用 OWL 开发 Windows 应用程序的方法

冯　文　向永谦　（华资公司武汉自动化站）

摘要：本文以笔者创建的一个名为 FWHW.EXE 的 Window 应用程序为实例，简单概要地阐述了用 BC++3.1 的 OWL—ObjectWindow Library(面向对象的窗口库函数)技术来创建一个 Window 应用程序的方法和过程。

一、OWL 技术概述

OWL实际上是一个面向对象的类库，其类库依赖于 BC++3.1 的 Classlib 子目录中的 INCLUDE 和 LIB。它利用面向对象的方法将 Window 应用程序通常所具有的行为封装起来，包括应用程序级和窗口级的行为。主要功能为：

1.窗口信息封装

由 ObjectWindow 提供一些界面对象来代表 Window 的界面元素(如窗口、对话框、控制等)，用 Creat 成员函数使两者联系起来，即将界面元素的"句柄"作为界面对象的数据成员，这样就将窗口信息，即以上提到的三种或其他种类的界面封装在对象里，作为对象的数据成员，供应用程序调用。封装的目的在于便于用户应用程序操作。

2.对 Window API 函数进行抽象

Window API 大约有 600 个函数，为便于用户完成更高层次的任务，ObjectWindow 对这些函数进行了抽象、组装。该种方法允许用户直接调用 API 函数，同时也降低了用户应用程序对 API 函数的依赖。

3.自动消息响应

ObjectWindow 提供了自动接受 Window 消息并转化成对 BC++成员函数调用的机制，大大方便了用户应用程序对消息的处理。当用传统的方式对消息进行捕捉和响应时，要用到很多的 Switch 语句，使应用程序庞大而庸肿，而用 OWL 面向对象的方法只需要两个步骤：

(1)在类定义中构造一个虚拟的可动态派遣的成员函数，形如：

```
virtual void CMDisplay=[201];
```

(其中 201 为一整数派遣标识，可在一定范围内任选)；

(2)构造响应动作，形如：

```
WMyWindow::CMDisplay
{
```

二、创建一个 OWL 应用程序的一般过程：

1.概述：

创建一个 OWL 应用程序应包含以下六种类型的文件：

(1)资源描述文件(.rc)，它是一个普通的 Ascii 文件，定义了应用程序所使用的各种资源，如图标(.ico)、位图(.bmp)、菜单、对话框(.dlg)、字体(.fnt)、加快键等；

(2)源文件(.c 或.cpp)；

(3)头文件(.h)，该文件用来定义应用程序中使用的常量、变量和函数，可以利用 Window 操作系统和 BC++3.1 语言系统的头文件，同时也可根据需要编制用户自己的头文件；

(4)库文件(.lib)，包括 Window 函数库和 BC++3.1 语言函数库；

(5)模块定义文件(.def)，这个文件是链接程序的输入文件，定义了 Window 应用程序的名字、内存要求、代码段/数据段属性、输入输出函数等；

(6)工程文件(.prj)，BC++3.1 的工程文件是一个二进制文件，它包含了前面所述的所有类型的文件，以供编绎器和连接器使用。

由这六种类型的文件生成一个 OWL Application 的过程如图 1 所示：

下面将对以上几种主要文件的创建进行描述。

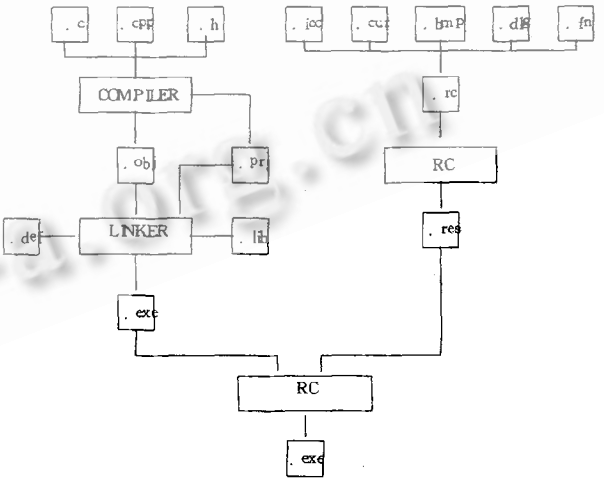

图 1

2.创建资源

创建一个 Window 应用程序几乎都不可避免地涉及到资源，所以创建资源文件就显得非常重要。Window 应用程序所涉及的资源主要有这样几种：图标、位图、字体、对话框、菜单和加快键等。创建资源就是要编写出一个资源描述文件(.rc)，使其包含对上述资源的定义，通过 RC(资源编绎器)的两步工

作将这些资源加入到 Window Application 中去。这两步工作是:首先用 RC -R<资源描述文件>将.RC 文件编绎成二进制文件.res;然后再用 RC 将.res 文件加入到 LINK 连接器生成的.exe 中使之形成真正可执行的 Window Application,如图 1 所示。

如前所述,创建资源就是要去编写一个资源描述文件,具体来说,就是要用一些单行语句或语句组去描述上述资源,从而形成一个.rc 文件。其中资源可被定义为预先装入/调用时装入,在内存中固定/可移动/可废弃等方式。

(1)定义图标(.ico)、位图(.bmp)和字体(.fnt)资源。用 BC++3.1 中所提供的 Resource Workshop 工具去生成一个.ico,.bmp 和.fnt 文件。为应用其资源,必须要在.h 文件中定义和在.rc 文件中定义。以应用图标资源为例:

在.h 中应写上: # define IDXXICON 100(其中 IDXXICON 为图标资源标识符)

在.rc 中应写上单行语句:IDXXICON ICON LOADONCALL MOVEABLE

同样道理,在应用位图和字体资源时也应作类似定义。

(2)定义对话框资源(.dlg)。以 FWHW.EXE 实例中的一个输入对话框 InputDialog 为例说明,其形式如图 2 所示:

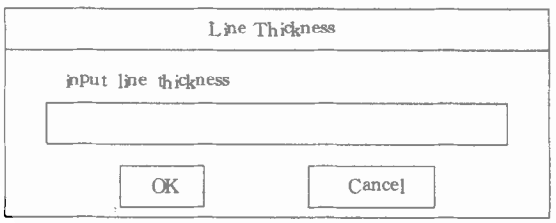

图 2

定义描述语句组形式如下:

```
SD-INPUTDIALOG DIALOG DISCARDABLE LOADON
        CALL PUREMOVEABLE 20,24,180,64
        STYLE WS-POPUP | WS-CAPTION |
         DS-SETFONT FONT 8, "Helv"
BEGIN
  CONTROL "" ID-PROMPT,"STATIC", WS-CHILD | WS-
VISIBLE | WS-GROUP, 10, 8, 160, 10
  CONTROL "" ID-INPUT, "EDIT", WS-CHILD | WS-VISI-
BLE | WS-BORDER | WS-TABSTOP | ES-AUTOHSCROLL, 10,
20, 160, 12
  CONTROL "&OK" IDOK, "BUTTON", WS-CHILD | WS-
VISIBLE | WS-TABSTOP | BS-DEFPUSHBUTTON, 47, 42, 40,
14
  CONTROL "&Cancel" IDCANCEL, "BUTTON", WS-
CHILD | WS-VISIBLE | WS-TABSTOP, 93, 42, 40, 14
END
```

不同的对话类对应着不同的对话形式,如输入对话和文件对话描述形式不尽相同。对话框语句组可定义对话框的名字、装载及存储方式选项、框的尺寸、位置及各控制项的名称等等。

(3)定义菜单资源。以 FWHW.EXE 中的主菜单 FWCOMMANDS 为例说明,其定义描述形式如下:

```
FWCOMMANDS MENU LOADONCALL MOVEABLE
PURE DISCARDABLEBEGIN
  POPUP "&Display"
BEGIN
  MenuItem "&Clear          F3", CM-CLEARMenuItem
  SEPARATOR
  MenuItem "&Display", CM-DISPLAY1
  MenuItem "Input&Box", CM-INPUTBOX
    MenuItem "&OpenFileDialog", CM-OPENFILEDIAL
    MenuItem "&SaveFileDialog", CM-SAVEFILEDIAL
    MenuItem "&Quit", CM-QUITM
 END
   POPUP "&Help"
   BEGIN
  MenuItem "Help &Text        F2", CM-HELP1
    MenuItem "&Beep", CM-HELP2
    MENUITEM "Help&3", CM-HELP3, GRAYED
    MenuItem SEPARATOR
    POPUP "&Windows"
    BEGIN
   MenuItem "Window&1", CM-WINDOW1
     MenuItem "Window&2", CM-WINDOW2,
   GRAYED
     MenuItem "Window&3", CM-WINDOW3,
   GRAYED
   END
 END
END
```

菜单语句组可定义菜单的外型和功能,菜单外型有条状和弹出型两种,对菜单项可以进行分组,对单个菜单项还可以定义其初始状态为黑、灰或选中等。

(4)定义加快键资源。以 FWHW.EXE 中的 FWACC 加快键定义为例,其描述形式为:

```
    FWACC ACCELERATORS
    BEGIN
      VK-F2, CM-HELP1, VIRTKEY
      VK-F3, CM-CLEAR, VIRTKEY
  END
```

定义 F2、F3 为加快键。

(5)模块定义文件(.def)。以 FWHW.EXE 中的 OWL.DEF

为例，其描述形式为：

```
EXETYPE WINDOWS
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 5120
```

模块定义文件是一个普通文本文件(如上例)，它定义了应用程序的名字、内存的需求、代码/数据段属性以及输入输出函数列表等，作为链接程序的输入文件。其主要语句有：NAME, SEGMENTS, EXPORTS, IMPORTS, HEAPSIZE, STACKSIZE等。

任意一个 Window Application 都离不开模块定义文件.def，虽然 BC++3.1 可以在建立应用程序时利用内部缺省的.def 文件，但当用户的应用程序不满足缺省条件时就必须写出相应的.def 文件。

(6)创建工程文件。将所有类型的文件，如.cpp,.rc,.def等连成一体并提供给编绎器和连接器使用这一工作是由.prj工程文件来完成的。它与 Turbo C 的文本工程文件不一样，它是非文本的工程文件，即二进制文件，其中比较重要的是它的 Location 项目指示，它指示出某一文件的路径，当该文件目录改变时，必须用 BC++3.1 的 Project 项重新创建这一文件到工程文件中去。

另外值得注意的是编绎和连接的头文件(.h)和库文件(.lib)的路径或称目录问题。OWL Application 主要涉及这样三个目录的.h 和.lib，即：

```
[C:]\BC\OWL\INCLUDE(LIB);
[C:]\BC\INCLUDE(LIB);
[C:]\BC\CLASSLIB\INCLUDE(LIB)
```

3. 创建 OWL 源程序的设计逻辑

一个 ObjectWindow 应用程序至少应有两个类和一个程序入口。两个类是：应用程序类和主窗口类，程序入口是：WinMain()函数。

用户自定义的应用程序类是由 TApplication 派生出来的，由该应用程序类通过进入 Window 程序入口，即进入 WinMain()函数中而构造出应用程序对象，该对象就作为应用程序自身面向对象的一个"替身"。用户还必须在应用程序类中定义一个虚拟的初始化主窗口的成员函数 InitMainWindow()，该函数通过调用 TWindow 类或其派生类的实例来构造一个主窗口对象，将它存于应用程序对象的 MainWindow 数据成员中。

另外一个类就是主窗口类，它是由 TWindow 类或其派生类派生出来的，由该主窗口类的构造函数构造出应用程序的主窗口对象，再在主窗口类下定义出功能各异的成员函数来完成主窗口上的所有操作，如加设菜单，进行对话，生成子窗口等等。如果要增加子窗口的功能，也可以象构造主窗口对象一样构造出子窗口类，由子窗口类构造函数构造子窗口对象，并在该类下定义成员函数来完成子窗口上的所有窗口功能。这充分体现了窗口系统以及面向对象程序设计(OOP)的意义，所有操作都是以窗口为对象，以消息(Message)机制来通讯响应的。

一个 Window 应用程序的运行是在定义了应用程序类和主窗口类后调用应用程序对象的 Run 成员函数(如 MyApp.Run();)，完成对消息机制的循环处理，直至应用程序的终结，最后应用程序返回一个状态值 Status(MyApp.Status)给 Window，从而结束应用程序的运行。

一个功能丰富的 ObjectWindow 程序除了前面提到的两个类之外，还有着其他派生类，用户可以由这些派生类再派生出自己的窗口类，完成特定功能的窗口封装，这种封装就包括对数据成员的定义，对基类成员函数的重定义，构造对菜单命令的响应成员函数，构造对消息的响应成员函数以及对其他应用程序(即.cpp)的调用方式等等。要注意的是，不同类的派生类(如 TWindow 和 TMDIFrame 的派生类)其封装格式不同，比如菜单生成格式，消息参数调用格式等。举例来说，如果：

```
class TMyWindow1:public TWindow
class TMyWindow2:public TMDIFrame
```

那么生成菜单的时候，对 TMyWindow1 来说，其构造窗口菜单的形式为：

```
TMyWindow1::TMyWindow1(PTWindowObject AParent, LPSTR ATitle)
        :TWindow(AParent, ATitle)
{
  AssignMenu("FWCOMMANDS");
  ...
}
```

而对于 TMyWindow2，其构造窗口菜单的形式为：

```
TMyWindow2::TMyWindow2(LPSTR ATitle, HMENU Amenu)
        :TMDIFrame(ATitle, Amenu)
{
...
}
```

三、结束语

由于 OWL 有信息封装，函数抽象及消息的自动响应等功能，使用户在开发 Windows 应用程序时省却了不少麻烦，只要我们清楚用 OWL 开发 Windows 应用程序的设计逻辑及步骤，就可以快速而有效地开发出功能强大的 Windows 应用程序。