2020/06/22_SDK_第2课_创建一个窗口

笔记本: SDK编程
创建时间: 2020/6/22 星期一 15:24
作者: ileemi
标签: SDK 创建一个窗口, 创建窗口的六要素

创建一个窗口

创建窗口的六要素

- 设计注册窗口类
- 创建窗口实例
- 显示窗口
- 更新窗口
- 建立消息循环
- 实现窗口过程函数

设计注册窗口类

窗口类

在Windows上窗口是分类的，基本要素相似的窗口属于同一个窗口类。窗口的标题被称为窗口实例。

API函数： RegisterClass

函数作用：RegisterClass 函数注册了一个窗口类，以便在调用CreateWindow或CreateWindowEx函数时使用。

查看MSDN时，对应的函数介绍由Note时，代表该函数由升级的版本，如下图所示：

RegisterClass

The **RegisterClass** function registers a window class for subsequent use in calls to the **CreateWindow** or **CreateWindowEx** function.

Note The **RegisterClass** function has been superseded by the **RegisterClassEx** function. You can still use **RegisterClass**, however, if you do not need to set the class small icon.

```
ATOM RegisterClass(
  CONST WNDCLASS *lpWndClass  // class data
);
```

RegisterClass 函数有一个参数：lpWndClass，其指向 WNDCLASS 结构的指针。

WNDCLASS 结构体类（窗口类）各个成员：

```
typedef struct _WNDCLASS {
    UINT        style;          // 指定类样式。这个成员可以是类样式的任何组合。
    WNDPROC     lpfnWndProc;    // 函数指针，回调函数 -- 指向窗口过程的指针
    int         cbClsExtra;     // 用于窗口类的额外大小
    int         cbWndExtra;     // 指定要在窗口实例之后分配的额外字节数
    HINSTANCE   hInstance;      // 包含类窗口过程的实例的句柄
    HICON       hIcon;          // 类图标的句柄。此成员必须是图标资源的句柄。如果该成员为NULL，系统将提供一个默认图标。
    HCURSOR     hCursor;        // 类光标的句柄
    HBRUSH      hbrBackground;  // 类背景画笔的句柄
    LPCTSTR     lpszMenuName;   // 菜单的名字
    LPCTSTR     lpszClassName;  // 窗口类名，指向以空结尾的字符串 类名
} WNDCLASS, *PWNDCLASS;
```

WNDCLASS 结构体类成员 style 参数作用举例：

CS_DBLCLKS：当用户在属于类的窗口中双击鼠标时，向窗口过程发送一个双击消息。

默认参数：

CS_HREDRAW（Horizontal Redraw）：如果移动或大小调整改变了客户区域的宽度，则重新绘制整个窗口。

CS_VREDRAW（Vertical Redraw）：如果移动或大小调整改变了客户区域的高度，则重新绘制整个窗口。

使用实例：

```
    static TCHAR szBuffTitle[] = TEXT("MyWindowClass");
    // 设计注册窗口类
    WNDCLASS wndclass;
    //移动调整窗口，进行重绘，CS_HREDRAW -- 垂直，CS_VREDRAW -- 水平
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = FirstWindowProc; // 函数指针，回调函数 -- 指向窗口过程的指针
    wndclass.cbClsExtra = 0;                // 预留的额外空间，用于窗口类的额外大小，一般为0
    wndclass.cbWndExtra = 0;                // 指定要在窗口实例之后分配的额外字节数，一般为0
    wndclass.hInstance = hInstance;         // 实例句柄
    wndclass.hIcon = NULL;                  // 为所有基于该窗口类的窗口设定一个图标
    wndclass.hCursor = NULL;                // 光标
    wndclass.hbrBackground = NULL;          // 指定窗口背景色为NULL
    wndclass.lpszMenuName = NULL;           // 菜单的名字
    wndclass.lpszClassName = szBuffTitle;   // 窗口类名，指向以空结尾的字符串 类名
```

位运算在SDK中的使用

使用 位或 进行位表示的优点：

- 状态可以自由组合，方便相同的风格同时设置

代码示例：

```
// 设计注册窗口类
WNDCLASS wndclass;
//移动调整窗口，进行重绘，CS_HREDRAW -- 垂直，CS_VREDRAW -- 水平
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;

/*
位运算在SDK中的广泛应用:
int - 1 2 3 4 5 6, 不方便多种风格的组合，表示不明确
*/

// 取消上述的CS_DBLCLKS风格
//wndclass.style &= ~CS_DBLCLKS; // ~CS_DBLCLKS --> 0111 ### 1011 & 0111 == 0011
wndclass.style ^= CS_DBLCLKS;      // 1011 ^ 1000 == 0011

```

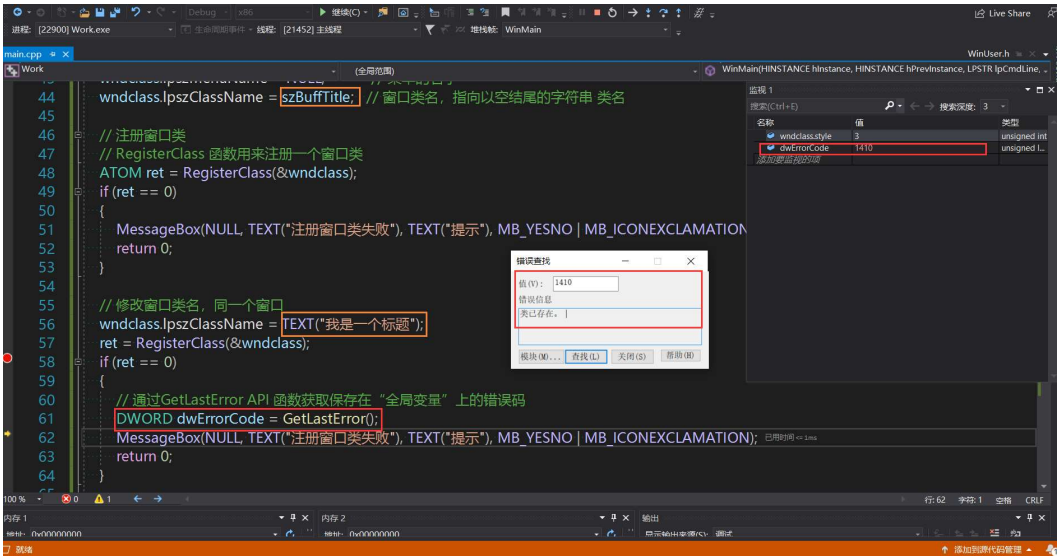
取消风格：

```
WNDCLASS wc;
wc.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS; // 11

// 取消样式的两种风格
wc.style &= ~CS_DBLCLKS;  // 3
wc.style ^= CS_DBLCLKS;
```

通过GetLastError API 获取错误码

通过 RegisterClass 函数注册一个窗口类，通过 GetLastError 函数 获取API函数的错误码并返回错误码。

示例：



为了方便，Windows 提供 FormatMessage API 函数可以将错误码信息转换成对应的错误信息。

使用方式如下：

Remarks

The **FormatMessage** function can be used to obtain error message strings for the system error codes returned by **GetLastError**, as shown in the following sample code.

```
LPVOID lpMsgBuf;
FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER |
    FORMAT_MESSAGE_FROM_SYSTEM |
    FORMAT_MESSAGE_IGNORE_INSERTS,
    NULL,
    GetLastError(),
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
    (LPTSTR) &lpMsgBuf,
    0,
    NULL
);
// Process any inserts in lpMsgBuf.
// ...
// Display the string.
MessageBox( NULL, (LPCTSTR)lpMsgBuf, "Error", MB_OK | MB_ICONINFORMATION );
// Free the buffer.
LocalFree( lpMsgBuf );
```

封装成一个函数，使用即可

```
// 显示错误信息
void ShowErrorMsg()
{
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        GetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
        (LPTSTR)&lpMsgBuf,
        0,
        NULL
    );
    // Process any inserts in lpMsgBuf.
    // ...
    // Display the string.
    MessageBox(NULL, (LPCTSTR)lpMsgBuf, TEXT("Error"), MB_OK | MB_ICONINFORMATION);
    // Free the buffer.
    LocalFree(lpMsgBuf);
}
```

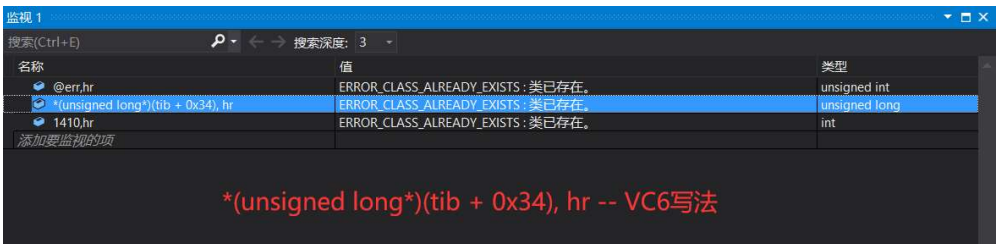
获取错误代码

```
    // 修改窗口类名，同一个窗口
    wndclass.lpszClassName = TEXT("我是一个标题");
    ret = RegisterClass(&wndclass);
    if (ret == 0)
    {
        // 通过GetLastError API 函数获取保存在“全局变量”上的错误码
        //DWORD dwErrorCode = GetLastError();
        ShowErrorMsg();    // 调用 ShowErrorMsg，将错误代码转换成错误信息并显示出来
        MessageBox(NULL, TEXT("注册窗口类失败"), TEXT("提示"), MB_YESNO | MB_ICONEXCLAMATION
        return 0;
    }

}

// 显示错误信息
void ShowErrorMsg()
{
```

VS 提供直接在监视窗口查看错误码信息

- @err,hr
- *(unsigned long*)(tib + 0x34), hr
- 错误码,hr (1410,hr)

  操作示例：



创建窗口实例

API -- CreateWindow

CreateWindow 说明

CreateWindow函数创建一个重叠、弹出窗口或子窗口。它指定窗口类、窗口标题、窗口样式和(可选)窗口的初始位置和大小。该函数还指定了窗口的父窗口或所有者(如果有的话)以及窗口的菜单。

```
    HWND CreateWindow(
    LPCTSTR lpClassName,  // 窗口类名称
    LPCTSTR lpWindowName, // 窗口标题
    DWORD dwStyle,        // 窗口风格，或称窗口格式
    int x,                // 初始 x 坐标，将其参数设置为CW_USEDEFAULT时，由系统提供一个默认值
    int y,                // 初始 y 坐标，将其参数设置为CW_USEDEFAULT时，由系统提供一个默认值
    int nWidth,           // 初始 x 方向尺寸，将其参数设置为CW_USEDEFAULT时，由系统提供一个默认值
    int nHeight,          //  初始 y 方向尺寸，将其参数设置为CW_USEDEFAULT时，由系统提供一个默认值
    HWND hWndParent,      // 父窗口句柄，不使用填NULL
    HMENU hMenu,          // 窗口菜单句柄，不使用填NULL
    HINSTANCE hInstance,  // 程序实例句柄
    LPVOID lpParam        // 创建参数，不使用填NULL
);
```

参数三 窗口风格的基本风格如下（三种风格互斥）：

- WS_CHILD -- 子窗口（子窗口必须要有父窗口的存在）
- WS_OVERLAPPED -- 存在更多的子窗口，内容比较丰富
- WS_POPUP -- 常用于弹出的对话框

三种基本风格互斥，且该参数的使用三者必须有一个出现。

子窗口的创建必须要有父窗口的存在。

一般使用的风格为 WS_OVERLAPPEDWINDOW，该风格覆盖了其它几种风格的属性：

创建带有WS_OVERLAPPED、WS_CAPTION、WS_SYSMENU、WS_THICKFRAME、WS_MINIMIZEBOX和WS_MAXIMIZEBOX样式的重叠窗口。与WS_TILEDWINDOW样式相同。

使用示例：

```
// 创建窗口实例
HWND hwnd = CreateWindow(
    szBuffTitle,    // 窗口类的名称
    TEXT("我是一个标题"), //窗口标题
    WS_OVERLAPPEDWINDOW,    // 窗口风格，重叠窗口
    CW_USEDEFAULT,      // 初始化 x 坐标 -- CW_USEDEFAULT 系统提供随机值
    CW_USEDEFAULT,      // 初始化 y 坐标
    CW_USEDEFAULT,      // 初始化 x 方向尺寸
    CW_USEDEFAULT,      // 初始化 y 方向尺寸
    NULL,   // 父窗口句柄
    NULL,   // 窗口菜单句柄
    hInstance,  // 程序实例句柄
    NULL    // 创建参数
);
// 如果函数失败，返回值为NULL。要获得扩展的错误信息，请调用GetLastError。
if (hwnd == NULL)
{
    ShowErrorMessage();
    return 0;
}
```

Windows提供了一个函数，可以将错误代码转换成它的文本描述。该函数称为**FormatMessage**:

显示窗口

API函数 -- ShowWindow

ShowWindow 函数设置指定窗口的显示状态。

定义如下：

```
57
58    BOOL ShowWindow
59    (
60        HWND hWnd,      // 窗口的句柄
61        int nCmdShow    // 显示状态
62    );
63
```

使用示例：

```
91
92    // 显示窗口
93    ShowWindow(hwnd, SW_SHOW);
94
```

更新窗口 API函数 -- UpdateWindow

如果窗口的更新区域不是空的，UpdateWindow函数通过发送WM_PAINT消息来更新指定窗口的客户区域。函数直接发送WM_PAINT消息到指定窗口的窗口过程，绕过应用程序队列。如果更新区域为空，则不会发送任何消息。

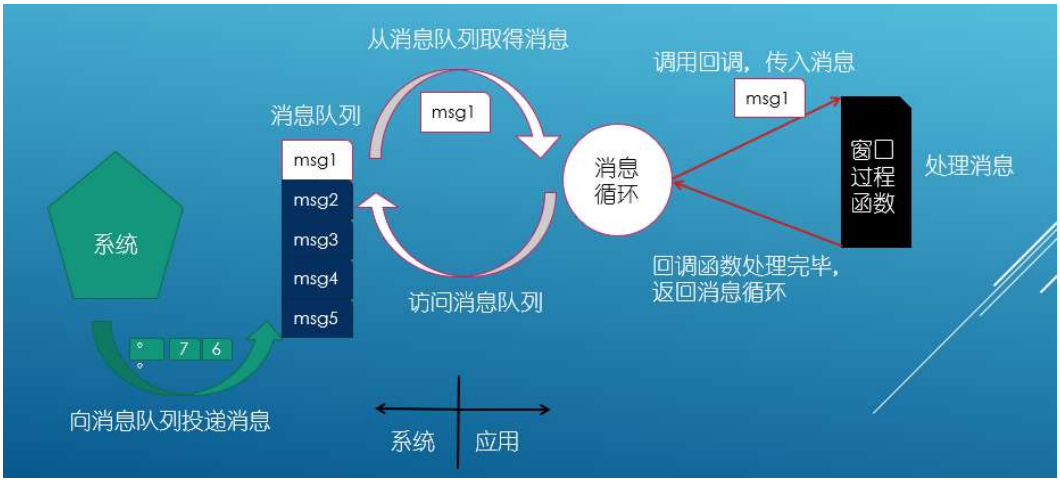
定义如下：

```
64
65    BOOL UpdateWindow
66    (
67        HWND hWnd    //窗口的句柄
68    );
69
```

使用示例：

```
95    // 4. 更新窗口
96    UpdateWindow(hwnd);
97
```

消息循环



建立消息循环 API函数 -- GetMessage

简单来说就是，函数从系统中的消息列表中获取消息。

MSDN说明：GetMessage函数从调用线程的消息队列中检索消息。该函数分派传入发送的消息，直到发布的消息可供检索为止。

GetMessage 从消息队列中获取消息后，该消息就从消息队列中弹出。

定义：

```
BOOL GetMessage
(
    LPMSG lpMsg,          // 传出参数，指向一个结构体MSG,该结构体是系统对消息的封装
    HWND hWnd,            // 窗口的句柄，用于获取指定窗口的消息 或者 获取所有窗口的消息
    UINT wMsgFilterMin,   // 获取消息的最小值，如果要获取所有消息，参数为0
    UINT wMsgFilterMax    // 获取消息的最大值
);

typedef struct tagMSG
{
    HWND   hwnd;         // 用于获取指定的窗口消息
    UINT   message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD  time;
    POINT  pt;
} MSG, *PMSG;
```

使用示例：

```
// 5. 建立消息循环 -- 通过GetMessage API 函数从系统中的消息列表中获取消息
MSG msg;
BOOL bRet = GetMessage(&msg, NULL, 0, 0);
```

将获取到的消息进行处理，需要将消息传递给回调函数（窗口过程函数）-- DispatchMessage。

函数说明：**DispatchMessage** 函数向窗口过程分派消息。它通常用于分派由GetMessage函数检索的消息。

DispatchMessage -- 主要作用：根据窗口调用对应窗口的回调函数（窗口过程函数），会进行自动区分。

定义：

```
LRESULT DispatchMessage(
  CONST MSG *lpmsg    // 将MSG 当作参数传递，其会自动调用回调函数
);
```

使用实例：

```
    // 使用循环，使其不定的从系统的消息队列中获取消息
    while (GetMessage(&msg, NULL, 0, 0))
    {
        DispatchMessage(&msg);  // 调用回调函数，这里只是获取消息，并未处理
    }
    return 0;
```

DefWindowProc：-- 自动执行内定的消息进行处理。

函数说明：**DefWindowProc** 函数调用默认窗口过程来为应用程序不处理的任何窗口消息提供默认处理。此函数确保处理每条消息。使用窗口过程接收到的相同参数调用 **DefWindowProc**。

定义：

```
    LRESULT DefWindowProc
    (
        HWND hWnd,      // handle to window
        UINT Msg,       // message identifier
        WPARAM wParam,  // first message parameter
        LPARAM lParam   // second message parameter
    );
```

实现窗口过程函数 -- WindowProc

使用示例：

```
// 窗口过程函数
LRESULT CALLBACK FirstWindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

PostQuitMessage

使用 **PostQuitMessage** API函数向系统消息队列中压入一个Quit消息。其参数为一个退出码。

```
    VOID PostQuitMessage
    (
    int nExitCode   // exit code
    );
```

什么时候使用?
当点击 "X"（关闭窗口的时候）-- 使用 WM_CLOSE

WM_CLOSE：WM_CLOSE 消息作为一个窗口或应用程序应该终止的信号被发送。

```
115
116     LRESULT CALLBACK WindowProc
117     (
118         HWND hwnd,        // handle to window
119         UINT uMsg,        // WM_CLOSE
120         WPARAM wParam,    // not used  -- 不带任何参数
121         LPARAM lParam     // not used
122     );
123
```

鼠标消息

鼠标消息有左，右键按下和弹起，移动，双击。

API函数 -- WM_LBUTTONDOWN

说明：WM_LBUTTONDOWN 消息是当用户按下鼠标左键时，而光标是在一个窗口的客户区。如果未捕获鼠标，则消息将被发送到光标下方的窗口。否则，消息将被发布到捕获鼠标的窗口。

窗口通过其WindowProc函数接收此消息。

定义：

```
125     LRESULT CALLBACK WindowProc
126     (
127         HWND hwnd,        // 窗口的句柄
128         UINT uMsg,        // WM_LBUTTONDOWN
129         WPARAM wParam,    // 按键说明
130         LPARAM lParam     // 坐标 -- 将int值进行拆分 高字X坐标，低字Y坐标
131     );
132
```

参数 LPARAM lParam 说明：

低阶单词指定光标的x坐标。坐标相对于客户区域的左上角。
高阶字指定光标的y坐标。坐标相对于客户区域的左上角。

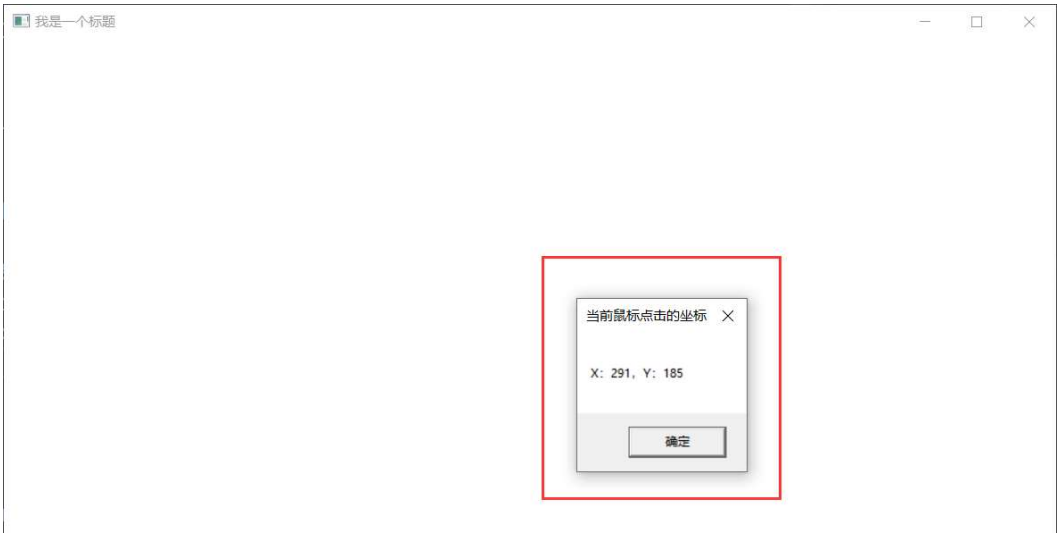
Remarks

Use the following code to obtain the horizontal and vertical position:

```
xPos = GET_X_LPARAM(lParam);
yPos = GET_Y_LPARAM(lParam);
```

使用示例：

```
145     case WM_LBUTTONDOWN:
146     {
147         WORD wXPos = LOWORD(lParam);    // 获取X坐标
148         WORD wYPos = HIWORD(lParam);    // 获取Y坐标
149
150         TCHAR szBuff[MAXBYTE] = { 0 };
151         wsprintf(szBuff, TEXT("X: %d, Y: %d"), wXPos, wYPos);
152         MessageBox(NULL, szBuff, TEXT("当前鼠标点击的坐标"), MB_OK);
153         break;
154     }
```


我是一个标题
当前鼠标点击的坐标
X：291，Y：185
确定