程序员书库

冉林仓　编著

Windows API 编程

清华大学出版社

程序员书库

Windows API 编程

冉林仓 编著

清华大学出版社

北 京

内容简介

本书在介绍Win32 API函数调用的基础上，重点介绍如何使用Windows SDK API开发Win32动态链接库和应用程序，并结合进程管理、进程通信、钩子函数、窗口子类化、API HOOK、Internet Explorer开发、网络编程等介绍了API函数在这些方面的综合应用。

本书中的实例源代码可通过http://www.tupwk.com.cn/downpage/index.asp下载。

本书主要面向熟悉Windows开发且有一定编程基础的中高级用户，旨在帮助用户提高系统编程的能力。

版权所有，翻印必究。举报电话：010-62782989　13501256678　13801310933
本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目(CIP)数据

Windows API编程/冉林仓编著. —北京：清华大学出版社，2005.5
(程序员书库)
ISBN 7-302-10571-5

Ⅰ.W…　Ⅱ.冉…　Ⅲ.窗口软件，Windows—软件接口—程序设计　Ⅳ.TP316.7

中国版本图书馆CIP数据核字(2005)第014344号

出 版 者：清华大学出版社　　**地　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
组稿编辑：胡伟卷　　**文稿编辑**：刘金喜
封面设计：王　永　　**版式设计**：康　博
印 装 者：北京鑫霸印务有限公司
发 行 者：新华书店总店北京发行所
开　本：185×260　**印张**：22　**字数**：508千字
版　次：2005年5月第1版　2005年5月第1次印刷
书　号：ISBN 7-302-10571-5/TP・7169
印　数：1～4000
定　价：32.00元

前　　言

API 函数是 Windows 提供的最基本的开发接口，所有的开发工具提供的类库或者辅助函数库都是基于这个函数接口实现的。使用 API 函数可以突破编程语言的固有局限性，方便对系统功能进行扩展，从而提高开发效率和程序的运行效率。

本书共分 10 章，主要内容如下：

第 1 章主要介绍 Win32 API 函数接口的基本概念，以及利用 Win32 API 对程序进行跟踪调试。还介绍了如何使用各种开发环境调用 Win32 API 函数及相关调用注意事项。

第 2 章主要介绍动态链接库的基本功能及其各种调用加载方法。

第 3 章主要介绍进程的基本概念以及进程创建、遍历、关闭进程、进程监视、进程代码注入等方面的内容。

第 4 章介绍钩子函数和窗口子类化的系统挂接，以实现程序功能扩展。

第 5 章介绍各种进程间数据通信的方法，这些方法可以应用于同一进程的不同模块。

第 6 章主要介绍 Win32 API 和本机 API 函数的拦截，并给出相关应用实例。

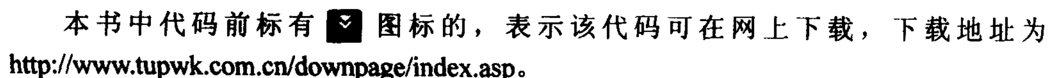
第 7 章主要介绍 Winlogon 编程，对各个进程的运行、窗口创建进行监视。

第 8 章主要介绍 Internet Explorer 开发，内容涉及 BHO 和网络实名开发，以及实名的拦截。

第 9 章主要介绍 DDK API 函数的应用，主要涉及硬件端口读写。

第 10 章主要介绍原始 IP 和以太数据包的发送和接收，同时介绍了 WinPCAP 开发包的应用，并通过一个混杂模式侦听的服务程序介绍了如何实现局域网内部的密码侦听。

本书不是介绍每一个 API 函数的调用方法，主要介绍 Win32 API 在系统开发方面的应用。

本书中代码前标有  图标的，表示该代码可在网上下载，下载地址为 http://www.tupwk.com.cn/downpage/index.asp。

本书主要由冉林仓编著，下列人员在本书编写过程中给予了很大帮助，他们是尹建民、薛年喜、刘伟、徐日强、赵磊、张江涛、李志伟、刘旭、赵海云、宋利军、刘咏、郑砚、许社村、黄丽娜、唐兵等，在此一并表示感谢。

由于时间仓促，加上作者水平有限，不妥之处希望读者批评指正。

作　者

目　　录

第 1 章　Win32 API 编程简介……1
1.1　Windows API 概述……1
1.2　Windows API 函数帮助的使用……3
1.3　Windows API 的组成……5
1.4　Windows API 调用的注意事项……6
1.5　程序的调试信息输出……11
1.6　Visual Basic 中调用 Windows API 函数……16
1.7　使用汇编语言调用 Win32 API……29
1.8　使用 PowerBuilder 调用 Win32 API 函数……31
1.9　.NET 框架下调用 Win32 API 函数……36
1.10　小结……42
1.11　思考题……42
1.12　练习题……42
第 2 章　动态链接库……44
2.1　动态链接库简介……44
2.2　用程序加载动态链接库……47
2.3　动态链接库的入口点……50
2.4　动态链接库的数据共享……52
2.5　Windows API 调用显式加载和隐式加载……53
2.6　动态链接库与插件实现……55
2.7　动态链接库的延迟加载……58
2.8　小结……63
2.9　思考题……63
2.10　练习题……63
第 3 章　进程……64
3.1　进程概述……64
3.2　进程的定向输入和输出……65
3.3　进程运行的监视……67
3.4　进程枚举……78
3.5　进程的终止……81
3.6　进程与端口……85
3.7　进程代码的注入……92
3.8　缓冲区溢出实现代码注入……98

试读结束：需要全本请在线购买：　www.ertongbook.com

3.9 小结 ······113
3.10 思考题 ······113
3.11 练习题 ······113

第 4 章 钩子函数和窗口子类化 ······114
4.1 钩子函数 ······114
4.2 键盘钩子的应用 ······117
4.3 使用钩子函数截取 Windows 密码 ······123
4.4 钩子函数与窗口子类化 ······128
4.5 Shell 子类化监视系统事件 ······137
4.6 小结 ······149
4.7 思考题 ······149
4.8 练习题 ······149

第 5 章 进程间通信 ······150
5.1 只启动一份程序实例 ······150
5.2 使用共享内存实现进程间通信 ······151
5.3 使用窗口消息实现进程间通信 ······155
5.4 使用邮槽实现进程间通信 ······160
5.5 使用剪贴板实现共享 ······164
5.6 使用管道实现进程间通信 ······168
5.7 驱动程序和 Win32 应用程序之间的数据通信 ······172
5.8 小结 ······176
5.9 思考题 ······176
5.10 练习题 ······176

第 6 章 API HOOK ······177
6.1 API HOOK 综述 ······177
6.2 使用增强图元文件实现屏幕截获 ······185
6.3 用 Detours 实现 API HOOK ······191
6.4 内核态应用程序的拦截实现 ······209
6.5 基于 SPI 实现的 HTTP Tracer ······224
6.6 Windows 9x 环境目录隐藏 ······236
6.7 小结 ······247
6.8 思考题 ······247
6.9 练习题 ······247

第 7 章 WinLogon 编程 ······248
7.1 WinLogon 概述 ······248
7.2 WinLogon 通知包的创建 ······249

7.3 GINA 动态链接库编程……251
7.4 WinLogon 进程的注入……263
7.5 小结……277
7.6 思考题……278
7.7 练习题……278

第 8 章 Internet Explorer 编程……279
8.1 BHO 插件扩展……279
8.2 HTTP URL 的跟踪……286
8.3 网络实名及其实现……290
8.4 小结……291
8.5 思考题……291
8.6 练习题……291

第 9 章 底层开发……292
9.1 基于 Windows NT 操作系统的端口直接读写……292
9.2 用本机 API 开发 Native NT 应用程序……303
9.3 用户模式应用程序运行 Ring0 特权指令……306
9.4 小结……319
9.5 思考题……319
9.6 练习题……319

第 10 章 网络编程……320
10.1 主机扫描的实现……320
10.2 WinPCAP 的使用……325
10.3 局域网范围内的密码侦听……329
10.4 小结……341
10.5 思考题……341
10.6 练习题……341

参考文献……342

第 1 章　Win32 API 编程简介

本章主要介绍 Win32 API 函数的基本概念、主要组成，及如何使用各种编程调用这些 API 函数。

1.1　Windows API 概述

编程人员在开发程序的时候，并不是完全从零开始的，操作系统本身在提供进程管理、设备管理、文件管理、内存管理的同时，也为开发人员提供了丰富的应用编程接口。这些接口就是我们所说的 API(Application Programming Interface，简称 API)。几乎所有的程序开发都是基于或者依赖这些接口实现的。

DOS 功能调用和 BIOS 中断是最早使用的应用编程接口，通过这些中断调用可以非常方便地实现对文件、磁盘、打印机、显示卡及通信设备的存取。尽管采用 C、Fortran 及 Basic 等中高级语言不需要直接和中断打交道，但是归根结底这些语言实现的静态库和目标文件最终还是借助于中断实现的，生成的二进制代码经过反汇编之后，可以看到大量的中断调用。可以说在 Windows 95 问世之前，DOS 功能调用和 BIOS 中断是编程人员主要采用的编程接口。

Windows 95 横空出世彻底把 DOS 赶出了历史舞台，从此几乎所有的 PC 用户都迈入了一个崭新的视窗世界。Windows 32 位操作系统要比以前 DOS 操作系统先进、复杂，功能相对比较单一的中断调用已经无法满足对 32 位复杂应用程序开发的需要。在这种情况下，微软公司提供了一种新的 Win32 API 接口，以便开发人员在开发程序时能够使用 Windows 家族操作系统强大的功能。

这种崭新的 API 编程接口能够实现自上而下的高度兼容。这样在最早的 Windows 95 环境下开发的应用程序几乎不用重新编译，就可以畅通无阻地在 Windows 2003 乃至未来的 Longhorn 操作系统环境下运行。尽管基于 Windows 9x 内核和 Windows NT 内核的操作系统在实现方面存在着一些差异，比如一些函数只能用于基于 Windows NT 内核的平台，一些函数对 Windows NT 内核的系统进行更好的优化和增强，但是就大多数 API 函数而言，这些函数基本上是一致的。Windows XP 的出现终结了这两种内核同时并存造成的各种困惑，从此 Win32 平台的开发实现了统一。

Win 32 提供的 API 接口最初是基于动态链接库输出函数实现的，应用程序可以间接或者直接地调用这些 API 函数，实现文件的存取和窗口消息的传递。在 Win32 环境下的所有应用程序都要间接或者直接地调用 Windows 提供的 Win32 API 函数。对一些编译成本机代码的应用程序，用户可以借助于 Visual Studio 提供的 Depends 或者 DumpBin 工具查看这些程序间接或者直接调用了哪些动态链接库的 API 函数。而利用

Visual Basic、Power Builder 开发的应用程序都会依赖支持其运行的辅助动态链接库，这个动态链接库实现了对大部分 Win32 API 函数的封装，这些语言在开发时尽管让用户感觉不到 Win32 API 的存在，但是这些程序的运行千真万确离不开 Win32 API 的支持。同样，对于 Java 开发环境和.NET 框架下开发的应用程序也没有完全脱离 Win32 而存在，因为这些解释运行环境是在 Win32 下开发实现的，它本身需要 Win32 API 的支持，特别是.NET 框架提供的互操作性。可以发现.NET 应用程序的开发和 Win32 API 还存在着千丝万缕的联系。

基于动态链接库输出函数实现的 API 接口相对是比较复杂的，因为这些 API 接口只有通过文档一种方式才能让用户熟悉和使用，没有文档的动态链接库输出 API 接口几乎是没有办法使用的，因为用户除了反汇编之外，没有办法知道这个 API 函数调用时究竟需要几个参数，这些参数分别是什么类型，返回什么类型的值。另外动态链接库输出函数的使用和该文件的存放路径有着直接关系，它决定着用户具体使用哪一个路径下的动态链接库。由于大部分动态链接库为了使用方便，都存放在 Windows 根目录或者系统目录下，这样很容易造成不同版本的同名动态链接库交叉覆盖。一个程序卸载时会把覆盖后的动态链接库也给卸载掉，这样其他程序在运行的时候就会因为找不到这个动态链接库，导致无法正常运行，这就是臭名昭著的“动态链接库地狱”。可以说基于动态链接库输出函数实现的 API 接口无法解决这个问题，这样微软又提供了另外一个基于组件对象模型(COM)实现的编程接口。

基于组件对象模型实现的编程接口相对纯粹的 API 函数调用有着先进性，首先组件库本身包含了类型库说明，通过类型库，用户不依赖文档就可以了解组件提供了哪些方法、属性和事件，这些接口的方法、属性及事件的参数类型和个数都可以通过类型库得到。这样，调用的复杂性就被大大简化。另外客户端程序调用组件提供的接口是借助于组件的类型标识符实现的，这些标识信息连同组件的存放路径在组件注册的时候就被写入到系统注册表了。系统会从注册表信息确定调用哪一个版本的组件，加载哪一个目录下的组件模块。组件的这种调用方法使得因为组件重名导致的版本冲突交叉覆盖的机会大大减少，甚至不复存在。所以这种 COM 接口类型的 API 越来越被系统所采纳，成为二进制代码复用的最好方法。事实上，.NET 框架的实现也是在组件对象模型的基础上发展起来的。

这两种接口方法是 Windows 提供 API 接口的主要方法，它们可以应用在所有支持动态链接库调用和 COM 组件调用的场合，与用户采用的编程语言无关，对函数的调用也不存在太多的性能差异，运行效率是一样的。由于微软提供的编程帮助文档和软件开发包(SDK)是以 C(C++)语言规范提供的，所以对于 C 语言的用户可能调用起来更方便些。当然各种开发环境对 Win32 API 支持的程度不一样，一些语言在使用某些函数的时候会存在一些限制，比如 PowerBuilder 无法提供对指针类型的支持，在调用钩子函数和提供枚举函数、窗口过程的回调函数时就存在一些限制，不过这些问题可以借助于多种语言的混合编程来解决，以此来发挥各种开发工具的自身优势，实现优势互补。

Windows 提供的 API 接口同时给编程人员功能模块分工、实现和功能扩展提供了一个很好的思路，通过编写动态链接库和 COM 组件，可以更好地实现跨语言、跨平台的二进制代码复用。

在后面的章节中，前面部分主要介绍动态链接库函数的调用，同时也会给出动态链接库的编程方法。后面的部分会介绍组件对象的调用方法，并给出 COM 组件的编写方法，但是本书不是介绍 COM 原理的，所以在 COM 接口实现原理上不会占用太多的篇幅。本书的例子除了介绍各种语言调用 API 的方法之外，主要采用 C 和 C++ 语言，偶尔会使用汇编语言和 Visual Basic。用户如果需要把这些代码移植到 Power Builder、.NET 框架环境下，可以仔细参考第 1 章的方法举例。

本书用到的所有 API 函数，在 MSDN 文档中都会有详细说明，用户在参考这部分内容的时候，最好安装最新并且完整的 MSDN 文档。

1.2　Windows API 函数帮助的使用

包括微软自己在内，谁也无法说清楚 Windows 到底提供了多少个 Windows API 函数，面对众多的 API 函数，争论用户了解多少 Windows API 就会变得毫无意义。在利用 Visual Basic 编程时，用户往往会借助于 API Viewer 这个工具来添加对函数、结构以及常量类型的声明，但是用户应该知道这个文件只提供了 1551 个函数声明，它并没有覆盖所有的 Windows API。MFC 尽管是一个庞大复杂的类库，但是它也一样，只是有限的 Windows API 的封装，也不是 Windows API 的全集。一些微软没有公开的 API 函数和升级操作系统提供的增强 API 函数，还需要用户手工来调用和声明。不过，这些函数已经足够我们使用了。

学习使用 Windows API，并不是要刻意来记忆这些函数的具体声明，而是要学会如何在需要的时候，在庞大的 MSDN 库中和浩瀚的 Web 上找到这些函数的蛛丝马迹。

应该说 MSDN 是最好的老师，尽管各种开发工具包括 Delphi、PowerBuilder 甚至 Java 都提供了自己的帮助文档，但是从可用性、易用性、丰富性评价，没有任何一个其他公司的产品能够超越微软公司的 MSDN。所以无论用户是从事 Delphi 还是 PowerBuilder 的开发，手头上准备一套 MSDN 的帮助文档是完全必要的。很多这些语言无法解决的问题都可以在这里找到答案。

当然用户如果需要了解一些 API 函数的使用举例，那么最好的方法是使用 Google 搜索引擎，用户只要在 Google 主页的搜索框中粘贴上 API 函数的名字，相信 Google 一般不会让用户失望。当然了，网上 API 函数应用的例子的鼻祖最终还是来自 MSDN。用户在使用 Web 检索之前，最好还是先用 MSDN 提供的搜索碰碰运气。尽管这种原汁原味的文档让一些英文比较差的用户读起来比较吃力，但是借助于联机翻译工具，还是要比那些"驴头不对马嘴"的翻译强得多。

MSDN 提供的函数的帮助一般包括下面几个部分，如图 1-1 所示。

GetTickCount

The **GetTickCount** function retrieves the number of milliseconds that have elapsed since the system was started. It is limited *to the resolution of the system timer*. To obtain the system timer resolution, use the GetSystemTimeAdjustment function.

```
DWORD GetTickCount(void);
```

Parameters

This function has no parameters.

Return Values

The return value is the number of milliseconds that have elapsed since the system was started.

Remarks

The elapsed time is stored as a **DWORD** value. Therefore, the time will wrap around to zero if the system is run continuously for 49.7 days.

If you need a higher resolution timer, use a multimedia timer or a high-resolution timer.

To obtain the time elapsed since the computer was started, retrieve the System Up Time counter in the performance data in the registry key **HKEY_PERFORMANCE_DATA**. The value returned is an 8-byte value. For more information, see Performance Monitoring.

Example Code

The following example demonstrates how to handle timer wrap around.

```
DWORD dwStart = GetTickCount();

// Stop if this has taken too long
if( GetTickCount() - dwStart >= TIMELIMIT )
    Cancel();
```

Note that TIMELIMIT is the time interval of interest to the application.

For an additional example, see Starting a Service.

Requirements

Client: Included in Windows XP, Windows 2000 Professional, Windows NT Workstation, Windows Me, Windows 98, and Windows 95.
Server: Included in Windows Server 2003, Windows 2000 Server, and Windows NT Server.
Header: Declared in Winbase.h; include Windows.h.
Library: Use Kernel32.lib.

See Also

Time Overview, Time Functions

图 1-1　MSDN 帮助举例

首先，帮助文档会给出函数能够实现哪些功能。

接着函数会以黑粗体字给出函数的 C++调用规范，如果用户采用 C++或者 C 开发，这可能是用户最需要复制粘贴到代码中的部分。

Parameters 部分给出函数调用参数的调用说明，包括各个参数的具体类型和含义。如果调用函数后参数值不会发生变化，文档用[in]来标识；如果调用后一些参数的值会发生变化，或者说将返回新值，文档会用[out]来标识。

Return Value 部分主要介绍返回值的具体类型和含义。

Remark 主要介绍函数调用的具体注意事项。

对于一些函数，文档会在 Example Code 部分给出使用举例。

Requrements 部分指出了函数能够应用的场合，特别要注意一些比较新的 API 函数无法在早期的操作系统环境下得到支持。

另外 Header 部分说明了这个函数是在哪个头文件中定义的，对 C++的用户需要使用＃include <xxx.h>包含该部分指出的头文件。如果程序是通过应用程序向导创建的，可能一些头文件的引用已经被开发环境包含进去了。

同样 Library 指出了程序在链接时需要链接的库文件，用户可以对项目进行设置，把该库文件加入链接库列表中，也可以使用#pragma comment(lib,"xxx.lib")预编译指令实现对该库文件的链接。

对于非 C++的用户，最幸运的应该算是 Visual Basic 的用户，Visual Basic 开发环境为用户提供了一个 API Viewer 的工具软件，用户可以直接借助于这个工具查找符合

试读结束：需要全本请在线购买： www.ertongbook.com

Visual Basic 语法的函数、结构和常量声明，并把这些声明直接粘贴到代码中，如图 1-2 所示。

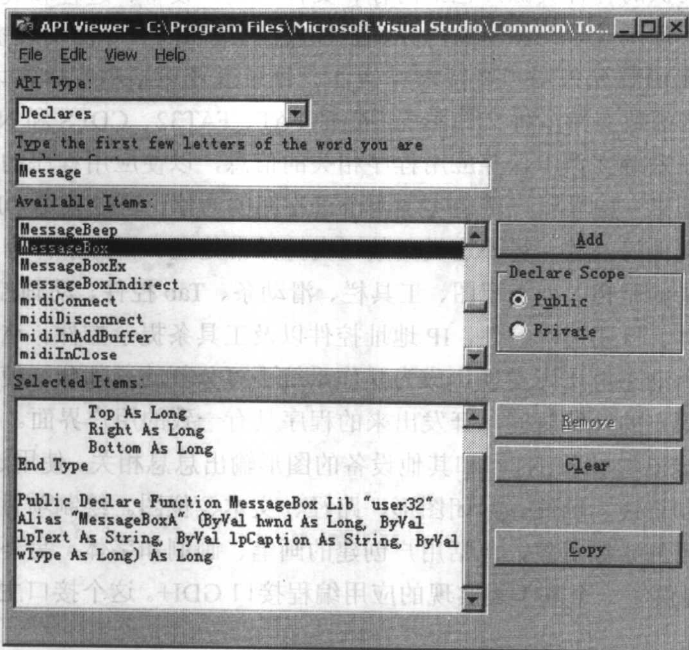


图 1-2　Visual Basic 提供的 API 查看器

对于使用 Delphi、PowerBuilder 及.NET 框架应用开发的用户，使用 Win32 API 函数的确是件郁闷的事情，一方面这些工具自身提供的库函数还不足以应付所有的 Windows 编程，在某些情况下，还必须调用这些 Windows API 函数，然而这些语言既没有提供像 C 语言使用的头文件和库文件，也没有提供 Visual Basic 这样的 API 转换工具，所有的翻译转换的工作都需要用户自己手工来完成，这的确是一件让人难以接受的事情。这可能也是众多 C 程序员不愿意使用这些工具开发的重要原因吧，因为这些工具在提供快速应用开发的同时，却牺牲了 C/C++ 语言的灵活性。

1.3　Windows API 的组成

尽管存在着数目繁多的 Win32 API 函数，但是还是可以根据它们提供的功能进行分类的。Win32 API 按照它们提供的功能，大致分为下面几个大类：

- 基础服务类
- 公共控件库
- 图形设备接口
- 网络服务
- 用户界面
- Windows Shell 函数

在这 6 个大类中，基础服务类是用户需要重点掌握的，因为这一类函数允许用户存取操作系统提供的各种资源，包括内存、文件系统、设备、进程和线程。比如用户可以使用内存管理函数分配和释放内存、通过进程管理和同步函数实现多任务的调度。文件输入和输出函数允许用户直接操作文件、目录以及包括串行口在内的各种输入/输出设备，这些函数支持多种文件系统，包括 FAT、FAT32、CDFS 和 NTFS 系统。注册表函数能够在系统文件中保存应用程序相关的信息，以便应用程序的新实例能够获取和使用这些信息。如果用户希望实现两个进程间的通信，可以采用 DDE、管道、邮槽、内存共享文件等函数。另外这类函数还支持异常处理、日志事件等。

公共控件指的是树、列表视图、工具栏、滑动条、Tab 控件、动画控件、Rich Edit 控件、月历控件、日期选取控件、IP 地址控件以及工具条提示控件。这些控件被集成到 Shell 中，由动态链接库实现，成为操作系统不可分割的一部分。使用这些控件可以成倍地节省用户的开发时间，开发出来的程序具有一致的用户界面。

图形设备接口与显示、打印和其他设备的图形输出息息相关。使用这些函数接口，可以方便地绘制直线、曲线、封闭图形、路径、文本、位图。绘制项目的颜色和风格依赖于用户采用的绘制对象，包括用户创建的画笔、画刷和字体。另外微软还对 GDI 进行了改进，提供了一个基于类实现的应用编程接口 GDI+。这个接口主要面向 C/C++ 用户。

网络函数可以用于实现网络上的不同计算机之间的相互通信，实现网络信息资源共享。

用户界面函数主要用于维护和操作 Windows 中的各种常用控件，实现窗口消息驱动。

Shell 函数主要用于和 Shell 进行交互，实现 shell 操作。

除了上面这 6 大类函数之外，Windows 还提供了其他 API 接口，这些接口尽管不是操作系统必需的，但是和用户开发关系密切。比如使用 ADO、ODBC 接口实现数据库存取，使用 XML SDK 解析操作 XML 文件，使用 DirectX 接口实现图像和多媒体开发，利用 Windows 和 Internet Explorer 提供的接口实现 Web 开发等。

1.4 Windows API 调用的注意事项

前面提到，我们平常所说的 Windows API 都是通过动态链接库输出函数实现的。应用程序在调用这些函数的时候，会按照如下的顺序查找这些动态链接库文件。

(1) 应用程序所在的当前目录

(2) Windows 目录

(3) Windows 系统目录

(4) 系统环境变量指示的目录

如果在这些目录中找不到，程序就会出现运行异常。所以，为了保证程序的正常运行，用户最好把这些动态链接库放在程序所在的目录中，这样也可以避免各种潜在的覆盖冲突。

调用 API 函数最简单的方法是采用 C 和 C++编码，这样也可以充分发挥这些语言的性能优势。由于微软提供的 Windows Platform SDK 是基于 C/C++语言给出的，所以使用这些语言几乎不需要做任何转换。

使用其他语言可没有那么幸运了，参数变量的类型定义、常量的定义、函数的声明给用户添加了最大的工作量。特别是参数变量的类型定义，对于普通用户来说，的确是个难啃的骨头。

各种语言的变量类型的定义差别很大，比如 C 语言中字符串和 Visual Basic 中的字符串就存在着质的区别。C 语言的字符串是以零字符结尾的。如果是宽字符集，则使用两个字节零字符结尾。而 Visual Basic 中的字符串采用 UNICODE 编码，是宽字符集，而且并非是以零字符结尾而是以字符串长度开头，这种类型和 COM 中的 BSTR 数据类型是一致的。而 Windows 提供的 API 函数对于有字符串的参数类型的情况，一般都提供了两个版本，分别对应 ANSI 和 UNICODE 类型。这些函数一般都会以 A 和 W 字符结尾标识。

比如 MessageBox 函数。这个函数声明如下：

```
int MessageBox(HWND hWnd,LPCTSTR lpText,LPCTSTR lpCaption,UINT uType);
```

其实，这个函数实际上是不存在的，我们调用这个函数时实际上是调用了 MessageBoxA 和 MessageBoxW 其中之一。这两个函数才是 User32.DLL 的输出函数，在调用这两个函数时，用户应该分别使用不同的参数，MessageBoxA 使用 ANSI C 字符串，而后者使用 WCHAR 类型的 C 字符串。

用户可以使用 Visual Studio 提供的 Depends 工具了解这一点。如图 1-3 所示即是使用该工具查看 Windows 2000 系统下 User32.DLL 输出函数的情况，这个动态链接库是一个系统文件，它位于 system32 目录下。如果在 Windows 98/Me 系统下它应该位于 system 目录下。对于不同的操作系统，显示的结果是不一样的。在 Windows98/Me 环境下，这个动态链接库和 GDI32.DLL 以及 Kernel32.DLL 是 Windows 的核心文件。这 3 个库不依赖其他任何文件。然而在基于 NT 内核的操作系统下，这 3 个动态链接库是借助于 NTDLL.DLL 系统文件实现的。这 3 个文件只是函数调用的跳板，使用这 3 个文件主要是考虑与 Windows 9x 系统保持兼容，因为在 Windows 9x 环境下输出使用的 API 函数，在 Windows NT 内核的系统中都会通过这 3 个文件实现其对应版本。这样，在 Windows 9x 下编写的应用程序，几乎不用修改就可以在 Windows NT 内核的系统下运行。在 Windows NT 环境下，真正的 Win32 实现是通过调用 NTDLL.DLL 提供的本机 API 实现的。通过图 1-3 左边的部分很容易了解这一点。

图 1-3 右边视图是这个函数的输出函数。通过这个视图，我们可以看到几乎所有与字符串相关的函数都存在 A 和 W 字符后缀的两个版本。

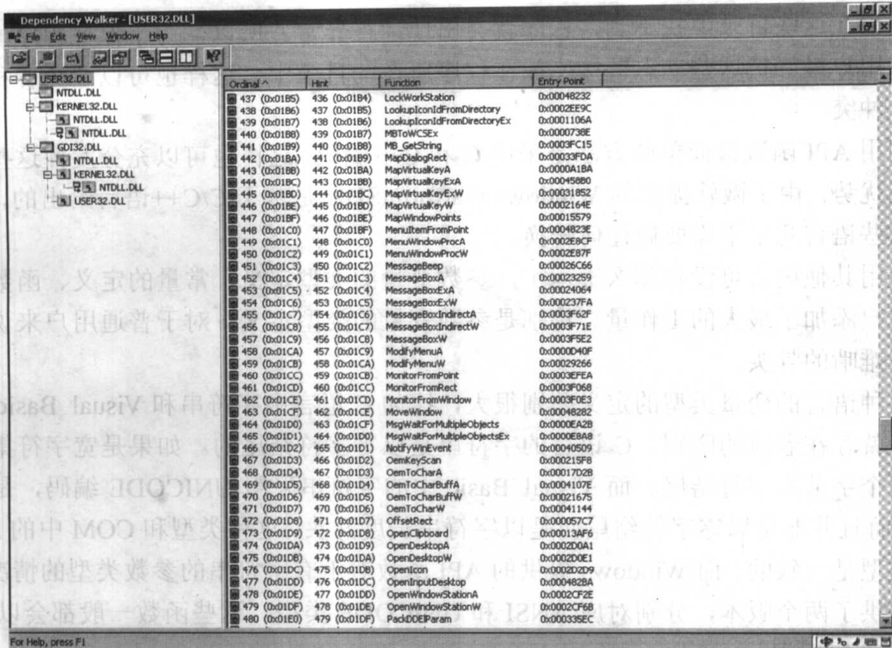


图 1-3　User32.DLL 动态链接库的 API 输出

不过话不能说绝，也有些函数只有 ANSI 版本，比如 Kernel32.DLL 输出的 GetProcAddress 函数，这是因为这类函数不存在宽字符串调用的情况。也有一些函数尽管调用时不需要字符串参数，但是也提供了 ANSI 和 UNICODE 两个版本，比如 User32 提供的 SetWindowLongA 和 SetWindowLongW 函数。当然用户还可以使用 BumpBin 工具查看动态链接库的输出。格式如下：

```
dumpbin /exports user32.dll |more
```

一般地，除非特殊需要，在编程的时候，用户只需要使用其中一个版本就足够了。对于 C++用户而言，究竟使用哪个版本的函数，一方面要看用户调用的方式，另一方面还要看用户的编译设置是否采用 UNICODE 编码。一般地，只要在程序首部添加 #define UNICODE 预编译指令，编译器会自动链接和使用 UNICODE 版本的函数系列。对于其他语言，则需要明确指出使用哪一个版本，比如下面的 Visual Basic 函数声明就明确指出使用 MessageBoxA，这里的 A 是不能省略的，去掉 A 或者去掉 Alias "MessageBoxA"都是万万不能的，否则就会产生如图 1-4 所示的运行错误。当然，如果用户不想使用函数别名，可以直接使用 MessageBoxA 函数名。

```
Public Declare Function MessageBox Lib "user32" Alias "MessageBoxA"
(ByVal hwnd As Long, ByVal lpText As String, ByVal lpCaption As String,
ByVal wType As Long) As Long
```

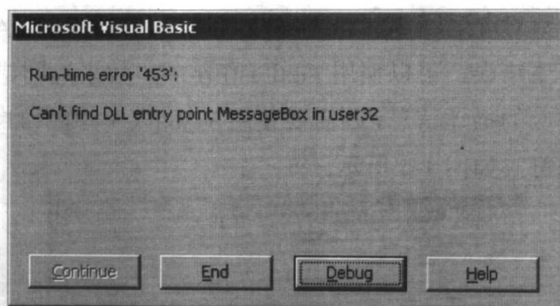


图 1-4　函数名匹配造成调用异常

很明显，这个错误是因为没有找到 MessageBox 函数入口造成的，实际上 User32 压根就不存在 MessageBox 这样的输出函数，它只有上面提到的两个版本。

当然用户完全可以采用下面的方式调用，除了多出来的 A 字符看起来不舒服外，运行完全正常。

```
Private Declare Function MessageBoxA Lib "user32" (ByVal hwnd As Long,
ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long)
As Long
MessageBoxA Handle, "Hello World", "VB", MB_OK
```

在调用 API 函数的时候，最大的困惑是指针的使用，C/C++和 Delphi 提供了对指针的最好支持，而其他编程语言可能压根就不存在指针的概念。其实指针是一个 32 位的无符号变量，它可以使用 32 位无符号长整型来声明。即便用户使用有符号的长整型来声明，调用也不会错误，关键是参数的字长必须保证是 32 位，8 位或者 16 位字长是不能使用的，因为这种类型在调用时会把一个 16 位的变量入栈，而函数调用后会自动按照 32 位恢复堆栈，这样会造成堆栈异常。

同样，上面这种规则也适用于各种句柄，在 Win32 API 中存在着各种各样的句柄，这些句柄都是 32 位的，用户可以使用 32 位无符号整型变量声明。

函数指针是最让人头疼的，因为在 Win32 中存在着大量的枚举函数、钩子函数、窗口过程、回调函数、线程函数，这些函数都需要用户定义一个函数过程，然后使用这个函数过程的入口地址作为参数来调用。然而并不是所有的编程语言都能够实现这种转换，对于 Visual Basic，由于该语言可以把函数实现写入到一个模块中，用户可以使用这个函数名作为地址来使用。然而对于 PowerBuilder 就没有那么幸运了，它无法实现定义一个回调过程，而是让这些 API 来使用。这样 PowerBuilder 在使用这类函数时就存在一些限制，它无法提供一个线程函数、钩子过程、枚举函数让这些 API 来调用。这种限制的确让用户遗憾。弥补这种遗憾的方法只能是采用混合编程，

当然 PowerBuilder 对指针支持也不是很好，为了能够使用一个指向自定义结构的指针变量，不得不使用内存分配函数，开辟一块内存，然后把这个结构的内容复制到这个内存地址中，再把这块内存返回的句柄参数转换成一个无符号长整型变量来使用。而在 C 语言中只需要在这个地址变量的前面加一个&符号，取得这个变量的地址指针即可。

Windows API 函数对于返回 32 位的类型返回值，都是通过 EAX 寄存器返回的。如果在执行过程中产生错误，可以调用 GetLastError 函数来返回错误代码。如果用户需要了解错误代码对应的描述信息，则需要使用 Visual C++提供的 Error Lookup 工具，输入返回的错误码即可，如图 1-5 所示。

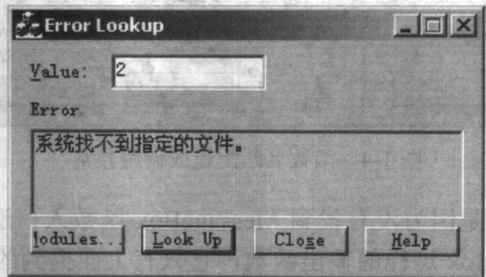


图 1-5　出错信息查找

例 1-1　把错误代码转换成错误描述信息。

```
void winerr(const char *fn)
{
    char *win_msg = NULL;
    DWORD code = GetLastError();
    if (code == 0)
      return;
    else {
      FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
      FORMAT_MESSAGE_FROM_SYSTEM,NULL,code,MAKELANGID(LANG_NEUTRAL,
      SUBLANG_DEFAULT),(LPTSTR)&win_msg,0,NULL);
      if (win_msg != NULL) {
        int len = strlen(win_msg);
        if (len >= 1)
          win_msg[len-1]=0;
        if (len >= 2)
          win_msg[len-2]=0;
          LocalFree(win_msg);
      }
    }
    SetLastError(code);
}
```

这是一段用 C 给出的代码，用户可以根据采用的编程语言进行移植，作为一个编写良好的应用程序，应该包含容错处理代码，以便程序能够体面地发现不可预知的错误。当错误发生时，程序能够请求用户干预或者能够自我修复，在极端的情况下，还可以注销或者关闭系统。

试读结束：需要全本请在线购买： www.ertongbook.com