

Windows 编程

学习 C 语言很久了，是不是一直在跟黑乎乎的画面打交道，像 QQ、360、VC6.0 这样的软件都是带界面的，怎么做到的呢？

前面我们讲的”黑屏“叫控制台应用程序(Win32 Console Application)，也称 DOS 程序（或 MS-DOS 程序）。DOS 是早期的命令式操作系统，很难做出漂亮的界面，除了开发人员，”黑屏“对普通用户很不友好。

带界面的程序叫 Windows 应用程序(Win32 Application)。Windows 是一款现代操作系统，有丰富的交互界面，使用简单，无需记忆繁杂的命令。

使用 C 语言可以开发出 Windows 应用程序，也就是带界面的程序，只是绝大部分 C 语言教程没有讲，它们只讲了基本语法，让很多初学者认为学 C 语言没用，什么都做不出来。

其实不是这样的，C 语言只是一种工具，需要与 Windows 系统结合，借助 Windows 提供的函数才能开发出漂亮的程序。

这套教程，将以简单明了的方式教你使用 C 语言来编写带界面的 Windows 应用程序。

Windows API 编程、Windows 编程、Windows SDK 编程是一个概念，阅读完教程第一节你就会明白。

Windows API 的概念

在 C 语言中，使用 `fopen()` 函数可以打开一个文件，感觉非常简单。文件保存在硬盘上，要经过复杂的处理才能显示，这些细节对我们来说是透明的，由操作系统完成。也就是说，我们调用 `fopen()` 函数来通知操作系统，让操作系统打开一个文件。

那么，我们如何告诉操作系统打开文件呢？

看似简单的操作到底层都非常复杂，打开文件首先要扫描硬盘，找到文件的位置，然后从文件中读取一部分数据，将数据放进 I/O 缓冲区，放进内存；这些数据都是 0、1 序列，还要对照 ASCII 表或 Unicode 表”翻译“成字符，再在显示器上显示出来。这个过程如果要是程序员来完成，那简直是噩梦！

怎么办呢？Windows 想了一个很好的办法，它预先把这些复杂的操作写在一个函数里面，编译成动态链接库(DLL)，随 Windows 一起发布，程序员只需要简单地调用这些函数就可以完成复杂的工作，让编程变得简单有趣。

这些封装好的函数，叫做 API(Application Programming Interface)，即应用程序编程接口。

API 函数以 C 语言的形式向外暴露，可以通过 C 语言直接调用。

除了函数，Windows 还预先定义了很多数据类型（使用 C 语言的 `typedef` 关键字定义）。广义上来说，这些数据类型也是 API 的一部分。

API 屏蔽了很多细节，大大简化了程序员的工作，这就是操作系统的威力，不但让普通用户使用方便，也让程序员如释重负。

在 Windows 上运行的程序（包括 MS-DOS 程序），本质上都是通过调用 Windows API 来完成功能的，包括 QQ、360、VC6.0 等，别看这些团队牛，也不可能从底层做起，那简直不可想象。

C 语言也一样，也是调用 Windows API，fopen() 函数就是通过调用 CreateFile() 函数实现的。CreateFile() 是 Windows API 中的一个函数，可以用来打开或创建文件。

通常所说的 SDK 编程就是直接调用 API 函数进行编程。SDK 是 Software Development Kit 的缩写，即软件开发工具包。

Windows API 函数成千上万，详细了解每一个函数的用法是不可能的，也是完全没有必要的。只需知道哪些功能由哪些 API 函数提供就行了，等使用它们时再去查阅帮助文件。

带界面的程序的专业称呼是 GUI 程序。GUI 是 Graphical User Interface 的简写，即图形用户界面。本教程将教你使用 Windows API 来编写 GUI 程序，编程语言为C 语言。

第一个真正的 Windows 程序

编写 Windows 程序，首先要包含 windows.h 头文件。windows.h 还包含了其他一些 Windows 头文件，例如：

- `windef.h`：基本类型定义
- `winbase.h`：内核函数
- `wingdi.h`：用户接口函数
- `winuser.h`：图形设备接口函数

这些头文件定义了 Windows 的所有数据类型、函数原型、数据结构和符号常量，也就是说，所有的 Windows API 都在这些头文件中声明。

在 C 语言中，程序都是“黑屏”的，称为控制台程序(Console Application)。这套教程要讲的是带界面的 Windows 程序(Windows Application)，也称为 GUI 程序(GUI Application)。

控制台程序以 main() 为入口函数，Windows 程序以 WinMain() 为入口函数，动态链接库(DLL)以DllMain() 为入口函数（请查看 [动态链接库 DLL 教程](#)），不同的入口函数决定了不同类型的程序。

WinMain() 函数的原型为：

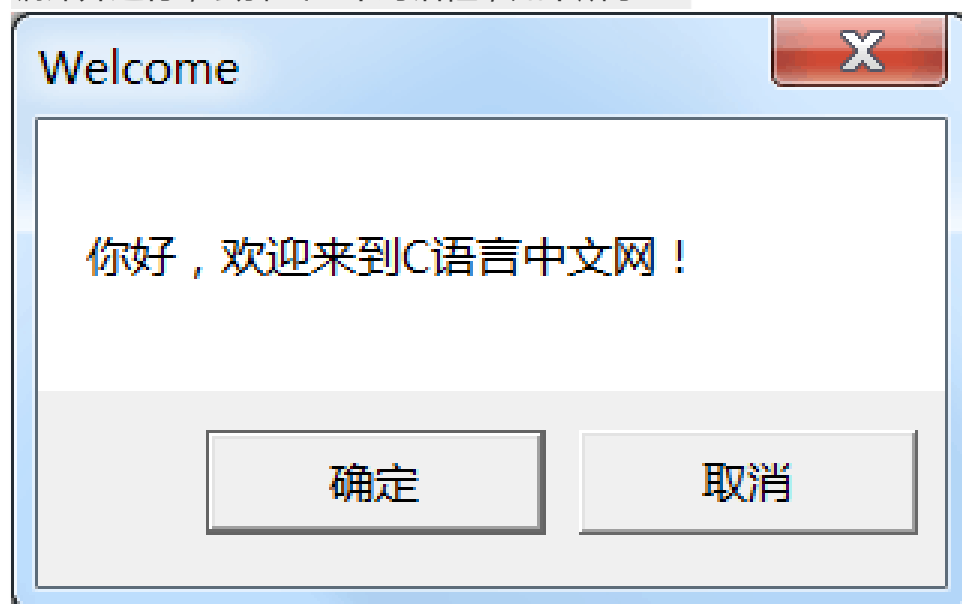
1. `int WINAPI WinMain(`
2. `HINSTANCE hInstance, // 当前窗口句柄`
3. `HINSTANCE hPrevInstance, // 前一个窗口句柄，Win32 下为 NULL（Win16 留下的废物，目前已弃用）`

```
4.     LPSTR lpCmdLine, // 命令行参数
5.     int nCmdShow // 窗口显示方式
6. );
```

先不要急于理解这些参数的含义，我先给大家写一个简单的不带黑屏的、真正的 Windows 程序：

```
1.     #include <windows.h>
2.
3.     int WINAPI WinMain(
4.         HINSTANCE hInstance,
5.         HINSTANCE hPrevInstance,
6.         LPSTR lpCmdLine,
7.         int nCmdShow
8.     ){
9.         // 调用 API 函数 MessageBox
10.        int nSelect = MessageBox(NULL, TEXT("你好，欢迎来到 C 语言中文网！"),
11.        TEXT("Welcome"), MB_OKCANCEL);
12.        if(nSelect == IDOK){
13.            MessageBox(NULL, TEXT("你点击了“确定”按钮"), TEXT("提示"), MB_OK);
14.        }else{
15.            MessageBox(NULL, TEXT("你点击了“取消”按钮"), TEXT("提示"), MB_OK);
16.        }
17.        return 0;
18.    }
```

编译并运行，会弹出一个对话框，如下所示：



点击“确定”或“取消”按钮，又会弹出一个新的提示框。大家可以亲自运行一下，会有真实的体验。

原来没有 main() 函数，没有 #include <stdio.h> 的 C 语言程序也是可以运行的！

MessageBox() 函数是众多 API 中的一个，用于弹出一个指定风格的对话框，其原型为：

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

WINAPI 为宏定义#define WINAPI _stdcall，表示函数调用方式，暂时可以不理睬，不影响代码编写，只需要知道 MessageBox()返回值为 int，表示按下的按钮。有兴趣的读者可以查看：[__stdcall](#)，[_cdecl](#)，[_pascal](#)，[__fastcall](#) 的区别

参数说明：

- hWnd：该消息框的父窗口句柄，如果此参数为NULL，则该消息框没有拥有父窗口。大家不用急于理解这个参数，后续会详细讲解。
- lpText：消息框的内容。LPCTSTR 是自定义数据类型，等价于 const char *。
- lpCaption：消息框的标题。
- uType：对话框的按钮样式和图标。

uType 支持的按钮样式：

按钮	含义
MB_OK	默认值，有一个“确认”按钮在里面
MB_YESNO	有“是”和“否”两个按钮在里面
MB_ABORTRETRYIGNORE	有“中止”，“重试”和“跳过”三个按钮在里面
MB_YESNOCANCEL	有“是”，“否”和“取消”三个按钮在里面
MB_RETRYCANCEL	有“重试”和“取消”两个按钮在里面
MB_OKCANCEL	有“确定”和“取消”两个按钮在里面

这些按钮都是宏定义：

1. #define MB_OK 0x00000000L
2. #define MB_OKCANCEL 0x00000001L
3. #define MB_ABORTRETRYIGNORE 0x00000002L
4. #define MB_YESNOCANCEL 0x00000003L
5. #define MB_YESNO 0x00000004L
6. #define MB_RETRYCANCEL 0x00000005L

你也可以尝试用数字来表示按钮，例如：

```
MessageBox(NULL, TEXT(“你好，欢迎来到C 语言中文网!”), TEXT(“Welcome”), 1);
```

也会生成与上面相同的对话框。

除了按钮，uType 还支持图标（图标对用户有提醒作用）：

图标	含义
----	----

MB_ICONEXCLAMATION	一个惊叹号出现在消息框： 
MB_ICONWARNING	一个惊叹号出现在消息框（同上）
MB_ICONINFORMATION	一个圆圈中小写字母i 组成的图标出现在消息框： 
MB_ICONASTERISK	一个圆圈中小写字母 i 组成的图标出现在消息框（同上）
MB_ICONQUESTION	一个问题标记图标出现在消息框： 
MB_ICONSTOP	一个停止消息图标出现在消息框： 
MB_ICONERROR	一个停止消息图标出现在消息框（同上）
MB_ICONHAND	一个停止消息图标出现在消息框（同上）

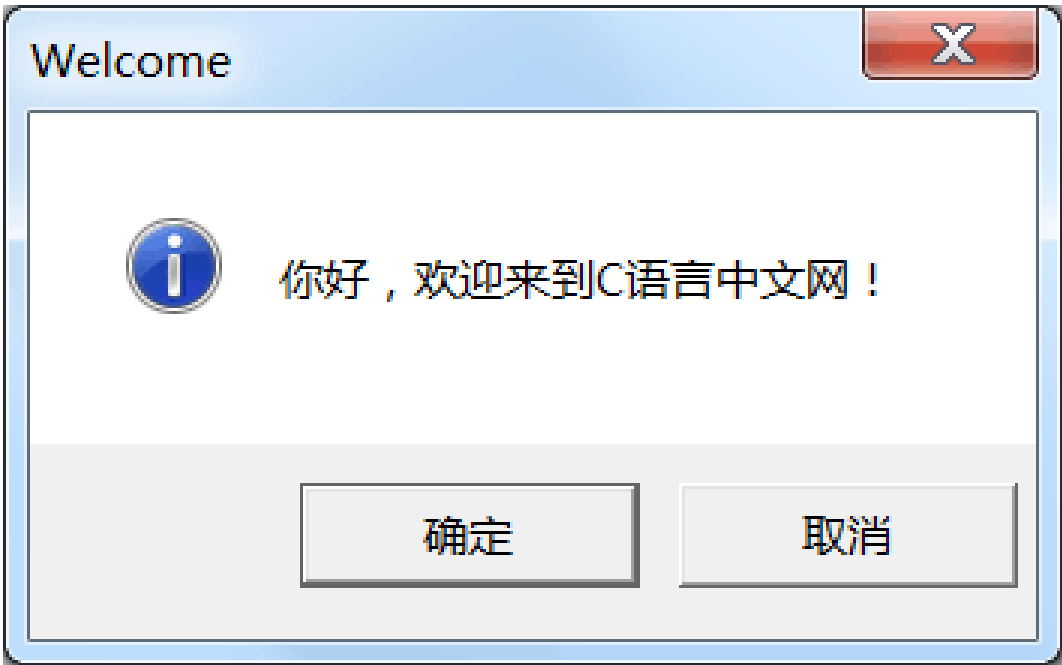
这些图标也都是宏定义：

- 1. #define MB_ICONHAND 0x00000010L
- 2. #define MB_ICONQUESTION 0x00000020L
- 3. #define MB_ICONEXCLAMATION 0x00000030L
- 4. #define MB_ICONASTERISK 0x00000040L

如果希望同时定义按钮和图标的样式，可以使用或运算|，例如：

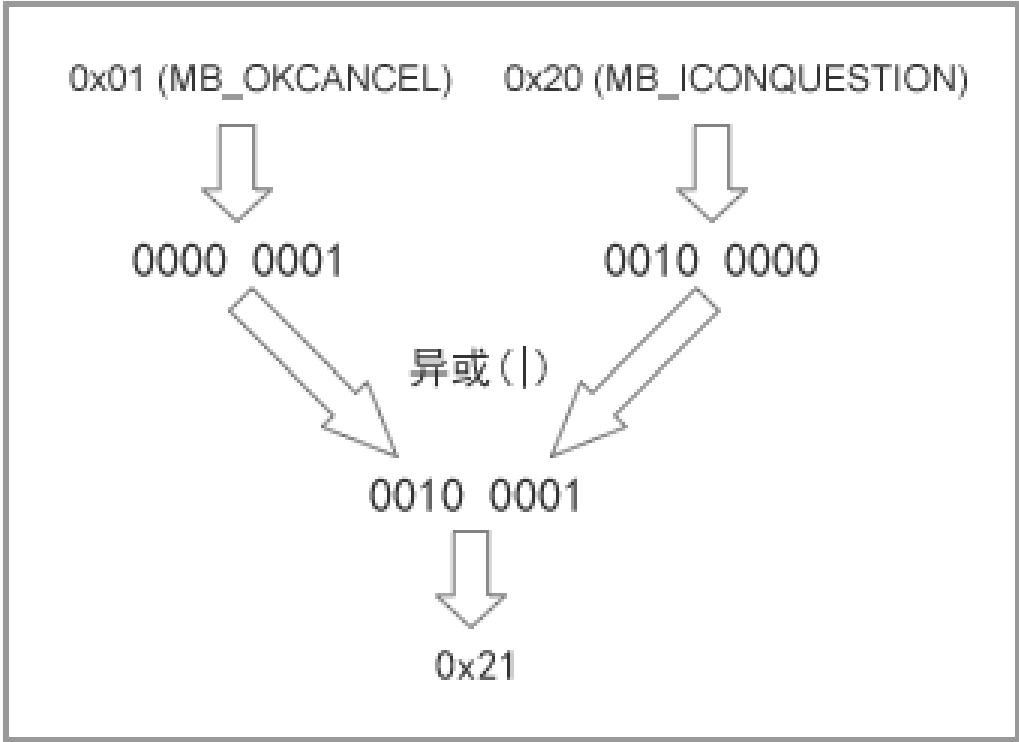
- 1. MessageBox(
- 2. NULL,
- 3. TEXT("你好，欢迎来到 C 语言中文网! "),
- 4. TEXT("Welcome"),
- 5. MB_OKCANCEL | MB_ICONINFORMATION
- 6.);

会弹出如下的对话框：



与上面的对话框相比，多出了一个图标，同时还能听到提示音（Win7 有，XP 和 Win8 读者可以亲自测试）。

大家有没有发现，按钮都是用十六进制的第1 位（二进制前4 位）来表示，图标都是使用十六进制第2 位（二进制第 5~8 位）来表示，进行或运算，每个位都不会改变，如下图所示：



Windows 通过检测第 1 位的值来确定按钮的样式，检测第 2 位的值来确定图标样式。

再来看一下 MessageBox() 的返回值。

MessageBox() 返回被按下的按钮，以数字表示，这些数字都被定义成了宏，如下所示：

返回值	含义
IDOK	用户按下了“确认”按钮
IDCANCEL	用户按下了“取消”按钮
IDABORT	用户按下了“中止”按钮
IDRETRY	用户按下了“重试”按钮
IDIGNORE	用户按下了“忽略”按钮

IDYES	用户按下了“是”按钮
IDNO	用户按下了“否”按钮

对应的宏定义为：

- 1. #define IDOK 1
- 2. #define IDCANCEL 2
- 3. #define IDABORT 3
- 4. #define IDRETRY 4
- 5. #define IDIGNORE 5
- 6. #define IDYES 6
- 7. #define IDNO 7

Windows 数据类型

Windows 使用 typedef 或#define 定了很多新的数据类型，前面几节中我们已经领略到了。它们虽然多，但是都有规律可循，很多都是对 C/C++ 中数据类型的简单加工，而且很容易“见名知意”。要想学习 Windows 编程，必须要了解常用的数据类型。

如果你对 C 语言或者 C++ 的数据类型比较熟悉的话，那么对于一些基础的内容这个过程就是慢慢熟悉的过程。

这些数据类型在 windows.h 头文件中定义：

- 1. typedef int INT; /* 整形 */
- 2. typedef unsigned int UINT; /* 无符号整形 */
- 3. typedef unsigned int *PUINT; /* 无符号整形指针 */
- 4. typedef int BOOL; /* 布尔类型 */
- 5. typedef unsigned char BYTE; /* 字节 */
- 6. typedef unsigned short WORD; /* WORD (无符号短整型) */
- 7. typedef unsigned long DWORD; /* DOUBLE WORD (无符号长整形) */
- 8. typedef float FLOAT; /* 浮点型 */
- 9. typedef FLOAT *PFLOAT; /* 指向 float 类型指针 */
- 10. typedef BOOL near *PBOOL; /* 指向布尔类型指针 */
- 11. typedef BOOL far *LPBOOL;
- 12. typedef BYTE near *PBYTE; /* 指向字节类型指针 */
- 13. typedef BYTE far *LPBYTE;
- 14. typedef int near *PINT; /* 整形指针 */
- 15. typedef int far *LPINT;
- 16. typedef WORD near *PWORD; /* 指向 WORD 类型的指针 */
- 17. typedef WORD far *LPWORD;
- 18. typedef long far *LPLONG; /* 指向长整形的指针 */

19. `typedef DWORD near *PDWORD; /* 指向 DWORD 类型的指针 */`
20. `typedef DWORD far *LPDWORD;`
21. `typedef void far *LPVOID; /* 指向 void 类型的指针 */`
22. `typedef CONST void far *LPCVOID; /* 指向 void 类型的常指针 */`

简单说下，大部分类型都没有什么离奇的地方，可能很多读者会对那个 far 和 near 觉得有疑问，实际上 F12 查看定义会发现他们就定义上方：

1. `#define far`
2. `#define near`

看到这里你是否还是有疑问？其实，这个 far 和 near 只是用来标识变量的新旧的（预处理阶段 far 和 near 会被替换成空字符串）。例如 PINT 和 LPINT 实际上都是 int *，只不过一个是老式写法，一个是新式写法，这都是为了兼容问题。

简单的看下这些数据类型，就可以总结出：但凡是以“P”开头的都是指针（pointer）类型（“LP”是老式写法）。撇开这些不谈，那么实际上这些 Windows API 常用的一些数据结构跟我们原本所学的 C/C++ 变量差别就是一个 typedef 而已，基础好的熟悉一下就行了。不过，也不要想得这么简单，Windows 编程还有一些比较复杂的类型比如 HWND、HANDLE 等。

在碰到不熟悉的类型时请熟练的使用“右键转到定义”或者 F12。大家不用太过害怕这些复杂的类型名称，Windows 数据类型并不是内建的数据类型类型，而都是从 C 类型重定义得到的。

Windows 数据类型命名的规律

- 无符号类型：一般是以“U”开头，比如“INT”对应的“UINT”。
- 指针类型：其指向的数据类型前加“LP”或“P”，比如指向 DWORD 的指针类型为“LPDWORD”和“PDWORD”。
- 句柄类型：以“H”开头。比如，HWND 是 window（WND 简写）也就是窗口的句柄，菜单(MENU)类型对应的句柄类型为“HMENU”等等。

宽字符与 Unicode

在 C 语言中，我们使用 char 来定义字符，占用一个字节，最多只能表示 128 个字符，也就是 ASCII 码中的字符。计算机起源于美国，char 可以表示所有的英文字符，在以英语为母语的国家完全没有问题。

但是世界上存在很多不同的语言，例如汉语、日语等有成千上万个字符，需要用多个字节来表示，称之为宽字符(Wide Character)。Unicode 是宽字符编码的一种，已经被现代计算机指定为默认的编码方式，Windows 2000 以后的操作系统，包括 Windows 2000、XP、Vista、Win7、Win8、Win10、Windows Phone、Windows Server 等（它们统称为 Windows NT）都从底层支持 Unicode，存取效率比 char 要高。

更多内容请查看：[ASCII 编码与 Unicode 编码](#)

C 语言中的宽字符

在 C 语言中，使用 wchar.h 头文件中的 wchar_t 来定义宽字符，例如：

```
wchar_t ch = 'A';
```

wchar_t 被定义为 typedef unsigned short wchar_t，和一个无符号整型一样，[占用两个字节](#)。

如果定义宽字符串，需要加前缀 L，例如：

```
wchar_t *str = L"C 语言中文网";
```

L 是必须要加的，并且与字符串之间不能有空格，只有这样编译器才知道每个字符占用两个字节。

宽字符示例：

```
1.      #include <stdio.h>
2.      #include <wchar.h>
3.      int main(){
4.          char ch = 'A';
5.          wchar_t wch = 'A';
6.          char str[] = "C 语言中文网";
7.          wchar_t wstr[] = L"C 语言中文网";
8.          printf("ch=%d, wch=%d, str=%d, wstr=%d\n", sizeof(ch), sizeof(wch), sizeof(str),
sizeof(wstr));
9.          return 0;
10.     }
```

运行结果：

```
ch=1, wch=2, str=12, wstr=14
```

wstr 之所以比 str 多两个字节是因为：字符 'C' 占用两个字节，字符串结束标志 '\0' 也占用两个字节。

宽字符串的长度

计算 ASCII 字符串长度使用 strlen 函数，计算宽字符串长度使用 wcslen 函数：

```
1.      #include <stdio.h>
2.      #include <wchar.h>
3.      #include <string.h>
4.      int main(){
5.          char str[] = "C 语言中文网";
6.          wchar_t wstr[] = L"C 语言中文网";
7.          printf("strlen(str)=%d, wcslen(wstr)=%d\n", strlen(str), wcslen(wstr));
8.          return 0;
9.     }
```

运行结果：

```
strlen(str)=11, wcslen(wstr)=6
```

strlen 的运行结果显然不正确，因为它把一个字节作为一个字符计算，而 wcslen 把两个字节作为一个字符计算。

注意：wcslen 在 string.h 和 wchar.h 头文件中均有说明。

维护一个版本的源代码

在 Windows NT 以前的操作系统中，甚至包括 Windows 98，对宽字符的支持都不是很好，所以大多情况下使用 ASCII 编码。Windows NT 推出以后，已经从底层支持了 Unicode，所以在 Windows NT 上的程序大多使用 Unicode。

如果你希望程序能够在各种版本的 Windows 操作系统中运行，那么就需要维护两个版本的源代码，ASCII 版和 Unicode 版。ASCII 字符和 Unicode 字符的定义、使用都不一样，要想在一个版本的源代码中做兼容处理会非常困难，要做大量的工作，对程序员来说简直是噩梦。

不过，Windows 又为我们做了一件好事，已经处理了兼容性问题。它是怎么做到的呢？

例如对于字符串，ASCII 中使用 char 来定义，而 Unicode 中使用 wchar_t 来定义，并且需要添加前缀 L。那么在 windows.h 头文件中（或者是它包含的其他头文件）就这样来处理：

1. `#ifdef UNICODE`
2. `typedef wchar_t TCHAR;`
3. `#define TEXT(quote) L##quote`
4. `#else`
5. `typedef char TCHAR`
6. `#define TEXT(quote) quote`
7. `#endif`

我们在源码中可以这样来使用：

```
TCHAR str[] = TEXT("C 语言中文网");
```

如果是 Unicode 版，也就是定义了 UNICODE 宏，那么上面的语句等价于：

```
wchar_t str[] = L"C 语言中文网";
```

如果是 ASCII，也就是没有定义 UNICODE 宏，那么等价于：

```
char str[] = "C 语言中文网";
```

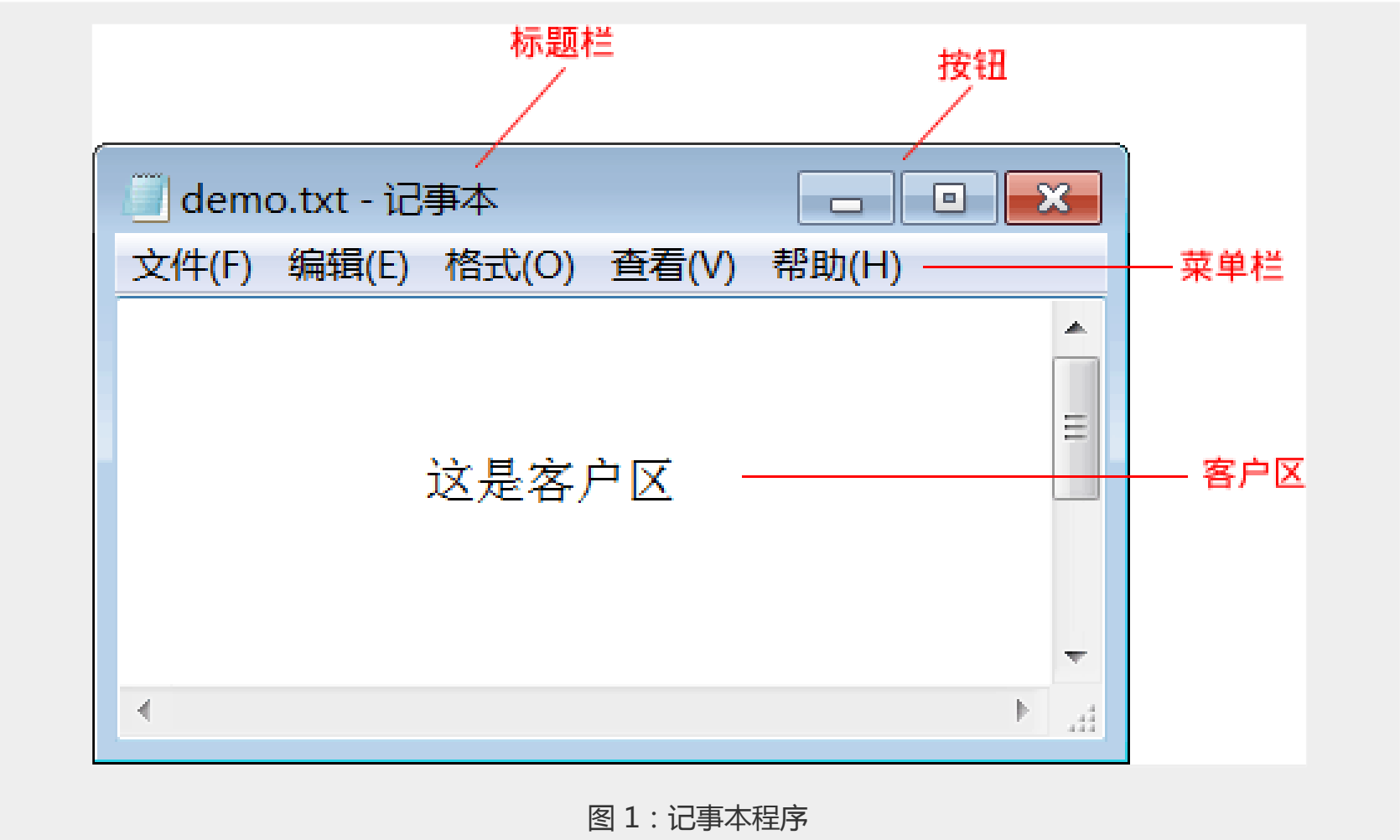
在 Windows 中，随处可见这样的处理。虽然现代操作系统都已经支持 Unicode，无需再考虑与 ASCII 的兼容性问题，但是依然要为这些历史问题付出代价。

总结：由于各种各样的原因，我们优先使用 Windows 定义的数据类型、宏、结构体等，这样编写的程序兼容性较好，不用考虑 ASCII 和 Unicode 的问题。但这也带来了一个挑战，就是要熟悉 Window 定义的数据类型、宏、结构体等。

与 windows 编程有关的重要概念

窗口

窗口的概念很容易理解，就是我们使用软件时看到的界面。Windows 的核心就是窗口，它是 Windows 一统 PC 操作系统市场的杀手锏，如下图所示：



我们使用的软件都有自己的窗口，比如 QQ、计算器、记事本等。这些窗口可以包含输入框、下拉菜单、单选按钮、多选按钮、文本区域等各种各样的**控件(Controls)**，有的甚至还有动画！

窗口、控件、图像、音频视频等都称为**资源(Resource)**，在程序中都可以使用、创建、添加、修改等。

句柄

在 Windows 编程中，不同窗口、控件、图像等都对应一个唯一的数字（初学者可以理解为 ID）称为句柄(Handle)。通过句柄，程序可以获取对应资源的各种信息，也可以使用、修改、删除该资源。

你可以将句柄理解为学号，你不需要记住学生的姓名、住址、成绩等各种信息，当你需要了解这名学生时，只要去教务处，将学号（句柄）告诉那里的工作人员(Windows)，他就能够帮你找到这个学生。

句柄屏蔽了很多细节，程序员不需要了解背后的机制。例如用 `CreateFile()` 函数创建文件后会返回一个文件句柄，然后通过这个句柄就可以读写、删除该文件，而不需要了解 Windows 是如何将句柄与文件关联起来的，也不需要了解句柄到底保存了哪些信息，Windows 是闭源的，这些背后的细节只有微软知道。

Windows 的消息机制

在一般的编程中，我们都是通过 API 函数来调用系统功能，让操作系统来帮我们完成很多工作，例如调用 `CreateFile()` 函数，操作系统会帮我们创建一个文件，而不需要我们参与任何工作，非常方便。

反过来，操作系统也会“偷懒”，会调用我们程序中的函数，让我们自己处理某些事情。例如用户敲击键盘，操作系统会首先收到通知，但它并不会处理，而是调用程序中的函数，告诉程序用户敲击了键盘，你自己处理好了；如果程序不处理，操作系统才会进行默认的操作。

当然，这不能理解为操作系统“偷懒”，而是给我们一个机会，让我们自行处理某些事情，从而使程序更加灵活和健壮，也让程序员有了更多发挥的空间。

用户敲击键盘、点击鼠标、拖动窗口、选择菜单、输入文字等所有的操作都称为**事件(Event)**。这与我们平时理解的“事件”是类似的，都表示发生了某些情况，好的或者坏的。

当有事件发生时，Windows 会生成一条**消息(Message)**，告诉程序发生了什么事情。这与我们平时理解的“消息”是类似，都表示一种传递信息的载体。

那么，Windows 是如何通过消息将发生的事件通知给应用程序的呢？

每当事件发生时，Windows 会生成一条消息，并放到一个由系统维护的队列中。然后，程序会自己从这个队列中获取消息并分析，调用事件处理函数（处理事件的代码也就在这个函数中），对用户的操作进行响应。

队列是一种先进先出的数据结构，不明白的请自行 Google 或百度。

注意：Windows 向队列中分派消息和应用程序从队列中获取消息并不是同步的，Windows 不管队列中有没有消息，不管应用程序有没有处理完毕，只要有事件发生，就会将消息丢进队列，什么时候处理完毕是应用程序的事。

可见，消息是连接 Windows 和应用程序的纽带，Windows 通过消息告诉应用程序发生了什么，应用程序通过消息知道该做什么。

消息结构体

消息其实是一个结构体，名字为 MSG，定义为：

```
1.      typedef struct tagMSG {
2.          HWND hwnd;
3.          UINT message;
4.          WPARAM wParam;
5.          LPARAM lParam;
6.          DWORD time;
7.          POINT pt;
8.      } MSG;
```

Windows 向队列中投递消息，其实就是将一个类型为 MSG 的结构体变量丢进队列。

MSG 结构体中各成员变量的含义如下：

1) hwnd 表示消息所属的窗口。用户一般是在程序的窗口下进行操作，所以一个消息一般都是与某个窗口相关联的。例如在某个活动窗口中按下鼠标左键，产生的按键消息就是发给该窗口的。

2) message 表示消息类型，是一个数值。在 Windows 中，消息是由一个数值来表示的，不同类型的消息对应不同的数值。但是由于数值不便于记忆，所以 Windows 将消息对应的数值定义为 WM_XXX 宏（WM 是 Window Message 的缩写）的形式，XXX 对应某种消息的英文拼写的大写形式。例如，鼠标左键按下消息是 WM_LBUTTONDOWN，键盘按下消息是 WM_KEYDOWN，字符消息是 WM_CHAR，等等。在程序中我们通常都是以 WM_XXX 宏的形式来使用消息的。

3) 第三、第四个成员变量 wParam 和 lParam，用于指定消息的附加信息。例如，当我们收到一个字符消息的时候，message 成员变量的值就是 WM_CHAR，但用户到底输入的是什么字符，那么就由 wParam 和 lParam 来说明。wParam、lParam 表示的信息随消息的不同而不同。

4) 最后两个变量分别表示消息投递到消息队列中的时间和鼠标的当前位置。

一个完整的 Windows 程序框架

前面我们演示了带界面的 Windows 程序，但那仅仅是一个弹窗，调用 MessageBox 函数就可以实现，不是一个真正意义上的窗口。我们通常所说的窗口包含最大化、最小化、关闭按钮，也包含菜单、单选框、图像等各种控件。

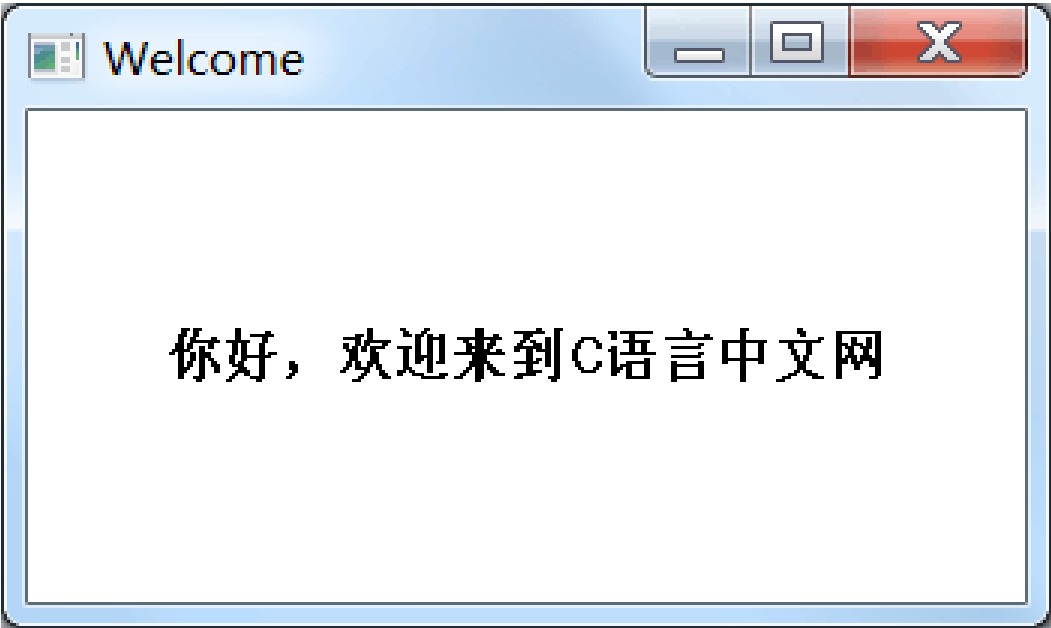
一个完整的 Windows 程序框架：

```
1.      #include <windows.h>
2.
3.      LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
4.
5.      int WINAPI WinMain(
6.          HINSTANCE hInstance,
7.          HINSTANCE hPrevInstance,
8.          PSTR szCmdLine,
9.          int iCmdShow
10.     ){
11.         static TCHAR szClassName[] = TEXT("HelloWin"); //窗口类名
12.         HWND      hwnd; //窗口句柄
13.         MSG        msg; //消息
14.         WNDCLASS wndclass; //窗口类
15.
16.         /*****第①步：注册窗口类*****/
17.         //为窗口类的各个字段赋值
18.         wndclass.style = CS_HREDRAW | CS_VREDRAW; //窗口风格
19.         wndclass.lpfnWndProc = WndProc; //窗口过程
20.         wndclass.cbClsExtra = 0; //暂时不需要理解
21.         wndclass.cbWndExtra = 0; //暂时不需要理解
```

```
22.     wndclass.hInstance      = hInstance;   //当前窗口句柄
23.     wndclass.hIcon          = LoadIcon (NULL, IDI_APPLICATION);   //窗口图标
24.     wndclass.hCursor        = LoadCursor (NULL, IDC_ARROW);   //鼠标样式
25.     wndclass.hbrBackground= (HBRUSH) GetStockObject (WHITE_BRUSH);   //窗口背
    景画刷
26.     wndclass.lpszMenuName = NULL ;   //窗口菜单
27.     wndclass.lpszClassName= szClassName;   //窗口类名
28.
29.     //注册窗口
30.     RegisterClass(&wndclass);
31.
32.     /*****第②步：创建窗口(并让窗口显示出来)*****/
33.     hwnd = CreateWindow(
34.         szClassName,   //窗口类的名字
35.         TEXT("Welcome"),   //窗口标题（出现在标题栏）
36.         WS_OVERLAPPEDWINDOW,   //窗口风格
37.         CW_USEDEFAULT,   //初始化时 x 轴的位置
38.         CW_USEDEFAULT,   //初始化时 y 轴的位置
39.         500,   //窗口宽度
40.         300,   //窗口高度
41.         NULL,   //父窗口句柄
42.         NULL,   //窗口菜单句柄
43.         hInstance,   //当前窗口的句柄
44.         NULL   //不使用该值
45.     );
46.
47.     //显示窗口
48.     ShowWindow (hwnd, iCmdShow);
49.     //更新（绘制）窗口
50.     UpdateWindow (hwnd);
51.
52.     /*****第③步：消息循环*****/
53.     while( GetMessage(&msg, NULL, 0, 0) ){
54.         TranslateMessage(&msg);   //翻译消息
55.         DispatchMessage (&msg);   //分派消息
56.     }
57.
```

```
58.         return msg.wParam;
59.     }
60.
61.     /*****第④步：窗口过程*****/
62.     LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam){
63.         HDC          hdc; //设备环境句柄
64.         PAINTSTRUCT ps;
65.         RECT         rect;
66.
67.         switch (message){
68.             //窗口绘制消息
69.             case WM_PAINT:
70.                 hdc = BeginPaint (hwnd, &ps) ;
71.                 GetClientRect (hwnd, &rect) ;
72.                 DrawText(
73.                     hdc,
74.                     TEXT("你好，欢迎来到 C 语言中文网"),
75.                     -1,
76.                     &rect,
77.                     DT_SINGLELINE | DT_CENTER | DT_VCENTER
78.                 );
79.                 EndPaint (hwnd, &ps) ;
80.                 return 0 ;
81.
82.             //窗口销毁消息
83.             case WM_DESTROY:
84.                 PostQuitMessage(0) ;
85.                 return 0 ;
86.             }
87.         return DefWindowProc(hwnd, message, wParam, lParam) ;
88.     }
```

运行结果：



对于初学者，这段代码“又臭又长”，难于理解，有点吓人。但这是一个Windows 程序的基本框架，只不过不像 C 语言的框架那么简单，几行代码搞定。大家不要急于理解每行代码的含义，大部分代码直接拿来使用就可以。

1) 注册窗口类

在 Windows 中，调用 `CreateWindow()` 函数可以创建一个窗口（请看上面的代码）。窗口有很多属性，比如大小、位置、标题、背景颜色、鼠标样式、图标等，在创建窗口时都需要指定。这些属性比较多，超过10 个，但是有一部分是通用的，不同的窗口，它们的值一般相同，Windows 将这些通用的属性抽取出来，用一个结构体表示，就是上面代码中 `WNDCLASS`（window class 缩写）：

1. `WNDCLASS wndclass; //定义窗口类`
2. `//为窗口类的各个字段赋值`
3. `wndclass.style = CS_HREDRAW | CS_VREDRAW; //窗口风格`
4. `wndclass.lpfnWndProc = WndProc; //窗口过程`
5. `wndclass.cbClsExtra = 0; //暂时不需要理解`
6. `wndclass.cbWndExtra = 0; //暂时不需要理解`
7. `wndclass.hInstance = hInstance; //当前窗口句柄`
8. `wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION); //窗口图标`
9. `wndclass.hCursor = LoadCursor (NULL, IDC_ARROW); //鼠标样式`
10. `wndclass.hbrBackground= (HBRUSH) GetStockObject (WHITE_BRUSH); //窗口背景画刷`
11. `wndclass.lpszMenuName = NULL; //窗口菜单`
12. `wndclass.lpszClassName= szClassName; //窗口类名`

这个结构体，我们称之为窗口类。如果你有面向对象的编程经验，那么会很容易理解，没有的话也没关系，你可以认为，基于该结构体创建的窗口属于同一个类别，有很多属性是相同的。

注意最后的字段 `lpszClassName`，它指明了当前窗口类的名字，将这个名字传递给 `CreateWindow()` 函数，就能根据该窗口类来创建窗口。也就是说，以后想使用窗口类，只要知道它的名字就可以（也就是字段 `lpszClassName` 的值）。

窗口类仅仅是一个结构体，如果只是定义了结构体变量，那么在使用时并不能通过 lpszClassName 字段的值找到这个结构体。所以还要调用 RegisterClass() 来注册，让系统知道窗口类的名字，下次使用时才能找到。

作为简明教程，我们并不打算深入研究窗口类的每一个字段的含义，下面是对它们的简要说明：

字段	说明
style	窗口风格。对于初学者，常用的取值为 CS_HREDRAW CS_VREDRAW，表示当窗口大小改变时重绘窗口，这样才能保证文字始终处于窗口中间。style 还有很多取值，这里不一一讲解，有兴趣的读者可以查看： WNDCLASS 中 style 字段的取值 （ wndclass.style 的取值 ）
lpfnWndProc	窗口处理过程，下面会详细讲解。
hInstance	当前窗口句柄。
hIcon	窗口图标。也就是程序运行时在左上角和任务栏看到的图标，需要通过 LoadIcon 函数加载。
hCursor	鼠标样式。需要通过 LoadCursor 函数加载。
hbrBackground	窗口背景画刷。也就是窗口背景的填充颜色，后面我们会讲解画笔、画刷和画布的概念。
lpszMenuName	窗口菜单。也就是标题栏下方看到的多种多样的菜单，上面的程序没有菜单，所以值为 NULL。
lpszClassName	窗口类的名字。每个窗口类的名字都是不同的，以便与其他窗口类区分。

2) 创建窗口

有了窗口类，就可以根据它来创建窗口了。创建窗口使用 CreateWindows() 函数，如下所示：

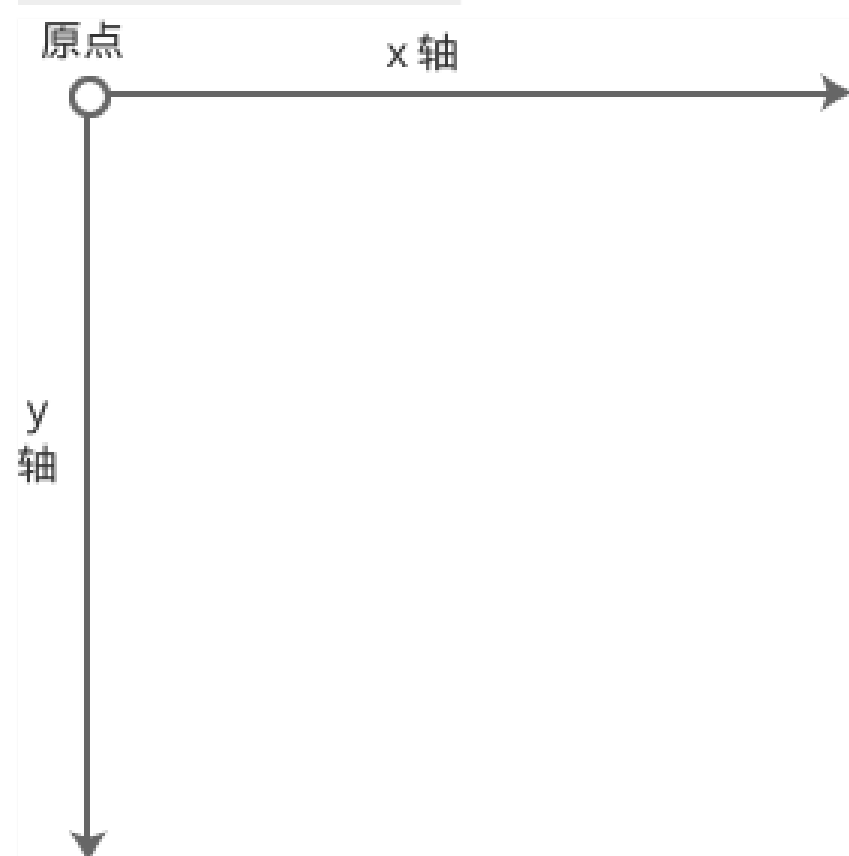
```
1.      hwnd = CreateWindow(  
2.          szClassName,    // 窗口类的名字  
3.          TEXT("Welcome"), //窗口标题（出现在标题栏）  
4.          WS_OVERLAPPEDWINDOW, //窗口风格  
5.          CW_USEDEFAULT,  //初始化时窗口 x 轴坐标  
6.          CW_USEDEFAULT,  //初始化时窗口 y 轴坐标  
7.          500,            //窗口宽度  
8.          300,            //窗口高度  
9.          NULL,          //父窗口句柄。这里没有父窗口，所以为 NULL  
10.         NULL,          //窗口菜单句柄。当前窗口没有菜单，所以为 NULL  
11.         hInstance,     //当前窗口的句柄，通过 WinMain 函数传入。  
12.         NULL           //不使用该值  
13.     );
```

几点说明：

A) CreateWindow 的第一个参数就是窗口类的名字，通过这个名字可以找到刚才注册的窗口类，然后再根据它来创建窗口。

B) 显示器上的坐标与数学中的不同，显示器的左上角是坐标原点，从原点向右是x 轴，向下是y 轴，都是正坐

标，没有负数。如下图所示：



C) 参数 `hInstance` 是通过主函数 `WinMain` 传入的。

注意：通过 `CreateWindows()` 函数创建窗口后，仅仅是为窗口分配了内存空间，获得了句柄，但窗口并没有显示出来，所以还需要调用 `ShowWindow()` 函数来显示窗口。

而调用了 `ShowWindow()` 函数又仅仅是将窗口显示出来，但不会进行客户区的绘制，所以还需要调用 `UpdateWindow()` 函数，生成 `WM_PAINT` 消息，将客户区中的各种控件绘制出来，下面会讲解。

至此，一个窗口的创建工作就已经完成了。窗口的各种属性，在窗口类和 `CreateWindow()` 函数的参数中都进行了说明。

注意：在窗口类 `wndclass` 中指定的窗口样式以 `CS` 开头，是通用的；而在 `CreateWindow`

函数中指定的窗口样式以 `WS` 开头，只对当前窗口有效，详情请查看《[CreateWindow 窗](#)

[口风格取值](#)》。

3) 进行消息循环

在 `UpdateWindow` 函数被调用之后，新建的窗口在屏幕中就可以显示了。此时，程序必须能够接受用户的键盘或鼠标事件，例如按下回车键、右击鼠标等。

在《[与 windows 编程有关的重要概念](#)》一节中讲到了 Windows 的消息机制。Windows 会为每个应用程序维护一个消息队列，当有事件发生时，Windows 会自动将这些事件转换为“消息”，并投递到消息队列。

在我们的程序中，可以通过一段“消息循环”代码来从消息队列中获取消息：

1. `while( GetMessage(&msg, NULL, 0, 0) ){`
2. `TranslateMessage(&msg); //翻译消息`

3. DispatchMessage (&msg); //分派消息
4. }

GetMessage 函数用来从消息队列中获取一条消息，并保存到 MSG 结构体变量中。作为简明教程，我们不再详细分析 GetMessage 函数的各个参数，读者根据上面的代码“照猫画虎”就可以，不会影响你后续的学习。

注意：GetMessage 的返回值永远为非零值，while 循环会一直进行下去。如果队列中没有消息，GetMessage 函数会等待，直到有消息进入。

获取到消息后，需要调用 TranslateMessage 函数对消息进行转换（翻译），然后再调用 DispatchMessage 函数将消息传给窗口过程去处理（调用窗口过程）。

4) 窗口过程

所谓窗口过程，就是处理窗口事件的函数，也就是上面代码中最后的 WndProc 函数。GetMessage 每获取到一条消息，最终都会丢给窗口过程去处理。

窗口过程有一个参数 message，会传入发生的事件类型，常用的有：

- WM_CREATE：窗口被创建。WM_PAINT
- ：窗口需要更新或重绘。WM_WM_DESTROY
- ：窗口被销毁（关闭）。

WM_CREATE 和 WM_DESTROY 很容易理解，WM_PAINT 将在下节中详细讲解，它非常重要，不理解 WM_PAINT 可以说就没有学会 Windows 编程。

不同的消息往往需要进行不同的处理，所以一般通过 switch case 语句来匹配。

注意：你可以对获取到的消息进行处理，加入自己的业务逻辑；也可以不处理，让Windows 自己看着办（默认处理方式）。窗口过程最后一条语句：

```
return DefWindowProc (hwnd, message, wParam, lParam) ;
```

它的作用就是让 Windows 自己处理应用程序没有处理的消息，必须要有该语句。

窗口过程在窗口类中指明，然后就不用管了，不需要我们显式调用。

最后的总结

上面讲到的，是开发一个 Windows 应用程序的基本流程，也是 Windows 应用程序的代码模板，你不需要记住每个细节，直接套用就可以。

编写 Windows 应用程序的步骤：

- 注册窗口类

- 根据窗口类来创建窗口
- 进入无休止的消息循环
- 编写窗口过程

有了代码模板，剩下的主要工作就是处理各种各样的事件了，也就是在窗口过程中编写代码。

绘制和重绘，有效矩形和无效矩形

当用户在屏幕移动其他程序的窗口时，可能会遮住当前程序的部分窗口。Windows 不会保存被遮住的那部分窗口，当其他程序的窗口被移开时，Windows 会要求你的程序重新绘制刚才被遮住的部分区域。如果你自己不重绘，Windows 是不会管的。

Windows 是一个消息驱动的系统，它使用两种方式把各种事件通知给应用程序：

- 把消息放到应用程序的消息队列中，让程序自己通过 GetMessage 函数获取；
- 向窗口直接发送消息。

WM_PAINT 消息

WM_PAINT 消息表示绘制窗口的客户区。窗口过程会经常收到 WM_PAINT 消息，而且能够在必要的时候绘制客户区。

以下任何一个事件发生时，窗口过程都会收到一条 WM_PAINT 消息：

- 用户移动一个窗口，导致原来被遮住的部分窗口暴露出来。
- 用户调整窗口的大小（当窗口类中 style 字段的值设定为 CS_HREDRAW | CS_VREDRAW 时）。客户区滚动条滚动时。
- 当然，你可以可以调用相应的函数强制生成一条 WM_PAINT 消息。

注意：窗口类中 style 字段的值经常被设定为 CS_HREDRAW | CS_VREDRAW，只有这样，调整窗口宽度或高度时才会发生重绘。上节我们在客户区的中间位置输出一段文本，这是一个明显的例子，如果 style 字段的值不是 CS_HREDRAW | CS_VREDRAW，那么调整窗口大小时文本就不会自动调整，也就是不在客户区中间了。

在少数情况下，Windows 总是会保存被覆盖的部分窗口，然后再恢复，例如：

- 鼠标指针在客户区内移动。
- 在客户区内拖动图标。

读者需要记住，在 Windows 中一切界面都是绘图，一旦界面发生改变，就需要重新绘制。

有效矩形和无效矩形

Windows 一般不重绘整个客户区，而是绘制客户区的一部分。最明显的一个例子是当一个对话框覆盖了客户区的一部分时，对话框关闭时，只有先前被覆盖的那个矩形部分需要重新绘制。

需要重新绘制的部分被称为“无效区域”或“更新区域”。在客户区中有一个无效区域将导致 Windows 向应用程序队列中放置一条 WM_PAINT 消息。也就是说，只有当客户区的一部分失效时，窗口过程才会接收到 WM_PAINT 消息。

无效区域有时候是矩形，有时候是圆、多边形等，但总有一个最小的矩形可以覆盖整个无效区域，如下图所示：

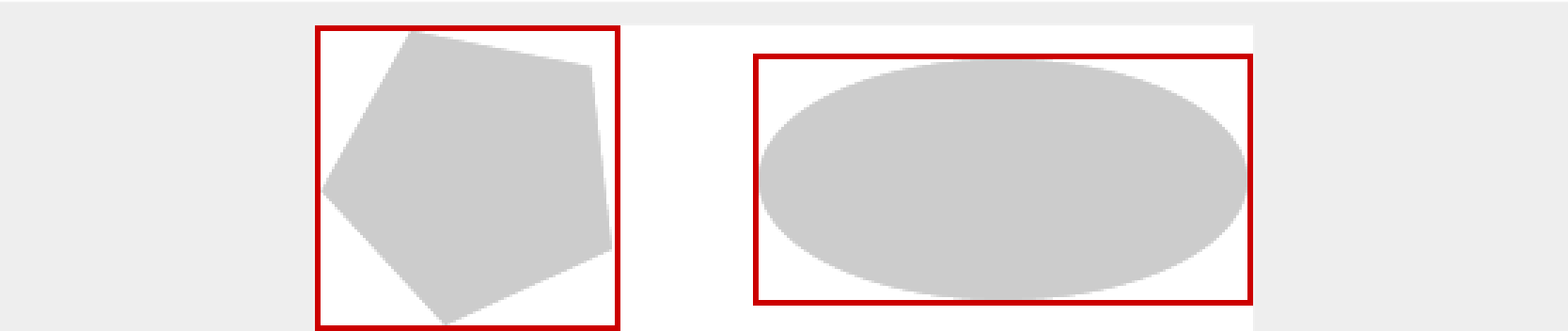


图 1：可以覆盖无效区域的最小矩形

上图中，灰色阴影表示无效区域，红色边框表示最小矩形。这个最小的矩形，就称为**无效矩形**。Windows 会自动保存无效矩形的坐标和大小，无需我们操心，当发生重绘时，总是重绘整个无效矩形区域。

当程序处理完 WM_PAINT 消息，无效矩形就会变成有效的，整个客户区也就有效了。

向窗口输出文字--TextOut 和 DrawText 函数

在《[一个完整的 Windows 程序框架](#)》一节中，我们使用 DrawText() 向客户区的中间位置输出了一段文本，但并没有详细解释，这节我们来重点说一下文字的输

设备环境（DC）

在 Windows 中，一切能在显示器上看到的都是图形，不但包括点、线、矩形、圆等，也包括文字、字体。对用户来说文字和图片是不同的，但它们在本质上都是图形，都需要在显示器上画出来。这也是 Windows 被称作图形操作系统的原因。

Windows 不允许我们直接访问硬件，如键盘、显示器、打印机等，在与这些硬件通信前要获得**设备环境（Device Context，简称 DC）**，进而间接的访问硬件。

设备环境有时也被称为设备上下文或设备描述表。

所谓设备环境，就是与当前硬件设备有关的各种信息，它是硬件设备的抽象。也就是说，设备环境就代表当前的硬件。

与文字输出、图形绘制有关的函数，在使用时大都需要传入一个参数，就是设备环境句柄。获取设备环境句柄有多种方式，常用的是 BeginPaint 函数。绘图完成后，还要释放句柄，使用 EndPaint 函数。

BeginPaint 和 EndPaint 函数

当发生 WM_PAINT 事件时，应用程序开始绘图。绘图之前需要先调用 BeginPaint 函数通知 Windows，告诉它当前程序需要使用显示器，你安排一下吧。BeginPaint 函数执行完毕就返回设备环境句柄，接下来就可以使用 GDI 函数绘图了。

绘图结束后，也需要调用 EndPaint 函数通知 Windows，告诉它绘图结束了，你可以继续后面的工作了。

TextOut 函数

TextOut 函数可以在客户区输出一段文本，原型为：

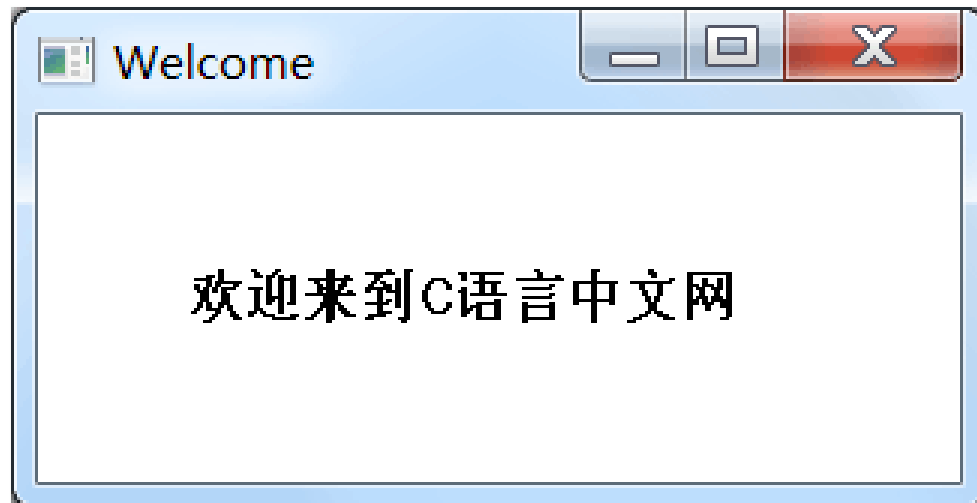
```
1.      BOOL TextOut(  
2.          HDC hdc,    //设备环境句柄  
3.          int nXStart, //开始输出位置的 x 坐标  
4.          int nYStart, //开始输出位置的 y 坐标  
5.          LPCTSTR lpString, //要输出的字符串  
6.          int cbString //字符串的长度  
7.      );
```

每当发生 WM_PAINT 事件时，窗口就会发生重绘，这个时候需要向窗口输出文字：

```
1.      LRESULT CALLBACK WndProc(  
2.          HWND hwnd,  
3.          UINT message,  
4.          WPARAM wParam,  
5.          LPARAM lParam  
6.      ){  
7.          HDC          hdc; //设备环境句柄  
8.          PAINTSTRUCT ps; //存储绘图环境的相关信息  
9.          TCHAR szText[20] = TEXT("欢迎来到 C 语言中文网");  
10.  
11.         switch (message){  
12.             case WM_PAINT:  
13.                 //开始绘图并返回环境句柄  
14.                 hdc = BeginPaint (hwnd, &ps);  
15.                 //输出文字  
16.                 TextOut(hdc, 50, 50, szText, wcslen(szText));  
17.                 //结束绘图并释放环境句柄  
18.                 EndPaint (hwnd, &ps) ;  
19.                 return 0 ;  
20.  
21.             case WM_DESTROY:
```

```
22.         PostQuitMessage(0);
23.         return 0;
24.     }
25.     return DefWindowProc(hwnd, message, wParam, lParam);
26. }
```

运行效果：



几点说明：

1) PAINTSTRUCT 结构体用来保存与绘图环境有关的信息，本教程不会使用到，不再讲解，有兴趣的读者可自行 Google 或百度。

2) BeginPaint 和 EndPaint 函数用来开始绘图和结束绘图，必须成对出现，输出文字或绘制图形的代码也必须放在两个函数之间。

3) 字符串一般使用宽字符，对应的计算字符串长度的函数为 wcslen()。

DrawText 函数

TextOut 在一个起点开始输出文本，可以精确定位，不能自动换行；DrawText 在指定区域内输出文本，可以控制格式对齐（居左、居中、居右），可以换行。

DrawText 在内部其实也是调用 TextOut，不过它作了很多处理，比如 DrawText 无需知道输出多少个字符。更重要的是，DrawText 可以处理换行符而 TextOut 不能，从这个角度看，DrawText 应该是一种更高级的用法，是在 TextOut 基础上的封装。

DrawText 的原型是：

```
1.     int DrawText(
2.         HDC hdc,    //设备环境句柄
3.         LPCTSTR lpString,  //将要输出的字符串
4.         int nCount,  //字符串的长度
5.         LPRECT lpRect,  //指向一个矩形区域的结构体 RECT 的指针
6.         UINT uFormat  //输出格式
7.     );
```

两点说明：

1) 如果 nCount 为 -1，则表明 lpString 指向的字符串是以'\0'结束的，DrawText 会自动计算字符数。

2) 参数 uFormat 有四种取值，它们可以任意组合：

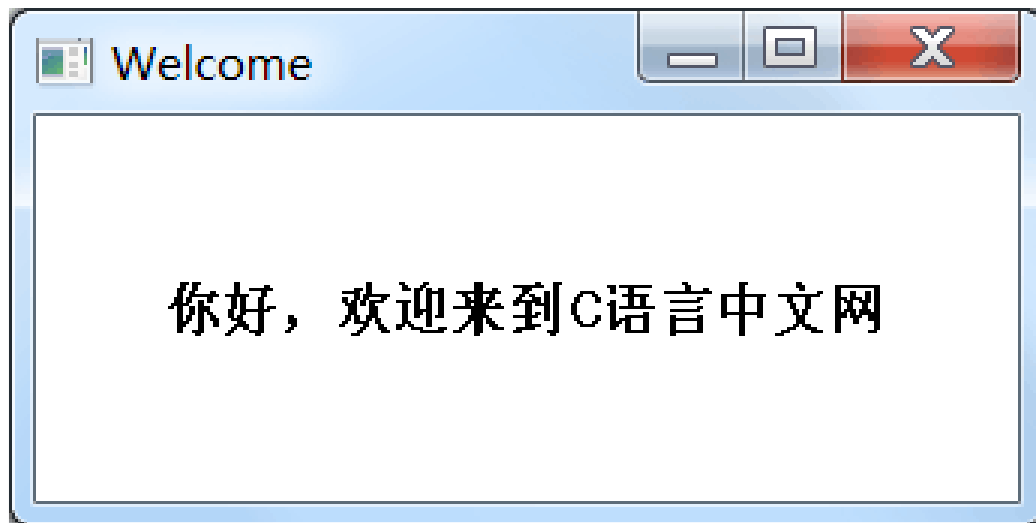
- DT_CALCRECT：这个参数比较重要，可以使 DrawText 函数计算出输出文本的尺寸。如果输出文本有多行，DrawText 函数使用 lpRect 定义的矩形的宽度，并扩展矩形的底部以容纳输出文本的最后一行。如果输出文本只有一行，则 DrawText 函数改变矩形的右边界，以容纳下正文行的最后一个字符。出现上述任何一种情况。
- DT_CENTER：指定文本水平居中显示。
- DT_VCENTER：指定文本垂直居中显示。该标记只在单行文本输出时有效，所以它必须与 DT_SINGLELINE 结合使用。
- DT_SINGLELINE：单行显示文本，回车和换行符都不换行。
-

DrawText 示例：

- ```
1. //窗口过程
2. LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam){
3. HDC hdc; //设备环境句柄
4. PAINTSTRUCT ps;
5. RECT rect;
6.
7. switch (message){
8. //窗口绘制消息
9. case WM_PAINT:
10. hdc = BeginPaint (hwnd, &ps);
11. GetClientRect (hwnd, &rect);
12. DrawText(
13. hdc,
14. TEXT("你好，欢迎来到 C 语言中文网"),
15. -1,
16. &rect,
17. DT_SINGLELINE | DT_CENTER | DT_VCENTER
18.);
19. EndPaint (hwnd, &ps);
20. return 0;
21.
22. //窗口销毁消息
23. case WM_DESTROY:
```

```
24. PostQuitMessage(0);
25. return 0;
26. }
27. return DefWindowProc(hwnd, message, wParam, lParam);
28. }
```

运行结果：



改变窗口大小时，文本会自动调整位置，一直处于窗口中间。

GetClientRect 函数用来获得窗口客户区的坐标，也就是获得客户区这个巨型。

## Windows GDI 绘图基础与轻量进阶

GDI 是 Graphics Device Interface 的缩写，称为图形设备接口，主要用来绘图，由动态链接库 GDI32.DLL 提供支持。

GDI 就是一个函数库，提供了很多绘图函数（也就是 GDI32.DLL 中的导出函数），上节使用的 TextOut 就是其中之一。GDI 非常重要，不但应用程序使用它来绘图，Windows 本身也使用 GDI 来显示用户界面，比如菜单、滚动条、图标和鼠标指针等。

### GDI 基础

这一部分讲解如何绘制简单的图形，而在轻量进阶部分讲解如何美化图形。

#### 1) 绘制矩形

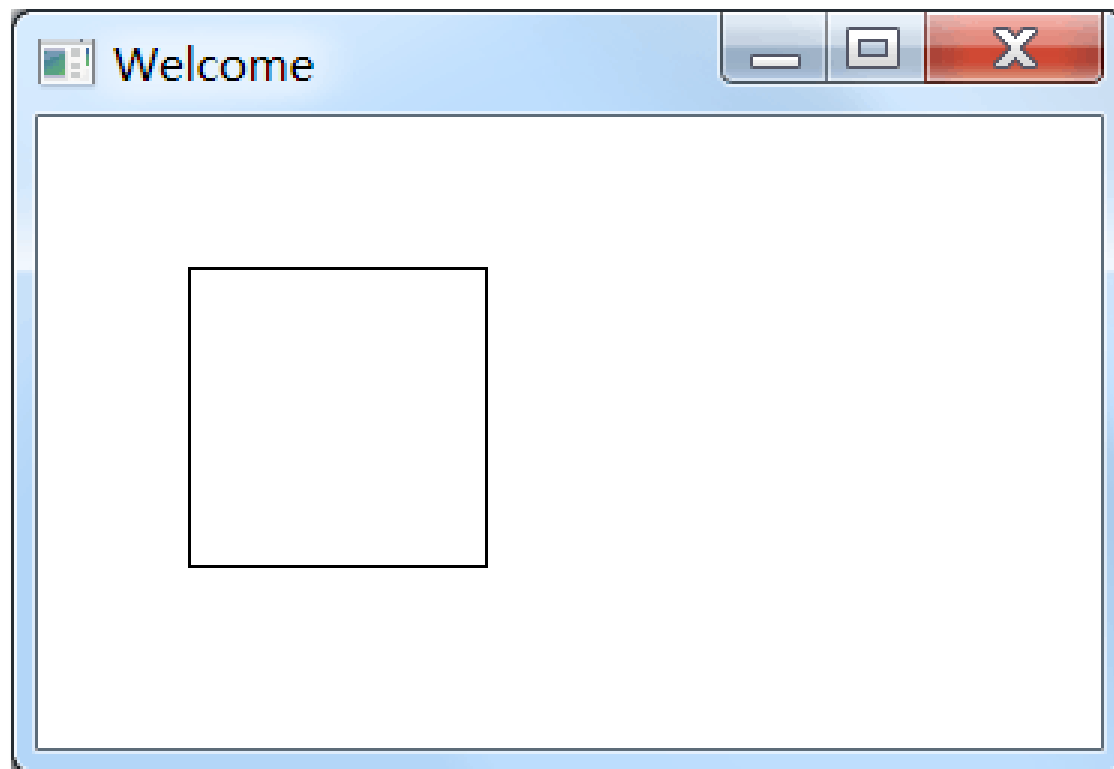
Rectangle 函数可以在窗口上绘制一个矩形，它的原型为：

```
1. BOOL Rectangle(
2. HDC hdc, //设备环境句柄
3. int nLeftRect, //矩形左上角 x 坐标
4. int nTopRect, //矩形左上角 y 坐标
5. int nRightRect, //矩形右下角 x 坐标
6. int nBottomRect //矩形右下角 y 坐标
7.);
```

示例代码：

```
1. case WM_PAINT:
2. hdc = BeginPaint(hwnd, &ps);
3. Rectangle(hdc, 50, 50, 150, 150);
4. EndPaint(hwnd, &ps);
5. return 0;
```

运行效果：



注意：坐标的原点都是客户区的左上角。

## 2) 带圆角的矩形

如果您觉得上面矩形过于方正，那么可以使用 RoundRect 函数，它可以画出带有圆角边框的矩形，原型为：

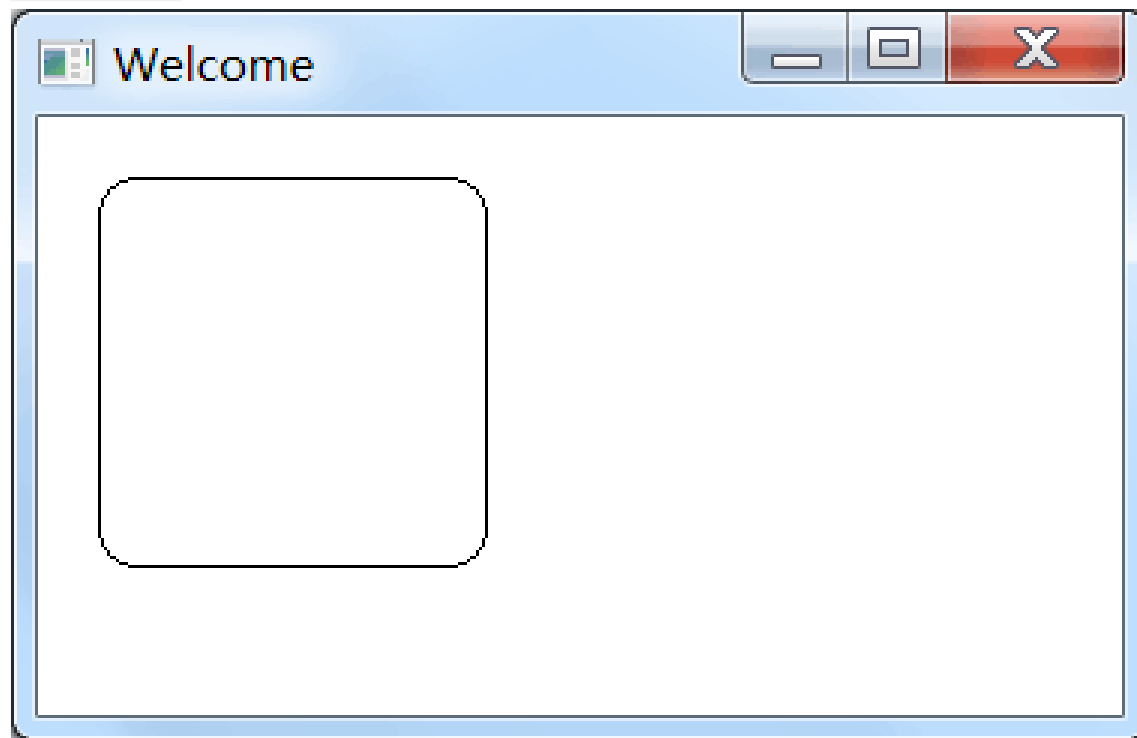
```
1. BOOL RoundRect(
2. HDC hdc, //设备环境句柄
3. int nLeftRect, //矩形左上角 x 坐标
4. int nTopRect, //矩形左上角 y 坐标
5. int nRightRect, //矩形右下角 x 坐标
6. int nBottomRect, //矩形右下角 y 坐标
7. int nWidth, //用来画圆角的椭圆的宽度
8. int nHeight //用来画圆角的椭圆的高度
9.);
```

注意：当  $nHeight \geq nBottomRect$  且  $nWidth = nRightRect$  时，那么绘制出的就是一个圆。

示例代码：

```
1. case WM_PAINT:
2. hdc = BeginPaint(hwnd, &ps);
3. RoundRect(hdc, 20, 20, 150, 150, 25, 25);
4. EndPaint(hwnd, &ps);
5. return 0;
```

运行效果：



### 3) 绘制椭圆

Ellipse() 函数可以用来绘制椭圆，它的原型为：

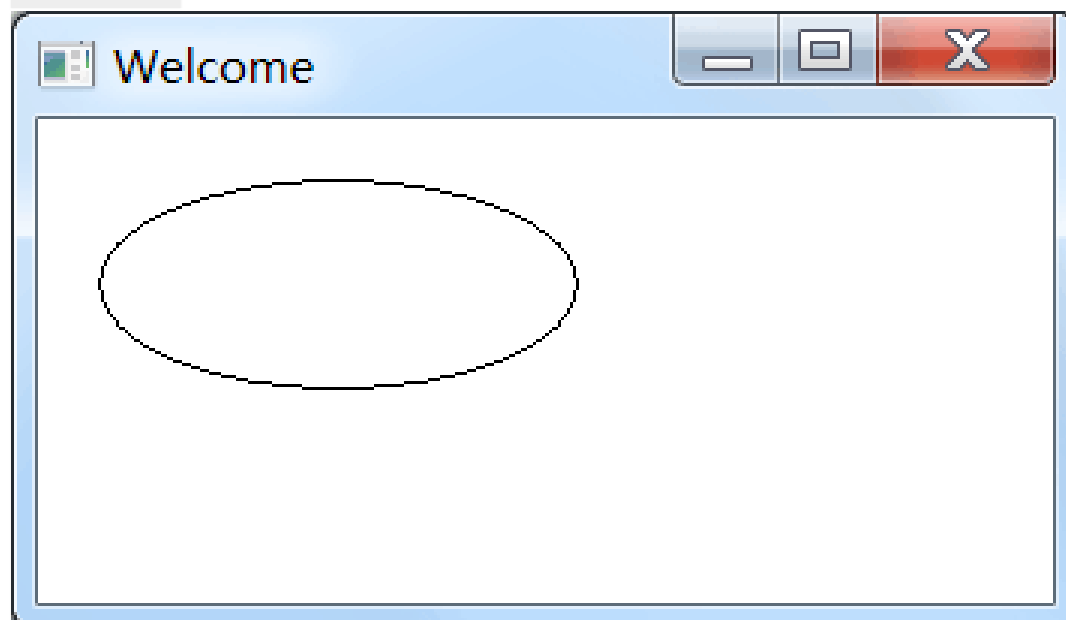
1.        **BOOL** Ellipse(  
2.        HDC hdc,    //设备环境句柄  
3.        **int** nLeftRect,    //左上角 x 坐标  
4.        **int** nTopRect,    //左上角 y 坐标  
5.        **int** nRightRect,    //右下角 x 坐标  
6.        **int** nBottomRect    //右下角 y 坐标  
7.        );

注意：当  $nRightRect - nLeftRect = nBottomRect - nTopRect$  时绘制出的是一个圆。

示例代码：

1.        **case** WM\_PAINT:
2.        hdc = BeginPaint(hwnd, &ps);
3.        Ellipse(hdc, 20, 20, 180,90);
4.        EndPaint(hwnd, &ps);
5.        **return** 0;

运行效果：



#### 4) 绘制直线

绘制直线需要确定起点和终点。

确定起点使用 MoveToEx 函数。MoveToEx 用来指定画笔的起始位置，也就是从哪里开始画，它的原型为：

```
1. BOOL MoveToEx(
2. HDC hdc, //设备环境句柄
3. int x, //起始位置 x 坐标
4. int y, //起始位置 y 坐标
5. LPPOINT lpPoint //指向用于保存当前位置的 POINT 结构体的指针
6.);
```

对于参数 lpPoint，我们并不需要保存当前位置，所以直接指定为 NULL 即可。

注意：win32 不再支持 MoveTo，只支持它的扩展函数 MoveToEx。

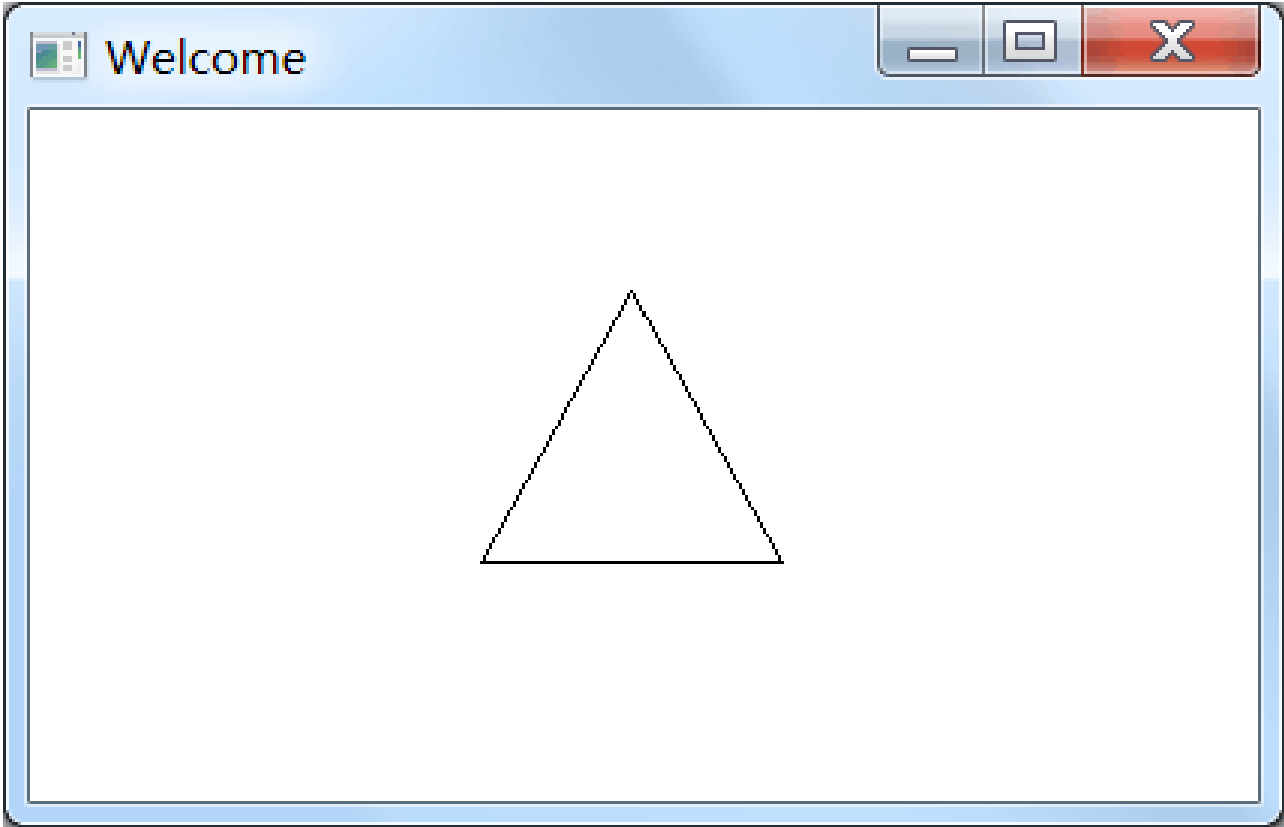
有了起点，接下来就可以使用 LineTo 函数画直线了。LineTo 函数用于从当前绘图位置向指定点绘制一条直线，它的原型为：

```
1. BOOL LineTo(
2. HDC hdc, //设备环境句柄
3. int xEnd, //终点的 x 坐标
4. int yEnd //终点的 y 坐标
5.);
```

示例代码：

```
1. case WM_PAINT:
2. hdc = BeginPaint(hwnd, &ps);
3. MoveToEx(hdc, 150, 150, NULL); //设定起始点,不保存当前点坐标
4. LineTo(hdc, 200, 60); //第一条线
5. LineTo(hdc, 250, 150); //第二条线
6. LineTo(hdc, 150, 150); //第三条线
7. EndPaint(hwnd, &ps);
8. return 0;
```

运行效果：



## GDI 绘图轻量进阶--画笔和画刷

画笔和画刷都用来在画布上绘图。画布就是用来绘画的一块背景，可以有颜色也可以没有，也可以有图案（比如条纹、网格等）；画布可以理解为我们平时作图时使用的纸。

画笔用来画线，可以是封闭的也可以是开放的，比如直线、曲线、圆形、矩形等。

画刷用来填充背景或者一块区域，一般带颜色或图案。

比如画一个矩形，需要先找一块画布，然后用画笔画出矩形的轮廓（线条），再用画刷给矩形区域喷涂上颜色。

画笔与画刷的区别：画笔一般用来画线条，画轮廓；画刷一般用来进行大面积绘制，比如给背景着色，填充画笔画出的一个封闭的区域等。

### 1) 创建和使用画笔

上面的绘图使用的是 Windows 的默认画笔，也就是宽度为1 个像素，颜色为黑色的画笔。我们也可以创建自己的画笔。

创建画笔的 API 函数为 CreatePen：

```
1. HPEN CreatePen(
2. int nPenStyle, //画笔的样式
3. int nWidth, //画笔的宽度
4. COLORREF crColor //画笔的颜色
5.);
```

画笔样式 nPenStyle 有 7 种取值：

| 宏定义 | 宏定义对应的值 | 说明 |
|-----|---------|----|
|-----|---------|----|

|                |   |                    |
|----------------|---|--------------------|
| PS_SOLID       | 0 | 实线                 |
| PS_DASH        | 1 | 虚线（段线），要求画笔宽度 <= 1 |
| PS_DOT         | 2 | 点线，要求画笔宽度 <= 1     |
| PS_DASHDOT     | 3 | 线、点，要求画笔宽度 <= 1    |
| PS_DASHDOTDOT  | 4 | 线、点、点，要求画笔宽度 <= 1  |
| PS_NULL        | 5 | 不可见                |
| PS_INSIDEFRAME | 6 | 实线，但画笔宽度是向里扩展的     |

画笔宽度 nWidth 指逻辑宽度。iWidth 为 0 则意味着画笔宽度为一个像素。如果画笔样式为点线或者虚线，同时又指定一个大于 1 的画笔宽度，那么 Windows 将使用实线画笔来代替。

画笔的颜色 crColor 可以直接使用 RGB 颜色。RGB 是一种标准颜色，通过红(R)、绿(G)、蓝(B)三原色的叠加得到各种不同的颜色。

细心的读者可能已经发现，CreatePen 函数在创建画笔时并没有指定设备环境，也就是说，新创建的画笔与当前设备环境并没有关联，无法使用。

画笔、画刷、字体等被称为 **GDI 对象**。你可以将 GDI 对象理解为工具，可以供 GDI 函数使用。新创建的 GDI 对象必须通过 SelectObject 函数选入设备环境才能使用。

SelectObject 函数将 GDI 对象与设备环境关联起来，它的原型为：

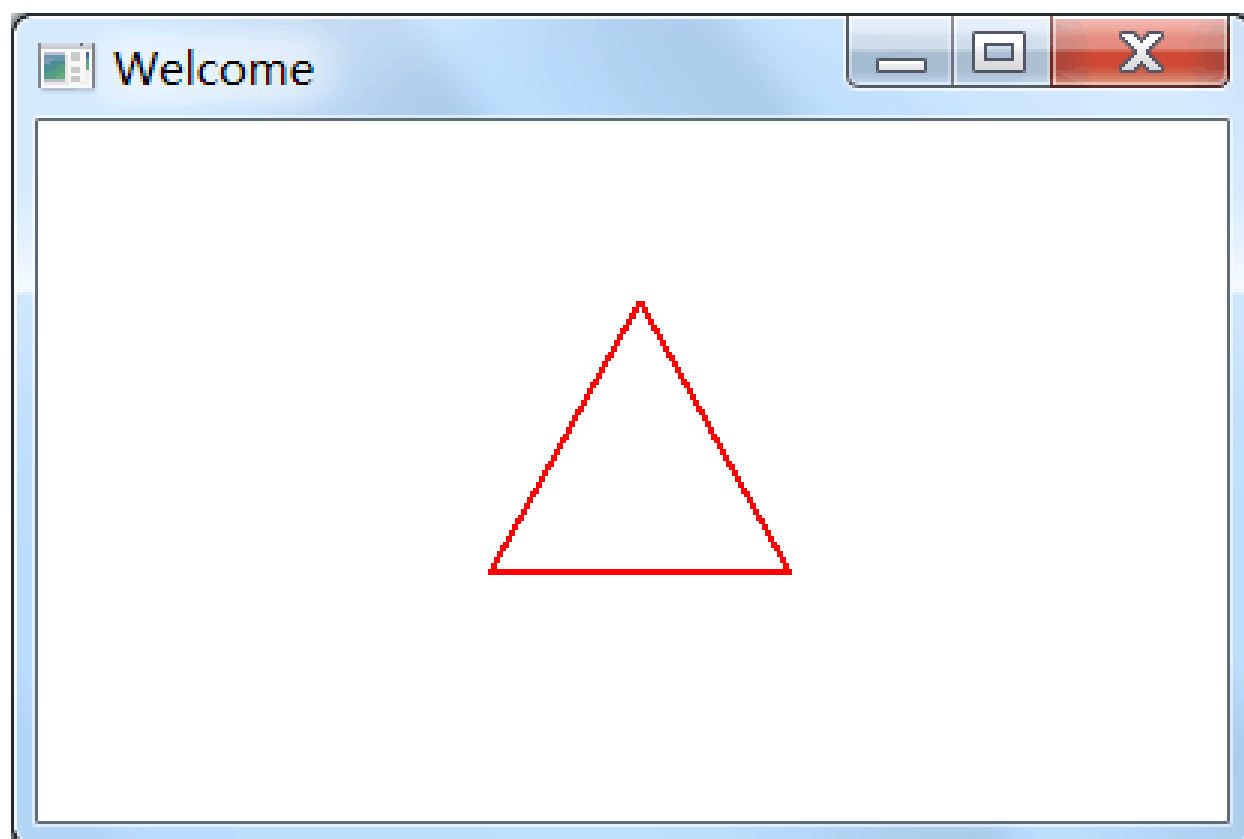
```
1. HGDIOBJ SelectObject(
2. HDC hdc, //设备环境句柄
3. HGDIOBJ ho //GDI 对象句柄
4.);
```

下面的代码会创建一个红色的画笔，并画出一个三角形：

```
1. //窗口过程
2. LRESULT CALLBACK WndProc(HWND hWnd, UINT message , WPARAM wParam, LPARAM
 lParam){
3. PAINTSTRUCT ps;
4. HDC hdc;
5. //定义一个画笔句柄,请定义为静态变量
6. static HPEN hPen;
7.
8. switch (message){
9. case WM_CREATE:
10. //创建宽度为 2 个像素的红色点线画笔，保存句柄到 hPen 变量
11. hPen = CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
```

```
12. break;
13. case WM_PAINT:
14. hdc = BeginPaint(hWnd, &ps);
15. //选入画笔到设备环境
16. SelectObject(hdc, hPen);
17. //绘制三角形
18. MoveToEx(hdc, 150, 150, NULL);
19. LineTo(hdc, 200, 60); //第一条线
20. LineTo(hdc, 250, 150); //第二条线
21. LineTo(hdc, 150, 150); //第三条线
22. EndPaint(hWnd, &ps);
23. break;
24. case WM_DESTROY:
25. //请做好善后工作,处理 WM_DESTROY 消息时删除之前我们创建的一
 切 GDI 对象
26. DeleteObject(hPen);
27. PostQuitMessage(0);
28. break;
29. default:
30. return DefWindowProc(hWnd, message, wParam, lParam);
31. }
32. return 0;
33. }
```

运行效果：



两点注意：

A) 画笔最好在 WM\_CREATE 事件中创建，因为当应用程序运行时频繁地触发 WM\_PAINT 事件，比如窗口被覆盖后再显示、窗口被拖动、窗口被拉伸等，每次都需要重新创建画笔，浪费资源，也没有必要。

B) 所有创建的 GDI 对象，在窗口被关闭时（会触发 WM\_DESTROY 事件）都要删除掉，以释放内存。

## 2) 创建和使用画刷

Windows API 中有两个函数可以用来创建画刷。

CreateSolidBrush 函数可以用来创建一个指定颜色的实心画刷，原型为：

```
HBRUSH CreateSolidBrush(COLORREF crColor); // crColor 为画刷颜色
```

CreateHatchBrush 函数可以用来创建一个指定颜色的含有特定阴影样式的画刷，原型为：

```
1. HBRUSH CreateHatchBrush(
2. int fnStyle, //画刷样式
3. COLORREF crColor //画刷颜色
4.);
```

fnStyle 可以有 6 种取值：

- HS\_BDIGNAL：45 度向上，自左至右的阴影(//)
- HS\_CROSS：表示水平直线和垂直直线交叉阴影(+++)
- HS\_DIAGCROSS：45 度交叉阴影(XXX)
- HS\_FDIAGONAL：45 度向下自左至右的阴影(\\)
- HS\_HORIZONTAL：水平阴影(---)
- HS\_VERTICAL：垂直阴影

画刷使用举例：

复制纯文本新窗口

```
1. //窗口过程
2. LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
 lParam){
3. PAINTSTRUCT ps;
4. HDC hdc;
5. //定义两个画刷,请定义为静态变量
6. static HBRUSH hSolidBrush;
7. static HBRUSH hHatchBrush;
8. switch (message){
9. case WM_CREATE:
10. //创建蓝色实心画刷，保存句柄到 hSolidBrush 变量
11. hSolidBrush = CreateSolidBrush(RGB(0, 0, 255));
12. //创建绿色交叉阴影画刷，保存句柄到 hHatchBrush 变量
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：  
<https://d.book118.com/505232112322011230>