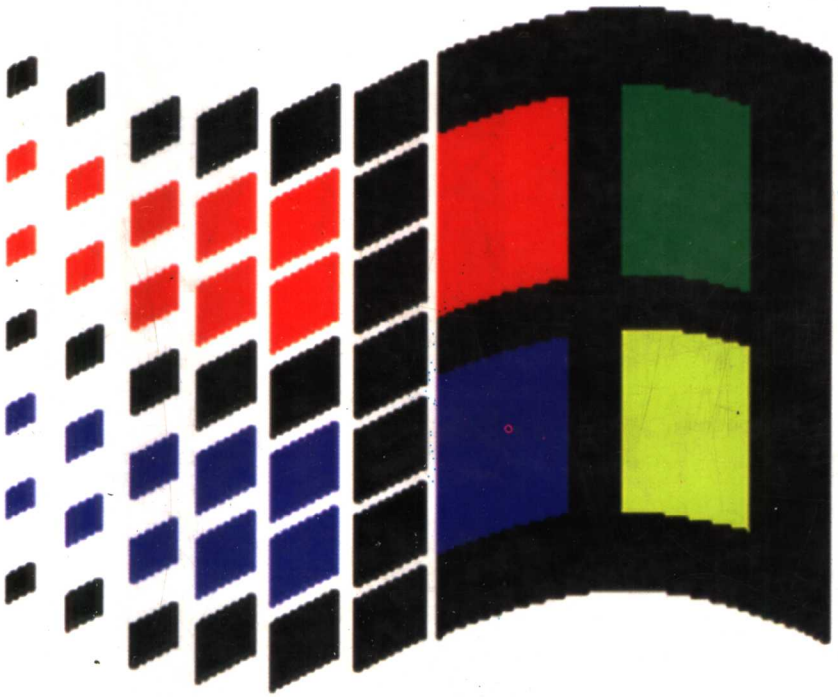
# Windows 3.1
# 程序开发指南

DISK INCLUDED

[美] Jeffrey M. Richter 著
席永斌 等译
田学锋 校



- 深入讨论Windows操作系统最高级和最强有力的性能及应用开发技术
- 随附软盘，提供多种Windows应用程序开发工具
- 介绍开发专业化Windows应用程序所需的技术和技巧
- 如何扩充Windows的能力，以有效开发应用程序
- 多个示例程序，充分展示Windows的应用开发技术

PHEI M&T BOOKS
電子工業出版社
Publishing House of Electronic Industry

# Windows 3.1 程序开发指南

[美] Jeffrey M. Richter 著
席永斌等 译
田学锋 校

电子工业出版社

(京)新登字 055

## 内 容 提 要

本书适用于那些想进一步了解复杂的操作环境的 Windows 开发者。现在有很多关于 Microsoft Windows 编程的书，大部分只是简单讨论了 Windows 的基本特征、能力和资源。本书包括其它书中很少或没有提到的一些高级特征。同时也示范了如何将基本的 Windows 模块连成完整的应用程序。

第一章介绍了 Windows 窗口类及其注册过程。第二章描述子类化和超类化概念及其编程实例。第三、四、五章分别介绍了 Windows 的对话框技术、设计自定义子控制技术以及设置打印机技术。第六、七章介绍了 Windows 任务、队列、钩概念及编程方法。第八、九章分别绘出了编制 MDI 应用程序的方法及实现拖放的技巧。第十章着重介绍了安装商用应用程序的方法和技巧。

对于每个专题，都深入到 Windows 内部进行透视，然后给出了示范用的实例源代码。明白这一点就能更好地掌握 Windows 提供的设备，使应用程序具有更强的适应性及更有效。本书的目的是提供给你开发专业 Windows 程序时必需的工具和知识。



**Windows 3.1 程序开发指南**

[美] Jeffrey M. Richter 著

席永斌 等 译

田学锋 校

责任编辑：胡毓坚

*

电子工业出版社（北京市万寿路）

电子工业出版社发行 各地新华书店经销

北京怀柔东晓印刷厂印刷

*

开本：787×1092 毫米 1/16 印张：35.25 字数：864 千字

1994 年 11 月第一版 1994 年 11 月北京第一次印刷

印数：10100 册 定价：65 元（含磁盘）

ISBN7－5053－2741－0/TP・860

# 译者的话

本书是写给那些有经验的 Windows 程序员的。书中讲述了 Windows 底层编程的各种技术和概念。几乎每一章都至少有一个程序例子示范其中的概念。其中有一些是完整的应用程序，可以不加修改地使用。这些源代码都经过调试和运行，无需修改就可以用于自己的 Windows 应用程序。有一天，这些程序会成为商业产品。

本书的第 1～4 章由席永斌翻译，5、6 章由刘颖翻译，第 7 章由章世嵩翻译，第 8 章由苏宝文翻译，第 9 章由代桂玉翻译，第 10 章以及附录 A、B、C 由田军翻译。全书由田学锋审校。

# 前 言

1990 年 5 月 22 日，推出了 Microsoft Windows 3.0 版，它的图形界面给 IBM-PC 系列计算机带来一场风暴。Windows 3.0 的成功有很多原因：改善的界面、增强的内存管理、丰富的应用程序。Windows 的一些技术特征确实令人印象深刻，不过我相信 Windows 的成功极大地依赖于第三方公司，他们投入时间和金钱来开发 Windows 应用程序。毕竟人们不是要买操作环境，他们要买的是能使他们工作更有效的应用程序。

Microsoft 自己也认识到了这一点，他们搞了一个“Microsoft Windows 预备版”。这个程序使 Windows 开发商在其商业版本发布 1 年前就用到了 Windows 3.0，这意味着许多商业应用程序可以和 Windows 3.0 同一天发布。没有这些第三方的产品，Microsoft Windows 不会有今天这样的需求量。

可能你会觉得 Windows 3.0 的预备版很不错，但它根本不能同 Windows 3.1 版相比。对应它的测试版，Microsoft 有一万以上的测试者。许多公司也参与其中，他们一直在开发自己产品的 Windows 版本。当 Windows 3.1 最终发布时，他们同时能更新自己产品的拷贝。

在此期间，许多公司放弃对 DOS 的支持，而把精力集中在 Windows 的开发上。现在 Windows 3.1 又有了多媒体的扩展，支持光笔识别，允许编写新型的应用程序。这些新技术把计算机带给了从未用过的人们。

另外，Microsoft 同时还在开发 Windows NT 和 Win32。Windows NT 是一个“Windows 的可移植版”，它能在不使用 Intel 微处理器的机器上运行。在 Microsoft 使 Windows 能在不同硬件平台上运行之后，其潜在市场将进一步扩大。

还有特别的 Windows 版本用于家用。近几年来，人们正将 Windows 用于录相机、CD 唱机、电视机、微波炉等等。看看这些强劲的功能，你一定觉得应该赶紧学习 Windows！对于 Windows 开发者的需求量每天都在增加。

本书是写给那些有经验的 Windows 程序员的，他们希望对这种复杂系统的一些最强有力的特征作更深一步的了解。几乎每一章都已至少有一个程序例子示范其中的概念。其中有一些是完整的应用程序，可以不加修改地使用，另外一些显示了怎样结合 Windows 的各种特征，使其功能要比各部分之和强很多。许多程序的代码段或模块解释了一些更基本的概念。这些段很通用，无需修改，就可以用于自己的 Windows 应用程序。所有的解释和源代码都经过调试和运行。有一天它们会成为商业产品。

写一段 Windows 应用程序并不难，但要做得很好并不容易。一个好的应用程序要非常注意细节并经过全面地测试。下面是在编程序时所需注意的一些东西：

1. 在最常见的屏幕分辨率下进行测试，显示对话框时最可能暴露问题。文本常被裁掉或绕到下一行去。我发现在用户将要使用的最低分辨率下进行对话框的设计是最佳的。
2. 同时在彩显和单显上测试。许多应用程序对颜色进行硬件编码操作，或使用缺省的颜色。在单显上，各项之间会不易分辨。比如一个字处理程序在一个黑色背景顶部显示黑色文本，这样很不好。

3. 仅在键盘上进行测试应用程序，当然几乎每个 Windows 用户都有一个鼠标或类似的指向设备，但一些人还是喜欢用键盘。这样使程序适应不同的需求。例如，要确保菜单的助记符键都是唯一的。
4. 对于对话框，要测试控制的标志顺序。这一点常被忽略。当我们使用一个应用程序的对话框，按下 Tab 键希望转向下一个控制时，其结果却转向另一个控制，我们常陷入其它一些控制之中。
5. 测试对话框的助记符键。要确保它们不要互相重复，并且助记符键确实与控制互相对应。
6. 与 Windows 的风格保持一致，这里有两点原因：

   A. 用户希望应用程序这样做，这样他们容易使用。

   B. Windows 的这种设计方法使程序员易于编写自己的应用程序。

   我见过一些公司为了使它们的应用程序有些不同的"样子和感觉"而花费了大量的时间和金钱。这样最终导致了一些难以捉摸的小错误，常需要对代码作些修改以纠正这些错误。
7. 和最终用户一起对程序进行测试，如果存在问题，可通过用户来发现。让一个开发者来发现他自己的问题通常较为困难。最终用户会以与你完全不同的方法来使用软件。但不幸的是，让一个测试者在发现错误时能够报告很困难。他们对自己缺乏自信并觉得寻求帮助显得太傻。我觉得作为开发者，应尽力同用户沟通。

   报歉我扯得太远了，但我对这类事确实有着很强烈地感受，无论如何，在设计一个 Windows 应用程序时注意前面所述的那些细节是非常重要的，这个概念将贯穿于本书之中。

## 编译实例源代码所需的工具

本书所有的示范应用程序都使用 Borland C++编译器（版本 3.1）编译，在 Windows 3.0和 3.1 下进行测试（除了那些需要 3.1 的挂接和拖放应用程序）。虽然程序都在 Borland 编译器下开发，但也可使用其它的 C/C++编译器。

在编码时我采用了一些惯例，这样使你易于接受。这些惯例包括：

1. 所有变量名采用标准 Hungarian 记法。
2. 所有全局变量都以一个下划线作为前缀。这样使一个函数更易于理解，所有不带下划线的变量要么是函数参数，要么是自动的或静态的局部变量。
3. 说明一个函数时，在开括号前边加一个空格。例如：

```
int FunctionName (int nX,int nY){}
                  ^ space character
```

而在调用这个函数时，就不使用这个空格了。例如：

```
nZ=FunctionName(nX,nY);
                ^ no space character
```

这是为了易于在源代码模块中查找函数说明。当在编辑器中搜索“FunctionName”(注意后面的空格)时，编辑器可找到函数本身而不是它的调用。

4. Windows 3.1提供了一个新的头文件 WINDOWS.H。这个文件中包含一些宏使得编写 Windows 应用程序更为容易和快速。这些宏帮助完成一些必要的造型，而不必在代码各处反复添加它们。在我的代码中，使用了许多宏。还有一些宏叫消息分流器(message cracker)，我只在 MDI 子应用程序中使用了这些宏，因为其它的应用程序不使用这些宏也很好懂。

如果开发一个复杂的窗口过程，我极力推荐使用消息分流器，但如果是一个简单的窗口过程，就不用了。Microsoft 宣称当一个应用程序从16位 Windows 移植到 Win32时，使用消息分流器将很简单。当然选择权在你。

# 目　录

# 第一章　分析一个窗口

在 Windows 编程中，窗口类(window class)起着重要的作用，它控制着窗口的外观和行为。在创建任何窗口前，必须给其以明确的定义。

在 Windows 中定义一个类的过程称为"注册类"(registering the class)。注册一个类并不创建任何窗口。实际上当定义一个类之后就可创建任意多个窗口了。

应用程序可以注册许多窗口类。Windows 将保持这些类的信息，直到这些注册窗口程序的所有实例都终止运行或这些被显示的注销。

## 1.1　注册一个窗口类

要创建一个窗口类，必须编写一段窗口过程；初始化一个 WNDCLASS 结构，再调用 RegisterClass 函数。结构 WNDCLASS 中最重要的成员有：

- lpszClassName　类的命名。
- lpfnWndProc　完成本类窗口操作过程的地址。
- hInstance　注册类的应用程序或动态链接库(DLL)的句柄或拥有者。

WNDCLASS 结构的其余成员定义了本类窗口的缺省属性。

如果不再需要一个窗口类，可调用 UnregisterClass 函数将其清除出内存。类在注销之前应将其所有窗口清除。当应用程序或 DLL 终止运行时，Windows 自动注销它们所定义的窗口类。

## 1.2　其它的类型

窗口类有三种类型：系统全局类(system global)，应用程序全局类(application global)和应用程序局部类(application local)。

### 1.2.1　系统全局类(System Global Classes)

系统全局类只能在启动系统环境时由 Windows 注册，应用程序不能注册系统全局类。在外壳程序(通常是"程序管理页"(Program Manager)，出现在屏幕上之前，Windows 会内部定义这些类的所有窗口过程，初始化每个过程的 WNDCLASS 结构，调用 RegisterClass 函数。只有在 Windows 终止运行时，才会注销系统全局类。也就是说这些类在 Windows 整个运行期间都是有效的。

要使用这些系统全局类，应调用 CreateWindow 或者 CreateWindowEx 函数。同时传入要创建的窗口类型的类名。下面列出了可在应用程序中使用的系统全局类：

BUTTON　　COMBOBOX　　EDIT　　LISTBOX

MDICLIENT　　SCROLLBAR　　STATIC

这些类常被称为“子控制”(Child Controls),不过除了被用于所有的应用程序之外,它们同你自己注册的类没有什么区别。

Windows 也注册一些在应用程序中隐含使用的窗口类。Windows 会在需要时自动创建这些窗口类的事例。比如,当调用 DialogBox 或 CreateDialog 函数时,Windows 创建一个对话框窗口,同时创建做为这个窗口的所有子控制。当应用程序最小化时,Windows 创建一个标题窗口,这时,出现在应用程序图标下的文字会画在 Windows 隐式创建的标题窗口内。

### 1.2.2 应用程序全局类

应用程序全局类由应用程序或经常由 DLL 来注册。通常在 DLL 的初始化代码中含有类的注册,而在终止时注销它。这意味着在 DLL 中注册的类只在 DLL 运行期间有效。

一旦注册一个类,任何应用程序都可以创建该类的窗口。你自己创建的自定义控制应注册为应用程序全局类。例如,要设计一个名为 SPIN 的窗口类,想要它在两个应用程序中使用,那么可在 DLL 中注册一个应用程序全局类,就可确保两个程序都可创建类 SPIN 的窗口了。

注册一个应用程序全局类的方法是在初始化 WNDCLASS 结构的 Style 成员时包含 CS _ GLOBALCLASS风格标志。

### 1.2.3 应用程序局部类

应用程序局部类是应用程序为了自己独特的需要而注册的。应用程序不能创建基于其它应用程序所注册的应用程序局部类的窗口。这些类在它们被注册和注销之间或在注册它们的应用程序终止之前是有效的。

一个类缺省注册为应用程序局部类,在 WNDCLASS 结构的 Style 成员初始化时不使用 CS _ GLOBALCLASS 标志。

### 1.2.4 同名的窗口类

不同的应用程序在注册窗口类时可能重名。当一个应用程序要创建一个窗口时,Windows首先试着匹配由同样应用程序所注册的局部窗口类指定的类名,若未找到匹配的应用程序局部类名,Windows 在同名的应用程序全局类中查找,若还没有查到,Windows 就在系统全局类中查找。

Windows 的这种查找类的方法,使得一旦定义了一个名为 EDIT 的应用程序局部类之后,应用程序所创建的所有 EDIT 窗口(包括在对话框样板中的那些)都属于应用程序局部类而不是 Windows 系统全局类。这种方法也许两个应用程序注册各自的 SPIN 类,并保证了应用程序会使用各自相应的 SPIN 类来创建窗口。

下表总结了当一个应用程序试图注册一个与已注册过的类同名的窗口类时调用函数 RegisterClass 时的结果:

| | 先前注册类如何 | | | |
|---|---|---|---|---|
| 应用程序要注册的新类型 | 系统全局 | 应用全局 | 同一程序内的应用程序局部 | 其它程序内的应用程序局部 |
| 应用程序全局 | 成功 | 失败 | 失败 | 成功 |
| 应用程序局部 | 成功 | 失败 | 失败 | 成功 |

逻辑上可以假设一个应用程序局部类能覆盖一个应用程序全局类。不过实际上为保持同以往版本的兼容性，Windows 并不允许这样。

## 1.3　窗口类的各部分

注册一个窗口类的第一步是初始化 WNDCLASS 的成员。下面是其中一些成员的说明：

- lpszClassName。WNDCLASS 的 lpszClassName 成员域指定了窗口类的名字。当请求 Windows 创建某个类的一个窗口时，它会把所要创建窗口的类名同已注册的类的名字相比较，如果找到了一个相匹配的名字，Windows 会创建这个窗口并返回其窗口句柄。如果找不到，Windows 返回 NULL。
- hInstance。WNDCLASS 的 hInstance 类指明了窗口类的所有者或创建者。如果是个应用程序，这个值将传入 WinMain 函数。如果是 DLL，则传入 LibEntry 函数。这个值可使 Windows 在这些类的创建者的所有事例终止时自动注销它们。
- lpfnWndProc。WNDCLASS 的 lpfnWndProc 成员指明了一个过程。此过程定义了本类所有窗口的行为。如果发生了什么或有所请求，这个窗口过程要被调用，同时传入描述发生事件或所需服务的消息。这个过程会执行一些操作来满足这些请求。

  Windows 提供一个名为 DefWindowProc 的缺省窗口过程。这个过程所处理的消息一般都是关于窗口的系统菜单和非客户区的行为。在《Microsoft Windows 程序员参考》中可查到一个可识别的消息的完整清单。同时这个清单描述了对应每个消息、过程 DefWindowsProc 所执行的操作。这些操作是 Microsoft 为一个窗口所定义的“标准”行为。

  在注册一个类之前把 WNDCLASS 的 lpfnWndProc 成员设为 DefWindowsProc 是可能的，这样就注册了一个具有标准或者缺省行为的窗口类。不过这样的窗口做不了任何令人感兴趣的事，所以一般不这么做。
- Style。WNDCLASS 结构的 Style 成员指明类的窗口的一些外观表现。WINDOWS.H中定义了用以注册类的有效的风格。列表如下：

| 类模式 | WINDOWS.H 中的标识符 | ID 值 |
|---|---|---|
| 绘图 | CS_VREDRAW | 0x0001 |
| | CS_HREDRAW | 0x0002 |

（续）

| 类模式 | WINDOWS.H 中的标识符 | ID 值 |
|---|---|---|
| | CS_OWNDC | 0x0020 |
| | CS_CLASSDC | 0x0040 |
| | CS_PARENTDC | 0x0080 |
| | CS_SAVEBITS | 0x0800 |
| | CS_BYTEALIGNCLIENT | 0x1000 |
| | CS_BYTEALIGNWINDOW | 0x2000 |
| 输入转换 | CS_KEYCVTWINDOW | 0x0004 |
| | CS_DBLCLKS | 0x0008 |
| | CS_NOKEYCVT | 0x0100 |
| | CS_NOCLOSE | 0x0200 |
| 类的类型 | CS_GLOBALCLASS | 0x4000 |

- 其余成员。WNDCLASS 结构的成员还有：cbClsExtra，cbWndExtra，hIcon，hCursor，hbrBackground 和 lpszMenuName。这些成员描述注册类的窗口所具有的缺省属性和行为。

## 1.4 Windows 如何存储窗口类

调用 RegisterClass 时，Windows 执行下列顺序的事件：

1. 首先检查在相同的模块中是否已注册过同名的类，如果是这样，RegisterClass 返回 NULL。
2. 分配一块内存空间用以存储类的有关信息。这块空间足以存放所有 WNDCLASS 的信息加上其 cbClsExtra 成员所指明的字节数。
3. 结构 WNDCLASS 的内容被拷入内存块，额外字节被初始化为0。这些额外字节专供应用程序使用，Windows 是不会用的。所有基于本类的窗口都可使用存于这些额外字节中的信息。这些额外字节的管理全部留给编程人员来做。
4. 分配内存存放类菜单的名字，lpszMenuName 字段被修改指向这个内存块。
5. 取得与 WNDCLASS 结构中 hInstance 成员所指明的数据实例句柄相关联的模块句柄。然后把这个模块句柄存入窗口类内存块中。这就是为什么 GetClassWord 函数使用 GCW_HMODULE 标识符而不使用 GCW_HINSTANCE 标识符的原因。
6. 类名被加入一个原子表(atom table)中，原子值被存入窗口类的内存块中。以后使用 GCW_ATOM 标识符调用 GetClassWord 函数可得到此值。
7. 最后，函数 RegisterClass 返回从上述步骤中获取的原子值。得到了原子值之后，它就能用于需要类名做参数的函数中了。例如：创建一个刚注册的窗口事例，可以这样：

```
atomClass=RegisterClass(&WndClass);
```

hWnd=CreateWindow(MAKEINTRESOURCE)(atomClass),NULL,
WS _ CHILD,...);

图1-1　显示了内部如何表示一个窗口类。

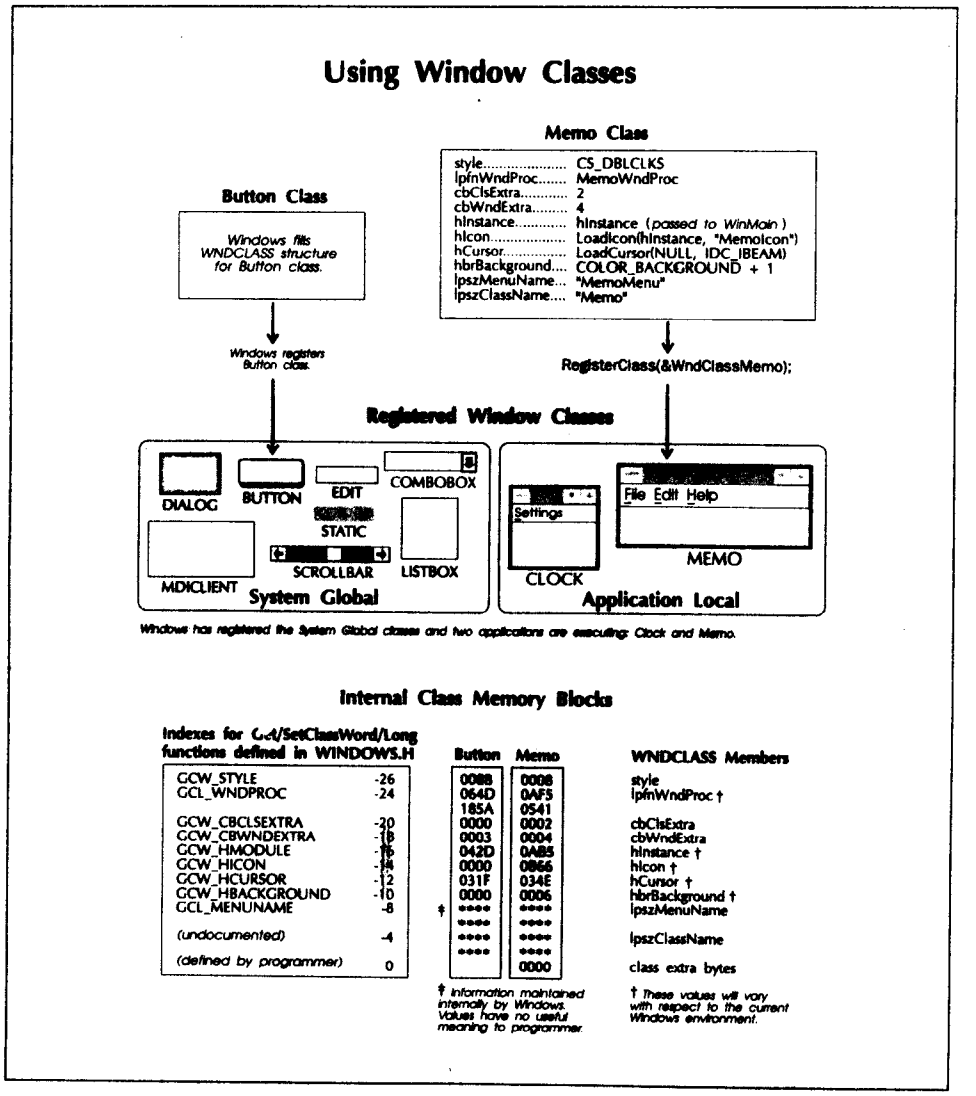



图1-1　使用窗口类

Windows 提供了在类注册之后能够得到或修改类结构中信息的函数。这里将说明这些函数。

GetClassInfo 能根据一个已注册的有关信息填充一个 WNDCLASS 结构。此函数的第一个参数是注册了本类的数据实例句柄。如果需要的是 Windows 系统全局类的信息，应将其值置为 NULL。这个参数也可以是一个程序模块实例句柄而不是数据实例句柄。本章中所

示范的 Voyeur 程序就是这样的。第二个函数或者是所需要的类的名字字符串的地址，或者是用宏 MAKEINTRESOURCE 所传入的类名的原子值。最末的一个参数是所需填充的 WNDCLASS 结构的地址。

使用 GetClassInfo 函数时应弄清楚一些事情。因为注册一个类时使用的数据实例句柄和菜单名没有被存放起来，所以 GetClassInfo 不能设置 WNDCLASS 的 hInstance 和 lpszMenuName 成员。另外，lpszClassName 成员不能用来指向一个有效字符串(因为类名是一个原子值存储)，所以它以传送给 GetClassInfo 的 lpszClassName 参数的相同值初始化。WNDCLASS 结构填满后，GetClassInfo 函数返回类的原子值。

不用说，GetClassInfo 函数不能返回类额外字节的内容。Windows 不提供在一次调用中就可修改所有类的内容的反函数。这个函数和其它类函数的不同之处就在于它不需要一个已存在的窗口句柄来存取数据。

函数 GetClassLong 和 GetClassWord 用以获取窗口类单个成员。要获取类成员的信息，必须提供一个从正关注类中创建的已存在窗口的句柄和在窗口类数据结构中的偏移量。WINDOWS.H 中定义了窗口类数据结构中各成员的偏移量。列表如下：

| 类成员 | WINDOWS.H 标识符 | 偏移量 |
|---|---|---|
| 原子值(Windows 3.0之后) | GCW_ATOM | −32 |
| 模式 | GCW_STYLE | −26 |
| 窗口过程 | GCW_WNDPROC | −24 |
| 类额外字节 | GCL_CBCLSEXTRA | −20 |
| 窗口额外字节 | GCW_CBWNDEXTRA | −18 |
| 所有者模块事例 | GCW_HMODULE | −16 |
| 图标 | GCW_HICON | −14 |
| 光标 | GCW_HCURSOR | −12 |
| 刷子背景 | GCW_HBRBACKGROUND | −10 |
| 菜单名 | GCL_MENUNAME | −8 |
| 类名 | (未定义) | −4 |
| 类额外字节起始 | (程序员定义) | 0 |

SetClassLong 和 SetClassWord 函数用来修改窗口类结构中单个成员。使用此函数需提供三个参数：已存在窗口的句柄，窗口类数据结构中偏移量，所需要的新值。上表中为 GetClassLong 和 GetClassWord 函数提供的标识符同样适用这两个函数。

在 Windows 3.1之前，这些函数都不对参数进行合法性的检查，也就是说可以传入任意的索引值并且改变任何类元素。而在 Windows 3.1中需要做大量的合法性检查。例如，现在 Windows 会检查参数中索引的值以确保它是一个预定义的索引值或者没有超过在注册类时所指明的类额外字节数。另外，当索引值是 GCL_MENUNAME、GCW_HMODULE 或 GCW_ATOM 时，Windows 会忽略任何对 SetClassWord/Long 函数的调用，因为这种调用会对系统造成灾难性的影响。附录 C 中给出了一个确保在访问类额外字节时使用有效索引值的方法。