PowerBuilder 10.0 实用教程

崔 巍 陈雍先 编著

清华大学出版社

- 知识—介绍全面、叙述详尽、浅显易懂
- 示例—典型丰富、前后贯通、操作性强
- 结构—围绕主题、构思严谨、层层深入

ISBN 7-302-13428-6

9 787302 134282 >

定价：34.00元

新书查询及技术支持：http://www.wenyuan.com.cn
读者服务邮箱：service@wenyuan.com.cn

PowerBuilder 10.0 实用教程

崔　巍　陈雍先　编著

清华大学出版社
北　京

内 容 简 介

PowerBuilder 10.0 是 PowerSoft 公司最新推出的可视化数据库系统集成开发工具，非常适用于开发管理信息系统。本书围绕 PowerBuilder 10.0 提供的技术，系统阐述了 PowerScript 编程基础、应用对象及事务对象、窗体、菜单、数据窗口、管道对象、用户对象、库文件以及分布式应用；重点讨论了应用对象的设计、数据窗口的设计、数据库的连接以及用户对象和库文件的创建。

在讲解理论知识的同时，本书利用实例来介绍具体的设计方法及过程，例如通过身份验证窗体的设计来介绍如何来设计窗体，通过学生数据库的建立和连接介绍 PowerBuilder 最具特色的数据窗口的使用，等等。同时为便于读者复习与练习，各章节都附有复习思考题及参考答案。

本书内容全面，实例丰富，概念清晰，图文并茂，既适合数据库编程初学者的入门学习，也可作为系统开发程序员和数据库编程爱好者的参考书。

版权所有，翻印必究。举报电话：010-62782989　13501256678　13801310933
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。
本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目(CIP)数据

PowerBuilder 10.0 实用教程/崔巍，陈雍先编著. —北京：清华大学出版社，2006.8
ISBN 7-302-13428-6

Ⅰ. P…　Ⅱ. ①崔… ②陈…　Ⅲ. 数据库系统—软件工具，PowerBuilder 10.0—程序设计—教材
Ⅳ.TP311.56

中国版本图书馆 CIP 数据核字(2006)第 079686 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969

组稿编辑：邹　杰
文稿编辑：彭　欣
排版人员：王　婷
印 刷 者：北京密云胶印厂
装 订 者：北京市密云县京文制本装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：24.75　**字数**：591 千字
版　　次：2006 年 8 月第 1 版　2006 年 8 月第1 次印刷
书　　号：ISBN 7-302-13428-6/TP·8432
印　　数：1～4000
定　　价：34.00 元

前　言

当今时代信息技术已成为发展最迅速、最引人瞩目的技术之一。与此同时，系统的概念和系统的方法也已经渗透到社会发展的多个方向。对于计算机程序员和工程师来讲，在开发数据库应用系统或信息管理系统时，都希望能够拥有一个有效并且易于掌握的开发工具。为此，我们编写本书向广大读者介绍使用 PowerBuilder 10.0 进行程序开发的方法。

本书的特点是：基于单机开发，通过大量简单、实用而又有代表性的开发实例，向读者讲解如何使用 PowerBuilder 10.0 中提供的工具和功能来实现所需要的系统，并介绍一些设计和编程的技巧。同时安排在课后的习题能够让读者加强对本章知识的掌握。

对于初学者，此书可以作为 PowerBuilder 10.0 不可多得的入门教程；对于有一定基础的中高级水平的开发人员，本书也提供了 PowerBuilder 10.0 的新特点和使用方法，并提供了相应的实例，可以帮助其快速提高。

本书共分为 15 章，各章内容介绍如下：

第 1 章首先介绍了 PowerBuilder 的版本、特点、安装和启动，然后是它的集成开发环境和应用程序界面的开发。本章还给出了第一个数据库应用程序，使读者对数据库应用程序有一个初步的感性认识。

第 2 章介绍了常用的后台数据库 SQL Server 2000，并且介绍了常用的 SQL 语句，从而为读者理解实例的实现过程奠定基础。

第 3 章是有关使用 PowerBuilder 编程的基本概念和知识：程序的结构和机制、语言基础、程序的控制结构、函数和结构及编程规范。

第 4 章介绍应用对象和事务对象的相关概念和设计方法。

第 5 章详细介绍了如何创建窗体对象和菜单对象、设置属性以及编写事件处理程序。

第 6 章介绍控件的相关知识。利用控件可以完成许多界面设计任务而无需编写代码；还可以迅速构造出操作方便、使用灵活、安全稳健及界面友好的应用程序。

第 7 章介绍如何创建菜单对象、设置属性以及编写事件处理程序。菜单是各种操作的集合，可以将各种操作进行分类。

第 8 章和第 9 章介绍数据库基础、连接数据库以及数据窗口的建立、应用，然后详细介绍了数据窗口画板、事务对象的使用、数据窗口缓冲区内容的读取以及存取数据窗口对象的属性和数据窗口缓冲区数据的共享，最后介绍了一个关于增加数据窗口数据的实例，读者可以深入了解数据窗口的高级应用。

第 10 章介绍了如何在应用程序中使用管道对象(PipelineObject)把数据从一个或多个源表中输送到一个新的或现有的目标表中。

第 11 章介绍了用户对象的相关知识。用户对象实际上是对 PowerBuilder 的一种扩展，熟练使用 PowerBuilder 提供的标准控件和用户自定义对象，能够满足各种各样的程序开发需求。

第 12 章介绍了库文件的相关知识，目的是让用户全面了解 PowerBuilder 10.0 中的库画笔，并掌握管理库文件的方法。

第 13 章介绍程序调试和发布技术，包括如何设置断点、单步运行和错误处理等以及如何编译、打包、安装和卸载程序等。

第 14 章介绍分布式应用。所谓分布式应用是指在事务过程中，整个事务的处理分布到多个实体上来完成(在计算机网络中即多台计算机)。

第 15 章介绍如何使用 PowerBuilder 开发 Jaguar 组件以及在 PowerBuilder 中创建和使用 Jaguar CTS 服务器的客户应用程序的方法。

由于作者水平有限，书中难免有不足和疏忽之处，恳请广大专家读者批评指正。

编　者

2006 年 4 月

目　录

第 1 章　PowerBuilder 10.0 概述 1

1.1 PowerBuilder 10.0 简介 1
1.2 安装和启动 PowerBuilder 10.0 2
1.2.1 安装 PowerBuilder 10.0 2
1.2.2 启动 PowerBuilder 10.0 4
1.3 PowerBuilder 10.0 开发环境 5
1.3.1 菜单系统 6
1.3.2 工作区 8
1.3.3 工具栏 8
1.3.4 画板 10
1.3.5 帮助系统 10
1.4 窗口和菜单简介 11
1.4.1 窗口简介 11
1.4.2 菜单简介 14
1.5 实例：使用 PowerBuilder 创建第一个应用程序 16
1.6 习题 21

第 2 章　SQL 基础 22

2.1 SQL Server 数据库管理概述 22
2.1.1 SQL 语句简介 22
2.1.2 企业管理器 22
2.1.3 创建数据库 23
2.1.4 创建表 24
2.1.5 创建视图 27
2.2 SQL 的数据定义功能 30
2.2.1 Select 子句 31
2.2.2 Where 子句 31
2.2.3 From 子句 34
2.2.4 更名运算 34
2.2.5 元组变量 35
2.2.6 排列元组的显示次序 (order by) 35
2.2.7 集合操作 35
2.2.8 统计查询 36
2.2.9 嵌套子查询 38
2.2.10 SQL 查询示例 40
2.3 数据库修改 43
2.3.1 删除 43
2.3.2 插入 44
2.3.3 更新 45
2.4 实例：利用 SQL 操作进行身份验证 45
2.5 习题 48

第 3 章　PowerBuilder 编程基础 49

3.1 面向对象的编程特点 49
3.2 PowerScript 语言基础 50
3.2.1 标识符，注释，代词 50
3.2.2 数据类型 52
3.2.3 变量与常量 53
3.2.4 数组 55
3.2.5 运算符 56
3.2.6 表达式 57
3.3 PowerScript 编程基础 58
3.3.1 控制结构 58
3.3.2 函数和结构 62
3.3.3 编程规范 73
3.4 实例：文件复制 74
3.5 习题 77

第 4 章　PowerBuilder 应用基础 78

4.1 应用对象基础 78
4.1.1 应用对象概述 78
4.1.2 应用对象和应用对象画板 78
4.1.3 应用对象的事件 85
4.2 使用代码编辑器 87
4.2.1 代码编辑器 87
4.2.2 编辑代码 87
4.2.3 粘贴函数 89

4.2.4 粘贴语句 90
4.2.5 代码编辑器的键盘操作 90
4.3 事务对象基础 91
4.3.1 事务对象概述 91
4.3.2 创建自定义事务对象 92
4.4 实例：创建应用程序对象 93
4.5 习题 95

第 5 章 窗体技术 96

5.1 窗体概述 96
5.2 设计窗体 97
5.2.1 窗体画板概述 98
5.2.2 创建新窗体 98
5.2.3 创建继承窗体 99
5.2.4 设计窗体属性 100
5.2.5 保存、预览、打印窗体 103
5.3 窗体事件和函数 104
5.3.1 窗体事件 104
5.3.2 窗体函数 107
5.4 窗体变量和窗体数组 111
5.4.1 窗体变量 111
5.4.2 窗体数组 111
5.5 窗体设计实例 112
5.5.1 创建图标栏应用 112
5.5.2 创建美观的状态条 114
5.5.3 在 Response 窗口中使用 Menu 114
5.5.4 移动窗口中的控件 115
5.5.5 捕获控制菜单中的 Close 事件 115
5.5.6 将可视控件放置于其他的控件上 115
5.5.7 实现动态显示控件 116
5.5.8 动态创建可视控件 116
5.5.9 使用多窗口实例 117
5.5.10 在实例窗口中引用控件 118
5.5.11 向窗口传递多个参数 119
5.5.12 设计实时帮助条 119
5.5.13 实现“跑马灯” 120
5.5 习题 121

第 6 章 常用控件技术 122

6.1 按钮控件 122
6.1.1 命令按钮 122
6.1.2 图形按钮 123
6.1.3 设计带滚动条的图形控件 123
6.2 文本显示与编辑控件 126
6.2.1 静态文本框 126
6.2.2 单行编辑框 127
6.2.3 多行编辑框 128
6.2.4 掩码编辑框 129
6.2.5 超文本编辑框 129
6.2.6 设计记事本程序 131
6.3 选项控件 134
6.3.1 单选按钮 135
6.3.2 复选框 135
6.3.3 组合框 136
6.4 列表选择控件 136
6.4.1 列表框 137
6.4.2 图片列表框 138
6.4.3 下拉列表框 139
6.4.4 下拉图片列表框 140
6.5 图像显示控件 141
6.6 树形视图控件 143
6.6.1 属性 143
6.6.2 添加与删除列表项 144
6.6.3 事件 148
6.6.4 使用树形视图控件显示数据 149
6.7 使用标签控件 150
6.7.1 相关概念 150
6.7.2 建立标签控件 151
6.7.3 属性 153
6.7.4 事件 157
6.8 OLE 控件 157
6.8.1 放入 OLE 控件 157
6.8.2 设置 OLE 控件 159
6.8.3 激活 OLE 对象的方法 160

6.8.4 使用 OLE 访问 Excel 数据 161
6.9 习题 164
第 7 章 菜单技术 165
7.1 菜单概述 165
7.2 菜单操作 166
7.2.1 创建新菜单 166
7.2.2 修改菜单 167
7.2.3 设置菜单属性 169
7.3 菜单事件 170
7.3.1 菜单事件触发时机 171
7.3.2 编写菜单事件处理程序 171
7.4 菜单设计实例 171
7.4.1 制作弹出式菜单 171
7.4.2 为主窗口配置菜单 172
7.4.3 设计图形菜单 175
7.5 习题 177
第 8 章 PowerBuilder 数据库应用基础 178
8.1 数据库的基本概念 178
8.2 PowerBulider 的数据库接口 179
8.2.1 标准数据库接口 180
8.2.2 专用数据库接口 181
8.3 数据库画板 181
8.3.1 数据库画板窗口 181
8.3.2 数据库画板功能 181
8.3.3 修改数据库选项 182
8.4 创建和删除数据库 183
8.4.1 创建数据库 183
8.4.2 删除数据库 184
8.5 连接数据库 184
8.5.1 数据库连接简介 184
8.5.2 使用事务对象 185
8.5.3 创建学生数据库 190
8.6 操作数据库表 193
8.6.1 表的创建与删除 193
8.6.2 定义表的属性、列的属性和扩展属性 194
8.6.3 表的索引 196
8.6.4 表的主键、外键 197
8.6.5 操作数据 198
8.7 实例：创建、配置并连接后台数据库 199
8.7.1 选择数据库管理系统 199
8.7.2 数据库设计 200
8.7.3 配置数据库连接 200
8.8 习题 204
第 9 章 数据窗口技术 205
9.1 数据窗口概述 205
9.1.1 一个简单的数据窗口设计示例 205
9.1.2 数据窗口分类 208
9.1.3 数据窗口画板 211
9.1.4 数据窗口数据源 215
9.2 数据窗口对象操作 218
9.2.1 划分区域 218
9.2.2 添加和删除列 219
9.2.3 计算字段 219
9.2.4 过滤数据 221
9.2.5 数据排序 222
9.2.6 指定检索数据方式 222
9.3 数据窗口控件的属性、事件和函数 223
9.3.1 数据窗口控件的属性 223
9.3.2 数据窗口控件的事件 223
9.3.3 数据窗口控件的常用函数 226
9.4 数据窗口设计实例 229
9.4.1 增加数据窗口数据 229
9.4.2 按学号排序 233
9.4.3 数据窗口排序 234
9.4.4 利用 DataStore 共享数据 235
9.4.5 关联与条件查询 237
9.4.6 数据查询 239

9.4.7 模糊查询 240
9.5 习题 242
第 10 章 数据管道技术 243
10.1 数据管道概述 243
10.2 创建对象 243
10.2.1 创建管道对象 244
10.2.2 创建支撑用户对象 245
10.2.3 创建窗口 246
10.3 数据管道操作 246
10.3.1 初始化操作 246
10.3.2 启动和终止数据管道 247
10.3.3 处理行错误 249
10.3.4 结束操作 251
10.3.5 数据库之间的数据传递 251
10.4 数据管道设计实例 254
10.4.1 创建数据管道 254
10.4.2 数据管道程序设计 256
10.4.3 系统集成 259
10.5 习题 261
第 11 章 用户对象和用户事件技术 263
11.1 用户对象概述 263
11.1.1 面向对象编程技术 263
11.1.2 用户对象的分类 264
11.2 创建用户对象 267
11.2.1 创建标准可视用户对象 267
11.2.2 创建定制可视用户对象 268
11.2.3 创建外部可视用户对象 269
11.2.4 创建定制类用户对象 270
11.2.5 创建标准类用户对象 271
11.2.6 继承创建 272
11.3 定制用户对象 273
11.3.1 插入不可视用户对象 273
11.3.2 插入结构 275
11.3.3 插入函数 276
11.3.4 插入事件 278
11.4 使用用户对象 279
11.4.1 使用可视用户对象 279
11.4.2 使用类用户对象 281
11.5 窗口与用户对象间的通信 285
11.5.1 直接引用用户对象属性 285
11.5.2 使用函数 286
11.5.3 使用用户事件 287
11.6 制作一个通用对话框实例 288
11.6.1 制作通用对话框 288
11.6.2 设计计算器 297
11.7 习题 301
第 12 章 管理库文件 302
12.1 库文件概述 302
12.1.1 库文件 302
12.1.2 库画板 303
12.2 管理 PBL 库 306
12.2.1 创建和删除库 306
12.2.2 创建动态库 307
12.2.3 优化库文件 308
12.2.4 设置搜索路径 308
12.2.5 重建与移植库 310
12.3 操作库与维护库及其对象 310
12.3.1 打开和预览对象 310
12.3.2 复制、移动、粘贴和删除对象 311
12.3.3 搜索目标程序、库和对象 312
12.3.4 导入和导出对象 313
12.3.5 生成库内容报告 314
12.4 根据已有的库文件创建目标 315
12.5 习题 317
第 13 章 程序调试与发布 318
13.1 调试器概述 318
13.2 断点操作 319
13.2.1 调试应用程序的切换 319
13.2.2 设置断点 320
13.3 调试方法 321

13.3.1 调试的开始、继续和停止 322
13.3.2 其他调试方法....322
13.4 查看信息 323
13.4.1 查看各种变量值....323
13.4.2 查看其他信息....323
13.4.3 运行错误解释....324
13.5 发布应用程序....326
13.5.1 概述 327
13.5.2 建立工程对象....327
13.5.3 编译生成各类对象....328
13.5.4 编译生成可执行应用....329
13.6 制作安装盘 335
13.6.1 找出软件发布的所有必需文件 335
13.6.2 确保软件的目录结构同软件一致....335
13.6.3 在注册表里的设置....335
13.6.4 确定要生成的程序包的名字和目标....336
13.7 习题 336

第 14 章 分布式体系....337

14.1 分布式体系结构概述....337
14.1.1 分布式计算....337
14.1.2 远程对象 339
14.1.3 应用服务器....339
14.2 分布式体系应用....340
14.2.1 建立客户端通信....342
14.2.2 控制用户登录应用服务器 343
14.2.3 创建分布式 PowerBuilder 服务器 344
14.3 分布式体系设计实例....347
14.3.1 创建 EAServer 组件 347
14.3.2 发布客户端应用....351
14.4 习题 357

第 15 章 建立 Jaguar 组件和客户 358

15.1 Jaguar 组件概述 358
15.1.1 管理程序变量空间 359
15.1.2 定义组件接口 359
15.2 存取数据库 361
15.2.1 概述 361
15.2.2 使用连接缓存 362
15.2.3 提供事务支持 363
15.2.4 执行检索操作 366
15.2.5 执行更新操作 367
15.2.6 传递结果集 368
15.2.7 支持实例缓冲池 369
15.3 使用共享组件 371
15.3.1 创建一个共享组件 371
15.3.2 创建一个服务组件 372
15.3.3 调用另一个服务器组件的方法 372
15.3.4 访问组件属性 373
15.4 组件测试与调试 375
15.5 向 Jaguar 分发组件 376
15.6 Jaguar 客户概述 377
15.7 与 Jaguar 服务器连接 377
15.7.1 编写代码 377
15.7.2 使用向导 378
15.8 生成 Jaguar 代理对象 379
15.9 调用组件方法 379
15.9.1 调用方法 379
15.9.2 清除代理对象实例 380
15.9.3 请求服务器返回消息 381
15.10 处理通信错误 382
15.10.1 概述 382
15.10.2 连接对象 Error 事件脚本 383
15.11 分发客户应用 384
15.12 习题 384

第 1 章　PowerBuilder 10.0 概述

本章首先为读者介绍 PowerBuilder 的发展历史及 PowerBuilder 的安装和启动，然后介绍 PowerBuilder 的集成开发环境及如何在 PowerBuilder 开发环境中添加、设计和使用窗口和菜单，最后给出一个简单的 PowerBuilder 数据库应用程序，使读者对使用 PowerBuilder 开发数据库应用程序有一个初步的了解。通过本章的学习，读者可以初步了解 PowerBuilder 的特点，能安装和运行 PowerBuilder，并且可以建立一个简单的数据库应用程序。

1.1　PowerBuilder 10.0 简介

PowerBuilder 是由著名的数据库应用开发工具厂商 Sybase 公司推出的产品，于 1991 年 6 月正式投入市场。它最初是按照客户机/服务器体系结构、采用面向对象技术、图形化的应用开发环境进行设计，是数据库的前端开发工具。PowerBuilder 一经投入市场，就以其独特的体系结构，友好的界面，优异的性能受到了广大开发人员的欢迎。到 1997 年，根据 Meta Group 公司统计，PowerBuilder 已占有全球客户机/服务器前端开发市场份额的 40%，雄踞第一。

对于选择 PowerBuilder 作为开发工具的用户来说，之所以会选择 PowerBuilder，是因为 PowerBuilder 具有以下 6 大特点。

(1) PowerBuilder 提供了对目前流行的大多数关系数据库管理系统的支持。

(2) 和大多数的 Windows 应用程序一样，PowerBuilder 也是事件驱动工作方式。

(3) PowerBuilder 是一种面向对象的开发工具，各种 Windows 应用程序中常见的窗口、菜单、控件等在 PowerBuilder 中都是独立的对象。

(4) PowerBuilder 提供了良好的跨平台性，在 PowerBuilder 中，利用 Windows 平台开发的各种对象可以方便地应用到 UNIX 平台中。

(5) PowerBuilder 具有自己的编程语言 PowerScript，这一语言除了提供基本的流程控制语句外，还提供了几百个操纵各种对象的函数来支持诸如 DDE，OLE 等技术。

(6) PowerBuilder 最具特色的一点就是提出了数据窗口对象的概念。数据窗口对象也是 PowerBuilder 中的一种对象类型，与其他对象不同的是数据窗口对象是专门为访问后台数据库服务的。在数据窗口对象中定义了数据的来源和数据的显示风格，这样在应用程序中就可以把精力完全放在程序的运行流程控制上，而不用关心具体数据的来源，因为在数据窗口对象中已经定义好了数据的来源。如果需要使用数据库中不同的数据，也只要对数据窗口对象进行修改就可以了。特别要指出的是 PowerBuilder 在数据窗口对象中提供了丰富的数据显示方式，可以满足各种不同的需要。

PowerBuilder 可以做的事情很多，因此也就成为现在各软件公司进行系统开发的一种常用工具，如果你将从事下列工作或有这些需求，则需要了解 PowerBuilder。

(1) 数据库系统前端开发中的程序设计。

(2) 开发咨询或技术支持。
(3) 数据库系统分析及设计。
(4) 需要掌握 Visual Basic，Delphi，Developer 2000 以外的其他工具。
(5) 了解 Internet 网站建设中可以保护现有 PowerBuilder 资源的开发方式。

1.2 安装和启动 PowerBuilder 10.0

1.2.1 安装 PowerBuilder 10.0

安装 PowerBuilder 10.0 系统软件，可以选择最小安装、典型安装和用户自定义安装三种方式，不同安装需要的硬盘空间不同。下面介绍的是典型安装。

PowerBuilder 10.0 系统软件的安装过程与大多数 Windows 应用软件一样。不同的是需要安装 PowerBuilder 10.0 许多相关组件。下面先介绍 PowerBuilder 10.0 的这些组件，PowerBuilder 10.0 中有以下组件：

- PowerBuilder 10.0。
- InfoMaker：是快速原型开发工具。
- PowerDynamo：是动态页面服务器。
- Jaguar CTS：是事务处理服务器。
- Sybase Online Books。
- Adaptive Server Anywhere：是随 PowerBuilder 提供的一个完整的客户/服务器数据库管理系统。

本小节介绍在中文 Windows 环境下 PowerBuilder 10.0 的安装过程。具体步骤如下：

(1) 将 PowerBuilder 10.0 的安装光盘放入光盘驱动器，打开光盘，双击 Setup 图标，弹出 Welcome 对话框，如图 1-1 所示。

(2) 单击图 1-1 中的 Next 按钮，弹出如图 1-2 所示的安装协议，用户应该认真阅读该协议。

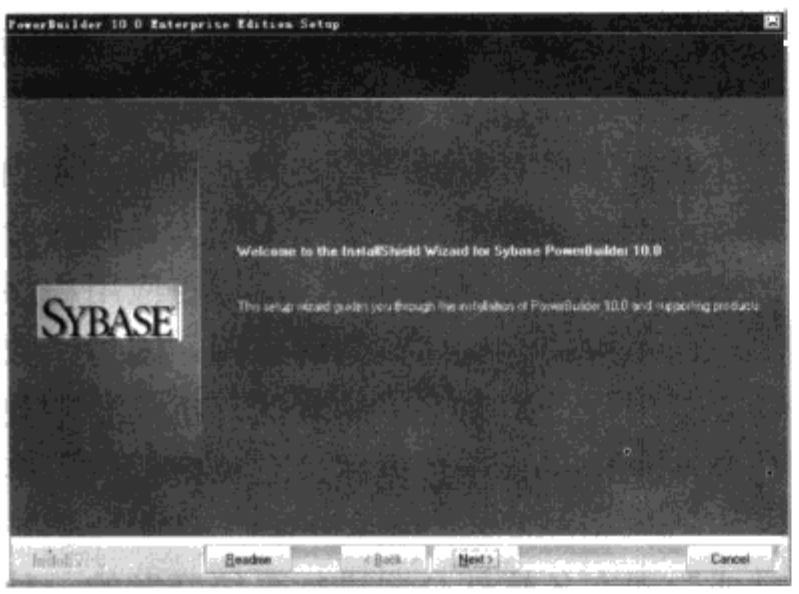


图 1-1 Welcome 对话框

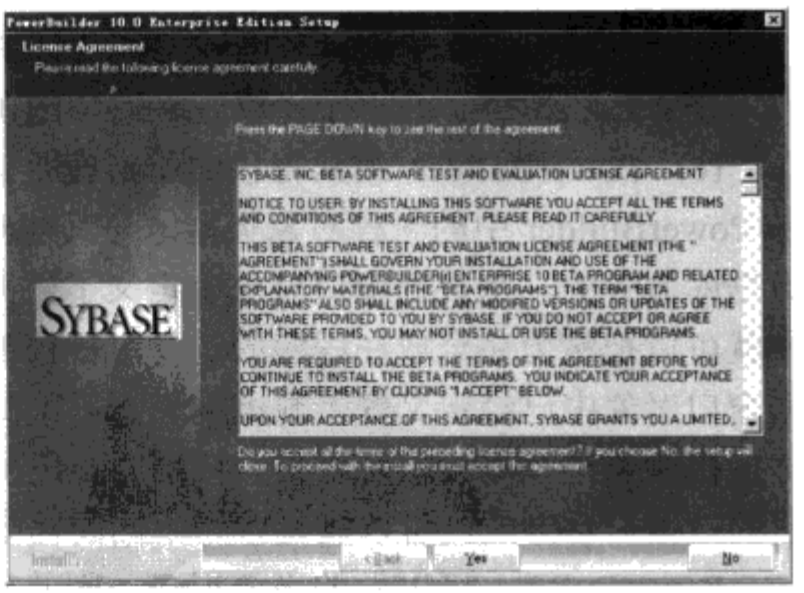


图 1-2 安装协议窗口

(3) 单击图 1-2 中的 Yes 按钮，弹出如图 1-3 所示的用户名称和公司名称输入界面。

(4) 单击 Next 按钮，进入如图 1-4 所示的安装目录窗口，如果要改变安装目录，则单击 Browse 按钮选择自己想要的安装目录。

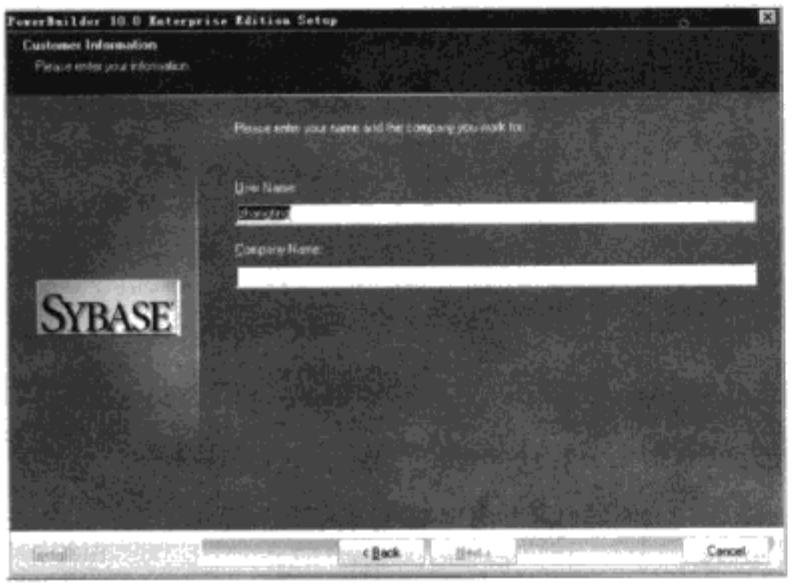


图 1-3　用户公司名称输入界面

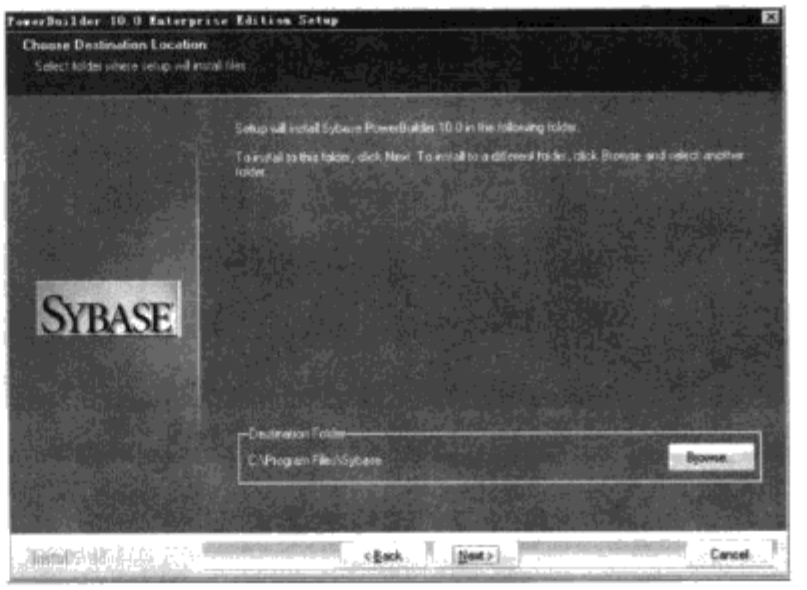


图 1-4　选择安装目录

(5) 单击 Next 按钮，进入如图 1-5 所示的共享文件安装目录窗口，如果要改变安装目录，则单击 Browse 按钮选择自己想要的安装目录。

(6) 在组件选择窗口中，用户可以根据需要选择组件，这里只选择了 PowerBuilder 10.0 系统软件，如图 1-6 所示。

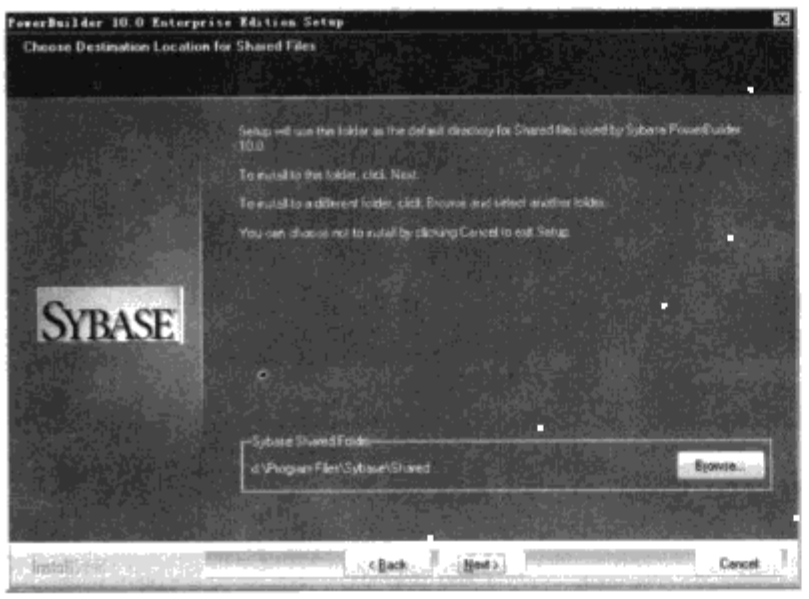


图 1-5　选择共享文件安装目录

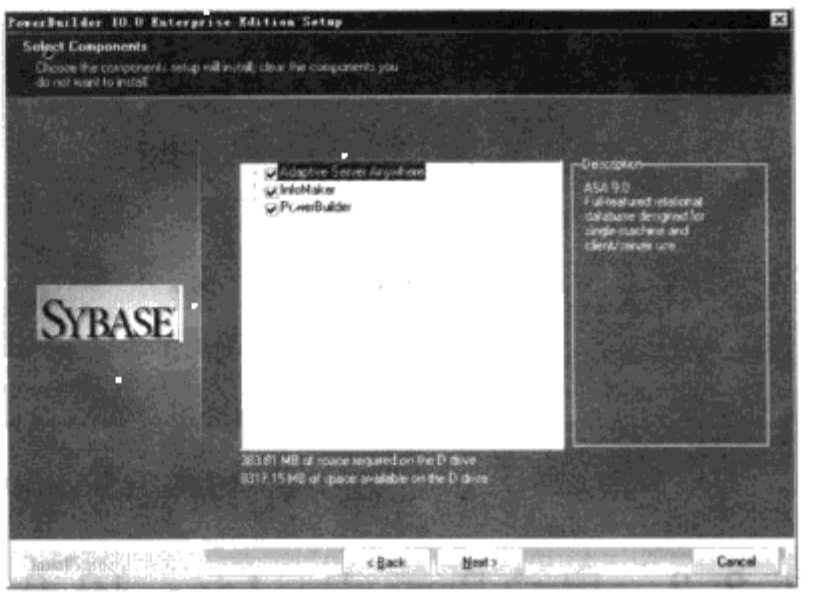


图 1-6　PowerBuilder 10.0 部件选择窗口

(7) 在类型选择窗口，选取 Typical 选项将按典型方式安装，如图 1-7 所示。

(8) 下面几步选择默认设置，然后进入 PowerBuilder 软件的安装程序窗口，如图 1-8 所示。

(9) 安装完成后，出现 InstallShield Wizard Complete 对话框，如图 1-9 所示。

(10) 安装结束后，重新启动计算机，完成 PowerBuilder 10.0 的安装。

提示： 如果在如图 1-6 所示的组件选择窗口中选择其他组件，则安装向导将会继续安装其他组件。其过程和安装 PowerBuilder 10.0 类似，这里不再一一赘述。

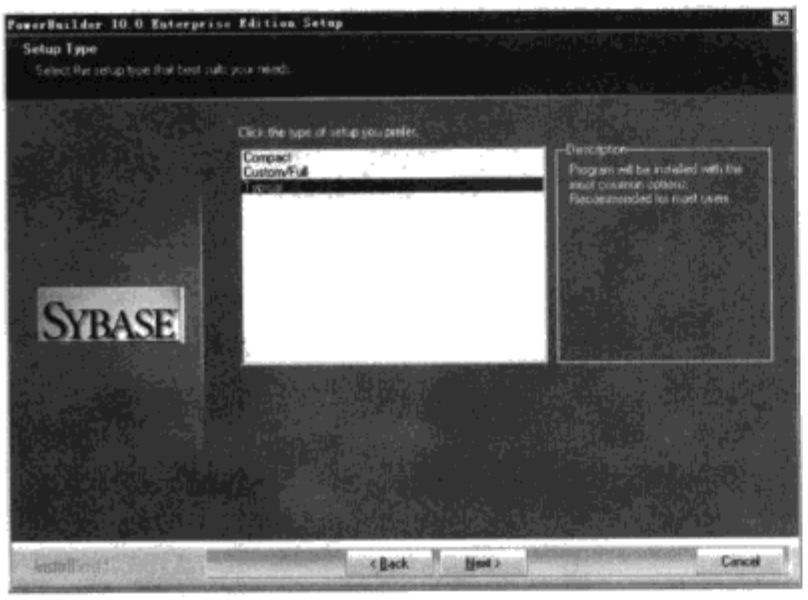


图 1-7　选择安装类型

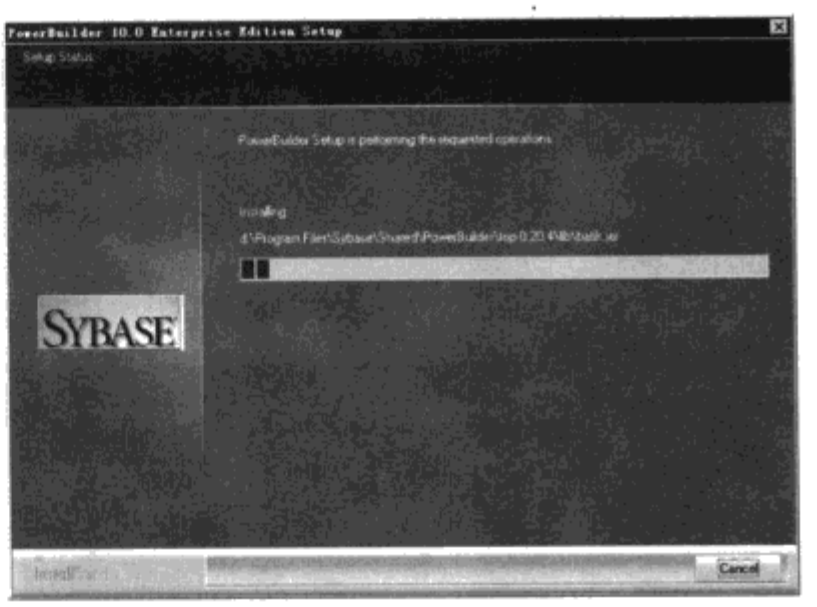


图 1-8　安装程序窗口

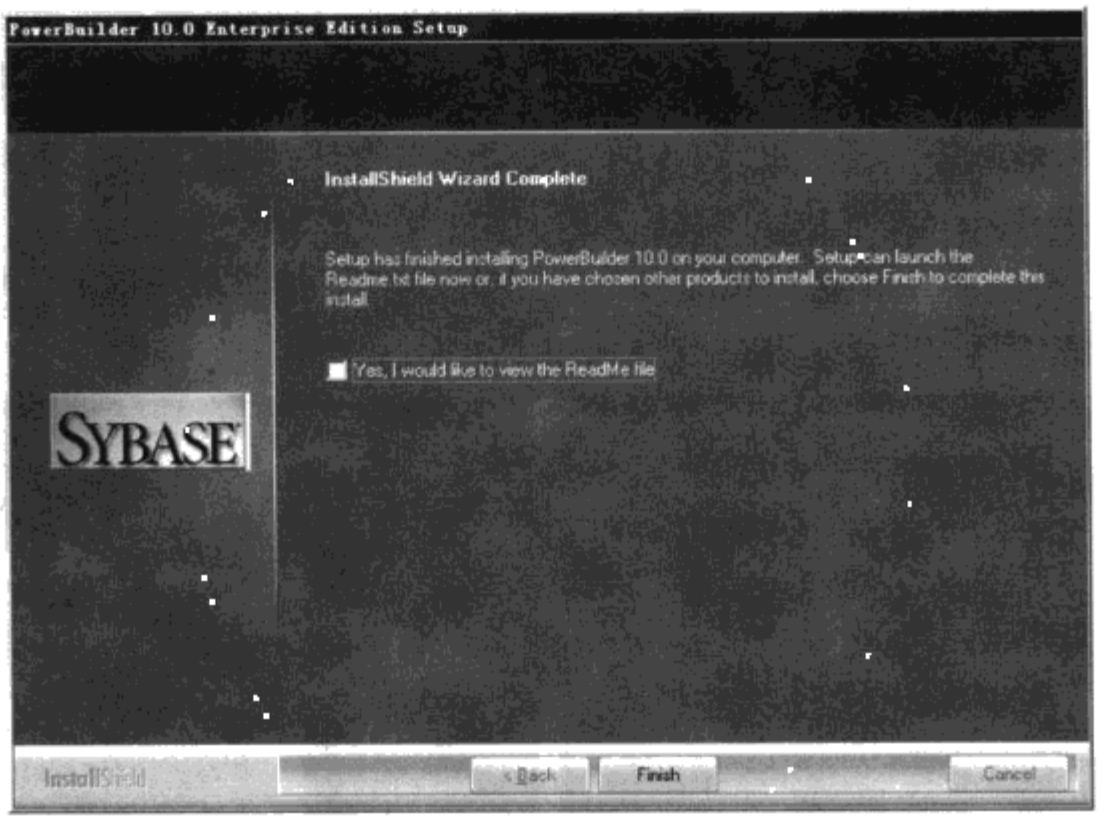


图 1-9　InstallShield Wizard Complete 对话框

1.2.2　启动 PowerBuilder 10.0

安装完毕后，就可以启动 PowerBuilder 10.0 了，启动的步骤如下：

(1) 单击任务栏的“开始”菜单命令。

(2) 选择“程序”中的 PowerBuilder 10.0 程序组，然后选择 PowerBuilder 10.0 菜单命令，启动 PowerBuilder 10.0 应用程序。也可以在资源管理器的 PowerBuilder 10.0 安装目录下运行 pb10.exe 或建立 PowerBuilder 10.0 的快捷方式，然后双击快捷方式启动 PowerBuilder 10.0 应用程序。

(3) 第一次启动 PowerBuilder 时，会出现“Welcome to PowerBuilder”对话框，如图 1-10 所示，共有 5 个选项，两个复选框，用户可以根据需要选择不同的选项。如果启动 PowerBuilder 时不想显示“Welcome to PowerBuilder”对话框，选中“don’t show this dialog again”复选框，那么下次启动 PowerBuilder 时，就不会出现这个对话框。

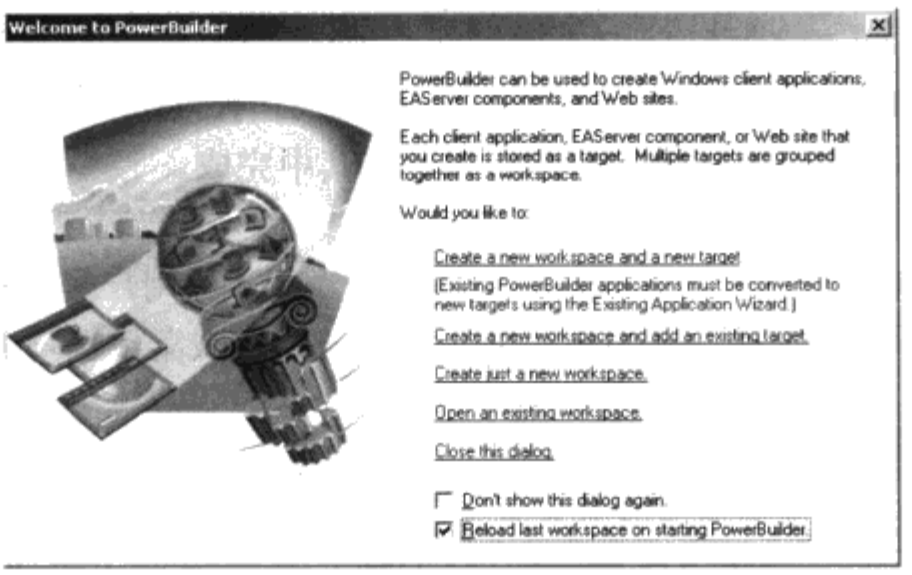


图 1-10　Welcome to PowerBuilder 对话框

1.3　PowerBuilder 10.0 开发环境

PowerBuilder 10.0 的开发环境主要分为菜单系统、PowerBar 工具栏、工作区以及状态栏 4 个部分，如图 1-11 所示。

(1)　菜单系统

菜单系统位于 PowerBuilder 10.0 开发环境的顶部，包含 5 个菜单：File 菜单、Run 菜单、Tools 菜单、Window 菜单和 Help 菜单。其中 File 菜单用于处理与文件相关的操作；在 Run 菜单中，可以对应用程序进行编译和调试；在 Tools 菜单中，可以设置 PowerBuilder 10.0 的各种选项以及使用多种工具；在 Window 菜单中，可以控制窗口在屏幕上的位置；Help 菜单提供了全面的帮助信息。

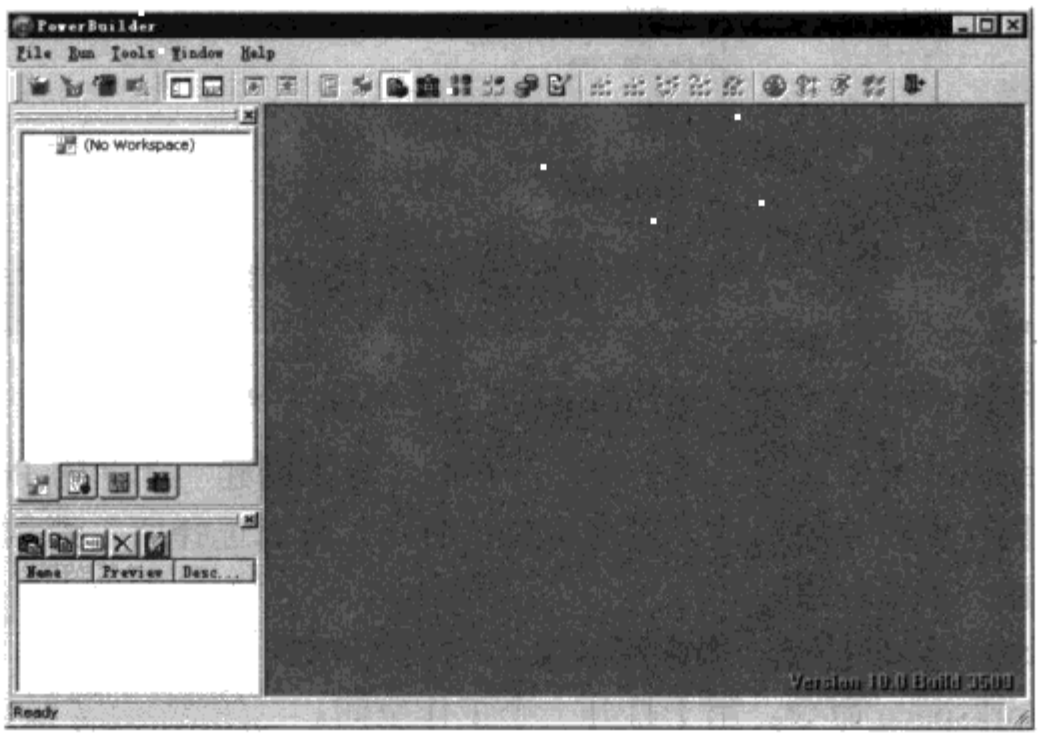


图 1-11　PowerBuilder 10.0 的开发环境

(2)　PowerBar 工具栏

PowerBar 工具栏位于 PowerBuilder 10.0 菜单系统的下面，其中每一个工具按钮都对应一个特定的功能。

(3) 工作区

PowerBuilder 10.0 开发环境窗口的中间部分是空白的，这是 PowerBuilder 10.0 的工作区，PowerBuilder 10.0 的各种画板执行的操作都在其中完成。

(4) 状态栏

状态栏位于 PowerBuilder 10.0 开发环境窗口的底部，为用户提供重要的提示。例如将鼠标停留在某个菜单选项上时，状态栏将显示关于该按钮或菜单选项的详细的文本说明。

1.3.1 菜单系统

菜单是应用程序的主要工具，用户通过菜单执行操作命令，以便完成指定任务(如打开文件和保存文件等)。

除了菜单系统包含的五个子菜单，如果打开不同的画板，则出现不同的菜单。单击某个菜单标签，将显示下拉菜单中的各个命令。单击某个命令，即完成相应操作。另外，还可以使用快捷键执行某个命令。下面将介绍一些菜单命令。

1. File 菜单

File 菜单如图 1-12 所示，其中包含操作文件的命令。

(1) New 命令。用于创建一个新的 PowerBuilder 10.0 对象，包括应用程序、用户对象、构件对象以及窗口、数据窗口等，如图 1-13 所示。

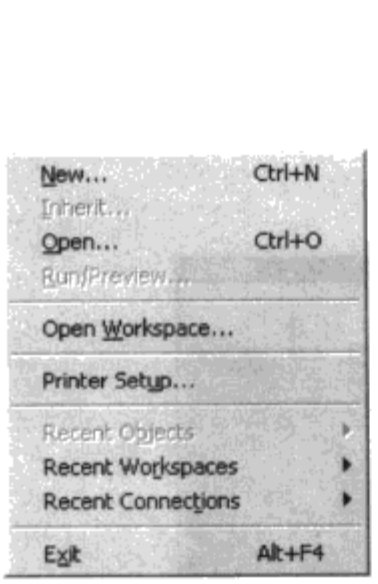


图 1-12 File 菜单

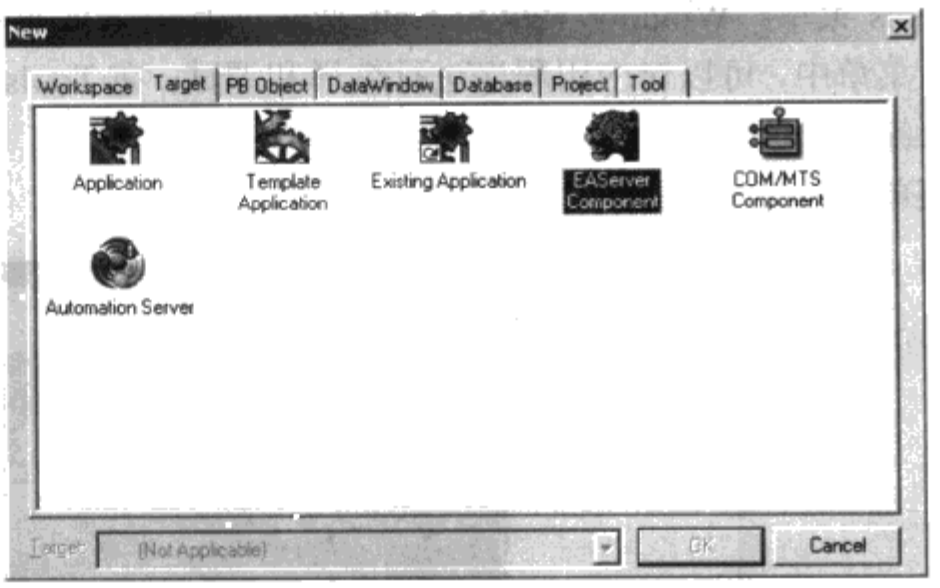


图 1-13 New 对话框

(2) Inherit 命令。用于通过继承一个已存在的对象来创建一个新的 PowerBuilder 10.0 对象，该对象将拥有原对象的属性。通过该菜单命令，可以选择所需继承对象的类型：窗口对象、用户对象、菜单对象。

(3) Open 命令。用于打开一个应用对象。

(4) Open Workspace 命令。用于打开一个工作空间。

(5) Recent Workspace 子菜单。从 Recent Workspace 子菜单可以知道当前有哪些工作空间，并且可以单击其中的命令来打开哪个工作空间。

(6) Recent Connections 子菜单。从 Recent Connections 子菜单可以知道当前有哪些数据库连接，并且可以单击其中的命令来建立一个数据库连接。

2. Run 菜单

Run 菜单如图 1-14 所示，可以对应用程序进行调试和编译。

(1) Incremental Build Workspace 命令。对工作空间中修改的对象进行编译。

(2) Full Build Workspace 命令。对工作空间全部对象进行编译。

(3) Debug 命令。单击此按钮将进入应用程序调试窗口。

(4) Run 命令。单击此按钮将开始运行应用程序。

3. Tools 菜单

在 Tools 菜单中，用户可以设置系统选项。此外，Tools 菜单还提供了多个工具，如图 1-15 所示。

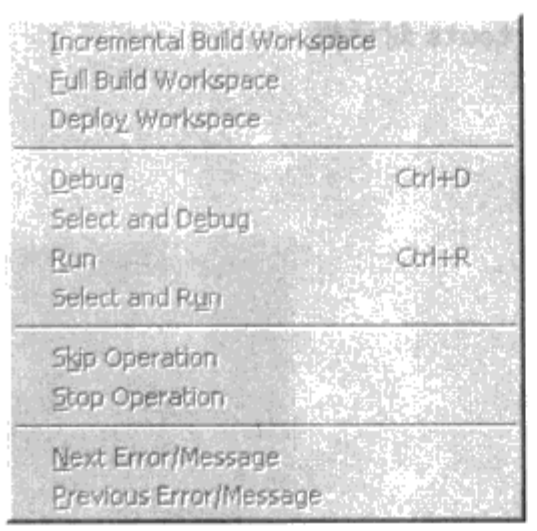


图 1-14　Run 菜单

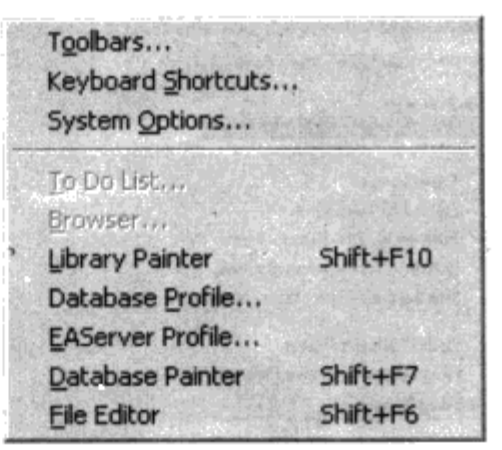


图 1-15　Tools 菜单

(1) Toolbars 命令。用于设置工具栏。选择该命令后，将会打开 Toolbars 对话框，如图 1-16 所示。在此对话框中可以设置工具栏的位置和字体等，还可以自定义工具栏。

(2) Keyboard Shortcuts 命令。用于编辑键盘快捷键。选择该命令后，将打开 Keyboard Shortcuts 对话框，如图 1-17 所示。在此对话框中，可以设置和编辑 PowerBuilder 10.0 菜单系统中每个菜单命令的快捷键。

(3) System Options 命令。用于设置或编辑系统选项。选择该命令后，可打开 System Options 对话框，如图 1-18 所示。在此对话框中，可以设置 PowerBuilder 10.0 的一些系统选项，这些选项将在下一次打开 PowerBuilder 10.0 时生效，如初始化路径、字体属性、日志文件等。

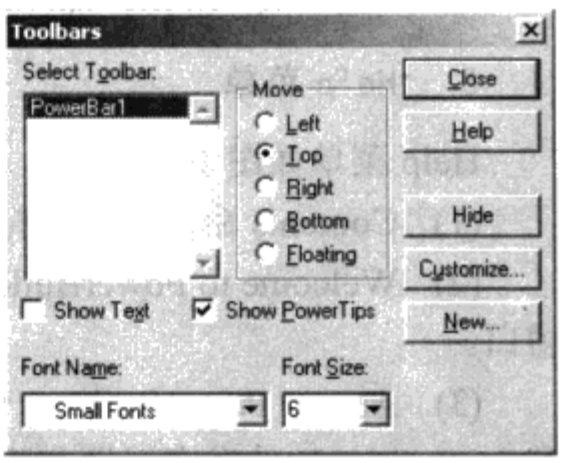


图 1-16　Toolbars 对话框

4. Window 菜单

Window 菜单如图 1-19 所示，提供了窗口在屏幕上位置的控制。

(1) System Tree 命令、Output 命令和 Clip 命令。用来打开或关闭各自的窗口。

(2) Close 命令。关闭画板上的所有窗口。

(3) 其他命令。用来控制窗口的显示方式。

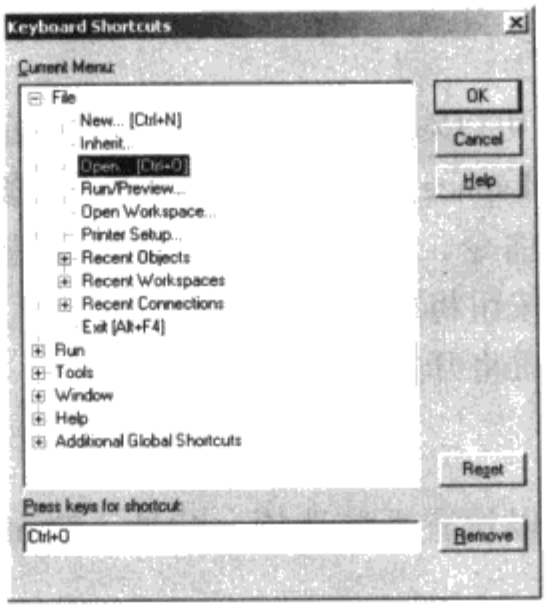


图 1-17 Keyboard Shortcuts 对话框

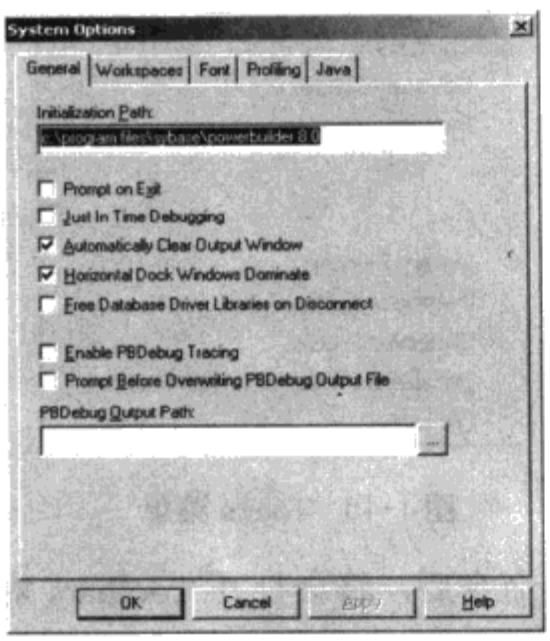


图 1-18 System Options 对话框

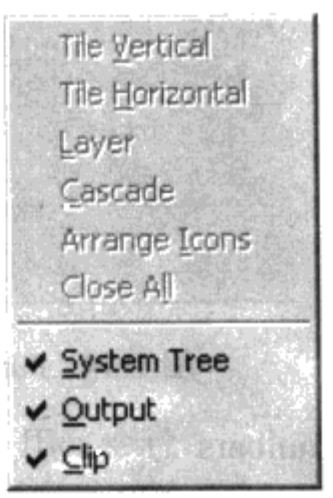


图 1-19 Window 菜单

5. Help 菜单

Help 菜单如图 1-20 所示，提供了帮助信息。

(1) Contents 命令。用来打开帮助目录窗口。

(2) Welcome to PowerBuilder 10.0 命令。用来打开帮助信息窗口。

(3) About PowerBuilder 命令。用于说明 Sybase 公司 PowerBuilder 软件发布日期和版权等内容。

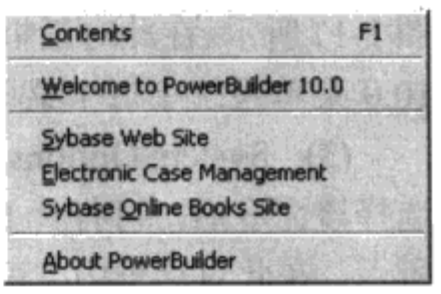


图 1-20 Help 菜单

1.3.2 工作区

用来设计数据库、用户窗口、数据窗口以及各种组件等的区域，如图 1-11 灰色部分所示。

1.3.3 工具栏

PowerBuilder 10.0 工具栏中的按钮对应常用的操作命令，如图 1-21 所示。用户可以根据需要增加自己的工具栏。

图 1-21　PowerBuilder 10.0 工具栏

各主要按钮作用如下：

- New：启动 Wizards(向导)工具创建新的应用对象、工程文件。
- Inherit：用继承的方式创建一个新的对象。
- Open：打开一个已经存在的对象。
- Run/Preview Object：运行或预览一个对象。
- System Tree：给出 Workspace(工作空间)的信息。
- Output：弹出 Output(输出)窗口，在窗口给出不同操作的信息。
- Next Error/Message：调试程序时显示下一条错误信息。
- Previous Error/Message：调试程序时显示上一条错误信息。。
- To-Do List：显示 To-Do List 列表，跟踪开发任务，PowerBuilder 10.0 能自动增加列表和项目，快速进入想要找的画板和对象。
- Browser：打开浏览对话框。
- Clip Window：在此窗口中可以把经常使用的代码保存到适当的文件夹中。
- Library：单击此按钮将打开应用程序的库，在此界面中可以管理应用程序的各种对象。
- DB Profile：新建或打开一个数据库描述。
- EAServer Profile：打开一个对话框，可以定义一个特定数据库的连接参数。
- Database：单击此按钮将新建或打开一个数据库。
- Edit：单击此按钮将创建和打开一个文件并编辑文件。
- Incremental Build Workspace：参看“Run 菜单”。
- Full Build Workspace：参看“Run 菜单”。
- Deploy Workspace：部署 Workspace(工作空间)。
- Skip Build/Deploy/Search：用于跳过应用的编译、配置和搜索。
- Stop Build/Deploy/Search：用于中止应用的编译、配置和搜索。
- Run：运行对象。
- Debug：调试对象。
- Select and Debug：选择对象进行调试。
- Select and Run：选择对象进行运行。

- Exit：单击此按钮将退出 PowerBuilder 10.0 系统。

1.3.4 画板

画板是根据英文 Painter 翻译过来的，它的实际含义是工具或工具箱，或者可以把它看作是 PowerBuilder 10.0 大环境下的一个小环境。PowerBuilder 10.0 使用画板来建立应用程序的各个组件，PowerBuilder 10.0 为每一类对象提供了一个画板。例如用户可在 Window 画板中建立窗口，为窗口定义属性和添加控件等，如图 1-22 所示。

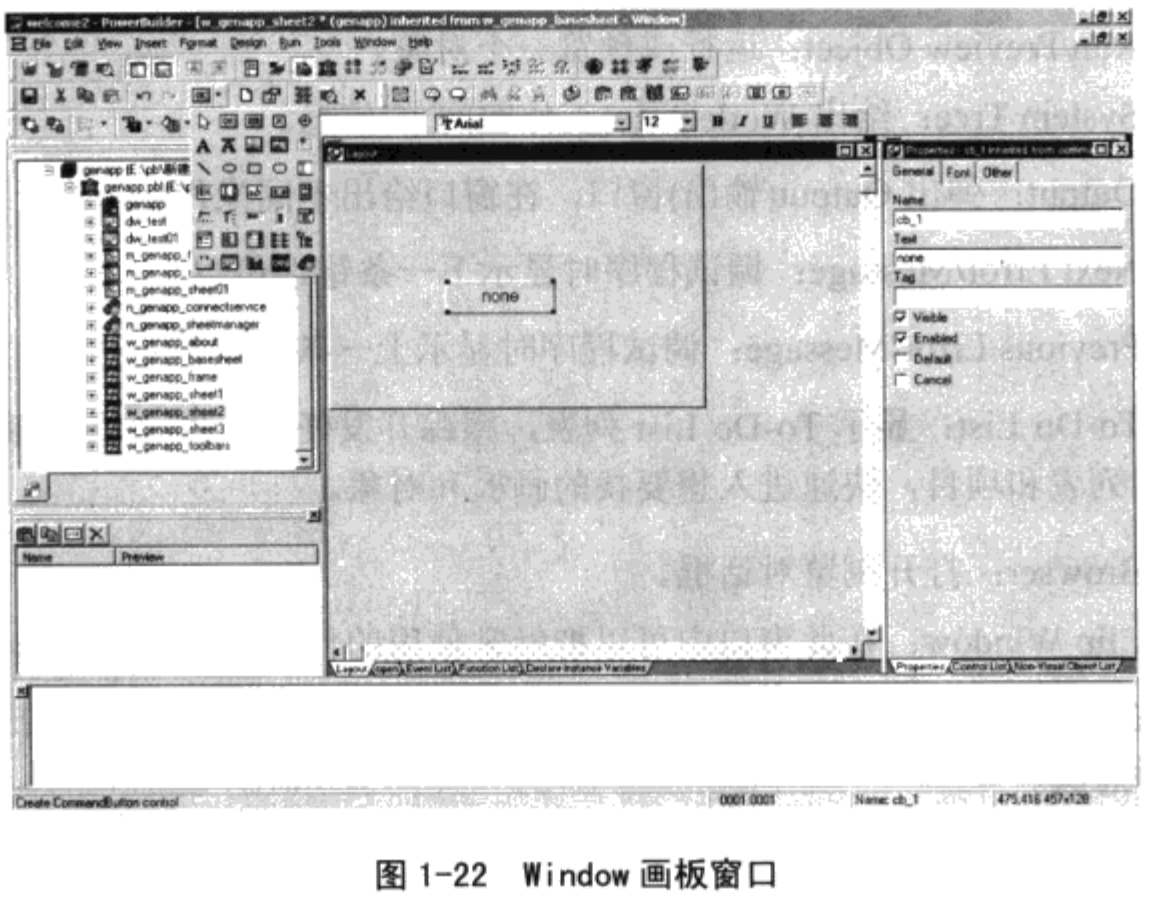


图 1-22 Window 画板窗口

1.3.5 帮助系统

在实际开发 PowerBuilder 10.0 应用程序的过程中，可以通过 PowerBuilder 10.0 的联机文档系统得到帮助。

在 PowerBuilder 10.0 集成环境中选择 Help 菜单下的 Contents 命令，即可进入 PowerBuilder Help 窗口，如图 1-23 所示，窗口包括了两个页面，“目录”页面和“索引”页面。

定位 PowerBuilder 10.0 帮助主题有以下几种方式：

通过目录定位：选择“目录”页面，双击某个节点，就可以展开该节点，然后单击要查看的内容，如图 1-23 所示。

通过“索引”查找信息：在“索引”页面中输入或选择一个与所需查找的信息有关的关键字，然后单击“显示”按钮，会弹出一个查获的主题列表窗口，从中选择所需的主题，然后单击“显示”按钮，可以得到相关内容的文档，如图 1-24 所示。

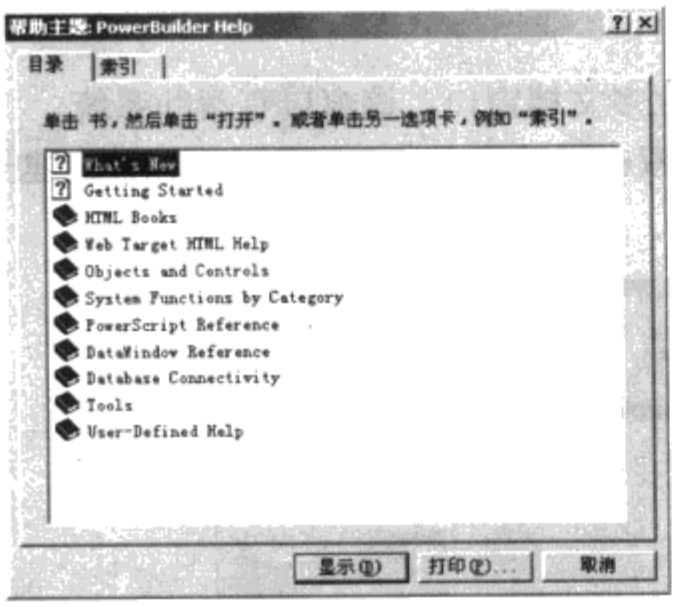


图 1-23　PowerBuilder 10.0 帮助目录窗口

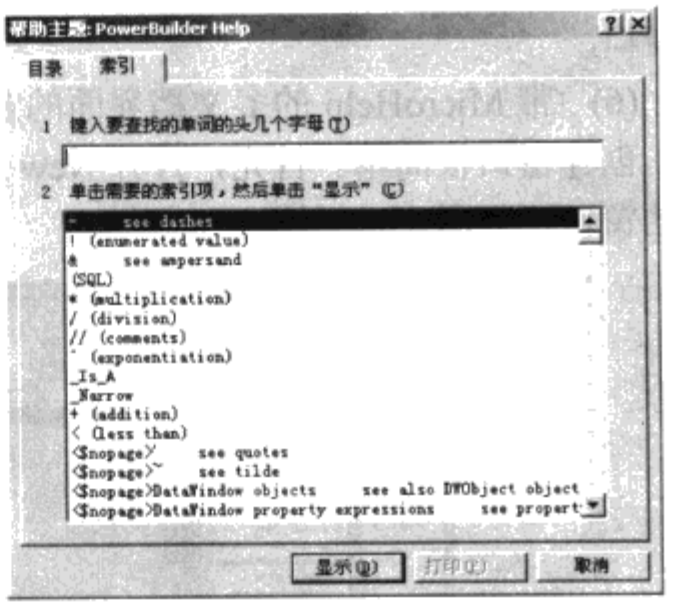


图 1-24　PowerBuilder 10.0 帮助索引窗口

1.4　窗口和菜单简介

窗口和菜单是用户和 PowerBuilder 10.0 应用程序联系的主要接口。可以通过菜单接收用户要求，通过窗口向用户显示信息和从用户那里得到信息。PowerBuilder 10.0 给出了强大的菜单和窗口功能，可以通过简单的操作得到美观、实用的用户界面。下面将介绍窗口和菜单的设计方法。

1.4.1　窗口简介

1.　创建窗口

创建窗口的方法有两个，一个是直接打开 New 对话框，双击标签页中的 Window 图标，即可创建一个新的窗口；二是从菜单中选择 Inherit 菜单通过继承已有窗口的格式、事件、函数、结构、变量和程序代码，生成新的窗口。

在 PowerBuilder 10.0 中提供了 6 种类型的窗口，分别是主窗口(Main)、弹出窗口(Popup)、子窗口(Child)、应答窗口(Response)、多文档界面的窗口(MDI Frame)、带 MicroHelp 的多文档界面的窗口。

(1) 主窗口(Main)。主窗口是不依赖于其他窗口的独立窗口，它既可以覆盖别的窗口，也可以被别的窗口覆盖。通常使用主窗口作为应用程序的基本界面。

(2) 弹出窗口(Popup)。弹出窗口通常是在其他窗口中被打开，这个其他窗口从而成为弹出窗口的父窗口。如在响应应用程序的 Open 事件时打开弹出窗口、则该窗口没有父窗口。它的工作方式同主窗口相同。

(3) 子窗口(Child)。子窗口通常在主窗口和弹出窗口中被打开。子窗口只有在它的父窗口里才能存在。可以在父窗口中移动子窗口，但是不能将它移出父窗口。

(4) 应答窗口(Response)。应答窗口用于从用户那里获得信息，它们通常在别的窗口中被打开。一般来说，应答窗口在它的父窗口里发生某些事件时被打开。当应答窗口被打开时，只有当前活动窗口可以被访问，查到用户对该应答窗口做出响应。

(5) 多文档界面的窗口(MDI Frame)。多文档窗口是一种框架窗口，可以同时打开多个文

档窗口。

(6) 带 MicroHelp 的多文档界面的窗口。相比多文档窗口，该窗口有帮助系统。

创建窗口很简单。首先，打开 New 对话框，双击标签页中的 Window 图标，创建一个新的窗口，如图 1-25 所示。

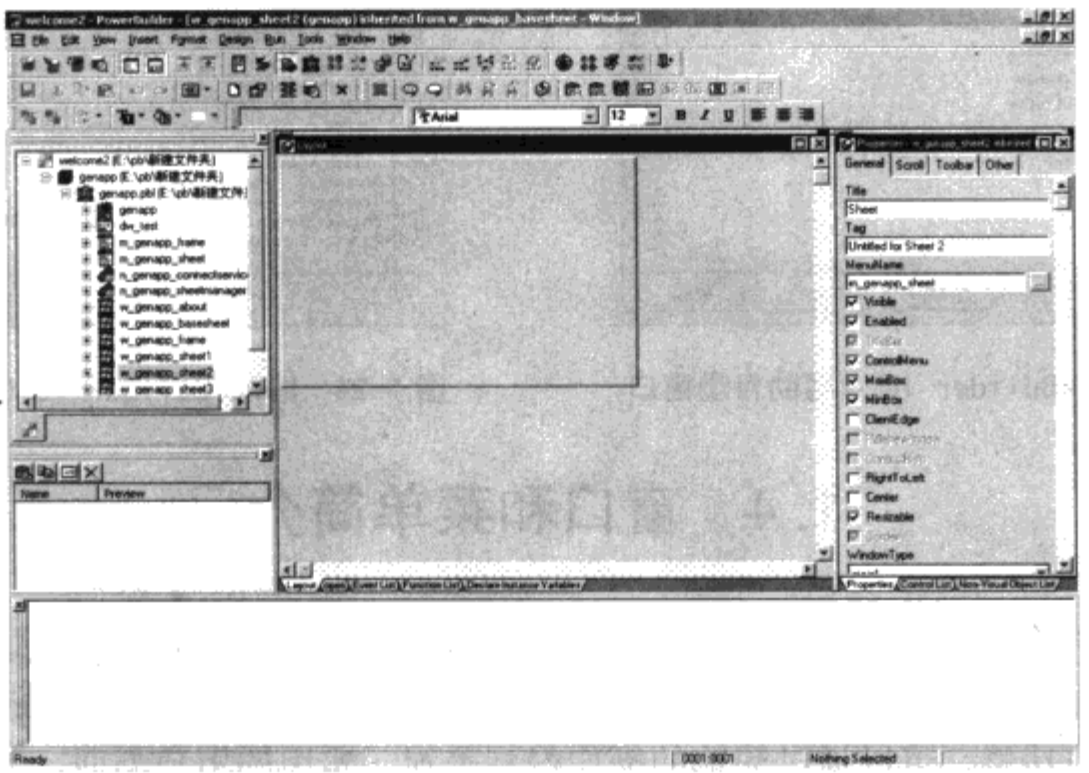


图 1-25 窗口画板

2. 窗口的设计

窗口的定义主要包括属性、事件和函数。窗口的属性决定了窗口的显示风格，定义子窗口的表现形式。事件指当某个事件触发时，与之相关的程序脚本被执行，窗口做出响应的动作。函数能触发窗口事件、处理事务、改变窗口或者提供有关此窗口的信息。下面，分别介绍窗口的主要属性和事件。

(1) 窗口的属性

创建窗口后，定义窗口的属性。选中窗口对象的同时即打开窗口属性对话框，在各个选项卡中赋值即可，如图 1-26 所示。

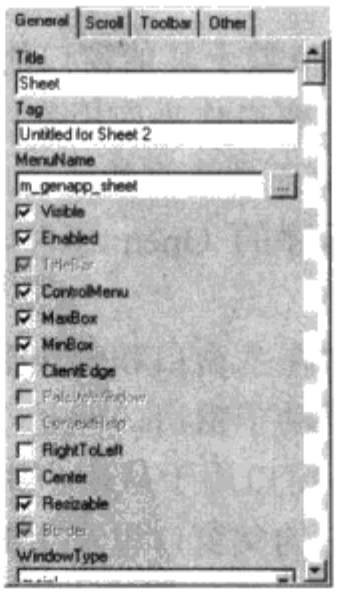


图 1-26 窗口的属性对话框(General 选项卡)

表 1-1 列出了窗口对象的常用属性。

表 1-1　窗口对象的常用属性

属　性	功　能
Title	设置窗口的标题
Tag	设置窗口的标签
MenuName	连接到该窗口的菜单
Title Bar	设定窗口是否含有标题栏，只有在 Border 属性设置为 True 时，才可以对该属性进行设置
Control Menu	当窗口含有标题栏时，是否显示标题栏的控制菜单
Context Help	设置窗口上下文帮助文本
RightToLeft	设置字符按照从右到左的顺序显示
Resizeable	设置窗口大小是否任意放大或缩小
Border	设置窗口是否有边界
WindowType	从下拉列表框中选择窗口的类型
WindowState	设置窗口状态
MDIClient Color	当该窗口为 MDI 窗口和带 MicroHelp 的 MDI 窗口时，选择该窗口在运行时客户区域的颜色
Icon	设置窗口最小化时的图标
HscrollBar	设置横向滚动条
UnitsPerLine	设置用户单击纵向滚动条中的上箭头或下箭头时，向上或向下滚动的距离
ColumnsPerPage	当用户单击横向滚动条时窗口滚动的列数
ToolBarVisible	设置窗口中的工具条是否可见
ToolBarAlignment	设置窗口上的工具条的安放位置
ToolBarX	当选中 Float 属性时，输入工具条悬浮位置的横坐标值
X	设置窗口左上角的横坐标
Width	设置窗口的宽度
Pionter	设置窗口的鼠标形状

(2)　窗口事件

窗口的事件处理程序决定了窗口的动作。当窗口的事件被触发时，便执行为该事件编写的程序。表 1-2 列出了窗口上的事件及其触发时机。

表 1-2　窗口事件及其触发时机

事　件	触 发 时 机
Activate	仅当窗口成为活动的窗口之前
Clicked	当用户单击窗口中的空白区域时
Close	当窗口关闭时
CloseQuery	当关闭窗口时，Power Builder 触发 CloseQuery 事件并检测 Message.ReturnValue 的值。如果 Message.ReturnValue 的值为 1，则窗口不能关闭
Deactivate	当窗口成为不活动的窗口时
DoubleClicked	当用户双击窗口的空白区域时

续表

事　件	触 发 时 机
DragDrop	当在窗口中拖动控件并释放鼠标时
DragEnter	当拖动控件进入窗口时
DragLeave	当拖动控件离开窗口时
DragWithin	当在窗口中拖动控件时
Hide	仅当窗口隐藏时
HotLinkAlarm	当 DDE 服务器应用送来新的数据并且客户 DDE 应用接收到时
Key	当用户按键盘并且插入点不在 RichTextEdit 或数据窗口的编辑控件上时
MouseDown	当用户在窗口的空白区域按下鼠标的左键时
MouseUp	当用户在窗口的空白区域按下鼠标的右键时
Open	当事件处理程序执行窗口的 Open 函数时，事件发生在窗口打开以后但在显示以前
Other	当一个非 Power Builder 事件的消息发生时
RbuttonDown	当用户在窗口的空白区域按下鼠标的右键时
RemoteExecl	当 DDE 客户应用发送一个命令时
RemoteHotLinkStart	当 DDE 客户应用想要建立一个热连接时
RemoteHotLinkStop	当 DDE 客户应用想要结束一个热连接时
RemoteRequest	当 DDE 客户应用要求数据时
RemoteSend	当 DDE 客户应用发送数据时
Resize	当用户事件处理程序打开或改变一个窗口的大小时
Show	当用户事件处理程序执行窗口的显示函数，仅在窗口显示之前时
SystemKey	当插入点不在编辑点且当用户按 ALT 及 ALT 加其他的键时
Timer	当定时函数执行后过了制定的时间后
ToolBarMoved	在 MDI 窗口中移动工具条时

1.4.2　菜单简介

PowerBuilder 10.0 提供了两类菜单，分别是下拉式菜单和弹出式菜单，下拉式菜单在应用窗口打开时随之显示，弹出式菜单一般与鼠标右键操作相对应。下面分别介绍这两种菜单的创建过程。

1.　设计下拉式菜单

在 New 对话框的 PB Object 选项卡中双击 Menu 图标，出现如图 1-27 所示的窗口面板。下面将在这个画板中进行菜单的设计。

(1)　菜单的常用属性

菜单的属性对话框有两个选项卡，分别如图 1-28 和图 1-29 所示，各项意义如下：

- Text。设定当前菜单项的名字。
- MicroHelp。给 MDI 窗口做的帮助，只有主窗口是 MDI 类型的窗口才有此帮助。
- Tag。给该菜单项写的一些说明，这个 Tag 对于菜单的运行没有任何影响，只是为了方便用户的检查。

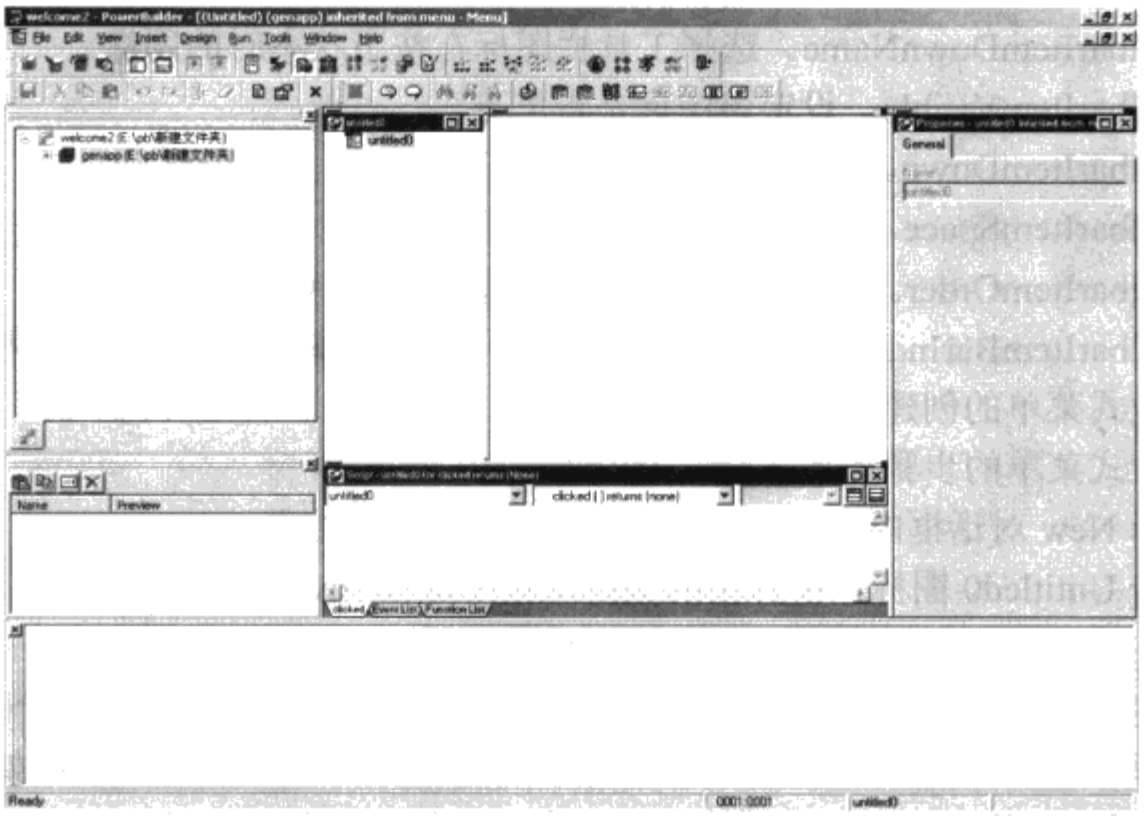


图 1-27　窗口画板

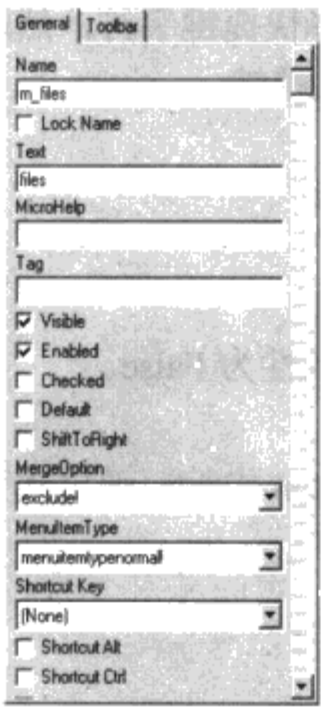


图 1-28　菜单属性对话框的 General 选项卡

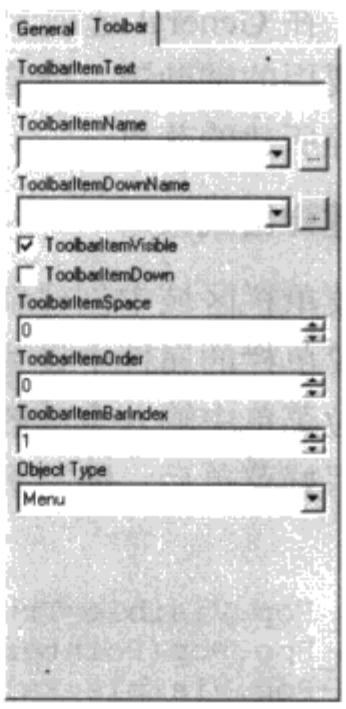


图 1-29　菜单属性对话框的 ToolBar 选项卡

- Visible。设定该菜单项可见。如果不选择这个选项，在运行时菜单是不可见的。
- Enable。设定该菜单项功能可用。如果不选这个选项，该菜单灰色显示。
- Check。选择了该选项，在菜单显示时会在当前菜单的文本前边打上勾。
- Default。设定该菜单项为默认值。
- ShiftToRight。选中该选项，改变菜单项在菜单栏位置，从右到左排列。
- MergeOption。改变菜单方式，该选项只有当窗口存在 OLE 控件时才有效。
- MenuItemType。设定菜单项的类型。
- ShortCutKey。设定菜单项快速选择键。
- ToolbarItemText。给工具图标加提示信息，当鼠标移到该图标时，会弹出此信息。
- ToolbarItemName。选择当前菜单的在工具栏上图标。

- ToolbarItemDownName。选择工具栏图标在按下状态时的图标。
- ToolbarItemVisible。设定该图标是否可见。
- ToolbarItemDown。设定该图标是否可按下。
- ToolbarItemSpace。设定在图标前的空格数。
- ToolbarItemOrder。设定图标的排列序号，其中序号小的排在左侧。
- ToolbarItemBarIndex。设定图标的栏号，相同栏号的图标在同一个栏内。

(2) 下拉式菜单的创建

创建下拉式菜单的步骤如下：

① 选择 New 对话框的 PB Object 选项卡中的 Menu 选项。

② 右击 Untitled0 图标，在弹出的快捷菜单中选择 Insert Submenu Item 命令，创建一个菜单项，在属性对话框中设置该菜单项的属性。

③ 在建好的菜单项上右击，选择 Insert Submenu Item 命令，创建这个菜单项的子菜单。如果选择 Insert Menu Item，表示在该菜单前面添加一个平级菜单项。如果选择 Insert Menu Item At End，表示在该菜单后边添加一个平级的菜单项。

④ 如果在 General 选项卡下的 Text 中输入一个减号“—”，则会在该菜单项中添加一个凹横线。在 General 选项卡的 ShortCut Key 一栏中选择快捷键字符，然后在下面的复选框中选取要用的辅助键，就会给相应的菜单项添加一个快捷键。

⑤ 保存所建的菜单。

2. 创建弹出式菜单

① 在菜单栏区域中创建新的菜单栏。

② 在菜单栏的属性中设置其名称为 m_Pop，Visible 属性为 False。

③ 下拉菜单中输入全部的要弹出的菜单项。

④ 编写好菜单后，编写事件代码，可以在对象的 Button 中编写弹出式菜单代码，如下所示：

```
m_new.m_Pop.Visible=True
m_new.m_Pop.Pop(PointerX(),PointerY())
m_new.m-Pop.Visible=False.
```

代码的含义在以后的章节中介绍。

1.5 实例：使用 PowerBuilder 创建第一个应用程序

PowerBuilder 10.0 提供了 Wizard(向导)功能，它可以更快捷方便地建立应用程序。利用应用程序向导可以快速地建立一个基本应用。利用模板程序向导方法自动产生一个比较复杂的应用，包括菜单和 MDI 窗口的应用。下面介绍一下利用 Wizard 快速创建具有窗口界面、与数据库相连接并且显示数据的应用程序。

1. 使用应用程序向导设计应用

步骤如下：

(1) 在 PowerBuilder 菜单栏中选择 File|New 命令，弹出 New 对话框，选择 Workspace

选项卡中的 Workspace 图标，如图 1-30 所示。单击 OK 按钮，在弹出的对话框中，输入要创建的 Workspace 名，然后自动生出一个 Workspace。

(2) 双击图 1-31 中的 Template Application 图标，出现如图 1-32 所示的对话框，该对话框将帮助用户建立一个应用程序。

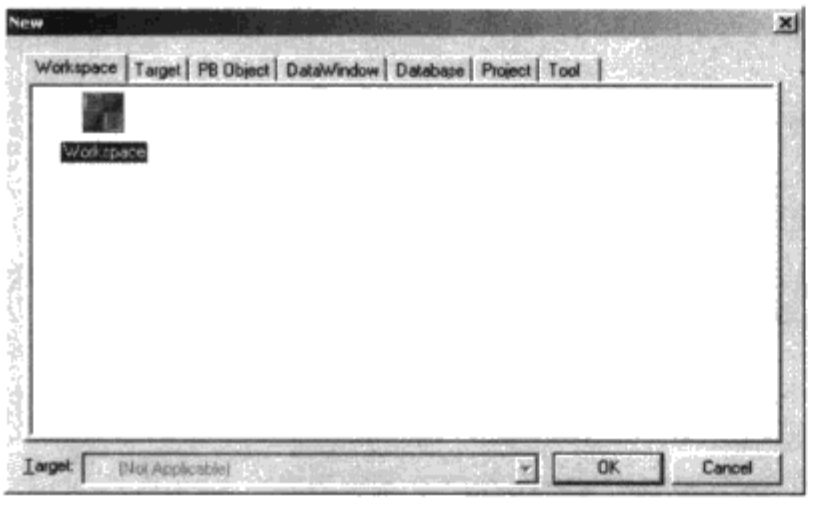


图 1-30　创建一个新的 Workspace

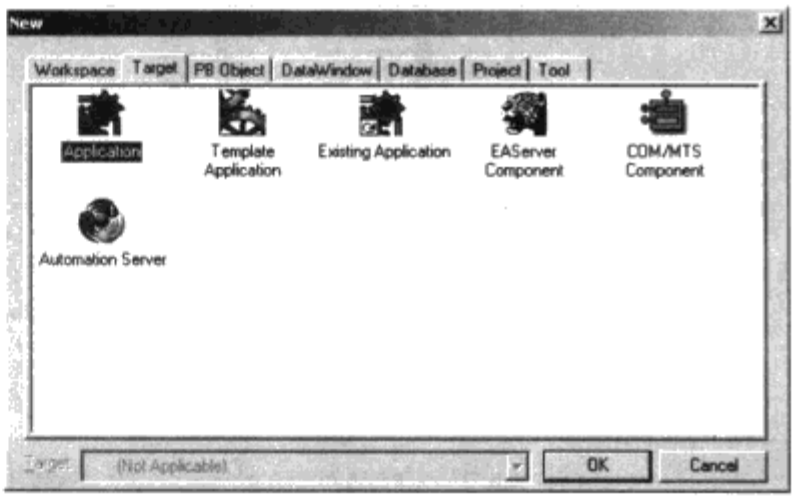


图 1-31　建立一个应用程序对话框之一

(3) 单击 Next 按钮，将出现下一个对话框，如图 1-33 所示，这个对话框将提示用户将建立的应用程序需要 8 个步骤。

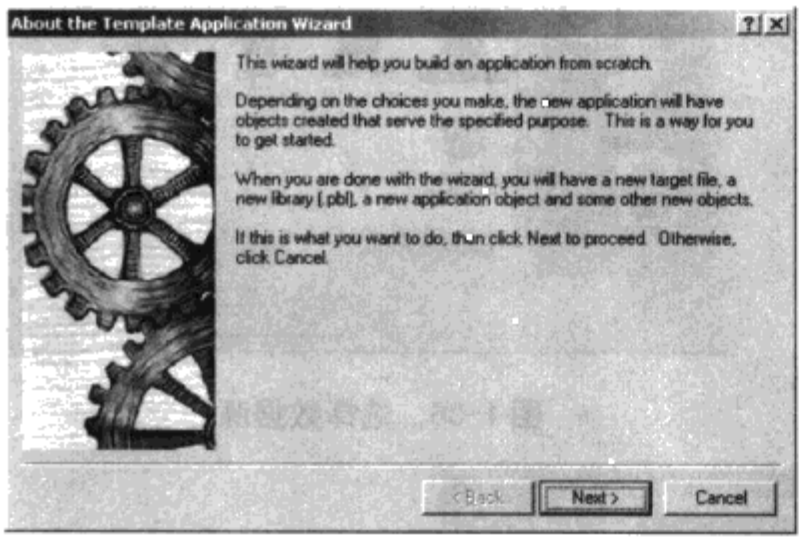


图 1-32　建立一个应用程序对话框之二

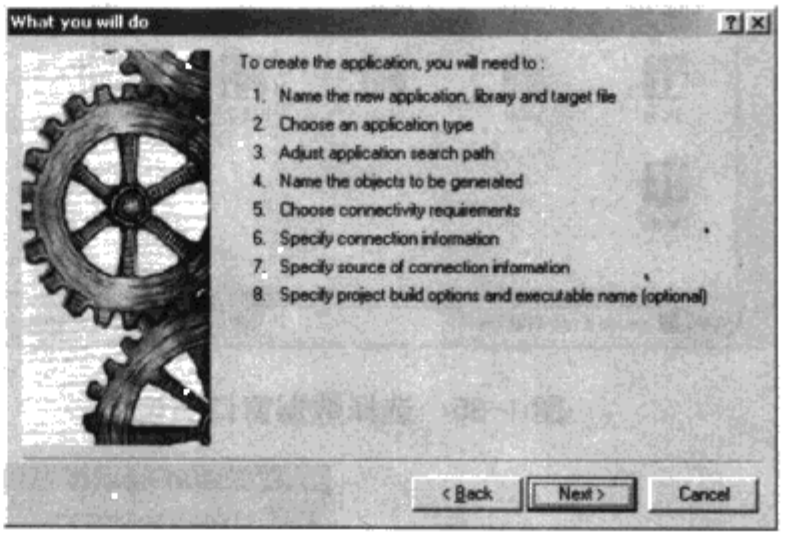


图 1-33　建立一个应用程序对话框之三

(4) 按照屏幕的提示，所有的选项都采用默认选项，完成应用程序的建立，如图 1-34 所示。

2. 创建数据窗口

步骤如下：

(1) 在 PowerBuilder 10.0 的菜单栏中选择 File | New 命令，打开 New 对话框，选择 DataWindow 选项卡，选中 Grid 图标，如图 1-35 所示。

(2) 单击 OK 按钮，进入 Grid DataWindow 的数据源选择对话框。选中 Quick Select 图标，如图 1-36 所示。单击 Next 按钮，进入数据库表选择对话框，如图 1-37 所示，选择所要选择显示的数据库表的字段。

(3) 进入数据显示窗口的背景、文字和栏颜色设置，如图 1-38 所示。

(4) 进入 Grid DataWindow 配置清单窗口，如图 1-39 所示。

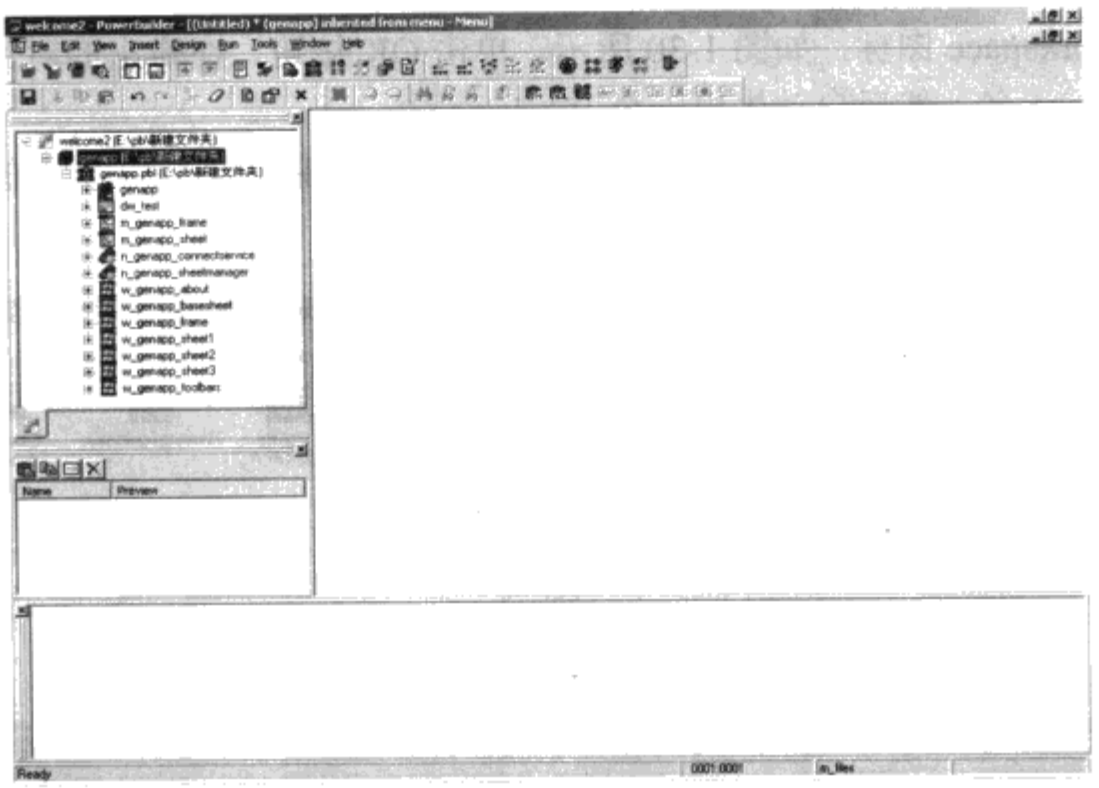


图 1-34　应用程序创建完成后的界面

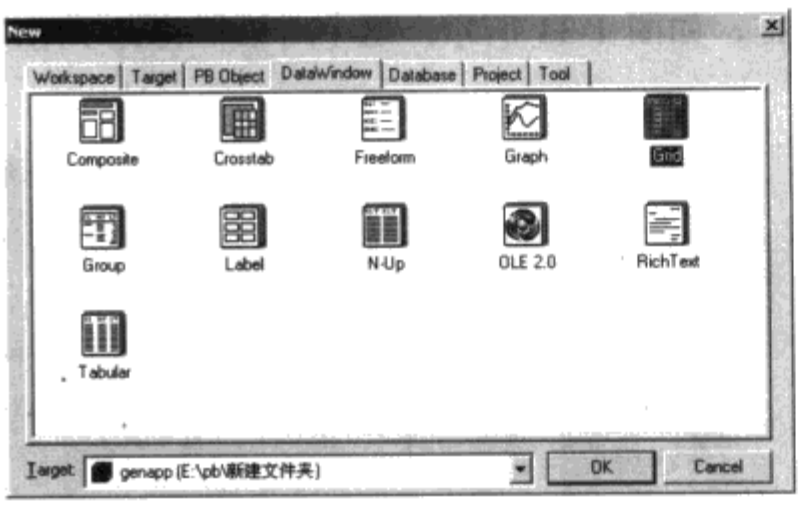


图 1-35　选择数据窗口类型

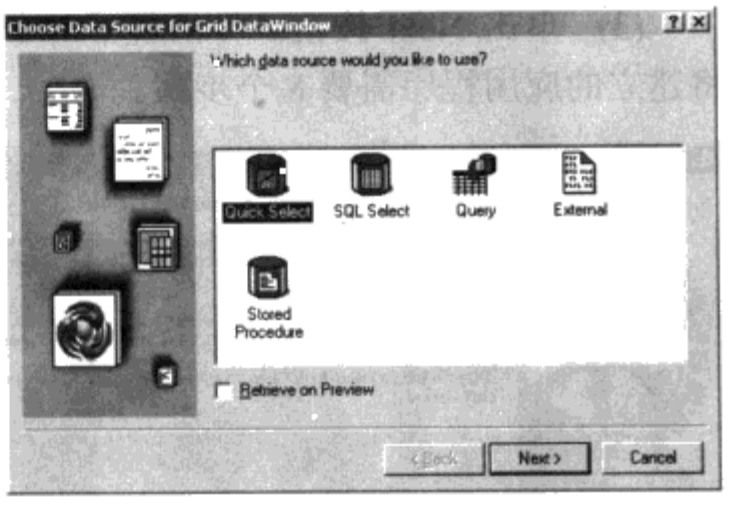


图 1-36　选择数据源

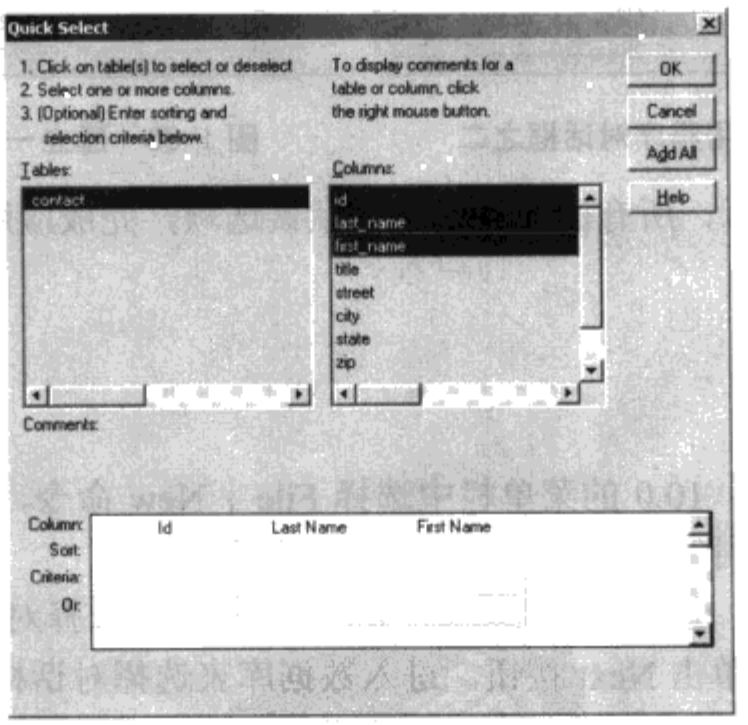


图 1-37　定义数据库表的字段

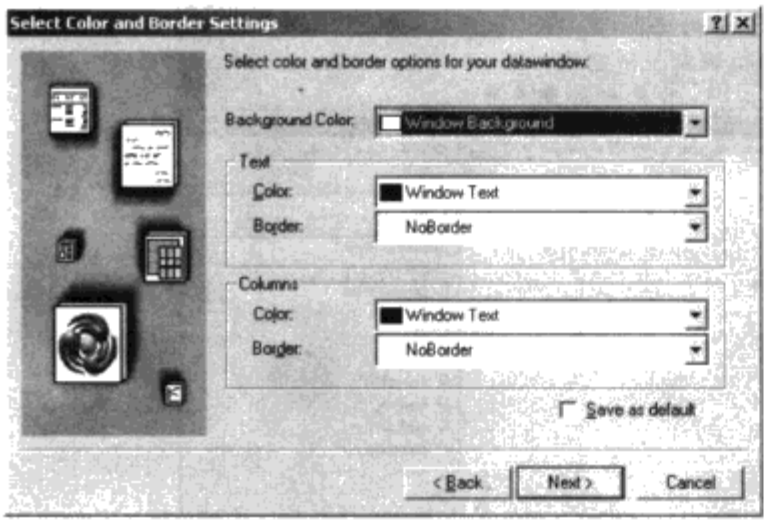


图 1-38　背景、文字和栏颜色设置

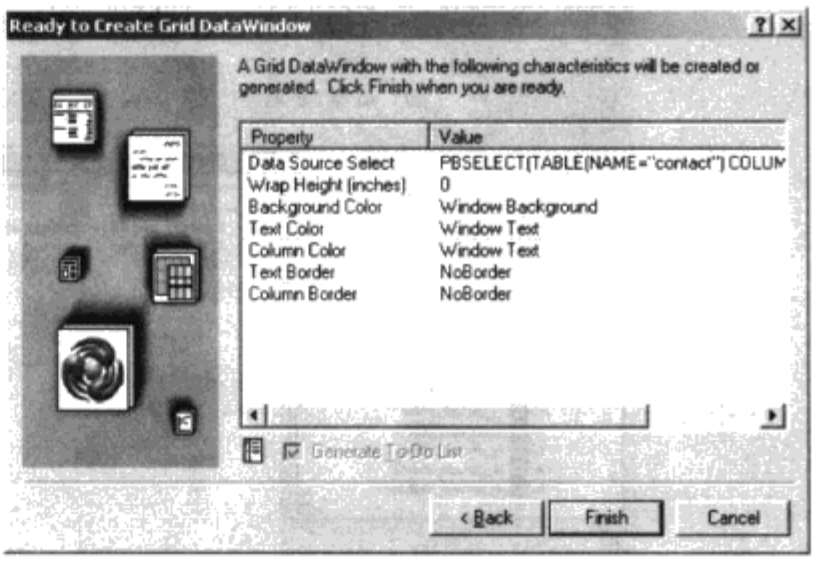


图 1-39　数据窗口的配置清单

完成数据窗口的创建，保存数据窗口文件为 dw_test，如图 1-40 所示。

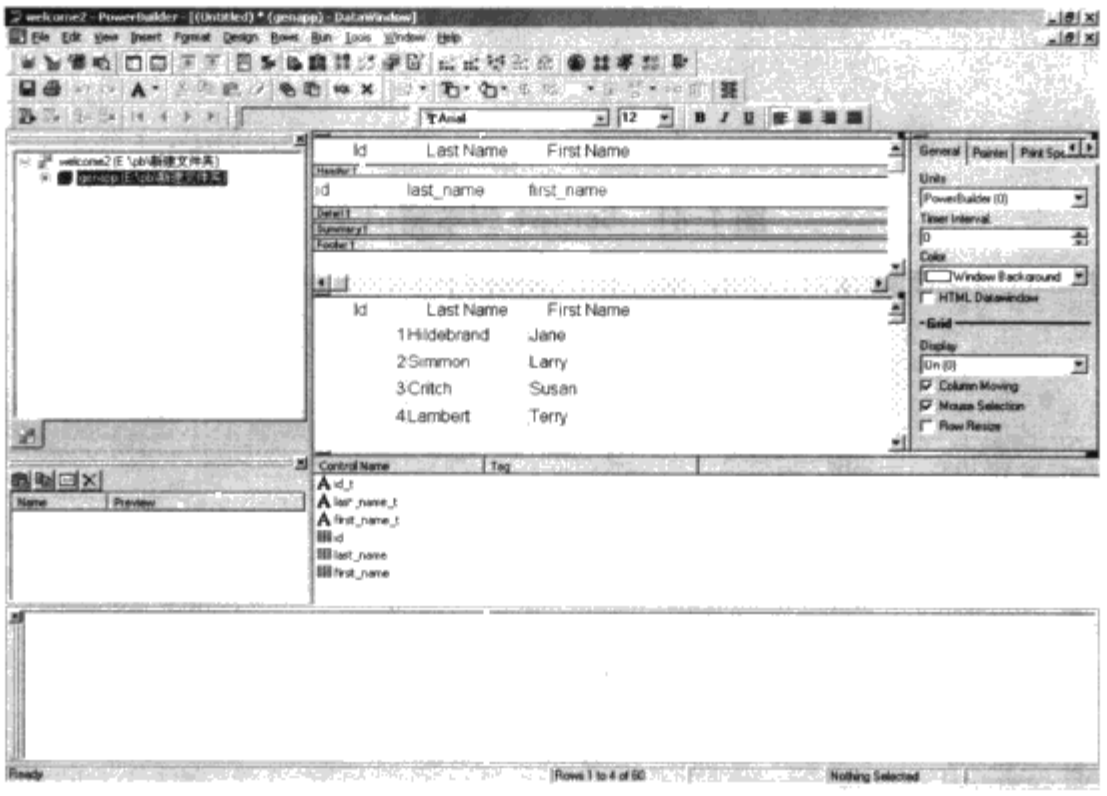


图 1-40　显示数据窗口对象

3.　连接数据窗口

打开 w_genapptest_sheet1 应用窗口，在数据窗口上添加数据窗口控件，把数据窗口控件的 DataObject 属性设为 dw_test，在 Script 窗口编写如下的脚本：

```
connect;
dw_1.settransobject(sqlca)
dw_1.retrieve()
```

如图 1-41 所示。

在 w_genapp_frame 窗口的 Script 窗口中添加如下脚本：

```
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "ConnectString='DSN=SQL Anywhere 5.0 Sample;UID=;PWD='"
```

如图 1-42 所示。

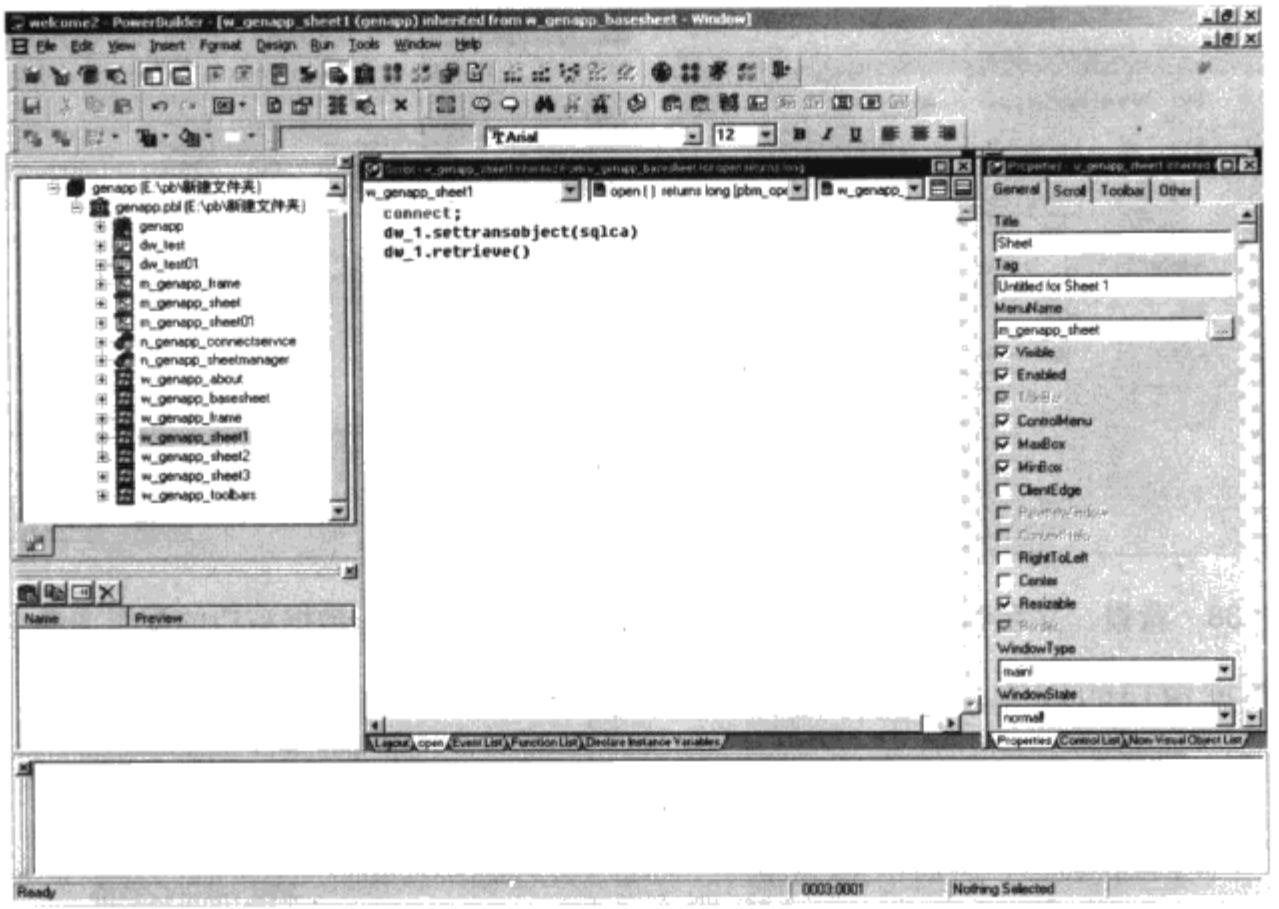


图 1-41　Script 脚本窗口 1

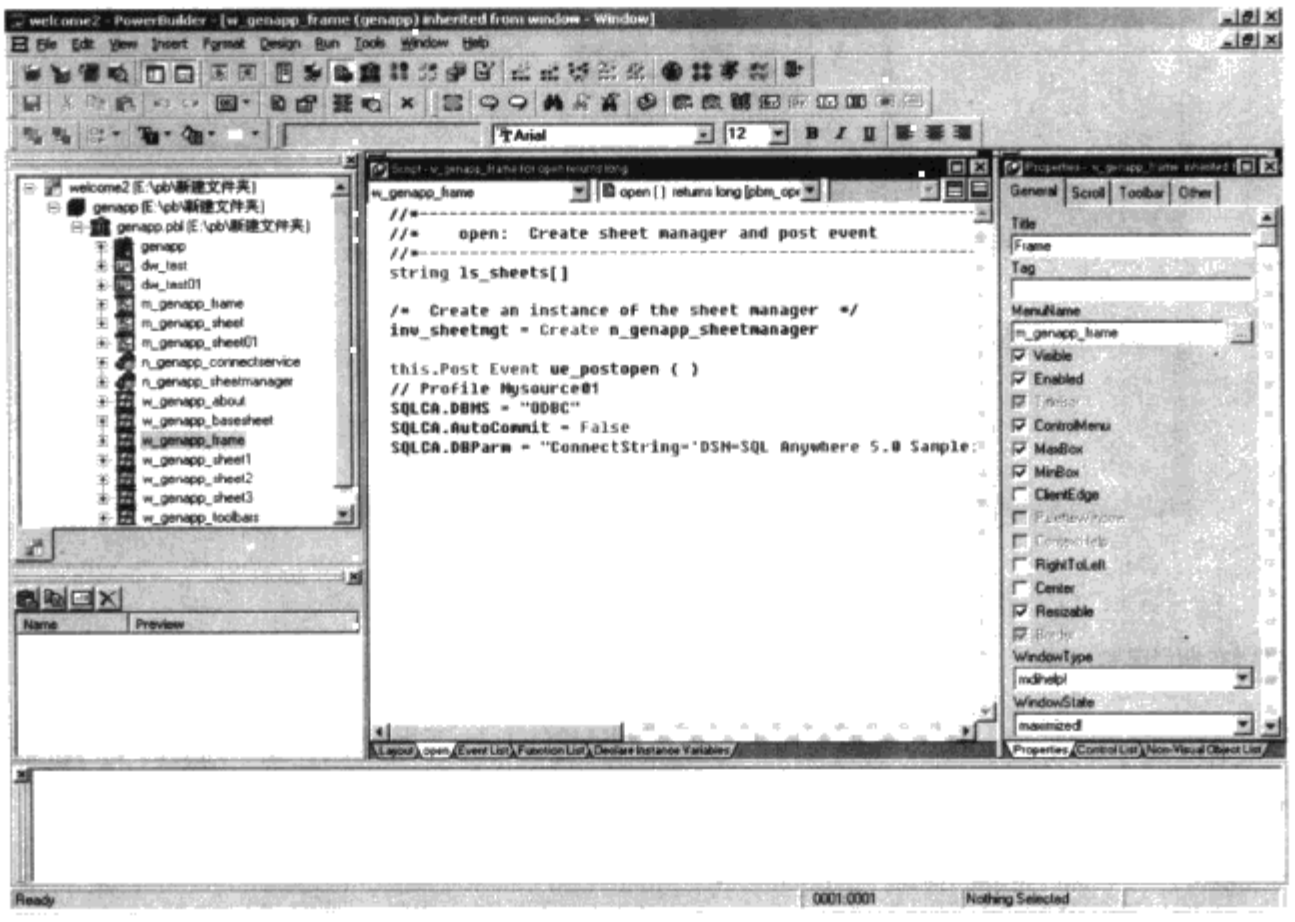


图 1-42　Script 脚本窗口 2

4.　应用程序的运行

在 Run 菜单中选择 Run Genapp 命令或在工具栏中单击 Run 图标。运行结果如图 1-43 所示。

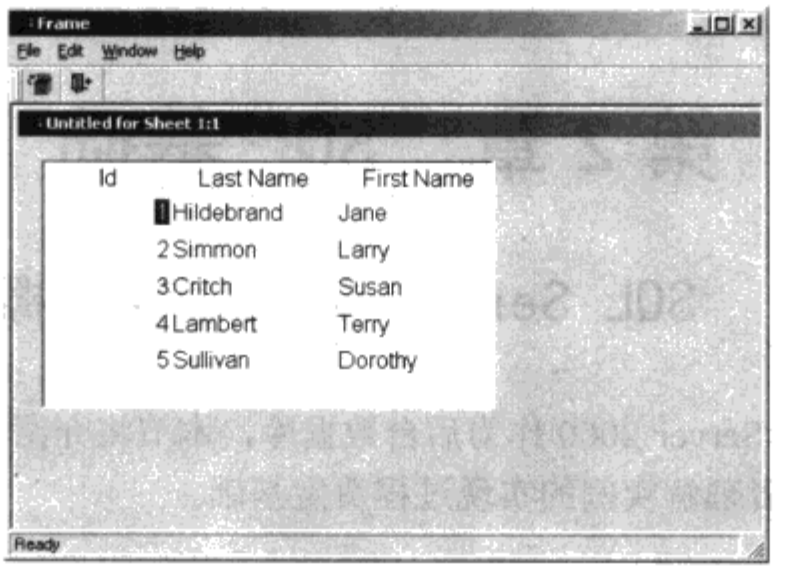


图 1-43　应用程序的运行结果

1.6　习　　题

1．填空题

(1) PowerBuilder 最具特色的就是提出了_____的概念。

(2) PowerBuilder 10.0 安装程序中含有以下组件：_____、_____、_____、_____、_____、_____。

(3) PowerBuilder 10.0 的开发环境主要分为_____、_____、_____、_____4 个部分。

(4) 单击 PowerBuilder 10.0 集成环境中_____，即可进入 PowerBuilder Help 窗口。

2．问答题

(1) PowerBuilder 有哪六大特点？

(2) 试根据 1.2 节安装 PowerBuilder 10.0。

第 2 章　SQL 基础

2.1　SQL Server 数据库管理概述

本书实例采用 SQL Server 2000 作为后台数据库，本节将介绍管理 SQL Server 数据库的基本方法，从而为读者理解实例的实现过程奠定基础。

2.1.1　SQL 语句简介

SQL 全称是 Structured Query Language(结构化查询语言)，它是一种非过程化的语言，诞生于本世纪 70 年代。

SQL 是数据库操作的标准语言，具有如下特点：

- 语言简洁、规范；
- 非过程化；
- 面向集合操作；
- 提供交互式、嵌入式两种方法。

SQL 具有以下功能：

- 查询数据库中的数据；
- 更新数据库中的数据；
- 设置数据库用户访问权限；
- 修改数据库表的结构；
- 增加删除数据库中的表、视图。

SQL 已经确立起自己作为标准关系数据库语言的地位。SQL 有许多种版本，最早的版本是由 IBM 的 San Jose 研究室提出的。该语言最初叫做 SQUEL(Structured English Query Language)，作为 System R 项目的一部分于 20 世纪 70 年代初付诸实施，发展到现在，它的名字已变为 SQL(结构化查询语言)。

SQL 语言可分为以下 3 个部分：

- 数据定义语言(DDL)。使用 DDL 可创建和删除表、模式、域、索引和视图，并可对它们进行修改。
- 数据操纵语言(DML)。不仅包括基于关系代数和元组关系演算的查询语言，还包括在数据库中插入、删除、修改元组的命令。
- 数据控制语言(DCL)。数据控制包括安全性控制、完整性控制、事务控制和并发控制。

2.1.2　企业管理器

企业管理器是 SQL Server 2000 数据库系统中最重要的管理工具，可以在企业管理器中对 SQL Server 数据库的进行各种操作。

依次单击“开始”|“程序”| Microsoft SQL Server |“企业管理器”，运行 SQL Server 2000 企业管理器。企业管理器的主界面如图 2-1 所示。

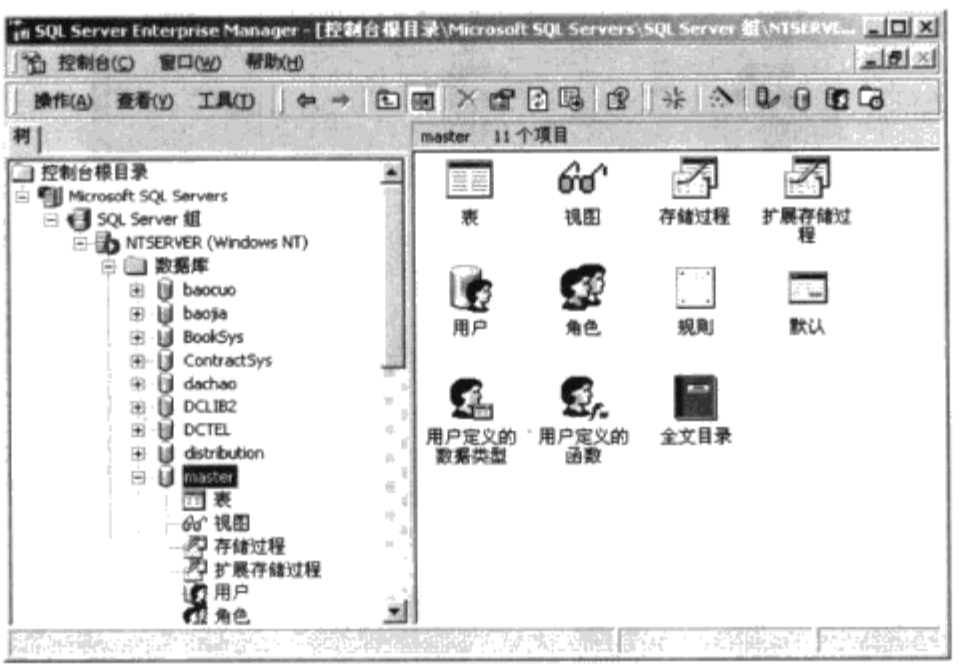


图 2-1　SQL Server 企业管理器

企业管理器的窗口分为两个部分，左侧窗口使用树状结构描述了 SQL Server 2000 的逻辑结构，右侧窗口显示左侧窗口中选择对象的具体内容。SQL Server 组可以包含多个 SQL Server 服务器，右击“SQL Server 组”，并选择菜单中的“新建 SQL Server 注册”，就可以将 SQL Server 服务器添加到树中，从而实现对网络中所有 SQL Server 数据库的管理。

在一个 SQL Server 服务器下可以创建多个数据库，展开“数据库”节点，将显示指定服务器中的所有 SQL Server 数据库。数据库又由表、视图、用户等数据库对象组成，在企业管理器中，可以对这些数据库对象进行添加、修改、删除和查看等管理。

2.1.3　创建数据库

在创建数据库之前，必须先确定数据库的名称、所有者(创建数据库的用户)、大小以及用于存储该数据库的文件和文件组。除了数据库名称外，其他参数都可以由系统默认设置。

打开 SQL Server 企业管理器，可以按照下面的方法来创建数据库。

(1) 展开 SQL Server 组，然后展开服务器实例。

(2) 右击“数据库”，然后单击“新建数据库”命令，打开“数据库属性”窗口，如图 2-2 所示。

(3) 输入新数据库的名称，假定数据库名为 UserMan。默认情况下，系统自动地使用指定的数据库名作为前缀创建主数据库和事务日志文件，例如：UserMan_Data.mdf 和 UserMan_Log.ldf。

(4) 如果需要设置数据文件和事务日志文件，可以分别单击“数据文件”和“事务日志”选项卡。建议读者采用默认设置。

(5) 单击“确定”按钮，完成创建数据库的操作。在企业管理器中可以看到新建的数据库。

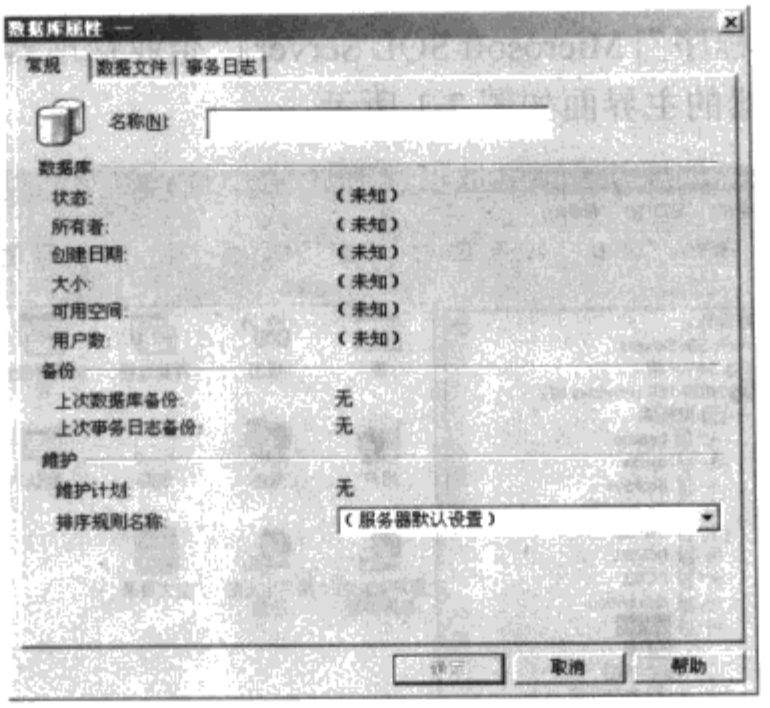


图 2-2　数据库属性

2.1.4　创建表

表是包含数据库中数据的数据库对象。在设计数据库结构时，最主要的工作就是决定数据库由哪些表组成，然后定义每个表的结构。

表的结构由若干个列(字段)组成。在定义表结构时，需要指定列名、数据类型、长度和其他列属性。

SQL Server 的常用数据类型如表 2-1 所示。

表 2-1　SQL Server 常用数据类型说明

数据类型	描　　述
Image	可以用来存储超过 8 KB 的可变长度的二进制数据
Char	固定长度的字符数据类型。Char[(*n*)]表示长度为 *n* 个字节的固定长度且非 Unicode 的字符数据。*n* 必须是一个介于 1~8 000 之间的数值。存储大小为 *n* 个字节
Varchar	是非固定长度的字符数据类型。Varchar[(*n*)]表示长度为 *n* 个字节的可变长度且非 Unicode 的字符数据。*n* 必须是一个介于 1~8 000 之间的数值。存储大小为输入数据的字节的实际长度，而不是 *n* 个字节。所输入的数据字符长度可以为零
Datetime	从 1753 年 1 月 1 日到 9999 年 12 月 31 日的日期和时间数据，精确度为 0.003s(等于 3.33ms 或 0.00333s)
Int	从 -2^{31} (−2 147 483 648) − 2^{31}～1(2 147 483 647)的所有整型数据。存储大小为 4B
Smallint	从 -2^{15} (−32 768) ～2^{15} − 1 (32 767)的整型数据。存储大小为 2B
Tinyint	从 0～255 的整型数据。存储大小为 1B

续表

数据类型	描　　述
Decimal 和 Numeric	带定点精度和小数位数的 Numeric 数据类型。Decimal[(*p*[, *s*])] 和 Numeric[(*p*[, *s*])] 表示定点精度和小数位数。使用最大精度时，有效值从 $-10^{38}+1$～$10^{38}-1$。*p* 表示精度，指定小数点左边和右边可以存储的十进制数字的最大个数。精度必须是从 1 到最大精度之间的值。最大精度为 38。*s* 表示小数位数，指定小数点右边可以存储的十进制数字的最大个数。小数位数必须是从 0 ~ *p* 之间的值。默认小数位数是 0，因而 $0 \leqslant s \leqslant p$。最大存储大小基于精度而变化
Float [(*n*)]	从-1.79E + 308～1.79E + 308 之间的浮点数字数据。*n* 为用于存储科学记数法 Float 数尾数的位数，同时指示其精度和存储大小。*n* 必须为从 1～53 之间的值
Real	从 -3.40E + 38～3.40E + 38 之间的浮点数字数据。存储大小为 4B
Bit	Bit 数据类型只能包括 0 或 1。可以用 Bit 数据类型代表 TRUE 或 FALSE、YES 或 NO。例如，询问客户是否为初次访问的问题可存储在 bit 列中，类似于其他数据类型中的布尔型

在定义 char、varchar、decimal、numeric 和 float 数据类型时，需要指定数据的长度。下面在数据库 UserMan 中创建一个表 Users，用于保存用户信息，结构如表 2-2 所示。

表 2-2　　表 Users 的结构

字段名	数据类型	长度	具体说明
UserId	int		用户编号
UserName	Char	10	用户名
UserPwd	Char	10	密码

在 SQL Server 2000 企业管理器中，展开 SQL Server 组，然后展开服务器实例。展开要添加表的数据库，右击“表”目录。选择菜单中“新建表”命令，打开表设计器，如图 2-3 所示。

首先输入列名。列名必须惟一，而且要符合 SQL Server 的标识符规范。

选择一列，在表设计器的上半部分可以编辑列名、数据类型、长度和允许空等属性。如果选中“允许空”复选框，则该列数值可以为空，否则在输入数据时，必须输入该列数据。在表设计器的下半部分，可以编辑列的其他属性，包括描述、默认值、精度、小数位数和标识等。

输入完成后，单击“保存”图标，将弹出“输入表名”对话框，如图 2-4 所示。

输入表名后单击“确定”按钮，可以在企业管理器中看到新建的表。右击表，在弹出菜单中选择“设计表”命令，可以打开表设计器对其进行修改。

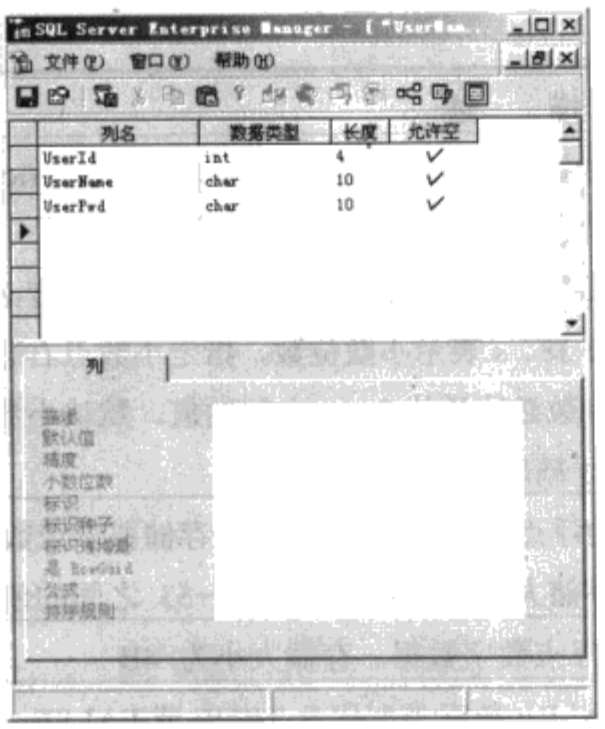


图 2-3 表设计器

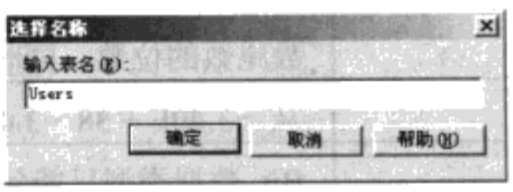


图 2-4 输入表名

右击表，在弹出菜单中选择“打开表”|“返回所有行”，可以表格的形式查看表中的数据，如图 2-5 所示。

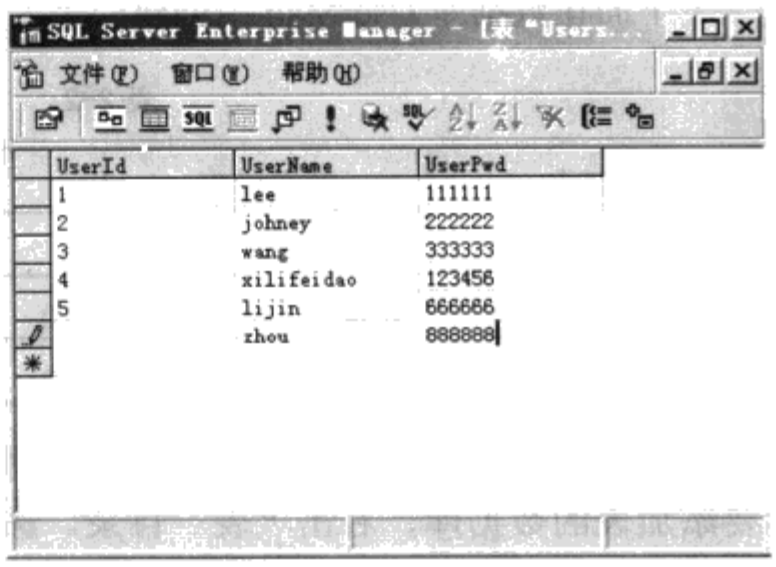


图 2-5 查询表中数据

在表中，有的列可以被设置作为行的惟一标识，也就是说在表的所有行中，此列的数据是惟一的。通常情况下，可以把编号列设置为惟一标识列。为了描述这种惟一标识列，数据库中引入了键(key)的概念。在 SQL Server 中，有两种类型的键：

- 主键(PRIMARY KEY)。表中的一组列，对表中的每一行这些列都有惟一的值。如果行的主键值已经存在，则此行不允许被插入。表只能有一个主键。
- 外键(FOREIGN KEY)。是用于建立和加强两个表数据之间的链接的一列或多列。有时称这两个参与表为外键表和主表。外键可以约束表中的行，因此也称为外键约束。外键确保添加到外键表中的任何行在主表中有对应的行。

在表设计器中单击要定义为主键的数据列，如果要选择多个列，可同时按下 Ctrl 键。右击该列，然后选择“设置主键”命令。此列的前面出现了主键符号，同时“允许空”的标记被去掉。

通常使用编号列作为主键。列 UserId 并没有实际意义，只是用来标识表中的每条记录。因此，没有必要由用户设置列 UserId 的值。可以定义该列的值由系统自动分配。在表设计器中选择列 UserId，在下面的属性栏中设置“标识”为“是”、“标识种子”为 1、“标识递增量”为 1，表示该列的值由系统自动生成，每次递增量为 1，如图 2-6 所示。

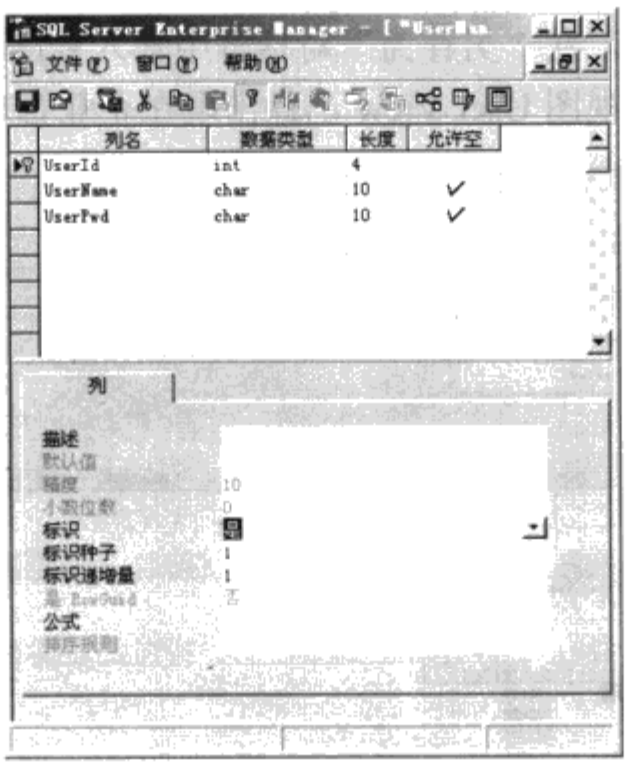


图 2-6　设置标识列

2.1.5　创建视图

视图是虚拟的表，它根据用户指定的条件从表中查询数据，并以表格的形式返回结果。视图中的数据可以来自于一个或多个表。

例如，可以从用户表 Users 中提取 UserName 和 UserPwd 创建一个视图，方法如下：

(1) 在企业管理器中，展开指定实例的数据库，右击“视图”项，在弹出菜单中选择“新建视图”命令，打开“视图设计器”窗口。此窗口由 4 部分窗格组成，由上到下分别为关系图窗格、网格窗格、SQL 窗格和结果窗格，如图 2-7 所示。

(2) 右击关系图网格，选择“添加表”命令，打开“添加表”对话框，如图 2-8 所示。

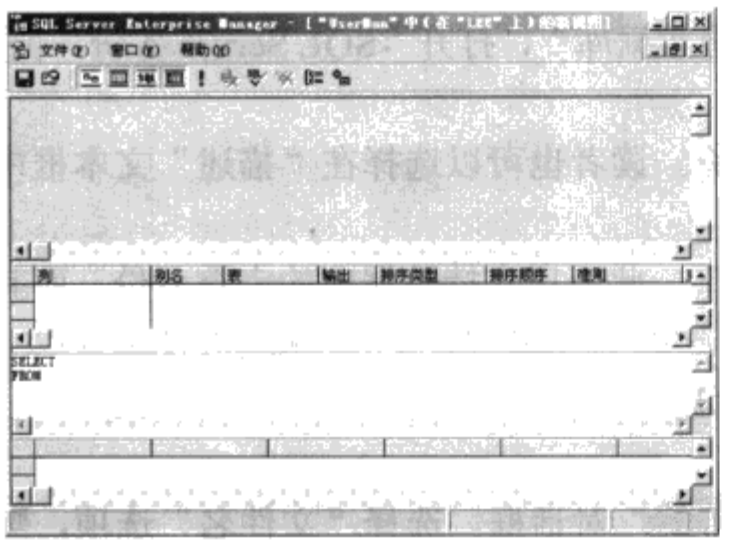


图 2-7　视图设计器

图 2-8　“添加表”对话框

(3) 选择表 Users，单击“添加”按钮，将表 Users 添加到关系图网格中。

(4) 在关系图网格中，选择表 Users 中的 UserName 和 UserPwd 字段。

(5) 设置完成后，单击工具栏中的“运行”按钮！，运行视图。结果出现在“结果窗格”中，如图 2-9 所示。

(6) 单击“保存”图标，在“另存为”对话框中输入视图名 UserView。

关闭“视图设计器”，视图 User View 已经出现在企业管理器中。

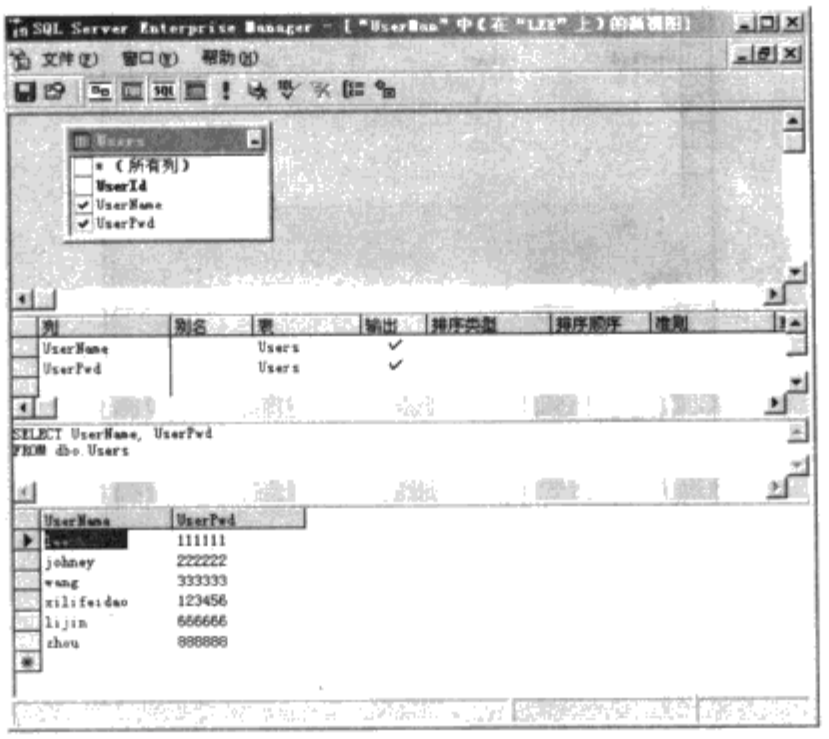


图 2-9 创建新视图

2.1.6 备份和还原数据库

为了保证用户数据的安全性，防止数据库中的数据意外丢失，SQL Server 2000 提供了备份和还原数据库组件。定时备份数据库是保护数据库的最佳选择。

在企业管理器中可以按照如下方法对数据库进行备份。

(1) 在企业管理器中，展开要备份的数据库所在的服务器组，然后展开服务器。展开“数据库”文件夹，右击要备份的数据库。

(2) 在弹出菜单中选择“所有任务”|“备份数据库”，打开“SQL Server 备份”窗口，如图 2-10 所示。

(3) 在“名称”文本框内，输入备份集名称。读者也可以选择在“描述”文本框中输入对备份集的描述。

(4) 在“备份”选项下单击“数据库-完全”，此选项支持数据库的完整备份，而其他选项只能备份数据库的一部分。

(5) 在“目的”选项下，单击“磁带”或“磁盘”。如果计算机上没有安装磁带设备，则此选项不可用，只能选择“磁盘”。

(6) 单击“添加”按钮，打开“选择备份目的”对话框。选择“文件名”选项，默认的文件为 C:\Program Files\Microsoft SQL Server\MSSQL\BACKUP\。注意，这只是一个目录，读者还需要自己输入一个备份文件名。如果要备份到指定的目录，请单击...按钮。如图 2-11 所示。

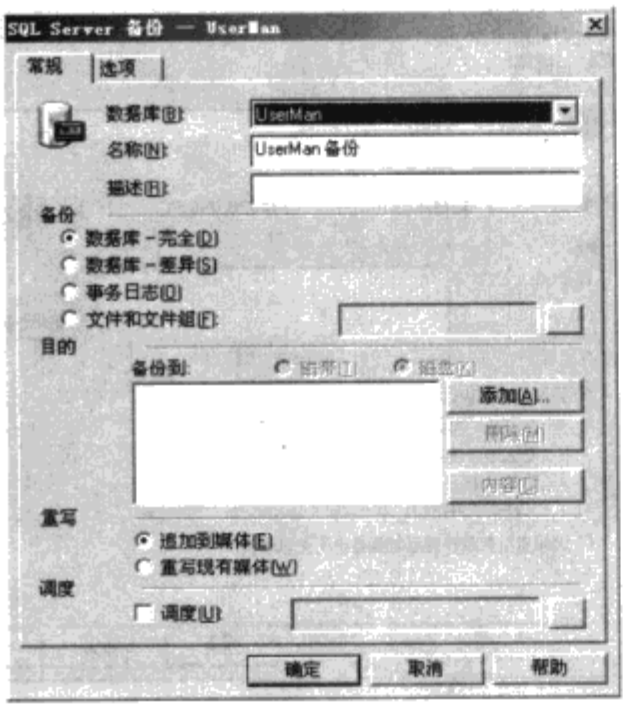


图 2-10　SQL Server 备份对话框

图 2-11　选择备份目的

(7)　单击“确定”按钮，返回“SQL Server 备份”对话框。

(8)　在“重写”选项下，读者可以选择：

- “追加到媒体”，将备份追加到备份设备上任何现有的备份中；
- “重写现有媒体”，将重写备份设备中现有的任何备份。

(9)　如果需要设置定期执行备份操作，请选择“调度”复选框，然后单击后面的...按钮，打开“编辑调度”对话框，如图 2-12 所示。

(10) 在“编辑调度”对话框中设置完成后，请单击“确定”按钮，返回“SQL Server 备份”对话框。

(11) 如果一切都设置完成，则在“SQL Server 备份”对话框中，单击“确定”按钮，开始备份。

如果备份远程计算机上的数据库，则备份文件将在远程计算机上生成。

还原 SQL Server 数据库的过程如下：

(1)　在企业管理器中，展开要还原的数据库所在的服务器组，然后展开服务器。右击“数据库”，在弹出菜单中选择“所有任务”|“还原数据库”命令，打开“还原数据库”对话框，如图 2-13 所示。在“还原为数据库”文本框中输入要还原的数据库名称。

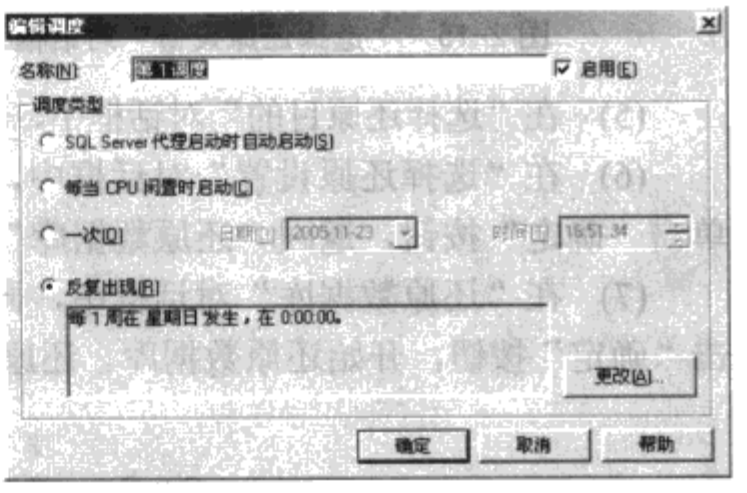


图 2-12　“编辑调度”对话框

(2)　在“还原”选项中选中“从设备”单选按钮，如图 2-14 所示。

(3)　单击“选择设备”按钮，打开“选择还原设备”对话框，如图 2-15 所示。

(4)　在“选择还原设备”对话框中，同样可以看到“磁盘”和“磁带”两种设备。选中“磁盘”单选按钮，然后单击“添加”按钮，打开“选择还原目的”对话框，如图 2-16 所示。选中“文件名”单选按钮，并输入目的文件。也可以单击...按钮，在弹出的窗口中选择目的文件。

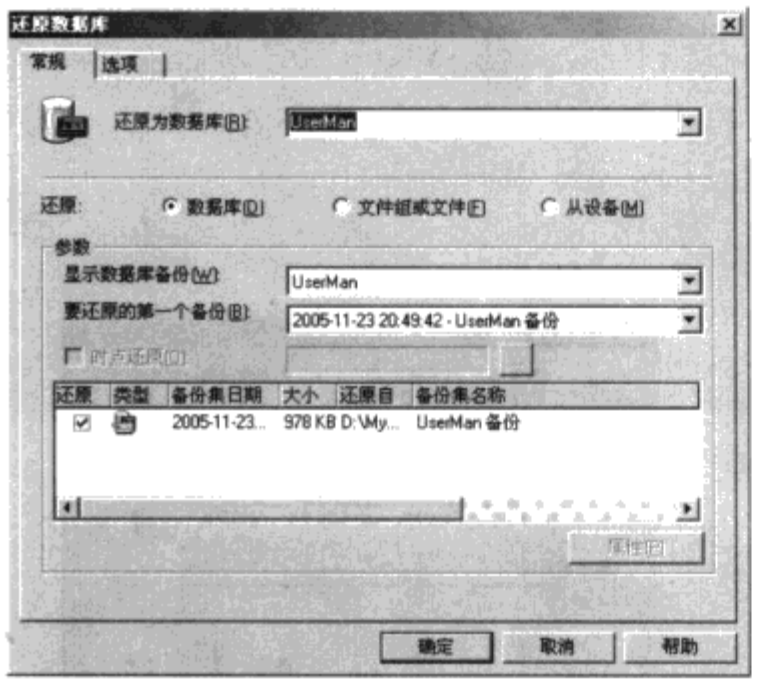


图 2-13 “还原数据库”对话框

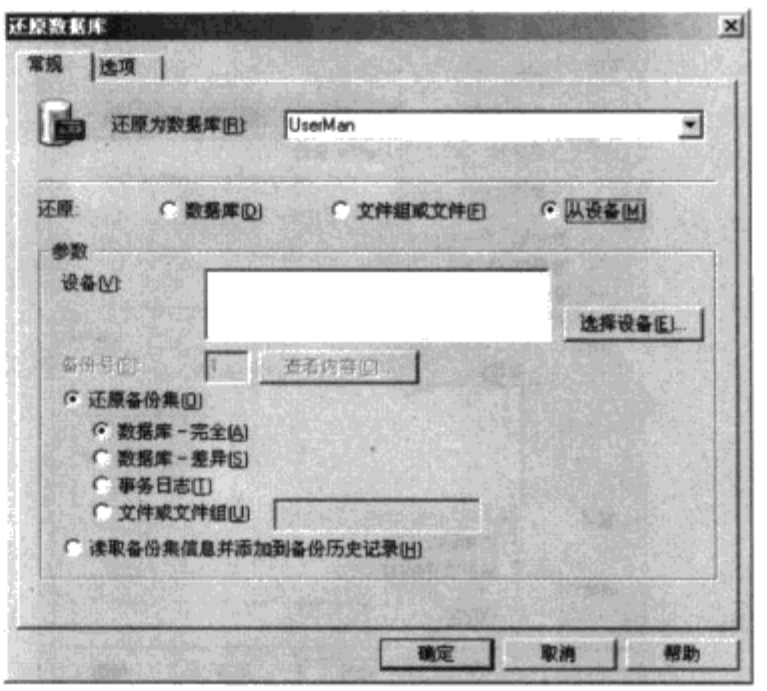


图 2-14 选中“从设备”单选按钮

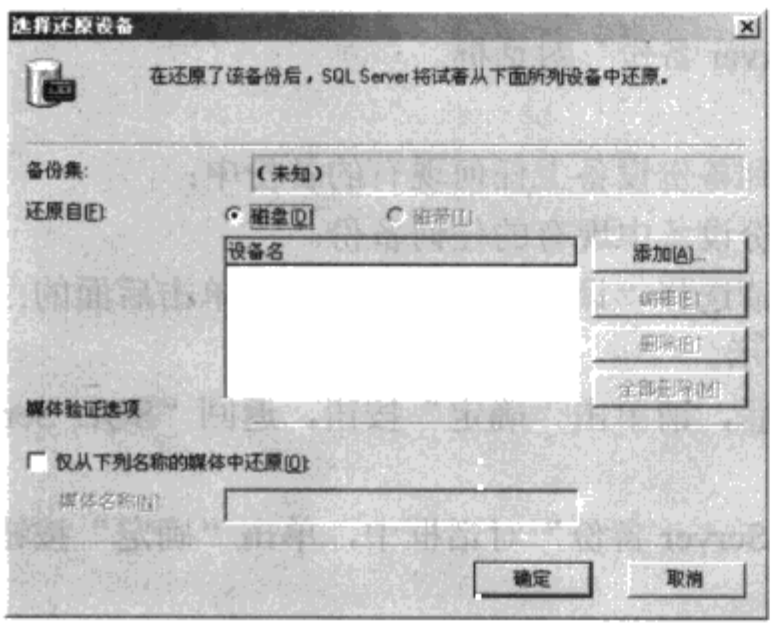


图 2-15 “选择还原设备”对话框

图 2-16 “选择还原目的”对话框

(5) 在“选择还原目的”对话框中，单击“确定”按钮，返回“选择还原设置”对话框。

(6) 在“选择还原设置”对话框中，可以看到“设备名”列表中出现了选择的设备。单击“确定”按钮，返回“还原数据库”对话框。

(7) 在“还原数据库”对话框中，同样可以看到“设备”列表中出现的选择设备。单击“确定”按钮，开始还原数据库。还原操作完成后，将出现一个对话框，提示还原成功。

2.2 SQL的数据定义功能

一个典型的 SQL 查询表达式的基本结构为：

```
Select 属性名表
From 关系名表
Where(条件表达式);
```

Select 子句对应关系代数中的投影运算，其中的属性名表指出要查询和列出的属性名

集合；From 子句中的关系名表是要查询的属性名及条件表达式中的属性名所在的关系名集合；Where 子句指出要查询的属性必须满足的条件，其条件表达式由运算符、属性名及常数组成。

SQL 先构造 From 子句中关系的笛卡儿积，根据 Where 子句中的条件表达式进行关系代数的选择运算，然后将结果投影到 Select 子句中的属性上。

2.2.1　Select 子句

Select 子句从数据库中检索数据并将查询结果以表格的形式返回，即结果必是一个关系。来看一个简单的查询“找出关系 s 中所有学生的名字”：

```
Select  sname
Froms
```

结果是一个只有属性 sname 的关系。

SQL 像大多数其他商业查询语言一样，允许在关系和 SQL 表达式结果中出现重复。如果想要强迫删除重复，可在其后加入关键词 distinct。例如：

```
Select distinct  s#, c#
From   sc
```

SQL 允许使用关键词 all 来显式指明不删除重复：

```
Select  all  s#, c#
From   sc
```

因为保留重复元组是默认的，可以不使用 all。为了保证在查询结果中删除重复元组，我们将在任何有必要的地方使用 distinct。

星号“*”可用来表示“所有的属性”，形式为 Select *的 Select 子句表示出现在 From 子句中的所有关系的所有属性都应该出现在结果中。

子句还可以包含带+，-，×，/ 的算术表达式，运算对象可以是常数或元组的属性。例如查询：

```
Select  branch-name, loan- number,   amount*100
Fromloan
```

将返回一个与 loan 一样的关系，只是属性 amount 的值是原来的 100 倍。

SQL-92 还提供了一些特殊的数据类型，如各种形式的日期类型，并包含一些作用于这些类型的算术函数，如：

```
Select  name, month (hire-time),year   (hire- time)
From employer
```

2.2.2　Where 子句

带上 Where 子句的查询就是条件查询。SQL 提供了一组丰富的搜索条件，可使用户有效地指定许多不同种类的查询。下面总结了 5 种基本的搜索条件，并分别在后面予以介绍：

- 比较测试

比较一个表达式与另一个表达式的值。例如，找出贷款额超过$1200 的贷款的贷款号。

- 范围测试

测试表达式的值是否进入某个特定的范围。例如，找出存款在$700 和$900 之间的账户。

- 设定成员测试

检查表达式的值是否与设定的值匹配。例如，选择在 Brooklyn，Rye 或 Horseneck 的银行分行。

- 样式匹配测试

检查包含字符串数据字段的值是否与指定的样式匹配。例如，选择名字以“H”开头的客户。

- NULL 值测试

检查字段中是否具有 NULL(未知的)值。

1. 比较测试

比较测试是 SQL 查询中使用最普遍的搜索条件。在比较测试中，SQL 计算和比较数据行的两个包含比较运算符<，<=，>，>=，=和<>的 SQL 表达式。SQL 允许使用比较运算符比较字符串和算术表达式以及特殊类型，如日期类型。

例如，列出账号不是 A-201 的客户：

```
Select customer-name
From depositor
Where account-number <> 'A-201'
```

SQL 在比较测试中比较两个表达式值时，会出现 3 种结果：

```
TRUE, FALSE, NULL
```

只有搜索结果为 TRUE 的行才能放入查询结果中。

2. 范围测试(between)

为了简化 Where 子句，SQL 还提供 between 比较运算符，说明一个值小于或等于某个值同时大于或等于另一个值。如想找出“贷款额在 90 000～100 000 美元之间的贷款的贷款号”，可以使用 between 比较，书写如下：

```
Select  loan-number
From   loan
Where  amount  between 90 000 and 100 000
```

同时，还可以使用 not between 比较运算符。

3. 组成员测试(in)

另外一种普遍使用的搜索条件是组成员测试(in)。它测试数据值是否与一列目标值匹配。

例如，列出在 Rye，Stamford，Harrison 的客户。

```
Select  customer-name, customer-street
From customer
Where  customer-city in ('Rye', 'Stamford', 'Harrison')
```

同样能通过 not in 形式的组成员测试检查数据值是否不匹配任何目标值。下面的查询是找出在银行中有贷款的用户，他的名字既不是“Smith”，也不是“Jones”。

```
Select distinct  customer-name
From  borrower
Where customer-name not in ("Smith","Jones")
```

in 测试中的测试表达式除了字段名外，可以是任何 SQL 表达式。

4.　样式匹配测试(like)

样式匹配测试用于检查一字段中的内容能否与特定样式匹配。样式是包含一个或多个不确定字符的字符串，用百分号(%)和下划线(_)分别来代表零个或多个字符组成的字符串及单个字符。例如：

```
Select  branch-name, assets
From   branch
Where  branch-city like  'Benn%'
```

在需要时，也可以使用“not like”比较运算符。

5.　NULL 值测试(is null)

有时，在搜索条件中明确地检查 null 值和直接处理是有实际意义的。SQL 提供了一个特殊 null 值测试来完成这个任务。例如，想找出样本数据库中还没有开设银行分行的城市，则用如下语句实现：

```
Select  branch-city
From branch
Where  branch-name  is  null
```

null 的否定形式(is not null)找出不包含 null 值的行。

与以前的搜索条件不同，null 值测试不会产生 null 值的结果，它总是 TRUE 或 FALSE。

6.　组合搜索条件(And, Or 和 Not)

SQL 在 Where 子句中使用逻辑运算符 and，or 和 not 来将简单的搜索条件组合，形成更复杂的搜索条件。

注意：　不是使用符号∧，∨和¬。

关键字 and 用来结合两个搜索条件，两个条件必须都为真才满足条件。例如，想找出在 Perridge 分行贷款而且贷款额多于$1300 的贷款号。该查询用 SQL 可书写如下：

```
Select loan-number
From  loan
Where  branch-name="Perridge" and amount>1300
```

而用关键字 or 来结合两个搜索条件，这时其中一个为真就满足条件。例如，想找出在 Perridge 分行贷款或贷款额多于$1300 的贷款号。该查询用 SQL 可书写如下：

```
Select loan-number
From  loan
Where branch-name="Perridge"  or  amount>1300
```

还可用关键字 not 去选择，其搜索条件为 FALSE。例如。列出在 Perridge 分行贷款，而且贷款额不多于$1300 的贷款号：

```
Select loan-number
From  loan
Where branch-name="Perridge"  and not amount>1300
```

当由 and，or 和 not 结合的搜索条件多于两个时，not 的优先级别最高，其次为 and 和 or。

2.2.3 From 子句

最后，来介绍 From 子句的用法。From 子句中列出的表是查询结果中数据的来源，称为查询的源表。源表的个数可为一个或多个，即为单表查询和多表查询。单表查询的情况比较简单，实际应用中往往是多表查询，即多个关系的多种连接。

举一个简单的例子，为了查询“找出从银行贷款的所有用户的名字和贷款号”，可用 SQL 书写如下：

```
Select distinct customer-name, borrower.loan-number
From   borrower, loan
Where  borrower  loan-number = loan.loan-number
```

在例子中，为了避免一个属性名出现在多个关系中所引起的混乱，使用了 relation-name.attribute-name 的写法，如：

```
borrower.loan-number
```

2.2.4 更名运算

SQL 提供可为关系和属性重新命名的机制，这是通过使用具有如下形式的子句来进行的：

```
old-name  as  new-name
```

该子句既可出现在 Select 子句中，也可出现在 From 子句中。

考虑上一个例句，查询结果中的属性名 customer-name,loan-number 是从 From 子句中的关系的属性名派生出来的。这种办法使用起来还是很不方便的，原因如下：

(1) 在子句中的两个关系可能含有相同名字的属性，在这种情况下，结果中就会有一个属性名重复出现；

(2) 如果在 Select 子句中使用了算术表达式，结果属性就没有名字了；

(3) 有时用户想要更改结果中的属性名。因此，SQL 提供了更改关系中属性名的方法。

例如，上例可改写为：

```
Select distinct customer-name, borrower.loan-number  as loan-id
From   borrower, loan
Where  borrower.loan-number = loan.loan-number
```

2.2.5　元组变量

SQL 中的元组变量必须和特定的关系相联系。元组变量是通过在 From 子句中使用 as 子句来定义的。为了说明这一用法，再次重写查询“找出从银行贷款的所有用户的名字和贷款号”：

```
Select distinct customer-name, borrower.loan-number
From   borrower as T, loan as S
Where  T.loan-number =S.loan-number
```

注意： 在 From 子句中定义元组变量是通过将其放在与之相联系的关系名之后，而关键词 as 放在中间(as 是可选的)。

元组变量在比较同一关系中的两个元组时非常有用。假设需要查询“找出资产至少比位于 Brooklyn 的某一家分支机构高的分支机构名”，写成 SQL 表达式如下：

```
Select distinct T.branch-name
From   branch as T, branch as S
Where T.assets>S.assets and S.branch-city ="Brooklyn"
```

注意： 不能使用 branch.asset 这样的写法，因为我们并不清楚到底对 branch 的哪个引用是所需要的。

2.2.6　排列元组的显示次序(order by)

SQL 中 order by 子句可以让查询结果按排列顺序显示，用 desc 表示降序，用 asc 表示升序。order by 子句默认时使用升序。此外，排序可在多个属性上进行。假设希望关系按 amount 降序排列，但如果有相同贷款额，再将它们按贷款号升序排列。该查询表达式如下：

```
Select *
From loan
order by amount desc, loan-number asc
```

2.2.7　集合操作

SQL 在关系上的 union，intersect 和 except 操作对应于关系代数中的运算∪，∩和-。SQL 中参加集合操作的关系也必须是相容的。

考虑在以下两个集合上进行的 union，intersect 和 except 操作的查询，这两个集合分别是在银行有账户的客户集合和在银行有贷款的客户集合。用 SQL 表达式分别表达这两个集合如下：

```
Select customer-name
From   depositor
Select  customer-name
From borrower
```

1. 并操作

为了找出“在银行有账户、贷款或两者兼具的所有客户”，可写出：

```
Select  customer-name
From depositor
union
Select  customer-name
From borrower
```

与 Select 子句不同，union 操作自动去除重复。如果想保留所有重复，必须用 union all 代替 union。

2. 交操作

为了找出在银行同时有账户和贷款的客户，可写出：

```
Select  customer-name
From depositor
intersect
Select  customer-name
From borrower
```

intersect 操作也是自动去除重复的。如果想保留所有重复，必须用 intersect all 代替 intersect。

3. 差操作

为了找出在银行有账户但无贷款的客户，可写出：

```
Select  customer-name
From depositor
except
Select  customer-name
From borrower
```

except 操作自动去除重复。如果想保留所有重复，必须用 except all 代替 except。

2.2.8 统计查询

有很多用户的请求需要对数据库内容进行计算才能得到结果，即需要统计数据后得到的值。SQL 支持这些统计数据的请求，这是通过聚集函数和 Select 语句中的 group by 和 having 来完成的。

1. 聚集函数

聚集函数是以一个值集合为输入，返回单个值的函数。SQL 提供了以下 6 个预定义聚集函数：

- 求某一列值的平均值：avg()
- 求某一列值的最小值：min()
- 求某一列值的最大值：max()
- 求某一列值的总和：sum()
- 计算元组个数：count()
- 计算某一列值的个数：count(*)

聚集函数的自变量可以是简单的字段名，也可以是 SQL 表达式。

例如，求出销售人员的销售量占销售目标的百分比的平均值。

```
Select avg(100*(sales/quota))
From salesreps
```

sum 和 avg 的输入必须是数字，而其他函数还可作用在非数字数据类型如字符串、日期 / 时间上。

经常使用聚集函数 count 计算一个关系中数据值或元组的个数。

例如，求资产超过 $2 100 000 的银行分行个数。

```
Select  count(assets)
From branch
Where  assets>2 100 000
```

注意： count() 函数忽略了数据项的值，只是简单地计算有多少数据项。所以，它实际上不关心指定那个字段作为 count() 函数的自变量。

因此，SQL 支持一个特殊的聚集函数 count (*)。该函数计算行(元组)而不是数据值。所以，上面的查询可重写为：

```
Select count (*)
From branch
Where  assets>2 100 000
```

SQL 不允许在用 count(*)时使用 distinct。在用 max 和 min 时使用 distinct 是合法的，尽管结果是一样的。

在 SQL 中是不允许对聚集函数进行复合的。因而，max(avg(…)) 等是不允许的。

2. 分组查询

有时候不仅希望将聚集函数作用在单个元组集上，而且也希望将其作用在一组元组集上。SQL 中可以用 group by 子句实现这个功能。在 group by 子句中的一个和多个属性是用来构造分组的。group by 子句中所有属性有相同值的元组放在一个组中。

例如，查询“找出每个分支机构的账户结算平均额”，该查询书写如下：

```
Select branch-name, avg(balance)
From account
Group by branch-name
```

像上例这样，在计算平均值时保留重复元组是很重要的。而有些情况下在计算聚集函数前需先删掉重复元组，使用关键词 distinct。

如“找出每个分支机构储户数”，在该例中不论一个客户有几个账户，作为一名储户只计算一次，查询就该这样书写：

```
Select branch-name, count (distinct customer-name)
From depositor, account
Where depositor.account-number = account.account- number
```

3. 分组查询条件

与 Where 子句能在查询中选择和排除单个元组的功能一样，having 子句也可用于选择

和排除一个元组集。而且有时，对分组限定条件比对元组限定有用。例如，我们也许只对账户平均结算大于$1200 的分支机构感兴趣。该条件并不是针对单个元组，而是针对 group by 子句形成的分组。为了表达这样的查询，使用 SQL 的 having 子句。

having 子句中的谓词在形成分组后才起作用，因此可以使用聚集函数。用 SQL 表达该查询如下：

```
Select branch-name, avg(balance)
From account
Group by branch-name
having avg(balance)>1200
```

如果在同一个查询中同时存在 Where 子句和 having 子句，那么首先应该用 Where 子句中的条件表达式。满足 Where 条件表达式的元组通过 group by 子句形成分组。having 子句若存在，将作用于每一分组。

2.2.9 嵌套子查询

SQL 提供嵌套子查询机制。子查询是嵌套在另一个查询中的 Select-From-Where 表达式，一般子查询的使用是为了对集合的成员资格、集合的比较等进行测试。

1. 集合成员资格

SQL 运用连接词 in 来测试集合中的成员，该集合是由 Select 子句产生的一组值的集合。 连接词则是测试非集合中的成员。

例如，再考虑查询“找出在银行中同时有账户和贷款的客户”，可用以下表达式：

```
Select distinct customer-name
From borrower
Where customer-name  in
(Select customer-name
From  depositor)
```

在这里，in 和 not in 操作符的使用形式和前面讲到的简单测试类似。

2. 集合的算术比较

集合成员算术比较的形式如下：

```
元组  θ    some(集合)
元组  θ    all  (集合)
```

这里，θ 是算术比较运算符：<，<=，>，>=，=，< >。“θ some”操作表示左边的那个元组与右边集合中至少一个元组满足 θ 运算。“θ all”操作表示左边的元组与右边集合中所有元素都满足 θ 运算。

考虑查询“找出那些总资产至少比位于 Brooklyn 的某一家分支机构要多的分支机构名字”，在前面将此查询写作：

```
Select distinct T.branch-name
From branch as T, branch as S
Where T.assets > S.assets and S.branch-city ="Brooklyn"
```

SQL 提供另外一种方法重写上面的查询。短语“至少比一个要大”在 SQL 中用

“>some”表示。这一结构允许重新组织上面的查询，书写如下：

```
Select   branch-name
From branch
Where   assets > some
(Select assets
From   branch
Where  branch-city= "Brooklyn")
```

注意： in 与“= some”等价，但“not in”不等价于“<> some”，而是与“<> all”等价。

3. 空关系测试

SQL 还有一个特性可测试一个子查询的结果是否有元组。其形式如下：

```
EXISTS(集合)
NOTEXISTS (集合)
```

前一个操作，当作为子查询结果的集合非空时返回 TURE，否则返回 FALSE。后一个操作则情况相反。

例如，可以用另外一种方法表示“找出在银行既有账户又有贷款的客户”如下：

```
Select customer-name
From borrower
Where exists
(Select *
From depositor
Where depositor.customer-name = borrower.customer-name)
```


注意： 搜索条件根本不使用真正子查询的结果，它仅仅测试一下子查询是否产生一些结果。因此，SQL 允许子查询不必一定返回某单字段数值，而是可使用 Select * 的形式。

另如，查询“找账户平均结算大于$1 200 的分支机构”可表达为：

```
Select branch-name, avg (balance)
From account
Where not exists
( Select *
From  account
Where avg (balance) <=1 200)
Groupby branch-name
```

4. 元组的测试

可以用结构来测试一个集合中是否有重复元组的存在。形式如下：

```
unique(集合)
nounique(集合)
```

对于前一个操作，当作为子查询结果的集合中不存在重复元组时，其返回值为.T.，否则为.F.。后一个操作则相反。

例如，可将查询语句“找出在 Perridge 分行中只有一个账户的客户”书写如下：

```
Select T  customer-name
From depositor as T
Where unique
(Select R  customer-name Fromaccount, depositor as R
Where T  customer-name = R  customer-name and  R
Account-number=account.account-number  )
```

2.2.10 SQL 查询示例

1. 示例一

取出一个表中满足过滤条件记录的所有字段，分别进行处理，表名在程序运行中由字符串变量传递，字段信息是不确定的。这里假设通过字符串变量中传递的表名是 base。

```
//定义 4 个变量
    STRING lsString,lsSQL,lsTable,lsColumn
INT liInt
DATETIME liTime
//编写有效的 SQL 语句
    LsSQL="SELECT * FROM base WHERE code like ?"
//使用全局变量 SQLCA 作为动态策略区变量
DECLARE cursor_base DYNAMIC CURSOR FOR SQLSA;
    PREPARE SQLSA FROM lsSQL;
//使用全局变量 SQLDA 作为动态描述区变量
    DESCRIBE SQLSA INTO SQLDA;
//SQLDA 中含有输入参数的描述
//传递参数值
    SetDynamicparm(SQLDA,1,"32%")
    OPEN DYNAMIC cursor_base USING DESCRIPTOR SQLDA;
//读取数据
    FETCH cursor_base USING DESCRIPTOR SQLDA;
//检查 SQL 语句执行是否成功
    DO WHILE SALCA.SQLCODE=0
       FOR liInt=1 TO SQLDA.NumOutPuts
       CHOOSE CASE SQLDA.OutParmType[liInt]
       CASE Typestring!
            lsString=GetDynamicString(SQLDA,liInt)
     //处理该字符型的字段
     CASE TypeDateTime!
                 LsDateTime=GetDynamicDateTime(SQLDA,liInt)
     //处理该日期型的字段
          ...
//处理其他类型的字段
                ...
        END CHOOSE
        NEXT
//将一条记录的所有字段取完后作相应的处理
        FETCH cursor_base USING DESCRIPTOR SQLDA;
    LOOP
    CLOSE cursor_base;
```

DynamicDescriptionObject 是 PowerBuilder 提供的一个数据类型，在 PowerBuilder 中提供了一个 DynamicDescriptionObject 类型的全局数据类型 SQLDA，用来存放动态 SQL 语句的输入/输出参数。

2. 示例二

在 ls_sql 中准备一个查询 auths 表的 SELECT 语句，把查询结果返回给 PowerBuilder

变量(用游标处理结果集):

```
//定义 3 个变量
string name,ls_sql,code
code="A%"
//编写有效的 SQL 语句
ls_sql="SELECT auths_name FROM auths WHERE auths_id like ?"
//使用全局变量 SQLSA 作为动态策略区变量
//定义游标为 c1
DECLARE c1 DYNAMIC CURSOR FOR SQLSA;
PREPARE SQLSA FROM :ls_sql;
OPEN DYNAMIC c1 USING :code;
Lable:
FETCH c1 INTO :name;
If SQLCA.SQLCode=0 then    //如果成功取出记录
   //进行相应处理
...
   goto lable              //取下一条
end if
CLOSE c1;              //关闭游标
```

3. 示例三

下面是一个使用动态 SQL 格式 4 的例子，此例有一个字符串输入参数，但输出参数的类型和个数不定。

```
//定义 2 个变量
    string ls_sql,ls_val
    integer i,li_boolean
//编写有效的 SQL 语句
    ls_sql="SELECT * FROM auths WHERE auths_id=?"
//使用全局变量 SQLSA 作为动态策略区变量
//定义游标为 c1
    DECLARE c1 DYNAMIC CURSOR FOR SQLCA;
    PREPARE SQLSA FROM :ls_sql;
    DESCRIBE SQLSA INTO SQLDA;
//若 DESCRIBE 成功，则输入描述符数组将包含一个输入描述符，
//在打开游标前必须先给输入描述符赋值。
    SetDynamicParm(SQLDA,1,'A00001')      //这里的"1"表示第一个参数
    OPEN DYNAMIC c1 USING DESCRIPTOR SQLDA;
    lable:
      FETCH c1 USING DESCRIPTOR SQLDA;
    IF SQLCA.SQLCODE=0 THEN //若 FETCH 成功
      FOR i=1 to SQLDA.NumOutputs
//SQLDA.NumOutputs 含有输出参数的个数
//针对对不同的情况作不同的操作
       CHOOSE CASE SQLDA.OutParmType[i]
       CASE TypeString!
            ls_val=GetDynamicString(SQLDA,i)
       CASE TypeDate!
            ls_val=string(GetDynamicDate(SQLDA,i),'dd mm yyyy')
       CASE TypeTime!
            ls_val=string(GetDynamicTime(SQLDA,i),'hh:mm:ss')
       CASE TypeDataTime!
            ls_val=string(GetDynamicDateTime(SQLDA,i),&
                          'dd mm yyyy hh:mm:ss')
       CASE TypeInteger!,TypeLong!,TypeDecimal!,TypeDouble!,TypeReal!
            ls_val=string(GetDynamicNumber(SQLDA,i))
       CASE TypeBoolean!
            li_boolean=GetDynamicNumber(SQLDA,i)
```

```
            IF li_boolean=1 THEN
               ls_val="TRUE"
            ELSE
               ls_val="FALSE"
            END IF
         END CHOOSE
         …… //进行相应处理
      NEXT
      GOTO lable  //取下一条记录
      END IF
      CLOSE c1; //关闭游标
```

4. 示例四

下面介绍一个用单表单查询条件生成的通用窗口的实现，在该窗口的实现中使用了动态 SQL 格式四。

该窗口是一个响应式窗口，在调用该窗口时传递所要查询的表名给此窗口，在字段名下拉列表中显示出此表中的所有字段，选定字段名后，在数值下拉列表中显示出选定字段的内容，单击“确定”按钮，生成查询条件并返回。下面是脚本。

定义窗口的实例变量：

```
string  tablename   //用于存放表名
string  s_field      //用于存放选择的字段名
string  typename    //用于存放所选字段的数据类型
```

窗口 OPEN 事件中的脚本：

```
tablename=message.stringparm
//定义 CURSOR(游标)将表名为 TABLENAME 的所有列从系统表" PBCATCOL"中取出，
//并显示在下拉列表中
int i
string str
declare c1 cursor for select PBC_HDR from PBCATCOL
       where PBC_TNAM=:tablename;
open c1;
fetch c1 into :str;
do while sqlca.sqlcode=0
ddlb_1.additem(str)
i=i+1
fetch c1 into :str;
loop
close c1;
```

字段名下拉列表 SELECTCHANGED 事件中的脚本：

```
string s_ddlbtext,sql
s_ddlbtext=string(ddlb_1.text)
SELECT pbc_cnam  INTO :s_field FROM pbcatcol
   WHERE pbc_tnam=:tablename and pbc_hdr=:s_ddlbtext;
ddlb_2.reset()
//定义 CURSOR(游标)将字段名为 S_field 的字段值从表"tablename"中取出，
//并显示在下拉列表中(使用了动态 SQL 格式四)
string str
sql="select distinct "+s_field+" from "+tablenamesql
prepare sqlsa from : sql;
describe sqlsa into sqlda;
declare c1 dynamic cursor for sqlsa;
open dynamic c1 using descriptor sqlda;
```

```
if sqlca.sqlcode<0 then
    messagebox("提示信息","动态游标无法执行!"+sqlca.sqlerrtext)
    return sqlca.sqlcode
end if
lab1:
  fetch c1 using descriptor sqlda;
if sqlca.sqlcode=0 then
        choose case sqlda.outparmtype[1]
            case typestring!
                typename="string"
                str=getdynamicstring(sqlda,1)
            case typedate!
                typename="date"
                str=string(getdynamicdate(sqlda,1),"yyyy-mm-dd")
            case else
                typename="number"
                str=string(getdynamicnumber(sqlda,1))
        end choose
        ddlb_2.additem(str)
   goto lab1
end if
close c1;
```

“确定”按钮的 CLICKED 事件的脚本：

```
string wheresql
if string(ddlb_1.text)<>"" and string(ddlb_2.text)<>"" &
   and string(ddlb_3.text)<>"" then
     CHOOSE CASE typename
          CASE "string"
             wheresql=s_field+" "+ddlb_3.text+"
'"+string(ddlb_2.text)+"'"
        CASE "date"
             wheresql=s_field+" "+ddlb_3.text+"
date("+string(ddlb_2.text)+")"
         CASE "number"
             wheresql=s_field+" "+ddlb_3.text+" "+ddlb_2.text
      END CHOOSE
       closewithreturn(parent,wheresql)
else
       messagebox("提示信息","请正确输入字段名、运算符和数值！")
    return 1
end if
```

2.3　数据库修改

下面介绍如何用 SQL 语句，删除、插入和更新数据库信息。

2.3.1　删除

删除请求的表达与查询非常类似。可以删除整个元组，但不能只删除某些属性上的值。其格式如下：

```
delete From R
Where F
```

其意义是：从关系 R 中删除满足条件 F 的元组。如果省略 F 语句，则 R 中所有元组都被删除。

注意： delete 命令只对一个关系起作用。如果想从多个关系中删除元组，就要为每个关系写一条 delete 命令。

delete 子句中的条件可以嵌套，可以访问任意数目的关系。

下面是一些例子：

(1) 删除 Perryridge 分支机构的所有账户：

```
Delete From account
Where branch-name ="Perryidge"
```

(2) 删除位于 Needlham 的每一个分支机构的所有账户：

```
Delete From
Where branch-name in
(Select branch-name
From branch
Where branch-city = "Needlham")
```

(3) 删除余额低于银行平均余额的账户记录：

```
Delete From account
Where balance <
(Select avg(balance)
From account)
```

2.3.2 插入

要在关系中插入数据，可以指定被插入的元组(单个或多个)，或者是用一条查询语句，该语句的查询结果是希望被插入的元组集合。显然，插入元组的属性值必须在属性域中。

下面是几个例子：

假设要插入的信息是 Perryridge 分支机构中余额为 $ 1200 的账户 A-9732，可写成：

Insert into account(branch-name, account-number, balance)

values("Perridge","A-9732",1200)

如果插入元组值的个数、排列顺序与关系的结构完全对应，account 后面的属性序列可以省略。

如果插入元组中只有部分属性被赋值，关系 account 后的元组序列不可省略，而且其余默认的属性将被赋为 null。

如果想要插入多个指定的元组，可不必用多个 insert 语句，而是用“表”表达式来处理。例如，往关系 depositor 中连续插入 3 个元组，可表示为：

```
insert into depositor
(table (Hayes,A-102)
(Jones,A-217)
(Joy, A-715))
```

更多的情况是在查询结果的基础上执行插入。假设 Perryidge 分行想给每个在该行贷款的客户赠送一个价值 200 的新存款账户，并以贷款号作为新存款账户的账号，可写作：

```
insert into account
Select branch-name,loan-num,200
From loan
Where branch-name="Perryidge"
```

这里先执行 Select 子句，选出所有要插入的元组，再执行 insert 子句。这个顺序是非常重要的。

2.3.3　更新

当需要修改关系中元组部分属性的值时，可用 update 语句实现。其结构如下：

```
update R
set column1=A1,column2=A2,…
Where  F
```

其意义是：修改关系 R 中满足条件表达式 F 的元组中的指定属性值，修改的值在 set 子句中给出。含条件表达式 F 的 Where 子句，可以是任何类型合法的 Where 子句。

例如，想要执行“为所有存款大于平均数的账户增加 5%的利息”可以写为：

```
update account
set balance = balance*1.05
Where balance>(Select avg(balance)
From account)
```

和 delete，insert 语句类似，首先根据 update 语句中嵌套的 Select-From-Where 语句检查所有元组是否应该被更新，然后才执行更新。

2.4　实例：利用 SQL 操作进行身份验证

本节主要是利用 SQL 操作创建一个用户身份验证窗口，使每次进入该应用系统的时候会自动提示并验证用户的身份，以达到安全管理的目的。

该用户身份验证模块所需要实现的功能如下：

(1)　当用户进入系统的时候，首先要进行管理员的身份验证。用户初次使用该系统的时候，由系统设计者提供系统管理员的编码及口令。利用该信息进入系统以后，需要将口令自行修改。另外，还要为每位管理员创建各自的编码和口令。

(2)　每次启动该系统，系统将显示当前时间，同时提示用户输入用户的编码及口令。用户按要求输入并确认后，系统检验该编码与口令是否正确。如果输入正确，则进入系统主窗口；反之，系统提示错误。

需要做的工作是：在数据库中创建管理员信息表——manager，建立窗体，设置、排列各个控件，并设置它们的属性，编写窗体和各个控件的事件脚本。下面具体说明各个步骤。

1.　在数据库中创建管理员信息表

在数据库中创建管理员信息表——manager，用以存储和检验管理员的编码和口令。这是身份验证模块的后台数据库。

2. 建立窗体

(1) 在 PowerBuilder 的工具栏中单击 New 按钮，在随后弹出的窗口中选择 PB Object 标签。

(2) 在如图 2-17 所示的 New 对话框中，选择 Window(窗体)，单击 OK 按钮。

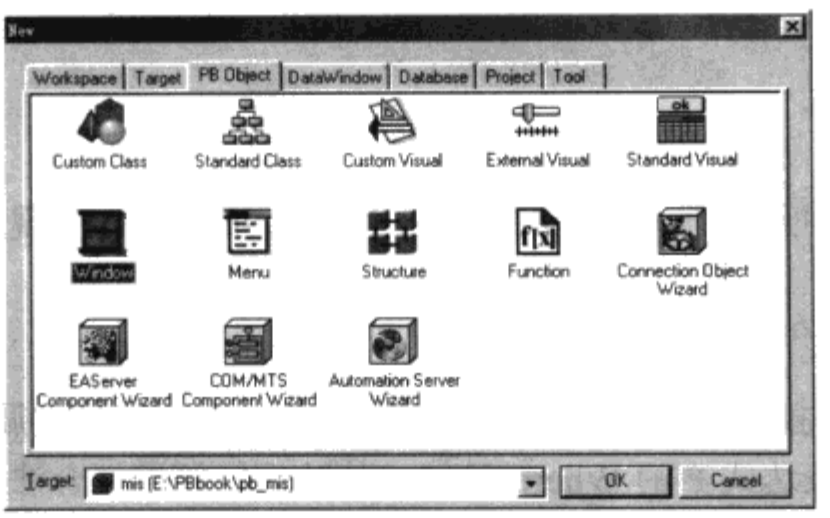


图 2-17 PB Object 标签内容

(3) 然后，PowerBuilder 会打开窗体的设计画板。给该窗体命名为 w_manager，把窗体的标题设为“身份验证”。

(4) 最后编写 Open 事件脚本。

程序运行后，首先打开此身份验证窗口。其 Open 事件脚本如下：

```
//定义 5 个全局变量
int iretlong
int imcode
string smpsd, smname
int flag
//取本机的系统日期和时间显示在窗口中
st_4.text=" 当前系统日期和时间: " +string (year(today()))+'年'&
+string (month(today()))+'月'&
+string (day(today()))  +'日'&
+string (hour(now()))  +'点'&
+string (minute(now())) +'分'&
+string (second(now())) +'秒'
st_5.text="(如果系统日期和时间有误，请修改系统时间后再进入本系统！)"
//设置光标
sle_1.setfocus()
```

3. 设置、排列各个控件

在该窗体中设置以下几个对象：

- 静态文本框：st_1~st_5。
- 单行文本框：sle_1、sle_2。
- 命令按钮：cb_1、cb_2。

设置和排列情况以及各自的属性效果具体如图 2-18 所示。

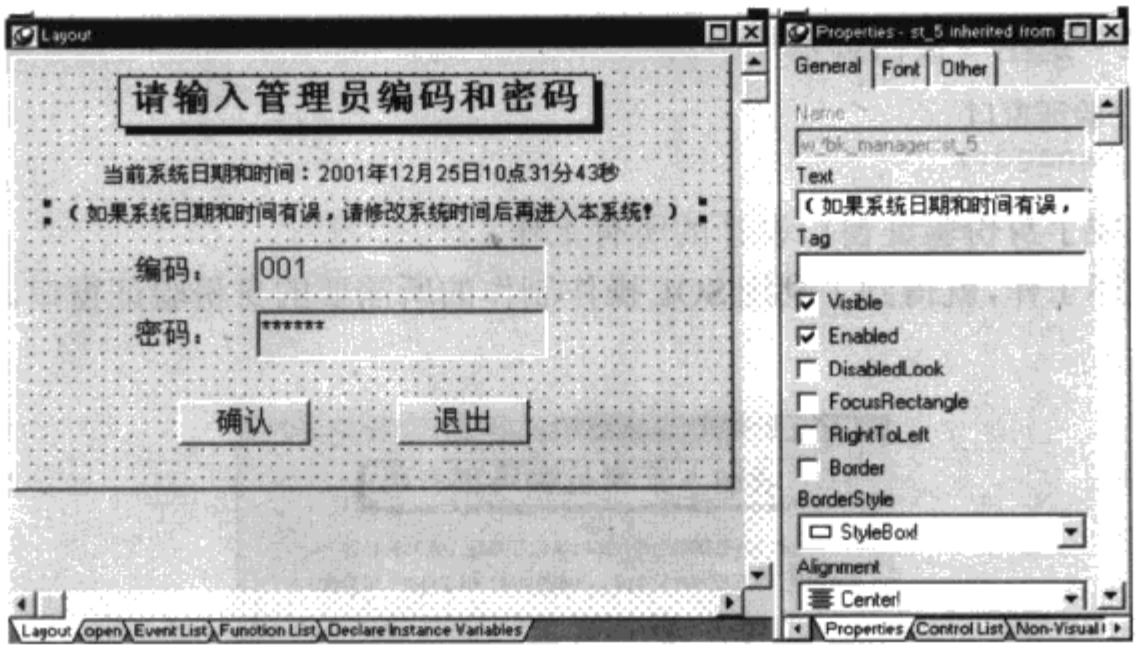


图 2-18　设置窗体 w_manager 及控件的效果

4.　编写窗体和各个控件的事件脚本

(1)　Cb_1“确认”按钮脚本

对身份验证的处理都将在这个事件脚本中完成。

```
//定义变量
int itemp
string stemp1,stemp2
//判断 sle_1 中是否输入内容
if IsNull(sle_1.text) or len(trim(sle_1.text))=0 then
messagebox('系统提示','请输入管理员编码',Exclamation!)
return
end if
//传送变量
itemp=integer(trim(sle_1.text))
stemp1=trim(sle2.text)
//比较代码及口令字
select mname
into :stemp2
from manager
where trim(mcode)=:itemp and trim(mpsd)=:stemp1;
//若输入错误
if SQLCA.sqlcode=100 then
messagebox("系统提示","输入错误,请重新输入!",Exclamation!)
return
end if
if SQLCA.sqlcode=-1 then
messagebox("系统提示","系统打开数据库时出现错误!",Exclamation!)
end if
//传送代码、口令字以及管理员的名字
smpsd=stemp1
imcode=itemp
smname=stemp2
//从数据表 manager 中取默认使用期限.该值可以在系统维护菜单中修改.
Select manager
Into :iretlong
From manager;
//打开程序主窗口,关闭口令验证窗口
open(w_main)
close(w_manager)
return
```

(2) Cb_2“退出”按钮脚本

```
//关闭口令验证窗口
close(w_manager)
```

至此，完成了身份验证窗口设计的所有工作。

完成了上述工作，就得到了利用SQL操作制作的所需要的身份验证窗口。效果如图2-19所示。

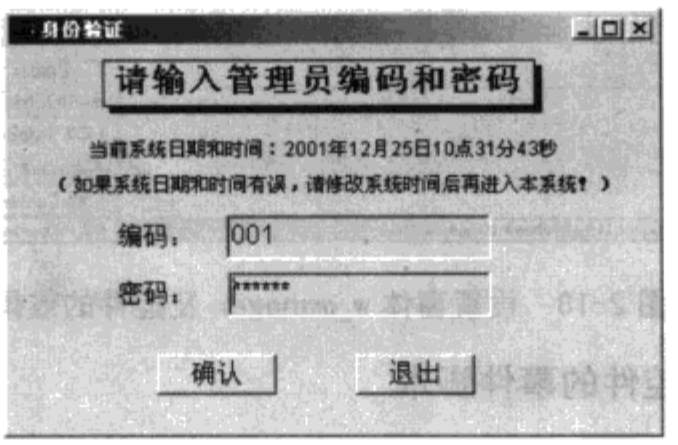


图 2-19 身份验证的实例效果

在这个窗口里填入管理员的编码和密码，单击“确认”按钮通过系统验证。如果出现错误，系统会给予提示信息。

本节中读者可以学习到利用SQL操作进行创建一个用户身份验证窗口的基本方法和步骤。完成了这项功能之后，当用户每次进入该应用系统的时候，系统就会自动提示并验证用户的身份，以达到安全管理的目的。通过这个实例，读者可以掌握身份验证模块的基本构建方法。

2.5 习 题

1. 填空题

(1) SQL 语言可以分为以下 3 部分：_____、_____、_____。

(2) 数据控制包括_____、_____、_____、_____。

(3) Select 子句对应关系代数中的_____，其中的属性名表指出要查询和列出的属性名集合。

(4) SQL 提供了一组丰富的搜索条件，包括_____、_____、_____、_____、_____。

(5) SQL 支持对统计数据的请求，这是通过_____和_____来完成的。

2. 问答题

(1) 如何创建 SQL SERVER 数据库？

(2) 如何备份和还原数据库？

第 3 章　PowerBuilder 编程基础

虽然 PowerBuilder 提供了许多优秀的工具(如数据窗口)，可以省去传统开发工具所需的大量编程过程，但是有时通过编程可以使应用程序功能更强大、使用更方便。PowerBuilder 提供了专用的编程语言 PowerScript 语言，这是一种高级的、结构化的编程语言，用它可以编写出各类函数和事件脚本程序，且编程过程也像其他高级语言一样容易方便。

本章将介绍 PowerScript 语言的语法基础知识及使用方法。

3.1　面向对象的编程特点

PowerBuilder 是高度可视化、面向对象的集成开发工具。在 PowerBuilder 中有许多种对象，例如应用程序、窗口、控件、菜单等，每一个对象就是一个应用程序组件，其中有许多称为属性的特性和称为方法(包括事件和函数)的行为。采用面向对象的方法可以大大缩减开发周期。

面向对象的开发方法的主要特点是：

(1)　继承性

继承性允许根据已经存在的祖先对象定义子孙对象，子孙对象继承其祖先对象定义的所有属性和方法。

(2)　封装性

对象的封装性使被封装的对象之间保持了最大的独立性，对一个对象的修改一般不会影响到其他对象。这种特性大大提高了对象的可重用性并减少了系统维护的代价。

(3)　多态性

多态性是指不同的对象可以对相同的消息做出不同的反应，它的主要作用是消息的发出者不需要知道接收者的确切消息就可以发送消息。

PowerBuilder 是事件驱动的应用程序，事件发生时所要处理的事情需要靠编写脚本来完成。PowerScript 与众不同之处在于它不仅支持标准的 SQL 语句，而且还提供了一套完整的嵌入式 SQL 语句，这就大大增强了程序操纵和访问数据库的能力。可以利用 PowerBuilder 提供的可视化工具生成嵌入式 SQL 语句，然后把它粘贴到程序中，而无须逐字输入。丰富的数据类型和众多的函数进一步提高了 PowerScript 的适应性和灵活性。PowerScript 不仅支持常规的数据类型，同时也支持用于处理多媒体的二进制对象数据，当然它也完全支持对象类型。总之，熟练掌握 PowerScript 语言是灵活高效地开发高质量 PowerBuilder 应用程序的基本要求。

3.2 PowerScript 语言基础

3.2.1 标识符,注释,代词

1. 标识符

标识符是程序中用来代表变量、标号、函数、窗口、菜单、控件、对象等名称的符号。标识符的命名规则为:

- 必须以字母或下划线开头。
- 可以由任何字母、数字及一些特殊符号(一、_、$、#、%)组成。不能是 PowerScript 保留字。
- 不区分大小写(PART、Part 和 part 是等价的)。
- 最长可以到 40 个字符,但中间不能包含空格。

例如,下面是一组合法的标识符:

```
rt                  //返回值
f_add               //函数
Button#1                 //按钮一
_abc                     //以下划线开头
```

下面的标识符是非法的:

```
2fa-quantity             //不是以字母或下划线开头
AXD Code            //中间有空格
Child'sId                //含有非法的特殊字符
```

注意: 短横线与减号是同一个字符,而短横线可以用在标识符中。因而,表达式中使用减号运算符时,必须在减号的两边各加上 1 个空格,否则可能产生语法甚至语义错误。

2. 注释

PowerScript 提供了两种加注释的方法:

一种是双斜线方法:

```
//注释
```

双斜线方法一般是为单行语句加注释,它可以是一整行,也可以在可执行语句后面,如下所示:

```
//decimal add1
amt = qty * cost  //给 amt 赋值
```

另一种是斜线一星号方法:

```
/*注释*/
```

斜线一星号方法一般用于加大段注释,它以“/*”开始,以“*/”结束,它可以是一行或多行,如:

```
/* flag = 1          表示按下的数字是前面数字的一部分
flag = 0            表示按下的数字是一个新的数字的开始    */
```

双斜线方法在工具栏上有对应的工具按钮，如图 3-1 所示。

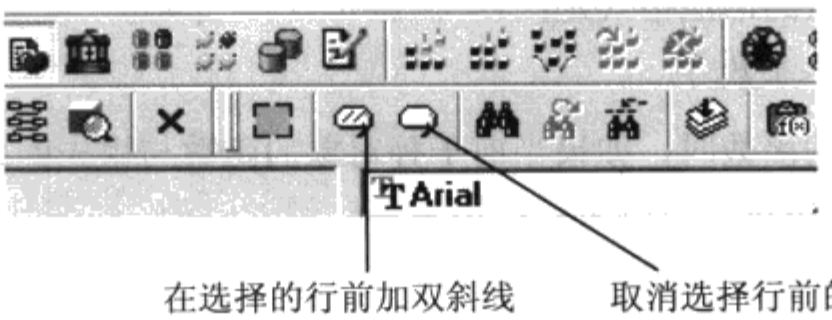


图 3-1　注释按钮

3. 代词

PowerBuilder 的代词用于指代特定的对象或控件。使用代词可以避免因对象或控件名被修改等情况而使引用出错，增加代码的通用性。PowerBuilder 提供的代词有 Parent、This 和 Super。

(1) Parent

Parent 指包含当前对象的对象，可以在窗口的控件、自定义用户对象、菜单等脚本程序中使用该代词。如窗口中某命令按钮的脚本若使用 Parent，则表示包含该控件的窗口；用户对象中控件的脚本若使用 Parent，则表示该用户对象；菜单脚本若使用 Parent，则表示该菜单项的上一级菜单项。

例如，窗口 w_1 上有命令按钮控件 cb_1，cb_1 的 Clicked 事件中有语句：

```
close(Parent)
```

当单击命令按钮 cb_1 时将同时关闭 w_1。

例如，在一可视用户对象 uo_1 上有复选框(CheckBox)cbx_1，在其 Clicked 事件下有语句：

```
Parent.Hide ()
```

在单击该复选框时将隐藏用户对象 uo_1。

例如，“输入”菜单项有下一级菜单项“禁止输入”，并且 Chicked 事件有语句：

```
Parent.Disable()
```

则当单击“禁止输入”菜单项时它的上一级菜单项“输入”被禁止。

(2) This

This 指具有当前脚本的对象，如窗口、用户对象、菜单、应用对象或控件。在对象或控件的脚本中，可以无限制地调用它的属性，在脚本中使用 This 可以使程序清晰明了、易于阅读。

有些情况下则必须使用 This，例如若全局/局部变量名与实例变量相同时，则 PowerBuilder 首先发现全局/局部变量，用 This 限制全局/局部变量便可以使用该实例变量。

例如：

```
This.Check()、ReCalc(This)、This.x = This.x + 50。
```

(3) Super

在编写后代控件或对象的脚本时，可以调用祖先的脚本。脚本程序中既可以直接利用祖先对象的名称调用它们，也可以使用 Super 代词来引用。

调用祖先对象函数的语法格式是：

```
Super::<函数名>(<参数>)
```

在调用祖先对象的函数时，如果当前子孙对象的函数与祖先的函数同名，则必须使用 Super 来限定调用祖先函数，否则可以不用限定。

调用祖先对象事件的语法格式是：

```
Call Super::<事件名>      或      Super::Event <事件名>
```

3.2.2 数据类型

数据类型是编程语言中的一个重要概念。一种编程语言所能支持的数据类型的多寡是其能力强弱的重要标志。PowerScript 的数据类型十分丰富，包括标准数据类型、Ang 数据类型、系统对象数据类型等。

1. 标准数据类型

表 3-1 是标准数据类型的列表。

表 3-1 标准数据类型

数据类型	示 例	说 明
Blob		该类型表示的是数量可变的二进制数据，如图像、文本等。特点是数据一般是变长的，而且字节数都比较大
Boolean	TRUE/FALSE	表示条件
Character	A，B	类型标识可略为 char
Date	1993-03-03	日期类型格式为 mm-dd-yyyy
DateTime	2000-13-02 19:30:25	常用做时间标记
Decimal	123.45	小数点可出现在 18 个数字中的任何位置，这 18 个数字中不计小数点和符号位，类型标识可略为 dec
Double	3.53e19	精度为 15 位小数
Integer	321	类型标识可略为 int
Long	12345678	
Real	3.25	精度为 6 位小数
String	“ computer ”	每个字符可为任一 ASCII 字符
Time	12:23:34	
Unsigned Integer	654	无符号整型，类型表示可略为 uint
Unsigned Long	321654	无符号长整型，类型标识可略为 ulong

2. Ang 数据类型

PowerScript 中支持 Ang 数据类型。Ang 类型是一种可变的数据类型，程序中给它赋什么类型，它就变成什么类型。Ang 类型可以接收标准数据类型、结构、对象及数组等各种类型的数据。

可以使用函数 ClassName()查看 Ang 类型变量中保存的实际数据的类型。Ang 赋值的规则如下：

- 可以将任何内容赋给 Ang 变量。
- 从 Ang 变量赋值给兼容的类型时必须知道它的内容。
- 对于已知数据类型变量，应该避免使用 Ang 类型。

Ang 类型变量参加运算时，必须保证其值的数据类型正确且符合操作符的要求，例如，A、B、C 是 3 个 Ang 类型的变量，下述语句：

```
Ang C
C = A + B
```

在 A、B 的值类型能够同为数值或字符串时，语句能够正确执行；如果其中一个为数值，另一个为字符串，程序在运行时将出错。

3. 系统对象数据类型

系统对象数据类型是 PowerScript 特有的类型。在浏览器的 System 标签页中列出了所有的系统对象。在 PowerBuilder 应用中，可以操纵各类对象，如窗口、菜单、命令按钮等，也可以定义将这些对象作为数据类型的对象。通常不必把它们作为数据类型来考虑，只需在相应的画板中定义它们。

每个类型的对象为一个类。PowerBuilder 在类等级上管理它的系统对象。这些类组成祖先和后代的继承等级。

例如，定义窗口变量：

```
Window mywin
```

4. 枚举类型

枚举类型是 PowerScript 的一种特殊数据类型，它有以下两种用途：

- 作为函数的参数
- 指定对象或控件的属性

需要注意的是，用户不可以说明枚举类型的变量。

每个枚举类型的参数或属性都有一组固定的枚举值。枚举类型的值总是以叹号(！)结尾。如 Arrow!为 1，Beam!为 2，鼠标指针函数 SetPointer() 就可以调用它们，如 SetPointer(Arrow!)。

3.2.3　变量与常量

1. 变量

在程序代码中使用变量前必须先说明它的数据类型和名称。一个变量的类型可以是标准数据类型、结构或对象。

(1) 变量种类

PowerBuilder 的变量可以分为全局变量、实例变量、共享变量和局部变量。不同种类的变量有不同的作用域和用途。

① 全局变量

全局变量是在整个应用程序的任何地方都可访问的变量，它的作用域是整个应用程序。

② 实例变量

实例变量是某个特定类的实例化。可以在应用对象、窗口对象、用户对象或菜单对象的画板中定义实例变量，并在定义的范围中使用。

实例变量在它第一次打开与之相关联的对象时，被创建，在打开对象或事件之前，实例变量是不存在的。

实例变量可以被其对象的事件脚本或函数访问，此外，其他外面的对象也可以通过引用该对象来访问该实例变量。所有实例变量的默认值都是公共的(Public)。可以对其加以限定，在变量声明时加入 Private 前缀：

```
Private int RecordNumber
```

关键字 Private 使 RecordNumber 成为私有变量，除本身所在的窗口外，它的后代及其他任何外部窗口都不允许访问 RecordNumber。如果允许其后代访问而又限制其他窗口访问，则可以使用保护变量，其用法如下：

```
Protected int RecordNumber
```

关键字 Protected 和 Private 的用法如下：

ProtectedRead：允许其后代和所在的窗口读，但不允许其他窗口读取该实例变量。

ProtectedWrite：允许其后代窗口写，但不允许其他窗口修改该实例变量。

PrivateRead：只允许所在的窗口读，不允许后代及其他窗口读取该实例变量。

PrivateWrite：只允许所在的窗口写，不允许后代窗口及其他窗口对该实例变量进行写操作。

③ 局部变量

在相应的代码段定义和使用的临时变量，一旦代码段执行结束相应的变量就不复存在。

④ 共享变量

共享变量类似于实例变量，但不如实例变量那样用的频繁。属于一个对象但是可以在这个对象的所有实例中引用。当一个对象关闭时赋值给共享变量的值保持不变，再次打开这个对象时可以继续使用该值。

如图 3-2 所示，可以在声明窗口中选择变量的种类。

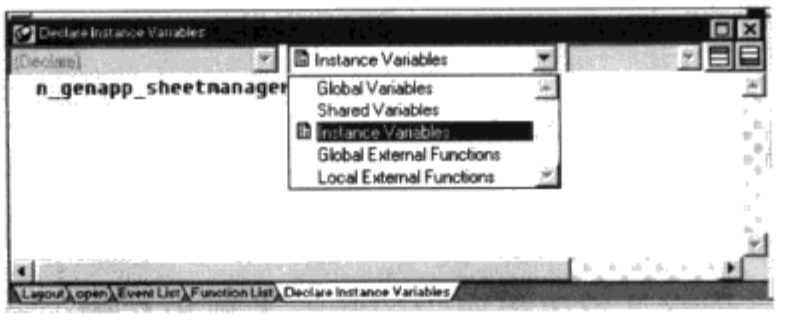


图 3-2 选择变量的种类

如果由于某种原因不同范围的变量使用了相同的名字，PowerBuilder 是按局部变量、共享变量、全局变量和实例变量的顺序来搜索变量的。

(2) 变量说明

变量说明的一般方式是：

数据类型 变量名

变量说明的完整格式如下：

```
{ access } Datatype {{ Size }}{{ Precision }} Variablename { = Value }{,
Variablename { = Value }}
```

其中参数：

- access：为实例变量专用，有 3 种选择：Public、Protected、Private。默认为 Public。
- Datatype：数据类型。可以为标准数据类型、系统对象或定义过的结构等。
- Size：为 Blob 类型专用，指定 Blob 变量以字节为单位的长度，默认该项时默认长度为 0，且系统会根据实际数据自动调整其长度。
- Precision：为 Decimal 类型专用，括在花括号中的数字指出了小数点后的位数，如果没有指定精度，则在程序中根据变量的赋值来确定精度。
- Variablename：变量名。可以定义多个同一类型的变量，它们由逗号分隔开，每个变量可以在此赋值。
- Value：变量的初值。Blob 不能有初值。

例如：

```
Integer ii_count = 100
```

2. 常量

与变量用法相同，只需在声明的同时赋予一个给定的值，而这个值在整个程序的运行过程中始终不变。用户可以根据需要定义任何标准类型的、任何作用域的常量。声明时前面加关键字 constant，例如：

```
constant real ii = 12.36
constant private int z_count = 1000
constant String Z_String = "This is a film"
```

3.2.4 数组

数组是一系列数据类型相同的元素的有序集合。PowerScript 可以创建任何数据类型的数组，一维数组的定义格式如下：

类型说明 数组名[下标]

多维数组的定义格式如下：

类型说明 数组名[下标，下标，下标，…]

例如：

```
int ii[3]
Real lr_rate[5,5]
```

在声明的同时可以对数组进行赋值，例如：

```
Integer li_stats[2]
Li_stats = {100 ,100}
```

或

```
Integer li_stats[2] = {100,100}
```

数组的引用方式为：

数组名[下标，下标，下标]

在通常的情况下，数组的元素是从 1 开始到最大下标排序的。但 PowerBuilder 允许使用不同的下标，例如：

```
Integer li_stats[-3 To 5]
Integer RecordNumber[-2 To 6,5 To 10]
```

PowerBuilder 允许使用可变数组。程序在运行的时候再来确定数组元素的个数。

注意： 可变数组只能是一维的，而且不能用 To 短语来说明或改变数组的下界，可变数组的下界永远是 1。

3.2.5 运算符

PowerScript 支持 4 类运算符：算术运算符、关系运算符、逻辑运算符和字符串连接运算符。

1. 算术运算符

PowerScript 使用表 3-2 中所列的算术运算符来对数字变量和常量进行运算。

表 3-2 算术运算符

运 算 符	说 明
+	加
-	减
*	乘
/	除
^	幂
++	增 1
--	减 1
+=	加并赋值
-=	减并赋值
*=	乘并赋值
/=	除并赋值
^=	乘方并赋值

2. 关系运算符和逻辑运算符

表 3-3 列出了 PowerScript 的关系运算符和逻辑运算符。

表 3-3　关系运算符和逻辑运算符

运　算　符	说　　明
>	大于
<	小于
>=	大于等于
<=	小于等于
=	等于
<>	不等于
AND	逻辑与
OR	逻辑或
NOT	逻辑非

3. 字符串连接运算符

字符串连接操作是指将两个或多个字符串用拼接连接成一个字符串的操作，PowerScript 在字符串操作时用“+”表示拼接，例如：

```
String Name
Name = "北京"+"大学"
```

最后 Name 的内容是“北京大学”。

4. 操作符的优先级

不同的运算符在表达式中的优先级如表 3-4 所示。

表 3-4　操作符的优先级

运　算　符		说　　明
优先级 高	()	括号
	+,-	正号和负号
	^	乘方
	*,/	乘和除
	+,-	加和减，字符串连接
	=,>,<,<=,>=,<>	关系运算符
	NOT	逻辑非
	AND	逻辑与
低	OR	逻辑或

3.2.6　表达式

在计算表达式时，如果存在多种类型，PowerBuilder 将把它们转化为表达式中已有的级别最高的数据类型。数值数据类型的级别如下：

PowerBuilder 中的 String 型数据赋值给 Char 型时，Char 型数据得到字符串中的第一个字符，例如：

```
Char c ="xyz"
```

结果变量 c 被赋值为字符 xyz。

使用字符串值可以将特殊字符(如回车、换行等)赋值给字符变量，例如：

```
Char c ="~ n"
```

结果变量 c 被赋值为控制字符换行。

对于动态 Char 型数组可以直接赋值，对于定长数组，如字符串超长，则截去多余部分。当数组长度大于字符串长度时，在字符串后自动加终止符。Char 型数组也可赋值给 String 型变量，如果在 Char 数组中存在字符串终止符，复制仅进行到终止符。

3.3 PowerScript 编程基础

3.3.1 控制结构

1. 条件分支结构

PowerBuilder 的条件语句共有两类：If 语句和 Choose Case 语句。If 语句先计算一个逻辑表达式，然后根据表达式的值是 True 还是 False 来决定下一步执行的语句块；Choose Case 语句可同时处理多种情况。本节将介绍条件语句的结构和用法。

(1) If 语句

① If...Then 结构

If 后面是条件，它们是由等式或不等式组成的语句，其形式为：

```
If  <表达式> 关系符 <表达式> Then
语句块
End If
```

条件表达式也可以用逻辑运算符连接，构成各种相对复杂的条件。常用的逻辑运算符为：逻辑与(AND)、逻辑或(OR)、逻辑非(NOT)。

② If...Then...Else 结构

If...Then 语句中的 Else 子句可以在条件不满足的情况下去执行 Then 语句块以外的语句或语句块，其形式为：

```
If  <表达式> 关系符 <表达式> Then
    语句块 1
Else
    语句块 2
End If
```

③　If...Then...ElseIf...Else...EndIf 结构

有的时候，情况是很复杂的或为了程序的可读性，可以使用这个结构，其形式如下：

```
If  <表达式> 关系符 <表达式> Then
    语句块 1
ElseIf  <表达式> 关系符 <表达式> Then
    语句块 2
ElseIf  <表达式> 关系符 <表达式> Then
    语句块 3
Else
    语句块 4
End If
```

例如，下面的示例将学生成绩的百分制转换成 A、B、C、D、E 等级制：

```
If  PercentCorrect >= 90 Then
Grade="A"
ElseIf  PercentCorrect >= 80 AND PercentCorrect < 90  Then
Grade="B"
ElseIf  PercentCorrect >= 70 AND PercentCorrect < 80  Then
Grade="C"
ElseIf  PercentCorrect >= 60 AND PercentCorrect < 70  Then
Grade="D"
Else
Grade="E"
End If
```

(2)　Choose...Case 分支结构

Choose...Case 语句根据测试表达式的值选择不同的操作，其形式为：

```
Choose Case      测试的表达式
Case          表达式表
              语句块
{ Case   表达式
              语句块
…
Case     表达式}
{ Case Else
              语句块}
End Choose
```

其中大括号中是可选项，例如：

```
Choose Case      num
Case 6
num = 7
Case is> 36
num = 36
Case Else
num = 56
End Choose
```

2.　循环结构

PowerScript 提供了循环语句来处理重复的操作，它的循环语句有 3 种形式：For...Next 循环、Do 循环和 GoTo。下面将分别介绍这三种循环语句，同时也介绍与循环有关的语句 Exit 和 Continue。

(1) For...Next 循环

For...Next 是一个计数循环体。使用该语句可以使循环体中的语句被执行规定的次数。该语句的语法格式为：

```
For 变量名 = 初值 To 终值 [step 步长]
语句块
Next
```

例如，下面的示例给数组 array1[]赋值：

```
integer array1[16]
For I = 1 to 16
array1[I] = I
Next
```

(2) Do 循环

Do...Loop 结构提供了 4 种基于条件的循环。它以 Do 作为循环的开始，以 Loop 作为循环的结束。其语法格式为：

① Do Until...Loop

```
Do Until 条件
    语句块
Loop
```

其执行过程为：先计算条件，如果结果为 False，执行语句块，接着再检查条件，直到条件的结果为 True 时退出循环。

② Do While...Loop

```
Do While 条件
    语句块
Loop
```

其执行过程为：先计算条件，如果结果为 True，执行语句块，接着再检查条件，直到条件的结果为 False 时退出循环。

③ Do...Loop Until

```
Do
    语句块
Loop Until 条件
```

其执行过程为：先执行语句块，然后计算条件，如果其结果为 False，则继续执行语句块，直到结果为 True 时退出循环。

④ Do...Loop While

```
Do
    语句块
Loop While 条件
```

其执行过程为：先执行语句块，然后计算条件，如果其结果为 True，则继续执行语句块，直到结果为 False 时退出循环。

这 4 种循环结构中，前两种结果可以一次也不执行，后两种则至少执行一次。在实际开发过程中，可以根据实际情况的需要，选择不同的循环方式。

(3) GoTo ,Continue ,Exit

① GoTo 语句

GoTo 语句控制程序流程，其语法格式如下：

```
GoTo Label
```

其中 label 代表语句标号，其命名应符合标号的命名规则。

例如，下面的示例用 GoTo 来实现一个循环：

```
Label1:
Num++
…
GoTo Label1      //跳到 Label1 处
```

② Continue 语句

在 Do...Loop 或者 For...Next 语句循环体当中，遇到 Continue 语句的时候，将不执行在 Continue 后面到循环结束之前的语句，开始新一轮循环。

例如在下面的例子中，如果 Num1 的值大于 36，则执行函数 Function1：

```
  For Num1 = 1 To 56
  If Num1 <= 36 Then Function1() Else &        //续行
  Function1()                          //如果 Num1 的值大于 36，则执行函数 Function1
Else
  continue
End if
  Next
```

③ Exit 语句

在 Do...Loop 或者 For...Next 语句循环体当中，使用 Exit 可以跳出循环。

例如：

```
For I = 1 To 36
If  Array[I]  > 10  Then Exit          //如果 Array[I]大于 10，跳出循环
Next
```

(4) 循环嵌套

循环嵌套就是把一个循环放在另一个循环的内部。PowerScript 中的 3 种循环方式都可以进行循环嵌套。下面通过一个实例来说明循环嵌套的工作方式。

```
Int Rm[8,5]                          //声明二维数组
Int i,j
For i = 1 To 8                       //外层循环
   For j = 1 To 5                    //内层循环
      Rm[i,j] = i + j
   Next                              //内层循环结束
Next                                 //外层循环结束
```

在程序运行的时候，首先进入外层的循环，此时 i ＝ 1，然后进入内层的循环。内层的循环结束后，外层循环开始下一次循环，然后又进入内层循环，最后整个循环运行结束，最后一个数组元素的值为 40。

3. 返回和终止

在编写事件处理程序或函数时，使用 Return 语句返回控制给用户或调用函数的地方。当希望终止应用程序的运行时，使用 Halt 语句。

(1) Return 语句

Return 语句用于从脚本中返回调用程序。其语法格式为：

```
Return {Expression}
```

其中 Expression 代表该脚本的返回值。

在程序中调用函数或事件处理程序时，执行到 Return 语句后，该语句立刻终止事件处理程序或函数的执行，并把控制返回到调用程序。

例如：

```
If Num>= 200 Then
   Return 0       //如果 Num >= 200，使用 Return 语句结束本函数，以后的语句不再执行
Else
   Return Num
End If            //使用 Return 语句结束本函数，以后的语句不再执行
```

(2) Halt 语句

Halt 语句用于终止应用程序的运行。该语句的语法为：

```
Halt {Close}
```

遇到不包含 Close 选项的 Halt 语句时，应用程序立即终止。当包含 Close 选项的时候，应先执行 Close 事件所对应的处理程序，然后再终止应用程序。

3.3.2 函数和结构

1. 使用系统函数

系统函数是 PowerScript 的通用函数，它们不隶属于任何对象，是 PowerBuilder 提供的内部函数，在应用程序的任何地方都可以不加说明而直接使用。系统函数使 PowerScript 的编程工作大为简化，这些函数一般包括下面各种类别。

(1) 数组函数

数组函数只有两个，用于得到数组的上界和下界，如表 3-5 所示。

表 3-5 数组函数

函 数	返回值	功 能
LowerBound(array {, n})	Long	得到指定数组第 n 维的下界
UpperBound(array {, n})	Long	得到指定数组第 n 维的上界

(2) Blob(大二进制对象)函数

Blob 函数得到 Blob 数据类型变量的信息、完成数据类型转换以及操作 Blob 类型数据，如表 3-6 所示。

表 3-6 Blob(大二进制对象)函数

函 数	返回值	功 能
Blob (text)	Blob	将字符串转换成 Blob 类型数据
BlobEdit (blobvariable, n, data)	Unsigned long	将任意类型的数据插入到 Blob 变量的指定位置

续表

函　数	返回值	功　能
BlobMid (data, n {, length})	Blob	提取 Blob 变量中的一段数据
Len (stringorblob)	Long	得到 Blob 类型变量的数据长度，以字节为单位
String (blob)	String	将 Blob 类型变量的值转换成字符串类型。如果 Blob 类型变量的值不是文本型数据，String()函数将其值解释成字符

(3)　数据类型检查与转换函数

完成数据类型检查与转换功能的函数如表 3-7 所示。

表 3-7　数据类型检查与转换函数

函　数	返回值	功　能
Integer (string or blob)	Integer	将字符串或 Blob 值转换成 Integer 类型的值
Long (lowword, highword)	Long	将两个无符号整数组合成一个长整数
Real (string or blob)	Real	将字符串或 Blob 值转换成 Real 类型的值
Double (string or blob)	Double	将字符串或 Blob 值转换成 Double 类型的值
Date (datetime)	Date	将 DateTime 类型或第一个值是 DateTime 或 Date 的 Blob 值转换成 Date 类型的值
DateTime (date {, time })	DateTime	把日期和时间值组合成 DateTime 类型的值
IsNull (any)	Boolean	测试变量或表达式的值是否是 NULL
String (data, {format})	Boolean	按指定格式将数据转换成字符串
Time (datetime)	Time	将 DateTime 类型或第一个值是 DateTime 或 time 的 Blob 值转换成 time 类型
Time (string)	Time	将其值为有效时间的字符串转换成 Time 类型

(4)　日期、时间函数

日期、时间函数如表 3-8 所示。

表 3-8　日期、时间函数

函　数	返回值	功　能
Day (date)	Integer	得到日期型数据中的号数(1～31 之间的整数值)
DayName(date)	String	得到指定日期是一周中的星期几(例如，Sunday, Monday，…)
Hour(time)	Integer	得到时间值中的小时，采用 24 小时制
Month(date)	Integer	得到日期值中的月份，有效值在 1~12 之间
Now( )	Time	得到客户机的当前系统时间，返回值为 Time 类型
RelativeDate(date, n)	Date	得到指定日期前多少天或后多少天的日期
Today( )	Date	得到系统日期和时间

(5) 文件操作函数

文件操作函数如表 3-9 所示。

表 3-9 文件操作函数

函 数	返回值	功 能
FileClose (fileno)	Integer	关闭文件
FileDelete (filename)	Boolean	删除文件
FileOpen(filename{,filemode{,fileaccess{ ,filelock ,writemode{creator,filetype}}}}))	Integer	打开文件
FileWrite (fileno, variable)	Integer	写文件
GetFileOpenName(title, pathname,filename {,extension {, filter}})	Integer	显示打开文件对话框，让用户选择要打开的文件

(6) 数值计算函数

数值计算函数如表 3-10 所示。

表 3-10 数值计算函数

函 数	返回值	功 能
Abs (n)	n 的数据类型	计算绝对值
Cos (n)	Double	余弦函数
Exp (n)	Double	计算 e 的 n 次方
Max (x, y)	x、y 中更精确的数据类型	求最大值
Mod (x, y)	x、y 中更精确的数据类型	求余数
Sign (n)	Integer	确定 n 是整数、负数还是零
Sqrt (n)	Double	计算平方根
Truncate(x,n)	Decimal	截断数值到指定的小数位

(7) 打印函数

打印函数用于在打印机上格式化输出数据。这组函数与各对象的 Print()函数有所不同，程序在使用这组函数时，需要使用 PrintOpen()函数打开一个打印作业，然后根据需要执行这组函数中的一个或多个函数，最后使用 PrintClose()函数关闭打印作业。

打印函数如表 3-11 所示。

表 3-11 打印函数

函 数	返回值	功 能
Print (printjobnumber,{,tab1} string {, tab2 })	Integer	以当前字体在打开的打印作业中打印一行或多行文本
PrintOpen({ jobname })	Long	启动打印作业并返回作业号
PrintClose (printjobnumber)	Integer	关闭打印作业并把打印页面发送到打印机上
PrintRect (printjobnumber, x, y, width, height, thickness)	Integer	在指定位置以指定线宽打印矩形

续表

函　数	返回值	功　能
PrintText (printjobnumber, string, x, y {,fontnumber})	Integer	在指定位置打印一行文本
PrintX (printjobnumber)	Integer	返回打印光标的 *x* 坐标位置

(8) 字符串操作函数

字符串操作函数如表 3-12 所示。

表 3-12　字符串操作函数

函　数	返回值	功　能
Asc (string)	Integer	得到字符串第一个字符的 ASCII 码整数值
Char (n)	Char	抽取字符串的第一个字符或把整数转换成字符
Lower (string)	String	将字符串中的大写字母转换为小写字母
Right (string, n)	String	从字符串尾部取指定个数字符
Space (n)	String	生成一个由空格组成的指定字符个数的字符串
Trim (string)	String	删除字符串首部和尾部的空格

这里只给出了一部分内部函数，对于内部函数的使用及其他的内部函数请读者查看 PowerBuilder 的参考手册。

2. 创建自定义函数(Fuction)

PowerScript 语言有许多内部函数，但有时开发人员根据所实现的功能还需要定义自己的函数。在 PowerBuilder 中可以自定义两类函数：全局函数和对象级函数。全局函数在整个应用程序中都能使用，而对象级函数则根据定义可以在整个程序中使用，也能够只在对象内部使用。自定义函数能够执行特定的任务。定义函数后，可以将函数保存在应用库中，这样在其他的应用中也可以使用这些函数。

(1) 创建自定义全局函数

使用函数画板创建自定义全局函数，其步骤为：

① 从菜单栏中选择 File“1” New，打开 New 对话框，在 PB Object 选项卡中选择 Function，系统将打开函数画板，如图 3-3 所示的 New Function 对话框。该对话框中各部分的意义为：

- Function Name 编辑框用于指定新建函数的名称，用标识符表示，通常以 f_开头(指示这是个用户自定义函数)，以便与系统内置函数相区别。
- Access 下拉列表框指定函数的访问范围，因为当前定义的函数是全局函数，因此该列表框不能选择。

- Return Type 下拉列表框用于指定函数返回值的类型，默认时为 Integer。
- 在 Pass By 下拉列表框中指定参数传递方式，共有 3 种方式：选择“value”时传值，将实际参数的值传递给函数；选择 reference 时传地址，把实际参数的地址传递给函数，此时，函数如果修改了形式参数的值，那么实际参数的值也就被修改了；选择“readonly”也是传地址，不过不允许修改参数的值。
- 在 Argument Type 下拉列表框中选择参数类型。
- 在 Argument Name 下指定参数名称。在 Argument Name 中单击右键，可以在弹出菜单中添加参数、插入参数或删除参数。
- Throws 用于声明触发事件异常，是可选项。

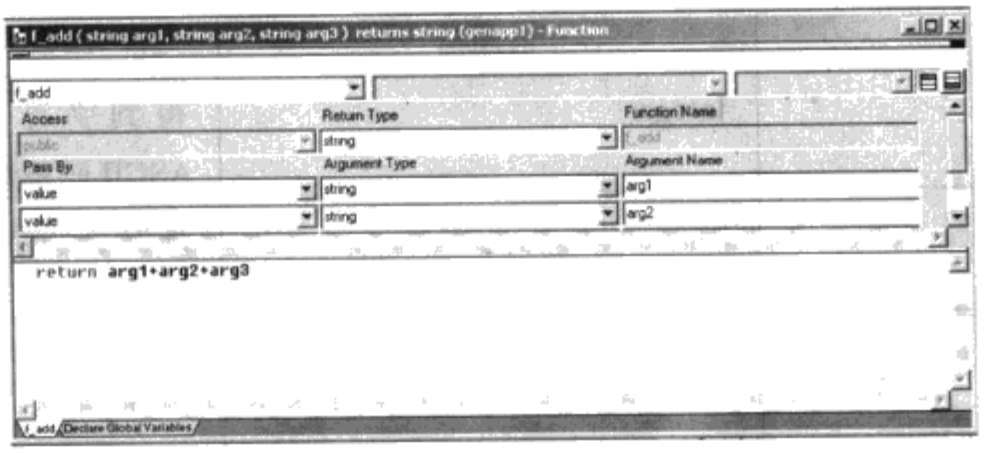


图 3-3　全局函数 New Function 对话框

② 添加代码。

③ 编写完代码后，单击函数画板工具栏上的返回(Return)图标，系统弹出 Save Function 对话框，输入函数对象名并选择要保存的应用库后，单击 OK 按钮完成自定义全局函数的创建。

例如，在程序中自定义函数 f_add，以 3 个字符串为参数并返回 3 个字符串的连接结果，具体的定义和调用如下：

① 从菜单栏中选择 File | New 命令，打开 New 对话框，并在 PB Object 选项卡中选择 Function，定义函数如图 3-3 所示。

② 在一个窗口(w_genapp1_sheet1)中添加一个单行文本编辑框(SingleLineEdits)。

③ 在 w_genapp1_sheet1 的 Open 事件中添写脚本如下，其中调用 f_add 函数：

```
string arg1,arg2,arg3
arg1="Welcome"
arg2=" to"
arg3=" PowerBuilder 10.0"
sle_1.text = f_add(arg1,arg2,arg3)
```

④ 程序的运行结果如图 3-4 所示。

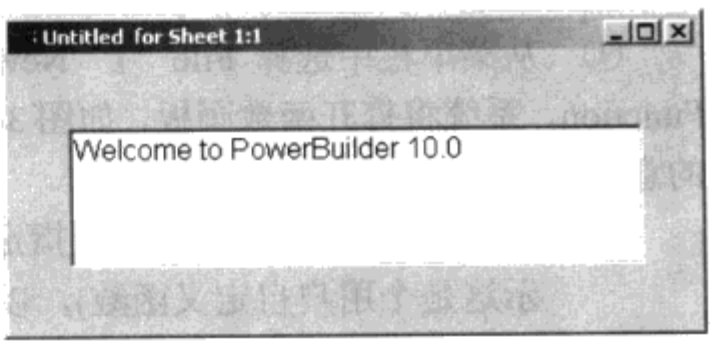


图 3-4　程序的结果

(2) 创建自定义对象函数

可以在应用对象画板、菜单画板、窗口画板或用户对象画板中创建自定义对象函数，这些函数与对象相关，并且与对象保存在一起。下面介绍创建窗口自定义对象函数的步骤，其他自定义对象函数的创建步骤可以依此类推。

创建窗口自定义对象函数的步骤为：

① 进入窗口画板，从 Declare 菜单中选择 Functions 菜单项，系统显示如图 3-5 所示的 New Function 对话框。该对话框与创建自定义全局函数时的 New Function 对话框只有一点不同：就是 Access 下拉列表框在这里可用，通过它指定自定义窗口函数的访问范围。当把 Access 设置为 public 时，该函数在整个程序中都可访问；当把 Access 设置为 private 时，该函数只能在附属于当前窗口的程序中使用；当把 Access 设置为 protected 时，该函数只能在附属于当前窗口的程序以及附属于该窗口的后继窗口的程序中使用。Access 的默认值为 public。New Function 对话框中其他部分的意义与创建自定义全局函数时的 New Function 对话框相同，在此不再重复。

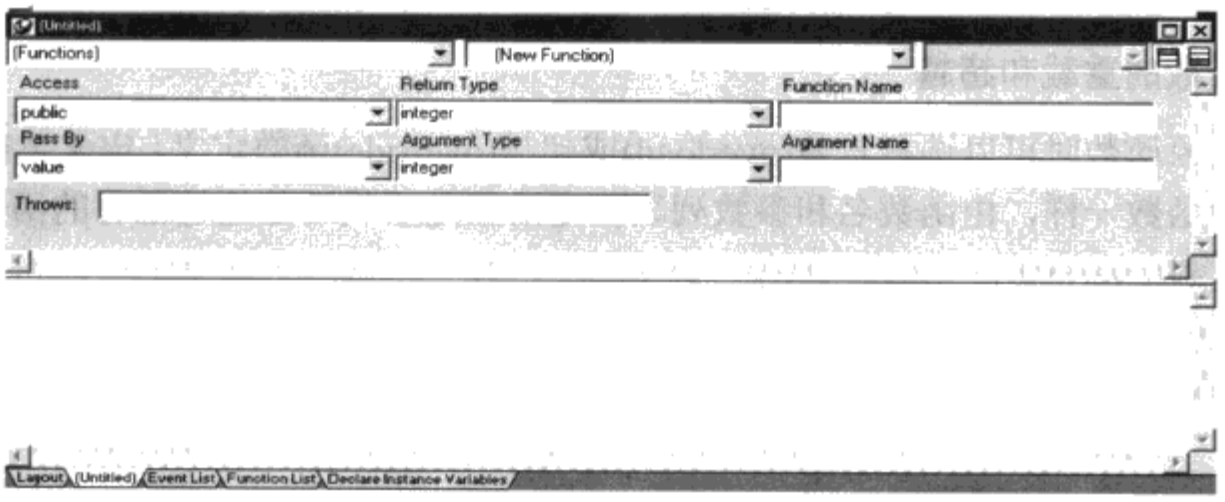


图 3-5　自定义对象函数 New Function 对话框

② 添加代码。

③ 编写完代码后，单击函数画板工具栏上的返回(Return)图标，系统弹出 Save Function 对话框，输入函数对象名并选择要保存的应用库后，单击 OK 按钮完成窗口自定义对象函数的创建。

例如，创建一个窗口函数，用于实现窗口的隐藏，具体的实现过程如下：

① 打开窗口画板，定义窗口函数如图 3-6 所示。

② 单击窗口画板中的 Layout，打开布局窗口。在窗口中添加一个命令按钮(CommandButton)，在窗口的属性页中把按钮的 Text 属性改为“隐藏”。

③ 在命令按钮的 Clicked 事件中添写如下脚本：

```
f_wpro(w_01)
```

④运行程序结果如图 3-7 所示。

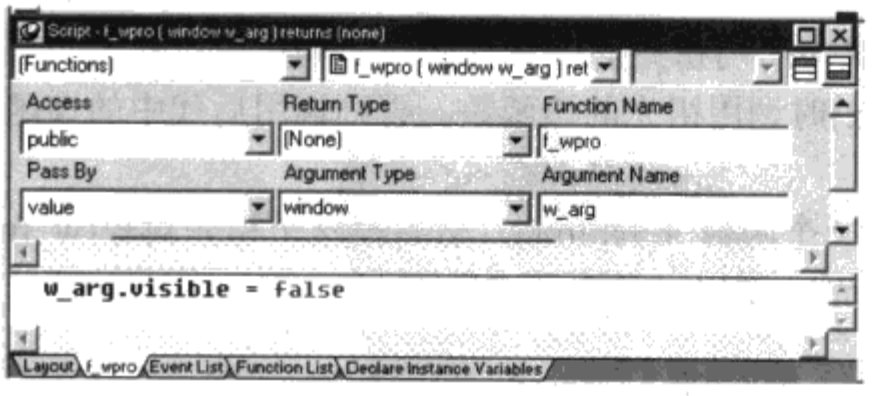


图 3-6　定义对象函数的对话框

图 3-7　程序的运行结果

⑤ 单击“隐藏”按钮，窗口消失。

3. 函数调用

在脚本中，调用内部函数和调用用户自定义函数的方法是一样的。用户可以在事件脚本或其他用户自定义函数中调用函数。在编程窗口中，可以手工输入函数名来调用函数，也可以通过粘贴功能来输入函数。有 3 种通过粘贴功能输入函数的方法：

- 从菜单栏中选择 Edit | Paste Special | Function 命令，然后从 Function 菜单中选择内部函数、用户自定义函数或外部函数。
- 选择使用 AutoScript 功能并在弹出的列表框中双击函数名。
- 在 Browser 中选择函数并将其粘贴至脚本中。

4. 函数的重载和超越

继承对象函数时可以选择重载(overload)或超越(override)函数定义。定义重载/超越函数与定义其他函数一样，由函数名和参数列表决定函数是重载还是超越已有的函数。

(1) 重载(overload)

重载意味着可以多个函数同名，但参数列表不同。额外的函数可以在同一对象中或该对象的后代中定义。PowerBuilder 比较函数调用的参数列表和原型，从而决定调用哪个函数。

当重载一个函数以便一个版本使用数值变量、另一个版本使用字符串变量时，对于函数，每个版本的调用参数是很明显的。使用不相关的数据类型的重载是很好的方法，可以提供应用程序所需的功能。

例如，定义两个函数：

```
f_add(integer arg1,integer arg2)
f_add(integer arg1,integer arg2,integer arg3)
```

在某一个对象中添写如下脚本：

```
if arg3 = 0 then
    f_add(arg1,arg2)                    //比较参数列表，如果是两个参数，调用第一个函数
else
    f_add(arg1,arg2,arg3)               //比较参数列表，如果是 3 个参数，调用第二个函数
end if
```

这样就可以用一个函数名来表示不同的操作，减少了函数名的使用，增加了脚本的可读性。

(2) 超越(override)

超越意味着在后代对象中可以定义一个与祖先的函数有相同名称和参数列表的函数。在后代对象中，只有使用全局作用域(::)时调用祖先的该函数，否则调用后代中的该函数。

例如：

在一个祖先对象(w_frame)中定义一个函数 f_return()。然后在这个祖先对象(w_frame)的后代对象(w_deframe)中定义了与祖先对象相同名字的函数 f_return()。在这个后代对象(w_deframe)中添写脚本如下：

```
f_return()                      //调用后代对象(w_deframe)中的 f_return()
w_frame :: f_return()           //调用祖先对象(w_deframe)中的 f_return()
```

用户可以根据不同的需要，调用祖先函数的功能或只调用后代对象中函数的功能。这样提高了编写脚本的灵活性，增加了 PowerBuilder 的适用性。

5. 结构的创建和使用

结构实际上定义了一个新的数据类型，它由一个或多个具有相同或不同类型的相关变量组成。在 PowerBuilder 中可以自定义两种类型的结构：全局结构和对象级结构。全局结构不与任何对象相关联，并且能够在整个应用程序中使用。对象级结构必须与具体对象相关联，例如窗口对象、用户对象、菜单对象或应用对象。这些结构通常在对象内部使用。当然，也可以通过一定的方法在其他脚本中使用对象级结构。本小节将主要介绍如何创建和使用结构。

(1) 创建全局结构

全局结构能够在整个应用程序中使用，创建全局结构的步骤为：

① 从菜单栏中选择 File | New 命令，打开 New 对话框，并在 PB Object 选项卡中选择 Structure，系统将打开 New Structure 对话框，如图 3-8 所示。

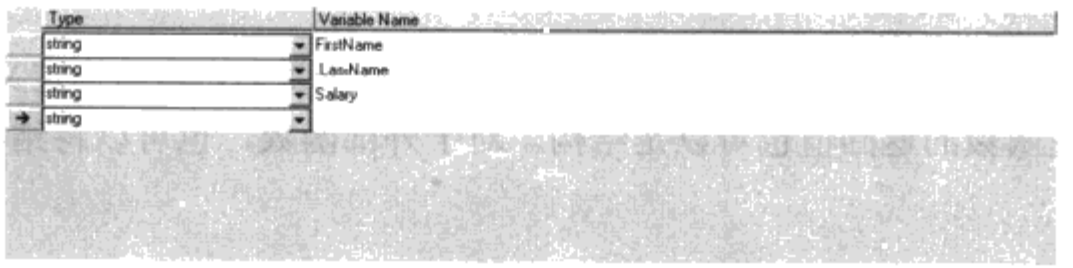


图 3-8　全局结构 New Structure 对话框

该对话框中各部分的意义为：

- Variable Name 列定义结构分量的名称。
- Type 列指定结构分量的数据类型。

根据需要定义各结构分量。如果要在当前结构分量前插入一个分量或删除当前结构分量，在结构画板中右击，从弹出菜单中选择正确选项。

② 定义好各结构分量后，保存结构。在 Save Structure 对话框中输入新结构的名称，单击 OK 按钮完成全局结构的定义。

(2) 创建对象结构

下面介绍创建窗口对象结构的步骤，其他对象结构的创建步骤可以依此类推。

创建窗口对象结构的步骤为：

① 打开要声明结构的窗口对象。从菜单中选择 Insert | Structure 命令，打开结构窗口，如图 3-9 所示。

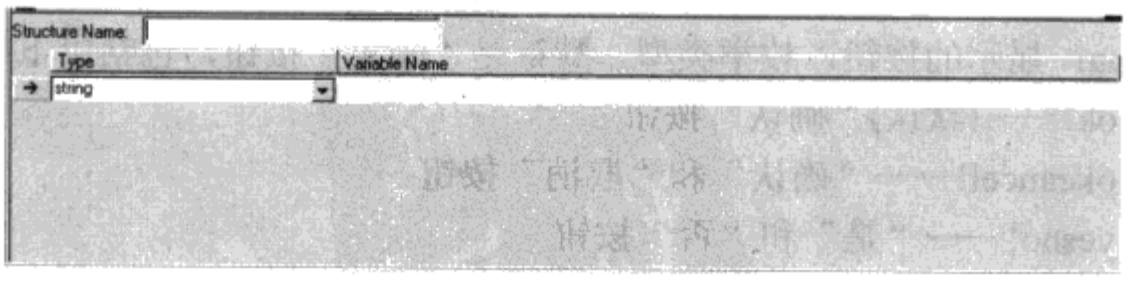


图 3-9　窗口对象结构 New Structure 对话框

② 按定义全局结构相同的方法定义窗口结构。

(3) 结构的使用

定义一个结构，也就定义了一个新的数据类型。只要该结构定义保存在应用库中，用户就可以在脚本中和用户自定义函数中使用该结构。

在脚本和用户自定义函数中使用结构的方法：

① 声明结构变量。

② 在程序中引用结构的成员变量。

引用结构中的成员变量与引用 PowerBuilder 对象的属性类似，都是通过点号(.)来实现：

```
structure.variable
```

例如有一个结构 Man，它有 3 个分量 FirstName、LastName、Salary，使用下述语句说明结构变量 lp_Man 并给分量赋值：

```
Man lp_Man                          //声明一个类型为 Man 的变量
Lp_Man.FirstName="Dale"             //给结构变量赋值
lp_Man.LastName="Carnegie"
lp_Man.Salary=3000
```

用户也可以将结构作为参数在函数中传递。在定义函数时，将结构定义为函数的参数。同样用户自定义函数的返回值也可以是结构。对于外部函数，也可以将结构作为函数的参数。

6. 常用函数

(1) MessageBox()

显示消息提示框，语法格式如下：

```
MessageBox ( title, text {, icon {, button {, default } } } )
```

参数选项说明如下：

- {}中内容表示可选参数。
- Title：标题。
- Text：文本，需要显示给用户看的消息，可以是字符串、数字、逻辑型。
- Icon：消息框中的图标，枚举类型，默认是 Information!，可以是如下枚举类型值。
  - Information!——信息文本
  - StopSign!——错误终止
  - Exclamation!——惊叹号
  - Question!——问题
  - None!——没有图标
- Button：显示的按钮，枚举类型，默认是“确定”按钮，包括以下 6 种。
  - ok!——(默认)“确认”按钮
  - okcancel!——“确认”和“取消”按钮
  - yesno!——“是”和“否”按钮
  - YesNoCancel——“是”、“否”和“取消”按钮
  - RetryCancel——“重试”、“取消”按钮

◆ AbortRetryIgnore——“放弃”、“重试”、“忽略”按钮

● Default：上述按钮是默认的。

返回对应按钮的位置，如指定按钮为 okcancel!，用户单击“确认”按钮，则返回 1，若单击“放弃”按钮，则返回 2。

例如，以下语句将弹出如图 3-10 所示的提示框。

```
MessageBox ('提示框', '欢迎使用 PB!',Exclamation!,yesno! )
```

(2) Is 族函数和转换函数

以 Is 开头的函数有很多，例如 IsNumber、IsDate、IsTime 等，它们用来判断一个字符串是否符合某种数据类型的格式，它的返回值是一个布尔类型的值，当字符串符合类型要求时，返回值为 True，否则返回值为 False。诸如此类的函数统称为类型判断函数。之所以要用到这些函数是因为当用文本框输入数据时，需要用类型转换函数将输入的字符串转化为所需要的数据类型，如果该字符串与所要类型不匹配，转化时就会出现异常错误。

图 3-10　提示框

比较好的解决方法是先用类型判断函数来检验输入的字符串，然后再进行类型转化。类型转化函数的作用是将某个字符串转化为另一种类型的数据格式。常见的类型转化函数有 Integer(将字符串转化为整形)，Long(将字符串转化为长整形)，Real(将字符串转化为实形)，Double(将字符串转化为双精度形)，Dec(将字符串转化为十进制数)。另外有 String 函数将各种不同的数据类型转化为字符串，并可以规定转化的格式。

(3) Trigger、Post 函数

可以使用 Trigger 来激活一个函数或事件。它的返回值可以在 Script 中使用。当用户使用 Post 来激活一个函数或事件时，它被加入到对象队列中并且按顺序执行。在大多数情况下，它在当前 Script 完成后被执行。但是如果在这段时间中其他系统事件发生了，那它可能在队列中被排到其他 Script 之后。它的返回值不能被 Script 使用。因为 Post 使得返回值对调用者不可见，所以可以把它想像成将函数或事件调入一种状态。

一般情况下，如果需要将所在 Script 执行完毕后再调用其他函数或事件，则应该使用 Post 而不是 Trigger。TriggerEvent 和 PostEvent 是 PowerBuilder 中的事件函数，被用来触发某一个特定事件。尽管它们的功能已经被 Post 和 Trigger 所取代并得到加强。但为了向后兼容，它们仍然被保留了下来。但是不能在调用事件时传递参数。必须将数据通过消息对象传递给事件的数据。

以下示例带参数执行事件，并且从消息对象中反馈参数值：

```
w_xxx.postevent('ue_myevent',0,'this is a test')
```

在窗口 w_xxx 的 ue_myevent 事件中接收参数：

```
string ls_passed_string
ls_passed_string = Message.stringparm
```

(4) 文件操作

文件操作在程序设计中是不可缺少的。可以用文件函数来打开一个文本或 Blob 文件并

对它进行读写操作。可以用流(Stream!)模式或行(Line!)模式打开一个文件。在行模式中，文件每次被一行行地读入，每行之间被回车符和换行符隔开，最后以 EOF(文件结束)为结束标志。在流模式中，文件被整个读入，并且换行符和回车符不被写入文件。

用户一般用流模式将整个文件读入到 MultilineEditor 中，再整个写到文件中去。

当打开文件时，PowerBuilder 分给打开的文件一个独特的整数，并且将指针放在文件中所规定的地方。可以用这个整数来标识文件以便读写或关闭文件。文件指针规定了下一次文件读写的起始位置并且在每次读写完成后由 PowerBuilder 自动更改。文件函数如下：

- FileClose：关闭一个打开的文件。
- FileDelete：删除一个文件。
- FileExists：判断文件或目录是否存在。
- FileLength：得到文件长度。
- FileOpen：打开一个文件，如果不存在则创建。
- FileWrite：写入一个文件。
- FileRead：从某打开的文件中读取。
- FileSeek：移动文件中读写位置。
- GetFileOpenName：弹出标准文件打开对话框，得到一个需要打开的文件名。
- GetFileSaveName：弹出标准文件保存对话框，得到一个需要保存的文件名。

(5) RGB 函数

语法格式如下：

```
RGB (红,绿,兰) ,RGB(R,G,B)
```

- R：0～255，红色。
- G：0～255，绿色。
- B：0～255，蓝色。

返回一个颜色值。

常用颜色对应的 RGB 值：

- 红色：rgb(255,0,0)
- 绿色：rgb(0,255,0)
- 蓝色：rgb(0,0,255)
- 黄色：rgb(255,255,0)
- 黑色：rgb(0,0,0)
- 白色：rgb(255,255,255)
- 中灰：rgb(192,192,192)
- 浅灰：rgb(128,128,128)
- 深红：rgb(128,0,0)
- 棕色：rgb(128,128,0)
- 深绿：rgb(0,128,0)

(6) 其他函数

① Open 函数

Open 函数用来显示一个窗口，并且使得该窗口的一切属性和控件都对 Script 可用。Open

函数在打开窗口的同时，也调用了该窗口的 Open 事件。

② Close 函数

Close 函数关闭窗口并释放被窗口和窗口中所有控件占据的系统资源。在执行 Close 函数后，系统首先触发窗口的 Close-Query 事件，若事件返回值为 1，则触发 Close 事件并关闭窗口。

③ Create 语句

Create 语句为某一对象类型创建一个实例。一个对象被创建后，对象的属性可以被访问。Create 语句创建完毕后，创建生成的对象实例可以被当作一个变量来使用，例如：

```
gu_registry = create u_sys_registry
```

④ Destroy 语句

与 Create 相对应，从内存中删除该对象，如：

```
Destroy gu_registry
```

3.3.3 编程规范

1. 基本要求

程序结构清晰，简单易懂，单个函数的程序行数不得超过 100 行。
打算干什么，要简单，直截了当，代码精简，避免垃圾程序。
尽量使用标准库函数和公共函数。
不要随意定义全局变量，尽量使用局部变量。
使用括号以避免二义性。

2. 可读性要求

可读性第一，效率第二。
保持注释与代码完全一致。
主要变量(结构、类或对象)定义或引用时，注释能反映其含义。
常量定义(DEFINE)有相应说明。
处理过程的每个阶段都有相关注释说明。
在典型算法前都有注释。
循环、分支层次不要超过五层。
注释可以与语句在同一行，也可以在上行。
空行和空白字符也是一种特殊注释。
一目了然的语句不加注释。
注释的作用范围可以为：定义、引用、条件分支以及一段代码。
注释行数(不包括程序头和函数头说明部分)应占总行数的 1/5 到 1/3 。

3. 结构化要求

禁止出现两条等价的支路。

禁止 GOTO 语句。
用 IF 语句来强调只执行两组语句中的一组。
用 CASE 实现多路分支。
避免从循环引出多个出口。
函数只有一个出口。
不使用条件赋值语句。
避免不必要的分支。
不要轻易用条件分支去替换逻辑表达式。

4. 正确性与容错性要求

程序首先是正确，其次是优美。
无法证明你的程序没有错误，因此在编写完一段程序后，应先回头检查。
改正一个错误时可能产生新的错误，因此在修改前首先考虑对其他程序的影响。
所有变量在调用前必须被初始化。
对所有的用户输入必须进行合法性检查。
不要比较浮点数的相等，如：10.0 * 0.1 == 1.0，不可靠。
程序与环境或状态发生关系时，必须主动去处理发生的意外事件，如文件能否逻辑锁定、打印机是否联机等。

5. 可重用性要求

重复使用的完成相对独立功能的算法或代码应抽象为类。
类应考虑面向对象思想，减少外界联系，考虑独立性或封装性。

3.4 实例：文件复制

本例的任务是应用动态链接函数库(DLL)的外部接口，通过调用 Windows API 函数，在 PB 中实现文件的复制功能。本例将主要使用到 LZOpenFile()、CopyLZFile()和 LZClose()等 Windows API 函数。

Windows 操作系统在像文件操纵一类的低级函数调用方面是相当灵活的。而在 PB 中若要实现这一些低级操作，开发者们必须用第三方开发库(如：FUNCky for PowerBuilder 库)或用 C 语言之类的语言开发自己的库。为了避免在程序发布时不需要将额外的库打包进 PB 的应用程序中，下面介绍如何通过调用 Windows API 函数，来实现简单而又常用的低级操作：文件复制。

具体设计步骤如下：

(1) 建立窗体 w_5_3，设置窗体属性，并编写 Open 事件的脚本。
Title=“文件复制”。

(2) 设置控件。

- 静态文本(StaticText)：
st_1：st_1.text=“源文件：”。

st_2：st_2.text=“目标文件：”。

- 分组框架(GroupBox)：

  gb_1：gb_1.text =“请填写以下信息”。

- 命令按钮(CommandButton)：

  cb_1：cb_1.text=“复制”

  cb_2：cb_2.text=“退出”

(3) 完成上述 3 步后，窗体 w_5_3 的效果如图 3-11 所示。

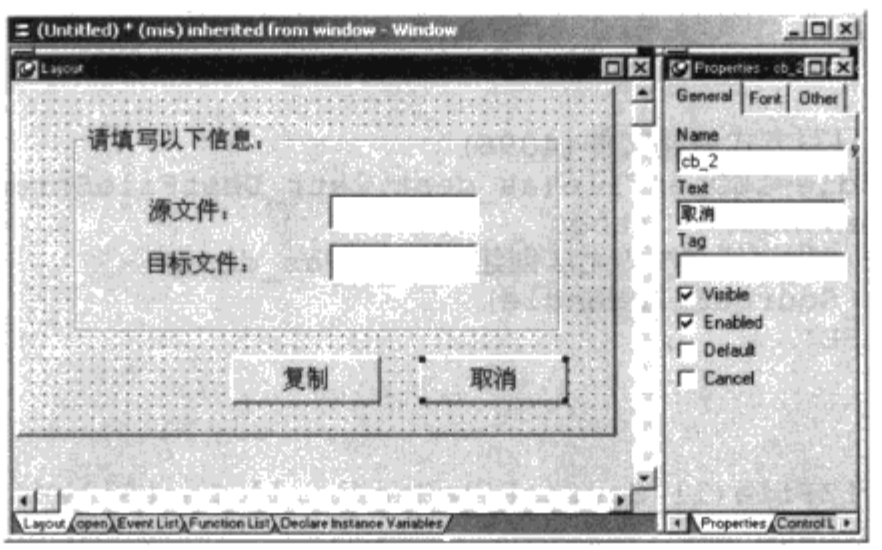


图 3-11　建立窗体 w_5_3

(4) 在 Open 事件(窗体 w_5_2 或 Application)中声明如下函数：

```
function Long LZOpenFile(string FileName,ref ws_ofstruct FileStructure,uint
style)Library "lzexpand.dll"
function Long CopyLZFile(uint SourceHandle,uint DestHandle)Library
"lzexpand.dll"
function Long LZClose(uint FileHandle)Library "lzexpand.dll"
```

(5) 定义与 C 语言对等的 ws_ofstruct 窗口级结构，如下：

```
byte[1]        character
fixed_disk[1]   character
errorcode      unsignedinteger
reserverd[4]    character
pathname[128]  character
```

PB 中的结构(Structure)设计面板如图 3-12 所示，在相应的空白栏里填入上述各项内容。

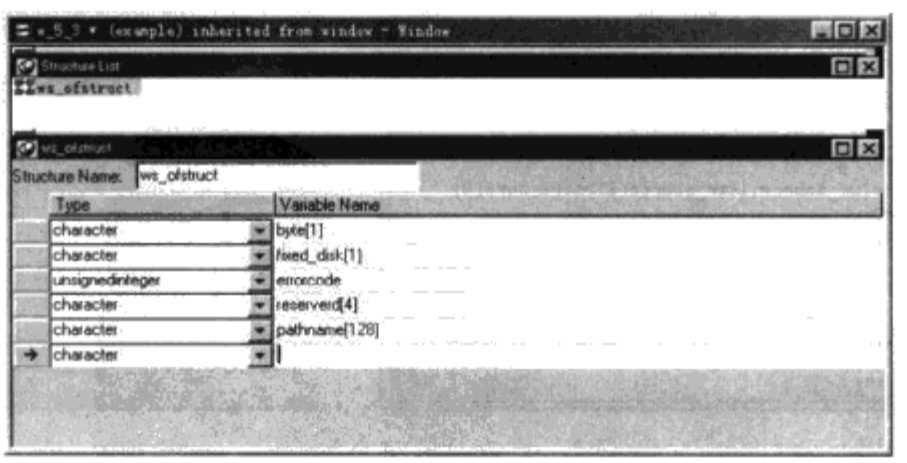


图 3-12　结构(Structure)设计面板

(6) 编写实现文件复制的函数 wf_copy(string as_source,string as_dest)returns boolean，代码如下：

```
//定义变量
long ll_SourceFileHandle,ll_DestFileHandle,ll_result
ws_ofstruct_SourceFileStructure,lstr_DestFileStructure

//打开源文件，只读方式打开文件(0)
ll_SourceFileHandle=LZOpenFile(as_source,lstr_SourceFileStructure,0)
if  ll_SourceFileHandle=-1 then
    messagebox("程序错误","无法打开文件："+as_source)
    return FALSE
end if

//打开目标文件，只写方式创建文件(4096)
ll_DestFileHandle=LZOpenFile(as_dest,lstr_DestFileStructure,4096)
if  ll_DestFileHandle=-1 then
    messagebox("程序错误","无法创建文件："+as_dest)
    LZClose(ll_SourceFileHandle)
    return FALSE
end if

//复制文件
ll_result=CopyLZFile(ll_SourceFileHandle,ll_DestFileHandle)
if ll_result<0 then
    messagebox("程序错误","无法复制文件："+as_source+"到文件"+as_dest)
    LZClose(ll_SourceFileHandle)
    LZClose(ll_DestFileHandle)
    return FALSE
end if
LZClose(ll_SourceFileHandle)
LZClose(ll_DestFileHandle)
return TRUE
```

函数(Function)的设计画板具体如图 3-13 所示，需要在相应的空白栏里填入上述内容。

(7) 运行调试。至此完成本例的所有工作。完成了上述工作就得到了一个通过调用 Windows API 函数，在 PB 中实现文件的复制功能的完整窗体和程序。运行效果如图 3-14 所示。

图 3-13 函数(Function)设计画板

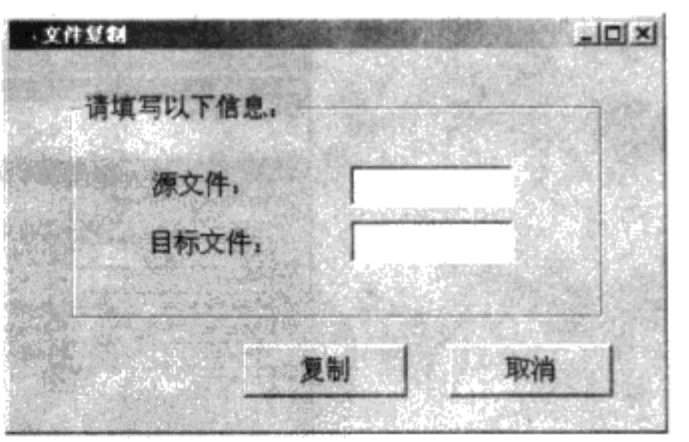


图 3-14 文件复制的实例效果

用户需要填入源文件和目标文件的信息，然后，单击“复制”按钮之后，系统将自动

完成文件复制的工作。

3.5　习　　题

1．填空题

(1) 面向对象的开发方法的主要特点是：_____、_____、_____。

(2) 标识符是程序中用来代表_____、_____、_____、_____、_____、_____、_____等名称的符号。

(3) PowerScript 的数据类型十分丰富，包括_____、_____、_____等。

(4) PowerBuilder 的变量可以分为_____、_____、_____、_____。

(5) PowerBuilder 的条件语句共有两类：_____和_____。

2．问答题

(1) Powerscript 如何定义数组？

(2) PowerScript 支持的运算符有哪 4 大类？

(3) 如何创建自定义全局函数？请描述其创建步骤。

第 4 章　PowerBuilder 应用基础

应用(Application)通常被看作一个对象，在 PowerBuilder 中进行的所有开发都是围绕应用对象的，应用对象是整个应用的入口点，而 PowerBuilder 通过事务来连接应用程序与数据库。本章主要介绍应用对象和事务对象的相关概念和设计方法。

4.1　应用对象基础

4.1.1　应用对象概述

在 PowerBuilder 中，应用是一组窗口(Window)的集合。窗口是用户与计算机交互的主要工具，它总是与一系列的活动相关。

应用对象(Application Object)是执行这些动作集合的入口点，正如在 C 语言中的主函数(main)是程序的入口点一样。作为对象，应用对象被存储在 PowerBuilder 库中，实际上，其他对象(如窗口、函数、数据窗口和菜单等)也存放在 PowerBuilder 库中。

PowerBuilder 10.0 提供了一个面向对象的开发环境，用它开发的应用程序是以对象和事件驱动为基本特征的。应用对象作为 PowerBuilder 10.0 的对象之一，也有一组事件。

当用户运行应用程序时，首先触发应用对象的 Open 事件，在这个事件的处理程序中可以设置应用程序的初始状态、装入其他对象、连接数据库等。当用户终止应用程序时，应用对象的 Close 事件被触发，在这个事件的处理程序中完成必要的系统清理工作，例如断开与数据库的连接、清除用户自定义对象等。当程序运行过程中发生严重错误时，应用对象的 SystemError 事件被触发，这时可以报告错误类型和性质，以便及时排除错误。

4.1.2　应用对象和应用对象画板

应用对象(Application Object)是 PowerBuilder 10.0 的众多对象之一，也是每个 PowerBuilder 应用程序都必须具备的对象。应用对象可在应用对象画板中创建和修改。下面首先介绍应用对象的用途，其次讲述应用库的概念及要求，最后讲解应用对象画板的组成以及创建应用对象的方法。

1.　应用对象

应用对象是启动 PowerBuilder 应用程序时装入的第一个对象，除了标识应用程序和作为应用程序的入口点外，应用对象还保存和定义了下述信息。

- 默认字体：应用对象是个非可视对象，它的 4 个字体属性用于定义应用程序中其他对象使用的默认字体。把应用对象的字体属性设置成应用程序中使用最频繁的字体，将有助于加快应用程序的开发进程(避免了为每个对象设置所需字体)。
- 应用程序图标：指定用哪个图标代表运行中的应用程序。
- 应用库搜索路径：该路径指明了在查找应用程序中包含的各种对象时都检索哪些

应用库。放置在前面的应用库将首先被查找。

- 默认的全局变量：PowerBuilder 为每个应用程序提供 5 个默认的全局变量，分别为 SQLCA、SQLDA、SQLSA、Error 和 Message。
- 应用对象结构、函数和实例变量。
- 其他全局变量。
- 全局外部函数：当要使用 PowerBuilder 本身没有提供的功能时，可以通过访问外部函数(比如 Windows 系统中的动态链接库)来解决。

2. 应用库

用 PowerBuilder 开发应用程序时，建立的所有对象(包括对象函数、事件处理程序)、函数、结构和定义的所有变量，最终都存储在磁盘文件中，该文件以.pbl 为后缀，称为应用库。一个应用程序既可以存放在一个应用库中，也可以分别存储在多个库中。

注意： 不要使单个应用库过大，通常不宜超过 800KB，否则将增加系统的查库时间。库文件不能直接使用文本编辑器查看，但通过库管理画板(LibraryPainter)可以清楚地浏览、操纵应用库。

3. 应用对象画板

PowerBuilder 10.0 应用程序的开发过程实际上是在各个画板工作窗口中完成的。画板的工作窗口是对象的设计窗口，也可以看成是对象设计器。

在应用对象画板工作窗口中可以完成如下工作。

- 创建或选择应用对象。
- 设置应用对象的属性。
- 编写应用对象事件处理程序。
- 保存应用对象。
- 显示应用对象的组成成分。

1)　创建新的应用对象

建立 PowerBuilder 应用程序的第一项任务就是建立新的应用对象，其操作步骤为：

(1)　单击画板工具栏的应用画板图标，进入应用画板工作区。

(2)　选择 File 菜单中的 New 命令，打开 New 对话框，如图 4-1 所示。

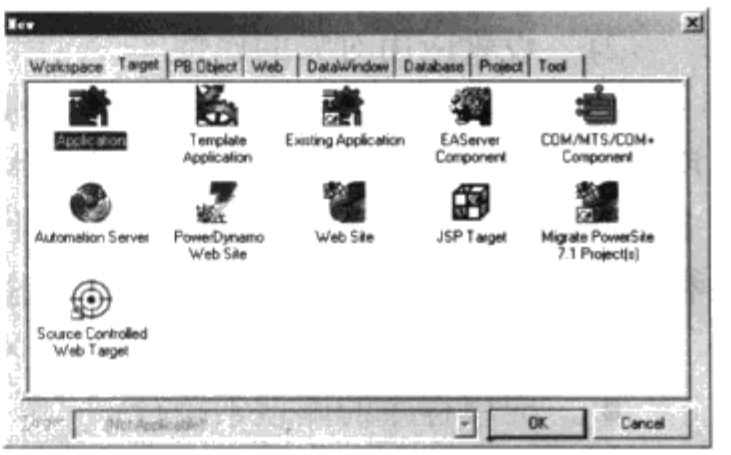


图 4-1　New 对话框

(3)　在 New 对话框中选择 Target 选项卡，选中 Application 项，单击 OK 按钮，弹出 Specify New Application and Library 对话框，如图 4-2 所示。在此对话框中输入创建应用的名称，单击 Library 和 Target 后面的按钮，即可打开新应用对象库和新目标的对话框，分别如图 4-3 和图 4-4 所示。

(4)　在 Application Name 编辑框中输入一个字符串作为应用对象的名称，字符串的长度范围为 1~40 个字符。以后打开应用程序时，对象名称将出现在 PowerBuilder 的标题栏中。

(5)　选择相应的文件后，在图 4-2 中单击 Finish 按钮，完成应用对象的创建，结果如

图 4-5 所示。

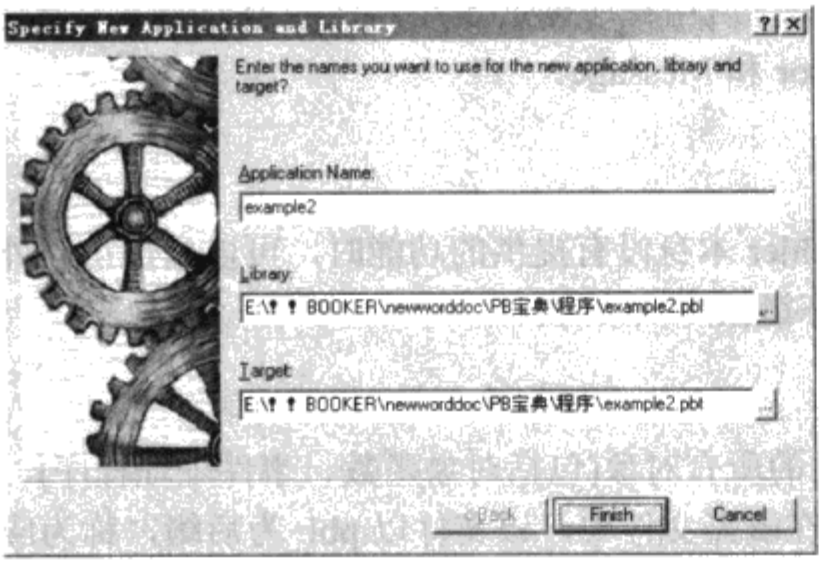


图 4-2 Specify New Application and Library 对话框

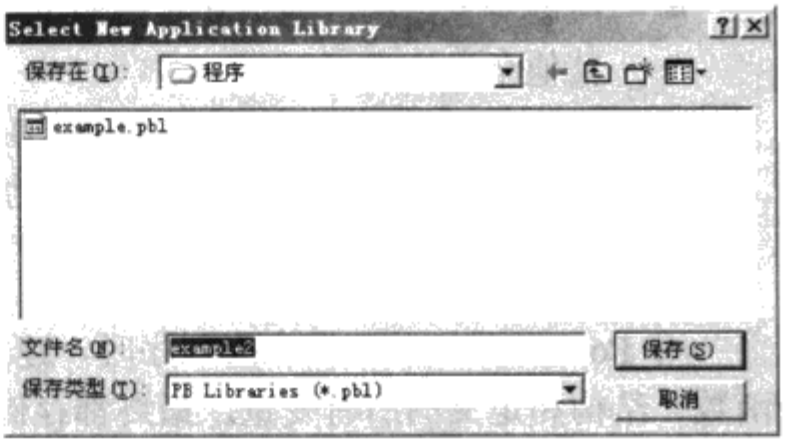


图 4-3 Select New Application Library 对话框

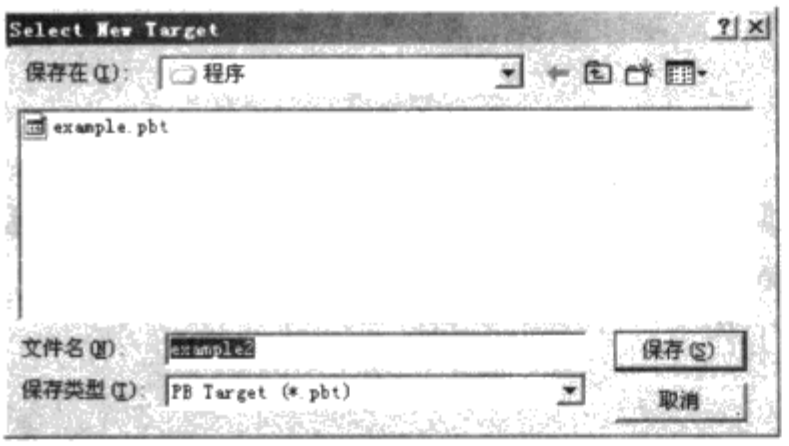


图 4-4 Select New Target 对话框

图 4-5 完成结果

2) 设置应用对象的属性

应用对象的属性对整个应用程序的开发起着举足轻重的作用。它的字体属性决定了应用程序开发过程中其他对象的文本字体，因此在设计应用程序的开始就选择一种适合绝大多数对象的字体可以减轻开发压力，让用户能够在最短的时间内开发出优质的应用系统。利用应用对象可以为应用程序设置下述属性：

- 默认的字体属性(包括字体、风格、大小和颜色)；
- 应用程序的库搜索路径；
- 应用程序的图标；
- 默认的全局对象的类型。

下面简述这些属性的用途与设置方法。

(1) 设置默认字体

应用对象的 4 个字体属性决定了在开发应用程序过程中经常使用的 4 种字体，这 4 个字体属性分别为 Text Font、Column Font、Header Font 和 Label Font。Text Font 决定窗口中的控件、用户对象和数据窗口中静态文本的默认字体，Column Font 指定数据窗口中检索出的数据使用的默认字体，Header Font 定义列表风格(tabular)和网格风格(grid)数据窗口对象中列标题的默认字体，Label Font 确定自由风格(freeform)数据窗口对象中列选项卡的默认

字体。

在开发应用程序时，根据需要先设定这些字体，以后在设计其他对象时就可以避免重复设置字体了。设置字体包括以下 4 方面的内容：

- 字体类型(如宋体、楷体、行楷等)；
- 字体大小(如 10、11、12)；
- 字体风格(如常规、斜体、黑体、黑斜体)；
- 字体颜色(包括文字本身的颜色和背景颜色)。

下面以设置 Toolbar 属性为例，说明默认字体属性的设置方法，其他字体属性设置过程与此相似。步骤如下：

① 单击应用画板图标，进入应用画板工作区(如果已经在该工作区中，可以省略该步)。

② 单击应用画板工具栏上的 Properties(属性)图标，打开属性对话框，如图 4-6 所示。界面显示的是 General 选项卡，该选项卡列出了应用对象的名称等信息。

③ 单击 Toolbar 标签，显示如图 4-7 所示的选项卡，该选项卡列出了工具栏标题等信息。

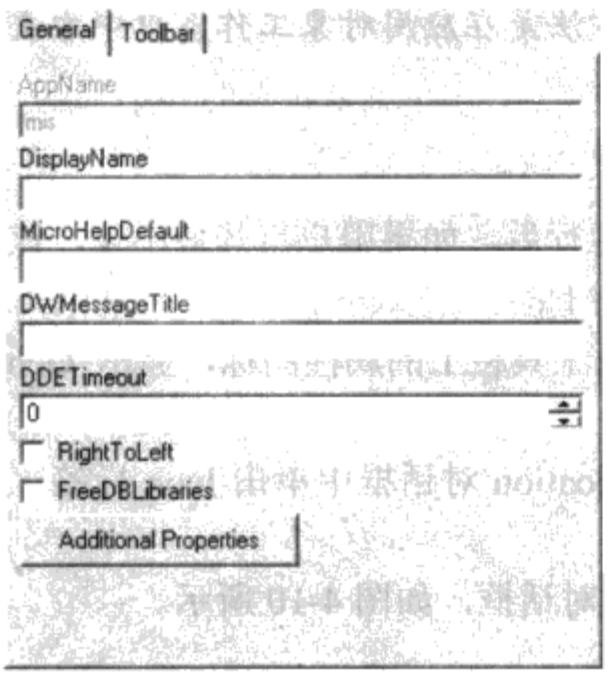


图 4-6　General 选项卡

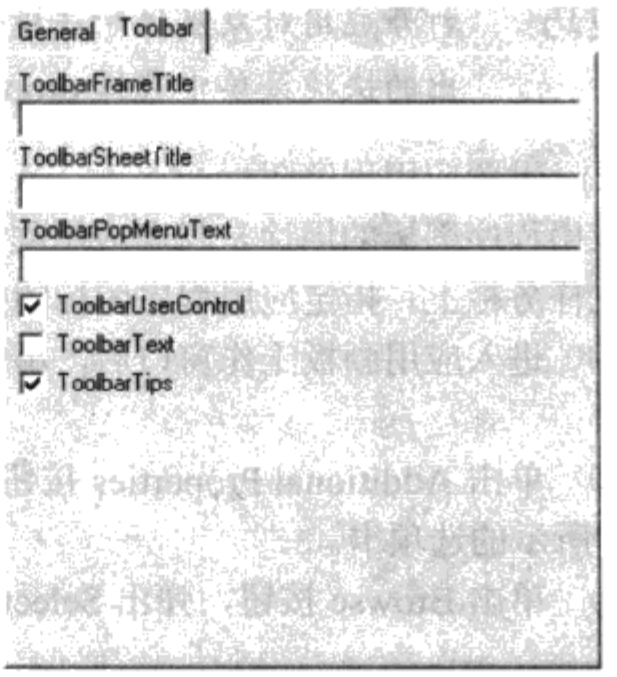


图 4-7　Toolbar 选项卡

选择 General 选项卡，然后单击 Additional Properties 按钮，弹出 Application 对话框，如图 4-8 所示。选中 Text Font 选项卡，设置字体并根据需要进行修改。

- 要改变字体，可以在 Font 列表框中找到所需的字体，单击选择该字体，此字体名称便出现在 Font 下的编辑框中。
- 要改变字体大小，可以从 Size 列表框中选择所需字体大小或直接在列表框中输入数值。
- 要改变字体风格，可以从 Font Styles 列表中选择所需的类型，其中 Regular 表示常规、Italic 表示斜体、Bold 表示黑体、Bold Italic 表示黑斜体。

- 要改变字体效果，可以从 Effects 选项组中选中 Strikeout 或 Underline 复选框。

注意： Strikeout 指删除线，Underline 指下划线，这两个复选框并非对所有字体都有效。

- 要改变文本的前景颜色(字符颜色)，可以在 Text Color 下拉列表框中选择所需颜色。
- 要改变文本的背景颜色，可以在 Background 下拉列表框中选择所需颜色。

注意： 对 Column Font、Header Font 和 Label Font 来说，此处不能修改前景和背景颜色，该工作可在数据窗口画板中完成。在字体属性设置过程中，Sample 组合框随时显示最新设置的效果。

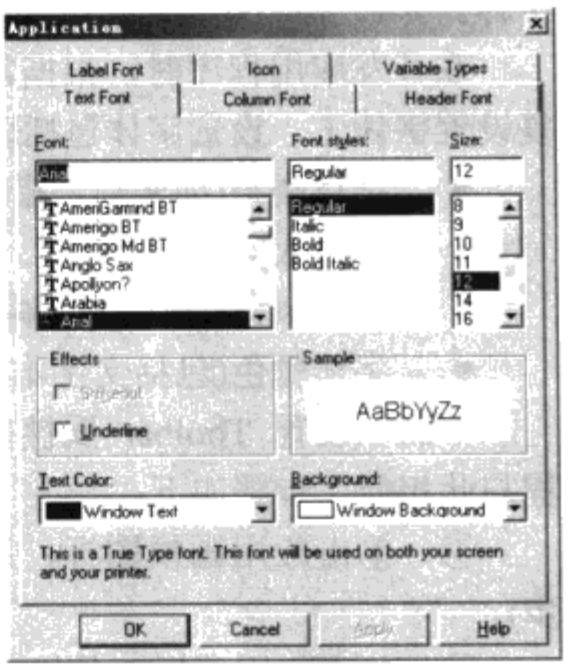


图 4-8 Application 对话框

技巧： 打开应用对象属性对话框的另一种方法是在应用对象工作窗口中右击，在弹出的快捷菜单中选择 properties 命令。

(2) 设置应用程序图标

应用程序图标的用途是代表应用程序。程序运行后，如果用户将其最小化，该图标就出现在任务栏上。指定应用程序图标的操作步骤如下：

① 进入应用画板工作窗口后，单击应用画板工具栏上的属性图标，显示 General 选项卡。

② 单击 Additional Properties 按钮，在 Application 对话框中单击 Icon 标签，显示如图 4-9 所示的选项卡。

③ 单击 Browse 按钮，弹出 Select Drag Icon 对话框，如图 4-10 所示。

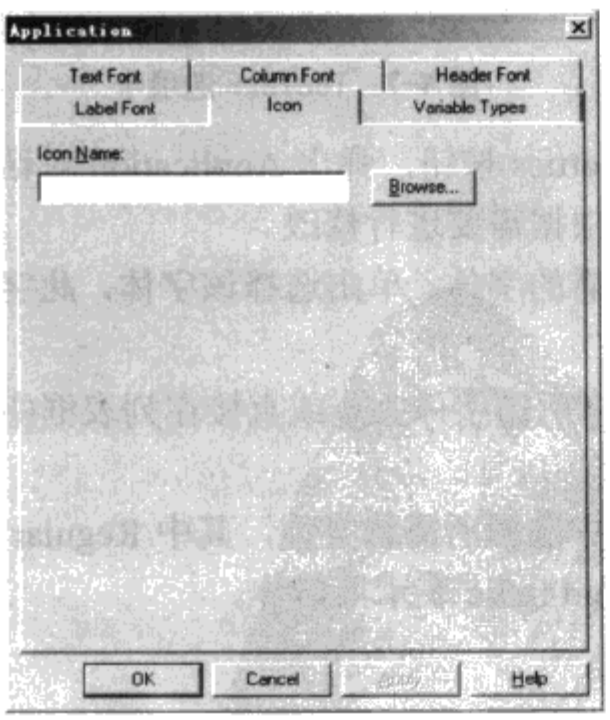


图 4-9 Icon 选项卡

图 4-10 Select Drag Icon 对话框

④　选择一个合适的图标文件(后缀名为.ico)，单击“打开”按钮后，其文件名出现在 Icon Name 编辑框中，图标图案显示在 Browse 按钮下面，该图标就成为应用程序的图标。

⑤　单击 OK 按钮，完成应用程序图标设置。

技巧:　利用与 PowerBuilder 一起发行的 Image Editor(Watcom 图像编辑器)可以自行创建图标。

(3)　指定默认全局变量的类型

PowerBuilder 10.0 提供了 5 个内置的全局变量(SQLCA、SQLDA、SQLSA、Error 和 Message)，它们都有默认的类型，但可对其进行修改，步骤如下：

①　使用用户对象画板(User Object Painter)创建标准类用户对象，该类必须继承相应的标准类。

②　进入应用画板工作窗口后，单击应用画板工具栏上的属性图标，显示 General 选项卡。

③　单击 Additional Properties 按钮，在 Application 对话框中单击 Variable types 标签，显示如图 4-11 所示的选项卡，在相应的编辑框中输入新的类型。

④　单击 OK 按钮完成类型设置。

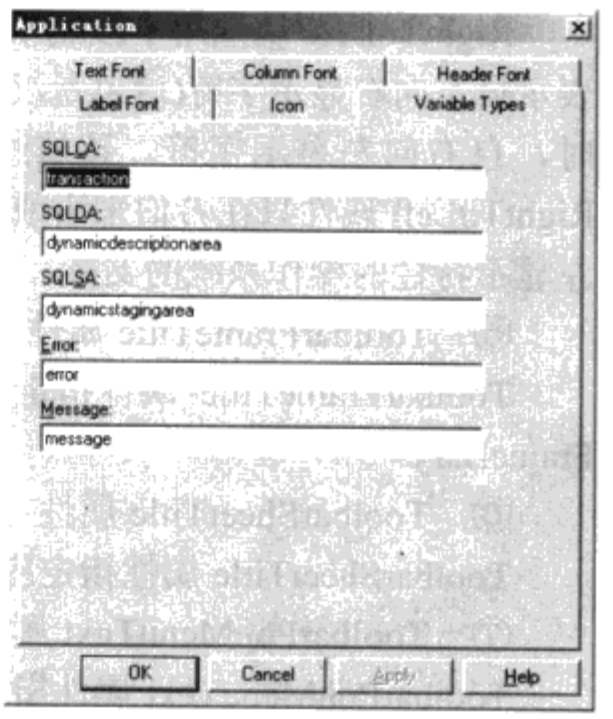


图 4-11　Variable Types 选项卡

修改某个全局变量类型的目的是为了方便应用程序的开发，比如可以定制 Error 对象，使其能够直接提供中文版的出错信息。

(4)　应用对象的其他属性

除了在应用对象的属性对话框中设置的属性外，在代码中还可以使用其他应用对象属性，下面分别予以介绍。

①AppName 和 DisplayName 属性

应用对象的 AppName 属性指明了应用对象的名称，该属性在运行时不能被修改，它是个只读属性。DisplayName 属性指定了应用对象的显示名，它能够方便用户的阅读。例如，在使用 OLE 时，各种对话框中显示的应用名称可能就是 DisplayName 属性的值(是否显示应用名称由 OLE 服务器决定)。如果未给 DisplayName 属性赋值，则其默认值使用 AppName 属性的值。

下面的语句假定放置在应用对象的 Open 事件处理程序中，其功能是给当前应用对象的 DisplayName 赋值：

```
This.DisplayName=PB"演示"
```

②　DDETimeOut 属性

当 PowerBuilder 应用程序作为 DDE 客户端时，DDETimeOut 属性指定了该应用程序等待服务器响应的时间，超过这段时间后，客户程序将放弃数据交换。DDETimeOut 属性指定的时间以秒为单位，默认值为 10 秒。

③　MicroHelpDefault 属性

当应用程序使用多文档界面(MDI)时，使用 MicroHelpDefault 属性可以设置帮助栏上显

示的默认文本，例如，在应用对象的 Open 事件处理程序中写上以下语句：

```
This.MicroHelpDefault="准备好"
```

程序运行后帮助栏默认地显示“准备好”3 个字，MicroHelpDefault 属性的默认值为 Ready。

④ RightToLeft 属性

RightToLeft 属性用于控制 MessageBox()对话框中字符显示的方向，默认值为 False，表示自左向右显示字符(这也是汉语和英语的书写方式)。当把 RightToLeft 属性设置为 True 时，自右向左显示字符。该属性主要用于开发多国语言的应用程序，需要注意的是，RightToLeft 属性只在希伯来语或阿拉伯语版本的 PowerBuilder 中有效，且需要具备自右向左显示特性的操作系统的支持。

⑤ ToolbarFrameTitle 属性

ToolbarFrameTitle 属性指定应用程序中 MDI 框架窗口浮动工具条的标题，默认值为 FrameBar。

⑥ ToolbarSheetTitle 属性

ToolbarSheetTitle 属性指定应用程序中工作表浮动工具条的标题，默认值为 SheetBar。

⑦ ToolbarPopMenuText 属性

ToolbarPopMenuText 属性指定工具栏弹出菜单中的菜单选项，默认时，显示的界面如图 4-12 所示。

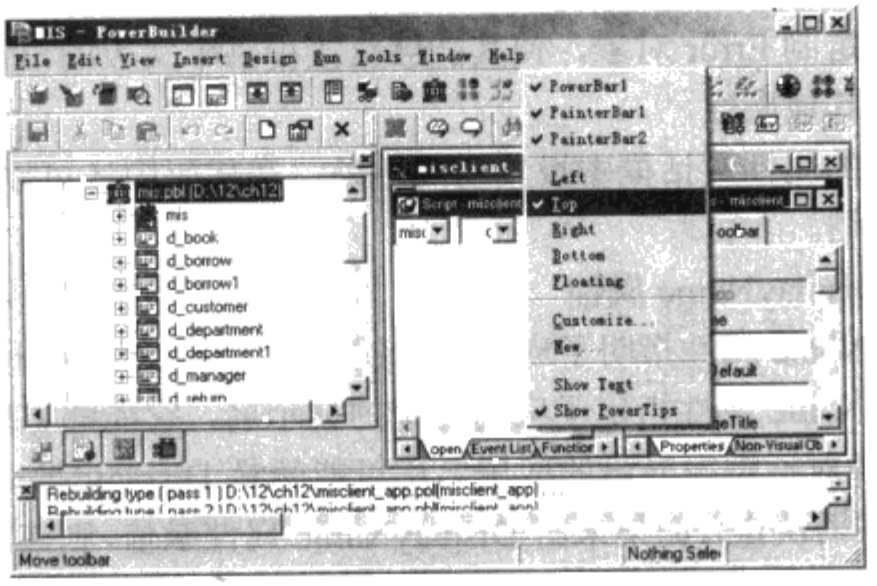


图 4-12 工具栏弹出菜单

⑧ ToolbarText 和 ToolbarTips 属性

这两个属性的类型都是 Boolean，ToolbarText 属性决定是否在工具栏图标上显示文字提示，其值为 True 时显示，为 False 时不显示。ToolbarTips 属性指定当鼠标指针移动到工具栏图标上时是否显示跟踪提示，其值为 True 时显示，为 False 时不显示。

注意： 只有在 ToolbarText 属性设置为 False 时，ToolbarTips 属性的设置才有效。

⑨ ToolbarUserControl 属性

若要限制用户控制工具栏(比如移动工具栏)，可通过将 ToolbarUserControl 属性设置为 False 来实现。此时，关闭工具栏的弹出菜单功能，用户也不能通过拖曳操作移动工具栏的

位置。

4.1.3　应用对象的事件

应用对象的事件一共有 6 个，其中 Open 事件必须包含事件处理程序，否则整个应用程序在打开后，将不执行任何操作就结束。通常情况下，在 Open 事件处理程序中需要设置应用系统的初始环境、建立与数据库的连接及打开应用的第一个窗口等。

下面分别介绍应用对象的 Open、Close、Idle、SystemError、ConnectionBegin 和 ConnectionEnd 事件的发生时机与常见用法。

1.　Open(打开)事件

应用对象的 Open 事件在启动应用程序时发生，且只发生一次。对所有应用程序来说，都必须对该事件编程，否则运行应用程序时系统将出现一个警告对话框，提示必须对应用对象的 Open 事件编程。

一般来说，在 Open 事件处理程序中应该完成下述工作：

- 完成初始化工作。
- 利用 INI 文件设置某些属性，或初始化一个事务对象。
- 查询环境，保证运行应用程序所需的所有资源都可用。
- 连接要访问的数据库。
- 打开应用程序的第一个窗口。

为了方便编写连接数据库的代码，PowerBuilder 在安装目录下提供了一个名称为 SQLCA.SCR 的文件，可以将该文件的内容插入到应用程序中，以完成对事务对象的通用赋值。SQLCA.SCR 文件的内容为：

```
//该程序从 PB.INI 中读取所有数据库值并将其存储在 SQLCA 中
SQLCA.DBMS=ProfileString(PB.INI,Database,DBMS, "")
SQLCA.Database=ProfileString(PB.INI,Database,DataBase, "")
SQLCA.LogID=ProfileString(PB.INI,Database,LogID,"" )
SQLCA.LogPass=ProfileString(PB.INI,Database,LogPassword,"")
SQLCA.ServerName=ProfileString(PB.INI,Database,ServerName, "")
SQLCA.UserID=ProfileString(PB.INI,Database,UserID, "")
SQLCA.DBPass=ProfileString(PB.INI,Database,DatabasePassword,"" )
SQLCA.Lock =ProfileString(PB.INI,Database,Lock, "")
SQLCA.DbParm=ProfileString(PB.INI,Database,DbParm,"" )
```

2.　Close(关闭)事件

应用对象的 Close 事件只发生一次，它发生在退出应用程序和返回到操作系统之前，通常在这个事件的处理程序中完成下述工作：

- 删除应用程序运行过程中创建的对象(例如额外的事务对象等)。
- 在 INI 文件中保存状态信息，以便下次运行程序时恢复状态。
- 关闭应用程序中打开的文件。
- 断开与数据库的连接。

3.　Idle(空闲)事件

在应用程序的某段程序中使用 Idle()函数设置一个定时器时间间隔，在间隔规定的时间

内用户没有操作应用程序，即没有敲击键盘或使用鼠标，就会产生应用对象的 Idle 事件。Idle 事件的用途有下述几种：

- 创建独具特色的屏幕保护程序。
- 隐藏机密数据。
- 当用户长时间未操作应用程序时，应用程序可以自动退出并返回系统，有助于减少数据库的连接数目并节省数据库资源。

4. SystemError(系统出错)事件

SystemError 事件是处理运行错误的最后一道关卡，它用来处理最严重的错误，可以在此处编写错误处理和可能的恢复代码，简单的措施是显示出错信息。如果未对此事件编程，那么系统出错时 PowerBuilder 会通过对话框显示错误代码和错误信息。

5. ConnectBegin(连接开始)事件

ConnectBegin 事件用于分布式 PowerBuilder 应用的服务器端。当客户程序试图连接到服务器程序时，将触发服务器程序应用对象的 ConnectBegin 事件。

6. ConnectEnd(连接结束)事件

ConnectEnd 事件用于分布式 PowerBuilder 应用的服务器端。当客户程序试图断开与服务器程序的连接时，将触发服务器程序应用对象的 ConnectEnd 事件以便开发者重新修改程序，否则将返回到应用对象画板工作窗口。

7. 编写应用对象的事件处理程序

编写应用对象事件处理程序的步骤为：

(1) 进入应用对象画板工作窗口后，单击工具栏上的 Script 按钮，打开代码编辑器。

(2) 单击窗口上方 Select Event(选择事件)下拉列表框，列出所有可以编写事件处理程序的事件，如图 4-13 所示。

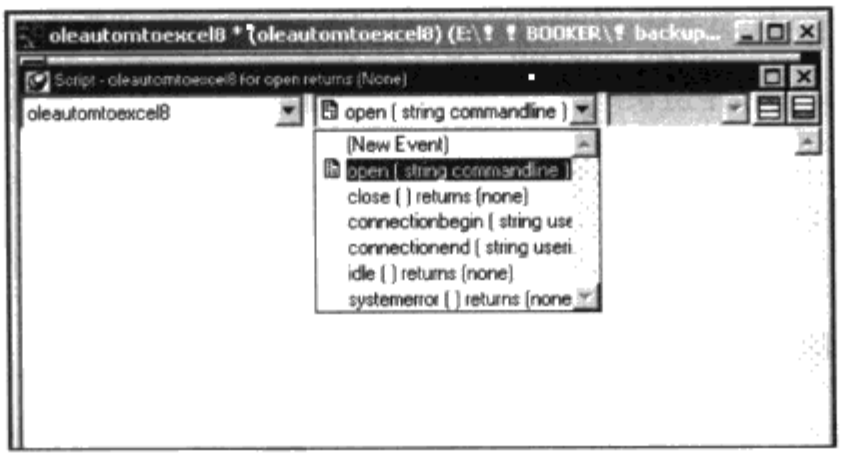


图 4-13 选择要编写事件处理程序的事件

(3) 单击要编程的事件，这时对象名称、事件名称及事件的返回值类型都会出现在编辑器窗口的标题栏上。

(4) 使用 PowerScript 语言编写事件处理程序。

(5) 编写完事件处理程序后，单击编辑器工具栏上的 Return 按钮，系统编译程序，当程序出现错误时，系统自动显示错误类型并提示需要重新修改程序，否则返回到应用对象画板工作窗口。

4.2　使用代码编辑器

PowerBuilder 10.0 具有功能强大、易学易用的代码编辑器，编辑器中的不同语法成分使用不同的颜色区分。代码编辑器除了具备复制、剪切、粘贴文本等常用功能外，还提供了粘贴变量、函数、SQL 语句等方法，提高了代码的输入速度。下面介绍代码编辑器的结构和用法。

4.2.1　代码编辑器

在 PowerBuilder 中可以在 5 个地方编写程序：应用对象画板、菜单画板、窗口画板、用户对象画板和函数画板，这些画板均需打开代码编辑器来编写代码。前 4 种对象画板的工具栏上都有编程(Script)图标，其图案为或，前一个图标表示尚未对该对象(窗口、菜单等)编写任何代码，后一个图标表示已经对该对象的某个或某些事件编写了代码。对这 4 种画板来说，只要单击画板工具栏上的编程图标，就可以进入代码编辑器中。代码编辑器工具栏的各图标如图 4-14 所示。

图 4-14　代码编辑器工具栏

选择 Paste Special 子菜单(如图 4-15 所示)后，可以执行以下操作：

- 粘贴窗口对象(Window)。
- 粘贴全局变量(Global)。
- 粘贴实例变量(Instance)。列表框分别将窗口对象的名称、全局变量名和实例变量名粘贴到编辑器的插入点位置。
- 在编辑区输入事件处理程序。

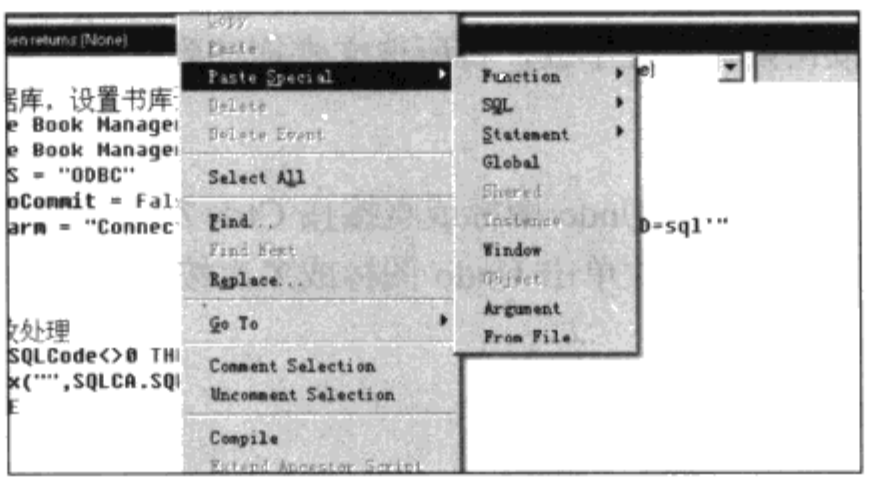


图 4-15　使用 Paste Special 选项

4.2.2　编辑代码

代码编辑器是一个功能强大的文本编辑器，它简单易用。下面简单介绍代码编辑器的常用功能。

1. 选择文本

在代码编辑器中，文本的选择方法与大多数 Windows 下的字处理程序的方法相同。具体方法如下：

(1) 在要选择文本的开始处按住鼠标左键不放，拖动鼠标指针到要选择文本的终点，释放鼠标左键，起点到终点之间的文本被选择，被选文本将反相显示。

(2) 选择一个单词的方法很简单，双击它即可。

(3) 单击代码编辑器工具栏上的 Select All 图标或直接按 Ctrl+A 组合键都可以选择代码编辑器中的所有文本。

2. 删除、剪切、复制、粘贴文本

删除一段文本的步骤如下：

(1) 选中该文本块。

(2) 单击代码编辑器工具栏上的 Clear 图标或直接按 Delete 键。

把一块文本剪切到剪贴板上的步骤如下：

(1) 选中该文本块。

(2) 单击代码编辑器工具栏上的 Cut 图标或直接按 Ctrl+X 组合键。

把一块文本复制到剪贴板上的步骤如下：

(1) 选中该文本块。

(2) 单击代码编辑器工具栏上的 Copy 图标或直接按 Ctrl+C 组合键。

把剪贴板的内容粘贴到代码中的步骤如下：

(1) 通过单击把插入点移动到要粘贴文本的地方。

(2) 单击代码编辑器工具栏上的 Paste 图标或直接按 Ctrl+V 组合键。

3. 移动文本

剪切、粘贴操作可以移动文本，但下面移动文本的方法更简单。步骤如下：

(1) 选择要移动的文本。

(2) 在选中区域按住鼠标左键不放，将所选文本拖动到所需位置后放开鼠标左键。

4. 撤消操作

单击代码编辑器工具栏上的 Undo 图标或直接按 Ctrl+Z 组合键可取消上次修改，并恢复到修改前的状态。实际上，多次单击 Undo 图标或多次按 Ctrl+Z 组合键时，可以撤消前几次的修改。

5. 查找和替换文本

查找文本的步骤如下：

(1) 从 Search 菜单中选择 Find 命令或单击工具栏上的 Find 图标。

(2) 显示 Find Text 对话框，在 Find 编辑框中输入要查找的文本并根据需要设置其他选项。

(3) 单击 Find Next 按钮，系统找到指定文本第一次出现的位置。

(4) 重复执行第(3)步，可找出指定文本的所有出现位置。

结束查找时，单击 Close 按钮关闭对话框。

替换文本的步骤如下：

(1) 从 Search 菜单中选择 Replace 命令或单击工具栏上的 Replace 图标。

(2) 在 Replace 编辑框中输入替换前的文本。

(3) 在 With 编辑框中输入替换后的文本。

(4) 单击 Find Next 按钮查找替换当前文本的下一次出现位置。

(5) 单击 Replace 按钮替换当前文本。

(6) 单击 Replace All 按钮完成所有替换。

6. 添加注释和去掉注释

PowerScript 使用“//”表示单行注释，“//”符号后的所有字符被忽略。可以通过手工输入字符“//”将某行语句变成注释，但用下面的方法可快速地将多行语句变成注释。

(1) 选择要变成注释的行(一行或多行)。

(2) 单击工具栏上的 Comment 图标，所选行即变成注释。

快速地将注释变成代码语句的步骤如下：

(1) 选择要变成语句的行(一行或多行)。

(2) 单击工具栏上的 Uncomment 图标，所选行即变成语句。

4.2.3 粘贴函数

编写代码过程中用到函数时，既可以直接输入函数，也可以使用粘贴方法把函数名粘贴到插入点所在位置。粘贴函数的过程为：

(1) 通过单击选择代码中插入点位置。

(2) 单击画板工具栏上的粘贴函数(Paste Function)图标，显示 Paste Function 对话框，如图 4-16 所示。

(3) 在 Type 列表框中选择要粘贴函数的类型，其中：

- Built-in 为内置函数，例如求绝对值函数 abs()，界面如图 4-17 所示。
- User-Defined 为自定义函数。
- External 为外部函数。

(4) 双击列表框中所需的函数，该函数即被粘贴到代码的插入点位置。

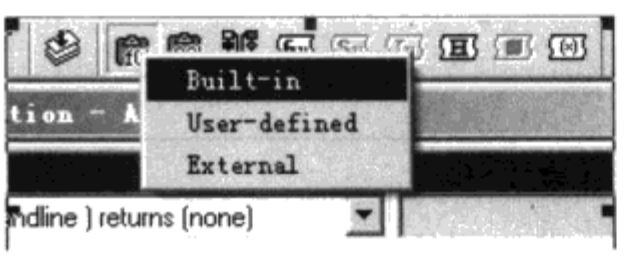


图 4-16　粘贴函数

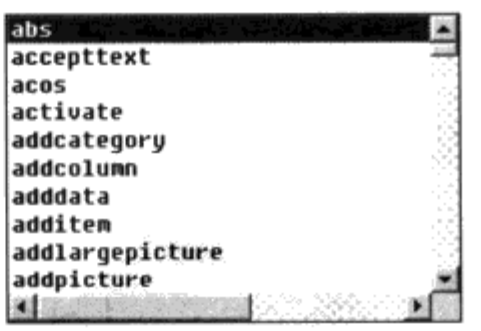


图 4-17　选择内置函数，如 abs()

4.2.4 粘贴语句

与粘贴函数相似，在代码输入过程中也可以粘贴控制语句，其步骤如下：

(1) 单击选择代码中插入点的位置。

(2) 单击画板工具栏上的粘贴语句(Paste Statement)图标，显示 Paste Statement 对话框，如图 4-18 所示。

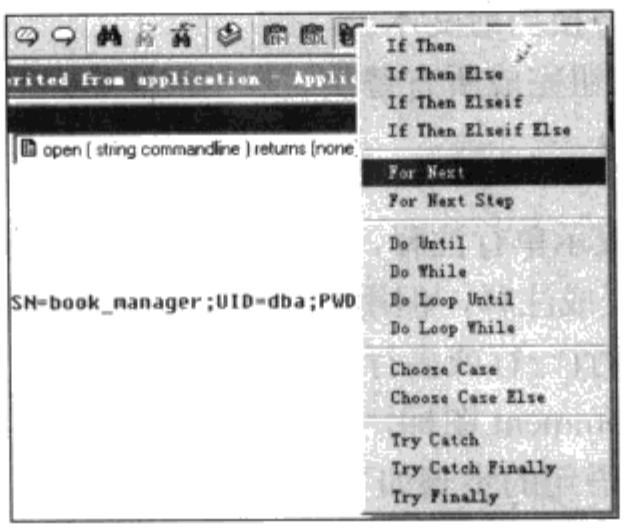


图 4-18 粘贴语句

(3) 单击要插入的语句类型，即可将相应的语句粘贴到 Script 对话框中。

4.2.5 代码编辑器的键盘操作

表 4-1 所示为代码编辑器常用快捷键及其意义，快捷键有助于偏爱键盘操作的读者快速编辑代码。

表 4-1 代码编辑器常用快捷键

快捷键	动 作
Ctrl+n(n=1～7)	打开第 n 个列表框(自左向右数)
Ctrl+F4	关闭代码编辑器
Ctrl+A	选中所有代码
Ctrl+C	复制所选文本
Ctrl+F	查找文本
Ctrl+G	继续查找
Ctrl+Shift+G	转到指定行
Ctrl+L	编译当前程序
Ctrl+Q	退出 PowerBuilder 开发环境
Ctrl+V	粘贴文本
Ctrl+X	剪切文本
Ctrl+Z	取消上次操作
Shift+F6	启动文本编辑器
Shift+F1	显示所选文本的帮助

4.3　事务对象基础

4.3.1　事务对象概述

事务对象(Transaction Object)是 PowerBuilder 中连接应用程序与数据库的专门对象，应用程序与数据库之间的所有通信也都通过事务对象来完成。通常情况下，应用程序需要与数据库相连接，为了使用方便，在启动应用程序时，系统会自动创建一个 SQLCA(SQL Communication Area)的全局事务对象，该对象在应用程序的任何地方都可以被访问。除了直接使用系统的默认事务对象 SQLCA 外，也可以创建自己的事务对象。应用程序只与一个数据库有连接时，直接使用 SQLCA 作为与数据库进行通信的事务对象。只有当应用程序与多个数据库连接时，才需要创建自己的事务对象。

事务对象与 PowerBuilder 中的其他对象不同，它可以由 PowerBuilder 自动创建，也可以由用户自己创建。事务对象均不可见，它是在应用对象画板中的代码编辑器中创建的。无论用嵌入式 SQL 还是通过数据窗口访问数据库，都要使用事务对象；事务对象驻留在内存中，与变量和接口十分相似，它具有用于连接数据库以及从数据库中得到反馈数据的所有信息。事务对象包含了两组属性：

第 1 组属性包含 PowerBuilder 连接数据库需要的所有信息，如数据库厂商名称、数据库实例名称、用户名和密码等。

第 2 组属性反映了最近一次数据库操作完成的状况，包括操作是否成功、错误时的出错信息及出错脚本等。

PowerBuilder 应用程序与存储数据的数据库进行通信的步骤如下：

(1) 设置事务对象的属性值。

(2) 与数据库建立连接。

(3) 执行所需的数据库操作。

(4) 断开与数据库的连接。

事务对象共有 15 个属性，前 10 个用于建立与数据库的连接，后 5 个用于返回数据库的操作状态，在使用时，往往不必对每一个属性都给出数据。每一种 DBMS 所要求的必不可少的属性是不同的，如对 ODBC 要求：

```
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = FALSE
SQLCA.DBParm =" ";
```

表 4-2 所示为事务对象各属性的名称、数据类型及意义。

表 4-2　事务对象的属性

属性名称	数据类型	意　义
DBMS	String	数据库厂商的名称(如 OR7，Sybase，ODBC 等)
Database	String	要连接的数据库名称
UserID	String	与数据库连接的用户名或用户标识

续表

属性名称	数据类型	意　义
DBPass	String	与 UserID 对应的数据库用户的密码
Lock	String	用于连接数据库的隔离层(专用于支持锁值和隔离层的数据库管理系统)
LogID	String	登录数据库服务器所需的用户名或用户标识
LogPass	String	登录数据库服务器所需的密码
ServerName	String	数据库服务器所在的网络名称，即数据库服务器所在网络的地址名(x：=SQLSPX，t：=TCPIP)
AutoCommit	Boolean	定义应用程序是自动提交事务还是手工提交事务
DBParm	String	与具体数据库管理系统相关的一组连接参数
SQLCode	Long	指示最近一次 SQL 语句是操作成功还是失败，其可能取值为：0 表示操作成功；-1 表示操作失败；100 表示操作成功，但无返回数据
SQLNRows	Long	最近一次 SQL 操作所影响的行数(即记录数)，该值由数据库厂商提供，对不同的数据库管理系统(DBMS)，该值含义有所不同
SQLDBCode	Long	由数据库厂商提供的数据库错误代码，对不同的 DBMS，该代码各不相同，但大部分数据库厂商用 0 表示成功，100 表示没找到数据，负数表示错误编号
SQLErrText	String	与数据库错误代码 SQLDBCode 相对应的出错信息
SQLReturnData	String	返回 DBMS 想告诉用户的任何附加信息，例如在程序与 Informix 建立连接的情况下，执行嵌入式 INSERT 语句后，该属性包含插入行的序列号

Autocommit 属性定义应用程序如何提交事务，该属性是为支持自动提交的数据库管理系统而提供的，其取值如下：

- True：PowerBuilder 自动提交每一条 SQL 语句，如果 SQL 语句执行成功，数据库管理系统立即更新数据库，就像 SQL 语句后面紧接着执行了 COMMIT(提交)语句一样。
- False：程序通过提交语句(COMMIT)提交事务。PowerBuilder 在与数据库建立连接之后开始一个事务，在执行 COMMIT 或 ROLLBACK 语句后结束前一个事务并开始下一个事务。False 为默认设置值。

访问事务对象属性的方法与访问其他对象属性的方法相同，都通过点操作符(.)来实现，例如连接 Sybase 数据库所需的代码如下：

```
SQLCA.DBMS = "Sybase"
SQLCA.Database = "BookDB"
SQLCA.LogID = "DBALL"
SQLCA.LogPass = "mypasswd"
SQLCA.ServerName = "playboy"
SQLCA.AutoCommit = FALSE
```

4.3.2 创建自定义事务对象

PowerBuilder 自动为每个应用程序创建一个默认的全局事务对象 SQLCA，如果应用程

序一次只访问一个数据库，那么使用 SQLCA 也就足够了。但如果应用程序需要同时访问两个数据库时，就需要创建第二个事务对象，以完成与第二个数据库的连接。

创建新的事务对象时，首先需要说明一个 Transaction 型变量。变量说明的位置决定了该变量的作用域。与其他变量相似，事务对象变量可以是局部的、实例的、共享的或全局的。说明事务对象的语法为：

```
Transaction my_trans
```

my_trans 就是说明的事务对象变量。

说明事务对象实际上只是说明了一个指向事务对象的指针，在使用新说明的事务对象之前，必须使用 Create 语句完成创建工作，例如：

```
my_trans = Create transaction
```

新创建的事务对象不再使用时，应该及时删除它。使用 Destroy 语句删除事务对象，例如：

```
Destroy my_trans
```

注意： 不需要说明、创建、删除事务对象 SQLCA，PowerBuilder 已经自动处理了这一切。Create、Destroy 语句只用于新建的事务对象。

4.4　实例：创建应用程序对象

本例的任务是在 PowerBuilder 中创建应用程序对象，这是 PowerBuilder 程序开发的第一步。本例将创建 Workspace 文件——MIS.pbw、Target 文件——MIS.pbt、库文件(PBL 文件)——MIS.pbl 和应用(Application)——MIS。

在创建应用对象之前，先要创建一个 Workspace，生成一个 PBW 文件。在这个 Workspace 环境下，创建所需的应用对象(Application)。

创建应用对象有两种方式：Application 和 Template Application。其中通过 Application 方式可以创建简单的应用，生成一个 Application 对象和一个保存对象的 PBL 文件。通过 Template Application 方式可以创建基于模板如 MDI、SDI、PFC 的应用，并且在生成应用对象和 PBL 文件的同时还生成附加的对象。

本例将采用 Application 方式直接创建应用对象。这种方式比较简单，读者只需按照对话框的提示填入相应内容即可。

(1) 在 PowerBuilder 中，单击 PowerBar 中的 New 图标，在打开的 New 对话框中有 7 个选项卡供选择：Workspace；Target；PB Object；DataWindow；Database；Project；Tool。

在 New 对话框中上选择 Workspace 选项卡，其中只有一个图标对象——Workspace 供选择。选择 Workspace 图标，单击 OK 按钮即可。新建 Workspace 窗口的界面如图 4-19 所示。

(2) 在 New Workspace 对话框中填入自定义的 PBW 文件名——MIS，并选择需要的文件夹，单击“保存”按钮。

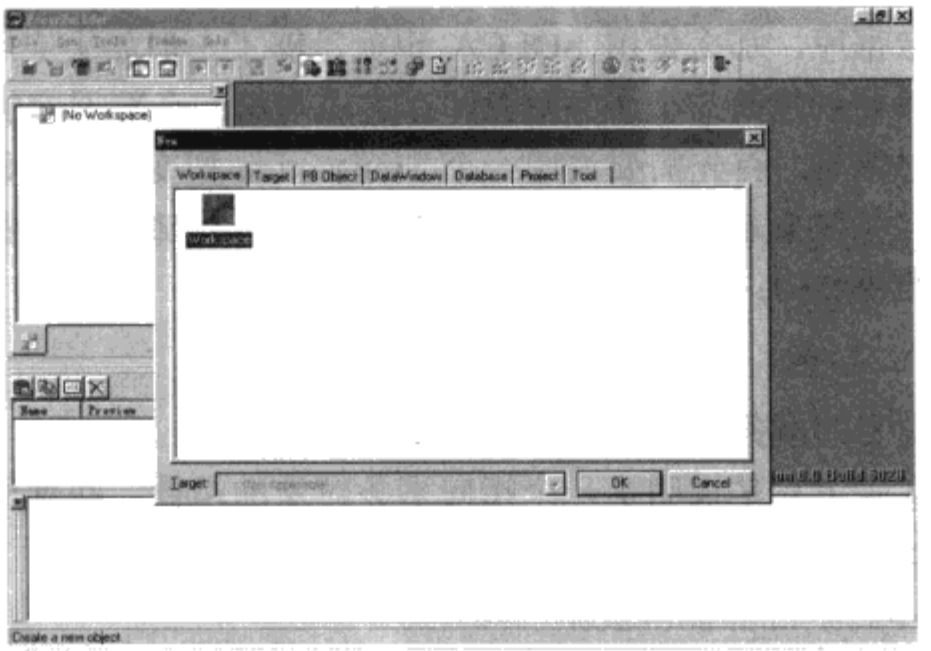


图 4-19　新建 Workspace 窗口

(3)　再次单击 PowerBar 中的 New 图标，在打开的 New 对话框中选择 Target 选项卡，其中有 6 个图标对象供选择：Application；Template Application；Existing Application；EAServer Component；COM/MTS Component；Automation Server。

在 Target 选项卡中选择 Application 图标，如图 4-20 所示。

(4)　在如图 4-21 所示的对话框中填入自定义的 Application 的名字——mis，系统将自动设置与 Application 同名的 Library 和 Target 的文件名：

- Library：E:\PBbook\pb_example\mis.pbl
- Target ：E:\PBbook\pb_example\mis.pbt

填写检查无误后，单击 Finish 按钮完成工作。

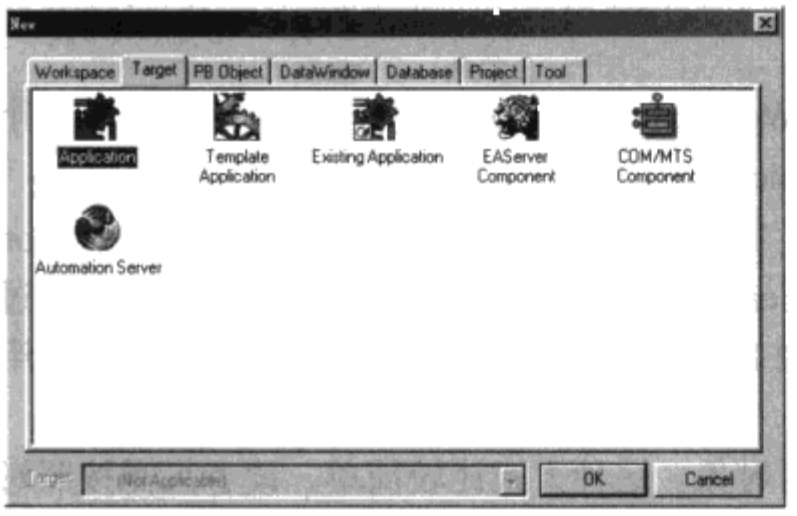


图 4-20　New 对话框的 Target 选项卡

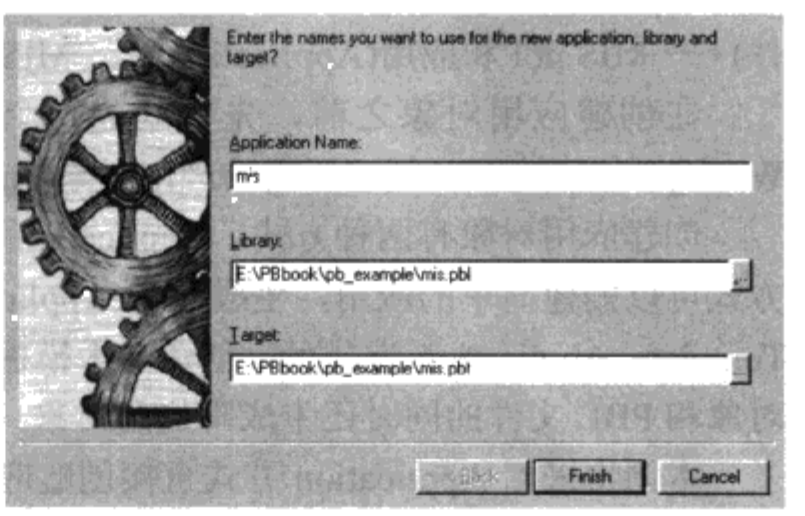


图 4-21　新建 Application 的对话框

(5)　在 Application 面板中单击 Script 图标，编写 Open 事件的脚本如下：

```
//连接数据库，设置数据库连接参数
// Profile db_mis
SQLCA.DBMS = "MSS Microsoft SQL Server 6.x"
SQLCA.Database = "db_mis"
SQLCA.ServerName = "linlin"
SQLCA.LogId = "linda"
SQLCA.AutoCommit = False
SQLCA.DBParm = ""
CONNECT USING SQLCA;
```

```
//连接失败处理
IF SQLCA.SQLCode<>0 THEN
MessageBox("",SQLCA.SQLErrText)
HALT CLOSE
Return
END IF
//打开程序主窗口
open(w_main)
```

完成了上述工作后就得到一个应用程序对象的基本框架。在工作画板左上方的 System Tree 窗口中可以看到工作成果的结构图，Application 脚本经过系统检查无误，结果如图 4-22 所示。

图 4-22　创建应用程序对象的实例效果

通过本例学习了 PowerBuilder 应用程序对象，掌握了创建 PowerBuilder 的 Workspace、Target、库文件(PBL 文件)和应用(Application)的基本方法和步骤。

4.5　习　　题

1.　填空题

(1)　当用户运行应用程序时，首先触发应用对象的_____。

(2)　PowerBuilder 为每个应用程序都提供 5 个默认的全局变量，分别为_____、_____、_____、_____、_____。

(3)　应用对象的 4 个字体属性决定了开发应用程序过程中经常使用的 4 种字体，这 4 个字体属性分别为：_____、_____、_____、_____。

2.　问答题

(1)　应用对象中保存有哪些信息？

(2)　在应用对象画板中可以完成哪些工作？

(3)　添加窗口有哪两种方法？请描述其中一种的具体步骤。

(4)　试创建一个 Workspace 文件和一个应用(Application)。

第5章　窗体技术

窗体是图形用户界面的两个基本要素，通过窗体可以向应用程序提供必要的数据、指示所需完成的工作，显示应用程序的运行状态和执行结果。本章主要讲述如何创建窗体对象和菜单对象、设置属性及编写事件处理程序。

5.1　窗体概述

窗体是很重要的对象，一般用窗体画板来定义，它可以将各种控件和菜单融为一个整体，作为用户与应用程序交互的界面。窗体可以显示信息，并能响应用户的键盘或鼠标操作。窗体有多种风格，在不同的操作系统中，其风格也不同，本节主要介绍 Windows 环境中的窗体特性。

窗体有以下 6 种类型：

- 主窗体(MainWindow)；
- 弹出式窗体(Popup Window)；
- 子窗体(Child Window)；
- 响应窗体(Response Window)；
- MDI 框架窗体(MDI Frame)；
- 带微帮助的 MDI 框架窗体(MDI Frame With Microhelp)。

下面简单地做一下介绍。

1.　主窗体(MainWindow)

主窗体是一个独立的窗体，它不依赖于其他任何的窗体，这种类型的窗体既能够覆盖其他窗体，也能够被其他窗体覆盖。主窗体可以带有菜单，能够被最大化、最小化或改变大小。对功能简单的应用程序，通常直接使用主窗体而不加菜单，图 5-1 就是一个主窗体示例。

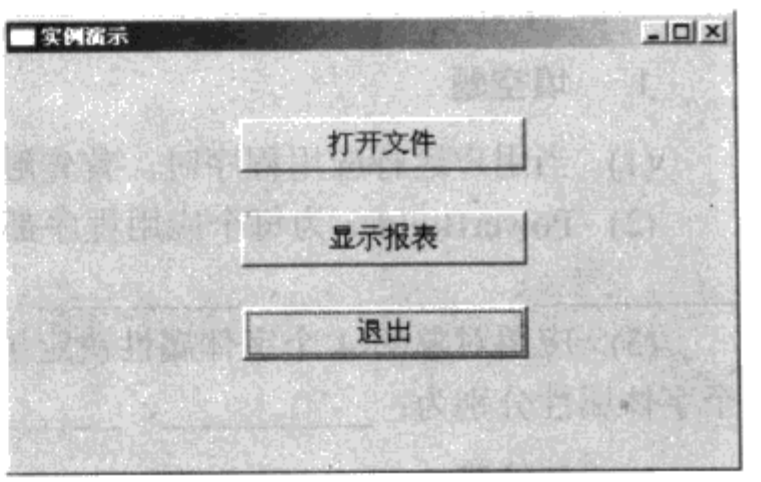


图 5-1　主窗体示例

2.　弹出式窗体(Popup Window)

弹出式窗体通常被另一个窗体打开，它显示在父窗体前面，既可以移出父窗体，也可以覆盖父窗体，但不能被父窗体覆盖。当父窗体最小化时，弹出式窗体随之隐藏；父窗体关闭时，弹出式窗体也随之关闭。弹出式窗体也可以带有菜单，能够被最大化、最小化或改变大小。弹出式窗体用途广泛，可以显示文字信息，也可以做成图形化的进度提示窗体，还能够用于输入用户的选择等。

3. 子窗体(Child Window)

子窗体从属于另一个窗体，它总是在主窗体或弹出式窗体中打开。子窗体可以在父窗体内部移动，但不能移出父窗体。当子窗体的一部分移出父窗体时，系统将自动剪裁子窗体，使得只有父窗体内的部分可见。移动父窗体时，子窗体随之移动，关闭父窗体时，子窗体也随之关闭。子窗体可以有标题条，可以被最大化、最小化及改变大小，但它没有菜单，也不能被作为活动窗体。子窗体在最大化时，它充满父窗体的工作区；子窗体在最小化时，它缩小成图标显示在父窗体的底部。

一般情况下，子窗体应用不多，如果需要子窗体，则应该选用 MDI 窗体类型，它能够自动完成许多窗体的管理工作，避免了一切从头做起的麻烦。

注意： 子窗体的位置总是相对于父窗体，而不是相对于屏幕。

4. 响应窗体(Response Window)

响应窗体向用户请求响应操作，它总是在另一个窗体(父窗体)中打开。在响应窗体显示时，必须在关闭该窗体后，才能操作同一个应用程序中的其他窗体。响应窗体不能被最大化、最小化，但可以带有控制菜单。应用程序中经常使用的 About(关于)窗体、MessageBox(消息)对话框都是响应窗体应用的具体实例。

5. MDI 框架窗体和带微帮助的 MDI 框架窗体

MDI 框架窗体及其子窗体构成了应用程序的多文档界面风格，它有两种形式：普通的 MDI 框架窗体(MDI Frame)和带微帮助的 MDI 框架窗体(MDI Frame WithMicrohelp)。两者的区别在于后一种框架窗体在底部多了一个状态栏，用于显示应用程序的当前信息，称为微帮助(Microhelp)。典型的 MDI 框架窗体由标题、菜单、工具栏、客户区和状态栏组成。

5.2 设 计 窗 体

在应用程序使用具体窗体前，必须首先根据需要创建适宜的窗体，窗体画板正是为此目的设计的。在窗体画板中，能够以直观的、图形化的方式定义窗体和修改窗体，并能够在窗体上放置各种控件，编写窗体及其所属对象的事件处理程序等。

窗体主要由以下 4 部分组成。

- 属性：定义窗体的外观及行为。例如窗体是否有标题栏，窗体的颜色、大小、显示位置和边框形状等。
- 事件：发生在窗体上的事情，通过编写事件处理程序来控制对用户操作的响应。
- 函数：用于操作窗体。
- 控件：放置在窗体中的图形对象，用于请求、显示信息。

由于每个 PowerBuilder 应用程序都要使用窗体，因此创建窗体成为开发应用程序必不可少的步骤之一，创建窗体实际上就是给出窗体的定义，然后在程序中通过调用窗体的 Open 函数来引用定义好的窗体、生成窗体实例等。在 PowerBuilder 中可以创建两类窗体：

- 创建独立窗体，该窗体的创建与其他窗体无关。

- 创建继承窗体，这类窗体继承已定义好的窗体，并增加自己所需的功能，当需要创建一个与已有窗体类似的新窗体时，使用继承的方法创建将收到事半功倍的效果。

如果对已有的窗体不满意，可以修改。创建窗体和修改窗体的工作都是在窗体画板中完成的，而且利用窗体画板还可以在窗体上增加各种控件，以丰富应用程序的功能和界面的表现能力。

5.2.1 窗体画板概述

在开发应用程序时，窗体的创建、修改均通过窗体画板来完成，下面简要介绍窗体画板的组成。进入窗体画板后，会出现如图 5-2 所示的界面。窗体画板工作窗体由 4 部分构成，窗体画板工具栏、文字风格栏、窗体画板工作区和状态栏。窗体画板工具栏由常用图标构成，文字风格栏用于设置所选对象的字体、字型及对齐风格，窗体画板工作区用于定义窗体及安排窗体上的控件，状态栏显示当前操作状态及微帮助。

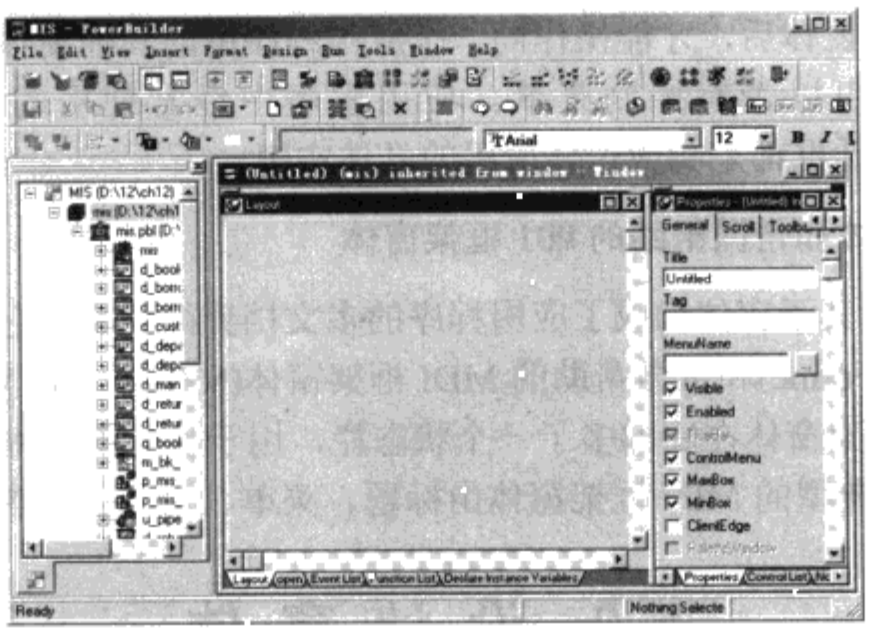


图 5-2 窗体画板工作界面

窗体画板中的文字工具栏用于设置当前对象的标题、字体、字型、大小和文字对齐方式等特性。

技巧： 如果在窗体画板工作界面中没有看到文字风格栏，可以右击画板工具栏，在鼠标光标位置就会弹出一个菜单，选择其中的 StyleBar 菜单项，文字风格栏就出现在屏幕上。

5.2.2 创建新窗体

创建新窗体的步骤如下：

(1) 单击工具栏上的(New)图标，或者选择 File|New 命令，弹出 New 对话框，如图 5-3 所示。

(2) 在 New 对话框中，单击 PB Object 选项卡，然后双击 Window 图标，即可打开窗体设计窗口，如图 5-4 所示。

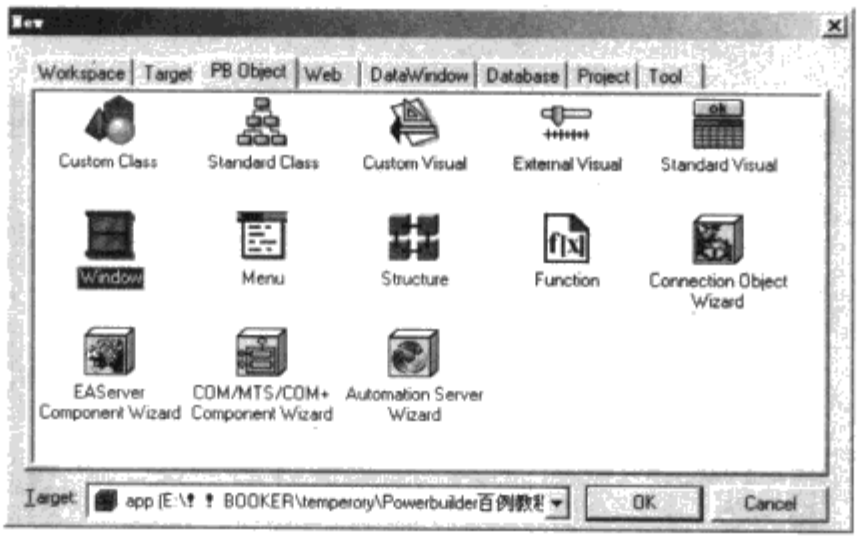


图 5-3　New(新建)对话框

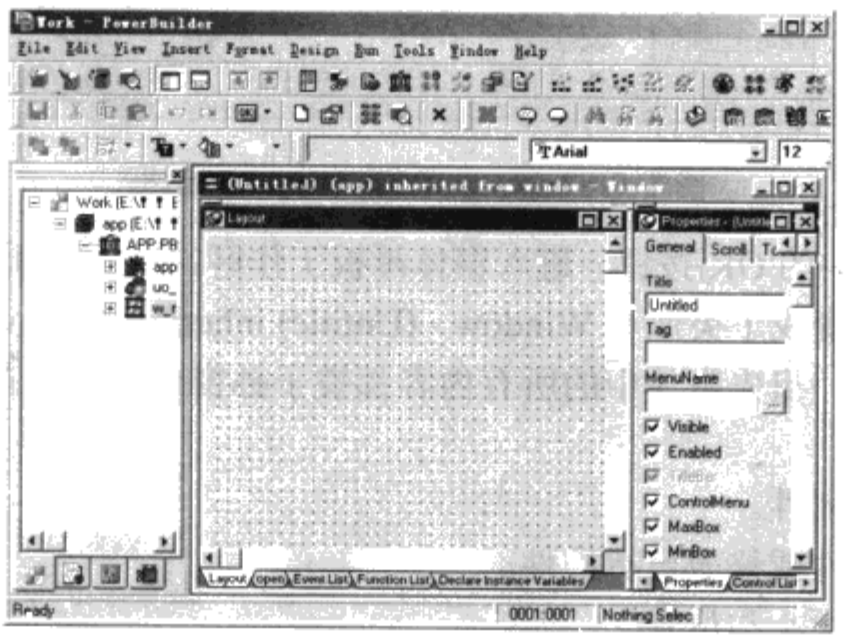


图 5-4　新建窗体

(3)　定义窗体的属性并放置所需的控件。

(4)　单击窗体画板工具栏上的保存图标，保存窗体。

5.2.3　创建继承窗体

继承是 PowerBuilder 面向对象的最重要特征之一。可以把继承简单地看成一种特殊的复制，当对象 B 继承了对象 A(祖先对象)时，对象 B(子孙对象)就拥有一条指向对象 A 的链，当对象 A 改变时，对象 B 也自动跟着改变。

利用继承创建窗体，能够大大减少开发和维护系统的工作量，同时也便于形成标准一致的界面。

在创建继承窗体前，首先需要建立祖先窗体，通常在祖先窗体中放入全部继承窗体需要的公共特征，包括控件与事件处理程序等。创建祖先窗体与创建普通窗体方式相同。

在祖先窗体的基础上，就可以创建继承窗体了。其步骤如下：

(1)　打开窗体画板，单击工具栏上的 Inherit 图标，弹出 Inherit from Object 对话框，如图 5-5 所示。

(2)　单击要继承的窗体，此时在 Windows 编辑框中会出现窗体名。如果要继承的窗体不在当前应用库中，则需要通过单击 Application Libraries 列表框中的选项，选择合适的应

用库。

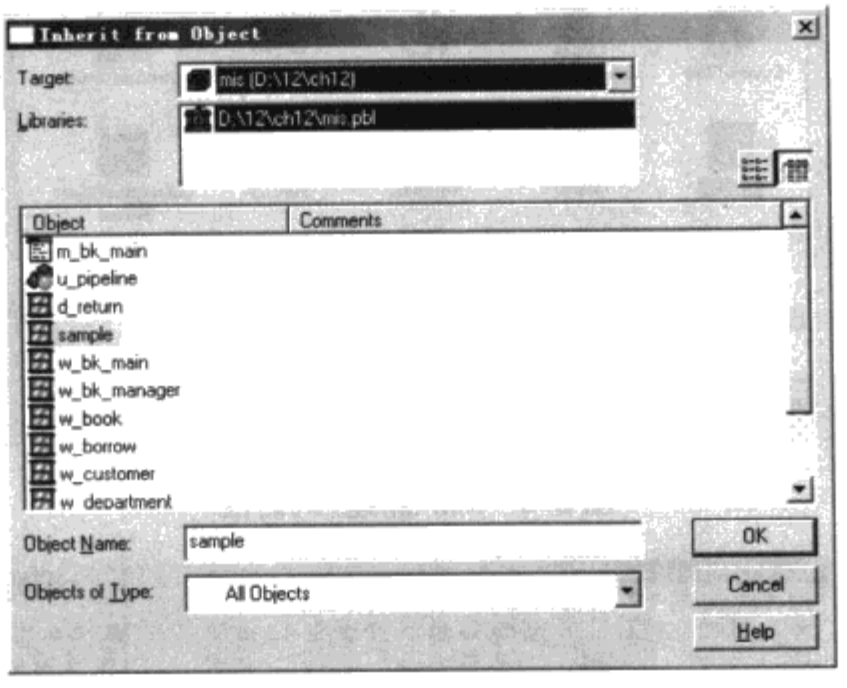


图 5-5 Inherit from Object 对话框

(3) 单击图 5-5 中的 OK 按钮，进入窗体画板工作区，会看到新窗体与祖先窗体相似，只是其标题栏的标题变成了类似于 Window - (Untitle) inherited from w_data 的形式。

创建继承窗体后，祖先窗体中的所有内容都被子孙窗体继承。在子孙窗体中可以完成以下工作：

- 改变窗体的属性。
- 增加新的控件或修改已有控件。
- 改变窗体的大小与位置。
- 编写新的事件处理程序。
- 引用祖先窗体的函数、事件和结构。
- 扩展或替代继承的事件处理程序。
- 声明新的函数、接收结构和变量。
- 声明用户事件。

但在继承窗体中不能删除继承的控件，对于这类控件，当不需要时，可以通过设置其 Visible 属性为 False 隐藏它们。

5.2.4 设计窗体属性

窗体是 PowerBuilder 的对象之一，它也有 3 方面的特征：属性、事件和函数。窗体属性定义窗体的外观、行为和状态，例如窗体是哪种类型的窗体，是否要标题，是否要菜单，窗体的颜色、位置、大小等。

窗体的属性有以下两种类型：

- 预先定义，开发程序时直接使用。
- 自定义，通过定义实例变量和共享变量实现。

下面介绍 PowerBuilder 开发环境中设置窗体属性的步骤，同时分类说明常用窗体属性的作用和用法。

窗体有 30 多个属性，在窗体画板中可以设置大部分常用的属性，其设置步骤如下：

(1) 单击画板栏上的窗体画板图标，打开要查看或设置属性的窗体。

(2) 单击窗体画板工具栏上的属性(Properties)图标，弹出“属性”对话框，如图 5-6 所示。

技巧： 打开“属性”对话框的另一种方法是右击窗体的空白位置，显示一个弹出菜单，从中选择 Properties 菜单项。

(3) 该对话框有 4 个选项卡标签，分别对应 4 个选项卡，通过单击选项卡标签选择所需的选项卡。每个选项卡中都放置了一组功能相近的属性，这些属性可以灵活设置，下面介绍各选项卡中的属性。

1．一般属性

一般属性在 General 选项卡中设置，如图 5-6 所示，根据所选窗体的类型，部分属性可以重新设置，部分属性是只读的，只读属性说明该类型的窗体不具备或不允许修改这些属性。

下面简要介绍选项卡中各属性的意义。

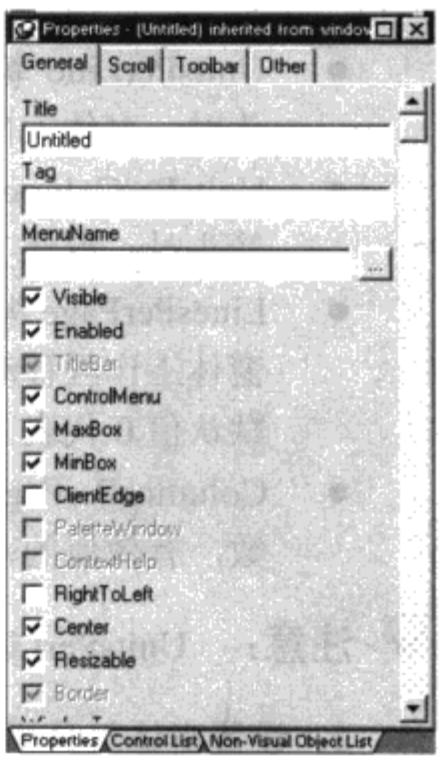


图 5-6　设置一般属性

- Title：Title 编辑框定义窗体的标题，在默认情况下窗体标题将自动置为 Untitled。
- MenuName：MenuName 编辑框决定窗体是否带有菜单，其内容为空时说明没有菜单，否则该编辑框的内容就是菜单的名称。可以直接在该编辑框中输入菜单名称，也可以单击 Browse 按钮，从弹出的 Select Menu 对话框中选择菜单名称。
- Tag：在 Tag 编辑框中可以输入一个与窗体相关的字符串。Tag 属性本身没有特殊的含义，在应用程序中能够随意使用该属性，它的具体含义可以自己定义。
- Visible：指定窗体的可见性，选中该属性后，窗体被打开后出现在屏幕上，否则窗体被隐藏。
- Enabled：决定窗体是否能够接收和发送事件，它并不影响窗体的可见性，选中该属性后，窗体能够接收和发送事件，否则不能操作此窗体。
- Border：指定窗体是否要边框。
- TitleBar：指定窗体是否要标题栏。
- ControlMenu、MaximizeBox、MinimizeBox：指定在窗体的标题栏上是否出现控制菜单按钮、最大化按钮、最小化按钮，选中属性后，相应的按钮出现在窗体的标题栏，否则不出现。
- Resizable：指定当程序运行时是否能够改变窗体的大小。
- WindowType：指定窗体的类型，不同类型的窗体用途不同。
- WindowColor：指定窗体的颜色。

- MDIClientColor：指定 MDI 框架窗体客户区的颜色。
- Icon：设置窗体图标。当程序运行时，窗体图标显示在窗体标题栏的控制菜单位置，当窗体最小化时，它显示在标题旁边。与鼠标指针的设置方法相似，既可以选择系统预定义图标，也可以选择图标文件(以.ico 为后缀)。

2．设置窗体滚动属性

在 Scroll 选项卡中可以设置窗体滚动方式，该选项卡如图 5-7 所示。它有两个复选框，HScrollBar 和 VScrollBar，当选中 HScrollBar 复选框时，在窗体上可以增加水平滚动条；当选中 VScrollBar 复选框时，在窗体上可以增加垂直滚动条；两个复选框都被选中时，在窗体上可以同时增加水平和垂直滚动条。有了滚动条后，可以通过单击滚动条移动窗体工作区，每单击一次滚动条时，窗体工作区移动的多少由该选项卡上部的 4 个编辑框决定，在默认(值为 0)状态下，单击滚动条上的箭头时，滚动的距离为窗体高度或宽度的 1/100，单击滚动条内部时，滚动 10 行或 10 列。根据需要设置这 4 个编辑框的值。

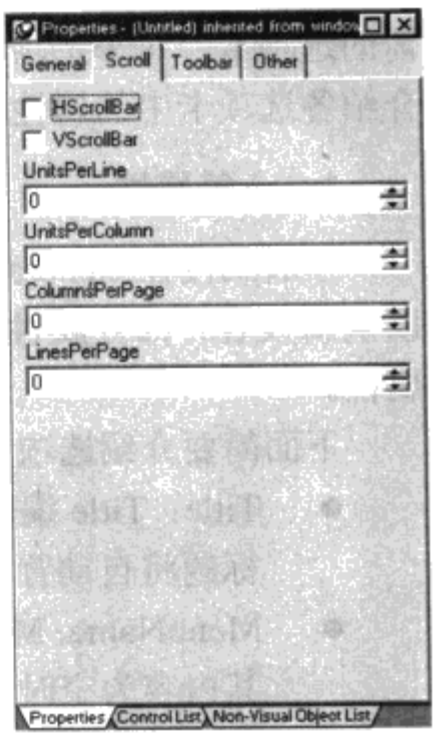


图 5-7　设置滚动属性

- UnitsPerLine 编辑框：指定每次单击垂直滚动条上的箭头时，窗体工作区移动的大小。
- UnitsPerColumn 编辑框：指定每次单击水平滚动条上的箭头时，窗体工作区移动的大小。
- LinesPerPage 编辑框：指定每次单击垂直滚动条内部时，窗体工作区移动的行数，有效值的范围在 1~100 之间，默认值 0 代表 10 行。
- ColumnsPerPage 编辑框：指定每次单击水平滚动条内部时，窗体工作区移动的列数，有效值的范围在 1~100 之间，默认值 0 代表 10 列。

注意： UnitsPerLine 和 UnitsPerColumn 编辑框中指定的大小以 PBU 为单位。

3．设置窗体工具栏属性

在 ToolBar 选项卡中设置窗体工具栏的初始位置，界面如图 5-8 所示。

ToolBarVisible 复选框：指定工具条是否可见，选中时表示可见，否则不可见。

ToolBarAlignment 下拉列表框：指定工具栏显示位置。其中，Top 是指工具条显示在窗体工作区的顶部；Left 是指工具条显示在窗体工作区的左部；Right 是指工具条显示在窗体工作区的右部；Bottom 是指工具条显示在窗体工作区的底部；Float 是指工具条浮动显示在窗体工作区中。

4．窗体位置及大小

窗体位置及大小在 Other 选项卡中设置，该选项卡如图 5-9 所示，各属性的意义如下：

- X、Y：这两个属性指定窗体左上角的位置，对大多数窗体来说，该位置相对于屏幕，但对子窗体来说，该位置相对于其父窗体。

- Width、Height：分别用于指定窗体的宽度和高度。

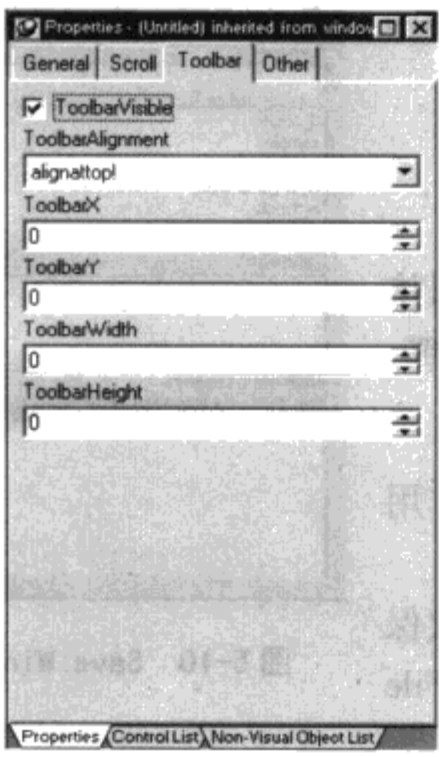


图 5-8　设置工具栏属性

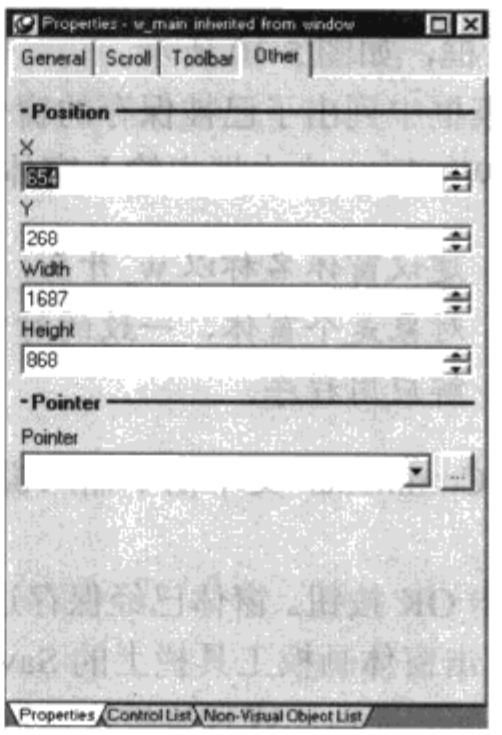


图 5-9　设置其他属性

技巧： 除了直接在 Position 选项卡中输入窗体的起始位置、宽度、高度外，通过拖曳下部大方框中的小窗体的标题栏可以定位设计窗体的 X、Y 位置；将光标移到小窗体的边界上，直到出现双箭头光标，按住鼠标左键不放进行拖曳，能够改变窗体的大小。放开鼠标左键后，新的 X、Y、宽度和高度值出现在相应的编辑框中。

5. 鼠标指针

窗体的鼠标指针在 Other 选项卡中设置。窗体的 Pointer 属性用于指定当程序运行时用户将鼠标指针移动到该窗体中后鼠标指针的形状，在 Pointer Name 编辑框中可以输入两类值：

- 单击 Stock Pointer 列表框中的选项，选择一个预定义鼠标指针。
- 单击 Browse 按钮，从打开的对话框中选择光标文件(以.cur 为后缀)。

无论使用哪种方式选择一个鼠标指针，鼠标指针的样式均出现在列表框的右部。可以为每个窗体及窗体中的控件指定特定指针，一般情况下使用默认值。

5.2.5　保存、预览、打印窗体

下面介绍保存、预览和打印窗体的方法。

1. 保存窗体

在窗体画板中创建新窗体后，第一次保存窗体时，可以进行以下操作：

(1) 单击画板工具栏中的保存(Save)图标或从 File 菜单中选择 Save 命令，或直接按快捷键 Ctrl+S，弹出 Save Window 对话框，如图 5-10 所示。

(2) 列表框中列出了已被保存的窗体。

(3) 在 Windows 文本框中输入窗体的名称。

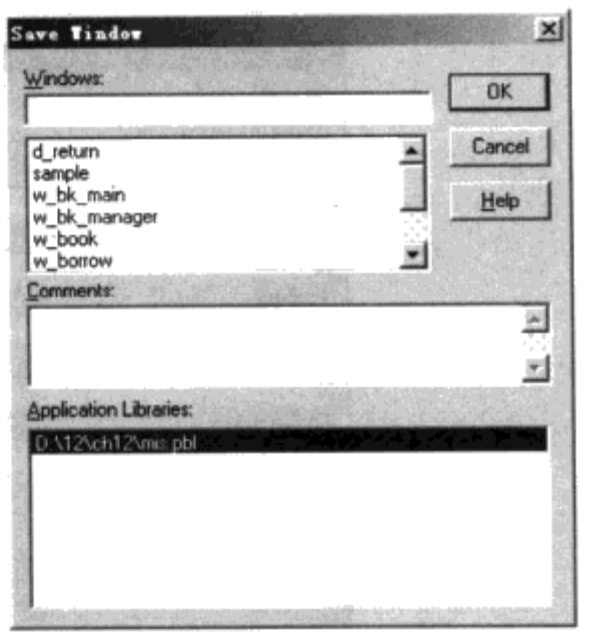


图 5-10 Save Window 对话框

注意： 建议窗体名称以 w_开头，这样易于识别该对象是个窗体，一致的对象命名有助于理解应用程序。

(4) 在 Comments 文本框中输入说明当前窗体作用的注释。

(5) 单击 OK 按钮。窗体已经保存过，若需要再次保存时，只需单击窗体画板工具栏上的 Save 图标，或从 File 菜单中选择 Save 选项，或直接按快捷键 Ctrl+S 即可。

2. 预览窗体

在窗体设计过程中，可以使用预览窗体的方法，查看窗体运行后的情况。其步骤如下：

(1) 单击窗体画板工具栏上的(Preview)图标，窗体运行后的情况即呈现在屏幕上。

(2) 再次单击 Preview 图标，恢复到设计状态。预览窗体显示的是窗体运行时的情况，但实际上并没有真正地运行窗体。

3. 输出窗体定义

输出窗体定义的另一种方法是打印窗体。这里所说的打印窗体不是把窗体的图像在打印机上输出，而是打印窗体的定义，包括窗体的属性、事件处理程序及其控件属性等。

打印窗体的操作步骤很简单，打开窗体后，从 File 菜单中选择 Print 菜单项即可。

5.3 窗体事件和函数

5.3.1 窗体事件

作为应用程序的对象之一，窗体不仅有属性，而且还有丰富的预定义事件，这些事件是应用程序完成预定功能的主要方式。当为某个事件编写了处理程序后，例如编写了窗体的 Clicked(单击)事件处理程序，当事件发生时，事件处理程序就会执行，执行完毕后，窗体等待下一个事件的发生。

PowerBuilder 有很多窗体事件，但不需要对每个事件都编写事件处理程序，而只要对需要响应、需要处理的事件编写事件处理程序即可。不同事件发生的时机不同，有些事件相互间有先后次序之分，充分了解事件的发生时机，才能得心应手地开发 PowerBuilder 面向对象的应用程序。

窗体对象中最常用的事件是 Open、Close、CloseQuery、DragDrop 和 Resize，现在简单地介绍 Open 事件。

当第一次打开窗体时，在窗体显示之前，触发 Open(打开)事件，在 Open 事件发生时，系统已经创建了窗体及窗体中的控件，因此，在这个事件的处理程序中，能够引用这些对象、修改它们的属性等，例如，在 Open 事件中可以将某个按钮暂时隐藏起来。窗体被打开后，窗体类型(Window Type)属性不能再被更改，例如主窗体类型的窗体只能作为主窗体使用，而不能在事件处理程序中将它更改为响应窗体或弹出窗体。

图 5-11　代码编辑器

编写窗体的事件程序的步骤如下：

(1)　单击窗体画板中设计的窗体。

(2)　单击窗体画板工具栏的(编程)图标，进入代码编辑器窗口，如图 5-11 所示。

(3)　在事件列表框中选择要编程的事件，然后根据应用需要编写特定的程序。

窗体对象具备丰富的预定义事件，有 20~30 个之多，表 5-1 所示为 PowerBuilder 预定义的窗体事件、事件参数及触发时机。

表 5-1　窗体事件概览

窗体事件	事件参数	事件触发时机
Activate	无	在激活(Active)窗体前触发，该事件触发后，窗体中第一个跳转次序号最小的对象首先得到焦点。如果窗体中没有这样的对象，则窗体本身得到焦点
Clicked	UnsignedLong flags Integer xpos Integer ypos	用户单击窗体中空白区域时触发，窗体的空白区域指窗体内未被有效控件占据的区域，即窗体中没有可见、有效控件的地方。flags 参数指明用户按了鼠标的哪个键以及按键时是否按住了 Alt 或 Ctrl 键；参数 xpos 指明单击时鼠标指针离窗体左边缘的距离；参数 ypos 指明单击时鼠标指针离窗体上边缘的距离
Close	无	窗体被关闭时触发。触发该事件后，没有办法能够阻止窗体关闭操作
CloseQuery	无	在开始关闭窗体时，触发该事件，该事件返回一个 0 或 1 的返回值。执行该事件的处理程序后，系统检查返回值，如果返回值为 1，则窗体不被关闭，通常情况下，紧随其后触发的 Close 事件不被产生；如果返回值为 0，则窗体被关闭。利用该事件的这种特性，程序能够根据当前状态提醒用户保存数据、询问用户是否真要关闭窗体。窗体被关闭时，同时关闭任何与之相关的子窗体和弹出窗体
Deactivate	无	当窗体变为不活动时触发，例如切换到其他窗体
DoubleClicked	UnsignedLong flags Integer xpos Integer ypos	当双击窗体客户区中任何未被有效控件占用的部分时触发。各参数的意义与 Clicked 事件相同

续表

窗体事件	事件参数	事件触发时机
DragEnter	DragObject source	当可拖放对象进入窗体客户区时触发。参数 source 是个引用，指明被拖动的是哪个对象
DragLeave	DragObject source	当可拖放对象离开窗体客户区时触发。参数 source 是个引用，指明被拖动的是哪个对象
DragWithin	DragObject source	当可拖放对象在窗体客户区中被拖动时触发。参数 source 是个引用，指明被拖动的是哪个对象
Hide	无	在窗体被隐藏前的瞬间触发
HotLinkAlarm	无	在动态数据交换(DDE)服务器应用发送了新的(修改后的)数据、且客户 DDE 应用程序已经接收到数据时触发
Key	key (枚举型键码) Ulong keyflags	当在键盘上按下一个键且插入点不在编辑区域(如单行编辑框、超文本编辑框等)中时触发。参数 key 是个枚举型数据，指示用户按下了哪个键，例如 KeyA!或 KeyF1!等(其他值可从对象浏览器中查到)；参数 keyflags 指明用户按键时，是否同时按住了 Alt 和/或 Ctrl 键
MouseDown	UnsignedLong flags Integer xpos Integer ypos	当用户在窗体客户区中任何未被有效控件占用的部分按下鼠标左键时触发。各参数的意义与 Clicked 事件相同，flags 的值总为 1
MouseMove	UnsignedLong flags Integer xpos Integer ypos	当鼠标在窗体内移动时触发。各参数的意义与 Clicked 事件相同
MouseUp	UnsignedLong flags Integer xpos Integer ypos	当用户在窗体客户区中任何未被有效控件占用的部分放开鼠标左键时触发。各参数的意义与 Clicked 事件相同
Open	无	在窗体打开之后、显示之前触发，此时系统已经构造好了窗体的所有属性及其上的所有控件。下述函数触发窗体的 Open 事件：Open、OpenWithParm、OpenSheet、OpenSheetWithParm 函数
Other	Ulong wparam Long lparam	当非 PowerBuilder 事件的 Windows 消息出现时触发，该事件包括了除 PowerBuilder 10.0 的窗体事件外的所有 Windows 消息。其参数对应于 Windows 消息的参数

续表

窗体事件	事件参数	事件触发时机
RbuttonDown	UnsignedLong flags Integer xpos Integer ypos	当用户在窗体客户区中任何未被有效控件占用的部分按下鼠标右键时触发。各参数的意义与 Clicked 事件相同
RemoteExec	无	RemoteExec 无当一个 DDE 客户应用程序发送了一条命令时触发
RemoteHotLinkStart	无	当一个 DDE 客户应用程序要开始一个热连接(Hotlink)时触发
RemoteHotLinkStop	无	当一个 DDE 客户应用程序要结束一个热连接(Hotlink)时触发
RemoteRequest	无	当一个 DDE 客户应用程序请求数据时触发
RemoteSend	无	当一个 DDE 客户应用程序已经发送了数据时触发
Resize	ULong sizetype Integer newwidth Integer newheight	当窗体大小变化时触发，窗体被打开时也触发此事件。参数 sizetype 指明改变窗体大小的类型(最小化、最大化、恢复等)；newwidth 指明窗体的新宽度；newheight 指明窗体的新高度
Show	Booleashow Long status	窗体显示之前触发。窗体的 Show 函数将触发该事件。参数 Booleashow 的值总为 TRUE；status 指明窗体的状态
SystemKey	key(枚举型键码) ULong keyflags	当插入点不在编辑框中且用户按下 Alt 或 Alt+其他键时触发。各参数的含义与 Key 事件类似
Timer	无	在调用 Timer 函数启动定时器、设定时间后触发
ToolbarMoved	无	当 MDI 窗体上的工具栏被移动时触发

5.3.2　窗体函数

PowerBuilder 提供了一组函数用于操作窗体，这组函数既包括系统函数(如 Open()、Close()函数等)，也包括窗体对象函数(如 Print()函数)，下面介绍几个常用函数。

1．打开窗体函数

在应用程序中使用窗体的第 1 步工作是通过 Open()函数打开窗体，打开之后才能对窗体进行各种操作。Open()函数有两种语法格式，第 1 种语法格式为：

```
Open( windowvar{, parent})
```

其中，参数解释如下：

- 参数 windowvar：要显示窗体的名称，可以直接使用窗体画板中定义的窗体名。
- 参数 parent：可选项。未省略时，由该参数指定父窗体的名称，此时被打开的窗

体以子窗体形式出现。

Open()函数返回整型值 1 表示打开窗体成功，返回整型值-1 表示打开窗体失败。

例如用窗体画板定义了一个窗体 w_main，在应用对象的 Open 事件中写入代码：

```
Open(w_main)
```

程序运行后，窗体 w_main 就会出现在屏幕上。

Open()函数的第 2 种语法格式为：

```
Open(windowvar ,windowtype{, parent } )
```

其中，参数解释如下：

- 参数 windowvar、parent 的意义与第 1 种格式相同。
- 参数 windowtype 是个字符串，其值是希望打开的窗体的类名。

注意： windowtype 必须与参数 windowvar 的类继承链中某个类相同。

下面的代码首先打开由字符串变量 s_w_name 指定类型的窗体，并用窗体变量 w_to_open 保存对该打开窗体的引用，SELECT 语句用于从数据库中检索窗体类型数据，并保存到 s_w_name 变量中。

```
Window w_to_open
string s_w_name
SELECT next_window INTO :s_w_name FROM routing_table
WHERE …;
Open(w_to_open, s_w_name)
```

2. 关闭窗体函数

当打开的窗体不再使用时，应该及时将其关闭，以释放系统资源。关闭窗体的工作由 Close()函数实现，其语法格式为：

```
Close ( windowname )
```

其中，windowname 是要关闭窗体的名称。

该函数返回 1 表示关闭窗体成功，返回-1 表示关闭不成功。

针对前面小节示例中打开的窗体 w_main，当需要关闭时，在事件处理程序中填写以下语句：

```
Close(w_main)
```

3. 带参数打开窗体函数

有些窗体应用程序要反复使用，每次使用时都向窗体提供不同的参数，此时，利用带参数打开窗体函数 OpenWithParm ()既方便又简单。

与 Open()函数一样，OpenWithParm()函数也有两种语法格式。

第 1 种格式：在打开一个设计程序之前就知道其类型的窗体对象。

第 2 种格式：打开一个程序运行时才知道其类型的窗体对象。

下面分别予以介绍。第 1 种语法格式为：

```
OpenWithParm ( windowvar, parameter {,parent})
```

其中 windowvar 是想显示的窗体的名称；parameter 是希望传递的参数，该参数将被放置到系统默认的全局对象 Message 中，parameter 的数据类型只能是下述 4 种类型之一。

- String：字符串。
- Numeric：系统支持的所有数值数据类型，包括 PowerBuilder 的数值数据类型(Integer、Long、Decimal、Real、Double、Unsigned Integer 和 Unsigned Long)及当前数据库管理系统支持的数值数据类型。
- PowerObject：系统对象的基类，实际上包括了 PowerBuilder 的所有对象。
- Parent：可选参数，未省略时，由该参数指定父窗体的名称，此时被打开窗体作为子窗体形式出现。

OpenWithParm()函数返回整型值 1 表示打开窗体成功，返回整型值-1 表示打开失败。

打开窗体时，OpenWithParm()函数把参数 parameter 保存到系统默认的全局对象 Message 中，根据数据类型的不同，在被打开窗体的事件处理程序中，通过检查 Message 对象的相应属性得到打开函数传递来的参数值，参数类型及 Message 对象相应属性的对应关系如表 5-2 所示。

表 5-2　parameter 数据类型及其使用的 Message 属性

parameter 数据类型	使用的 Message 属性
Numeric	Message.Double
ParmPowerObject(包括各种对象以及用户自定义结构)	Message.PowerObjectParm
String	Message.StringParm

在打开窗体中，应该尽早地检查 Message 对象相应的属性，原因在于该对象是全局对象，其他程序完全可能为其他目的使用该对象。

下面的语句打开窗体 w_employee，并用 w_employee 引用该窗体，打开窗体的同时向窗体传递一个字符串型参数“张峰”：

```
OpenWithParm(w_employee, "张峰")
```

在窗体 w_employee 的 Open 事件中，可以用下述语句把 OpenWithParm()传递来的参数显示在单行文本框 st_empname 中：

```
st_empname.Text = Message.StringParm
```

OpenWithParm()函数的第 2 种语法格式为：

```
OpenWithParm( windowvar, parameter , windowtype {,parent})
```

其中参数 windowvar、parameter 和 parent 的意义与第 1 种格式相同。参数 windowtype 是个字符串，其值是希望打开窗体的类名。

下面的语句打开由字符串变量 c_w_name 指定的窗体，并把窗体引用保存到变量 wc_to_open 中，且使窗体 w_emp 成为当前要打开窗体的父窗体，同时向被打开窗体传递整型参数 age(因此该参数值将保存在 Message.DoubleParm 中)。

```
window wc_to_open
string c_w_name
```

```
integer age = 60
c_w_name = "w_c_emp1"
OpenWithParm(wc_to_open, age, c_w_name, w_emp)
```

技巧： 同时向窗体传递多个参数时，可以首先定义一个覆盖各参数的结构，然后说明该结构的一个变量，为结构赋值后，将整个结构作为一个变量，并使用 OpenWithParm()函数传递。

4. 带参数关闭窗体函数

在响应窗体(Response Window)中，可以使用 CloseWithReturn()函数关闭窗体同时向其父窗体的事件程序返回一个值。与 OpenWithParm()函数相似，CloseWithReturn()函数同样借用全局对象 Message 保存返回的值。

CloseWithReturn()函数的语法格式为：

```
CloseWithReturn(windowname, returnvalue)
```

其中，windowname 是要关闭的窗体；returnvalue 是返回值，该值保存在消息对象 Message 的属性中，可为下述 3 种数据类型之一，String、Numeric 或 PowerObject，具体解释参见 OpenWithParm()函数。

假设下面的语句放在某个响应窗体上的某按钮的事件处理程序中，单击了该按钮后将关闭窗体，同时返回一个字符串“张峰”：

```
CloseWithReturn(Parent, "张峰")
```

通过下述语句得到响应窗体的返回值：

```
ret_name=Message.StringParm
```

5. 打印窗体函数

利用 Print()函数可以将窗体的内容打印输出。该函数的语法格式为：

```
objectname.Print( printjobnumber, x, y {,width, height})
```

其中，参数解释如下：

- objectname 是任意从 DragObject 类继承得到的对象，包括窗体以及窗体上的控件。
- printjobnumber 是个表示打印作业号的长整数，打印作业号可用函数 PrintOpen()来得到。
- x、y 参数是个整型值，指定当前对象在打印页上开始打印的位置，以 1/1000 英寸为单位。
- width、height 是两个可选参数，分别指定当前对象在打印页上打出后的宽度和高度，以 1/1000 英寸为单位，省略这两个参数时，按对象本身的大小进行打印。

在使用 Print()函数打印窗体之前，必须先调用 PrintOpen()函数建立打印作业，打印完毕后，使用 PrintClose()函数关闭打印作业。

示例：打印 w_about 窗体。

```
long Job
Job = PrintOpen()
If  Job <> -1 Then
```

```
 w_about.Print(Job, 500,1000)
PrintClose(Job)
End If
```

上述程序在打印页的 500/1000、1000/1000 英寸处开始以窗体本身的大小打印 w_about 窗体。

如果程序需要打印整个屏幕，可以使用 PrintScreen()函数，该函数与 Print()函数有相同的参数，使用 PrintScreen()函数时无须用对象名限定，下面是打印屏幕的一段示例程序：

```
long Job
  Job = PrintOpen( )
 If  Job <> -1 Then
  PrintScreen(Job, 500,1000)
 PrintClose(Job)
End If
```

5.4　窗体变量和窗体数组

当应用程序需要使用多个窗体实例时，窗体数组是个良好的解决方案。本节主要介绍如何声明窗体变量和窗体数组，并给出了具体的实例加以说明。

5.4.1　窗体变量

在画板中定义窗体后，实际上也定义了一个与窗体同名的数据类型，因此也可以说明该数据类型的变量。说明窗体类型变量的方法与说明整型变量的方法相同，格式如下：

数据类型 变量名

例如要打开一个窗体的多个实例，可以使用如下语句说明一个窗体类型为 w_person 的窗体变量 w_my_person。

```
w_person w_my_person
```

使用如下语句创建 w_person 类型的窗体实例。

```
open(w_my_person)
```

把上述两条语句分别放到某个按钮的事件处理程序中，程序运行后，每单击一次按钮就会创建一个新的窗体实例。

但是以这种方式处理时，却无法从一个实例中引用另外一个确定的实例，如何解决这个问题呢？可以使用窗体数组。

5.4.2　窗体数组

首先是说明窗体数据类型的数组。例如下面的语句说明了一个名称为 w_array[6]的窗体数组，它有 6 个数组元素，用于保存至多 6 个窗体实例。

```
w_person  w_array[6]
```

使用 Open()函数打开一个窗体实例，例如：

```
Open(w_array[1])    //打开 w_person 的第 1 个实例
```

```
Open(w_array[2])    //打开 w_person 的第 2 个实例
```

利用窗体数组，能够操纵指定的窗体实例，例如使用下面的语句可以将第 2 个窗体实例隐藏起来：

```
w_array[2].Hide()
```

访问指定窗体实例中控件的属性也很容易，例如 w_person 上有个单行文本框 sle_name，若需要知道在第 2 个窗体实例中向该文本框中输入的内容时，可以使用以下语句：

```
name=w_array[2].sle_name.text
```

字符串变量 name 保存了用户输入的内容。

注意： 在前面的例子中，窗体数组实例保存的窗体有相同的窗体类型，实际上，也可以使用同一个数组保存不同类型的窗体实例，这里需要了解窗体的继承关系。

所有窗体均是从内部数据类型 Window 继承得到的，也就是说，定义的所有窗体都是 Window 的后代，Window 数据类型定义了所有窗体共同使用的属性，例如窗体起始位置、宽度、高度、标题等。如果在应用中说明一个 Window 类型的变量，则该变量可以引用应用中任何类型的窗体。假设窗体 w_customer 从窗体 w_person 继承得到，由于所有窗体都是 Window 类型的后代，因此窗体间的完整继承关系如图 5-12 所示。

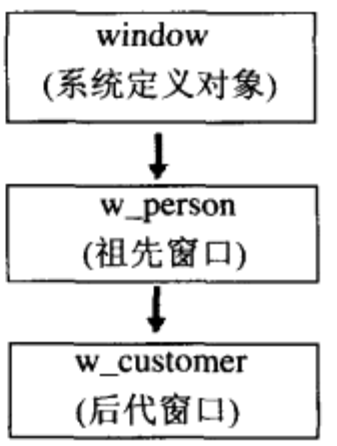


图 5-12 窗体间的继承关系

5.5 窗体设计实例

5.5.1 创建图标栏应用

在一些应用场合，需要在 Windows 中的图标栏中显示出应用程序的图标，在单击图标时弹出相应的窗口。下面介绍在 Windows 下显示图标栏图标的方法。

1. 建立一个应用程序

在创建程序前，准备一个用于在图标栏中显示的图标文件(.ico)和用于弹出窗口显示的图片(BMP 256 色)。启动 PowerBuilder，创建一个应用(Application)。

2. 创建要显示的图标窗口

创建一个用于显示图标的窗口，取名为 W_systray，在上面放置一幅 BMP 图片，然后设置窗口的其他有关信息。

3. 编程

(1) 定义全局外部函数。PowerBuilder 本身并不提供用于在图标栏中把窗口显示成图标的函数，但可以通过 Windows API 调用来实现。本例在全局外部函数中调用如下 API 函数：

```
Public Function Integer Shell_NotifyIcon
(Long dwMessage, Any lpData) Library "shell32" Alias For "Shell_NotifyIconA"
Public Function Long LoadImage(Long hInst, String lpsz, Long un1, Long n1, &
Long n2, Long un2) Library "user32" Alias For "LoadImageA"
Public Function Long DestroyIcon(Long hIcon) Library "user32" Alias For
"DestroyIcon"
Public Function Long SetForegroundWindow (Long hwnd)Library "user32"&
Alias For "SetForegroundWindow"
Public Function Long OpenIcon(Long hwnd) Library "user32" Alias For
"OpenIcon"
```

(2) 编写 Open 事件。在 Application 的 Open Scripts 中，输入下列函数。

```
Open(Win_systray)
```

(3) 窗口函数和 Scripts。在 W_systray 窗口中，创建一个名为 notifyicondata 的结构。

(4) 定义以下两个函数。

① AddToTray()。参数：无；返回值：布尔值(True/False)。

```
// AddToTray ()函数
Any nid
if hIcon = 0 then
hIcon = LoadImage(0, "SysTray.ico", IMAGE_ICON, 0, 0, LR_LOADFROMFILE)
end if
if hIcon = 0 then
MessageBox ("错误", "不能加载图标!")
Return False
else
nid = SetNotifyIconData(Handle(This), 0, NIF_MESSAGE or NIF_ICON&
or NIF_TIP, WM_MOUSEMOVE,hIcon, "TestTip")
Shell_NotifyIcon(NIM_ADD, nid)
Return True
end if
```

② RemoveFromTray ()。参数：无；返回值：无。

```
// RemoveFromTray ()函数
Any nid
nid = SetNotifyIconData(Handle(This), 0, NIF_MESSAGE or NIF_ICON &
or NIF_TIP, 0, hIcon, "")
Shell_NotifyIcon(NIM_DELETE, nid)
if hIcon < > 0 then DestroyIcon(hIcon)
hIcon = 0
SetNotifyIconData(Long hWnd, Long ID, Long Flags, Long CallbackMessage,&
   Long Icon, String Tip);
//返回值: notifyicondata
Char MyTip [64]
NotifyIconData NidTemp
NidTemp.cbSize = 88 // Len (NidTemp)
NidTemp.hWnd = hWnd
NidTemp.uID = ID
NidTemp.uFlags = Flags
NidTemp.uCallbackMessage = CallbackMessage
NidTemp.hIcon = Icon
MyTip = Tip + Char (0)
NidTemp.szTip = MyTip
return NidTemp
```

(5) 最后，在窗口的有关事件中输入函数。

```
//Open 事件
```

```
if AddToTray() then Visible = False

//Close 事件
RemoveFromTray()

//CloseQuery 事件
if CanClose then
Return 0
else
Visible = False
Return 0
End if

//Resize 事件
if sizetype = 1 then Visible = False
```

4. 测试

完成上述工作后，就可以运行程序并进行测试了。运行该程序，在图标栏中注册一个图标，单击该图标，弹出一个窗口。最小化该窗口后，它不会在 Windows 的任务栏中显示，而是缩成一个图标栏中的图标。

5.5.2 创建美观的状态条

MDI Frame with Microhelp 窗口自带的状态条只能显示一行信息，且中文字体单一，因此有必要创建一个美观的状态条。下面提供了一种创建美观状态条的方法，步骤如下：

(1) 创建一个数据窗口(External)，此数据窗口包含了需要在状态条上显示的内容，如用户、时间、帮助信息等。

(2) 将此数据窗口添加到 MDI Frame Without Microhelp 式的窗口中，命名为 dw_frame_status_bar。

(3) 在 Resize 事件中调整对应的数据窗口及 MDI_1 的位置。

```
integer li_height,li_width
li_height = this.workspaceHeight()
li_width = this.workspaceWidth()
mdi_1.x = this.workspacex()
mdi_1.y = this.workspacey() - 4
mdi_1.height = li_height - dw_frame_status_bar.height + 8
mdi_1.width= li_width
dw_frame_status_bar.x = this.workspacex() - 4
dw_frame_status_bar.y = mdi_1.y + mdi_1.height
dw_frame_status_bar.width = li_width + 4
```

(4) 重载 setmicrohelp()为 Global Function 函数，在数据窗口中显示相关信息。

经过以上步骤，可以制作出较为美观的状态条。

5.5.3 在 Response 窗口中使用 Menu

Response 窗口不能直接放置 Menu，但可以通过脚本来实现，只需在窗口的 Open 事件中加入以下代码：

```
This.changemenu(m_test)
```

5.5.4　移动窗口中的控件

在 Mousedown 事件中写入以下脚本即可在窗口中任意移动此控件：

```
Send(handle(this), 274, 61458, 0)
```

如果控件没有 mousedown 事件，定义事件如下：

```
ue_mousedown pbm_lbuttondown
```

将以上代码写在窗口的 mousedown 事件中，在窗口中任意位置按下鼠标左键即可拖动窗口。

5.5.5　捕获控制菜单中的 Close 事件

单击控制菜单中的 Close 按钮，关闭窗口的同时会触发 pbm_syscommand 事件，定义事件如下：

```
ue_close pbm_syscommand
```

在事件中加入以下代码：

```
IF Message.WordParm = 61536 THEN
MessageBox("","窗口将关闭")
END IF
```

5.5.6　将可视控件放置于其他的控件上

具体实现方法如下：

(1) 创建用于放置在状态栏上的可视控件的用户对象，这里以一个进度条控件(ProgressBar)为例。

(2) 获取窗口状态栏的句柄。

```
//wf_getstatusbar(window awFrame):
long ll_null,ll_hstatusbar
string ls_classname,ls_windowname
Environment lEnv
if isvalid(awFrame) and (GetEnvironment(lEnv) = 1 then
if (awFrame.WindowType = MDIHelp!) then
choose case lEnv.PowerBuilderMajorRevision
case 5  ls_classname = PowerBuilder5_StatusClass
case 6  ls_classname = PowerBuilder6_StatusClass
case 7  ls_classname = PowerBuilder7_StatusClass
end choose
setnull(ls_windowname)
setnull(ll_null)
//首先声明 FindWindowExA 是 Windows API 函数
ll_hstatusbar = &
FindWindowExA(Handle(awFrame),ll_null,ls_classname,ls_windowname)
end if
end if
return ll_hstatusbar
```

(3) 利用 SetParent 函数将控件放在状态栏上，窗口的 Instance 变量如下：

```
u_ProgressBar iuo_ProgressBar
long il_hstatusbar
```

窗口必须是 MDIHelp 类型，在窗口的 Open 事件中编写以下代码：

```
long ll_hstatusbar,ll_parts[]
ll_hstatusbar = wf_getstatusbar(this)
if (ll_hstatusbar > 0) then
il_hstatusbar = ll_statusbar
OpenUserObject(iuo_ProgressBar,0,8)
if isvalid(iuo_ProgressBar) then
//首先声明 SetParent 是 Windows API 函数
SetParent(handle(iuo_ProgressBar),ll_hstatusbar)
end if
end if
```

5.5.7 实现动态显示控件

在应用程序中有时需要动态显示控件，例如在一个向导窗口中需要根据用户当前的选择来决定下一步要显示的控件。一般可以使用两种方法来实现：

(1) 设计时将需要显示的控件全都放置在窗口中，运行时通过控制控件的 Visible 属性来显示控件。

(2) 运行时动态生成控件。

下面以动态生成一个 SingleLineEdit 控件为例，说明动态生成控件的方法。

```
SingleLineEdit lsle_1
OpenUserObject( lsle_1, "singlelineedit", 400, 400)
lsle_1.Width = 1000
lsle_1.Height = 80
lsle_1.TabOrder = 30
lsle_1.Border = True
lsle_1.BorderStyle = StyleLowered!
lsle_1.text = "这是一个动态创建的控件！"
lsle_1.visible = True
parent.control[ UpperBound( parent.control ) + 1] = lsle_1
```

5.5.8 动态创建可视控件

下面简单介绍如何创建动态可视控件，以创建一个 Button 控件为例进行说明。

(1) 建立一个 uo_button 用户对象(从 VisualStandard|CommandButton 派生)。

① 声明用户对象实例变量：

```
Private String h_event
PowerObject u_obj
```

② 声明用户对象函数：

```
void SetWidth(long u_width)
void SetHeight(long u_height)
void SetX(long u_x)
void Sety(long u_y)
void EnableDrag()
void DisEnableDrag()
void SetEvent(window obj,String u_event)
//这里用 window 就只能在 window 中调用
```

③　函数的实现：

```
void SetWidth(long u_width) {this.width=u_width}
void SetHeight(long u_height) {this.height=u_height}
void SetX(long u_x) {this.x=u_x}
void Sety(long u_y) {this.y=u_y}
void EnableDrag() {this.dragauto=true}
void DisEnableDrag() {this.dragauto=false}
void SetEvent(window obj,String u_event) { this.h_event=u_event
this.u_obj=obj }
```

④　uo_button 的 Clicked 事件加入如下代码：

```
u_obj.triggerevent(h_event)
```

(2)　建立以上用户化的按钮后，就可以在程序运行时动态创建控件了，步骤如下：

①　建立一个 w_main 窗体和一个 m_button 按钮。

②　声明窗体实例变量 uo_button test。

③　w_main 的 Open 事件加入以下代码：

```
test=create uo_button
```

④　w_main 的 Close 事件加入以下代码：

```
destroy test
```

⑤　声明窗体用户事件 My_event 并加入以下代码：

```
Messagebox("测试","这是一个动态创建的按钮!")
```

⑥　m_button Clicked 事件加入以下代码：

```
Open Userobject (test)
test.SetEvent(parent,"MY_EVENT")
```

这样就动态创建了一个按钮，调用 Enable Drag()函数，并在 w_main 的 Dragwithin 事件中加入以下脚本，就可拖动按钮定位。

```
SetX(this.pointerX())
SetY(this.pointerY())
```

注意： 还可以根据自己的需要设计用户对象函数。如果将 test 声明成数组就可以创建一系列动态控件，还可以把动态控件的属性放入数据库中，实现设计向导的功能。

5.5.9　使用多窗口实例

窗口实际上也是数据类型，它可以说明窗口类型的变量。当为一个窗口类型说明了多个变量时，这个窗口类型就可以有多个窗口实例。

说明窗口类型的变量就像说明整型或字符串型变量一样简单。例如建立 w_employee 窗口的同时，自动生成一个 w_employee 数据类型，可以使用 w_employee 数据类型来说明变量，在程序中可以使用如下语句：

```
w_employee mywin
```

它说明了一个类型为 w_employee，名称为 mywin 的变量。

下面通过一个实例来说明多窗口实例的使用方法。

(1) 建立窗口 w_employee，窗口类型为 Popup!，在窗口上有“关闭窗口”按钮，在该按钮的 Clicked 事件中加入以下语句：

```
Close(parent)
```

(2) 建立窗口 w_main，窗口类型为 Main!，说明两个为 w_employee 窗口定位的实例变量：

```
int xpos=200,ypos=200
```

在窗口上有“建立窗口实例”按钮，在该按钮的 Clicked 事件中加入以下语句：

```
w_employee mywin
open(mywin)
xpos = xpos + 100
ypos = ypos + 100
mywin.x = xpos
mywin.y =ypos
```

(3)在应用对象的 Open 事件中使用以下命令：

```
Open(w_main)
```

按照以上步骤完成的应用程序，执行的结果如图 5-13 所示。每次单击 w_main 窗口的“建立窗口实例”按钮，都将建立一个新的 w_employee 窗口实例。单击 w_employee 窗口实例上的“关闭窗口”按钮，则关闭当前的窗口实例。

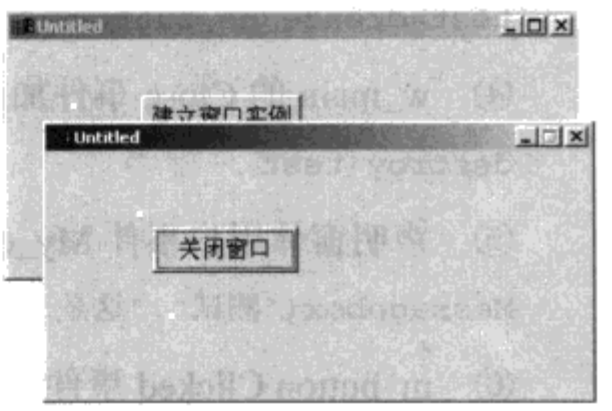


图 5-13 多窗口实例执行结果

5.5.10 在实例窗口中引用控件

可以通过对象类型的变量(如 Windows)引用对象实例中的控件。

示例 1：假设有 w_employee 窗口对象。

```
w_employee myemp
Open(myemp)
Myemp.st_name.text = "张峰"
```

这里 myemp 是 w_employee 类型的变量，而 w_employee 有静态文本控件 st_name，所以以上程序段的最后一条语句是合法的。

示例 2：

```
window myemp
string winname = "w_employee"
Open(myemp,winname)
Myemp.st_name.text = "张峰"
```

这里的 myemp 是预定义的 Window 类型的变量，它是所有窗口对象的祖先，但是它不包含静态文本控件 st_name，所以以上程序段的最后一条语句是非法的。

如果将示例 2 的程序改为：

```
w_employee myemp  //myemp 的数据类型为 w_employee
```

```
string winname = "w_employee"
Open(myemp,winname)
Myemp.st_name.text = "张峰"
```

这时最后一条语句是合法的，因为 myemp 的数据类型为 w_employee，而它含有控件 st_name。

另外还可以将示例 2 的程序改为：

```
window myemp
w_employee empwin  //empwin 的数据类型为 w_employee
string winname = "w_employee"
Open(myemp,winname)
empwin = myemp
empwin.st_name.text = "张峰"
```

这里最后一条语句也是合法的，因为 empwin 的数据类型为 w_employee，而它含有控件 st_name。

注意：　PowerBuilder 采取的是“强类型检测”的编译策略，任何类型不匹配的赋值或引用都不被识别。

5.5.11　向窗口传递多个参数

用 OpenWithParm 打开窗口并向窗口传递参数时，只允许传递一个参数，例如：

```
OpenWithParm(w_abc, "hello")
```

当传递多个参数时可以创建一个结构，将需要传递的参数都定义到这个结构中去。例如，首先定义 str_parms 结构，然后在 script 中声明结构变量：

```
str_parms l_str_parms
```

给结构变量赋值：

```
l_str_parms.name = "Smith"
l_str_parms.tel = "1234567"
OpenWithParm(w_abc, l_str_parms)
```

在窗口 w_abc 的 Open 事件中输入代码如下：

```
str_parms l_str_parms
l_str_parms = Message.PowerObjectParm
```

这个方法同样适用于 CloseWithReturn。

5.5.12　设计实时帮助条

许多菜单管理器都提供了实时帮助条，用它可以增强系统的易用性，在 PowerBuilder 中也可以实现类似的功能。当鼠标指针移动到窗口中的某些控件(如编辑器、图片等)时，会在鼠标指针附近产生帮助条，实时地提示操作要领。

首先在窗口 w_main 中任意位置定义一个黄底黑字的静态文本 st_help，设定 st_help.visible=false，st_help.text=&Help；然后在该窗口模块的 Declare\WindowFunctions... 下定义 show_help()函数，只有一个参数，参数名为 text，类型为 String，通过传值方式接收参数，无返值。show_help()代码如下：

```
if st_help.visible then return
st_help.text = text
st_help.width = Len(st_help.text) * 38
st_help.x = w_main.PointerX()
st_help.y = w_main.PointerY() + 50
if st_help.x + st_help.width > w_main.Workspacewidth() then
st_help.x = w_main.Workspacewidth()- st_help.width
end if
if st_help.y + st_help.height >w_main.Workspaceheight() then
st_help.y = w_main.Workspaceheight()- st_help.height
end if
st_help.visible= true
end if
```

接下来，就可以调用 show_help()函数了。但 PowerBuilder 提供的所有控制均缺乏当鼠标指针移至其上就触发的事件，需要定义相应的用户事件。

选中准备定义用户事件的控件，如某个单行编辑器，然后在窗口模块的 Declare\UserEvents...菜单下，双击 PasteEventID：中的 pbm_mousemove 条目，将其复制至 EventID 下，取 EventName 为 Mouseon，这样，就定义了相应控制的用户事件 Mouseon。

在该控件的用户事件 Mouseon 中，写入调用函数 show_help()的语句：

```
if st_help.visible then Hide(st_help)
show_help("瞧！这便是实时帮助条！")
```

5.5.13 实现“跑马灯”

有时需要用一个矩形条来显示一些重要信息，这些信息串通常首尾相连，向同一个方向循环滚动，通常将其称作“跑马灯”。证券业中常用“跑马灯”来显示不断变化的股票行情；实际应用中也常通过“跑马灯”来监视是否死机。

用一个简单的函数 running_horse()来实现“跑马灯”的显示。该函数有两个参数，第一个参数为 textline，类型为 string，传值；第二个参数为 num，类型为 Int，传值；函数返值类型为 String。该函数的代码仅有一句：

```
returnMid(textline,(num+1))+Left(textline,num)
```

下面就可以调用 running_horse()函数了。在一个窗口里定义好单行编辑器 sle_running_horse，在该窗口的 Open 事件下添加以下代码：

```
sle_running_horse.text = "I am testing running_horse!"
Timer(0.2)
```

然后在该窗口的 Timer 事件下调用 running_horse()函数，代码如下：

```
sle_running_horse.text = running_horse(sle_running_horse.text,1)
```

这样，当打开这个窗口时，“跑马灯”便会运转起来。可以在程序中加些语句，适时地增减 sle_running_horse.text 中的内容，便能在“跑马灯”中看到相应的变化信息。

5.5 习　　题

1. 填空题

(1) PowerBuilder 10.0 中提供了 6 种类型的窗口，分别是：_____、_____、_____、_____、_____、_____。

(2) 窗体主要由 4 部分组成，包括：_____、_____、_____、_____。

(3) 窗体画板工作窗体由 4 部分构成_____、_____、_____、_____。

(4) 利用创建_____，能够大大减少开发和维护系统的工作量，同时也便于形成标准一致的界面。

2. 问答题

(1) 如何创建继承窗体？

(2) 如何保存、预览和打印窗体？

(3) 如何向窗口传递多个参数？

第 6 章　常用控件技术

控件是构成应用程序用户界面、完成数据输入/输出的强有力工具。利用控件可以完成许多界面设计任务而无需编写代码；还可以迅速构造出操作方便、使用灵活、安全稳健及界面友好的应用程序。本章主要介绍在窗体设计中经常使用的控件及其相关知识。

6.1　按 钮 控 件

按钮控件是大多数用户界面不可缺少的要素之一，通过单击按钮，可以完成一个或一系列操作。按钮有以下两种形式。

- 命令按钮(CommandButton)。
- 图形按钮(PictureButton)。

6.1.1　命令按钮

命令按钮(CommandButton)是标准的 Windows 按钮，只显示提示按钮功能的文字。它总以三维状态显示，不具备 Border(边框)属性，也不能修改按钮的标题字符颜色和背景颜色。

1．属性

命令按钮有一组属性，选中命令按钮后，单击窗口画板工具栏的 Properties 图标，打开属性设置对话框，如图 6-1 所示。

该对话框由 3 个选项卡组成。

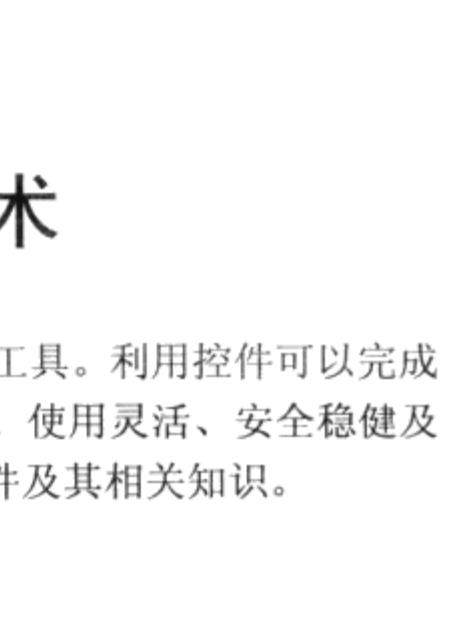


图 6-1　按钮的属性对话框

(1) General 选项卡：用于定义命令按钮的一般属性，包括名称、标题、是否显示和是否可用等。在 General 选项卡中各选项的意义如下。

- Name 编辑框用于指定按钮的变量名。
- Text 编辑框用于指定显示在按钮上的文字(按钮标题)。
- Visible 复选框用于控制按钮是否显示在窗口上，选中该复选框，表示显示；否则隐藏。
- Enabled 复选框用于控制按钮是否可用，选中该复选框，表示能够操作该按钮，否则不能操作。
- Default 复选框用于设置当前窗口的默认按钮，当程序运行时，即使输入焦点位于其他控件上，按下 Enter 键将触发该按钮的 Clicked 事件。
- Cancel 复选框用于设置当前窗口的取消按钮，按下 Esc 键将触发该按钮的 Clicked 事件。

(2) Font 选项卡：用于定义命令按钮标题的显示字体。

(3) Other 选项卡：用于定义命令按钮的放置位置、宽度和高度；定义当鼠标指针位于

该命令按钮上时鼠标指针的形状；定义当拖动该命令按钮时鼠标指针的形状。

2. 事件

命令按钮有十几个事件，其中最常用、最重要的事件是 Clicked 事件，它在被单击时触发。在这个事件的处理程序中编写按钮被按下时要执行的代码。

6.1.2　图形按钮

图形按钮(PictureButton)与命令按钮基本相同，只是增加了额外功能，它除了具备文本标题外，还可以指定显示在按钮上的图像。它实现功能与命令按钮类似，区别在于可以在该按钮上显示 BMP、RLE 或 WMF 格式的图像，而且能够以不同图像表示按钮处于允许和不允许两种状态。

为图形按钮指定图片的步骤如下：

(1) 把图形按钮放置到窗口上后，双击该按钮。

(2) 显示属性对话框，选择 General 选项卡，如图 6-2 所示。

(3) 单击 PictureName 属性框后的…按钮，定位图形按钮上要显示的图像文件。

(4) 在 HTextAlign 下拉列表框中选择图形按钮标题的水平对齐方式。

- Left：左对齐。
- Right：右对齐。
- Center：居中对齐。

(5) 在 VTextAlign 下拉列表框中选择图形按钮标题的垂直对齐方式。

- Bottom：底部对齐。
- Multiline：分成多行。
- Top：顶部对齐。
- Center：居中对齐。

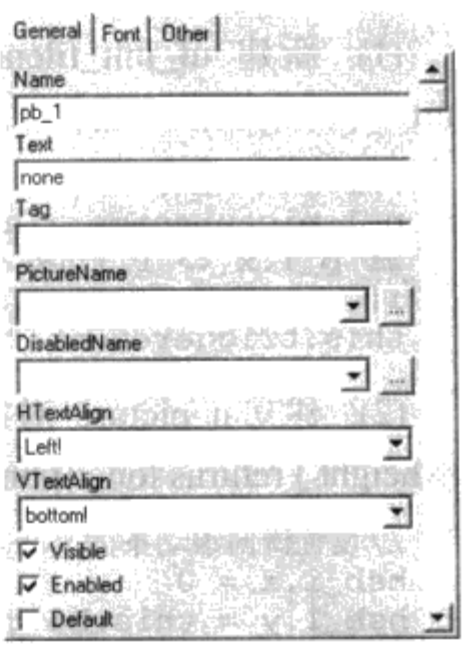


图 6-2　图形按钮 General 选项卡

注意： 在同一个窗口中，不要把多个按钮(包括命令按钮和图形按钮)指定为默认按钮或取消按钮。虽然在设计时可以选中所有按钮的 Default 和 Cancel 复选框，但只有一个默认按钮和取消按钮起作用，就是最后设置的那个按钮。

6.1.3　设计带滚动条的图形控件

本节将制作一个带滚动条的图形控件。其功能为当图片大小超过图片控件大小时，自动显示滚动条，自动调整对象内的图片控件的大小及滚动条的滚动范围。本例演示了在 PowerBuilder 中利用定制可视用户对象制作控件的方法。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

2. 建立定制可视用户对象

(1) 新建定制可视用户对象。

(2) 添加对象控件。在新建的用户对象中添加一个 Picture 控件、一个 HScrollBar 控件、一个 VScrollBar 控件和一个 StaticEdit 控件。各控件名称分别为 p_1、hsb_1、vsb_1 和 st_1，控件大小与位置如图 6-3 所示。

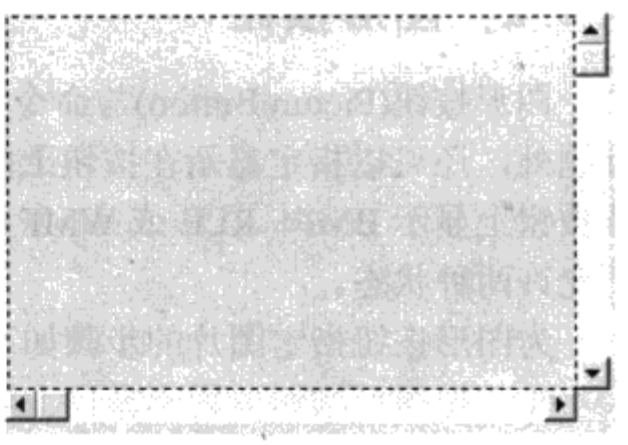


图 6-3 可视用户对象基本框架

(3) 将可视用户对象保存为 u_picture。

3. 编写可视用户对象驱动代码

(1) 新建 uf_init_filename (string as_filename) returns (none)函数，输入以下代码：

```
//设置图片名称
p_1.picturename = as_filename
if p_1.x <> 0 then p_1.x = 0
if p_1.y <> 0 then p_1.y = 0
this.triggerevent("ue_resize")
```

(2) 建立 u_picture 对象的 ue_resize (unsignedlong sizetype，integer newwidth，integer newheight) returns long [pbm_size]事件，输入以下代码：

```
//设置横向滚动条的 x、y 坐标
hsb_1.x = 0
hsb_1.y = this.height - hsb_1.height
//设置横向滚动条的长度
hsb_1.width = this.width - vsb_1.width
//设置纵向滚动条的 x、y 坐标
vsb_1.x = this.width - vsb_1.width
vsb_1.y = 0
//设置纵向滚动条的高度
vsb_1.height = this.height - hsb_1.height
//设置横向滚动条的最大滚动值
hsb_1.MaxPosition = p_1.width - hsb_1.width
if hsb_1.MaxPosition < 1 then hsb_1.MaxPosition = 0
//设置纵向滚动条的最大滚动值
vsb_1.MaxPosition = p_1.height - vsb_1.height
if vsb_1.maxposition < 1 then vsb_1.maxposition = 0
//设置横向和纵向滚动条的初始位置
hsb_1.position = 0
vsb_1.position =0
p_1.x = 0
p_1.y = 0
st_1.x = hsb_1.width + 1
st_1.y = vsb_1.height + 1
//如果图片的横向可以显示完整，就将横向滚动条隐藏
if hsb_1.width > p_1.width then
    hsb_1.visible = false
else
    hsb_1.visible = true
```

```
end if
//如果图片的纵向可以显示完整，就将纵向滚动条隐藏
if vsb_1.height > p_1.height then
    vsb_1.visible = false
else
    vsb_1.visible = true
end if
//因为纵向滚动条被隐藏，所以，调整横向滚动条的长度
if vsb_1.visible = false and hsb_1.visible <> false then
    hsb_1.width = this.width
end if
//因为横向滚动条被隐藏，所以，调整纵向滚动条的高度
if hsb_1.visible = false and vsb_1.visible <> false then
    vsb_1.height = this.height
end if
```

(3)建立 hsb_1 的 lineleft () returns long [pbm_sbnlineup]事件，输入以下代码：

```
this.position = this.position - 5
if this.position < this.minposition then this.position = 0
//this.minposition
p_1.x= 5 - this.position
```

(4) 建立 hsb_1 的 lineright () returns long [pbm_sbnlinedown]事件，输入以下代码：

```
this.position = this.position + 5
if this.position > this.maxposition then this.position = this.maxposition
p_1.x= 5 - this.position
```

(5) 建立 hsb_1 的 moved (integer scrollpos) returns long [pbm_sbnthumbtrack]事件，输入以下代码：

```
p_1.x= 5 - this.position
```

(6) 建立 hsb_1 的 pageleft ( ) returns long [pbm_sbnlineup]事件，输入以下代码：

```
this.position = this.position - 50
if this.position < this.minposition then this.position =0   //
this.minposition
p_1.x= 5 - this.position
```

(7)建立 hsb_1 的 pageright () returns long [pbm_sbnlinedown]事件，输入以下代码：

```
this.position = this.position + 50
if this.position > this.maxposition then this.position = this.maxposition
p_1.x= 5 - this.position
```

(8) 建立 vsb_1 的 lineup () returns long [pbm_sbnlineup]事件，输入以下代码：

```
this.position = this.position - 5
if this.position < this.minposition then this.position = this.minposition
p_1.y= 5 - this.position
```

(9) 建立 vsb_1 的 linedown () returns long [pbm_sbnlinedown]事件，输入以下代码：

```
this.position = this.position + 5
if this.position > this.maxposition then this.position = this.maxposition
p_1.y= 5 - this.position
```

(10) 建立 vsb_1 的 moved (integer scrollpos) returns long [pbm_sbnthumbtrack]事件，输入以下代码：

```
p_1.x= 5 - this.position
```

(11) 建立 hsb_1 的 pageup () returns long [pbm_sbnlineup]事件，输入以下代码：

```
this.position = this.position - 50
if this.position < this.minposition then this.position = this.minposition
p_1.y= 5 - this.position
```

(12) 建立 hsb_1 的 pagedown () returns long [pbm_sbnlinedown]事件，输入以下代码：

```
this.position = this.position + 50
if this.position > this.maxposition then this.position = this.maxposition
p_1.y= 5 - this.position
```

4. 建立窗口

(1) 新建 w_picture 窗口，Title 设置为“带滚动条的图形控件”。

(2) 建立控件。向窗口中添加一个 u_picture 控件和一个 CommandButton 控件。分别命名为 uo_1 和 cb_exit。

5. 编写事件驱动代码

(1) 在 w_main 窗口的 Open 事件中输入以下代码：

```
uo_1.uf_init_filename("pic.jpg")
```

(2) 在 cb_exit 的 Clicked 事件中输入以下代码：

```
close(parent)
```

(3) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入以下代码：

```
Open(w_main)
```

单击 PowerBuilder 10.0 开发界面上的 按钮或按 Ctrl+R 快捷键，程序就可以运行，如图 6-4 所示。

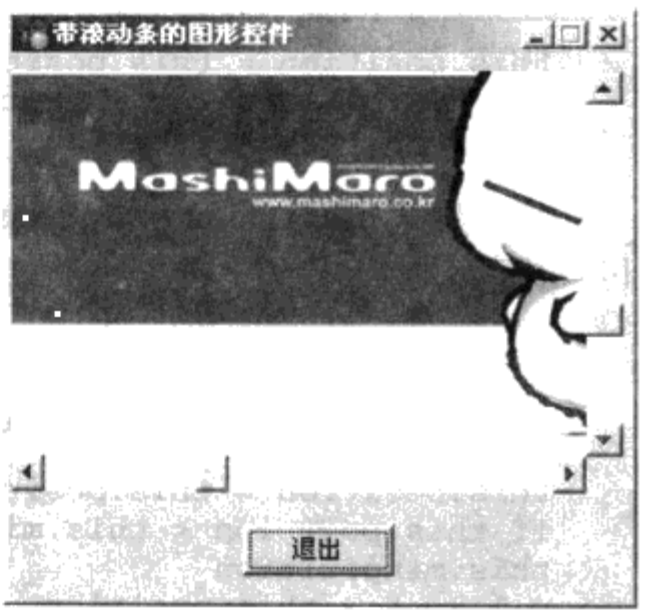


图 6-4 运行效果图

6.2 文本显示与编辑控件

为了满足不同的需求，PowerBuilder 10.0 还提供了多种用于显示、输入或编辑文字的控件。静态文本控件用于显示不需要用户更改的信息，单行编辑框用于输入一行文字，多行编辑框用于输入和编辑长篇文章，掩码编辑框自动格式化用户的输入，而超文本编辑框本身就是个功能强大的超文本编辑器。通过灵活地运用这些控件，不仅可以丰富程序界面，还可以提供更强、更完整的功能。

6.2.1 静态文本框

静态文本框(StaticText)常用于显示提示信息或说明没有标题控件的作用等。例如在单

行编辑框前面加个静态文本框可以说明在单行编辑框中输入何种类型的数据。当很多情况下，也把静态文本控件显示的文本称作标签。

注意： 不要与后续章节介绍的标签(Tab)控件混淆。

1. 属性

静态文本控件是 PowerBuilder 众多对象之一，该属性设置对话框如图 6-5 所示。

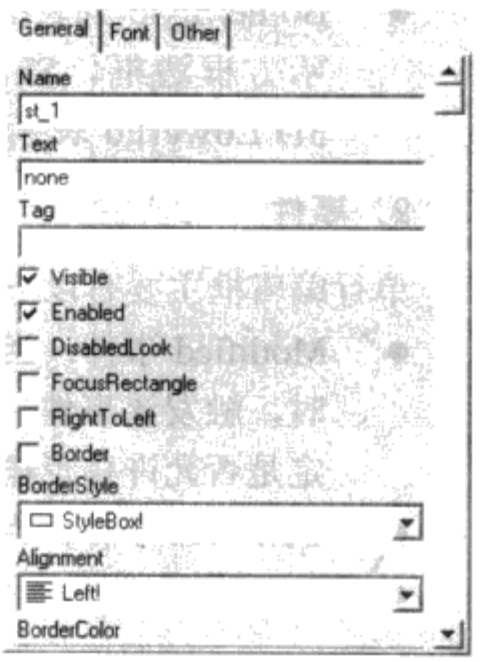


图 6-5　静态文本控件属性对话框

其中各选项意义如下：

- Text 文本框用于设置静态文本控件上显示的文本。
- BorderStyle 下拉列表框用于选择该控件的边框。
- Alignment 下拉列表框用于指定显示文本的对齐方式。
- BorderColor 下拉列表框用于选择边框的颜色。
- FillPattern 下拉列表框用于指定填充图案。

2. 事件

静态文本控件有一组事件，例如 Clicked、DoubleClicked 等，但通常极少对其编程。

技巧： 静态文本控件中的静态二字是对用户而言的，程序中通过修改它的 Text 属性也可改变静态文本控件的显示内容。

6.2.2　单行编辑框

单行编辑框(SingleLineEdit)通常用于输入一行文本、输入用户密码等。

1. 属性

单行编辑框的属性对话框如图 6-6 所示，其中各选项的意义如下。

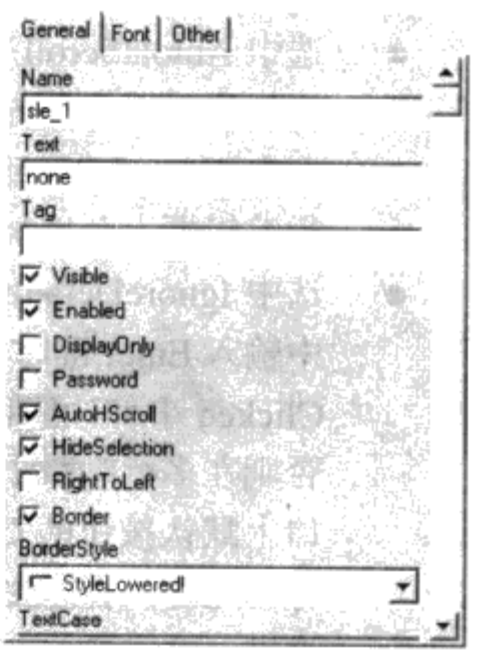


图 6-6　单行编辑框的属性对话框

- Text 编辑框保存输入的内容以及当前显示的文本。
- 选中 Password 复选框，单行编辑框中显示星号(*)而不是输入的实际字符，实际字符保存在 Text 属性中，该功能常用于设计保密字的输入。
- 选中 AutoHScroll 复选框，允许输入超过单行编辑框实际长度的文本。当到达单行编辑框的末尾时，随着输入的继续，文本自动向左滚动。
- 选中 DisplayOnly 复选框，将禁止用户输入。
- TextCase 列表框指定是否强制性地将输入的字母进行大小写转换，Any 表示不作转换，Lower 表示全部转换成小写，Upper 表示全部转换成大写。

- Limit 列表框指定输入的最大字符个数，有效值在 0～32 767 之间，其中 0 表示不限制输入字符的个数。
- Accelerator 文本框中可以输入一个字符(比如 m)，用于指定单行编辑框的访问键，这样通过按 Alt+字符(比如 Alt+m)键就能将输入焦点转到该控件。
- BorderStyle 列表框指定单行编辑框的边框风格，其中 None 表示无边框，Box 表示方框边框，ShadowBox 表示阴影方框边框，3D Raised 表示三维上凸，3D Lowered 表示三维下凹。

2. 事件

单行编辑框主要有以下 3 个事件。

- Modified 事件：当修改了单行编辑框显示的文本并用 Tab 键或鼠标移走输入焦点时，触发该事件。可以在这个事件的处理程序中检查用户输入是否有效，从而决定是否允许移走输入焦点。
- GetFocus 事件：当用 Tab 键或鼠标把输入焦点移动到该控件时，触发该事件。
- LoseFocus 事件：当移走输入焦点时触发该事件。

技巧： 设置了单行编辑框的访问键后，通常都要在与之相应的静态文本控件的标签上予以体现。比如访问键是 A，标签可以写成姓名(&A)，因为标签的 Enabled 属性为 False，所以按下 Alt+A 键后，输入焦点就转移到该单行编辑框。

6.2.3 多行编辑框

多行编辑框(MultiLineEdit)常用于输入大块的文字，比如个人简历、长篇文章等。

1. 属性

多行编辑框的属性对话框如图 6-7 所示。

与单行编辑框同名的属性和功能在此就不赘述了。下面介绍多行编辑框特有的属性。

- 选中 AutoVScroll 复选框将自动下滚文本。
- 选中 AutoHScroll 复选框后，随着输入或删除字符，多行编辑框中的内容自动向左或向右滚动。
- 选中 HScrollBar 或 VScrollBar 复选框后，将在多行编辑框上分别加上水平或垂直滚动条。
- 选中 IgnoreDefaultButton 复选框后，在多行编辑框中输入 Enter 键时不会触发当前窗口上默认按钮的 Clicked 事件，而是在多行编辑框中生成一个新行。否则在多行编辑框中输入 Enter 键时将触发当前窗口上默认按钮的 Clicked 事件，而不是生成一个新行，此时，若想生成新行时，需要按 Ctrl+Enter 组合键。
- Alignment 列表框指定多行编辑框中文字的对齐方

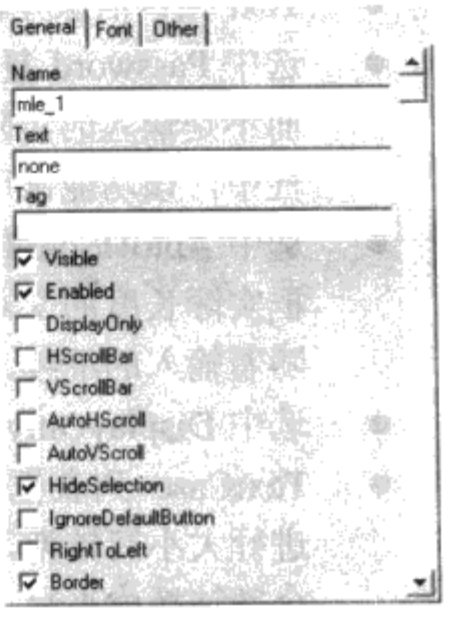


图 6-7 多行编辑框的属性对话框

式。Left 表示左对齐，Right 表示右对齐，Center 表示居中对齐。

- Tab Stop 列表框定义按 Tab 键时跳转多少个字符，默认值为 8。

2. 事件

多行编辑框使用的常用事件有 Modified、GetFocus 和 LoseFocus，其触发时机和用法与单行编辑框相同。

技巧： 在单行编辑框或多行编辑框中输入制表符(Tab 字符)，可用符号~t 来表示，直接按 Tab 键并不能在编辑框中输入制表符，而是移动了输入焦点。

6.2.4　掩码编辑框

掩码编辑框(EditMask)能够创建一个掩码，使用户必须按指定格式输入字符，即只接受指定格式的字符，并且它能够完成用户输入的自动格式化。

1. 属性

掩码编辑框的属性对话框的 General 选项卡与单行编辑框相同，各属性的含义也一样，在此不再累述。单击 Mask 选项卡，显示如图 6-8 所示的对话框。

各选项的意义如下：

- 在 MasksDataType 列表框中指定要输入数据的类型，选择不同类型后。Mask 列表框中的掩码也随之改变。
- Mask 编辑框定义所需的用户输入格式。通过单击 Masks 编辑框右侧箭头选择格式。例如输入掩码 ##-#!!#，当输入 345ab6 后，该编辑框自动将用户输入格式化为：34-5AB6。
- 选中 Spin 复选框后，其他几个选项才能使用。这时掩码编辑框的右边界上出现两个滚动小按钮，单击该按钮可调整掩码编辑框的值。
- 当选中 Code Table 复选框时，以列表形式调整，否则以数值形式调整。
- 对特殊按键，掩码编辑框可实现以下功能：
  按退格键(Backspace 或←键)总是删除前一个字符。
  按删除键(Delete 或 Del 键)删除所有选中的字符。

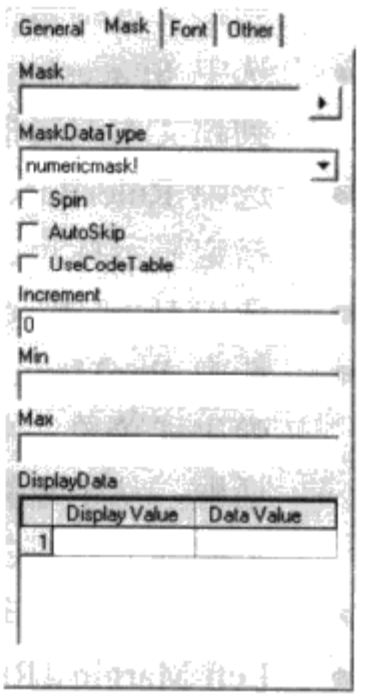


图 6-8　掩码编辑框的属性对话框

2. 事件

掩码编辑框的常用事件为 Modified、GetFocus 和 LoseFocus，其触发时机和用法与单行编辑框相同。

6.2.5　超文本编辑框

超文本编辑框(RichTextEdit)是个功能十分强大的文本编辑器，它可以使用 Windows 系

统支持的所有字体、字型及颜色，它有工具栏和标尺。利用该控件，能够输入、编辑、格式化以及保存与打印文本。

1. 属性

超文本编辑框的 General 选项卡比较简单，其中各选项的意义如下：

- HScrollBar 与 VScrollBar 复选框指定是否在超文本编辑框上添加水平与垂直滚动条。
- Backgroud Color 用于选择超文本编辑框的背景颜色。
- Border 下拉列表框用于选择超文本编辑框的边框风格，其中，None 表示无边框，Box 表示方框边框，ShadowBox 表示阴影方框边框，3DRaised 表示三维上凸，3D Lowered 表示三维下凹，Resize 表示运行时可改变超文本编辑框的大小。

单击 Document 选项卡，显示如图 6-9 所示的对话框，这组属性用于控制超文本编辑框的外观和功能。

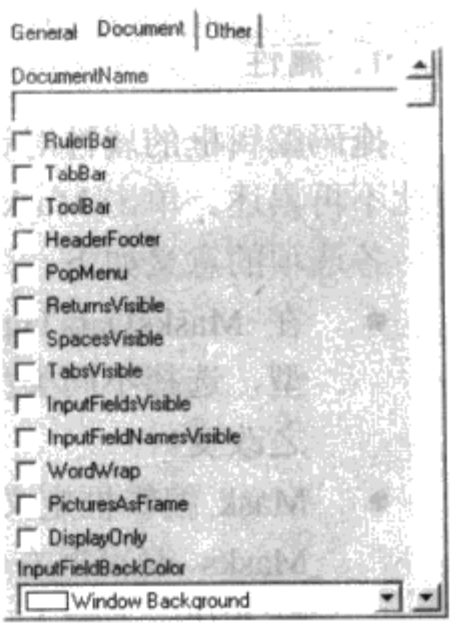


图 6-9 Document 选项卡

各选项的意义如下：

- 选中 TabsVisible 复选框时，以字符“□”显示输入到超文本编辑框中的字符。
- 选中 ReturnsVisible 复选框时，以字符“¶”显示输入到超文本编辑框中的换行字符。
- 选中 SpacesVisible 复选框时，以字符“.”显示输入到超文本编辑框中的空格字符。
- 选中 RulerBar、TabBar、ToolBar 复选框时分别用于指定显示标尺、Tab 条和工具栏。
- 选中 HeaderFooter 复选框用于指定是否加题头/脚注。
- 选中 PopMenu 复选框后，右击可以打开用于编辑的弹出式菜单。单击 Properties...菜单项，打开一个对话框，从中设置所选文本的属性，例如字体、字型等；InsertFile 用于向超文本框中插入文件；Cut、Copy、Paste 或 Clear 分别用于剪切、复制、粘贴或清除文本。
- Undo Depth 编辑框用于指定取消最近操作的最大深度。
- Left Margin、Right Margin、TopMargin 和 BottomMargin 编辑框分别用于指定打印页的左空白、右空白、上空白和下空白的大小。

2. 事件

超文本编辑框的事件较多，下面介绍几个常用的、特有的事件。

- FileExists：当把超文本编辑框的内容保存到文件并且指定的文件已经存在时，触发该事件。相应的文件名作为事件的参数被传递。
- Modified：在第一次修改超文本编辑框的内容时被触发。
- PictureSelected：双击超文本编辑框中的图片或在图片上按下 Enter 键时触发该事件。
- PrintFooter：打印当前页的脚注时被触发，事件参数包括当前页的页号、总页数以及与之关联的数据窗口的当前行的行号。

- PrintHeader：打印当前页的题头时被触发，事件参数包括当前页的页号、总页数以及与之关联的数据窗口的当前行的行号。
- RButtonDown：当在超文本编辑框中右击且该控件的 PopMenu 属性设置为 False 时触发。

技巧： 需要应用多行标签时，可使用多行文本框，此时需要选中属性对话框中的 Display Only 复选框，目的是避免用户修改，当然，也能够给它加上滚动条。另外，如果需要多种精美字体与优雅颜色显示大量提示信息，那么超文本框是个更好的选择。

6.2.6　设计记事本程序

本例将利用 PowerBuilder 10.0 制作一个记事本程序。运行程序后将弹出一个“记事本”窗口，在该窗口中可以进行跟普通记事本一样的写入和存储功能。同时也可以进行文本格式的设置。本例较为综合地运用了 PowerBuilder 10.0 中 MDI 窗口的技巧。

1. 创建程序的基本框架

(1) 建立 work 工作区。
(2) 建立 app 应用。
(3) 建立 w_file 窗口，Title 设置为“记事本”。
(4) 建立 w_font 窗口，Title 设置为“字体”。
(5) 建立 Menu 菜单，保存为 m_main。

2. 设置 w_file 窗口

(1) 建立控件。在窗口中添加两个 MultiLineEdit 控件，分别为 mle_edit 和 mle_time。钩选 mle_edit 控件的 VScrollBar 属性并将两个控件 Position 属性中的 X 和 Y 都设置为 0。

(2) 创建全局变量

```
string current_file_name
```

(3) 创建全局扩展函数

```
Function string CreateFile(char FileName) LIBRARY "user32.dll"
```

(4) 在窗口的 resize 事件下输入以下代码：

```
mle_edit.width  = this.workspacewidth()
mle_edit.height = this.workspaceheight()
```

3. 设置 w_font 窗口

(1) 建立控件。在窗口中添加 3 个 StaticEdit 控件、3 个 SingleLineEdit 控件、两个 ListBox 控件和两个 CommandButton 控件。各控件名次分别为：st_1、st_2、st_size、sle_font、sle_style、sle_size、lb_font、lb_style、cb_OK 按钮和 cb_cancel。

(2) 设置各控件属性。如图 6-10 设置 lb_font 的 Items 属性；如图 6-11 设置 lb_style 的 Items 属性；其余控件的大小与位置如图 6-12 所示。

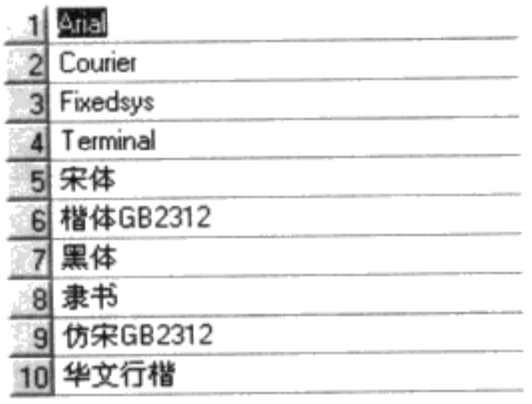


图 6-10　lb_font 的 Items 属性

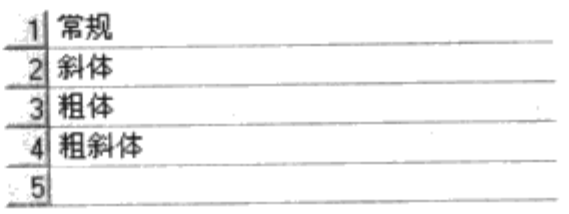


图 6-11　lb_style 的 Items 属性

(3) 在 cb_cancel 控件的 clicked 事件中输入以下代码：

```
close(w_font)
```

(4) 在 cb_OK 按钮控件的 clicked 事件中输入以下代码：

```
if sle_font.text <> "" then
w_file.mle_edit.facename = sle_font.text
//字形
if sle_size.text <> "" then
w_file.mle_edit.textsize =
integer(sle_size.text) //大小
if sle_style.text <> "" then    //风格
...
end if
close(w_font)
```

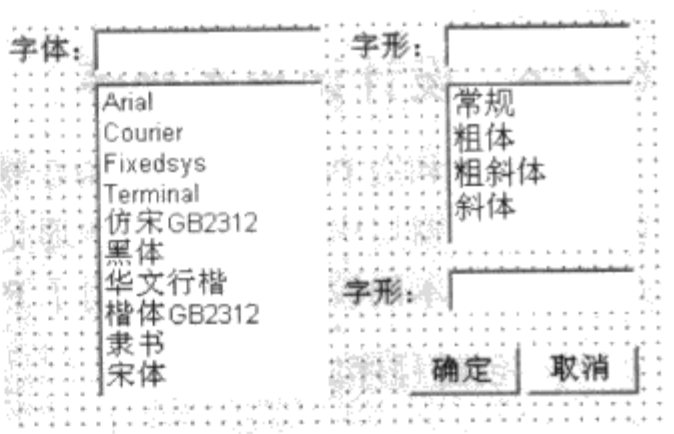


图 6-12　w_font 窗口基本框架

(5) 在 lb_font 控件的 selectionchanged 事件中输入以下代码：

```
int i , j
i = lb_font.selectedindex()
sle_font.text = lb_font.selecteditem()
//sle_font.text = lb_font.text(i)
j = lb_font.setstate(i,false)
```

(6) 在 lb_style 控件的 selectionchanged 事件中输入以下代码：

```
int i , j
i = lb_style.selectedindex()
sle_style.text = lb_style.selecteditem()
//sle_font.text = lb_font.text(i)
j = lb_style.setstate(i,false)
```

4. 设置 m_main 菜单

(1) 建立 m_main 菜单基本框架，如图 6-13 所示。

(2) 在“新建”菜单命令下输入以下代码：

```
//清空编辑区内容
w_file.mle_edit.text = ""
//新建文件的统一名称为 Untitle
current_file_name = "无标题"
//修改窗口标题
```

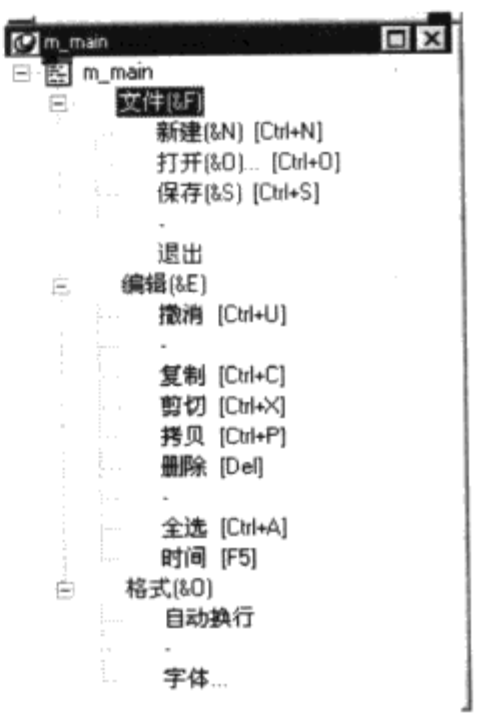


图 6-13　m_main 菜单的基本框架

```
w_file.title = current_file_name+" - 记事本"
```

(3) 在“打开”菜单命令下输入以下代码：

```
// string docname,named
integer value,rc,fd
long ll_Flength
//选择要打开的文件
value = GetFileOpenName("选择要打开的文件",docname,named,"TXT","文本文件
(*.txt),*.txt,所有文件(*.*),*.*")
if value = 1 then
    //获取文件长度
    ll_Flength = FileLength(docname)
    //打开文件
    fd = FileOpen(docname,StreamMode!)
    if ll_Flength < 32767 then
        //读取文件内容到多行编辑框中
        rc = FileRead(fd,w_file.mle_edit.text)
        FileClose(fd)
        //保存文件名，用于将来保存文件
        current_file_name = docname
        //修改窗口标题
        w_file.title = "记事本__文件" + current_file_name
    end if
end if
```

(4) 在“保存”菜单命令下输入以下代码：

```
string docname,named
integer value,rc,fd,ret
long ll_Flength
boolean file_exist
value = GetFileSaveName("保存为",docname,named,"TXT","文本文件(*.txt),*.txt,
所有文件(*.*),*.*")
if value = 1 then
...
end if
fd = FileOpen(current_file_name,StreamMode!,Write!,LockWrite!,Replace!)
//将多行编辑框 mle_edit 中的内容写入文件
FileWrite(fd,w_file.mle_edit.text)
FileClose(fd)
w_file.title = current_file_name+" - 记事本"
```

(5) 在“退出”菜单命令下输入以下代码：

```
close(w_file)
```

(6) 在“撤消”菜单命令下输入以下代码：

```
w_file.mle_edit.undo()
```

(7) 在“复制”菜单命令下输入以下代码：

```
w_file.mle_edit.copy()
```

(8) 在“剪切”菜单命令下输入以下代码：

```
w_file.mle_edit.cut()
```

(9) 在“复制”菜单命令下输入以下代码：

```
w_file.mle_edit.paste()
```

(10) 在“删除”菜单命令下输入以下代码：

```
w_file.mle_edit.clear()
```

(11) 在“全选”菜单命令下输入以下代码：

```
w_file.mle_edit.setfocus()
w_file.mle_edit.selecttext(1,888888)
```

(12) 在“时间”菜单命令下输入以下代码：

```
w_file.mle_time.text = string(now()) + " " + string(today())
w_file.mle_time.setfocus()
w_file.mle_time.selecttext(1,20)
w_file.mle_time.copy()
w_file.mle_edit.setfocus()
w_file.mle_edit.paste()
```

(13) 在“自动换行”菜单命令下输入以下代码：

```
if m_main.m_option.m_retuen.checked = true then
    m_main.m_option.m_retuen.checked = false
    w_file.mle_edit.hScrollBar = true
else
    m_main.m_option.m_retuen.checked = true
    w_file.mle_edit.hScrollBar = false
end if
```

(14) 在“字体”菜单命令下输入以下代码：

```
open(w_font)
```

(15) 在 w_file 窗口中，将 MenuName 设置为 m_main。

(16) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入以下代码：

```
Open(w_file)
```

单击 PowerBuilder 10.0 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行，运行效果如图 6-14 所示。

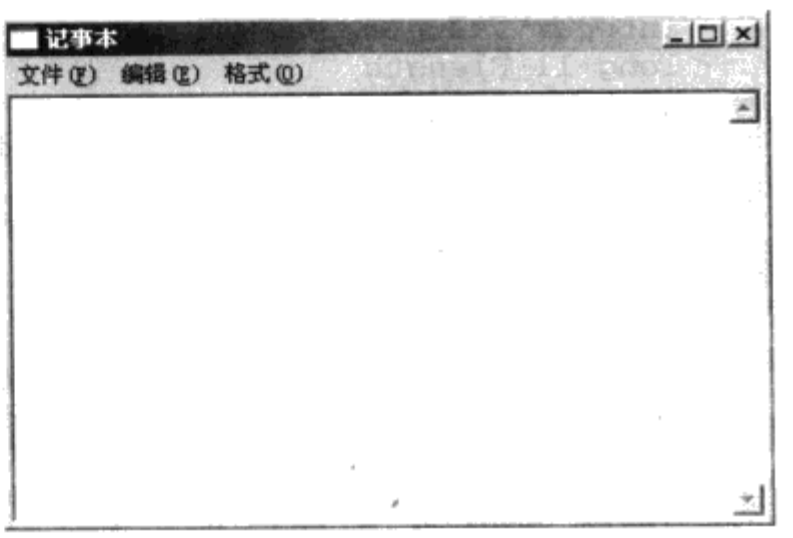


图 6-14 运行效果图

6.3 选项控件

使用图形界面控件的目的是为了方便用户的操作，且直观明了。单选按钮、复选框正是为了方便用户做出选择而设计的。单选按钮用来表示一组互斥的选项，只能从中选择一个。单选按钮带有一个圆形图形，当被选中时，其图形中心出现一个黑点；未被选中时，其图形中心为空白。复选框用来表示是/否或真/假两种状态，它通常被成组使用，同一组中

可有多个复选框被选中。当复选框被选中时，方框内出现一个(√)号；未被选中时，方框内为空白。组合框用来实现单选按钮、复选框的分组。

6.3.1　单选按钮

单选按钮(RadioButton)也叫选项按钮，它在窗口画板工具栏控件图标列表中的图标为◉。既可以把多个单选按钮放在一个窗口中；也可以用组合框把单选按钮分成不同的组，对于同组的单选按钮，当选中其中之一时，其余单选按钮自动解除选中状态。

1．属性

单选按钮的属性对话框如图 6-15 所示，各属性的意义如下：

- Automatic 属性用于指定当单击该单选按钮时，是系统自动地将其置为选中状态还是通过编写事件处理程序而将其置为选中状态。选中 Automatic 复选框时系统将自动处理，否则需要在单选按钮的 Clicked 事件中编写相应的事件处理程序。需要说明的是，当单选按钮处于组合框中时，Automatic 属性被忽略，系统自动调整选中单选按钮的选中状态。
- 在 Text 属性编辑框中输入的字符，将显示在单选按钮上的标题。
- LeftText 属性用于指定单选按钮的标题显示在圆形图形的左边还是右边。选中时表示左边；否则为右边。
- Checked 属性指定该单选按钮初始时是否被选中。
- Border 属性用于选择单选按钮的边框类型，其中，

  None 表示无边框，3D Raised 表示三维上凸，3D Lowered 表示三维下凹。

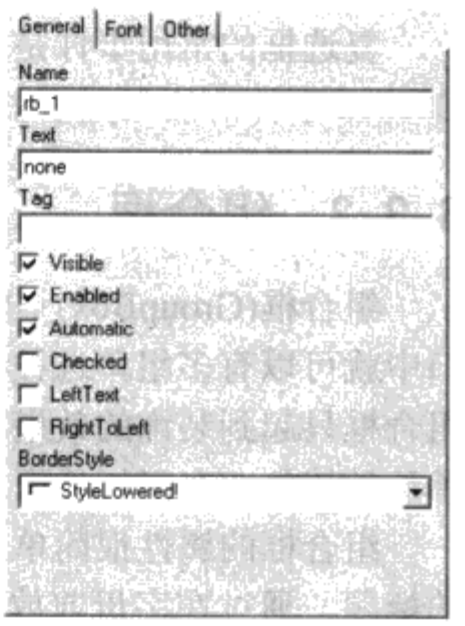


图 6-15　单选按钮属性对话框

单选按钮的文本字体、前景颜色和背景颜色等属性在字体(Font)选项卡中设置。

2．事件

单选按钮的常用事件是 Clicked，在单击单选按钮时触发。

6.3.2　复选框

复选框(CheckBox)用于设置独立的选项，多个复选框彼此相互独立，即使把它们放到一个组合框中，也不影响复选框的行为。

1．属性

复选框的属性对话框如图 6-16 所示。

其中大部分属性的含义与单选按钮相同，下面介绍复选框特有的属性。

- Automatic 属性用于指定是系统自动处理还是通过程序进行手工控制选中状态的设置。选中 Automatic 复选框时，系统自动处理；否则需要编写相应的事件处理程序。

- ThreeState 属性用于指定复选框是否使用除是/否两种状态之外的第三种选择，称为第三态。选中 ThreeState 复选框后，应用程序将在 3 种状态之间循环。复选框的第三态看上去像选中状态，但其方框的背景是灰色的。
- ThirdState 属性用于指定第三态是否作为初始状态，选中 ThirdState 复选框后，应用程序的初始状态为第三态。注意：ThirdState 与 Checked 属性互斥，两者只能选择其一。

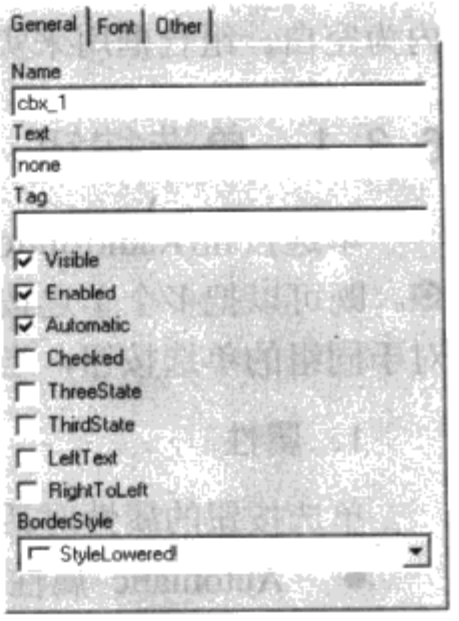


图 6-16　复选框属性对话框

2. 事件

复选框的常用事件是 Clicked，它响应用户的单击操作，在 Clicked 事件的处理程序中可以做一些验证与处理工作。

6.3.3　组合框

组合框(GroupBox)不仅可以使界面美化，还可以对单选按钮进行分组，这样在一个窗口中就可以有多组单选按钮了。组合框的分组作用只对单选按钮有效，而对于其他控件，组合框只起到装饰的效果，图 6-17 所示为应用组合框分组单选按钮的一个示例界面。

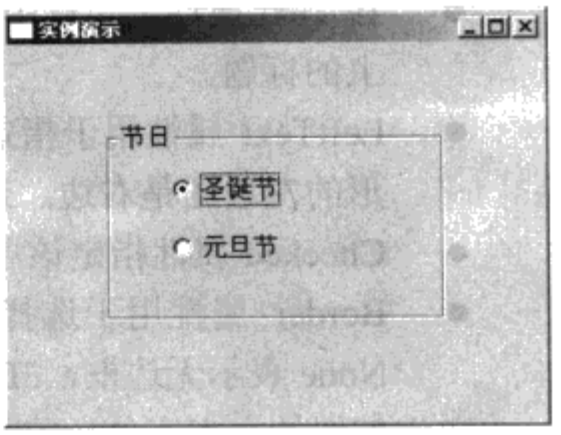


图 6-17　组框应用示例

组合框的属性很简单，Text 属性指定显示在组合框上的标题，通过在字母前放上&字符，可以为组合框指定访问键；组合框的其他属性与其他控件的相同属性功能类似，在此不再赘述。在组合框中放置单选按钮的步骤如下：

(1) 在窗口上放置一个组合框。

(2) 在窗口画板工具栏上单击单选按钮(RadioButton)图标。

(3) 移动鼠标指针到组合框中，然后单击。

(4) 当放置多个单选按钮时，按 Ctrl+T 键进行连续复制。

注意： 当单选按钮位于组合框中时，移动组合框同时也将移动组合框中的单选按钮，单选按钮已经成为组合框的一部分，但对其他控件却不然，它们并不随组合框的移动而移动。要想同时移动组合框及组合框中的其他控件，必须同时选中要移动的所有控件，然后通过拖动或键盘操作移动它们。

6.4　列表选择控件

当应用程序只需要少数几个选项时，使用单选按钮和/或复选框既简单又方便。然而用户的需求是多方面的，当选项个数较多并且每次运行都可能变化时，单选按钮、复选框就不能满足应用需求了，这时可以使用列表方式提供多重选择。PowerBuilder 提供了 4 个控

件用于完成列表选择，它们是：列表框、图片列表框、下拉列表框和下拉图片列表框。

列表框列出一系列选项供用户选择。

图片列表框在列出选项的同时，每个选项前还可以显示一幅小图片。

下拉列表框将单行编辑框与列表框结合成一个整体。

下拉图片列表框则又进一步，它不仅具有下拉列表框的功能，而且在每个选项前还可以放上一幅图片。

下面介绍各种列表框的功能和用法。

6.4.1　列表框

列表框(ListBox)在一个矩形框中显示多行文本，通过设置属性，程序可以允许同时选择一项或多项。在典型的 Windows 应用程序中，单击列表框中的一项则选中该项，双击列表框中的一项则可进行某种操作。

1．属性

列表框的属性对话框如图 6-18 所示。

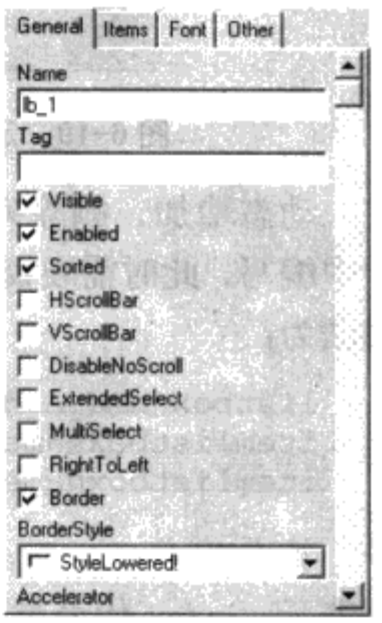


图 6-18　列表框属性对话框

各选项的意义如下。

- Visible 复选框用于指定列表框初始化时是否可见。
- Enabled 复选框用于指定列表框初始化时是否能用。
- HScrollBar 和 VScrollBar 复选框分别用于指定是否显示水平和垂直滚动条，比较常用的是 VScroll Bar 复选框。
- 当选中 Sorted 复选框时，列表框中的选项将按字母顺序排序，否则按建立次序显示。
- DisableScroll 复选框用于指定列表框上滚动条的行为。默认时未被选中，它表示只在需要时(项目多或超出了列表框的宽度)才显示滚动条。当该复选框被选中时，则无论列表框中有多少选项，滚动条总显示在列表框中。
- MultiSelect 复选框用于指定是否允许用户进行多项选择。选中该复选框时，允许用户进行多项选择，此时只要单击多个选项就能选择多个选项，否则只能选择一个选项。
- 当选中 ExtendedSelect 复选框时，同样允许进行多项选择，只是其选择方法与选中 MultiSelect 复选框时不同。如果同时选中这两个复选框，则与单独选中 Extended Select 复选框时的效果相同。

技巧：　选中 ExtendedSelect 复选框后，通过下述方法选择选项：

(1) 单击其中一个选项。
(2) 按住鼠标左键拖动选中多个连续选项。
(3) 先选中一个选项，按住 Shift 键，然后单击选中连续多个选项。
(4) 按住 Ctrl 键，然后单击选中多个不相连选项。

下面讲解如何定义列表框选项的方法。如果在设计应用程序时就已经知道了列表框各选项，则可使用直接输入法，否则就要通过编写代码来完成。直接输入列表框选项的步骤如下：

(1) 单击列表框属性设置对话框的 Items 标签。

(2) 在 Text 下的编辑框中依次输入列表框各选项，定义的列表项如图 6-19 所示，运行结果如图 6-20 所示。

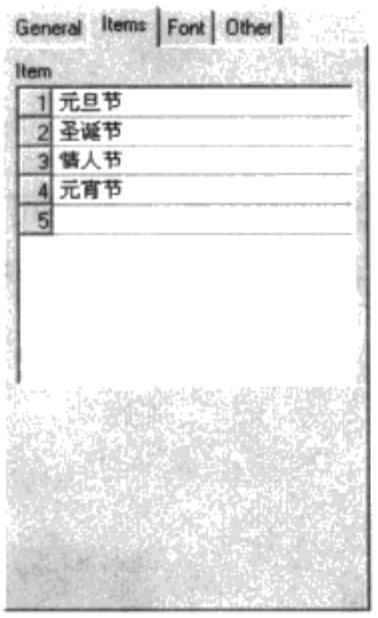


图 6-19 定义列表框选项

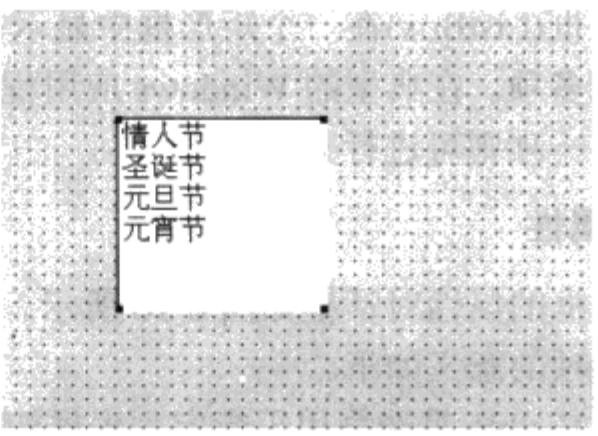


图 6-20 列表框运行结果示例

动态增加、删除列表框选项时需要编写相应的代码。通常可以在窗口的打开(Open)事件中编写，此时需要调用列表框的 AddItem()、InsertItem()和 DeleteItem()对象函数，其调用方法为：

```
listboxname.AddItem ( item )  //在列表框尾部增加列表项
itemlistboxname.InsertItem (item, index )  //在第 index 项前面插入列表项
itemlistboxname.DeleteItem (index )  //删除第 index 个列表项
```

其中 listboxname 是列表框的变量名；item 是要添加或插入到列表框中的选项，用字符串表示；index 代表第几个选项，数据类型为整型。

2. 事件

列表框有 10 多个事件，常见的有双击(DoubleClicked)事件、得到焦点(GetFocus)事件、失去焦点(LoseFocus)事件。选择一个新的选项时都将触发变更选择(SelectionChange)事件；双击列表框的选项，将触发 DoubleClicked 事件。

6.4.2 图片列表框

图片列表框(PictureListBox)与列表框十分相似，不同之处在于图片列表框内不仅有选项，而且每个选项前还可以有一个小图片。

1. 属性

图片列表框的 General 属性页与列表框基本相同，相应属性的含义请参阅 6.4.1 节中列表框的解释。图片列表框与列表框的主要区别表现在 Pictures 选项卡上，如图 6-21 所示。

在这个选项卡中可以定义一组图片，同时每个图片前面都有一个编号，该编号用于将列表项与图片联系起来。定义图片的步骤如下：

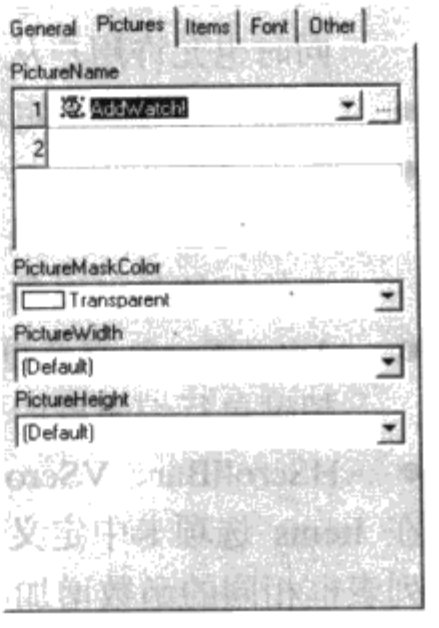


图 6-21　图片列表框的属性对话框

(1) 打开图片列表框的属性对话框，单击 Pictures 选项卡。

(2) 单击 PictureName 下的编辑框。

(3) 从 StockPictures 列表中选择一个图片，或单击 Browse 按钮，从磁盘中选择一个.bmp 文件、.ico 文件或.cur 文件，图片旁边的数字即是该图片的编号(用于与列表项联系)。

(4) 在 Width 和 Height 列表框中选择图片的宽度和高度，当选择 Default 时将按图片本身的大小显示。

(5) 在 PictureMaskColor 列表框中选择对图片做掩码运算的颜色，默认时为透明(Transparent)方式。

(6) 单击 Apply 按钮确认选择。定义了图片之后，还需要将图片与列表项联系起来，方法是单击 Items 按钮，输入图片列表框选项之后，修改 PictureIndex 下的编号，该编号与定义图片时的编号相对应。

注意： 当选择图片的宽度和高度时，实际上指定的是第 1 个图片的高度与宽度，其他图片将以此数据进行放大或缩小。

2. 事件

图片列表框的事件及触发时机与列表框完全相同，在此不再重复。

技巧： 程序中通过使用图片列表框的对象函数 AddItem()、InsertItem()、DeleteItem()完成列表项的添加、插入、删除操作。添加、删除图片的对象函数是 AddPicture()和 DeletePicture()。对象函数 DeletePictures()删除列表框中的所有图片。对象函数 DirList()用指定路径下的文件名填充列表框。对象函数 SelectedItem()返回用户选中的第一个列表项。

6.4.3　下拉列表框

下拉列表框(DropDownListBox)也叫组合框，样式如图 6-22 所示，它把单行编辑框与列表框的功能结合起来，具有这两种控件的特点。下拉列表框有两种类型，不可编辑的下拉列表框和可编辑的下拉列表框。对可编辑的下拉列表框来说，可以在下拉列表框的编辑框中输入该控件中没有的列表选项。

1. 属性

下拉列表框的一般属性如图 6-23 所示，从中可以看到它的属性融合了列表框与编辑框的特点。

各选项的意义如下：

- 选中 AllowEditing 复选框时，表示允许用户在列表框的编辑框中输入自己的数据，同时也允许用户从列表中进行选择，否则就只能从列表中选择数据。
- Sorted 复选框指定是否对各选项进行排序，对中文来说，排序的意义不大。
- 选中 AlwaysShow List 复选框时，下拉列表框的选项列表总显示在窗口上；未选中该复选框时，只有当用户单击下拉列表框上的箭头时才显示选项列表，这种做法的好处是可以节省窗口空间。
- AutoHScroll 复选框是针对单行编辑框部分而言的，选中该复选框时，当文本长度超过单行编辑框的长度时，文本自动向左滚动。
- HScrollBar、VScrollBar 复选框指定是否显示水平、垂直滚动条。

在 Items 选项卡中定义下拉列表框的选项，其用法与列表框完全相同，并且也可以使用与列表框相同的函数增加、插入或删除选项。

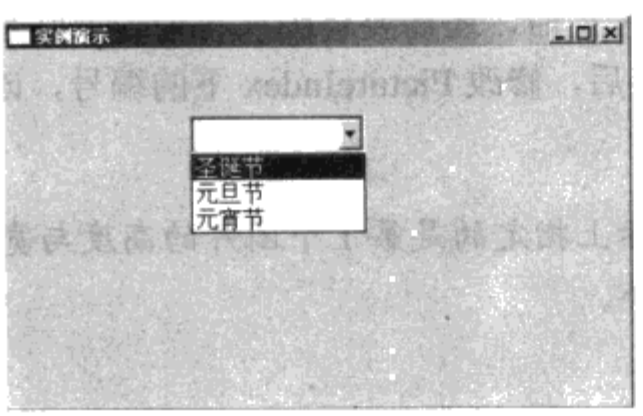


图 6-22　下拉列表框示例

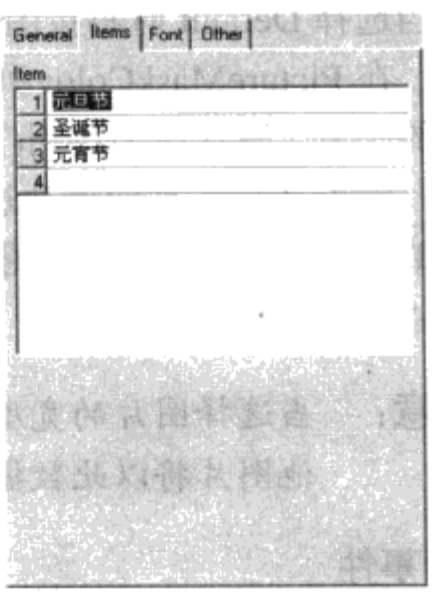


图 6-23　下拉列表框的属性对话框

2. 事件

下拉列表框有 10 多个事件，常用事件如下：

- SelectionChange 和 DoubleClicked 事件的触发时机和作用与列表框相同。
- Modified 事件在改变下拉列表框上的单行编辑框中的文本时触发。

6.4.4　下拉图片列表框

下拉图片列表框(DropDownPictureListBox)与下拉列表框的区别在于前者在选项的前面可以显示一个图像。

1. 属性

下拉图片列表框的 General 属性页与下拉列表框基本相同，可参照下拉列表框予以设置。下拉图片列表框的 Pictures 和 Items 属性页与图片列表框也相同，同样可以参照设置，在此不再赘述。

2. 事件

下拉图片列表框的事件与下拉列表框的事件相同，这里不再赘述。

技巧： 对于下拉列表框和下拉图片列表框来说，如果仅仅在相应控件的单行文本框部分输入数据，并不会触发该控件的 SelectionChange 事件。当在程序中删除图片列表框、下拉图片列表框的图片时，使用相应控件的 DeletePicture()函数。

6.5　图像显示控件

PowerBuilder 10.0 提供的图片框控件以便利的方式展现多姿多彩的丰富图像。在图片框控件中能够显示 BMP、RLE 或 WMF 格式的图像，如果与定时器函数(Timer())配合起来，通过对窗口的 Timer 事件编程，可以让图片框中的图像动起来，达到简单的动画效果。下面扼要叙述图片框的属性与事件，并通过实例说明运用图片框制作简单动画的过程。

图片框(Picture)用于显示 BMP、RLE 或 WMF 格式的图像，它在窗口画板工具栏控件图标列表中的图标为。

1．属性

图片框的属性对话框如图 6-24 所示。各选项的意义如下：

- Name 编辑框指定图片框的变量名。
- PictureName 编辑框指定显示在图片框中图像的文件名，在这里既可以直接输入一个图像文件名，也可以单击 Browse 按钮，从打开的对话框中选择一个图像文件名(注意，图像文件的格式只能是 BMP、RLE 或 WMF 三者之一)。
- 选中 FocusRectangle 复选框时，当图片框得到焦点后，系统将在图片框的边缘显示一个由虚线组成的方框(虚框)，以提示用户当前的焦点位置；否则即使图片框得到焦点也不显示虚框。
- 选中 OriginalSize 复选框时，将根据图像尺寸自动改变图片框的大小；否则压缩或拉伸图像以适应图片框的大小。
- 选中 InvertImage 复选框后，将反转图片颜色，它将所有黑色变成白色，所有白色变成黑色，其他颜色也变成与之相对的颜色。

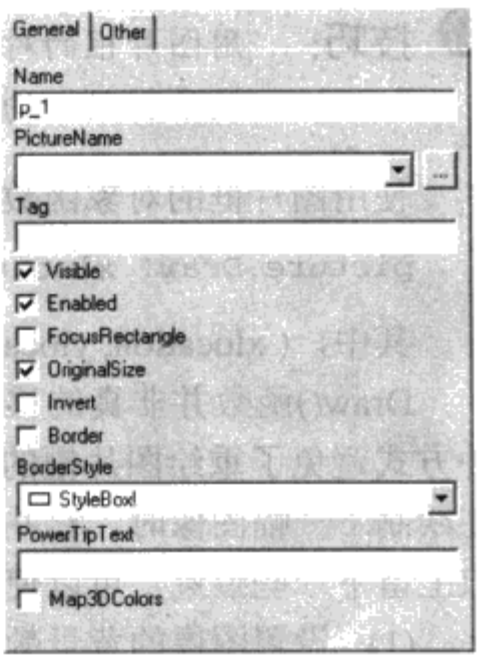


图 6-24　图片框的属性对话框

2．事件

图片框有两个重要事件，分别是 Clicked 和 DoubleClicked 事件，Clicked 事件在单击图片框时触发，DoubleClicked 事件在双击图片框时触发。

3．示例

动画不仅能够使应用程序界面更活泼生趣，还可以吸引用户的注意力。利用图片框，通过简单的编程就可以构造出动画。动画的基本做法是选择一幅图像，然后在程序运行时动态变换图像或改变图像的显示位置。因此，在具体制作动画之前，首先必须设计或选择

一幅或多幅图像，当选择多幅图像时图像之间应该相对连贯。

使用的图像可以是 PowerBuilder 10.0 自身携带的，也可以是 PowerBuilder 10.0 携带的图像编辑器 WATCOM Image Editor 新创建的。

在窗口的 DoubleClicked 事件中添加以下代码：

```
integer li_x,li_y
li_x=p_1.x
li_y=p_1.y
Do  While p_1.x < (this.Width - 250)
p_1.x++
Loop
Do  While p_1.y < (this.Height / 2)
p_1.y++
Loop
Do  While p_1.x > li_x
p_1.x --
Loop
Do  While p_1.y > li_y
p_1.y--
Loop
```

运行程序后在窗口上双击，图像就会在窗口中移动，基本的动画效果就有了，但缺点是在移动图像的过程中出现了闪烁现象。

技巧： 用图片框的对象函数 Move()移动图片框的效果与改变该对象的 X、Y 属性的效果相同，均存在闪烁现象。

使用图片框的对象函数 Draw()可避免移动过程中的闪烁现象。Draw()函数的语法为：

```
picture.Draw( xlocation, ylocation)
```

其中，(xlocation, ylocation)是图像移动到的新位置，picture 是图片框对象的变量名。

Draw()函数并非真的移动图片框的位置，而是直接在窗口上绘制图片框中的图像，这种方式避免了重绘图片框的控件和窗口，因此速度更快，减少和避免了闪烁。Draw()函数在绘制下一幅图像时，它并不自动清除前一幅图像，当使用 Draw()函数制作动画时会在窗口上留下一些痕迹，可以使用以下两个办法解决。

(1) 设置图像的背景颜色与窗口的背景颜色相同。

(2) 在窗口上再放置一个图片框，该图片框中的图片选择窗口的背景颜色，移动前一个图片框时，让后一个图片框跟在前一个图片框后面移动，由后一个图片框覆盖前一个图片框遗留的痕迹。

现在修改前面的示例，使用 Draw()函数替代修改图片框的 X、Y 属性，代码如下：

```
integer li_x,li_y
li_x=p_1.x
li_y=p_1.y
Do  While li_x < (this.Width - 250)
li_x++
p_1.Draw(li_x,li_y)
Loop
Do  While li_y < (this.Height / 2)
li_y++
p_1.Draw(li_x,li_y)
Loop
```

```
Do  While li_x > p_1.x
li_x --
p_1.Draw(li_x,li_y)
Loop
Do  While li_y > p_1.y
li_y --
p_1.Draw(li_x,li_y)
Loop
```

运行程序后，双击窗口，动画效果就顺畅多了。

6.6　树形视图控件

树形视图(TreeView)控件提供了表现数据间层次关系的直观方法，可以移动、删除或复制相应的列表项；也能够以大图标、小图标、列表及报告方式等多种形式表现同一组数据。

使用树形视图控件可以实现类似于 Windows 的资源管理器的功能。在它左部的列表中以层次结构列出了驱动器、目录及子目录，通过单击层次结构中的加号(+)、减号(-)来展开或折叠目录，同时每个目录前面都有一个代表其类别的图标，可分辨出相应目录的路径。另外数据还可以被分层次地显示在列表中。

6.6.1　属性

在窗口画板工具栏中，树形视图控件图标为，选中该图标，然后在窗口上单击，树形视图控件就被放置到窗口上了。树形视图控件的属性较多，其属性对话框如图 6-25 所示。各选项的意义如下。

- Name 编辑框用来指定 TreeView 控件的变量名。
- 选中 Visible 复选框时该控件可以显示，否则被隐藏。
- 选中 Enabled 复选框时该控件可用，否则用户不能操作该控件。
- 选中 DeleteItems 复选框时，能够使用 Delete 键删除 TreeView 控件中选中的列表项，否则不能删除列表项。
- 选中 ShowButtons 复选框时，树形视图控件在父节点项前显示加(+)号或减(-)号。
- 选中 HideSelection 复选框时，即使 TreeView 控件失去焦点，选中的列表项依然加亮显示，否则，当 TreeView 控件失去焦点时，选中的列表项恢复常态不再加亮显示。
- 选中 DisableDragDrop 复选框时，不能对列表项执行拖放操作，否则用户能够拖放列表项。
- 选中 EditLabels 复选框时，能够编辑列表项的标题，否则不能编辑。
- 选中 ShowLines 复选框时，树形视图控件在节点间显示连线。
- 选中 LinesAt Root 复选框时，树形视图控件用连线连接所有根节点。
- Indentation 编辑框用于指定缩进的宽度。
- Sort 列表框用于指定列表项的排序方式，其中 Ascending 表示升序；Descending 表示降序；Unsorted 表示不排序；UserDefined 表示用户定义排序方式。选择 UserDefined 排序方式后，程序运行时将会触发树形视图控件的 Sort 事件。

在树形视图的 Pictures 选项卡中设置各节点前显示的图像，如图 6-26 所示。

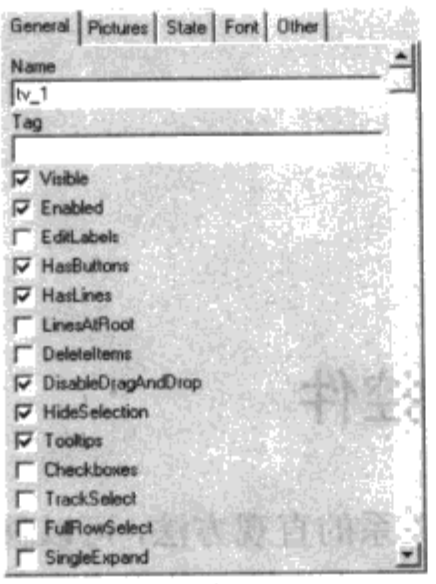


图 6-25　树形控件的属性对话框

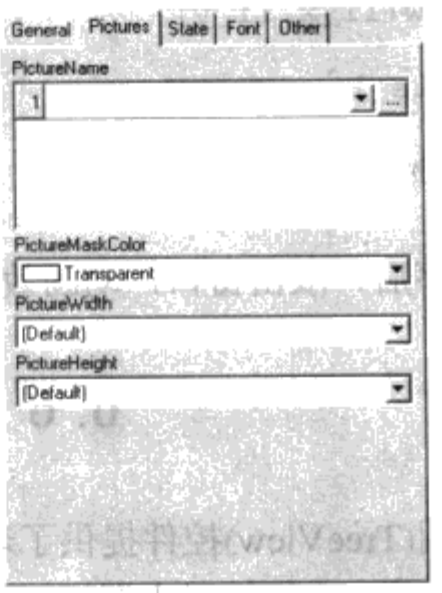


图 6-26　Pictures 选项卡

既可以通过单击选用 StockPictures 列表中系统预定义图片，也可以通过单击 Browse 按钮，从打开的对话框中选择一个图片文件(文件类型为 BMP、RLE 或 WMF)，图片的名称将出现在 PictureName 列表中。

PictureMaskColor 下拉列表框用于选择图片的背景颜色，默认值为透明(Transparent)；Width 和 Height 下拉列表框分别用于选择图片的宽度和高度。在 StatePicture 属性页中按类似方式设置状态图像。

6.6.2　添加与删除列表项

当设置完树形视图的各种属性后，接下来的工作就要建立它的数据项了，但在树形视图的属性对话框中不能直接添加和删除列表项，这就是说，只能通过编程才能向树形视图中增加或删除数据。树形视图的每个数据项都是一个 TreeViewItem 对象，通过调用相应函数增加和删除列表项。它包括一组属性，各属性的意义如表 6-1 所示。

表 6-1　TreeViewItem 的属性

属性名	数据类型	用途与意义
Data	Any	需要与数据项一起保存的任何数据
Bold	Boolean	指定是否以黑体字形式显示各节点的文本
Children	Boolean	指定当前项是否带有子项
CutHighLighted	Boolean	指定当当前项是剪切(Cut)操作的对象时该项是否加亮显示
DropHighLighted	Boolean	指定当当前项是拖放(DragDrop)操作的对象时该项是否加亮显示
Expanded	Boolean	指定当前项是否能进一步展开。如果该项后面有子项，那么该属性的值为 True，否则为 False
ExpandedOnce	Boolean	指定当前项是否被至少扩展一次
HasFocus	Boolean	指定当前项是否得到焦点
ItemHandle	Long	引用当前项的惟一编号

续表

属性名	数据类型	用途与意义
Label	String	指定当前项的显示文本
Level	Integer	指定当前项在树形视图层次结构中的层数
OverlayPictureIndex	Integer	指定当前项使用的重叠图像索引号。重叠图像显示在项目图像的上方，该值为 0 时不显示重叠图像
PictureIndex	Integer	这是个图片索引号，相应图片显示在该项标题的左边
SelectedPictureIndex	Integer	用于指示当前项被选中的图片的索引
Selected	Boolean	指定当前项是否被选中
StatePictureIndex	Integer	该项使用的状态图像列表中的图片索引号

1．添加树形视图列表项

树形视图的列表项只能通过代码来添加，为此 PowerBuilder10.0 提供了 4 个函数来完成这个任务，各函数的用途和参数略有区别，编程时根据需要选用，下面简单介绍一下这 4 个函数。

(1)　InsertItem()函数

InsertItem()函数在指定位置插入列表项，有两种语法格式：

```
treeviewname.InsertItem( handleparent, handleafter, label,pictureindex )
treeviewname.InsertItem( handleparent, handleafter, item )
```

InsertItem()函数在 handleafter 项后插入一个列表项，并使 handleparent 项成为当前项的父项。这两种语法格式的区别在于前一种语法格式只需指定当前项的标题(label)和使用的图片索引(pictureindex)；后一种语法格式要求定义当前项的完整数据(在 item 变量中)。比较而言，前一种格式要求的信息较少，因此功能也弱些。

各参数的意义如下：

- treeviewname：树形视图控件的名称。
- handleparent：该项父节点项的句柄，数据类型为长整型，每个节点都有惟一代表该节点的句柄，对于第一层节点，该项的值为 0。
- handleafter：在同一层中句柄为 handleafter 的项后插入新项。
- label：当前项的标题，即显示在树形视图控件中的文字。
- pictureindex：当前项标题前显示图片的索引号。
- item：一个完整的列表项，是 TreeViewItem 对象。

(2)　InsertItemFirst()　函数

InsertItemFirst()函数在指定父节点层中的第一个位置插入列表项，有两种语法格式：

```
treeviewname.InsertItemFirst( handleparent, label,pictureindex )
treeviewname.InsertItemFirst( handleparent, item )
```

父节点相同的项位于同一层中。InsertItemFirst()函数在指定层中插入一个新的列表项，并把它放在列表的第一个位置上。前一种语法格式只需指定当前项的标题(label)和使用的图片索引(pictureindex)，后一种语法格式要求定义当前项的完整数据(在 item 变量中)。

(3) InsertItemLast()函数

InsertItemLast()函数在指定父节点的层中的最后位置插入列表项，有如下两种语法格式：

```
treeviewname.InsertItemLast( handleparent, label, pictureindex )
treeviewname.InsertItemLast( handleparent, item )
```

该函数与 InsertItemFirst()函数相似，只是在指定层的末尾插入列表项。

(4) InsertItemSort()函数

该函数以排序方式在指定父节点的层中插入列表项，有两种语法格式：

```
treeviewname.InsertItemSort( handleparent, label, pictureindex )
treeviewname.InsertItemSort( handleparent, item )
```

当树形视图控件的 SortType 属性设为 Unsorted!之外的值时(即 Ascending!、Descending!或 UserDefinedSort!)，树形视图控件总是以排序方式显示列表项，这时没有必要使用 InsertItemSort()函数来排序插入列表项，使用其他格式的 InsertItem 类函数将得到相同的结果。

当树形视图控件的 SortType 属性设置为 Unsorted!后，InsertItemSort()函数在插入列表项时，把它插到以字母顺序升序排序的适当位置，当树形视图控件原有各列表项未曾排序时，InsertItemSort()函数插入列表项的结果不确定。

要灵活使用上述函数，需要理解树形视图列表项的结构。由于树形视图中的数据以层次方式显示，因此每一层都有一个整数值。

树形视图控件的每一层中可以包含一个或多个列表项，每个列表项都利用惟一的句柄(对应于 ItemHandle 属性)来标识。虽然可以指定 ItemHandle 属性的值，但常在插入列表项时由 PowerBuilder 自动生成。

2. 删除树形视图列表项

使用 DeleteItem()函数，删除树形视图控件中指定的列表项，该函数的语法格式为：

```
treeviewname.DeleteItem (itemhandle )
```

其中 treeviewname 是树形视图控件名；itemhandle 是要删除项的句柄。DeleteItem()函数在删除列表项时，若该列表项还有子项，子项一同被删除。当 itemhandle 值为 0 时，DeleteItem()函数删除树形视图控件中所有列表项。例如：

```
tv_1.DeleteItem(1)  //删除句柄为 1 的列表项及其子项
tv_1.DeleteItem(0)  //删除所有列表项
```

3. 查找列表项的句柄

从前面叙述中可以看出，在操作列表项时，往往需要知道列表项的句柄。这时可以使用 FindItem()函数，其语法格式如下：

```
treeviewname.FindItem ( navigationcode, itemhandle )
```

参数解释如下：

- treeviewname 是树形视图控件的名称。
- itemhandle 是查找起点列表项的句柄。

- navigationcode 是查找代码，也就是说按什么要求查找列表项，该参数的类型是枚举数据类型 TreeNavigation。
- RootTreeItem!：查找第一层的第一个列表项，如果树形视图控件中还没有列表项，那么 FindItem()函数返回-1。
- NextTreeItem!：查找与 itemhandle 列表项同一个父节点中同一层的下一个列表项，无下一个列表项时返回-1。
- PreviousTreeItem!：查找与 itemhandle 列表项同一个父节点中同一层的前一个列表项。
- ParentTreeItem!：查找 itemhandle 列表项的父节点，如果 itemhandle 列表项是第一层的列表项，那么 FindItem()函数返回-1。
- ChildTreeItem!：查找 itemhandle 节点的第一个子列表项。如果该列表项被折叠，那么 ChildTreeItem!参数调用 FindItem()函数后，该列表项被展开显示。如果没有子列表项，该函数返回-1。
- FirstVisibleTreeItem!：找出树形视图控件当前可视区域中显示的第一个列表项，而不管该列表项位于哪一层上。树形视图控件上滚动条的位置决定了哪一个列表项是第一个可视列表项。
- NextVisibleTreeItem!：找到 itemhandle 之后的下一个已展开列表项，而不考虑该列表项的层次。当已经到达树形视图控件的最后一个已展开列表项时，该函数返回-1。
- PreviousVisibleTreeItem!：找到 itemhandle 前面一个已展开列表项，而无须考虑该列表项的层次。当已经到达树形视图控件的第一个根列表项时，该函数返回-1。
- CurrentTreeItem!：找出当前所选项。
- DropHighlightTreeItem!：找出 DropHighlighted 属性最近被设置为 True 的列表项。

注意： 除了使用 ChildTreeItem!参数外，FindItem()函数不查找任何折叠的列表项。

下面是 FindItem()函数的两个示例。

示例 1：

```
long ll_tvi
ll_tvi = tv_list.FindItem(CurrentTreeItem!, 0)
```

可以得到树形视图控件中当前所选项的句柄。

示例 2：

```
long ll_tvi
ll_tvi = tv_list.FindItem(RootTreeItem!, 0)
```

可以得到树形视图控件中第一层第一个列表项的句柄。

除了前面介绍的函数外，树形视图控件还有一些常用函数，表 6-2 所示为这些函数的名称、参数及主要用途。

表 6-2 树形视图控件的其他常用函数

函数名	参 数	用 途
AddPicture()	picturename	在图像列表中增加图片
DeletePicture()	index	从图像列表中删除 index 指定的图片
DeletePictures()	无	删除图像列表中的所有图片
CollapseItem()	itemhandle	折叠 itemhandle 指定的列表项
EditLabel ()	itemhandle	让 itemhandle 指定的列表项处于编辑状态
ExpandAll()	itemhandle	递归地展开指定列表项的所有子项
ExpandItem()	itemhandle	展开指定列表项的子项
GetItem()	itemhandle, item	将指定项的所有数据放到 TreeViewItem 变量 item 中
SelectItem()	itemhandle	将指定项选为当前项
SetItem()	itemhandle, item	使用 TreeViewItem 变量 item 中的数据设置指定项
Sort()	itemhandle ,sorttype	按指定方式排序 itemhandle 项的子项
SortAll()	itemhandle, sorttype	按指定方式递归排序 itemhandle 项的所有子项

6.6.3 事件

下面讲述树形视图控件几个重要事件及其触发时机，具体如下。

- BeginDrag：当在树形视图控件的某个列表项的标题上按下鼠标左键并开始拖曳鼠标时触发，该事件有一个 handle 参数，它是被拖曳列表项的句柄。
- BeginLabelEdit：当编辑某个列表项的标题时触发，该事件有一个 handle 参数，它是被编辑列表项的句柄。通过设置该事件的返回值为 1 可以禁止编辑该列表项的标题，返回值为 0 时允许编辑。
- BeginRightDrag：当在树形视图控件的某个列表项的标题上按下鼠标右键并开始拖曳鼠标时触发，该事件有一个 handle 参数，它是被拖曳列表项的句柄。
- DeleteItem：删除列表项时触发，而不论是直接删除列表项还是在程序中使用 PowerScript 函数删除列表项。该事件有一个 handle 参数，它指示被删除列表项的句柄。
- EndLabelEdit：当结束某个列表项标题的编辑时触发。该事件返回 1 放弃修改，返回 0 接受修改。这个事件有两个参数，分别为 handle 和 newtext，handle 参数指示被修改列表项的句柄，newtext 参数是用户编辑的新标题。
- ItemCollapsed：在某个列表项被折叠后触发。
- ItemCollapsing：在某个列表项要被折叠时触发。该事件返回 1 阻止被折叠，返回 0 完成折叠。
- ItemExpanded：在某个列表项被展开后触发。
- ItemExpanding：在某个列表项要被展开时触发。该事件返回 1 阻止被展开，返回 0 完成展开。每次展开列表项时都需要填充该列表项下的子项，可通过编写该事件的处理程序来完成。若列表项没有子项，就不能被展开。

- ItemPopulate：当第一次展开某项时触发，该事件返回 1 阻止被展开，返回 0 完成展开。如果列表项没有子项，那么就不能被展开。一般来说，该事件和 ItemExpanding 事件常用于生成相应列表项的子列表项。
- SelectionChanged：当选择新的选项后触发。
- SelectionChanging：当选择新的选项时触发，该事件返回 1 阻止改变当前选项，返回 0 允许改变当前选项。
- Sort：当使用 UserDefinedSort!参数调用函数 Sort()时触发，并且每做一次比较都会触发该事件。利用该事件可以定义排序次序。将树形视图的 SortType 属性设置为 UserDefinedSort!，然后根据需要返回下述 3 个值之一。
  - -1：handle1 项小于 handle2 项时。
  - 0：handle1 项等于 handle2 项时。
  - 1：handle1 项大于 handle2 项时。

其中，handle1、handle2 是 Sort 事件的两个长整型参数，handle1 是排序操作中第 1 个比较项的句柄，handle2 是第 2 个比较项的句柄。

6.6.4　使用树形视图控件显示数据

本实例将介绍 TreeView 控件的使用，即在一个 TreeView 控件以树状方式中显示数据表中的详细数据。从本实例中读者可以学到与 TreeView 控件相关的对象和函数的用法，了解用 TreeView 控件显示数据的特点。

1. 创建程序的基本框架

(1) 建立 work 工作区。
(2) 建立 treeview 应用。
(3) 建立 w_treeview 窗口，其类型为 main!，其 title 为“TreeView 控件之显示数据”。
(4) 建立控件。在 w_treeview 窗口上添加如表 6-3 所示的控件。
(5) 建立数据窗口对象。建立一个名为 d_product 的数据窗口对象。

表 6-3　本实例控件列表

控件名	Name	用途
TreeView	tv_product	显示 product 表中详细信息
CommandButton	cb_scan	扫描数据表按钮
	cb_expand	展开数据表按钮
	cb_close	退出窗口按钮
DataWindow	dw_product	显示 product 表中的数据

2. 编写事件驱动代码

(1) 在应用的 Open 事件中编写连接数据库和打开主窗口的代码。
(2) 在主窗口 w_treeview 的 Open 事件中编写以下代码：

```
dw_product.settransobject(sqlca)
```

```
//数据窗口连接事务
dw_product.retrieve()
//取出数据
```

为窗口创建一个实例变量：

```
long xt_rootitem
```

(3) 在 cb_scan 控件的 Clicked 事件中编写以下代码：

```
long xt_NewItem
integer rowcount
treeviewitem xt_tvitem
int i
//定义变量
tv_product.setredraw(false)
tv_product.deleteitem(xt_rootitem)
xt_tvitem.label="PRODUCT ID"
xt_tvitem.pictureindex=1
xt_tvitem.selectedpictureindex=1
xt_rootitem=tv_product.insertitemlast(0,xt_tvitem)
//插入根项目"PRODUCT ID"
rowcount=dw_product.rowcount()
for i=1 to rowcount step 1
...
next
//以上为循环插入各个产品的详细信息
tv_product.setredraw(true)
```

(4) 在 cb_expand 控件的 Clicked 事件中编写以下代码：

```
tv_product.expandall(xt_rootitem)
```

(5) 在 cb_close 控件的 Clicked 事件中编写以下代码：

```
close(w_treeview)
```

(6) 单击 PowerBuilder 10.0 开发界面上的 按钮或按下 Ctrl+R 快捷键，程序就可以运行。本实例最终效果如图 6-27 所示。

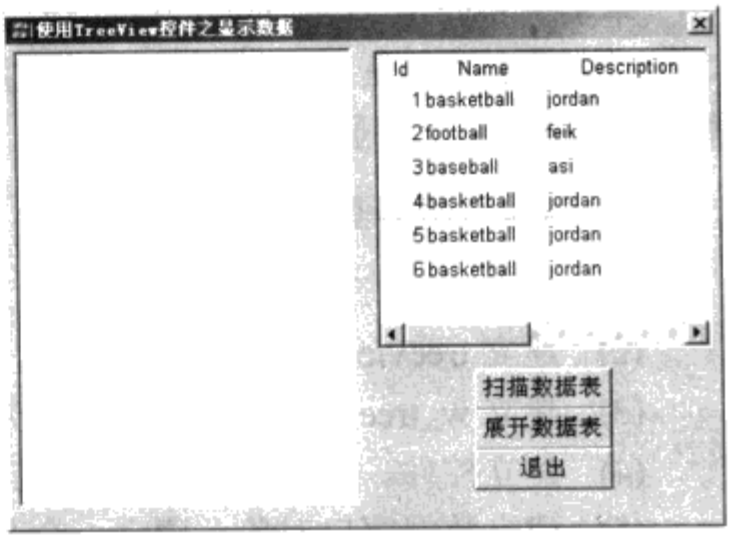


图 6-27 程序运行界面效果图

6.7 使用标签控件

标签控件(Tab Control)是在 PowerBuilder 开发环境中使用最多的控件之一，当设置某个对象的属性时，打开的属性对话框使用的就是标签控件。标签控件能够把多种数据和程序选项封装在一个控件中，并允许通过简单的操作访问这些数据和选项。一般来说，标签控件由多个标签组成。本书介绍标签控件的概念、属性、事件和用法。

6.7.1 相关概念

下面简单地介绍一下与标签控件相关的概念。

1. 标签控件

与按钮、复选框、文本框相似，标签控件(Tab Control)也可以放在窗口或用户对象的控件上。它由两部分组成，一是放置各选项卡标题的标签部分，二是放置选项卡的区域。

2. 标签页

标签页可以用来展示数据和选项，一般来说，每个标签控件都有多个选项卡，分别对应多组数据或选项。这些选项卡占据标签控件中的相同区域，只能看到当前选项卡，其他选项卡被当前选项卡覆盖。

3. 标签

标签(Tab)是选项卡的标题，同时也是选项卡的标识。当某个选项卡被其他选项卡遮挡时，单击该选项卡的标题，这个选项卡就成为当前标签页，显示在标签控件中，同时遮住其他标签页。

也可以把标签控件看作是一个有多个抽屉的文件柜，每个抽屉上都有一个说明其内容的标签(标题)，抽屉是存放具体物品(控件)的容器，这个文件柜一次只能打开一个抽屉，并且总有一个抽屉处于打开状态。从抽屉的标签上可以看出抽屉可能存放的内容，需要具体内容时，再打开相应的抽屉。

6.7.2　建立标签控件

1. 放置标签控件

在窗口上放置标签控件的方法如下：

(1) 用窗口画板打开要放置标签控件的窗口。

(2) 在窗口画板工具栏中，单击控件图标列表中的标签控件图标，然后在窗口上单击，标签控件就被放置到窗口上。

2. 增加新的标签页

在窗口上放置了标签控件后，该控件自动携带了一个空白的标签页，该标签页作为嵌入式标签页(TabPage)。在这个标签页中，可以像在窗口上放置控件那样放置所需的控件。除了嵌入式标签页外，还可以把一个可视用户对象作为标签控件的标签页。下面分别介绍增加这两种标签页的方法。

增加一个新的嵌入式标签页的步骤如下：

(1) 在显示标签的区域右击，弹出如图 6-28 所示的快捷菜单。

(2) 从弹出的快捷菜单中选择 Insert TabPage 命令。

(3) 一个新的空白嵌入式标签页插入到标签控件中。

(4) 在该标签页上添加所需控件。

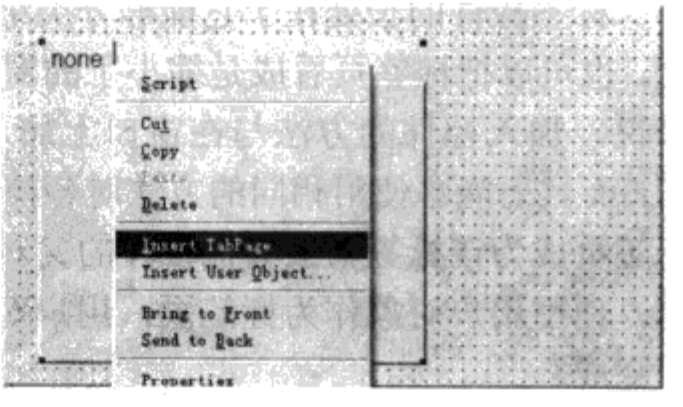


图 6-28　标签页右键菜单

把一个用户对象增加为标签页的步骤如下：

(1) 使用用户对象画板，创建要作为标签页使用的定制可视用户对象，编写所需的事件处理程序，并且调整用户对象的大小，使其与标签控件标签页的显示区域大小一致。

(2) 在用户对象属性对话框的 TabPage 标签页中设置该对象作为标签页时的标签、图像、提示、前景颜色和背景颜色，然后关闭用户对象画板并保存该用户对象。

(3) 在标签控件上部显示标签的区域右击。

(4) 从弹出的快捷菜单中选择 Insert User Object 命令，显示 Select User Object 对话框，从中选择作为标签页使用的用户对象，单击 OK 按钮关闭对话框。

(5) 所选用户对象作为标签页插入到标签控件中。

嵌入式标签页与用户对象标签页的区别在于，嵌入式标签页只能够服务于当前的标签控件，而用户对象标签页能够应用于多种需要该用户对象的标签控件，具有较高的可重用性。

注意： 用户对象标签页只能在用户对象画板中修改和编程，这些工作不能够直接在窗口画板中完成。

3. 删除标签页

当不需要某个标签页时，可以将其删除，方法如下：

(1) 单击要删除标签页的标签，该标签页显示在标签控件中。

(2) 右击标签页区域，显示弹出菜单。

(3) 从中选择 Clear 或 Cut 菜单项。两者的区别在于 Clear 菜单项清除当前标签页，Cut 菜单项将当前标签页剪切到 PowerBuilder 剪贴板上。

4. 调整标签位置

重新组织标签控件中各标签的左右次序，具体操作步骤如下：

(1) 在控件上部显示标签的区域右击，从弹出菜单中选择 Properties...菜单项，打开标签控件的属性对话框。

(2) 单击 Page Order 选项卡，如图 6-29 所示。

(3) 在 Page Order 选项卡中通过拖曳操作调整各标签的位置，位于该列表框上部的标签将出现在标签控件的左部。

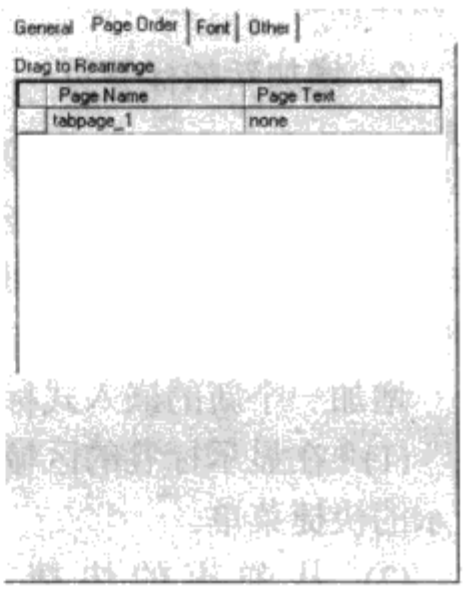


图 6-29 打开 Page Order 选项卡

5. 标签页中的控件

标签控件的长处在于它能够分类组织多种其他控件，因此，也可以把标签页看成是缩小了的窗口。对嵌入式标签页来说，加入控件的方法与在窗口上加入控件的方法基本相同，而且还能够使用相同的方法对各控件进行编程，相应的标签页成为该标签页上其他控件的父对象。

使用用户对象作为标签页，用户对象上控件的放置、修改、编程均需在用户对象画板中完成。

通过复制、剪切、粘贴操作可以在标签页之间、标签页与窗口之间移动或复制对象，

但是不能通过拖曳操作直接把标签页中的控件拖曳到窗口上，反之也不能把窗口上的控件直接拖曳到标签页上，使其成为标签页的一部分。

6.7.3　属性

在设计标签控件时，常常要用到两个属性对话框，一个是标签控件的属性对话框，另一个是标签页的属性对话框。标签控件属性对话框设置该控件以及所有标签页公用的属性，标签页属性对话框设置各个标签页的具体属性，每个标签页都有相应的标签页属性对话框。图 6-30 和图 6-31 分别显示了这两个属性对话框。

打开标签控件属性对话框的方法是在标签控件上部显示标签的区域内右击，从弹出菜单中选择 Properties...菜单项，或通过单击选中标签控件后单击窗口画板工具栏上的 Properties 图标，打开标题为 Tab 的标签控件属性对话框，如图 6-30 所示。

打开标签页属性对话框的方法如下：

(1)　单击设置属性的标签页的标签，使该标签页成为当前标签页。

(2)　右击标签页区域(不要在标签区域右击，也不要右击标签页上的任何控件)，从弹出菜单中选择 Properties...菜单项。

(3)　打开标题为 TabPage 的标签页属性对话框，如图 6-31 所示。

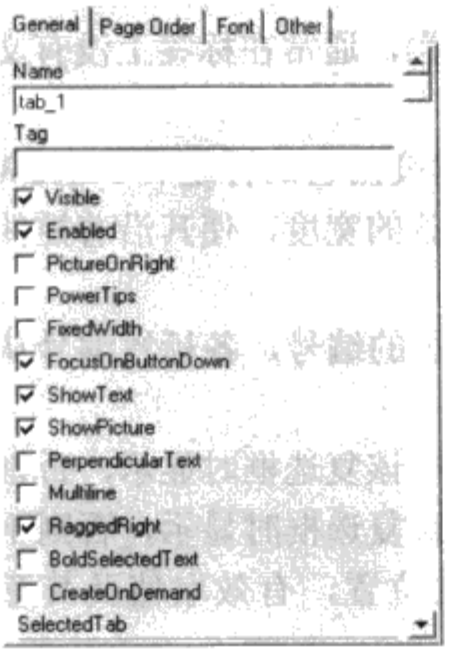


图 6-30　标签控件属性对话框

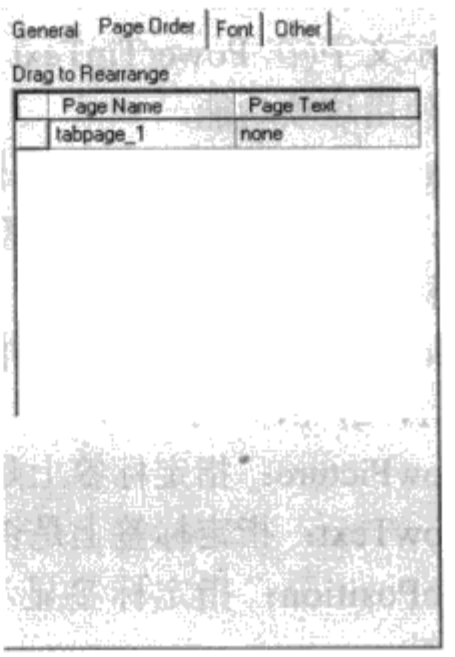


图 6-31　标签页属性对话框

1．标签控件属性

在标签控件属性对话框中可以设置如下标签控件的选项。

- Name：标签控件的变量名称，为了方便编码，该名称应该具备清晰的意义。
- Tag：用于保存任何文字。
- Alignment：是枚举类型属性，用于指定标签在标签控件上的对齐方式，其中 Left 左对齐，Right 右对齐，Center 居中对齐。
- BoldSelectedText：指定当前标签页的标题(标签)是否以加粗方式显示，选中该复选框时加粗，否则与其他标签一样显示。
- CreateOnDemand：如果选中该复选框，那么在程序运行后选择某个标签时，相应

的标签页才创建，优点是程序运行后节省系统资源，否则在创建标签控件时就创建相应的标签页。

- FixedWidth：指定标签是固定宽度还是根据文字长短自动调整宽度。选中该复选框后，标签控件上所有标签的宽度都调整到文字最长标签的宽度，否则，各标签根据自身文字的长短调整宽度。
- FocusOnButtonDown：选中该复选框后，当单击选择某标签页后，相应的标签上出现一个由点组成的方框。
- Multiline：指定标签一行显示不下时的处理方式。选中该复选框后，当一行显示不下所有标签时，标签被多行显示；否则当一行显示不下所有标签时，标签控件的右上角将显示一个双箭头，可以通过单击箭头使被隐藏的标签出现在标签控件上。
- PerpendicularText：当该复选框被选中时，标签以竖条方式显示，这样每行能放置更多的标签，但标签页的显示区域却变小了；否则标签以横条方式显示。
- PictureOnRight：指定标签上图像的显示位置。选中该复选框时，图像放在标签文字的右边，否则放在左边。
- PowerTips：指定当把鼠标指针移动到标签上并稍微停留时是否显示该标签的简短提示，需要显示提示时，选中该复选框，并在标签页的属性对话框中设置相应的提示文字(在 PowerTipText 单行编辑框中设置)，通常在标签上没有文字只有图片时使用这项功能。
- RaggedRight：选中该复选框时，各标签的宽度由它的标题和 FixedWidth 选项确定；未选中该复选框时，系统自动调整各标签的宽度，使其沿着控件边沿填满整个标题区域。
- SelectedTab：指定标签控件初始显示的标签页的编号．各标签编号从左到右依次为 1，2，3，…。
- ShowPicture：指定标签上是否显示图片。选中该复选框时显示，否则不显示。
- ShowText：指定标签上是否显示文字。选中该复选框时显示，否则不显示。
- TabPosition：指定标签显示在标签控件上的位置，有效取值及其意义如表 6-4 所示。
- BackgroundColor：设置标签控件的背景颜色。

标签控件的上述属性除了能够直接在属性对话框中设置外，也可以通过编写代码在程序运行时予以修改。

2．标签页属性

每个标签页也都有一组属性，用于设置各标签页的外观和功能。在标签页属性对话框的 General 标签页中可以设置标签页的如下属性。

- Name：标签页的变量名，应该起个有意义的名字。
- TabText：显示在标签页标签上的文字，简明扼要是对该文字的基本要求。
- Tag：用于保存与标签页一起保存的任何字符串。
- PowerTipText：标签的提示文字。当标签控件的 PowerTips 属性设置为 True 后，

将鼠标指针移动到标签上并稍微停留时，该提示文字出现在鼠标指针的下方。

- BackgroundColor：设置标签页的背景颜色。
- Tab BackgroundColor：标签的背景颜色。
- TabTextColor：标签的文字颜色。

表 6-4　Tab Position 的取值及其意义

属性对话框中列表项	相应的枚举类型值	意　义
Top	TabsOnTop!	标签显示在标签控件顶部
Bottom	TabsOnBottom!	标签显示在标签控件底部
Left	TabsOnLeft!	标签显示在标签控件左部
Right	TabsOnRight!	标签显示在标签控件右部
Top and Bottom	TabsOnTopAndBottom!	所选标签及该标签前面的标签显示在标签控件的顶部，其他标签显示在标签控件的底部
Bottom and Top	TabsOnBottomAndTop!	所选标签及该标签后面的标签显示在标签控件的底部，其他标签显示在标签控件的顶部
Left and Right	TabsOnLeftAndRight!	所选标签及该标签前面的标签显示在标签控件的左部，其他标签显示在标签控件的右部
Right and Left	TabsOnRightAndLeft!	所选标签及该标签后面的标签显示在标签控件的右部，其他标签显示在标签控件的左部

3．在标签上显示图片

当需要在标签上显示图片时，除了在标签控件的属性对话框中选中 Show Picture 复选框外，还要在标签页属性对话框的 Pictures 选项卡中指定要在标签上显示的图片，方法如下：

(1) 单击 Pictures 选项卡，显示该标签页。

(2) 从 StockPictures 列表框中选择一个预定义图片或单击 Browse 按钮从磁盘上选择一个图片文件。

(3) 从 PictureMaskColor 列表项中选择图片的背景颜色。

4．用代码访问标签控件中的各种对象

从前面的介绍可以看出，标签控件分为几个层次：标签控件放置在窗口上，窗口是它的父对象；标签页放置在标签控件中，标签控件是标签页的父对象；而放置在标签页上的控件则以标签页为父对象。因此，访问标签控件上不同对象时，需要考虑这些层次，一般方法是一次或多次使用点操作符(.)。

5．动态创建与撤消标签页

除了在设计窗口时直接设计好标签控件的所有标签页外，也可以在程序运行时动态地创建和撤消标签页。要达到这一目的，设计程序时需要使用用户画板创建并保存用作标签页的定制可视用户对象，然后在程序中使用 OpenTab()或 OpenTabWithParm()函数动态地创

建标签页。当动态创建的标签页不再使用时，应该使用 CloseTab()函数及时地予以撤消。

注意： 不能使用 CloseTab()函数撤消(或称做关闭)窗口画板中创建的标签页，它只能撤消用 OpenTab()或 OpenTabWithParm()函数动态地创建标签页。

OpenTab()函数能够创建新的标签页，它有如下两种语法格式，分别为：

```
controlname.OpenTab ( userobjectvar1, index )
controlname.OpenTab ( userobjectvar2, userobjecttype, index )
```

语法 1 用于编写程序时已经确定哪个定制可视用户对象作为标签页；语法 2 用于程序运行后才能确定哪个定制可视用户对象作为标签页。这两个函数返回整数 1 时，表示执行成功；返回整数-1 时，表示执行失败。各参数的意义如下。

- tabcontrolname：标签控件名。
- userobjectvar1：要作为标签页使用的定制可视用户对象名，这个对象名既可以使用户对象画板保存该对象时的名称，也可以使用该用户对象数据类型说明的变量名。
- index：新创建的标签页将作为第几个标签显示在标签控件上。如果 index 为 0 或大于标签总数，那么新建标签页的标签将成为最后面的标签。
- userobjectvar2：类型为 DragObject 的变量，OpenTab()函数将把打开用户对象的引用存放在该变量中。
- userobjecttype：字符串或字符串变量，其值为要打开用户对象的名称。

用 CloseTab()函数关闭(撤消)由 OpenTab()或 OpenTabWithParm()函数打开的标签页，释放该标签页占用的资源，该函数的语法为：

```
tabcontrolname.CloseTab ( userobjectvar )
```

其中，tabcontrolname 是标签控件名；userobjectvar 是打开标签页时使用的用户对象名。程序中调用 CloseTab()函数后，从标签控件中删除相应的标签页，关闭该用户对象，并执行该标签页的 Destructor 事件处理程序，完成这些任务后，继续执行调用 CloseTab()函数语句后面的语句。

6. 其他常用标签控件对象函数

表 6-5 所示为标签控件常用的其他对象函数。

表 6-5 其他常用标签控件对象函数

函数名	参 数	用途
MoveTab()	source, destination	将由 source 指定的标签页移动到第 destination 个位置
OpenTabWithParm()	无	在打开标签页的同时传递参数
SelectTab()	tabidentifier	将指定的标签页设置为当前标签页
TabPostEvent()	event {, word, long }	将 event 消息放在每个标签页消息队列的尾部
TabTriggerEvent()	event {, word, long }	对每个标签页都触发 event 消息

6.7.4　事件

标签控件特有的事件主要有 SelectionChanging 和 SelectionChanged，下面分别介绍这两个事件的触发时机。

- SelectionChanging：当在选择新的标签页时触发。该事件有两个整型参数 oldindex 和 newindex。oldindex 是当前标签页的位置号，newindex 是新选择标签页的位置号。在该事件处理程序中，可以检查当前标签页的各种选项是否符合程序要求，如果不符合时，返回 1 将阻止转移到其他标签页；返回 0 允许选择新的标签页。
- SelectionChanged：当选择了新的标签页后触发。其参数与 SelectionChanging 事件相同。该事件通常用于初始化新选择的标签页，例如设置某些选项为数据窗口提取数据等。

6.8　OLE 控件

在开发软件时，需要访问其他数据库。例如应用程序中需要处理 Word 文档，那么把它集成到应用程序中就可以了。PowerBuilder 10.0 中的 OLE 控件正是为这个目的而设计的。借助 OLE 控件，既可以在应用程序中嵌入或者链接 OLE 服务器程序，也可以使用 OLE 定制控件(OCX)。下面介绍 OLE 控件的各种用法。

6.8.1　放入 OLE 控件

使用 OLE 控件之前，首先需要把它放置到窗口上，其操作步骤如下：

(1) 用 Window 画板打开放置 OLE 控件的窗口。

(2) 在 Window 画板工具栏中，单击控件图标列表中的 OLE 控件图标，打开 Insert Object 对话框，如图 6-32 所示。

该对话框有 3 个选项卡，每个选项卡对应的用途如下。

- Create New 选项卡为所定义的 OLE 控件选择一个服务器应用程序。
- Create From File 选项卡为所定义的 OLE 控件选择一个已存在的对象。
- Insert Control 选项卡为在 OLE 控件中插入一个 OCX 定制控件。

下面针对这 3 种用途分别说明其操作步骤。

1．选择一个服务器应用程序

选择一个服务器应用程序的步骤如下：

(1) 在图 6-32 所示的 Create New 选项卡下，Object Type 列表框中列出了当前计算机中已经登记的 OLE 服务器应用程序，单击选择所需的 OLE 服务器。

(2) 如果需要在 OLE 控件中显示服务器的图标，可以选中 Display As Icon 复选框，并单击 Browse 按钮为 OLE 服务器选择一个新的图标。

(3) 单击 OK 按钮，关闭对话框。

在窗口中需要放置 OLE 控件的位置上单击，启动 OLE 服务器，建立服务器的初始化数据。

选择 OLE 服务器的退出菜单，关闭服务器，此时 OLE 控件放置到窗口上并与所选的 OLE 服务器建立联系。

2. 链接或嵌入已存在对象

链接或嵌入已存在对象的步骤如下：

(1) 在图 6-32 所示的对话框中，单击 Create From File 选项卡标签，显示如图 6-33 所示的选项卡。

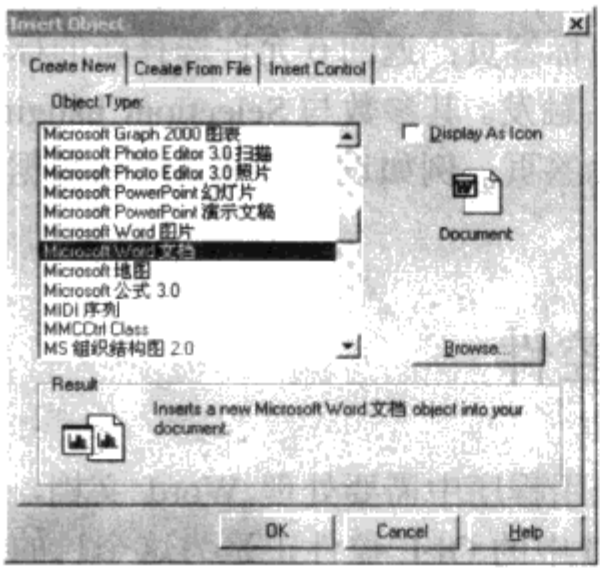


图 6-32 Create New 选项卡

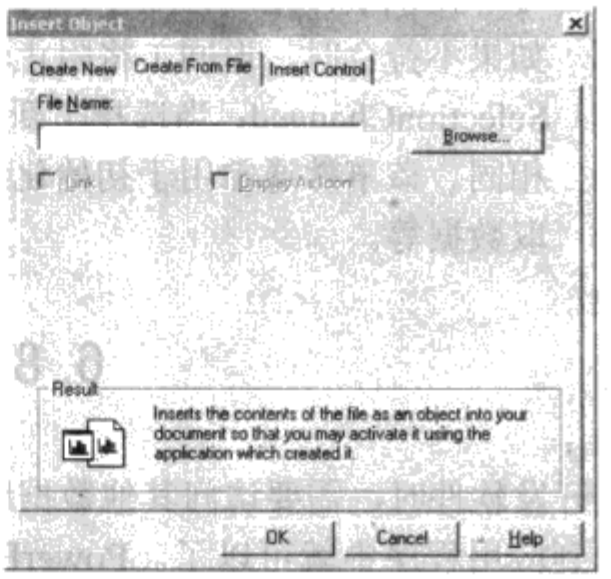


图 6-33 Create From File 选项卡

(2) 单击 Browse 按钮，从磁盘中选择一个文件(例如声音文件.wav、Word 文档.doc、图片文件.bmp)。

(3) 当选中 Link 复选框时，将建立指向所选文件的链接。这样，用其他方法修改所选文件后，修改结构将立即反映到 OLE 控件中；当未选中 Link 复选框时，系统将把所选文件嵌入到 OLE 控件中，其他程序对所选文件的修改不影响 OLE 控件的内容。

(4) 选中 Display As Icon 复选框，将在 OLE 控件中可以显示服务器的图标，单击 Change Icon 按钮可以改变显示的图标。

(5) 单击 OK 按钮关闭对话框。

3. 插入一个 OCX 定制控件

插入一个 OCX 定制控件的步骤如下：

(1) 在图 6-32 所示的对话框中，单击 Insert Control 标签，显示如图 6-34 所示的选项卡。

(2) 在 Control Type 列表框中，列出了所有已经注册的 OCX 定制控件，通过单击选择其中所需的定制控件。

(3) 查看所选定控件的信息，可以单击 Browse 按钮，打开 OLE Object Browse 对话框，显示该定制控件的类信息、属性、事件和函数，单击 OK 按钮关闭该对话框。

(4) 所需的定制控件还没有注册，可以单击

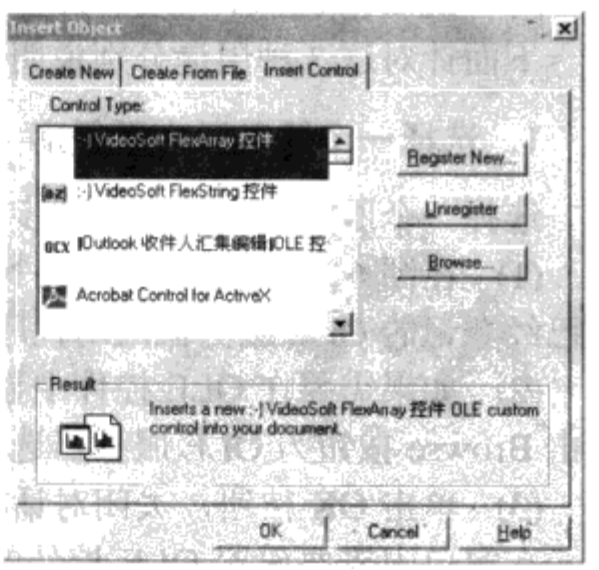


图 6-34 Insert Control 选项卡

Register New 按钮进行注册，之后按已注册定制控件操作。

(5) 单击 OK 按钮关闭对话框。

(6) 在窗口上需要放置 OLE 控件的位置上单击，该控件即放置到窗口上。

6.8.2 设置 OLE 控件

放置了 OLE 控件之后，可通过设置该控件的属性控制它的行为。对选择一个服务器应用程序链接或嵌入已存在对象这两种情况，OLE 控件的属性对话框完全相同。

1. 设置 OLE 控件的属性

双击 OLE 控件，打开 OLE Control 属性设置对话框，如图 6-35 所示。

其中各选项的意义如下：

- Name 编辑框指定 OLE 控件的变量名。
- DisplayName 编辑框中用于输入一些提示性文字供 OLE 服务器使用，有些 OLE 服务器会把该文字显示在窗口的标题栏上。
- 选中 FocusRectangle 复选框时，当 OLE 控件得到焦点后，将在 OLE 控件的周围显示一个由虚线组成的方框，以提示用户当前的焦点位置；否则，即使 OLE 控件得到焦点也不显示虚框。
- ContentsAllowed 下拉列表决定在运行时能够插入的对象类型，有效值如下。
  - Any：既可以插入一个链接对象也可以插入一个嵌入对象。
  - Embeded：可以嵌入一个对象。
  - Linked：可以链接一个对象。
- DisplayType 下拉列表框控制 OLE 控件内的显示内容，有效值如下。
  - Contents：在控件内以内容形式显示。
  - Icon：显示与数据相关的图标，通常该图标由服务器程序提供。

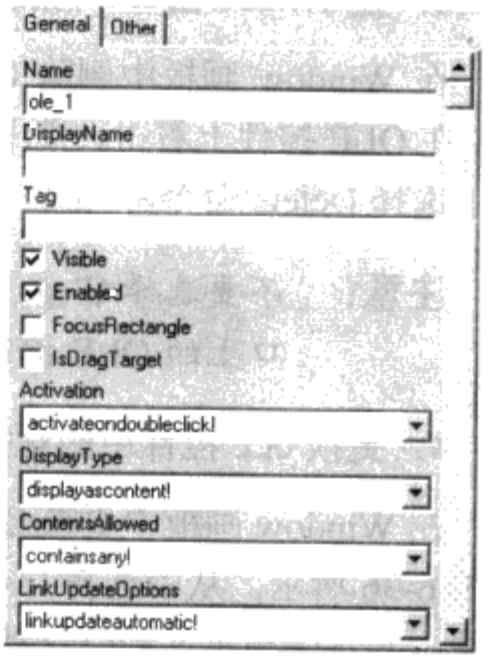


图 6-35　OLE Control 属性对话框

- LinkUpdate 下拉列表框指定在链接方式如何修改链接信息，有效值如下。
  - Automatic：如果由于某种原因链接被断开，找不到要链接的文件，自动显示对话框让用户重新定义链接。
  - Manual：如果由于某种原因链接被断开，则对象不能激活，此时只在程序中。
- Activation 下拉列表框用于定义 OLE 控件的激活方式，有效值如下。
  - DoubleClick：用户双击控件时激活服务器应用程序。
  - GetFocus：当 OLE 控件得到输入焦点(例如用户单击了该控件或用 Tab 键将焦点移到该控件)时激活服务器应用程序。
  - Manual：OLE 控件只能通过在程序中调用 Activate()函数来激活服务器应用程序。

- Border 下拉列表框用于指定 OLE 控件的边框类型，其中：
  - None：无边框。
  - Box：方框边框。
  - ShadowBox：阴影方框边框。
- Resize：运行时用户改变超文本编辑框的大小。
- Background Color 下拉列表框用于选择 OLE 控件的背景颜色。

2. 在 Window 画板中修改 OLE 控件中的对象

当在 Window 画板中需要修改 OLE 控件中的对象时，要启动相应的服务器来完成，操作步骤如下：

(1) 在 OLE 控件上右击，弹出快捷菜单，如图 6-36 所示。

(2) 选择 Open 命令，启动相应的服务器，用服务器提供的功能完成对象的修改，然后退出服务器。

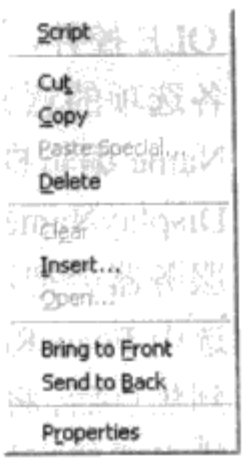


图 6-36 当右击 OLE 控件时弹出的菜单

3. 在 Window 画板中删除 OLE 控件中的对象

在 Window 画板中删除 OLE 控件中对象的方法很简单：在 OLE 控件上右击，弹出快捷菜单，如图 6-36 所示，从中选择 Delete 命令。

注意： 不要选择 Clear 命令，那样会删除放置到窗口上的 OLE 控件。

4. 更换 OLE 控件中的对象

在 Window 画板中更换 OLE 控件中对象的方法是在 OLE 控件上右击，显示弹出菜单，如图 6-36 所示。从中选择 Insert 菜单项，打开 Insert Object 对话框，然后指定一个所需的对象，其步骤与定义 OLE 控件时的步骤相似。

在代码中可以通过 OLE 控件的对象函数 InsertObject()、InsertFile()、InsertClass()或 LinkTo()在 OLE 控件中插入不同的 OLE 对象，也可以使用 Cut()或 Clear()函数清除 OLE 控件中的对象。

6.8.3 激活 OLE 对象的方法

利用 OLE 控件，PowerBuilder 应用程序能够把可以作为 OLE 服务器的应用程序(例如 Word、Excel 等)集成进来，进一步丰富应用程序的功能，简化开发工作。应用程序运行后，用户激活 OLE 控件中的对象有 Inplace 和 OffSite 两种方式。这两种方式的显示界面和菜单项位置各不相同，下面分别予以介绍。

1. InPlace 方式

采用 Inplace 激活方式时，服务器应用的工作界面显示在 OLE 控件内部，而且服务器应用的菜单和 PowerBuilder 应用程序的菜单合并在一起，看起来就像应用程序的一部分一

样。图 6-37 所示为 OLE 控件以 InPlace 方式激活 Word 时显示的界面。

当以 InPlace 方式激活 OLE 控件中的对象时，OLE 控件的周围会有一个阴影边框。

图 6-37　InPlace 激活方式示例

2. OffSite 方式

当采用 OffSite 激活方式时，OLE 控件中的对象将作为一个文档显示在服务器应用程序的窗口，且服务器应用程序的所有菜单均提供给用户。

在代码中使用 Activate()函数指定激活方式并激活 OLE 对象。例如把 OLE 控件中的 Activation 属性设置为 Manual，可以在该 OLE 控件中的 DoubleClicked 事件中用以下代码以 OffSite 方式激活对象：

```
ole_1.Activate(OffSite!)
```

也可以使用以下代码以 InPlace 方式激活对象：

```
ole_1.Activate(InPlace!)
```

6.8.4　使用 OLE 访问 Excel 数据

本例介绍如何通过 OLE 对象实现把数据窗口中的文件生成 Excel 文件。该实例主要使用的函数有创建 OLE 对象(Create OLE Object)、连接对象(Connect To New Object)和打开目标文件(Application.Workbooks.Open)等。

创建一个 PowerBuilder 生成 Excel 文件的窗口，目的是作为数据窗口的载体，使数据生成工作通过命令按钮完成。为了完成整个功能，需要建立 Workspace、Target、Application、一个数据窗口和一个窗口。

该窗口主要难点在于如何把数据窗口中的文件生成 Excel 文件的程序编写，以及窗口事件——生成数据按钮功能(OLE 对象应用)的实现。

1. 建立应用

在 PowerBuilder 10.0 中建立 Workspace、Target 和 Application。在 Application 中设置数据源、连接数据库，并打开主窗口。代码如下：

```
// Profile Powersoft Demo DB V6
SQLCA.DBMS = ODBC
SQLCA.Database = Powersoft Demo DB V6
SQLCA.AutoCommit = False
SQLCA.DBParm = ConnectString='DSN=Powersoft Demo DB V6;UID=dba;PWD=sql'
connect using sqlca;
if sqlca.sqlcode <> 0 then
 MessageBox (Cannot Connect to Database, sqlca.sqlerrtext)
 return
end if
Open(w_Excel_loop)
```

2. 建立数据窗口

前面没有介绍数据窗口的设计，下一章将详细介绍数据窗口的概念及设计方法，本节只是简单介绍采用表格风格、外部数据源的数据窗口格式设计。

在工具栏中单击 New 图标，在弹出的 New 对话框中选择 DataWindow 标签。在该选项卡中选择 Grid，即表格风格的显示风格。

在数据源——Data Source 选项中选择 External。在出现的面板中定义字段和类型等。单击 Add 按钮增加字段，单击 Delete 按钮删除字段，单击 Insert 按钮在当前字段之前插入字段。

接着，在设置数据窗口的颜色和边界的选项卡里，设置背景颜色；文本(Text，字段名称)和数据项(Column)的边界均设置为无边框式(NoBorder)。

在数据窗口设计画板中，调整各个字段的显示属性。数据窗口(d_excel)设计的具体样式如图 6-38 所示。

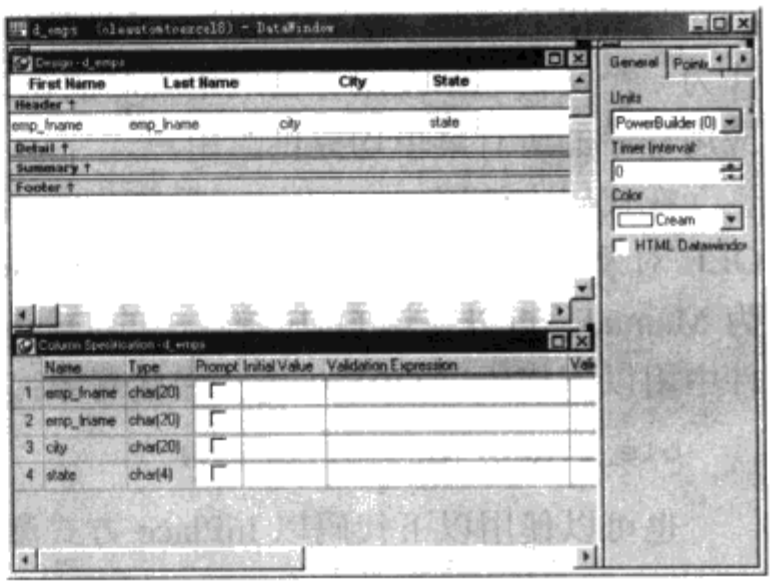


图 6-38 数据窗口(d_emps)的设计效果

3. 建立窗体

在工具栏中单击 New 图标，在弹出的窗口中选择 PB Object 标签。在其中选择 Window——窗体，单击 OK 按钮，打开该窗体的数据画板。命名该窗体为 w_Excel_loop，把标题设定为 PB to Excel。

其中包括两个命令按钮、两个静态文本框、两个单行编辑框和一个数据窗口控件，其名称和属性如下：

```
//数据窗口：dw_1
//命令按钮：cb_1、cb_2。
cb_1.text=Copy DW Data to Excel。
cb_2.text=Close。
//静态文本框：st_1、st_2。
st_1.text=Column No.。
st_2.text=Row   No.。
//单行编辑框：sle_1、sle_2。
```

编写窗体的 Open 事件脚本，代码如下：

```
dw_1.SettransObject(SQLCA)
dw_1.Retrieve()
// column No. in dw_1
sle_1.text = string(dw_1.Object.DataWindow.Column.Count)
// row No.in dw_1
sle_2.text = string(dw_1.RowCount())
```

下面编写控件的脚本，单击 Copy DW Data to Excel 按钮，将通过 OLE 对象实现把数据窗口中的文件生成 Excel 文件的功能。具体代码如下：

```
long numcols , numrows , c, r
OLEObject xlapp , xlsub
```

```
int ret

numcols = long(dw_1.Object.DataWindow.Column.Count)
numrows = dw_1.RowCount()

//创建 OLE 对象变量 xlapp
xlApp = Create OLEObject

//连接到 Excel 并检查返回代码
ret = xlApp.ConnectToNewObject(Excel.Sheet )
if ret < 0  then
    MessageBox(Connect to Excel Failed ! ,string(ret))
   return
end if

//打开一个指定的 Excel 文件
xlApp.Application.Workbooks.Open(c:\file.xls) //,false,true

//显示 Excel 文件
xlApp.Application.Visible = true

xlsub = xlapp.Application.ActiveWorkbook.Worksheets[1]

// for/next 循环复制所有行
For c = 1 to numcols
   For r = 1 to numrows
     xlsub.cells[r,c] = dw_1.object.data[r,c]
   Next
Next
For c = 1 to numcols
   xlsub.cells[numrows + 1,c] ='好! '
Next
```

```
xlApp.DisConnectObject()
Destroy xlapp
```

至此，完成创建一个在 PowerBuilder 中实现生成 Excel 文件的窗体实例。运行该程序后，主界面效果如图 6-39 所示。

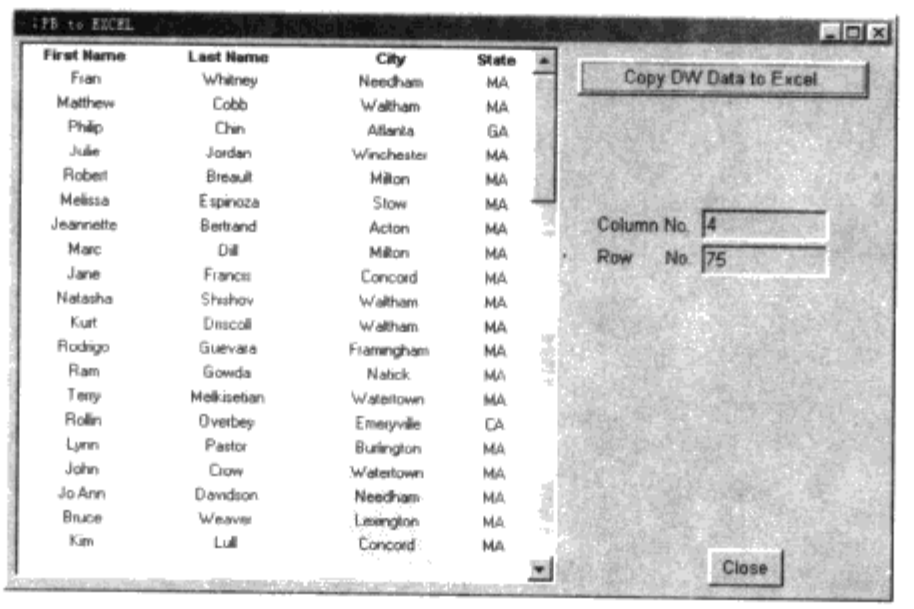


图 6-39　PB 与 Excel 通信实例的主界面

单击 Copy DW Data to Excel 按钮，将打开 Microsoft Excel，把数据窗口的数据拷入该 Excel 新文件，实现 PowerBuilder 生成 Excel 文件的功能。

6.9 习 题

1. 填空题

(1) _____是标准的 Windows 按钮，只显示提示按钮功能的文字。

(2) _____常用于显示提示信息或说明没有标题控件的作用等。

(3) 多行文本框使用的常用事件有_____、_____、_____，其触发时机和用法与单行文本框相同。

(4) PowerBuilder 提供了 4 个控件用于完成列表选择，包括：_____、_____、_____、_____。

(5) 图片框有两个重要事件，分别是_____和_____事件。

2. 问答题

(1) 如何设计带滚动条的图形控件？

(2) 试根据 6.2.6 节设计记事本程序。

(3) 如何设置 OLE 控件？如何激活 OLE 对象？

第7章　菜单技术

除了简单的情况外，绝大多数应用程序都使用菜单，菜单已经成为商业化程序的标准界面元素。菜单能够把程序各功能有效地组织起来，在有限的屏幕空间中分组展示应用程序的功能。

菜单是各种操作的集合，将各种操作进行分类。本章主要讲述如何创建菜单对象、设置属性以及编写事件处理程序。

7.1　菜单概述

菜单是由一系列命令或选项(菜单项)构成的，通过选项发出命令，指示程序完成用户指定的任务。除了子窗体和响应窗体外，其他类型的窗体都可以拥有菜单。菜单也是对象，在应用设计过程中，随时都可以创建和修改菜单。本节介绍创建与修改菜单的方法以及菜单对象的命名协议。

下面首先说明菜单的基本概念。菜单有 3 种类型，分别为下拉菜单、弹出式菜单和级联菜单。

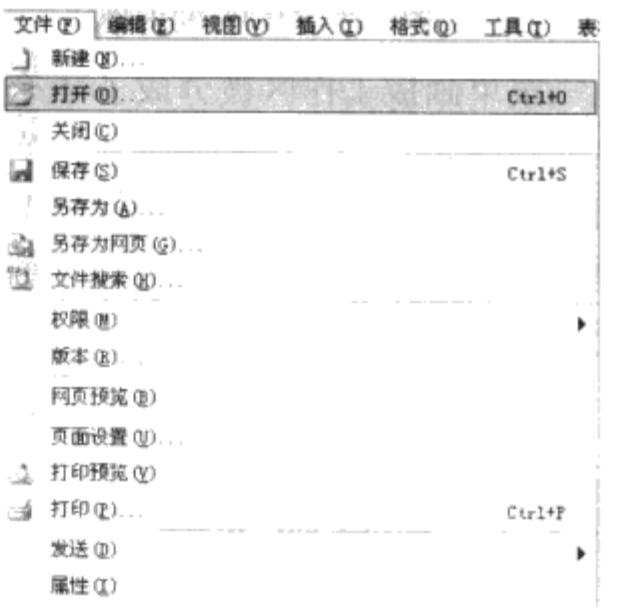


图 7-1　下拉菜单

下拉菜单有菜单标题、菜单项组成，如图 7-1 所示。用户选择菜单标题后，该标题下的菜单项即被弹出。

弹出式菜单与对象相关联，通常又称为上下文相关菜单。图 7-2 是一个弹出式菜单的示例，当右击某个对象时，出现相应的弹出式菜单。

级联菜单可以出现在前两种菜单中，在其父菜单项后面有个箭头符号，指示该菜单项后面有级联菜单，如图 7-3 所示。级联菜单的层次不宜超过两层。

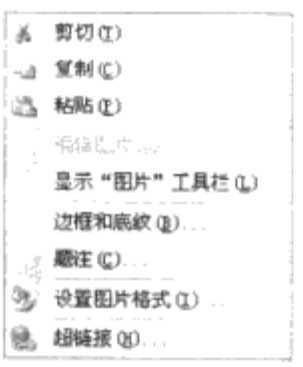


图 7-2　弹出式菜单

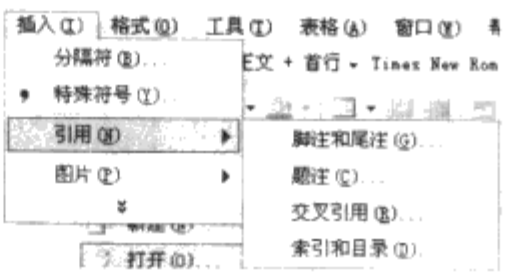


图 7-3　级联菜单

使用菜单画板创建和修改菜单，启动菜单画板的步骤如下：

(1)　单击 File 菜单中的 New 选项，弹出 New 对话框。

(2) 打开 PB Object 选项卡，如图 7-4 所示。选中 Menu(菜单)图标，单击 OK 按钮。即可进入菜单画板工作区，如图 7-5 所示。

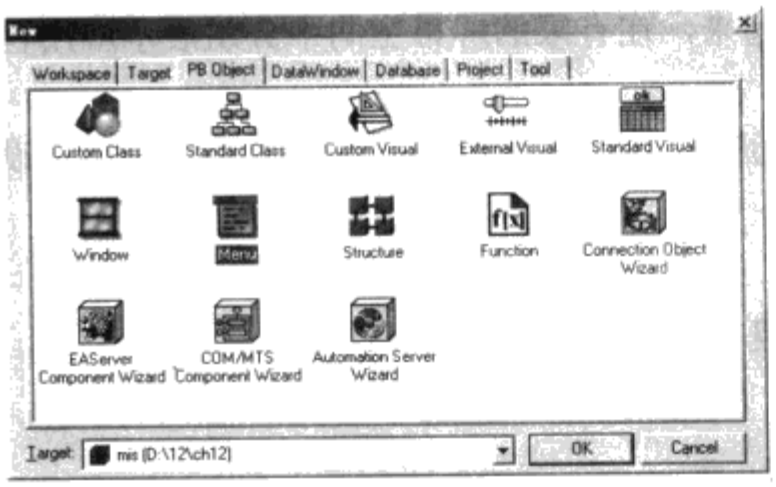


图 7-4 PB Object 选项卡

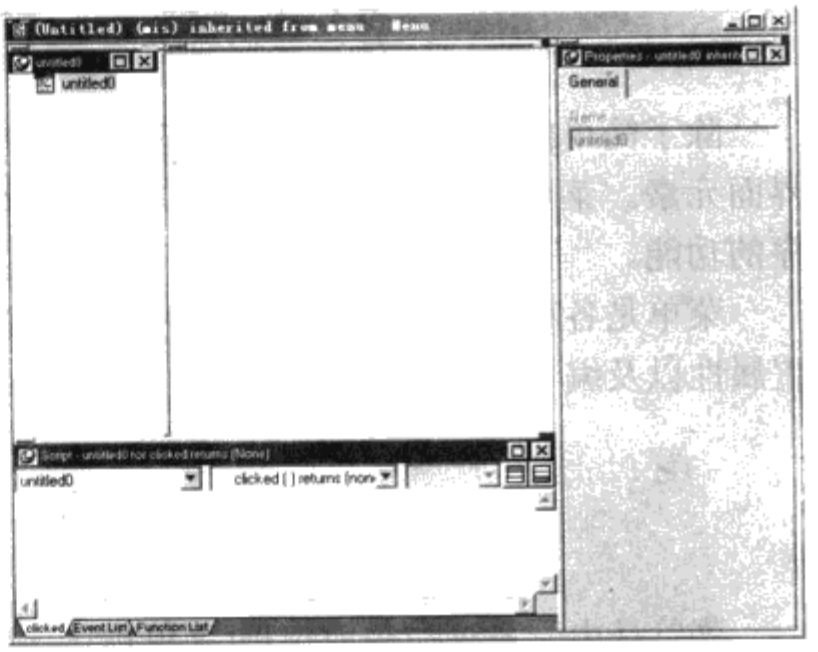


图 7-5 菜单画板工作区

菜单画板工作区被分成了 4 个部分，菜单栏、菜单项或级联菜单以及菜单项属性表。这 4 个部分作为一个整体使用，当单击菜单栏上的某个菜单标题时，该菜单标题下的级联菜单就显示在菜单项或级联菜单工作区中，而单击菜单项或级联菜单工作区中的一个菜单项时，它的属性就显示在菜单项属性表中。

7.2 菜 单 操 作

7.2.1 创建新菜单

创建新菜单的基本步骤如下：

(1) 按照上一节的方法，进入菜单画板工作区，如图 7-6 所示。

(2) 单击 Insert 菜单下的 Submenu Item 选项，如图 7-7 所示。

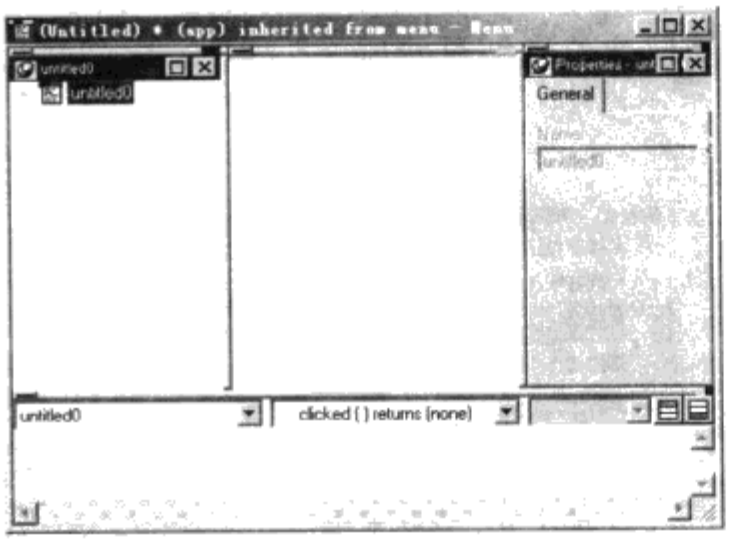


图 7-6 菜单画板工作区

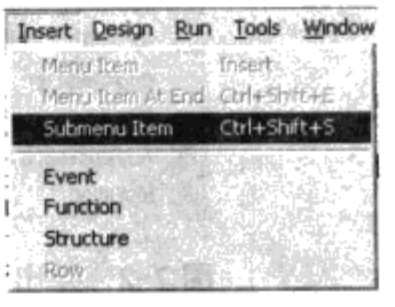


图 7-7 选择 Submenu Item 选项

(3) 这时在画板工作区中显示一个编辑框，在此编辑框中输入菜单项，例如这里输入

File，按 Enter 键，显示如图 7-8 所示的编辑结果界面。

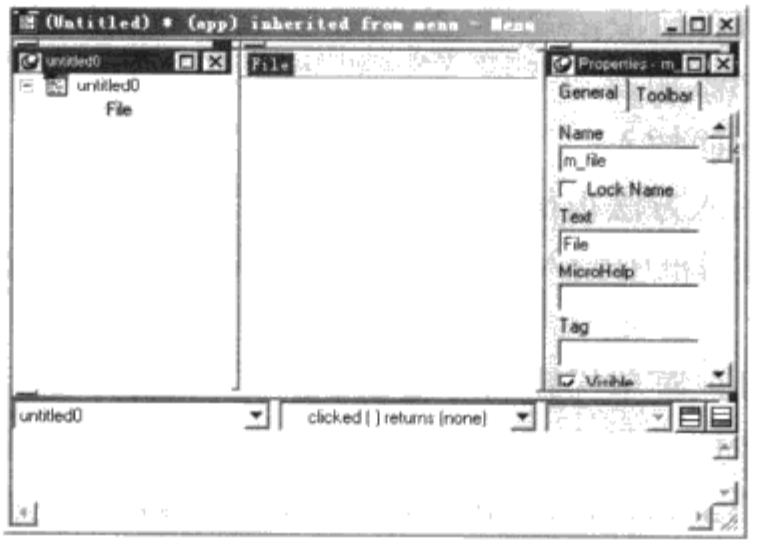


图 7-8　编辑结果

(4)　当创建了一级菜单项时，首先选中 File 菜单，再单击 Insert 菜单的 SubmenuItem 选项，为 File 菜单添加子菜单，例如添加 New 子菜单，添加完后的结果如图 7-9 所示。

(5)　按照以上的方法可以再添加一级菜单和子菜单，甚至可以为子菜单再添加下一级菜单。

(6)　定义好所有菜单项后，选择 File 菜单中的 Save 命令，打开 Save Menu 对话框，如图 7-10 所示。输入菜单名称和注解之后，单击 OK 按钮保存菜单。

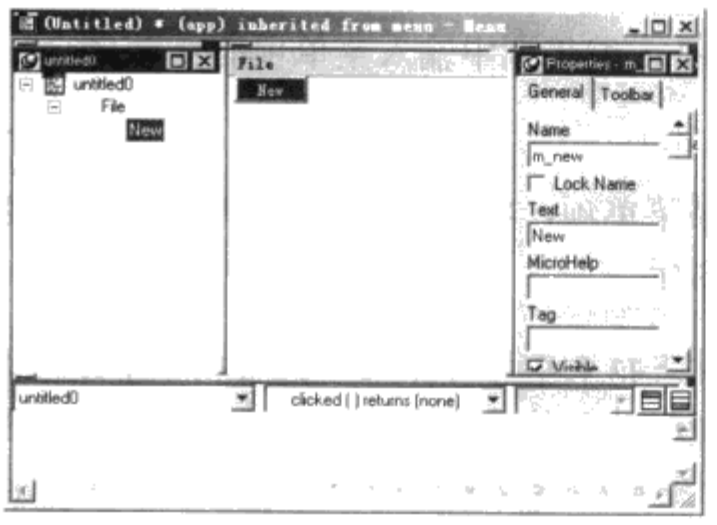


图 7-9　添加子菜单

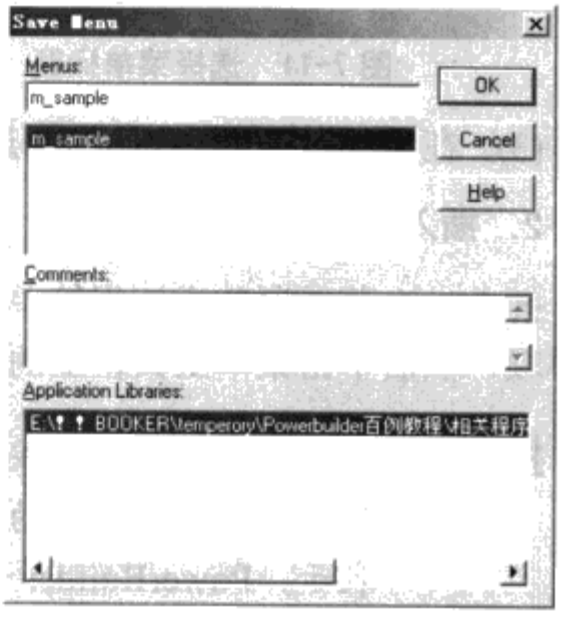


图 7-10　保存菜单

注意：　菜单对象使用标识符命名协议，一般情况下，菜单项的名称能够提示其功能，菜单项的名称以 m_开头，以便于与应用程序中的其他对象相区分。

7.2.2　修改菜单

当原有的菜单不能满足应用程序的需求时，就需要修改菜单对象的定义。菜单对象的修改包括菜单标题的插入和菜单项的插入、删除、移动等内容。下面分别介绍具体的方法。

1. 插入菜单标题

插入菜单标题分为两种情况：

- 在菜单标题尾部添加。
- 在某菜单标题前面插入。

对于第一种情况，插入方法很简单，单击菜单栏最右边菜单后边的空白位置，即显示一个新的编辑框，在该编辑框中输入菜单标题，在 Menu For 下的编辑框中依次输入各菜单项即可。

对于第二种情况，插入菜单标题的步骤如下：

(1) 选中将在其前面插入菜单中的某个菜单标题，如图 7-11 所示。

(2) 单击菜单工具栏上的(插入)图标，插入一个编辑框，在该编辑框中输入菜单标题，如图 7-12 所示。

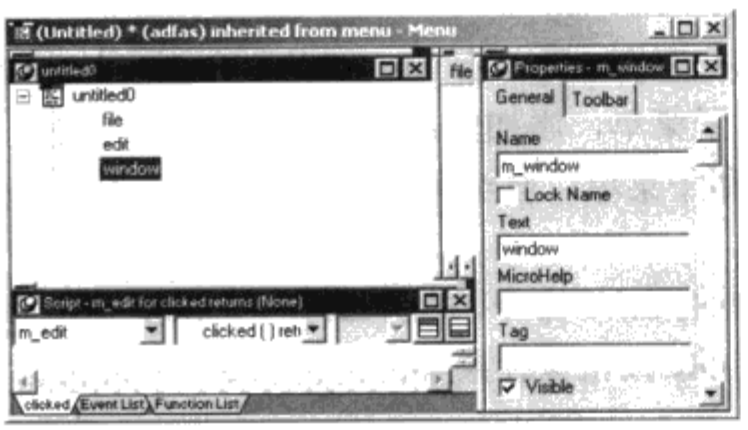


图 7-11 选择菜单标题

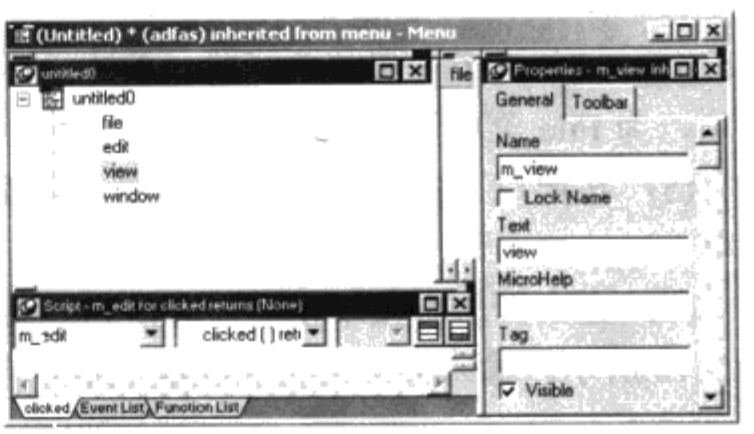


图 7-12 插入菜单标题

(3) 在属性对话框中 General 选项卡下的 Name 编辑框中输入菜单项的变量名。

2. 插入菜单项

插入一个菜单项的方法很简单。首先单击某菜单项，将在该菜单项前面插入一个新的菜单项，然后单击菜单工具栏上的插入(Insert)图标。

技巧： 可以使用分隔线，按功能的异同对下拉菜单进行分类。要增加一条分隔线，只需在放置分隔线的菜单项前面增加一个菜单项，该菜单项的标题用减号(-)表示，PowerBuilder 会自动将这个字符解释成分隔线。

3. 删除菜单项

删除菜单项的方法很简单，首先单击要删除菜单项的名称，然后单击菜单工具栏上的删除(Delete)图标，该菜单项即被删除。

注意： 删除菜单标题时一定要小心，系统将把菜单标题所属的菜单项、级联菜单以及相应的事件处理程序一同删除，且在删除时没有给出提示信息。

4. 移动菜单项

菜单项只能在同级菜单中移动，其移动方法如下：

(1) 单击菜单工具栏上的移动(Move)图标。

(2)　在要移动的菜单对象上按住鼠标左键不放，此时鼠标指针变成手形指针。

(3)　拖动菜单对象到菜单中的一个新位置，然后放开鼠标左键，菜单项即被移动到新位置。

7.2.3　设置菜单属性

菜单和菜单项都是对象，它们也有一组确定其特征与行为的属性。实际上，菜单项的快捷键与图标是常用的两类属性，除此之外，菜单对象还有其他属性，例如在初始状态下，菜单项是否显示、是否有效，菜单项前面是否显示选中符号(P)等。通过设置属性，能够有效地、明确地控制菜单行为与外观，下面介绍菜单属性的设置方法以及这些属性的用途。

1．定义菜单项的快捷键

目前，几乎所有的菜单系统都支持快捷键。快捷键的作用是在不需要打开菜单的情况下，只按下快捷键，就可以实现快捷键对应的菜单项的功能，避免重复打开菜单。定义了菜单项的快捷键后，会自动在菜单项的右部显示快捷键提示，如 File 菜单的 New 菜单项后提示 Ctrl+N。

注意：　快捷键只能在菜单画板中定义，而不能在程序中利用代码进行改变。

在菜单画板中定义菜单项快捷键的步骤为：

(1)　单击要定义快捷键的菜单项，这时手形图标指向该菜单项。

(2)　选择菜单属性窗体上的 General 选项卡，显示如图 7-13 所示的对话框。

(3)　在 Shortcut Key 下拉列表框中选择一个按键，然后这个键会自动出现在左部菜单项旁边的第二列编辑框中，运行时它会显示在菜单项文本的右边，这就是该菜单项对应的快捷键，None 表示不定义快捷键。

(4)　根据需要，选中 Alt、Ctrl、Shift 复选框中的一个或多个，然后这个组合键将显示在菜单设计区中的第二列编辑框中。例如在 Key 列表框中选择字母 A，然后选中 Ctrl、Shift 复选框，这时将以组合键 Ctrl+Shift+A 作为当前菜单项的快捷键。

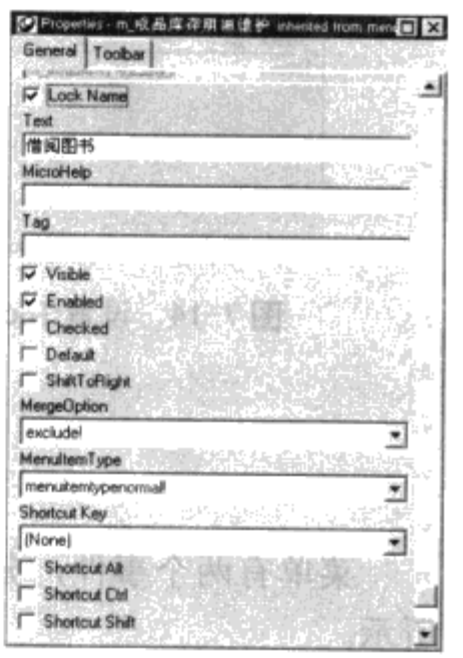


图 7-13　设置快捷键

注意：

(1) 同一个菜单中，作为快捷键使用的组合键应当惟一，否则重复的快捷键将不起作用。另外，选择快捷键时尽量与标准用法一致，例如在绝大多数应用程序中，使用 Ctrl+C 作为“复制”菜单项的快捷键、Ctrl+V 作为“粘贴”菜单项的快捷键等。

(2) 避免下述按键作为快捷键：单个字母(比如 B)，否则程序运行时该字母将不能输入到单行或多行编辑框中；Alt+字母，这样的组合键通常作为访问键；Ctrl+Alt+Del 通常作为重启系统或关闭应用程序。

2. 定义菜单工具栏

对 MDI 风格的应用程序来说，可以通过设置菜单项工具栏的属性定义工具栏。当把菜单添加到 MDI 框架窗体或其子窗体上后，与工具栏所关联的窗体被打开时，工具栏就自动出现在框架窗体上。可以修改工具栏，例如调整工具栏位置、设置是否显示简短提示等。所有这些功能都是 PowerBuilder 内置的，无需编写任何程序。

下面讲述如何定义菜单项工具栏属性，具体步骤如下：

(1) 在菜单画板工作区中添加一个新的菜单项，然后在菜单属性设置区中单击 Toolbar 选项卡，显示如图 7-14 所示的属性页。

(2) 单击 ToolbarItemName 下拉列表框，从中选择需要的工具按钮。如选择 Bold!，添加完的效果如图 7-15 所示。

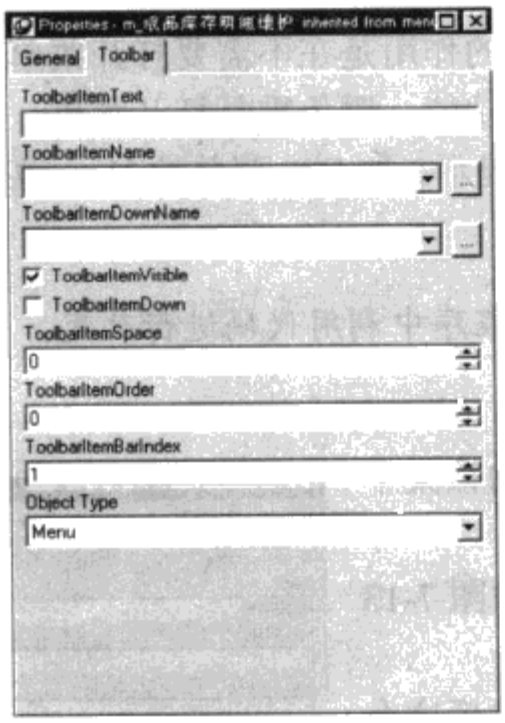


图 7-14 设置 ToolBar 属性

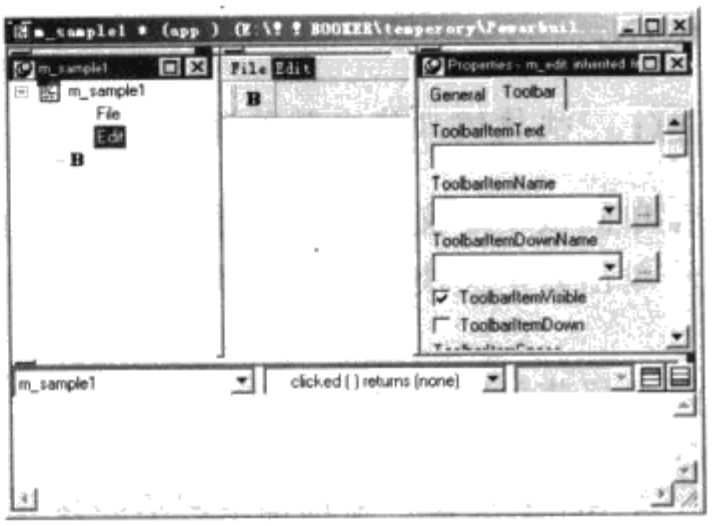


图 7-15 添加结果

7.3 菜 单 事 件

菜单有两个事件，分别为 Clicked 事件和 Selected 事件。事件编写脚本窗体如图 7-16 所示。

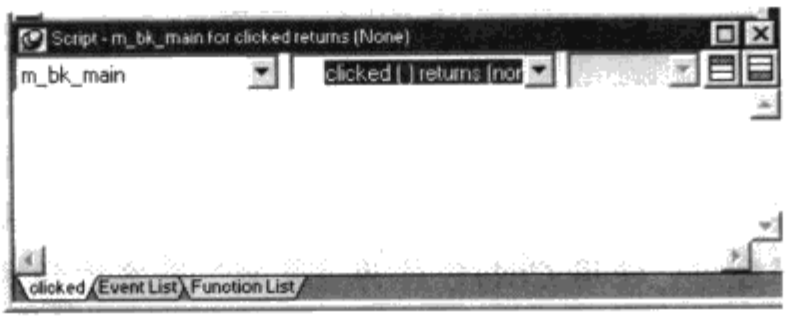


图 7-16 事件编写脚本窗体

一般来说，菜单中的每个菜单项都需要对 Clicked 事件进行编程，以响应用户选择的操作，而 Selected 事件用于显示提示信息，以说明菜单项的作用。菜单编程实际上就是响应

用户及系统对菜单项的操作，为了编写出通用代码，必须使用关键字 ParentWindow。

7.3.1　菜单事件触发时机

Clicked 事件触发时机如下：

- 单击菜单对象。
- 用键盘选中菜单对象，然后按回车(Enter)键。
- 按菜单的快捷键。

菜单对象被显示在屏幕上后，按访问键即可。

注意： 只有菜单对象的 Visible 和 Enabled 属性均为 True 时，菜单对象才会响应单击或键盘操作。

Selected 事件只有在菜单对象被选中时，才会触发。

通常对每个菜单项的 Clicked 事件都要编写事件处理程序，但对于以下菜单项，一般不编写事件处理程序：

- 菜单栏上的菜单标题，通常菜单标题用于列出下拉菜单。
- 弹出级联菜单的菜单项。

7.3.2　编写菜单事件处理程序

菜单事件的处理程序也在代码编辑器中编写。编写菜单事件处理程序的步骤如下：

(1) 进入菜单画板，通过单击选择要编程的菜单项。
(2) 单击菜单画板工具栏上的编程图标，启动代码编辑器。
(3) 在事件列表框中选择要编程的事件。
(4) 键入所需代码。

7.4　菜单设计实例

7.4.1　制作弹出式菜单

弹出式菜单可以提供一个上下文相关的操作环境，丰富了应用程序界面的表达能力。一般来说，通过右击可以激活弹出式菜单，因此需要在对象的 RbuttonDown(右击)事件中编写激活弹出式菜单的代码，下面介绍如何制作弹出式菜单。

1.　弹出窗体上的菜单

在实际开发时，往往需要把窗体上某菜单标题的下拉菜单作为弹出式菜单显示。这样，既可以使用窗体上的菜单项完成操作，也可以使用弹出式菜单完成操作。实现上述要求的方法很简单，只要在某对象或某控件的 RButtonDown 事件中编写如下语句：

```
m_main.m_title.PopMenu(PointerX(), PointerY())
```

其中，m_main 是放置在窗体上的菜单名，m_title 是某个菜单标题，该标题下的菜单项

将构成弹出式菜单。

PopMenu()显示弹出式菜单的菜单对象函数，它在参数指定的位置上显示弹出式菜单。使用 PointerX()、PointerY()函数得到右击的位置。

制作一个与窗体菜单栏中的任何菜单都不同的弹出菜单，其步骤如下：

(1) 编辑与该窗体相关的菜单，例如菜单名称为 m_menu。

(2) 在菜单栏的最右端创建一个新的菜单标题，用空格作为该标题的提示。

(3) 在 Menu Item Name 编辑框中输入该菜单标题的变量名，例如变量名为 m_pop。

(4) 建立该菜单标题下的各菜单项。

(5) 保存该菜单对象。

(6) 在要被显示的弹出式菜单对象的 RButtonDown 事件中，输入如下代码。

```
m_menu.m_pop.Visible=True
m_menu.m_pop.PopMenu(PointerX(), PointerY())
m_menu.m_pop.Visible=False
```

注意： 用自己的菜单对象名和标题为空格的菜单标题对象名代替 m_menu 和 m_pop。

当右击被定义了弹出式菜单的对象后，弹出式菜单即可被显示。

将菜单标题设置为空格且放到菜单栏的最右边的目的是为了在菜单栏上看不到该菜单标题。使用

```
m_menu.m_pop.Visible=True
```

语句可以使该菜单可见，否则就不能显示弹出式菜单。

2. 弹出其他菜单

应用程序中也可以通过编写代码把尚未与窗体相关联的菜单作为弹出式菜单使用。实际上，在用菜单画板创建菜单时，同时创建了一个菜单对象和一个名称相同的菜单类型。当把菜单关联到窗体时，实际连接的是菜单对象。由于菜单未曾与窗体关联，因此要显示弹出式菜单，首先必须创建菜单对象，其方法是(例如已经创建了一个名称为 m_main 的菜单)：

```
m_main m_new_main  //说明菜单变量
m_new_main= Create m_main  //创建菜单实例
```

现在，就可以像窗体菜单那样使用 m_new_main 菜单了：

```
m_new_main.m_pop.PopMenu(PointerX(), PointerY())
```

7.4.2 为主窗口配置菜单

本例的任务是创建一个主窗口并为它配置菜单，使用户在主菜单(或子菜单)上可以看到并选择所需要的功能模块入口，而且可以在此选择退出该系统。

在应用系统中，都会有一个主窗口，而此窗口主要以显示菜单为主要功能，通过菜单来对系统进行控制。

该应用程序的主窗口设计为使用菜单进行控制。需要使它的层次结构简洁、清晰，使用户容易操作。主窗口菜单中包含的菜单项有：“成品库库存信息管理”、“零件库库存信息管理”、“公共信息”和“退出”。其中，“成品库库存信息管理”和“零件库库存信息管理”还有子菜单项。

下面将分层进行详细分析，使读者掌握主窗口与菜单相结合的设计方法和技巧。分两步来完成需要做的工作：

第一步，制作菜单；

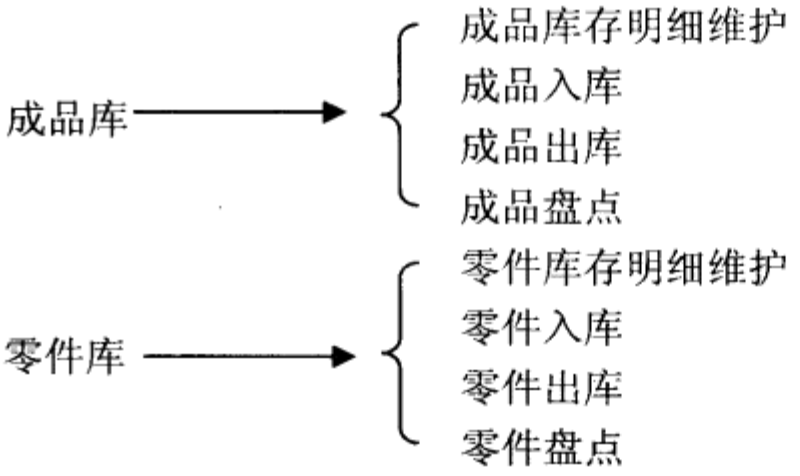
第二步，建立主窗体并加载菜单。

另外还要在这两步中编写窗体和菜单的各个事件的脚本。下面具体说明各个步骤。

1. 制作菜单

(1) 首先要设计出菜单项的内容(主菜单和子菜单)。

菜单包含的菜单项有：“成品库库存信息管理”、“零件库库存信息管理”、“公共信息”和“退出”。其中“成品库库存信息管理”和“零件库库存信息管理”还包括下一级菜单项，如下所示：



下面将开始菜单的制作过程。

(2) 在 PowerBuilder 的工具栏中单击图标 New，在随后弹出的窗口中选择 PB Object 标签。在其中选择 Menu 菜单，单击 OK 按钮。然后，PowerBuilder 会打开菜单的设计画板。

(3) 右击 untitled0 选择 Insert Submenu Item 插入子菜单项。在此菜单项的光标处填入菜单项的名称“成品库库存信息管理”。重复此过程，加入菜单项“零件库库存信息管理”、“公共信息”和“退出”，完成一级菜单项的制作。

(4) 右击“成品库库存信息管理”选择 Insert Submenu Item 插入下一级子菜单项：“成品库存明细维护”、“成品入库”、“成品出库”、“零件盘点”。同样方法，加入“零件库库存信息管理”的下一级子菜单项。

注意： 在右击某一菜单项的时候，选项中会有 Insert bmenu Item 和 Insert Submenu Item 选项。它们的区别在于：前者是添加与此菜单项同一级的菜单项；后者是添加比此菜单项低一级的菜单项。注意根据需要选择。

(5) 双击各个菜单项，加入打开相应窗口的脚本。如“成品库存明细维护”的脚本为：

```
open(w_cpk , parent)
```

(6) 给该菜单命名为 m_main，保存。至此完成了菜单的制作。制作好的菜单效果如图 7-17 所示。

图 7-17 菜单 m_main 的制作窗口

2. 建立主窗体并加载菜单

(1) 建立主窗体。

在 PowerBuilder 的工具栏中单击 New 图标，在随后弹出的窗口中选择 PB Object 标签。在其中选择 Window 窗体，单击 OK 按钮。然后，PowerBuilder 会打开窗体的数据画板。给该窗体命名为 w_main，把菜单的标题设为“库存信息管理系统”。

(2) 设置控件，并设置其属性。

在该窗体中设置一个静态文本框：st_1.text=“您好！欢迎进入库存信息管理系统！”设置和排列情况，以及各自的属性效果具体如图 7-18 所示。

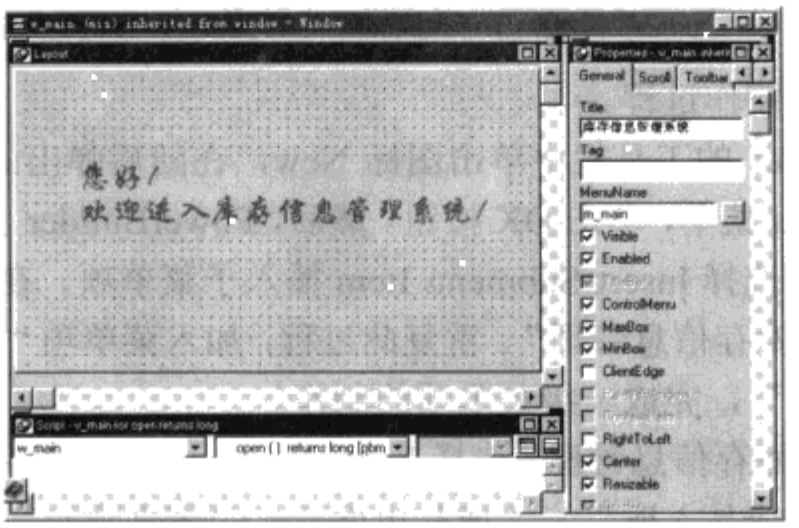


图 7-18 设置窗体 w_main 及控件的效果

至此，完成了建立主窗口并为它配置菜单的所有工作，实际中可以根据具体的需要设置和改变各个控件的属性。

完成上述工作，就得到了一个应用程序的主窗口和为它配置的菜单，我们可以从这里看到整个系统的大致结构。具体的运行效果如图 7-19 和图 7-20 所示。图 7-19 显示的是成品库库存信息管理的简要结构。图 7-20 显示的是零件库库存信息管理的结构，从这里可以进入任何一个零件库库存信息管理的功能操作窗口。

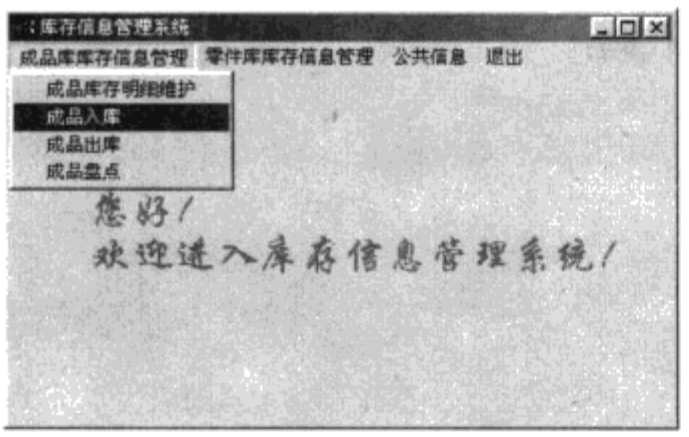


图 7-19　主窗口及菜单的实例效果 1

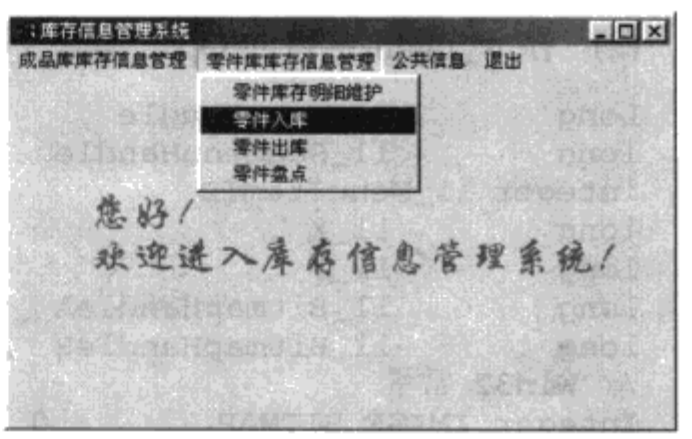


图 7-20　主窗口及菜单的实例效果 2

在窗口的菜单里选择所需要的功能模块入口，即可进入该模块的窗口界面，进行工作。而且可以在此窗口的菜单项中选择退出该系统。

技巧： 在应用程序中应考虑到用户工作的需要，经常采用柔和的颜色和界面，这也是提高软件的友好程度的一个方面，希望引起读者的注意。

主窗口是系统的总体结构框架的体现，需要将该部分的各个窗口或者各项功能意义有条理地列举在菜单中。这样，会方便用户查找所需要的窗口或者功能进入相应的部分进行操作。因此主窗口制作的要求是：简洁、清晰、条理清楚。为了美观起见，在主窗口和菜单项中添加图形标志不失为一种很好的方法，但需要注意的是，图形务必实用且简单精练，以免喧宾夺主，影响正常的操作。

7.4.3　设计图形菜单

本例将制作一个图形菜单。运行后，在弹出的“图形菜单”窗口中，单击菜单条上的“图形菜单 1”或“图形菜单 2”，都会出现由图形显示的菜单。本例演示了 PowerBuilder 10.0 中的图形菜单技术，在许多程序和软件中，常常运用图形菜单技术使程序和软件的界面变得更加友好。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 Menu 菜单，保存为 m_mymenu。

(4) 建立 w_main 窗口，Title 设置为“图形菜单”，将 MenuName 属性设置为 m_mymenu。

2. 设置 Menu 菜单

(1) 建立菜单基本框架。如图 7-21 所示，建立 m_main 菜单的基本框架。

(2) 保存菜单。

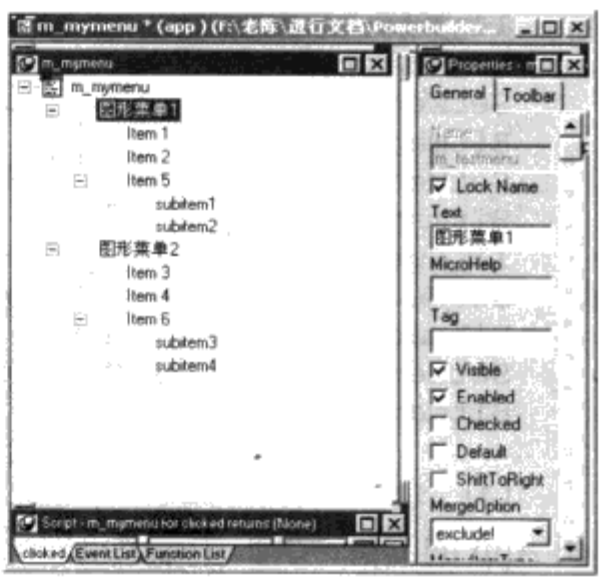


图 7-21　m_mymenu 菜单的基本框架

3. 编写窗口事件驱动代码

(1) 定义本地扩展函数。

(2) 在 w_main 窗口的 Open 事件中输入如下代码：

```
Long        ll_MainHandle
long        ll_SubMenuHandle
integer li_MenuItemID
long        ll_X
long        ll_Y
long        ll_BitmapHandleA
long        ll_BitmapHandleB
// Win32 常量
Integer IMAGE_BITMAP      = 0
Integer LR_LOADFROMFILE = 16
Integer SM_CXMENUCHECK  = 71
Integer SM_CYMENUCHECK  = 72
Integer MF_BITMAP           = 4
Integer MF_BYCOMMAND        = 0
Integer MF_BYPOSITION   = 1024
// 获取菜单句柄
ll_MainHandle = GetMenu(Handle(this))
//获取第一个菜单的句柄
ll_SubMenuHandle = GetSubMenu(ll_MainHandle,0)
//以原始大小装入图片
ll_BitmapHandleA = LoadImageA(0,'1.bmp',0,0,0,LR_LOADFROMFILE)
ll_BitmapHandleB = LoadImageA(0,'2.bmp',0,0,0,LR_LOADFROMFILE)
li_MenuItemID = GetMenuItemID(ll_SubMenuHandle,0)
ModifyMenu(ll_SubMenuHandle,li_MenuItemID,MF_BITMAP,li_MenuItemId,ll_Bit
mapHandleA)
li_MenuItemID = GetMenuItemID(ll_SubMenuHandle,1)
ModifyMenu(ll_SubMenuHandle,li_MenuItemID,MF_BITMAP,li_MenuItemId,ll_Bit
mapHandleB)
ll_SubMenuHandle = GetSubMenu(ll_SubMenuHandle,2)
li_MenuItemID = GetMenuItemID(ll_SubMenuHandle,0)
ModifyMenu(ll_SubMenuHandle,li_MenuItemID,MF_BITMAP,li_MenuItemId,ll_Bit
mapHandleA)
li_MenuItemID = GetMenuItemID(ll_SubMenuHandle,1)
ModifyMenu(ll_SubMenuHandle,li_MenuItemID,MF_BITMAP,li_MenuItemId,ll_Bit
mapHandleB)
ll_SubMenuHandle = GetSubMenu(ll_MainHandle,1)
ll_x = GetSystemMetrics(SM_CXMENUCHECK)
ll_y = GetSystemMetrics(SM_CYMENUCHECK)
ll_BitmapHandleA = LoadImageA(0,'1.bmp',  IMAGE_BITMAP,ll_x,ll_y,
LR_LOADFROMFILE)
ll_BitmapHandleB = LoadImageA(0,'2.bmp',IMAGE_BITMAP,ll_x,ll_y,
LR_LOADFROMFILE)
SetMenuItemBitmaps(ll_SubMenuHandle,0,MF_BYPOSITION,ll_BitmapHandleA,ll_
BitmapHandleB)
SetMenuItemBitmaps(ll_SubMenuHandle,1,MF_BYPOSITION,ll_BitmapHandleB,ll_
BitmapHandleA)
ll_SubMenuHandle = GetSubMenu(ll_SubMenuHandle,2)
SetMenuItemBitmaps(ll_SubMenuHandle,0,MF_BYPOSITION,ll_BitmapHandleA,ll_
BitmapHandleB)
SetMenuItemBitmaps(ll_SubMenuHandle,1,MF_BYPOSITION,ll_BitmapHandleB,ll_
BitmapHandleA)
```

(3) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下代码：

```
Open(w_main)
```

单击 PowerBuilder 10.0 开发界面上的 按钮或按 Ctrl+R 快捷键，程序就可以运行。运

行效果如图 7-22 所示。

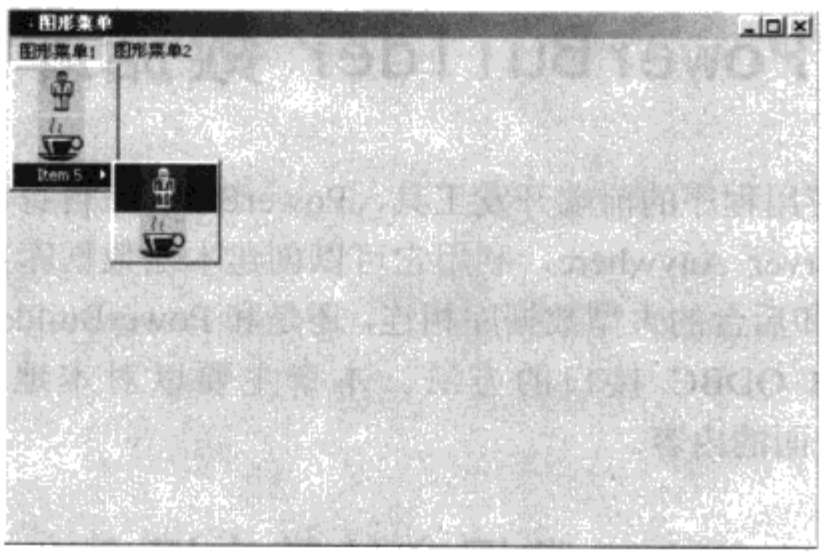


图 7-22　运行结果

7.5　习　　题

1．填空题

(1)　菜单有 3 种类型，分别为_____、_____、_____。

(2)　菜单画板工作区被分成了 4 个部分，包括：_____、_____、_____、_____。

(3)　菜单有两个事件，分别为_____和_____。

(4)　菜单编程实际上就是响应用户及系统对菜单项的操作，为了编写出通用代码，必须使用关键字_____。

2．问答题

(1)　如何制作弹出式菜单？

(2)　如何为主窗口配置菜单？

(3)　如何设计图形菜单？

第 8 章　PowerBuilder 数据库应用基础

作为一个数据库应用程序的前端开发工具，PowerBuilder 自身也带有一个本地数据库管理系统 Adaptive Server Anywhere，利用它可以创建本地数据库并对其进行操作。对于 PowerBuilder，无论是和后台的大型数据库相连，还是和 PowerBuilder 自建的本地数据库相连，都是采用标准的 ODBC 接口的方法。本章主要以对本地数据库的操作来认识 PowerBuilder 数据库方面的内容。

8.1　数据库的基本概念

作为应用系统的核心和管理对象，数据库就是一组相关数据的集合，这组集合以一定的组织方式将相关的数据组织在一起，存放在计算机存储器上，同时能为多个用户共享，并且与应用程序是彼此独立的。现在的开发工作所涉及的数据库大多都是关系型数据库，关系型数据库是一个二维表，由行和列组成。与传统概念不同的是，这里所说的数据库可以包括多个表，如公司数据库除了有员工信息表，还可以有工资发放表。表 8-1 所示为某公司的员工信息表。

表 8-1　员工信息

工作证号	姓名	性别	出生年月	部门	职称	电话	备注
001	李达	男	1963/8/6	办公室	工程师	62275386	
002	张红	女	1952/2/23	市场部	工程师	62278574	
003	王维	男	1965/7/8	维修部	高工	62273458	
004	赵晓华	女	1979/8/10	财务部	会计	62271209	

下面以表 8-1 为例介绍几个与数据库相关的概念。

1.　列(Column)

表中的列称为字段，列的名称叫作字段名，字段用来描述表的结构。如员工信息表就是由“工作证号”、“姓名”、“性别”等 8 个字段构成的。

2.　行(Row)

表中的行叫作记录，是组成表的基本数据。在员工信息表中，一个记录包含了一个人的信息。每个记录(行)中对应列的内容称作字段的值，例如：第 1 个记录中“姓名”对应字段的值是“李达”，第 2 个记录中“工作证号”字段的值是“002”。

3.　数据类型(Type)

数据类型是指字段的取值范围。字段必须具有数据类型，如字符型、整数型、小数型和日期型等。在表 8-1 中字段“姓名”的数据类型是字符型，字段“出生年月”的数据类

型是日期型。

4. 主键(Primary Key)

主键又称为关键字，它是表中的一个字段或者是由几个字段构成的字段组。主键的值惟一确定了其他字段的值，也就是说表中任意两行主键的值都不相同。假定公司每个员工的工作证号是惟一的，在表 8-1 中，“工作证号”字段就是一个主键，它可以惟一确定其他字段的值，因为一个人只有一个工作证号。但“姓名”字段就不可以作为主键，因为公司中可以有同名同姓的两个或多个人存在，知道了员工的姓名，并不能确定该员工的“工作证号”、“工作部门”等其他信息。

注意： 一个表中只允许有一个主键，主键可以防止表中行与行之间的重复。

5. 外键(Foreign Key)

如果存在 A、B 两个表，当表 A 中的一个或多个字段是表 B 的主键时，称这个或多个字段为 A 表的外键。外键表明某个表中的记录同其他表中记录的连接关系。这种关系可以是一对一的，也可以是一对多的。下面的例子就是一对多的连接关系。

假设公司内部还有一个部门信息表，如表 8-2 所示。

表 8-2　部门信息表

部门	部门主管	主管房间
市场部	李意	2104
财务部	周良	1025
办公室	胡卫	1024
维修部	李远	3102

在部门信息表中，由“部门”字段可以惟一确定“部门主管”及“主管房间”等字段的值，它是部门信息表的主键。员工信息和部门信息两个表可以通过字段“部门”联接成一个新表，如表 8-3 所示。

表 8-3　员工信息表和部门信息表连接构成的新表

工作证号	姓名	性别	出生日期	部门	职称	电话	部门	部门主管
001	李达	男	1963/8/6	办公室	工程师	62275386	市场部	李意
002	张红	女	1952/2/23	市场部	工程师	62278574	维修部	李远
003	赵晓华	女	1979/8/10	财务部	会计	62271209	财务部	李远
004	王维	男	1965/7/8	维修部	高工	62273458	维修部	李远

8.2　PowerBulider 的数据库接口

数据库管理系统(DBMS)是管理数据库的工具，是为数据库的建立、使用和维护而配置的软件。它们建立在操作系统上，实现对数据库统一的管理和控制。PowerBuilder 可以和

多个 DBMS 相联，也可以同时使用这些 DBMS。

根据操作系统的不同，在 PowerBuilder 开发环境中与数据库连接一般有以下两种方法：

- 标准数据库接口(如 ODBC、JDBC 和 OLE DB)。
- 专用数据库接口。

下面分别对这两种方法进行介绍。

8.2.1 标准数据库接口

标准数据库接口通过适应标准的驱动程序(ODBC 和 JDBC)和数据供应器(OLE DB)与某数据库相连，它们将标准 API(应用编程接口)定义的抽象函数调用翻译成某特定数据库可理解的调用。为了使用标准接口，一般需要安装标准的 API 和相应的驱动程序或数据供应器，然后再从 PowerBuilder 安装程序中选择安装来访问该 DBMS 的标准数据库接口。

PowerBuilder 现在支持的标准接口有：

- 开放式数据库连接(ODBC)。
- Java 数据库连接(JDBC)。
- Microsoft 的通用数据访问组件 OLE DB。

ODBC 和 JDBC 可以访问基于 SQL 的数据库，OLE DB 可以访问基于 SQL 和非基于 SQL 的数据管理。

PowerBuilder 本身自带了一个独立的 Adaptive Server Anywhere 的数据库 EAS Demo DB。因为数据库 EAS Demo DB 使用的是 Adaptive Server Anywhere 的 ODBC 驱动程序，所以认为它是一个 ODBC 数据源。

1. ODBC 接口

ODBC(Open Database Connectivity，开放式数据库连接)是微软公司开发的标准应用编程接口(API)。它允许单个应用程序(使用 SQL 语言作为标准数据访问语言)访问多个 ODBC 兼容的数据源。

PowerBuilder 应用程序提供了 ODBC 接口，它可以访问有 ODBC 驱动程序的数据源。ODBC 驱动程序是一个实现 ODBC 函数调用的动态链接库(DLL)或共享库。

2. JDBC 接口

JDBC(Java Database Connectivity，Java 数据库连接)是一个标准的应用编程接口，它允许 Java 应用程序连接到支持 SQL 语言的数据库。JDBC API 包含公共 SQL 数据库活动的类，从而可以打开与数据库的连接，并可以使用 SQL 命令处理检索结果。因此，Java 程序可以使用常用的 SQL 语言设计模式来处理结果数据。

3. OLE DB 接口

OLE DB 是微软公司开发的标准应用编程接口，是数据访问软件中的组件。OLE DB 允许应用程序访问那些有 OLE DB 数据供应器的数据。它提供了统一的应用程序来访问以不同格式保存的数据。

8.2.2　专用数据库接口

专用数据库接口就是在 PowerBuilder 中直接与数据库连接的接口。每个专用数据库接口使用它自己的接口 DLL 或共享库，通过厂家指定的数据库 API 与指定数据库连接。专用数据库接口提供了多种数据库和 DBMS 的专用连接。

8.3　数据库画板

8.3.1　数据库画板窗口

单击集成环境工具栏上的 Database 图标，打开数据库画板，如图 8-1 所示。在 PowerBuilder 中进行的与数据库相关的工作都可以在数据库画板中实现。数据库画板的窗口功能如表 8-4 所示。

表 8-4　数据库画板的窗口功能

窗口名称	描　述
Object	列出了数据库接口和简介；对于活动的数据库连接，还列出了表、列、主键和外键、索引、用户、窗口、驱动信息以及其他信息，显示的信息由数据库和用户权限决定
Object Layout	显示表的图形表示
Activity Log	显示当前执行的操作的 SQL 语法
Columns	用于创建或修改表中的列
Object Details	显示对象的属性：有些对象的属性是只读的，有些对象的属性是可以修改的
Extended Attributes	列出了连接数据库里定义的显示格式、编辑风格以及有效性规则
ISQL Session	用于建立、执行或解释 SQL

图 8-1　数据库画板

8.3.2　数据库画板功能

在数据库画板中，大部分工作开始于 Object 窗口。有些工作只需将对象拖放到不同的

窗口中就可以完成。所有的工作都可以使用命令按钮来完成，也可以使用主菜单或者弹出菜单中的菜单项命令完成。下面列出了要实现的任务以及相应任务的操作简介，如表 8-5 所示。

表 8-5 任务以及完成任务的操作

任　务	操　作
修改数据库特征内容	选择要修改的数据库特征项，通过 Object 菜单或者在弹出菜单中的 Properties 选项或使用 Properties 按钮
连接到数据库	选择该数据库，从 File 菜单里选择 Connect，或者从弹出菜单里选择 Connect；使用 File	Recent Connection 菜单，可以选择已进行过的连接进行连接；数据库连接可以通过单击 Database Profile 按钮来建立
创建一个新的表、视图、列、键和索引	选择数据库对象，从 Object 菜单或者在弹出菜单中选择 New 命令，或者使用 Create 按钮
修改数据库对象	将对象拖到 Object Details 窗口中
图形化显示表	从 Object 窗口中将表的图标拖到 Object Layout 窗口，或者选择该表并从 Object 菜单或者在弹出菜单中选择 Add To Layout
操作数据	选择表并从 Object	Data 菜单或者在弹出菜单的 Edit Data 项中选择 Grid、Tabular 或者 FreeForm，或者使用适当的数据操作按钮
建立、执行或者解释 SQL	在 Interactive SQL 窗口中建立 SQL 语句；使用 Paste SQL 按钮将 SELECT、INSERT、UPDATE 和 DELETE 语句复制到窗口的工作区。要执行或者解释 SQL 语句，可以从 Design 菜单或者在弹出菜单中选择 Execute SQL 或者 Explain SQL
定义或者修改扩展属性	从 Object	Insert 菜单中选择要定义或者要修改的扩展属性的类型或在 Extended Attributes 窗口的列表里选择扩展属性，并从弹出菜单中选择 New 或者 Properties 选项
操作数据库工具	在 Objects 窗口里双击一个工具
记录工作日志	从菜单栏中选择 Design	tart Log 命令；如果要查看生成的 SQL 语句，可显示 Activity Log 窗口

8.3.3 修改数据库选项

要修改数据库选项，可以从菜单栏中选择 Design | Options 命令。在数据库选项窗口中，有些选项是为了连接特定数据库设置的，其他选项则是为了数据库画板服务的，如图 8-2 所示。

在 General 选项卡中，Shared Database Profiles、Connect To Default Profile、Keep Connection Open、Use Extended Attributes 和 Read Only 选项用于设置特定数据库的连接。

Columns in Table Display：当 PowerBuilder 图形化显示表时，最多可以同时显示 8 个列中的数据，也可以自定义显示数据列数。

SQL Terminator Character：使用分号作为 SQL 语句的终结符，当然也可以自定义 SQL

语句终结符。

Refresh Table List：当第一次显示表中的列时，PowerBuilder 从数据库中检索表的列并显示；为了节省时间，PowerBuilder 在系统内部将把这个列保存起来，当再次检索到这个表的列时，该列不必重新产生。该属性用来确定表的列在系统内部的保存时间，通常每隔 30min 刷新一次，也可以自定义刷新时间。

其他的几个选项卡可以自定义数据库画板的颜色、文字大小等信息。

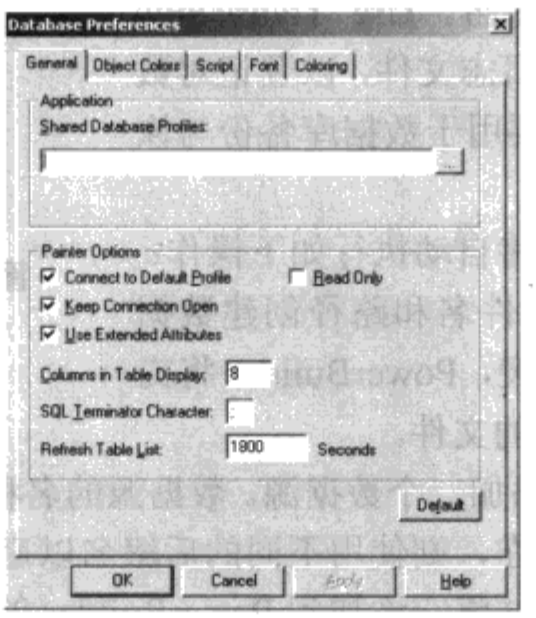


图 8-2　数据库选项窗口

8.4　创建和删除数据库

一般来说，PowerBuilder 访问数据库前，需要使用相应的数据库管理系统(比如 Oracle、Sybase 等)创建自己的数据库(通常需要数据库管理员在服务器上完成创建工作)，然后通过 ODBC 接口或专用接口把 PowerBuilder 或用户开发的应用程序连接到数据库上。PowerBuilder 内置的数据库管理系统 Adaptive Server Anywhere 有些特殊，它能够在 PowerBuilder 中创建和删除本地的 Adaptive Server Anywhere 数据库，这样避免将用户的数据与 PowerBuilder 的实例数据混为一谈。下面讨论建立和删除本地 Adaptive Server Anywhere 数据库的方法。

8.4.1　创建数据库

创建本地 Adaptive Server Anywhere 数据库的操作步骤如下：

(1) 单击工具栏的 Database 画板图标，进入数据库画板工作窗口。

(2) Object 窗口提供了 ODBC 接口的 Create ASA Database 工具。首先在 Database Profiles 窗口中展开 ODBC 的 Utilities 项，双击其中的 Create ASA Database 选项，弹出 Create Adaptive Server Anywhere Database 对话框，如图 8-3 所示。

(3) 在 User ID 编辑框中键入数据库用户名，默认值为 DBA。

(4) 在 Password 编辑框中键入保密字，默认值为 SQL。

(5) 在 Database Name 编辑框里定义新建的数据库的名字和所在路径。如果没有提供

文件的扩展名，默认扩展名为 DB。

(6) 选中 Prompt For Password During Connect 复选框，在 PowerBuilder 连接到新建的数据库时提示键入用户名和保密字，否则不提示。

(7) 定义数据库的其他属性。如果使用了非英文的数据库，需要在 Collation Sequence 列表框里选择一项用来支持非英文数据的存储。Use Transaction Log 复选框用于指示是否生成日志文件，以登记对数据库的所有操作，日志文件主要用于数据库备份与恢复。

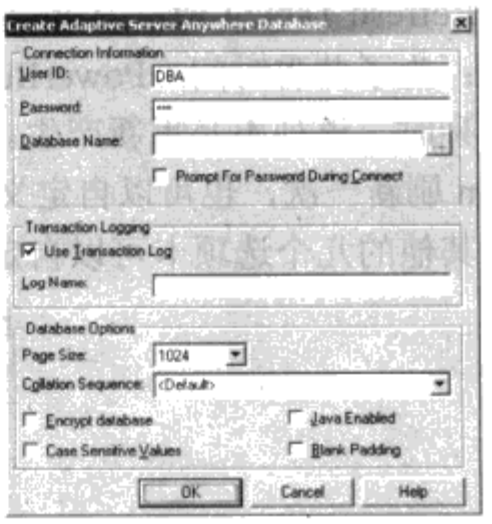


图 8-3 CreateAdaptive Server Anywhere Database 对话框

(8) 单击 OK 按钮结束，将自动执行如下操作：

- 使用定义的数据库文件名和路径创建数据库。如果存在同名冲突，PowerBuilder 将提示用户是否覆盖原来的文件。
- 在 ODBC.ini 文件中添加一个数据源。数据源的名称与数据库名称相同，如果相同名称的数据源已经存在，则使用不同的后缀名以避免冲突。
- 创建一个数据库简介并将它添加到 PowerBuilder 的初始化文件中。数据库简介具有与数据库相同的名称，如果存在相同名称的冲突，可以使用不同的后缀名以避免冲突。

8.4.2 删除数据库

删除 Adaptive Server Anywhere 数据库的步骤如下：

(1) 打开数据库画板。

(2) 首先在 Database Profiles 窗口中展开 ODBC 的 Utilities 项，双击其中的 Delete ASA Database 选项，弹出 Delete Local Database 对话框，如图 8-4 所示。

(3) 选择要删除的数据库并单击“打开”按钮。

(4) 询问是否要删除，单击“是”按钮后删除指定的数据库，同时还将删除该数据库相关描述文件。

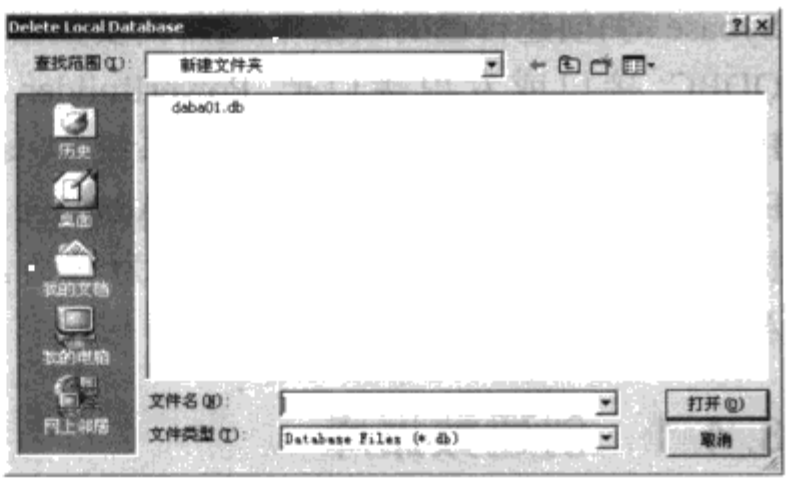


图 8-4 Delete Local Database 对话框

8.5 连接数据库

8.5.1 数据库连接简介

使用数据窗口访问数据库的一般步骤如下：

(1) 连接需要操作的数据库。

(2) 创建数据窗口对象，定义数据的显示风格、数据源，并定义数据的输入/输出格式、编辑风格。

(3) 在窗口中建立一个数据窗口控件。

(4) 通过数据窗口控件的属性设置将数据窗口对象和数据窗口控件连接起来。

(5) 在窗口中为数据窗口控件编写代码，完成对数据库中数据的访问。

下面介绍其中的一些主要步骤。

要建立一个数据库窗口对象，首先必须同需要访问的数据库进行连接，下面以 PowerBuilder 自带的数据库(数据源名为 EAS Demo DB V9)为辅助向读者介绍如何同数据库建立连接。

同数据库连接有如下两种方法：

单击 DB Profile 快捷按钮，在出现的 Database Profiles 对话框中选择要连接的数据库，如图 8-5 所示，单击 Connect 按钮进行连接。

单击 Database 快捷按钮，在弹出的 Database 中选取要连接的数据库，如图 8-6 所示，单击鼠标右键，在弹出的菜单中选取 Connect 菜单项进行连接。

连接完数据库后，按照前面的描述来创建数据窗口对象，定义数据的显示风格、数据源，并定义数据的输入/输出格式、编辑风格。

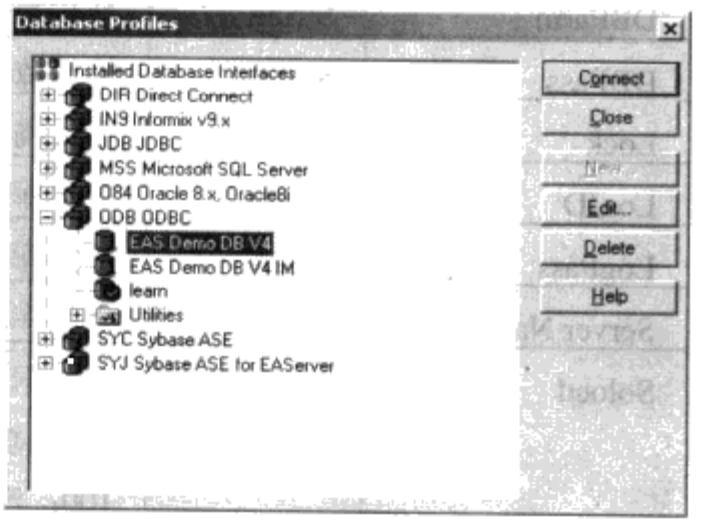


图 8-5　Database Profiles 对话框

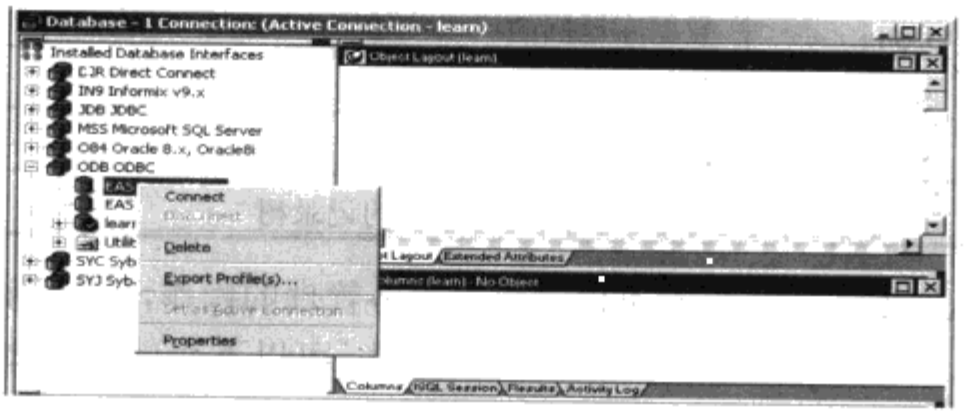


图 8-6　Database 窗口中的管理数据库界面

8.5.2　使用事务对象

1.　事务对象概念及属性

在数据库中，事务是指一组逻辑操作单元，使数据从一种状态变换到另一种状态。为确保数据库中数据的一致性，数据的操纵应当是离散的、成组的逻辑单元。当它全部完成时，可以保持数据的一致性，而当这个单元中的一部分操作失败，整个事务应全部视为错误操作，所有从起始点以后的操作应全部回退到开始状态。

PowerBuilder 提供了一个事务对象 Transaction 来建立与数据库的连接。Transaction 对象提供了两类属性，一类用于设置数据库连接参数，另一类用于给用户返回信息。

Transaction 的主要属性如表 8-6 所示。

表 8-6 Transaction 对象的各种属性

属性名	数据类型	取值范围和定义
AutoCommit	Boolean	True：每次执行 SQL 语句后自动提交事务 False：每次执行 SQL 语句后自动提交 一般来说，这个属性设置为 False，由程序员自己来控制事务的执行
Database	String	连接的数据库名
DBMS	String	连接数据库的 PowerBuilder 驱动引擎标识
DBParm	String	连接数据库所需要的连接参数，不同类型数据库的参数都不同
DBPass	String	连接数据库所需要的数据库口令
Lock	String	事务隔离级别
LogID	String	数据库服务器登录名
LogPass	String	数据库服务器登录口令
Server Name	String	数据库服务器名
Soloed	Long	对数据库操作的返回值 0：成功 100：没有发现记录 -1：错误，使用 SQLDBCode 和 SQLErrText 属性可以得到更详细的信息
SQLDBCode	Long	数据库返回的错误信号
SQLErrText	String	数据库返回的错误信息
UserID	String	连接数据库的用户名
SQLNRows	Long	对数据库操作所影响的记录树

事务对象的 AutoCommit 属性可以简化对事务管理的操作，这一属性可以用 True 或 False 对其赋值。当设置 AutoCommit 属性为 True 时，可以不通过其他额外的交互就将 SQL 语句传输给后台数据库，而且执行完毕后自动提交事务。设置 AutoCommit 属性为 False(默认值)时，使用 COMMIT 或 ROLLBACK 这样的关键词提交或回退事务。

在大多数应用中，一部分的数据库操作是要成组提交的，而另一部分则不需要。可以利用 AutoCommit 属性的特性来确定事务的起点，当把 AutoCommit 属性设为 False 时，设定此时为事务的起点；当 AutoCommit 属性设为 True 时，系统自动取消这一事务。

要建立与数据库的连接，必须设置好 Transaction 对象的属性值，即定义与连接数据库相匹配的连接参数。

2. 操作事务对象

操作事务对象的步骤如下：

先定义开始一个事务，然后对数据做修改操作，这时如果提交 COMMIT，这些修改就永久地保存下来；如果退回 ROLLBACK，数据库管理系统将放弃所做的修改而返回到事务

开始时的状态。

此外，有些数据库支持事务的存储点(Savepoint)这一概念，即在一个事务进程中任意一点都可以进行当前状态的存储，回退时只是回到设定的存储点，而不必退回全部的事务。如果事务可以分成几组对数据库的修改，那就可以设置多个存储点，根据需要可以回退到任意一个存储点，而不会丢失所有事务的修改数据。

正确地管理事务可以保证数据的完整性，在所做的工作尚未全部完成和得到确认之前，没有任何数据真正地写进数据库。

例如：在一个银行应用系统中，前台使用者做出将储户甲账下的一百元存款划归储户乙账下的操作；在后台的数据库中，这两个客户的记录分储在两个表中，当使用者在屏幕上做出如上操作时，在后台需要对两个表进行修改。如果在数据库中对甲用户存款余款作减去一百元的修改后，对乙用户加上一百元的操作修改却失败时，前一个表也必须回到修改前的状态，否则数据库的内容不统一，储户甲白白损失一百元，信息必然不正确。因此进行事务管理是必需的。

传统上认为一个事务包括对一个或多个表的修改，而随着分布式数据库和数据仓库的发展，事务可以包括对一个或多个数据库的修改。在上例中，甲、乙两储户有可能是异地用户，信息分储在不同地域的不同数据库中。

PowerScript 中常用的事务管理的语句有 4 个，分别为 COMMIT、ROLLBACK、CONNECT 和 DISCONNECT。

当应用与数据库建立连接时使用 CONNECT 操作命令，取消连接时执行 DISCONNECT 命令，一般这两个命令分别用在应用的开始和结束，也就是 Application 的 Open 和 Close 事件中。

当一个事务的数据库修改完成后，修改需提交给数据库. COMMIT 语句是一个旧事务结束和一个新事务开始的界线。在修改被提交前，数据库的数据并没有被真正地修改，这些修改被保留在某个工作区，只有做修改的用户才能看到这些被修改以后的值，提交之后，所有的用户都可以看到新值。

如果使用的是多窗口的应用，却只用一个事务对象，就应格外注意 ROLLBACK 和 COMMIT 命令会影响事务的逻辑一致性。在某个窗口执行的这两个指令会使其他窗口应用中所进行的工作提交或回退。

在多用户系统中，修改和提交的时间越接近，提交成功的可能性就越高。因为在一个事务中，尽管所有的 SQL 语句全部执行成功，但在提交时却可能失败。如果在事务过程中，其他用户修改了数据并提交，这时提交将失败。

上述的事务管理方式虽然简单方便，但是在某些应用中，也需要利用数据库系统的嵌套事务和存储点的特性，而 PowerBuilder 内部的事务管理没有提供这样的功能，需要自己设计。

自己进行事务管理的方式是直接使用数据库本身的事务指令。当使用自己的管理方式时，就应使 PowerBuilder 停止管理事务，即设置 AutoCommit 属性为 True，系统内部就不会自动建构事务处理的命令。实现人工事务管理的方式是采用 EXECUTE IMMEDIATE 指令来执行任意的数据库操作，将数据库指令编辑成一个字符串，就可以执行任意的数据定义语句，如建表、建主键、存储过程等。例如：可以用 execute immediate begin transaction

trans-name 这样的指令开始一个事务。采用这种方法，只要所用数据库支持嵌套事务和存储点等事务管理，通过 PowerBuilder 开发出的应用同样也可以实现。

3. 全局事务对象——SQLCA

作为 PowerBuilder 应用与数据库通信区域的事务(Transaction)对象，在应用开始时应建立一个全局的事务对象 SQLCA。由于大多数的应用只用到一个数据库，所以一般开发者主要用 SQLCA 作为与惟一数据库连接的事务对象。

一般来说，可以在应用的 Open 事件或窗口的 Open 事件中为 SQLCA 定义连接参数和连接数据库脚本，也可以在应用 Close 事件或窗口的 Close 事件中断开与数据库的连接。

下面是在应用的 Open 事件中定义 SQLCA 参数连接数据库，并根据 SQLCA 的返回值确定是否连接成功的示例。代码如下：

```
//连接数据库
SQLCA.DBMS = "ODBC"
SQLCA.Database = "EAS Demo DB V4"
SQLCA.AutoCommit = False
SQLCA.DBParm = "ConnectString='DSN=EAS Demo DB V4;UID=dba;PWD=sql' "
Connect Using SQLCA;
//判断连接是否成功
IF SQLCA.SQLCode <> 0 THEN
MessageBox("提示","数据库连接不成功！")
Halt//连接不成功，退出应用程序
END IF
```

在应用的 Close 事件中断开与数据库的连接，可使用如下语句：

```
Disconnect Using SQLCA;
```

注意： 不需要记忆这些繁琐的连接属性和参数值，只需要打开 Database Profile 对话框，选定所连接数据库后单击 Edit 按钮，打开 Database Profile Setup 对话框，如图 8-7 所示，在对话框中设置好各种连接属性，单击 Preview 选项卡，会发现在 Database Connection Syntax 列表框中 SQLCA 所有需要设置的属性都已经设置好，只需把它复制下来，粘贴到应用程序中即可。

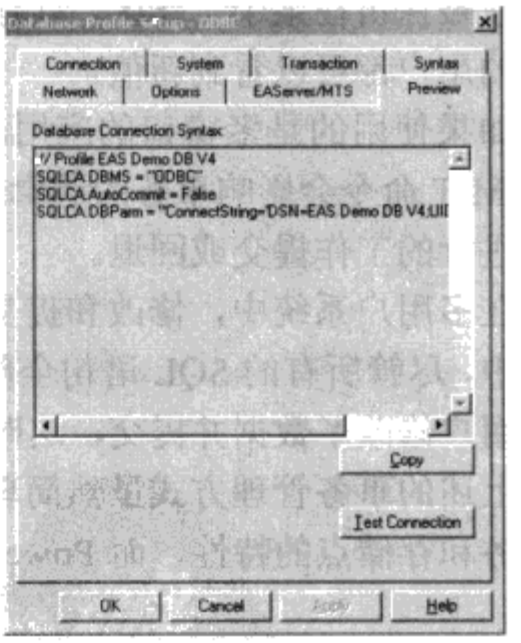


图 8-7 Database Profile Setup 对话框

不仅可以通过编写代码为 SQLCA 的连接参数直接赋值，还可以通过 INI 文件或注册表来为连接参数赋值。

(1) 通过 INI 文件为连接参数赋值，INI 文件的一般文本格式如下：

```
[Section1]
Variable1 = Value1
Variable2 = Value2
……
```

```
[SectionN]
Variable1 = Value1
Variable2 = Value2
```

这样就可以给 SQLCA 定义一个 INI 文件来保存它的各种连接参数。例如，可以对上例建立一个名为 Connect.ini 文件，其代码如下：

```
[Database]
DBMS=ODBC
Database=EAS Demo DB V4
UseID=
DatabasePassword=
LogPassword=
Server name=
LogId=
Lock=
SQLCA.DBParm="ConnectString='DSN=EAS Demo DB V4;UID=dba;PWD=sql' "
Prompt=0
AutoCommit=0
```

这样就可以在程序中通过 ProfileString 或 ProfileInt 函数读取 INI 文件的内容来为参数赋值。

ProfileString 函数用于从 INI 文件中读取一个字符串值，ProfileInt 用于从 INI 文件中读取一个整数值，它们的调用语法如下：

```
ProfileString(filename, section, key, default)
ProfileInt(filename, section, key, default)
可以通过以下代码来读取以上建立的 Connect.ini 文件:
SQLCA.DBMS=ProfileString("Connect.ini", "database","DBMS","")
SQLCA.Database=ProfileString("Connect.ini", "database"," database","")
SQLCA.DBParm=ProfileString("Connect.ini", "database"," DBParm","")
SQLCA.AutoCommit=ProfileString("Connect.ini", "database"," utoCommit","")
```

(2) 通过注册表来为连接参数赋值

注册表文件同 INI 文件类似，它一般是由主键和主键附带的许多变量和变量值组成。例如可以为上例建立主键 HKEY_Local_Machine\Software\Learn\Database，并为该主键建立 DBMS、Database 和 DbParm 3 个变量，它们的值分别为 ODBC、EAS Demo V4 和 ConnectString='DSN=EAS Demo DB V4；UID=dba；PWD=sql'，如图 8-8 所示。

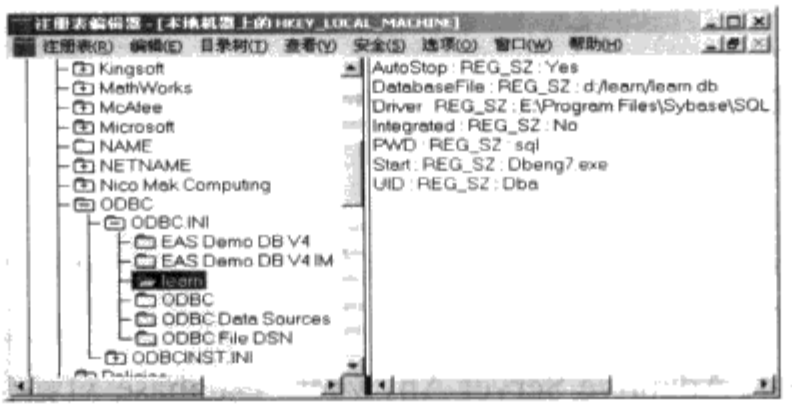


图 8-8　利用注册表来设置参数

通过 RegistryGet 函数就可以读取注册表，RegistryGet 函数的调用语法如下：

```
RegistryGet(key, valuename, vauletype, valuevariable)
```

读取以上建立的注册表的脚本如下：

```
string KeyName = "HKEY_Local_Machine\Software\Learn\Database"
RegistryGet("KeyName", "DBMS", RegString!, SQLCA.DBMS)
RegistryGet("KeyName", "Database", RegString!, SQLCA.Database)
RegistryGet("KeyName", "Dbparm", RegString!, SQLCA.Dbparm)
```

(3) 使用自定义 Transaction 对象来建立连接数据库

一般来说，使用 SQLCA 进行数据库连接已经足够了，但是当需要不只一个连接时，就必须建立自定义 Transaction 对象。建立自定义 Transaction 对象的步骤为：

① 定义一个 Transaction 对象变量。

```
Transaction my_trans
```

② 创建一个 Transaction 对象。

```
my_trans=Create Transaction
```

③ 为 Transaction 对象变量的连接参数赋值，赋值的方法与 SQLCA 赋值的方法相同，例如：

```
my_trans.DBMS="ODBC"
```

④ 使用该 Transaction 对象连接数据库。

```
Connect Using my_trans
```

⑤ 当使用完该 Transaction 对象后，将它从内存中释放。

```
IF Is Valid(my_trans) THEN
Destroy my_trans
END IF
```

8.5.3 创建学生数据库

本节将建立一个数据库，并演示数据库的连接。在运行程序后，将弹出一个“学生管理系统”窗口，上面显示了学生的学号、姓名、性别、所在系、生源和出生时间等信息。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 值设为“学生管理系统”。

2. 创建新数据库

单击工具栏上的 Tools|Database Profiles 命令，或直接单击 按钮，在弹出的对话框中展开 ODB ODBC 项中的 Utilities，如图 8-9 所示，双击 Create ASA Database 选项。在弹出的如图 8-10 所示的 Create Adaptive Server Anywhere Database 对话框中单击 Database Name 项后的 按钮，在 Create Local Database 对话框中建立新的数据库 student，然后单击“OK”按钮保存。一般来说，建立的数据库的 User ID 为 DBA，Password 为 sql，如图 8-10 所示。

3. 创建表

(1) 建立新表：从菜单栏中选择 Tools|Database painter 命令或直接单击工具栏中的

按钮，进入 Database 数据库画板。展开 ODB ODBC 项，右击 Student 数据库，并在弹出的快捷菜单中选择 Connect 命令，连接数据库。在 Student 数据库的展开列表中，右击 Tables，在弹出的快捷菜单中选择 New Tables 命令或直接单击工具栏中的按钮。如图 8-11 建立表，单击工具栏中的按钮，并保存新表为 Student，如图 8-12 所示。

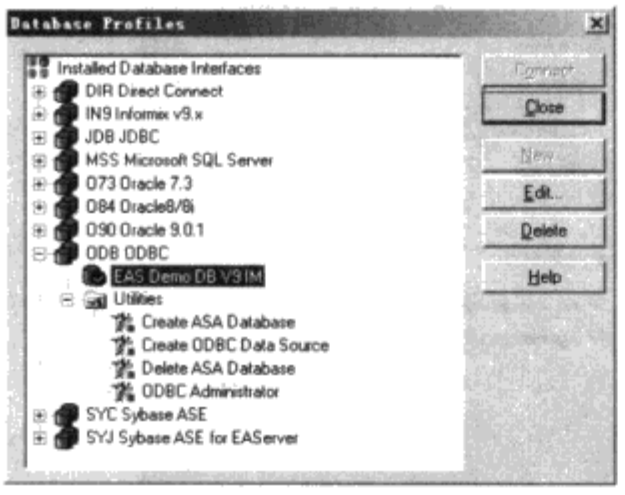


图 8-9　Database Profiles 对话框

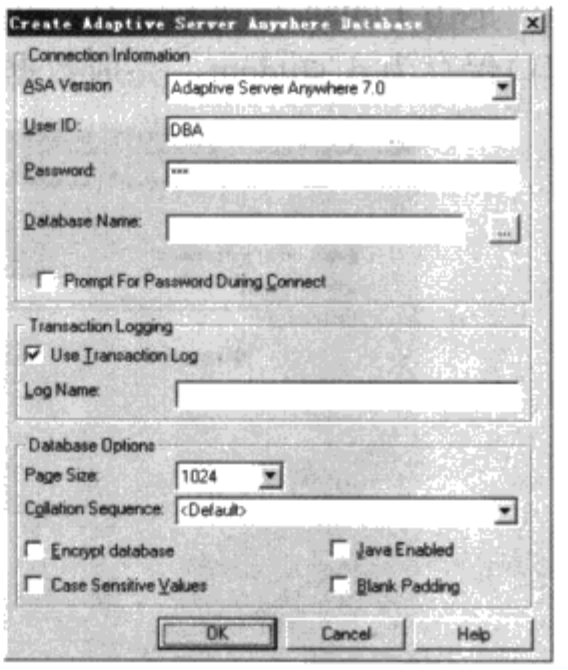


图 8-10　Create Adaptive Server Anywhere Database 对话框

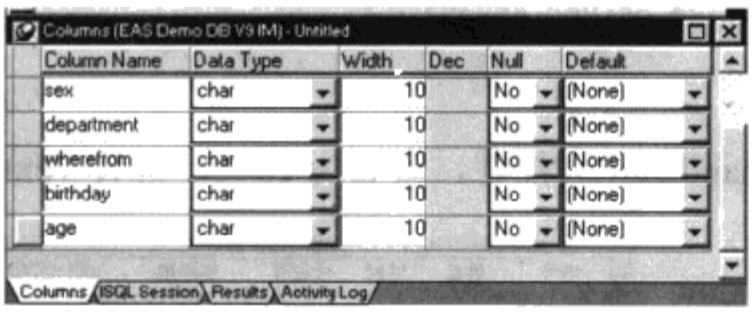


图 8-11　定义新表

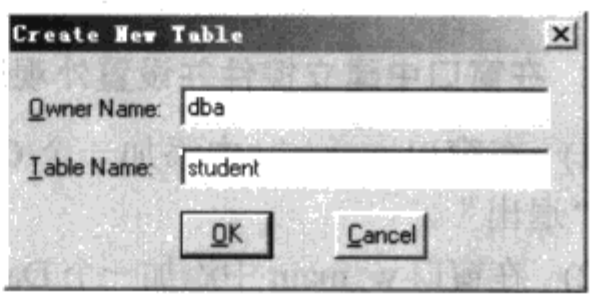


图 8-12　保存新表

(2) 输入数据：选择在 Student 数据库中建立的新表 Student 并右击，在弹出的快捷菜单中选择 Edit Data|Grid 命令，或者直接单击工具栏中的按钮，在弹出的表格中输入新数据，完成后单击工具栏中的按钮，新的表格就得以保存。

4. 建立 ODBC 数据源

在 ODB ODBC，Utilities 展开项中双击 Create ODBC Data Source 选项，在弹出的创建新数据源对话框中，选取“文件数据源(与机器无关)(F)”选项并单击“下一步”按钮，在弹出的下级对话框中选取 Adaptive Server Anywhere 10.0 项并单击“下一步”按钮，在弹出的对话框中单击“浏览”按钮，建立新的 ODBC 数据源 student，然后单击“完成”按钮，在弹出的 Connect to Adaptive Server Anywhere 对话框的 Login 选项卡中输入与数据库相同的 User ID 和 Password，在 Database 选项卡中的 Database file 项中列出新建的数据库，然后单击“确定”按钮，完成数据源的建立，如图 8-13 所示。

5. 建立数据窗口

在菜单栏中选择 File | New 命令，在弹出的 New 对话框中选择 DataWindow 选项卡中

的 Grid 风格的数据窗口，单击 OK 按钮。在出现的 Choose Data Source for DataWindow 对话框中选择 Quick Select 方式，单击 Next 按钮，在出现的快速选择对话框中单击 Table 栏中的 Student 数据表，Student 数据表字段立即显示在 Column 栏中，然后单击 Add All 按钮，单击 OK 按钮。出现一个外观设置对话框，单击 Next 按钮，选择默认设置，在继续出现的对话框中单击 Finish 按钮并保存，在出现的新数据窗口中将表的标题用中文表示，然后将数据窗口保存为 d_student，完成数据窗口的建立。

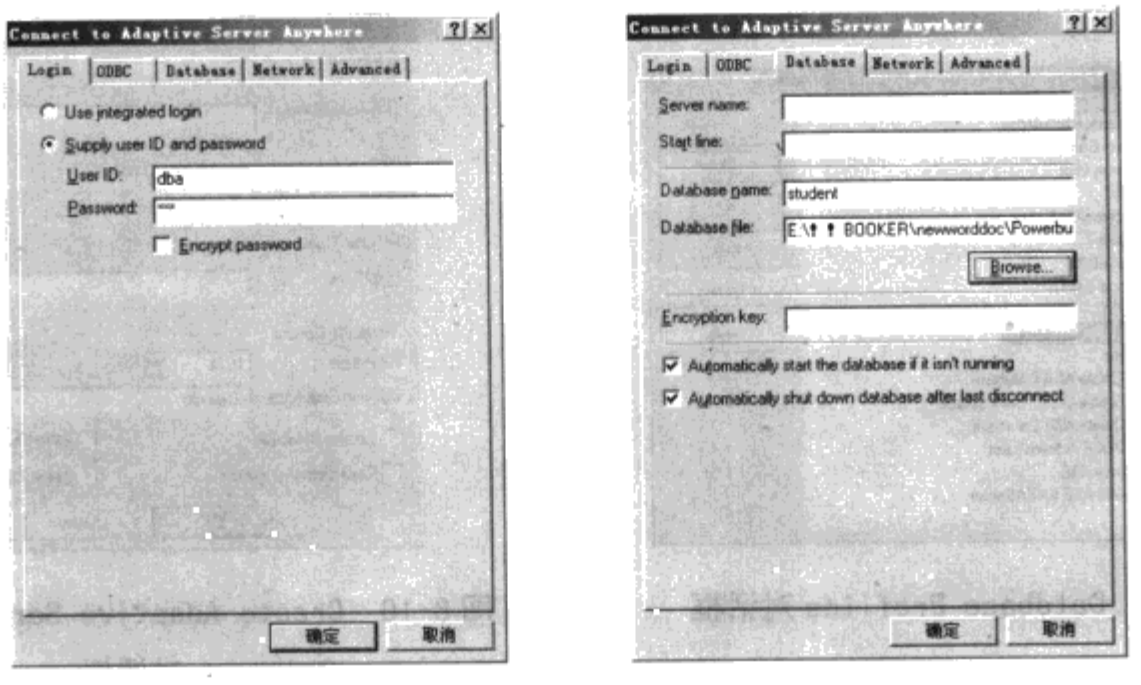


图 8-13 Connect to Adaptive Server Anywhere 对话框

6. 在窗口中建立控件并设置外观属性

(1) 在窗口 w_main 中添加一个 CommandButton 控件，命名为 cb_1，将其 Text 属性设为“退出”。

(2) 在窗口 w_main 中添加一个 DataWindow 控件，名称为 dw_1，选中控件命名 HScrollBar 和 VScrollBar 复选框，单击其 DataObject 属性框后的…按钮，在弹出的 Select Object 对话框中选取 d_student 数据窗口，单击 OK 按钮。

(3) 调整控件的大小及位置，如图 8-14 所示。

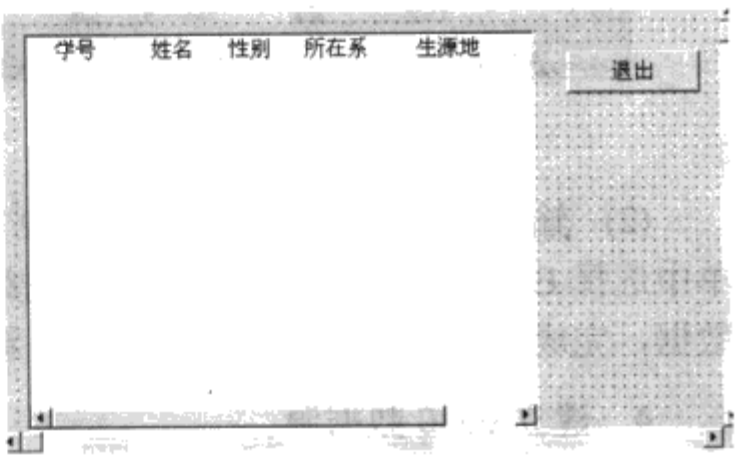


图 8-14 窗口框架

7. 编写事件驱动代码

(1) 在 w_main 窗口的 open 事件下输入如下脚本：

```
//使数据窗口与事务对象相连
dw_1.SetTransObject(sqlca)
//执行检索操作
dw_1.retrieve()
```

(2) 在 w_main 窗口 cb_1 控件的 Clicked 事件下输入如下脚本：

```
Close(w_main)
```

(3) 在 PowerBuilder 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App

的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(4) 在 PowerBuilder 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

单击 PowerBuilder 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行，实例效果图如图 8-15 所示。

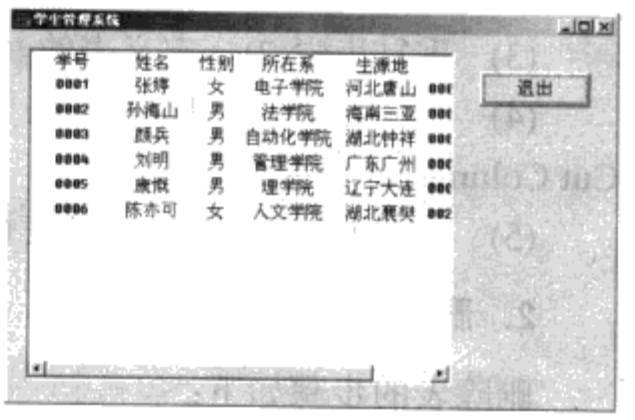


图 8-15　实例效果图

使用 PowerBuilder 可以操作目前大多数流行的数据库，特别是到 10.0 版，在这方面又有很多新的变化，使得操作数据库更方便、更有效。PowerBuilder 开发环境和应用程序在操作数据库之前，必须首先建立与数据库的连接，这种连接是建立在驱动程序(数据库接口)之上的，PowerBuilder 对常用的大型数据库管理系统(如 Oracle、Sybase、SQL Server 等)提供了专用的数据库接口，而对一些小型数据库管理系统(如 xBase)和其他数据源(如 Excel)则提供了 ODBC 接口。

要通过 ODBC 连接数据库，必须首先定义 ODBC 数据源。通过 PowerBuilder 的 Database Profiles 对话框调用 ODBC 管理程序。

8.6　操作数据库表

数据库的表用行和列来表示数据，行就是记录，列就是字段。操作数据库表包括几方面的含义：创建表、定义表中列的扩充属性、定义主键、创建索引、定义外部键、修改表、删除表、删除索引和删除主键等。本节将介绍操作数据库表的方法。

8.6.1　表的创建与删除

1. 创建表

首先创建表，可以采用如下两种方式创建：

- 利用数据库管理系统本身提供的工具创建表。
- 使用 PowerBuilder 的数据库画板或表操作画板创建表。

下面将介绍第二种方法，具体操作步骤如下：

(1) 单击数据库画板工具栏上的 Create table 图标；或者在 Columns 窗口中右击，从弹出的快捷菜单中选择 New Table 命令；或者在 Objects 窗口里右击 Table 并从弹出的快捷菜单中选择 New Table 命令。创建新表的模板显示在 Columns 窗口中，所看到的 Columns 窗

口将根据 DBMS 的不同而不同，如图 8-16 所示。

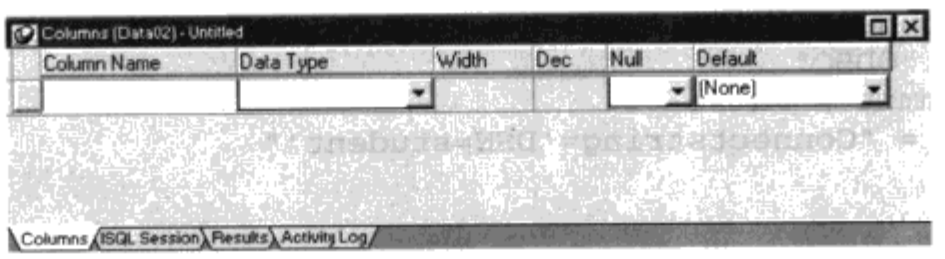


图 8-16 Columns 窗口

(2) 定义数据列的属性。在定义数据列属性时，可以使用 Tab 键来切换输入焦点。在完成了列定义的最后一项后，按下 Tab 键将自动切换至下一列的工作区。

(3) 重复步骤(2)，定义表的其他列。

(4) 当删除多余的数据列时，可以在所要操作的列上右击，在弹出的快捷菜单中选择 Cut Column 删除该列。

(5) 单击 Save 按钮，然后在 Create New Table 对话框中输入新表的名称。

2. 删除表

删除表的步骤如下：

(1) 在 Object 窗口或者在 Objects Layout 窗口里选择该表并右击，从弹出的快捷菜单中选择 Drop Table 命令，或者选择 Object 菜单中的 Delete 命令。

(2) 在弹出的对话框中，单击 Yes 按钮，删除该表。如果在 PowerBuilder 外删除了一个表，那么该表和扩展属性的信息仍然保留在系统中，这些信息称为丢失关联表的表信息，这时应该将这些表信息从系统表中删除。

选择 Design 菜单中的 Synch Extended Attributes 命令。如果试图删除已经不存在的丢失关联表的表信息，将会在状态栏中通知用户，这种删除是不必要的。否则将弹出一个确认对话框，单击 Yes 按钮，删除丢失关联表的表信息。

8.6.2 定义表的属性、列的属性和扩展属性

创建并保存了表以后，可以定义表和列的属性。表的属性包括标题、标签和数据的字体以及与该表相关的注释。列的属性包括标题与标签的文本、显示格式、校验规则、数据编辑风格以及与该列相关的注释。

1. 定义表的属性

除了可以为该表定义注释信息外，还可以在数据窗口对象里为显示的信息选择字体，定义字体、大小、颜色以及风格等属性。

定义表的属性的步骤如下：

(1) 在 Object 窗口或者在 Objects Layout 窗口里选择该表并右击，从弹出的快捷菜单中选择 Properties 命令，出现如图 8-17 所示的对话框。

(2) 选择一个选项卡并定义该表的属性，如表 8-7 所示。

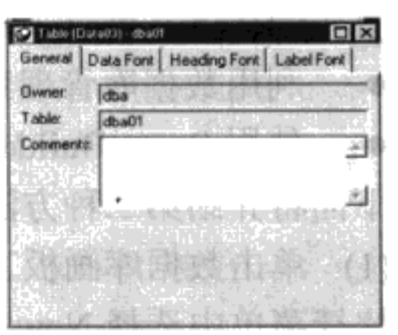


图 8-17 Table Properties 选项卡

(3)　保存所做的修改。

表 8-7　Columns 的选项卡以及属性

选项卡	属　性
General	关联到表的声明
Data Font(数据字体)	从数据库中检索到并显示在数据窗口对象里的数据字体
Lable Font(标签字体)	FreeForm 风格的数据窗口对象显示在数据操作窗口中的列标识的字体
Heading Font(标题字体)	Grid、Tabular 和 N-UP 风格的数据窗口对象显示在数据操作窗口的列标识的字体

2．定义列的属性

在创建新表时必须定义每个表的数据类型等属性。在 Columns 窗口中显示的每个列的域与具体的 DBMS 有关，表的属性说明如表 8-8 所示。

表 8-8　Columns 的域以及可以输入的内容

域	输 入 内 容
Columns Name	列的名称(必需)
Data Type	从下拉列表框中选择一种数据类型，当前 DBMS 支持的所有的数据类型都列在此列表框中(必需)
Width	对可变长的数据类型，指定字段中字符串的最大长度
Dec	对于数字型数据类型，指定小数点后保留的位数
Null	从 Null 下拉列表框里选择 Yes 或者 No 来定义该列是否允许空值
Default	在数据窗口对象中插入一行时，如果没有为该列提供一个值，则使用这个值，下拉列表框中列出了内置的选择，也可以自己输入一个值

3．定义列的扩展属性

除了为列添加注释信息外，还可以为每个列定义扩展属性。扩展属性用于加强列定义的特殊信息。

定义列的扩展属性的步骤如下：

(1)　在 Objects 窗口或 Objects Layout 窗口中选择列并右击，从弹出的快捷菜单中选择 Properties 命令，Column Details 选项显示在属性窗口的 General 选项卡中，如图 8-18 所示。

(2)　选择一个选项卡并定义相关的扩展属性，如表 8-9 所示。

(3)　保存修改。

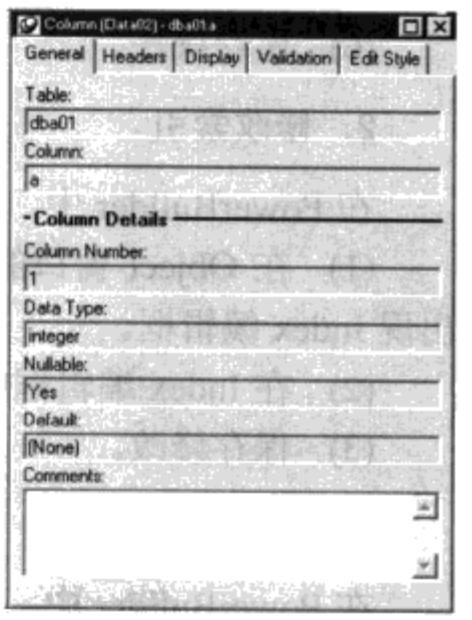


图 8-18　列的扩展属性窗口

扩展属性被保存在 PowerBuilder 的数据库系统表中。在数据库画板中自动使用这些信息来显示、提供和校验数据窗口对象的数据。当在数据库画板中创建了一个视图时，列的所有扩展属性使用默认值。

表 8-9 Object Details 的选项卡及扩展属性

选项卡	可以修改的扩展属性
General	列声明
Headers	为 FreeForm 风格数据窗口对象定义标签文本，为 Grid、Taular 和 N-UP 风格数据窗口对象定义标题文本
Display	数据窗口对象显示时的高度、宽度和位置等数据格式信息
Edit Type	列在数据窗口中的显示方式，例如：可以将列数据显示为单选按钮或下拉列表框
Validation	在数据窗口对象里数据必须通过的校验标准

8.6.3 表的索引

1. 创建索引

在 Adaptive Server Anywhere 数据库里，不能把一个已定义为外键的列作为索引列，因为这个列已经作为外部参照进行了优化。

创建索引的步骤如下：

(1) 选择要创建索引的表，在 Object Layout 的窗口中选择表并右击，在弹出的快捷菜单中选择 New | Index 命令，或者在菜单中选择 Object | Insert | Index 命令，Index 选项显示在 Object Details 窗口中，如图 8-19 所示。

(2) 在 Index 编辑框中输入索引名称。

(3) 选择是否允许多值索引。

(4) 定义数据库需要的其他信息。

(5) 单击组成索引的列。

(6) 保存操作。

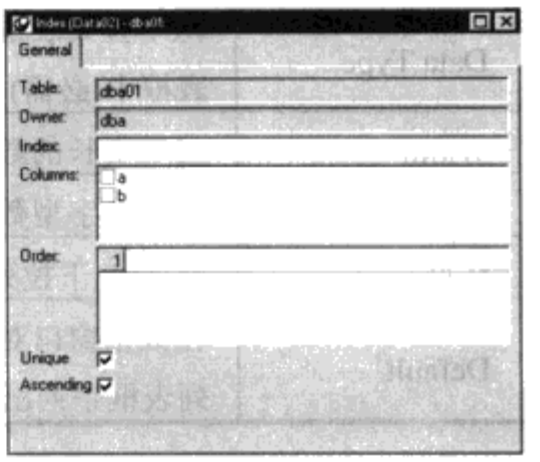


图 8-19 Index 编辑框

2. 修改索引

在 PowerBuilder 中，可以修改已定义的索引，步骤如下：

(1) 在 Object 窗口的表的展开树上右击，从弹出的快捷菜单中选择 Properties 命令，出现 Index 编辑框。

(2) 在 Index 编辑框中按照需要对 Index 进行修改。

(3) 保存修改。

3. 删除索引

在 PowerBuilder 中，可以删除已定义的索引，操作如下：

(1) 在数据库画板中，选择要删除的索引并右击。

(2) 从弹出的快捷菜单中选择 Drop Index 命令，并在弹出的确认对话框中单击 Yes 按钮，完成删除。

8.6.4　表的主键、外键

在数据库应用中，使用主键和外键可以加强数据库的参照完整性。通过这些键，可以使 DBMS 确保只有符合校验规则的数据才能被数据库系统接收。

1．创建主键

(1) 选择要创建的表，单击 Create Primary Key 按钮，或者从菜单中选择 Object | Insert | Primary Key 命令，或者在表上右击，从弹出的快捷菜单中选择 New | Primary Key 命令，General 选项卡显示在 Object Details 窗口中，如图 8-20 所示。

(2) 选择一列或者多列作为主键，只有不允许为 Null 的列才允许作为主键，否则将显示关于 DBMS 的警告。

(3) 定义 DBMS 需要的其他信息。

(4) 保存操作。

2．创建外键

如果 DBMS 支持外键，就可以在 PowerBuilder 中定义表的外键。

(1) 选择要创建外键的表，单击 Create Foreign Key 按钮，或者从菜单中选择 Object | Insert | Foreign Key 命令，或者在表上右击从弹出的快捷菜单中选择 New | Foreign Key 命令，General 选项卡显示在 Object Details 窗口中，如图 8-21 所示。

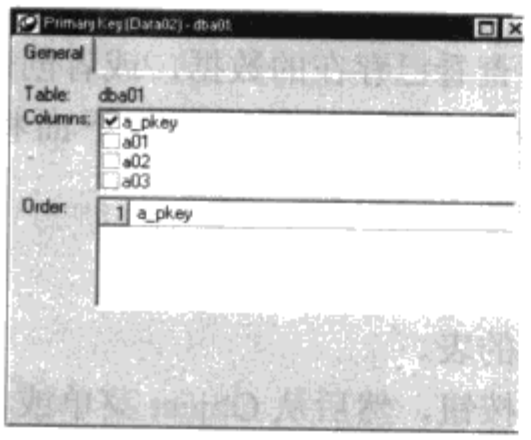


图 8-20　Primary Key 的 General 选项卡

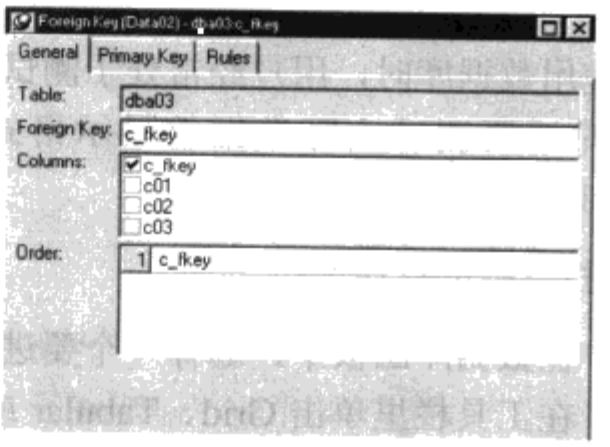


图 8-21　Foreign Key 的 General 选项卡

(2) 在 Foreign Key 编辑框中命名外键。

(3) 为外键选择列。

(4) 在 Primary Key 的选项卡中选择外键所参照的表和列的主键。外键包括数据类型、精确度和小数位等，它必须与参照主键相匹配。

(5) 在 Rules 选项卡中，选择删除从表时所对应的主表记录，此时数据库可进行如下操作：

- Disallow if Dependent Rows Exits(RESTRICT)：当从表中存在与该记录相对应的记录时，不允许删除主表中的记录。
- Delete any Dependent Rows(CASCADE)：当删除主表中的记录时，与该记录相关联的从表中的记录都将被删除。
- Set Dependent Columns to NULL(SET NULL)：当删除主表中的记录时，与该记录

相关联的从表中的记录中的字段设为 NULL，如果该字段不允许为空，则该操作失败，系统将提示出错信息。

(6) 保存操作。

3. 修改键

修改键的步骤如下：

(1) 选择列在 Object 窗口中表的展开树中的主键/外键，单击 Properties 按钮；或者从 Object 菜单中选择 Properties 命令。

(2) 对主键/外键的信息进行修改。

(3) 保存修改。

4. 删除键

删除键的步骤如下：

(1) 在 Object 窗口中表的展开树中的主键/外键上，或者在 Object Layout 窗口中的表的视图上右击要删除的键。

(2) 在弹出的快捷菜单中选择 Drop Primary Key 或者 Drop Foreign Key 命令。

(3) 在弹出的确认对话框中，单击 Yes 按钮。

(4) 保存修改。

8.6.5 操作数据

在使用数据库时，用户经常处于测试的目的需要查看已存在的数据，或者创建一些数据；有时还要测试实际数据的显示格式、编辑风格以及校验规则等。下面将介绍在 PowerBuilder 中进行的数据操作。

1. 打开数据操作窗口

(1) 在数据库画板中，选择一个要进行数据操作的表。

(2) 在工具栏里单击 Grid、Tabular 或 FreeForm 按钮，然后从 Object 菜单或者右击选择的表弹出的快捷菜单中选择 Edit Data 命令，这里单击 FreeForm 按钮，显示 Free 表，如图 8-22 所示。

2. 检索数据

在工具栏中单击 Retrieve 按钮，或者选择 Rows 菜单中的 Retrieve 命令开始检索数据的操作。检索数据时，Retrieve 按钮的标题将变为 Cancel，这时单击该按钮可以中断对数据的检索。

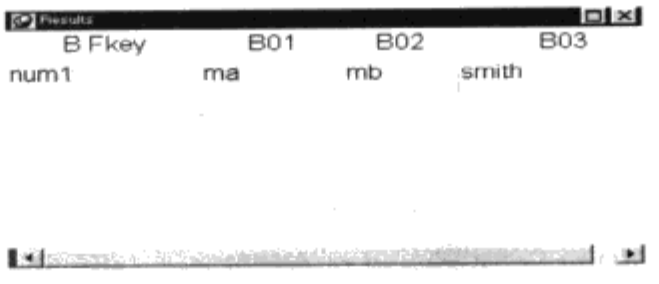


图 8-22 Free 表

3. 修改数据

在检索完数据后，需要对检索出来的数据进行添加、修改或删除等操作。完成了对数据的这些操作后，提交给数据库，以保存所作的修改。

修改存在的数据，将鼠标指针定位在要修改的字段上，并输入一个新值。

添加一行，将鼠标指针定位在要插入的位置，单击 Insert Row 按钮并为添加的行输入数据。删除一行，选择要删除的数据行，单击 Delete Row 按钮。

4. 导入数据

在数据库操作过程中，需要从外部导入数据。PowerBuilder 提供了导入外部数据并将导入的数据显示在数据操作窗口中的功能，然后还可以将这些导入的数据保存到数据库中，步骤如下：

(1) 选择 Rows 菜单中的 Import 命令，将显示 Select Import File 对话框。

(2) 定义要导入数据的文件名，可以将导入到画板的数据文件类型显示在 File of Type 下拉列表框中。

(3) 选择要导入的数据文件后，单击 Open 按钮。

5. 保存数据

要实现数据的导出功能，可以将显示的数据保存到外部文件中，操作步骤如下：

(1) 选择 File 菜单中的 Save Rows As 命令，将弹出 Save Rows As 对话框。

(2) 选择要导出的文件格式，PowerBuilder 支持多种格式的导出文件，包括 PSR 和 HTML 文件。

(3) 命名并保存文件。

8.7 实例：创建、配置并连接后台数据库

本例的任务是在数据库管理系统中创建后台数据库，配置数据库并连接。创建数据库后台是 PowerBuilder 程序开发的基础。本例选用的是 SQL Server 数据库，这个数据库中的各个表在以后的实例编程中将会陆续用到。在本例的最后，介绍把一个 Access 数据库连接到 PowerBuilder 应用中来的具体方法。

8.7.1 选择数据库管理系统

PowerBuilder 是一种数据库应用程序开发软件，因此利用数据库管理系统(DBMS)创建数据库后台是一切程序设计工作的基础。数据库管理系统维护数据库的安全性、一致性、完整性和可靠性。在应用程序开发时根据实际工作要求的系统范围、数据库大小、性能比较、预算水平等因素，选择适当的开发工具是很重要的。开发人员在确定使用哪种数据库管理系统时，还要着重于技术性能方面，包括伸缩性、平台支持和技术支持等。

PowerBuilder 与数据库管理系统的连接需要三部分接口。一是 PowerBuilder 自身携带的数据库接口，即一组动态连接数据库(DDL)组成。二是由数据库厂商提供的接口文件，这些文件提供与 PowerBuilder 调用的 API。三是数据库网络支持程序，这些程序随数据库

服务器所处的具体网络类型不同两不同。下面是 PowerBuilder 本身驱动程序支持的一些数据库管理系统：

- Informix 7.x
- Informix 9
- JDBC，Jagular(JDS,JDM)
- MS SQL Server 7.0(MSS)
- ODE DB
- Oracle 8.0(OR8)
- Sybase SQL Server 10.x&11.x(SYS)
- ASE Jagular
- Direct Connect(Access Services)TRS Only

本例采用的是 SQL Server 7.0 数据库。

8.7.2 数据库设计

数据库名称为 DB_MIS，主要涉及的表有，成品类型表、成品库库存文件表、成品入库/出库单存档表等，表的清单如表 8-10 所示。

表 8-10 表清单

编号	表 名	中文含义	备注
1	(cc_cplx)	成品类型表	公共信息表 1
2	(cc_cpk)	成品库库存文件表	成品库表 1
3	(cc_cpckrkd)	成品入库/出库单存档表	成品库表 2

8.7.3 配置数据库连接

在用 PowerBuilder 以外的数据库管理系统环境创建了数据库之后，需要把这个数据库连接到 PowerBuilder 中来，以供使用。

在 PowerBuilder 的 DB Profile 配置面板里列出了如下数据源以供选用，DIR Direct Connect Interface、IN7 Informix 7.x、MSS MS SQL Server 7.0 (MSS)、073 Oracle 7.3、ODBC、Sybase SQL Server 10.x&11.x。

在选择 ODBC 数据源的时候，可以单击 DB Profile 配置面板右边的 Configure ODBC 按钮进行数据源配置，也可以在 PowerBuilder 工作栏中单击图标 Configure ODBC 打开其配置面板进行数据源配置。

(1) 按照数据库系统的设计方法与步骤设计数据库的结构，注意数据库各个数据项的依赖性、传递性等符合第三范式的要求。

(2) 设计各个表的结构及完整性约束。下面以成品类型表(cc_cplx)、成品库库存文件表(cc_cpk)和成品入库/出库单存档表(cc_cpckrkd)的结构为例说明各个表的结构及完整性约束的设计。

成品类型表(cc_cplx)的结构设计(数据类型、主键、是否可以为空值等)如表 8-11 所示。

表 8-11　成品类型表(cc_cplx)

字段名称	类型	可否空	是否主键	是否外键	备注
cplx_bh	Char(20)	N	Y	N	类型编号
cplx_mc	Char(50)	N	N	N	品名
cplx_sm	Char(50)	Y	N	N	规格
cplx_ys	Char(10)	Y	N	N	颜色
cplx_dw	Char(2)	N	N	N	单位
cplx_ccl	Numeric(9,18)	N	N	N	仓储量
cplx_aqkc	Numeric(9,18)	N	N	N	安全库存量
cplx_position	Varchar(200)	N	N	N	存放位置
cplx_others	Varchar(200)	Y	N	N	备注
Ready	Varchar(200)	Y	N	N	备注

成品库库存文件表(cc_cpk)的结构设计(数据类型、主键、是否可以为空值等)如表 8-12 所示。

表 8-12　成品库库存文件表(cc_cpk)

字段名称	类型	可否空	是否主键	是否外键	备注
cpk_mc	Char(50)	N	N	N	品名
cpk_gg	Char(10)	Y	N	N	规格
cpk_djhm	Char(14)	N	Y	N	单机号码
cpk_lx	Char(20)	N	N	Y	类型编号
cpk_ys	Char(10)	N	N	N	颜色
cpk_dw	Char(2)	N	N	N	单位
cpk_dqzt	Char(2)	N	N	N	当前状态
cpk_others	Varchar(200)	Y	N	N	备注
Ready	Varchar(200)	Y	N	N	备注

成品入库/出库单存档表(cc_cpckrkd)的结构设计(数据类型、主键、是否可以为空值等)如表 8-13 所示。

表 8-13　成品入库/出库单存档表(cc_cpckrkd)

字段名称	类型	可否空	是否主键	是否外键	备注
cpckrkd_bh	Char(15)	N	Y	N	单号
cpckrkd_rq	Datetime(8)	N	N	N	日期
ckrkd_djhm	Char(14)	N	Y	N	单机号码
cpckrkd_lx	Char(20)	N	N	Y	类型编号
cpckrkd_mc	Char(50)	N	N	N	名称

续表

字段名称	类型	可否空	是否主键	是否外键	备注
cpckrkd_gg	Char(10)	Y	N	N	规格
cpckrkd_ys	Char(10)	N	N	N	颜色
cpckrkd_dw	Char(2)	N	N	N	单位
cpckrkd_dgcs	Varchar(200)	N	N	N	供应/订购厂商
cpckrkd_others	Varchar(200)	Y	N	N	备注
Ready	Varchar(200)	Y	N	N	备注

其完整性约束如下：

- Varchar 类型一般取 200 位。
- 名称一般用 char(50)。
- 单台机器号一般取 15 位，单号格式如下：

								9	10	11	12	13	14	15
年、月、日								出/入用 0/1 表示	预留位(00)		实际的单据排号			

- 成品单台机器号一般取 14 位，机器号格式如下：

								9	10	11	12	13	14
年、月、日								生产地编码	实际的机器序列号				

- 成品类型编码如下：

X-XXXX-XX

其中，第一段表示颜色，中间段表示编码，第三段表示构型差异。

(3) 在 SQL Server 2000 中创建数据库 DB_MIS 及各个表，已备本书以后讲的 PowerBuilder 应用程序使用。

(4) 在 PowerBar 中单击 Database 图标，弹出如图 8-23 所示的窗口。在 Objects 子窗口中的 Installed Database Interfaces 树列表中，列出已经安装的可以与 PowerBuilder 连接的各个数据库，DIR Direct Connect、IN9 Informix v9.x、JDB JDBC、MSS Microsoft SQL Server、084 Oracle 8.x、Oracle8i、ODB ODBC、SYC Sybase ASE、SYJ Sybase ASE for EAServer 。

(5) 选择并右击 MSS Microsoft SQL Server，在弹出的快捷菜单中选择 New Profile 命令，打开 Database Profile Setup 对话框，在 Profile Name 编辑框中填入文件名 DB_MIS，在 Server 编辑框中填入服务器名称 linlin，在 Login ID 编辑框中填入 MSS Microsoft SQL Server 的注册身份 sa，在 Database 编辑框中填入数据库名称 db_mis，如图 8-24 所示，单击 OK 按钮，则数据库树列表中生成新建的数据库 DB_MIS。

(6) 在数据库的树列表中双击新建的数据库 DB_MIS，或右击它在弹出的快捷菜单中选择 Connect 命令来连接数据库。

数据库名称 DB_MIS 前有“√”出现，表明该数据库连接成功，至此完成了本例的所有工作。

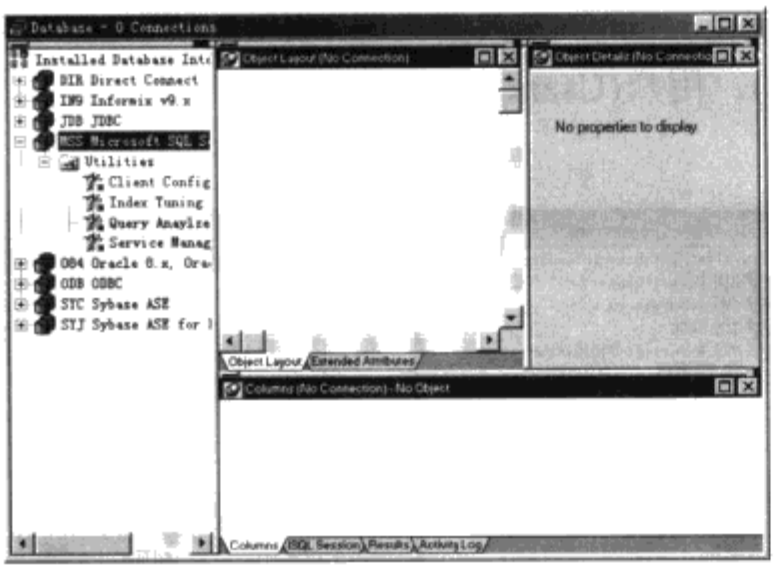


图 8-23　Database 窗口

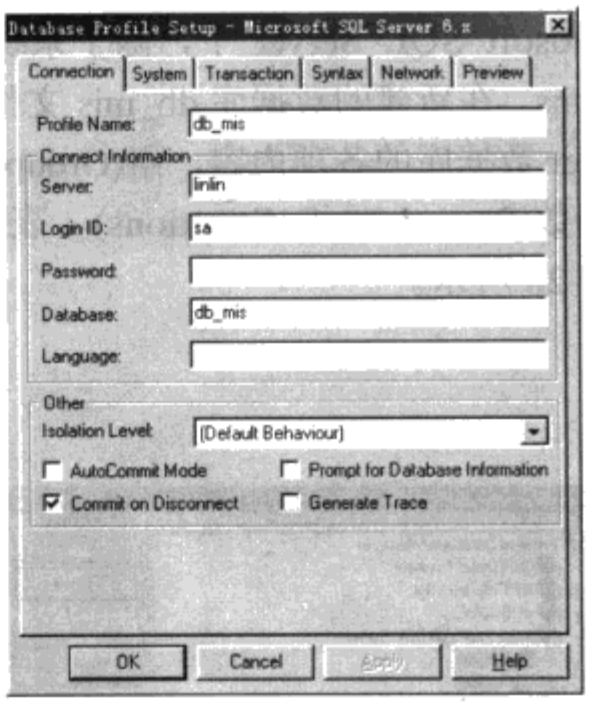


图 8-24　SQL Server 数据库设置窗口

若把一个 Access 数据库连接到 PowerBuilder 应用中，需要应用 ODBC 数据源，则相应的步骤如下：

(1)　创建 ODBC 数据源 bfz。

(2)　在 PowerBuilder 工具栏 PowerBar 中单击 DB Profile 图标，打开其配置面板，如图 8-25 所示。选中 ODB ODBC，单击右侧的 New 按钮。

(3)　填入文件名，并选择数据源，单击 OK 按钮，如图 8-26 所示。

(4)　在数据库树列表中双击新建的数据库 pass，或右击它并在弹出的快捷菜单中选择 Connect 命令来连接数据库。结果 pass 数据库名称前有“∨”出现，表明该数据库连接成功。

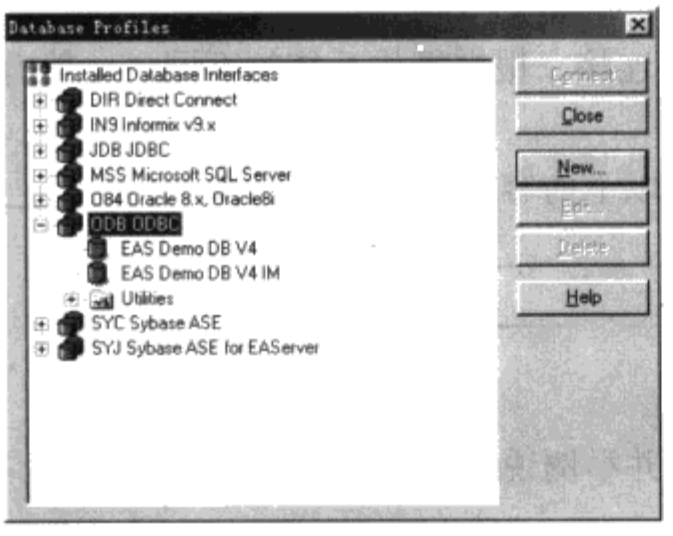


图 8-25　DB Profile 配置面板

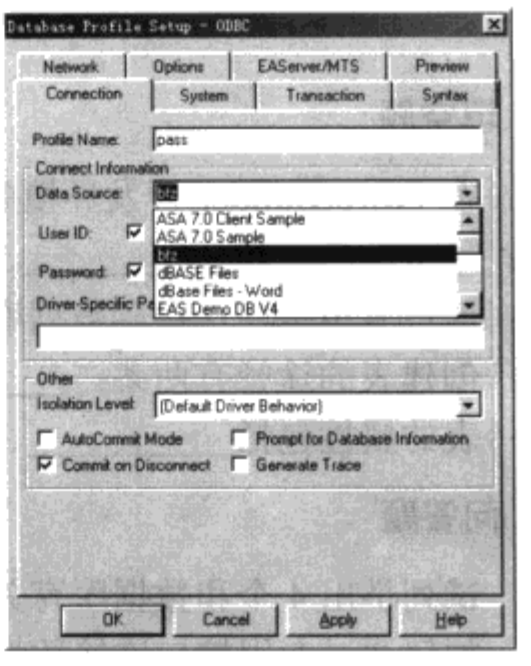


图 8-26　ODBC 数据库设置窗口

利用 ODBC 数据源连接完成的数据库 pass 在数据库树列表中的情况如图 8-27 所示。在图 8-27 中可见，在 ODB ODBC 下，除了系统原有的 Utilities 文件夹、EAS Demo DB V4 文件夹和 EAS Demo DB V4 IM 文件夹以外，还有本例新建的数据库 pass 数据库。

可以从数据库的树列表中看到所创建的数据库 DB_MIS 的连接情况。在 MSS

Microsoft SQL Server 下，除了系统原有的 Utilities 文件夹以外，还有本例新建的数据库 db_mis。在新建的数据库 db_mis 文件夹中，有以下七个子文件夹，它们负责存放管理 SQL Server 数据库的各项内容，组(Groups)、注册(Logins)、元数据类型(Metadata Types)、程序和函数(Procedure & Functions)、表(Tables)、用户(Users)、视图(Views)，具体的效果如图 8-28 所示。

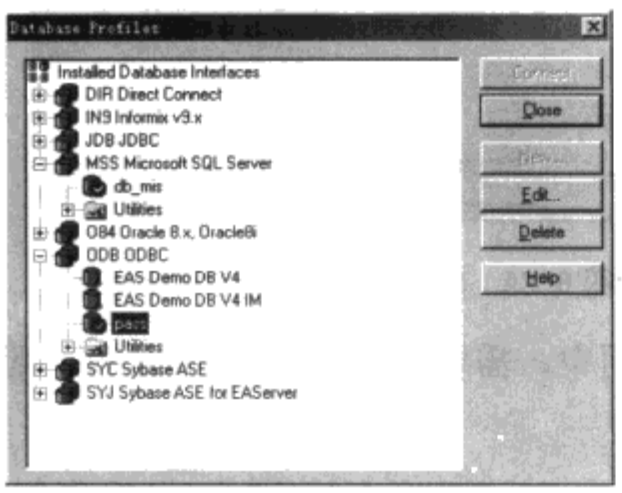


图 8-27 配置并连接 ODBC 数据库的实例效果

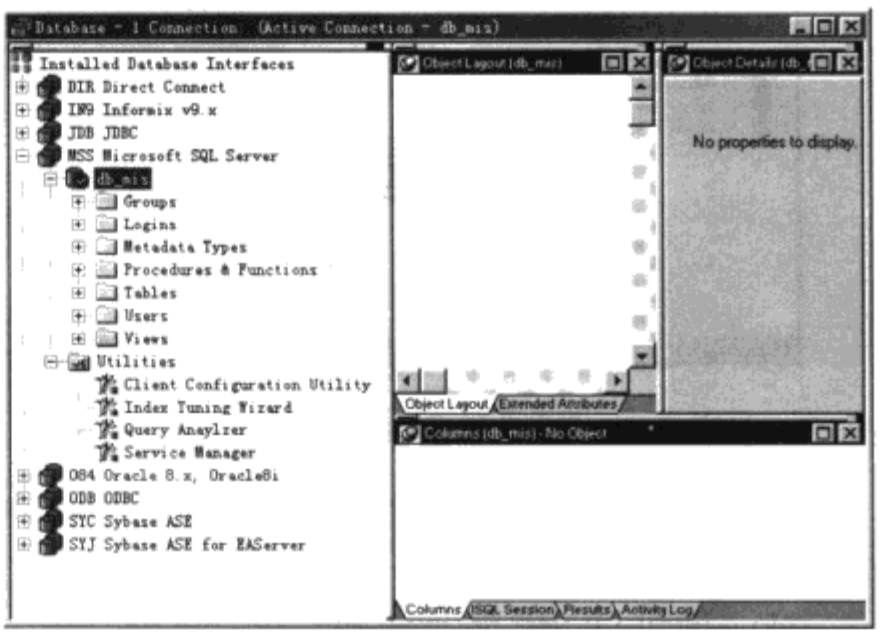


图 8-28 创建数据库后台的实例效果

在本例中，我们学习了为 PowerBuilder 应用程序创建数据库后台。掌握了在 PowerBuilder 中创建 SQL Server 数据库、配置数据库连接的方法和步骤，还掌握了把一个 Access 数据库连接到 PowerBuilder 应用中的具体方法。

8.8 习 题

1. 填空题

(1) 在 PowerBuilder 开发环境中与数据库连接、根据操作系统的不同，一般有以下两种方法：_____和_____。

(2) 通过专用接口连接到数据库前应检查以下内容：_____和_____。

(3) 创建表的途径有两条：_____和_____。

(4) 表的属性包括_____、_____、_____。

2. 问答题

(1) 试列举出 4 个和数据库有关的概念，并举例说明。

(2) 如何通过专用接口连接数据库？

(3) 如何删除数据库？

(4) 如何为数据表添加主键、外键？

第 9 章　数据窗口技术

数据窗口是 PowerBuilder 的一大特色，数据窗口是一个对象，它包含了对数据库中的数据进行特定操作的信息。定义完一个数据窗口对象后，就可以在多个应用程序中使用这个数据窗口对象。可以把数据窗口看作是封装了操作数据库中数据的对象，它极大地方便了应用程序对数据库的使用。

9.1　数据窗口概述

9.1.1　一个简单的数据窗口设计示例

本节首先以一个简单的数据窗口设计示例来简单介绍数据窗口的基本设计方法。首先在 Adaptive Server Anywhere 中创建了一个数据库(data04)，其中包含两个表，分别是员工信息表(dba01)和部门信息表(dba02)，如表 9-1 和表 9-2 所示。员工信息表的主键是工作证号，部门信息表的主键是部门，它对应着员工信息表的外键部门。

表 9-1　员工信息表

工作证号	姓名	性别	出生年月	部门	职称	电话	备注
001	李达	男	1963/8/6	办公室	工程师	62275386	
002	赵晓华	女	1979/8/12	财务部	会计	62271209	
003	张红	女	1952/2/23	市场部	工程师	62278574	
004	王维	男	1965/7/8	维修部	高级工程师	62273458	

表 9-2　部门信息表

部　门	部门主管	主管房间
市场部	李意	2124
办公室	胡卫	1224
财务部	周良	4122
维修部	李远	3122

创建数据窗口的步骤如下：

(1) 建立与数据库的连接，这里建立了与 data04 数据库的连接。

(2) 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 Workspace 选项卡，然后再双击其中的 Workspace 图标建立一个新的工作空间。

(3) 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 Target 选项卡，双击 Application 图标，建立一个应用。

(4) 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 DataWindow 选项卡，这

里共有 11 种不同的数据窗口样式，如图 9-1 所示。

(5) 选择其中的 Grid 图标，单击 OK 按钮。弹出如图 9-2 所示的 Choose Data Source for Grid DataWindow 对话框，在此对话框中选择数据源，这里选择 SQL Select 图标，单击 Next 按钮。

(6) 弹出选择数据表(Select Tables)对话框，可以选择一个或多个表作为数据源，如图 9-3 所示。这里选择 dba01 表，单击 Open 按钮。

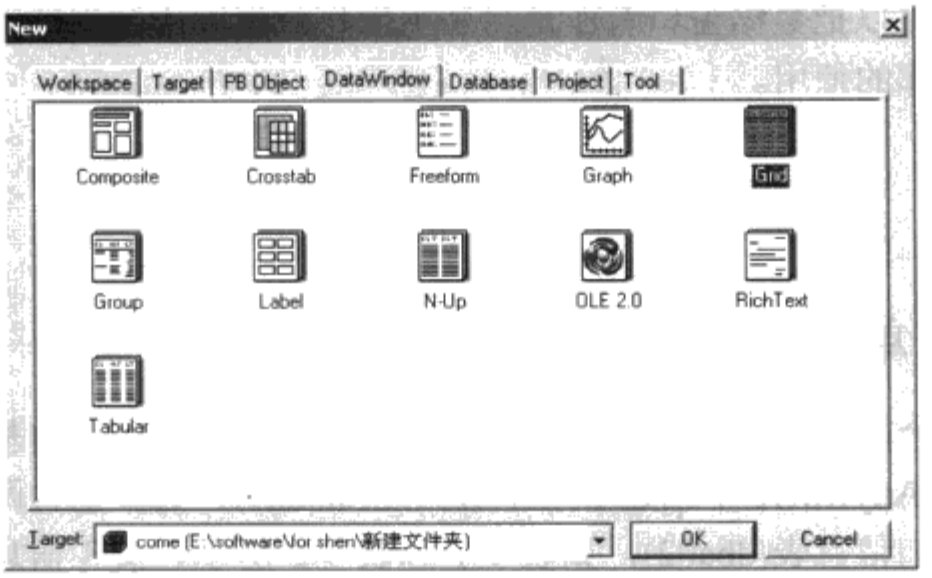


图 9-1 DataWindow 选项卡

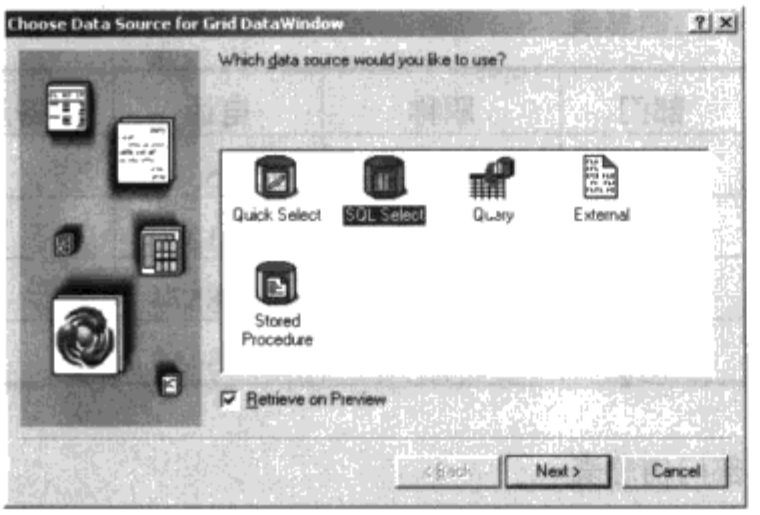


图 9-2 选择数据源窗口

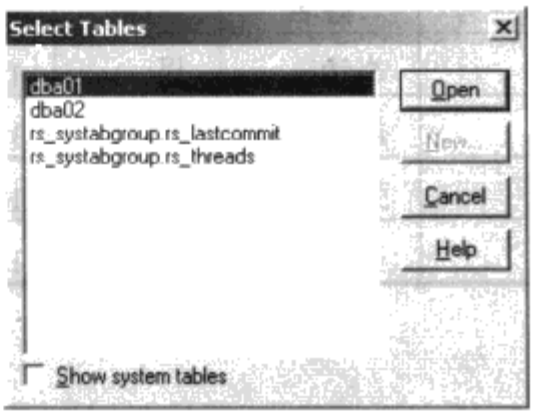


图 9-3 选择数据表窗口

(7) 出现选择数据列(Table Layout)对话框，如图 9-4 所示。这里选中全部数据列，然后单击菜单栏上的 Return 图标。在图 9-4 对话框的底部，可以设置检索条件。

(8) 单击 Where 标签，显示 Where 选项卡，如图 9-5 所示。

单击 Column 下的第 1 个空白行，显示一个列名下拉列表框，从中选择一个列名。这里选择工作证号。

单击 Operator 下的第 1 行，显示一个运算符下拉列表框，从中选择所需的运算符，可选择的操作符有“=”、“>”、“>=”、“<”、“<=”、“<>”、“LIKE”、“IN”和“EXISTS”等，默认时使用等号“=”。

图 9-4　选择数据列窗口

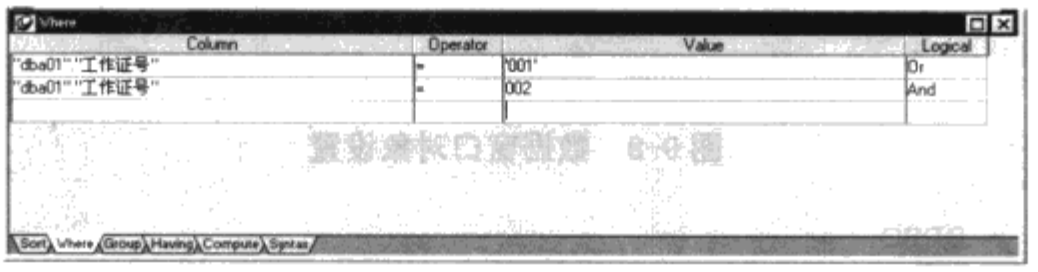


图 9-5　数据检索条件

在 Value 列下的下拉列表框中输入检索数值或参数名。

需要多个条件时，单击 Logical 下的第一行，根据需要选择 And/Or 逻辑运算符后，在下一行继续重复上面介绍的步骤。

(9)设置完检索条件后，关闭图 9-4 所示的选择列对话框，此时出现编辑数据窗口的外观颜色窗口，如图 9-6 所示。这里选取默认值(数据窗口颜色背景为白色，前景为黑色，列框颜色为黑色)，单击 Next 按钮。

(10) 出现数据窗口配置清单(Ready to Create Grid DataWindow)对话框，如图 9-7 所示。单击 Finish 按钮。

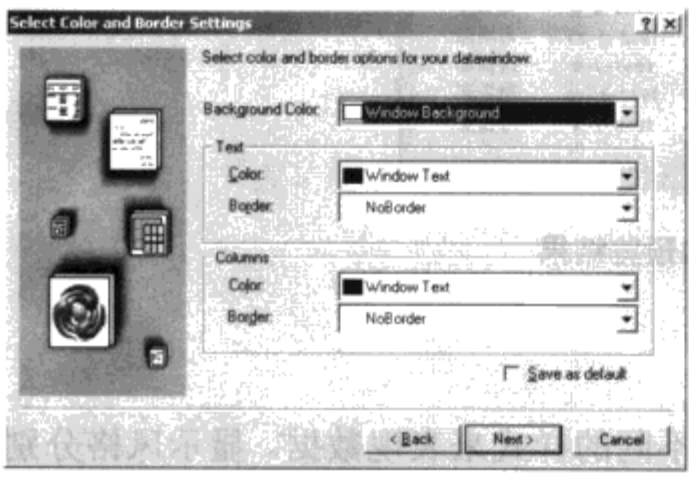


图 9-6　数据窗口的外观颜色窗口

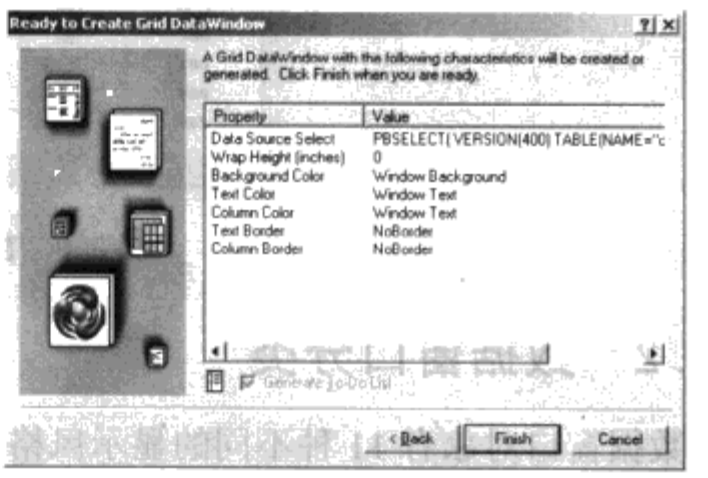


图 9-7　数据窗口配置清单

(11) 出现数据窗口对象设置窗口，如图 9-8 所示。这里可以根据需要对数据窗口的布局进行修改。单击“保存”按钮，在“保存”对话框中键入数据窗口的名字 dw_Grid，完成数据窗口的创建。

(12) 选择 File 菜单中的 New 命令，弹出 New 对话框，在 PB Object 选项卡中双击 Window 图标，然后单击工具栏上的控件图标，从下拉工具列表中选择数据窗口控件。在窗口工作区上单击鼠标左键即可在窗口上添加数据窗口控件(dw_1)，在数据窗口控件的

属性页的 DataObject 文本框中添写 dw_Grid。在窗口的 Open 事件中添加脚本如下表示。

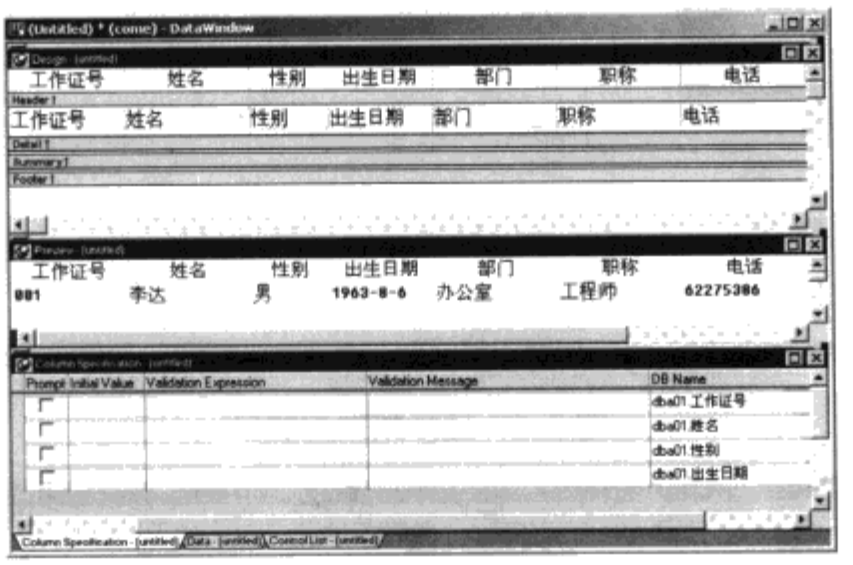
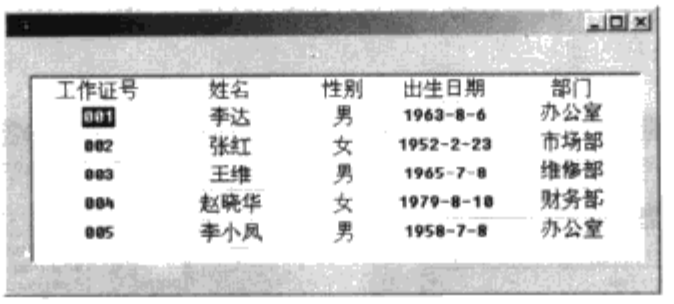


图 9-8　数据窗口对象设置

```
SQLCA.DBMS = ODBC
sqlca.logid = dba
sqlca.logpass = sql
SQLCA.AutoCommit = False
SQLCA.DBParm = Connectstring='DSN=data04'
connect using sqlca;

connect;
dw_1.settransobject(sqlca)
dw_1.retrieve()
```

(13) 单击工具栏上的预览图标 ，在弹出的对话框中选择窗口对象，运行结果如图 9-9 所示。



工作证号	姓名	性别	出生日期	部门
001	李达	男	1963-8-6	办公室
002	张红	女	1952-2-23	市场部
003	王维	男	1965-7-8	维修部
004	赵晓华	女	1979-8-18	财务部
005	李小凤	男	1958-7-8	办公室

图 9-9　窗口的预览结果

9.1.2　数据窗口分类

数据窗口对象有 11 种不同的显示风格，以不同的样式来表现数据。显示风格分别为列表(Tabular)、交叉列表(Crosstab)、自由格式(Freeform)、统计图(Graph)、表格(Grid)、分组(Group)、标签(Label)、分栏(N-Up)、外部文本(OLE 2.0)、超文本(RichText)和复合(Composite)。在创建新的数据窗口对象时，可以在数据窗口显示样式对话框中，通过选择数据窗口的显示样式、设置数据窗口对象和它所包含的对象的属性，构造出各种适合与用户交互的显示界面。下面将分别介绍这 11 种显示样式的表现形式和用途。

1.　列表(Tabular)

在列表风格中，数据按列排列在数据窗口中，每列的上方都有一个标题，说明该列的

数据。每屏显示的数据行数取决于数据窗口的大小，而且运行时用户可以调整各列位置。另外，在每页中可以显示页标题。列表风格的数据窗口常用于数据输入，有时也用于生成报表，如图 9-10 所示。

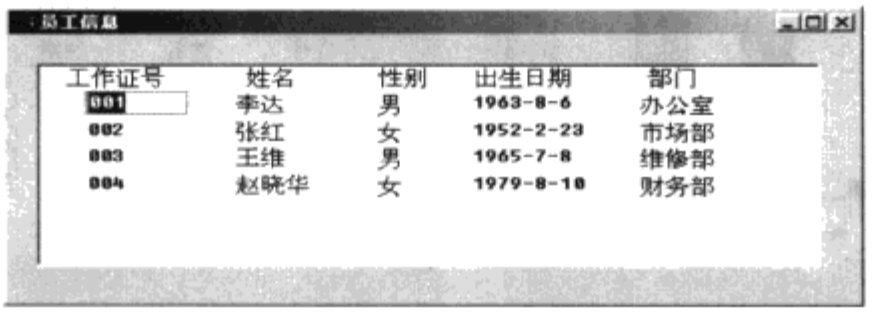


图 9-10　列表风格的窗口

2. 交叉列表(Crosstab)

交叉列表风格支持按行或列分析数据，该风格的数据窗口对数据进行加工处理后以汇总形式展现出来，如图 9-11 所示。

3. 自由格式(Freeform)

自由格式的显示样式十分灵活，所有字段前面都加有字段标题(称为标签)，并且以垂直方式排列在数据窗口中。在数据窗口画板里，可以根据需要来灵活地安排字段、标签以及其他对象的位置。一般情况下，自由格式的数据窗口一屏显示一条记录，常用于单记录输入，如图 9-12 所示。

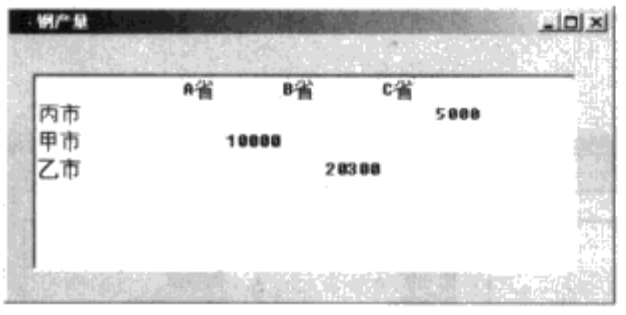


图 9-11　交叉列表风格的窗口

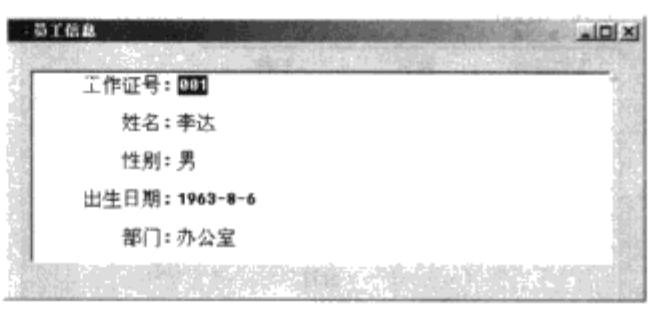


图 9-12　自由格式风格的窗口

4. 统计图(Graph)

统计图风格的数据窗口以统计图的形式表现数据，数据不是通过行和列孤立地显示出来，而是以图形的方式呈现。该风格的数据窗口提供了多种统计图，包括面积图(Area)、条形图(Bar)、列形图(Column)、线形图(Line)、饼图(Pie)、散点图(Scatter)和堆积图(Stacked Bar)等，如图 9-13 所示。

5. 表格(Grid)

在表格风格数据窗口中，数据的行与列之间通过网格线分隔，数据的每一项都位于其中的一个网格中，运行时通过鼠标的拖曳操作可以改变列的宽度，也可以调整列的左右位置，但是在设计数据窗口对象时不能移动列以及改变列标题的左右次序，这一点有别于列表风格和自由风格。表格风格的数据窗口既可用于数据输入，又能作为报表输出，如图 9-14 所示。

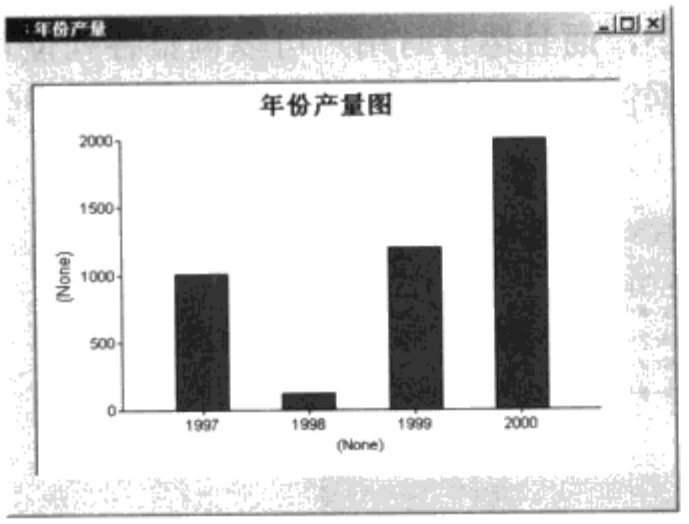


图 9-13　统计图风格的窗口

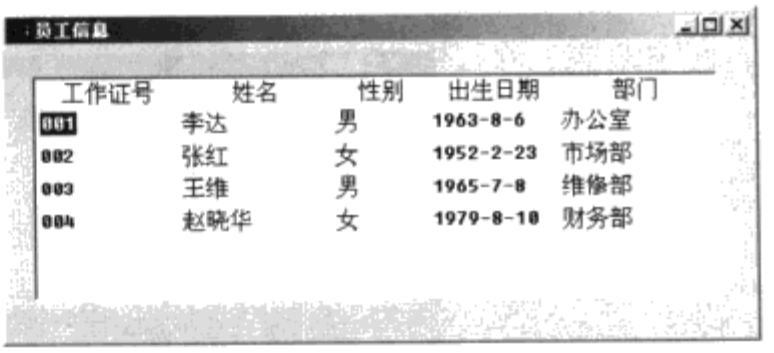


图 9-14　表格风格的窗口

6.　分组(Group)

分组风格的数据窗口提供了一种对数据进行分组的简便途径，在这种风格下，数据被逐个分组，组中可以带有统计数据。可以把它看成是带有分组特性的列表风格，如图 9-15 所示。

7.　标签(Label)

标签风格的数据窗口以标签形式显示每行数据，可以制作各种标签，示例如图 9-16 所示。

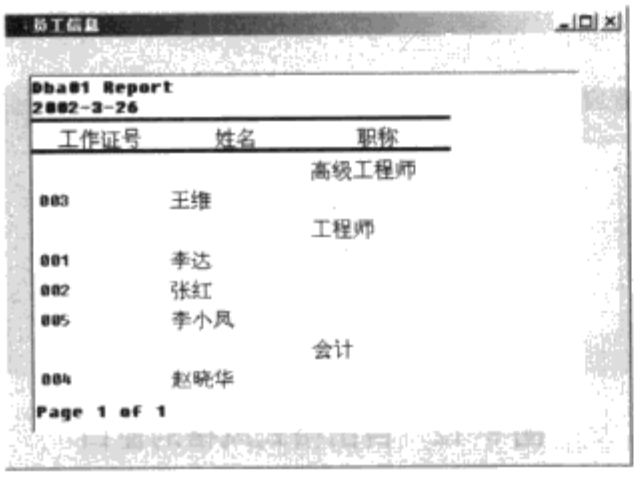


图 9-15　分组风格的窗口

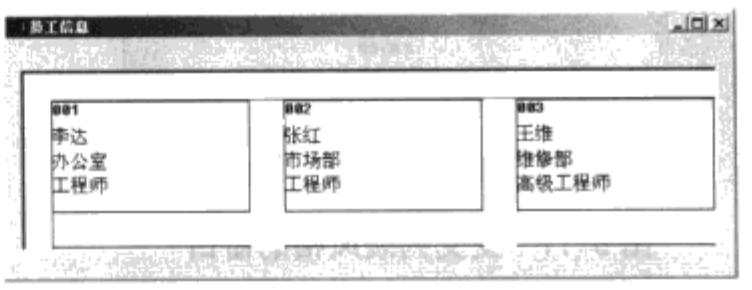


图 9-16　标签风格的窗口

8.　分栏(N-Up)

分栏风格能够在一行中显示多条记录，如图 9-17 所示。

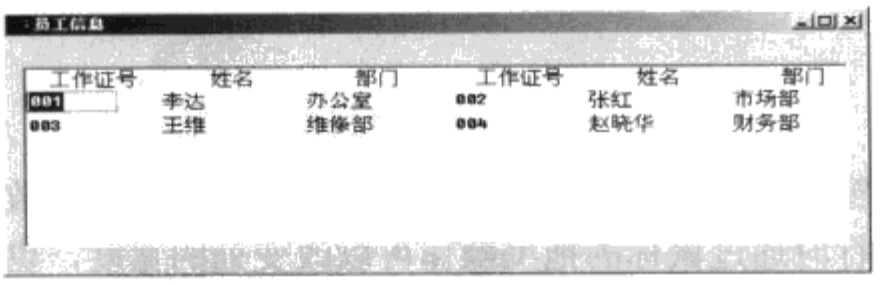


图 9-17　分栏风格的窗口

9. 外部文本(OLE 2.0)

外部文本(OLE 2.0)风格的数据窗口既能够显示非数据库数据(比如 Word 文档)，也能够显示数据库中的 Blob(二进制大对象)列。

10. 超文本(RichText)

超文本风格的数据窗口能够处理超文本数据，它可以利用 Windows 系统的字体、字型和颜色等属性，以丰富多采的形式显示或编辑超文本，并且能够将数据库中的数据插入到超文本文档中。

11. 复合(Composite)

复合风格的数据窗口没有自己的数据源，它通过特殊方式将其他数据窗口对象组合起来，从而创建形式更复杂的数据窗口。

9.1.3 数据窗口画板

数据窗口画板提供了 6 个与数据窗口相关的视图。通过这些视图可以对当前工作数据窗口进行操作，如图 9-18 所示。其中显示了 3 个视图，包括设计视图(the Design view)、预览视图(the Preview view)和属性视图(the Properties view)。

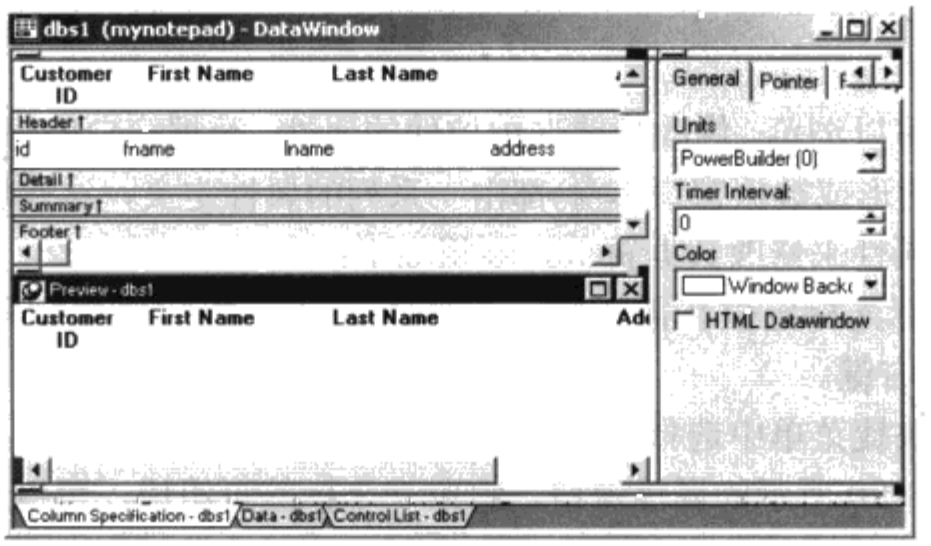


图 9-18　数据窗口的视图

PowerBuilder 的数据窗口画板通过对不同视图之间的协同使用，极大地减轻了开发人员的设计工作，下面具体地解释和说明每个视图的功能和使用方法。

1. 设计视图

设计视图用来实时显示数据窗口对象和数据窗口控件的层次和外观。为表现不同的显示风格，设计视图被划分为标题栏(Header)、细节栏(Detail)、合计栏(Summary)、脚注栏(Footer)4 个工作栏(Bands)，每个工作栏显示数据窗口不同的部分。每个工作栏都在设计视图上分别用箭头指出相应的位置，如图 9-19 所示，其功能如表 9-3 所示。

修改视图栏尺寸的方法是将鼠标光标定位在指示栏位置的分割条(bar)上，然后通过拖动分割条来缩小或扩大指示栏的尺寸。

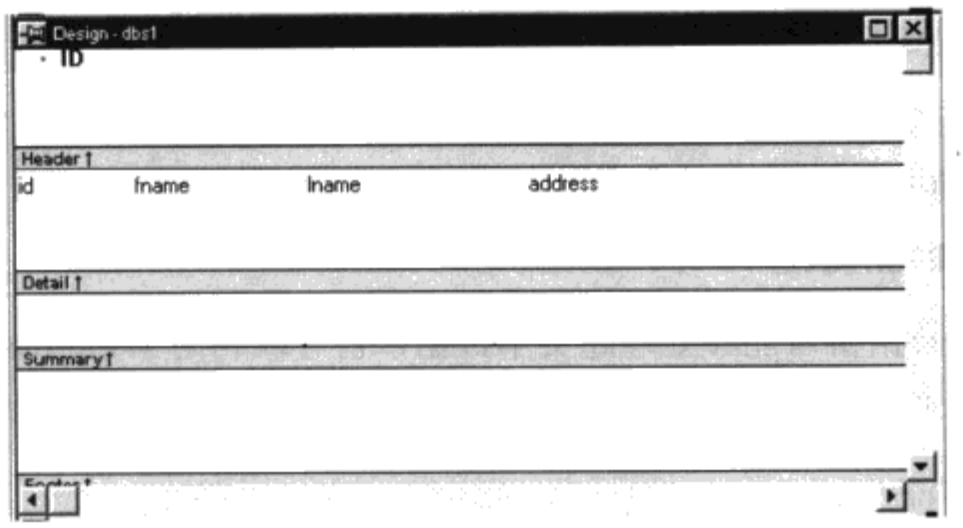


图 9-19　设计视图栏

表 9-3　设计视图栏的功能

栏	栏的使用方法
标题栏	显示在屏幕或页的顶部，例如：报告名称和当前日期
细节栏	显示从数据库和其他数据源的数据
脚注栏	显示在每页底部的信息，例如：页码和页数
合计栏	显示所有数据的合计信息，例如：总计和计算

2. 属性视图

在属性视图上可以浏览、修改当前选中控件的属性。数据窗口的每个对象(例如：文本、列、计算域、栏、图形或数据窗口本身)都由一系列的属性定义，这些属性显示在属性视图上，可以通过改变属性来修改数据窗口，步骤如下：

(1) 首先将鼠标光标定位在需要修改的数据窗口部位上(这里定位在数据窗口中的列上)，然后单击鼠标右键。

(2) 在弹出的快捷菜单中选择 Properties(属性)选项。

(3) 弹出如图 9-20 所示的属性窗口，它由 7 个标签页组成，可以选择不同的标签页，如选择 Edit 页，则 Edit 页的信息将显示在最前面，然后根据不同选项进行修改。

3. 控件列表视图

控件列表视图显示数据窗口对象的所有控件，如图 9-21 所示，在控件列表中可以直接选择需要操作的数据窗口控件，然后就可以在属性列表上进行修改，并可以使用 Ctrl+Alt 组合键来快速进行控件的多选。

实际上，PowerBuilder 的数据窗口画板提供了多种选择数据窗口控件的方法，下面对其做一简要介绍：

(1) 直接选择：在数据窗口的设计视图上单击选择控件，此控件会在四周显示小黑块表示已被选中，以前被选择的控件则恢复正常显示。

(2) 通过鼠标拖动对邻近控件进行多选，方法如下：

- 在被选择的控件上单击。
- 拖动鼠标到需要选择的地方。

- 释放鼠标，此时会显示选中的控件。

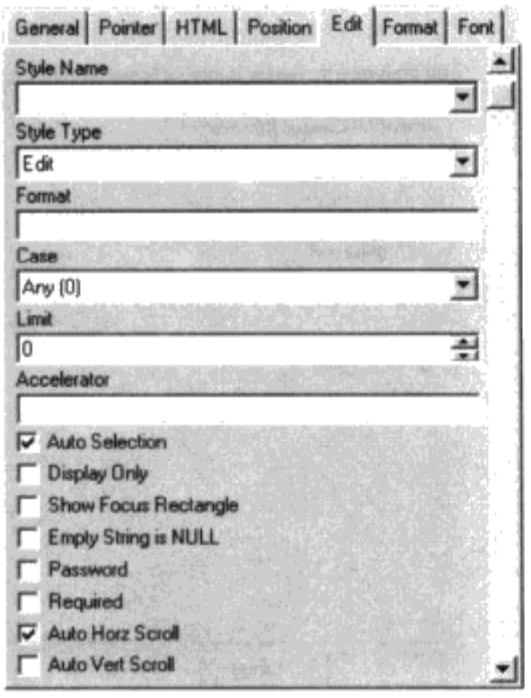


图 9-20　数据窗口属性视图

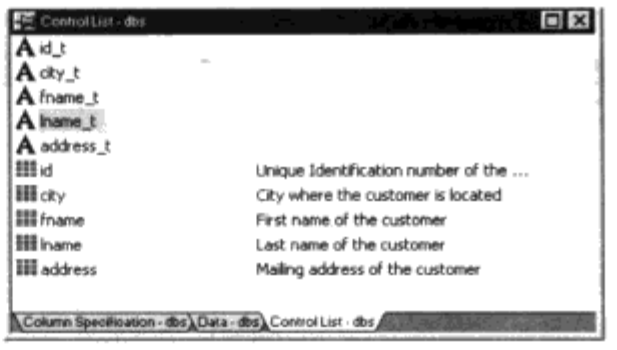


图 9-21　控件列表视图

(3) 通过菜单进行分类选择，具体操作方法如表 9-4 所示。

表 9-4　通过菜单进行分类选择控件

选中的控件	通过菜单选择的方法
所有控件	Edit->Select->Select All
所有文本控件	Edit->Select->Select Text
所有列	Edit->Select->Select Columns
选择当前控件之上的控件	Edit->Select->Select Above
选择当前控件之下的控件	Edit->Select->Select Below
控件当前选择右边的控件	Edit->Select->Select Right
选择当前控件左边的控件	Edit->Select->Select Left

4. 预览视图

预览视图用于显示数据窗口在实际执行时的情况(可带数据)，如图 9-22 所示。使用数据窗口的预览视图可以在设计中对数据进行处理和显示。下面介绍关于预览视图使用中的几个常用问题。

(1) 控制数据窗口打开时预览视图是否显示数据。在使用预览视图时可以通过选择菜单 Design|Option，然后在弹出的 Option 对话框中选择 General 选项卡，当选中 Retrieve On Preview 选项时，打开数据窗口后预览视图中没有数据显示，如图 9-23 所示。

(2) 关于数据缓存。通常情况下，PowerBuilder 第一次检索时，从数据库中得到的数据将会存放在数据缓存中，以后刷新预览视图时，就不会再从数据库中检索数据了，而只是显示存在数据缓存中的数据，这样就避免了频繁访问数据库，极大地提高了使用效率。如果操作的数据库中的数据随时会发生变化，就会使得在预览视图中无法真实地反映当前数据，这时就需要关闭数据缓存选项。

(3) 对数据进行修改。修改存在数据的方法是，首先选中要修改的域，然后输入新值，注意此时数据同样需要遵循有效性规则。

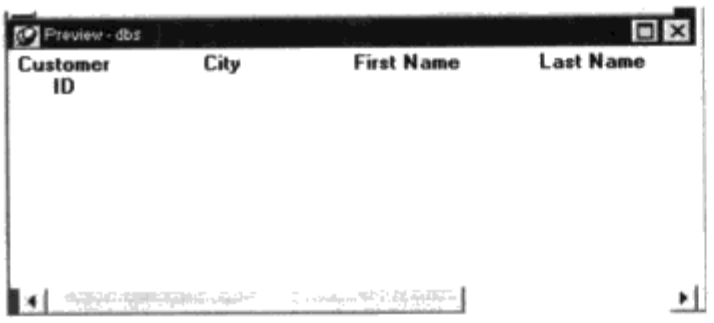


图 9-22 预览视图

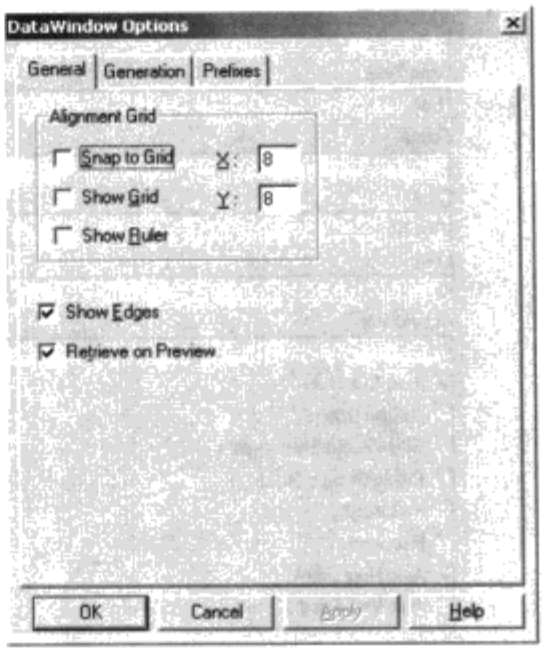


图 9-23 DataWindow Options 对话框

5. 数据视图

数据视图显示数据窗口的数据，并且允许对它进行改变，如图 9-24 所示。在数据视图上能反映出当前数据窗口的数据。

当 Retrieve on Preview 选项处于选中状态时，数据视图显示数据，否则数据视图中无数据显示。另外，在数据视图上选中 ShareData 选项，可使数据视图和预览视图共享数据，即在数据视图中修改数据后可反映到预览视图中，在预览视图中修改数据也可反映到数据视图中。修改 ShareData 方法是在数据视图上单击鼠标右键，然后在弹出的菜单上选择 ShareData 命令。

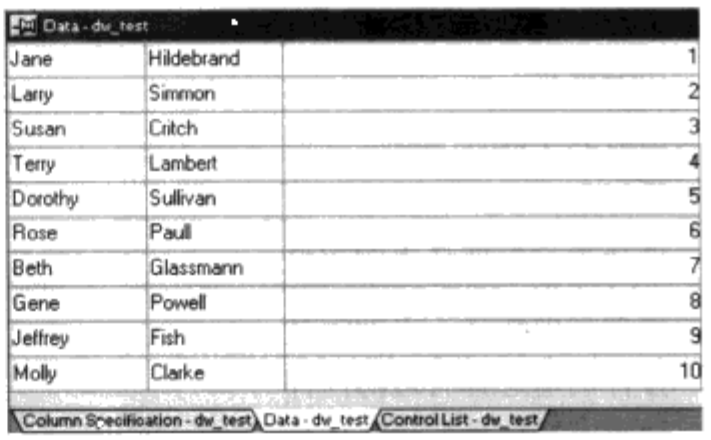
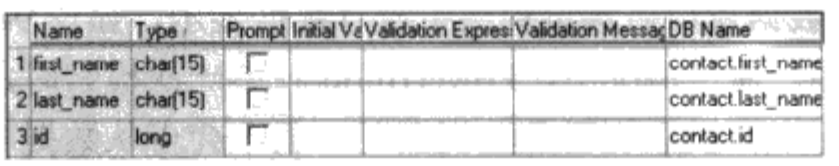


图 9-24 数据视图

6. 列定义视图

列定义视图在数据源中显示不同的列，对于在数据窗口的列，不仅能够进行增加、删除、修改操作，还能够对初始值、有效性验证和某列被检索的控制进行定义。当要将列添加到数据窗口时，可采用拖动方式将列从列定义视图中直接拖动到设计视图。对于外部数据源和存储的数据源，可以对列的名称、类型和长度等进行增加、删除、编辑等操作，如图 9-25 所示。



	Name	Type	Prompt	Initial Va	Validation Expres	Validation Messa	DB Name
1	first_name	char(15)	☐				contact.first_name
2	last_name	char(15)	☐				contact.last_name
3	id	long	☐				contact.id

图 9-25 列定义视图

下面介绍几个有效性规则的定义方法。

(1) 检查当前列的数据是否为负数。

```
Integer(GetText())>0
```

(2) 如果当前列是折扣价，full_price 列是全价，使用下面的有效性规则可以满足当前项是数字且小于 full_price。

```
Match(GetText(), " ^[0-9]+$ ")
AND Real(GetText()<Full_Price)
```

9.1.4 数据窗口数据源

1. Quick Select 数据源

Quick Select 是最简单的一种数据源，它允许从一个表或与该表有外键关联的表中选择一个或多个列。Quick Select 数据源定义出一条简单的 Select 语句，这条语句可以指定选择的列、查询条件及排序方式，但不支持分组、计算列、提取参数等复杂的 SQL Select 功能。快速选择数据源的步骤如下：

(1) 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 DataWindows 选项卡，如图 9-1 所示。双击 Grid 图标，显示如图 9-2 所示的窗口。

(2) 选择 Quick Select 图标，出现如图 9-26 所示的窗口。选择表 dba01 和 dba02，选择字段工作证号、姓名和部门主管。

窗口下面部分的 Sort 行用于指定排序方式。如果数据检索结果按某列排序，那么必须在单击该列下的 Sort 行，然后从下拉列表框中选择所需排序方式(Ascending/Descending)。

窗口下面部分的 Criteria 行用于指定查询条件。查询条件中使用的 SQL 关系操作符有“=”、“>”、“>=”、“<”、“<=”、“<>”、“LIKE”、“IN”和“EXISTS”等。如果未指定操作符而只输入一个值，默认操作符为“=”。如果有多个检索条件，可以使用逻辑操作符 AND 和 OR 来连接。默认同一行不同列的逻辑操作符为 AND，而默认同行同列的逻辑操作符为 OR。

(3) 单击 OK 按钮，出现数据窗口的外观颜色窗口(采用默认值)。单击 Next 按钮，出现数据窗口配置清单窗口。

(4) 单击 Finish 按钮，完成 Quick Select 数据源的创建。

2. SQL Select 数据源

SQL Select 数据源能够从多个表中选择列、指定查询条件、对数据排序、分组、增加计算列和定义提取参数等。在创建数据窗口采用的是 SQL Select 数据源，这里不再重复。

3. Query 数据源

Query 数据源选取 Query 对象作为数据源，Query 对象实际上就是保存在应用库中的 SELECT 语句。定义 Query 对象的目的是为了避免在多个数据窗口中重复使用相同的 SELECT 语句。与定义 SQL Select 数据源相似，在 Query 对象中可以定义检索参数、指定排序/分组方式、定义检索条件等。

(1) 定义 Query 对象

① 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 Database 选项卡，如图 9-27 所示。

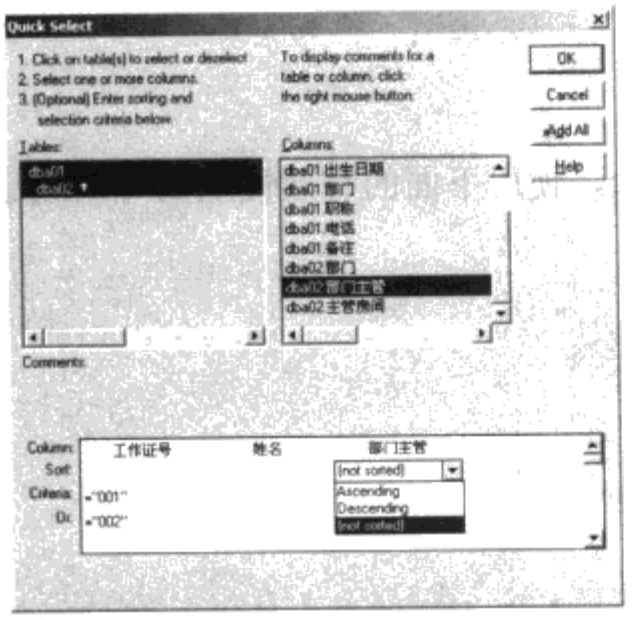


图 9-26 Quick Select 窗口

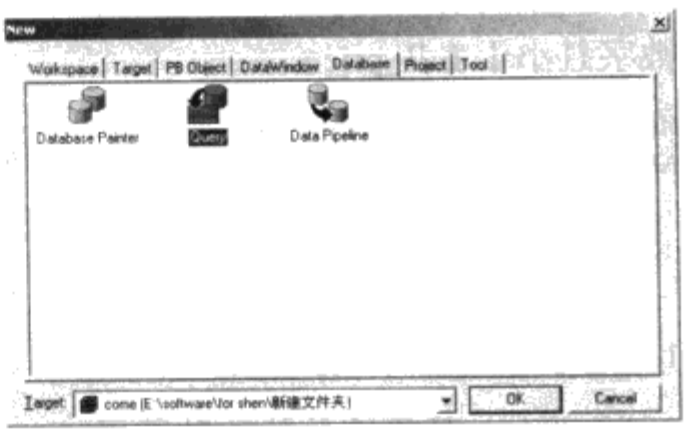


图 9-27 Database 选项卡

② 单击 OK 按钮，弹出 Select Tables 对话框，如图 9-28 所示。选择所需要的表，这里选择表 dba01 和 dba02。

③ 单击 Open 按钮，弹出 Table Layout 对话框，如图 9-29 所示。这个对话框与 SQL Select 数据源的选择数据列窗口相同，用法也是完全一样。定义好选择的数据后，单击“保存”图标，完成 Query 对象的定义。

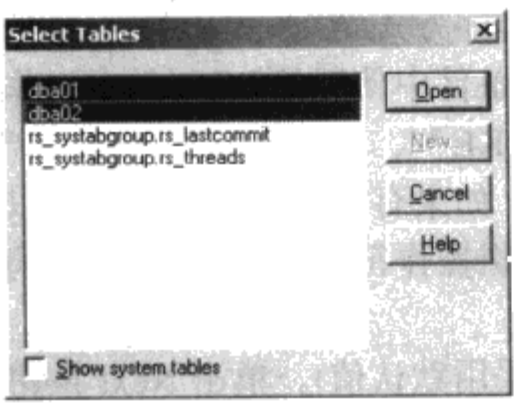


图 9-28 Select Tables 对话框

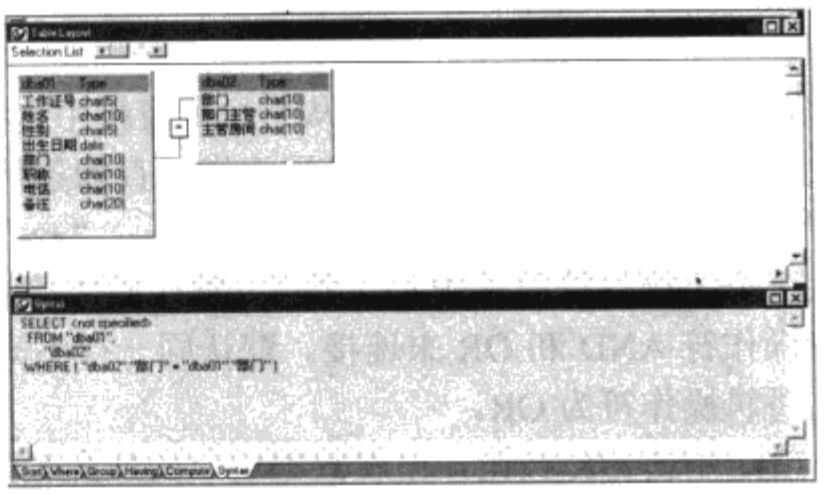


图 9-29 Table Layout 对话框

(2) 定义 Query 数据源

① 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 DataObject 选项卡，如图 9-1 所示。双击 Grid 图标，显示如图 9-2 所示的窗口。

② 选择 Query 图标，弹出如图 9-30 所示的对话框。选择刚定义的 Query 对象(query_01)。

(3) 单击 OK 按钮，出现数据窗口的外观颜色窗口，采用默认值。单击 Next 按钮，出现数据窗口配置清单窗口。

(4) 单击 Finish 按钮，完成 Query 数据源的创建。

4. External 数据源

External 数据源从外部文件(比如文本文件)中提取数据，它是数据窗口惟一不需要连接数据库的数据源，其数据由应用程序生成或由用户输入。定义外部数据源的数据窗口时，

必须定义它的每一列及其数据类型。

定义 External 数据源的步骤如下：

(1) 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 DataObject 选项卡，如图 9-1 所示。双击 Grid 图标，显示如图 9-2 所示的窗口。

(2) 选择 External 图标，在出现的定义列及其数据类型窗口中，定义所需的列以及相应的类型和长度。可以使用 Add、Insert、Delete 按钮来实现增加、插入、删除数据列的操作，如图 9-31 所示。

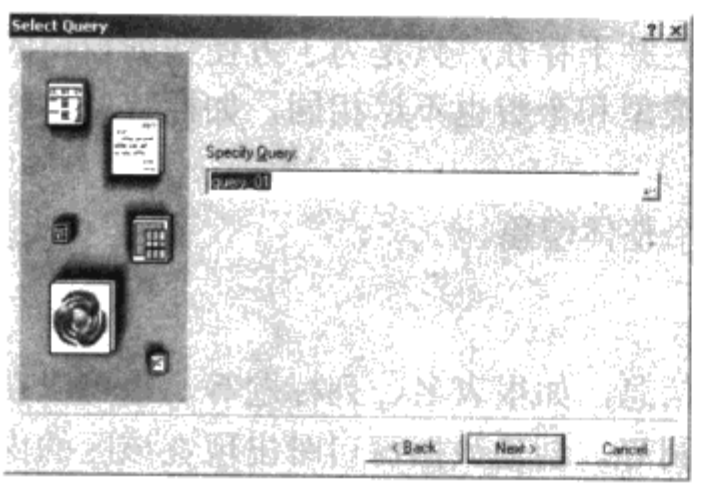


图 9-30　Select Query 对话框

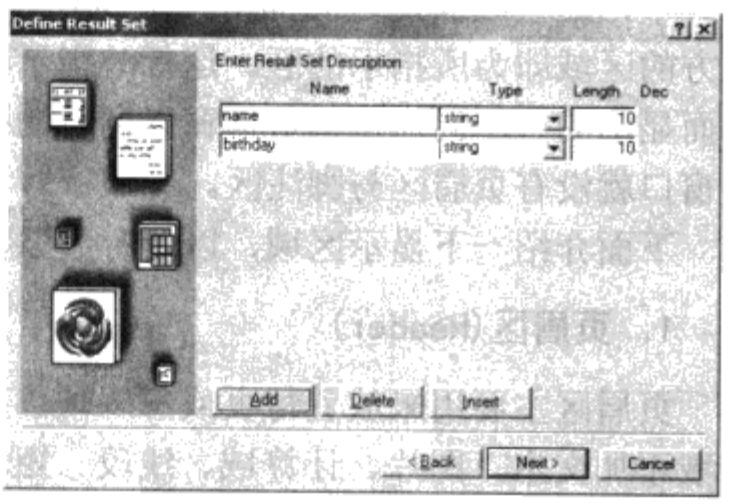


图 9-31　定义列及其数据类型对话框

(3)单击 OK 按钮，出现数据窗口的外观颜色窗口(采用默认值)，单击 Next 按钮，出现数据窗口配置清单窗口。

(4)单击 Finish 按钮，完成 External 数据源的创建。

5. Stored Procedure 数据源

Stored Procedure 数据源(存储过程)是一种高效率的数据源。存储过程是一组保存在数据库中的、经过预先编译和优化的、执行数据库操作的 SQL 语句，数据库系统仅在第一次执行存储过程时，才对 SQL 语句进行编译和优化。数据库系统必须支持存储过程时，存储过程才有效。

定义 Stored Procedure 数据源的步骤如下：

(1) 选择 File 菜单中的 New 命令，弹出 New 对话框，选择 DataObject 选项卡，双击 Grid 图标，选择数据源窗口。

(2) 选择 Stored Procedure 图标，弹出选择存储过程对话框，如图 9-32 所示，选择所需要的存储过程。

(3) 单击 Next 按钮，弹出数据窗口的外观颜色对话框(采用默认值)。

(4) 单击 Next 按钮，弹出数据窗口配置清单对话框。

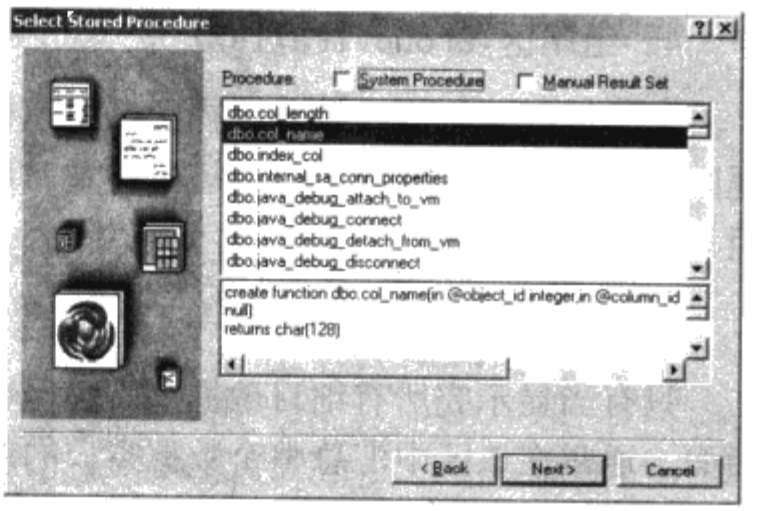


图 9-32　选择存储过程窗口

(5) 单击 Finish 按钮，完成 External 数据源的创建。

9.2 数据窗口对象操作

9.2.1 划分区域

数据窗口的外观与日常使用的报表十分相似，参照报表可以将数据窗口显示区域划分为页眉区、组标题区、细目区、组尾区、汇总区和脚注区。在数据窗口画板工作区的设计模式中，这些区彼此之间都用一条标有区域名称及上箭头的栏隔开(称为区标栏)，每个栏上方的区域即为所指示的区。运行时指示区域的栏并不存在，只是为了方便数据窗口的设计而显示的。不同风格的数据窗口拥有的区域的类型和个数也不尽相同，如标签风格的数据窗口就没有页眉区与脚注区。

下面介绍一下显示区域，以便对数据窗口有个整体印象。

1. 页眉区(Header)

页眉区主要用来放置每页或每屏顶部显示的信息，如报表名、列标题等，也可以在该区域添加文本、图片、计算域、线段、矩形和椭圆等。创建数据窗口时出现在该区的内容与选择的显示风格相关，当选择了列表、表格和分栏风格时，页眉区中显示数据列的列标题。

2. 组标题区(Group Header)

只有选择了分组风格或创建了分组后，组标题区才会自动地放进数据窗口。组标题区中通常放置每组数据名称，如按部门分组时的部门名称。

3. 细目区(Detail)

细目区主要用于显示查询到的数据，数据窗口会根据窗口或报表页的大小尽可能多地安排细目区。细目区的数据可以排列成一行，也可以排列成多行，如列表风格数据窗口把细目区排成两行，如图 9-10 所示。

4. 组尾区(Group Trailer)

与组标题区相似，只有选择了分组风格或创建了分组时，才会自动地把组标题区与组尾区同时显示在数据窗口，在该区中通常放置每组数据的统计与汇总信息。另外，每个分组只有一个组尾区。

5. 汇总区(Summary)

只有当显示完所有细目数据后汇总区中的数据才会出现在最后一屏或最后一页数据的后面。通常在该区中汇总整个数据窗口的统计信息，如数据行总数等。

6. 脚注区(Footer)

脚注区与页眉区相对应，该区中的数据出现在每一屏或每一页的底部。通常在脚注区中放置一些计算列，如显示页码等。默认情况下，数据窗口画板中各区的位置十分紧凑，可以采用下面的方法调整区域的大小：

- 在相应区的区标栏(灰色长条)上按住鼠标左键不放。
- 通过移动鼠标光标来改变相应区域的大小。
- 当相应的区域达到合适大小时放开鼠标左键。

9.2.2　添加和删除列

下面将结合实例，讲解如何添加和删除列，删除姓名列的步骤如下：

(1)　首先打开一个数据窗口，如图 9-33 所示。

(2)　在数据窗口画板的工作区中选择要删除的姓名列。

(3)　按 Delete 键或单击工具栏上的 Delete 图标，选中列即被删除。

在数据窗口中增加列对象的步骤如下：

(1)　在菜单上选择 Insert | Control | Column 命令。

(2)　在需要添加该列的位置上单击鼠标左键，弹出 Select Column 对话框，如图 9-34 所示。

(3)　在对话框中选中要加入的项，单击 OK 按钮。

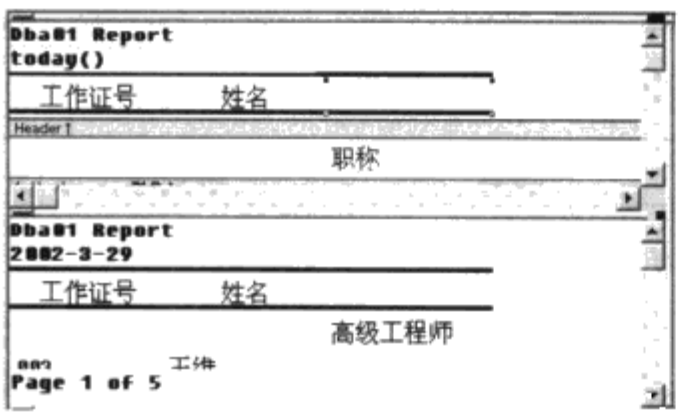


图 9-33　数据窗口

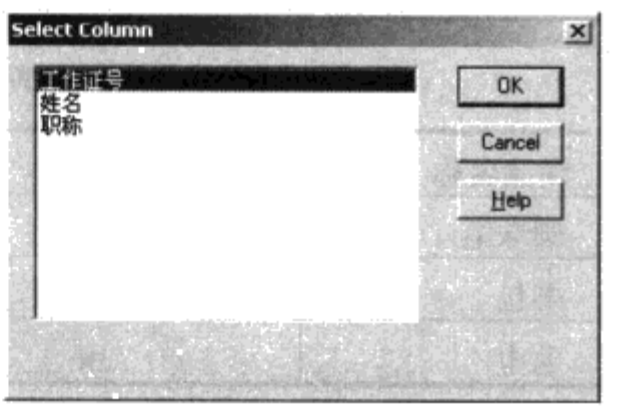


图 9-34　Select Column 对话框

9.2.3　计算字段

在定义数据窗口对象的数据源时，能够定义一个计算列，此时计算列的数值在数据检索时就被计算出来，在下次检索之前，计算列的值保持不变。计算过程由数据库管理系统承担，数据窗口只不过按要求的格式显示出来。因为有时计算列的内容需要随用户数据的输入而调整，所以要在数据窗口中定义计算列(计算字段)。

下面举例说明数据源中定义的计算列与计算字段的区别。假设有一个公司的商品价值表，内容如表 9-5 所示。

表 9-5　商品价值表

商品名称	单价(元)	数量
圆珠笔	2	200
书包	30	20
毛巾	8	120

在定义数据窗口对象的数据源时，定义了一个计算列，得到的结果如表 9-6 所示。

表 9-6 计算列

商品名称	单价(元)	数量	进货总价值(元)
圆珠笔	2	200	400
书包	30	20	600
毛巾	8	120	800

当表的数据发生改变时，发现用这两种方式计算的结果是不同的，分别如表 9-7 和表 9-8 所示。稍作比较就可以发现，计算字段计算的结果正确，而计算列的结果总有错误。下面将介绍如何在数据窗口中定义计算字段。

表 9-7 计算列

商品名称	单价(元)	数量	进货总价值(元)
圆珠笔	2	300	400
书包	30	30	600
毛巾	8	200	800

表 9-8 计算字段

商品名称	单价(元)	数量	进货总价值(元)
圆珠笔	2	200	400
书包	30	30	900
毛巾	8	200	1600

计算字段的定义步骤如下：

(1) 单击工具栏上的数据窗口工具箱下拉按钮展开工具箱，打开如图 9-35 所示的数据窗口工具箱，选择计算域图标。

(2) 单击要放置计算域的位置，弹出如图 9-36 所示的对话框。

- 如果计算域随每行数据而变化，它应该被放置在细目区。
- 如果要计算域显示某个组的统计信息，则应该把它放置到组标题区或组尾区。
- 每页出现的统计信息应该放置在页眉区或脚注区。

(3) 在多行编辑框中为该列输入表达式。表达式可以包括列、其他的计算列及 PowerScript 运算符号函数。如图 9-36 所示，该表达式计算的就是员工的出生年份。

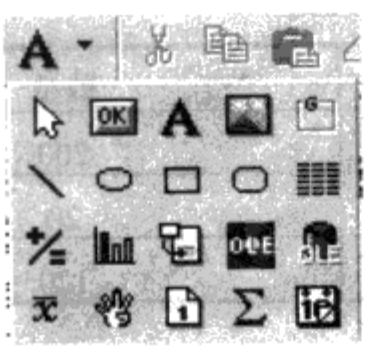


图 9-35 数据窗口工具箱

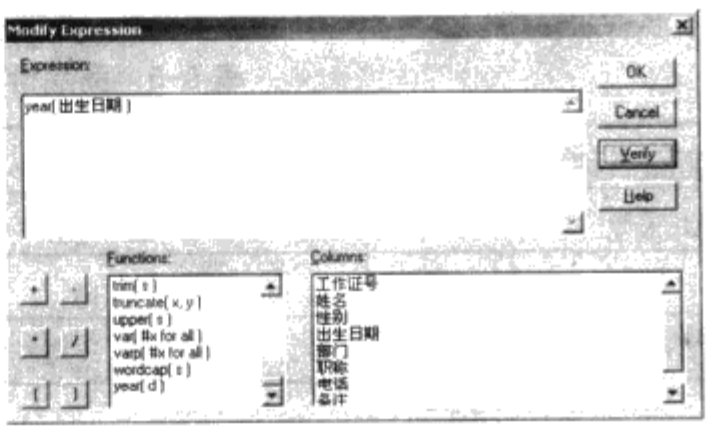


图 9-36 Modify Expression 对话框

(4) 单击 Verify 按钮，可以检验该表达式并确保它的合法性。

(5) 单击 OK 按钮，在数据窗口中添加了一个计算字段，如图 9-37 所示。可以根据需要，用鼠标拖动计算字段来调整它的大小和位置。

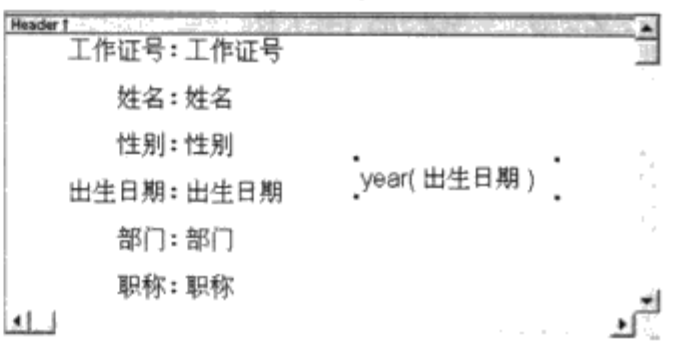


图 9-37　增加了计算字段的数据窗口

9.2.4　过滤数据

过滤数据就是在定义数据窗口的数据源时，通过 WHERE、HAVING 子句或检索参数来限定从数据库中提取的数据。

例如，在开始的时候显示数据库中数据的一部分，当需要的数据条件发生改变时，再显示所需要的数据，这样就需要使用过滤条件。

定义过滤条件的步骤如下：

(1) 选择 Row 菜单中的 Filter 命令，出现 Specify Filter 对话框，如图 9-38 所示。

(2) 在对话框中输入一个逻辑表达式，用来检测每条记录是否满足条件，使逻辑表达式的值为真(True)的记录值，不显示值为假(False)的记录值。

(3) 单击 Verify 按钮，可以检验该表达式并确保它的合法性。

在对话框中添加如下逻辑表达式，并单击 Verify 按钮通过检验。

```
year( 出生日期 ) >= 1970
```

(4) 运行应用程序，出生年份大于 1970 的员工将不被显示。下面列出了数据过滤前后的显示结果，分别如图 9-39 和图 9-40 所示。

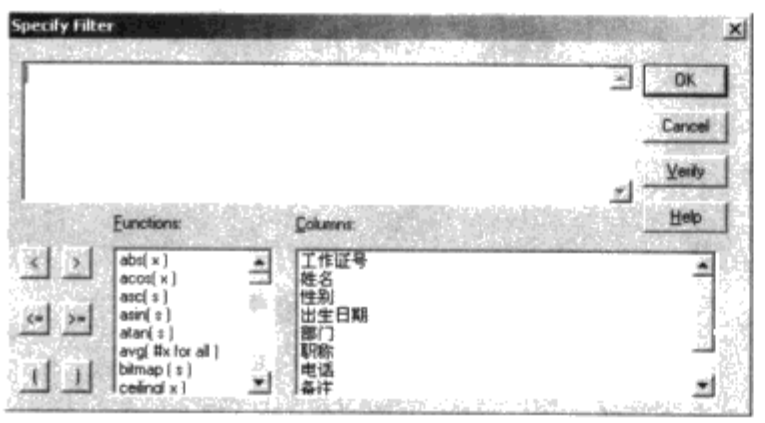
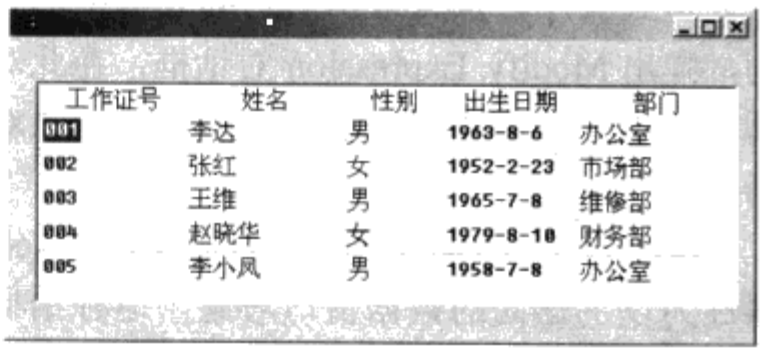
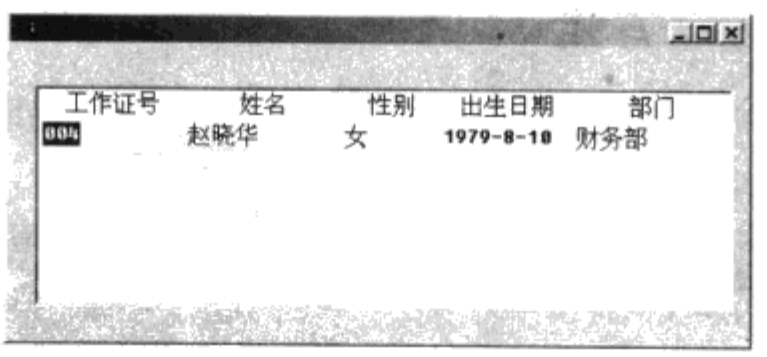


图 9-38　Specify Filter 对话框



工作证号	姓名	性别	出生日期	部门
001	李达	男	1963-8-6	办公室
002	张红	女	1952-2-23	市场部
003	王维	男	1965-7-8	维修部
004	赵晓华	女	1979-8-10	财务部
005	李小凤	男	1958-7-8	办公室

图 9-39　数据过滤前的显示结果



工作证号	姓名	性别	出生日期	部门
004	赵晓华	女	1979-8-10	财务部

图 9-40　数据过滤后的显示结果

9.2.5 数据排序

定义数据源时，通过 Order By 子句可以对检索到的数据进行排序，排序工作由 DBMS 完成。检索出数据后需要按不同顺序查看数据时，可以通过指定数据窗口的排序方式完成，此时的排序工作由客户端的数据窗口完成。数据窗口的排序功能是在客户端对该组数据进行排序，这样有利于减轻 DBMS 的负担，还可以根据各种不同的表达式进行排序。

下面结合一个实例来讲解数据排序的实现。

(1) 使用一个已经定义好了的数据窗口，如图 9-41 所示。

(2) 选择 Rows 菜单中的 Sort 命令，弹出 Specify Sort Columns 对话框，如图 9-42 所示。

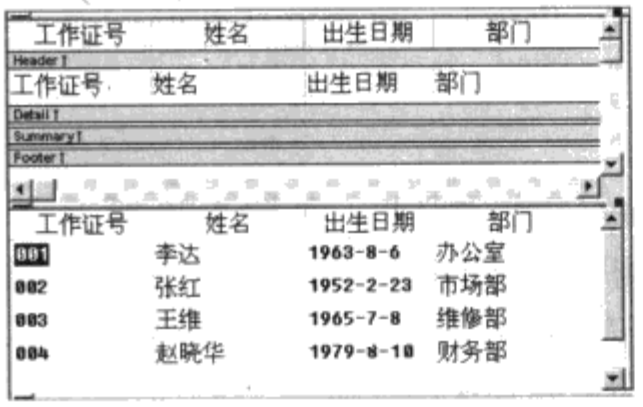


图 9-41 定义好了的数据窗口

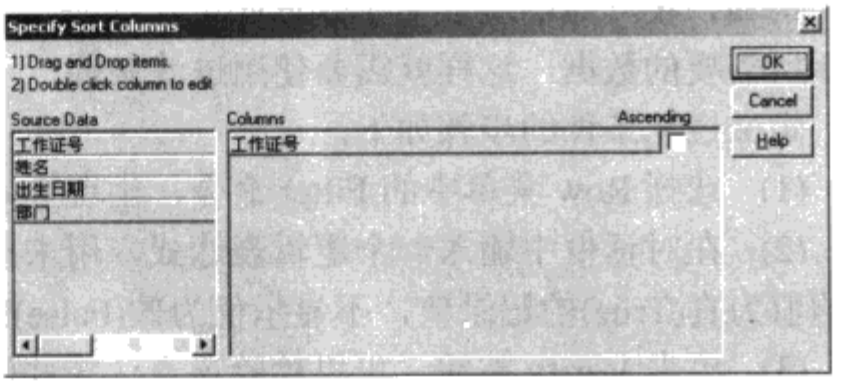


图 9-42 Specify Sort Columns 对话框

(3) 将要排序的列依次从左边的列表框中拖曳到右边的列表框中，此时每个排序项后面的 Ascending 列都有一个复选框，选中该复选框表示按升序排序，否则按降序排序。这里将工作证号按降序排列。如果需要按某个表达式排序，那么双击右边列表框中的一列，弹出 Modify Expression 对话框，在此对话框中键入所需要的表达式，然后关闭该对话框。

(4) 设定这个数据窗口控件的 DataObject 属性为以上建立的数据窗口对象，运行结果如图 9-43 所示。

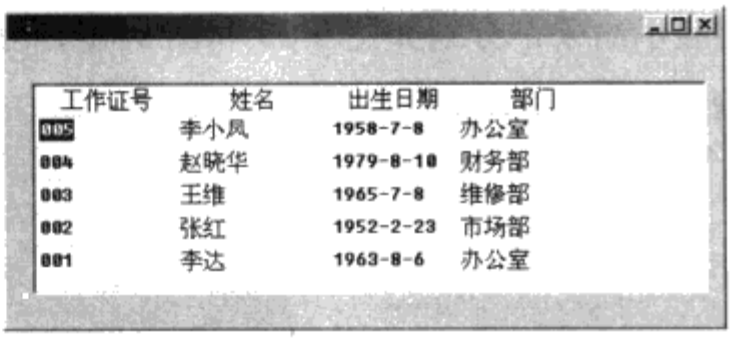


图 9-43 数据排序后的运行结果

9.2.6 指定检索数据方式

通过使用检索条件和指定过滤方式，可以控制从数据库中提取出的数据量。这样既节约系统资源，又能够得到较快的响应速度。当检索出大量数据时，为了避免等待，可以让数据窗口只提取当前所需显示的数据，查看其他数据时，再把相应数据提取到数据窗口中。

指定检索数据的方式如下：

选择 Rows 菜单中的 Retrieve Option 命令，Retrieve Option 选项下面有两个子选项，分别是 Row As Needed 选项和 Row To Disk 选项。

Row As Needed 选项使数据窗口只提取显示数据所需要的行，满屏时可以滚动浏览数据，而不需要等待所有的数据都检索完成后才开始使用数据。因为数据还没有全部准备好，

所以在滚动浏览的时候，可能会有一定的滞后，因为此时数据窗口正在调用下一批数据行。这种方法很适合用户使用。

Row To Disk 选项把多余的行缓存到磁盘上，需要时才调入内存使用。此选项可以解决因结果数据过大而使客户机的内存耗尽或出现问题。

9.3　数据窗口控件的属性、事件和函数

使用数据窗口控件是应用程序在窗口中展示数据窗口对象的惟一途径，数据窗口控件与数据窗口对象的结合构成了应用程序访问和操作数据库数据的主要手段。下面将详细介绍数据窗口控件的属性和事件。

9.3.1　数据窗口控件的属性

与其他控件类似，数据窗口控件也有一组属性，通过设置这组属性可以决定数据窗口控件的外观和行为。比如，数据窗口控件是否带有标题栏，是否带有水平和垂直滚动条。若带有标题栏，那么标题栏上是否带极小化、极大化和关闭按钮，是否带有控制菜单以及运行时是否允许用户改变数据窗口的大小等。

在设计窗口时可以设置数据窗口控件的属性，方法如下：

用鼠标右键单击数据窗口控件，从弹出的菜单中选择 Properties 菜单项，弹出 Properties 对话框，如图 9-44 所示。

下面说明属性对话框中各选项的意义。

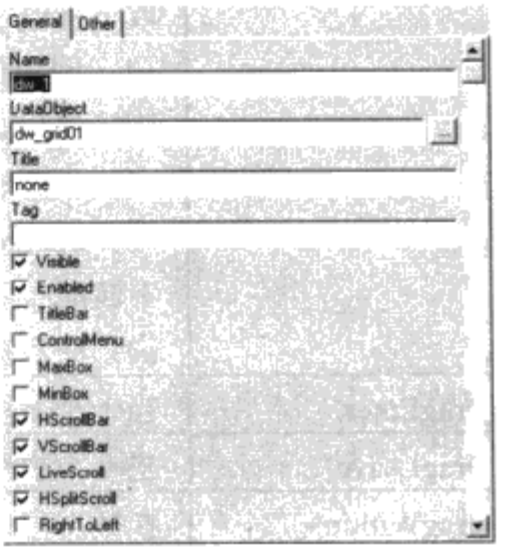


图 9-44　Properties 对话框

- Name：该数据窗口控件的名称通常使用 dw_前缀。
- DataObject：该数据窗口控件相关联的数据窗口对象名称。
- Title：数据窗口控件的标题。
- HScroll Bar：选中该复选框时显示水平滚动条。
- VScroll Bar：选中该复选框时显示垂直滚动条。
- Visible：选中该复选框时该控件可见。
- HSplitScroll：选中该复选框的同时，系统自动选中 HScroll Bar 复选框，其功能是在数据窗口控件中同时显示两个数据窗口域，通过拖动滚动条上的分隔线改变这两个域的大小。
- LiveScroll：选中该复选框，运行程序，当拖动滚动条上的滑块时，数据窗口的内容也随之滚动；未选中该复选框时，拖动滑块并放开鼠标键时才调整显示。

在 Other 选项卡中，可以设置该数据窗口控件的大小和位置。

9.3.2　数据窗口控件的事件

数据窗口的强大功能不仅表现在数据窗口对象有丰富的显示风格和灵活的数据源上，

还表现在数据窗口控件的众多事件上。透彻地理解数据窗口的事件有助于精细地控制数据窗口的行为。不同事件触发的时机不同，返回值的意义各异，只有充分理解事件的这些特性，才能根据需要灵活地控制程序。数据窗口控件的常用事件、触发时机及返回值的意义如表 9-9 所示。

表 9-9　数据窗口控件的常用事件、触发时机及返回值

事件	触发时机	返回值
Clicked	单击事件，当单击某个不可编辑字段或在数据窗口的字段间单击时触发	0：(默认值)继续随后的处理过程 1：终止处理过程
Constructor	构造事件，紧挨着窗口的 Open 事件，在窗口的 Open 事件前触发	
Destructor	紧接在窗口的 Close 事件后面触发	
DoubleClicked	当双击某个不可编辑字段或在数据窗口控件的工作区中双击时触发	
DragDrop	当把拖动着的对象放到数据窗口控件时触发	
DragEnter	当一个被拖动对象进入数据窗口控件时触发	
DBError	调用数据窗口对象函数 Retrieve()或 Update()而导致某种数据库错误时触发，该事件的参数 sqldbcode 中包含由数据库厂商提供的特定出错代码，如果厂商未提供出错代码，则 sqldbcode 取下列值之一： -1：因为事务对象中未正确提供某些值而不能连接到数据库 -2：不能连接到数据库 -3：update 或 Retrieve 中指定的键值与现有行不匹配 -4：向数据库中写 Blob 型数据时失败	0：(默认值)显示保存在事务对象的 SQLErrText 属性中的出错信息 1：不显示出错信息
DragEnter	当一个被拖动对象进入数据窗口控件时发生	
DragLeave	当一个被拖动对象离开数据窗口控件时发生	
DragWithin	在数据窗口中拖动某一对象时发生	
EditChanged	当在某个可编辑字段上输入内容时，每按一次键就触发一次	
Error	当发现数据窗口对象的数据或属性表达式有错时发生	
GetFocus	当焦点切换到数据窗口控件时触发	
ItemChanged	当修改完某一可编辑字段，然后通过按回车键、Tab 键、箭头键或单击其他字段等操作使当前字段失去输入焦点时(输入值通过有效性检查后)触发。如果该字段的新值与修改前的原有值相同，则不触发该事件	0：(默认值)，接收当前数据值 1：拒绝输入值且不允许移走焦点 2：拒绝输入值但允许移走焦点

续表

事件	触发时机	返回值
ItemFocusChanged	当焦点从一个可编辑字段切换到另一个时触发	
ItemError	当修改了字段、然后移走焦点、但数据未能通过该列的有效性检查时触发	0：(默认值)，拒绝输入值并显示有效性验证出错信息，同时不允许移走焦点 1：拒绝输入值，但不显示出错信息，同时不允许移走焦点 2：接收输入值 3：拒绝输入值，但允许移走焦点
LoseFocus	当数据窗口控件本身失去焦点时触发	
PrintEnd	数据窗口打印结束后触发	
Other	当非 PowerBuilder 事件的 Windows 消息发生时触发	
UpdateEnd	当数据窗口更新了数据库后触发	
PrintPage	在数据窗口中的每页已经被格式化并准备打印前触发	0：不跳页 1：跳页
PrintStart	在数据窗口开始打印前触发	
RetrieveEnd	当为数据窗口检索完数据后发生	
RetrieveRow	每检索一条记录并送到数据窗口后触发	0：(默认值)继续随后的检索 1：终止检索过程
RetrieveStart	就在为数据窗口检索数据前触发	0：(默认值)继续检索 1：不执行检索 2：从数据库中检索数据前不复位数据及缓冲区
SQLPreview	在调用 Retrieve、Update、ReselectRow 函数后，SQL 语句被发送到 DBMS 前触发	
UpdateStart	调用 Update()函数后、修改数据库数据前触发	0：(默认值)继续更新过程 1：不执行更新
ButtonClicking	当数据窗口对象上按钮对象的 SuppressEvent Processing 属性设置为 no 时，单击该按钮时触发。该事件在系统处理按钮的默认动作前触发	返回值为 1 不执行按钮的指定动作，但仍会触发 ButtonClicked 事件
ButtonClicked	当数据窗口对象上按钮对象的 SuppressEvent Processing 属性设置为 no 时，单击该按钮时触发。该事件在系统处理完按钮的默认动作后触发	

9.3.3 数据窗口控件的常用函数

1. 插入行与删除行

(1) 插入行 InsertRow()

InsertRow()函数在数据窗口的主缓冲区中插入一个空行，其语法格式为：

```
dwcontrol.InsertRow( row )
```

row 是个长整数(Long 型)，它指定插入行的插入位置。row 等于 0 时有特殊含义，它表示在主缓冲区最后一行后插入一行。

InsertRow()函数执行成功时，返回一个 Long 型值指示插入行的行号，执行失败时返回-1。

例如：在数据窗口 dw_1 中插入一行，并让该行成为数据窗口中的第 15 行，使用语句为：

```
dw_1.InsertRow(15)
```

(2) 删除行 DeleteRow()

DeleteRow()函数在数据窗口的主缓冲区(即显示在用户面前的数据中)中删除一行，语法格式为：

```
dwcontrol.DeleteRow( row )
```

row 为要删除行的行号，row 为 0 时删除当前行。

DeleteRow()函数执行成功时返回 1，失败时返回-1。

例如：删除数据窗口控件 dw_1 中的第 15 行，使用语句为：

```
dw_1.DeleteRow(15)
```

2. 得到与设置当前行和列

当前行和当前列是目前正操作的行和列，在编辑数据时，也就是编辑数据项所在的行和列。

(1) 得到当前行号 GetRow()

GetRow( )函数返回当前行的行号。语法格式为：

```
dwcontrol.GetRow()
```

该函数返回一个 Long 型值，表示当前行号。如果没有选中任一行，则返回 0，出错时返回-1。

(2) 得到当前列号 GetColumn()

GetColumn( )返回当前列号。语法格式为：

```
dwcontrol.GetColumn()
```

该函数返回一个 Integer 型值，表示当前列号。返回值为 0 表示无当前列，返回值为-1 表示操作失败。

(3) 设置某行为当前行 SetRow()

SetRow()函数把参数 row 指定的行设置成当前行。语法格式为：

```
dwcontrol.SetRow( row )
```

设置成功时返回 1，否则返回-1。如果 row 的值小于 1 或大于数据窗口中的总行数，则 SetRow()函数执行失败。

SetRow()函数可能触发下述事件：

- ItemChanged 事件
- ItemError 事件
- ItemFocusChanged 事件
- RowFocusChanged 事件

(4)　设置某列为当前列 SetColumn ()

SetColumn()把参数 column 指定的列设置为当前列。语法格式为：

```
dwcontrol.SetColumn( column )
```

设置成功时该函数返回 1，否则返回-1。

如果 column 的值小于 1 或大于数据窗口中的总列数，则 SetColumn()函数执行失败。

参数 column 可以是个列号(Integer)，也可以是列名(String)。

SetColumn( )函数可能触发下述事件：

- ItemChanged 事件
- ItemError 事件
- ItemFocusChanged 事件

3．得到与设置数据项的值

脚本中可以使用数据窗口控件的对象函数访问指定行和列数据项的值。本节介绍的函数主要针对数据窗口缓冲区的操作，它们不访问漂浮在数据窗口上编辑控件中的数据。

(1)　得到数据项的值 GetItem 簇函数

GetItem 簇函数从指定的行和列取得数据项的值，对不同数据类型的列使用的 GetItem 簇函数也不同。对于字符串、日期、日期时间、时间、数值和 Decimal 等各种类型都有相应的一个函数，且各函数的返回值与其函数名后缀相同。例如 GetItemString()函数的返回值类型为 String。GetItem 簇函数语法格式为：

```
dwcontrol.GetItemString( row, column {,dwbuffer, originalvalue} )
dwcontrol.GetItemDate( row, column {,dwbuffer, originalvalue} )
dwcontrol.GetItemDateTime( row, column {,dwbuffer, originalvalue} )
dwcontrol.GetItemTime( row, column {,dwbuffer, originalvalue} )
dwcontrol.GetItemNumber( row, column {,dwbuffer, originalvalue} )
wcontrol.GetItemDecimal( row, column {,dwbuffer, originalvalue} )
```

①　row 是个长整型数，指明数据所在行的行号。

②　column 指明数据所在的列，既可以使用一个整数指明列号，也可以使用字符串指明列的名称。

③　dwbuffer 和 originalvalue 是两个可选参数，其意义如下：

dwbuffer 指明从哪个缓冲区中读取数据，有效取值为：

- Primary!：默认值，从主缓冲区中读数据。
- Delete!：从删除缓冲区中读数据。
- Filter!：从过滤缓冲区中读数据。

Originalvalue：指示函数是返回当前值，还是返回最初从数据库中检索到的值。该参数的有效取值为：

True：返回原始值，即返回最初从数据库中检索到的值。

False：默认值，返回当前值。

例如，使用下面的语句得到数据窗口控件 dw_employee 主缓冲区中第 3 行、emp_name 列的当前值，该列的数据类型为 String：

```
String Lname
LName = dw_employee.GetItemString(3, emp_name)
```

下面的语句得到数据窗口控件 dw_employee 删除缓冲区中第 3 行、emp_name 列的当前值，该列的数据类型为 String：

```
String Lname
LName = dw_employee.GetItemString(3, emp_name, Delete!)
```

下面的语句得到数据窗口控件 dw_employee 删除缓冲区中第 3 行、emp_name 列的原始值，该列的数据类型为 String：

```
String Lname
LName = dw_employee.GetItemString(3, emp_name, Delete!, True)
```

(2) 设置数据项的值 SetItem()

SetItem()函数为指定行和列的项赋值，直接将这个值放置到数据窗口的主缓冲区中而不通过校验规则。

SetItem()函数语法格式为：

```
dwcontrol.SetItem( row, column, value )
```

- row 是个长整型数，指明赋值项所在的行。
- column 是个整数或字符串，代表列号或列名，用于指明赋值项所在的列。
- value 是要为数据项设置的数值。

使用 SetItem()函数时，value 的类型必须与列的数据类型完全匹配，否则将引发严重错误，导致应用程序失败退出。SetItem()函数执行成功时返回 1，失败时返回-1。

4. 清除所有行

Reset( )函数清除数据窗口中的所有行，语法格式为：

```
dwcontrol.Reset( )
```

该函数执行成功时返回 1，否则返回-1。

数据窗口在检索数据前自动清除原有数据，因此检索前不需要调用该函数。

5. 修改过滤条件及过滤数据

使用 SetFilter()函数可以在程序中动态地修改数据窗口的过滤条件，修改之后，应用程序将调用 Filter()函数过滤数据。

SetFilter()函数的语法格式为：

```
dwcontrol.SetFilter(format)
```

format 是个字符串，其值是作为过滤条件的逻辑表达式，表达式中可以包括列名或列号。

SetFilter()函数执行成功时返回 1，失败时返回-1，一般很少使用该返回值。

执行 SetFilter()函数后，新的过滤条件取代原有的过滤条件。定义了新的过滤条件后，使用 Filter()函数把不满足条件的数据移动到过滤缓冲区中。

Filter()函数的语法为：

```
dwcontrol.Filter()
```

过滤条件由列名或列号、关系操作符(“<”、“=”、“>”等符号)、数值及逻辑运算符(AND、OR、NOT)组成，使用括号改变逻辑表达式的运算次序。

下面是几个过滤条件的示例：

```
item_id > 5
NOT item_id = 5
(NOT item_id = 5) AND customer <> "李四"
item_id > 5 AND customer = " 张"
```

下面的语句定义一个过滤表达式，并把该表达式设置为 dw_Employee 的过滤条件。数据窗口中只显示那些 cust_qty 列的值大于 120、cust_code 列的值大于 30 的行：

```
string Dwfilter
DWfilter = cust_qty > 120 and cust_code >30
dw_Employee.SetFilter(DWfilter)
dw_Employee.Filter()
```

过滤条件中不能包含变量，但可以通过构造字符串的方法把变量的值结合到过滤条件中。例如：

```
string var1
var1 = ME
dw_Employee.SetFilter(emp_state = '+ var1 + ')
```

9.4　数据窗口设计实例

9.4.1　增加数据窗口数据

本例创建一个增加零件类型的表单，在此窗口中可以增加零件类型信息，提供自动检索和信息提示，还实现了增加数据并存储等功能。主要使用的数据窗口事件有数据滚动(scrolltorow(x))、单击数据窗口(dw_1.clicked(xpos,ypos,row,dwo))事件。

需要注意的是，示例使用的数据窗口是自由风格的。

窗口提供的辅助功能如下：

- 显示当前日期。
- 显示当前记录号。
- 数据窗口自动检索相关的类型信息，而无须用户输入。
- 部分填表框设置下拉式列表，提供数据项以供用户选择。

注意： 为用户提供强大的系统功能是应用程序除了自身固有的功能以外必须完善和

重视的一部分内容。这将关系到用户使用该应用软件的方便程度，以至于影响软件的友好程度。

1. 建立数据窗口

(1) 单击工具栏中的 New 图标，在弹出的窗口中选择 DataWindow 选项卡。在该窗口中选择 Freeform，即自由风格的显示风格，如图 9-45 所示。

(2) 单击 OK 按钮弹出如图 9-46 所示的选择数据源对话框，在此对话框中选择 Quick Select 图标。

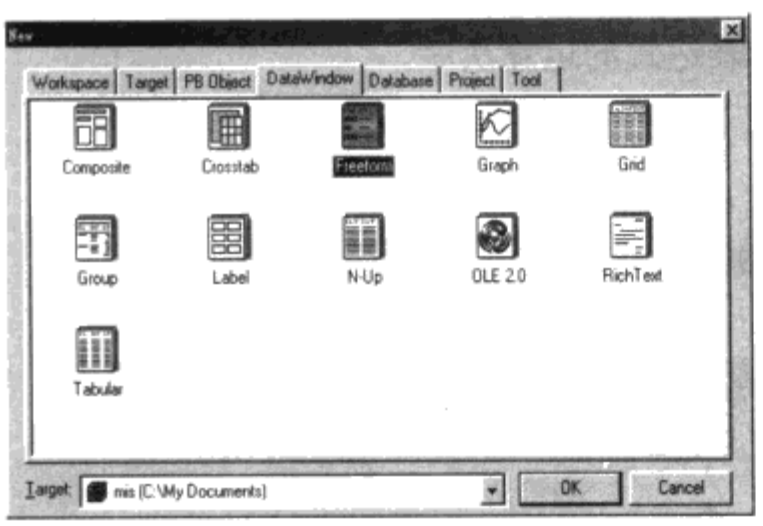


图 9-45 选择 Freeform

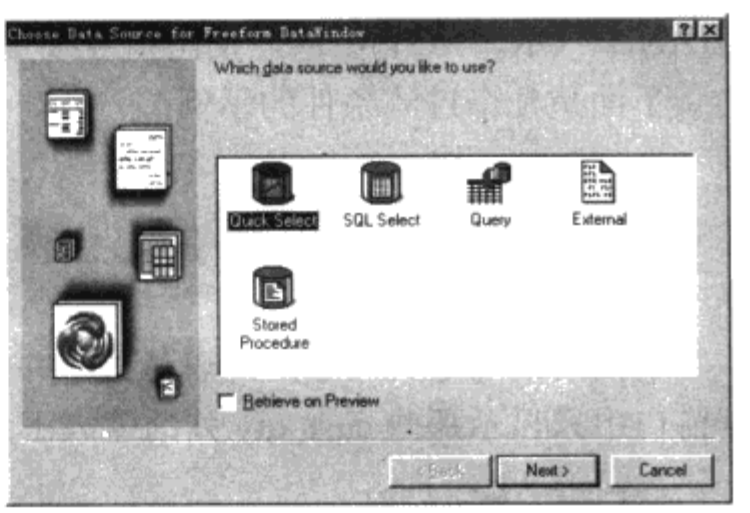


图 9-46 选择数据源对话框

(3) 在随后出现的 QuickSelect 面板中的 Tables 栏下，选中 cc_ljklx(零件库类型信息文件表)，在 Columns 栏中选中所有字段，如图 9-47 所示。

(4) 在设置数据窗口的颜色和边界的选项卡中，设置背景颜色；文本(Text，字段名称)的边界设置为阴影式(ShadowBox)；把数据项(Column)的边界设置为下凹式(Lowered)，如图 9-48 所示。

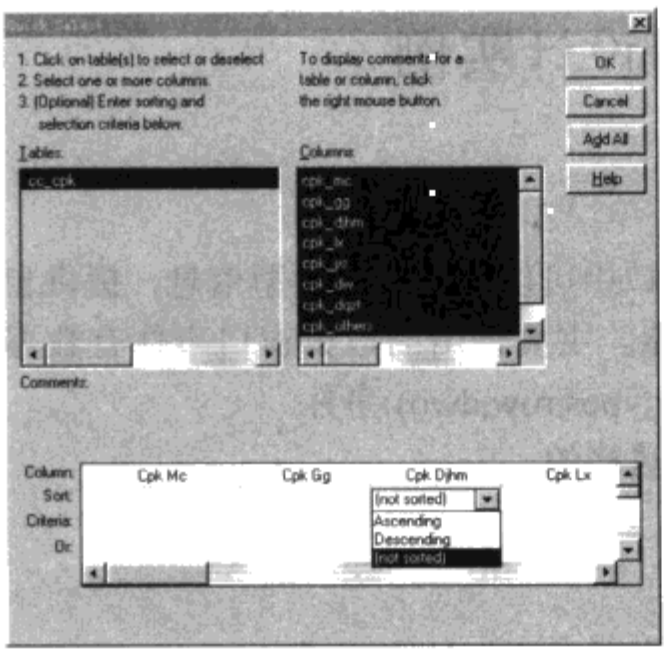


图 9-47 选择所有字段

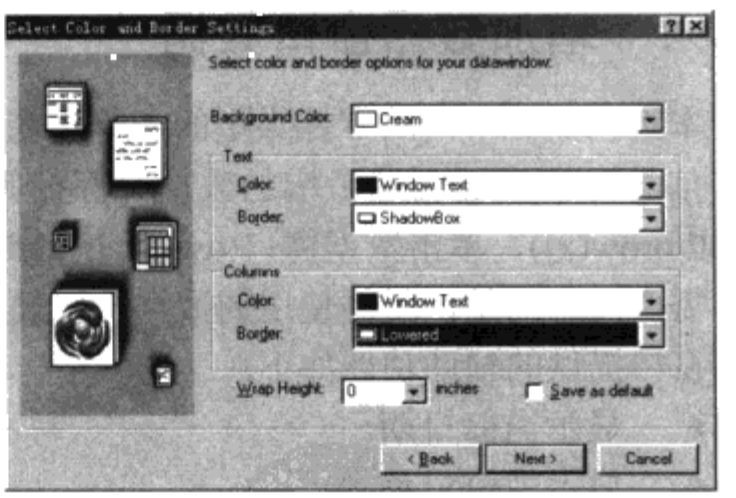


图 9-48 设置数据窗口的颜色和边界的对话框

(5) 在数据窗口设计画板中，调整各字段的显示属性，把关键字用较为醒目的颜色区别表示。

2. 建立窗体

(1) 在 PowerBuilder 的工具栏中单击 New 图标，在弹出的窗口中选择 PB Object 选项卡，在其中选择 Window 图标，单击 OK 按钮，如图 9-49 所示。

图 9-49　选择 Window 图标

(2) 然后，打开窗体的数据画板。将该窗体命名为 w_ljklx_add，将标题设为“零件物料信息增加窗口”。窗体中的控件及其属性设置如下：

```
//数据窗口：dw_1
//命令按钮：cb_1~ cb_5
cb_1.text=增加
cb_2.text=向前翻
cb_3.text=向后翻
cb_4.text=确认
cb_14.text=退出
//单行文本框：sle_1、sle_2
//静态文本框：st_1
st_1.text=当前记录号
```

(3) 最后编写 Open 事件脚本及各控件脚本。

① 增加事件(cb_1.clicked())的脚本

在窗口的 Open 事件中，需要让数据窗口进行读取数据。设置时间和统计记录条数等操作，并且将数据项滚动到当前记录。脚本如下：

```
dw_1.settransobject(sqlca)                                    //设置参数
dw_1.retrieve()                                               //读取数据
sle_1.text=string(today(),"yyyy-mm-dd hh:mm:ss")              //设置时间格式
select count(*) into :howmuch from cc_cpk;                    //统计记录条数
sle_2.text=string(howmuch)
dw_1.scrolltorow(long(trim(sle_2.text)))                      //滚动到当前记录
```

② 增加事件(cb_1.clicked())的脚本

单击此按钮后，会增加一条空的新记录，并滚动到该记录，添加新的记录数据。

```
dw_1.insertrow(0)
howmuch = howmuch +1
sle_2.text=string(howmuch)
dw_1.scrolltorow(howmuch)
```

③ 向前翻事件(cb_4.clicked())的脚本

单击此按钮后，会滚动到当前显示记录的上一条记录，并显示在数据窗口中，以供用

户察看。

```
dw_1.scrollpriorrow( )
IF long(trim(sle_2.text))>0 THEN
   sle_2.text=string(long(trim(sle_2.text))-1)
END IF
```

④ 向后翻事件(cb_1.clicked())的脚本

单击此按钮后，会滚动到显示当前记录的下一条记录，并显示在数据窗口中，以供用户察看。

```
dw_1.scrollnextrow( )
IF long(trim(sle_2.text))+1<=HOWMUCH THEN
    sle_2.text=string(long(trim(sle_2.text))+1)
END IF
```

⑤ 确认事件(cb_4.clicked())的脚本

直接用 update()函数在数据库中存储所填入的数据。无论存储成功与否，都会有提示信息。

```
if dw_1.update()=1 then
    messagebox("操作结果提示","操作成功！ ")
else
    messagebox("操作结果提示","操作失败！请检查！")
    rollback;
    return
end if
```

⑥ 数据窗口事件(dw_1.clicked(xpos,ypos,row,dwo))的脚本

该数据窗口事件可以实现数据窗口自动检索相关类型信息的功能，对于一些需要由系统直接检索出来的数据无须用户输入。

```
integer ljkdtjs
if dwo.name="ljk_mc" then
   ljkbh=getitemstring(row, "ljk_bh")
   select ljk_mc,ljk_gg,ljk_xz,ljk_position,ljk_dtjs,ljk_dw
   into :ljkmc,:ljkgg,:ljkxz,:position,:ljkdtjs,:ljkdw
   from cc_ljklx where ljk_bh=:ljkbh;
   //显示相关内容
   dw_1.setitem(row, "ljk_mc",ljkmc)
   dw_1.setitem(row, "ljk_gg",ljkgg)
   dw_1.setitem(row, "ljk_dw",ljkdw)
   dw_1.setitem(row, "ljk_cfwz",position)
   dw_1.setitem(row, "ljk_dtjs",ljkdtjs)
   dw_1.setitem(row, "ljk_xz",ljkxz)
end if
```

⑦ 退出事件(cb_6.clicked())的脚本

```
close(parent)
```

完成了上述工作，就得到了一个增加零件类型的表单窗口，运行结果如图 9-50 所示。

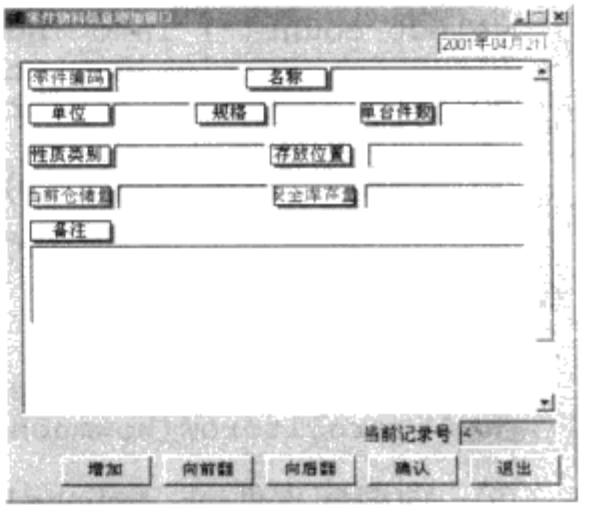


图 9-50 增加零件类型的表单效果

在零件类型信息增加窗口中，单击“增加”按钮，则数据区域中显示空白页，录入新的数据。单击“向前翻”、“向后翻”按钮，可以向前或向后翻页查看

数据。此窗口在录入数据时，可以根据零件编码自动从相关的零件库类型表中提取数据并显示出来。

9.4.2　按学号排序

本例将实现使列表中的数据按照要求排序的功能。运行后，在弹出“学生信息管理”窗口中的数据窗口列出了学生的基本信息，数据窗口的下面有 1 个“按学号排序”的复选框，其后有两个单选按钮，分别为“升序”和“降序”。先勾“按学号排序”复选框，通过选择“升序”或者“降序”按钮，学生的信息就可以在数据窗口中按照要求进行排序了。单击窗口右侧的“退出”按钮即可退出程序。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 设置为“学生信息管理”。

(4) 建立 Grid 风格数据窗口对象。连接数据库 Student，单击菜单栏上的 File|New 命令，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 Quick Select 数据源，然后选择 student 表，并选择其中所有字段，建立数据窗口对象，并保存为 d_student。

(5) 建立窗口控件。向窗口中添加 1 个 DataWindow 控件、1 个 CommandButton 控件、1 个 CheckBox 控件和两个 RadioButton 控件。依次为 dw_1、cb_1、cbx_1、rb_1 和 rb_2。

(6) 设置窗口控件。dw_1 控件的 DataObject 项选择 d_student，并勾选 HScrollBar 和 VScrollBar 复选框，cb_1 控件 Text 设置为“退出”，cbx_1 控件 Text 设置为“按学号排序”，rb_1 控件 Text 设置为“升序”，rb_2 控件 Text 设置为“降序”。

2. 编写事件驱动代码

(1) 在 w_main 窗口的 open 事件下输入如下脚本：

```
//使数据窗口与事务对象相连
dw_1.SetTransObject(sqlca)
//执行检索操作
dw_1.retrieve()
```

(2) 在 w_main 窗口中 cbx_1 控件的 Clicked 事件下输入如下脚本：

```
if cbx_1.checked = true then
...
end if
```

(3) 在 w_main 窗口中 rb_1 控件的 Clicked 事件下输入如下脚本：

```
if cbx_1.checked = true then
    dw_1.setsort("sno A")
    dw_1.sort()
end if
```

(4) 在 w_main 窗口中 rb_2 控件的 Clicked 事件下输入如下脚本：

```
if cbx_1.checked = true then
    dw_1.setsort("sno D")
    dw_1.sort()
end if
```

(5) 在 w_main 窗口中 cb_1 控件的 Clicked 事件下输入如下脚本：

```
Close(w_main)
```

(6) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(7) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

单击 PowerBuilder 10.0 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行，如图 9-51 所示。

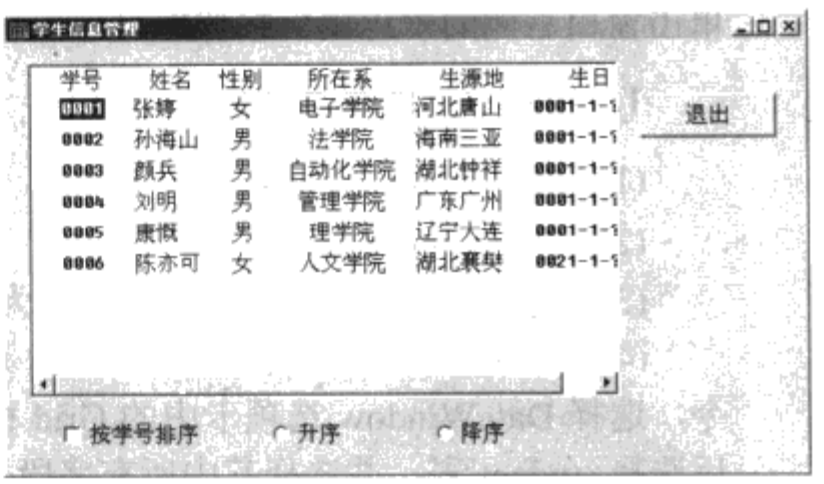


图 9-51 实例效果图

9.4.3 数据窗口排序

本例将制作一个将数据窗口中的数据进行排序的程序。运行后，在弹出“学生信息管理”窗口中，单击任何一个数据栏上的标题，该栏的数据就会自动进行排序，通过不断单击，还可以改变排序规则，即是按顺序排列还是按逆序排列。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 设置为“学生信息管理”。

(4) 建立 Grid 风格数据窗口对象。连接数据库 Student，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 Quick Select 数据源，然后选择 student 表，并选择其中 sno、sname、sex、department 和 wherefrom 字段，建立数据窗口对象并保存为 d_student。

2. 建立标准可视用户对象

(1) 从菜单栏中选择 File|New 命令，在打开的 New 对话框的 PB Object 选项卡中选择 Standard Visual 图标，单击 OK 按钮，弹出 Select Standard Visual 对话框，在其中选择 datawindow 选项，单击 OK 按钮，建立标准数据窗口可视用户对象，并保存为 uo_order。

(2) 在 uo_order 用户对象的 clicked 事件中输入如下代码：

```
string ls_AddPict, ls_CurObj, ls_Picture, ls_CurCol
integer li_PictPos
ls_CurObj = String(dwo.Name)
If Row = 0 AND This.Describe(ls_CurObj + ".Text") <> "!" AND
This.Describe(ls_CurObj + ".Band") = "header" Then // Valid header object?
```

```
        ls_CurCol = Left(ls_CurObj,Len(ls_CurObj) - 2)
        If is_OrderCol <> ls_CurCol Then // Different Column
...
        End If
    End If
```

(3) 关闭标准可视用户对象设置窗口，并在 w_main 窗口中添加标准可视用户对象 uo_order，在窗口中命名为 dw_1，并将 DataObject 设置为 d_student，钩选 HScrollBar 和 VScrollBar 属性。

3. 编写事件驱动代码

(1) 在 w_main 窗口的 Open 事件中输入如下脚本：

```
dw_1.SetTransobject(sqlca)
dw_1.retrieve()
```

(2) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm =
"Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(3) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

(4) 单击 PowerBuilder 10.0 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行。实例效果图如图 9-52 所示。

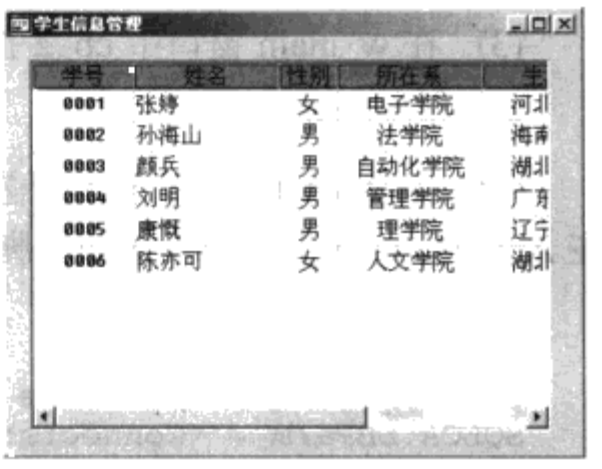


图 9-52　实例效果图

9.4.4　利用 DataStore 共享数据

本例将实现在窗口中利用 DataStore 共享数据。运行后，弹出“学生信息管理”窗口，在窗口的上部有一个数据窗口以列表的形式显示每名学生的信息，右边有两个按钮，分别为“共享数据”按钮和“退出”按钮。单击“共享数据”按钮，上面窗口的数据将共享到下面的窗口中。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 设置为“学生信息管理”。

(4) 建立 Grid 风格数据窗口对象。连接数据库 Student，单击菜单栏上的 File|New 命令，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 Quick Select 数据源，然

后选择 student 表，并选择其中所有字段，建立数据窗口对象，在出现的新数据窗口中将表格的标题用中文表示，然后将数据窗口保存为 d_student。

(5) 建立窗口控件。向窗口中添加两个 DataWindow 控件和两个 CommandButton 控件。依次为 dw_1、dw_2、cb_1 和 cb_2。

(6) 设置窗口控件。dw_1 控件的 DataObject 选项选择 d_student，并选中 HScrollBar 和 VScrollBar 复选框；cb_1 控件 Text 设置为“共享数据”；cb_2 控件 Text 设置为“退出”。

2. 编写事件驱动代码

(1) 在 w_main 窗口的 open 事件下输入如下脚本：

```
dw_1.SetTransObject(sqlca)
dw_1.retrieve()
```

(2) 在 w_main 窗口中 cb_1 控件的 Clicked 事件下输入如下脚本：

```
datastore ds_student
ds_student = create datastore
ds_student.dataobject = "d_student"
ds_student.settransobject(sqlca)
ds_student.retrieve()
dw_2.dataobject = ds_student.dataobject
ds_student.sharedata(dw_2)
```

(3) 在 w_main 窗口中 cb_2 控件的 Clicked 事件下输入如下脚本：

```
close(parent)
```

(4) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(5) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

(6) 单击 PowerBuilder 10.0 开发界面上的 按钮或按 Ctrl+R 快捷键，程序就可以运行。实例效果图如图 9-53 所示。

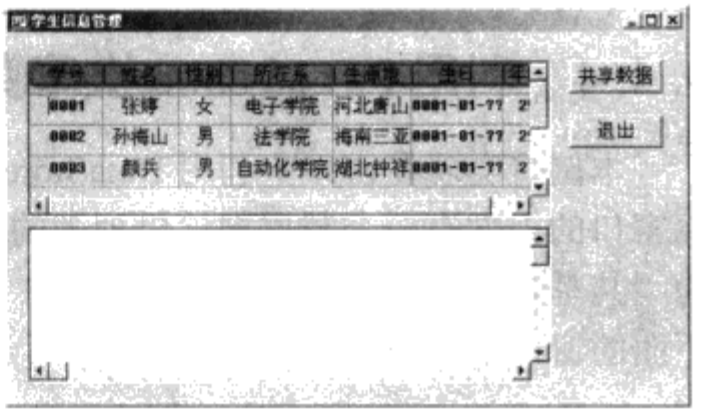


图 9-53 实例效果图

在 PowerBuilder 10.0 中，数据存储对象实际上就是去掉了可视特征的数据窗口控件。除了可视特征之外，数据存储对象与数据窗口控件的功能和用法相同。因此，它们有许多相同的属性、事件和函数。

数据存储对象的属性很简单，只有两个，其说明见表 9-10。

表 9-10　DataStore 对象属性说明

属性	说　明
DataObjectString	指定与数据存储对象相关联的数据窗口对象名或报表对象名
ObjectDWObject	用于在代码中直接操作数据窗口对象中的对象，包括设置对象的属性、得到数据窗口中的数据等

9.4.5　关联与条件查询

本例将实现两个不同数据表格的关联与条件查询。运行后，在弹出“学生信息管理”窗口中左边数据窗口列出每名学生的基本信息，右边数据窗口显示学生考试的科目及其分数。用鼠标指定左边窗口中的一名同学，相应的在右边窗口就会显示出该同学的考试成绩。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 设置为“学生信息管理”。

(4) 建立 Grid 风格数据窗口对象。连接数据库 Student，单击菜单栏上的 File|New 命令，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 Quick Select 数据源，然后选择 student 表，并选择其中所有字段，建立数据窗口对象，并保存为 d_student。

2. 建立数据表的外部关键字

从菜单栏中选择 Tool|DataBase Painter 命令，连接数据库 Student，展开 Tables 文件夹中的表 grade，在 Foreign Key 文件夹上右击并在弹出的快捷菜单中选择 New Foreign Key 命令，或者直接选中表 grade 并单击工具栏上的图标的下拉图标列表中的按钮，如图 9-54 所示。出现定义外部关键字对话框，在 General 选项卡中选中 sno 项，在 Primay Key 选项卡中的 Table 框中选择 student 项，在 Columns 列表框中选中 sno 项，如图 9-55 所示，单击按钮进行保存。然后继续用鼠标右击 grade 表，在弹出的快捷菜单中选择 New Foreign Key 命令，在定义外部关键字对话框的 General 选项卡中选中 cno 项，在 Primay Key 选项卡中的 Table 框中选择 Course 项，在 Columns 项中选中 cno 项，单击按钮进行保存。

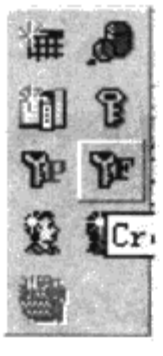
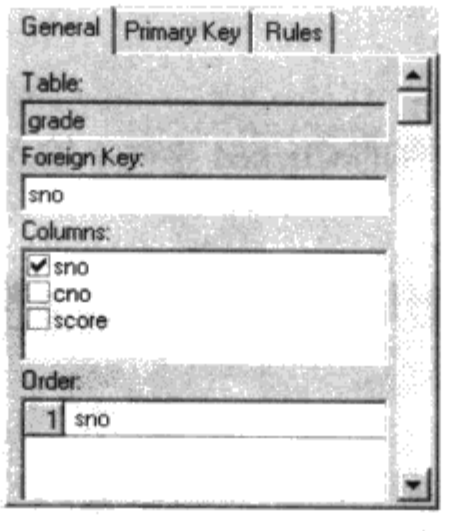


图 9-54　添加新外部关键字



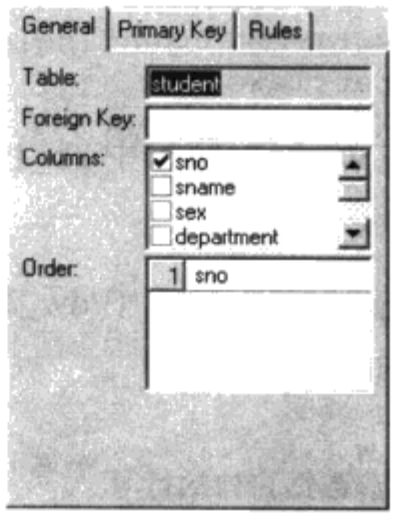


图 9-55　定义外部关键字对话框

3. 建立 Grid 风格关联数据窗口对象

(1) 建立显示字段。单击菜单栏上的 File|New 命令，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 SQL Select 数据源，单击 Next 按钮打开 Select 画板。分别单击 course 和 grade 两个表，单击 Open 按钮打开。此时 Select 画板的 Table Layout 窗口显示两个表及其关联情况，如图 9-56 所示。然后在 Table Layout 窗口中选择 cname、score 和 sno 字段(直接在字段名上单击鼠标)。

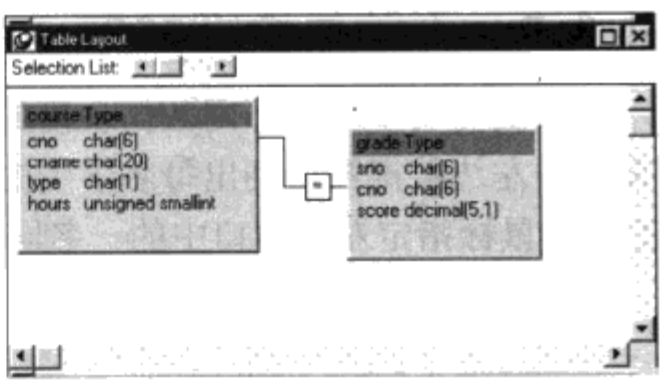


图 9-56 Table Layout 窗口

(2) 指定参数。单击菜单栏上的 Design|Retrieval Argument 命令，打开 Specify Retrieval Arguments 对话框。在 Name 编辑框中输入参数名为 sno，在 Type 下拉列表单中选择 string。单击 OK 按钮，完成参数的指定。

(3) 指定参数的使用。继续单击 Specify Retrieval Argument 对话框底部的 Where 标签，单击 Column 列下的单元格，选择"grade"."sno"(表示 grade 表的 sno 字段)，Operator 下的单元格自动为“=”号，接着用鼠标右击 Value 栏下的单元格，从弹出的快捷菜单中选择 Arguments。从打开的参数列表框中选择参数 sno 并单击 Paste 按钮。最后单击工具栏上的按钮，出现设置颜色和边框的对话框，直接单击 Next 按钮，然后继续单击 Finish 按钮，在出现的新数据窗口中将表格的标题用中文表示，然后将数据窗口保存为 d_score。

4. 设置窗口控件

(1) 建立窗口控件。向窗口中添加两个 DataWindow 控件和一个 CommandButton 控件。依次为 dw_1、dw_2 和 cb_1。

(2) 设置窗口控件。dw_1 控件的 DataObject 项选择 d_student，并选中 HScrollBar 和在 VScrollBar 复选框，dw_2 控件的 DataObject 项选择 d_score，并选中 HScrollBar 和 VScrollBar 复选框，cb_1 控件 Text 设置为“退出”。

5. 编写事件驱动代码

(1) 在 w_main 窗口的 open 事件下输入如下脚本：

```
dw_1.SetTransObject(sqlca)
dw_1.retrieve()
```

(2) 在 w_main 窗口中 cb_1 控件的 Clicked 事件下输入如下脚本：

```
Close(w_main)
```

(3) 在 w_main 窗口中 dw_1 控件的 rowfocuschanged 事件下输入如下脚本：

```
//变量说明
string  s1
int row
//取当前数据窗口的行号
row = GetRow(dw_1)
//如果点在范围以外则返回
if row < 1 then return
```

```
//设置显示方式：原行选择为假
SelectRow(dw_1, 0, False)
//设置显示方式：当前行选择为真
SelectRow(dw_1, row, True)
//读取数据窗口指定行的第一列数据(已知为 string 类型)
s1 = dw_1.GetItemString(row, 1)
//使数据窗口控件 dw_2 建立与 SQLCA 事务的连接
dw_2.settransobject(sqlca)
//根据参数 s1 完成检索
dw_2.retrieve(s1)
```

(4) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(5) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

(6) 单击 PowerBuilder 10.0 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行。实例效果图如图 9-57 所示。

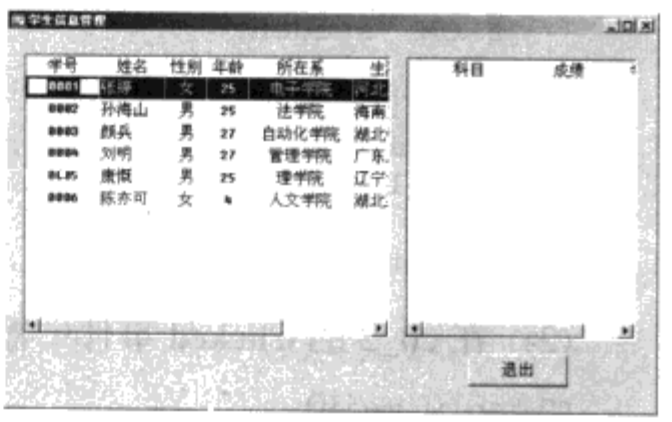


图 9-57　实例效果图

9.4.6　数据查询

本例将制作一个数据查询系统。运行后，在弹出“学生信息管理”窗口中制定的位置输入所要查询的学生的学号，单击“查询”按钮，在窗口的下端就会将该学生的基本信息显示出来，如果系统中没有所查询的学号，则会弹出警告窗口，说明“没有此学生！”。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 设置为“学生信息管理”。

(4) 建立 Grid 风格数据窗口对象。连接数据库 Student，选择菜单栏上的 File|New 命令，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 Quick Select 数据源，然后选择 student 表，并选择其中所有字段，建立数据窗口对象，并保存为 d_student。

(5) 建立窗口控件。在窗口中添加 6 个 StaticText 控件、5 个 SingleLineEdit 控件和两个 CommandButton 控件，分别命名为 st_1~st_6、sle_1~sle_5、cb_1 和 cb_2。

(6) 设置控件属性。st_1~st_6、cb_1 和 cb_2 控件的 Text 值分别为“学生信息查询系统”、“请输入所要查询学生的学号”、“姓名”、“性别”、“年龄”、“所在系”、“查询”和“退出”。选中 st_1 控件的 Border 属性和 cb_1 控件的 Default 属性。

(7) 保存窗口。

2. 编写事件驱动代码

(1) 在 cb_1 的 Clicked 事件中输入如下脚本：

```
sle_2.text=""
sle_3.text=""
sle_4.text=""
sle_5.text=""
SELECT "student"."sno",
       "student"."sname",
       "student"."sex",
       "student"."age",
       "student"."department"
  INTO :sle_1.text,
       :sle_2.text,
       :sle_3.text,
       :sle_4.text,
       :sle_5.text
  FROM "student"
 WHERE "student"."sno" = :sle_1.text;
IF sle_2.text="" THEN
   MessageBox("Not found","没有此学生!")
   sle_1.text=""
END IF
sle_1.setfocus()
```

(2) 在 cb_2 的 Clicked 事件中输入如下脚本：

```
Close(w_main)
```

(3) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm =
"Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(4) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

单击 PowerBuilder 10.0 开发界面上的 按钮或按 Ctrl+R 快捷键，程序就可以运行。实例效果图如图 9-58 所示。

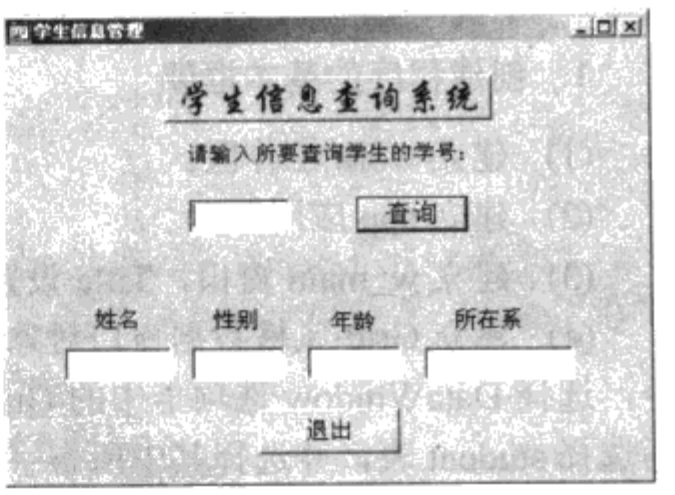


图 9-58 实例效果图

9.4.7 模糊查询

本例将制作一个在数据窗口中进行模糊查询的程序。运行后，在弹出“模糊查询”窗口中有一个文本框，在文本框中输入学生的学号的模糊查询写法，即在不清楚的学号数字

处用“%”替代，如“0163%1”和“01632%”等。按 Enter 键时窗口中就会显示模糊查询结果。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

(3) 建立 w_main 窗口，Title 设置为“模糊查询”。

(4) 建立 Grid 风格数据窗口对象。连接数据库 Student，选择 DataWindow 选项卡中的 Grid 格式数据窗口，接着选择 Quick Select 数据源，然后选择 student 表，并选择其中所有字段，建立数据窗口对象并保存为 d_student。

(5) 建立窗口控件。在窗口中添加 1 个 DataWindow 控件、1 个 StaticEdit 控件、1 个 SingleLineEdit 控件和 3 个 CommandButton 控件，分别命名为 dw_1、st_1、sle_1、cb_1、cb_2 和 cb_3。

(6) 设置控件属性。将 dw_1 控件的 DataObject 设置为 d_student，选中 HScrollBar 和 VScrollBar 复选框。将 st_1、cb_1、cb_2 和 cb_3 控件的 Text 值依次设置为“请输入学号”、“模糊查询”、“重置”和“退出”，将 st_1 控件的 Text 值设置为空。

(7) 保存窗口。

2. 编写事件驱动代码

(1) 在 w_main 窗口中定义实例变量，输入如下代码：

```
string oldsql
```

(2) 在 cb_1 的 clicked 事件中输入如下脚本：

```
string wheresql, newsql
wheresql = "where student.sno like'"+ sle_1.text+"'"
newsql = oldsql + wheresql
dw_1.setsqlselect(newsql)
dw_1.retrieve()
```

(3) 在 cb_2 的 Clicked 事件中输入如下脚本：

```
dw_1.setsqlselect(oldsql)
dw_1.retrieve()
```

(4) 在 cb_3 的 Clicked 事件中输入如下脚本：

```
close(w_main)
```

(5) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，在 App 的 Open 事件中输入如下脚本：

```
// Profile Student
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "Connectstring='DSN=student'"
Connect;
Open(w_main)
```

(6) 在 PowerBuilder 10.0 开发界面左边的 System Tree 窗口中，双击 App 应用对象，

在 App 的 Close 事件中输入如下脚本：

```
Disconnect;
```

单击 PowerBuilder 10.0 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行。实例效果图如图 9-59 所示。

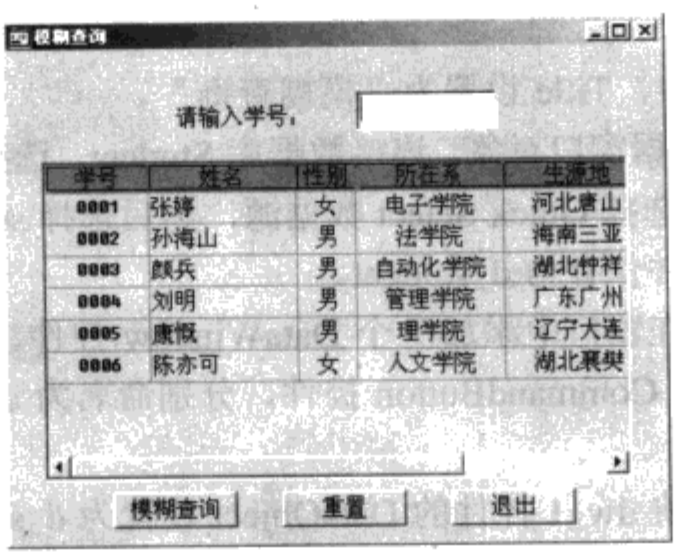


图 9-59 实例效果图

9.5 习 题

1. 填空题

(1) 数据窗口对象有 11 种不同的显示风格：_____、_____、_____、_____、_____、_____、_____、_____、_____、_____、_____。

(2) 统计图风格的数据窗口提供了多种统计图，包括：_____、_____、_____、_____、_____、_____、_____等。

(3) PowerBuilder 提供了 5 种不同的数据源，包括：_____、_____、_____、_____、_____。

(4) 数据窗口直接参照报表而将显示区划分成 6 个部分，包括_____、_____、_____、_____、_____、_____。

2. 问答题

(1) 试根据 9.1.1 节的设计步骤自己设计一个数据窗口。

(2) 如何定义 Query 数据源，请将它与其他数据源的定义进行比较。

(3) 如何为数据窗口中的数据排序？

(4) 熟悉数据窗口控件的常用函数，并练习如何使用这些函数。

第 10 章　数据管道技术

PowerBuilder 提供了数据管道(Data Pipeline)功能。本章介绍如何在应用程序中使用管道对象(Pipeline Object)把数据从一个或多个源表中输送到一个新的或现有的目标表中。

10.1　数据管道概述

PowerBuilder 提供的数据管道功能可以在不同的数据库表之间移动数据。这种功能能将一个或多个源表中的数据复制到一个新的或已经存在的目标表中。这种数据移动可以在同一个数据库中进行，也可以在不同数据库之间进行。

PowerBuilder 提供了如下两种不同方法来使用数据管道：

- 第 1 种方法是作为实用工具提供给应用程序开发者的。当在 PowerBuilder 环境中进行开发时，可能常常因为某种原因移动数据，如从一个大型的产品表中抽取数据建立一个小型的测试表。在这种情况下，可以在数据管道画板中交互式地在不同的表之间移动数据。
- 第 2 种方法是在应用程序中实现数据移动。如果开发一个需在不同表之间移动数据的应用程序，可以在数据管道画板中设计适当的数据管道对象并保存，然后再编写程序允许用户通过执行应用程序实现数据移动操作。这种技术在很多情况下是十分有用的。例如，当希望从数据库服务器上向一个远程用户下载本地表的副本时，或者当需要把各自事务表集中起来提交给一个主事务表时，使用数据管道是十分有效的。

在应用程序中使用数据管道有以下 5 个基本步骤：

(1) 创建所需对象。
(2) 执行初始化操作。
(3) 启动数据管道。
(4) 处理行错误。
(5) 执行结束操作。

本章将详细介绍每一个步骤的具体实现方法。

注意： 因为建立管道操作的应用程序中需要经常动态地引用(通过字符串变量)相应的管道对象，所以，必须把每个管道对象打包在一个或多个动态数据库中，而不能把它们包含在可执行文件中。

10.2　创 建 对 象

为了在一个应用程序中实现数据管道，需要建立以下几个不同的对象：

- 一个管道对象。

- 一个支撑用户对象。
- 一个窗口。

10.2.1 创建管道对象

实现管道功能必须创建一个管道对象(Pipeline Object)，在此管道对象中定义应用程序需执行的数据定义及数据访问。可在 PowerBuilder 提供的数据管道画板(DataPipeline Painter)中，建立一个管道对象并定义它的特征。

在数据管道画板中能够为数据管道定义的特征有：

- 需访问的源表以及需检索的数据(也可以把数据库存储过程作为数据源)。
- 数据传输的目标表。
- 管道操作(创建、替换、刷新、增加或修改)。
- 在管道操作过程中的提交频率(当传输 n 个行或所有行传输完毕后提交，如果想在程序中定义自己的提交逻辑也可以不定义此特征)。
- 允许的错误数量，如果超出此数量则终止管道操作。
- 是否向目标数据库传输源数据库中的扩展属性，该扩展属性保存在 PowerBuilder 信息库中。

创建数据管道的步骤如下：

(1) 选择 File 菜单中的 New 命令，出现 New 对话框，单击对话框中的 Database 选项卡，选择 Data Pipeline 图标，如图 10-1 所示。

(2) 单击 OK 按钮，出现如图 10-2 所示的对话框，在 New Data Pipeline 对话框中选择数据源和数据库连接，这里选择默认设置。

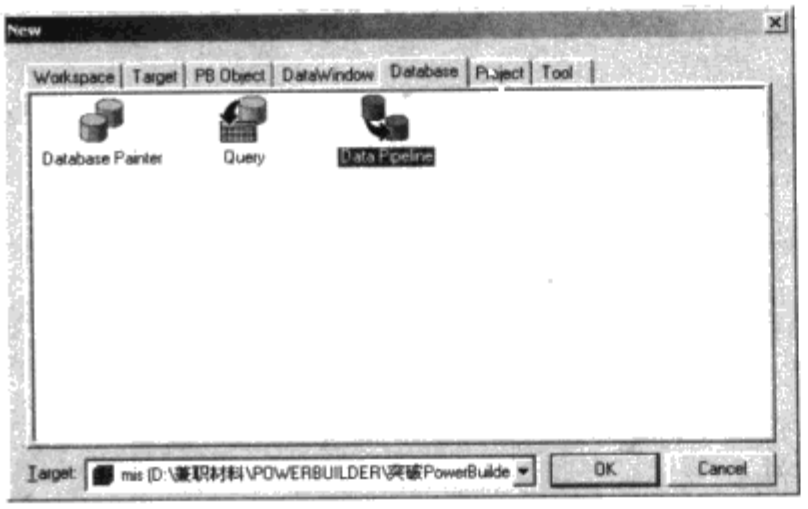


图 10-1 New 对话框

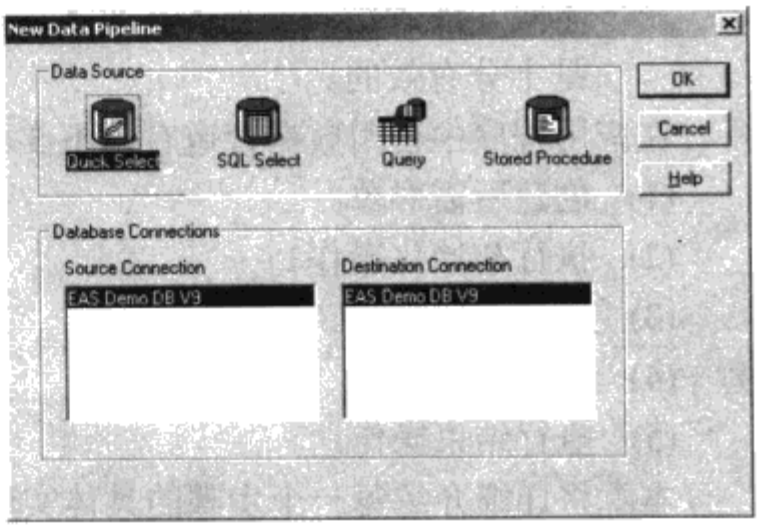


图 10-2 New Data Pipeline 对话框

(3) 单击 OK 按钮，出现如图 10-3 所示的 Quick Select 对话框。

(4) 在 Quick Select 对话框中选择表、列，并建立相应的规则后就完成了一个数据管道的建立。本例中建立的数据管道如图 10-4 所示，在该界面中可以修改数据管道属性。

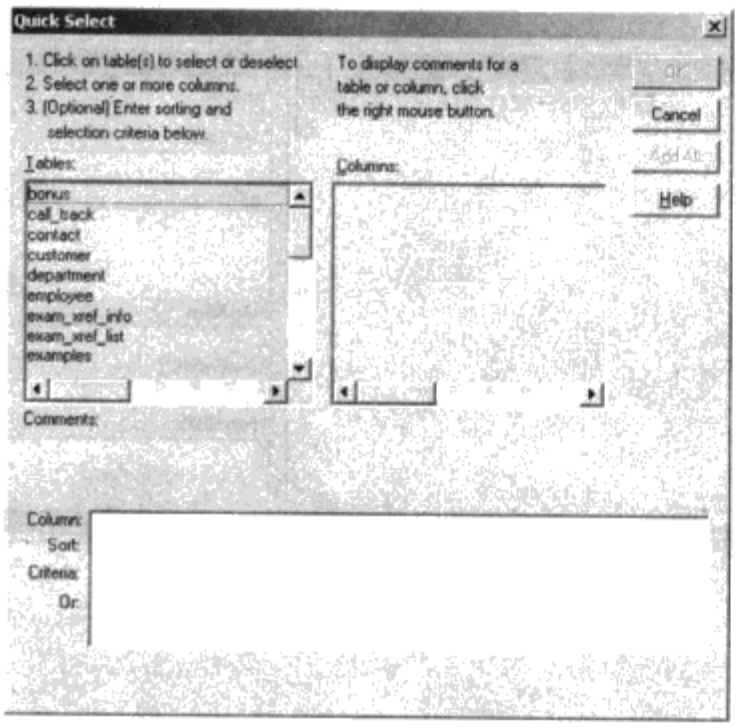


图 10-3　Quick Select 对话框

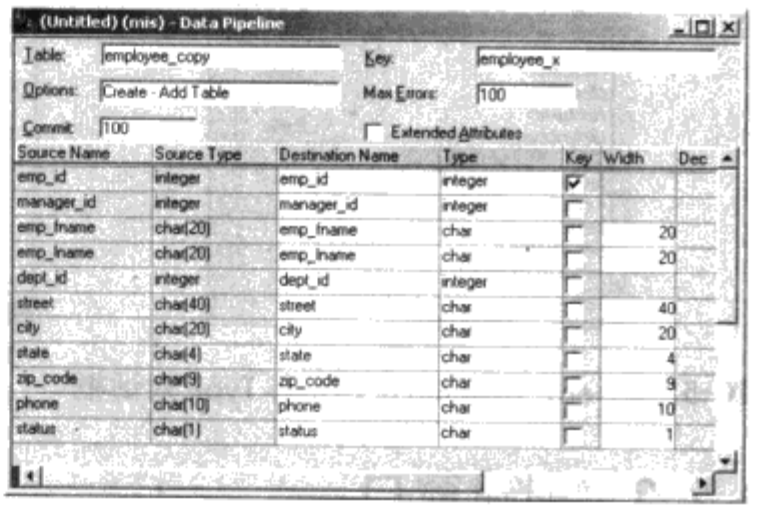


图 10-4　新建数据管道

10.2.2　创建支撑用户对象

在上节中已经定义了管道对象的数据和访问方式。但是，一个管道对象并没有提供对属性、事件及函数的支持，而在应用程序中需要利用这些属性、事件及函数来处理管道操作并进行一些相应控制。

为了提供对属性、事件以及函数的支持，需要创建一个继承 PowerBuilder 管道系统对象的用户对象，通过这个用户对象来管理管道操作，即支撑管道对象所定义的管道操作。管道系统对象包括了各种不同的属性、事件及函数，在程序运行时可以用来管理管道对象。如表 10-1 所示列出了这些属性、事件及函数。

表 10-1　管道对象的属性、事件及函数

属性	DataObject，RowsRead，RowsWritten，RowsInError，Syntax
事件	PipeStart，PipeMeter，PipeEnd
函数	Start，Repair，Cancel

为一个管道对象创建支撑用户对象的基本步骤如下：

(1)　在 New 对话框的 PB Object 选项卡中选择 Standard Class 图标，弹出 Select Standard Class Type 对话框，如图 10-5 所示。从中选取一个内置系统对象，所创建的用户对象将继承这个系统对象。

(2)　选中 Pipeline 并单击 OK 按钮，屏幕上将出现用户对象画板工作区，如图 10-6 所示。

(3)　根据需要对用户对象进行修改，如编写用户对象的相应事件脚本、定义函数及变量等。

(4)　保存用户对象。

在设计用户对象时，可以把它设计成为一个通用的支持管道对象的用户对象，这样就不必为每个管道对象都单独创建一个支撑用户的对象了。

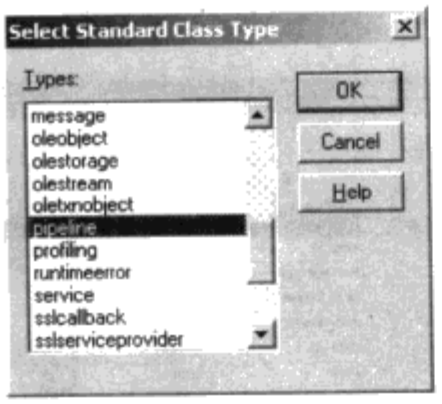


图 10-5 Select Standard Class Type 对话框

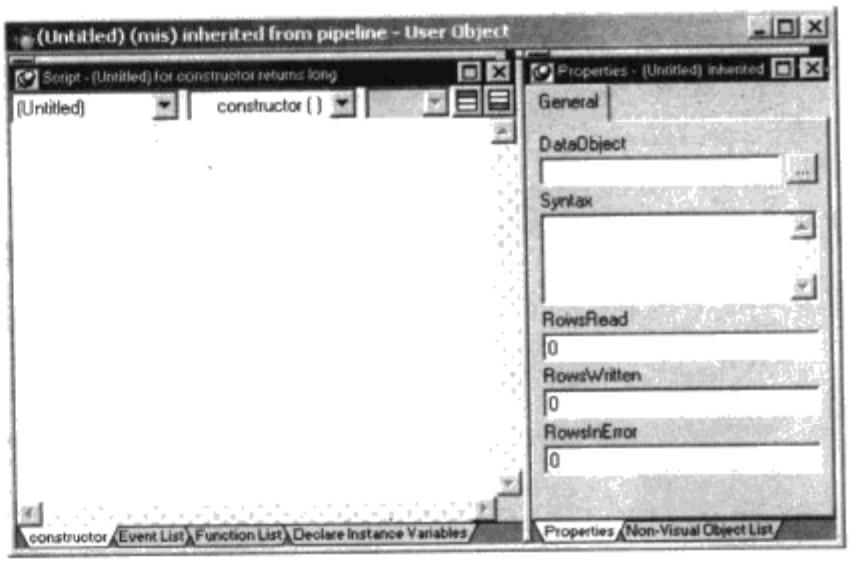


图 10-6 用户对象画板工作区

10.2.3 创建窗口

在应用程序中建立管道操作所需的另一个对象就是窗口。窗口为管道操作提供了一个用户界面，通过这个界面可以与管道对象进行交互，对管道操作进行监控。该窗口具有的基本操作是：

- 启动管道操作。
- 当发生错误时，显示并修复错误。
- 在必要时终止管道操作。

在创建窗口时，必须在窗口中包括一个数据窗口控件，这个数据窗口控件是为管道对象显示错误行(因为某些原因而不能传输到目标表中的行)而提供的。不用给数据窗口控件关联一个数据窗口对象，管道会在程序运行中提供自己的数据窗口对象。

除了为窗口增加一个数据窗口控件外，还可以在窗口中添加如下控件：

- 命令按钮(CommandButton)控件或图片按钮(PictureButton)控件：提供管道启动、修复或终止操作等初始化工作。
- 静态文本(Static Text)控件：显示管道状态信息。
- 其他数据窗口控件：显示源表或目标表的内容。

10.3 数据管道操作

10.3.1 初始化操作

要让管道在应用程序中运转之前首先还必须完成一些初始化工作，为应用程序处理管道运行做准备。

应用程序管道操作的初始化步骤如下：

(1) 连接管道操作的源数据库和目标数据库，为完成此步，需为应用程序的适当事件编写脚本。在脚本中准备两个事务对象，让一个事务对象连接源数据库，让另一个事务对象连接目标数据库。

注意： 不管源数据库与目标数据库是否是同一个数据库，都需要让两个事务对象分

别进行连接。

(2) 创建支撑用户对象的一个实例，以便在应用程序中能够使用该支撑用户对象的属性、事件及函数。

(3) 定义需使用的特定管道对象，即在相应事件脚本中把某一管道对象名字的字符串赋给支撑用户对象实例的 DataObject 属性值。

下面来看一个连接源数据库的实例。一个公司的销售数据库 ABNCSALE.DB 是源数据库也是目标数据库。在一个名为 uevent_pipe_setup 的用户事件中写入代码以完成与 ABNCSALE.DB 数据库的连接操作及其他初始化工作。uevent_pipe_setup 用户事件将在 w_sales_extract 窗口中触发执行。

(1) 连接源数据库及目标数据库

建立与源数据库的连接，脚本如下：

```
//创建一个新的事务对象实例，并保存在事先声明的
//itrans_source 事务变量中
itrans_source = CREATE transaction
//下一步为 itrans_source 事务对象赋值
//然后与源数据库进行连接
   CONNECT USING itrans_source;
```

建立与目标数据库的连接，脚本如下：

```
//创建一个新的事务对象实例，并保存在事先声明的
//itrans_destination 事务变量中
 itrans_destination = CREATE transaction
//下一步为 itrans_destination 事务对象赋值
//然后与源数据库进行连接
CONNECT USING itrans_destination;
```

(2) 创建一个支撑用户对象实例

假设已经创建了一个名为 u_sales_pipe_logistics 的支撑用户对象，要在应用程序中使用这个用户对象必须事先为它声明一个变量：

```
//iuo_pipe_logistics 是 w_sales_extract 窗口的一个实例变量
u sales pipe_logistics  iuo_pipe_logistics
```

然后再在 uevent_pipe_setup 用户事件中创建一个 u_sales_pipe_logistics 的实例，并且将它保存在 iuo_pipe_logistics 变量中：

```
iuo_pipe_logistics=CREATE u_sales_pipe_logistics
```

(3) 定义所使用的管道对象

根据用户所需的管道操作，在此例中使用了两个不同的管道对象。pipe_sales_extractl 建立一个名为 Quartefiy_extract 的表(该表事先不存在)，而 pipe_sales_extract2 在 Quartefiy_extract 表中插入行。

10.3.2　启动和终止数据管道

1.　启动管道

做好初始化工作后，就可以启动管道操作了。为了实现这一功能，首先需在相应的事

件脚本中写入启动程序代码，利用支撑用户对象的 Start 函数来启动某一指定的管道操作。在 Start 函数中需指明：

- 源数据库的事务对象。
- 目标数据库的事务对象。
- 显示错误信息的数据窗口控件。
- 在管道对象中定义的检索参数值，如果省略该参数，PowerBuilder 将会在运行过程中自动让用户输入该参数值。

当启动管道操作后，还需测试 Start 函数的返回值，以判断 Start 函数是否成功执行。

2. 显示管道操作的统计数据

测试 Start 函数的返回值并不是监控管道操作状态的惟一方法，还有一种技术可以获得支撑用户对象对已处理行的统计数据，这些统计数据包括：

- 管道从源数据表中读取的行数。
- 管道向目标表写入的行数或向错误数据窗口控件中写入的行数。
- 管道向错误数据窗口控件中写入的行数(不包括向目标表中写入的行数)。

在程序运行过程中这些统计数据可以从支撑用户对象中检索出来，在窗口中向用户动态显示管道操作的过程。显示管道操作统计数据的实现步骤如下：

(1) 在用户对象画板中双击打开支撑用户对象，屏幕上出现用户对象画板工作区。

(2) 声明 3 个类型为静态文本的实例变量：

```
statictext ist_status_read, ist_status_written, ist_status_error
```

这 3 个实例变量将在窗口中动态显示相应的统计数据。

(3) 编写支撑用户对象的 PipeMeter 事件脚本。在此事件脚本中把 RowsRead、RowsWritten 和 RowslnError 从管道系统变量继承来的属性分别赋给 3 个静态实例变量的 Text 属性：

```
ist_status_read.text=String(RowsRead)
ist_status_written.text=String(RowsWritten)
iSt_status_error.text=String(RowslnError)
```

(4) 保存所做的修改，然后关闭用户对象画板。

(5) 在窗口画板中打开处理管道操作的窗口(如 w_sales_extract)，让用户在出现的窗口画板工作区中编辑所打开的窗口。

(6) 在窗口中加入 3 个静态文本控件，一个用于显示 RowsRead 的值，一个用于显示 RowsWritten 的值，还有一个用于显示 RowslnError 的值。在 w_sales_extract 窗口加入了 3 个静态文本控件 st_status_read、st_status_written 和 st_status_error。

(7) 编辑窗口的 Open 事件(或其他在窗口打开后所需执行的事件脚本)。在 Open 事件的事件脚本中把窗口中的 3 个静态文本控件分别赋给前面定义的 3 个静态文本变量，这样用户对象就可以直接操纵这些静态文本控件了。

```
Iuo_pipe_logistics.ist_status_read=st_status_read
iuo_pipe_logistics.ist_status_written=st status_written
iuo_pipe_logistics.ist_status_error=st_status_error
```

(8) 保存窗口，然后关闭窗口画板。

当启动 w_sales_extract 窗口中的管道后，用户对象的 PipeMeter 事件被触发，在 3 个静态文本控件中将显示出管道操作所处理的行统计数据。

3. 终止管道运行

在很多情况下，都需要向用户或应用程序本身提供终止管道运行的功能(当管道操作仍在进行之中时)。例如，当用户误启动了管道操作，或管道操作执行的时间比预期的要长，就需要应用程序提供一个终止该管道运行的功能。

在程序中实现终止管道运行，首先需要在应用程序相应的事件脚本中执行支撑用户对象的 Cancel 函数。

注意： 一旦管道操作开始，用户或应用程序本身可以在任何时间执行该函数。当执行终止(Cancel)操作时，管道将停止传输任何行到目标表中。而在此之前传输的行是否提交到目标表取决于在数据管道画板中建立管道对象时定义的 Commit 属性。执行 Cancel 函数后，与 Start 函数一样需要对其返回值进行测试。

4. 向数据库提交更新操作

当运行一个管道对象时，将根据在数据管道画板中为该管道对象定义的 Commit 属性向目标数据库提交表的更新操作。不需要在应用程序中加入 COMMIT 语句来完成提交操作。

注意： 如果在数据管道画板中为该管道对象定义的 Commit 属性值为 None 时，则需要自己在应用程序中处理数据的提交操作。

例如，前面例中的 pipe_sales_extractl 和 pipe_sales_extract2 管道对象的 Commit 属性都定义为 All，当所有需要从源表中向目标表中传输的行都传输完毕后，才向目标表中提交所有的更新操作。这样，Start 函数(或 Repair 函数)将在所有行都传输完毕后才执行一个 COMMIT 语句。

在终止一个正在运行的管道操作时，也是由在数据管道画板中为该管道对象定义的 Commit 属性来决定 PowerBuilder 的处理方式。

当 Commit 属性值为 All 时，PowerBuilder 将撤消由 Start 函数或 Repair 函数向目标表传输的所有行，而当 Commit 属性值指定了一个行数 n 时(如 1、10 或 100)，即管道操作传输 n 行就执行一次 COMMIT 语句，PowerBuilder 将提交管道操作终止时已经传输的所有行。

当管道操作因为出错而终止时，PowerBuilder 也是按上述方法处理管道操作的提交工作。

10.3.3 处理行错误

1. 使用管道错误数据窗口

管道在运行过程中，也许会因为某种原因不能将某一行写入目标表中。例如，若在目标表中写入行的码值与表中某一行码值相同的话，该操作就不能完成。为了处理这类错误，

管道把这些错误行放置于 Start 函数调用参数中指定的一个数据窗口控件中。当 Start 函数为管道指定一个数据窗口控件时，管道自动把自己的特殊的数据窗口对象(管道错误数据窗口，pipeline-errorDataWindow)与该数据窗口控件连接起来。

考虑一下前面所提到的 w_sales_extract 窗口中的管道执行情况，Start 函数把所有的错误行都放置于 dw_pipe_errors 数据窗口控件中。在此数据窗口控件中，还为每一行提供了一个错误信息列。

如果管道的目标事务对象是指向一个 ODBC 的话，可以通过设置 DBParm 的 MsgTerse 参数让数据窗口中显示的错误信息简短些。尤其是当在 DBParm 中包括下面参数时，将不会显示 SQLState 错误码：

```
MsgTerse = 'Yes'
```

一旦数据窗口控件中放入了错误行，有两项可以选择：

- 修复某些行
- 删除某些行

2. 修复错误行

在大多数情况下，可以对错误进行修复，让修复后的行能够传输给目标表。修复工作是通过修改某一错误行的一列或多列的值，使目标表能够接受这一行来完成的。有两种方法可以完成错误行的修复工作：

- 在数据窗口控件中直接编辑错误行，这种方法对于应用程序开发者来说是比较容易的，因为免去了许多编程工作。
- 通过执行应用程序代码，代替编辑错误数据窗口控件中的错误行。

当修复工作完成后，下一步要实现把数据窗口中修复的行提交给目标表的工作：

(1) 在应用程序的相关事件中调用 Repair 函数，在函数参数中指定目标数据库的事务对象。

(2) 测试 Repair 函数的返回值。

当 Repair 函数执行后，可能在错误数据窗口控件中仍然有错误行。这些错误行的存在可能为以下情况：

- 已被用户或应用程序修改过，但仍然有错的行。
- 没有被用户或应用程序修改过的行。
- 因为执行 Cancel 函数还没有向目标表中写入或因为 Cancel 操作而从源表中恢复的行。

在这些情况下，可以再尝试对这些行进行修复，然后用 Repair 函数写入目标表中。另外，也可以放弃对这些错误行进行修复。

3. 放弃错误行

在有些情况下，可能需要完全抛弃错误数据窗口中一个或多个错误行。这对于处理那些不值得修复的错误行是十分有用的。可以用两种技术来放弃错误行：

- 使用 Reset 函数放弃数据窗口控件中的所有错误行。
- 使用 RowsDiscard 函数放弃数据窗口控件中某一个或多个指定行。

例如：在 w_sales_extract 窗口中的 Forgo Fixes 按钮(cb_forgofixes)提供了放弃 w_pipe_errors 数据窗口控件中所有错误行的功能，cb_forgofixes 命令按钮的 Clicked 事件脚本如下：

```
dw_pipe_errors.Reset()
```

同时，在 w_sales_extract 窗口某些事件脚本中还对 cb_forgofixes 命令按钮的状态进行了控制。当 dw_p 中 e_errors 数据窗口控件有错误行时，cb_forgofixes 命令按钮的状态为有效状态，否则为无效状态。

10.3.4　结束操作

当应用程序完成了所有的管道操作后，要注意完成一些相应的收尾工作。这些收尾工作主要是释放在管道操作开始时获得的一些资源。管道的结束操作包括以下几项：

- 清除所创建的支撑用户对象实例
- 关闭与源数据库和目标数据库之间的连接
- 清除源事务对象实例和目标事务对象实例

例如：在 w_sales_extract 窗口的 Close 事件中将完成上述的管道结束工作：

```
//清除所创建的支撑用户对象实例
DESTROY iuo_pipe_logistics
//关闭与源数据库的连接
DISCONNECT USING itrans_source;
IF itrans_source.SQLCode = -1 THEN
Beep(1)
MessageBox("数据库连接出错","Problem when disconnection from+the
source(Sales) database"+&
String(itrans_source.SQLDBCode)+ itrans_source.SQLErrText,Exclamationl)
END IF
//清除源事务对象实例
DESTROY itrans_source
//关闭与目标数据库的连接
DISCONNECT USING itrans_destination;
//判断 DISCONNECT 语句的成败
IF  itrans_destination.SQLCode = -1 THEN
Beep(1)
MessageBox("数据库连接出错","Problem when disconnection from +the
source(Sales) database" +&
String(itrans_destination.SQLDBCode) +
itrans_destination.SQLErrText,Exclamation!)
END IF
//清除目标事务对象实例
DESTROY itrans_destination
```

10.3.5　数据库之间的数据传递

在应用程序的开发和使用过程中，在不同数据库之间进行数据传递是一项不可忽视的功能需求。实现该需求的方法有多种，其中 PowerBuilder 10.0 提供的数据管道对象(Pipeline)是一种比较简捷地完成数据迁移的有效工具。

本节的任务是在 PowerBuilder 10.0 中创建数据管道并实现两个数据库之间的数据传递(由 DB_mis 到 DB_mis_copy)。首先要在数据管道画板中定义数据管道，然后，实现由数据

库 DB_mis 到数据库 DB_mis_copy 之间的数据传递。

注意： PowerBuilder 10.0 的数据管道有两种使用方式，一是在数据管道画板中定义数据管道后直接运行数据管道。这种方式方便了开发人员开发和测试应用程序，例如，可以把服务器中的数据下载到本地机，然后不依赖于服务器的进行前期开发工作。二是在应用程序中通过编写代码使用数据管道，这种方式提供了灵活运用数据管道的手段。

1. 创建数据管道对象

创建数据管道对象，具体步骤如下：

(1) 在 PowerBuilder 10.0 工具栏的图标中选择 New，或者在 File 菜单下选择 New 命令项，在弹出窗口中选择 Database 选项卡。

(2) 选取该选项卡中的 Data Pipeline 项(数据管道)，如图 10-7 所示，弹出 New Data Pipeline 对话框，在此窗口中选择数据管道的源数据库、目标数据库和数据源，如图 10-8 所示。其中，Data Source 栏用于选择数据源，这些数据源的作用与创建数据窗口时数据源的作用完全相同，它们生成指定的 SQL Select 语句，Database Connections 栏则用于指定提取数据的源表所在数据库的数据库配置和目标表所在数据库的数据库配置。指定源表与目的表所在数据库的数据库配置均为数据库 db_mis。

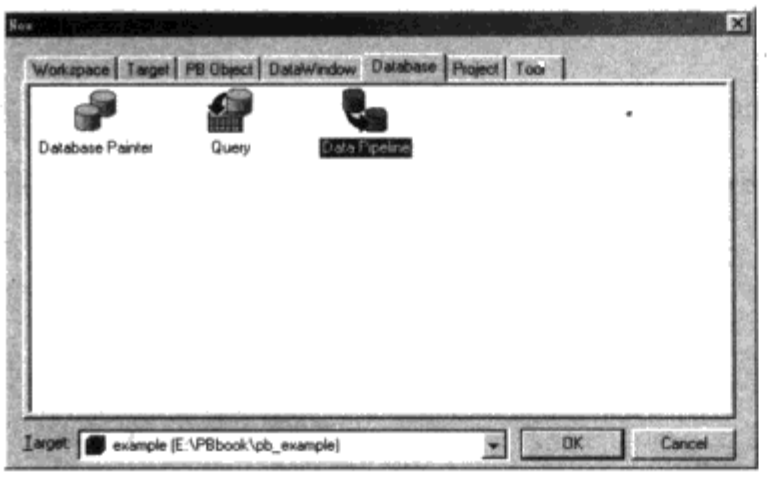


图 10-7 New 对话框

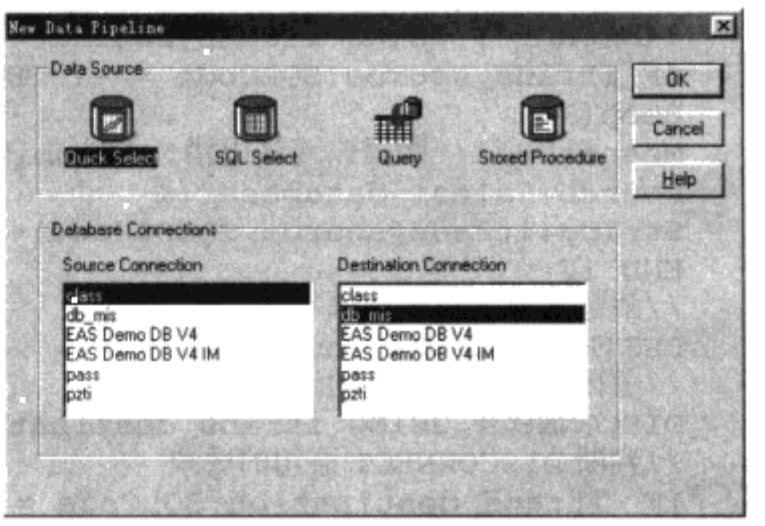


图 10-8 New Data Pipeline 对话框

(3) 选择好源数据库、目标数据库后，单击 OK 按钮，将根据所选数据源的不同，出现不同的窗口，按照定义数据窗口数据源的方法定义数据源。本例中把原表设为数据库 db_mis 中的表 cc_ljk。

(4) 定义了数据源之后，进入数据管道画板工作区，如图 10-9 所示。

2. 设置数据管道的属性

可以在数据管道画板中对目标表的属性、数据管道操作方式的选择等进行修改。

数据管道的目标表部分用于定义数据管道要把数据传输到什么地方去。在这里可以定义目标表的列名、数据类型、主键等信息，包括如下内容。

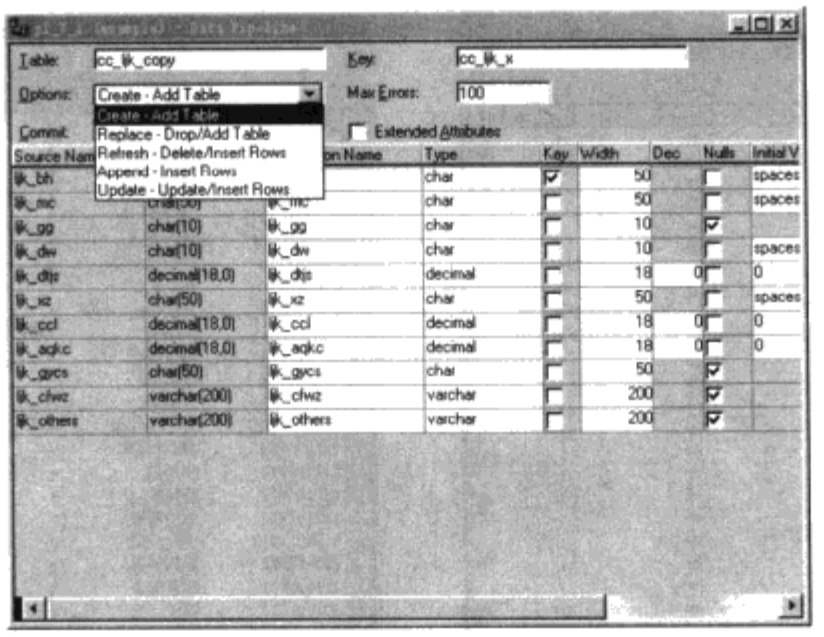


图 10-9　数据管道画板

- Destination Name 栏：指定目标表列的列名。
- Type 栏：选择列的数据类型。
- Key 栏：定义主键。
- Width 栏：指定列的宽度(Data 类型无须定义此项)。
- Dec 栏：指定数据精度(Dec 类型)。
- Nulls 栏：指定相应列是否允许空值(NULL)。
- Initial Value 栏：指定相应列的初始值。
- Default 栏：指定相应列的默认值。

在数据管道画板工作区上制定数据管道选项，这些选项决定了将以何种方式运行数据管道，选项如下：

- Table 编辑框：修改目标表的名称。
- Key 编辑框：修改目标表主键的名称。
- Options 下拉式列表：选择管道操作方式。
- Max Error 编辑框：定义允许出现的最多错误个数。
- Commit 编辑框：定义多少条记录作为一个事务提交。
- Extended Attributes 复选框：选中后，在复制数据的同时也复制指标的扩展属性。

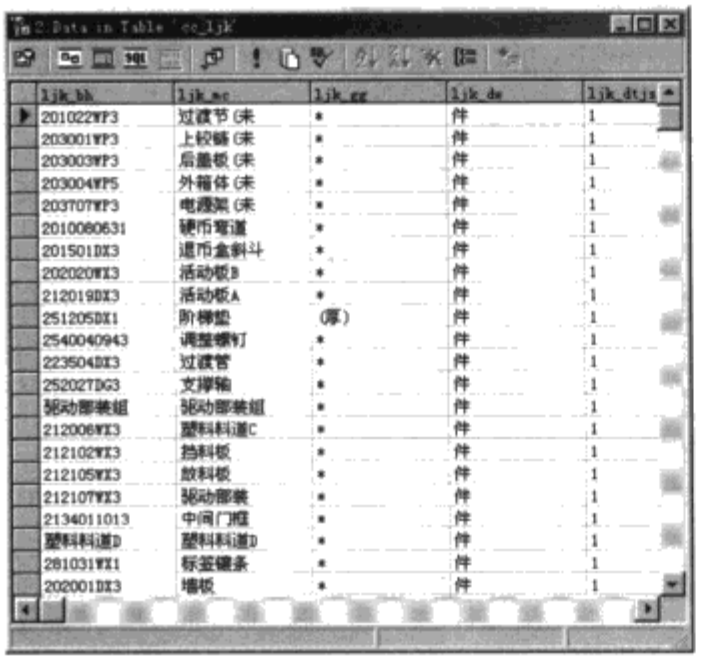
这里把目标表的名称设为 cc_ljk_copy。

3. 运行数据管道

运行数据管道的方法很简单，单击数据管道画板工具栏上的 Execute 图标，或者从 Design 菜单中选择 Execute 项。运行结果是在目标数据库中得到与源数据库中事先选中的表中的相同数据。至此，完成了在 PowerBuilder 10.0 中创建数据管道的所有步骤。

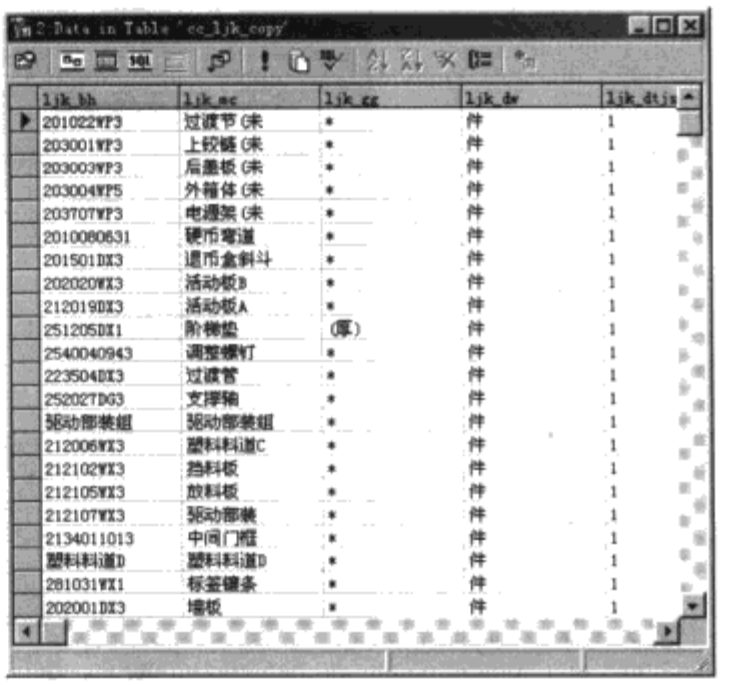
完成了上述工作，就得到了一个数据管道的数据传递结果。源数据表 cc_ljk 如图 10-10 所示，目的数据表 cc_ljk_copy 如图 10-11 所示。

可见这两个表的结构和数据是完全一样的，系统已经把一个表中的某些数据注入到另一个不同的数据库中的一个表中了。



ljk_bh	ljk_mc	ljk_gg	ljk_dw	ljk_dtjs
201022WP3	过渡节(未	*	件	1
203001WP3	上铰链(未	*	件	1
203003WP3	后盖板(未	*	件	1
203004WP5	外箱体(未	*	件	1
203707WP3	电源架(未	*	件	1
2010080631	硬币弯道	*	件	1
201501DX3	退币盒斜斗	*	件	1
202020WX3	活动板B	*	件	1
212019DX3	活动板A	*	件	1
251205DX1	阶梯垫	(厚)	件	1
2540040943	调整螺钉	*	件	1
223504DX3	过渡管	*	件	1
252027DG3	支撑轴	*	件	1
驱动部装组	驱动部装组	*	件	1
212006WX3	塑料料道C	*	件	1
212102WX3	挡料板	*	件	1
212105WX3	放料板	*	件	1
212107WX3	驱动部装	*	件	1
2134011013	中间门框	*	件	1
塑料料道D	塑料料道D	*	件	1
281031WX1	标签镶条	*	件	1
202001DX3	墙板	*	件	1

图 10-10　源数据表 cc_ljk



ljk_bh	ljk_mc	ljk_gg	ljk_dw	ljk_dtjs
201022WP3	过渡节(未	*	件	1
203001WP3	上铰链(未	*	件	1
203003WP3	后盖板(未	*	件	1
203004WP5	外箱体(未	*	件	1
203707WP3	电源架(未	*	件	1
2010080631	硬币弯道	*	件	1
201501DX3	退币盒斜斗	*	件	1
202020WX3	活动板B	*	件	1
212019DX3	活动板A	*	件	1
251205DX1	阶梯垫	(厚)	件	1
2540040943	调整螺钉	*	件	1
223504DX3	过渡管	*	件	1
252027DG3	支撑轴	*	件	1
驱动部装组	驱动部装组	*	件	1
212006WX3	塑料料道C	*	件	1
212102WX3	挡料板	*	件	1
212105WX3	放料板	*	件	1
212107WX3	驱动部装	*	件	1
2134011013	中间门框	*	件	1
塑料料道D	塑料料道D	*	件	1
281031WX1	标签镶条	*	件	1
202001DX3	墙板	*	件	1

图 10-11　目的数据表 cc_ljk_copy

10.4　数据管道设计实例

10.4.1　创建数据管道

本实例将介绍使用 PowerBuilder 10.0 中的数据管道进行数据共享的方法。本实例创建了由 MyDataSource 数据库中 product 表到 MyDataDestination 数据库中 des_product 表的数据管道。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 pipeline 应用。

2. 创建数据管道

(1) 打开 Data Pipeline 画板。单击 PowerBar 上的 New 按钮，打开 New 对话框，并选择 DataBase 选项卡。然后在 DataBase 选项卡中选择 Data Pipeline 图标，并单击 OK 按钮，进入 New Data Pipeline 对话框。

(2) 配置数据源方式和数据库连接。数据源方式决定了在执行数据管道的时候，如何从源数据库中检索数据。PowerBuilder 被允许使用 4 种数据源方式，他们包括 Quick Select、SQL Select、Query、Stored Procedure，它们与建立 DataWindow 对象时选择的数据源方式相似。

- Quick Select：该方式生成针对一个表或者是共享一个键关系的多个表的 SQL 语句。如果要进行的数据管道是表对表之间的转移(例如本例)，则一般使用这种数据源方式。
- SQL Select：该方式实际上是图形化生成一个复杂 SQL 语句的方法。使用该方法可以从一个关系型源数据库中检索数据。一般用于源数据库表之间没有主键和外键相联系的情况。

- Query：该方式是由一个在 Query 画板中生成的 SQL SELECT 语句组成的查询对象，它的好处是可以作为一个独立的对象保存到库文件中。
- Stored Procedure：要传输的数据已经被定义为一个存储过程。只有当使用的 DBMS 支持存储过程时才能显示 Stored Procedure 数据源，实际上就是驻留在 DBMS 中的预先编译的 SQL SELECT 语句中 。

本例要实现的是表对表的数据转移，因此选择 Quick Select 方式即可。

在下面的两个列表框中已经显示了定义好的数据库连接，用户必须选择源数据库连接和目标数据库连接。按照实例的要求，我们选择 MyDataSource 作为源数据库连接，选择 MyDataDestination 作为目标数据库连接，如图 10-12 所示。然后单击 OK 按钮，进入 Quick Select 设置对话框。

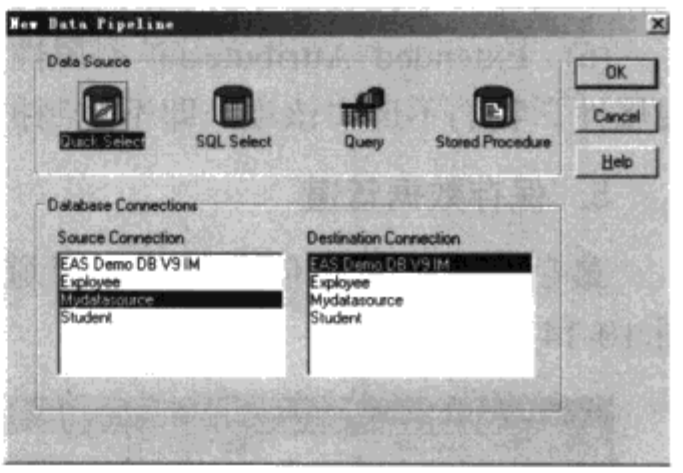


图 10-12　配置数据源方式及数据库连接

3. 设置 Quick Select

在这里我们选择要传输数据的表列，选择 product 表中的所有列作为数据传输列，将 MyDataSource 数据库连接 product 表中的所有数据传输到 MyDataDestination 中。当然，根据需要我们还可以只选择一部分列进行数据传输，单击 OK 按钮，进入数据管道属性设置工作区。

4. 设置数据管道属性

我们需要设置的数据管道属性包括以下选项：

(1) Table(目标表)：说明数据要传输到目标数据库连接中的哪个表，在这里我们输入“des_product”。

(2) Key(目标表主关键字)：说明目标表中所使用的主关键字的名称，这里我们取默认值。

(3) Options(管道传输选项)：说明对目标表的更新方式，包括以下 5 种方式：

Create-Add Table：创建指定的目标数据表，如果目标表已经存在，则提示重命名目标数据表，这里我们选择 Create-Add Table 方式。

a) Replace-Drop/Add Table：创建指定的目标数据表，如果目标表已经存在，则删除现有的数据表并创建一个新的数据表。

b) Refresh-Delete/Insert Rows：要求目标数据表存在时，删除目标数据表中的所有行，然后插入新的数据行。

c) Append-Insert Rows：要求目标数据表存在时，将源数据行插入到目标数据表中。

d) Update-Update/Insert Rows：关键字与目标表中某个数据行的关键字相匹配时，生成一个 UPDATE SQL 语句，由该语句修改目标表中相应行。关键字与目标表中某个数据行的关键字不匹配时，生成一个 INSERT SQL 语句，由该语句将源表中的相应行数据插入到

目标表中。

(4) Max Error(最大错误数)：说明在管道操作过程中出现多少个错误时停止管道操作，我们取默认值 100。

(5) Commit(事务提交频率)：说明当达到该项所指定的数据行总数时，在目标数据库连接上执行一个提交操作，我们取默认值 100。

(6) Extended Attributes(扩展属性)：说明是否复制任何与指定的源数据列相关联的扩展属性。我们不选中该项，即不复制扩展属性。

5. 保存数据管道

最后，我们将所创建的数据管道保存为 p_product，最终的实例效果如图 10-13 和图 10-14 所示。

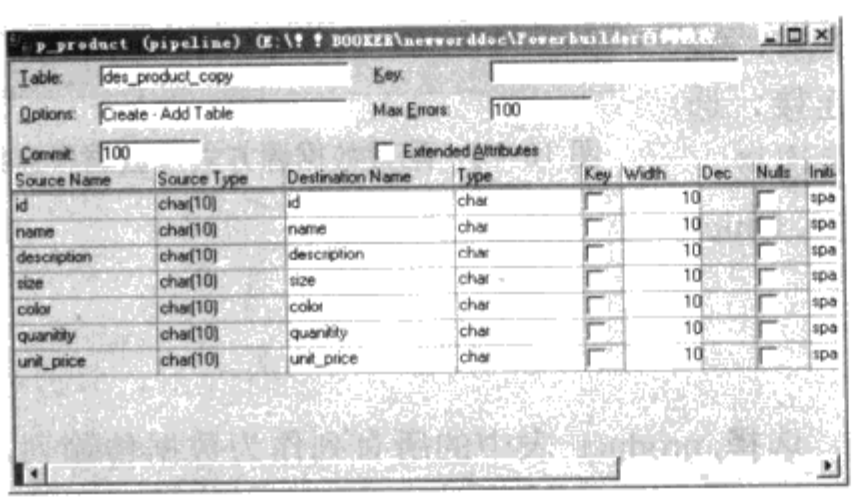


图 10-13 数据管道结构图

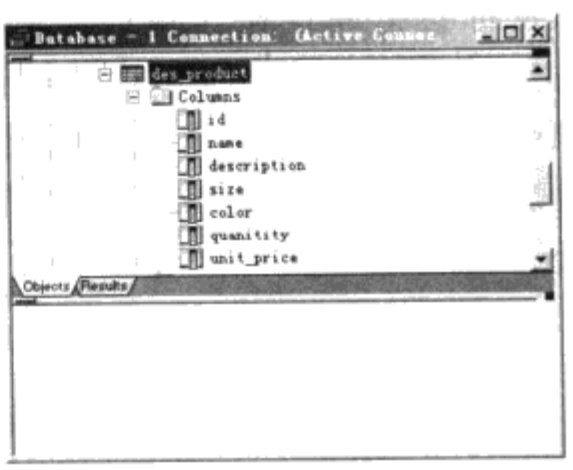


图 10-14 目标表及其数据

6. 执行数据管道操作

单击画板工具栏中的 Execute 按钮，所定义的数据管道就可以执行了。如果在 SQL SELECT 语句中制定了检索参数，则当执行管道传输时，PowerBuilder 会提示用户输入适当的值。然后，PowerBuilder 生成必要的 SQL 语句来转移管道传输选项所指定的数据。

当数据管道传输错误时，PowerBuilder 会将错误显示在一个标准的管道传输错误 DataWindow 中。该 DataWindow 为所有操作失败的数据行显示数据列和指出问题所在的错误信息。我们可以对出现问题的数据列做出适当修改，然后单击画板工具栏中的 Update DB 按钮，PowerBuilder 将尝试重新将这些数据行传输到目标数据库连接，并且在处理过程完成后，显示任何错误。

10.4.2 数据管道程序设计

本实例将介绍在应用程序中使用数据管道技术的方法。本实例创建了由 MyDataSource 数据库中的 3 个表到 MyDataDestination 数据库中的 3 个表的数据管道。数据管道建立以后，接着建立一个数据管道用户对象，利用该用户对象所包含的应用程序在执行数据管道操作时所需要的属性、事件和函数，实现数据管道相关操作，并获得数据管道执行时的读记录行数、写记录行数和出错记录行数。

1. 创建程序的基本框架

(1) 建立 workspace 工作区。

(2) 建立 datapipeline 应用。

(3) 创建数据管道。按照上一实例的操作方法，建立 3 个源表为 customer、department、employee 数据管道对象。其 Data Source 为 Quick Select，Source Connection 为 student 数据库，Destination Connection 为 MyDataDestination，其目标表分别命名为 des_customer、des_department、des_employee。最后，我们将所创建的数据管道分别保存为 p_customer、p_department、p_employee。

(4) 创建 Data Pipeline 用户对象，建立名为 u_pipeline 的 Data Pipeline 标准类用户对象。

(5) 建立 w_pipe 窗口，其类型为“main！”，其 Title 为“数据管道之传递数据”。

(6) 建立控件，在 w_pipe 窗口上添加如表 10-2 所示的控件。

表 10-2　本实例控件列表

控件名	Name	用　途
StaticText	st_ select	选择数据管道对象的标签
	st_ errorinfo	显示管道操作错误信息标签
	st_read	显示读数据行数
	st_written	显示写数据行数
	st_error	显示出错数据行数
DataWindow	dw_error	显示出错信息
ListBox	lb_select	数据管道对象列表
GroupBox	gb_read	读数据行数分组框
	gb_written	写数据行数分组框
	gb_error	显示出错数据行数分组框
CommandButton	cb_start	开始数据管道操作按钮
	cb_stop	停止数据管道操作按钮
	cb_repair	修复记录按钮
	cb_exit	退出本程序按钮

(7) 建立一个名为 d_list 的数据窗口对象，其源表为 MyDataSource 数据库中建立的 list 表，该表用以装载数据管道操作的出错信息。

2. 编写驱动事件代码

(1) 为 Data Pipeline 用户对象声明静态文本框型实例变量，分别用来显示数据管道操作的读行数、写行数、出错行数：

```
StaticText ist_read,ist_written,ist_error
```

(2) 为 pipeline 事件编写如下代码：

```
ist_read.text=String(RowsRead)
ist_written.text=String(RowsWritten)
```

```
ist_error.text=String(RowsInError)
//将读数据行数、写数据行数、出错数据行数值赋给用以显示的静态文本框
```

(3) 为主窗口定义实例变量如下：

```
u_pipeline u_pipe
Transaction sourcetrans,destinationtrans
```

(4) 在主窗口 w_pipe 的 Open 事件中编写如下代码：

```
int i,count
String ls_entries
Datastore ds_list
//定义变量
ls_entries=LibraryDirectory("…\datapipeline.pbl",DirPipeline!)
//从库文件得到所有 Datapipeline 对象的目录
ds_list=Create Datastore
ds_list.dataobject="d_list"
ds_list.Reset()
ds_list.ImportString(ls_Entries)
//将得到的 Datapipeline 对象目录放入数据存储对象 ds_list
count=ds_list.Rowcount()
for i=1 to count
    lb_select.additem(ds_list.getitemstring(i,1))
next
//从 ds_list 中将各 Datapipeline 对象的名称添加到 lb_select 列表框中供用户选择
lb_select.SelectItem(1)
lb_select.PostEvent(selectionchanged!)
sourcetrans=CREATE Transaction
…
//连接到目标数据库
u_pipe=CREATE u_pipeline
u_pipe.ist_read=st_read
u_pipe.ist_written=st_written
u_pipe.ist_error=st_error
//为显示读行数、写行数和出错行数赋值
```

(5) 在 cb_start 控件的 Clicked 事件中编写如下代码：

```
int li_result
cb_stop.enabled=true
dw_error.reset()
u_pipe.DataObject=lb_select.selectedItem()
li_result=u_pipe.start(sourcetrans,destinationtrans,dw_error)
choose case li_result
    …
end choose
//判断开始管道操作
cb_stop.enabled=false
```

在 cb_stop 控件的 Clicked 事件中编写如下代码：

```
this.enabled=false
if u_pipe.cancel()=1 then
            …
end if
//判断停止管道操作
```

在 cb_repair 控件的 Clicked 事件中编写如下代码：

```
if u_pipe.repair(destinationtrans)<>1 then
        messagebox("错误","修复目标表的记录时出错!")
end if
```

(6) 在主窗口 w_pipe 的 Close 事件中编写断开数据库连接的代码：

```
disconnect;
```

(7) 在 cb_exit 控件的 Clicked 事件中编写关闭窗口的代码：

```
close;
```

(8) 单击 PowerBuilder 10.0 开发界面上的按钮或按 Ctrl+R 快捷键，程序就可以运行。选择一个数据管道对象，单击“开始管道操作”按钮，效果如图 10-15 所示。

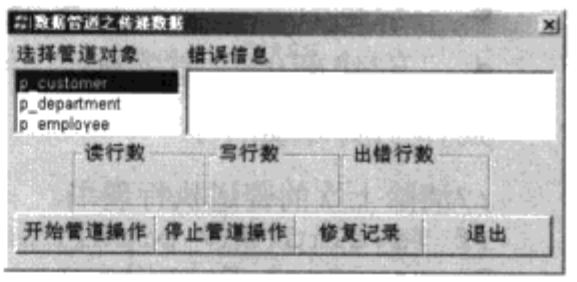


图 10-15　实例效果图

10.4.3　系统集成

在实际应用开发过程中，常常会遇到两个不同应用系统之间需要进行数据或信息共享的情况。在这时，可以采用所提供的数据管道技术，来相互读取所需要的数据到自己的数据库表中。

本节的任务是应用数据管道技术实现系统集成。为了应用数据管道技术实现系统集成，需要创建一个系统集成控制窗口。在该窗口中，可以一次执行多个数据管道，同时，可以对数据管道的执行进行锁定，通过窗口函数和按钮事件来实现系统集成的功能。上一节已经介绍了有关数据管道的创建和派生数据管道子对象的方法和步骤，这里将把这些内容省略，主要介绍通过窗口函数和按钮事件来实现系统集成的功能。

本例主要利用窗口函数和按钮事件的定义来实现系统集成的功能。首先建立一个窗体，步骤如下：

(1) 在 PowerBuilder 的工具栏中单击 New 按钮，在随后弹出的对话框中选择 PB Object 选项卡。在其中选择 Window 图标，单击 OK 按钮。

(2) 然后，PowerBuilder 会打开该窗体的数据画板。给该窗体命名为 w_pl，把标题设定为系统集成控制。

(3) 下面设置窗体中的控件及其属性。窗体 w_pl 的上半部分是一个显示待执行数据管道的数据窗口控件，其后台对应一个包含数据管道相关信息的数据窗口对象，如数据管道名称、源数据库配置等，最底层存放数据管道信息的是数据库表。窗体的下半部分是管道出错显示数据窗口的控件。窗口右边是一系列系统功能按钮，用于实现相应的数据管道操作。

具体的控件如下：

```
数据窗口：dw_1、dw_2。
命令按钮：cb_1~ cb_4。
cb_1.text=数据库。
cb_2.text=执行。
cb_3.text=中断。
cb_4.text=退出。
```

(4) 编写窗口、函数及各个控件的脚本。这里需要定义窗口实例变量、定义窗口函数、编写各个按钮的脚本。其中，执行按钮的脚本是整个程序的核心。

① 函数 wf_create_pipeline_instance()的脚本

函数 wf_create_pipeline_instance(long al_row)returns integer 的功能是：

- 清除上次的管道执行现场。
- 根据管道源数据的配置，初始化源数据库连接参数，并连接数据库。
- 创建新的管道实例，并初始化内部参数。

具体的脚本如下：

```
//清除上次的管道执行现场。
If IsValid(inv_pipeline)then Destroy inv_pipeline
Dw_messages.DataObject=  //清除出错数据窗口控件中的内容
St_status.Text=  //清除执行状态行中的内容
//根据管道源数据的配置，初始化源数据库联接参数，并连接数据库
If wf_initial_sourcetrans(al_row)<0 then return -1
//创建新的管道实例，并初始化内部参数
Inv_pipeline=create n_pl
Inv_pipeline.uf_initialize(itr_sourcetrans,SQLCA,dw_messages,st_status)
Return 0
```

② 函数 wf_initial_sourcetrans()的脚本

函数 wf_initial_sourcetrans (long al_row)returns integer 的功能是连接数据库。脚本如下：

```
String ls_cursourcedb
String ls_sourcedb,ls_parm[]

//比较当前连接的数据库是否与管道数据库相同
If dw_pipelines.RowCount()>0 then
  Ls_cursourcedb=dw_pipelines.Object.source_db[ii_sourcedb]
Else//没有待执行的管道
  Return -1
  End if
  Ls_sourcedb=dw_pipelines.Object.source_db[al_row]
  If al_row<>1 and Upper(ls_sourcedb)=Upper(ls_cursourcedb) then return 1   //
  数据库未变
  Ii_sourcedb=al_row

  //分配新的数据库连接参数
  if UpperBound(ls_parm)<>8 then return -1
  Disconnect Using itr_sourcetrans;
  ……
  connect Using tr_sourcetrans;
  if tr_sourcetrans.SqlCode<0 then
    MessageBox("错误提示：","连接数据库失败！ ")
    Return -1
  End if
  Return 0
```

③ 执行(cb_2.clicked())的脚本

单击此按钮，执行利用数据管道进行数据传递的操作，以此来实现系统集成的功能。具体脚本如下：

```
//变量声明，略
SetPointer(HourGlass!)
Ib_stop=False
Do while ll_row<=dw_pipelines.RowCount()
Ll_selected_row=dw_pipelines.GetSelectedRow(ll_row)
If ll_selected_row<1 then
  Exit
```

```
Else
  Ll_row=ll_selected_row
  Ls_validation=dw_pipelines.Object.pipeline_validation[ll_selected_row]
  If ls_validation<> 'y' then    //管道被锁定
Dw_pipelines.SelectRow(ll_selected_row,False)
Continue
End if
If wf_create_pipeline_instance(ll_selected_row)<0 then exit
Ls_pipelinename=dw_pipelines.Object.pipeline_name[ll_selected_row]
Inv_pipeline.dataobject=ls_pipelinename

//执行管道，计算时间
……
//清除执行好的管道
……
//管道执行结果处理
if li_rtn<>1 then
  dw_pipelines.Object.extend_field1[ll_selected_row]=F
  inv_pipeline.uf_Error(li_rtn)
else
  if dw_messages.RowCount()>0 then   //没有更新目标数据表
  dw_pipelines.Object.extend_field[ll_selected_row]=P
  else
dw_pipelines.Object.extend_field1[ll_selected_row]=S
  end if
  dw_pipelines.Object.extend_field2[ll_selected_row]=st_status.Text
end if
  //出错处理
……
  end if
loop
SetPointer(Arrow!)
```

至此，完成了利用数据管道进行数据传递的操作来实现系统集成的功能的所有步骤。

完成上述工作，就得到了一个利用数据管道进行数据传递的操作来实现系统集成的窗口，具体的效果如图 10-16 所示。在这个窗口中，可以实现系统集成的操作。

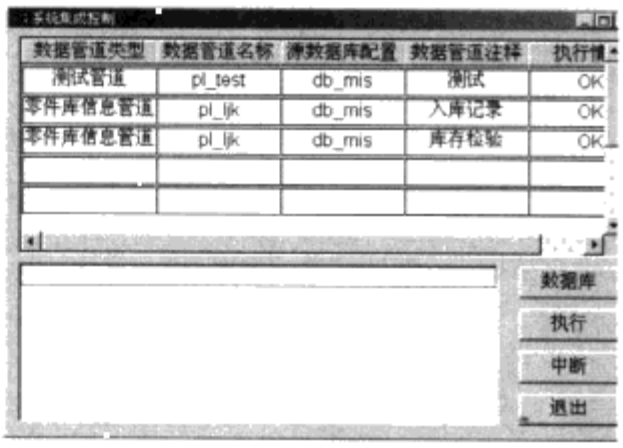


图 10-16　系统集成的实例效果

10.5　习　　题

1. 填空题

(1) 为了在一个应用程序中实现数据管道，需要建立以下 3 个不同的对象：_____、_____、_____。

(2) _____为管道操作提供一个用户界面，通过这个界面可以与管道对象进行交互，对管道操作进行监控。

(3) 在脚本中准备两个事务对象，让一个事务对象连接_____，让另一个事务对象连接_____。

(4) 一旦数据窗口控件中放入了错误行，用户可以有两项选择：_____和_____。

2. 问答题

(1) PowerBuilder 提供了哪两种不同方法来使用数据管道？

(2) 如何创建管道对象，在数据管道画板中能够为数据管道定义哪些特征？

(3) 如何创建支撑用户对象？

(4) 如何启动管道操作？

第 11 章　用户对象和用户事件技术

PowerBuilder 10.0 提供了大量的控件和对象，使用这些控件可以建立标准的 Windows 应用程序。大多数情况下，这些控件能够满足开发 Windows 应用程序界面的需要。但是对于某些特殊的情况，如需要错误对象能够返回中文的出错信息，这些标准的控件就用不上了。对于这种特殊情况，PowerBuilder 也提供一种解决方案，使用用户对象开发符合自己需要的自定义对象。用户对象实际上是对 PowerBuilder 的一种扩展，熟练使用 PowerBuilder 提供的标准控件和用户自定义对象，能够满足各种各样的程序开发需求。

用户对象是 PowerBuilder 提供的一种对自身进行扩展的机制，使用用户对象可以开发出符合自己要求的对象。用户对象和 PowerBuilder 提供的标准对象互相配合，为 PowerBuilder 提供了广阔的应用空间。

11.1　用户对象概述

用户对象同数据窗口一样，也是 PowerBuilder 的一个亮点。用户对象为开发人员提供了一种扩展 PowerBuilder 集成开发环境的途径。用户对象不但可以开发 PowerBuilder 不能提供的对象，而且还可以将一些经常使用的代码或控件封装在一起，提高软件的可重用性。软件的可重用性是指同一段代码或对象不经过修改或稍加修改就能够重复使用。很显然，软件重用可以节省开发成本，提高软件的生产效率。PowerBuilder 是一个面向对象的开发语言，对软件的重用性有很好地支持。为全面了解用户对象，应首先了解面向对象的编程技术。

11.1.1　面向对象编程技术

以前为了提高程序的可重用性，只是单纯地建立许多的标准程序库，但实际上这些标准程序库除了一些基本的标准函数被经常使用外，其他函数都要针对不同系统进行修改，因此重用性并不好。到了 20 世纪 80 年代，面向对象语言的出现使得软件开发由原来的结构化开发转变为面向对象编程(OOP)，从而大大提高了软件的开发效率，推动了软件的巨大进步。

在面向对象的编程中需要尽可能地模仿人类的思考和处理问题方式，将人类解决和处理问题的过程以程序化的方法表现出来。面向对象编程中经常用到的两个概念是类(Class)和对象(Object)。

类是对客观事物的一个抽象，是在较高层次上描述事物的一种方法；对象是类的一个具体表现，类的功能由对象来实现。在面向对象的编程中，认为客观世界是由对象组成的，任何的事务都是对象，复杂的事务也是由简单的对象组合成的。对这些对象进行合理的抽象，将一些共同的特点提取出来并用程序的语言表达出来，就形成了计算机中的类。类实际上是一种特殊结构的数据类型，它有自己的属性(Property)、方法(Method)和事件(Event)，

用来描述一些非常相似的对象所具有的共同行为和特征，例如：“人类”表示人所具有的共同行为和特征。对象是类的一个具体表现，对象拥有类所描述的全部属性、方法和事件。如果说类是在宏观上描述事物，那么对象则是在微观上描述事物，例如：一名程序员就是“人类”的一个具体体现，具有“人类”的共同特征，如能够学习、劳动等。

继承性是面向对象的一个重要特征，通过继承能够获得父类已有的性质和特征，不必重新定义和编写原有的代码来实现。在子类中可以扩充一些子类所特有的属性、方法和事件，这样一级一级地继承下来最终会形成一个丰富的类树，有时也叫家族树，就如同一个家谱。例如：从“人类”可以派生出脑力劳动者类和体力劳动者类，而程序员类则是由脑力劳动者类继承而来，这样的继承关系如图 11-1 所示。

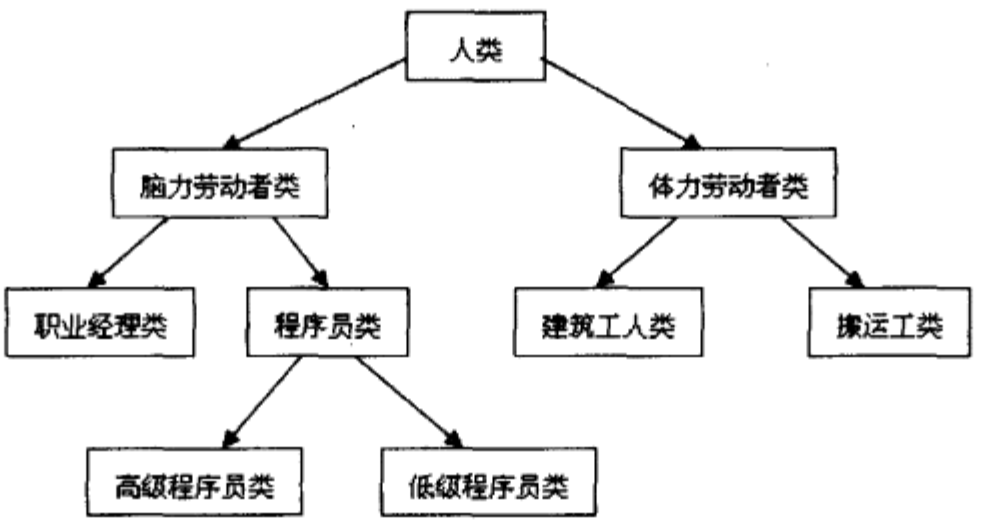


图 11-1 类树

继承的一个最大的好处就是重用代码，通过继承可以获得从父类继承来的全部东西，通过继承的方式可以新建一个窗口对象或者一个菜单对象。

注意： 子类并不一定能够继承父类的全部东西，子类只能继承那些父类允许继承的东西，这个特点是由面向对象的封装性决定的。

11.1.2 用户对象的分类

PowerBuilder 把用户对象分为两类，包括可视用户对象(Visual User Object)和类用户对象(Class User Object)。各种类型用户对象的作用和使用方法也不完全相同，下面对各种用户对象类型分别进行介绍。

1. 可视用户对象

可视用户对象(Visual User Object)都是可见的，用户可以在界面上看到并使用该对象。可视用户对象主要用于完成应用程序和用户之间的交流，如命令按钮可以接受用户操作，并执行相应功能，单行编辑框可以接受用户输入的信息等。可视用户对象可以作为一个控件被重复使用，也可以为可视用户对象添加脚本以实现某种功能或者添加一些事件并在适当的时候触发这些事件，这样就可以像添加一个标准控件一样在窗口中使用可视的用户对象。可视用户对象包括标准可视用户对象(Standard Visual User Objects)、定制可视用户对象(Custom Visual User Objects)和外部可视用户对象(External Visual User Objects)。

(1) 标准可视用户对象。标准可视用户对象(Standard Visual User Objects)是一个从可视的系统标准控件继承而来的用户对象，通过添加脚本代码和修改属性，该对象可以在窗口中被当作一个控件使用。例如，经常在窗口中放置一个命令按钮，当单击该按钮时弹出对话框询问是否关闭窗口，用户确认后关闭当前窗口。为此，可以从命令按钮继承一个标准可视用户对象，再让这个继承来的标准可视用户对象完成上述操作，使用时只要将这个标准可视用户对象放置在窗口中就可以了。标准可视用户对象的工作原理如图 11-2 所示。

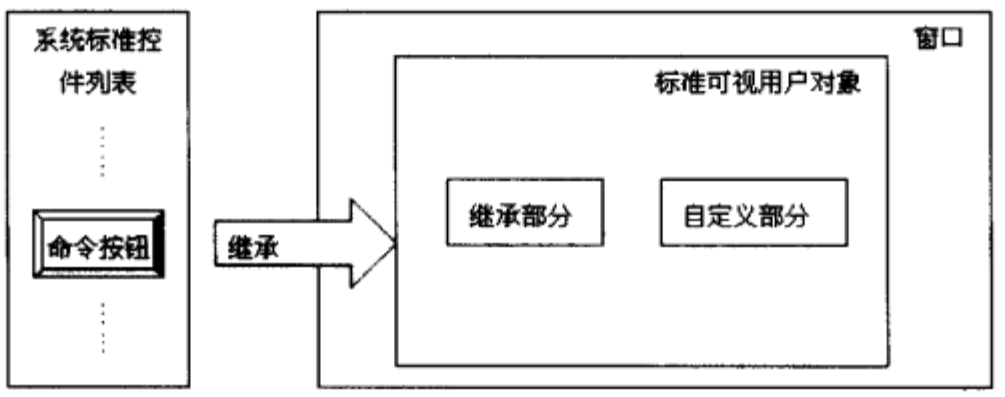


图 11-2　标准可视用户对象的工作原理

提示： 可用于派生标准可视用户对象的 PowerBuilder 标准控件包括文本框类控件、位置指示类控件、列表类控件、视图类控件、图形类控件和其他类控件。

(2) 定制可视用户对象。定制可视用户对象(Custom Visual User Objects)可以包含多个系统标准控件和函数等单元，可以作为一个标准的控件单元在窗口中使用。从某种意义上，可以把定制可视用户对象看作一个在窗口中使用的独立的窗口。标准可视用户对象中可以包括多个控件，这些控件在一起被作为一个整体使用，例如：将经常使用的打印预览命令按钮、打印按钮和关闭窗口按钮作为一组放在一个定制可视用户对象内，使用时只要在窗口上添加这个定制可视用户对象就可以了，从而大大提高了开发效率。定制可视用户对象的工作原理如图 11-3 所示。

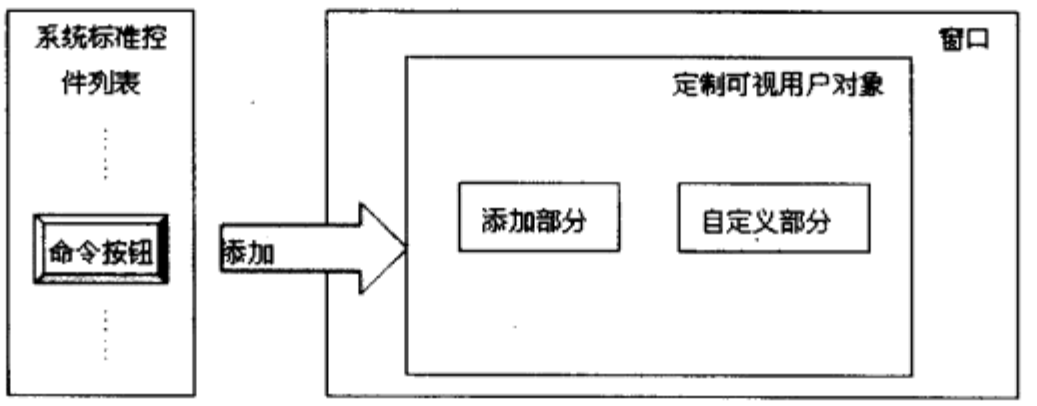


图 11-3　定制可视用户对象的工作原理

(3) 外部可视用户对象。外部可视用户对象(External Visual User Objects)包含一个来自于外部动态链接库的可视控件，也可以像窗口中其他控件一样使用。例如：购买别人的控件库或者因为 PowerBuilder 不能实现某些功能，而必须使用其他语言开发等情况，这时就需要使用外部可视用户对象来将外部定义的可视控件添加到 PowerBuilder 的开发环境中。在对执行效率有要求的情况下，也可以使用 C++语言开发某些控件，然后再通过外部可视

用户对象引入 PowerBuilder 中。外部可视用户对象的工作原理如图 11-4 所示。

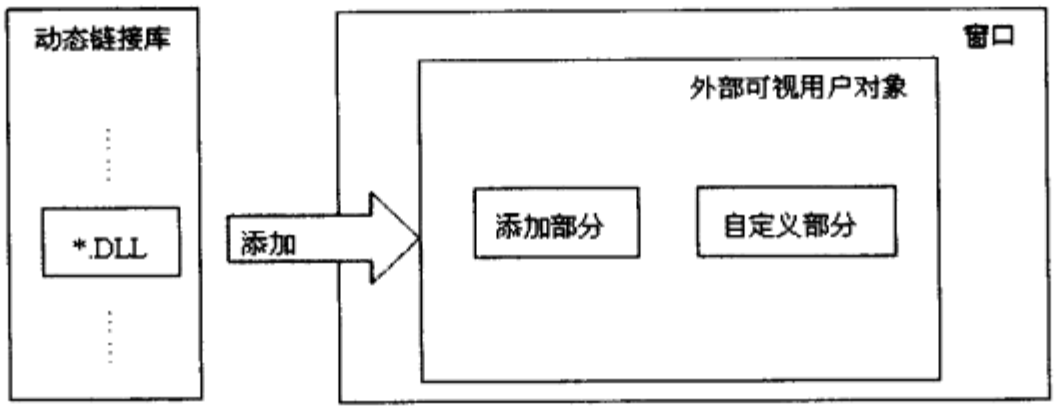


图 11-4 外部可视用户对象的工作原理

注意： 要使用某个动态链接库创建外部可视用户对象，就必须了解该动态链接库支持哪些类、响应什么消息以及包含哪些显示属性等。

2. 类用户对象

类用户对象不能在窗口上显示，它们主要用于将一些可重用的业务逻辑或者属性和函数封装起来。创建类用户对象时需要指定实例变量(Instance Variables)和对象级函数(Object-Level Function)。类用户对象包括定制类用户对象(Custom Class User Objects)和标准类用户对象(Standard Class User Objects)。

(1) 定制类用户对象。定制类用户对象(Custom Class User Objects)用于将函数、实例变量或结构封装在一起，实现某个业务规则或者处理过程，如对一组数据进行统计计算出最大值、最小值和平均值。定制类用户对象的工作原理如图 11-5 所示。

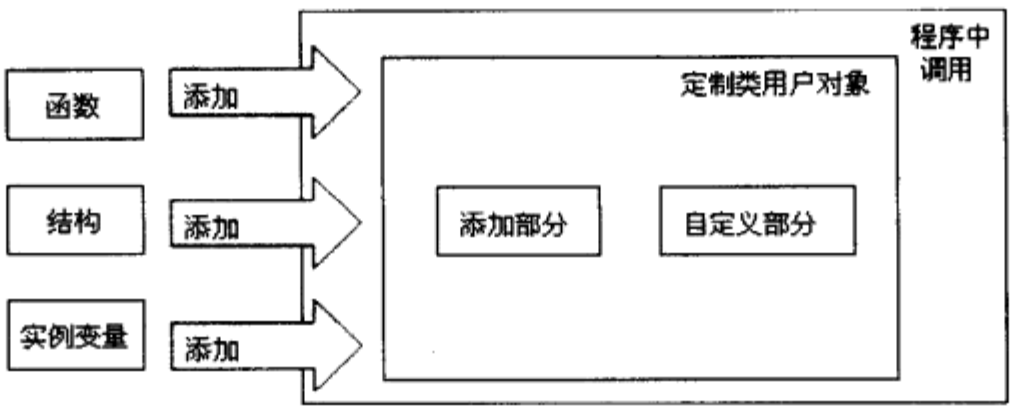


图 11-5 定制类用户对象的工作原理

(2) 标准类用户对象。标准类用户对象(Standard Class User Objects)从 PowerBuilder 中不可视对象(如事务对象、定时对象)继承而来，它可以修改标准类用户对象，可以增加实例变量或者函数以增强 PowerBuilder 中对象的功能。创建了标准类用户对象后，就可以在应用程序中使用创建的对象，而不是使用 PowerBuilder 提供的对象。标准类用户对象的工作原理如图 11-6 所示。

注意： 在各种用户对象中自定义部分不一定是必需的部分，自定义部分通常是用于对用户对象的扩展，可以根据需要选择是否进行扩展。但是对于大多数的情况下，都会为用户对象添加脚本代码进行扩展。

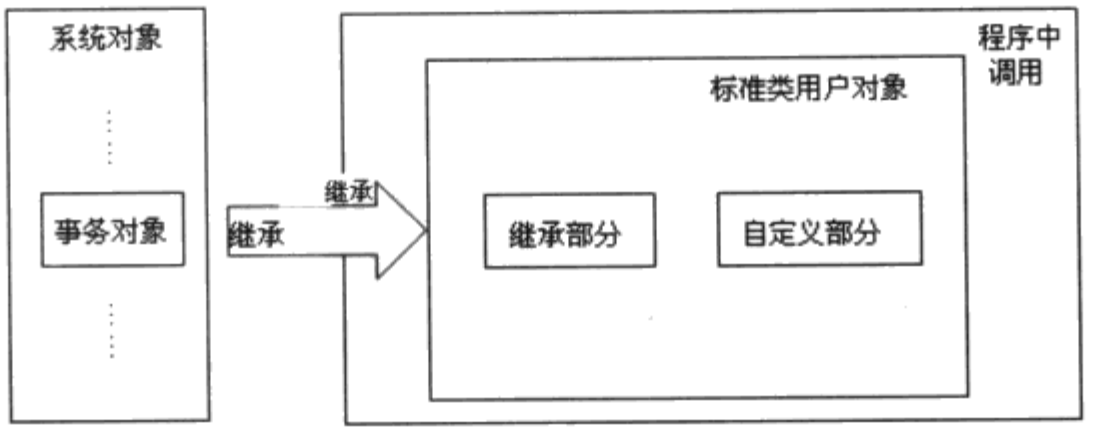


图 11-6　标准类用户对象的工作原理

11.2　创建用户对象

本节将介绍对不同用户对象的创建方法。对于每种用户对象，既可以通过创建向导建立一个全新的用户对象，又可以通过继承的方式创建用户对象，本节首先介绍通过创建向导创建用户对象的方法，然后再介绍通过继承方式创建新用户对象的方法。

使用创建向导创建用户对象是通过 New 对话框完成的，New 对话框的 PB Object 选项卡中的前 5 个图标用于创建不同类型的用户对象，如图 11-7 所示。

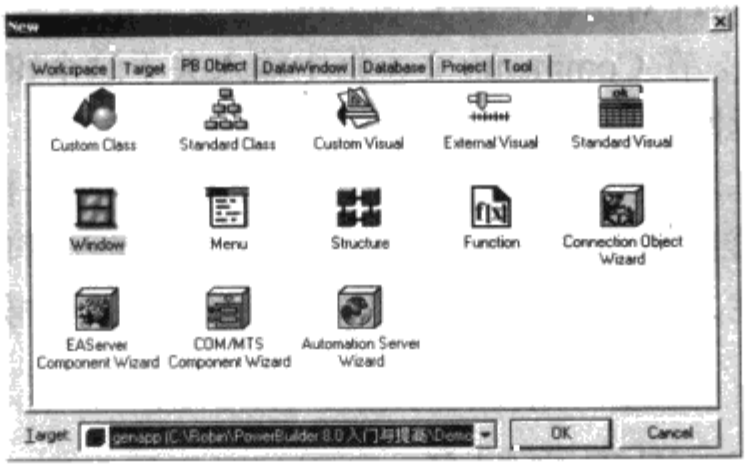


图 11-7　通过 New 对话框创建用户对象

11.2.1　创建标准可视用户对象

创建标准可视用户对象的步骤如下：

(1)　单击工具栏上的 New 按钮，打开 New 对话框，如图 11-7 所示。

(2)　选择 PB Object 选项卡，单击 Standard Visual 图标，创建标准可视用户对象，然后单击 OK 按钮，打开 Select Standard Visual Type 对话框，如图 11-8 所示。

(3)　在 Select Standard Visual Type 对话框中选择要继承的 PowerBuilder 标准控件，这里选择命令按钮控件(commandbutton)，然后单击

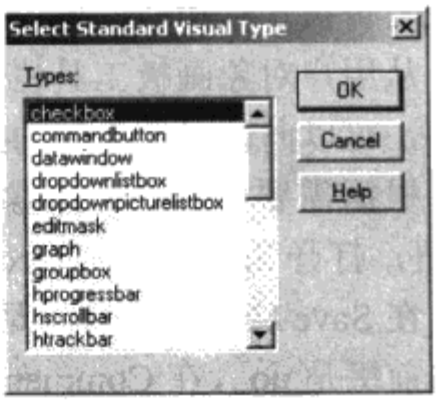


图 11-8　Select Standard Visual Type 对话框

OK 按钮，打开用户对象画板，如图 11-9 所示。

(4) 在用户对象画板的 Properties 视图中设置新建对象的属性，设置完成后单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-10 所示。

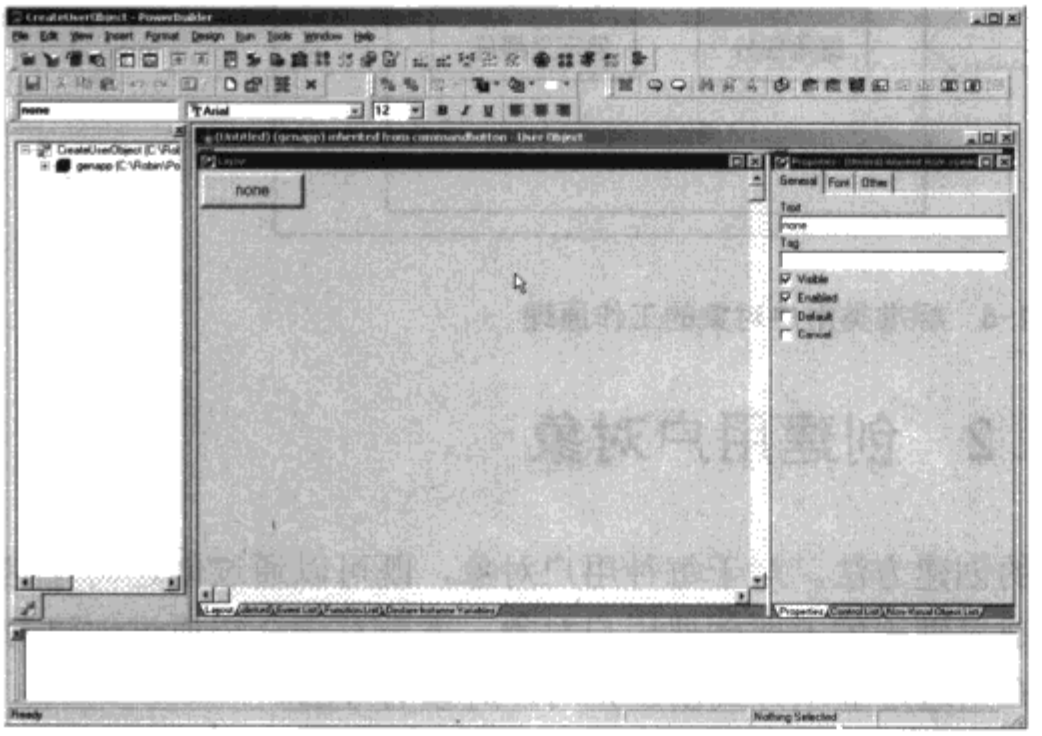


图 11-9 用户对象画板

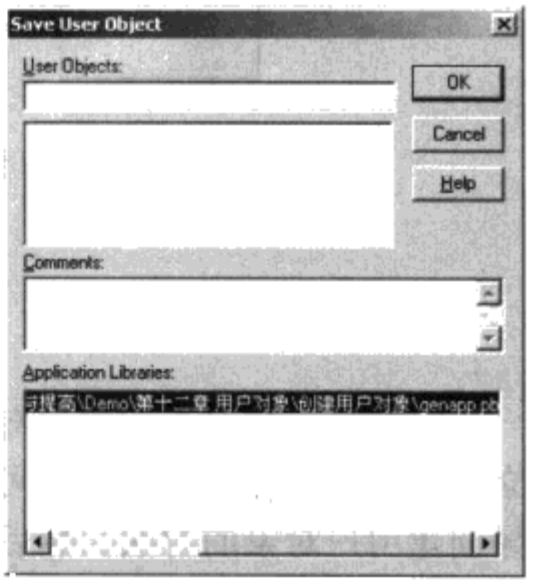


图 11-10 Save User Object 对话框

(5) 在 Save User Object 对话框中的 User Objects 文本框中输入要保存的对象名称，用户对象的默认前缀是 uo_，在 Comments 文本框中输入关于这个窗口的注释，然后单击 OK 按钮，保存用户对象并返回用户对象画板。

至此，创建了一个基本的标准可视用户对象，如果不增加任何的设置，这个新创建的对象和 PowerBuilder 标准的命令按钮控件是一样的。可以像在窗口上编辑控件一样，在布局视图中修改对象的属性、在 Script 视图中添加用户对象的事件处理代码或者定义需要的用户事件等。

11.2.2 创建定制可视用户对象

创建定制可视用户对象的步骤如下：

(1) 单击工具栏上的 New 按钮，打开 New 对话框，如图 11-7 所示。

(2) 选择 PB Object 选项卡，单击 Custom Visual 图标创建定制可视用户对象，然后单击 OK 按钮，打开用户对象画板，此时用户对象画板的 Layout 视图中有一个空白的区域，这个区域与窗口非常相似，可以在上面添加控件，如图 11-11 所示。

(3) 从用户对象画板工具栏中的控件列表复合按钮中选择控件，向定制可视用户对象添加控件，可以向该对象添加的控件如图 11-12 所示。

(4) 根据需要修改用户对象的属性，并为用户对象编写脚本，然后单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-10 所示。

(5) 在 Save User Object 对话框中的 User Objects 文本框中输入要保存的名字，用户对象的默认前缀是 uo_，在 Comments 文本框中输入关于这个窗口的注释，然后单击 OK 按钮，保存用户对象并返回用户对象画板。

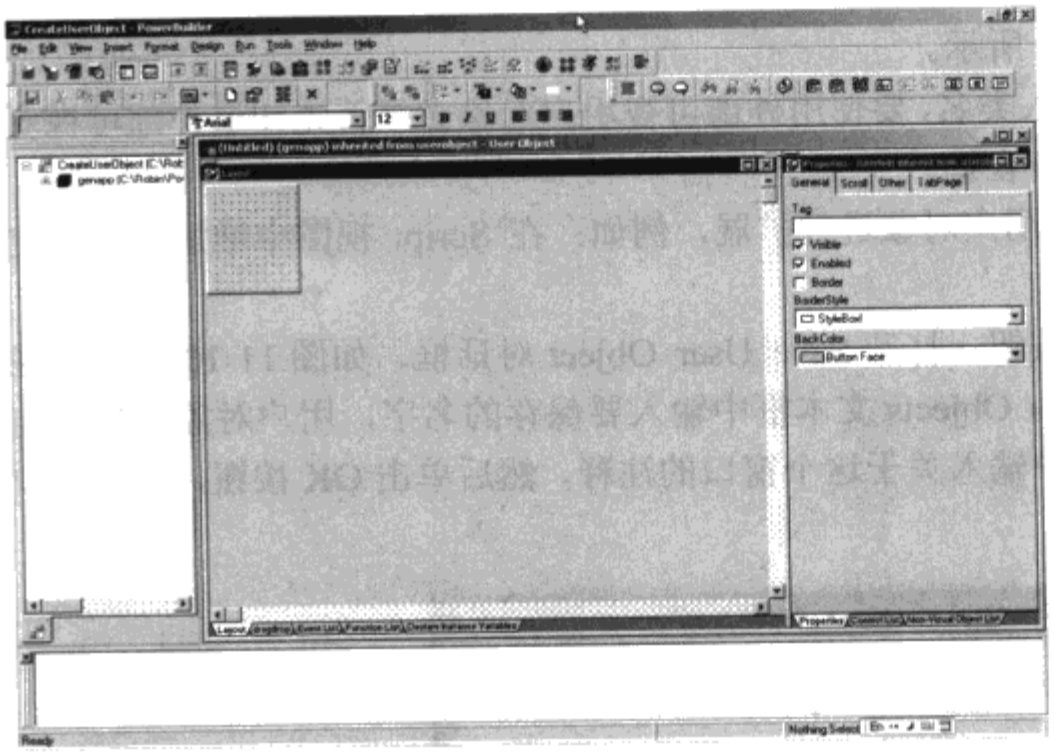


图 11-11　定制可视用户对象

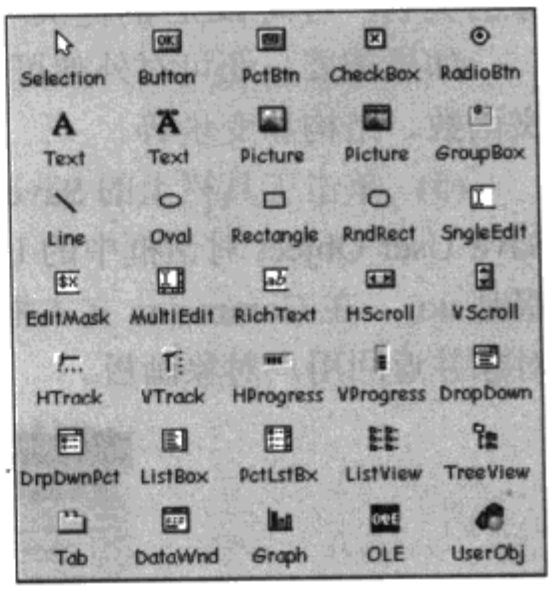


图 11-12　可以向定制可视用户对象控件添加的控件

11.2.3　创建外部可视用户对象

创建外部可视用户对象的步骤如下：

(1)　单击工具栏上的 New 按钮，打开 New 对话框，如图 1i-7 所示。

(2)　选择 PB Object 选项卡，单击 External Visual 图标创建外部可视用户对象，然后单击 OK 按钮打开用户对象画板，此时画板的 Layout 视图中有一个空白的区域，这个区域用于显示外部的可视控件，如图 11-13 所示。

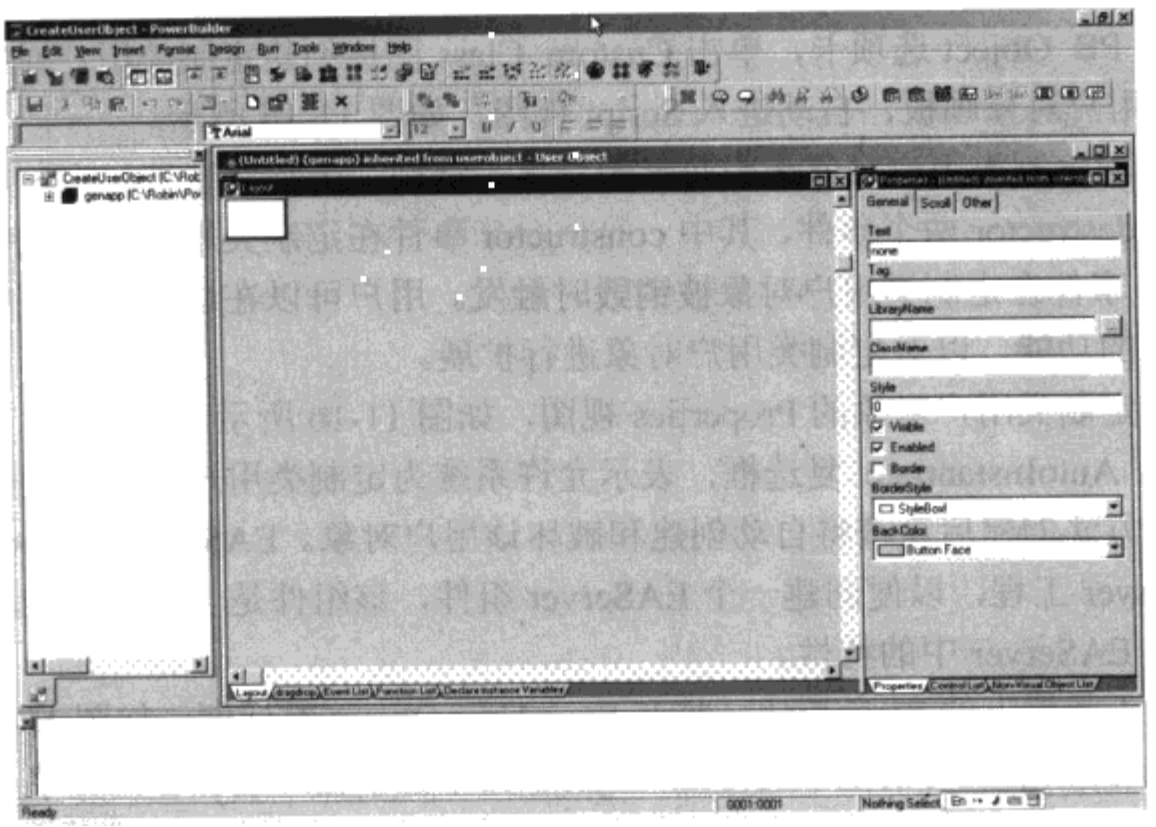


图 11-13　外部可视用户对象

(3)　打开 Properties 视图，单击 LibraryName 属性右边的小按钮，打开 Select Custom

Control DLL 对话框，如图 11-14 所示。

(4) 在 Properties 视图中输入类名，要使用外部可视用户对象必须首先知道外部可视对象的类名，可向 DLL 的提供厂商咨询。

如果需要，还可对外部可视用户对象进行扩展，例如：在 Script 视图中增加代码，定义函数、结构和变量等。

(5) 单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-10 所示。在 Save User Object 对话框中的 User Objects 文本框中输入要保存的名字，用户对象的默认前缀是 uo_，在 Comments 文本框中输入关于这个窗口的注释，然后单击 OK 按钮，保存用户对象并返回用户对象画板。

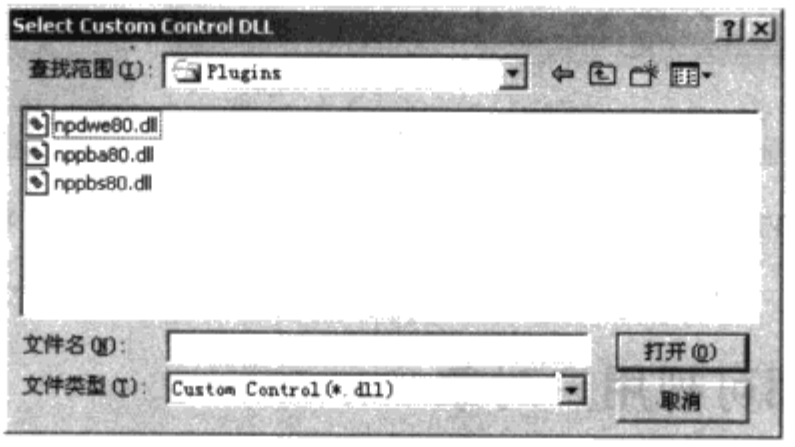


图 11-14 Select Custom Control DLL 对话框

11.2.4 创建定制类用户对象

创建定制类用户对象的步骤如下：

(1) 单击工具栏上的 New 按钮，打开 New 对话框，如图 11-7 所示。

(2) 选择 PB Object 选项卡，单击 Custom Class 图标创建定制类用户对象，然后单击 OK 按钮打开用户对象画板，自动进入 Script 视图，如图 11-15 所示。

为新创建的定制类用户对象声明必要的变量、结构、函数及事件。定制类用户对象有 constructor 和 destructor 两个事件，其中 constructor 事件在定制类用户对象被创建的时候触发，destructor 事件在定制类用户对象被销毁时触发。用户可以在这两个事件中添加脚本代码，完成适当的功能，以对定制类用户对象进行扩展。

(3) 打开定制类用户对象的 Properties 视图，如图 11-16 所示。

(4) 选中 AutoInstantiate 复选框，表示允许系统为定制类用户对象创建一个默认的实例化方法，该方法创建后系统将自动创建和破坏该用户对象。EAServer Project 属性用于指定一个 EA Server 工程，以便创建一个 EAServer 组件，该组件是一个定制类用户对象，具有能够部署到 EAServer 中的特性。

(5) 单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-10 所示。

在 Save User Object 对话框中的 User Objects 文本框中输入要保存的名字，用户对象的默认前缀是 uo_，在 Comments 文本框中输入关于这个窗口的注释，然后单击 OK 按钮，保存用户对象并返回用户对象画板。

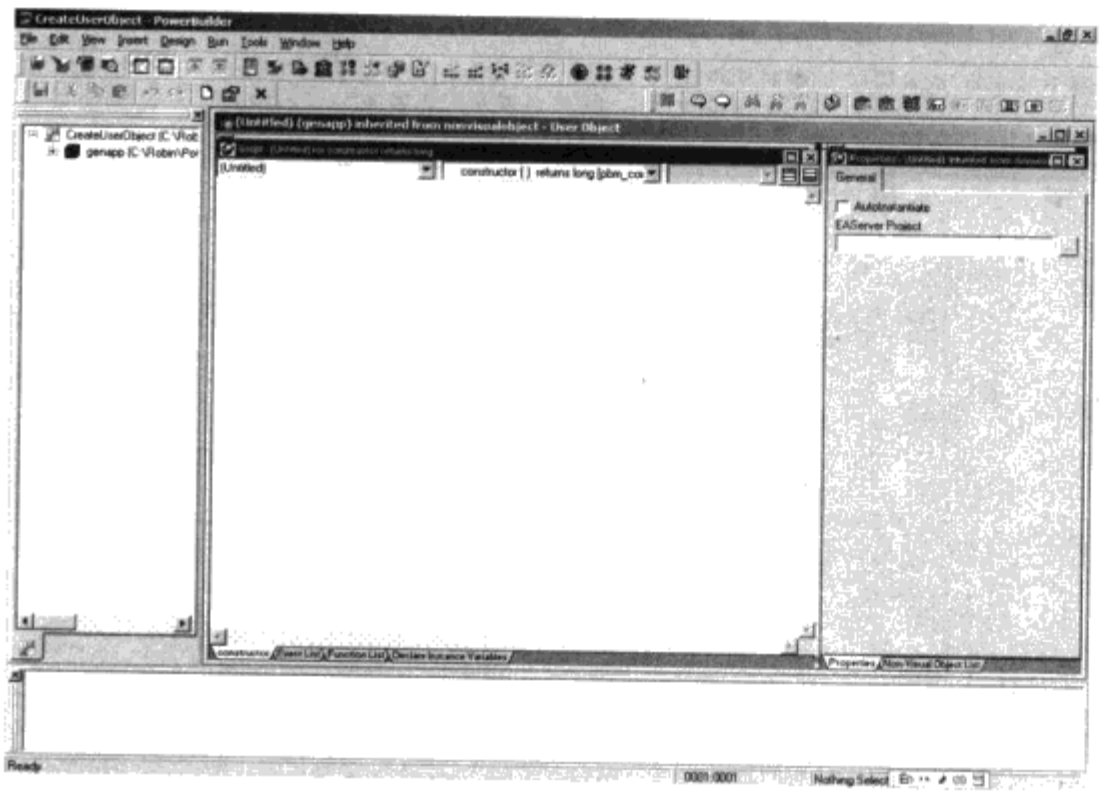


图 11-15　定制类用户对象

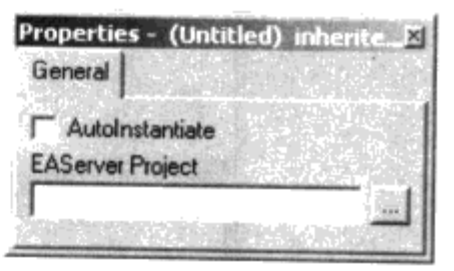


图 11-16　定制类用户对象的 Properties 视图

11.2.5　创建标准类用户对象

创建标准类用户对象的步骤如下：

(1)　单击工具栏上的 New 按钮，打开 New 对话框，如图 11-7 所示。

(2)　选择 PB Object 选项卡，单击 Standard Class 图标创建标准类用户对象，单击 OK 按钮，打开 Select Standard Class Type 对话框，如图 11-17 所示。

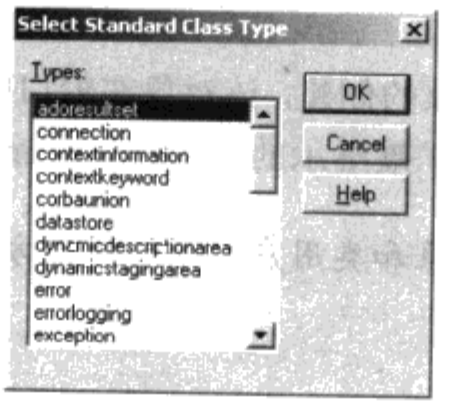


图 11-17　Select Standard Class Type 对话框

(3)　在 Select Standard Class Type 对话框中选择要继承的标准 PowerBuilder 对象，这里选择 error 对象，然后单击 OK 按钮，创建标准类用户对象。标准类用户对象创建完成后，自动打开用户对象画板，并显示 Script 视图，如图 11-18 所示。

注意：　创建的标准类用户对象是从 error 对象继承来的，因此 Properties 视图中显示的属性都是关于 error 对象的，如果是从其他对象创建的标准类用户对象，Properties 视图中显示的属性也会随之改变。

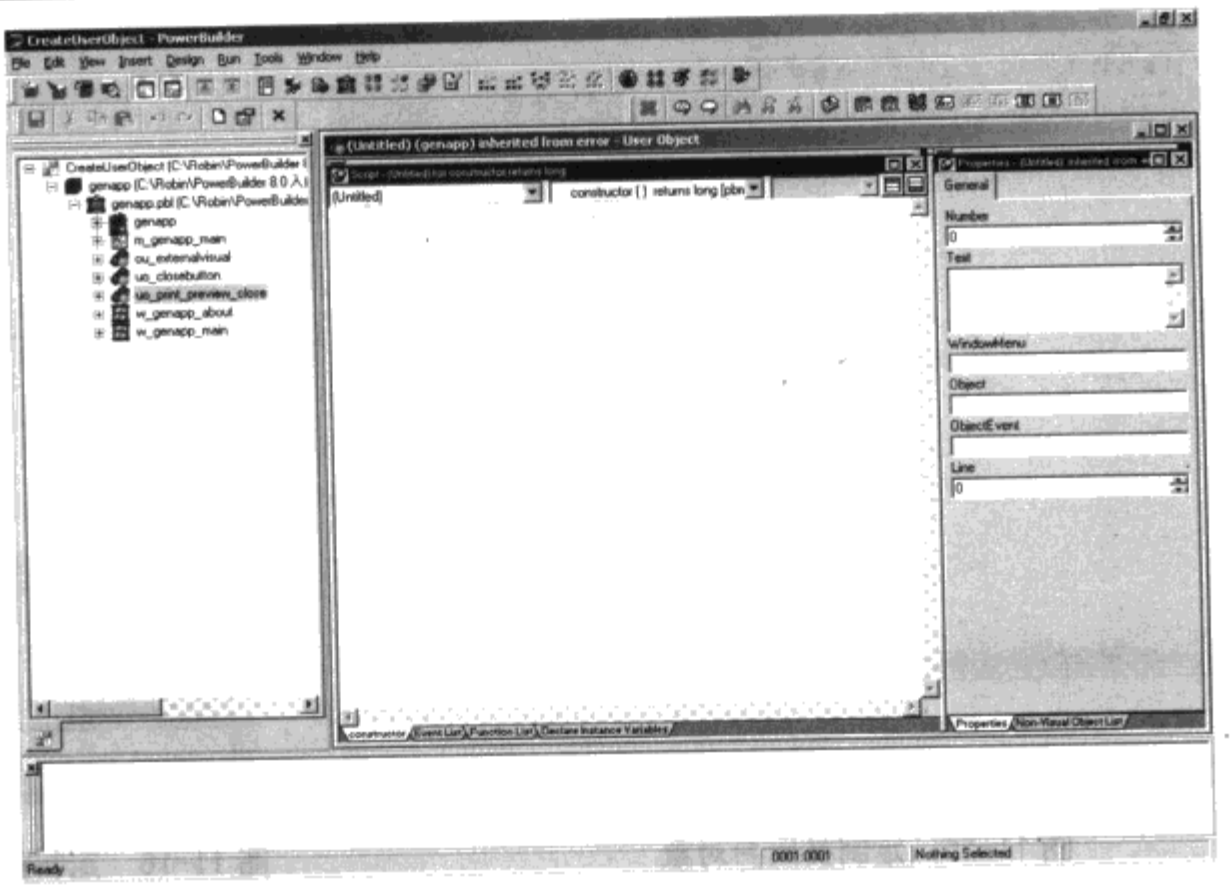


图 11-18　标准类用户对象

根据需要，通过 Script 视图为用户对象定义函数、结构或变量，以扩展标准类用户对象。

为用户对象定义需要的事件，必要时添加一些脚本处理代码。

(4)　单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-10 所示。

在 Save User Object 对话框中的 User Objects 文本框中输入要保存的名字，用户对象的默认前缀是 uo_，在 Comments 文本框中输入关于这个窗口的注释，然后单击 OK 按钮，保存用户对象并返回用户对象画板。

至此，完成了标准类用户对象的创建，该对象在窗口中不能像其他标准控件一样使用，但是可以在程序中被使用，用来扩展继承的 PowerBuilder 标准类对象。

提示： 关于可视类用户对象和类用户对象的使用方法请参阅本章第 4 节。

11.2.6　继承创建

前面详细介绍了如何通过创建向导建立不同用户对象的方法，本节介绍如何通过继承的方式创建用户对象。通过继承方式创建用户对象可以继承父用户对象的属性、结构、函数、变量、方法、事件、代码和控件，只要进行必要的修改就可以符合自己应用程序的需要，提高代码的可重用性，是一种值得推荐的用户对象创建方法。使用这种方式开发用户对象，可以快速地开发出符合需要的用户对象，大大提高软件的开发效率。

通过继承方式创建用户对象的方法为：

(1)　从菜单栏中选择 File|Inherited 命令或者单击工具栏上的 Inherited 图标，打开 Inherited form Object 对话框，如图 11-19 所示。

(2)　Inherited form Object 对话框用于选择要继承的对象，该对话框不仅可以用于用户对象的继承创建，还应用于菜单和窗口的继承创建。Target 下拉列表框用于选择要操作的

目标程序，当目标程序改变后，窗口的内容也相应发生变化，以显示当前目标程序中的内容。对于工作空间中只有一个目标程序的情况，可以不用修改 Target 下拉列表框。Libraries 列表框中列出了当前目标程序的所有 PBL 库文件，选择不同的库文件时下面的对象列表中也会相应显示当前库文件中的对象。通过改变 Object of Type 下拉列表框可以改变对象列表中显示对象的类型，可选项有 User Objects(用户对象)、Menus Objects(菜单对象)、Windows Objects(窗口对象)和 All Objects(所有对象)在对象列表中单击要继承的对象，然后单击 OK 按钮，创建用户对象，并返回用户对象画板。

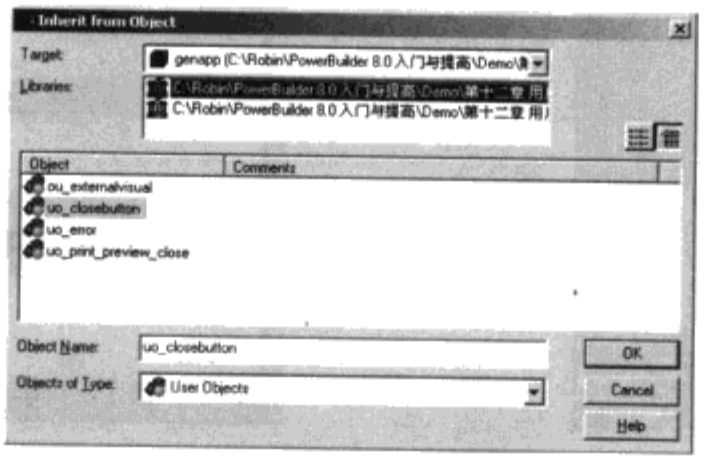


图 11-19　Inherited form Object 对话框

(3) 为继承的用户对象，添加变量和结构，定义自己的事件和编写脚本代码对该对象进行必要的扩展。

(4) 单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-10 所示。

(5) 在 Save User Object 对话框中的 User Objects 文本框中输入要保存的名字，用户对象的默认前缀是 uo_，在 Comments 文本框中输入关于这个窗口的注释，然后单击 OK 按钮，保存用户对象并返回用户对象画板。

至此，通过继承的方式创建了一个用户对象，该用户对象的可视与否跟其父用户对象一致。

11.3　定制用户对象

上一节中详细介绍了创建各种用户对象的方法，在创建的过程中提到了要对用户对象添加必要的结构、函数、变量和定义新事件等操作，本节将详细地介绍这些定制用户对象的方法。定制用户对象有如下 4 个方面：

- 插入不可视用户对象
- 插入结构
- 插入函数
- 插入事件

11.3.1　插入不可视用户对象

可以通过插入的方式在任何类型的用户对象中创建不可视的对象，插入的对象可以是定制类用户对象或者大多数的标准用户对象。插入不可视用户对象的方法如下：

(1) 打开 Insert|Object 菜单，如图 11-20 所示，该菜单列出了所有可以插入到当前用户对象中的对象，包括 PowerBuilder 中的标准类用户对象和程序中的不可视用户对象。

(2) 单击要插入到用户对象中的任何一个标准类用户对象。本例中单击 User Object 用户对象，打开 Select Object 对话框，如图 11-21 所示。

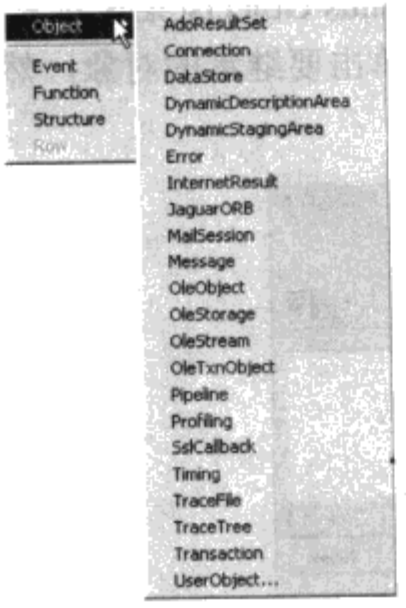


图 11-20 Insert|Object 菜单

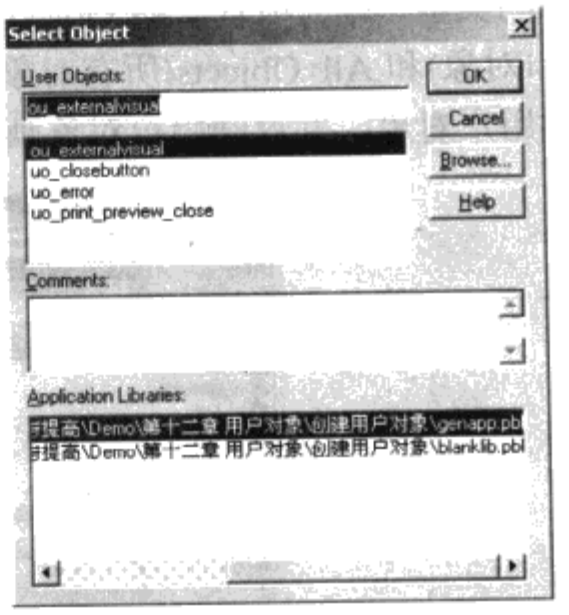


图 11-21 Select Object 对话框

(3) Select Object 对话框用于选择存在于库中的用户，以插入到当前用户对象中。Application Libraries 中列出了所有的库文件，选择不同的库文件，用户对象列表中的用户对象也随之改变。该列表框也支持选择多个库文件。如果库文件很多或者库文件中的用户对象很多，可以单击 Browse 按钮，打开如图 11-22 所示的 Browse User Objects 对话框。

(4) Browse User Objects 对话框用于在库文件中定位一个对象。在 Search For 文本框中输入要查询的字符串，查询字符串中不支持通配符。Search Libraries 列表框中列出了可以查询的库文件，该列表框支持选择多个库文件。Match Upper/Lowercase 复选框用于指定查询时是否进行大小写匹配，选择该复选框表示进行大小写匹配，否则不匹配。查询结果显示在窗口最下面的列表框中，双击该列表框中的用户对象或者选择用户对象再单击 Open 按钮都会将选中的用户对象插入到当前用户对象当中。单击 Cancel 按钮返回到 Select Object 对话框。

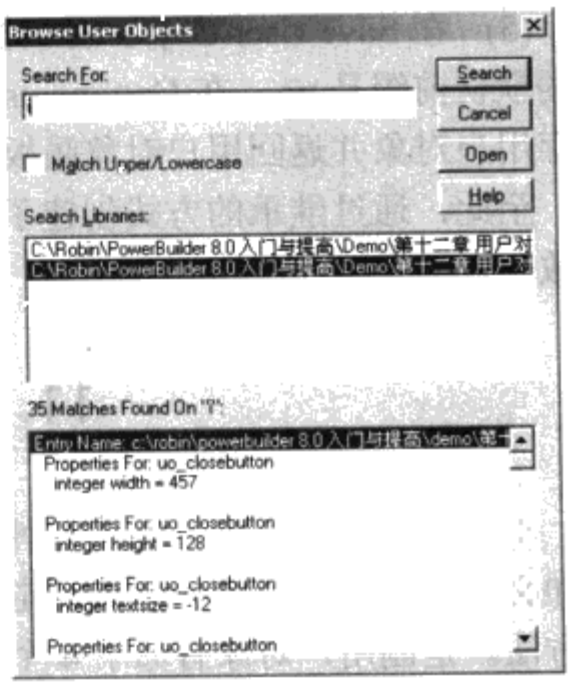


图 11-22 Browse User Objects 对话框

(5) 在 Select Object 对话框表中选择要插入的用户对象，然后单击 OK 按钮，插入用户对象，并返回用户对象画板。

注意： 虽然在 Select Object 对话框中列出了全部的用户对象，包括可视的用户对象和不可视的类用户对象，但是只有不可视的类用户对象才能够插入到当前的用户对象中。如果选择插入一个可视的用户对象，则系统会弹出如图 11-23

所示的对话框。

图 11-23　插入可视的用户对象时的错误提示

当在应用程序中创建用户对象时，也同时创建用户对象内的不可视对象，当释放用户对象时，也同时释放不可视对象。不可视控件的访问方法有两种，第一种是通过用户对象画板的 Non-Visual Object List 视图访问，如图 11-24 所示。该图中列出了当前用户对象中的所有不可视对象(如果用户对象也是不可视的类用户对象，那么也包括用户对象本身)，通过右键菜单中的命令可以剪切、复制、粘贴和删除用户对象，也可以打开 Script 视图添加代码，或者打开 Properties 视图修改用户对象的属性。另外一种方法是通过系统目录树访问用户对象，如图 11-25 所示。

图 11-24　Non-Visual Object List 视图

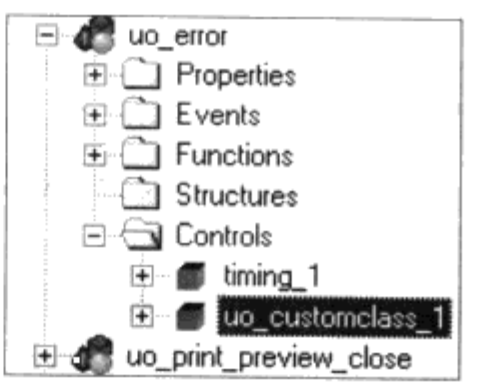


图 11-25　通过系统目录树访问用户对象

11.3.2　插入结构

选择 Insert 菜单中的 Structure 命令可以在当前用户对象中插入结构，并自动显示用户对象画板的 Structure 视图，如图 11-26 所示。

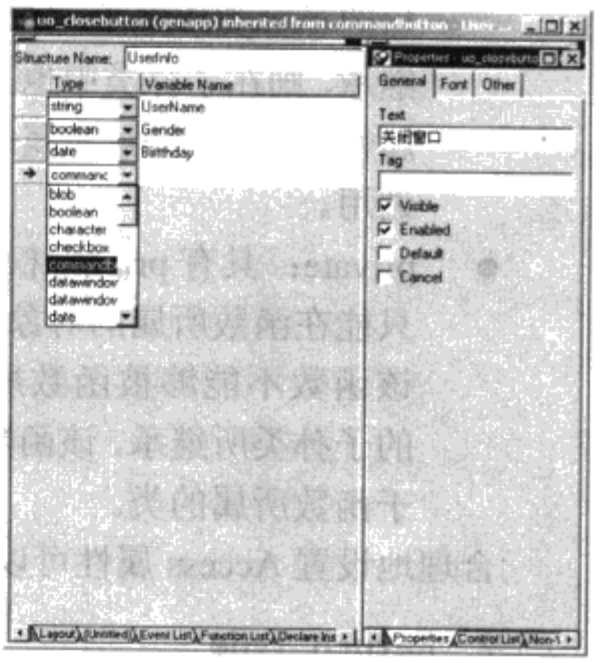


图 11-26　插入结构

在 Structure Name 文本框中输入结构名字，通过 Type 域中的下拉列表框选择合适的数据类型，能够使用的类型可以是 PowerBuilder 中的标准数据类型，例如 Integer、Boolean、Character、String、date 等；也可以是 PowerBuilder 中的对象和控件，例如 commandbutton、datawindow、dropdownpicturelistbox 等。在 Variable Name 中输入变量的名字。结构的删除、插入行和删除行等操作可以在如图 11-27 所示的 Structure 视图右键菜单中完成。

新插入的结构可以通过用户对象画板的 Structure List 视图访问，如图 11-28 所示。在 Structure List 视图中，通过右键菜单可以复制、剪切、粘贴和删除结构，也可以添加新结构。设置完结构以后，单击工具栏中的 Save 按钮，保存

新建的结构。

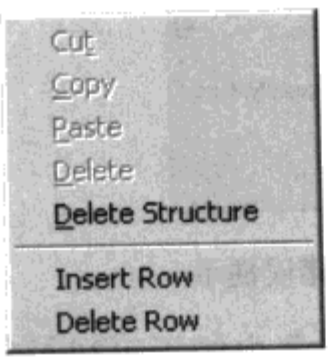


图 11-27 Structure 视图的右键菜单

图 11-28 用户对象画板的 Structure List 视图

11.3.3 插入函数

选择 Insert 菜单中的 Function 命令可以在当前用户对象中插入函数，并自动显示用户对象画板的 Script 视图，如图 11-29 所示。

在 Script 视图中可以定义新插入的函数，该视图包括许多部分，每一部分都有其特殊的意义。

1. Access

Access 下拉列表框用于设置函数的可访问权限，可以设置的取值为 public、protected 和 private。

- public：具有 public 权限的函数，可以在任何代码中被调用。该函数也能够被函数所属的对象的子孙类所继承，如果该函数被继承，则在子孙类中也是 public 属性，能够被任何的代码调用。
- protected：具有 protected 权限的函数，只能在函数所属的对象中被调用。该函数也能够被函数所属的对象的子孙类所继承，如果该函数被继承，则在子孙类中也是 protected 属性，只能在函数所属的对象中被调用。
- private：具有 private 权限的函数，只能在函数所属的对象中被调用。该函数不能够被函数所属的对象的子孙类所继承，该函数只是专属于函数所属的类。

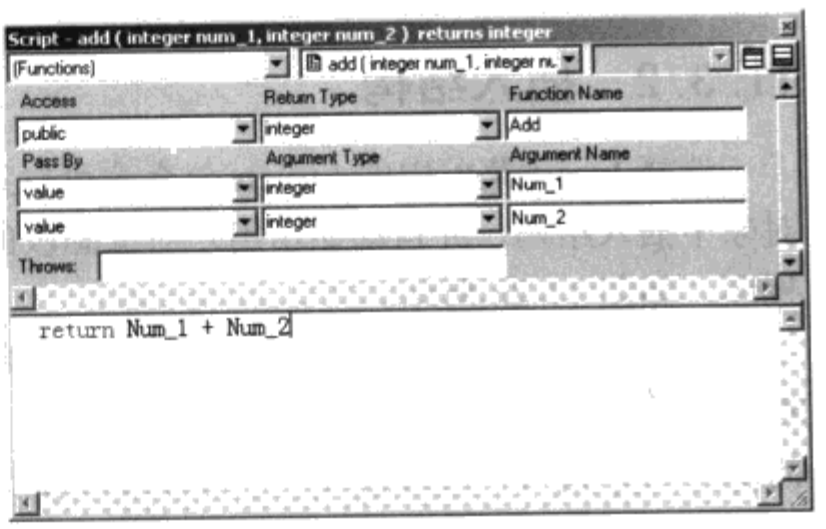


图 11-29 插入函数

合理地设置 Access 属性可以将对象正确的封装，体现了面向对象语言封装性的特点。

2. Return Type

很多函数都有返回值，返回值通常用于返回函数的处理结果或者用于指示当前程序是否成功地被执行。Return Type 下拉列表框用于指定函数的返回值，可以通过下拉列表框选择合适的返回值类型，能够使用的类型可以是 PowerBuilder 中的标准数据类型，例如：

Integer、Boolean、Character、String、Date 等；也可以是 PowerBuilder 中的对象和控件，例如：commandbutton、datawindow、dropdownpicturelistbox 等。在代码中使用 return 语句返回函数的返回值，例如 return1，表示函数返回 1。如果要指定返回值，直接从下拉列表框中选取适当的类型就可以了。从下拉列表框中选择 None，表示函数没有返回值。

3. Function Name

Function Name 文本框用于输入函数的名字，函数名字最多可以有 40 个有效字符。函数的名字是区别于其他函数的重要标志，在调用函数的时候也是调用的函数的名字，因此一定要能代表函数的功能。对于对象级的函数，当焦点离开 Function Name 文本框时，函数的名字就被添加到了对象的 Function List 视图中，当保存对象时，函数也作为对象的一部分被保存。

4. Pass By

Pass By 下拉列表框用于指定函数参数的传递类型，可以取值为 value、reference 和 readonly，其含义分别如下：

- value：参数是值传递形式。在定义函数时函数名后面括号中的变量名称为“形式参数”(简称“形参”)，在脚本代码中调用函数的时候，函数名后面的表达式被称为“实际参数”(简称“实参”)。当设置值传递后，实参对形参的值传递是单向的，只能实参将值传递给形参，形参不能将值传回来。这种情况下实际上是实参将值复制到形参的地址空间，形参和实参不互相干扰，它们之间的关系如图 11-30 所示。
- reference：引用传递形式。在引用传递中形参直接引用实参的值，实参和形参共用一个地址空间，因此值的传递方向是双向的，当形参改变时实参也随之改变。在引用传递形式中实参和形参的关系，如图 11-31 所示。

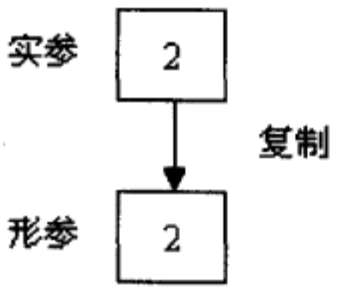


图 11-30　值传递中实参和形参的关系

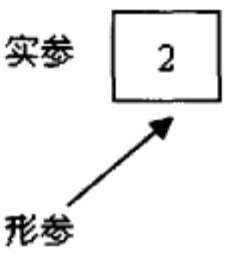


图 11-31　引用传递中实参和形参的关系

- readonly：只读形式。表示形参值是只读的形式，在程序中可以将形参值作为常数使用。

5. Argument Type

Argument Type 下拉列表框用于选择参数的数据类型，可以是 PowerBuilder 中的标准数据类型，例如 Integer、Boolean、Character、String 和 Date 等；也可以是 PowerBuilder 中的对象和控件，例如 commandbutton、datawindow、dropdownpicturelistbox 等。

6. Argument Name

Argument Name 下拉列表框用于选择参数的名字，参数的名字不应当重复，这个参数的名字在程序中可以被直接使用。默认情况下，Prototype 中只有一个参数可以输入，但是有时需要定义多个参数，因此需要再增加函数的参数。增加函数参数的方法是在 Prototype 上单击鼠标右键，在如图 11-32 所示的弹出菜单中单击 Add Parameter 或 Insert Parameter 命令，增加另一个参数。单击 Delete Parameter 命令可以删除不需要的函数参数。

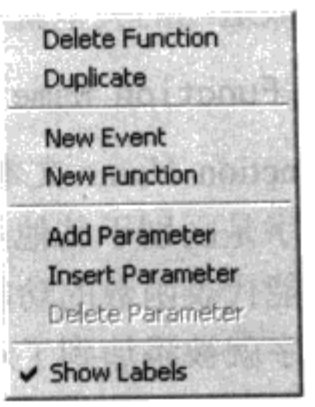


图 11-32 Prototype 的弹出菜单

7. 脚本编写

用户对象画板 Script 视图下面的空白处用于添加函数的脚本代码，如果函数定义了返回值，则使用 return 语句返回值，return 语句的语法格式为：

```
RETURN expression
```

其中 expression 是函数的返回值，返回值的数据类型应当和定义的函数返回值类型一致。例如：定义两个参数 Num_1 和 Num_2，函数中首先判断 Num_2 是否为 0，如果不为 0，则让 Num_1 除以 Num_2 并返回计算结果，其代码如下：

```
if Num_2<> 0 then
RETURN Num_1 / Num_2
else
RETURN -1//如果除数为 0，则返回-1
end if
```

11.3.4 插入事件

选择 Insert 菜单中的 Event 命令可以在当前用户对象中插入事件，并自动显示用户对象画板的 Script 视图，如图 11-33 所示。

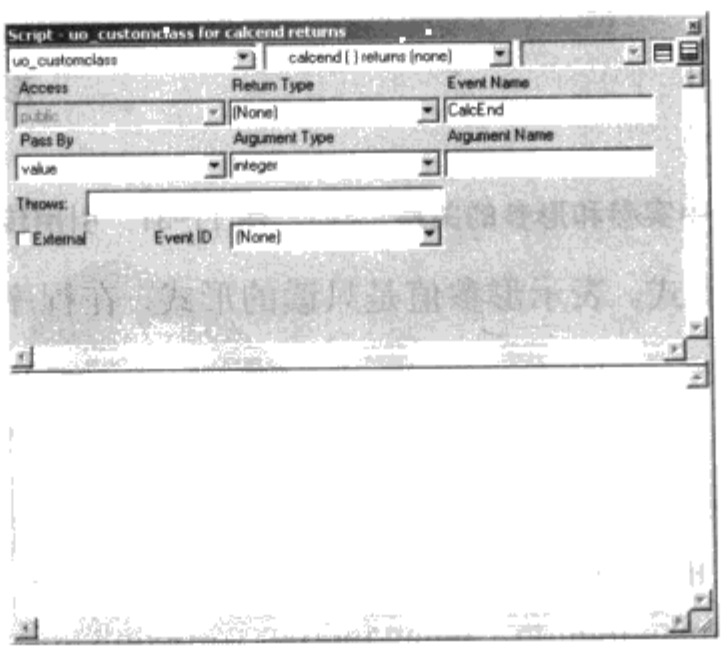


图 11-33 插入事件

事件的定义和函数的定义有些相似，Event Name 文本框用于输入事件的名字；在 Event ID 下拉列表框中列出了可以选择的事件编号，用户可以选择系统的事件编号也可以使用自定义事件编号(从 pbm_custom01 到 pbm_custom75)。

提示： 在输入完事件名后，并且将焦点从 Event Name 文本框转移走后，就可以在 Event List 视图中看到新增加的事件。当保存用户对象时，新插入的事件将作为用户对象的一部分被保存。

11.4　使用用户对象

本节将在前几节的基础上介绍用户对象的使用方法。

11.4.1　使用可视用户对象

可视用户对象只有通过用户对象控件才能最终在窗口中显示，用户对象控件和用户对象之间的关系就像数据窗口控件和数据窗口对象之间的关系。

可视用户对象可以像其他窗口控件一样在窗口中使用，使用可视用户对象的一般方法如下：

(1) 创建标准可视用户对象、定制可视用户对象或者外部可视用户对象，并修改属性、添加事件、变量、函数和脚本代码，以扩展用户对象。

(2) 打开窗口，在窗口中添加用户对象控件，并为该控件选择合适的用户对象。

(3) 设置用户对象控件的属性，为用户对象控件添加必要的脚本代码。

(4) 在程序中通过脚本动态修改用户对象控件的属性。

下面创建一个标准可视用户对象，并在窗口中使用它，步骤如下：

(1) 单击工具栏上的 New 按钮，打开 New 对话框，选择 PB Object 选项卡，单击 Standard Visual 图标创建标准可视用户对象，然后单击 OK 按钮，打开 Select Standard Visual Type 对话框，如图 11-34 所示。

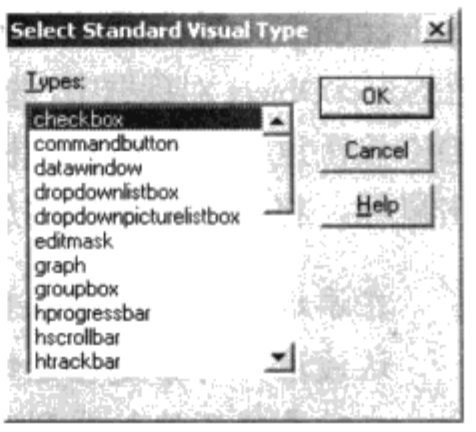


图 11-34　Select Standard Visual Type 对话框

(2) 在 Select Standard Visual Type 对话框中选择 commandbutton 项，然后单击 OK 按钮继承该命令按钮，打开用户对象画板。在用户对象画板的 Properties 视图中设置 Text 属性为“关闭窗口”，如图 11-35 所示。

(3) 在用户对象画板 Script 视图中添加用户对象的 clicked 事件脚本，代码如下：

```
int retval
//显示对话框，询问用户是否退出
retval = MessageBox("退出窗口提示","确认要关闭当前窗口吗？",Question!,YesNo!)
//判断用户的操作
if retval = 1 then
```

```
Close(Parent);  //当用户选择 Yes 时关闭当前窗口
end if
```

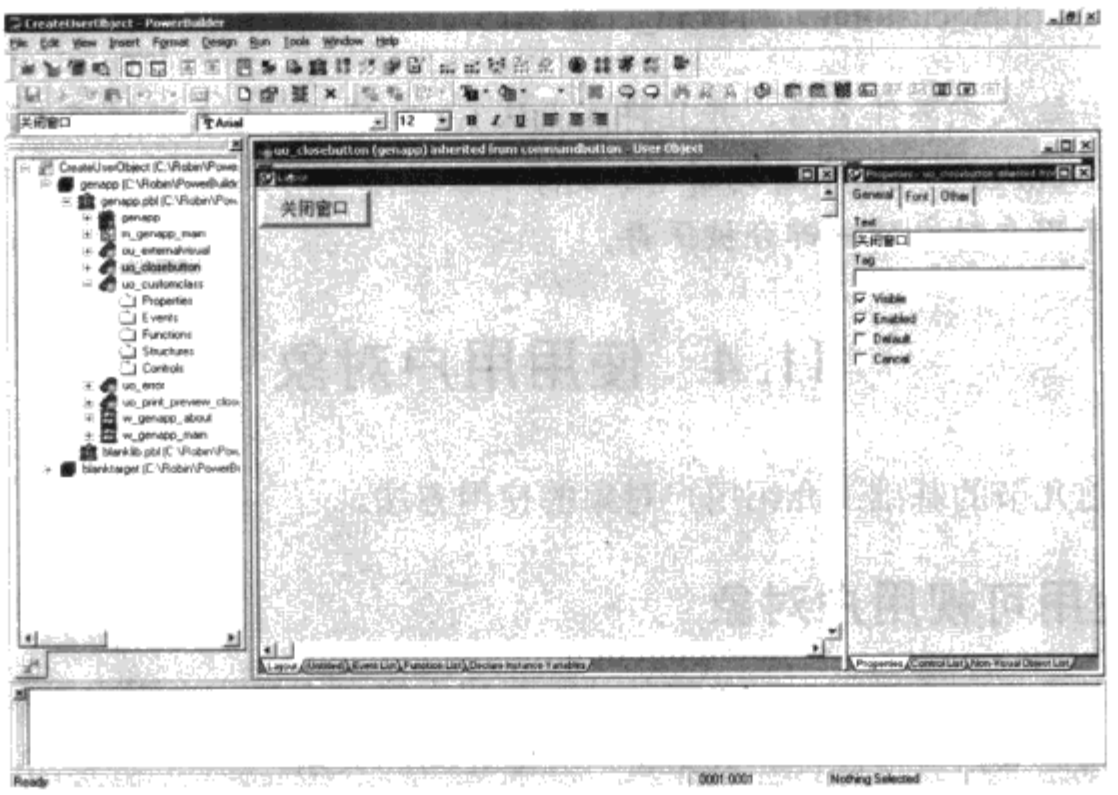


图 11-35　用户对象画板

(4) 单击工具栏上的 Save 按钮，在 Save User Object 对话框中将新建立的标准可视用户对象命名为 uo_closebutton，并返回用户对象画板。

(5) 新建一个窗口，将其命名为 w_genapp_main，设置窗口 Title 属性为“标准可视用户对象演示”。

(6) 从窗口画板工具栏的控件列表复合按钮中选择 User Object 控件，为窗口增加一个用户对象控件，打开 Select Object 对话框，如图 11-36 所示。

(7) 在 Select Object 对话框中，选择 uo_closebutton，然后单击 OK 按钮返回窗口对象画板。在窗口中希望的位置单击鼠标，将用户对象控件放置在该位置。

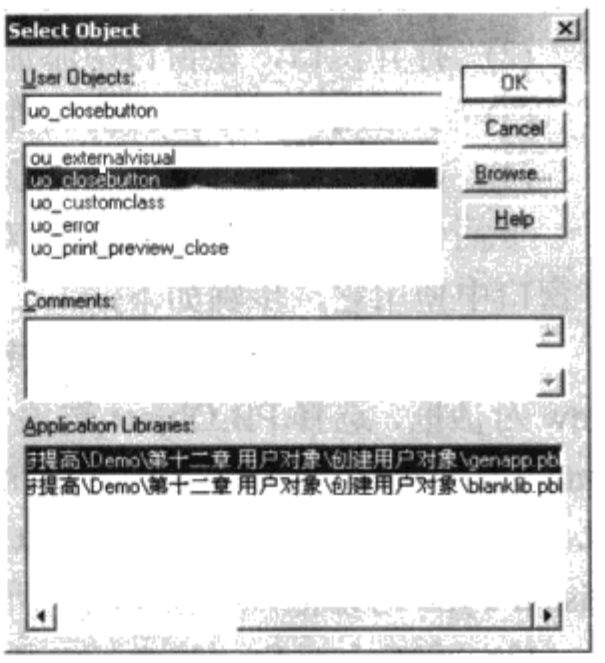


图 11-36　Select Object 对话框

注意： 在添加用户对象 uo_closebutton 前请确认该用户对象是否已经关闭，否则该用户对象将不能加载到窗口中，并会显示如图 11-37 所示的提示信息。

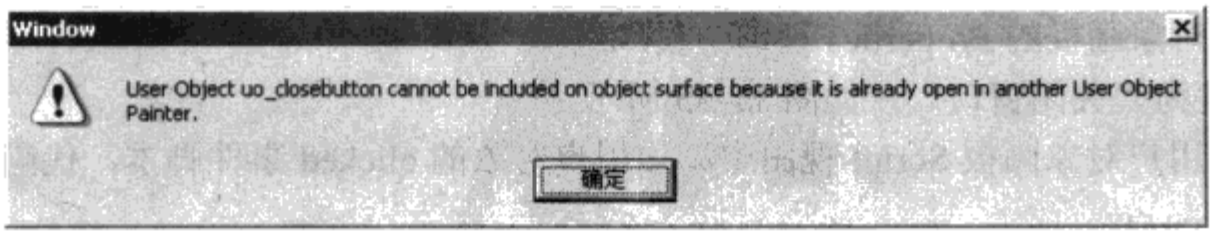


图 11-37　提示信息

(8)　单击工具栏上的 Run 按钮运行程序，程序运行界面如图 11-38 所示。

(9)　单击程序窗口中央的命令按钮(标准可视用户对象)，显示如图 11-39 所示的对话框，提示用户是否要关闭窗口。单击“否”按钮，程序不做任何事情；单击“是”按钮窗口被关闭。

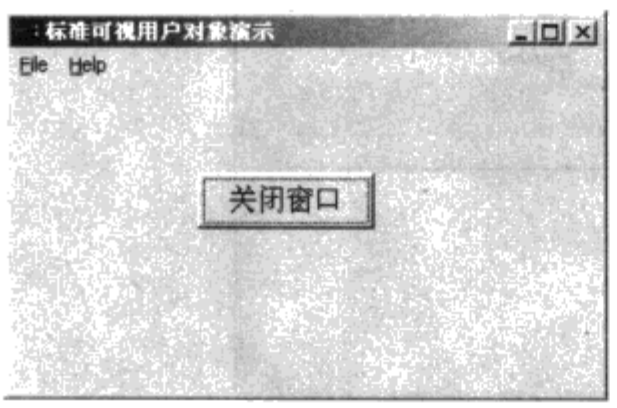


图 11-38　程序运行界面 1

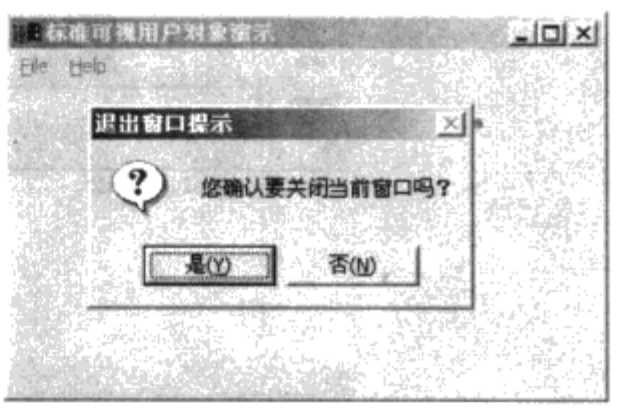


图 11-39　程序运行界面 2

11.4.2　使用类用户对象

在 PowerBuilder 中将经常使用的一些功能进行封装，将其定制为类用户对象，以便在使用的时候直接在程序中调用。如果需要对 PowerBuilder 提供的标准对象进行扩展就需要使用标准类用户对象了。类用户对象不能在窗口中使用，在代码中使用的时候首先要先定义一个类用户对象类型的变量，然后用 Create 语句为该变量分配一块内存空间，创建类用户对象的一个实例，这样在类用户对象中的函数、属性等就可以被调用了。在使用完类用户对象后要调用 Destroy 语句将类用户对象变量删除并回收内存空间。如果忘记使用 Destroy 语句回收空间，那么就会造成内存的浪费，降低整体使用性能。类用户对象的使用方法如下：

(1)　创建一个类用户对象，并扩展该用户对象。

(2)　定义类用户对象变量。

(3)　使用的时候调用 Create 语句，创建类用户对象的实例。

(4)　使用该类用户对象变量，引用类用户对象中函数、属性等。

(5)　使用结束，调用 Destroy 语句删除类用户对象变量，回收内存空间。

下面创建一个定制类用户对象，并为这个用户对象添加 4 个函数，用于计算两个数值的和、差、乘积和商，然后在窗口中调用这个定制类用户对象实现一个简单的计算器。具体步骤如下：

(1)　新建一个定制类用户对象，并为其插入一个函数，如图 11-40 所示。

(2)　将函数命名为 Add，返回值为 Double 类型，第 1 个参数类型为 Double，参数名字为 num1，第 2 个参数的类型为 Double，参数名字为 num2，函数代码如下：

```
//求和运算
return num1 + num2
```

(3)　增加第 2 个函数，函数命名为 Sub，返回值为 Double 类型，第 1 个参数类型为 Double，参数名字为 num1，第 2 个参数的类型为 Double，参数名字为 num2，函数的代码如下：

```
//求差运算
return num1 - num2
```

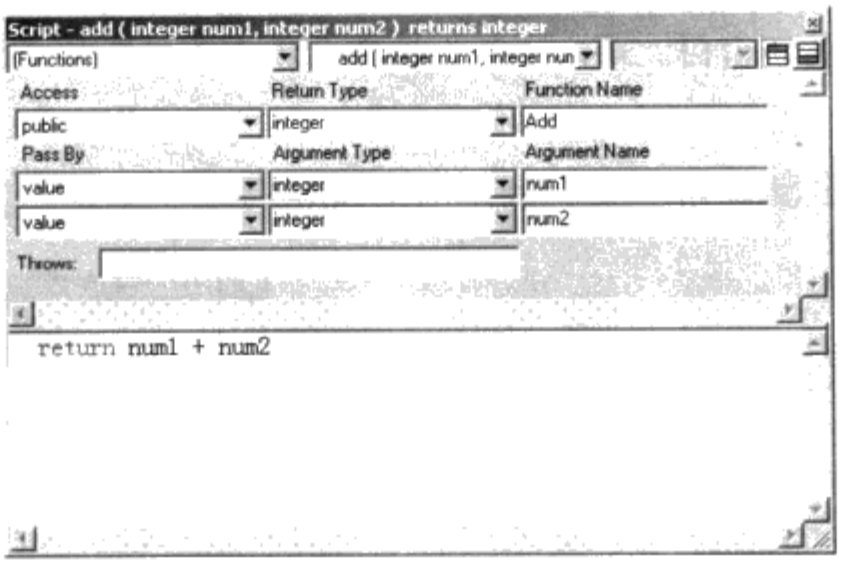


图 11-40　插入 Add 函数

(4) 为对象增加第 3 个函数，函数命名为 Multiply，返回值为 Double 类型，第 1 个参数类型为 Double，参数名字为 num1，第 2 个参数的类型为 Double，参数名字为 num2，函数的代码如下：

```
//计算乘积
return num1 * num2
```

(5) 为对象增加第 4 个函数，函数命名为 Divide，返回值为 Double 类型，第 1 个参数类型为 Double，参数名字为 num1，第 2 个参数的类型为 Double，参数名字为 num2，函数的代码如下：

```
//除法运算
if num2<> 0 then
return num1 / num2
else
return -1//如果除数为 0，则返回-1
end if
```

(6) 单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-41 所示。

(7) 将新建的定制类用户对象命名为 uo_calc，在 Comments 框中输入“定制类用户对象用于执行计算器功能”，然后单击 OK 按钮保存对象，并返回用户对象画板。

(8) 新建一个窗口将其命名为 w_calculatorapp_main，设置其 Title 属性为“简单的计算器”。在窗口中增加如表 11-1 所示控件，控件的布局如图 11-42 所示。

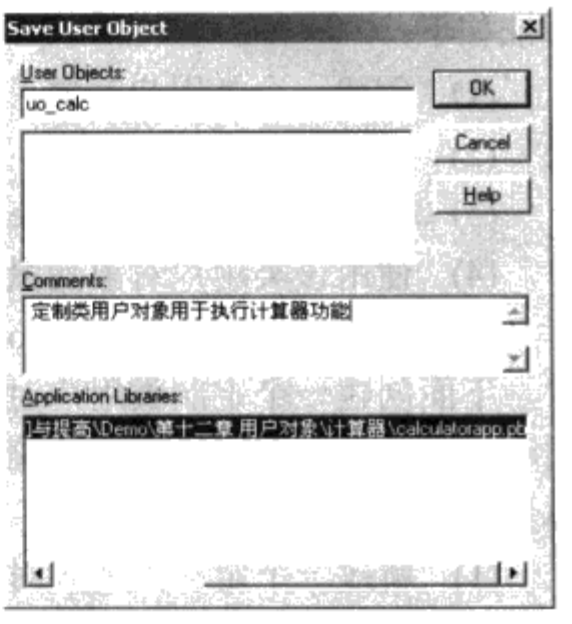


图 11-41　Save User Object 对话框

表 11-1　窗口中的控件

控件类型	属性	属　性　值
静态文本框	Text	第一个数
静态文本框	Text	第二个数

续表

控件类型	属性	属　性　值
静态文本框	Text	计算结果
单行编辑框	Name	sle_num1
	Text	空
单行编辑框	Name	sle_num2
	Text	空
单行编辑框	Name	sle_result
	Text	空
命令按钮	Name	cb_add
	Text	+
命令按钮	Name	ecb_sub
	Text	—
命令按钮	Name	cb_multiply
	Text	×
命令按钮	Name	cb_divide
	Text	/

(9) 在窗口对象画板的 Declare Instance Variables 视图中声明定制类用户对象类型的变量，代码如下：

```
//声明定制类用户对象变量
uo_calc cal
```

(10) 在窗口对象画板的 Script 视图中为窗口的 Open 事件添加事件处理代码，用于创建定制类用户对象实例，代码如下：

```
//创建定制类用户对象实例
cal = Create uo_calc
```

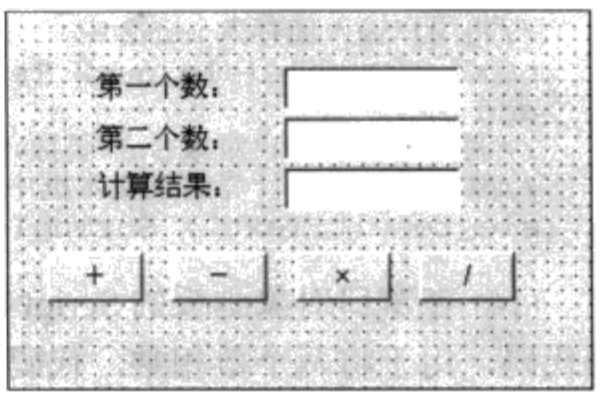


图 11-42　控件布局

(11) 在窗口对象画板的 Script 视图中为窗口的 close 事件添加事件处理代码，用于删除定制类用户对象实例，回收内存空间，代码如下：

```
//删除定制类用户对象变量，回收内存
Destroy cal
```

(12) 在窗口对象画板的 Script 视图中为命令按钮 cb_add 的 clicked 事件添加事件处理代码，用于将两个数相加，代码如下：

```
//定义变量用户保存输入值
Double num1
Double num2
if sle_num1.Text ="" then
//如果输入值为空则认为 num1=0
num1 = 0;
else
//将用户输入值存入 num1 中
num1 = Double(sle_num1.Text)
```

```
end if
if sle_num2.Text ="" then
//如果输入值为空则认为 num2=0
num2 = 0;
else
//将用户输入值存入 num2 中
num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Add 函数，计算两个数的和
//将结果显示在计算结果文本框中
sle_result.Text = String(cal.Add(num1,num2))
```

(13) 在窗口对象画板的 Script 视图中为命令按钮 cb_sub 的 clicked 事件添加事件处理代码，用于将两个数相减，代码如下：

```
//定义变量用户保存输入值
Double num1
Double num2
if sle_num1.Text ="" then
//如果输入值为空则认为 num1=0
num1 = 0;
else
//将用户输入值存入 num1 中
num1 = Double(sle_num1.Text)
end if
if sle_num2.Text ="" then
//如果输入值为空则认为 num2=0
num2 = 0;
else
//将用户输入值存入 num2 中
num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Sub 函数，计算两个数的差
//将结果显示在计算结果文本框中
sle_result.Text = String(cal.Sub(num1,num2))
```

(14) 在窗口对象画板的 Script 视图中为命令按钮 cb_multiply 的 clicked 事件添加事件处理代码，用于将两个数相乘，代码如下：

```
//定义变量用户保存输入值
Double num1
Double num2
if sle_num1.Text ="" then
//如果输入值为空则认为 num1=0
num1 = 0;
else
//将用户输入值存入 num1 中
num1 = Double(sle_num1.Text)
end if
if sle_num2.Text ="" then
//如果输入值为空则认为 num2=0
num2 = 0;
else
//将用户输入值存入 num2 中
num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Multiply 函数，计算两个数的乘积
//将结果显示在计算结果文本框中
sle_result.Text = String(cal.Multiply(num1,num2))
```

(15) 在窗口对象画板的 Script 视图中为命令按钮 cb_divide 的 clicked 事件添加事件处

理代码，用于将两个数相除，代码如下：

```
//定义变量用户保存输入值
Double num1
Double num2
if sle_num1.Text ="" then
//如果输入值为空则认为 num1=0
num1 = 0;
else
//将用户输入值存入 num1 中
num1 = Double(sle_num1.Text)
end if
if sle_num2.Text ="" then
//如果输入值为空则认为 num2=0
num2 = 0;
else
//将用户输入值存入 num2 中
num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Divide 函数，计算两个数的商
//将结果显示在计算结果文本框中
sle_result.Text =
String(cal.Divide(num1,num2))
```

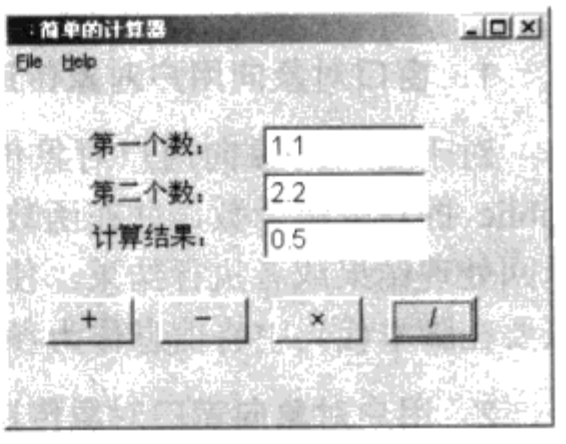


图 11-43　计算器程序运行界面

(16) 单击工具栏上的 Run 按钮，程序运行界面如图 11-43 所示。

11.5　窗口与用户对象间的通信

在实际的应用中经常需要在窗口对象和可视的用户对象间交换信息，例如：可视对象中有一组命令按钮，这组命令按钮根据一个窗口对象上的单行编辑框中的内容执行相应操作，因此需要将单行文本框中的内容从窗口对象传递给用户对象；或者有一个定制可视用户对象，该对象包含一个选择颜色的调色板，当用户单击了调色板中的某个颜色值后，需要将颜色值从用户对象传递给窗口对象，以便将窗口对象中的控件改变为相应的颜色。窗口对象和用户对象的通信方式有如下 3 种：

- 直接引用用户对象属性。
- 使用函数。
- 使用用户事件。

11.5.1　直接引用用户对象属性

直接引用用户对象的属性是窗口对象和用户对象之间最直接的通信方式。直接引用用户对象属性的语法定义为：

```
userobjcontrol.objectname.property = newvalue
```

其中，userobjcontrol 是用户对象控件的名字，objectname 是用户对象中其他对象的名字，property 是用户对象包含的其他对象的属性，newvalue 是新设置的属性值。

例如：在用户对象控件 uo_edit 中放置了一个定制可视用户对象，该对象中包含了一个单行编辑框 sle_input 和一个静态文本框 st_inputhint，在窗口对象的脚本代码中可以给用户

对象中单行编辑框和静态文本框的显示文本赋值：

```
uo_edit.st_inputhint.text="请输入用户名"
uo_edit.st_sle_inpu=" "
```

11.5.2 使用函数

使用函数进行窗口对象和用户对象间的通信分为两种情况：

- 窗口对象向用户对象传递信息。
- 用户对象向窗口对象传递信息。

1. 窗口对象向用户对象传递信息

对于窗口对象向用户对象传递信息的情况，需要在用户对象中定义一个访问权限为 public 的对象级函数，并在函数中定义要接收窗口对象信息的参数，函数可以有返回值，返回处理结果或者执行结果。使用时将创建的用户对象放在窗口中，如果需要传递信息，只要在脚本代码中将信息作为参数调用函数即可。

2. 用户对象向窗口对象传递信息

对于用户对象向窗口对象传递信息的情况，需要在窗口对象中定义一个访问权限为 public 的窗口级函数，并在函数中定义要接收用户对象信息的参数，如果需要函数可以有返回值，返回处理结果或者执行结果。使用时将创建的用户对象放在窗口中，如果需要传递信息，只要在脚本代码中将信息作为调用函数的参数即可。

下面以一个简单的“最大化”、“最小化”窗体为例来介绍在利用函数进行窗口对象和用户对象通信的步骤。

(1) 新建一个定制可视用户对象，在该对象上添加两个命令按钮，一个命名为 cb_max，另一个命名为 cb_min。将命令按钮 cb_max 的 Text 属性设置为“最大化”；将命令按钮 cb_min 的 Text 属性设置为“最小化”。

(2) 在 Declare Instance Variables 视图中增加一个窗口类型的实例变量，代码如下：

```
//定义变量，保存当前窗口
window CurrentWindow
```

(3) 在窗口对象画板的 Script 视图中为命令按钮 cb_max 的 clicked 事件添加事件处理代码，用于将窗口最大化，代码如下：

```
//最大化窗口
CurrentWindow.WindowState = Maximized!
```

(4) 在窗口对象画板的 Script 视图中为命令按钮 cb_min 的 clicked 事件添加事件处理代码，用于将窗口最小化，代码如下：

```
//最小化窗口
CurrentWindow.WindowState = Minimized!
```

(5) 打开用户对象画板的 Function List 视图，在该视图上右击，在弹出的快捷菜单中选择 Add 命令，增加一个函数，函数命名为 SetWindow，无返回值，参数名字为 ParentWindow，类型为 Window，如图 11-44 所示。添加该函数的脚本代码如下：

```
//保存当前窗口
CurrentWindow = ParentWindow
```

(6) 单击工具栏上的 Save 按钮，保存定制可视用户对象，将其命名为 uo_maxminwindow _by_function。

(7) 关闭用户对象画板，新建一个窗体，将其命名为“w_maxminwindow_by_function”，设置其 Title 属性为“通过函数的方式进行通信”。

(8) 在窗口中添加用户对象控件，选择显示 uo_maxminwindow_by_function 用户对象，将其命名为 uo_maxmin。

(9) 在窗口的 Open 事件中增加如下代码，用于将当前窗口传递给用户对象：

```
//保存当前窗口
CurrentWindow = ParentWindow
```

(10) 保存窗口，并运行程序，界面如图 11-45 所示。

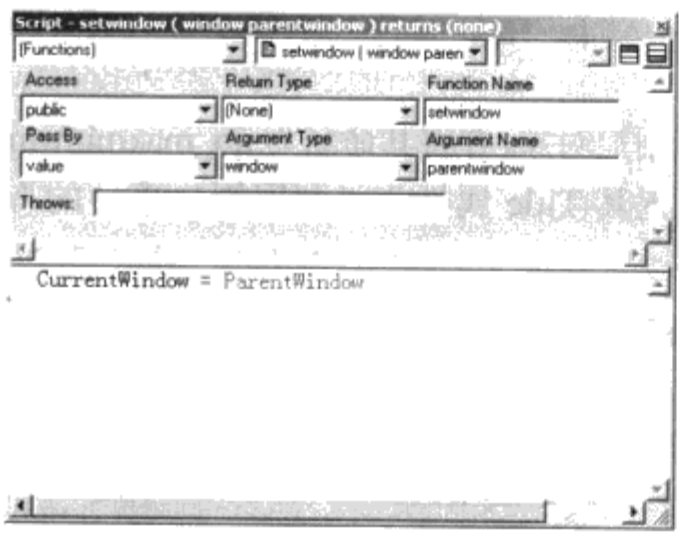


图 11-44　SetWindow 函数

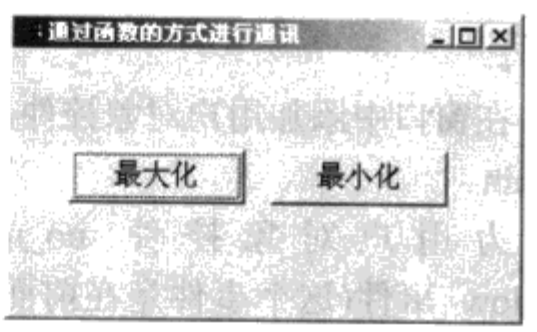


图 11-45　通过函数的方式进行通讯程序运行界面

11.5.3　使用用户事件

除了使用函数的方式和窗口对象进行通信外，还可以使用用户事件来进行通信，下面通过一个例子具体介绍通过用户事件与窗口对象通信的方法。

(1) 新建一个定制可视用户对象，在该对象上添加两个命令按钮，一个命名为 cb_max，另一个命名为 cb_min。将命令按钮 cb_max 的 Text 属性设置为“最大化”，将命令按钮 cb_min 的 Text 属性设置为“最小化”。

(2) 在 Event List 视图中右击，在弹出的快捷菜单中选择 Add 命令，增加一个事件，命名为 maxwindow，如图 11-46 所示。

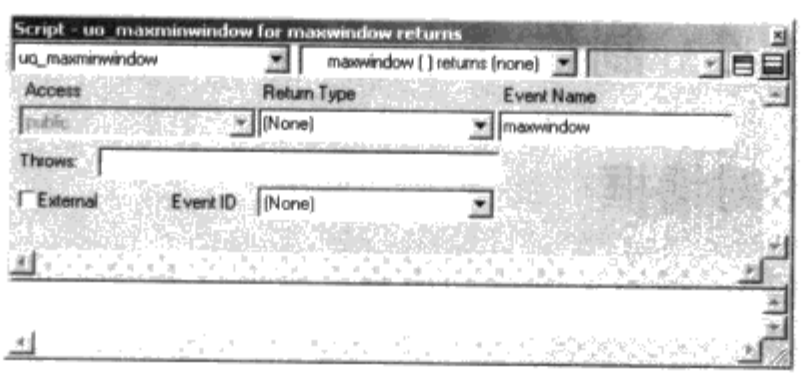


图 11-46　新建 maxwindow 事件

(3) 按照上一步的方法增加另一个事件 minwindow。

(4) 在窗口对象画板的 Script 视图中为命令按钮 cb_max 的 clicked 事件添加事件处理代码，用于将窗口最大化，代码如下：

```
//触发 MaxWindow 事件
Parent.TriggerEvent("MaxWindow")
//Parent.Event maxwindow( )
```

注意： 使用 Parent.TriggerEvent("MaxWindow")和 Parent.Event maxwindow( )的作用相同，都是用于触发 MaxWindow 事件。

(5) 在窗口对象画板的 Script 视图中为命令按钮 cb_min 的 clicked 事件添加事件处理代码，用于将窗口最小化，代码如下：

```
//触发 MinWindow 事件
Parent.TriggerEvent("MinWindow")
//Parent.Event
```

(6) 单击工具栏上的 Save 按钮，保存定制可视用户对象，并将其命名为 uo_maxminwindow。

(7) 关闭用户对象画板，新建一个窗体，设置其 Title 属性为“利用用户事件的方式进行通信”。

(8) 在窗口中添加用户对象控件，选择显示 uo_maxminwindow 用户对象，将其命名为 uo_maxmin。

(9) 为用户对象控件 uo_maxmin 的 MaxWindow 事件(这个事件是在前面定义的)添加处理代码，用于最大化窗口：

```
//将窗口最大化
Parent.WindowState = Maximized!
```

(10) 为用户对象控件 uo_maxmin 的 MinWindow 事件(这个事件是在前面定义的)添加处理代码，用于最小化窗口：

```
//将窗口最小化
Parent.WindowState = Minimized!
```

(11) 保存窗口，并运行程序，界面如图 11-47 所示。

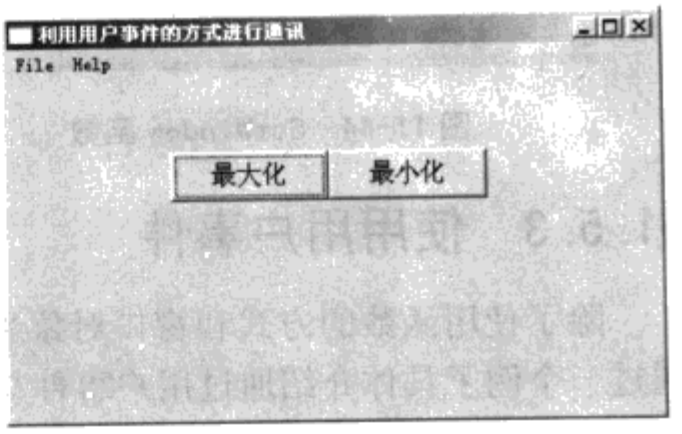


图 11-47 利用用户事件的方式进行通讯程序运行界面

11.6 制作一个通用对话框实例

11.6.1 制作通用对话框

本例使用用户对象通过调用一些规则函数并依靠主窗体功能代码制作一个通用对话框，在此窗口中用户实现一般通用对话框的所有基本功能。

本例主要使用用户对象 User Object 中的不可视用户对象(Non Visual Object-User Object)

来实现通用对话框的各项功能——“打开对话框”、“保存对话框”、“打印对话框”、“字体设置”等功能。其中与对话框和打印设置有关的各个函数都起了重要作用，本例将有选择、有重点地介绍它们。

创建一个通用对话框的窗口，主要通过 PowerBuilder 10.0 提供的 User Object 控件调用编写的各个和对话框、打印设置有关的函数来实现。目的是使用户在此窗口中实现通用对话框的所有基本的功能。为了完成整个功能，需要建立 Workspace、Target 和 Application 以及一个数据窗口和一个窗体。

1. 建立应用

在 PowerBuilder 10.0 中先建立 Workspace、Target 和 Application。本例无须和数据库连接，因此无需设置 ODBC 数据源，只需在 Application 中打开主窗口 w_main 即可。Application 的具体脚本如下：

```
//打开主窗体 w_ main
Open( w_ main )
```

2. 建立函数

为整个程序创建一个函数 bitor()，返回类型为 long，执行一个与二进制计算的功能，具体脚本如下：

```
// Performs a bitwise or between 2 =>long<= values...
Integer li_Bit, li_Size, li_Number = 2
long ll_RetValue, ll_ValueA, ll_ValueB

li_Size = UpperBound(Values)
If li_Size < 2 Then
    MessageBox("error","This function needs 2 values!",StopSign!,Ok!)
    Return(-1)
End If

ll_RetValue = 0
ll_ValueA = Values[1]

Do
    ll_ValueB = Values[li_Number]

    For li_Bit = 0 To 31
        If Mod(Long(ll_ValueA /  2^li_Bit), 2) > 0 Or Mod(Long(ll_ValueB /
2^li_Bit), 2) > 0 Then
            If Not Mod(Long(ll_RetValue /  2^li_Bit), 2) > 0 then
                ll_RetValue += 2^li_Bit
            End If
        End If
    Next

    ll_ValueA = ll_RetValue
    li_Number++
Loop Until li_Number > li_Size

Return(ll_RetValue)
```

3. 建立客户类

在建立窗体之前，还需要建立一个客户类(Custom Class)，即一个非视图用户对象——

nvo_commdlg，具体步骤如下：

(1) 在 PowerBuilder 的工具栏中单击 New 按钮，在弹出的对话框中选择 PB Object 选项卡，在其中选择 Custom Class(客户类)，单击 OK 按钮。

(2) 然后 PowerBuilder 会打开该客户类的工作画板。选中属性中的 AutoInstantiate 复选框，并保存该客户类。

在编写具体程序之前，要先声明变量和函数，变量声明又包括局部声明和实例声明。部分声明如下：

(1) 局部声明

```
//普通对话框外部函数声明
Function long GetOpenFileNameA(REF OPENFILENAME OpenFileName) library
"comdlg32.dll"
Function long …

//内存函数声明
Function long GetDevMode(REF DEVMODE Destination, long Source, long Size)
library "kernel32.dll" Alias For "RtlMoveMemory"
Function long …

// Misc 函数声明
Function long GetWindowsDirectoryA(REF string Buffer, long Size) library
"kernel32.dll"
```

(2) 实例声明

```
// 所有对话框声明
PUBLIC long Flags = 0                          // Dialog Flags
PUBLIC long hWndParent = 0                     // Parent Window

// Open 和 Save 对话框全局成员声明
Public:
string Filter = ""
Integer FilterIndex = 1                        // 默认的 filter 索引
string InitialDir  = ""                        // 默认目录
string Title = ""                              // 对话框标题
PROTECTEDWRITE string Filename = ""            // 返回选定的文件
PROTECTEDWRITE string Files[ ]                 // 选定文件
PROTECTEDWRITE string PathName                 // 路径
PROTECTEDWRITE Integer FileCount               // 选定文件的个数
// 标识符常量声明
CONSTANT long OFN_READONLY = 1
CONSTANT long …

// 打印对话框全局成员声明
Integer FromPage    = 1
Integer ToPage  = 1
Integer MinPage     = 1
Integer MaxPage     = 0
PROTECTEDWRITE Integer Copies = 1
// 标识符常量声明
CONSTANT long PD_ALLPAGES= 0
CONSTANT long …

// 字体对话框全局成员变量声明
long RGBColors  = 0
Integer FontType    = 0
Integer …
```

```
// 标识符常量声明
CONSTANT long CF_SCREENFONTS = 1
CONSTANT long …
// 字体类型常量声明
CONSTANT Integer BOLD_FONTTYPE = 256
CONSTANT Integer …
// 对象局部常量声明
Private: CONSTANT Integer LMEM_ZEROINIT = 64 // Alloc constant => Zero
allocated memort
// 打开对话框常量
CONSTANT Integer MAXFILENAME=260
CONSTANT Integer …
```

下面以“保存对话框”(savedialog())、“打印对话框”(printdialog())、setprinter()(打印机设置)和 fontdialog()(字体设置对话框)的函数为例说明这些函数的编写与功能。这是整个程序的核心，对这个实例的实现很重要。

(3) savedialog()函数的脚本

```
/*这个函数将使得“保存”对话框能够显示在 COMDLG32 库中，可用的成员变量为：
string Filter InitialDir, Title, Filename, Files[],PathName;Integer
FilterIndex, FileCount ;long Flags*/

OPENFILENAME SaveFileName
string ls_Token, Filters[]
Integer li_TabPos, li_Start

SaveFileName.lStructSize = 76
SaveFileName.hWndOwner = hWndParent
SaveFileName.hInstance = 0
SaveFileName.lpstrFilter = LocalAlloc(LMEM_ZEROINIT,Len(Filter) + 2) // 2
nulls to signal end
If SaveFileName.lpstrFilter = 0 Then
    MessageBox("error","Cannot alloc requested memory!",StopSign!,Ok!)
    Return(False)
End If

// Tab separator to Null separator
li_Start  = 1
li_TabPos = Pos(Filter,"~t",1)
Do While li_TabPos > 0
    ls_Token = Mid(Filter,li_Start,li_TabPos - li_Start)
    StrCopy(SaveFileName.lpstrFilter + (li_Start -
1),ls_Token,Len(ls_Token))
    li_Start        = li_TabPos + 1
    li_TabPos   = Pos(Filter,"~t",li_TabPos + 1)
    Filters[UpperBound(Filters) + 1] = Mid(ls_Token,2)
Loop
ls_Token    = Mid(Filter,li_Start)
StrCopy(SaveFileName.lpstrFilter + (li_Start - 1),ls_Token,Len(ls_Token))

SaveFileName.lpstrCustomFilter = 0
SaveFileName.nMaxCustomFilter = 0
SaveFileName.nFilterIndex = FilterIndex
SaveFileName.lpstrFile = LocalAlloc(LMEM_ZEROINIT,MAXFILENAME)
SaveFileName.nMaxFile = MAXFILENAME
SaveFileName.lpstrFileTitle = LocalAlloc(0,MAXFILENAME)
SaveFileName.nMaxFileTitle = MAXFILENAME
SaveFileName.lpstrInitialDir = LocalAlloc(0,MAXPATHNAME)
StrCopy(SaveFileName.lpstrInitialDir,InitialDir,Len(InitialDir))
SaveFileName.lpstrTitle = LocalAlloc(0,255)
```

```
StrCopy(SaveFileName.lpstrTitle,Title,Len(Title))
SaveFileName.Flags = Flags
SaveFileName.nFileOffSet = 0
SaveFileName.nFileExtension = 0
SaveFileName.lpstrDefExt = 0
SaveFileName.lCustData = 0
SaveFileName.lpfnHook = 0
SaveFileName.lpTemplateName = 0

If GetSaveFileNameA(SaveFileName) = 1 Then // Pressed OK button
    PathName =
Left(String(SaveFileName.lpstrFile,"address"),SaveFileName.nFileOffSet -
1)
    FileName = String(SaveFileName.lpstrFile +
SaveFileName.nFileOffSet,"address")
    If SaveFileName.nFileExtension = Len(PathName) + Len(FileName) + 1 Then
        FileName += Filters[2 * SaveFileName.nFilterIndex]
    End If
Else
    FileName = ""
End If

LocalFree(SaveFileName.lpstrFilter)
LocalFree(SaveFileName.lpstrFile)
LocalFree(SaveFileName.lpstrFileTitle)
LocalFree(SaveFileName.lpstrTitle)
LocalFree(SaveFileName.lpstrInitialDir)
Return(FileName <> "")
```

(4) printdialog()函数的脚本

```
/*显示打印对话框
成员变量:
Integer FromPage                          // 从页面打印
Integer ToPage                            // 打印到页面
Integer MinPage                           // 最小页面
Integer MaxPage                           // 最大页面
PROTECTEDWRITE Integer Copies = 1   // 复制打印的数量
*/

PRINTDLG PrintDlg
DEVMODE DevMode
DEVNAMES DevNames
long pDevNames, pDevMode
PrintDlg.lStructSize = 66
PrintDlg.hWndOwner = hWndParent
PrintDlg.hDevMode = 0
PrintDlg.hDevNames = 0
PrintDlg.hDC = 0
PrintDlg.Flags = Flags
PrintDlg.nFromPage = FromPage
PrintDlg.nToPage = ToPage
PrintDlg.nMinPage = MinPage
PrintDlg.nMaxPage = MaxPage
PrintDlg.nCopies = 0
PrintDlg.hInstance = 0
PrintDlg.lCustData = 0
PrintDlg.lpfnPrintHook = 0
PrintDlg.lpfnSetupHook = 0
PrintDlg.lpPrintTemplateName = 0
PrintDlg.lpSetupTemplateName = 0
```

```
PrintDlg.hPrintTemplate = 0
PrintDlg.hSetupTemplate = 0

If PrintDlgA(PrintDlg) = 1 Then

    pDevMode = LocalLock(PrintDlg.hDevMode)
    GetDevMode(DevMode,pDevMode,148) // Lock dynamic memory handle
    LocalUnlock(pDevMode) // Unlock dynamic memory handle
    pDevNames = LocalLock(PrintDlg.hDevNames)
    GetDevNames(DevNames,pDevNames,8) // Lock dynamic memory handle
    LocalUnlock(pDevNames) // Unlock dynamic memory handle
    FromPage    = PrintDlg.nFromPage
    ToPage  = PrintDlg.nToPage
    MinPage     = PrintDlg.nMinPage
    MaxPage     = PrintDlg.nMaxPage
    Copies  = PrintDlg.nCopies
    If Copies = 1 Then // Copies are provided by devmode..
        Copies = DevMode.dmCopies
    End If
    SetPrinter(String(pDevNames + DevNames.wDeviceOffset,"address"), &
      String(pDevNames + DevNames.wDriverOffset,"address"), &
      String(pDevNames + DevNames.wOutPutOffset,"address"))
    Return(True)
End If
Return(False)
```

(5) fontdialog()函数的脚本

```
/*这个函数显示了 COMDLG32 库中的字体对话框
成员变量:
Integer FontType, SizeMin, SizeMax;
string FaceName,long Height, Weight;
Boolean Italic, Underline,StrikeOut
*/
CHOOSEFONT ChooseFont
LOGFONT LogFont
Boolean lb_Result
if Flag=0 then
Flags
=BitOr({CF_FORCEFONTEXIST,CF_BOTH,CF_EFFECTS,CF_INITTOLOGFONTSTRUCT})

LogFont.lfFaceName  = FaceName
LogFont.lfHeight        = Height
LogFont.lfWeight        = Weight
If Italic Then LogFont.lfItalic = Char(255) Else LogFont.lfItalic = Char(0)
If Underline Then LogFont.lfUnderline = Char(1) Else LogFont.lfUnderline =
Char(0)
If StrikeOut Then LogFont.lfStrikeOut = Char(1) Else LogFont.lfStrikeOut =
Char(0)

ChooseFont.lStructSize = 60
ChooseFont.hWndOwner = hWndParent
ChooseFont.hDC = 0
ChooseFont.lpLogFont = LocalAlloc(0,60)
PutLogFont(ChooseFont.lpLogFont,LogFont,60)
ChooseFont.iPointSize = 0
ChooseFont.Flags = Flags
ChooseFont.RGBColors = RGBColors
ChooseFont.lCustData = 0
ChooseFont.lpfnHook = 0
ChooseFont.lpTemplateName = 0
```

```
ChooseFont.hInstance = 0
ChooseFont.lpszStyle = 0
ChooseFont.nFontType = FontType
ChooseFont.nSizeMin = SizeMin
ChooseFont.nSizeMax = SizeMax

If ChooseFontA(ChooseFont) = 1 Then
    GetLogFont(LogFont,ChooseFont.lpLogFont,60)
    FontType    = ChooseFont.nFontType
    FaceName    = LogFont.lfFaceName
    Height  = LogFont.lfHeight
    Weight  = LogFont.lfWeight
    Italic  = (Asc(LogFont.lfItalic)     = 255)
    Underline= (Asc(LogFont.lfUnderline) = 1)
    StrikeOut= (Asc(LogFont.lfStrikeOut) = 1)
    lb_Result = True
Else
    FontType    = 0
    FaceName    = ""
    Height  = 0
    Weight  = 0
    Italic  = False
    Underline= False
    StrikeOut= False
    lb_Result = False
End If
LocalFree(ChooseFont.lpLogFont)
Return(lb_Result)
```

(6) setprinter()函数的脚本

```
//基于打印对话框来设置默认打印机的函数
//语句:
//打印机名称 => 新打印机的名称; 驱动 => 新打印机的驱动; 端口 => 新打印机的端口

Environment Env
string ls_WinDir
GetEnvironment(Env)
Choose Case Env.OSType
    Case Windows!
    RegistrySet("HKEY_LOCAL_MACHINE\Config\0001\System\CurrentControlSet
\Control\Print\Printers", "Default", PrinterName)
        ls_WinDir = Space(260)
        GetWindowsDirectoryA(ls_WinDir,260) //获得 Windows 目录
        SetProfileString(ls_WinDir +
"\WIN.INI","Windows","Device",PrinterName + "," + Driver + "," + Port)
    Case WindowsNT!
        RegistrySet("HKEY_CURRENT_USER\Software\Microsoft\Windows
NT\CurrentVersion\Windows","Device", PrinterName)
End Choose
```

4. 建立窗体

建立窗体的具体步骤为:

(1) 在 PowerBuilder 的工具栏中单击 New 按钮，在弹出的对话框中选择 PB Object 选项卡，在其中选择 Window 图标，单击 OK 按钮。

(2) 然后，PowerBuilder 会打开该窗体的工作画板，将该窗体命名为 w_main，标题设置为“Common Dialogs Example”，控件及其属性设置如下:

命令按钮：cb_1~cb_4。
cb_1.text=“Open Dialog”。
cb_2.text=“Save Dialog”。
cb_3.text=“Print Dialog”。
cb_4.text=“Font Dialog”。
分组框：gb_1。
gb_1.text=“Dialogs”。
静态文本框：st_1。
st_1.text=“lin.yahoo.com”。

(3) 最后编写窗体和各个控件的脚本。

① “Open Dialog”事件(cb_1.clicked())的脚本

设置参数，调用 OpenDialog()函数。

```
nvo_commdlg Dialogs
Integer li_Files

Dialogs.hWndParent  = Handle(Parent)
Dialogs.Title       = "Open image or text file"
Dialogs.Filter      = "Text Files (*.txt)~t*.TXT~tBitmap files
(*.bmp)~t*.BMP"
Dialogs.FilterIndex = 2
Dialogs.InitialDir  = "C:\WINDOWS"
Dialogs.Flags           =
BitOr({Dialogs.OFN_EXPLORER,Dialogs.OFN_FILEMUSTEXIST,Dialogs.OFN_LONGNA
MES, Dialogs.OFN_ALLOWMULTISELECT})

If Dialogs.OpenDialog() Then
    For li_Files = 1 To Dialogs.FileCount
        MessageBox("Files",Dialogs.PathName + "\" +
Dialogs.Files[li_Files])
    Next
End If
```

② “Save Dialog”事件(cb_2.clicked())的脚本

设置参数，调用 SaverDialog()函数。

```
nvo_commdlg Dialogs
Integer li_Files

Dialogs.hWndParent  = Handle(Parent)
Dialogs.Title = "Export"
Dialogs.Filter = "Export to SQL (*.sql)~t*.SQL~tExport to ANSI
(*.ans)~t*.ANS~tExport to ASCII (*.asc)~t*.ASC"
Dialogs.FilterIndex = 1
Dialogs.InitialDir = "C:\"
Dialogs.Flags = BitOr({Dialogs.OFN_EXPLORER,Dialogs.OFN_LONGNAMES,
Dialogs.OFN_PATHMUSTEXIST})

If Dialogs.SaveDialog() Then
    MessageBox("FileTitle",Dialogs.PathName + "\" + Dialogs.FileName)
End If
```

③ “Print Dialog”事件(cb_1.clicked())的脚本

设置参数，调用 PrintDialog()函数。

```
nvo_commdlg Dialogs

Dialogs.hWndParent= Handle(Parent)
Dialogs.MinPage = 1
```

```
Dialogs.MaxPage = 5
Dialogs.FromPage = 1
Dialogs.ToPage = 5
Dialogs.Flags = itOr({Dialogs.PD_ALLPAGES,Dialogs.PD_DISABLEPRINTTOFILE})

If Dialogs.PrintDialog() Then
    MessageBox("Message","Choose to print from page " +
String(Dialogs.FromPage) + " to " + & String(Dialogs.ToPage) + ",  " +
String(Dialogs.Copies) + " copies")
End If
```

④ “Font Dialog”事件(cb_1.clicked())的脚本

设置参数，调用 FontDialog()函数。

```
nvo_commdlg Dialogs
Dialogs.hWndParent= Handle(Parent)
Dialogs.FaceName = "Times New Roman"
Dialogs.Height = -15
Dialogs.Underline = True
Dialogs.Italic = True
Dialogs.Flags =
BitOr({Dialogs.CF_FORCEFONTEXIST,Dialogs.CF_BOTH,Dialogs.CF_EFFECTS,Dial
ogs.CF_INITTOLOGFONTSTRUCT})
If Dialogs.FontDialog() Then
    MessageBox("Font Options","Font is " + Dialogs.FaceName + " of size "
+ String(Dialogs.Height) + ", italic is " + String(Dialogs.Italic) + ",
underline is " + String(Dialogs.Underline))
End If
```

完成了上述工作，就得到了一个使用简单的 PowerBuilder 程序设计实现的通用对话框，主界面如图 11-48 所示。

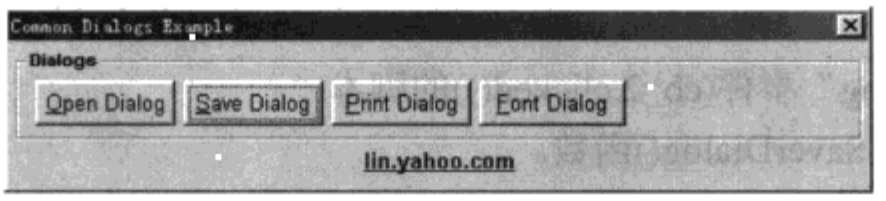


图 11-48　通用对话框实例的主界面

单击 Open Dialog 按钮，弹出打开文件对话框，如图 11-49 所示。

单击 Save Dialog 按钮，弹出保存文件对话框，如图 11-50 所示。

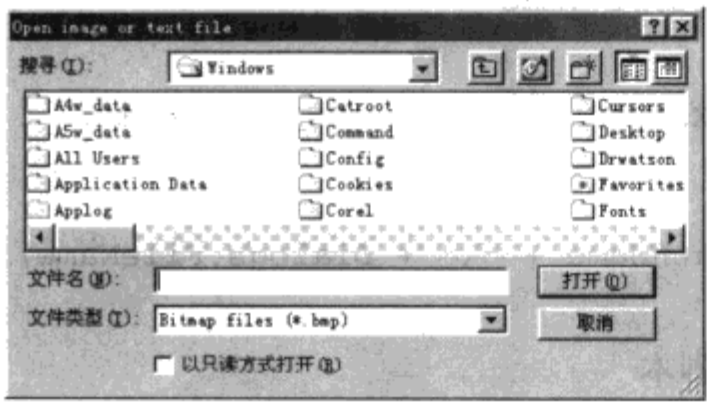


图 11-49　打开文件对话框

图 11-50　保存文件对话框

单击 Print Dialog 按钮，弹出打印对话框，如图 11-51 所示。

单击 Font Dialog 按钮，弹出字体设置对话框，如图 11-52 所示。

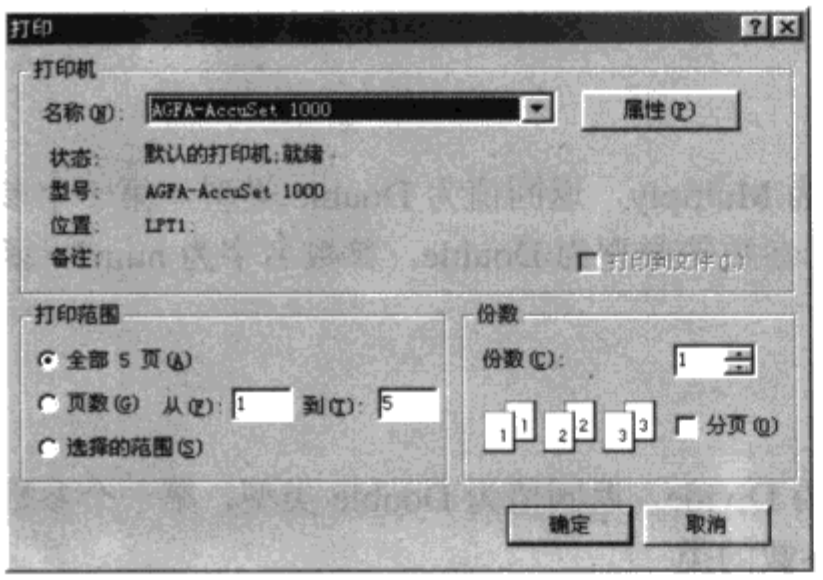


图 11-51　打印对话框

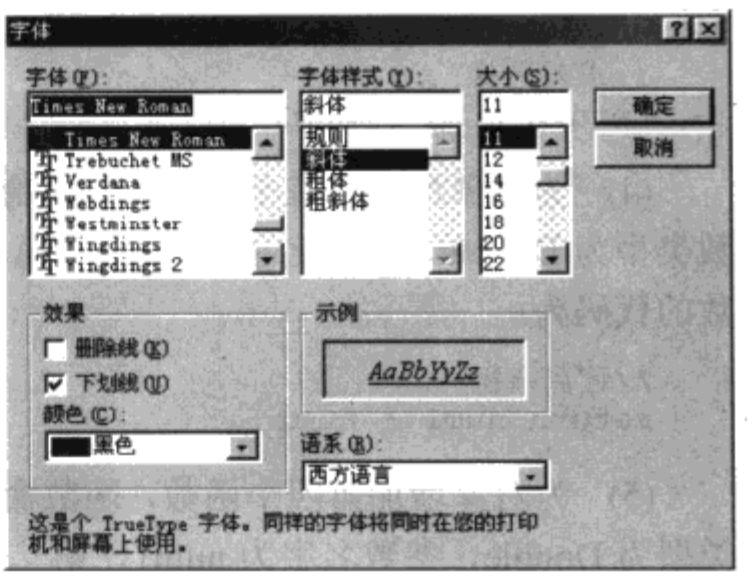


图 11-52　字体设置对话框

11.6.2　设计计算器

下面创建一个定制类用户对象，并为这个用户对象添加四个函数用于计算两个数值的和、差、乘积和商，然后在窗口中调用这个定制类用户对象实现一个简单的计算器。

1. 创建程序的基本框架

(1) 建立 work 工作区。

(2) 建立 app 应用。

2. 建立类用户对象

(1) 新建一个定制类用户对象，并为其插入一个函数，如图 11-53 所示。

(2) 将函数命名为 Add，返回值为 Double 类型，第一个参数类型为 Double，参数名字为 num1，第二个参数的类型为 Double，参数名字为 num2，函数的代码为：

```
//求和运算
return num1 + num2
```

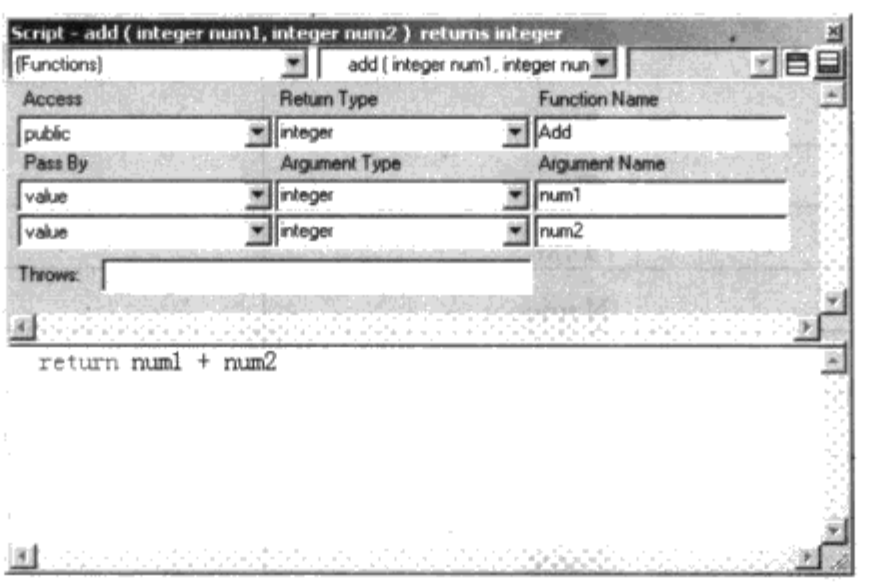


图 11-53　插入 Add 函数

(3) 增加第二个函数，函数命名为 Sub，返回值为 double 类型，第一个参数类型为 Double，参数名字为 num1，第二个参数的类型为 Double，参数名字为 num2，函数的代码为：

```
//求差运算
return num1 - num2
```

(4) 为对象增加第三个函数，函数命名为 Multiply，返回值为 Double 类型，第一个参数类型为 Double，参数名字为 num1，第二个参数的类型为 Double，参数名字为 num2，函数的代码为：

```
//计算乘积
return num1 * num2
```

(5) 为对象增加第四个函数，函数命名为 Divide，返回值为 Double 类型，第一个参数类型为 Double，参数名字为 num1，第二个参数的类型为 Double，参数名字为 num2，函数的代码为：

```
//除法运算
if num2<> 0 then
    return num1 / num2
else
    return -1          //如果除数为 0，则返回-1
end if
```

(6) 单击工具栏上的 Save 按钮，打开 Save User Object 对话框，如图 11-54 所示。

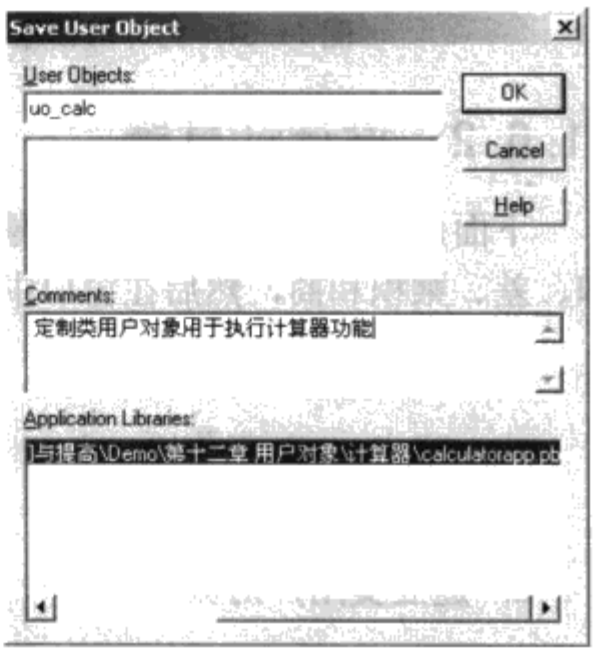


图 11-54 Save User Object 对话框

(7) 将新建的定制类用户对象命名为“uo_calc”，在 Comments 中输入“定制类用户对象用于执行计算器功能”，然后单击 OK 按钮保存对象，并返回用户对象画板。

(8) 新建一个窗口将其命名为 w_calculatorapp_main，设置其 Title 属性为“简单的计算器”。在窗口中增加控件见表 11-2，控件的布局如图 11-55 所示。

表 11-2 窗口中的控件

控件类型	属 性	属 性 值
静态文本框	Text	第一个数
静态文本框	Text	第二个数
静态文本框	Text	计算结果
单行文本框	Name	sle_num1
	Text	空
单行文本框	Name	sle_num2
	Text	空
单行文本框	Name	sle_result
	Text	空

续表

控件类型	属　性	属　性　值
命令按钮	Name	cb_add
	Text	+
命令按钮	Name	cb_sub
	Text	−
命令按钮	Name	cb_multiply
	Text	×
命令按钮	Name	cb_divide
	Text	/

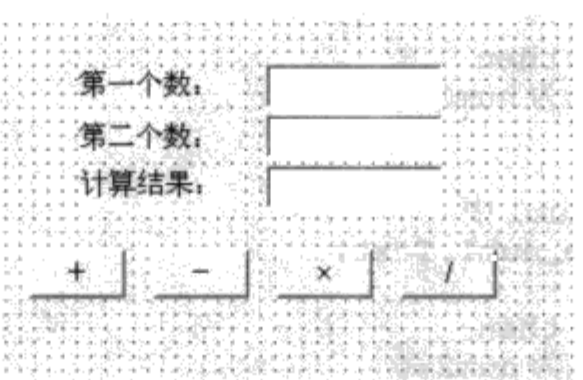


图 11-55　控件布局

(9) 在窗口对象画板的 Declare Instance Variables 视图中，声明定制类用户对象类型的变量，代码如下：

```
//声明定制类用户对象变量
uo_calc cal
```

(10) 在窗口对象画板的 Script 视图中为窗口的 Open 事件添加事件处理代码，用于创建定制类用户对象实例，代码如下：

```
//创建定制类用户对象实例
cal = Create uo_calc
```

(11) 在窗口对象画板的 Script 视图中为窗口的 Close 事件添加事件处理代码，用于删除定制类用户对象实例，回收内存空间，代码如下：

```
//删除定制类用户对象变量，回收内存
Destroy cal
```

(12) 在窗口对象画板的 Script 视图中为命令按钮 cb_add 的 clicked 事件添加事件处理代码，用于将两个数相加，代码如下：

```
//定义变量用户保存输入值
double num1
double num2
if sle_num1.Text ="" then
    //如果输入值为空则认为 num1=0
    num1 = 0;
else
    //将用户输入值存入 num1 中
    num1 = Double(sle_num1.Text)
```

```
end if
if sle_num2.Text ="" then
    //如果输入值为空则认为 num2=0
    num2 = 0;
else
    //将用户输入值存入 num2 中
    num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Add 函数，计算两个数的和
//将结果显示在计算结果文本框中
sle_result.Text = String(cal.Add(num1,num2))
```

(13) 在窗口对象画板的 Script 视图中为命令按钮 cb_sub 的 clicked 事件添加事件处理代码，用于将两个数相减，代码如下：

```
//定义变量用户保存输入值
double num1
double num2
if sle_num1.Text ="" then
    //如果输入值为空则认为 num1=0
    num1 = 0;
else
    //将用户输入值存入 num1 中
    num1 = Double(sle_num1.Text)
end if
if sle_num2.Text ="" then
    //如果输入值为空则认为 num2=0
    num2 = 0;
else
    //将用户输入值存入 num2 中
    num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Sub 函数，计算两个数的差
//将结果显示在计算结果文本框中
sle_result.Text = String(cal.Sub(num1,num2))
```

(14) 在窗口对象画板的 Script 视图中为命令按钮 cb_multiply 的 clicked 事件添加事件处理代码，用于将两个数相乘，代码如下：

```
//定义变量用户保存输入值
double num1
double num2
if sle_num1.Text ="" then
    //如果输入值为空则认为 num1=0
    num1 = 0;
else
    //将用户输入值存入 num1 中
    num1 = Double(sle_num1.Text)
end if
if sle_num2.Text ="" then
    //如果输入值为空则认为 num2=0
    num2 = 0;
else
    //将用户输入值存入 num2 中
    num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Multiply 函数，计算两个数的乘积
//将结果显示在计算结果文本框中
sle_result.Text = String(cal.Multiply(num1,num2))
```

(15) 在窗口对象画板的 Script 视图中为命令按钮 cb_divide 的 clicked 事件添加事件处理代码，用于将两个数相除，代码如下：

```
//定义变量用户保存输入值
double num1
double num2
if sle_num1.Text ="" then
    //如果输入值为空则认为 num1=0
    num1 = 0;
else
    //将用户输入值存入 num1 中
    num1 = Double(sle_num1.Text)
end if
if sle_num2.Text ="" then
    //如果输入值为空则认为 num2=0
    num2 = 0;
else
    //将用户输入值存入 num2 中
    num2 = Double(sle_num2.Text)
end if
//调用定制类用户对象 Divide 函数,计算两个数的商
//将结果显示在计算结果文本框中
sle_result.Text =
String(cal.Divide(num1,num2))
```

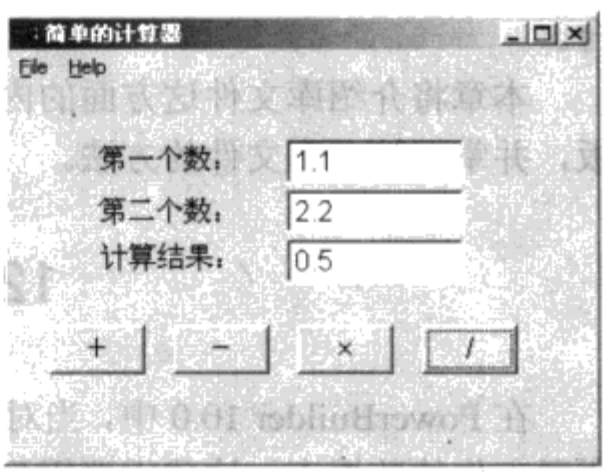


图 11-56　计算器程序运行界面

(16) 单击工具栏上的 Run 按钮，程序运行界面如图 11-56 所示。

11.7　习　　题

1.　填空题

(1) _____是面向对象的一个重要特征，通过它能够获得父类已有的性质和特征，不必重新定义和编写代码实现。

(2) PowerBuilder 把用户对象分为两大类分别是_____和_____。

(3) 可视用户对象包括_____、_____和_____。

(4) 类用户对象包括_____和_____。

2.　问答题

(1) 如何创建标准可视用户对象？

(2) 如何定制用户对象？

(3) 如何实现在窗口与用户对象间的通讯？

(4) 如何继承创建？

第 12 章　管理库文件

PowerBuilder 中包含窗口、菜单、数据窗口、报表等对象，用户对对象所做的各种操作都被存储在库文件(PBL 文件)中，而库画板中又包含了所有对库文件的操作，库画板是用来检查并管理库文件的一个有效工具。用户可以在自己的库画板中进行各种针对数据库的操作，而且库画板中树形视图、列表视图也为用户提供了一个更为方便、易于操作的工作平台。

本章将介绍库文件这方面的内容，目的是让用户全面了解 PowerBuilder 10.0 中的库画板，并掌握管理库文件的方法。

12.1　库文件概述

在 PowerBuilder 10.0 中，当对一个 PowerScript 对象进行操作时，首先要指定这个对象所要工作的数据库，这就需要用到库文件及库画板。本节将介绍库文件的基本概念，并介绍库画板的基本使用方法。

12.1.1　库文件

当在某个画板中保存一个对象(如窗口对象、菜单对象等)时，PowerBuilder 将自动把它们存放在一个后缀名为.pbl 的库文件中。同样，当在某个画板中打开一个对象时，PowerBuilder 将从库文件中重新获得该对象。用户对对象所做的各种操作都会被记录在该应用程序的库文件中，这也体现出了库文件在 PowerBuilder 中的重要性。

库画板、库文件与各种其他画板的关系如图 12-1 所示。

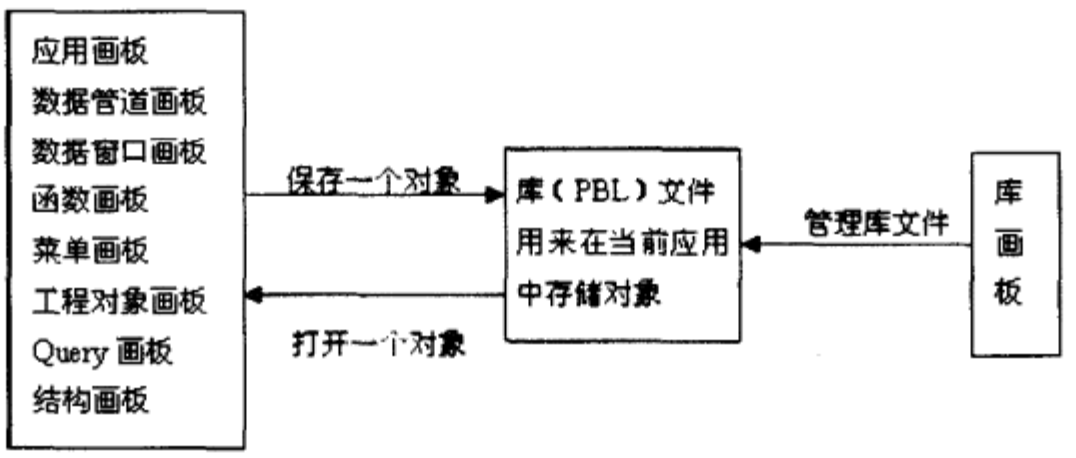


图 12-1　库画板、库文件与其他画板关系图

在应用程序中，PowerScript 能够使用任意多个库，这些库可以被存放在开发者的个人计算机上、工作站上或服务器上。当创建了一个目标程序之后，就可以为它指定所使用的库，而在开发过程中，还可以随时为一个目标程序更换库的搜索路径。

1. 库的使用

存放在库中的对象有两种形式，包括源文件形式和目标文件形式。

- 源文件形式：它是一种以语法来表现对象的形式，被包含在脚本代码中。
- 目标文件形式：它是一种以二进制来表现对象的形式，它与 C 和 C++语言中的目标文件相似。在保存一个对象时，PowerBuilder 会自动对其进行编译。

在一个小型应用程序中，也许只会用到一个库，但是在大型目标程序中，用户可能希望将应用程序分别存放在不同的库中。PowerBuilder 可以使用户在任意时刻都能很便捷地对库进行增/删调整。

在 PowerBuilder 10.0 中，系统对所使用的库文件的个数并没有要求，可以根据需要自由地进行设置。但是，出于对程序整体运行性能和开发难度方面的考虑，用户应该遵守以下的一些规则：

- 对象的数量：在一个库中，所使用的对象最好不要超过 60 个。虽然对象数量的多少并不会直接影响应用程序的运行性能，但它将会影响应用程序的开发难度。如果在一个库中拥有过多的对象，将会使库文件的列表框变得难以操作和管理，并且库画板也会变得很难使用。
- 平衡：最好不要在只有少数几个对象时设计较多数量的库。库数量的增多将会使搜索路径加长，这样将迫使 PowerBuilder 必须在多个库中去搜索一个对象，会降低应用程序的整体运行性能。因此，开发者应该注意在应用程序的大小与库的数量关系间保持相对平衡。

2. 库的管理

用户能够以任何方式对自己的库进行管理。例如，可以将所有相同类型的对象放置在各自的库中，也可以将对象分为若干个子系统，并将每个子系统放置在各自的库中。

如果正在与其他开发者共同进行一项大型目标程序的开发工作，最好在一个网络服务器上将包含有对象的库与其他开发者共享，这样每个开发者只能有权进入自己的开发部分，避免不同用户间的相互干扰。无论是在 PC 机或是工作站上，开发者都可以将对象放置在一个只有个人开发和维护的库中，这样可以直接避免不同开发者对同一个库的交叉操作。

提示： 如果用户正在使用的是一个支持添加目标程序和更新状态的源代码版本控制系统，那么在使用这些支持特性时能够拥有一个属于自己的本地共享复制库，并且能够使它与源代码控制系统保持同步。

12.1.2　库画板

在库画板中，可以进行以下操作：

- 创建一个新库；
- 在当前工作空间的目标程序中创建新对象；
- 在任何库中复制、移动和删除对象；
- 打开当前工作空间库列表中的对象并在适当的画板中进行编辑；
- 在当前工作空间中升级、重建并重新生成库；
- 使用检入(check-in)和检出(check-out)或版本控制系统控制对库对象的修改；

- 创建一个包含当前对象库的对象，并创建相关资源对象的动态库。

注意： 用户不能在一个不属于库列表的 PowerBuilder 库中进行升级或打开对象的操作，也不能对一个库重命名。

可以使用两种途径打开库画板：

- 在工具栏上单击 Library 图标；
- 从菜单栏中选择 Tools | Library Painter 命令。

库画板有两个视图区，分别为树形视图和列表视图，如图 12-2 所示。在库画板中不仅可以显示出 PowerBuilder 对象，而且还可以显示出当前系统中所有目录。

树形视图用来显示当前用户计算机上的驱动器、工作空间、目标程序、库、对象和它们所包含的文件等，用户可以将驱动器、文件夹和库等通过单击鼠标的操作展开。列表视图用来显示用户所选中的驱动器、文件夹、库等的内容。对于库，列表视图中显示一些与库相关的注释；对于库中的对象，列表视图中列出了对象名称、修改日期、大小和其他与该对象相关的注释。

树形视图和列表视图都可以从 View 菜单中选取。默认情况下，库画板在视图区域的左侧显示树形视图，右侧显示下拉视图。当打开库画板时，树形视图和列表视图将同时显示用户计算机上的包括网络驱动器在内的所有驱动器。

注意： 两个视图的相对位置不是固定不变的，可以根据自己的习惯通过拖曳操作来重新划分该视图区域的结构。

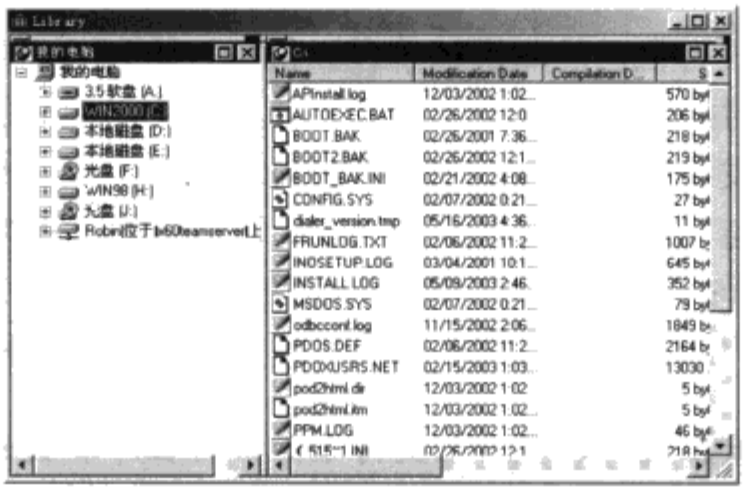


图 12-2 库画板视图

在库画板中的树形视图与系统目录树视图中的 Workspace 选项卡视图相似。通过使用右键菜单、工具栏 PainterBar 和库画板的菜单栏，用户可以在树形视图与列表视图中完成绝大部分的操作。当系统目录树以及一个库画板同时打开时，工具条和菜单条中的操作只是相对于库画板的。

库画板工具栏如图 12-3 所示，在默认情况下只有一个可用的工具栏 PainterBar，它所包含的各种按钮作用如下。

图 12-3 库画板工具栏

- CreateLibrary：创建新的库文件。
- SelectAll：选择当前列表视图中的全部对象。
- Edit：编辑当前所选择的 PowerBuilder 对象。
- Copy：复制当前所选择的 PowerBuilder 对象。
- Move：将当前所选择的 PowerBuilder 对象移动到其他库文件中。
- Delete：删除当前所选择文件。
- Export：将当前所选择的 PowerBuilder 对象导出保存到另外一个独立文件中。
- Import：将一个以独立文件形式存储的 PowerBuilder 对象导入到当前库文件中。
- Regenerate：当检入、检出状态发生改变后用于重建库中的 PowerBuilder 对象。
- Search：在当前所选择的一个或多个 PowerBuilder 对象中查找字符串。
- Properties：设置所选库文件或 PowerBuilder 对象的属性。
- CheckIn：检入 PowerBuilder 对象。
- CheckOut：检出 PowerBuilder 对象，以便修改。
- CompareDifferences：与存档对象进行比较。
- DisplayMostRecentObject：显示最近打开过的对象。
- Close：关闭库画板。

当使用库画板时，每次单击工具栏上的图标按钮 Library 后，PowerBuilder 都将打开一个新的库画板实例。与之相比，使用系统目录树的一个优点就是无论单击工具栏上的系统目录树按钮多少次，都只打开惟一的一个实例。

虽然库画板与 Windows 系统中的资源管理器在外观和操作上非常相似，但是它也有其与众不同的地方，库画板可以显示两个或两个以上的树形视图或列表视图，这个可以在不同的树形视图或列表视图中显示出不同对象及对象内容列表的特点，可以使用户在工作中得到更大的便利。下面就以设置包含有一个树形视图和 3 个列表视图的库画板的方法为例进行介绍，界面如图 12-4 所示。

(1) 打开一个库画板，默认显示为一个树形视图和一个列表视图，如图 12-2 所示。

(2) 打开 View 菜单，执行下拉菜单中的 List 命令，此时显示为一个树形视图和两个列表视图。

(3) 重复上一步骤的操作，此时显示为一个树形视图和 3 个列表视图。

(4) 拖动列表视图的活动窗口标题栏，调整各视图的相对位置。

(5) 选中第 1 个列表视图，然后在树形视图中选中一个文件夹，此时在该列表视图中显示该文件夹中所包含的各文件名称、修改日期、文件大小等属性信息。

(6) 选中第 2 个列表视图，然后在树形视图中选中一个库文件，此时在该列表视图中显示该库文件中所包含的数据窗口、用户等对象的名称、修改日期、编辑日期和文件大小等信息。

(7) 选中第 3 个列表视图，重复上一步骤的操作，选中另外一个库文件并在该列表视图中显示相应信息。

(8) 这时可以看到，在不同的列表视图中显示了不同的信息，方便用户进行选择性操作。

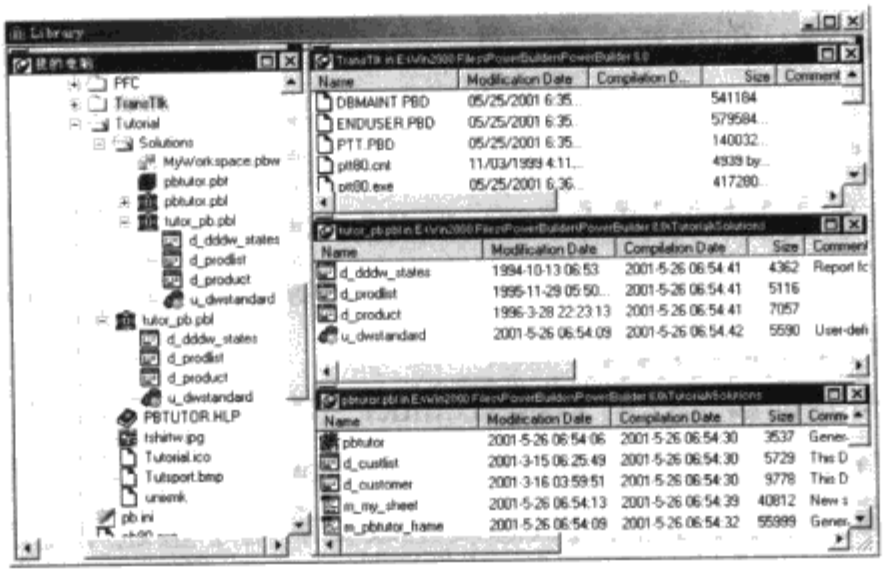


图 12-4 设置树形视图和列表视图

12.2 管理 PBL 库

PowerBuilder 中的库画板是对库文件进行管理的工具，库文件的管理包括对库文件的创建、删除、优化、重建以及搜索路径设置等操作。

12.2.1 创建和删除库

PBL 库一般是在用户初始化文件时创建的，但除此之外，也可以在开发过程中创建库，创建库的方法如下：

(1) 单击库画板工具栏上的 Create Library 图标按钮，或在菜单栏上选择 Entry Library Create，打开当前路径 Create Library 对话框，如图 12-5 所示。

(2) 在文件名编辑框中输入希望创建的库文件名称，并指定存放路径，库文件的扩展名为.pbl。

(3) 单击“保存”按钮保存对库文件名称及路径的设置，弹出 Properties 对话框，如图 12-6 所示。

(4) 在 Comments 框中输入与该库有关的注释信息。

(5) 单击 OK 按钮保存设置并退出，即可完成一个新库的创建。

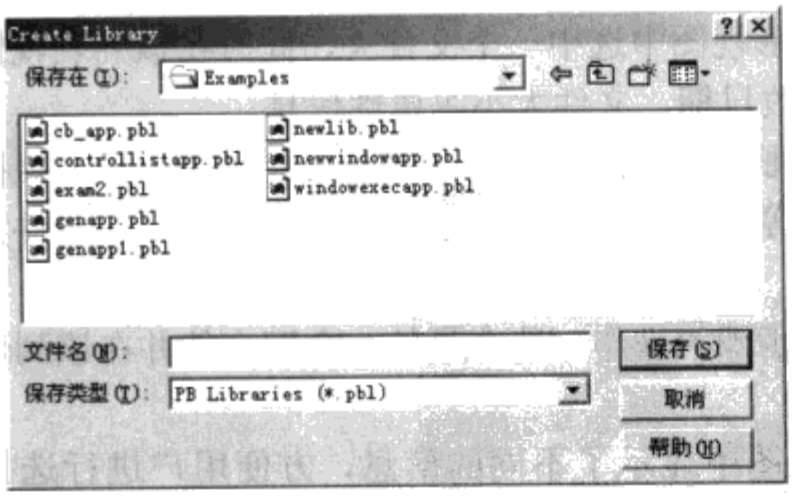


图 12-5 Create Library 对话框

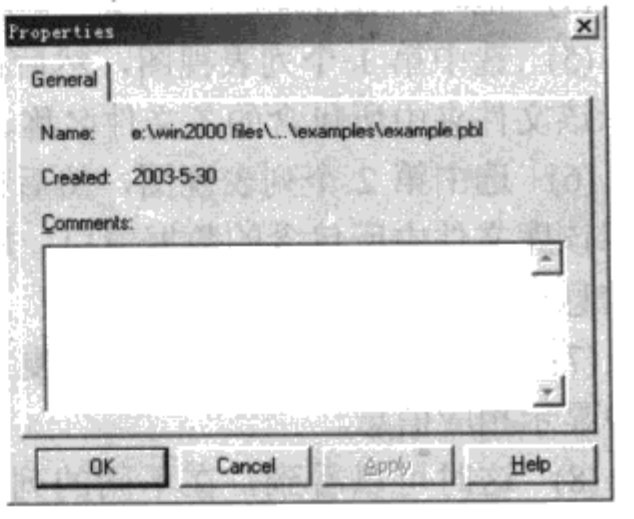


图 12-6 Properties 对话框

将一个库从 PowerBuilder 当前开发环境中删除的方法如下：

(1) 在库画板的树形视图或列表视图中选中希望删除的库。

(2) 在菜单栏上选择 Entry Delete，或在右击弹出的菜单中选择 Delete 命令。

(3) 在弹出的 Delete Library 对话框中显示了希望删除的库的绝对路径及名称，如果确认要删除该库，单击“是”按钮，如图 12-7 所示。

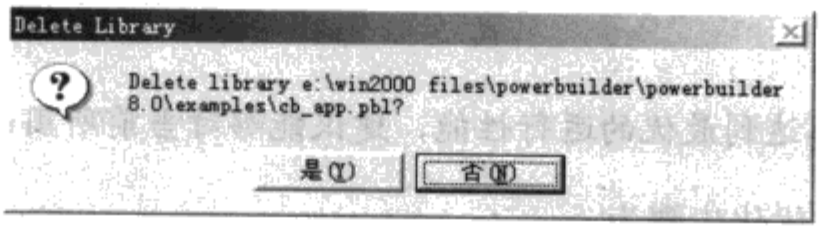


图 12-7　Delete Library 对话框

(4) 返回到库画板视图，可以看到，该库及库中的目标和存储信息已经全部被删除，该操作不能被撤消。

提示： 用户可以在应用程序执行过程中，通过在脚本代码中使用 LibraryCreate()函数和 LibraryDelete()函数完成对库的创建和删除。

12.2.2　创建动态库

PowerBuilder 10.0 库画板中不仅支持在静态环境下库的创建，而且也支持动态库的创建。创建动态库的步骤为：

(1) 选中一个将要用来创建动态库的库。

(2) 在菜单栏上选择 Entry Library Build Runtime Library，或从右击该选中库时弹出的快捷菜单中选择 Build Runtime Library 命令，打开列有被选中库名称的 Build Runtime Library 对话框，如图 12-8 所示。

(3) 如果要为所创建的动态库指定资源文件，应在 Resource File Name 编辑框中指定 PowerBuilder 资源文件名或单击 Browse 按钮导入文件。

(4) 在 Build Runtime Library 对话框中的大多数选项只有在 Machine Code 复选框被选中后才会生效。选中 Machine Code 复选框，可编译生成机器代码(DLL 文件)，否则生成伪代码(PBD 文件)。

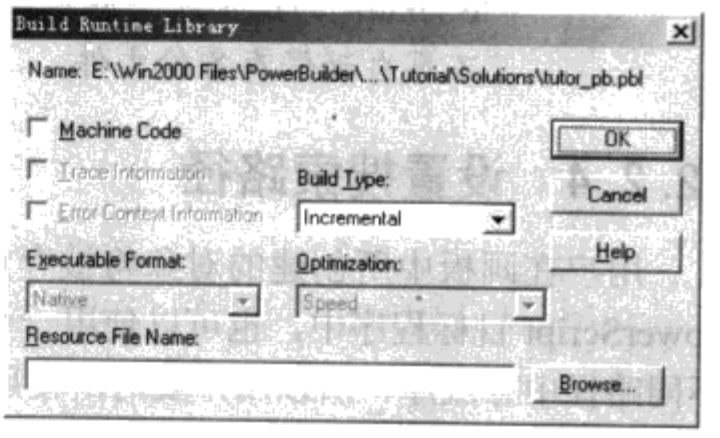


图 12-8　Build Runtime Library 对话框

(5) 单击 OK 按钮保存设置并退出。

PowerBuilder 会在关闭 Build Runtime Library 对话框后，创建一个与所选库同名，扩展名为.dll 或.pbd 的动态库。

12.2.3 优化库文件

用户应该定期对自己的库文件进行优化。在优化过程中，库中的间隙将被删除，而且也同时将库中的碎片进行整理，以便提升整体运行性能。优化只会影响在磁盘上的文件布局，不会对其中所包括的对象内容有影响。在对一个库进行优化的过程中，该库中的对象将不能被编译。

提示： 为了使库达到最优的运行性能，建议能够对当前所用的库每周定时优化。

PowerBuilder 库的优化步骤为：

(1) 在库画板的树形视图或列表视图中，选中要进行优化的库。

(2) 在菜单栏上选择 Entry Library Optimize，或在所选中库的右键菜单中选择 Optimize 选项。

PowerBuilder 在优化过程中将对库结构进行改组，以便对其中的对象、数据碎片和索引定位进行优化。需要注意的是，PowerBuilder 并不会改变优化前库中对目标、事件等存储信息的修改日期。默认情况下，PowerBuilder 将会把本次优化之前的库作为备份文件保存在相同路径下。

通常情况下，库文件经过优化后所生成的优化文件在其默认路径下将拥有默认的使用权限，用户并不需要去做专门的设置，但在某些系统中，新生成的优化文件却不能被默认共享。如果用户发现优化后的文件不能被保存或出现连接错误信息，那么请查看 PBL 库文件的使用权限。

提示： 在默认情况下，PowerBuilder 将会把本次优化之前的库作为备份文件保存在相同路径下。但如果不愿意把优化前的文件保存为备份文件，可以在菜单栏中选择 Design Options，在所打开的对话框中将复选框 Save Optimized Backups 的选中状态取消即可。这样，在以后的库文件优化过程中，系统将不再自动保存备份文件。

12.2.4 设置搜索路径

用户在画板中所创建的对象将被存放在 PowerBuilder 的一个或多个库中，同样，在 PowerScript 目标程序中，也可以使用一个或任意多个库中的对象。因为目标程序中可能在不同路径中存放库，所以用户要对所使用的每一个库定义搜索路径。

在应用程序执行过程中，PowerBuilder 会在用户所指定的搜索路径中查找在该应用程序中被引用的对象，例如：窗口对象、菜单对象或数据窗口对象等。当一个新的对象被引用时，PowerBuilder 将按照用户在库中所指定的搜索路径对库中的对象进行查询，直到该对象目标被找到为止。

在当前应用程序目标属性对话框中的 Library List 选项卡中，可以改变与当前目标程序相关库的设置。

指定目标程序库搜索路径的方法是：

(1) 在系统目录树的 Workspace 选项卡上，选中包含有应用程序的目标程序并单击鼠标右键，在弹出菜单中选择 Properties 选项。这时弹出目标程序的属性对话框，如图 12-9 所示。

(2) 选择 Library List 选项卡，当前被包含在库搜索路径中的所有库都显示在 Library Search Path 列表框中。

(3) 向搜索路径中添加新库的方法有两种：

- 将希望被包含在库搜索路径中的每一个库文件的绝对路径及名称输入到 Library Search Path 列表框中，并用分号“；”将它们分隔开。
- 通过单击 Browse 按钮，在如图 12-10 所示的对话框中查找希望被包含在库搜索路径中的其他 PBL 文件，选中后添加到 Library Search Path 列表框中。

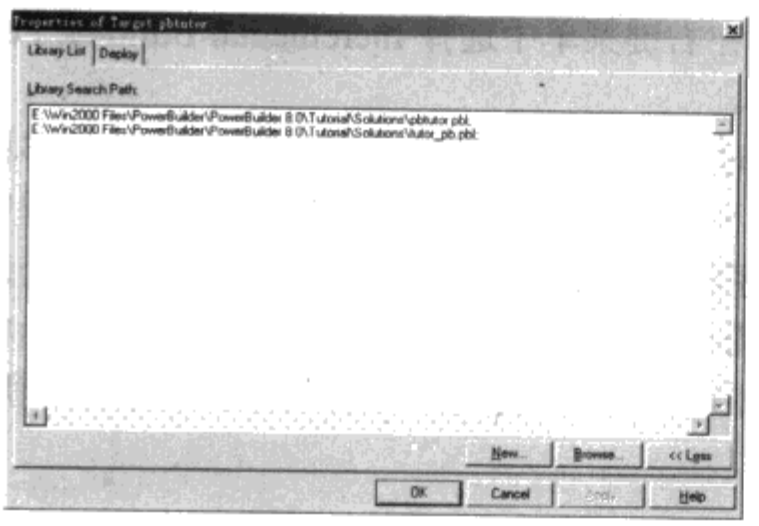


图 12-9　目标程序属性对话框

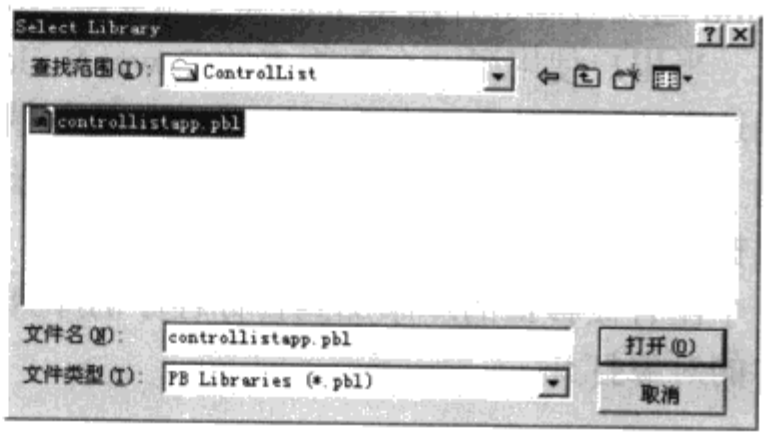


图 12-10　Select Library 对话框

提示： 库文件在 Library Search Path 列表框中的排列顺序决定了该库文件在搜索时的先后顺序，所以应恰当地安排好所希望的搜索顺序。利用右键菜单中的剪切、粘贴命令可以对 PBL 文件的排列顺序进行再调整。

(4) 单击 OK 按钮保存设置并退出，PowerBuilder 将为目标程序更新搜索路径的设置。

注意： PowerBuilder 只有在用户初始化文件的时候才会存储目标程序的库文件搜索路径。在用户定义的每一个应用程序初始化文件的应用设置中都会有一个字符串，这个字符串上列出了在当前应用程序搜索路径中的全部库文件。因此，如果需要将一个正在编译的应用程序中的 PBL 文件添加到其他 PowerBuilder 开发者的程序中，一定要将适当的字符串从初始化文件中复制到其他开发者的初始化文件中，或者添加到这一部分中所讲的其他开发者打开的应用对象和指定库的搜索路径中，这样才能使应用程序的库搜索路径被正确定义到其他开发者的计算机上。

当希望将一个库文件从搜索路径中删除的时候，只需要将该库文件选中，从右击弹出的菜单中选择删除命令，或按下键盘上的 Delete 键，就可以完成删除的操作。

12.2.5 重建与移植库

当用户对目标程序做过改动后，库中对象可能发生变化，这就需要对一个或多个库进行更新。库重建操作可以对所有库对象依照当前排列顺序进行更新。

可以采用以下两种方法对工作空间或目标程序进行重建：

- 增量重建：在参考自从前次创建工作空间或目标程序以后修改过的对象的基础上，对所有对象和库进行更新。
- 全部重建：更新用户工作空间或目标程序中全部的对象和库。

对库中的目标程序重建的方法是，在库画板中选中该目标程序，并在菜单栏上选择 Entry Target Incremental 对库进行增量重建或选择 Entry Target Full 对库进行全部重建。也可以在系统目录树或库画板中选中目标程序，并在右键菜单中选择 Incremental Build 或 Full Build 选项对库进行增量重建或全部重建的操作。

当用户使用了一个新版本的 Power Builder 时，所有现存的目标程序也需要一同被移植到这个新版本的 Power Builder 上去。通常情况下，当打开一个包含有需要移植的目标程序的工作空间时，或将一个需要被移植的目标程序添加到工作空间上时，Power Builder 会及时提示移植该目标程序。但在有些情况下，也需要人工操作来移植目标程序。例如，当将一个没有被移植的库添加到目标程序的库列表中时，只有当该目标被移植后，才能将该库中的对象打开。

12.3 操作库与维护库及其对象

对库及其对象的操作包括库及对象的打开、预览、搜索、导入、导出、生成信息报告等多项内容。

12.3.1 打开和预览对象

如果窗口、菜单等 PowerBuilder 对象在当前工作空间的一个 PBL 库中，那么它们只能被打开。打开一个对象的方法是在树形视图或列表视图中双击该对象，或者是在该对象的右键菜单中选择 Edit 命令。

在库画板中，可以打开显示在视图中大多数类型的文件。当用户选中一个对象并双击鼠标时，PowerBuilder 首先判断该对象是否可以在文件编辑器中被打开，如果可以打开，将该文件打开后显示，例如，后缀名为.txt、.ini 和.sr*的文件都可以在文件编辑器中直接打开。如果判断结果为该对象不能在文件编辑器中被打开，那么 PowerBuilder 会随后判断该对象是否可以在某个画板或超文本(HTML)编辑器中被打开，如果判断结果为该对象还是不能在 HTML 编辑器中被打开，最后 PowerBuilder 会对该对象是否与 HKEY_CLASSES_ROOT 部分中的程序有关联进行判断，如果判断结果为真，则登录到该应用程序中。

可以在库画板中运行窗口或预览数据窗口对象，只需在该对象上单击鼠标右键，在弹出的右键菜单中选择 Run/Preview 命令即可。

12.3.2　复制、移动、粘贴和删除对象

在用户的目标程序需要发生变化时，就要求在库中的对象能够重新排列。例如，当要将目标程序分割成几个不同的库时，用户就需要在库与库之间进行复制与移动的操作，而对那些不再需要的文件来说，就需要将它们删除。

1．利用拖曳操作完成对象的复制

首先，在库画板的树形视图或列表视图中选中希望复制的对象；其次，利用鼠标将该对象拖曳到任意视图的一个库中，就可以完成复制操作。

需要注意的是：在进行对象复制时，如果发现在目标路径中有一个和源文件同名的文件存在，会给出提示信息询问是否覆盖，默认选项为“是”。如果同意覆盖，则由源文件取代目标路径中的同名文件。

2．利用拖曳操作完成对象的移动

首先，在库画板的树形视图或列表视图中选中希望移动的对象；其次，按下 Shift 键并同时用鼠标将该对象拖曳到任意视图的一个库中，就可以完成移动的操作。PowerBuilder 在进行移动的操作时，如果发现目标路径中存在一个和源文件同名的文件，也会给出提示信息询问是否覆盖该同名文件。

提示： 这里所指的“移动”对象的操作就是在 Windows 系统中的剪切、粘贴操作，只不过移动操作将这两步操作合并简化成为一个步骤。

3．使用图标按钮或菜单选项复制或移动对象

在库画板的树形视图或列表视图中选中希望复制或移动的对象，然后按下库画板工具条上的 Copy 或 Move 图标按钮即可进行复制或移动操作，也可以在右键菜单中选择 Copy 或 Move 命令，或在菜单栏上选择 Entry Library Item Copy、Entry Library Item Move 进行复制或移动操作。最后，在弹出的 Select Library 对话框中输入希望复制到的目标路径，单击 OK 按钮确认并保存设置即可。

4．删除对象

在库画板的树形视图或列表视图中选中希望删除的对象，然后按下库画板工具栏上 Delete 图标按钮进行删除操作，也可以在右键菜单中选择 Delete 命令，或在菜单栏上选择 Entry Delete 图标按钮进行删除操作。最后，在弹出的确认对话框中确认该删除操作，单击 OK 按钮即可删除对象。

提示： 默认情况下，PowerBuilder 将对所有的删除操作进行确认，在弹出的对话框中单击 OK 按钮后删除该对象。如果不让系统弹出该确认对话框，而是希望立即执行删除操作，可以在菜单栏上选择 Design Options，打开操作属性对话框，取消对 General 选项卡的 Confirm on Delete 复选框的选中状态，这样在以后的删除操作中，系统就不会再弹出确认删除的对话框了。

12.3.3 搜索目标程序、库和对象

1. 目标程序的全局搜索

在开发 PowerBuilder 应用程序的过程中，可以根据各部分功能、类别等的不同，将源代码分散于不同程序块中独立开发。但是在整个应用程序中，这些单独开发的程序块并不是彼此孤立的，可以通过搜索操作在目标程序中对一个指定文本字符串进行定位。可以搜索以下内容：

- 使用 SetTransObject 函数的全部脚本代码；
- 包括按钮控件 cb_exit 的所有窗口对象(在一个窗口中包含的所有控件都将以列表的形式存放在库中该窗口定义的源代码中，所以用户能够以文本字符串的形式对它们进行查找)；
- 所有在数据库中连接到 Employee 数据表的数据窗口对象。

2. 搜索库和对象

可以对库及一个或多个 PowerBuilder 对象执行搜索操作。在库和 PowerBuilder 对象中搜索文本字符串的步骤如下：

(1) 在树形视图或列表视图中选中一个目标程序、库或对象，也可以在列表视图中按下键盘上的 Shift 或 Ctrl 键进行选择多项。

(2) 在库画板工具栏上选择 Search 图标按钮或在右键菜单中选择 Search 选项，弹出 Search Library Entries 对话框，如图 12-11 所示。

(3) 在 Search Library Entries 对话框的 Search For 编辑框中输入要搜索定位的字符串，该字符串可以是在属性、脚本代码或变量中使用的一个单词的全部或部分字符，也可以是一个短语。

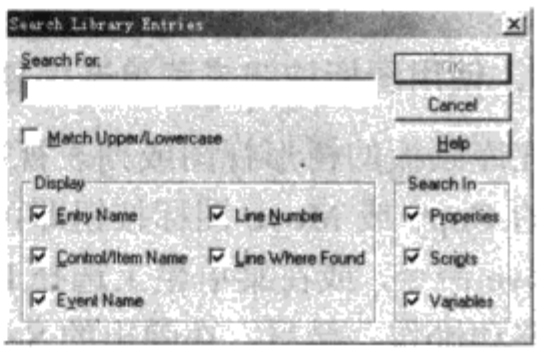


图 12-11 Search Library Entries 对话框

注意： PowerBuilder 搜索过程中不支持通配符，所以在所搜索的字符串中不能使用通配符。

(4) 在 Search Library Entries 对话框的 Display 选项栏中选择希望显示在搜索结果中的信息。

(5) 在 Search Library Entries 对话框的 Search In 选项栏中选择希望 PowerBuilder 在搜索过程中检查到的对象范围，包括属性、脚本和变量。

(6) 单击 OK 按钮，PowerBuilder 开始在库中搜索与输入字符串匹配的部分。

当搜索结束后，将在输出窗口中显示匹配信息。例如，在一个菜单对象中搜索与字符串“menu”匹配的结果显示，如图 12-12 所示。

图 12-12　搜索结果显示信息

12.3.4　导入和导出对象

有时也许会希望将目标程序中的对象以文本文件的格式储存起来，以作为该对象的备份文件，有时也许会希望将对象以文本文件的格式从一台计算机转移到另外一台计算机上，在这种情况下，可以将对象信息以 ASCII 文本文件的格式导出，而所导出的文本文件将会包含对该对象定义的所有信息。在本质上，这些被导出的文件中所有对象的语法格式与被存储在库中的源代码的格式是相同的，因此不必担心在对象被导出的过程中有遗失信息的现象发生。

在将对象信息成功导出之后，用户还可以在需要的时候将该对象文本文件通过一定的操作移植到 PowerBuilder 的指定库中存储，这就用到了对象的导入功能。

注意： PowerBuilder 对象导入、导出功能的主要用途是用来导入、导出源代码，而不是用来对源代码进行修改。如果只是修改源代码，可以使用源代码编辑器直接对源代码进行修改，最好不要在所导出的 ASCII 文本文件中对源代码进行改动。

将对象导出的步骤是：

(1)　选中一个或多个要导出的对象。

(2)　在右键菜单中选择 Export 命令，或选择菜单栏上的 Export 图标按钮，或从菜单栏上选择 Entry Library Item Export 命令，打开 Export Library Entry 对话框，如图 12-13 所示。

(3)　在 Export Library Entry 对话框中可指定该对象的存储路径及导出文件名，注意不要修改 PowerBuilder 所指定添加的文件扩展名。

(4)　单击 OK 按钮保存导出文件。

(5)　重复步骤(3)和(4)，直到将所有选中对象全部导出为止。

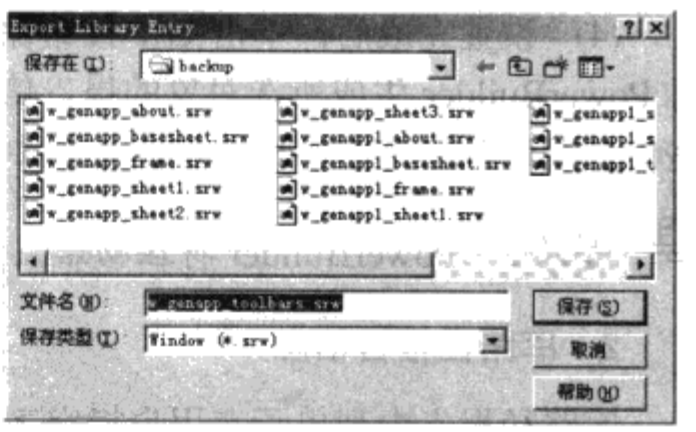


图 12-13　Export Library Entry 对话框

提示： PowerBuilder 导出文件的扩展名为 SRX，“X”是指该导出对象的对象类型。例如，扩展名 SRM 的导出对象类型为菜单对象(Menu)，扩展名为 SRW 的导出对象类型为窗口对象(Window)。

将对象保存信息导入的步骤是：

(1) 在系统目录树或库画板中选中要导入对象的库。

(2) 在该库上右击，在弹出的快捷菜单中选择 Import 命令，或在库画板工具栏上单击 Import 图标按钮，打开 Select Import Files 对话框。

(3) 在 Select Import Files 对话框中选择要导入的一个或多个文件。

(4) 单击 Open 按钮完成导入文件的操作。

注意： 当用户通过导入操作向一个库中导入与当前库中对象同名的存储信息文件时，旧信息将首先被删除，然后导入新的信息。但如果导入文件操作失败，旧信息也已经被删除，不可恢复。

12.3.5 生成库内容报告

在库画板中，可以生成 3 种库内容的报告，它们分别为：

- 搜索信息报告；
- 库对象信息报告；
- 库路径报告。

搜索信息报告的生成方法已经在前面做过介绍，这里就不再赘述，下面介绍其他两种报告的生成方法。

1. 库对象信息报告

库报告提供了当前目标程序中所选对象的信息，用户可以使用这些报告来获得所创建的目标程序对象的打印文档。创建库对象信息报告的方法是：

(1) 选中所需要获得信息报告的库对象。

(2) 从菜单栏上选择 Entry Library Item Print 命令，打开 Print Options 对话框，如图 12-14 所示。

(3) 在该对话框中为这些对象类型设置打印信息。

(4) 设置完成后，单击 OK 按钮保存设置，同时 PowerBuilder 生成所选对象的报告信息，并将这些信息发送到指定打印机的打印队列中。

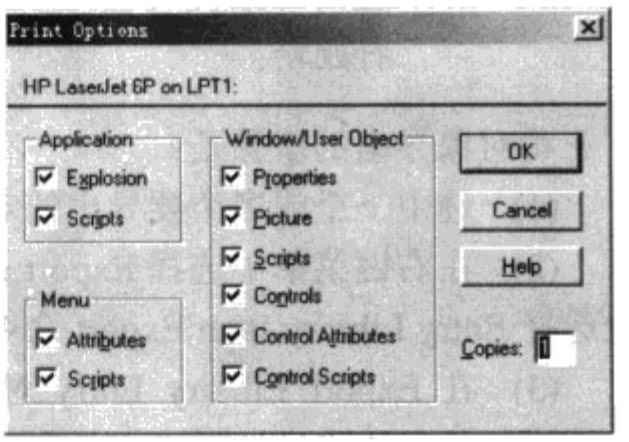


图 12-14 Print Options 对话框

提示： PowerBuilder 将在初始文件库中记录这些打印设置信息。

2. 库路径报告信息

库路径报告中列出了在用户指定工作空间下所有选中对象的路径信息，并按照对象名、修改日期及时间、编译对象的大小、注释等信息排列顺序。

创建库路径报告信息的方法是：

(1) 选中需要生成路径报告信息的库(该库必须在当前工作空间中)。

(2) 从菜单栏上选择 Entry Library Print Directory 命令，或从右键菜单中选择 Print Directory 命令。

(3)　在打开的对话框中单击 OK 按钮，PowerBuilder 将库路径信息发送到指定打印机的打印队列中。

12.4　根据已有的库文件创建目标

在实际的应用中，有时候需要将 PBL 库文件复制到其他的机器或者给其他的开发人员使用，这时就需要根据已有的库文件来创建目标，具体的操作步骤如下：

(1)　单击工具栏上的 New 按钮，打开 New 对话框，选择 Workspace 选项卡，如图 12-15 所示。

(2)　选中 Workspace 选项卡中的 Workspace 图标，然后单击 OK 按钮，打开 New Workspace 对话框，如图 12-16 所示。

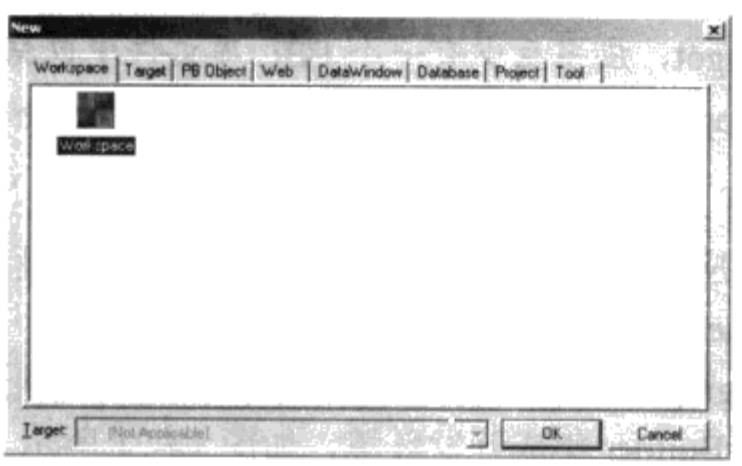


图 12-15　New 对话框的 Workspace 选项卡

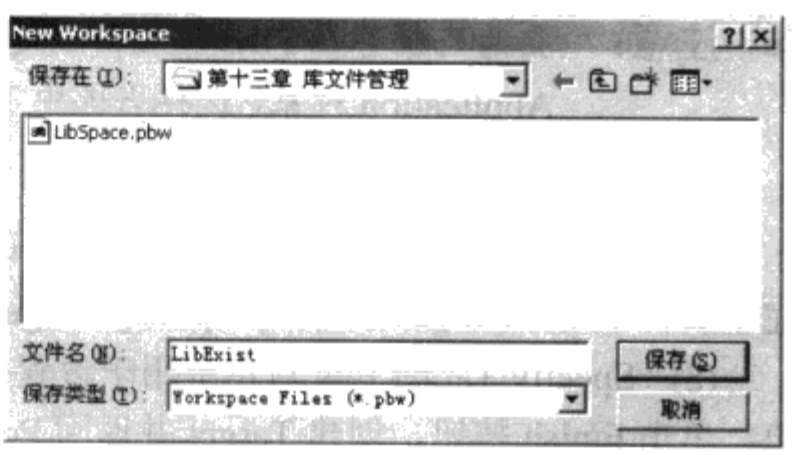


图 12-16　New Workspace 对话框

(3)　在 New Workspace 对话框中选择工作空间存储的路径，并命名为 Lib Exist。单击“保存”按钮，创建工作空间并返回 PowerBuilder 集成开发环境。

(4)　单击工具栏上的 New 按钮，打开 New 对话框，选择 Target 选项卡，如图 12-17 所示。

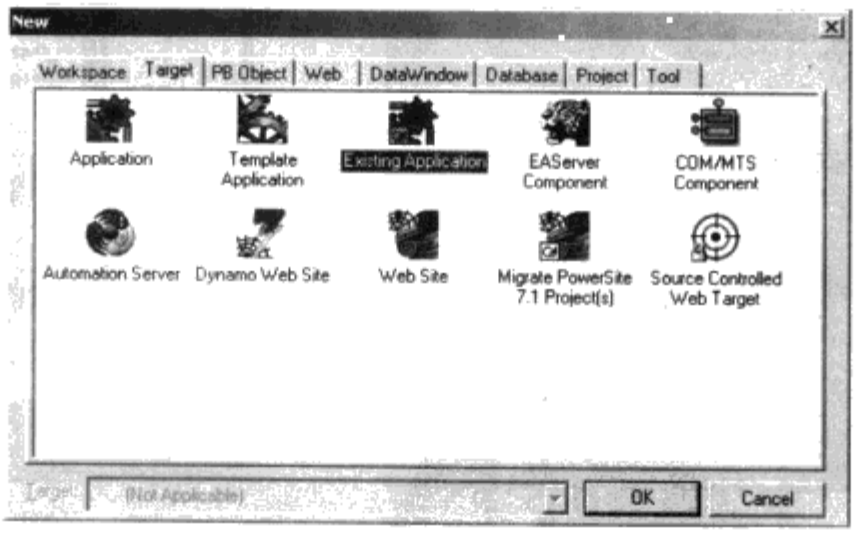


图 12-17　New 对话框的 Target 选项卡

(5)　在 Target 选项卡中选择 Existing Application 图标，然后单击 OK 按钮，打开 Choose Library and Application 对话框，如图 12-18 所示。

(6)　Choose Library and Application 对话框用于选择已存在的 PBL 库文件和 Application

对象。选择完成后单击 Next 按钮，打开 Set Library Search Path 对话框，如图 12-19 所示。

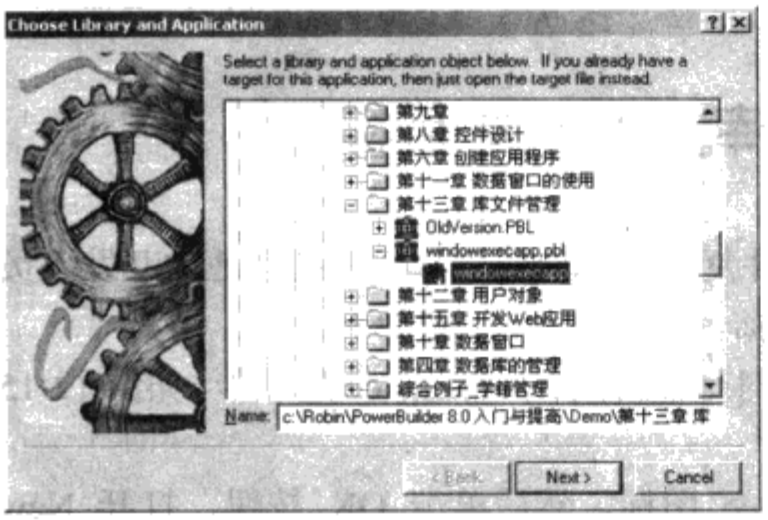


图 12-18 Choose Library and Application 对话框

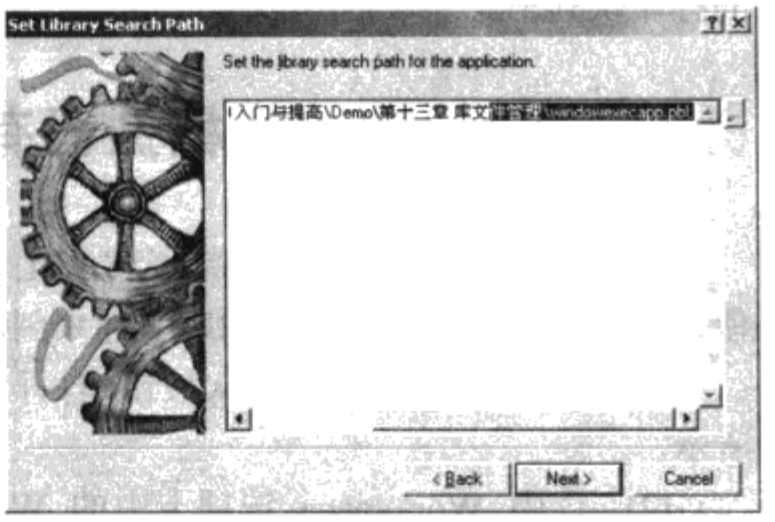


图 12-19 Set Library Search Path 对话框

注意： 要为已经存在 PBL 库文件创建 Target，应当至少有一个 PBL 库文件中包括 Application 对象，这样当选中了 Application 对象后，单击 Next 按钮时才会转到下一步。

(7) Set Library Search Path 对话框用于设置应用程序中 PBL 库文件的搜索路径，设置完毕后，单击 Next 按钮，打开 Specify Target File 对话框，如图 12-20 所示。

(8) Specify Target File 对话框用于设置 Target 文件的存储路径和名称，这里使用默认值，单击 Finish 按钮，创建 Target 并返回 PowerBuilder 集成开发环境，如图 12-21 所示。

提示： 如果 PBL 库文件的版本不是当前版本，系统会自动打开 Migrate Application 对话框用于移植 PBL 库文件。

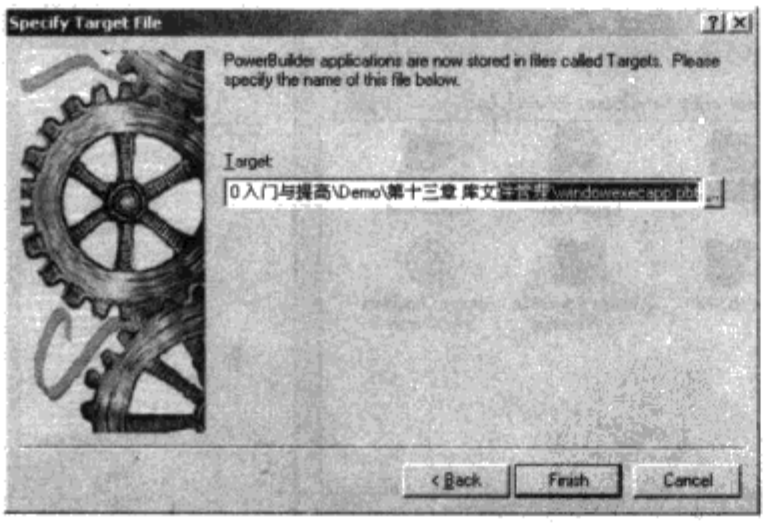


图 12-20 Specify Target File 对话框

图 12-21 新建立的 Target

至此，为一个已经存在的 PBL 库文件创建了工作空间和 Target，可以在新创建的工作空间中利用已有的 PBL 库文件开发应用程序。

12.5　习　　题

1.　填空题

(1)　存放在库中的对象有两种形式，包括_____和_____。

(2)　库画板有两个视图区，分别为_____和_____。

(3)　PowerBuilder 中的库画板是对库文件进行管理的工具，库文件的管理包括对库文件的_____、_____、_____、_____和_____等操作。

(4)　可以采用_____和_____两种方法对工作空间或目标程序进行重建。

2.　问答题

(1)　如何设置搜索路径？

(2)　试列举在库画板中可以进行哪些操作。

(3)　如何优化库文件？

第 13 章　程序调试与发布

当完成了应用程序各部分的创建并保存了这些对象后，可以运行应用程序检查运行的结果是否让人满意。在 PowerBuilder 10.0 中，有两种运行应用程序的模式，包括常规模式和调试模式。常规模式是最终用户所使用的一种模式，这种模式只有在用户中断或者发生错误时才会中止。在多数情况下，需要通过调试模式来修改应用程序中的错误和进一步完善应用程序的功能。因为该模式提供了多种跟踪调试手段，并可随时查看相关的变量值。程序调试无误后，编程者就可以发布程序以供实际使用。

本章将主要介绍调试环境、调试的方法以及程序的发布。

13.1　调试器概述

在调试器(Debugger)画板中完成调试应用程序，选择 File 菜单中的 Debug 命令，即可打开 Debugger 画板，如图 13-1 所示。

图 13-1　Debugger 画板

默认情况下调试画板工作区分为 4 个窗口，分别为查看脚本窗口、资源浏览窗口、查看变量窗口和断点查看窗口，每个窗口包含一个或者多个视图。

1.　查看脚本窗口

该窗口位于屏幕的左上角，其中包含 Source 视图。在窗口中，可以查看选定对象的脚本并设置断点。

2.　资源浏览窗口

该窗口位于屏幕的右上角，用于选择要跟踪调试的对象。在窗口中显示了该应用程序中所有的对象。双击任意一个对象，可打开其中的子对象或者函数，再双击任意一个函数，该函数将显示在查看脚本窗口中。该窗口包含 Source Browse 和 Source History 视图。

3.　查看变量窗口

该窗口位于屏幕的左下角，用于查看应用程序运行时各变量的变换情况。该窗口包含 Instances、Objects in Memory 等视图。

4.　断点查看窗口

该窗口位于屏幕的右下角，用于观查断点的状态和已经查看过的函数或变量列表。该窗口包含 Call Stack、Breakpoints 等视图。

在调试应用程序中使用断点的一般步骤如下：

(1) 打开调试画板。

(2) 在程序中设置断点。

(3) 在跟踪模式下运行应用程序。

(4) 当应用程序在断点处中断执行时，可以查看变量的值、改变变量的值或检查内存中对象的属性。

(5) 逐行调试应用程序。

(6) 需要时在应用程序中增加新的断点或者修改已存在的断点。

(7) 当用户找到问题时，修改有问题的代码并重新在调试模式下运行该应用程序。

以上给出的只是调试的一般步骤，可根据实际情况灵活进行。由于实际调试的多样性，下面将不按照以上步骤逐步介绍，只是对调试中常用的几个关键环节进行介绍。包括设置、激活和删除断点，单步跟踪以及查看变量值及其他相关信息等。

13.2 断 点 操 作

调试过程中，断点是应用程序代码某处的一个圆点标记，用来中断一个应用程序的正常执行过程。若检查某段脚本、函数调用过程是否有错误，可在脚本的开始处、函数调用处设置一个断点，通过调试来查看或者修改错误。

下面从调试程序的切换开始，讲解断点的设置、清除等方法。

13.2.1 调试应用程序的切换

调试过程中，可选择某一对象的脚本进行测试，这就涉及到调试程序的切换。切换调试程序可在 3 种视图中完成，即 Source、Source Browse 和 Source History 视图。

1. 使用 Source 视图

Source 视图显示应用程序脚本内容。若设置断点，也将显示断点的位置和脚本执行的位置。在 Source 视图中还可以对调试的程序进行切换。

具体操作步骤如下：

(1) 在 Source 视图中，打开 Edit 菜单，选择 Select Script 命令，弹出 Select Script 对话框，如图 13-2 所示。

(2) 在对话框中选择需要调试的对象。还可以从 Call Stack、Source Browse 或者 Source History 视图中将需要调试的程序名直接拖到 Source 视图中。

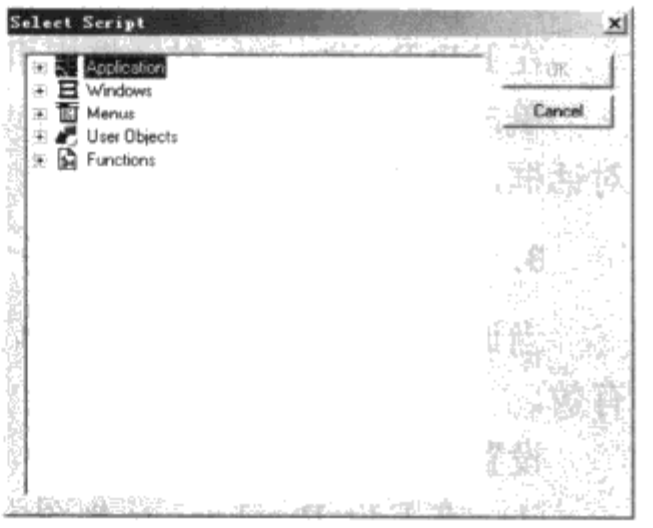


图 13-2　Select Script 对话框

2. 使用 Source Browse 视图

Source Browse 视图以树形结构显示应用程序中的所有对象，树形结构提供了一种查看应用程序结构和快速切换调试程序的方法。

切换程序的操作步骤如下：

(1) 在 Source Browse 视图中，双击对象名或者单击对象名前的加号，可展开树形结构；在展开的结构中选择要调试的对象或者事件。

(2) 双击所选事件或者用鼠标右击所选事件，然后从弹出的菜单中选择 Open Source 命令，则在 Source 视图中显示所选对象的脚本。

3. 使用 Source History 视图

Source History 视图列表显示所有在调试过程中打开过的脚本名，切换调试程序的方法与 Source Browse 视图的切换方法相同。

13.2.2 设置断点

按照断点的不同用途，可以将断点分为简单断点和特殊断点。

- 简单断点主要用于一般程序的调试，其触发条件是程序执行到断点处。
- 特殊断点主要用于一些比较复杂的程序或者变化细微的程序，其触发条件可以是执行的次数、表达式的结果或变量值的改变。

1. 设置简单断点

设置简单断点的步骤如下：

(1) 选择所要调试的应用程序。

(2) 在 Source 视图中，将光标置于欲设置断点的行。

(3) 双击该行或者用鼠标右键单击该行并从弹出菜单中选择 Add Breakpoint 命令，则在设置断点的行前显示一个红的实心圆点，如图 13-3 所示。

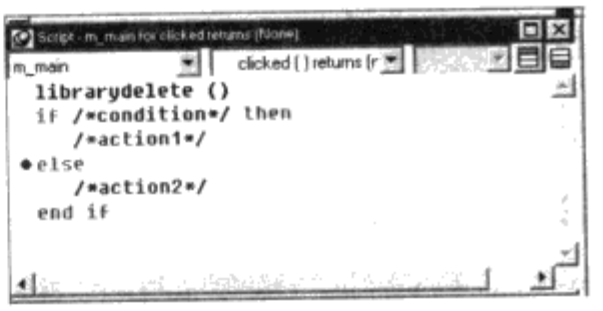


图 13-3 设置断点

2. 设置特殊断点

特殊断点主要分为临时断点、条件断点以及变量断点等。特殊断点可以在如下 3 种情况下触发：

- 当一段循环程序执行到指定次数时，可触发临时断点。
- 当一个条件表达式为真时，可触发条件断点。
- 当一个变量值改变时，可触发变量断点。

在 Edit Breakpoints 对话框中可以设置特殊断点，步骤如下：

单击工具栏中的 Edit Stop 图标或在 Edit 菜单中选择 Breakpoints 命令，打开 Edit Breakpoints 对话框，如图 13-4 所示。

3. 断点的无效和有效

当调试过程中需要跳过某一断点时，可将断点设置为无效，若不跳过，可将其设置为有效。

设置断点为无效的步骤如下：

(1) 在 Edit Breakpoints 视图中单击断点标记(红色实心圆点)，这断点标记变为空心圆点，如图 13-5 所示。

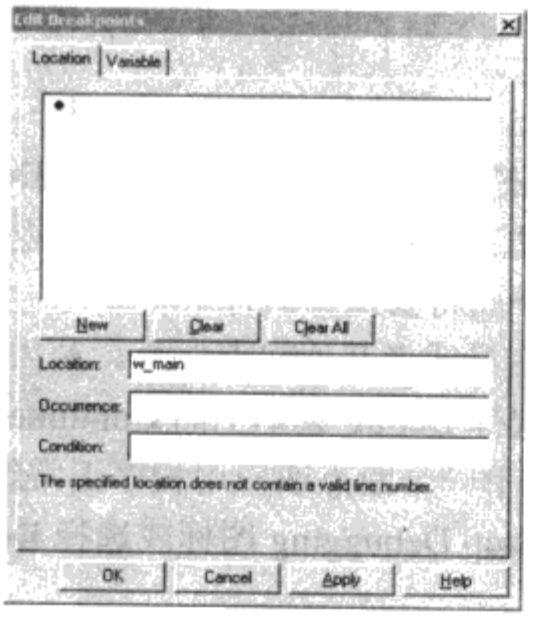


图 13-4　Edit Breakpoints 对话框

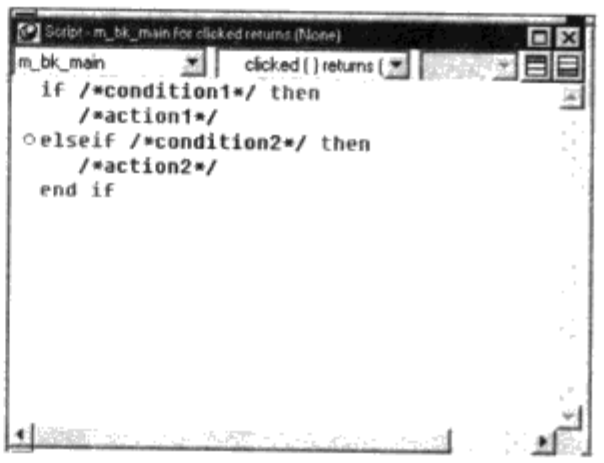


图 13-5　将断点设为无效

(2) 在 Source 视图中选中相应语句，用鼠标右击该语句后，在弹出的菜单中选择 Disable Breakpoint 命令。

(3) 在 Edit Breakpoints 视图中选中相应语句，用鼠标右击该语句后，在弹出菜单中选择 Disable 命令。

以上方法只能逐一将断点设为无效，若将断点全部设为无效，可在 Edit Breakpoints 视图中单击鼠标右键，在弹出的菜单中选择 Disable All 命令。

如果断点设置重新生效，可在相应的视图中用鼠标右击，在弹出菜单中选中 Enable 或 Enable All 命令，也可直接单击断点标记。

4. 清除断点

当调试完成之后或者不再需要某些断点时，可以将断点清除。清除断点的操作方法如下：

(1) 在 Source 视图中双击包含断点的行，则断点标记消失。

(2) 在 Source 视图中用鼠标右键单击相应语句，从弹出菜单中选择 Clear Breakpoints 命令。

(3) 在 Breakpoints 视图中用鼠标右键单击相应语句，从弹出菜单中选择 Clear 命令。

(4) 打开 Edit 菜单，选择 Breakpoints 命令，在打开的 Edit Breakpoints 对话框中选中要清除的断点，单击 Clear 按钮。

如果需要一次清除所有断点，可在 Edit Breakpoints 对话框中单击 Clear All 按钮，也可以在 Breakpoints 视图中单击鼠标右键，从弹出菜单中选择 Disable All 命令。

13.3　调 试 方 法

按照以上方法设置断点后，就可以在调试模式下运行应用程序。应用程序将按正常的模式执行到断点处。在查看了相关的信息后，可使用单步跟踪的方法仔细地观察应用程序的执行过程或停止当前的调试或关闭调试画板。

13.3.1 调试的开始、继续和停止

当选择了调试程序并设置了断点后，可单击工具栏上的 Start 图标或者选择 Debug 菜单的 Start 命令，启动调试任务。应用程序开始运行，直到遇到一个断点为止。在返回到调试画板后，将显示包含断点的行，并用黄色箭头指示下面将要执行的行。

此时，可在调试画板中的各种视图中查看各种变量或者其他状态信息。若需要应用程序继续执行，可单击工具栏上的 Continue 图标或者选择 Debug 菜单上的 Continue 命令。调试程序将从黄色箭头指示的语句继续执行，直到遇到下一个断点或应用程序正常结束为止。

若需要结束当前调试任务，可单击工具栏上的 Stop Debugging 图标或选择 Debug 菜单上的 Stop 命令。

调试结束后，可重新设置应用程序的状态并使调试的所有视图恢复到调试初始状态。

13.3.2 其他调试方法

上面介绍的调试方法是最常用的调试程序方法。此外，PowerBuilder 10.0 还提供了多种单步调试方法，用于观察程序细微变化和每步执行的结果。当调试程序停在断点处时，可使用几种调试方法来单步跟踪调试程序的执行情况。此时，调试画板的各个视图将随着每步的执行而改变，并用黄色箭头指示程序执行的位置。

1. 单步跟踪至函数内部(Step In)

调试程序将执行到函数内部，并检查函数中每条语句执行情况。

2. 单步跟踪并跨越函数(Step Over)

调试程序只将函数作为一步处理，并不跟踪至函数内部。

3. 在函数内部一次执行完(Step Out)

对于已经跟踪至函数内部并需要结束函数内部调试的情况，可用 Step Out 命令继续执行直至函数返回。

4. 执行至指定的光标处(Run To Cursor)

对于大的循环或确认无误的代码段，使用此调试方法可跳过该段程序代码或执行到下一个断点处。

5. 设置下一条语句(Set Next Statement)

对于包含错误的代码段或者用于测试的代码段，可用 Set Next Statement 跳过并从指定的语句执行。

以上几种调试方法的使用非常简单，只需在某一个断点处中断后，根据需要选择以上调试方法，单击工具栏上的相应图标即可，逐步执行调试代码并观察执行结果。

13.4　查 看 信 息

当调试程序在某个断点中断时，可在 Variable、Watch、Call Stack 和 Objects In Memory 视图中查看各种状态信息，如各种变量值、表达式、内存中的对象和函数列表等。

13.4.1　查看各种变量值

变量视图可显示一种或者多种变量列表。双击变量名或者单击变量前的加号或者减号可展开或折叠所选择的变量。默认情况下，一种变量显示在一个变量视图中，视图所代表的含义如下。

- Local：当前程序或函数的局部变量值。
- Global：应用程序定义的所有全局变量值和所有打开对象的属性值。
- Instance：当前对象的属性值和当前对象定义的实例变量值。
- Parent：当前实例祖先的属性值。
- Shared：已打开对象的共享变量值。

在变量视图中，不仅可以查看各类变量值，还可以修改这些变量的值，以便观察修改后的执行结果。

修改变量的步骤如下：

(1)　选择要修改的变量视图和变量。以 Instance 实例变量视图为例，如图 13-6 所示。

(2)　双击所选变量，弹出如图 13-7 所示的 Modify Variable 对话框。

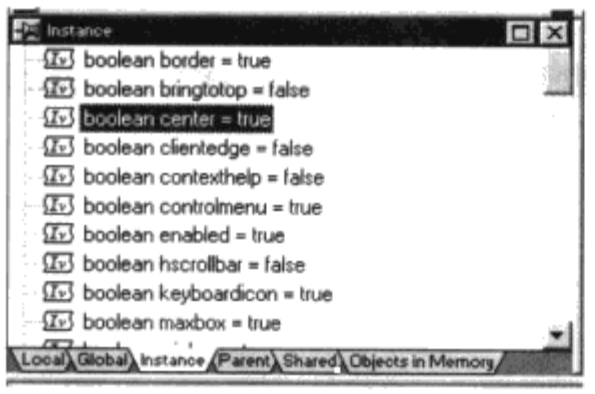


图 13-6　Instance 视图

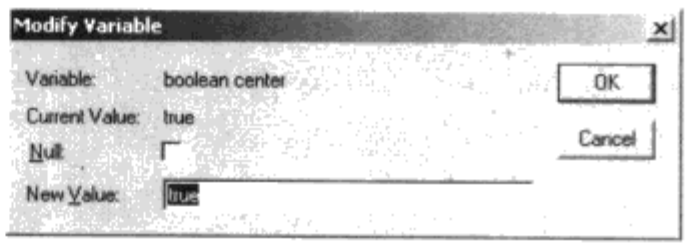


图 13-7　Modify Variable 对话框

(3)在 New Value 文本框中输入新的变量值后，单击 OK 按钮。

13.4.2　查看其他信息

可查看各种变量的信息，也可查看表达式、内存中的对象和函数列表。下面介绍查看这些信息的方法。

1. Watch 视图

Watch 视图允许在应用程序运行时监视所选变量和表达式的变化情况。当变量和表达式在调试范围内时，可在 Watch 视图中显示变量或者表达式的值，如图 13-8 所示。

在 Watch 视图中添加表达式，可在该视图中用鼠标右击，在弹出的菜单中选择 Insert

命令，然后在如图 13-9 所示的 New Expression 对话框中输入表达式。

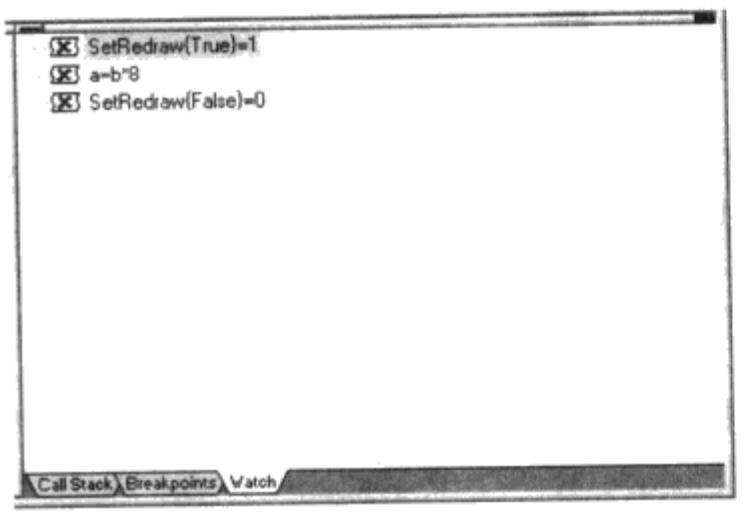


图 13-8　Watch 视图

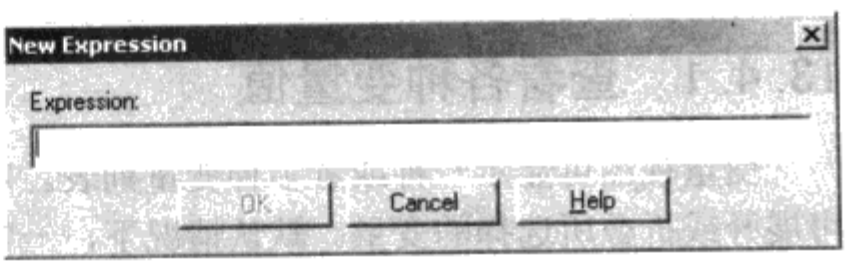


图 13-9　New Expression 对话框

2. Call Stack 视图

Call Stack 视图给出了调试程序调用函数的列表或停在断点处的函数。黄色箭头指示当前中断的程序和行号，绿色箭头指示当前 Call Stack 视图中所显示的程序。可随时查看 CallStack 视图中任意行的相关程序，Call Stack 视图与 Source 视图相结合观察更有效，Call Stack 视图如图 13-10 所示。

3. Objects in Memory 视图

Objects in Memory 视图显示当前内存中的对象列表。双击对象名或单击对象前的加号，可查看对象的实例变量名和内存位置以及实例变量的属性值，Objects in Memory 视图如图 13-11 所示。

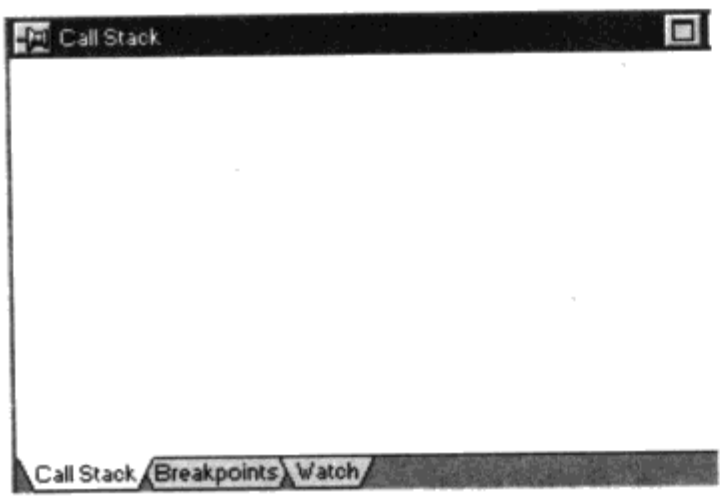


图 13-10　Call Stack 视图

图 13-11　Objects in Memory 视图

13.4.3　运行错误解释

当程序代码还未编写完成时，运行程序会出现错误，同时触发 SYSTEM ERROR 事件。如果没有为错误事件编写脚本，系统不会自动显示错误提示框。内容包括错误消息的数目和文本。它是调试程序修改错误的重要依据。这些错误信息的具体含义见表 13-1。

表 13-1　运行错误解释

错误号	含　义
1	被零除
2	引用 NULL 对象
3	数组越界
4	枚举值超出函数范围
5	函数中遇到负值
6	指定的数据窗口行/列无效
7	链接引用时，出现不可解决的外部引用
8	引用带有 NULL 下表的数组
9	DLL 函数未在当前程序中
10	DLL 函数不支持的参数类型
11	对象文件已过期，必须转化为当前版本
12	数据窗口列类型与 GETITEM 类型不匹配
13	不可知的属性引用
14	为外部函数打开错误 DLL 库
15	错误调用外部函数
16	超出最大字符串的长度
17	数据窗口控件中引用的数据窗口对象不存在
18	函数未返回值
19	不能将任意变量转化为 NAME 类型
20	没有准备好数据库命令
21	运行时，错误地引用函数
22	不可知的对象类型
23	不能将 NAME 类型的对象赋值为 NAME 类型的对象
24	函数调用与其定义不匹配
25	DOUBLE 或 REAL 表达式溢出
26	不支持对 NAME 字段的赋值
27	非整型指数不能为负
28	VBX 错误：NAME
29	ANY 变量需要非整组值
30	外部对象不支持数据类型：NAME
31	不支持外部对象数据类型：NAME
32	调用外部对象函数 NAME 时，未找到名字
33	调用外部对象函数 NAME 时，参数类型不合法
34	调用外部对象函数 NAME 时，使用不正确的参数数目
35	错误调用外部对象函数 NAME

续表

错误号	含　义
36	访问外部对象属性 NAME 时，未找到名称
37	访问外部对象属性 NAME 时，类型不匹配
38	询问外部对象属性 NAME 时，下标数目不正确
39	错误访问外部对象属性 NAME
40	与表达式中任意数据类型不匹配
41	表达式中有不合法的任意数据类型
42	在 DLL 函数 NAME 中，运行时所指定的参数类型与所要求的参数类型不相同
43	父对象不存在
44	祖先的函数中具有发生冲突的参数及返回类型
45	外部表溢出
46	空的对象引用不能被赋值或传递给同类型的变量
47	变量需要数组值
48	数组的大小与对象的转换不匹配
49	数组的类型与对象的转换不匹配
50	分布服务错误
51	函数或事件的参数不合法
52	分布通信错误
53	未激活所需服务器
54	服务器未接受请求
55	请求非正常终止
56	非完整地相应请求
57	未连接
58	对象实例不存在
59	不合法的列范围
60	不合法的行范围
61	将 N 维数组不合法地转换成对象
62	服务器忙
63	无返回值的函数或事件用于表达式
64	赋值需要数组对象

注意： 第 50 号错误会立即中断应用程序，但并不触发 SYSTEM ERROR 事件。

13.5 发布应用程序

完成应用程序的编程和调试工作以后，剩下的工作就是应用程序的编译和发行，这涉及到工程对象的创建、可执行应用的生成以及发行环境的配置等问题。一个应用程序成功

与否，归根到底在于最终用户的使用情况。由于 PowerBuilder 是一个数据库应用开发工具，用它开发的应用程序通常要与数据库连接，因此涉及到数据库环境配置问题。在发布应用时并不能将开发环境发布出去，最好的方法就是将应用程序编译成可执行文件，只发布应用程序运行所需的环境，这样既可以减小应用程序的大小，又可提高应用程序的运行速度。

下面就针对以上问题，从创建工程对象开始，说明如何将应用程序编译成可执行文件并发布给最终用户。

13.5.1　概述

在 PowerBuilder 中，可以创建多种类型的应用。对于传统的客户/服务器应用，需要创建一个可执行的应用分发给用户。若用户创建的是分布式应用，则需建立一个客户端可执行文件，并为事务服务器建立一个服务器端可执行文件或者服务器组件。此外，还可以建立一个代理对象、HTML 文件或 Java 类。

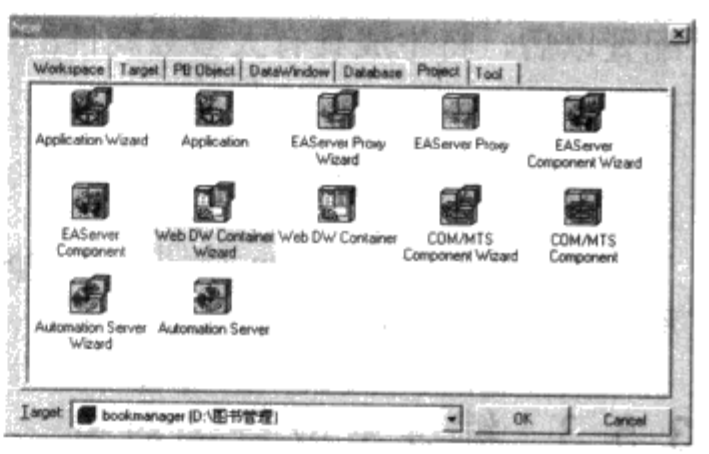


图 13-12　New 对话框中的 Project 选项卡

在 PowerBuilder 中，可使用工程画板来完成以上工程对象的创建和维护。大多数工程对象可通过向导完成。选择 File 菜单中的 New 命令打开对话框，选择 Project 选项卡，显示了多种工程对象的向导，如图 13-12 所示。

针对图 13-12 中各种工程对象，表 13-2 列出了其中较为常用的工程对象的功能以及是否支持向导。

表 13-2　工程对象

工程对象名	用　途	向　导
Application	生成一个可执行的文件和可选的动态链接库	支持
Automation Server	可被 COM 客户端通过一个定制类用户对象访问	不支持
COM/MTS Component	生成一个 COM 服务器，由 MTS 创建并可被 COM 客户端通过一个或者多个定制类用户对象访问	支持

下面介绍工程对象的创建过程，并着重说明如何将一个应用程序编译成一个可执行应用程序。

13.5.2　建立工程对象

由于工程对象的种类较多，所以创建的过程也不相同。但通过向导创建的过程是相同的，不同的仅是所设置的参数，这需要根据实际情况进行选择。

下面介绍一般的步骤和操作方法，并详细说明应用工程对象。

使用模板向导创建一个新应用程序的同时也创建了一个工程对象，而使用 EAServer、COM/MTS 自动服务器组件既可以通过开始向导(New 对话框中的 Start Wizards 选项卡)，也可以通过对象向导(New 对话框中的 Object 选项卡)创建。

在 Project 选项卡中有两类图标。

- 一种图标用于打开模板向导，通常在对应的图标名称中包含 Wizard 字样，便于设置工程对象的各项参数。
- 另一种图标直接打开工程画板，便于在如图 13-1 所示的 Debugger 画板中进行设置和编译。

以下介绍如何创建和打开工程对象。

1. 创建工程对象

(1) 单击工具栏上的 New 按钮，打开 New 对话框。

(2) 在 New 对话框中选择 Project 选项卡。

(3) 选择所需的工程对象，单击 OK 按钮。若选择一个模板向导，在完成了向导的各种配置以后，将生成具有大多数属性的工程对象，还可以打开工程对象对属性进行修改。若未选择模板向导，可打开指定对象的工程画板，在其中指定工程对象的属性。

2. 打开工程对象

单击工具栏上的 Open 按钮，在 Open 对话框中选择 Project 选项卡，并选择要打开的工程对象，最后单击 OK 按钮。

13.5.3 编译生成各类对象

无论使用向导还是使用工程画板生成工程对象，在完成各项属性的设置以后，均可编译生成所需的各类对象。

选择应用类型的工程对象，打开工程画板，根据需要直接修改各项参数。

打开工程画板后，单击工具栏上的“编译”按钮编译所生成的对象。编译成功，将生成应用的可执行文件。单击 Save 按钮，可保存此工程对象。

若选择其他类型的工程对象，则打开的工作区是只读的并显示相关选项。下面以 EAServer Proxy 对象为例，打开的画板如图 13-13 所示。

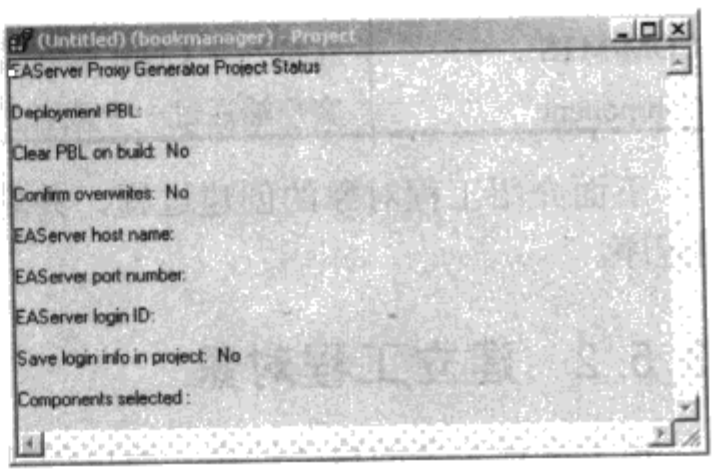


图 13-13 EAServer Proxy 对象的工作画板

在图 13-13 中，工作区是只读的，对于不同的工程对象，其工作区所显示的内容也不相同。编译生成的过程与应用工程对象也有一些区别，具体步骤如下：

(1) 选择 Edit 菜单中的 Select Object 命令或单击选择对象，在弹出 Select Objects 对话框中选择所需要的对象。

(2) 选择 Edit 菜单中的 Properties 命令或单击“属性”按钮，指定或修改工程对象的属性。

(3) 选择 Design 菜单中的 Build 命令或者单击编译“属性”按钮，编译生成所需的 EAServer Proxy(组件或其他)对象。

13.5.4　编译生成可执行应用

如何把开发好的应用程序编译成可执行文件并成功地发行给最终用户在整个开发过程中是一个很重要的环节。下面将详细说明如何将应用程序编译成可执行应用。

1．编译参数的介绍

当建立了应用性工程对象后，必须配置工程对象的各参数，打开工程画板，如图 13-14 所示。配置各项参数，各参数的说明如表 13-3 所示。

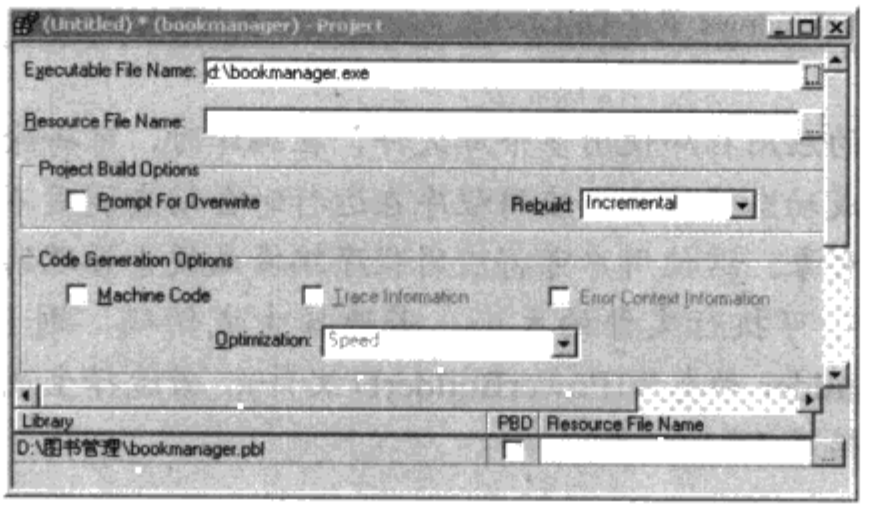


图 13-14　工程画板工作区

表 13-3　工程画板中的参数

参　数	用　途
Executable File Name	指定可执行文件名。这个文件名必须符合 Windows 平台的 EXE 命名规则。若改变文件保存的目录，可单击“浏览”按钮进行选择
Resource File Name	指定资源文件名。在脚本中动态地引用位图、图标等资源，并且需要资源包括在可执行文件中而不是单独分发这些资源，可将这些资源做成 PowerBuilderR 文件。既可以直接输入资源文件名，也可以单击“浏览”按钮进行选择
Prompt for Overwrite	指定覆盖文件时是否提示。当编译用户应用时，将覆盖由它创建的所有文件
Rebuild	在生成可执行文件和动态链接库时，决定 PowerBuilder 重生成库文件中全部对象的方式，包括两个选项：Full 表示重新生成全部应用；Increament 表示仅生成新增部分应用。如果选择 Increament，则 PowerBuilder 将只生成改变的对象和被其他(改变)对象引用的对象。在多数情况下，用户在重编译用户工程对象之前，要重生成用户的对象
Maching Code	指定是否生成机器码。编译生成机器码而不是伪码(Pcode)，则选中该复选框。必须在工程画板中设置机器码和其他代码生成选项，而不是在向导中完成
Trace Information	指定是否生成跟踪文件。当用户运行编译好的可执行文件时，选择此复选框可生成跟踪文件。用户可利用跟踪文件来分析应用出现故障的原因
ErrorContext Information	指定是否显示错误信息。选中该复选框时，发生运行错误时，PowerBuilder 能显示(如对象名、事件名和脚本行号)错误信息

续表

参 数	用 途
Optimization	选择编译时优化的级别，包括如下选项：Speed 表示速度优化；No Optimization 表示不优化
Resource File Name	为动态链接库所指定的资源文件名
PowerBuilderD 或 DLL	这个复选框依赖于选择的是机器码还是伪码。选中库存文件前的复选框，把库文件编译成一个动态链接库；若选择生成伪码，则生成 PowerBuilder 动态库文件(PowerBuilderD)，若选择生成机器码，则生成平台指定的动态库文件(在 Windows 下是 DLL)

注意： 若用户的应用程序使用多个库文件，在编译时，可将除应用对象之外的库文件编译成动态链接库，应用程序在运行时会动态调用不包含在可执行文件中的其他对象。这使用户可将应用程序编译成便于管理的几个小单元，同时也可以减小可执行文件的大小。若选择生成伪码，则 PowerBuilder 将生成 PowerBuilder 动态库(PowerBuilderD 文件)；若选择生成机器码，则编译生成动态链接库(DLL 文件)。

技巧： 将一个库文件的对象分解成几个库文件的操作可在管理画板中完成。完成分解工作之后，在工程画板中选中库文件名前的 PowerBuilderD 或者 DLL 复选框，选中的库文件将编译成动态链接库。若此动态链接库需要资源文件，则可单击浏览按钮进行选择。

下面介绍如何创建资源文件以及打包发行的方式。

2. 资源文件的创建

应用中用到的各种图标、图片和光标在编译时，并不会自动编译成可执行文件的一部分，但在运行时却会自动检查这些资源是否可用。可通过两种方法解决这个问题。

(1) 单独分发资源

若一个资源包括在可执行文件或者动态链接库中，则应用在执行时将在搜索路径中查找此资源，在发布时，需要将应用所需的资源分发到最终用户的搜索路径中。

例如，在如下脚本中使用两个图片 Logon.bmp 和 Logoff.bmp：

```
IF Balance < 0 THEN
P_logo.pictureName = logon.bmp
ELSE
P_logo.PictureName = logoff.bmp
END IF
```

在分发此应用时，必须将这两个图片随同应用一起分发出去。只要这两个文件在搜索路径中，应用在执行时就会自动装入。这里所指的搜索路径包括：

- 当前目录
- Windows 目录
- Windows 系统目录

- 所有由 PATH 指定的目录

(2) 使用资源文件

也可通过 PowerBuilder 的资源 PowerBuilderR 文件来发行资源，而不是单独发行。资源文件是 ASCII 码的文本文件，列出了应用程序所需要的资源(位图、图标等文件)所在的位置及其文件名。在建立可执行文件或动态链接库时，可根据资源文件中搜索所需要的资源，并将它们连入可执行文件或动态链接库中。

创建和使用资源文件的步骤如下：

(1) 使用任意一种文本编辑器(如记事本)新建一个文本文件，在此文件中列出所有资源名，如.bmp、.cur、.ico、.rle 和.wmf 等，要求每项资源列为一行。如图 13-15 所示是一个资源文件的内容。若资源不在当前路径下，则应给出完整路径。

(2) 在工程画板中为可执行文件和动态链接库指定所需的资源文件名。

当一个诸如图片的资源被应用在执行时引用，则首先在可执行文件搜索；若搜索失败，则在应用定义的 PowerBuilderD 文件中查找；若再次失败，则在文件的搜索路径中寻找。

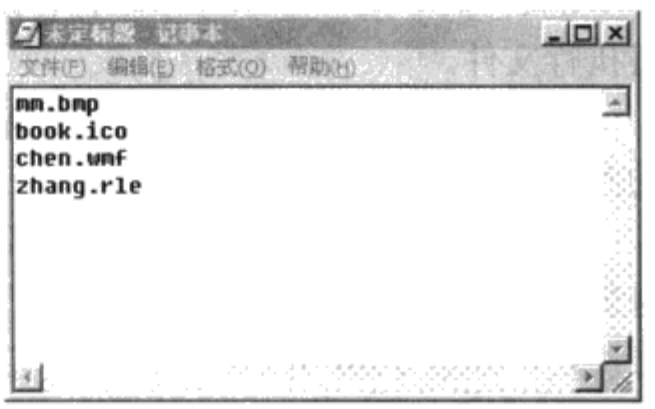


图 13-15　资源文件

3. 选择打包方式

下面讲述如何选择编码格式、将哪些内容打包发布以及选择何种打包模型。由于这些问题关系到发布的成功与否，所以有必要进一步说明。为使对打包方式有一个整体的了解，分别介绍。

(1) 编码格式

当决定发布一个应用时，首先要决定应用生成代码的格式，因为 PowerBuilder 提供两种选择，伪码(Pcode)和机器码(Machine Code)。

① 伪码

伪码是所有支持 PowerBuilder 平台的一种解释性语言，封装在伪码中的各种对象与其在库文件中具有相同的格式。伪码最大的优点是方便性和可移植性。

② 机器码

PowerBuilder 中的机器码同样也可生成和编译成可执行代码或动态链接库。机器码的优点是执行速度快。

两种编码格式各有优点，可根据如下原则进行选择。

- 速度：如果首要目标是优化执行速度，同时应用也要求加强脚本的处理，则应选择机器码。若应用的代码大量地使用循环、浮点运算或函数调用，则使用机器码会比伪码执行的更快。伪码同样也有速度优势，即编译生成要比机器码快，这对于快速生成测试用途的可执行应用是非常方便快捷的。
- 大小：伪码生成的的文件要比机器码生成的文件小。在应用分发的机器中，若文件大小是主要问题，则应该放弃机器码的速度而选择伪码。

- 可移植性：伪码适用于跨平台的应用，可将同一 PowerBuilderD 文件分发给使用 Windows 和 Unix 平台的用户。但是在不同的平台上，还需生成各自的可执行文件。机器码只能用于指定的平台，因此必须为每个平台生成一套可执行文件和动态链接库。

具体选择何种编码格式，可根据运行的环境、应用的文件大小等多种情况综合考虑。

(2) 打包文件

无论选择哪种编码格式，在 PowerBuilder 中生成的应用都将包括如下内容，可执行文件、动态链接库和资源文件。

① 可执行文件。若只是创建一层或两层应用并分发给最终用户，则生成可执行文件(EXE)文件而不是一个服务器组件；若创建一个分布式应用，则分别生成客户端和服务器端可执行文件。

一个最小可执行文件需要包括能使应用正常运行的代码，最终用户可双击可执行文件的图标来运行此应用。

② 动态链接库。为了不使全部应用打包成一个大的可执行文件，可将应用中的一些对象打包成一个或多个动态链接库文件。所生成的动态链接库文件要根据所选的编码格式来决定，如表 13-4 所示。

表 13-4 编码格式

编码格式	生成的动态库文件
机器码	在 Windows 下生成扩展名为 DLL(动态链接库)的文件。这些动态库文件与其他标准的共享库文件类似，惟一不同的是这些动态库不能被外部程序调用
伪码	生成的 PowerBuilderD(PowerBuilder 动态库)文件，这些动态库文件与应用执行时链接的 DLL 类似，但并不能与 DLL 互换，因为它们内部有不同的格式。在发行同一应用时不能同时使用两种动态库(DLL 和 PowerBuilderD)文件

动态库文件并不包括任何开始代码，它们不能独立执行，而只能在应用执行需要某对象时，才会在动态库中查找。相反，应用的开始代码只能编译成可执行文件，并通过双击该图标启动。

③ 资源文件。当使用资源时，必须将这些资源作为应用的一部分发布，一个 PowerBuilder 应用可使用多种类型的资源，并且不同的对象使用不同的资源，如表 13-5 所示。

表 13-5 资源类型

对象名	可使用的资源
窗口和用户对象	图标(ICO 文件) 图片(BMP、GIF、JPEG、RLE 和 WMF 文件) 光标(CUR 文件)
菜单(在 MDI 窗口中)	图片(BMP、GIF、JPEG、RLE 和 WMF 文件)
数据窗口	图片(BMP、GIF、JPEG、RLE 和 WMF 文件)

(3)　打包模型

为帮助用户正确选择的打包模型，下面给出几种常用的打包模型，可根据实际情况选择。

①　一个单独的可执行文件

在此模型中，所有对象和资源都包括在可执行文件中。发布时可只发布一个文件，范例模型如图 13-16 所示。这种模型适用于简单小巧的应用，是最简单的一种模型。

②　一个可执行文件和资源文件

在此模型中，所有对象和大多数资源包括在可执行文件中，发布时除要发布可执行文件外，还需为某些特定的资源发布一些文件，范例模型如图 13-17 所示。

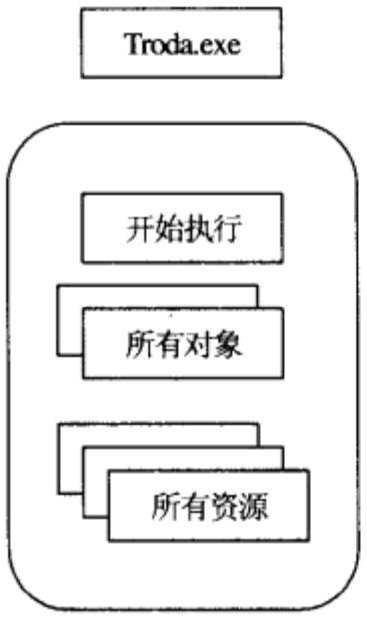


图 13-16 单独可执行文件

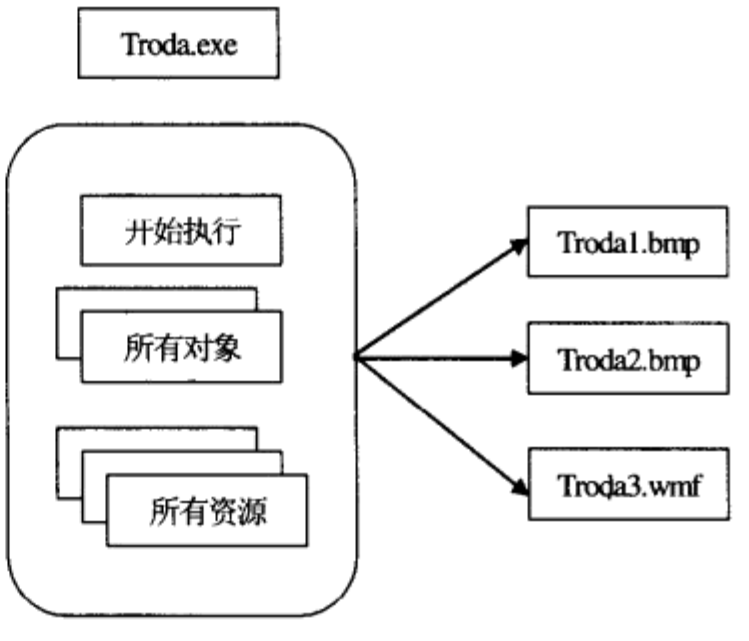


图 13-17　可执行文件和资源文件模型

这种模型同样适用于简单小巧的应用。但不同于前一种模型的是它对于维护经常改变的资源是非常方便的。也就是说，只对软件修订所需的资源发布一些文件，而不必再次发布可执行文件。另外，还可使用此模型来处理某些大型且不经常使用的资源或必须被其他应用共享的资源。

③　一个可执行文件和动态库文件

在此模型中，应用被分成可执行文件和一个或多个动态库文件(DLL 和 PowerBuilderD)。当使用此模型时，可以选择不同的方法来组织对象和资源，如表 13-6 所示。

表 13-6　组织对象和资源的方法

对象类型	组织方法
资源	将大多数或全部资源打包进动态库，或者将大多数或全部资源打包进可执行文件
对象	将全部对象打包进动态库或将少量经常访问的对象打包进可执行文件，而将其余的对象打包进动态库

范例模型如图 13-18 所示。这种模型适用于大多数应用，因为它赋予用户在组织、发布和维护上更大的灵活性。它允许用户修正应用系统的个别部分(如打包在动态库中的部分)并单独发布，而不必再次分发应用的其他部分。

④　一个可执行文件、动态库文件和资源文件

此模型除发布包括在可执行文件、动态库文件中的资源外，还需单独为某些特定的资源发布一些文件，如图 13-19 所示。

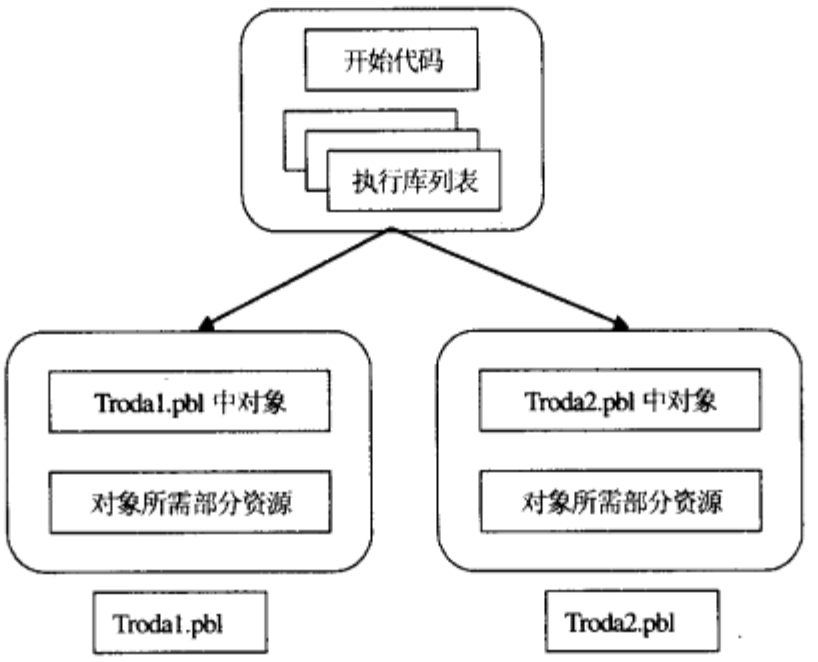


图 13-18　可执行文件和动态链接库模型

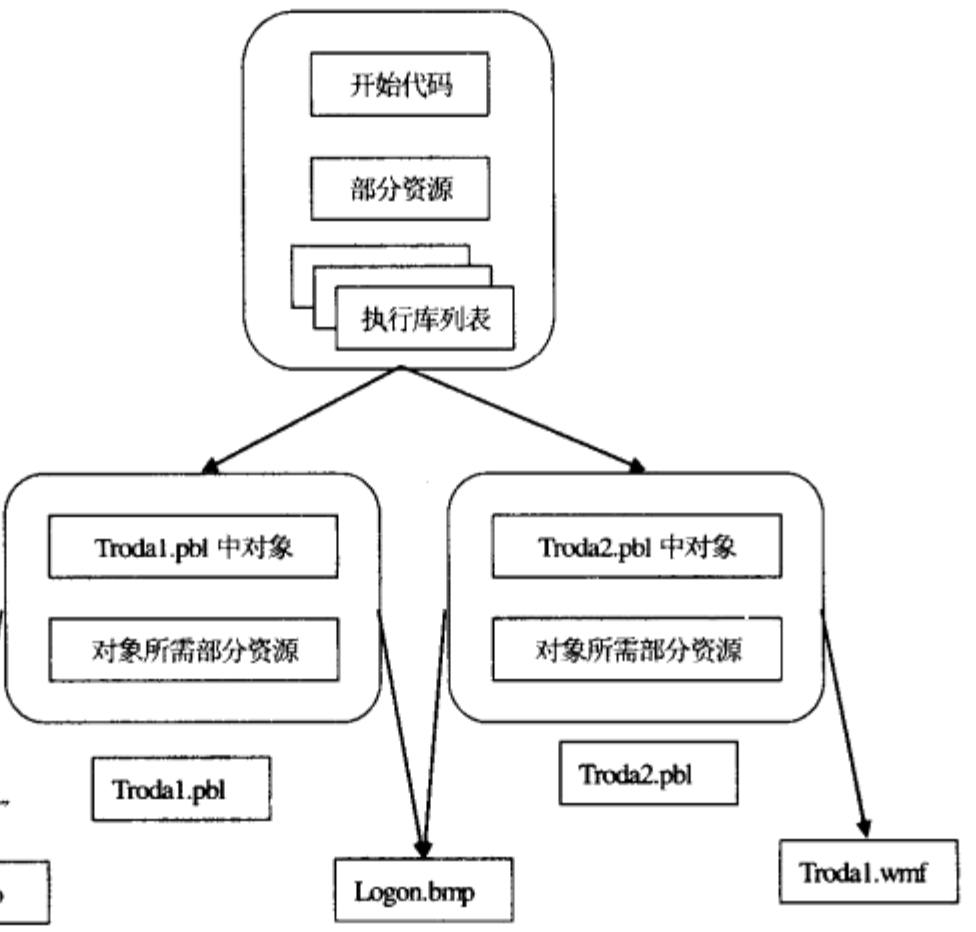


图 13-19　可执行文件、动态库文件和资源文件模型

此模型适用于要求灵活处理各类资源的应用程序，可利用此模型处理部分资源被修正的应用程序，或者是处理共享资源很大而且不经常使用的应用程序。

进行以上各项选择之后，就可在工程画板中选择编码方式，并将资源文件与可执行文件或动态链接库链接起来，最后选择编译生成所要发布的可执行应用。若编译通过，还必须在发布前对生成的可执行应用进行测试。测试步骤如下：

(1) 关闭 PowerBuilder，返回到操作环境下。

(2) 确保生成的可执行应用能够访问 PowerBuilder 的发布库文件。方法是将编译好的文件(包括可执行文件、动态库以及数据库文件等)复制到 Sybase/Shared/PowerBuilder 目录下。

(3) 运行应用的可执行文件。

通过以上测试的可执行应用就可以发布了。下节将介绍安装盘的制作。

13.6　制作安装盘

制作安装盘是一件非常难的事，因为 PowerBuilder 的应用涉及的范围很广。在制作安装盘的过程中，要完成以下几项工作：

- 找出软件发布的所有必需文件，包括 INI 文件、DLL 文件及一些资源。
- 确保软件的目录结构与软件要求一致。
- 确定所需的登记表的键值。
- 确定要生成的程序包的名称、图标等。

13.6.1　找出软件发布的所有必需文件

在这里，最令人头疼的就是 DLL 文件。实际上，运行程序一直在调用着 PowerBuilder 动态链接库里的函数。所以首先要找到 PowerBuilder 的动态库，动态链接库一般都放在了 PowerBuilder 的 Shared 子目录下，但是在 Windows 的 System 子目录下也有几个动态链接库。如 PowerBuilder 链接 FoxPro 库的 ODBC 驱动程序。其次，基于 PowerBuilder 与多种数据库的链接，需要有这些数据库的直联驱动，如 INFORMIX 的 CLI、SYBASE 的 NET CIAENT 以及 MICROSOFT 的 SQL SERVER NET CLIENT 等。这些直联驱动，因数据库的不同而相异。但一般来说，这些客户端的文件都放在一个目录里，所以，安装盘制作时可以把它们包括进去。当然，如果想减少安装盘的大小，有必要对这些文件进行筛选工作。

对于软件发布所需要的资源，主要是指在程序中用到但又没包括在建立工程时的资源文件中的一些资源，如 BMP 文件和 ICO 文件等。

INI 文件是指用来放置应用程序运行时所需的变量设置。实际上，自从 Windows 的登记表出来后，许多应用程序的发布都已逐渐放弃了使用 INI 文件的做法，而是采用宏登记的方法。

13.6.2　确保软件的目录结构同软件一致

习惯上把一些资源文件单独放在当前目录的子目录下，而在程序中，均是以相对路径来指向它。这样在做软件时，就必须保证这种路径的相对位置。例如：以@app 代表的应用程序发布后的第一级目录，安装时 BMP 文件(/BMP/res1.bmp)必须安装到@app\BMP 子目录下。

13.6.3　在注册表里的设置

一个程序的发布一般都需要设置注册表。把软件的版本信息、环境变量和软件包的名称等写在登记表里，以便统一管理。

一般软件的设置放在\HKEY_LOCAL_MACHINE\SOFTWARE 目录下。如 PowerBuilder 的设置如下：

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\PowerBuilder]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\PowerBuilder\10.0]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\PowersoftHelp]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\PowersoftHelp\inipathsForHelp]
PB60=C:\\Program Files\\Powersoft\\pb6
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\ObjectCycle]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\ObjectCycle\2.0.00]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\ObjectCycle\2.0.00\Client]
InstallDir=C:\\Program Files\\Powersoft\\ObjCyc
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\ObjectCycle\2.0.00\MSIDE]
SCCServerName=ObjectCycle
SCCServerName=C:\\Windows\\System\\mside020.dll
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\PowerBuilder Version 10.0 Online
Books]
[HKEY_LOCAL_MACHINE\SOFTWARE\Powersoft\PowerBuilder Version 10.0 Online
Books\10.0]
```

13.6.4 确定要生成的程序包的名字和目标

发布软件，总要生成一个软件包，所以必须给软件包命名，这些名称一定要反映软件的功用及公司的特点。

完成了这些准备工作就可以进行具体的安装盘制作了，安装盘软件很多，最典型的是 Visual Basic 的 Install Shield 以及 PowerBuilder 的 Install Builder。关于安装的具体制作在这里就不详述了，读者可以参看相关的书籍。

13.7 习　　题

1．填空题

(1) 默认情况下调试画板工作区分为以下 4 个窗口，在定义区域边界的操作中，系统使用的分析边界方法是基于选择点的_____、_____、_____、_____。

(2) 资源浏览窗口包含以下两种视图，包括_____、_____。

(3) 特殊断点主要分为_____、_____、_____。

(4) 无论用户选择哪种编码格式，在 PowerBuilder 中生成的应用都将包括如下内容：_____、_____、_____。

2．问答题

(1) 断点按用途共分为几类，各是如何来设置？

(2) 如何将断点设置为有效或无效？

(3) 如何创建和使用资源文件？

(4) 总结一下本章总共提到了多少种程序调试的方法？并比较其优劣性。

第 14 章　分布式体系

本章简要介绍分布式应用，所谓分布式应用是指在事务过程中，整个事务的处理分布到多个实体上来完成(在计算机网络中即多台计算机)，与传统的集中处理方式不同。所谓分布式计算体系，指的是一个使用分布式计算的方法来设计、编写和运行的应用系统。

14.1　分布式体系结构概述

就其本质上来说，它和面向对象编程一样，指的都是一种程序设计和软件发布的方式，而与具体的语言和编译器无关。

14.1.1　分布式计算

分布式计算体系从概念上来讲，可以分为以下 3 个部分：

- 数据库服务器：数据库服务器负责有关数据库的管理工作，包括数据库的建立、组织和查询以及对数据进行统计等与数据操作有关的功能。
- 客户程序：客户程序主要实现与用户进行交互的功能，从用户那里收集信息和命令，反馈给系统；从系统得到数据和结果，通过显示器或打印机等其他输出设备反馈给用户。
- 应用服务器：应用服务器是数据库服务与客户程序之间的桥梁，客户程序通过应用服务器向数据库服务器发送命令、请求数据，数据库服务器通过应用服务器响应命令、返回数据。应用服务器在此过程中对所有的命令和数据进行控制，以实现商业逻辑。

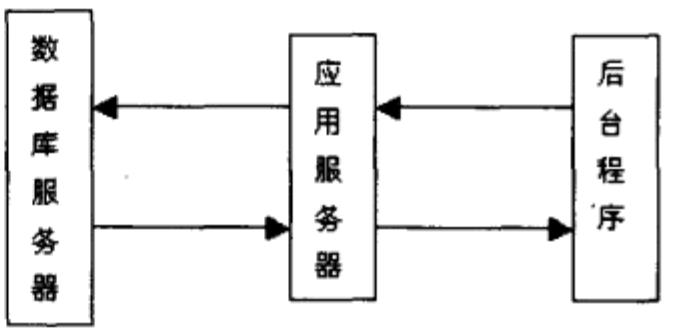


图 14-1　分布式计算体系工作流示意图

如图 14-1 所示可以更直观地了解分布式计算体系。

注意： 在这个体系结构中，前台程序并不直接同数据库服务器打交道，而是间接地从应用服务器来获取数据。

与传统的 C/S 结构体系相比，使用分布式计算体系来编写应用程序有以下好处：

- 分布式计算体系更安全可靠。第一先，客户端不和数据库服务器直接相连，甚至不在同一物理网络上，充分保证了数据的安全性，保证用户只能通过客户端应用程序来存取数据；第二，只要系统设置有相应权限管理，用户就只能进行与其权限相符的操作，从而进一步保证系统数据的安全性；第三，应用服务器的分布，

使得相应的商业逻辑由不同的人员管理，使系统更具安全性。

- 分布式计算体系对客户端的要求更低，可以充分发挥服务器的能力。所有的商业逻辑都在应用服务器和数据库服务器上实现，并且大量的统计和计算工作都在服务器上完成，这样可以充分发挥服务器的能力，并且客户端所要做的工作就只是与用户进行交互，而不要进行大量的计算工作，对客户端的要求比较低。
- 使用分布式计算体系，可以轻易地实现系统的无缝升级。如果商业逻辑变化了，那么只要对应用服务器进行修改和升级，而不要到用户那里去升级其客户端程序，这样更方便快捷、省时省力。

当然，分布式计算体系带来的好处还有更多，虽然上面只列出了主要的几点，但足以增强选择分布式计算体系来编写应用系统的信心。

在 PowerBuilder 的分布式应用系统的实现中，其结构图如图 14-2 所示。分布式应用主要由数据库服务器、应用服务器(远程对象、服务器接口)和客户端程序(远程对象代理、客户端接口、用户界面)组成，下面将分别对其加以介绍。

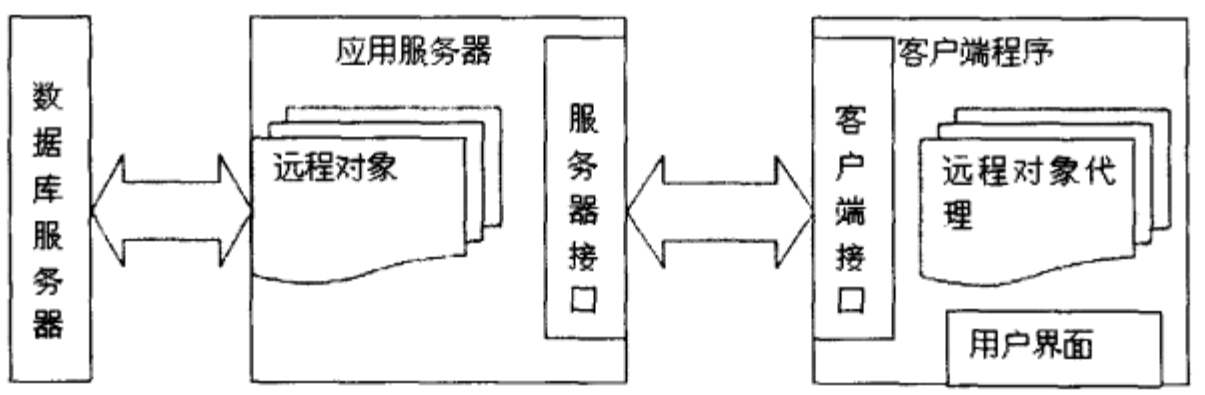


图 14-2 PowerBuilder 中分布式应用结构示意图

(1) 数据库服务器。数据库服务器的主要工作是进行数据的管理工作，包括数据库、数据存储过程、触发器等其他数据库对象。目前流行的数据库服务器管理系统主要有 Oracle、MS SQL Server、SYBASE SQL Server 等大、中型数据库以及 SYBASE ANYWHERE、DBF、BTRIEVE 等小型及桌面型数据库。

(2) 远程对象。远程对象指的是存在于应用服务器中供客户端程序调用的对象，其功能由编写者确定。客户端程序通过网络调用应用服务器上的远程对象的功能来实现自己的功能；远程对象负责与数据库服务器进行通信，取得数据，并将数据传递给客户端程序。

(3) 服务器接口。它的主要功能是负责应用服务器与客户端的网络通信工作，编写者在设置好相应的网络特性，如使用的协议和服务器端口号等其他特性后，启动服务器接口，监听客户端对服务器的请求，并通过远程对象执行请求，然后将结果发送给客户端应用程序。

(4) 客户端接口。它的功能主要是负责客户端程序与服务器接口之间的网络通信工作，客户端程序通过客户端接口与应用服务器相连，并发送请求到应用服务器，然后接收结果。

(5) 远程对象代理。远程对象代理实际上就是远程对象的一个定义，客户端程序通过此代理来正确的对应用服务器发送请求并正确地接收返回结果，即客户端程序通过远程对象代理来获知应用服务器上所提供的服务的类型，从而正确使用应用服务器。

(6) 用户界面。它主要存在于客户端程序中，主要是用于与用户的交互。客户端程序

通过用户界面接收用户的请求，执行相应的操作，然后将结果通过用户界面以一定方式展示给用户。

14.1.2　远程对象

应用服务器可以提供的服务就是远程对象。先明确一个问题，在 PowerBuilder 中，到底什么是远程对象？简单地说，远程对象就是一种不可视的用户自定义对象，所有其提供的声明为公有的函数和成员变量都可以被客户端程序调用和访问。远程对象与一般的不可视用户对象相比，还要遵循以下规则：

(1) 远程对象不能为自动生成对象，而只能使用 Create 来生成。

(2) 远程对象的公有函数的参数不能为指针。

(3) 远程对象的公有函数的参数的类型可能为 PowerBuilder 支持的所有简单类型和结构，但不能为 PowerBuilder 内部对象，如 DATASTORE。

(4) 远程对象的公有函数的参数可以为用户自定义的不可视对象。

(5) 上述对参数的描述同样适应于返回值。

(6) 远程对象的公有函数中不能包含有对可视对象操作的代码。

只要是符合上述条件的用户自定义不可视对象，都可以作为应用服务器的服务的提供载体，客户端程序可以通过远程对象代理来调用这些对象的函数或使用其成员。定义应用服务器的服务实际上就是在服务器应用中定义远程对象。

14.1.3　应用服务器

在应用服务器端应用的 Open 事件中：

```
//设置好 SQLCA 的各项属性
……
connect using sqlca;
```

在应用服务器端有一远程对象，其中有一段代码：

```
datastore dsUser
dsUser = create datastroe
dsUser.dataobject='dw_user'
dsUser.SetTransObject(sqlca)
dsUser.Retrieve()
MessageBox("信息",dsUser.RowCount())
```

在预览 dw_user 数据窗口时，共有 100 行数据，上述代码运行结果如何？结果显示，dsUser.RowCount() = 0！

当一个客户端程序与应用服务器建立连接时，应用服务器即为此连接启动一独立的会话(Session)，此会话具有如下特点：

- 会话具有单独的地址空间。对于会话来说，其他会话或服务器主会话都是不可见的。
- 会话生命期从服务器的 ConnectionBegin 事件开始，到 ConnectionEnd 事件结束。
- 会话具有单独的全局对象的副本，就是说每一会话都具有其自己独立的全局变量。

如上所述，就会知道，会话中的 SQLCA 对象和应用服务器中的 SQLCA 变量不同，前

面所说的代码是不会成功的。

注意： 可以在 ConnectionBegin 事件中初始化会话的全局变量。

可以在 ConnectionEnd 事件中处理会话的结束工作，如注销动态分配的对象。

建立应用服务器的步骤如下：

(1) 定义服务器接口全局对象。

```
TransPort g_Server
```

(2) 设置好服务器接口对象的属性。

```
g_Server = CREATE TRANSPORT
g_Server.Driver = WINSOCK   //使用的协议
g_Server.Application = 9988    //服务器端口号
```

(3)调用服务器接口对象的函数，运行应用服务器。

```
if 0 <> g_Server.Listen() then
   failure
else
   success
end if
```

在应用服务器退出之前，调用服务器接口对象的函数，退出应用服务器监听程序。

```
g_Server.StopListening()
DESTROY g_Server
```

客户端程序通过远程对象代理来存取应用服务器提供的服务。所谓远程对象代理，就是具有远程对象所有公有成员和函数定义的对象，它描述了远程对象所能提供的服务。可以使用 PROXY 类型的工程来生成远程对象的相应远程对象代理。

如有远程对象代理 uo_timeserver 包含如下成员函数 function datatime getdate()，在客户端调用此函数，代码如下：

```
//客户端接口对象为 g_connection
uo_timeserver timeserver                          //定义一远程对象代理对象
g_connection.CreateInstance(uo_timeserver)        //建立远程对象实例
messageBox("服务器时间",uo_timeserver.getdate()) //调用远程对象的函数
```

MessageBox 显示的时间是服务器的时间。归纳出客户端调用应用服务器的服务的步骤如下：

(1) 定义相应的远程对象代理对象。

(2) 调用客户端接口对象 Connection 的 CreateInstance 函数建立远程对象实例。

(3) 通过远程对象代理对象调用远程对象的函数或存取其数据成员。

14.2 分布式体系应用

建立分布式体系应用的步骤如下：

(1) 定义 Connection 类型的变量(客户端连接对象)。

```
CONNECTION g_connection
```

(2)　设置好客户端连接对象的各项属性。

```
g_connection.Driver = WINSOCK
g_connection.Location = 主机名    //服务器域名或 IP 地址
g_connection.Application = 9988    //服务器端口号
```

(3)　调用成员函数，连接应用服务器。

```
g_connection.ConnectToServer()
```

(4)在使用完成后，断开与应用服务器的连接。

```
g_connection.DisconnectServer()
```

下面介绍如何在分布式应用中使用数据窗口对象作为客户应用程序和应用程序服务器间的数据传输工具。

(1)　使用数据窗口作为数据传输工具的实现前提。在 PowerBuilder 中为通过程序在后台处理数据窗口对象提供了一个系统对象 DataStore。它除了没有可视属性外，与数据窗口控制完全相同，都可以连接数据窗口对象查找数据。

(2)　使用数据窗口传输数据用到的技术。在将数据窗口作为传递数据的工具时，并不是将数据窗口直接传递给对方，而是将数据窗口对象的数据连同它的全部状态保存到一个 BLOB 变量中，然后再将该变量传递给对方。

(3)　在此过程中用到以下函数：

- GetFullState()：取得数据窗口对象或 DataStore 对象的全部信息，并保存到 BLOB 变量中。
- SetFullState()：设置 BLOB 变量中包含的全部信息到数据窗口控制或 DataStore 中。
- GetChanges()：调用 SetFullState 函数成功后在服务器应用取得数据窗口状态设置结果。
- SetChanges()：通过 GetChanges 函数取得的 BLOB 变量设置变化了的 DataStore 或数据窗口控制。

(4)　使用数据窗口对象产生的 BLOB 传送函数的操作步骤如下：

①　服务器端连接到数据库服务器。

②　查询数据到 DataStore。

③　调用 GetFullState(BLOB_var)函数，将 DataStore 连接的数据窗口对象中的数据与状态放入 BLOB 变量中。

④　客户调用 GetChanges(tiny_BLOB1)函数，取得 BLOB 变量，并将其设置到数据窗口。

⑤　服务器数据发生变化后设置改变后的数据窗口对象。

```
SetChanges(tiny_BLOB1)
DS_1.Update()
Commit;
GetChanges(tiny_BLOB2)
```

⑥　客户程序复位数据窗口的状态。

```
SetChanges(tiny_BLOB2)
GetChanges(tiny_BLOB1)
```

⑦ 服务器端设置数据的变化。

```
SetChanges(tiny_BLOB1)
DS_1.Update()NITS
Commit;
GetChanges(tiny_Blob2)
```

⑧ 客户端。

重复第⑥步。

⑨ 服务器端。

重复第⑦步。

14.2.1 建立客户端通信

会话之间是相互独立的，采用传统的方法(如全局变量)进行会话间通信是行不通的。需要进行如下步骤才可以实现会话间的通信。

(1) 在远程对象中将要交换的数据写入文件或表中。

(2) 远程对象可以在需要的时候从文件或表中读入需要的数据。

可以看出，使用文件或表来存放会话间共同的数据，会话间通信是可以实现的，但它具有以下两个缺点：

- 需要自己解决会话间并行冲突的问题，决定何时读数据、何时写数据。
- 要进行磁盘读写或网络存取，速度慢，不适合大用户量的应用。

最好的方法，当然是在内存中建立一共享内存区，下面介绍这种方法。

从 PowerBuilder 6.0 开始，提供了一种共享对象的技术，这种技术提供了一种会话间共享对象的方法。下面介绍在 PowerBuilder 中如何使用这一技术。

(1) 定义并完成一远程对象(以 uo_sharedObject 为例)，并生成相应远程对象代理。

(2) 在服务器应用的 OPEN 事件中加入以下代码。

```
SharedObjectRegister('uo_sharedObject ', 'sharedObject ')
```

(3) 在服务器应用的 CLOSE 事件中加入以下代码。

```
SharedObjectUnregister(' sharedObject ')
```

(4) 在客户端程序要使用共享对象的代码处加入如下代码。

```
uo_sharedObject sharedObject
SharedObjectGet(' sharedObject ',sharedObject)
//调用 sharedObject 的函数或存取其属性
```

上述代码主要通过以下几个函数实现其功能，下面逐个讲解。

(1) SharedObjectRegister 函数

在调用其他所有函数之前，一定要调用此函数。此函数的功能是在应用服务器上注册一个共享对象，即在应用服务器上启动了一个独立会话；在此会话空间中建立一个指定类型的共享对象。

(2) SharedObjectGet 函数

此函数取得一个已经注册了的共享对象的引用。通过此引用来调用共享对象提供的函数或存取其数据成员。

(3)　SharedObjectUnregister 函数

此函数与 SharedObjectRegister 函数的功能相反，其功能为注销一个注册过的共享对象，即注销此共享对象，结束与此共享对象相关的会话。共享对象具有以下特性：

- 共享对象是一个远程对象，它具有远程对象的所有特性。
- 共享对象运行在一独立的会话中，具有会话的所有特性。
- 共享对象的会话开始于对象的构造事件，结束于其析构事件。

注意： 共享对象的初始化可以放在其构造事件中。共享对象的结束代码可以放在其析构事件中。

14.2.2　控制用户登录应用服务器

客户程序在使用应用服务器提供的服务之前，总是要先连接到应用服务器。在实际应用中，一般情况下都要对连接的用户进行核对，以验证其身份，只有身份合法的用户才被许可连接到应用服务器上。在 PowerBuilder 中，系统提供了一种系统手段来支持这一功能。

(1)　服务器应用的 ConnectionBegin 事件。

事件的参数：

```
string userId       //客户端用于登录到应用服务器的用户名 connection.userid
string password     //客户端用于登录到应用服务器的密码 connection.password
string connectstring //其他可以附带的信息 connection.connectString
```

事件的返回值：

```
ConnectPrivilege!                  //普通用户，允许连接
ConnectWithAdminPrivilege! //超级用户，允许连接
NoConnectPrivilege                 //非法用户，不允许连接
```

如上所述，可以通过参数取得用户名和密码，可以通过返回值来控制是否允许用户登录到应用服务器上及用户的身份，这样就可以达到控制用户登录的目的。

请看下列代码：

```
if userid = 'Administrator ' and password = ' system ' then
   return ConnectWithAdminPrivilege!
Elseif userid = ' Guest ' then
   Return ConnectPrivilege!
Else
   Return NoConnectPrivilege!
End if
```

(2)　在线用户监视器。

请看下列应用服务器上的代码：

```
//g_connection 为客户端接口
//w_monitor 为一窗口
//dw_monitor 为一外部数据源数据窗口，列为 userid,connectid
//w_monitor 的 Open 事件
timer(1)
//w_monitor 的 timer 事件
connectionInfo info[ ]
long userCount,row
userCount = g_Connection.GetServerInfo(info)
dw_monitor.setRedraw(false)
```

```
dw_monitor.reset()
for I = 1 to userCount
row = dw_monitor.insertRow(0)
dw_monitor.object.UserID[row] = info[I].connectUser
dw_monitor.object.ConnectID[row] = info[I].clientId
end for
dw_monitor.setRedraw(TRUE)
```

以上代码即可实现在线用户监视器。

14.2.3 创建分布式 PowerBuilder 服务器

本节的目的是为生产管理信息系统创建分布式 PowerBuilder 服务器。要完成创建分布式 PowerBuilder 服务器需要做如下工作：

- 创建用户界面。
- 管理客户连接。
- 创建远程对象。
- 访问数据库。
- 使用共享对象。
- 传递消息到客户端。
- 发布服务器应用。

1. 创建用户界面

首先服务器必须打开一个窗口，目的是为了监听客户端的连接。这里利用的技术是在应用的 Open 事件中创建了 Transport 对象并调用 Listen 函数，服务器应用将会在执行完 Open 事件中的脚本后自动终止。

除了启动监听的界面外，在服务器的应用中还可以提供一个控制台界面。服务器应用的控制台界面可以用于监测系统的各项活动，为系统管理人员提供关于客户端的连接信息。创建的服务器应用的控制台界面设计画板如图 14-3 所示。

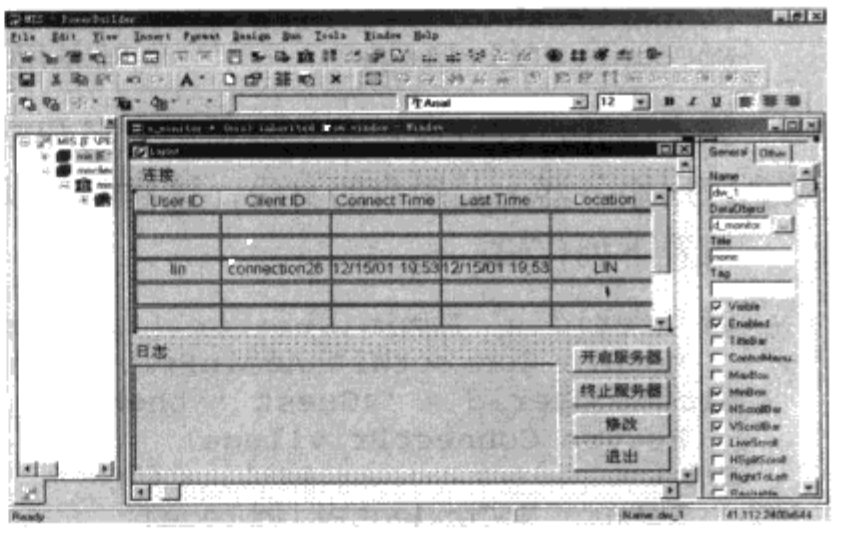


图 14-3 服务器应用的控制台界面设计画板

注意： 同样的控制台界面可以用于多服务器，在这种情况下，创建一个普通的用户界面就可以监测所有的服务器了。控制台的用户界面不需要很花哨。

然后就可以从服务器中监测活动了。控制台还可以用来终止客户端的连接，以减少服务器的拥塞。

2. 管理客户连接

为了使服务器能够成功地处理客户端的连接，需要编写一些代码来规定服务器的监听

控制和处理客户端提出的服务请求。

注意： PowerBuilder 6 创建的客户端应用是无法连接到版本 PowerBuilder 7、8 或 9 所创建的服务器应用的，因为 PowerBuilder 的结构做了较大的调整。

(1) 监听客户端。使用 Transport 对象来监听客户端的连接，这是一个非可视的用户对象，和 Transaction(事务)对象很相似。为实现监听所要完成的操作如下：

- 实例化 Transport 对象。
- 设置 Transport 对象的属性。
- 调用 Listen 函数进行监听。
- 进行错误检查。

实现监听的脚本如下：

```
//Transport
long rc
trans=create transport
trans.driver=winsock
trans.zpplication=dpbserv
rc=trans.listen()
if rc<>0  then
messagebox(Start Server Fails!,rc0
endif
```


注意： 以上操作顺序不可以改变。但是，这些操作可以在一个脚本里完成，也可以在多个脚本里完成，读者可以任意选择。

(2) 处理客户端请求。服务器将首先验证客户端请求的有效性。若有权建立连接，则接受连接请求并授予权限；否则拒绝。在 ConnectBegin 和 ConnectEnd 事件中编写适当的功能脚本。

3. 创建远程对象

创建远程对象是服务器应用中最主要的组成部分。这些远程对象可以被一个或多个客户端调用。在这里可以编写用户对象以供调用，同时设置远程对象(即一些非可视的用户对象)。

注意： 为了简化维护工作，建议将远程对象放置在一个单独的库文件中，而不是和客户端与服务器端的库文件放在一起。

4. 访问数据库

可以使用 Datastore 使服务器应用中的对象和数据库进行交互。Datastore 是一种非可视的数据窗口控件，它在分布式应用中提供了在远程服务器中执行数据库处理的能力。

注意： 在服务器应用中的 Datastore 对象所对应的数据窗口对象不能使用 RichText 风格。

在这一步里，所要做的工作是在服务器和客户端共享数据、执行检索、更改操作、进

行数据库事务缓存等。在此是用一些常见的读取数据、检索数据和更改数据的脚本，用法与以前基本相同。

5．使用共享对象

PowerBuilder 提供了一个共享对象(Shared Object)，用于在分布式应用中共享数据。可共享的对象包括所创建的任何非可视用户对象。

6．传递消息到客户端

向客户端传递消息的步骤和方法是，先把请求放入队列，然后等待处理。另外，为了给客户端发送消息，服务器需要知道接受消息的客户端的具体对象。

7．发布服务器应用

在发布服务器应用时，注意必须使用动态链接库，必须确保所有对象都已经被包含。除此之外，还要尝试处理一些如下的高级功能：

- 处理异步请求。
- 当客户端进行异步调用的时候，服务器不会立即对请求进行处理，而客户端可以进行其他的工作，不需要等待服务器处理完请求。此时，请求被放在队列中，且遵循同步请求有限的原则。
- 避免死锁。

以下给出了一些常用的避免死锁的方法：

- 使用异步调用将服务器消息传递给客户端。
- 不允许在共享对象会话期间进行同步调用。
- 不允许将客户会话对象的引用传递给客户端本身和共享对象以外的其他会话中。

完成上述工作后，最终得到一个为生产管理信息系统创建的分布式 PowerBuilder 服务器，运行效果如图 14-4 所示。可以在这个控制台上监听和控制客户端的操作。

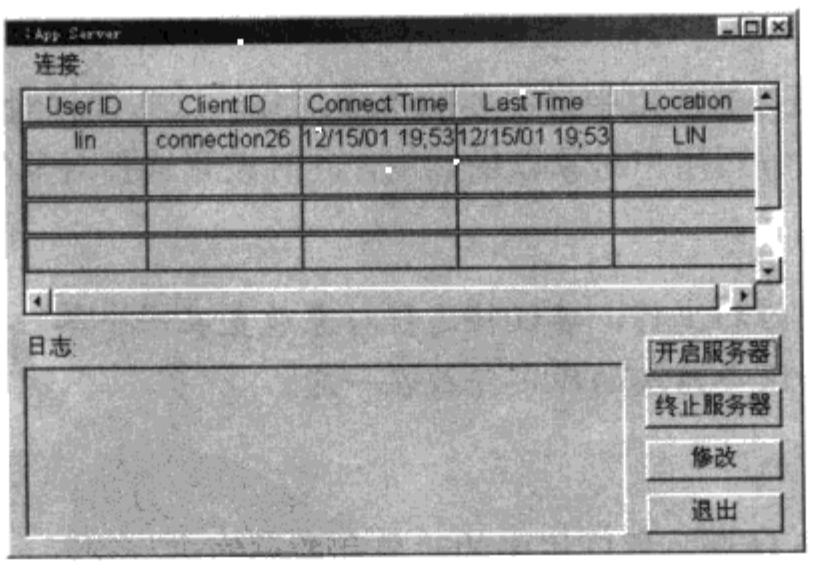


图 14-4　创建分布式 PowerBuilder 服务器的控制台界面

14.3　分布式体系设计实例

14.3.1　创建 EAServer 组件

本节的任务是为生产管理信息系统(MIS)在 PowerBuilder 10.0 中创建 EAServer 组件。即在 PowerBuilder 中创建非可视的用户对象，然后再将它们发布成 EAServer 组件。

PowerBuilder 分布式计算是为了解决在 Internet/Intranet 环境下构建应用软件的体系结构问题，是在传统的客户/服务器(Client/Server)模式的基础之上提出来的，它广泛应用在各种 Internet/Intranet 环境中，而且越来越显示出它的重要性。

本节将介绍有关 EAServer 组件的内容。EAServer 是 Sybase 公司为了充分发挥基于组件和多层次体系结构的应用特点而专门设计的高性能提交环境，PowerBuilder 10.0 对此又有较大的改进和扩充。在应用分布式 PowerBuilder 的时候，常常要用到 EAServer 组件的相关技术。

需要掌握的函数及语句有 createdatastore、this.getcontextserveice(transactionserver、…)和 create transaction 等。

注意： PowerBuilder 10.0 的 EAServer 组件包括了 PowerBuilder 7.0 的 Jaguar 组件的内容，并进行了一定的改进和扩充。EAServer 集成了操作方便且功能强大的 Web 应用服务器 PowerDynamo 和高性能的组件事务处理服务器 Jaguar CTS。请读者注意区别 PowerBuilder 10.0 与 PowerBuilder 9.0 两个版本的内容以及各自的特点。

要在 PowerBuilder 10.0 中创建 EAServer 组件，首先要在 PowerBuilder 中创建非可视的用户对象，然后再将它们发布成 EAServer 组件。为了将组件发布到 EAServer 上，需要创建和生成一个新的工程。在这个工程里需要指出生成 EAServer 组件的 PowerBuilder 非可视的用户对象，并且还需要定义输出库的名称，用来包含所生成的组件。使用向导来创建工程，之后打开此工程，执行相关操作将组件发布到 EAServer 上，并作其他一些处理(如 CORBA IDL 接口、PROPS 文件和 PBD 文件等)，下面将具体介绍这些步骤。

(1) 选择“开始”|“程序”| EAServer 命令，启动服务器。

(2) 单击 PowerBuilder 工具栏里的 New 图标，弹出 New 对话框，选择 Target 选项卡，在其中选择 EAServer Component 图标，单击 OK 按钮，如图 14-5 所示。

(3) 在 About the EAServer Component Start Wizard 对话图中浏览向导说明，单击 Next 按钮，如图 14-6 所示。

(4) 在 What you will do 对话框口中列出接下来的几个步骤，单击 Next 按钮继续，如图 14-7 所示。

(5) 在 Specify New Application and Library 对话框中，根据需要填入相应内容，然后单击 Next 按钮继续，如图 14-8 所示。

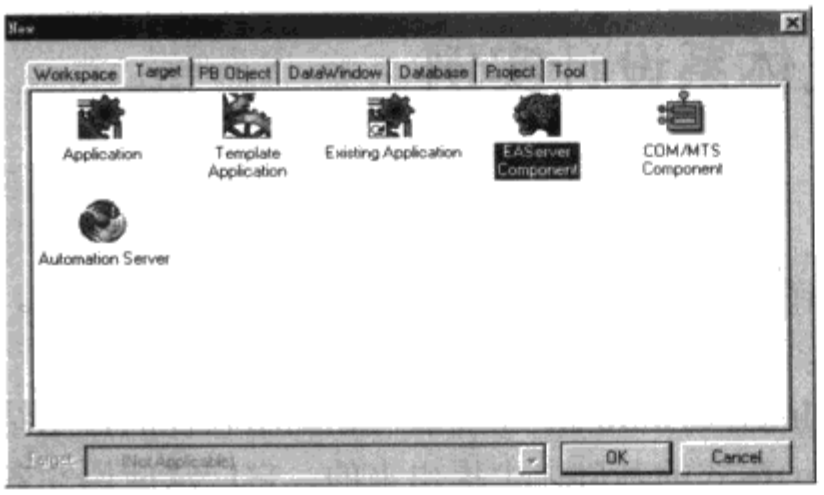


图 14-5 Target 选项卡

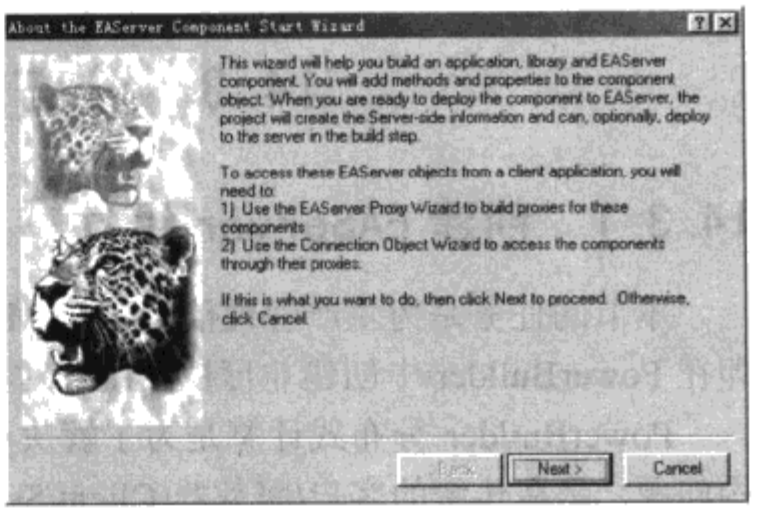


图 14-6 About the EAServer Component Start Wizard 对话框

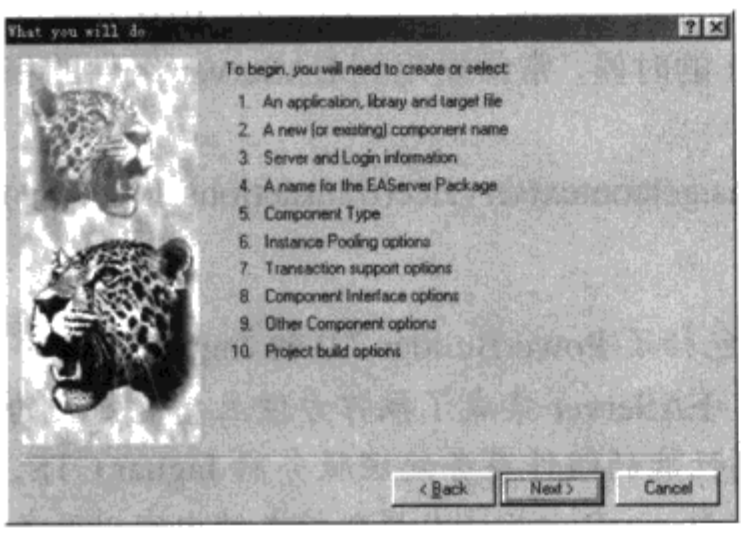


图 14-7 What you will do 对话框

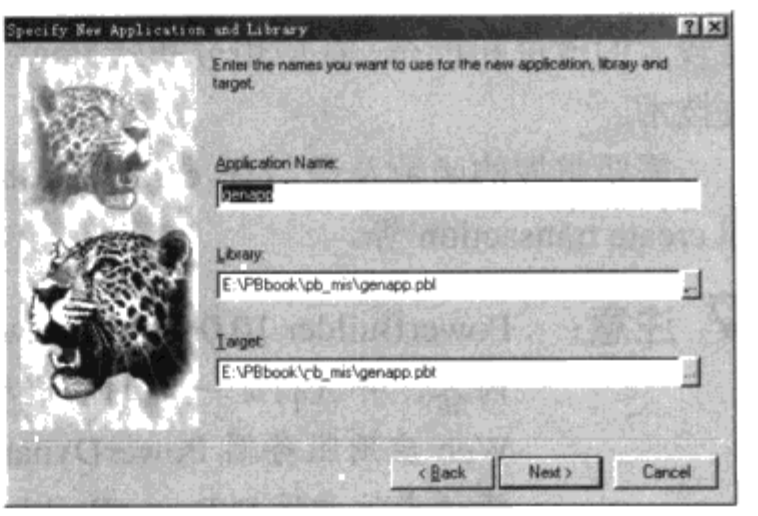


图 14-8 Specify New Application and Library 对话框

(6) 在 Adjust Application Library Search Path 对话框中，直接单击 Next 按钮继续，如图 14-9 所示。

(7) 在 Specify Interface Implementation 对话框中，根据需要选择 Implemente New Interface 或 Inplemente an Existing EAServer Remote Interface。在此，选择 Implemente New Interface 选项，然后单击 Next 按钮继续，如图 14-10 所示。

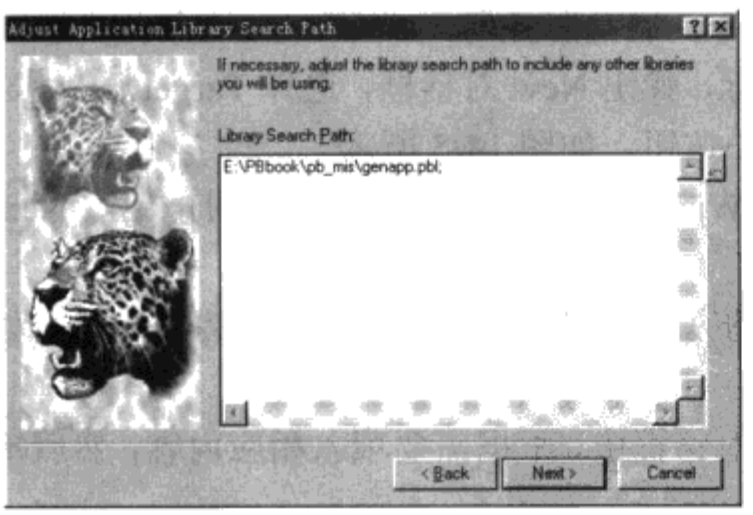


图 14-9 Adjust Application Library Search Path 窗口

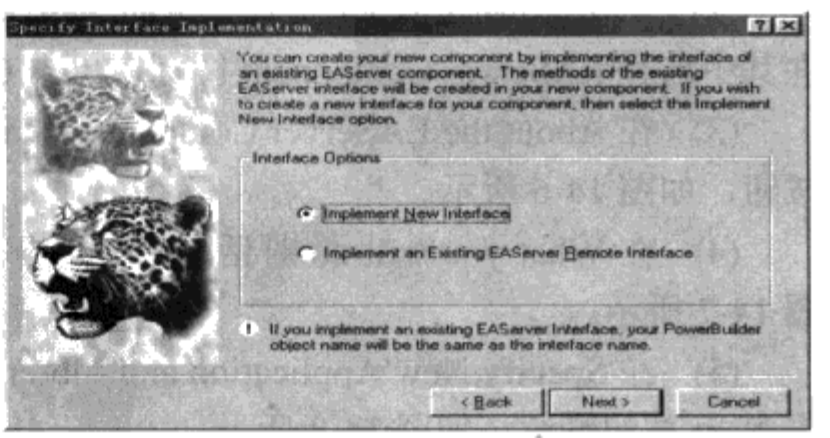


图 14-10 Specify Interface Implementation 窗口

(8)　在 Specify New PowerBuilder Object 对话框中，填入所要生成的组件非可视用户对象名称(n_genapp)，然后单击 Next 按钮继续，如图 14-11 所示。

(9)在 Specify EAServer Component Name 对话框中，填入 EAServer 组件名称(n_genapp)，可以使用和 PowerBuilder 用户对象相同的名称，然后单击 Next 按钮继续，如图 14-12 所示。

(10) 在 Choose EAServer Profile 对话框中，根据需要选择外形，然后单击 Next 按钮继续，如图 14-13 所示。

(11) 在以后的各个对话框中，按要求填入或选择相应的内容，包括指定 EAServer 服务器信息、组件类型、实例缓冲池的支持、事务的支持类型、设置工程名称、设置动态链接库名称等。

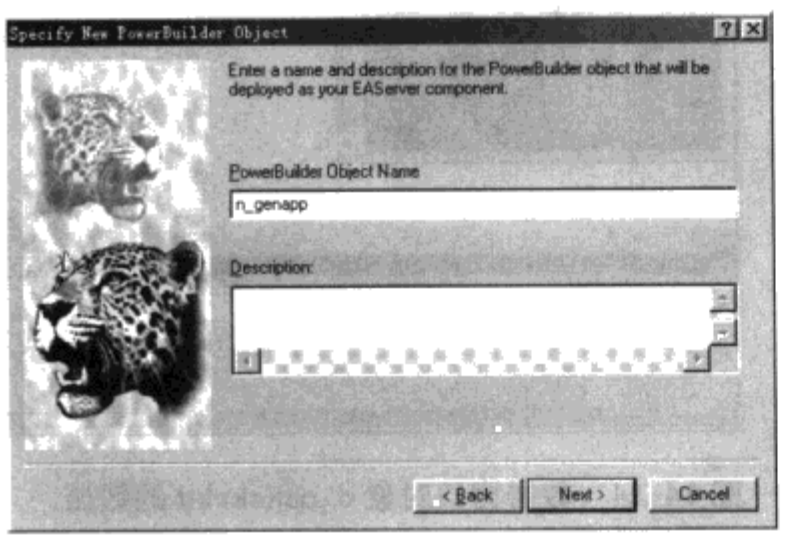


图 14-11　Specify New Power Builder Object 窗口

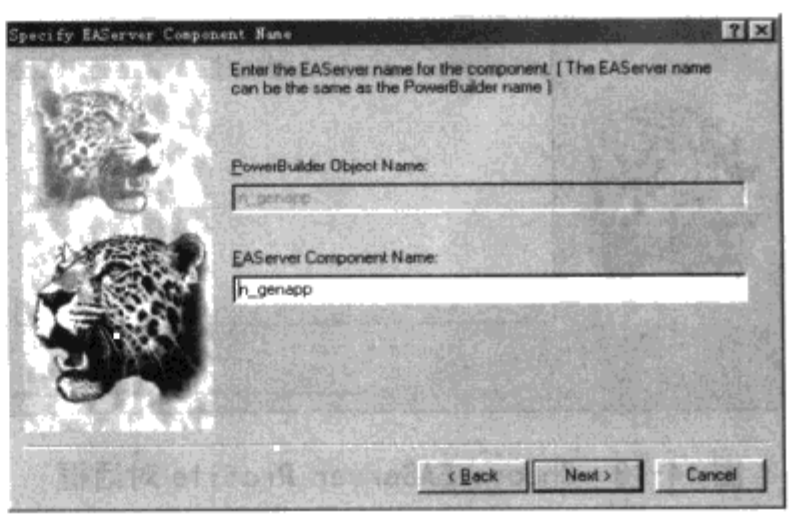


图 14-12　Specify EAServer Component Name 窗口

以上各个步骤完成后，向导将生成 3 个对象，应用对象(genapp)、用户对象(n_genapp)和工程对象(p_genapp)，这些对象可以在库画板中看到。

(12) 创建数据窗口对象 d_cpkckrkd。选择成品库的出库表单，并选择所有的列，如图 14-14 所示。生成的数据窗口对象如图 14-15 所示。

(13) 在用户对象画板中编写代码。包括 n_genapp 的 Active 事件、n_genapp 的 Deactivate 事件和一些函数。下面以 n_genapp 的 Active 事件为例，说明代码的编写方法。

```
integer I_rc

//获得事务服务器
I_rc=this.getcontextserveice(transactionserver,its_jag)
If I_rc<>1  then
Messagebox(error,transactionserver error!)
Return -1
End if
//创建事务
if not isvalid(itr_trans)  then
itr_trans=create transaction
end if
//连接到数据库
itr_trans.DBMS= "ODBC"
itr_trans.DATABASE= "EAS Demo DB V9"
itr_trans.DBParm = "ConnectString='DSN= EAS Demo DB V9;UID=dba;PWD=sql',
UseContextObject='yes',CacheName='EAS Demo'"
CONNECT USING itr_trans;
//连接失败处理
```

```
IF itr_trans.SQLCode<>0 THEN
MessageBox("",itr_trans.SQLErrText)
Return -1
END IF
//创建数据存储
if not isvalid(ids_data)  then
ids_data=createdatastore
end if
ids_data.daatobject=d_customer
ids_data.settransobject(itr_trans)
```

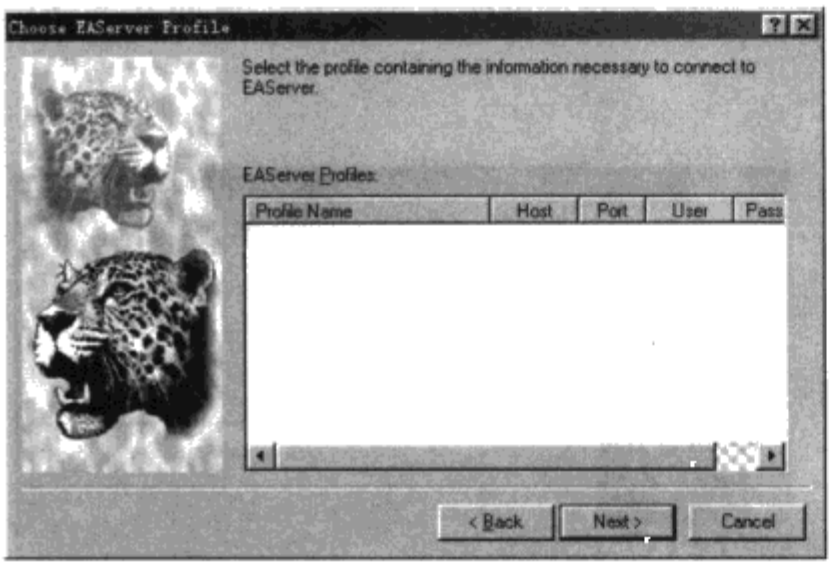


图 14-13　Choose EAServer Profile 对话框

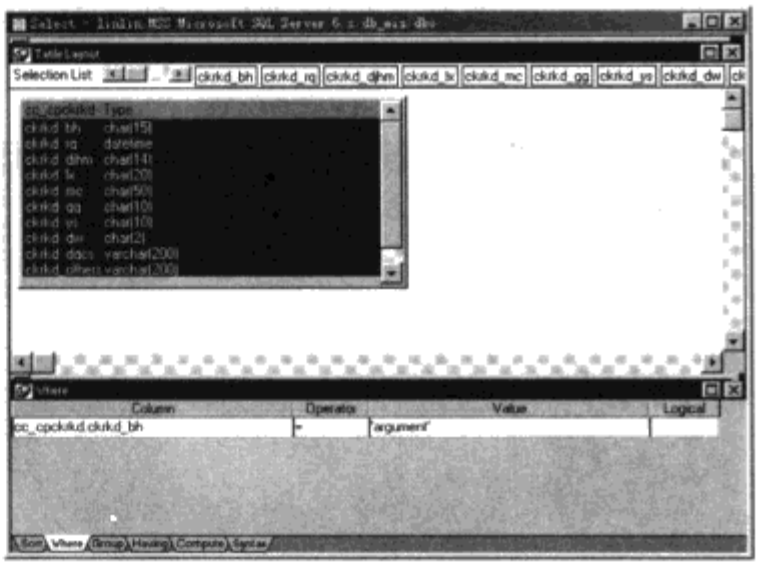


图 14-14　数据窗口对象 d_cpkckrkd 的数据

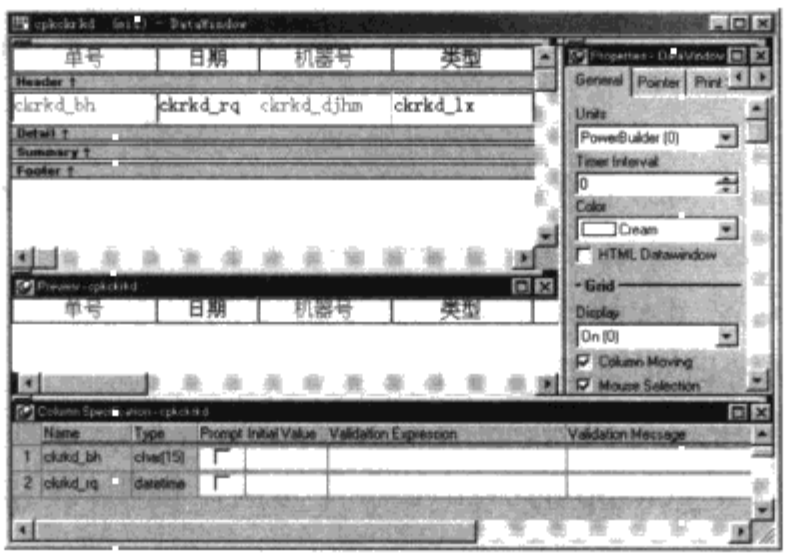


图 14-15　数据窗口对象 d_cpkckrkd

(14) 在工程画板中打开工程对象，单击 Build 按钮。

注意： 在开始创建 EAServer 组件之前，一定要先启动 EAServer 服务器。

至此，就完成了为生产管理信息系统(MIS)创建EAServer组件的工作了，结果如图14-16所示。

本例学习了在 PowerBuilder 10.0 中创建 EAServer 组件的具体方法和步骤。在 PowerBuilder 中创建了非可视的用户对象，然后再将它们发布成 EAServer 组件。为了将组件发布到 EAServer 上，需要创建和生成一个新的工程。在这个工程里需要指出生成 EAServer 组件的 PowerBuilder 非可视的用户对象，并且还需要定义输出库的名称，用来包含所生成的组件。使用向导来创建工程，之后打开此工程，执行相关操作将组件发布到

EAServer 上，并做一些其他处理。Sybase 公司为了充分发挥基于组件和多层次体系结构的应用特点而专门设计了 EAServer 这个高性能提交环境，PowerBuilder 10.0 对此又有较大的改进和扩充。在应用分布式 PowerBuilder 的时候，常常要用到 EAServer 组件的相关技术。需掌握的函数及语句有 createdatastore、this.getcontextserveice(transactionserver,…)、create transaction 等。

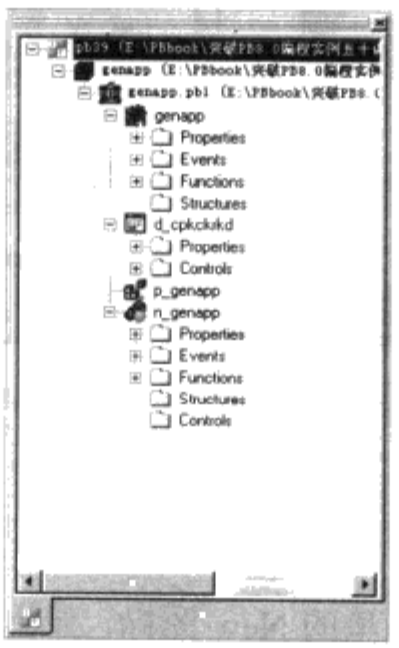


图 14-16　创建 EAServer 组件的实例效果

14.3.2　发布客户端应用

本节的任务是为生产管理信息系统(MIS)发布客户端应用。需要为系统创建 EAServer 的客户端和连接 EAServer 的服务器、生成 EAServer 组件的代理对象、调用 EAServer 组件，并尝试实现异步调用及处理通信错误的功能。需要完成的工作有如下几个方面：

- 创建 EAServer 的客户端。
- 连接 EAServer 的服务器。
- 生成 EAServer 组件的代理对象。
- 调用 EAServer 组件。

并在以上基础上尝试实现以下功能：

- 异步调用。
- 处理通信错误的功能。

下面将重点介绍创建 EAServer 的客户端应用。这里使用 14.3.1 节创建的 EAServer 组件。

发布分布计算环境中的客户端应用过程与 PowerBuilder 应用是一样的，也有两种打包应用的方法。

- 独立的可执行文件(EXE 文件)，该文件包含了应用中的所有对象。
- 可执行文件加上一个或多个动态链接库。

1. 创建 PowerBuilder 应用

(1) 启动 PowerBuilder 应用，并确保 EAServer 服务器以及启动。

(2) 单击 PowerBuilder 工具栏里的 New 图标，弹出 New 对话框，选择 Target 选项卡，

选择 Application 图标，单击 OK 按钮，如图 14-17 所示。

(3) 在 Specify New Application and Library 对话框中，填入 Application Name(应用)、Library(库)和 Target(目标)的名称，然后单击 Finish 按钮结束，如图 14-18 所示。

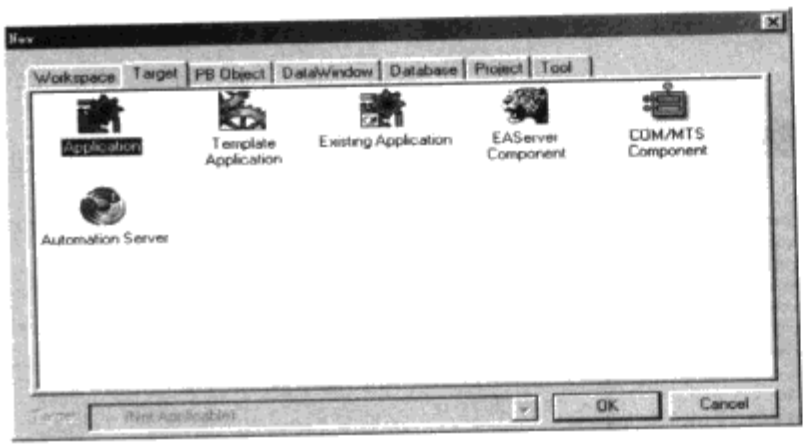


图 14-17 Target 选项卡

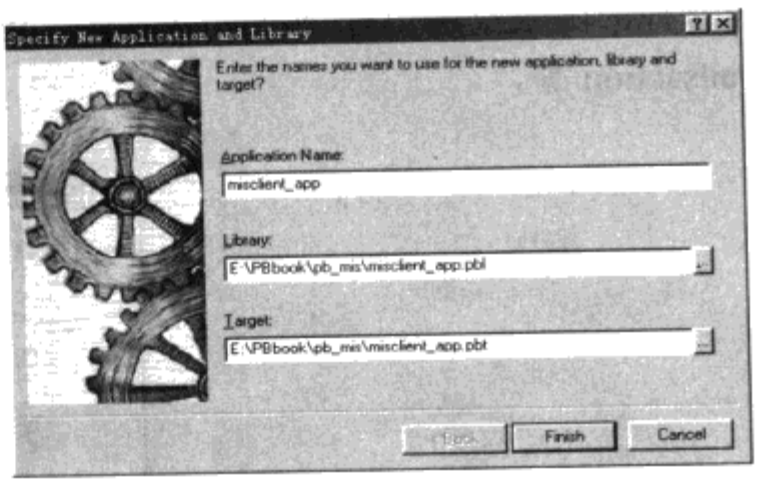


图 14-18 Specify New Application and Library 对话框

2. 创建连接对象

(1) 单击 PowerBuilder 工具栏里的 New 图标，弹出 New 对话框，选择 PB Object 选项卡，选择其中的 Connection Object Wizard 图标，单击 OK 按钮，如图 14-19 所示。

(2) 在 About the Connection Object Wizard 对话框中，直接单击 Next 按钮继续，如图 14-20 所示。

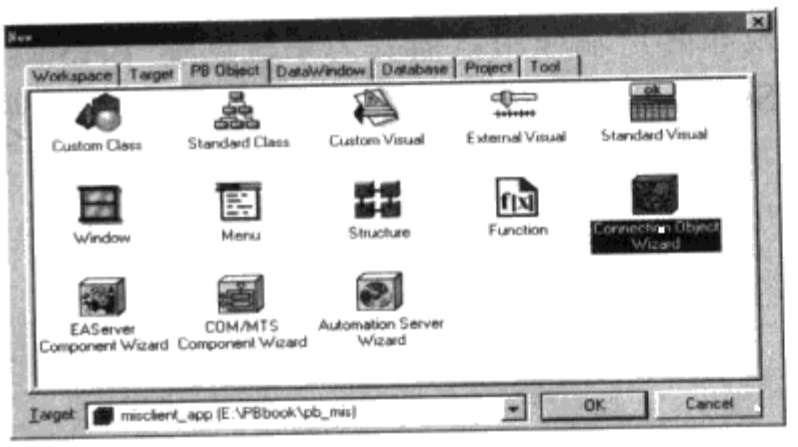


图 14-19 PB Object 选项卡

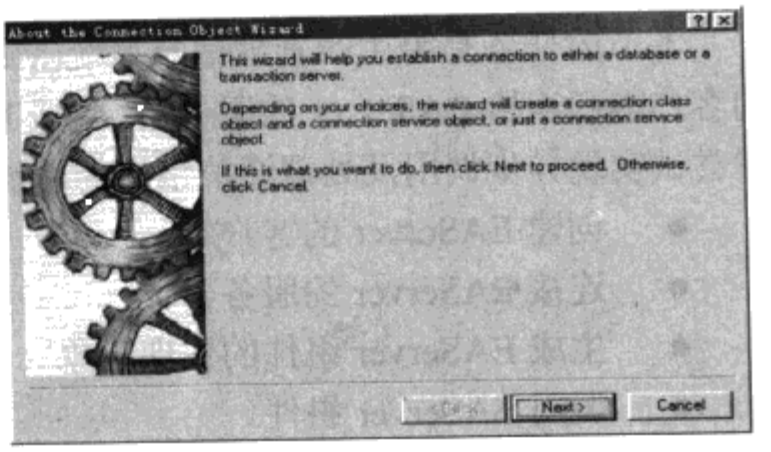


图 14-20 About the Connection Object Wizard 对话框

(3) 在 Specify Destination Library 对话框中，选择存放连接对象的库 E:\Pbbook\pb_mis\misclient_app.pbl，然后单击 Next 按钮继续，如图 14-21 所示。

(4) 在 Specify Connectivity 对话框中，选择 Requires EAServer Connection，然后单击 Next 按钮继续，如图 14-22 所示。

(5) 在 Choose EAServer Profile 对话框中，根据需要选择外形，然后单击 Next 按钮继续，如图 14-23 所示。

(6) 在以后的各个对话框中，按要求填入或选择相应的内容，包括指定 EAServer 服务器信息、选择默认包名、指定连接对象名称、选择获取连接信息的位置及选择创建任务表等。

3. 创建组件代理对象

(1) 单击 PowerBuilder 工具栏里的 New 图标，弹出 New 对话框，选择 Project 选项卡，选中 EAServer Proxy Wizard 图标，单击 OK 按钮，如图 14-24 所示。

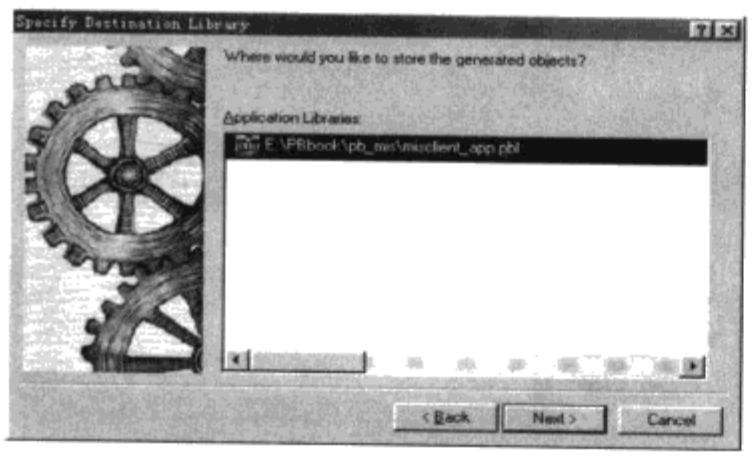


图 14-21　Specify Destination Library 对话框

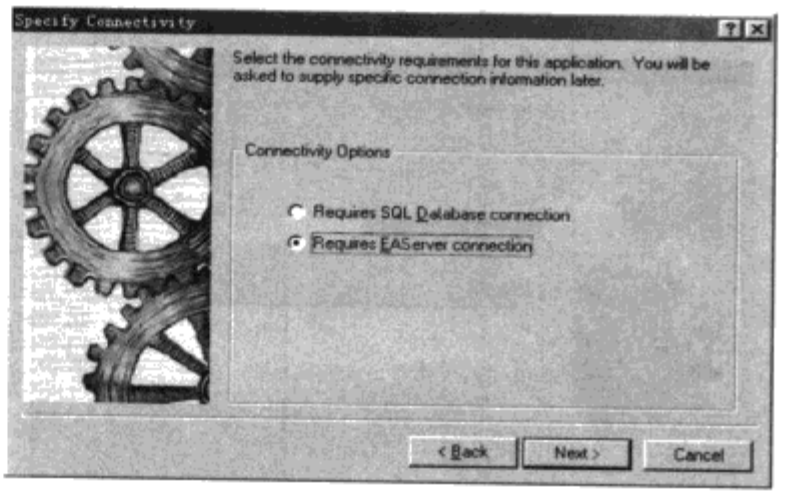


图 14-22　Specify Connectivity 对话框

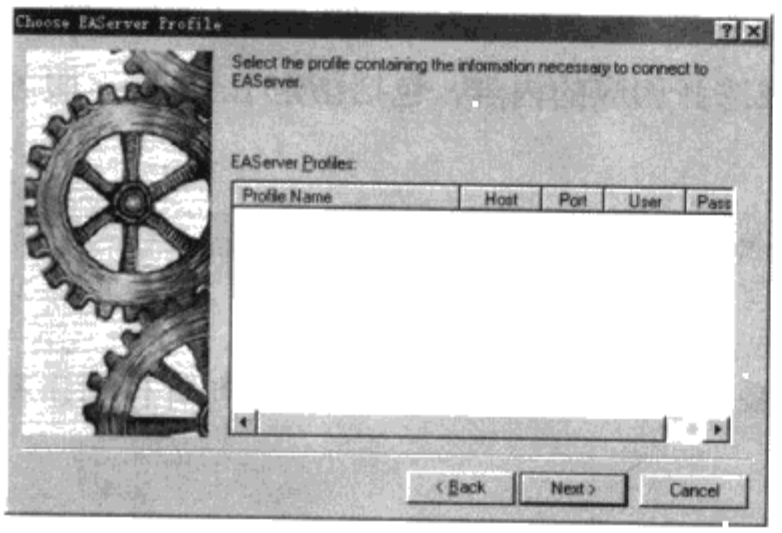


图 14-23　Choose EAServer Profile 对话框

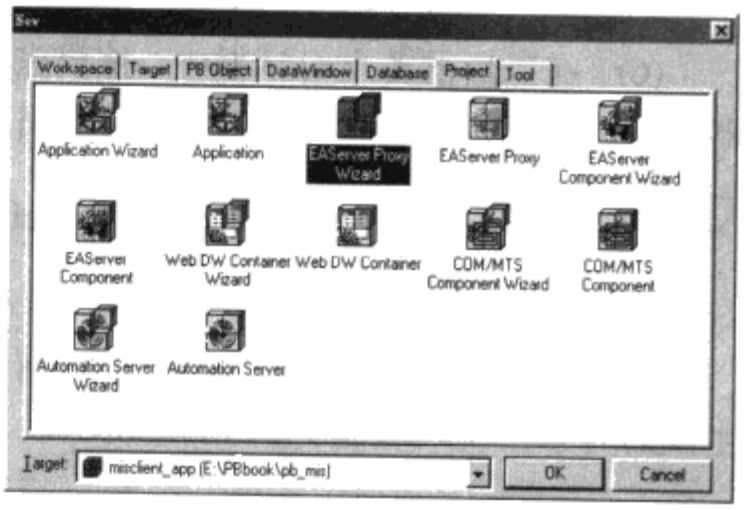


图 14-24　Project 选项卡

(2) 在 About the EAServer Proxy Wizard 对话框中，直接单击 Next 按钮继续，如图 14-25 所示。

(3) 在 Specify Destination Library 对话框中，选择存放工程对象的库 E:\Pbbook\pb_mis\misclient_app.pbl，然后单击 Next 按钮继续，如图 14-26 所示。

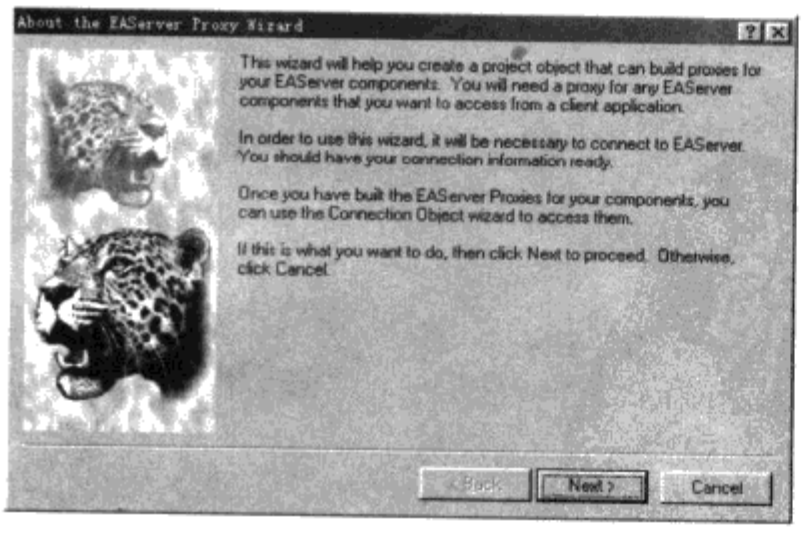


图 14-25　About the EAServer Proxy Wizard 对话框

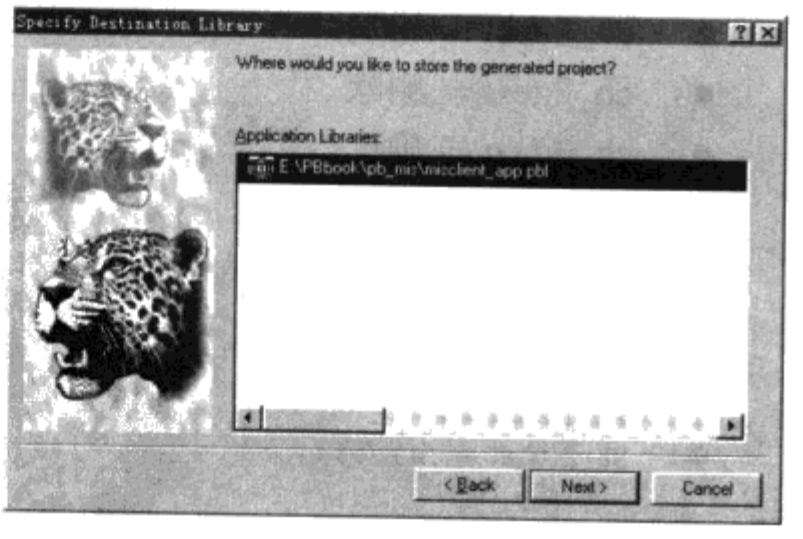


图 14-26　Specify Destination Library 对话框

(4) 在 Specify Project Object 对话框中，填入所要生成的工程的名称 p_misclient_app_easproxy，然后单击 Next 按钮继续，如图 14-27 所示。

(5) 在 Choose EAServer Profile 对话框中，根据需要选择外形，然后继续单击 Next 按钮，如图 14-28 所示。

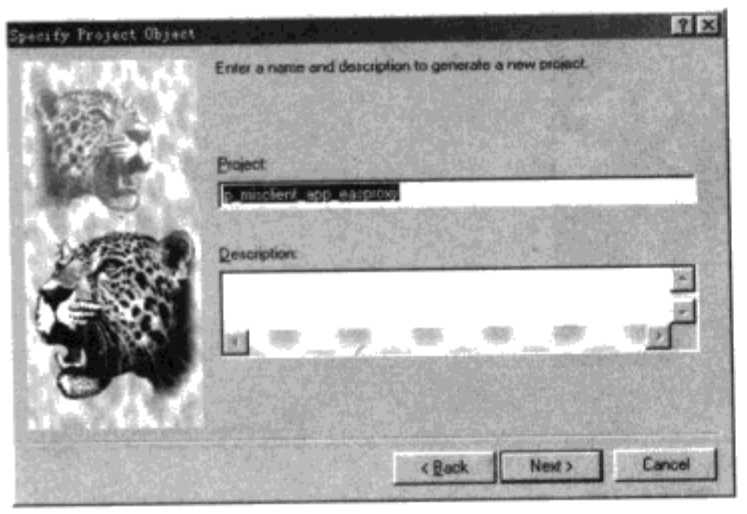


图 14-27 Specify Project Object 对话框

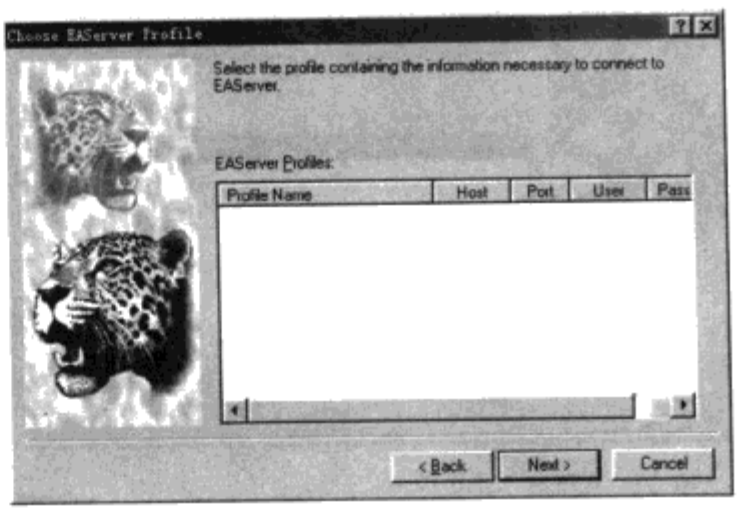


图 14-28 Choose EAServer Profile 对话框

(6) 在以后的各个对话框中，按要求填入或选择相应的内容，包括指定 EAServer 服务器信息、选择包、选择代理对象的存放库等。

注意： 在 Ready to Create EAServer Proxy 对话框中，选择生成任务列表，这样就可以看到生成的任务列表了。

EAServer 将产生 n_misclient_app 的代理对象。此时，在库画板中可以看到所有创建和生成的对象。

4. 创建客户端用户界面

(1) 在库画板中，从 genapp.pbl 中复制 d_cpkckrkd 数据窗口对象，把它复制到 misclient_app.pbl 中。

(2) 创建窗口对象 w_cpkckrkd，设置控件，并设置其属性。

- 数据窗口：dw_1。
- 命令按钮：cb_1~ cb_4。
- cb_1.text = 增加。
- cb_2.text = 删除。
- cb_3.text = 确认。
- cb_4.text = 退出。
- 单行编辑框：sle_1、sle_2。
- 静态文本框：st_1、st_2。
- st_1.text = 出库单号。
- st_2.text = 记录数。

注意： 为数据窗口设置水平和垂直的滚动条，以方便大量数据的查看。

(3)　编写各个控件的脚本。

①　增加事件(cb_1.clicked())的脚本

在单击此按钮后，系统会增加一条空的新记录行，并滚动到该记录，以供用户添加新的记录数据。

```
bh=trim(em_1.text)
if mid(bh,12)="0000" then
  messagebox("警示","请先填写出库单号!")
  dw_1.enabled=false
  return
else
  if len(bh)<>15 then
    messagebox("警示","出库单号的长度不正确，请修改!")
    dw_1.enabled=false
    return
   else
//检查出库单号是否重复
    integer i
    i=0
    select count(*) into :i from cc_cpckrkd where ckrkd_bh =:bh;
    //using sqlca;
    if i<>0 then
       messagebox("错误警示","出库单号有重复，请重新输入!")
       return
    else
       dw_1.enabled=true
       dw_1.insertrow(0)
       records=records+1//记录数加 1
       em_2.text=string(records)//显示当前记录条数
      dw_1.scrolltorow(records)
     end if
  end if
end if
```

②　删除事件(cb_2.clicked())的脚本

把光标移至要删除的行的位置，单击此按钮就可以删除该行。删除之前，系统会给与提示信息来确认删除。

```
if (long(em_2.text)-1)>=0 then
  integer flag
  flag=messagebox("确认删除","确实要删除这条记录吗? ",exclamation!,yesno!)
  if flag=1 then
    dw_1.deleterow(0)
    em_2.text=string(long(em_2.text)-1)
  records=(records)-1
  end if
end if
```

③　确认事件(cb_3.clicked())的脚本

因为每件产品都要进行跟踪记录，所以采用逐条查重插入的方法向数据库中存储所填入的数据。先要检查输入是否合乎要求，在确认无重复之后，还要进行 3 项操作，更新成品出库入库单、更新状态(已出库 02)、更新成品类型的当前仓储量。无论存储成功与否，系统都会有提示信息。

```
integer row,ccl
string dqzt
```

```
row=0
if dw_1.rowcount()=0 then
   messagebox("提示","本次出库单没有任何数据信息!")
else
  //检查库存量
do
   row=row+1
   lx=dw_1.getitemstring(row,"ckrkd_lx")
   select cplx_ccl into :ccl from cc_cplx where :lx=cplx_bh;
   if ccl<0 then
       messagebox("警告","操作失败! 第"+string(row)+"个记录库存量为 0!")
     return
   end if
loop until dw_1.rowcount()=row
if dw_1.update()=1 then//更新成品出库入库单
   messagebox("操作结果提示","操作成功!")
   //实际更新数据
   row=0
   do
     row=row+1
     rq=today()
     djhm=dw_1.getitemstring(row,"ckrkd_djhm")
     mc=dw_1.getitemstring(row,"ckrkd_mc")
     ys=dw_1.getitemstring(row,"ckrkd_ys")
     dw=dw_1.getitemstring(row,"ckrkd_dw")
     lx=dw_1.getitemstring(row,"ckrkd_lx")
     dgcs=dw_1.getitemstring(row,"ckrkd_dgcs")
     others=dw_1.getitemstring(row,"ckrkd_others")
     //更新状态(已出库 02)
     update cc_cpk set cpk_dqzt="02" where cpk_djhm=:djhm
     USING sqlca;
     //更新成品类型的当前仓储量
      update cc_cplx set cplx_ccl=(:ccl)-1 where cplx_bh=:lx;
   loop until dw_1.rowcount()=row
else
   messagebox("操作结果提示","操作失败!请检查!")
   rollback;
   return
  end if
end if
```

④ 数据窗口事件(dw_1.clicked(xpos,ypos,row,dwo))的脚本

此处编程的目的在于要实现系统检查输入的规范性，并实现数据窗口自动检索相关的类型信息的功能。

```
string dqzt,dh0
dh0=trim(em_1.text)
rq=today()
if dwo.name="ckrkd_lx" then
  djhm=dw_1.getitemstring(row,"ckrkd_djhm")
  if djhm<>"" then
    select cpk_lx,cpk_mc,cpk_ys,cpk_dw,cpk_others
    into :lx,:mc,:ys,:dw,:others
    from cc_cpk
    where cpk_djhm=:djhm;
    dw_1.setitem(row,"ckrkd_lx",lx)
    dw_1.setitem(row,"ckrkd_mc",mc)
    dw_1.setitem(row,"ckrkd_dw",dw)
      dw_1.setitem(row,"ckrkd_ys",ys)
    dw_1.setitem(row,"ckrkd_rq",rq)
```

```
    dw_1.setitem(row,"ckrkd_bh",dh0)
      dw_1.setitem(row,"ckrkd_others",others)
else
    messagebox("警告","机器号不能为空!")
  end if
end if
```

⑤　退出事件(cb_4.clicked())的脚本

```
close(parent)
```

(4) 在应用对象的 Open 事件中运行 w_cpkckrkd。

```
Open(w_cpkckrkd)
```

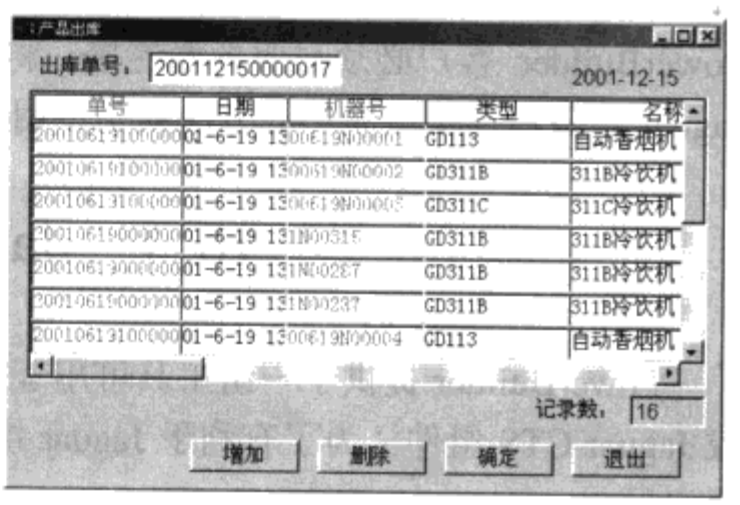


图 14-29　发布客户端应用的实例效果

完成上述工作后，就得到了为生产管理信息系统而发布的客户端应用实例效果，运行效果如图 14-29 所示。在此窗口中可以在给定的标准记账凭证格式下，按照国际统一会计凭证规定的格式插入、删除、存储记录。

本节引导读者学习了发布客户端应用的基本方法和步骤。在本节中，为生产管理信息系统 MIS 发布了客户端应用。为该系统创建了 EAServer 的客户端和连接 EAServer 的服务器、生成 EAServer 组件的代理对象、调用 EAServer 组件，并尝试实现异步调用以及处理通信错误的功能，还掌握了创建 EAServer 的客户端应用的步骤和技巧。

14.4　习　　题

1. 填空题

(1) 分布式计算体系从概念上来说，基本上可以分为以下 3 个部分，包括_____、_____、_____。

(2) 应用服务器可以提供的服务就是_____。

(3) 客户端程序是通过_____来存取应用服务器提供的服务。

(4) 客户端程序由_____、_____、_____组成。

2. 问答题

(1) 什么叫分布式计算，与传统的 C/S 结构体系相比有何优点？

(2) 远程对象与一般的不可视用户对象相比有哪些特点？

(3) 如何建立应用服务器？

(4) 什么是远程对象代理，它和远程对象有何区别？

第 15 章　建立 Jaguar 组件和客户

本章将介绍如何使用 PowerBuilder 开发 Jaguar 组件以及在 PowerBuilder 中创建和使用 Jaguar CTS 服务器的客户应用程序的方法。若要访问 Jaguar 服务器上某个组件的方法，PowerBuilder 客户必须与服务器连接，将这个组件实例化，然后调用这个组件的方法，与 Jaguar 服务器连接最简单的方法就是使用连接对象。

15.1　Jaguar 组件概述

PowerBuilder 提供了一组工具可用于开发定制类(不可视)用户对象，并把这些对象分发成 Jaguar CTS 组件。为了有利于 Jaguar 组件的开发与分发，PowerBuilder 还提供了以下几个向导：

- 开始向导(StartWizard)：用于帮助创建一个新的应用程序、定制类用户对象和工程。
- 对象向导(ObjectWizard)：用于帮助在一个现有应用程序和新工程中创建一个新的定制类用户对象。
- 工程向导(ProiectWizard)：用于帮助创建一个工程，并用它在一个现有定制类用户对象中生成 Jaguar 组件。

为了在定制类用户对象中建立并分发一个 Jaguar 组件的操作步骤如下：

(1)　使用 Jaguar 组件开始向导在一个新应用程序中创建一个用户对象。如果工作在一个现有应用程序中，也可以使用 Jaguar 组件对象向导创建这个对象。

(2)　在用户对象画板中为新建的用户对象增加函数、事件及实例变量。

(3)　测试并调试这个组件。

(4)　把这个组件分发给 Jaguar 组件。

为了测试或者分发在 PowerBuilder 中开发的 Jaguar 组件，需要创建并生成一个工程对象。这可以使用开始向导、对象向导或工程向导来完成。

为了分发一个组件，需要在工程画板中打开这个工程，并对工程设置进行相应修改，然后建立这个工程。当完成这些工作后，Jaguar 组件生成器把这个组件的接口和接口的 PowerBuilder 实现分发给目标 Jaguar 服务器。

如果想测试，可以使用活编辑(Live Editing)功能从用户对象画板中自动建立这个工程，这样就不用在工程画板中建立这个工程了。如果把用户对象画板的活编辑功能置为有效，那么每次保存用户对象时，PowerBuilder 就会为 Jaguar 组件建立工程。

当使用 Jaguar 开始向导或对象向导创建新用户对象时，可以选择创建一个 To-Do List(任务列表)。如果在向导的最后一步选中了 Generate To-Do List 复选框，向导就会在 To-Do List 中添加任务清单，以提醒开发者完成开发阶段的所有工作。

15.1.1　管理程序变量空间

Jaguar 服务器管理程序变量的方法与分布式 PowerBuilder 服务器不一样。对于每个分布式 PowerBuilder 服务器的客户连接，PowerBuilder 都维护了一个独立的客户会话。由一个客户初始化的所有服务器对象都共享相同会话，因此这些服务器对象都共享相同的全局变量和应用对象。除此之外，如果两个或多个对象的类是一样的话，那么任何共享(类)变量都被这个类的实例所共享着。Jaguar 的情况却并非如此，每个运行于 Jaguar 的 PowerBuilder 用户对象都有自己的全局变量及共享变量的副本。

分布式 PowerBuilder 服务器是面向会话的。与它不同的是，Jaguar 服务器体系结构是面向组件的，每个组件维护着自己的状态。当某个客户在服务器上初始化多个对象时，Jaguar 为每个对象独立地维护程序变量空间。PowerBuilder 对象不共享状态，它们只能通过方法 Jaguar 共享组件、服务器文件和数据库进行相互间的通信。

为了使多个客户能共享状态信息，Jaguar 提供了对共享组件(shared component)的支持。通过使用共享组件，多个客户能够共享相同的组件实例。共享组件为访问共同数据提供了一个方便的访问途径，否则，每个客户连接需要分别检索这些所需数据。共享组件还降低了数据库的访问次数，让数据库服务器进行其他处理。

15.1.2　定义组件接口

1．接口定义方法

Jaguar 把所有的组件接口都存储在 CORBA 接口定义语言模块(IDL，Interface Information Language)中。IDL 是由对象管理组(Object Management Group)定义的一个用 F 定义组件接口的标准语言。当把 PowerBuilder 定制类用户对象分发成 Jaguar 组件时，为这个对象定义的方法(函数和事件)与实例变量就会添加到这个组件的接口中。因为 Jaguar 组件生成器会自动生成 IDL，所以开发者不需要为接口编写 IDL。

2．接口包括的内容

Jaguar 组件生成器把所有为用户对象定义的公共函数都包括在组件接口中。根据组件生成时所定义的选项，生成器还可以包括公共实例变量的附加方法，也可以将用户事件作为方法。

3．方法名与方法超载

虽然 IDL 不支持方法超载，但可以将 PowerBuilder 定制类用户对象分发给具有超载方法的 Jaguar。为了不超出 IDL 的限制范围，组件生成器为要超载的方法名添加了两个下划线“_”和一个惟一后缀。如果查看为 PowerBuilder 对象生成的 IDL，就会发现为在 PowerBuilder 中超载的方法所添加的后缀。

当为具有超载方法的组件生成 Stubs 或代理对象时，Jaguar 脚本将 IDL 后缀去掉，这样客户可以使用正确的方法名对这个方法进行访问。因为 Jaguar 把两个连接下划线“_”作为保留定界符，所以，不能在要分发成一个 Jaguar 组件的定制类用户对象的函数名中使用两个连续的下划线。

4. 数据类型

Jaguar 组件分发的用户对象接口中可以使用以下数据类型：

- 标准数据类型
- 结构
- 已经分发成 Jaguar 组件的定制类用户对象。

这些数据类型可以用于公共实例变量和公共方法的参数及返回值。私有实例变量和保护实例变量及方法可以使用 PowerBuilder 支持的所有数据类型。

组件的公共接口中不支持 Any 数据类型。另外，除了 ResultSet 和 ResultSets 对象外，组件接口不能包含内置 PowerBuilder 系统对象，组件接口也不能包括可视化对象。组件方法可以传递标准数据类型数组和结构数组，并且可以使用定制类用户对象传递数组。

5. 通过引用传递

可以通过引用向组件方法传递参数。不过，其行为与非分布式应用下的行为多少有些不同。当通过引用传递参数时，所传递的变量在方法执行前被复制一份给服务器，当方法执行完后，又复制回来，这种行为对应用程序来说通常是透明的。不过，在某些情况下它会影响处理的结果。

例如，假设定义了一个名为 increment_values 的方法，这个方法有两个参数 x 和 y，这两个参数都是通过引用进行传递的。该方法增加 x 和 y 值的脚本如下：

```
x = x + 1
y = y + 1
```

客户使用下面代码调用这个方法：

```
int z
z = 1
increment_values(z,z)
```

在非分布式应用中，z 的值在方法执行完成后结果为 3。在分布式环境下，z 的结果为 2。

6. 传递只读数值

当传递一个只读参数值时，其行为与通过值传递相似，只是这个值不能被修改。数据的备份通过网络传给服务器。

7. 传递对象

在 Jaguar 组件中创建的对象可以传给客户，但是这些对象必须安装 Jaguar 组件。如果要传递一个不是 Jaguar 组件的 PowerBuilder 对象，就会产生一个运行错误。为了使用从服务器传回的组件，客户必须具有相应的 Jaguar 代理或 Stub。

因为客户应用不能向 Jaguar 传递 PowerBuilder 对象引用，所以不能像分布式 PowerBuilder 应用那样使用 PowerBuilder 对象引用从服务器推送给 PowerBuilder 客户的消息。不过可以通过在客户端使用共享对象与 Jaguar 通信来模拟这个行为。

要模拟服务器推送，客户需使用 SharedObjectRegister 和 SharedObjectGet 函数创建一个共享对象。一旦创建了共享对象之后，客户就可以向共享对象投递一个方法，传递给它

一个回调对象，当 Jaguar 服务器上的处理结束后就要通知这个回调对象。共享对象的方法向执行处理的 Jaguar 组件方法发出一个同步调用。因为共享对象运行于客户上一个独立的线索中，所以，当服务器正在处理这个同步调用时客户应用可以处理其他工作。

8. 支持 NULL 值

PowerBuilder 允许定义一个 Jaguar 组件来判断是否可以接收值为 NULL 的函数参数或返回类型。在生成 Jaguar 组件的工程的属性页中有一个 SupportNULL Values 复选框，为了在组件接口中提供对 NULL 值的支持，需要选中这个复选框。如果没有选中该复选框，客户不能在任何参数中传递 NULL 值，并且服务器不能把任何参数设成 NULL 或返回一个 NULL 值。

9. Jaguar 有效性检查

当设计一个将要分发成 Jaguar 组件的定制类用户对象时，可以让 PowerBuilder 在开发者使用了在 Jaguar 中无效的代码元素时做出警告。Jaguar 对系统类型、可视化类型、结构的公共实例变量和公共函数以及任何变量进行有效性检查。

在用户对象画板中，选择菜单条上 Design 菜单下的 Jaguar Validation 选项。当保存这个对象时，Compiler Information Messages 对话框会列出类似于下面的警告信息：

```
Information C0157=Component Validation
Warning C0158=illegal Jaguar type: "any "arg type for
Function='of_badfunc'
Warning C0158:illegal Jaguar type= 'window'arg type
for function='of_badfunc'
```

有效性检查是与当前正在编辑的对象相关联的，而不是用户对象画板。当重新打开一个对象时，它的有效性检查状态与上一次关闭时相同。

15.2　存取数据库

15.2.1　概述

1. 数据库连接性

在 Jaguar 组件中可以进行数据库存取操作。如果想要利用 Jaguar 支持连接缓冲池及事务管理的优势，必须使用下面所列的数据库接口之一与数据库进行连接。

(1) ODBC 数据库接口。

(2) Sybase SYJ 数据库接口。通过 Jaguar 提供对 Sybase Adaptive Server Enterprise 11.5 的连接。

(3) JDS 数据库接口。通过 Sun 的 Java 虚拟机提供对 JDBC 驱动程序的连接。

(4) Oracle 数据库接口。

Jaguar 在 PowerBuilder 中使用不同的 Sybase Open Client Library 软件。因此，在应用程序运行时，需要使用 SYJ 而不是 SYC 连接 Adaptive Server Enterprise 数据库。然而，PowerBuilder 开发环境不支持 SYJ 接口，因此，在开发阶段必须使用 SYC 与数据库进行连接。

2. 使用数据存储(DataStore)

在 PowerBuilder 中开发的 Jaguar 组件可以使用数据存储与数据库进行交互。数据存储是非可视化数据窗口控件。数据存储除了没有可视化属性外，其行为与数据窗口控件一样。数据存储在分布式应用中作用很大，它允许在一个远程服务器上执行数据库处理，而不用在每个客户计算机上执行这些操作。

3. 服务器与客户机共享数据

如果想给从服务器上检索出来的数据提供一个可视化界面，需要在客户机中包括一个窗口，在这个窗口中插入一个数据窗口控件。每当从服务器上检索出数据后，就要刷新该数据窗口，以显示服务器上数据存储的结果集。同样，每当用户在客户端修改了这些数据后，也需要刷新服务器上数据存储的内容，以反映客户端数据窗口的当前状态。

不能使用 ShareData 函数来完成客户与服务器之间的共享数据，必须通过编程实现服务器数据存储与客户数据窗口控件的同步化。如果想让应用程序具有数据库更新功能，就需要在客户与服务器之间传递数据窗口数据缓冲区及状态标识。

15.2.2 使用连接缓存

1. 连接缓存的作用

为了优化数据库处理，Jaguar 提供了对连接缓存的支持。利用连接缓存，Jaguar 组件可以共享为一个远程数据库服务器预分配的连接缓冲池，这样就避免了每个组件实例创建独立连接所带来的系统开销。通过建立一个连接缓存，Jaguar 服务器可以重用与相同数据资源建立的连接。连接缓存与在分布式 PowerBuilder 中支持的事务共享十分相似。

2. 工作原理

一般来说，在 PowerBuilder 应用程序与数据库进行连接时，当提交了 DISCONNECT 语句后，PowerBuilder 会从物理上终止每个数据库的连接。相反，当 PowerBuilder 组件使用了 Jaguar 组件连接缓存时，Jaguar 在逻辑上终止数据库连接，但并不真正从物理上断开这个连接，而是在连接缓存中保持连接状态，这样，其他数据库操作就可以重用这个连接。缓存中的所有连接必须具有相同的用户名、密码、服务器名以及连接库。

3. 通过用户信息存取

如果想用一组特定的用户名、密码、服务器名以及连接库从缓存中检索一个连接，不需要修改数据库存取代码，只需简单地在 Jaguar 管理器中创建一个具有所需连接属性的新缓存。在程序运行时，当组件要与数据库进行连接时，Jaguar 自动从缓存中返回一个与组件所需连接值相匹配的连接。

4. 通过缓存名存取

如果想通过指定缓存名检索一个缓存，需要在 DBParm 参数中将 CacheName 设置成要使用的缓存。通过名字存取缓存，就可以在 Jaguar 管理器中修改用户名、密码或服务器，而不需要对组件源代码做相应修改。在使用缓存名对缓存进行存取之前，需要在 Jaguar 管

理器中选中 Enable Cache-By-Name Access 选项，因为这个选项默认是没有选中的。下列示例的代码说明了如何通过名字存取缓存。

```
SQLCA.DBMS="ODBC "
SQLCA.Database= "EAS Demo DB V9"
SQLCA.AutoCommit=False
SQLCA.DBParm="ConnectString = 'DSN = EAS Demo DB V9;
UID = dba;PWD=sql',CacheName='mycache' "
```

注意： 缓存名是大小写相关的，所以，需要确保在脚本中定义的缓存名与 Jaguar 中的相匹配。

5. 无空闲连接时的处理

当缓存中无空闲连接时，通过在 DBParm 参数中设置 GetConnectionOption 可以对其行为进行控制。GetConnectionOption 可以设成以下 3 个值：

- JAG_CM_NOWAIT：没有连接返回则连接失败。
- JAG_CM_WAIT：等待，直到有空闲连接。
- JAG_CM_FORCE：分配并打开一个新的连接。新连接不在缓存中，当使用完后就被清除。

PowerBuilder 的默认值为 JAGCM_FORCE。

6. 连接释放时的处理

当一个连接被释放时有两种处理情况，ReleaseConnectionOption 的参数值决定了将采取哪种处理方法。ReleaseConnectionOption 参数可以设成以下两个值：

- JAG_CM_DROP：关闭并释放连接。如果连接来自于缓存，则将在原处新建一个连接。当某连接因为出错而不能再使用时，利用 JAG_CM_DROP 进行清除。
- JAG_CM_UNUSED：如果连接来自于缓存，则将它放回缓存。不在缓存中创建的连接将被关闭并清除。PowerBuilder 的默认值为 JAG_CM_UNUSED。

15.2.3　提供事务支持

1. Jaguar 事务支持的作用

在 PowerBuilder 中开发的 Jaguar 组件可以参与 Jaguar 事务，Jaguar 事务是一个由 Jaguar 决定其边界及输出的事务。可以通过设置让组件提供事务支持。当一个组件提供事务支持时，Jaguar 会保证这个组件的数据库操作作为一个事务的一部分进行执行。多个 Jaguar 组件可以参与一个 Jaguar 事务，Jaguar 确保所有参与组件所做的任何数据库修改操作都被同时提交或撤消。通过让组件使用 Jaguar 事务，可以保证参与一个事务的多个组件做的所有工作都会按预期的发生。

2. 组件支持事务的方式

每个 Jaguar 组件都有一个事务属性，这个属性的设置将决定组件参与 Jaguar 事务的方式。当在 PowerBuilder 中开发一个 Jaguar 组件时，可以在向导中设置这个事务属性。事务属性可以取值为以下 4 种：

- Notsupport
- SupportsTransaction
- RequiresTransaction
- RequiresNewTransaction

当事务属性设为 Notsupport 时，表示组件任何时候都不作为事务的一部分运行。如果该组件被其他运行于一个事务之中的组件激活，那么这个新产生的实例将工作于事务之外。

当事务属性设为 SupportsTransaction 时，表示组件可以在一个 Jaguar 事务的上下文中运行，但是，组件方法的执行并不需要事务来完成。如果这个组件直接由一个客户初始化，那么 Jaguar 不会开始一个事务。如果组件 A 由组件 B 初始化，并且组件 B 运行于一个事务之中，那么组件 A 将在相同的事务中运行。

当事务属性设为 RequiresTransaction 时，表示组件总是运行于一个事务之中。当组件直接由一个客户初始化时，将开始一个新的事务。如果组件 A 由组件 B 初始化，并且组件 B 运行于一个事务之中，那么组件 A 将在相同的事务中运行。如果 B 没有运行于一个事务之中，那么 A 将在一个新事务中运行。

当事务属性设为 Requires New Transaction 时，表示每次组件初始化都将开始一个新事务。如果组件 A 由组件 B 初始化，并且组件 B 运行于一个事务之中，那么组件 A 将开始一个新事务，这个新事务不受组件 B 所在事务输出的影响。如果组件 B 没有运行于一个事务之中，那么组件 A 将在一个新的事务中运行。

3. 使用事务服务上下文对象

组件方法可以调用 Jaguar 的事务状态源于控制 Jaguar 是否提交或撤消当前事务。为了让用户能够访问到 Jaguar 的事务状态原语，PowerBuilder 提供了一个事务服务上下文对象(Transaction Service Context Object)，叫做 TransactionServer。

要使用 TransactionServer 上下文对象，必须在 DBParm 参数中将 UseContextObject 设成 Yes。这就是告诉 PowerBuilder，要使用 TransactionServer 对象的方法代替 COMMIT 和 ROLLBACK 来决定这个组件在当前事务中的工作是否已经完成。

在使用这个事务上下文服务之前，还需要声明一个类型为 TransactionServer 的变量，并且调用 GetContextService 函数为该服务创建一个实例。

示例：在一个组件的 Activate 事件中可以调用 GetContextService 函数初始化 Transaction Server 服务。

```
//实例变量：
//TransactionServer ts
Integer li_rc
li_rc = this.GetContextService("TransactionServer",tS)
IF li_rc<>1 THEN
… //错误处理
END IF
```

然后，可以在组件的一个方法中对数据库进行修改。如果修改成功，则调用 SetComplete，否则，调用 SetAbort。

```
//实例变量：
//DataStore ids_datastore
Long ll_rv
```

```
...
...
ll_rv = ids_datastore.Update()
IF ll_rv = 1 THEN
  ts.SetComplete()
ELSE
  ts.SetAbort()
ENDIF
```

TransactionSener 接口提供了一些方法，可用来访问 Jaguar 的事务原语。这些方法及相应说明如表 15-1 所示。

表 15-1　TransactionServer 方法

方法	说　明
DisableCommit	表示组件的工作还未完成，当前事务不能提交。该方法返回之后，组件实例仍然保持活动状态
EnableCommit	表示该方法调用之后组件不能退出工作。如果该组件实例被释放，则允许提交当前事务
IsInTransaction	判断当前方法是否运行于一个事务之中
IsTransactionAborted	判断当前事务是否被撤消
SetAbort	表示当前组件不能完成事务中的工作，事务应该被恢复。该方法返回后，组件实例将被释放
SetComplete	表示当前组件已经完成事务中的工作，可以提交事务和释放组件实例变量

4. 自动结束

如果想让组件在每次方法调用后自动结束工作，需要把组件的 AutomaticDemarcation/ Automatic Demarcation/ Deacdvation 设置成有效，这将把组件的 tx_vote 属性设置成 False。当 Automatic Demarcation/Deactivation 设置成有效时，不需要另外单独调用 Setcomplete，因为默认执行了 SetComplete。若要想撤消事务，可以调用 SetAbort。

如果不想让组件在每个方法调用后自动结束，则需要把组件的 AutomaticDemarcation/ Deactivation 设置成无效，这将把组件的 tx_vote 属性设置成 True。当 Automatic Demarcation/Deactivation 设置成无效时，Jaguar 等待通知才结束事务。因此必须在程序中单独调用 SetComplete 退出工作。

5. COMMIT 与 ROLLBACK

可以选择将 TransactionServer 上下文对象设置成无效，不为组件定义 Jaguar 事务状态，而是使用 COMMIT 和 ROLLBACK 语句。提供这项功能是为了不用修改代码就可以把 PowerBuilder 对象移植到 Jaguar 中。要将 TransactionServer 上下文对象设置成无效，需要在 DBParm 参数中把 UsecontextObject 设置成 No。这样，COMMIT 语句等效于 SetComplete，ROLLBACK 语句等效于 Setaboft。

在 TransactionServer 上下文对象无效的非事务化组件中，COMMIT 语句不会调用 SetComplete，ROLLBACK 语句也不会调用 SetAbort。例如，如果将事务类型设置成 Not Supported，设置 Automatic Demarcation/Deactivation 为无效，并且将 UseContextObiect 参数

设置成 No，当执行 COMMIT 语句时，PowerBuilder 虚拟机不会提交 SetComplete，当执行 ROLLBACK 语句时，也不会提交 SetAbort。在这种情况下，Jaguar 永远都不会释放组件，因为它一直等待 SetComplete 和 SetAbort 的调用。

如果某个组件从不执行数据库存取操作，那么若将其 Automatic Demarcation/Deactivation 置为无效，就必须使用 TransactionServer 对象调用 SetComplete，让这个组件退出工作，否则，这个组件将永久处于活动状态。

6. 事务与组件生命周期

Jaguar 的事务模型和组件的生命周期紧密地集成在一起。参与一个事务的组件实例不会释放，除非事务结束，或者这个组件指示它的工作已经完成或者必须恢复。一个实例的活动状态时间与它参与事务的起始时间是一致的。

15.2.4 执行检索操作

使用数据存储执行检索操作，必须首先在脚本中创建一个数据存储实例，并且把数据窗口对象赋给这个数据存储，然后为数据存储设置事务对象。一旦这些设置步骤完成之后，就可以检索数据到数据存储中，对检索出来的结果集执行打印或其他一些处理工作。

示例 1：此例说明如何利用数据存储在远程服务器上打印数据。服务器组件 uo_employees 有一个函数 print_employees，此函数生成数据存储的一个实例 ds_datastore，然后打印这个数据存储的内容。

这是 print_employees 的函数声明：

```
print_employees()returns integer
```

下面是 print_employees 函数的脚本：

```
datastore ds_datastore
int li_rc
ds_datastore=create datastore
ds_datastore.dataobject=d_empdata
ds_datastore.SetTransObject(SQLCA)
ds_datastore.Retrieve()
li_rc=ds_datastore.Print()
return li_rc
```

注意： 示例 1 不能在 UNIX 平台上运行。因为在 UNIX 上，Jaguar 采用的是 PowerBuilder 运行环境的无窗口版本，它不支持打印图形化操作。

示例 2：此例说明了如何利用数据存储在一个服务器组件中检索数据。这个服务器组件 uo_customers 有一个函数 retrieve_custlist。retrieve_custlist 函数生成数据存储的一个实例 ds__datastore，然后使用这个数据存储检索 Customer 表中的所有记录。检索出数据之后，retrieve_custlist 函数把数据传给客户程序。retrieve_custlist 函数有一个参数 customers，这个参数定义成一个类型为 st_custlist 结构的数组。st_custlist 与 d_custlist 的构成相同。d_custlist 是用来进行数据库存取的数据窗口对象。retrieve_custlist 返回检索出来的记录数，其类型为 Long。

这是 retrieve_custlist 的函数声明：

```
retrieve_custlist(REF st_custlist customersC))returns long
```

下面是 retrieve_custlist 函数的脚本：

```
datastore ds_datastore
long ll_rowscount
ds_datastore = create datastore
ds_datastore.dataobject x,d_custlist
ds_datastore.SetTransObject(SQLCA)
IF ds_datastore.Retrieve()<> -1 THEN
  ll_rowcount = ds_datastore.RowCount()
END IF
customers = ds_datastore.obiect.data
destroy ds_datastore
return ll_rowcount
```

处理结束时，retrieve_custlist 清除所用的数据存储，然后向客户程序返回检索出来的记录数。

15.2.5　执行更新操作

1．数据窗口同步化

在传统的客户/服务器应用下，数据库更新是由运行在客户计算机上的单一的应用程序来启动的。PowerBuilder 可以自动地管理数据窗口的状态信息。但是，在分布式应用下，情况有些不同。因为应用程序被分成客户部分和服务器部分，因此需要编写代码让客户端数据窗口控件缓冲区中的数据及状态标识与服务器端数据存储缓冲区中的内容保持同步。

PowerBuilder 为分布式应用下的数据窗口与数据存储之间的同步提供了以下 4 个函数：

- GetFullState
- SetFullState
- GetChanges
- SetChanges

这些函数不仅在分布式应用中十分有用，在非分布式应用中，当多个数据窗口或数据存储需要同步时也能发挥它们的作用。

2．传递数据窗口缓冲区及状态标识

为了使客户上的数据窗口控件与服务器上的数据存储保持同步，一旦数据发生了变化，必须从客户向服务器或从服务器向客户传递数据窗口、数据缓存器及状态标识。无论数据变化发生在客户端还是服务器端，其基本的处理过程是一样的。

为了从一个数据窗口向另一个数据窗口提供完全的状态信息，需要执行以下操作：

(1)　调用 GetFullState 函数捕获源数据窗口的当前状态。

(2)　调用 SetFullState 函数向目标提供源数据窗口的状态。

为了从一个数据窗口向另一个数据窗口提供变化了的数据，需要执行以下操作：

(1)　调用 GetChanges 函数捕获源数据窗口所做的修改。

(2)　调用 SetChanges 函数向目标提供源数据窗口所做的修改。

可以调用 SetChanges 函数向一个空数据窗口提供源数据窗口所做的修改。目标数据窗

口不需要包含以前检索操作产生的结果集。但是，这个数据窗口对数据窗口的定义必须具有存取权，这意味着在调用 SetChanges 函数之前必须把数据窗口对象赋给目标数据窗口。

注意： 同步化只能发生在一个源与一个目标数据窗口之间，而不能发生在一个源与多个目标数据窗口，或者多个源与一个目标数据窗口之间。

3. blob 存储数据窗口状态

当调用 GetFullState 或 GetChanges 函数时，PowerBuilder 会在一个 blob 中返回数据窗口状态。GetFullState 函数返回的 blob 提供了重新创建数据窗口所需的一切东西，包括数据缓冲区、状态标识和完全的数据窗口定义。GetChanges 函数返回的 blob 只为修改的或删除的行提供了数据缓冲区和状态标识。

4. 更新后的同步

通常 Update 函数在成功完成更新操作后就把更新标识复位了。因此，当在服务器中调用 Update 函数时，其数据存储的状态标识自动地进行复位。但是，相应的客户端数据窗口控件的更新标识没有被复位。所以，如果服务器数据存储的 Update 函数执行成功，则需要在客户端调用数据窗口的 ResetUpdate 函数，将状态标识复位。

15.2.6 传递结果集

PowerBuilder 提供了 ResultSet 和 ResultSets 两个系统对象。这两个系统对象可用来从运行于 Jaguar 上的组件中获取结果集以及从作为 Jaguar 组件的 PowerBuilder 用户对象中返回结果集。设计这两个系统对象是为了简化事务服务器与数据存储对象之间结果集的传递。它们不是为数据库更新而设计的。使用数据存储对象的 CreateFrom 和 GenerateResultSet 函数，可以把存储于这两个系统对象中的结果集转换到数据存储对象中，也可以从数据存储对象获取结果集到这两个系统变量中。

返回结果集的组件方法采用 TabularResults 模块。单个结果集以 TabularResults::ResultSet 结构返回。多个结果集以一组连续的 ResultSet 结构返回，其数据类型为 TabularResults::ResultSets。

1. 从 PowerBuilder 客户中存取 Jaguar 组件结果集

当在 PowerBuilder 中为一个返回 TabularResults=ResultSet 的 Jaguar 组件方法生成一个 Jaguar 代理对象时，这个代理服务器上的方法返回一个 PowerBuilderResultSet 对象。用返回多个结果集的方法返回一个 PowerBuilderResultSets 对象。

在 PowerBuilder 浏览器(Browser)的 Proxy 选项卡中可以查看 Jaguar 代理对象的属性和方法。例如，Sybase 虚拟大学实例中 SVUBookstore 组件有几个方法返回 TabularResults=ResultSet 值。当为这个组件创建一个 Jaguar 代理对象时，以下是 GetMajors 方法作为一个 PowerBuilder 用户对象函数在浏览器中的显示：

```
SVUBookStore.GetMajors()returns ResultSet
```

从 PowerBuilder 客户中可以对结果集进行存取，其方法是创建这个组件的一个实例，调用方法，然后利用 CreateFrom 函数把结果集中的数据装入一个数据存储中。

示例：创建一个 SVUBookstore 组件的实例并调用 GetMajors 方法。

```
SVUBookstore lcst_mybookstore
Resultset lrs_resultset
Datastore ds_local
Integer li_rc
//myconnect 是一个连接对象
li_rc=myconnect.CreateInstance(lcst_mybookstore)
IF li_rc<>0 THEN
   MessageBox("创建一个实例",string(li_rc))
   Myconnect.DisconnectServer()
   RETURN
END IF
lrs_resultset t lcst_mybookstore.GetMajors()
ds_local=CREATE datastore
ds_local.CreateFrom(lrs_resultset)
```

2. 从 Jaguar 组件中返回结果集

从一个要分发给 Jaguar 的 PowerBuilder 用户对象中返回结果集，需要把一个函数的参数或返回值的数据类型设置成 ResultSet(对单个结果集而言)或 ResultSets(对多个结果集而言)。当这个用户对象已经分发成 Jaguar 组件时，ResultSet 或 ResultSets 返回值将会被显示在组件的 IDL 接口中，其数据类型为 TabularResults=ResultSet 和 TabularResults=ResultSets。

示例：创建一个数据存储对象，并在其中检索数据，然后调用 GenerateResultSet 函数生成一个结果集，返回给一个客户。

```
datastore ds_datastore
resultset lrs_resultset
integer li_rc
ds_datastore;
create datastore
ds_datastore.dataobject="d_empdata"
ds_datastore.SetTransObject(SQLCA)
IF ds_datastore.Retrieve() =-1 THEN
   … //出错报告并返回
END IF
li_rc = ds_datastore.GenerateResultSet(rs_resultset)
IF li_rc<>1 THEN
… //出错报告并返回
END IF
return lrs_resultset
```

15.2.7　支持实例缓冲池

Jaguar 组件可以支持实例缓冲池。实例缓冲池允许 Jaguar 客户重用组件实例，避免了由于重复创建组件实例所导致的资源耗尽，从整体上提高了 Jaguar 服务器的性能。

1. 在向导中设置缓冲池选项

当使用 PowerBuilder 向导创建 Jaguar 组件时，可以对缓冲池选项进行设置。缓冲池选项可以设为 Supported 或 Notsupposed。

当缓冲池选项设为 Supported 时，在每个客户使用后，组件都要放到缓冲池中。组件的 CanBePooled 事件不会被触发。设置该选项会将组件的缓冲池属性设置成 True。

如果组件的 Automatic Demarcation/Deactivation 被设成有效，则在每次方法调用之后，实例都会放到缓冲池中。如果 Automatic Demarcation/Deactivation 被设成无效，每当组件调用 TransactionServer 上下文对象的 SetComplete 方法后，实例才会放到缓冲池中。

当缓冲池选项设为 Not supported 时，每次客户使用后，组件默认都不放到缓冲池中。通过编写 CanBePooled 事件代码，可以改变这种默认行为。在 CanBePooled 事件中，可以编程定义是否某个组件实例要被缓冲。CanBePooled 事件在每次客户使用后都将被触发。

2. 控制缓冲实例的状态

如果创建了一个支持实例缓冲池的 Jaguar 组件，该组件在每次客户使用了被缓冲的实例后，可能需要复位自己的状态。在组件的生命周期中 Jaguar 会触发以下一个或多个事件。利用这些触发事件，可以对组件的状态进行控制：

- Activate
- CanBePooled
- Deactivate

当组件的缓冲池选项被设置成 Supported(缓冲池属性设成 True)时，可能需要为 Activate 和 Deactivate 事件编写脚本来复位缓冲后的组件状态。如果组件通过一个实例变量、共享变量或全局变量维护组件的状态，这种复位操作就是必需的了。

当组件的缓冲池选项被设置成 Not Supported(缓冲池属性设成 False)时，可以在 CanBePooled 事件中指定一个组件实例是否被缓冲。如果为 CanRePooled 事件编写了脚本，可能同时还需为 Activate 和 Deactivate 事件编写脚本来复位组件的状态。如果没有在 CanBePooled 事件中编写脚本，那么组件实例将不会被缓冲。

Jaguar 组件的开始向导和对象向导自动为一个将被分发成 Jaguar 组件的定制类用户对象加入 Activate 和 Deactivate 事件。而 CanBePooled 事件则需自己手动加入，这需要把事件映射成正确的 PBM 代码。Activate、Deactivate 和 CanBePooled 事件的 PBM 代码分别为 PBM_COMPONENT_ACTIVATE、PBM_COMPONENT_DEACTIVATE 和 PBM_COMPONENT_CANBEPOOLED。

注意： 当实例缓冲池被设置成有效时，组件的 Constructor 和 Destructor 事件只激活一次，这两个事件不会在每次新客户使用组件实例时被激活，因此对被缓冲的组件实例状态的复位操作必须在 Activate 和 Deactivate 事件中实现，而不能在 Constructor 和 Destructor 事件中实现。

3. 组件的生命周期

为了理解实例缓冲池的工作过程，需要了解组件实例的生命周期。下面是组件生命周期的过程：

(1) 组件通常在第一次方法调用时被初始化。如果这个组件是在 PowerBuilder 中被开发的，那么 Jaguar 会创建一个新的 PowerBuilder 会话，让这个组件在其中运行。

(2) PowerBuilder 会话创建代表 Jaguar 组件的 PowerBuilder 非可视化对象。对象的创建将触发 Constructor 事件。

(3) 当对象已经被初始化后，Jaguar 触发该对象的 Activate 事件，通知它将要被一个

新客户使用了，组件必须保证它已经处于准备运行状态。

(4) Jaguar 然后执行由客户调用的组件方法。

(5) 当组件表示它的工作已经结束时，Jaguar 触发 Deactivate 事件让组件清除它的状态。如果组件的 Automatic Demarcation/Deactivation 设置为有效的，那么每次方法调用后都自动触发 Deactivate 事件。如果组件的 AutomaticDemarcation/Deactivation 设置为无效的，那么当组件调用 TransactionServer 上下文对象的 SetComplete(或 SetAbort)方法时才触发 Deactivate 事件。

(6) 如果已经选择了 Not Supported 缓冲池选项(或者将组件的缓冲池属性设为 False)，并且也为 CanBePooled 事件编写了事件脚本，那么 Jaguar 将触发这个事件，看看这个组件是否可以被缓冲。CanBePooled 事件允许组件实例有选择地接受或拒绝被缓冲。CanBePooled 事件的返回值决定了这个组件实例是否将被缓冲。返回值为 1，表明可以被缓冲；返回值为 0，表示不被缓冲。如果 CanBePooled 事件没编写事件脚本，那么这个实例默认不被缓冲。

(7) 如果一个实例在结束工作后不被缓冲，那么 Jaguar 触发 Destructor 事件，清除 PowerBuilder 对象并终止会话的运行。

15.3　使用共享组件

为了让客户能够共享状态信息，Jaguar 提供了对共享组件的支持。共享组件允许多个客户共享同一个组件实例。当向 Jaguar 分发一个 PowerBuilder 非可视化对象时，可以将这个对象设置成一个 Jaguar 共享组件。

共享组件为访问共同数据提供了一个方便的访问途径，否则，每个客户连接都需要分别检索这些数据。共享组件还降低了数据库的访问次数，提高了数据库服务器的可使用性。Jaguar 共享组件像 PowerBuilder 共享对象一样具有很多功能。不过，Jaguar 共享组件与 PowerBuilder 共享对象不同。分发给 Jaguar 的 PowerBuilder 组件可以作为 Jaguar 共享组件的客户，也可以作为 PowerBuilder 共享对象的客户。Jaguar 共享组件可以被不是在 PowerBuilder 中开发的组件和客户所访问。

不过，Jaguar 并不把 PowerBuilder 共享对象当作是 Jaguar 共享组件。因此，用于操纵 PowerBuilder 共享对象的函数不能用于 Jaguar 共享组件。如果在运行于 Jaguar 上的 PowerBuilder 组件中调用这些函数，将会导致出错。

15.3.1　创建一个共享组件

可以在 PowerBuilder 向导中将一个组件标识为共享的，其效果与在 Jaguar 管理器的 ComponentProperties 对话框的 Instance 标签下将组件设为共享是一样的。客户及其他服务器组件访问一个共享组件的行为就像访问其他组件一样。

15.3.2 创建一个服务组件

可以在 PowerBuilder 向导中将一个共享组件标识为服务组件。一个服务组件为 Jaguar 客户及其他 Jaguar 组件执行后台处理。Jaguar 在服务器启动时装载服务组件。

当在一个 PowerBuilder 向导中把一个组件标识为服务组件后，PowerBuilder 在分发时会把这个组件作为一个服务安装到Jaguar中。其效果与通过修改服务器的com.sybase.jaguar.server.services 属性把组件添加到服务器的服务列表中是一样的。

PowerBuilder 向导为服务组件添加 Start、Run 和 Stop 共 3 个函数。这些函数可用来控制与该服务相关的后台处理行为。

(1) Start 函数

Jaguar 服务器在服务组件已经被装载后调用 Start 函数。可以在这个函数中添加逻辑，执行一些服务的初始化工作。

(2) Run 函数

Jaguar 服务器在第一次调用 Start 函数返回后调用 Run 函数。可以利用 Run 函数执行一些重复的后台处理操作。Run 函数必须定期调用 JagSleep 函数把它自己挂起。JagSleep 函数释放 CPU 去执行其他任务。要使用 JagSleep 函数，必须在 PowerBuilder 中为 JagSleep 声明一个外部函数，其语法格式如下：

```
subroutine JagSleep(Long seconds)LIBRARY" libjdispatch.dll"
```

(3) Stop 函数

使用 Stop 函数可以终止 Run 函数中的后台处理代码的运行。可以为服务建立一个服务管理器客户程序，在其中调用 Stop、Start 及 Run 函数，这样就可以不需为重新运行这个服务而重新启动 Jaguar 服务器了。还可以在 Stop 函数脚本中清除 Start 函数分配的所有资源。

15.3.3 调用另一个服务器组件的方法

Jaguar 允许一个服务器组件的方法调用另一个服务器组件的方法。这个被调用的服务器组件不一定必须是 PowerBuilder 组件，它可以是用 Jaguar 所支持的任何一种语言实现的。

1. 在当前服务器中访问组件

要在当前服务器中访问另一个 Jaguar 组件的方法，可以使用 Connection 对象与这个组件进行通信，就像 PowerBuilder 客户与这个组件进行通信一样。除此之外，也可以使用 PowerBuilder 提供的事务服务上下文对象 TransactionServer。TransactionServer 接口提供了一个称为 CreateInstance 的方法，它可以访问本地的其他组件。CreateInstance 使用与调用它的组件相同的用户和密码信息。

在使用事务服务上下文对象之前，必须声明一个类型为 TransactionServer 的变量并且调用 GetContextService 函数为这个服务创建一个实例。

示例：在组件的 Activate 事件中调用 GetContextService 初始化 TransactionServer 服务。

```
//实例变量:
//TransactionServer ts
Integer rc
```

```
rc = this.GetContextService("TransactionServer",ts)
IF rc <> 1 THEN
  … //错误处理
ENDIF
```

然后在组件的一个方法中调用 CreateInstance 初始化第 2 个组件并调用它的一个方法。

```
//第二个组件的实例变量:
//nvo_comp2 mycom2
Integer rc
rc = ts.Createinstance(mycomp2, "mypackage/nvo_comp2")
IF rc <> 0  THEN
  … //错误处理
ELSE
  mycomp2.method1()
END IF
```

2. 在不同服务器中访问组件

访问不同服务器上的服务器组件的过程与从 PowerBuilder 客户中访问服务器组件的过程基本相同。访问不同服务器上的 JaSuar 组件，必须创建一个连接对象，为这个连接对象设置相应属性，然后调用 ConnectToServer 函数。

15.3.4　访问组件属性

1. ContextKeyword 服务对象

使用 ContextKeyword 服务对象可以获得组件属性值。在使用 ContextKeyword 服务对象之前，必须把 Keyword 作为服务名调用 GetContextKeywords 函数创建一个指向这个对象的引用。

2. PowerbuilderJaguar 属性

表 15-2 所示为 Jaguar 组件的 PowerBuilder 定制类用户对象的组件属性。组件属性都以 com.sybase.jaguar.components 字符串为前缀。所有组件属性的属性值都显示在 Jaguar 管理器 Component Properties 对话框的 All Properties 选项卡中。一些属性也映射到其他选项卡中的项目，如表 15-2 所示。

表 15-2　PowerBuilderJaguar 组件属性

属性	描　述	显示位置
Bind.thread	指出是否必须在创建线索时调用组件实例，有效值为 True 或 False。当 PowerBuilder 组件通过方法调用对数据存储状态进行维护时，此属性必须设为 True	Instances(Bind.read 选项)
interfaces	标识组件实现的接口。这是一个用逗号分隔的 IDL 接口名列表，每个名字的格式都为 module=interface	General
name	定义组件名，其值的形式必须为 package/component	General(只有组件部分)
pb.appname	定义 PowerBuilder，应用程序名	General
pb.class	定义 PowerBuilder 定制类用户对象名	General

续表

属性	描　述	显示位置
pb.cookie	提供了一个用于构造库列表路径的数值。这个路径的格式为：Repository\Component\package\component\Ccookie	所有属性
pb.isdebuggable	指出是否可以在 PowerBuilder 调试程序中调试这个组件	所有属性
pb.Hbrarylist	定义 PowerBuilder 库列表。当一个库名前面有一个 S 符号时，Jaguar 就认为它指定的存放位置是相对 Jaguar Repository 路径而言的。如果没有 S 符号，Jaguar 就认为这个库名是一个绝对路径	General
pb.trace	定义记录组件活动的跟踪选项	所有属性
pooling	指出是否组件被缓冲。当这个属性设为 True 时，组件总是被缓冲而且 CanBePooled 事件不会被触发；当这个属性设为 False 时，CanBePooled 事件被触发，允许有选择地拒绝缓冲。如果 tx_vote 属性为 False，那么组件在每次方法调用之后都将被缓冲，否则它将在事务结束时被缓冲	Instances
sharing	指出是否是一个共享组件。当这个属性设为 True 时，所有客户共享一个组件实例。共享组件没有提供缓冲池选项	Instances
thread.safe	指出是否可以同时处理多个调用。除了只允许一个组件实例在同一时间处于活动状态之外，这个属性必须设为 True	lnstances(Concurrency 选项)
timeout	定义一个活动组件实例在方法调用之间多长时间处于空闭状态之后就要自动结束工作	所有属性
tx_timeout	定义一个 Jaguar 事务的最长持续时间。每次方法返回时，Jaguar 要检查它是否超时	所有属性
tx_type	指出组件是否要参与 Jaguar 事务和参与方式。有效值为 not_supported、supports、requires 和 requires new	Transactions
tx_vote	指出组件是否支持自动结束。当 tx_vote 为 Tme 时，组件必须控制事务状态，并且显式调用 TransactionServer 服务对象的方法来终止组件。当 tx_vote 为 False 时，Jaguar 在每次调用方法之后自动结束组件	Transactions
type	定义组件的类型。Jaguar 将 PowerBuilder 9 对象的 type 值设为 pb。为了允许用户将组件分发给不同版本的 PowerBuilder 虚拟机，可以在 Jaguar 中定义支持相应虚拟机的 DLL 名。例如：可以将 PowerBuilder 9 对象的组件类型定义成 pbjag90(代替 pb)	General

表 15-3 所示为 Jaguar 组件的 PowerBuilder 定制类用户对象的服务器属性。服务器属性都以 com.sybase.jaguar.server 字符串为前缀。所有服务器属性的属性值都显示在 Jaguar 管理器 Server Properties 对话框的 All Properties 选项卡中。

表 15-3　PowerBuilder Jaguar 服务器属性

属　性	描　述	显示位置
packages	分发在服务器上的组件包(package))列表	与服务器的 Packages 目录中所列的相同
services	提供了一个将与服务器一起自动启动的服务组件列表(用逗号分隔)	ServerProperties 对话框中所有属性选项卡

15.4　组件测试与调试

在测试与调试一个在 PowerBuilder 中开发的 Jaguar 组件之前，必须建立一个新工程。在这个新建的工程中列出所有涉及的对象，定义将要包含所有生成组件的输出库的库名。一旦建立了这个工程，就可以运行一个访问该组件的客户程序从而对这个组件进行测试，也可以使用 PowerBuilder 调试工具对这个组件进行调试。

可以使用 Jaguar 组件的开始、对象和工程向导为建立 Jaguar 组件而创建工程。在这些向导中可以选择所需建立的组件以及设置控制工程行为的属性。当创建一个工程之后，还可以使用工程画板对工程的属性进行修改。

要把一个完成的组件分发给 Jaguar，必须在工程画板中打开这个工程，对工程属性进行相应的设置，然后生成这个工程。如果只是简单地想测试或调试一个组件，情况则不一样。如果要进行测试，可以使用 PowerBuilder 的“活编辑”功能。利用活编辑可以在用户对象画板中自动生成这个工程。当允许进行活编辑时，PowerBuilder 将在每次保存相应用户对象的时候为 Jaguar 组件生成这个工程。生成器并不是把 PBD 分发给 Jaguar，而是告诉 Jaguar 服务器如何访问包含所需对象定义的 PBL。若把一个用户对象的活编辑设置成有效，需完成以下步骤：

(1)　创建一个工程，其中包括生成 Jaguar 组件所需的用户对象。

(2)　根据需要修改工程的活编辑库列表。当用一个远程计算机上的 Jaguar 服务器来测试组件时，必须告诉 Jaguar 如何定位所需的 PBL。其做法是，修改工程属性窗口的 Advanced 属性页中 Live Editing Library List 文本框中的库列表。所定义的库列表必须使用通用命名规则包含完整路径，格式为：\Xservename\sharename\pathXf'fie。活编辑库列表默认与应用库列表相同。如果是本地 Jaguar 服务器，则不需要修改活编辑库列表。

(3)　指定将要用来在用户对象画板中生成这个组件的工程。必须在 Jaguar Proiect 文本框中输入这个工程名。Jaguar Project 框在用户对象属性窗口的 General 属性页上，所输入的工程名必须满足以下 3 个条件：

- 必须是一个 Jaguar 组件工程
- 必须包括当前在用户对象画板中打开的用户对象
- 工程的库列表必须与当前应用库列表相匹配

在用户对象画板中，选择 File 菜单下的 Save 项命令可以生成 Jaguar 组件。当在用户对象画板中建立一个工程时，Jaguar 组件生成器将执行以下操作：

(1) 生成描述所要保存的非可视化对象的 CORBA IDL。再用生成的 IDL 建立 Stubs 和 Skeletons。IDL 文件、stubs 和 skeletons 的文件名是根据对象名生成的。新的 IDL 文件保存在 Jaguar 安装目录下的 Repository 子目录中。

(2) 生成一个描述 Jaguar 组件属性的 PROPS 文件。这个 PROPS 文件存储在 Jaguar 安装目录的以下子目录中：

```
Repository\Component\package-name
```

当在用户对象画板中建立一个工程时，除了不为组件生成 PBD 文件外，组件生成器就像分发组件一样建立该组件。另外，它还将 pb.live_edit 属性设置成 True 并且把开发者为活编辑指定的库列表赋给 pb.librarylist 属性。

如果工程建立过程中出错，那么 PowerBuilder 会在它的标准编译器错误对话框中显示出错信息。如果实例缓冲池被设置成有效，那么生成器会在当前的组件建立过程中令缓冲池无效。“活编辑”不支持缓冲池，因为，如果包含用户对象的 PBL 被 Jaguar 服务器锁住，PowerBuilder 则不能保存这个对象。

15.5 向 Jaguar 分发组件

要把组件分发给 Jaguar，需要创建并生成一个新工程。在新工程中加入组件中的所有对象，并指定一个输出库，以保存所生成的组件。可以使用 Jaguar 组件的开始、对象或工程向导帮助创建工程，生成 Jaguar 组件。利用向导可以选择所需生成的组件，设置及控制工程的行为。当创建了一个工程后，可以使用工程画板修改工程设置。向 Jaguar 分发组件时，需要打开所建工程，然后选择 Design 菜单下的 Build Project 命令。当向 Jaguar 分发组件时，组件生成器执行以下操作：

(1) 生成描述所要保存的非可视化对象的 CORBA IDL，再用生成的 IDL 建立 Stubs 和 Skeletons。IDL 文件、Stubs 和 Skeletons 的文件名是根据对象名生成的。新的 IDL 文件保存在 Jaguar 安装目录下的 Repository 子目录中。

(2) 生成一个描述 Jaguar 组件属性的 PROPS 文件。这个 PROPS 文件存储在 Jaguar 安装目录的以下子目录中：

```
Repository\Componentkpackage-name
```

(3) 为分发组件生成一个或多个 PBD 文件。生成的 PBD 文件保存在 Jaguar 安装目录的以下子目录中：

```
Repository\Component\packagelcomponent\cookie
```

其中 cookie 表示组件生成的次数。如果库名没有包含路径，那么组件生成器会在库名前加一个$符号，通常 Jaguar 使用组件的最新版本。

15.6　Jaguar 客户概述

PowerBuilder 应用程序可以作为 Jaguar CTS 服务器的客户。若要访问 Jaguar 服务器上某个组件的方法，PowerBuilder 客户必须与服务器连接，将这个组件实例化，然后调用这个组件的方法。PowerBuilder 为简化 Jaguar 客户的开发提供了两个向导：

- 连接对象向导
- Jaguar 代理向导

连接对象向导可帮助生成与 Jaguar 服务器连接所需的代码。Jaguar Proxy 向导帮助创建工程，在这个工程中建立客户所需访问的 Jaguar 组件的代理对象。若建立和分发一个 Jaguar 客户，需完成以下几个步骤：

(1) 使用 Jaguar 连接对象向导来创建一个继承连接对象的标准类用户对象。然后可以在脚本中使用这个对象建立一个连接。

(2) 使用 Jaguar 代理向导为建立代理对象创建一个工程，然后生成这个代理对象。

(3) 创建用户界面所需的窗口、菜单及脚本。

(4) 编写所需代码创建 Jaguar 组件实例，并且在客户中调用一个或多个组件方法。

(5) 测试和调试客户。

(6) 分发应用。

15.7　与 Jaguar 服务器连接

与 Jaguar 服务器连接最简单的方法就是使用连接对象。连接对象是一个处理与服务器之间通信的非可视化对象。使用连接对象连接 Jaguar 服务器与连接一个分布式 PowerBuilder 服务器应用很相似。

可以自己编写代码实现与 Jaguar 服务器的连接，也可使用连接对象向导帮助完成。本节后面部分将介绍如何使用连接对象与 Jaguar 服务器进行连接。

15.7.1　编写代码

1. 声明连接变量

连接对象不是内置全局变量，因此需要声明一个类型为 Connection 的变量。一般情况下，将这个变量声明为一个全局变量或实例变量。

2. 建立连接

要建立一个与 Jaguar 服务器的连接，需要执行 PowerScript 语句完成以下操作：

(1) 使用 Create 语句初始化连接对象。

(2) 设置连接对象的属性。

(3) 调用 ConnectToServer 函数与 Jaguar 服务器建立一个连接。

(4) 检查错误。

上面这些语句既可以在一个脚本中完成，也可以在多个脚本中完成。但是它们必须按以上的顺序执行。

示例：初始化 myconnect 连接对象，并设置这个对象的属性(Jaguar 通信驱动程序、Jaguar 服务器的主机名和端口号、默认组件包)。然后调用 ConnectToServer 函数建立连接并检查连接是否成功。

```
//全局变量;
//connection myconnect
Long ll_rc
myconnect.create connection
myconnect.driver = "jaguar"
myconnect.location = "Jagserver1: 9000"
myconnect.application = "PB_pkg_1"
myconnect.userID = "bjones"
myconnect.password = "mypass"
ll_rc = myconnect.ConnectToServer()
IF ll_rc<> 0 THEN
  MessageBox("连接失败",ll_rc)
ENDIF
```

3. 设置连接对象属性

与 Jaguar 服务器连接所需设置的连接属性和与分布式 PowerBuilder 服务器连接所需设置的连接属性不同。与 Jaguar 进行通信时所需设置的连接对象属性及说明如表 15-4 所示。

4. 建立多连接

PowerBuilder 允许实例化多个连接对象，可以在一个客户应用中建立多个连接。例如：实例化两个独立的连接对象，把一个客户与两个不同的 Jaguar 服务器连起来。

表 15-4 与 Jaguar 连接时连接对象属性的设置

属性	说　明	实例
Application	Jaguar 组件使用的默认组件包	"PB_pkg_1"
ConnectString	不在 Jaguar 中使用	
Driver	Jaguar 驱动器名	"Jaguar"
Location	Jaguar 服务器的主机名和端口号(由分号分隔)。如果要利用 Jaguar 的负载平衡和故障平息功能，还可以用分号定义一组服务器列表	
Password	Jaguar 密码	" mypass"
UserID	Jaguar 用户标识	" bjones"
Options	一个或多个 Jaguar ORB 属性设置	"RBLogFile = 'jaglog.log'"

15.7.2 使用向导

当在连接对象向导中选择 Jaguar 作为连接类型时，PowerBuilder 创建一个继承连接对象的标准类用户对象。在向导中定义连接对象属性，指定是否在注册表、INI 文件或脚本中

提供连接信息。连接对象的 Constructor 事件调用函数 of_getconnectioninfo，这个函数可以从指定的数据源中获取存储起来的连接信息。

一旦使用连接对象向导创建了一个连接对象，需要执行 PowerScript 语句完成以下几个操作：

(1) 使用 Create 语句初始化连接对象。

(2) 调用 ConnectToServer 函数与 Jaguar 服务器建立一个连接。

(3) 检查错误(可选)。

由上面几个步骤可以看出，当利用向导创建一个连接对象时，不需手动编写代码设置连接对象的连接属性。

示例：初始化由向导创建的 n_myclient_connect 对象的 myconnect 实例，然后调用 ConnectToServer 函数建立连接并检查连接是否成功。

```
Long ll_rc
myconnect=create n_myclient_connect
ll_rc=myconnect.ConnectToServer()
IF ll_rc<>0 THEN
  MessageBox("连接失败",ll_rc)
END IF
```

利用向导也可以在一个客户应用中建立多个连接。如果想把一个客户与两个不同的 Jaguar 服务器连接起来，可以重新运行连接对象向导，用不同的连接属性创建一个新用户对象。

15.8　生成 Jaguar 代理对象

每个 Jaguar 组件在客户应用程序中都有一个相应的代理对象。对 Jaguar 组件的访问都是通过这个 Jaguar 代理对象实现的。在为一个 Jaguar 客户生成代理对象之前，必须创建一个新工程。新工程中列出了全部所需的对象，并且指定一个包含所生成的代理对象的输出库。

Jaguar Proxy 向导可用来帮助创建生成 Jaguar 代理的工程。利用这个向导可以与一个 Jaguar 服务器进行连接，选择客户所需访问的组件。然后可使用工程画板修改工程设置并建立代理库。

15.9　调用组件方法

当与 Jaguar 服务器的连接已经建立起来并且生成了一个代理对象后，客户应用程序就可以开始使用服务器组件了。

15.9.1　调用方法

要调用一个组件方法，必须执行 PowerScript 语句完成以下操作：

(1) 使用 CreateInstance 方法创建一个组件实例。

(2) 调用方法。

示例 1：实例化服务器上的一个组件并调用这个组件方法。在此例中，CreateInstance 函数没有指定一个组件包，因此 Jaguar 使用在 Application 属性中定义的默认组件包。

```
//全局变量:
//connection myconnect
uo_customer iuo_customer
string ls_custid
Long ll_rc
ls_custid=Trim(sle_custid.text)
ll_rc=myconnect.CreateInstance(iuo_customer)
IF ll_rc<>0 THEN
  MessageBox("创建实例失败",ll_rc)
END IF
IF iuo_customer.retrieve_balance(ls_custid) !=1 THEN
  MessageBox("出错", "检索失败!")
ENDIF
```

示例 2：实例化服务器上的一个组件并把这个组件的引用赋给一个变量，这个变量的类型是这个组件类的祖先。CreateInstance 函数的第 2 个参数定义了组件的类名和 Jaguar 组件包名。

```
//全局变量:
//connection myconnect
uo_person lnv_object
string ls_custid
10ng ll_rc
ls_custid=Trim(sle_custid.text)
ll_rc=myconnect.CreateInstance(lnv_object,,PB_pkg_1/uo_customer,)
IF ll_rc<>0 THEN
  MessageBox("创建实例失败",ll_rc)
ENDIF
  IF iuo_customer.retrieve_balance(ls_custid)  !=1 then
MessageBox("出错","检索失败!")
ENDIF
```

15.9.2 清除代理对象实例

当使用完 Jaguar 组件后，可以使用 DESTROY 语句明显地清除 Jaguar 代理对象，也可以利用 PowerBuilder 的无用存储单元收集功能来自动清除。不论采用哪一种方法，对客户端代理对象的清除工作都不会影响到服务器组件的生命周期。服务器组件的清除由 Jaguar 来处理。

如果组件的 Automatic Demarcation/Deactivation 设成无效，并且在组件仍然与客户约束在一起时关闭了客户应用，那么这个组件将不会结束工作。为了保证组件实例被释放，可以采用以下方法之一来实现：

(1) 在客户应用程序的 Close 事件中调用一个服务器组件清除这个组件的方法(通过调用 SetComplete 或 SetAbort)。

(2) 将组件 Timeout 属性设置成一个非 0 值。如果组件的 Timeout 属性被设成 0，表明这个组件永远都不会超时。

15.9.3　请求服务器返回消息

客户应用不能向 Jaguar 传递一个 PowerBuilder 对象引用，因此不能像在分布式 PowerBuilder 应用中那样使用 PowerBuilder 对象引用从服务器向 PowerBuilder 客户推送消息。不过可以在客户中使用一个共享对象与 Jaguar 通信来模拟服务器消息推送功能。因为客户上的共享对象把数据从服务器上推送回来，所以可以把这个技术理解为客户推送。

为了模拟服务器推送，客户首先需使用 SharedObjectRegister 和 SharedObjectGet 函数创建一个共享对象。然后客户上的主线索异步调用共享对象的一个方法，向它传递一个回调对象，当 Jaguar 服务器上的处理结束时就需要通知这个对象。这个共享对象的方法同步调用 Jaguar 组件方法进行一些处理工作。因为共享对象是运行在一个独立的线索中，所以，在这期间客户上的主线索还可以处理其他工作。

示例：这个例子说明如何使用共享对象同步调用 Jaguar 组件方法，并且向一个客户应用窗口返回数据。此例分成客户应用窗口、共享对象定义 Jaguar 组件和回调对象定义 4 个部分来说明。

1. 客户应用窗口

客户应用有一个窗口 w_employee，在这个窗口的数据窗口控件中显示职工数据。当用户单击窗口中的 Retrieve 按钮时，客户创建与 Jaguar 通信的一个共享对象。另外再创建一个用来处理共享对象返回消息的用户对象实例。

w_employee 窗口有以下实例变量：

```
uo_sharedobject iuo_sharedobject
uo_callback iuo_callback
```

2. 共享对象定义

Retrieve 按钮创建与 Jaguar 通信的一个共享对象，另外再创建一个用来处理共享对象返回消息的用户对象实例。为了允许回调对象通知窗口处理结束，脚本中调用了这个回调对象的 PassObject 函数，向它传递一个窗口引用。最后同步调用共享对象的 RetrieveData 函数，向它传递这个回调对象的引用。这个 Retrieve 按钮有如下脚本：

```
Long ll_rv
SharedObjectRegister("uo_sharedobject","myshare")
SharedObjectGet("myshare",iuo_sharedobject)
iuo_callback=CREATE uo_callback
iuo_callback.passobject(parent)
iuo_sharedobject.post retrievedata(iuo_callback)
```

SetDW 函数把从 Jaguar 服务器返回的数据窗口 Blob 内容装入客户窗口中的数据窗口控件中。SetDW 函数接收类型为 Blob 的 ablb_data 参数，并返回一个 Long 类型值。该函数脚本如下：

```
Long ll_rv
ll_rv=dw_employee.SetFullState(ablb_data)
IF ll_rv= -1 THEN
   MessageBox("出错","SetFullState 调用失败!")
ENDIF
RETURN ll_rv
```

3. Jaguar 组件

Jaguar 组件是 uo_employee 的 PowerBuilder 用户对象。这个 uo_employee 对象有一个函数 RetrieveData，它使用了一个数据存储从数据库检索职工数据。uo_employee 对象有以下实例变量：

```
protected TransactionServer txnsrv
protected DataStore ids_datastore
```

RetrieveData 函数的参数是 ablb_data，类型为 Blob，函数返回值为 Long 类型。该函数有如下脚本：

```
Long ll_rv
ll_rv=ids_datastore.Retrieve()
ll_rv=ids_datastore.GetFullState(ablb_data)
txnsrv.SetComplete()
return ll_rv
```

4. 回调对象定义

当 Jaguar 组件已经完成了处理工作后，共享对象通知 uo_callback 用户对象，然后这个用户对象再通知 w_employee 窗口。uo_callback 对象有两个函数：Notify 和 PassObject。

Notify 函数调用 w_employee 窗口的 SetDW 函数，传递给它从服务器组件返回的数据窗口 blob。Notify 函数带有类型为 blob 的参数 ablb_dam，并返回一个 Long 类型值。该函数的脚本如下：

```
Long ll_rv
ll_rv=iw_employee.setdw(ablb_data)
IF ll_rv=-1 THEN
   MessageBox("出错","调用 SetDW 失败!")
END IF
RETURN ll_rv
```

PassObject 函数在 iw_employee 实例变量中缓存一个对 w_employee 窗口的引用。该函数带有类型为 w_employee 的参数 aw_employee，返回一个 Long 类型值。该函数的脚本如下：

```
iw_employee=aw_employee
return 1
```

15.10 处理通信错误

15.10.1 概述

每当客户与服务器之间发生通信故障时，客户应用都可以截获所产生的错误并做出相应处理。在一个错误事件中，PowerBuilder 把有关的错误信息记录在一个内置的错误对象中。

客户应用可以访问这个错误对象，找出错误发生原因并做出相应处理。客户应用可以用几种不同的方法处理通信错误。例如，如果一个客户与一个服务器连接并且调用一个不存在的对象的方法，那么，这个客户可以断开与服务器的连接，转而连接到另一个服务器，并重复调用对象方法的操作。这个客户也可以向用户显示出错误信息，让用户决定下一步操作。

1. 与不同服务器连接

当发生错误时，如果客户与一个服务器连接并且重试出错的操作，那么在调用远程对象的方法之前，客户必须初始化新服务器上的远程对象。

2. 编写错误处理脚本

要在客户应用中处理所发生的错误，可以为以下两个事件编写脚本：

- 连接对象的 Error 事件
- 应用对象的 SystemError 事件

PowerBuilder 先触发连接对象的 Error 事件，然后再触发应用对象的 SystemError 事件。

3. 系统异常处理器

PowerBuilder 有一个试图捕获致命系统错误的系统异常处理器。它防止服务器因为个别客户连接故障而终止。每当在客户连接中发生致命系统错误时，PowerBuilder 都要努力终止客户连接，使它不会导致服务器运行异常中断或影响其他客户连接。当发现错误后，系统异常处理器就会触发应用对象的 SystemError 事件。

4. CORBA 异常

CORBA 为组件提供了错误通知或警告的一个标准方法。CORBA 支持两种异常类型：

- 系统异常
- 用户异常

系统异常属于由服务器产生的一组标准错误集。用户异常是由组件的 IDL 定义的错误或警告。用户异常是一个新的数据类型，用以说明一组数据元素，这些数据元素将在错误产生时返回给客户。

如果 Jaguar 组件的一个 PowerBuilder 客户接收到一个系统异常，那么这个异常将被默认显示成一个运行错误。从客户的角度上看，这个错误处理机制对于 Jaguar 客户和对于 PowerBuilder 分布式客户是一样的。

PowerBuilder 组件的用户异常是用 IDL 定义的，所以，非 PowerBuilder 客户可以使用由 PowerBuilder 运行错误传递的信息。这个运行错误异常被称为 PBUserException。如果 Jaguar 组件的 PowerBuilder 客户接收到一个 PBUserException，那么 PowerBuilder 虚拟机将会检索数据，并把这个错误重新生成一个普通的运行错误。

15.10.2　连接对象 Error 事件脚本

在连接对象 Error 事件脚本中处理错误，首先需要创建一个定制连接对象定义的用户对象。当创建了这个定制连接对象后，就可以在脚本中任何使用连接对象的地方引用这个定制连接对象。连接对象向导可用来帮助创建定制连接对象。

定制连接对象的 Error 事件有几个参数，它们提供了错误产生的有关情况。例如，这些参数不仅提供了错误号和错误说明文字，还提供了发生错误的对象名以及错误产生之处的全部脚本。

除了提供错误信息外，Error 事件还有一个 Action 参数可用来设定错误发生后将采取什

么动作。这个参数的有效值为 4 个枚举值，ExecptionFail!、ExceptionRetry!、ExcepOonIgnore! 和 Exception- SubstituteReturnValue!。

15.11 分发客户应用

在分布式环境中，分发一个客户应用的过程与分发其他 PowerBuilder 应用程序的过程几乎一样。打包客户应用程序有两个基本方法：

- 包含应用中所有对象的一个独立的可执行文件
- 一个可执行文件和一个或多个动态库文件

可以提供一些应用程序中需要应用到的附加资源，如位图和图标。资源文件可以与可执行文件或动态库文件打包到一起，也可以作为独立的文件进行分发。

15.12 习 题

1. 填空题

(1) PowerBuilder 提供了一组工具可用于开发定制类(不可视)用户对象，并把这些对象开发成_____。

(2) Jaguar 组件分发的用户对象接口中可以使用以下数据类型，包括_____、_____、_____。

(3) 如果想要利用 Jaguar 支持连接缓冲池及事务管理的优势，必须使用_____、_____、_____、_____与数据库进行连接。

(4) PowerBuilder 提供了_____和_____两个系统对象。

(5) 连接对象不是内置全局变量，因此需要声明一个类型为_____的变量。

2. 问答题

(1) 如何与 Jaguar 服务器连接？

(2) 如何处理通信错误？

(3) 如何分发客户应用？

[General Information]
书名=POWERBUILDER 10
作者=崔巍　　陈雍先编著
页数=384
出版社=0实用教程
出版日期=清华大学出版社
SS号=11728410
DX号=
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=&d=20401835 0E153ABB9B73FEBC82B567 15

封面
书名
版权
前言
目录
第1章　PowerBuilder 10.0概述
1.1　PowerBuilder 10.0简介
1.2　安装和启动PowerBuilder 10.0
1.2.1　安装PowerBuilder 10.0
1.2.2　启动PowerBuilder 10.0
1.3　PowerBuilder 10.0开发环境
1.3.1　菜单系统
1.3.2　工作区
1.3.3　工具栏
1.3.4　画板
1.3.5　帮助系统
1.4　窗口和菜单简介
1.4.1　窗口简介
1.4.2　菜单简介
1.5　实例：使用PowerBuilder创建第一个应用程序
1.6　习题
第2章　SQL基础
2.1　SQL Server数据库管理概述
2.1.1　SQL语句简介
2.1.2　企业管理器
2.1.3　创建数据库
2.1.4　创建表
2.1.5　创建视图
2.2　SQL的数据定义功能
2.2.1　Select子句
2.2.2　Where子句
2.2.3　From子句
2.2.4　更名运算
2.2.5　元组变量
2.2.6　排列元组的显示次序（order by）
2.2.7　集合操作
2.2.8　统计查询
2.2.9　嵌套子查询
2.2.10　SQL查询示例
2.3　数据库修改
2.3.1　删除
2.3.2　插入
2.3.3　更新
2.4　实例：利用SQL操作进行身份验证
2.5　习题
第3章　PowerBuilder编程基础
3.1　面向对象的编程特点
3.2　PowerScript语言基础
3.2.1　标识符，注释，代词
3.2.2　数据类型
3.2.3　变量与常量
3.2.4　数组
3.2.5　运算符
3.2.6　表达式
3.3　PowerScript编程基础

3．3．1　控制结构
3．3．2　函数和结构
3．3．3　编程规范
3．4　实例：文件复制
3．5　习题
第4章　PowerBuilder应用基础
4．1　应用对象基础
4．1．1　应用对象概述
4．1．2　应用对象和应用对象画板
4．1．3　应用对象的事件
4．2　使用代码编辑器
4．2．1　代码编辑器
4．2．2　编辑代码
4．2．3　粘贴函数
4．2．4　粘贴语句
4．2．5　代码编辑器的键盘操作
4．3　事务对象基础
4．3．1　事务对象概述
4．3．2　创建自定义事务对象
4．4　实例：创建应用程序对象
4．5　习题
第5章　窗体技术
5．1　窗体概述
5．2　设计窗体
5．2．1　窗体画板概述
5．2．2　创建新窗体
5．2．3　创建继承窗体
5．2．4　设计窗体属性
5．2．5　保存、预览、打印窗体
5．3　窗体事件和函数
5．3．1　窗体事件
5．3．2　窗体函数
5．4　窗体变量和窗体数组
5．4．1　窗体变量
5．4．2　窗体数组
5．5　窗体设计实例
5．5．1　创建图标栏应用
5．5．2　创建美观的状态条
5．5．3　在Response窗口中使用Menu
5．5．4　移动窗口中的控件
5．5．5　捕获控制菜单中的Close事件
5．5．6　将可视控件放置于其他的控件上
5．5．7　实现动态显示控件
5．5．8　动态创建可视控件
5．5．9　使用多窗口实例
5．5．10　在实例窗口中引用控件
5．5．11　向窗口传递多个参数
5．5．12　设计实时帮助条
5．5．13　实现“跑马灯”
5．5　习题
第6章　常用控件技术
6．1　按钮控件
6．1．1　命令按钮
6．1．2　图形按钮
6．1．3　设计带滚动条的图形控件

6．2　文本显示与编辑控件
6．2．1　静态文本框
6．2．2　单行编辑框
6．2．3　多行编辑框
6．2．4　掩码编辑框
6．2．5　超文本编辑框
6．2．6　设计记事本程序
6．3　选项控件
6．3．1　单选按钮
6．3．2　复选框
6．3．3　组合框
6．4　列表选择控件
6．4．1　列表框
6．4．2　图片列表框
6．4．3　下拉列表框
6．4．4　下拉图片列表框
6．5　图像显示控件
6．6　树形视图控件
6．6．1　属性
6．6．2　添加与删除列表项
6．6．3　事件
6．6．4　使用树形视图控件显示数据
6．7　使用标签控件
6．7．1　相关概念
6．7．2　建立标签控件
6．7．3　属性
6．7．4　事件
6．8　ＯＬＥ控件
6．8．1　放入ＯＬＥ控件
6．8．2　设置ＯＬＥ控件
6．8．3　激活ＯＬＥ对象的方法
6．8．4　使用ＯＬＥ访问Ｅｘｃｅｌ数据
6．9　习题
第7章　菜单技术
7．1　菜单概述
7．2　菜单操作
7．2．1　创建新菜单
7．2．2　修改菜单
7．2．3　设置菜单属性
7．3　菜单事件
7．3．1　菜单事件触发时机
7．3．2　编写菜单事件处理程序
7．4　菜单设计实例
7．4．1　制作弹出式菜单
7．4．2　为主窗口配置菜单
7．4．3　设计图形菜单
7．5　习题
第8章　ＰｏｗｅｒＢｕｉｌｄｅｒ数据库应用基础
8．1　数据库的基本概念
8．2　ＰｏｗｅｒＢｕｌｉｄｅｒ的数据库接口
8．2．1　标准数据库接口
8．2．2　专用数据库接口
8．3　数据库画板
8．3．1　数据库画板窗口
8．3．2　数据库画板功能

8．3．3　修改数据库选项
8．4　创建和删除数据库
8．4．1　创建数据库
8．4．2　删除数据库
8．5　连接数据库
8．5．1　数据库连接简介
8．5．2　使用事务对象
8．5．3　创建学生数据库
8．6　操作数据库表
8．6．1　表的创建与删除
8．6．2　定义表的属性、列的属性和扩展属性
8．6．3　表的索引
8．6．4　表的主键、外键
8．6．5　操作数据
8．7　实例：创建、配置并连接后台数据库
8．7．1　选择数据库管理系统
8．7．2　数据库设计
8．7．3　配置数据库连接
8．8　习题
第9章　数据窗口技术
9．1　数据窗口概述
9．1．1　一个简单的数据窗口设计示例
9．1．2　数据窗口分类
9．1．3　数据窗口画板
9．1．4　数据窗口数据源
9．2　数据窗口对象操作
9．2．1　划分区域
9．2．2　添加和删除列
9．2．3　计算字段
9．2．4　过滤数据
9．2．5　数据排序
9．2．6　指定检索数据方式
9．3　数据窗口控件的属性、事件和函数
9．3．1　数据窗口控件的属性
9．3．2　数据窗口控件的事件
9．3．3　数据窗口控件的常用函数
9．4　数据窗口设计实例
9．4．1　增加数据窗口数据
9．4．2　按学号排序
9．4．3　数据窗口排序
9．4．4　利用ＤａｔａＳｔｏｒｅ共享数据
9．4．5　关联与条件查询
9．4．6　数据查询
9．4．7　模糊查询
9．5　习题
第１０章　数据管道技术
１０．１　数据管道概述
１０．２　创建对象
１０．２．１　创建管道对象
１０．２．２　创建支撑用户对象
１０．２．３　创建窗口
１０．３　数据管道操作
１０．３．１　初始化操作
１０．３．２　启动和终止数据管道
１０．３．３　处理行错误

10.3.4 结束操作
10.3.5 数据库之间的数据传递
10.4 数据管道设计实例
10.4.1 创建数据管道
10.4.2 数据管道程序设计
10.4.3 系统集成
10.5 习题
第11章 用户对象和用户事件技术
11.1 用户对象概述
11.1.1 面向对象编程技术
11.1.2 用户对象的分类
11.2 创建用户对象
11.2.1 创建标准可视用户对象
11.2.2 创建定制可视用户对象
11.2.3 创建外部可视用户对象
11.2.4 创建定制类用户对象
11.2.5 创建标准类用户对象
11.2.6 继承创建
11.3 定制用户对象
11.3.1 插入不可视用户对象
11.3.2 插入结构
11.3.3 插入函数
11.3.4 插入事件
11.4 使用用户对象
11.4.1 使用可视用户对象
11.4.2 使用类用户对象
11.5 窗口与用户对象间的通信
11.5.1 直接引用用户对象属性
11.5.2 使用函数
11.5.3 使用用户事件
11.6 制作一个通用对话框实例
11.6.1 制作通用对话框
11.6.2 设计计算器
11.7 习题
第12章 管理库文件
12.1 库文件概述
12.1.1 库文件
12.1.2 库画板
12.2 管理PBL库
12.2.1 创建和删除库
12.2.2 创建动态库
12.2.3 优化库文件
12.2.4 设置搜索路径
12.2.5 重建与移植库
12.3 操作库与维护库及其对象
12.3.1 打开和预览对象
12.3.2 复制、移动、粘贴和删除对象
12.3.3 搜索目标程序、库和对象
12.3.4 导入和导出对象
12.3.5 生成库内容报告
12.4 根据已有的库文件创建目标
12.5 习题
第13章 程序调试与发布
13.1 调试器概述
13.2 断点操作

13.2.1 调试应用程序的切换
13.2.2 设置断点
13.3 调试方法
13.3.1 调试的开始、继续和停止
13.3.2 其他调试方法
13.4 查看信息
13.4.1 查看各种变量值
13.4.2 查看其他信息
13.4.3 运行错误解释
13.5 发布应用程序
13.5.1 概述
13.5.2 建立工程对象
13.5.3 编译生成各类对象
13.5.4 编译生成可执行应用
13.6 制作安装盘
13.6.1 找出软件发布的所有必需文件
13.6.2 确保软件的目录结构同软件一致
13.6.3 在注册表里的设置
13.6.4 确定要生成的程序包的名字和目标
13.7 习题
第14章 分布式体系
14.1 分布式体系结构概述
14.1.1 分布式计算
14.1.2 远程对象
14.1.3 应用服务器
14.2 分布式体系应用
14.2.1 建立客户端通信
14.2.2 控制用户登录应用服务器
14.2.3 创建分布式PowerBuilder服务器
14.3 分布式体系设计实例
14.3.1 创建EAServer组件
14.3.2 发布客户端应用
14.4 习题
第15章 建立Jaguar组件和客户
15.1 Jaguar组件概述
15.1.1 管理程序变量空间
15.1.2 定义组件接口
15.2 存取数据库
15.2.1 概述
15.2.2 使用连接缓存
15.2.3 提供事务支持
15.2.4 执行检索操作
15.2.5 执行更新操作
15.2.6 传递结果集
15.2.7 支持实例缓冲池
15.3 使用共享组件
15.3.1 创建一个共享组件
15.3.2 创建一个服务组件
15.3.3 调用另一个服务器组件的方法
15.3.4 访问组件属性
15.4 组件测试与调试
15.5 向Jaguar分发组件
15.6 Jaguar客户概述
15.7 与Jaguar服务器连接
15.7.1 编写代码

15.7.2 使用向导
15.8 生成Jaguar代理对象
15.9 调用组件方法
15.9.1 调用方法
15.9.2 清除代理对象实例
15.9.3 请求服务器返回消息
15.10 处理通信错误
15.10.1 概述
15.10.2 连接对象Error事件脚本
15.11 分发客户应用
15.12 习题