返回总目录

第 五 章

使用数据窗口对象

本章导读

上一章已经学习到，在 DataWindow 对象处理大部分的数据库交互工作，如检索数据、执行更新、处理删除和插入等操作。但用户脚本仍然需要初始化 DataWindow 执行的过程，还必须将 DataWindow 集成到应用程序中。本章详细介绍了以下内容：如何创建一个数据窗口控件并把它与数据窗口对象联系起来；如何创建一个事务对象并连接事务对象和数据库；如何将数据窗口和某一个事务对象联系起来；在窗口中如何动态访问数据窗口及如何动态修改数据窗口中的数据；什么是事务管理，它的作用何在；如何为数据窗口事件编写代码；最后用一个例子，说明如何动态修改与数据窗口控件相关联的数据窗口对象。

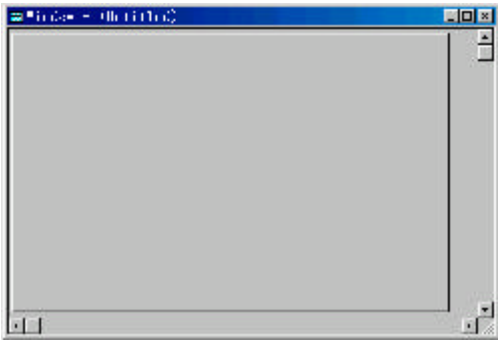
知识卷

例

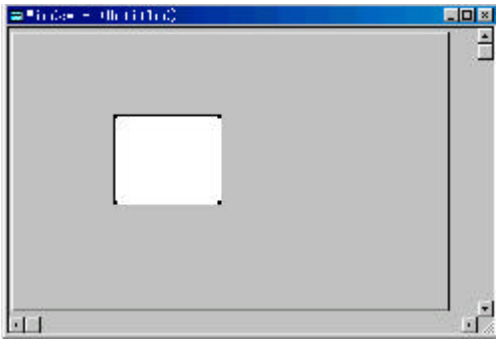
数据窗口控件
设置数据窗口控件

解

1

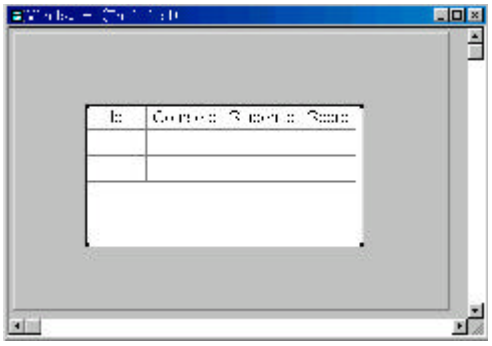


2 在窗口上添加数据窗口控件（DataWindow Control）。添加方法同添加其他控件，通过单击窗口画板上的 Control Select 按钮，在弹出的工具板上单击 DataWindow 按钮，再在窗口上某位置单击，见下图。

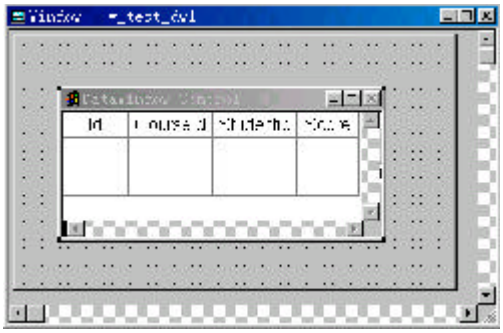


3 双击新建的数据窗口控件，打开其属性对话框如图。按 Browse 按钮为控件选择一个数据窗口对象，通过这种方式将对象与控件联系起来，控件将用来显示此对象中的数据信息。

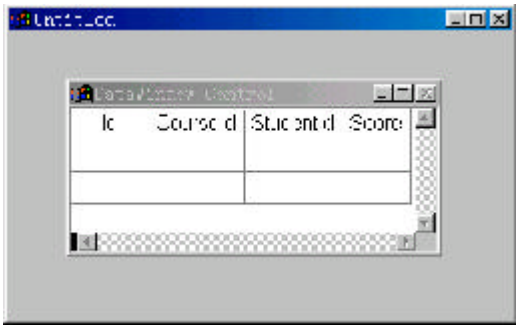
4



5 在步骤 3 对话框的 General 页中核选除了 Control Menu 和 Right To Left 之外的所有复选框；在 Title 编辑框中输入“ DataWindow Control ”作为数据窗口控件的标题，关闭对话框，结果如下图。



6 把窗口保存为 w_test_dw1 （以备后用），预览窗口如下图。



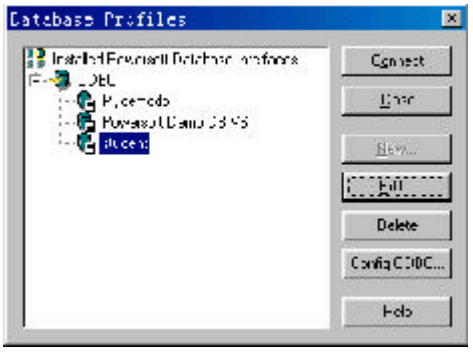
7 关闭窗口，完成本次操作。

例

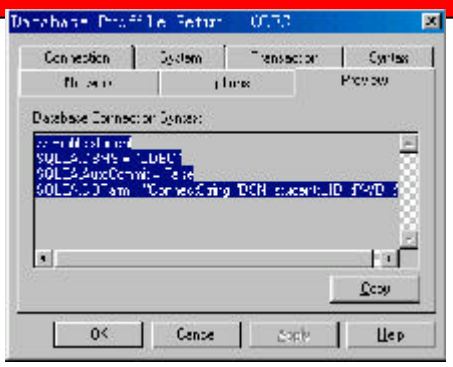
事务对象
创建和设置事务对象

解

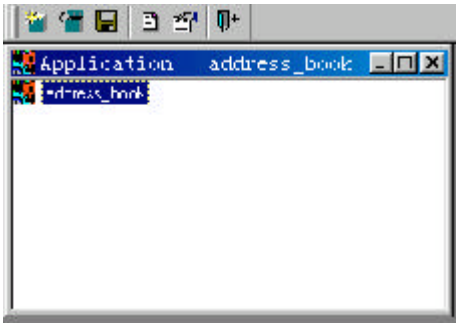
1



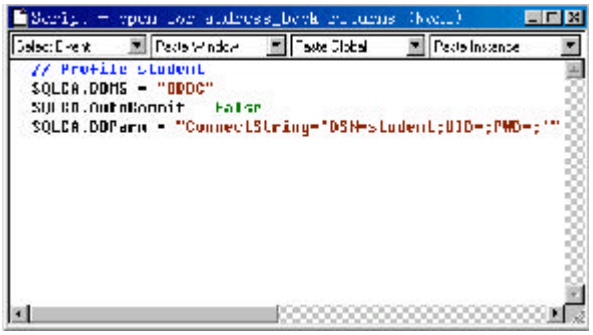
2 选择对话框的 Preview 页，如下图。拷贝 Database connection syntax 编辑框中的内容（可以按 Copy 按钮）。



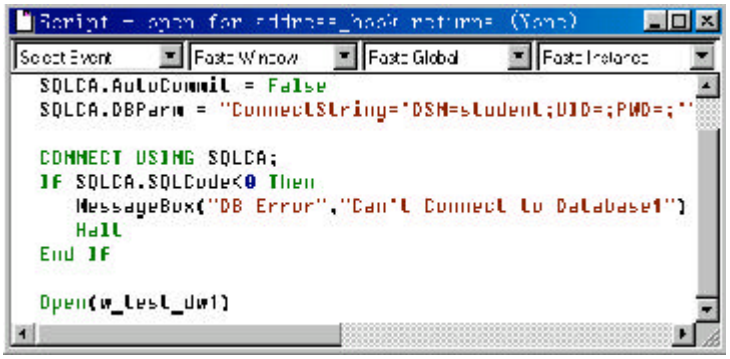
3 单击 PowerBar 上的 Application 按钮，打开当前的 Application（确信它与上一章所建的数据窗口对象及上节所新建的窗口同在一个 PBL 中），如下图。



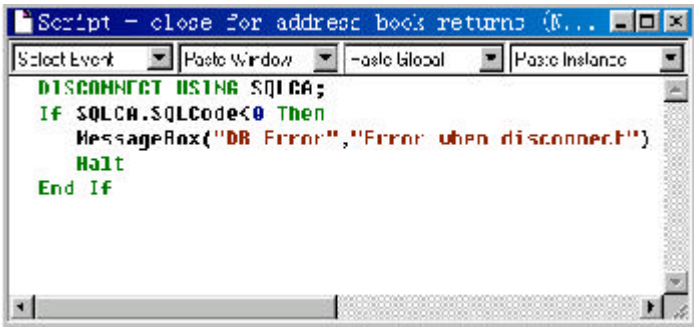
4



5 步骤 4 所添加的代码用于设置事务对象的属性。下面再给 Open 事件增加一些代码，如下图（从“CONNECT USING SQLCA”开始）。这些代码用来完成数据库与事务对象的连接和打开窗口 w_test_dw1。



6 从 Select Event 下拉列表框选择 Close，为 Close 事件增加代码如下图。这些代码用来断开数据库与事物对象的连接（后面的 If 语句用于当断开连接失败时显示错误提示）。



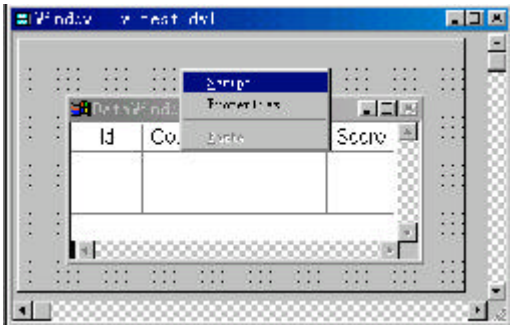
7 以上 6 步从数据库描述文件（Profile）中取属性信息从而设置事物对象的属性；然后分别在应用程序对象的 Open 和 Close 事件中建立和断开事务对象与数据库的连接。

例

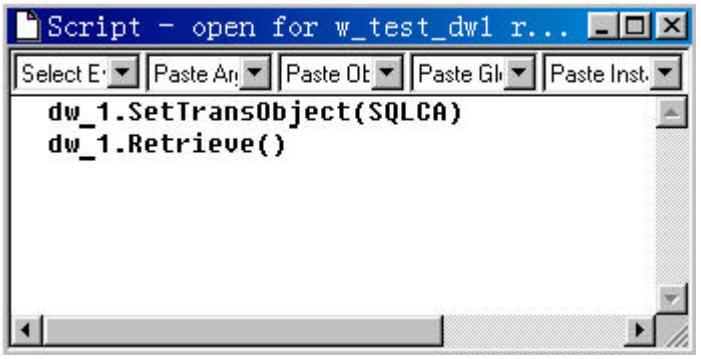
事物对象（续）
创建和设置事务对象

解

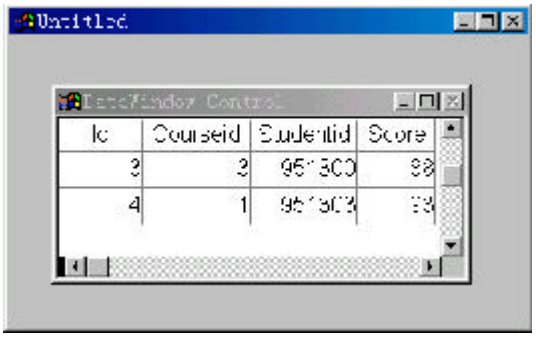
1



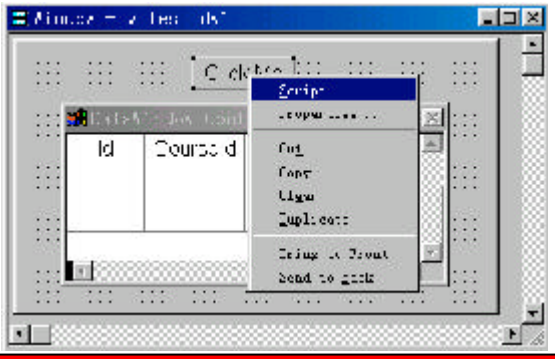
2 选择 Script 菜单项，为窗口事件增加 Script 语句处理过程。在 Select Event 下拉式列表框中选择 Open 事件，为其增加代码如下图。两行代码分别用来给数据窗口分配事务对象和让数据窗口检索数据。



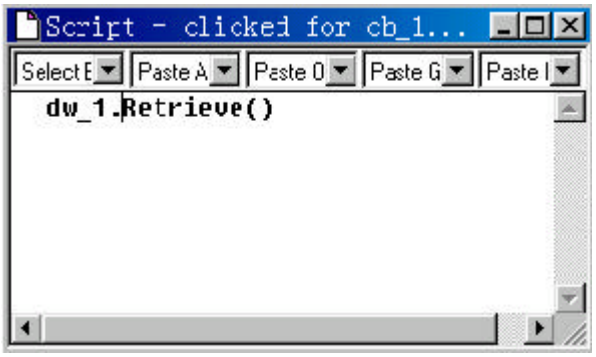
3 退出 Script 编辑窗口，按 PowerBar 上的 Run 图标按钮（一个跑动的小人），运行程序，结果如下图。发现数据窗口控件中出现了数据，和上一章在数据窗口的预览窗口中显示的结果是一样的。



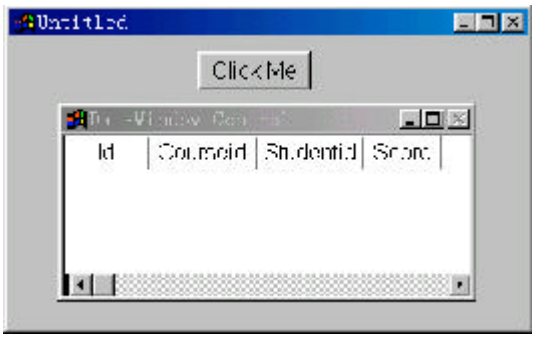
4



5 在菜单上选择 Script，进入下图窗口。在 Select Event 下拉式列表框中选择 Clicked，为按钮的单击事件增加处理过程如下图。“ Retrieve ”是数据窗口控件的成员函数，其功能是检索数据。



6 从窗口 w_test_dw1 的 Open 事件的处理过程中去掉第二行（即 Retrieve 函数那一行，因为这一行已经写在“ Click Me ”的事件处理过程中了），保存并运行窗口，结果见下图。



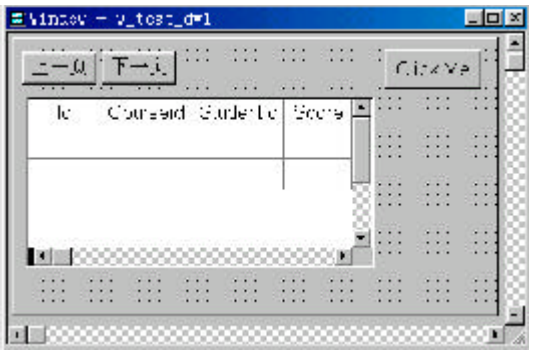
7 单击 Click Me 按钮，观察结果。

例

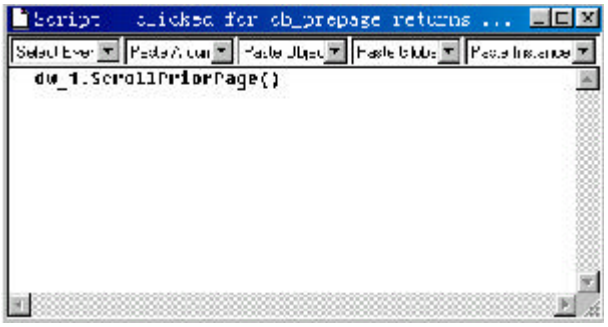
动态访问数据窗口
用命令按钮控制数据

解

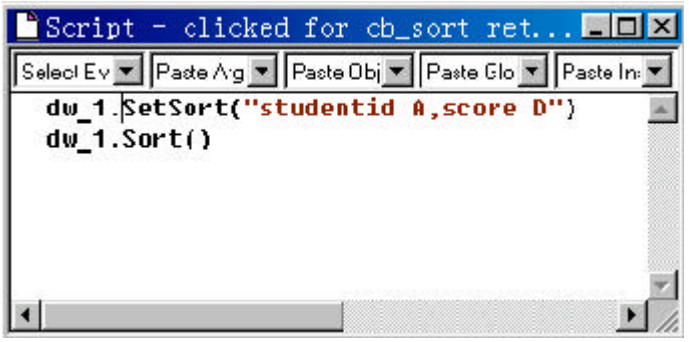
1



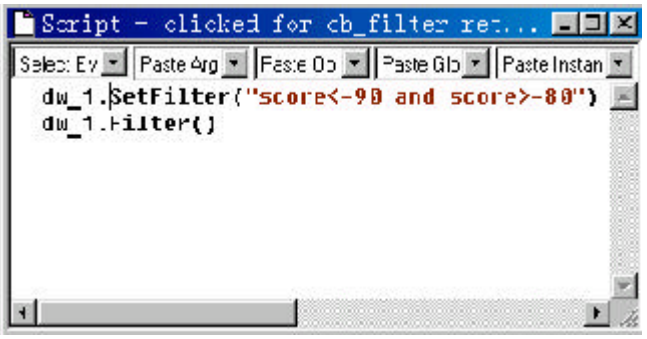
2 在“上一页”按钮的 Clicked 事件中添加代码如下图。同理在“下一页”按钮的 Clicked 事件中添加代码：dw_1.ScrollNextPage()。



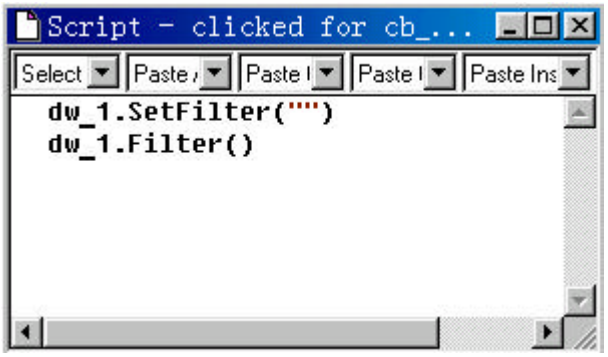
3 在窗口上增加标题为“排序”的命令按钮，在其 Clicked 事件中添加代码如下图。



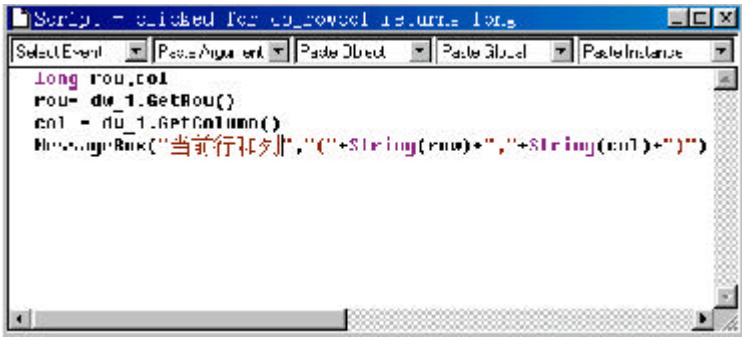
4



5 在窗口上增加标题为“取消过滤”的命令按钮，在其 Clicked 事件中添加代码如下图。



6 在窗口上增加标题为“所选行列”的命令按钮，在其 Clicked 事件中添加代码如下图。



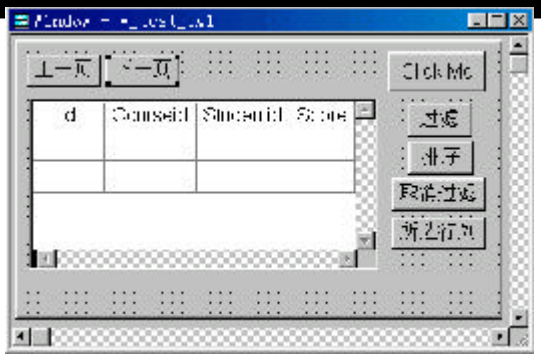
7 以上步骤所添加的按钮和代码用于在窗口中对数据窗口的数据进行浏览控制。请接下一节。

例

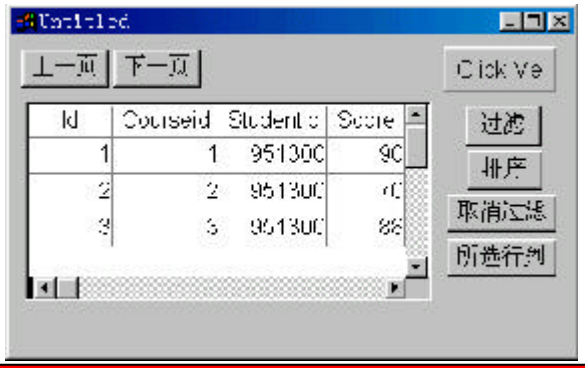
动态访问数据窗口(续)
用命令按钮控制数据

解

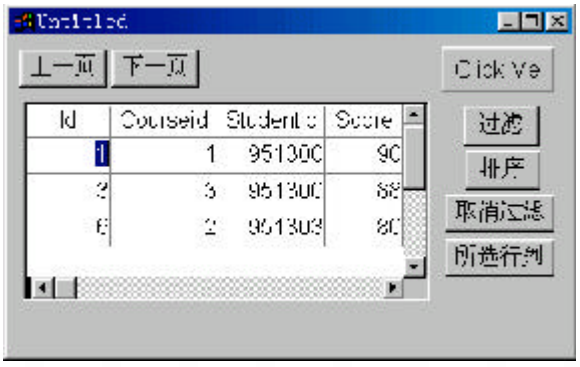
1



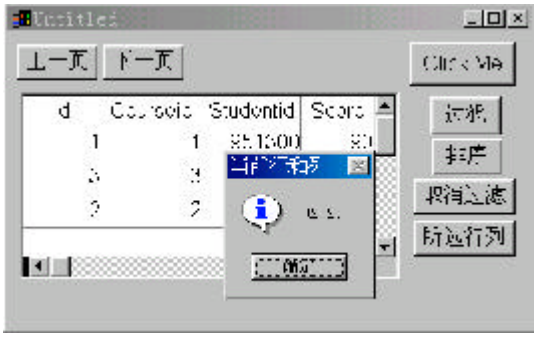
2 按 PowerBar 上的 Run 图表按钮运行程序，单击按钮 Click Me 后，数据窗口控件中显示出数据，见下图。单击“上一页”和“下一页”按钮，看数据窗口有何变化。



3 单击“过滤”按钮，表中只显示了 score 值在 80 和 90 之间的数据，如下图。读者可以按下“取消过滤”和“排序”按钮测试其功能。



4



5 下图和步骤 6 图是数据窗口控件的一些成员函数介绍，用户可以自己编写代码测试它们的功能。图中参数“ row ”表示行号，“ col ”表示列号。format 的格式见本节注释。

类别	函数	功能
行列获取	GetRow()/SetRow(row)	获取/设置当前行
	GetColumn()/SetColumn(col)	获取/设置当前列
滚动函数	ScrollToRow(row)/ScrollToRow(row)	相对当前行/直接滚动到某行
	ScrollNextPage()/ScrollPriorPage()	向后/前滚动一页
	ScrollNextRow()/ScrollPriorRow()	向后/前滚动一行
排序函数	SetSort(format)	设置排序（按哪几列，升序还是降序）
	Sort()	进行排序

6 下图 SetFilter 函数的参数 format 的格式为：过滤条件外加双引号。其中过滤条件是一个 Power Script 表达式。

类别	函数	功能
保存	SaveAs(format)	保存数据
行统计	DeletedCount()	返回 Delete 缓冲区中行数
	ModifiedCount()	返回已被修改但未更新的行数
	RowCount()	返回 Primary 缓冲区中行数
	FilteredCount()	返回 Filter 缓冲区中行数
过滤函数	SetFilter(format)	设置过滤条件
	Filter()	进行过滤

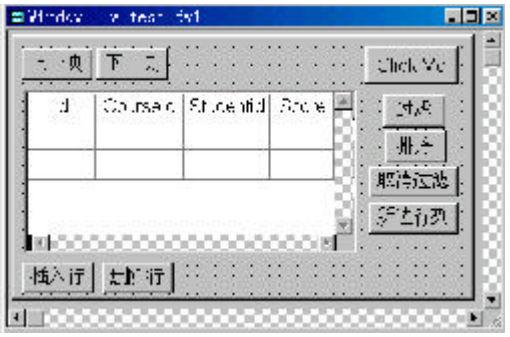
7 关闭窗口，完成操作。

例

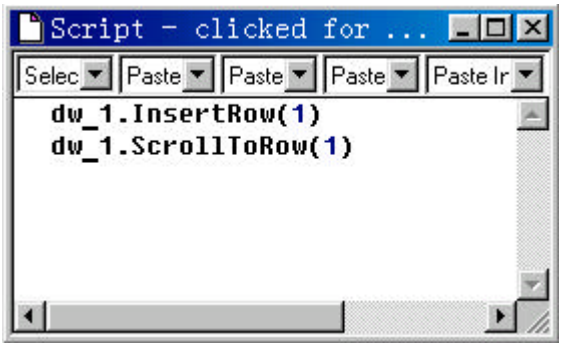
动态修改数据窗口
插入删除行

解

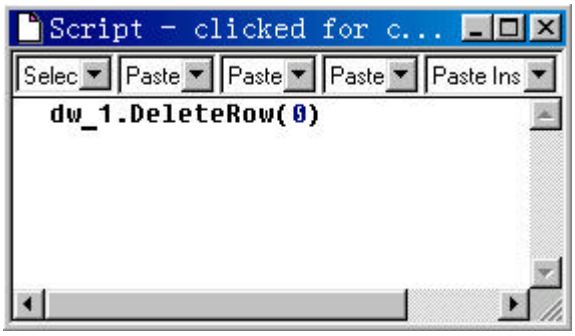
1



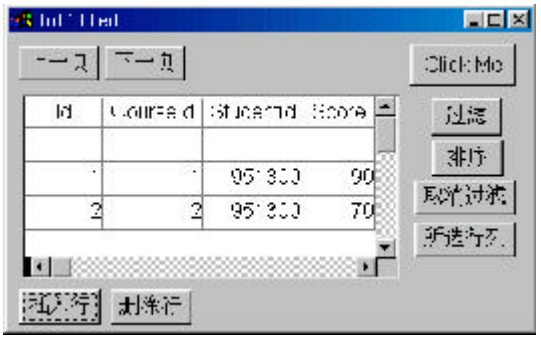
2 为“插入行”命令按钮的 Clicked 事件增加代码如下图。其中第一行代码在数据窗口的第一行前插入一个空行，第二行代码滚动到新添加的行（第一行）以便用户添加数据。



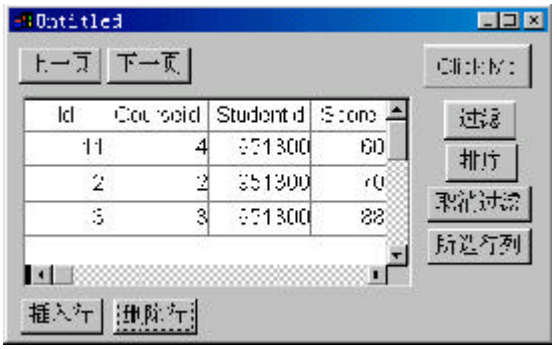
3 为按钮控件“删除行”的 Clicked 事件增加代码如下图。此行代码删除当前行。



4



5 在空行添加数据（从左到右分别填：11,4,951300,60），再选中第 2 行中任意一列数据项（按“所选行列”按钮，确信选了第 2 行），按“删除行”按钮，结果见下图。



6 下图列出了一些修改数据窗口数据的函数，它们都是数据窗口控件的成员函数。其中 row 表示行号，col 表示列号，value 表示数据，text 表示数据的字符串形式。

类别	函数	功能
插入删除	InsertRow(row)	在 row 行前插入一个空行(row=0 表示最后一行)
	DeleteRow(row)	删除 row 行(row=0 表示删除当前行)
修改数据项	SetItem(row,col,value)	修改和设置 row 行 col 列数据项的数据
	SetText(text)	用 text 字符串替代当前行列数据项的值

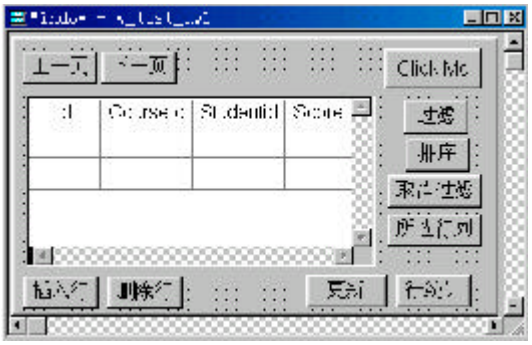
7 退出程序，再重新运行，发现对数据窗口数据所作的修改都消失了，出现在数据窗口中的还是以前的数据，这是为什么？请看下一节。

例

事务管理
数据的真正更新

解

1



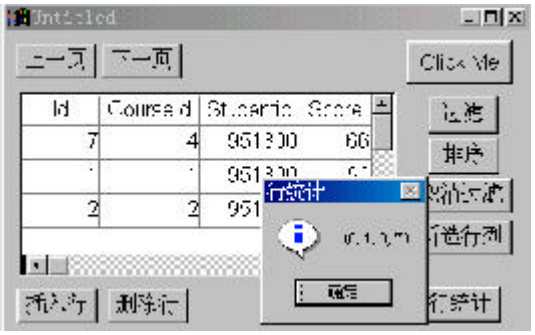
2 在按钮控件“行统计”的 Clicked 事件中增加如图代码。代码用到了本章第 5 节步骤 6 图中所示的四个行统计函数，用来统计数据窗口控件的几个缓冲区的情况。

```
long delc,modc,filc,rowc;
String str;

delc=dw_1.DeletedCount()
modc=dw_1.ModifiedCount()
filc=dw_1.FilteredCount()
rowc=dw_1.RowCount()

str="("+String(delc)+","+String(modc)+","+String(filc);
MessageBox("行统计",str+","+String(rowc)+")")
```

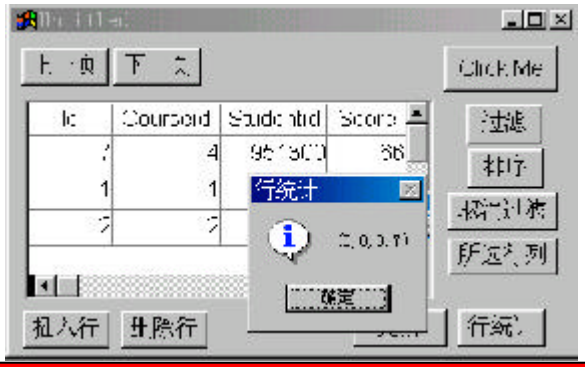
3 在按钮控件“更新”的 Clicked 事件中增加如图代码。代码用到了数据更新函数 Update()和事务管理语句 COMMIT、ROLLBACK（这两个函数完成对数据库的真正修改）。

```
int result;

result=dw_1.Update();
If result>0 then
   COMMIT USING SQLCA;
Else
   ROLLBACK USING SQLCA;
   MessageBox("消息","Can't update DB!")
End If
```

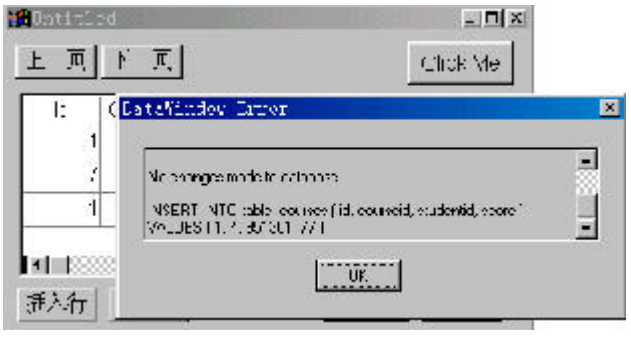
4



5 按“更新”按钮之后再单击“行统计”，弹出的对话框显示修改的行数由刚才的 1 变为 0。这是因为刚才的修改已经保存到数据库中，现在数据窗口控件的 Delete 缓冲区已经清空。



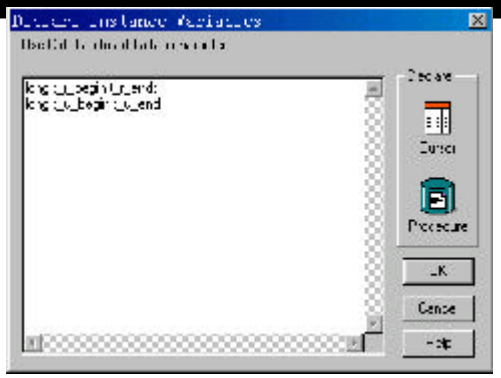
6 再按“插入行”按钮，在新插入的空行上填加数据（其它列随意，Id 列填 1）；这时按“更新”按钮，出现 DataWindow Error 对话框如下图。这是因为 Id 是关键字，不容许出现重复值。



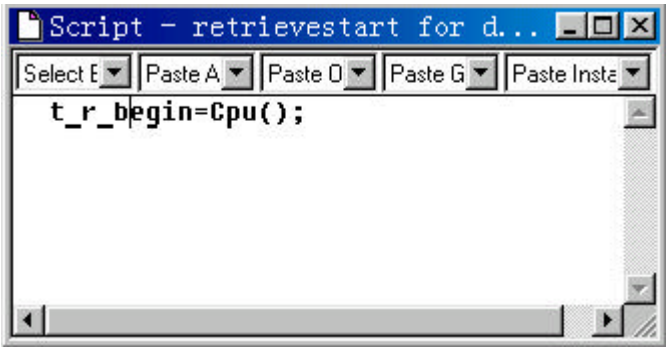
7 读者可以在按了“删除”和“过滤”按钮后，再按“行统计”，观察结果。关闭程序，操作结束。

例 数据窗口控件的事件处理 显示数据更新时间 解

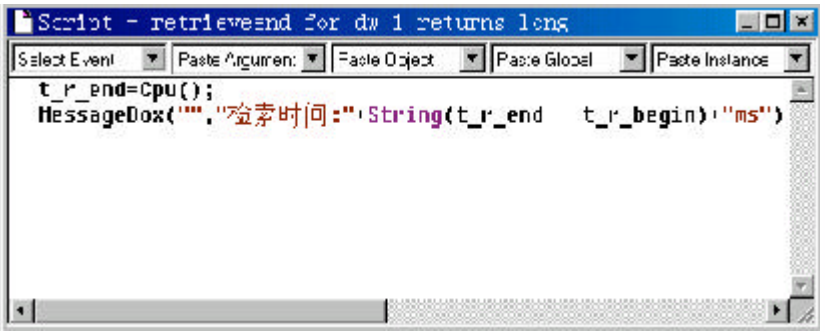
1



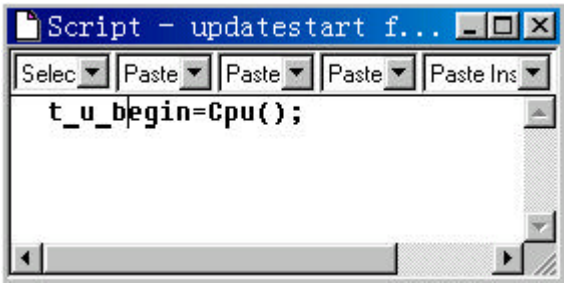
2 为数据窗口控件的 retrievestart 事件增加代码如下图。代码在开始检索数据操作时计时。



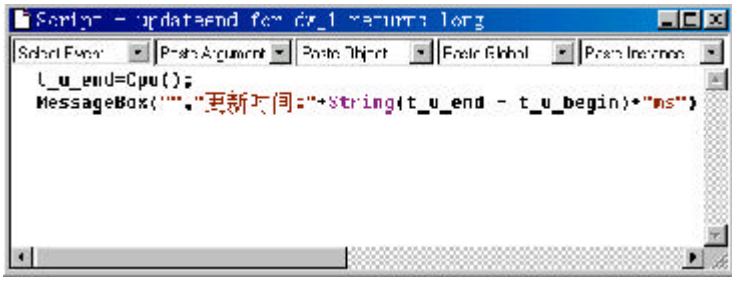
3 为数据窗口控件的 retrieveend 事件增加代码如下图。代码在完成检索数据操作时计时，计算并显示检索时间。



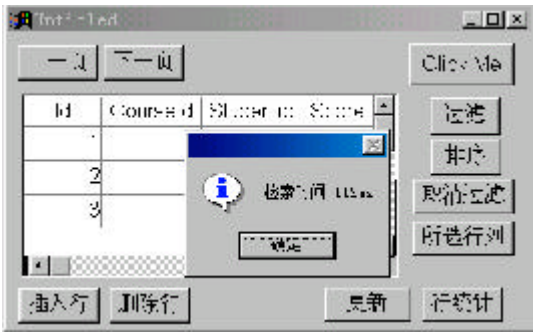
4



5 为数据窗口控件的 updateend 事件增加代码如下图。代码在数据更新操作完成时计时，计算并显示数据更新时间。



6 运行程序，当按“Click Me”按钮检索数据完毕后弹出如下对话框，显示数据检索所花的时间是 112 毫秒。



7 可以按“更新”按钮，查看数据的更新速度。

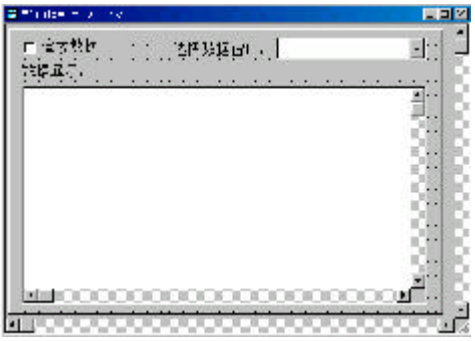
知识卷

例

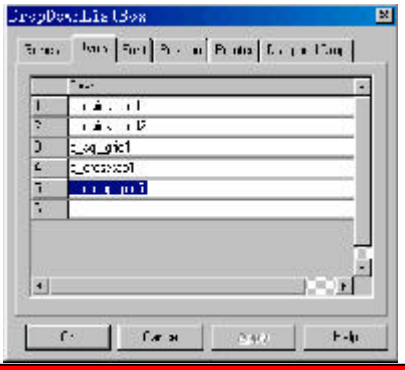
动态连接数据窗口
多个数据窗口的预览

解

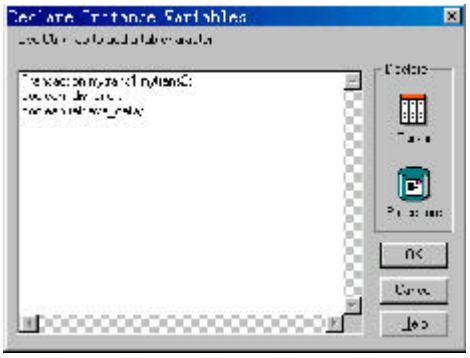
1



2 双击窗口中的下拉式列表框，在它属性对话框的 Items 页中添加数据项，其中每一项是一个数据窗口的名字，这些数据窗口都是在上一章所创建的。



3 选择 Declare|Instance Variables 菜单项，在弹出的对话框中声明几个变量如下图。第一行是两个事务对象变量，布尔变量 dis_error 和 retrieve_data 分别指示是否显示系统数据库错误信息和是否检索数据。



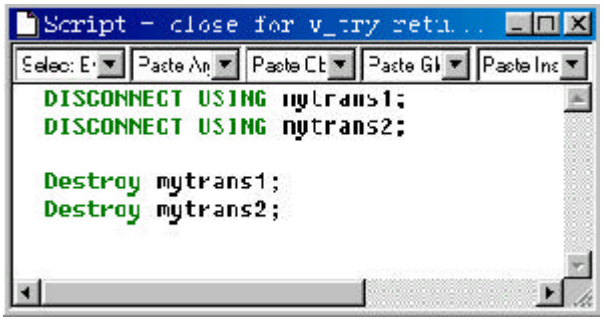
4

```
mytrans1 = Create Transaction;
mytrans2 = Create Transaction;

dis_error=true;

// Profile student
mytrans1.DBMS = "ODBC"
mytrans1.AutoCommit = False
mytrans1.DBParm = "ConnectString='DSN=student;UID=;PWD=;'"
```

5 下图接上图继续显示窗口 Open 事件的代码。前三行代码以数据源 mydemodb 的 profile 设置事务对象 mytrans2 的属性；然后连接两个事务对象和相应的数据库。

```
// Profile Mydemodb
mytrans2.DBMS = "ODBC"
mytrans2.AutoCommit = False
mytrans2.DBParm = "Connectstring='DSN=MyDemoDB'"

CONNECT USING mytrans1;
CONNECT USING mytrans2;
```

6 为窗口的 Close 事件添加代码见下图。断开两个事务对象与数据库的连接并释放事务对象所占的内存（即销毁事务对象）。



7 在开始阅读下一节之前，请读者思考，如何为以不同数据库为数据源的数据窗口对象（如 d_quick_grid 和 d_group_profit）选择合适的事务对象与之连接，以保证成功地检索到数据？

例 **动态连接数据窗口（续）多个数据窗口的预览** 解

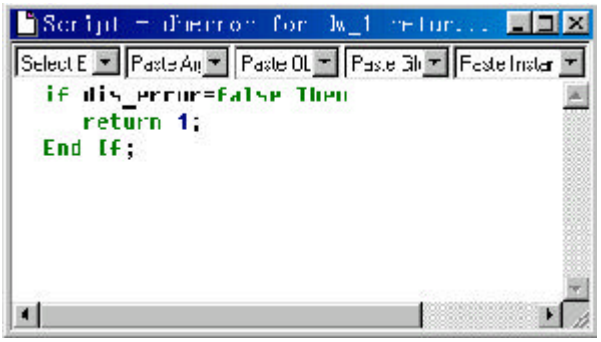
1

```
String dw_name;

dw_name=ddlb_1.text;
dw_1.dataobject=dw_name;

dw_1.SetTransObject(mytrans1)
dis_error=false;
```

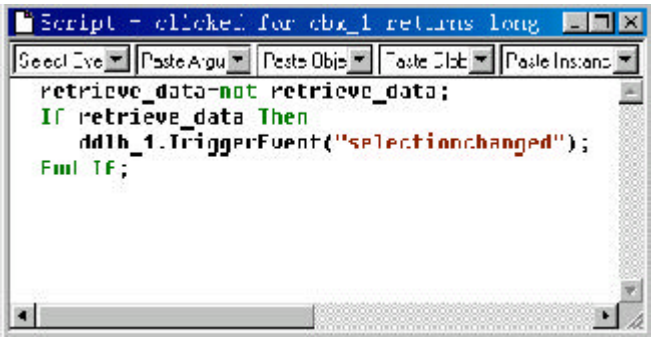
2 下图接着显示 Selectionchanged 事件的代码。检查布尔变量判断是否需要检索数据，检索时，先利用事务对象 mytrans1 进行检索，如果检索失败，更换事务对象 mytrans2 重新检索。

```
If retrieve_data Then
   If dw_1.Retrieve() = -1 Then
      dis_error=true;
      dw_1.SetTransObject(mytrans2);
      dw_1.Retrieve();
   End If;
End If;
dis_error=true;
```

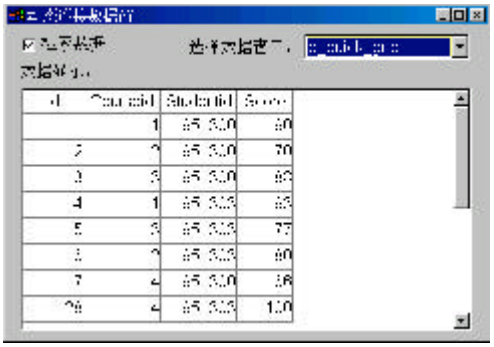
3 为数据窗口控件的 dberror 事件添加代码，如下图。代码检查布尔变量 dis_error 的值，如果 dis_error=false，返回 1 表示不显示系统数据库错误信息。



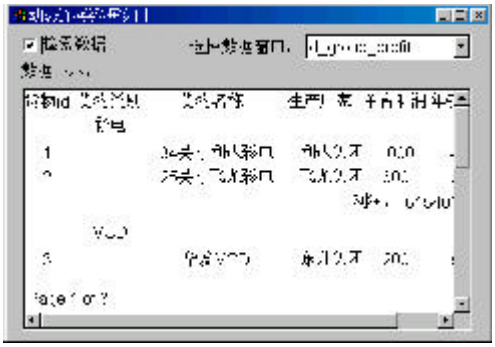
4



5 运行窗口；选中复选框；在下拉式列表框中选择 d_quick_grid（其数据源来自 Access 数据库 student 的表 table_score），结果见下图。检索数据所用的事务对象是 mytrans1。



6 在下拉式列表框中选择 d_group_profit （数据来自表 sale_goods 和 sale_goods_num，与数据源 mydemodb 相关），结果见下图。检索数据所用的事务对象是 mytrans2。



7 至此完成操作。在操作中遇到陌生的函数请查阅 Power Builder 的联机帮助。

返回总目录

第六章

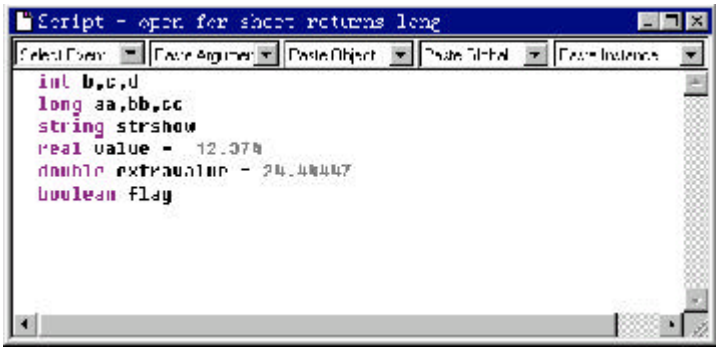
PowerScript编程

本章导读

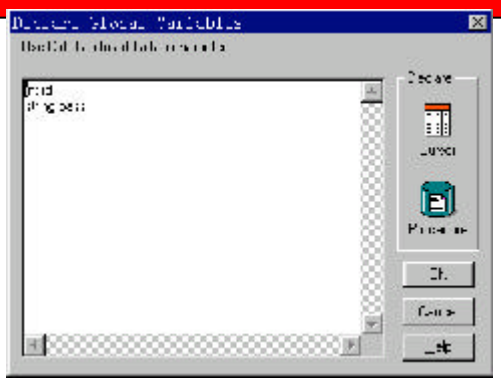
任何程序的编制归根结底都需要由代码完成，虽然可视化编程为我们提供了极大的方便。但是如果没有完善的代码，程序就不可能有完善的功能。PowerScript 是 PowerBuilder 的编程语言。我们在学习时可能已经发现，PowerScript 语言与传统的 Basic 语言有很多相似之处。PowerScript 是一种自由格式的语言，在编写程序代码时，编译器忽略它的空格、缩进、空行等，在这方面它又与 C 语言有几分相近。在本章中，我们将介绍 PowerScrpit 的数据类型，变量及作用范围，程序结构，以及一些常用函数和语句，尤其是一些和数据库有关的操作，将是本章内容的重点。

例 **变量的类型和作用域**
在不同范围内定义变量 解

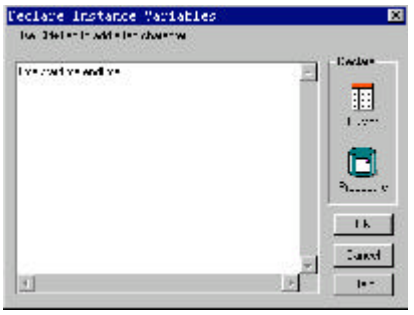
1



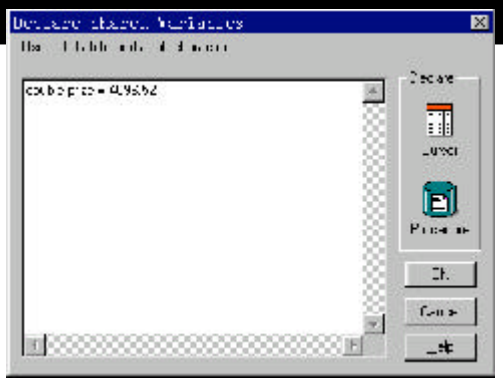
2 作用域最大的变量是全局变量，用户可以在任何地方访问它，它的声明方法是通过菜单上的 Declare|Global Variables 来进行访问。



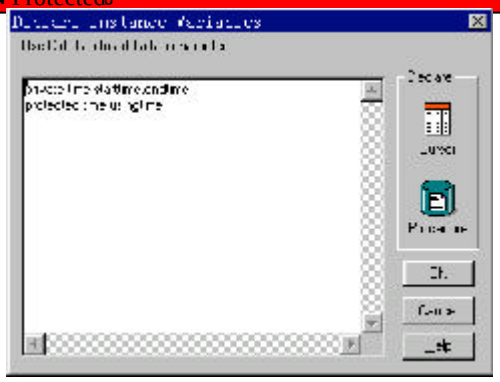
3 实例变量是我们用到的最多的一种变量，它可以在一个对象的范围内使用，例如窗口、菜单以及用户对象，用户可以通过菜单上的 Declare|Instance Variables 来定义实例变量。



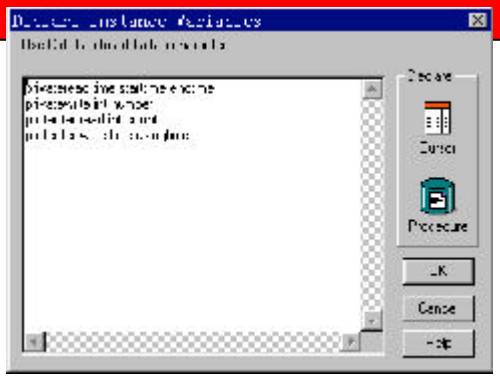
4



5 一般的实例变量可以从外部访问，为了禁止其它对象访问当前变量，或者禁止除当前对象的派生对象以外的对象访问当前变量，可分别加上限定前缀 Private 和 Protected。



6 Protected 和 Private 前缀的用法还可以进一步细化。分别是 PrivateWrite、PrivateRead、ProtectedWrite 和 ProtectedRead。



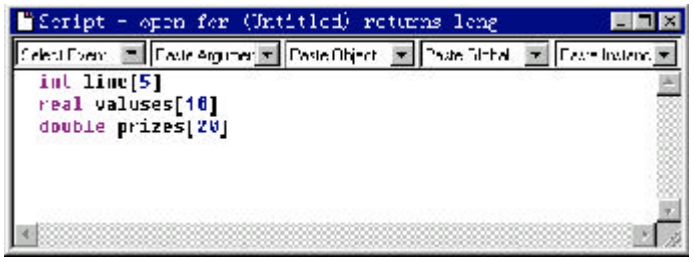
7 在本例中，我们介绍了各种变量的定义方法和它们的作用范围，读者应本着多使用局部变量，少定义全局变量的原则来进行程序设计。

例

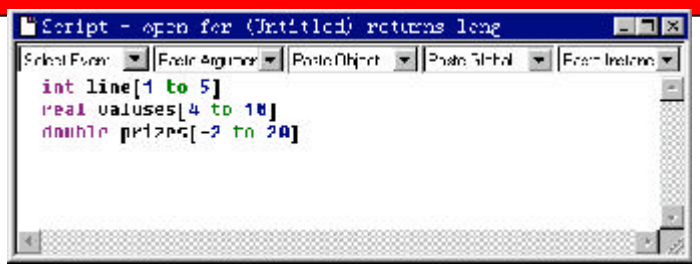
数组
数组的使用

解

1

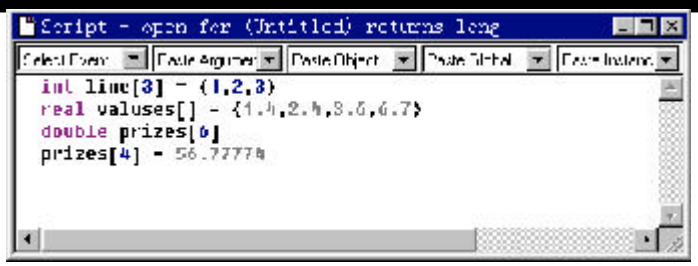


2 一般情况下，数组的下标都是从 1 开始，但是也可以使用不同的下标，如下图所示。

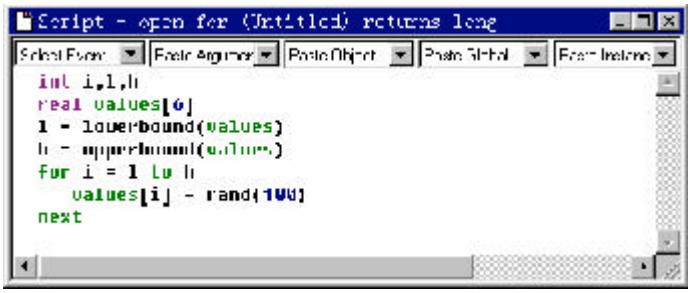


3 变长数组是在变量的个数不确定时定义的，长度可变的数组实际上是一个链表，这种定义和 C 语言很相似。

4



5 数组的作用范围可以由 UpperBound 和 LowerBound 来确定，但是由于这两个函数比较耗时，所以比较好的方法是先计算出它们的值，然后再放到循环体中。



6 多维数组常常用来存储表格，标示矩阵，平面和立方体，它的定义方法如下。

7 本例中说明了对数组的各种定义赋值和操作的方法，需要指出的是，数组之间不能比较大小，只能用“=”和“<>”来判断两个数组是否相等。

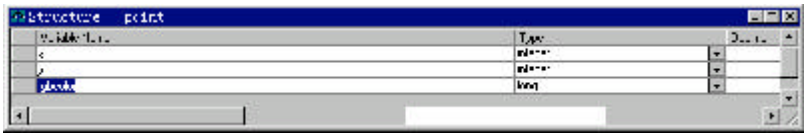
例

结构
定义和使用结构

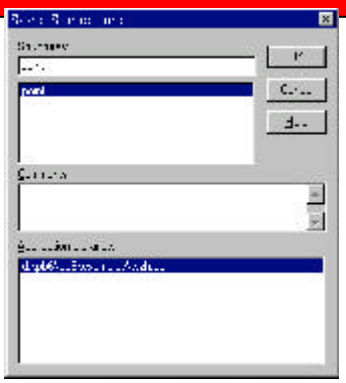
解

1

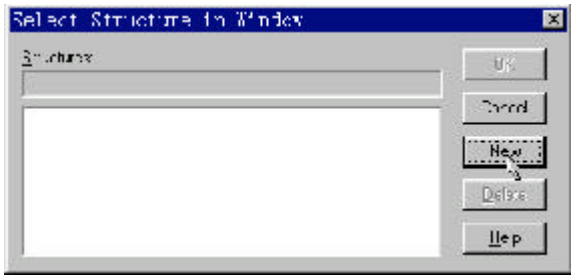
2 在结构定义对话框中，右侧为 Variable Name，用来定义结构成员的名称，左侧为 Type，用来定义结构成员的类型。



3 将当前结构命名并保存，至此完成了全局结构的定义。



4



5 对象层结构的定义对话框和全局结构的定义对话框相同，但是无需专门的保存。

6 在 Script 中使用结构，只需将结构像一个变量一样事先声明即可使用。

7 在本例中我们介绍了两种不同结构的定义方法，尽管定义方法不同，但是其使用方法却基本相同，所不同的只是对象层结构仅能在本对象中使用。

例

程序结构
几种常用的程序结构

解

1

```
IF X=Y THEN
Beep(?)
ELSEIF X=Z THEN
   Show (lb_parts); lb_parts.SelState(5,TRUE)
ELSEIF X="" THEN
   Show (lb_choose)
ELSE
   Hide(cb_empty)
   Show(cb_full)
END IF

IF Num = 1 THEN Open(w_first) ELSE Open(w_rest)
```

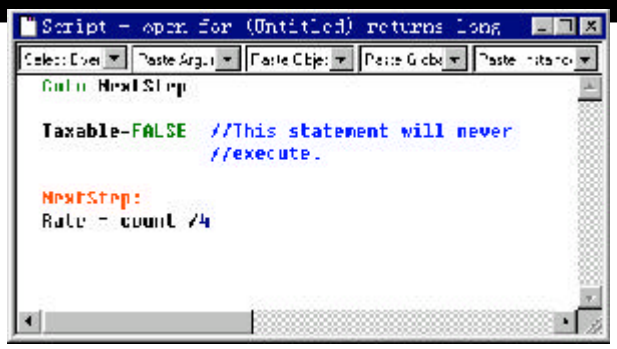
2 Choose Case 语句在条件很多的情况下尤其有用，它能使程序清晰易懂，条理清楚。

```
CHOOSE CASE Weight
CASE IS<16
Postage Weight*0.80
   Method "USPS"
CASE 16 to 48
   Postage 4.50
   Method "UPS"
CASE ELSE
   Postage 25.00
   Method "FedEx"
END CHOOSE

CHOOSE CASE Real(sle_real.Text)
CASE is < 10.99999
   sle_message.Text   "Real Case < 10.99999"
CASE 11.00 to 48.99999
   sle_message.Text   "Real Case 11 to 48.9999"
CASE is > 48.9999
   sle_message.Text   "Real Case > 48.9999"
CASE ELSE
   sle_message.Text   "Cannot evaluate!"
END CHOOSE
```

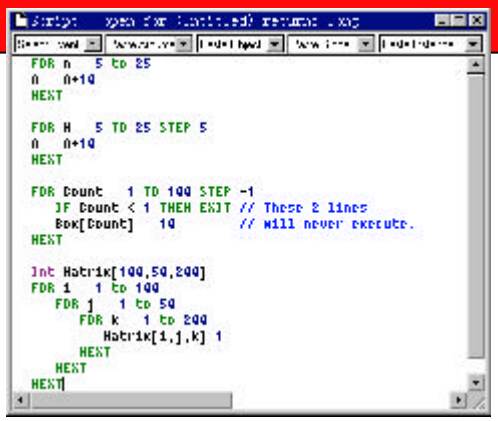
3 Do Loop 结构是一种基本的循环结构，它有四种循环判断的形式，当符合形式时，循环体继续执行，否则就跳转出来。

```
DO UNTIL A > 15
Beep(A)
   A   (A + 1) * 8
LOOP

DO WHILE A <  15
Beep(A)
   A   (A + 1) * 8
LOOP

DO
Beep(A)
   A   (A + 1) * 8
LOOP UNTIL A > 15

DO
Beep(A)
   A   (A + 1) * 8
LOOP WHILE A <  15
```

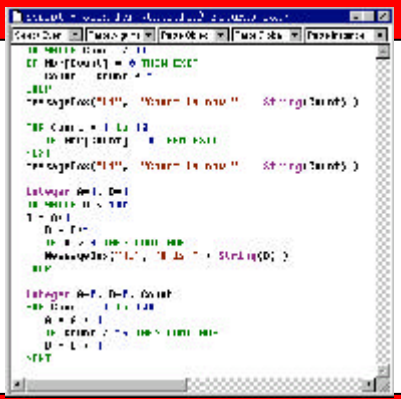
4



5 PowerScript 中的 For Next 语句与 Basic 中的基本相同，只是 Next 不加循环变量。



6 Exit 和 Continue 语句都是循环体内的跳转语句，所不同的是 Exit 直接跳出循环体，而 Continue 结束当前循环开始下一次循环。



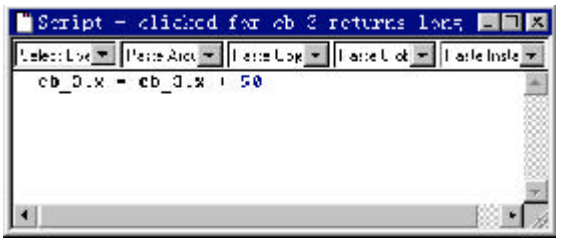
7 本例中介绍了几种 PowerScrpit 中常见的程序结构，合理的使用这些结构会使程序更加清晰有条理，执行效率也更高。

例

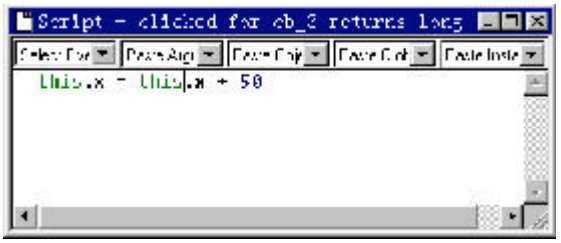
代词
几种常见代词的使用

解

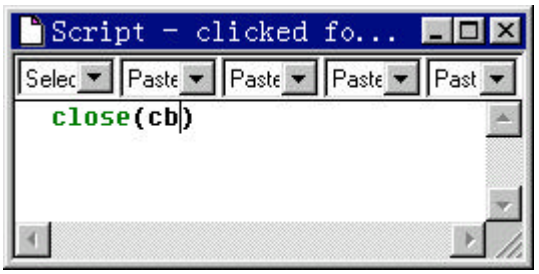
1



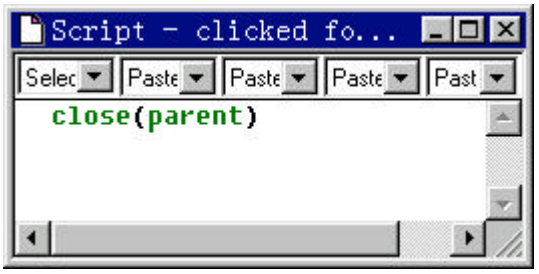
2 这样的代码隶属关系不清晰，而且无法移植，使用代词 This 后代码会变得清晰易懂，而且可以很方便的移植。



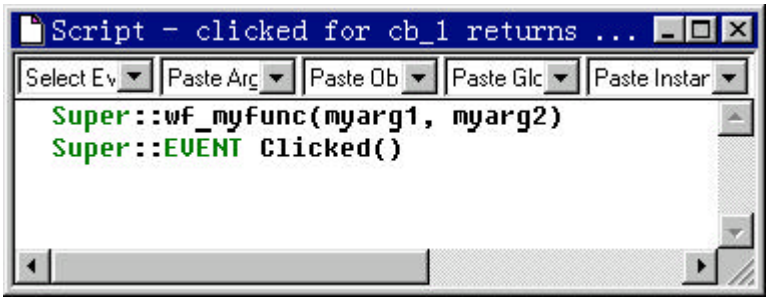
3 Parent 代词最经常的用途是指代控件所在的窗口。例如要点击按钮关闭一个名为 CB 的窗口，可在按钮的 Clicked 事件中加入如下代码。



4



5 Super 用于引用祖先的事件或函数，注意要用“ ::”将代词和函数或事件隔开，这和 C 语言比较相似。



6 ParentWindow 是菜单的一个属性，但是它可以当作代词来用，用来指代和菜单相联系的窗口。

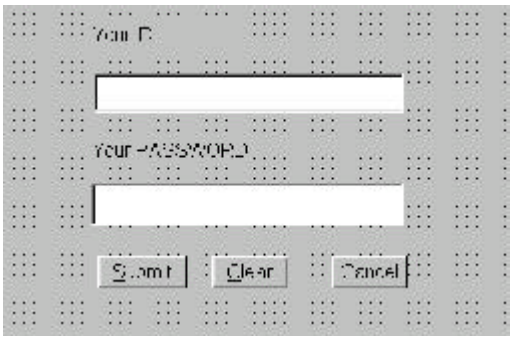
7 在本例中，介绍了几种代词的基本用法。灵活的使用这些代词，尤其是 Parent 和 This，会使你的程序更简洁清晰，便于移植。

例

常用函数和语句（一）
利用 MessageBox 函数传递消息

解

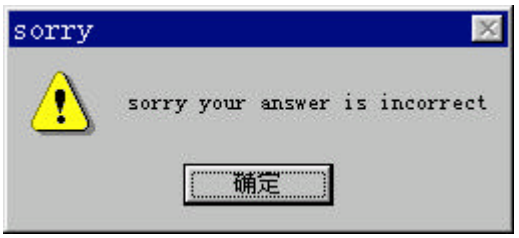
1



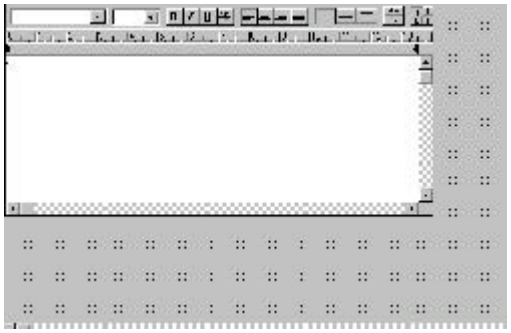
2 在按钮 Submit 的 Script 中，可以看到当输入密码正确和错误时，程序分别用 MessageBox 发送了两种不同信息。

```
string password
integer id
id = integer(sle_1.text);
  SELECT "pass"."password"
    INTO :password
    FROM "pass"
   WHERE "pass"."id"   :id   ;

if password = sle_2.text then
   messagebox("Mingle","Yes,you are correct",Information!,OK!)
else
   messagebox("sorry","sorry your answer is incorrect",Exclamation!,OK!)
end if
```

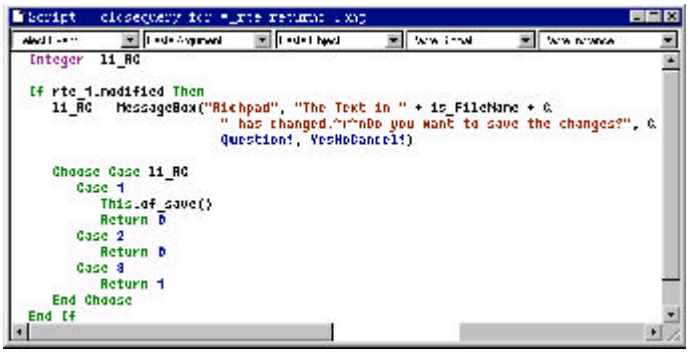
3 下图为当用户密码输入错误时出现的提示信息。



4



5 在窗口的 CloseQuery 中可以看到，当用户修改文本后未经保存而退出时，程序会给出一个有 Yes、No、Cancel 三个按钮的 MessageBox，并根据返回值进行操作。



```
Integer  li_RC

If rtc_1.modified Then
   li_RC   MessageBox("Richpad", "The Text in " + is_FileName + &
                        " has changed.~r~nDo you want to save the changes?", &
                        Question!, YesNoCancel!)

   Choose Case li_RC
      Case 1
         This.of_save()
         Return 0
      Case 2
         Return 0
      Case 3
         Return 1
   End Choose
End If
```

6 执行程序，当未经存盘而退出时，系统会给出如下的对话框，提示用户是否保存未经存盘的修改。

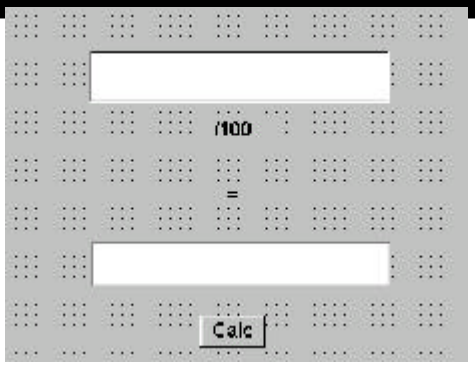
7 在本例中，列举了两个如何使用 MessageBox 的例子。一般来说，MessageBox 的用途无外乎两种，一种是单纯的提示信息，另一种是提出问题并根据返回值进行操作。

例

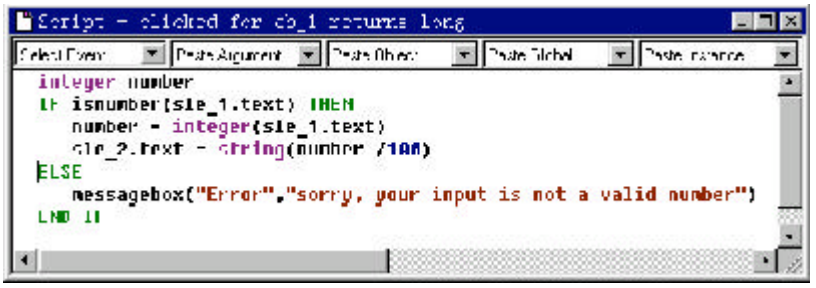
常用函数和语句（二）
Is 族函数和类型转换函数

解

1

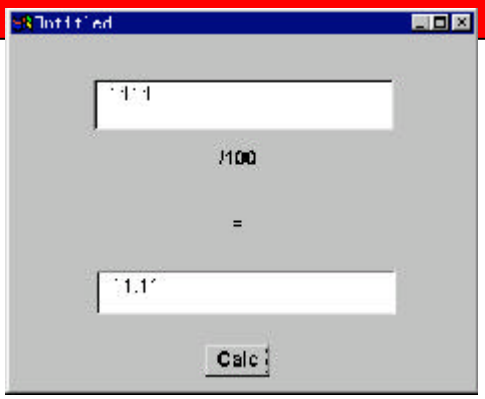


2 在命令按钮的 Script 中，加入如下代码。首先判断在第一个文本编辑器中输入的字符是否是一个数字，如果是则进行类型转换，否则给出出错信息。

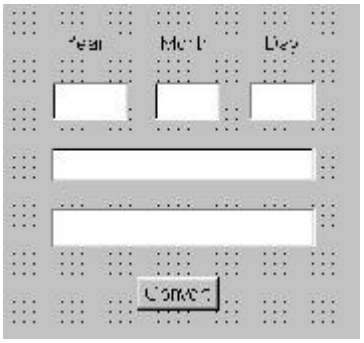


```
integer number
if isnumber(sle_1.text) then
    number = integer(sle_1.text)
    sle_2.text = string(number /100)
else
    messagebox("Error","sorry, your input is not a valid number")
end if
```

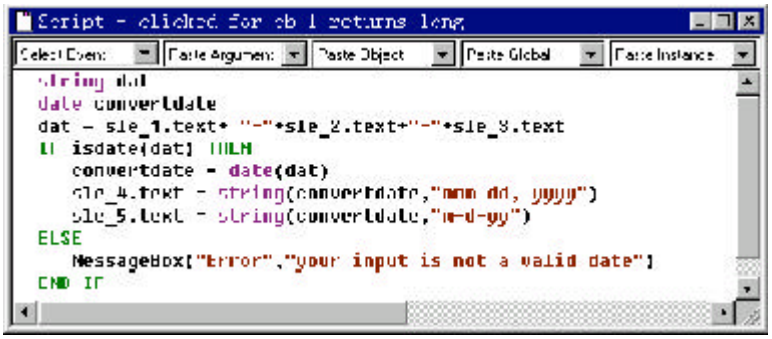
3 执行程序，其画面如下。



4



5 在命令按钮的 Script 中加入如下代码。首先判断输入的字符是否是一个合法的日期，如果是则利用 String 函数将其转化为不同的形式并输出，否则给出出错信息。



6 执行程序，如下图所示，输入的日期被转化为不同形式的字符串并输出。

7 在本例中，介绍了如何将类型判断函数和类型转换函数相结合的使用方法。尽管可以用 EditMask 控件来实现限制性输入，但在有些场合我们仍需要 Is 族函数来进行类型判断。

例

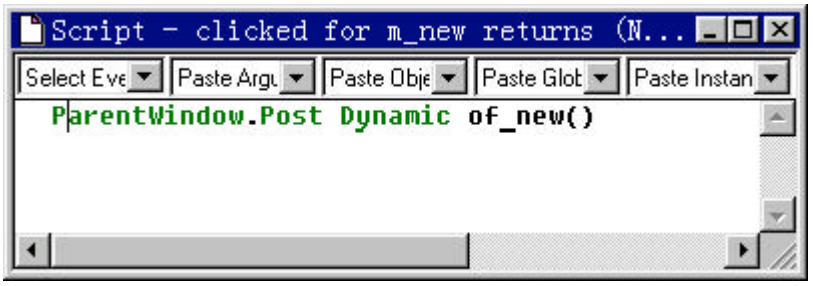
常用函数和语句（三）
Trigger 和 Post

解

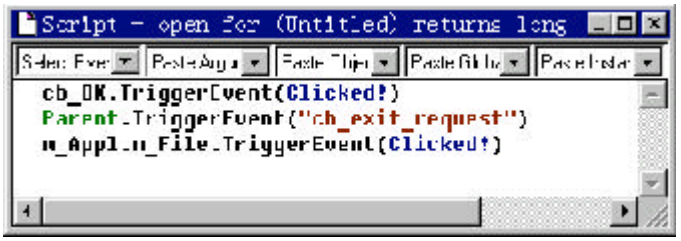
1

2 如下为窗口 W_RTE 的窗口函数，接着要在菜单中调用这些函数。

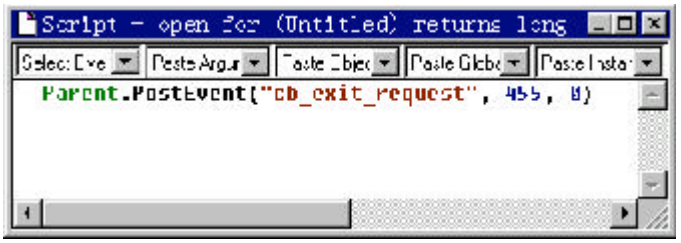
3 如下为菜单中调用窗口函数的代码。Dynamic 的作用是通知编译器所调用的函数在编译时无需存在，而只需相信即可。



4



5 下面这个例子带参数的执行某个事件，并且从消息对象中反馈参数值。首先带参数的执行该事件，如图所示。



6 在所要执行事件的开始部分加入如下代码，从消息对象中获取执行该事件的参数。

7 在本例中，介绍了 Post 和 Trigger 两种不同的调用方法，读者要认真研究这两种调用的区别。

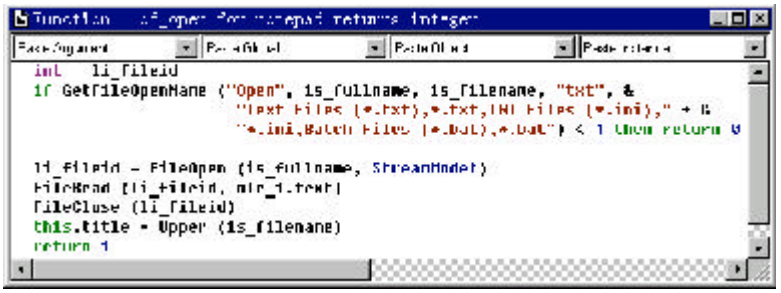
例

常用函数和语句（四）
文件函数的使用

解

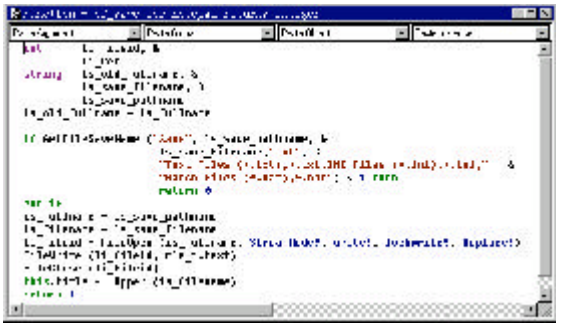
1

2 选择菜单上的 Declare|Window Functions，建立一个名为 Of_Open 的函数，代码如下图所示。用 GetOpenFileName 获取文件名，再用 FileOpen 以流模式打开文件，最后使用 FileRead 将文件读到文本编辑器中。

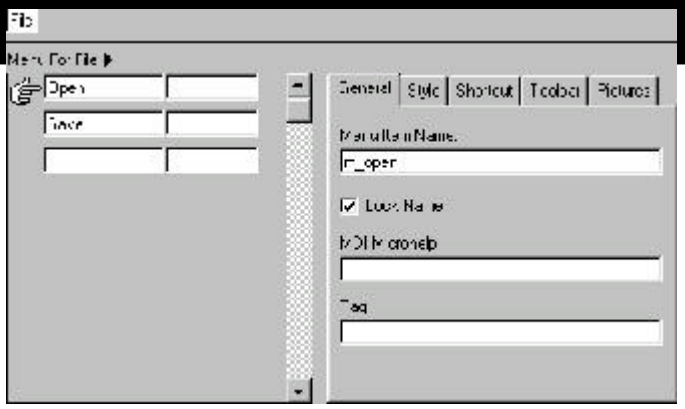


```
int   li_fileid
if GetFileOpenName ("Open", is_fullname, is_filename, "txt", &
                    "Text Files (*.txt),*.txt,INI Files (*.ini),"
                    "*.ini,Batch Files (*.bat),*.bat") < 1 then return 0

li_fileid = FileOpen (is_fullname, StreamMode!)
FileRead (li_fileid, mle_1.text)
FileClose (li_fileid)
this.title = Upper (is_filename)
return 1
```

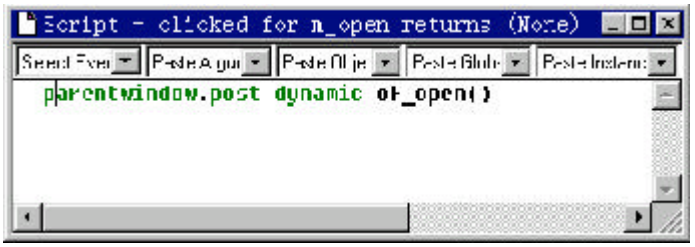
3 选择菜单上的 Declare|Window Functions，建立一个名为 Of_Save 的函数，代码如下图所示。用 GetSaveFileName 指定文件名，再用 FileOpen 用流模式打开文件，再用 FileWrite 将本编辑器内容写入文件。



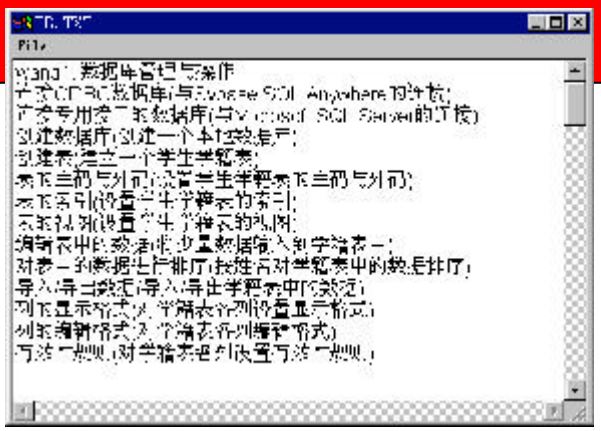
4



5 在菜单项的 Script 中，分别调用 Of_Open()和 Of_Save()函数，注意使用 Post 和 Dynamic。



6 将菜单连接到窗口上，然后执行程序，如下图所示。



7 在本例中，介绍了各种文件函数的结合操作。这种结合形式一般是固定的，希望读者牢加掌握。

例

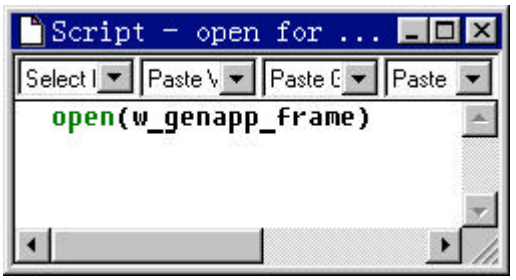
常用函数和语句（五）
其它函数和语句

解

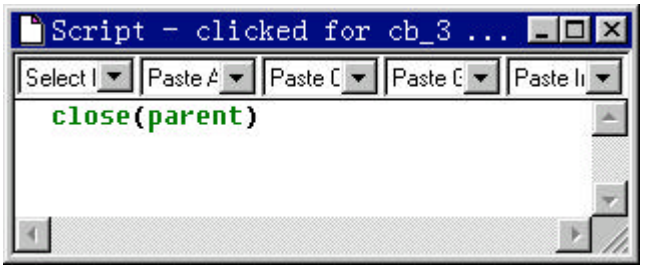
1

2 运行该窗口，可以看到鼠标距离窗口左上角越远，窗口颜色越红。

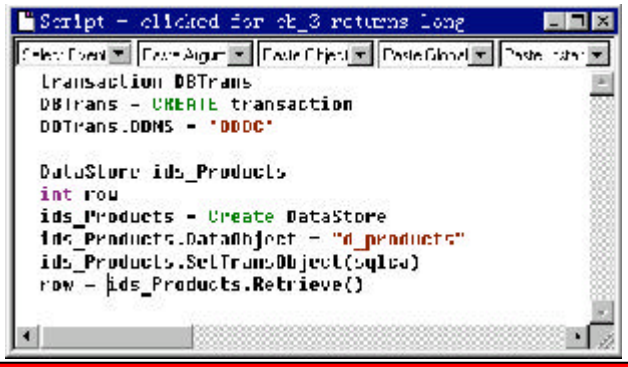
3 Open 和 Close 函数用来打开和关闭窗口，如下为应用程序 Open 事件中的代码，打开主窗口。



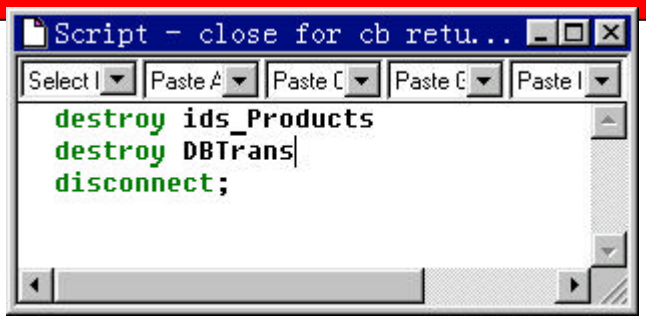
4



5 Create 和 Destroy 语句用来创建和释放类。如下为创建一个事物对象和 DataStore 对象的代码。



6 在使用完对象以后，需要用 Destroy 语句将这些对象释放。



7 在本例中，介绍了 PowerBuilder 中的几种常用函数和语句，实际上 PowerBuilder 的函数远不止这些，有兴趣的读者可以参阅帮助。

返回总目录

第八章

PBL库

本章导读

每一个 PowerBuilder 中的应用程序和对象，都被保存在 PowerBuilderLibrary（PBL 库）中，该库是以 PBL 为扩展名的文件，也是编译以前从 Windows 浏览器中唯一可见的文件对象。有时一个较大的应用程序被存储在多个 PBL 库中，每个库中分别存放不同类型的对象。库的大小应在 800K 左右为宜，对象个数应保持在 50 个左右，这主要是出于检索速度和文件个数的考虑。

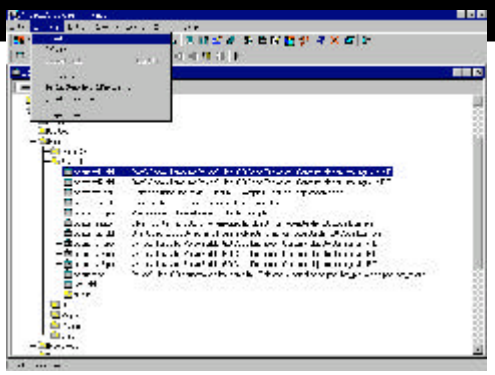
本章将从两个层次阐述和 PBL 库有关的操作。一个层次是库操作，如创建库、删除库、优化库等等；另一个层次是对象操作，包括在不同库之间复制、移动对象，在同一个库中删除对象等等。本章的重点将是介绍 PBL 描绘器的使用方法。

例

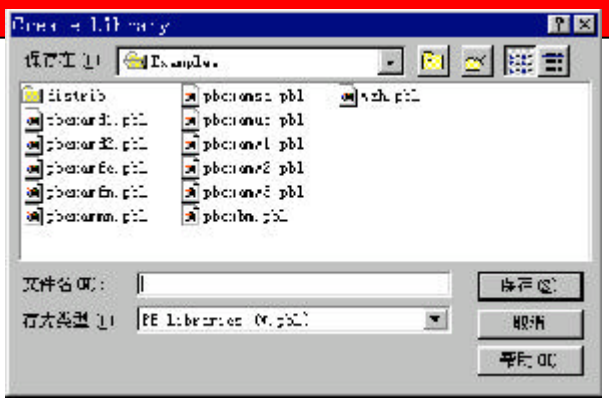
创建/访问 PBL 库
库的基本操作

解

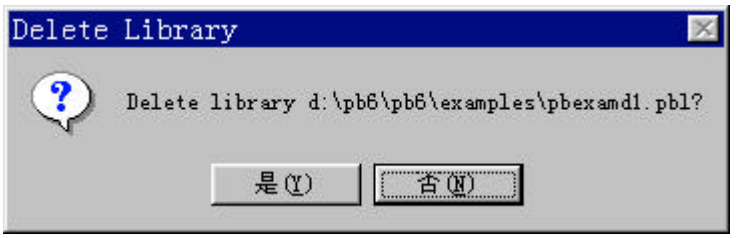
1



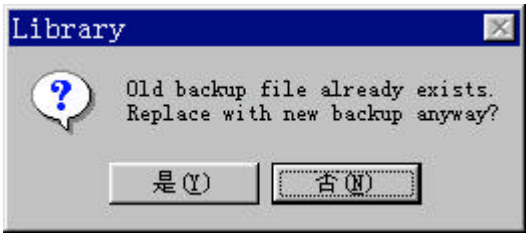
2 用户要创建 PBL 库，只需选择菜单上的 Library|Create，然后在对话框中输入要创建的 PBL 库的名称并保存即可。



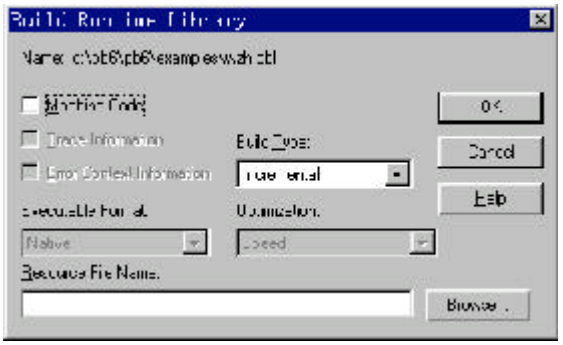
3 用户要删除一个 PBL 库，只需选择菜单上的 Library|Delete 命令，然后在确认对话框中按“是”即可。注意：不能删除当前库。



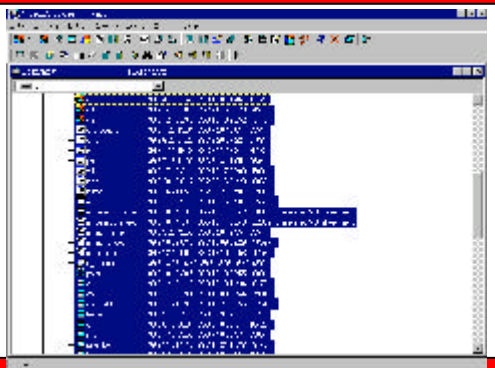
4



5 选择菜单上的 Library|Build Runtime Library，可以在当前库上创建动态库（PBD）。在对话框中，Resource File Name 用来指定资源文件名，Machine Code 生成机器码。单击 OK 即可。



6 在菜单上选择 Library|Select All，可选定当前 PBL 库下的全部对象。注意：PBL 库必须处于打开状态。再灵活使用 Shift 和 Ctrl 键，可对对象进行各种灵活多样的选择。



7 在本例中介绍了库操作中较为重要的几项操作，打印目录和属性功能比较简单，有兴趣的读者可自行上手。

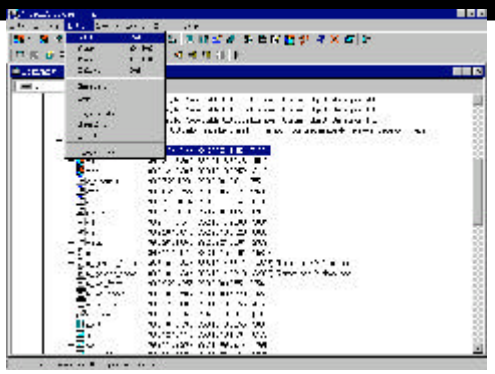
知识卷

例

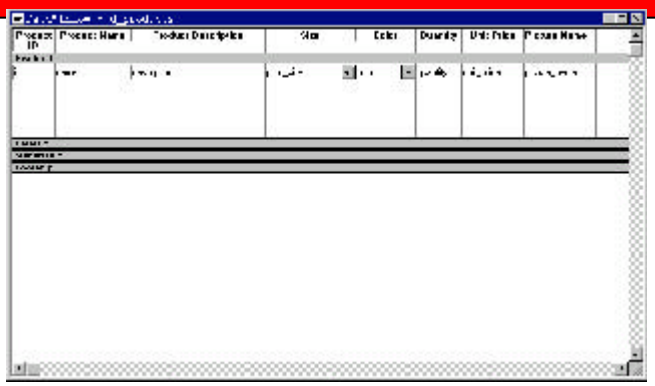
PBL 库中的对象
对象操作

解

1



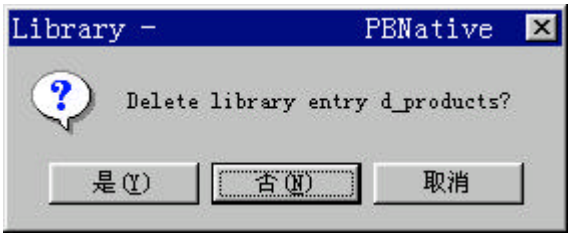
2 选择菜单上的 Entry|Edit，或按回车键，或用鼠标双击对象，就会弹出该对象的编辑画面。



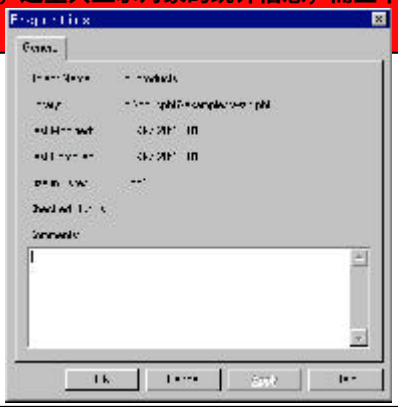
3 选择菜单上的 Entry|Copy，或按 Ctrl+C，可以将当前对象复制到另外一个 PBL 库中，下图为所要移植到 PBL 库的选择窗口。

4

5 在菜单上选择 Entry|Delete，或直接按 Del 键，当前对象将被删除。下图为删除前的确认对话框。



6 选择菜单上的 Entry|Properties，将弹出对象的属性窗口，与编辑状态所显示的属性不同，这里只显示对象的统计信息，而且不能修改，如下图所示。

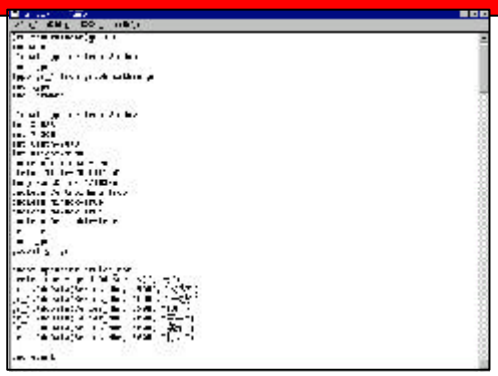


7 在本例中，我们介绍了一些对对象的基本操作，它有点类似于文件操作。在这里，也可以将 PBL 库当作一个文件目录来进行理解。

例 PBL 库中的对象 对象操作(续) 解

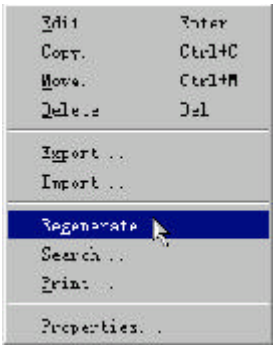
1

2 可以用文本编辑器打开输出的 sr*文件，从中可以看到 PowerBuilder 的编码风格。

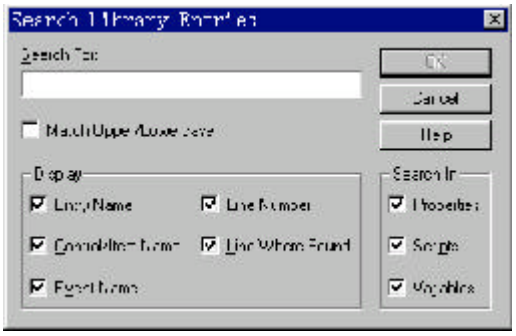


3 执行菜单上的 Entry|Import 命令，会弹出 Select Import Files 对话框，选择扩展名为 sr*的文件，在指定完目的库之后，即可导入文件。

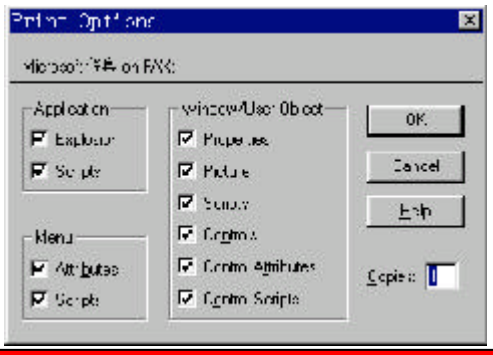
4



5 选择菜单上的 Entry|Search 命令，可以从一个对象的代码中查找某关键字，用户可以通过它查询变量、函数和语句的调用情况。



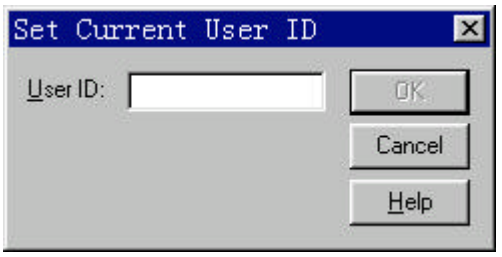
6 选择菜单上的 Entry|Print 命令，用户可以将当前对象中的代码打印出来，方便阅读。下图为打印参数的设置窗口。



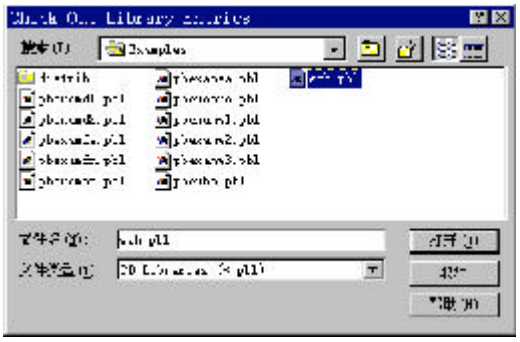
7 在本例中，介绍了对象操作的几项重要功能。它们对于编制和调试程序有很大帮助，希望读者能灵活掌握。

例 对象访问控制 取出和调回对象 解

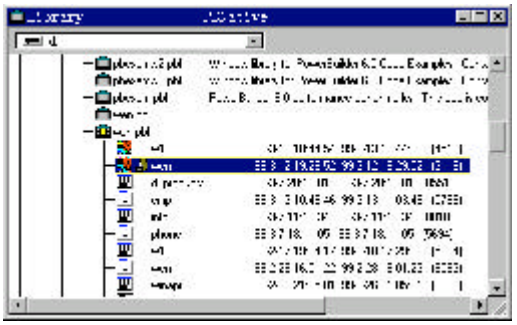
1



2 在弹出的 Check Out Library Entries 对话框中，用户被要求选择将对象取出后存放的 PBL 库，该 PBL 库应该在应用程序的库列表中。



3 当一个对象被取出后，在该对象旁将出现如下图所示的取出标记。如果试图打开该对象，PowerBuilder 都会给出警告信息，说明该对象已禁止写。



4

5 选择菜单中的 Source|View Check Out Status 命令，用户可以观察一个对象的取出状态，其中包括 ID，对象从哪里取出，存放到哪个库中。

6 当取出的对象修改完毕后，选择菜单上的 Check In 命令，将该对象调回到原来的 PBL 库中，下图为调入完成时的信息。

7 在本例中，我们介绍了将对象取出和调入的全过程。在多人通过网络协作编程时，保持代码的清洁非常重要。这时我们就需要用到 Check In 和 Check Out 命令。

返回总目录

第九章

生成应用程序

本章导读

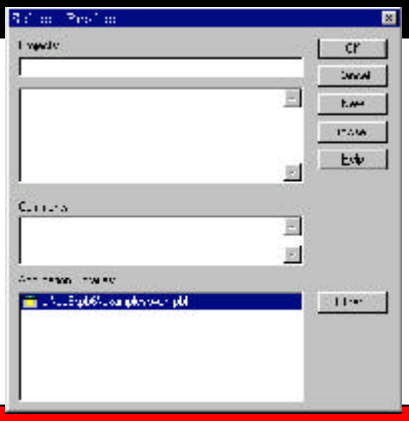
对任何一种开发软件来说，所编制的程序最终都要被生成为一个可执行文件。只有这样才能脱离开发环境而独立运行。在 PowerBuilder 中，有两种办法可生成可执行文件。一种是将所有对象都包括在一个 EXE 文件中，另一种是生成动态库 PBD 实现代码共享。

对任何编程工具来说，一个好的调试工具都是极其重要的，因为即使是一段很小的程序，都难免有这样或那样的错误。尽管语法错误可以由编译器即时发现，但逻辑错误却只能自己跟踪检查。PowerBuilder 为用户提供了 Debug 描绘器，它拥有一般调试工具的基本功能。

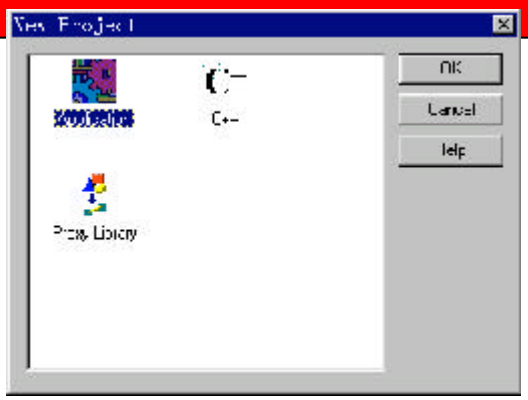
本章将分别介绍生成可执行文件与调试程序的方法，在中间会穿插两者的结合。

例 **生成可执行文件**
生成一个简单的 EXE 文件 解

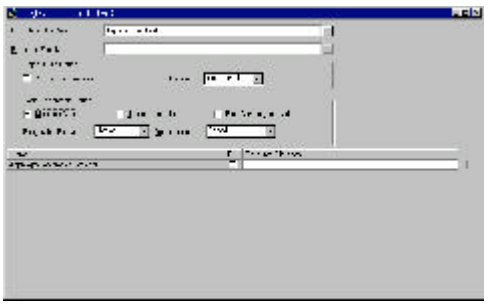
1



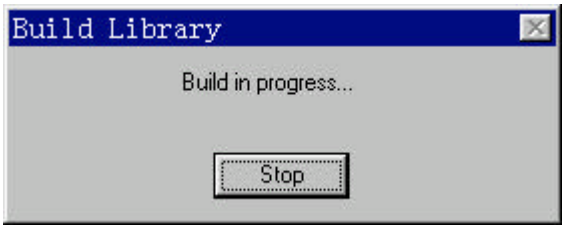
2 在下图所示的对话框中选择 Application。



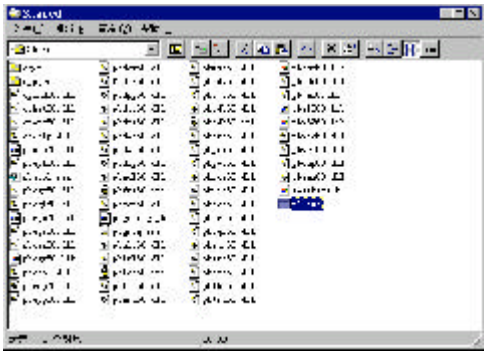
3 在出现的工程描绘器中设置如下，在 Executable File Nmae 中输入可实行文件名，将文件放在 Shared 目录下，在 Code Generation Options 中选中 Machine Code（机器码），然后单击工具栏上的 Build 按钮。



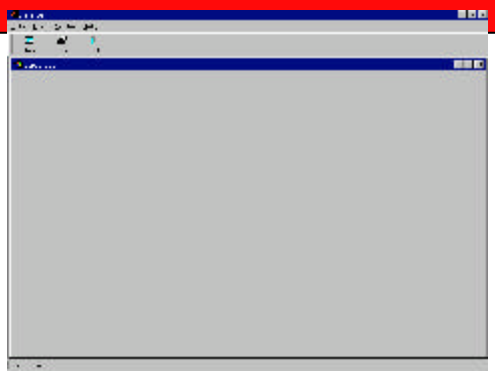
4



5 生成的 EXE 文件在 Shared 目录下，由于 PowerBuilder 生成的可执行文件都需要 Shared 目录下的某些 Dll 文件才能运行，所以运行时要把需要的 Dll 文件拷至执行文件目录下。



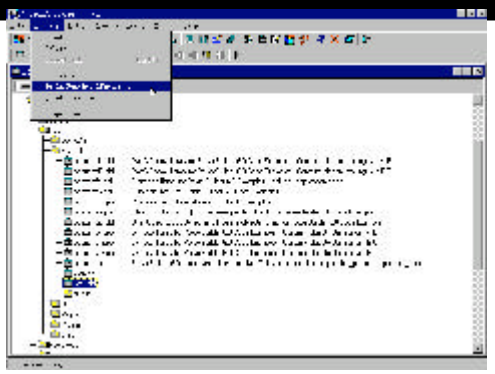
6 执行程序如下图所示，可以看到和在 PowerBuilder 环境下执行时一样。



7 在本例中，介绍了在 PowerBuilder 下生成可执行文件的一种方法，这种方法尽管简单，但并不经常使用。通常使用的方法将在下面介绍。

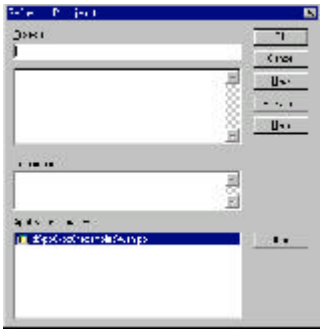
例 动态库 PBD 文件 用不同的方法生成动态库 解

1

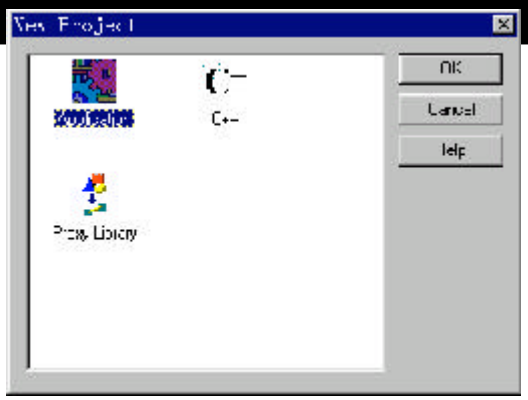


2 在该对话框中，选择 Machine Code 将生成机器码的 Dll，否则将生成伪码的 PBD。单击 OK 即可生成动态库文件。

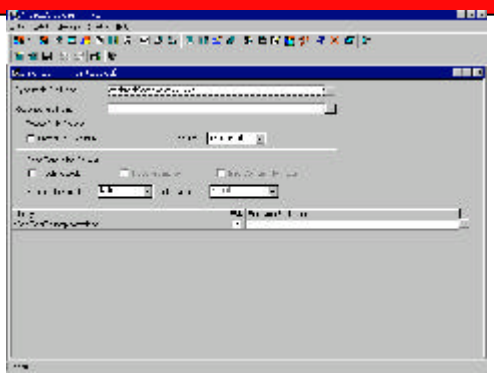
3 单击 PowerPanel 上的 Project 按钮，进入 Project 对话框，创建一个工程。



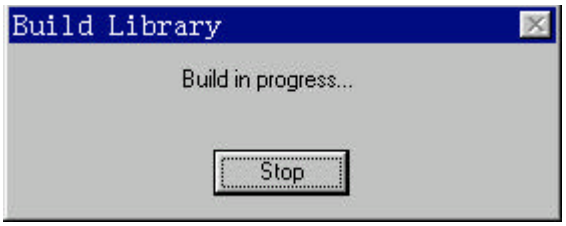
4



5 在 Project 描绘器中，将 PBD 一栏打上对勾，如下图所示。然后单击工具栏上的 Build 按钮。



6 当编译完成之后，可执行文件和 PBD 动态库文件将在同一个目录下，应把它们都拷贝到 Shared 目录下。



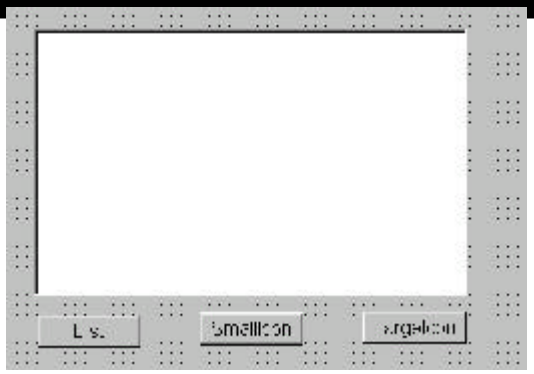
7 本例中介绍了两种建立动态库的方法，动态库的优点在于代码共享，多个应用程序可使用同一个 PBD 文件。

例

资源文件
创建和使用资源文件

解

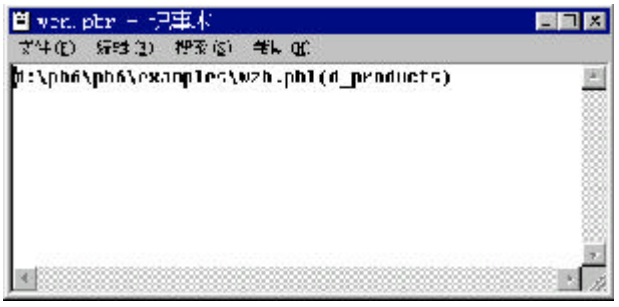
1



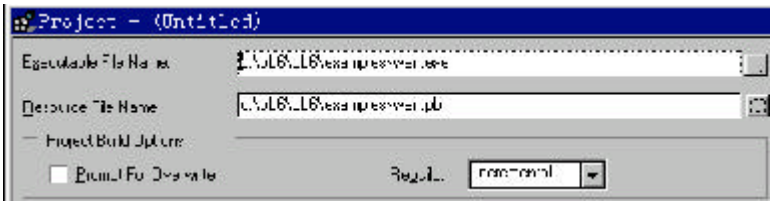
2 在代码中可以看到，程序动态的将数据窗口 D_Products 和 DataStore 控件相连，因此需要资源文件。

```
Integer     li_Rows, li_Cnt
ListViewItem   llvi_Item
ids_Products   Create DataStore
ids_Products.DataObject   "d_products"
lv_1.LargePictureWidth   48
lv_1.LargePictureHeight   48
lv_1.SmallPictureWidth   24
lv_1.SmallPictureHeight   24
lv_1.DeleteItems()
ids_Products.SetTransObject(sqlca)
li_Rows   ids_Products.Retrieve()
For li_Cnt   1 To li_Rows
   llvi_Item.PictureIndex   lv_1.AddLargePicture(ids_Products.Object.picture_name[li_Cnt])
   lv_1.AddSmallPicture(ids_Products.Object.picture_name[li_Cnt])
   llvi_Item.Data   ids_Products.Object.id[li_Cnt]
   llvi_Item.Label   ids_Products.Object.name[li_Cnt]
   lv_1.AddItem(llvi_Item)
Next
return 1
```

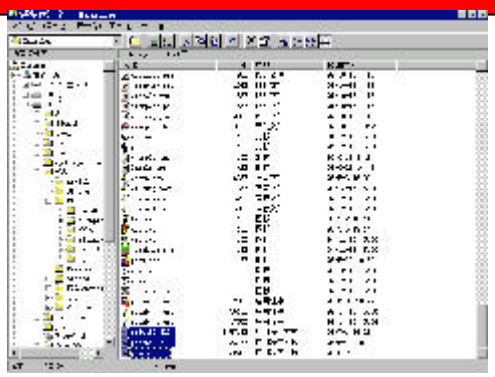
3 创建并保存一个 PBR 文件，如下为一个用记事本编辑的资源文件，内容为在 PBL 库 WZH 下的数据窗口 D_Products。



4



5 可执行文件和 PBD 库生成完毕后，将 Shared 目录下的 PBDWE60.DLL、PBODB60.DLL、PBVM60.DLL 三个文件拷贝至可执行文件所在的目录中。



6 执行文件，如下图所示，ListView 控件与数据库联接状态良好。

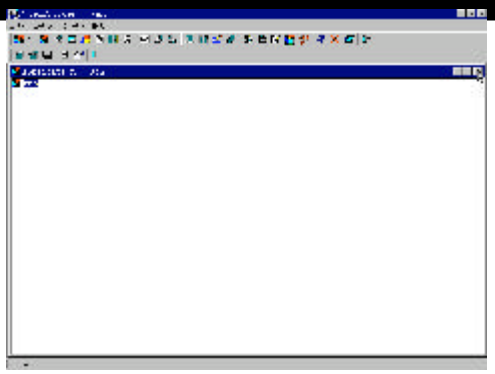
7 在本例中，介绍了一个将数据窗口加入资源文件的例子。除此之外，资源文件中常包括的还有 BMP 文件和 ICO 文件。

例

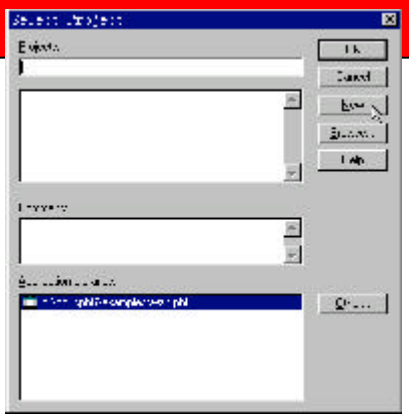
生成可执行文件
生成一个完整的应用程序

解

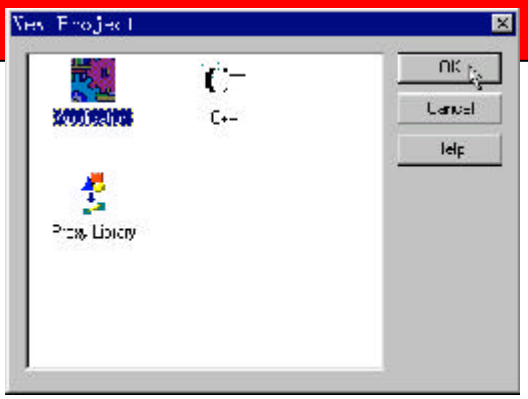
1



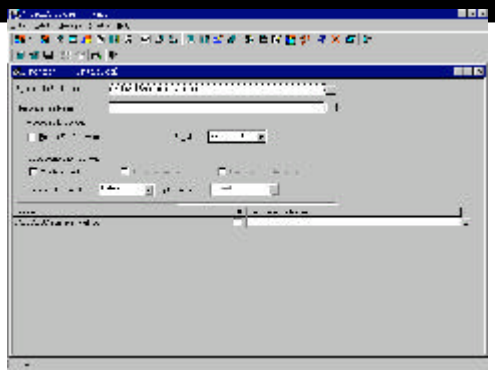
2 单击 PowerPanel 上的 Project 按钮，在 Select Project 对话框中选择 New。



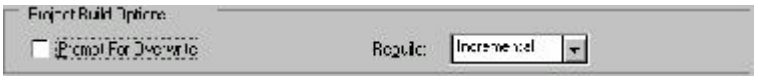
3 在弹出的 New Project 对话框中选择 Application 图标，然后单击 OK。



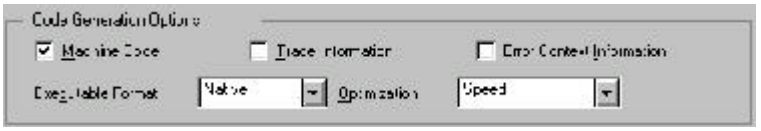
4



5 Prompt for Overwrite 复选框用来检查生成文件时是否有覆盖。Rebuild 用来确定生成文件的方式，将重建 PBL 库中所有的对象，增量再生只重建发生变化的对象或被发生变化的对象所影响的对象。



6 Machine Code 用来指定生成的是机器码还是伪码。若选定复选框，则生成机器码的 DLL，否则生成伪码的 PBD。



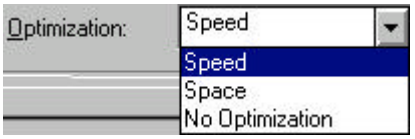
7 本例对 Project 描绘器有一个大概的描述，读者要想获取更详细的信息，可参阅帮助中的 Project painter overview 一节。

例 生成可执行文件 生成一个完整的应用程序（续） 解

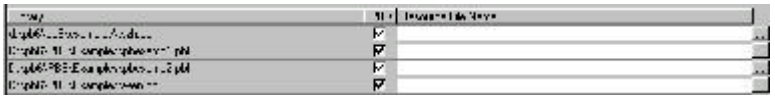
1

2 Executable Format 程序的运行方式，可在下拉框中选择生成程序的运行状态是 32 位（Native）还是 16 位。

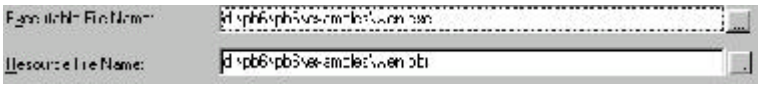
3 Optimization 优化模式，用户可在下拉框中选择对程序进行速度优化还是空间优化，或者根本不进行优化。



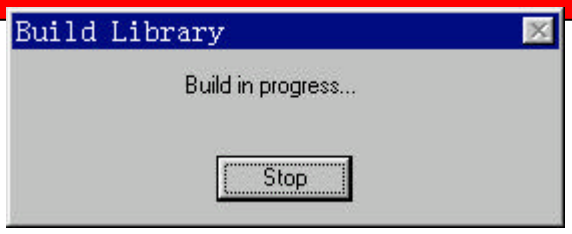
4



5 为程序指定资源文件。



6 单击工具栏上的 Build 按钮，生成应用程序。



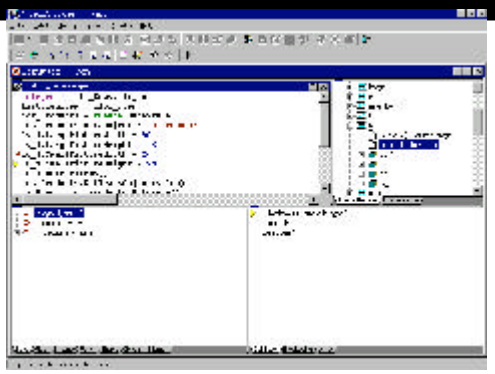
7 在本实例中，给出了一个完整的生成可执行应用程序的过程，这也是在 PowerBuilder 中生成可执行程序的一般方法。

例

应用程序的调试（一）
Debug 描绘器简介

解

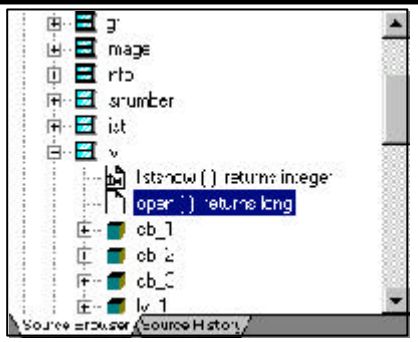
1



2 用户在 Debug 描绘器的工具栏中，可以执行一些调试工具的常用命令，如 Step In、Step Out、Step Over、编辑断点，添加 Watch 等功能。

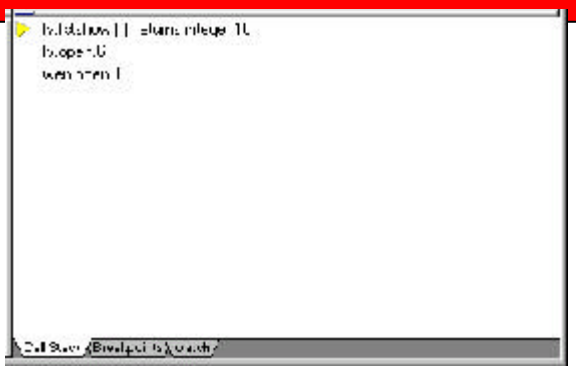
3 用户可以在 Debug 描绘器左上角的工作区调试代码。工作区的代码为当前正在进行调试的代码，用户可双击某行代码为其设置断点，并可单步执行各行代码。

4



5 在 Debug 描绘器的左下角，可以察看当前对象中各变量的数值以及随着程序执行各变量值发生变化的情况。

6 用户可以通过 Debug 描绘器右下角的窗口来观察各数据栈的内容。



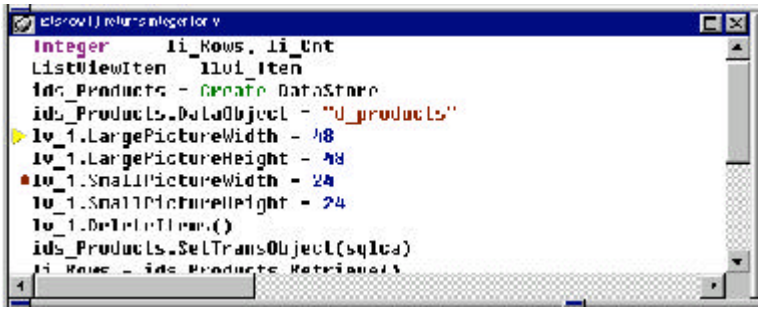
7 在本例中，我们介绍了 Debug 描绘器的基本组成，在以下几个小节中，将就其各项功能一一进行讨论。

例

应用程序的调试（二）
利用断点调试程序

解

1

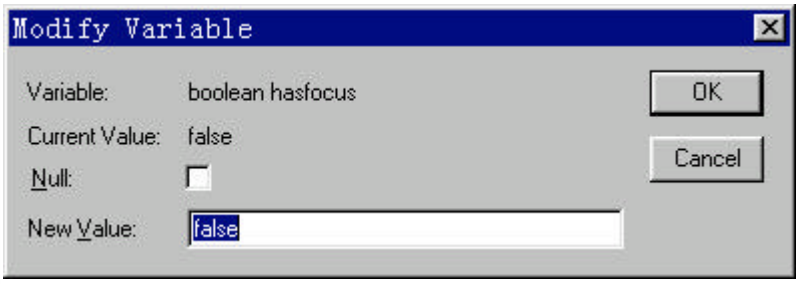


2 选择工具栏上的 Edit Stop 按钮，可以进行断点编辑。用户可以添加删除断点，也可以为已有的断点添加条件。

3 工具栏中的如下五个按钮，分别是按不同方式进行逐行代码调试的功能按钮。

4

5 为了便于调试，用户可以通过双击变量窗口中的某变量来改变该变量的当前值，如下所示。



6 在 Call Stack 窗口中，用户可以观察当前执行 Script 的调用关系，即当前程序的调用属主的层次关系。

7 在本例中，介绍了利用断点进行程序调试的方法，这是程序化设计中的一种普遍方法。

例

应用程序的调试（三）
跟踪并记录应用程序的运行

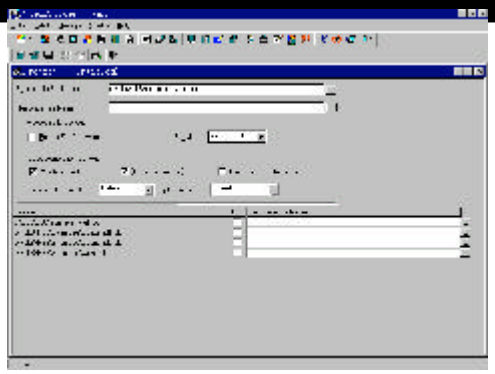
解

1

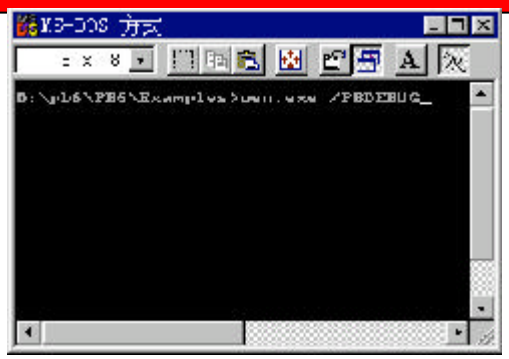
2 在 System Options 窗口的 Gerneral 标签页中，用户可以设定允许对程序运行过程进行跟踪，并可以规定跟踪文件的输出位置。

3 在 Profiling 标签页中，用户可以规定所要跟踪的语句和活动的范围，并可以规定时间的记录方式。

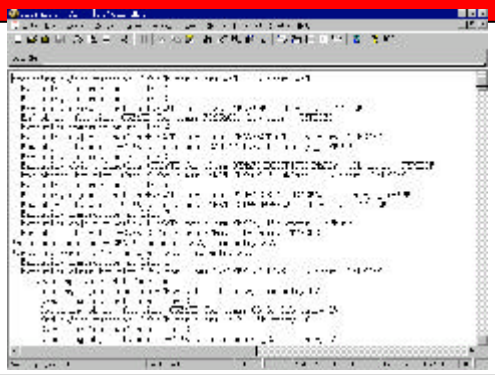
4



5 对于已经生成的可执行文件，可以用参数/PBDEBUG 来执行该文件，使得该程序在执行完毕后会产生一个记录文件。



6 生成的 DBG 跟踪文件如下图所示，它记录了程序执行的全过程。



7 在本例中，我们介绍了三种在执行文件时进行跟踪的方法，用户也可以通过代码来实现以上功能。