

第 3 章 PowerScript 编程语言

教学提示：PowerBuilder 作为一种程序开发工具，具有使用编程语言编写代码的功能。这种编程语言就是 PowerScript。它是由语句、命令、函数的集合以及嵌入式 SQL 语句组成。PowerScript 是一种面向对象的、结构化的高级编程语言，代码的执行由事件或另一个脚本发送的消息激活，通过为对象的事件或函数编写代码将 PowerScript 代码与 PowerBuilder 应用程序的对象相关联。为了深入理解 PowerBuilder、灵活高效地编写功能强大的 PowerBuilder 应用程序，掌握 PowerScript 语言是非常必要的。

教学要求：通过本章的学习，读者应熟悉并掌握以下内容：PowerScript 基本概念，变量与常量的数据类型、作用域、声明及引用，PowerScript 基本语句的格式要求，函数与结构的定义及引用，嵌入式 SQL 的应用。

3.1 PowerScript 语言基础

每种编程语言都有自己的一组基本语法约定，PowerScript 语言作为 PowerBuilder 的编程语言也不例外。事件处理程序、用户自定义函数等都是使用 PowerScript 语言编写的。这是一种自由格式的语言，它很宽容，也就是说语句行中的空格缩进等格式编排信息完全被编译器忽略，为了阅读方便可以随意安排语句行的位置。与其他大多数编程语言相似，PowerScript 提供了断行、续行的方法。如果愿意在一行中写上几条语句也毫无问题，另外该语言提供了两种注释方法，通过注释可以进一步提高程序的可读性。本节简单介绍一下 PowerScript 语言的书写规则、标识符、注释、续行符、空值、代词等成分的格式和用法。

3.1.1 书写规则

PowerScript 语言的书写格式比较自由灵活。为阅读清晰起见，一般每行只写一条语句（当然也可以按照一定的规则书写多条语句），语句行中的空格、缩进、制表符等格式编排信息将被编译器完全忽略。由于编译器即时编译方式、语句中的任何错误都将导致程序的库文件不能被保存，因而每行只写一行语句将便于用户及时发现并修正程序错误。

在 PowerScript 中，一条语句结束后不需要在句末追加分号或其他任何符号，这一点同 Pascal 语言和 C 语言是不同的。但当在某一行中书写多条语句时，此行中除最后一条语句之外的其他语句均需加在句末加上分号，例如：

```
E=A+B; F=C+D; G=E+F
```

PowerScript 中语句的概念同其他常见的编程语言也有一定的区别。例如，在 C 语言中以下语句：

```
If (Startflag=tured) write ("Hello World!"); else
```

```
Write (" Not prepared yet! ");
```

是合法的。但在 PowerScript 中，若某行有语句写为：

```
IF (StartFlag = true) THEN return 1 else return 0 END IF
```

则是错误的，因为这一行中实际涉及 5 条语句，正确的写法应该是：

```
IF (StartFlag = true) THEN; return 1; ELSE; return 0; END IF
```

或者去掉分号将这 5 条语句分别写到 5 行中。在 PowerScript 中，书写条件语句、循环语句等语句时需要特别注意这些问题。

在书写代码的过程中，用户更多地还将遇到断句、续行、注释等问题。其具体的书写格式将在后面的有关章节做详细介绍。

3.1.2 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。在 PowerBuilder 9.0 中，标识符的命名遵从下述规则。

- (1) 标识符必须以字母或下划线“_”开头。
- (2) 标识符只能由字母、数字和如下特殊字符组成：下划线“_”、美元符号“\$”、短横线“-”、号码符号“#”和百分符号“%”。
- (3) 标识符不能是 PowerScript 的保留字。
- (4) 标识符不区分大小写。
- (5) 标识符最长 40 个字符，并且中间不能插入空格。

例如：下面是一组正确的标识符。

```
Name  
f_add  
Button#1  
_specialID
```

而下面标识符的写法是错误的。

```
total book           //标识符中间不能有空格  
this                 //不能使用保留字 this  
abc>def              //标识符中间有非法字符  
3rd_user             //不能以数字开头
```

提示：短横线与减号是一个字符，而 PowerScript 中允许短横线用在标识符中。因此表达式中使用减法运算时，必须在减号的两边分别加上空格，否则可能产生语法错误。

为了避免出现这种二意性错误，可以在 PowerScript 中设置禁止在标识符中使用连字符。设置的方法为：在菜单中选择【Design】|【Options】菜单，将弹出【Options】对话框，选中对话框的【Script】标签页，取消【Allow Dashes In Identifiers】复选框的选中标记，这样就可以禁止连字符在标识符中出现，这样，在程序中使用的连字符都作为减号来处理。

3.1.3 注释


注释用于书写说明，PowerScript 有两种类型的注释：行注释和块注释。

(1) 单行注释用“//”，从双斜杠开始到行尾均为注释。

(2) 块注释以“/*”开始，到“*/”结束。在“/*.....*/”中的所有的代码均为注释。

例如：下面就是两种注释的例子。

```
//这是一个计算器程序
decimal    add1          //add1 表示中间结果
char       op            //op 表示按下的操作符
int        flag
/* flag=1 表示按下的数字是前面数字的部分
   flag=0 表示按下的数字是一个新的数字的开始*/
```

PowerScript 在工具栏上提供了将所选中的行加注释和取消注释的两个图标，用起来非常方便。

默认情况下，PowerBuilder 9.0 的编辑窗口中所有注释均以蓝色显示。

3.1.4 续行符

多数情况下，每条语句占一行，但有时会遇到语句超长的情况，为阅读方便可以将语句分成几行，这就需要用到续行符“&”将语句串起来。续行符必须加在行尾。

如果行尾恰好是符号“&”，那么必须再加一个“&”才能续行。例如：

```
IF side1=5 AND &
   side2=6 THEN area=6*8
// 相当于语句 IF side1=5 AND side2=6 THEN area=6*8
```

提示：不能在标识语言或保留字的中间续行。

3.1.5 空值

空值(NULL)是 PowerBuilder 与数据库交换数据时使用的一种特殊值，它代表变量的数据未定义或不可知。它与空字符串、数值零和日期 0000-00-00 的意义完全不同，与 NULL 进行运算的结果都为 NULL。空值既不是数值零，也不是非零的任何数值。

所有 PowerBuilder 数据类型都支持空值，但 PowerBuilder 并不是把它作为变量的默认初值。例如，当一个变量被说明而未被初始化时，PowerBuilder 把 0 赋给数值型变量，把 False 赋给布尔型变量，把空串“”赋给字符串变量等。

为变量赋空值有两种方法。

(1) 从数据库中读取空值。

(2) 用 SetNull()函数给变量赋值。下面是使用 SetNull()函数的一个例子。

```
String city          //这时 city 的值为 ""
SetNull(city)        //这时 city 的值为 NULL
```

PowerScript 中测试变量和表达式的值是否为空值，应该调用 IsNull()函数，而不应该用等号，例如：

```
IF IsNull(city) THEN RETURN //正确的写法
IF city=NULL THEN RETURN //不正确的写法
```

3.1.6 常用代词

PowerScript 中提供了 4 个代词，分别是：This、Parent、ParentWindow 和 Super，常用的是前面 3 个。使用代词主要是为了增加代码的通用性，不受所指代的控件或对象的名称发生变化的影响。

1. This

代词 This 代表窗口、用户对象、菜单、应用对象或控件对象本身，即代表正在为之编写事件处理程序的对象。

例如，窗口中有一个名称为 cb_button 的按钮，按钮上显示的文本为“请单击”，在该按钮的 Clicked 事件处理程序中可以写上代码：

```
cb_button.text = “再单击一次”
```

程序运行后，单击该按钮时其显示文本变成“再单击一次”，但当把这段程序粘贴到其他窗口的按钮事件处理程序中时，它可能不能正常工作。原因在于其他按钮的名称并不一定是 cb_button。现在把上面的程序段修改为：

```
This.text = “再单击一次”
```

那么无论将它粘贴到哪个按钮的事件处理程序中，它都能正常工作，这里的 This 代表该按钮。

2. Parent

Parent 可以在窗口的控件、用户定制对象和菜单的程序中使用，当在窗口的控件当中使用 Parent 的时候，Parent 指向包容该控件的窗口。

例如：要对 w_win 窗口对象中的按钮 Clicked 事件编写处理程序，其功能是单击后关闭窗口 w_win，可以在按钮的 Clicked 事件处理程序中写上代码：

```
Close(w_win)
```

但采用代词 Parent 把这条语句修改为：

```
Close(Parent)
```

则使用代词具备更强的通用性和可移植性，并且也更容易理解。

3. ParentWindow

ParentWindow 代表运行时菜单所在的窗口，该代词只能在菜单的事件处理程序中使用。例如，在一条菜单命令的脚本中输入如下代码的话将关闭菜单所在的窗口。

```
Close(ParentWindow)
```

4. Super

在编写控件或对象的子对象程序时，可以调用祖先的程序，用户可以直接使用祖先对

象的名称调用它们，也可以使用 Super 来引用直接双亲。例如要调用父类的 Clicked 事件，脚本可以这样编写：

```
CALL Super :: Clicked
```

当调用在子类中被重载的父对象的函数时，也可以使用 Super 来指代父对象。例如子对象重载了父对象的函数 f_func()，在子对象中调用父对象的 f_func() 函数，语句可以这样编写：

```
Super: : f_func()
```

3.2 数据类型

PowerBuilder 中提供了丰富的数据类型，包括标准数据类型、枚举类型、系统对象数据类型 3 大类，程序中通过数据类型限定变量的取值范围。

3.2.1 标准数据类型

标准数据类型包括数值型、字符型、日期型、布尔型等一些基本的数据类型。

- Blob 类型：大二进制类型，用于存放图像、大文本等数据。
- Boolean 类型：布尔型或逻辑型，取值为 True 或 False。
- Char 或 character 类型：字符类型，用于存放单个 ASCII 字符。
- Date 类型：日期类型，基本格式为 yyyy-mm-dd，其中年的取值范围是 1000~3000。
- Datetime 类型：日期时间类型，仅用于读取数据库中的 datetime 字段的值。可以使用 Date(Datetime)或 Time(Datetime)函数将 Datetime 值转为 PowerBuilder 的数据类型 date 或 time；可以使用 DateTime(date, time)函数将 date 和 time 值转换为 DateTime 值写入数据库中。
- Decimal 或 Dec 类型：十进制数类型，最大 18 位精度。如：123.45。
- Double：双精度浮点数类型，有效精度为 15 位，绝对值范围为 $2.2\text{E}-308 \sim 1.7\text{E}+308$ 。
- Integer 或 Int 类型：整数类型，16 位，取值范围为 $-32\,768 \sim +32\,767$ 。
- Long 类型：长整型，32 位，取值范围为 $-2\,147\,483\,648 \sim +2\,147\,483\,647$ 。
- Real 类型：单精度浮点数类型，有效精度为 6 位，绝对值范围为 $1.175495\text{E}-38 \sim 3.402822\text{E}+38$ 。
- String 类型：字符串类型，字符串最大长度可达 2 147 483 647。
- Time 类型：时间类型，24 小时格式，时的取值为 00~23，分的取值为 00~59，秒的取值为 00~59，时、分、秒之间用冒号分隔，如 21: 10: 15。
- UnsignedInteger 、UnsignedInt、UInt 类型：无符号整型类型，16 位，取值范围为 0~65 535。
- UnsignedLong、ULong 类型：无符号长整型，32 位，取值范围为 0~4 294 967 295。

3.2.2 枚举类型

枚举数据类型是 PowerScript 中的一种特殊数据类型，它有以下两种用途。

- (1) 作为函数的参数；
- (2) 指定对象或控件的属性。

在 PowerBuilder 中，有些系统函数的参数、对象或控件的属性是枚举型的，每个枚举型的参数或属性都有一组固定的取值，由系统预先定义好，用户不能增加或修改。枚举数据类型的值总是用感叹号(!)结尾。比如枚举数据类型 **Alignment** 用来说明文本的对齐方式，它的取值可以是：**Center!**、**Left!** 和 **Right!**，而且只能取其中一个值。例如，多行编辑器设置为右对齐方式：

```
mel_edit.Alignment=Right!
```

提示：枚举值是系统定义好的一组特定的值，不能把这些值看作是字符串而加上引号来使用。

3.2.3 系统对象类型

系统对象数据类型也是 PowerScript 中的一种特殊数据类型，在建立 PowerBuilder 应用程序时，要用到这些对象，如 **Window**、**Menu**、**CommandButton**、**ListBox** 和 **Graph** 等类型。在 **Browser** 画板中可以通过【**System**】选项卡浏览到 PowerScript 的全部系统对象数据类型，如图 3.1 所示。

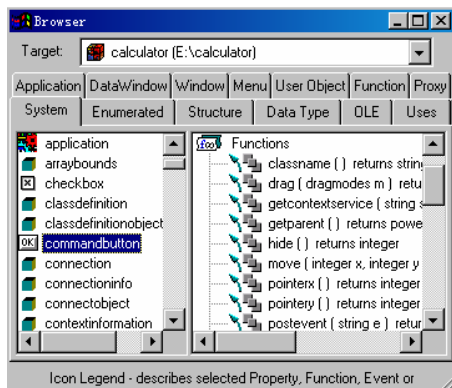


图 3.1 浏览系统对象数据类型

在 PowerBuilder 中，窗口、菜单、各种控件都是系统对象，每一种系统对象实际上都是定义在 PowerBuilder 内部的一种数据类型。一般情况下，不必把这些对象当作数据类型来考虑，而只需在相应的画板中使用它们即可。

3.3 变量和常量

对数据进行处理是 PowerScript 语言的基本功能之一，数据处理的基本对象是常量和变

量。无论是常量或是变量在使用前都应先声明，以便编译时分配适当的存储空间，并且声明的格式、位置的不同，其作用域和可见性也不同。

3.3.1 变量

变量是指在程序运行过程中其值可以改变的量。变量在程序运行过程中被赋值，在执行各种运算和控制时其值可以改变，以实现程序员预先设定的各种操作、完成各种任务。PowerScript 中提供了大量各种类型的变量，使程序设计更简捷、效率更高、内容更丰富。

在 PowerBuilder 中，所有变量在使用之前，都要首先予以说明，系统预定义的 5 个全局变量(SQLCA、SQLDA、SQLSA、Error、Message)除外。

变量声明的一般语法格式为：

[存取权限] 数据类型 标识符 [=初值], [标识符=初值][, ...]

其中方括弧括起的内容为可选项。

(1) 存取权限可以设为以下方式。

① **PUBLIC**(默认)：任何在应用中的脚本都默认使用该方式，非对象成员可使用点操作符标明域。

② **PROTECTED**：定义变量对象的脚本及其后代有权使用。

③ **PRIVATE**：除定义变量对象的脚本之外，任何对象无权使用。

(2) 数据类型可以是标准数据类型、枚举类型、在对象浏览器中出现的系统对象数据类型或者已经定义用户对象数据类型的任一种。

(3) 标识符为任何合法标识符。

在一个优秀的设计中，命名风格是良好项目管理的一部分，就像在下面所举的例子中，除符合标识命名规则外，其标识名的前两个字母有标识意义：第一个字符代表变量的作用范围，第二个代表数据类型。另外，命名需要考虑不能与对象的属性变量重名。例如 X 和 Y，它们保存对象在屏幕上的位置坐标，虽然定义了标识名 X 或 Y 的变量不会引起编译错误，但在程序运行时，PowerBuilder 将认为这两个变量保存了对象的屏幕坐标，从而引起错误。下面是一些标识符的例子：

```
Integer ii_total = 200      //实例的整型变量
Date id_date               //实例的日期变量
Time sta_begin             //共享的时间变量
String gs_text             //全局的字符串变量
Integer li_cont,li_next    //局部的整型变量，一次声明了两个变量
```

当说明变量时不赋予初值，系统将给变量自动地赋予默认值，各种数据类型的默认值见上一节介绍。

说明变量时可以直接把一常量或表达式的值赋给变量，例如：

```
Integer li_b = 100, li_total
Date sd_birth=2000-12-2
String ls_headers = "Name~tAddress~tCity"
```

但是变量的初始化是在编译阶段完成，而不是在运行时完成的，因此不要使初值随当时状态改变。例如下面一条语句中，id_date 是想保存运行时的日期，但是它不会实现设计的

初衷。

```
Date id_date= Today()
```

其中 id_date 保存的是编译时的日期而不是运行时的日期，正确的语句应该是下面的：

```
Date id_date  
id_date = Today()
```

3.3.2 变量的作用域

在 PowerBuilder 中，有四种不同范围的变量：Local（局部变量）、Instance(实例变量)、Global(全局变量)、Shared(共享变量)。

1. Global 全局变量

全局变量是在整个应用程序中都起作用的变量，其作用域为整个应用程序。

全局变量的声明方法如下。

(1) 在 Window 画板、User Object 画板或 Menu 画板中选择任意一个对象，单击右键，在弹出的菜单中选择【Script】菜单项，进入 Script 编辑器。

在事件和函数中定义的变量是 Local 变量，它的作用范围仅在所在的事件和函数内起作用，在其他事件和函数中不能被使用。

(2) 在 Script 编辑器中选择第 1 个下拉列表框，选择其中的【Declare】选项，如图 3.2 所示。

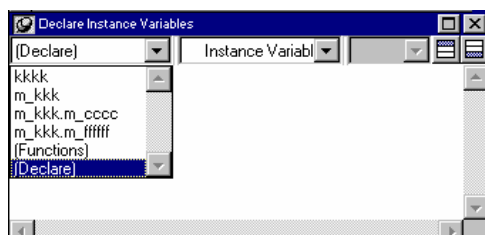


图 3.2 选择【Declare】项声明变量

(3) 在 Script 编辑器中选择第 2 个下拉列表框，选择其中的【Global Variables】选项，如图 3.3 所示。

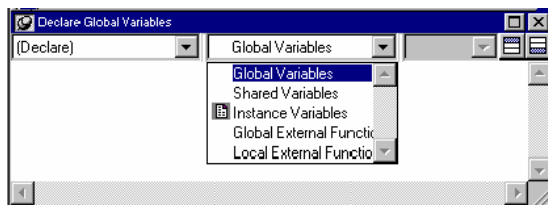


图 3.3 声明全局变量

(4) 最后在 Script 编辑器中键入所要声明的全局变量就可以了，如图 3.4 所示。

2. Instance 实例变量

实例变量是某个特定对象的数据变量，可看作该对象实例的属性。可以在应用对象、

窗口对象、用户对象或菜单对象的 Script 编辑器中定义它，定义实例变量时要指定访问权限，缺省为 PUBLIC。在其对象的事件脚本或函数中，可直接访问它，不受访问权限的限制；只有 PUBLIC 权限的实例变量，才能在其他对象的脚本中访问它，并且要通过点符号“.”。例如，在窗口 W_1 中声明了一个整型实例变量 ii_total，当在 W_1 中给它赋值时，可以这样写：ii_total=100；而当要在窗口 W_2 中使用 ii_total，将它赋值给 W_2 中的一个整型变量 i_num 时，应该这样写：i_num=W_1.ii_total。

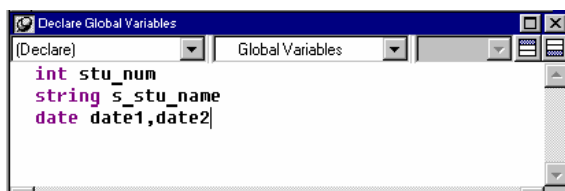


图 3.4 输入要声明的全局变量

声明一个实例变量与声明一个全局变量的方法是一样的，只是在第(3)步中选择【Instance Variables】选项，而不是选择【Global Variables】选项。

当且仅当打开与实例变量相关的对象时，实例变量才被创建，在打开对象之前。实例变量是不存在的。

通过分析一个具体的例子来加深对实例变量的理解。

假设创建了一个窗口 w_1，在该窗口中声明一个整型的实例变量 ii_total(初值为 100)。在程序运行过程中，打开窗口 w_1 之前，实例变量 ii_total 是不存在的。当打开窗口 w_1 时，将创建实例变量 ii_total，并给它初值 ii_total=100。程序继续执行，运行一段代码之后，给实例变量 ii_total 赋值为 200，然后关闭窗口 w_1，实例变量将随之消失。当再次打开窗口 w_1 时，系统又会重新创建实例变量 ii_total，并且还是赋给它初值 100。

3. Shared 共享变量

共享变量用于同一对象类型的各个实例之间共享数据。声明一个共享变量和声明一个全局变量的方法是一样的，只不过在第(3)步时选择【Shared Variables】选项，而不是【Global Variables】选项。

共享变量与实例变量很相似，它们都与某个对象相关联，但实例变量是与对象的某个特定实例相关联，而共享变量则是与该对象的定义相关联。

下面用一个例子来说明二者的区别。假设窗口 W_1 中声明了一个共享变量 ii_share(初值为 500)。当窗口 W_1 第一次被打开时，将创建共享变量 ii_share 并初始化，ii_share=500。运行一段程序代码之后，共享变量的值变为 300。然后窗口 W_1 关闭，但是共享变量并不像实例变量那样消失，而是仍然存在。当再次打开窗口 W_1 时，共享变量无须再被创建和初始化，所以当第二次打开窗口 W_1 时，ii_share 的值是 300。

实例变量是随着它的对象被打开、关闭和再次打开而被创建、删除和创建的。共享变量是与对象的定义相关联的，而不是它的任何特定实例，所以从其对象的第一个实例打开开始它总是存在的。

4. Local 局部变量

在这 4 种变量的作用范围中, 局部变量的作用范围是最严格的。这种变量只在某个程序段内或函数内部起作用。它可以在过程中声明, 在代码和函数内部的任何地方都可以使用局部变量, 但是在该程序段或函数外部的任何地方都不能使用该局部变量。在不同的函数内部使用相同的局部变量名是可以的, 不会引起混淆, 这样就增加了程序的方便性。

5. 优先级顺序

如果在一个脚本的执行过程中, PowerBuilder 发现了对某个变量的调用, 则将按照下面的顺序搜索该变量的位置。

- (1) 局部变量
- (2) 共享变量
- (3) 全局变量
- (4) 实例变量

如果 PowerBuilder 无法在上面的任一层中找到该变量, 就向上搜索该对象的继承关系, 也就是搜索对象的父对象, 查看父对象的实例变量。

3.3.3 常量

在编程过程中, 有些数据在程序的整个运行过程中均保持不变, 则这些数据在使用前适用于声明成常量。将这些数据声明成常量可以简化数据的书写, 避免发生不经意的修改。同时, 提高了程序代码的可读性和可维护性。

声明常量时需使用关键字 **CONSTANT**, 其语法格式为:

```
CONSTANT { access } datatype constname = value
```

其中, 访问权限 **access** 是可选项, 它限定了常量的使用范围, 具体取值可参见前面有关变量的说明。下面对常量的声明作以下几点说明。

(1) 数据类型 **datatype** 为除 **Blob** 类型之外的任何标准数据类型(用户不能声明 **Blob** 类型的常量), 对于 **decimal** 类型的常量, 还可在 **decimal** 后以大括号指明精度。

(2) 常量名 **constname** 为用户标识符, 其命名应尽量采用含义明确、易于领会的名称, 并建议用户在书写常量名时全部采用大写的形式。

(3) 常量值 **value** 为赋给常量标识符的值, 此值可以是常量, 或者适当类型表达式的值来指定。

(4) 在声明一个常量的同时必须为其赋初始值, 此值一旦确定就不能在程序代码的其他地方改变, 否则将产生编译错误。

以下是声明常量的几个例子。

```
CONSTANT Date CD_NATIONAL = 1949-10-01
CONSTANT Decimal {2} CE_Price = 48           //实际值为 48.00
CONSTANT Real CR_PI = 3.14159265
CONSTANT String CS_THU = "Tsinghua university"
CONSTANT Time CT_BEGAIN = 08:00:00
```

3.4 运算符与表达式

PowerBuilder 中的运算符有 4 种：算术运算符、关系运算符、逻辑运算符和连接运算符。运算符有优先级和结合方向。

3.4.1 算术运算符

算术运算符有加、减、乘、除和乘方，它们的表示方法及功能见表 3-1。

表 3-1 算术运算符

运 算 符	含 义	示 例
+	加	$C=a+b$
-	减	$C=a-b$
*	乘	$C=a*b$
/	除	$C=a/b$
\wedge	乘方	$C=a\wedge b$ a 的 b 次方

在表达式中，优先级顺序为乘方高于乘除、乘除高于加减、同级运算遵循从左至右的原则。

PowerBuilder 中还提供了一组扩展的算术操作符(与 C 语言中的算术运算符相同)，参见表 3-2。

表 3-2 扩展算术运算符

运 算 符	意 义	示 例	等 价 形 式
++	增 1	$a++$	$a=a+1$
--	减 1	$a--$	$a=a-1$
+=	加赋值	$a+=b$	$a=a+b$
-=	减赋值	$a-=b$	$a=a-b$
=	乘赋值	$a=b$	$a=a*b$
/=	除赋值	$a/=b$	$a=a/b$
$\wedge=$	幂赋值	$a\wedge=b$	$a=a^b$

3.4.2 关系运算符

关系运算符用来对相同类型的量进行大小比较运算，常用于条件语句和循环语句中。关系运算的表示及功能见表 3-3。

关系运算符的结果是 True 或 False。关系运算符可以用于数值比较，同样可以用于字符串的比较。字符串比较时，逐个字符按字符的 ASCII 值比较，所以进行字符串比较时是区分大小写的，即“A”不等于“a”，并且空格也参与比较，即“abc”不等于“a bc”。

需要时可以使用 PowerScript 提供的字符串操作函数进行适当的转换，如：Upper()转换成大写、Lower()转换成小写、RightTrim()删除右边空格、LeftTrim()删除左边空格、Trim()删除两端空格。

表 3-3 关系运算符

运 算 符	意 义	示例(假设 a=3, b=4)	结 果
=	等于	a=b	False
>	大于	a>b	False
<	小于	a<b	True
>=	大于等于	a>=b	False
<=	小于等于	a<=b	True
<>	不等于	a<>=b	True

3.4.3 逻辑运算符

逻辑运算符用来对布尔型的量进行运算，结果是 True 或 False。PowerBuilder 中有 3 个逻辑运算符，其表示和功能见表 3-4。

表 3-4 逻辑运算符

运 算 符	含 义	示 例
NOT	逻辑反	If NOT a=10 Then ...
AND	逻辑与	If a>10 AND a<100 Then ...
OR	逻辑或	If a>10 OR a<100 Then ...

表 3-5 为逻辑运算的真值表，它表示当 a 和 b 为不同的取值组合时各种逻辑运算的结果值。

表 3-5 真值表

a	b	NOT a	NOT b	a AND b	a OR b
True	True	False	False	True	True
True	False	False	True	False	True
False	True	True	False	False	True
False	False	True	True	False	False

3.4.4 连接运算符

连接运算符只有一个，就是符号“+”，用于把两个 String 类型或 Blob 类型的变量内容连接在一起，形成新的字符串或 Blob 型数据。

例如：S1=“computer”+“book”，则 S1=“computerbook”

S2=“book”+“computer”，则 S2=“bookcomputer”

3.4.5 运算符的优先级

在表达式中，运算符按照优先级顺序进行运算，共分 9 级，见表 3-6。其中括号()优先

级最高，同级运算自左到右。

表 3-6 运算符的优先级

优先级	运算符	名称
1	()	括号
2	+, -, ++, --	正号, 负号, 自增, 自减
3	^	幂运算
4	*, /	乘, 除
5	+, -	加, 减及连接运算
6	=, >, <, >=, <=, <>	关系运算符
7	Not	逻辑非
8	And	逻辑与
9	Or	逻辑或

3.5 PowerScript 语句

PowerBuilder 语句主要用于控制程序的流程，主要有赋值语句、条件语句、循环语句等。

3.5.1 赋值语句

赋值语句用等号“=”把数据值或表达式的结果赋给一个变量或某对象的属性或成员变量，这是应用程序中使用最频繁的语句，其语法格式为：

```
variable_name=expression
```

其中 `variable_name` 表示变量名，`expression` 代表表达式，也可以是字符串、数字、变量、常量以及数组等，它的作用是把表达式的值赋给等号左边的变量。例如：

```
area=3.14*a*r  
ls_str="hello"  
li_num=10  
lr_resu=A^2+12
```

在赋值语句中，若等号右边是表达式，则先将其结果转化为等号左边变量的类型后，再赋值给变量。

提示：因为等号“=”在表达式中可以作为关系运算符，所以在赋值语句中不能实现连续赋值。例如：

```
li_a=li_b=10
```

上面的语句不能使变量 `li_a` 和 `li_b` 的值都为 10。因为第二个等号“=”被看做关系运算符处理，即第一个等号右边的部分作为一个表达式，`li_b=10` 的值为 `True` 或 `False`，然后把 `True` 或 `False` 赋给变量 `li_a`。如果 `li_a` 不能接受布尔型变量，那么在编译时就会出错。

3.5.2 条件控制语句

PowerScript 的条件语句有两大类：IF 语句和 CHOOSE CASE 语句。IF 语句是根据一个逻辑表达式的值决定下一步执行的语句块；CHOOSE CASE 语句是一个条件多分支结构，可以处理多种情况。

1. 条件语句 IF...THEN

条件语句分为单行和多行两种形式。

(1) 形式一：单行 IF...THEN 语句。

语法格式如下：

```
IF condition THEN action1 {else action2}
```

其中，condition 用于判断的条件，action1 是在判断条件为真的情况下，需要执行的脚本；action2 是在判断的条件为假的情况下，需要执行的脚本，其内容只能是单行语句。例如：IF x+1=y THEN x=0 ELSE x=-1。

(2) 形式二：多行 IF...THEN 语句。

语法格式如下：

```
IF condition1 THEN
    action1
[ELSEIF condition2 THEN
    action2...]
[ELSE
    action3]
END IF
```

其中参数如下。

condition1：要检测的第一个条件。

action1：当条件 condition1 为 True 时要执行的一条或几条语句。

condition2：当 condition1 为 False 时要检测的条件，在一个 IF...THEN 控制结构中可以有多个 ELSEIF...THEN 语句。

action2：当条件 condition2 为 True 时，要执行的一条或几条语句。

action3(可选的)：当前面的所有条件均不为 False 时要执行的一条或几条语句。这种多行格式的条件语句有两种常见的格式，分别为：

```
IF condition1 THEN
    action1
END IF
```

和

```
IF condition1 THEN
    action1
ELSE
    action2
END IF
```

例如：

```
IF score>=90 THEN
    Grade="A"
ELSEIF score>=80 AND score<90 THEN
    Grade="B"
ELSEIF score>=70 AND score<80 THEN
    Grade="C"
ELSEIF score>=60 AND score<70 THEN
    Grade="D"
ELSE
    Grade="E"
END IF
```

上面的例子将学生成绩的百分制转换成等级制。

2. 条件分支语句 CHOOSE…CASE

条件分支语句通过测试表达式或变量的值来控制程序进行的方向，适用于多个条件的分支结构。

条件分支语句的格式为：

```
CHOOSE CASE testexpression
    CASE expressionlist1
        statements1
    [CASE expressionlist2
        statementblock2
    ...
    CASE expressionlistn
        statementsn]
    [CASE ELSE
        statementsn+1]
END CHOOSE
```

其中，expressionlist 可以是以下形式之一：

- (1) 单个值。
- (2) 用逗号分隔的若干个值，例如：2, 3, 5。
- (3) 用 TO 表示某一区间，如 1 to 10, ‘b’ to ‘h’。
- (4) 用 IS 代表测试值，与关系运算符一起构成关系表达式，例如：
is>20 //is 是保留字，代表 testexpression 的值。
- (5) 混合：2, 4, 6, 7 to 10, is>30。

执行条件分支语句时，PowerBuilder 将逐条查找 CASE，如果找到与测试值匹配的判断表达式，就执行该 CASE 后的语句块，然后执行 END CHOOSE 后的第一条语句。如果条件分支语句包含了 CASE ELSE 子句，则未找到任何匹配的 CASE 条件时，执行 CASE ELSE 子句中的语句块。

例如：用 CHOOSE CASE 语句实现学生成绩百分制到等级制的转换。

```
CHOOSE CASE score
    CASE IS >=90   grade= "A"
    CASE 80 TO 89 grade= "B"
```

```

CASE 70 TO 79 grade= "C"
CASE 60 TO 69 grade= "D"
CASE ELSE      grade= "E"
END CHOOSE

```

3.5.3 循环控制语句

循环控制语句可以实现任意次地重复执行一行或多行语句，被重复执行的语句叫循环体。循环体的执行次数可以是任意次，但不能是无限次，即循环必须有结束的时候，不能使程序陷入死循环中。

PowerScript 中的循环语句有两种形式：FOR…NEXT 语句、DO…LOOP 语句。

1. FOR 循环语句

FOR…NEXT 语句的功能是按照预先规定的次数重复执行一段代码。其格式为：

```

FOR varname = start TO end [step increment]
    Statementblock
NEXT

```

其中含义如下。

- varname 是循环控制变量名，可以是任意的数据类型，最常用的是整数类型。
- start 为循环变量的初始值。
- end 为循环变量的终止值。
- increment(可选)是每次循环变量的增量(默认为 1)。当 increment 为正数时，初值 start 要小于终值 end；同样当 increment 为负数时，初值 start 要大于终值 end，这样才有意义。
- statementblock 是一组要重复循环的语句，称作循环体。

如图 3.5 所示为 FOR…NEXT 语句的执行过程。

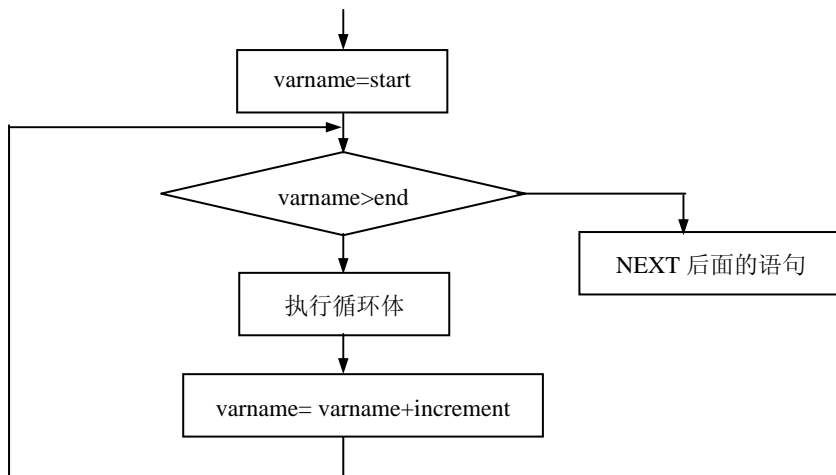


图 3.5 FOR…NEXT 语句的执行过程

下面列举 FOR…NEXT 循环的几种使用方法。

【例 3.1】 重复执行 A=A+10，直到和循环变量 n 的值大于 25，每次循环 n 的值增加 1。


```
FOR n=5 TO 25  
  A=A+10  
NEXT
```

【例 3.2】求 $1+2+\cdots+100$ 的和。

```
FOR n=1 TO 100  
  Sum =sum+n  
NEXT
```

2. DO...LOOP 语句

DO...LOOP 语句控制重复执行一段代码，直到条件表达式的值为 True 或 False。它有以下四种形式。

格式一：

```
DO WHILE condition  
  statementblock  
LOOP
```

说明：当 condition 的值为 True 时，执行循环体 statementblock；condition 的值为 False 时，退出循环体。其功能和执行过程如图 3.6 所示。

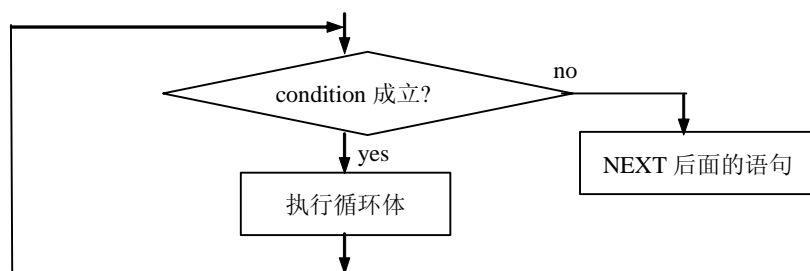


图 3.6 DO WHILE...LOOP 语句的执行过程

格式二：

```
DO UNTIL condition  
  statementblock  
LOOP
```

说明：当 condition 的值为 False 时，执行循环体；直到其值为 True 时退出循环体。其功能和执行过程如图 3.7 所示。

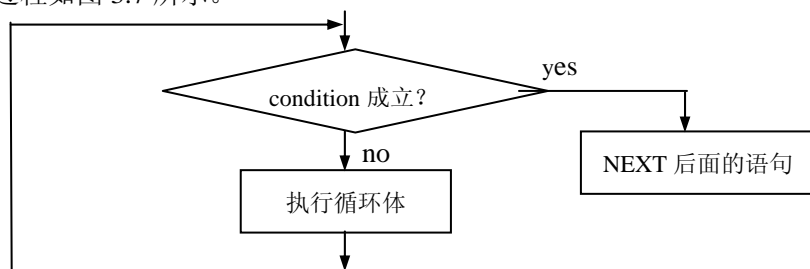


图 3.7 DO UNTIL...LOOP 语句的执行过程

格式三：

```
DO
    statementblock
LOOP WHILE condition
```

说明：先执行循环体 `statementblock`，再判断 `condition` 的值。如果其值为 `True`，则执行循环体 `statementblock`，当 `condition` 的值为 `False` 时退出循环体。其功能和执行过程如图 3.8 所示。

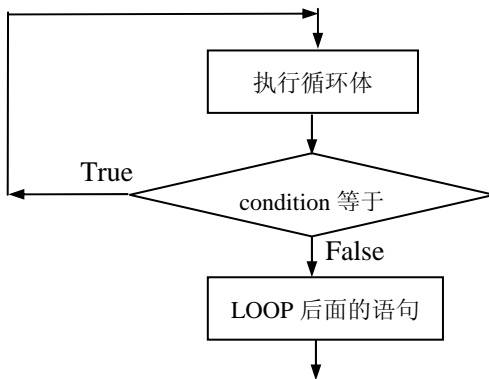


图 3.8 DO LOOP WHILE 语句的执行过程

格式四：

```
DO
    statementblock
LOOP UNTIL condition
```

说明：先执行循环体 `statementblock`，再判断 `condition` 的值。如果其值为 `False`，则重复执行循环体 `statementblock`，直到 `condition` 的值为 `True` 时退出循环体。其功能和执行过程如图 3.9 所示。

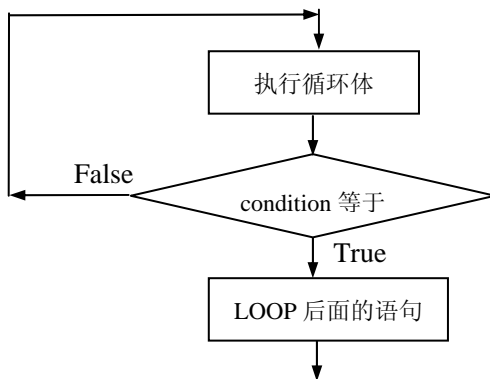


图 3.9 DO...LOOP UNTIL 语句的执行过程

下面列举 `DO...LOOP` 的几种不同使用方法。

例如：分别用 `FOR...NEXT` 和 `DO...LOOP` 语句求 $s=1+3+5+7+\dots+99$ ，注意其中各种语句的差别，特别是循环条件。

1) 用 FOR…NEXT 循环语句

```
Integer s,i
s=0
For i=1 to 99 step 2
    s=s+i
Next
st_1.text="s="+string(s) //将运算结果显示在静态文本框 st_1 中，因为 s 是整数，
                          //所以要用 string() 函数将它转换为字符型。
```

2) 用 DO…LOOP 语句

(1) 用 DO WHILE…LOOP 语句。

```
integer s,i
s=0
i=1
Do while i<=99
    s=s+i
    i=i+2
Loop
st_1.text="s="+string(s)
```

(2) 用 DO UNTIL…LOOP 语句。

```
integer s,i
s=0
i=1
Do until i>99
    s=s+i
    i=i+2
Loop
st_1.text="s="+string(s)
```

(3) 用 DO LOOP…WHILE 语句。

```
integer s,i
s=0
i=1
Do
    s=s+i
    i=i+2
Loop while i<=99
st_1.text="s="+string(s)
```

(4) 用 DO LOOP…UNTIL 语句。

```
integer s,i
s=0
i=1
Do
    s=s+i
    i=i+2
Loop until s>99
st_1.text="s="+string(s)
```

3.5.4 GOTO 语句

跳转控制语句，在程序中直接从一条语句跳转到另一条有标号的语句执行。其语句格式为：

```
GOTO Label
```

其中,Label 是任一合法的标识符,用来标识 GOTO 语句将要跳转到的语句。标号 Label 在标识语句时的格式是：

```
Label:
```

标号 Label 可以单独占一行，也可以与所标识的语句同处一行。

需要说明的是，由于 GOTO 语句不符合结构化程序设计的思想，所以不提倡使用。

3.5.5 HALT 语句

格式：HALT 或 HALT CLOSE

HALT 结束当前的 PowerBuilder 应用程序。

HALT CLOSE 先触发当前 PowerBuilder 应用程序的 application 对象的 Close 事件，然后再结束当前的应用程序。

3.5.6 RETURN 语句

利用 RETURN 语句，可以立即终止脚本或函数的执行。在事件脚本执行过程中，遇到 RETURN 语句，系统将终止事件脚本的执行，等待下一个事件；函数中使用 RETURN 语句，系统将返回调用该函数的语句的下一条语句。其格式有两种：

格式一：RETURN

格式二：RETURN expression

其中 expression 代表函数的返回值，可以是数据值或表达式，其数据类型必须与定义的返回值的数据类型一致。

例如：下面的函数将返回 arg1 被 arg2 整除后的结果，如果 arg2 等于 0，则返回-1。

```
If arg2<>0 then  
    Return arg1/arg2  
Else  
    Return -1  
End If
```

3.6 嵌入式 SQL 语句

在 PowerBuilder 开发的应用程序中，并非所有情况与数据库的交互操作都是通过数据窗口完成的，有时需要在程序中直接使用 SQL 语句操作数据库。例如，读取一条记录。针对这类应用需求，PowerScript 提供了一整套嵌入式 SQL 语句。利用嵌入式 SQL 语句，能

够在程序中灵活地操纵数据库。

PowerScript 支持在程序中使用嵌入式 SQL 语句，并且支持在 SQL 语句中使用具体数据库管理系统 DBMS 特有的 SQL 语句、函数和保留字。实际上对这类语句，PowerBuilder 在将其发送到 DBMS 之前并不进行任何处理，而是由 DBMS 完成相应操作，最后 PowerBuilder 得到处理结果。

SQL 语句在书写时必须以分号 (;) 作为语句的结束符，一条 SQL 语句可在一行书写，也可以分为几行书写，当在多行书写时不需要续行符号。在 SQL 语句中可以使用常量或合法的变量，但使用变量时必须在变量前加个冒号，通常称为绑定变量或引用变量，多个变量或常量之间用逗号分隔。下面介绍几个常用的嵌入式 SQL 语句。

3.6.1 连接语句

在 PowerBuilder 应用程序中，如果要连接和操作数据库，必须首先建立事务对象。事务对象是实现应用程序和数据库通信的对象。事务对象的属性可以分为两大部分，一部分是连接数据库的参数，另一部分是接收返回的数据库操作状态信息。每个事务对象变量有 15 个属性，其中 10 个属性用于连接数据库，5 个属性用于接收数据库返回的操作状态信息。

在 PowerBuilder 应用程序中，使用 SQL 语句操作数据库包括以下步骤。

- (1) 为连接数据库的事务对象的属性赋值。
- (2) 连接数据库。
- (3) 执行数据库操作。
- (4) 断开与数据库的连接。

PowerBuilder 提供了一个默认的全局事务对象 SQL (SQL Communications Area)，开发人员可以使用这个默认的事务对象建立数据库的连接。如果应用程序需要同时操作多个数据库，则必须建立另外的数据库事务对象变量。

不管是使用数据窗口操作数据库，还是使用 SQL 语句操作数据库，都必须连接数据库。连接数据库的语句是 CONNECT，断开连接的语句是 DISCONNECT。

1. CONNECT 语句

CONNECT 语句的功能是根据指定的事务对象完成对数据库的连接，其语法格式为：

```
CONNECT {USING transactionobject};
```

其中，参数 transactionobject 是用来连接数据库的事务对象的名称，默认的事务对象名是 SQLCA。

例如：使用默认的事务对象 SQLCA 连接到数据库。

```
CONNECT USING SQLCA;  
IF SQLCA.SQLcode<>0 THEN  
    MessageBox("连接数据库", "连接数据库失败!" + SQLCA.SQLErrtext)  
ELSE  
    MessageBox("连接数据库", "成功连接数据库")  
END IF
```

一般来说，在应用程序中使用 SQL 语句操作数据库之后，应该使用事务对象的 SQLcode

属性来判断数据库操作是否成功，如上面的程序所示。

2. DISCONNECT 语句

DISCONNECT 语句的功能：首先对指定的事务对象执行 COMMIT 语句提交事务，然后断开与指定数据库的连接，该语句的格式是：

```
DISCONNECT {USING transactionobject};
```

其中参数 transactionobject 用来指定要断开数据库连接的事务对象的名称，默认的事务对象是 SQLCA。

3.6.2 事务处理语句

事务是数据库管理系统完成一项完整工作的逻辑单位，数据库管理系统保证一个事务要么被完整地做完(称做提交)，要么被彻底地取消(称做回滚)。在应用程序中可以通过 COMMIT 语句和 ROLLBACK 语句控制事务操作。注意，事务和事务对象是两个完全不同的概念。

1. 提交事务语句 COMMIT

COMMIT 语句的功能是提交事务，完成数据库的物理修改。执行该语句后，将关闭所有先前打开的游标(CURSOR)和过程(PROCEDURE)，并开始下一个新的事务。COMMIT 语句的语法格式如下：

```
COMMIT [USING TransactionObject];
```

其中，TransactionObject 是事务对象名，默认时使用事务对象 SQLCA。

2. 回滚事务语句 ROLLBACK

ROLLBACK(回滚)语句的功能是：放弃自上一个 COMMIT、ROLLBACK 或 CONNECT 语句以来的所有数据库操作，关闭所有的游标和过程，并开始一个新的事务。其语法格式如下：

```
ROLLBACK [USING TransactionObject];
```

其中，TransactionObject 是事务对象名，默认时使用事务对象 SQLCA。

3.6.3 数据库操作语句

1. 单行检索语句 SELECT

一般的 SQL SELECT 语句的查询结果都是一组满足条件的记录，但是这样的语句嵌入到程序中使用时，需要满足一些特殊的约束和要求。这里介绍的 SQL SELECT 语句则只能返回一条记录，如果要处理返回多条记录的情况，则应该使用 Cursor(游标)技术。

SELECT 语句的格式是：

```
SELECT col1,col2, ...coln  
INTO :var1,:var2, ..., :varn  
FROM table_name  
WHERE condition_expression
```

```
[USING transaction_object];
```

其中参数：

- col1、col2 等均为列名。
- table_name 为表名。
- condition_expression 为条件表达式。
- var1、var2 等均为 PowerScript 中定义的变量。
- transaction_object 表示当前连接数据库的事务对象，默认值 SQLCA。

方括号[]表示可选部分。

功能：从数据库中检索一条满足条件的记录，并将结果存放到变量 var1、var2...varn 中。

例如：从表 student 中查找 name 为“黎明”的记录，如果找到，则将该记录中 id、sex、birthday 字段的值存入变量 s1、s2、date1 中。编写代码如下。

```
String s1,s2,x1
Date date1
x1= '黎明'
Select id,sex,birthday
Into :s1,:s2,:date1
From student
Where name = :x1
Using sqlca;
```

在上面的程序中，SQL SELECT 语句只能从数据库中读取一条记录，若要处理多条记录，则必须使用游标。

提示：PowerScript 的变量用在 SQL 语句中时必须冠以冒号前缀，以此来表示这个标识符是 PowerScript 的变量，而不是数据库的字段名。

2. 插入语句 INSERT

INSERT 语句向数据库中插入一条记录，该语句的格式是：

```
INSERT INTO table_name [(col1,col2,...coln)]
VALUES (v1,v2,...,vn)
[USING transaction_object];
```

其中：

- 方括号表示可选部分。
- col1, col2 均为列名。
- table_name 为表名。
- v1, v2 等为 PowerScript 表达式。
- transaction_object 表示当前连接数据库的事务对象，默认值为 SQLCA。

功能：向表 table_name 中插入一条记录，各列的值依次分别为 v1, v2 等，若某列的列名未给出，则值为 NULL。

提示：如果所有的列名都未给出，则必须在 VALUES 中依次给出所有的列值；给出的列值

的类型必须与对应的列的数据类型一致。

例如：向表 student 中插入一条记录，其字段 stu_id, stu_name 的值为程序变量 s_id、s_name 所对应的值。编写代码如下。

```
string s_id,s_name
s_id= '99030001'
s_name= '李利'
insert student (stu_id,stu_name,sex) values (:s_id,:s_name. '男');
```

3. 删除语句 DELETE

若要从数据库的表中删除一条或若干条记录，则使用 DELETE 语句。其语法格式为：

```
DELETE FROM table_name
    [WHERE condition_expression]
    [USING transaction_object];
```

其中各个参数的含义同以上 SQL 语句，这里就不再赘述了。

DELETE 语句的功能是从表 table_name 中删除满足条件 condition_expression 的记录。

例如：

```
string s_id
delete from student
where stu_id=:s_id;
```

提示：如果 DELETE 语句中不给出条件，则会删除表中所有的记录。

4. 修改语句 UPDATE

可以使用 UPDATE 语句来更新指定表中的数据。其语法格式为：

```
UPDATE table_name SET col1=v1[,col2=v2, ...,coln=vn]
    [WHERE condition_expression]
    [USING transaction_object];
```

其中：

- 方括号为可选部分。
- col1, col2 等均为列名。
- table_name 为表名。
- condition_expression 为条件表达式。
- v1,v2 等为 PowerScript 表达式。

UPDATE 语句的功能是更新表 table_name 中满足条件的记录，使这些记录的列 col1 的值为 v1，列 col2 的值为 v2 等。需要特别说明的是，如果 UPDATE 语句中没有给出条件，则更新表中所有的记录。

3.6.4 游标语句

在前面章节中我们介绍了如何使用 SQL SELECT 语句从数据库中查询记录。SELECT 语句只能返回一条满足条件的记录，若 SELECT 语句返回多条记录，则会出现错误。当需

要从数据库中查询多条记录时，需要使用游标(CURSOR)来处理。游标可以看作是由一个查询结果集组成的一个临时只读文件，在程序中可以从这个临时文件中读取一条记录给程序的变量进行处理。

使用游标的基本步骤如下。

- (1) 用 DECLARE 语句声明游标。
- (2) 用 OPEN 语句打开游标。
- (3) 使用 FETCH 语句提取数据。
- (4) 用 CLOSE 语句关闭游标。

1. DECLARE 语句

像使用其他类型的变量一样，使用一个游标之前，首先应当声明它。游标的声明包括两部分：游标名称和这个游标所用到的 SQL SELECT 语句。

声明游标的格式为：

```
DECLARE 游标名 CURSOR FOR
      Sqlstatement;
```

例如：声明一个名为 student 的游标，用以查询 student 表中家庭住址在重庆的学生的姓名、学号和性别，编写代码如下。

```
DECLARE student CURSOR FOR
      SELECT stu_id,stu_name,sex
      FROM student
      WHERE address LIKE '重庆';
```

如同其他的变量声明一样，声明游标的这段代码行是不执行的，不能将 debug 时的断点设在这一代码行上，也不能用 IF...END IF 语言声明两个同名的游标。如下列的代码就是错误的。

```
IF address='重庆' THEN
      DECLARE student CURSOR FOR
      SELECT stu_id,stu_name,sex
      FROM student
      WHERE address= '重庆';
ELSE
      DECLARE student CURSOR FOR
      SELECT stu_id,stu_name,sex
      FROM student
      WHERE address<>'重庆';
END IF
```

2. OPEN 语句

要从游标中读取数据，必须首先打开它，打开游标的格式为：

```
OPEN 游标名;
```

例如：打开上例中声明的游标 student，格式如下。

```
OPEN student;
```

由于打开游标是对数据库进行一些 SQL SELECT 的操作，它将耗费一段时间，这主要取决于所使用的系统的性能和这条语句的复杂度。如果时间过长，可以考虑将屏幕上显示的鼠标形状改为 hourglass。

3. FETCH 语句

当用 OPEN 语句打开游标，并在数据库中执行了查询后，我们还不能利用查询的结果集中的数据，必须使用 FETCH 语句来取得数据。一条 FETCH 语句一次可以将一条记录放入指定的变量中，事实上，FETCH 语句是游标使用的核心。

在后面的章节中，我们将学习 DataWindow 技术，即执行 DataWindow 的 Retrieve() 函数后，所有的结果集都可以得到；而使用游标，只能逐条记录地得到查询结果。

提取数据的格式为：

```
FETCH 游标名 INTO :变量 1, :变量 2, ...变量 n;
```

注意：变量前的冒号(:)不能省略。

已经声明并打开了一个游标后，就可以用 FETCH 语句将数据放入程序变量中。

例如：

```
FETCH student INTO :s_id,:s_name,:s_sex;
```

游标一次只能从后台数据库中读取一条记录，而在多数情况下，我们需要处理全部记录。所以在程序中一般将 FETCH 语句放在一个循环体内，以便将结果集中的全部记录都提取处理。

在程序中使用循环语句处理结果集中的记录，一般通过检测事务对象的 SQLCODE 的值来判断是否从结果集中提取完数据。当 SQLCODE 的值为 0 时，表示正常提取数据；若 SQLCODE 的值为 100，表示没有找到，即结果集中的数据已经提取完；而其他值均表示数据库操作出现错误。

例如：

```
fetch student into :s_id,:s_name,:s_sex;
do while sqlca.sqlcode=0
    lb_1.additem(s_id+ "---" +s_name+ "---"+s_sex)
    //对读取的记录进行处理
    ...
    Fetch student into:s_id,s_name,s_sex;
Loop
```

4. CLOSE 语句

在游标操作的最后不要忘记关闭游标，这是一个好的编程习惯，以使系统释放游标所占用的资源。

关闭游标的格式为：

```
CLOSE 游标名;
```

例如：CLOSE student;

3.6.5 Blob 列操作语句

在选择数据源时，不能把 Blob 类型的数据列为数据源的一部分来选择，一般的 SELECT 语句不能处理这类型数据，但 PowerScript 嵌入式 SQL 语句中 SELECT BLOB 语句可以处理。下面介绍两个有关 Blob 列操作的语句。

1. SELECTBLOB 语句

SELECTBLOB 语句用来检索 Blob 型数据。当找到多条符合查询条件的数据时，事务对象状态属性将指明错误。其语法格式为：

```
SELECTBLOB BlobFieldOfTableList  
INTO VariableList  
FROM TableName  
WHERE Criteria  
[USING TransactionObject];
```

其中，BlobFieldOfTableList 代表 Blob 型字段名列表，VariableList 代表接受数据的 Blob 变量名列表，TableName 代表被检索的表名，Criteria 是检索条件，TransactionObject 代表在使用非默认事务对象时的事务对象名。

2. UPDATEBLOB 语句

UPDATEBLOB 语句用来修改 Blob 型数据。其语法格式为：

```
UPDATEBLOB TableName  
SET BlobColumn=BlobVariable  
WHERE Criteria  
[USING TransactionObject];
```

其中，TableName 是需修改的 Blob 数据所在表的表名，BlobColumn 指 Blob 数据所在的列名，BlobVariable 代表包含 Blob 数据的变量。

3.6.6 存储过程语句

存储过程(Stored Procedure)是在数据库中预先定义的且已经被编译好的事务。由于存储过程和数据同处于服务器端，使得客户端只需向服务器传送存储过程名以及少量的参数(若有参数的话)而不是传送构成存储过程的所有 SQL 语句，便可获得数据库服务器的响应。这种方式加快了系统响应速度，大大提高了客户端同服务器端的交互效率。

并不是所有的数据库系统都支持存储过程。若所用数据库系统支持存储过程，则用户可在 PowerScript 中声明并使用那些定义在数据库中的存储过程。存储过程的使用方法同游标十分类似，下面仅作简要说明。

使用存储过程的基本步骤包括：声明存储过程、执行存储过程、读取数据、关闭存储过程。在这 4 个步骤中，分别要用到存储过程的 DECLARE 语句、EXECUTE 语句、FETCH 语句以及 CLOSE 语句。

1. DECLARE 语句

使用存储过程前需用 DECLARE 语句进行声明，其语法格式为：

```
DECLARE ProcedureName PROCEDURE FOR StoredProcedureName  
@Param1=Value1, @Param2=Value2, ...  
{USING TransactionObject};
```

其中：ProcedureName 是为调用存储过程而声明的过程名，可以为任意合法的 PowerScript 标识符；StoredProcedureName 为数据库中的存储过程名；ParamN 是数据库存储过程中所定义的第 N 个参数，参数前面必须加 @ 号；ValueN 是对相应参数的赋值，可为 PowerScript 主变量；TransactionObject 代表所使用的事务对象，其默认值为事务对象 SQLCA。

2. EXECUTE 语句

EXECUTE 语法用来执行所声明的过程，其实质即执行相应的数据库存储过程。此语句的语法格式为：

```
EXECUTE ProcedureName;
```

其中，ProcedureName 为所要执行的过程，该过程必须已被声明。

3. FETCH 语句

FETCH 语句用来从存储过程的执行结果集中读取一条记录，使用方法类似于游标。其语法格式为：

```
FETCH ProcedureName INTO HostvariableList;
```

其中，ProcedureName 为已执行的过程名，HostvariableList 是接收存储过程执行结果的 PowerScript 主变量列表。

4. CLOSE 语句

CLOSE 语句用来关闭先前声明的过程。执行此语句之后，就不能再使用 FETCH 语句读取数据了。其语法格式为：

```
CLOSE ProcedureName;
```

其中，ProcedureName 为所要关闭的过程名。

由于存储过程的使用方法同游标十分类似，在使用的过程中需要注意的问题请直接参考前面对游标的有关叙述，此处不在赘述。

3.6.7 SQLCode

嵌入式 SQL 语句的执行有可能成功，也有可能失败，良好的编程风格是对每条可执行的 SQL 语句进行其执行情况的检查。

每当执行一条语句后，与该语句相关的事务对象的 SQLCode 属性都给出一个值指示 SQL 语句的执行是否成功。SQLCode 取值为：

0——最近一次 SQL 语句执行成功。

-1——最近一次 SQL 语句执行失败。

100——最近一次 SQL 语句没有返回数据。

当 SQLCode 的值为-1 时(即最近一次 SQL 语句执行失败时)，事务对象的 SQLCode 属

性中存放着数据库厂商提供的错误代码,事务对象的 `SQLErrText` 属性中存放着数据库厂商提供的错误信息,利用这两个属性,可以得到出错原因。

下面是一个检查 SQL 语句的执行是否成功的示例。

```
int Emp_num
string Emp_Lname, Emp_Fname
Emp_num=Integer (sle_Emp_Num.Text)
SELECT employee.Emp_Lname,employee.Emp_Fname
    INTO :sle_Lname.text, :sle_Fname.text
    FROM Employee
    WHERE Employee.Emp_nbr=:Emp_num
    USING Emp_tran;
If Emp_tran.SQLCode=100 then          //未找到满足条件的数据
    MessageBox(“查询雇员”,“未找到指定雇员”)
Elseif Emp_tran.SQLCode<0 then        //SELECT 语句执行不成功
    MessageBox(“数据库错误”,Emp_tran.SQLErrText,Exclamation!)
                                     //出错信息
End If
```

3.6.8 编程举例

用游标的方法将数据库 xscj 中的表 jiben 的记录显示在窗口上的 ListBox 中。

创建一个 xscj.pbw 工作区,其 application 为“app_xscj”;再新建一个 w_jiben 窗口,在该窗口上分别创建一个 ListBox 名为“lb_1”,一个单行编辑框名为“sle_1”,一个 CommandButton 名为“cb_1”。

在 app_xscj 的 Open 事件中输入代码:

```
SQLCA.AutoCommit=True
Sqlca.DBMS = "odbc"
Sqlca.database = ""
Sqlca.servername = ""
Sqlca.dbparm = "Connectstring= 'DNS=xscj;UID=dba;PWD=sql;'"
Sqlca.logid= ""
Sqlca.logpass= ""
Sqlca.dbpass= "sql"
Sqlca.userid= "dba"
connect;
if sqlca.sqlcode<> 0 then
    messagebox("==== 错误信息提示====",&
        "不能连接数据库!~r~n~r~n 请询问系统管理员",stopsign!)
    return
end if
open(w_jiben)
```

在 cb_1 的 Clicked 事件中输入以下的代码:

```
string s_id,s_name,s_sex,s_addr
//声明游标 student
DECLARE student CURSOR FOR
    SELECT xm,xh,xh
```

```

FROM jiben
Where jtzz=:s_addr;
S_addr=trim(sle_1.text)           //要查询的家庭地址
Lb_1.reset()                     //清除 ListBox 中内容
Open student;                   //打开游标
fetch student into :s_name; :s_id, :s_sex; //提取数据
do while sqlca.sqlcode =0       //将读取的数据加到 ListBox 中
    lb_1.additem(s_id+ "---"+s_name+ "--"+s_sex)
    fetch student into :s_name, :s_id, :s_sex;
loop
close student;                  //关闭游标

```

3.7 数 组

数组是具有相同数据类型变量的集合，这些变量共用一个变量名即数组名，其数据访问则通过数组的下标来进行。PowerScript 中的数组可以是一维的也可以是多维的，可以是静态数组也可以是动态数组。

3.7.1 定义数组

数组的定义实际上是变量定义的一种特殊形式。定义数组的语法格式如下：

```
datatype variablename [ {d1,...,dn} ] { = {valuelist} }
```

下面对上述的定义格式作以下几点说明。

(1) 数据类型 **datatype** 可以是标准数据类型、系统对象数据类型或者已被定义过的结构类型。

(2) 数组名 **variablename** 必须是合法的用户标识符，用户可以一次同时定义多个数组，甚至可以将具有相同 **datatype** 的数组和一般变量的定义一块儿进行。

(3) 方括号 **[{d1,...,dn}]** 中的 **{d1,...,dn}** 用来对数组的维数和规模进行说明。当 **n=1** 时，方括号内可为空，此时表示定义一个动态数组，系统将根据程序运行时的实际需要动态调整数组元素的个数。当 **n>1** 时，则 **di** (**i=1~n**, **di≥1**, 且 **di∈Z**) 的值均必须指出，此时不能定义动态数组。

(4) 可选项 **{valuelist}** 用于对数组元素值的初始化，所赋值按顺序表示在大括号之中并用逗号分开，值的数据类型必须与所声明的类型相匹配。用户可以只给数组的前几个元素赋初值；当所赋初值个数超过数组规模时，系统将忽略多余赋值。

(5) 对于数组可定义维数的多少以及数组规模的大小，一般仅受系统可用内存大小的限制。一维数组的大小可以是固定的也可以是可变的，而多维数值的大小在定义后都是固定的。

以下是几个相关的示例。

```

Blob bl_pictures[ ]           //定义一个 Blob 类型的动态数组
Int Myarray[10],Myarray2[20] //定义两个 integer 类型的数组，大小固定

```

```
String S_Messagetext [-1 to 1]      //定义一个包含3个字符串的数组，下标从
                                   // -1 到 1
Uint ui_staff[ 10,0 to 20,-5 to 5 ] //定义一个三维数组，并自定义第2维和第3
                                   // 维的编址范围
//以下定义一个由3个元素组成的数组，并对其初始化
Real r_Rate[3] = { 0.10,0.20,0.30 }
//以下定义一个元素个数为3×4的二维数组，并对其初始化
Integer MyGrid[ 3,4 ] = { 1,2,3,1,2,3,1,2,3,1,2,3 }
```

提示：初始化多维数组时，首先是对第1维初始化，然后是第2维、第3维……依此类推。用户可以只初始化数组的部分元素，但必须是从第1个数组元素开始连续进行，不可以从中间开始，也不可以跳跃式赋值。未被指定初始化值的剩余数组元素将由系统赋予初始值，此默认初始值同简单变量是一致的。当大括号内所赋初值个数超过数组规模时，系统将自动忽略多余赋值。

3.7.2 使用数组

数组必须先定义，然后才能使用。PowerScript 语言规定只能逐个引用数组元素而不能一次引用整个元素。

1. 一维数组的引用

一维数组元素的表示形式为：

数组名[下标]

例如：

```
integer li_date[4]={1,2,3,4}
integer x
x=li_date[3]+li_date[4]
```

2. 多维数组的引用

二维数组元素的表示形式为：

数组名[下标1,下标2]

例如：

```
integer li_arr[2,2]={11,12,21}
integer y
y=li_arr[1,2]/2
```

提示：在使用数组时，无论是大小确定的数组还是动态数组，都不要越界使用，否则会带来编译错误。

3.8 结 构

结构是一个或多个相关变量的集合，这些变量可以具有相同的数据类型，也可以具有不同的数据类型。在其他语言中，有的称为记录。结构里的所有元素既可以作为一个整体

引用，也可以分别引用其中的元素。

在 PowerBuilder 中，就像函数一样，结构也有如下两种类型。

(1) 全局结构：这种结构与具体对象无关，在程序中的任何地方都能使用。

(2) 对象层结构：这种结构总是与特定的窗口、菜单、用户对象等相关联，是对象的一部分，可以在该对象中使用，也可以在其他对象中使用(当该结构所在的对象被打开时)。

结构名一般以“s_”作前缀。

3.8.1 定义结构

1. 定义全局结构

在 PowerBuilder 开发环境的主窗口中，选择工具栏的【New】图标或选择【File】|【New】命令，将出现标题为【New】的窗口，如图 3.10 所示，选择【PB Object】标签页的【Structure】项，双击【Structure】项或单击【OK】按钮，进入全局结构的定义，如图 3.11 所示。

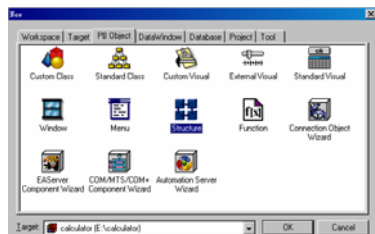


图 3.10 定义全局结构

在如图 3.11 所示的全局结构定义画板中，输入元素名，并选择元素的数据类型。按鼠标右键将出现弹出式菜单，如图 3.12 所示。选择【Insert Row】命令将插入元素，选择【Delete Row】命令将删除元素。

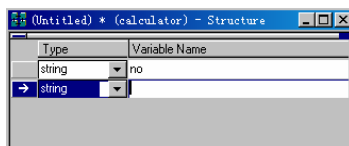


图 3.11 全局结构定义画板



图 3.12 增加、删除结构元素

定义好结构的元素后，选择保存，出现如图 3.13 所示的窗口，输入结构名。

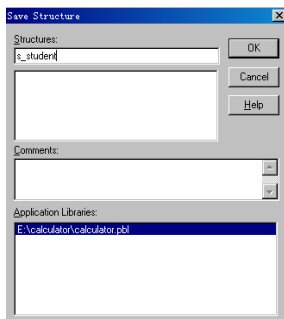


图 3.13 保存结构

2. 定义对象层结构

可以为 Application Object 对象、窗口对象、菜单对象、用户对象等创建结构，这种结构称为对象层结构。对象层结构一般只能在该对象内使用，当该对象正在打开时，其他对象的程序可以使用该结构。

首先打开要定义结构的对象，然后选择【Insert】|【Structure】菜单命令，如图 3.14 所示，将出现结构定义窗口，如图 3.15 所示。输入结构名、元素名及类型。右击将出现弹出式菜单，如图 3.12 所示，选择【Insert Row】命令将插入元素，选择【Delete Row】命令将删除元素。对象层的结构与所在的对象一起保存，不能单独保存。

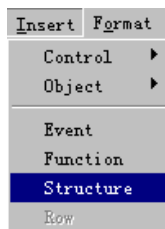


图 3.14 定义对象层结构

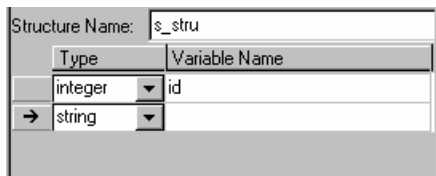


图 3.15 结构定义画板

3. 删除全局结构

要删除自定义全局结构，需用打开要删除的全局结构所在的系统树(System Tree)，选择要删除的全局结构，右击出现弹出式菜单，选择【Delete】命令将删除所选的全局结构，如图 3.16 所示。

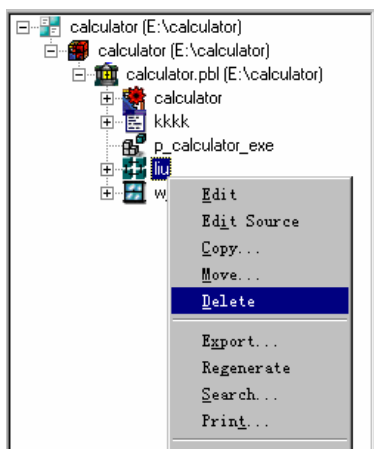


图 3.16 删除全局结构

4. 删除对象层结构

要删除对象层结构，首先打开结构所在的对象，然后选择【View】|【StructureList】命令，如图 3.17 所示。

单击【StructureList】选项后将列出对象的全部结构，然后右击要删除的结构，出现一个弹出式菜单，如图 3.18 所示，选择【Delete】命令将删除所选的结构。

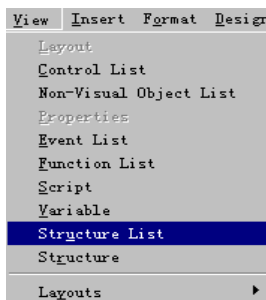


图 3.17 【View】菜单

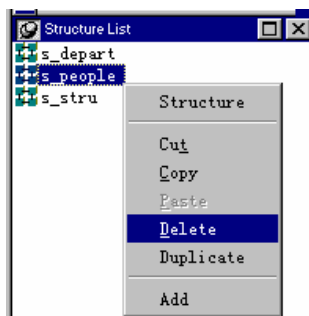


图 3.18 删除对象层结构

3.8.2 使用结构

定义了一个结构后相当于定义了一种新的数据类型。要使用某结构，应该先声明一个该结构类型的实例变量，然后再引用该结构的变量。

例：设 `s_student` 是定义的一个全局结构，则在程序中使用该结构类型的代码例子为：

```
s_student s1,s2           //定义两个 s_student 型的实例变量 s1 和 s2
s1.no= "320108800123204" //给结构 s1 的元素 no 赋值
s1.name=sle.text          //给结构 s1 的元素赋值 name
s1.math=96                //给结构 s1 的元素赋值 name
s2=s1                    //给结构 s1 的元素赋值给结构 s2 的对应元素
                        //只有同一类型的结构才能这样整体赋值
```

对象层的结构的引用方法与全局结构基本一样，但在其他对象的代码中引用时，需指明结构所在的对象(像引用对象函数那样)。

例如：`s_stru` 是在窗口 `w_main` 中定义的结构，在窗口中定义一个实例变量 `s3`。

```
s_stru s3                //在窗口的 Declare 中定义 Instance Variable
```

要在另一窗口 `w_sub` 中引用 `s3`，格式为：

```
sle_1.text=w_main.s3.id
sle_2.text=w_main.s3.name
```

3.9 面向对象编程

面向对象的设计是基于面向对象思想的一种设计方法。面向对象方法是基于对象的，它的核心思想就是在对象上包含了操作，即面向对象的方法是一体化的方法，它将数据与应用结合在一起。**PowerBuilder** 是面向对象的开发工具，它支持面向对象的程序设计技术和面向对象的概念。下面首先介绍面向对象的基本概念及其在 **PowerBuilder** 中的实现。

3.9.1 类和对象的概念

1. 类

在上面我们提到的是对象本身的一些集中的特性。但是，从编程的角度来说，如何来

定义和描述一个对象呢？这里需要使用到类和子类的概念。类是一组几乎相同的对象的描述，是由概括了一组对象共同性质的方法和数据(包括属性)组成的。从一组对象中抽象出公共方法和数据并将它们保存在一类中是面向对象功能的核心。对类的定义意味着将可用代码放在公共区中，而不是一遍又一遍地对之加以表示。换句话说，类是创建对象的蓝图、模板。当我们用类来创建一个对象的时候，实际上是生成了类的一个实例。根据实际情况把这个实例对应的数据赋予我们所需要的值时，便创建了一个我们需要的、在一定程度上具有自己的个性的对象了。

类可以具有子类，可以为一组子类概括公共元素。子类可以直接从父类中继承一些方法和属性。通过使用子类，面向对象的程序员就可以将应用程序描述成一组通用或抽象模块。

2. 对象

在面向对象编程中，对象是指包括数据和与之联系的方法的自包容模块。PowerBuilder 的大多数实体都是对象，包括窗口和控件之类的可视对象、事务对象和错误处理对象等非可视对象、自定义的可视对象和非可视对象。

一个对象的类是指一个对象的定义。用户可以在窗口画板、菜单画板、应用程序画板和用户对象画板中定义对象的类，也可以在窗口中添加控件，给对象属性赋初值，并为事件和函数编写脚本。

一个对象实例是指在应用程序运行期间对象的一次创建过程。用户的代码可以使一个对象实例化，并为该对象分配内存，同时按照类来定义实例。

对象的引用实际上是引用对象实例的句柄。我们通过把对象句柄赋给一个变量来同对象进行交互。

PowerBuilder 包括两种对象：一是系统对象，它们由 PowerBuilder 定义，并从基类 PowerObject 继承而来，而且它们是所有自定义对象的祖先。可以使用对象浏览器查看系统对象继承关系；二是在画板中自定义的对象，这些对象可以在窗口画板、菜单画板等不同画板中定义，它们都是从系统对象或另一个自定义对象继承而来的。

3.9.2 继承、封装、重载和多态性

1. 继承

继承是面向对象编程思想的一个重要的概念，它是自动共享各个类、子类中的方法和数据的有效机制。作为过程式系统所没有的一个有力机制，继承使得我们可以通过从父类中继承公共方法和数据，进而通过添加新的数据和方法定义成一个新的类。其后，我们又可以把新创建的子类作为父类，来创建它的新的子类。从上面的描述中可以看出，面向对象的编程中，类的创建是沿着树形的结构发展起来的。正是通过继承的机制，实现了我们编写的代码的重用，不仅节省了重复编码的时间，而且可以逐渐完善我们的类，直到创建出满足我们需要的各种类。

2. 封装

对对象最基本的理解是把数据和代码组合在同一个结构中，这就是对象的封装特性。

将对象的数据域封闭在对象的内部,使得外部程序必须而且只能使用正确的方法才能对要读写的数据域进行访问。封装性意味着数据和代码一起出现在同一结构中,如果需要的话,可以在数据周围砌上“围墙”,只有用对象类的方法才能在“围墙”上打开缺口。

3. 函数重载

函数重载是指同一个函数有几种调用格式。也就是说,一个相同的函数名字有多种参数格式和实现方法。与多态性不同,函数重载是指同一个对象中的同一名字的函数有多种实现方法。这很像 C 语言中提供的 `print()` 函数,可以使用多种格式调用 `print()` 函数,系统根据调用的参数决定使用哪一个 `print()` 函数。同样,在 PowerBuilder 中,一个对象也可以声明多个同名参数,在运行时,PowerBuilder 根据传送的参数个数和类型来决定调用哪一个。

每一个同名函数必须有不同的参数集。例如,创建一个窗口函数并保存它,然后再创建另一个相同名字的函数,但具有不同数目或类型的参数,保存它;在调用函数时,给出函数名和一组参数,相应的过程就会被运行。重载函数仅对对象级的函数有效。全局函数不能重载。

4. 多态性

多态性是在对象体系中把设想和实现分开的手段,这意味着几个不同的对象具有相同名字的方法(函数和或者事件),但每个对象为该方法提供的参数和实现过程都可能不同。

如果说继承性是系统的布局手段,多态性就是其功能实现的方法。多态性意味着某种概括的动作可以由特定的方式来实现,这取决于执行该动作的对象。多态性允许以类似的方式处理类体系中类似的对象。根据特定的任务,一个应用程序被分解成许多对象,多态性把高级处理的设想如新对象的创建、对象在屏幕上的重显、程序运行的其他抽象描述等,留给知道该如何完美地处理它们的对象去实现。

3.9.3 属性、事件和函数

1. 属性

属性用来描述对象的特征,描述一个对象不同于另一个对象的地方。一般来说,每个对象有多种属性,例如,“你的汽车”包括颜色、生产厂商、型号、价格等多种属性。所有汽车都有这些属性,但不同汽车的具体取值却不尽相同,“你的汽车”颜色是红的,“我的汽车”颜色可能是蓝的。这就是说,同一个类的两个对象虽然有相同的属性集合,但具体取值并非完全一样。PowerBuilder 的对象也都有属性,少到三至五个,多达数十种。对可视对象来说,查看对象具体属性的方法很简单:右击对象,从弹出菜单中选择【**Properties**】菜单项打开对话框,这样,对象属性就分门别类地呈现在屏幕上,你可以根据需要尽情修改。

对象的大多数属性在程序代码中可以动态地修改(不允许修改的属性称作只读属性),格式为:对象名.属性=属性值,例如在事件处理程序中写上代码:

```
Sb_submit.text="提交"
```

事件发生后,程序就把按钮 `sb_submit` 的标题修改为“提交”。其中, `sb_submit` 是按钮名称, `text` 是按钮对象的标题属性。

2. 事件

事件(Event)是指可能发生在对象上的事情。在 Windows 系统中,用户的所有操作都由系统自动转化为某个事件,比如,用户移动了鼠标,就会触发当前鼠标指针下对象的 MouseMove 事件;用户按下键盘上的某个按键,系统就会将这个操作变成键盘事件。另外,系统的某些行为也将换成事件来表现,比如定时器事件。在 PowerBuilder 中系统的运行是由事件驱动的,也就是说,系统运行是通过事件的触发来实现的。为了简化应用程序的开发,大多数 PowerBuilder 对象都预先定义了一组事件,称为系统事件,如窗口对象有 Open、Close、Clicked 等 29 个事件。如果我们想对某个事件作出反应,那么就需要编写该事件的事件处理程序,那么当该事件发生时,事件处理程序就会被执行。如果没有对某个事件编写事件处理程序,那么当该事件发生时,应用程序什么也不做,就像没有发生这样的事件一样。除了用户操作应用程序能够产生事件外,程序本身也能够主动地触发事件,其目的在于简化编程任务。

3. 函数

在大多数面向对象的编程语言中,将函数称作“方法”,这只是叫法的不同,本质并无差异。PowerBuilder 中的函数通常有以下类型。

(1) 系统(system)函数:系统提供的独立于其他对象的函数。

(2) 对象(object)函数:是对象定义的一部分,PowerBuilder 有许多预定义的对象函数,也可以定义自己的对象函数。

(3) 全局(global)函数:可以在任何程序中调用的函数,PowerBuilder 的系统函数都是全局的。

(4) 局部外部(local external)函数:属于某个对象的外部函数。

(5) 全局外部(global external)函数:在任何一个画板中都可以说明,可被所有的程序调用。

(6) 远程过程调用(remote procedure call, RPC)函数:可调用数据库中的存储过程。

(7) 用户自定义(user-defined)函数:可以在窗口画板或用户对象画板中自己定义的全局函数和对象函数。

函数一般封装在对象内部,其实现步骤和细节,用户既看不见,也不能修改,开发人员能做的工作就是按照约定直接使用它。大多数函数都有返回值,该值往往用来指示函数的执行情况。

3.9.4 实现面向对象的编程

PowerBuilder 的对象就是由 PowerBuilder 系统提供的、可以用来构造应用程序的一些“部件”,应用程序由一系列对象组成,包括窗口、菜单、函数、结构、数据窗口、通用对象等。PowerBuilder 为这些对象定制了属性和事件,开发人员可以给属性指定特定的值,可以为事件编写特定的程序。

使用 PowerBuilder 9.0 进行开发的实质是定义对象和控件及其属性,其次定义对象和控件上某个事件发生时要执行的程序。

一个 PowerBuilder 应用程序一般由若干个窗口组成,每个窗口上有若干控件(如命令按

钮、控制菜单、显示的文本等), 每个对象或控件都有若干事件(命令按钮上的单击或双击等), 每个事件将对应一段程序。

窗口是 PowerBuilder 应用程序中重要的组成部分, 是用户与 PowerBuilder 应用程序之间进行交互的主要界面。通过窗口界面, 用户可以操作应用程序、实现用户的目的、得到用户所需的数据。另外, 还可以在窗口中放置一些 PowerBuilder 控件, 通过这些控件接受和请求用户信息以及向用户提供信息。

用户在运行应用程序时, 主要通过放在窗口中的控件打交道, 来实现与应用程序的交互。控件也是 PowerBuilder 应用程序界面的重要组成部分, 是系统预先定义好的可视图形对象, 开发人员通过把这些对象添加到窗口上直接使用它们。控件并不单独保存, 而是与相应的窗口一起保存在应用程序库文件中。

3.10 函 数

在 PowerBuilder 中, 不仅可以直接使用系统所提供的各种函数, 还可以自己定义具有特定功能的用户函数。本节将介绍系统函数和用户自定义函数的功能及使用方法。

3.10.1 系统函数

大量的系统函数被提供给程序开发人员进行直接调用。对于初学者来说, 对这些系统函数作一个初步的概况性了解是很有必要的。本节将选择若干常用系统函数的功能及其使用方法。

系统函数是 PowerBuilder 中的全局通用函数, 这些函数不隶属于任何对象或控件, 在程序中的任意地方都可以不加任何说明地直接使用。熟练地使用这些系统函数, 不仅可以方便用户编程、加快应用程序的开发进度, 而且有益于保证程序代码质量、提高程序运行效率。

下面简要介绍一些常用的系统函数, 这些函数在应用程序的实际开发过程中经常使用到。限于篇幅, 其他系统函数的功能和用法在此不可能一一介绍, 请读者参考用户手册或者使用联机帮助。

1. 大二进制对象函数

● Blob()

函数功能: 将字符串转换成 Blob 类型数据。

函数语法: Blob (text)。

参数说明: text: String 类型变量或常量, 用以指定所要转换的数据。

返回结果: Blob 类型。函数执行成功则返回转换后的 Blob 类型数据; 若参数 text 参数的值为 NULL, 则函数返回值为 NULL。

使用说明: 与之相应, 利用 String(blob)函数可以将 Blob 类型的数据转换字符串类型的值。

2. 系统与环境函数

● Clipboard()

函数功能：提取或替换 Windows 系统剪贴板的文本内容。

函数语法：Clipboard ({string})。

参数说明：string: String 类型变量或常量，可选，用以指定所要复制到系统剪贴板上的文本。若剪贴板上已有内容，该文本将取代剪贴板上的当前内容。

返回结果：String 类型。函数执行成功时，若剪贴板上的内容为文本数据，则返回剪贴板的当前内容；若剪贴板上的内容为非文本数据(例如位图)或无任何数据，那么函数将返回空字符串；若 string 参数的值为 NULL，则函数返回值为 NULL。

使用说明：当指定 string 参数时，剪贴板上的原有内容将被 string 参数的值取代；当省略 string 参数时，将仅得到剪贴板的原有内容。无论是否指定 string 参数，Clipboard () 函数都返回剪贴板的当前内容。

● Run()

函数功能：运行一个指定的 Windows 或 UNIX 应用程序。

函数语法：Run(string{,windowstate})。

参数说明：String 类型，指定所要运行应用程序和名称，其中可以包括其路径以及相关的参数信息。

windowstate: WindowState 枚举类型，可选，指定程序运行时的窗口状态。其有效取值包括：Maximized! (最大化窗口)、Minimized!(最小化窗口)、Normal!(默认值)。

返回结果：Integer 类型。函数执行成功时返回 1，发生错误时返回-1。若有参数的值为 NULL，则函数返回 NULL。

使用说明：使用此函数，应用程序能够启动操作系统中的任何程序。当在函数的参数中指定了所要启动应用程序的有关参数时，这些参数的格式、个数、意义等由具体的应用程序确定。若函数的 string 参数中指定了文件名但没给出其扩展名，则默认该文件的扩展名为 .EXE；若所要运行应用程序的扩展名不是 .EXE(例如 .BAT、.COM、.CHM 等)，则必须在函数的参数中指出文件的扩展名。

3. 文件操作函数

● GetFileOpenName()

函数功能：显示打开文件对话框，让用户选择将要打开的文件。

函数语法：GetFileOpenName(title,pathname,filename{,extension{Filter}})。

参数说明：

title: String 类型，指定对话框的标题。

Pathname: String 类型变量，用于保存该对话框返回的文件路径及文件名。

Filename: String 类型变量，用于保存该对话框返回的文件名。

Extension: String 类型，可选，使用 1 到 3 个字符指定默认的扩展文件名。

Filter: String 类型，可选，其值为文件名掩码，指定显示在该对话框的列表框中供用户选择的文件名满足的条件(比如*.*, *.TXT, *.EXE 等)。

返回结果：Integer 类型。函数执行成功时返回 1；当用户单击了对话框上的【取消】

按钮时函数返回 0；发生错误时返回-1。若有参数的值为 NULL，则函数返回 NULL。

使用说明：该函数只是得到一个文件名，并没有真正打开该文件。若要打开该文件，则需使用 FileOpen()函数。

- GetFileSaveName()

函数功能：显示保存文件对话框，让用户选择要保存到的文件。

函数语法：GetFileSaveName(title,pathname,filename{,extension{,filter}})。

参数说明：

title: String 类型变量，指定对话框的标题。

Pathname: String 类型变量，用于保存该对话框返回的文件路径及文件名。

Filename: String 类型变量，用于保存该对话框返回的文件名。

Extension: String 类型，可选，使用 1~3 个字符指定默认的扩展文件名。

Filter: String 类型，可选，其值为文件名掩码，指定显示在该对话框的列表框中供用户选择的文件名满足的条件(比如*.*, *.TXT, *.EXE 等)。

返回结果：Integer 类型。函数执行成功时返回 1；当用户单击了对话框上的“取消”按钮时函数返回 0；发生错误时返回-1。若有参数的值为 NULL，则函数返回 NULL。

使用说明：该函数只是得到一个文件名，并没有真正打开文件。当需要打开文件时，依然需要使用 FileOpen()函数。

4. 字符串操作函数

- Match()

函数功能：确定字符串中是否包含指定模式的字符。

函数语法：Match(string,textpattern)

参数说明：

string: String 类型，为所要检查的是否匹配指定模式的字符串。

Textpattern: String 类型，为所用的文本匹配模式。

返回结果：Boolean 类型。若字符串 string 与模式 textpattern 相匹配，则函数返回值为 True，否则为 False；若指定的匹配模式无效或函数的两参数中任何一个未被赋值，那么函数返回值为 False；若两参数中一个参数的值为 NULL，则函数返回 NULL。

使用说明：textpattern 参数的值由元字符和普通字符组成，每个元字符都有不同的匹配含义。限于篇幅，此处对各元字符的匹配含义从略。

5. 打印与打印机设置函数

- PrintSetup()

函数功能：打开打印机设置对话框。

函数语法：PrintSetup()

参数说明：无参数。

返回结果：Integer 类型。函数执行成功时返回 1，发生错误返回-1。如果有参数的值为 NULL，则函数返回 NULL。

使用说明：当系统中安装了多种打印机时，会出现提示用户选择打印机的对话框，然后单击【Setup】按钮设置打印机各种特性；如果系统中只有一个打印机，则直接打开该打印机的打印设置对话框。

6. 数值计算函数

● Rand()

函数功能：得到 1 与 n 之间的一个伪随机数。

函数语法：Rand(n)。

参数说明：n：数值类型的变量或表达式，用以指定所要产生的随机数的上界，其有效值在 1 到 32 767 之间。

返回结果：与参数 n 的数据类型相同。函数执行成功时返回 1 到 n 之间的一个伪随机数(包括 1 和 n 在内)。若参数 n 的值为 NULL，则函数返回 NULL。

使用说明：另外可使用 Randomize()函数初值伪随机数发生器，这样可使其每次生成不同的伪随机数序列。

7. 提供帮助函数

● ShowHelp()

函数功能：显示应用程序帮助，此帮助使用 Windows 帮助系统进行操作。

函数语法：ShowHelp(helpfile,helpcommand{,typeid})。

参数说明：helpfile：String 类型，指定所要显示的帮助文件名称。

Helpcommand：HelpCommand 枚举类型，指定显示帮助的方式。其有效取值包括：Index!(显示目录主题，使用此值时不可指定 typeid 参数的值)、Keyword!(转移到由指定关键字所确定的主题)、Topic!(显示指定主题的帮助)。

Typeid：整型或字符串类型，可选，用来指定帮助主题。

返回结果：Integer 类型。函数执行成功时返回 1，发生错误时返回-1。若有参数的值为 NULL，则函数返回 NULL。

8. 定时操作函数

● Timer()

函数功能：在指定的时间间隔内反复触发指定窗口的定时器(Timer)事件。

函数语法：Timer(interval{,windowname})。

参数说明：interval：Long 类型，用来指定触发定时器事件的时间间隔(以秒为单位)，其有效值为 0 至 4 294 967。若该参数的值被设置为 0，则将关闭定时器，不再触发指定窗口的定时器事件。

返回结果：Integer 类型。函数执行成功时返回 1，发生错误时返回-1。如果有参数的值为 NULL，则函数返回 NULL。

使用说明：使用此函数可以周期性地触发指定窗口的定时器事件。但需要注意的是，在 Windows 系统中，此函数能够把计时的最小时间间隔为 0.055 秒(约 1/18 秒)，若把 interval 参数的值设置为小于 0.055，那么该定时器只能每隔 0.055 秒触发一次指定窗口的定时器事件。

9. 日期时间函数

● Today()

函数功能：得到当前系统日期。在某些情况下可同时得到当前系统时间。

函数语法: Today()。

参数说明: 无参数。

返回结果: Date 类型。该函数返回当前系统的系统日期。

使用说明: 单独调用该函数时, 此函数总是返回系统的当前日期。但当 Today() 函数用作某些函数的参数而该参数要求为 DateTime 类型时, Today 函数也能够同时返回当前系统时间。此外, 使用 Now() 函数也可得到客户机的当前系统时间。

10. 窗口操作函数

● Open()

函数功能: 用来打开一个 PowerBuilder 窗口。

函数语法: Open(window_name)

使用说明: 打开窗口并触发窗口的 Open 事件。

● Close()

函数功能: 用来关闭一个 PowerBuilder 窗口。

函数语法: Close(window_name)。

使用说明: 触发窗口 Close 事件、关闭窗口并释放窗口以及窗口上的控件所占据的内存。

● OpenSheet()

函数功能: 在 MDI 框窗口中打开 MDI 子窗口, 并在指定菜单中创建选择该子窗口的菜单项。

函数语法: OpenSheet(sheetrefvar{, windowtype}, mdiframe{, position{, arrangeopen}})。

参数说明: sheetrefvar: Window 类型, 指定要作为工作表打开的窗口名。

Windowtype: String 类型, 可选, 指定要打开窗口的类型(也就是窗口画笔中保存的窗口对象名)。

Mdiframe: 可选参数, 指定要旋转工作表的 MDI 框架窗口名。

Position: 可选参数, 指定所打开的工作表的名称作为一个菜单项显示在第几个菜单标题下面, 默认时, 被放在倒数第 2 个菜单标题下。

Arrangeopen: ArrangeOpen 枚举类型, 可选, 但若选用了此参数, 那么 position 参数也必须同时指定。Arrangeopen 参数告诉系统如何显示打开的工作表。

返回结果: Integer 类型。函数执行成功时返回成功时返回 1, 发生错误时返回-1。如果有参数的值为 NULL, 则函数返回 NULL。

使用说明: arrangeopen 参数的可能取值为: Cascaded!, 把个工作表放在另一个的上面, 每个向右下方偏移一点, 这样所有工作表的标题栏用户都能看到。该值是 OpenSheet() 函数的默认选择; Layered!, 将工作表显示在客户区的左上角, 并最大化工作表, 使其充满 MDI 框架窗口的整个客户区; Original!, 操作动作与 Cascaded! 参数相同, 只是不放大窗口, 而以窗口定义时的大小显示。

11. 其他系统函数

● SetNull()

函数功能: 将指定变量的值设置为 NULL。这里的变量可以是除数组、结构、自动实

例化对象之外的任何数据类型。

函数语法: `SetNull(anyvariable)`。

参数说明: **anyvariable**: 要将其值设置为 NULL 的变量。

返回结果: **Integer** 类型。函数执行成功时返回 1, 发生错误时返回-1。如果参数的值为 NULL, 则函数返回 NULL。

使用说明: **PowerBuilder** 应用程序在说明变量时, 并不自动地将其初值设置为 NULL, 而是根据类型的不同而设置不同的初值, 例如: 数值类型的变量自动初始化为 0, 字符串型变量自动初始化为空字符串(“”)。因此, 如果需要将某个变量的值设置为 NULL, 就需要使用 `SetNull()` 函数来完成任务了。一般来说, NULL 值往往应用在数据库值未确定的列上。

● `MessageBox()`

函数功能: 向用户显示或提示各种信息。

函数语法: `MessageBox(title,text[,icon[,button[,default]]])`。

参数说明: **title**: **String** 类型, 指定消息对话框的标题。

Text: 指定消息对话框中显示的消息, 该参数可以是数值数据类型、**String** 类型或 **boolean** 值。

Icon: **Icon** 枚举类型, 可选, 指定要在该对话框左侧显示的图标。

Button: **Button** 枚举类型, 可选, 指定显示在该对话框底部的按钮。

Default: **Integer** 类型, 可选, 指定作为默认按钮的按钮编号, 按钮编号自左向右依次计数, 默认值为 1。若该参数指定的编号超过了显示的按钮个数, 则函数将使用默认值。

返回结果: **Integer** 类型。函数执行成功时返回用户选择的按钮编号(例如 1、2、3 等), 发生错误时返回-1。如果任何参数的值为 NULL, `MessageBox()` 函数返回 NULL。

使用说明: 此函数的使用率非常高。当应用程序需要显示一段简短信息(例如显示出错、警告等信息)时, 没有必要自己从头创建窗口、安排控件, 使用 `MessageBox()` 函数既简单又方便。只有在响应该窗口后, 程序才能继续运行下去。`MessageBox()` 函数的 **icon** 参数指定显示在窗口中的图标, 它是枚举类型, 可能取值包括: **Information!**, **StopSign!**, **Exclamation!**, **Question!**, **None!**(无图标)。其中 **Information!** 是 **Icon** 参数的默认值。**Button** 参数指定窗口中显示哪些按钮, 有效取值包括: **OK!**(确定)按钮; **OKCancel!**(确定取消)按钮组; **YesNO!**(是否)按钮组; **YesNoCancel!**(是否取消)按钮组; **RetryCancel!**(重试取消)按钮组和 **AbortRetryIgnore!**(终止重试忽略)按钮组。

● `KeyDown()`

函数功能: 检查用户是否按了键盘上指定的键。

函数语法: `KeyDown(keycode)`。

参数说明: **keycode**: **KeyCode** 枚举类型或 **Integer** 类型, 指明所要检测的按键或某个键的 ASCII 值。

返回结果: **Boolean** 类型。若用户按了 **keycode** 参数指定的按键, 则函数返回 **True**, 否则返回 **False**。若参数 **keycode** 的值为 NULL, 则函数返回 NULL。

使用说明: 此函数通常在某个事件的事件处理程序中调用, 它并不指明用户键入了哪个字符, 而是说明当前事件(即事件处理程序中调用 `KeyDown()` 函数的事件)发生时用户正按着哪个按键。一般来说, 应用程序在窗口的 **Key** 事件或控件的按键事件中调用 `KeyDown()` 函数, 以检测用户是否按了某个特殊键。

3.10.2 用户定义函数

和其他程序设计语言一样, PowerBuilder 给出了几百个功能强大的标准函数, 为应用程序的开发提供了极大的方便。但由于应用程序的要求千差万别, 标准函数有时仍然满足不了要求, 所以还需要创建符合自己要求的函数。

自定义函数也分两种类型: 自定义全局函数和自定义对象函数。自定义全局函数独立于任何对象, 在整个应用程序中都能使用; 而自定义对象函数则与特定的窗口、菜单、用户对象等相关联, 是对象的一部分, 根据定义可能在整个程序中使用也可能只在对象内部使用。

1. 创建自定义全局函数

在 PowerBuilder 开发环境的主窗口中, 选择工具栏的【New】图标或选择【File】|【New】命令, 将出现标题为【New】的窗口, 如图 3.19 所示。选择【Object】标签的【Function】选项, 双击【Function】选项或单击【OK】按钮, 进入全局函数的定义, 如图 3.20 所示。

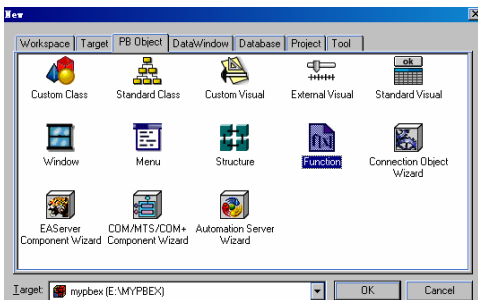


图 3.19 定义全局函数图

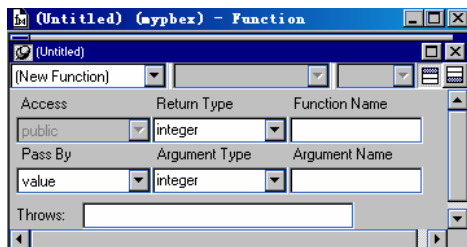


图 3.20 定义全局函数的属性

在图 3.20 中, 在【Function Name】选项中输入函数名、【Return Type】下拉框中选择函数返回值的类型、【Argument Name】选项中输入函数参数名(又称为形式参数, 简称形参)、【Argument Type】下拉框中选择函数参数类型、【Pass By】下拉框中选择参数传递方式。

自定义全局函数的命名一般用“f_”作前缀。

参数传递方式有 3 种。

Value: 值传递, 即将实际参数的值传递给函数参数。

Reference: 地址传递, 即把实际参数的地址传递给函数。此时, 如果函数修改了形式参数的值, 那么实际参数的值也就被修改了。

ReadOnly: 地址传递(只读), 即把实际参数的地址传递给函数。不过不允许修改参数的值。

要增加或删除参数, 可在图 3.20 中单击鼠标右键, 出现如图 3.21 所示的弹出式菜单。

【Add Parameter】选项为增加参数, 【Insert Parameter】选项为插入参数, 【Delete Parameter】选项为删除参数。

函数名及参数定义好后, 开始输入函数代码。在定义函数的下面窗口内编辑代码, 如图 3.22 所示, 或选择【View】|【Script】菜单命令, 将打开函数代码编辑窗口。函数返回

值的类型、参数名、个数及类型可以随时更改。若函数有返回值，必须立即输入代码，即必须写一条 `return` 语句，否则产生错误。

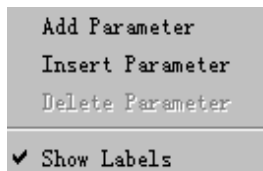


图 3.21 增加、插入、删除参数

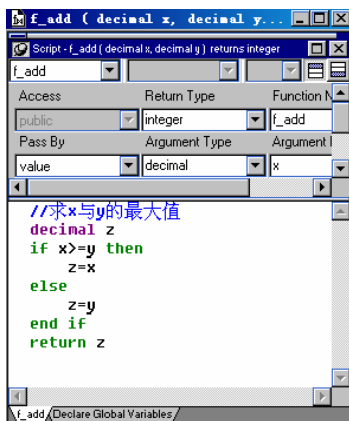


图 3.22 编辑函数代码

定义好的全局函数和标准函数一样使用。

2. 创建自定义对象函数

可以为 Application Object 对象、窗口对象、用户对象创建自定义函数，这种函数称为对象函数。对象函数一般只能在该对象内使用。当该对象正在打开且该函数的 `Access` 属性为 `public` 时，其他对象的程序可以调用该函数，不过需要在函数前加对象名，如 `w_pipe.wf_initial()`。如果函数所有的对象没有被打开(即不在内存中)，则该对象函数不能被其他对象的程序调用。

首先打开定义函数的对象，然后打开【Script】代码编辑窗口，先选择【Functions】选项，再选择【New Function】选项，将出现函数定义窗口，如图 3.23 所示。

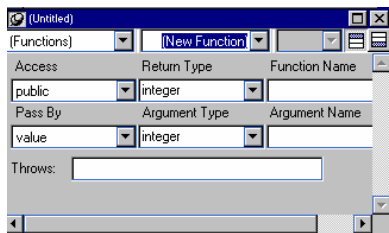


图 3.23 定义对象函数的属性

如图 3.23 所示定义对象函数和如图 3.20 所示定义全局函数几乎一样，不同之处是定义对象函数可以规定该函数的访问属性 `access`，而全局函数不可以。`Access` 属性缺省值为“`public`”。

访问属性 `access` 有 3 个选择。

Public: 该函数在整个程序中都可访问。

Private: 该函数只能在当前对象和程序中使用，但不能在该对象的后代的程序中使用。

Protected: 该函数只能在当前对象的程序以及该对象的后代的程序中使用。

对象函数的命名规则一般与对象有关,如应用对象 Application Object 的函数一般以“af_”作前缀,窗口对象 window 的函数一般以“wf_”作前缀,菜单对象 menu 的函数一般以“mf_”作前缀,用户自定义对象的函数一般以“uf_”作前缀。这些规则清楚地表明了函数所在对象的类型,便于程序的维护。

函数名及参数定义好后,开始输入函数代码。在定义函数的下面窗口内编辑代码,若函数有返回值,必须立即输入代码,即必须写一条 `return` 语句,否则产生错误。

与全局函数一样,对象函数的返回值的类型、参数名、个数及类型也能随时更改,不过与全局函数有点不同。这里系统会给出一个提示信息,以确认是否更改。

定义好的对象函数和标准函数一样使用。不同之处是在其他对象的程序中调用时,应在函数名前加上函数所在的对象名如 `w_pipe.wf_error(num)`,当然 `w_pipe` 必须已被打开。

3.11 小 结

本章介绍了 PowerScript 语言的基础知识及其用法。PowerScript 语言是运用 PowerBuilder 进行程序设计的基础,各种函数、事件处理程序的设计都离不开 PowerScript 语言,读者必须对其了解和掌握。掌握 PowerScript 语言对于深入理解 PowerBuilder、灵活高效地编写更加强大的 PowerBuilder 应用程序是十分必要的,尤其是嵌入的 SQL 语句的运用,对基于数据库的应用开发非常重要。

3.12 实 训

实训目的

- (1) 熟悉创建全局函数的方法及函数设计画板的使用。
- (2) 掌握 SQL 语句生成画板的操作方法。

实训内容

- (1) 复习有关函数及 SQL 语句的内容。
- (2) 预习相关窗口及控件的基本使用方法。
- (3) 在数据库中有一个存放用户名以及口令的表 Users,该表至少有 2 个字符类型的列,列名分别为 `mingcheng`(用户名)和 `password`(口令)。该表有一条记录数据,`mingcheng` 列值为“SYSTEM”,`password` 列值为“SQL”。
- (4) 函数原形为: `int f_GetUserPassword(String as_username,Ref String as_password)`。
其中,`as_username` 是一个值传递的字符串形参数,`as_password` 是一个引用传递的字符串型参数,用于返回指定用户名对应的口令。操作成功返回 1,失败(找不到对应用户的口令)返回-1。

实训步骤

(1) 打开【New】对话框，选择【PB Object】标签中的【Function】图标，如图 3.24 所示。

(2) 单击【OK】按钮，进入全局函数设计画板，在相应的编辑框中输入函数名、返回值类型、参数名、类型、传递方式，如图 3.25 所示。

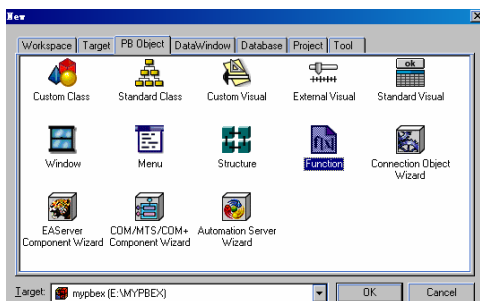


图 3.24 【New】对话框

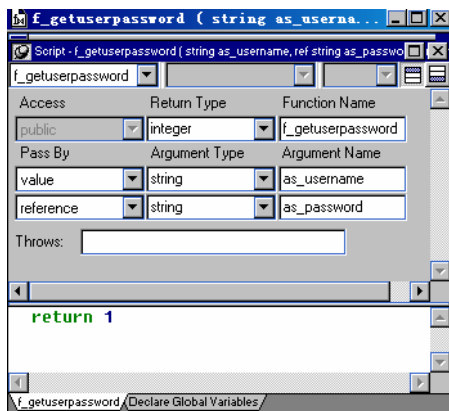





图 3.25 全局函数设计画板窗口

(3) 生成一个 SELECT 语句，用来从数据库中检索数据。把光标放在函数脚本编辑视图上，单击工具栏的粘贴 SQL 语句图标 ，在弹出的快捷菜单中(见图 3.26)选择【Select】命令，进入 SELECT 语句生成画板，如图 3.27 所示。选择 password 项后，单击工具栏上的按钮 ，弹出【Into Variables】对话框，在该对话框的 user.password 对应的编辑栏中输入: as_password，再单击【OK】按钮返回 SELECT 语句生成画板，在该窗口中选择【Where】标签，可通过移动的滚动条来选择相应 SELECT 语句中的 WHERE 子句。最后单击工具栏上的返回按钮  回到函数设计窗口，此时会在函数的脚本区生成对应的 SELECT 语句，再根据设计要求完成该函数的代码。

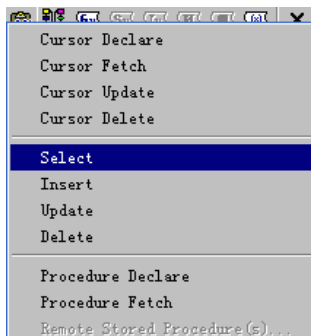


图 3.26 选择【Select】命令

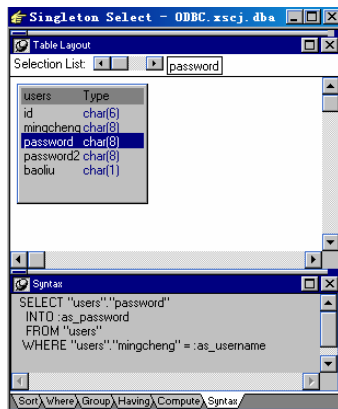


图 3.27 Select 语句生成画板

```

SELECT "users"."password"      //自动生成的 SQL 语句,
    INTO: as_password          //如果你所连接的 Database Profile Setup 对话
                                //框的 Syntax 标签不选中
    FROM "users" //Enclose Table and Column Name in Quotes 选项, 则生成的
                                //语句中不会加引号
    WHERE "users"."mingcheng" =: as_username
IF SQLCA.SQLCode=0 THEN        //SQL 语句执行正确
RETURN 1
ELSE                            //SQL 语句执行错误或找不到满足条件的记录
    RETURN -1
END IF

```

(4) 最后, 保存设计好的全局函数。

(5) 运行测试。

函数自己不会运行, 必须通过调用才能执行函数体中的代码, 可以用表 Users 中已有的一条记录来测试该函数, 在应用程序的 Open 事件中编码如下。

```

SQLCA.DBMS= "ODBC"
SQLCA.AutoCommit=False
SQLCA.DBParm= "Connectstring='DNS=xscj;UID=;PWD=' "
CONNECT;
String ls_password
Integer li_ret
Li_ret=f_getuserpassword("SYSTEM",ls_password)
If li_ret=1 AND ls_password="sql" THEN
    MessageBox( "", "有效的用户与口令")
ELSE
    MessageBox( "", "无效的用户与口令或数据库连接错误")
END IF

```

3.13 习 题

1. 填空题

- (1) PowerScript 语言中用做续行的符号是_____。
- (2) 要想查看 PowerBuilder 都支持哪些系统对象以及它们之间的继承关系, 可以使用_____。
- (3) 要查看系统都有哪些枚举类型以及对应的枚举值, 可以打开_____窗口的选项卡。
- (4) PowerBuilder 的变量作用域共有 4 种: _____变量、_____变量、_____变量和_____变量, 在使用它的事件处理程序或函数中说明其作用域仅限于说明它的程序段的变量是_____变量。根据变量的命名规则, ii_abc 应该是_____类型的_____变量, ls_abc 应该是_____类型的_____变量。
- (5) 结构体分为全局结构体和对象结构体。_____结构体能够在应用程序的任何地方使用, 声明全局结构体变量的作用域是_____。_____结构体只能在对象以及继承于该对

象的派生对象中使用,对象级结构在_____画板中定义。一般将在整个应用程序中都要使用的结构体应定义为_____结构体,将只在某个对象中使用的结构体定义为_____结构体。

(6) 表达式中如果有一个变量的值为 NULL,那么表达式的值为_____。

(7) 函数使用时除了要了解函数的功能以外,还要了解函数参数的____、____、____以及函数的_____类型。

2. 简答题

(1) This、Parent、ParentWindow、Super 这些代词指的是什么?在代码中用这些代词有什么好处?

(2) 下面的标识符哪些是合法的,哪些是不合法的?

-page, this, lock_open, 4in, @mail, per%, test2, _odd, abc#fd

(3) 注释有什么作用?PowerScript 中的注释方法有几种?它们之间的区别是什么?

(4) 给变量赋空值(NULL)的途径有几条,如何实现?怎样测试变量或表达式的值是否为空值?

(5) PowerBuilder 有哪几种数据类型?

(6) PowerBuilder 有哪几种运算符?

(7) 中止程序的运行,中止函数的运行,中止循环语句的运行分别用什么语句?

(8) 比较 PowerScript 中的条件语句和 Choose 语句、For 循环语句和 Do 循环语句的功能和使用场合。

(9) 下面的语句执行后变量 i 的值是多少,试说明理由。

```
integer i
i=32767
i=i+1
```

(10) 在定义函数参数的传递方式时,Pass By 列表框有哪 3 种供选值?分别表示什么含义?

(11) 在定义函数参数时可以指定哪三种函数的访问类型?分别表示什么含义?

(12) 如何使用游标操作从数据库中读取多行数据?

3. 编程题

(1) 设计一个全局函数返回指定班级(编号)的学生人数。

(2) 设计一个应用,能求任意输入两个数的最大公约数。(提示:在一个窗口上设置两个单行文本编辑框用来输入整数,一个命令按钮求最大公约数并输出,也可以定义一个求最大公约数的窗口函数)

(3) 求 Fibonacci 数列的前 20 项之和。Fibonacci 数列是指前两项分别为 0 和 1,从第 3 项起,每一项都是前两项之和。例:0, 1, 1, 2, 3, 5, 8, 13, 21, ...。

(4) 判断一个数 m 是否是素数。

(5) 用穷举法找出 1~100 之间的质数并显示出来,分别使用 WHILE、DO-WHILE、FOR 循环语句。