DOI:10.16184/j.cnki.comprg.2011.19.004

基于 Windows SDK 的消息反射机制的实现

于炳林

摘　要：分析了消息反射机制的原理和用途，介绍了基于 Windows SDK 的消息反射机制的设计思路和实现方法。

关键词：消息反射；窗口子类化

1　前言

在 Windows 中，控件产生的消息有两类，一类由控件自身处理，另一类则发送到父窗口由父窗口处理。这样可以增强控件的通用性和灵活性，使用户可以在父窗口中了解控件的状态变化，并控制控件的某些行为。但由此也带来了控件自身能动性的降低，不利于控件的封装和重用。

最典型的是 Edit、Button、ListBox、Static 等控件，在绘制操作前会向父窗口发送一个 WM_CTLCOLORXXXX 消息，要求父窗口为其提供一个画刷（Brush）。这样，控件的绘制颜色就完全依赖于父窗口，而控件自己却不能设置自身的颜色。

为了解决类似的问题，MFC4.0 使用了消息反射的机制，即将控件发送到父窗口的某些消息返回给控件，由控件自己进行处理。然而，这一技术目前仅限于 MFC，在 Windows SDK 开发中极少使用。以增加 Edit 控件的颜色设置为例，介绍在 Windows SDK 中实现消息反射机制的方法。

2　设计思路

消息反射，顾名思义就是将控件发送到父窗口的某些消息反射回控件，由控件自行处理。由此，要实现消息反射，必须考虑两个方面的因素：（1）控件发送到父窗口消息的截获和反射；（2）控件对反射回来的消息的处理。

截获和反射控件发送给父窗口的消息，不能在父窗口的窗口过程中进行，否则不仅不利于控件的封装和重用，而且使消息反射机制失去意义。由于控件总是建立在父窗口的基础上，因此可以对其父窗口子类化，以其子类化的窗口过程实现对消息的截获、筛选和反射。

由于控件都是进行了封装的，其窗口过程是无法更改的，反射回来的消息是不能直接通过控件的窗口过程进行处理的。因此必须对控件进行子类化，在其子类化的窗口过程中处理反射回来的消息。

3　实现方法

下面以为 Edit 控件增加颜色设置，实现 Edit 控件扩展为例，介绍基于 Windows SDK 消息反射机制的实现（原码为 Delphi 7.0）。

3.1　示例代码

```
unit EditEx;
interface
uses
  Windows, Messages;
const
  EM_SETCOLOR = WM_USER + 1;
// 颜色设置消息
function CreateEdit (hParent: HWND; dwStyle: DWORD; dwLeft, dwTop, dwWidth, dwHeight: DWORD): HWND;
implementation
const
  EM_CTLCOLOREDIT = WM_USER + WM_CTLCOLOREDIT; // 反射消息
var
  fnDefEditProc: TFNWndProc = nil;
// Edit 控件默认过程指针
  fnDefParentProc: TFNWndProc = nil;
// 父窗口默认过程指针
  dwTextColor: Integer = 0;
// 文本颜色
  dwBkColor: Integer  = $0FFFFFF;
// 背景颜色
function ParentWindowProc (hWnd: HWND; uMsg: UINT; wParam: WPARAM;
  lParam: LPARAM): LRESULT; stdcall;
// 父窗口子类化过程
begin
  if uMsg = WM_CTLCOLOREDIT then
    Result := SendMessageW (lParam, EM_CTLCOLOREDIT, wParam, lParam)
  else Result := CallWindowProcW (fnDefParentProc, hWnd, uMsg, wParam, lParam);
end;
function EditWindowProc (hEditWnd: HWND; uMsg: UINT; wParam: WPARAM;
```

```
        lParam: LPARAM): LRESULT; stdcall;
// Edit 控件子类化过程
        begin
          Result := 1;
          case uMsg of
            EM_CTLCOLOREDIT:
           begin
             SetTextColor(wParam, dwTextColor);
             SetBkColor(wParam, dwBkColor);
             Result := CreateSolidBrush(dwBkColor);
           end;
             EM_SETCOLOR:
             begin
          if (wParam >= 0)and(wParam <> dwTextColor) then
               begin
                 dwTextColor := wParam;
         Result := Integer(InvalidateRect(hEditWnd, nil, True));
               end;
               if (lParam >= 0)and(lParam <> dwBkColor) then
               begin
                 dwBkColor := lParam;
        Result := Integer(InvalidateRect(hEditWnd, nil, True));
               end;
             end;
              else Result := CallWindowProcW(fnDefEditProc,
hEditWnd, uMsg, wParam, lParam);
        end;
         end;
          function CreateEdit (hParent: HWND; dwStyle:
DWORD; dwLeft, dwTop, dwWidth,
          dwHeight: DWORD): HWND;
        begin
          Result := CreateWindowExW (0, 'Edit', nil, dwStyle,
dwLeft, dwTop, dwWidth,
            dwHeight, hParent, 0, HInstance, nil);
          SendMessageW(Result, EM_LIMITTEXT, 0, 0);
           fnDefEditProc := TFNWndProc (SetWindowLongW
(Result, GWL_WNDPROC,
            Integer(@EditWindowProc)));
          if fnDefParentProc = nil then
          begin
            fnDefParentProc := TFNWndProc (SetWindow-
LongW(hParent, GWL_WNDPROC,
              Integer(@ParentWindowProc)));
          end;
          SetFocus(Result);
        end;
        end.
```

3.2 原理分析

上述示例代码实现了对 WM_CTLCOLOREDIT 消息的反射，从而使用户可以在父窗口中通过向 Edit 控件发送 EM_SETCOLOR 消息设置 Edit 的文本颜色和背景色，具体实现过程如下：

ParentWindowProc 为父窗口的子类化过程，其将代替父窗口的默认窗口过程。在其中可以对发送到父窗口的消息进行筛选，得到 WM_CTLCOLOREDIT 消息后，将其转换为 EM_CTLCOLOREDIT 消息（该消息为内部消息，在父窗口中是不可见的。其参数与 WM_CTLCOLOREDIT 相同），并发送回 Edit 控件，其他消息通过 CallWindowProcW 调用父窗口的默认窗口过程，仍然交给父窗口的默认窗口过程进行处理，这样就可以将 Edit 控件发送给父窗口的消息重新反射回 Edit 控件，实现了消息反射功能。

EditWindowProc 为 Edit 控件窗口的子类化过程，其将代替 Edit 控件的默认窗口过程。在其中，既可以处理反射消息，也可以处理用户的自定义消息或其他需要功能变化的消息。其他无功能变化的消息通过 CallWindowProc 调用 Edit 控件的默认过程，由 Edit 控件的默认过程进行处理。在这里，除对反射消息 EM_CTLCOLOREDIT 进行了处理外，还定义并处理了一个 EM_SETCOLOR 消息（该消息为外部消息，在父窗口中是可见的。其 wParam 参数为文本颜色值，lParam 参数为背景颜色值），用户可以在父窗口向 Edit 控件发送该消息改变 Edit 控件的文本颜色和背景颜色。

CreateEdit 为 Edit 控件的创建函数，也是实现窗口子类化、完成消息反射的关键函数。在其中，除完成 Edit 控件的创建外，还完成了父窗口和 Edit 控件窗口的窗口过程替换，实现了父窗口和 Edit 控件窗口的子类化，建立了消息反射机制。

3.3 应用示例

上述示例代码以 Delphi 的单元呈现。用户既可以作为 Delphi 程序的一个单元直接使用，也可以将其转换为 DLL 程序封装为 Windows 的标准控件，在其他应用中调用。下面将其作为 Delphi 的程序单元，示例其应用方法。

```
program SimpleNote;
{$R 'SimpleNote.res' 'SimpleNote.rc'}
uses
  Windows,
  Messages,
  EditEx in 'EditEx.pas';
const
  IDM_NEW       = 1001;
  IDM_OPEN      = 1002;
  IDM_SAVE      = 1003;
  IDM_SAVEAS    = 1004;
  IDM_EXIT      = 1005;
  IDM_UNDO      = 2001;
  IDM_COPY      = 2002;
  IDM_CUT       = 2003;
  IDM_PASTE     = 2004;
  IDM_DELETE    = 2005;
  IDM_SELECTALL = 2006;
  IDM_DELETELINE = 2007;
  IDM_FONT      = 3001;
```

```
  IDM_COLOR      = 3002;
  IDM_WORDWRAP   = 3003;
  IDM_ABOUT      = 4001;
var
  msg: TMSG;
  hMainWnd, hEditWnd: HWND;
  clsMainWnd: WNDCLASSEXW;
  cxScreen, cyScreen: Word;
  wLeft, wTop, wWidth, wHeight: Word;
function MainWindowProc(hMainWnd: HWND; uMsg:
UINT; wParam: WPARAM;
  lParam: LPARAM): LRESULT; stdcall;
var
  rc: TRect;
begin
  Result := 1;
  case uMsg of
    WM_CREATE:
    begin
      GetClientRect(hMainWnd, rc);
  hEditWnd : = CreateEdit (hMainWnd, WS_CHILD or
WS_VISIBLE or WS_HSCROLL or WS_VSCROLL or
WS_BORDER or ES_MULTILINE or ES_AUTOVSCROLL or
    ES_AUTOHSCROLL or ES_WANTRETURN or
ES_LEFT, rc.Left, rc.Top, rc.Right,  rc.Bottom);
    end;
    WM_COMMAND:
    begin
      case LoWord(wParam) of
        IDM_NEW,
        IDM_OPEN,
        IDM_SAVE,
        IDM_SAVEAS,
        IDM_UNDO,
        IDM_COPY,
        IDM_CUT,
        IDM_PASTE,
        IDM_DELETE,
        IDM_SELECTALL,
        IDM_DELETELINE,
        IDM_FONT,
        IDM_WORDWRAP,
        IDM_ABOUT:
        begin
        end;
        IDM_EXIT:
        begin
    SendMessageW(hMainWnd, WM_DESTROY, 0, 0);
        end;
        IDM_COLOR:
        begin
          // 将文本颜色设置为红色,背景颜色设置为绿色
     SendMessageW  (hEditW nd,  EM_SETCOLOR,  RGB
(255, 0, 0), RGB(0, 255, 0));
        end;
      end;
    end;
    WM_CLOSE:
    begin
  SendMessageW(hMainWnd, WM_DESTROY, 0, 0);
    end;
    WM_DESTROY:
    begin
  if hEditWnd <> 0 then DestroyWindow(hEditWnd);
      PostQuitMessage(0);
    end;
  else Result := DefWindowProcW(hMainWnd, uMsg, wParam,
lParam);
  end;
end;
begin
  clsMainWnd.cbSize := Sizeof(clsMainWnd);
 clsMainWnd.style := CS_VREDRAW or CS_HREDRAW;
  clsMainWnd.lpfnWndProc := @MainWindowProc;
  clsMainWnd.cbClsExtra := 0;
  clsMainWnd.cbWndExtra := SizeOf(Longint);
  clsMainWnd.hInstance := HInstance;
 clsMainWnd.hIcon := LoadIcon(HInstance, ´SimpleNote´);
  clsMainWnd.hCursor := LoadCursor(0, IDC_ARROW);
   clsMainWnd.hbrBackground  : =  CreateSolidBrush
(RGB(236, 233, 216));
 clsMainWnd.lpszMenuName := ´SimpleNote´;
 clsMainWnd.lpszClassName := ´MainWindowClass´;
 clsMainWnd.hIconSm := LoadIcon(HInstance, ´SimpleNote´);;
  if RegisterClassExW(clsMainWnd) <> 0 then
  begin
    cxScreen := GetSystemMetrics(SM_CXSCREEN);
    cyScreen := GetSystemMetrics(SM_CYSCREEN);
    wWidth := cxScreen*680 div 1000;
    wHeight := wWidth*680 div 1000;
    wLeft := (cxScreen – wWidth)div 2;
    wTop := (cyScreen – wHeight)div 2;
    hMainWnd := CreateWindowExW  (0, clsMainWnd.
lpszClassName,  ´简易记事本´,
  WS_OVERLAPPEDWINDOW,  wLeft,  wTop,  wWidth,
wHeight, 0, 0, HInstance, nil);
    if hMainWnd <> 0 then
    begin
      ShowWindow(hMainWnd, SW_SHOWNORMAL);
      UpdateWindow(hMainWnd);
    end;
  end;
  while GetMessageW(msg, 0, 0, 0) do
  begin
    TranslateMessage(msg);
    DispatchMessageW(msg);
  end;
```

（下转第 19 页）

知道了想要的结果，现在来看一看完成上述任务的 Canvas 类的代码。

```
public class BrowseImageCanvas extends Canvas {
  private boolean scroll = false;
  private int currentImage = 0;
  private Image[] images;  // 存放多个图片,相当于图片库
  private int pressX, releaseX, dragX = 0;
  public BrowseImageCanvas() {
    // 启动时加载图片,这里加载 3 个图片
    images = new Image[3];
    images[0] = Image.createImage("/flower0.jpg");
    images[1] = Image.createImage("/flower1.jpg");
    images[2] = Image.createImage("/flower2.jpg");
  }
  protected void paint(Graphics g) {
    g.setGrayScale(255);
    g.fillRect(0, 0, getWidth(), getHeight());  // 清屏
    if (scroll) {     // 滚动当前图片
      g.drawImage(images[currentImage], -dragX, 0,
        Graphics.LEFT | Graphics.TOP);
      scroll = false;
      return;
    }
    if (pressX < releaseX) { // 向右滑动,显示下一幅图片
      currentImage++;
   if (currentImage == images.length)  currentImage = 0;
    }
    if (pressX > releaseX) { // 向左滑动,显示前一幅图片
      currentImage--;
  if (currentImage < 0)  currentImage = (images.length - 1);
    }
    // 绘制新的图片
    g.drawImage (images [currentImage], 0, 0, Graphics.
LEFT | Graphics.TOP);
  }
  protected void pointerPressed(int x, int y) {        // 记下
//开始触摸的水平位置
    pressX = x;
  }
  // 记下触摸释放的水平位置,判断滑动的距离决定是否加
//载新图片
  protected void pointerReleased(int x, int y) {
    if (scroll)  return;
    releaseX = x;
    if (Math.abs(releaseX - pressX) > 20) {  // 滑动距离超
//过 20 个点加载新图片
      repaint();
    }
  }
  protected void pointerDragged(int x, int y) {     //在这里
//决定是否滚动图片
    int deltaX = pressX - x;
    if (Math.abs(deltaX) <= 20) {  // 滑动距离少于 20 个点
//滚动当前图片
      int imageWidth = images[currentImage].getWidth();
      if (imageWidth > getWidth()) {
        dragX += deltaX;
        if (dragX < 0)  dragX = 0;
        else if (dragX + getWidth() > imageWidth)  dragX
= imageWidth - getWidth();
      }
      scroll = true;
      repaint();
    }
  }
}
```

程序中，pointerPressed ()、pointerReleased ()、pointerDragged () 方法设置各种参数，paint () 利用这些参数来决定是否滚动当前图片或显示新图片。在 pointerPressed () 方法中，保存了用户触摸位置的 x 轴坐标，然后，在 pointerReleased () 和 pointerDragged () 方法中确定用户手指沿水平方向移动的距离大小（deltaX）。如果移动超过 20 点，则显示新图片，否则就沿 x 轴方向卷动图片。可以左右两个方向移动。当然，为了简化问题，程序忽略了沿 y 轴方向的移动。

3 结语

自 Java ME 诞生以来，它就已经具备了满足触摸屏界面的能力，只是那些提供 Java ME 实现的设备制造商没有跟上步伐。介绍了 Canvas 类中的能够捕获触摸屏上指点运动的接口，并以一个类似于 iPhone 手机上的图片浏览程序来演示其使用方法。希望能够起到抛砖引玉的作用，启发读者写出更加实用的触摸屏 Java 程序。

（收稿日期：2011-07-23）

（上接第 17 页）

```
    ExitCode := msg.wParam;
  end.
```

上述代码中，在颜色设置菜单项下，向 Edit 控件发送 EM_SETCOLOR 消息，将 Edit 控件的文本颜色设置为红色，背景颜色设置为绿色。

4 结语

消息反射机制不仅可以提高控件自身的能动性，而且有利于控件的封装和重用，特别适合于对 Windows 标准控件的扩展。

（收稿日期：2011-07-09）