

2020/06/24_SDK_第4课_资源和对话框

笔记本: SDK编程

创建时间: 2020/6/24 星期三 15:46

作者: ileemi

- [什么是资源](#)
- [资源的存储:](#)
- [如何在SDK中使用资源](#)
 - [添加窗口图标](#)
 - [LoadIcon](#)
 - [添加光标 API -- LoadCursor](#)
 - [窗口背景 API -- RegisterClass](#)
 - [窗口菜单 -- 输入菜单ID即可](#)
 - [LoadMenu](#)
 - [菜单消息的相应 API -- WM_COMMAND](#)
- [快捷键的使用 \(组合按键\)](#)
- [带资源的编译流程](#)
- [命令行编译](#)
- [将图标绘画在窗口内](#)

什么是资源

使用的图片, 音频, 内存可以称之为使用了资源

在程序中, 代码以外的东西, 代码所使用的东西称之为资源。

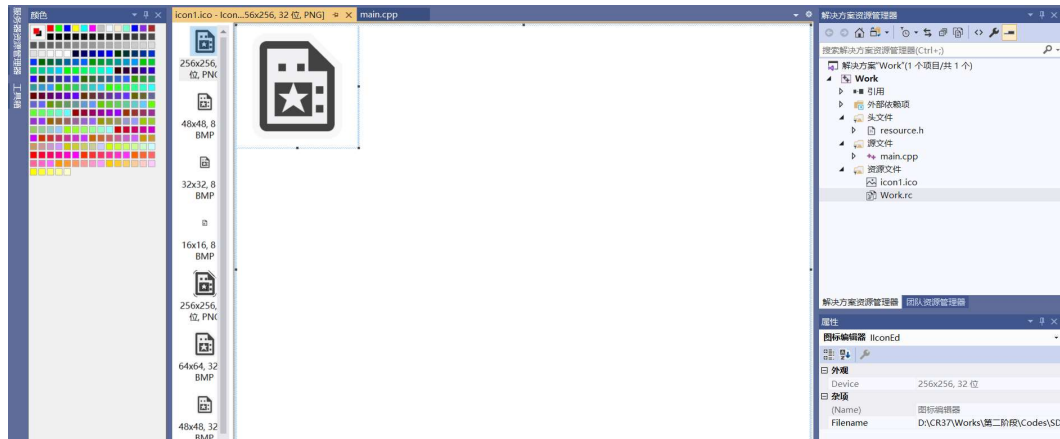
资源的存储:

- 资源和代码分开存储 (例如: html), 优点: 修改方便, 缺点: 数据文件易被改写、易丢失。
- windows使用的数据存储方式: 将资源和代码放在同一个文件中, 优点: 资源文件不易丢失, 缺点: 资源更改较为麻烦

如何在SDK中使用资源

在工程添加资源:

点击项目（右键）--> 添加 --> 添加资源



添加窗口图标

窗口添加图标的位置：注册类的地方

获取图标句柄API -- **LoadIcon**

LoadIcon

函数说明：LoadIcon 函数从与应用程序实例关联的可执行文件(.exe)加载指定的图标资源。

函数定义：

```
5
6 HICON LoadIcon
7 (
8     HINSTANCE hInstance, // handle to application instance -- 应用程序实例句柄
9     LPCTSTR lpIconName    // name string or resource identifier -- 名称字符串或资源标识符
10 );
11 // 使用MAKEINTRESOURCE宏创建这个ID值
12
13 LPCTSTR MAKEINTRESOURCE
14 (
15     WORD wInteger // 指定要转换的整数值
16 );
17
```

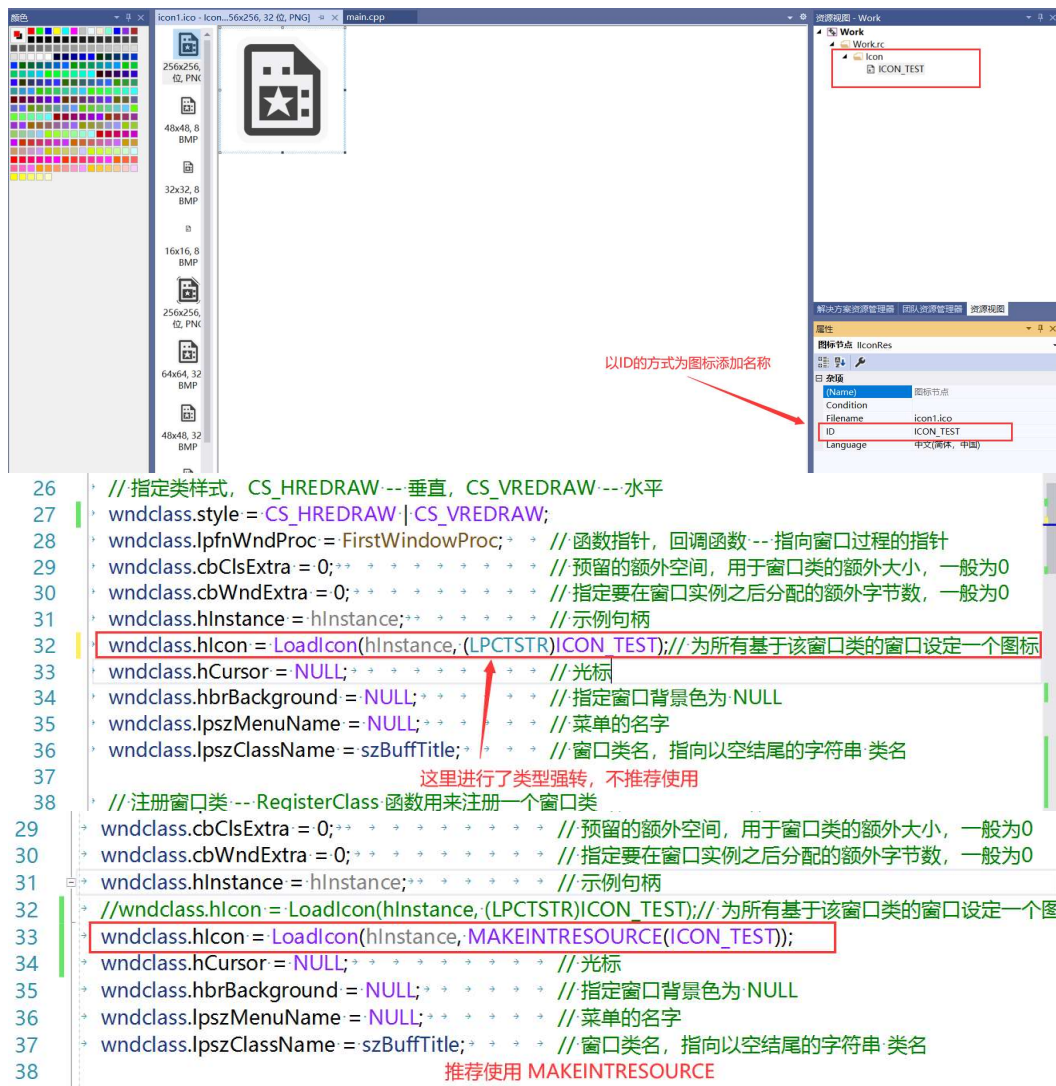
注意，这个函数已经被LoadImage函数取代了

窗口图标 Icon 添加值的两种方式：

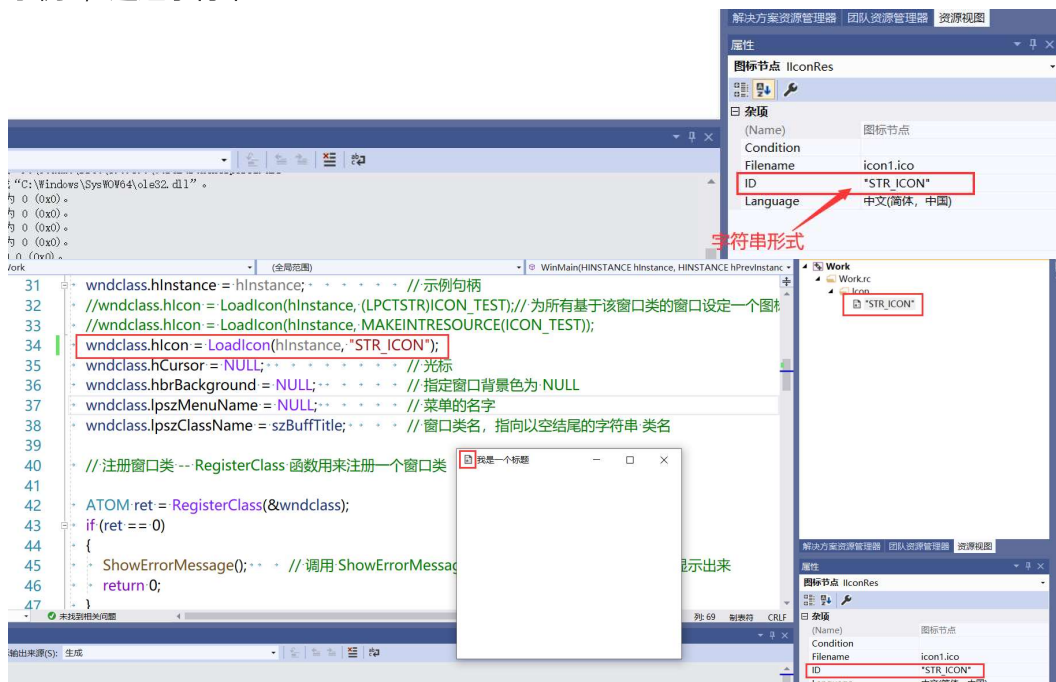
- 指向一个以空结尾的字符串的指针，该字符串包含要加载的图标资源的名称 -- 字符串
- ID -- 图标资源的标识符 -- 数值

使用示例：

示例1，通过ID



示例2, 通过字符串



一般使用ID

添加光标 API -- LoadCursor

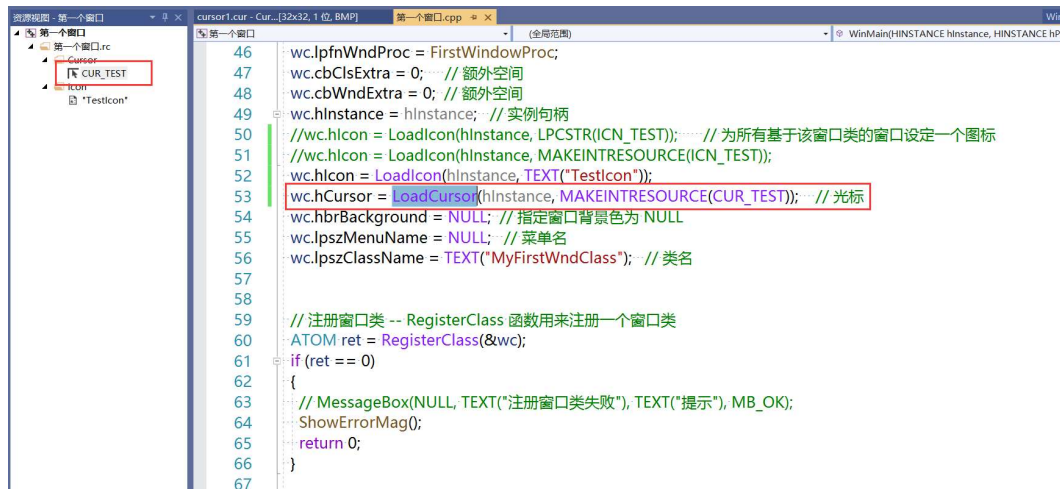
函数说明：

LoadCursor函数从与应用程序实例关联的可执行文件(. exe)加载指定的游标资源。注意，这个函数已经被 LoadImage 函数取代了。

函数定义：

```
19
20 HCURSOR LoadCursor(
21     HINSTANCE hInstance, // handle to application instance -- 应用程序实例句柄
22     LPCTSTR lpCursorName // name or resource identifier -- 名称或资源标识符
23 );
24
25 LPTSTR MAKEINTRESOURCE
26 (
27     WORD wInteger // 指定要转换的整数值
28 );
```

使用方法类似于 LoadIcon：



窗口背景 API -- RegisterClass

RegisterClass类中的参数指向的结构体中



注意：

使用系统预定义的颜色是值需要加 1

使用示例：

```
32 + wndclass.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(ICON_TEST));
33 + wndclass.hCursor = LoadCursor(hInstance, MAKEINTRESOURCE(CUR_TEST)); // 光标
34 + wndclass.hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE + 1); // 指定窗口背景色
35 + wndclass.lpszMenuName = NULL; // 菜单的名字
36 + wndclass.lpszClassName = szBuffTitle; // 窗口类名 指向以空结尾的字符串 类名
37
38 + // 注册窗口类 -- RegisterClass 函数用来注册一个窗口类
39
40 + ATOM ret = RegisterClass(&wndclass);
41 + if (ret == 0)
42 + {
43 +     ShowErrorMessage(); // 调用 ShowErrorMessage
```



窗口菜单 -- 输入菜单ID即可

添加窗口菜单：



```
+ // 示例句柄
+ MAKEINTRESOURCE(ICON_TEST));
+ MAKEINTRESOURCE(CUR_TEST)); // 光标
+ OR_APPWORKSPACE + 1); // 指定窗口背景色
+ // 菜单的名字
+ // 窗口类名, 指向以空结尾的字符串 类名
```

一个窗口类

为菜单添加ID



使用示例：

MSDN Library - October 2001

文件(F) 编辑(E) 查看(V) 转到(G) 帮助(H)

隐藏 查找 上一步 下一步 前进 后退 刷新 主页 打印

活动子集(S)

Platform SDK: Windows User Interface

WNDCLASS

The **WNDCLASS** structure contains the window class attributes that are registered by the **RegisterClass** function. This structure has been superseded by the **WNDCLASSEX** structure used with the **RegisterClassEx** function. You can still use **WNDCLASS** and **RegisterClass** if you do not need to set the small icon associated with the window class.

```
typedef struct _WNDCLASS {
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HINSTANCE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpMenuName;
    LPCTSTR lpClassName;
} WNDCLASS, *PWNDCLASS;
```

Members

- style** Specifies the class style(s). The class style is a combination of the **WS_** and **WS_EX_** macros.
- lpfnWndProc** Pointer to the window procedure. For more information, see **Window Procedure**.
- cbClsExtra** Specifies the number of extra bytes to zero.
- cbWndExtra** Specifies the number of extra bytes to zero.
- hInstance** Handle to the instance of the application.
- hIcon** Handle to the icon.
- hCursor** Handle to the cursor.
- hbrBackground** Handle to the brush used to fill the background of the window.
- lpMenuName** Pointer to a null-terminated character string that specifies the resource name of the class menu, as the name appears in the resource file. If you use an integer to identify the menu, use the **MAKEINTRESOURCE** macro. If this member is NULL, windows belonging to this class have no default menu.
- lpClassName** Pointer to a null-terminated character string that specifies the class name, as the name appears in the resource file. If you use an integer to identify the class, use the **MAKEINTRESOURCE** macro.

Members

- style** Specifies the class style(s). The class style is a combination of the **WS_** and **WS_EX_** macros.
- lpfnWndProc** Pointer to the window procedure. For more information, see **Window Procedure**.
- cbClsExtra** Specifies the number of extra bytes to zero.
- cbWndExtra** Specifies the number of extra bytes to zero.
- hInstance** Handle to the instance of the application.
- hIcon** Handle to the icon.
- hCursor** Handle to the cursor.
- hbrBackground** Handle to the brush used to fill the background of the window.
- lpMenuName** Pointer to a null-terminated character string that specifies the resource name of the class menu, as the name appears in the resource file. If you use an integer to identify the menu, use the **MAKEINTRESOURCE** macro. If this member is NULL, windows belonging to this class have no default menu.
- lpClassName** Pointer to a null-terminated character string that specifies the class name, as the name appears in the resource file. If you use an integer to identify the class, use the **MAKEINTRESOURCE** macro.

```
33  wndclass.hCursor = LoadCursor(hInstance, MAKEINTRESOURCE(CUR_TEST)); // 光标
34  wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1); // 指定窗口背景色
35  wndclass.lpMenuName = MAKEINTRESOURCE(MENU_TEST); // 窗口菜单
36  wndclass.lpClassName = szBuffTitle; // 窗口类名, 指向以空结尾的字符串. 类名
37
38  // 注册窗口类 -- RegisterClass 函数用来注册一个窗口类
39
40  ATOM ret = RegisterClass(&wndclass);
```

我是一个标题

文件 编辑

打开

保存

另存为

注意：

菜单需要添加ID 子菜单同样

添加菜单的位置：

- 1、注册窗口类的地方
- 2、创建窗口类的地方 -- 需要调用 API函数 -- LoadMenu

LoadMenu

函数说明：

LoadMenu函数从与应用程序实例关联的可执行文件(.exe)加载指定的菜单资源。

在创建窗口类的地方添加菜单，使用示例：

```
47  // 创建窗口示例
48  HWND hwnd = CreateWindow(
49      szBuffTitle, // 窗口类的名称
50      TEXT("我是一个标题"), // 窗口标题
51      WS_OVERLAPPEDWINDOW, // 窗口风格, 重叠窗口
52      CW_USEDEFAULT, // 初始化 x 坐标 -- CW_USEDEFAULT 系统提供随机值
53      CW_USEDEFAULT, // 初始化 y 坐标
54      1000, // 初始化 x 方向尺寸
55      600, // 初始化 y 方向尺寸
56      NULL, // 父窗口句柄
57      LoadMenu(hInstance, MAKEINTRESOURCE(MENU_TEST)), // 窗口菜单, 此处会覆盖定义处的窗口菜单
58      hInstance, // 程序实例句柄
59      NULL, // 创建参数
60  );
61
```

我是一个标题

文件 编辑

打开

保存

另存为

行: 50 字符: 29 列: 74 制表符 CRLF

菜单创建后，点击没有反应，应为这个时候还没有进行菜单消息相应（没有处理菜单消息）

菜单消息的相应 API -- WM_COMMAND

函数说明：当用户从菜单中选择命令项时，当控件向父窗口发送通知消息时，或者当一个快捷键被翻译时，WM_COMMAND消息被发送。

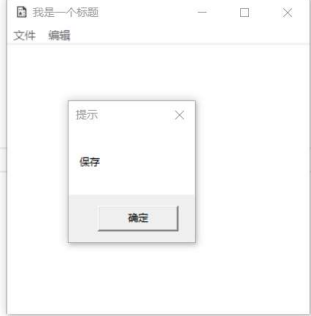
accelerator keystroke --> 快捷键

函数定义：

```
30
31 LRESULT CALLBACK WindowProc(
32     HWND hwnd,          // handle to window
33     WM_COMMAND,         // the message to send
34     WPARAM wParam,      // notification code and identifier
35     LPARAM lParam       // handle to control (HWND)
36 );
37
38 /*
39 参数:
40 WPARAM wParam
41 如果消息来自控件，则高阶字指定通知代码。如果消息来自加速器，则此值为1。如果消息来自菜单，则此值为零。
42 低阶字指定菜单项、控件或加速器的标识符。
43 */
```

使用示例：

```
91 case WM_COMMAND:
92 {
93     WORD wFrom = HIWORD(wParam); // 高字
94     WORD wID = LOWORD(wParam); // 低字
95
96     // 来自于菜单
97     if (wFrom == 0)
98     {
99         // 菜单消息
100         switch (wID)
101         {
102             case MENU_FILE_OPEN:
103                 MessageBox(NULL, "打开", "提示", MB_OK);
104                 break;
105             case MENU_FILE_SAVE:
106                 MessageBox(NULL, "保存", "提示", MB_OK);
107                 break;
108             case MENU_FILE_SAVEAS:
109                 MessageBox(NULL, "另存为", "提示", MB_OK);
110                 break;
111         }
112     }
113 }
```



通过WM_COMMAND 的参数：WPARAM wParam 可知：

如果消息来自控件，则高阶字指定通知代码。如果消息来自加速器，则此值为1。如果消息来自菜单，则此值为零。

低阶字指定菜单项、控件或加速器的标识符。

所以在WM_COMMAND 这里需要判断输入的是不是组合按键

快捷键的使用（组合按键）

GetAsyncKeyState 可以在 LBUTTONDOWN 中使用组合按键

GetAsyncKeyState -- 可以获取其它按键的状态。

函数说明：

GetAsyncKeyState 函数确定在调用该函数时键是向上的还是向下的，以及该键是否在之前调用 GetAsyncKeyState 后被按下。

上述的方法比较麻烦，可以通过**资源的方式**使用快捷键（较为简单）：

添加快捷键资源：



使用快捷键时需要进行加载，加载快捷键的 API 函数 -- LoadAccelerators

使用示例：

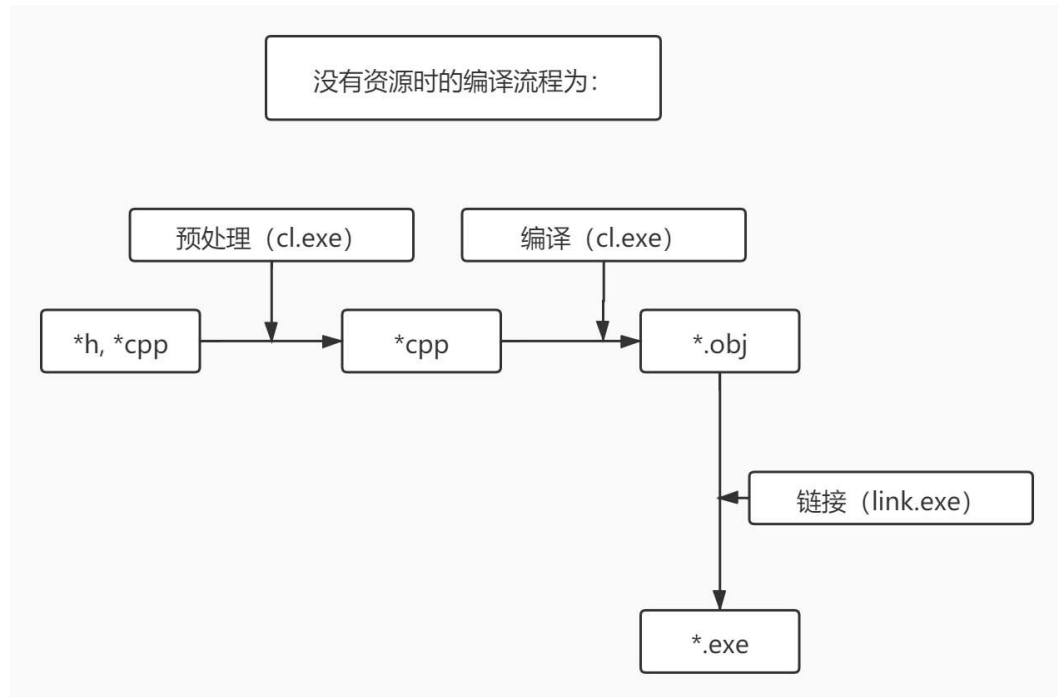
```
// 加载快捷键
HACCEL hAccel = LoadAccelerators(hInstance, MAKEINTRESOURCE(ACCEL_TEST));
```

同时还需要 **TranslateAccelerator** 函数，用来将键盘按键消息转换成快捷键消息

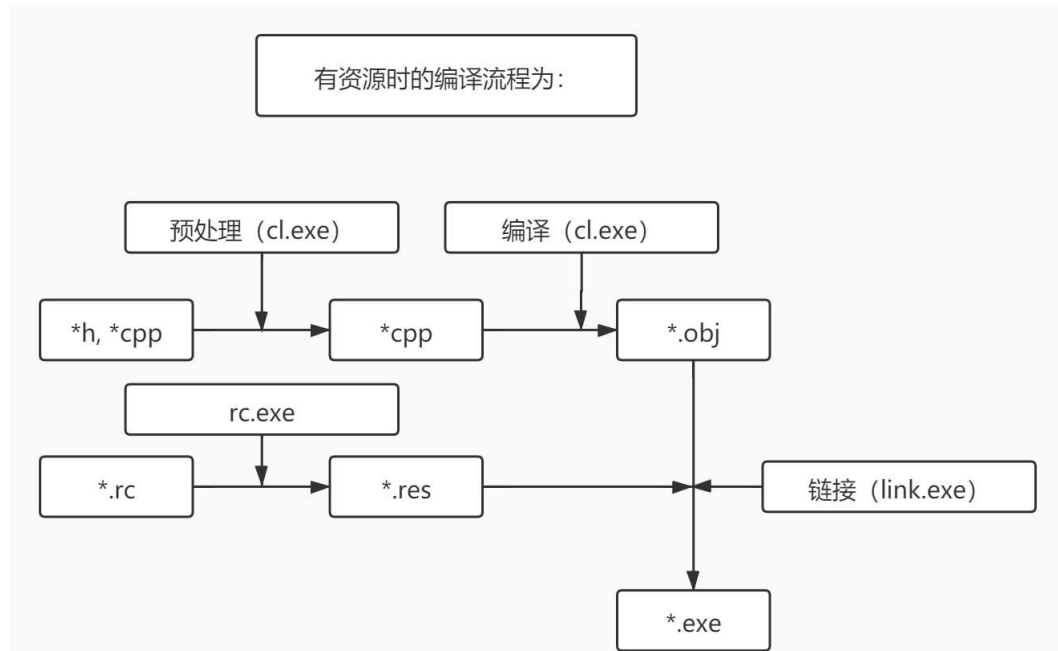
```
80
81 // 建立消息循环 -- 通过 GetMessage API 函数从系统中的消息列表中获取消息
82 MSG msg;
83 while (GetMessage(&msg, NULL, 0, 0))
84 {
85     // 将键盘消息转换成快捷键消息
86     // 当 TranslateAccelerator 为真，将键盘消息转换成快捷键消息
87     // TranslateAccelerator 函数内部会自动判断键盘的按键数值，有没有组合按键
88     // 如果有快捷键其函数内部会自动发送一个快捷键消息
89     if (!TranslateAccelerator(hwnd, hAccel, &msg))
90     {
91         // 如果 TranslateAccelerator 函数检测到的按键操作没有组合按键，执行这里的语句
92         DispatchMessage(&msg); // 根据窗口调用对应窗口的回调函数
93     }
94 }
```

带资源的编译流程

没有资源时的编译流程为：



有资源时的编译流程为：



名称	修改日期	类型	大小
Work.tlog	2020/6/24 23:35	文件夹	
main.obj	2020/6/24 23:35	OBJ 文件	26 KB
vc142.idb	2020/6/24 23:35	VC++ Minimum ...	227 KB
vc142.pdb	2020/6/24 23:35	Program Debug ...	132 KB
Work.Build.CppClean.log	2020/6/24 23:35	文本文档	2 KB
Work.log	2020/6/24 23:35	文本文档	1 KB
Work.res	2020/6/24 23:35	Compiled Resou...	46 KB
Work.vcxproj.FileListAbsolute.txt	2020/6/24 23:35	文本文档	0 KB

通过link.exe 得到main.exe

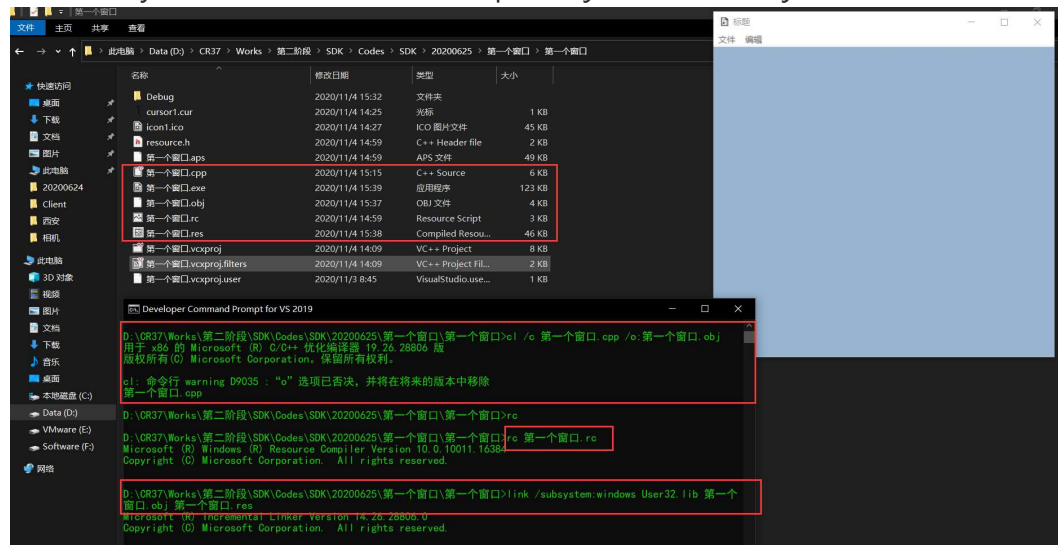
命令行编译

命令:

```
cl /c pch.cpp 第一个窗口.cpp /o:第一个窗口.obj
```

```
rc 第一个窗口.rc
```

```
link /subsystem:windows User32.lib pch.obj 第一个窗口.obj 第一个窗口.res
```



将图标绘画在窗口内

使用API:

DrawIcon -- DrawIcon函数将图标或光标绘制到指定的关联设备中。

GetModuleHandle -- 获取模块句柄

实例:

```
308 case WM_PAINT:
309 {
310     PAINTSTRUCT ps;
311     HDC hDC = BeginPaint(hwnd, &ps); // 获取DC句柄
312
313     HICON hlcon = LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(ICON_DRAW))
314
315     DrawIcon(hDC, 0, 0, hlcon);
316
317     EndPaint(hwnd, &ps);
318     return 0;
319 }
```