VC中利用WinAPI实现自绘按钮类 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/273/2021_2022_VC_E4_B8_AD_E5_88_A9_E7_94_c97_273528.htm 俗话说佛要金装、人要衣装，作软件的当然得要个好界面啦。网上提供的控件自绘基本上是MFC或WTL封装好的类，对于不想用MFC的人来说是一无是处的，我可是WIN32API的坚决拥护者。因为MFC等也是用WIN32API封装起来的，学好了WIN32API，可以深入的了解Windows内部的机制，编写出来的程序才能得到更好的优化。 下面分析一下自绘按钮的原理，用过MFC自绘按钮的人都知道，是通过重载了父窗口WM_DRAWITEM的响应消息实现的。同时也要子类化按钮来得到按钮的其他有用的消息，比如WM_MOUSEMOVE、WM_KEYDOWN等消息。因为MFC的消息循环都是封装好的，所以只要派生一下基本控件类就可以了。当是用WIN32API做的话就需要自己来子类化按钮窗口的消息循环了，相信经常编程的朋友都知道，子类化控件要用到SetWindowLong来改变窗口的回调过程，然后在回调窗口内添上自己需要处理的消息即可。因为我们要实现自绘按钮所以最好把子类化的过程做成一个类，然后传给它要自绘的按钮句柄就行了。因为要在类里面实现消息回调函数，但是类里面的消息回调函数只能是静态的，所以不能对应每个实例的消息回调。在我实现的按钮子类化类里，我用到Thunk技术或SetProp函数来实现的，具体请网上查找。下面我来谈谈自绘按钮里最重要的部分，就是响应按钮消息函数里的WM_PAINT消息，我们所有的自绘动作都在这里进行的。WM_PAINT里的绘图操作与普通窗口的操作一样，但

是为了跟踪按钮的当前状态，我们还要响应按钮窗口的WM_MOUSEMOVE、WM_SETFOCUS、WM_KILLFOCUS、WM_LBUTTONDOWN、WM_ENABLE等消息来得到当前按钮的状态。从而在WM_PAINT里面绘出不同的状态，能实现的东西很多可以说你想多少基本就能实现多少，看个人喜好了，我提供源代码大家可以自行修改。我也是参看了ButtonST里面自绘的代码，我自己添加了右键拖动功能，鼠标掠过发生功能大家有兴趣可以自己添加，锻炼一下自己的编程能力。 下面我说一下我做的这个类的一个问题，我把按钮类做成了一个动态库，调用时只要加上我的头文件和连接的lib库就可以了。我的动态库在WIN32的程序加载是没有问题的，但是在MFC里面，必需要响应父窗口的WM_DRAWITEM消息，在里面直接返回，而不要调用MFC默认的处理就OK了。这是因为我没有截获父窗口的WM_DRAWITEM消息，否则在关闭程序时会出现非法操作！主要代码分析如下： 自绘按钮类声明：class DLLPORT CWINButton { public: //初始化按钮(这是第一步!) BOOL GetItemhWnd(HWND hWnd). //还原按钮区域设置 BOOL Restore(). //设置按钮是否可以拖动 BOOL SetDrag(BOOL Enable). //设置按钮图标 BOOL SetIcon(HICON icon). //设置按钮文字 BOOL SetText(char *text, HFONT font). BOOL SetText(char *text). BOOL SetText(char *text, COLORREF color). //设置按钮有效区域 BOOL SetupRegion(COLORREF TransColor). LRESULT OnPaint(HDC hdc). //设置按钮无效时的图片 BOOL SetDisablePic(HBITMAP bmp). //设置按钮按下时的图片 BOOL SetPressPic(HBITMAP bmp). //设置悬停按钮时

的图片 BOOL SetHovERPic(HBITMAP bmp). //设置按钮背景图片，第二个参数是是否根据图片调整按钮大小 BOOL SetBackPic(HBITMAP bmp, BOOL bReSize). //设置按钮的提示消息 BOOL SetToolTip(char *text). CWINButton(). virtual ~CWINButton(). private: static LRESULT WINAPI stdProc(HWND hWnd,UINT uMsg,UINT wParam,LONG lParam). WNDPROC GetThunk(). WNDPROC CreateThunk(). LRESULT CALLBACK WINProc(UINT message, WPARAM wParam, LPARAM lParam). BOOL DrawInsideBorder(HDC dc, RECT *rect). BOOL DrawFlat(HDC dc, RECT *rect). BOOL DrawDefault(HDC dc). HWND m_ToolTip. HWND m_hWnd. HWND m_hWndParent. LONG m_OldProc. WNDPROC m_thunk. TOOLINFO ti. HICON m_icon. HBITMAP m_Back. //按钮背景图片 HBITMAP m_Hove. //鼠标悬停时按钮背景图片 HBITMAP m_Press. //鼠标按下时按钮背景图片 HBITMAP m_Disable. //按钮无效时背景图片 BITMAP bm. COLORREF m_textcolor. //按钮文字的颜色 BOOL m_bMouseTracking. //判断鼠标是否在窗口内 BOOL m_bPress. //判断鼠标是否按下 BOOL m_Enable. //控件是否有效 BOOL m_bFocus. //按钮是否处于输入焦点 BOOL m_bOwnerDraw. //判断是否用户自己贴图 BOOL m_bDrag. //是否处于拖动状态 BOOL m_bDragEnable. //是否允许拖动 char m_text[MAX_TEXTLEN]. //按钮文字 char m_tiptext[MAX_TEXTLEN]. //按钮提示文字 HFONT m_font. //按钮文字字体 HCURSOR m_OldCursor. RECT m_ParentRt. RECT m_BeginRt. RECT m_CurrentRt. POINT m_BeginPt. POINT m_CurrentPt. int m_CaptionHeight. int m_BorderWidth.

int m_EdgeWidth. protected: //按钮的外边框 HPEN m_BoundryPen. //鼠标指针置于按钮之上时按钮的内边框 HPEN m_InsideBoundryPenLeft. HPEN m_InsideBoundryPenRight. HPEN m_InsideBoundryPenTop. HPEN m_InsideBoundryPenBottom. //按钮获得焦点时按钮的内边框 HPEN m_InsideBoundryPenLeftSel. HPEN m_InsideBoundryPenRightSel. HPEN m_InsideBoundryPenTopSel. HPEN m_InsideBoundryPenBottomSel. //按钮的底色，包括有效和无效两种状态 HBRUSH m_FillActive. HBRUSH m_FillInactive.}. 消息回调类里的实现代码：CWINButton::GetItemhWnd（）里面if(SetProp(m_hWnd, "CWINBUTTON", (HANDLE)this) == 0){ OutputDebugString("SetProp ERROR"). return FALSE.}m_OldProc = SetWindowLong(m_hWnd,GWL_WNDPROC,(LONG)stdProc). CWINButton::stdProc（）里面{ CWINButton* w = (CWINButton*)GetProp(hWnd, "CWINBUTTON"). return w->WINProc(uMsg,wParam,lParam).} Thunk 代码可看我的代码或者去网上查询。