第一章 ObjectWindows 概述

ObjectWindows 是一个面向对象的应用程序框架，它封装了 Windows 界面对象的通用行为，简化了 Windows 用户界面开发的复杂性，使程序员可以将主要精力用于解决程序特有的要求方面，这不但能加快程序开发的速度，而且也能提高用户界面的质量。本章介绍用户界面的特点和组成元素，以及 ObjectWindows 基本框架，并给出一个程序来说明 ObjectWindows 应用程序的特点。

1.1 图形用户界面

用户界面是这样一类软件系统或软件工具，它用于帮助实现计算机系统、应用软件系统和使用系统的人即用户三者之间的恰当联结和协调一致。换句话说，用户界面是专门处理人机交互活动的软件。图形用户界面通过使用窗口、选单(又称菜单)、图标和对话框等组件来建立用户界面，并允许用户使用像鼠标器等点输入设备对屏幕上显示的各种对象进行直接操作。

所谓物理隐喻(又称桌面隐喻)的概念开创了面向对象的用户界面的研究，结果使计算机更接近最终用户的自然环境。物理隐喻是指一个计算机系统所显示的可视图像应以某种程度的抽象显现出真实物理对象的映像。桌面隐喻对指导用户界面的开发有很大的影响。使用这种方法，计算机所表现的信息和对象与它们在真实世界中的表像和行为具有某种相似性。例如，在 Windows 用户界面中，我们常使用图标来符号化公用于不同情况的词汇或概念。如在某些绘图软件中，使用图标代表各种绘图工具，每个图标表示一种特定类型的绘制行为，选择某种类型的图标，就可以进行某种类型的绘图操作。图标还用于表示像文件夹、打印机等概念。在使用了桌面隐喻和引入了直接操作特性以后，最终用户的数据和程序就不再是分离的实体、而或为表示最终用户信息的屏幕对象，这些对象封装了数据和操作数据的过程，用户直接操作屏幕对象来操作程序。

除图标外，控制也是 Windows 用户界面中的一个重要构件。控制多用于模拟真实世界中的设备控制面板中的控制。使用控制，用户界面就可以在屏幕上构筑各种物理设备的虚似控制面板，使用户在屏幕上可以像操作物理设备那样操作软件系统。这样，面向对象的用户界面就为用户提供了一个非常类似于真实世界的易于操作的计算机环境，加强了应用程序的真实性。

面向对象的用户界面构件中另一类主要对象是选单和对话框。选单的使用可以减少用户记忆操作命令的负担。代替记忆各种命令，用户从选单中选择一个命令。当选单中的一个选项被选中后，相应的命令就被执行。对话框则允许用户和计算机之间进行更复杂的交互。对话框是由像按钮、滚动杠、列表框和编辑框等各类控制对象组成的组合对象。

面向对象的用户界面的另一个重要特点是，相同的操作可以用于不同类型的对象上，例如将 Cut、Copy、Paste 和 Delete 等这种通用命令用于一个单词、一段文章或一个图形上，这也减少或简化了用户为执行操作所需记忆的命令。

由于图形用户界面上的对象模拟着现实环境中的真实对象，因而面向对象的图形用户界

面易于为用户理解和使用。为支持直接操作特性，像鼠标这样的点输入设备就成为面向对象的用户界面的重要输入工具，用户可以使用鼠标启动应用程序，或将对象从一个地方移动到另一个地方。在面向对象的用户界面中，复杂的任务可以使用简单的操作来完成。

面向对象的用户界面也使用户界面的开发人员受益。面向对象方法的重要意义是改变了程序员看待软件系统的方式，即程序中数据和代码应归属于某个对象，对象具有封装性、自治性和通信性。封装隐藏了对象的内部实现，外界或说对象的用户只能从对象的行为来认识对象，这些行为由为对象上定义的公有操作接口集合来说明。

由于对象的封装性和自治性，一个界面对象可以用于许多应用程序中。由于一个对象可以与其它对象进行通信，这样，从简单的界面对象组装复杂的界面对象就成为一个较简单的过程。复杂对象可以像简单对象那样，用于装配更复杂的对象，Windows 中的对话框就是这方面的一个典型的例子。但程序员要真正从中受益，必须使用面向对象的开发技术，面向对象的软件系统的一个最大的优势就来源于程序中的对象以及这些对象之间的关系与屏幕对象具有比较直接的对应，程序对象的行为决定着屏幕对象的行为，屏幕对象是程序对象的可视化结果。程序对象是对现实环境中的真实对象的模拟，这些真实对象使用桌面隐喻的概念被直接表达为屏幕对象。

使用面向对象的开发技术，常用对象的类可以放在程序库中供在设计各种用户界面时再用，类继承和多态性保证了在再用这些对象的类时，可以根据需要对它们进行裁剪，以满足特定场合的需要。ObjectWindows 框架就是为此目的而开发的。

现今用户界面的新概念(桌面隐喻和直接操作)需要开发复杂的软件来显示屏幕对象和处理与这些对象进行交互时有关的事件，像 ObjectWindows 这样的用户开发环境提供了对象的类的层次构成的库，用于简化用户界面的设计和开发。层次中的每个类为它的对象定义了所必须的结构和行为，并从父类继承结构和行为，另外，子类可以覆盖、细化或扩充结构和行为。界面的设计者可以通过增加自己的屏幕对象设计来进一步扩展类层次。新的屏幕对象的类能够继承已有类的特征，另外，还可以定义新的特征或细化已有的特征。

这样，开发用户界面的复杂性大为降低，并且所开发的用户界面易于被用户使用，因为对象的再用意味着对象行为的再用，因此，对用户而言，会使用一个应用程序也就意味着可以很快地掌握使用新的应用程序。

1.2 ObjectWindows 框架

框架是专为开发特定类型的应用程序而设计的类库，ObjectWindows 是 Borland 公司为方便 C++程序员开发 Windows 用户界面而设计的一个 C++类库，称为应用程序框架(AF——Application Framework)。它提供了构造用户界面的基本组件，使用这个框架，可以简化程序员开发用户界面的工作，使其主要精力专注于应用程序所完成的任务的设计上。

ObjectWindows 用对象来表示 Windows 应用程序中那些非常复杂的元素，这些对象的类示于图 1－1 中，每个类定义了所描述的对象的行为，其中最主要的类是 TModule 类和 TWindowsObject 类。TWindowsObject 类是一个抽象类，它提供了对窗口数据管理和处理常见的窗口消息。由 TWindowsObject 类派生出 Windows 用户界面中最主要的两类窗口对象，即所谓的对话框(TDialog 类)和窗口(TWindow 类)。在 Windows 中，MDI 窗口和控制都是特殊

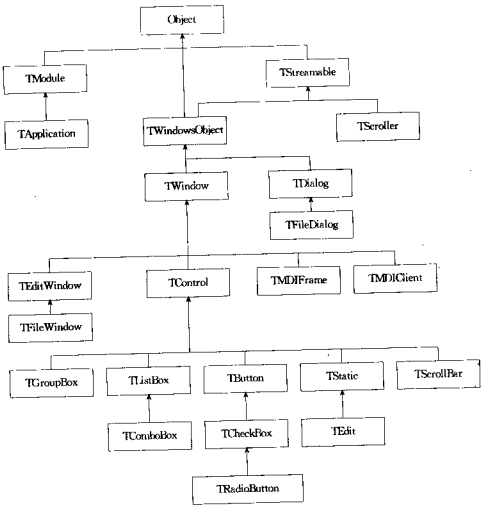

图1－1 ObjectWindows的继承层次结构

的窗口对象，因而它们都是 TWindow 类的子类(或称派生类)。TModule 类定义了动态链接库模块和应用程序模块的共同行为，由 TModule 类派生的 TApplication 类则定义了应用程序模块特有的行为，即消息处理。

ObjectWindows 框架提供了一个 Windows 应用程序所必须具有的最基本的特征，即含有一个窗口。下面这个程序用于说明这点。

```
#include <owl.h>
int PASCAL WinMain(
HINSTANCE hInstance,
HINSNTANCE hPrevInstance,
LPSTR lpszCmdLine,
int nCmdShow )
{
```

```
    TApplication MyApp("Welcome to Windows",hInstance,hPrevInstance,
                       lpszCmdLine,nCmdShow);
    MyApp.Run();
    return MyApp.Status;

}
```

在每个 ObjectWindows 应用程序中都必须嵌入文件 owl.h。

图 1－2 ObjectWindows 程序的基本特征

图 1－2 是这个程序的运行结果。在第二章，我们将介绍与这个程序有关的知识。

1.3 模块和实例

在 Windows 中，模块是指任何能装入内存的可执行代码和数据的集合。一个模块或是一个应用程序，或是一个动态链接库。

当一个模块被装入内存之后，Windows 在内存中建立该模块的一个实例，每个实例使用句柄作标识。对于应用程序模块，在内存中可以有多个实例同时运行；一个动态链接库模块在内存中只能有一个实例存在。

在 Windows 中，每个模块都附带有一些资源，这些资源是一些数据集合，它们为程序中所要建立的对象（例如选单、对话框、光标、图标和位图以及其它类型的对象）提供数据。一个模块的实例句柄决定了当程序中的某个对象需要使用资源时，应从哪个模块中装入所需的资源。Windows 面向对象的特征主要是建立在窗口对象上。窗口对象是各种资源的使用者，在程序中创建窗口对象时，都需要一个模块实例句柄。另外，模块实例句柄也标识了哪个模块的实例拥有所创建的窗口对象，以便在该模块实例中为所创建窗口对象保存有关数据，或在窗口对象被删除时清除有关数据。模块的实例句柄决定着窗口对象的类（即 WNDCLASS 结构）和窗口对象的拥有者，因此，在创建 ObjectWindows 的每个类的对象时，都需要一个指向 TModule 类的对象的指针。

应用程序模块的实例句柄由 Windows 执行应用程序时传给 WinMain 函数。WinMain 函数带有四参数，第一个参数是正执行的应用程序实例的句柄；第二个参数是该应用程序模块前

一个实例的句柄。如果所执行的实例是应用程序模块的第一个实例，则该参数的值为 NULL；第三个参数是命令行字符串；第四个参数指示程序以何种方式显示其主窗口。

Windows 要求在应用程序实例终止时向 Windows 返回一个状态码，TApplication 类的数据成员 Status 保存有程序的运行状态，这个状态在 WinMain 中使用 return 语句返回给操作系统。

1.4 对象标识

对象标识是一个对象所固有的特征，它将该对象与其它对象区别开来。使用对象标识，对象可以包含或引用其它对象，对象标识将一个面向对象的程序所处理的对象空间中的对象有机地组织起来。继承则用于组织对象的类。

使用对象标识，程序员可以动态地构造任意图结构的复杂对象，即使用子对象构造的对象。由于有了对象标识，才有可能使同一个对象用作多个对象的子对象。对象标识也使对象可以在运行时被动态创建和删除。

由上述对象标识定义可知，标识是与机器或实现无关而与对象相关的一个概念。但 C++ 语言将标识简单地和内存地址等同起来，即使用指针或对内存地址的引用来标识一个对象。我们在这里不讨论这种标识技术的利和弊，而是讨论在使用 ObjectWindows 设计程序时，这种标识技术对程序设计的影响。

在 Windows 中存在着各种对象，例如应用程序实例、窗口对象和选单对象等。在 Windows 中，每个对象使用句柄来标识，对象的标识是在对象被创建时，由系统赋给每一个对象的，并且在整个系统中具有唯一性，这种对象标识技术的好处在于，即使对象被移动，对象的标识也不变。句柄是对 Windows 管理的一张内部表的索引，在表中存储着与对象有关的数据，如图 1－3 所示。在程序系统中，句柄被存储在一个变量中，在图 1－3 中，句柄被存储在一个 C++ 对象的数据成员 HWindow 中。

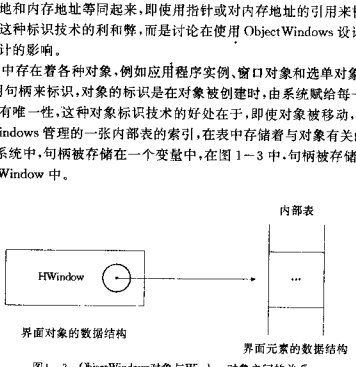

图 1－3 ObjectWindows对象与Windows对象之间的关系

C++ 使用变量名来标识一个对象，变量用来模拟客观世界里具有状态的对象，它代表内存中的一块连续的存储单元。因此，当一个 C++ 应用程序在 Windows 上运行时，面临两种标识对象的方法。使用 C++ 标识对象的方法时，我们用下述表达式来表示对对象所要施加的操作：

AWindowsObject.Create(...)

或

```
PointerToAWindowsObject->Create(...)
```

但当使用 Windows 系统标识对象的方法时，对这个对象的操作只能采用这种形式：

```
MoveWindow( AWindowsObject.HWindow, ... )
```

或

```
MoveWindow( PointerToAWindowsObject->HWindow, ... )
```

这是由于对象标识方法不同，访问对象的方法也就不一样。

ObjectWindows 除提供有构造应用程序的基本框架结构和对窗口消息的自动响应之外，并没有完整地提供对窗口对象进行处理所需的全部操作。对窗口的操作要使用 Windows 的 API 函数来进行，这样，对同一个可视对象，就存在使用两种标识方法访问的情况。这是我们在使用 ObjectWindows 设计 Windows 应用程序时应注意的一个问题。在本书的程序中，我们将每个 API 函数名前冠以 C++的作用域运算符"::"，以使读者能方便地区分程序中使用的函数哪些是 API 函数，哪些是 ObjectWindows 框架中的成员函数。例如，下述语句表示使用的是 API 函数：

```
::MoveWindow( AWindowsObject.HWindow, ... );
```

这个问题也与 ObjectWindows 的工作机制有关。ObjectWindows 是罩在 Windows API 上的一个外壳，它是基于 API 的，所有对窗口对象的操作都转交 API 来进行。由于这个原因，在创建窗口对象的过程中要特别留意。

当我们基于 ObjectWindows 框架准备创建屏幕上的一个可视对象时，在计算机内存中产生两个对象：一个是基于 ObjectWindows 框架中的类或其子类所创建的对象，一个是使用 API 函数 CreateWindowEx()所创建的对象。这两个对象决定了可视对象的属性和行为：属性由这两个对象中存储的数据来表示，行为由归属于这两个对象的代码来定义。前者的数据由类中定义的操作来处理，后者的数据由有关的 API 函数来处理。在本书中，为对此进行区别，我们将这个类类型的对象称为界面对象；而把由 Windows API 函数所创建的对象称为界面元素，其原因是只有创建了这个对象，屏幕上才显示出一个可视的对象。在一般情况下，如果能从上下文中区别开它们，我们多使用窗口对象一词。

当在程序中创建一个类类型的对象时，对应的可视对象并未被创建，必须使用 TModule::MakeWindow()或 TModule::ExecDialog()成员函数创建界面元素，这两个函数使用 API 函数 CreateWindowEx()创建界面元素，并在创建前完成一些必须的其它工作(例如，向 Windows 注册窗口类等)。当界面元素被创建以后，界面对象的数据成员 HWindow 存储有这个界面元素的句柄。在这个句柄取得合法值以后，我们称界面对象和界面元素建立了关联。

只有在这个句柄取得合法值以后，我们才可以在程序中调用那些需要将该句柄用作参数的 API 函数。在以后各章，我们还将谈到这个问题。

由于上述原因，我们在设计程序时应注意，如果改变了界面对象的数据，而这种改变又没有影响对应的界面元素的数据，那么，屏幕上的可视对象的属性也不会改变。保持这两个对象数据的一致性也是程序设计中应注意的问题。

1.5 继承和关联的设计

面向对象的程序设计的重要特点是分类，即在程序设计中，应注重于对象的类，而不是单

个对象，所设计的程序是由类组成的。

在程序设计中，必须明白类的继承关系和对象的关联。继承反映了对同一个对象在不同的层次上的抽象描述。例如，我们要设计一个文本窗口对象，对该对象可以定义一个名为 TTextWindow 的类来描述。但我们若对文本窗口对象进行抽象，即舍弃其文本方面的特殊性，而保留其它方面的特性，则可以将其视为一个更普通的窗口对象，我们为这个普通的窗口对象定义一个名为 TWindow 的窗口类。由于这种层次抽象方法，所得到的结论是，一个 TTextWindow类的对象可以视为 TWindow 类的对象，这样，通过结构和行为继承，我们可以由 TWindow 类来定义 TTextWindow 类：

```
class TTextWindow : public TWindow {
    // 定义一个文本窗口所特有的结构和行为
};
```

因此，继承机制是通过再用已有类的结构和行为来快速定义新类的一种机制。

在现实世界中的对象不是孤立存在的，而是有着各种联系，关联用于表示对象之间的物理或概念联系。在面向对象的系统中，关联使用对象标识来实现，这样在程序中就可以模拟现实世界的对象的各种联系。

由于面向对象的设计是类的设计，因此，对象的关联被设计为对象的类之间的关联。在 C++中，类 A 与类 B 的关联被表示为：

```
class A {
 // ……
 B b ;
 } ;
```

这种设计一般用于表示一种强关联形式，例如，汽车与发动机之间的关系。在Windows用户界面中，对话框与其中的控制的关系也是强关联的例子之一。强关联是对象之间的一种组装关系，其中 A 的对象被称为组合对象（在 ObjectWindows 中称为父对象），B 的对象被称为 A 的对象的子对象。一个组合对象可以被当作一个简单对象在程序中使用，或用于组装更复杂的对象。

对象之间的关联一般也被设计为：

```
class A {
 // ……
 B *b ;
 } ;
```

它一般用于表达对象之间的弱关联。上述设计表示一个 A 的对象与一个 B 的对象有关联，但在程序运行时，所关联的对象一般可以被动态地改变，即一个 A 的对象可以改变为与 B 的另一个对象相关联。像教师与学生的联系等都是弱关联的例子。

但在程序设计实践中，各种形式的关联多用后面的表示方法，这与指针在 C++程序设计中的独特地位，以及指针所指向的对象可以被动态创建等有关。例如，在 ObjectWindows 程序设计中，父窗口与其控制窗口的关联被使用后者来表达。

掌握 C++中类的设计技术，对编写面向对象风格的 ObjectWindows 应用程序有重要的

意义。在学习后面各章的内容时，诸读者注意这方面的问题。正如 1.1 节所言，用户界面上的可视对象（或广义上说，问题域中的对象）以及它们之间的关系与程序系统中的对象以及它们之间的关系有比较直接的对应，在学习 ObjectWindows 程序设计时，注意体会面向对象的程序系统的这一特点。

第二章　应用程序对象和窗口对象

每个 Windows 应用程序必须要做两件事，第一件事是执行一些初始化工作和消息循环，第二件事是创建一个主窗口。在 ObjectWindows 中，这两件事在 TApplication 类和 TWindow 类中描述。从第一章中的例子我们已知道这两个类提供了一个 Windows 应用程序的基本行为（或称缺省行为）。对特定的应用程序，程序员可以通过派生它们的子类来修订这些行为。本章首先介绍这两个类及其子类，然后介绍使用继承为特定的应用程序裁剪 ObjectWindows 框架中的类的方法。本章还介绍了各类消息的处理问题。

2.1　TModule 类和 TApplication 类

在第一章我们已谈到，Windows 有两种类型的模块，即应用程序模块和动态链接库模块。这两种模块的主要区别是，一个应用程序模块可以有多个实例，它有一个消息循环，用于从应用程序的消息队列中检索消息，并将消息发送到相应的窗口对象上；动态链接库模块只能有一个实例，它为应用程序提供服务，不能单独运行，也没有自己的消息循环，TModule 类和 TApplication 类反映了 Windows 中这两种模块之间的差异。表 2—1 和表 2—2 是 TModule 类的说明，表 2—3 和表 2—4 是 TApplication 类的说明。

表 2—1　TModule 类的公有数据成员

数据成员	含义
HINSTANCE hInstance	模块的实例句柄。
LPSTR lpCmdLine	命令行参数。
int Status	实例的运行状态，非零表示出错。
LPSTR Name	应用程序模块或 DLL 模块的名字。

表 2—2　TModule 类的公有成员函数

TModule(LPSTR AName, HINSTANCE AnInstance, LPSTR ACmdline) 构造函数。AName 是应用程序或 DLL 模块的名字；AnInstance 是模块的实例句柄；ACmdline 为指向命令行参数的指针。这三个参数对应表 2—1 中的三个数据成员。
PTWindowsObject MakeWindow(PTWindowsObject AWindowsObject) 虚函数。将 AWindowsObject 标识的一个界面对象与一个界面元素联系起来。界面元素是一个窗口或是一个无模式对话框。如果执行成功，返回指向界面对象的指针（即 AWindowsObject），否则返回 NULL。

int ExecDialog(PTWindowsObject ADialog)

虚函数。同 MakeWindow，只是将一个界面对象与一个模式对话框联系起来。这个函数在第五章将作详细讨论。

void Error(int ErrorCode) ;

虚函数。将 ErrorCode 所表示的错误代码显示在一个信息框内，并询问用户是否继续，若用户回答 No，则终止程序的运行。

PTWindowsObject ValidateWindow(PTWindowsObject AWindow) ;

该函数检查 AWindow 是否是一个指向有效的对象的指针，若是，则返回该指针，否则，返回 NULL。

ObjectWindows 习惯在类名前冠以一个字母"T"来表示类型，用"PT"表示指针类型，"RT"表示引用类型。例如，表 2－2 中的 PTWindowsObject 表示是一个指向 TWindowsObject 类型对象的指针类型。

表 2－3 TApplication 类的公有数据成员

数据成员	含义
HINSTANCE hPrevInstance	应用程序实例的前一个实例的句柄。
int nCmdShow	如何显示主窗口。
PTWindowsObject MainWindow	指向应用程序主窗口对象的指针。

每个应用程序有一个主窗口，它由 TApplication 类的对象创建。主窗口是用户与应用程序交互的主要渠道，其它窗口都是在主窗口的控制下创建和工作的。

表 2－4 TApplication 类的公有成员函数

TApplication(LPSTR AName,HINSTANCE hInstance,HINSTANCE hPrevInstance,
LPSTR ACmdLine,int ACmdShow)

构造函数。将 WinMain 的参数传递给 TApplication 类的对象。

void Run()

虚函数。它初始化一个实例，创建主窗口，进行消息循环。在主窗口关闭后才从该函数返回。

BOOL CanClose()

查询是否能关闭应用程序。该成员使用表达式 MainWindow－>CanClose()询问主窗口能否关闭。如果该表达式返回 TRUE，则表示可以关闭应用程序。当应用程序在程序终止前要完成一些工作时，例如，将数据存储于盘中，可以利用该函数来完成这些工作。

在表 2—4 中，与表 2—2 同名且含义一样的参数没有介绍。在本书中，为叙述方便，凡是参数名与已介绍的参数同名，并且含义一样的参数，就不再重复介绍。

2.2　TWindowsObject 类和 TWindow 类

TWindowsObject 类是一个抽象类，它定义了 Windows 中各种窗口对象的基本操作和共同行为。Windows 中有这样几类窗口对象：

. 重叠窗口。

. 隶属窗口：它可以有也可以没有一个父窗口。如果有父窗口，它可以显现在父窗口的用户区之外。

. 控制窗口：简称控制。控制是对真实世界中控制面板上控制（例如无线电按钮等）的抽象或仿真，它提供了应用程序接收用户输入或向用户显示信息的通用方法。这类窗口是使用 WS_CHILD 风格创建的。它必须有一个父窗口，并且它在父窗口的用户区之外的部分是不可见的。

表 2—5 和表 2—6 是 TWindowsObject 类的说明。

表 2—5　TWindowsObject 类的公有数据成员

HWND HWindow	窗口界面元素的句柄。如果一个界面对象没有与一个界面元素相关联，它为 0。
LPSTR Title	窗口的标题。
PTWindowsObject Parent	指向父窗口界面对象的指针。如果没有父窗口，它为 NULL。

表 2—6　TWindowsObject 类的公有成员函数

TWindowsObject(PTWindowsObject AParent, PTModule AModule = NULL) 构造函数。AParent 是指向所创建的窗口对象的父窗口界面对象指针，如果所创建的窗口是主窗口或没有父窗口，该参数为 NULL。在应用程序中创建一个窗中对象时，模块指针 AModule 参数使用 NULL，这表明是应用程序对象拥有该窗口对象；如果在动态链接库中创建一个窗口对象，AModule 是该动态链接库中模块对象的指针，这表示是动态链接库模块拥有该窗口对象。ObjectWindows 需要该指针来标识一个模块，以便确定窗口对象的归属，或为窗口对象检索所需要的资源。
PTModule GetModule() 返回指向拥有该窗口对象的一个模块对象的指针。
PTApplication GetApplication() 返回指向拥有该窗口对象的一个应用程序对象的指针。

BOOL CanClose() 虚函数。如果这个窗口能被关闭，则返回 TRUE，否则返回 FALSE。如果该窗口对象含有子窗口，它还要调用子窗口的 CanClose()，询问子窗口是否允许关闭。如果它所拥有的子窗口都同意关闭，该函数才返回 TRUE，否则返回 FALSE。
void SetCaption(LPSTR Attle) 设置窗口的标题为 ATitle 所指向的串。
void Show(int ShowCmd) 见 2.3 节。
void SetParent(PTWindowsObject NewParent) 将该窗口的父窗口设置为 NewParent 指定的窗口对象。
void DefWndProc(RTMessage Msg) ; 虚函数。窗口消息缺省处理函数。
void DefChildProc(RTMessage Msg) ; 虚函数。控制消息缺省处理函数。
void DefCommandProc(RTMessage Msg) ; 虚函数。命令消息缺省处理函数。
void DefNotificationProc(RTMessage Msg) ; 虚函数。控制消息的通知码缺省处理函数。
void EnableKBHandler() ; 允许窗口或无模式对话框响应 TAB 键或箭头键在各个控制之间移动输入焦点。

TWindow 类是 TWindowsObject 类的子类，它提供了窗口所特有的行为。应用程序中的主窗口、各种类型的子窗口都通过裁剪 TWindow 来定义。表 2—7 和表 2—8 是 TWindow 类的说明。

表 2-- 7 TWindow 类的公有数据成员

TWindowAttr Attr 参见 2.3 节。
PTScroller Scroller 指向一个辅助 TWindow 进行滚动显示的对象。
HWND FocusChildHandle 如果该窗口含有子窗口，则它是当前拥有键盘输入焦点的子窗口的句柄。键盘输入被送到拥有输入焦点的窗口上。

表 2　8 TWindow 类的公有成员函数

TWindow(PTWindowsObject AParent, LPSTR ATitle, PTModule AModule = NULL)
构造函数。ATitle 是窗口的标题。
TWindow(HWND AnHWindow, PTModule AModule = NULL)
构造函数。在动态链接库中构造一个 TWindow 类的对象。AnHWindow 是界面元素的句柄，所创建的界面对象被用作该界面元素的别名。在知道了一个界面元素的句柄，需要构造一个 ObjectWindows 界面对象时，使用该构造函数。

2.3　创建窗口对象

窗口对象是 Windows 中最重要的一类对象，Windows 管理着一个界面元素的所有方面，而 ObjectWindows 只维护着它的一个句柄。当 ObjectWindows 创建一个窗口对象时，它向 Windows 注册一个窗口类的有关信息（类名、类风格、光标和窗口背景刷子等）。ObjectWindows 调用成员函数 TWindow::GetClassName()来获取一个类的名字，该成员函数的定义为：

```
virtual LPSTR GetClassName() ;
```

TWindow 类的子类应替换该成员函数，返回子类的类名。

ObjectWindows 调用成员函数 TWindow::GetWindowClass()来获取类的其它信息，该函数的定义是：

```
virtual void GetWindowClass( WNDCLASS& AWndClass) ;
```

WNDCLASS 类型是在 windows.h 中定义的一个类型，其主要部分为：

```
struct WNDCLASS {
    UINT style ;                    类风格。
    HICON hIcon ;                   一个图标对象的句柄。当窗口以极小化方式显示时，显示该图标。
    HCURSOR hCursor ;               一个光标对象的句柄。当光标进入该窗口的用户口时，该句柄所
                                    标识的对象决定了光标显示的形状。
    HBRUSH hbrBackground ;          产生窗口背景的刷子对象的句柄。一个窗口显示的内容使用该
                                    刷子擦除。
    LPCSTR lpszMenuName;            选单资源名，该选单资源定义了选单的结构和选项文本。
} ;
```

ObjectWindows 缺省的类风格是 CS_HREDRAW | CS_VREDRAW。

在 ObjectWindows 框架中，hIcon 的缺省值是 LoadIcon(NULL, IDI_APPLICATION)。hCursor 的缺省值是 LoadCursor(NULL, IDC_ARROW)。hbrBackground 的缺省值是 HBRUSH (COLOR_WINDOW + 1)，它一般指的是一个白色刷子。这些缺省值由 TWindow::GetWindowClass()函数设定。

除了窗口类信息之外，TWindow::Attr 还附加有一些信息来定义一个界面元素的更具体的属性。Attr 的类型为 TWindowAttr，这个类型的定义为：

```
struct TWindowAttr {
    DWORD Style ;                   窗口风格。
```

DWORD ExStyle;	扩展的窗口风格。
int X,Y,W,H;	窗口左上角的坐标以及窗口的宽度和高度。
LPSTR Menu;	选单名，如果指定了该域，则它将取代类中指定的选单，而成为窗口元素显现时的选单。
int Id;	子窗口的标识符。
LPSTR Param;	辅助数据，一般不用此域。

};

表 2－9 是 ObjectWindows 为 Style 域设置的缺省值。

表 2－9 Style 域的缺省值

窗口对象	缺省值
主窗口	WS_OVERLAPPEDWINDOW
隶属窗口	WS_VISIBLE
控制窗口	WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP
MDI 子窗口	WS_CLIPSIBLINGS

为保持与 ObjectWindows 的设置不相矛盾，在修改 Attr.Style 域时，应使用 & 或 | 运算符。例如：

```
Attr.Style |= LBS_NOTIFY ;
```

特别是对 TCntrol 类的子窗口对象，Windows 要求 Style 域设置有特定的值，ObjectWindows 总是为 Style 域设置适当的值。ObjectWindows 为 X、Y、W 和 H 设置合适的值，这些值根据屏幕显示情况自动调整，因此，一个窗口对象可以不设置这四个域的值，而使用缺省值。

Menu 域、Id 域和 Param 域一般被置为 0。对于子窗口，Id 域由构造函数置为相应的值。数据成员 Menu 只能读取，若要为它设置新值，要使用成员函数

```
BOOL TWindow::AssignMenu( LPSTR AMenuName );
```

来进行。

下面的程序演示了如何使用本节介绍的知识创建一个特定的窗口对象的方法。由于用 ObjectWindows 编写的任何程序必须至少有两类对象：TApplication 类的对象和 TWindow 类的对象，所以，在我们这个示例程序中，分别由 TApplication 类和 TWindow 类派生了一个 TFirstApplication 类和 TFirstWindow 类。TFirstWindow 类替换了 GetClassName() 和 GetWindowClass() 成员函数，来设置特定的窗口特性：使用十字光标和黑色的背景刷子。

必须在 TFirstApplication 类中替换 InitMainWindow。主窗口是由 TApplication 类的对象创建的，当 TApplication 类创建一个主窗口对象时，它调用 InitMainWindow()。在缺省情况下，InitMainWindow() 创建一个 TWindow 类的对象作为主窗口对象。我们这个程序对 InitMainWindow() 进行了替换，创建了一个 TFirstWindow 类的对象作为主窗口对象。

```
// first.h
#if ! defined( _FIRST_H )
#define  _FIRST_H
#include <owl.h>
```

```
class TFirstWindow : public TWindow {
    public:
        TFirstWindow( PTWindowsObject AParant, LPSTR Title ) ;
        virtual LPSTR GetClassName() { return "FirstWindow" ; }
        virtual void GetWindowClass( WNDCLASS& ) ;
} ;

class TFirstApplication : public TApplication {
    public:
        TFirstApplication( LPSTR AName, HINSTANCE hInstance,
            HINSTANCE hPrevInstance, LPSTR ACmdLine, int nCmdShow ) ;
        virtual void InitMainWindow() ;
} ;
#endif // _FIRST_H

// first.cpp
#include "first.h"

TFirstWindow::TFirstWindow( PTWindowsObject AParent, LPSTR Title )
        : TWindow( AParent, Title )
{
}

void TFirstWindow::GetWindowClass( WNDCLASS& wc )
{
    // call the base class for proper operation
    TWindow::GetWindowClass( wc ) ;

    // now change what you want change
    wc.hCursor = LoadCursor( 0, IDC_CROSS ) ;
    wc.hbrBackground = (HBRUSH)GetStockObject( LTGRAY_BRUSH ) ;
}

TFirstApplication::TFirstApplication(
LPSTR AName, HINSTANCE hInstance,
HINSTANCE hPrevInstance, LPSTR ACmdLine, int nCmdShow )
        : TApplication( AName, hInstance, hPrevInstance, ACmdLine, nCmdShow )
{
        TApplication::nCmdShow = SW_SHOWMAXIMIZED ;
}

void TFirstApplication::InitMainWindow()
```

```
{
        MainWindow = new TFirstWindow( 0, Name );
}

int PASCAL WinMain(
HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPSTR lpszCmdLine,
int nCmdShow )
{
        TFirstApplication First("First", hInstance, hPrevInstance,
                                        lpszCmdLine, nCmdShow );
        First.Run();
        return First.Status;
}
```

在结束本节之前，我们谈一下 ObjectWindows 的运行机理。当创建了一个 TApplication 类的对象之后，程序紧接着是调用成员函数 Run()。基于所运行的应用程序实例是否是第一个实例，成员函数 Run()有不同的行为。我们下面看一下 TApplication::Run 是怎样定义的。

```
void TApplication::Run()
    {
        if( !hPrevInstance )        // 如果是当前应用程序的第一个实例
            InitApplication();      // 执行应用程序的初始化工作
        if( Status == 0 )           // 如果到目前为止未出现错误
            InitInstance();         // 执行实例的初始化工作
        if( Status == 0 )
            MessageLoop();          // 执行消息循环。应用程序实例在此处等待，
                                    // 直到关闭了主窗口，才继续向下执行
        else
            Error( Status );        // 终止并显示错误信息
    }
```

Windows 将发送到某一个应用程序(的窗口对象)上的消息放在该应用程序的一个消息队列中，由应用程序执行一段消息循环代码来检索这些消息。当检索到一条消息之后，消息循环代码就将消息发送到相应的窗口对象上；如果未检索到消息，应用程序就被挂起，等待 Windows 在其应用程序的消息队列中放入消息。ObjectWindows 最大的益处之一是封装了消息循环，使程序员不关心消息是怎样发送的。

TApplication::Run()只为应用程序的第一个实例调用 InitApplication()，而对于应用程序的每一个实例，Run()都要调用 InitInstance()。我们看一下 InitInstance()成员函数是怎样实现的。

```
void TApplication::InitInstance()
    {
```

```
        InitMainWindow();          // 调用用户替换后的函数建立一个界面对象
        MainWindow = MakeWindow( MainWindow );   // 将界面对象与
                                                 // 界面元素联系起来
        if ( MainWindow )
            MainWindow->Show( nCmdShow );        // 显现界面元素
        else
            Status = EM_INVALIDMAINWINDOW;
    }
```

所有的界面对象必须使用 new 运算符在堆上动态创建。当一个界面元素被撤销，或在将界面对象与界面元素相关联时出现错误时，ObjectWindows 都自动地使用 delete 运算符删除界面对象。读者在后面会发现，本书的程序在创建界面对象时使用了 new 运算符，但几乎没有显式使用 delete 运算符删除界面对象的原因就在此。成员函数 InitApplication()、InitInstance() 和 InitMainWindow() 是在 TApplication 类中说明的保护的虚函数。

2.4　事件与消息

事件或消息是信息从一个对象到另一个对象的单向转递，Windows 将系统中所发生的任何事件存储在一个 TMessage 结构中，该结构的定义是：

```
struct TMessage {
    HWND Receiver;              // 消息的接收者
    WORD Message;               // 消息标识符
    union {
        WORD WParam;            // 随同消息一起发送的十六位数据
        struct tagWP {
            BYTE Lo;            // 低字节部份
            BYTE Hi;            // 高字节部份
        } WP;
    };
    union {
        DWORD LParam;           // 随同消息一起发送的三十二位数据
        struct tagLP {
            WORD Lo;            // 低十六位
            WORD Hi;            // 高十六位
        } LP;
    };
    long Result;                // 消息处理之后的返回值
};
```

一个消息被发送之后，处理消息的对象应向发送消息的对象发送一条消息来回答消息处理的结果，但是在 Windows 中，发送消息是以函数调用的方式处理的，因此，消息的处理结果也被放到了消息结构中。在 Windows 中，主要有两大类消息：用户界面与用户相互作用的