post 方法用来向数据库提交一个修改的记录。在编辑修改数据库中的数据后, 例如调用了 Edit, Insert 或 Append 等方法, 需要调用 Post 方法将修改的数据写回数据库。例如:

```
procedure TForm1.Button1Click(Sender:TObject)
begin
    Table1.Edit;
    Table1.FieldName('年龄').AsString:='30';
    Table1.Post;
end;
```

(7) SetFields 方法: SetFields 方法以参数的形式来修改当前记录中多个字段值。其基本格式如下:

```
Procedure SetFields(const Values:arrayofconst);
```

Values 中包含的是将插入每个字段的值, 插入的顺序是根据表中字段的先后顺序决定的。这些值可以是常量、变量或 0, 还可以是空指针。如果 Values 中所含的字段值数目比记录中的字段数目少, 缺少的字段会作为 NULL 处理。

在调用 SetFields 方法之前, 必须先调用 Edit 方法将数据库表置于 dsEdit 状态, 在调用 SetFields 方法后, 需要调用 Post 方法将更改提交给数据库。如果只是想更改记录中的部分字段而保持其他值不变, 给那些不需要更改的字段赋空指针。

(8) Cancel 方法: 调用 Cancel 方法用来取消对当前记录中一个或多个字段的修改。在这些更改还未提交给数据库时, Cancel 方法使记录恢复到修改前的状态, 并将数据库表置于 dsBrowse 状态。典型的 Cancel 方法是针对用户的需要取消修改, 或者在使字段生效的格式中取消非法的字段值。例如下面的代码说明该方法的使用。

```
if MessageDlg('Update Record?', mtConfirmation, [mbYes, mbNo], 0) = mrYes
then
    Table1.Post ;           //若为“是”则提交
else
    Table1.Cancel;          //若为“否”则撤消
```

4. 新表的创建和删除

(1) CreateTable 方法: 该方法用来建立一个使用新的结构信息的表。例如下面的代码说明 CreateTable 方法的使用。

```

if not Table1.Exists then           //避免覆盖已存表
begin
  with Table1 do
  begin
    Active:=False;                 //设置前表组件必须被关闭
    DatabaseName:='DBDEMOS';       //确定表的数据库
    TableType:=ttParadox;          //确定表的类型
    TableName:='CustInfo';         //确定表的名称
  with FieldDefs do
  begin
    Clear;
    Add('字段 1',ftInteger,0,True);
    Add('字段 2',ftString,30,False);
  end;
  with IndexDefs do               //确定索引
  begin
    Clear;
    Add('','字段 1',[ixPrimary,ixUnique]);
    Add('Fld2Indx','字段 2',[ixCaseInsensitive]);
  end;
  CreateTable;
end;
end;

```

(2) Delete Table 方法: 该方法删除一个已存在的数据库表。下面的程序代码说明了如何删除一个数据库表。

```

with Table1 do
begin
  Active:=False;                 //设置为非活动状态
  DatabaseName:='DBDEMOS';
  TableName:='pastor';
  TableType:=ttParadox;
  DeleteTable;
end;

```

5. 书签 (BookMark) 的使用

使用标签可以在数据表中快速地定位记录指针。在应用程序中我们有时需要保存当前记录指针所在的位置,以便以后能快速地返回到这个位置,此时就要用到书签。Delphi 7 提供了三个方法用于操作书签,他们是:

- GetBookMark 方法: 用于设置书签,返回一个 TBookMark 类型的值,该值为当前记录的指针。

- GotoBookMark 方法: 用于快速地将记录指针定位到具有书签的记录处。

- FreeBookMark: 用于释放书签。

这三个方法一般都是配套使用。下面的代码是使用书签的一般方法:

```

var
  BookMark:TBookMark; //定义书签
begin
  ...

```

```

Bookmark:=Table1.GetBookmark;           //在当前记录处放置一书签
Table1.DisableControls;
Table1.First;
While not EOF do                          //浏览数据表中所有记录
begin
    .....                                //对记录的操作
    Table1.Next;                          //向后移动记录指针
end;
Table1.GotoBookmark(BookMark);           //回到书签设置处
Table1.EnableControls;
Table1.FreeBookmark(BookMark);           //释放书签
end;

```

6. 设定数据库表的使用范围

在实际应用中数据库表里可能包含着大量的记录，而我们只需对其中一部分记录进行操作处理，此时为应用程序指定在数据库表中的一个使用范围就显得特别重要了，Table 组件中提供了下列方法来设定数据库表的使用范围。

(1) SetRangeStart 和 SetRangeEnd 方法：SetRangeStart 方法用于指定检索范围的起始记录。SetRangeEnd 方法用于指定检索范围的结束记录。

(2) EditRangeStart 和 EditRangeEnd 方法：EditRangeStart 方法用于修改检索范围的起始记录。EditRangeEnd 方法用于修改检索范围的结束记录。

(3) SetRange 方法：SetRange 方法包含了 SetRangeStart 和 SetRangeEnd 方法的功能，它可以同时指定检索范围的起始和结束记录，其基本形式为：

```
SetRange([起始值],[结束值])
```

(4) ApplyRange 方法：该方法是设定的检索范围有效。

(5) CancelRange 方法：该方法用于取消为表设定的检索范围，可以是对表的操作也可以是面向所有的记录。

7. 查询数据库中的记录

在数据库中查询符合某些条件的数据，是对数据库的一种重要操作。查询的数据库字段限定为表中的关键字段或辅助索引。

(1) SetKey 方法：该方法将 Table 组件置于查询状态。例如下面的代码，说明该方法的使用。

```

with Table1 do
begin
    SetKey;                                //置于查询状态
    FieldByName('State').AsString:='CA';
    FieldByName('City').AsString:='Santa Barbara'; //设置属性
    GotoKey;
end;

```

(2) FieldByName 方法：该方法是根据特定的字段名查找字段。

(3) GotoKey 方法：使用 GotoKey 方法查询数据表之前必须首先调用 SetKey 方法，使数据表进入查询状态。查询的过程如下：

- 确保待查找的字段是关键字或者已经在该字段上记录了索引，并且正确设置 Table

组件的 `IndexName` 属性。

- 调用 `TTable` 类的 `SetKey` 方法，把数据库表切换到查询状态。
- 设置被查找字段的值。
- 调用 `TTable` 类的 `GotoKey` 方法。
- 如果找到了匹配记录，`GotoKey` 方法返回 `True`，并且数据库表的指针指向该记录。

否则，`GotoKey` 方法返回 `False`，指针不变。下面就是一个使用 `GotoKey` 方法的实例。

```

Procedure TForm1.Button1Click(Sender:TObject)
begin
  with Table1 do
  begin
    SetKey;
    FieldByName('姓名').AsString:=Edit1.Text;
    //设置查询的姓名字段等于 Edit1 组件的值

    if GotoKey then
    ...                               //查找到后的执行代码
    else
    ...                               //没有找到的执行代码
  end;
end;

```

(4) `FindKey` 方法：该方法提供了更为简便的查找方法。它们将 `SetKey`、指定查找值、执行查找三个步骤融合在一步里完成。`FindKey` 方法的参数用于指明需要查找的值，可以有多个参数，它们之间用逗号分隔，每个参数对应一个查找字段。如果要查找数据表中的多个字段，必须把查找的字段名赋给 `Table` 组件的 **IndexFieldName** 属性，并用逗号分开各个字段的名称，然后把查找值赋给 `FindKey`。如下程序代码就是该方法的例子。

```

Procedure TForm1.Button1Click(Sender:TObject)
begin
  with Table1 do
  begin
    IndexFieldName:='姓名';
    //设置索引字段为姓名字段

    if FindKey([Edit1.Text]) then
    ...                               //查找到后的执行代码
    else
    ...                               //没有找到的执行代码
  end;
end;

```

(5) `Locate` 方法：如果要查询的字段没有建立索引，就不能使用前面的方法。`Locate` 方法弥补了这个缺陷，适用于所有情况。如果被查询的字段有索引，`Locate` 方法将自动使用索引（查询速度更快），否则，将进行普通的查询。

`Locate` 方法带有 3 个参数，函数原型如下：

```

Function Locate(Const KeyFields:string;Const KeyValues:Variant;
Options:TlocateOptions):Boolean;

```

其中 `KeyFields` 是用分号隔开的查询字段名称，`KeyValues` 是要查询的字段值，参数 `Options` 是查询选项，如果 `Options` 中包括 `loCaseInsensitive`，则查询时不区分 `KeyValues` 中的大小写；如果 `Options` 中包括 `loPartialKey` 时，可以实现近似查询。


8. Table 组件的主要事件

OnNewRecord 事件: 在应用程序插入一条记录之时, 触发事件。其他常用事件如表 11-2 所示。

表 11-2 Table 组件的常用事件及说明

事 件	说 明
BeforeOpen/AfterOpen	在数据库表组件被打开之前/之后被触发
BeforeOpen/AfterOpen	在数据库表组件被关闭之前/之后被触发
BeforeInsert/AfterInsert	在数据库表组件插入状态之前/之后被触发
BeforeEdit/AfterEdit	在数据库表组件被编辑之前/之后被触发
BeforePost/AfterPost	在数据库表组件发送大量修改的记录之前/之后被触发
BeforeCancel/AfterCancel	在数据库表组件取消前一步操作之前/之后被触发

11.1.2 Query (查询) 组件

Query 组件 ，位于【BDE】选项卡上，使用 SQL 语句通过 BDE 从一个物理数据库表文件中取出数据，并通过 DataSource 组件把数据提供给一个或多个数据控制组件；另一方面，用 SQL 语句把来自数据控制组件的数据通过 BDE 发送到物理数据库。

Query 组件也是一个数据集组件，它封装了结构化查询语句 (SQL)。应用程序通过组件的查询语句从本地或远程数据库的一个或多个数据库表中获取、插入、删除和更新数据。

1. Query 组件和 Table 组件区别

(1) Query 组件主要功能是用来支持 SQL 语言访问本地或远程数据库，Query 组件提供了一些与 Table 组件不同的特殊的属性、方法和事件。下面将要介绍。

(2) Query 组件允许用户同时访问多个表，而 Table 组件一次只能访问一个表。

(3) Query 组件访问的是表中的一个子集，即符合查询条件的特定数据；而 Table 组件只有提供过滤或限定检索范围才能访问表中的特定数据内容，否则，Table 组件访问的将是表中的全部数据。

(4) Query 组件的重要性体现在只有使用 SQL 语言才能完成查询的时候必须使用该组件，它支持复杂的嵌套查询。

(5) Query 组件和 Table 组件以不同的方式与 SQL 服务器进行交互，在执行数据定义语句时，应当使用 Query 组件，而在以非集中方式访问数据库时应使用 Table 组件。

2. Query 组件的主要属性

Query 组件与 Table 组件在许多属性上是相同的，下面仅仅介绍 Query 组件特殊的属性。

(1) SQL 属性: 字符串类型，它包含了需要进行查询 SQL 语句。

在 Delphi 应用程序中编写和使用的 SQL 语句有两种: 静态 SQL 语句和动态 SQL 语句。静态 SQL 语句是在程序设计阶段，将 SQL 语句作为 Query 组件的 SQL 属性值设置，可以单击对象查看器中的 SQL 属性栏激活【String List Editor】窗口，然后在窗口中编辑 SQL 语句，如图 11-1 所示。而动态 SQL 语句编程则是 SQL 语句中包含的一系列参数，在程序运行过程中各参数值是可变的，即可以动态地给 SQL 语句中的参数赋值。

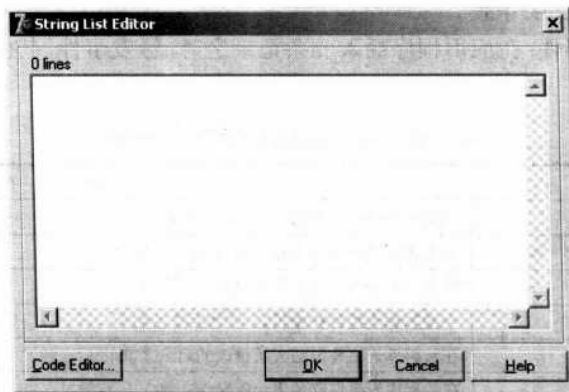


图 11-1 SQL 语句编辑器

SQL 属性中的 SQL 语句的编写也有两种方法:一种方法是在程序设计阶段便将相应的 SQL 语句写入到 Query 的 SQL 属性中;另一种方法是在开发应用程序时将 SQL 语句包含在程序代码中。

(2) Params 和 ParamCount 属性: TParams 类型, Params 属性包含了 SQL 语句需要使用的参数。

在运行中可以访问 Params 来动态地观察和设置参数名、参数值和数据类型,在设计阶段可以通过编辑 Params 属性来设置参数的形式。

检查 ParamCount 的值可以知道 Params 属性中有多少参数。如果 ParamCheck 属性设置为 True,则 ParamCount 与 SQL 查询语句中的实际参数数目对应。应用程序对 Params 属性的操作会自动地反映在 ParamCount 中。

当为 Query 组件编写动态 SQL 语句时,Delphi 会自动地建立一个数组 Params,数组 Params 是以 0 下标开始的,依次对应动态 SQL 语句中的参数,也就是说动态 SQL 语句中第一个参数对应 Params[0],第二个参数对应 Params[1],依此类推。

例如:一个 Query 组件 Query1,我们为它编写的动态 SQL 语句是:

```
Insert Into Customer(CustNo, Name, Country) Values(: CustNo, : Name, : Country)
```

对于上述这条动态 SQL 语句中的参数,我们可以利用 Query 组件的 Params 属性为参数赋值:

```
Query1.Params[0].AsString:="1988";
Query1.Params[1].AsString:="Lichtenstein";
Query1.Params[2].AsString:="USA";
```

上述语句将把“1988”赋给参数: CustNo,“Lichtenstein”赋给参数: Name,“USA”赋给参数: Country。

(3) DataSource 属性:该属性为动态 SQL 语句中尚存在没有赋值的参数时,Delphi 会自动检查 Query 组件的 DataSource 属性,如果为 DataSource 属性设置了属性值(该属性的值是另一个 DataSource 组件的名字),Delphi 会把没有赋值的参数与 DataSource 组件中的各字段比较,Delphi 会将相应的字段值赋给与其相匹配的参数,实现所谓的连接查询。

3. Query 组件的主要方法

(1) ExecSQL 方法: 调用 ExecSQL 方法来执行当前 SQL 属性中的 SQL 查询语句。ExecSQL 用于不需要返回记录指针的数据访问, 例如执行 INSERT、UPDATE、DELETE 和 CREATE TABLE 语句。而对于 SELECT 语句, 则应调用 Open 方法。如果 ExecSQL 执行的查询没有完成准备工作, 它会先完成准备工作。所以如果想要提高执行速度, 应在第一次调用 ExecSQL 方法前调用 Prepare 方法。

(2) ParamByName 方法: 调用 ParamByName 方法根据参数名来设置和使用参数信息。ParamByName 方法主要用来在运行时设置参数值。该方法在 SELECT 语句中参数值不能为空, 但在 UPDATE 和 INSERT 语句中可以。

ParamByName 是一个函数, 用动态语句中的参数作为调用 ParamByName 函数的参数, 这样便可以为它们赋值, 使用这种赋值方法, 必须要知道动态 SQL 语句参数的名字。

例如可以用下述方法给参数赋值:

```
Query1.ParamByName('CustNo').AsString:="1988";
Query1.ParamByName('Name').AsString:="Lichtenstein";
Query1.ParamByName('Country').AsString:="USA";
```


使用这种方法同样可以为各参数赋值, 而且更加直观一些。

(3) Prepare 方法: 调用 Prepare 方法让 BDE 和远程的数据库服务器为查询分配资源并执行一些优化操作, 称为执行查询前的准备工作。在执行查询前调用 Prepare 方法会提高应用程序的执行效率。在第一次执行查询前, 会自动地完成准备工作。

(4) UnPrepare 方法: 执行查询的准备工作会消耗一定的资源, 所以在完成查询以后应该调用 UnPrepare 方法来释放分配给查询的资源。

因为 Query 组件的事件非常少, 所以在这里就不介绍了。Query 组件的其他属性和方法可参考 Table 组件。

11.1.3 StoredProc (存储过程) 组件

StoredProc 组件 , 位于【BDE】选项卡上, 与 Query 组件、Table 组件一样, 属于数据集组件。StoredProc 组件是用来完成数据库服务器上的存储过程。当一个客户的应用程序必须在一个远程服务器的数据库中进行存储操作时, 就会用到 StoredProc 组件。一个存储过程是一系列作为服务器的一部分存储的表达式, 存储过程执行与数据库相关的重复性任务, 并将结果传递给用户。

StoredProc 组件的主要属性和方法:

1. Overload 属性

该属性用来说明在一个 Oracle 服务器上执行哪一种重载的存储过程。一个 Oracle 重载的存储过程是和其他的存储过程同名的。Oracle 服务器通过给每个过程一个特有的数码标识来区别重载的存储过程。应用程序可以通过 Overload 属性来指明数码标识。默认时, Overload 属性值为 0, 表明没有存储过程的重载。如果 Overload 属性值为 1, BDE 将执行第一种重载过程, 依此类推。

注意: Oracle 的重载过程是同名的, 但它们的参数列表却各不相同, 应用程序必须确保提供的参数列表与重载过程相符。

2. StoredProcName 属性

该属性用来指明服务器调用的存储过程名。如果 StoredProcName 属性中指明的存储过程名与服务器上存在的存储过程名不相符,则当应用程序为存储过程做准备时,会产生一个异常事件。

3. Create 方法

该方法用来初始化一个已声明的存储过程。Create 方法调用它派生的 Createconstructor,为该存储过程建立一个新的空参数列表,并将其参数、服务器和记录缓冲区都清空。

4. ExecProc 方法

该方法用来执行服务器上的存储过程。

在调用 ExecProc 方法之前,必须进行如下操作,才可调用 ExecProc 方法。

(1) 在 Params 属性中给出所有存储过程需要的参数。在设计阶段,可以通过使用 Parameters Editor 对话框给出参数,如图 11-2 所示。在运行时,应用程序需要直接访问 Params 属性。

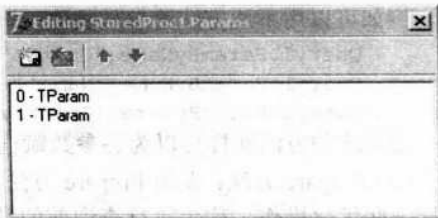


图 11-2 Parameters Editor 对话框

(2) 调用 Prepare 方法联系各参数。当 ExecProc 将控制权重新交给应用程序时,如果存储过程返回了输出参数,它们将被存储在


Params 属性中。通过 Params 属性的参数列表索引,或者通过调用 ParamByName 方法,应用程序可以访问这些输出参数。

5. GetResults 方法

调用 GetResults 方法使一个 Sybase 存储过程返回给客户结果。对其他服务器,存储过程的结果会自动返回给客户;而对于 Sybase 存储过程,直到记录指针指向数据集的末尾它才会返回结果。GetResults 方法正是使记录指针指向数据集的末尾。

StoredProc 组件的其他属性和方法可参考 Table 组件和 Query 组件。

11.1.4 DataSource (数据源) 组件

DataSource 组件 , 位于【Data Access】选项卡上,用做其他数据读写组件,如 Table、Query、TstoredProc 等数据集组件,与可视化数据组件,如 DBGrid 之间的连通道。

它主要为数据控制组件提供服务,数据控制组件通过数据源组件可以从数据集中取得数据,数据在数据控制组件中显示,用户在数据控制组件中对数据进行操作。

DataSource 组件的主要属性、方法和事件有:

1. AutoEdit 属性

该属性值是一个布尔类型,用于说明是否将与 DataSource 组件相连的数据集置于编辑状态,允许用户通过数据控制组件编辑数据集中的数据。AutoEdit 属性的默认值为 True,用户想要改变在当前数据控制组件中显示的数据时,应用程序就会自动地调用 Edit 方法允许用户对当前数据进行编辑。如果该属性的值设置为 False,可以防止用户对当前数据的无意更改。

2. DataSet 属性

该属性用来指明与当前数据源组件相联系的数据集组件对象的名字。可以在设计阶段通过对象观察器来设置 DataSet 属性,也可以在运行时设置该属性。应用程序通过在运行时改变 DataSet 属性可以实现在同一个数据控制组件中显示和编辑不同数据集组件中的数据。

3. Enabled 属性

该属性决定了与此数据源组件相连的数据控制组件是否显示数据。如果 Enabled 属性值为默认值 True,数据控制组件将显示数据;如果 Enabled 属性值设置为 False,则所有与此数据源组件相连的数据控制组件都将不显示任何数据。

4. Edit 方法

调用 Edit 方法允许用户对与该数据源组件相连的数据集进行编辑修改。Edit 方法首先检查 AutoEdit 的属性值是否为 True,State 属性是否表示数据集组件处于 dsBrowse 状态。然后,再调用数据集的 Edit 方法。再调用数据集的 Edit 方法检查这些属性值,目的是为了 保证数据集组件处于支持编辑的状态,而数据源组件也同样支持编辑。

5. OnDataChange 事件

当与该数据源组件相连的数据集提交被编辑的当前记录时触发该事件。在数据控制组件中,对 DataSource 组件的 OnDataChange 事件会作出相应的反应:转向一条新的记录或者转向新的字段。


6. OnStateChange 事件

该事件在与该数据源组件相连的数据集的状态发生改变时被触发。OnStateChange 事件的处理程序可以对这种状态的改变作出相应的操作,例如使某些菜单项或按钮生效或失效。

7. OnUpdateData 事件

该事件在向数据库提交对数据的改变之前被触发。OnUpdateData 事件处理程序的主要作用是执行附加的数据处理和确认操作。

11.1.5 Session (会话) 组件

Session 组件 , 位于【BDE】选项卡上,在一个数据库应用中,提供对于一组数据库连接的全局管理。当生成一个数据库应用项目时,它将自动包含一个 Session 组件。在随后往项目中添加数据库和数据集组件时,它们自动与这个默认的 Session 关联。Session 还控制对于有密码保护的 Paradox 文件的读写。通过在使用 Session 组件的属性、事件和方法,可以控制数据库连接与访问。可以在一个应用中用一个默认 Session 控制所有的数据库连接,也可以用附加的 Session 组件来控制一个数据库子集的连接。

下面介绍 Session 组件的属性和方法:

1. DatabaseCount 属性

通过检查 DatabaseCount 属性值来获悉与此 Session 组件相连的活动数据库的数目。当与数据库的联系打开或中断时,DatabaseCount 属性值会发生改变。DatabaseCount 属性通常是和 Database 属性一起使用来遍历与此 Session 组件相连的所有处于活动状态的数据库组件。

2. Database 属性

通过使用 Database 属性来访问与此 Session 组件相连的处于活动状态的数据库组件，即当前与数据库服务器相连的数据库组件。

3. KeepConnections 属性

该属性用来说明在没有任何活动的数据集与数据库组件相连时，该数据库组件是否与数据库服务器保持联系。

4. GetAliasNames 方法

该方法是用来获取一个已存在的 BDE 数据库别名的列表。

5. GetAliasParams 方法

该方法是用来获得与一个 BDE 数据库别名有关的参数信息。

6. GetDatabaseNames 方法

该方法是用来获得 BDE 数据库别名和所有 Database 组件的名字。

7. GetDriverNames 方法

该方法是用来获得此 Session 组件可以使用的所有 BDE 驱动器。


8. GetDriverParams 方法

该方法是用来获取一个指定的 BDE 驱动器的参数信息。

9. GetTableNames 方法

该方法是用来获得与数据库组件相连的所有表的名称。

11.1.6 DataBase (数据库) 组件

DataBase 组件  位于【BDE】选项卡上，连接到单一数据库上，为应用程序提供了对数据库的控制。在以下几种情况下用到 DataBase 组件：需要与数据库的持续连接；使用了定制的数据库服务；使用事务处理组件；程序中确定了 BDE 别名。DataBase 组件主要用于 SQL 数据库的应用中。

当应用程序连接到一个远程的 SQL 数据库服务器上并要求控制与 BDE 相关事件的处理时，Database 组件显得非常重要。

DataBase 组件的主要属性、方法和事件如下：

1. AliasName 属性

用来指定一个已经存在的 BDE 数据库别名。BDE 别名里包含了连接需要的数据库的特征信息。这些由 BDE 数据库别名提供的信息根据不同的数据库类型而不同。

- 如果应用程序要设置 AliasName 属性，所有原先赋给 DriverName 的属性值都会给清空。因为数据库别名将自动地指明驱动器名作为它的一个参数。其他数据库别名提供的信息将会在设置 AliasName 属性时存储在 Params 属性中。

- 如果需要的 BDE 数据库别名不存在，应用程序可以通过指明数据库的 DatabaseName、DriverName 以及 Params 属性来代替设置 AliasName 属性。

- 如果在 Connected 属性值为 True 的时候设置 AliasName 属性会导致一个异常事件

的产生。设置 AliasName 属性可以通过在设计阶段双击 Database 组件激活数据库编辑对话框来完成, 如图 11-3 所示。

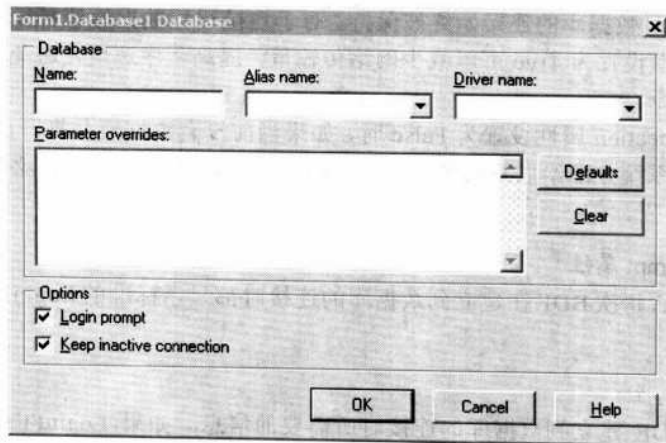


图 11-3 Database 对话框

2. Connected 属性

该属性用来确定数据库连接是否可用。当 Connected 属性设置为 True 时, 不打开数据库而建立连接。Connected 属性设置为 False 时, 关闭连接。

3. DatabaseName 属性

该属性用来指定与数据库组件相联系的数据库名。如果 DatabaseName 属性值与一个已经存在的 BDE 数据库别名相同, 则 AliasName 和 DriverName 属性都不用再设置。如果 DatabaseName 属性值与已存在的 BDE 数据库别名不匹配, 那么应用程序除了给出 DatabaseName 属性值, 还必须在 AliasName 里提供一个有效的数据库别名; 或者提供 DriverName 和 Paramsproperties 属性值。DatabaseName 在连接到一个 Paradox 或 dBASE 数据库时可以是一个完全确定的路径名。

同 AliasName 属性一样, 如果在 Connected 属性值为 True 时设置 DatabaseName 属性会导致一个异常事件的产生。设置 DatabaseName 属性可以通过在设计阶段双击 Database 组件激活 DatabaseEditor 对话框来完成。

4. DataSets 属性

应用程序可以通过使用 DataSets 属性来访问与该 Database 组件相连的所有处于打开状态的数据集。

5. DriverName 属性

该属性用来指明 BDE 驱动器名。DriverName 必须是一个有效的 BDE 驱动器名。

注意: 如果应用程序设置了 DriverName 属性, 它必须同时指明 Params 属性中的联系参数。通常, 在 AliasName 属性中指明的数据库别名提供了联系中的参数值; 但在设置了 DriverName 属性后, AliasName 属性值会被自动清除。

6. KeepConnection 属性

用来决定程序是否与数据库保持连接,即使数据库没有打开。当 KeepConnection 属性设置为 True 时,与数据库的连接始终被保持。对于连接远程数据库或频繁打开关闭数据库的情况,将该属性设置为 True 可以减少网络传输量,提高程序速度,避免每次建立连接时输入用户名和口令。

当 KeepConnection 属性设置为 False 时,如果当前没有数据集是打开的,联系会中断。中断联系会释放系统分配给联系的资源,但在以后一个数据库打开时,必须重新建立和初始化联系。

7. LoginPrompt 属性

用来指明是否每次 BDE 在建立到数据库的连接时都显示标准的 Login 对话框以要求输入用户名和口令。

8. Params 属性

用来存放 BDE 建立到数据库的连接时所需要的信息。如果 LoginPrompt 属性设置为 False 时,在建立到数据库的连接时将不显示 Login 对话框,BDE 将从 Params 属性中查找用户名和口令。

9. TransIsolation 属性

该属性用来说明 BDE 控制 Database 事务的独立等级。事务的独立等级决定了对相同数据表操作时,一个事务与其他事务之间的相互作用。该属性的取值如表 11-3 所示。

表 11-3 TransIsolation 属性取值以及含义

取 值	含 义
tiDirtyRead	在其他同时的事务对数据库的修改还未提交时,允许读入这些修改。未提交的修改不是持久性的,它可能在任意时候恢复原来的状况。在这个等级,与其他事务的相互独立程度最低
tiReadCommitted	在其他同时的事务对数据库的修改提交后,允许读入这些持久性的修改。此属性值为默认值
tiRepeatableRead	在同一时间只允许单个事务读取数据库。在这个等级,一个事务与其他事务完全独立。它的独立程序最高

不同的数据库服务器支持不同的独立等级。如果应用程序设置的 TransIsolation 属性是一个不被远程 SQL 服务器支持的等级,BDE 会使用下一级的独立等级。

10. ApplyUpdates 方法

调用 ApplyUpdates 方法向数据库服务器提交当前数据集的缓存更新。ApplyUpdates 方法只有在数据集的 CachedUpdates 属性值为 True 时才有意义。

DataSets 是提交更新的数据集的名字的列表,DataSets 不需要列出每一个当前处于打开状态的数据集;而对于列出的数据集,ApplyUpdates 方法会调用数据集的 ApplyUpdates 和 CommitUpdates 方法来向数据库服务器提交当前的缓存更新。

11. Close 方法

调用 Close 方法来中断与数据库服务器的联系,并释放系统分配给联系的资源。注意当 Database 组件的 KeepConnection 属性值为 False 而当前没有打开状态的数据集时,Close 方法会被自动地调用。

12. Commit 方法

该方法永久地保存当前事务对数据的更新、插入或删除的操作，并结束当前的事务。当前的事务是指最后一次调用 StartTransaction 开始的事务。

13. Rollback 方法

该方法用来撤消当前事务对数据的更新、插入或删除的操作，并结束当前的事务。

14. StartTransaction 方法

该方法用来在数据库服务器上开始一个新的事务。

15. OnLogin 事件

该事件在应用程序与数据库建立联系的时候触发。OnLogin 事件的处理程序默认时，数据库要求用户登录，当前的 USERNAME 从 Params 属性中得到，并打开标准的登录对话框。

11.1.7 Field (字段) 组件

在 Delphi7 中有一个特殊的组件，该组件在组件面板上找不到，但可在应用程序的声明语句中找到它的身影，这就是 Field 组件。该组件一般附属于某一个 DataSet 组件，由 DataSet 组件管理它的创建和撤消。每当 DataSet 组件从数据库文件中获得数据，就将其放入 Field 组件中。若要修改数据库文件，需要通过给 Field 组件赋值来实现。由于字段组件直接对应着数据库表中的字段，浏览和修改表中的数据必须通过字段组件，同时字段组件所拥有的属性可以用来说明数据库表中对应的字段的数据类型、当前的字段值、显示格式、编辑格式等原因，字段组件就显得相当重要。

设置 Field 组件的属性，首先应该打开 Fields Editor 对话框，选择某个组件，在对象查看器中会显示这个组件所有设计时可见属性，见图 11-4、图 11-5。

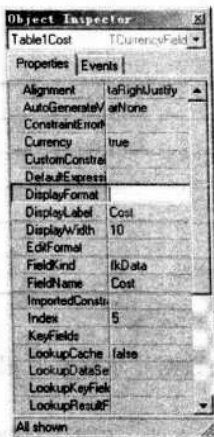


图 11-4 Field 组件的属性

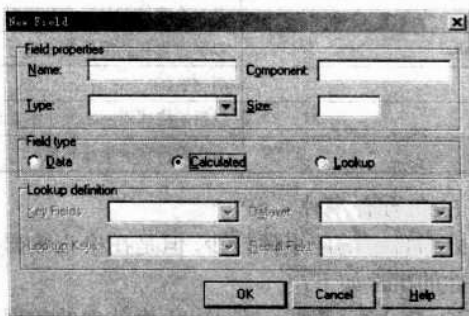


图 11-5 New Field 对话框

1. Field 组件主要属性

- Alignment 属性：用来设置数据在【DataControl】选项卡中组件的对齐方式。

- Calculated 属性：用来反映字段是否是计算字段。
- DisplayFormat 属性：用来设置数据在【DataControl】选项卡中组件的显示形式，如 Emp#0000，后面四个 0 表示四个数字，是该 Field 组件的真正数据，而前面的几个字符将自动加在数据前面显示，该属性只为数字型 Field 子组件具有。
- DisplayLabel 属性：用来在【DBGrid】组件上的相应字段显示的列标题。
- DisplayWidth 属性：用来在【DBGrid】组件上对应列的宽度，以字符个数为单位，StringField 字段的显示宽度为该字段的字符数，其他字段的默认值为 10。
- EditFormat 属性：用来确定数据在【DataControl】选项卡中组件中的编辑格式。
- FieldName 属性：Field 组件所对应域的域名。
- Index 属性：Field 组件在数据集组件中的索引号。用户可以改变它，以重新调整各个 Field 组件的顺序，但它不会改变 Field 组件在【DBGrid】组件列的位置。
- Lookup 属性：用来反映该字段是否是查找字段。
- MaxValue 和 MinValue 属性，用来设置该域的最小值和最大值，只用于实数域或整数域，当 MaxValue 为 0 时，表示没有最大值限制。
- ReadOnly 属性：用来确定该组件为只读，不能修改该 Field 组件所代表域的数据。当 DataSet 组件的 ReadOnly 属性被设置为 True 时，可以用它来使某些特定的域变成不可修改域。
- Required 属性：用来确定该域中至少有一个非零值，即不能为空。
- Size 属性：这个属性对不同类型的 Field 组件有不同含义，如 StringField 组件的 Size 属性表示最大字符个数，ByteField 组件则表示最大字节数等。
- Visible 属性：用来确定 Field 组件在【DBGrid】组件中是否可见。默认可见。

2. Field 组件的常用方法

Field 组件的常用方法如表 11-4 所示。

表 11-4 Field 组件的常用方法

方 法	含 义
Assign	从一个字段向另一个字段复制值
AssignValue	以字符形式为字段指定值
Clear	清空字段
GetData	以原始格式返回字段中的数值
SetData	为字段指定一个原始值

3. Field 组件事件

Field 组件可响应事件较少，但是在检查数据是否合法，或修改数据显示格式时很有用。Field 组件的事件如下：

- OnChange 事件：在 Field 组件中的数据将被改动时触发该事件，用来控制选单功能是否禁用的事件函数。与该 Field 组件相连的【DataControl】选项卡中的组件的值被改动后，并不一定会触发该事件，而要等到用 Post 方法发送数据时才会被触发。
- OnGetText 事件：当从 Field 组件相连的【DataControl】选项卡的组件的 Text 属性中获取数据时触发该事件，用在自定义数据输入格式中。

- OnSetText 事件: 当从 Field 组件相连的【DataControl】选项卡中的组件的 Text 属性中发送数据时触发该事件, 用在自定义数据输入格式中。

- OnValidate 事件: 在 Field 组件中的数据将被改动时触发该事件, 用来在字段数据写入数据库前的验证动作。

4. Field 组件的建立

有两种方式可以建立 Field 组件, 一种是由应用程序在打开某个数据库文件时自动建立, 在关闭数据库文件时自动取消这些组件。打开和关闭数据库文件是通过分别与数据库文件相连的 Table 组件的【Active】属性设置为 True 或 False 来实现的, 这种建立方式称为动态建立。以这种方式建立的 Field 组件在头文件中没有声明语句。另一种方式是通过【Fields Editor】(字段编辑器)来建立 Field 组件, 这种方式建立的 Field 组件将作为固定的组件, 在关闭文件时不会取消。

在窗体上双击 Table 组件, 或先右击 Table 组件, 然后选择弹出菜单中的【Fields Editor】命令(如图 11-6 所示), 都会弹出如图 11-7 所示的字段编辑器, 当第一次打开时, 在字段列表中没有任何内容。

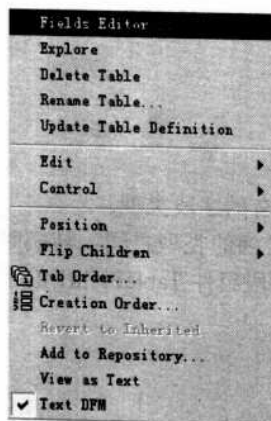


图 11-6 弹出菜单

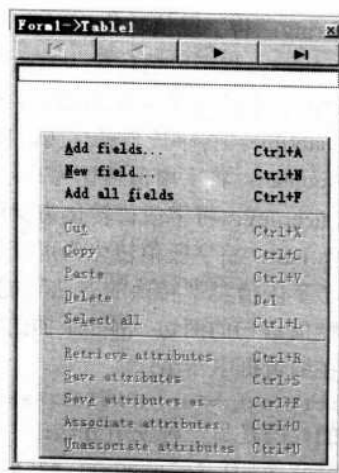


图 11-7 字段编辑器

用鼠标右击字段编辑器, 在快捷菜单中选择【Add all fields】命令, 就可以为 Table 组件所连接的那个数据库文件中的所有字段均建立一个 Field 组件, 并将在字段列表框中显示各个字段的字段名。如图 11-8 所示。

如果只需要选择其中的某几个字段, 而不是所有的字段, 则应该选择【Add fields】命令。

5. Field 组件的类型转换

字段组件有自己的数据类型, 有时需要在不同的类型之间进行转换, 字段组件提供了一些方法用于转换字段值的类型, 对于不同的字段类型, 这些方法的作用是不一样的, 这些方法包括:

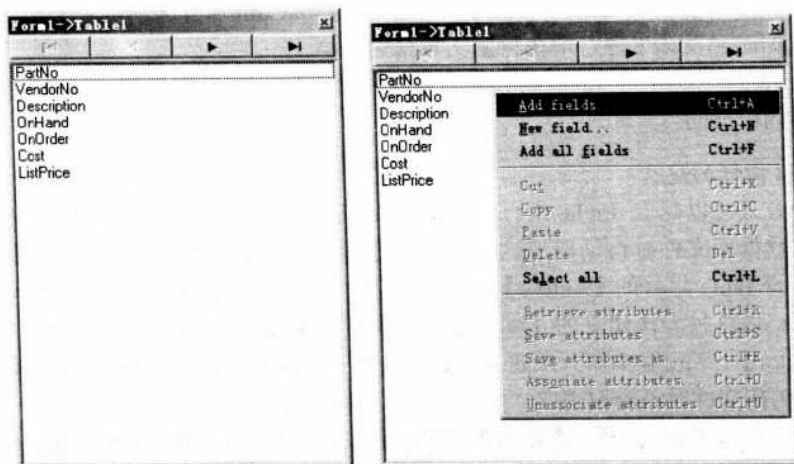


图 11-8 字段编辑器

- **AsString**: 将字段值转化为字符串类型。几乎适用于所有的字段类型。
- **AsInteger**: 将字段值转化为整数类型。
- **AsFloat**: 将字段值转化为实数类型。
- **AsDatetime**: 将字段值转化为时间日期类型。
- **AsBoolean**: 将字段值转化为布尔类型。
- **AsCurrency**: 将字段值转化为货币类型。
- **AsVariant**: 将字段值转化为可变类型，适用于所有的字段类型。

这些用于字段值类型转换的方法只能用于能够进行转换的类型之间。转换时这些方法可以出现在赋值语句的两边。例如下面的程序代码是将字段组件 **TableField1** 的字段值以字符串的形式在编辑框中输出:

```
Edit1.Text:=TableField1.AsString;
```

而下面的代码是另一种使用方法:

```
TableField1.AsInteger:=strtoint(Edit1.Text);
```

将编辑框中的字符串转化为整数类型赋值给 **TableField1** 字段。

6. Field 组件的存取

在 **Field** 组件中保存着当前记录中的数据, 要存取数据库文件中各个字段的数据必须通过它来实现。利用 **Field** 组件存取数据有如下三种手段:

- 利用数据集组件总的 **Fields** 属性。该属性是一个运行期间可见的属性, 是由数据集组件所管理的所有 **Fields** 组件组成的一个数组。如下面代码:

```
Table1.Fields[0].AsString:=Edit1.Text;
```

其中[]内的数字为该 **Field** 组件的索引号。

- 利用数据集组件的 **FieldByName** 方法。如下面代码:

```
Table1.FieldsByName("China").AsString:=Edit1.Text;
```

- 直接用各个 **Field** 组件的名称。如下面代码:

```
Table1.Name.AsString:=Edit1.Text;
```


11.2 数据控制组件


数据集组件使数据库应用程序与数据库之间建立了联系，同时数据库应用程序还需要将这种联系提供给用户，让用户也能浏览和运算数据库，这就是数据库应用程序的用户接口。数据控制组件主要用于显示和编辑数据库表中的数据，因而也称为数据浏览组件，它为编程人员设计用户接口提供了极大的方便。

数据控制组件集中在数据控制选项卡上，Delphi 7 中【Data Controls】选项卡及其包含的组件如图 11-9 所示。



图 11-9 Data Controls 选项卡上的组件

11.2.1 DBGrid 组件

DBGrid 组件 ，该组件提供了一个可视化的表格，以显示编辑数据库的表，是 Delphi 7 中功能强大的数据控制组件。DBGrid 组件显示从 DataSource 空间处获取的数据，可以直接在 DBGrid 组件上修改数据、增加记录（按 **Enter** 键即可在当前记录前插入一空白记录）、删除记录（**Ctrl+Del**）等。

DBGrid 组件与 DBEdit 组件不同，DBEdit 组件只能显示一个字段，而 DBGrid 组件以网格形式显示数据库表中的全部记录的所有字段。

1. DBGrid 组件的主要属性

(1) Column 属性：用来设置 DBGrid 组件的字段。它的属性如表 11-5 所示。

表 11-5 Column 属性以及含义

属 性	含 义
Alignment	指定该列内容的对齐方式，可设为 taLeftJustify（左对齐）、TaCenter（居中）、taRight Justify（右对齐）
ButtonStyle	设为 cbsAuto 表示加上一个组合框，设置为 cbsEllipsis 表示加上一个省略号按钮，当用户单击此按钮将触发 OnEditButtonClick，设置为 cbsNone 表示既没有组合框也没有省略号按钮
Color	设定背景颜色
DropDownRows	当使用下拉列表为列选内容时，这个属性决定了下拉列表下拉出的行数
Expanded	如果列对象所对应的字段是 ADT 或数组字段，这个属性用于控制是否展开
FieldName	指定这个列显示数据库表中哪个字段的内容
PickList	指定一个字符列表，当为这个列的单元格输入内容时，可以用下拉列表选择内容，这个下拉列表的内容就是这个属性的内容
Title	设定列的标题
ReadOnly	设置为 True，表示该列是只读的，不能编辑
Width	用于设置列的宽度，默认值是 Field 的 DisplayWidth
Font	用于设置该列中字符的宽度，默认值是 DBGrid 的 Font

(2) DataSource 属性：该属性用于指明与数据控制组件链接的数据源组件 DataSource，数据控制组件是从 DataSource 组件中获取数据的。

(3) Enabled 属性: 当 DBGrid 组件连接到数据集组件时, 它的 Enabled 属性决定了 DBGrid 组件能否接受来自鼠标、键盘和定时器事件的信息。

(4) Options 属性: 配置 DBGrid 如何显示数据。它的取值如表 11-6 所示。

表 11-6 Options 属性的取值及含义

取 值	含 义
dgEditing	运行用户在 DBGrid 中编辑数据, 如果选择了 dgRowSelect, 这个值将被忽略
dgAlwaysShowEditor	DBGrid 的表格常处于编辑状态, 也就是说, 鼠标直接点击单元格后就可以编辑数据, 不需要按 Enter 键才进入编辑状态
dgTitles	显示列的标题
dgIndicator	显示表示当前记录的标示符
dgColumnResize	决定在运行时列的宽度是否可以调节
dgColLines	在列之间显示表格线
dgRowLines	在行之间显示表格线
dgTabs	用户可以使用 Tab 键和 Shift+Tab 在单元格之间跳转
dgRowSelect	在运行时可以选择一整行
dgConfirmDelete	如果删除记录, 弹出对话框让用户确定
dgAlwaysShowSelection	显示选择提示
dgCancelOnExit	在退出程序时, DBGrid 取消新增但没有输入内容的记录
dgMultiSelect	已同时选择多行

(5) DragMode 属性: 该属性用来设置能否用鼠标改变各列在网格中的显示顺序和位置。该属性取值如表 11-7 所示。

表 11-7 DragMode 属性的取值及含义

取 值	含 义
dmManual	在应用程序运行过程中, 用户可以用鼠标拖放网格中的各列, 改变各列在网格中的显示顺序和位置。当用鼠标拖放网格中的一列、改变它在网格中的位置时, 只是改变了该列在数据集中的位置, 并没有改变它对应的数据库表中的位置
dmAutomatic	用户不能用鼠标拖放网格中的各列而改变它在网格中的位置

(6) DefaultDrawing 属性: 该属性的值是布尔型, 它用于控制网格中各网格单元的绘制方式。在默认情况下, 该属性的值设置为 True 时, 使用网格本身默认的方法绘制网格中各网格单元, 并填充各网格单元中的内容, 各网格单元中的数据根据其对应的字段组件的 DisplayFormat 属性和 EditFormat 属性进行显示和绘制。如果 DefaultDrawing 属性被设置为 False 时, 将不会自动地绘制网格中各网格单元和其中的数据, 用户必须自己为 DBGrid 组件的 OnDrawDataCell 事件编写相应的程序用于绘制各网格单元和其中的数据。

2. DBGrid 组件的主要事件

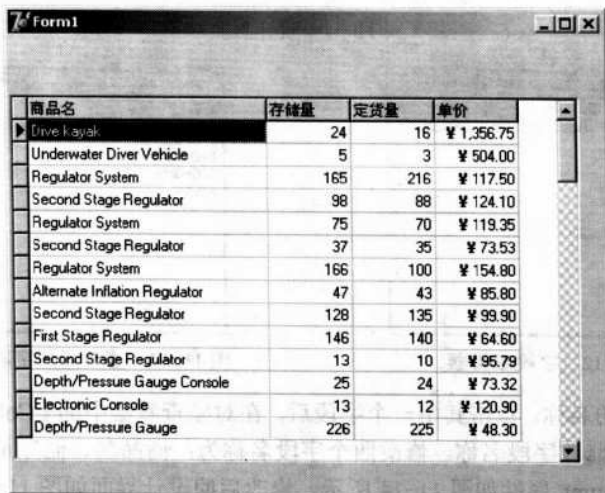
DBGrid 组件的主要事件如表 11-8 所示。

[例 11-1] DBGrid 的使用

效果: 利用 DBGrid 组件实现显示、编辑和修改数据库的表。执行效果如图 11-10 所示。

表 11-8 DBGrid 组件的主要事件

事 件	说 明
OnColEntor	当用户进入网格各列时, 触发该事件
OnColExit	当用户离开网格各列时, 触发该事件
OnDblClick	当用户在网格中双击鼠标左键时, 触发该事件
OnDragDrop	当用户在网格中用鼠标进行拖放操作时, 触发该事件
OnDragOver	当用户在网格中用鼠标拖动网格时, 触发该事件
OnDrawDataCell	用于定制绘制网格中各网格单元, 当向网格中填充数据时, 触发该事件
OnEndDrag	当用户停止拖动网格时, 触发该事件
OnEnter	当网格获得焦点时, 触发该事件
OnExit	当网格失去焦点时, 触发该事件
OnKeyDown	当用户在网格中按下任何键或组合键时, 触发该事件
OnKeyPress	当用户在网格中按了任何一个数字键或字母键时, 触发该事件
OnKeyUp	当用户在网格中释放任何被按下的键时, 触发该事件



商品名	存储量	定货量	单价
Drive Kayak	24	16	¥ 1,356.75
Underwater Diver Vehicle	5	3	¥ 504.00
Regulator System	165	216	¥ 117.50
Second Stage Regulator	98	88	¥ 124.10
Regulator System	75	70	¥ 119.35
Second Stage Regulator	37	35	¥ 73.53
Regulator System	166	100	¥ 154.80
Alternate Inflation Regulator	47	43	¥ 85.80
Second Stage Regulator	128	135	¥ 99.90
First Stage Regulator	146	140	¥ 64.60
Second Stage Regulator	13	10	¥ 95.79
Depth/Pressure Gauge Console	25	24	¥ 73.32
Electronic Console	13	12	¥ 120.90
Depth/Pressure Gauge	226	225	¥ 48.30

图 11-10 执行结果

(1) 在窗体上放置一个 Table1 组件, 设置它的属性【DataBaseName】属性为“DBDEMOS”, 【TableName】属性为“parts.db”, 把【Active】属性设置为“true”。选取一个 DataSource1 组件, 将其【DataSet】属性设置为“Table1”。拖动 DBGrid1 组件到窗体上, 设置它的【DataSource】属性为“DataSource1”。设计界面如图 11-11 所示。

(2) 上图中的 DBGrid 只是把 parts 表的内容全部显示出来, 我们可以修改 parts 表的字段。在 DBGrid 组件上右击, 在弹出的快捷菜单中选择【Columns Editor】命令, 出现字段编辑器, 在字段编辑器上右击, 选择【Add all fields】命令, parts 所有的字段被抓取到 DBGrid 中, 如图 11-12 所示。

(3) 修改 parts 表字段, 删除 PartNo、VendorNo、ListPrice 字段(选中字段后, 使用键盘上的 Delete 键), 保留 Description、OnHand、OnOrder、Cost 字段。如图 11-13 所示。

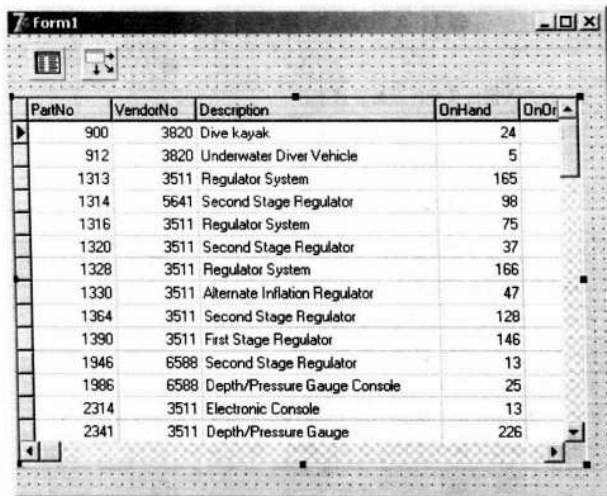


图 11-11 设计界面

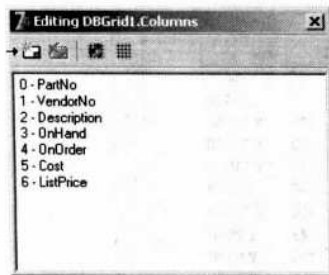


图 11-12 字段编辑器

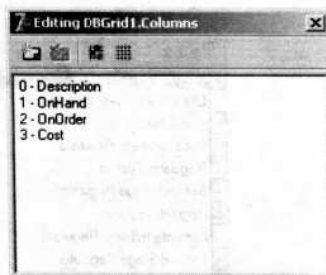


图 11-13 修改后的字段编辑器

(4) 修改字段的名称。点击其中一个字段后，在对象查看器中的【Column】属性的 Title 属性中的 Caption，修改字段名称。修改四个字段名称为：商品名、储存量、定货量和单价。对象查看器中的 Column 属性如图 11-14 所示。修改后的设计界面如图 11-15 所示。

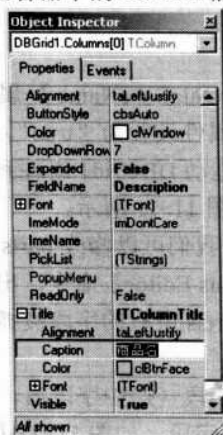


图 11-14 对象查看器中的 Column 属性

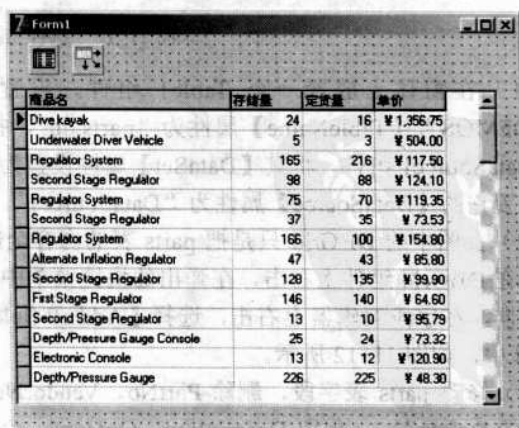



图 11-15 修改后的设计界面

(5) 运行程序，在窗体中显示该数据库中的所有数据。

11.2.2 DBNavigator 组件

DBNavigator 组件，用于在数据集中为用户导航，提供了一个工具条帮助浏览、修改、删除等数据库表的内容。工具条的外观如图 11-16 所示。

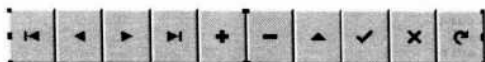


图 11-16 工具条

这个工具条上各个按钮的名称和作用从左至右如表 11-9 所示。

表 11-9 DBNavigator 组件的按钮

按 钮	作 用
First	把当前记录的指针指向第一条记录，相当于 First 方法
Prior	移动到上一条记录处，相当于 Prior 方法
Next	移动到下一条记录处，相当于 Next 方法
Last	移动到最后一条记录处，相当于 Last 方法
Insert	在当前记录之前插入一条记录，相当于 Insert 方法
Delete	删除当前记录，相当于 Delete 方法
Edit	进入编辑状态，可以对当前记录进行修改，相当于 Edit 方法
Post	把修改过的内容写入到数据库中，相当于 Post 方法
Cancel	撤销所作的修改，相当于 Cancel 方法
Refresh	刷新对应的 DataSet 的缓冲区，可以用于改变记录的显示格式

DBNavigator 组件的主要属性和事件：

(1) VisibleButtons 属性：用于设置需要工具条按钮。如果不需要按钮，双击该属性，将某些子按钮的属性，设置为 False，就可以将不需要的按钮去掉。默认情况下为 True。

(2) ShowHint 属性：该属性决定是否显示 DBNavigator 组件中各按钮的帮助信息，如果 ShowHint 属性设置为 True 时，鼠标指针在 DBNavigator 组件中某一按钮上停留一段时间后，就会自动地出现帮助信息。

(3) Hints 属性：用来为每一个按钮增加帮助信息。如果将 ShowHint 设置为 true，才可以为每一个按钮增加帮助信息。双击 Hints 属性，就可在随后弹出的对话框中输入多行提示文字，如图 11-17 所示。

(4) ConfirmDelete 属性：该属性用来决定用户在删除记录时是否显示要求用户确认的消息框。使用这个属性可以防止用户无意中从数据集里删除了记录。如果 ConfirmDelete 属性设置为 True，在用户按下删除按钮后会弹出一个消息框要求确认，直到用户按下【OK】按钮，记录才会被删除。如果 ConfirmDelete 属性设置为 False，在删除记录前就不会出现要求确认删除的消息框。

(5) OnClick 事件：该事件在 DBNavigator 组件中的某个按钮被单击并执行完相应的操作之后被触发。

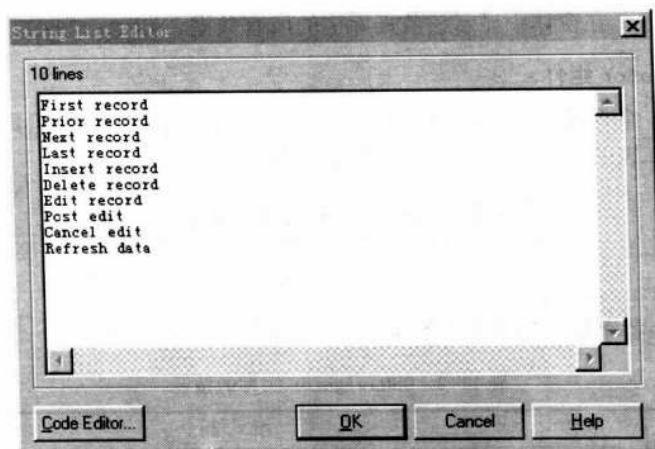


图 11-17 Hints 属性对话框

下面的执行结果就是在窗体上放置了 DBNavigator 组件，设置其 DataSource 属性为 DataSource1 后的效果。在 DBNavigator 组件中单击一个按钮，DBGrid 中就会执行相应的操作。如图 11-18 所示。

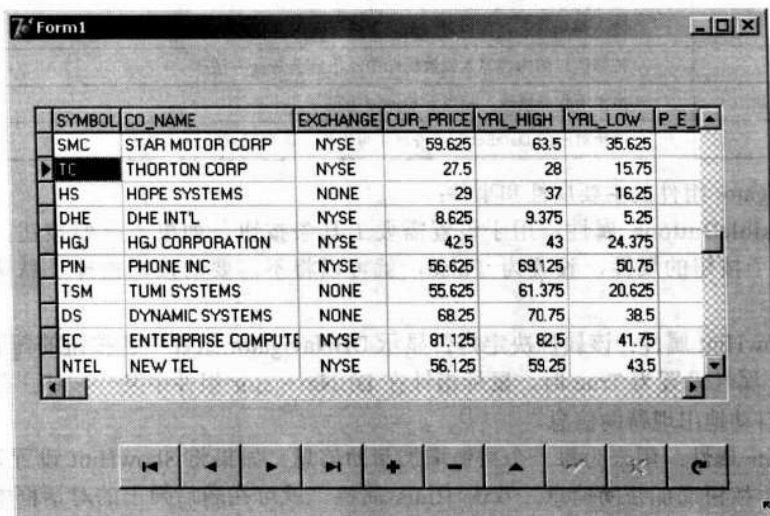



图 11-18 加入 DBNavigator 组件对象的运行结果

11.2.3 DBCtrlGrid 组件

DBCtrlGrid 组件 ，也是表格的一种，但和 DBGrid 不同，DBGrid 以行和列显示记录，DBCtrlGrid 像名片一样显示记录。

DBCtrlGrid 组件的主要属性：

- AllowDelete: 用来决定是否可以用 Ctrl+Delete 删除当前记录。

- AllowInsert: 用来决定是否可以用 Insert 键插入记录。
- ColCount: 用来决定显示多少列版面。
- DataSource: 用来指定数据源。
- EditMode: 用来决定是否可以增加、删除、修改记录。
- Orientation: 当有多行和多列版面时, 用来决定数据库记录的显示顺序是从左到右还上从上到下。
- PanelHeight: 用于设置窗格的高度, 默认是 72。
- PanelWidth: 用于设置窗格的宽度, 默认是 200。
- RowCoun: 用来显示多行版面。
- SelectedColor: 用来设定选中时的颜色。
- ShowFocus: 用来当某个版面接受焦点时, 是否显示虚线框。

下面通过例子说明如何使用 DBCtrlGrid 组件

[例 11-2] DBCtrlGrid 组件的使用。

能够在窗体中以名片的形式显示数据。程序效果如图 11-19 所示。

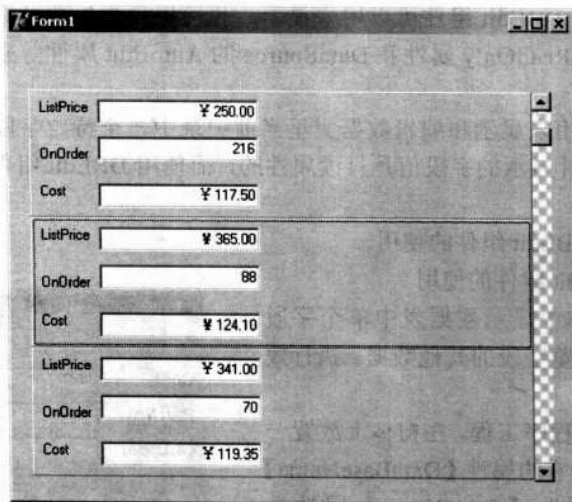


图 11-19 以名片的形式显示数据


(1) 在窗体上放置一个 Table1 组件, 设置它的属性【DataBaseName】属性为“DBDEMOS”, 【TableName】属性为“parts.db”, 把【Active】属性设置为“true”。选取一个 DataSource1 组件, 将其【DataSet】属性设置为“Table1”。拖动 DBCtrlGrid1 组件到窗体上, 设置它的【DataSource】属性为“DataSource1”。

(2) 打开 Table1 字段编辑器, 第一次打开时字段编辑器是空的, 单击右键, 选择【Add All fields】选项, 将字段添加到字段编辑器里。

(3) 选中“ListPrice”、“Cost”和“On Order”三个字段, 把它们拖到 DBCtrlGrid1 组件中。

(4) 运行程序, 在窗体中显示象名片一样的数据记录。


11.2.4 DBText 组件

DBText 组件 ，是一个只读的数据显示组件，它类似于 Label 组件。由于该组件显示的是表中当前记录的值，因此它的显示内容是动态变化的。用户在数据库中移动光标时，组件中显示的数据会相应地发生改变。

DBText 组件的主要属性有：

- DataField 属性:用来确定 DBText 组件显示的是当前记录中哪个特定字段的值。在为 DBText 组件对象指定了数据源后，在对象查看器中该对象的 DataField 属性框会自动产生一个下拉列表。包含了当前打开数据集中的所有字段。
- Transparent 属性:该属性值为布尔类型，当 Transparent 属性设置为 True 时，文字会以透明方式显示（即不覆盖背景图案）。

11.2.5 DBEdit 组件

DBEdit 组件 ，是专门用来显示和编辑数据库表中的单个字段值，在应用程序常常用一个 DBEdit 组件对应一个字段，通过设置它的 DataSource 属性和 DataField 属性来指定对应的表和字段。DBEdit 组件可以用来显示，也可以用来修改和编辑数据。可以通过设置 DBEdit 组件的 ReadOnly 属性和 DataSource 的 AutoEdit 属性等来设定不同的工作方式。

DBEdit 组件是用来显示和编辑数据集里当前记录中一个特定字段的值的数据控制组件。而用 DBText 组件显示的字段值是只读属性的，而使用 DBEdit 组件允许用户编辑修改数据。

下面举例说明 DBEdit 组件的使用。

【例 11-3】DBEdit 组件的使用

程序能够在窗体中显示数据表中单个字段值，通过按钮可以浏览字段的其他数据。执行效果如图 11-20 所示。

(1) 新建一应用程序工程。在窗体上放置一个 Table1 组件，设置它的属性【DataBaseName】属性为“DBDEMOS”，【TableName】属性为“parts.db”，【Active】属性为“true”。

(2) 再添加一个 DataSource1 组件，将其【DataSet】属性设置为“Table1”。

(3) 在窗体上放置 3 个标签组件，其【Caption】属性分别为“Cost”、“PartNo”、“OnOrder”。

(4) 分别添加三个 DBEdit 组件，其【DataSource】属性为“DataSource1”，【DataField】属性分别为“Cost”、“PartNo”、“OnOrder”。

(5) 拖动 DBNavigator 组件到窗体中，其【DataSource】属性为“DataSource1”。

(6) 运行程序，在窗体中显示“Cost”、“PartNo”、“OnOrder”三个字段的值，并可以通过按钮浏览字段其他的值。

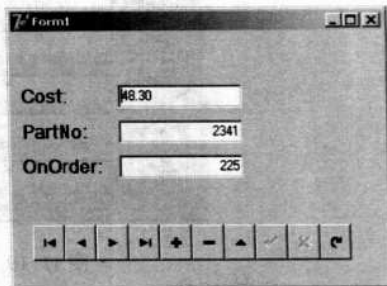



图 11-20 执行结果

11.2.6 DBMemo 组件

DBMemo 组件，主要用于浏览数据库中备注型的字段。可以用来显示数据库中当前记录中的 BLOB 型字段。

DBMemo 组件除了具有和其他数据控制组件一样的 DataSource 和 DataField 属性以外，还具有以下不同的属性：

(1) AutoDisplay 属性：该属性用来决定是否自动显示一个数据集中的备注类型的字段值。

当 AutoDisplay 属性值设置为 True 时，DBMemo 组件将随着当前记录的改变自动显示该字段新的内容。如果 AutoDisplay 属性值设置为 False 时，DBMemo 组件在当前记录改变时只显示字段名。


使用 AutoDisplay 属性的目的是为了提高应用程序的运行速度，因为 DBMemo 组件是使用 Text 属性来显示大量文本信息的，当 AutoDisplay 属性值设置为 True 时，应用程序必须消耗大量时间来更新显示信息。这样会影响程序运行速度。

(2) MaxLength 属性：该属性用来限定可以输入 DBMemo 组件中的最大字符数。此属性值为 0 时，表示不限制最大字符数。

(3) WordWrap 属性：该属性值为布尔类型，用来指明向 DBMemo 组件中输入的文本到了右边界处是否自动换行。

(4) ReadOnly 属性：该属性值为布尔类型，ReadOnly 属性设置为 True 时，组件中显示的文本是只能读而不能被编辑修改。

11.2.7 DBImage 组件

DBImage 组件，用于浏览数据库中数据的图像框。可以用于显示、复制、粘贴数据库中图像类型的字段。

DBImage 组件的重要属性与 DBMemo 组件类似，它也主要包含 DataSource 属性、DataField 属性和 AutoDisplay 属性。

在 DBImage 组件中是允许用户对位图图像进行编辑的，如将图像剪切或拷贝到剪切板上或从剪切板上粘贴到 DBImage 组件中等操作，同时也可以程序中调用 CutToClipboard、CopyToClipboard、PasteFromClipboard 方法来实现剪切、拷贝、粘贴等操作，要进行上述操作，必须保证 DBImage 组件的 ReadOnly 属性值为 False。

下面举例说明 DBImage 组件的使用。

[例 11-4] DBImage 组件的使用

效果：在窗体中可以显示数据表中的图片。如图 11-21 所示。

(1) 新建一应用程序工程。在窗体上放置一个 Table1 组件，设置它的属性【DataBaseName】属性为“DBDEMOS”，【TableName】属性为“Clients.db”，把【Active】属性为“true”。选取一个 DataSource1 组件，将其【DataSet】属性设置为“Table1”。

(2) 在窗体上放置 2 个标签组件，其【Caption】属性分别为“CITY”、“DATE_OPEN”，即选择“Clients.db”数据库中的“CITY”、“DATE_OPEN”两个字段。

(3) 增加一个 DBImage 组件，设置其【DataSource】属性为“DataSource1”，【DataField】属性为“IMAGE”（字段）。

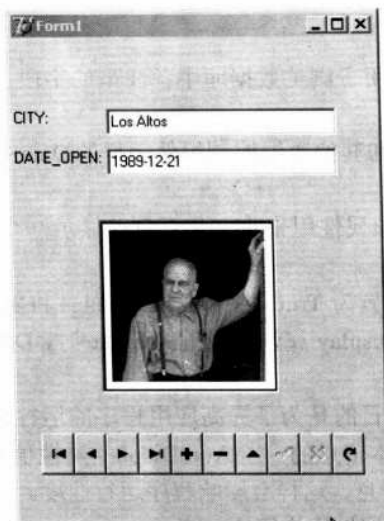










图 11-21 执行结果

(4) 拖动 DBNavigator 组件到窗体中，其【DataSource】属性为“DataSource1”。

(5) 点击 DBNavigator 组件的按钮，可以浏览数据库中所有照片。

(6) 运行程序，在窗体中显示“CITY”、“DATE_OPEN”两个字段的值，显示该数据表中“IMAGE”（字段）的图片，同时可以通过按钮浏览数据。

11.2.8 其他数据浏览组件

- DBListBox 组件 ：浏览数据库中数据的列表框。
- DBComboBox 组件 ：浏览数据库中数据的组合框。
- DBCheckBox 组件 ：浏览数据库中数据的复选框。用来显示和编辑表中的布尔型字段。
- DBRadioGroup 组件 ：浏览数据库中的数据单选按钮组。
- DBLookupList 组件 ：浏览数据库中数据的列表框。在基于一个表的应用中，用它可以显示另一个表中一个指定的字段值。
- DBLookupComboBox 组件 ：浏览数据库中数据的组合框。在基于一个表的应用中，用它可以显示另一个表中一个指定的字段值。
- DBRichEdit 组件 ：主要用于 RichEdit 文本类型字段的显示和编辑，可以进行多行显示。
- DBChart 组件 ：由 Chart 组件派生而来，该组件中包含 DataSource 属性，用于数据集中数据汇总得到的图形。

11.3 人事管理系统开发

开发数据库应用程序，除了要掌握 Delphi 7 提供的数据库组件和一些辅助工具的使用

外, 还需要能对具体的任务进行分析。本节介绍了一个人事管理系统的设计过程。通过这个实例来了解开发数据库应用程序的一般方法。

11.3.1 功能说明

人事管理系统就是要实现对某单位的职工进行管理, 整个系统包括人事资料输入、资料查询、资料删除和打印报表等功能。

在应用系统使用中, 为保证系统的安全, 必须设置用户检测程序。合法用户可进入, 非法用户拒绝登录。同时要能够设置用户的使用权限, 用户登录后只能进行其权限所允许的操作。用户可以修改本人密码, 系统管理员可以修改、冻结或删除普通用户。

11.3.2 创建数据表

为了人事管理系统正常运行, 需要创建两个数据表: 一个是操作用户数据表; 另一个是人事信息数据表。本系统数据库存取方式采用 BDE。数据表驱动采用 Paradox 数据表。

11.3.2.1 创建及配置数据库别名

利用 BDE 存取数据表, 必须事先创建数据库别名, 该别名定义了数据表的路径和所使用的数据库驱动程序。下面说明创建和配置数据库别名的步骤:

(1) 进入 BDE 管理程序中, 进行数据库别名管理, 如图 11-22 所示。

(2) 单击 BDE 管理界面的下拉菜单中【Object】/【New】选项, 开始创建一个新的数据库别名, 首先要选择数据库驱动程序名, 如图 11-23 所示。我们选择采用 Paradox 数据库表格式, 选择 Standard 驱动。



图 11-22 BDE 管理界面

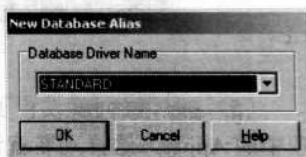


图 11-23 选择数据库驱动程序

(3) 单击【OK】按钮, 就增加了一个默认别名为 Standard1 的设置项, 该别名为 rsgl, 并且在该别名的定义项中, 修改其中的路径项的值, 该值为我们将要建立数据表的目录路径, 如图 11-24 所示。

(4) 另外, 要使由该别名驱动的数据表能正常支持中文输入, 必须要修改 Paradox 驱动设置。单击 BDE 管理器中 Configuration 页进行数据库驱动设置, 单击【Native】的【Paradox】, 出现 Paradox 数据表驱动的配置项, 修改【LangDriver】项中的值为“aradox China 936”。



图 11-24 设置别名 rsgl

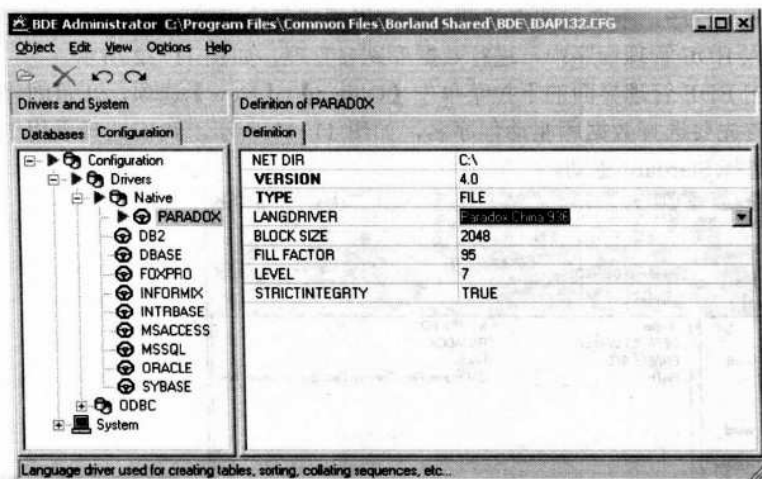


图 11-25 修改 Paradox 默认设置值

(5)这时,我们就设置了数据库别名 rsgl,要使该别名立即生效,单击菜单中的【Object】项中的 Apply,这样别名就设置完毕。

11.3.2.2 数据表说明

在该系统中,将定义两个数据表,第一个数据表 Operator.db,该数据表每条记录对应一个操作用户,该记录包含操作员的用户名、密码、所在单位和操作权限等,如表 11-10 所示。

第二个数据表 Info.db,该数据表每条记录对应一名职工的信息,包括职工姓名、性别、出生年月、通信地址、工作单位、身份证号码、联系电话、职称等信息,如表 11-11 所示。

表 11-10 Operator.db 数据表说明

序 号	字 段 名	属 性	宽 度	说 明
1	Name	A	8	用户名, 主键
2	Password	A	8	密码
3	Department	A	20	所在单位
4	Right_1	L		权限一: 管理权限
5	Right_2	L		权限二: 操作权限
6	Right_3	L		权限一: 查询权限
7	Stamp	Date		最后操作时间

表 11-11 Info.db 数据表说明

序 号	字 段 名	属 性	宽 度	说 明
1	Number	A	6	职工编号
2	Name	A	8	职工姓名
3	Sex	A	2	性别
4	Age	S		
5	Entrance	Date		参加工作时间

11.3.2.3 创建数据表

下面以“Operator.db”为例, 讲述如何利用 Delphi 7 中的 Database Desktop 创建数据表的步骤:

(1) 启动【Database Desktop】系统, 如图 11-26 所示。



图 11-26 【Database Desktop】系统

(2) 单击【File】菜单中的【New】的【Table】选项, 出现数据表选择对话框, 如图 11-27 所示。

(3) 单击【OK】按钮, 就进入了创建数据表的对话框。

(4) 输入字段信息, 如图 11-28 所示。

(5) 单击【Save As】按钮, 将保存已创建的数据表结构, 这时提示输入数据表名。可以通过 Alias 选择已定义的别名, 并且在文件名栏中输入您要命名的文件名, 如图 11-29 所示。



图 11-27 数据表选择对话框

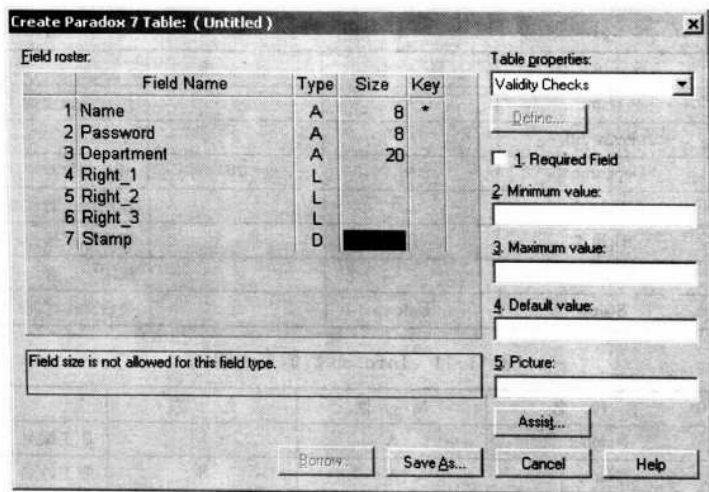


图 11-28 创建数据表对话框

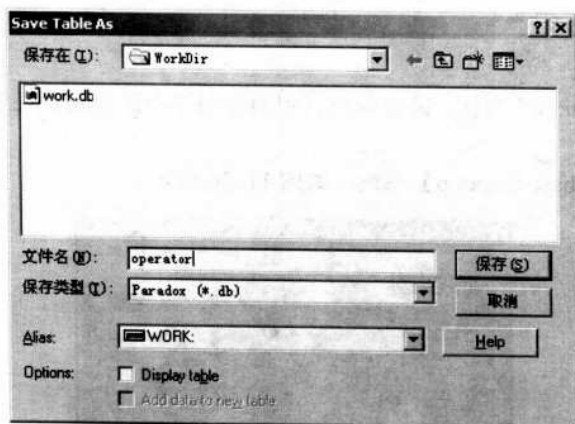


图 11-29 保存表结构

(6) 单击【保存】按钮即可，可以浏览刚创建的空的数据库。

(7) 用同样的方法创建数据库“Info.db”。

11.3.3 程序设计

本程序的主界面主要通过一个下拉菜单来实现各项功能，我们把这些功能分为 3 组。第一组主要是系统功能，包括用户管理模块，该模块主要是实现操作用户的增加、删除和修改。密码修改模块，该模块主要实现各操作用户修改自己的操作密码，系统管理用户可以修改其他用户的密码。系统初始化模块主要用来实现初始化功能，即清除所有的数据库表中的信息，只在 Operator.db 数据库中保留一条默认的管理员信息。退出模块就是退出系统。第二组功能就是人事管理，这里主要实现人事信息的增加、修改和删除等功能。第三组功能主要实现人事信息查询，可以实现按各种条件进行查询，并且打印查询结果。

11.3.3.1 主界面程序设计

在主界面利用快捷键实现以上各项功能,如图 11-30 所示。

设计步骤如下:

(1) 新建一个工程文件,保存源程序文件名为 main.pas,工程文件名为 grsl.dpr,并且修改【Name】属性值为“Main_Form”,【Caption】属性为“人事管理系统”。

(2) 在窗体中,加入 MainMenu 组件,其【Name】属性为“MainMenu1”。并分别定义相应的菜单项。

(3) 往窗体中,加入 ToolBar 组件,其【Name】属性为“ToolBar1”。

(4) 再往窗体中加入 6 个 SpeedButton 组件按钮,作为快捷按钮,这些组件对象的属性值如表 11-12 所示。

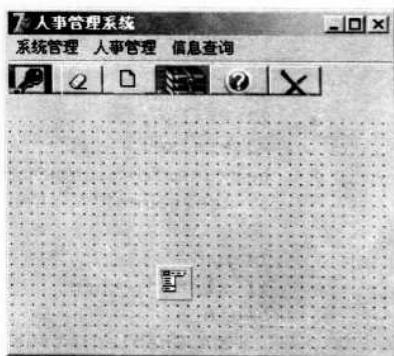


图 11-30 系统主界面

表 11-12 快捷键配置说明

序 号	对 象 名	属 性 名	属 性 值	说 明
1	SpeedButton1	Hint	“用户管理”	用户管理按钮
		ShowHint	True	
		Glyph	用户管理.bmp	
2	SpeedButton2	Hint	“修改密码”	修改用户密码模块
		ShowHint	True	
		Glyph	修改密码.bmp	
3	SpeedButton3	Hint	“初始化”	系统初始化模块
		ShowHint	True	
		Glyph	系统初始化.bmp	
4	SpeedButton4	Hint	“人事管理”	人事管理模块
		ShowHint	True	
		Glyph	人事管理.bmp	
5	SpeedButton5	Hint	“信息查询”	信息查询模块
		ShowHint	True	
		Glyph	信息查询.bmp	
6	SpeedButton6	Hint	“退出”	退出应用系统
		ShowHint	True	
		Glyph	退出.bmp	

(5) 系统中设置全局变量 username (用户名, String 类型), x1, x2, x3 (用户操作权限, Boolean), 这些变量在用户调用用户检测模块后, 得到登录用户的用户名和权限, 然后对用户能操作的菜单项和快捷按钮进行相应的屏蔽。

(6) 为窗体创建 OnActive 事件处理, 当程序启动时, 触发该事件处理, 该事件主要用作初始化。该事件处理代码如下:


```

procedure Tmain_Form.FormActive(Sender :TObject)
begin
    //用户检测初始化
    username:='';
    password:='';
    x1:=false;
    x2:=false;
    x3:=false;
    checkuserform.ShowModal(); //启动用户检测模块
    if username<>' ' then //用户登录,根据读取的用户权限,相应设置对
        应的菜单允许或屏蔽
    begin
        if x1=ture then //管理员权限
        begin
            n4.Enabled:=true;
            n6.Enabled:=true;
            speedbutton1.Enabled:=true;
            speedbutton3.Enabled:=true;
        end
        else
        begin
            n4.Enabled:=false;
            n6.Enabled:=false;
            speedbutton1.Enabled:=false;
            speedbutton3.Enabled:=false;
        end;
        if x2=ture then //操作员权限
        begin
            n2.Enabled:=true;
            speedbutton4.Enabled:=true;
        end
        else
        begin
            n2.Enabled:=false;
            speedbutton4.Enabled:=false;
        end;
        if x3=ture then //查询权限
        begin
            n3.Enabled:=true;
            speedbutton3.Enabled:=true;
        end
        else
        begin
            n3.Enabled:=false;
            speedbutton5.Enabled:=false;
        end;
    end
    end
    else
        close; //当用户输入用户名或密码错误时,或取消登录
    end; //时,退出系统
end;

```


11.3.3.2 用户检测模块设计

用户检测模块就是在启动应用系统时, 先进行用户登录, 只有合法用户才能允许进入系统进行操作, 该模块允许用户有三次输入机会, 如果三次输入均出现用户名或密码错误, 则拒绝用户登录。检测方法是待用户输入完用户名和密码后, 则打开数据表 Operator.db, 检测表中是否有用户名和密码都正确的记录, 如果有, 则读取其相应的操作权限, 如果没有, 则退出。该模块的设计步骤如下:

(1) 往工程文件 rsgl.dpr 中, 加入一个新的窗体, 保存该文件为 user.pas, 设置其【Caption】属性为“用户检测”, 设置其【Name】属性为“userform”。

(2) 往该窗体中加入 2 个 Edit 组件, 2 个标签组件, 2 个命令按钮组件。组件在窗体上的布置如图 11-31 所示。

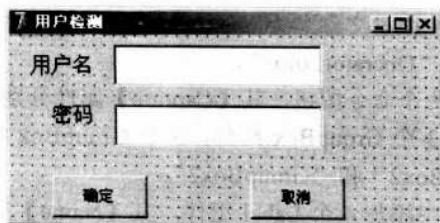


图 11-31 用户检测模块设计界面

(3) 双击【确定】按钮, 建立 OnClick 事件, 程序代码如下:

```
procedure TuserForm.Button1Click(Sender:TObject)
var
    Table1:TTable;
begin
    Table1:=TTable.Create(self);
    Table1.DatabaseName:='rsgl';
    Table1.TableName:='operator.db';
    Table1.Open;
    Table1.SetKey;
    Table1.FieldName('name').AsString:=Edit1.Text;
    if (Table1.GotoKey)and (Table1.FieldName('password').AsString
        =Edit2.Text) then
    begin
        //用户名和密码验证正确, 读取相应的操作权限
        username:=Edit1.Text;
        password:=Edit2.Text;
        x1:=Table1.fieldByName('right_1').asboolean;
        x2:=Table1.fieldByName('right_2').asboolean;
        x3:=Table1.fieldByName('right_3').asboolean;
        Table.Close;
        Close;
    end
    else
    begin
        //用户名和密码输入错误
        Table.Close;
        Application.MwssageBox('用户名或密码输入错误, 请检查再重试!', '提示信
```

```

        息', mb_ok);
    Edit1.Text:='';
    Edit1.Text:='';
    Edit1.SetFocus;
    I:=i-1;
    if i=0 then
        Close;
    End;
end;

```

11.3.3.3 操作员管理模块设计

该模块主要是维护 Operator.db 数据表，用来增加、修改和删除操作员信息。并且设计操作员的允许权限。此模块设计过程如下：

(1) 新建一个窗体，存盘文件名为 operator.pas，该窗体的【Caption】属性为“用户管理”，【Name】属性值为“OperatorForm”。

(2) 在窗体中加入两个命令按钮，其【Caption】属性分别为“保存”和“退出”。

(3) 往窗体中添加 3 个 GroupBox 组件，3 个 GroupBox 组件的【Name】属性分别为“GroupBox1”、“GroupBox2”和“GroupBox2”。

(4) 在 GroupBox1 中添加 4 个 Label 组件，其【Name】属性分别为：“用户名”、“密码”、“单位”和“操作时间”；在添加 4 个 Edit 组件，其【Text】属性都为空，其中操作时间对应的 Edit4 组件的【ReadOnly】属性设置为“True”，【TabStop】属性设置为“False”。

(5) 在 GroupBox2 中添加 3 个复选框，其【Caption】属性分被为“管理员权限”、“操作员权限”和“查询权限”。

(6) 在 GroupBox3 中添加 3 个单选按钮，其【Caption】属性分被为“增加”、“修改”和“删除”。

(7) 设计的窗体界面如图 11-32 所示。

图 11-32 用户管理模块设计界面

(8) 该模块窗体的 OnActive 事件处理过程代码如下：

```

procedure TOperatorForm.FormActive(Sender:TObject)
begin
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit4.Text:='';
    Edit4.ReadOnly:=True;
    Edit4.TabStop:=False
    Checkbox1.Checked:=False;
    Checkbox2.Checked:=False;
    Checkbox3.Checked:=False;
    Edit4.Text:=Datetostr(Date());
    Table1:=TTable.Create(self);
    Table1.DatabaseName:='rsgl';
    Table1.TableName:='operator.db';
    Table1.Open;
    if not Table1.Eof then
        LoadFromTable;
    Table1.Close;
    RadioButton2.Checked:=True;
end;

```

(9) 为【保存】按钮建立 OnClick 事件处理程序代码:

```

procedure TOperatorForm.Button1Click(Sender:TObject)
begin
    Table1:=TTable.Create(self);
    Table1.DatabaseName:='rsgl';
    Table1.TableName:='operator.db';
    Table1.Open;
    Table1.SetKey;
    Table1.FieldName('name').AsString:=Edit1.Text;
    if Table1.gotokey then
    begin
        if RadioButton1.Checked then
        begin
            ShowMessage('用户名相同');
            Edit1.Text:='';
            Edit2.Text:='';
            Edit3.Text:='';
            Edit1.SetFocus;
        end
        else if RadioButton2.Checked then
        begin
            Table1.Edit;
            SaveToTable;
            Table1.Post;
        end
        else
        begin
            Table1.Edit;
            Table1.Delete;
        end
    end
end;

```



```

        Table1.First;
        LoadFromTable;
    end;
end
else
begin
    if RadioButton1.Checked then
    begin
        Table1.Append;
        Table1.Edit;
        SaveToTable;
        Table1.Post;
    end
    else
    begin
        ShowMessage('没有该用户');
        Edit1.Text:='';
        Edit2.Text:='';
        Edit3.Text:='';
        Edit1.SetFocus;
    end;
end;
Table1
end;

```

11.3.3.4 密码修改

该模块主要是用来修改用户密码的,如果登录用户具有管理权限,则可以修改其他用户的密码,若该用户不具备管理员权限,则只能修改自己的密码,该模块设计过程如下:

(1) 新建一个窗体,存盘文件名为 `changepassword.pas`,设置该窗体的【Caption】属性为“密码修改”,设置其【Name】属性为“PasswordForm”。

(2) 打开密码修改窗体,在窗体上添加 3 个标签组件,将其分别命名为“原密码”、“新密码”和“确认密码”。

(3) 再添加 3 个编辑组件,将它们的【Text】属性设置为空,分别用于输入原密码(Edit1)、新密码(Edit2)和确认新密码(Edit3)。

(4) 加入两个 Button1 组件,将其【Caption】属性分别设置“确定”和“取消”。

(5) 该模块的界面如图 11-33 所示。

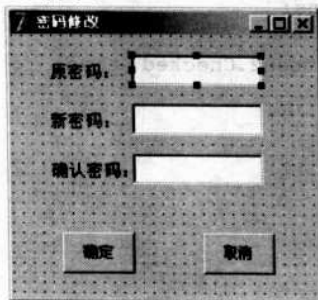


图 11-33 密码修改设计界面

(6) 为【确定】按钮添加单击事件处理过程代码:

```
procedure TPasswordForm.Button1Click(Sender:TObject)
var
  Table1:TTable;
begin
  Table1:=TTable.Create(self);
  Table1.DatabaseName:='rsgl';
  Table1.TableName:='operator.db';
  Table1.Open;
  Table1.SetKey;
  Table1.FieldByName('Password').AsString:=Edit1.Text;
  if (Table1.GotoKey) then
  begin
    if Edit1.Text=Edit2.Text then
    begin
      if Edit2.Text=Edit3.Text then
      begin
        Table1.Edit2;
        Table1.FieldByName('Password').AsString:=Edit2.Text;
        Table1.post;
        ShowMessage('密码修改成功');
      end
    end
    else
    begin
      ShowMessage('新密码不相等');
    end
  end
  else
  begin
    ShowMessage('原密码错误');
  end
end;
```

(7) 运行程序, 密码修改成功时界面如图 11-34 所示。

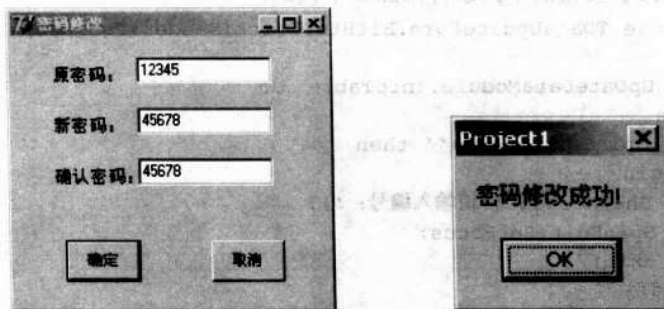


图 11-34 密码修改运行结果

11.3.3.5 人事信息模块设计

该模块主要进行人事信息管理, 包括人事信息的添加、删除和修改等功能, 模块主要对 Info.db 数据表进行存取。在主菜单中, 通过单击“人事管理”菜单项, 触发事件调用该模块运行。模块设计如下:

(1) 在工程文件中, 新建窗体, 存盘并且命名为 rsxx.pas, 设置其【caption】属性为“人事管理”, 【Name】属性值为“DataUpdateForm”。

(2) 在窗体中添加 5 个 Label 组件, 将其分别命名为“姓名”、“编号”、“性别”、“年龄”和“工作时间”。

(3) 然后再添加 3 个 Edit 组件, 分别对应于“姓名”(Edit1)、“性别”(Edit3)和年龄“(Edit4)项, 用于输入和修改职工的这几项信息, 并将他们的【Text】属性都设置为空。

(4) 添加一个 SpinEdit 组件, 用于输入和修改职工的“编号”信息, 组件的最大值和最小值分别设定为“19999999”和“10000000”。

(5) 接着添加一个 DateTimePicker 组件, 用于输入和修改职工的“工作时间”信息。

(6) 添加 3 个 Bitbtn 组件, 将它们的【Caption】属性分别设置为“修改”、“添加”和“删除”。此时窗体界面设计如图 11-35 所示。

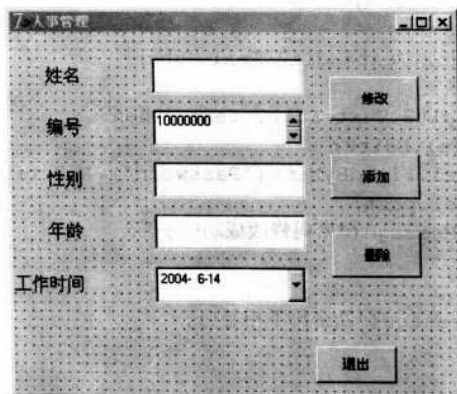


图 11-35 界面设计

(7) 为【修改】按钮的单击事件添加如下代码:

```
procedure TDataUpdateForm.BitBtn1Click(Sender:TObject)
begin
  with UpDateDataModule.InfoTable do
  begin
    if SpinEdit1.Text='' then
    begin
      ShowMessage('请输入编号: ');
      SpinEdit1.SetFocus;
      exit;
    end;
    if not Locate('Number',VarArrayOf([SpinEdit1.Text]),
      [LopartialKey]) then
    begin
      ShowMessage('无此记录, 不能修改! ');
      Exit;
    end;
    Edit;
    FieldByName('Name').AsString:=Edit1.Text;
    FieldByName('Number').AsString:=SpinEdit1.Text;
    FieldByName('Sex').AsString:=Edit3.Text;
    FieldByName('Age').AsString:=Edit4.Text;
```

```

        FieldByName('Entrance').AsDateTime:=DateTimePicker1.Date;
    Post;
    ShowMessage('修改成功!');
end;
end;

```

(8) 为【添加】按钮的单击事件添加如下代码:

```

procedure TDataUpdateForm.BitBtn2Click(Sender:TObject)
begin
    with UpDateDataModule.InfoTable do
    begin
        if ((SpinEdit1.Text<>'') and (Edit1.Text<>'')) then
        begin
            if Locate('Number',VarArrayOf([SpinEdit1.Text]),[LopartialKey]) then
            begin
                ShowMessage('此职工已经存在!');
                exit;
            end;
            try
                Append;
                FieldByName('Name').AsString:=Edit1.Text;
                FieldByName('Number').AsString:=SpinEdit1.Text;
                FieldByName('Sex').AsString:=Edit3.Text;
                FieldByName('Age').AsString:=Edit4.Text;
                FieldByName('Entrance').AsDateTime:=DateTimePicker1.Date;
                Post;
                ShowMessage('添加成功!');
            except
            begin
                Active:=false;
                ShowMessage('输入编号的范围应在: 10000000-19999999 之间!');
                Active:=true;
                exit;
            end;
        end;
    end else ShowMessage('姓名和编号不能为空!');
    end;
end;

```

(9) 为【删除】按钮的单击事件添加如下代码:

```

procedure TDataUpdateForm.BitBtn3Click(Sender:TObject)
begin
    with UpDateDataModule.InfoTable do
    begin
        if Locate('Number',VarArrayOf([SpinEdit1.Text]),[LopartialKey])
        then begin
            Delete;
            Edit1.Text:='';
            Edit3.Text:='';
            Edit4.Text:='';

```

```

        ShowMessage('删除成功!');
    end else ShowMessage('无此记录!');
end;
end;
end;

```

(10) 给 SpinEdit 组件的几个事件添加完整性控制代码:

```

procedure TDataUpdateForm.SpinEdit1Change(Sender:TObject)
begin
    DataUpdateForm.SpinEdit1Exit(Sender);
end;
procedure TDataUpdateForm.SpinEdit1Exit(Sender:TObject)
begin
    if (length(SpinEdit1.Text)<>8) then begin
        ShowMessage('请输入 8 个字符');
        SpinEdit1.SetFocus; exit;
    end;
    with UpDateDataModule.InfoTabel do
    begin
        if Locate('Number',VarArrayOf([SpinEdit.Text]),[LopartialKey])
        then begin
            Edit1.Text:=FieldByName('Name').AsString;
            SpinEdit1.Text:=FieldByName('Number').AsString;
            Edit3.Text:=FieldByName('Sex').AsString;
            Edit4.Text:=FieldByName('Age').AsString;
            DateTimePicker1.Date:=FieldByName('Entrance').AsDateTime;
        end else begin
            Edit1.Text:=''; Edit3.Text:=''; Edit4.Text:='';
        end;
    end;
end;
end;
procedure TDataUpdateForm.SpinEdit1KeyPress(Sender:TObject;var
    Key:Char)
begin
    if key=#13 then
        DataUpdateForm.SpinEdit1Exit(Sender);
end;
end;

```

(11) 程序执行结果如图 11-36 所示。

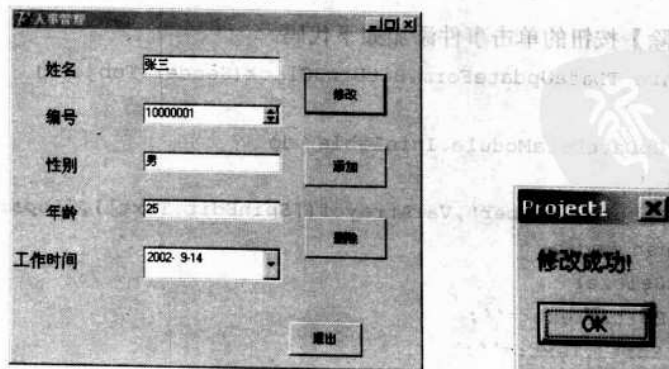


图 11-36 执行结果

11.3.3.6 信息查询模块设计

该模块主要用来对数据表 Info.db 进行数据查询。该模块可以按照职工编号、职工姓名条件进行查询，最终采用 DBGrid 组件对象显示查询到的数据。模块设计如下：

- (1) 在工程文件中，新建窗体，存盘并且命名为 seek.pas，设置其【Caption】属性为“信息查询”，【Name】属性值为“QueryForm”。
- (2) 打开信息查询窗体，在窗体上添加一个 PageControl 组件，在该组件中设置 2 个页面，分别命名为“按编号查询”、“按姓名查询”。将 PageControl 组件的【Align】属性设置为“alClient”以使它填满窗体的窗口区域。
- (3) 在“按编号查询”标签页添加一个 GroupBox 组件。将它的【Caption】属性设置为“输入编号”，并在它内部添加一个 Edit 组件，将此组件的【Text】属性的值清空。
- (4) 添加一个 DBGrid 组件和一个 DBNavigator 组件，它们的属性都使用默认值。
- (5) 最后在窗体中添加一个用来查询的 Bitbtn 按钮组件，将其【Caption】属性设置为“查询”。
- (6) 按编号查询标签页设计完后的界面如图 11-37 所示。

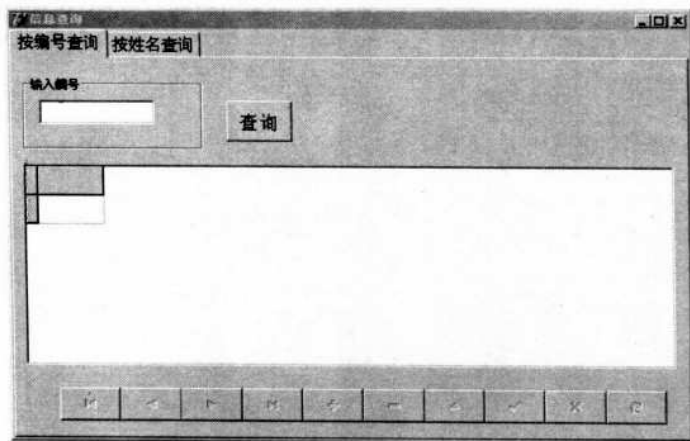


图 11-37 “按编号查询”标签的窗体

- (7) 编写查询代码，将窗口切换到代码编辑器，在 implementation 关键字的下面添加 uses QueryDataModuleUnit 语句。

- (8) 为【查询】按钮的单击 OnClick 事件的处理过程添加如下代码：

```
procedure TqueryForm.BitBtn1Click(Sender:TObject)
begin
  with QueryDataModule do
  begin
    with InfoQuery do
      Close;
      if sql.Count=3 then SQL.Delete(2);
      SQL.Append('where Number=:tt');
      ParamByName('tt').AsString:=Edit1.Text;
      Prepare;
```

```
Open;  
DBNavigator.DataSource:=Info;  
DBGrid1.DataSource:= Info;  
end  
end;
```

(9) “按名字查询”同“按编号查询”的设计方法一样，请读者自行尝试。

11.4 小结

本章主要讲述了数据库的基本组件性、方法和事件，以及这些组件的使用步骤，为读者深入学习和使用其他组件打下基础。最后以人事管理系统的实例阐述如何完成系统的开发，为读者以后进行大系统开发，提供了参考。Delphi 7 数据库无论是组件，还是程序设计方法都非常丰富，不可能都讲解到，其他问题可参考其他书籍。



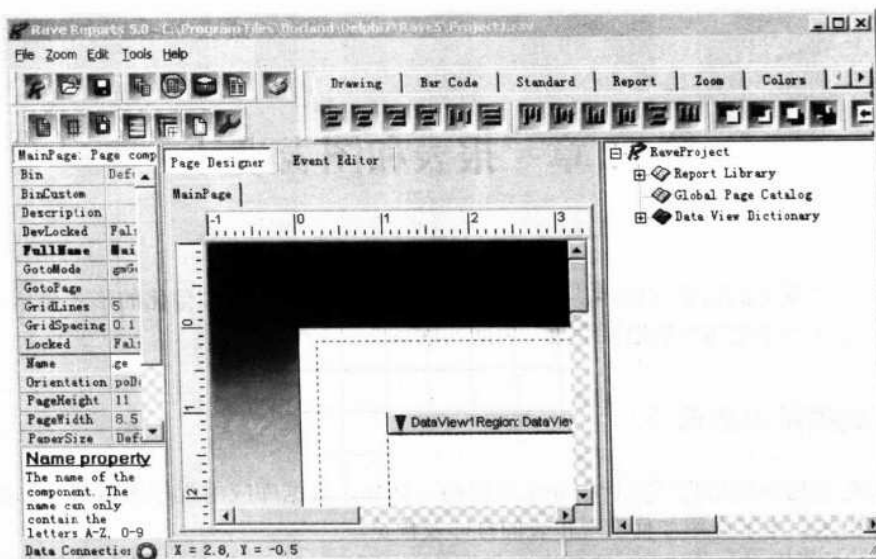


图 12-2 程序的主画面



图 12-3 报表设计向导

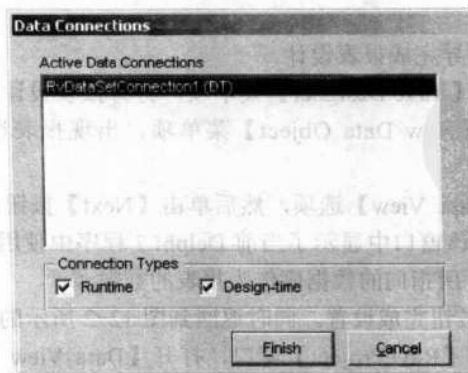


图 12-4 选择数据源

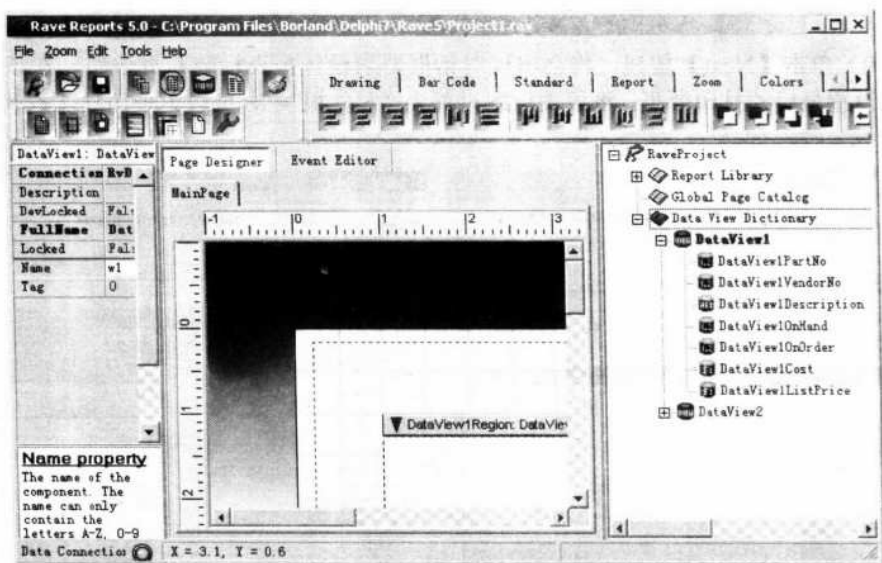


图 12-5 程序中显示了所有的字段

(6) 选择其中的一个字段，然后在窗体的左侧对应着该字段的属性，其中显示了字段名称，是否处于锁定状态，以及简介等属性。

(7) 单击【Tools】菜单中的【Report Wizards】的【Simple Table】选项，将会出现报表设计向导，如图 12-6 所示。

(8) 选择 DataView1，然后单击【Next】按钮，将会出现一个对话框，要求选择要显示的字段，选择其中的 PartNo、VendorNo、Description、OnHand、OnOrder 和 Cost 字段。如图 12-7 所示。

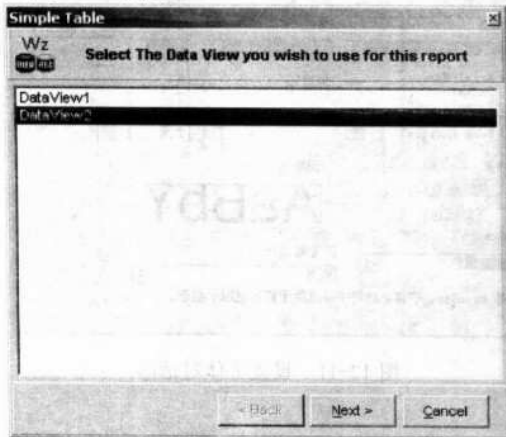


图 12-6 报表设计向导

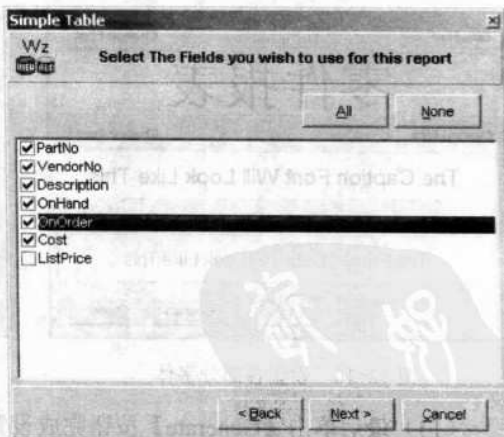


图 12-7 设置要显示的字段

(9) 单击【Next】按钮，将会出现如图 12-8 所示的设置字段显示顺序对话框。要更改某一个字段的显示顺序，可以先选中它，然后单击窗口上方的上下箭头更改字段的显示

顺序。

(10) 单击【Next】按钮, 将会出现对话框要求设置报表标题和页边距, 在该对话框中可以设置最后的打印效果。将报表标题设置为“零件报表”, 页边距采用默认设置, 如图 12-9 所示。

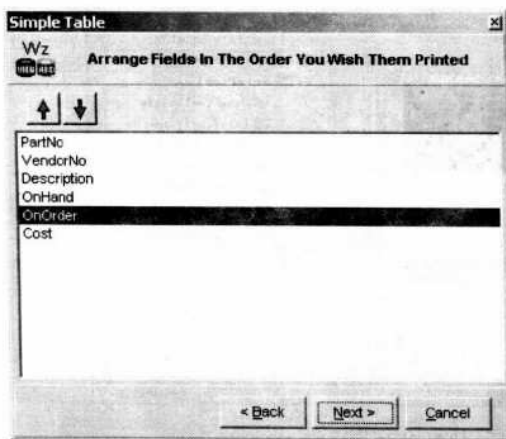


图 12-8 设置字段显示顺序图



图 12-9 设置标题和页边距

(11) 单击【Next】按钮, 将会出现如图 12-10 所示的设置各部分字体对话框, 其中包括主题文字、标题文字以及正文文字等三部分。

(12) 单击每部分对应的【Change Font】按钮, 将会出现如图 12-11 所示的设置字体对话框, 在其中可以选择合适字体。



图 12-10 设置各部分字体

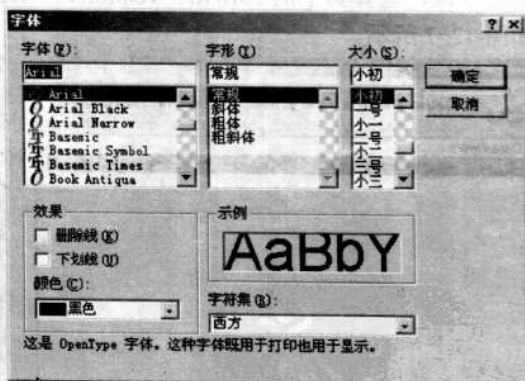


图 12-11 设置字体对话框

(13) 最后单击【Generate】按钮完成设置。

(14) 单击【File】/【Save As】选项将文件保存。

12.1.3 编写程序代码及运行结果

(1) 将【Rave Designer】最小化, 返回到 Delphi 7 程序窗口。

(2) 将【Rave】选项卡上的 RvProject 组件添加到窗体中, 设置该组件的【Project File】属性为刚才制作的 Rave 文件。

(3) 编写 Button1 组件的 OnClick 事件, 来显示预览打印的窗口, 其代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    RvProject1.Execute;
end;
```

(4) 运行程序, 然后单击窗口中出现的按钮, 将会出现如图 12-12 所示的输出选项对话框。

(5) 选择【Preview】选项, 然后单击【OK】按钮, 此时会出现如图 12-13 所示的打印预览效果。

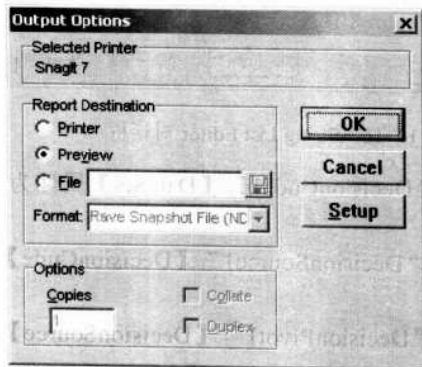


图 12-12 输出选项对话框



图 12-13 打印预览效果

12.2 图表设计

进行数据统计要用到 Decision Cude 组件, 它能对数据做统计并将统计结果用图或表的形式直观地表达出来。Decision Cude 组件与数据库表连接, 能反映数据的变化。用户可以根据自己的需要, 设计形式多样的反映数据库表的统计结果。

Delphi 7 的组件面板的 Decision Cude 选项卡上, 有 6 个组件, 如图 12-14 所示。

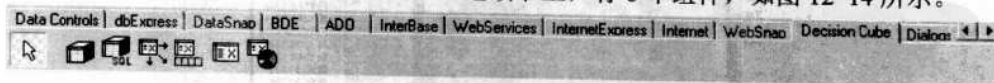


图 12-14 Decision Cude 组件

12.2.1 简单统计图的生成

下面通过使用 Decision Cude 组件生成一个简单的统计图, 来了解统计图的生成过程。

[例 12-1] 简单统计图的设计。

(1) 在一个空白窗体上放置 Decision Query、Decision Cude、Decision Source、Decision Pivot 和 Decision Graph 五个组件。如图 12-15 所示。

(2) 设置 DecisionQuery 组件的【Name】属性为“DecisionQuery1”，【DatabaseName】属性为“DBDEMOS”，SQL 属性设置如图 12-16 所示。

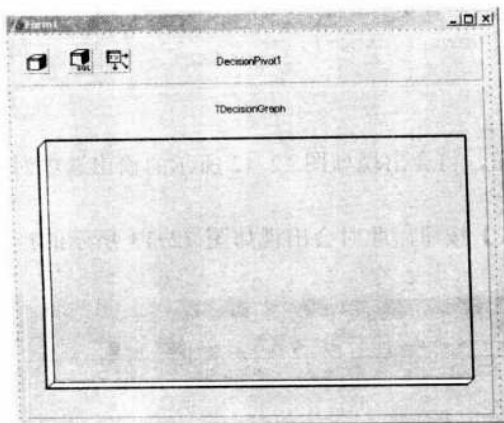


图 12-15 统计图窗体界面

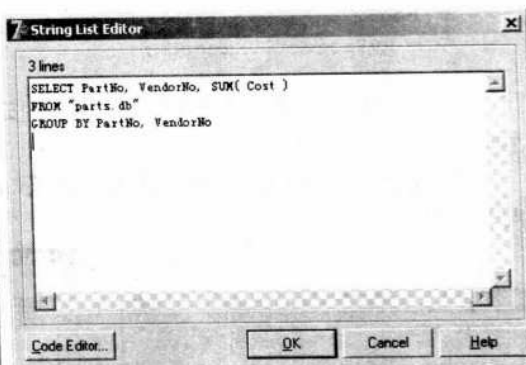


图 12-16 String List Editor 对话框

(3) 设置 DecisionCude 组件的【Name】属性为“DecisionCude1”，【DataSet】属性为“DecisionQuery1”。

(4) 设置 Decision Source 组件的【Name】属性为“DecisionSource1”，【DecisionCude】属性为“DecisionCude1”。

(5) 设置 Decision Pivot 组件的【Name】属性为“DecisionPivot1”，【DecisionSource】属性为“DecisionSource1”。

(6) 设置 Decision Graph 组件的【Name】属性为“DecisionGraph1”，【DecisionSource】属性为“DecisionSource1”。

(7) 设置 DecisionQuery1 对象的【Active】属性为“True”。效果如图 12-17 所示。图中显示的是以 PartNo 为组的统计条状图。

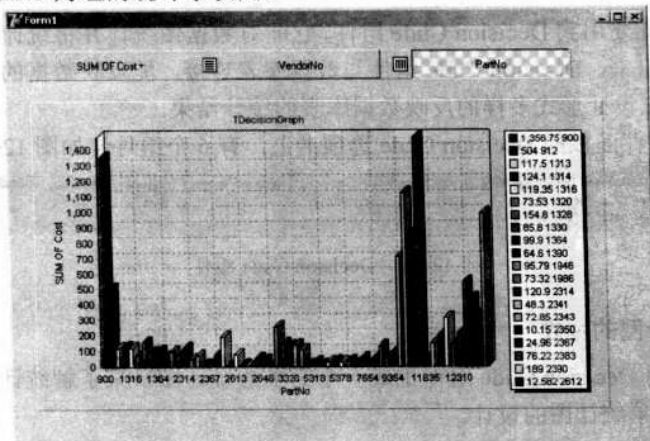


图 12-17 以 PartNo 为组的统计效果

(8) 按下 Decision Pivot 上的【VendorNo】按钮，将显示以 VendorNo 为组的统计图。如图 12-18 所示。

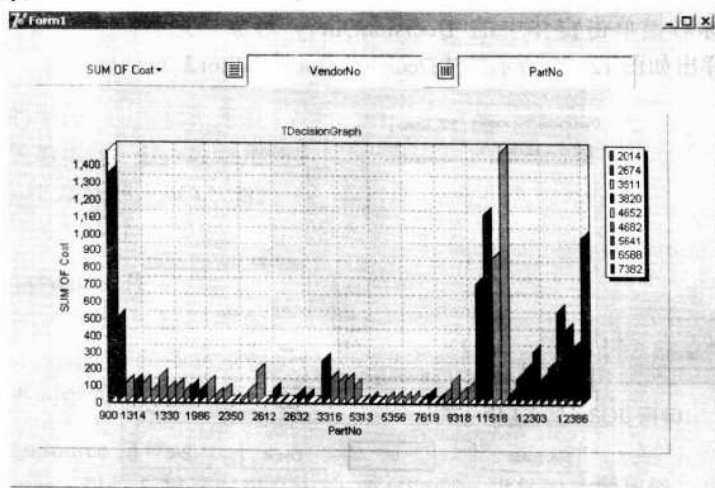


图 12-18 以 VendorNo 为组的统计效果

(9) 同时按下 Decision Pivot 上的【VendorNo】按钮和【PartNo】按钮，会得到如图 12-19 所示的柱状图。

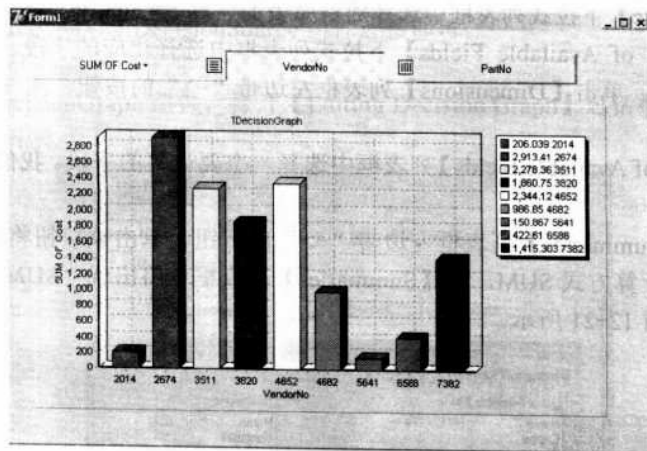


图 12-19 以 VendorNo 和 PartNo 为组的统计效果

12.2.2 统计组件的介绍

下面介绍一下统计组件。

1. DecisionQuery 组件

该组件与 BDE 组件中的 Query 组件的功能相似。SQL 是它的一个重要属性，通过 SQL 属性为 DecisionQuery 组件添加 SQL 语句。

为 DecisionQuery 组件设置 SQL 属性有两种方法：

- 在对象查看器中单击 DecisionQuery 对象的 SQL 属性右边的省略号按钮，在弹出的【String List Editor】对话框中编写 SQL 语句。
- 用鼠标右键单击窗体中的 DecisionQuery 对象，选择菜单项【Decision Query Editor...】，将弹出如图 12-20 所示的【Decision Query Editor】对话框。

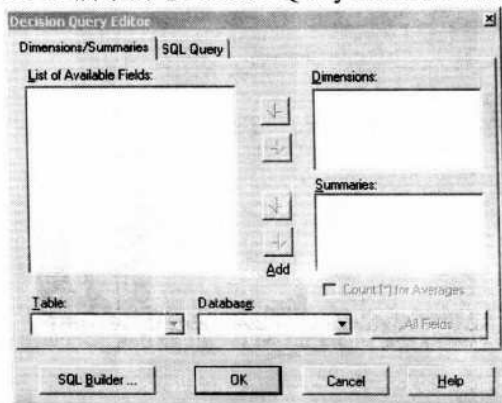


图 12-20 Decision Query Editor 对话框

在该对话框的 Dimensions/Summaries 选项中按下面的步骤设置相应字段：

- (1) 在【Database】下拉式列表框中选择数据库的别名 DBDEMOS。
 - (2) 在【Table】下拉式列表框中选择数据库表的名称 country.db。
 - (3) 在【List of Available Fields】下拉式列表框中选择相应的字段。我们选择 Name 字段和 Continent。单击【Dimensions】列表框左边带“->”的按钮，向【Dimensions】列表框中添加字段。
 - (4) 在【List of Available Fields】列表框中选择一个要计算的字段。我们选择 Population 字段。
 - (5) 单击【Summaries】列表框左边带“->”的按钮。单击该按钮将弹出一个快捷菜单，选择其中的计算方式 SUM，在【Summaries】列表框中将出现“SUM (Population)”。
- 此时的对话框如图 12-21 所示。

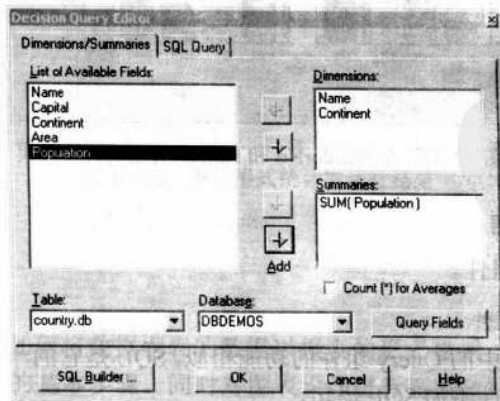


图 12-21 设置完毕的 Decision Query Editor 对话框

(6) 选择 SQL 选项, 自动生成 SQL 语句。如图 12-22 所示。

2. DecisionCube 组件

DecisionCube 组件主要用来分析数据库表中的字段, 任何数据库表都可以作为它的数据源, 但一般都用 Decision Query 作为 DecisionCube 的数据源。

3. DecisionPivot 组件

DecisionPivot 组件与数据控制组件中的 DBNavigator 组件的功能相似, 用于导航不同组的统计图。

4. DecisionSource 组件

DecisionSource 组件与数据库组件中的 DataSource 组件的功能相似, 起到连接数据集和数据控制组件的作用。

5. DecisionGraph 组件

DecisionGraph 组件用于显示与数据相对应的图形。使用 DecisionGraph 组件时, 将组件的 DecisionSource 属性与 DecisionSource 组件连接即可。

设置 DecisionGraph 组件的步骤如下:

(1) 双击 DecisionGraph 组件, 打开【Editing Decision Graph】对话框来设计图形的外观, 如图 12-23 所示。

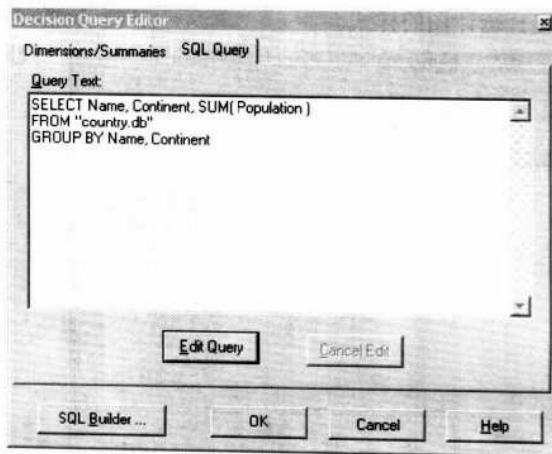


图 12-22 SQL 语句

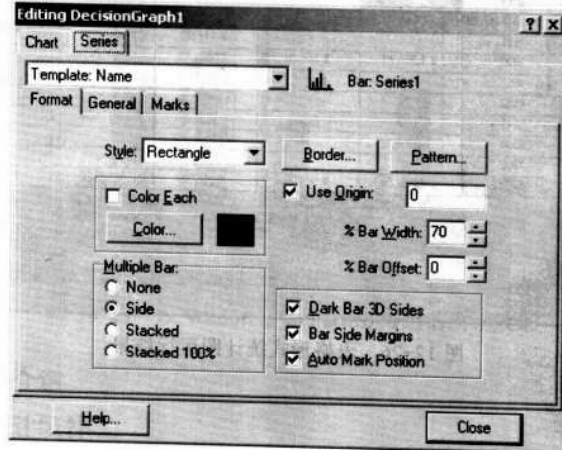


图 12-23 Editing Decision Graph 对话框

(2) 单击“Chart”标签进入【Chart】选项卡, 其中【Series】子选项用来选择不同的图形以满足用户的需要, 如图 12-24 所示。

(3) 单击【Add】按钮, 就会出现如图 12-25 所示的【TeeChart Gallery】对话框, 要

求选择统计图的样式。

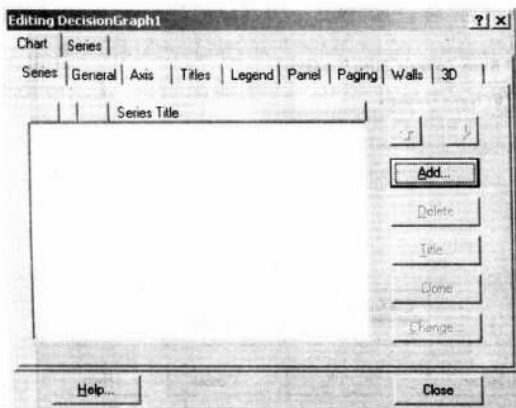


图 12-24 Chart 选项的 Series 子选项

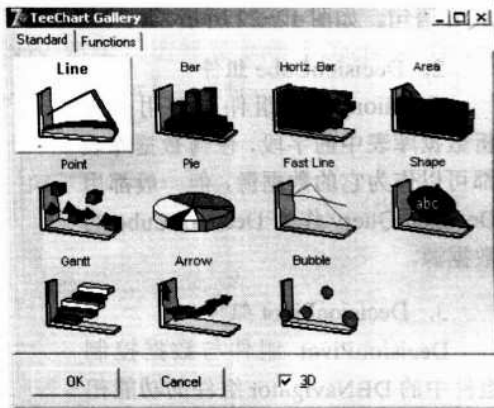


图 12-25 TeeChart Gallery 对话框

6. DecisionGrid 组件

与 DecisionGraph 组件不同的是该组件以表格形式对数据进行统计显示，可以根据需要设计表格的外观。我们将上面的例子把 DecisionGraph 组件换成 DecisionGrid 组件，结果如图 12-26 所示。

VendorNo	PartNo	900	912	1313	1314	1316
2014						
2674						
3511				¥ 117.50		¥ 119.35
3920	¥ 1,356.75	¥ 504.00				
4652						
4682						
5641					¥ 124.10	
6598						
7382						
Sum	¥ 1,356.75	¥ 504.00	¥ 117.50	¥ 124.10	¥ 119.35	

图 12-26 表格形式统计图运行结果

12.3 小结

本章主要讲述了如何将数据库表的数据以报表的形式显示、打印，并以 Rave 组件为例讲述完成报表设计的基本步骤。同时对图表设计进行简单介绍，通过完成一个简单图表，使读者对图表设计有初步的了解，更深的学习可参考其他书。

第 13 章 网络编程技术

随着互联网应用的深入发展,网络已经深入到我们社会生活的各个角落。基于网络的程序设计已经成为很重要的软件开发技术。Delphi 7 除了在数据库方面有出色表现外,也具有强大的网络应用能力。本章将简单描述 TCP/IP 编程、ISAPI 编程的基本机制和编程步骤,以及一些对这两种编程有辅助作用的 Internet 组件,使读者对网络编程有一定的了解。

13.1 网络基础知识

所谓计算机网络,是指两台或两台以上独立的计算机通过电缆(光纤、同轴电缆、双绞线、电话线,或者红外线、微波等)连接起来,可以实现相互之间的通信和资源共享的一个系统。我们常说的互联网,就是最大的计算机网络;而我们校园或办公室里的小网络,就是局域网。

对于网络上的计算机,不管它处于什么位置,如果它允许别的计算机访问自己的资源,那么它就是一台服务器(Server),而访问它的计算机就是所谓的客户机(Client)。我们常说的 C/S 编程方式,就是指基于客户机/服务器环境下的编程技术。

网络上的服务器一般可以提供以下几种网络服务:

1. FTP 服务

FTP(File Transfer Protocol, 文件传输协议)就是指文件传输服务,这种服务是一种实时的联机服务。在应用时,客户机首先需要连接到 FTP 服务器上,通过提供用户名和密码进行登录。登录完成后,客户机就可以浏览服务器上的 FTP 服务目录,并进行文件的上传和下载。

2. WWW 服务

WWW(World Wide Web, 万维网)是提供一种超媒体形式的网络信息服务。客户机要获得 WWW 服务器上的信息资源,需要使用浏览器软件(如 Microsoft 等 IE 浏览器等)访问服务器,对服务器提供的 HTML 格式的数据进行解读,并将具体的信息页面呈现给读者。目前,WWW 服务是网络上应用最为广泛的技术,几乎所有的信息都可以通过 WWW 服务提供。

3. 电子邮件服务

电子邮件也是网络上应用非常广泛的一种服务。电子邮件服务器一般包括接收邮件服

务器和发送邮件服务器,前者使用的是 POP3 协议,后者使用的是 SMTP 协议。一般情况下,接收邮件服务器和发送邮件服务器都是由一台计算机来承担。客户需要发送电子邮件,必须连接到电子邮件服务器。

当然,网络服务远不止这些,还有许多其他服务类型,如 Gopher 服务、远程登录服务等,我们就不一一介绍了。实际上,并非一个服务器就是一台计算机,在一些较小型的网络上,往往一台计算机要承担好几个角色,如既是 WWW 服务器,也是 FTP 服务器;而在一些大型网络上,往往要多台计算机共同承担一项服务。从另外一个角度来说,一台计算机既可以充当服务器,同时又可以充当客户机。

13.2 TCP/IP 编程

TCP/IP (传输控制协议/互联协议)是在 Internet 上传输数据时使用的网络协议,以保证数据的正确传输。常用的网络服务,如 Telnet、FTP 和 HTTP 等服务都是使用 TCP/IP 协议完成的,本节介绍利用 TCP/IP 协议来传输数据。

在 Delphi 中使用 TCP/IP 编程 TCP/IP 协议可以建立两个终端的连接,并让数据在其间正确的传输。为保证数据传输的正确性,不丢失或错传数据,采用了许多复杂的机制。在 TCP/IP 协议中,每个终端有一个 IP 地址,终端上使用 TCP/IP 的应用程序有一个端口号。使用不同的 IP 地址区分不同的终端,而在同一个终端上根据不同的端口号将数据送到相应的应用程序中。

Delphi 7 提供了一些基于 TCP/IP 编程的网络组件,使用它们可以开发出功能强大的网络应用程序。

13.2.1 TCPServer 组件和 TCPClient 组件

TCPServer 组件主要是用来实现服务器端程序的组件,而 TCPClient 组件主要用于实现客户端程序。这两个组件都是继承自 TSocket 对象,因此有许多相同的属性。下面我们首先来了解这两个组件的基本属性和方法。

1. 常用属性

(1) BlockMode: 用来设定网络的阻塞模式,它有 3 个选项:

- bmBlocking: 使用阻塞模式。
- bmNonBlocking: 使用非阻塞模式。
- bmThreadBlocking: 使用线程阻塞模式。

(2) LocalHost: 该属性用来定义本地计算机的 IP 地址。

(3) LocalPort: 该属性用来定义本地计算机的端口号。

(4) RemoteHost: 该属性用来定义远程计算机的 IP 地址。

(5) RemotePort: 该属性用来定义远程计算机的端口号。

(6) Active: 该属性用来判断服务程序是否处于激活状态。

2. 常用方法

(1) Accept: 该方法的原型如下:



```
function Accept :Boolean;
```

当 TCPServer 组件接收 TCPClient 组件的消息时, 将自动调用 Accept 方法。如果接收成功, 返回 True; 否则返回 False。

(2) Close: 该方法用来关闭 Socket 连接。

(3) LocalDomainName: 该函数原型为:

```
function LocalDomainName: string;
```

使用该函数可以取得当前计算机的域名。

下面的代码说明了该函数的使用方法:

```
Procedure TForm1.Button1Click(Sender:TObject);
begin
    ShowMessage(TcpServer1. LocalDomainName);
End;
```

(4) LocalHostAddr: 该函数可以取得当前计算机的 IP 地址。

下面的代码说明了该函数的使用方法:

```
Procedure TForm1.Button1Click(Sender:TObject);
begin
    ShowMessage(TcpServer1. LocalHostAddr);
End;
```

(5) LocalHostName: 该函数可以取得当前计算机在网络上的名称。

(6) LookupHostAddr: 函数原型为:

```
function LookupHostAddr(const hn:string):string;
```

该函数可以查找到指定的域名所对应的 IP 地址。

下面的代码说明了该函数的使用方法:

```
Procedure TForm1.Button1Click(Sender:TObject);
begin
    ShowMessage(TcpClient1. LookupHostAddr('borland.com'));
End;
```

(7) LookupHostName: 函数原型为:

```
function LookupHostName (const ipaddr:string):TsocketHost;
```

该函数可以根据 IP 地址取得其相应的域名。

3. TCPServer 组件的常用事件

(1) OnAccept: 事件的原型为:

```
property OnAccept:TsocketAcceptEvent;
```

当 TCPServer 组件接收到信息时触发该事件, 可以在该事件中对信息进行处理。

(2) OnListening: 事件的原型为:

```
property OnListening:TnotifyEvent;
```

当程序开始监听时触发该事件。

4. TCPClient 组件的常用事件

(1) OnConnect: 事件原型为:

property OnConnect: TSocketNotifyEvent;

当客户端连接服务器时触发该事件。

(2) OnDisconnect: 事件原型为:

property OnDisconnect: TSocketNotifyEvent;

当客户端断开与服务器的连接时触发该事件。

13.2.2 网络聊天室

通过 TcpServer 组件和 TcpClient 组件, 可以很方便地管理一个基于 TCP/IP 网络程序的服务器和客户机之间的 Socket 连接。通过调用组件所提供的内部方法, 能够在已经建立的 Socket 连接的一端写入数据, 并在另一端读出这些数据。下面我们通过一个实例来说明这种应用。

[例 13-1] 网络聊天室

在这个聊天室程序中, 首先需要对服务器(对方计算机)的地址、端口和本地通信的端口进行设置; 然后就可以同对方进行聊天了; 同时, 当前的 IP 地址、连接到的服务器地址以及发送的字符数, 都能够显示在窗体的状态栏上。程序效果如图 13-1 所示。

(1) 创建一个新的应用程序。

(2) 首先在窗体上添加 1 个 StatusBar 组件、1 个 Label 组件、1 个 Button 组件, 2 个 Memo 组件和 1 个 GroupBox 组件。

(3) 选择该 GroupBox 组件, 在其中再添加 3 个 Label 组件、3 个 Edit 组件和一个 Button 组件, 并将各组件按照程序要求布置。

(4) 再在窗体中添加 1 个 TcpClient 组件和 1 个 TcpServer 组件, 如图 13-2 所示。



图 13-1 网络聊天室

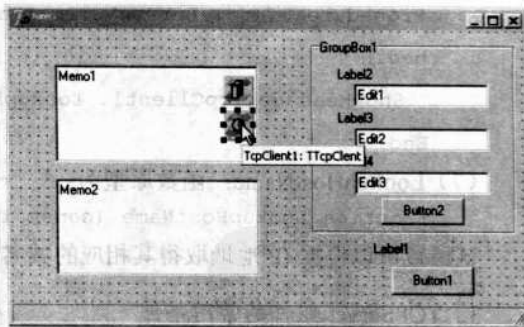


图 13-2 在窗体中添加组件

(5) 为各组件设置属性, 如表 13-1 所示。

(6) 选中 InfoMemo 组件, 再选择其【Lines】属性, 打开【String List Editor】对话框, 将其中的内容清空, 如图 13-3 所示。这样, 初始状态下, 组件窗口中就不会显示内容。

(7) 同理将 SendMemo 中的内容也清空。

表 13-1 各组件属性设置

组件类型	组件名称	组件属性	属性值	属性意义
Memo	InfoMemo	Align	alTop	组件窗口充满窗体上部
		Color	BtnFace	Memo 窗口颜色
		ReadOnly	True	文本内容为只读
		ScRollBars	ssVertical	显示垂直滚动条
	SendMemo	Lines		不在 Memo 窗口中显示内容
		WordWrap	False	自动加入回车符
GroupBox	GroupBox1	Lines		不在 Memo 窗口中显示内容
		Caption	设置	分组框的标题
Label	Label1	Caption	零点聊天室	说明信息
	Label2	Caption	服务器地址	
	Label3	Caption	服务器端口	
	Label4	Caption	本地端口	
Edit	Edit1	Text	127.0.0.1	默认情况下服务器为本机
	Edit2	Text	10	定义服务器端口号
	Edit3	Text	10	定义本地端口号
Button	SetButton	Caption	聊天室设置	设置服务器的地址和端口
	SendButton	Caption	发送	发送聊天信息
StatusBar	StatusBar1	SimplePanel	False	要将状态栏分为 3 个部分
TcpServer	TcpServer1	BlockMode	bmThreadBlocking	自动为每一个 Socket 连接建立一个线程
TcpClient	TcpClient1	BlockMode	bmBlocking	使用阻塞模式

注意: SendMemo 组件的【WordWrap】属性必须设置为 False, 即去掉文本内容中的回车符。否则, 一次只能发送一行文字。这是因为在一次 Socket 连接中, 函数 ReceiveIn 只能接收一次以“\r\n”结尾的字符串, 即遇到回车符就不再读入信息了。

(8) 选中 StatusBar1 组件, 再选择其【Panels】属性, 打开【Editing StatusBar1 Panels】对话框, 为其添加 3 个面板项, 如图 13-4 所示。并分别设置 3 个面板项的【Width】属性为 150、150、130。

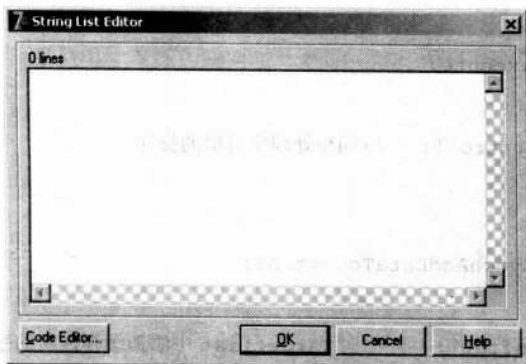


图 13-3 将 InfoMemo 组件中的内容清空

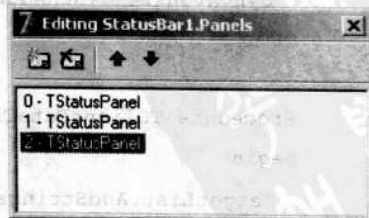


图 13-4 为 StatusBar1 组件添加 3 个面板项

(9) 现在需要定义一个线程类型 `TclientDatathread`, 用来实现向 `InfoMemo` 组件异步写入交谈双方的内容。类型的声明如下:

```
type
  TClientDataThread=class (TThread)           //对每个客户端都有一个线程在运行
  private
  public
    ListBuffer:tStringList;                   //缓存区,缓存远程发送的当前聊天信息
    TargetList:tStrings;                      //聊天的信息存放
    Procedure SynchAddDataToControl;          //同步执行过程
    Constructor Create(CreateSuspended:Boolean); //构造函数
    Procedure Execute;override;              //重载线程类的执行函数
    Procedure Terminate;                      //用于结束线程,释放资源
  end;
```

(10) 对于定义的线程类型, 需要将其中的方法具体实现:

```
Constructor TClientDataThread.Create(CreateSuspended:Boolean);
begin
  inherited Create(CreateSuspended);
  FreeOnTerminate:=True;                      //线程结束时自动释放资源
  ListBuffer:=TStringList.Create;             //创建字符串列表变量,作为缓存区
end;

Procedure TClientDataThread.Terminate;
begin
  ListBuffer.Free;                            //释放字符串列表所占资源,即释放缓存区
  inherited;
end;

Procedure TClientDataThread.execute;
begin
  Synchronize(SynchAddDataToControl);        //同步处理字符串的操作
end;

Procedure TClientDataThread.SynchAddDataToControl;
begin
  TargetList.AddStrings(ListBuffer);          //向 TargetList 中添加缓存区新的字符串
end;
```

(11) 为了获得用户当前机器的 IP 地址, 需要在主窗体的 OnCreate 事件中添加如下代码:

```
Procedure TForm1.FormCreate(Sender: TObject);
var
  HostName, HostIp: string;
begin
  Hostname:=TcpServer1.LookupHostName('127.0.0.1'); //获得主机名称
  HostIp:=TcpServer1.LookupHostAddr(HostName); //获得主机 IP 地址
  StatusBar1.Panels[0].Text:='您的 IP: '+HostIP; //在状态栏第一项
                                                    中显示 IP
end;
```

(12) 在程序开始的时候, 首先要设置服务器的参数。在【服务器设置】按钮的 OnClick 事件中输入如下代码:

```
Procedure TForm1.SetButtonClick(Sender: TObject);
begin
  TcpServer1.LocalPort:=ServerPort.Text; //设置服务器端口
  TcpServer1.Active:=True; //激活服务器
  SendMemo.SetFocus; //将焦点定位于信息输入区域
end;
```

(13) 为【发送】按钮添加如下代码, 用以发送交谈内容:

```
Procedure TForm1.SendButtonClick(Sender: TObject);
var
  i: integer;
begin
  TcpClient1.RemoteHost:=ServerAddr.Text; //设置对方主机名
  TcpClient1.RemotePort:=ServerPort.Text; //对方主机端口号
  try
    if TcpClient1.Connect then //如果已经连接
      for i:=0 to SendMemo.Lines.Count-1 do
        TcpClient1.Sendln(SendMemo.Lines[i]); //发送信息输入区域的文本
      finally
        TcpClient1.Disconnect; //信息发送完毕, 暂时断开
                                Socket 连接
      end;
    SendMemo.SetFocus; //将焦点定位于信息输入区域
  end;
```

(14) TcpServer 接收到 Socket 连接请求时, 自动触发 OnAccept 事件。下面为 OnAccept 事件添加代码, 将接收到的信息增加到 InfoMemo 组件中, 并将连接情况显示在状态栏的第二项。

```

Procedure TForm1.TcpServer1Accept(Sender:TObject;
  ClientSocket: TCustomIpClient);
var
  s:string;
  DataThread: TclientDatathread;
begin
  //本地服务器探测到新的消息, 开始创建一个新的线程
  DataThread:=TclientDatathread.Create(True);
  //在信息显示框中显示对方的信息
  DataThread.TargetList:=InfoMemo.Lines;
  //在状态栏的第二项中显示对方的主机名和 IP 地址
  StatusBar1.Panels[1].Text:='连接到'
    +ClientSocket.LookupHostName(ClientSocket.RemoteHost)
    + ' ('+ClientSocket.RemoteHost+')';
  //接收对方消息, 并保存到缓冲区 ListBuffer 中
  DataThread.ListBuffer.add('==== begin message ====');
  s:=ClientSocket.ReceiveIn;
  while s<>' ' do
    begin
      DataThread.ListBuffer.Add(s);
      s:=ClientSocket.ReceiveIn;
    end;
  DataThread.ListBuffer.Add('==== end of message ====');
  //继续执行线程, 把 ListBuffer 中的人写入 TargetList, 亦即 InfoMemo 组件
  DataThread.Resume;
end;

```

(15) 为了显示 Socket 连接的状态, 需要在 TcpClient 组件的 OnSend 事件中, 向状态栏输出提示信息:

```

Procedure TForm1.TcpClient1Send(Sender: TObject;Buf: PAnsiChar;
  var DataLen: Integer);
begin
  StatusBar1.Panels[2].Text:='发送信息'+IntToStr(DataLen)+'个字符';
end;

```

(16) 运行程序, 对于服务器的设置可以采用默认值, 直接单击【聊天室设置】按钮, 然后就可以在本地实现聊天功能。

如果在两台不同的计算机上运行我们的聊天室程序, 就需要分别设置不同的“服务器地址”(对方的计算机 IP 地址), 端口号可以使用默认值。在单击【聊天室设置】按钮后, 就可以开始聊天, 一方的信息就能够发送到另外一台机器上了, 如图 13-5 所示。



图 13-5 在两台不同的计算机上聊天

注意: 如果某台计算机安装了防火墙, 需要设置防火墙允许程序访问网络。

这里涉及到一个线程的概念, 我们在此简单介绍一下。

由于 Windows 系统是一个多任务的操作系统, 因此设计 Windows 程序就会使用到线程和进程的概念。所谓进程, 是指应用程序的执行实例, 每个进程是由私有的虚拟地址空间、代码、数据和其他各种系统资源组成的。进程在运行过程中窗间资源随着进程的终止而被销毁, 所使用的系统资源在进程终止时被释放或关闭。

线程是进程内部的一个执行单元 (如可以是一个函数或一个活跃的对象等)。系统创建好进程后, 实际上就启动执行了该进程的主执行线程。主执行线程终止了, 进程也就随之终止。

每一个进程至少有一个线程 (即主执行线程, 它无需由用户去主动创建, 是由系统将应用程序启动后创建的), 用户根据需要在应用程序中创建其他线程, 多个线程并发地运行在同一个进程中。一个进程中所有线程都在该进程的虚拟地址空间中, 使用这些虚拟地址空间、全局变量和系统资源, 所以线程之间的通信要比进程之间的通信容易得多。多线程设计在实际中使用很广泛。

由于多线程程序的设计相对比较复杂, 本书没有详细讲解。有兴趣的读者可以参考其他高级编程方面的书籍。

13.2.3 Web 浏览器

WebBrowser 组件封装了 TCP/IP 协议, 可以用于网页的浏览, 它也位于【Internet】选项卡上。常用的方法有:

- 【Navigate】: 浏览网页。
- 【Stop】: 停止当前页面的下载操作。
- 【Refresh】: 刷新网页。
- 【GoHome】: 返回主页。
- 【GoBack】: 回退到前一个页面。
- 【GoForward】: 前进到下一个页面。

下面我们利用 WebBrowser 组件来设计一个简单的 Web 浏览器。

[例 13-2] 我的 Web 浏览器

在这个简单的浏览器中，可以通过输入地址来打开互联网站点；具有浏览器基本的功能按钮，能够实现在页面之间的导航。程序效果如图 13-6 所示。

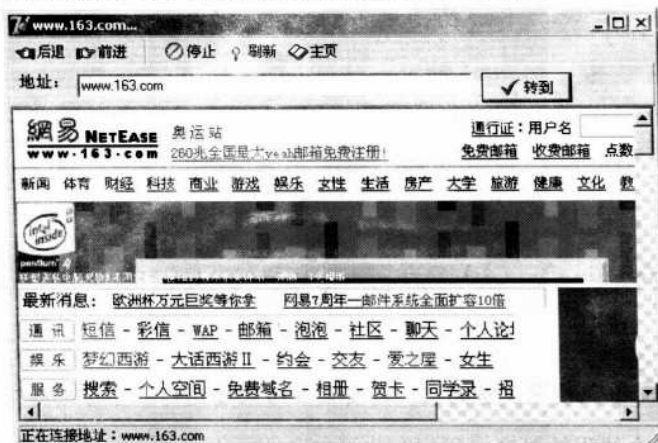


图 13-6 我的 Web 浏览器

- (1) 创建一个新的应用程序。
- (2) 定义窗体的【Caption】属性为“我的 Web 浏览器”。
- (3) 在窗体上添加 1 个 ToolBar 组件、1 个 Panel 组件、1 个 StatusBar 组件和 1 个 ImageList 组件，再添加一个 WebBrowser 组件。
- (4) 双击 ImageList 组件，打开【ImageList】窗口，在其中添加 5 个图标，如图 13-7 所示。
- (5) 选择 ToolBar1 组件，设置其【Images】属性为“ImageList1”，这样工具条上的按钮就能够使用 ImageList 中引入的图标；设置【ShowCaption】属性为“True”，这样按钮上就能够显示文字；设置【List】属性为“True”，这样文字和图标就并排排列。

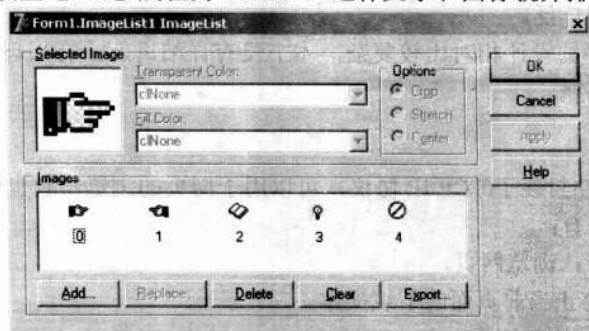


图 13-7 在 ImageList 组件中添加 5 个图标

- (6) 使用右键快捷菜单向 ToolBar1 组件中添加 6 个按钮，并设置 ToolButton3 按钮的【Style】属性为“tbsDivider”，使其作为一个分隔线。
- (7) 为各个按钮定义适当的文字，并在【ImageIndex】属性设置相应的图像索引值，使按钮显示出对应的图像，如图 13-8 所示。

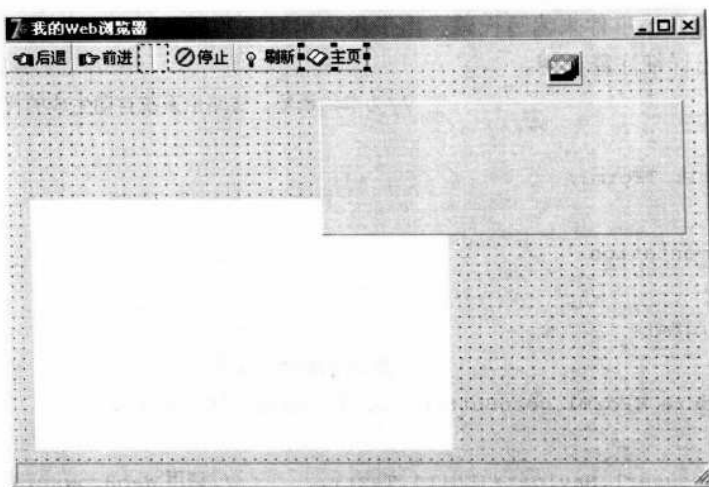


图 13-8 为各个按钮定义适当的文字和图像

(8) 为各按钮定义【Name】属性，要使按钮名称能够反映其功能，如 backButton、forwardButton、homeButton 等。

(9) 选择 Panel1 组件，设置其【Align】属性为“alTop”。在其中添加 1 个 Label 组件，1 个 Edit 组件和 1 个 BitBtn 组件。

(10) 为 BitBtn1 组件定义【Name】属性为“gotoButton”，【Caption】属性为“转到”；在【Glyph】属性中引入一个图标；设置【Default】属性为“True”，这样，它就可以直接响应 Enter 键按下的事件了。

(11) 选择 WebBrowser1 组件，设置其【Align】属性为“alClient”，使页面窗口能够充满整个窗体。如图 13-9 所示。

(12) 选择 StatusBar1 组件，设置其【SimplePanel】属性为“True”。

至此，程序界面设计完成，窗体上各个组件之间的关系可以通过【Object TreeView】窗口表现出来，如图 13-10 所示。

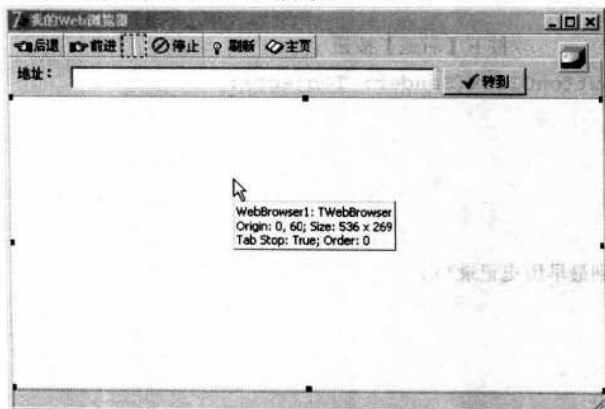


图 13-9 程序界面设计

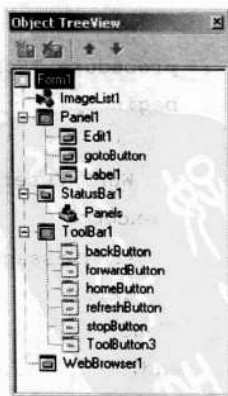


图 13-10 窗体上各个组件之间的关系

下面需要为各个事件来编写代码。由于代码相对比较简单，我们在此直接给出全部事件代码，并给出详细注释说明。

```
... ..                                //为节约篇幅，省略由系统自动生成的代码
var
    Form1: TForm1;

implementation

{$R *.dfm}

//按下【转到】按钮
procedure TForm1.gotoButtonClick(Sender: TObject);
begin
    WebBrowser1.Navigate(Edit1.Text);      //调用 WebBrowser 的浏览网页功能
end;

//连接网页过程中，标题栏和状态栏同步显示连接状态
procedure TForm1.WebBrowser1DownloadBegin(Sender: TObject);
begin
    Form1.Caption:=Edit1.Text+'...';
    StatusBar1.SimpleText:='正在连接地址: '+Edit1.Text;
end;

//网页连接完成后，在标题栏和状态栏显示连接到的地址
procedure TForm1.WebBrowser1DownloadComplete(Sender: TObject);
begin
    Form1.Caption:=WebBrowser1.LocationURL;
    StatusBar1.SimpleText:='完成'+WebBrowser1.LocationURL;
end;

//按下【后退】按钮
procedure TForm1.backButtonClick(Sender: TObject);
begin
    try
        WebBrowser1.GoBack;
    except
        showMessage('已经达到最早历史记录');
        exit;
    end;
end;

//按下【前进】按钮
```




```

procedure TForm1.forwardButtonClick(Sender: TObject);
begin
    try
        WebBrowser1.GoForward;
    except
        showMessage('已经达到最新历史记录');
        exit;
    end;
end;

//按下【停止】按钮
procedure TForm1.stopButtonClick(Sender: TObject);
begin
    WebBrowser1.Stop;
end;

//按下【刷新】按钮
procedure TForm1.refreshButtonClick(Sender: TObject);
begin
    WebBrowser1.Refresh;
end;

//按下【主页】按钮
procedure TForm1.homeButtonClick(Sender: TObject);
begin
    WebBrowser1.GoHome;
    Edit1.Text:='about:blank';
end;

end.

```

(13) 运行程序，我们就可以用自己的浏览器来观看网页了。

13.2.4 Ping 命令检查联网状态

Ping 命令是 Dos、Unix、Linux 等操作系统下检查联网状态最简单和常用的操作。Ping 操作主要是测试一帧数据从一台主机传输到另一台主机所需要的时间，从而判断网络的线路状况。

Delphi 7 提供了一个 IdcmpClient 组件，能够实现这种 Ping 操作，调用该组件的 Ping 方法就可以对指定主机进行连接，在指定时间内触发 Reply 事件，在该事件中可以显示出远程主机的状态。IdcmpClient 组件位于【Indy Clients】选项卡上。

下面我们利用该组件来制作一个程序，实现对远程主机的 Ping 操作。

【例 13-3】 Ping 命令检查联网状态

在目的 IP 栏中输入一个 IP 地址，然后单击【Ping】按钮，就会在下面的 ListBox 中显示出 Ping 操作的反馈情况；并且，在进行 Ping 操作的过程中，按钮不可用。程序效果如图 13-11 所示。

(1) 创建一个新的应用程序，定义窗体【Caption】为“Ping 命令检查联网状态”。

(2) 向窗体中添加 1 个 Bevel 组件、1 个 Label 组件、1 个 Edit 组件、1 个 Button 组件和 1 个 ListBox 组件，再加入 1 个 IdlcmpClient 组件，调整各组件的位置如图 13-12 所示。



图 13-11 Ping 命令检查联网状态

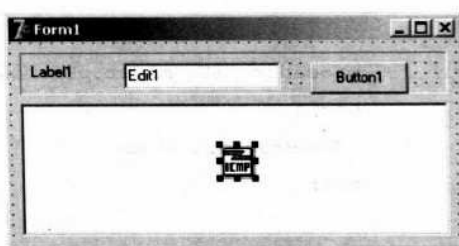


图 13-12 向窗体中添加组件

(3) 修改各组件的属性，使其符合程序需要。

(4) 定义 Button1 组件的【Name】属性为“PingButton”，定义 IdlcmpClient1 组件的【Name】属性为“ICMP”。

(5) 在“PingButton”的 OnClick 事件中添加如下代码，以启动 IdlcmpClient1 组件，利用其 Host 属性连接到指定的主机。

```
procedure TForm1.PingButtonClick(Sender: TObject);
var
    i: integer;
begin
    ICMP.OnReply:=ICMPReply;           //设置回应属性
    ICMP.ReceiveTimeout:=1000;         //设置连接时间
    PingButton.Enabled:=False;         //将按钮设置为不可用状态
    Try                                //尝试连接到指定的主机
        ICMP.Host:=Edit1.Text;
        for i:=1 to 4 do
        begin
            ICMP.Ping;                 //开始 Ping 操作
            Application.ProcessMessages; //发送消息
        end;
    finally
        pingButton.Enabled:=true;     //将按钮设置为可用状态
    end;
end;
```

```
end;
```

```
end;
```

(6) 在 `IdIcmpClient1` 组件的 `OnReply` 事件中添加如下代码, 判断服务器的状态。

```
procedure TForm1.ICMPReply(ASender: TComponent;
                           const AReplyStatus: TReplyStatus);

var
    sTime:string;
begin
    if(AReplyStatus.MsRoundTripTime=0) then
        sTime:='<1'
    else
        sTime:='=';
    listbox1.Items.Add(Format('%d byte from %s:icmp_seq=%d TTL=%d
                              time%sdms', [AReplyStatus.BytesReceived,
                              AReplyStatus.FromIpAddress, AReplyStatus.SequenceId,
                              AReplyStatus.TimeToLive, sTime, AReplyStatus.MsRoundTripTime]));
end;
```

(7) 运行程序, 在地址栏中输入一个 IP 地址, 然后单击【Ping】按钮, 就能够看到网络连接状况。

提示: 很多计算机安装了防火墙, 它会拒绝别的机器对主机的 Ping 操作。所以如果你要连接的主机有防火墙, 可以暂时先把它关闭。

当我们连接远程主机时, 如果无法 Ping 通, 就会出现如图 13-13a 所示的情况, 得不到对方主机的反馈; 而如果正常 Ping 通的话, 就会显示如图 13-13b 所示的情况。

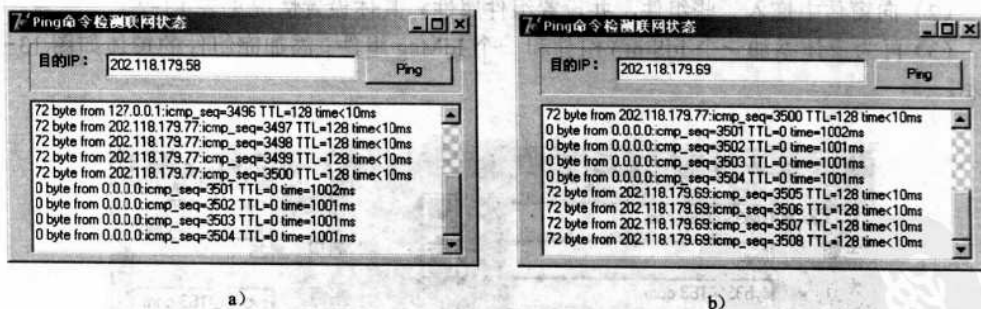


图 13-13 Ping 操作不通和通的情况

13.2.5 发送电子邮件

邮件是我们网络上应用最广泛的服务项目, 一般情况下我们都使用网络服务商提供的邮件收发程序来收发邮件。Delphi 7 提供的 `IdSmtp` 组件、`IdMessage` 组件, 使我们可以自己设计一个发送邮件的程序。这两个组件都位于【Indy Client】选项卡中。

程序主要用到的是 IdSmtpServer 组件，其主要属性有：

- Host: SMTP 服务器的地址;
- Port: SMTP 服务器的端口;
- Username: 用户邮箱的帐号;
- Password: 用户帐号的密码;
- Connect: 连接服务器事件;
- Disconnect: 断开与服务器的连接。

下面我们来设计这个程序。

【例 13-4】通过 SMTP 服务器发送邮件

填写上相关的邮件服务器信息、个人帐号信息，然后编辑自己的邮件，单击【发送邮件】按钮，就能够将信息发送到对方的邮箱。程序界面和执行情况如图 13-14 所示。



图 13-14 通过 SMTP 服务器发送邮件

- (1) 创建一个新的应用程序。
- (2) 向窗体中拖入一些组件，并设置组件属性。具体设置情况见表 13-2。
- (3) 再为窗体添加一个 IdSmtp 组件和一个 IdMsg 组件。添加完组件的窗体如图 13-15 所示。

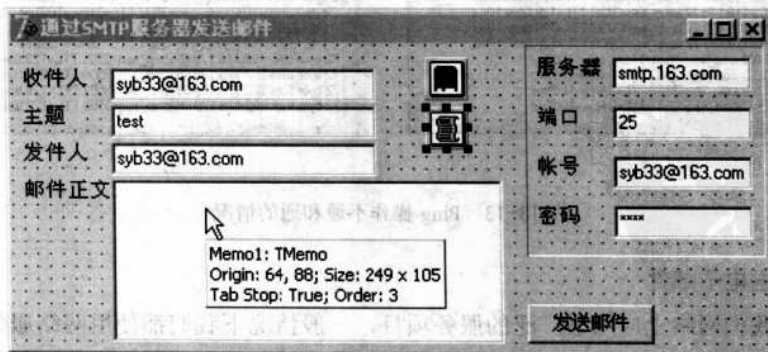


图 13-15 向窗体中添加组件

表 13-2 窗体中组件的属性

组件类型	组件名称	组件属性	属性值	属性意义
Form	Form1	Caption	通过 SMTP 服务器发送邮件	窗体标题
Bevel	Bevel1			
Label	Label1	Caption	收件人	标签上显示的文字
	Label2		主题	
	Label3		发件人	
	Label4		邮件正文	
	Label5		服务器	
	Label6		端口	
	Label7		帐号	
	Label8		密码	
Edit	Edit1	Text		对应“收件人”
	Edit2			对应“主题”
	Edit3			对应“发件人”
	Edit4			对应“服务器”
	Edit5		25	对应“端口”
	Edit6			对应“帐号”
	Edit7	Text		对应“密码”
		PasswordChar	*	保护密码内容
Memo	Memo1	Lines		对应“邮件正文”
Button	Button1	Caption	发送邮件	发送邮件的按钮

(4) 下面为程序添加代码, 其实程序中只有一个事件, 那就是【发送邮件】按钮的 OnClick 事件。为了方便读者学习, 我们给出详细的代码:

```

... .. //为节约篇幅, 省略由系统自动创建的代码
var
    Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject); //按钮的单击事件
begin
    //定义程序与 SMTP 服务器的连接类型: atLogin 为用户密码登录, atnone 为匿名登录
    SMTP1.AuthenticationType := atLogin;
    SMTP1.username:= edit6.text;           //用户帐号
    SMTP1.Password := edit7.text;         //用户密码
    SMTP1.Host := edit4.text;             //服务器地址

```

```

SMTP1.Port := StrToInt(edit5.text);           //服务器端口号
try
    SMTP1.Connect;                           //连接服务器
except
    Showmessage('连接 SMTP 服务器失败!');
    Exit;
end;

try                                           //顺利连接上
    with IdMsg do                             //传送信件基本信息
    begin
        body.Clear;                          //清空正文原始内容
        Body.Assign(memol.lines);            //设置发信者地址
        From.address := edit3.text;          //发信人的地址
        Recipients.EmailAddresses := edit1.text; //收件人地址
        Subject:=edit2.text                 //邮件主题
    end;
    SMTP1.Send(IdMsg);                       //发送邮件
finally
    showmessage('您的信件已成功发送');
    SMTP1.Disconnect;                       //发送完成后，断开连接
end;
end.

```

(5) 运行程序，我们就可以使用它给自己发一封邮件了。

如果不能连接到服务器，就会出现一个错误信息，说明连接失败。如图 13-16 所示。

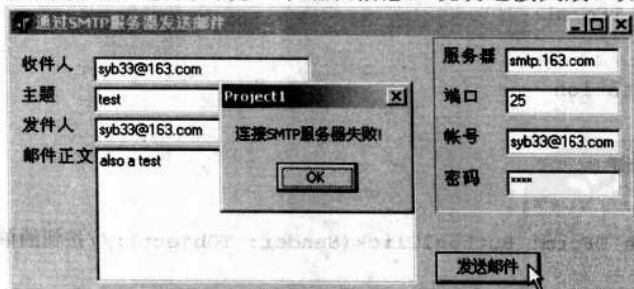


图 13-16 不能连接到服务器的情况

13.3 使用网络函数编程

虽然 Delphi 7 提供了许多组件来完成网络功能，但是通过调用 API 函数，可以提供更

加灵活和丰富的网络服务。

下面我们通过几个实例来说明如何调用 API 函数来完成特定的网络功能。有些函数、属性或方法比较难以理解，大家只需要知道其基本用法，能够完成练习就可以了。

13.3.1 获取主机 MAC 地址

网络上的每台计算机都拥有其唯一的 MAC 硬件地址，这常常是我们在软件中标识计算机的重要手段。我们可以通过程序来直接获得计算机的 MAC 地址。

在程序中，要用到下面的知识：

- nb30.pas 单元：支持 NetBios3.0 协议的标准单元文件。
- NetBIOS 函数：存在于动态链接库 netAPI32.dll 中，能够查询 MAC 地址。

[例 13-5] 获取主机 MAC 地址

单击按钮，就可以获得当前计算机的 MAC 地址，如图 13-17 所示。

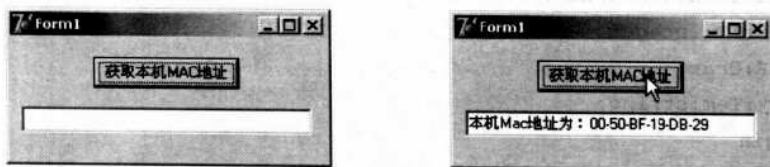


图 13-17 获取主机 MAC 地址

- (1) 创建一个新的应用程序。
- (2) 向窗体中添加一个 Button 组件和一个 Edit 组件。
- (3) 定义 Button1 组件的【Caption】属性为“获取本机 MAC 地址”。
- (4) 下面是程序完整的代码：

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, nb30; //需要将 nb30 单元包含进来

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    function NBGetAdapterAddress(a: integer): String; //创建自定义函数
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```

var
    Form1: TForm1;

implementation
{$R *.dfm}

function TForm1.NBGetAdapterAddress(a:integer):string; //函数的实现
var
    Nc:TNcb;
    Adapte:TAdapterStatus;
    LanaEnu:TLanaEnum;
    intID:integer;
    cR:Char;
    strTem:string;
begin
    result:='';
    try
        zeroMemory(@Nc,SizeOf(Nc)); //适配器清零
        NC.ncb_command:=chr(NcbEnum); //执行 Enum 命令
        cR:=NetBios(@NC);
        NC.ncb_buffer:=@LanaEnu;
        NC.ncb_length:=SizeOf(LanaEnu);
        cR:=NetBios(@NC);
        if Ord(cR)<>0 then
            exit;

        ZeroMemory(@NC,sizeof(NC)); //整个适配器参数重置
        NC.ncb_command:=chr(NCBReset); //适配器清零
        NC.ncb_lana_num:=lanaenu.lana[a];
        cR:=NetBios(@NC);
        if Ord(cR)<>0 then
            exit;

        ZeroMemory(@NC,sizeof(NC));
        NC.ncb_command:=chr(ncbastat);
        NC.ncb_lana_num:=lanaenu.lana[a];
        strcpy(nc.ncb_callname,'');
        nc.ncb_buffer:=@adapte;
    end;
end;

```

//查询 MAC 过程


```

nc.ncb_length:=sizeof(adapte);
cR:=NetBios(@NC); //得到 MAC 地址
StrTem:='';
for intId:=0 to 5 do //输出适配器地址,一共 6*2=12
begin

StrTem:=StrTem+IntToHex(integer(adapte.adapter_address[intId]),2);
if intId<>5 then StrTem:=StrTem+'-'; //在每个 16 位值之间插入一短线
end;
result:=StrTem;
finally
end;
end;

procedure TForm1.Button1Click(Sender: TObject); //按钮单击事件
var
i:integer;
s:string;
begin
for i:=1 to 20 do //循环查询计算机中前
//若干个适配器
begin
s:= NBGetAdapterAddress(i); //得到适配器的 MAC 地址
if s='' then //未查到相关适配器的信息
Edit1.Text:=''
else
begin
Edit1.Text:='本机 Mac 地址为: '+s; //输出地址信息
exit;
end;
end;
Edit1.Text:='您的机器没有安装网络适配器';
end;
end.

```

(5) 运行程序, 单击按钮, 就能够得到本机的 MAC 地址。

13.3.2 信使服务

信使服务就是向指定计算机发送一个字符串消息, 该计算机将弹出带有文字消息的对话框。该功能使我们能够在不同的机器上互通消息。

信使服务实际上就是 Windows 9x 下的 WinPopup.exe 的功能, Windows2000 将该功能

集成到了 NetAPI32.dll 中, 由 NetMessageBufferSenf 函数来实现。下面我们就来使用这个函数实现信使服务的功能。

【例 13-6】信使服务

在地址栏中输入需要发送信息的主机 IP 地址, 编辑信息内容, 然后单击【发送】按钮, 就可以将信息发送到设定的主机。在该主机上将弹出一个【信使服务】消息框, 如图 13-18 所示。

- (1) 创建一个新的应用程序。
- (2) 向窗体中拖入一些组件, 并设置组件属性。具体设置情况见表 13-3。
- (3) 添加完组件的窗体如图 13-19 所示。

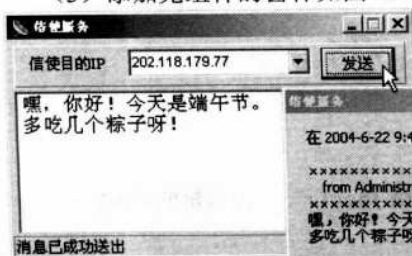


图 13-18 信使服务

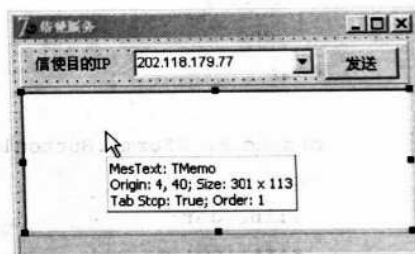


图 13-19 向窗体中添加组件

表 13-3 窗体中组件的属性

组件类型	组件名称	组件属性	属性值	属性意义
Form	FrmMain	Caption	信使服务	窗体标题
Bevel	Bevel1			
Label	Label1	Caption	信使目的 IP	标签上显示的文字
ComboBox	AddrBox	Text	202.118.179.77	默认为本机 IP 地址
Button	BtnSend	Caption	发送	发送信息的按钮
Memo	MsgText	Lines		清空初始信息内容
StatusBar	StatusBar1	SimplePanels	True	定义为单面板

(4) 从菜单中选择【File】/【New】/【Unit】命令, 创建一个新的单元 Unit2, 在其中编写如下代码, 创建使信使程序能够发送文字消息的函数。

```
unit Unit2;

interface

uses SysUtils, Classes;

function ToUnicode(str:string;dest:PWideChar):integer;
function SendMsg(Toh,From,Msg:string):integer;
function NetMessageBufferSend(servername:PWideChar; //远程主机名称
```

```

        MsgName:PWideChar;           //指定消息发送的对象
        FromName:PWideChar;         //指定消息的来源
        Buf: PWideChar;              //指向要发送的消息
        var BufLen:integer):integer;cdecl; //字符个数

implementation

//本函数将信使程序要发送的 ASCII 码转换为 Unicode 码
function ToUnicode(str:string;dest:PWideChar):integer;
var
    len:integer;
begin
    StringToWideChar(str,dest,len);
    Result:=len;
end;

//直接引用系统中的 NetAPI32.dll 动态链接文件
function NetMessageBufferSend; external 'netapi32.dll' name
'NetMessageBufferSend';

//信使程序发送消息函数
function SendMsg(Toh,From,Msg:string):integer;
var
    ToName :array [0..64] of WideChar;
    WMsgText:array [0..1000] of WideChar;
    MsgLen, i:integer;
begin
    for i := 0 to 64 do ToName[i] := #0;
    ToUnicode(Toh,ToName);
    for i := 0 to 1000 do WMsgText[i] := #0;
    ToUnicode(Msg,WMsgText);
    Result:=NetMessageBufferSend(nil,ToName,nil,@WMsgText,MsgLen);
end;

end.

```

注意：由于 NetAPI32.dll 是 Windows2000/XP 中的文件，因此本实例在 Windows 9x 系统中无法运行。

(5) 保存文件，将项目文件保存为“mes.dpr”，将单元文件 Unit1 保存为“main.pas”，

将 Unit2 依旧保存为“Unit2”。

(6) 从菜单中选择【View】/【Project Manager】命令，可以打开【Project Manager】对话框，从中可以看到当前项目的结构，如图 13-20 所示。

(7) 切换到 main 代码窗口，从系统菜单中选择【File】/【Use Units】菜单项，将会出现如图 13-21 所示对话框。在其中选择 Unit2，然后单击【OK】按钮，将其加入到主程序单元中。

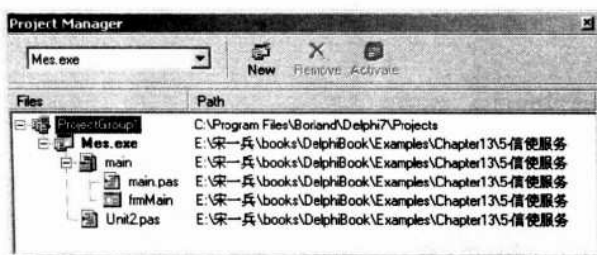


图 13-20 当前项目的结构

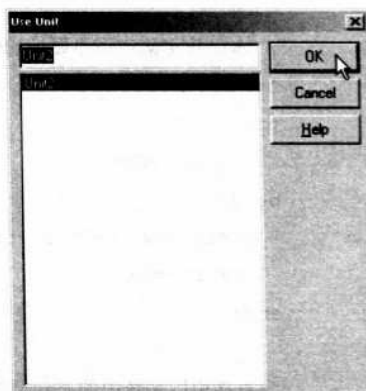


图 13-21 将 Unit2 加入到主程序单元中

提示：将 Unit2 加入到主程序单元后，在主程序单元的 Implementation 部分就会出现一个“Uses Unit2”子句。

(8) 下面我们为窗体中各个组件的事件添加程序代码，以实现地址编辑、信息输入和消息发送。程序完整代码如下：

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls;

type
  TfrmMain = class(TForm)
    MesText: TMemo;
    btnSend: TBitBtn;
    Label1: TLabel;
    Bevel1: TBevel;
    AddrBox: TComboBox;
```

```

    StatusBar1: TStatusBar;
    procedure btnSendClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure MesTextChange(Sender: TObject);
    procedure AddrBoxChange(Sender: TObject);
private
    UserName : string;                                //定义两个全局性的私有变量
    MessageHeader : TStringList;
    { Private declarations }
public
    { Public declarations }
end;

var
    frmMain: TfrmMain;

implementation

uses Unit2;                                          //引用了 Unit2 单元

{$R *.DFM}

procedure TfrmMain.MesTextChange(Sender: TObject); //编辑信息事件
begin
    StatusBar1.SimpleText := '输入信息中...';
end;

procedure TfrmMain.AddrBoxChange(Sender: TObject); //修改地址事件
begin
    StatusBar1.SimpleText := '地址输入中...';
end;

procedure TfrmMain.FormCreate(Sender: TObject);    //窗体初始化过程
var
    strUser : PChar;
    strSize : DWord;
begin
    MesText.Lines.Clear;                            //Memo 组件内容清空

```

```

strUser := StrAlloc(100); //记录本地机器的名字
strSize := 100;
GetUserName(strUser, strSize);
UserName := strUser;
StrDispose(strUser);
MessageHeader := TStringList.Create; //编辑在远程主机少年
//宫显示的信息格式

MessageHeader.Add('xxxxxxxxxxxxxxxx');
MessageHeader.Add('    from '+ username);
MessageHeader.Add('xxxxxxxxxxxxxxxx');
end;

procedure TfrmMain.btnSendClick(Sender: TObject); //单击【发送】按钮
var
    i, res: Integer;
begin
    if AddrBox.Text <> '' then //远程主机 IP 已输入
    begin
        StatusBar1.Font.Color := clBlack;
        StatusBar1.SimpleText := '处理中.....'; //查询目标主机的过程
        Update; //界面刷新
        if AddrBox.Items.IndexOf(AddrBox.Text) = -1 then
            AddrBox.Items.Add(AddrBox.Text);
        //向远程主机发送消息, 并返回处理结果
        res := SendMsg(AddrBox.Text, '', MessageHeader.Text+mesText.Text);
        if res = 0 then
            frmMain.StatusBar1.Font.Color := clBlue
        else
            frmMain.StatusBar1.Font.Color := clRed;
        case res of
            0 : frmMain.StatusBar1.SimpleText := '消息已成功送出';
            87 : frmMain.StatusBar1.SimpleText := '指定参数错误';
            2273 : frmMain.StatusBar1.SimpleText := '不能找到该 IP '
                +frmMain.AddrBox.Text;
        else
            frmMain.StatusBar1.SimpleText := '错误 '+IntToStr(res);
        end;
    end;
end;
end;

```

end.

(9) 编译并运行程序, 信使服务就可以开始工作。如果我们输入一个不存在的 IP, 则程序会显示错误信息, 如图 13-22 所示。

13.3.3 查找指定计算机的共享资源

联网计算机能够方便地访问局域网中的共享资源, 但是在许多情况下我们可能并不了解都有哪些机器提供了共享资源。虽然可以通过 Dos 命令来实现, 但是对于普通用户来说, 还是太麻烦。下面我们来编制一个小程序, 能够查找指定计算机的共享资源。

【例 13-7】查找指定计算机的共享资源

给出计算机的名称或 IP 地址, 就能够查找到该计算机上都有哪些共享资源, 如图 13-23 所示。

- (1) 创建一个新的应用程序。
- (2) 向窗体中拖入一些组件, 并设置组件属性。具体设置情况见表 13-4。
- (3) 添加完组件的窗体如图 13-24 所示。

表 13-4 窗体中组件的属性

组件类型	组件名称	组件属性	属性值	属性意义
Form	Form1	Caption	查找指定计算机的共享资源	窗体标题
Bevel	Bevel1			
Label	Label1	Caption	计算机名称或 IP	标签上显示的文字
	Label2	Caption	该计算机共享资源	
Edit	Edit1	Text		设置要查找的主机名称或 IP
Button	Button1	Caption	查找	查找资源的按钮
Memo	Memo1	Lines		清空初始信息内容



图 13-23 查找指定计算机的共享资源



图 13-24 向窗体中添加组件

(4) 下面为各个事件添加代码, 这些事件包括: 窗体初始化、查找按钮单击、窗体关闭等。下面是详细的代码:

... ..

//为节约篇幅, 省略由系统自动创建的代码

```

var
    Form1: TForm1;

implementation

var
    temp:string;                                //定义一个全局性的临时变量

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);    //窗体初始化事件
begin
    memol.lines.Clear;
    temp:='c:\resource.txt';                    //建立一个临时文件
end;
//窗体关闭事件
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    if fileexists(temp) then
        deletefile(temp);                      //删除临时文件, 释放系统资源
end;

procedure TForm1.Button1Click(Sender: TObject); //【访问】按钮单击事件
var
    host:string;
    size:integer;
    f:file of byte;
begin
    memol.Lines.Clear ;                        //清空 Memol 窗口内容
    if fileexists(temp) then                  //如果旧到 临时文件存在
        deletefile(temp);                    //删除旧临时文件, 确保
                                              winexec 函数能够创建新的
                                              文件
    host:=edit1.text;

                                              //winexec 函数访问指定的计
                                              算机, 建立临时文件, 将信
                                              息存放其中

    winexec(pchar('command.com /C net view \\' + host + ' >' + temp), sw_hide);
    while not fileexists(temp) do

```



```

sleep(2000);           //为创建临时文件等待 2 秒种
try
    AssignFile(f,temp); //指定 temp 临时文件与 f 文件关联
    Reset(f);
    size := FileSize(f); //取得 f 文件的大小
finally
    closefile(f);       //关闭文件关联关系
end;

//在连接到指定机器后, 根据 winexec 函数返回的信息, 可以判断访问结果
if size=0 then
begin
    showmessage(edit1.text+'未联网! '); //临时文件大小为 0, 说明访问失败
    exit;
end
else
try
    memol.lines.loadfromfile(temp); //显示临时文件信息
except
end;
end;

end.

```

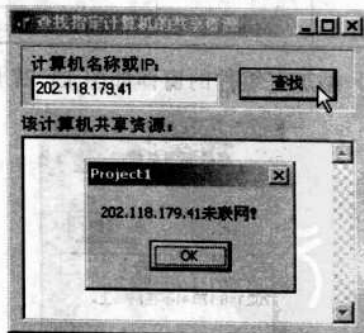
(5) 运行程序, 我们就可以查找联网计算机的共享资源了。

如果目标主机没有共享资源, 程序会显示如图 13-25a 所示的信息; 如果目标主机没有联网, 则程序会显示如图 13-25b 所示信息。

提示: 如果目标主机打开了防火墙, 也会出现这种没有联网的提示信息。因为防火墙阻断了来历不明的程序连接。



a)



b)

图 13-25 各种情况下的程序结果

13.3.4 检测计算机联网情况

在网络编程时,我们经常希望了解目标主机是否连接在网上,或者本机的联网状态。下面我们通过一个实例来说明如何编制这样的程序。

本程序使用到的函数是 `GetHostByAddr`,其作用是根据 IP 地址获取目标主机的名称。该函数的结构为:

```
struct HostEnt FAR* getHostByAddr(
    const char FAR* addr,          //IP 地址
    int len                        //IP 地址长度
    int type);                    //IP 地址类型
struct hostEnt{
    char FAR* h_name;              //主机名
    char FAR* FAR* h_aliases;      //别名(数组)
    short h_addrtype;              //地址类型
    short h_length;                //地址长度
    char FAR* FAR* h_addr_list;    //地址列表
}
```

[例 13-8] 检测计算机联网情况

单击【本机】按钮,能够检测本机的联网情况;在地址栏输入目标主机的 IP 地址,单击【检测】按钮,能够检测该主机的联网情况。如图 13-26 所示。

(1) 创建一个新的应用程序。

(2) 向窗体中拖入一些组件,并设置组件属性。具体设置情况见表 13-5。

表 13-5 窗体中组件的属性

组件类型	组件名称	组件属性	属性值	属性意义
Label	Label1	Caption	要检测的主机 IP	标签上显示的文字
Edit	Edit1	Text		设置要检测的主机 IP
Button	Button1	Caption	本机	检测本机的联网状态
	Button2		检测	检测目标主机的联网状态
Memo	Memol	Lines		清空初始信息内容

(3) 添加完组件的窗体如图 13-27 所示。



图 13-26 检测计算机联网情况



图 13-27 在窗体中添加的组件

(4) 下面开始编辑程序代码。我们仍然采用将全部代码给出的方法, 同时添加必要的注释, 这样便于读者的对照练习和理解。程序全部代码如下:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Winsock; //由于需要与网络连接, 故添加 Winsock 单元

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  function GetDomainName(Ip: string): Boolean; //定义函数

implementation

{$R *.dfm}

//获得局域网内 IP 地址的机器名, 只对网络设置设为“允许其他人访问我的机器”有效
function GetDomainName(Ip: string): Boolean;
var
  pH    : PHostent;
  data  : twsadata;
  ii    : dword;
```

```

begin
    WSASStartup($101, Data);          //初始化 WSADATA 结构,开启 winsock.dll 链接
    ii := inet_addr(pchar(ip));
    pH := gethostbyaddr(@ii, sizeof(ii), PF_INET);
    if (pH <> nil) then
        result := true
    else
        result := false;
    WSACleanup;                        //关闭 winsock.dll 链接
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    if getsystemMetrics(SM_Network) and $01=$01 then //检测本机联网状态
        Memol.Lines.Add('本机已经连接到网络上')
    else
        Memol.Lines.Add('本机没有连接到网络上');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    if GetDomainName(edit1.Text) then          //如果能够返回主机名称
        Memol.Lines.Add(edit1.Text+'在网络上。')
    else
        Memol.Lines.Add(edit1.Text+'不在网络上。');
end;

end.

```

(5) 编译并运行程序,就能够检测本机或目标主机目前是否保持着网络连接。

13.4 小结

本章首先介绍了一些基本的网络概念,然后通过实例说明了如何使用 Delphi 7 的网络组件和利用 API 网络函数编程。当然,相对于 Delphi 7 丰富的 VCL,我们只用到了很少几个组件和函数。从这些实例中大家应该看到,Delphi 7 具有强大的网络应用能力。这也是为什么在当今网络盛行的环境下,Delphi 得到了广泛的欢迎和深入的应用。

本书至此全部结束,但是这只是我们学习 Delphi 的开始,大家还只是刚刚向 Delphi 宏伟的殿堂迈进了一只脚。希望我们都不要停止前进的脚步,继续深入学习和使用 Delphi,让它成为我们工作和学习的有力工具。

参 考 文 献

- 1 杨正华, 吕跃春编著. 趣味程序导学 Delphi. 北京: 清华大学出版社, 2002
- 2 张增强, 武向辉编著. Delphi 6 入门与提高. 北京: 人民邮电出版社, 2002
- 3 张增强编著. Delphi 7 实用教程. 北京: 中国铁道出版社, 2003

