第 3 章

Delphi 中的消息机制

本章导学

在本章中将学习：

- 消息的基本原理
  - Windows 的消息机制
  - Delphi 的消息机制
- Delphi 中的消息处理
  - TApplication 对象的 OnMessage 事件
  - 消息处理过程
  - WndProc 过程的覆盖
  - Delphi 的事件机制
- 发送消息
  - Delphi 的消息发送机制
  - 同一程序窗体间的消息发送
  - 不同应用程序间的消息发送
- 独立实践

3.1 消息的基本原理

Windows 是一个消息驱动的操作系统，消息传递和处理是实现对象间通信和控制的主要手段，理解消息机制是进行 Windows 编程的重要基础之一。Delphi 的 VCL 组件库中已经以事件的形式封装了 Windows 的绝大部分消息，使用更加方便。但是如果要编写新的组件或更加灵活地运用现有组件以及截获、过滤消息，使程序功能更加强大，就必须深刻理解消息处理机制。

3.1.1 Windows 的消息机制

在 Windows 操作系统中，消息即是 Windows 发出的一个通知，它告诉应用程序某一个事件发生了（如鼠标单击、键盘按下、窗口尺寸改变等），然后相关的对象就去执行相应的操作，也就是消息处理。

消息是作为一个记录传递给应用程序的，在 Delphi 的 Windows 单元中封装了这个名称为 TMsg 的记录，它包含了消息的类型标识以及其他一些附加信息，声明如下：

```
TMsg = tagMSG;
tagMSG = packed record
    hwnd: HWND;          //窗口句柄
    message: UINT;       //消息常量
    wParam: WPARAM;      //消息附加信息
    lParam: LPARAM;      //消息附加信息
    time: DWORD;         //消息创建的时间
    pt: TPoint;          //消息创建时的鼠标位置坐标
end;
```

Windows 中定义了很多的消息常量，主要分为标准消息、通知消息和用户自定义消息。标准消息都以 WM 作为前缀，与发生在 Windows 中的标准活动相对应。例如，WM_Close 表示窗口关闭消息，WM_LBUTTONDOWN 表示鼠标左键按下的消息。自定义消息可以方便程序员进行编程，自定义消息的数值一般为 WM_USER + xxx 和 WM_APP + xxx。消息常量 WM_USER 和 WM_APP 是 Windows 预定义的，值分别为 0x0400 和 0x8000，这两个常量主要是为了防止自定义消息常量和系统已经定义的消息常量发生冲突。

Windows 的消息系统由以下 3 个部分组成。

(1) 消息队列。Windows 为每个应用程序维护一个消息队列，发生的消息首先被放入消息队列。应用程序从消息队列中获取消息，然后分派给相应的窗口去处理。

(2) 消息循环。通过消息循环机制，应用程序周而复始地从消息队列中获取消息，并分派给相应的窗口。

(3) 窗口过程。每一个窗口都有一个窗口过程，接收并处理分派给该窗口的消息。它是一个回调函数，负责获取传递来的消息并处理消息。

Windows 操作系统中的消息从产生到被处理一般需要 5 个步骤。

(1) 系统中发生一个事件；

(2) Windows 将事件翻译成相应的消息，放入消息队列中；

(3) 应用程序从消息队列中获取消息并封装在 TMsg 记录中；

(4) 应用程序通过消息循环把消息分派给对应的窗口过程；

(5) 窗口过程处理响应该消息并进行处理。

3.1.2 Delphi 的消息机制

在 Delphi 中，Windows 消息被封装在 Messages 单元的 TMessage 记录类型中，其声明如下：

```
TMessage = packed record
    Msg: Cardinal;     //消息常量标识
    case Integer of
      0: (
        WParam: Longint;
        LParam: Longint;
        Result: Longint);
```

```
        1: (
          WParamLo: Word;
          WParamHi: Word;
          LParamLo: Word;
          LParamHi: Word;
          ResultLo: Word;
          ResultHi: Word);
end;
```

TMessage 类型是一个通用的数据结构,适用于所有的消息。但是为了简化程序设计,Delphi 还为每一个 Windows 消息定义了一个特定的消息记录类型,这样,就无须从 WParam 和 LParam 中去分解数据。例如,鼠标消息 WM_MOUSE 对应的消息记录为 TWMMouse,在 Messages 单元中定义如下:

```
TWMMouse = packed record
    Msg: Cardinal;
    Keys: Longint;
    case Integer of
      0: (
        XPos: Smallint;
        YPos: Smallint);
      1: (
        Pos: TSmallPoint;
        Result: Longint);
end;
```

我们可以看到 TWMMouse 中已经包含了鼠标位置信息,无须再去分解得到,显然这给编程带来了更多的便利。

在传统的 Windows 程序设计中,程序员需要自己编写消息循环代码,并且用一个庞大的分支结构处理每一个不同的消息。而在 Delphi 中,Windows 消息系统已经被 VCL 封装成事件和方法,程序员无须编写这些冗长而繁杂的代码。在 VCL 中,每个消息处理过程只负责处理一条消息。

3.2 Delphi 中的消息处理

在 VCL 内部,消息处理流程是非常复杂的。对于要处理的消息,在其流通过程中我们在很多地方可以截获和处理它。本章我们介绍 4 种常用的消息处理机制。

3.2.1 TApplication 的 OnMessage 事件

在 Delphi 应用程序中,任何窗口接收到一个 Windows 消息都会触发一次 TApplication 的 OnMessage 事件,所以,我们可以通过响应 TApplication 的 OnMessage 事件来捕获并处理任何发给本程序的消息。

OnMessage 事件的处理过程的原型为:

```
procedure  (var Msg:TMsg; var Handled:boolean) of object;
```

这个过程类型有两个参数，其中参数 Msg 表示的是被截获的消息记录，而参数 Handled 则用来表示消息是否已经处理完成。如果在程序中将 Handled 设为 true，则表示该消息已经处理完毕，后续消息处理机制不再处理该消息。反之，如果设为 false，则后续消息处理机制还要处理该消息。

有一点需要注意，OnMessage 事件只会处理发送到应用程序消息队列中的消息，而通过 API 函数 SendMessage 直接发送的消息不会被 OnMessage 处理。

利用 OnMessage 事件处理消息的步骤是：①依据 OnMessage 事件过程原型声明一个自定义的事件处理过程；②将 TApplication 的 OnMessage 事件指向该自定义事件处理过程；③在自定义事件处理过程中截获并处理所需的消息。

下面通过一个实例来说明 OnMessage 事件处理消息的过程。实例中，要处理的是鼠标左键按下的消息，其消息常量为 WM_LBUTTONDOWN。

实例 **3-1**　TApplication 的 OnMessage 事件处理消息。

(1) 创建一个工程 PMessage. dpr，将主窗体单元命名为 UMessage. pas，将主窗体 Name 属性设为 frmMain。

(2) 在窗体上放置 1 个 Memo 组件，Name 属性命名为 memDisplay，如图 3-1 所示。

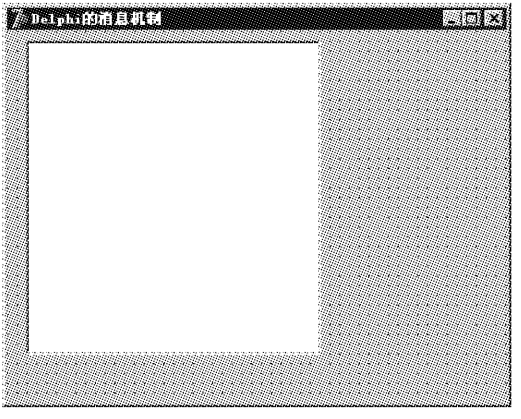

图 3-1　实例 3-1 的窗体设计界面

(3) 在窗体类的声明中加入自定义事件处理过程的声明：

```
type
  TfrmMain = class(TForm)
    memDisplay: TMemo;
    procedure AppMessage(var Msg:TMsg;var Handled:boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

(4) 按 Ctrl + Shift + C 键生成事件处理框架，加入截获并处理的 WM_LBUTTONDOWN 消息的代码：

```
procedure TfrmMain. AppMessage( var Msg: TMsg; var Handled: boolean);
begin
  if Msg. message = WM_LBUTTONDOWN then
    memDisplay. Lines. Add('Application 的 OnMessage 事件处理消息。');
  Handled: = false;
end;
```

注意,Handled 参数设为 false,表示后续消息处理机制会继续处理该消息。

(5) 编写窗体的 OnCreate 事件过程,将 OnMessage 事件的处理指向自定义的事件处理过程:

```
procedure TfrmMain. FormCreate( Sender: TObject);
begin
  Application. OnMessage: = AppMessage;
end;
```

(6) 保存并运行程序,在窗体上按下鼠标左键,结果如图 3-2 所示。

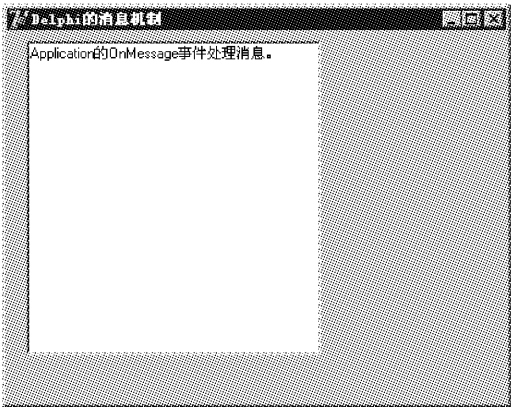

图 3-2 实例 3-1 的运行结果

3.2.2 消息处理过程

前面我们曾经提到过,Delphi 的每一个消息处理过程只处理一个特定的消息。Delphi 提供了一种消息映射机制,将特定的 Windows 消息与对应的消息处理过程联系起来,当窗口捕捉到这个消息的时候,就会自动调用对应的消息处理过程。

我们扩充实例 3-1,加入消息处理过程对 WM_LBUTTONDOWN 消息的处理。

实例 **3-2** 消息处理过程处理消息。

(1) 打开实例 3-1 的工程 PMessage. dpr,在窗体类中加入自定义的消息处理过程声明。在过程声明的后面,必须用关键字 message 指明该过程需要处理的消息:

```
type
  TfrmMain = class(TForm)
    memDisplay: TMemo;
    procedure AppMessage(var Msg:TMsg;var Handled:boolean);
    procedure FormCreate(Sender: TObject);
```

```
    procedure MyMessage(var Msg:TWMMouse);message WM_LBUTTONDOWN;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

(2) 按 Ctrl + Shift + C 键生成事件处理框架,并加入处理 WM_LBUTTONDOWN 消息的代码:

```
procedure TfrmMain. MyMessage(var Msg: TWMMouse);
begin
  memDisplay. Lines. Add('消息处理函数处理消息。');
  inherited;
end;
```

注意,在处理过程的最后加上语句 Inherited,表示还要调用系统对该消息的默认处理。

(3) 保存并运行程序,在窗体上按下鼠标左键,发现以上两种消息处理机制都处理了该消息,结果如图 3-3 所示。

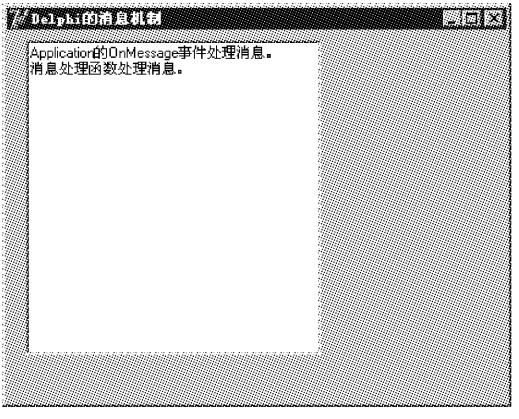

图 3-3 实例 3-2 的运行结果

3.2.3 WndProc 过程的覆盖

WndProc 过程是 VCL 类库中 TControl 类中定义的一个虚拟方法,可以被覆盖以提供自定义的消息处理代码。WndProc 过程是在消息分派之前调用的,所以我们可以覆盖该过程得到一个在消息分派之前过滤和处理消息的机会。

在 WndProc 过程中,可以对自己关心的消息进行截获和处理,而对自己不关心的消息则应继承父类的默认处理。WndProc 过程会调用 Dispatch 过程进行消息的分派。

覆盖 WndProc 过程的格式如下(注意后面的 override 关键字):

```
procedure WndProc(var Mess:TMessage);override;
```

我们继续扩充实例 3-2,加入 WndProc 过程对 WM_LBUTTONDOWN 消息的处理。

实例 **3-3**　覆盖 WndProc 过程处理消息。

(1) 打开实例 3-2 的工程 PMessage. dpr,在窗体类声明中加入 WndProc 过程的覆盖声明:

```
type
  TfrmMain = class(TForm)
    memDisplay: TMemo;
    procedure AppMessage(var Msg:TMsg;var Handled:boolean);
    procedure FormCreate(Sender: TObject);
    procedure MyMessage(var Msg:TWMMouse);message WM_LBUTTONDOWN;
    procedure WndProc(var Mess:TMessage);override;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

(2) 按 Ctrl + Shift + C 键生成过程框架,加入截获并处理 WM_LBUTTONDOWN 消息的代码:

```
procedure TfrmMain. WndProc(var Mess: TMessage);
begin
  if Mess. Msg = WM_LBUTTONDOWN then
    memDisplay. Lines. Add('覆盖 WndProc 过程处理消息。');
  inherited;
end;
```

注意,过程末尾的 Inherited 表示对于其他不截获的消息由系统默认处理。

(3) 保存并运行程序,在窗体上按下鼠标左键,发现以上 3 种消息处理机制都处理了该消息,结果如图 3-4 所示。

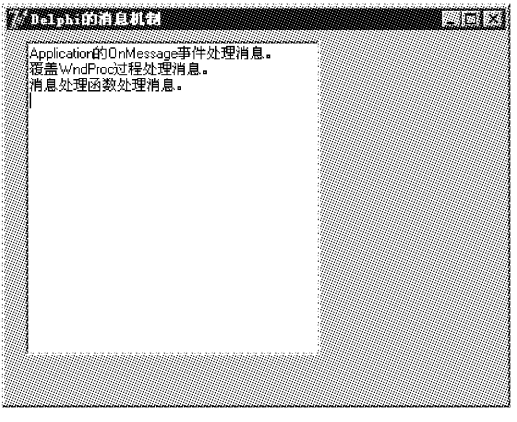

图 3-4　实例 3-3 的运行结果

3.2.4　Delphi 的事件机制

Delphi 中 VCL 的事件系统是为了使程序更好地与 Windows 消息系统接口而设计的,

它封装了大多数 Windows 消息。大多数 VCL 事件都对应一个 Windows 消息，而有些 VCL 事件则是几个消息的组合。例如，OnClick 事件就是消息 WM_LBUTTONDOWN 和消息 WM_LBUTTONUP 等的组合。由 Delphi 自动生成的事件处理过程框架一般都提供封装好的消息附加参数，因此，使用 VCL 事件比直接使用消息更为简洁。所以，一般 Delphi 编程应该尽量使用 Delphi 的事件。

下面我们用事件来处理 WM_LBUTTONDOWN 消息。我们选择的是窗体的 OnMouseDown 事件，该事件当在窗体上按下鼠标左键或右键（有的还有中键）都会发生。我们可以在该事件过程中通过代码过滤掉其他鼠标键，只考虑鼠标左键按下时的情况。

实例 **3-4**　Delphi 事件机制处理消息。

（1）打开实例 3-3 的工程 PMessage.dpr。选中窗体，在对象观察器中选择 Events 页，找到 OnMouseDown 事件所在行，在右边空白栏中双击鼠标左键，窗体单元中将自动生成该事件处理过程的声明和实现框架。

```
type
  TfrmMain = class(TForm)
    memDisplay: TMemo;
    procedure AppMessage(var Msg:TMsg;var Handled:boolean);
    procedure FormCreate(Sender: TObject);
    procedure MyMessage(var Msg:TWMMouse);message WM_LBUTTONDOWN;
    procedure WndProc(var Mess:TMessage);override;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
⋮
procedure TfrmMain.FormMouseDown(Sender: TObject; Button: TMouseButton;
                                 Shift: TShiftState; X, Y: Integer);
begin

end;
```

（2）在事件过程的实现框架中加入处理鼠标左键按下的代码：

```
procedure TfrmMain.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
    memDisplay.Lines.Add('窗体事件处理消息。');
end;
```

（3）保存并运行程序，在窗体上按下鼠标左键，发现以上 4 种消息处理机制都处理了该消息，结果如图 3-5 所示。

从运行结果中，我们可以发现 4 种消息处理机制处理消息的顺序：OnMessage 事件是

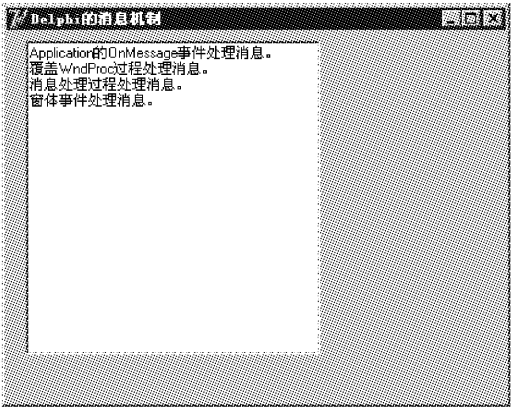

图 3-5　Delphi 事件处理消息

最早处理消息的地方;随后是覆盖的 WndProc 过程处理消息,在 WndProc 过程中会分派消息;紧接着消息处理过程得到分派的消息并处理;Delphi 的事件机制是对消息最高一级的封装,所以在最后对消息进行处理。

以上实例的完整的窗体单元 UMessage.pas 的源代码清单如下:

```
unit UMessage;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TfrmMain = class(TForm)
    memDisplay: TMemo;
    procedure AppMessage(var Msg:TMsg;var Handled:boolean);
    procedure FormCreate(Sender: TObject);
    procedure MyMessage(var Msg:TWMMouse);message WM_LBUTTONDOWN;
    procedure WndProc(var Mess:TMessage);override;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation
```

```
{$R *.dfm}

{ TfrmMain }

procedure TfrmMain.AppMessage(var Msg: TMsg; var Handled: boolean);
begin
  if Msg.message = WM_LButtonDown then
    memDisplay.Lines.Add('Application 的 OnMessage 事件处理消息。');
  Handled := false;
end;

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  Application.OnMessage := AppMessage;
end;

procedure TfrmMain.MyMessage(var Msg: TWMMouse);
begin
  memDisplay.Lines.Add('消息处理过程处理消息。');
  inherited;
end;

procedure TfrmMain.WndProc(var Mess: TMessage);
begin
  if Mess.Msg = WM_LBUTTONDOWN then
    memDisplay.Lines.Add('覆盖 WndProc 过程处理消息。');
  inherited;
end;

procedure TfrmMain.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
    memDisplay.Lines.Add('窗体事件处理消息。');
end;

end.
```

3.3 发送消息

在3.2节中，讲到了消息是由 Windows 系统发出的。其实，应用程序也可以像 Windows 一样在窗口或组件之间发送消息。Delphi 中提供了几种途径，其中最常用的是 TControl 类的 Perform 方法和 API 函数 SendMessage、PostMessage。

3.3.1 Delphi 的消息发送机制

Perform 方法适用于向所有由 TControl 类派生的对象发送消息。方法 Perform 的原型如下：