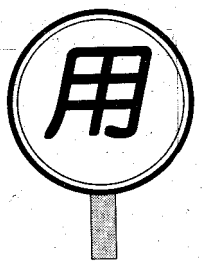

用C++开发 WINDOWS 3.0 的应用程序

张兴滔 （四川红泉仪表厂）

摘要：本文介绍了如何通过 *Borland C++*开发 *WINDOWS 3.0* 的应用程序，同时说明了新一代面向对象的程序设计方法 *OOP(Object-Oriented Programming)*的基本特性和实现方法。

在 C++中对 WINDOWS 应用程序的开发和设计，实际上就是 OOP 程序设计方法的充分体现，下面谈谈它的实现过程和 OOP 的一些基本特性和基本概念。

一、WINDOWS 应用程序开发的软环境

在 WINDOWS 环境下，有应用程序和非应用程序之分，但是 Boriand C++本身不是 WINDOWS 的应用程序，它只能在 DOS 命令行运行或作为 WINDOWS 的非应用程序运行，如果用户的硬件配置（存储空间、时钟频率等）不太好，笔者建议最好先在 C++集成环境程序员平台下编辑 WINDOWS 应用程序的源文件，经编译、链接成功后，存储源文件退出 C++，再在 WINDOWS 环境下运行刚开发的 WINDOWS 应用程序，这就可以减少许多 C++作为 WINDOWS 的非应用程序运行时，开发软件所带来的麻烦，如重新编译当前已装入的程序或 DLL 事例（Inatance），或应用程序调试时，总可能会因为程序中的致命错误或一般保护性错误忽略考虑，而导致系统崩溃，应用程序源文件未写盘而丢失等不必要的麻烦。

二、WINDOWS 与其应用程序的工作原理及在 C++上的实现

WINDOWS 应用程序是一个以事件驱动、信息传递、以 WINDOW 对象为组织单位的系统，在 C++中可看作是一个基于对类实例（对象）组织设计编程的系统。

1.WINDOWS 与 OOP 及其实现

在 WINDOWS 环境下进行系统开发设计的主要步骤就是构造一系列图视窗口（即窗口数据），以及实现对这些窗口进行各种操作的函数（即窗口函数）的设计。在 C++中 OOP 编程方法的封装性（Encapsulation）正好完成上述任务，比传统程序设计方法实现起来更加优越；在这一系列窗口的构造中，包括主窗口（数据和函数）、子窗口（数据和函数）等，它们之间的语义（指成员函数和成员数据）可以通过 OOP 的继承性（Inheritance）实现，另外 OOP 中的多态性保证了不同窗口使用同名的窗口函数不会引起混乱。这使得用户开发 WINDOWS 应用程序成为可能。

2.WINDOWS 应用程序的构成及功能

WINDOWS 应用程序的主控部分非常简单，它主要完成三项工作：

(1)定义窗口类并且进行初始化登记；

(2)创建、显示、更新窗口；

(3)进行信息处理循环。

可以简单地说，应用程序主控部分的任务就是构造所需的窗口并且提供给 WINDOWS 进行管理，WINDOWS 又为每个窗口调用其窗口函数，处理输入/输出以及其它事件，而这一切都是通过表达不同意思的信息在两者之间传递的结果，因而程序设计主体在于对信息处理过程的设计。

3.程序主体设计技术

WINDOWS 支持一系列属性，当用户选择了适当的属性，就可构成所需要的窗口，而窗口函数的处理方式则带来了一种与传统程序设计方法（如分支、循环，调用等）不同的风格，取而代之的是窗口函数对信息（事例）处理的多路开关 式流程。WINDOWS 对每一个窗口的一次信息传递，就会告诉窗口有一个事例发生并以信息方式传过来了；窗口函数就可以根据信息代表的不同意思进行处理，这同样适用于主窗口与子窗口之间；另一方

面这样的信息传输方式在多个事件构成一个操作的情况下，就会给窗口函数带来破坏，这样就必须引入大量控制变量来协调各个事件之间的关系，从而带来了窗口函数的复杂结构。由上可见，WINDOWS 应用程序设计的主要工作就是窗口和窗口函数的设计，而关键之处在于窗口函数内部对不同信息进行处理的结构安排上。后面举了一个简单的 WINDOWS 应用程序的设计例子，来说明设计一个完整的应用程序的步骤以及 OOP 程序设计特性在其中的体现，下面首先谈谈窗口及其函数处理。

三、窗口及窗口函数的构造方法

在 WINDOWS 环境下，应用程序的组织实际上就是对窗口及其窗口函数的组织，主要实现构造相应窗口及编制相应窗口函数，在 C++中用户可以通过 WINDOW 类(在 WINDOWS.H 中定义)加以实现，下面简单说明一下它们的定义及构成。

1.主窗口

登记主窗口类的代码如下：

```
static void Register(void)
{
    WNDCLASS  wndclass; / / 定义WNDCLASS的实例
                        来登记主窗口
                        / / 下面是对主窗口的属性进
                        行设置
    wndclass. sytle = CS-HREDRAW / CS-VREDRAW;
    wndclass. lpfnWndPrcc = ::WndProc;
    wndclass. cbClsExtra = 0;
    wndclass. cbWndExtra = sizeof(Main Window * );
    wndclass. hInstance = Main::hInstance;
    wndclass. hIcon = LoadIcon(Main::hInstance , "MyIcon
");
    wndclass.  hCursor = Loadcursor( NULL, IDC -
ARROW);
    wndclass. hbrBackground- GetStockObject ( WHITE-
BRUSH);
    wndclass. lpszMenuName = szClassName;
if (! RegisterClass( wndclass)) / / 注册登记失败则退出
      exit(FALSE);
}
```

WINDOWS 需要知道在你的应用程序中所有不同类型的窗口类信息，这样你就需要把这些信息填入到一个 WNDCLASS 的结构(在 WINDOW. H 中定义)中并通过 RegisterClass 来注册登记，你就只需要为每一个 WINDOWS 类登记一次，特别地，同一程序的以前事例已向 WINDOWS 登记了此窗口类，那么就可以不再登记了。

WINDOWS 将根据用户在窗口类中指定的窗口信息来建立一个窗口，在窗口类中不仅定义了窗口的显示特性，而且还命名了窗口名字并指定了窗口函数，WINDOWS 必须调用它来给窗口(应用程序)传递信息；因为一个应用程序中的各个事例可以使用主窗口的相同定义(OOP 中的继承性)，因而每一个窗口类只需要定义一次(虽然一个应用程序可以有多个窗口类)。

2.窗口函数

在上面登记主窗口时，WndProc 为窗口函数，它主要控制处理下列 WINDOWS 发送的事例：

(1)窗口创建事例

(2)窗口显示事例

(3)窗口更新事例

(4)窗口撤消事例

(5)窗口菜单选择事例

3.菜单资源的定义及使用方法

主窗口中有一个带有名为：MyMenu"的菜单条菜单，它是用来实现在主窗口建立时，在菜单条中建立一个由菜单资源描述文件中描述的信息的菜单，在主窗口函数中有处理对菜单选择的控制，在处理菜单项的选择时，要遇到一系列对话框，它们也是资源文件描述定义的。菜单资源文件建立及描述如下：

```
#include < windows.h >
SIMPLEPAINT  MENU        / / 定义菜单名
BEGIN
MENUITEM"&Quit!",100        / / 菜单 1-1
popup"&Shape"
    BEGIN
      MENUITEM"&Line",201        / / 菜单 2-1
      MENUITEM"&Ellipse",202        / / 菜单 2-2
      MENUITEM"&Rectangle",203        / / 菜单 2-3
    END
POPUP"&Pen"
    BEGIN
      POPUP"&Thickness"
        BEGIN
          MENUITEM"&Thin",301        / / 菜单 3-1-1
```

```
        MENUITEM"&Regular",302  //菜单 3-1-2
        MENUITEM"T&hick",303  //菜单 3-1-3
      END
    POPUP"&Color
      BEGIN
        MENUITEM"&Red",304        //菜单 3-2-1
        MENUITEM"&Green",305  //菜单 3-2-2
        MENUITEM"&Black",306  //菜单 3-2-3
      END
    END
END
MyIcon ICON MyIcon.ico        //图符描述
```

4.图符(ICON)资源的建立和使用方法

在上面登记主窗口和菜单资源描述中都有一个MYICON量，它表示该应用程序的图符，是由WINDOWS的SDK建立的;要使用它只需在登记主窗口时,象以上那样表明,而且要在菜单所在的资源文件中正确描述.

四、工程文件管理编译和链接WINDOWS应用程序

C++集成环境程序员平台的优越性，为WINDOWS应用程序的开发提供了极大方便,但它从结构设计、模型构造等方面又不同于其它程序设计。而且开发一个WINDOWS应用程序往往需要多个源文件(头文件、资源文件等等)这就使得多个相互联系的文件难于管理,而C++的工程管理文件正好解决了用户开发WINDOWS应用程序时,多个文件难于管理的麻烦,建立一个WINDOWS应用程序的工程管理文件方法如下:

(1)在C++的程序员平台分别编辑所需的C++源文件(.CPP)、头文件(.H)、模块定义文件(.DEF)、以及资源文件(.RC)等几个文件,并把它们放在同一个目录中;

(2)打开工程文件管理对话框,建立一个新的工程文件,把以上几种文件一起放到这个工程文件中;

(3)打开工程文件后,只有在选择了正确的编译和链接选择项后,才能通过Build All命令,对工程文件中给出的相关源文件进行编译和链接,产生正确的应用程序,它包括许多选择项:

.在硬件没有配置协处理器时,WINDOWS应用程序应使用浮点仿真数学库链接。

.选择编译、链接最终文件生成形式为.DLL或.EXE。

.选择自动依赖检查(Auto-Dependency),以便每次所需的多个源文件都能重新编译。

(4)系统头文件、库函数的路径必须加以说明(使编译器和链接系统能找到相应的Include、lib、classlib文件)。

(5)WINDOWS.H头文件必须放在源文件的开始处。

五、具体例子

下面举个完整的例子,该程序在AST 386SX/16上通过,其源程序如下:

```
#include<windows.h>        //头文件的定义
#include<stdlib.h>
#include<string.h>
long FAR PASCAL -export WndProc(HWND hWnd,
WORD iMessage,WORD wParam,LONG lParam);
class Main                    //定义 Main 类并进行初始化
{
    public:
      static HANDLE hInstance;
      static HANDLE hPrevInstance;
      static int ncmdShow;
      static int MessageLoop(void);
}
HANDLE Main::hInstance=0;     //初始化
HANDLE Main::hPrevInstance=0;
int main::nCmdShow=0;
int Main::MessageLoop(void)     //与 WINDOWS 之间信息传递函数
{
      MSG mag;
      while(GetMessage(&mag,NULL,0,0))
      {
        TranslateMessage(& mag);
        DispatchMessage(& mag);
      }
      return mag.wParam;
}
class Window        //基类 window 的定义
```

```
{
    protected:
      HWND hWnd;
    public:              //定义窗口类的公有函数
      HWND GetHandle(void){return hWnd;}
      BOOL Show(int nCmdShow){return Show
Window(hWnd,nCmdShow);}
      void Updats(void){UpdateWindow(hWnd);}
      virtual long wndProc(WORD iMessage,WORD
wParam,LONG lParam)=0;
};
class Main Window;public window //派生类的定义
{
    private:
      static char szClassName[14];
    static void FAR PASCAL Linefunc(int X,int
Y,LPSTR ipData);
    public:
      static void register(void)    //登记主窗口
        {
          WNDCLASS wndclass;//定义WNDCLASS类
                                的实例
                wndclass.sytfe
=CS-HREDRAW/CS-VREDRAW;
          wndclass.lpfnWndProc=::Wndproc;
          wndclass.cbclsExtra=0;
          wndclass.cbWndExtra=sizeof(MainWindow*);
          wndclass.hInstance=Main::hInstance;
          wndclass.hIcon=loadIcon(Main::hInstance;
          wndclass.hCursor=Loadcursor(NULL,IDC-AR
ROW);
          wndclass.hbrBackground=GetStockObject(WHI
TE-BRUSH);
          wndclass.lpszClassName=szClassName;
          if(!RegisterClass(&wndclass)){        //注册
                                                失败则退
                                                出
          exit(FALSE);}
      }
}
Mainwindow(void)      //mainWindow的构造函数
      {                    //创建窗口
       hWnd=createWindow(szClassName,
          szClassName,
          WS-OVERLAPPEDWINDOW,
          CW-USEDEFAULT.
          0,
          CW-USEDEFAULT,
          0,
          NULL,
          NULL,
          Main::hInsdtance,
          (LPSTRthis);
       if(!hwnd)          //创建窗口失败就退出
         exit(FALSE);
       Show(Main::nCmdShow);    //显示窗口
       Update();          //更新窗口
      }
long WndProc(WORD iMessage,WORD wParam,LONG
lParam);
void paint(void);
struct LINEFUNCDATA
    {
    HDC hDC;
    char FAR *LineMessage;
    int MessageLegntgh;
    LINEFUNCDATA(char *Message)//定义构造函数
      {
      hDC=0;
      MessageLength=strlen(Message);//取字符串长度
      LineMessage=new char far [MessageLength+1;
      lstrcpy(LineMessage,Message);
    };
LINEFUNCDATA(void){delete LineMessage;}//定义析
                                        构
                                        函数
    };
};
char MainWindow::szClassName[]*"Hello,WINDOWS
    V3.0!";  //定义串量
void mainWindow::paint(void)
{
    PAINTSTRUCT ps;
    RECT rect;
    FARPROC ipLinefune;
    LINEFUNCDATA LinefuncData("Welcome!");//初
                                          始化
    lpLineFunc=MakeProcInstance(    (    FARPROC)
MainWindow:: LineFunc, Main:: hInstance);
BeginPaint (hWnd, pa);
GetClientRect (hWnd, (LPRECT)rect);
    Linefunc Data. hDC, TA-BOTTOM);
    SetTextAlign(ps.hdc,TA-BOTTOM);
```

```
    LineDDA(0,0,0, rect. bottom / 2,lpLinefunc, (LPSTR)&
LinefuncData);
    EndPaint (hWnd, & ps);
    FreeProc Instance (lpLineFunc);
}
void FAR PASCAL Main window:: LlineFunc(int X,int Y,
LPSTR lpData)
{
    LINEFUNCDATA FAR * lpLinefuncDate=
(LINEFUNC DATA FAR * )lpDate;
    TextOut (lpLinefuncData->hDC,X,Y,
        IpLineFuncData->LineMessage,
lpLineFuncData-> MessageLength);
}
long Mainwindow:: WndProc(WORD iMessage,WORD
wParam,LONG IParam)
{
    switch(imessage) / / 信息处理过程
    {
      case WM-CREATE:
          break;
      case WM-PAINT:
          Paint();
          brak;
      case WM-DESTROY:
          PostQuitMessage(0);
          break;
      default:
          return DefWindowProc (hWnd,iMessage,
wParam, iParam) ;
    }
}
long FAR PASCAL-export WndProc(HWND
hWnd,WORD iMessage, WORD wParam
                    ,LONG IParam)
{
    if (iMessage=WM-CREATE)
    {
      LPCREATESTRUCT lpcs;
      lpcs= (LPCREATESTRUCT) LParam;
      return MainWindow:: WndProc(iMessage, wParam,
          IParam);
    }
    else
      return DefWindowProc(hWnd, iMessage, wParam,
          LParam);
}
#pragma argsused      / / 关闭形参未使用警告
int PASCAL WinMain( HANDLE hinstance, HANDLE
hPrevInstance, LPSTRLPszCmdLine, int ncmdShow)
{
    Main:: hInstance=hInstance;
    Main:: hPrevInstance=hPrevInstance;
    Main:: nCmdShow=nCmdShow;
    if (! Main:: hPrevInstance){
      MainWindow:: Register();
    }
    Main Window MainWnd; / / 定义 MainWindow 的实例
    return Main:: MessageLoop();
}
```

其图符资源源文件描述如下:

```
My Icon ICON MyIcon. ico
```

其模块定义源文件如下:

```
NAME          MyWindow
DESCRTPTION   'My Windows Application Test'
EXETYPE       WINDOWS
CODE          PRELOAD MOVERABLE
DATA          PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      1024
STACKSTZE     5120
```