Day01-基础框架介绍

#include <windows.h>

*HINSTANCE* **g\_hInstance**;//全局程序实例句柄

//窗口程序的流程如下——这是系统自动完成的/也可自己纯手工：

//第二步：注册窗口类型

*BOOL* MyRegister(*LPSTR* **pszClassName**)

{

*ATOM* **nAtom**;

*WNDCLASS* **wndClass**{ 0 };//定义一个窗口类型的窗口对象

**wndClass**.*style* = *CS\_VREDRAW* | *CS\_HREDRAW*;//窗口风格，这个参数可以修改（自行查资料）

**wndClass**.*lpfnWndProc* = WndProc;//窗口的消息处理函数（前面定义了某个消息的处理函数，这里把函数名

// 传递给某个窗口，当在这个窗口按下了对应的按钮，就会执行该函数）

**wndClass**.*cbClsExtra* = 0;//窗口其他参数

**wndClass**.*cbWndExtra* = 0;//窗口其他参数

**wndClass**.*hInstance* = **g\_hInstance**;//窗口程序的句柄

**wndClass**.*hIcon* = *NULL*;//窗口默认图标没有

**wndClass**.*hCursor* = *NULL*;//窗口默认光标没有

**wndClass**.*hbrBackground* = (*HBRUSH*)(*COLOR\_BTNFACE*);//窗口默认背景颜色

**wndClass**.*lpszMenuName* = *NULL*;//窗口默认菜单名--使用系统的

**wndClass**.*lpszClassName* = **pszClassName**;//窗口注册的具体类型名

//到此为止就声明了一个窗口的基本元素，接下来是注册这个类型的窗口

**nAtom** = *RegisterClass*(&**wndClass**);

if(!**nAtom**)//如果注册失败

{

*MessageBox*(*NULL*, "抱歉，注册窗口失败", "温馨提示", *MB\_OK*);

return *FALSE*;

}

else *MessageBox*(*NULL*, "恭喜，注册窗口成功", "温馨提示", *MB\_OK*);

return *TRUE*;

}

//第三步：创建一个窗口

*HWND* MyCreate(*LPSTR* **pasClassName**) {

*HWND* **hWnd** = *CreateWindow*( **pasClassName**, "最简单的Win32窗口",

*WS\_OVERLAPPEDWINDOW*, 100, 100,

300, 500, *NULL*, *NULL*,**g\_hInstance**, *NULL*);

if (!**hWnd**) {

*MessageBox*(*NULL*, "抱歉，创建窗口失败", "温馨提示", *MB\_OK*);

return *NULL*;

}

else *MessageBox*(*NULL*, "恭喜，创建窗口成功", "温馨提示", *MB\_OK*);

return **hWnd**;//重点：创建成功后必要把该窗口就返回出来

}

//第四步：显示窗口

void ShowWnd(*HWND* **hWnd**)

{

*ShowWindow*(**hWnd**,*SW\_SHOW*);

*UpdateWindow*(**hWnd**);

}

//第五步：从窗口中获得消息

void Message()

{

*MSG* **msg** = { 0 };//MSG是一个消息结构体

while (*GetMessage*(&**msg**,*NULL*,0,0))//从消息队列中获得消息

*DispatchMessage*(&**msg**);//分发消息到对应的消息回调函数

}

//第六步：窗口各类事件消息的处理函数（有什么消息都需要自己手工完成--这个是回调函数），

// 由系统来调用此函数

*LRESULT* *CALLBACK* WndProc( *HWND* **hWnd**,//窗口的句柄--代表某一个窗口

*UINT* **nMsg**,//消息的ID

*WPARAM* **wParam**,//消息对应的参数

*LPARAM* **lParam**)//消息对应的参数

{

switch (**nMsg**)

{

case *WM\_DESTROY*://如果在窗口点击了关闭类按钮----窗口销毁的消息

*PostQuitMessage*(0);//发送退出消息

return 0;

break;

}

//调用系统默认消息处理函数

return *DefWindowProc*(**hWnd**, **nMsg**, **wParam**, **lParam**);//如果点击了其他按钮--则把形参传入这个默认消息处理函数中

}

//第一步：整个程序的入口函数

int \_stdcall *WinMain*( *\_In\_* *HINSTANCE* **hInstance**,

*\_In\_opt\_* *HINSTANCE* **hPrevInstance**,

*\_In\_* *LPSTR* **lpCmdLine**,

*\_In\_* int **nShowCmd** )

{

*HWND* **hwnd** = *NULL*;//窗口变量

*BOOL* **bRet** = *FALSE*;

*CHAR* **tempLPSTR**[] = "最简单的Win32窗口";

**bRet** = MyRegister(**tempLPSTR**);

if (!**bRet**) {

*MessageBox*(*NULL*, "注册失败", "温馨提示", *MB\_OK*);

return -1;//失败返回-1

}

else *MessageBox*(*NULL*, "注册成功", "温馨提示", *MB\_OK*);

**g\_hInstance** = **hInstance**;

**hwnd** = MyCreate(**tempLPSTR**);

if (!**hwnd**) {

*MessageBox*(*NULL*, "创建窗口失败", "温馨提示", *MB\_OK*);

}

ShowWnd(**hwnd**);//成功--直接显示出来

Message();//开始获取消息

return 0;

}

【总结】

第一：WinMain()入口🡪注册窗口类型(即窗口的各类初始参数)🡪创建窗口、显示窗口、消息捕获分发🡪WndProc()窗口处理回调🡪程序退出

第二：注册窗口类时，可以加载应用图标🡪 **wndClass**.*hIcon* = *NULL*;//窗口默认图标没有，

**wndClass**.*hIcon* = *LoadIcon(g\_hInstance,MAKEINTRESOURCE(对应的图标宏定义));*

第三：利用VS自动生成窗口时，基本逻辑跟前面一样，只是封装了显示窗口函数、增加了关于消息的回调函数。

Day02-makefile与Win32字符编码介绍

一 NMAKE和Makefile

1.1 NMAKE - 命令解释器，根据Makefile文件中定义的脚本，完成项目的编译等操作。

1.2 Makefile - 定义编译/链接等脚本语言。

1.3 Makefile文件的编写

1.3.1 基本语法规则

window.exe:window.obj // 依赖行 cl.exe window.c /c // 命令行

link.exe window.obj user32.lib

window.exe的依赖项是window.obj，如果window.obj被重新改写,window.exe将重新生成。通过时间戳(time stamp)判断程序是否需要重新编译链接，如果当文件修改最后时间与时间戳不同，将会重新编译链接。

1.3.2 执行过程

1 NMAKE首先找到第一个依赖行，根据依赖行之间的关系，建立依赖树。例如：

A:B

B:C

C:D

NMAKE会建立对应的依赖树

A

|-B

|-C

|-D

2 在树建好后，NMAKE会首先执行D的命令行，然后依次执行父结点的命令行

3 在A的命令执行结束后,退出NMAKE.

4 如果需要执行指定依赖行,需要在执行NMAKE时增加依赖行的名称NMAKE /f Makefile.mak B <--指定从B执行

1.4 使用

1.4.1 NAMKE指定文件名----->NMAKE /f Makefile.mak

1.4.2 使用缺省文件名Makefile----->NMAKE在执行时会自动查找这个文件.

二 字符编码

2.1 编码的历史

2.1.1 ASCII 0-127 共7位二进制位表示

2.1.2 ASCII扩展码 0-255 8位表示；代码页:通过代码页来切换对应的字符

2.1.3 双字节字符集 DBCS

使用一个或两个字节表示字符.

"A中B国"对应字节数为：1 2 1 2 =====>A: 0x41 中:0x8051 B: 0x42 国:0x8253

2.1.4 Unicode

全部使用2个字节表示字符==>"A 中 B 国"对应字节数为：2 2 2 2

A: 0x0041 中:0x8051 B: 0x0042 国:0x8253

2.2 C语言和编码

2.2.1 单字节的字符和字符串

char cText = 'A'; char \* pszText = "ABCD";

2.2.2 宽字节的字符

wchar\_t cText = 'A' wchar\_t \* pszText = L"ABCD";

2.2.3 TCHAR

为了程序中可以方便的支持的Unicode和多字节字符等,所以使用TCHAR来定义字符和字符串.根据\_UNICODE宏开关,会将TCHAR编译成不同字符类型.

#ifndef \_UNICODE

typedef char TCHAR

#define \_\_T(x) x

#else

typedef wchar\_t TCHAR

#define \_\_T(x) L##x

#endif

使用时,要增加TCHAR.H头文件支持,使用\_UNICODE 宏开关进行编译CL window.c /D\_UNICODE

#define \_UNICODE和#include "tchar.h"

定义方式:

TCAHR \* pszText = \_\_T("ABCDEF"); 代码使用：使用UNICODE宏开关，通知编译器选择编译的代码。

#ifndef \_UNICODE

int nLen = strlen( pszText );

#else

int nLen = wcslen( pszText );

#endif

2.3 Win32程序与编码

2.3.1 Win32 API的定义--->每个API对多字节字符和UNICODE分别有不同的版本。

　 MessageBox MessageBoxA 多字节字符 MessageBoxW UNICODE字符

2.3.2 字符的定义，使用TEXT, 由Winnt.h提供定义

#ifdef UNICODE

#define \_\_TEXT(quote) L##quote

#else /\* UNICODE \*/

#define \_\_TEXT(quote) quote

#endif /\* UNICODE \*/

TCHAR \* pszText = TEXT( "ABCD" );

　　2.3.3 字符转换函数

int WideCharToMultiByte(

UINT CodePage, //代码页

DWORD dwFlags, //转换方式

LPCWSTR lpWideCharStr, //需要被转换WCHAR地址

int cchWideChar, //需要被转换WCHAR的长度

LPSTR lpMultiByteStr,//用于存放转换后的结果BUFF

int cchMultiByte, //BUFF的长度

LPCSTR lpDefaultChar,//使用的缺省字符串的地址

LPBOOL lpUsedDefaultChar //缺省字符串被使用的标识

);

int MultiByteToWideChar(

UINT CodePage,// 代码页

DWORD dwFlags,// 转换方式

LPCSTR lpMultiByteStr, // 需要被转换CHAR地址

int cchMultiByte,//需要被转换CHAR的长度

LPWSTR lpWideCharStr,//用于存放转换后的结果BUFF

int cchWideChar );//BUFF的长度

使用方法：

1 将要转换的字符串，传递给函数，从返回值中获取转换后字符串的长度。

2 分配字符串空间

3 再次调用函数，并将分配的空间传递给函数，获取结果.

Day03-Win32窗口的创建过程详解

一 窗口注册和创建

1 Win32窗口程序创建步骤：1.1 WinMain入口函数的定义 1.2 WindowProc函数的定义(声明)

1.3 注册窗口类RegisterClass/RegisterClassEx 1.4 创建窗口CreateWindow/CreateWindowEx

1.5 窗口的显示和刷新ShowWindow/UpdateWindow 1.6 消息处理GetMessage/DispatchMessage

1.7 窗口退出WM\_DESTROY/PostQuitMessage

2 窗口的注册

2.1 窗口类的分类

2.1.1 系统全局的窗口类.比如按钮("BUTTON")、文本编辑框("EDIT")等。

2.1.2 应用程序的全局窗口类。可以在一个应用程序中EXE、DLL等所有模块中使用的窗口类。

2.1.3 局部窗口类。只能在本模块中使用的窗口类。

2.2 实现窗口类的注册

2.2.1 系统全局的窗口类，无需注册直接使用。使用CreateWindow函数，在CLASSNAME中指定系统已经定义好的窗口类型名称即可。

2.2.2 应用程序的全局窗口类,需要用代码实现注册，在注册时需要增加CS\_GLOBALCLASS定义。

实现方式：

WNDCLASS wc = {0};

wc.style = CS\_GLOBALCLASS|CS\_HREDRAW..;

....

RegisterClass( &wc );

2.2.3 局部窗口类，不是增加CS\_GLOBALCLASS定义。使用RegisterClass和RegisterClassEx注册

typedef struct \_WNDCLASSEX {

UINT cbSize; //结构体的大小

UINT style;

WNDPROC lpfnWndProc;

int cbClsExtra;

int cbWndExtra;

HANDLE hInstance;

HICON hIcon;

HCURSOR hCursor;

HBRUSH hbrBackground;

LPCTSTR lpszMenuName;

LPCTSTR lpszClassName;

HICON hIconSm; //小图标

} WNDCLASSEX;

2.2.4 窗口类风格

CS\_HREDRAW 窗口水平变化，重新绘制窗口。 CS\_VREDRAW 窗口垂直变化，重新绘制窗口。

CS\_DBCLICK 窗口可以接收鼠标双击消息 CS\_GLOBALCLASS 创建应用程序全局窗口类。

CS\_BYTEALIGNWINDOW 窗口对齐方式,以8的倍数对齐 CS\_BYTEALIGNCLIENT 窗口客户区对齐方式,以8的倍数对齐

CS\_CLASSDC 所有这种类型的窗口使用同一个DC(设备描述表,绘图使用) CS\_OWNDC 每个窗口拥有自己的DC

CS\_PARENTDC 使用父窗口的DC CS\_SAVEBITS 是用位图保存窗口界面,可以提高窗口界面的刷新性能

CS\_NOCLOSE 禁止关闭命令.

2.2.5 窗口类的附加数据 cbClsExtra，在窗口类的数据信息中添加自己信息：

cbClsExtra 用于添加信息的内存的大小

SetClassLong 将信息保存到内存中

GetClassLong 将信息从内存中取出

DWORD SetClassLong(HWND hWnd,//窗口的句柄int nIndex, //值的索引号LONG dwNewLong //值);

DWORD GetClassLong(HWND hWnd,//窗口的句柄int nIndex, //值的索引号);其中cbClsExtra的长度一般是4字节的倍数

2.2.6 窗口附加数据 cbWndExtra，在窗口的数据信息中添加自己信息：

cbWndExtra 用于添加信息的内存的大小

SetWindowLong 将信息保存到内存中

GetWindowLong 将信息从内存中取出

2.3 窗口类的相关函数

RegisterClass/RegisterClassEx 注册 UnregisterClass 注销

GetClassInfo/GetClassInfoEx 获取信息 GetClassName 获取窗口的窗口类名称

GetClassLong/SetClassLong 窗口类附加数据的设置和获取

GetWindowLong/SetWindowLong 窗口的附加数据的设置和获取

2.4 创建窗口时如何定位窗口类

2.4.1 查找应用程序局部的窗口类。如果找到，执行2.4.2，否则执行2.4.3

2.4.2 如果找到名称一致的窗口类，会比较HISNTANCE句柄。如果相等，就使用找到的窗口类信息创建窗口，如果不同，继续查找，执行2.4.3。

2.4.3 如果未找到名称一致的窗口类，会继续在应用程序全局窗口类中查找。如果找到，执行2.4.4，否则执行2.4.5

2.4.4 全局窗口类找到一致的，使用找到的窗口类信息创建窗口。

2.4.5 系统全局窗口类中查找。如果找到，创建窗口，否则返回失败。

3 窗口创建

3.1 窗口创建函数

CreateWindow/CreateWindowEx

HWND CreateWindowEx(

DWORD dwExStyle,//窗口的扩展样式

LPCTSTR lpClassName, // pointer to registered class name

LPCTSTR lpWindowName, // pointer to 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, // handle to menu, or child-window identifier

HINSTANCE hInstance, // handle to application instance

LPVOID lpParam ); // pointer to window-creation data

3.2 窗口风格和扩展风格

窗口风格: WS\_XXXX定义的风格,是窗口的基本风格。

扩展风格: WS\_EX\_XXXXX 定义的风格，是窗口的扩展风格。比如: ToolWindow窗口等等。在CreateWindow可以使用基本窗口风格，扩展的窗口风格,需要使用CreateWindowEx设置。

WS\_OVERLAPPED窗口,层叠式窗口

WS\_POPUP窗口,弹出式窗口

WS\_CHILD窗口,子窗口

3.3 父窗口和子窗口

3.3.1 CreateWindow时,指定父窗口

3.3.2 将窗口的风格增加WS\_CHILD

3.3.3 可以使用SetParent和GetParent函数设置和获取指定窗口的父窗口。其他: MoveWindow 移动窗口。

3.4 MDI窗口的创建

3.4.1 创建主窗口

注册主窗口类型,创建主窗口.

HWND hWnd = CreateWindowEx( 0,

pszClassName, "MainWnd",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, NULL, NULL, g\_hInst,

NULL );

3.4.2 MDICLIENT窗口

1 添加CLIENTCREATESTRUCT结构,作为附加数据

CLIENTCREATESTRUCT cs = { 0 };

cs.idFirstChild = 1000;//MDI第一个子窗口的ID

2 创建时,使用"MDICLIENT"窗口类型.

3 它的父窗口是主窗口

HWND hWnd = CreateWindowEx( 0,

"MDICLIENT", "MainWnd",

WS\_CHILD|WS\_VISIBLE, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, hParent, NULL, g\_hInst,

&cs );

3.4.3 创建一个MDI子窗口

1 注册主窗口类型,创建子窗口

2 它的父窗口为MDICLIENT窗口

3 增加WS\_EX\_MDICHILD的风格

HWND hWnd =

CreateWindowEx( WS\_EX\_MDICHILD,

pszClassName, "ChildWnd",

WS\_CHILD|WS\_VISIBLE, CW\_USEDEFAULT,

CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, hParent, NULL,

g\_hInst, NULL );