第一章

课程内容：

介绍采用WIN API和C/C++语言开发Windows应用(被称为SDK)、wxPython开发Windows和Linux跨平台应用的方法。

我们要设计的是**desktop应用**

API – Application Programming Interface　　操作系统提供的一组基本函数
SDK – Software Developer's Kit　　最基本的程序开发方法

Windows应用程序与控制台程序的差异

①显示界面：Windows为应用程序提供带有窗口、菜单、对话框等特征的图形用户界面，多个应用程序用“窗口”共享显示屏；
　控制台程序利用stdout为程序准备系统显示。

②输入：Windows系统接收来自键盘、鼠标、定时器等的输入，并分配给相应应用程序的“消息队列”，应用程序从消息队列中检索消息，并且消息包含丰富的内容（如WM_KEYDOWN、WM_KEYUP中包含虚拟键码、键盘扫描码及Shift、Control等控制键的状态等); 控制台程序使用getchar, scanf, operator>> 等函数从键盘读入数据，它等待用户输入，然后返回字符码（通常只是字符在指定字符集中的编码）给程序。

③输出：Windows采用与设备无关的图形输出界面（GDI）, 使应用程序能方便地进行图形和文字输出（例：用同一API函数在显示器和打印机中输出同一图形或文字）；控制台程序使用putchar, printf, operator<< 等函数直接存取设备。

④资源使用：Windows 多任务，各应用程序共享系统资源（如CPU时间、内存、I/O设备等）；控制台程序认为可以独占资源。**Windows程序通过句柄(handle)使用系统资源。**

Windows编程模型

①窗口：包含标题栏、菜单栏、工具栏、系统按钮、客户区、滚动条、状态栏等。应用程序创建一个窗口后，从技术上说对它具有独占权，**但实际管理是由应用程序和Windows操作系统协作完成的。**因此，**每个窗口必须存在一个相应的“窗口过程/函数”**（window procedure / window function），它接收必须作出适当反应的窗口管理消息，并对消息进行处理。

②菜单：用户输入的方式之一。创建应用程序后，Windows系统为应用程序显示和管理菜单，并在用户选择菜单项时向窗口过程发送一个消息，该消息是应用程序执行命令的信号。

③对话框：**是一个窗口**，为用户显示更多的有关命令的信息，它包括若干**控件**（如编辑控件、按钮控件、列表框等）。

④消息循环：应用程序通过消息队列接收输入，因此**必须有一个消息循环**，它从应用程序自己的消息队列中检索消息，并把消息发送给相应窗口的窗口过程。

例子：①用户按一个键Z，在应用程序的窗口中显示该字符。

按键相关消息进入系统队列，分配给相应应用队列，WinMain函数的消息循环通过调用API函数 GetMessage 让操作系统取出消息，然后调用 DispatchMessage ，**通过操作系统将得到的消息调用窗口过程函数(非直接调用，通过回调callback机制)**，在这个函数中调用API TextOut，通过操作系统在窗口中显示。

②用户窗口关闭产生 WM_QUIT消息，消息循环回调窗口过程函数，调用API PostQuitMessage，产生WM_QUIT消息；消息循环检索到WM_QUIT消息后，结束消息循环并终止程序。

Windows程序组成：①一个WinMain函数（含消息循环及其它功能）；②若干窗口过程；③资源描述文件 (.rc) 描述菜单、对话框、加速键等资源以及各种资源文件 (.bmp, .ico, .cur .....)

动态链接库DLL

运行过程中寻找API函数的代码，连接时只使用输入库(.lib)。(对比静态链接，连接阶段与函数库(.lib/.a)中的函数代码进行连接)

三个基本DLL(包括几千个API函数)：[windows\system32]

user32.dll　　提供窗口管理，包括整个Windows环境和应用的窗口管理。
kernel32.dll　提供系统服务，如：多任务、内存管理和其他资源管理等。
gdi32.dll　　提供图形设备接口。

常见DLL库后缀：.DLL .DRV .EXE .FON .TTF等

优点：减小应用代码量；方便资源共享；节省内存资源；方便应用程序功能的扩展(API代码更新)；可使用其他合适的程序设计语言
方便项目管理；方便软件的国际化(本地化)；不同平台都能适应

Windows应用程序开发步骤

①建立 Win32 GUI Project 或者 Win32 Application empty Project

②建立资源及资源描述文件(.rc)

③建立 .c / .cpp源程序文件

④将资源描述文件和源程序文件加入到Project中，建立执行文件　.rc → .res (RC – resource compile)

.cpp → .o (obj) (CPP compile)

.o(obj) + .a(lib)(需要输入库给出DLL的信息，输入库是静态库) → .exe (LINK)　　**. exe + .res → Windows Application.exe**

编程注意点

不要独占CPU资源；不要直接存取内存或设备等资源(通过句柄完成)；Windows能调用的函数的**连接方法**(参数的入栈顺序和堆栈的恢复)应使用WINAPI（WinMain等）、 CALLBACK(窗口过程)等；必须有一个主函数WinMain( )(为建立窗口做准备，建立窗口，显示窗口，建立一个消息循环)；使用API函数时，一般应检查其返回值(表示执行是否成功) (比如返回一个句柄等于0表示没有建立起来)；编程时，要注意移植性问题；标识符的命名常用匈牙利记法(Hungarian Notation)

标识符以一个或多个小写字母(表示其类型)开头，后加大写开头、大小写混写的单词(表示其意义)。如：

c:wchar tchar　　by: BYTE　　n:short　　i:int　　w WORD(unsigned short)　dw DWORD(unsigned long)

sz: string terminated by 0 character　h:handle

第二章

简单的程序由WinMain( )函数 和 窗口过程回调函数 MainWndProc( ) 构成。

WinMain函数

注册窗口类；建立和显示窗口；其它初始化工作；建立消息循环（消息循环至检索到WM_QUIT时终止）

窗口类（WindowClass） —— 描述窗口的属性（窗口过程名、菜单、光标等），必须注册，用于建立窗口

程序实例（Application Instance） —— **一个应用程序只有一个实例**(hInstance)，有些API函数需要该值作为参数

句柄（Handle） —— 标识一个确定对象（程序实例、窗口、菜单、内存块等）而由系统分配的一个整数值（系统内部表的索引值）。它是多任务环境下保护系统资源的一种手段，可防止应用程序直接存取各种资源。应用程序只能通过句柄存取各种资源。

```
//入口函数
int WINAPI WinMain(HINSTANCE hInstance,           //(实例句柄，为固定值)
                   HINSTANCE hPrevInstance,         //现在不需要这个参数
                   LPSTR     lpCmdLine,               //命令行传入的字符串，一般不用
                   int       nCmdShow);               //窗口的显示方式(如最大化)
```

WNDCLASS定义在 windows.h中

```
typedef struct _WNDCLASS {
    UINT style;                 //CS_HREDRAW(窗口大小改变需要重新显示窗口内容),
CS_VREDRAW,  CS_DBLCLKS(支持双击操作)
    WNDPROC lpfnWndProc;        //指向窗口过程函数的指针
    int cbClsExtra;
    int cbWndExtra;                         //设置为0即可
    HINSTANCE hInstance;
    HICON hIcon;                            //图标
    HCURSOR hCursor;                        //光标
    HBRUSH hbrBackground;                   //刷的句柄(设置背景颜色)
    LPCTSTR lpszMenuName;                   //菜单(用字符串来表示)
    LPCTSTR lpszClassName;                 //类名
 } WNDCLASS, *PWNDCLASS;
```

LPCTSTR:　Long Pointer Constant (T表示适应于不同编码风格)，可以作为一个字符串常数

注册窗口类：定义WNDCLASS变量；设置WNDCLASS变量的每个成员；**调用RegisterClass( )函数，返回0表示注册失败**(变量名不正确，类名与已存在类名冲突)

TEXT：是一种宏，适用于ANSI 和 UNICODE编码，可以自动匹配应用工程的编码方式。

建立窗口(11个参数)：

```
HWND CreateWindow (
    LPCTSTR lpClassName,      // registered class name
    LPCTSTR lpWindowName,  // window name
    DWORD dwStyle,                 // window style
    int x,                                    // horizontal position of
window
    int y,                                   // vertical position of window
    int nWidth,                           // window width
    int nHeight,                         // window height
    HWND hWndParent,           // handle to parent or owner window
    HMENU hMenu,                  // menu handle or child identifier
    HINSTANCE hInstance,      // handle to application instance
    LPVOID lpParam                 // window-creation data
);
```

第三个参数窗口风格常是WS_OVERLAPPEDWINDOW（WS_OVERLAEDPP 每一个窗口位置不同稍微有分开，WS_CAPTION，有标题栏WS_SYSMENU 有系统菜单，WS_THICKFRAME 薄边框，,WS_MINIMIZEBOX 有最小化按钮， WS_MAXIMIZEBOX 位或)

显示窗口： ShowWindow(hWnd, nCmdShow); 显示窗口，产生 WM_SIZE消息 和 WM_SHOWWINDOW消息

更新窗口：UpdateWindow(hWnd); 更新窗口，产生 WM_PAINT消息

建立消息循环：

```
typedef struct tagMSG {
      HWND        hwnd;                    //窗口句柄
      UINT          message;         //消息名称
      WPARAM  wParam;
      LPARAM    lParam;                 //两个参数
      DWORD     time;
      POINT        pt;
} MSG;
```

消息编号：大部分以WM_开头，由Windows.h定义，也有使用其它前缀的消息编号
例：WM_PAINT　　WM_CREATE　　WM_DESTROY　WM_QUIT　　WM_SIZE　　WM_KEYDOWN
WM_MOUSEMOVE　LB_ADDSTRING

```
MSG msg;
 while (GetMessage(&msg, NULL, 0, 0))      //取出头消息，是WM_QUIT退出
 {
    TranslateMessage(&msg);
    DispatchMessage(&msg);                    //对窗口函数进行回调，可以看成操作系统调用窗口过程函数
 }
//调用频率比较高，不能时间很慢
```

WinMain( )的**返回值** return (int)msg.wParam; 是WM_QUIT,由PostQuitMessage( ) 设置，通常为0

窗口过程函数

处理各种Windows分配来的消息（**大量未被处理的消息可由DefWindowProc 进行缺省处理**，如窗口的移动，显示的变化），可以来自消息队列(如窗口移动)，或直接来自应用程序(如ShowWindow产生)
主体为switch case 语句

```
LRESULT CALLBACK MainWndProc(
    HWND hwnd,            // handle to window
    UINT   uMsg,             // message identifier
    WPARAM wParam,  // first message parameter
    LPARAM lParam)     // second message parameter
 {   static HINSTANCE hInst;
  PAINTSTRUCT ps;
     HDC hdc;
      switch (uMsg)
      { case WM_CREATE:
        hInst = ((LPCREATESTRUCT) lParam) -> hInstance ;       //获取窗口的实例句柄
        // hInst =GetModuleHandle(NULL);
        return 0;
      case WM_PAINT: //  ……Paint the window's client area.
       hdc = BeginPaint (hWnd, &ps);                    //获取设备上下文句柄(Device Context handle)
             TextOut(hdc,20,10,hello,lstrlen(hello));       //显示一行信息，最后一个参数表示长度，不同的编码长度可能长度不同，用C的话可以使用-1
             EndPaint (hWnd, &ps);                    //释放句柄
             return 0;
      case WM_DESTROY: //  ……Clean up window-specific data objects.
                PostQuitMessage(0);
                return 0;
       default: return DefWindowProc(hwnd, uMsg, wParam, lParam);
```

在Window.h中定义的一些常用数据类型： HANDLE HWND HDC HCURSOR HBRUSH HPEN HICON HINSTANCE HMENU HGOLBAL HLOCAL HFONT ……
BOOL LRESULT UINT DWORD(四个字节不带符号) BYTE LPCSTR LONG
WPARAM LPARAM WNDPROC(函数名称)
COLORREF (00BBGGRR)
在Window.h中定义的一些常用宏：**MAKEINTRESOURCE**(将资源ID转成字符串类型)，RGB ，MAKEWPARAM，MAKELPARAM，**HIWORD，LOWORD （取32位数的高低16位）**

使用wxPython

```
import wx
class MyFrame(wx.Frame):          //相当于窗口
    def __init__(self):              //定义初始化构造函数
        wx.Frame.__init__(self, None, -1,  “My 1st Frame”, size=(800, 600))
//调用基类
        panel = wx.Panel(self, -1)          //相当于建立一个容器，上面可以放很多部件，-1表示和窗口一样大
        panel.Bind(wx.EVT_MOTION, self.OnMove)           //将一个事件和函数进行绑定，这个事件比消息要粗一点(可能几个消息对应一个事件)
        wx.StaticText(panel, -1, “Pos:”, pos=(10, 12))          //建立静态文本
          self.posCtrl = wx.TextCtrl(panel, -1, “”,pos=(40, 10))
          wx.StaticText(panel, -1, “HELLO”, pos=(10, 120))

def OnMove(self, event):
        pos = event.GetPosition()
        self.posCtrl.SetValue(“%s, %s” % (pos.x, pos.y))     //设置编辑框的内容

if __name__ == ‘__main__’:
   app = wx.App()           //建立一个应用，相当于建立一个实例句柄
   frame = MyFrame()              //建立一个窗口
   frame.Show(True)                 //显示窗口
   app.MainLoop()                 //进行消息循环
```

第三章

资源：图标，菜单，键盘加速键，对话框，光标，字符串，位图 7种基本资源

资源描述文件中的各种**可重复使用的只读数据**，包括系统定义的各种资源和自己定义的资源(使用各种资源编辑工具创建)。

图标

表示一个应用，大图标（*3232*） *小图标（1616*），分为系统定义和自定义两种，可以使用Image Editor(也称Graphics Editor)绘制，保存文件后缀为 .ico，还生成一个 .rc 资源描述文件（列出所有资源），以及一个让程序引用的资源头文件 resource.h

使用VS创建图标：创建一个共享文件，添加新建项（资源文件），然后添加资源(添加类型为Icon)，新建图像类型（分为大小），可以改变ID大小。点开资源包含，将只读符号指令改成 “winresrc.h”。保存，生成三个文件：1个ico文件，1个resource.h，一个rc 文件

使用自定义图标：添加 rc 文件(与源程序编码方式相同)，可以在注册窗口类的时候如下修改：

```
//wc.hIcon            = LoadIcon (NULL, IDI_APPLICATION);
    wc.hIcon            = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON1));
```

系统自定义的图标：IDI_APPLICATION ， IDI_ASTERISK，IDI_EXCLAMATION，IDI_HAND，IDI_QUESTION ， IDI_WINLOGO

获取图标句柄： 系统图标 HICON hIcon= LoadIcon(NULL,IDI_APPLICATION);

若是自定义句柄：LoadIcon(HINSTANCE,ICONNAME); ICONNAME表示法：1. "name" 2. MAKEINTRESOURCE(ID)

也可以通过SetClassLong(HWND,GCL_HICON,(long)图标句柄) ： 改变运行时的图标(文件浏览器的图标不会变)，也可以改变注册类中其他的属性

wx中： icon = wx.Icon(name="ICON文件"，type=wx.BITMAP_TYPE_ICO) frame.SetIcon(icon)

菜单

主菜单（顶层菜单） 下拉菜单（弹出菜单、子菜单） 级联菜单 浮动弹出式菜单 系统菜单

部分菜单项有选择标记，属性有菜单项的启用、禁用、灰化(一般灰化之后就禁用)

菜单项三要素：显示内容（文字或位图）菜单项ID、属性（选中、启用、灰化）

使用VS创建菜单：创建一个共享文件，添加新建项（资源文件），然后添加资源(添加类型为Menu),给菜单项写文字，前面加&确定访问键，设置属性，设置ID(**顶层菜单不能设置ID，只能用position指代指定顶层菜单**)

使用自定义图标：①添加 rc 文件，可以在注册窗口类的时候如下修改：

```
//wc.lpszMenuName     = NULL;
wc.lpszMenuName     = MAKEINTRESOURCE(IDR_MENU_CH);
```

②在createWindows设置菜单的句柄：

```
 HMENU hMenu=LoadMenu(hInst,MR(MenuID))
    CreateWindow(…,hMenu, , );            //override class menu，以下面为准
```

③在运行过程中改变：

```
SetMenu(HWND,HMENU);
DestroyMenu(HMENU);       //不在窗口中使用的菜单需要在程序中销毁
```

在程序中建立菜单:

```
MENU CreateMenu( );   //主菜单
 HMENU GetMenu(HWND); //主菜单
 HMENU CreatePopupMenu( ); //下拉菜单
 InsertMenu(HMENU,(UINT)pos, MF_..,    //MenuFlag, 句柄，显示的文字(可以添加访问键)
    ID,"facetext")
 AppendMenu(HMENU,MF_..,ID,"facetext");         //直接在顶层菜单后面加
```

MF_STRING： 用字符串表示
MF_SEPARATOR
MF_POPUP： 下拉菜单
MF_BYPOSITION： 指定菜单项的名称

```
GetSubMenu(HMENU,pos);          //获取菜单句柄的其他方法
    GetSystemMenu(HWND);
```

```
const int IDM_ALIGNLEFT=1000;
 const int IDM_ALIGNCENTER = IDM_ALIGNLEFT+1;
 const int IDM_ALIGNRIGHT= IDM_ALIGNCENTER+1;
  HMENU hMenuMain=GetMenu(hWnd);                    //获取主菜单
   HMENU hMenuAlign=CreatePopupMenu( );              //建立下拉菜单
   InsertMenu(hMenuMain, 3, MF_BYPOSITION|MF_STRING |MF_POPUP, (UINT)hMenuAlign,
 TEXT("Format(&M)")); //插入
   AppendMenu(hMenuAlign, MF_STRING,  IDM_ALIGNLEFT,
             TEXT("LeftAlign"));
   AppendMenu(hMenuAlign, MF_STRING, IDM_ALIGNCENTER,
              TEXT("CenterAlign"));
   AppendMenu(hMenuAlign,MF_STRING, IDM_ALIGNRIGHT,
              TEXT("RightAlign"));
```

若是unicode工程，rc文件也应该是unicode编码，还需要告诉编译器用unicode方式进行编译，修改project中的build option,在other resource compiler options 中添加 **--codepage = 65001(一种unicode编码)**，ANSI则不需要

菜单相关消息：WM_INITMENU：　　wParam=主菜单句柄　设置与菜单初始化相关的设置

WM_MENUSELECT：　lParam=选中项的菜单句柄　　LOWORD(wParam)=菜单项ID
HIWORD(wParam)=选择标志
如 MF_GRAYED　　MF_DISABLED　　MF_CHECKED　　MF_POPUP　MF_SYSMENU
MF_ENABLED

WM_INITPOPUPMENU：　wParam=弹出菜单句柄　LOWORD(lParam)=菜单位置
HIWORD(lParam)=系统菜单标志
 系统菜单=1，其他=0(每次打开下拉菜单)

WM_COMMAND：　　LOWORD(wParam)=ID

WM_SYSCOMMAND：　wParam=ID

菜单的位置与状态　指定菜单项　MF_BYPOSITION　MF_BYCOMMAND(用位置或者ID)

```
//状态
EnableMenuItem(HMENU,ID,状态)  3种
                        MF_GRAYED
                    MF_DISABLED
                    MF_ENABLED
CheckMenuItem(HMENU,ID,状态)  2种
                        MF_CHECKED
                    MF_UNCHECKED
CheckMenuRadioItem(HMENU,ID1,ID2,ID,标志)      //单选
```

修改菜单

```
AppendMenu
DeleteMenu(HMENU,ID,flag))
InsertMenu
ModifyMenu(HMENU,oldID,flag,ID,"facetext")
```

```
//其他API
DrawMenuBar(HWND)                    //使得修改后的菜单更新
GetMenuItemCount(HMENU)
GetMenuItemID(HMENU,pos)
GetMenuState(HMENU,ID,flag)
     返回状态组合MF_CHECKED  MF_UNCHECKED
                                MF_ENABLED   MF_GRAYED
```

//代码细节可见 MenuDemo.c

wx

建立菜单栏：mb=wx.MenuBar()
建立菜单标题：m=wx.Menu()
菜单事件：wx.EVT_MENU
单选菜单标志：wx.ITEM_RADIO

添加菜单栏：frame.SetMenuBar(mb)
添加菜单项：m.Append(id,"facetext","tip")
添加菜单标题：mb.Append(m,"facetext")
复选菜单标志：wx.ITEM_CHECK

frame.GetMenuBar()取得菜单栏
m.FindItemById(id)取得对应id的菜单项mi
evt.IsChecked()获取CHECK标志
mb.EnableTop(id, True/False)设置菜单标题状态

evt.GetId() 取得被选中菜单的id
mi.GetText()/mi.GetItemLabel()获取facetext
mb.Enable(id, True/False)设置菜单项状态
mb.Check(id, True/False)设置菜单项状态

//代码细节可见 MENUDEMO.PY

在OnPaint Paint事件要用 PaintDC, 其他地方是 ClientDC

消息框

int MessageBox(HWND,text,title,style);
          //HWND父窗口句柄 GetFocus( ) / NULL
// style: (下页表) 两类中各选一项进行位或(|)

```
if (MessageBox(hWnd,"太阳落了吗？","打开", MB_YESNO|MB_ICONQUESTION)==IDYES)
          MessageBox(hWnd,"good night!","hello", MB_OK|MB_ICONEXCLAMATION);
 else
          MessageBox(hWnd,"good afternoon!","hello", MB_OK|MB_ICONEXCLAMATION);
```

加速键

建立资源，装入加速键资源 HACCEL hAccel=LoadAccelerators(HINSTANCE,ID); 修改消息循环

WX 菜单加速键：在facetext中在\t后指定加速键 如："&Accelerated\tCtrl-A" 加速键：控制键 +/- 字母等

对话框

与用户交互(I/O)的弹出窗口，由各种类型的多个控件组成。

模式对话框：父窗口等待模式对话框，直至关闭。不能为WS_CHILD类型。显示用DialogBox( ) , 关闭用EndDialog( )

无模式对话框：父窗口无需等待无模式对话框关闭，可在父窗口和无模式对话框之间切换。类型为WS_POPUP 。建立用CreateDialog( ) , 而关闭用DestroyWindow( )，**需修改消息循环**。(无模式对话框的消息在父窗口的消息队列中；而模式对话框的消息在它自己的的消息队列中，由系统自动进行分配。)

```
HWND hDlgModeless = 0;
 MSG msg;
 while (GetMessage(&msg, NULL, 0, 0))
 {
   if (!IsWindow(hDlgModeless) ||
       !IsDialogMessage(hDlgModeless,&msg))
     {
       TranslateMessage(&msg);
     DispatchMessage(&msg);
     }
 }
 //  if ((!IsWindow(hDlgModeless) ||            //如果也有加速键
 //      !IsDialogMessage(hDlgModeless,&msg))
  //     && !TranslateAccelerator(hWnd,hAccel,&msg))
```

建立对话框资源：设置大小及其ID ， 设置其中的各个控件及其ID、属性等，对话框资源也称为对话框模板(template)。(可以设置按钮的缺省，有输入焦点)

控件简符：PUSHBUTTON　DEFPUSHBUTTON　CONTROL　CHECKBOX　　CTEXT　　GROUPBOX

建立对话框

```
//建立无模式对话框
HWND CreateDialog( HINSTANCE hInstance,
                    LPCTSTR lpTemplate,
                    HWND hWndParent,          //父窗口
                   DLGPROC lpDialogFunc
                                );
//关闭无模式对话框
BOOL DestroyWindow( HWND hDlg );
hDlgModeless=0;
```

```
case IDM_OPENMODELESS: //打开对话框的菜单命令
    if (!IsWindow(hDlgModeless)) //避免重复建立
    {   hDlgModeless=CreateDialog(
                hInst,
                MAKEINTRESOURCE(DIALOG1),
                hWnd,
                (DLGPROC)ModelessDlgProc);
          ShowWindow(hDlgModeless,SW_SHOW);
    }
```

```
//建立模式对话框,返回-1 失败,否则为EndDialog( )的第2个参数的值
int      DialogBox     ( HINSTANCE hInstance,
                                    LPCTSTR lpTemplate,
                                    HWND hWndParent,
                                    DLGPROC lpDialogFunc
                                  );
```

```
case IDM_OPENMODAL: //打开对话框的菜单命令
                DialogBox(  hInst,
                                    MAKEINTRESOURCE(DIALOG1),
                                    hWnd,
                                    (DLGPROC)ModeDlgProc
                                  );
               InvalidateRect(hWnd,NULL,TRUE);
                //当对话框的操作需要改变主窗口的客户区时

BOOL   EndDialog  ( HWND hDlg,
                                    int status
                                  );
```

正常返回TRUE。

建立对话框函数

```
BOOL CALLBACK DialogProc
                  ( HWND hDlg,
                    UINT uMsg,
                    WPARAM wParam,
                    LPARAM lParam
                  )
{   switch (uMsg)
      { case WM_INITDIALOG: // Initialize the dialog，对应create消息
                return TRUE;
        case WM_COMMAND:  // Process Command from control
                                                    // in the dialog
                return TRUE;
          //  Process other messages.
         default: return FALSE;
      }
}

//参数与窗口过程函数相同。
//差别:
//①BOOL    vs    LRESULT
//②WM_INITDIALOG   vs   WM_CREATE
//③default:     return FALSE       vs         return DefWindowProc(…)
```

TRUE表示DialogProc已处理过该消息，不需要系统的内部处理。
FALSE表示DialogProc没有处理过该消息，需要系统进行缺省处理。
例外情况：WM_INITDIALOG经过DialogProc处理后，返回TRUE表示需要系统做进一步处理（设置缺省按钮），而返回FALSE时，系统不做进一步处理（如用SetFocus( )指定输入焦点时）。

```
case WM_INITDIALOG :
{

        HWND x=GetDlgItem(hDlg,IDCANCEL);
        SetFocus(x);
}          return FALSE ;
 //          return TRUE ;
```

wx建立模式对话框:类wx.Dialog → SubclassDialog
建立：dialog = SubclassDialog()
显示：result = dialog.ShowModal()
关闭：dialog.Destroy()　　//详见WXDialog

对话框与主窗口的交互

①通过全局变量

②向父窗口发送消息

```
SendMessage (
       GetParent(hDlg),
       UINT Msg,                // message to send
       WPARAM wParam,  // first message parameter
       LPARAM lParam      // second message parameter
      );
```

③**使父窗口客户区无效**InvalidateRect(GetParent(hDlg),...)，即发送WM_PAINT消息

```
BOOL InvalidateRect (
        HWND hWnd, // handle to window
        CONST RECT *lpRect, // rectangle coordinates
        BOOL bErase // erase state
   );

typedef struct _RECT
{  LONG left;
   LONG top;
   LONG right;
   LONG bottom;
} RECT, *PRECT;
```

指定需要更新的窗口客户区域，产生WM_PAINT消息 lpRect==NULL 表示整个客户区，即rcPaint
bErase==TRUE 表示先需要擦掉原显示内容，产生WM_ERASEBACKGROUND消息

WM_PAINT消息放入消息队列的尾部，且需要与队列中**原有的WM_PAINT消息（若有的话）进行合并**，并在处理完其他消息后再处理WM_PAINT消息，以尽量减少窗口的实际更新次数，减少闪烁。

frame.Refresh() 产生EVT_PAINT事件。

WX获取对话框的值：①定义对话框的一个取值函数返回值，②主窗口通过对话框对象调用取值函数

//详见wxDialogGetValue

WX无模式对话框:类wx.Dialog → SubclassDialog
建立：dialog = SubclassDialog()
显示：dialog.Show()
关闭：dialog.Destroy()

使用对话框控件

设置复选按钮的状态：

```
 BOOL CheckDlgButton( HWND hDlg,
                                              int ID,
                                              UINT nCheck
                                            );
   nCheck :  BST_UNCHECKED / BST_CHECKED
                    / BST_INDETERMINATE
    BM_SETCHECK → 控件

       //读取复选按钮的状态：
UINT IsDlgButtonChecked ( HWND hDlg,
                                                 int ID,
                                               );
 返回 :      BST_UNCHECKED / BST_CHECKED
               / BST_INDETERMINATE

         BM_GETCHECK□控件
```

设置单选按钮的状态：

```
BOOL CheckRadioButton( HWND hDlg,
                                                 int IDFirst,
                                                 int IDLast,
                                                 int ID
                                             );

          BM_SETCHECK→控件
//将ID按钮的状态设置为BST_CHECKED ，
//原选中按钮的状态设置为BST_UNCHECKED
```

向控件发消息：

```
LRESULT SendDlgItemMessage(                    //同sendMessage
       HWND hDlg,              // handle to dialog box
       int nIDDlgItem,           // control identifier
       UINT Msg,                 // message to send
       WPARAM wParam,   // first message parameter
       LPARAM lParam       // second message parameter
   );

//获取控件句柄：
  HWND GetDlgItem (
       HWND hDlg,              // handle to dialog box
       int nIDDlgItem           // control identifier
  );
```

```
//改变控件的使用/禁用状态：
  BOOL EnableWindow (      //可以改变窗口
        HWND hDlgItem,
        BOOL bEnable
  );

//设置控件的显示文字：
 BOOL SetDlgItemText (
        HWND hDlg,              // handle to dialog box
        int nIDDlgItem,           // control identifier
        LPCTSTR lpString    // text to set
   );

//获取控件的显示文字：
 BOOL GetDlgItemText (
        HWND hDlg,              // handle to dialog box
        int nIDDlgItem,           // control identifier
        LPTSTR lpString,     // pointer to buffer for text
        int nMaxCount           // maximum size of string
   );

//设置控件的显示数值：
BOOL SetDlgItemInt (
        HWND hDlg,          // handle to dialog box
        int nIDDlgItem,      // control identifier
        UINT uValue,        // value to set
        BOOL bSigned   // signed or unsigned
                                        indicator
   );

//获取控件的显示数值：
 UINT GetDlgItemInt (
        HWND hDlg,               // handle to dialog box
        int nIDDlgItem,            // control identifier
        BOOL *lpTranslated, // success state            //表示是否成功，不是看返回值
        BOOL bSigned         // signed or unsigned
                                              value
   );
```

WX控件

wx.TextCtrl　　wx.EVT_TEXT_ENTER
　wx.Button　　wx.EVT_BUTTON
　wx.StaticText
　wx.StaticBitmap
　wx.CheckBox　　wx.EVT_CHECKBOX
　wx.RadioButton　wx.EVT_RADIOBUTTON
　wx.RadioBox　　wx.EVT_RADIOBOX
　wx.ListBox　　wx.EVT_LISTBOX

通用对话框：统一的用户界面(打开文件,另存文件,查找和替换,选择颜色,选择字体,打印文件)

步骤：#include <commdlg.h>，需要改变工程中的设置，build option中linker settings中添加libcomdlg32.a；定义特定数据结构的变量；将变量传递给相应的函数；函数返回相应的信息

OPENFILENAME GetOpenFileName
GetSaveFileName
FINDREPLACE FindText
ReplaceText
CHOOSEFONT ChoseFont
CHOOSECOLOR ChoseColor
PRINTDLG PrintDlg

```
OPENFILENAME ofn; // common dialog box structure
TCHAR szFile[260]= " "; // buffer for file name
HWND hwnd; // owner window

// Initialize OPENFILENAME
ZeroMemory(&ofn, sizeof(OPENFILENAME));       //清零
ofn.lStructSize = sizeof(OPENFILENAME);           //提供版本标记(每个版本大小不一样)
ofn.hwndOwner = hwnd;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFilter = "All(*.*)\0*.*\0Text(*.txt)\0*.TXT\0PDF
Files(*.pdf)\0*.PDF\0";          //限定文件
ofn.nFilterIndex = 1;
ofn.lpstrFileTitle = NULL;
ofn.nMaxFileTitle = 0;
ofn.lpstrInitialDir = NULL;
ofn.Flags = OFN_HIDEREADONLY | OFN_CREATEPROMPT;

// Display the Open dialog box.
if (GetOpenFileName(&ofn)==TRUE)
  MessageBox(hWnd,szFile, "文件名",MB_OK);
```

wx.FileDialog wx.DirDialog
wx.FontDialog wx.ColourDialog //详细见wxGenericDialog.py

光标

系统光标 自定义光标(设置作用点)
IDC_ARROW IDC_CROSS IDC_WAIT IDC_IBEAM IDC_...

获取光标句柄：
系统光标 HCUSOR hCusor = LoadCusor(NULL,IDC_ARROW); 同图标
自定义光标 LoadCusor(HINSTANCE,CUSORNAME);
CURSORNAME表示法：1."name" 2.MAKEINTRESOURCE(ID)

使用光标句柄：
1. WNDCLASS中 hCursor=光标句柄;
2.
2. 动态改变 SetCursor(光标句柄) (通常写在WM_MOUSEMOVE等的处理代码段中)

```
3. 动态改变 SetClassLong(HWND,GCL_HCURSOR,(long)光标句柄)
```

光标相关的其他API

```
HCURSOR GetCursor(VOID);
     int ShowCursor( BOOL bShow
                                                    // cursor visibility
                               );
     BOOL GetCursorPos( LPPOINT lpPoint
                                                    // cursor position
                                         );
     BOOL SetCursorPos( int X,   // horizontal position
                                          int Y     // vertical position
                                         );
```

```
 cursor = wx.Cursor(cursorName="Cursor文件",
                       type=wx.BITMAP_TYPE_CUR)

   frame. SetCursor(cursor)

StockCursor：wx.CURSOR_ARROW,
                        wx.CURSOR_ARROWWAIT,
                        wx.CURSOR_PENCIL……
```

字符串

用于开发多语言版本的应用程序，不同语言的版本只需重新编辑字符串资源，而无需改变源程序。程序中显示的文字用字符串资源进行描述，所有的字符串组成一个字符串表。

在程序中使用字符串

```
//读取字符串到程序变量：
  int LoadString( HINSTANCE hInstance,
                                          // handle to resource module
                           UINT uID,             // resource identifier
                           LPTSTR lpBuffer, // resource buffer
                           int nBufferMax      // size of buffer
                          );

TCHAR szBuffer[20];
LoadString(hInst,IDS_STR1,szBuffer,20);
```

位图

图片最原始的格式，用于在客户区显示。

在程序中使用位图资源

```
//获取位图句柄：
  HBITMAP LoadBitmap( HINSTANCE hInstance,
                                          // handle to application instance
                                          LPCTSTR lpBitmapName
                                          // name of bitmap resource
                                        );
//删除位图资源：
  DeleteObject( HBITMAP);
```

使用位图句柄在窗口客户区显示位图：(早期效率较差，需要中转一下，也叫兼容设备描述表)

1. 获取设备描述表句柄
2. 获取内存设备描述表句柄
3. 选择位图句柄，放入内存设备描述表
4. 从内存设备描述表复制位图到设备描述表

```
//获取内存设备描述表句柄
HDC CreateCompatibleDC(
            HDC hdc            // handle to DC
                                          );
// BOOL DeleteDC(
            HDC hdc           // handle to DC
                             );
//选择位图句柄，放入内存设备描述表
HGDIOBJ SelectObject(
      HDC hdc,     // handle to DC
      HGDIOBJ hgdiobj
                          // handle to object
                                        );
//从内存设备描述表复制位图到设备描述表
BOOL BitBlt (       // Bit Block Transfer
HDC hdcDest,  // handle to destination DC ，beginPaint的值
int nXDest,       // x-coord of destination upper-left corner
int nYDest,      // y-coord of destination upper-left corner
int nWidth,      // width of destination rectangle
int nHeight,     // height of destination rectangle
HDC hdcSrc,  // handle to source DC
int nXSrc,       // x-coordinate of source upper-left corner
int nYSrc,      // y-coordinate of source upper-left corner
DWORD dwRop  // raster operation code );           //光栅操作码，原来的点与现在的点现在的操作，SRCCOPY 表示无关
```

```
HBITMAP hBitmap =LoadBitmap(hInstance,
                                MAKEINTRESOURCE(IDB_BRICK));
    BITMAP bitmap; GetObject(hbitmap,sizeof(BITMAP),&bitmap);
    int  width=bitmap.bmWidth, height=bitmap.bmHeight;
    HDC hdc = BeginPaint (hWnd, &ps);
    HDC hMemDc=CreateCompatibleDC(hdc);
    SelectObject(hMemDc,hBitmap);
    BitBlt(hdc,100,200,width,height,hMemDc,0,0,SRCCOPY);
    DeleteDC(hMemDc);
    DeleteObject(hBitmap);
```

WX Bitmap　　详见wxBitMap

```
Image = wx.Image(name="bmp文件",
                         type=wx.BITMAP_TYPE_BMP)
   Bitmap = Image.ConvertToBitmap()
dcMem = wx.MemoryDC( )
 dcMem.SelectObject(Bitmap)

DC::Blit (       # Bit Block Transfer
 XDest,     # x-coord of destination upper-left corner
 YDest,     # y-coord of destination upper-left corner
 Width,      # width of destination rectangle
 Height,     # height of destination rectangle
 dcSrc,      # source DC
 XSrc,       # x-coordinate of source upper-left corner
 YSrc,       # y-coordinate of source upper-left corner
 logicalFunc # raster operation code
 )
```

第四章

键盘输入，字符输入， 鼠标输入，定时器输入，滚动条输入，菜单输入

键盘输入，鼠标输入，定时器输入，直接对应于硬件的输入；菜单输入，字符输入，滚动条输入是响应窗口的非客户区中的键盘或鼠标操作而产生的，或是翻译键盘消息而产生的。

键盘输入

击键→键盘驱动程序→ Windows系统→产生键盘消息

键盘扫描码：与具体设备相关的键编码　　VK_... A-Z 0-9 无 VK-... 形式的定义，它们的虚拟键码就是ASCII码

虚拟键码：与设备无关的键编码(关心这个)　　**键盘扫描码→键盘驱动程序→虚拟键码**

VK_RETURN　　VK_TAB　　VK_SHIFT　　VK_CONTROL
VK_MENU　　VK_CAPITAL　　VK_ESCAPE　　VK_PRIOR
VK_NEXT　　VK_INSERT
VK_DELETE　　VK_NUMLOCK　　winuser.h中定义值
VK_SCROLL

VK_LBUTTON　　VK_RBUTTON(鼠标按键)

WX 虚拟键码：1.wx.WXK_***　　_

2.For Windows:
import win32con
*win32con.VK****，　两种方式值不同

输入焦点：应用程序仅当它具有"输入焦点"时才能接收到键盘消息。

获取"输入焦点"的方法：用户选中应用窗口；用SetFocus( )指定，获得"输入焦点"时产生WM_SETFOCUS消息，而失去"输入焦点"时产生WM_KILLFOCUS消息。

```
HWND SetFocus( HWND hWnd
                                  // handle to window
                                );
 // 返回0表示出错，其他表示原具有“输入焦点”窗口的句柄。
HWND GetFocus( );
  //返回具有“输入焦点”窗口的句柄。
```

```
wx.EVT_ACTIVATE
event.GetActive()
frame.SetFocus()
frame.FindFocus()        //true表示有输入焦点
```

WM_KEYDOWN　　WM_KEYUP

WM_SYSKEYDOWN　　WM_SYSKEYUP　　系统键 – Alt+Key

wParam : 虚拟键码　　lParam : 附加信息

```
case WM_KEYDOWN:
    switch (wParam)
   {  case VK_LEFT:     … break;
       case VK_RIGHT:  … break;
       case VK_UP:          … break;
       case VK_DOWN: … break;
   }
```

连续按键会产生一系列down消息， 最后产生一个up消息

```
wx.EVT_KEYDOWN
 wx.EVT_KEYUP
 event. GetKeyCode()
```

字符消息

WM_CHAR　　(WM_KEYDOWN(可打印字符)　　TranslateMessage( ) → WM_CHAR)

```
MSG msg;
   while (GetMessage(&msg, NULL, 0, 0))
   {
       TranslateMessage(&msg);            //部分键盘消息转为字符消息
       DispatchMessage(&msg);
   }
```

wParam : ANSI码值，lParam : 附加信息（与WM_KEYDOWN相同）

A :　WM_KEYDOWN (A) →WM_CHAR ('A' / 'a') → WM_KEYUP

SHIFT A: WM_KEYDOWN (VK_SHIFT) →WM_KEYDOWN(A) → WM_CHAR('A'/'a') →WM_KEYUP →WM_KEYUP

读取字符值时，响应WM_CHAR，读取控制键时，响应WM_KEYDOWN

```
wx.EVT_CHAR
event. GetKeyCode()  //获得虚拟键码
```

鼠标输入

单击 双击 移动 滑轮滚动

支持双击，注册窗口类时设置风格,CS_DBLCLKS,双击是500ms内的两次单击

```
BOOL SetDoubleClickTime( UINT uInterval
                      // double-click interval
                         );
 // 0 表示 500
//可改变500ms间隔，并对整个系统生效。
```

分为客户区消息 ，非客户区消息 （系统缺省处理,WM_NC开头）

WM_MOUSEMOVE WM_MOUSEWHEEL WM_LBUTTONDOWN WM_LBUTTONUP WM_LBUTTONDBLCLK WM_R...

消息参数:
lParam : (y,x)坐标，可小于 0 (表示在窗口外)，也可大于窗口大小。 左上角为源点，右为x，下为y

wParam : 鼠标和键盘(Shift、Crtl等)的状态。 MK_SHIFT MK_CONTROL MK_LBUTTON MK_RBUTTON

```
case WM_LBUTTONDOWN:
      if (wParam & MK_CONTROL)
          MessageBox(NULL,"",
               "Crtl+Lbutton",…);
//也可使用GetKeyState来读取按键状态:
 // GetKeyState(VK_…)返回<0表示相应按键按下
case WM_LBUTTONDOWN:
      if (GetKeyState(VK_CONTROL)<0)
          MessageBox(NULL,"",
               "Crtl+Lbutton",…);
```

WM_MOUSEMOVE消息产生的频率依赖于鼠标设备和移动速度。

拖动：左键按下，鼠标移动，左键放下，一系列消息

在非活动窗口中单击，则变为活动窗口，并得到WM_LBUTTONDOWN消息。在其他窗口中按下鼠标键，然后移动到当前窗口，则只得到WM_LBUTTONUP消息。在当前窗口中按下鼠标键，然后移动到其他窗口，则只得到WM_LBUTTONDOWN和一些WM_MOUSEMOVE消息。

如要连续得到鼠标消息，则可以"捕获"鼠标。(暂时归某窗口使用)

```
HWND SetCapture( HWND hWnd );
BOOL ReleaseCapture( VOID );
```

WM_MOUSEWHEEL 等同于滚动条，LOWORD(wParam)　key states
　HIWORD(wParam)　wheel rotation +, -120　lParam　mouse position

wx.EVT_MOTION　wx.EVT_LEFT_DOWN _UP _DCLICK
.....　wx.EVT_MOUSEWHELL wx.EVT_ENTER_WINDOW　wx.EVT_LEAVE_WINDOW

evt.GetPosition()　evt.LeftIsDown()

frame.CaptureMouse()　frame.ReleaseMouse()　frame.HasCapture()

定时器输入

WM_TIMER消息，SetTimer设置定时器，KillTimer取消定时器，**对WM_TIMER消息进行响应或设置定时器处理的回调函数。**

定时器的硬件基础：内部时钟，分辨率10ms左右，定时器依赖于系统时钟，并不是很精确的，并且在消息队列中的WM_TIMER消息有可能不会及时得到响应，甚至丢失。

```
UINT_PTR SetTimer(
  HWND hWnd,// handle to window
  UINT_PTR nIDEvent,// timer identifier 定时器编号
  UINT uElapse,// time-out value        定时器间隔，单位为毫秒
  TIMERPROC lpTimerFunc// timer procedure
                 );
//重新设置定时间隔
SetTimer(hWnd,IDT_TIMER1,100,NULL);

BOOL KillTimer(
  HWND hWnd, // handle to window
  UINT_PTR uIDEvent // timer identifier
               );
case WM_TIMER:
   switch (wParam)
   {
     case IDT_TIMER1:
   ……
   }
  break;

VOID CALLBACK TimerProc(
  HWND hwnd, // handle to window
  UINT uMsg, // WM_TIMER message
  UINT_PTR idEvent, // timer identifier
  DWORD dwTime // current system time
                       );
GetClientRect(HWND,RECT*);
                  获取客户区大小
    FillRect(HDC,RECT*,HBRUSH);
```

```
wx.EVT_TIMER
timer = wx.Timer(frame)
timer.Start(elapse)
timer.Stop()                GetSize
```

滚动条输入

水平滚动条WS_HSCROLL,垂直滚动条WS_VSCROLL,Windows处理所有有关滚动条的鼠标操作

```
hWnd = CreateWindow ("scroll", "Scroll demo"), WS_OVERLAPPEDWINDOW | WS_HSCROLL
| WS_VSCROLL, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL,
NULL, hInstance, NULL) ;
```

滚动条的范围（用nMin/nMax表示）缺省值为[0，100]

```
//用SetScrollInfo改变滚动条的范围和位置。
int SetScrollInfo(
   HWND hwnd, // handle to window
   int fnBar, // scroll bar type   SB_HOR, SB_VERT
   LPCSCROLLINFO lpsi, // scroll parameters
   BOOL fRedraw // redraw flag  最后一个为true
                        );
typedef struct tagSCROLLINFO
{  UINT cbSize;
   UINT fMask;       //SIF_RANGE  SIF_POS  SIF_PAGE  SIF_ALL，表示这次设置哪一个信息
   int nMin;
   int nMax;
   UINT nPage;
   int nPos;
   int nTrackPos;
 } SCROLLINFO, *LPSCROLLINFO; typedef SCROLLINFO CONST *LPCSCROLLINFO;
//例子
SCROLLINFO si;
 case WM_SIZE:
     TotalLines=…;
     si.cbSize=sizeof(si);
     si.fMask=SIF_RANGE;
     si.nMin=0;
     si.nMax=TotalLines-1;
     SetScrollInfo(hWnd,SB_VERT,&si,TRUE);
```

设置范围和位置，处理滚动条消息，**更新滚动块位置**，更新客户区的相应显示内容(根据位置)

WM_HSCROLL，WM_VSCROLL，LOWORD(wParam): 通知码 SB_LINEUP SB_THUMBTRACK

SB_THUMBTRACK和SB_THUMBPOSITION：
    HIWORD(wParam): 位置值
   与SCROLLINFO中的nTrackPos相同

这两个消息的一般处理：如对客户区的操作较慢，则只处理SB_THUMBPOSITION，否则只处理SB_THUMBTRACK

所有对滚动条操作后发生的位置改变必须由程序来设置，使用 SetScrollInfo 。要获得滚动条的当前状态，使用 GetScrollInfo 。

```
BOOL GetScrollInfo(
  HWND hwnd, // handle to window
  int fnBar, // scroll bar type
  LPSCROLLINFO lpsi // scroll bar parameters
                                );
```

```
case WM_VSCROLL:
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_ALL ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    OriginalPos = si.nPos ;       //记录原始值
    switch (LOWORD (wParam))
   {  case SB_TOP:
        si.nPos = si.nMin ; break ;
      case SB_BOTTOM:
        si.nPos = si.nMax ; break ;
      case SB_LINEUP:
        si.nPos -= 1 ; break ;
      case SB_LINEDOWN:
        si.nPos += 1 ; break ;
      case SB_PAGEUP:
        si.nPos -= si.nPage ; break ;
      case SB_PAGEDOWN:
        si.nPos += si.nPage ; break ;
            case SB_THUMBTRACK:
       si.nPos = si.nTrackPos ; break ;
     default: break ;
   }
   si.fMask = SIF_POS ;
   SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;      //改变
   GetScrollInfo (hwnd, SB_VERT, &si) ;

   if (si.nPos != originalPos)        //若是改变了
   {  ScrollWindow (hwnd, 0,                  //对客户区内容进行改变
        (originalPos - si.nPos)*charHeight,
        NULL, NULL) ;
      UpdateWindow (hwnd) ;
    }
    return 0 ;

BOOL ScrollWindow(
  HWND hWnd, // handle to window
  int XAmount, // horizontal scrolling left<0
  int YAmount, // vertical scrolling up<0
  CONST RECT *lpRect, // client area
  CONST RECT *lpClipRect // clipping rectangle
                                  );
```

菜单(浮动弹出式菜单)输入

通常响应WM_RBUTTONDOWN消息，获取弹出式菜单句柄(主菜单的部分)，确定显示的屏幕坐标，显示与选择菜单项

```
//获取弹出式菜单句柄，通常为主菜单中的一个菜单标题
   HMENU hMenu=LoadMenu(hInst,…);
    HMENU hMenuTrackPopup=GetSubMenu(hMenu,pos);
//确定显示的屏幕坐标，客户区坐标→屏幕坐标(显示浮动式菜单的API所需要的)
BOOL ClientToScreen(
     HWND hWnd, // handle to window
     LPPOINT lpPoint // screen coordinates
                                      );
```

```
//显示与选择菜单项
BOOL TrackPopupMenu(
   HMENU hMenu, // handle to shortcut menu
   UINT uFlags, // options
   int x, // horizontal position
   int y, // vertical position
   int nReserved, // reserved, must be zero
   HWND hWnd, // handle to owner window
   CONST RECT *prcRect // ignored  );

/* TPM_CENTERALIGN    TPM_LEFTALIGN       TPM_RIGHTALIGN        TPM_BOTTOMALIGN
     TPM_TOPALIGN
TPM_VCENTERALIGN        TPM_LEFTBUTTON(LBUTTON)
TPM_RIGHTBUTTON(RBUTTON|LBUTTON)*/

    case WM_RBUTTONUP:
          point.x = LOWORD (lParam) ;
          point.y = HIWORD (lParam) ;
          ClientToScreen (hwnd, &point) ;

          TrackPopupMenu (hMenu, TPM_RIGHTBUTTON, point.x, point.y,
                          0, hwnd, NULL) ;
          return 0 ;
```

wx.EVT_CONTEXT_MENU event.GetPosition() pos = ScreenToClient(pos)
PopupMenu(menu, pos)

第五章

应用程序通过GDI (Graphics Device Interface,图形设备界面)实现输出，实现了**输出的设备无关性。**

应用设备→GDI→设备驱动程序→输出设备

设备描述表：Device Context (DC)是GDI内部保存的一个数据结构，用于定义图形对象(画笔、画刷、字体、文本颜色、背景色、映射方式、坐标原点、当前绘图位置等)及其属性，并且影响图形或文本的输出方式。

①程序需要进行输出操作时，必须首先获取DC的句柄HDC。获取HDC时，Windows系统使用缺省的设备描述表属性值。

②应用程序可使用一些特定的API改变其中的某些属性值。

③在窗口客户区进行输出操作时，按照DC中当前的属性进行显示。

④输出完成后，需要释放HDC。释放后，不能继续使用。一般在一个消息处理过程中获取、使用和释放HDC, 不在两个消息之间保存一个HDC。

获取HDC的方法：

①在WM_PAINT消息处理过程中，使用BeginPaint和EndPaint。BeginPaint能指定无效区域，一般在开始输出前将无效区域的背景擦除，并且**使无效区域变为有效。**因此，在处理WM_PAINT时，必须调用BeginPaint。 wx.PaintDC(frame)

```
HDC BeginPaint(
    HWND hwnd, // handle to window
    LPPAINTSTRUCT lpPaint // paint information
                              );
 BOOL EndPaint(
    HWND hWnd, // handle to window
    CONST PAINTSTRUCT *lpPaint // paint data
                              );

BOOL InvalidateRect (
        HWND hWnd, // handle to window
        CONST RECT *lpRect, // rectangle coordinates
        BOOL bErase // erase state
   );
/*指定需要更新的窗口客户区域，产生WM_PAINT消息
 lpRect==NULL 表示整个客户区，即rcPaint
 bErase==TRUE 表示先需要擦去原内容，产生WM_ERASEBACKGROUND消息*/
```

②在非WM_PAINT消息处理过程中，使用GetDC和ReleaseDC。GetDC不能指定无效区域(重绘整个客户区，背景不变，并且不会使无效区域变为有效)。要使整个客户区变为有效，可使用：
ValidateRect(hWnd,NULL);　　wx.ClientDC(frame)

```
HDC GetDC(
      HWND hWnd // handle to window
                    );

 int ReleaseDC(
     HWND hWnd, // handle to window
     HDC hDC // handle to DC
                       );
```

获取设备描述表信息

```
int GetDeviceCaps(HDC,iIndex)
iIndex: HORZSIZE,VERTSIZE,HORZRES,VERTRES,LOGPIXELSX,LOGPIXSELSY
    //GetTextMetrics()
```

像素点 DPI，每一英寸用多少个点来表示，通过系统注册表修改这个值

PaintDC → ClientDC → WindowsDC

在DC中设置绘图对象

```
//SelectObject把画笔、画刷、字体、位图等对象选入到DC中。
 HGDIOBJ SelectObject(
    HDC hdc, // handle to DC
    HGDIOBJ hgdiobj   // handle to object
                                        );

 //释放画笔、画刷、字体等对象（建立方法后面介绍）。
 BOOL DeleteObject(
    HGDIOBJ hObject
             // handle to graphic object
                                    ); //不能删除当前选入DC中的图形对象!
```

画点:

```
//所有输出操作的基础。
 COLORREF SetPixel(
    HDC hdc, // handle to DC
    int x, // x-coordinate of pixel
    int y, // y-coordinate of pixel
    COLORREF crColor // pixel color );
//返回所设像素点的颜色值，有可能与所设参数的值有极其细微的区别。（某些颜色值在有些显示设备中无法显示出来）。
COLORREF GetPixel(
    HDC hdc, // handle to DC
    int x, // x-coordinate of pixel
    int y // y-coordinate of pixel);
 //当(x,y)不正确时，返回CLR_INVALID 小于0的数
BOOL SetPixelV(
    HDC hdc, // handle to device context
    int x, // x-coordinate of pixel
    int y, // y-coordinate of pixel
    COLORREF crColor // new pixel color );
    //不返回颜色值，效率比SetPixel要高。
```

dc.SetPen(wx.Pen("red", 1, wx.SOLID))
dc.DrawPoint(p.x,p.y)

画直线

```
BOOL MoveToEx (
       HDC hdc, // handle to device context
       int X, // x-coordinate of new current position
       int Y, // y-coordinate of new current position          //从画笔当前位置画，当前位置可以用下面的API
       LPPOINT lpPoint // old current position
                                    );
//画笔位置缺省值为上次画线结束处，MoveToEx函数和LineTo函数要一起使用才能达到画线的效果
     // 初始值为坐标原点 (0,0)
BOOL GetCurrentPositionEx (
       HDC hdc, // handle to device context
       LPPOINT lpPoint // current position
                                    );
```

```
BOOL LineTo(
     HDC hdc, // device context handle
     int nXEnd, // x-coordinate of ending point
     int nYEnd // y-coordinate of ending point
     );
```

dc.DrawLine(p1.x,p1.y, p2.x,p2.y)

```
//画折线
BOOL Polyline(
   HDC hdc, // handle to device context
   CONST POINT *lppt, // array of endpoints
   int cPoints // number of points in array ); //不会改变画笔的当前位置
BOOL PolylineTo(
   HDC hdc, // handle to device context
   CONST POINT *lppt, // array of endpoints        //起始点也算
   DWORD cCount
            // number of points in array
                                );
  //从当前位置开始画，
  //画完后改变画笔的当前位置。
dc.DrawLines(pList)

//画几条折线
BOOL PolyPolyline(
   HDC hdc, // handle to device context
   CONST POINT *lppt, // array of points
   CONST DWORD *lpdwPolyPoints, // array of values
   DWORD cCount // number of entries in values array
                                 );
  //不会改变画笔的当前位置

    //画曲线
BOOL Arc (
  HDC hdc, // handle to device context
  int xLeft, // x-coord of rectangle's upper-left corner
  int yTop, // y-coord of rectangle's upper-left corner
  int xRight, // x-coord of rectangle's lower-right corner
  int yBottom, // y-coord of rectangle's lower-right corner
  int xStart, // x-coord of first radial ending point
  int yStart, // y-coord of first radial ending point
  int xEnd, // x-coord of second radial ending point
  int yEnd // y-coord of second radial ending point
                  );

    BOOL PolyBezier(
  HDC hdc, // handle to device context
  CONST POINT *lppt, // endpoints and control points
  DWORD cPoints // count of endpoints and control points
                             );

 //两个端点(p0,p3)和两个控制点(p1,p2)
BOOL PolyBezierTo(
  HDC hdc, // handle to device context
  CONST POINT *lppt, // endpoints and control points
```

```
  DWORD cCount // count of endpoints and control points
                                  );
//一个端点(p3)和两个控制点(p1,p2)
 //p0是当前画笔位置
```

```
gc = wx.GraphicsContext.Create(dc)
path = gc.CreatePath()
path.MoveToPoint(p0)
path.AddCurveToPoint(p1,p2,p3)
gc.DrawPath(path)

dc.DrawSpline(pList)
```

(p0)LBUTTONDOWN→MOUSEMOVE→LBUTTONUP(p3)

LBUTTONDOWN→MOUSEMOVE→LBUTTONUP(p1)

LBUTTONDOWN→MOUSEMOVE→LBUTTONUP(p2)

设置绘图模式 当使用画笔画图形时，执行画笔像素与目标位置处原像素之间的某种二进制位运算。这种位运算称为两元光栅运算（ROP2 – raster operation）。

```
int SetROP2(
   HDC hdc, // handle to DC
   int fnDrawMode // drawing mode
                  );
//缺省值为R2_COPYPEN，画一条线
//使用SetROP2(hdc,R2_NOTXORPEN) 时，在同一区域再画一次时的效果是擦除所画内容。
```

dc.SetLogicalFunction(wx.COPY)
dc.SetLogicalFunction(wx.INVERT)　对应

画笔

缺省画笔：BLACK_PEN: 1像素宽的黑色实线笔

库存画笔：WHITE_PEN,NULL_PEN,BLACK_PEN，用GetStockObject获取库存画笔

```
//自定义画笔
1.HPEN CreatePen(
         int iPenStyle, // pen style
         int nWidth, // pen width
        COLORREF crColor // pen color
                                 );
//nWidth=0表示1。
 //nWidth>1表示style为PS_SOLID或PS_NULL。

2. HPEN CreatePenIndirect(
      CONST LOGPEN *lplgpn
           // style, width, color
                                          );
typedef struct tagLOGPEN
   {   UINT lopnStyle;
       POINT lopnWidth; //只用到宽度
```

```
        COLORREF lopnColor;
    } LOGPEN, *PLOGPEN;

LOGPEN GreenPen =
                {PS_SOLID , 1 , 0 , RGB(0,255,0) };
HPEN pen=CreatePenIndirect(&GreenPen);
```

WX: PEN　　wx.RED_PEN　　wx.CYAN_PEN　wx.TRANSPARENT_PEN　　wx.GREY_PEN

```
wx.Pen(colour, width, style)
Style:
   wx.SOLID
   wx.TRANSPARENT
   wx.DOT
```

画封闭图形：封闭图形用画笔画轮廓，用画刷填充封闭区域。不需要用到画笔的当前位置，也不会改变当前位置。

```
//矩形

BOOL Rectangle(
  HDC hdc, // handle to DC
  int nLeftRect, // x-coord of upper-left corner of rectangle
  int nTopRect, // y-coord of upper-left corner of rectangle
  int nRightRect, // x-coord of lower-right corner of rectangle
  int nBottomRect // y-coord of lower-right corner of rectangle  );
  //圆角矩形
BOOL RoundRect (
  HDC hdc, // handle to DC
  int xLeft, // x-coord of upper-left corner of rectangle
  int yTop, // y-coord of upper-left corner of rectangle
  int xRight, // x-coord of lower-right corner of rectangle
  int yBottom, // y-coord of lower-right corner of rectangle
  int xCornerEllipse, // width of ellipse
  int yCornerEllipse // height of ellipse
                              );
//（椭）圆
BOOL Ellipse (
  HDC hdc, // handle to DC
  int xLeft, // x-coord of upper-left corner of rectangle
  int yTop, // y-coord of upper-left corner of rectangle
  int xRight, // x-coord of lower-right corner of rectangle
  int yBottom // y-coord of lower-right corner of rectangle
                        );
```

dc.DrawRectangle(x,y,w,h)　//左上角，宽，高　　dc.DrawCircle(x,y,r)　　dc.DrawEllipse(x,y,w,h)

画刷

缺省画刷：WHITE_BRUSH

库存画刷：WHITE_BRUSH, LTGRAY_BRUSH，GRAY_BRUSH, DKGRAY_BRUSH, BLACK_BRUSH, NULL_BRUSH 用GetStockObject获取库存画刷句柄

自定义画刷：实心刷，阴影刷

```
//自定义实心刷
HBRUSH CreateSolidBrush(
  COLORREF crColor // brush color value  );
//自定义阴影刷
HBRUSH CreateHatchBrush(
  int iHatchStyle,        // hatch style
  COLORREF crColor   // foreground color  );
```

WX: BRUSH：wx.BLUE_BRUSH　wx.GREEN_BRUSH　wx.WHITE_BRUSH

```
wx.Brush(colour, style)
//实心刷  wx.SOLID
//阴影      wx.TRANSPARENT      wx.BDIAGONAL_HATCH    wx.FDIAGONAL_HATCH
```

文本输出

```
//单行文本输出
BOOL TextOut (
         HDC hdc, // handle to DC
         int nXStart,
             // x-coordinate of starting position
        int nYStart,                                //此坐标以左上方为原点，右为x,下为y
            // y-coordinate of starting position
        LPCTSTR lpString, // character string
        int cbString // number of characters
                              );
```

确定逻辑坐标系统(映射方式)： 坐标单位 坐标轴　原点等

```
//设置映射方式，开始时的缺省方式为MM_TEXT。此坐标以左上方为原点，右为x,下为y，单位是一个像素
//MM_LOENGLISH ：上为y,单位为0.01英寸
int SetMapMode(
       HDC hdc, // handle to device context
       int iMapMode // new mapping mode
                              );

//改变坐标原点
BOOL SetWindowOrgEx(
        HDC hdc, // handle to device context
        int X, // new x-coordinate of window origin
        int Y, // new y-coordinate of window origin
        LPPOINT lpPoint // original window origin
   );
```

WX: 坐标系统　dc. SetMapMode(mode)（wx.MM_TEXT,wx.MM_TWIPS ,wx.MM_LOMETRIC)

dc.SetAxisOrientation(xLeftRight, yBottomUp)　TRUE朝右，上
dc.SetDeviceOrigin(x,y)

```
//支持包含\r ,\n的多行输出，支持一定格式
int DrawText (
      HDC hDC, // handle to DC
      LPCTSTR lpString, // text to draw
      int nCount, // text length,  -1表示CString
      LPRECT lpRect, // formatting dimensions
      UINT uFormat // text-drawing options  ，如对齐
                          );
```

文本的设备描述表属性：

SetTextAlign	缺省TA_TOP|TA_LEFT	SetTextColor	black
SetBkMode	OPAQUE	SetBkColor	white
SetTextCharacterExtra	0		

```
COLORREF SetTextColor (
   HDC hdc, // handle to DC
   COLORREF crColor // text color
                                          );
int SetBkMode(
   HDC hdc, // handle to DC
   int iBkMode // background mode
                        );
 int SetTextCharacterExtra (
   HDC hdc, // handle to DC
   int nCharExtra // extra-space value
                                          );
 //nCharExtra可为负数
```

```
dc.DrawText(text,x,y)     //对应TextOut
dc.DrawLabel(text,rect,wx.ALIGN_**)     //对应DrawText

dc.SetBackgroundMode(wx.SOLID/TRANSPARENT)
dc.SetTextForeground(colour)
dc.SetTextBackground(colour)
```

字体

三种基本字体技术：点阵（光栅）字体　（设备相关）；矢量字体（设备无关）；TrueType写真字体（设备无关）(Apple和Microsoft开发无级缩放,实现WYSIWYG，**所见即所得**）缩写为ttf

字体属性:字样,风格,大小(常用单位为磅 — 1/72 inch)　1inch = 2.54cm

sdk中有逻辑字体和物理字体，为实现设备无关性，通过设备驱动程序将逻辑字体转为物理字体

库存字体：SYSTEM_FONT（缺省），SYSTEM_FIXED_FONT(所有字母等宽)，ANSI_FIXED_FONT

```
HFONT GetStockObject(SYSTEM_FONT);          //库存字体只用于MM_TEXT映射方式 !
```

```
typedef struct tagLOGFONT
{ LONG lfHeight;            LONG lfWidth;
  LONG lfEscapement;   LONG lfOrientation;
  LONG lfWeight;
  BYTE lfItalic;
  BYTE lfUnderline;
  BYTE lfStrikeOut;
  BYTE lfCharSet;
  BYTE lfOutPrecision;
  BYTE lfClipPrecision;
  BYTE lfQuality;
  BYTE lfPitchAndFamily;
  TCHAR lfFaceName[LF_FACESIZE]; //32个字符
 } LOGFONT, *PLOGFONT ;

/*lfHegiht与lfWidth  是用逻辑单位表示的字符高度与宽度。fHeight=0表示缺省大小(5号)  >0表示物理高度    <0表示镑值
lfHeight = -MulDiv(PointSize,
                GetDeviceCaps(hDC, LOGPIXELSY), 72);    LOGIXELSY在不同设备可能不同，一般是96
lfWidth=0表示缺省大小,与lfHeight相适应
                >0表示物理高度
                <0表示镑值
*/
通常，  DPI = 96
  GetDeviceCaps(hDC, LOGPIXELSX);
  GetDeviceCaps(hDC, LOGPIXELSY);

lfEscapement使一个字符串以一定角度书写，lfOrientation是一个字符以一定角度书写，单位为1/100,逆时针计数
lfWeight，粗细值[0，1000]，FW_NORMAL 400   FW_BOLD 700  FW_HEAVY 900  FW_THIN  100
lfCharSet:   DEFAULT_CHARSET ，ANSI_CHARSET
lfOutPrecision  输出精度  OUT_DEFAULT_PRECIS
lfClipPrecision   剪裁精度        lfQuality  输出质量  只用于点阵字体
lfPitchAndFamily  DEFAULT_PITCH  FIXED_PITCH
lfFaceName 字样名称
```

获取逻辑字体句柄

①设置LOGFONT个属性后使用CreateFont等函数建立(比较困难)

②枚举系统安装的字体，供用户选择后建立

③使用通用对话框的ChooseFont函数选择(资源那章，比如添加libcomctl32.a，在MSDN中查找CHOOSEFONTA)

```
//定义逻辑字体
HFONT CreateFont (
  int nHeight, // height of font
  int nWidth, // average character width
  ……
  DWORD fdwPitchAndFamily, // pitch and family
  LPCTSTR lpszFace // typeface name
                              );
HFONT CreateFontIndirect(
```

```
  CONST LOGFONT *lplf  //characteristics
                                          );

//枚举安装字体
int EnumFontFamiliesEx(
   HDC hdc, // handle to DC
   LPLOGFONT lpLogfont, // font information
   FONTENUMPROC lpEnumFontFamExProc,
                                          // callback function
   LPARAM lParam, // additional data , 传递给回调函数
   DWORD dwFlags // not used; must be 0
                                       );
//lpLogfont
//lfCharset                   DEFAULT_CHARSET
//lfFaceName                ""                  默认参数可以枚举出所有字体
//lfPitchAndFamily     0

//FONTENUMPROC回调函数
int CALLBACK EnumFontFamExProc(
     ENUMLOGFONTEX *lpelfe,
                                          // logical-font data
     NEWTEXTMETRICEX *lpntme,
                                       // physical-font data
    DWORD FontType,       // type of font    三种类型  DEVICE_FONTTYPE...
    LPARAM lParam // application-defined data  );

typedef struct tagENUMLOGFONTEX         //做了扩充
{  LOGFONT elfLogFont;
   TCHAR elfFullName[LF_FULLFACESIZE];
   TCHAR elfStyle[LF_FACESIZE];
   TCHAR elfScript[LF_FACESIZE];
}  ENUMLOGFONTEX,
   *LPENUMLOGFONTEX;
```

WX: FONT wx.NORMAL_FONT wx.SMALL_FONT wx.ITALIC_FONT wx.SWISS_FONT

```
//对应sdk三种方法
wx. Font(size,family, style,weight, facename=…)
Family:   wx.DEFAULT / MODERN / SCRIPT ……
Style: wx.NORMAL/SLANT/ ITALIC……
Weight: wx.NORMAL/LIGHT/BOLD……

e = wx.FontEnumerator()
fontList = e.GetFacenames()

dialog = wx.FontDialog(parent, wx.FontData())
dialog.ShowModal()
data = dialog.GetFontData()
font = data.GetChosenFont()
colour = data.GetColour()
fontName = font.GetFaceName()
fontSize = font.GetPointSize()
```

第六章

控件是一种子窗口，实现某种形式的I/O。有标准控件和公共控件。在对话框中建立（对话框由一组各种类型的控件组成），在窗口的客户区域中建立（常用预定义窗口类来建立） - 指定窗口类、控件风格、父窗口、控件ID等。

标准控件

STATIC 静态控件 BUTTON 按钮 SCROLLBAR 滚动条 LISTBOX 列表框

EDIT 编辑框 COMBOBOX 组合框(可在CreateWindow( )中使用这些窗口类名,已经定义好)

公共控件窗口类 使用头文件commctrl.h 库comctl32.dll libcomctl32.a

也可在CreateWindow( )中使用。

```
HWND hButtonWnd=CreateWindow (
   "BUTTON","OK",            类名
   BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE,     风格
   20,40,30,12,
   hWnd,              父窗口
   IDOK,          控件ID
   hInst,
   NULL );
```

控件窗口类决定了控件的基本风格以及如何对用户的输入作出响应，其对应的窗口过程函数对与其有关的消息进行处理。

控件风格：决定控件的外观和功能。 如 BS_PUSHBUTTON BS_DEFPUSHBUTTON LBS_STANDARD CBS_DROPDOWN

父窗口：**控件是子窗口，必须有父窗口。**改变父窗口将影响到控件。关闭父窗口，也会关闭其中的控件；控件可改变大小、移动，但必须限制在父窗口的客户区中。

控件ID：使用的ID需事先定义。控件在发送给父窗口的通知消息中将提供其ID。

应用程序通过消息影响控件，控件通过消息来通知应用程序

```
//应用程序→控件
SendMessage( )
SendDlgItemMessage( )
特定API ( 如ChkDlgButton( ) )
消息：
          WM_SETTEXT
          WM_SETFONT
          BM_SETCHECK
          LB_ADDSTRING
//控件→应用程序
知消息：
                    WM_COMMAND(lParam: 控件窗口句柄,HIWORD(wParam): 通知代
码,LOWORD(wParam): 控件ID)
                    WM_HSCROLL
                    WM_VSCROLL
                    WM_NOTIFY
                     ……
```

```
//BN_DOUBLECLICKED表示双击按钮。LBN_DBLCLK表示双击列表项。
//WM_NOTIFY，常用于一些公共控件，wParam: 控件ID，lParam: NMHDR*
```

WX 控件： panel = wx.Panel(frame, wx.ID_ANY)
sizer = wx.FlexGridSizer(cols, hgap, vgap)　　//把控件均匀分布
各种控件类：
　wx.StaticText, wx.TextCtrl, wx.Button.....
各类控件类事件：
　EVT_TEXT，EVT_BUTTON.....

STATIC控件：一般只用作显示文本或位图。SS_LEFT, SS_CENTER,SS_BITMAP,SS_WHITERECT,SS_BLACKFRAME。

BUTTON控件：命令按钮；单选按钮；复选按钮

风格： BS_PUSHBUTTON，BS_DEFPUSHBUTTON，在一个窗口或对话框中只能有一个命令按钮的风格为 BS_DEFPUSHBUTTON,按SPACE键相当于鼠标单击。在WM_CREATE或WM_INITDIALOG中重新设置具有输入焦点的命令按钮。

单选按钮： BS_RADIOBUTTON　BS_AUTORADIOBUTTON 在一组单选按钮中只能有一个被选中。

复选按钮： BS_CHECKBOX，BS_AUTOCHECKBOX，在一组复选按钮中能有0个、1个或多个复选按钮被选中。

```
//对按钮的操作
//使控件失效/生效：
   EnableWindow(hCtlWnd,FALSE/TRUE)
//获取按钮的选中状态，发送BM_GETCHECK消息，返回TRUE或FALSE
SendMessage(hCtlWnd,BM_GETCHECK,0,0);
//设置按钮的选中状态：发送BM_SETCHECK消息，返回TRUE或FALSE。
 SendMessage(hCtlWnd,BM_SETCHECK,TRUE,0);
//对按钮的操作的例子
翻转按钮的选中状态：
   SendMessage( hCtlWnd,
                            BM_SETCHECK,
                            (WPARAM)!SendMeaasge(hCtlWnd,

BM_GETCHECK,
                                                                    0,
                                                                    0),
                             0
                           );
//更改按钮文本：
   SetWindowText(hCtlWnd, string);
//显示/隐藏按钮：
   ShowWindow (hCtlWnd,
                            SW_SHOWNORMAL);

   ShowWindow (hCtlWnd,
                            SW_HIDE);
 //按钮通知代码
BN_CLICKED  BN_DOUBLECLICKED（BN_DBLCLK）  BN_SETFOCUS  BN_KILLFOCUS
```

WX按钮： EVT_BUTTON

自画按钮：BS_OWNERDRAW 在显示时发送WM_DRAWITEM消息，lParam指向DRAWITEMSTRUCT的指针

SCROLLBAR控件：标准滚动条(水平滚动条和垂直滚动条，窗口的一部分)与滚动条控件（可以在窗口，也可以在消息框）的区别

风格 SBS_HORI SBS_VERT

```
HWND hScrollbar=CreateWindow(
    "SCROLLBAR", "",
    WS_CHILD|SBS_VERT|WS_VISIBLE,
    10,30-GetSystemMetrics(SM_CYVSCROLL),
    19,100+2*GetSystemMetrics(SM_CYVSCROLL),
    hWnd,NULL,hInst,NULL   );
```

与标准滚动条的使用类似(设置滚动条范围，位置，接收滚动条消息等。)使用SetScrollInfo( )时，fnBar应为SB_CTL。hWnd应为控件句柄。

```
int SetScrollInfo(
   HWND hwnd, // handle to window
   int fnBar, // scroll bar type
   LPCSCROLLINFO lpsi, // scroll parameters
   BOOL fRedraw // redraw flag
                        );
//滚动条控件的通知消息
WM_HSCROLL    WM_VSCROLL    lParam为控件句柄
```

窗口子类化 让滚动条可以用键盘控制

```
//获得注册窗口类的属性
LONG GetWindowLong (
   HWND hWnd, // handle to window
   int nIndex // offset of value to retrieve
                                    );
LONG GetWindowLong(hCtlWnd,GWL_HINSTANCE);
hWnd→HINSTANCE

LONG GetWindowLong(hCtlWnd,GWL_ID);
hWnd→ID

LONG GetWindowLong(hCtlWnd,GWL_WNDPROC);
hWnd→WNDPROC
```

扩展窗口的功能，把发送给窗口的消息先通过自己定义的窗口过程函数获取，实现一部分扩展的功能，再通过原窗口过程函数实现大部分原来的功能。

```
①获取控件的窗口函数
LONG GetWindowLong(hCtlWnd,GWL_WNDPROC);
②设置控件的窗口函数
LONG SetWindowLong(hCtlWnd,GWL_WNDPROC,
```

```
                                    (LONG)MyWndProc);
③调用控件的原窗口函数
LRESULT CallWindowProc(
  WNDPROC lpPrevWndFunc,
                           // pointer to previous procedure
  HWND hWnd, // handle to window
  UINT Msg,      // message
  WPARAM wParam, // first message parameter
  LPARAM lParam // second message parameter
                                          );
```

详见colors.c

WX滚动控件： wx.Slider　EVT_SCROLL_CHANGED

LISTBOX控件

列表框类型：单选；多选；非选（只用于显示）

列表框风格： LBS_STANDARD (LBS_NOTIFY|LBS_SORT| WS_VSCROLL|WS_BORDER | LBS_SORT)

发送给列表框的消息：LB_ADDSTRING　，0　，字符串(添加新的列表选项)　SendMessage

LB_INSERTSTRING　，位置，字符串　　LB_RESETCONTENT，0　，0

LB_FINDSTRING　，开始位置，字符串

LB_SELECTSTRING，开始位置，字符串(开始位置为-1时表示0)
-- LB_SETCURSEL，位置，0(-1全部取消)

LB_GETCURSEL，0，0
-- LB_SELITEMRANGE，TRUE/FLASE，(结束位置，开始位置)
-- LB_GETSELCOUNT，0，0
-- LB_GETSELITEMS，最大项数，位置数组
-- LB_GETTEXT，位置，字符串
-- LB_GETTEXTLEN，位置，0
-- LB_GETSEL，位置，0
-- LB_SETSEL，TRUE/FALSE，位置

LB_SETITEMDATA，位置，数据
-- LB_GETITEMDATA，位置，0
-- LB_DIR, 文件属性，文件（夹）名

向列表框发送消息后的返回　　LB_OKAY　LB_ERR　LB_ERRSPACE(内存处理错误)

列表框通知代码　LBN_SELCHANGE　LBN_DBLCLK　LBN_SELCANCEL　　switch

WX ListBox　wx.ListBox　EVT_LISTBOX　EVT_LISTBOX_DCLICK

EDIT控件：编辑框控件，用于编辑文本，编辑框拥有输入焦点时，显示插入符。

```
RECT rc;
GetClientRect(hWnd,(LPRECT)&rc);
HWND hWndEdit=CreateWindow (
  "EDIT",NULL,
  WS_CHILD|WS_VISIBLE|WS_BORDER|WS_HSCROLL|
  WS_VSCROLL|ES_LEFT|ES_MULTILINE|
  ES_AUTOHSCROLL|ES_AUTOVSCROLL,
  0,0, rc.right - rc.left, rc.bottom - rc.top,
  hWnd,
  IDC_EDIT,
  hInst,
  NULL           );

//与客户区大小一致
case WM_CREATE:
 HWND hWndEdit =CreateWindow (
  "EDIT",NULL,
  WS_CHILD|WS_VISIBLE|WS_BORDER|WS_HSCROLL|
  WS_VSCROLL|ES_LEFT|ES_MULTILINE|
  ES_AUTOHSCROLL|ES_AUTOVSCROLL,
  0,0,0,0,
  hWnd,
  IDC_EDIT,
  hInst,
  NULL                      );
case WM_SIZE:
  MoveWindow(hWndEdit,0,0,
             LOWORD(lParam),HIWORD(lParam),TRUE);
//风格ES_MULTILINE  ES_NUMBER  ES_AUTOHSCROLL ES_AUTOVSCROLL ES_LEFT ES_RIGHT
ES_CENTER ES_PASSWORD

//编辑框操作
WM_GETTEXT,0,buffer      WM_SETTEXT,length,buffer   WM_SETFONT,hFont,0
 SetDlgItemText
 GetDlgItemText
 SetDlgItemInt
 GetDlgItemInt

UINT GetDlgItemText (
  HWND hDlg, // handle to dialog box
  int nIDDlgItem, // control identifier
  LPTSTR lpString, // pointer to buffer for text
  int nMaxCount // maximum size of string
  );

 BOOL SetDlgItemText (
  HWND hDlg, // handle to dialog box
  int nIDDlgItem, // control identifier
  LPCTSTR lpString // text to set );
  );

//编辑框操作
EM_SETPASSWORDCHAR,char,0
例如:
  SendMessage(GetDlgItem(hDlg,IDC_EDIT),
                    EM_SETPASSWORDCHAR,'?',0 );
EM_SETSEL,startpos,endpos
```

```
   startpos=0,endpos=-1表示Select All
EM_GETSEL,startpos,endpos+1
EM_REPLACESEL,canundo,replacetext
```

编辑框通知代码：EN_UPDATE 显示之前 EN_CHANGE 显示之后 EN_HSCROLL EN_VSCROLL EN_MAXTEXT EN_ERRSPACE EN_SETFOCUS EN_KILLFOCUS

WX TextCtrl wx.TextCtrl EVT_TEXT

COMBOBOX控件：组合框 – 列表框和编辑框的组合

组合框分类：
简单组合框 CBS_SIMPLE（列表不是下拉的）
下拉式组合框 CBS_DROPDOWN
下拉式列表框 CBS_DROPDOWNLIST(不能编辑)

发送给组合框的消息：两大类：分别对应列表框和编辑框

CB_ADDSTRING ，0 ，字符串
-- CB_INSERTSTRING ，位置，字符串
-- CB_RESETCONTENT ，0 ， 0
-- CB_FINDSTRING ，开始位置，字符串
-- CB_SELECTSTRING ，开始位置，字符串(开始位置为-1时表示0)
-- CB_SETCURSEL，位置，0

CB_GETCURSEL，0，0
-- CB_GETLBTEXT，位置，字符串
-- CB_GETLBTEXTLEN，位置，0

组合框通知代码： CBN_SELCHANGE
CBN_DBLCLK
CBN_EDITUPDATE
CBN_EDITCHANGE
CBN_DROPDOWN
CBN_CLOSEUP
……

```
HWND ChildWindowFromPoint(
    HWND hWndParent,
             // handle to parent window
    POINT Point
              // structure with point coordinates
                                                  );
hWndEditTextString =
 ChildWindowFromPoint(
        hWndComboBoxTextString,point );
//从COMBOBOX中得到EDIT框的句柄
```

wx.ComboBox EVT_COMBOBOX, EVT_TEXT

Statusbar公共控件（需要特殊的头文件和链接库）:建立状态栏窗口，也可使用CreateStatusWindow建立

```
hWndStatusBar =CreateWindow(STATUSCLASSNAME, "",
    WS_CHILD|WS_VISIBLE|SBARS_SIZEGRIP,
    0,0,0,0,
    hWnd,
    (HMENU)ID_STATUSBAR,
     hInst,
    NULL);

hWndStatusBar    =CreateStatusWindow(
  WS_CHILD|WS_VISIBLE|
  SBARS_SIZEGRIP,
  "",
  hWnd,
  ID_STATUSBAR);
```

分区最多255个,SB_SETPARTS,分区数，分区位置数组,位置-1表示至客户区右端 ,SB_GETPARTS,分区数，分区位置数组

SB_SETTEXT,分区号，要显示的信息

```
HMODULE LoadLibrary (
  LPCTSTR lpFileName
                // file name of module
                        );

BOOL FreeLibrary(
   HMODULE hModule
            // handle to DLL module
                    );
```

第七章

动态链接库：DLL与应用程序不同，它不直接执行，也不接收与处理消息。DLL是一些独立的文件，其中包括能被其它应用和其它DLL调用的完成一定任务的函数。还存在由资源组成的DLL(如字库), 也称为纯资源DLL库。

SLL存在效率较低的问题，因为同一函数在不同应用中分别有自己的副本，占用内存较多。

Windows系统API以DLL的形式存在，**在内存中每个API函数只存在一个副本，在不同的应用之间共享。**

DLL库的常用后缀为.DLL，也可为 .EXE，.DRV， .TTF等。Linux DLL库的常用后缀为.so。

后缀为.DLL的库可由系统自动加载，其它库需使用LoadLibrary函数加载。

```
HMODULE LoadLibrary("库名")
  BOOL FreeLibrary(HMODULE)
```

输入库：输入库(import library) - 后缀为.a的目标代码库，它不包含执行代码，而是**包含linker用于确定应用中调用的DLL函数的位置。**通常，每一个动态链接库都有一个相应的输入库供程序员使用。

查找DLL顺序：应用所在的目录 当前目录 Windows目录(如: /windows) Windows系统目录(如: /windows/system32)

PATH设置的其它目录

DLL的建立：工程类型为 : Dynamic Link Library，**需要建立包含DLL函数说明的头文件，建立包含DllMain( )函数和各个DLL函数定义的源程序文件**，编译、连接DLL工程，**生成.dll, 输入库.a及.def**

```
//EDRLIB.H
#ifdef __cplusplus
#define EXPORT extern "C" __declspec(dllexport)  C，C++按照相同方式变换
#else
#define EXPORT __declspec(dllexport)
#endif

EXPORT dll函数的原型;
……
EXPORT BOOL CALLBACK CenterTextOutA(HDC, PRECT, PCSTR);
EXPORT BOOL CALLBACK CenterTextOutW(HDC, PRECT, PCWSTR);
#ifdef UNICODE
  #define CenterTextOut CenterTextOutW
#else
  #define CenterTextOut CenterTextOutA
#endif
```

declspec (dllexport)：将对应的函数导出，即：把函数名登记在输入库中。extern "C" declspec (dllexport) 防止C++编译时发生Name Mangling(函数名分割)。

```
EDRLIB.C
#include <windows.h>
#include "ErdLib.h"
EXPORT BOOL CALLBACK CenterTextOutA
(HDC hdc, PRECT prc, PCSTR pString)
{    int iLen=lstrlenA(pstring);
     SIZE size ;
     GetTextExtentPoint32A(hdc, pString,
                 ilen, &size) ;
     return TextOutA(hdc,
     (prc->right-prc->left-size.cx)/2,
     (prc->bottom-prc->top-size.cy)/2,
     pString, iLen) ;
}
EXPORT BOOL CALLBACK CenterTextOutW
(HDC hdc, PRECT prc, PCWSTR pString)
{    int iLen=lstrlenW(pstring);
     SIZE size ;
     GetTextExtentPoint32W(hdc, pString,
                 ilen, &size) ;
     return TextOutW(hdc,
     (prc->right-prc->left-size.cx)/2,
```

```
     (prc->bottom-prc->top-size.cy)/2,
     pString, iLen) ;
}
```

DLL使用示例程序：在测试程序的工程中要加入DLL输入库，在测试程序中包含DLL头文件，在适当的目录中放置DLL库。

多个应用可共享使用同一个DLL函数，但应用之间不会相互影响。每个进程共享相同的DLL函数代码，但为每一个进程保存的数据都不同。每个进程都为DLL使用的全部数据分配了自己的地址空间。

so的建立： soSample.h　soSample.c

gcc –c soSample.c(编译)　　gcc –shared –fPIC –o libsoSample.so soSample.o(生成libsoSample.so)

PIC: Position Independent Code（代码格式，可以让多个文件共享）

使用libsoSample.so: gcc -o myapp myapp.c -L. –lsoSample

so的位置：mv　libsoSample.so　/usr/lib，或　export LD_LIBRARY_PATH=...

纯资源库：在dll工程中加入rc文件和包含DllMain的C程序文件，生成资源DLL库。

```
#include <windows.h>

BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
(没有这个函数也行)
{
     return TRUE ;
}
//纯资源库的使用
HMODULE hDll = LoadLibrary("BitmapLib.dll");

  LoadBitmap(hDll,MAKEINTRESOURCE(ID));

  FreeLibrary(hDll);
```

def 和 a 文件都是空的

WX: 调用WINDLL：使用 module： ctypes(连接C和python的桥梁)

```
dll=ctypes.WinDLL(DLL库名)        //CDLL
dll.function(……)       //类型c_int, ARRAY等
```

需要指定连接方式，WINAPI 和_cdecl

sdk中建立链接库是32位，与64位python不兼容，一种解决方式是使用32位的python，一种是建立64位的动态链接库(17.12 cb的编译器不支持64位编译链接)，需要下载64位的MinGW, 在cb工具链中改变成自己下载的。

WX: 调用Linux SO

用 module： ctypes

ctypes.cdll.LoadLibrary("libc.so.6")
或:
libc = ctypes.CDLL("libc.so.6")

```
pyIntx=100
ctInta=c_int(pyIntx)
pyInty=ctInta.value

pyStrx="Hello, World"
ctWcpa=c_wchar_p(pyStrx)
pyStry=ctWcpa.value

pyStrx="Hello, World"
ctCpa=c_char_p(bytes(pyStrx,encoding="utf-8"))
pyStry=str(ctCpa.value,encoding="utf-8")

ArrayType=c_int*100
a=ArrayType()
L=[4,7,8,9]
b=ArrayType(*L)
c=ARRAY(c_int,100)(*L)

for  in range(len(L)): L[i]=c[i]

p1 = create_string_buffer(10)
               # create a 10 byte buffer
   p1.value
   p1.raw           //列出每一个元素的值
   p1.value=b"Hello"
   p2 = create_string_buffer(b"Hello")

   p3 = create_string_buffer(b"Hello",10)

 //指针:
  a=c_int(10)
          p=pointer(a)
          p.contents
          p[0]=100  // a □ 100

PI = POINTER(c_int)
          nullPtr=PI()

指定函数参数:
   Ex:
        func.argtypes = [c_char_p, c_char_p, c_int, c_double]
//函数指针：用python使用qsort
   Ex:
        cdll=CDLL("msvcrt.dll ")
              IntArray5 = c_int * 5  ;     ia = IntArray5(5, 1, 7, 99, 33)
              qsort = cdll.qsort ;   qsort.restype = None
```

```
CMPFUNC = CFUNCTYPE(c_int, POINTER(c_int), POINTER(c_int))
def py_cmp_func(a, b):  return a.contents.value - b.contents.value
cmp_func = CMPFUNC(py_cmp_func)
qsort(ia, len(ia), sizeof(c_int), cmp_func)
```

第八章

Python 模块非常丰富，与GUI应用程序的融合很重要。

numpy 数值计算基础模块，是其它很多科学工程相关模块的基础

```
numpy 基础类型array
Ex: a = array([1,2])
          b= array([[1,2],[3,4]])
                 b[i,j]，b[i][j]指定元素
          c = array([[1,2],[3,4]],dtype= 'd')
                 c.dtype获取元素类型
```

a[x:y:s]获取部分数组
a.shape是结构信息
a.reshape重构结构
a.flatten重构为一维结构
a.transpose矩阵转置

矢量化计算
Ex1: b*5*
*Ex2: def f(x): return x***2+3*x+5
        f(b)

函数矢量化
    def f(x): return x**2+3*x+5
    g=vectorize(f)
    g(b)

矩阵类型 mat
  Ex: A=mat(array([[0,1,2],[1,0,3],[4,-3,8]]))

矩阵运算 linalg (linear algebra)
Ex1: inverse = linalg.inv(A)
       检查 A * inverse 是单位矩阵
Ex2: A=mat([[1,-2,1],[0,2,-8],[-4,5,9]])
        b=array([0,8,-9])
        x = linalg.solve(A, b)
        检查 dot(A , x) 等于 b

矩阵运算

另一种表示法求逆矩阵 A**-1

随机数
Ex1: N=100
        Normal_values = random.normal(size=N)
Ex2: N=100
        Log_values = random.lognormal(size=N)

随机数衍生
Ex: N=10
    random.randint(100, size=N)

其它函数:

Ex: linspace
    x = linspace(0, 10, 100)

PIL: Python image library

```
from PIL import Image
im = Image.open("lena.ppm")
print(im.format, im.size, im.mode)
im.show()
im.save("lena.jpg", "JPEG")
box = (50, 50, 100, 100); region = im.crop(box)
region = region.transpose(Image.ROTATE_180)
im.paste(region, box)
```

sympy：符号定义：x,y,a=symbols("x y a")
方程定义：eq=Eq(x**2-a+1,0)
解方程：res= solve(eq,x)
str=latex(res)
str是Latex语言描述字符串

Latex: 学公式描述语言
https://www.latex-project.org/
str=latex(res)
""%str 在matplotlib中显示

公式例子:
    Rational(3,2)*pi* + *exp(*x) / (x**2 + y)**
公式展开:
    f = ((x+y)2 * (x+1))
    expand(f)
公式化简:
    f = 1/x + (x*sin(x) - 1)/x
    simplify(f)

f = x**2-y**2
因式分解：res=f.factor()

f = limit((sin(x)-x)/x**3, x, 0)
极限计算：res=f

f=integrate(x**2 * cos(x), (x, 0, pi/2))
  定积分计算：res=f

matplotlib：x=linspace(0,5,11)
y=x**3

fig,axes=pyplot.subplots()
axes.plot(x,y, 'r')
fig.show()

axes.plot(x,y,'r--')
axes.plot(x,y, 'g*-',lw=3.0)

axes.axis("on")
axes.set_xlabel("...")
axes.set_ylabel("...")
axes.set_title("...")

fig.savefig("filename")

axes.plot(x, x**2, label="y = x**2")
axes.plot(x, x**3, label="y = x**3")
axes.legend(loc=2) # upper left corner

axes.plot(x, x**2, label=r"$y = \alpha^2$")
axes.plot(x, x3, label="y = x**3")
axes.legend(loc=2) # upper left corner

axes.scatter(x, x*2)
*axes.scatter(x,0.25*numpy.random.normal(size=len(x)))

x=random.randint(10,size=1000)
axes.hist(x)

x=random.randint(10,size=1000)
y=random.randint(10,size=1000)
axes.hist(x+y)
axes.hist(x+y,bins=1000)