

可下载教学资料

<http://www.tup.tsinghua.edu.cn>

21世纪普通高校计算机公共课程规划教材

# Delphi 2007 程序设计教程

杨盛泉 刘白林 主编

刘萍萍 白小军 副主编

王中生 主审

清华大学出版社

更多资源请访问稀酷客([www.ckook.com](http://www.ckook.com))

## 丛书特点

- \* 案例驱动的教学模式
- \* 一线优秀教师担纲编写
- \* 立体化教学资源解决方案

ISBN 978-7-302-21971-2



9 787302 219712 >

定价：29.50元



21 世纪普通高校计算机公共课程规划教材

# Delphi 2007 程序设计教程

杨盛泉 刘白林 主编  
刘萍萍 白小军 副主编  
王中生 主审

清华大学出版社  
北京

## 内 容 简 介

Delphi 是一种深受广大程序开发人员喜爱的快速开发工具,其简单、高效、灵活的特点使它得到了广泛的应用。本书以 Delphi 2007 for Win32 为开发平台,对 Delphi 开发做了较全面的介绍。本书使用面向对象可视化程序开发的方法,解决实际工作中的工程应用系统设计与开发工作。本书内容翔实,实例丰富,浅显易懂,图文并茂,知识点难易结合,可使学生更容易掌握 Delphi 2007 程序设计的知识和技巧。

本书的读者对象为计算机软件编程人员,也可作为大学计算机相关专业本科生和研究生的编程教材和参考书。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

## 图书在版编目(CIP)数据

Delphi 2007 程序设计教程/杨盛泉,刘白林主编. —北京:清华大学出版社,2010.4  
(21 世纪普通高校计算机公共课程规划教材)

ISBN 978-7-302-21971-2

I. ①D… II. ①杨… ②刘… III. ①软件工具—程序设计—高等学校—教材  
IV. ①TP311.56

中国版本图书馆 CIP 数据核字(2010)第 018848 号

责任编辑:梁 颖 赵晓宁

责任校对:李建庄

责任印制:杨 艳

出版发行:清华大学出版社

地 址:北京清华大学学研大厦 A 座

<http://www.tup.com.cn>

邮 编:100084

社 总 机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62776969, [c-service@tup.tsinghua.edu.cn](mailto:c-service@tup.tsinghua.edu.cn)

质 量 反 馈:010-62772015, [zhiliang@tup.tsinghua.edu.cn](mailto:zhiliang@tup.tsinghua.edu.cn)

印 装 者:北京市清华园胶印厂

经 销:全国新华书店

开 本:185×260 印 张:20 字 数:495 千字

版 次:2010 年 4 月第 1 版 印 次:2010 年 4 月第 1 次印刷

印 数:1~4000

定 价:29.50 元

---

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系  
调换。联系电话:(010)62770177 转 3103 产品编号:036331-01



# 出版说明

---

随着我国改革开放的进一步深化,高等教育也得到了快速发展,各地高校紧密结合地方经济建设发展需要,科学运用市场调节机制,加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度,通过教育改革合理调整和配置了教育资源,优化了传统学科专业,积极为地方经济建设输送人才,为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是,高等教育质量还需要进一步提高以适应经济社会发展的需要,不少高校的专业设置和结构不尽合理,教师队伍整体素质亟待提高,人才培养模式、教学内容和方法需要进一步转变,学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007年1月,教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》,计划实施“高等学校本科教学质量与教学改革工程(简称‘质量工程’)”,通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容,进一步深化高等学校教学改革,提高人才培养的能力和水平,更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中,各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势,对其特色专业及特色课程(群)加以规划、整理和总结,更新教学内容、改革课程体系,建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上,经教育部相关教学指导委员会专家的指导和建议,清华大学出版社在多个领域精选各高校的特色课程,分别规划出版系列教材,以配合“质量工程”的实施,满足各高校教学质量和教学改革的需要。

本系列教材立足于计算机公共课程领域,以公共基础课为主、专业基础课为辅,横向满足高校多层次教学的需要。在规划过程中体现了如下一些基本原则和特点。

(1) 面向多层次、多学科专业,强调计算机在各专业中的应用。教材内容坚持基本理论适度,反映各层次对基本理论和原理的需求,同时加强实践和应用环节。

(2) 反映教学需要,促进教学发展。教材要适应多样化的教学需要,正确把握教学内容和课程体系的改革方向,在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养,为学生知识、能力、素质协调发展创造条件。

(3) 实施精品战略,突出重点,保证质量。规划教材把重点放在公共基础课和专业基础课的教材建设上;特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版,逐步形成精品教材;提倡并鼓励编写体现教学质量和教学改革成果的教材。

(4) 主张一纲多本,合理配套。基础课和专业基础课教材配套,同一门课程针对不同层次、面向不同专业的多本具有各自内容特点的教材。处理好教材统一性与多样化,基本教材与辅助教材、教学参考书,文字教材与软件教材的关系,实现教材系列资源配套。

(5) 依靠专家,择优选用。在制定教材规划时要依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时,要引入竞争机制,通过申报、评审确定主题。书稿完成后要认真实行审稿程序,确保出书质量。

繁荣教材出版事业,提高教材质量的关键是教师。建立一支高水平教材编写梯队才能保证教材的编写质量和建设力度,希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21 世纪普通高校计算机公共课程规划教材编委会

联系人:梁颖 liangying@tup.tsinghua.edu.cn



# 前言

---

计算机的产生和发展,彻底改变了人们的工作和生活方式,为人们带来了极大的方便。随着社会信息化技术的进一步发展,必将对人们使用计算机提出更高的要求。计算机应用技术已经成为 21 世纪必不可少的基本技能之一,计算机应用技术与应用范围的日益扩展,主要得益于丰富的应用软件,其中常见工具软件的使用是每个计算机使用者首先要掌握的,为此,编写了这本适合在校学生和广大计算机爱好者使用的《Delphi 2007 程序设计教程》一书。

Delphi 是由著名的 Borland 公司开发的可视化编程工具,拥有功能强大的集成开发环境(IDE)和速度快、效率高的编译器,兼具 Visual C++ 的功能强大和 Visual Basic 易学易用的特点,使得其程序设计可视化程度高,代码简捷易读,因此深受编程人员的喜爱。

本书为适应不同层次读者的需要,从 Delphi 的基本知识讲起,由浅入深,全面讲述 Delphi 2007 for Win32 的集成开发环境、Delphi 的编程语法基础——Object Pascal、常用组件的使用、窗体的设计、菜单的设计、对话框的设计、多文档界面程序的设计、多媒体编程和数据库应用程序设计等内容。本教材以培养应用能力为基本出发点,主要以理论加实例的方式讲述 Delphi 可视化程序设计的主要原理、方法与知识点,目的在于通过主要内容介绍,让读者自己摸索相应的次要部分,培养读者自我学习的能力,最终能够使得读者迅速掌握实用的可视化编程技术。

本书是按照教育部新世纪人才创新项目教材编写要求编写的系列教材之一。在参考有关资料的同时,结合本科学生以及职业教育学生的特点,将实践能力培养放在首位。本书在内容组织上,力求做到先进、简单、实用。本书共分 9 章,前面一部分章节详细介绍了 Delphi 语言中的 Object Pascal 语法,给出了大量经典的控制台算法程序;后面章节介绍了 Delphi 面向对象程序设计原理与技术和 Windows 可视化程序方法,其中包括 Delphi 的强大功能、最新特性 VCL 组件库、界面程序设计、图形图像程序设计、多媒体与动画程序设计、数据库程序设计等。本书内容由浅入深,对 Delphi 程序设计的精华部分做了详细讲解,并提供了丰富而且经典的范例供读者参考。

本教材在编写过程中得到了许多高校教师的关心和帮助,并提出许多宝贵的修改意见,对于他们的关心、帮助和支持,编者表示十分感谢。

在编写本书的过程中参考了大量的相关资料,从中汲取了许多宝贵经验,在此谨表谢意。由于作者水平有限,书中的不妥和错误在所难免,恳请各位专家、读者不吝指正。联系邮箱: xatuysq@163.com。

编者

2010 年 2 月

# 目 录

---

第 1 章 Delphi 概述	1
1.1 Delphi 2007 简介	1
1.2 Delphi 2007 的安装	3
1.2.1 安装 Delphi 2007 for Win32 的系统要求	3
1.2.2 Delphi 2007 for Win32 的安装	3
1.3 Delphi 2007 的集成开发环境	4
1.3.1 主窗口	5
1.3.2 设计视图	6
1.3.3 代码编辑器	7
1.3.4 对象观察器	9
1.3.5 结构视图	10
1.3.6 项目管理器、模型视图和数据管理器	10
1.3.7 欢迎页面	11
1.3.8 历史代码页面	12
1.3.9 帮助系统	12
1.4 Delphi 2007 程序设计简介	13
1.5 Delphi 2007 程序基本结构	16
1.5.1 项目和项目文件	16
1.5.2 窗体文件	18
1.5.3 单元文件	18
1.5.4 命名空间	20
1.5.5 Delphi 2007 的文件类型	21
本章小结	22
思考与练习	22
第 2 章 数据类型、运算符和表达式	23
2.1 控制台程序设计	23
2.1.1 控制台程序的建立	23
2.1.2 基本输出语句 Write	26
2.1.3 基本输入语句 Read	27



2.2	标识符、常量与变量 .....	29
2.2.1	标识符 .....	29
2.2.2	保留字 .....	30
2.2.3	常量 .....	30
2.2.4	变量 .....	32
2.2.5	注释 .....	33
2.3	基本数据类型 .....	34
2.3.1	有序类型 .....	34
2.3.2	实数类型 .....	39
2.3.3	日期时间类型 .....	40
2.3.4	字符串类型 .....	41
2.3.5	可变类型 .....	41
2.3.6	类型转换 .....	42
2.4	运算符与表达式 .....	43
2.4.1	表达式 .....	43
2.4.2	运算符 .....	43
2.5	构造数据类型 .....	48
2.5.1	集合类型 .....	48
2.5.2	数组类型 .....	50
2.5.3	字符串类型 .....	53
2.5.4	记录类型 .....	54
2.5.5	指针类型 .....	57
本章小结 .....		59
思考与练习 .....		59
<b>第3章 程序结构、函数与过程 .....</b>		<b>60</b>
3.1	顺序结构程序设计 .....	60
3.1.1	Delphi 程序基本架构 .....	61
3.1.2	顺序程序举例 .....	62
3.2	选择结构程序设计 .....	63
3.2.1	if 语句 .....	64
3.2.2	case 语句 .....	68
3.3	循环结构程序设计 .....	70
3.3.1	while 语句 .....	70
3.3.2	直到循环 .....	72
3.3.3	for 语句 .....	74
3.3.4	辅助控制语句 .....	75
3.3.5	循环嵌套 .....	77
3.4	函数与过程 .....	79

3.4.1	函数 .....	79
3.4.2	过程 .....	82
3.4.3	函数、过程的数据传递 .....	84
3.4.4	全局变量、局部变量及它们的作用域 .....	85
3.4.5	函数和过程的递归调用 .....	86
3.4.6	函数和过程重载 .....	88
3.4.7	Forward 声明 .....	89
3.4.8	函数和过程默认参数 .....	90
本章小结	.....	91
思考与练习	.....	91
<b>第 4 章</b>	<b>面向对象程序基础与理论 .....</b>	<b>92</b>
4.1	类与对象 .....	92
4.1.1	类 .....	93
4.1.2	类的成员 .....	94
4.1.3	对象 .....	94
4.1.4	self 引用 .....	97
4.1.5	类运算符 .....	97
4.2	方法 .....	98
4.2.1	方法的特征与调用 .....	98
4.2.2	方法的类别 .....	99
4.2.3	方法的绑定 .....	102
4.3	类的继承、封装和多态 .....	105
4.3.1	类的继承性 .....	105
4.3.2	类的封装性 .....	106
4.3.3	类的多态性 .....	107
4.3.4	类的可见性 .....	109
4.4	异常处理 .....	110
4.4.1	异常分类 .....	110
4.4.2	异常保护和处理机制 .....	112
本章小结	.....	113
思考与练习	.....	114
<b>第 5 章</b>	<b>Windows 窗体和常用组件 .....</b>	<b>115</b>
5.1	Delphi 程序与窗体 .....	115
5.1.1	窗体的属性 .....	116
5.1.2	窗体常用的方法 .....	120
5.1.3	窗体常用事件 .....	121
5.2	常用 Windows 组件概述 .....	124



5.3 文本显示输入与按钮类组件 .....	125
5.3.1 TLabel .....	125
5.3.2 TButton .....	126
5.3.3 TEdit .....	127
5.3.4 TMemo .....	132
5.3.5 TRadioButton .....	134
5.3.6 TRadioGroup .....	134
5.3.7 TCheckBox .....	136
5.3.8 TBitBtn .....	138
5.4 列表类与滚动条组件及时钟组件 .....	139
5.4.1 TListBox .....	139
5.4.2 TComboBox .....	143
5.4.3 TScrollBar .....	144
5.4.4 TTimer .....	146
5.5 组件排列布局 .....	146
本章小结 .....	148
思考与练习 .....	149
<b>第6章 应用程序界面设计 .....</b>	<b>150</b>
6.1 创建主菜单 .....	150
6.1.1 TMainMenu 组件概述 .....	151
6.1.2 TMainMenu 组件的主要属性介绍 .....	152
6.1.3 主菜单的设计过程 .....	153
6.1.4 TMainMenu 菜单设计综合示例 .....	153
6.2 鼠标右键弹出式菜单 .....	157
6.2.1 TPopupMenu 组件 .....	157
6.2.2 鼠标右键弹出式菜单设计 .....	158
6.3 工具栏与状态行设计 .....	158
6.3.1 TPanel 组件 .....	158
6.3.2 TSpeedButton 组件 .....	158
6.3.3 工具栏设计举例 .....	159
6.3.4 TStatusBar 组件 .....	161
6.3.5 状态栏设计举例 .....	162
6.4 对话框函数 .....	165
6.4.1 对话框与模态窗口 .....	165
6.4.2 Delphi 对话框函数 .....	165
6.5 对话框组件 .....	168
6.5.1 用于文件选择的 TOpenDialog 对话框组件 .....	169
6.5.2 用于文件保存的 TSaveDialog 对话框组件 .....	171

6.5.3	用于字体选择的 TFontDialog 对话框组件 .....	171
6.5.4	用于颜色选择的 TColorDialog 对话框组件 .....	173
6.5.5	用于打印的 TPrintDialog 对话框组件 .....	174
6.5.6	用于打开图像的 TOpenPictureDialog 对话框组件 .....	174
6.5.7	用于打印模式设置的 TPrinterSetupDialog 对话框组件 .....	175
6.5.8	用于查找的 TFindDialog 对话框组件 .....	175
6.6	多文档界面程序设计 .....	176
6.6.1	利用 Delphi 模板创建 MDI 程序 .....	176
6.6.2	用户设计常规 MDI 程序 .....	179
6.6.3	MDI 应用程序菜单的合并 .....	181
6.6.4	多页窗体设计 .....	183
6.7	Delphi 拖放技术编程 .....	184
6.7.1	拖放技术编程概述 .....	184
6.7.2	拖放编程举例 .....	184
6.8	窗体的分割技术 .....	186
6.8.1	分割技术概述 .....	186
6.8.2	Delphi 分割窗体操作 .....	187
	本章小结 .....	187
	思考与练习 .....	188
<b>第 7 章</b>	<b>图形图像程序设计 .....</b>	<b>189</b>
7.1	TCanvas .....	189
7.1.1	TColor .....	189
7.1.2	Pen 属性 .....	190
7.1.3	MoveTo 与 LineTo 方法 .....	192
7.1.4	Brush 属性 .....	194
7.1.5	Rectangle、RoundRect 与 Ellipse 方法 .....	195
7.1.6	Font 属性以及 TextOut 方法 .....	199
7.1.7	Pixels 属性、Draw 方法和 StretchDraw 方法 .....	200
7.1.8	其他方法 .....	202
7.2	常用图形、图像类 .....	202
7.2.1	TGraphic 类和 TPicture 类 .....	202
7.2.2	TBitMap 位图类 .....	203
7.2.3	TImage 组件 .....	206
7.2.4	TImageList 组件 .....	208
7.2.5	TShape 组件 .....	208
7.2.6	TPaintBox 组件 .....	208
7.3	图形图像组件重绘和鼠标事件 .....	208
7.3.1	处理绘图组件的 OnPaint 事件 .....	208

7.3.2 图形组件鼠标事件	211
本章小结	216
思考与练习	216
<b>第8章 动画与多媒体程序设计</b>	<b>217</b>
8.1 声音播放程序设计	217
8.1.1 Windows 的默认声音	217
8.1.2 使用 API 函数播放 wav 声音文件	219
8.2 TAnimate 组件动画程序设计	224
8.2.1 TAnimate 组件的主要属性和常用方法	224
8.2.2 使用 TAnimate 组件实现动画播放的实例程序	225
8.3 多媒体播放程序设计	227
8.3.1 MediaPlayer 的属性和事件	227
8.3.2 TTrackBar	231
8.3.3 多媒体播放程序设计范例	231
8.4 Flash 动画播放程序设计	235
8.4.1 Delphi 2007 下 Flash ActiveX 组件安装	235
8.4.2 TShockwaveFlash 的主要属性、方法和事件	236
8.4.3 Delphi 程序与 Flash 组件的信息交换	238
8.4.4 简单的 Flash 播放程序设计范例	239
本章小结	241
思考与练习	242
<b>第9章 数据库应用程序设计</b>	<b>243</b>
9.1 数据库基础知识	243
9.1.1 数据库的基本概念	243
9.1.2 Delphi 可访问的主要数据库产品简介	246
9.1.3 Delphi 连接数据库方式	247
9.1.4 数据库应用程序结构	252
9.2 Delphi 数据集组件	253
9.2.1 数据集概述	253
9.2.2 TADOTable 数据集组件的主要属性、方法与事件	254
9.2.3 TADOTable 数据集组件记录的读取与修改	258
9.2.4 TADOTable 数据集组件记录的添加与删除	259
9.2.5 TADOTable 数据集组件的数据查询方法	260
9.2.6 TADOTable 数据集组件的记录移动	263
9.2.7 TADOTable 数据集组件的数据过滤	263
9.3 数据源组件和数据控制组件	264
9.3.1 TDataSource 组件	265

9.3.2	TDBGrid 组件 .....	267
9.3.3	TDBNavigator 组件 .....	270
9.3.4	TDBText 组件与 TDBEdit 组件 .....	274
9.3.5	TDBMemo 组件与 TDBComboBox 组件 .....	276
9.3.6	TDBListBox 组件与 TDBImage 组件 .....	277
9.3.7	TDBCheckBox 组件与 TDBRadioGroup 组件 .....	278
9.4	ADO 组件及应用 .....	280
9.4.1	TADOConnection 组件 .....	280
9.4.2	TADOCommand 组件 .....	291
9.4.3	TADODataSet 组件 .....	294
9.4.4	TADOQuery 组件 .....	297
本章小结 .....		302
思考与练习 .....		302
参考文献 .....		303

面向对象的程序设计思想和方法,是近几年来软件设计、开发和维护技术的一次革命。作为这种新技术的典型代表,Delphi 语言以其灵活性、高效性和可复用性得到广泛的认可、推崇和应用。

Delphi 是著名的 Borland(现在已和 Inprise 合并)公司开发的可视化软件开发工具。“真正的程序员用 C,聪明的程序员用 Delphi”这句话是对 Delphi 最经典、最实在的描述。Delphi 被称为第 4 代编程语言,它具有简单、高效、功能强大的特点。和 Visual C++ 相比,Delphi 更简单、更易于掌握,而在功能上却丝毫不逊色;和 Visual Basic 相比,Delphi 则功能更强大、更实用。可以说 Delphi 同时兼备了 Visual C++ 功能强大和 Visual Basic 简单易学的特点,一直是程序员至爱的编程工具。

Delphi 具有以下特性:基于窗体和面向对象的方法,高速的编译器,强大的数据库支持,与 Windows 编程紧密结合,强大而成熟的组件技术。但最重要的还是 Object Pascal 语言,它才是一切的根本。Object Pascal 语言是在 Pascal 语言的基础上发展起来的,简单易学,具有面向过程和面向对象语言的特点。

Delphi 提供了各种开发工具,包括集成环境、图像编辑(Image Editor)以及各种开发数据库的应用程序,如 Desktop DataBase Expert 等。除此之外,还允许用户挂接其他的应用程序开发工具,如 Borland 公司的资源编辑器(Resource Workshop)。

Delphi 是 Borland 公司推出的基于 Windows 环境的高效程序开发工具,也是一个优秀的数据库应用开发工具。本章简要介绍 Delphi 语言的发展历史、语言特点、编程环境等相关知识。通过本章的学习,要求读者尽快熟悉 Delphi 集成开发环境,并了解如何创建简单的应用程序,建立起 Delphi 项目的概念。

本章主要包括以下内容:

- Delphi 2007 简介;
- Delphi 2007 安装;
- Delphi 2007 集成开发环境;
- Delphi 2007 程序设计简介;
- Delphi 2007 程序基本结构。

## 1.1 Delphi 2007 简介

Delphi 是第 4 代编程语言,是 RAD(Rapid Application Development,快速应用程序开发)工具的代表。从核心上说,Delphi 是一个 Pascal 编译器。

Delphi 2007 是 CodeGear(From Borland) 公司发布的功能强大的面向对象软件开发包 RAD Studio 2007。RAD Studio 2007 包含了许多创新的功能以帮助开发人员使用相同的技术同时开发原生 32 位窗口和 .NET 框架 2.0 的应用程序,并且在未来延伸至原生 64 位窗口。

Delphi 2007 具有良好的可视化应用程序开发环境,其拖放式的可视化设计实现了“所见即所得”; Delphi 2007 使用完全面向对象的 Object Pascal 语言,语法严谨,编译的代码运行效率很高; Delphi 中采用了事件处理机制,对控件进行了很好的封装,隐藏了事件处理的具体细节,方便程序员进行快速开发; 功能强大的 IDE 可以自动生成很多复杂的代码,大大地节省了开发时间。

RAD Studio 2007 可以用于开发各种不同的应用方案,从单机, C/S, 分布式架构到 ASP.NET, Web Service, AJAX。支持的开发框架则从 VCL、VCL.NET 到 ASP.NET。支持的程序语言从 Object Pascal、Delphi、C/C++ 到 Delphi.NET 和 C#。

跟以往版本的 Delphi 比, Delphi 2007 增加了如下功能:

(1) IDE。工具采用 .Net 2.0 编写, 安装时需要 .Net Framework 2.0, 但编译出来的 Exe 是纯正的 Win32 程序, 发布不需要 .Net Framework 2.0。新 IDE 采用 .Net Framework 2.0 是为了更好地统一 IDE 平台, 利用 .Net 的反射、泛型等高级特性, 节省编写 IDE 时间。新的 IDE 确实比 D7 启动要快, 大概五六秒左右就启动, 跟 VS.Net 2005 差不多。

(2) Vista 支持。Delphi 2007 支持 Vista 界面, 封装了 Vista 的新 API 函数。新增属性 Application.MainFormOnTaskBar, 用 Delphi 2007 新建一工程, 然后查看工程文件的源代码, 发现多了一行代码 Application.MainFormOnTaskBar := True; Delphi 2007 默认已将 MainForm 显示于任务栏, 而不是之前版本的 Application。这个功能在以前很多 Delphi 都讨论过, 现在 Delphi 自身支持了。设计此属性很明显, 应该是为了兼容 Windows Vista。

(3) DBX4。DBX4 是 Delphi 2007 最新的数据库存取技术, 不但提供 RDBMS 存取能力, 更可以用来同时开发 Win32 和 .NET 的数据库应用程序, 让撰写一次的程序代码可以支持 Win32 和 .NET 2.0 两个平台, 未来更能够直接移植到原生 Win64 平台。

(4) Blackfish SQL。Delphi 2007 中最新的纯 .NET 数据库。允许使用 Delphi.NET 语言撰写预储程序(Stored Procedure), 触发器(Trigger)。这样可以开发出 .NET 环境的数据库应用程序, 更提供了简易部署能力, 用户甚至只需要复制 Blackfish SQL 的数据库档案就可以轻松完成系统的部署, 省时省成本。Blackfish SQL 更可以和 DBX4 结合在一起, 可以同时开发单机, C/S, 分布式和 Web 应用的解决方案, 一套程序代码可以提供 4 种应用系统的解决方案。

(5) 泛型(Generics)能力。最新的 Delphi 程序语言不但支持 Win32 和 .NET 2.0 平台, RAD Studio 2007 更为 Delphi .NET 程序语言加入了泛型的能力, Delphi 的泛型不会只把用户限制在 .NET 平台, 在 Delphi 2007 中的泛型程序代码未来可以同样在 Win32/Win64 平台中执行。

(6) Web 应用方案。例如 AJAX、REST 等, Delphi 2007 不但提供了完整的 Web 开发能力, 而且同时在 Win32/.NET 两个平台提供 AJAX 开发的功能。可以选择使用 VCL For Web 开发 Win32 的 AJAX 应用, 也可以选择使用 ASP.NET 开发 .NET 的 AJAX 应用。

(7) MDA/DDA 和建模。Delphi 2007 中的 ECO IV 更延伸支持 VCL.NET, 从而可以使用 ECO IV 同时开发 ASP.NET 和 VCL.NET 的应用程序。

## 1.2 Delphi 2007 的安装

### 1.2.1 安装 Delphi 2007 for Win32 的系统要求

- (1) Intel Pentium 166MHz 或配置更高的处理器(建议使用 P2 400MHz 以上的处理器)。
- (2) 128MB 以上内存(建议 256MB 以上)。
- (3) 完全安装专业版大约要占 623MB 硬盘空间。安装完后,系统盘应留有足够的剩余空间以保证系统执行效率,通常应保持 30% 的自由空间。
- (4) 一般使用 Microsoft Windows XP、Vista 或最新的 Windows 7 操作系统平台。建议操作系统为 Windows XP 专业版本,因为最好能在本机安装 MS SQL Server 2000 或者 Access 数据库系统。
- (5) 此外,还要求系统配有 DVD-ROM 驱动器、VGA 或性能更高的彩色显示器(推荐分辨率为 1024×768)、鼠标等外设。

### 1.2.2 Delphi 2007 for Win32 的安装

将 Delphi 2007 for Win32 系统 DVD 盘放入到 DVD-ROM 驱动器中,找到 setup.exe 文件双击运行,即可启动 Delphi 2007 的安装向导,根据向导提示即可完成安装。安装过程中需要输入各种必要的信息,包含正确的序列号,然后根据安装提示就能顺利地完成了所有安装。

安装过程主要分成 4 个部分:收集系统信息、准备安装、安装和结束安装,主要安装过程如图 1-1 所示。

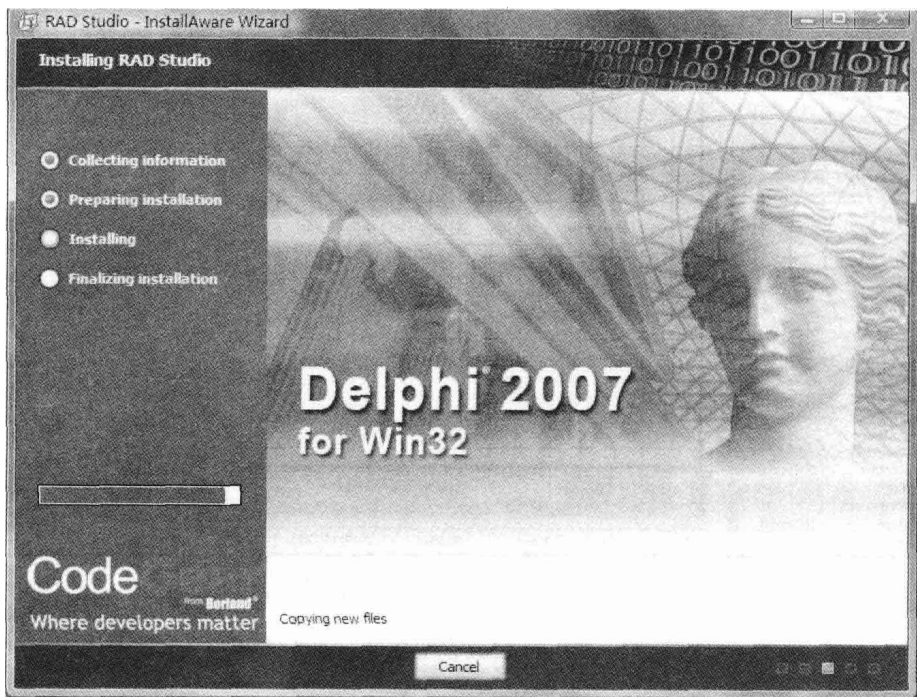


图 1-1 Delphi 2007 安装的主要界面



### 1.3 Delphi 2007 的集成开发环境

Delphi 2007 安装完成后,选择 Windows 的“开始”→“程序”→CodeGear RAD Studio→Delphi 2007 命令运行 Delphi 2007,Windows 菜单启动过程如图 1-2 所示。

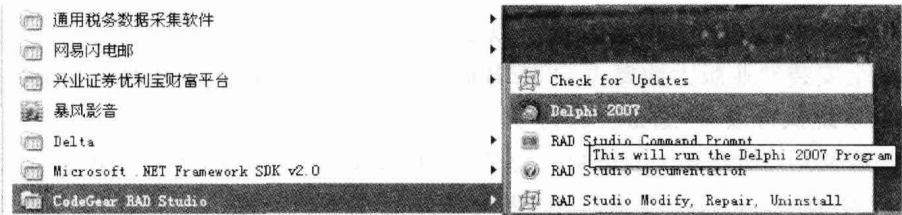


图 1-2 Delphi 2007 的软件启动

Delphi 2007 的编程环境是一个集成环境(Integrated Development Environment, IDE)。所谓集成环境,是指将程序设计所需工具以及帮助系统集中在一起,使得程序的编辑、编译、调试、运行都在同一个可视化环境中进行,十分方便。

Delphi 2007 的集成开发环境如图 1-3 所示,这是 Delphi 2007 标准默认界面,用户可以根据自己的习惯配置 Delphi 界面。为了统一,本书采用 Delphi 2007 默认的标准界面。



图 1-3 Delphi 2007 的界面及各部分说明

要退出 Delphi 2007,选择 File→Exit 命令。

在图 1-3 中,已经标示出了 Delphi 2007 初始窗口中的一些主要部分,下面将介绍这些主要部分。

### 1.3.1 主窗口

Delphi 的主窗口位于屏幕的上端,包括系统菜单栏(Menu)、系统工具条(Tool Bar)和组件面板(Component Panel)。

#### 1. 菜单栏

系统菜单是下拉式菜单,提供了 Delphi 2007 集成开发环境中开发应用程序所需要的各种功能。

#### 2. 系统工具条

工具条位于主窗口的左下端,如图 1-4 所示,由两排工具按钮组成,这些按钮是系统菜单命令的快捷方式,各种图标直观地表示了它能执行的动作。



图 1-4 Delphi 2007 系统默认工具条按钮

(1) 工具条选择组: 选择 View→Toolbars 命令,可以根据自己的需要选择各种类型的标准工具组合。

(2) 工具栏按钮的个性化设置: 选择 View→Toolbars→Customize 命令,将出现 Customize 对话框,如图 1-5 所示。在该对话框中用户可以根据自己的习惯与喜好设置个性化的工具条布局。

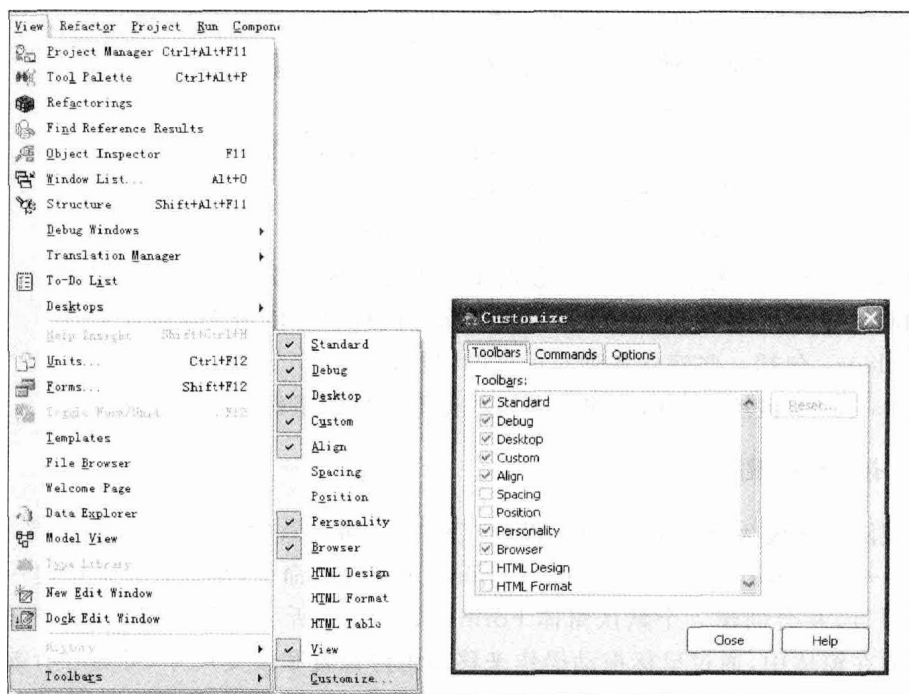


图 1-5 Delphi 2007 系统工具条个性化设置菜单以及定制对话框

### 3. 组件面板

组件是 Delphi 用来实现面向对象程序开发的最基本工具。为了方便程序开发人员使用, Borland Delphi 把常用的组件按照不同的功能排列在 IDE 的组件面板里。在 Delphi 2007 中, 组件面板沿用了 Delphi 8 中的风格, 即不再像以前版本那样采用位于 IDE 上方的分项选项卡形式, 而改成了图 1-6 所示的组件列表形式。在组件面板的上部有一个带下箭头的 Categories 按钮, 单击该分类按钮可展开组件的分类列表以迅速定位于不同的组件分类。

组件面板中所包含的是用户在应用程序开发中所要用到的工具, 其中显示的项目随着用户当前所查看的内容而变化。比如, 现在正在窗体设计器中处理一个窗体, 则组件面板中所显示的就是供窗体设计所使用的各种类型的组件(Delphi 2007 大概提供了约 610 个组件, 归类后分别放在 44 页中), 这时只要双击所需的某一组件, 该组件就会添加到当前窗体中。如果用户现在正在代码编辑器中编辑代码, 则组件面板中所显示的是用户在程序设计中可使用的一些常用代码段, 这时只要双击所需的某一代码段, 该代码段就会添加到程序的当前位置。另外, 组件面板还有定位机制。只要输入组件的第一个字母就能立即得到一个只包含起始字母与输入相符的过滤后的类别列表。同时会高亮显示输入的字母, 且能够在用户继续输入字母时予以进一步的过滤。按 Enter 键将会在设计器窗口放置一个当前选择的组件。

常用的组件页简单解释如下:

- Standard: 包括一些 Windows 标准的控制项或选择菜单;
- Additional: 包括一些常用的控制项;
- Data Access: 包括一些存取数据库的不可见组件;
- Data Controls: 包括一些设计数据库应用程序界面使用的可视化组件;
- dbGo: 包括一些使用 ADO 开发数据库应用程序时使用的数据库组件;
- Dialogs: 包括一些建立各种对话框的组件;
- DataSnap: 包括一些用于建立多层数据库应用的组件。

### 1.3.2 设计视图

设计视图是开展大部分项目程序设计工作的区域。首次启动 Delphi 2007, 然后选择 File→New→VCL Forms Application—Delphi for Win32 命令, 系统将自动创建一个普通的应用程序项目, 并会创建一个默认窗体 Form1, 如图 1-7 所示。窗体相当于组件的容器, 可以把组件放在窗体中, 通过鼠标拖动操作来移动组件位置和改变尺寸, 可随心所欲地安排它们, 以此来设计应用程序的用户界面。窗体上有网格(Grids), 放置组件时网格可以用于定位, 在程序运行时网格是不可见的。

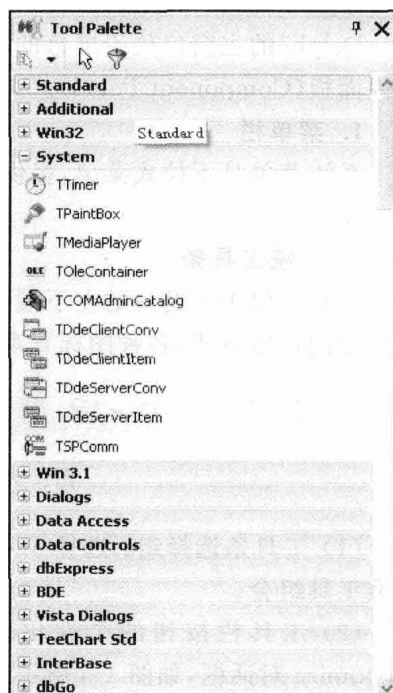


图 1-6 Delphi 2007 组件面板



图 1-7 Delphi 2007 的窗体设计器

### 1.3.3 代码编辑器

代码编辑器如图 1-8 所示,又称做单元窗口,是编写程序代码的地方。可以通过选择 View→Toogle Form/Unit 命令或单击快捷工具栏中的快捷键 Toogle Form/Unit 来显示

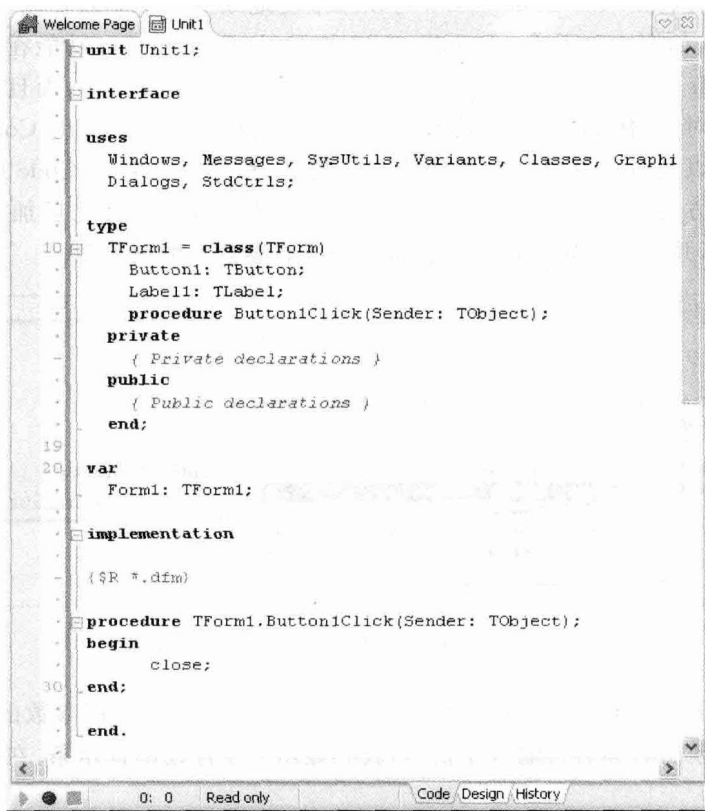


图 1-8 Delphi 2007 的代码编辑器

它。在默认情况下,在代码编辑器和设计视图之间进行切换可以按 F12 键,进一步地还可以单击编辑器右下角的 Code 或 Design 切换。

代码编辑器中,Delphi 2007 提供了很多快捷的代码编辑功能,主要有以下功能。

### 1. 程序调试功能

如果在程序编译中发生错误或产生警告,会在代码编辑器下方 Message 窗口显示相关的错误、警告信息,单击某条信息,光标就会移动到代码中相应的行。

### 2. 帮助查询功能

当程序员对代码中的某个组件或关键字不清楚时,只需要将光标移到该单词上,然后按 F1 键就会自动打开帮助,并显示相关内容。

### 3. 代码分析

代码分析主要包括以下几方面的功能:

#### 1) Class Completion

使用 Delphi 2007 提供的 Class Completion 功能,只需要输入函数或过程的声明,通过按 Ctrl+Shift+C 组合键即可自动创建代码框架。

#### 2) Code Insight

Code Insight 即代码预测,Delphi 能够根据用户的输入产生相应的提示。它又包括:

(1) Code Completion。即代码完成功能。在 Delphi 5 中就已经加入了这项功能,非常受程序员的欢迎,因为 Code Completion 可以大幅度地减少程序员需要输入的程序代码,并且减少输入错误,如图 1-9 所示。Delphi 2007 的 Code Completion 功能在原有的基础上继续改善,新的 Code Completion 窗口不但可以由程序员自行调整大小,而且可以使用不同的颜色代表不同的对象,例如变量、方法和属性等。Delphi 2007 的 Code Completion 窗口中加入了色彩分析以及对对象分门别类的能力。另外,程序员在新的 Code Completion 窗口中选择使用某个方法之后,Code Completion 会自动地在方法名称之后加上“()”,把光标停在圆括号之中,并且自动显示这个方法需要的所有参数及数据类型。

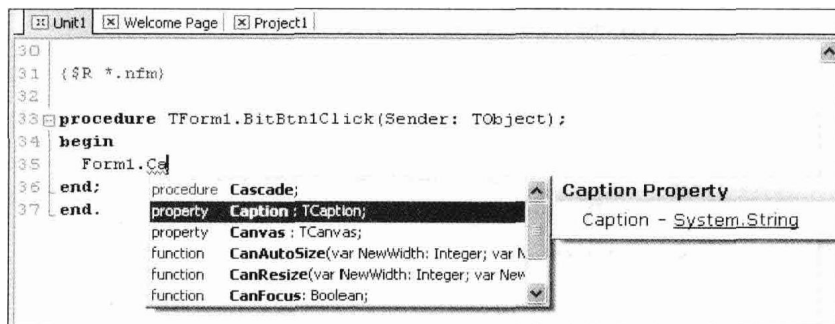


图 1-9 Delphi 2007 的代码 Code Completion 演示

(2) Code Parameters。即代码参数提示。当用户输入一个使用参数的过程或函数时,在输入过程或函数的名称后再输入左括号,就会显示一个浮动的提示条,在提示中有参数的数据类型和说明。

(3) ToolTip Expression Evaluation。即表达式计算。这个功能是在调试程序时使用的,在程序被调试进入断点后,如果想知道这时候某个表达式的值,只需要在代码编辑窗口

将鼠标指向这个表达式,Delphi 就会计算表达式的值并显示出来。

(4) ToolTip Symbol Insight。即符号预测。当用户想知道一个引用自外部文件的类的定义,并查看声明这个类的文件时可以使用这个功能。当将鼠标停留在代码编辑器的任何标识符上时,将弹出一个浮动提示条,提示条中显示该标识符的有关信息,包括其种类、声明的单元文件及其行号,用户可以借此迅速查找到该段代码。

### 1.3.4 对象观察器

对象观察器窗口含有两个选项卡:属性选项卡和事件选项卡,如图 1-10 所示。

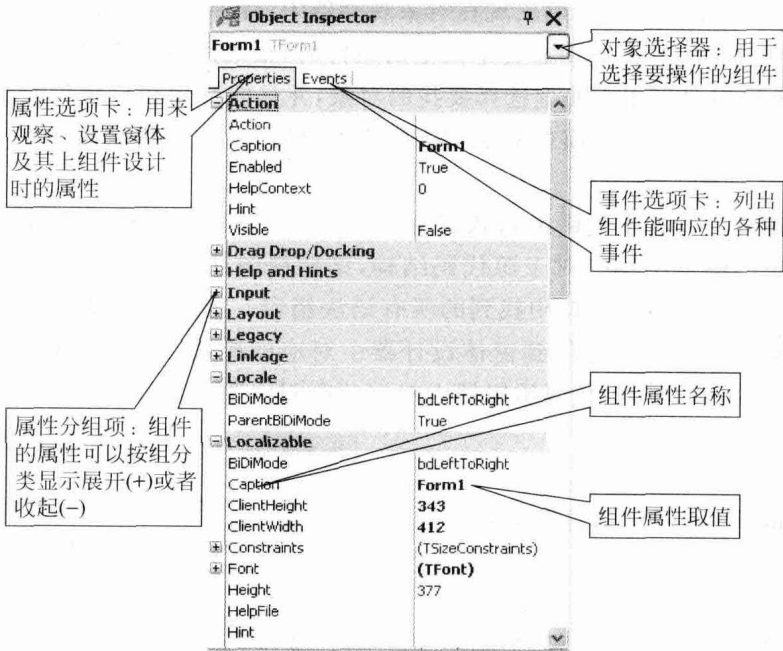


图 1-10 Delphi 2007 的代码编辑器

- 属性选项卡(Properties): 显示窗体中当前被选择组件的属性信息,并允许改变对象的属性。
- 事件选项卡(Events): 列出了当前组件可以响应的事件。单击 Object Inspector 下端的 Events 标签,使得 Events 选项卡可见。在某个事件后边的空白处,可以定义对象接收到相应事件时执行的动作。

首次启动时, Object Inspector 窗口显示的是当前窗体 Form1 的属性。Object Inspector 根据对象属性的多少决定是否显示滚动条。通过移动滚动条可以查看当前对象的全部属性。

此外, Object Inspector 上还有 Object Selector(对象选择器),它位于 Object Inspector 上方的下拉式菜单中。Object Selector 显示了窗体上所有组件的名称和类型,也包含窗体本身。用 Object Selector 可以很容易地在窗体的各个组件之间切换,也可以快速地回到窗体本身。当窗体中含有较多的对象时,这是切换对象,尤其是回到窗体的最快捷途径。

想使 Object Inspector 一直可见,可将鼠标移到 Object Inspector 上,右击以启动 Object

Inspector 的弹出式菜单,将其设置为保持在最顶层(Stay On Top)。这对初学者常是一个很重要的设置方式。

### 1.3.5 结构视图

Delphi 2007 集成开发环境的左上角包含了一个结构视图。这个窗口在某些场合特别有用。

当系统窗体设计器处于工作状态时(如图 1-11 所示),它可以显示窗体中可视化控件的继承结构,用树型结构表达组件之间的包含关系。当程序员在 Structure View 窗口中选择一个组件之后,这个组件会立刻出现在对象观察器中,程序员可以改变这个对象的属性值和添加事件处理过程。当窗体中放置了大量的组件时,很难用鼠标直接选择对象,这时通过 Structure View 窗口可以很方便地选择要找的对象,并且能看到和它相关的组件。开始时, Object TreeView 中只有一个窗体对象,随着组件的加入,这个树状视图的内容会越来越丰富。

当系统窗体设计器处于代码编辑状态时(如图 1-12 所示),同样,它也能显示代码编辑器中源代码对象的继承结构。对于源代码结构,结构视图还能动态地在顶层出现一个“错误”节点,显示错误实时提示窗口中找到的所有语法错误。当查看可视控件的结构时,能双击结构视图中对应的条目来定位到窗体设计器中对应的控件。当观察相应源代码的结构时,能双击相应的条目来转到代码编辑器中对应的声明位置。

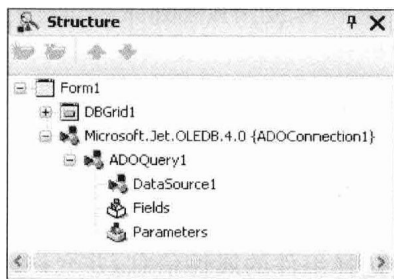


图 1-11 处于设计模式下的结构视图

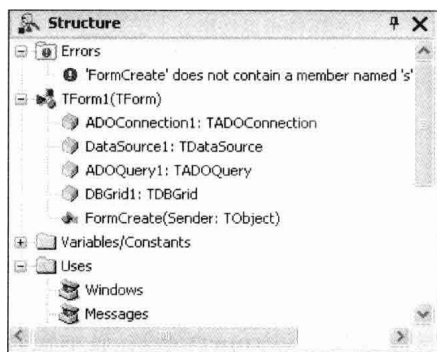


图 1-12 处于代码模式下的结构视图

### 1.3.6 项目管理器、模型视图和数据管理器

#### 1. 项目管理器

默认情况下,Delphi 2007 的项目管理器位于 IDE 的右上边,它以树状列表的形式列出了当前工程所包含的所有文件以及工程共同引用的动态链接库文件等,如图 1-13 所示。一方面使编程人员对当前工程的结构一目了然,另一方面也便于工程文件的管理,如利用该窗口弹出的快捷菜单打开、编辑、添加或删除文件的某些单元文件等。

Delphi 2007 自带的项目管理器现在能显示整个项目

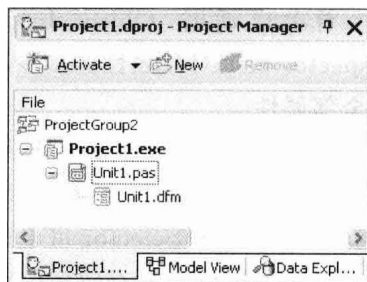


图 1-13 项目管理器

的目录结构(如果是 ASP.NET 项目,甚至还提供了在当前项目目录下创建和管理子目录的功能),这样能更直观地知道文件的放置位置和决定哪个文件是要分发给用户的。在一个项目组中,能够增加项目到不同的目标(和特性)中,或者从一个项目移动到另外一个项目中,也就是将一个特性立即转到另外一个特性中。

## 2. 模型视图

模型视图提供了项目的逻辑表示:命名空间和图节点。仅在 ECO 框架项目中,可向模型添加新的元素;剪切、复制、粘贴和删除元素,等等。模型视图每个节点的右击菜单命令都可能不同。选择 View→Model Views 命令可以打开模型视图。在默认情况下,模型视图显示可扩展的节点树,树中的每个节点可以显示其所包含的元素。也可以隐藏这个节点树,这样可以使得项目的可视化显示更加简单。

## 3. 数据管理器

Delphi 的一个巨大优势是具有强大的数据库支持能力,能方便快捷地开发各种各样的数据库应用程序。

数据管理器如图 1-14 所示,主要为跟数据库相关联的应用服务程序服务,可以用来浏览数据库中的各种对象,如表、字段、存储过程、触发器和索引等,也可以用来创建和管理数据库链接。

而 Delphi 2007 提供的数据库浏览器,可以为开发人员建立、修改、删除数据库连接以及对数据库进行管理提供很多的方便。

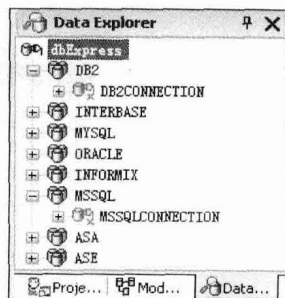


图 1-14 数据管理器

## 1.3.7 欢迎页面

Delphi 2007 的欢迎页面(Welcome Page)如图 1-15 所示,它不仅可以引导用户进入 Delphi 精彩的开发过程,同时可以显示用户最近打开过的项目,还显示开发者(Borland)网络和 RSS 的最新新闻(News)、资源(Resources)。

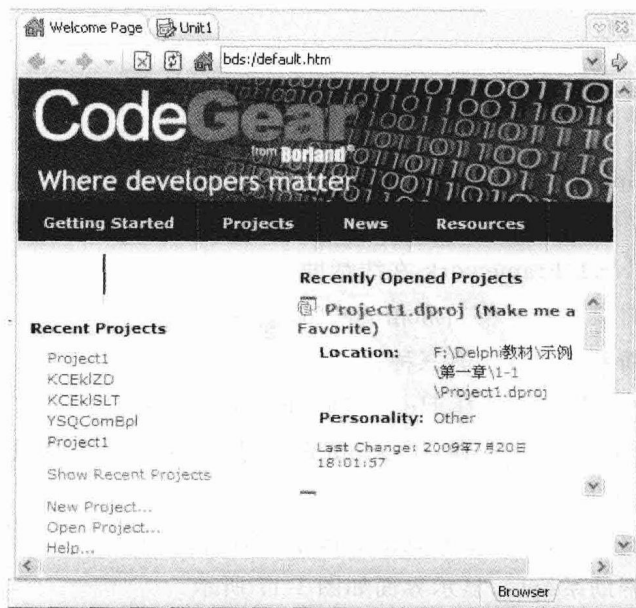


图 1-15 Delphi 2007 的 Welcome Page



### 1.3.8 历史代码页面

Delphi 2007 的历史代码页面(History Page)如图 1-16 所示,它详细地记录项目文件历史编辑过程的日志,并且显示当前单元文件的编辑日期和编辑作者,这样可以按照软件工程的规范有效地管理工程中的各个文件。其作用是让开发人员不断更新和保存应用程序的多个备份,以便检查和比较程序的不同版本、已保存的局部变化或尚未保存的编辑内容。对于团队开发,历史代码页面还可提供源文件的版本信息。

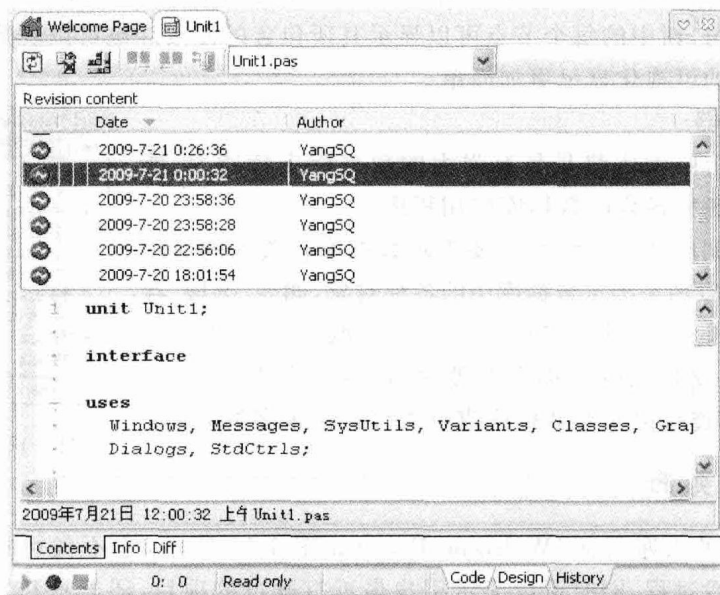


图 1-16 Delphi 2007 的 History Page

### 1.3.9 帮助系统

Delphi 2007 提供了功能非常强大的帮助系统,通过该帮助系统可以获取多方面的信息,如:

- (1) Borland Delphi 2007 快速启动指南。
- (2) Delphi 2007 在线帮助。
- (3) Microsoft .NET Framework 在线帮助。
- (4) Borland 软件开发技术支持和相关网站。

进入 Delphi 帮助系统的方法有三种:

- (1) 在 Delphi 的主界面上选择 Help→Delphi Help 命令。
  - (2) 通过选择 View→Toolbars→Custom 命令显示 Custom 工具栏,然后单击该工具栏上的 Help Contents 工具按钮。
  - (3) 在代码编辑器窗口选中要查询的关键字,然后按下 F1 键。
- 进入 Delphi 的帮助系统后,显示界面如图 1-17 所示。

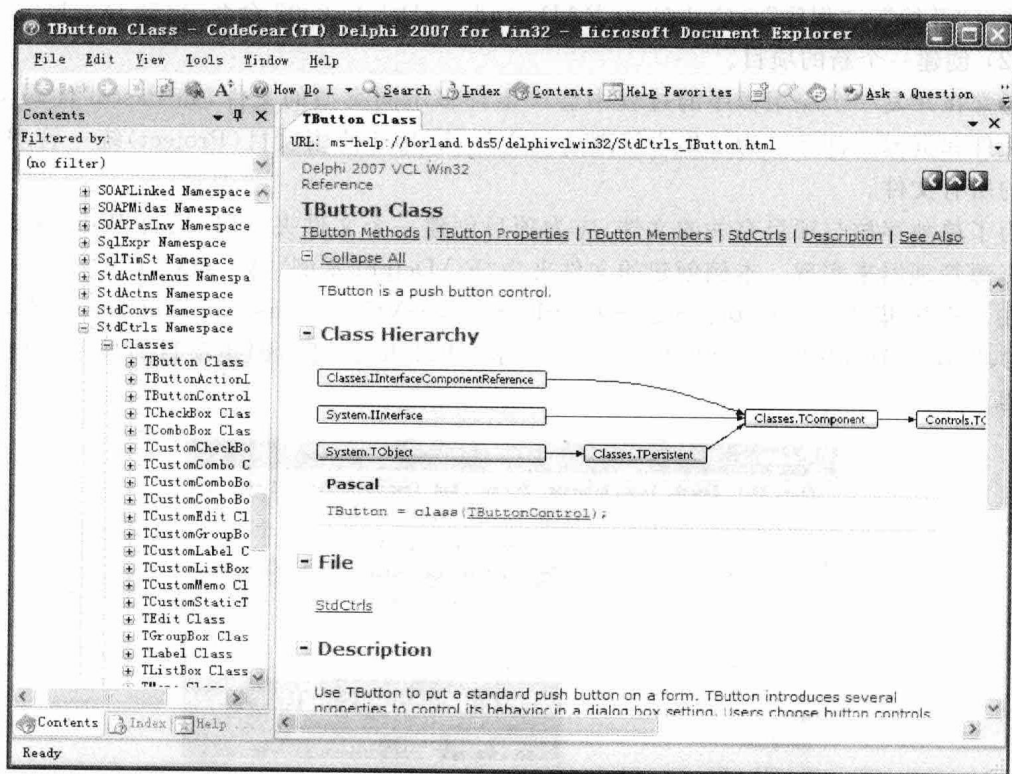


图 1-17 Delphi 2007 的帮助系统

## 1.4 Delphi 2007 程序设计简介

为了更好地理解 Delphi 2007 的应用程序,下面通过一个简单的实例来讲解如何在 Delphi 2007 集成开发环境中编写 Windows 应用程序。

**例 1.1** Delphi 2007 窗体程序设计简单示例。

编写一个应用程序,在窗体中显示“Welcome to study Delphi language!”或者“欢迎学习 Delphi 语言!”,当单击其下的按钮时可以交替显示中英文标签。运行时显示的界面如图 1-18 所示。

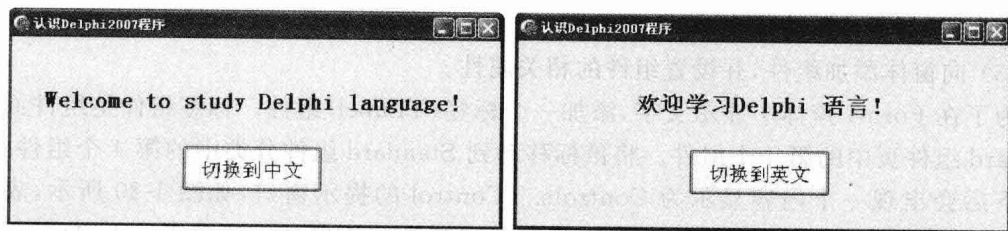


图 1-18 交替显示中英文界面

程序的设计步骤如下:

(1) 启动 Delphi 2007。

选择“开始”→“程序”→CodeGear RAD Studio→Delphi 2007 命令,启动 Delphi 2007。

(2) 创建一个新的项目。

通常在开发一个应用程序的过程中会产生许多不同类型的文件,如 Pascal 代码文件、窗体文件和资源文件等。为了集中管理这些文件,Delphi 使用项目(Project)统一管理应用程序的所有文件。

为了单独存放与项目有关的文件,应为项目创建一个文件夹,可以通过 Windows 环境中的资源管理器来实现。本例创建的文件夹为“F:\Delphi 示例\第一章\1-1”。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32,创建一个新的应用程序,如图 1-19 所示。Delphi 会自动创建项目文件及其他的相关文件。

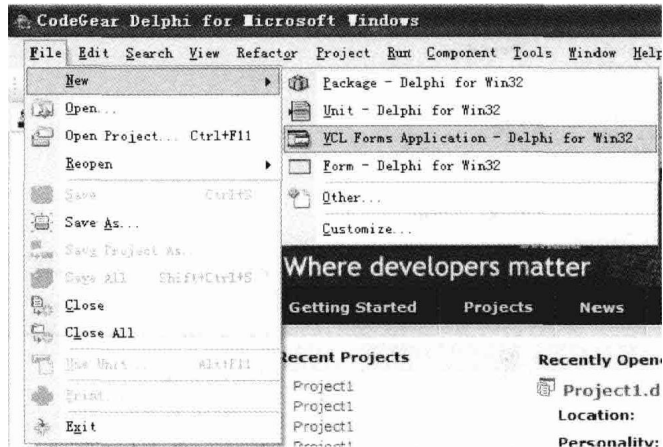


图 1-19 通过菜单创建新的应用程序

选择 File→Save 命令保存与项目有关的所有文件。此时首先会弹出 Save Unit1 As 对话框。定位到刚才创建的文件夹后,单击“确定”按钮,接着会弹出 Save Project1 As 对话框,单击“确定”按钮。

如果事先没有为新建的项目创建一个文件夹,可以在项目生成以后,通过选择 File→Save As 命令打开 Save As 对话框,在文件、目录列表框的空白位置处右击,在弹出的快捷菜单中选择“新建”→“文件夹”命令创建一个新的文件夹,或者单击 Save As 对话框中的“新建文件夹”按钮新建一个文件夹。

(3) 向窗体添加组件,并设置组件的相关属性。

为了在 Form1 窗体中显示文字,添加一个标签(TLabel)组件。标签组件是组件面板的 Standard 组件页中的第 4 个组件。将鼠标移动到 Standard 组件分类中的第 4 个组件上,稍停一下后会出现一个内容显示为 Controls. TControl 的提示窗口,如图 1-20 所示,表示该 .NET 组件所属的名字空间。单击,然后移动鼠标到 Form1 窗体中再单击一下,标签组件就被添加到窗体中了。

为了切换中英文显示内容,采用同样的方法,往 Form1 窗体中添加一个按钮(TButton)组件。按钮组件是 Standard 组件页中的第 7 个组件。

添加完成的标签组件和按钮组件如图 1-21 所示。



图 1-20 组件选取与提示

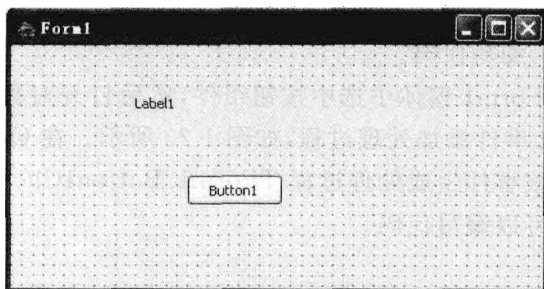


图 1-21 添加标签组件和按钮组件

首先改变窗体 Form1 的显示标题,单击窗体标题栏,选择窗体为当前设计对象,然后在对象观察器窗口中设置 Caption 属性,如表 1-1 所示。

表 1-1 窗体显示标题的属性设置

属 性	属 性 值	说 明
Caption	认识 Delphi 2007 程序	在窗体标题栏上显示的字符串

在 Form1 窗体中选中标签组件,然后在对象观察器窗口中设置标签组件的主要属性,如表 1-2 所示。

表 1-2 标签组件的属性设置

属 性	属 性 值	说 明
Alignment	taCenter	设置标签上显示的文字水平居中
AutoSize	False	使标签不根据 Caption 的长度自动改变宽度
Caption	欢迎学习 Delphi 语言!	在标签上显示的字符串
Font	字体: 宋体,大小: 小二	设置标签显示的文字效果
Name	Label1	为标签对象指定名称,便于在程序中调用

改变字体属性时,可以单击组合框右边的“...”按钮,在弹出的“字体”对话框中设置字体、大小和颜色等。也可以展开 Font 属性,对子属性进行设置。

设置好组件的属性后,可以改变窗体的大小。将鼠标移动到窗体的边缘时,鼠标的形状会变成双箭头模式,然后按住鼠标左键拖动就可以改变窗体的大小。

在 Form1 窗体中选中标按钮组件,然后在对象观察器窗口中设置按钮组件的主要属性,如表 1-3 所示。

表 1-3 按钮组件的属性设置

属 性	属 性 值	说 明
Alignment	taCenter	设置按钮上显示的文字水平居中
AutoSize	False	使按钮不根据 Caption 的长度自动改变宽度
Caption	切换到英文	在按钮上显示的字符串
Font	字体: 宋体,大小: 三号	设置按钮上显示的文字效果
Name	Button1	为按钮对象指定名称,便于在程序中调用

设计好的窗体如图 1-22 所示。

(4) 编写代码。

在 Form1 窗体中选中按钮组件,然后打开对象编辑器窗口中的事件页,为按钮组件的 OnClick 事件添加处理过程,如图 1-23 所示。在 OnClick 栏右边的组合框中双击,系统就会自动为该事件生成处理过程 TForm1.Button1Click,并自动将光标定位到事件处理过程处,然后就可以编写代码。

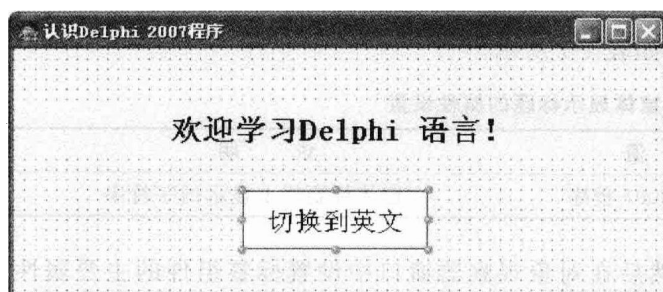


图 1-22 设置好的 Form1 窗体

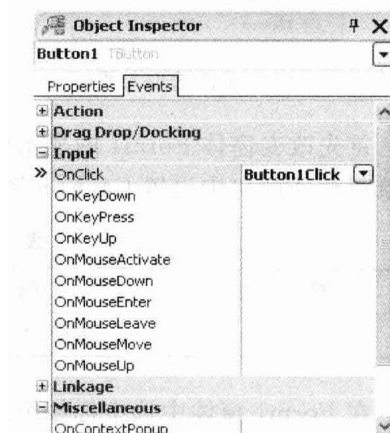


图 1-23 事件页中添加 OnClick 事件

在 TForm1.Button1Click 处理过程中添加以下代码:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if Label1.Caption = '欢迎学习 Delphi 语言!' then           //当前为中文显示
    begin
        Label1.Caption := 'Welcome to study Delphi language!';
        Button1.Caption := '切换到中文';
    end
    else                                                         //当前为英文显示
    begin
        Label1.Caption := '欢迎学习 Delphi 语言!';
        Button1.Caption := '切换到英文';
    end;
end;
```

(5) 运行程序。

选择 Run→Run 命令运行程序。单击目标应用程序中的按钮,可以交替地显示中英文标签。如果源程序没有经过编译和链接,在运行前系统会自动进行编译和链接,生成目标应用程序。

## 1.5 Delphi 2007 程序基本结构

### 1.5.1 项目和项目文件

Delphi 中,应用程序的所有相关文件都被组织在一个项目中,每个项目包含的文件很

多,大部分是由 Delphi 自动创建并维护的。在这些文件中,只需要关注项目文件、窗体文件和单元文件。

每一个 Delphi 应用程序都有一个扩展名为 .dpr 与 .dproj 的项目文件,是由 Delphi 自动建立并维护的,一般情况下不必修改它的内容。

项目文件是特殊的单元文件,可以理解为主单元文件或主程序。在 Delphi 中选择 File→Open Project 命令打开一个项目文件,就意味着装入整个项目。

一个项目是多个文件的集合,其中一些文件是设计项目时生成的,另一些文件则是在编译项目时形成的。一般情况下,不必对每个文件进行处理,尽管可以直接编辑大多数文件的源代码,但使用 Delphi 的集成开发环境及可视化工具更方便,也更可靠地自动维护这些文件。

在打开 Delphi 时,会自动建立一个项目文件,如果选择 File→Save Project As 命令保存整个项目文件,则可以看到该文件包括如下文件:项目文件(.dpr)、项目选项文件(.dof)、配置文件(.cfg)、资源文件(.res)、窗体文件(.dfm)和单元文件(.pas),其中前 4 个文件都使用项目的名字,如 project1;后两个文件使用单元的名字,如 Unit1。

每个项目都包括 Object Pascal 源代码,Delphi 将其编译为最终的应用程序或动态库,项目文件一般包含对使用的所有窗体和单元的引用,在装载、保存或编译一个项目时,Delphi 查看项目文件就知道各类文件的作用。项目文件最终会生成 .exe 文件(可执行文件,即应用程序)或 .dll 文件(动态链接库文件)。

**注意:** 由于 Delphi 本身维护项目文件,因此一般不需要手工修改它。对项目文件的改变也可以通过项目管理器实现,可以选择 View→Project Manager 命令打开项目管理器。

要查看项目文件的源代码,可以通过选择 Project→View Source 命令打开它,也可以选择 View→Units 命令或快捷按钮打开它。默认生成的项目源文件代码如下:

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

该源代码各部分的作用如下:

- program: 关键字,指出该项目是一个应用程序,其名字为 Project1。如果该项目是一个动态库,则会使用 library 关键字。
- uses: 该语句告诉编译器当前项目中使用了哪些单元,并将其链接到项目上,在窗体中添加新的组件时,与其有关的标准单元文件会自动添加到该语句中。所以,一般不需要对其修改。如果要引用自己建立的单元,可以再使用一个 uses 语句,并将其放置在{\$R \*.res}上面,这样源代码显得比较清晰,这种情况一般出现在窗体单元文件或单独的单元文件中。
- Unit1: 默认生成的单元标识符,它对应默认建立的窗体 Form1。

- Unit1.pas: 包含该单元源代码的文件名,它与单元标识符必须一致,否则项目无法正确被编译。所以如果修改了单元标识符,则也应以新的名字保存单元文件。
- in: 该关键字告诉编译器如何找到每个单元的源文件,注释{Form1}表示该单元文件对应窗体的名字。
- {\$R \*.res}: \$R 是一条编译器指令,它告诉编译器应该将与项目文件同名的资源文件 \*.res(\* 表示与当前的项目文件同名)链接到项目中,项目的资源文件包括项目的图标、图像等内容。
- begin...end: 该部分是当前项目的主要源代码块,其中 Application.Initialize 语句用于对应用程序初始化。Application.CreateForm(TForm1,Form1)语句则用于建立参数中指定的窗体。如果项目中有多个窗体,则也会有多条与其对应的这种语句。Application.Run 语句会运行整个应用程序。

### 1.5.2 窗体文件

窗体在设计阶段可以用来放置各种组件,在运行阶段是与用户交互的界面。窗体中的所有信息保存在两个主名相同扩展名不同的文件中,一个是扩展名为 dfm 的窗体文件,另一个是每个窗体对应的同名单元文件。

窗体文件一般以二进制的形式存储在 Delphi 中,其文件扩展名是 .dfm。可以在代码编辑器中打开一个窗体文件并修改这些数据的文本,将窗体文件转换为文本形式的方法是在窗体上面单击鼠标右键,然后从弹出的快捷菜单中选择 View as Text 命令。例如,在一个空白的窗体中执行 View as Text 命令显示如下内容:

```
object Form1: TForm1
  Left = 0
  Top = 0
  Caption = 'Form1'
  ClientHeight = 206
  ClientWidth = 339
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
end
```

如果要将文本形式窗体重新转换到以前的形式,则可以选择 View as Form 命令。单元文件中包括了处理窗体上各个组件相应事件的代码。

### 1.5.3 单元文件

单元是程序模块化的基础,类是继它之后才有的。在 Delphi 应用程序中,每个窗体都有一个相对应的单元。在工程中添加一个新窗体,实际上是增加了一个新单元,也就是建立

了该新窗体的类。

单元文件保存了 Delphi 程序的基本模块,一般的单元文件都与一个窗体对应,包含了窗体及其组件的事件处理程序,在 Delphi 中编写的程序代码,绝大多数被保存在这种文件中,其扩展名为 .pas。也可以单独建立跟窗体不关联的单元文件。

一个单元文件可以划分为接口和实现两部分,其中接口部分以 interface 关键字开始,到 implementation 关键字为止,该部分用于单元文件中使用的类或变量的声明等。implementation 关键字以后的代码是该单元文件中实现的功能,包括对各个组件事件的处理及实现不同功能的过程和函数等。

一般单元文件的源代码结构详细介绍如下:

```
unit Unit1;                                //单元文件的名字
interface                                  //接口部分
    {接口部分开始}
uses                                        //引用的标准单元文件
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs;
    {引用单元列表,这是可选的,如果包含必须紧跟 interface 关键字}
    {接口部分声明常量/类型/变量/过程和函数,这些声明对引用单元就像自己的声明一样}
    {在接口部分声明的过程和函数,就像使用了 forward 关键字}
type                                        //类型声明
    TForm1 = class(TForm)
    private                                //声明私有成员
        { Private declarations }
    public                                //声明公有成员
        { Public declarations }
    end;                                    //结束类型声明
var                                        //声明变量或类的实例
    Form1: TForm1;
    {接口部分结束}
implementation                              //程序代码实现功能部分的开始
    {实现部分}
uses
    {如果包含 uses 子句,必须紧跟关键字 implementation}
    {在这里实现 interface 中定义的过程和函数,可以任意顺序地定义和调用}
    {在这里可以省略过程和函数的列表,如果包括,必须一样}
    {可以定义单元私有的常量/类型(包括类)/变量/过程和函数,但这些对引用单元的客户是不可见的}
    {$R *.dfm}                            //通过编译指令 $R 链接窗体文件
    {如果是对应窗体的单元文件,会有这句。$R 指令用于加载一个外部资源文件,这里是指加载同名的窗体文件一起编译}
Initialization                              //窗体单元里一般没有这一部分
    {初始化部分}
    {程序启动时先执行,并顺序执行}
    {一个单元的初始化代码运行之前,就运行了它使用的每一个单元的初始化部分}
Finalization                                //窗体单元里一般没有这一部分
    {结束化部分,程序结束时执行}
end.                                        //实现部分结束
```

接口区头部的 uses 子句表示需要访问的外部单元,这些外部单元中定义了需要引用的



数据类型,如自定义窗体内所用的控件。

实现区头部的 `uses` 子句用于表示只在实现部分访问的单元。如果例程或方法的代码需要引用其他单元,应该把这些单元加到实现区子句中,而不是接口区。所引用的单元必须在工程文件目录中能找到,或在工程选项对话框的 Directories/Conditionals 页设定这些单元的搜索路径。

C++ 程序员应该知道 `uses` 语句和 `include` 指令是不同的。`uses` 语句只是用于输入引用单元的预编译接口部分,引用单元的实现部分在单元编译时才考虑。引用的单元可以是源代码格式(PAS),也可以是编译格式(DCU),但是必须用同一版本的 Delphi 进行编译。

在单元的接口区中可以声明许多不同的元素,包括过程、函数、全局变量及数据类型。在 Delphi 应用程序中,数据类型可能用得最频繁。每创建一个窗体,Delphi 会在单元中自动建立一个新的数据类型——类(class)。在 Delphi 单元中不仅能定义窗体;还能像传统单元一样,只包含过程及函数;还可以定义与窗体和任何可视控件无关的类。

#### 1.5.4 命名空间

Delphi 语言中的 `Uses` 语句是访问其他单元工作空间的标准技术,通过该语句能访问其他单元的定义内容。如果恰巧两个单元声明的标识符同名,也就是说可能有两个同名的类或例程,遇到这种情况,可以用单元名作前缀定义类型或过程名,由此进行区分。

例如用语句 `Kiln.FireOk` 访问 Kiln 单元中的 FireOk 过程。不过这种情况最好不要经常遇到,因此强烈建议不要在同一程序中用同一名字表示两个不同的变量、过程或函数等。

然而,如果查阅 VCL 库和 Windows 文件,会发现一些 Delphi 函数和 Delphi 可用的 Windows API 函数同名,不过参数往往不同,下面以 Beep 过程为例说明这个问题。

新建一个 Delphi 项目,添加一个按钮,然后写入以下代码:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Beep;                                //发出蜂鸣声音
end;
```

执行程序,单击该按钮会听到一个短促的声音。现在移到单元的 `uses` 语句,把原先的代码:

```
uses
    Windows, Messages, SysUtils, Classes, ...
```

修改一下,把 SysUtils 单元移到 Windows 之前:

```
uses
    SysUtils, Windows, Messages, Classes, ...
```

现在如果重新编译这段代码,Delphi 系统会提示一个编译错误:“Not enough actual parameters(实际参数不够)”。问题在于 Windows 单元定义了另一个带两个参数的 Beep 函数。应该说 `uses` 子句中第一个单元的定义被后面单元的定义覆盖,解决的办法就是给该过程限定单元名称,修改后的代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
    SysUtils.Beep;    //指定调用 SysUtils 的过程来发出蜂鸣声音
end;
```

这样,不管 uses 子句中单元顺序如何排列,以上代码都能编译通过,并且能够正确运行。在 Delphi 中很少有其他命名冲突的情况,因为 Delphi 代码通常放在类的方法中,如果不同类中有两个同名的方法不会产生任何冲突,只是使用全程例程会产生冲突问题。

### 1.5.5 Delphi 2007 的文件类型

Delphi 开发的应用程序或动态链接库是以项目(Project)的形式组织的,一个项目是多个文件的集合。其中一些文件是设计项目时生成的,另一些文件则是在编译项目时形成的。一般情况下,不必对每个文件进行处理,尽管可以直接编辑大多数文件的源代码和脚本。

在 Windows 的资源管理器中打开例 1.1 所保存的文件夹,可以看到 Delphi 创建了许多文件。表 1-4 列出了一些常见的由 Delphi 产生的文件类型。

表 1-4 Delphi 产生的主要文件类型

文件扩展名	文件类型说明	产生时间
BMP、ICO、CUR	位图、图标及光标图像文件	程序设计时
BGP	项目组文件,由多目标项目管理器产生	程序设计时
BPL	Borland Package Library(组件库文件)	编译链接后
CBA	压缩格式文件,做 Web 发布时使用	设计时
CFG	项目配置文件。项目配置文件保存着项目的配置信息	设计时
DCP	Delphi Component Package(Delphi 组件包)	编译时
DCU	Delphi Compiled Unit,编译原始文件后的中间产物	编译时
DFM	Delphi Form File(窗体文件)	程序设计时
~DFM	DFM 的备份文件	程序设计时
DLL	Dynamic Link Library(动态链接库文件)	编译链接时
DOF	Delphi Option File,设计多语言项目时使用的语言翻译配置文件,多语言项目中每个窗体的每一种语言都有一个 DOF 文件	程序设计时
DPK	Delphi Package,软件包项目的源代码文件	程序设计时
DPR, DPROJ	项目文件	程序设计时
~DPR	DPR 的备份文件	程序设计时
DSK	Desktop File,保存现在 Delphi 视窗的位置、正在编辑的文件以及其他桌面的设定文件	程序设计时
LIC	OCX 文件相关的授权文件	编译链接时
OCX	OLE 控件文件,是一个特殊的 DLL 文件,可包含 Activex 控件或窗体	编译链接时
PAS	Delphi 源代码文件	程序设计时
~PAS	PAS 的备份文件	程序设计时
RES、RC	项目的资源文件,包含项目的图标、光标及字体等信息	程序设计时
EXE	可执行文件	编译链接时
TLB	类型库文件	程序设计时

## 本章小结

本章主要介绍了 Delphi 2007 的产生历史以及软件安装过程。同时,还讲解了 Delphi 2007 的集成开发环境和应用开发程序的设计步骤,并通过一个简单的应用程序直观地为读者展示了编写应用程序的完整过程。通过第 1 章的讲解,希望读者能够熟练掌握 Delphi 2007 开发环境以及各种功能部件,为今后全面学习 Delphi 2007 带来方便。

## 思考与练习

1. 简述 Delphi 2007 for Win32 的安装要求与安装过程。
2. Delphi 2007 for Win32 集成开发环境由哪几部分组成? 简述各部分的功能。
3. 什么是项目(Project)? Delphi 中为什么要使用项目?
4. Delphi 单元文件主要由哪几个部分组成? 简述各部分的功能。
5. 模仿例 1.1,编写一个简单的窗体程序,显示如下内容:

```
*****  
    欢迎学习 Delphi 语言!  
*****
```

提示: 使用三个 TLabel 直接设置其 Caption 属性值。

## 第 2 章

## 数据类型、运算符和表达式

数据和运算符是程序的基本要素,数据是程序处理的对象,运算符是对数据进行处理的具体描述。Delphi 语言中数据是有类型的,不同类型的数据具有不同的运算性质,在计算机中的存储也是不同的。本章将详细介绍 Delphi 语言的数据类型、标识符、常量与变量的表示方法,各种运算符的功能及其运算规则、表达式和语句的书写方法等知识。

Delphi 2007 的编程语言是在 Object Pascal 的基础上发展起来的,它继承了 Object Pascal 语言语法结构严谨和编译代码高效优化等优点。Delphi 是一种面向对象的高级编程语言,具有可读性好、编写容易、支持结构化和面向对象编程等优点。

本章主要包括以下内容:

- 控制台程序设计;
- 标识符、常量与变量;
- 基本数据类型;
- 运算符与表达式;
- 构造数据类型。

### 2.1 控制台程序设计

控制台程序就是一个没有图形界面的 Windows 可执行文件。它可以接受用户的输入,然后给出结果。这一切操作都是在控制台上进行的,类似于 DOS 的命令窗口。可以通过按 Win+R 键(就是同时按键盘上有 Windows 标志的那个键和 R 键),然后输入“cmd”来打开控制台。

控制台程序指的是 console 下命令行方式的程序,任何操作系统都支持控制台程序,如 DOS、Windows、Linux 和 UNIX 等都支持。Delphi 2007 控制台是纯 API 下的 Win32 编程,它一般不涉及用户界面的设计,本章主要讲解 Delphi 2007 语法,所以采用控制台程序可以方便地阐述,便于读者理解。

Delphi 中控制台输入输出一般还要使用 Read、Readln、Write 和 Writeln 语句。

#### 2.1.1 控制台程序的建立

建立控制台程序的步骤如下:

(1) 选择“开始”→“程序”→CodeGear RAD Studio→Delphi 2007 命令,启动 Delphi 软件系统。在 Delphi 集成开发环境中选择 File→New→Other,如图 2-1 所示。

(2) 在弹出的图 2-2 所示 New Items 对话框中,选中控制台应用程序系统 Console Application。

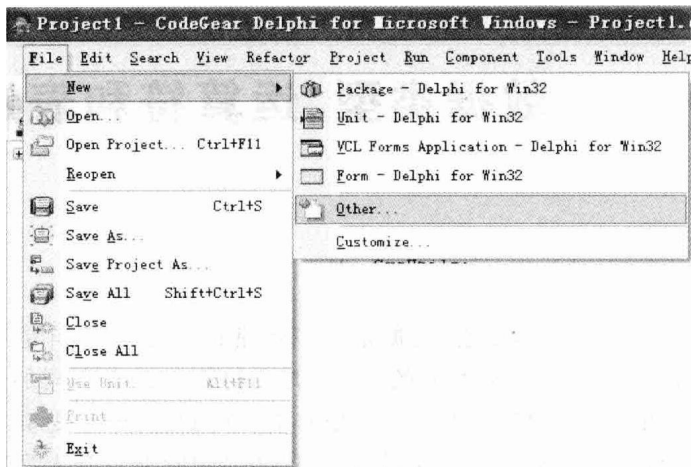


图 2-1 选择建立控制台程序的菜单

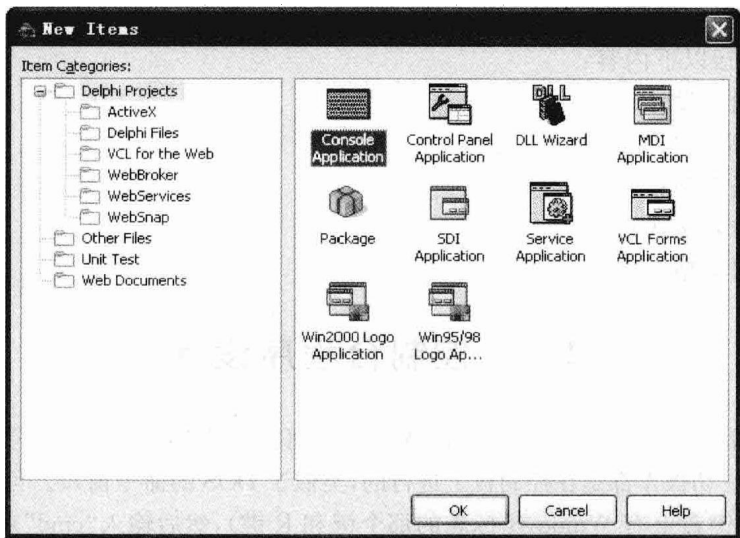


图 2-2 选择建立控制台程序的 New Items 对话框

(3) 选择好控制台应用程序系统后,单击 OK 按钮,建立一个名为 Project1 应用程序,用户可以在其中添加自己的程序代码,系统自动产生了很多代码,如图 2-3 所示。try...except...end 为系统异常处理结构,初学者可以暂时不用管它,只要记住该结构是控制台编程所必需的固定格式就行了。

如果觉得 try...except...end 影响视线,可以直接去掉它们,用户代码直接加入到 begin...end 之间。因为系统直接默认引用 SysUtils 单元,所以也可以去掉它的引用声明。

精简后的控制台代码如下:

```
program Project1;                                     //控制台程序名称 Project1
{$ APPTYPE CONSOLE}                                   //程序编译性质
begin
    //这里面添加用户程序代码——开始
```

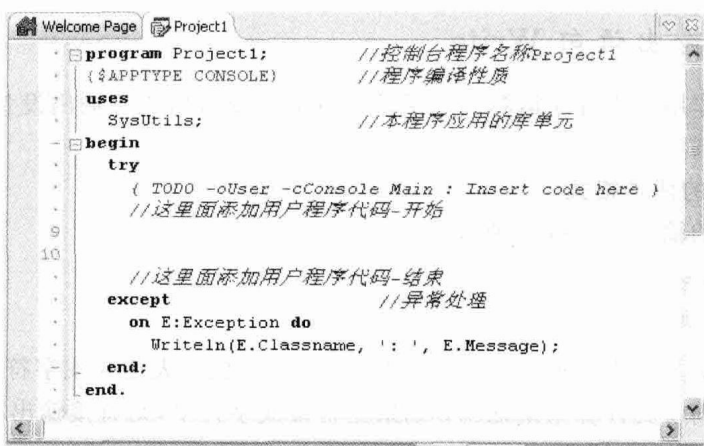


图 2-3 控制台程序系统产生的默认代码窗口

...  
//这里面添加用户程序代码——结束  
end.

(4) 编辑完程序代码后,选择 File→Save All 命令可以保存项目文件,或者选择 File 中的 Save As、Save Project As 命令更换名称保存。

(5) 选择 Run→Run 命令或者工具条上的快捷运行按钮,实现编译、链接、运行目标应用程序,如果有错,根据系统提示修改错误,然后再执行此过程,直到程序运行正确为止。

**例 2.1** 在显示器屏幕上输出文字信息:“欢迎使用 Delphi 2007!”,文字上下用多个“#”字符进行装饰。

实现这个功能的 Delphi 2007 语言源程序如下:

```

program Project1;           //控制台程序名称 Project1
{$APPTYPE CONSOLE}         //程序编译性质
begin
  //这里面添加用户程序代码——开始
  writeln('#####');        //输出符号
  writeln('欢迎使用 Delphi 2007! '); //输出问候语
  writeln('#####');        //输出符号
  readln;                  //输入语句,为了让程序暂停一下
  //这里面添加用户程序代码——结束
end.

```

程序运行结果如图 2-4 所示。

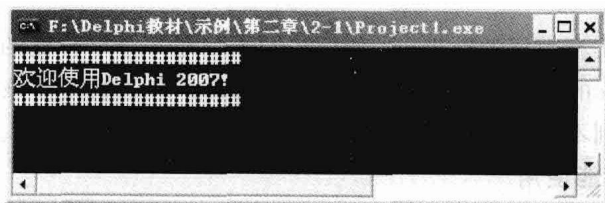


图 2-4 运行结果

## 2.1.2 基本输出语句 Write

输出语句的作用是将程序运算的结果输出到屏幕或打印机等输出设备。控制台应用程序通常是指输出到屏幕。

### 1. 输出语句的两种格式

Delphi 语言的输出语句有两种形式：

```
Write(<输出项表>);  
Writeln(<输出项表>);
```

其中<输出项表>是一串用逗号分隔的常量、变量、函数名、表达式或字符串。如果是变量、函数名、表达式,则将其计算结果输出;如果是常量或字符串,则直接输出其值。

#### 1) Write 语句

格式:

```
Write(表达式 1, 表达式 2, ...);
```

如:

```
Write(1, 2, 3, 4);  
Write(1.2, 3.4, 5);  
Write('My name is Liping');
```

#### 2) Writeln 语句

格式:

```
Writeln(表达式 1, 表达式 2, ...)或 Writeln;
```

Write 和 Writeln 的区别在于: Write 语句是输出项输出后不换行,光标停留在最后一项后; Writeln 语句按项输出后自动换行,光标则停留在下一行的开始位置。

Writeln 语句允许不含有输出项,即仅 Writeln; 表示换行。

Delphi 语言把输出项的数据显示占用的宽度称为域宽,可以根据输出格式的要求在输出语句中自动定义每个输出项的宽度。定义宽度时分为单域宽和双域宽。

单域宽输出格式:

```
Writeln(I: n);
```

在 n 个字符宽的输出域上按右对齐方式输出 I 的值,若 n 大于 I 的实际位数,则在 I 值前面补(n-I 的实际位数)个空格。若 I 的实际位数大于 n,则自动突破限制。n 必须是整数。

双域宽输出格式:

```
Writeln(a: n: m);
```

双域宽主要用于实型数据的输出。n 的用法同上。在 n 个字符宽的输出域上按右对齐方式用小数点形式输出 a 的数值,m 是小数点后的位数。原来的数据按该格式指定的小数位数四舍五入。若 m=0,则不输出小数部分和小数点,原数据四舍五入取整。n,m 必须是整数。

### 2. 输出语句的功能使用

计算机执行到某一输出语句时,先计算出输出语句中每个表达式的值,并将每一个表达式的值一个接一个地输出到屏幕上。

Write 语句与 Writeln 语句格式上都相似,但它们在功能上有所不同,两个语句的区别在于,Write 语句将其后括号中的表达式一个接一个输出后没有换行,而 Writeln 语句则在输出各个表达式的值后换行。

例如,执行以下两个程序段的输出分别为:

(1) Write(1,2,3,4);Write(5,6);

输出为:

123456

(2) Writeln(1,2,3,4);Write(5,6);

输出为:

1234

56

### 2.1.3 基本输入语句 Read

在程序中变量获得一个确定的值,固然可以用赋值语句,但是如果需要赋值的变量较多,或变量的值经常变化,则使用本节介绍的输入语句——读语句,将更为方便。

#### 1. 读语句的一般格式

读语句是在程序运行时由用户给变量提供数据的一种很灵活的输入动作,它有两种格式:

Read(<变量名表>);

Readln(<变量名表>);

<变量名表>是一个或几个由逗号隔开的变量标识符,它们必须在程序说明部分预先定义,可以是整型、实型或字符型,布尔型不可以直接读入。

#### 2. 读语句的使用

在控制台程序中,通过读语句可以从标准输入文件(即 INPUT,一般对应着键盘)中读入数据,并依次赋给相应的变量。

Read 和 Readln 是标准过程名,它们是标准标识符。执行到 Read 或 Readln 语句时,系统处于等待状态,等待用户从键盘上输入数据,系统根据变量的数据类型的语法要求判断输入的字符是否合法。

如执行 Read(a)语句,a 是整型变量,则输入的字符为数字字符时是合法的,当输入结束时,则自动将刚接收的一串数字字符转换为整数赋给变量 a。

在输入数值型(整型或实型)数据时,数据间要用空格或回车分隔开各个数据,输入足够个数的数据,否则仍要继续等待输入,但最后一定要有回车,表示该输入行结束,直到数据足够,该读语句执行结束,程序继续运行。

例如,设 a、b、c 为整型变量,其输入语句为 Read(a,b,c),系统执行该语句时,假设分别赋以 10,20,30,那么用户通过键盘操作可以按照如下的几种格式(说明:“□”这里表示空格,以下同):

(1) 10□20□30✓

(2) 10□20✓

30✓



```

(3) 10↵
    20□30↵
(4) 10↵
    20↵
    30↵

```

其中“↵”表示 Enter 键。

### 3. Read 与 Readln 的区别

Read 语句是一个接一个地读数据,在执行完本 Read 语句(读完本语句中变量所需的数据)后,下一个读语句接着从该数据输入行中继续读数据,也就是说不换行。

如:

```

Read(a,b);
Read(c,d);
Read(e);

```

如果输入数据行如下:

```
1□2□3□4□5□6□↵
```

则 a,b,c,d,e 的值分别为 1,2,3,4,5。如果后面无读语句,则数据 6 是多余的,这是允许的。

Readln 则不同,在读完本 Readln 语句中变量所需的数据后,该数据行中剩余的数据多余无用,或者说,在读完本 Readln 语句中变量所需数据后,一定要读到一个回车,否则多余的数据无用。

Readln 语句与 Read 语句的另一个重要区别是:Read 后一定要有参数表,而 Readln 可以不带参数表,即可以没有任何输入项,只是等待读入一个换行符(回车)。编程时通常在控制台程序的最后一行中放置一个不带参数的 Readln 语句,其目的是为了使用方便用户看运行结果。

**例 2.2** 已知矩形的两条边长分别是 a 和 b,求矩形的面积。

实现这个功能的 Delphi 2007 语言源程序如下:

```

program Project1;
{ $APPTYPE CONSOLE}
uses
  SysUtils;
var
  a,b:Real;           //长方形的长与宽变量定义
  area:Real;          //面积变量定义
begin
  try
    { TODO - oUser - cConsole Main : Insert code here }
    writeln('请输入长方形的长与宽: '); //输入提示信息
    readln(a,b);                       //通过输入语句读入长与宽数据值
    area := a * b;                     //根据公式计算面积
    writeln('面积为: ',area);          //输出面积数值
    readln;                            //输入语句,为了让程序暂停一下
  except
    on E:Exception do
      Writeln(E.Classname, ': ', E.Message);
    end;
  end.

```

代码中也可以去掉自动产生的代码 `uses` 与 `try...except...end` 结构。  
程序运行结果如图 2-5 所示。

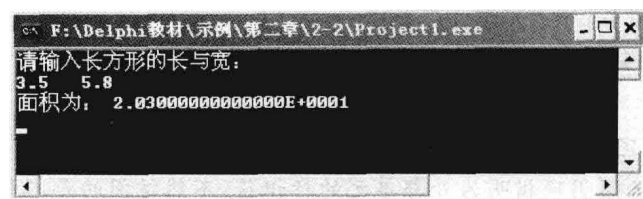


图 2-5 运行结果

## 2.2 标识符、常量与变量

### 2.2.1 标识符

在 Delphi 语言中,标识符是用来标识变量名、符号常量名、过程名、函数名、单元名及程序名、文件名等实体的有效字符序列。简单地说,标识符就是一个名字。标识符必须使用它字符集中的字符。

Delphi 语言的符号系统(Delphi 的字符集)如下:

- 英文字母: A—Z, a—z(不区分大小写)。
- 数字字符: 0—9。
- 运算符: +、-、\*、/、@、^等。
- 标点符号: ,、;、:、"和各种括号。
- 其他符号: \_ (下划线)、\$ (美元符号)和 # (井号)等。

标识符由一个或多个 ASCII 码字符序列组成,定义标识符的规则如下:

- (1) 标识符由字母、数字或下划线组成;
- (2) 标识符的第一个字符必须是字母或下划线;
- (3) 标识符的长度不应超过 255 个字符,超过 255 个字符只有前 255 个字符有效;
- (4) 不能将关键字(保留字)用作标识符;
- (5) 标识符不区分大、小写。

例如, `x`、`x34`、`max`、`a0`、`_mmm` 和 `Button_OK` 都是合法的标识符,而 `6y`、`z-y`、`α` 和 `ax6.5` 等都是非法的标识符。

由于标识符不区分大、小写,因此标识符 `Myname`、`MyName`、`myName` 和 `MYNAME` 是完全相同的。

然而大小写不敏感也有不便之处:第一,必须注意大小写不一致的标识符实际上是相同的,以避免把它们当成不同的标识符使用;第二,必须尽量保持大写使用的一致性,以提高代码的可读性。大写使用的一致性不是编译器强制要求的,但是保持大写使用的一致性是一种好习惯。

在 Delphi 语言中,有一类标识符是系统预先定义的,它们用于标识系统预先定义的标准函数、标准过程、标准类型、标准常量及标准文件等。

- 标准常量:如 `False`、`Maxint` 和 `True` 等。
- 标准类型:如 `Boolean`、`Char`、`String` 和 `TDateTime` 等。

- 标准函数：如 Abs、Eof、ShowMessage 和 Sqrt 等。
- 标准过程：如 Dispose、New、Read、Readln 和 Reset 等。
- 标准文件：如 Input、Output 等。

标准标识符是可以重新定义的。

**注意：**

- (1) 当程序中自定义的标识符与其引用单元中所定义的标识符重名时，如果要访问被引用单元的标识符，则必须明确指明为外部单元的标识符，不然访问的是本程序定义的标识符。
- (2) 因 System 单元是自动引用的，不必也不允许在引用部分列出 System 单元。

### 2.2.2 保留字

程序如同一篇文章，由字符组成单词，再由单词和符号构成句子——语句。Delphi 语言中具有特殊含义的单词称为保留字。

保留字又称为关键字，是一种预先定义的、具有特殊意义的标识符。用户不能重新定义关键字，也不能把关键字定义为一般的标识符，如关键字不能作变量名、函数名和过程名等。

Delphi 语言的关键字有类型标识符、控制流标识符、预处理标识符和其他标识符等，这些关键字是该语言保留的，用于实现如 Delphi 控制结构等不同特性。

表 2-1 中的关键字不能被重新定义或用作标识符。

表 2-1 Delphi 中使用的关键字

and	array	As	asm
begin	case	class	const
constructor	destructor	dispinterface	div
do	downto	else	end
except	exports	file	finalization
finally	for	function	goto
if	implementation	In	inherited
initialization	inline	interface	is
label	library	Mod	nil
not	object	Of	or
out	packed	procedure	program
property	raise	record	repeat
resourcestring	set	Shl	shr
string	then	threadvar	to
try	type	unit	until
uses	var	while	With
xor			

### 2.2.3 常量

所谓常量，是指在程序运行过程中其值保持不变的量。

常量也是数据，所以常量也应该有数据类型。Delphi 语言中常量的类型有 4 种：

- 整型常量：如 18、0、-9 等；

- 实型常量：如 3.6、-1.48 等；
- 字符常量：如 'a'、'c'、'8' 等；
- 字符串常量：如 'china'、'shaanxi' 等。

常量分为直接常量和符号常量。从其字面形式即可判断出的常量称为字面常量或直接常量，如 3.6、'a'、-5；用一个标识符来代表一个常量，则称之为符号常量。

符号常量定义的一般形式为：

```
const
    <常量标识符> = 表达式；
```

以保留字 const 后开始常量声明。“=”号左边为常量标识符；“=”号右边的表达式可以由常量、部分在程序编译时可计算的函数及先定义的常量标识符等构成，表示符号常量的值。

对于在程序运行期间保持不变的数据，Delphi 允许通过声明符号常量来调用。声明符号常量不必指定数据类型，但需指定常量所代表的数据的值。例如：

```
CONST
    Thousand = 1000;           //整型常量
    Pi = 3.14159;             //浮点类型常量
    ErrorMessage = '类型错误'; //字符串类型常量
```

Delphi 根据常量的值来决定它的数据类型。如果想告诉 Delphi 采用特定的类型，可在声明中加入类型名，方法如下：

```
CONST
    Thousand: Integer = 1000;
```

Delphi 对符号常量定义有如下要求：

- (1) 必须遵循先定义后使用的原则，即只有已定义的常量标识符才能在程序中使用。
- (2) 不能改变符号常量的值。

可用于常量定义的函数，即在程序编译时可计算的函数有 Abs、Chr、Hi、Length、Lo、Odd、Ord、Pred、Ptr、Round、SizeOf、Succ、Swap 和 Trunc 等。使用常量定义的意义在于减少常量值差错机会与修改程序的工作量，并提高程序的可读性。

**例 2.3** 符号常量的定义和使用：根据价格与数量计算总钱数。

```
program Project1;
{ $ APPTYPE CONSOLE}           //编译预处理指令,表示控制台程序
const
    PRICE = 10;                 //定义符号常量
var
    num: integer = 20;           //声明变量并初始化
    total: integer;              //声明总价变量
begin
    total := num * PRICE;        //计算
    writeIn(' 计算得到的 total 为: ', total); //输出
    readln;
end.
```

运行结果如图 2-6 所示。

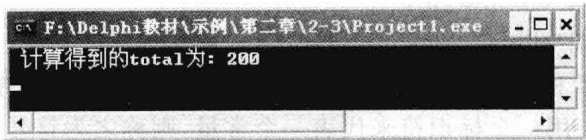


图 2-6 运行结果

在这个程序中,用 const 命令使 PRICE 这个标识符在整个程序中代表数字 10。定义后的 PRICE 则称为符号常量,即标识符形式的常量,它的值在程序运行期间不能改变。

习惯上,符号常量名用大写,变量用小写,以示区别。使用符号常量可以提高程序的可读性,便于修改,具有以下优点。

(1) 含义清楚,便于理解。如上面的程序中,看程序时从 PRICE 就知道它代表价格。因此,定义符号常量名应考虑“见名知意”。

(2) 修改方便,一改全改。在需要改变一个常量的值时,能做到“一改全改”。例如,上面的程序作如下修改:

```
Const
    PRICE = 15;
```

则在程序中出现的所有 PRICE 都代表 15。

**注意:** 符号常量是常量,不同于变量,它的值在其作用域内不能改变。程序代码中如再用下面的赋值语句对 PRICE 赋值是错误的。

```
PRICE := 30;
```

2.2.4 变量

在程序运行过程中,其值可以改变的量称为变量。一个变量应该有一个名字作为标识,变量名的命名必须遵循标识符规则。在命名时应考虑“见名知意”的原则。如用 sum 代表“总和”。

在使用变量前必须对它进行说明,即说明其存储的数据类型,并为其指定名称。变量说明以关键字 VAR 开头。一个变量说明部分可包含对多个变量的说明,每个变量说明用分号表示结束。

变量说明的一般形式为:

```
VAR
    变量名列表:类型名;
```

其中,“变量名列表”中可以包含一个或多个变量名,若有多个变量名时,相邻的两个变量名之间应使用逗号“,”隔开;“类型名”用以指定这些变量的数据类型。

例如:

```
VAR
    iCount: Integer;           //说明了一个整型变量
    bCorrect: Boolean;         //说明了一个布尔型变量
    cX,cY: Char;               //说明了两个字符型变量
```

实际上变量在它存在期间,在内存中占据一定的存储单元,在该存储单元中存放变量的值。变量名实际上是一个符号地址,在对程序编译连接时由系统给每一个变量分配一个内存地址,而在程序运行过程中从变量中取值,则实际上是通过变量名找到相应的内存地址,从其存储单元中读取数据。

变量也分为不同的类型,如整型变量、实型变量和字符型变量等。在 Delphi 语言中要求对所有用到的变量作强制定义,即应“先定义,后使用”;否则,在编译时会指出有关“出错信息”。这样做的好处是:

(1) 保证程序中变量使用不会发生错误,例如,如果在定义部分有 `var student:integer;`,而在执行语句中错写成了 `statent:=30;`,在编译时会检查出 `statent` 没有进行定义,可以避免变量名使用时出错。

(2) 对变量指定类型后,在编译时就能为其分配相应的存储单元。

(3) 一个变量类型确定后,也就确定了该变量所能进行的操作。例如,整型变量 `a` 和 `b` 可以进行求余运算: `a mod b`。`mod` 是“求余”,得到 `a div b` 的余数。如果将 `a`、`b` 指定为实型变量,则不允许进行“求余”运算。在编译时,可以根据其类型来检查该变量所进行的运算是否合法。

变量的值一般通过赋值运算符或调用标准输入语句获取。变量赋值的一般形式为:

变量名 := 表达式;

例如:

```
var a:integer;           //声明部分
a:=8;                   //begin...end 中的语句给 a 赋值
```

则是将整型数据 8 赋给整型变量 `a`,从而使变量 `a` 对应的存储单元中存放的数据为 8。也可以在定义变量时给变量赋值,称之为变量的初始化。

其一般形式为:

var 变量名: 类型符 = 表达式;

例如:

```
var
    num:integer = 20 * 30 + 5;           //声明变量并初始化
```

变量分为全局变量和局部变量,在过程或函数中说明的变量叫做局部变量,而在过程或函数之外说明的变量叫做全局变量。局部变量只在说明它的过程或函数中有效。

### 2.2.5 注释

注释是添加在程序中用来说明代码功能的语句,它是非执行语句,编译器编译程序时将注释忽略,对程序的执行部分不会产生任何影响,但它有助于提高程序的可读性与可维护性。

如果不给程序加上适当的注释,一段时间后可能就很难理清程序的流程与语句含义。建议初学者一开始就养成编程时给代码添加注释的良好习惯。

Delphi 中的注释通常有三种形式:

### 1. 花括号注释

该注释的形式为：组合符号“{”和“}”的成对使用，表示它们之间的内容是注释部分。例如：

{ 由一对花括号所包含的文字构成注释 }

### 2. 圆括号/星号注释

该注释的形式为：组合符号“( \* ”和“ \* )”的成对使用，表示它们之间的内容是注释部分。例如：

( \* 左圆括号加一个星号和一个星号加右圆括号之间的文字也构成注释 \* )

### 3. 双斜杠注释

该注释的形式与 C++ 风格注释一致：主要是符号“//”的单独使用，表示后面的内容是注释部分。例如：

//由两个斜杠开始直到这一行的结束，这里的文字是注释

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，例如前面的控制台程序例子中都有这么一行代码：

```
{ $ APPTYPE CONSOLE }      //编译器指令，表明代码为控制台程序标识
```

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。此形式比较适用于单行和少量几行注释的情况。

**注意：**相同类型的注释不要嵌套使用。虽然不同类型的注释进行嵌套在语法上是合法的，但不建议这样做。例如：

```
{ ( * 这是合法的注释 * ) }
( * { 这是合法的注释 } * )
( * ( * 这是非法的注释 * ) * )
{ { 这是非法的注释 } }
```

## 2.3 基本数据类型

数据类型是指定义了一组数据以及定义在这一组数据的操作，它是程序中最基本的元素。Delphi 是一种强类型语言，其变量在使用之前都要声明其数据类型。变量的数据类型决定了它能够存储数据的形式、数据的范围以及它能够进行的运算。在声明常量时，常量值本身就代表其类型，同时也决定了它所能参与的运算。

Delphi 数据类型十分丰富，类型大致可以分为简单类型、字符串类型、结构类型、指针类型、过程类型和变体类型。简单类型又分为有序类型和实数类型。本节仅对常用基本数据类型进行介绍，其他类型后续介绍。

### 2.3.1 有序类型

有序类型定义一个有次序的数值集合，除了它的第一个数值以外，其他每个数值都有一个唯一的前驱值。其特点是数据的分布是离散的，除了最后一个外，其他每个数值都有一个

唯一的后继值,并且每个数值都有一个序数决定它在这个类型中的位置。

Delphi 定义的有序数据类型有整型、字符型、布尔型、枚举型和子界型 5 种类型。常用的有序数据操作函数有 ord(功能:取字符 ASCII 数值或者确定序号)、chr(功能:将整型数转换成字符)、prec(功能:取前驱数据)、succ(功能:取后继数据)、high(功能:返回参数的上限值)和 low(功能:返回参数的下限值);常用过程有 inc(功能:自增)和 dec(功能:自减)。

1. 整型

整型数在计算机内部一般采用定点表示法,用于存储整型量(如 123,-7 等),存储整数的位数依机器的不同而异。

整型常量是用来表示一般数学中的整数,包括正整数、负整数和 0,所以整型常量也称整常数。

在 Delphi 语言中,整型常量可用十进制数和十六进制数两种形式表示。为识别不同进制表示整型常量,凡是以 \$ 开头的数字序列都是十六进制整数,如 \$10、\$ffff 等。

常见的整型取值范围和存储格式如表 2-2 所示。整型有通用整型和基本整型之分,基本整型主要有 Byte、Word、ShortInt、SmallInt 和 LongInt 等,通用整型包括 Integer 和 Cardinal。在程序中应当尽量使用通用整型,因为它们针对 CPU 和操作系统作了优化。

表 2-2 常见整型取值范围和存储格式

类 型	取 值 范 围	格 式
Byte	0~255	8 位无符号
Word	0~65 535	16 位无符号
ShortInt	-128~127	8 位有符号
SmallInt	-32 768~32 767	16 位有符号
LongInt	-2 147 483 648~2 147 483 647	32 位有符号
Integer	-2 147 483 648~2 147 483 647	32 位有符号
Cardinal	0~4 294 967 295	32 位无符号

注意:正整数可以在前面加“+”号,也可以不加“+”号。负整数的前面必须加“-”号。

2. 字符型

在内存中,字符数据以 ASCII 码存储,如字符'a'的 ASCII 码为 97。字符常量有两种表示方式:

(1) 用单引号对括起来的单个字符,如'a','\*'和'2'等。字符常量中的单引号只起分隔作用,称为字符常量的分隔符,它并不是字符常量的一部分。注意,'a'和'A'是不同的字符常量。

(2) 用#引导一个整数,整数表示该字符的 ASCII 码。例如,#13、#\$20、#\$30 和 #65 分别表示回车符、空格符、数字符号'0'和字母'A'。

后者还可用 Chr 函数表示为 Chr(78),该函数作用是将整数 78 转换成其 ASCII 值为 78 的对应字符。用 Ord 函数可作相反的转换 Ord('N')。

因单引号用于定界字符常量,所以用''''表示单引号字符。

一般来说,对字母、数字或符号如'5','@','+'等,用代表它们的本身符号来表示较好;而涉及特殊字符如是不可显字符特殊形式的常量时,用数字符号表示较好。下面列出了常



用的特殊字符：

- #9：跳格(Tab 键)
- #10：换行
- #13：回车(Enter 键)

字符有两种不同的存储格式类型：ANSIChar(或 Char)和 WideChar,如表 2-3 所示。第一种类型代表 8 位的字符,与 Windows 一直沿用的 ANSI(美国国家标准协会)字符集相对应；第二种类型代表 16 位的字符,与 Windows NT、Windows XP 和 2003 支持的双字节字符(Unicode)相对应。在 Delphi 中,Char 类型字符与 ANSIChar 一致。切记,不管在什么环境,前 256 个 Unicode 字符与 ANSI 字符是完全一致的。

表 2-3 通用字符类型

类 型	字 节 数	取 值 范 围
AnsiChar	1	ANSI 字符集
WideChar	2	Unicode 字符集
Char	1	相当于 AnsiChar

记住,因为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。Sizeof 函数返回类型或实例的字节长度。

字符型的数据只能是单个字符,不能是一串字符。例如'ABC'、'x=?' 等都不是字符型的数据,而是字符串。

字符和整数一样,当因为增加或减少而超过它的取值范围的开头或尾部时,它的值将回转(除非开启了边界检查)。

3. 布尔型(Boolean)

布尔类型又称为逻辑类型,其标识符为 Boolean。布尔型变量的取值仅有 False 和 True 两个值。布尔类型是有序类型。Delphi 中规定: False 的序数为 0,True 的序数为 1。

布尔类型变量一般用于存放关系表达式或者逻辑表达式的结果。

定义形式为：

```
var 变量名表:Boolean;
```

例如：

```
Var
  a,b:Boolean;                                //声明了逻辑类型变量 a,b
  c:integer = 10;
  d:integer = 5;
begin
  a := (c<d);                                //a 最后的值应该为 false
  b := ((1<2) and (c>d));                    //b 最后的值应该为 true
end;
```

4. 枚举类型(Enumerated)

程序不仅只用于数值计算,还更广泛地用于处理非数值的数据。例如,性别、月份、星期、颜色、单位名、学历和职业等都不是数值数据。

如果用一个数值来代表某一状态,这种处理方法不直观,易读性差。如果能在程序中用自然语言中有相应含义的单词来代表某一状态,则程序就很容易阅读和理解。也就是说,事先考虑到某一变量可能取的值,尽量用自然语言中含义清楚的单词来表示它的每一个值,这种方法称为枚举方法,用这种方法定义的类型称为枚举类型。

枚举类型是一种自定义有序类型。在枚举类型中列出了所有该类型可能的取值,而不是指定现有类型的范围。换句话说,枚举类型是个可取值的序列。声明枚举类型的语法如下:

```
TYPE
typeName = (val1, ..., valn);
```

其中 typeName 和 valn 是有效的标识符, typeName 是枚举类型名, valn 是枚举值的标识符,它不能再定义为其其他标识符。

下例定义了一个枚举类型 TWeekDay 来表示一周中的 7 天,并说明了一个变量 WeekDay 为 TWeekDay 类型。

```
TYPE
TWeekDay = (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);
VAR
WeekDay: TWeekDay;
```

根据定义类型时各枚举元素的排列顺序确定它们的序号,第一个枚举元素的序号为 0。例如,设有定义:

```
type days = (sun, mon, tue, wed, thu, fri, sat);
```

则

```
ord(sun) = 0, ord(mon) = 1, ord(sat) = 6; succ(sun) = mon, succ(mon) = tue, succ(fri) = sat; pred
(mon) = sun, pred(tue) = mon, pred(sat) = fri
```

应注意的是,枚举类型中的第一个元素无前趋,最后一个元素无后继。

使用枚举类型还要注意如下问题:

(1) 对枚举类型只能进行赋值运算和关系运算。

一旦定义了枚举类型及这种类型的变量,则在语句部分只能对枚举类型变量赋值,或进行关系运算,不能进行算术运算和逻辑运算。

在枚举元素比较时,实际上是对其序号的比较。当然,赋值或比较时,应注意类型一致。例如,设程序有如下说明:

```
type days = (sun, mon, tue, wed, thu, fri, sat);
colors = (red, yellow, blue, white, black, green);
var color: colors;
weekday: days;
```

则下列比较或语句是合法的。

```
weekday := mon;
if weekday = sun then write('rest');
weekday <> sun
```



而下面比较或语句是不合法的。

```
mon := weekday;
mon := 1;
if weekday = sun or sat then write('rest');
sun > red
weekday <> color
```

(2) 枚举变量的值只能用赋值语句来获得。

也就是说,不能用 read(或 readln)读一个枚举型的值。同样,也不能用 write(或 writeln)输出一个枚举型的值。如 write(red)是非法的,会发生编译错误。千万不要误认为该语句的结果是输出 red 三个字符。

但对枚举数据的输入与输出可通过间接方式进行。输入时,一般可输入一个代码,通过程序进行转换;输出时,也只是打印出与枚举元素相对应的字符串。

(3) 同一个枚举元素不能出现在两个或两个以上的枚举类型定义中。例如:

```
type color1 = (red, yellow, white);
color2 = (blue, red, black);
```

是不允许的,因为 red 属于两个枚举类型。

## 5. 子界类型(Subrange)

如果定义一个变量 i 为 small int 类型,那么 i 的值在微型机系统的 Delphi 语言中使用 2 字节的定义表示法,取值范围为 -32 768~32 767。而事实上,每个程序中所用的变量值都有一个确定的范围。

例如,人的年龄一般不超过 150 岁,一个班级的学生不超过 100 人,一年中的月数不超过 12 个,一个月中的天数不超过 31 天,等等。

如果能在程序中对所用变量的值域作具体规定的话,就便于检查出那些不合法的数据,这就能更好地保证程序运行的正确性。

子界类型就很好地解决了上面的问题。子界类型定义了某种类型的取值范围。声明形式如下:

```
TYPE
    类型标识符 = 上界值 .. 下界值;
```

定义子界类型时,不需要指定基本类型的名字,而只需提供该类型的上、下界值。所用基本类型必须是有序类型,定义结果将是另一种有序类型。例如:

```
TYPE
    TAge = 18..60;
    TCaps = 'A'..'Z';
VAR
    age: TAge;
    cap: TCaps;
```

变量 age 的有效值为 18~60 之间的所有整数,而变量 cap 的有效值为大写的英文字母。子界类型变量的值是有序的,对其进行增/减量操作都要在其定义范围内。

子界类型数据的运算规则：

(1) 凡可使用基类型的运算规则同样适用于该类型的子界类型。

例如,可以使用整型变量的地方,也可以使用以整型为基类型的子界类型数据。

(2) 对基类型的运算规则同样适用于该类型的子界类型。

例如,div,mod 要求参加运算的数据为整型,因而也可以为整型的任何子界类型数据。

(3) 基类型相同的不同子界类型数据可以进行混合运算。

例如,设有如下说明：

```
type
  a = 1..100;
  b = 1..1000;
  c = 1..500;
var
  x:a;
  y:b;
  t:c;
  z:integer;
```

则下列语句也是合法的：

```
Z := Sqr(x) + y + t;
```

下列语句：

```
t := x + y + z;
```

当  $x+y+z$  的值在 1~500 范围内时是合法的,否则会出错。

## 2.3.2 实数类型

实型也称为浮点型。浮点类型和整数不同的地方是浮点数采用的是浮点表示法,也就是说,浮点数的小数点位置不同,给出的精度也不相同。

实型常量包括正实数、负实数和 0。Delphi 中表示实型常量的形式有两种。

### 1. 十进制表示法

这是人们日常使用的带小数点的表示方法。

如 0.0、-0.0、+5.61、-8.0 和 -6.050 等都是实型常量,而 0.、.37 都不是合法的实数形式。

### 2. 科学记数法

科学记数法是采用指数形式的表示方法,如  $1.25 \times 10^5$  可表示成 1.25E+05。在科学记数法中,字母 E 表示 10 这个“底数”;而 E 之前为一个十进制表示的小数,称为尾数;E 之后必须为一个整数,称为“指数”。

如 -1234.56E+26、+0.268E-5 和 1E5 是合法形式,而 .34E12、2.E5、E5、E 和 1.2E+0.5 都不是合法形式的实数。

无论实数是用十进制表示法还是科学表示法,它们在计算机内的表示形式是一样的,总是用浮点方式存储。微型机系统中浮点方式 Real 类型存储一般占 8 个字节(64 位)内存空



间。与整型数据的存储方式不同,实型数据按指数形式存储。系统把一个实型数据分成小数部分和指数部分分别存放。指数部分采用规范化的指数形式。例如,实数 1.23456 在内存中的存放形式如表 2-4 所示。

表 2-4 实数存储示意

+	.123456	1
↑	↑	↑
数符	小数部分	指数

表中是用十进制数来示意的,实际上在计算机中是用十进制数来表示小数部分以及用 2 的幂次来表示指数部分的。

表 2-5 给出了基本实数类型的范围和存储格式。

表 2-5 实数类型

类 型	取 值 范 围	有 效 位 数	占 字 节 数
Real48	$\pm 2.9 \times 10^{-39} \sim 1.7 \times 10^{38}$	11~12	6
Single	$\pm 1.5 \times 10^{-45} \sim 3.4 \times 10^{38}$	7~8	4
Double, Real	$\pm 5.0 \times 10^{-324} \sim 1.7 \times 10^{308}$	15~16	8
Extended	$\pm 3.6 \times 10^{-4951} \sim 1.1 \times 10^{4932}$	19~20	10
Comp	$-2^{63}+1 \sim 2^{63}-1$	19~20	8
Currency	$-922337203685477.5808 \sim 922337203685477.5807$	19~20	8

实数类型代表不同格式的浮点数。Single 类型占的字节数最少,为 4 个字节;其次是 Double 浮点类型,占 8 个字节;接着是 Extended 浮点类型,占 10 个字节。这些不同精度的浮点数据类型都与 IEEE(电气和电子工程师协会)标准的浮点数表示法一致,并且 CPU 数字协处理器直接支持这些类型,处理速度也最快。

另外还有两种奇怪的数据类型:Comp 类型和 Currency 类型,Comp 类型用 8 个字节描述非常大的整数(这种类型可支持带有 18 位小数的数字);Currency 类型(16 位版的 Delphi 不支持该类型)表示一个有 4 位小数位的值,它的小数位长度是固定的,同 Comp 类型一样也占 8 个字节。正如名字所示,Currency 数据类型是为了操作很精确的 4 位小数货币数值才添加的。

2.3.3 日期时间类型

Delphi 也用实型数表示日期和时间数据。但为了更准确起见,Delphi 特别定义了 TDateTime 数据类型,这是一个浮点类型,因为这个类型必须足够宽,使变量能容纳年、月、日、时、分和秒、甚至毫秒。日期值按天计数,从 1899-12-30 开始,放在 TDateTime 类型的整数部分;时间值则位于十进制数的小数部分。

TDateTime 不是编译器可直接识别的预定义类型,它在 System 单元定义:

```
type
  TDateTime = type Double;
```

使用 TDateTime 类型很简单,因为 Delphi 为该类型定义了一系列操作函数,表 2-6 列

出了这些函数。

表 2-6 TDateTime 类型系统函数

例 程 函 数	作 用
Now	返回当前日期及时间
Date	返回当前日期
Time	返回当前时间
DateTimeToStr	按默认格式将日期和时间值转换为字符串。特定格式转换可用 FormatDateTime 函数
DateTimeToString	按默认格式将日期和时间值复制到字符串缓冲区
DateToStr	将 TDateTime 值的日期部分转为字符串
TimeToStr	将 TDateTime 值的时间部分转为字符串
FormatDateTime	按特定格式将日期和时间值转换为字符串
StrToDateTime	将带有日期和时间信息的字符串转换为 TdateTime 类型值,如串有误将引发一个异常
StrToDate	将带有日期信息的字符串转换为 TDateTime 类型格式
StrToTime	将带有时间信息的字符串转换为 TDateTime 类型格式
DayOfWeek	根据传递的日期参数计算该日期是一星期中的第几天
DecodeDate	根据日期值返回年、月、日值
DecodeTime	根据时间值返回时、分、秒、毫秒值
EncodeDate	组合年、月、日值为 TDateTime 类型值
EncodeTime	组合时、分、秒、毫秒值为 TDateTime 类型值

## 2.3.4 字符串类型

字符串表示一个字符序列。Delphi 共有三种字符串类型：

- ShortString: 短字符串类型也就是传统 Pascal 字符串类型。这类字符串最多只能有 255 个字符。短字符串中的每个字符都属于 ANSIChar 类型(标准字符类型)。
- ANSIString: 长字符串类型就是 Delphi 新增的可变长字符串类型。这类字符串的内存动态分配,引用计数,并使用了更新前拷贝(copy-on-write)技术。这类字符串长度没有限制(可以存储多达 20 亿个字符),其字符类型也是 ANSIChar 类型。
- WideString: 长字符串类型与 ANSIString 类型相似,只是它基于 WideChar 字符类型,WideChar 字符为双字节 Unicode 字符。

如果只简单地用 String 定义字符串,那么该字符串可能是短字符串,也可能是 ANSI 长字符串,这取决于 \$H 编译指令的值,\$H+(默认)代表长字符串(ANSIString 类型)。长字符串是 Delphi 库中控件使用的字符串。

## 2.3.5 可变类型

可变类型(Variant)是指可以在程序运行期间确定或改变的数据类型。这些数据在编译期间不能确定其数据类型,而且它们比固定类型的数据占用更多的存储空间和更多的操作时间。

默认的情况下,可变类型可以是除了记录类型、集合类型、静态数组类型、文件类型、类类型和指针类型之外的任何类型,也就是说,可变类型可以是除了构造类型和指针类型之外

的任何其他类型。

有如下变量声明：

```
Var  v:Variant;
      i:Integer;
      str:String;
```

则以下变量的使用是合法的：

```
i:= 10 ;
v:= i;
str:= v;
```

### 2.3.6 类型转换

在 Delphi 程序中,只有赋值号两端的数据类型一致或相容才可以进行赋值的操作。有时在一个表达式中包含有各种类型的数据,这就需要将不同类型的操作数转换为同一类型的数据,使所得到的结果只能是某一类型的数据。

强制类型转换就是将一种类型的变量当作另一种类型。因为 Delphi 是对数据类型要求严格的面向对象程序设计语言,编译器检查一些语句的数据类型是非常严格的,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种类型。

强制类型转换的格式如下：

类型标识符(变量名);

例如,有以下程序段：

```
Var
  c1:char;
  b1:byte;
begin
  c1 := 'a';
  b1 := byte(c1);           //将字符型强制转换成 8 位整型
End
```

另外,类型相容的数据之间可以进行关系运算,不需要强制类型转换。类型相容是赋值相容的前提,也是进行数据运算的前提。在 Delphi 中,两种数据类型只有满足下列条件之一时才是相容的。

- (1) 两种类型的数据一致。
- (2) 两种类型的数据都是实型。
- (3) 两种类型的数据都是整型。
- (4) 一种数据类型是另一种数据类型的子界。
- (5) 两种数据类型是另外一种宿主类型的子界。
- (6) 两种数据类型都是另外一种相容基类型的集合类型。
- (7) 两种数据类型都是紧凑字符串类型,并且具有相同的元素个数。
- (8) 一种数据类型是字符串类型,另一种数据类型是字符串类型、紧凑字符串类型或字符类型;或者一种数据类型是 Char 类型,另一种数据类型是形式为 `array[0..n] of char` 的字符数组。

## 2.4 运算符与表达式

在程序中,表达式是计算求值的基本单位,它是由运算符和运算数组成的式子。运算符是表示进行某种运算的符号。运算数包含常量、变量和函数等。

运算是针对数据进行计算的过程,记述各种不同运算的符号称为运算符。根据运算规则,用运算符将常量、变量、数值和函数组合起来就形成表达式,表达式运算的结果就是表达式的返回值。表达式可以传递给过程或函数的值参,但不能传递给过程或函数中的引用参数。

### 2.4.1 表达式

最简单的表达式是变量和常量,更复杂的表达式由简单表达式使用运算符、函数调用、集合构造器、索引和类型转换构成。也就是说,表达式是变量、常量、字符串、运算符及函数按照一定规则的组合。

例如:

X	//变量
15	//整数常量
abs(X)	//函数调用
X * Y	//X 和 Y 的乘积
X >= Y	//条件表达式
['a', 'b', 'c'] { 集合 }	
Char(48) { 类型转换 }	

### 2.4.2 运算符

运算符是在代码中对各种数据类型进行运算的符号。例如,有能进行加、减、乘、除的运算符,有能访问一个数组的单个单元地址的运算符。表达式由运算对象和运算符两部分组成。不同数据类型的数据所能进行的运算是不同的,下面分别对各种运算符及其功能进行介绍。

根据运算符给定的操作数数据类型,可以将其分为赋值运算符、算术运算符、关系运算符、逻辑运算符、集合运算符、位运算符和其他运算符。

#### 1. 赋值运算符

赋值语句格式:

变量 := 表达式;

其中,“:=”称为赋值号。赋值运算符“:=”是先计算赋值运算符右边表达式的值,再将结果赋给左边的变量。Delphi 语言中赋值运算符“:=”不能漏掉“:”,它不是表示“等同”的关系,而是进行“赋值”的操作。例如:

- x := 12;
- y := 5 \* x + 6;
- str1 := 'w';



注意：

- (1) 赋值符号“:=”就是赋值运算符，它的作用是将一个数据赋给一个变量。例如，赋值表达式  $m := m + 1$  表示取变量  $m$  中的值加 1 后再放入到变量  $m$  中，使变量  $m$  的值增 1。
- (2) 赋值运算符的左边只能是变量而不能是常量或表达式。如  $a + b := c$  这个式子就不是合法的赋值表达式。因为系统已为  $a, b, c$  分别分配一一对应的存储单元，内存中并无  $a + b$  这样的存储单元。

2. 算术运算符

算术运算符对浮点数和整数进行加、减、乘、除和取模运算，如表 2-7 所示。

表 2-7 算术运算符

运 算 符	作 用	操作数类型	结 果 类 型
+	表示正值	Integer 或 Real	Integer 或 Real
	算术加	Integer 或 Real	Integer 或 Real
-	表示负值	Integer 或 Real	Integer 或 Real
	算术减	Integer 或 Real	Integer 或 Real
*	算术的乘运算	Integer 或 Real	Integer 或 Real
/	浮点数的除运算	Integer 或 Real	Real
Div	整型数的除运算	Integer	Integer
Mod	整型数的模运算	Integer	Integer

其中“+”和“-”运算符还可以作为单目运算符，放在浮点数或整数前，分别表示正数和负数。此外，“+”运算符还可以用在字符串的运算表达式中，可以将两个字符串连在一起。“+”、“-”和“\*”运算符还可以用在集合运算表达式中。

在进行“+”、“-”、“\*”运算中，只有一个运算分量为 Real 类型，则结果就为 Real 类型；只有两个运算分量都为 Integer 类型时，结果才为 Integer 类型。

- 加法运算符“+”：应有两个量参与运算，为双目运算符。如  $5 + 6$ 、 $b + c$  等。
- 减法运算符“-”：双目运算符。如  $6 - 4$ 、 $y - 1$  等。但“-”也可作负值运算符，如  $-x$ 、 $-5$  等，此时为单目运算符，具有右结合性。
- 乘法运算符“\*”：双目运算符。如  $6 * 8$ 。
- 除法运算符“/”：双目运算符，其结果为浮点数。如  $4/3$  结果为 1.333， $8/3$  结果为 2.667， $6.0/4$  结果为 1.5。
- 除法运算符 div：双目运算符。要求参与运算的两个数为整型。其结果为整数。如  $4 \text{ Div } 3$  结果为 1， $8 \text{ Div } 3$  结果为 2。
- 求余运算符 Mod：也称模运算符，为双目运算符。求余运算要求参与运算的量均为整型，运算的结果等于两数相除后的余数。如  $6 \text{ Mod } 4$  的值为 2。求余运算符 Mod 不能用于 Real 和 double 类型。

## 例 2.4 算术运算符与算式表达式综合举例。

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
  n,m,t,r:integer;
  x,y,z:real;
begin
  n:=20;m:=6;x:=2.5;                                //给三个变量赋值
  r:=n mod m;                                         //将 n 除以 m 所得的余数赋给 r
  z:=n+x;                                             //求和
  t:=n div m;                                         //求相除结果(整数)
  y:=n/m;                                             //求相除结果(实数)
  writeln('r=',r);
  writeln('z=',z);
  writeln('t=',t);
  writeln('y=',y);
  readln;
end.
```

运行结果如图 2-7 所示。

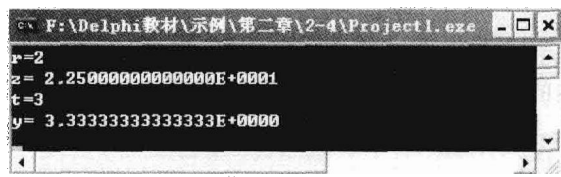


图 2-7 运行结果

算术表达式是指用算术运算符和括号将运算对象(也称操作数)连接起来符合 Delphi 语法规则的式子。这里所说的运算对象包括常量、变量和函数等。 $a * b / c - 1.5 + \text{ord}('a')$  就是一个合法的 Delphi 算术表达式。

对运算符的优先级和结合性,Delphi 语言有专门的规定。

(1) 在表达式求值时,先按运算符的优先级别高低次序执行。例如先乘除后加减。如  $a - b * c$ ,  $b$  的左侧为减号,右侧为乘号,而乘号优先于减号,因此相当于  $a - (b * c)$ 。

(2) 如果在一个运算对象两侧的运算符的优先级别相同,则按规定的“结合方向”处理。

Delphi 规定了各种运算符的结合方向(结合性),算术运算符的结合方向为“自左至右”,即先左后右。例如  $a - b + c$ ,运算时  $b$  先与减号结合,执行  $a - b$  的运算,再执行加  $c$  的运算。“自左至右的结合方向”又称“左结合性”,即运算对象先与左边的运算符结合。以后可以看到有些运算符的结合方向为“自右至左”,即右结合性(例如赋值运算符)。

如果一个运算符两侧的数据类型不同,则会先自动进行类型转换,使二者具有同一种类型,然后进行运算。

### 3. 关系运算符

关系运算符可以对两个普通数据类型、类、对象、接口类型或字符串类型的数据进行比

较,结果数据类型为布尔类型。关系运算符如表 2-8 所示。

表 2-8 关系运算符

运 算 符	作 用	结 果 类 型	运 算 符	作 用	结 果 类 型
=	等于	Bealoon	<=	小于或等于	Bealoon
<>	不等于	Bealoon	>=	大于或等于	Bealoon
<	小于	Bealoon	In	属于	Bealoon
>	大于	Bealoon			

关系运算符用来比较两个操作数的大小,被比较的两个操作数类型必须相容,若操作数是数值,则按照数值的大小进行比较;若操作数是字符(串),则根据字符的先后顺序按照 ASCII 值进行比较。例如:

```
'a'>'A'           //结果为 True. 'a'的 ASCII 值为 97, 'A'的 ASCII 值为 65
'15'>'A'          //结果为 False
```

用关系运算符将两个表达式(可以是算术表达式或关系表达式、逻辑表达式)连接起来的式子称为关系表达式。例如下面都是合法的关系表达式:  $a>b$ ,  $a+b>b+c$ ,  $(a=3)>(b>=5)$ ,  $'a'<'b'$ ,  $(a>b)>(b<c)$ 。

关系表达式的值是一个逻辑值,即“真”或“假”。以 True 代表“真”,以 False 代表“假”。若关系表达式表达的关系成立,则它的结果值为 True; 否则为 False。例如:  
 $x=y$   $x$  等于  $y$  时,关系表达式成立,其值为 True;  $x$  不等于  $y$  时,关系表达式不成立,其值为 False。

$X<>y$   $x$  不等于  $y$  时,关系表达式成立,其值为 True; 否则,关系表达式不成立,其值为 False。

$a:=x>y$  等价于  $a:=(x>y)$ , 若  $x>y$ , 则  $a:=True$ ; 否则  $a:=False$ 。

关系表达式常用于流程控制中作分支或者循环的条件,关系运算符的结合方向为自左至右。

4. 逻辑运算符

Delphi 语言用 and 和 or 作为逻辑与和逻辑或运算符。Delphi 的逻辑非运算符是 not, 用来对一个布尔表达式取反。

逻辑运算符对逻辑类型的操作数进行运算,数据结果也为逻辑类型,如表 2-9 所示。

表 2-9 逻辑运算符

运 算 符	作 用	操作数类型	结 果 类 型
not	逻辑非	Bealoon	Bealoon
and	逻辑与	Bealoon	Bealoon
or	逻辑或	Bealoon	Bealoon
xor	逻辑异或	Bealoon	Bealoon

逻辑运算符对逻辑类型的操作数进行运算,结果为布尔型。

- not(逻辑非): 将逻辑结果取反,即原先为 True 的变成 False,原先为 False 的变成 True

- and(逻辑与): 有且仅有两个操作数为真,结果才为真; 否则为假。相当于汉语中“并且”的意思,只有当两个条件同时满足时,结果才为 True。
- or(逻辑或): 仅当一个操作数为真时,值为真; 否则为假。
- xor(逻辑异或): 当两个操作数不同时,即一个为 True,另一个为 False 时结果为真; 同时为 True 或同时为 False 时为假。

逻辑表达式是用逻辑运算符将关系表达式或逻辑量连接起来的式子。

**例 2.5** 关系运算符、逻辑运算及其表达式综合举例。

```
program Project1;  
{ $APPTYPE CONSOLE}           //编译预处理指令,表示控制台程序  
var  
    x,y: Integer;  
    b1,b2,b3: Boolean;  
begin  
    x := 10;                     //给 x 赋值 10  
    y := 20;                     //给 y 赋值 20  
    b1 := not(x > y);            //逻辑非  
    b2 := x + 5 = y - 1;        //复杂的表达式  
    b3 := b1 and b2;            //逻辑与  
    writeln('b1 = ',b1);  
    writeln('b2 = ',b2);  
    writeln('b3 = ',b3);  
    readln;  
end.
```

运行结果如图 2-8 所示。

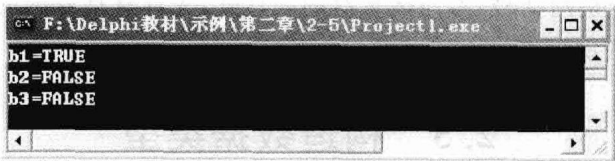


图 2-8 运行结果

5. 集合运算符

集合运算符只对两个集合进行操作,判断两个集合之间的关系,如表 2-10 所示。

表 2-10 集合运算符

运 算 符	作 用	操作数类型	结 果 类 型
+	集合的并集	集合类型	集合类型
-	集合的差集	集合类型	集合类型
*	集合的交集	集合类型	集合类型
<=	A<=B,A 是否是 B 的子集	集合类型	Bealoon
>=	A>=B,B 是否是 A 的子集	集合类型	Bealoon
=	两个集合是否相等	集合类型	Bealoon
<>	两个集合是否不相等	集合类型	Bealoon
in	s in A,s 是否属于集合 A	有序类型,集合类型	Bealoon

例如,有以下说明和集合的运算:

```
type Numbers = 1...100; Numset = set of Numbers;
var set1, set2, set3, set4, set5: Numset;
begin
  set1 := [1, 3, 5, 7];
  set2 := [2, 4, 5, 8];
  set4 := [1, 3];
  set5 := [2, 5];
  set3 := set1 + set2;
end;
```

那么 set3 值应该为[1,3,5,7,2,4,8]。

6. 运算符的优先级

在各类表达式混合运算当中,由于存在不同的运算符,那么要确定运算符运算优先级的顺序。表 2-11 列出了 Delphi 各运算符的优先级。

表 2-11 常见运算符的优先级

运 算 符	优 先 级	运 算 符	优 先 级
Not	1(最高)	+, -, or, xor	3
*, /, div, mod, and	2	关系运算符	4(最低)

表达式在求值时,遵循以下原则:

- (1) 两个操作符之间的操作数总是先参加高优先级的运算。
- (2) 在优先级相等的情况下,操作数按从左到右的顺序参加运算。
- (3) 有括号(只能是圆括号)参与运算符的表达式,先计算括号内的表达式值,有多对括号时,括号由内到外依次运算。

2.5 构造数据类型

Delphi 中构造数据类型是能够描述和扩展定义较复杂数据的存储类型,通过构造类型,程序中可以存储结构复杂的数据。

构造类型可通过将多种基本数据类型结合在一起,也可以通过扩展预先定义的数据类型来得到。

构造数据类型有下面几类: 集合类型(set)、数组类型(array)、记录类型(record)、文件类型(file)、类类型(class)、类引用类型(class reference)和接口类型(interface)。

在使用构造类型时,Delphi 语言提供了一种压缩机制,可以最大限度地节省构造类型的存储空间。在声明构造类型时引用保留字 packed,数据将以压缩方式存储,但是,这样延缓了数据的访问速度,还会产生类型的兼容性问题。

2.5.1 集合类型

集合是由具有某些共同特征的元素构成的一个整体。在 Delphi 中,一个集合是由具有同一有序类型的一组数据元素所组成,这一有序类型称为该集合的基类型。

## 1. 集合声明

集合类型的一般形式为：

```
set of <基类型>;
```

说明：

(1) 基类型可以是任意顺序类型，而不能是实型或其他构造类型。同时，基类型的数据的序号不得超过 255。

例如，下列说明是合法的：

```
type letters = set of 'A'..'Z';
numbers = set of 0..9;
s1 = set of char;
ss = (sun, mon, tue, wed, thu, fri, sat);
s2 = set of ss;
```

如下声明是非法的：

```
type
    TSetA = set of Integer;    //错误
    TSetB = set of 255..300;    //错误
    TSetC = set of WideChar;    //错误
```

(2) 与其他自定义类型一样，可以将类型说明与变量说明合并在一起。例如：

```
type numbers = set of 0..9;
var s: numbers;
```

与

```
var s: set of 0..9;
```

等价。

## 2. 集合的值

集合的值是用“[”和“]”括起来，中间为用逗号隔开的若干个集合的元素。例如[]（空集）、[1,2,3]和['a','e','i','o','u']都是集合。

说明：

- (1) 集合的值放在一对方括号中，各元素之间用逗号隔开。
- (2) 在集合中可以没有任何元素，这样的集合称为空集。
- (3) 在集合中，如果元素的值是连续的，则可用子界型的表示方法表示。例如，[1,2,3,4,5,7,8,9,10,15]可以表示成[1..5,7..10,15]。
- (4) 集合的值与方括号内元素出现的次序无关。例如，[1,5,8]和[5,1,8]的值相等。
- (5) 在集合中同一元素的重复出现对集合的值没有影响。例如，[1,8,5,1,8]与[1,5,8]的值相等。
- (6) 每个元素可用基类型所允许的表达式来表示。如[1,1+2,4]、[ch]和[succ(ch)]。

## 3. 集合的运算

### 1) 赋值运算

只能通过赋值语句给集合变量赋值，不能通过读语句赋值，也不能通过 Write(或 Writeln)语句直接输出集合变量的值。

## 2) 集合的并、交、差运算

可以对集合进行并、交、差三种运算,每种运算都只能有一个运算符、两个运算对象,所得结果仍为集合。三种运算符分别用“+”、“\*”和“-”表示。注意它们与算术运算的区别。

例如,x 为[1,3,5],y 为[3,4,5],则  $x * y$  为[3,5], $x + y$  为[1,3,4,5], $x - y$  为[1]。

## 3) 集合的关系运算

集合可以进行相等或不相等、包含或被包含的关系运算,还能测试一个元素是否在集合中。所用的运算符分别是=、<>、>=、<=和 in。它们都是二目运算,且前 4 个运算符的运算对象都是相容的集合类型,最后一个运算符的右边为集合,左边为与集合基类型相同的表达式。

例如,表达式  $[1,3] \geq [3]$  的值为 True;  $3 \text{ in } [3,5]$  的值为 True。

## 2.5.2 数组类型

数组类型数据表示的是同种类型数据的集合。数组类型的数据是排列有序的,每个数据元素都有一个唯一的索引号。与集合类型不同的是,数组类型的数据可以重复。数组的维数是指数组元素下标的个数。根据数组的维数可以将数组分为一维、二维、三维和 multidimensional 数组。

### 1. 一维数组的定义

#### 1) 类型定义

要使用数组类型以及前面要学习的自定义类型(枚举类型与子界类型)等构造类型,应在说明部分进行类型说明。这样定义的数据类型适用于整个程序。

类型定义的一般格式为:

```
type
<标识符 1> = <类型 1>;
<标识符 2> = <类型 2>;
:
<标识符 n> = <类型 n>;
```

其中 type 是 Delphi 保留字,表示开始一个类型定义段。在其后可以定义若干个数据类型定义。<标识符>是为定义的类型取的名字,称为类型标识符。

类型定义后,也就确定了该类型数据取值的范围以及数据所能执行的运算。

#### 2) 一维数组类型的定义

一维数组类型的一般格式:

```
array[下标 1..下标 2] of <基类型>;
```

说明: 其中 array 和 of 是 Delphi 保留字。下标 1 和下标 2 是同一顺序类型,且下标 2 的序号大于下标 1 的序号。它给出了数组中每个元素(下标变量)允许使用的下标类型,也决定了数组中元素的个数。基类型是指数组元素的类型,它可以是任何类型,同一个数组中的元素具有相同类型。因此可以说,数组是由固定数量的相同类型的元素组成的。

再次提醒注意: 类型和变量是两个不同概念,不能混淆。就数组而言,程序的执行部分使用的不是数组类型(标识符)而是数组变量(标识符)。

一般在定义数组类型标识符后定义相应的数组变量,如:

```
type arraytype = array[1..8] of integer;
var a1,a2:arraytype;
```

其中 arraytype 为一个类型标识符,表示一个下标值可以是 1~8,数组元素类型为整型的一维数组;而 a1,a2 则是这种类型的数组变量。

也可以将其全并起来:

```
var a1,a2:array[1..8] of integer;
```

当在说明部分定义了一个数组变量之后,Delphi 编译程序为所定义的数组在内存空间开辟一串连续的存储单元。

例如,设程序中有如下说明:

```
type rowtype = array[1..8] of integer;
coltype = array['a'..'e'] of integer;
var a:rowtype;b:coltype;
```

这里系统为数组 a 开辟一串连续  $8 \times 4 = 32$  字节的存储单元,为数组 b 开辟  $5 \times 4 = 20$  字节的存储单元。

## 2. 一维数组的引用

当定义了一个数组,则数组中的各个元素就共用一个数组名(即该数组变量名),它们之间是通过下标不同以示区别的。对数组的操作归根到底就是对数组元素的操作。一维数组元素的引用格式为:

数组名[下标表达式]

说明:

(1) 下标表达式值的类型,必须与数组类型定义中下标类型完全一致,并且不允许超越所定义的下标下界和上界。

(2) 数组是一个整体,数组名是一个整体的标识,要对数组进行操作,必须对其元素操作。数组元素可以像同类型的普通变量那样使用。如:  $a[3] := 34$ ; 是对数组 a 中第三个下标元素赋以 34 的值。  $read(a[4])$ ; 是从键盘读入一个数到数组 a 第 4 个元素中去。

特殊地,如果两个数组类型一致,它们之间可以整个数组元素进行传送。例如:

```
var a,b,c:array[1..100] of integer;
begin
  c := a; a := b; b := c;
end.
```

在上述程序中,a,b,c 三个数组类型完全一致,它们之间可以实现整数组传送。例子中先将 a 数组所有元素的值依次传送给数组 c,同样 b 数组传给 a,数组 c 又传送给 b,上述程序段实际上实现了 a,b 两个数组所有元素的交换。

对于一维数组的输入与输出,都只能对其中的元素逐个地输入与输出。

## 3. 多维数组的定义

当一维数组元素的类型也是一维数组时,便构成了二维数组。二维数组定义的一般格式:

```
array[下标类型 1] of array[下标类型 2] of 元素类型;
```



但一般这样定义二维数组：

```
array[下标类型 1, 下标类型 2] of 元素类型;
```

说明：其中两个下标类型与一维数组定义一样，可以看成“下界 1..上界 1”和“下界 2..上界 2”，给出二维数组中每个元素（双下标变量）可以使用下标值的范围。of 后面的元素类型就是基类型。

一般地，n 维数组的格式为：

```
array[下标类型 1, 下标类型 2, ..., 下标类型 n] of 元素类型;
```

其中，下标类型的个数即数组的维数，且说明了每个下标的类型及取值范围。

#### 4. 多维数组元素的引用

多维数组的数组元素引用与一维数组元素引用类似，区别在于多维数组元素的引用必须给出多个下标。

引用的格式为：

```
<数组名>[下标 1, 下标 2, ..., 下标 n]
```

说明：显然，每个下标表达式的类型应与对应的下标类型一致，且取值不超出下标类型所指定的范围。

例如，设有说明：

```
type matrix = array[1..5, 1..4] of integer;
var a: matrix;
```

则表示 a 是二维数组，共有  $5 \times 4 = 20$  个元素，它们是：

```
a[1,1]a[1,2]a[1,3]a[1,4]
a[2,1]a[2,2]a[2,3]a[2,4]
a[3,1]a[3,2]a[3,3]a[3,4]
a[4,1]a[4,2]a[4,3]a[4,4]
a[5,1]a[5,2]a[5,3]a[5,4]
```

因此可以看成是一个矩阵，a[4,2]即表示第 4 行、第 2 列的元素。由于计算机的存储器是一维的，要把二维数组的元素存放到存储器中，Delphi 是按行（第一个下标）的次序存放，即按 a[1,1]a[1,2]a[1,3]a[1,4]a[2,1]...a[5,4]的次序存放于存储器中某一组连续的存储单元之内。

对于整个二维数组的元素引用时，大多采用二重循环来实现。

#### 5. Delphi 还支持动态数组

动态数组是指在定义时并没有确定数组的大小或长度，而是在访问之前用 SetLength 过程为数组动态或重新分配其存储空间。

定义动态数组的语法形式如下：

```
Var 数组类型标识符: array of baseType;
```

例如：var DynArr: array of Integer; 声明 DynArr 为整型的动态数组。

也可以先定义类型，再声明变量，如：

```
type TdynIntArr = array of integer;
var DynArr: TdynIntArr;
```

语句 `SetLength(DynArr,10)`; 为动态数组 `DynArr` 分配 10 个元素的存储空间,下标为 0~9(动态数组的下标总是从 0 开始)。如要释放动态数组占用的存储空间,可以将 `nil` 赋值给该动态数组变量,或调用过程 `SetLength(DynArr,0)` 实现。

多维动态数组通过使用嵌套的“array of ...”结构实现,例如:

```
type TGrid = array of array of String;
```

### 2.5.3 字符串类型

字符串是由字符组成的有穷序列。

字符串类型的定义:

```
type <字符串类型标识符> = string[n];
```

```
var
```

```
字符串变量: 字符串类型标识符;
```

其中 `n` 是定义的字符串长度,必须是 0~255 之间的自然整数,第 0 号单元中存放串的实际长度,程序运行时由系统自动提供,第 1~`n` 号单元中存放串的字符。若将 `string[n]` 写成 `string`,则默认 `n` 的值为 255。例如:

```
type
```

```
man = string[8];
```

```
line = string;
```

```
var
```

```
name: man;
```

```
screenline: line;
```

另一种字符类型的定义方式为把类型说明的变量定义合并在一起。例如:

```
VAR
```

```
name: STRING[8];
```

```
screenline: STRING;
```

Delphi 字符串类型还有 `AnsiString`,也称作 `LongSting`,是多数情况下推荐使用的类型。各字符串类型可以混合在赋值和表达式中,编译器将自动执行转换。`String` 型是字符串的通用类型。在默认的 `{ $ H + }` 状态,编译器将 `String` 解释为 `AnsiString`; 用 `{ $ H - }` 编译指令可以指示编译器将 `String` 解释为 `ShortString`。

在不指定串的最大长度的情况下,`String` 被视为 `AnsiString` 型,此时字符串以 `Null` 为串的结束标志。在串中不保存串的长度;`ShortString` 不能指定最大的字符串长度,其最大字符长度固定为 255。

对字符串访问就像对数组的访问一样,如 `str` 是字符串变量,`i` 是整型表达式,则 `str[i]` 表示该字符串的第 `i` 个字符。例如:

```
var
```

```
name: string;
```

```
begin
```

```
readln(nsme);
```

```

    for i := 1 to ord(name[0])do
        writeln(name[i]);
    end.

```

语句 `writeln(name[i])` 输出 `name` 串中第 `i` 个字符。

由字符串的常量、变量和运算符组成的表达式称为字符串表达式。

字符串运算符包括：

### 1. + (连接运算符)

例如, 'Delphi' + '2007' 的结果是 'Delphi 2007'。

若连接后的字符串存放在定义的字符串变量中, 当其长度超过定义的字符串长度时, 超过部分字符串被截断。例如：

```

var
    str1, str2, str3: string[8];
begin
    str1 := 'Delphi ';
    str2 := '2007';
    str3 := str1 + str2;
end.

```

则 `str3` 的值为 'Delphi 2'。

### 2. =、<>、<、<=、>、>= (关系运算符)

两个字符串的比较规则为：从左到右按照 ASCII 码值逐个比较, 遇到 ASCII 码不等时, 规定 ASCII 码值大的字符所在的字符串为大。例如：

```

'AB' < 'AC' 结果为真;
'12' < '2' 结果为真;
'Delphi' = 'Delphi ' 结果为假。

```

### 3. Delphi 系统提供了很多常用的标准函数

- (1) 合并字符串函数: `Concat(s1, s2[, s3, s4, ...])`。
- (2) 取子串函数: `Copy(字符串, 开始位置, 长度)`。
- (3) 取左子串: `LeftStr(字符串, 长度)`。
- (4) 取右子串: `RightStr(字符串, 长度)`。
- (5) 判断一个子串在一个字符串中的起始位置: `Pos(字符串 A, 字符串 B)`。如 `Pos('abc', 'axabcdef')` 的值为 3。
- (6) 去除字符串的左空格: `TrimLeft(字符串)`。
- (7) 去除字符串的右空格: `TrimRight(字符串)`。
- (8) 去除字符串的左、右空格: `Trim(字符串)`。
- (9) 求字符串的长度: `Length(字符串)`。
- (10) 字符串转为大写的函数: `UpperCase(字符串)`。

## 2.5.4 记录类型

记录类型描述一组不同类型的数据元素集合, 每个数据元素称为“域”。在定义一个记录类型时, 需要指定每个数据域的数据类型。在一个记录中, 可以包含不同类型并且互相相

关的一些数据。

### 1. 记录类型的定义

在 Delphi 中,记录由一组称为“域”的分量组成,每个域可以具有不同的类型。

定义记录类型的语法如下:

```
type
  record
    <域名 1>:<类型 1>;
    <域名 2>:<类型 2>;
    : :
    <域名 n>:<类型 n>;
  end;
```

说明:

(1) 域名也称域变量标识符,应符合标识符的语法规则。在同一个记录类型中,各个域不能取相同的名;但在不同的记录类型中,两个类型中的域名可以相同。

(2) 记录类型的定义和记录变量可以合并为一个定义,如:

```
type
  date = record
    year:1900..1999;
    month:1..12;
    day:1..31
  end;
var x:date;
```

可以合并成:

```
var x: record
  year:1900..1999;
  month:1..12;
  day:1..31
end;
```

(3) 对记录的操作,除了可以进行整体赋值外,还能对记录的分量——域变量进行。

(4) 域变量的表示方法如下:

记录变量名.域名

如前面定义的记录 x,其三个分量分别为: x.year、x.month 和 x.day。

(5) 域变量的使用和一般的变量一样,即域变量是属于什么数据类型,便可以进行那种数据类型所允许的操作。

### 2. 开域语句

在程序中对记录进行处理时,经常要引用同一记录中不同的域,每次都按“记录变量名.域名”格式引用非常乏味。为此 Delphi 提供了一个 with 语句,可以提供引用域的简单形式。

开域语句的一般形式:

```
with <记录变量名表> do
  <语句>
```

功能：在 do 后的语句中使用 with 后记录的域时，只要直接写出域名即可省略记录变量名和“.”。

例如：

```
write('Input year:'); readln(x.year);
write('Input month:'); readln(x.month);
write('Input day:'); readln(x.day);
```

可以改写成：

```
with x do
begin
  write('Input year:'); readln(year);
  write('Input month:'); readln(month);
  write('Input day:'); readln(day);
end;
```

### 3. 变体记录

记录中的域可分为固定部分和可变部分。固定部分定义一个或者多个独立的域，可以为每个独立的域指定一个标识符和数据类型。

含有可变部分的记录类型称为变体记录，它提供了不同域共享内存的方法，其中每个命名的变体是共享内存的，每个变体在内存中占用同一空间，采用一个常量加以区别。

一般格式：

```
Type
  <类型标识符> = record
    <域名表 1> : <类型 1>;
    <域名表 2> : <类型 2>;
    ...
    <域名表 n> : <类型 n>;
    case [<标志域>:] <类型> of
      <常量表 1>: ([<域表 1>]);
      <常量表 2>: ([<域表 2>]);
      ...
      <常量表 n>: ([<域表 n>]);
    end;
```

说明：

- (1) 变体的标志域为顺序类型，一般用枚举类型(Delphi 中允许非顺序类型)。
- (2) 变体标志域和变体域都是记录中可访问的域。
- (3) case 中的<常量表>不可使用形如<常量 1>..<<常量 2>:的子界形式。
- (4) 可以省略变体的标志域，只写类型。但会带来判读困难。
- (5) 变体的域表可以是空，但必须保留空括号。如<常量表>: ( );。
- (6) 在定义记录变体时，case [<标志域>:] <类型> of 可以写成如下形式：

① <标志域>: <类型>

② case <类型> of

## 2.5.5 指针类型

前面介绍的各种简单类型的数据和构造类型的数据属于静态数据。在程序中,这些类型的变量一经说明,就在内存中占有固定的存储单元,直到该程序结束。

程序设计中,使用静态数据结构可以解决不少实际问题,但也有不便之处。如建立一个大小未定的姓名表,随时要在姓名表中插入或删除一个或几个数据。而用新的数据类型——指针类型,通过指针变量可以在程序的执行过程中动态地建立变量,它的个数不再受限制,可方便高效地增加或删除若干数据。

### 1. 指针类型和指针变量

在 Delphi 中,指针变量(也称动态变量)存放某个存储单元的地址。也就是说,指针变量指示某个存储单元。

指针类型的格式为:

^基类型

说明:

(1) 一个指针只能指示某一种类型数据的存储单元,这种数据类型就是指针的基类型。基类型可以是除指针、文件外的所有类型。例如下列说明:

```
type pointer = ^ Integer;  
var p1,p2:pointer;
```

定义了两个指针变量 p1 和 p2,这两个指针可以指示一个整型存储单元(即 p1、p2 中存放的是某存储单元的地址,而该存储单元恰好能存放一个整型数据)。

(2) 和其他类型变量一样,也可以在 var 区直接定义指针型变量。例如:

```
var a:^real; b:^boolean;
```

又如:

```
type person = record  
    name:string[20];  
    sex:(male,female);  
    age:1..100  
end;  
var pts:^person;
```

(3) Delphi 规定所有类型都必须先定义后使用,但只有在定义指针类型时可以例外,如下列定义是合法的:

```
type pointer = ^ rec;  
    rec = record  
        a:integer;  
        b:char  
    end;
```

### 2. 指针类型使用

在 Delphi 中,指针变量的值一般是通过系统分配的,开辟一个动态存储单元必须调用

标准过程 new。释放动态存储单元使用 dispose 标准过程。

#### 1) new 过程

new 过程调用的一般格式：

**New**(指针变量)

功能：开辟一个存储单元，此单元能存放的数据类型正好是指针的基类型，并把此存储单元的地址赋给指针变量。

说明：

(1) 这实际上是给指针变量赋初值的基本方法。

例如，设有说明：var p: ^Integer;

这只定义了 P 是一个指示整型存储单元的指针变量，但这个单元尚未开辟，或者说 P 中尚未有值(某存储单元的首地址)。当程序中执行了语句 new(p)才给 p 赋值，即在内存中开辟(分配)一个整型变量存储单元，并把此单元的地址放在变量 p 中。

(2) 一个指针变量只能存放一个地址。如再一次执行 New(p)语句，将在内存中开辟另外一个新的整型变量存储单元，并把此新单元的地址放在 p 中，从而丢失了原存储单元的地址。

(3) 当不再使用 p 当前所指的存储单元时，可以通过标准过程 Dispose 释放该存储单元。

#### 2) dispose 过程

释放动态存储单元使用 dispose 语句，其一般格式为：

**dispose**(指针变量)

功能：释放指针所指向的存储单元，使指针变量的值无定义。

### 3. 动态存储单元的引用

在给一个指针变量赋以某存储单元的地址后，就可以使用这个存储单元。

引用动态存储单元一般格式：

<指针变量> ^

说明：

(1) 在用 new 过程给指针变量开辟了一个它所指向的存储单元后，要使用此存储单元的唯一方法是利用该指针。

(2) 对动态存储单元所能进行的操作是该类型(指针的基类型)所允许的全部操作。

例如：

```
var
  p: ^integer; i: integer;
begin
  New(p);
  P ^ := 4;
  i := p ^;
  Dispose(p);
End
```

## 本章小结

在这一章中主要要求学生了解的要点有 Delphi 2007 控制台程序结构、Delphi 2007 的基本字符、常量与变量的定义与使用、各种复杂数据类型的概念与使用。重点是 Delphi 2007 的控制台程序的基本语法知识。难点是输入输出语句的格式控制以及复杂数据类型(如数组、字符串、记录与指针)的理解与使用。

为了做好程序设计,必须首先分析所给问题,明确要求。标识输入量与输出量,确定它们的数据类型。然后再确定从所给输入到输出需执行的步骤,即进行算法设计。在编写程序时应正确使用 Delphi 2007 语句,并注意标点符号的正确使用,不要漏写或写错。在程序中最好每行包含一个语句,并注意把各个语句按层次对齐,在必要的地方添加注释,便于提高程序的可读性。

## 思考与练习

1. Delphi 语言中有哪些常用的数据类型? 变量在使用前必须先定义,如何定义各种数据类型的变量?
2. 简述 Delphi 中标识符的命名应遵循哪些规则?
3. Delphi 中提供了哪些类型运算符,在表达式中其优先级从高到低如何排列?
4. 注释语句有哪几种形式?
5. 编写程序,输入两个浮点数,输出其和、差、积各为多少。
6. 编写程序,输入 5 个整数,输出这 5 个数的平均值。



现实生活中的流程是多种多样的,如汽车在道路上行驶,要顺序地沿道路前进,碰到交叉路口时,驾驶员就需要判断是转弯还是直走,在环路上是继续前进,还是需要从一个出口出去等。又比如,生产线上零件的流动过程,应该顺序地从一个工序流向下一个工序,但当检测不合格时,就需要从这道工序中退出,或继续在这道工序中再加工直到检测通过为止。因此,现实生活中的流程可以概括为以下几个方面:

(1) 顺序流程:一个流程(语句或者复合语句)执行完后,无条件地执行下一个流程(语句或者复合语句)。

(2) 流程分支:一个流程(语句或者复合语句)到达一定点时,需要分为两个或多个分支继续进行。

(3) 流程循环:或者可以说是重复不停地进行同一工作(语句或者复合语句)。

从程序流程的角度来看,程序可以分为三种基本结构,即顺序结构、分支结构和循环结构。这三种基本结构可以组成所有的各种复杂程序。Delphi 语言提供了多种语句来实现这些程序结构。

语句是构造程序的最基本单位。当用程序语句编写的程序越来越大、越来越复杂的时候,为了使程序更简洁、可读性更好、更便于复用,以及更便于维护,就有必要将它分成若干个模块,每个模块完成一项任务。在 Delphi 语言中,这些模块就是一个一个的函数与过程。函数与过程也是 Delphi 语言构造程序的重要的基本单位。

本章主要包括以下内容:

- 顺序结构程序设计;
- 选择结构程序设计;
- 循环结构程序设计;
- 运算符与表达式;
- 函数与过程。

### 3.1 顺序结构程序设计

顺序语句是任何程序的基本语句。程序中各条语句按照程序书写的顺序依次执行,语句体比较简单,一般都是一行语句作为一个语句体,通常由赋值语句等简单的操作语句组成。

在 Delphi 中,一条语句可以写在一行,也可以写在多行,但在一条语句的末尾必须加上分号“;”,用来表示一条语句的结束。

### 3.1.1 Delphi 程序基本架构

任何一个 Delphi 程序基本都包含过程和函数,过程或者函数一般又由头部与主体构成。主体部分一般由声明部分与实现部分组成。声明部分主要有标号声明、符号常量声明、类型定义声明和变量声明,实现部分有一对 Begin...End。

#### 1. 标号声明

标号一般是和一些控制转移语句一起使用,它可以是一个 0~9999 的整数,也可以是一个标识符。

标号声明的格式如下:

```
label label1[,label2[,label3[, ...]]];
```

#### 2. 符号常量声明

常量声明的格式如下:

```
const 常量名 = 表达式;
```

#### 3. 类型定义声明

类型声明的格式如下:

```
type 类型名 = 类型定义;
```

例如:

```
type DefType = Array[1...200] of real;           //声明一个具有 200 个元素的实数数组
```

#### 4. 变量声明

变量声明的格式如下:

```
var 变量名表: 类型;
```

例如:

```
var x,y: Integer;
```

#### 5. 基本语句

Delphi 简单语句中不包含任何别的语句。简单语句用分号隔开,包含过程、函数调用语句等,其中赋值语句是形式简单,使用最频繁的语句,它的功能是为变量赋值。

赋值语句的一般格式为:

```
<变量> := <表达式>;
```

其中符号“:=”是赋值运算符,它表示将运算符右侧表达式运算的结果存入左侧变量相对应的存储单元中,作为左侧变量当前的值。

#### 6. 复合语句

Delphi 用 begin 和 end 将简单语句括起来即组成复合语句,复合语句用法与普通的 Delphi 语句相同,见下例:

```
begin
  A := B;
```

```

    C := A * 2;
end;

```

end 之前的最后一条语句末尾分号不是必需的,可以写成:

```

begin
    A := B;
    C := A * 2
end;

```

这两种写法都是正确的。第一种多了一个无用(但也无害)的分号。分号实际上是一个空语句,也就是说,是一个没有代码的语句。有时,空语句可用在循环体或其他特殊情况中。

**注意:**虽然最后一条语句末尾的分号没有用,笔者却总是加上它,并且建议读者也这样做。因为有时可能需要在末尾添加语句,如果最后没有加分号,就必须记得加上它,与其如此不如一开始就加上它。

### 3.1.2 顺序程序举例

到目前为止,可以用读、写语句和赋值语句编写一些简单的程序。通过阅读这些程序,可以逐步熟悉 Delphi 程序的编写方法和应遵循的规则。

在顺序结构程序中,各语句(或命令)是按照位置的先后次序顺序执行的,且每个语句都会被执行到。

顺序结构 N-S 图如图 3-1 所示,执行顺序为先 A 后 B。

**例 3.1** 输入圆的半径,输出圆的周长和面积。

分析:

- (1) 定义实型变量 r、c、s 用于存放半径、周长、面积;
- (2) 调用输入语句,输入 r;
- (3) 分别利用周长公式和面积公式求出 c、s;
- (4) 调用输出语句输出 c、s。

程序:

```

program Project1;                                //程序头部
{$APPTYPE CONSOLE}                               //编译预处理指令,表示控制台程序
const
    Pi = 3.14159;                                //定义常量
var
    r, c, s: Real;                                //声明变量
begin
    writeln('请输入半径: ');                      //提示用户输入
    readln(r);                                     //输入一个浮点数据,存放到 r
    c := 2 * Pi * r;                               //计算周长,赋值给 c
    s := Pi * r * r;                               //计算面积,赋值给 s
    writeln('周长 c = ', c:8:2);                   //周长,数据共占 8 个字符宽,2 位小数
    writeln('面积 s = ', s:8:2);                   //面积,数据共占 8 个字符宽,2 位小数
    readln;                                        //暂停一下,让用户看运行结果
end.

```

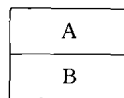


图 3-1 顺序结构 N-S 图

程序运行结果如图 3-2 所示。

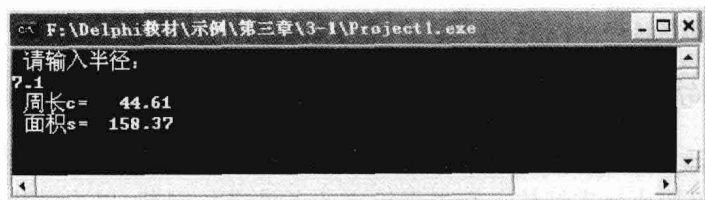


图 3-2 运行结果

### 例 3.2 交换两个数据并输出。

分析：两个数据相互交换是程序设计的一项基本方法。假定有两个变量 a、b 保存有数据，要交换它们的数据。

方法：使用一个额外的变量 c 作为交换数据的中转站。

(1)  $c := a$ ，将 a 中原始值放入 c 中保存起来；

(2)  $a := b$ ，将 b 的值赋给 a；

(3)  $b := c$ ，再将 c 中保存的 a 的原始值赋给 b，这样就实现了 a 和 b 中数据的互换。

参考程序如下：

```
program Project1;
{$APPTYPE CONSOLE}                                     //编译预处理指令,表示控制台程序
var
    a,b,c: integer;
begin
    writeln('请输入要交换的两个数: ');                  //提示用户输入
    readln(a,b);                                         //输入数据到两个变量
    writeln('交换前两个数为:a= ',a,' ,b= ',b);          //交换前
    //交换过程,注意次序
    c := a;                                              //第一步,将 a 的原始值赋给 c
    a := b;                                              //第二步,将 b 的值赋给 a
    b := c;                                              //第三步,将 c 中存放的 a 的原值赋给 b
    writeln('交换后两个数为:a= ',a,' , b= ',b);         //交换后
    readln;
end.
```

运行结果如图 3-3 所示。

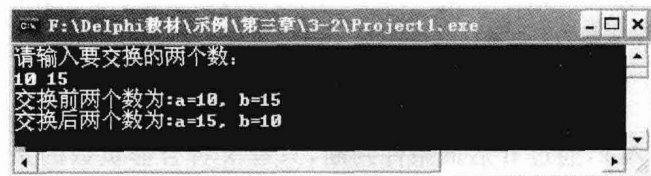


图 3-3 运行结果

## 3.2 选择结构程序设计

选择结构是根据运行时的情况自动选择要执行的语句。在 Delphi 语言中,用 if 语句或 case 语句实现选择结构。if 语句称作条件语句,case 语句称作分情况语句。如何表示条件,

如何使用选择结构控制程序流程,是学习本节要掌握的主要内容。

3.2.1 if 语句

1. if 语句的三种形式

用 if 语句可以构成分支结构。它根据给定的条件进行判断,以决定执行某个分支程序段。Delphi 语言的 if 语句有三种基本形式。

if 语句中的“表达式”一般为逻辑表达式或关系表达式,表达式的值必须为布尔值 True 或者 False。

1) 形式一:

IF <布尔表达式> THEN 语句;

形式一是不带 else 的 if 语句,是 if 语句的基本形式,其执行过程为:如果布尔表达式的值为 True,则执行其后的语句,否则不执行该语句。语句可以为简单语句,也可以是复合语句(Begin...End)。执行过程的流程图如图 3-4 所示。

例 3.3 输入一个数,如果该数大于等于 0,则输出它的平方根;当它小于 0,则不做任何处理。

分析:

- (1) 输入的数存入变量 x;
- (2) 使用选择 if(x>=0)。

程序:

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
    x:Double;
begin
    writeln('请输入一个浮点数: ');                    //提示用户输入
    readln(x);                                         //输入数据到 x 变量
    //sqrt(x)功能为开平方的库函数,输出占两位小数
    if x >= 0 then                                     //x 大于等于 0,才输出
        writeln('平方根为:',sqrt(x):8:2);
    readln;
end.
```

本例程序中,输入 x,通过 if 后面条件判断,只有 x 符合非负数的条件才输出其开平方的值,否则没有任何动作。

运行结果如图 3-5 所示。

2) 形式二:

IF <布尔表达式> THEN 语句 1 ELSE 语句 2;

注意: IF 语句中语句 1 后无“;”号,每一个语句可以为简单语句,也可以是复合语句(Begin...End)。

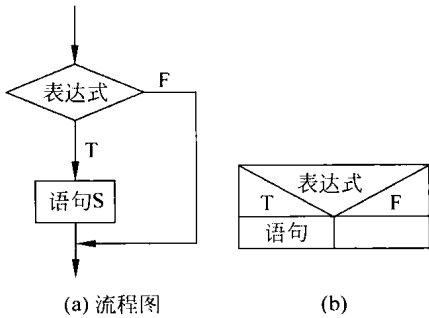


图 3-4 if 语句(形式一)执行流程图

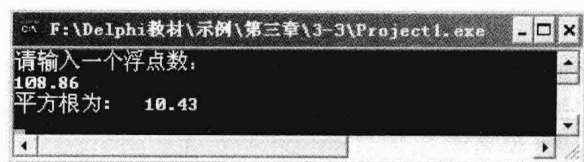


图 3-5 运行结果

形式二称为 if-else 形式,其执行过程为:如果表达式的值为真,执行语句 1,否则执行语句 2。执行过程的流程图如图 3-6 所示。它通常用在表示条件满足或不满足分别执行两组不同操作的情况中。

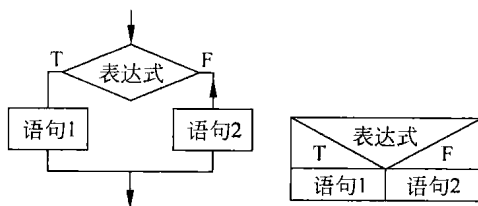


图 3-6 if 语句(形式二)执行流程图

显然,形式二的 if 结构语句相当于两条形式一的 if 结构语句,它也可表示为:

```
If 布尔表达式 then 语句 1;
If not (布尔表达式) then 语句 2;
```

**例 3.4** 输入一个整数 a,判断是否为偶数(是输出 yes,否则输出 no)。

参考程序为:

```
program Project1;
{ $ APPTYPE CONSOLE}                                     //编译预处理指令,表示控制台程序
var
    x: Integer;
begin
    write('请输入 x = ');                                   //提示用户输入
    readln(x);                                              //输入数据到 x 变量
    if (x mod 2 = 0) then                                    //条件可以用 not odd(x)代替
        writeln('yes')                                     //x 对 2 求余等于 0,表明是偶数
    Else
        writeln('no');                                     //条件为假,表明是奇数
    readln;
end.
```

运行结果如图 3-7 所示。

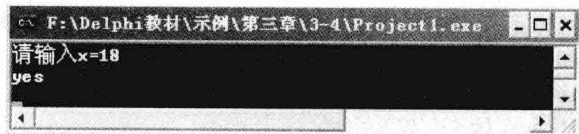


图 3-7 运行结果

3) 形式三:

```
if(表达式 1) then 语句 1
else if(表达式 2) then 语句 2
else if(表达式 3) then 语句 3
:
else if(表达式 m) then 语句 m
else 语句 n;
```

形式三称为 if-else-if 形式,其执行过程为:依次求 if 后表达式的值,如果某个表达式的值为真,则执行其后的语句,并跳过其后的其他语句;如果没有一个表达式的值为真,则执行最后一个 else 后的语句 n。执行过程的流程图如图 3-8 所示。

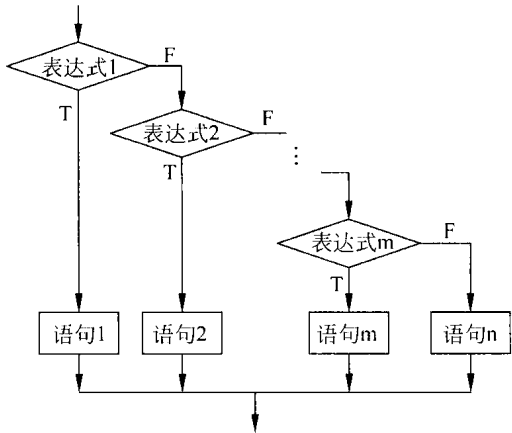


图 3-8 if 语句(形式三)流程图

**注意:** 前面每一个 else 后的语句末尾不能有分号,只有最后一个才有分号。每一个语句可以为简单语句,也可以是复合语句(Begin...End)。

**例 3.5** 用键盘输入字符输出其类别,要求根据 ASCII 码来判断该字符属于控制字符、数字、大写字母、小写字母还是其他字符。

程序分析:由 ASCII 码表可知 ASCII 值小于 32 的为控制字符,在"0~9"之间的为数字,在"A~Z"之间的为大写字母,在"a~z"之间的为小写字母,其余则为其他字符。

使用 if-else 选择结构进行多分支,程序设计如下:

```
program Project1;
{ $ APPTYPE CONSOLE}                                     //编译预处理指令,表示控制台程序
var
    c:Char;                                                //定义字符变量
begin
    write('请输入一个字符');                               //提示用户输入
    readln(c);                                             //输入数据到字符变量
    if (c<#32) then
        writeln('这是一个控制字符!')
    Else if (c>= '0') and (c<= '9') then
        writeln('这是一个数字字符!')
    Else if (c>= 'A') and (c<= 'Z') then
```

```

        writeln('这是一个大写字母!')
    Else if (c>= 'a') and (c<= 'z') then
        writeln('这是一个小写字母!')
    Else
        writeln('这是一个其他字符!') ;
    readln;
end.

```

这是一个多分支选择的问题,用 if-else-if 语句编程,判断输入字符 ASCII 码所在的范围,分别给出不同的输出。例如输入为“t”,输出显示它为小写字母。

运行结果如图 3-9 所示。

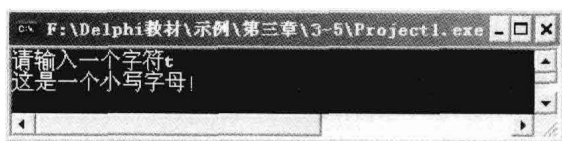


图 3-9 运行结果

## 2. if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套。一般形式如下:

```

if(布尔表达式 1) then
    if(布尔表达式 2) then 语句 1
    else 语句 2
else
    if(布尔表达式 3) then 语句 3
    else 语句 4;

```

} 内嵌 if

} 内嵌 if

嵌套内的 if 语句可能又是 if-else 型的,这将会出现多个 if 和多个 else 重叠的情况,这时要特别注意 if 和 else 的配对问题。

在 if 语句的嵌套结构中并不需要对称,只根据需要决定嵌套的形式。在写 if 语句的嵌套结构时,要注意 else 与 if 配对的规则,else 与同一层最接近它而又没有其他 else 语句与之相匹配的 if 语句配对。如果忽略了 else 与 if 配对,就会发生逻辑错误。如果读者采用较好的编程习惯,即缩进形式,那么 if 与 else 的配对也许就更明了。

**例 3.6** 有一个函数如下:编写程序,实现任意输入一个 x 的值,输出对应的 y 值。

$$y = \begin{cases} 1 & (x > 0) \\ 0 & (x = 0) \\ -1 & (x < 0) \end{cases}$$

分析:

输入 x 后,对 x 进行判断,显然 x 的值有三种可能:

- (1) 如果  $x > 0$ ,则  $y := 1$ ;
- (2) 如果  $x = 0$ ,则  $y := 0$ ;
- (3) 如果  $x < 0$ ,则  $y := -1$ 。

图 3-10 给出了这个问题的算法描述。

在这个算法描述的 N-S 图中有两个判断,第二个判断

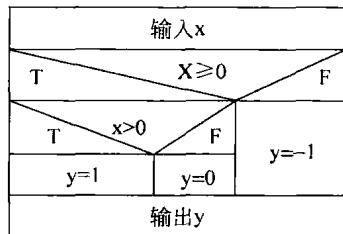


图 3-10 算法描述(N-S 图)



是嵌套在第一个判断之中的,可以用 if 语句的二重嵌套来实现。程序如下:

```

program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
    x:Real;                                           //定义输入变量
    y:integer;                                        //定义分段函数输出变量 y
begin
    write('请输入一个数');                            //提示用户输入
    readln(x);                                        //输入数据到变量 x
    if(x >= 0) then
        if(x > 0) then                                //内嵌 if 语句
            y := 1;
        else
            y := 0;
    else
        y := -1;
    writeln('y = ', y);
    readln;
end.

```

运行结果如图 3-11 所示。

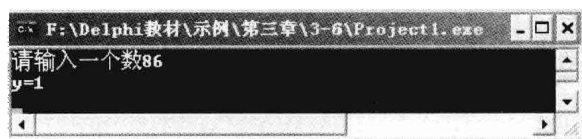


图 3-11 运行结果

### 3.2.2 case 语句

多分支结构可以使用嵌套的 if 语句处理,但如果分支较多,嵌套的 if 语句层数也增多,程序冗长,降低了可读性。因此,Delphi 语言提供了 case 语句,也叫开关语句,它是多分支选择语句,每个分支、每种情况可通过一个常量表达式取不同的值来描述。

#### 1. case 语句的一般形式

```

case <条件表达式> of
    <条件标号表 1>: 语句 1;
    <条件标号表 2>: 语句 2;
    :
    <条件标号表 n>: 语句 n;
    [Else      语句 n+1;]           //可以默认
end;

```

其中 case、of、end 是 Delphi 的保留字,表达式的值必须是顺序类型,它可以是整型、布尔型及字符型、枚举型和子界型。条件标号表是一串用逗号隔开的与表达式类型一致的常量序列。语句可以是任何语句,包括复合语句和空语句。

#### 2. case 语句的执行过程

先计算表达式(称为条件表达式)的值,如果它的值等于某一个常量(称为条件常量,也称

条件标号),则执行该情况常量后面的语句,在执行完语句后,跳到 case 语句的末尾 end 处。

### 3. 说明

- (1) 条件表达式必须是顺序类型的;
- (2) 条件常量是条件表达式可能具有的值,因而应与条件表达式具有相同的类型;
- (3) 条件常量出现的次序可以是任意的;
- (4) 同一条件常量不能在同一个 case 语句中出现两次或两次以上;
- (5) 每个分语句前可以有一个或若干个用逗号隔开的条件常量;
- (6) 如果条件表达式的值不落在条件常量的范围内,则认为本 case 语句无效,执行 case 语句的下一个语句。Delphi 允许用户增加了一个“否则”的情况,即增加一个 else 子句,但也是可省略的。
- (7) 每个常量后面只能是一个语句或一个复合语句。

**例 3.7** 用条件语句编写程序,判断学生考试成绩属于哪个档次(优、良、中、及格、不及格)。

程序分析: 根据输入的百分制成绩来判定学生成绩的等级,其判定标准为: 90 分以上为优; 80~89 分为良好; 70~79 分为中; 60~69 分为及格; 60 分以下为不及格,分别用 A、B、C、D、E 表示。

如果用 score 代表学生成绩,只需分别给出属于优、良、中、及格、不及格的成绩范围,判断 score 在哪一个成绩段内,然后再输出该成绩段属于哪个档次即可。

用 case 语句编写的程序如下:

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
    score:Real;                                       //定义输入变量
    grade:integer;                                    //定义 case 条件变量
begin
    write('请输入一个分数');                          //提示用户输入
    readln(score);                                    //输入数据到变量
    grade := round(score) div 10;                      //Round 取整,确定成绩属于哪个档次
    case grade of
        10,9: writeln('分数属于优: A!');
        8 : writeln('分数属于良: B!');
        7 : writeln('分数属于中: C!');
        6 : writeln('分数属于及格: D!');
        5,4,3,2,1,0: writeln('分数属于不及格: E!');
        else writeln('分数输入非法!');
    end;
    readln;
end.
```

运行结果如图 3-12 所示。

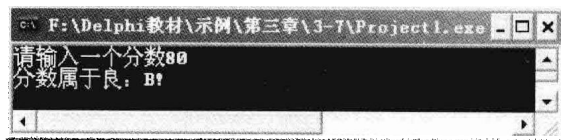


图 3-12 运行结果

### 3.3 循环结构设计

许多问题的求解归结为重复执行的操作,比如数值计算中的方程迭代求根,非数值计算中的对象遍历。重复执行就是循环,这是计算机特别擅长的工作之一。

循环结构是程序中一种很重要的结构。其特点是在给定条件成立时,反复执行某程序段,直到条件不成立为止。给定的条件称为循环条件,反复执行的程序段称为循环体。Delphi 语言提供了多种循环语句,可以组成各种不同形式的循环结构。

Delphi 语言中主要提供了三种最基本的循环结构:

- (1) while 语句构成的循环结构(“条件当型循环”)。
- (2) Repeat...Until 语句构成的循环结构(“直到型循环”)。
- (3) for 语句构成的循环结构(“计数当型循环”)。

#### 3.3.1 while 语句

while 语句在 Delphi 语言中用得比较多,它是通过判断循环控制条件是否满足来决定是否继续循环,又称“当型”循环。

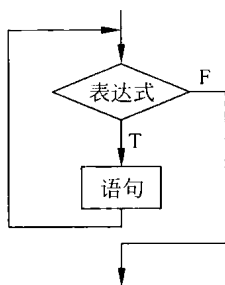


图 3-13 while 循环执行流程图

它的一般语句格式如下:

```
While <布尔表达式> do
    <循环体语句;>
```

执行过程:先计算布尔表达式的值,当条件为“真”时,执行循环体语句,直到条件为“假”时才跳出循环,继续执行循环体外的后续语句,如图 3-13 所示。

需要说明的是:

- (1) while 语句的特点是先计算布尔表达式的值,然后根据表达式的值决定是否执行循环体中的语句。因此,如果表达式的值一开始就为“假”,那么循环体一次也不执行。
- (2) 当循环体为多个语句组成,必须用 Begin...End 封闭起来构成复合语句。如果不加 Begin...End,则 while 语句的范围只到 while 后的第一个分号处。
- (3) 在循环体中应有使循环趋于结束的语句,以避免“死循环”的发生。
- (4) 单独一个分号也为一条有效语句,即空语句,表示什么也不执行。空语句作为循环体时,一般用作延时。

**例 3.8** 用 while 语句编程求  $1+2+\dots+100$ 。

分析:这是一个多个数求和的问题,可以用循环语句来解决。根据计算过程,可以画出其执行流程图,如图 3-14 所示。

参考程序如下:

```
program Project1;
{ $ APPTYPE CONSOLE }
var
    i:integer;
```

//编译预处理指令,表示控制台程序

//定义循环计数变量

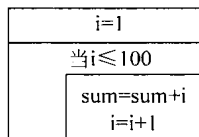


图 3-14 算法描述(N-S图)

```

sum:integer;                                //定义累计和变量
begin
  //初始化循环控制变量 i 和累计器 sum
  i:=1;
  sum:=0;
  while(i>=100) do                          //条件判断,控制循环
  begin
    sum:=sum+i;                             //实现累加
    i:=i+1;                                 //循环控制变量 i 自增 1
  end;
  writeln('sum=',sum);
  readln;
end.

```

程序运行结果如图 3-15 所示。



图 3-15 运行结果

在程序的循环体中,循环控制变量  $i$  每次加 1,使表达式  $i \geq 100$  的值趋于为假。

**例 3.9** 写一个程序,输入一个班学生的成绩,求全班的平均成绩。

分析:输入成绩、计算平均成绩都是一个重复性过程,因此可以用循环语句来实现。在这里,并不知道有多少个学生,也就是说不知道循环到底有多少次,但考虑到成绩没有负数,这样就可以把循环条件定为:每当输入的分数大于等于 0 时就继续输入成绩;输入的分数小于 0 时就停止输入。

程序步骤如下:

- (1) 输入一个分数。
- (2) 当“分数  $\geq 0$ ”时,做下列工作:
  - ① 累计总分;
  - ② 人数加一;
  - ③ 输入下一分数。
- (3) 重复第(2)步,直到“分数  $< 0$ ”。

参考程序:

```

program Project1;
{$APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
  i:integer;                                       //定义循环计数变量
  sum,average:Real;                               //定义累计和与平均值变量
  score:Real;
begin
  //初始化循环控制变量 i 和累计器 sum
  i:=0;
  sum:=0;

```

```

writeln('请输入所有学生分数,负分数表示输入结束'); //提示用户输入
readln(score); //输入第一个学生的分数
while(score >= 0) do //条件判断,控制循环
begin
    sum := sum + score; //实现累加
    i := i + 1; //循环控制变量 i 自增 1
    readln(score); //输入下一个学生的分数
end;
if i > 0 then
    average := sum/i //求平均成绩 average
else
    average := 0;
writeln('平均分 Average = ', average:8:2); //输出平均成绩,保留两位小数
readln;
end.

```

程序运行结果如图 3-16 所示。

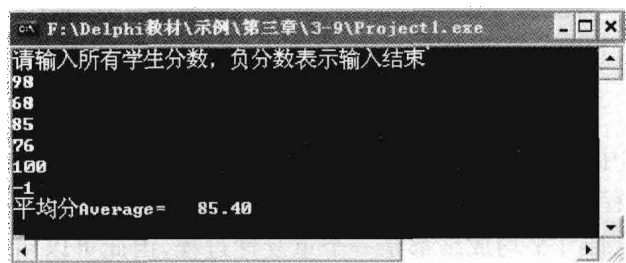


图 3-16 运行结果

在编制循环程序时,要注意下面几个方面:

(1) while 语句中的表达式一般是关系表达式或逻辑表达式,只要表达式的值为布尔值“真”(True)即可继续循环。

(2) 循环体如果包含一个以上的语句,应该用 Begin...End 括起来,以复合语句的形式出现。如果不加 Begin...End,则 while 语句的范围只到 while 后面的第一个分号处。比如上例中,while 语句中如无 Begin...End,则 while 语句范围只到 sum := sum + score;。

(3) 在循环体中应有使循环趋向于结束的语句。比如上例中,循环结束的条件是 score >= 0,那么当把这个班的成绩全部输入完后,一定要输入一个负数,才能使循环结束,然后程序继续执行。

(4) “当型循环”循环体语句有可能一次也不执行。如本例中,当第一次输入的分数就为负数时,则循环体一次也不执行。

### 3.3.2 直到循环

用 while 语句可以实现“当型循环”,用 repeat...until 语句可以实现“直到循环”。repeat...until 语句的含义是“重复执行循环,直到指定的条件为真时为止”。

直到循环语句的一般形式:

```
repeat
```

```

<语句 1>;
:
<语句 n>;
until <布尔表达式>;

```

其中 Repeat、until 是 Delphi 保留字,repeat 与 until 之间的所有语句称为循环体。

说明:

(1) repeat 语句的特点是先执行循环,后判断结束条件,因而至少要执行一次循环体。

(2) repeat...until 是一个整体,它是一个(构造型)语句,不要误认为 repeat 是一个语句,until 是另一个语句。

(3) repeat 语句在布尔表达式的值为真时不再执行循环体,且循环体可以是若干个语句,不需 begin 和 end 把它们包起来,repeat 和 until 已经起到了 begin 和 end 的作用。while 循环和 repeat 循环是可以相互转化的。

直到循环(repeat...until 语句)的具体执行过程为:先执行一次循环体语句,然后判别布尔表达式,若表达式的值为假,返回重新执行循环体语句,直到表达式的值为真时才结束循环,如图 3-17 所示。

**例 3.10** 用 repeat...until 语句编程求  $1+2+\dots+n$ ,  $n$  由用户输入。

本例基本与例 3.8 相似。流程如图 3-18 所示。

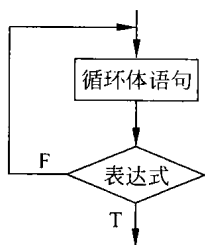


图 3-17 repeat...until 循环执行流程图

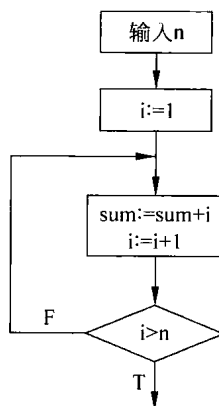


图 3-18  $1+2+\dots+n$  算法流程图描述

参考程序如下:

```

program Project1;
{ $APPTYPE CONSOLE}
var
    i:integer;
    n,sum:integer;
begin
    //初始化循环控制变量 i 和累计器 sum
    i:=1;
    sum:=0;
    write('请输入 n= ');
    readln(n);
    Repeat

```

//编译预处理指令,表示控制台程序

//定义循环计数变量

```

        sum := sum + i;           //实现累加
        i := i + 1;             //循环控制变量 i 自增 1
    Until i > n;                 //条件判断,控制循环
    writeln('sum = ', sum);
    readln;
end.

```

程序运行结果如图 3-19 所示。

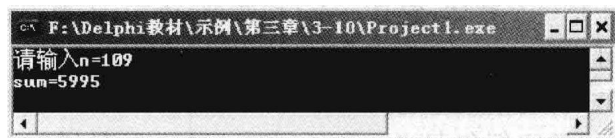


图 3-19 运行结果

可以看到,对循环问题可以用 while 语句处理,也可以用 repeat...until 语句处理,但不管用哪种语句处理,对循环控制变量赋初值时,一定不能将赋值语句放到循环体语句中,而要放到循环体语句之前,否则程序将是一个死循环。

### 3.3.3 for 语句

for 语句是计数的循环语句,可用于循环次数已经确定的情况。

#### 1. for 语句的一般格式

```

for <控制变量> := <表达式 1> to <表达式 2> do
    <语句>;
或者
for <控制变量> := <表达式 2> downto <表达式 1> do
    <语句>;

```

其中 for、to、downto 和 do 是 Delphi 保留字。表达式 1 与表达式 2 的值也称为初值和终值。

例如前面  $1+2+3+\dots+n$  的例子中主要语句用 for 可以写成如下:

```

for i := 1 to n do
    sum := sum + i;

```

或

```

for i := n downto 1 do
    sum := sum + i;

```

#### 2. for 语句执行过程

- (1) 先将初值赋给左边的变量(称为循环控制变量);
- (2) 判断循环控制变量的值是否已“超过”终值,如已超过,则跳到步骤(6);
- (3) 如果未超过终值,则执行 do 后面的那个语句(称为循环体);
- (4) 循环变量递增(对 to)或递减(对 downto)1;
- (5) 返回步骤(2);
- (6) 循环结束,执行 for 循环下面的一个语句。

### 3. 说明

(1) 循环控制变量必须是顺序类型。例如,可以是整型、字符型等,但不能为实型。

(2) 循环控制变量的值递增或递减的规律是:选用 to 则为递增;选用 downto 则为递减。

(3) 所谓循环控制变量的值“超过”终值,对递增型循环,“超过”指大于后面的表达式;对递减型循环,“超过”指大于前面的表达式。

(4) 循环体可以是一个基本语句,也可以是一个复合语句。

(5) 循环控制变量的初值和终值一经确定,循环次数就确定了。但是在循环体内对循环变量的值进行修改,常常会使得循环提前结束或进入死循环。建议不要在循环体中随意修改控制变量的值。

(6) for 语句中的初值、终值都可以是顺序类型的常量、变量、表达式。

**例 3.11** 求正整数  $n$  的阶乘  $n!$ ,其中  $n$  由用户输入。

分析:  $n! = 1 \times 2 \times \cdots \times n$ 。设置变量 fact 为累乘器(被乘数), $i$  为乘数,兼做循环控制变量。

程序如下:

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
    i: integer;                                       //定义循环计数变量
    n: integer;
    fact: integer;                                    //定义存放结果阶乘变量
begin
    fact := 1;                                       //初始化
    write('请输入 n = ');
    readln(n);
    for i := 1 to n do
        fact := fact * i;                            //实现累乘
    writeln(n:2, '!= ', fact);
    readln;
end.
```

程序运行结果如图 3-20 所示。



图 3-20 运行结果

#### 3.3.4 辅助控制语句

前面介绍的循环只能在循环条件不成立的情况下才能退出循环。可是有时人们希望从循环中直接退出来或重新下一次循环。要想实现这样的功能就要用到本节介绍的 break、continue 语句。

通常可以在前面介绍的三种循环体中调用 break 和 continue 语句。如果调用 break,会



使程序立刻跳出循环而执行循环后的那条语句；如果调用 continue, 会使循环体内 continue 后的代码本次不再执行而返回去再次判断循环条件, 以决定是否继续循环。

### 1. break 语句

break 语句的形式为:

```
break;
```

break 语句是限定转向语句, 它使流程跳出所在的结构, 把流程转向所在结构之后。break 语句在循环结构中的作用是跳出所在的循环结构, 转向执行该循环结构后面的语句。

说明:

(1) 通常 break 语句总是与 if 语句联在一起, 即满足某条件时便跳出循环。

(2) break 语句只能跳出它所在的那一层循环, 即在多层循环中, break 语句只向外跳一层, 而不能一下跳出最外层。

**例 3.12** break 语句应用示例: 打印输出 0~10 的数字。

参考程序如下:

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
    i: integer;                                       //定义循环计数变量
begin
    i := 1;
    while i >= 100 do                                //条件范围比较大
    begin
        write(i:5);                                  //按照固定宽度输出
        if i = 10 then break;                        //如果满足等于 10 要求,跳出循环
        i := i + 1;                                   //i 累加
    end;
    readln;
end.
```

这段程序将在屏幕上显示 0~10 的数字。虽然循环条件是  $i \leq 100$ , 但因为有了 break 语句, 导致程序在  $i$  等于 10 的时候从循环中立即退出, 循环也终止了。

程序运行结果如图 3-21 所示。



图 3-21 运行结果

### 2. continue 语句

continue 语句的形式为:

```
continue;
```

continue 语句被称为继续语句。该语句的功能是使本次循环提前结束, 即跳过循环体

中 continue 语句后面尚未执行的循环体语句,继续进行下一次循环的条件判别。

说明:

(1) continue 语句常与 if 语句一起使用,起到加速循环的作用。

(2) continue 语句与 break 语句的区别是: continue 语句只结束本次循环,而不是终止整个循环的执行;而 break 语句则是结束整个循环过程,不再判断循环条件是否成立。

**例 3.13** continue 语句应用示例:求输入的 10 个整数中正数的个数及其平均值。

参考程序如下:

```
program Project1;
{ $ APPTYPE CONSOLE }                                //编译预处理指令,表示控制台程序
var
    i,a:integer;                                       //定义变量
    num:integer = 0;sum:integer = 0;                 //定义数量与累计和,并初始化 0
    aver:real = 0;                                     //定义平均变量,并初始化 0
begin
    writeln('请输入 10 个数:');                       //输入提示
    for i := 1 to 10 do                               //i 为循环控制变量
    begin
        read(a);                                       //读入整数,将其值赋给变量 a
        if(a >= 0) then continue;                   //如果为负数则结束本次循环
        num := num + 1;                               //对输入的正数计数
        sum := sum + a;                               //求和
    end;
    readln;                                           //读一个回车,目的是为了暂停一下
    if num > 0 then
        aver := sum/num;
    writeln(num,'个正整数累计和:',sum);
    writeln(num,'个正整数平均值:',aver);
    readln;
end.
```

程序运行结果如图 3-22 所示。

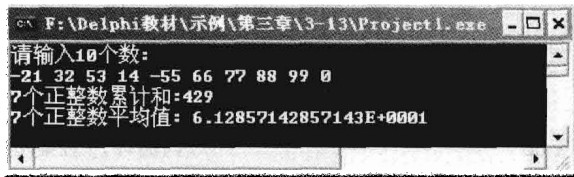


图 3-22 运行结果

### 3.3.5 循环嵌套

在循环体语句中又包含有另一个完整的循环结构的形式,称为循环的嵌套。嵌套在循环体内的循环体称为内循环,外面的循环体称为外循环。如果内循环体中又有嵌套的循环语句,则构成多重循环。while、repeat...until 和 for 三种循环都可以互相嵌套。

循环可以嵌套很多层,内嵌的循环中还可以嵌套循环,这就是多重循环。按循环层次数,分别称之为二重循环、三重循环等。处于内部的循环叫内循环,处于外部的循环叫外

循环。

嵌套的循环是这样执行的：因为内循环是外循环的循环体语句，所以外循环控制变量的值每变化一次，则内循环要执行一个“轮回”，即内循环控制变量的值从“初值”变化到“终值”，也就是说内循环执行到退出为止。下面以一个二重循环说明其执行过程。

```
For m := 1 to 2 do
  For n := 1 to 3 do
    S := m + n;
```

该程序段的执行过程如表 3-1 所示。

表 3-1 嵌套循环执行示例

外循环控制变量 m	内循环控制变量 n	语句 s := m+n
m := 1	n := 1	S := m+n, 其值为 2
	n := 2	S := m+n, 其值为 3
	n := 3	S := m+n, 其值为 4
m := 2	n := 1	S := m+n, 其值为 3
	n := 2	S := m+n, 其值为 4
	n := 3	S := m+n, 其值为 5

例 3.14 打印九九乘法表。

分析：一般二维表格可以用双重循环处理，输出。若要输出九九乘法表，只需设两个循环变量 i 和 j 分别用来控制行和列的输出即可。

参考程序如下：

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
var
  i,j:integer;                                         //定义循环控制变量
begin
  Writeln('===== 九九乘法表 ===== ');
  for i := 1 to 9 do                                  //i 为外循环控制变量
  begin
    j := 1;
    while j <= i do                                    //内循环,j 控制列数
    begin
      write(i,' × ',j,' = ',i * j:2);                //按格式控制输出
      write(' ');                                       //输出间隔
      j := j + 1;                                       //使内循环趋于结束
    end;
    writeln;                                           //输出换行符号
  end;
  readln;                                              //暂停一下,让用户看输出结果
end.
```

程序运行结果如图 3-23 所示。

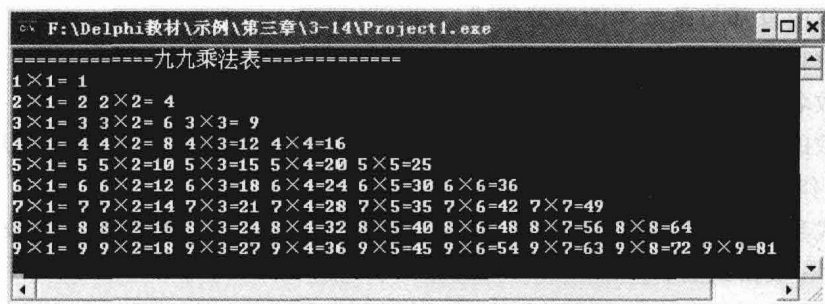


图 3-23 运行结果

## 3.4 函数与过程

通常,在程序设计中,会发现一些程序段在程序的不同地方反复出现,此时可以将这些程序段作为相对独立的整体,用一个标识符给它起一个名字,凡是程序中出现该程序段的地方,只要简单地写上其标识符即可。这样的程序段称为子程序。

子程序的使用不仅缩短了程序,节省了内存空间及减少了程序的编译时间,而且有利于结构化程序设计。因为一个复杂的问题总可将其分解成若干个子问题来解决,如果子问题依然很复杂,还可以将它继续分解,直到每个子问题都是一个具有独立任务的模块。这样编制的程序结构清晰,逻辑关系明确,无论是编写、阅读、调试还是修改,都会带来极大的好处。

在一个程序中可以有主程序而没有子程序(前面的控制台程序都是如此),但不能没有主程序,也就是说不能单独执行子程序。Delphi 中子程序有两种形式:函数和过程。

### 3.4.1 函数

在此之前,曾经介绍并使用了 Delphi 提供的各种标准函数,如 Sqrt(开平方)、Ord(求序号)等,这些函数为编写程序提供了很大的方便。但这些函数只是常用的基本函数,编程时经常需要自定义一些函数。

#### 1. 函数的说明

在 Delphi 中,函数也遵循先说明后使用的规则,在程序中,函数的说明放在调用该函数的程序(主程序或其他子程序)的说明部分。

函数定义的一般格式:

```
function <函数名> (<形式参数表>):<函数返回类型>; //函数首部
    <说明部分>;
begin
    语句 1;
    ...
    语句 n
end;
```

//函数体

函数体执行部分,需要给函数名赋值或者给 Result(系统隐含变量)赋值,这样执行时才会给主程序返回数据。

说明:

(1) 函数由首部与函数体两部分组成。

(2) 函数首部以关键字 function 开头。

(3) 函数名是用户自定义的标识符。

(4) 函数的类型也就是函数值的类型,所求得的函数值通过函数名传回调用它的程序。可见,函数的作用一般是为了求得一个值。

(5) 形式参数简称形参,形参即函数的自变量。自变量的初值来源于函数调用。在函数中,形参的一般格式为:

变量名表 1: 类型标识符 1; 变量名表 2: 类型标识符 2; ...; 变量名表 n: 类型标识符 n;

可见形参表相当于变量说明,对函数自变量进行说明。但应特别注意,此处只能使用类型标识符,而不能直接使用类型。

(6) 当没有形参表(当然要同时省去一对括号)时,称为无参函数。

(7) 函数体与程序体基本相似,由说明部分和执行部分组成。

(8) 函数体中的说明部分用来对本函数使用的标号、常量、类型、变量和子程序加以说明,这些量只在本函数内有效。

(9) 函数体的执行部分由 begin 开头,end 结束,中间有若干用分号隔开的语句,只是 end 后应跟分号,不能像程序那样用句号“.”。

(10) 在函数体的执行部分,至少应该给函数名赋一次值,以使在函数执行结束后把函数值带回调用程序。但是 Delphi 中更流行的做法是用 Result 给函数赋返回值,而不是用函数名,这样的代码更易读。

在主程序中,把函数的全部说明放在主程序的变量说明和程序体之间,然后在主程序的执行部分就可以直接调用自定义函数了。

**注意:** 在函数的说明部分要用形参,但在程序的执行部分调用自定义函数时就得用实参了。

自定义的函数在调用前要先说明,在主程序中的位置如下:

```
PROGRAM 程序名 (INPUT, OUTPUT);
    FUNCTION 函数名 (形参表): 函数类型;
    VAR 函数变量说明;
    BEGIN
        函数语句
    END; {FUNCTION}
    VAR 主程序变量说明;
    BEGIN
        主程语句
    END . {PROGRAM}
```

## 2. 函数的调用

可以在任何与函数值类型兼容的表达式中调用函数,或者说,函数调用只能出现在允许表达式出现的地方,或作为表达式的一个因子。

函数调用方式与标准函数的调用方式相同。

函数调用的一般格式:

<函数名>

或

<函数名>(<实在参数表>)

说明:

(1) 实在参数简称实参。实参的个数必须与函数说明中形参的个数一致,实参的类型与形参的类型应当一一对应。

(2) 调用函数时,一般实参必须有确定的值。

(3) 函数调用时首先计算实参的值,然后将该值“赋给”对应的形参。

(4) 对于控制台程序,函数定义的位置应该放在主程序 program 之后,begin 之前,也就是主程序的说明部分;当程序规模比较大时,应该放到一个单元文件中,然后在主程序中通过 uses 引用。

**例 3.15** 函数调用举例:编写判断一个整数  $n$  是否是素数的函数。

分析:一个数,如果只有 1 和它本身两个因数,这样的数叫做质数,又称素数。

根据题目要求,如果对任意输入的一个数  $n$ ,利用枚举法只要能找出  $2 \sim n-1$  之间的任一个数  $i$ ,只要满足  $n$  能被  $i$  整除,就可以说明  $n$  不是素数;反过来, $n$  只能被 1 和它自己整除, $n$  即为素数。

参考程序如下:

```
program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
function prime(n:integer):Boolean;                    //定义判断是否是素数的整型函数
var i:integer;
begin
    prime := True;                                     //开始假设是素数,可以用 Result := True;代替
    for i := 2 to n-1 do                               //根据定义找因子
    begin
        if(n mod i = 0) then                           //如果为真,表明不是素数
        begin
            prime := False;                             //不是素数,可以用 Result := False;代替
            break;                                       //提前结束循环
        end;
    end;
end;
var
    num:integer;                                       //定义变量
begin
    writeln('请输入一个数:');                          //输入提示
    readln(num);                                       //读入整数,将其值赋给变量 num
    if prime(num) then                                //调用自定义函数
        writeln(num,'是素数!')
    else
        writeln(num,'不是素数!');
    readln;
end.
```

程序运行结果如图 3-24 所示。

自定义函数只是主程序的说明部分,若主程序中没有调用函数,则系统不会执行函数子程序。当主程序调用一次函数时,则将实在参数的值传给函数的形式参数,控制转向函数子

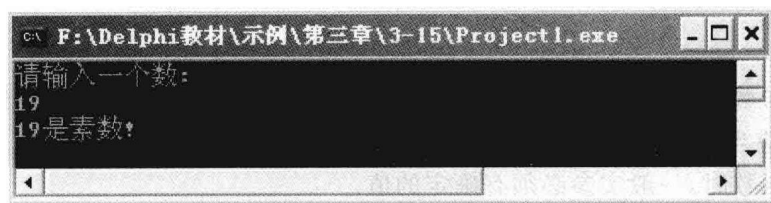


图 3-24 运行结果

程序去执行,子程序执行完毕后自动返回调用处。

### 3.4.2 过程

过程和函数一样,也是子程序。一个过程对应一个需要完成的任务。Delphi 中提供了不少标准过程,如 READ、WRITE、DELETE、NEW 和 DISPOSE 等,这些标准过程在程序中可以直接调用。但仅仅这些标准过程还不能满足需要,还要自己定义过程,就像函数一样。但函数必须以值的形式返回,而过程不一定返回一个值,只是执行一个任务而已;函数只能返回一个值,而过程可以返回不止一个值。所以函数不能取代过程。

#### 1. 过程的说明

过程说明的一般格式为:

```
procedure <过程名> (<形式参数表>); {过程首部}
<说明部分>;
begin
    语句 1;
    ...
    语句 n
end;
```

//过程体

说明:

- (1) 过程首部以关键字 procedure 开头。
- (2) 过程名是用户自定义的标识符,只用来标识一个过程,不能代表任何数据,因此不能说明“过程的类型”。
- (3) 形参表没有(当然要同时省去一对括号)时,称为无参过程。
- (4) 形参表的一般格式如下:

[var] 变量名表: 类型; ...; [var] 变量名表: 类型

其中带 var 的称为变量形参,简称为变参;不带 var 的称为值形参,简称为值参。在函数中,形参一般都是值形参,很少用变量形参(但可以使用)。

例如,下列形参表中:

```
(x,y:real;n:integer;var w:real;var k:integer;b:real)
```

x、y、n、b 为值形参,而 w、k 为变量形参。

调用过程时,通过值形参给过程提供原始数据,通过变量形参将值带回调用程序。因此可以说,值形参是过程的输入参数,变量形参是过程的输出参数。有关变参,将在后面的内

容中具体叙述。

(5) 过程体与程序、函数体类似。与函数体不同的是：函数体的执行部分至少有一个语句给函数名赋值，而过程体的执行部分不能给过程名赋值，因为过程名不能代表任何数据。

(6) 过程体的说明部分可以定义只在本过程有效的标号、常量、类型、变量和子程序等。

## 2. 过程的调用

过程调用是通过一条独立的过程调用语句来实现的，它与函数调用完全不同。过程调用与调用标准过程(如 write、read 等)的方式相同。调用的一般格式为：

<过程名>

或

<过程名>(实在参数表)

说明：

- (1) 实参的个数、类型必须与形参一一对应。
- (2) 对应于值形参的实参可以是表达式，对应于变量形参的实参只能是变量。
- (3) 过程调用的步骤为：计算实参的值；将值或变量的“地址”传送给对应的形参；执行过程体；返回调用处。

## 3. 过程与函数的主要区别

- (1) 过程的首部与函数的首部不同。
- (2) 函数通常是为了求一个函数值，而过程可以得到若干个运算结果，也可用来完成一系列的数据处理，或用来完成与计算无关的各种操作。
- (3) 调用方式不同。函数的调用出现在表达式中，而过程调用是一个独立的语句。

**例 3.16** 过程调用举例：编写一个程序输出下面的图形。

```
*
* *
* * *
* * * *
* * * * *
* * * * *
```

分析：从图中可以看出每行 '\*' 的数量跟它的行号有倍数的关系，可以使用前面学习的二重循环打印出上图形，程序中可以设置一个过程打印出 n 个连续的 \* 号与空格。

参考程序如下：

```
program Project1;
{$ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
procedure draw_a_line(n:integer); {该过程打印出连续 n 个星号,并换行}
var j:integer;
begin
    for j := 1 to n do
        write(' * ');
    writeln;
```



```

    end;
var
    i:integer;           //定义变量
begin
    for i:=1 to 6 do
        draw_a_line(i);{调用过程,第 I 行打印 i 个连续星号}
    readln;
end.

```

程序运行结果如图 3-25 所示。

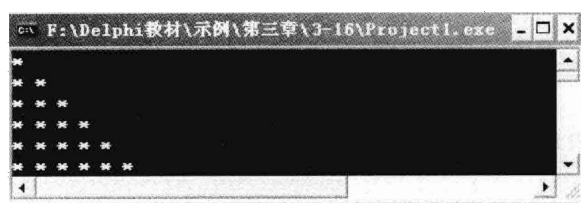


图 3-25 运行结果

### 3.4.3 函数、过程的数据传递

在程序调用函数或过程子程序时,调用程序将数据传递给被调用的子程序,而当子程序运行结束后,结果又可以通过函数名、变参返回给调用程序。当然也可以用全局变量等形式实现数据的传递。

Delphi 语言子程序中的形式参数有数值形参(值参)和变量形参(变参)两种。

#### 1. 数值参数

值参的一般格式:

形式变量: 类型

应该强调的是:

- (1) 形参表中只能使用类型标识符,而不能使用类型。
- (2) 值形参和对应的实参必须一一对应,包括个数和类型。
- (3) 实参和值形参之间数据传递是单向的,只能由实参传送给形参,相当于赋值运算。
- (4) 一个特殊情况是,当值形参是实型变量名时,对应的实参可以是整型表达式。
- (5) 值形参作为子程序的局部变量,当控制返回程序后,值形参的存储单元释放。

#### 2. 变量形参

变量形参的一般格式:

var 形式变量: 类型

跟值参不同的是,形参前加关键字 var。

应该注意的是:

- (1) 与变量形参对应的实参只能是变量名,而不能是表达式。
- (2) 与变量形参对应的实参可以根据需要决定是否事先有值。
- (3) 变量形参与对应实参的类型必须完全相同。
- (4) 对变量形参,运行时不另外开辟存储单元,而是与对应的实参使用相同的存储单元。

元。也就是说,调用子程序时,是将实参的地址传送给对应的变量形参。

(5) 当控制返回到调用程序后,变量形参的存储单元不释放,但变量形参本身无定义,即不得再使用。

(6) 选用形参时,到底是使用值形参还是变量形参,应慎重考虑。值形参需要另开辟存储空间,而变量形参会带来一些副作用。一般在函数中使用值形参,而在过程中才使用变量形参,但也有例外。

数值参数与变量形参比较举例:写出下列两个程序的运行结果。

```
program ex1;
{ $ APPTYPE CONSOLE}
//自定义过程(值参)
procedure swap(x,y:integer);
var t:integer;
begin
    t:=x;x:=y;y:=t;
end;
var a,b:integer;
begin
    a:=1;b:=2;
    writeln(a:3,b:3);
    swap(a,b); //调用过程
    writeln(a:3,b:3);
end.
```

```
program ex2;
{ $ APPTYPE CONSOLE}
//自定义过程(变参)
procedure swap(Var x,y:integer);
var t:integer;
begin
    t:=x;x:=y;y:=t;
end;
var a,b:integer;
begin
    a:=1;b:=2;
    writeln(a:3,b:3);
    swap(a,b); //调用过程
    writeln(a:3,b:3);
end.
```

分析:这两个程序唯一的区别是 ex1 中将 x,y 作为值形参,而 ex2 中将 x,y 作为变量形参,因此在 ex2 中对 x,y 的修改实际上是对调用该过程时与它们对应的变量 a,b 的修改,故最后 a,b 的值为 2,1。而 ex1 中调用 swap 过程时,只是将 a,b 的值传递给 x,y,之后在过程中的操作与 a,b 无关。

ex1 的运行结果为:

```
1 2
1 2
```

ex2 的运行结果为:

```
1 2
2 1
```

### 3.4.4 全局变量、局部变量及它们的作用域

在主程序的说明部分和子程序的说明部分均可以说明变量,但它们的作用范围是特定的。

#### 1. 局部变量及其作用域

在介绍过程和函数的说明时,凡是在子程序内部作用的变量,应该在本子程序内加以说明。这种在子程序内部说明的变量称为局部变量。形式参数也只是在该子程序中有效,因此也属于局部变量。

一个变量的作用域是指在程序中能对此变量进行存取的程序范围。因此,局部变量的作用域就是其所在的子程序。实际上,局部变量只是当其所在的子程序被调用时才具有确定的存储单元,当控制从子程序返回到调用程序后,局部变量的存储单元就被释放,从而变得无定义。

事实上,在子程序内定义的标号、符号常量、类型、子程序也与局部变量具有相同的作用

用域。

## 2. 全局变量及其作用域

全局变量是指在主程序的说明部分中说明的量。全局变量的作用域分为两种情况：

(1) 当全局变量和局部变量不同名时,其作用域是整个程序范围(自定义起直到主程序结束)。

(2) 当全局变量和局部变量同名时,全局变量的作用域不包含局部变量的作用域。

### 例 3.17 变量作用范围举例。

```
program Project1;
{ $ APPTYPE CONSOLE}           //编译预处理指令,表示控制台程序
var
m:integer; {m 为全局变量}
    procedure test2;
        var m:integer; {定义 m 为局部变量}
        begin
            m := 100;
        end;
begin {主程序}
    m := 5;
    writeln('Before the test2 call,the m is: ',m);{输出过程调用前的 m 值}
    test2; {调用过程 test2}
    writeln('After the test2 call,the m is: ',m); {输出过程调用后的 m 值}
    readln;
end.
```

程序运行结果如图 3-26 所示。

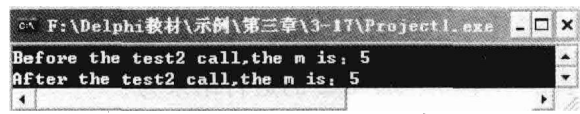


图 3-26 运行结果

主程序开始执行后,全局变量  $m$  被赋初值 5;调用过程  $\text{test2}$  后,由于有与全局变量同名的局部变量  $m$ ,故过程中起作用的是局部变量  $m$ ,而全局变量  $m$  暂时被屏蔽不受影响。过程调用完毕返回主程序后,全局变量  $m$  起作用,局部变量  $m$  所占用的临时单元被释放。从这个程序可以看出,当过程或函数内包含与全局变量同名的变量时,全局变量的作用域将不包括在此过程或函数中。

## 3.4.5 函数和过程的递归调用

函数和过程调用它们本身,称为递归调用。过程或函数  $A$  直接调用  $A$  本身,称为直接递归。过程或函数  $A$  调用过程或函数  $B$ , $B$  又调用  $A$ ,称为间接递归。在递归调用中,一个过程或函数执行的某一步要用到它自身的上一步(或上几步)的结果。

递归在解决某些问题中,如在处理阶乘运算、级数运算和幂指数运算等方面是十分有用的方法。它可以使某些看起来不易解决的问题变得容易解决,写出的程序较简短。但是递归通常需要花费较多的机器时间和占用较多的存储空间,总体运行效率不太高。

为了防止递归调用无终止地进行,必须在函数或者过程内有终止递归调用的手段。常用的办法是加条件判断,满足某种条件后就不再作递归调用,然后逐层返回。

函数或者过程的递归调用过程可以分为两个阶段:一个是递推阶段,将原问题不断地分解为新的子问题,最终达到已知的条件,这时递推阶段结束。另一个是回归阶段,从已知条件出发,按照“递推”的逆过程逐一求值回归,最终到达“递推”的开始处,完成递归调用。

**例 3.18** 用递归法求  $n!$ 。

分析: 因为  $n! = (n-1)! \times n$ , 而  $(n-1)! = (n-2)! \times (n-1) \cdots, 1! = 1$ 。故其递归方式为: 当  $n > 1$  时, 有  $n! = n \times (n-1)!$ ; 递归终止条件为: 当  $n = 0$  或  $1$  时, 有  $n! = 1$ 。其数学公式如下:

$$n! = \begin{cases} 1 & (n = 0, 1) \\ n \times (n-1)! & (n > 1) \end{cases}$$

可以看到,在定义  $n!$  的表达式中又出现了  $(n-1)!$ , 这种定义方式称为递归定义。通常,对于采用递归定义的数学公式可以编写成递归函数。

参考程序如下:

```
program Project1;
{$APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
//定义求阶乘的递归函数
function ff(n:integer):integer; //n为形式变量,局部作用
var f:integer; {定义 f 为局部变量}
begin
    if (n=0) or (n=1) then
        f:=1
    else
        f:=ff(n-1)*n; //递归调用
        ff:=f;        //返回计算数据,它也等价于 result:=f;
    end;
var n,fact:integer; //此处 n 为主程序变量
begin {主程序}
    writeln('请输入一个数');
    readln(n);
    fact:=ff(n); {调用递归函数 ff}
    writeln(n,'!= ',fact);
    readln;
end.
```

程序运行结果如图 3-27 所示。



图 3-27 运行结果

本例中,函数  $ff()$  是递归函数。如果  $n := 4$ , 则计算  $4!$  的执行过程如图 3-28 所示。

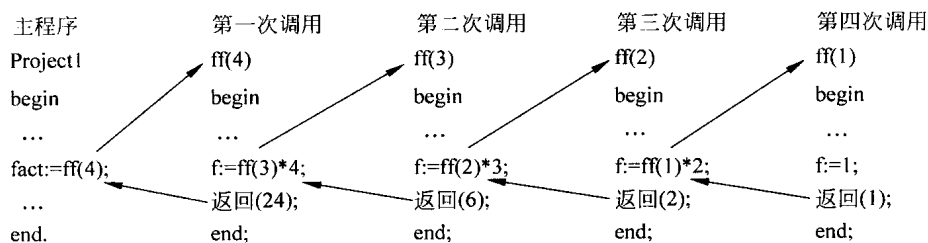


图 3-28 递归调用示意图

第一次调用时,形参接受值为 4,满足  $n>1$  的条件,所以执行语句  $f:=ff(n-1)*n;$ ,在执行该语句时又调用  $ff(n-1)$ ,执行  $ff(3)$ ,这是第二次调用该函数,此时  $n$  为 3,仍满足  $n>1$  的条件,所以进入第三次调用,执行  $ff(2)$ ,同理,继续进入第四次调用  $ff(1)$ ,这时执行语句  $f:=1;$ ,然后返回  $f$  的值,至此递推阶段结束,回归阶段开始。每次返回时,函数的返回值乘以  $n$  的当前值,结果作为本次调用的返回值返回给上次调用中,最后返回值为 24,这就是  $4!$  的计算结果。

一般能够使用递归函数解决的问题具有以下几个特点:

- (1) 原问题能分解为一个新问题,而新问题又用到了原有的解法,这就出现了递归。
- (2) 按照这个原则分解下去,每次出现的新问题都是原问题简化的子问题。
- (3) 最终分解出来的新问题是一个已知解的问题。也就是说,递归应该是有限的,不能让函数无休止地调用其自身。即编写递归函数时,应该有使递归结束的约束条件,以便在不需要递归调用时,函数能返回。

### 3.4.6 函数和过程重载

Delphi 允许定义多个同名函数或者过程,只要这些函数或者过程有不同参数集(至少有不同类型的参数)。这个功能称为函数或者过程重载。调用重载函数或者过程时,Delphi 编译器通过检查调用中的参数个数、类型和顺序来选择相应的函数或者过程。函数或者过程重载常用于生成几个进行类似任务而处理不同数据类型的同名函数。

函数或者过程重载通常用于不同数据类型用不同程序逻辑进行类似的操作。

重载的思想很简单:编译器允许用同一个名字定义多个函数或过程,只要它们所带的参数不同。实际上,编译器是通过检测参数来确定需要调用的例程。

下面是从 VCL 的数学单元(Math Unit)中摘录的一系列函数:

```
function Min(A,B: Integer): Integer; overload;
function Min(A,B: Int64): Int64; overload;
function Min(A,B: Single): Single; overload;
function Min(A,B: Double): Double; overload;
function Min(A,B: Extended): Extended; overload;
```

当调用方式为  $Min(10,20)$  时,编译器很容易就能判定调用的是上列第一个函数,因此返回值也是个整数。

声明重载过程或函数(可以不区分,称为例程)有两条原则:

- (1) 每个例程声明后面必须添加 `overload` 关键字。
- (2) 例程间的参数个数或(和)参数类型必须不同,返回值不能用于区分各例程。

### 例 3.19 函数和过程重载举例：重载 show 过程。

程序如下：

```
program Project1;  
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序  
    procedure show(msg:integer);overload;              //重载过程,参数类型为整型  
    begin  
        writeln(msg:8);  
    end;  
    function show(msg:Double):integer;overload;        //重载函数,参数类型为浮点型  
    begin  
        writeln(msg:8:2);  
        Result := 0;                                  //或者 show := 0;  
    end;  
    procedure show(msg:String);overload;              //重载过程,参数类型为字符串型  
    begin  
        writeln(msg);  
    end;  
begin { 主程序 }  
    show(12);  
    show('你好,世界!');  
    show(3.1415);  
    readln;  
end.
```

程序运行结果如图 3-29 所示。

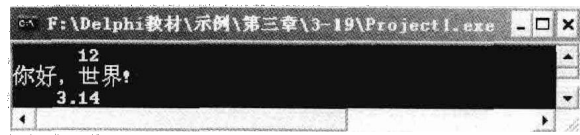


图 3-29 运行结果

上面的程序段定义了三个有相同名字的过程和函数 show,但形参不同,分别为 integer、double 和 string 类型。

#### 3.4.7 Forward 声明

当使用一个标识符(任何类型)时,编译器必须已经知道该标识符指的是什么。为此,通常需要在例程使用之前提供一个完整的声明。然而在某些情况下可能做不到这一点,例如过程 A 调用过程 B,而过程 B 又调用过程 A,那么写过程代码时,不得不调用编译器尚未看到其声明的例程。

欲声明一个过程或函数,而且只给出它的名字和参数,不列出其实现代码,需要在句尾加 forward 关键字:

```
procedure AddData (d1:integer;?d2:byte;d3:real); forward;
```

在后面应该补上该过程的完整代码,不过该过程代码的位置不影响对它的调用。下面的例子没什么实际意义,看过后读者会对上述概念有所认识。

其程序如下：

```

program Project1;
{ $ APPTYPE CONSOLE}                                //编译预处理指令,表示控制台程序
    procedure AddData (d1:integer; d2:byte;d3:real); forward; //提前声明
    procedure Compute;
    begin
        AddData(10,100,1000);                        //被调用的过程在主调过程之后
    end;
    procedure AddData (d1:integer; d2:byte;d3:real); //被调用过程代码
    var sum:real;
    begin
        sum := d1 + d2 + d3;
        writeln('d1 + d2 + d3 = ',sum:5:0);
    end;

begin {主程序}
    Compute;                                          //调用子程序
    readln;
end.

```

尽管 forward 过程声明在 Delphi 中不常见,但是有一个类似的情况却经常出现。当在一个单元的 interface 部分声明一个过程或一个函数时,它被认为是一个 forward 声明,即使没有 forward 关键字也一样。实际上不可能把整个例程的代码放在 interface 部分,不过必须在同一单元中提供所声明例程的实现。

### 3.4.8 函数和过程默认参数

Delphi 语言允许给函数和过程的参数设定默认值,这样调用函数和过程时该参数可以加上,也可以省略。下面将前面过程的三个形式参数设定了两个变量的默认值:

```

procedure AddData (d1:integer;d2:byte = 10;d3:real = 100);
var sum:real;
begin
    sum := d1 + d2 + d3;
    writeln('d1 + d2 + d3 = ',sum:5:0);
end;

```

使用这一定义,就可以用下面任一种方式调用过程:

```

AddData(20);                                //默认使用 d2 的 10 和 d3 的 100
AddData(20,15);                             //默认使用 d3 的 100
AddData(20,15,200);                         //都使用实际参数的值

```

默认参数使用的主要规则:调用时只能从最后一个参数开始进行省略,换句话说,如果要省略一个参数,必须省略它后面所有的参数。例如,

```

procedure AddData(d2:byte = 10;d3:real = 100;d1:integer);

```

这种形式是错误的。

默认参数的使用规则还包括:

(1) 带默认值的参数必须放在参数表的最后面。

(2) 默认值必须是常量。显然,这限制了默认参数的数据类型,例如动态数组和界面类型的默认参数值只能是 nil。至于记录类型,则根本不能用作默认参数。

默认参数必须通过值参或常参传递。变量参数 var 不能有默认值。

## 本章小结

在这一章中主要要求学生了解的要点有基本语句和结构语句的不同使用方法以及过程和函数的概念、声明定义与使用。重点是顺序结构、分支结构和循环结构的控制语句实现以及过程和函数的使用。难点是循环嵌套以及过程和函数中参数的传递,嵌套及递归调用。

顺序结构用计算机解题的基本方法解决简单的问题,可以帮助读者培养良好的程序设计习惯和风格。分支结构是根据输入数据域或中间计算结果的情况,选择一组语句执行(在不同情况下,选择不同的语句组来执行)。在编程时,需要将所有这些情况都考虑进去,并写出在各种情况下所对应的语句组。循环结构是利用计算机的高速运算处理特性和逻辑控制特性,重复执行某些计算语句,以完成大量的计算处理要求。

函数通常用来求一个值,而过程可以用于求解多个值或完成某个动作。函数与过程是将原来较大的问题分解成许多较小的容易解决的问题。较小的问题,常常可以用一些单独的函数或过程来描述。求解原来的问题就变成了对这些函数与过程的调用。

## 思考与练习

1. 条件语句有哪几种形式? 它们在什么情况下适用?
2. 循环语句有哪几种形式? break 语句与 continue 语句有何异同?
3. 如何声明、定义及调用函数和过程?
4. 什么是变量的作用域? 根据变量的作用域,可将变量分为哪几类?
5. 某市出租车 3 公里的起租价为 6 元,3 公里以外按 1.5 元/公里计费。现编写程序,要求:输入行车里程数,输出应付车费。
6. 编写计算阶乘的函数,在主程序中调用函数求  $4!+6!+9!$ 。
7. 输入 0~6,调用自定义函数来显示英汉对照的星期。
8. 有两个红球、三个黄球、四个白球,任意取四个球,其中必须有一个红球,编程输出所有可能的方案。



面向对象技术的特点可以概括为抽象性、继承性、封装性和多态性。

(1) 抽象性：指对现实世界中某一类实体或事件进行抽象，从中提取共同信息，找出共同规律；反过来，又可以把它们集中在一个集合中，定义为所设计目标系统中的对象。

(2) 继承性：新的对象类可以通过继承原有对象类的某些特征或全部特征而产生出来，原有的对象类称为基类，新的对象类称为派生类，派生类可以直接继承基类的共性，还可以添加自己所独有的特点。继承性简化了新对象类的设计。

(3) 封装性：是指对象的使用者通过预先定义的接口关联到某一对象的服务和数据时，无须知道对象内部运行的细节。

(4) 多态性：是指不同类型的对象对相同的操作做出适当的不同响应的能力。

前面几章为了讲 Delphi 基本语法，主要采用面向过程的程序设计思想阐释。实际上 Delphi 是基于面向对象(OOP)技术的开发环境。面向对象技术的使用在可视化开发环境中得到了增强。

通过本章的学习，读者应该可以掌握面向对象的基本概念以及面向对象技术在 Delphi 中的实现方法，进行基本的面向对象编程。面向对象技术为开发复杂的程序提供了重要的手段。

本章主要包括以下内容：

- 类与对象；
- 方法；
- 类的继承、封装和多态；
- 异常处理。

### 4.1 类 与 对 象

类和对象并不是一回事。类是一种类型的定义，而对象则是这种叫做“类”的类型的实例。在 Delphi 中有很多种数据类型，如 Real、Char、Integer 和 Record 等类型。而类则是一种用户定义的数据类型，它具有自己的说明和一些操作。类中含有一些内部数据和一些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。

类的定义在一些面向对象的书上可能比较晦涩。可以把类想象为一种特殊的 Record 类型，其中不但可能包含数据，而且还可能包含函数和过程(在 OOP 中称为方法)。这些数据和函数被统称为类的成员。

一旦定义一个类类型，就可以创建基于该类的对象，且对象的数量可以无限。这就好像

用一个做饼干的模具来生产大量的同一种饼干一样。做饼干的模具相当于类,而生产出的饼干则是对象——饼干的实例,也就是具体的某一块饼干。

### 4.1.1 类

类(class)描述了具有相似性质的一组对象,这组对象具有相同的数据结构,相同的操作,它定义了这组对象共同的属性和操作。类是一个抽象的概念,也称类类型,可以把类视为特殊数据类型。

类是面向对象程序设计的核心,它实际上是一种新的数据类型,也是实现抽象数据类型的工具。类描述了数据结构(对象属性)、算法(方法)和外部接口(消息协议),因而提供了完整的解决问题的能力。

一个类具有一个基类(也称父类),它从中继承所有的数据成员、属性和方法。如果不列出一个明确的基类,Delphi 将使用 TObject 作为它的父类。

声明类数据类型使用关键字 class。语法如下:

```
TYPE
  类名 = Class(父类)
  private
    变量或者属性列表;
    方法列表;
  Protected
    变量或者属性列表;
    方法列表;
  Public
    变量或者属性列表;
    方法列表;
END;
```

这里,类名是任何有效标识符,父类是可选的,成员列表声明类的各成员,也就是它的字段、方法和属性。若省略了父类,则新定义类直接继承自内置的类 TObject。Delphi 一般习惯上使用以 T 开头的类型名称,但这不是必须的。

例如:以下代码声明了一个 TMyClass 类。

```
Type                                     //类型定义关键字
  TMyClass = Class                       //类声明
  private
    FNum : Integer;                     //定义类的私有变量
    Procedure SetNum(Value: Integer);   //定义类的方法
  public
    Property pNum: Integer read FNum write SetNum; //定义类的属性
  end;
Procedure TMyClass.SetNum(Value: Integer); //TMyClass 类的方法实现
begin
  FNum := Value;
end;
```

类的访问属性分为三个级别: private(私有)、public(公有)和 protected(保护)。在任何类中,都有外部可以访问的 public 部分和仅供内部访问的 private 部分。类将内部实现细节

隐藏起来,使类的用户不用对此操心,而用户接口是公开的。

一般将成员数据设置为 `private`,对这些数据的访问可通过它的某些成员函数来访问,而这些成员函数将被声明为 `public`。访问级别为保护的(`protected`)类成员和私有的(`private`)类成员相似,该类的用户不能访问,但它的派生类的用户可以访问该派生类中相应的成员。

### 4.1.2 类的成员

在类的声明中,注意到一个类中包含了数据成员(Field)、方法(Method)和属性(Property)。通常把类的数据成员、方法和属性称为类成员。

#### 1. 数据成员

又称为域,它们是一些在类中声明的定义好的私有或者公有变量。也可以把数据成员称为“字段”,但它跟数据库中字段的含义有本质区别。

例如前面 TMyClass 类中的私有变量 FNum 为该类的数据成员。

#### 2. 方法

方法则是一些封装在类中的过程和函数,用于执行类的操作,完成类的任务。

例如前面 TMyClass 类中的过程 Procedure SetNum(Value:Integer)为该类的方法。

#### 3. 属性

通常用做访问对象数据的接口,对象的数据一般存储在数据成员中。属性有存取设定,它决定数据如何被读取和修改。从程序的其他地方(在对象本身之外)来看,属性在很大程度上很像一个数据成员,但本质上它更像方法,比如类的实例(对象)中并不为它分配内存。可以这么说,这种叫做属性的方法是用于对私有数据成员进行读写操作的,但在使用时与直接使用数据成员一样。不妨把它看做是一种含有约束机制的数据成员。

例如前面 TMyClass 类中的 pNum 为该类的属性。

### 4.1.3 对象

在 Delphi 程序中,光有类还无法使用,必须对类进行实例化。

#### 1. 对象的概念

对象(Object)是类的实例。通俗点说,对象就是类类型的变量。对象可以使用类的方法、属性和事件。例如:

```
Var fmyClass:TMyClass;//声明了一个 fmyClass 对象的变量,其类型为 TMyClass.
```

类和对象是两个常用的词,也很容易混淆。一个类是一种用户定义的数据类型,一个对象是类的一个实例。对象是真正的实体。对象和类之间的关系与变量和类型之间的关系是相同的。

#### 2. 构造函数

Delphi 通过调用类的一个构造函数来建立类的实例,构造函数主要用来为创建对象并为对象中的成员分配内存并进行初始化,以使得对象处在可以使用的状态。

Delphi 的类定义时一般有一个称为 Create()的构造函数,如果没有则隐含使用父类的构造函数,甚至一个类还可以有定义多个重载的构造函数。

在 Delphi 中构造函数不能自动调用,程序员必须自己调用构造函数创建对象。调用构造函数的语法如下:

```
fmyClass := TMyClass.Create;
```

```
//创建 fmyClass 对象
```

**注意：**这里调用构造函数的方法有点特殊，是通过类型来引用一个类的 Create 方法，而不是像其他方法那样通过实例来引用。

### 3. 对象的特点

通过调用构造函数创建类的实例就是所谓的实例化，对象是类的动态实例。动态实例包含了在类及它的祖先类中声明的所有类成员生成的独自的内存单元。使用的对象变量实际上是对象的一个引用。该引用保存在栈中，而对象总是动态分配在堆上，因此对象引用是真正指向对象的指针。程序员有责任管理对象的生命期，即负责生成对象并在适当的时候释放它们。

每个对象都有对类及其所有数据成员的一个独立的备份。一个数据成员不能在多个对象中共享；如果需要共享一个变量，应该在单元层次声明它或者间接使用它。

**例 4.1 类与对象示例：**设计一个简单模仿银行个人用户账户的类，并在程序中对其操作。

**分析：**根据现实中的银行账户功能，需定义一个类，能够模拟开户、存钱、取钱、查询余额功能。

在 Delphi 2007 中，把一个类的定义放在一个独立的单元文件中处理起来比较方便，因为单元具有接口部分和实现部分。

编程步骤如下：

(1) 建立新的应用程序。

在 Delphi 2007 集成开发环境中通过选择 File→New→Other 命令，在对话框中选择 Console Application，创建一个新的控制台应用程序。然后选择 File→Save 命令保存应用程序。

(2) 建立新的单元文件。

选择 File→New→Unit—Delphi for Win32 命令，创建一个新的单元文件 Unit1，这个单元文件被系统自动引用到控制台应用程序中。

(3) 在 Unit1 中编写一个模仿银行个人用户账户的类。

编写的代码如下：

```
unit Unit1;                                //单元文件的名字
interface                                  //接口部分的开始
Type                                       //类型定义关键字
  TBankAccount = Class                    //类声明
  private                                  //私有部分
    Balance: Real;                         //余额
    AccountNumber: string;                 //账号
  public                                   //公有部分
    Constructor Create(AccountNum: String; Amount: Real = 0.0); //构造函数, 开户
    procedure Deposit(Amount: Real);        //存钱
    procedure Withdraw(Amount: Real);       //取钱
    function InquiryBalance: Real;          //查询余额
  End;
Var                                       //声明变量或类的实例
  BankAccount : TBankAccount;
```

```

Implementation                                     //程序代码实现功能部分的开始
constructor TBankAccount.Create(AccountNum:String;Amount:Real = 0.0);    //初始化,开户
Begin
    AccountNumber := AccountNum;                                     //账号
    Balance := Amount;
End;
procedure TBankAccount.Deposit(Amount:Real);           //存钱
begin
    Balance := Balance + Amount;
end;
procedure TBankAccount.Withdraw(Amount:Real);         //取钱
begin
    Balance := Balance - Amount;
end;
function TBankAccount.InquiryBalance:Real;           //查询余额
begin
    Result := Balance;
end;
end.                                                    //实现部分结束

```

(4) 在控制台程序中编写模拟操作账户的行为代码。

```

program Project1;                                     //程序头部
{$APPTYPE CONSOLE}                                   //编译预处理指令,表示控制台程序
uses                                                  //引用其他单元的定义
    SysUtils,                                       //系统单元
    Unit1 in 'Unit1.pas';                          //用户自定义的单元
begin
    //直接使用 Unit1 中声明的变量或者对象
    BankAccount := TBankAccount.Create('User - 80',109.0);    //用 109.0 元创建账户 User - 80
    Writeln('开户以后,账户余额: ',BankAccount.InquiryBalance:8:2); //打印余额
    BankAccount.Deposit(200);                             //存钱
    Writeln('存完钱后,账户余额: ',BankAccount.InquiryBalance:8:2); //打印余额
    BankAccount.Withdraw(150.8);                           //取钱
    Writeln('取完钱后,账户余额: ',BankAccount.InquiryBalance:8:2); //打印余额
    Readln;                                                //暂停一下,让用户看屏幕显示结果
end.

```

(5) 保存运行。

运行结果如图 4-1 所示。

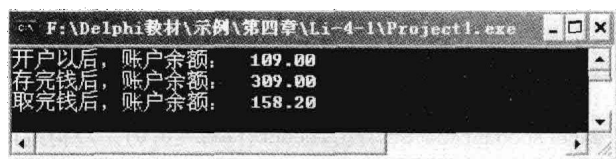


图 4-1 运行结果

TBankAccount 类具有两个数据成员: Balance 和 AccountNumber, 分别表示存款余额和账号, 它们是私有成员。该类还具有三个方法: Deposit、Withdraw 和 InquiryBalance, 分

别用于存款、取款和查询余额,它们都是公有成员。用户程序不能直接对 balance 进行存取,必须调用有关的方法 Deposit 或 Withdraw 才能修改 Balance,实现了对重要数据成员的保护。

#### 4.1.4 self 引用

在 Delphi 面向对象的编程中,类中所有的方法对数据成员进行存取,都是利用 Self 通过引用方式处理的。在大多数场合下,Delphi 允许忽略 Self 引用,而直接使用以变量方式存取所说明的成员。

其实,在每一个方法中,Delphi 声明 Self 变量为一个隐含参数。在普通方法中,Self 变量的值是对象的引用;在一个类方法中,Self 变量的值是类的引用。Self 是一个隐含的参数,是表示自身的变量,调用对象方法时,该参数便被传递了过来。

这相当于每个方法体中都有一个隐式的 with self do。也就是说,所有的数据成员、方法和属性在作用域中,无须显式地引用 Self 就可以访问它们。

它常被用于以下三种情况:

(1) 如果在方法内调用另外一个对象的方法时,需要将自身参数传递过去,那么就把它传递过去。

(2) 如果派生类和基类出现了数据成员的重名,想要访问基类的数据成员,而每次只能得到派生类本身的重名数据成员,使用强制转换“基类名(Self).数据成员”。

(3) 如果在方法内调用了与数据成员同名的局部变量,则使用变量名字只能访问到局部变量,使用“Self.变量名”便访问到了数据成员。

因此,在例 4.1 中的 TBankAccount 类中,procedure TBankAccount. Deposit(Amount: Real)方法中使用的 Balance := Balance + Amount 等价于 Self. Balance := Self. Balance + Amount。

#### 4.1.5 类运算符

##### 1. is 运算符

is 运算符执行动态类型检查,用来验证运行时一个对象的实际类型。

object is class

若 object 对象是 class 类的一个实例,或者是 class 派生类的一个实例,上面的表达式返回 True,否则返回 False(若 object 是 nil,则结果为 False)。比如:

```
if ActiveControl is TEdit then TEdit(ActiveControl).SelectAll;
```

上面的语句先检查一个对象(变量)是不是 TEdit 或它的派生类的一个实例,然后再决定是否把它转换为 TEdit。

##### 2. as 运算符

as 运算符执行受检查的类型转换。表达式:

object as class

返回和 object 相同的对象引用,但它的类类型是 class。在运行时,object 对象必须是

class 类的一个实例,或者是它的派生类的一个实例,或者是 nil,否则将产生异常。若 object 声明的类型和 class 不相关,也就是说,若两个类不同并且其中一个不是另一个的祖先,则发生编译错误。比如:

```
with Sender as TButton do
begin
  Caption := '&Ok';
  OnClick := OkClick;
end;
```

因为运算符优先权的问题,经常需要把 as 类型转换放在一对括号中,比如,

```
(Sender as TButton).Caption := '&Ok';
```

## 4.2 方 法

方法是在类中定义,用来实现对象操作的过程或者函数。方法是属于一个给定对象的过程和函数,反映的是对象的行为而不是数据。

### 4.2.1 方法的特征与调用

方法与普通函数和过程不同,是因为方法只应用于特定类及其祖先类的对象。另外,每一个方法都有一个隐含的参数,称为 Self,它引用作为方法调用主体的对象,这也是普通函数和过程所没有的。

调用方法和调用普通函数或过程相同,但要以对象引用作为方法名称的开头,例如:

```
BankAccount.Deposit(200); //调用对象 BankAccount 的存款方法
```

方法可以应用于类或通过继承作用于其子类。也就是说,可以调用一个在对象的类或它的任何祖先类中声明的方法。如果相同的方法在祖先类和派生类里面都有声明,则 Delphi 会根据情况来调用哪个层次的方法,如下面程序所示。

```
//类的声明
type
  TFigure = class //定义父类 TFigure
    procedure Draw;
  end;
  TRectangle = class(TFigure) //定义子类 TRectangle
    procedure Draw;
  end;
...
//类在程序中使用
var
  Figure: TFigure;
  Rectangle: TRectangle;
begin
  ...
  Figure.Draw; //调用父类 TFigure.Draw
  TRectangle(Figure).Draw; //调用 TRectangle.Draw, Figure 被强制转换为 TRectangle 类型
```

```

Rectangle.Draw;
...
end;
//调用子类 TRectangle.Draw

```

## 4.2.2 方法的类别

Delphi 中方法通常按照用途分,可分为普通方法、构造方法(Constructor 也称为构造函数、构造过程)、析构方法、类方法和消息处理方法;按照运行绑定机制分,可分为静态方法、虚方法和动态方法。

### 1. 普通方法

此类方法是在类内定义、单元内实现的,可以是过程,也可以是函数。普通方法通过设置保护方式确定它的调用范围。

#### 1) 定义普通方法

定义普通方法的语法如下:

```

//类的定义部分
TYPE
  TClassName = class(父类)                //某个类的类名为 TClassName
  Private                                  //或者 Public 或者 Protected
    ...;                                  //数据成员、属性或者其他方法
    procedure MethodNameM(参数列表);      //类内声明的普通过程方法 MethodNameM
    function MethodNameN(参数列表):类型;  //类内声明的普通函数方法 MethodNameN
  End;
//类的实现部分
  procedure TClassName.MethodNameM(参数列表);
  ...                                     //常量或者变量的声明
  Begin
  ...                                     //过程方法内语句
  End;
  function TClassName.MethodNameN(参数列表):类型;
  ...                                     //常量或者变量的声明
  Begin
  ...                                     //函数方法内语句
  End;

```

#### 2) 调用普通方法

调用一个普通方法的语法格式如下:

对象实例.方法名(实际参数)

假设声明了一个对象 Var fObject:TClassName;,则调用它的方法可以为:

```

fObject.MethodNameM(参数列表);
某变量 := fObject.MethodNameN(参数列表);

```

普通方法的实现与子程序的实现在形式上的差别是方法名前面要加上类名的限定。在一般方法的实现中,最好只访问自己类中的成员,不要访问类外的对象,防止对象未建立而发生保护性错误。可以调用祖先类的方法,但如果在派生类和基类中都定义了一个同名的方法,则派生类的方法会覆盖(或隐藏)基类的方法。





## 2. 构造方法

构造方法习惯上称为构造函数,用来创建类的实例(对象),并且对对象的数据成员进行初始化。构造方法之所以被称为构造函数,是因为它返回的是一个类的实例指针,即对象的引用。构造方法很像一般方法,只是将 Procedure 换成限定符 Constructor。

### 1) 构造方法定义

构造方法的格式如下:

```
//类的定义部分
TYPE
  TClassName = class(父类)                //某个类的类名为 TClassName
  Private                                  //或者 Public 或者 Protected
    ...;                                  //数据成员、属性或者其他方法
    Constructor CreateClassName(参数列表); //类内声明的构造函数
  End;

//类的实现部分
  Constructor TClassName. CreateClassName(参数列表);
  ...                                     //常量或者变量的声明
  Begin
    ...                                  //构造方法内语句
  End;
```

例如:

```
Type                                     //类型定义关键字
  TBankAccount = Class                  //类声明
  ...
  public                                //公有部分
    Constructor Create(AccountNum:String;Amount:Real = 0.0); //构造函数声明
  ...
  End;
  ...

//类实现部分
constructor TBankAccount.Create(AccountNum:String;Amount:Real = 0.0); //构造函数实现
Begin
  AccountNumber := AccountNum;          //账号
  Balance := Amount;
End;
...
```

### 2) 构造方法调用

调用构造方法创建对象实例的语法如下:

对象变量名 := 类名.构造方法名(多数表);

例如声明一个类的实例(对象)如下:

```
Var BankAccount : TBankAccount;
```

可以通过该类的构造函数生成该对象:

```
BankAccount := TBankAccount.Create('User - 80',109.0); //用 109.0 元创建账户 User - 80
```

构造方法创建类的实例,创建后返回该实例的引用。构造方法除了继承基类的构造方法外,一般还创建其他数据结构,还要对类的所有数据成员进行初始化。构造方法不是必须的,如果不声明自己的构造方法,就默认使用最近祖先类的构造方法。这就是说,如果不想对基类的构造方法进行扩充,就没有必要在程序中声明构造方法。

如果由于某种原因,构造方法执行失败了,对象引用的返回值便是 nil,因此在程序中有必要对那些对资源要求比较苛刻的构造方法实行保护。如果创建未成功,程序却仍按正常执行来访问新建对象的数据成员或方法,就会出现保护性错误。

### 3. 析构方法

析构方法习惯上称为析构函数,它是一个特殊的方法,用来销毁调用的对象并且释放它的内存。通常情况下,一个类只有一个析构方法 Destroy。这个析构方法通常不带任何参数(带参数的析构方法很少),因为目标明确,就是关闭、销毁。析构方法必须以 destructor 开头,其语法格式如下:

```
TYPE
    TClassName = class(父类)                //某个类的类名为 TClassName
    ...;                                     //数据成员、属性或者其他方法
    destructor Destroy;                      //析构函数
End;
```

要调用析构函数,必须使用一个实例对象的引用。比如:

```
MyObject.Destroy;
```

当析构函数被调用时,它里面的命令首先被执行,这通常包括销毁所有的嵌入对象以及释放为对象分配的资源;接下来,为对象分配的内存被清除。

考虑到一个类的析构方法可以不止一个,类声明时建议覆盖继承下来的析构方法,并不再声明其他的析构方法。

```
destructor Destroy; override;              //使用覆盖技术
```

在析构方法的实现部分最好有语句:

```
inherited Destroy;                         //继承祖先类的析构方法
```

不过,在程序中,一般情况下不要直接调用 Destroy 来销毁对象,而应该用 free 方法,因为 free 在销毁之前会检测对象是否为 nil,如果不为 nil 才销毁对象变量。

### 4. 类方法

在定义类的成员函数或过程时,如果在 procedure 或 function 前加上 class 保留字,则该方法称为类方法。类方法用类调用,而不是用对象调用,它们对类进行操作,而不是对具体的对象操作。

类方法(Class methods)是一类特殊的方法,它们在声明时要以 class 开头:

```
type
    TFigure = class
    public
    ...
    class procedure GetInfo(var Info: TFigureInfo); virtual;    //类方法声明
    ...
end;
```

实现时也以 class 开头：

```
class procedure TFigure.GetInfo(var Info: TFigureInfo);    //类方法实现
begin
    ...
end;
```

普通方法只能通过对象实例来调用，而类方法既可以通过对象实例调用，也可以通过类本身引用。类方法只是表明这个方法在逻辑上与这个类有联系。

类方法是作用于类而不是对象上的方法，要考虑到在类方法中使用类名调用时，各数据域都未创建，因此在类方法内不允许访问类的数据域。

### 5. 消息处理方法

消息处理方法（即 Message 方法）是用来响应动态分派的消息的。用 Message 方法来响应 Windows 的消息，这样就不用直接来调用它了。

在声明时，通过包含 Message 限定符来创建一个 Message 方法，并在 Message 后面跟一个介于 1~49 151 之间的整数常量，它指定消息的编号（ID）。对于 VCL 组件，Message 方法中的整数常量可以是 Message 单元中定义的 Windows 消息编号，这里还定义了相应的记录类型。一个 Message 方法必须是具有单一引用参数（Var）的过程。

例如：

```
const WM_COMMNOTIFY = WM_USER + $100;           //定义用户消息常量
TComm = class(TComponent)
private
    ...
    procedure WndProc(var Msg: TMessage);message WM_COMMNOTIFY;    //消息方法声明
    ...
End;
```

### 4.2.3 方法的绑定

所谓的方法绑定，就是建立方法调用（Method Call）和方法本体（Method Body）之间的关联。如果绑定动作发生于程序执行之前（由编译器和连接器完成，简称编连），就称为“早绑定”或“先期绑定”（Early Binding）；如果绑定动作在程序执行期才根据对象类型进行，则称为“晚绑定”或“后期绑定”（Late Binding）。晚绑定也被称为运行期绑定（Run-time Binding）或动态绑定（Dynamic Binding）。

根据方法的绑定机制可以将方法分为静态（static）方法、虚（virtual）方法和动态（dynamic）方法，其中虚方法和动态方法又可以是抽象（abstract）方法。例如：

```
TSomeClass = class
    procedure FMethodStatic;                //静态方法声明
    procedure IMethodVirtual;virtual;      //虚方法声明
    procedure IMethodDynamic;dynamic;      //动态方法声明
    procedure IMethodVirtualAbstract;virtual;abstract; //虚抽象方法声明
end;
```

## 1. 静态方法

静态方法是方法的默认类型,对它就像对通常的过程和函数那样调用。例如:

```
TSomeClass = class
    procedure FMethodStatic;                //静态方法声明
end;
```

当调用一个静态方法时,类或对象所声明的类型决定了哪种实现(方法体)被执行。也就是说,调用哪种方法实现是在编译时决定的,编译器知道这些方法的地址,所以调用一个静态方法时它能把运行信息静态地链接进可执行文件。静态方法执行的速度最快,但它们却不能被覆盖来支持多态性。

```
//类的声明
type
    TPhone = class                        //定义父类 TPhone
        procedure Talk;
    end;
    TMobilePhone = class(TPhone)         //定义子类 TMobilePhone
        procedure Talk;
    end;
...
```

按照上面的声明,下面的代码演示了静态方法的执行结果。在第二个 Phone.Talk 中,变量 Phone 引用的是一个 TMobilePhone 类型的对象,但却执行 TPhone 中的 Talk 方法,因为变量 Phone 声明的类型是 TPhone。

```
//类在程序中使用
var
    Phone: TPhone;
    MobilePhone: TMobilePhone;
begin
    ...

    Phone := TPhone.Create;
    Phone.Talk;                        //调用 TPhone.Talk
    Phone.Destroy;                    //将 Phone 对象析构
    Phone := TMobilePhone.Create;
    Phone.Talk;                        //调用 TPhone.Talk,因为 Phone 声明为 TPhone 类型
    TMobilePhone(Phone).Talk;         //调用 TMobilePhone.Talk,被强制转为 TMobilePhone 类型
    Phone.Destroy;
    MobilePhone := TMobilePhone.Create;
    MobilePhone.Talk;                  //调用 TMobilePhone.Talk
    MobilePhone.Destroy;
    ...
end;
```

## 2. 虚方法和动态方法

要实现虚方法或动态方法,在方法声明时要包含 virtual 或 dynamic 限定符。不同于静态方法,虚方法和动态方法能在派生类中被覆盖。当调用一个被覆盖的方法时,类或对象的

实际类型决定了哪种实现被调用,而不是它们被声明的类型。这一切都动态地发生在运行时,而不是在程序编译连接时。这意味着使用虚方法或动态方法在编程实现上有着极大的灵活性。

例如:

```
TSomeClass = class
    procedure IMethodVirtual;virtual;           //虚方法声明
    procedure IMethodDynamic;dynamic;         //动态方法声明
end;
```

虚方法和静态方法的调用方式相同。由于虚方法能被覆盖,在代码中调用一个指定的虚方法时编译器并不知道它的地址,因此编译器通过建立虚方法表(VMT)来查找在运行时的函数地址。所有的虚方法在运行时通过 VMT 来调度,一个对象的 VMT 表中除了自己定义的虚方法外,还有其祖先的所有虚方法,因此虚方法比动态方法用的内存要多,但它执行得比较快。

动态方法与虚方法基本相似,只是它们的调度系统不同。编译器为每一个动态方法指定一个独一无二的编号,用这个编号和动态方法的地址构造一个动态方法表(DMT)。与 VMT 表不同,在 DMT 表中仅有它声明的动态方法,并且这个方法需要祖先的 DMT 表来访问它其余的动态方法。正因为这样,动态方法比虚方法用的内存要少,但执行起来较慢,因为有可能要到祖先对象的 DMT 中查找动态方法。

要覆盖一个方法,使用 `override` 限定符重新声明它即可。声明被覆盖的方法时,它的参数类型和顺序以及返回值(如果有的话)必须和祖先类相同。

在下面的例子中,TPhone 中声明的 Talk 虚方法在它的两个派生类 TMobilePhone 与 TTelePhone 中就被覆盖了。

```
type
    TPhone = class                               //定义父类 TPhone
        procedure Talk; virtual;                 //虚方法声明
    end;
    TMobilePhone = class(TPhone)                 //定义子类 TMobilePhone
        procedure Talk; override;               //覆盖声明
    end;
    TTelePhone = class(TPhone)                   //定义子类 TTelePhone
        procedure Talk; override;               //覆盖声明
    end;
...

```

给定上面的声明,下面的代码演示了虚方法在调用时的结果。

```
var
    Phone: TPhone;                               //声明成 TPhone 类类型
begin
    ...
    Phone := TMobilePhone.Create;
    Phone.Talk;                                  //调用 TMobilePhone.Talk
    Phone.Destroy;                               //将 Phone 对象析构
    Phone := TTelePhone.Create;
end;
```

```

Phone.Talk;                                //调用 TTelePhone.Talk
Phone.Destroy;                             //将 Phone 对象析构
...
end;

```

上面这个程序段在计算机中运行时,对于执行 Talk 方法的变量 Phone 而言,Phone 的实际类型是变化的。

### 3. 抽象方法

抽象方法是一种在声明它的类中暂时没有实现,而由它的派生类来实现的虚方法或动态方法。声明抽象方法时,必须在 virtual 或 dynamic 后面使用限定符。

例如:

```

TSomeClass = class
    procedure IMethodVirtualAbstract1;virtual;abstract;    //虚抽象方法声明
    procedure IMethodVirtualAbstract2; dynamic;abstract;    //动态抽象方法声明
end;

```

只有当抽象方法在一个类中被覆盖时,才能使用这个类或它的实例进行调用。如果创建基类的一个实例并强行调用它的一个抽象方法,Delphi 会产生一个运行错误。

如果派生类没有实现(或者不需要实现)一个基类的抽象方法,则可以在类定义中忽略这个方法,或者用 override 和 abstract 限定符(必须是这个顺序)来声明该方法。后一种方案可以表明程序员的意图(表明该方法作为接口,在这个派生类中仍然是没实现的抽象方法),可避免错误调用没有实现的抽象方法。

## 4.3 类的继承、封装和多态

### 4.3.1 类的继承性

类的继承性是指当一个类派生出自己的子类时,子类将具有和父类相同的特征,即在派生类中包含父类所有的字段、属性和方法。例如:

(1) 定义一个 THuman 父类。

```

type
    THuman = class
        Name : String;                //姓名
        Age : Integer;                //年龄
        bSex : Boolean;               //性别
        Hometown : String;            //家乡
        Procedure GetID;               //获取特征号
    end;

```

(2) 定义一个 TStudent 类,该类的基类为 THuman。

```

TStudent = class(THuman)
    StudentID : String;               //学号
    Class : Integer;                  //班级
End;

```

本例中子类 TStudent 包含其父类 THuman 所有的数据成员和方法,并且还增加了自己的新数据成员。

继承节省了在定义新类中的大量工作,因为编程者可以方便地重用代码。但一个派生类不必非得使用继承下来的属性和方法。一个派生类可以选择覆盖已有的属性和方法,或添加新的属性和方法。用户也可添加其他的属性和方法来满足具体的需要。

只有当用户想向自己新类的定义中添加新的操作,或者把已存在类的默认行为融合进自己的新类中时,才需要继承一个已存在的类定义。在 Delphi 中,这称为派生一个类。要添加新的方法和属性,只需要定义它们即可。要覆盖一个已存在的方法,就要在派生类中定义一个新方法,该方法与基类中覆盖的方法有相同的名字和参数列表。

如果派生类没有覆盖方法,那么基类中的方法就可以使用。如 TStudent 中就可以使用其基类中的 GetID 方法。

### 4.3.2 类的封装性

在封装出现以前,面向过程的程序中,其代码的任何部分都能存取其他任何部分的数据,这使得程序维护管理变得十分困难,因为程序员永远不知道这种数据修改变动会对什么过程产生什么样的影响。当发现一个变量拥有一个糟糕的值时,用户永远也不知道如何变成这样的情况。封装简化了编程和维护,因为程序改动的副作用往往被局限于程序中一个小的区域内,这极大地缩小了潜在问题的影响范围。所以,封装性的管理数据的做法给程序带来了极大的可维护性与条理性。

把数据和方法放在同一个数据结构中,如图 4-2 所示,这就是类的封装性。封装是一个面向对象的术语。它的意思很简单,就是把东西包装起来。换言之,属性定义和方法都包装于类定义之中,类定义可以看成是封装构成类的属性和方法。

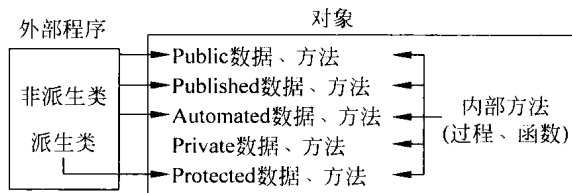


图 4-2 类的封装性示意图

可以把类定义看做是建立在组合不同的数据及其行为之上的数据结构,因为它封装的不仅是数据定义,而且是操纵这些数据的方法。通过限定类成员的可见性,可以使得类成员中的某些属性和方法能够不被程序的其他部分看见,它们“隐藏”了起来,只能在所定义的类中使用。

封装的根本目的是保证对象的属性只能通过对象的方法进行存取。这种实现需要另外的编码,但它保证了任何使用该对象的编码都独立于该对象执行的实现细节。这使得用户可以按个人意愿改变对象的执行过程。牢记这样一点:只要对象的程序设计接口不变,也就是对象方法的结构不变,那么任何使用该对象的代码都能像以前一样正常工作。

### 4.3.3 类的多态性

同样的方法名因为调用的方式不同而执行完全不同例程的能力带来了面向对象编程中另一个强大的方面——多态性。

类的多态性含义为：虚拟方法和动态方法在调用时，程序不根据调用方法的对象的声明类型来决定调用哪个方法，而是在运行阶段根据所调用方法的对象实例来确定调用哪一个方法，因此使虚拟方法和动态方法的调用出现多态性。

例如，某程序声明部分定义如下三个类：

```
type
    TPicture = class                                //基类
        procedure Paint(X, Y : Integer); virtual;    //定义基类动态方法
    end;
    TBmpPic = class(TPicture)                       //派生类定义
        procedure Paint(X, Y : Integer); override;   //定义可覆盖的方法
    end;
    TJpgPic = class(TPicture)
        procedure Paint(X, Y : Integer); override;   //定义可覆盖的方法
    end;
```

给定上面的声明，下面的代码演示了虚方法被调用时的结果。在运行时，执行方法的变量的实际类型是变化的。

```
var
    ...                                              //其他
    FMyPicture :TPicture;
    x, y : integer;
begin
    x := 50;
    y := 50;
    FMyPicture := TBmpPic.Create;                  //创建 TBmpPic 对象
    FMyPicture.Paint(x, y);                        //调用 TBmpPic.Paint(x, y), 体现多态性
    FMyPicture.Free;                               //对象释放
    FMyPicture := TJpgPic.Create;
    FMyPicture.Paint(x, y);                        //调用 TJpgPic.Paint(x, y), 体现多态性
    FMyPicture.Free;                               //对象释放
    ...                                              //其他
end;
```

本例中，虽然一开始声明的 FMyPicture 为 TPicture 类类型，但是在运行时可以创建成不同的它的子类对象，它的方法在运行时按照它的实际类型而调用不同的方法。

顺便说一下，方法重载 (overload) 也可以提供类似于多态性的好处。方法重载意味着两个方法有相同的名字仅参数不同。方法的名字与它的参数的个数及类型决定了该方法的特征。一个类可以拥有多个同名的方法，只要每个方法都有不同的特征即可。

但是重载和多态的机制有本质的区别。重载不同于覆盖，准确地讲，它不是面向对象专有的。面向过程的程序设计中也可以有过程重载，但它不能实现多态。

**例 4.2** 类的多态性示例：编写一个程序，动态生成不同国家(以两个国家为例)的人，



输出其不同语言的见面问候语。

分析：无论什么国家的人，它都有一些共同的特征，可以设计一个 TPerson 类来描述，然后在此基础上派生两个类，例如 TAmerican 与 TChinese。

编程步骤如下：

(1) 建立新的应用程序。

在 Delphi 2007 集成开发环境中通过选择 File→New→Other 命令，在对话框中选择 Console Application，创建一个新的控制台应用程序。然后选择 File→Save 命令保存应用程序。

(2) 建立新的单元文件。

选择 File→New→Unit—Delphi for Win32 命令，创建一个新的单元文件 Unit1，这个单元文件被系统自动引用到控制台应用程序中。

(3) 在 Unit1 中编写类代码。

编写的代码如下：

```
unit Unit1;
interface                                     //接口部分
type
    TPerson = class                           //基类
    public
        procedure SayHello; virtual; abstract; //定义基类虚抽象方法
    end;
    TAmerican = class(TPerson)                //派生类定义
        procedure SayHello; override;         //定义可覆盖的方法
    end;
    TChinese = class(TPerson)
        procedure SayHello; override;         //定义可覆盖的方法
    end;
implementation                               //实现部分
    procedure TAmerican.SayHello;
    begin
        Write(self.ClassName, '问候语: '); //输出类的名称
        Writeln('Hello, Everyone! ');
    end;
    procedure TChinese.SayHello;
    begin
        Write(self.ClassName, '问候语: '); //输出类的名称
        Writeln('大家好! ');
    end;
end.
```

(4) 在控制台程序中编写对象多态性的行为代码。

```
program Project1;
{ $ APPTYPE CONSOLE}                          //编译预处理指令,表示控制台程序
uses                                           //引用其他单元的定义
    SysUtils,                                //系统单元
    Unit1 in 'Unit1.pas';                    //用户自定义的单元
var aPerson: TPerson;                         //声明一个人的对象
```

```

begin
    aPerson := TAmerican.Create;           //创建 TAmerican 对象
    aPerson.SayHello;                      //输出问候用语,体现多态性
    aPerson.Free;                          //释放对象
    aPerson := TChinese.Create;            //创建 TChinese 对象
    aPerson.SayHello;                      //输出问候用语,体现多态性
    aPerson.Free;                          //释放对象
    readln;
end.

```

(5) 保存运行。

运行结果如图 4-3 所示。

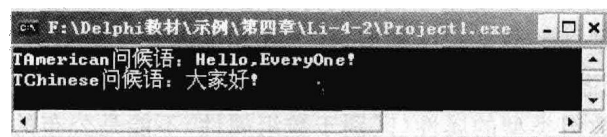


图 4-3 运行结果

在声明方法时,如果它和继承的方法具有相同的名称,但它不包含 override 限定符,则新方法仅仅是隐匿了继承下来的方法,并没有覆盖它。这两个方法在派生类中不存在,方法名是静态绑定的。

#### 4.3.4 类的可见性

类成员包括数据成员和方法,类的每个成员都有一个称为可见性的属性,用来保护类成员。Delphi 中类成员的可见性有 4 种: public(公有的)、protected(保护的)、private(私有的)和 published(公布的)。加上这些限定符,类的定义语法为:

```

type
    类名 = class(基类)
    private
        {类成员定义;}
    protected
        {类成员定义;}
    public
        {类成员定义;}
    published
        {类成员定义;}
end;

```

在默认情况下,类成员的可见性是 public; 当在 {\$M+} 状态下编译类时,可见性是 published。为了使程序具有良好易读的风格,建议最好在声明类时用可见性来组织类成员。

public 成员是完全可访问的成员,可见性最大。该成员访问方便,不受限制。默认情况下,类成员可见性是 public。published 与 public 具有相同的可见性。不同的是,published 成员会产生 RTTI(运行时刻类型)。

private 成员只能在声明它的类中被访问,它的派生类和实例都无法访问。Delphi 的 private 并不是严格意义上的私有,在同一个单元文件中,类的私有成员是可以被其他类访问的。通过私有成员的限制,可以更好地封装和保护自己的类。

protected 成员在声明它的单元文件中是随时可用的,并且在它的派生类中也是可用的。在派生类的所有方法定义中,既可以调用 protected 方法,也能读取或写入 protected 数据成员或属性。

在继承中,可以在派生类中通过重新声明来扩大一个类成员的可见性,但不能降低它的可见性。如一个 protected 属性在派生类中不能改为 private,但是可以被改为 public。另外,published 成员不能在派生类中改为 public。

可公布属性的数据类型是受限制的。有序类型、字符串、类、接口和方法的指针可以被公布;当集合类型的基础类型是有序类型,且上界和下界介于 0~31 之间时,集合类型也是可以公布的。也就是说,集合必须符合 Byte、Word 或 DWord。除了 Real48 外,任何实数类型都是可以公布的;数组类型的属性(区别于数组属性)不能是公布的。

所有的方法都可以是公布的,但一个类不能使用相同的名字公布两个或两个以上被重载的方法。只有当数据成员属于类或接口类型时,它才是可以公布的。

## 4.4 异常处理

程序运行过程中,不可避免地会出现异常或错误,平时所使用的操作系统也经常会出现各种问题,因此在程序的开发中如何检测和处理程序的运行错误是一个关键问题。发现错误有多种方法,主要是通过广泛地测试。但软件工程的有关理论表明,测试一般不可能发现所有的错误。何况测试是在特定软硬件环境下进行的,因而会受到许多限制。

异常处理是对可能出现的错误进行预防,一旦发生即可迅速捕获,根据预先制定的方案对现场作处理,尽可能避免损失并及时准确地将发生的情况向用户报告或记录到日志中去。Delphi 的异常处理机制可以捕获这些异常并进行处理。Delphi 全面支持异常处理,定义了大量的异常处理对象,使程序能够处理几乎所有的异常情况。

### 4.4.1 异常分类

异常处理机制建立在保护块的基础上,保护块是指介于关键字 try 和 end 之间的一段代码,当保护块中的代码发生异常时自动创建一个相应的异常类,程序可以捕获并处理这个异常,以确保程序的正常结束以及资源的释放,若无法处理则会弹出一个消息框。

异常类是 Delphi 异常处理机制的核心,Delphi 提供的所有异常类都从 Exception 类继承,Exception 类包含在 SysUtils 单元中。

#### 1. 运行期间库异常类

运行期间库异常类用于处理运行期间的一些异常,运行期间库异常可以分为 7 类,它们都定义在 SysUtils 单元中。

(1) I/O 异常。I/O 异常 EInOutError 是在程序运行中试图对文件或外部设备进行操作失败后产生的,它从 Exception 派生后增加了一个公有数据成员 ErrorCode,用于保存所发生错误的代码。在发生异常时,通过访问该属性可以针对不同情况采取不同的对策。

通过编译开关{\$I}可以指定是否使用 I/O 异常类。当编译开关为{\$I}时,将在程序中使用 I/O 异常类。当编译开关为{\$I-}时,不产生 I/O 异常类,而是直接把错误代码返回到预定义变量 IOResult 中。

(2) 堆异常。堆异常是在动态内存分配中产生的,包括两个类 EOutOfMemory 和 EInvalidPointer。

(3) 整数异常。整数异常都是从 EIntError 类派生的,但在程序运行中引发的总是它的子类: EDivByZero、ERangeError 和 EIntOverflow。

(4) 浮点异常。浮点异常都是从 EMathError 类派生,与整数异常相似,在程序运行中总是引发它的子类: EInvalidOp、EZeroDivide、EOverFlow 和 EUnderFlow。浮点异常是在进行实数操作时产生的。

(5) 类型匹配异常。当试图用 As 操作符把一个对象强制匹配为另一类对象失败后,引发类型匹配异常 EInvalidCast。

(6) 类型转换异常。当试图从一种数据类型转换为另一种数据类型时,如果不能直接转换则引发类型转换异常 EConvertError。

(7) 硬件异常。硬件异常在以下两种情况下发生:

- ① 处理器检测到一个它不能处理的错误;
- ② 程序产生一个中断企图中止程序的执行。

## 2. 组件异常类

在操作和使用组件的过程中,不可避免地会遇到组件异常。组件异常分为预定义异常和通用异常。Delphi 为某些特定的组件预定义了一些常见的异常情况,这类异常被称为预定义异常。所有组件都可能出现的异常被称为通用异常。

(1) 预定义异常。Delphi 为许多组件都定义了异常类,下面介绍几个典型的预定义异常:

- EMenuError
- EInvalidGridOperation
- EDatabaseError 和 EReportError

(2) 通用异常。Delphi 定义了许多通用异常,下面介绍几个常用的通用异常:

- EInvalidOperation
- EComponentError
- EOutOfResource

## 3. 其他异常

其他异常是指除了组件类以外的其他类引发的异常,包括流异常、图形异常、打印异常和字符串链表异常等。

(1) 流异常。流异常包括 EStreamError、EFCREATEError、EFOpenError、EFileError、EReadError、EWriteError 和 EClassNotFound,在 Classes 库单元中定义。

(2) 图形异常。包括 EInvalidGraphic 和 EInvalidGraphicOperation 两类,在 Graphic 库单元中定义。

(3) 打印异常。当打印发生错误时(如向一个不存在的打印机发送打印任务)引发,它在 Printers 库单元中定义。

(4) 字符串链表异常。包括 EStringListError 和 EListError 两类,它们在用户对字符串链表进行非法操作时引发。由于许多控件(如 ListBox、Memo)都有一个 TStrings 类的重要属性,因此字符串链表异常在编程中非常有用。

#### 4.4.2 异常保护和处理机制

通常为容易出现错误的代码添加保护,将代码块放入 try 语句之后实施保护,异常处理通常有两种结构: try...except...end 和 try...finally...end。前者为开发者提供了一个按自己的需要进行异常处理的机制;后者与资源有关,确保异常发生后资源能够正常释放。

##### 1. 使用 try...except...end 处理异常

```
try
    {要进行保护的代码,这些代码很容易出现异常}
except
    on exception1 do
        {处理 exception1 的语句}
    on exception2 do
        {处理 exception2 的语句}
    ...
    [else {提供默认响应}]           //else 语句放在[ ]中表明 else 语句可有可无
end;
```

try 和 except 之间的语句是希望正常执行的代码,在执行过程中如果出现了异常,则程序跳入 except 部分执行。值得注意的是:正常情况下,except 之后的语句是不执行的,只有在出现异常的情况下才会执行。

##### 2. 使用 try...finally...end 处理异常

try...finally...end 主要用于在发生异常的情况下,及时释放程序所占用的资源,如内存、文件等。因为整个系统的资源总是有限的,所示程序必须在运行结束后及时释放所占用的资源才能保证系统的正常运行。try...finally...end 的结构语法如下:

```
try
    {要进行保护的代码,这些代码很容易出现异常}
finally
    {不管是否发生异常都必须执行的代码,一般用来释放资源}
end;
```

#### 例 4.3 类的异常示例程序。

(1) 编写一个有错误的控制台程序,其代码如下:

```
program Project1;
{ $ APPTYPE CONSOLE}
uses SysUtils;
var j,k:integer;
begin
    k := 0;
    j := k div k;
    writeln('j = ',j);
```

```
    readln;  
end.
```

该程序运行时,发生的异常信息窗口如图 4-4 所示。

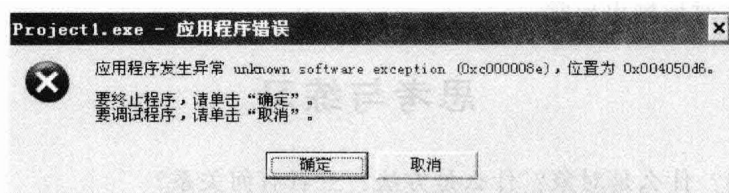


图 4-4 异常错误提示信息窗口

(2) 使用 try...except...end 处理异常程序,其代码如下:

```
program Project1;  
{ $ APPTYPE CONSOLE}  
uses SysUtils;  
var j,k:integer;  
begin  
  try  
  
    k := 0;  
    j := k div k;  
    writeln('j = ', j);  
  except  
    on E:Exception do  
      Writeln(E.Classname, ': ', E.Message);    //捕捉异常  
    end;  
  readln;  
end.
```

该程序运行时,程序能够继续运行下去,同时可以显示捕捉异常的信息,如图 4-5 所示。

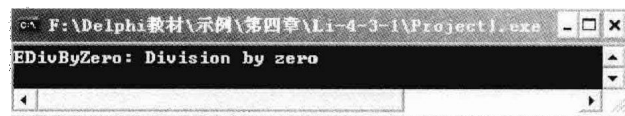


图 4-5 显示捕捉异常的信息

开发健壮的程序是开发人员所追求的目标,这就需考虑到多种可能引发异常的情况,适时地使用 try...except...end 和 try...finally...end 两种结构代码可以使用 Delphi 完善的异常处理机制提供的默认处理方法。

## 本章小结

在这一章中主要要求学生了解的要点有面向对象的基本概念、面向对象程序的特点以及 Delphi 中的异常种类与定义。重点是类的定义与使用、自定义异常的使用方法。难点是

方法的绑定以及类的继承、封装和多态的理解使用。

通过本章的学习,读者应该可以掌握进行面向对象编程基本概念与基本方法。面向对象技术为开发复杂的程序提供了重要的手段。同时开发大型程序时需要考虑到多种可能引发异常的情况,适时地抛出异常。

## 思考与练习

1. 什么是类? 什么是对象? 什么是方法? 三者有何关系?
2. 构造函数和析构函数的功能分别是什么?
3. 简述类的继承、封装和多态含义。
4. Delphi 提供了几种异常类?
5. Delphi 提供了哪两种常用的异常处理结构?

## 第 5 章

## Windows 窗体和常用组件

在 Windows 操作系统中,人机交互的界面主要是通过一些窗口和对话框实现的。在 Delphi 中,这些窗口和对话框就是程序设计阶段的窗体,Delphi 的可视化设计工作就是在窗体中进行的。使用 Delphi 开发应用程序时,广泛地使用到组件。可视组件库 (Visual Component Library, VCL) 是 Delphi 中最重要的部分,通过 VCL 可以在窗体中方便地构建与用户交互的界面,其中大部分组件 (又称为控件) 显示在组件面板 (Components Palette) 上。

本章主要包括以下内容:

- Delphi 程序与窗体;
- 常用 Windows 组件概述;
- 文本显示输入与按钮类组件;
- 列表类与滚动条组件及时钟组件;
- 组件排列布局。

### 5.1 Delphi 程序与窗体

Windows 应用程序是以窗口为基础的,Delphi 的窗体 (Form) 其实也就是一个窗口,但它同时又是应用程序中的一个对象。当在 Delphi 启动并新建一个 VCL 项目 (VCL Forms Application—Delphi for Win32) 后,系统会创建一个空白窗体和一个 TForm1 类,窗体对应的代码单元文件为 Unit1.pas。该单元的代码定义了一个派生于 TForm 的 TForm1 类 (TForm 是 Delphi 在 VCL 中定义的窗体的基类)。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,同时选取组件面板中 Standard 组件页中的 TLabel 与 TButton 组件放置在窗体上,其设计界面如图 5-1 所示。

TForm1 是 TForm 的一个派生类,它具有继承自 TForm 的全部特性,其中包括如何创建自身及响应各种外部输入等。用户在设计阶段通过 IDE 定制该派生类,程序运行时看到的窗口即是由系统为 TForm1 类生成的实例。

TForm 类是一种容器对象,即它的内部可以根据需要包含一些其他对象 (如图 5-1 中的 Label 和 Button)。窗体设计时所做的大部分工作,往往就是将组件面板上的各种组件放入窗体,并设置这些组件的属性和为它们的事件编写代码。由于 Delphi 的 IDE 让用户以可视化方式工作,因此除了在处理事件时需编写少量代码外,大部分工作是在窗体设计区和对象监视器内以“所见即所得”的方式进行设计的。



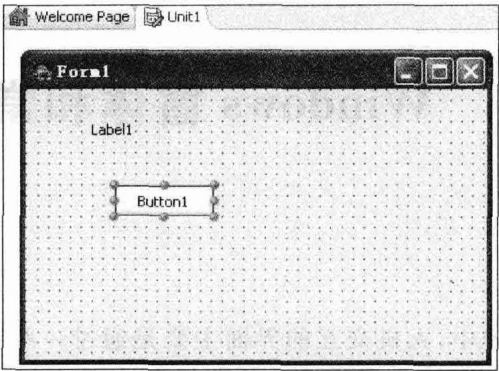


图 5-1 VCL 窗体设计

### 5.1.1 窗体的属性

#### 1. Name 属性

窗体的 Name 属性就是由窗体类生成窗体实例对象的名字,用于在程序中识别不同的窗体对象。Delphi 在创建一个新程序时将主窗体的 Name 默认定义为 Form1。

如果读者不喜欢 Delphi 默认给出的毫无个性的窗体名称 Form1、Form2 等,那么可以为它们改名字,该操作可通过在 Object Inspector(对象查看器)中修改 Name 属性完成。但要注意,窗体(其他对象也一样)改名不能和其他已存在的对象重名并且其名称必须遵循标识符命名规则,否则造成错误。此外应注意,虽然在 Delphi 默认定义下主窗体的 Name 和 Caption 属性都是 Form1,但实际上两者是可以不同的,它们的含义也不同。例如,将 Form1 的默认标题更改为“欢迎进入精彩的 Delphi 2007 世界!”,如图 5-2 所示。



图 5-2 通过 Object Inspector 查看修改对象属性

值得注意的是,在 Delphi 中,一般采用动态方法为 VCL 对象分配内存,Form1 只有被应用程序创建后才能使用。使用窗体对象的属性或方法时,在单元代码编辑的时候必须使用间接成员符“.”。可以通过代码根据需要随时更改对象的属性数值。代码编辑如图 5-3 所示。

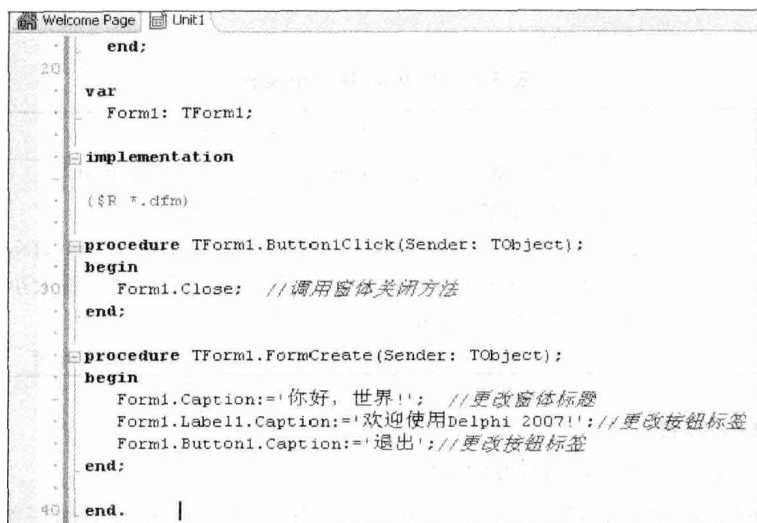


图 5-3 窗体代码编辑

## 2. Caption 属性

Caption 属性用于显示窗体的标题,该标题位于窗体的上边框,应用程序运行时用户可以通过 Caption 来识别不同的窗体,明白窗体设计的含义和功能。如果 Caption 文本超过了窗体标题栏所能容纳的最大长度,则多余部分被截取,代之以“...”,表示标题内容没有被完全显示。在 Object Inspector 中修改的 Caption 属性立即显示在窗体标题栏上,如图 5-2 所示。

## 3. Color 和 Font 属性

Color 属性设置窗体的背景色,设置时可在 Color 名称右边的组合框中选取一种颜色。这些颜色都用颜色常量名称表示。很多对象都有此属性,用户可以根据自己的喜好更改对象的个性颜色。

Font 属性设置窗体使用的字体,窗体内控件默认使用的也是该字体。在 Object Inspector 中设置 Font 时可单击属性位置右端的“...”按钮打开 Font 对话框进行对字体名称、字号、字形、颜色和效果等特性的设置。

## 4. Left、Top、Width、Height、ClientWidth、ClientHeight 和 Position 属性

Left、Top、Width、Height、ClientWidth、ClientHeight 和 Position 是一组同窗体的位置及大小有关的属性值。要确定窗口的位置及大小,只要指定窗口的左上角坐标(Left,Top)和窗体的宽度(Width)、高度(Height)即可。但是,有一点要注意,有时设置的这些属性值并没有发挥它应有的作用,这是因为这些属性本身要受到 Position 属性的制约。Position 属性的说明如表 5-1 所示。

有时,窗口的大小并不是我们最关心的,反而窗口内部的大小才是真正重要的。例如,要在窗口内显示图形、图像,而窗口的大小必须依据图形或图像的大小作调整。这时,我们关心的不再是 Width 和 Height 了,而是 ClientWidth 和 ClientHeight。Client 区域即用户区(又称客户区域),是指窗口除去标题和外框以后的内部区域。调整 ClientWidth 与 ClientHeight 属性时,窗口的 Width 和 Height 属性也会自动调整。

表 5-1 Position 属性值及含义

取 值	含 义
PoDesigned	以(Lef、Top、Width、Height)指定的窗口绝对坐标出现
PoDefault	窗口出现的位置及其大小由 Delphi 自己决定
PoDefaultPosOnly	窗口出现的位置由 Delphi 自己决定,窗口大小由 Width、Height 决定
PoDefaultSizeOnly	窗口出现的大小由 Delphi 自己决定,Delphi 会自动调整大小使右边及底边与屏幕切平,窗口位置由(Left,Top)决定
PoScreenCenter	窗口出现在屏幕正中央的位置,大小由 Width、Height 决定

5. FormStyle 属性

FormStyle 属性用来指定窗体的类型。

从窗体类型的角度来看,Windows 环境中的应用程序可以分为下面三类:

(1) 多文档界面(MDI)应用程序。一般这种应用程序具有一个父级窗口和多个子窗口,可以同时打开多个文档,分别在多个子窗口中显示。例如,常用的字处理软件 Word 等,可以同时编辑多个文档。

(2) 单文档界面(SDI)应用程序。这种应用程序同时只能打开一个文档。例如,Windows 系统附件中自带的“记事本”程序,只能同时编辑一个文本文件。

(3) 对话框应用程序。这种应用程序的主界面基于一个对话框类型的窗体。例如,Windows 系统中自带的“扫雷”游戏程序。

此外,有些应用程序在运行期间可以总是显示在桌面的最前端,例如 Windows 操作系统中的“程序”→“附件”→“系统工具”→“系统监视器”。如果在菜单上选择“查看”→“前端显示”命令,则系统监视器会一直显示在其他应用程序的窗口之上。

属性 FormStyle 可以分别实现上面所说的各种类型的应用程序,取值如表 5-2 所示。

表 5-2 FormStyle 属性值及含义

取 值	含 义
fsNormal	普通类型的窗体。既不为 MDI 应用程序的父级窗口,也不为 MDI 应用程序的子窗口
fsMDIChildMDI	应用程序中的子窗体
fsMDIFormMDI	应用程序中的父窗体
fsStayOnTop	在桌面最前端显示的窗体

通常不要在程序运行期间改变窗体的类型。

6. BorderIcons 属性

BoderIcons 属性是集合型的属性,用来指定窗体标题栏上的图标,可以自由地搭配组合该集合中的元素以控制窗口所能提供的系统功能,有关说明如表 5-3 所示。

7. BorderStyle 属性

BorderStyle 属性用来设置窗体的外观和边框,也是集合型的属性,主要描述窗体边界风格。可以指定为表 5-4 中的取值。

表 5-3 BorderIcons 属性值及含义

取 值	含 义
biSystemMenu	可以通过单击标题栏左边的图标或在标题栏上单击鼠标右键来显示控制菜单。控制菜单有时也称为系统菜单
biMinimize	在标题栏右边显示最小化按钮
biMaximize	在标题栏右边显示最大化按钮
biHelp	在标题栏右边显示帮助按钮。只有窗体的 BorderStyle 属性设置为 bsDialog 或者窗体属性 BorderIcons 中不包括 biMinimize 和 biMaximize 时,biHelp 设置才有效

表 5-4 BorderStyle 属性值及含义

取 值	含 义
bsDialog	窗体为标准的对话框,边框大小不可以改变
bsSingle	窗体具有单线边框,大小不可以改变
bsNone	窗体没有边框,也没有标题栏,边界的大小不可以改变
bsSizeable	边框大小可变的标准窗体
bsToolWindow	风格与 bsSingle 相同,只是标题栏比较小
bsSizeToolWin	风格与 bsSizeable 相同,只是标题栏比较小

## 8. Hint 和 ShowHint 属性

Hint 和 ShowHint 这两个属性提供提示信息。Hint 属性是 AnsiString 类型的字符串。ShowHint 属性值为 true 时,程序运行期间,当鼠标在窗体上停留的时间超过 1 秒钟时,在鼠标位置处就会出现一个矩形小框,其显示内容为 Hint 的属性值;ShowHint 的属性值为 false 时,不显示 Hint 的内容。

## 9. WindowState 属性

WindowState 属性代表窗体的当前窗口状态,它的值可以是 WsMinimized、WsMaximized 或者 WsNormal,分别表示最小化、最大化和正常状态。该属性默认值是 WsNormal,可通过对其写操作改变窗口状态。

## 10. Icon 以及其他属性

Icon 属性用来指定标题栏中显示的图标。单击对象编辑器 Icon 属性右边的省略号按钮,在弹出的 PictureEditor 对话框中单击 Load 按钮,就可以装入一个制作好的图标。

TForm 类还有一些其他很有用的属性,如 Cursor 属性可设置鼠标箭头的图形等。用户可以根据系统帮助获取相关知识。

我们已经学会在窗体的设计期间通过 Object Inspector 改变属性的值。

窗体属性的设置和改变通常有两种方法可采用(其他控件对象同)。一种是在设计时通过 Object Inspector 窗口为其设定各种属性值;另一种是在程序代码中设置或改变属性值。需要特别指出的是,一个对象不是所有的属性都可以在设计时设置,有的属性只能在代码中设置;反之,不是所有的属性都可以在代码中设置,有的属性只能在设计时设置。

但有时在窗体的设计期间,属性值还不能确定,或者有些属性虽然在设计时已被设定了一个值,但在运行期间又要改变该属性的值,这就要求能在程序的运行期间修改属性值。

在程序代码中设置或改变对象属性值使用如下赋值表达式:

<对象名>.<属性名>:= 属性值

**例 5.1** 动态改变属性示例,完整代码请参考配套教学资源。

利用 Caption 属性在程序运行时动态显示鼠标在窗体上位置的 X 坐标。由于在设计时不知道鼠标在窗体上的动态位置,因此只能在窗体的运行期间通过代码来完成 Caption 属性的设置。为此,可以通过响应窗体的 OnMouseMove 事件得到 X 坐标(该事件函数有参数 X 和 Y)并将它传给 Caption,完成后的代码如下所示:

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
    Form1.Caption := '鼠标所在位置: X := ' + IntToStr(x) + ' Y := ' + IntToStr(y);
end;
```

运行效果如图 5-4 所示。

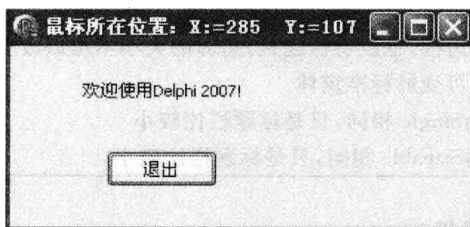


图 5-4 通过代码在运行时动态改变对象属性

本例中 IntToStr 函数的功能是将整型数据转换成字符串。

### 5.1.2 窗体常用的方法

窗体的方法是指窗体可以执行的动作和行为。下面介绍窗体的常用方法以及如何通过代码调用这些方法。

#### 1. Delphi 窗体的常用方法

(1) Close 方法。可调用 Close 方法来关闭一个窗体,关闭应用程序的主窗体通常会结束应用程序。

(2) Hide 和 Show 方法。Hide 方法用来隐藏窗体,被隐藏的窗体虽然看不见,但还在内存中。Show 方法则使原先隐藏的窗体显现。这些方法主要用在有多个窗体的应用程序中。

(3) Print 方法。Print 方法无参数,调用该方法可以在打印机上输出当前窗体的画面。

(4) Update 方法。刷新窗体画面的作用,此方法一般情况下系统自动调用实现重画窗体。

(5) Cascade、Tile 和 ArrangeIcons 方法。这三种方法主要是子窗口管理方法,常在 MDI 系统父窗体中使用。Cascade 功能是窗口平铺,Tile 功能是窗口层叠,ArrangeIcons 功能是排列最小化子窗口标题。

#### 2. Delphi 窗体方法的使用

方法是对象可以执行的动作和行为,在程序代码中,对象调用方法的一般格式为:

<对象名>.<方法名>(<参数 1, 参数 2, ...>)

调用方法时,是否需要参数须根据是何方法以及具体的使用情况而定。调用方法必须遵循系统过程或者函数的语法规则。

### 例 5.2 窗体方法使用示例:显示或者隐藏窗体。

程序功能:利用定时器,1 秒钟显示或者隐藏窗体一次。

(1) 在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。

(2) 选取组件面板中“System 组件页”中的 TTimer 组件放置在窗体上,TTimer 组件可以周期地运行处理代码,系统默认的间隔周期为 1000ms 即 1s,用户可以更改其 Interval 属性来改变其定时周期,本例中使用默认值。

(3) 双击 TTimer 组件,产生定时器运行定时处理代码。

(4) 编辑定时器控制窗体代码。

设计界面效果如图 5-5 所示。

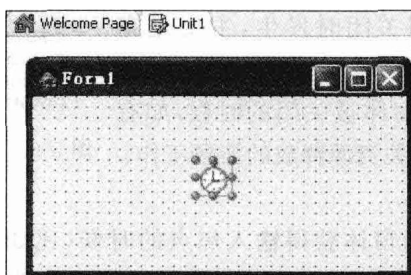


图 5-5 调用 Show 或者 Hide 方法控制窗体

完成后的定时器定时运行代码如下所示:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    if Form1.Visible then           //如果当前窗体处于显示状态
        Form1.Hide                 //调用窗体隐藏方法
    else
        Form1.Show;                //调用窗体显示方法
end;
```

保存工程,然后运行系统,可以观察到每秒钟窗口闪烁一次,即通过定时器周期地自动显示或隐藏。

## 5.1.3 窗体常用事件

事件驱动的程序设计是一种面向用户的程序设计方式。相对于面向过程的程序设计方式来说,事件驱动的程序设计方式是一种被动的程序设计方式。

Windows 是一种多任务的操作系统,可以同时运行多个程序。每一个程序都不能独占系统资源,而是共享各种系统资源,比如 CPU 和存储器等。事件驱动程序恰好适合于 Windows 的多任务特点。事件驱动编程方式完全不同于面向过程的设计方式,它是由事件的产生来驱动的。事件的产生是随机的、不确定的,没有预定顺序的。因此,各种事件可以以各种不同的、合理的顺序出现,那么依赖于事件驱动的程序也可以按各种不同的、合



理的流程来执行。也正是这一点,使得多个程序共享系统资源成为可能。

事件是作用于对象上的一种动作或行为,也可以理解为是对象能够识别(也称响应)的一种操作。Delphi 系统为 VCL 组件对象可以响应哪些事件都预先进行了规定,用户不可以随意增加。通过代码编辑器窗口,读者可以非常方便地了解一种对象可以响应哪些事件。一旦在设计界面上选定特定的对象后,在 Object Inspector 窗口的“事件”栏可以查看该对象可以响应的各种事件,图 5-6 显示了窗体对象可以响应的事件。

### 1. OnCreate、OnClose 和 OnDestroy 事件

OnCreate 事件发生在窗体创建的时候,可以把窗体创建时要完成的一系列代码放在 OnCreate 事件发生的时候执行,OnCreate 事件由 TCustomForm 类的构造函数激发。

OnClose 事件当窗体即将关闭时发生,主要是完成关闭用户清理工作。

OnDestroy 事件则发生在窗体被关闭的时候,如有必要,可在该事件代码中进行有关资源释放的清场工作。

### 2. OnActivate 事件

OnActivate 事件发生在当窗体获得输入焦点的时候(比如用户单击窗体),当焦点位于某一窗体时,该窗体即为活动窗体。一般在窗体创建时或原先隐藏的窗体显示时也会触发该事件。此外,当在一个多窗体程序中的不同窗体间进行切换时也能触发该事件。

### 3. OnResize 事件

OnResize 事件发生在窗体的大小改变时触发。

### 4. OnPaint 事件

OnPaint 事件发生在 Windows 认为窗体需要重画的时候(如另一个窗口从窗体上拖过之后)。窗体在创建时不仅会触发 OnCreate 事件,而且要激发一些其他事件,这些事件按其先后发生顺序通常为 OnCreate、OnShow、OnActivate、OnPaint 和 OnResize。

### 5. OnShow 与 OnHide 事件

OnShow 事件当窗体显示时发生,OnHide 事件当窗体隐藏时发生。

### 6. 窗体鼠标或者按键操作主要事件

- OnClick: 鼠标单击事件,多用在某个对象控制范围内的鼠标单击。
- OnDblClick: 鼠标双击事件。
- OnMouseDown: 鼠标上的键被按下时发生。
- OnMouseUp: 鼠标键按下后,松开时激发的事件。
- OnMouseEnter: 当鼠标移动到某对象范围的上方时触发的事件。
- OnMouseMove: 鼠标移动时触发的事件。
- OnMouseLeave: 当鼠标离开某对象范围时触发的事件。
- OnKeyPress: 当键盘上的某个键被按下并且释放时触发的事件。
- OnKeyDown: 当键盘上某个按键被按下时触发的事件。

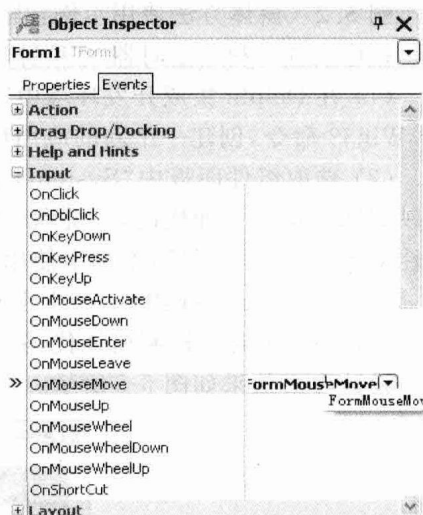


图 5-6 Form 对象可以响应的各种事件

## 7. 窗体的其他主要事件

- OnHelp: 当窗体收到 HELP 请求时发生。
- OnDragDrop: 当一个对象拉进此窗体并丢下时发生。
- OnDragOver: 当一个对象拉进此窗体时发生。
- OnConstrainedDresize: 在 OnCanResize 事件发生后发生。
- OnCanResize: 当企图改变窗体尺寸时确认是否改变时发生。
- OnResize: 当窗体尺寸改变时发生。
- OnStartDock: 当对象开始停泊时发生。
- OnEndDock: 当对象停泊结束时发生。
- OnUndock: 当窗体解除停泊时发生。
- OnDockDrop: 当其他窗体停泊到此组件时发生。
- OnDockOver: 当其他窗体向此组件停泊接近时发生。
- OnCloseQuery: 在窗体即将关闭时确认是否真的关闭窗体时发生。
- OnActivate: 当窗体获得 Focus(焦点)时发生。
- OnDeactivate: 当窗体失去 Focus(焦点)时发生。

**例 5.3** 窗体事件编程使用示例: 创建窗体与单击或者双击的时候, 显示事件发生时间与事件名称于窗体标题之上, 并改变窗体背景颜色表示该事件发生。

首先说一下, Delphi 语言中使用 Now 这个标准库函数可以获取系统当前实际时间; FormatDateTime 函数功能是格式化转换时间的函数, 将时间按一定的格式转换成字符串, 其详细使用方法可查看系统的帮助文件。

(1) 在 Delphi 集成开发环境 (IDE) 中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序。

(2) 在 Object Inspector 窗口的“事件”栏, 找到窗体 OnCreate 事件, 双击编辑如下代码:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    //Self 是隐含默认对象本身, 也可以直接使用 Form1
    Self.Caption := FormatDateTime('创建事件:"yyyy-mm-dd' + ' hh:mm:ss AM/PM', NOW);
    Self.Color := ClRed; //将窗体背景颜色设置成红色
end;
```

(3) 在 Object Inspector 窗口的“事件”栏, 找到窗体 OnClick 事件, 双击编辑如下代码:

```
procedure TForm1.FormClick(Sender: TObject);
begin
    //Self 是隐含默认对象本身, 也可以直接使用 Form1
    Form1.Caption := FormatDateTime('单击事件:"yyyy-mm-dd' + ' hh:mm:ss AM/PM', NOW);
    Self.Color := ClGreen; //将窗体背景颜色设置成绿色
end;
```

(4) 在 Object Inspector 窗口的“事件”栏, 找到窗体 OnDblClick 事件, 双击编辑如下代码:

```
procedure TForm1.FormDblClick(Sender: TObject);
begin
```



```
Form1.Caption := FormatDateTime('"双击事件:"yyyy-mm-dd' + ' hh:mm:ss AM/PM', NOW);
Form1.Color := ClBlue; //将窗体背景颜色设置成蓝色
end;
```

完成各种代码后,保存工程,然后运行系统,一开始从窗体标题上可以看到创建事件的发生和发生时间,窗体颜色为红色;单击窗体客户区,可以看到单击事件的发生和发生时间,窗体颜色为绿色;双击窗体客户区,可以看到双击事件的发生和发生时间,窗体颜色为蓝色。图 5-7 显示了窗体响应双击事件的运行效果。

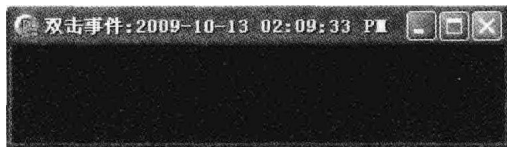


图 5-7 窗体响应鼠标双击事件的运行效果

## 5.2 常用 Windows 组件概述

在 Delphi 编程的过程中,要经常使用到各种组件。Delphi 中有两类组件:可视化组件和非可视化组件。它们都是从 TComponent 类派生而来的。组件是一种对象模板,它可以出现在组件面板上,也可以被拖放到窗体设计器上。



图 5-8 组件面板的 Standard 页  
常用控件

可视化组件是可视的用户界面元素,通过可视化组件可以在窗体中方便地构建与用户交互的界面。可视化组件也称为控件(Control)。控件又有两种不同的类型,即基于窗口的和基于图形的。基于窗口的控件(即窗口控件)是指基于系统窗口的可视组件,具有窗口句柄,可以作为输入焦点,并可以含有其他控件,如 Edit 组件。图形控件没有窗口句柄,不能作为输入焦点,也不能含有其他控件,如 TLabel 组件。

非可视组件没有可视化的特征,可以为程序增加许多强大的功能,如 TTimer 组件用于定时功能。

本节将详细介绍一些常用组件,如标签、编辑框、文本框和命令按钮等的用法,这些控件都位于组件面板的 Standard 选项卡上,如图 5-8 所示。读者应注重掌握各种控件的共性与常用事件方法,并提高通过 Help 系统查阅各个组件用法的能力。

有两种方法可以将组件放到窗体或其他容器对象内:一种方法是首先单击位于组件面板上的某一图标,然后单击一下窗体或容器对象上合适的位置将组件放入;另一种方法是双击组件图标,将一个组件放置于当前窗体或容器对象中。

### 组件的常用属性及事件

在 Delphi 中,每一个组件都具有特定的属性、事件和方法。组件的属性是组件特性的描述,包括组件的外观特性(如位置、尺寸、外形、字体和可视性)和非可视化的特性。组件常

见的基本属性如表 5-5 所示。

表 5-5 组件的常见基本属性

属 性	说 明
Height	高度
Width	宽度
Left	组件在容器内的水平坐标,相对于容器左边
Top	组件在容器内的垂直坐标,相对于容器上边
Align	组件上的对齐方式(居上、居下、居左、居右、居中)
Visible	设置组件是否可见,默认值为可见(值为 true)
Caption	显示组件的标题
Color	组件的背景颜色
Font	设置组件显示文本的字体
Ctl3D	是否以 3D 方式显示组件,默认值为 true
ShowHint	是否显示组件的提示信息,默认值为 true,与 Hint 连用
Hint	当鼠标指针移到组件上时,组件显示的提示信息
Enabled	是否允许用户操作组件,true 表示允许,false 表示不允许
Name	用于标识组件的名称,在程序中通过 Name 可以调用该组件
TabOrder	Tab 次序

组件的事件是对组件所做的某个动作或系统的某些行为(如按下鼠标、双击鼠标和窗体装入等)的反应。每个组件都提供满足用户的各类事件处理器,当发生特殊事件时,应用程序将执行事件处理器中的代码。组件的常用事件及触发条件如表 5-6 所示。组件的方法是指该类或对象类型实例的函数或过程,每一类组件都有自己的方法,以实现各种功能。

表 5-6 组件的常用事件及触发条件

事 件	触 发 条 件
OnClick	当鼠标单击时触发本事件
OnDbClick	当鼠标双击时触发本事件
OnMouseDown	当鼠标左键按下时触发本事件
OnMouseMove	当鼠标移动时触发本事件
OnKeyDown	当按下任意键(包括组合键)时触发本事件
OnKeyPress	当按下任意键(单字符键)时触发本事件
OnKeyUp	当松开已按下键时触发本事件
OnEnter	当获得焦点时触发本事件
OnExit	当失去焦点时触发本事件
OnStartDrag	当开始拖动时触发本事件
OnDragDrop	当组件拖动操作结束时触发本事件

5.3 文本显示输入与按钮类组件

5.3.1 TLabel

TLabel(标签)组件位于组件面板的 Standard 页上(如图 5-8 所示),可以显示一个字符

串。通常利用 TLabel 组件在窗体上显示静态文本,如显示提示信息;也可显示动态文本,如用 TLabel 显示不断变化的当前时间。

TLabel 组件主要用来标识应用程序中的其他对象,最常见的用法是把标签放在其他组件的旁边,如放在文本框、Memo 框及单选按钮等组件的左侧或上方。标签上的文字信息有助于用户操作,也可为用户提供信息。

标签除了一般组件所具有的名称、Enabled、Color 和 Visible 等基本属性外,还经常使用以下这些属性。

(1) Caption 属性:用来设置在标签控件中显示的文本。

(2) Alignment 属性:用来设置标签显示文本的对齐方式,有文本居中(taCenter)文本左对齐(taLeftJustify)和文本右对齐(taRightJustify)三个选项。

(3) AutoSize 属性:在默认情况下,AutoSize 为 true,标签能够自动调整宽度以适应 Caption 的变化。如果将 AutoSize 设置为 false,当 Caption 属性的文本超过控件宽度时,超出部分将被截掉。

(4) wordWrap 属性:若 wordWrap 属性为 true,则当标签内容较长时允许折行显示,但具体显示方式与 AutoSize 以及 Label 的尺寸都有一定相关性。

(5) Align 属性:该属性决定 TLabel 组件的显示位置(使其位于窗口的某个部分或者全屏)。

(6) Layout 属性:该属性设置文字在垂直方向的对齐方式。

(7) TransParent 属性:该属性非常有用,它能够决定标签是否透明显示,主要在图形操作时使用。如果将其设置为 true,则可透过标签看到背景图。

标签控件主要用来显示说明文字,一般较少处理其事件,但有时也可能会处理它的 OnClick(单击)事件。

### 5.3.2 TButton

TButton(命令按钮)组件位于组件面板的 Standard 选项卡上,如图 5-8 所示。TButton 用于为用户提供选择执行的命令,通常称为命令按钮。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,单击后可触发执行特定的操作。

下面是命令按钮的常用属性和事件。

(1) Caption 属性:用于设置显示在按钮上的文本标题。文本标题显示在按钮的中部,如果标题文本超过了命令按钮的宽度,则对称两端的内容将被截取。如果要改变命令按钮上显示的字体,可设置命令按钮的 Font 属性。可以给命令按钮定义快捷键,只需在 Caption 中作为快捷键的字母前添加字符“&”即可。

例如,在窗体中放入两个按钮,将 Button1 按钮的 Caption 属性设置为 &Enter,将 Button2 按钮的 Caption 属性设置为 Le&ave,则 E 和 a 分别成为这两个按钮的快捷键,即在程序运行中,按 Alt+E 组合键与单击 Enter 按钮产生的作用是等效的,而按 Alt+a 组合键则与单击 Leave 按钮产生的作用是等效的。可以看到 Caption 中的“&”字符不会在按钮标题中出现,但被设置成快捷键的字母底下有一下划线,如图 5-9 所示。

(2) Default 属性:若命令按钮的 Default 属性设置为 true,则在程序的运行期间,当该按钮得到输入焦点时,按下 Enter(回车)键将会触发其 OnClick 事件。

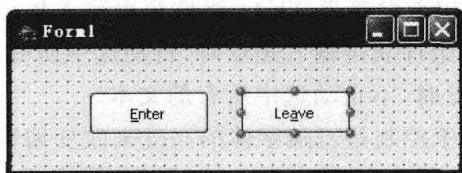


图 5-9 为按钮设置快捷键

(3) Cancel 属性: 若命令按钮的 Cancel 属性设置为 true,则在程序的运行期间,当该控件得到输入焦点时,按下 Esc(取消)键将会触发其 OnClick 事件。

(4) Enabled 属性: 当 Enabled 属性为 true 时,按钮可正常操作;而当其为 false 时,按钮颜色为灰暗色,变为不可操作。

(5) OnClick 事件: 命令按钮最重要的事件就是 OnClick(单击)事件。必须注意,如果双击命令按钮,则其中每次单击都将分别处理,即命令按钮不支持双击事件。单击命令按钮也将引发 MouseDown 和 MouseUp 事件。

### 5.3.3 TEdit

TEdit(编辑框)组件位于组件面板的 Standard 选项卡上,如图 5-8 所示,可以显示、编辑单独的一行文本。TEdit 组件提供了一个编辑区域,用户可在区域中输入单行信息。TEdit 不但允许用户在编辑框中输入数据,也可以用来显示查询的数据及提示信息。

(1) Text 属性: 编辑框没有 Caption 属性,但有 Text 属性。使用 Text 属性可以读取编辑框中的文本或者给编辑框赋值。在默认设置情况下,Text 属性的初始值就是编辑框控件的名称。实际使用中,在设计时通常需要将 Text 清空。

(2) MaxLength 属性: 用来设置最多能输入到编辑框中的字符个数。默认值为 0,表示编辑框输入的字符数不受限制。改变该属性不会对编辑框的当前内容产生影响,只影响以后文本框内容的改变。

(3) PasswordChar 属性: 通常用于设置口令或密码。该属性的默认设置为 #0 字符,表示显示实际的文本。如果将该属性设置为其他的字符,用户在编辑框内输入字符时,显示的不是输入的字符,而是所设置的字符。不过,此时编辑框的 Text 属性仍是用户所输入的内容。

(4) ReadOnly 属性: ReadOnly 属性为 true 时,编辑框为只读。

(5) SelStart、SelLength 和 SelText 属性: 这三个属性在设计时是不可用的,即这些属性只能在程序运行阶段通过程序代码进行设置或访问。

- SelStart: 返回或设置所选择文本的起始点,如果没有文本被选中,则返回插入点的位置。
- SelLength: 返回或设置所选择文本的字符数。
- SelText: 返回或设置所选择文本的内容,可以通过 SelText 属性来读取被选中文本的值,也可以通过设置 SelText 属性用一个新的字符串代替编辑框中原先选中的文本。如果原先没有文本被选中,则设置 SelText 等价于在光标处插入新字符串。

(6) **AutoSelect** 属性: 用来设置当 TEdit 组件获得输入焦点时, 自动选中所有的文本。若它的值为 true, 则当 Edit 组件获得输入焦点时, 自动选中所有的文本; 若它的值为 false, 则当 Edit 组件获得输入焦点时, 不自动选中所有的文本。

(7) **AutoSize** 属性: 用于控制是否随字体的高度来改变编辑框的高度。当 AutoSize 设置为 true 时, 编辑框的高度会自动适应输入文字的字体高度。但要注意, 该属性只有当 BorderStyle 属性设置为 bsSingle 时才有效。

(8) **CharCase** 属性: 用于强制组件中的文本全部为大写(或小写)字母。

(9) **Enabled** 属性: 该属性决定文本框中的内容是否处于激活状态, 其默认值为 true, 即编辑框中的文本可以修改。当它的值为 false 时, 编辑框处于灰色冻结状态, 其中的文本不能修改。

(10) **Hint** 属性: 用来设置所显示的提示信息。例如通过将该属性设置为“请输入借书证号”, 并将 ShowHint 属性设置为 true, 则在应用程序运行时当用户将鼠标移到文本框并停顿片刻后, 将显示引号中的提示信息。

(11) **IME** 属性: 包括 ImeMode 属性和 ImeName 属性, 通过设置 IME(Input Method Editor, 输入法编辑器)属性可以在输入焦点定位在 Edit 组件上时, 自动实现输入法的切换。

- ImeMode 属性: 如果要设置为中文输入法, 可以将 ImeMode 属性设置为 imChinese。

- ImeName 属性: 可以通过下拉组合框选择一个具体的输入法。

(12) **Clear** 方法: 删除编辑框中的所有文本。

(13) **ClearSelect** 方法: 删除编辑框中被选择的文本。如果编辑框中没有被选择的文本, 则不删除任何内容。

(14) **CopyToClipboard** 方法: 将编辑框中已选择的文本复制到剪贴板上, 并取代原来剪贴板中的所有内容。如果用户未在编辑框中选择任何文本, 该方法的执行将不删除原来剪贴板的所有内容。

(15) **CutToClipboard** 方法: 将编辑框中已选择的文本复制到剪贴板上, 并取代原来剪贴板中的所有内容, 然后删除在编辑框中被选中的内容。

(16) **PasteFromClipboard** 方法: 将剪贴板上的内容复制到编辑框中, 并插入到编辑框中光标所在的当前位置。

(17) **SelectAll** 方法: 该方法选择编辑框中的所有文本。

(18) **SetFocus** 方法: 将输入焦点移到编辑框中。

(19) **OnKeyPress** 事件: 当在键盘上按下一个键时触发 OnKeyPress 事件; 除了普通字符键外, 按下的也可以是特殊键或组合键。

(20) **OnChange** 事件: 编辑框的 Text 属性中内容有所改变时触发 OnChange 事件, 如果往编辑框内输入若干字符, 那么每输入一个字符就会触发一次该事件。

(21) **OnEnter** 事件: 当输入焦点刚进入编辑框时触发此事件。

(22) **OnExit** 事件: 当输入焦点离开编辑框时触发此事件。

**例 5.4** 编辑框使用示例: 用编辑框输入内容, 信息可以同时自动输出到另一个标签上。

(1) 在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi

for Win32 命令,创建一个新的应用程序。并从工具栏拖动一个 TLabel、一个 TEdit、两个 TButton 组件到窗体上,同时修改它们的 Caption 等属性。设计好的程序界面如图 5-10 所示。

(2) 选中 Button1(重新输入),在 Object Inspector 窗口的“事件”栏找到 Button1 的 OnClick 事件,双击编辑如下代码;

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    //调用 Clear 方法清空编辑框,调用 SetFocus 设置该对象焦点
    Edit1.Clear;
    Edit1.SetFocus;
end;
```

(3) 选中 Button2(关闭),在 Object Inspector 窗口的“事件”栏找到 Button2 的 OnClick 事件,双击编辑如下代码;

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    //使得窗体关闭,调用窗体关闭方法
    Form1.Close;
end;
```

(4) 选中 Edit1 编辑框,在 Object Inspector 窗口的事件栏找到 Edit1 编辑框 OnChange 事件,双击编辑如下代码;

```
//为 Edit1 的 OnChange 事件添加的代码
procedure TForm1.Edit1Change(Sender: TObject);
begin
    //使得 Edit1 内容的变化输出到标签上
    Label1.Caption := '输出: ' + Edit1.Text;
end;
```

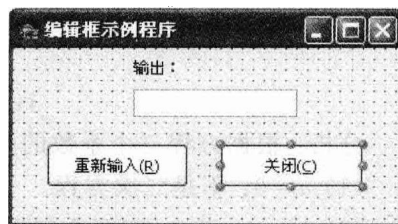


图 5-10 编辑框示例界面设计

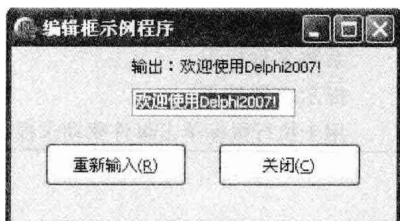


图 5-11 编辑框示例运行效果

完成各种代码后,保存工程,然后运行系统,可以发现只要编辑框内容变化,标签上输出同样内容。单击“重新输入”按钮或者按 Alt+R 组合键,就会清除编辑框内容;单击“关闭”按钮可以退出窗体程序。图 5-11 显示了编辑框操作的运行效果。

**例 5.5 标签、按钮、编辑框综合示例:** 求方程  $ax^2 + bx + c = 0$  的根,其中,  $a$ 、 $b$ 、 $c$  由键盘输入到编辑框中。

**理论准备:** 在数学中,利用求根公式来求解一元二次方程的根是具有普遍性的方法,所以在算法设计中应选用此方法来实现。一元二次方程的求根模型为:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

令

$$p = \frac{-b}{2a}, \quad q = \frac{\sqrt{b^2 - 4ac}}{2a}$$

$$x_1 = p + q, \quad x_2 = p - q$$

分析：根据方程三个系数的不同情况，方程的根有如下几种情况：

- (1)  $a=0$ ：不是二次方程。
- (2)  $b^2 - 4ac \geq 0$ ：有两个实根，求  $x_1$  和  $x_2$ 。
- (3)  $b^2 - 4ac < 0$ ：没有实数解。

算法步骤：

- (1) 输入实型数  $a, b, c$ ，到三个编辑框中（命名为 EditA、EditB 和 EditC）。
- (2) 求判别式 disc。
- (3) 如果  $\text{disc} \geq 0$ ，调用求平方根函数  $\text{sqrt}()$ 。
- (4) 根据公式求方程的两个根  $x_1$  和  $x_2$ 。
- (5) 输出方程的两个根到两个标签中（命名为 Labelx1 和 Labelx2）。

详细实现步骤如下：

- (1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令，创建一个新的应用程序。

- (2) 定制窗体。

添加三个 TEdit 组件作为方程系数输入，更改 name 属性分别为 EditA、EditB 和 EditC；并为三个编辑框配置三个 TLabel 组件作为编辑框的系数名称标识，更改其 Caption 分别为 a, b, c；进而添加两个 TLabel 组件分别作为方程的根显示结果用，更改其 name 属性为 Labelx1 和 Labelx2，并将其 Caption 属性值更改为  $x_1=?$  和  $x_2=?$ ；最后添加一个 TButton 组件，更改其 Caption 为“求解”。

各组件属性设置如表 5-7 所示。

表 5-7 各组件属性设置

组 件 名	属 性	属 性 值	说 明
Form1	Caption	求 $ax^2 + bx + c = 0$ 方程的根	窗体显示标题
EditA、EditB、EditC	Text	'1', '-2', '1'	设置默认的方程系数
Label1、Label2、Label3	Caption	'a', 'b', 'c'	系数名称标识
Labelx1、Labelx2	Text	' $x_1=?$ ', ' $x_2=?$ '	提示求解结果
Button1	Caption	' 求解 '	用于执行鼠标单击事件驱动求根

调整组件位置及大小后，窗体设计界面如图 5-12 所示。

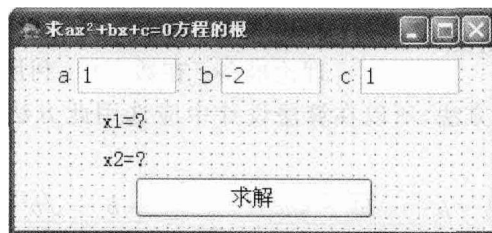


图 5-12 设计窗体界面

### (3) 编写代码。

双击 Button1, 为 Button1 添加 OnClick 事件处理过程代码:

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b, c, x1, x2, disc: real;           //定义对应方程参数的实型变量
begin
    //数据输入: 将编辑框里的数据送到内存变量中
    a := StrToFloatDef(EditA.Text, 1);    //将编辑框的字符串转换成浮点数据
    b := StrToFloatDef(EditB.Text, -2);    //将编辑框的字符串转换成浮点数据
    c := StrToFloatDef(EditC.Text, 1);     //将编辑框的字符串转换成浮点数据
    if(abs(a)<1E-6) then                   //如果方程系数等于 0, abs 为求绝对值的函数
        begin
            //结果输出
            Labelx1.Caption := '非法二次方程!';
            Labelx2.Caption := '';
            Exit;                          //求解结束, 退出过程, 此处可以不要
        end
    else                                  //如果是合法的一元二次方程, 则按步骤如下求解
        begin
            disc := b * b - 4 * a * c;      //求判别式
            if(disc < 0) then                //根据数学公式知道, 方程无实数解
                begin
                    //结果输出
                    Labelx1.Caption := '方程无实根!';
                    Labelx2.Caption := '';
                end
            else
                begin
                    x1 := (-b + sqrt(disc)) / (2 * a); //根据公式求 x1
                    x2 := (-b - sqrt(disc)) / (2 * a); //根据公式求 x2
                    //结果输出
                    Labelx1.Caption := 'x1 = ' + FloatToStr(x1); //x1 显示到标签上
                    Labelx2.Caption := 'x2 = ' + FloatToStr(x2); //x2 显示到标签上
                end
            end;
        end;
end;
```

### (4) 保存并运行程序。

选择 File→Save 命令, 保存单元文件与项目文件。

按 F9 键或者单击 Run 按钮运行程序, 在三个编辑框中输入任意系数, 然后单击“求解”按钮执行过程, 显示方程结果如图 5-13 所示。

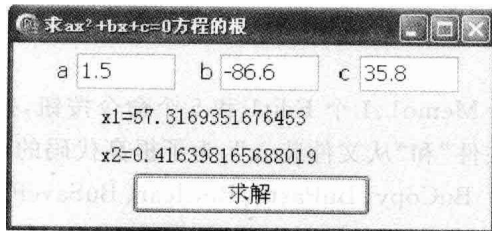


图 5-13 显示方程结果界面



本例中用到了 StrToFloatDef 这个函数,该函数的主要作用是将指定的字符串转换成浮点数,里面有两个参数,前一个是要转换的源串,后一个如果转换失败,就用此值返回。FloatToStr 函数的作用也是将浮点数转换成字符串,其只有一个参数,没有转换失败默认值。Sqrt 为数学上的开平方函数。

### 5.3.4 TMemo

TMemo(备注框)组件位于组件面板的 Standard 选项卡上(如图 5-8 所示),可以显示、编辑多行文本。TMemo 组件为用户提供了一种处理多行文本的方法,还可用来快速阅读文件中的数据。尽管 TLabel、TEdit 和 TMemo 组件有许多相似之处,但它们的使用场合却有所不同。如果只显示用户不能修改的信息,可采用 TLabel 组件;如果用户只处理(可以修改)单行信息,则使用 TEdit 组件;如果用户要处理(可以修改)多行文本,那么只能采用 TMemo 组件。

TMemo 组件的很多属性(如 MaxLength、ReadOnly 和 Text 等)和 TEdit 组件的属性类似,相同属性的含义不再赘述。下面介绍 TMemo 组件的特殊属性。

(1) ScrollBars 属性:用来设置 TMemo 组件是否有滚动条。

(2) WantTabs 属性:用来设置是否可以使用 Tab 键进行文本编辑。其值设置为 true,这时可以用 Tab 键使 TMemo 组件获得输入焦点,但不能用 Tab 键从 TMemo 组件切换到其他组件。默认值为 false。

(3) Lines 属性:与编辑框将 TEdit 组件输入内容保存在 AnsiString 类型的 Text 属性中不同,备注框 TMemo 组件的文本内容保存在 Lines 属性中,这是一个 TStringList 类型的属性,适合输入多行文本。在设计阶段可在 Object Inspector 中单击 Lines 属性右端的“...”按钮进入 String List Editor 对其进行编辑。

在程序运行中则应使用 Lines 的 Add、Insert、Delete 等方法对其修改。例如:

```
Memol.Lines.Add('在备注框底部添加一行内容。');
```

(4) SaveToFile 和 LoadFromFile 方法:严格来讲,SaveToFile 和 LoadFromFile 两个方法不是直接属于备注框的,它们是 Lines 属性(TStringList)的方法。使用这两个方法,可以分别将备注框中的文本内容保存到指定文件和从文本文件中读取内容到备注框,具体用法如例 5.6 所示。

**例 5.6** TMemo 综合示例:应用程序与操作系统剪贴板交换数据以及 SaveToFile、LoadFromFile 重要方法的应用。

详细实现步骤如下:

(1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。

(2) 定制窗体。

在 Form1 中加入 1 个 Memol、1 个 Edit1 和 5 个命令按钮,按钮的标签分别为“复制”、“粘贴”、“清除”、“保存到文件”和“从文件装入”,为了提高代码的可读性,将这 5 个按钮的名称 name 属性分别更改为 BuCopy、BuPaste、BuClear、BuSaveFile 和 BuLoadFile。其中, BuCopy 按钮将备注框内选中的文本复制到剪贴板; BuPaste 按钮则将剪贴板上内容粘贴到备注框中的当前插入点; BuClear 按钮可同时清空编辑框和备注框; BuSaveFile 和

BuLoadFile 按钮分别将备注框内容保存到文本文件和从文本文件中读取内容到备注框,该文本文件的路径和文件名应预先输入到 Edit1 中。为了显示内容全面且便于操作,将 Memo1 的 ScrollBars 属性设置成 ssBoth。

将各组件属性设置好并调整组件位置及大小后,窗体的设计界面如图 5-14 所示。

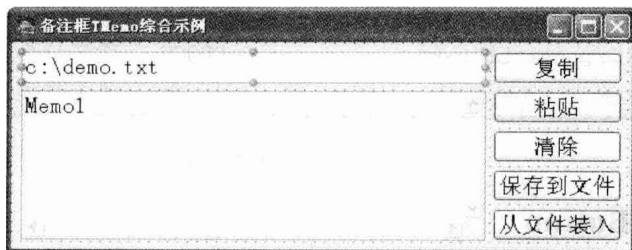


图 5-14 设计窗体界面

### (3) 编写代码。

依次双击命令按钮 BuCopy、BuPaste、BuClear、BuSaveFile 和 BuLoadFile,为它们依次添加 OnClick 事件处理过程代码:

```
procedure TForm1.BuCopyClick(Sender: TObject);
begin
    //将备注框的内容复制到剪贴板中
    Memo1.CopyToClipboard;
end;
procedure TForm1.BuPasteClick(Sender: TObject);
begin
    //将剪贴板中的内容粘贴到备注框
    Memo1.PasteFromClipboard;
end;
procedure TForm1.BuClearClick(Sender: TObject);
begin
    //将备注框的内容删除掉
    Memo1.Lines.Clear;
end;
procedure TForm1.BuSaveFileClick(Sender: TObject);
begin
    //将备注框中的内容保存到磁盘文件,用户必须确保 Edit1 中的文件名合法
    Memo1.Lines.SaveToFile(Edit1.Text);
end;
procedure TForm1.BuLoadFileClick(Sender: TObject);
begin
    //备注框从磁盘文件装入内容,用户必须确保 Edit1 中的文件名合法
    Memo1.Lines.LoadFromFile(Edit1.Text);
end;
```

### (4) 保存并运行程序。

选择 File→Save 命令,保存单元文件与项目文件。

按 F9 键或者单击 Run 按钮运行程序,在编辑框中输入合法磁盘文件,然后单击“从文件装入”按钮读入文件内容到备注框中,如图 5-15 所示。操作其他按钮可以实现题目要求。



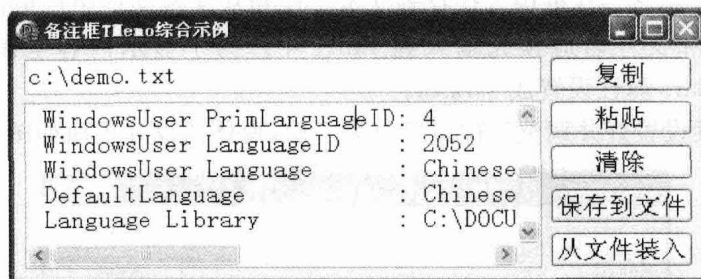


图 5-15 备注框运行效果界面

### 5.3.5 TRadioButton

TRadioButton(单选按钮)组件位于组件面板的 Standard 选项卡上(如图 5-8 所示),可以在多个条件中选择唯一的一个。通常在使用单选按钮时,总是将其进行分组。在同一组中,只能同时选中一个按钮,其余按钮自动取消选中。在实现单选按钮的分组时,利用分组框组件(TGroupBox),首先向窗体中添加 TGroupBox,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 TRadioButton 组件的 Checked 属性来确定哪一个单选按钮被选中。

TRadioButton 组件也称为开关按钮。单选按钮为用户提供了一组相互排斥的选项按钮,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另一个按钮,则先前被选择的按钮会自动变成未选中。

主要属性如下:

(1) Checked 属性:表示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。默认值为 false,表示未选中。

(2) Enabled 属性:用来控制单选按钮是否处于激活状态。默认值为 true,即激活状态。当该属性为 false 时,处于非激活状态,此时组件上显示的字体为灰色,表示该按钮不起作用。

(3) Visible 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。

### 5.3.6 TRadioGroup

前述的单选按钮一般用于在单选框里分组使用,在程序界面中可供用户在一组选项中进行选择。在一个单选框中,用户只能选择其中的一个按钮,不能多选也不能不选。Delphi 语言提供的 TRadioGroup(单选按钮组)比 TRadioButton 使用起来更容易一些,它在组件面板 Standard 选项卡上的位置如图 5-16 所示。

除了一般控件的共同属性外,RadioGroup 有以下常用属性和事件。

(1) Columns 属性:指示单选按钮在单选框内显示时分成几个列。如果框内有 4 个单选按钮,当 Columns=1 时,这 4 个按钮从上往下排成一行;当 Columns=4 时,从左向右排为 4 列;当 Columns=2 时,这 4 个按钮分成两列显示,每列含有两个单选按钮。

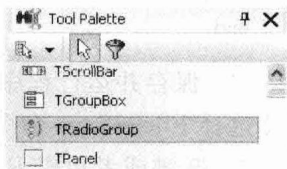


图 5-16 TRadioGroup 组件

(2) Items 属性: 是 TStrings 类型的多行字符串(文本), 可单击 Items 属性右端的“...”按钮打开 String List Editor 进行输入, 输入完成后, 该属性中文本的行数即为单选按钮的个数, 各行的内容即为对应按钮的标题。

(3) ItemIndex 属性: 为单选按钮组内当前选中项的索引号。ItemIndex=0 时, 当前选中位置为单选框内第一项。默认状态下为-1, 表示未选中任一项。

因为 Items 是 TStrings 类型的, 所以可以用 RadioGroup1.Items.Strings[RadioGroup1.ItemIndex] 取得 RadioGroup1 中当前选项的内容(为 AnsiString 字符串类型)。

(4) OnClick 事件: 通常在 OnClick 事件发生时, 根据 ItemIndex 属性判定选中的按钮并作出相应处理。

**例 5.7 TRadioGroup 综合示例: 通过选择色彩改变窗体背景。**

详细实现步骤如下:

(1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序。

(2) 定制窗体。

在 Form1 中加入一个 RadioGroup1 组件, 并设置其 Columns 属性为 2, 即让显示内容为两栏, 然后设置 Items 属性, 通过对话框设置“白色”、“绿色”、“红色”、“黄色”、“灰色”和“蓝色”颜色单选项; 将窗体背景的 Color 属性设置成初始白色, 将 RadioGroup1 对象的 ItemIndex 属性设置成 0, 表示选择“白色”。

将属性设置好并调整组件位置及大小后, 窗体界面如图 5-17 所示。

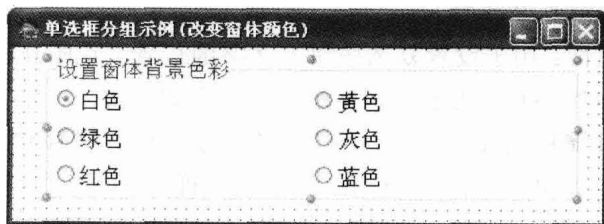


图 5-17 设计窗体界面

(3) 编写代码。

双击 RadioGroup1 添加 OnClick 事件处理过程代码:

//单选分组框单击事件程序

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
```

```
begin
```

```
    //多分支选择结构
```

```
    case RadioGroup1.ItemIndex of
```

```
        0: Form1.Color := clWhite;
```

```
        //选择“白色”
```

```
        1: Form1.Color := clGreen;
```

```
        //选择“绿色”
```

```
        2: Form1.Color := clRed;
```

```
        //选择“红色”
```

```
        3: Form1.Color := clYellow;
```

```
        //选择“黄色”
```

```
        4: Form1.Color := clGray;
```

```
        //选择“灰色”
```

```
        5: Form1.Color := clBlue;
```

```
        //选择“蓝色”
```

```
end;
end;
```

(4) 保存并运行程序。

选择 File→Save 命令,保存单元文件与项目文件。

按 F9 键或者单击 Run 按钮运行程序,选择任何一种颜色,窗体客户区背景颜色立刻发生相应的变化。图 5-18 所示为选择“黄色”单选按钮的运行效果图。

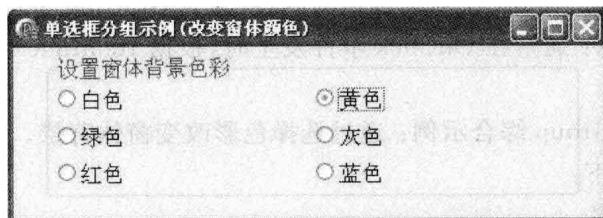


图 5-18 RadioGroup 运行效果界面

### 5.3.7 TCheckBox

TCheckBox(复选框)组件是应用程序中允许用户进行多项选择时使用的控件,位于组件面板的 Standard 选项卡上(如图 5-8 所示)。一个复选框由若干复选按钮组成,用户可以根据需要选择其中若干项(包括 0 项)。用户可通过单击一个选项来选择或取消选择该选项,在一个选择组框中一次可做出多项选择。复选按钮被选定时,会显示选定标记。

复选按钮除了一般对象所具有的 Name、Caption、Font、Enabled 和 Color 等基本属性外,其他主要的属性和重要事件如下:

(1) Alignment 属性:用于设置复选按钮上文字的位置。taRightJustify 表示文字显示在小方框的右边;taLeftJustify 表示文字显示在小方框的左边。

(2) AllowGrayed 属性:默认值为 false,这时复选框只有两种状态,即“选中”和“未选中”。但当该属性为 true 时,复选框则有三种状态,即“选中”、“未选中”和“部分选中”(灰色)。

(3) State 属性:用来设置或返回复选框的状态,可取以下值。

- cbChecked: 表示复选框处于启用状态。
- cbUnchecked: 表示复选框处于未启用状态。
- cbGrayed: 表示复选框处于启用且变灰状态。

(4) Checked 属性:用来检测复选框当前处于什么状态,Checked 属性为 true 时,表示该复选按钮被选中。默认状态为 false。实际应用中,由用户单击复选按钮来指定选中或未选中状态,程序检测控件的状态并执行相应的操作。如果复选框的 State 属性为 cbGrayed 或 cbUnchecked,则 Check 属性为 false。

(5) OnClick 事件。复选按钮最主要的事件,当单击发生时 Checked 属性的逻辑值会自动取反,应用程序对此应作出相应的反应。

下面的例子示范了复选框的用法。

**例 5.8 TCheckBox 综合示例:**通过多项选择改变文本字体风格。

详细实现步骤如下:

### (1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序。

### (2) 定制窗体。

为了将窗体设计得更个性一点, 首先通过颜色对话框更改窗体颜色为浅绿色; 然后在 Form1 中加入三个复选框 CheckBox1、CheckBox2 和 CheckBox3, 并设置其 Caption 属性依次为“粗体”、“下划线”和“斜体”, 为了使得看上去友好, 将三个复选框的字体调整为如其标签所述风格。

再将一个 Label1 组件放置到窗体上。Label1 的 Caption 设置为“类的概念: 客观世界中具有相似特性的事物的一种抽象, 如‘飞机’、人。在 Delphi 中, 类是用户自定义的结构数据类型。”。为了使 Label1 中文字能够以紧凑多行的方式显示, 要将 Label1 的 WordWrap 属性设置为 true, AutoSize 属性设置为 false, 并将 Label1 的长、宽调整合适。

建立图 5-19 所示的窗体, 要求运行时能根据复选框的状态设置 Label1 中的字体。此外, 设计时应注意 Label1 的字体应与三个复选按钮的 Checked 属性的初始状态相一致。

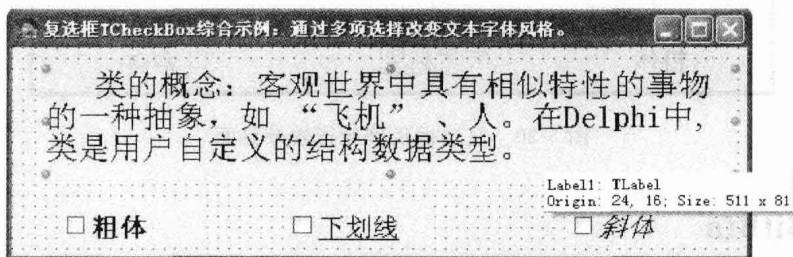


图 5-19 复选框示例设计界面

### (3) 编写代码。

依次双击三个复选框 CheckBox1、CheckBox2 和 CheckBox3, 为它们依次添加 OnClick 事件处理过程代码:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    //字体是否为粗体,集合操作
    if CheckBox1.Checked then
        Label1.Font.Style := Label1.Font.Style + [fsBold]           //集合并
    else
        Label1.Font.Style := Label1.Font.Style - [fsBold];         //集合减
end;
procedure TForm1.CheckBox2Click(Sender: TObject);
begin
    //下划线选择操作
    if CheckBox2.Checked then
        Label1.Font.Style := Label1.Font.Style + [fsUnderLine]     //集合加入下划线风格
    else
        Label1.Font.Style := Label1.Font.Style - [fsUnderLine];     //集合去除下划线风格
end;
procedure TForm1.CheckBox3Click(Sender: TObject);
```

```

begin
    //字体是否为斜体,集合操作
    if CheckBox3.Checked then
        Label1.Font.Style := Label1.Font.Style + [fsItalic]           //集合并
    else
        Label1.Font.Style := Label1.Font.Style - [fsItalic];         //集合减
end;

```

(4) 保存并运行程序。

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,这三个复选按钮的选取状态可以任意组合,即可以选取其中的任意一个、两个或三个以改变标签中的字体,或者全部都不选。部分选择的效果如图 5-20 所示。

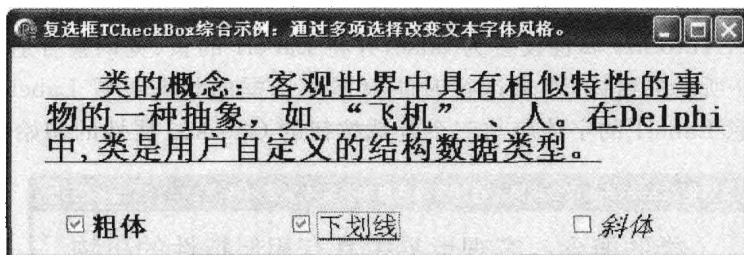


图 5-20 复选框示例运行效果界面

### 5.3.8 TBitBtn

TBitBtn(位图按钮)组件位于组件面板的 Additional 选项卡上,为图形按钮组件。TBitBtn 组件与其他按钮一样,在对话框和窗体中广泛使用。该组件的作用和 TButton 组件相同,唯一的区别是可以在位图按钮上同时显示文本信息和位图。

主要属性如下:

(1) Glyph 属性:指定在用户选择的位图按钮表面设置图形。单击该属性右边的“...”按钮,弹出装入位图文件的 Picture Editor 对话框,在其中单击 Load 按钮选择用户需要的位图,此时选择的位图将显示在对话框中,再单击 OK 按钮后,位图将显示在按钮上。

(2) Kind 属性:决定位图按钮的种类,包括 bkAbort、bkAll、bkCancel、bkClose、bkCustom、bkHelp、bkIgnore、bkNo、bkOK、bkRetry 和 bkRetry。

(3) Layout 属性:决定位图在按钮上的显示位置,即位图可以显示在按钮的左边、右边、上边或下边。

(4) Margin 属性:确定位图按钮边沿和位图边沿之间的距离,单位为像素。该属性的默认值为-1,表示位图和文字都在按钮的中间显示,但相互不重叠。

(5) NumGlyphs 属性:当用户为位图按钮在 Glyph 属性中指定多个图像时,必须用该属性指定按钮要显示的图像,它的值可以是 1~4 之一,默认值为 1。

(6) Spacing 属性:该属性确定位图按钮上图像和标题之间的间隔。该属性值可以是负数、0 和正数,默认值为 4。

(7) Style 属性:决定位图按钮的外观,共有三种可能的值: bsAutoDetect、bsWin31 和

bsNew,即位图按钮采用 Windows 哪一种版本的风格。

- bsAutoDetect: 表示当使用 Windows 3. x 时,位图按钮采用标准的 Windows 3. x 外观;当使用 Windows 3. x 以后的版本时,位图按钮采用较新的外观。
- bsWin31: 表示不考虑所运行的 Windows 版本,而采用标准的 Windows 3.1 外观。
- bsNew: 表示不考虑所运行的 Windows 版本,而采用新的位图按钮外观。

## 5.4 列表类与滚动条组件及时钟组件

### 5.4.1 TListBox

TListBox(列表框)组件位于组件面板的 Standard 选项卡上(如图 5-8 所示),可以显示一系列项目条文列表,用户可以选中其中的一个或多个项,但不能直接对这些项目显示文本进行修改。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、删除和插入编程操作。

TListBox 组件的主要属性和重要方法如下:

(1) Columns 属性:用来限定列表框中不用水平滚动条就可以看到的列数,默认值为 0,表示列表框单列显示。每列的宽度由列表框的 Width 属性和 Columns 属性共同决定。

(2) ExtendedSelect 属性:设置是否允许使用 Shift 键和 Ctrl 键对列表框中的项目进行连续选择。只有当该属性为 true 时,组件的 MultiSelect 属性才起作用,通过配合 Shift 键选择连续多个项目,通过配合 Ctrl 键选择不连续的多个项目。如果将该属性设置为 false,那么 MultiSelect 属性即便设置成 true 也无意义。

(3) IntegralHeight 属性:设置列表框在窗体上的显示方式。设置为 true 时,在垂直方向上可完整地看到列表框的项目;设置为 false 时,列表框的高度由 ItemHeight 属性决定。

(4) ItemHeight 属性:当列表框的 Style 属性设置为 lbOwnerDragFixed 时,列表框中每一个项目的高度由 ItemHeight 属性来决定;当 Style 属性为其他值时,ItemHeight 属性无意义。

(5) ItemIndex 属性:只能在程序中设置或引用,表示在列表框控件中当前所选项目的索引号。选中第一项时,ItemIndex 的值为 0;选中最后一项时,ItemIndex 的值为 Items.Count-1;如果没有选中任何项目,ItemIndex 的值为-1。

(6) Items 属性:用于设置列表框中显示的内容。Items 属性为 TStrings 类型,用于访问列表框中的项目。列表框中的每个项都是以字符串形式表示的数组元素。Items 属性可以在 Object Inspector 中进行设置,当单击 Items 属性右边的“...”按钮时,将弹出 String List Editor 窗口,用户可在该编辑窗口中输入将要在列表框中显示项目的默认值。也可以在程序运行阶段设置或引用。引用列表框中的项目时应使用如下格式:

列表框名称.Items.Strings[索引号]

用户在程序执行过程中通过调用方法动态地向列表框中增加项目,如在窗体中有一个 ListBox1 组件,则可通过代码改变列表框的内容,其方法如下。

- ① 向列表框中增加项目:ListBox1.Items.Add('增加的新项目')。
- ② 删除当前选中的项目:ListBox1.Items.Delete(ListBox1.ItemIndex)。



③ 清除所有项目: `ListBox1.Clear`。

(7) `MultiSelect` 属性: 用来设置用户是否可从列表框中一次选中多项。`MultiSelect` 属性的值如果为 `true`, 则表示用户可以一次选择多个项目; 如果 `MultiSelect` 属性的值为 `false` (默认值), 则表示用户一次只能选择一个项目。

(8) `SelCount` 属性: 当 `MultiSelect` 属性设置为 `true` 时, 该属性返回在列表框中用户选择项目的个数。

(9) `Selected` 属性: 只能在程序中设置或引用, 反映列表框中的项目是否被选中。该属性是一个逻辑型数组, 各项的取值为 `true` 或 `false`。例如, `Selected[0]` 为 `true` 表示第一项被选中。

(10) `Sorted` 属性: 该属性用来控制列表框中的数据是否自动进行排序。如果将其设置为 `true`, 那么列表框的数据按字母顺序进行排序; 默认值为 `false`, 即不对数据进行排序。注意: 排序时不区分大小写。

(11) `Style` 属性: 用来设置列表框中项目的显示方式, 常取以下值。

- `lbOwnerDrawFixed`: 列表框中的每个项目可以是字符, 也可以是图像, 其高度由 `ItemHeigh` 属性决定。
- `lbOwnerDrawVariable`: 列表框中的每个项目可以是字符或图像, 其高度可变。
- `lbStandard`: 默认值, 即列表框中的每个项目作为一个高度相同的字符串显示。

(12) `OnClick` 事件: 当单击(选择)列表框中的项目时, 触发 `OnClick` 事件, 此时程序可根据选定项目确定应该执行的任务。

(13) `Add` 方法: 列表框中的项目除了可以在设计阶段输入 `Items` 进行添加外, 也可在程序中(如在窗体的 `OnCreate` 事件中)调用 `Add` 方法添加。`Add` 实际上是 `ListBox` 的 `Items` 属性的方法(`Items` 是 `TStrings` 类型的, 因此与 `Memo` 的 `Lines` 有一样的方法), 常用于在程序运行阶段将项目添加到列表框。例如:

```
ListBox1.Items.Add('Hello');
ListBox1.Items.Add('New');
```

(14) `Delete` 方法: `Delete` 实际上是 `ListBox` 的 `Items` 属性的方法, 该方法可用在程序运行阶段将指定的项目从列表框中删除。例如:

```
ListBox1.Items.Delete(8);
//将索引号为8的项目从列表中删除
ListBox1.Items.Delete(ListBox1.ItemIndex);
//删除列表框中当前被选定的项目
```

(15) `Clear` 方法: 用于在程序运行阶段删除列表框中的所有项目。用法:

```
ListBox1.Clear;
```

下面举例示范列表框的用法。

**例 5.9** `TListBox` 组件综合示例: 编写对列表框中的项目进行添加、删除和统计的应用程序。

详细实现步骤如下:

(1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序。

### (2) 定制窗体。

将 Form1 的 Color 属性通过对话框修改为浅黄色; 在窗体上放置一个 ListBox1 组件, 更改背景色为 clYellow, 并设置它的 MultiSelect 属性为 False; 放置 CheckBox1 和 CheckBox2 组件, 并更改其 Caption 属性分别为“允许选择多项”和“允许自动排序”; 添加一个 TEdit 组件, 目的是为用户输入新的项目内容, 并将其 Text 属性清除; 进而添加三个 TButton 组件, 为了使代码可读性良好, 将这三个按钮的 name 属性修改为 buAdd、buDelete 和 buExit, 同时修改其 Caption 属性的值依次为“添加”、“删除当前选择项”、“退出”。

最后给窗体上放置 Label1 和 Label2 对象, 并初始化其 Caption 属性的值依次为“当前选择项 0 条!”和“列表总项数: 0 条!”。

将各组件属性设置好并调整组件位置及大小后, 窗体界面如图 5-21 所示。

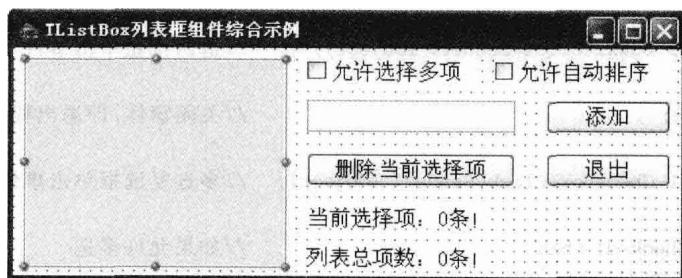


图 5-21 TListBox 组件示例设计界面

### (3) 编写代码。

首先为 TForm1 类编写一个自定义方法用于统计并显示列表的当前选择项数目和列表的总选择项数目。

在 Unit1 单元的接口部分(interface)TForm1 类代码的私有(Private declarations)位置声明这个方法原型如下:

```
procedure fmyCount;
```

在 Unit1 的单元实现部分(implementation)给出该方法的完整代码如下:

```
procedure TForm1.fmyCount;
var i, SelNum, TotalNum: integer;           //定义变量
begin
    TotalNum := ListBox1.Items.Count;       //获取列表框总项数
    SelNum := 0;                            //初始化当前选择的项数为 0
    for i := 0 to ListBox1.Items.Count - 1 do //扫描所有的项
    begin
        if (ListBox1.Selected[i]) then     //如果当前项处于选择状态
            SelNum := SelNum + 1;
    end;
    //将数据输出到标签上, IntToStr 函数功能是将整型数转换成字符串
    Form1.Label1.Caption := '当前选择项: ' + IntToStr(SelNum) + '条!';
```

```
Form1.Label2.Caption := '列表总项数: ' + IntToStr(TotalNum) + '条!';
end;
```

程序的任何一个动作都应该调用此 fmyCount 方法,确保统计数据得到正确的显示。  
本程序事件的其他代码编写如下:

```
procedure TForm1.BuAddClick(Sender: TObject);      //添加按钮单击事件
begin
    //如果输入框的内容不为空白,Length 为求字符串长度函数
    if Length(Edit1.Text)>0 then
        ListBox1.Items.Add(Edit1.Text);           //添加内容
        fmyCount;                                   //调用自定义方法,刷新统计显示
end;
procedure TForm1.BuDeleteClick(Sender: TObject);   //删除选择按钮单击事件
begin
    ListBox1.DeleteSelected;                        //删除所有选择项,对象自有的方法
    fmyCount;                                       //调用自定义方法,刷新统计显示
end;
procedure TForm1.BuExitClick(Sender: TObject);     //退出按钮单击事件
begin
    Form1.Close;                                   //关闭窗体,即退出程序
end;
procedure TForm1.CheckBox1Click(Sender: TObject);  //多选复选框单击事件
begin
    if (CheckBox1.Checked) then                    //如果允许多选
        ListBox1.MultiSelect := true
    else
        ListBox1.MultiSelect := false;
    fmyCount;                                       //调用自定义方法,刷新统计显示
end;
procedure TForm1.CheckBox2Click(Sender: TObject);  //排序复选框单击事件
begin
    if (CheckBox2.Checked) then                    //如果允许自动排序
        ListBox1.Sorted := true
    else
        ListBox1.Sorted := false;
    fmyCount;                                       //调用自定义方法,刷新统计显示
end;
procedure TForm1.FormCreate(Sender: TObject);      //窗体创建事件
begin
    fmyCount;                                       //调用自定义方法,刷新统计显示
end;
procedure TForm1.ListBox1Click(Sender: TObject);   //列表框单击事件
begin
    fmyCount;                                       //调用自定义方法,刷新统计显示
end;
```

(4) 保存并运行程序。

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,单击“添加”按钮,可将编辑框中输入的内容添加到左面的列表框中;单击

“删除当前选择项”按钮,可删除列表框中选定的项目,如果没有选定删除项目,则“删除当前选择项”按钮没有任何动作;当“允许选择多项”复选框被选中时,列表框中的项目可以多选,否则只能单选;当“允许自动排序”复选框被选中时,列表框中的项目按其内容排序,否则按添加的先后排列,其效果如图 5-22 所示。

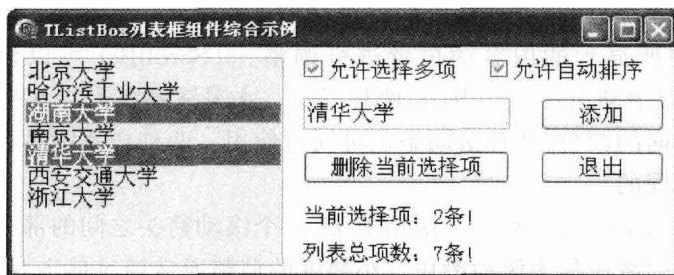


图 5-22 TListBox 组件示例运行效果界面

### 5.4.2 TComboBox

TComboBox(组合框)组件位于组件面板的 Standard 选项卡上(如图 5-8 所示)。组合框由一个编辑框和一个下拉式列表框组成,可以从多个列表条目中选择一个。TComboBox 组件是设计 Windows 应用程序时使用较多的组件之一,该组件汇集了列表框和文本框的功能,其作用是供用户从下拉列表中选择数据或直接向组合框中输入数据。组合框一方面具有编辑框的 Text、MaxLength 等属性,另一方面又具有列表框的 Items、ItemIndex 和 Sorted 等属性,并且用法也是相似的。

主要属性介绍如下:

(1) DropDownCount 属性:用来设置组合框下拉列表中可显示的文本行数。默认值为 8,即下拉列表在不需要滚动的情况下可以显示 8 个项目(8 行文本)。

(2) Enabled 属性:用来设置是否使组合框处于激活状态。如果将其设置为 false,则组合框处于非激活状态,程序运行时组合框区域呈灰色显示。

(3) Style 属性:用来设置组合框中项目的显示方式。可取以下值:

- CsDropDown: 为默认值,是由编辑框和列表框组合而成的组合框,列表框中的每一个项目都是高度相同的字符串。
- CsDropDownList: 该方式只能显示而不能输入,用户只能从下拉式列表中选择项目。
- CsOwnerDrawFixed: 和 ListBox 组件的 lbOwnerDrawFixed 方式相同。
- CsOwnerDrawVariable: 和 ListBox 组件的 lbOwnerDrawVariable 方式相同。
- CsSimple: 只有编辑框而没有列表框。

(4) Text 属性:用来设置或返回编辑框上显示出来的文字内容,也可在程序中给该属性赋值。该属性的赋值语句为:

```
<组合框名>.Text := <组合框名>.Item[索引号];
```

(5) ItemIndex 属性:返回 ComboBox 中被选中项目的索引号,从 0 开始,依次加 1。

(6) SelText 属性:返回 ComboBox 中被选中的文本。



另外,用户可通过 Add、Delete 和 Insert 方法对列表中的项目(Item)进行增加、删除和插入操作。

### 5.4.3 TScrollBar

TScrollBar(滚动条)组件在面板上的位置如图 5-8 所示。利用滚动条控件可对其相关联的其他控件中所显示的内容的位置进行调整。TScrollBar 可以设置成水平滚动条(HScrollBar)和垂直滚动条(VScrollBar)两种形式。水平滚动条进行水平方向的调整,垂直滚动条进行垂直方向的调整,两种滚动条也可同时使用。两种样式滚动条除外观不同外,作用和使用方法是相同的。

滚动条两端带箭头的按钮称之为滚动箭头,两个滚动箭头之间的部分称为滚动框,滚动框中可以左右移动的滑块称为滚动滑块。小幅度的调整通常通过单击或连续单击滚动箭头来实现;如果要进行较大幅度的调整,可单击或连续单击滚动框,如果要进行快速调整,则可拖动滚动滑块。

滚动条有如下常用属性、事件与方法。

- (1) Kind 属性:有两个可选值: sbHorizontal(水平滚动条)和 sbVertical(垂直滚动条)。
- (2) Position 属性:表示滑块在滚动条中的当前位置,其值始终介于 Max 和 Min 属性值之间,也可以等于这两个值。
- (3) Max 和 Min 属性:Max 和 Min 的值设置了 Position 的变化范围。当滚动滑块位于水平滚动条最右端或垂直滚动条最底端时,Position 取得最大值 Max;当滚动滑块位于水平滚动条最左端或垂直滚动条最顶端时,Position 取得最小值 Min。Max 的默认值为 100,Min 的默认值为 0。
- (4) LargeChange 和 SmallChange 属性:SmallChange 属性表示单击滚动条两端的箭头时,Position 属性增加或者减少的数值。LargeChange 属性表示单击滚动滑块和滚动箭头之间的区域时,Position 属性增加或者减少的数值。它们的默认值都为 1。
- (5) OnChange 事件:滚动滑块的位置改变时,触发 OnChange 事件。
- (6) OnScroll 事件:在拖动滚动滑块时,触发 OnScroll 事件;在单击滚动箭头或滚动条时不触发该事件。

**例 5.10** TComboBox 组件与 TScrollBar 组件的综合示例:通过组合框动态选择改变窗体的标题,通过滚动条调整窗体的背景色。

详细实现步骤如下:

- (1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。

- (2) 定制窗体。

在 Form1 窗体上放置三个标签 Label1、Label2、Label3,并将其 Caption 属性更改为“红色成分”、“绿色成分”和“蓝色成分”。然后放置三个滚动条组件 ScrollBar1、ScrollBar2 和 ScrollBar3,因为任何一个颜色都是由 Red、Green、Blue 组成,并且它们的值位于 0~255 之间,所以需要设置所有滚动条的 Min 属性值为 0,Max 属性值为 255。

进而,在 Form1 窗体上放置一个 Label4 组件,设置其 Caption 属性为“选择标题”。放

置一个 ComboBox1 组件,并修改其 Items 属性,通过对话框输入若干文字内容,以供改变窗体标题用。

将各组件属性设置好并调整组件位置及大小后,窗体界面如图 5-23 所示。

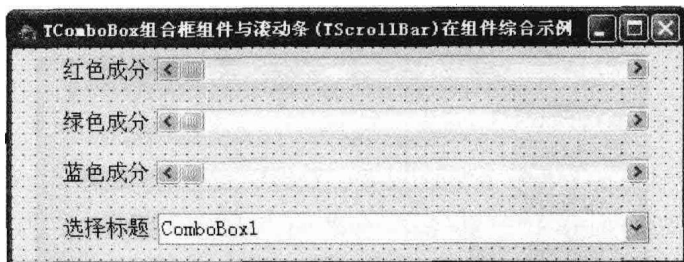


图 5-23 TComboBox 组件与 TScrollBar 组件示例设计界面

### (3) 编写代码。

Delphi 中 RGB 函数将三个颜色参数组合成一个新的颜色数值。

在窗体中选中 ScrollBar1 组件,编辑其 OnScroll 事件,输入滚动触发事件代码:

```
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  //将背景窗体颜色设置成三个滚动条当前位置组合的颜色
  Form1.Color := RGB(ScrollBar1.Position, ScrollBar2.Position, ScrollBar3.Position);
end;
```

本例中三个滚动条滚动触发的事件代码实际上是完全一样的,所以也可按如下方式处理:输入完 ScrollBar1 的 OnScroll 代码后,同时选中 ScrollBar2 与 ScrollBar3 组件,在 Object Inspector 的 Events 页上,从 OnScroll 右端组合框中选中 ScrollBar1Scroll 项,如图 5-24 所示,表示 ScrollBar2、ScrollBar3 与 ScrollBar1 的 OnScroll 使用同一个处理函数。

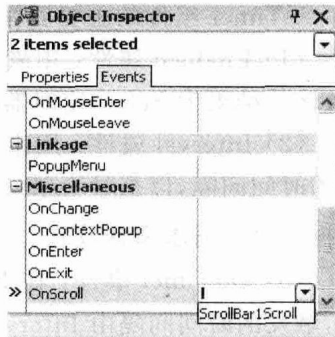


图 5-24 多个组件相同事件使用同一个处理函数

双击 ComboBox1 组件,输入其 OnChange 事件代码:

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  //用组合框的选择内容更改窗体标题
  Self.Caption := ComboBox1.Text; //Self 在这里与 Form1 等价
end;
```

为了使得窗体启动的时候,颜色跟三个滚动条初始表示颜色一致,故要给其创建窗体事件加入如下代码:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  //Self 在这里与 Form1 等价,背景色设置成三个滚动条当前位置组合的颜色
  self.Color := RGB(ScrollBar1.Position, ScrollBar2.Position, ScrollBar3.Position);
end;
```

(4) 保存并运行程序

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,通过滚动条的滚动变化可以动态调整窗体颜色,同时组合框内容的变化也会反应到窗体的标题上,其效果如图 5-25 所示。

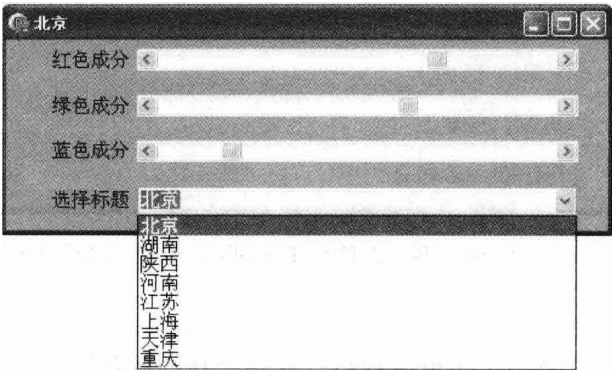


图 5-25 TComboBox 组件与 TScrollBar 组件示例运行效果界面

### 5.4.4 TTimer

TTimer(定时器)组件的主要功能是系统每隔一定的时间可以触发一个 OnTimer 事件,该组件非常有用,常用于自动化系统中。该组件位于组件面板 System 页面的第一行,如图 5-26 所示。

TTimer 组件的主要属性和事件如下:

(1) Enabled 属性:用来设置程序运行时定时器是否正在运行,该值结果为 False,定时器将不定时触发 OnTimer 事件。

(2) Interval 属性:用来设置定时器两次 OnTimer 事件发生的时间间隔,以毫秒为单位,系统默认值为 1000ms,用户可以根据自己的需求来修改它。

(3) OnTimer 事件:在 Enabled 属性值为 True 时,该事件每隔一定时间间隔自动触发,触发的时间间隔由 Interval 属性指定。

TTimer 定时器组件示例读者可查看本书的例 5.2 相关内容。

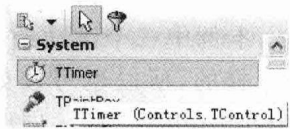


图 5-26 TTimer 组件

## 5.5 组件排列布局

使用 Delphi 设计 Windows 应用程序界面时,将对象添加到窗体上仅仅完成了界面设计的基本工作,接下来还必须对各个对象的位置、大小、对象间的间距等进行调整,对窗体上的所有对象进行整体布局,这样才能设计出美观的程序界面。

使用系统提供的排列组件命令可以快速而准确地排列组件,在排列组件之前首先要选定组件。

### 1. 组件的选定

(1) 单选:单击某个组件可以单独选中某个组件,以前选中的组件处于非选中状态。

(2) 多选：设计界面时经常会遇到需要同时对一组对象的位置和大小进行调整的情况，这就需要在窗体上同时选中多个对象，具体操作方法是按住 Shift 键的同时，逐个单击需要调整的对象(如图 5-27 所示)。

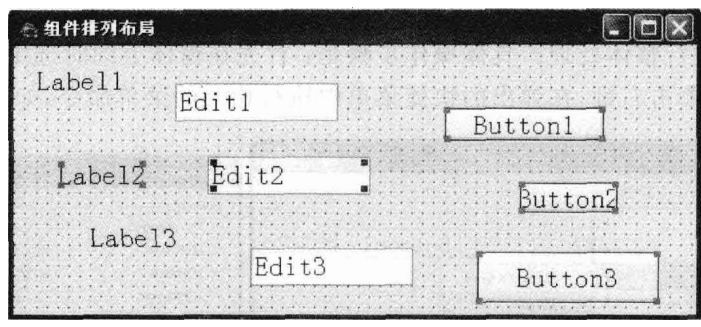


图 5-27 使用 Shift 键同时选取多个组件

## 2. 使用鼠标或者方向键拖拉调整组件对象的位置和大小

调整组件对象的位置和大小最简单的方法是先单击要调整的对象，这时对象周围将出现 8 个蓝色小方块(称为“拖拽柄”，如图 5-28 所示)，表示该对象处于选中状态。接下来如果要移动对象，只要将鼠标移至该对象上，按住鼠标将对象拖至目标位置，然后松开鼠标即可。如果要调整对象的大小，可将鼠标移到对象相应的“拖拽柄”上，然后按住鼠标进行拖放。

除了使用鼠标进行调整以外，也可以使用键盘上的 Ctrl、Shift 和方向键对组件对象的位置和大小进行调整。按住 Ctrl 键的同时，按下相应的方向键可以对对象的位置进行调整；按住 Shift 键的同时，按下相应的方向键可以对对象的大小进行调整。

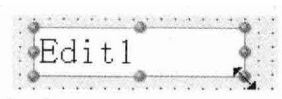


图 5-28 鼠标拖拉调整组件对象

一组对象同时被选中后，可以同单个对象的调整方法一起来调整它们的位置和大小。

## 3. 使用系统快捷菜单调整多个组件大小

设计界面时要求一组对象高度相同或宽度相同或两者都相同的情况也会经常遇到，如果采用逐个处理的方法既费时又费力，最为简单的方法是使用菜单命令进行整体处理。具体操作步骤是：首先在窗体上同时选中要进行处理的各个对象，如图 5-29 所示。然后单击右键，在弹出的快捷菜单中执行相关的菜单命令，按照对话框可以进行多种形式的调整。

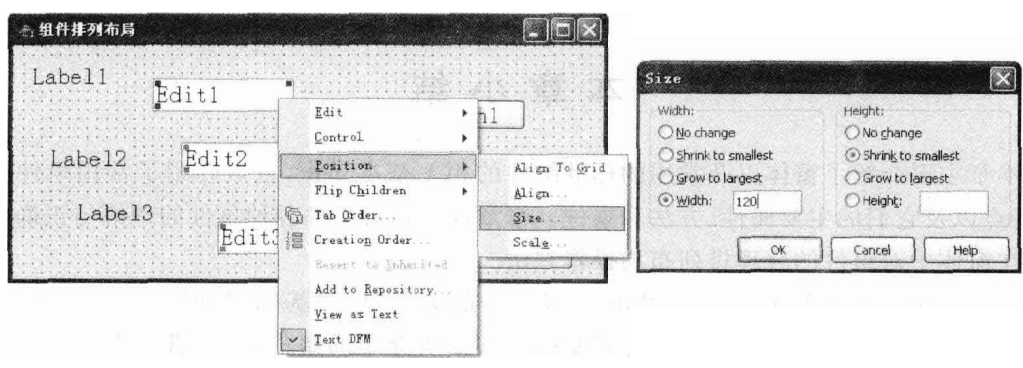


图 5-29 系统快捷菜单调整多个组件大小



#### 4. 组件对象的对齐与对象的间距调整

设计界面时经常需要对一组对象进行对齐处理或者间距调整。对齐方式有左对齐、右对齐、中间对齐和顶端对齐等多种方式。不管界面上的对象是横向排列,还是纵向排列,合理调整对象之间的间距,对于界面的美观都是非常必要的。遇到这种情况,最为简单的方法是使用菜单命令进行整体处理。具体操作步骤是:首先在窗体上同时选中要进行对齐处理的各个对象,然后单击右键,在弹出的快捷菜单中执行相关的菜单命令,如图 5-30 所示。

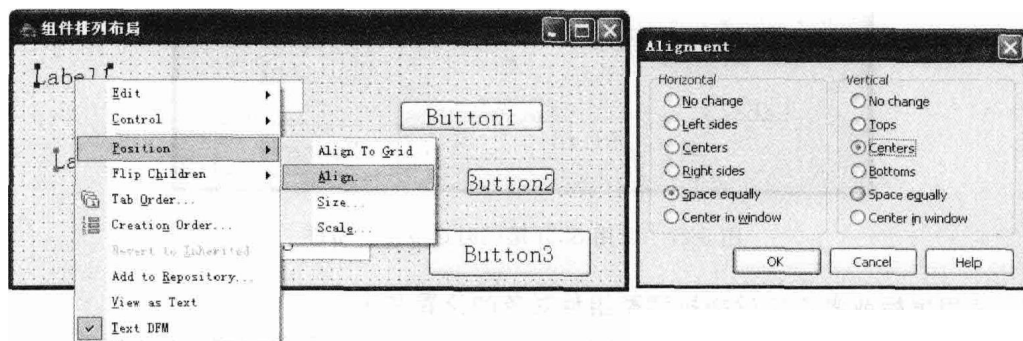


图 5-30 组件对象的对齐与对象的间距调整

#### 5. 利用系统工具条按钮调整

选择 View→ToolBars→Align 命令,可以打开组件排列调整快捷工具条,如图 5-31 所示。

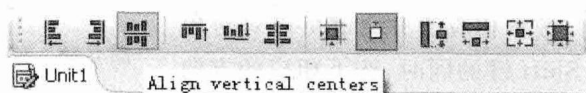


图 5-31 组件排列调整快捷工具

操作: 在窗体上同时选中要进行对齐处理的各个对象,然后单击工具条按钮,执行相关的排列调整命令。

#### 6. 组件的锁定

选择 Edit→Lock Controls 命令可锁定窗体上的组件,组件被锁定后能避免误操作所引起的位移、删除或尺寸改变。

## 本章小结

本章主要介绍了窗体以及常用的标准组件的相关基本知识,重点介绍了常用组件的分类、特点以及它们的主要属性、方法和事件。本章难点是对组件各种事件的理解与正确应用以及在窗体中对组件的合理排列布局操作方法。

这些常用组件是 Delphi 开发 Windows 程序的关键,也是基础,掌握它们的属性、方法和事件才能更好地为工程项目设计打好基础。本章所介绍的组件,可以满足大部分中小型项目开发的需要。

## 思考与练习

1. 按照组件在运行期间是否可见, Delphi 中的组件可以分为哪两大类型? 各有何特点?
2. TEdit 组件和 TMemo 组件在使用上有何不同?
3. TRadioButton 组件和 TCheckBox 组件的用途分别是什么? 在使用场合上有何区别?
4. TListBox 组件和 TComboBox 组件的用途有何不同?
5. Delphi 2007 for Win32 的组件排列布局的方法有哪些?
6. 利用常用组件设计一个具有加、减、乘、除功能的简易计算器, 如图 5-32 所示。



图 5-32 计算器界面

界面设计是软件开发的重要组成部分,一般的 Windows 应用程序界面包含菜单设计、工具条设计、状态栏设计、窗体设计和分割技术等内容。

菜单是 Windows 应用程序设计中的重要组成部分,大部分应用程序的功能靠菜单来实现。Delphi 2007 的可视化开发环境提供了大量的开发工具,使得 Delphi 2007 应用程序开发中菜单的设计变得方便灵活。用户可以通过组件面板的 Standard 选项卡的 MainMenu 组件来创建用户程序的主菜单,可以用 PopupMenu 组件来设计下拉菜单。

本章主要包括以下内容:

- 创建主菜单;
- 鼠标右键弹出式菜单;
- 工具栏与状态行设计;
- 对话框函数;
- 对话框组件;
- 多文档界面(MDI)程序设计;
- Delphi 拖放技术编程;
- 窗体的分割技术。

### 6.1 创建主菜单

常见的 Windows 应用程序的菜单通常分为两级:第一级是窗口标题下的菜单栏,称为主菜单;第二级是这些菜单所包含的下拉子菜单,称为菜单项。当用户选择了顶层某个菜单项时,就会弹出与其相连的菜单——子菜单。下拉式的子菜单由多个菜单项组成,可以将多个菜单项进行分组,把相关的菜单项作为一组,组与组之间以分隔线隔开。

按照菜单项的功能,可以将菜单项分为 4 种类型:

- (1) 命令菜单:用来执行某项操作的菜单项,这是用户最常见、使用最频繁的菜单项。
- (2) 状态设置菜单:用来对系统包括菜单本身的某些状态进行设置和说明,通常这些菜单项旁边都用对号或者复选框来表示其是否处于开启状态。
- (3) 对话框菜单项:用来激活对话框,通常不执行具体的操作,而是由用户通过对话框的操作来完成相应的任务。通常这些菜单项的旁边都有省略号(…)来进行识别。

(4) 下拉菜单：用来打开下级菜单。通常在这些菜单项的右边用黑色小三角号(►)标识,表示选中它后又会弹出一个菜单,这样就可以形成多级菜单。

### 6.1.1 TMainMenu 组件概述

MainMenu 组件位于组件面板的 Standard 页上(如图 6-1 所示),将它放入一个应用程序的窗体,然后就可以利用菜单设计器(Menu Designer)设计菜单了。

TMainMenu 组件用来创建主菜单,一般通过鼠标和键盘的热键来激活特定的菜单项。当用户将一个 TMainMenu 组件添加到窗体(TMainMenu 组件不能改变大小)时, Object Inspector 中会显示 TMainMenu 组件的属性。用鼠标双击该组件,可打开菜单设计器进行菜单设计或者编辑,如图 6-2 所示。

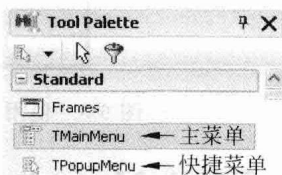


图 6-1 TMainMenu 和 TPopupMenu 组件

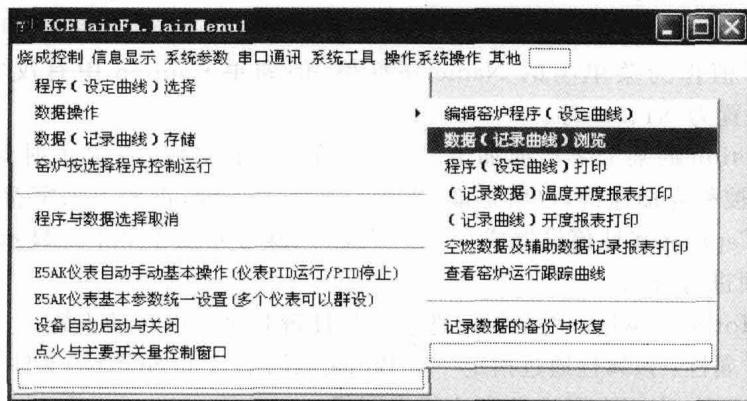


图 6-2 用菜单设计器进行某工控系统菜单设计

在 Menu Designer 中的操作主要用鼠标。当选中某个菜单项时, Object Inspector 会切换到对应该项的 MenuItem 对象,此时可输入它的 Caption 属性,也就是菜单项的名称。若要删除某菜单项,可在 Menu Designer 内选中该项后按 Delete 键,也可右击该菜单项打开它的 MenuItem 快捷菜单,再选该菜单上的 Delete 命令。若要插入菜单项,可在选中插入位置后按 Insert 键,也可右击该菜单项打开它的 MenuItem 快捷菜单,再选该菜单上的 Insert 命令。若要移动菜单项的位置,可以用鼠标进行拖曳完成。

此外,如图 6-3 所示,还可用 MenuItem 的快捷菜单上的 Create SubMenu 选项为菜单项增加一个子菜单,这样就可构建任意层数的菜单系统。如果要在程序运行中动态地增减菜单项,则可在程序中调用 TMenuItem 对象的 Add 和 Delete 方法。

根据菜单在应用程序中的作用,它的选项常被指派执行某项功能或操作。Delphi 中的 MenuItem 对象都有 OnClick 事件,为该事件编写的代码即被指派到菜单项对应的操作中。在 Menu Designer 中双击菜单项即可打开对应的 OnClick 事件代码段;在窗体设计区中单击菜单项也能打开 OnClick 事件代码段。当然,还可以在 Object Inspector 中打开 MenuItem 的 OnClick 事件。

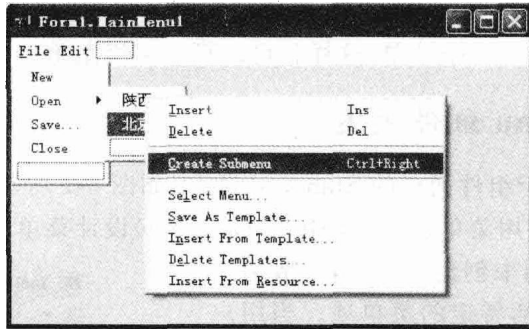


图 6-3 使用 MenuItem 的快捷菜单进行用户菜单设计编辑

### 6.1.2 TMainMenu 组件的主要属性介绍

TMainMenu 组件最重要的属性为 Items 属性,该属性可以通过菜单设计器设计时的对话框修改。Items 属性的原型为 TMenuItem 类,下面对它进行详细的介绍。

(1) Caption 属性:菜单项上的文字就是该菜单项的 Caption 属性值。Delphi 一般会默认将该 Caption 值作为菜单项的 Name 属性值,但如果 Caption 中有汉字,则菜单项的 Name 被默认设置为 N1、N2 等。

如果在 Caption 的某个字母前加上一个 & 符号,则该字符在显示时就会有一个下划线,这一字符就被称为加速键,也就是说,当用户按下 Alt 键,再加上该字符时,即可选择该菜单。如果将 Caption 属性值设置成一个“-”符号(减号),则在菜单上显示为一条横线,可用此线对菜单项进行分组。

(2) AutoHotkeys 属性:用来指定选项的快捷键是否可以自动重置。

(3) Images 属性:该属性通过指定一组 Bitmap(位图)或 Icon(图标)文件,在 MenuItem 中指定 Images 的索引,实现在各选项旁边加入图像。

(4) Items 属性:用于存储主菜单中各菜单项的内容。双击 Items 属性右侧的“...”按钮,进入菜单设计器。在菜单设计器下,所添加的每一个选项也将对应一组属性。

(5) AutoMerge 属性:用来设置在多文档的应用程序中,非主窗体的主菜单是否和主窗体中的主菜单合并。当 AutoMerge 属性的值为 false 时,非主窗体的主菜单不能和主窗体的主菜单合并。

在该属性不为 true 时,用户仍可执行 Merge 方法将两个菜单合并。UnMerge 方法则将已合并的菜单拆分。

(6) Checked 属性:若为 true,则会在菜单项左方显示一个钩,使该菜单项具有和一个复选按钮类似的作用。但注意,与复选按钮不同的是,Delphi 不会自动对 Checked 置值,用户必须在该菜单项的 OnClick 中写入改变 Checked 值的代码。

(7) Enabled 属性:表示使能开关,与其他控件的 Enabled 属性的用法类似,该属性为 false 时,对应的菜单项颜色变灰且不能被选择。该属性不仅可用于控制主菜单上的项,也可控制下拉菜单上的项。若要相应菜单项变为不可见,则可设置它的另外一个属性 Visible 为 false。

(8) GroupIndex 属性:该属性是整型值,可控制下拉菜单在主菜单中的位置(即顺序)。一般用于在菜单合并时,确定合并后菜单项的位置。

(9) ShortCut 属性:定义了一个可快速访问某个菜单项的快捷键。一般地,该快捷键

显示在菜单项的右边。在菜单中可以作为快捷键的一般都是功能键(如 F1, F2, ...) 或组合键(如 Ctrl + A, Ctrl + F2, Alt + F3, ...)。菜单项被选中时,在 Object Inspector 中打开 ShortCut 的组合框选取或输入。

(10) Visible 属性: 为 false 时,对应的菜单项不可见(不可见的项当然也是不能被选取的)。当某一项不可见时,排列在该项之后的各项位置会自动上移。

### 6.1.3 主菜单的设计过程

要为应用程序的窗体创建主菜单,首先从 Standard 组件页上选择 TMainMenu 组件,将其加入到窗体中;然后双击 TMainMenu 组件,就会弹出菜单设计器。首先出现在菜单设计器中的是一级菜单,可以逐级输入菜单项名称以创建菜单项。完成后,再在 Object Inspector 窗口中设置各菜单项 TMenuItem 的属性。通常会对菜单项 TMenuItem 进行如下设置:

#### 1. 设置快捷键

在 Caption 属性的输入域中,将“&”符号放到需要指定为热键的字母前面,该字母将被用下划线显示,运行时,按 Alt+热键字母可以激活该菜单项。对于中文菜单,可在说明文字之后增加一个括号,括号中添加一个具有代表性的字符作为快捷键。

#### 2. 定义热键

也就是定义由 Ctrl、Alt 或 Shift 键与某个字符键的组合,如 Ctrl+O 组合键通常用来打开文件。单击 ShortCut 属性右边的下拉列表,从中选择热键组合。

#### 3. 建立菜单分组

将下拉式菜单中的菜单项分组是菜单设计中的常见手段,分组主要是使用分隔条,可以通过设置 Caption 属性为“-”(减号)来实现菜单的分隔条(注意,只要输入一个“-”就够了,不要多输入)。

#### 4. 为菜单项增加图标显示

可以通过两种方法为每一个菜单项增加图标。一种方法是设置菜单项的 Bitmap 属性;另一种方法是为 TMainMenu 组件的 Images 属性指定图标列表(与一个 TImagesList 组件关联),然后设置每一个菜单项的 ImageIndex 属性以指定要使用的图标。

#### 5. 设计级联菜单

所谓级联菜单,就是当鼠标选中某一菜单项时,如果该菜单项有子菜单,则子菜单并列显示于该菜单的右边。要建立级联菜单,选中要建立子菜单的菜单项,单击鼠标右键,在弹出的快捷菜单中选择 Create Submenu 命令,然后就会在原菜单项旁边出现一个向右的箭头,并带有一个空白菜单项的级联菜单。子菜单项的设置与前面所讲的设置方法相同。

#### 6. 为菜单项指定动作

设置了菜单项的外观后,菜单栏在程序运行时只有一个空架子,需要为每一个菜单项定义 OnClick 事件处理函数,以便完成一系列的命令。在菜单设计器中双击菜单项,这时系统自动生成该菜单项的 OnClick 事件处理函数的框架,用户为其增加特定代码便可完成特定的功能。

### 6.1.4 TMainMenu 菜单设计综合示例

例 6.1 TMainMenu 组件综合示例:在窗体上建立一个下拉菜单,通过设置其菜单项

属性值,演示其各种功能。

详细实现步骤如下:

(1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。

(2) 定制窗体。

在 Form1 窗体上放置一个主菜单组件 MainMenu1 和一个图片列表组件 ImageList1 (该组件位于组件面板的 Win32 页,用于美化菜单项使用,可以给菜单项增加图片显示)。双击 ImageList1 组件,通过对话框(如图 6-4 所示)从磁盘上增加若干幅位图。设置 MainMenu1 的 Images 属性为 ImageList1,目的是让菜单组件跟图片列表组件关联起来。

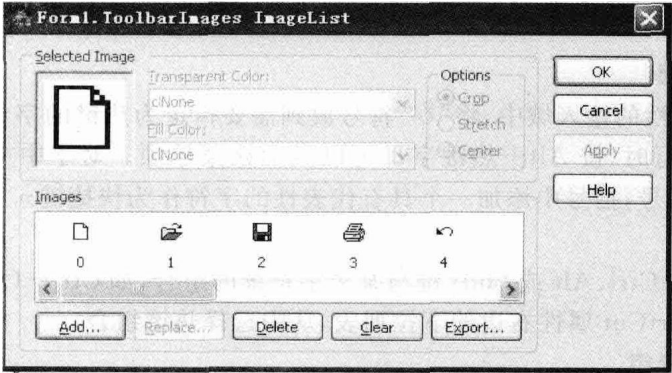


图 6-4 ImageList 增加位图对话框

用鼠标双击 MainMenu1 组件,进入菜单编辑器窗口,设计图 6-5 所示的各个菜单项。



图 6-5 MenuItem 各个菜单项详细设计界面

一级菜单项含 6 个菜单,分别为 File、“显示隐藏”、“编辑”、“编辑菜单项语言”、“工具条”和“退出”。设计好的界面如图 6-6 所示。

这 6 个菜单项的各菜单内容详细设计如图 6-5 所示,通过对象查看器进行编辑设置其属性值。

File 下拉菜单有 New、Open、Save、Save As 和 Print 这 5 项。它们被分成两个组,除 Save As 之外都定义了快捷键,如 New 的快捷键是 Ctrl+N 等。为各个菜单项设置合适的 Name 属性,如图 6-5 所示;为各个菜单项的 ImageIndex 属性通过对象查看器设置合适的图片,如图 6-7 所示。其他菜单组属性设置类似,就不再阐述。

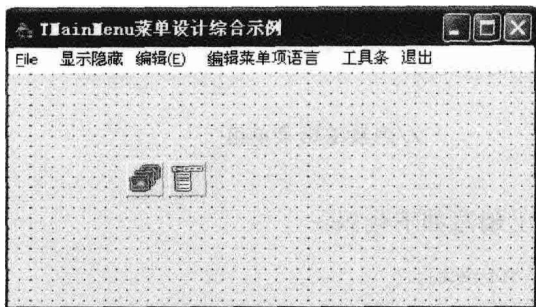


图 6-6 MenuItem 菜单设计界面

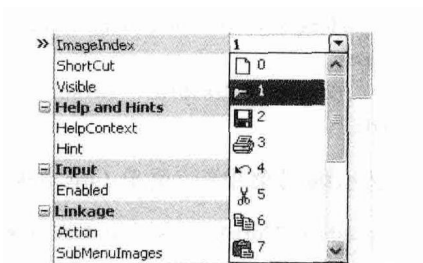


图 6-7 MenuItem 菜单项设置图片序号

### (3) 菜单项功能设计并编写代码。

本例中该菜单系统能实现以下的功能：

① 当选择 New、Open、Save、Save As 或者 Print 菜单项时，出现一个对话框提示相应的功能还没有完成。

② 当选择“撤销”、“剪切”、“复制”、“粘贴”和“字体”菜单项时出现同样提示。

选择 New 菜单项，在其 OnClick 事件上编写如下代码：

```
procedure TForm1.FileNewItemClick(Sender: TObject);
begin
    //调用通用函数 ShowMessage 输出信息
    ShowMessage (TMenuItem(Sender).Caption + '菜单功能还没有完成!');
    //Sender 参数表示发送事件消息的对象
end;
```

用鼠标同时选中 Save、Save As 和 Print 菜单项，在 Object Inspector 事件页的 OnClick 事件中选择 FileNewItemClick 事件处理程序。这样可以减少编程的工作量。

用鼠标同时选中“撤销”、“剪切”、“复制”、“粘贴”和“字体”菜单项，在 Object Inspector 事件页的 OnClick 事件中选择 FileNewItemClick 事件处理程序。

③ 当选择“显示隐藏”菜单时自动刷新“显示编辑菜单组”和“隐藏编辑菜单组”的使能状态。主要是根据“编辑菜单组”当前是否为可见状态来确定。

“显示隐藏”菜单的 OnClick 事件上编写如下代码：

```
procedure TForm1.View1Click(Sender: TObject);
begin
    //控制菜单项状态过程
    if EditMenu.Visible then                                     //如果编辑菜单可见
    begin
        ViewShowEdit.Enabled := False;                         //显示命令失效
        ViewHideEdit.Enabled := True;                           //隐藏命令有效
    end
    else
    begin
        ViewShowEdit.Enabled := True;                           //显示命令有效
        ViewHideEdit.Enabled := False;                          //隐藏命令失效
    end;
end;
```



“隐藏编辑菜单组”菜单的 OnClick 事件上编写如下代码：

```
procedure TForm1.ViewHideEditClick(Sender: TObject);
begin
    EditMenu.Visible := False;           //隐藏编辑菜单组
end;
```

“显示编辑菜单组”菜单 OnClick 的事件上编写如下代码：

```
procedure TForm1.ViewShowEditClick(Sender: TObject);
begin
    EditMenu.Visible := True;           //显示编辑菜单组
end;
```

④ 下面编写语言切换功能代码。

当执行“中文”时，编辑菜单组的 Caption 全部显示中文，并且“中文”处于选中状态。其 OnClick 事件上编写如下代码：

```
procedure TForm1.LangCHClick(Sender: TObject);
begin
    //显示菜单选中状态
    LangCH.Checked := true;
    LangEN.Checked := False;
    //将编辑组菜单项的标签显示成中文
    EditMenu.Caption := '编辑(&E)';
    EditUndoItem.Caption := '撤销';
    EditCutItem.Caption := '剪切';
    EditCopyItem.Caption := '复制';
    EditPasteItem.Caption := '粘贴';
    miEditFont.Caption := '字体 ...';
end;
```

当执行“英文”时，编辑菜单组的 Caption 全部显示英文，并且“英文”处于选中状态。其 OnClick 事件上编写如下代码：

```
procedure TForm1.LangENClick(Sender: TObject);
begin
    //显示菜单选中状态
    LangCH.Checked := False;
    LangEN.Checked := True;
    //将编辑组菜单项的标签显示成英文
    EditMenu.Caption := '&Edit';
    EditUndoItem.Caption := 'Undo';
    EditCutItem.Caption := 'Cut';
    EditCopyItem.Caption := 'Copy';
    EditPasteItem.Caption := 'Paste';
    miEditFont.Caption := 'Font ...';
end;
```

⑤ 当选择“退出”时能关闭程序。其命令代码如下：

```
procedure TForm1.ExitClick(Sender: TObject);
```

```
begin
    Form1.Close;                                //调用窗体方法关闭系统
end;
```

(4) 保存并运行程序。

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,对照上述菜单功能一一验证。当执行“英文”命令时,编辑菜单组的 Caption 全部显示成英文,其效果如图 6-8 所示。

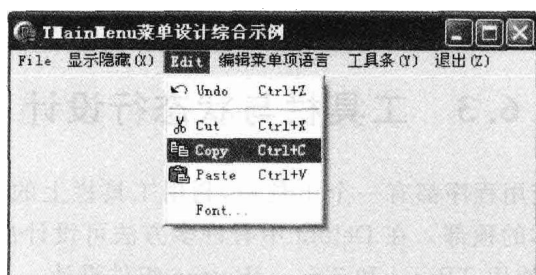


图 6-8 MenuItem 菜单项控制运行效果

## 6.2 鼠标右键弹出式菜单

应用程序中的鼠标右键弹出式菜单可以方便用户的操作,灵活性也很好。可以给窗体添加弹出式菜单,也可以给具体的某个控件添加弹出式菜单。另外,窗体中的某几个组件可以共用一个弹出式菜单,只要将这几个组件的 PopupMenu 属性设置成同一个弹出式菜单的名称就可以了。

### 6.2.1 TPopupMenu 组件

#### 1. 概述

TPopupMenu 组件位于组件面板的 Standard 选项卡上(如图 6-1 所示),用来建立快捷菜单。所谓快捷菜单,通常是具有常用功能的菜单,通过单击鼠标右键来激活快捷菜单。

#### 2. 主要属性

(1) AutoPopup 属性:确定当右击时,菜单是否出现。其默认值为 true,当右击后出现菜单,若为 false 时,右击后菜单不出现。

(2) Handle 属性:允许用户访问表示 HMENU 对象实例的 4 字节标识符。

(3) HelpContext 属性:提供一个用于联机帮助的文本号。

(4) Items 属性:是菜单项的描述,使用它可以访问关于菜单项的信息,菜单项属于 TMenuItem 类,其详细内容参考 6.1.2。

(5) PopupComponent 属性:指出响应右击显示快捷菜单的组件名字。

**注意:**当用户设计好一个菜单后,并不代表在程序执行时就可以使用,因为窗体(Form)并不知道还有快捷菜单,为此还要设置窗体的 PopupMenu 属性。单击 PopupMenu 属性右边的下三角按钮,选择窗体中所创建快捷菜单的名字如 PopupMenu1,这样在程序运行时,当右击窗体后,便会弹出相应的菜单。

## 6.2.2 鼠标右键弹出式菜单设计

设计快捷菜单与设计主菜单类似,只是它只能有一个菜单(其中的项可以有子菜单)。选择组件面板的 Standard 选项卡上 TPopupMenu 组件,添加到窗体中,双击 TPopupMenu 组件弹出快捷菜单设计器。该窗口的菜单项设计与 MainMenuDesigner 完全相同。

如果要在某个窗体或组件上实现单击鼠标右键弹出菜单,可以在该窗体或组件上添加弹出式菜单,设计完毕后,还要在窗体或组件的 PopupMenu 属性中选中该弹出式菜单。对于部分可视组件来说,不能设置 PopupMenu 属性。

## 6.3 工具栏与状态行设计

大部分 Windows 应用程序都有一个工具栏,利用工具栏上的图形按钮可加快操作速度,一般工具栏位于窗体的顶部。在 Delphi 中有许多方法可设计出不同风格的工具栏,其中较简单直接的方法是使用 TPanel 和 TSpeedButton 组件设计。

### 6.3.1 TPanel 组件

TPanel(面板)组件位于组件面板的 Standard 页上倒数第二个位置,它是一个容器组件,即可以将其他组件放入面板。它的主要功能就是制作工具栏和状态栏。以下是 TPanel 面板组件的常用属性。

(1) Align 属性:设置面板在窗体上的位置。该属性取值范围为 alTop、alRight、alBottom、alLeft、alClient 和 alNone。

默认值为 alNone,此时可用鼠标操作确定其位置和尺寸。当作为工具栏使用时,面板常被放置在窗体顶部菜单条之下,可以用 alTop 指定该位置。这样,即使窗体尺寸改变,面板也能准确定位,而且能自动改变宽度以保持横贯整个窗体。alRight、alLeft、alBottom 和 alTop 类似,分别适合将面板定位到窗体的右、左和底部。alClient 则定位在整个窗体的客户区。

(2) BevelInner 和 BevelOuter 属性:BevelInner 和 BevelOuter 这两个属性决定了面板的外观,适当搭配二者的值可以产生不同的三维效果。它们的取值范围为 bvLowered、bvNone、bvRaised 和 bvSpace。默认值 BevelInner 为 bvNone,BevelOuter 为 Raised。

(3) Caption 属性:面板也有 Caption 属性,因此可在面板上显示文字。此时,它的功能类似于 TLabel 组件,但可以制作出三维效果,简单的状态栏就是利用面板的 Caption 制作的。在用作工具栏时,一般应将 Caption 置为空白。

(4) Visible 属性:为 false 时,可隐藏面板及面板上的组件。可用来隐藏工具栏。

(5) Color 属性:根据需要可以给面板设置各种颜色,可以直接输入颜色数值,也可以根据颜色对话框设置。

### 6.3.2 TSpeedButton 组件

TSpeedButton(加速按钮)组件是位于 Additional 页上第二个位置的组件,如图 6-9 所示。它与按钮类似,但可以在表面上放置图形。与按钮(TButton)或位图按钮(TBitBtn)相

比,它没有窗口句柄,不消耗 Windows 资源,绘制速度也更快一点。所以,非常适合用作工具栏上的工具按钮。加速按钮有以下主要属性。

(1) Glyph 属性:指定一个图像文件放到加速按钮的表面。单击 Object Inspector 中该属性右侧的“...”按钮打开 Picture Editor 对话框,单击 Load 按钮指定一个图片文件,然后单击 OK 按钮完成装入,如图 6-10 所示。一般应装入小型位图或图标(20×20 像素左右),位图文件既可通过各种途径收集得到,也可利用专用绘图工具软件自行绘制。在 Delphi 2007 的目录 C:\Program Files\Common Files\CodeGear Shared\Images\Buttons 下就有许多按钮图标。

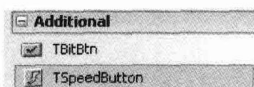


图 6-9 TSpeedButton 组件

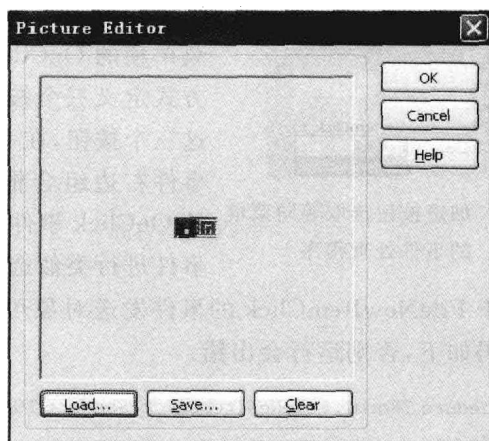


图 6-10 Picture Editor 对话框

(2) NumGlyphs 属性:可输入数字 1~4,默认值为 1,表示加速按钮中包含的位图数目。当该数目为 2 时,第一个位图为按钮的标准状态,第二个位图用于按钮失效时。当该数目为 4 时,第一、二个位图作用同前,第三个位图用于单击按钮时,第四个位图用于按钮持续保持按下状态时。前面提到 Buttons 目录下有许多按钮的位图,如果打开这些位图观察可以发现它们都是“两位一体”的,即图片中横向并排着两个同样尺寸的类似图形。

Delphi 2007 的目录 C:\Program Files\Common Files\CodeGear Shared\Images\Buttons 下许多按钮图标就是专门被定制成可以用作 NumGlyphs 的值为 2 时的按钮表面贴图。

类似地,如果用于 NumGlyphs 的值为 4 时的贴图,图片中必须包含从左到右排列的 4 个同样尺寸的小图形。

(3) Hint 和 ShowHint 属性:用于产生提示,这两个属性并非加速按钮专有。但由于仅靠一幅简单的位图难以表达复杂的意义,加速按钮更加需要有自我提示的功能。Hint 是字符串类型的属性,应输入提示的内容;ShowHint 则是 boolean 型,当它为 true 时组件具有提示功能。

### 6.3.3 工具栏设计举例

**例 6.2** 使用 TPanel 和 TSpeedButton 组件设计工具栏综合示例。

设计构想:在例 6.1 的基础上,保留主菜单上的 File、“工具条”和“退出”三个下拉菜

单,删去其余菜单项,并删除它们的关联事件代码。

往该程序窗体内放入一个由面板 Panel1 和 4 个加速按钮组成的工具栏。Panel1 的 Align 设置为 alTop; 4 个加速按钮的 graph 中分别装入 4 个位图,它们的 ShowHint 都设置为 true, Hint 属性分别输入 New、Open、Save 和“退出”。

再将菜单项“工具条显示”的 Checked 属性设置为 true,表示程序在初始状态下显示工具栏。各菜单项的 OnClick 基本上与例 6.1 中相同,但“工具条显示”的 OnClick 事件中应增加改变 Panel1 的 Visible 属性的语句。

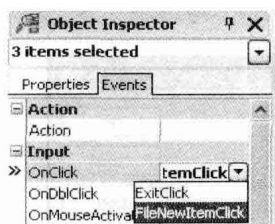


图 6-11 加速按钮选取等同菜单的事件处理程序

程序中,工具栏上 4 个加速按钮的功能应分别等同于菜单上的 New、Open、Save 和“退出”,即它们可以分别与 4 个菜单项的 OnClick 事件分享代码。如图 6-11 所示,可按如下方式定义三个按钮的 OnClick 事件:选中 New、Open、Save 这三个按钮,在 Object Inspector 中选 Events 页,在 OnClick 事件右边组合框中选择 FileNewItemClick(即菜单项 New 的 OnClick 事件的名称),然后对“退出”加速按钮的 OnClick 事件进行类似选取 ExitClick 处理程序。

由于 FileNewItemClick 的事件发送对象可能是菜单项,也可能是命令对象,因此要修改其代码如下,否则运行会出错。

```
procedure TForm1.FileNewItemClick(Sender: TObject);
begin
    //调用通用函数 ShowMessage 输出信息
    if Sender is TMenuItem then //如果是菜单项命令
        ShowMessage (TMenuItem(Sender).Caption + '菜单功能还没有完成!')
    else //反之是加速按钮发送的命令消息
        ShowMessage (TSpeedButton(Sender).Name + '功能还没有完成!');
    //Sender 参数表示发送事件消息的对象
end;
```

菜单中,“工具条显示”可以显示或者隐藏窗体的工具栏,并且把工具栏的当前状态在该菜单项的 Checked 属性值上体现出来。

下面是该示例为“工具条显示”的 OnClick 事件编写的代码:

```
procedure TForm1.ToolBarShowItemClick(Sender: TObject);
begin
    //菜单项的选中状态切换
    ToolBarShowItem.Checked := not ToolBarShowItem.Checked;
    //刷新状态是否需要显示
    if ToolBarShowItem.Checked then
        Panel1.Visible := true
    else
        Panel1.Visible := False;
end;
```

程序运行开始时的画面如图 6-12 所示。

若把鼠标靠近加速按钮并停留,就会看到提示信息。选择“工具条显示”菜单项,能打开

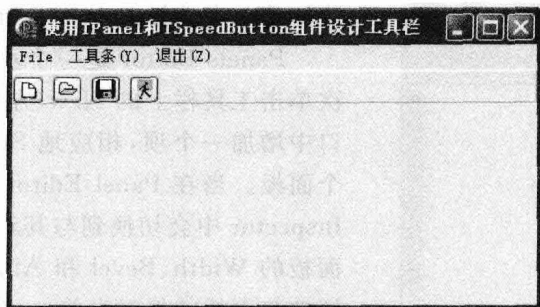


图 6-12 工具栏综合示例初始运行画面

或关闭工具栏。当单击三个加速按钮时,与选择菜单项 New、Open 和 Save 时一样,会出现一个显示“功能还没有完成!”的信息框。

实际工程系统中,可以把组合框、标签、微调按钮(UpDown)等控件放到面板组件上,做出一个像 Word 中那样有更加复杂功能的工具栏,如图 6-13 所示。

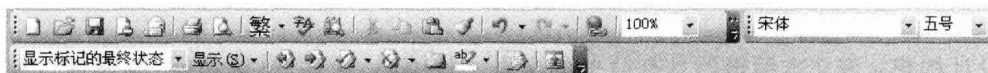


图 6-13 功能复杂的工具栏

#### 6.3.4 TStatusBar 组件

通常在标准的 Windows 应用程序的底部都能看到一个状态栏,用以提供一些系统信息和提示。Delphi 中可以利用面板组件来间接地制作状态行,但效果较差,过程比较烦琐。这里介绍 Delphi 系统自带的非常适合用作状态栏的 TStatusBar 组件,它位于 Win32 页上,如图 6-14 所示。

StatusBar 可以看做是面板的容器,它包含有若干面板 TPanel,从左向右排列。其中每个面板都能被独立控制,用于显示一项信息。图 6-15 为含有多个面板的 TStatusBar 组件。

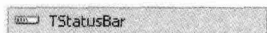


图 6-14 位于 Win32 页的 TStatusBar 组件



图 6-15 含有多个面板的 TStatusBar 组件

下面简单介绍一下 TStatusBar 组件的主要属性。

(1) Align 属性:用法与面板组件 TPanel 的 Align 一样,但作为状态行组件,其默认值被设置为 alBottom。

(2) Font 属性:可设置状态行中使用的字体,但必须同时设置 UseSystemFont 属性值为 false,否则状态行使用系统定义的字体。

(3) SimplePanel 属性:为 true 时,StatusBar 具有一个简单面板(Panel)的一切特征,而不能作为面板容器使用。一般应设置 SimplePanel 为 false。

(4) Panels 属性: TStatusBar 最主要的属性,它是一个数组,其中的每个元素代表一个面板。当 SimplePanel 为 false 时,可单击 Panels 属性值右侧的“...”按钮打开一个面板设置对话框,如图 6-16 所示。打开该对话框的另一个办法是右击 StatusBar 组件对象,在弹出的

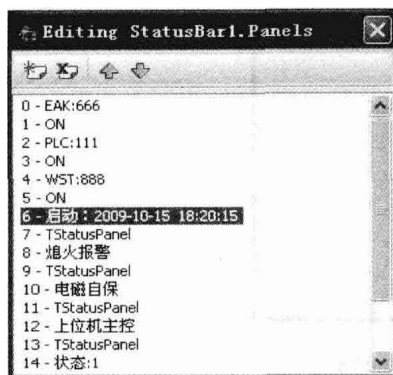


图 6-16 使用 Panels Editor 操作状态栏显示信息

快捷菜单中选择 Panels Editor 命令。

Panels Editor 是一个小窗口,开始时为空,每次单击工具栏上的 New(左面第一个)按钮就在窗口中增加一个项,相应地 StatusBar 中也增加了一个面板。当在 Panel Editor 中选中某项时,Object Inspector 中会切换到与其对应的面板,就能设置该面板的 Width、Bevel 和 Alignment 等属性。Bevel 属性与面板的外观有关,一个状态行组件的各个面板可分别有不同的 Bevel 值。Alignment 的可选值为 taCenter、taLeftJustify 和 taRightJustify,分别表示显示文字的位置为居中、靠左和靠右。

### 6.3.5 状态栏设计举例

**例 6.3** 使用 TMemo、TStatusBar 以及 TMainMenu 等组件设计一个最简单的文本编辑器。

设计构想:该文本编辑器具有简单的文本文件编辑、保存、浏览功能,能够进行和操作系统剪贴板进行数据交换。文本编辑器下面显示的状态包含:

- (1) 时间,显示为 XX:XX:XX;
- (2) Memo 控件中当前插入点的位置,它用 Y 行、X 列形式表示,Y、X 起始值都是 0;
- (3) Memo 中文本的字数;
- (4) 键盘上 Insert 键的状态,显示为 INS 或空白;
- (5) 键盘上 NumLock 键的状态,显示为 NUM 或空白。

详细实现步骤如下:

- (1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。

- (2) 按照设计构想定制窗体各个对象的属性。

① 修改 Form1 的 Caption 值为“简单的文本编辑器”。在 Form1 窗体上放置一个主菜单组件 MainMenu1,并用鼠标双击它进入菜单编辑器,设置两个下拉菜单项分别为“文件”和“编辑”,其中“文件”菜单下包含“新建”、“打开”和“保存”这三个菜单项;“编辑”菜单下包含“复制”、“剪切”和“粘贴”。为了程序代码的可读性,将这些菜单项的 Name 属性设置成合适的名称。

② 为了使得本程序的文件打开和保存有一个友好的界面,特使用两个对话框组件放置在 Form1 窗体上,这两个都是位于 Dialog 页面的 TOpenDialog 和 TSaveDialog 组件,本书将在 6.5 节详细介绍这两个组件的详细用法。

- ③ 窗体中间放了一个 TMemo 控件,用于输入文本,将该组件的显示内容清空。

④ 在窗体底部放置一个用于显示状态栏 TStatusBar 组件,并通过对话框设置产生 5 个面板,并调整其合适的宽度。

- ⑤ 在 Form1 窗体上放置一个 TTimer 组件(位于 Delphi 组件面板的 System 页面),目

的是定时刷新编辑器状态参数到状态栏上,设置其定时间隔属性(Interval)为 200ms。  
将各组件属性设置好并调整组件位置及大小后,窗体界面如图 6-17 所示。

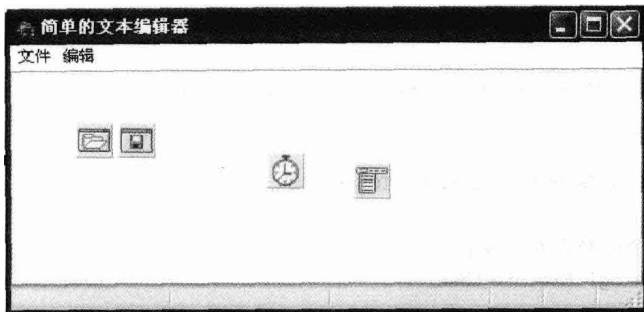


图 6-17 设计窗体界面

### (3) 编写代码。

为 Timer1 编写 OnTimer 事件的代码如下:

```
procedure TForm1.Timer1Timer(Sender: TObject);
var Str:String;
    DT:TDateTime;
begin
    DT := Time;
    StatusBar1.Panels.Items[0].Text := '时间:' + TimeToStr (DT); //定义一个字符串类型的变量
                                                                //定义一个日期时间型的变量
                                                                //获取系统当前的时间
                                                                //显示当前时间
                                                                //显示文本编辑框插入点位置

    str := '位置: ';
    str := str + IntToStr (Memo1.CaretPos.y) + '行';
    str := str + IntToStr (Memo1.CaretPos.x) + '列';
    StatusBar1.Panels.Items[1].Text := str;
    //显示字数
    str := Memo1.Lines.Text;
    StatusBar1.Panels.Items[2].Text := '字数:' + IntToStr (Length(str));
    //显示 NUM 状态
    if (GetKeyState(VK_NUMLOCK)and $ 01) = $ 01 then
        StatusBar1.Panels.Items[3].Text := 'NUM'
    else
        StatusBar1.Panels.Items[3].Text := '  ';
    //显示 INS 状态
    if (GetKeyState(VK_INSERT)and $ 01) = $ 01 then
        StatusBar1.Panels.Items[4].Text := 'INS'
    else
        StatusBar1.Panels.Items[4].Text := '  ';
end;
//说明: GetKeyState 为 Windows 接口函数,主要是取按键的状态
```

为各个菜单项编写 OnClick 事件的依次代码如下:

```
procedure TForm1.N_NewClick(Sender: TObject);
begin
    Memo1.Clear;
end;
//清除文本框
```



```

procedure TForm1.N_OpenClick(Sender: TObject);
begin
    //打开一个已经存在的文件
    if OpenFileDialog1.Execute then
    begin
        Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
    end;
end;
procedure TForm1.N_SaveClick(Sender: TObject);
begin
    //打开一个已经存在的文件
    if SaveDialog1.Execute then
    begin
        Memo1.Lines.SaveToFile(SaveDialog1.FileName);
    end;
end;
procedure TForm1.N_CutClick(Sender: TObject);
begin
    Memo1.CutToClipboard;                                     //剪切
end;
procedure TForm1.N_CopyClick(Sender: TObject);
begin
    Memo1.CopyToClipboard;                                   //复制
end;
procedure TForm1.N_PasteClick(Sender: TObject);
begin
    Memo1.PasteFromClipboard;                                //粘贴
end;
end;

```

#### (4) 保存并运行程序。

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,对照上述菜单功能一一验证。当执行“打开”命令时,出现一个对话框,选择一个文本文件装入到编辑框中。状态栏中可以实时显示编辑状态,其效果如图 6-18 所示。

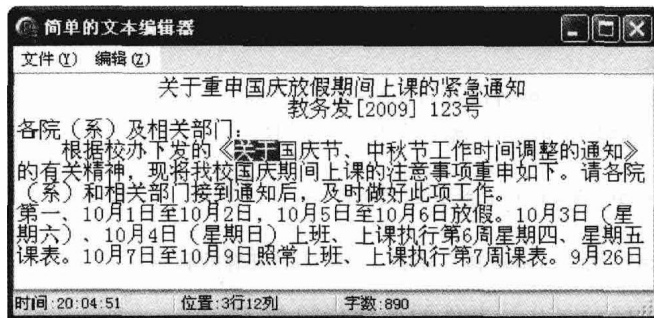


图 6-18 MenuItem 菜单项控制运行效果

本例中 GetKeyState 是一个 API 函数,可取到键盘上各个键的状态。使用了 Timer 组件,该组件起到定时控制的作用。它的 Interval 属性指定时间间隔,本例中设置为 200,表示

每间隔 200ms(0.2s)调用一次时控过程(由 OnTimer 事件确定)。

**注意:** 设置 Timer 的 Enabled 为 true,以启动 Timer 的实时控制功能。

## 6.4 对话框函数

对话框的界面设计重点在于应满足用户应用程序中信息提示或者所需数据的直观查找,相应用户数据的输入和参数的选择处理等。

对话框一般分为两种类型:模态类型(modal)与非模态类型(modeless)。所谓模态对话框,就是指除非采取有效的关闭手段,否则用户的鼠标焦点或者输入光标将一直停留在其上的对话框。非模态对话框则不会强制此种特性,用户可以在当前对话框以及其他窗口间进行切换。本文介绍如何使用 Delphi 语言组件或者标准过程来创建这两种类型的对话框、控制其大小和位置、改变其外观以及在对话框间的数据传递。

### 6.4.1 对话框与模态窗口

设计对话框与设计普通的窗体没什么本质区别,但对话框一般具有如下特征:

(1) 在程序中不作为主窗体,通常被默认命名为 Form2 或 Form3 等。在 Delphi 中为了给应用程序加入第二个窗体(或第三个窗体等),可选择 File→New→Form 命令或单击相应的工具按钮。

对于有两个以上(含两个)窗体的程序,在与主窗体 Form1 对应的 Unit1.pas 中一般要加入 uses Unit2.pas 语句,否则在 Form1 中就不能识别 Form2 中定义的对象。至于是否要在 unit2.pas 中加入 uses Unit1.pas,则应看具体情况。

(2) 窗口标题一般应为对话框名称,窗口内没有主菜单。边框上无控制按钮,也不能改变窗口尺寸。要做到这一点,较简单的方法是设置 BorderStyle 属性为 bsDialog。

(3) 窗口往往以模态方式打开,Windows 的窗口可定义为模态窗口或非模态窗口。当应用程序打开一个模态窗口后,只要该窗口未关闭,就不能对程序中的其他窗口进行任何操作,以此方式强制用户对该模态窗口作出响应。所谓以模态方式打开,即指使打开的窗口成为模态窗口。

在 Delphi 中一个窗口是否为模态一般并非取决于其设计阶段,而是由打开该窗口时使用的方法所确定。如果一个窗口是被 Show 方法打开的,那么它就是非模态方式的;如果用 ShowModal 方法打开窗口,那么该窗口就是模态的。

### 6.4.2 Delphi 对话框函数

对话框是 Windows 操作系统中程序与用户沟通的一种常见的交互方式,对话框可以向用户提供当前程序的运行状况,也可以接受用户输入的信息。在 Delphi 中,对话框函数大体上可以分为两种:输入对话框函数和输出对话框函数。

输入对话框函数用于接收用户在程序运行过程中输入的信息,其中包括 InputBox 函数;而输出对话框函数则用于显示一个对话框窗体和向用户报告当前程序的运行状态等信息,它包括 ShowMessage 函数和 MessageDlg 函数。这些函数产生的对话框都是模态的。

下面首先准备新建一个应用程序,在窗体 Form1 上放置一个 Edit1 组件,用于接受用户信息的输入,一个 Label1 作为输入标签,然后放置用于驱动产生三个模态信息框的三个 TButton 组件,并更改它们的相关属性。设计好的准备界面如图 6-19 所示,完整详细设计与代码参考本书配套学习资源源文件。

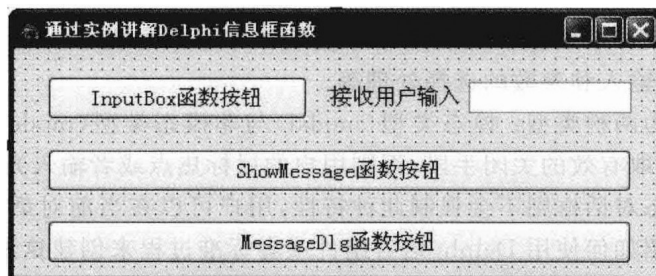


图 6-19 消息框演示程序界面

下面对各个函数通过示例分别加以介绍。

### 1. InputBox 函数

功能描述:对话框函数中的 InputBox 函数用于在程序运行的过程中显示包含一个字符串和按钮信息的输入对话框,对话框执行完毕可以为用户返回一个字符串。

语法结构如下所示:

```
function InputBox(const ACaption, APrompt, ADefault: string): string;
```

本例中 But\_InputBox 按钮的 OnClick 事件使用 InputBox 函数,代码如下:

```
procedure TForm1.But_InputBoxClick(Sender: TObject);
var TempStr:string;                                     //定义字符串变量
begin
    //显示一个输入对话框,将输入的数据传给 TempStr 变量
    TempStr := InputBox('某工程系统计算','请输入原始数据','');
    //显示输入数据的平方值
    Edit1.Text := TempStr;
end;
```

运行的对话框效果界面如图 6-20 所示。

### 2. ShowMessage 函数

功能描述:对话框函数中的 ShowMessage 函数用于在程序运行的过程中显示包含一个字符串信息的对话框。该对话框没有返回数据。

语法结构如下所示:

```
ShowMessage(const Msg:string);
```

本例中 But\_ShowMessage 按钮的 OnClick 事件使用 ShowMessage 函数,代码如下:

```
procedure TForm1.But_ShowMessageClick(Sender: TObject);
var TempStr:string;                                     //字符串变量
```

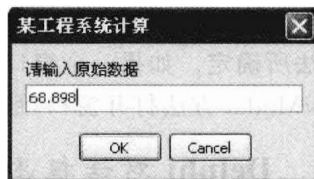


图 6-20 InputBox 函数运行效果界面

```

begin
    //准备消息内容,只要最终是字符串类型就行
    TempStr := FormatDateTime('yyyy-mm-dd' + ' hh:mm:ss AM/PM', NOW) + ': 欢迎进入 Delphi 世界!';
    //调用函数,显示一个输出对话框
    ShowMessage(TempStr);
end;

```

运行的对话框效果界面如图 6-21 所示。

### 3. MessageDlg 函数

功能描述：对话框函数中的 MessageDlg 函数用于在程序运行的过程中显示包含一个字符串、位图和按钮信息的对话框，对话框执行完毕可以为用户返回一个无符号整型(WORD)的数据。

语法结构如下所示：

```

function MessageDlg(const Msg: string; AType: TMsgDlgType; AButtons: TMsgDlgButtons; HelpCtx:
Longint): Word;

```

本例中 But\_MessageDlg 按钮的 OnClick 事件使用 MessageDlg 函数,代码如下：

```

procedure TForm1.But_MessageDlgClick(Sender: TObject);
begin
    //显示一个问号和 Yes、No 两个按钮的输出对话框
    if MessageDlg('选择对话框：你想退出本演示程序吗?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
        //如果单击 yes 按钮就显示一个感叹号的输出对话框
        MessageDlg('信息提示：按下 OK 表示退出!', mtInformation,
            [mbOk], 0);
        Form1.Close;
    end;
end;

```

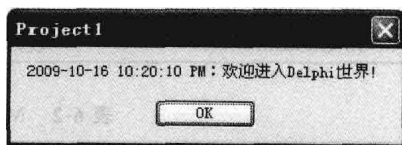


图 6-21 ShowMessage 函数运行效果界面

运行后出现第一个对话框，单击 Yes 按钮出现第二个信息提示对话框，显示效果界面如图 6-22 所示。

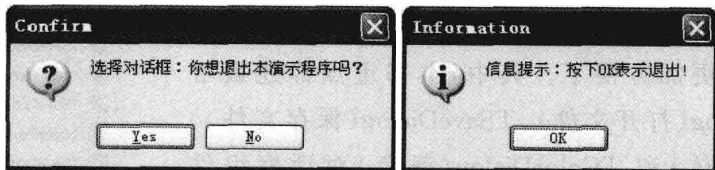


图 6-22 MessageDlg 函数运行效果界面

在本例中 MessageDlg 函数使用到的参数 Atype 是 TmsgDlgType 类型，它的取值范围及含义如表 6-1 所示。

而 MessageDlg 函数中的参数 Abuttons 是 TmsgDlgButtons 类型，它的取值范围及含义如表 6-2 所示。

表 6-1 MessageDlg 函数中 Atype 参数的取值

参 数 取 值	含 义 说 明
mtWarning	表示警告：显示一个带有蓝色惊叹号的信息框
mtError	表示错误提示：显示一个带有红色停止符号的信息框
mtInformation	表示信息提示：显示一个带有蓝色“i”标志的信息框
mtConfirmation	表示信息确认：显示一个带有绿色问号的信息框
mtCustom	表示用户定制：框中没有位图，标题为应用程序文件名

表 6-2 MessageDlg 函数中 Abuttons 参数的取值

参 数 取 值	含 义 说 明
mbYes	表示对话框中含有 Yes 按钮，含义：是
mbNo	表示对话框中含有 No 按钮，含义：不
mbOK	表示对话框中含有 OK 按钮，含义：可以
mbCancel	表示对话框中含有 Cancel 按钮，含义：取消
mbHelp	表示对话框中含有 Help 按钮，含义：帮助
mbAbort	表示对话框中含有 Abort 按钮，含义：放弃
mbRetry	表示对话框中含有 Retry 按钮，含义：重试
mbIgnore	表示对话框中含有 Ignore 按钮，含义：不理睬
mbAll	表示对话框中含有 All 按钮，含义：全部

读者可以参照上面的代码将这两个参数按照表格中的取值修改来体会一下。

总结：Delphi 中使用 ShowMessage 和 MessageDlg 函数可以产生标准的 Windows 信息框，这种信息框是一个模态窗口。InputBox 函数用于人机数据信息交互，不太常用。ShowMessage 使用方便但功能较弱，只能显示一个带有 OK 按钮的信息框。MessageDlg 函数适用范围更广。

6.5 对话框组件

在 Delphi 组件面板的 Dialog 页上大约有 10 多个通用对话框组件可供使用，如图 6-23 所示。这些对话框实际上是由 Windows API 函数提供的，Delphi 把它们包装了一下，使用更加方便了。其中，本书重点详述最常用的 TOpenDialog(打开文件)、TSaveDialog(保存文件)、TFontDialog(字体)和 TColorDialog(颜色)对话框组件等。另外几个是 TOpenPictureDialog(打开图形)、TSavePictureDialog(保存图形)、TPrintDialog(打印)、TPrinterSetupDialog(打印机设置)、TFindDialog(查找)和 TReplaceDialog(替换)对话框组件，操作和使用比较类似。

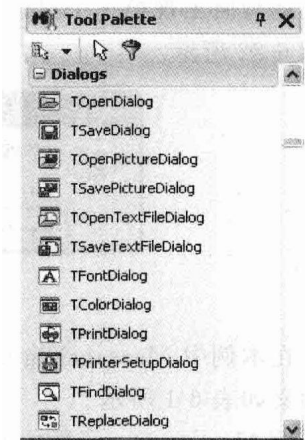


图 6-23 位于 Dialog 页面的对话框组件

### 6.5.1 用于文件选择的 TOpenDialog 对话框组件

TOpenDialog(打开对话框)组件用于显示一个“文件选择”对话框。它显示时,在一个列表框中显示当前目录下的所有文件名,用户通过鼠标或键盘指定其中一个文件名,单击“打开”按钮,就完成了选择文件的操作。

#### 1. TOpenDialog 组件概述

“打开”对话框是用 TOpenDialog 组件实现的,TOpenDialog 组件是非可视组件。

Filter 属性用于设置文件过滤器,让对话框只列出特定类型的文件。在设计时可以单击 Filter 属性旁的“...”按钮,打开 Filter Editor 对话框,在 Filter Name 文本框中输入关于过滤器的简短说明,在 Filter 文本框中输入通配符。在设计期,把“打开”对话框组件加到窗体上,看到的是 TOpenDialog 组件的图标,只有调用对话框的 Execute 方法,才能在运行期看到真正的对话框,示例代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if OpenDialog1.Execute then                //打开对话框执行
    begin
                                                //将选择打开的文件名放置到 Caption 上
        Form1.Caption := OpenDialog1.FileName;
    end;
end;
```

一个运行的 OpenDialog 对话框如图 6-24 所示。



图 6-24 一个运行的 OpenDialog 对话框

#### 2. TOpenDialog 组件的重要属性

(1) DefaultExt 属性:用于指定一个默认的扩展名。如果用户在对话框中选择的文件没有带扩展名;就用这个属性作为文件的扩展名;如果不需要默认扩展名,应当把这个属性设为空。

(2) FileName 属性:可以指定一个文件名。在打开对话框时,这个文件名出现在对话框的“文件名”下拉列表框中。在对话框中,可以指定其他文件名,如果单击 OK 按钮,对话

框将关闭,并且 FileName 属性变成用户新指定的文件名,包括文件的路径和扩展名。FileName 属性也可以指定一个不存在的文件名,表示将打开一个新文件。

(3) Files 属性:如果 Options 属性包含 ofAllowMultiSelect 元素,表示允许一次选择多个文件,这些文件可以通过 Files 属性访问。Files 属性是一个 TStrings 对象。

(4) Filter 属性:用于设置文件过滤器,让对话框只列出特定类型的文件。在设计时,可以单击 Filter 属性旁的“...”按钮,打开 Filter Editor 对话框,在 Filter Name 文本框中输入关于过滤器的简短说明,在 Filter 文本框中输入通配符,例如通配符 \*.txt 表示让对话框只显示扩展名为.txt 的文件。当然,用户可以在“打开”对话框的“文件名”下拉列表框内直接输入一个扩展名不为.txt 的文件。同时,Filter 属性可以设置多个过滤器,这些过滤器将显示在“打开”对话框的“文件类型”下拉列表内。

(5) FilterIndex 属性:如果有多个过滤器,这个属性用于指定哪个过滤器是默认过滤器。默认过滤器将显示在“打开”对话框的“文件类型”下拉列表中。注意,过滤器的序号从 1 开始。

(6) InitialDir 属性:用于设置“打开”对话框第 1 次打开时的默认打开目录。例如要将“打开”对话框的默认打开目录设置为 C 盘根目录,就可以将 InitialDir 属性设置为“C:\”。

(7) Options 属性:用于设置“打开”对话框的选项。它设定了如下 16 个用户可选值。

- ofAllowMultiSelect:用户可以一次选择多个文件。
- ofCreatePrompt:如果用户输入的文件名是不存在的,当用户单击 OK 按钮时,将显示一个提示框,询问要不要建立这个文件。
- ofExtensionDifferent:如果用户所选文件的扩展名与 DefaultExt 属性设置的默认扩展名不同,就包含这个元素。
- ofFileMustExist:用户输入的文件名必须是已存在的。
- ofHideReadOnly:对话框中不显示“以只读方式打开”复选框。
- ofNoChangeDir:即使用户在对话框中选择了其他目录,但对程序来说,当前目录总是对话框第一次打开时的目录。
- ofNoDereferenceLinks:如果用户选择的是快捷方式文件(.lnk),FileName 属性是快捷方式文件本身,而不是快捷方式文件指向的文件。
- ofNoLongNames:不允许长文件名。
- ofNoReadOnlyReturn:不允许用户选择只读的文件,否则将显示一个警告框。
- ofNoTestFileCreate:保存文件时不进行写保护、磁盘满、驱动器门打开等检查。
- ofNoValidate:不对文件名中的字符进行合法性检查。
- ofOverwritePrompt:用于“另存为”对话框中,如果用户指定的文件名已存在,将显示一个警告框让用户选择是否要覆盖已有的文件。
- ofPathMustExist:用户输入的路径必须是已经存在的,否则将显示警告。
- ofReadOnly:选中“以只读方式打开”复选框。
- ofShareAware:对话框不理睬所有的共享错误。
- ofShowHelp:对话框中将显示“帮助”按钮。

(8) Title 属性:用于设置对话框显示时的窗口标题。例如大多数数据库程序都有还原数据库功能,其“打开”对话框的标题就可以设置为“打开还原文件”。

### 3. TOpenDialog 组件的重要事件

(1) OnCanClose 事件：当用户试图关闭对话框时将触发这个事件。这样就有机会对用户输入的文件名进行检查。如果文件名不符合要求，就把 CanClose 参数设为 False，表示不允许关闭对话框。

(2) OnFolderChange 事件：当用户在对话框中改变、扩展或折叠了一个目录（文件夹），将触发该事件。在程序中可以利用此事件获取用户浏览过的目录。

(3) OnSelectionChange 事件：当用户打开对话框、选择某个目录或文件、选择另一个过滤器或建立一个新的目录时将触发这个事件。

OnSelectionChange 事件与 OnFolderChange 事件相比，触发的概率比较高。

(4) OnTypeChange 事件：当用户在对话框中选择了另一个过滤器时，将触发这个事件。在此事件中结合 FilterIndex 属性的使用，可以获取当前对话。

### 4. TOpenDialog 组件的重要方法

TOpenDialog 组件最重要的方法就是 Execute 方法。此方法能够显示对话框，以供用户设置文件路径及文件名。其使用方法如下：

```
OpenDialog1.Execute;
```

## 6.5.2 用于文件保存的 TSaveDialog 对话框组件

TSaveDialog(另存为对话框)组件用于显示一个“另存为”对话框，让用户选择可输入一个要保存的文件名。TSaveDialog 用户编程时需要调用 Execute 方法。运行的核心代码如下：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    if SaveDialog1.Execute then           //保存对话框执行
    begin
        //将选择保存的文件名放置到 Caption 上
        Caption := SaveDialog1.FileName;
    end;
end;
```

此方法能够显示保存对话框，如图 6-25 所示，以供用户设置文件路径及文件名。

TSaveDialog 组件是直接从对象 TOpenDialog 中导出来的，继承了 TOpenDialog 对象的一切属性、事件和方法，相关内容参照 TOpenDialog 组件。

## 6.5.3 用于字体选择的 TFontDialog 对话框组件

### 1. TFontDialog 组件概述

TFontDialog(字体对话框)组件显示一个字体选择对话框，如果用户要选择字体，就要打开“字体”对话框。通过该对话框，用户可以设置字体的名称、大小以及风格等信息。“字体”对话框是用 TFontDialog 组件实现的。同样，需要调用该组件的 Execute 方法才能在运行期看到其对话框。





图 6-25 一个运行的 SaveDialog 对话框

例如要改变一个 Label 的显示字体,操作 FontDialog 对话框的用户代码如下:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    FontDialog1.Font.Assign(Label1.Font); //读取要设置的字体当前风格
    if FontDialog1.Execute then
    begin
        Label1.Font.Assign(FontDialog1.Font); //设置字体到目标对象
    end;
end;
```

通过对话框可以立刻改变字体风格,其运行效果如图 6-26 所示。本示例的完整代码请参照本书配套教学资源。

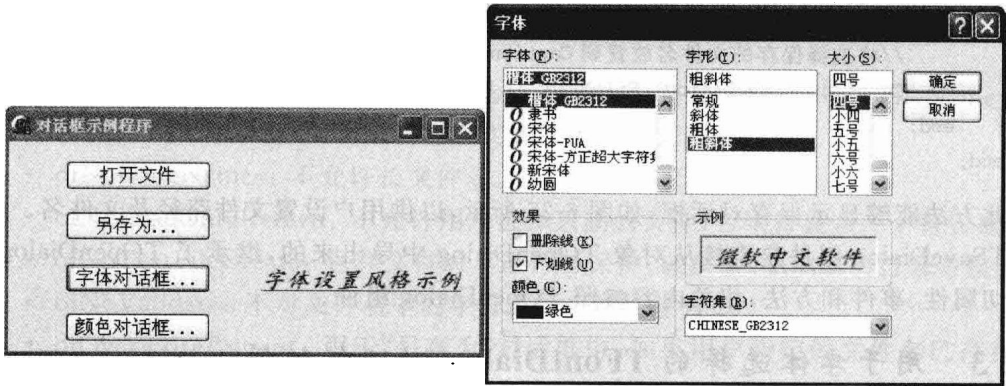


图 6-26 改变字体风格运行效果

2. TFontDialog 组件的主要属性

(1) Font 属性: 用于指定一种字体,对话框打开时这种字体将出现在对话框的“字体”文本框内。当然,用户可以选择其他字体。当用户单击 OK 按钮返回后,Font 属性就是用户新选择的字体。

(2) Options 属性：用于设置“字体”对话框的选项。它有如下 16 个用户可选值。

- fdAnsiOnly：对话框中只列出使用 ansi 字符集的字体。
- fdApplyButton：对话框中将显示“应用”按钮。
- fdEffects：对话框中将显示“效果”选项区域和“颜色”下拉列表框。
- fdFixedPitchOnly：对话框中只列出等宽字体。
- fdForceFontExist：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 属性和 MaxFontSize 属性设置有效。
- fdNoFaceSel：对话框打开时，“字体”文本框中不预先选定一种字体。
- fdNoOEMFonts：对话框中不列出矢量字体。
- fdScalableOnly：对话框只列出可以缩放的字体。
- fdNoSimulations：对话框不列出 GDI 仿真字体。
- fdNoSizeSel：对话框的“大小”文本框不预先选定一种风格。
- fdNoStyleSel：对话框的“字体样式”文本框不预先选定一种风格。
- fdNoVectorFonts：与 fdNoOEMFonts 相同。
- fdShowHelp：对话框中将显示“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。

用户操纵 TFontDialog 组件时可以使用 Execute 方法，还可以使用该组件的 OnApply 事件在用户单击对话框中的“应用”按钮时触发。在处理这个事件的句柄中，可以把用户选择的字体赋给某个组件的 Font 属性，这样在不退出对话框的情况下就可以看到新字体的效果。

#### 6.5.4 用于颜色选择的 TColorDialog 对话框组件

##### 1. TColorDialog 组件概述

TColorDialog(颜色对话框)组件用于显示“颜色”对话框，用户可以在对话框中选择合适的颜色。

##### 2. TColorDialog 组件的主要属性

(1) Color 属性：当用户在对话框中选择了某种颜色并单击“确定”按钮，Color 属性值就是用户新选择的颜色。程序示例如下：

```
procedure TForm1.Button4Click(Sender: TObject);
begin
    if ColorDialog1.Execute then           //颜色对话框运行
        Form1.Color := ColorDialog1.Color; //选择颜色设置到窗体背景
end;
```

运行效果如图 6-27 所示。

(2) Options 属性：用于设置“颜色”对话框的选项。它设定了如下 5 个用户可选值。

- cdFullOpen：对话框打开时可以自定义颜色。
- 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。

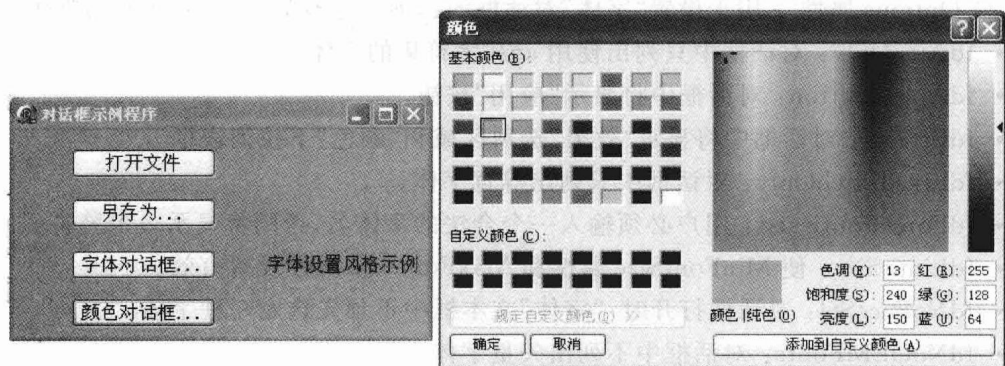


图 6-27 在“颜色”对话框中改变窗体颜色运行效果

- cdSolidColor: 让 Windows 使用与所选颜色最接近的基本颜色。
- cdAnyColor: 让用户选择非基本颜色。

### 6.5.5 用于打印的 TPrintDialog 对话框组件

TPrintDialog(打印对话框)组件用于显示“打印”对话框。用户可以在对话框中进行打印机选择(如果安装了多个打印机)、设置页面和设置打印份数等操作。

TPrintDialog 组件的主要属性如下:

(1) Collate 属性: 如果此属性设为 True, 对话框中的“分页”复选框就被选中, 表示按文档归类, 意思是打印完文档的所有页后再打印这个文档的第 2 份备份, 打印顺序是 1, 2, 3, 1, 2, 3。如果不选中这个复选框, 就按 1, 1, 2, 2, 3, 3 的顺序打印。

(2) Options 属性: 设置“打印”对话框的选项。它有如下 6 个用户可选值。

- poPageNums: 用户可以选择页的范围。
- poPrintToFile: 对话框上将出现“打印到文件”复选框。
- poSelection: 用户可以只打印文档中选择的部分。
- poWarning: 如果没有安装打印机, 将显示一个警告框。
- poHelp: 对话框中将显示一个“帮助”按钮。
- poDisablePrintToFile: 对话框上的“打印到文件”复选框将变灰。

### 6.5.6 用于打开图像的 TOpenPictureDialog 对话框组件

TOpenPictureDialog(打开图像对话框)组件用于显示模态对话框, 用以选择打开图像的文件名, 它是从对象 TOpenDialog 中导出来的, 它的外观与“打开”对话框很相似, 只是增加了一个图像预览区域。如果被选择的图像能够由 TPicture 读取, 它将显示在预览区内。

TPicture 可支持的文件格式有位图(.bmp)、图标(.ico)、Windows 图元文件(.wmf)以及增强的 Windows 图元文件(.emf)等。如果被选择的图像不能显示, 预览区内将显示 None。如果选择一个不能识别的图像格式, 将产生一个 EInvalidGraphic 异常。

## 6.5.7 用于打印模式设置的 TPrinterSetupDialog 对话框组件

TPrinterSetupDialog(打印设置对话框)组件用于显示“打印设置”对话框,对话框的内容随打印机驱动程序的改变而改变。

另外,在“打印”对话框(TPrintDialog)中单击“属性”按钮可以打开“打印设置”对话框。对不同的打印机来说,打印机的属性是不同的。

## 6.5.8 用于查找的 TFindDialog 对话框组件

TFindDialog 组件用于显示一个“查找”对话框,用户可以在对话框中输入要查找的字符串。与前面几个对话框不同的是,“查找”对话框是非模态对话框,就像一般的窗口一样,用户不需要关闭该对话框即可把输入焦点移到其他的窗口。

“查找”对话框是用 TFindDialog 组件实现的。

### 1. TFindDialog 组件的重要属性

(1) FindText 属性:指定要搜索的文本,用户也可以在“查找内容”文本框内输入要搜索的文本,当单击“查找下一个”按钮时,就自动把“查找内容”文本框内的文本赋给 FindText 属性。

(2) Options 属性:用于设置“查找”对话框的选项。它有 13 个用户可选值。

- frDown: 从当前插入点向后搜索。
- frFindNext: 如果用户单击了“查找下一个”按钮,就包含此元素。
- 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(“区分大小写”复选框被选中)。
- frDisableUpDown: “方向”分组框变灰。
- frWholWord: 表示要全字匹配(“全字匹配”复选框被选中)。
- frShowHelp: 对话框中将显示“帮助”按钮。
- frReplace: 如果用户单击了“替换”按钮,就包含此元素。
- frReplaceAll: 如果用户单击了“全部替换”按钮,就包含此元素。
- frDisableMatchCase: “区分大小写”复选框变灰。
- frDisableWholeWord: “全字匹配”复选框变灰。

### 2. TFindDialog 组件的重要事件

TFindDialog 组件的 OnFind 事件,当用户单击了对话框中的“查找下一个”按钮就会触发。

### 3. TFindDialog 组件的重要方法

TFindDialog 组件的 CloseDialog 过程用于关闭“查找”对话框和“替换”对话框。

## 6.6 多文档界面程序设计

多文档(又称多窗口)界面(Multiple Document Interface,MDI)是指在一个父窗口下面可以同时打开多个子窗口。它是从早期 Windows 下的 Microsoft Excel 电子表格程序开始引入的,这是因为 Excel 电子表格用户有时需要同时操作多份表格,MDI 正好为这种操作多表格提供了很大的方便,于是就产生了 MDI 程序。在后续的视窗系统版本中,MDI 得到了更大范围的应用。其中系统中的程序管理器和文件管理器都是 MDI 程序。

用一般的开发工具调用 Windows API 函数开发 MDI 应用程序相当麻烦:必须创建并注册框架窗口和子窗口类、创建框架窗口和客户窗口、写消息循环和回调函数、创建子窗口等,其中的每一步都不是轻而易举的事。

但是在 Delphi 中实现 MDI 应用程序却相对简单,只要在项目中创建好父窗体和子窗体后,设置一下相关窗体的 FormStyle 属性即可:把框架窗口的 FormStyle 设为 fsMDIForm,子窗口的 FormStyle 设为 fsMDIChild。

### 6.6.1 利用 Delphi 模板创建 MDI 程序

Delphi 本身也提供了 MDI 程序模板,选择 File→New→Other 命令,出现一个对话框,如图 6-28 所示。

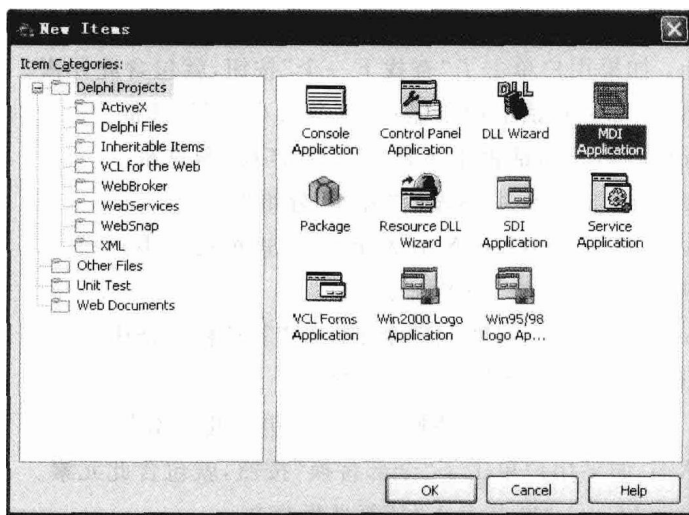


图 6-28 在对话框中选择 MDI 应用程序

在该对话框中选择 MDI Application,单击 OK 按钮,Delphi 就会产生一个 MDI 应用程序框架。直接编译运行该程序,就可以看到该程序已经具有一个 MDI 程序的基本特征了。

MDI 应用程序一般包括两类窗口:父窗口与子窗口。

#### 1. 父窗口

父窗口一般有可改变大小的边框、标题栏和系统菜单等。父窗口提供对下属 MDI 子窗口的管理,每一个 MDI 应用程序都有一个客户窗口。文档或子窗口被包含在父窗口中,父

窗口为应用程序中所有的子窗口提供工作空间。

由 Delphi 模板产生的 MDI 应用程序的父窗口 name 属性为 MainForm,其设计的外观如图 6-29 所示。

窗体的 FormStyle 属性可以确定窗体的类型,它一共有 4 种属性值。

- fsNormal: 表示窗体既不是 MDI 父窗口也不是 MDI 子窗口,可能是单文档 SDI 窗口或者对话框。这个属性值是默认值。
- fsMDIChild: 表示这个窗体是一个 MDI 子窗口。
- fsMDIForm: 表示这个窗体是一个 MDI 父窗口。
- fsStayOnTop: 表示窗体始终保持在窗体所属工程中其他窗体的上面,除非还有别的窗体的 FormStyle 属性也被设置成了 fsStayOnTop。

通过对象查看器(Object Inspector)可以看到,由 Delphi 模板产生的 MDI 父窗体 MainForm 的 FormStyle 属性值为 fsMDIForm,如图 6-30 所示。



图 6-29 Delphi 模板产生的 MDI 应用程序父窗口的  
设计外观

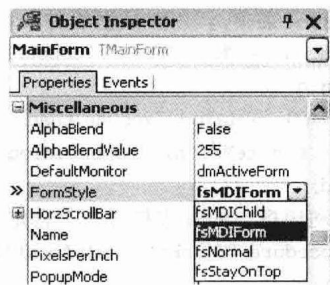


图 6-30 MDI 应用程序父窗体  
FormStyle 的属性值

一般来说,在程序运行期间不要修改窗体的 FormStyle 属性。

## 2. 子窗口

当用户打开或创建一个文档时,客户窗口便为该文档创建一个子窗口。每个子窗口都有可以改变大小的边框、标题栏、系统菜单、最小最大化按钮等。任何时刻只有一个子窗口是活动的。子窗口不能超出父窗口客户区域的范围。子窗口归属于父窗口管理,如果父窗口关闭,则所有子窗口全部关闭。

由 Delphi 模板产生的 MDI 应用程序子窗口的 name 属性为 MDIChild,其外观如图 6-31 所示。

该窗体 MDIChild 的 FormStyle 属性值为 fsMDIChild,如图 6-32 所示。

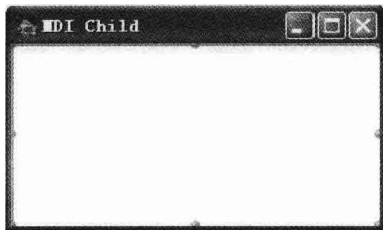


图 6-31 Delphi 模板产生的 MDI 应用程序  
父窗口的设计外观

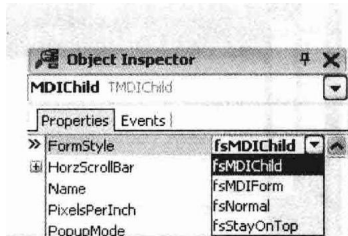


图 6-32 MDI 应用程序子窗体  
FormStyle 的属性值

父窗口中的主要代码如下：

```
//创建子窗口过程
procedure TMainForm.CreateMDIChild(const Name: string);
var
    Child: TMDIChild;
begin
    { create a new MDI child window }
    Child := TMDIChild.Create(Application);    //调用子窗体创建方法
    Child.Caption := Name;
    if FileExists(Name) then Child.Memo1.Lines.LoadFromFile(Name);
end;

//菜单中新建一个子窗口,调用创建过程
procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
    CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
end;

//打开菜单,调用对话框组件
procedure TMainForm.FileOpen1Execute(Sender: TObject);
begin
    if OpenFileDialog.Execute then
        CreateMDIChild(OpenDialog.FileName);
end;

//利用模态的方式打开一个自定义的"关于"窗体
procedure TMainForm.HelpAbout1Execute(Sender: TObject);
begin
    AboutBox.ShowModal;
end;

//退出菜单,作用是关闭父窗口
procedure TMainForm.FileExit1Execute(Sender: TObject);
begin
    Close;
end;
```

该 MDI 程序运行以后,其效果如图 6-33 所示。

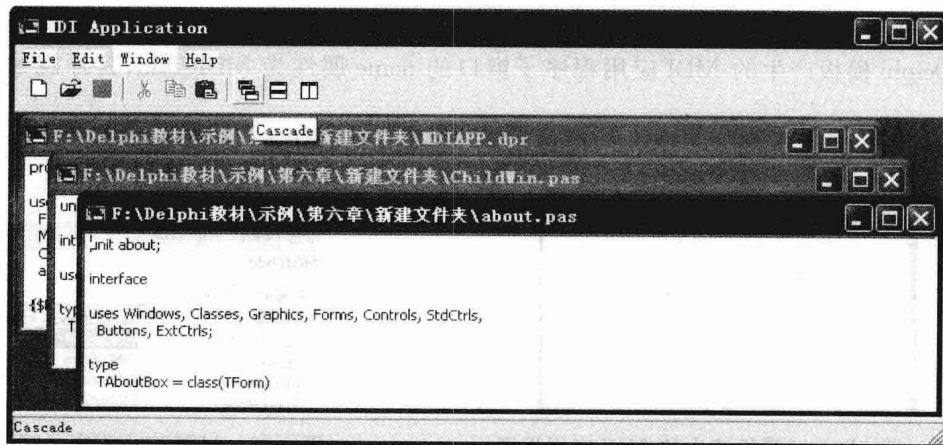


图 6-33 MDI 应用程序运行效果图

## 6.6.2 用户设计常规 MDI 程序

Delphi 创建的窗体中,Form1 一般是主窗体,它在程序中具有较重要的地位,如 Form1 被关闭时,程序通常也就结束了。其他窗体与 Form1 的关系往往就是主从关系,如 Form2 可能是 Form1 在某项操作时需要的一个操作窗口等。一般可在主窗体中调用从属窗体的 Create, Show 和 Close 等方法控制它们的创建、显示和关闭等。

从上述的知识中知道 MDI 窗口中至少有两个窗体程序,在多窗体应用程序中窗体之间是父子关系。利用这些知识来自定义 MDI 应用程序,步骤如下:

### (1) 新建应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统默认创建一个窗体 Form1,该窗体对应的代码单元为 unit1,当然用户可以更改成其他名称。

### (2) 设计 MDI 父窗口。

首先将 Form1 的 Caption 设置成“用户自定义 MDI 窗口示例”,然后设置 Form1 的 FormStyle 属性值为 fsMDIForm。给 Form1 放置一个主菜单 MainMenu1,通过菜单设计器设计两个下拉菜单:“操作”和“管理”。“操作”下面只有一个菜单项“创建子窗口”,“管理”下面有“平铺”、“层叠”和“排列”三个菜单项。设计界面如图 6-34 所示。

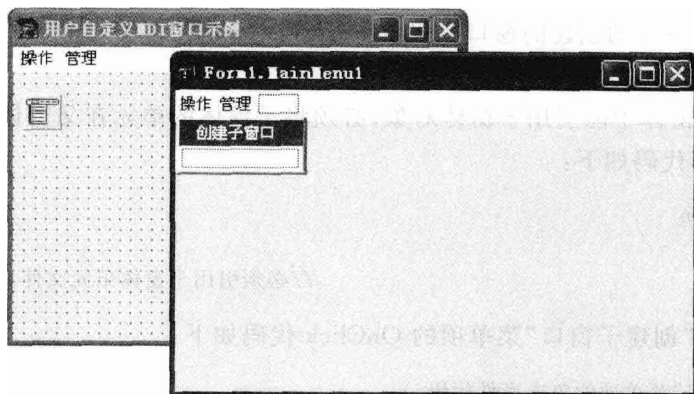


图 6-34 自定义 MDI 应用程序父窗口设计

### (3) 设计 MDI 子窗口。

选择 File→New→Form—Delphi for Win32 命令,创建一个新的窗体程序,窗体名称默认为 Form2,该窗体对应的代码单元为 Unit2。

Form2 的 Caption 值由程序运行创建的时候自动设置。将 Form2 的 FormStyle 属性值设为 fsMDIChild。

Form2 的窗体可以根据用户需要放置任何组件,为了简单起见,本例中什么组件都不放置。

### (4) 修改窗体创建选项。

MDI 应用程序运行时一般只需要自动父窗体,子窗体由父窗体在程序运行时根据用文指令动态创建,所以需要修改项目窗口创建顺序。

选择 Project→Options 命令,在打开的对话框中进行设置,如图 6-35 所示。



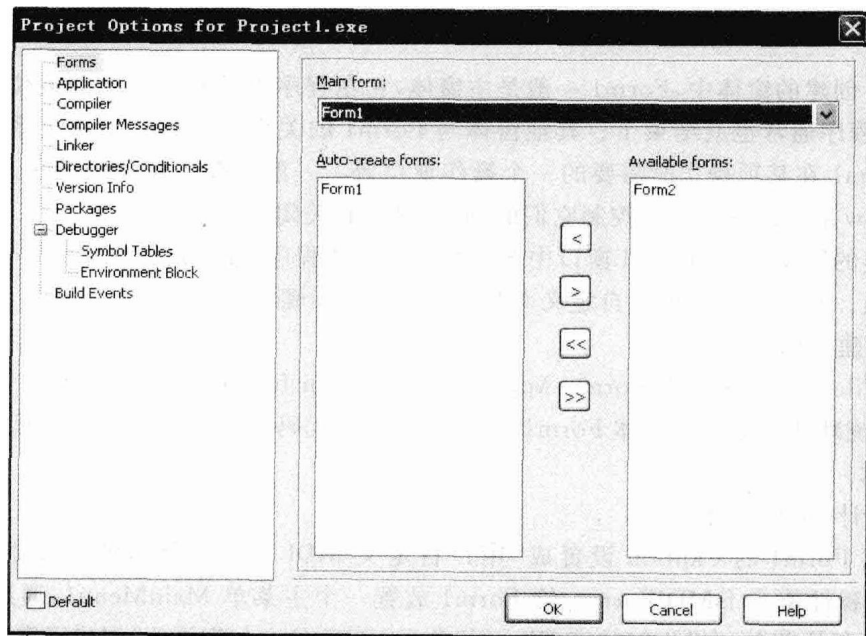


图 6-35 自定义 MDI 应用程序父窗口设计

在对话框中,将自动创建的窗口只保留 Form1。

(5) 父窗体代码编写。

① 要使得父窗体中能引用子窗体对象,必须将子窗体的单元在父窗体的实现部分通过 uses 应用进来,其代码如下:

```
implementation
{ $R *.dfm}
uses Unit2;                                //必须引用子窗体单元文件
```

② 然后编写“创建子窗口”菜单项的 OnClick 代码如下:

```
//“创建子窗口”菜单项的单击事件代码
procedure TForm1.N3Click(Sender: TObject);
var aChildFm:TForm2;                        //定义子窗口变量
begin
    aChildFm := TForm2.Create(Self);        //通过窗体构造函数创建子窗体对象
                                           //为子窗体设置个性标题
    aChildFm.Caption := '子窗口 创建时间: ' + FormatDateTime('yyyy-mm-dd'
        + ' hh:mm:ss AM/PM', NOW);
    aChildFm.Show;                          //显示本次创建的子窗体
end;
```

还有子窗体管理的代码如下:

```
procedure TForm1.N4Click(Sender: TObject);
begin
    self.Tile;                              //层叠当前已经创建并打开的子窗口
end;
```

```

procedure TForm1.N5Click(Sender: TObject);
begin
    self.Cascade;                                //平铺当前已经创建并打开的子窗口
end;
procedure TForm1.N6Click(Sender: TObject);
begin
    self.ArrangeIcons;                            //排列最小化子窗口图标
end
end

```

(6) 子窗体代码编写。

子窗体的关闭必须通过窗体的 OnClose 事件编写,否则不能关闭,只能最小化。

选择 Form2,在 Object Inspector 的事件栏双击 OnClose,为其编写窗体关闭时的释放代码如下:

```

procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Action := CaFree;                            //表明关闭时释放窗体
end;

```

(7) 保存并运行。

运行效果如图 6-36 所示,用户可以测试,各项特性均符合 MDI 应用程序要求。

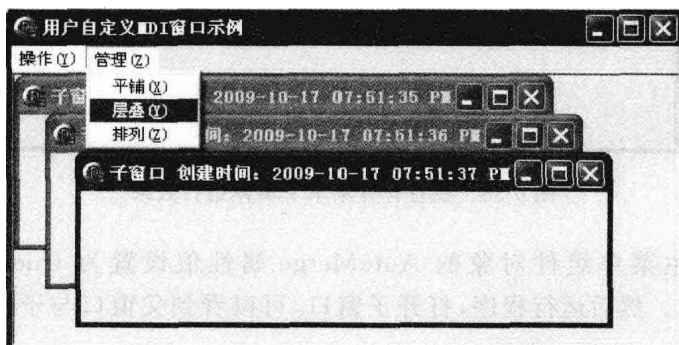


图 6-36 自定义 MDI 应用程序运行效果

需要说明的是,本程序子窗体的创建采用了动态创建技术:

```

aChildFm := TForm2.Create(Self);                //通过窗体构造函数创建子窗体对象

```

aChildFm 窗体是在程序运行阶段完成的。在程序代码中调用窗体的 Create 方法。静态创建的子窗体在程序刚运行时就全部装入内存当中,如果一个应用程序的子窗体数目很多,将占用过多的内存资源。动态创建的子窗体在程序刚运行时并没有装入内存中,当使用到该子窗体时才装入内存,使用完后立即将其从内存中释放,以节省内存资源。

### 6.6.3 MDI 应用程序菜单的合并

在多窗体应用程序中,从属于主窗体的窗口(不管它是否为子窗口)上如果有菜单,通常会将其与主窗体上的菜单合并。在 Delphi 中要做到这一点很容易,只要适当设置菜单的 AutoMerge 和 GroupIndex 属性即可。

在 6.6.2 节的例子程序中继续为子窗体 Form2 中添加一个主菜单,并随意建立几个菜单项,如图 6-37 所示。

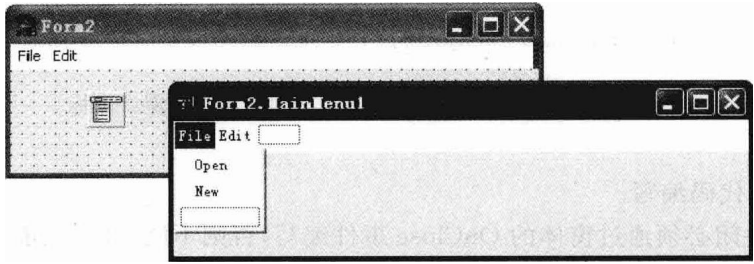


图 6-37 子窗体建立菜单设计界面

保存运行后,由于没有设置菜单合并,运行效果如图 6-38 所示。

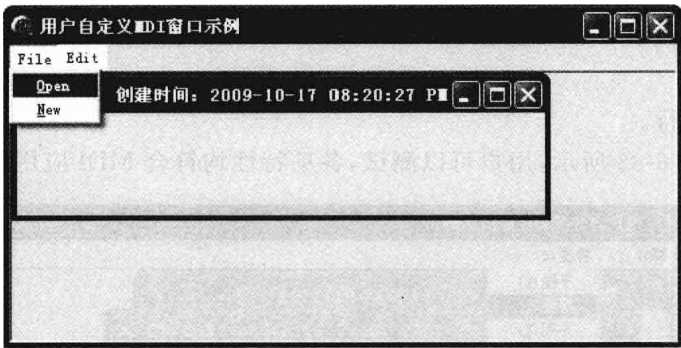


图 6-38 没有合并菜单子窗体运行效果

在 Form2 将主菜单组件对象的 AutoMerge 属性值设置为 true, File 下拉菜单的 GroupIndex 设为 2。然后运行程序,打开子窗口,可以看到父窗口与子窗口的菜单以及合并,如图 6-39 所示。

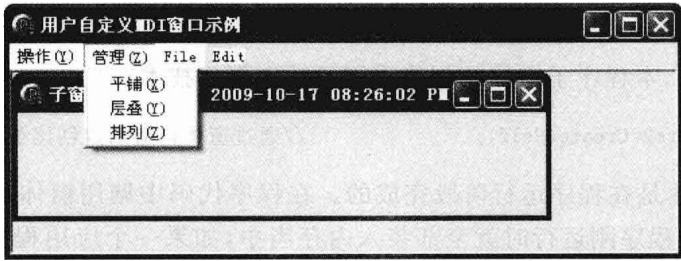


图 6-39 合并菜单子窗体运行效果

**注意:** 其中的菜单已被合并,合并后各下拉菜单的顺序是按照 GroupIndex 从小到大排列的。

有趣的是,若父窗体中“管理”的 GroupIndex 设置为 2,则在合并时会用 File 覆盖具有同一 GroupIndex 的“管理”项。

### 6.6.4 多页窗体设计

多页窗体,也可称作属性对话框,是一种较新颖的图形界面。它使用某些控件将窗体分成若干个页面,用户可单击页标签在不同的页面间切换。这种界面的一个明显好处是扩大了窗体的客户区,使其能排列大量的控件,也便于区分对话框的不同功能区域。

Delphi 提供了几种用于建立多页窗体的组件,它们是 TTabControl、TPageControl 和 TNotebook 等,都位于 Win32 组件页上,并且在用法上大同小异。

通过 TPageControl 组件的 Pages 属性可以访问多页组件的每一页,Pages 属性是由所有页组成的数组,它的每一个元素都是一个 TTabSheet 对象。TTabSheet 对象本身又是个容器,页上的组件都是它的子组件,可以通过 Control 属性访问它们。Delphi 利用 TPageControl 设计多页窗体如图 6-40 所示。

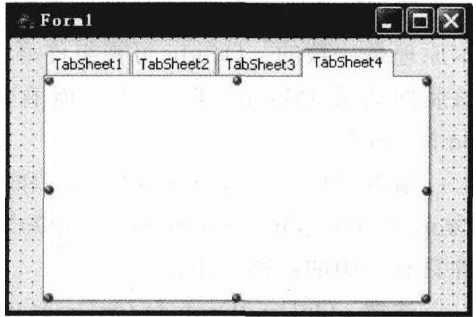


图 6-40 Delphi 利用 TPageControl 设计多页窗体

#### 1. TTabSheet 组件的重要属性

(1) Highlighted 属性: 如果此属性设为 True, 页将用一种特别的颜色来显示。注意,一个多页组件只能有一个页是活动的,但可以有多页的 Highlighted 属性设为 True。

(2) ImageIndex 属性: 如果 TPageControl 的 Images 属性(图像)指定了一个图像列表,这个属性用于指定其中一个图像的序号,该图像将显示在页的标签上。

(3) PageIndex 属性: 用于设置返回页的序号。多页组件的每一页都有一个相异的序号(包括暂时隐去的页),序号从 0 开始,如果删除了某页,序号将重新排列。

(4) TabIndex 属性: 当 TabIndex 的属性设置为 True 时,它的属性与 PageIndex 属性是一致的;但如果有一个或多个页的 TabVisible 属性为 False,TabIndex 属性与 PageIndex 属性就不同。例如,假设多页组件上有三个页,其 PageIndex 属性分别是 0、1、2,如果前两页的 TabVisible 属性为 False,那么最后一页的 TabIndex 属性就是 0 而不是 2。TabVisible 属性为 False 的那两页的 TabIndex 属性为 -1。

(5) TabVisible 属性: 程序有时候需要暂时把某一页隐去,这就需要将这一页(TabSheet 对象)的 TabVisible 属性设为 False,这一页(包括这一页的按钮)将不再显示,用户也就不能直接选择它。

#### 2. TTabSheet 组件的重要事件

(1) OnHide 事件: 当一个页被暂时隐去时将触发该事件,也就是当前页被切换到其他页时触发该事件。

(2) OnShow 事件: 当一个暂时隐去的页重新恢复显示时将触发该事件,也就是在其他页切换到本页时触发该事件。

有了这两个事件,就可以利用 Highlighted 属性设置标签的显示颜色,例如在 OnHide 时将 Highlighted 属性设置为 False,在 OnShow 时将 Highlighted 属性设置为 True,就可以清楚地找到当前打开的页了。

## 6.7 Delphi 拖放技术编程

拖放是鼠标操作的基本动作之一,在 Windows 应用程序的界面设计中起着重要的作用。在 Delphi 中,可以使用组件的有关属性和事件对拖放操作进行控制。

### 6.7.1 拖放技术编程概述

在 Windows 下,窗口可以在桌面上拖放,控件则可以在窗口内或窗口间拖放,它们的基本原理是一致的。Delphi 可视组件通常有一个 DragMode 属性用于控制其是否能被拖动,该属性为 dmManual 时组件不能直接拖动,为 dmAutomatic 时则可以拖动,默认值为 dmManual。

拖放操作可以被分解为“拖动”和“释放”两个基本动作。与拖放有关的事件有 OnDragDrop、OnDragOver、OnStartDrag、OnEndDrag、OnStartDock 和 OnEndDock 等,但也有些控件只有其中的前两个事件。

**注意:** OnDragDrop 和 OnDragOver 事件不属于被“拖”的组件,而是属于接收其“放”的那个组件或容器。例如,Edit1 编辑框在 Form1 窗体内被拖放,则应处理 Form1 的 OnDragDrop 和 OnDragOver 事件。

OnDragOver 用于控制外来组件是否允许被拖入,一般只要写上 Accept := true; 表示允许被拖入(这里 Accept 是方法参数)。在较复杂的情况下,则应根据被拖动对象的类型或某些条件确定是否允许其被拖入。当 Accept 为 false 时不接受(不会发生)OnDragDrop 事件。

OnDragDrop 事件中的代码确定当某组件被拖入并释放时应如何响应,通常这是需要处理的主要事件。在处理该事件时,往往需要对被拖入的对象进行识别。

OnStartDrag 用于组件开始被拖动时的处理,OnEndDrag 则用于组件拖动结束时(即释放之前的瞬间)的处理。

在特殊情况下,如果使用这几个事件还不能对拖放操作进行有效的控制,那么可直接处理鼠标的 OnMouseDown、OnMouseMove 和 OnMouseUp 事件。

在拖放时还可以通过设置 Cursor 属性改变鼠标的光标形状。

### 6.7.2 拖放编程举例

下面介绍一个拖放标签组件的示例,读者仔细体会拖放技术的应用。

#### 例 6.4 拖放编程示例。

**设计构想与过程:** 建立一个应用程序项目,在 Form1 中放入 4 个标签: Label1、Label2、Label3 和 Label4。Label1、Label2 和 Label3 在程序中可以被拖放,因此它们的 DragMode 属性都应设置为 dmAutomatic; Label4 为拖放接收区域。此外,将 Label1、Label2 和 Label3 的 Color 属性分别设置为 clRed、clBlue 和 clYellow,并且相应地,将 Caption 属性分别设置为“红色”、“蓝色”和“绿色”,并将它们的字体设置成“黄色”。

Label4 的 Caption 设置为“拖放接收区域”,需要调整其大小合适,设置其表面颜色为 clWhite,字体颜色为“浅蓝色”。

将所有的标签 Transparent(透明)属性值设置成 False, AutoSize 属性值设置成 False。  
在 Form1 窗体中放置一个按钮组件对象 Button1, 并更改其 Caption 值为“Reset(还原初始状态)”。

设计好的窗体界面如图 6-41 所示。

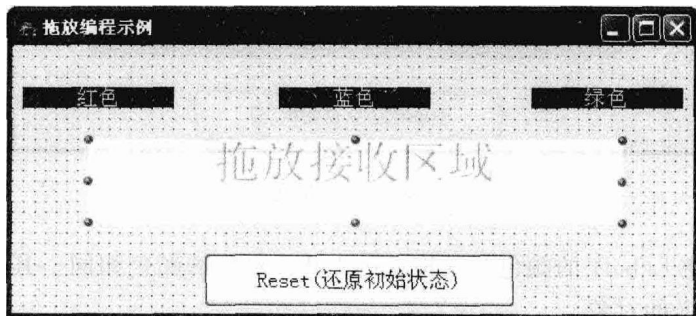


图 6-41 拖放编程界面设计

程序中需要处理 Label4 的 OnDragOver、OnDragDrop 和 Button1 的 OnClick 事件, 下面是有关的事件代码:

```
procedure TForm1.Label4DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
    //目标标签接收拖放源的颜色
    Label4.Color := TLabel(Source).Color;
    //将拖放源对象隐藏
    TLabel(Source).Visible := false;
end;
procedure TForm1.Label4DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
begin
    //表示本对象接收拖动操作
    Accept := true;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    //还原初始状态
    Label1.Visible := true;           //还原红色标签可见
    Label2.Visible := true;           //还原蓝色标签可见
    Label3.Visible := true;           //还原绿色标签可见
    Label4.Color := clWhite;          //目标区域颜色还原
end;
```

程序运行时, Label1、Label2 和 Label3 可被拖入 Label4, 当 Label1 拖入到 Label4 时, Label4 的颜色会变成红色, 如图 6-42 所示。相应地, 拖入 Label2 或 Label3 会使 Label4 变成蓝或绿色。若单击 Button1, 则使程序还原。

本例中 Label4 的 OnDragOver 事件执行 Accept := true; 语句, 使任何组件都能拖入 Label4。从函数声明部分可看出, Accept 是可以返回给系统的布尔型变量。

Label4 的 OnDragDrop 事件执行了两个语句。语句 Label4.Color := TLabel

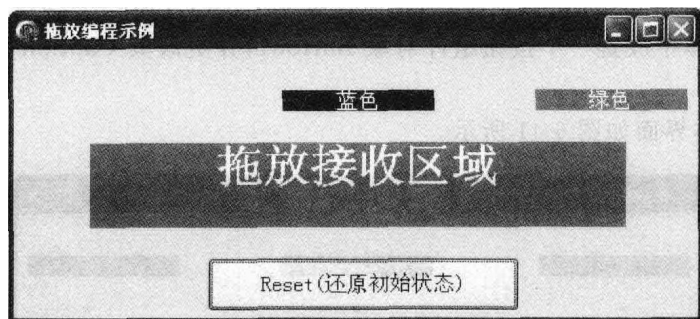


图 6-42 红色拖入时拖放运行效果

(Source).Color;使 Label4 的颜色与被拖入的 Label 对象的颜色相同。其中的 Source 参数代表“源”，即被拖入的对象。

但从函数声明中看到 Source 是 TObject 类型的(由于系统不能预先知道哪些对象可能被拖入,因此只能用最宽泛的 VCL 类型 TObject),因此要用 TLabel() 实行强制类型转换后才能访问其 Color 属性。语句 TLabel(Source).Visible := false;使被拖入的 Label 对象不可见,给人的感觉是它们被拖入到 Label4 中去了。

有时候,拖放的目的只是为了在窗体上移动组件的位置,在这种情况下,只需要处理窗体的 OnDragOver 和 OnDragDrop 事件。在 OnDragOver 中输入 Accept := true;语句,并在 OnDragDrop 中将被拖放对象的 Top 和 Left 属性设置为 X 和 Y。例如:

```
TLabel(Source).Left := X;
TLabel(Source).Top := Y;
```

从函数的声明部分可以看到 X 和 Y 是由系统传入的整型变量,代表对象被放下时的鼠标位置。

## 6.8 窗体的分割技术

Windows 自带的“资源管理器”界面给人一种简单明了的感觉,并且操作非常方便。其实分割窗体在这里面起到了很大的作用,窗体分割是非常实用的技术,它使用可以移动的分割条将一个窗体分割成几个区域,用户可以随意地调整这几个区域的大小。

### 6.8.1 分割技术概述

在 Delphi 中可以使用位于组件面板 Additional 页上的 TSplitter 组件作为分割条。该组件的常用属性有 AutoSnap、Beveled、MinSize、ResizeStyle 和 Align 等,常用的事件有 OnCanResize、OnMoved 等。其中,Align 用于定位;Beveled 和 ResizeStyle 与外观有关;MinSize 指定一个最小值(必须大于 0,单位是像素)以避免某一区域在分割后宽度或高度小于该值。

TSplitter 通常与 TPanel 等组件配合使用,每个 Panel 或控件可形成一个区域,Panel 之间用 Splitter 进行分割。适当设置 Align 等属性可使窗体实现水平分割或垂直分割,还可

以将这两种分割组合。

## 6.8.2 Delphi 分割窗体操作

在 Delphi 中可以不用一行代码就设计出分割窗体。现在举一个例子说明,具体操作如下:

(1) 在窗体中添加一个 TMemo 组件,设置 Name 属性为 Memo1,Align 属性为 alBottom。然后向窗体上添加一个 TSplitter 组件,设置 Name 属性为 Splitter1,Align 属性和 Memo1 组件的 Align 属性一样,设置为 alBottom,这样,Splitter1 组件就会紧贴在 Memo1 组件的上方。

(2) 在窗体中添加一个 TImage 组件,设置 Name 属性为 Image1,Align 属性为 alTop。设置 Image1 的 Picture 通过对话框给其设置一幅图片,同时设置其 Stretch 属性为 True,表明图像适应图形框大小。然后向窗体上再添加一个 TSplitter 组件,设置 Name 属性为 Splitter2,Align 属性和 Image1 组件的 Align 属性一样,设置为 alTop,这样,Splitter2 组件就会紧贴在 Image1 组件的下方。

(3) 最后向窗体上添加一个 TPanel 组件,设置 Name 属性为 Panel1,Align 属性为 alClient。

经过以上的设置,整个窗体就会在程序运行的过程中被分割成上、中、下三个部分,如图 6-43 所示。

(4) 保存程序并运行,如图 6-44 所示。可以根据光标实时地通过拖动鼠标来改变三个部分的大小。

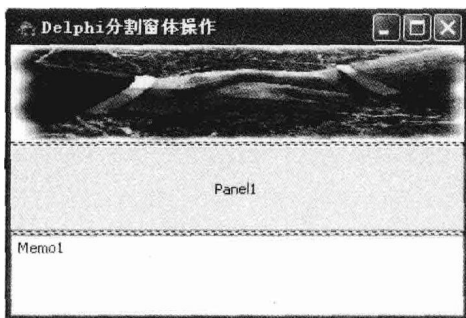


图 6-43 水平分割窗体设计界面



图 6-44 水平分割窗体运行效果

垂直分割技术同上面讲述的水平分割技术类似,读者可以根据本例改变相关组件的属性自己编程试验。

Delphi 中这种分割窗体不需要编写代码即可运行,非常简单有效。

## 本章小结

本章一开始主要介绍了主菜单和快捷菜单的概念、创建与使用方法,中间详细介绍了工具栏、状态栏等相关组件的属性、方法、事件以及对话框函数与对话框组件的概念与使用方



法,最后介绍了 MDI 程序设计方法以及常用的窗体控制技术。

通过本章的学习,读者应掌握利用 TMainMenu 和 TPopupMenu 组件创建主菜单和快捷菜单的方法。同时掌握对话框函数与组件的使用以及常用的 Windows 窗体控制技术,并能使用这些技术来开发不同风格的应用程序界面。

## 思考与练习

1. 按照菜单项的功能,菜单项分为哪三种类型?各有什么功能?
2. 如何设置快捷键和热键,如何进行菜单分组?
3. 利用 TMainMenu 组件创建一个主菜单“数据查询”,包括的菜单项有“查询基本情况”、“查询工作情况”和“查询工资情况”。
4. 利用 TPopupMenu 组件建立快捷菜单(包括的菜单项有“还原”、“最小化”、“移动”和“关闭”)。设计好后,如何使其在程序执行时右击窗体会弹出相应的菜单?
5. 如何利用菜单模板创建、插入、删除菜单?
6. 模仿 Windows 记事本的菜单界面编写一个程序。
7. Delphi 的对话框函数有哪些?各有什么特点?
8. 何为 MDI?简述在 Delphi 中编写 MDI 程序的过程。
9. 根据 6.8.2 节中的示例,设计一个具有 2 个垂直分割条窗体程序,并运行测试它。

Delphi 语言具有对文本、声音、图像和视频等多种媒体信息很强的控制和管理能力。在 Delphi 中,利用这些简单易学的可视化图形图像多媒体设计组件,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。

本章主要包括以下内容:

- TCanvas(画布);
- 常用图形、图像类;
- 图形图像组件重绘和鼠标事件。

### 7.1 TCanvas

Delphi 具有很强的图形图像编程能力,Delphi 的 TForm(窗体)组件以及很多其他的组件,例如 TPanel、TPaintBox、TDrawGrid 和 TBitBtn 都提供了图形图像显示绘制功能,这些组件都具有一个叫 Canvas(画布)的属性,属于 TCanvas 类。

使用 Canvas 的优点之一是可以统一的方法处理各种对象(包括屏幕、打印机和元文件等)上的图形操作。编程者使用 Canvas 属性可以在具有此属性控件的表面随心所欲地绘图,这对完善用户界面或者制作一些屏幕特技都有着非凡的作用。

#### 7.1.1 TColor

在 Windows 下可以根据硬件的性能将显示器设置为 16 色、256 色、16 位增强色和 24 位真彩色等模式。在使用 24 位真彩色时,每一种颜色可以用 R、G、B 三个非负整数表示。这三个整数都是小于等于 255 的(即可以用一个字节表示),它们分别代表红、绿、蓝三种基本色成分在该颜色中所占的比重,用这种方法表示的颜色有时又简称为 RGB 颜色。当显示模式不是 24 位真彩色时,某些 RGB 颜色就无法显示。但 Windows 从兼容性考虑,仍允许用户使用所有的 RGB 颜色,但在实际显示时将不能显示的颜色替换为某个与该 RGB 颜色最接近的可显示颜色。

Delphi 为 RGB 颜色定义了一个专门的类型——TColor(绘图颜色)。TColor 类型用于定义一个对象的颜色。很多绘图组件的颜色属性就是 TColor 类型。在 Graphics 单元中 TColor 定义如下:

```
TColor = -(COLOR_ENDCOLORS + 1)..$02FFFFFF;
```

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(浅蓝)、clBlack(黑色)、clBlue(蓝色)、clDkGray(深灰)、clFuchsia(紫红)、clGray(灰色)、clGreen(深绿)、clLime(柠檬绿)、clLtGray(浅灰)、clMaroon(褐红)、clNavy(深蓝)、clOlive(橄榄绿色)、clPurple(紫色)、clRed(红色)、clSilver(银灰)、clTeal(绿灰)、clWhite(白色)和 clYellow(黄色)共 18 种常见颜色以及 clWindow、clMenu、clCaptionText 和 clBackground 等若干系统配色。这些常量都是以小写字母 cl 开头的。

其实用 4 字节的二进制码可以直接定义颜色,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,则表示用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;最高位字节是 \$02,则用当前设备描述表中逻辑调色板的次相近颜色匹配。

Delphi 语言还可以使用预定义宏 RGB 将三个颜色分量转换为 TColor 类型。例如,RGB(255,0,0)是红颜色,RGB(255,255,0)是黄颜色。

显示器或打印机上最小的显示单位是像素,像素是有一定大小的。通常在 Windows 下可设置屏幕具有 640×480、800×600 或 1024×768 等不同的分辨率,其中的数字就代表了屏幕上水平和垂直方向的像素数目。Canvas 所具有的 Pixels 属性就是用来操纵像素的,它是一个二维数组,数组的下标表示像素的坐标,数组的元素是属于 TColor 类型的。

7.1.2 Pen 属性

在一幅图像中,设置 Pen 属性可指定画线和画图形轮廓而使用的画笔种类。Pen 属性的数值是 TPen 对象。设置 TPen 对象的属性,可以指定画笔的颜色、风格、宽度以及样式等。

注意: 仅对指定 TPen 对象的 Pen 属性进行设置,而不是替代当前的 TPen 对象。

TPen 类对象主要用于画线,它有 Width、Color、Style 和 Mode 等属性,分别表示笔的宽度、颜色、线型和模式。笔的宽度也就是用笔画出的线宽度,单位是像素,默认值为 1。

Style 属性(属于 TPenStyle 枚举类型)定义了线段的各种类型,其取值与含义如表 7-1 所示,常用的有 psSolid、psDash、psDot、psDashDot、psDashDotDot 和 psClear。其中,psSolid 指实线,为默认定义的线型;psClear 使笔划不出现,仅用于特殊目的;其余都是某种类型的虚线,其中有的虚线仅在笔的宽度为 1 时有效。

表 7-1 TPen 类的 Style 属性取值与含义

Style 属性值		含义说明
psSolid	=0	画固定线段,线段实心
psDash	=1	画由均匀短线与空格组成的线段,即线段为断线
psDot	=2	画由均匀等距点组成的线段,即线段为纯点线
psDashDot	=3	画由点、空格和短线段组成的线,即线段为点划线

Style 属性值	含义说明
psDashDotDot = 4	画由双点、空格和短线段组成的线,即线段为双点划线
psClear = 5	画不可见线段,仅用于特殊目的
psInsideFrame = 6	画实线,但笔宽是向里扩展
psUserStyle = 7	自定义, Delphi.Net 支持
psAlternate = 8	交替, Delphi.Net 支持

通过窗体 OnPaint 事件可以绘制出各种线型,如图 7-1 所示,使用如下代码可以实现

```
//本程序依次展示了画线的各种样式
procedure TForm1.FormPaint(Sender: TObject);
var
  i, x, y, n: Integer;
begin
  x := 10;
  y := 15;
  n := ClientWidth - 2 * x;           //线长
  Canvas.Pen.Width := 1;             //线宽
  for i := 0 to 8 do
  begin
    Canvas.Pen.Style := TPenStyle(i); //设置线性,通过循环包含了所有风格
    //设置一个随机颜色,Random()为取随机数函数
    Canvas.Pen.Color := RGB(random(255), 0, random(255));
    Canvas.MoveTo(x, y);              //设置画线起点
    Canvas.LineTo(x + n, y);          //画线到终点
    y := y + 15;                      //设置画下一条线的纵坐标
  end;
end;
```

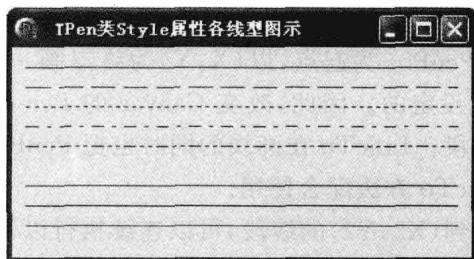


图 7-1 通过窗体 OnPaint 绘制各种线型

Mode 属性定义 Pen 实际画线的模式,Mode 取值来自于 TPenMode 集合(pmBlack、pmWhite、pmNop、pmNot、pmCopy、pmNotCopy、pmMergePenNot、pmMaskPenNot、pmMergeNotPen、pmMaskNotPen、pmMerge、pmNotMerge、pmMask、pmNotMask、pmXor 和 pmNotXor),其常用的取值与含义如表 7-2 所示。Mode 可对当前的颜色进行各种运算,如取反、与屏幕背景色逻辑运算等,主要目的是可以对线段的颜色重新定义,可以实现复杂算法的绘图工作,但不改变 Color 属性。

表 7-2 TPen 类的 Mode 常用属性取值与含义

Mode 属性值	含义说明
pmBlack	使得 Pen 使用纯黑颜色画线
PmWhite	使得 Pen 使用纯黑白色画线
pmNop	画笔无效
pmXor	使用当前 Color 颜色与背景色异或运算的颜色画线
PmCopy	使用当前笔颜色 Color 属性中的颜色
PmNotCopy	使用当前笔颜色 Color 的反转值
PmMergePenNot	使用当前笔颜色 Color 与屏幕颜色反转值的混合
pmMerge	画笔颜色与背景颜色的组合
pmMask	画笔颜色与背景公共颜色的组合
pmMaskNotPen	画笔颜色的反色与背景的公共色的组合

TCanvas 类有一个 PenPos 属性用于为该画布的 Pen 对象在画布上保存一个当前位置,它是 TPoint 类型的值。TCanvas 对象创建时, PenPos 属性被初始化为(0,0)。

读 PenPos 可以知道画笔当前图形区域的准确位置。设置 PenPos 属性相当于调用 MoveTo 方法。

### 7.1.3 MoveTo 与 LineTo 方法

MoveTo 方法在 Delphi 中的声明如下:

```
Procedure MoveTo(x,y : Integer);
```

该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,一般来说不要直接改变 PenPos 的值。

LineTo 方法在 Delphi 中的声明如下:

```
Procedure LineTo(x,y : Integer);
```

该方法可在画布上以 PenPos 为起点,以(X,Y)为终点画一线段。LineTo 画线所用的笔是由 Canvas 的 Pen 属性设定的。因此,改变 Canvas 的 Pen 属性能使 LineTo 画出各种不同宽度、颜色和线型的线段。LineTo 在画线的同时也把 PenPos 位置移到了(X,Y)。

LineTo 通常要和 MoveTo 方法配合使用。

例如,要画从(X1,Y1)到(X2,Y2)的线段,可以连续执行以下两个语句:

```
Canvas.MoveTo (X1, Y1);
Canvas.LineTo (X2, Y2);
```

**例 7.1** 利用 Canvas 的画笔 Pen 动态绘制渐变颜色的窗体。

设计构想:利用 Canvas 的画笔动态绘制渐变颜色的线条,即每画一次线,使得 Pen 选取不同的有规律变化的颜色,使窗体 Form1 呈现出的颜色自上而下由深色成亮色。

(1) 在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。并从组件面板上拖曳一个单选框组件 RadioGroup1,设置其有三个选项:“红色”、“绿色”和“蓝色”。调整组件位置及大小后,窗体界面如图 7-2 所示。

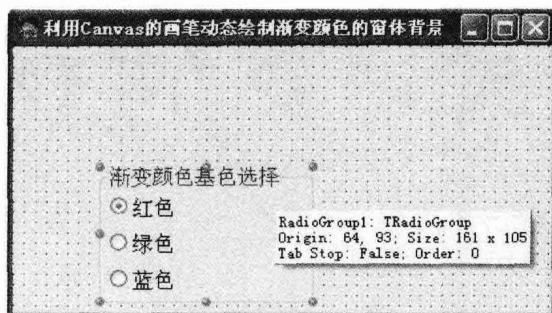


图 7-2 渐变颜色的窗体界面设计

(2) 选择窗体 Form1, 编写其 OnPaint 事件代码如下:

```
procedure TForm1.FormPaint(Sender: TObject); //窗体画面重绘事件
var i:integer; //定义循环变量
    c:Byte; //RGB 颜色分量
begin
for i:=0 to ClientHeight do //扫描窗体所有的像素行
begin
    c:= Round(i*255/ClientHeight); //设置渐变的颜色分量
    case RadioGroup1.ItemIndex of
        0: Canvas.Pen.Color:= RGB(c,0,0); //选择的基色为红色
        1: Canvas.Pen.Color:= RGB(0,c,0); //选择的基色为绿色
        2: Canvas.Pen.Color:= RGB(0,0,c); //选择的基色为蓝色
    end;
    Canvas.MoveTo(0, i); //将画笔 Pen 的起点设在窗体左端
    Canvas.LineTo(ClientWidth-1, i); //使用画笔画线到窗体右端
end;
end;
```

窗体大小如果改变后, 必须重新绘制窗体, 其事件代码如下:

```
procedure TForm1.FormResize(Sender: TObject);
begin
    self.Invalidate; //使得窗体背景重新刷新
end;
```

(3) 选择单选框组件 RadioGroup1, 如果用户重新选择窗体基色后, 需要刷新画面, 则编写其 OnClick 事件代码如下:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
    self.Invalidate; //使得窗体背景重新刷新
end;
```

(4) 保存并运行程序。

选择 File→Save 命令, 保存单元文件与项目文件。

运行该程序, 用户通过单选框组件 RadioGroup 重新选择窗体基色后, 窗体颜色立刻发生渐变, 其效果如图 7-3 所示。

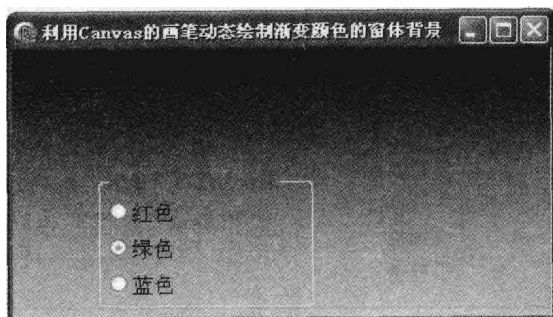


图 7-3 渐变颜色的窗体运行效果

### 7.1.4 Brush 属性

TCanvas 的 Brush 属性决定画布填充图形的背景颜色和填充图案模式。Brush 属性值为 TBrush 对象。当在画布上填充一个空间时,通过设置 TBrush 对象的属性,可以指定使用的颜色、模式或者位图。

**注意:** 设置 Brush 属性是为分配指定的 TBrush 对象,而不是替代当前的 TBrush 对象。

TBrush 类有 Color 和 Style 等属性,Color 表示画刷的颜色和填充模式,Style 表示 TBrush 类的填充样式。Delphi 为画刷定义了样式枚举 TBrushStyle, 包含以下样式: bsSolid=0, bsClear=1, bsHorizontal=2, bsVertical=3, bsFDiagonal=4, bsBDiagonal=5, bsCross=6, bsDiagCross=7。其中,bsSolid 为实心填充,为默认定义的模式; bsClear 为无背景填充(透明); 其余都是采用某种线纹进行背景填充。

下面使用 Form1 窗体的 OnPaint 事件绘制图形帮助读者理解 TBrush 绘图的各种填充模式。其代码如下:

```
//本程序展示了 TBrush 各种填充样式(Style)
procedure TForm1.FormPaint(Sender: TObject);
var
  i, n: Integer;
  r: TRect;
begin
  Canvas.Pen.Color := clRed;           //使用红色画笔
  Canvas.Pen.Width := 1;              //线宽
  n := 25;
  for i := 0 to 7 do
  begin
    Canvas.Brush.Color := clYellow;    //填充颜色为黄色
    Canvas.Brush.Style := TBrushStyle(i); //通过循环使用各种填充样式
    r := Rect(n, 10, n+50, ClientHeight-10); //矩形生成
    Canvas.FillRect(r);                //使用画刷填充矩形
    Canvas.Rectangle(r);               //使用画笔绘制矩形外框
    n := n + 60;                      //设定下一个矩形位置
  end;
end;
```

程序运行以后,窗体运行效果如图 7-4 所示。

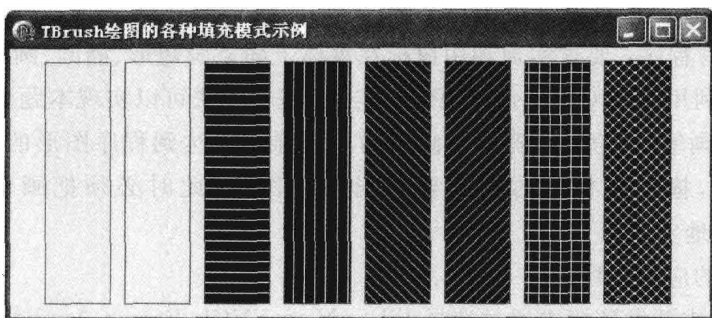


图 7-4 TBrush 各种填充样式运行效果

本例中 Canvas.Rectangle(r)的目的是调用画布的 Rectangle 方法画矩形,该矩形的边框用 Pen 绘制,矩形内部则使用 Brush 填充,该方法在 7.1.5 节将会详细介绍。

## 7.1.5 Rectangle、RoundRect 与 Ellipse 方法

### 1. Rectangle 方法

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。在 Delphi 中的声明如下:

```
procedure TCanvas.Rectangle(X1, Y1, X2, Y2: Integer);
```

其中,(X1,Y1)和(X2,Y2)为矩形上两个对角顶点的坐标。该矩形的边界使用 Pen 的当前属性绘制,矩形内部则用 Brush 的当前属性填充。当 Brush 的 Style 属性为 bsClear 时,矩形内部实际上未填充。

### 2. RoundRect 方法

RoundRect 方法画的是带有圆角的矩形。在 Delphi 中的声明如下:

```
procedure TCanvas.RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
```

方法的功能:该函数画一个带圆角的矩形,此矩形由当前画笔画轮廓(Pen),由当前画刷填充(Brush)。其中,(X1,Y1)和(X2,Y2)为矩形上两个对角顶点的坐标。

X3 用于绘制圆角的椭圆宽度(逻辑单位);Y3 用于绘制圆角的椭圆高度(逻辑单位)。必须两个都设置,否则不会生效。

说明:绘制圆角矩形,内部由当前画刷填充。该函数绘制的图形延伸到但不包括右边和底部坐标,即图形高度为 Y2-Y1,宽度为 X2-X1,外接矩形的宽度和高度都必须大于 2 个单位且小于 32 767 个单位。

### 3. Ellipse 方法

Ellipse 方法可以画一个椭圆。在 Delphi 中的声明如下:

```
procedure TCanvas.Ellipse(X1, Y1, X2, Y2: Integer);
```

该方法在画布指定的外接矩形边界上画一个椭圆,(X1,Y1)是外接矩形左上角的像素坐标,(X2,Y2)是外接矩形右下角的像素坐标。



椭圆的边界使用 Pen 的当前属性绘制,内部使用 Brush 的当前属性填充。当外接矩形是一个正方形时,所画的椭圆就成为一个圆。

**例 7.2** 编写程序:要求实现利用鼠标在窗体上随意画矩形、椭圆、圆或者圆角矩形。

设计构想:利用窗体 Canvas 绘制图形的三个基本方法可以实现本题要求,通过单选框让用户可以设置画笔的线型、线宽以及画刷的填充样式以达到程序图形的个性化。本程序要用到鼠标事件,拖曳鼠标的时候需要画出临时图形,此时必须把画笔的模式设置成 pmXor 才能较好地实现。

(1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。

(2) 用户界面设计。

给窗体中放置的组件有:

RadioGroup1: TRadioGroup;	//画笔的线型选择单选框
RadioGroup2: TRadioGroup;	//画刷的填充样式选择单选框
RadioGroup3: TRadioGroup;	//本次要绘制的图形,矩形、椭圆、圆或者圆角矩形
Label1: TLabel;	//线宽标签
SpinEdit1: TSpinEdit;	//线框调整编辑框
Button1: TButton;	//按钮,使得窗体刷新,可以将所有图形擦除

根据题目要求,给这些组件设置合适的属性初始值,并且调整它们的位置及大小后,窗体界面如图 7-5 所示。

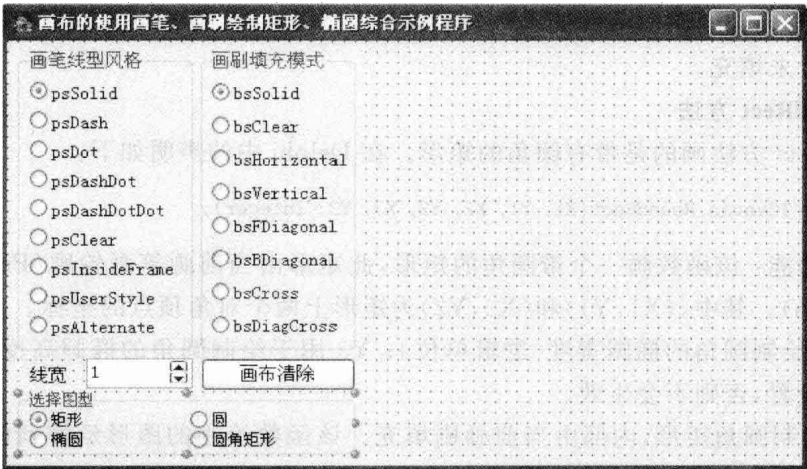


图 7-5 使用画笔、画刷绘制矩形、椭圆综合示例界面设计

(3) 程序代码编辑。

在 Form1 窗体单元 Unit1 的接口私有部分必须设置以下变量,它们的名称与含义如下:

fHasPic:Boolean;	//鼠标移动时曾经画过图
fMouseDown:Boolean;	//鼠标是否处于按下状态

```

fx,fy:Integer;           //鼠标按下初始坐标
fbx,fby:integer;        //鼠标动态移动的坐标

```

在 Form1 窗体单元 Unit1 的实现部分,为所有的组件编写事件代码:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    self.Invalidate;           //窗体刷新,清除窗体画布
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    fMouseDown := False;      //初始化,鼠标处于没有被按下状态
end;

//鼠标按下事件,表明用户绘图开始
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    fMouseDown := True;       //设置鼠标按下状态
    fx := X; fy := Y;         //获取鼠标初始坐标
    Canvas.Pen.Color := clred; //设置临时边框颜色
    Canvas.Brush.Color := clGreen; //设置临时填充颜色
    Canvas.Pen.Mode := pmXOR;  //画笔颜色跟屏幕背景色异或运算,为了方便擦除临时图形
    Canvas.Pen.Style := TPenStyle(RadioGroup1.ItemIndex); //获取线型
    Canvas.Brush.Style := TBrushStyle(RadioGroup2.ItemIndex); //获取填充风格
    Canvas.Pen.Width := SpinEdit1.Value; //设置线框
    fHasPic := False;         //一开始没有画图
end;

//鼠标按下并且移动,表明绘图进行时
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    if fMouseDown then       //如果鼠标按下并且移动
    begin
        if fHasPic then      //如果有动态临时图形
        begin                //擦除旧图形
            case RadioGroup3.ItemIndex of
                0: canvas.Rectangle(fx,fy,fbx,fby); //矩形
                1: canvas.Ellipse(fx,fy,fbx,fby); //椭圆
                2: canvas.Ellipse(fx,fy,fbx,fy+fbx-fx); //圆
                3: canvas.RoundRect(fx,fy,fbx,fby,15,15); //圆矩形
            end;
        end;
        case RadioGroup3.ItemIndex of //给本次鼠标移动的位置画图
            0: canvas.Rectangle(fx,fy,x,y); //矩形
            1: canvas.Ellipse(fx,fy,x,y); //椭圆
            2: canvas.Ellipse(fx,fy,x,fy+x-fx); //圆
            3: canvas.RoundRect(fx,fy,x,y,15,15); //圆矩形
        end;
        fbx := x; fby := y; //保存当前坐标,为下一次擦除图形用
    end;
end;

```

```

        fHasPic := true;                                //表示已经有动态临时图形
    end;
end;

//鼠标左键释放,表明绘图工作结束
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    if fHasPic then                                     //如果有动态临时图形
    begin                                                //擦除旧图形
        case RadioGroup3.ItemIndex of
            0: canvas.Rectangle(fx, fy, fbx, fby);      //矩形
            1: canvas.Ellipse(fx, fy, fbx, fby);        //椭圆
            2: canvas.Ellipse(fx, fy, fbx, fy + fbx - fx); //圆
            3: canvas.RoundRect(fx, fy, fbx, fby, 15, 15); //圆矩形
        end;
    end;
    Canvas.Pen.Mode := pmCopy;                          //设置最终画笔模式
    //绘制最终的图形
    case RadioGroup3.ItemIndex of
        0: canvas.Rectangle(fx, fy, x, y);              //矩形
        1: canvas.Ellipse(fx, fy, x, y);                //椭圆
        2: canvas.Ellipse(fx, fy, x, fy + x - fx);      //圆
        3: canvas.RoundRect(fx, fy, x, y, 15, 15);      //圆矩形
    end;
    fMouseDown := False;                                //表示鼠标已经弹起,本次鼠标操作结束
end;

```

#### (4) 保存及运行测试。

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,通过单选框组件操作画图,其效果如图 7-6 所示。

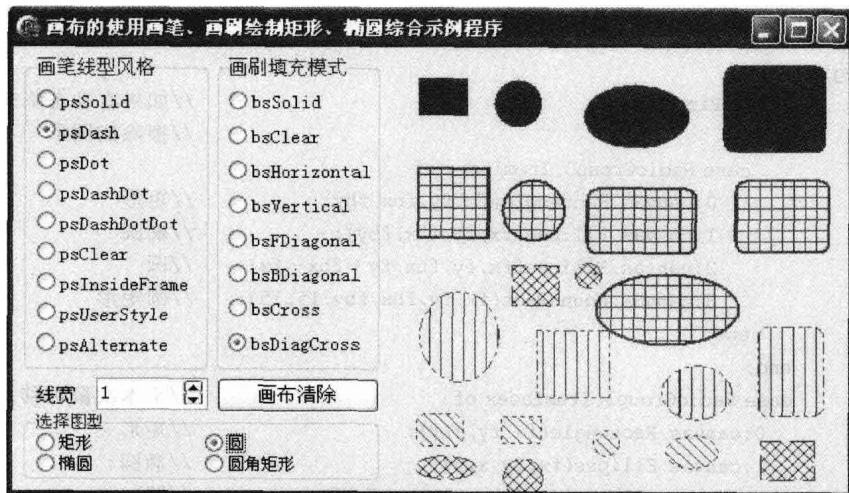


图 7-6 使用画笔、画刷绘制矩形、椭圆综合示例运行效果

## 7.1.6 Font 属性以及 TextOut 方法

### 1. Font 属性

Font 属性的类型为 TFont 类,它本身也是一个对象。可通过设置 Font 对象的属性来确定字体的名称、颜色、尺寸和风格。Canvas.Font 属性使程序可以用 Windows 字体画出文字。通过修改字体的颜色、名称、大小、高度和样式,就可以改变写在画布上的文字的外观。

Font 对象的重要属性如下:

(1) Color: 可以赋值为任何 Delphi 预定义的颜色。例如:

```
Canvas.Font.Color := clGreen; //将字体的颜色设为绿色
```

(2) Name: 用于指定 windows 字体名。例如:

```
Canvas.Font.Name := '黑体'; //把字体设为黑体
```

(3) Size: 以磅为单位指定字体的大小。例如:

```
Canvas.Font.Size := 12; //把字体大小设置成 12
```

(4) Style: 字体样式。Canvas.Font.Style 是由一个多种样式组成的集合,取值和含义为 fsBold(字体加粗)、faItalic(字体倾斜)、fsUnderline(字体加下划线)、fsStrikeOut(字体加删除线)。

(5) Pitch: 描述字间距相关的量,有三个枚举值可选,即 fpDefault、fpVariable 和 fpFixed。

(6) Orientation: 描述字体输出角度。

### 2. TextOut 方法

TextOut 方法在画布上指定位置处绘制文本字符串。在 Delphi 中的声明如下:

```
procedure TCanvas.TextOut(X, Y: Integer; const Text: String);
```

其中,(X,Y)为字符串显示区域左上角位置的坐标,Text 是被显示的字符串,被显示字符的字体由 Canvas 的 Font 属性确定,而字符的背景色由 Canvas 上 Brush 的颜色确定。该方法执行后会使 PenPos 移到所显示的最后一个字符的右上角位置。

下面一段代码使用 TextOut 在窗体画布上显示一段文本:

```
procedure TForm1.FormPaint(Sender: TObject); //窗体表面绘图事件
var
  Str:String; //定义字符串变量
begin
  Str := '欢迎学习 Delphi 图形图像编程!'; //初始化要输出的字符串
  Canvas.Font.Style := [fsBold,fsUnderline]; //字体风格
  Canvas.Font.Color := clBlue; //字体颜色
  Canvas.Font.Height := 32; //字体高度
  Canvas.Font.Orientation := 450; //字体输出角度
  Canvas.TextOut(0, ClientHeight - 20, Str); //绘制字体
end;
```

此段程序执行效果如图 7-7 所示。

第 2~5 句对画布的字体属性进行设置,使字符的大小为 32 个像素点,颜色为蓝色。文本绘制方法 TextOut 的前两个参数使显示位置处在窗体左下角。

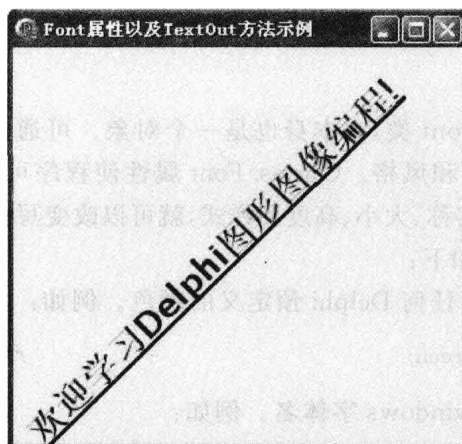


图 7-7 使用 TextOut 在窗体画布上显示一段个性文本

另外,Canvas 跟文本绘制相关的还有两个重要的函数:

- 取字符串高度函数: `function TCanvas.TextHeight(const Text: string): Integer;`
- 取字符串宽度函数: `function TCanvas.TextWidth(const Text: string): Integer;`

### 7.1.7 Pixels 属性、Draw 方法和 StretchDraw 方法

#### 1. Pixels 属性

Canvas.Pixels 属性是一个二维数组,它的每个元素代表窗体表面或客户区的一个像素的 Color 值。通常不需要用到 Pixels 属性,用它太慢。

窗体左上角的像素为: `Canvas.Pixels[0,0];`

窗体右下角的像素为: `Canvas.Pixels[ClientWidth,ClientHeight]`

可通过画布 Pixels 属性的巧妙运用实现窗体特效,如图 7-8 所示。

其核心代码如下:

```
procedure TForm1.FormPaint(Sender: TObject);
var i, j: integer;                                //定义循环变量
begin
    //通过像素设置窗体显示特效
    for i := 0 to ClientWidth-1 do                //扫描所有的列
        for j := 0 to ClientHeight-1 do          //扫描所有的行
            Canvas.Pixels[i, j] := RGB(i, j, round((i+j) / 2)); //改变现实像素
            //Round()为近似取整函数
end;
```

#### 2. Draw 方法

Draw 方法在画布上指定的位置处复制一个图像(该图像来自位图或元文件)。在 Delphi 中的声明如下:

```
procedure TCanvas.Draw(X, Y: Integer; Graphic: TGraphic);
```

一般来说,该方法绘制图像不失真,如图 7-9 所示。

其中,(X,Y)表示被复制图像的左上角在画布中的坐标。

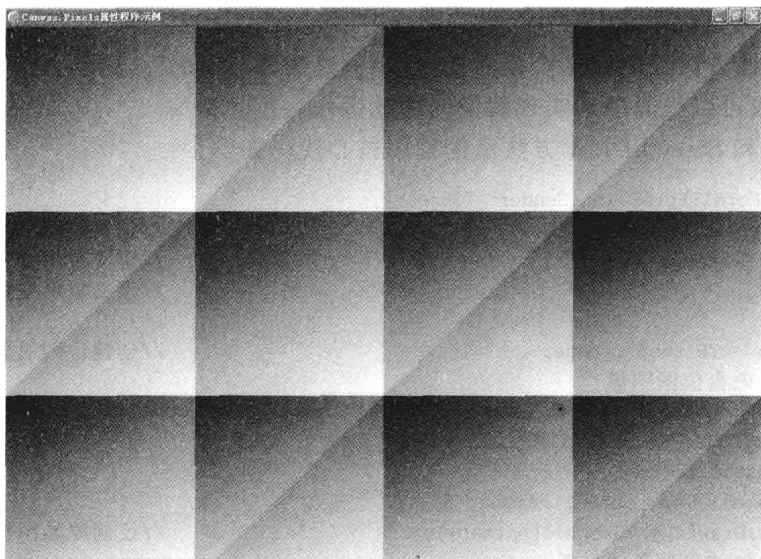


图 7-8 通过画布 Pixels 属性实现窗体特效

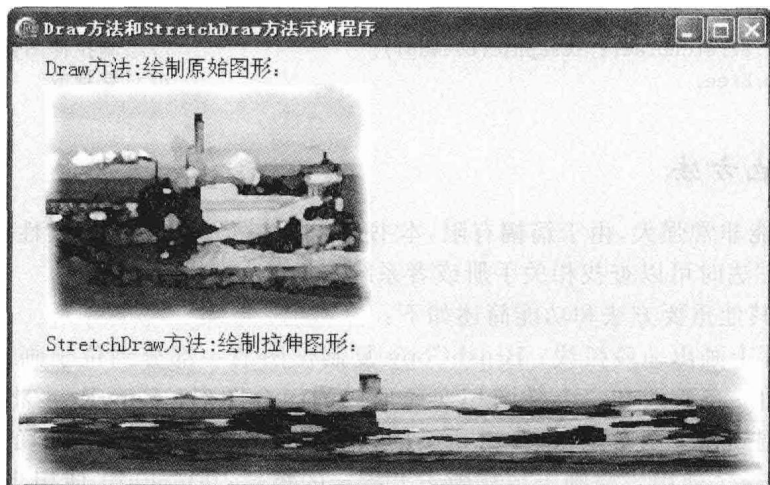


图 7-9 Draw 方法和 StretchDraw 方法画图示例

Graphic 属于 TGraphic 类型,它是 TBitmap、TIcon 和 Tmetafile 等对象的基类。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 和 Tmetafile),否则应该使用可储存任何图像类型的 TPicture 对象。

顺便说一下,在 TGraphic 类中定义的 LoadFromFile 和 SaveToFile 方法只是虚方法,并不能做图形文件的存取。但其子类(如 TBitmap、TIcon 以及 TJPEGImage 等)都继承了这两个方法,以处理不同格式的图形文件。

### 3. StretchDraw 方法

StretchDraw 方法在画布上指定的矩形区域里绘制(充满该矩形区域)一个图像(该图像来自位图或元文件)。在 Delphi 中的声明如下:

```
procedure TCanvas.StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

一般来说,该方法绘制图像会扭曲变形,也就是失真,如图 7-9 所示。

使用此方法可以很简单地实现图像的缩放操作。

Draw 方法和 StretchDraw 方法示例程序核心代码如下:

```
procedure TForm1.FormPaint(Sender: TObject);
var x,y: integer;                                //定义绘图坐标
    bitmap:TBitmap;                              //定义位图变量
    r:TRect;                                      //定义矩形变量
begin
    bitmap := TBitmap.Create;                    //创建位图对象
    //装入磁盘位图图像
    bitmap.LoadFromFile('C:\Program Files\Common Files\CodeGear Shared\Images\Splash\
256Color\FACTORY.BMP');
    x := Label1.Left;
    y := Label1.Top + Label1.Height + 5;
    Canvas.Draw(x,y,TGraphic(bitmap));          //绘制原始图像
    //准备拉伸矩形具体位置
    r.Left := 5;
    r.Right := self.ClientWidth - 5;
    r.Top := Label2.Top + Label2.Height + 5;
    r.Bottom := self.ClientHeight - 5;
    Canvas.StretchDraw(r,TGraphic(bitmap));      //绘制拉伸图像
    bitmap.Free;                                //对象释放
end;
```

### 7.1.8 其他方法

Canvas 功能非常强大,由于篇幅有限,本书不能把它所有的方法和属性详细讲述,读者需要用到这些方法时可以查找相关手册或者系统帮助。

Canvas 的其他重要方法和功能简述如下:

Arc 在图片上画出一段弧线;BrushCopy 复制位图的一部分到位于画布上的矩形中;Chord 画一个由一条直线和一个椭圆相交所形成的闭合图形;CopyRect 将另一画布上图形的一部分复制到这个画布上;DrawFocusRect 画一个矩形框表示矩形内的控件对象具有输入焦点;FillRect 使用当前刷子填充画布上指定的矩形;FloodFill 使用当前刷子填充画布上的一块区域;FraneRect 画出一个矩形的边框;Lock 使其他线条不在画布上画出;Pie 在画布上画一个扇形;PolyBezier 画一簇 Bezier 曲线;PolyBezierTo 画一簇 Bezier 曲线并更新 PenPos 的值;Polygon 在画布上画出闭合图形;Polyline 在画布上画出由一系列链接点(保存在数组中)组成的直线;TextExtent 返回字符串的像素宽度和高度;TextHeight 返回字符串的像素高度。

## 7.2 常用图形、图像类

### 7.2.1 TGraphic 类和 TPicture 类

#### 1. TGraphic 类

TGraphic 类是 TBitmap、TIcon、TMetafile 的基类,如果知道图像的具体类型,应将其

存入相应类的对象中,否则使用可存任何图像类型的 TPicture 类。

Delphi 中定义的 TPicture 类可用于处理各种图形、图像文件,它既包含 Bitmap、Metafile、Icon 对象属性,也包含 Graphic 对象属性。

## 2. TPicture 类

TPicture 类目前能识别的图形、图像文件包括 .jpg、.bmp、.ico、.emf 和 .wmf 等类型。图像的高度和宽度分别定义在 Height、Width 属性中,Graphic 属性表示图像。

TPicture 类的主要方法有:

(1) LoadFromFile: 该方法可以从文件中装载一幅图像。

例如,为位图按钮装入一幅图片的代码: BitBtn1. Glyph. LoadFromFile('train. BMP');

(2) SaveToFile: 该方法将当前图像保存到磁盘文件。要保存一个位图,则要用 SaveToFile 方法;要把图像复制到剪切板,可以调用 TClipboard 对象的 Assign 方法。

## 7.2.2 TBitMap 位图类

TBitMap 不存在组件面板,它封装了 Windows 的 HBITMAP 句柄和 HPALETTE 句柄,用于操纵位图和调色板。TBitMap 对象支持从文件、剪贴板、流中存取位图。

### 1. TBitMap 类的重要属性

(1) Empty 属性: 用于判断位图是否为空。Empty 属性是一个布尔型,如果位图为空则返回 True 值,否则返回 False 值。

(2) Height 属性和 Width 属性: 分别用于设置位图的高度和宽度。这两个属性必须在 TBitMap 组件调入图片后进行设置才能起到限定大小的作用。

(3) Monochrome 属性: 决定位图是单色还是彩色。

TBitMap 组件调入图片后,将 Monochrome 属性设置为 True 后,只要图像中的颜色不是白色的地方都将变成黑色,如果再将 Monochrome 属性设置为 False,图像也不能还原成以前的彩色图像了。

(4) PixelFormat 属性: 用于设置位图的内存格式和颜色深度。设定了如下 9 个用户可选值。

- pfDevice: 与设备有关。
- pf1bit: 1 位颜色。
- pf4bit: 4 位颜色。
- pf8bit: 8 位颜色。
- pf15bit: 15 位颜色。
- pf16bit: 16 位颜色。
- pf24bit: 24 位颜色。
- pf32bit: 32 位颜色。
- pfCustom: 定制颜色。

(5) TransparentColor 属性: 用于设置图像可以透明的颜色。例如要将图像中的黑色部分以透明方式显示,那么 TransparentColor 属性就可以赋值为 clBlack,并将 Transparent 属性设置为 True。



## 2. TBitmap 组件的重要方法

(1) Assign 方法：用于转换图像格式。例如将 TImage 组件调入一个 BMP 格式的图像，可以利用 Assign 方法将 TImage 组件的图像传递给 TBitmap 变量，整个过程如下：

```
var
  BMP:TBitmap;
begin
  BMP := TBitmap.Create;
  BMP.Assign(Image1.Picture.Bitmap);           //对 BMP 进行操作
  BMP.Free;
End;
```

(2) Create 方法：用来创建一个新的 TBitmap 对象。其方法是首先声明一个 TBitmap 变量，然后利用 Create 方法建立，实例参见 Assign 方法中代码的前 4 行。

(3) Free 方法：用来释放 TBitmap 组件。程序运行时建立 TBitmap 组件后会占用一定的内存，如果在使用完 TBitmap 组件后不利用 Free 方法释放，这个 TBitmap 组件会常驻内存，直到程序运行结束，所以提倡在用完此组件后用 Free 方法释放它占用的内存。实例及格式参见 Assign 方法中代码的例数第 2 行。

(4) FreeImage 方法：能够减少位图占用的内存。TBitmap 组件在调入图像后，可以利用此方法减少内存的占用，以后还可以继续使用 TBitmap 组件内的图像。

(5) LoadFromFile 方法：可以在运行时动态地调入扩展名为 .bmp 的图像文件。其语法格式为：

```
Bitmap.LoadFromFile('FileName');
```

(6) Mask 方法：可以把位图变成单色的。Mask 方法中包含一个 TColor 类型的参数 TransparentColor，此参数用来指定可以透明显示的颜色。除 TransparentColor 参数指定的颜色外，其余的都将变成黑色。

(7) SaveToFile 方法：可以将 TBitmap 组件变量中的图像内容保存到硬盘，其格式为 BMP 格式。其语法格式为：

```
Bitmap.SaveToFile('FileName');
```

### 例 7.3 TBitmap 示例：实现图像垂直交错显示效果。

设计构想：将要显示的图形拆成两部分，奇数行像素由上往下搬移，偶数行像素则由下往上搬移，而且两者同时进行。从屏幕上便可看到分别由上下两端出现的较淡图形向屏幕中央移动，直到完全清楚为止。

设计过程：在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令，创建一个新的应用程序。并从工具栏拖曳一个按钮组件 Button1，设置其 Caption 属性为“图像垂直交错显示”。

编写代码如下：

```
//按钮鼠标单击事件
procedure TForm1.Button1Click(Sender: TObject);
var
  SourceBmp:TBitmap;           //原始图像，从磁盘装入
```

```

dhBMP:TBitmap;                                //显示动画用的临时动态变化的图像
i, j, bmpheight, bmpwidth: integer;
begin
    SourceBmp := TBitmap.Create;                //创建原始图像对象
    //从磁盘装入图像文件
    SourceBmp.LoadFromFile('C:\Program Files\Common Files\CodeGear Shared\Images\Splash\
256Color\CHEMICAL.BMP');
    dhBMP := TBitmap.Create;                    //创建时动态变化的图像对象
    dhBMP.height := SourceBmp.Height;           //设置对象高度
    dhBMP.Width := SourceBmp.Width;             //设置对象宽度
    bmpheight := SourceBmp.Height;              //给高度变量赋值
    bmpwidth := SourceBmp.Width;                //给宽度变量赋值
    i := 0;
    while i <= bmpheight do                    //扫描行
    begin
        sleep(10);                             //进程暂停
        j := i;
        while j > 0 do                          //动态复制部分图像
        begin
            dhBMP.Canvas.CopyRect(Rect(0, j - 1, bmpwidth, j), SourceBmp.Canvas,
            Rect(0, bmpheight - i + j - 1, bmpwidth, bmpheight - i + j));
            dhBMP.Canvas.CopyRect(Rect(0, bmpheight - j, bmpwidth, bmpheight - j + 1),
            SourceBmp.Canvas, Rect(0, i - j, bmpwidth, i - j + 1));
            j := j - 2;
        end;
        form1.Canvas.Draw(50, 10, dhBMP);       //在窗体上绘制
        i := i + 2;
    end;
    dhBMP.free;                                 //释放动画临时图像对象
    SourceBmp.free;                             //释放原始图像对象
end;

```

程序保存运行后,单击 Button1 按钮,窗体上分别由上下两端出现的较淡图形向屏幕中央移动,直至整个图像的完整出现。一幅动画帧的运行过程如图 7-10 所示。



图 7-10 TBitmap 实现图像垂直交错显示效果

本程序利用了 Canvas(画布)的 CopyRect 方法,该方法在 Delphi 中定义的原型为:

```
procedure TCanvas.CopyRect(const Dest: TRect; Canvas: TCanvas;  
                           const Source: TRect);
```

它的主要功能是将源画布某一矩形区域的图像复制到另一个画布的矩形区域。由于是内存的成块复制,因此具有很高的执行效率。实际上读者可以模仿本例,查阅相关资料,使用一定的图像成形原理设计出各种美观演示效果,如百叶窗、推拉、马赛克、随机线和反像等。

### 7.2.3 TImage 组件

TImage 组件位于 Additional 页上,它的主要作用是装入并显示一个图形或图像文件。例如,要在窗口上显示一个图像,需要把 TImage 组件加到 Form 上,图像的格式可以是 JPG、BMP 和 ICO 等,显示的图像由 Picture 属性指定。

TImage 组件的主要功能是显示图像、美化界面。TImage 组件经常和 TPanel 组件结合使用,以 TPanel 组件的边框来划分 TImage 组件的边界。

在设计时,单击 Object Inspector 中 Picture 属性右侧的“...”按钮打开一个对话框,再单击 Load 按钮可以浏览方式选择某个图形或图像文件。Stretch 属性为 Boolean 型,它的值为 false 时,图形、图像文件按照其固有的尺寸显示;它的值为 true 时,则对图形、图像进行缩放,使其恰好充满 Image 的区域。TImage 对象还具有 Canvas 属性,所以可以在已装入的图片上绘画。

#### 1. TImage 组件的重要属性

(1) AutoSize 属性:如果此属性设为 True,TImage 组件将自动调整尺寸,以适应图像的大小;如果此属性设为 False,TImage 组件的尺寸不变。当图像的尺寸大于 TImage 组件的尺寸时,图像的一部分将被裁剪。

(2) Picture 属性:用来指定 TImage 组件上要显示的图像。可以单击 Picture 属性右侧的“...”按钮打开 PictureEditor 对话框,然后单击 Load 按钮调入图像文件。Picture 属性下还包含多个方法,例如打开图片的方法:

```
Image1.Picture.LoadFromFile(const Filename:String);
```

保存图像的方法:

```
Image1.Picture.SaveToFile(const Filename:String);
```

(3) Proportional 属性:此属性设置为 True 值时,将图像按原来的长宽比例进行调整,如果 TImage 组件不能够完全显示图像,图像将按比例进行缩放,并使 TImage 组件显示全部图像;如果 TImage 组件已经将图像全部显示,在设置此属性时,图像大小将不发生变化。

(4) Stretch 属性:此属性设置为 True 值时,不管图像的大小与长度的比较,只是将图像填充整个 TImage 组件。

(5) Transparent 属性:如果此属性设置为 True,图像就是透明的,透过图像的空白处可以看到 TImage 组件所在的背景。这个属性只适合用于图像是 BMP 格式的情况。图像组件(TImage)要在窗口上显示一个图像,需要把 TImage 组件加到 Form 上,图像的格式可以是 JPG、BMP 和 ICO 等,显示的图像由 Picture 属性指定。

## 2. TImage 组件的重要方法

(1) LoadFromFile 方法：在程序中用 Picture.LoadFromFile() 方法进行调入磁盘文件图像。

(2) SaveToFile 方法：利用 Picture.SaveToFile() 方法可以将 TImage 组件的当前显示内容保存为一个图像文件。

下面举一个简单的例子体会一下 TImage 组件的用法。

### 例 7.4 TImage 组件动画示例：图形整体拉出效果。

设计过程：在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令，创建一个新的应用程序。并从工具栏拖曳一个图像组件 Image1，设置其 Picture 属性，从磁盘上装入一幅 Delphi 2007 软件自带的位图，比如 C:\Program Files\Common Files\CodeGear Shared\Images\Splash\256Color 路径下的 SHIPPING.BMP。设置 Image1 的高度为 233，其宽度为 300。最后将 Image1 的 Stretch 属性值设置为 True。然后从组件面板的 System 页面给窗体 Form1 添加 Timer1 组件，将 Timer1 的定时间隔 Interval 属性设为 100，定时使能 Enabled 属性设为 True。

调整组件位置后，窗体设计界面如图 7-11 所示。



图 7-11 使用 TImage 实现图形整体拉出效果界面设计

编写代码如下：

```
//窗体创建时事件
procedure TForm1.FormCreate(Sender: TObject);
begin
    Image1.Width := 0; //设置图片显示宽度为 0, 为动画做准备
end;

//定时器定时事件
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    //设置增量
    Image1.Width := Image1.Width + 10; //每次使得图片宽度增加 10
    if image1.Width = 300 then //图形整体拉出完毕
        Timer1.Enabled := FALSE; //动画停止
end;
```

程序保存运行后,窗体中间的轮船图像由部分逐步变大,直至整体出现。本例的整体拉出动画效果比较简单,主要是动态地改变图形区域的大小就可以实现。

#### 7.2.4 TImageList 组件

当要为菜单、工具栏和树型视图等加入位图或图标时,常常需要 ImageList 组件,它位于组件面板的 Win32 页上。可使用 ImageList Editor 在设计时把位图和图标加到 ImageList 组件中。双击 ImageList 组件,或右击 ImageList 组件,再执行快捷菜单上的 ImageList Editor 命令,可以打开 ImageList Editor 窗口。

在一个 ImageList 组件中可以存放多个同样尺寸的图像(或图标),每个图像都使用一个索引号。ImageList Editor 中使用 Add 或 Delete 按钮可分别用来增加或删除 ImageList 中的图像。使用 Win32 页上的 CoolBar 组件设计工具栏时,需要在 CoolBar 组件内含的 ImageList 中添加工具按钮的图像,ImageList 之内图像的个数必须与按钮的个数相一致。

#### 7.2.5 TShape 组件

TShape 组件位于 Additional 页上,常被用来显示简单的图形。TShape 最基本的用途是在设计阶段产生一些简单图形装饰窗体。当然,TShape 组件也能在程序运行时动态创建和移动。由于 TShape 组件较简单,因此使用时较少消耗系统资源。

TShape 组件的一些属性如下:

- (1) Brush 属性:用于设置几何图形内部的填充特性,包括填充的颜色和图案。
- (2) Pen 属性:用于设置画笔的属性,包括画笔的颜色、线型和宽度等。
- (3) Shape 属性:用于指定要显示的几何图形的种类。

Shape 属性可能的取值为 stCircle(圆)、stEllipse(椭圆)、stRectangle(矩形)、stRoundRect(圆角矩形)、stRoundSquare(圆角方形)和 stSquare(正方形)。

#### 7.2.6 TPaintBox 组件

TPaintBox 组件位于组件面板的 System 页上,主要用于用户自定义绘图,没有处理图片的能力,所以只包含了 TCanvas。它也是从 TControl、TGraphicControl 继承,是能够交互的控件。

前面讲过 TImage 主要是为了显示图片,它主要包含的是 TPicture,具备了控件的基本能力(事件、消息等)。很显然,TImage 比 TPaintBox 的能力强大。但仅就绘图来讲,还是 TPaintBox 速度快,并且基本没有延迟,可以做到无闪烁。

TPaintBox 的属性中经常用到的只有 Canvas。Canvas 提供了绘制和修饰位图的所有方法和工具。在 TPaintBox 的方法中,只有 Paint 比较重要。此方法使 TPaintBox 对自己上面的图像进行强制更新。TPaintBox 的事件响应函数有 OnPaint,在每次 WM-Paint 消息发来时这个函数被调用,从而使其上面的图像被更新。

### 7.3 图形图像组件重绘和鼠标事件

#### 7.3.1 处理绘图组件的 OnPaint 事件

由于 Windows 具有多窗口、多任务的特性,因此任何窗口的画面都可能暂时地被其他

窗口遮盖。当被挡住的部分需要再次显示时必须重新绘制窗口,该项工作通常由 Windows 自动完成。但 Windows 重绘窗口时无法恢复用户在窗体上绘制的图形,它能做到的只是给用户发一个消息,该消息会触发 Delphi 窗体或者绘图组件的 OnPaint 事件,因此用户可在 OnPaint 事件中重画窗体。

为了进一步说明上述关系,做一个实验:在 Form1 上放一个按钮和一个 PaintBox1,单击该按钮时执行利用 PaintBox 画圆角矩形、椭圆和矩形示例。其代码为:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    //位于 PaintBox 表面绘画
    PaintBox1.Canvas.Pen.Color := clGreen;
    PaintBox1.Canvas.Pen.Width := 6;
    PaintBox1.Canvas.Brush.Color := clRed;
    PaintBox1.Canvas.Brush.Style := bsCross;
    PaintBox1.Canvas.Rectangle(20, 20, 220, 180);
    PaintBox1.Canvas.Brush.Color := clYellow;
    PaintBox1.Canvas.Ellipse(60, 60, 260, 200);
    PaintBox1.Canvas.Brush.Color := clBlue;
    PaintBox1.Canvas.RoundRect(230, 20, 320, 180, 15, 15);
end;
```

图 7-12 为该程序运行时单击 Button1 后出现的画面。假定随后使用“计算器”操作,使 Form1 窗体的 PaintBox1 的图形表面部分被遮挡,如图 7-13 所示。

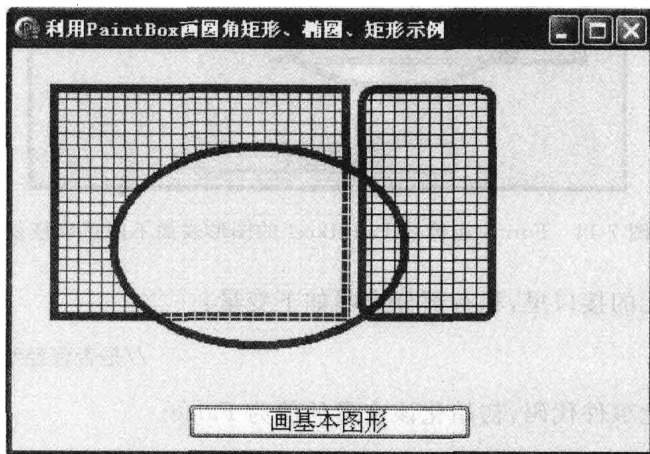


图 7-12 PaintBox1 表面绘制图形后的运行效果

当将“计算器”窗口关闭后,Form1 再次成为活动窗口,此时的画面如图 7-14 所示,窗体的 PaintBox1 的图形未能正确恢复。

如果在 OnPaint 中写下与 Button1 的 OnClick 事件中同样的绘图语句,则不会出现上述情况。但是只要程序一运行,还未单击 Button1 按钮就已经完成绘图。较好的解决方法是将绘图语句写入一个子程序,并用一个布尔型变量记录是否曾经单击 Button1 绘制过图形。下面列出修改后 unit1.pas 文件中部分要点供读者参考。

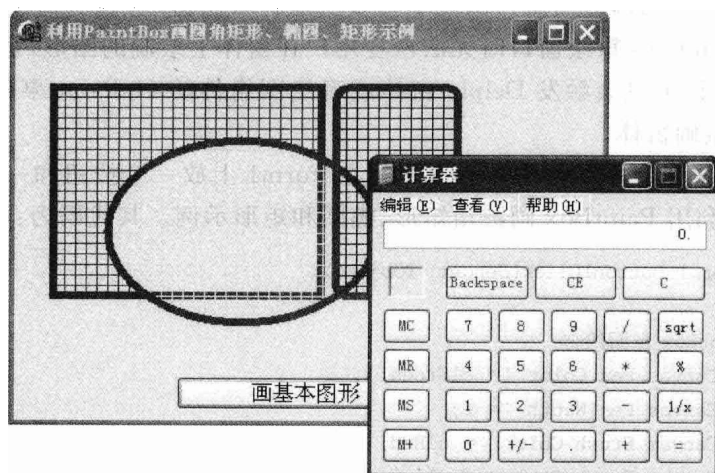


图 7-13 Form1 窗体的 PaintBox1 的图形表面部分被遮挡

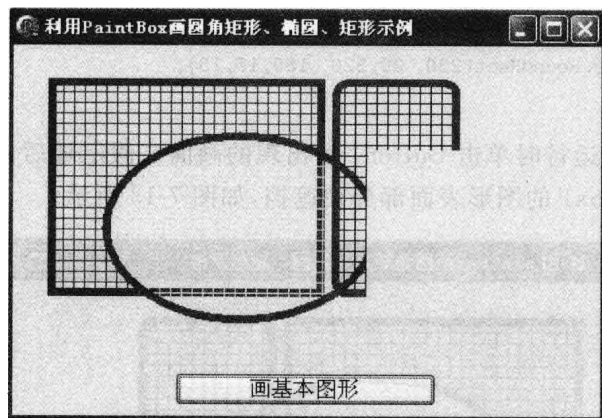


图 7-14 Form1 窗体的 PaintBox1 的图形表面不能正确恢复

(1) 在窗体单元的接口里,私有部分声明如下变量:

```
fHasPic:Boolean; //是否曾经绘制过图形
```

(2) 窗体初始化事件代码,初始化该变量的值为 False。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    fHasPic := False; //PaintBox1 表示一开始没有图形
end;
```

(3) 在 Button1 的鼠标 OnClick 事件代码中加入一行代码,表明已经绘制过图形。

```
fHasPic := True; //PaintBox1 表面已经有图形
```

(4) 给 PaintBox1 添加重绘事件代码如下:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
```

```

if fHasPic then                                //如果 PaintBox1 表面已经有图形
    Button1Click(nil);                          //则调用 Button1 事件处理程序重新绘制图形
end;

```

### 7.3.2 图形组件鼠标事件

绘图程序中用户需要知道鼠标指针的位置和鼠标按钮的状态,并且需要处理鼠标的输入事件才能实现图形图像正常绘制工作。

用户通过图形图像组件上的 OnMouseUp、OnMouseMove 和 OnMouseDown 事件可以精确地控制鼠标。绘制曲线或者图形的基本原理是:程序必须始终跟踪鼠标按钮的位置与状态。若按钮处于按下状态,并且移动了指针,则必须用画笔作出适当绘画动作。

**例 7.5** 使用图形组件鼠标事件处理技术设计开发简单的绘图工具系统。

程序的设计步骤如下:

(1) 新建一个 VCL Form 应用程序。

选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 为“使用图形组件鼠标事件处理技术设计开发简单的绘图工具系统示例”。

(2) 定制窗体:向窗体添加组件并设置相关的属性。

① 先向窗体添加一个主菜单组件 MainMenu1,用于打开、保存图形文件,并退出应用程序。通过菜单编辑器设计各菜单项,在“文件”菜单中包含“新建”、“打开”和“保存”三个菜单项;再加入一个“退出”菜单。

② 设计工具栏,该工具栏在主窗口底侧显示,可用于选择绘制对象如直线、矩形和带填充色矩形等,也可设置线条和填充颜色。

向窗体添加 TPanel 组件(Panell)。TPanel 组件是一种容器,主要用于窗口的排版布局,并可增加界面的美观性。设置 Panell 组件的 Align 属性为 alBottom,将其 Caption 属性置为空,然后向 Panell 组件中添加用于显示或者设置画笔颜色的 Shapel 组件及用于显示或者设置画刷颜色的 Shape2 组件。

向 Panell 组件中还添加如下组件:

```

RadioGroup2: TRadioGroup;                      //画刷的填充样式选择单选框
RadioGroup3: TRadioGroup;                      //要绘制的图形,矩形、椭圆、圆或者圆角矩形与直线
Label1: TLabel;                                //线宽标签
SpinEdit1: TSpinEdit;                          //线框调整编辑框

```

③ 向窗体添加 TColorDialog 对话框组件,用于画笔或者画刷颜色的选择设置。

④ 在窗体上添加 TPanel 组件(Panell2),设置 Panell2 组件的 Align 属性为 alClient,将其 Caption 属性置为空,然后向 Panell2 组件中添加一个 TImage 组件(Image1),设置该组件的 Align 属性为 alClient。

⑤ 向窗体添加 TOpenPictureDialog 和 TSavePictureDialog 两个公用对话框组件,分别用于打开和保存图形文件。

调整好各组件位置、大小及辅助其他属性后,窗体界面如图 7-15 所示。

(3) 程序代码编辑:主要编写菜单单击事件和图形图像组件鼠标事件代码。

① 窗体私有变量的定义。



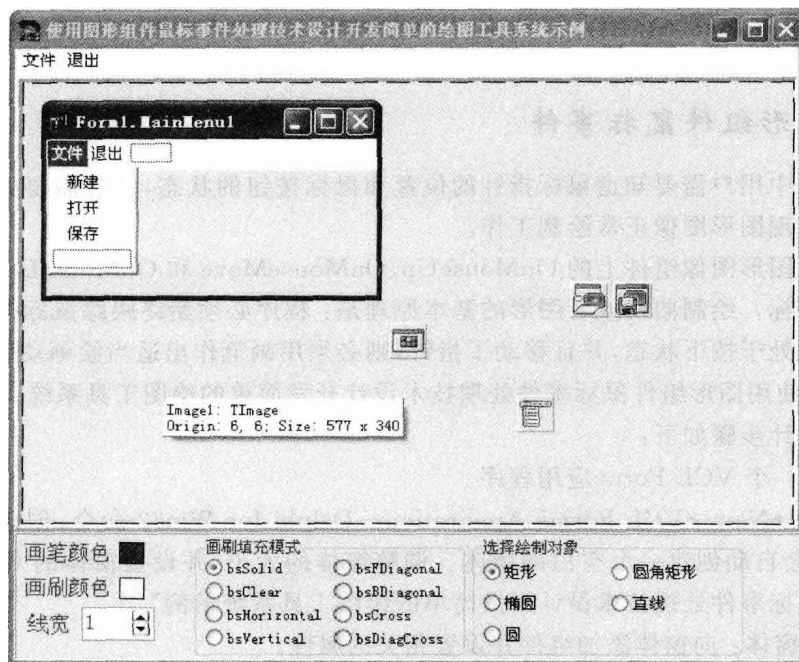


图 7-15 使用 TImage 实现绘图工具系统界面设计

在 Form1 窗体单元 Unit1 的接口私有部分必须设置以下变量,它们的名称与含义如下:

```
{ Private declarations }
fHasPic:Boolean;           //鼠标移动时曾经画过图
fMouseDown:Boolean;       //鼠标是否处于按下状态
fX,fY:Integer;            //鼠标按下初始坐标
fBx,fBy:integer;          //鼠标动态移动坐标
```

## ② 主菜单功能代码如下:

```
//退出菜单事件
procedure TForm1.N_ExitClick(Sender: TObject);
begin
    Close;                //关闭窗口体
end;
//新建菜单事件
procedure TForm1.N_NewClick(Sender: TObject);
begin
    //使其呈现白色画布
    Image1.Canvas.Brush.Color := clWhite;
    Image1.Canvas.Brush.Style := bsSolid;
    Image1.Canvas.FillRect(Rect(0,0, image1.Width, image1.Height));
end;
//打开图形文件菜单事件
procedure TForm1.N_OpenClick(Sender: TObject);
begin
```

```

//打开图形图像对话框执行
if OpenPictureDialog1.Execute then
begin
    //装入一幅已经存在的 bmp 图形
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
end;
//保存图像菜单事件
procedure TForm1.N_SaveClick(Sender: TObject);
begin
    //保存图形图像对话框执行
    if SavePictureDialog1.Execute then
    begin
        //保存当前编辑的图形为 bmp 图形文件
        Image1.Picture.SaveToFile(SavePictureDialog1.FileName);
    end;
end;

```

### ③ 窗体创建时的 OnCreate 事件代码如下：

```

//窗体初始化事件
procedure TForm1.FormCreate(Sender: TObject);
begin
    N_NewClick(nil);                //初始化呈现白色画布
    fMouseDown := False;            //初始化,鼠标处于没有被按下状态
end;

```

④ 为画笔、画刷显示设置颜色的事件代码,Shape1 与 Shape2 选择同一个事件处理程序,这样可以节省编程的工作量。

```

//设置颜色鼠标按下事件
procedure TForm1.Shape1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    //将当前对象的颜色赋值给"颜色"对话框
    ColorDialog1.Color := (Sender as TShape).Brush.Color;
    //如果"颜色"对话框执行
    if ColorDialog1.Execute then
        //将用户选择的色彩设置到对象上
        (Sender as TShape).Brush.Color := ColorDialog1.Color;
end;

```

⑤ 最核心的是 Image1 的 OnMouseDown、OnMouseMove 和 OnMouseUp 事件的代码。

```

//图形组件鼠标按下事件,表示本次绘图开始
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    fMouseDown := True;                //设置鼠标按下状态
    fX := X; fY := Y;                //获取鼠标初始坐标
    with Image1 do

```

```

begin
    Canvas.Pen.Color := Shape1.Brush.Color; //设置临时边框颜色
    Canvas.Brush.Color := Shape2.Brush.Color; //设置临时填充颜色
    Canvas.Pen.Mode := pmXOR; //画笔颜色跟屏幕异或,为了方便擦除临时图形
    Canvas.Brush.Style := TBrushStyle(RadioGroup2.ItemIndex); //获取填充风格
    Canvas.Pen.Width := SpinEdit1.Value; //设置线框
    if RadioGroup3.ItemIndex = 4 then
    begin
        Canvas.Pen.Mode := pmCopy; //设置最终画笔模式
        canvas.MoveTo(x, y); //为画线做准备
    end;
end;
fHasPic := False; //一开始没有画图
end;
//图形组件鼠标移动事件,如果鼠标同时按下,表示绘图进行时
procedure TForm1.Img1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    if fMouseDown then //如果鼠标按下并且移动
    begin
        with Img1 do
        begin
            if fHasPic then //如果有动态临时图形
            begin //擦除旧图形
                case RadioGroup3.ItemIndex of
                    0: canvas.Rectangle(fx, fy, fbx, fby); //矩形
                    1: canvas.Ellipse(fx, fy, fbx, fby); //椭圆
                    2: canvas.Ellipse(fx, fy, fbx, fy + fbx - fx); //圆
                    3: canvas.RoundRect(fx, fy, fbx, fby, 15, 15); //圆矩形
                end;
            end;
            case RadioGroup3.ItemIndex of //给本次鼠标移动的位置画图
                0: canvas.Rectangle(fx, fy, x, y); //矩形
                1: canvas.Ellipse(fx, fy, x, y); //椭圆
                2: canvas.Ellipse(fx, fy, x, fy + x - fx); //圆
                3: canvas.RoundRect(fx, fy, x, y, 15, 15); //圆矩形
                4: canvas.LineTo(x, y); //直线
            end;
        end;
        fbx := x; fby := y; //保存当前坐标,为下一次画出图形用
        fHasPic := true; //表示已经有动态临时图形
    end;
end;
//图像组件鼠标弹起事件,表明本次绘图结束
procedure TForm1.Img1MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    with Img1 do
    begin
        if fHasPic then //如果有动态临时图形
        begin //擦除旧图形

```

```

case RadioGroup3.ItemIndex of
    0: canvas.Rectangle(fx, fy, fbx, fby);           //矩形
    1: canvas.Ellipse(fx, fy, fbx, fby);           //椭圆
    2: canvas.Ellipse(fx, fy, fbx, fy + fbx - fx); //圆
    3: canvas.RoundRect(fx, fy, fbx, fby, 15, 15); //圆矩形
end;

end;

Canvas.Pen.Mode := pmCopy;                        //设置最终画笔模式
//绘制最终的图形
case RadioGroup3.ItemIndex of
    0: canvas.Rectangle(fx, fy, x, y);             //矩形
    1: canvas.Ellipse(fx, fy, x, y);             //椭圆
    2: canvas.Ellipse(fx, fy, x, fy + x - fx);    //圆
    3: canvas.RoundRect(fx, fy, x, y, 15, 15);    //圆矩形
end;

end;

fMouseDown := False;                             //表示鼠标已经弹起,本次鼠标操作结束
end;

```

#### (4) 系统运行测试。

选择 File→Save 命令,保存单元文件与项目文件。

运行该程序,通过菜单可以打开一幅 bmp 图形,然后可以使用自定义的绘图工具通过鼠标操作在图像上绘制各种大小图形,可以用直线工具写字,可以通过“颜色”对话框更改画笔颜色。其运行效果如图 7-16 所示。最后可以通过菜单将编辑后的图像保存到磁盘 bmp 格式的文件。

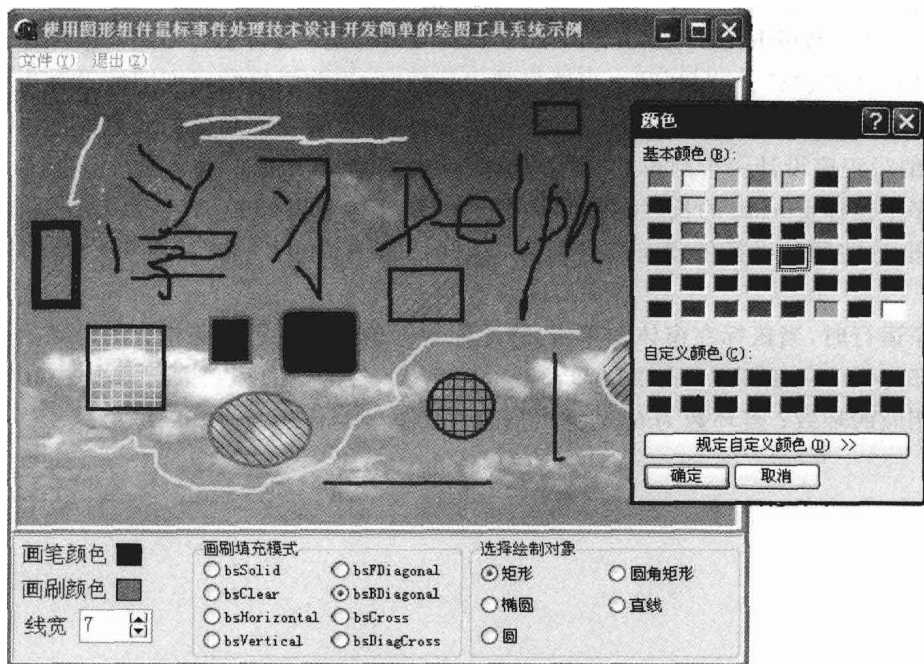


图 7-16 简单绘图工具系统示例运行效果

程序中 OnMouseDown、OnMouseMove 和 OnMouseUp 三个事件的参数表内都有 X 和 Y 两个整型变量,它们代表鼠标指针的当前位置。例如,选择画线时,OnMouseDown 事件发生时,使变量 fMouseDown 为 true,并调用 MoveTo 方法使画笔的位置与鼠标指针的位置一致。OnMouseMove 发生时根据变量 fMouseDown 的值区分两种情况:一种情况为未按下鼠标,此时不需要画线;另一种情况则需要调用 LineTo 方法移动画笔的位置到当前鼠标位置并画线。

## 本章小结

本章介绍了 Delphi 中常用图形与图像组件的属性、方法、事件和操作使用,重点是绘图相关的对象概念和操作使用,如 TCanvas(画布)、TPen(画笔)和 TBrush(画刷)类等。本章难点是对图形图像组件重绘和鼠标事件的理解与掌握。

通过本章学习,读者应该能够模仿实例快速地掌握 Delphi 基本的图形图像应用程序的设计与开发方法。

## 思考与练习

1. 简述 TCanvas(画布)组件的主要属性与方法。
2. 根据本章的例子,设计一个简单绘图程序,要求如下:

当用户用鼠标接连两次单击窗体时,在窗体 Canvas(画布)上两次单击的位置之间连一条直线,该直线是使用 Canvas 的 Pen(画笔)属性绘制的。给 Form1 加入一个主菜单,它有三个选项:Pen Width、Pen Color 和 Exit。Pen Width 是一个下拉子菜单,它含有 5 个菜单命令项:“1”、“2”、“3”、“5”和“8”。当选择其中某项时,当前画笔的宽度即被设置为相应的数值。Pen Color 菜单项被选中时,可打开“颜色”对话框设置当前画笔的颜色。

3. 编写程序设计一个简单的“射击”模拟游戏,其要求如下:

程序主窗体 Form1 的长度和宽度都是 600,在窗体上以(300,300)为圆心画 10 个绿色的同心圆,这些圆的半径依此递增分别为 10,20,30,...,100。将最小的那个圆的内部填充为红色。

程序运行时,当鼠标在窗体上单击,程序能根据单击的位置判定相应的环数:在红色小圆圈内部为 10 环,该圆以外依次为 9,8,...,1,0 环。若击中的环数大于 0,则应在窗体标题上显示相应的环数;若环数为 0,则弹出信息对话框“哦,你还需要多加练习!”。

## 第 8 章

## 动画与多媒体程序设计

动画与多媒体技术使计算机系统具有综合处理文字、语音、图像、图形和视频信息的能力,其丰富的多媒体信息、方便的交互性和实时性改善了人机界面,改善了计算机的使用方式,丰富了计算机的应用领域。

Delphi 利用 Windows API 函数对多媒体的强大支持使复杂的多媒体技术变得驾轻就熟。此外,还可以使用其 VCL 中的 TMediaPlayer 等组件对多媒体高效便捷的编程提供强烈支持。Delphi 还可以使用第三方的组件如 flash 相关的组件进行丰富多彩的多媒体动画程序设计。

本章主要包括以下内容:

- 声音播放程序设计;
- TAnimate 组件动画程序设计;
- 多媒体播放程序设计;
- Flash 动画播放程序设计。

### 8.1 声音播放程序设计

声音信息在存储时是按一定格式存储的,在不同的操作系统和不同的软、硬件环境下有不同的存储格式。最常见的格式主要是 WAV 文件,其扩展名为 .wav,是 Windows 所使用的标准数字音频格式,称为波形文件。WAV 文件格式支持存储各种采样频率和样本精度的声音数据,并支持音频数据的压缩,其缺点是产生的文件过大,相应的所需存储空间大,不适合长时间的记录,必须采用适当的方法进行压缩处理。

#### 8.1.1 Windows 的默认声音

Windows 运行过程中,每当启动、关机、提示用户出现了错误或执行了说明特殊事件时,会发出一个声音,这些声音与系统事件的对应关系可在“控制面板”中设置,如图 8-1 所示。

在 Delphi 中,可以编写程序,通过调用 MessageBeep 函数来产生这些 Windows 声音。该函数的原型如下:

```
Function MessageBeep(uType: DWord): Boolean;
```

该函数只使用一个整型的参数,执行该函数时系统根据不同的常数发出不同的声音,也就是调用了不同的系统 wav 文件。

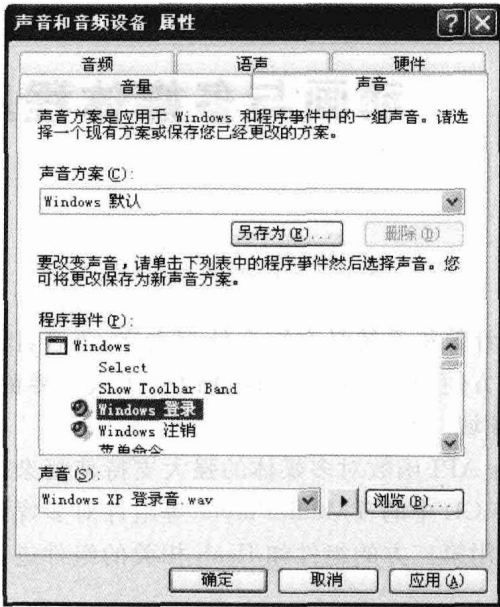


图 8-1 Windows 系统通过对话框设置声音参数

表 8-1 列出了该参数中可接受的预定义常量名及相应的含义。

表 8-1 Windows 系统声音

声音常量名称	声音含义
MB_ICONASTERISK=0x00000040L	SystemAsterisk
MB_ICONEXCLAMATION=0x00000030L	SystemExclamation
MB_ICONHAND=0x00000010L	SystemHand
MB_ICONQUESTION=0x00000020L	SystemQuestion
MB_OK= 0x00000000L	SystemDefault

**例 8.1** 使用 Delphi 编程实现对 Windwos 系统默认各种提示声音播放。

设计过程：在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,并从组件面板上拖曳一个单选框组件 RadioGroup1,设置其 item 属性,输入 Windows 系统声音各项名称。然后从组件面板上拖曳一个按钮组件 Button1,并设置其 Caption 属性为“播放 Windows 声音”。

调整组件位置与大小后,窗体设计界面如图 8-2 所示。

该程序很简单,下面列出的是为按钮 Button1 的 OnClick 事件编写的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var SoundType:DWord;
begin
    case RadioGroup1.ItemIndex of
        0: SoundType := MB_ICONASTERISK;    //( SystemAsterisk )
        1: SoundType := MB_ICONEXCLAMATION; //( SystemExclamation )
        2: SoundType := MB_ICONHAND;        //(SystemHand )
        3: SoundType := MB_ICONQUESTION;    //(SystemQuestion)
```

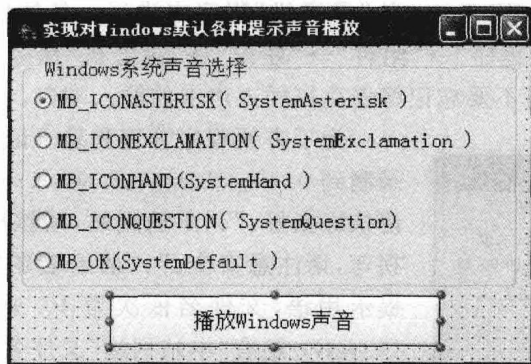


图 8-2 实现对 Windows 默认各种提示声音播放窗体界面设计

```
4:      SountType := MB_OK;                      //(SystemDefault )
end;
MessageBeep(SountType);                        //调用函数播放声音
end;
```

程序保存运行后,用户可以选择任何一个声音选项,然后单击发声。

### 8.1.2 使用 API 函数播放 wav 声音文件

Delphi 可以调用 Win32 API(Application Programming Interface)函数播放声音. wav 文件,Win32 API 也就是 Microsoft Windows 32 位平台的应用程序编程接口。

Delphi 类库和组件都是构架在 Win32 API 函数基础之上的,是封装了的 API 函数的集合。如果要开发出更灵活、更实用、更具效率的应用程序,必然要涉及直接使用 API 函数。虽然类库和组件使应用程序的开发简单得多,但它们只提供 Windows 的一般功能,对于比较复杂和特殊的功能来说,使用类库和控件是非常难以实现的,这时就需要采用 API 函数来实现。

波形声音文件(.wav)可以保存讲话、乐曲及模拟自然界存在的各种声音,该文件格式下的声音保真性较好。Delphi 可用 Win32 API 库中的 PlaySound 函数播放. wav 文件,该 API 函数在 Delphi 中声明如下:

```
function PlaySound(pszSound: PChar; hmod: HMODULE; fdwSound: DWORD): BOOL; stdcall;
```

它有三个参数,适用于各种调用方式。读者可在有关的帮助中查到所有的用法,这里介绍较常用的一种方式。在该方式中,pszSound 是一个波形声音文件的文件名,它必须用 PChar 格式的字符串;这里的 hmod 可以设置为 NULL; fdwSound 的值为常数 SND\_FILENAME。函数的返回值为布尔类型即 BOOL 型,表示是否成功地打开了声音文件。

要调用该 API 函数,需要将其库 MMSystem.dll 单元名称使用 uses 引用到用户代码中,即用户代码接口部分或者实现部分必须有下面的语句,系统才能成功编译、运行。

```
Uses MMSystem;
```

**例 8.2** 使用 Delphi 编程实现对模拟公共汽车自动或者手动语音报站系统。

素材准备:要实现语音报站,就要有这些站的语音文件,那么首先要录制若干个站名的



声音文件,这项工作可在 Windows 的“录音机”程序中进行。具体操作为:在 Windows 系统中选择“开始”→“所有程序”→“附件”→“娱乐”→“录音机”命令,打开“声音-录音机”窗口,如图 8-3 所示,操作时不要忘记将麦克风插入声卡的输入端口。



图 8-3 使用 Windows 的录音机录制声音素材

为了方便程序管理和易于记忆,按顺序将 10 个站名录制到 0. wav、1. wav、2. wav、...、09. wav;“车辆启动,请扶好坐好。”、“车辆到站,请按顺序依次下车。”、“车辆拐弯,请注意安全。”、“请给老弱病残幼让座。”4 个友情提示用语,文件名依次为 10. wav、11. wav、12. wav 和 13. wav;还有“本站到达”文件名为 14. wav,“下一站”文件名为 15. wav。这些文件录制完成后可保存到程序所在的子目录。

设计过程:在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序。并从组件面板上拖曳一个单选框组件 RadioGroup1,设置其 item 属性,输入 4 个友情提示用语;一个 ListBox 组件用于存放站名;一个单选框组件 RadioGroup2,设置其 item 属性,输入车辆两个方向文本;一个复选框用于自动、手动切换;还有两个按钮:Button1 用于手动语音报站,Button2 用于手动播放友情提示;一个 Panel1 用于容纳行驶倒计时的 SpinEdit2 组件对象。两个标签用于显示系统信息。最后还有两个定时器组件行驶控制 Timer1 和到站等待时间 Timer2。初始设置 Timer1 的 Enabled 为 True,Timer2 的 Enabled 为 False。系统假定每站等待时间为 3s,设置 Timer2 的 Interval 为 3000ms。

调整所有组件位置与大小等属性合适后,窗体设计界面如图 8-4 所示。

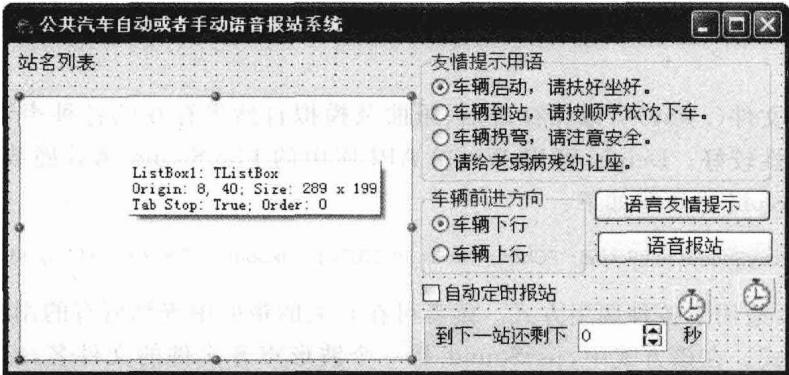


图 8-4 公共汽车自动或手动语音报站系统设计界面

代码编辑如下:

(1) 进入窗体的代码文件 Unit1. pas 的编辑窗口,首先将 MmSystem 引用到接口部分,然后在 Form1 窗体类代码的私有部分声明如下变量和过程:

```
private
{ Private declarations }
StationName:Array[0..9] of String;           //站的名称
```

```

StationTime:Array[0..8] of Integer;      //站的时间间隔
procedure fRefreshList;                  //生成站名列表
procedure fRefreshTime;                  //生成时间间隔

```

这两个过程的代码如下：

```

procedure TForm1.fRefreshList;            //生成站名列表
var i:integer;                            //定义循环变量
begin
    ListBox1.Clear;                      //清除列表
    if RadioGroup2.ItemIndex = 0 then    //车辆下行
    begin
        for i := 0 to 8 do
            ListBox1.Items.Add(StationName[i] + '(到下一站还剩下: ' + inttostr(StationTime
[i])) + '秒)');
            ListBox1.Items.Add(StationName[9] + '(本站为最后一站)');
            ListBox1.ItemIndex := 0;      //当前站号
        end
    else
    begin
        ListBox1.Items.Add(StationName[0] + '(本站为最后一站)');
        for i := 1 to 9 do
            ListBox1.Items.Add(StationName[i] + '(到下一站还剩下: ' + inttostr(StationTime
[i-1])) + '秒)');
            ListBox1.ItemIndex := 9;      //当前站号
        end;
    end;
end;
procedure TForm1.fRefreshTime;            //生成时间间隔
begin
    if RadioGroup2.ItemIndex = 0 then    //车辆下行
    begin
        if ListBox1.ItemIndex <= 8 then
        begin
            SpinEdit2.Value := StationTime[ListBox1.ItemIndex];
            Label1.Caption := '行驶中: 从"' + StationName[ListBox1.ItemIndex] + '" 到"' +
StationName[ListBox1.ItemIndex + 1] + '"';
        end
        else
            SpinEdit2.Value := StationTime[8];
        end
    else
    begin
        if ListBox1.ItemIndex > 0 then
        begin
            SpinEdit2.Value := StationTime[ListBox1.ItemIndex - 1];
            Label1.Caption := '行驶中: 从"' + StationName[ListBox1.ItemIndex] + '" 到"' +
StationName[ListBox1.ItemIndex - 1] + '"';
        end
        else
            SpinEdit2.Value := StationTime[0];
        end;
    end;
end;

```

```
end;
```

## (2) 窗体初始化代码:

```
procedure TForm1.FormCreate(Sender: TObject);
var i:integer;
begin
    StationName[0] := '西安路站';
    StationName[1] := '上海路站';
    StationName[2] := '北京路站';
    StationName[3] := '西光小学站';
    StationName[4] := '清华大学站';
    StationName[5] := '广州路站';
    StationName[6] := '兴庆宫公园站';
    StationName[7] := '动物园站';
    StationName[8] := '海洋公园站';
    StationName[9] := '大唐芙蓉园站';
    StationTime[0] := 255;
    StationTime[1] := 268;
    StationTime[2] := 366;
    StationTime[3] := 255;
    StationTime[4] := 300;
    StationTime[5] := 100;
    StationTime[6] := 222;
    StationTime[7] := 600;
    StationTime[8] := 90;
    fRefreshList;           //生成站名列表
    fRefreshTime;           //生成时间间隔
end;
```

## (3) 手动播放语音代码:

```
//手动报站
procedure TForm1.Button1Click(Sender: TObject);
var sn:String;
begin
    sn := inttostr(ListBox1.ItemIndex) + '.wav';
    PlaySound (PChar(sn), 0, SND_FILENAME);    //调用 API 函数
end;
//播放语言友情提示
procedure TForm1.Button2Click(Sender: TObject);
var sn:String;
begin
    sn := inttostr(RadioGroup1.ItemIndex + 10) + '.wav';
    PlaySound (PChar(sn), 0, SND_FILENAME);    //调用 API 函数
end;
```

## (4) 站方向变化控制代码:

```
procedure TForm1.RadioGroup2Click(Sender: TObject);
var curr:integer;
begin
    //车辆掉头
```

```

curr := ListBox1.ItemIndex;           //保存当前站号
fRefreshList;                         //生成站名列表
ListBox1.ItemIndex := Curr;          //设置当前站号
fRefreshTime;                        //生成时间间隔
end;

```

(5) 自动行驶控制和自动等待以及自动语音报站代码编写如下:

//行驶计时定时器时间到

```

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    if not CheckBox1.Checked then Exit;           //如果没有设置自动报站,退出
    SpinEdit2.Value := SpinEdit2.Value - 1;      //减去 1s
    if SpinEdit2.Value > 0 then Exit;             //如果还没到站
    if RadioGroup2.ItemIndex = 0 then             //车辆下行
    begin
        if ListBox1.ItemIndex = 9 then           //最后一站
        begin
            RadioGroup2.ItemIndex := 1;          //车辆掉头
            RadioGroup2Click(nil);
        end
        else
        begin
            ListBox1.ItemIndex := ListBox1.ItemIndex + 1;
            fRefreshTime;                        //生成时间间隔
        end;
    end
    else
    begin
        if ListBox1.ItemIndex = 0 then           //最后一站
        begin
            RadioGroup2.ItemIndex := 0;          //车辆掉头
            RadioGroup2Click(nil);
        end
        else
        begin
            ListBox1.ItemIndex := ListBox1.ItemIndex - 1;
            fRefreshTime;                        //生成时间间隔
        end;
    end;
    Label1.Caption := '车辆到达: ' + StationName[ListBox1.ItemIndex];
    PlaySound ('14.wav', 0, SND_FILENAME);       //本站到达
    PlaySound (PChar(inttostr(ListBox1.ItemIndex) + '.wav'), 0, SND_FILENAME);
    PlaySound ('11.wav', 0, SND_FILENAME);       //友情提示用语
    Timer2.Enabled := true;
    Timer1.Enabled := False;
end;
//到站等待定时器时间到
procedure TForm1.Timer2Timer(Sender: TObject);
begin
    Timer2.Enabled := False;

```

```
Timer1.Enabled := True;
fRefreshTime;                                //生成时间间隔
PlaySound ('10.wav', 0, SND_FILENAME);       //调用 API 函数
end;
```

选择 File→Save 命令,保存单元文件与项目文件。然后运行,实例效果如图 8-5 所示。

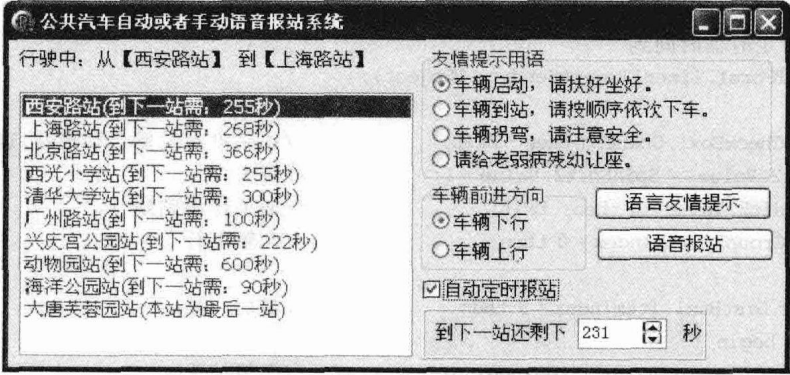


图 8-5 公共汽车自动或者手动语音报站系统运行效果画面

程序中每站的时间间隔设计成可以动态修改,就更能反映现实情况,读者可以根据本例代码自行完善。

## 8.2 TAnimate 组件动画程序设计

在 Windows 操作系统中,常常看到一些对话框运行时带有动画的效果,如删除、复制文件等,这为系统用户带来极大的直观性和动感色彩,是一种较好的开发方案。在 Delphi 中,也可以利用 TAnimate 组件创建与上述类似的动画效果。

### 8.2.1 TAnimate 组件的主要属性和常用方法

TAnimate 组件位于组件面板的 Win32 分类中,它主要用于播放含有视频流的 AVI 文件,这些视频流必须是非压缩的或使用 RLE8 压缩模式来压缩的文件,并且没有色调的变化,如果它们有声音,在播放的过程中也会被忽略。

TAnimate 组件播放的 AVI 文件可来源于以下两种途径:

- (1) 其 FileName 属性所指定的 AVI 文件。
- (2) Windows 系统所包含的内部图像序列,可通过该组件的 CommandAVI 属性来指定。

#### 1. 常用属性

- (1) Active 属性:用来设置是否播放动画。
- (2) AutoSize 属性:用来决定 TAnimate 组件的大小是否随着播放动画图像的大小而改变。
- (3) Center 属性:用来设置播放的动画图像是否在 TAnimate 组件的中央。
- (4) FileName 属性:用来指明要播放的 AVI 的文件名(包括文件所在的路径)。

- (5) CommonAVI 属性：用来设定播放 Windows 自带的哪种 AVI 动画。
- (6) StartFrame 属性：设置从动画的哪一帧开始播放。
- (7) StopFrame 属性：用来设置播放到动画的哪一帧结束。
- (8) FrameCount 属性：表示播放的 AVI 动画文件的总帧数,为只读属性。
- (9) Repetitions 属性：设置重复播放 AVI 文件的次数。
- (10) Transparent 属性：用来设置播放背景是否为透明的。

## 2. 常用方法

### 1) Play 方法

功能：播放。

格式：procedure Play(FromFrame, ToFrame: Word; Count: integer);

一共有三个参数,FromFrame 表示开始帧; ToFrame 表示结束帧; Count 表示次数。

### 2) Seek 方法

功能：帧定位。

格式：procedure Seek(Frame: SmallInt);

Frame 表示要定位的帧号。

### 3) Reset 方法

功能：系统复位。

格式：procedure Reset;

### 4) Stop 方法

功能：停止工作。

格式：procedure Stop;

## 8.2.2 使用 TAnimate 组件实现动画播放的实例程序

**例 8.3** 利用 TAnimate 组件播放 Windows 的预设动画或者用户. AVI 文件。要求能够任意选择要播放的预设动画种类,并能够设置是否循环播放。

程序的设计步骤如下：

### (1) 新建一个 VCL Form 应用程序。

在 Delphi 2007 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 属性值为“利用 TAnimate 组件播放 Windows 的预设动画或者用户. AVI 文件示例”。

### (2) 向窗体添加组件并设置相关的属性。

① 向窗体添加一个 TPanel 组件 Panell,为图像源控制面板,一方面便于窗体布局,以加强界面的美观;另一方面使组件的功能划分更明确。设置 Panell 的 Align 属性为 alTop,将 Caption 属性置为空。调整 Panell 大小至合适的范围。

② 向窗体添加 TAnimate 组件,将其 Align 属性设为 alClient,Color 属性通过对话框设置成浅黄色。再加入一个 TOpenDialog 对话框,用于选择用户. AVI 文件,将对话框的 Filter 属性设置为. avi 文件,如图 8-6 所示,目的是过滤文件类型。

③ 在图像源面板 Panell 组件上添加两个单选框组件 TRadioGroup,设置它们的

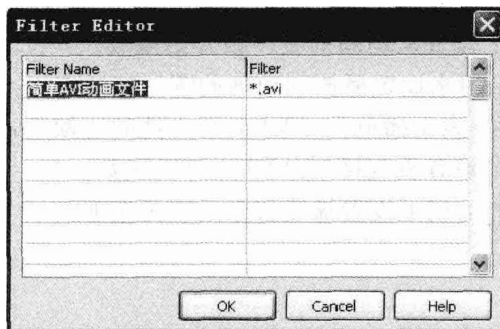


图 8-6 设置打开文件对话框的文件过滤属性

Caption 属性为“播放方式选择”和“Windows 系统动画选择”。

对于 TRadioGroup 组件,它的一个重要属性就是 Items,在对象查看器中单击该属性项右边的按钮,可打开字符串列表编辑器,对 Items 属性中包含的各个列表项进行编辑。RadioGroup1 单选框包含播放方式选择。RadioGroup2 单选框包含的系统动画各选项与 TAnimate 组件的 CommonAVI 属性的可能取值相对应。

④ 在控制面板 Panel1 组件上添加三个 TLabel 组件,三个 TEdit 组件和两个 TBitBtn 按钮组件,分别用于播放参数输入显示控制。两个位图按钮 BitBtn1 和 BitBtn2,一个用于控制动画的播放,一个用于控制动画的停止。

最后,调整所有组件位置与大小等辅助属性。

(3) 向单元添加组件事件代码。

```
//播放位图单击按钮
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    //按照设定参数,播放动画
    Animate1.Play(SpinEdit1.Value, SpinEdit2.Value, SpinEdit3.Value);
end;
//停止位图按钮鼠标单击
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
    Animate1.Stop;                                     //动画停止
end;
//用户方式选择
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
    if RadioGroup1.ItemIndex = 0 then                  //选择用户.AVI 文件
    begin
        Animate1.Stop;                                //将动画先停止
        if OpenFileDialog1.Execute then
            Animate1.FileName := OpenFileDialog1.FileName //选择用户文件
        else
            RadioGroup1.ItemIndex := 1;                //如果没选择,仍然指向系统预设动画
    end
    else
    begin
```

```

        Animatel.Stop;                                //停止动画
        Animatel.FileName := '';                       //删除用户文件名
    end;
end;
//选择系统预设动画种类
procedure TForm1. RadioGroup2Click(Sender: TObject);
begin
    //设置系统动画
    Animatel.CommonAVI := TCommonAVI(RadioGroup2. ItemIndex);
end;

```

(4) 保存并运行。

选择 File→Save 命令,保存单元文件与项目文件。然后运行,实例效果如图 8-7 所示。

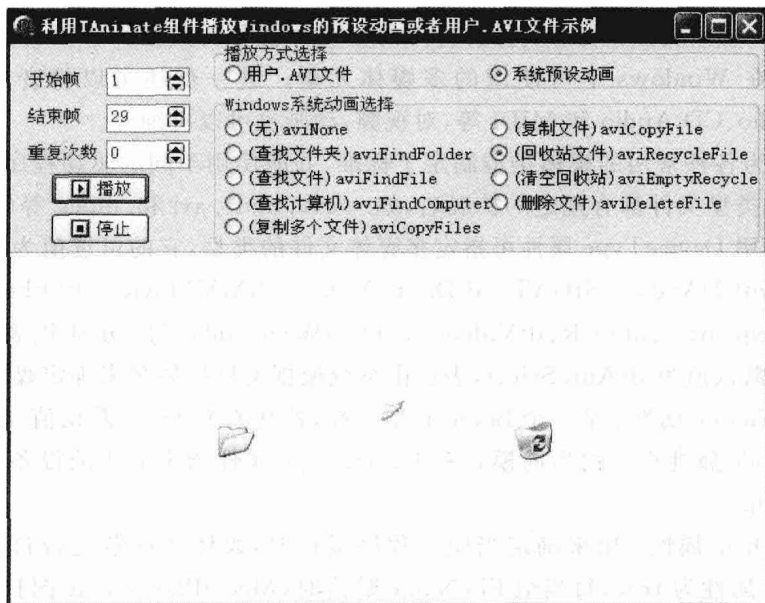


图 8-7 使用 TAnimate 控件实现动画播放运行界面

本程序如果选择用户. AVI 文件,将出现一个打开文件的对话框,用户选择一个符合要求正确格式的视频文件后,单击“播放”按钮,即可看到用户选择文件的动画。

## 8.3 多媒体播放程序设计

Delphi 2007 的 TMediaPlayer 组件对于开发多媒体音频、视频软件非常方便,该组件不仅可以播放 AVI 影片、Fic 和 Fli 动画文件,还可以通过 MCI(Media Control Interface,媒体控制接口)播放很多媒体文件,如 WAV、MIDI、MP3、CD 音乐文件和 WMV 视频文件。

### 8.3.1 MediaPlayer 的属性和事件

如图 8-8 所示,TMediaPlayer 位于组件面板的 System 页上。放到窗体中后,它的外观显示为一排按钮,如图 8-9 所示。这些按钮的名称从左到右分别为 Play(播放)、Pause(暂



停)、Stop(停止)、Next(至下一轨)、Prev(至前一轨)、Step(后移)、Back(前移)、Record(录制)和 Eject(退出)。

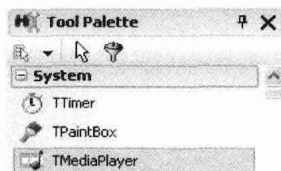


图 8-8 TMediaPlayer 组件

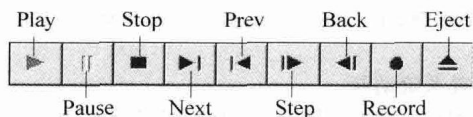


图 8-9 TMediaPlayer 组件各按钮功能解释

下面介绍该组件的主要属性、方法和事件。

### 1. 常用属性

(1) DeviceType 属性: 用来设置要打开的多媒体设备的类型。音频、视频、动画、图像和文本是 5 种在 Windows 下可使用的多媒体元素。进一步还可以将音频类型划分为 Waveform Audio、CD Audio 和 MIDI 等,对视频、动画也可以再细分。

Windows 为各种常用的媒体类型制定了标准,并能使用不同的驱动程序进行处理。通常,每一种媒体类型文件都有规定的扩展名,如 .wav、.mid、.avi 和 .mpeg 等(但这种规定不是强制的)。使用 DeviceType 属性可指定多媒体文件的类型,它的可选值为 dtAutoSelect、dtAVIVideo、dtCDAudio、dtDAT、dtDigitalVideo、dtMMMovie、dtOther、dtOverlay、dtScanner、dtSequencer、dtVCR、dtVideodisc 和 dtWaveAudio 等,分别代表不同的类型。DeviceType 的默认值为 dtAutoSelect,表示由系统根据文件扩展名来确定媒体的类型。

(2) AutoEnable 属性: 是一个 Boolean 型属性,若其值为 True(默认值)时,则媒体播放器可以根据 Mode 属性指定的当前模式和 DeviceType 属性指定的当前设备类型来确定哪些按钮可以使用。

(3) AutoOpen 属性: 用来确定当应用程序运行时,媒体播放器是否自动打开媒体设备。AutoOpen 属性为 true,每当给 FileName 赋值时,MediaPlayer 将试图打开该文件;该属性为 false 时,必须在程序中调用 Open 方法打开文件。AutoOpen 的默认值为 true。

(4) Display 属性: 用来指定一个窗口作为多媒体文件的输出屏幕。有些媒体类型(如动画等)在播放时需要一个供显示输出的窗口或控件,可用 Display 属性来指定。可代替输出窗口的控件主要有 Panel 等。若 Display 属性为 NULL,媒体的驱动程序在运行时会自己创建一个窗口供显示。Display 的默认值为 NULL。

(5) FileName 属性: 用于为多媒体播放设备指定一个待播放的多媒体文件名。一般来说,应该是一个可由 Open 方法打开的多媒体文件。

(6) EnabledButtons 属性: 设置多媒体播放器组件中哪些按钮是有效的,哪些是无效的。EnabledButtons 属性是一个集合类型的属性,可控制 MediaPlayer 上哪些按钮当前处于有效状态。若设置 AutoEnable 属性为 true,那么 Delphi 会根据媒体文件类型按一定的规则控制 MediaPlayer 上各个按钮在不同的环境下为有效或无效。

(7) DisplayRect 属性: 当 Display 不等于 NULL 时,可用 DisplayRect 进一步指定显示输出的一个矩形范围。

(8) Length 属性: 用来表示播放媒体的总长度。Length 属性仅在运行时可用且为只

读,用以获得 MediaPlayer 中打开的文件的长度,长度在这里指文件播放所需时间。该属性可以采用几种不同的时间格式表示长度,具体用哪种格式由 TimeFormat 属性确定。

(9) Position 属性:用来表示媒体当前的播放位置。Position 属性在运行阶段可用,它指定了 MediaPlayer 中打开的媒体文件的当前位置。使用的时间格式由 TimeFormat 确定。

(10) VisibleButtons 属性:用来决定 TMediaPlayer 组件中的各个按钮是否可见,可设置每个按钮的可见状态。在设计阶段可在 Object Inspector 中单击该属性左侧的“+”设置这些按钮是否为可见,也可在运行中改变 VisibleButtons。通常,处理不同类型的媒体文件时,VisibleButtons 应有所不同,如 Record 按钮一般都不需要。

(11) Wait 属性:该属性为 Boolean 类型,用于决定是否当媒体控制方法执行结束后,才将控制权交给应用程序。若该属性为 true,则在 MediaPlayer 执行一个方法期间应用程序必须等待。

(12) Mode 属性:用来返回多媒体设备的当前状态。Mode 属性仅在运行时可用且为只读,用以获得媒体播放器当前设备的状态。表 8-2 列出了 Mode 属性可能的取值及说明。

表 8-2 Mode 属性的值及说明

取 值	说 明	取 值	说 明
mpNotReady	尚未就绪	mpSeeking	正在查找
mpStopped	已停止	mpPaused	暂停状态
mpPlaying	正在演播	mpOpen	多媒体文件已打开
mpRecording	正在录音(或录像)		

(13) AutoRewind 属性:该属性为 Boolean 类型,用来设置在多媒体文件播放完毕以后,是否自动返回到起点。

(14) Notify 属性:用来决定当一个多媒体控制方法(如 Open、Play 等)完成以后,是否响应下一个多媒体方法。Notify 属性设置为 true 时,则 MediaPlayer 每次在执行某个命令时会将完成的情况产生一个 OnNotify 事件,通知应用程序以便进行相应的处理,并将所产生的通知信息放在 NotifyValue 属性中。

(15) NotifyValue 属性:NotifyValue 属性在运行阶段可用,且为只读。当 MediaPlayer 产生 OnNotify 事件时,相应的 Notify 消息就放在 NotifyValue 中。表 8-3 列出了 NotifyValue 属性的可能值及说明。

表 8-3 NotifyValue 的值及说明

取 值	说 明	取 值	说 明
NvSuccessful	命令已成功地完成	NvAborted	用户中止了该命令
NvSuperseded	命令已被另一命令取代	NvFailure	命令执行失败

(16) Visible 属性:用于决定 TMediaPlayer 组件是否可见,默认值是 True,表示可见。

(17) TimeFormat 属性:用来设置描述时间的格式,该属性是一个运行属性。

(18) Tracks 属性:用来表示 CD 音乐总轨数。

(19) TrackLength 属性:用来表示音乐轨的长度。

(20) TrackPosition 属性：用来表示当前音乐轨的播放位置。

(21) Frames 属性：用来确定调用 Back 或 Step 方法时,前进或后退的帧数。

## 2. 常用方法

TMediaPlayer 组件的常用方法及其作用如下：

(1) Close 方法：关闭已打开的多媒体设备。

(2) Next 方法：跳到下一个轨道的开始,如果媒体不用轨道,将跳到媒体的结尾。

(3) Eject 方法：从打开的设备中退出装入的媒体。单击 MediaPlayer 的 Eject 按钮,即发生一次对 Eject 方法的调用。

(4) Rewind 方法：Rewind 方法执行时回到播放的起点。

(5) Open 方法：打开一个多媒体设备。设备的类型由 DeviceType 属性指定,或者在 AutoOpen 为 true 时,设备的类型由文件的扩展名确定。

(6) Save 方法：保存已经打开的媒体信息。

(7) Pause 方法：暂停打开的多媒体设备。多媒体设备正在工作时,单击 MediaPlayer 的 Pause 按钮,即发生一次对 Pause 方法的调用。

(8) Back 方法：Back 方法执行后将在当前打开的媒体中向后移动一定数量的帧,移动的帧数由 Frames 属性决定。

(9) Play 方法：播放多媒体设备中打开的文件。单击 MediaPlayer 的 Play 按钮,即发生一次对 Play 方法的调用。

(10) Previous 方法：Previous 方法执行后将跳到前一个轨道的开始,如果媒体不用轨道,将跳到媒体的开始。

(11) Resume 方法：恢复执行被 Pause 方法暂停的操作。多媒体设备处于暂停状态时,单击 MediaPlayer 的 Pause 按钮,即发生一次对 Resume 方法的调用。

(12) StartRecording 方法：StartRecording 方法执行后将开始录制媒体信息。

(13) Step 方法：Step 方法执行后将在当前打开的媒体中向前移动一定数量的帧,移动的帧数由 Frames 属性决定。

(14) Stop 方法：停止 MediaPlayer 正在执行的操作。单击 MediaPlayer 的 Stop 按钮,即发生一次对 Stop 方法的调用。

## 3. 常用事件

TMediaPlayer 组件能够响应 OnClick、OnPostClick、OnEnter、OnExit 和 OnNotify 等多种事件,重点说一下 OnClick、OnNotify 事件的使用。

(1) OnClick 事件：当 TMediaPlayer 组件对象被单击时将触发该事件。它的事件声明如下：

```
procedure TForm1. MediaPlayer1Click (Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean);
```

其中的参数 Button 传入的是 MediaPlayer 上被单击的那个按钮。Button 属于 TMPBtnType 类型,可能的取值有 btPlay、btPause、btStop、btNext、btPrev、btStep、btBack、btRecord 和 btEject。

(2) OnNotify 事件：当属性 Notify 为 true 时,MediaPlayer 每次执行某个方法(如前面

所讲的 Open、Save、Back、Close、Eject 和 Pause 等), 不管执行成功或失败, 总要发出一个 Notify 通知应用程序, 该 Notify 会触发 OnNotify 事件, 用户可在该事件的处理代码中控制 MediaPlayer 的下一步操作。

### 8.3.2 TTrackBar

设计多媒体播放程序肯定少不了播放进度的显示与控制, Delphi 里提供的 TTrackBar (轨迹进度条) 组件就可以实现这样的功能。TTrackBar 组件位于组件面板 Win32 组件页面上, 如图 8-10 所示。

TTrackBar 组件与 TScrollBar 组件相似, 也有一个滚动条, 且两侧可以显示刻度, 但两端无按钮。程序运行时, 可用鼠标拖动滚动条中的滑块位置来改变 Position 的值。

下面简单地介绍一下 TTrackBar 组件的常用属性和事件。

#### 1. TTrackBar 组件的常用属性

(1) Max 属性: 用来设置 TTrackBar 组件 Position 属性的最大值。

(2) Min 属性: 用来设置 TTrackBar 组件 Position 属性的最小值。

(3) Position 属性: 代表滑块所在位置的值。

(4) SelStart 属性: 用来设置滑块拖动范围的起始点。

(5) SelEnd 属性: 用来设置滑块拖动范围的终止点。

(6) Frequency 属性: 用来设置刻度标记的频率, 此频率与取值范围有关。

(7) LineSize 属性: 用来设置按箭头键时, TTrackBar 组件的 Position 属性增加或减少的值。

(8) PageSize 属性: 用来设置按 PageDown、PageUp 键时或在 TTrackBar 组件上单击时, TTrackBar 组件的 Position 属性增加或减少的值。

(9) Orientation 属性: 用来定义 TTrackBar 组件是水平排列的还是垂直排列的。

(10) TickMarks 属性: 用于指出刻度出现的位置, 取值为 TmBottom-Right(右边或下边)、TmTopLeft(左边或上边)和 TmBoth(两侧)。默认值为 TmBottomRight。

#### 2. TTrackBar 组件的常用事件

TTrackBar 组件的常用事件是 OnChange, 该事件在 Position 属性值发生改变时触发。

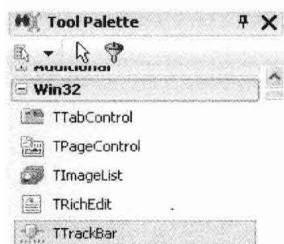


图 8-10 TTrackBar 组件

### 8.3.3 多媒体播放程序设计范例

**例 8.4** 使用 TMediaPlayer 组件制作一个带进度条(使用 TTrackBar 组件)的可以播放多种媒体格式的多媒体播放器。

程序的设计步骤如下:

(1) 创建一个新的项目。

在 Delphi 2007 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序, 系统会自动创建一个空白的窗体。调整窗体的大小, 并设置窗体的 Caption 值为“多媒体播放器”。

(2) 向窗体添加组件,并设置组件的相关属性。

① 在窗体上添加主菜单组件 MainMenu1,通过菜单编辑器编辑各个菜单项。该菜单包含“文件”和“退出”两个菜单项,在“文件”菜单项中又包含“打开”、“播放”和“停止”三个子菜单项。

② 在窗体上放置一个进度轨迹条 TrackBar1(TrackBar 在 Win32 组件页面上),TrackBar1 组件用于显示文件播放进度。

③ 在窗体上放置一个打开文件对话框 OpenFileDialog,用于打开媒体文件。该对话框的 Filter 属性的设置如图 8-11 所示。

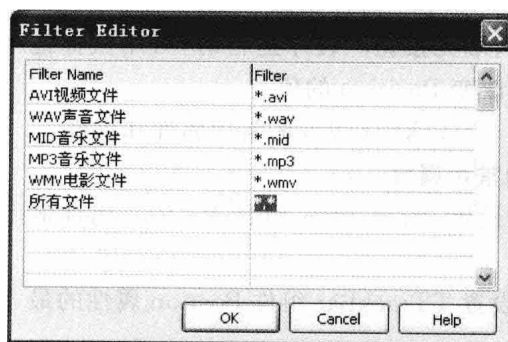


图 8-11 打开文件对话框设置媒体文件类型过滤

④ 在窗体中放置一个面板组件 Panel1,将 Panel1 的 Caption 属性置为空。

⑤ 在 Panel1 组件上放置一个媒体播放组件 MediaPlayer1,用于控制媒体的播放。

⑥ 在窗体上添加定时器组件 Timer1,以便定时记录媒体播放器的工作状态。将 Timer1 的 Interval 属性值设为 200。

⑦ 在窗体中放置两个标签组件 Label1 与 Label2,用于显示媒体时间信息。

调整所有组件位置与大小等属性合适后,窗体设计界面如图 8-12 所示。

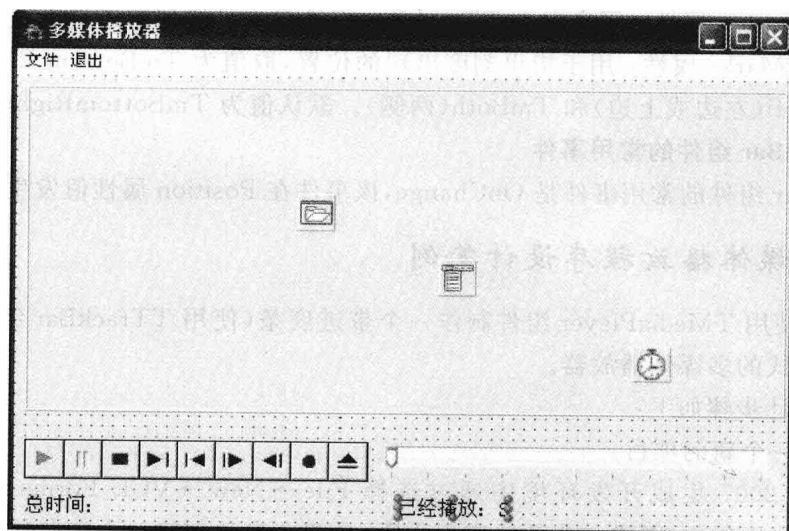


图 8-12 多媒体播放器界面设计

### (3) 代码编辑。

```
//窗体创建事件,为了初始化
procedure TForm1.FormCreate(Sender: TObject);
begin
    MediaPlayer1.DeviceType := dtAutoSelect;           //设置设备驱动类型为自动类型
end;
//媒体控制面板单击事件
procedure TForm1.MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
    var DoDefault: Boolean);
var total, HH, MM, SS: integer;
begin
    //根据选择的按钮来处理
    case Button of
        btPlay:
            begin
                TrackBar1.Max := MediaPlayer1.Length;           //设置轨迹条的最大长度
                total := MediaPlayer1.Length div 1000;           //总时间秒数
                HH := total div 3600;                             //提取小时
                MM := (total mod 3600) div 60;                   //提取分钟
                SS := total mod 60;                               //提取秒
                label2.Caption := '总时间: ' + FormatFloat('00', HH) + '小时 '
                    + FormatFloat('00', MM) + '分钟 ' + FormatFloat('00', SS) + '秒 ';
            end;
        end;
end;
//退出菜单单击事件
procedure TForm1.N_ExitClick(Sender: TObject);
begin
    Close;           //关闭窗口体
end;
//打开菜单单击事件
procedure TForm1.N_OpenClick(Sender: TObject);
begin
    if OpenFileDialog1.Execute then           //打开对话框执行
    begin
        MediaPlayer1.Close;           //先关闭旧的媒体文件
        MediaPlayer1.FileName := OpenFileDialog1.FileName;           //设置组件的媒体文件名
        MediaPlayer1.Open;           //打开媒体文件
        MediaPlayer1.Display := Panel1;           //设置播放使用对象
        //设置播放矩形实际区域大小
        MediaPlayer1.DisplayRect := Rect(1, 1, Panel1.Width - 2, Panel1.Height - 2);
        MediaPlayer1.TimeFormat := tfHMS;           //设置播放时间格式
        caption := '多媒体播放器' + OpenFileDialog1.FileName;           //给窗体标签显示当前播放文件名
    end;
end;
//播放菜单单击事件
procedure TForm1.N_PlayClick(Sender: TObject);
begin
    MediaPlayer1.Play;           //执行媒体组件对象 Play 方法
end;
```

```

//停止菜单单击事件
procedure TForm1.N_StopClick(Sender: TObject);
begin
    MediaPlayer1.Stop;                                     //执行媒体组件对象停止方法
end;
//定时器定时时间到事件
procedure TForm1.Timer1Timer(Sender: TObject);
var Curr, HH, MM, SS: integer;
begin
    TrackBar1.Position := MediaPlayer1.Position;           //设置进度轨迹条的当前位置
    if MediaPlayer1.Mode = mpPlaying then
    begin
        Label1.Caption := inttostr(MediaPlayer1.Position);
        Curr := MediaPlayer1.Position div 1000;           //当前已经播放秒数
        HH := Curr div 3600;
        MM := (Curr mod 3600) div 60;
        SS := Curr mod 60;
        label1.Caption := '已经播放: ' + FormatFloat('00', HH) + '小时 '
            + FormatFloat('00', MM) + '分钟 ' + FormatFloat('00', SS) + '秒 ';
    end;
end;
end;

```

#### (4) 保存并运行。

选择 File→Save 命令,保存单元文件与项目文件。程序开始运行时,必须通过“打开”菜单先打开媒体文件,然后再通过媒体播放器控制其播放,实例效果如图 8-13 所示。



图 8-13 多媒体播放器播放电影运行界面

程序代码中,在 MediaPlayer1 的 OnClick 事件中利用参数 Button 区分被单击的那个键,TrackBar1.Max 属性用于设置轨迹条的最大值,这是与终点位置对应的值,起点默认对应最小值为 0。OnTimer 事件执行语句 TrackBar1.Position := MediaPlayer1.Position;

使轨迹条进展与播放进度相一致。MediaPlayer1.TimeFormat := tfHMS;语句用于指定时间格式,.avi 文件一般采用 tfHMS(时分秒)格式。

## 8.4 Flash 动画播放程序设计

Flash 是由 Macromedia 公司推出的制作网络动画的二维矢量动画标准。Flash 动画在多媒体软件、商业产品演示、媒体教学和游戏等领域被广泛使用,特别是在网络动画中,Flash 动画已成为网络动画标准。无论是专业动画人员,还是业余爱好者,Flash 都成为必备的高级动画显示效果元素。

本节主要讲解在 Delphi 的用户程序中如何播放 Flash 动画。

一般的 PC 都已经有了 Flash 播放软件(IE 浏览器显示 Flash)。IE 之所以能显示 Flash 动画是因为其使用了由 Macromedia 公司提供的 Flash10c.ocx 组件,可以在操作系统安装盘的 Windows\System32\Macromed\Flash 文件夹中找到该文件。

### 8.4.1 Delphi 2007 下 Flash ActiveX 组件安装

要在 Delphi 2007 中编制能够播放 flash 动画的程序,就必须依赖于 Flash ActiveX 组件,所以要先安装它到 Delphi 2007 的组件面板中。

其安装不能像 Delphi 早期版本 D6、D7 中那样直接安装,在 Delphi 2007 下的操作步骤如下:

(1) 运行 Delphi 2007 后,选择 Component→Import Component 命令,出现一个让用户选择引入什么类型组件的对话框,选择 Import ActiveX Control,然后单击 Next 按钮进入下一步操作,进入选择 ActiveX 注册页面,如图 8-14 所示。

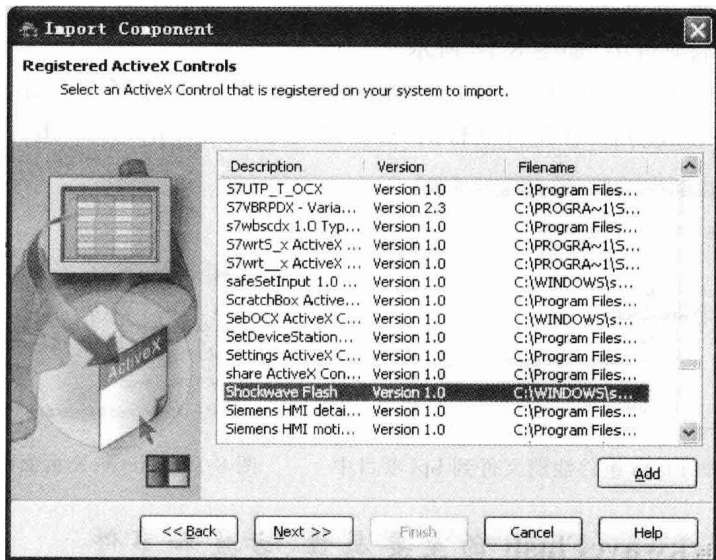


图 8-14 安装 Flash ActiveX 的主要对话框

(2) 在选择 ActiveX 注册页面对话框中找到 Shockwave Flash 一项,然后单击 Next 按钮进入下一步操作,出现指定生成一个引入该组件的 pas 单元文件,如图 8-15 所示。单击



Next 按钮进入下一步操作,在对话框中选择 Create Unit,最后单击 Finish 按钮。这样就得到了两个文件 ShockwaveFlashObjects\_TLB.dcr 和 ShockwaveFlashObjects\_TLB.pas。

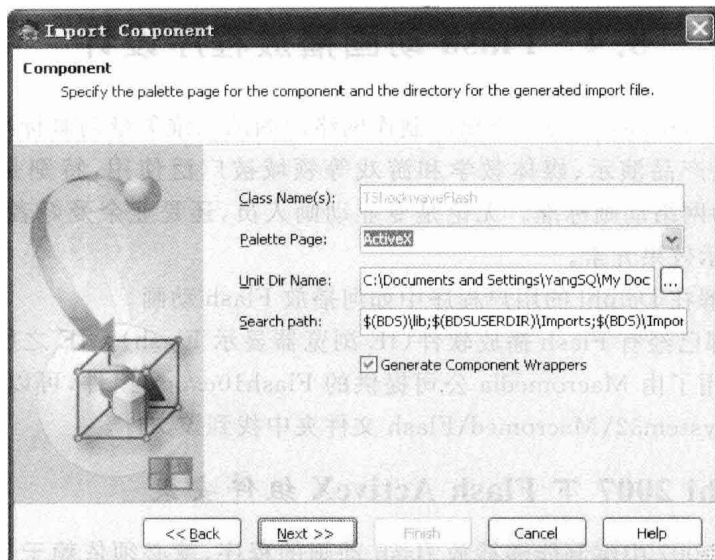


图 8-15 指定生成一个引入该组件的 pas 单元文件

(3) 选择 File→New→Other 命令,出现一个对话框,选择 Package 后单击“确定”按钮,这样就建立了一个组件安装包项目文件。把该项目文件(可以换名)存到用户目录中。然后在 Project Manager 中右击包的项目名称,在弹出的快捷菜单中选择 Add 命令,根据对话框提示把前面创建好的两个文件 ShockwaveFlashObjects\_TLB.pas 和 ShockwaveFlashObjects\_TLB.dcr 添加到本项目中,如图 8-16 所示。

(4) 右击组件安装包项目文件,在弹出的快捷菜单中选择 Install 命令,安装 Flash ActiveX 组件。安装成功后,在组件面板的 ActiveX 中出现想要的 Flash 组件,如图 8-17 所示。

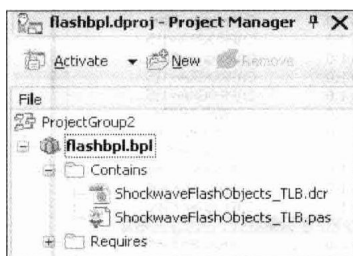


图 8-16 添加 Flash 组件注册文件到 bpl 项目中

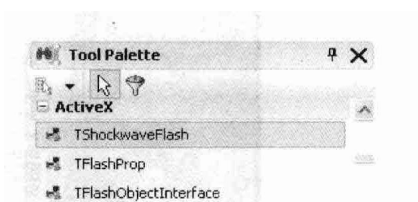


图 8-17 ActiveX 页面的 Flash 组件

## 8.4.2 TShockwaveFlash 的主要属性、方法和事件

### 1. TShockwaveFlash 的主要属性

(1) ReadyState: 主要表明读一个 Flash 文件时的状态,状态的取值包括 0=Loading、1=Uninitialized、2=Loaded、3=Interactive 和 4=Complete。

重点解释一下: 如果为 4, 意思为 Complete, 表明文件已经准备好, 用户可以播放; 如果

为 1,表明组件还没有初始化运行。

(2) TotalFrames: 表示要播放 flash 文件的总帧数,只有当 ReadyState=4 时才能访问该属性。

(3) FrameNum: 表明当前播放的帧号,只有当 Flash 动画文件处于播放状态时有效。

(4) Playing: 该属性为布尔型,可以设置它为 True 或者 False,控制播放或暂停一个已经装入的 flash 动画文件。

(5) Quality: 该属性是一个很重要的参数,指定动画播放显示效果,即控制当前画面渲染的质量,其取值主要包括 0=Low(低级)、1=High(高级)、2=AutoLow(自动低)和 3=AutoHigh(自动高)。

(6) ScaleMode: 定义播放动画画面的缩放模式,0=ShowAll(显示全部)、1=NoBorder(无边界)、2=ExactFit(适合)。

(7) Align: 定义组件在窗体上排列的模式,跟普通组件的 Align 属性含义一样,取值主要有 alLeft、alRight、alTop 和 alBottom 等。

(8) BackgroundColor: 设置组件的背景色,如果设置为 -1,表示使用默认颜色。

(9) Loop: 该属性为布尔型,表示动画播放是否需要循环。

(10) Movie: 最重要的一个属性,指定播放的用户 Flash 文件路径,可以为一个网路上 URL 路径。一般来说,需要用户通过打开对话框为其赋值。设置动画文件名还可以使用 LoadMovie 方法。

## 2. TShockwaveFlash 的主要方法

(1) Play 方法: 控制 Flash 文件动画的播放开始。

(2) Stop 方法: 控制 Flash 文件动画的播放暂停。

(3) StopPlay 方法: 控制 Flash 文件动画的播放终止。

(4) Back 方法: 控制 Flash 文件动画播放时回退到前一帧动画。

(5) Forward 方法: 控制 Flash 文件动画播放时前进到后一帧动画。

(6) Rewind 方法: 控制 Flash 文件动画重置,即回到第一帧动画播放。

(7) SetZoomRect 方法: 该方法的原型为 procedure TShockwaveFlash. SetZoomRect(left: Integer; top: Integer; right: Integer; bottom: Integer);,主要功能是控制设置缩放的区域。

(8) Zoom 方法: 该方法的原型为 procedure TShockwaveFlash. Zoom(factor: SYSINT);,主要功能是缩放(按百分比),Factor 为缩放的比例。

(9) Pan 方法: 该方法的原型为 procedure TShockwaveFlash. Pan(x: Integer; y: Integer; mode: SYSINT);,主要功能是控制缩放播放面板,其中参数 mode 为模式,取值 0 为按像素、取值 1 为按窗口百分比。

(10) LoadMovie 方法: 该方法的原型为 procedure TShockwaveFlash. LoadMovie(layer: SYSINT; const url: WideString);,主要功能是装入一部动画影片,等同于给 Movie 属性赋值。

## 3. TShockwaveFlash 的主要事件

(1) OnProgress(int percent)

说明: 发生在 Flash 影片下载时。percent 是影片已下载的百分比,取值为 0~100。读

取一个 Flash 时触发。

(2) OnReadyStateChange(int state)

说明：发生在控件的准备状态改变时。states 的值可以为 0=Loading、1=Uninitialized、2=Loaded、3=Interactive 和 4=Complete。

(3) OnFSCommand(const command, args: WideString)

说明：可用来读取 Flash 按钮中的参数。在 Flash 中为影片添加的 FSCommand 动作可以从影片中传递信息给 Flash 播放器程序，Flash 播放器或包含播放器控件的网页或程序播放这个影片时就根据得到的这些信息执行相应的动作，从而实现动画影片内部与外部程序应用的交互操作。

### 8.4.3 Delphi 程序与 Flash 组件的信息交换

Delphi 2007 与 Flash 交互通信的优点在于：开发 Delphi 应用程序时充分发挥 Flash 表现力强的特点，只要运用得当，Flash 开发的色彩斑斓的界面就会出现在 Delphi 应用程序当中，且无须 Delphi 程序员去堆放大量浪费资源的组件。

Delphi 与 Flash 交互具体存在两个问题：

(1) 如何能让 Delphi 应用程序接收到 Flash 发来的消息并处理。

(2) 怎样能从 Delphi 应用程序向 Flash 传递命令，控制 .swf 文件的运行。

#### 1. Flash 向 Delphi 发送消息：让 Delphi 应用程序接收到 Flash 发来的消息并处理

Flash 向 Delphi 发送消息其实很简单，只需要运用 Flash 中自带的 fscommand 命令即可，此命令在作者能见到的各版 Flash 中都存在，尽管后期版本从安全角度考虑不提倡使用它，但在这里此命令仍可以大加利用。

例如，假设 testflash.swf 文件在制作中有如下 Flash 中的代码：

```
on (click){
    fscommand("test_118","flash 向 delphi 程序发消息了");
}
```

//注意：这里不是 Delphi 语句，它是遵循 Flash 语法的代码

以上代表 Flash 中某一元件的单击事件会触发 fscommand 命令，其中参数 test\_118 是命令，后面的就是命令的值。

以下为在 Delphi 中使用的接收代码：

```
procedure TForm1.ShockwaveFlash1FSCommand(ASender: TObject; const command,
    args: WideString);
begin
    If Command = 'test_118' Then
        Begin
            ShowMessage(args);
        End;
end;
```

#### 2. Delphi 应用程序向 Flash 传递命令，控制 .swf 文件的运行

Delphi 方面，执行代码如下：

```
ShockwaveFlash1.SetVariable('fcom_118','这是 delphi 向 flash 传的命令');
```

从这一句代码可见,被执行的 Flash 中必须存在变量名为 fcom\_118 的元件,应为文本元件(必须),后面是传递的内容。执行后会在 Flash 中名为 fcom\_118 的文本元件中显示传入的“这是 delphi 向 flash 传的命令”这段内容。

发出去的消息,还可以通过 Getvariable 方法来读回进行核对。

另外注意,以上只是代码的片段,在执行以上代码之前千万别忘了在 Delphi 应用程序中播放 Flash,播放代码为:

```
ShockwaveFlash1.LoadMovie(0,'testflash.swf');           //通过方法装入 Flash 动画文件
ShockwaveFlash1.Play;                                     //调用方法播放 Flash 动画
```

## 8.4.4 简单的 Flash 播放程序设计范例

**例 8.5** 设计一个简单的 Flash 动画播放器。

程序的设计步骤如下:

(1) 创建一个新的项目。

在 Delphi 2007 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“一个简单的 Flash 动画播放器示例”。

(2) 向窗体添加组件,并设置组件的相关属性。

① 在窗体上添加主菜单组件 MainMenu1,通过菜单编辑器编辑各个菜单项。该菜单包含“文件”和“退出”两个菜单项,在“文件”菜单项中又包含“打开”、“播放”、“停止”、“前进”和“后退”5 个子菜单项。

② 在窗体上放置一个进度轨迹条 TrackBar1 (TrackBar 在 Win32 组件页面上), TrackBar1 组件用于显示 Flash 动画文件播放进度。设置 Align 属性为 alBottom。

③ 在窗体上放置一个打开文件对话框组件 OpenFileDialog1,用于打开媒体文件。该对话框的 Filter 属性设置为 \*.SWF 文件,同时设置选项 Options 属性要求打开的文件必须存在。

④ 在窗体中放置一个面板组件 Panel1,将 Panel1 的 Caption 属性置为空,设置其 Align 属性为 alClient。

⑤ 从组件面板的 ActiveX 页拖拉一个 ShockwaveFlash1 组件放置在 Panel1 组件上, MediaPlayer1 组件用于控制 Flash 动画文件的播放,设置其 Align 属性为 alClient。

⑥ 在窗体上添加定时器组件 Timer1,以便定时设置播放器的工作进度。将 Timer1 的 Interval 属性设为 1000,并将其 Enabled 属性设置成 False。

调整所有组件位置与大小等属性合适后,窗体设计界面如图 8-18 所示。

(3) 代码编辑。

在窗体单元文件中编写如下代码:

```
//后退菜单点击事件
procedure TForm1.N_BackClick(Sender: TObject);
begin
    ShockwaveFlash1.Back;           //控制向后一帧
    ShockwaveFlash1.Play;           //继续播放
end;
```



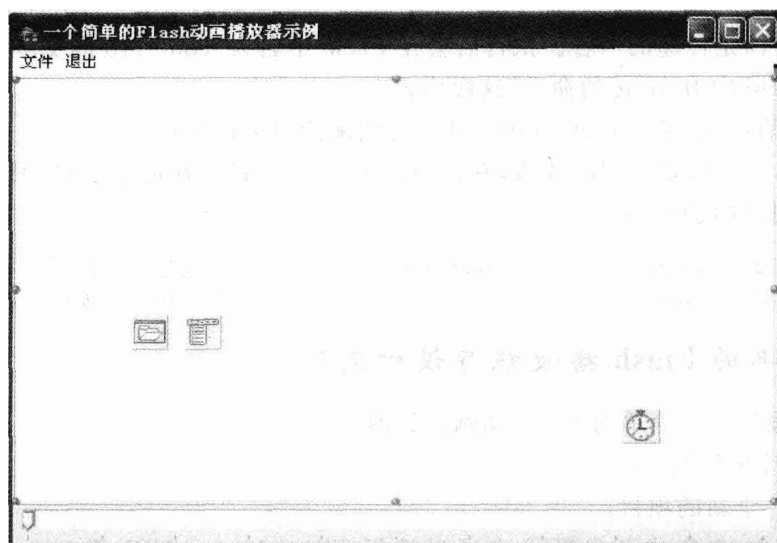


图 8-18 简单的 Flash 播放器界面设计

```
//退出菜单单击事件
procedure TForm1.N_ExitClick(Sender: TObject);
begin
    Close; //关闭窗口
end;
//前进菜单单击事件
procedure TForm1.N_NextClick(Sender: TObject);
begin
    ShockwaveFlash1.Forward; //控制向前一帧
    ShockwaveFlash1.Play; //继续播放
end;
//打开菜单单击事件
procedure TForm1.N_OpenClick(Sender: TObject);
begin
    if OpenFileDialog1.Execute then //打开对话框执行
    begin
        Caption := '简单的 Flash 动画播放器:' + OpenFileDialog1.filename; //设置显示标签
        ShockWaveFlash1.StopPlay; //先停止当前的播放
        ShockwaveFlash1.Movie := OpenFileDialog1.filename; //设置要播放的文件
        Timer1.Enabled := True; //定时器启动
        Trackbar1.max := ShockwaveFlash1.TotalFrames; //设置轨迹条的最大帧数
        ShockWaveFlash1.Play; //启动播放
    end;
end;
//播放菜单单击事件
procedure TForm1.N_PlayClick(Sender: TObject);
begin
    ShockwaveFlash1.Play; //执行 Flash 播放组件对象 Play 方法
end;
//停止菜单单击事件
procedure TForm1.N_StopClick(Sender: TObject);
```

```

begin
    ShockwaveFlash1.StopPlay;           //执行 Flash 播放组件对象停止方法
end;
//定时器定时时间到事件
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Trackbar1.position := ShockwaveFlash1.FrameNum;    //设置进度轨迹条的当前位置
end;

```

(4) 保存并运行。

选择 File→Save 命令,保存单元文件与项目文件。程序开始运行时,必须通过“打开”菜单先打开一个已经存在合法的 Flash 文件,然后可以看到其立即播放的画面,实例效果如图 8-19 所示。

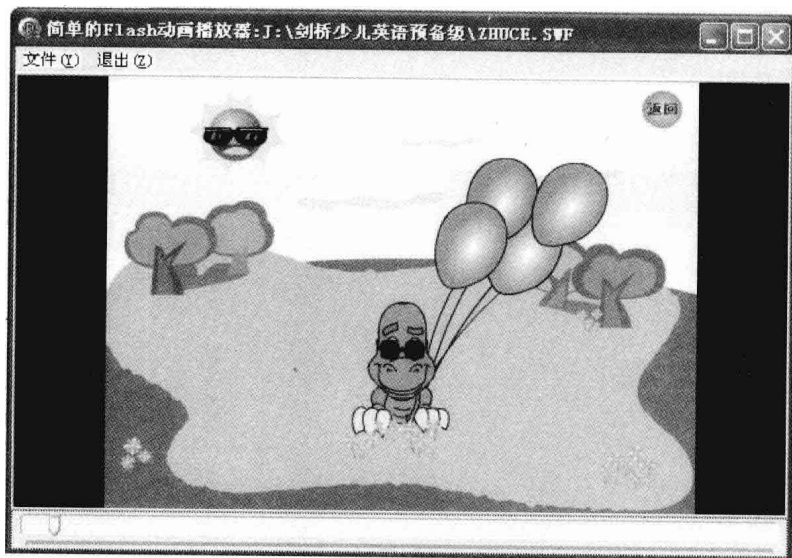


图 8-19 Flash 播放器播放动画文件的运行界面

## 本章小结

多媒体技术与计算机技术紧密结合,制作出了大量用户喜爱的生活、学习、娱乐软件, Delphi 为开发人员提供了功能全面的多媒体开发组件。

本章主要介绍了 Delphi 环境下,多媒体与动画程序的开发方法,其中包含声音的控制、简单动画的制作以及功能强大的多媒体播放器的设计与制作。本章重点是动画组件 TAnimate、多媒体 TMediaPlayer 组件的属性、方法、事件的掌握。难点是安装使用第三方组件设计 Flash 动画程序。

通过本章学习,读者应重点掌握学会使用 TAnimate 和 TMediaPlayer 组件开发多媒体应用程序的方法。

## 思考与练习

1. 简述 Delphi 2007 下 Flash ActiveX 组件安装的步骤。
2. Delphi 2007 的 TMediaPlayer 组件主要有哪些功能？
3. 利用 API 函数编写一个窗体背景音乐(.wav 文件类型)自动循环播放程序。
4. 根据本章的例子,利用 TAnimate 组件编写一个简单的动画播放程序。
5. 根据本章的例子,利用 TMediaPlayer 组件制作一个简易的 mp3 播放器,具有播放、暂停、停止、播放前一曲和播放后一曲的功能。

本章首先介绍数据库的基本概念,然后介绍 ADO 技术及基于 ADO 技术的数据库应用程序基本结构。理解数据库应用程序结构是编写数据库应用程序的关键,而重点在于如何将应用程序连接到数据库。

本章主要包括以下内容:

- 数据库基础知识;
- Delphi 数据集组件;
- 数据源组件和数据控制组件;
- ADO 组件及应用。

## 9.1 数据库基础知识

数据库技术是信息社会的重要基础技术之一,对于大量的数据使用数据库进行存储管理比通过文件进行存储管理具有更高的效率。

### 9.1.1 数据库的基本概念

#### 1. 数据

数据(Data)是数据库中存储的基本对象。所谓数据,就是能被计算机识别与处理的符号。数据的种类很多,如数字、文字、表格、图形、图像和声音等都属于数据。

#### 2. 数据库

数据库(Database,DB)就是以一定的组织方式存储在计算机存储介质中的互相关联的数据的集合。它能以最佳方式、最少重复、最大独立性为多种应用提供数据共享服务。一个数据库常包含许多数据表、索引信息以及其他相关信息。

#### 3. 数据库管理系统

数据库管理系统(Database Management System,DBMS)是数据库系统的核心,是负责数据库的建立、使用和维护的软件,主要功能包括数据库的定义、数据操纵、数据库运行管理、数据库的建立和维护。数据库管理系统是支持人们建立、使用和修改数据库的软件系统。它是位于用户和操作系统之间层面的数据管理软件。它为用户或应用程序提供访问数据库的方法,包括数据库的建立、查询、更新及各种数据控制。

DBMS 可以分为层次型、网状型、关系型和面向对象型等几种类型。

数据库在建立、使用和维护时由数据库管理系统统一管理,统一控制。数据库管理系统使用户方便地定义数据和操作数据,并能够保证数据的安全性、完整性、并发性及发生故障



后的系统恢复。

#### 4. 数据库系统

数据库系统(DataBase System, DBS)是指在计算机系统中引入数据库后的系统构成,一般由数据库、数据库管理系统及其应用开发工具、应用系统构成。

数据库系统实现了有组织地、动态地存储大量关联数据,方便多用户访问。通俗地讲,数据库系统可把日常的一些表格、卡片等的有组织地集合在一起输入到计算机中,然后通过计算机处理,再按一定要求输出结果。可以用图示来表示数据库系统中各部分之间的关系,如图 9-1 所示。

数据库系统从最终用户角度看,分为单用户结构、主从式结构、分布式结构和客户端/服务器结构。

##### 1) 单用户结构

整个数据库系统(包括应用程序、DBMS 和数据)都装在一台计算机上,由一个用户独占,不同机器之间不能共享数据。

##### 2) 主从式结构

此结构是指一个主机带多个终端的多用户结构,数据库系统(包括应用程序、DBMS 和数据)都集中存放在主机上,所有处理任务都由主机完成,各个终端用户并发地存取数据库,共享数据资源。

##### 3) 分布式结构

分布式结构的数据库系统是地理上(或物理上)分散而逻辑上集中的数据库系统。管理这种系统的软件称为分布式数据库管理系统。分布式数据库系统通常由计算机网络连接起来,被连接的逻辑单位(包括计算机、外部设备等)称为节点。地理上分散是指各个节点在不同的地方,逻辑上统一是指网络连接的各节点共同组成单一的数据库。

##### 4) 客户端/服务器结构

随着工作站功能的增加和广泛使用,人们开始把 DBMS 功能和应用分开,网络中专门用于执行 DBMS 功能的计算机称为数据库服务器(简称服务器,Server),其他安装 DBMS 的外围应用开发工具、且支持用户应用的计算机称为客户端(Client),这就是客户端/服务器(Client/Server, C/S)结构的数据库系统,它是目前普遍使用的数据库系统。

#### 5. 关系数据库

关系数据库(Relational Database)是目前使用最广泛的数据库,它以关系模型作为数据的组织存储方式。关系数据库通常包含多张表,表由若干记录组成,记录由若干字段组成。

- 表(Table): 一个表就是一组相关的数据按行排列,像一张表格一样。比如一个班所有学生的基本情况存在一个表中,如表 9-1 所示。
- 字段(Field): 在表中,每一列称为一个字段。每一个字段都有相应的描述信息,如数据类型、数据宽度等。
- 记录(Record): 在表中,每一行称为一条记录。
- 索引(Index): 索引是按照指定字段建立的顺序链表,能加快访问数据库的速度。

表 9-1 中每一行对应一名学生,在这一行中,包括学生的学号、姓名以及入学成绩等信息,一行的信息又称为记录。其中学号是唯一的,称为主键(Key);表头列名表示学生的一个属性,称为字段。另外,表显示的时候可以建立按学号索引,方便排序查找。

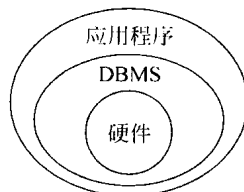


图 9-1 数据库系统的组成

表 9-1 学生基本情况

		字段					
主键	学 号	姓 名	年 龄	性 别	院(系)	籍 贯	入学成绩
按学号索引	03001	张维	20	男	计算机	山西	621
	03002	张珊	19	女	电子	湖南	578
	03003	李岷	21	男	机械工程	山东	603
	...	...	...	...	...	...	...

为了维护数据库中数据与现实世界的一致性,关系数据库中的数据进行更新与删除操作必须遵循以下三类完整性约束:

#### 1) 实体完整性规则

若属性 A 是基本关系 R 的主属性,则属性 A 不能取空值。实体完整性规则规定基本关系的所有主键的值不能为空,如果出现空值,那么主码起不了唯一标识的作用。

#### 2) 参照完整性规则

现实世界中的实体之间往往存在某种联系,在关系模型中实体及实体间的联系都是用关系来描述的。这样就自然存在着关系与关系间的引用。参照完整性规则就是定义外码与主码之间的引用规则。若属性 F 是基本关系 R 的外码,它与基本关系 S 的主码 Ks 相对应(基本关系 R 和 S 不一定是不同的关系),则对于 R 中每个元组在 F 上的值必须为:或者取空值(F 的每个属性值均为空值);或者等于 S 中某个元组的主码值。换句话说,外键要么在参照关系中有值,要么取空值。

#### 3) 用户定义的完整性

任何关系数据库系统都应该支持实体完整性和参照完整性规则。除此之外,不同的关系数据库系统根据其应用环境的不同,往往还需要一些特殊的约束条件,用户定义的完整性就是针对某一具体关系数据库的约束条件。它反映某一具体应用所涉及的数据必须满足的语义要求。例如某个属性必须取唯一值、某些属性值之间应满足一定的函数关系、某个属性的取值范围在 0~200 之间等。关系模型应提供定义和检验这类完整性的机制,以便系统进行统一处理,而不要由应用程序承担这一功能。

### 6. 数据库设计

在开发数据库应用程序之前,必须进行需求分析,对数据库的概念结构、逻辑结构和物理结构进行规范设计,这是决定数据库应用程序开发成败的关键。根据数据库原理,数据库设计一般分成 6 个阶段,如图 9-2 所示。

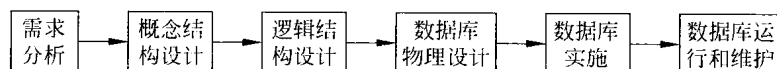


图 9-2 数据库设计步骤

设计一个完善的数据库应用系统,往往是这 6 个阶段不断重复的过程。需求分析是数据库设计的基础。不论采用什么方法,都必须扎扎实实搞好需求分析。概念设计阶段的关键是对需求分析的结果进行抽象,确定实体、属性以及实体之间的联系。逻辑设计阶段主要

是将 E-R 图向关系模型转换,要解决的关键问题是如何将实体和实体间的联系转换为关系模式,以及如何确定这些关系模式的属性和码,处理好属性命名的冲突问题,具有相同码的关系模式进行合并。数据库物理设计及实施是逻辑设计结果的具体实现,完全依赖于给定的硬件环境和数据库产品。总之,数据库设计各阶段是密切相关的,应保证各阶段工作的一致性和规范化。

9.1.2 Delphi 可访问的主要数据库产品简介

在 Delphi 中要开发数据库应用程序,首要的任务就是访问数据库。Delphi 2007 可以访问多种数据库,如表 9-2 所示,可以是早期的 dBASE 数据库、Visual FoxPro、Paradox 和 Access 等基于文件型的数据库;也可以是应用比较广泛的 InterBase、Oracle、Infomix、SQL Server 和 DB2 等大、中型网络数据库。对这些数据库的访问可以通过早期的 BDE 技术或者 ADO 组件中的 OLE DB 技术进行直接访问,也可以通过 ODBC 统一接口技术进行访问。

表 9-2 Delphi 2007 可以访问的数据源

序号	可访问的数据源	特性表述	扩展名
1	dBase 数据库: dBase、Visual FoxPro	数据库表通过 dBase 数据库管理系统建立,每个表是一个独立的文件	.DBF
2	Paradox 数据库	数据库表可通过 Paradox 数据库管理系统建立,每个表是一个独立的文件	.DB
3	Access 数据库	该数据库通过 MS Access 数据库管理系统建立,多个表包含在一个数据库文件中	.MDB
4	本地 InterBase 数据库服务器	该数据库通过 InterBase 数据库管理系统建立,多个表包含在一个数据库文件中	.GDB
5	SQL 数据库服务器: Oracle、InterBase、SQL Server 和 DB2	数据库通过相应的数据库服务器来提供,依赖不同的工具建立	依赖于不同的数据库管理系统
6	ODBC 数据源	具有 ODBC 接口的数据库,如 MS Access、SQL Server 和 Btrieve 等	ODBC 扩展名为 .DSN

其中 Visual FoxPro、Access 和 Paradox 等属于单用户版数据库产品。这类数据库的数据被按照一定格式储存在磁盘里,使用时由应用程序通过相应的驱动程序甚至直接对数据文件进行读取。也就是说,这种数据库的访问方式是被动式的,只要了解其文件格式,任何程序都可以直接读取,所以它的安全性十分不好、效率低下,不支持部分 SQL 命令,也不支持视图、触发器和存储过程等高级功能,这些特点决定了它越来越不适合数据库发展的需要。

Visual FoxPro 数据库是由 dBASE 数据库发展而来的,它的每一个表是用单独文件(.dbf)存放的,不支持密码保护。

Access 数据库是 MS Office 里的套件之一,和 Visual FoxPro 数据库不同,所有的表、索引都被整合在一个文件中(.mdb),这样就避免了数据库变大之后管理上带来的麻烦。同时它还提供密码保护功能,能与 MS Office 无缝结合。

早期 Delphi 自带的 Paradox 是 Borland 公司自己的产品,因此和 Borland 的系列开发

工具配合紧密。它支持密码保护,支持标准的 SQL,性能也还不错。它的每一个表都是一个独立的文件(.db),因为这些特点所以应用并不广泛。Delphi 2007 还可以通过 BDE 访问 Paradox,但是需要用户自行安装 BDE 驱动。

InterBase 是一种关系数据管理系统 (Relational Database Management System, RDBMS),提供了在单机或多用户环境中快速数据处理及共享的工具。InterBase 的核心是提供透明的多机种支持的网络运行服务器技术。

而 MS SQL Server、Oracle Universal Server、Informix-Universal Server 和 IBM DB2 Server 等属于大型数据库。这类数据库与单用户版数据库产品不同,它们的数据集中存放在服务器上,统一由运行在服务器上的数据库服务程序管理,用户使用客户端软件通过网络访问数据库服务程序,如图 9-3 所示。

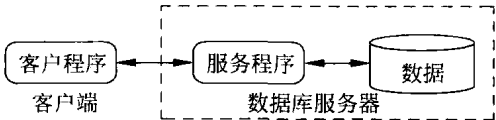


图 9-3 客户端/服务器数据库系统结构

这种类型数据库的特点是适合于网络应用,可以同时被多个用户所访问,数据库管理系统可以赋予不同的用户以不同的安全访问权限,支持的数据量大,能完全支持 SQL 语言。

9.1.3 Delphi 连接数据库方式

Delphi 对数据库的访问方法有多种,可以通过早期的 BDE 技术或者 ADO 组件中的 OLE DB 技术进行直接访问,也可以通过 ODBC 统一接口技术进行访问。这里重点介绍基于 ADO 技术的连接方式。

ADO(Microsoft ActiveX Data Objects)是微软公司开发的适用于 Windows 操作系统的数据库访问技术。OLE DB 可提供对存储在不同信息源的数据进行统一访问的能力,它是微软公司开发的一种底层数据库访问技术。OLE DB 无法满足应用程序设计的简单化要求,并不适用于所有语言。所以,微软公司在 OLE DB 基础上开发了 ADO,它位于 OLE DB 的上层,封装了 OLE DB 的所有功能,为那些不能直接访问 OLE DB 的语言提供编程接口。

1. Delphi 中连接数据库的 ADO 技术概述

Delphi 对 ADO 技术的支持,是通过将 ADO 常用对象封装进 Delphi 的 ADO 组件中,并结合 Delphi 本身的开放式数据库组件结构实现的。使用 ADO 组件,可以迅速实现数据库连接,建立数据库应用,利于升级和维护开发的系统。

Delphi 的 ADO 组件共有 7 个,位于 dbGo 面板上,如图 9-4 所示,分别是 TADOConnection、TADODataSet、TADOTable、TADOQuery、TADOStoredProc、TADOCommand 和 TRDSCConnection。其中后面的 6 个组件可以直接连接到数据库,但更为常用的方式是通过 TADOConnection 组件连接到数据库。



图 9-4 dbGo 页面中的 ADO 组件

这些组件的作用与功能概述如下：

(1) TADOConnection 组件：用于建立数据库的连接。ADO 的数据源组件和命令组件可以通过该连接组件运行命令,以及从数据库中提取数据。

(2) TADODataSet 组件：这是 ADO 提取及操作数据库数据的主要数据集,该组件可以从一个或多个基表中提取数据。

(3) TADOTable 组件：主要用于操作和提取单个基表的数据。与 TADODataSet 类似，它可以直接连接到数据库，也可以通过 TADOConnection 组件连接到数据库。

(4) TADOQuery 组件：该组件是通过 SQL 语句实现对数据库数据的提取及操作。它可以直接运行数据定义语言(DDL)，如 CREATE、ALTER 和 DROP 等 SQL 命令。其连接数据库的方式与前两种数据集一样。

(5) TADOStoredProc 组件：该数据集是专门用于运行数据库中的存储过程的。这些存储过程可能会提取数据，也可能不提取数据。

(6) TADOCommand 组件：用于运行一些 SQL 命令。这些命令没有数据集返回，所以该组件不是一个数据集组件。该组件可以与支持数据集的组件一起使用，也可以直接从基表中提取数据集传递给记录集。

(7) TRDSCONNECTION 组件：一个进程或一台计算机传递到另一个进程或计算机的数据集合，用于远程数据访问。

利用 ADO 访问数据的一般过程为：首先向应用程序添加一个 ADOConnection 组件用于建立与数据库的连接，然后使用一个 ADOConnection 组件或者 ADOQuery 组件向数据库发送 SQL 命令，最后通过数据集获得数据。这时，数据集组件必须将 Connection 属性指向所使用的 ADOConnection 组件。

## 2. Delphi 中 ADO 连接数据库的操作过程

Delphi 程序连接的数据库即可以是本地的数据库文件，也可以是远程的数据库服务器上的数据库。连接到各种数据库如远程的 MS SQL Server、本地 Access 或者 ODBC 数据源的操作过程基本差不多，只不过某些窗口选项不一样。下面通过一个使用 OLD DB 提供者连接 Access 数据库实例说明连接过程。将 TADOConnection 组件连接到数据库是通过设置 ConnectionString 属性实现的。

具体步骤如下：

(1) 在 Delphi 2007 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令，创建一个新的应用程序，系统会自动创建一个空白的窗体。

(2) 从组件面板上选择 dbGo 页，单击 TADOConnection 组件图标，然后单击窗体，添加一个 TADOConnection 组件，使用默认名 ADOConnection1，如图 9-5 所示。

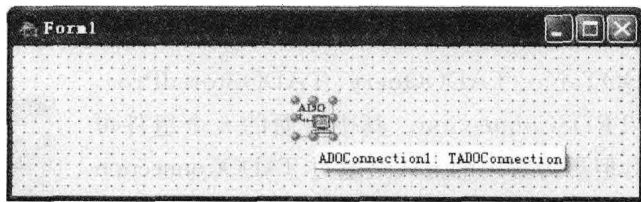


图 9-5 新建窗体中放置 TADOConnection 组件

(3) 双击 ADOConnection 组件或在 Object Inspector 上单击 ConnectionString 属性框，则显示图 9-6 所示的 ConnectionString 设置对话框。Use Connection String(使用连接字符串)是最常用的一种方式。在这里连接字符串可以直接输入，也可以通过 Build 弹出“数据链接属性”对话框生成。

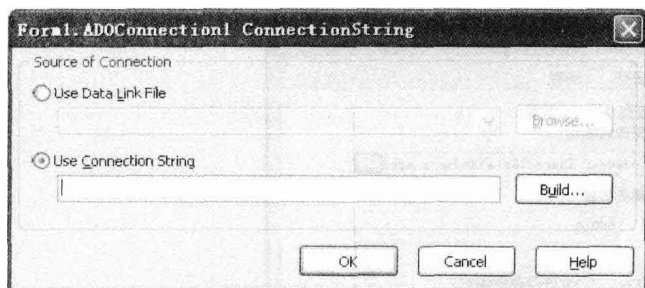


图 9-6 ConnectionString 设置对话框

(4) 使用默认的 Use Connection String,在该对话框中用鼠标单击 Build 按钮,弹出“数据链接属性”对话框,如图 9-7 所示。这是一种类似向导的界面,可以帮助用户一步一步地设置连接字符串。然后从“OLE DB 提供程序”列表框中选择 Microsoft Jet 4.0 (OLE DB Provider,单击“下一步”按钮。

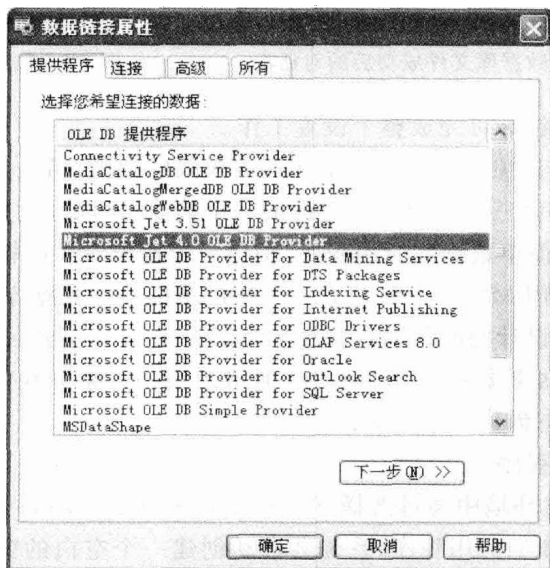


图 9-7 选择数据连接“提供者”的对话框

(5) 进入该对话框的“连接”选项卡,在该选项卡的“1. 选择或输入数据库名称:”下的文本框中可以直接输入数据库的路径与名称。单击右侧“...”按钮,则出现 \*.mdb 文件选择对话框,假设选择 Delphi 2007 系统自带的 C:\Program Files\Common Files\CodeGear Shared\Data\dbdemos.mdb。设置好以后的对话框如图 9-8 所示。

本 Access 数据库不需要输入密码,所以在该页面的“2. 输入登录数据库的信息:”项不需要用户操纵。若连接到有密码的 Access 数据库,还可以通过代码手工设置连接串中的 Jet OLEDB;Database Password 参数。例如 Jet OLEDB;Database Password=xatujsj。

(6) 在“连接”选项卡中,单击“测试连接”按钮,如果弹出“测试连接成功”的消息框,证明选择数据库连接操作成功,如图 9-9 所示。

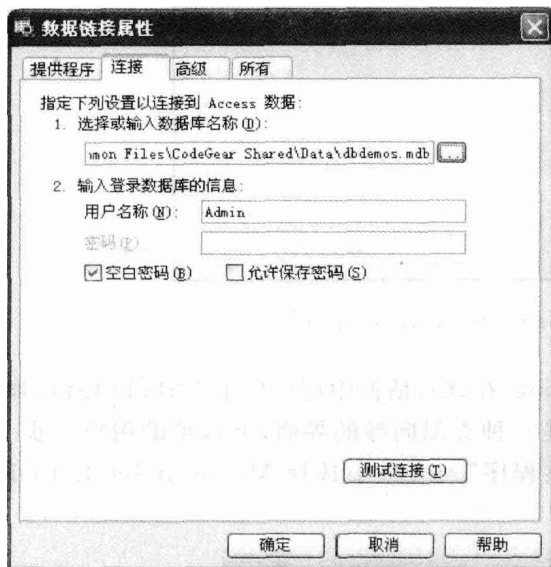


图 9-8 择数目标数据库文件成功后的对话框



图 9-9 测试连接成功的消息窗口

(7) 单击“确定”按钮即可完成整个设置工作。

对话框中的“高级”选项卡是关于数据在存取权限的细微设置,具体的可参看相关手册。“所有”选项卡则是对前面三个选项卡设置值的一个摘要,可以对设置的值进行手工调整。

**例 9.1** 编写一个简单数据库应用程序,构建一个通讯录表的输入输出窗口。

**数据准备:** 首先利用 MS Access 建立一个通讯录的空白数据库,其文件名为 tx1.mdb。

**设计构想:** 本程序设计的时候通过 TADOConnection 连接到目标数据库 tx1.mdb。程序窗体初始化时,检测该数据库里是否存在“通讯录”表,如果存在则打开;反之不存在,由程序使用 SQL 语句动态创建该表。

(1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“一个简单通讯录应用程序示例”。

(2) 用户界面设计。

① 在窗体上添加一个用于建立数据库的连接 ADOConnection1 组件(位于组件面板 dbGo 页),通过前述的步骤设置其成功地连接到目标数据库 tx1.mdb,并将其属性 LoginPrompt 设置为 False。

② 在窗体上放置一个用于执行 SQL 命令建表的 ADOCommand1 组件(位于组件面板 dbGo 页),设置其 Connection 属性为 ADOConnection1。

③ 在窗体上放置用于映射数据库表的 ADOTable1 组件(位于组件面板 dbGo 页),设置其 Connection 属性为 ADOConnection1。

④ 在窗体上放置一个数据源 DataSource1 组件(位于组件面板 Data Access 页),设置其 DataSet 属性为 ADOTable1。

⑤ 在窗体上放置一个控制数据库表的导航 DBNavigator1 组件(位于组件面板 Data

Controls 页),设置其 DataSource 属性为 DataSource1。然后设置其 Align 属性为 alTop。

⑥ 在窗体上放置一个控制读取或者写入数据的表格 DBGrid1 组件(位于组件面板 Data Controls 页),设置其 DataSource 属性为 DataSource1。然后设置其 Align 属性为 alClient。

窗体的设计界面如图 9-10 所示。

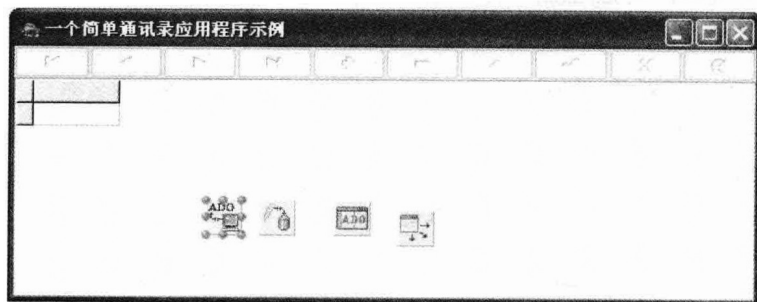


图 9-10 简单的通讯录应用程序设计界面

### (3) 代码编写。

首先需要在窗体单元文件 Unit1 的接口中声明自定义函数:

```
{ Private declarations }  
function fTableExists:Boolean;                                //自定义函数检测通讯录表是否存在
```

实现部分写出其详细代码:

```
//自定义函数  
function TForm1.fTableExists:Boolean;                        //检测通讯录表是否存在  
var  
    SL: TStringList;  
    index: Integer;  
begin  
    Result := False;  
    if not ADOConnection1.Connected then exit;                //如果没有连接,则退出  
    SL := TStringList.Create;                                  //创建列表  
    try  
        ADOConnection1.GetTableNames(SL, False);              //取用户的表  
        Result := (SL.IndexOf('通讯录')>= 0);                 //返回是否存在  
    finally  
        SL.Free;                                                //列表释放  
    end;  
end;
```

其他事件代码如下:

```
//窗体创建事件  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    ADOConnection1.Connected := True;                          //激活连接  
    if not fTableExists then                                    //如果数据库中不存在通讯录表  
    begin
```





```

try
    ADOCommand1.CommandText := 'CREATE TABLE 通讯录(姓名 varchar(10),' +
        '性别 varchar(2) DEFAULT ''男'',地址 varchar(20),电话 varchar(12),手机 varchar(12))' +
        'PRIMARY KEY(姓名),备注 varchar(40),自动编号 int IDENTITY(1,1))';
    ADOCommand1.Execute;
except
    Application.MessageBox('创建失败.', '请注意', MB_OK);
end;
end;
ADOTable1.TableName := '通讯录';           //设置表的名称
ADOTable1.Open;                             //打开数据库表
end;
//窗体释放事件
procedure TForm1.FormDestroy(Sender: TObject);
begin
    ADOConnection1.Connected := False;       //断开跟数据库连接
end;

```

#### (4) 保存并运行。

选择 File→Save 命令,保存单元文件与项目文件。程序开始运行后,用户可以插入、修改和删除通讯记录,实例效果如图 9-11 所示。



图 9-11 简单的通讯录应用程序运行界面

本程序的数据库通讯录表如果不存在,通过 SQL 语句可以自动创建,体现了 Delphi 具有自动创建表的能力。程序中用到的很多组件本章后续将详细介绍,读者只要会模仿做类似的题就行,现在对这些数据组件的属性和用法有了感性认识,后面理解起来就更容易。

### 9.1.4 数据库应用程序结构

从程序设计的角度来说,无论是访问本地的 Visual FoxPro 数据库、Access 数据库、Paradox 数据库,还是访问远程 InterBase、Oracle、Infomix、SQL Server 和 DB2 等大、中型网络数据库,其基本结构都是相同的。

在 Delphi 2007 的数据库应用程序中,最基本的构成是三类组件:数据控制组件、数据集组件和数据源组件。

(1) 数据控制组件(Data Control):用于显示和修改数据库中的信息,为用户操作数据库提供一个可视化的界面,常用的组件有 TDBEdit、TDBText、TDBGrid 和 TDBGrid。数据控制组件位于 Data Control 页。

(2) 数据集组件(DataSet): 一方面通过 ADO 与实际的数据库相连接,另一方面通过 TDataSource 组件与数据库控制组件相连,常用的有 TADODataSet、TADOTable 和 TADOQuery 组件。ADO 数据集组件位于 dbGo 页。

(3) 数据源组件(DataSource): 在 Data Access 页,负责将数据集组件和数据控制组件连接起来,使用时需要设定主要属性 DataSet,指定具体的数据集组件名称(用于指示数据源与哪个数据集对象相联系)。常用的是 TDataSource 组件。

使用 ADO 组件编写的应用程序通常具有图 9-12 所示的结构。

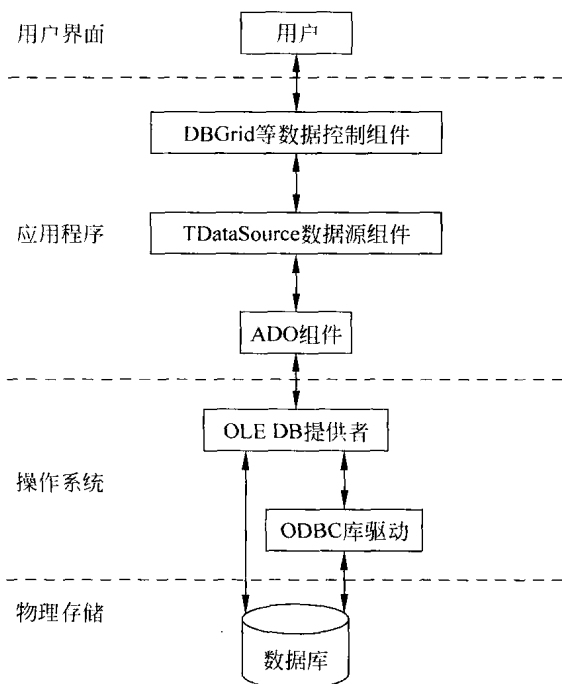


图 9-12 基于 ADO 的数据库应用程序结构

## 9.2 Delphi 数据集组件

Delphi 2007 是目前开发各种数据库应用程序的强有力工具。借助于 DBMS 的 API 接口函数或者组件使得 Delphi 具有强大的数据库定义、操纵、控制功能,利用 Delphi 提供的完善的数据库访问操作组件,甚至根本不需要编写任何代码便可以创建一个简单的数据库应用。

数据库组件对象的数据成员既可在设计阶段设置,也可在运行阶段通过程序代码进行设置。

### 9.2.1 数据集概述

在 Delphi 中,可以使用 VCL 数据库组件编写 Win32 框架下的 VCL Forms 应用程序,也可以采用 .NET 组件编写 .NET 框架下的 Windows Forms、ASP.NET Web 和 Web 服务等应用程序。VCL Forms 应用程序开发的数据库组件主要有 Data Access(数据访问组

件)、Data Controls (数据控制组件)、dbExpress、DataSnap、BDE、dbGo(ADO)和 InterBase 组件等。

BDE(Borland Database Engine)是 Borland 公司早期开发的一种数据存取引擎, Delphi 2007 仍然支持,但是必须额外安装其驱动。

数据集组件包含一个数据缓冲区,通常说的数据集就是这个缓冲区,数据控制组件中显示的数据就是缓冲区中的数据。缓冲区中有一个记录指针指向当前访问的记录。数据控制组件修改数据或者新增数据时,只是对缓冲区做修改,最后再由数据集组件提交到数据库。当然也可以使用数据集组件的方法和属性来对数据库进行操作。图 9-13 所示为 Delphi 各类数据集组件树型层次继承图。

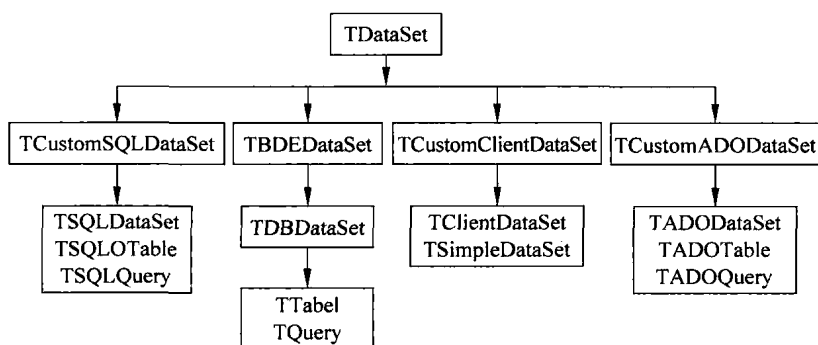


图 9-13 Delphi 数据集树型层次继承图

在 BDE 中,有三种类型的数据集组件: TTable、TQuery 和 TStoredProc。这三种类型的数据集组件的直接上级是 TDBDataSet 类,而 TDBDataSet 类的直接上级是 TBDEDataSet 类,而 TBDEDataSet 类是从 TDataSet 类继承下来的。

在 Delphi 2007 的组件面板中,dbGO 页面里提供的数据集组件主要有 TADODataset、TADOTable、TADOQuery 和 TADOStoredProc。

dbExpress 页面里提供的数据集组件主要有 TSQLDataSet、TSQLTable、TSQLQuery、TSQLStoredProc 和 TSimpleDataSet。

这些数据集的很多属性、方法和事件基本相同,下面重点讲解最常用的 TADOTable 数据集组件。

### 9.2.2 TADOTable 数据集组件的主要属性、方法与事件

TADOTable 数据集组件位于组件面板的 dbGo 页面,如图 9-14 所示,它只能通过 ADO 方式访问数据库中单个数据表的数据,因此,它是 ADO 数据集组件中最简单并且非常重要的一个组件。

#### 1. TADOTable 的主要属性说明

(1) Active 属性: 确定数据集是否处于打开状态。设置 Active 属性为 True,则数据集被打开,相当于调用 Open 方法,可以对数据库中的表进行读或写操作;设置 Active 属性为 False,则数据集被关闭,相当于调用 Close 方法。

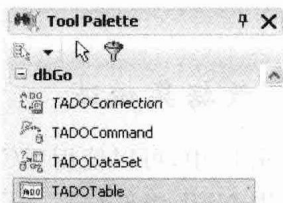


图 9-14 位于 dbGO 页的 TADOTable 组件

设置 Active 为 True,将触发 BeforeOpen 事件,然后设置 state 属性为 dsBrowse,打开一个数据库表游标,然后触发 AfterOpen 事件;设置 Active 为 False,将触发 BeforeClose 事件,然后设置 state 属性为 dsInactive,最后触发 AfterClose 事件。在更改影响到数据库表的状态或数据显示组件的状态的数据集属性时,要提前将数据集的 Active 属性设置为 False。

(2) State 属性:表明当前数据集的状态,其取值的含义如表 9-3 所示。

表 9-3 State 属性的取值及含义

状 态 值	含 义
dsInactive	数据集处于关闭状态
dsSetKey	数据集处于查找键状态
dsBrowse	数据集处于浏览状态,用户可以浏览数据,但不能修改
dsEdit	数据集处于编程状态
dsInsert	数据集处于插入状态
dsCalcFields	数据集处于计算字段数据时的状态,这时不能修改数据
dsFilter	数据集处于过滤状态,可以显示数据,不能修改
dsNewValue	这是一个临时状态,当某个字段对应的 TField 组件的 NewValue 属性被修改时数据集处于这个状态
dsOldValue	这是一个临时状态,当某个字段对应的 TField 组件的 OldValue 属性被修改时数据集处于这个状态
dsCurValue	这是一个临时状态,当某个字段对应的 TField 组件的 CurValue 属性被修改时数据集处于这个状态
dsBlockRead	数据集处于禁止读取状态
dsInternalCalc	数据集处于内部计算状态
dsOpening	数据集处于正被打开的状态

(3) Connection 属性和 ConnectionString 属性:指定所使用的数据源连接组件的名称,即 TADOConnection 组件的名称。通过这个属性使得 TADOTable 组件能与数据库连接起来。ConnectionString(连接字符串)用于指定数据库的连接信息,与 TADOConnectin 组件中的作用相同。Connection 属性与 ConnectionString 属性是互斥的,设置了其中一个,另外一个就被自动清除。

(4) CacheSize 属性:指定数据集的缓冲区大小。其原型为 property CacheSize: Integer;,默认值为 1,也是其最小值。数据集首先把数据从数据库中取出,然后保存在内存的一块区域中,这块内存区域就是所谓的缓冲区。

(5) CursorLocation 属性:指定数据库游标是采用客户端模式还是服务器端模式。其原型为 property CursorLocation: TCursorLocation;,取值为 clUseServer,含义为使用服务器端的数据库游标,适用于数据量大的数据集;或者取值为 clUseClient,含义为使用客户端的数据库游标时,数据将被下载到本地计算机上,并在本地进行操作。

(6) AutoCalcFields 属性:该属性设为 True 则允许应用程序触发 OnCalcFields 事件。计算字段依赖于当前记录的一个或多个字段,通过已有的字段数据进行计算。该值为 True,在记录数据被修改或者编辑时就触发 OnCalcFields 事件,应用程序自动更新计算字段的值,以保证数据的一致性。

(7) BOF 属性和 EOF 属性:BOF 属性判断当前记录指针的位置是否位于文件开始位

置,EOF 属性用于判断当前记录指针的位置是否位于结束处。它们都是只读的,为 Boolean 型。BOF 属性为 True 时,表示当前指针指向第一条记录。EOF 属性为 True 时,表示当前指针指向最后一条记录。Eof 经常用于程序语句中判断数据是否处理完毕。

例如:

```
if ADOTable1.EOF then
    ShowMessage('已经到了表的末尾。');
```

(8) Fields 属性: 表示数据集中的字段集合,用于访问数据集中的字段。

例如:

```
Edit1.Text := ADOTable1.Fields.Fields[2].AsString;    //读取字段值到编辑框中
ADOTable1.Edit;                                       //将数据集设为编辑状态
ADOTable1.Fields.Fields[2].AsString := Edit1.Text;   //将编辑框中数据写入字段
ADOTable1.Post;                                       //提交到数据库保存
Label1.Caption := ADOTable1.Fields[3].FieldName;     //将字段名称显示到标签上
```

(9) MasterSource 属性: 用于指定一个数据源组件,一般用于主-从表关系应用中。

(10) ReadOnly 属性: 指定数据集中的数据是否只读。ReadOnly 属性为 True,表示数据只能浏览,不能修改;反之为 False 时,数据集中的数据既可以浏览,也可以修改。

(11) TableDirect 属性: 指定这个表是通过表名访问还是通过 SQL 语句访问(仅限 Select 语句)。TableDirect 属性为 True 表示可以通过 SQL 语句访问,为 False 则只能通过表名访问。默认为 False。

(12) TableName 属性: 指定数据库的物理表名,数据集才能映射物理数据表。

2. TADOTable 重要方法的说明及使用

(1) Open 和 Close 方法: 这两个方法等同其 Active 属性值的设置。调用 Open 方法等同于将 Active 属性设置为 True。只有当 Active 属性为 True 时,可以从数据库表中读取数据或向数据库表中写数据。调用 Close 方法等同于将 Active 属性设置为 False。当 Active 属性值为 False 时,数据集将被关闭,无法对数据库表进行读/写操作。

例如,在实际程序代码中,语句 ADOTable1.Open;等价于语句 ADOTable1.Active := True;;而语句 ADOTable1.Close;等价于语句 ADOTable1.Active := False;;

(2) DeleteRecords 方法: 作用是删除数据集中的记录。在 Delphi 中定义原型为:

```
procedure DeleteRecords(AffectRecords: TAffectRecords = arAll);
```

AffectRecords 参数用于指定要删除的具体记录,其取值主要有 arCurrent、arFiltered、arAll 和 arAllChapters。这些参数的含义如表 9-4 所示。默认值为 arAll,删除当前记录集中的全部数据。

表 9-4 AffectRecords 参数的取值及功能

取 值	功 能
ArCurrent	仅删除当前记录
ArFiltered	删除满足过滤器过滤条件的数据
ArAll	删除记录集中的所有记录
ArAllChapters	删除 ADO 连接数据部分的全部记录

(3) SaveToFile 方法：该方法的功能是把当前数据集中的数据按照指定的格式保存到指定的文件中。其原型为：

```
procedure SaveToFile(const FileName: String = ''; Format: TPersistFormat = pfADTG);
```

参数 FileName 为指定的文件名,Format 为保存的文件格式,它可以选取下列值: pfADTG,按照 ADTG(Advanced Data Tablegram)格式生成文件; pfXML,按照 XML 格式保存文件,需要 ADO 2.1 或更高版本支持。

(4) GetIndexNames 方法：本方法的功能是获取表中的全部索引名。其原型为：

```
procedure GetIndexNames(List:TStrings);
```

该方法执行完毕,返回值将保存在参数 List 中,例如：

```
ADODataset1.GetIndexNames(ListBox1.Items);
```

(5) FieldByName 方法：该方法的功能是根据一个特定的字段名获取一个字段对象。该方法的原型为：

```
function FieldByName(const FieldName:string):TField;
```

通过 FieldByName 属性,应用程序可以直接获得关于该字段的特殊的属性和方法,而不用通过字段名数组的索引。

3. TADOTable 组件的主要事件

TADOTable 组件的主要事件如表 9-5 所示。

表 9-5 TADOTable 数据集类组件的主要事件

事件名称	说明
AfterCancel	在 Cancel 方法之后触发该事件
AfterClose	在 Close 方法之后触发该事件
AfterDelete	在 Delete 方法之后触发该事件
AfterEdit	在数据集设置为编辑模式之后触发该事件
AfterInsert	当一个新记录插入到该数据集中时触发该事件
AfterOpen	打开数据集之后触发该事件
AfterPost	在 Post 方法之后触发该事件
AfterRefresh	Refresh 方法之后触发该事件
AfterScroll	当光标定位被改变时触发该事件
BeforeCancel	当调用 Cancel 方法的时候,在执行 Cancel 行为之前触发该事件
BeforeClose	当调用 Close 方法的时候,在执行 Close 行为之前触发该事件
BeforeDelete	当调用 Delete 方法的时候,在执行 Delete 行为之前触发该事件
BeforeEdit	当调用 Edit 方法的时候,在执行 Edit 行为之前触发该事件
BeforeInsert	当调用 Insert 方法的时候,在记录被插入到数据集之前触发该事件
BeforeOpen	当调用 open 方法的时候,在数据集被打开之前触发该事件
BeforePost	当调用 Post 方法的时候,在记录被发送到数据集之前触发该事件
OnCalcFields	当 Calculated 字段需要指定一个值的时候触发该事件
OnEditError	在修改或添加记录时产生了意外会触发该事件
OnEndOfRecordset	在记录指针试图移动到末尾之后时发生

事件名称	说 明
OnFetchComplete	当一次从数据库读取数据的操作完成时触发该事件
OnFetchProgress	在异步方式运行时,数据集组件读取数据时会周期性触发该事件
OnFieldChangeComplete	当某个字段被更新后触发该事件
OnFilterRecord	编写这个事件处理程序来测试过滤条件对于每个记录的过滤过程
OnNewRecord	当一个新的记录添加到数据集中的时候触发该事件
OnPostError	在提交修改的记录时产生了意外会触发该事件
OnRecordChangeComplete	当一条记录被修改完成后触发该事件
OnRecordsetCreate	在记录集被初始化完成后触发该事件
OnWillChangeField	在字段更新之前发生
OnWillChangeRecord	在记录被改变之前发生
OnWillChangeRecordset	在记录集被更新之前发生
OnWillMove	在移动记录指针之前发生

### 9.2.3 TADOTable 数据集组件记录的读取与修改

ADO 数据集组件提供了丰富的方法和属性用于从记录中读取数据和修改记录中的数据,对记录的读取与修改操作的实质就是对字段(Field)的读取与修改操作。

#### 1. Field 对象

Field 对象对应着数据集中的字段。Field 对象可以在应用程序运行的过程中动态地产生,也可以利用字段编辑器创建成为永久字段。字段对象的数据类型也是与数据集中字段的数据类型相对应的。

数据集的 Fields 属性是 TField 类型对象的集合(即 Field 对象集合)。

TFields 有两个基本属性:一个是 Count 属性,指明 Fields 对象中的字段数;一个是 Fields 属性,包含 Fields 对象所管理的字段列表,通过指定索引号可以访问单字段,索引号从 0 开始。

假设 Field[1]为整型字段,如果要读取字段的值,可以使用语句:

```
NumV := ADOTable1.Fields.Fields[1].AsInteger;           //读取字段整型值送到变量中
Edit1.Text := ADOTable1.Fields[1].AsString;             //获取字段字符串值
Edit1.Text := ADOTable1.Fields.Fields[1].Value;         //Value 属性返回字段值
```

如果要修改字段的值,可以使用程序段:

```
ADOTable1.Edit;                                           //将数据集设为编辑状态
ADOTable1.Fields.Fields[1].AsInteger := StrToInt(Edit1.Text);
ADOTable1.Post;                                           //提交到数据库保存
```

#### 2. 使用 FieldValues 属性

使用 FieldValues 属性,可以通过字段名读取字段的值。如果要获取某字段的内容,可使用“数据集名.FieldValues['某字段名']”,该表达式返回该字段的值转换成字符串的结果。FieldValues 是数据集默认属性,可以省略。例如:

```
Label1.Caption := AdoTable1.FieldValues['地址'];
```

或者

```
Label1.Caption := AdoTable1 ['地址'];
```

### 3. 使用 FieldByName 方法

FieldByName 方法返回指定名称的字段对象,因此,可以使用 FieldByName 方法完成对字段的操作。

```
function FieldByName(const FieldName: String): TField;
```

例如,要显示某字段的值可以使用语句:

```
Label1.Caption := ADOTable1.FieldByName('某字段名').AsString;
```

## 9.2.4 TADOTable 数据集组件记录的添加与删除

可以使用数据集组件的 Append、Insert、AppendRecord 和 InsertRecord 方法添加记录,其中 Append 是在表末尾添加一条空白记录,Insert 是在表的当前位置添加一条空白记录,AppendRecord 用于在表末追加一条记录并对其赋值,InsertRecord 是在表的当前位置添加一条记录并对其赋值。调用数据集的 Delete 方法和 DeleteRecord 方法可以删除记录。

### 1. 使用 Append 和 Insert 方法添加记录

使用 Append、Insert 添加空记录后,数据集处于 dsInsert 状态,这时可以对字段赋值,然后调用 Post 方法提交数据或调用 Cancel 方法撤销操作。

例如,使用 Append 方法在表 customer 中添加一条新记录:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ADOTable1.Append;                                //添加新的一条空白记录
    ADOTable1.FieldValues['Company'] := Edit1.text;  //给字段赋新值
    ADOTable1.FieldValues['CustNo'] := StrToInt(Edit2.text); //给字段赋新值
    ADOTable1.Post;                                   //提交数据
end;
```

### 2. 使用 AppendRecord、InsertRecord 方法添加记录

这两个方法在 Delphi 中定义的原型为:

```
procedure AppendRecord(const Values : array of const );
procedure InsertRecord(const Values : array of const );
```

例如,给表添加一条记录可以使用下面的语句:

```
ADOTable3.AppendRecord(['李龙','男','清华大学']);
```

### 3. 使用 Delete 方法

Delete 方法用于删除当前记录。

例如,要删除当前记录,使用如下语句:

```
ADOTable1.Delete;
```

### 4. 使用 DeleteRecords 方法

该方法可以删除记录集中的当前记录或者所有记录。删除当前记录时使用参数



arCurrent, 删除所有记录时使用参数 arAll, 系统默认为 arAll。

例如:

```
ADOTable3.DeleteRecords(arCurrent);           //删除 ADOTable3 当前记录
ADOTable1.DeleteRecords(arAll);               //删除 ADOTable1 所有记录
ADOTable5.DeleteRecords();                   //删除 ADOTable5 所有记录
```

### 9.2.5 TADOTable 数据集组件的数据查询方法

ADO 数据集组件提供的有关数据查询的方法有三个: Locate 方法、lookup 方法和 Seek 方法。

#### 1. Locate 方法

该方法的作用是定位一条记录并把这条记录作为当前记录。

该方法在 Delphi 中的原型为:

```
function Locate(const KeyFields: String; const KeyValues: Variant; Options: TLocateOptions): Boolean; override;
```

其中参数 KeyFields 含义为要进行搜索的字段名; KeyValues 含义为要查找的值; Options 含义为定位数据的选项, 它的值可以是 loCaseInsensitive (定位数据不区分大小写)、loPartialKey (部分匹配定位查找数据)。例如:

```
ADOTable1.Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P',
'408 - 431 - 100']), [loPartialKey]);
```

#### 2. Lookup 方法

该方法的功能是寻找符合条件的记录, 获取指定字段的值。找到记录后, 当前的记录指针不移动。

该方法在 Delphi 中的原型为:

```
function Lookup(const KeyFields: String; const KeyValues: Variant;
const ResultFields: String): Variant; override;
```

其中参数 KeyFields 的含义为进行搜索的字段名; KeyValues 的含义为要搜索的字段值; ResultFields 的含义为 Lookup 将该字段的值作为函数的返回值。

例如, 在表 customer 中搜索 CustNo 为 1221 的记录, 然后将 company 的字段内容显示到 Edit1 编辑框中:

```
Edit1.Text := AdoTable1.Lookup('CustNo', 1221, 'company');
```

#### 3. Seek 方法

该方法的功能是搜索指定的记录并移动数据集的指针。如果搜索到索引的值, 则返回 True, 反之为 False。搜索动作是以当前数据集中的索引为搜索依据, 其中 KeyValues 为被搜索的值。

该方法在 Delphi 中的原型为:

```
function Seek(const KeyValues: Variant; SeekOption: TSeekOption = soFirstEQ): Boolean;
```

例如:

Success := ADODataSet1.Seek('Jones', soFirstEQ);

Seek 的第二个参数 SeekOption 限定了搜索行为的动作,其取值与含义如表 9-6 所示。

表 9-6 SeekOptions 的取值及含义

取 值	含 义
soFirstEQ	数据库指针定位在第一条匹配的记录处,如果没有任何匹配记录则指向数据库的末尾记录
soLastEQ	数据库指针定位在最后一条匹配的记录处,如果没有任何匹配记录则指向数据库的末尾记录
soAfterEQ	如果搜索到匹配记录,在指向匹配记录的下一条如果没有找到,则指向最近似匹配的记录上
soAfter	指向匹配记录的下一条
soBeforeEQ	如果搜索到匹配记录,在指向匹配记录的前一条如果没有找到,则指向最近似匹配的记录上
soBefore	指向匹配记录的前一条

该方法也可以同时搜索多个值,这就需要利用函数 VarArrayOf()构造一个数组参数传递给 KeyValues。例如:

ADODataSet1.Seek(VarArrayOf([90030, 90020]), soFirstEQ);

**例 9.2** 编写一个应用程序,对 Delphi 2007 系统自带的 C:\Program Files\Common Files\CodeGear Shared\Data\dbdemos. mdb 数据库中的表 Customer 进行以下操作:根据输入的 CustNo 修改 Company、Addr1 和 Addr2 字段的内容。

设计构想:本程序需要通过 TADOConnection 连接到目标数据库 dbdemos. mdb; 用一个文本框输入 CustNo,根据输入的 CustNo 查询表 Customer 中对应的记录。在文本框中修改 Company、Addr1 和 Addr2 的值后,再将修改后的内容写入到数据库中。为便于观察结果,在窗体中可放置一个 DBGrid 组件。

(1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application--Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“数据库表查找”。

(2) 用户界面设计。

① 在窗体上添加一个用于建立数据库连接的 ADOConnection1 组件(位于组件面板 dbGo 页),通过前述的步骤设置其成功地连接到目标数据库 dbdemos. mdb。并将其 LoginPrompt 属性设置为 False,Connected 属性设置成 True。

② 在窗体上放置用于映射数据库表的 ADOTable1 组件(位于组件面板 dbGo 页),设置其 Connection 属性为 ADOConnection1,选择其 TableName 属性为 Customer,将其 Active 设置为 True。

③ 在窗体上放置一个数据源 DataSource1 组件(位于组件面板 Data Access 页),设置其 DataSet 属性为 ADOTable1。

④ 在窗体上放置一个控制读取数据的表格 DBGrid1 (位于组件面板 Data Controls

页),设置其 DataSource 属性为 DataSource1,DBGrid1 的 ReadOnly 属性为 True。

⑤ 在窗体上放置 4 个 TLabel1 组件和 4 个 TEdit 组件,两个按钮,并且设置它们的 Caption 属性和位置大小。窗体的设计界面如图 9-15 所示。

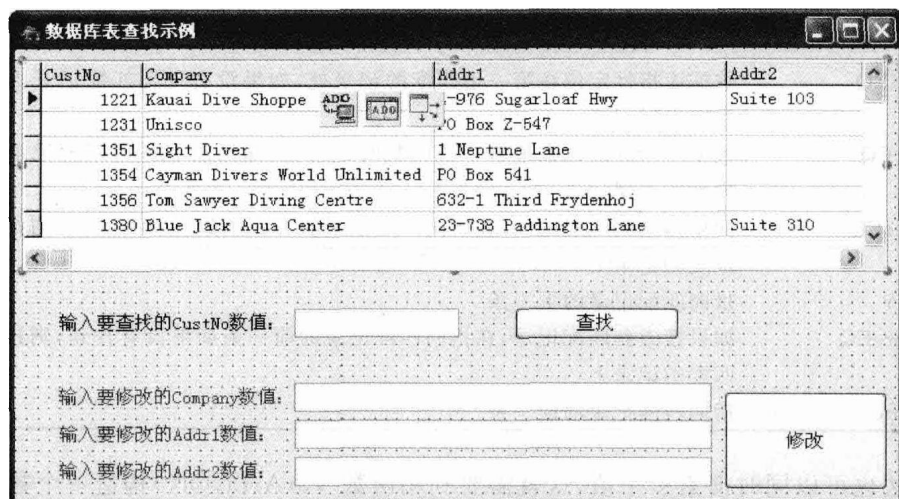


图 9-15 数据库表查找示例设计界面

### (3) 代码编写。

//查找按钮鼠标单击事件

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if Trim(Edit1.Text) <> '' then //Trim 函数是删除左右空格
        if ADOTable1.Locate('CustNo', Trim(edit1.Text), [loPartialKey]) then
            begin
                edit2.Text := ADOTable1.FieldValues['Company']; //显示公司
                edit3.Text := ADOTable1.FieldValues['Addr1']; //显示 Addr1
                edit4.Text := ADOTable1.FieldName('Addr2').AsString; //显示 Addr2
            end;
        end;
```

//修改按钮鼠标单击事件

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    ADOTable1.Edit; //将数据集设为编辑状态
    ADOTable1.FieldValues['Company'] := Edit2.Text; //将 Edit2 中的内容赋给字段
    ADOTable1.FieldName('Addr1').AsString := Edit3.Text; //将 Edit3 中的内容赋给字段
    ADOTable1.FieldName('Addr2').AsString := Edit4.Text; //将 Edit4 中的内容赋给字段
    ADOTable1.Post; //提交到数据库保存
end;
```

### (4) 保存并运行。

选择 File→Save 命令,保存单元文件与项目文件。通过选择 Run→Run 命令运行程序,然后输入一个 CustNo 数值,单击“查找”按钮,定位到该记录,在此基础上可以修改。实

例效果如图 9-16 所示。

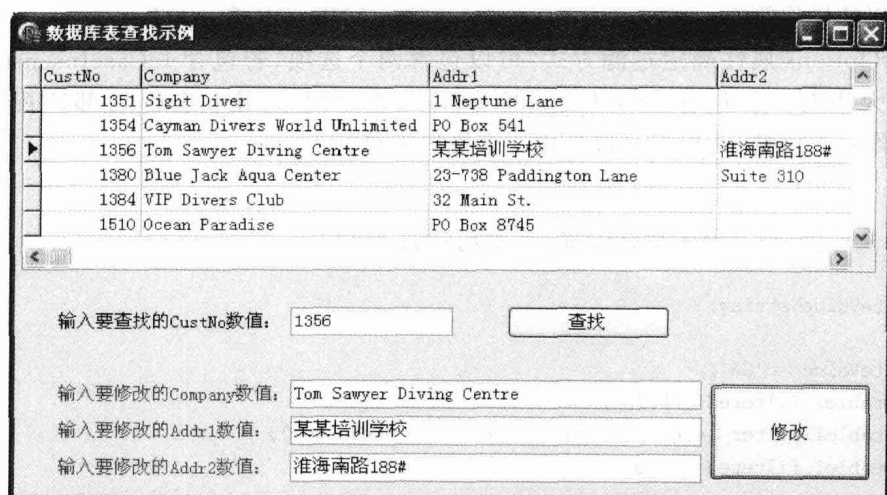


图 9-16 数据库表查找示例运行界面

### 9.2.6 TADOTable 数据集组件的记录移动

每一个激活的数据集都有一个记录指针指向数据集中的当前记录。通过移动记录指针,可以改变当前记录。在数据集中可能使用 First、Last、Next、Prior、MoveBy 方法移动记录指针。此外,TDBGrid 和 TDBNavigator 封装了这些方法,在运行时,用户可以通过按键或单击按钮执行这些操作。无论采用何种方法移动记录指针,都会触发两个事件:BeforeScroll 事件(离开当前记录前)和 AfterScroll 事件(到达新记录后)。针对这两个事件可以编写相应的处理代码。

在窗体中,TDBEdit、TDBText 和 TEBMemo 等组件显示的数据就是当前记录的字段值。如果数据集是可编辑的,则当前记录中的字段值可以被添加、删除和修改。

数据集组件还提供了两个属性: Bof 和 Eof,用于测试记录指针是否指向两个特殊位置。Bof 为 True 时,记录指针指向记录集开始处; Eof 为 True 时,记录指针指向记录集结束处。如果两者同时为 True,则表明数据集是空的。

例如:

- (1) 在数据集组件 ADOTable1 中移动记录指针到第一条记录: ADOTable1.First;
- (2) 移动记录指针到最后一记录: ADOTable1.Last;
- (3) 向前移一条记录: ADOTable1.Prior;
- (4) 向后移一条记录: ADOTable1.Next;
- (5) 向前移 5 条记录: ADOTable1.MoveBy(-5);
- (6) 判断数据集为空:

```
if ADOTable1.Bof and ADOTable1.eof then  
    ShowMessage('该数据表为空表.');
```

### 9.2.7 TADOTable 数据集组件的数据过滤

显示查询数据集时,往往需要对数据进行筛选。可以使用数据集组件的 Filter、Filtered

和 FilterOptions 属性实现该功能,其行为类似于 SQL 语句中的 WHERE 子句。设置 Filter 时可以使用的操作符有 <、>、<=、>=、=、<>、AND、OR 和 NOT。

FilterOptions 属性确定过滤方式,可以包含两个选项,若包含 foCaseInsensitive,表示忽略字母大小写;若包含 foNoPartialCompare,表示可以用来进行局部匹配。把星号 (“\*”) 当作一个字符,否则星号被当作一个掩码字符。

例如:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    StateValue:string;
begin
    StateValue := 'CA';
    ADOtable1.filtered := False;
    ADOtable1.Filter := Format('State = "%s"', [StateValue]);
    ADOtable1.filtered := True;
end;
```

本程序中的 ADOtable1.Filter := Format('State = "%s"', [StateValue]); 也可以改写为 ADOtable1.Filter := 'State = ' + QuotedStr(StateValue);。

QuotedStr() 的作用是将字符串转换为带引号的字符串,也可以使用双单引号。比如:

```
ADOtable1.Filter := 'State = "CA" OR State = "HI"'
```

另外,当 Filtered 属性由 False 变为 True 时触发该事件。在 OnFilterRecord 事件处理程序中可以设置筛选条件。例如:

```
procedure TForm1.ADOtable1FilterRecord(DataSet: TDataSet;
    var Accept: Boolean);
begin
    Accept := DataSet['State'] = 'CA';
end;
```

**注意:** 如果用户修改了带有过滤器数据集的数据,修改的结果如果不满足过滤器的条件,则修改的数据就自动从当前的数据集中消失。

## 9.3 数据源组件和数据控制组件

组件面板上的 Data Access 页包含了数据源组件 TDataSource,如图 9-17 所示,它是数据库应用程序设计中非常重要的组件,为连接到各种各样的数据源提供了方便。

Data Controls 的组件页包含了所有的数据控制组件,如图 9-18 所示,这些组件用于编辑和显示数据库表中的数据,用于建立数据操作的图形用户接口,其中 TDBGrid、TDBNavigator、TDBText、TDBEdit 和 TDBMemo 等组件是数据控制中比较常用的。通常也称其为数据敏感组件,因为它们能够随着数据的变化而相应地变化。它们的主要功能是与数据源组件相配合,提供给用户一个对数据进行浏览、编辑等操作的界面。

这些组件主要用于设计用户界面,对数据库中的数据进行浏览、编辑、插入和删除等操作。数据控制组件其实是在 Standard 页上标准组件的基础上,相应地增加了数据浏览功能,

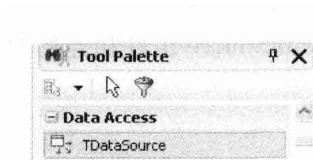


图 9-17 数据源组件 TDataSource

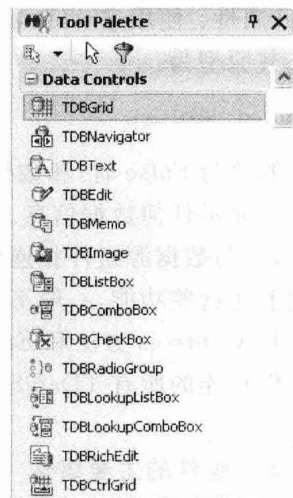


图 9-18 数据控制组件位于 Data Controls 的组件页

使得它们能够显示和编辑数据库中的数据。所有数据控制组件都有一个 DataSource 属性, 该属性指定与其相关的数据源。数据控制组件既能够把数据库中的数据 displays 到窗体中, 又可以将经过修改的数据写回到数据库中。

### 9.3.1 TDataSource 组件

TDataSource 组件用于连接数据控制组件和数据集组件。每一个数据控制组件都有一个 DataSource 属性, 用于指定连接的数据源组件, 但是没有直接连接到数据集, 而 DataSource 将连接追踪到数据集。

#### 1. TDataSource 组件的主要属性

(1) DataSet 属性: 指定 TDataSource 组件所连接的数据集的名字, 它可以是 TADOTable 组件的名字, 也可以是 TADOQuery 组件的名字, 甚至还可以是其他窗体内的数据集。其原型为:

```
property DataSet: TDataSet;
```

可以在设计时指定该属性的值, 也可以在程序中指定。例如:

```
DataSource1.DataSet := ADOTable2;
```

指定数据源 DataSource1 连接到数据集 ADOTable2。

(2) AutoEdit 属性: 该属性设置与 TDataSource 组件相连的数据集是否允许自动编辑。默认值为 True。其原型为:

```
property AutoEdit: Boolean;
```

当 AutoEdit 的值为 True 时, 与 TDataSource 相连的数据集组件自动地被设置成编辑状态, 当用户在与 TDataSource 组件相连的数据控制组件中输入新的值时, 数据集组件中的记录也随之改变。如果 AutoEdit 的值为 False, 用户想要修改数据集中的记录, 必须调用数据集组件的 Edit 方法, 将其置为编辑状态之后才能够进行。

(3) Enabled 属性: 使用 Enable 属性可以暂时性地关闭数据源组件和与之相连的数据集组件的连接。其原型为:

```
property Enabled: Boolean;
```

当 Enabled 的值为 False 时,和数据集组件的连接被关闭,且所有与数据源组件相连的数据控制组件中不显示任何数据信息。当 Enabled 的值变为 True 时,数据源组件和数据集组件的连接恢复,且与数据源组件相连的数据控制组件恢复显示数据。

不过要实现上述这些功能,一般不使用 TDataSource 组件的 Enabled 属性,而是调用数据集组件的 DisableControls 方法和 EnableControls 方法,因为调用这两个方法可以方便地控制与数据集组件相连的所有 TDataSource 组件以及与 TDataSource 组件相连的数据控制组件。

## 2. TDataSource 组件的主要事件

(1) OnDataChange 事件: 该事件一般用于保持应用中多个组件之间的同步。当与 TDataSource 相连的数据集中的记录修改后,其指针的位置发生改变时,该事件就被触发。

例如:

```
procedure TForm1.DataSource1DataChange(Sender: TObject; Field: TField);
begin
  if (Field <> nil) then
    ShowMessage( Field.DisplayName + ' changed to ' + Field.AsString);
end;
```

当一个记录被修改了,并且用户从一个组件转移到另外一个组件的时候将触发 OnDataChange 事件,执行以上过程,弹出对话框显示字段改变发生的情况。

(2) OnUpdateData 事件: 当数据集组件中当前记录将要被修改时,触发该事件。在应用程序中使用非数据浏览组件时,若要它与数据集保持同步,常使用该事件进行相关的处理。

例如在程序调用 post 方法之后,但在修改后的数据真正被写回磁盘中的数据库文件之前触发该事件。

(3) OnStateChange 事件: 当数据集的状态发生改变时,便触发该事件。因为数据集组件的 State 属性标明了数据集组件当前所处的状态,当数据集的状态发生变化时,使用该事件进行有关的处理是很有用的。

可以用下面例子中的程序代码将数据集组件当前的状态显示在一个 TLabel 组件上:

```
procedure TForm1.DataSource1StateChange(Sender: TObject);
var
  S: String;
begin
  Case ADOTable1.State of
    dsInactive: S := 'Inactive';
    dsBrowse: S := 'Browse';
    dsEdit: S := 'Edit';
    dsInsert: S := 'SetKey';
    dsSetKey: S := 'SetKey';
  end;
```

```
Label1.Caption := S;
end;
```

类似地,也可以通过检测数据源组件的状态来控制有关的按钮和菜单项是否有效。

### 9.3.2 TDBGrid 组件

在前面的例子里已经用到过 TDBGrid 组件。该组件是用来以表格形式显示和编辑数据集中记录信息的重要组件,表格中的行对应于数据集中的记录,表格中的列对应于数据集中的字段。TDBGrid 组件通过数据源组件与数据表组件连接。

在 TDBGrid 组件对象中可以直接对表格中的数据作修改,当记录指针移动后,数据自动被保存到数据库。可以使用→、←、↑、↓移动记录指针,使用滚动条浏览数据,还可以使用快捷键 Insert 在光标处插入新记录,使用 Ctrl+Delete 组合键删除当前记录,使用 Esc 键取消之前的操作。

在 Delphi 2007 设计界面中,可以通过改变 Columns 属性设置 TDBGrid 组件显示数据集中的哪些字段,还可以通过改变 Options 属性设置数据表格的属性(包括表格的可编辑、可删除和可添加等属性)。

#### 1. TDBGrid 组件的重要属性

(1) DataSource 属性:该属性设置为数据源组件名称,该数据源被指定为 TDBGrid 组件中显示数据的来源,它是 TDBGrid 组件中的最重要属性。

(2) Columns 属性:用来读取和设置表格中列的特征,所有 Column(列)对象都存储于 Columns 属性中。一个 Column 对象代表 DBGrid 组件中的一列。默认情况下,DBGrid 组件会在表格中显示数据集中的所有字段,表格每一列的列名自动采用字段的名字。如果希望自己定义表格的列,在设计时可以使用 DBGrid 组件列编辑器手动设置 Columns 集合。

设计时在 DBGrid 上右击,在弹出的快捷菜单中选择 Columns Editor 命令,激活列编辑器(如图 9-19 中间图所示)。双击 DBGrid 也可以打开列编辑器。还有一种方式就是在选中 TDBGrid 组件时单击 Object Inspector 的 Columns 属性。

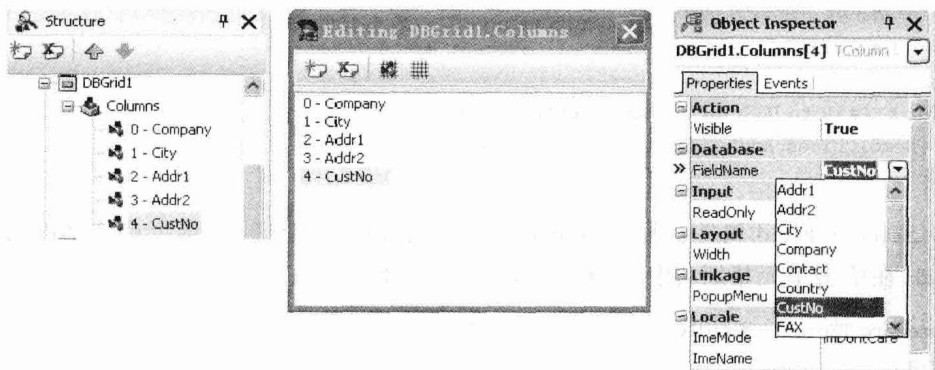


图 9-19 列 Column 对象树、列编辑器及与 Columns 字段的关系

如图 9-19 所示,可以在列编辑器中选择列对象,然后在 Object Inspector 中对它进行设置。表 9-7 中列出了 Column 对象的部分属性。只有正确设置了 DBGrid 数据源属性之



后(即数据库的连接已打开),列编辑器和 Object Inspector 中才能获得数据集中的字段名。

表 9-7 Column 对象属性及说明

列 属 性	说 明
ButtonStyle	TColumnButtonStyle 的对象,定义了显示在表格单元中的编辑按钮。其可能的选项是 cbsAuto、cbsEllipsis 和 cbsNone
Visible	控制列的可视性
Width	控制列宽度
Color	字段单元的背景颜色
DropDownRows	当 ButtonStyle 特性为 cbsAuto 时,DropDownRows 为显示的行数,该列有一个查找字段或者已经定义了的 PickList
Expanded	Expanded 如果为 True,那么 ObjectField 被扩展,显示对象字段的附加列
FieldName	数据集的字段名,列的数据从该字段中获取值
Font	设置列中每一个单元的单元字体
ImeMode	列的输入方法编辑器,用于转换亚洲字符
ImeName	所使用的输入方法编辑器的名称
PickList	静态的选择列表(TStrings 对象),该列表作为列字段的可能选项。例如一个 Boolean 字段可以在选择列表中使用 True 和 False 值
PopupMenu	指定 TPopupMenu 对象
ReadOnly	确定列中的数据是否可编辑
Title	一个嵌套的 TColumnTitle 对象,定义了本列的固定列单元的显示属性

(3) Fields 和 FieldCount 属性: Fields 属性表示当前记录的所有字段的集合,通过 Fields 属性可以直接访问 TDBGrid 组件表格中当前记录的某一列的值。而 FieldCount 属性则说明当前记录一共有多少列。这两个属性往往配合起来使用。

例如,显示表格当前行所有列的值到备注框中,可以通过循环处理,主要代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  k : Integer;
begin
  for k := 0 to DBGrid1.Columns.Count - 1 do
    Memo1.Lines.Add( DBGrid1.Fields[k].AsString);
end;
```

(4) SelectedField 属性: 通过 SelectedField 属性可以得到当前选定单元格内的值。

例如,在单击单元格时,用对话框显示单元格内容:

```
procedure TForm1.DBGrid1CellClick(Column: TColumn);
begin
  if DBGrid1.SelectedField <> nil then
    ShowMessage(DBGrid1.SelectedField.AsString);
end;
```

(5) Option 属性: 该属性包含了描述 TDBGrid 组件的显示和操作属性。Option 的主要选项参数名称与其取值含义如表 9-8 所示。

表 9-8 Options 属性的各个选项名称以及取值含义

选项名称	取值含义	
	True	False
dgEditing	确保用户能够在表格中编辑、插入和删除数据集中的记录,是默认值	在表格中不能编辑、插入和删除数据集中的记录
dgAlwaysShowEditor	当用户选中记录中的一个字段时,自动地使该字段处于编辑状态	当一个字段被选中,它不能自动地变成编辑状态,是默认值
dgTitles	在表格的第一行中显示字段名或字段标题,是默认值	在表格中不显示字段名或字段对应的标题
dgIndicator	在表格的最左边用一个黑箭头标注当前记录指针所在的位置,在插入状态时,箭头变成星状;在编辑状时,箭头变成 I 状,是默认值	在表格中不标识当前记录指针的位置
dgColumnResize	通过拖拉表格的垂直分隔线可以改变表格中各列的宽度,在具体操作时要拖拉各列中显示字段标题区域中的垂直分隔线,是默认值	表格中各列的宽度不能改变
dgColLines	在表格中显示各列之间的垂直分隔线,是默认值	在表格中不显示垂直分隔线
dgRowLines	在表格中显示各行之间的水平分隔线,是默认值	在表格中不显示水平分隔线
DgTabs	可以在记录的各字段之间移动输入焦点(即选择提示棒),是默认值	不能在记录的各字段之间移动输入焦点,在表格中按 Tab 键时,直接跳出表格
dgRowSelect	选择提示棒覆盖整条记录中的全部字段	选择提示棒一次只覆盖记录中的一个字段,是默认值
dgAlwaysShowSelection	在表格始终显示选择提示棒,即 Selection 使其组件获得焦点时,也是如此,是默认值	只在当表格获得焦点时,才显示选择提示棒
dgConfirmDelete	当在表格中删除记录时,弹出确认信息,是默认值	在表格中删除记录时不弹出确认信息

(6) DragMode 属性: 该属性描述表格列的拖拉属性,有两个可选的属性值 dmManual 和 dmAutomatic。

当它的值被设置为 dmManual 时,在应用程序运行过程中,用户可以用鼠标拖放表格中的各列,改变各列在表格中的显示顺序和位置。当用鼠标拖放表格中的一列,改变它在表格中的位置时,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集中的位置。

当该属性的值被设置成 dmAutomatic 时,用户不能用鼠标拖放表格中的各列而改变它在表格中的位置。

(7) DefaultDrawing 属性: 该属性主要用于控制表格中各表格单元的绘制方式。该属性的值为 True 时,Delphi 使用表格本身默认的方法绘制表格中各表格单元,并填充各表格单元中的内容。如果 DefaultDrawing 属性被设置为 False 时,用户必须自己为 TDBGrid 组件的 OnDrawDataCell 事件编写相应的程序用于绘制各表格单元和其中的数据。该属性默认值为 True。

2. TDBGrid 组件的主要事件

该组件的主要事件如表 9-9 所示。

表 9-9 TDBGrid 组件的主要事件

事件名称	事件说明
OnCellClick	当用户在单元格中单击鼠标左键时,触发该事件
OnColEnter	当用户进入表格各列时,触发该事件
OnColExit	当用户离开表格各列时,触发该事件
OnDblClick	当用户在表格中双击鼠标左键时,触发该事件
OnDragDrop	当用户在表格中用鼠标进行拖放操作时,触发该事件
OnDragOver	当用户在表格中用鼠标拖动表格时,触发该事件
OnDrawDataCell	用于定制绘制表格中各表格单元,当向表格中填充数据时触发该事件
OnEndDrag	当用户停止拖动表格时,触发该事件
OnEnter	当表格获得焦点时,触发该事件
OnExit	当表格失去焦点时,触发该事件
OnKeyDown	当用户在表格中按下任何键或组合键时,触发该事件
OnKeyPress	当用户在表格中按下任何一个数字键或字母键时,触发该事件
OnKeyUp	当用户在表格中释放任何被按下的键时,触发该事件

9.3.3 TDBNavigator 组件

TDBNavigator 组件主要用于在数据集中进行记录导航和为用户操纵数据集中的记录提供了一组简单明了的控制按钮。用户单击其中的按钮可以向前或向后移动记录指针、插入记录、修改现存记录、提交对记录的修改、取消修改、删除记录以及刷新记录的显示等,如图 9-20 所示。

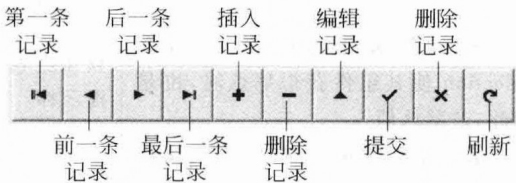


图 9-20 TDBNavigator 组件的各个按钮

TDBNavigator 组件的绝大多数功能都可以根据其按钮的图标很容易地识别出来,而且 TDBNavigator 部件本身能感知到很多事情,如当前指针是否在数据库表的开头或尾部。如果用户正在查看数据库表中的最后一个记录,Next 和 Last 按钮将会变灰成为非活动状态。同样,如果用户当前正在浏览数据库表中的第一条记录,TDBNavigator 上的 First 和 Previous 按钮会变灰而成为非活动状态。有关各个按钮作用的更详细说明如表 9-10 所示。

1. TDBNavigator 组件的主要属性

- (1) DataSource 属性: 该属性指定 TDBNavigator 组件所连接的数据源(DataSource) 组件名称,它是该组件中最重要的属性之一。
- (2) VisibleButtons 属性: 该属性主要是控制组件中按钮的可见性。TDBNavigator 组件中有多个功能按钮,但并不是所有的按钮对每一个数据库应用程序都是需要的。可以通过设置 TDBNavigator 组件的 VisibleButtons 属性来确定要在 TDBNavigator 中显示哪些按钮和不显示哪些按钮。

表 9-10 TDBNavigator 组件的按钮名称与功能

按钮名称	按钮功能
First	将当前记录指针移到数据集中第一条记录处
Prior	将记录指针移到当前记录的前一条记录处
Next	将记录指针移到当前记录的后一条记录处
Last	将当前记录指针移到数据集中最后一条记录处
Insert	调用数据集组件的 Insert 方法,在当前记录的前面插入一条新记录,并将数据集组件置为插入状态
Delete	删除当前记录,如果 TDBNavigator 组件的 ConfirmDelete 属性设置为 true 时,会弹出删除确认对话框
Edit	将数据集组件置为编辑状态,以便用户修改当前的记录
Post	提交对当前记录的修改
Cancel	取消对当前记录的修改,并将数据集组件置为浏览状态
Refresh	清除数据浏览组件的显示缓冲区,并用与其相连的数据集组件中的记录刷新显示缓冲区

例如,如果不允许用户修改表中的记录,就不需要 Add、Delete、Post、Cancel 或 Refresh 按钮,可以设置该属性让这些按钮的 VisibleButtons 属性为 False,这样在 TDBNavigator 组件中将不会出现这些按钮。

VisibleButtons 属性定义原型如下:

```
type TNavigateBtn = (nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost, nbCancel, nbRefresh);
```

VisibleButtons 属性所指定的集合由 TNavigateBtn 类型的数据组成,这种类型的数据分别表示了每一个按钮。

```
type TButtonSet = set of TNavigateBtn;  
property VisibleButtons: TButtonSet;
```

设置 VisibleButtons 属性可以通过 Object Inspector 在设计时设置,即把对应的按钮设置为 True 即可。也可以在程序运行时设置,这时需要使用集合数据的运算符。下面的代码会使 Delete 按钮不可见:

```
DBNavigator1.VisibleButtons := DBNavigator1.VisibleButtons - [nbDelete];
```

(3) ConfirmDelete 属性: TDBNavigator 组件的 ConfirmDelete 属性和 Delete 按钮配合使用,用户删除数据集中的记录是非常有用的。如果设置 ConfirmDelete 属性为 True(默认设置),当用户单击 Delete 按钮想要删除当前记录时,Delphi 会弹出一个确认框,要用户确认是否真的想删除当前记录。这样,在用户进行删除记录的操作时会更安全一些。如果用户不希望在单击 Delete 按钮时出现确认框,只要把 ConfirmDelete 设置为 False 就可以了。

(4) ShowHint 属性: 用于控制是否显示 TDBNavigator 组件中各按钮的动态提示信息。当设置它的值为 True 时,当用户将鼠标光标停留在 TDBNavigator 组件中某一个控制按钮上一段时间后,系统便会自动显示有关该控制按钮的提示信息。默认情况下该属性的

值为 False,表明不显示。

(5) Hints 属性: 在默认情况下,TDNavigator 组件中的各控制按钮都有相应的动态提示信息,如 First、Prior、Next 和 Last 等,用户可以根据自己的需要,通过设置 Hints 属性为各控制按钮设置其他的动态提示信息,用户自己设置的动态提示信息会覆盖原来的提示信息。

## 2. TDNavigator 组件的事件

(1) BeforeAction 事件: BeforeAction 事件发生在鼠标单击某个按钮之后,但是却在按钮执行对应的操作之前。通过参数 Button 可以知道单击了哪一个按钮。

(2) OnClick 事件: OnClick 事件在用户编程时经常使用。该事件发生在鼠标单击某个按钮之后,但与 BeforeAction 事件不同,是在该按钮执行对应的操作之后。通过参数 Button 可以知道单击了哪一个按钮。

### 例 9.3 TDGrid 组件与 TDNavigator 组件综合应用示例。

设计构想: 编写一个程序演示对 TDGrid 各属性的设置,使显示的表格更加漂亮美观。使用 Delphi 2007 系统自带的数据库 dbdemos.mdb,在 TDGrid 中显示表 Country 中的记录。同时本程序可以使用 TDNavigator 组件对表格进行导航控制。

#### (1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“TDGrid 组件与 TDNavigator 组件综合应用示例”。

#### (2) 用户界面设计。

① 在窗体上添加一个用于建立数据库连接的 ADOConnection1 组件(位于组件面板 dbGo 页),通过双击该组件在向导对话框设置其连接到目标数据库 dbdemos.mdb。并将其属性 LoginPrompt 设置为 False,Connected 属性设置成 True。

② 在窗体上放置用于映射数据库表的 ADOTable1 组件(位于组件面板 dbGo 页),设置其 Connection 属性为 ADOConnection1,选择其 TableName 属性为 Country,将其 Active 设置为 True。

③ 在窗体上放置一个数据源 DataSource1 组件(位于组件面板 Data Access 页),设置其 DataSet 属性为 ADOTable1。

④ 在窗体上放置一个控制读取数据的表格 DBGrid1 (位于组件面板 Data Controls 页),设置其 DataSource 属性为 DataSource1,DBGrid1 的 ReadOnly 属性为 True。双击 DBGrid 也可以打开列编辑器,在列编辑器中添加 4 个列对象,然后在 Object Inspector 中对它的标题属性进行设置,如图 9-21 所示。然后设置其 Align 属性为 alClient。

⑤ 在窗体上添加一个 TDNavigator 组件。在 Object Inspector 中展开 VisibleButtons 属性(单击该属性),将其子属性 nbInsert、nbDelete、nbEdit、nbPost、nbCancel 和 nbRefresh 设置为 false,使它们隐藏起来。然后设置其 Align 属性为 alTop。最后设置其 DataSource 属性为 DataSource1。

窗体的设计界面如图 9-22 所示。

#### (3) 代码编写。

在 DBGrid1 的 onDrawColumnCell 事件中输入以下代码:

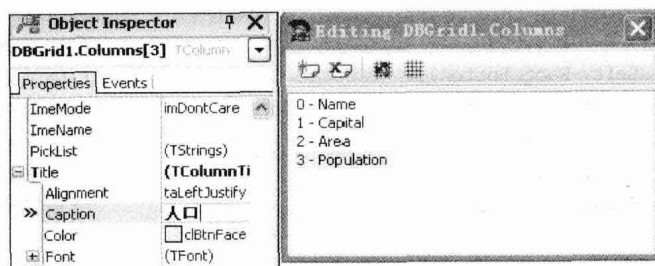


图 9-21 列编辑器与对象查看器设置标题

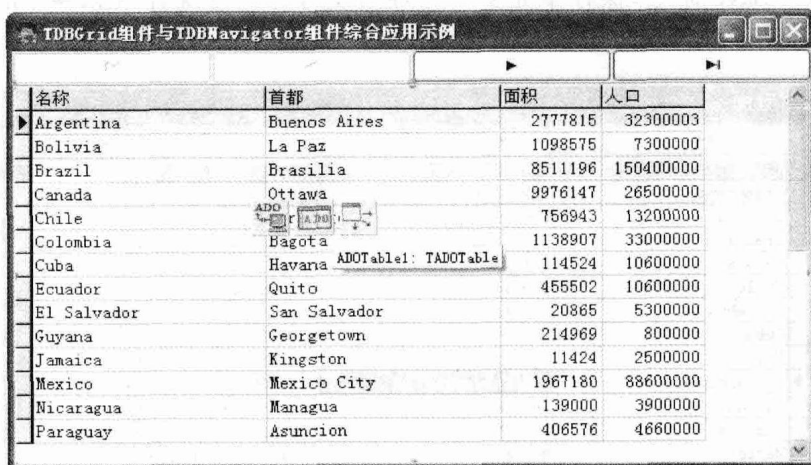


图 9-22 TDBGrid 组件与 TDBNavigator 组件综合应用示例设计界面

```

procedure TForm1.DBGrid1DrawColumnCell(Sender: TObject; const Rect: TRect;
  DataCol: Integer; Column: TColumn; State: TGridDrawState);
var
  i : integer;
begin
  if gdSelected in State then Exit;
  //定义表头的字体和背景颜色
  for i := 0 to (Sender as TDBGrid).Columns.Count - 1 do
  begin
    (Sender as TDBGrid).Columns[i].Title.Font.Name := '宋体';           //字体
    (Sender as TDBGrid).Columns[i].Title.Font.Size := 9;               //字体大小
    (Sender as TDBGrid).Columns[i].Title.Font.Color := $ 000000ff;      //字体颜色(红色)
    (Sender as TDBGrid).Columns[i].Title.Color := $ 0000ff00;           //背景色(绿色)
  end;
  //将奇偶行设置不同背景色
  if ADOTable1.RecNo mod 2 = 0 then
    (Sender as TDBGrid).Canvas.Brush.Color := clInfoBk
  else
    (Sender as TDBGrid).Canvas.Brush.Color := RGB(191, 255, 223);
  //定义表格线的颜色
  DBGrid1.DefaultDrawColumnCell(Rect, DataCol, Column, State);
  with (Sender as TDBGrid).Canvas do
    //画 cell 的边框

```

```
begin
    Pen.Color := $00ff0000;           //定义画笔颜色(蓝色)
    MoveTo(Rect.Left, Rect.Bottom);    //画笔定位
    LineTo(Rect.Right, Rect.Bottom);   //画蓝色的横线
    Pen.Color := $0000ff00;           //定义画笔颜色(绿色)
    MoveTo(Rect.Right, Rect.Top);      //画笔定位
    LineTo(Rect.Right, Rect.Bottom);   //画绿色的竖线
end;
end;
```

(4) 保存并运行。

选择 File→Save 命令,保存单元文件与项目文件。通过选择 Run→Run 命令运行程序,实例效果如图 9-23 所示。

名称	首都	面积	人口
Argentina	Buenos Aires	2777815	32300003
Bolivia	La Paz	1098575	7300000
Brazil	Brasilia	8511196	150400000
Canada	Ottawa	9976147	26500000
Chile	Santiago	756943	13200000
Colombia	Bagota	1138907	33000000
Cuba	Havana	114524	10600000
Ecuador	Quito	455502	10600000
El Salvador	San Salvador	20865	5300000
Guyana	Georgetown	214969	800000
Jamaica	Kingston	11424	2500000
Mexico	Mexico City	1967180	86600000
Nicaragua	Managua	139000	3900000
Paraguay	Asuncion	406576	4660000

图 9-23 数据库表查找示例运行界面

在 Delphi 语言的数据库编程中, TDBGGrid 是显示数据的主要手段之一。但是, TDBGGrid 默认的外观显得单调和缺乏创意。其实,可以使用本例的方法在程序中通过编程来达到美化 TDBGGrid 外观的目的。通过编程可以改变 TDBGGrid 的表头、表格、表格线的前景色和背景色,以及相关字体的大小和风格。

### 9.3.4 TDBText 组件与 TDBEdit 组件

TDBText 组件是一个只读的数据浏览组件,它和 TLabel 组件类似。TDBText 组件用于显示数据集中记录的指定字段的值。因为 TDBText 组件显示的是数据集中当前记录指定的字段的值,所以它显示的内容也是动态的,随着记录指针的移动而变化。但该组件不能用于编辑数据库表中的数据。

TDBEdit 组件用于显示、编辑数据表中当前记录各个字段值的数据控制组件,常用来对应数据集中的—个字段。

通过设置 TDBText 组件和 TDBEdit 组件的 DataSource 属性和 DataField 属性便可以为它们指定与数据集关联的字段。

TDBText 组件与 TDBEdit 组件的主要属性:

(1) DataSource 属性: 和前面的数据控制组件一样, DataSource 属性也是 TDBText 组件的重要属性, 通过它才能连接和读取数据库。

(2) DataField 属性: DataField 属性指定了组件应该显示当前记录中哪一个字段的值。在设计时可以单击 Object Inspector 中的 DataField 属性, 如果 DataSource 属性设置正确, 会出现一个显示所有字段的下拉框, 选择需要显示的字段即可。在运行时, 也可以直接对 DataField 属性赋值。

例如要让 DBText1 显示数据集中 area 字段的内容, 语句如下:

```
DBText1.DataField := 'area';
```

此外, TDBEdit 组件的 ReadOnly 属性用于将 TDBEdit 设置为只读。

#### 例 9.4 TDBText 组件与 TDBEdit 组件数据表单条记录浏览编辑示例。

设计构想: 本程序需要通过 TADOConnection 连接到目标数据库 dbdemos. mdb; 用两个 TDBText 组件显示 CustNo 字段与 Company 字段内容, 用若干个 TDBEdit 组件显示或者编辑 Addr1、City 等字段内容。

##### (1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序, 系统会自动创建一个空白的窗体。调整窗体的大小, 并设置窗体的 Caption 值为“TDBText 组件与 TDBEdit 组件数据表单条记录浏览编辑示例”。

##### (2) 用户界面设计。

① 在窗体上添加一个用于建立数据库连接的 ADOConnection1 组件(位于组件面板 dbGo 页), 通过双击该组件在向导对话框设置其连接到目标数据库 dbdemos. mdb。并将其 LoginPrompt 属性设置为 False, Connected 属性设置成 True。

② 在窗体上放置用于映射数据库表的 ADOTable1 组件(位于组件面板 dbGo 页), 设置其 Connection 属性为 ADOConnection1, 选择其 TableName 属性为 Customer, 将其 Active 设置为 True。

③ 在窗体上放置一个数据源 DataSource1 组件(位于组件面板 Data Access 页), 设置其 DataSet 属性为 ADOTable1。

④ 在窗体上添加一个 TDBNavigator 组件, 用于表格导航。设置其 DataSource 属性为 DataSource1。

⑤ 在窗体上依次放置 13 个 TLabel 组件作为字段标识, 并设置它们的 Caption 值为相应字段的名称。

⑥ 在窗体上放置两个 TDBText 组件, 设置其 DataSource 属性为 DataSource1。依次设置它们的 DataField 属性为 CustNo 字段与 Company 字段。

⑦ 在窗体上放置 11 个 TDBEdit 组件, 设置其 DataSource 属性为 DataSource1。依次设置它们的 DataField 属性为跟其标签组件一致。

设置好所有组件的辅助属性: 颜色、位置与大小合适后, 窗体界面如图 9-24 所示。

##### (3) 保存并运行程序。

本程序不需要用户另外编写代码就可以正常运行。



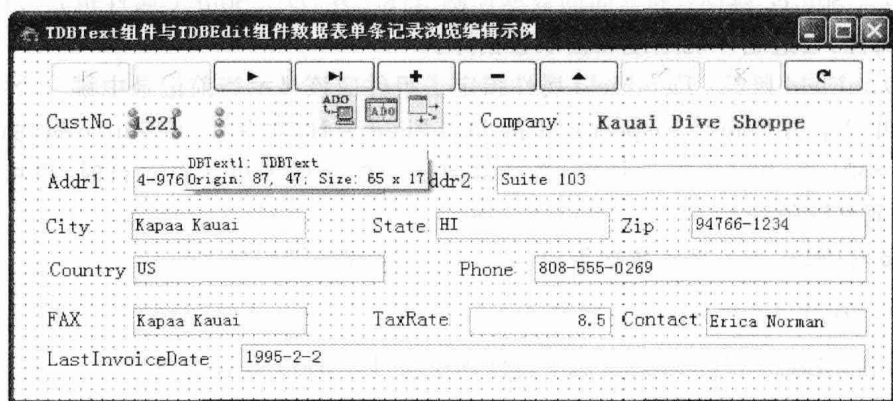


图 9-24 TDBText 组件与 TDBEdit 组件数据表单条记录浏览编辑示例设计界面

### 9.3.5 TDBMemo 组件与 TDBComboBox 组件

TDBMemo 组件主要用于显示和编辑数据集中的大二进制 (BLOB) 类型的字段值。TDBMemo 组件能够显示多行文本, 允许用户在其中输入和修改多行文本信息, 它是 Delphi 中用来显示和编辑数据集中大二进制类型的文本字段的唯一数据控制组件。

TDBComboBox 组件中包含了 TDBEdit 组件的全部功能, 它们具有相似性, 不同的是在运行过程中 TDBComboBox 组件同时有一个下拉式列表框, 在下拉式列表框中有一组可供选择的项供用户选择, 这些可选项是在设计阶段由程序设计人员提供给 TDBComboBox 组件的 Items 属性的。TDBComboBox 组件一定要对应数据集中的一个字段。

#### 1. TDBMemo 组件的主要属性

(1) ReadOnly 属性: 当 ReadOnly 属性为 True 时, 能在 TDBMemo 组件中显示 BLOB 型文本信息, 如 Access 和 dBASE 数据集中的备注型字段; 当为 False 时, 用户在 TDBMemo 组件中不仅可以显示 BLOB 文本信息, 而且还允许用户编辑修改其中的文本信息。

(2) MaxLength 属性: 该属性是整数型的, 其值用于限制用户向 TDBMemo 组件中输入字符的个数。若设置该值为 0 时, 表示输入字符的个数没有限制。

(3) ScrollBar 属性: 表示 TDBMemo 组件是否显示滚动条。

(4) WordWrap 属性: 表示在 TDBMemo 组件中输入文本信息时, 输入到右边界时是否自动换行。

(5) Alignment 属性: 表示文本信息在 TDBMemo 组件中的对齐方式, 它有 taLeftJustify、taCenter 和 taRightJustify 三个值。其含义分别是左对齐、居中和右对齐。

(6) AutoDisplay 属性: 因为 TDBMemo 组件中包含着大量的文本信息, 应用程序在运行过程中要显示其中的信息需要花费大量时间, 特别是当用户移动记录指针时, 都要更新 TDBMemo 组件中显示的信息, 花费的时间更是多得多, 所以 Delphi 为 TDBMemo 组件设定了 AutoDisplay 属性, 用来控制是否自动显示表中的备注型字段。当 AutoDisplay 设置为 False 时, 在 TDBMemo 组件中只显示其对应表中的字段名而不显示字段中的文本信息, 用户如果想浏览字段中的文本信息, 用鼠标左键双击 TDBMemo 组件的内部即可; 当设置

AutoDisplay 属性为 True 时,在 TDBMemo 组件中会自动地显示其对应数据集中的字段值。DBMemo 组件中显示和编辑文本信息的最大字节数为 32K,在使用中不能超过这一限制。

## 2. TDBComboBox 组件的属性

(1) Items 属性: Items 属性是 TDBComboBox 组件的一个重要属性,该属性中包含着 TDBComboBox 组件在运行过程中下拉式列表框中的可选项,Items 中的内容可以在设计阶段指定。

如果一个 TDBComboBox 组件对应着数据集中的一个字段,那么当用户要编辑修改该字段中的值时,可以打开下拉式列表框,从中选择一个可选项作为字段值,也可以自己在 TDBComboBox 组件中输入一个其他的字段值。

(2) Style 属性: 控制 TDBComboBox 组件列表框的显示格式,它的取值为:

- csDropDown: 默认情况下为此值,显示一个下拉式列表框和一个文本框,下拉式列表框中的可选项都是字符串且各选择项占据的高度一样。
- csSimple: 只显示一个列表框,列表框中的可选项都是字符串,且各选项占据一样的高度。
- csDropDownList: 显示一个下拉式列表框和一个文本框,但用户不能向文本框中输入一个在列表框中没有的值。

在列表框中不仅有字符串选项,而且还允许有其他类型的选项,如位图图像等,这方面的详细信息请参看联机帮助。

(3) DropDownCount 属性: 允许列表框中显示可选项的最大数目,当可选项数目大于该属性值时,用户可以用滚动条查看全部的可选项;当可选项数目小于该属性值时,列表框会自动调整其大小以足够显示全部可选项。

(4) ItemHeight 属性: 当 TDBComboBox 组件的 Style 属性被设置为 CSOwnerDrawFixed 时,用此属性来设置列表框中每个可选项占据的高度。

(5) Sorted 属性: 布尔型属性,它决定列表框中的可选项是否按字母的排列顺序排序。

## 9.3.6 TDBListBox 组件与 TDBImage 组件

TDBListBox 组件的基本功能与 TDBComboBox 组件基本上是一样的,它们的不同之处在于 TDBListBox 组件没有下拉式列表框而是一个列表框,在列表框中显示一组供用户选择的可选项,在运行过程中,用户单击其中的可选项可以为 TDBListBox 组件对应的字段赋一个字段值,但用户不能自己从键盘上输入一个列表框中不存在的字段值。如果在应用程序中,TDBListBox 对应数据集中一个具体的字段,那么当在数据集中移动记录指针时,当前记录中对应 TDBListBox 组件的字段的值在 TDBListBox 组件的列表框中将以高亮度显示;如果当前记录的该字段值不在列表框中,那么列表框中的可选项没有一项是高亮度地显示的。

TDBImage 组件与 TDBMemo 组件具有很多相似的属性,它是用来显示和编辑数据集中 BLOB 类型的位图图像字段的。默认情况下,在 TDBImage 组件中允许用户对位图图像进行编辑,如将图像复制到剪贴板上或从剪贴板上粘贴到 TDBImage 组件中,也可以在程序中调用 CutToClipboard、CopyToClipboard 和 PasteFromClipboard 方法来实现剪切、复

制、粘贴功能,进行操作时要保证 TDBImage 的 ReadOnly 属性值为 False。TDBImage 组件也具有一个 AutoDisplay 属性,该属性的控制和作用与 TDBMemo 组件的 AutoDisplay 属性是一样的。

**例 9.5** 利用数据源组件、TDBText 组件、TDBMemo 组件与 TDBImage 组件等设计具有单条记录与表格显示功能的编辑浏览程序界面。

设计分析:本程序要求使用的组件有备注和图片,那么必须准备这样一个数据库,本程序运行的条件是:机器必须安装了 BDE 数据库驱动,使用一个 Paradox 数据库(名为 DBDEMOS)中的 BIOLIFE.db 数据表来制作。

(1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“具有备注、图片数据表浏览编辑示例”。

(2) 用户界面设计。

① 在窗体上添加一个用于跟 BDE 数据库建立连接的 Table1 组件(位于组件面板 BDE 页),设置其 DataBaseName 属性为 DBDEMOS,TableName 属性设置为 BIOLIFE,Active 设置为 True。

② 在窗体上放置一个数据源 DataSource1 组件(位于组件面板 Data Access 页),设置其 DataSet 属性为 Table1。

③ 在窗体上放置一个控制读取数据的表格 DBGrid1(位于组件面板 Data Controls 页),设置其 DataSource 属性为 DataSource1。

④ 在窗体上添加一个 TDBNavigator 组件,用于表格导航。设置其 DataSource 属性为 DataSource1。

⑤ 在窗体上依次添加显示字段内容的备注 DBMemo1 组件、标签 TDBText1 组件和图片 DBImage1 组件,设置它们的 DataSource 属性为 DataSource1。然后设置 DBImage1 组件的 DataField 属性为 Graphic 字段,TDBText1 组件的属性为 Common\_Name 字段,DBMemo1 组件的 DataField 属性为 Notes 字段。

设置好所有组件的辅助属性:颜色、位置与大小合适后,窗体界面如图 9-25 所示。

(3) 保存并运行程序。

本程序不需要用户另外编写代码就可以正常运行。

选择 File→Save 命令,保存与项目有关的所有文件到项目目录。

通过选择 Run→Run 命令运行程序。如果程序没有编译和链接,在程序运行前系统会自动进行编译和链接,其运行效果如图 9-26 所示。

### 9.3.7 TDBCheckBox 组件与 TDBRadioGroup 组件

TDBCheckBox 组件是一个可以进行是非选择的复选框。在 TDBCheckBox 组件中,不但能表示是或非,而且还能指定任意两种数值。

TDBRadioGroup 组件提供一个由单选框组成的组。它也是和数据集中某个字段联系,并通过单选框的选择表示该字段的值。每一个 TDBRadioGroup 组件某个时刻只有一个单选框能被选中。

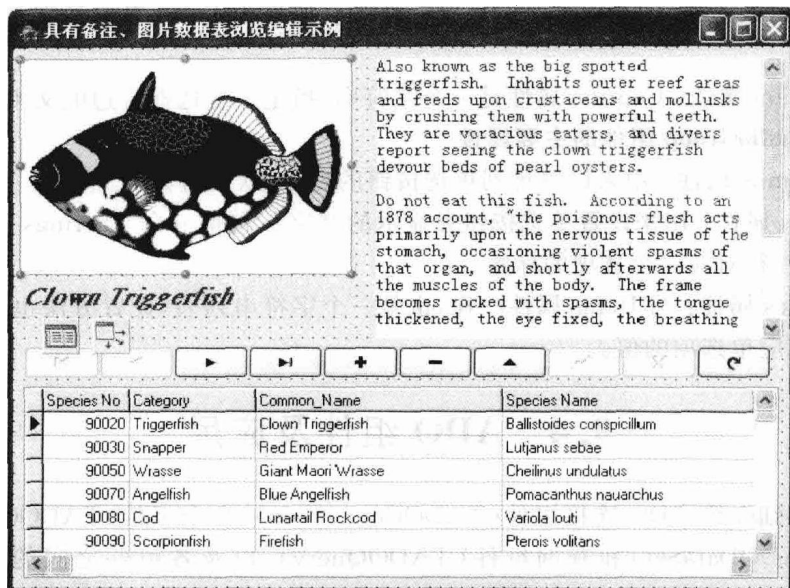


图 9-25 具有备注、图片数据表浏览编辑示例设计界面

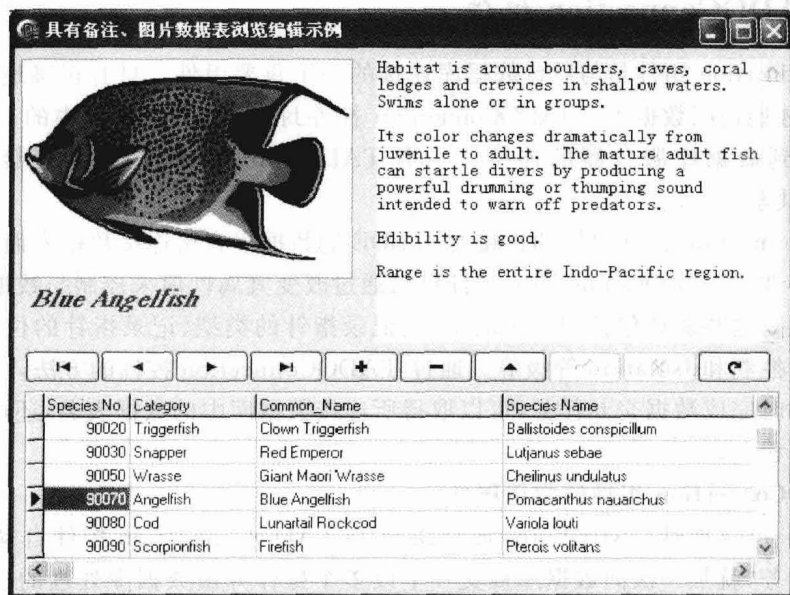


图 9-26 具有备注、图片数据表浏览编辑示例运行效果界面

### 1. TDBCCheckBox 组件的主要属性

(1) ValueChecked 属性: 该属性是字符串属性, 当字段的内容与该属性值匹配时, 该组件被选中。

(2) ValueUnchecked 属性: 该属性也是字符串属性, 当字段的内容与该属性值匹配时, 该组件被清除。

(3) Checked 属性: Checked 属性是一个 Boolean 类型变量, 表示复选框是否被选择。

如果为 True 表示已经被选中,框中应该有一个勾;如果为 False 表示没有被选中,框中是空白的。

(4) Caption 属性: Caption 属性是一个字符串,指定了复选框旁边的文本。

## 2. TDBRadioGroup 组件的主要属性

(1) Columns 属性: 用来设置排列单选按钮的列数,默认值为 1。

(2) Items 属性: 用来设置单选按钮旁显示的文字。它是一个 TStrings 类型的字符串数组,它的每一行对应一个选项的标签。

(3) Values 属性: 与 Items 属性一样,也是一个字符串列表,设置方法也与 Items 属性一样。表示相应单选项的值。

## 9.4 ADO 组件及应用

本节将详细介绍 ADO 连接组件(TADOConnection)、命令组件(TADOCommand)、数据集组件(TADODataset)和查询组件(TADOQuery),以及各组件之间的关系和用法。TADOConnection 是用来与 ADO 数据库建立连接的组件,各种 ADO 的数据集及操作组件可以共用这种连接来执行命令、读取数据并执行相应的操作。

### 9.4.1 TADOConnection 组件

TADOConnection 组件是与后台数据库连接的一个重要组件。只有正确地连接后,其他的 ADO 组件才能访问数据库。TADOConnection 控件封装了 ADO 技术中的 Connection 对象,提供一个到数据库服务器的连接。一个 TADOConnection 控件的连接可以为多个 ADO 数据集共享。

TADOConnection 控件提供的连接对比 BDE 的数据集直接连接数据库而言,有着更复杂的功能和属性。TADOConnection 控件可以通过改变其属性值来控制与数据库之间联系的属性和情况。这些属性包括记录锁的结构、记录指针的类型、记录指针的位置、连接数据库的超时时间控制和 Isolation 等级等。通过 TADOConnection 控件的方法,可以在使用联合数据集控件时完成数据交互任务,可以取得所连接数据库中的数据表名,还可以取得数据库中的存储过程名。

#### 1. TADOConnection 组件的常用属性

(1) Attributes 属性: Attributes 属性决定了 TADOConnection 控件自动处理事务的行为。所谓事务,就是一次向数据库提交一个或多个操作并由数据库处理的过程。该属性的原型定义如下:

```
property Attributes: TXactAttributes read GetAttributes write SetAttributes default [];  
TXactAttributes = set of TXactAttribute;  
TXactAttribute = (xaCommitRetaining, xaAbortRetaining);
```

可以看出 Attributes 属性包含两个选项: xaCommitRetaining,指定在连接的事务处理提交后启动新事务; xaAbortRetaining,指定在连接的事务处理回滚后启动新事务。系统默认值为空,不使用保留事务。

例如,将对象 ADOConnection1 的 Attributes 属性设置为[xaAbortRetaining]时,事务

处理如下：

```
ADODConnection1.BeginTrans;           //开始一个新事务
...                                     //一些其他处理代码
ADODConnection1.RollbackTrans;         //事务回滚后将自动执行 ADODConnection1.BeginTrans;
```

(2) CommandTimeout 属性：本属性指定了通过 TADODConnection 组件访问数据库的 TADODCommand 组件执行查询命令时允许的最长时间。该属性的原型定义如下：

```
property CommandTimeout: Integer read GetCommandTimeout write SetCommandTimeout default 30;
```

当一次命令执行时间超过 CommandTimeout 属性所指定的时间后还没有完毕，将取消命令做过的操作。CommandTimeout 属性单位是秒，默认值为 30s。例如：

```
ADODConnection1.CommandTimeout := 120;
```

该语句将连接 Command 对象尝试时间设置为 120s。

(3) Connected 属性：通过访问 Connected 属性可以知道 TADODConnection 控件是否连接到数据库，它是一个 Boolean 类型的变量，为 True 表示已经连接，而 False 表示没有连接。

通过 Open 和 Close 方法可以设置 Connected 属性，也可以直接设置 Connected 属性以建立和数据库的连接。该属性通常用于测试当前连接状态，例如：

```
With ADODConnection1 do
Begin
    Open;
    If Connected then
        {连接成功后的处理代码}
    Else
        {连接没有成功后的处理代码}
End;
```

(4) ConnectionString 属性：ConnectionString(连接字符串)是 TADODConnection 组件最重要的属性，用于指定数据库的连接信息。一般来说，ConnectionString 应由 5 项左右参数组成，参数说明如表 9-11 所示。

表 9-11 ConnectionString 的参数说明

参 数	说 明
Provider	数据提供者名称，例如 MSDASQL.1
Password	登录数据库的口令
Persist Security Info	支持安全登录
User ID	登录数据库用户名
Data Source	数据源名称，数据源的设置需要额外的操作

ConnectionString 属性所指定的连接字符串往往是非常复杂和冗长的，但是这并不意味着程序员的工作很复杂。在 Object Inspector 中双击这个属性，会弹出一个对话框，在这里可以选择构造 ConnectionString 属性的方法，设置步骤参考本章前面的相关内容。

(5) **ConnectOptions** 属性: 该属性指定数据库连接是按照同步方式还是异步方式连接,其原型定义如下:

```
property ConnectOptions: TConnectOption;  
type TConnectOption = (coConnectUnspecified, coAsyncConnect);
```

**coConnectUnspecified** 数据库连接采用同步方式连接,正常情况下采用这种方式连接;**coAsyncConnect** 异步方式连接数据库,当服务器负载很重的时候,这种连接方式很有用。引用这种连接方式,在第一次建立连接的时候,应用程序不能获得全部所需的数据。

异步连接的意义在于所有对数据库的操作都是不需等待操作完毕才返回的,如连接数据库,异步连接调用连接数据库的操作后马上就可以继续后面的代码,而同步连接必须确认连接正确后才能返回。默认为同步连接。

(6) **ConnectionTimeout** 属性: 指定尝试连接到数据源的最大允许时间。默认值为 15s。如果连接超时,则取消连接并产生异常,其原型为:

```
property ConnectionTimeout: Integer;
```

(7) **CursorLocation** 属性: 指定数据库游标是采用客户端模式还是服务器端模式。对于 **CursorLocation** 属性,它的两个选项分别是 **clUseServer**(表示服务器端方式)和 **clUseClient**(表示客户端方式)。默认值是 **clUseClient**,其原型定义如下:

```
property CursorLocation: TCursorLocation;  
type TCursorLocation = (clUseServer, clUseClient);
```

**clUseServer** 的含义是使用服务器端的数据库游标,适用于数据量大的数据集;**clUseClient** 的含义为使用客户端的数据游标时,数据将被下载到本地计算机上,并在本地进行操作。

(8) **DefaultDatabase** 属性: **DefaultDatabase** 属性是个字符串,它指定了一个用于连接的默认数据库。在连接字符串本来指定数据库位置或数据库位置无效时,连接将打开默认数据库;如果指定数据库位置正确,连接成功后该数据库即成为默认数据库。

(9) **DataSets** 属性和 **DataSetCount** 属性: 前面提到 ADO 数据集控件可以直接连接数据库,或者通过 **TADOConnection** 控件连接数据库。如果想知道哪些 ADO 数据集控件连接到一个 **TADOConnection** 控件,可以使用这两个属性。

**DataSets** 属性是一个数组,包含了所有连接的 ADO 数据集控件;而 **DataSetCount** 属性则说明了 ADO 数据集控件的个数。**DataSets** 原型定义如下:

```
property DataSets[Index: Integer]: TCustomADODataset read GetADODataset;
```

它们都是只读的属性,通常用来批量逐个访问所有 ADO 数据集控件。

(10) **InTransaction** 属性: 通过 **InTransaction** 属性可以得知一个 **TADOConnection** 控件是否处于处理事务的状态中。它是一个 **Boolean** 类型变量,为 **True** 表示有活动事务,为 **False** 表示没有活动事务,其定义如下:

```
property InTransaction: Boolean read GetInTransaction;
```

在开始一个事务后,InTransaction 被置为 True,直到提交该事务中所有的数据改变到数据库,或遇到放弃改变后,InTransaction 才会被置为 False。

(11) IsolationLevel 属性: 指定了不同事务之间的隔离级别。它的原型定义如下:

```
property IsolationLevel: TIIsolationLevel read GetIsolationLevel write SetIsolationLevel
default ilCursorStability;
TIIsolationLevel = (ilUnspecified, ilChaos, ilReadUncommitted, ilBrowse,
    ilCursorStability, ilReadCommitted, ilRepeatableRead, ilSerializable,
    ilIsolated);
```

IsolationLevel 属性表明了两个同时进行的事务处理作用于相同表时,一个事务能够看到多少另一个事务所做的工作。它有很多选项,默认值是 ilCursorStability。

(12) KeepConnection 属性: 指定如果在没有打开数据集的情况下是否仍然保持数据库的连接。频繁地打开和关闭数据库的操作将会影响系统的性能,特别是在网络上,会在一定的程度上增加网络的负载。这个属性设置数据源始终处于连接状态,可以显著提高程序的性能。

(13) LoginPrompt 属性: 将 LoginPrompt 属性设置为 True 时,会出现图 9-27 所示的登录对话框。将其设置为 False 时,可以避免显示对话框,不过还需要在连接字符串中设置好用户名和密码的信息。如果用户名和密码信息错误,登录对话框还是会被显示。其原型定义形式如下:

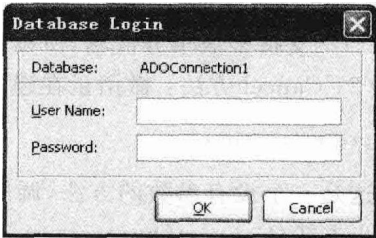


图 9-27 LoginPrompt 属性设置为 True 时出现登录对话框

```
property LoginPrompt default True;
```

这是一个 Boolean 类型变量,通过它可以避免连接数据库时显示登录对话框。

(14) Mode 属性: 决定本连接对数据库操作的权限。这种连接模式的值如表 9-12 所示。

表 9-12 连接模式参数及说明

参 数	说 明
cmUnknown	未指定数据库操作权限或无法确定
cmRead	对数据库只能读操作
cmWrite	对数据库只能写操作
cmReadWrite	对数据库可读写操作
cmShareDenyRead	禁止其他用户对数据库读操作
cmShareDenyWrite	禁止其他用户对数据库写操作
cmShareExclusive	禁止其他用户对打开数据连接
cmShareDenyNone	禁止其他用户对数据库任何操作

(15) State 属性: 表明当前连接的状态,这是一个只读的属性。其原型定义如下:

```
property State: TObjectStates read GetState;
TObjectState = (stClosed, stOpen, stConnecting, stExecuting, stFetching);
TObjectStates = set of TObjectState;
```



State 属性是一个集合,即状态可能是一个或者几个状态值的组合,所有的状态取值如表 9-13 所示。

表 9-13 State 的参数取值及含义

状 态 值	含 义	状 态 值	含 义
stClosed	处于关闭状态	stExecuting	正在执行命令
stOpen	处于打开状态	stFetching	正在从数据库中返回数据
stConnecting	正在连接数据库		

2. TADOConnection 的主要方法

(1) BeginTrans 方法: 执行这个方法启动一个新事务并触发 OnBeginTransComplete 事件,但是必须保证数据连接处于激活状态。BeginTrans 方法的原型定义如下:

```
function BeginTrans: Integer;
```

调用该方法同时还会将 InTransaction 属性设置为 True,该方法将返回一个 Integer 值,表示事务所处的嵌套层。大部分数据提供者(如 Microsoft OLE DB Provider for ODBC Driver)不支持多级事务嵌套,所以不推荐嵌套事务。

(2) Cancel 方法: 撤销正在建立的到数据库的连接。Cancel 方法的原型定义如下:

```
procedure Cancel;
```

这是一个没有参数的方法,调用这个方法可以放弃一个正在建立的连接。只有当连接为异步方式(ConnectOptions=coAsyncConnect)时才允许使用该方法。调用该方法必须在调用 Open 方法后成功连接到数据库之前。

(3) CommitTrans 方法: 向数据库提交一个事务,提交成功后,在事务中对数据库所作的修改则写入数据库中,同时一个事务也结束。CommitTrans 方法原型的定义如下:

```
procedure CommitTrans;
```

这是一个没有参数的方法。调用它将提交所有从调用 BeginTrans 开始的数据更新到数据库,并触发 OnCommitTransComplete 事件,同时将 InTransaction 属性设置为 False。

(4) Execute 方法: 可以用来直接执行一段 SQL 语句,和 TADOCommand 组件的功能差不多。它有两种原型定义,分别如下:

- 函数的形式

```
function Execute(const CommandText: WideString; const CommandType: TCommandType = cmdText; const ExecuteOptions: TExecuteOptions = []): _RecordSet; overload;
```

- 过程的形式

```
procedure Execute(const CommandText: WideString; const CommandType: var RecordsAffected: Integer; ExecuteOptions: TExecuteOptions = [eoExecuteNoRecords]); overload;
```

函数返回一个 RecordSet 对象,也就是一个记录集,一般用来执行返回数据集的 SQL 命令。过程是用来执行不返回数据集的 SQL 命令,如 Insert、Update。

CommandText 参数指定要执行的 SQL 命令; CommandType 参数指定 CommandText 参数的命令类型; RecordsAffected 参数指定该命令涉及的记录数目; ExecuteOptions 参数

指定命令特征选项与说明,如表 9-14 所示。

表 9-14 ExecuteOption 选项的值及说明

选 项	说 明
eoAsyncExecute	异步执行指定的命令
eoAsyncFetch	给定了 Cache 属性的值之后,再异步地取得数据
eoAsyncFetchNonBlocking	非阻塞式线程执行
eoExecuteNoRecords	没有返回记录

(5) GetProcedureNames 方法:调用 GetProcedureNames 方法可以得到当前连接的数据库中所有存储过程的名字,它的原型定义如下:

```
procedure GetProcedureNames(List: TStrings);
```

List 参数是一个字符串数组,每一个存储过程的名称将作为一个元素保存在其中。  
例如,下面的代码可以将所有存储过程的名字保存到一个 TMemo 控件中:

```
ADOConnection1.GetProcedureNames(Memo1.Lines);
```

注意: List 参数所指定的字符串数组中原来的内容将被清空。

(6) GetTableNames 方法:调用 GetTableName 方法可以得到当前连接的数据库中所有数据表的名字,它的原型定义如下:

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```

List 参数是一个字符串数组,每一个数据表的名称将作为一个元素保存在其中。  
例如,下面的代码可以将所有的表名字保存到一个 TMemo 控件中:

```
ADOConnection1.GetTableNames (Memo1.Lines)
```

(7) GetFieldNames 方法:该方法的功能是获取当前连接的数据库中某个指定数据表的所有字段名。它的原型定义如下:

```
procedure GetFieldNames(const TableName: String; List: TStrings);
```

TableName 参数:指定表名; List 参数:获取的字段存放在 List 中。

(8) Open 方法和 Close 方法:调用 Open 方法可以打开数据库的连接,Close 方法则用来关闭连接。它们的原型定义分别为:

```
procedure Open(const UserID: WideString; const Password: WideString); overload;  
procedure Close;
```

调用 Open 方法可以在连接的同时指定 UserID 参数和 Password 参数,也可以在连接字符串中指定。这时可以将 LoginPrompt 属性设置成 False,阻止登录对话框出现。

(9) RollbackTrans 方法:撤回一个没有全部执行的事务,事务撤回之后,事务中所作的任何修改都不会写入数据库。调用 RollbackTrans 方法可以回滚一个活动事务,废除所有从 BeginTrans 开始的更新,并触发 OnRollbackTransComplete 事件,设置 Intransaction 属性为 False。

3. ADOConnection 的主要事件

(1) OnBeginTransComplete 事件：这个事件在一次事务初始时触发，如调用 BeginTrans 方法。它的原型定义如下：

```
property OnBeginTransComplete: TBeginTransCompleteEvent read FOnBeginTransComplete write FOnBeginTransComplete;  
TBeginTransCompleteEvent = procedure(Connection: TADOConnection; TransactionLevel: Integer;  
const Error: Error;  
var EventStatus: TEventStatus) of object;  
TEventStatus = (esOK, esErrorsOccured, esCantDeny, esCancel, esUnwantedEvent);
```

触发后系统将传递 4 个参数。第一个参数 Connection 指定当前连接；第二个参数 TransactionLevel 指定事务嵌套层数；第三个参数 Error 引用一个 ADO Error 对象，最后一个参数 EventStatus 给出事件状态，其含义如表 9-15 所示。

表 9-15 EventStatus 事件状态及含义

状 态	含 义
esOK	操作执行成功
esErrorsOccured	操作导致错误，并将错误信息送入 Errors 集合中
esCantDeny	操作不能拒绝其他用户对数据源的访问(仅用于连接事件)
esCancel	操作被取消(仅用于连接事件)
esUnwantedEvent	操作中一个未预料的事件被激活

(2) OnCommitTransComplete 事件：这个事件在一次事务完成提交时触发，即 CommitTrans 语句之后。其原型定义为：

```
property OnCommitTransComplete: TConnectErrorEvent read FOnCommitTransComplete write FOnCommitTransComplete;  
TConnectErrorEvent = procedure(Connection: TADOConnection; const Error: Error; var EventStatus: TEventStatus) of object;
```

它的参数和 OnBeginTransComplete 事件类似，EventStatus 给出事件状态，可以参照表 9-14。

(3) OnConnectComplete 事件：这个事件发生在连接完成之后，如 Open 方法之后触发。原型定义如下：

```
property OnConnectComplete: TConnectErrorEvent read FOnConnectComplete write FOnConnectComplete;  
TConnectErrorEvent = procedure(Connection: TADOConnection; const Error: Error; var EventStatus: TEventStatus) of object;
```

它的参数和 OnBeginTransComplete 事件类似，一般用于异步连接后的处理。

(4) OnDisconnect 事件：这个事件发生在连接断开之后，如 Close 方法之后触发。原型定义如下：

```
property OnDisconnect: TDisconnectEvent read FOnDisconnect write FOnDisconnect;  
TDisconnectEvent = procedure(Connection: TADOConnection; var EventStatus: TEventStatus) of object;
```

它的参数和 OnConnectComplete 事件类似,一般用于异步断开后的处理。

(5) OnExecuteComplete 事件: 这个事件在调用 Execute 方法之后触发。原型定义如下:

```
property OnExecuteComplete: TExecuteCompleteEvent read FOnExecuteComplete write FOnExecuteComplete;  
TExecuteCompleteEvent = procedure (Connection: TADOConnection; RecordsAffected: Integer;  
const Error: Error; var EventStatus: TEventStatus; const Command: _Command; const Recordset: _  
Recordset) of object;
```

RecordsAffected 参数指定动作查询影响的记录数; Command 参数指向执行命令的 ADO Command 对象,如果没有使用 Command 对象而是直接使用 Connection 执行命令,参数值则为 nil(空);最后一个参数 Recordset 指向一个命令返回的 ADO Recordset 对象(使用 Select 语句时),如果没有 ADO Recordset 对象返回,则为 nil。EventStatus 参数前面已经叙述过,参照表 9-14。

(6) OnInfoMessage 事件: 当连接从数据库服务器端接收到一个消息时触发。原型定义如下:

```
property OnInfoMessage: TInfoMessageEvent read FOnInfoMessage write FOnInfoMessage;  
TInfoMessageEvent = procedure (Connection: TADOConnection; const Error: Error; var  
EventStatus: TEventStatus) of object;
```

服务器会在很多时候向客户端发送消息,如连接事件完成之后。EventStatus 参数前面已经叙述过,参照前面的表。

(7) OnRollbackTransComplete 事件: OnRollbackTransComplete 事件发生在回滚事务后,即调用 RollBackTrans 方法之后。原型定义如下:

```
property OnRollbackTransComplete: TConnectErrorEvent read FOnRollbackTransComplete write  
FOnRollbackTransComplete;  
TConnectErrorEvent = procedure (Connection: TADOConnection; const Error: Error; var  
EventStatus: TEventStatus) of object;
```

EventStatus 参数前面已经叙述过,参照表 9-14。

(8) OnWillConnect 事件: OnWillConnect 事件发生在客户端发出连接请求和服务器返回可以建立连接的信息之间,它可以临时更改连接的属性和信息。原型定义如下:

```
property OnWillConnect: TWillConnectEvent read FOnWillConnect write FOnWillConnect;  
TWillConnectEvent = procedure (Connection: TADOConnection; var ConnectionString, UserID,  
Password: WideString; var ConnectOptions: TConnectOption; var EventStatus: TEventStatus) of  
object;
```

在这个事件的处理代码中,可以对 ConnectionString、UserID 和 Password 这三个参数赋值,ConnectOptions 参数决定连接是同步还是异步。

(9) OnWillExecute 事件: OnWillExecute 事件发生在调用了 Execute 方法但是还没有将命令发出之前,它可以用来修改发出命令的内容和属性。原型定义如下:

```
property OnWillExecute: TWillExecuteEvent read FOnWillExecute write FOnWillExecute;  
TWillExecuteEvent = procedure (Connection: TADOConnection; var CommandText: WideString; var  
CursorType: TCursorType; var LockType: TADOLockType; var CommandType: TCommandType; var  
ExecuteOptions: TExecuteOptions; var EventStatus: TEventStatus; const Command: _Command;
```

```
const Recordset: _Recordset) of object;
```

在这个事件处理代码中,参数 CommandText 指定执行语句。参数 CursorType 指定数据集返回的记录指针类型。参数 LockType 指定数据集的锁定方式。CursorType 和 LockType 的含义见 TADODataSet 控件对象。参数 ExecuteOptions 指定执行方式,参见 ExecuteOptions 属性。这些属性都可以被处理代码更改。

(10) BeforeConnect 事件与 BeforeDisconnect 事件:均为无参数事件,BeforeConnect 事件发生在连接打开之前,而 BeforeDisconnect 事件发生在连接关闭之前。原型定义如下:

```
property BeforeConnect;  
property BeforeDisconnect;
```

(11) OnLogin 事件:发生在登录对话框关闭后。原型其定义如下:

```
property OnLogin;
```

(12) AfterConnect 事件与 AfterDisconnect 事件:均为无参数事件,AfterConnect 事件发生在连接建立之后,而 AfterDisconnect 事件发生在连接关闭之后。原型定义分别为:

```
property AfterConnect;  
property AfterDisconnect
```

#### 4. ADOConnection 组件的使用

许多远程数据库服务器包含安全特性以阻止未经授权的访问。通常,这些服务器在允许数据库访问之前要求输入用户名和密码进行注册。在设计时,如果服务器要求注册,当第一次连接到数据库时,一个标准注册对话框弹出来要求输入用户名和密码。

程序中可以使用 DataSets 属性与 DataSetCount 属性一起遍历当前所有激活的数据集。例如,以下代码循环设置所有数据集禁止任何与之关联的数据控制组件更新数据。

```
var  
    I: Integer;  
begin  
    with MyDBConnection do  
    begin  
        for I := 0 to DataSetCount - 1 do  
            DataSets[I].DisableControls;  
        end;  
    end;  
end;
```

在运行时可以用三种方式控制 Login。如果在服务器要求输入之前指定用户名和密码,并将 LoginPrompt 属性设为 False,那么 Login 对话框就不会出现。

(1) 在 ConnectionString 中加入用户名和密码。

例如:

```
procedure TForm1.Button2Click(Sender: TObject);  
begin  
    adoconnection1.ConnectionString := 'Provider = SQLOLEDB.1;Persist Security Info = False;  
Initial Catalog = pubs;Data Source = qq-1;User ID = sa;password = 123456';  
    adoconnection1.Open();
```

```
end;
```

(2) 将用户名和密码作为 Open 方法的参数。

例如:

```
ADOConnection1.Open('sa', '123456');
```

(3) 使用 OnWillConnect 事件。

```
procedure TForm1.ADOConnection1WillConnect(Connection: TADOConnection;  
    var ConnectionString, UserID, Password: WideString;  
    var ConnectOptions: TConnectOption; var EventStatus: TEventStatus);  
begin  
    UserID := 'sa';  
    password := '123456';  
end;
```

**例 9.6** TADOConnection 组件综合示例: 建立一个程序可以显示数据库中所有表的名称及其内容。

设计构想: 通过 TADOConnection 组件连接到数据库 dbdemos. mdb, 并调用 GetTableNames 方法获取数据库中所有表的名称, 显示在一个列表框中。用一个 TADOTable 组件作为数据集, 通过单击列表框中的表名来选取要打开的数据表, 并用一个 TDBGrid 组件显示表中的记录。

实现的详细步骤如下:

(1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令, 创建一个新的应用程序, 系统会自动创建一个空白的窗体。调整窗体的大小, 并设置窗体的 Caption 值为“TADOConnection 组件综合示例”。

(2) 用户界面设计。

① 在窗体上添加一个用于建立数据库连接的 ADOConnection1 组件(位于组件面板 dbGo 页), 通过双击该组件在向导对话框设置其连接到目标数据库 dbdemos. mdb。并将其 LoginPrompt 属性设置为 False。

② 添加一个 TADOTable 组件, 使用默认名 ADOTable1, ADOTable1 属性设置为 Connection= ADOConnection1。

③ 添加一个 TDataSource 组件, 使用默认名 DataSource1, DataSource1 属性设置为 DataSet= ADOTable1。

④ 在 Form1 窗体添加一个 TDBGrid 组件, 使用默认名 DBGrid1, DBGrid1 属性设置为 DataSource=DataSource1。

⑤ 在 Form1 窗体添加一个 TListBox 组件, 使用默认名 ListBox1。

(3) 代码编写。

① 编写窗体初始化代码。

选中 Form1, 将对象观察器切换到 Events 页, 双击 OnCreate 事件右边, 为 Form1 添加 OnCreate 事件处理过程。

```
//窗体初始化事件
```

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    ADOConnection1.Connected := true;           //设置它连接到数据库上
    //调用 GetTableNames 方法获取数据表名
    ADOConnection1.GetTableNames(ListBox1.Items, false);
end;

```

ListBox1 为列表框组件,其 Items 属性值为列表显示的内容,数据类型为 TStrings,可以当字符串数组使用,这里用来存放数据表名。参数 false 表示不读取数据库的系统表名。

## ② 编写 ListBox1 鼠标单击事件代码。

选中 ListBox1,将对象观察器切换到 Events 页,双击 OnClick 事件右边,为 ListBox1 添加 OnClick 事件处理过程。

```

//列表框鼠标单击选择数据库表,显示其内容
procedure TForm1.ListBox1Click(Sender: TObject);
begin
    //根据用户选择的表名重新打开新表中的内容
    with ADOTable1 do                               //开域语句
    begin
        active := False;                             //关闭数据表
        TableName := ListBox1.Items.Strings[ListBox1.ItemIndex]; //设置 TableName 属性
        active := true;                               //打开数据表
    end;
end;

```

当选择列表中的某项时,被选项的索引值可通过 ItemIndex 得到。因此,所选表名即为 ListBox1.Items.Strings[ListBox1.ItemIndex]。

## (4) 保存并运行。

选择 File→Save 命令,保存与项目有关的所有文件到项目目录。通过选择 Run→Run 命令运行程序,其运行效果如图 9-28 所示,用鼠标随意单击左边列表框中的数据表名,则右边表格中显示出该数据表中的数据。

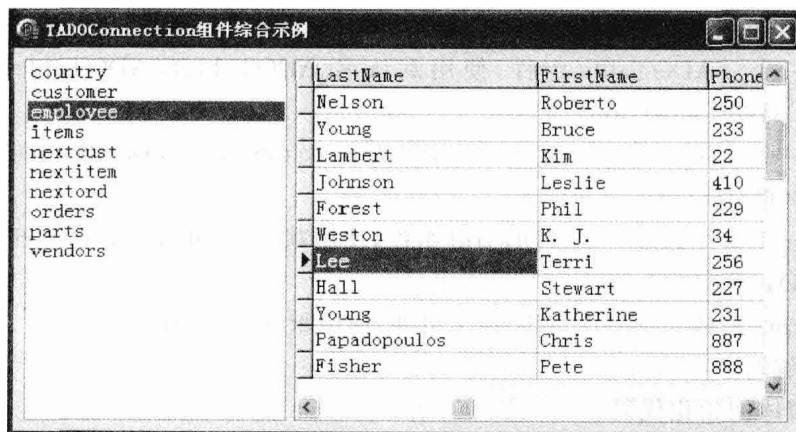


图 9-28 显示数据库有表名称及内容示例程序运行效果图

## 9.4.2 TADOCCommand 组件

TADOCCommand 组件主要用于运行一些数据定义语言(DDL)的 SQL 命令,或者运行一个没有返回结果集的存储过程,对于有返回结果集的 SQL 语句,则最好使用 TADODataset、TADOQuery 或 TADOStoredProc 组件。尽管 TADOCCommand 组件的 Execute 方法可以返回一个结果集,但却是通过另一个 ADO 数据集组件来使用该记录集。

TADOCCommand 组件执行的是其 CommandText 属性中设置的命令,通过调用 Execute 方法执行该命令,如果该命令中需要使用参数,则通过 Parameters 属性设置。

### 1. TADOCCommand 组件的常用属性

(1) CommandText 属性:指定要执行的 SQL 命令,可以手工编写,也可以激活 CommandText 编辑器来自动生成。

例如,假设已经存在连接目标数据库 dbdemos.mdb 的 ADOConnection1 组件,首先 ADOCommand1 对象的 Connection 属性设置成 ADOConnection1,然后可以在对象查看器(Object Inspector)中单击 CommandText 属性右侧的“...”按钮,在出现的图 9-29 所示的 CommandText Editor 对话框中设置这个属性。

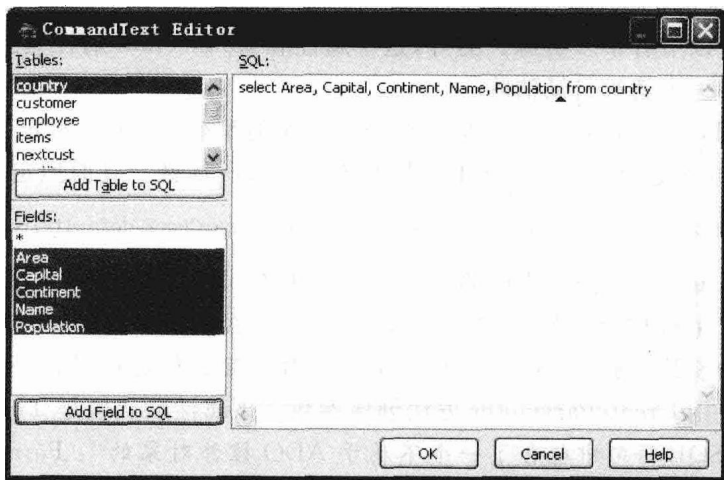


图 9-29 TADOCCommand 组件的 CommandText 编辑器

CommandText 编辑器是专门用来为 ADO 组件编写 SQL 命令的,Tables 列表框用来列出数据库中所有的表名,选中一个表,单击 ADD Table to SQL 按钮,CommandText 编辑器就会自动把表名插入到 SQL 命令的相应位置。在选中某个表的同时,这个表中的所有字段都会自动地列在 Fields 列表框中。同样,选中 Fields 列表框中的一个或者若干个字段,单击 Add Fields to SQL 按钮,字段就会插入到 SQL 命令中。

(2) CommandType 属性:指定要执行命令的种类,可以直接连接到数据库,也可以通过 TADOConnection 组件连接到数据库,还可以连接到其他的数据集。其原型定义为:

```
property CommandType: TCommandType;  
type TCommandType = (cmdUnknown, cmdText, cmdTable, cmdStoredProc, cmdFile, cmdTableDirect);
```

CommandType 可以是表 9-16 中所列值的一种。



表 9-16 CommandType 的选项及说明

选 项	说 明
cmdUnknown	CommandText 中的命令类型是未知的
cmdText	CommandText 中是一个 SQL 语句
cmdTable	CommandText 中指定的的是一个表的名称
cmdStoredProc	CommandText 中指定的的是一个存储过程的名称
cmdFile	CommandText 中指定的是保存数据集的文件名
cmdTableDirect	CommandText 中指定的是表的名称,并返回所有的列

(3) Connection 属性: 指定所使用的数据连接组件的名称,即 TADOConnection 组件的名称。通过这个属性使得 TADOCommand 组件能与数据库连接起来。

(4)ConnectionString 属性: 该属性原型定义如下。

```
property ConnectionString: WideString read FConnectionString write SetConnectionString;
```

如果直接连接数据库,那么就需要指定 ConnectionString 属性。这个属性的设置和使用在前面的 TADOConnection 组件中已有介绍。

如果和已有的 TADOConnection 组件连接到一个数据库,最好还是使用 TADOConnection 控件连接,这样可以节省很多资源。另外,这个属性和 Connection 属性是互斥的,设置了其中的一个以后,另一个就被自动清除。

(5) ParamCheck 属性: 该属性是一个 Boolean 类型变量,用于指定 TADOCommand 控件运行期间改动 SQL 语句时是否自动刷新 Parameters 属性。其原型定义如下:

```
property ParamCheck: Boolean read FParamCheck write FParamCheck default True;
```

该属性为 True 时,则根据新的 SQL 语句中“: Param”形式的参数自动初始化 Parameters 集合(生成相应数目的 Parameters 对象);否则,Parameters 集合不自动更新。如果是“?”形式的参数,则仅在 OLE DB Provider 返回参数信息的情况下自动初始化;否则,必须明确地使用 CreateParameter 方法创建参数。该属性默认值是 True。

**注意:** 如果 SQL 语句中包含了一个不属于 ADO 控件对象的“: Param”(如果用 DDL 语句创建一个存储过程,其中包含“: Param”,该参数属于存储过程),此时,需要将 ParamCheck 设置为 False,防止执行该语句时错误地初始化参数列表。

(6) ExecuteOption 属性: 指定调用 Execute 方法时的命令特征。介绍 TADOConnection 组件时就描述过它,详细取值与含义如表 9-13 所示。

(7) Parameters 属性: 在参数查询中,即在 SQL 命令中或在存储过程中需要传递参数的时候才需要设置这个值,并且在命令类型(CommandType)指定为 cmdText 或 cmdStoredProc 时,参数才有效。所谓参数,在使用上可以理解为变量,在执行 SQL 之前就被赋值。其原型定义如下:

```
property Parameters: TParameters;
```

(8) Prepared 属性: 这个属性用来表示一个命令在执行之前是否做好初始准备,这是一个 Boolean 类型变量,默认值是 False。原型定义如下:

```
property Prepared: WordBool read GetPrepared write SetPrepared default False;
```

(9) States 属性: States 属性和 TADOConnection 组件的 State 属性意义相同。原型定义如下:

```
property States: TObjectStates read GetState;
```

## 2. TADOCommand 的主要方法

(1) Assign 方法: 有时需要将一个 TADOCommand 控件的属性复制到另一个 TADOCommand 控件,这时可以使用 Assign 方法快速完成复制操作,其原型定义如下:

```
procedure Assign(Source: TPersistent); override;
```

例如:

```
ADOCommand1.Assign(ADOCommand2); //其功能是将 ADOCommand2 的属性复制到 ADOCommand1
```

(2) Cancel 方法: 用来撤销一次命令的执行。被撤销的命令必须是以异步方式运行的(在执行前被设置 eoAsyncExecute 参数)。如果命令不是被异步执行的,会产生一个错误。Cancel 方法必须在 CommandOut 属性指定的时间之内调用。

(3) Execute 方法: 用于在数据库端执行一条命令。它是 TADOCommand 组件最重要的方法,其语法原型为:

```
function Execute: _Recordset; overload;
function Execute(const Parameters: OleVariant): _Recordset; overload;
function Execute ( var RecordsAffected: Integer; const Parameters: OleVariant ): _
RecordSet; overload;
```

由定义可知,Execute 方法有三种执行方式:第一种是不使用任何参数;第二种是指定一个 Parameters 对象,Parameters 对象表明命令语句中的参数;第三种就是除了指定 Parameters 对象以外,还指定一个 RecordsAffected 参数,在执行完命令以后保存该命令涉及多少条记录的信息。如需要异步执行,则要在 ExecuteOptions 属性中设置。

执行 Insert、Update 和 Delete 等 SQL 语句时,没有返回值,其调用形式为 ADOCommand1.Execute;。

## 3. TADOCommand 组件的使用

TADOCommand 组件封装了 ADO 技术中的 Command 对象,通过它可以在数据库上执行 SQL 语句。在程序中使用 TADOCommand 组件,需要完成以下步骤:

(1) 使用 Connection 属性连接到一个 TADOConnetion 组件或使用 ConnetionString 属性直接连接数据库;

(2) 设置 CommandType 属性,指定 CommandText 属性中包括命令类型;

(3) 为 CommandText 属性设置命令文本,可以是 SQL 语句、表名和存储过程名;

(4) 调用 Excute 方法执行命令。如果调用 TADOCommand 组件的 Execute 方法执行一个需要返回结果的操作,Execute 方法会返回一个 ADO 记录对象。为了访问结果,需要一个数据集组件的 RecordSet 属性来接收结果。

例如:

```
with ADOCommand1 do
```

```

begin
  CommandType := cmdText;
  CommandText := 'Select company, state From Customer Where state = :stateParam';
  Parameters.ParamByName('stateParam').Value := 'HI';
  ADOQuery1.Recordset := Execute;
end;

```

ADO 数据集组件的 RecordSet 属性一旦设置以后,该数据集组件就会自动被激活,在程序中利用该数据集组件的方法和属性就可以访问这些数据。

如果调用 TADOCommand 组件的 Execute 方法执行一个不需要返回结果的操作,相对要简单些。

例如:

```

with ADOCommand1 do
begin
  CommandType := cmdText;
  CommandText := 'delete From Customer Where CustNo = 1221';
  Execute;
end;

```

### 9.4.3 TADODataset 组件

ADO 数据集组件有很多个,TADODataset 是 ADO 组件中最常用的一个。它可通过 SQL 命令从一个数据表或者是多个数据表获取数据集,也可以像 TADOTable 组件一样,直接获取整个数据表的数据,还可以执行存储过程,从磁盘文件获取数据。因此,TADODataset 有着与 TADOTable 相同的数据集相关的方法和属性,有着与 TADOCommand 相同的命令执行相关的方法和属性。

要使 TADODataset 数据集组件能够正常地发挥作用,则应首先设置其 Connection 或 ConnectionString 属性来建立起到数据库的连接。

由于 TADODataset 组件必须返回一个结果集,因此其 CommandText 属性中如果使用语句,则只能使用 SELECT 语句,而不能使用数据操纵语言(DML)如 DELETE、INSERT 和 UPDATE 语句。

TADODataset 的属性与方法跟前面介绍 TADOTable 的基本类似。TADODataset 是 ADO 组件中最常用的一个,有着许多与 TADOTable 组件基本类似的方法、属性和事件。

#### 1. TADODataset 的主要属性

其共同属性前面 TADOTable 组件介绍过了,在这里仅介绍 TADODataset 组件特有的属性。

(1) CommandText: 指定数据集合中所包含的命令,可以是 SQL 语句、一个表名或者一个存储过程名。应用于 TADODataset 组件中封装的 Command 对象,因此,可以参考 TADOCommand 组件的 CommandText 属性。

常用的调用形式为:

```

with ADODataset1 do
begin
  Close;

```

```

CommandType := cmdText;
CommandText := 'SELECT * FROM Customer';
Open;
end;

```

(2) DataSource: 设置数据源用来自另一个数据集的字段值去自动填充查询的参数。换句话说,在命令文本中的 SQL 语句所包含的参数与另一个数据集中的字段同名,执行 SQL 语句时,参数被同名字段的值取代。

## 2. TADODataset 组件的数据集类型

TADODataset 是一个多用途的数据集,通过设置 CommandType 和 CommandText 属性,它可以作为表类型、查询类型、存储过程类型和文件类型的数据集使用。

### 1) 表类型数据集

作为表类型数据集时,TADODataset 从单一的数据表中获取所有行和列。使用这种类型,需先设置 CommandType 属性为 cmdTable,然后设置 CommandText 属性为指定的表名。表类型数据集类似于 TADOTable 组件,具有 TADOTable 组件的所有功能。

CommandType 属性与 CommandText 属性可以在设计时指定,也可以在运行时设置。在程序代码中的使用方法如下:

```

with ADODataset1 do begin
  CommandType := cmdTable;
  CommandText := 'employee';
  Open;
end;

```

### 2) 查询类型数据集

TADODataset 也可以执行 SQL 查询命令以获取数据集,使用时需要先设置 CommandType 属性为 cmdText,然后在 CommandText 属性中设置要执行的 SQL 命令。

在设计时,在对象观察器中双击 CommandText 属性,打开 CommandText Editor 对话框,可以方便地创建 SQL 命令。运行时也可以动态设置 CommandText 属性。

查询使用的命令文本中也可以使用参数。运用参数,不需要修改 SQL 语句,给定不同的参数值就可以获得不同的执行结果。比如设置命令文本如下:

```

ADODataset1.CommandText := 'INSERT INTO Country (Name, Capital, Population)
VALUES (:pName, :pCapital, :pPopulation)'

```

语句中的“:pName”、“:pCapital”和“:pPopulation”相当于占位符,在程序运行时由参数的实际值取代。

在程序运行时,可以用下面的代码设置参数的值;

```

with ADODataset1 do
begin
  Close;
  Parameters.ParamValues['pName'] := edit1.text;
  Parameters.ParamValues['pCapital'] := edit2.text;
  Parameters.ParamValues['pPopulation'] := edit3.text;
  Open;
end;

```

### 3) 存储过程类型数据集

TADODataSet 需要指定所执行的存储过程的名称。使用这种类型,先设置 CommandType 属性为 cmdStoredProc,然后在 CommandText 属性中设置要执行的存储过程名称。存储过程的使用在第 8 章介绍过。

### 4) 文件类型数据集

使用这种类型数据集,需先设置 CommandType 属性为 cmdFile,然后在 CommandText 属性中设置要打开的文件名。

**注意:** 在设置 CommandText 和 CommandType 属性之前,必须连接 TADODataSet 到一个由 Connection 和 ConnectionString 属性所指定的数据存储。

**例 9.7** TADODataset 组件综合示例:设计一个程序显示两个具有主/从关系表的内容。

设计构想:在数据库 dbdemos.mdb 中,表 customer 存储的是顾客信息,表 orders 存储的是订单信息。编程实现能方便地通过选定顾客来浏览其订单信息。TADODataset 组件的参数与 DataSource 属性配合使用,可以建立主/从关系的数据集。

实现的详细步骤如下:

#### (1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“主/从关系示例程序”。

#### (2) 用户界面设计。

① 在窗体上添加一个用于建立数据库连接的 ADOConnection1 组件(位于组件面板 dbGo 页),通过双击该组件在向导对话框设置其连接到目标数据库 dbdemos.mdb。并将其 LoginPrompt 属性设置为 False。

② 添加一个 TADODataset 组件,使用默认名 ADODataset1。将其 Connection 属性设置为 ADOConnection1,CommandType 属性设置为 cmdTable,CommandText 属性设置为 'customer',Active 属性值为 True。

③ 添加一个 TDataSource 组件,使用默认名 DataSource1,将其 DataSet 属性设置为 ADODataset1。然后添加一个 TDBGrid 组件,使用默认名 DBGrid1,将其 DataSource 属性设置为 DataSource1。

④ 添加一个 TADODataset 组件,使用默认名 ADODataset2,将其 Connection 属性设置为 ADOConnection1,CommandType 属性设置为 cmdText,CommandText 属性设置为 'select \* from orders where CustNo=:CustNo',设置 DataSource 属性为 DataSource1。然后在对象查看器(Object Inspector)中单击 Parameters 属性右侧的“...”按钮,通过对话框设置 CustNo 参数的 DataType 为 ftInteger,最后设置 ADODataset2 的 Active 属性为 True。

⑤ 添加一个 TDataSource 组件,使用默认名 DataSource2;,设置 DataSet 属性为 ADODataset2。

⑥ 在 Form1 窗体添加一个 TDBGrid 组件,使用默认名 DBGrid2;,将其 DataSource 属性设置为 DataSource2。

⑦ 添加两个标签组件 Label1 与 Label2,设置其 Caption 属性分别为“主表内容显示”和“从表内容显示(通过 CustNo 关联)”。

调整好所有组件的位置及大小后,窗体如图 9-30 所示。

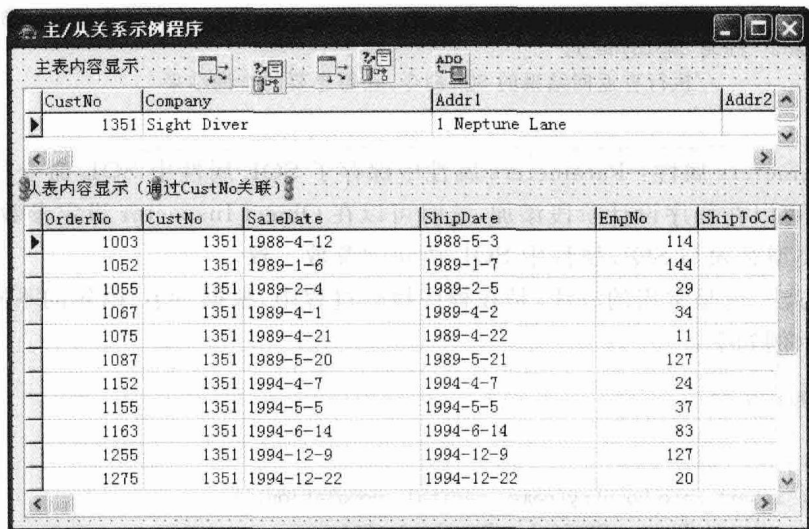


图 9-30 主/从关系示例程序设计窗体界面

(3) 保存并运行。

本程序不需要另外再编写任何代码即可以正常运行。

选择 File→Save 命令,保存与项目有关的所有文件到项目目录。通过选择 Run→Run 命令运行程序。用鼠标随意单击上面表格里主表中的记录,则下面从表中显示出跟该主表相关联的数据内容。

#### 9.4.4 TADOQuery 组件

TADOQuery 组件借助于 SQL 语言的强大功能访问多个数据表,可以实现数据浏览、修改和删除等操作,而且 TADOQuery 组件也可以实现参数查询。

通常情况下,使用 TADOQuery 组件是为了从数据集中查询一部分字段或记录,也可以使用 INSERT、DELETE、UPDATE 和 ALTER TABLE 等 SQL 命令实现数据库的更新、插入和删除记录的操作。如果数据集只包含一个基表,则可以使用 TADOQuery,也可以使用 TADOTable 组件。作为一个数据集组件,TADOQuery 也具有与 TADOTable 组件相同的与数据集相关的属性和方法;而作为专用组件,它有一些自己独有的属性和方法。下面仅讨论 TADOQuery 的主要属性和方法。

##### 1. TADOQuery 组件的常用属性

(1) SQL 属性: SQL 属性是 TStrings 类型的变量,包含了 TADOQuery 组件要执行的 SQL 命令,它是 TADOQuery 最为重要的属性之一,在应用程序中,可以调用 Open 方法或 ExecSQL 方法来执行在 SQL 属性中指定的 SQL 语句。在设计阶段,可以利用属性编辑器编写,在应用程序执行过程中也可以动态修改。请看下面的例子:

```
with ADOQuery1 do
```

```

begin
    //重新写入 SQL 时必须关闭原来的查询
Close;
    SQL.Clear;                                //清除原来的 SQL 命令
    SQL.Add('SELECT EmpNo, LastName, FirstName, HireDate'); //设置新 SQL 命令
    SQL.Add('FROM Employee');
Open;    //执行有返回结果的 SQL 命令,获得想要的查询结果
end;

```

(2) Parameters 属性: Parameters 属性中保存了 SQL 属性中 SQL 命令执行所需的参数,这些参数可以在程序设计阶段添加,这时可以在 Object Inspector 设定参数的值,并且参数的数量和类型必须与 SQL 属性中 SQL 语句的参数一致。

另一种方法,也是常用的方法,是在程序执行过程中,根据 SQL 语句的不同可以动态设置参数值。示例如下:

```

with ADOQuery1 do
begin
    SQL.Clear;
    SQL.Add('insert into country(name, capital, population)');
    SQL.Add('values(:Name, :Capital, :Population)');
    Parameters.ParamByName('Name').Value := 'Liechtenstein';
    Parameters.ParamByName('Capital').Value := 'Vaduz';
    Parameters.ParamByName('Population').Value := 420000;
    Prepared := true;
    ExecSQL;
end;

```

(3) RowsAffected 属性: 该属性返回最近一次查询所影响的记录数。其语法原型为:

```
property RowsAffected: Integer;
```

如果由于错误导致 SQL 语句不能执行,则返回-1。

## 2. TADOQuery 组件的常用方法

(1) Open 方法: 用于执行 SQL 属性所指定的 SQL 命令,要返回记录集。通过调用 Open 方法或设置 Active 属性为 True, TADOQuery 组件可以执行 Select 命令和存储过程,并返回结果数据集。TADOQuery 组件还可使用参数化查询,不需要修改 SQL 语句,给定不同的参数值,就可以获得不同的查询结果。

(2) ExecSQL 方法: 用于执行 SQL 属性所指定的 SQL 命令,不需要返回记录集。如 Insert、Update 或 Delete 等命令。通过调用 ExecSQL 方法, TADOQuery 组件可以执行 Insert、Update 或 Delete 等 SQL 命令,这些命令执行后不会返回数据集。

无论是调用 Open 方法还是 ExecSQL 方法,在执行它们之前,都必须调用 Close 方法关闭数据集。

**例 9.8** TADOQuery 组件综合示例: 设计一个程序根据输入条件查找表记录,然后对该记录实现更新(修改)操作。

设计构想: 编写应用程序用于修改数据库 dbdemos.mdb 中的表 employee 中的 LastName 字段。要求输入员工编号和姓,然后根据员工编号修改相应的记录。

实现的详细步骤如下：

(1) 建立新的应用程序。

在 Delphi 集成开发环境中通过选择 File→New→VCL Forms Application—Delphi for Win32 命令,创建一个新的应用程序,系统会自动创建一个空白的窗体。调整窗体的大小,并设置窗体的 Caption 值为“TADOQuery 组件综合示例”。

(2) 用户界面设计。

① 在窗体上添加一个用于建立数据库连接的 ADOConnection1 组件(位于组件面板 dbGo 页),通过双击该组件在向导对话框设置其连接到目标数据库 dbdemos.mdb。并将其 LoginPrompt 属性设置为 False。

② 在窗体上放置一开始用于映射数据库表的 TADOQuery 组件(位于组件面板 dbGo 页),使用默认名 ADOQuery1,设置其 Connection 属性为 ADOConnection1。

③ 在窗体中选中 ADOQuery1 组件对象,然后用鼠标单击其对象查看器 Object Inspector 中 SQL 属性右边的“...”按钮,出现一个 SQL 查询命令编辑器,输入“Select \* from employee”,如图 9-31 所示,再单击 OK 按钮。最后将 ADOQuery1 的 Active 属性设置为 True。

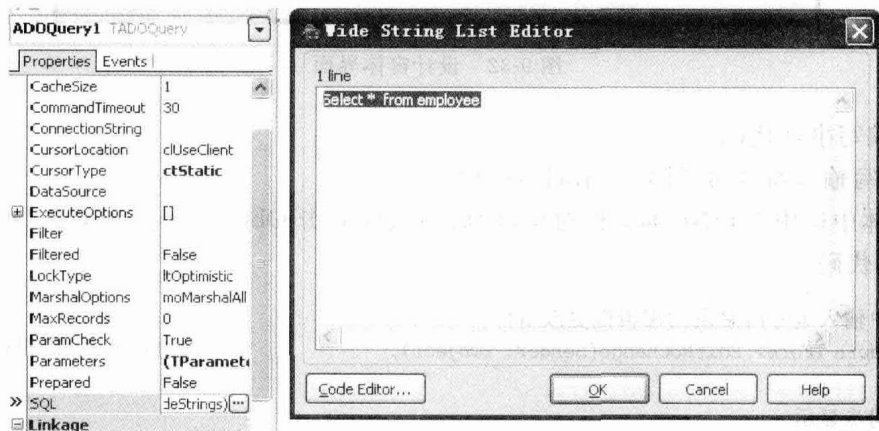


图 9-31 SQL 查询命令编辑器

④ 在窗体上放置一个数据源 DataSource1 组件(位于组件面板 Data Access 页),设置其 DataSet 属性为 ADOQuery1。

⑤ 在窗体上添加一个 TDBNavigator 组件(位于组件面板 Data Controls 页),用于表格导航。在 Object Inspector 中展开 VisibleButtons 属性(点击该属性),将其子属性 nbInsert、nbDelete、nbEdit、nbPost、nbCancel 和 nbRefresh 设置为 false,使它们隐藏起来。设置其 DataSource 属性为 DataSource1。

⑥ 在窗体上放置一个控制读取数据的表格 DBGrid1 (位于组件面板 Data Controls 页),设置其 DataSource 属性为 DataSource1,DBGrid1 的 ReadOnly 属性为 True。

⑦ 添加一个命令按钮(TButton 组件),设置其 Name 属性为 btnUpdate,Caption 属性为“更新”;添加两个 TLabel 组件,使用默认名称为 Label1、Label2,依次设置其 Caption 属性为“根据员工编号查找”和“要修改的员工 LastName”;最后再添加两个 TEdit 组件,依次



设置其 Name 为 editNo 和 editname,并清空它们的 Text 属性。

调整好所有组件的位置及大小后,窗体界面如图 9-32 所示。



图 9-32 设计窗体界面

### (3) 编写事件代码。

#### ① 编写输入编号时,姓名显示同步的代码。

在窗体中选中 EditNo 编辑框对象,然后用鼠标双击 Object Inspector 的 OnChange 事件,编写其代码。

```
//用户输入编号时触发的编辑框更改事件
procedure TForm1.EditNoChange(Sender: TObject); //使得姓名显示与编号同步
begin
    //同步显示
    With ADOQuery1 do
    begin
        if Active then
        begin
            if Locate('EmpNo', editNo.Text, []) then //查找用户输入的编号
                EditName.Text := ADOQuery1.FieldByName('LastName').AsString; //同步
        end;
    end;
end;
```

#### ② 编写定位到目标编号时,更新姓名的代码。

在窗体中用鼠标双击按钮 btnUpdate 为 btnUpdate 添加 OnClick 事件处理过程。

```
//鼠标单击更新按钮事件代码
procedure TForm1.btnUpdateClick(Sender: TObject);
var s:String;
begin
    //更新指定员工编号的 LastName
```

```

with ADOQuery1 do
begin
    if Active then
        Close;                                //关闭数据集
    SQL.Clear;                                //清空 SQL 字符串
    s := 'update employee set LastName = ''' + editname.Text + ''' +
        ' where EmpNo = ' + editNo.Text;
    SQL.Add(s);                                //加入本次更新命令
    ExecSQL;                                  //执行根据条件 update 的 SQL 语句
end;
//更新显示
With ADOQuery1 do
begin
    if Active then
        Close;                                //关闭查询表格内容
    SQL.Clear;                                //清空 SQL 字符串
    s := 'select * from employee';            //显示表格所有内容
    SQL.Add(s);                                //加入查询更新命令
    ExecSQL;                                  //执行显示表所有内容的 SQL 语句
    Active := True;                            //打开表显示
    Locate('EmpNo', editNo.Text, []);          //定位到新的编号
end;
end;

```

#### (4) 保存并运行。

选择 File→Save 命令,保存与项目有关的所有文件到项目目录。通过选择 Run→Run 命令运行程序,其运行效果如图 9-33 所示。输入要修改的员工编号,则可通过 SQL 语句的执行修改。

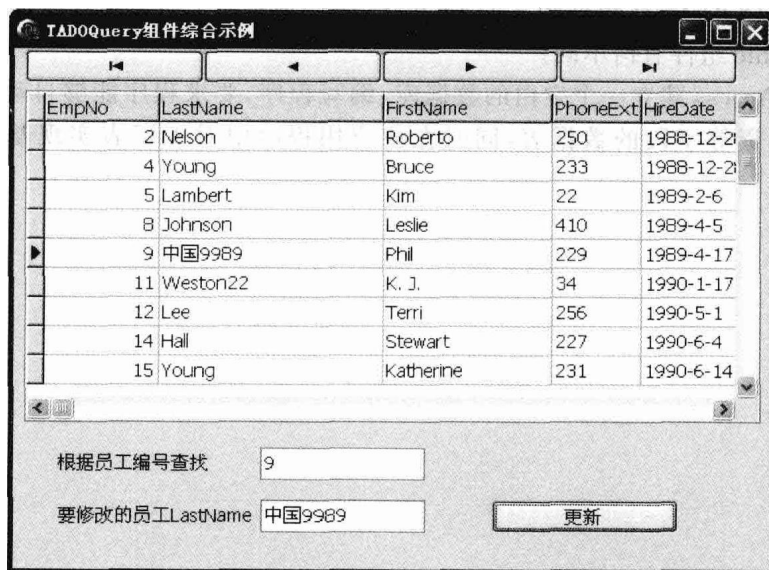


图 9-33 程序运行界面

## 本章小结

本章介绍了数据库主要基本概念以及 Delphi 访问数据库的常用技术与编程方法,详细介绍常用的数据访问组件和数据控制组件的属性、方法、事件以及使用方法。本章的重点就是 ADO 数据库访问方式。ADO(ActiveX Data Object)是 Microsoft 比较流行的数据访问接口,通过它可以访问多种类型的数据库。

本章通过多个实例介绍了访问数据库方式的一般步骤和具体实现方法,希望通过这些实例能让读者快速地掌握 Delphi 数据库应用程序的设计与开发。

## 思考与练习

1. 什么是数据库?什么是数据库管理系统?什么是数据库系统?
2. 简述在 Delphi 中创建数据库应用程序的过程。
3. 什么是数据集组件?数据集组件在数据库应用程序中起什么作用?
4. ADO 数据集组件有哪些?为什么说它们有许多共同的属性、方法和事件?
5. TADOTable 组件提供了哪些数据查询方法?如何使用?
6. 什么是记录指针?记录指针有哪些特殊位置?在程序中如何判断?
7. TADOTable 组件连接数据库有两种方式,请分别描述这两种方法的连接过程。
8. 简述 TADOConnection 组件连接 Access 数据库和 MS SQL 数据库服务器的方法。
9. Delphi 提供了哪些 ADO 组件?请说明这些组件之间的关系。
10. 比较 TADOTable、TADODataset 和 TADOQuery 组件之间的异同。
11. 用 TADODataset 组件和 TADOQuery 组件编写一个通讯录程序,体会在功能实现上与 TADOTable 组件有何不同。
12. 使用 Access 建立一个空白的数据库,编写程序,要求程序能够自动创建一个具有工号、姓名和工资等字段的数据表,同时在该应用程序中对工资表实现基本的增、删、改操作。

## 参 考 文 献

- 1 刘艺. Delphi 面向对象编程思想. 北京: 机械工业出版社, 2003.
- 2 董玉德. 面向对象的程序设计方法与技术(Delphi 语言). 北京: 清华大学出版社, 2008.
- 3 颜金传. Delphi 2007 典型开发实例. 北京: 电子工业出版社, 2009.
- 4 李文池. Delphi 程序设计基础. 北京: 中国水利水电出版社, 2006.
- 5 周果宏. Delphi 程序设计. 第 2 版(Delphi 2005). 北京: 清华大学出版社, 2006.
- 6 蒋先刚. 基于 Delphi 的数字图像处理工程软件设计. 北京: 中国水利水电出版社, 2006.
- 7 张铭华. 多媒体视频程序设计——使用 Delphi. 北京: 中国铁道出版社, 2006.
- 8 陈秋劲. Delphi 数据库编程. 北京: 机械工业出版社, 2007.
- 9 徐长梅等. Delphi 2005 数据库基础教程. 武汉: 武汉大学出版社, 2006.