INCLUDES ONLINE COMPANION

POWERBUILDER 4.0

FOR WINDOWS

深入编程

[美] Jason Coombs
Ted Coombs
Ed Ashley 著

舒少文 屈健 等译
张拥军 审校

为PowerBuilder
程序员提供
捷径和技巧

POWER TOOLKIT

VENTANA

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.co.cn

Power Builder 4.0 for Windows

深入编程

[美] Jason Coombs Ted Coombs Ed Ashley 著

舒少文 屈健 等译

张拥军 审校

電子工業出版社
Publishing House of Electronics Industry

内 容 简 介

本书包括十四章内容，主要有：用对象进行程序设计；建立类库；高级报表设计；交叉平台下的应用程序开发；群组开发管理；建立 Windows 3.11 DLL 的技巧；Power Builder 中的 Winsock TCP/IP 程序设计；用 Internet 资源进行程序设计；以及性能问题的讨论等。本书是一本非常有用的参考书，通过阅读本书，读者能很快掌握 Power Builder 技术，对深入编程很有帮助。

书　　名：Power Builder 4.0 for Windows 深入编程
著　　者：[美] Jason Coombs Ted Coombs Ed Ashley
译　　者：舒少文　屈健　等
审 校 者：张拥军
责任编辑：张丽华
排版制作：电子工业出版社计算机排版室排版
印 刷 者：农业出版社印刷厂
出版发行：电子工业出版社出版、发行　URL：http://www.phei.co.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036　发行部电话：68214070
经　　销：各地新华书店经销
开　　本：787 × 1092 1/16　印张：22.5　字数：572 千字
版　　次：1997 年 7 月第一版　1997 年 7 月第一次印刷
书　　号：ISBN 7-5053-3956-7 / TP · 1721
定　　价：36.00 元
著作权合同登记号 图字：01-96-1369

译者的话

PowerBuilder是国内目前用得最多的大型数据库应用软件开发平台之一，它支持 UNIX、Windows、Windows NT、Windows 95 和 Macintosh 等多种平台，可用它在交叉平台上进行应用程序开发，而且所开发出的应用程序可移植性好。作为一种成熟的数据库应用软件开发工具，PowerBuilder 已获得了广大用户的喜爱，但是目前国内市场上发行的有关 PowerBuilder 的中文资料内容大多都是入门性质的，而不少 PowerBuilder 应用程序的高级开发人员苦于找不到一本深入介绍 PowerBuilder 的资料。本书正是为满足广大读者的这一要求而引进翻译出版的，相信会对广大读者深入了解和使用 PowerBuilder 有所帮助。

本书译自《PowerBuilder 4.0 for Windows Power Toolkit》一书，作者是 PowerBuilder 方面的专家，有着从事 PowerBuilder 应用程序开发方面的丰富经验。书中用大量实例深入介绍了面向对象的概念、数据窗口、DLE 和 DDE、设计应用程序时应加以考虑的系统安全问题；高级数据库技术问题；高级 SQL 的使用、高级程序设计方面的问题；集体开发工程项目时的管理问题；创建 DLL 的技巧以及 Internet 方面的设计考虑等等。本书所包含的内容及其深入程度在一般的 PowerBuilder 书籍是找不到的。我们相信，本书的出版一定会对广大从事 PowerBuilder 应用程序开发的高级工程设计人员有很大的帮助。

参加本书翻译工作的还有王子恢、李明、郭达、范彬、王二、张一夫、刘博侃等。

因水平有限，翻译难免有不确切之处，还请广大同仁批评指正。

译者

1996 年 6 月

作者简介

Ed Ashley 是 Sun Microsystem 公司的系统工程师，他在 IS 领域已经工作了 12 年，他花了 6 年的时间开发事务和信息 Client - server 系统。除了软件开发之外，Ed 还在 San Diego 分校，即加利福尼大学教授计算机科学课。

Jason Coombs 不仅是科学家、工程师、企业家和作家，还是 Pacific Knowledge 的副校长。他是 Internet 和软件工程工具等科学方面的书的合作者，他已出版的著作包括《Setting up shop on the Internet for Dummies》和《PowerBuilder 4.0 Programming for Dummies》(均已由 IDG Books 发行)。

1972 年以来，Ted Coombs 从事过各种技术工作——电子、激光、机器人、计算机网络、电信、数据库和软件开发工作。作为一名研究方面的科学家，他是“Science and engineering think tank Pacific Knowledge of Santa Cruz，CA”的校长和“Pacific Knowledge Foundation”的董事长及“Center for Multidisciplinary Research”的研究员。

感谢：

好伟大的工程！当我第一次看到这本书的提纲时，就认为这是一项非常了不起的工作。回想起来，我料想这是对的。尽管在本书的封面上有少数几个人的名字，但是如果没有其他许多人的工作，本书将不会存在。我想感谢那些为产生出高质量的书籍进行说明性工作的人们。

我首先要感谢 Ted 和 Jason，他们启动了这项工作并贡献出他们的力量来解决所有的论题。也要感谢 UCSD 的 Jerry Singer 的鼓励才使我没有半途而废；Kerri Dekosier 扫清了一起障碍并做得那么好；感谢 NexGen SI 的 Tim Beck，他为我们筹集了资金。

没有 Matt Wagner、Margot Maley 和 Waterside 的所有伙计们，我们将不能完成此项工作。他们具有令人难以置信的大度，没有丝毫怨言。

当此项工程付诸行动时，Ventana 的大批人士团结在一起使各项工作得以正常进展。非常感谢你们各位。正如你们所知，如果没有你们大家的努力，本书将不会产生。特别要感谢“永远的乐天派”Cheri Robinson，她在本书的写作过程中一直给予支持；尤其要感谢 Eric Edstam、Tim Mattson、Enrique Villalobos、Pam Richardson、Walt Bruce、Nancy Crumpton、Lynn Jaluvka、Scott Hosa 和 John Cotterman。

我们中的每一位都要感谢我们各自的妻子和其他重要人士，在最后几个月里忍受我们这几个讨厌的工作狂。我尤其要感谢我终身的编辑和最好的朋友：我的妻子 Diana。

——Ed Ashley

目　录

前　言

众所周知，在您们当地书店的书架上可以得到大量的 PowerBuilder 方面的书籍。我们编写此书是想对现有书籍增加一些价值，而我们相信，我们已经达到了这个目标。总之，《PowerBuilder 4.0 for Windows Power Toolkit》(即本书)的作者有十年以上的开发 PowerBuilder 应用程序的经验。我们想把我们的实际经验带给大家。本书将集中讨论围绕 PowerBuilder 开发的高级主题。

一、谁应该阅读本书

我教授 PowerBuilder 的经验是：要花六个月左右的时间才能完全熟悉 PowerBuilder 的工具箱(toolset)。本书意在讨论 PowerBuilder 论题的高级内容。所以，读者是那些至少有六个月 PowerBuilder 使用经验的 PowerBuilder 开发人员。

那些准备扩展其工具箱和提高开发水平的读者应该阅读本书。我们讨论的问题在 PowerBuilder 手册中是找不到的，而且在某些情况下，我们自己通过试验也发现自己也出现过错误。读者通常熟悉一般的软件开发，希望探索一个全新的开发领域。

二、书中有些什么内容

在第一章“用对象进行程序设计”和第二章“建立类库”中，我们讨论面向对象的问题，主要讨论可重用软件的类。然后我们将向大家演示怎样创建自己的可重用的类，我们也提供我们自己创建的一些类给大家。

第三章“DataWindows(数据窗口)”和第四章“高级报表设计”将说明 DataWindows，然后通过讨论高级报表的特性来拓展这个论题。我们认为其它论题包括不了这个论题。

第五章“OLE 和 DDE”将向大家介绍开发 OLE 应用程序的过程，并用代码演示怎样建立自己的面向 OLE 的应用程序。

在我们的每一条 PowerBuilder 开发经验中，我们都面临着一种或另一种安全方面的问题。我们奉献第六章“安全问题”来说明涉及 PowerBuilder 内部的安全问题。另外，我们还讨论其它安全领域，这些领域肯定会影响开发者的 PowerBuilder 应用程序——数据库和网络安全。

在第七章“高级数据库技术”和第八章“高级 SQL”中，我们演示使用数据库和 SQL 的一些高级方法。本书作者中有两位做过数据库管理员，我们希望在这两章中阐述一些成果性的见解。

第九章“高级程序设计专题”对高级编程问题作一个一般性的讨论。

接着在第十章“开发交叉平台下的应用程序”中，我们开始从 PowerBuilder 开发核心叉开，来讨论 PowerBuilder 的辅助性问题。本章检查各种操作系统之间的一些差异，帮助读者准备应付将来不可避免的交叉平台的开发问题。

第十一章“群组开发的管理”解释使开发队伍保持一致的问题和技术。

第十二章“创建 Windows 3.1 DLL 的技巧”中，我们讨论通过建立 Windows 动态链接库 DLLs (Dynamic Link Libraries)来扩展 PowerBuilder 环境。有了 DDLs，就可以建立具有较好性能的代码，并从 PowerBuilder 内部来调用它。

我们不时地发现：开发者在创建客户机/服务器方式的应用程序时只使用 PowerBuilder。在第十三章“有关的软件开发工具”中，我们讨论 PowerBuilder 附带的外部实用程序，如 DataWindow Syntax 实用程序和一些第三方销售商提供的工具，如 RoboHELP。

第十四章“在 PowerBuilde 中进行 Winsock TCP/IP 程序设计”和第十五章“利用 Internet 资源进行程序设计”处理 Internet 资源并从 PowerBuilder 内部与 Internet 进行交互。这些章节将帮助读者开始扩充 PowerBuilder/Internet 开发的工具箱。

为演示本书中提到的一些技巧，我们开发了一个名为学生登记系统(Student Registration System)的样本应用程序。在第十六章我们将解释这个样本程序，这样读者就可以用它作为自己应用程序开发的起点。这个应用程序也包含在 CD-ROM 上。

最后，我们讨论一个对每位开发者来说都是重要的问题——性能。象安全问题一样，我们在每个 PowerBuilder 项目中都会面临性能问题。在第十七章“性能问题的讨论”中，我们奉献出我们已经发现的技巧和窍门。

第十七章后面是几个附录。

• 附录 A“关于 Online Companion”说明 PowerBuilder Online Companion 的在线资源(在 World Wide Web 上的 Ventana Online 场所中)是怎样来补充本书信息的。

• 附录 B“关于 Companion CD-ROM”说明本书附带的 Companion CD-ROM 上的内容。CD-ROM 上包含第三方厂家提供的产品，用于改善 PowerBuilder 的性能和可用性。读者也可以从中找到一些例子，它们提供本书中说明的建立应用程序的解释。

• 附录 C“PowerBuilder 与 Windows95”是一篇由美国 Management System 公司的 Dave Rensin、Gregory McIntyre 和 Timothy Jarrett 投递到《1995 Mid -Atlantic PowerBuilder Users Conference》的文章的翻版，它试图说明操作系统主要机构的一些变化，以及这些变化可能对 PowerBuilder 程序员所产生的影响。

• 附录 D“PowerBuilder ORCA 参考”包含了在 ORCA(Open Repository CASE API)上可能得到的 Powersoft 文档的重印材料，它为高级开发人员和 CASE 销售商提供了一个简单的访问 PowerBuilder 库函数的方法。本书作者感谢 Powersoft 允许我们使用这些文档。

三、结　束　语

我们的确不知道有关 PowerBuilder 的所有东西，坦率地说，任何人都是这样。当我们在处理 PowerBuilder 开发项目时得到的经验越多，增加到我们知识库中的技术也就越多，当然，每天出现的技术也就越多。本书应作为一本参考书，我们希望读者了解一些可以加到自己工具箱中的技术，我们希望使您们的下一个 PowerBuilder 工程更出色！

Ed Ashley，Jason Coombs 和 Ted Coombs

第一章　用对象进行程序设计

1969年，挪威的Kristin Nygaard先生发明了用面向对象(OO)来处理进出挪威海湾的船只复杂的模型化问题。创建一个能够处理进出海湾船只的所有变量的计算机程序所带来的挑战，对于传统的结构化程序设计来说的确是太巨大了。读者的程序设计可能不需要有挪威海湾的详细资料和复杂程度的报告，但是可以从使用对象来模型化现实世界中的事物和事件的思想和原理中获得启迪。在本章中我们将讨论什么是对象(object)和为什么要用它们来设计程序，我们也介绍在PowerBuilder中面向对象程序设计的一些基础知识(面向对象的讨论将在后面的章节中较详细地介绍)。

一、面向对象程序设计的基本原理

什么是对象？大家经常听到的有关面向对象程序设计的陈词滥调是“每个事物都是一个对象”，这并不确切。较精确的观察方式应该是：在程序中，任何事物均可用一个对象来表示。一个对象是一个程序实体(entity)，它把现实世界中事物所代表的行为和属性组合成一个独立的程序代码单位，这个独立的程序代码单位包含描述现实世界中事物的数据元素和操作那些数据元素的函数。数据元素代表对象的属性，函数则表示对象的行为。一个对象可以是人的模型(如雇员)；人们所看到的事物(如船或机器)；或者看不到的事物(如管理数据通信的一个进程或打印报表的一个进程)。人们所熟悉的一些较普遍的对象是窗口(window)和控件(control)，如检查框(checkbox)或无线按扭(radio button)。

一个对象中的特征数据元素叫做对象的属性(attribute)，属性是描述对象某些方面的信息。例如，一个雇员对象可能包含雇员的姓名、ID号码、出生日期和受雇日期这样的属性。

对象的属性可以用一些函数来进行操作。例如，雇员对象可能有几个函数用来分配新的ID号码、返回ID号码的值给其它函数和自动终止，这些函数叫做对象的方法(method)(在本书中，我们将交替地使用方法和函数这两个术语)。

每个对象都创建于一个叫做对象类型(object type)或类(class)的定义。读者可以认为对象类型是一个逻辑上相关的对象的模板(template)。这种定义包含函数、内部函数和存储属性的变量，从某种意义上说，它就是一个模板。程序员并不操作类本身，而是对类所建立的对象或实例进行操作(在技术上，严格地说，一个对象就是类的一个实例)。

面向对象可以使一个复杂的程序简单到模型，模型经常被用来表示现实世界中人们周围的人们和其行为过程。例如，读者可以开发邮递员对象来投递电子邮件；登记课程的学生对象和用于税务计算的记帐对象。当属性和函数与对象组合在一起时，这个对象就代表了现实世界的一个实体(比如一个学生对象)，然后开发者可以用应用程序结构来识别，这比把所有代码放在一个大的、顺序化的程序中处理起来要简单得多。当所有代码都放在一个程序中时，一个已知实体的不同数据元素和过程分散在程序的各个地方，给找到贴切的代码和维护带来困难。让我们仔细地看一下面向对象(OO)的主要原理和为什么在开发应用程序时要使用面向对象的方法。

1. 压缩(Encapsulation)

把属性和函数放在一起作为一个自包含代码单元的 OO 原理就叫做压缩(encapsulation)。在传统的过程化(程序设计)语言(如 COBOL 和 C)中,过程和数据元素可以被分散到源代码的各个部分。把函数和属性存放在一起,压缩在对象的内部,就使对象易于搬动和重复使用。由于属性和函数包含在对象内部,所以它们就可随对象一起搬动。在过程化语言中,要把单个实体的代码拷贝或搬移到其它程序中,就需要从几个地方剪切和粘贴。把函数(或方法)和属性(或特征)压缩到单个对象中可提供相当容易的维护。由于操作一个已知实体的代码存在该实体内部,所以维护人员就确切地知道到哪里查错或做一改动。例如,假定一个应用程序含有一个名为 w_search 的窗口,它提示用户输入一个搜索式(criteria),然后搜索一个 SQL 查询中使用的搜索变量。如果按这种方式构成的搜索变量有错的话,那么开发人员就知道有问题的代码是在 w_search 窗口的某个地方,而不是在其它对象的代码中。

压缩规则要求:一个对象中的方法只能对其内部的属性进行操作,而不可以操作其它对象的属性。相反,一个对象中的属性只能由该对象中的函数来操作,而不可以由另一个对象中的函数操作。因此,在一个面向对象的应用程序中,对象 A 不能在对象 B 上进行操作。为得到相同的结果,对象 A 可以向对象 B 发送一条消息(message),告诉它在自己内部操作(即调用对象 B 的一个函数)。在 PowerBuilder 中,一条消息就是对另一个对象的函数的调用或触发另一个对象的事件。

程序是一个由通过互相发送消息进行通信的对象组成的系统,而不是发向计算机的一串线性命令。对象消息通过函数或事件来发送和接收。一个对象通过调用一个函数(设计成用来提供信息)来请求另一个对象的信息。

对象就象是一些建筑模块,而应用程序的开发就是把一个环境中适当的对象集合在一起,让它们通过消息互相通信。在一个设计得很好的面向对象的应用程序中,对象可被修改或替换以便增加新的功能,而不用改变应用程序的其它部分。新的对象可以简单地"加入"。

这种设计有许多好处。例如,如果需要改变对象 B 中一个已知的函数,那么只需改变对象 B 中的代码。对象 A 被写成简单地给对象 B 发送消息,调用一些动作。这没什么变化,对象 A 仍然让对象 B 完成那个动作,改变的只是这个动作本身。如果对象 A 中有操作对象 B 的代码,那么,改变对象 B 的行为就需要程序员既修改对象 A 也修改对象 B。如果只是对象需要改变行为,那么为什么程序员还应该改变对象 A 的代码而冒增加新的错误的危险呢?

2. 继承(inheritance)

面向对象的程序设计最有价值的前提就是软件的可重用性(reusability),这种思想是:一个程序或程序的一部分可以一次写成而用在整个应用程序的多处。这样的好处是节省了时间,因为相同或相似的功能只设计一次,而可多次使用该功能;另一个好处是出错少,因为编写代码和调试是在一处进行,而不用考虑多么频繁的调用它。一旦建立了一个对象类型,就可以在相同的应用程序或其它需要它的程序中共享这个对象中的代码。一个对象的属性和函数可被重复地使用。当一个对象使用另一个对象类型的属性和函数时,就叫做继承。重用(reuse)与拷贝代码是不同的,当代码被重用时,原始代码就被实际调用(从重用的类中调用)。祖先(ancestor)是一些类,其它类则从它们那里继承;子孙(descendant)是从祖先那里继承而来的类。可从单一的祖先创建任意数量的新对象。有些对象类型可以既是祖先也是子孙。

多重继承(multiple inheritance)是指一个子孙可以从一个以上的直系祖先那里继承属性和函数,就象一个小孩继承了其父母的特征一样(这与一个从也是子孙的祖先那里继承而来的子孙的情况不一样——如孩子到父母到祖先的关系)。单一继承(single inheritance)是指子孙只从一个直系祖先那里继承。PowerBuilder 只支持单一继承。

从一个祖先那里继承为开发人员提供了扩展子孙对象类型属性和函数而不影响祖先的能力,扩展祖先的属性和函数意味着增加继承的功能。新建立的对象类型具有与祖先相同的属性和函数。开发人员通过选择做如下任何一件事情来处理祖先代码的继承:

- 接受祖先原来的代码
- 扩展祖先的代码
- 完全覆盖(override)祖先的代码

如果开发人员既不选择扩展也不选择覆盖祖先的代码,那么子孙对象的行为就和祖先完全一样了。

选择扩展祖先代码可使继承的效果最佳。在这种方式中,子孙使用祖先的代码并且需要额外的代码,这些额外的代码形成了子孙与祖先的差异。例如,用于维护"人员"的祖先窗口可被继承并扩展成维护一个"学生"。由于学生是人并且有姓名、地址等,所以学生窗口就可以扩展祖先的代码,增加操作学生的特别属性(学生 ID、声明的专业等)的代码。维护人员的祖先代码可以完全地用于学生窗口,因为它使用人员窗口的所有属性。开发人员只是增加新属性的特别代码。

样本应用程序

我们在本书中用来作为例子的样本程序叫做学生注册系统(Student Registration System),保存在 Companion CD-ROM 上,详细的解释在第十七章。该应用程序说明本书每一章中描述的 PowerBuilder 开发概念,读者可以把它作为自己开发的模板。

如果开发者选择覆盖祖先代码,那么子孙将实际地替换祖先的代码以便处理特殊的事件。一旦开发者覆盖了祖先的大部分代码,那么他就必须决定是否需要另外的对象。从一个祖先继承 50 个函数没有多少好处,那么就覆盖其中的 48 个。如果这种情况发生,那么这就暗示着需要重新设计这个祖先或者需要一个新的祖先对象类型。

3. 多形性(polymorphism)

面向对象的另一特性就是多形性。多形性这个词的意思是"具有多种形式"。这是一种增加一个对象的功能而不必完全建立新的函数的能力。多形性允许我们针对子孙对象的每一级增加特别的性能。这种能力叫做特性(specificity)。修改从较普通对象类型继承来的对象的函数可以满足较特殊对象的需求。从技术上讲,多形性是向不同对象发送相同名字消息和从这些对象接收不同结果的能力。

例如,读者可能有一个普通的带有 payment()函数的 emplyee 对象类型。从这个普通的对象类型可以产生出较特殊的类型:SalariedEmployee 和 HourlyEmployee。这个祖先对象 employee 有非常一般的函数,确保雇员能得到薪水。继承来的 HourlyEmployee 对象可能对 payment()函数有所修改以便允许处理如加班费用或时间卡跟踪的问题。SalariedEmployee 对象对 payment

()做了几处修改以便处理固定年薪均等地分期付款。多形性的实质就是这两个对象都有保证雇员都能得到工资的 payment()函数。HourlyEmployee 和 SalariedEmployee 对象的函数利用继承建立在祖先的 payment()函数之上。多形性可被看作是向不同对象发送相同消息的能力,每个对象按照自己的方式作出响应。

二、面向对象与基于对象

用有点松散的定义来说,面向对象(object_oriented)这个术语意味着任何支持多形性、继承和压缩的开发语言或开发环境。用于描述不纯粹是面向对象的开发语言或开发环境的术语就是基于对象(object_based)。PowerBuilder 是一个基于对象的开发环境,众多理由之一就是 PowerBuilder 不支持多重继承,它只支持单一继承。

PowerBuilder 不强迫人们以面向对象的风格进行开发。事实上,使用 PowerBuilder 时人们可以不遵守 OO 众多规则中每一个单独的规则。例如,读者看到将要发生的第一件事情就是:当试图写一个使用关系型数据库的面向对象程序时,就会违反压缩规则。保存在数据库中的信息对所有对象来说都是可存取的,而且如果一个对象需要学生的名字,(如果可以的话)为什么不让学生对象直接从数据库中得到这个名字呢?

使用 PowerBuilder 很容易受到诱惑而写出传统的由上至下的应用程序(通常,由上至下的应用程序以一个菜单开始并从那里执行)。然而 OO 程序设计有许多好处,正如本书和其它书中演示的那样。即使 PowerBuilder 不强迫完全的面向对象的程序设计,但是开发人员没有理由说不能遵守这些规则而好象 PowerBuilder 确实强迫实行这些规则似的。面向对象的好处仍然是现实的。我们在全书中通篇描述各种方法和正反面的理由,为的是在 PowerBuilder 中处理继承、压缩和多形性的问题。在这一点上要知道:对于多样化的层次(degree),每个 OO 需求均可在 PowerBuilder 中实现。

三、在 PowerBuilder 中建立对象

PowerBuilder 有一套标准的对象类型,读者可以用它们来建立自己的应用程序。这些对象类型具有开发者可以使用和操作的内部行为和特性。这些是对象的定义,而不是实际的对象。读者可把它们作为切饼的刀具,同时又可以用它们做饼。最常用的对象类型是窗口、用户对象和菜单。然而这些对象类型只是 PowerBuilder 提供的一些较高层次的子孙。

因为 OO 程序被设计成具有模拟现实世界中的人及其行为过程的对象,所以读者可能需要一个以上标准的 PowerBuilder 对象类型。PowerBuilder 允许用户(使用 User Object painter)创建自己的对象类型,而不是强迫人们只使用所提供的对象类型。这样的对象叫做用户对象(user object)。Powersoft 认为 DataWindow 是 PowerBuilder 中功能最强的对象。按照功能和实用性来说,用户对象是第二位的。开发者也可以建立用户对象以增加 PowerBuilder 内部的对象类型,例如可以建立一个具有附加值的按钮(如用户定义的其它事件)。

在 PowerBuilder 中建立一个对象类型和建立一个对象的区别是模糊的。PowerBuilder 的 painter 建立对象的定义(开发者可用库输出函数把一个由 painter 建立的定义输出到一个 ASCII 文件,然后用 PowerBuilder 编辑器查看这个定义。在 PowerBuilder 开发环境中按动 shift + F6 组合键即可使用编辑器)。用 Window painter 建立一个窗口时,就是在建立一个对象类型。您可

能认为自己正在建立一个对象，文档甚至会指出您正在建立一个对象，但是您不是，您正在建立的是一个窗口的定义。您的窗口实际上并没变成用户看得见的一个对象，直到运行时执行 PowerScript 函数为止。这时 PowerBuilder 使用您建立的新对象类型并用它建立一个对象。

这种二义性并不一定是件坏事。如果我们在 C++ 中编码，就不得不创建一个类，然后写一段代码从这个类中产生一个对象。但是在 PowerBuilder 中，当创建一个类以后，我们只须做很少工作来建立那个类的实例。例如当我们在 Window painter 中用其中的一些控件(control)创建一个窗口时，我们就建立了几个类。然而，当打开这个窗口时，我们不必为这个窗口上的每个控件写实例代码，PowerBuilder 为我们处理了这些控件的实例。由于 PowerBuilder 为我们做了一些工作，所以总是分不清对象类型和对象实际上是不同的东西。

从对象类型中建立对象有几种方法。我们已经向读者展示了对窗口使用 open()函数时发生的情况。至于特定的其它标准和用户对象，应由开发人员创建这个对象(尽管只需要为 non_visual/class 用户对象创建指针变量，但是他们适用于大部分对象，如果开发人员希望对对象的内存分配进行较紧凑控制的话)。这包括两个步骤：第一步是创建一个指针变量。不象 C 或 C++，该变量是一个指针这个事实只是要知道的事情，而除此之外并不需太多关心。在 PowerBuilder 中创建一个指针变量和创建任何其它变量一样。有一个例外：不使用 String、Integer、Long 和任一标准的数据类型，这个变量的数据类型将是任何一个有效的对象类型。

这里有一个例子，程序员已经使用菜单 painter 创建了一个菜单，并用名字 m_menu1 来保存。现在就有了一个新的对象类型 m_menu1，它是从 PowerBuilder 的菜单对象类型中派生出来的。这里我们将建立一个指针变量来保存一个 m_menu1 对象：

```
m_menu1 m_mymenu
```

我们仍然没有一个对象，但是我们在内存中有一个存放对象的地方，一旦有了对象，就会有一个变量指向它。CREATE 在内存中产生这个对象，然后返回这个新对象的内存地址，它就存在指针变量中。下面是这个例子的语法：

```
m_mymenu = CREATE m_mymenu1
```

从这时起，m_mymenu 就是新对象 m_menu1 的指针。为简单起见，我们现在可以把 m_mymenu 看作一个菜单对象。读者可以用相同的语法来产生要创建的任何对象。

DESTROY 语句用于除去内存中由 CREATE 语句创建的任何对象，这个语句把有效的内存返回给操作系统。代码中不再需要这个对象时应尽快地用 DESTROY 做删除工作是一个很好的想法，这在内存要求很紧张的应用程序中尤其重要。下面是删除我们的菜单对象的语句：

```
DESTROY m_mymenu
```

要知道：一旦删除一个对象，就不能再访问它了，因为它已不再存在；不能再调用它的函数或引用它的属性，因为它们都是这个对象的组成部分。

1. PowerBuilder 中的压缩

PowerBuilder 在 PowerBar 上提供了一个 Function Painter 按钮，然而这个按钮调用的是有效的"global"函数的 painter，也就是说，这些函数对整个应用程序来说是全局范围的(scope global)。范围(scope)这个术语指明函数或变量存在时间的长短和在应用程序中具有的可访问

性,这些全局函数在别的对象中没有被压缩,它们是独立的。为进行内存和性能管理以及从OO原理中获益,(程序设计的)目标应该是最大限度地少使用全局函数。较好的实现是压缩对象中的函数。PowerBuilder让开发者能够做到这一点。那么函数被保存在适当的对象中,仅当含有压缩函数的对象存在时才占用内存。

在Window、Menu和User Objects的painter中有一名为Window Functions、Menu Functions和User Functions的选项在Declare menu上很有效。选择这个选项打开PowerBar上非常相似的Function painter,除了函数的范围不是强置成Public(例如global)。这种调用Function painter的方法允许在相应的对象中创建压缩函数。例如,从Window painter的Declare菜单中选择Window Function允许开发者仅在那个窗口中创建压缩函数。菜单对象或用户对象中的压缩函数可以用相同的方式建立。

2. PowerBuilder中的继承

PowerBuilder支持单一继承。在PowerBuilder中,继承是以不同方式完成的,这依据所继承的对象类型。例如,当打开Window painter,在New按钮上单击一下开始生成一个窗口时,将要建立的新窗口实际上就是从PowerBuilder窗口的基类继承而来。一旦建立了自己的窗口,然后想从它继承时,就在Window painter上的Inherit按钮上单击而不用在New按钮上单击。User Object painter中也有一个Inherit按钮,用于继承用户自己的用户对象。

要在一个窗口上创建一个CommandButton,只需在Window painter中的PainterBar上选择CommandButton控件(control)并在该窗口上点击一下即可。这么做时,PowerBuilder实际上从内部的CommandButton类继承了一个对象类型。对Window painter中每一个有效的控件类来说,这个过程是相似的,用户以同样的方式继承自己的用户对象控件(control)。例如,假定用户已经建立了一个用户对象CommandButton,那么就可以在Window painter中的PainterBar上的用户对象控件按钮上单击一下,然后在窗口的某处单击一下,这个用户对象就会画在这个窗口中,它继承了用户对象的CommandButton类。

PowerBuilder的继承机制

以下代码部分摘自名为w_de_student_enrollment的一个窗口类的定义,它继承自一个名为bw_de的应用程序基类。这段代码演示PowerBuilder中的子孙类怎样重用(reuse)其祖先的代码(这不是开发人员平常写的代码,我们在这里仅做示范用)。

```
on w_de_student_enrollment.create
call bw_de::create
end on
```

当用open()函数打开一个窗口时,就调用PowerBuilder中名为Create的内部事件。这段代码演示的是:当打开子孙窗口w_de_student_enrollment时,PowerBuilder首先调用内部的Create事件打开祖先窗口bw_de。继承的代码没有拷贝,而实际上子孙对象调用它,因此代码是重用的。

就连用户可以建立其上的标准 PowerBuilder 对象类型也是较高层次的 PowerBuilder 对象类型的子孙。读者可以使用 PowerBuilder 的 Class Browser(类浏览器)来查看所有对象类型的继承层次,这个重要的工具允许读者看到每个对象类型是怎样从一个祖先对象类型派生而来的(说一个对象是从另一个对象派生而来与说它是从一祖先对象继承而来是相同的)。这就象能够看得见一个对象的完整家谱一样。使用 Class Browser 的步骤如下:

(1)在 PowerBar 的 Library Painter 按钮上点一下,打开 Library painter。

(2)从 Utilities 菜单上选择 Brower Class Hierarchy。

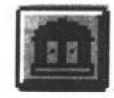

图 1-1 PowerBar 上的 Library Painter 按钮

Class Browser 浏览框右边有 4 个无线按钮,如图 1-2 所示。在四种对象类型中单击一下,可看到继承层次树形结构的流程图。在无线按钮 System 上点一下,可以看到从哪里开始。Powerobject 这个对象类型排在表的最前面,在 PowerBuilder 中它是最高层次的类或对象类型,PowerBuilder 中其它所有对象类型都是从这个对象类型派生出的。

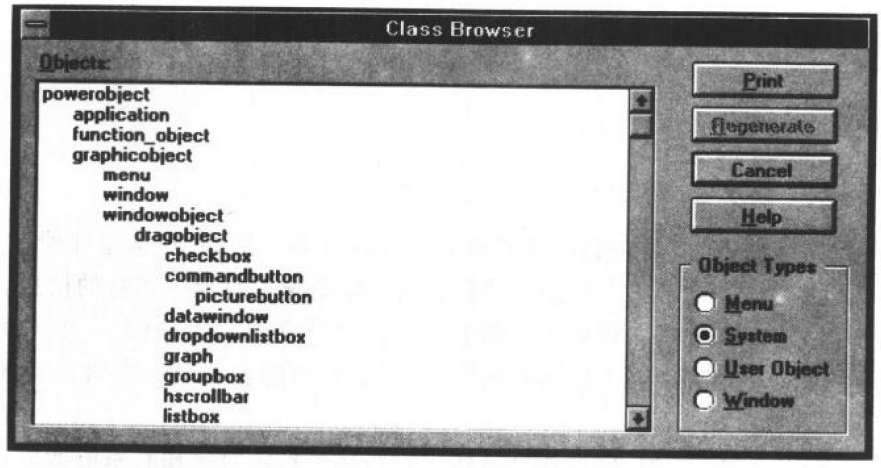

图 1-2 用 Class Browser 跟踪所有对象类型的继承层次

如果滚动整个表,就可以看到哪些对象是 powerobject 的直系子孙,它们是:

- appication
- function_object
- graphiobject
- graxis
- grdispattr
- nonvisualobject
- structure

其它所有标准的对象类型最终都是从这个表中的对象类型派生出的,如 window 对象类型是从 graphiobject 派生出的。除了 PowerBuilder 之外,用户创建的类也可在 Class Browser 中找到。