序 言

自 2012 年至今，编程达人一直致力于计算机领域的底层原理教学。先是采用实地教学的方式培养了一批优秀的计算机领域的人才。为了适应互联网教学的发展，降低学习成本，满足更多同学的学习愿望，自 2014 年开始转而进行网络教学，积累了大量的教学经验和资料。为了更为方便的学习，我们将这些资料整理为纸质教材和与之配套的视频。

2014 年整理了第一本纸质教材《滴水内部培训教材》。

2018 年整理完成第二本纸质教材《汇编 C 语言基础教程》。

在此基础之上，我们将继续完善并改进教材的编写工作，将上述两本教材融入编程达人系列教材中，形成一个前后衔接的完整知识体系。

编程达人之系列教材：《X86 汇编语言基础教程》、《汇编的角度——C 语言》、《Windows API 每日一练》、《Windows PE》、《Windows 核心编程解读》、《Windows 32 位内核分析》、《Windows 64 位内核分析》。

本套系列教材将覆盖 16 位、32 位和 64 位计算机的汇编、C\C++语言程序设计、计算机原理和操作系统。

内容简介

Windows 视窗操作系统自诞生以来，时至今日仍然在个人电脑领域居于主导地位。Windows 视窗操作系统人性化的窗口界面，与用户友好的交互方式被平板电脑与手机操作系统所广泛借鉴。本书的所有示例代码均使用 C 语言实现，直接调用 Windows 系统原生 API 函数实现，以方便读者更清晰地理解 Windows 视窗程序实现的过程和原理。我们可以将其理解为类似于 MFC 等 C++界面库的底层实现（源码）。相信读者学习完本书之后再学习 Windows 系统或其他平台操作系统类似于 MFC、QT 等界面库和开发框架将易如反掌。

本书是编程达人系列教材的第三本书《Windows API 每日一练》。全书包含上下两册，分为 Windows 程序设计基础、进阶、高级三个部分，共二十四章。其中第一部分基础包含了第一章到第十二章的内容。详细讲述了简单 Windows 程序的设计和实现方法。第二部分进阶包含了第十三章到第十七章的内容。详细讲述了打印机、位图、调色板、文本和字体、图元文件的设计和实现方法。第三部分包含了第十八章到第二十四章。详细讲述了多文档界面、进程与线程、动态链接库、过程控制、异常处理、声音和 WinSock 设计和实现方法。

本书配套有完整的 181 个示例代码。由于自己实现所有的示例代码并不是一件轻松的事情，而且也不能保证能够完美的体现所有的技术细节。因此，这些事例代码大多摘自《Windows 程序设计》（第 5 版 珍藏版），在此做郑重声明，并对该书的作者 Charles Petzold 表示感谢和深深的敬意。本书以每日一练的形式呈现所有的示例代码，并加以详细的注释和说明。部分示例在原有代码的基础上做了改变，以方便读者以实验的方法来比较不同实现方法之间的区别。

学习本书有三个先决条件：

首先了解 Windows 系统。

其次懂得如何使用 C 语言。

最后安装 Visual Studio C 语言编程环境。

著作权人：王德华
校对：李娜
错误勘正请发送邮件 bcdaren@126.com
联系电话：0512-57882866
地址：江苏省苏州市昆山市巴城镇学院路 88 号
官网地址：www.bcdaren.com
学习论坛：http://www.dtdebug.com

目　录

目　录

第一部分 基础

第一章 Windows

在正式学习 Windows 程序设计之前，我们先对 Windows 操作系统做一个全面的了解，并搭建编写 Windows 程序的开发环境。

本章学习知识概要：

●Windows 简介
●动态链接库与 API
●内存管理模式
●Unicode 简介
●编程环境

1.1Windows 简介

本节必须掌握的知识点：

◆ **Windows 操作系统发展历程**
◆ **Windows 与处理器**

1.1.1 Windows 操作系统发展历程

■1985 年 11 月，微软公司发布 Windows 1.0 （基于 DOS 结构的 16 位操作系统），这是第一个面对普通用户的视窗操作系统。我们今天所熟悉的窗口、菜单、对话框至此诞生。Windows 视窗操作系统这些非常人性化的设计，赢得了个人用户的青睐，获得了巨大的成功，大大加速了个人计算机的普及和推广。即使今天人类早已进入智能手机时代（可以视作为个人电脑的演化），我们仍不应该吝惜对 Windows 操作系统的赞美之词。更何况，时至今天，Windows 操作系统仍然是个人计算机操作系统领域无可置疑的霸主。

■1993 年 7 月，微软发布 Windows NT 架构的 32 位操作系统，这是一个 32 位、抢占式、多任务、多线程视窗操作系统，支持 X86 CPU 保护模式，以 2000 年微软发布的 Windows2000 系统和 2001 年微软发布的 Windows XP 系统为代表。

■2006 年 11 月微软发布 Vista 系统，支持双核处理器。

■2009 年 10 月微软发布 Windows 7 操作系统，分为 32 位版本和 64 位版本，支持 X64 处理器。Win7 操作系统是除了 XP 系统之外的第二经典的 Windows 操作系统。

■2012 年 10 月微软发布了 Windows 8 操作系统，同时支持 Intel、AMD 和 ARM 芯片架构，不仅应用于个人电脑，还支持手机、平板等移动设备。

■2015 年 7 月微软发布支持 X64 处理器的 Windows 10 操作系统。

■2021 年 6 月微软发布 Windows 11 操作系统，更好的多媒体体验，安全性更高，同时支持 Android 应用程序。

Windows 操作系统非常值得称赞的一点是其良好的兼容性。32 位操作系统向下兼容早

期的 16 位应用程序。同样，Windows64 位操作系统向下兼容 32 位应用程序，但不再支持 16 位应用程序。这当然离不开处理器的支持。此外，Windows 操作系统动态链接库中的 API 函数仍然保持非常好的延续性。我们会在本书的示例代码中发现，很多几十年以前非常古老的 API 函数今天仍在使用。

1.1.2 Windwos 与处理器

操作系统是整个计算机系统中的软件部分，是基于计算机硬件平台而开发的系统软件。操作系统最早的雏形是负责加载并监控程序运行的软件。随着计算机的发展逐步添加了许多其他功能。

■在 16 位处理器时代，个人计算机普遍使用的是 DOS 操作系统。DOS 操作系统只支持单任务，依赖系统中断实现任务切换。受制于处理器 20 根地址线的限制，最多可以访问 2^{16} 个字节（1MB）大小的内存空间。DOS 操作系统将硬件设备的 BIOS 程序封装为系统中断（21H 号中断）的各个功能号，以方便应用程序的调用。汇编语言可以越过操作系统，直接控制硬件。但是高级语言只可以运行在操作系统之上，不能直接控制硬件设备。

■在 32 位处理器时代，80%以上的个人电脑安装了 Windows 操作系统，其他操作系统诸如 IOS 苹果系统、Unix、Linux 操作系统。作为视窗操作系统，其应用程序实现的方法和原理非常类似。本书仅讲述 Windows 操作系统程序设计。由于 32 位处理器具有 32 根地址线，因此，Windows 32 位操作系统应用程序支持 2^{32} 个字节（4GB）的寻址范围。此外，Windows 操作系统支持 32 位处理器的保护模式，将应用程序分为 R0~R34 个特权级（Windows 操作系统只适用了 R0 和 R3 两个特权级），本书仅涉及 R3 特权级的应用程序。至于具有 R0 特权级的驱动程序，我们将在后续《Windows 32 位内核分析》和《Windows 64 位内核分析》书中讲述。

Windows32 位应用程序都是按照 4GB 空间的模板设计的，具有 R0 特权级的应用程序仅涉及 4GB 空间中的低 2GB 空间的访问，而高 2GB 空间则借助于 Windows 操作系统的系统调用或中断实现。有意思的是，在应用程序中，我们调用 Windows 系统动态链接库提供的 API 函数，然后经过系统调用，进入操作系统内核，最终由内核函数实现。意思是我们所调用的 API 函数仅仅只是一个外壳，至于 Windows 操作系统内核是如何实现的，与我们无关。这样设计的好处是应用程序员只需要关心 API 函数或第三方封装函数的调用，至于操作系统是如何实现的，以及计算机硬件如何实现的，这些非常麻烦的事情是不需要我们关心的，统统交给操作系统。因此，作为 Windows 程序员是一件非常惬意的事情，只需要按照现有的程序模板，修修改改就可以了，大大提高了 Windows 应用程序开发的效率。当然，如果想要作为一个优秀的 Windows 应用程序的开发者，还需要知其然而知其所以然。这也是本书想要达到的目的。

■1999 年 AMD 公司首选发布了向下兼容的 X86-64 处理器，两年后，Intel 公司也推出自己的 X64 处理器。与此配合默契的 WinIntel 联盟（Windows-Intel）的微软公司也同时发布了 Windows 64 位操作系统，支持 2^{64} 个字节的寻址范围，舍弃了 32 位保护模式中的段保护模式（保护模式我们将在 1.3 节内存管理模式中详细讲述）。Windows 64 位操作系统向下兼容支持 32 位应用程序。这就意味着，我们在开发应用程序时，既可以开发 32 位应用程序，也可以开发 64 位应用程序，全凭开发者的心情。当然，如果在乎性能，当然首选 64 位应用

程序了（更改一下编译器运行平台设置而已）。

1.2 动态链接库与 API

本节必须掌握的知识点：

- **kernel32.dll**
- **user32.dll**
- **gdi32.dll**

■动态链接库

最早的软件开发过程，所有的功能实现都是有程序员独立完成的。在这个过程中，我们很快就会发现，有很多常用的功能模块是可以重复利用的，我们将其封装为功能函数。我们将这些功能函数独立编译成统一格式的 OBJ 目标文件，并存放于一个 LIB 库文件中。源程序中可以调用这些第三方功能函数，当编译器编译时，暂时将这些第三方功能函数的地址空下，待链接器链接时，在 LIB 库中查找对应的 OBJ 功能模块，并将其添加到源代码中，填写空下的函数地址。这就是我们所谓的静态链接库了。

在 Windows 操作系统中，采用的是另外一种形式的链接库，我们将特定功能的第三方

功能函数封装到一个 dll 形式的库中，我们将其称为动态链接库。动态链接库 dll 文件通常保存在“C:\Windows\System32”系统目录下。源程序中同样直接调用第三方功能函数（需要头文件声明函数原型）。编译器编译链接时，会将第三方功能函数所在的 dll 动态链接库名称、函数名称、函数序号以及函数相对地址记录到一个 PE 文件（Windows 指定的 exe 二进制格式文件）的节区。待编译后的 EXE 文件由操作系统加载到低 2GB 空间时，加载 dll 动态链接库（除了系统链接库之外，也可以是自定义的 dll 动态链接库）。此时，应用程序和 dll 动态链接库同处于一个 4GB 虚拟空间内，然后根据所引用的函数名或序号在动态链接库内查找该函数真实的地址，替换函数地址表内的相对地址，这样形成一个完整的程序就可以正常运行了。

为何要采用动态链接库这种形式呢？随着应用程序的功能越来越多，所引用的库函数也变得越来越多，如果继续采用静态链接库的形式，无疑会大大增加程序的体积，基于有限的存储空间和程序加载效率的考虑，设计动态链接库的形式可以很好地解决上述问题。动态链接库的缺点是对操作系统环境的依赖。

■API 函数

对于程序员来说，Windows 操作系统的功能完全由 API（Application Programming Interface 首字母的缩写，意思为程序之间的接口）来定义。API 涵盖了应用程序所能调用的全部操作系统函数，以及相关的数据类型和结构。在 Windows 中，API 还隐含了一种特殊的程序结构，我们将在后续章节中详细探讨这种程序结构。

早期的 Windows 1.0 版本只能支持不到 450 个函数，而今天则支持数千个 API 函数。我们把 windows 1.0 到 3.1 版本称为 win16，win95 及 98 之后的所有 NT 版本称为 win32。从 win16 到 win32 转化过程中大部分函数保持不变，但也有一些需要扩展，比如图形坐标点数值从 win16 的 16 位扩展为 32 位。此外，win16 中有些函数调用返回的二维坐标点被压缩到一个 32 位的整数里，这在 win32 中就行不通了。为解决这些问题，win32 增加了一些新的函数。

所有 32 位版本的 windows 既支持 win16 API 以保证和原先的程序兼容，也支持 win32 API 以运行新的应用程序。有意思的是，windows NT 与 win95 和 win98 的工作方式不同。在 WINDOWS NT 中，win16 的函数调用通过一个翻译层先被转换成 win32 的函数调用，然后再由操作系统来处理。而在 windows95 和 windows 98 中正好相反，win32 的函数调用通过一个翻译层先被转换成 win16 的函数调用，然后再由操作系统来处理。

接下来，我们逐一介绍三个 Windows 应用程序必备的 dll 动态链接库。

1.2.1 kernel32.dll

kernel32.dll 是一个 Windows 操作系统核心动态链接库文件。库中包含了九百多个 API 函数，这些 API 函数主要负责内存管理、进程和线程管理、调试、错误处理和时间处理等功能。

■内存管理：kernel32.dll 包含了管理内存的 API 函数。例如：

●GlobalAlloc 和 LocalAlloc 函数用于从堆中分配内存。

●GlobalFree 和 LocalFree 函数用于释放之前分配的内存。

●VirtualAlloc 和 VirtualFree 函数用于分配和释放大块的进程虚拟地址空间（4GB）中的内存。

●HeapAlloc 和 HeapFree 函数用于管理堆中的内存。

●HeapCreate 函数创建私有堆。

类似的虚拟空间和堆栈的操作函数还有很多，不再一一列举。

■**进程和线程管理**：Windows 操作系统作为多任务系统，需要创建、管理、维护和切换多个任务。Windows 操作系统是进程强相关的操作系统，进程与进程之间可以进行读写操作。Windows 操作系统中的进程可以包含一个或多个线程，进程是一些资源的集合，进程内的所有线程可以共享这些资源。在多任务系统中，任务切换其实就是线程切换。可以是同一进程中的线程切换，也可以是不同进程间的线程切换。kernel32.dll 包含的常用的进程和线程函数有：

●CreateProcess 函数创建进程。

●CreateThread 函数创建线程。

●CloseHandle 函数减少一个内核对象计数。

●WaitForSingleObject 函数检测内核对象信号状态。

●CreateEvent 函数创建事件内核对象。

●TerminateProcess 函数结束进程。

●TerminateThread 函数结束线程。

■**调试函数**：kernel32.dll 还包含了一些和调试相关的函数，包括进程间的远程读写。例如：

●DebugActiveProcess 函数使调试程序能够附加到活动进程并对其进行调试。

●GetThreadContext 函数可以检索指定线程的上下文。

●ReadProcessMemory 函数从指定进程中的内存区域读取数据。

●WriteProcessMemory 函数将数据写入到指定进程中的内存区域。

■**错误处理**：在程序开发中，难免会出现一些错误和异常，检测和处理这些错误和异常需要使用 kernel32.dll 中的函数。例如：

●SetUnhandledExceptionFilter 函数安装一个顶层的全局异常处理程序。

●GetExceptionCode 函数用于获取异常的类型。

●GetExceptionInformation 函数用于获取异常的信息。

●RaiseException 函数在调用线程中引发异常。

●GetLastError 函数检索最后错误代码。

●若要访问消息字符串，需要使用 FormatMessage 函数。

■**时间处理**：Windows 操作系统提供了各种各样的时间处理函数，这些函数无疑都是在动态链接库 kernel32.dll 中的。例如：

●系统时间函数 GetSystemTime 检索 UTC 格式的当前系统日期和时间。

●SetSystemTime 函数设置当前系统时间和日期。

●本地时间函数 GetLocalTime 检索当前本地日期和时间。

●用于处理文件时间的函数 GetFileTime 检索指定文件或目录的创建、上次访问和上次修改的日期和时间。

● FileTimeToLocalFileTime 函数将 UTC 文件时间转换为本地文件时间。● FileTimeToSystemTime 函数将文件时间转换为系统时间格式。

1.2.2 user32.dll

user32.dll 主要负责处理用户图形界面，是 Windows 用户界面相关应用程序接口。user32.dll 中常用的 API 函数如下：

■窗口操作：

●CreateWindowEx 函数用指定方式创建一个窗口。

●CreateMDIWindow 函数创建一个多文档界面窗口。

●DestroyWindow 函数销毁指定的窗口。

●FindWindow 函数从类名或窗口名中返回一个相匹配的顶层窗口的句柄。

●GetWindow 函数返回指定窗口的句柄。

●RegisterClass 函数为以后调用 CreatWindow 函数注册一个窗口类。

■菜单操作：

●CreateMenu 函数创建一个菜单，然后用 AppendMenu 函数填充菜单项。

●DestroyMenu 函数销毁指定的菜单。

●GetMenuItemInfo 函数返回有关菜单项的信息。

●AppendMenu 函数在给定菜单的尾不增加新项。

■对话框操作：

●CreateDialotParam 函数从对话框模板资源中创建一个无模式对话框。

●DialogBoxParam 函数从对话框模板资源中创建一个模式对话框。

●MessageBox 函数创建、显示并操作一个消息框。

■GUI 对象操作：

●CreateCursor 函数用指定大小、位模式、热点创建一个光标。

●CreateIcon 函数用指定大小、颜色和位模式创建一个图标。

●CopyRect 函数拷贝一个矩形坐标。

●GetScrollInfo 函数返回滚动条的参数，包括最小/最大滚动位置，页大小及拇指框的位置。

●LoadImage 函数装入一个图标、光标或位图。

●SetScrollRange 函数设置滚动条最大或最小位置值。


提示

【注】Windows 对象可以分为 GUI 对象、GDI 对象和内核对象三类。

GUI 对象包括窗口、客户窗口、滚动条、菜单、标题栏等用户图形界面中的对象。

GDI 对象包括画笔、画刷、字体、位图、路径、调色板等绘图工具。

内核对象包括进程、线程、事件、信号量、互斥量等内核中创建的对象。

■消息操作：

●GetMessage 函数从指定线程的消息队列中检取一条消息。

●PeekMessage 函数检查应用程序的消息队列。

●PostMessage 函数在指定的窗口消息队列中放置一条消息。

●SendMessage 函数把一消息发送给指定的多个窗口。

●TranslateMessage 函数把虚键消息翻译为字符消息。

●DispatchMessage 函数传送一个消息给指定的窗口过程。

1.2.3 gdi32.dll

gdi32.dll 动态链接库用于图形设备接口（GDI）功能的实现。GDI 是一个图形接口，负责处理 Windows 系统中的图形和字体操作，如绘制图形、显示文本、打印等操作。GDI 函数有一个非常明显的特征是函数的第一个参数一定是设备环境上下文句柄（跟踪设备上下文地址的指针）。gdi32.dl 中常用的 API 函数如下：

■GDI 对象操作：

●CreateBitmap 函数创建具有指定宽度、高度和颜色格式的位图， (颜色平面和每像素位) 。

●CreateBrushIndirect 函数创建具有指定样式、颜色和图案的逻辑画笔。

●CreatePen 函数创建具有指定样式、宽度和颜色的逻辑笔。 笔随后可以选择到设备上下文中，并用于绘制线条和曲线。

●CreateSolidBrush 函数创建具有指定纯色的逻辑画笔。

●CreateCompatibleDC 函数创建与指定设备兼容的内存设备上下文 (DC) 。

●CreateDIBitmap 函数从 DIB 创建兼容位图 (DDB) ，并选择性地设置位图位。

●CreateFont 函数创建具有指定特征的逻辑字体。 随后可以选择逻辑字体作为任何设备的字体。

●CreatePalette 函数创建逻辑调色板。

●GetDC 函数检索设备上下文的句柄， (指定窗口的工作区或整个屏幕的 DC) 。

●getObject 函数 (wingdi.h) 检索指定图形对象的信息。

■绘图函数：

●LineTo 函数从当前位置绘制一条线，但不包括指定点。

●MoveToEx 函数将当前位置更新为指定点，并根据需要返回上一个位置。

●Rectangle 函数绘制矩形。 矩形使用当前笔轮廓，并使用当前画笔填充。

●RoundRect 函数绘制带圆角的矩形。 矩形使用当前笔轮廓，并使用当前画笔填充。

●Ellipse 函数绘制椭圆形。 椭圆的中心是指定边框的中心。 椭圆形使用当前笔轮廓，并使用当前画笔填充。

●CreateEllipticRgn 函数创建椭圆区域。

●CreatePolygonRgn 函数创建多边形区域。

●CreateRectRgn 函数创建矩形区域。

●DrawText 函数在指定矩形中绘制格式化文本。

●TextOut 函数使用当前所选字体、背景色和文本颜色在指定位置写入字符串。

●FillPath 函数关闭当前路径中所有打开的图形，并使用当前画笔和多边形填充模式填充路径的内部。

●FillRect 函数使用指定的画笔填充矩形。 此函数包括左边框和上边框，但不包括矩形的右边框和下边框。

●FrameRect 函数使用指定的画笔在指定矩形周围绘制边框。 边框的宽度和高度始终

是一个逻辑单元。

●FrameRgn 函数使用指定的画笔在指定区域周围绘制边框。

●GetDIBits 函数检索指定兼容位图的位，并使用指定格式将其作为 DIB 复制到缓冲区中。

●PlayMetaFile 函数显示存储在指定设备上的给定 Windows 格式图元文件中的图片。PolyDraw 函数绘制一组线段和 B zier 曲线。

●PolyBezierTo 函数绘制一条或多条 B zier 曲线。

●PolylineTo 函数绘制一条或多条直线。

●SetMapMode 函数设置指定设备上下文的映射模式。 映射模式定义用于将页面空间单位转换为设备空间单位的度量单位，还定义设备的 x 和 y 轴的方向。

●BitBlt 函数将对应于像素矩形的颜色数据从指定的源设备上下文传输到目标设备上下文中。

●BeginPaint 函数准备用于绘制的指定窗口，并使用有关绘制的信息填充 PAINTSTRUCT 结构。

■打印输出函数：

●StartDoc 函数启动打印作业。

●EndDoc 函数结束打印作业。

●EndPage 函数通知设备应用程序已完成对页面的写入。 此函数通常用于指示设备驱动程序转到新页面。

●StartPage 函数准备打印机驱动程序以接受数据。

●GetPrinter 函数检索有关指定打印机的信息。

●EnumPrinters 函数枚举可用的打印机、打印服务器、域或打印提供程序。