第一章 ObjectWindows 概述

ObjectWindows 2.0 是为 Windows 3.1、Win32s 和 Windows NT 开发的应用程序框架。ObjectWindows 让你快速容易地开发具有 Windows 所有风格的应用程序。

ObjectWindows 2.0 具有以下特点：

- 容易在 16 位和 32 位平台间方便移植；
- 自动消息分发；
- 意外差错和错误处理；
- 允许移植到其他编译器和环境；
- 封装了 Windows GDI 对象；
- Doc/View 类使得数据抽取和显示更加简便；
- 打印机和打印预浏览类；
- 支持 Visual Basic 控制；
- 输入合法性检测器；

本章简单浏览一下 ObjectWindows 2.0 的等级结构和 ObjectWindows 2.0 的类库，解释每种类是如何与其他类相配合的，指导读者获取每个类的具体信息以便使用它们。

1.1 使用类等级的继承

这一部分讲述类的基本属性，重点放在 ObjectWindows 类上。包括三个主题：

- 如何处理类
- 继承的成员
- 成员函数的类型

1.1.1 使用类

对一个类，可以做三件最基本的事：

- 从它派生出一个新类
- 将它加到另外一个类中
- 对该类实例化（创建该类的一个实例）

一、派生出一个新类

给一个类增加或修改一些动作，可以从该类派生出一个新类，例如，我们从 TWindow 派生出类 TNewWindow：

```
class TNewWindow:public TWindow
```

```
{ public:
  TNewWindow(...);
  //...
};
```

派生一个新类时,可做三件事:

- 增加新数据成员;
- 增加新的函数成员;
- 重写继承来的成员函数;

增加新成员来增加或修改基类的功能;通过给派生类定义构造函数来调用基类的构造函数,并初始化增加的数据成员。

二、组合对象的行为

ObjectWindows 设计成使用多重继承。可以派生一个类继承多个类的行为。像单一继承一样,可以给多重继承的派生类增加成员函数和重写继承来的成员函数。

三、实例化类

要使用一个类,必须创建该类的实例。可以有几种方法实例化一个类:

- 使用标准的语法声明。就像声明标准变量 int 或者 char 一样。下列通过调用不带参数的 TMyApplication 构造函数来实例化 app 对象。

  ```
  TMyApplication app;
  ```

  这种声明方法只有在该类具有缺省构造函数或者构造函数的所有参数具有缺省值时才行。
- 也可用声明带有参数的构造函数实例化一个类。下列通过调用带有 char * 参数的 TMyApplication 构造函数来实例化 app 对象。

  ```
  TMyApplication app("AppName");
  ```

- 通过调用 new 操作符分配空间来实例化一个对象。如:

  ```
  TMyApplication * app;
  app=new TMyApplication;
  ```

- 也可以用 new 操作符,调用带有参数的构造函数实例化对象。如:

  TMyApplication * app=New TMyApplication("AppName");构造函数调用基类的构造函数来初始化数据成员。只能实例化非抽象类;即不含纯虚拟函数的类。

四、抽象类

抽象类具有纯虚拟函数,这个虚拟函数必须在派生类中重写。抽象类具有两个主要功用。一是创建其他类时提供一个概念上的框架;二是产生编码效果。

例如,ObjectWindows 类 THSlider 和 TVSlider 可以直接从 TScrollBar 派生。尽管一个

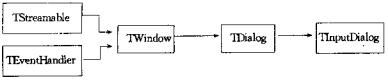
是垂直滚动块，另一个是水平滚动块，实际上它们具有相同的功能和响应。这种共性就保证可以创建一个抽象类 TSlider。THSlider 和 TVSlider 从 TSlider 派生，只须增加少量的特殊成员函数来画不同的滑块。

不可能实例化抽象类。其纯虚拟函数成员必须经过重写，以创建一个有用的实例。例如 TSlider 就不知道怎样画出自己或者响应鼠标事件。

如果要创建自己的滑块（例如图形滑块），可能想从 TSlider 派生，但从 THSlider 或 TVSlider 派生可能更简单，这取决于哪个更符合要求。两种情况下，都要增加数据成员，增加或重载函数成员来增加所需的功能。如果想创建一个对角形滑块（西北—东南和西南—东北走向），可能要创建一个中间抽象类 TAngledSlider。

1.1.2 继承的成员

下表显示了 TInputDialog 的继承关系：



TInputDialog 从 TDialog 继承而来，TDialog 又从 TWindow 继承，TWindow 则从 TStreamable 和 TEventHandler 继承而来。沿着继承等级路线可以增加更具体的动作。

下表列出了每个类的公共数据成员，包括从 TDialog 和 TWindow 基类继承来的成员：

TWindow	TDialog	TInputDialog
Status	Status	Status
HWindow	HWindow	HWindow
Title	Title	Title
Parent	Parent	Parent
Attr	Attr	Attr
DefautProc	DefaultProc	DefaultProc
Scroller	Scroller	Scroller
	IsModal	IsModal
		Prompt
		Buffer
		BufferSize

TInputDialog 继承了 TDialog 和 TWindow 的所有数据成员，并增加了一些数据成员。

要完全理解 TInputDialog，就必须理解它的继承关系：TInputDialog 既是一个对话框，又是一个窗口。TDialog 给窗口类增加模式的概念，TInputDialog 增加了存储和检索用户输入数据的能力。

1.1.3 成员函数的种类

有四种 ObjectWindows 成员函数：

- 虚拟函数
- 非虚拟函数
- 纯虚拟函数
- 缺省函数

一、虚拟函数

虚拟函数可在派生类中重载。虚拟函数和纯虚拟函数区别在于虚拟函数不一定非要被重载。虚拟函数提供了多态机制和一致的类接口能力,即使这些类功能大不相同。

二、非虚拟函数

不要重载非虚拟函数。因而,一个函数若要在派生类中重载,就应该将它定义成虚拟函数(事件响应表中的事件处理函数是一个例外)。例如,TWindow::CanClose 是虚拟函数,在派生类中可以重载它,以检测窗口是否可以关闭。而 TWindow::SetCaption 是非虚拟函数,因为不必要修改窗口标题设置的方式。

重载非虚拟函数导致的问题是:从派生类再派生的类可能试图使用重载函数。如果这个新的派生类不清楚父派生类已经改变了该派生的函数,这可能导致函数返回错误值,产生运行错误。

三、纯虚拟函数

在派生类中必须重载纯虚拟函数。通过下述方式将函数声明为纯虚拟函数。举例说明:TSlider::PaintRuler 是纯虚拟函数,声明是:

```
virtual void PaintRuler(TDC & dc)=0;
```

在派生类中,必须重载抽象基类中所有的纯虚拟函数。这样才能实例化派生类。绝大多数情况下,当使用标准的 ObjectWindows 类时,这样做不存在多大问题。纯大多数可以从其派生出类的 ObjectWindows 类不是抽象类。许多 ObjectWindows 类使用缺省的替代函数来代替纯虚拟函数。

四、替代函数

和纯虚拟函数不同,缺省的替代函数不用重载。这些函数提供最小的缺省动作或者干脆什么也不做。所以称为替代函数,是因为它们在程序中占有一个位置,在派生类中,可在这个位置放置代码。如 TWindows::EvLButtonDblClk:

```
inline void
TWindow::EvLButtonDblClk (UINT modkeys, TPoint &)
{ DefaultProcessing();
}
```

缺省时,EvLButtonDblClk 调用 DefaultProcessing 来响应双击鼠标事件。在自己定义的窗口类中,可以重载 EvLButtonDblClk 函数,将它定义在类的响应表中。定义的 EvLButtonDblClk 函数可以提供一些动作来处理用户双击鼠标的操作。也可调用基类的这个函数来

提供缺省处理动作。

1.2 对象类型

ObjectWindows 等级有很多不同的类，可以使用，修改或增加，可以将类分为以下几组：

- 窗口类
- 对话框类
- 控制类
- 图形类
- 打印类
- 模块和应用程序类
- Doc/View 类
- 各种 Windows 元素(杂项类)

1.2.1 窗口类

窗口是所有 Windows 应用程序重要组成部分，ObjectWindows 提供几种不同的窗口类：

- 窗口(一般窗口)
- 框架窗口
- MDI(多文档接口)窗口
- 修饰过的窗口

详细说明见后续章节。

一、窗口

TWindow 是所有窗口类的基类。它代表所有窗口的共同功能。不管这些窗口是对话框、控制还是 MDI 窗口。

二、框架窗口

框架窗口 TFrameWindow 从 TWindow 派生。它在 TWindow 上增加了框架窗口的功能，能拥有其他客户窗口。

三、*MDI* 窗口

多文档接口是管理多文档或一个应用程序里所有窗口的 Windows 标准。在 ObjectWindows 应用程序中，TMDIFrame，TMDIClient 和 TMDIChild 支持 MDI。

四、修饰窗口

TLayoutWindow 和 TLayoutMetrics 等几个类支持修饰控制(如工具条、状态条和消息条)。通过多重继承，可以在框架窗口和 MDI 框架窗口中增加修饰支持。这两个类是 TDecoratedFrame 和 TDecoratedMDIFrame。

1.2.2 对话框类

对话框 TDialog 从窗口类 TWindow 继承而来，用对话框类创建对话框，以处理各种各样的用户交互动作。典型的对话框包含控制来获得用户输入。后续章节将详细叙述对话框类。

一、公用对话框

除了应用程序创建的对话框外，ObjectWindows 支持 Windows 的公共对话框

- 选择文件(TFileOpenDialog，TFileSaveDialog)；
- 选择字体(TChooseFontDialog)；
- 选择颜色(TChooseColorDialog)；
- 选择打印选项(TPrintDialog)；
- 查找和替换文本(TFindDialog，TReplaceDialog)。

二、其他对话框

ObjectWindows 还提供其他一些对话框，这些对话框不是基于 Windows 公用对话框的。

- 输入文本(TInputDialog)
- 退出打印任务(TPrinterAbortDlg，常和 TPrinter 和 TPrintout 类一起用)。

1.2.3 控制类

控制类 TControl 从窗口类 TWindow 派生，以支持所有控制的通用行为。ObjectWindows 提供四种控制：

- 标准窗口控制
- 小工具(Widgets)
- 小配件(Gadgets)
- 修饰件(Decorations)

一、标准窗口控制

标准窗口控制有列表框、滚动条、按钮、复选框、互锁按钮、成组框、编辑控制、静态控制和组合框。

二、小工具

不同于标准窗口控制，ObjectWindows 小工具是用 C++写的特殊控制，ObjectWindows 提供的小工具有水平和垂直滚动条(THSlider 和 TVSlider)及量规(TGauge)。

三、小配件

小配件和标准窗口控制相似之处是：它也用来获取用户输入或给用户传送信息，但小配件和标准控制实现机理不同。不同于大多数界面元素，小配件不是窗口，不具备窗口句柄，不

能接收事件和消息，它不是基于 TWindow 的。

相反，配件必须位于配件窗口内，配件窗口负责配件的显示，所有的消息处理等，配件从配件窗口接收命令。

四、修饰件

修饰件是特殊的子窗口，用来让用户选择命令，给用户提供信息，有时还用来和用户交流特殊的信息。

- 控制条(TControlBar)让用户在窗口里安放一系列控制按钮，这些按钮常用作菜单的快捷键用。
- 工具框 TToo(Box)让用户在浮动板上安放一系列按钮。
- 消息条(TMessageBar)一般位于窗口的底部，用来向用户显示消息。例如 Borland C++集成环境用消息条显示菜单命令项的简单叙述。
- 状态条(TStatusBar)和消息条相似，不过同时可显示几条信息。例如在 Borland C++编辑窗口时，右下角显示 Insert 键，覆盖键(OverType)和一些错误信息。

1.2.4 图形类

Windows 提供功能强大但相当复杂的图形库叫 GDI。ObjectWindows 将 GDI 函数封装起来，这样使用设备上下文(DC)类和 GDI 对象(TGDIObject)时就不用太费神。

一、设备上下文类

通过 GDI 而不是直接在设备(如屏幕和打印机)上画，能够用设备上下文在位图里绘画。设备上下文是绘图工具，图形设置和有关图形设备及绘图状态等这些设备信息的集合。这样在使用 GDI 时，可以保证高度的设备无关性。下表是 ObjectWindows 将各种设备上下文封装后的图表：

设备上下文种类	ObjectWindows 设备上下文类
Memory	TMemoryDC
Metafile	TMetaFileDC
Bitmap	TDibDC
Printer	TPrintDC
Window	TWindowDC
Desktop	TDeskTopDC
Screen	TScreenDC
Client	TClientDC
Paint	TPaintDC

二、*GDI* 对象

TGDIObject 是其他几个类的基类，这几个类可以用来画图和控制图形显示。下表是 ObjectWindows 支持的这些类的列表：

GDI 对象	ObjectWindows GDI 类
Pens	TPen
Brushes	TBrush
Fonts	TFont
Palettes	TPalette
Bitmaps	TBitmap, TDib, TUIBitmap
Icons	TIcon
Cursors	TCursor
REgions	TRegion
Points	TPoint
Size	TSize
Rectangles	TRect
Color specifiers	TColor
RGB triple color	TRgbTriple
RGB quad color	TRgbQuad
Palette entries	TPaletteEntry
Metafile	TMetafilePict

1.2.5 打印类

由于封闭了和打印驱动程序通信的函数，TPrinter 类使得打印异常简单。TPrintout 将打印文档的任务封闭起来。第二章第十二节讲述如何使用打印类。

1.2.6 模块和应用程序类

Windows 应用程序负责初始化窗口，保证将 Windows 传递给它的消息送给合适的窗口。ObjectWindows 将这些行为封闭在类 TApplication 中。TModule 封闭了 DLL(动态连接库)行为。

1.2.7 文档/浏览器类

文档/浏览器类是通用文档——浏览器模块全部抽象而成的。TDocManager，TDocument 和 TView 是 Doc/View 的基类。Doc/View 模块是一个系统，数据包含在该系统中，由文档对象来存取这些数据。浏览器对象显示和操作这些数据。一个文档种类可以和任意数量的浏览器关联起来，用不同的方式来显示同一批数据。

例如，可以用两种方式显示一条线。一种是图形方式(在一个窗口中显示一条直线)，一种是数字方式(将组成这条线的坐标点显示出来)。这样，就需要一个文档和两个浏览器：一个浏览器在屏幕上显示一条线，另一个浏览器显示组成直线的坐标点。也可以通过浏览器来修改数据。在上述情况下，就可以在图形显示器里画线来修改直线的数据，或者在数字显示器里输入数字修改直线的坐标点。

1.2.8 杂项类

一、菜单类

菜单可以是静态菜单，也可以修改甚至可以装入新的菜单。TMenu 及其派生类(TSystemMenu 和 TPopupMenu)用来操作菜单。

二、剪贴板

Windows 剪贴板是应用程序之间共享数据的主要方式。ObjectWindows 的 TClipboard 类使用户能在应用程序中支持剪贴板。

第二章 学习编制 ObjectWindows 应用程序

ObjectWindows 2.0 讲述了使用 ObjectWindows 应用程序框架编写 Windows 程序的很多基本知识。手册由十多个由简到繁的 ObjectWindows 应用程序组成。学习完这十多个步骤之后，读者就会对 Windows 应用程序的一些特点如菜单、对话框、图形控制条、状态条、MDI 窗口等，有一个全面的了解。本手册假设读者熟悉 C++编程并有 Windows 编程经验。开始编程之前，阅读第一章会有所帮助。这一章简单地浏览了 ObjectWindows 2.0 的类等级。

2.1 应用程序对象

ObjectWindows 2.0 用类 TApplication 和 TModule 分别封闭了窗口应用程序和 DLL 模式。类 TModule 对象封闭了窗口动态链接库(DLL)的初始化和关闭函数，类 TModule 还包含了 hInstance 和 LpCmdline 参数，它们等同于在非 ObjectWindows 应用程序中传递给 WinMain 函数的同名参数，注意函数 WinMain 和 LibMain 都有这两个公共的参数。类 TModule 更详细的讨论见有关章节。

类 TApplication 对象封装了 Windows 应用程序中的初始化，运行管理和关闭函数，类 TApplication 对象还包含了参数 hPrevInstance 和 nCmdshow 的值，它们等同于在非 ObjectWindows 应用程序中传给 WinMain 的同名参数，由于类 TApplication 是类 TModule 的派生类，所以类 TApplication 也具有类 TModule 所包含的功能属性。

另外，类 TApplication 对象包含容易调用的 Borland Custom Controls Library 库和 Microsoft Controls Library 库的一些函数，还有一个函数自动从象 Microsoft 3-D 那样的控制中获取标准控制的子类。详细的内容参见后续章节的有关部分

设计 ObjectWindows 2.0 应用程序不必明确地提供一个 WinMain 函数，这里可以用函数 OwlMain 代替。函数 OwlMain 用整型的(int)argc 和字符型的(char) * * argv 作为其参数，并返回一个整型值，就像传统的 C 语言和 C++语言设计的程序中用函数 main 一样。详细的内容见前面的叙述部分。

这一节讲述怎样使用类 TApplication 对象。设计者不必创建自己的 TModule 类对象，在动态链接库(DLL)下工作是个例外。在 ObjectWindows 应用程序中使用动态链接库(DLL)的更详细内容参见有关 DLL 的说明。

为了使用类 TApplication 对象，首先应完成以下几步：

- 包含正确的头文件
- 创建一个对象
- 得到这个对象

2.1.1 包含正确的头文件

类 TApplication 是在头文件 OWL\applicat.h 中定义的，所以在使用类 TApplication

时，应包含这个头文件。另外，由于类 TApplication 是类 TModule 的派生类，而头文件 OWL\applicat.h 包含头文件 OWL\module.h，所以程序中包含头文件 OWL\applicat.h 就可以了。

2.1.2　创建一个类对象

设计者可以用两个构造函数中的一个创建一个 TApplication 类对象，通常使用的构造函数是这样定义的。

```
TApplication (const char far * name);
```

类 TApplication 中此版本的构造函数有一个字符串参数，这个字符串就成了应用程序的名字，如果不指定一个名字，默认状态下，构造函数将字符串定义为空。类 TApplication 将这个字符串定义为应用程序的名字。

类 TApplication 的第二版的构造函数定义了一些参数，相当于通常情况下传给函数 WinMain 的参数：

```
TApplication (const char far * name, HINSTANCE instance,
              HINSTANCE prevInstance, const char far * cmdLine, int cmdShow);
```

设计者可以用这个构造函数将命令行参数传给类 TApplication 对象。

2.1.3

类 TApplication 包含了几个需要在应用程序对象外面调用的成员函数和数据成员，为了能访问它们，类 TWindow 有一个成员函数 GetApplication，它返回一个指向应用程序对象的指针，可以用这个指针去调用类 TApplication 的成员函数和访问类 TApplication 的数据成员。

下面列出了使用函数 GetApplication 的一种可能情况：

```
void TWindow::Error()
{
// display message box containing the application name
Message("An error occurred!",
        GetApplication() ->Name,MB_OK);
}
```

2.1.4　创建最小的应用程序

下面设计了一个最小的 ObjectWindows 应用程序：

```
#include<OWL\applicat.h>
int OwlMain(int/ * argc * /,char/ * . * argv[] * /)
{
    return TApplication ("Minimum program").Run();
}
```

这里创建了一个带有标题“Mininum Program”的主窗口的 Windows 应用程序，用户可以改变窗口尺寸、移动、最小化、最大化和关闭这个窗口。在实际应用中，将会从这个应用类

派生出一个具有更多功能属性的新类。

2.1.5 初始化应用程序对象

初始化一个 ObjectWindows 应用程序需要四步：

- 构造应用程序对象
- 初始化应用程序对象
- 初始化每个事例
- 初始化主窗口

一、构造应用程序对象

构造一个 TApplication 类对象时，首先调用成员函数 InitApplication，InitInstance 和 InitMainWindow 启动应用程序，可以通过重载这些成员函数来显示应用程序是如何初始化的。为了重载类 TApplication 中的函数，必须从类 TApplication 中派生出自己的应用对象类。

用在这里的类 TApplication 的构造函数将应用程序名作为其唯一的参数，它的默认值为 0，表示没有名，这个应用程序名用于默认的主窗口标题和错误信息中。应用程序类的构造函数应该调用类 TApplication 的构造函数。下面的例子是类 TApplication 的派生类的框架一部分：

```
#include<_owl\applicat.h>
Class TMyApplication:public TApplication
{public:
  TMyApplication(const char far * name=0)
  :TApplication(name){}
  ...
};
```

ObjectWindows 2.0 应用程序不要求一个显式调用 WinMain 函数，ObjectWindows 库提供了错误处理和意外事件处理机制。可以在函数 OwlMain 中执行任何想要的初始化工作。缺省的 WinMain 函数调用 OwlMain 函数。

为了构造一个应用程序对象，在函数 OwlMain 中创建一个应用类的实例。下面的范例显示了一个简单的应用对象类的定义，并创建了实例：

```
#include<_owl\applicat.h>
class TMyApplication:public TApplication
{ public:
        TMyApplication(const char far *name=0)
        :TApplication(name){}
};
int OwlMain(int /*argc*/,char /* * argv[]*/)
{
    return TMyApplication("Wow!").Run();
```

```
}
```

(一)使用 WinMain 和 OwlMain 函数

在 ObjectWindows 2.0 中有一个默认的函数 WinMain，它提供了广泛的错误检查和意外事件处理功能，这个 WinMain 函数设置应用程序并调用 OwlMain 函数。

虽然可以在源文件中使用自己设计的 WinMain 函数，但似乎没有理由要这样做，任何情况下在函数 WinMain 中完成的工作都可以在函数 OwlMain 或者在类 TApplication 的初始化成员函数中完成。

下面的范例显示了在应用程序中函数 OwlMain 的典型应用：

```
#include<owl\applicat.h>
#include<string.h>
class TMyApplication:public TApplication
{ public:
      TMyApplication(char far *name=0):TApplication(name){}
};
int OwlMain(int argc,char *argv[])
{ char title[30];
   if (argc>=2)
      strcpy(title,argv[1]);
   else
      strcpy(title,"Wow!");
   return TMyApplication(title).Run();
}
```

如果经过考虑需要设计自己的 WinMain 函数，类 TAppliation 支持用另一个构造函数传递传统的 WinMain 函数的参数。下面的例子显示了怎样利用构造函数将函数 WinMain 的参数传给类 TApplication 对象：

```
#include<owl\applicat.h>
#include<string.h>
class TMyApplication:public TApplication
{ public:
      TMyApplication(const char far *name,
                                          HINSTANCE     instance,
                                          HINSTANCE     prevInstance,
                                          const char far * cmdLine,
                                          int           cmdShow)
      :TApplication(name,Instance,prevInstance,cmdLine,cmdShow)
      {}
};
int PASCAL WinMain(HINSTANCE Instance,HINSTANCE prevInstance,
                   LPSTR lpszCmdLine,int nCmdShow)
```

```
{ return
  TMyApplication("MyApp",Instance,prevInstance,lpszCmdLine,nCmdShow).Run();
}
```

这个程序和 Borland C++3.1 程序非常相近,在 BC 3.1 版本中,TApplication 的构造函数见有关资料,读者可以对比一下。

二、初始化应用程序对象

用户可以同时多次运行同一个应用程序,从 16 位的应用程序看,第一个事例的初始化仅仅在另一个相同的应用程序不在运行时发生,每个事例的初始化每次都在用户运行应用程序时发生。如果用户启动又关闭一个应用程序,然后又启动,等等,则每个事例都是第一个事例。因为这些事例都不是同时运行的。如果在 32 位上运行,则每个应用程序运行在自己的地址空间上,而不共享事例数据,所以每个事例都是以第一个事例的身份出现,因此每次运行 32 位应用程序时,既完成第一个事例初始化又完成每个事例初始化。

如果当前的事例是第一个事例(通过将数据成员 hPreInstance 设置为 0 显示出来),则函数 InitApplication 被调用。设计者可以在派生的应用程序类中重载函数 InitApplication,这个函数默认时不起作用。

例如,可以使用第一个事例初始化改变主窗口标题栏来反映当前的事例是不是第一个事例,为此需要在派生的应用程序类中增加一个名为 WindowTitle 的数据成员,在构造函数中,将 WindowTitle 设置为"Additional Instance"。重载函数 InitApplication 将 WindowTitle 置为"First Instance"。如果当前应用程序的事例是第一事例,调用函数 InitApplication,重写构造函数定义的主窗口标题名变量 WindowTitle 值。下面的例子列出了这段源程序代码:

```
#include<owl\applicat.h>
#include<owl\framewin.h>
#include<string.h>

class TTestApp:public Tapplication
{ public:
      TTestApp():TApplication("Instance Tester")
      { strcpy(WindowTitle,"Additional Instance");
      }
      protected:
        char WindowTitle[30];
        void InitApplication()
        { strcpy(WindowTitle,"First Instance");
        }
        void InitMainWindow()
        { SetMainWindow(new TFrameWindow(0,WindowTitle));
        }
};
int OwlMain (int argc,char * argv[])
{ return TTestApp().Run();
```

}

另外，应用程序在 32 位上运行时，可能并不像想像的那样工作，因为在 32 上位运行时，每个事例都把自己当作是应用程序的第一事例，同一应用程序同时多次运行将都在主窗口上显示标题"First Instance"。

三、初始化每个新的事例

用户可以同时多次运行同一应用程序，通过重载函数 TApplication::InitInstance 完成对每一事例所需的初始化。

函数 InitInstance 调用函数 InitMainWindow，然后创建并显示在函数 InitMainWindow 中定义的主窗口，如果重载函数 InitInstance，一定要保证新的 InitInstance 函数调用函数 TApplication::InitInstance。下面的例子显示如何利用函数 InitInstance 装入加速器表：

```
void
    TTestApp::InitInstance()
{
    TApplication::InitInstance()
    HAccTable=LoadAccelerators(MAKEINTRESOURCE(MYACCELS));
}
```

四、初始化主窗口

默认状态下，函数 TApplication::InitMainWindow 用与应用对象相同的名字创建一个框架窗口，这个窗口并不是很有用，因为它不能接受和处理任何输入的信息。必须重载函数 InitMainWindow 创建一个窗口对象来处理用户的输入，通常，函数 InitMainWindow 创建类 TFrameWindow 或 TFrameWindow 派生类的对象并调用函数 SetMainWindow。函数 SetMainWindow 有一个指针 TFrameWindow * 作为参数，并返回原来主窗口的指针(如果是一个没有设置主窗口的新的应用程序，则返回值为 0)。

下面例子中的应用程序创建了类 TFrameWindow 对象并将它作为主窗口：

```
#include<owl\applicat.h>
#include<owl\framewin.h>
class TMyApplication:public TApplication
{ public:
        TMyApplication():TApplication(){}
        void InitMainWindow();
};
void
TMyApplication::InitMainWindow()
{SetMainWindow(new TFrameWindow(0,"My First Main Window"));
}
int OwlMain(int argc,char * argv[])
{ return TMyApplication().Run();
```

```
}
```

运行这个程序时,标题栏上写的是"My First main Window"类 TApplication 的构造函数传递的应用程序名只有在没有提供主窗口时才有用,还是一样,这个例子不能完成太多的工作,因为仍然没有提供框架窗口来处理用户的输入,但是一旦派生出一个窗口类处理用户接口,就可以通过同样简单的办法显示这个窗口。

(一)指定主窗口显示模式

通过设置类 TApplication 中的数据成员是 nCmdShow,可以改变应用程序的主窗口,这个数据成员相当于函数 WinMain 中的参数 nCmdShow。一旦 Run 函数运行时便可设置这个变量值,直到调用函数 TApplication::InitInstance。这也显然意味着可以在函数 InitApplication 中或函数 InitMainWindow 中设置变量 nCmdShow 的值。

例如,如果你希望用户运行应用程序时,显示的窗口为最大尺寸,那么可以在函数 InitMainWindow 中设置 nCmdShow 的值:

```
#include<owl\applicat.h>
#include<owl\framewin.h>

class TMyApplication:public TApplication
{public:
        TMyApplication(char far * name):TApplication(name){}
        void InitMainWindow()
        {SetMainWindow(new TFrameWindow(0,"Maximum Window"));
                nCmdShow=SW_SHOWMAXIMIZED;
        }
};
int OwlMain(int argc,char * argv[])
{return TMyApplication("WOW").Run();
}
```

nCmdShow 作为函数 ShowWindow 或成员函数 TWindow::Show 的参数,可以设置为任意合适的值,像 SW_HIDE、SW_SHOWNORMAL 和 SW_NORMAL 等等。

(二)改变主窗口

在应用程序运行过程中,可以用函数 SetMainWindow 来改变主窗口。函数 SetMainWindow 有一个参数是 TFrameWindow *,并返回一个指向原来主窗口的指针(如果这是一个新的应用程序,还没有设置主窗口,则返回值是 0)。可以用这个指针保存原来的主窗口以便保存,这样可以有准备地用这个指针删除原来的主窗口对象。

2.1.6 应用程序的消息处理

一旦应用程序初始化后,应用程序对象的函数 MessageLoop 便开始运行,函数 MessageLoop 处理从 Windows 得来的所有消息,有两种方法可以简化从一个 ObjectWindows 应用程序中传来的消息。

■ 通过重载默认的消息处理函数，作额外的消息处理。
■ 挂起处理进程

一、额外的消息处理

类 TApplication 有一些成员函数为所有的 ObjectWindows 应用程序提供了消息处理功能。这些函数是 MessageLoop，IdleAction，PreProcessMenu 和 ProcessAppMsg。

二、空闲进程处理

空闲处理是让应用程序利用没有消息等待的空闲时间(包括用户输入)，如果没有消息等待处理，函数 MessageLoop 调用函数 IdleAction。

为了实现空闲进程，重载函数 IdleAction 实现真正的空闲进程。记住空闲进程是在用户不做任何事情时发生。因此空闲进程应该持续很短的时间，如果需要几十秒才能完成某一事情，则最好将其划分为几个进程。

函数 IdleAction 的参数(IdleCount)是 long 型的数值，函数 IdleAction 是在消息之间调用，可以用 IdleCount 参数在低优先权和高优先权之间选择，如果 IdleCount 达到一个较高的值，则有较长时间没有用户输入，所以较安全的是实现低优先权空闲处理。从函数 IdleAction 返回 TRUE，以便后来回头调用，应当经常调用基类中函数以便执行自己的过程。如果你正在编写 Windows NT 程序，你也可以使用多线程作背景过程。

2.1.7 关闭应用程序

用户通常选择菜单 File/Exit 或者按下 Alt+F4 键来退出 Windows 实用程序。这很重要，程序截获这个工作，并给用户机会存储未保存的文件。类 TApplication 能做到这一点。

一、改变关闭行为

类 TApplication 和其他所有类都具有或继承成员函数 CanClose，当应用程序试图关掉窗口时，它询问主窗口和文本管理窗口的 CanClose 函数，如果它们当中还有子窗口，则调用每个子窗口的 CanClose 函数，结果每个子窗口都依次调用自己的子窗口，如此类推。函数 CanClose 给每个对象一个准备关闭窗口的机会，如果必要，也提供给每个对象忽略关闭窗口*的机会，如果已完成清理过程，则其 CanClose 函数返回一个真值 TRUE。*

如果任何被调用的 CanClose 函数返回 FALSE，则忽略关闭操作。

二、关闭应用程序

函数 CanClose 系统给应用程序对象、主窗口、和其他任何窗口一个机会，或者为关闭窗口做准备，或者防止窗口被关闭。最后，应用程序对象完成关闭应用程序的过程。正常的关闭顺序是这样的：

(1) Windows 发出一个 WM_CLOSE 消息到应用程序主窗口。
(2) 主窗口对象的成员函数 EvClose 通过调用应用程序的成员函数 CanClose 予以响应。
(3) 应用程序对象的成员函数 CanClose 调用主窗口对象的成员函数 CanClose。