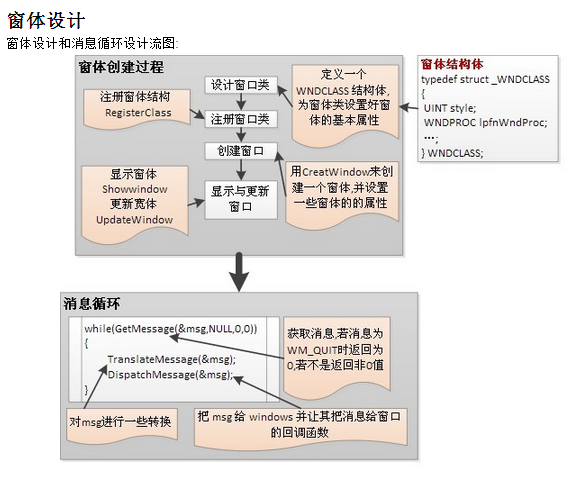
**// 设计窗口**

**WNDCLASS wndclass;**

**Wndclass.cbClsExtra=0;**

**Wndclass.cbWndExtra=0;**

**Wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE\_BRUSH);**

**Wndclass.hCursor=LoadCursor(NULL,IDC\_ARROW);**

**Wndclass.hIcon=LoadIcon(NULL,IDI\_ERROR);**

**Wndclass.hInstance= hiInstance;**

**Wndclass.ipfnWndProc=textprom;**

**Wndclass.lpszClassName=”text;**

**Wndclass.lpszMenuName=NULL;**

**Wndclass.style=CS\_HREDRAW | CS\_VREDRAW;**

**//注册窗口类**

**If(!RegisterClass(&wndclass))**

**{**

**MessageBox(NULL,”creat windows error!”,”error”,MB\_OK| MB\_ICONSTOP);**

**}**

**//创建窗口**

**HWND hwnd =CreateWindow(“text”,”hellow world “,WS\_DLGFRAME |WS\_MINIMIZEBOX| WS\_SYSMENU,CW\_USEDEFAULT,CW\_USEDE,CW\_USEDEFAULT,CW\_USEDEFAULT,NULL,NULL,hInstance,NULL);**

**// 显示更新窗口**

**ShowWindow(hwnd,nCmdShow);**

**UpdateWindow(hwnd);**

**MSG msg;**

**While(GetMessage(&msg,NULL,0,0))**

**{**

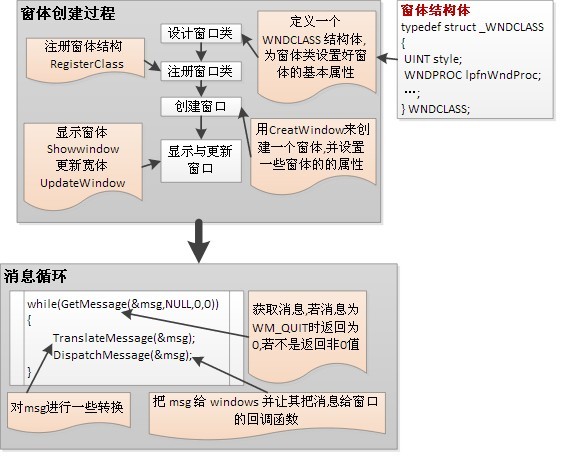
**TranslateMessage(&msg);**

**DispatchMessage(&msg);**

**}**

**窗体设计**

窗体设计和消息循环设计流图:



代码示例:

**[cpp]** [view plain](http://blog.csdn.net/kz_ang/article/details/8034469) [copy](http://blog.csdn.net/kz_ang/article/details/8034469)

[print?](http://blog.csdn.net/kz_ang/article/details/8034469)

1. //设计窗口
2. WNDCLASS wndclass;
4. wndclass.cbClsExtra=0;
5. wndclass.cbWndExtra=0;
6. wndclass.hbrBackground=(**HBRUSH**)GetStockObject(WHITE\_BRUSH);
7. wndclass.hCursor=LoadCursor(NULL,IDC\_ARROW);
8. wndclass.hIcon=LoadIcon(NULL,IDI\_ERROR);
9. wndclass.hInstance=hInstance;
10. wndclass.lpfnWndProc=textprom;
11. wndclass.lpszClassName="text";
12. wndclass.lpszMenuName=NULL;
13. wndclass.style=CS\_HREDRAW | CS\_VREDRAW;

16. //注册窗口类
17. **if**(!RegisterClass(&wndclass))
18. {
19. MessageBox(NULL,"create windows error!","error",MB\_OK | MB\_ICONSTOP);
20. }

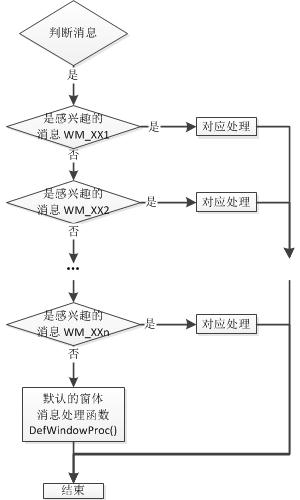
23. //创建窗口
24. **HWND** hwnd=CreateWindow("text","hellow world",WS\_DLGFRAME | WS\_MINIMIZEBOX | WS\_SYSMENU,CW\_USEDEFAULT,CW\_USEDEFAULT,
25. CW\_USEDEFAULT,CW\_USEDEFAULT,NULL,NULL,hInstance,NULL);

28. //显示更新窗口
29. ShowWindow(hwnd,nCmdShow);
30. UpdateWindow(hwnd);

33. //消息循环
34. MSG msg;
35. **while**(GetMessage(&msg,NULL,0,0))
36. {
37. TranslateMessage(&msg);
38. DispatchMessage(&msg);
39. }

**回调函数设计**

设计流图:



设计大致模型:

**[cpp]** [view plain](http://blog.csdn.net/kz_ang/article/details/8034469) [copy](http://blog.csdn.net/kz_ang/article/details/8034469)

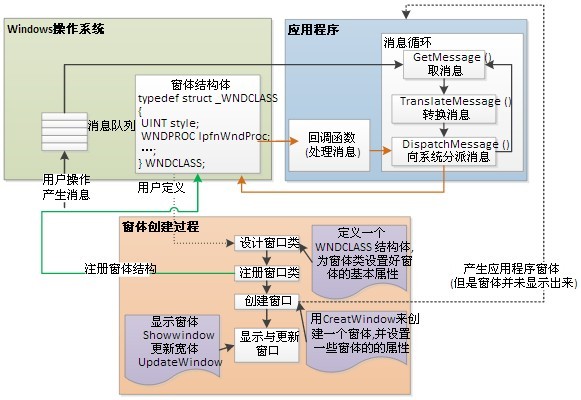
[print?](http://blog.csdn.net/kz_ang/article/details/8034469)

1. **LRESULT** CALLBACK WindowProc(
2. **HWND** hwnd,     // handle to window
3. **UINT** uMsg,     // message identifier
4. **WPARAM** wParam, // first message parameter
5. **LPARAM** lParam   // second message parameter
6. )
7. {
8. ……
9. **switch**(uMsg)
10. {
11. **case** ‥ : ……;
12. **break**;
13. ……
14. **case** WM\_DESTROY:  PostQuitMessage(0);//在消息队列尾部插入一个WM\_QUIT消息
15. **break**;
16. **default**: **return** DefWindowProc(hwnd,uMsg,wParam,lParam);
17. }
18. ……
19. **return** 0;
20. }

注意:

1. 必须把所有不处理的消息交给 DefWindowPro 函数处理,也要把它的返回值返回给 windows 否则 windows 就失去了与应用程序通信的途径也就是说不能在控制窗口的行为
2. WM\_DESTROY 是窗口函数必须处理的消息,因为在窗体销毁的时候并不会主动向程序发送一个 WM\_QUIT 这个消息,所以我们的窗体即使销毁了程序依旧还在消息循环中,为了达到在销毁窗体时候并且退出消息循环我们应用处理这个WM\_DESTROY 这个消息,在程序接收到这个消息时候向消息队列发一个 WM\_QUIT 消息来退出消息循环
3. WM\_CLOSE 默认由 DefWindowPro 函数处理,它会调用 DestroyWindow 函数销毁窗口
4. WM\_CREATE 为 WndProc 第一处理的信息
5. 在视窗大小改变时,会发 WM\_SIZE 这个消息且在 lParam 中 LOWORD(lParam) 中为窗口横坐标 HIWORD(lParam) 中为窗口纵坐标
6. 回调函数的参数与 MSG 结构的前四位成员相同

**总体开发流程**



**程序样例**

**[cpp]** [view plain](http://blog.csdn.net/kz_ang/article/details/8034469) [copy](http://blog.csdn.net/kz_ang/article/details/8034469)

[print?](http://blog.csdn.net/kz_ang/article/details/8034469)

1. #include<windows.h>
2. #include"resource.h"
3. #include<string.h>
5. **LRESULT** CALLBACK textprom(
6. **HWND** hwnd,      // handle to window
7. **UINT** uMsg,      // message identifier
8. **WPARAM** wParam,  // first message parameter
9. **LPARAM** lParam   // second message parameter
10. );

13. **int** WINAPI WinMain(  **HINSTANCE** hInstance,  // handle to current instance
14. **HINSTANCE** hPrevInstance,  // handle to previous instance
15. **LPSTR** lpCmdLine,      // pointer to command line
16. **int** nCmdShow          // show state of window
17. )
18. {
19. WNDCLASS wndclass;
21. wndclass.cbClsExtra=0;
22. wndclass.cbWndExtra=0;
23. wndclass.hbrBackground=(**HBRUSH**)GetStockObject(WHITE\_BRUSH);
24. wndclass.hCursor=LoadCursor(NULL,IDC\_ARROW);
25. wndclass.hIcon=LoadIcon(NULL,IDI\_ERROR);
26. wndclass.hInstance=hInstance;
27. wndclass.lpfnWndProc=textprom;
28. wndclass.lpszClassName="text";
29. wndclass.lpszMenuName=NULL;
30. wndclass.style=CS\_HREDRAW | CS\_VREDRAW;

33. **if**(!RegisterClass(&wndclass))
34. {
35. MessageBox(NULL,"create windows error!","error",MB\_OK | MB\_ICONSTOP);
36. }

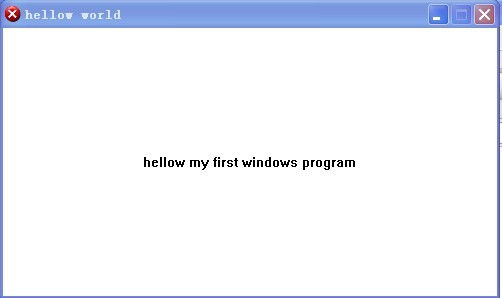
39. **HWND** hwnd=CreateWindow("text","hellow world",WS\_DLGFRAME | WS\_MINIMIZEBOX | WS\_SYSMENU,CW\_USEDEFAULT,CW\_USEDEFAULT,
40. CW\_USEDEFAULT,CW\_USEDEFAULT,NULL,NULL,hInstance,NULL);


44. ShowWindow(hwnd,nCmdShow);
45. UpdateWindow(hwnd);

48. MSG msg;
49. **while**(GetMessage(&msg,NULL,0,0))
50. {
51. TranslateMessage(&msg);
52. DispatchMessage(&msg);
53. }
55. **return** msg.wParam;
56. }


60. **LRESULT** CALLBACK textprom(
61. **HWND** hwnd,      // handle to window
62. **UINT** uMsg,      // message identifier
63. **WPARAM** wParam,  // first message parameter
64. **LPARAM** lParam   // second message parameter
65. )
66. {
67. **HDC** hdc;
68. PAINTSTRUCT ps;
69. RECT rect;
71. **switch**(uMsg)
72. {
73. **case** WM\_PAINT:
74. hdc=BeginPaint(hwnd,&ps);
75. GetClientRect(hwnd,&rect);
76. DrawText(hdc,"hellow my first windows program",strlen("hellow my first windows program"),&rect,
77. DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);
78. EndPaint(hwnd,&ps);
79. **break**;
80. **case** WM\_RBUTTONDOWN:
81. hdc=GetDC(hwnd);
82. TextOut(hdc,0,0,"success",strlen("success"));
83. ReleaseDC(hwnd,hdc);
84. **break**;
85. **case** WM\_RBUTTONUP:
86. GetClientRect(hwnd,&rect);
87. InvalidateRect(hwnd,&rect,**true**);
88. **break**;
89. **case** WM\_DESTROY:
90. PostQuitMessage(0);
91. **break**;
92. **default**:
93. ;
94. }
95. **return** DefWindowProc(hwnd,uMsg,wParam,lParam);
96. }

 示例图片



**透明窗口编写**

**步骤:**

* 调用 SetWindowLong 设置窗口属性,把窗口设置为具有GWL\_EXSTYLE扩展的窗口风格的窗口(可以用SetWindowLong(GetSafeHwnd(),GWL\_EXSTYLE,GetWindowLong(GetSafeHwnd(),GWL\_EXSTYLE) | WS\_EX\_LAYERED))
* 调用SetLayeredWindowAttributes设置窗口透明度

流程图如下:



**函数 SetLayeredWindowAttributes 说明:**

* 函数原型:   
  　　BOOL SetLayeredWindowAttributes(  
  　　　　HWND hwnd, // handle to the layered window 透明窗体的句柄  
  　　　　COLORREF crKey, // specifies the color key 颜色值,可以用RGB(r,g,b)来指定  
  　　　　BYTE bAlpha, // value for the blend function 透明度,取值范围是[0,255]  
  　　　　DWORD dwFlags // action 透明方式,  
  　　);
* dwFlags说明:
  + 当取值为LWA\_ALPHA = 0x2 (值为2) 时，crKey参数无效,bAlpha参数有效；
  + 当取值为LWA\_COLORKEY = 0x1 (值为1) 时,bAlpha参数无效,而窗体中的所有颜色为crKey的地方将变为透明
  + 也可以取两个值的组合：LWA\_ALPHA Or LWA\_COLORKEY.这样crKey的地方将变为全透明,而其它地方根据bAlpha参数确定透明度
* 要求:　　要使使窗体拥有透明效果,首先要有WS\_EX\_LAYERED扩展属性(可以调用SetWindowLong函数,设置窗体类具有扩展属性属性例如SetWindowLong(this->GetSafeHwnd(), GWL\_EXSTYLE, WS\_EX\_LAYERED) ;)

**窗体半透明控件不透明方法:**

也是使用SetLayeredWindowAttributes，不同的是，他还用了另一API函数SetParent，基本思路是将某个容器控件用SetParent另外指定一个父窗口，这样控件和主窗体就可以分别使用SetLayeredWindowAttributes函数，控制透明方式，以迅雷的悬浮窗而言就是，将主窗体以窗体透明的方式（LWA\_ALPHA标志）半透明，控件以指定颜色的方式（LWA\_COLORKEY标志）镂空特定颜色透明，但是由于控件指定了其他的父窗口，因此会有一个不主动跟随窗体移动和显示层次（窗体会覆盖控件）的问题，这些都需要手动控制,实际上就是等于创建两个窗体使用不同的透明方式，然后叠加在一起