实 验 八 windows 窗体程序结构与消息处理

8.1 实验目的

.NET 平台的窗体派生自基本的 Windows 窗体，掌握 Windows 窗体消息循环程序框架，理解回调函数作用和运行机制是深入 Windows 系统平台的必需条件。

8.2 窗体程序原理

用户采用 VB，Delphi，C++ 或 C# 语言都可生成 Windows 平台的窗体程序，它们都派生自基本的 Windows 窗体。基本的 Windows 窗体对象接收用户输入，在窗体上绘制显示内容。Windows 操作系统为每个窗体对象设置一个消息队列，应用程序从消息队列中取出消息并执行相应操作或算法，图8-1描绘了 Windows 消息机制与窗体对象。

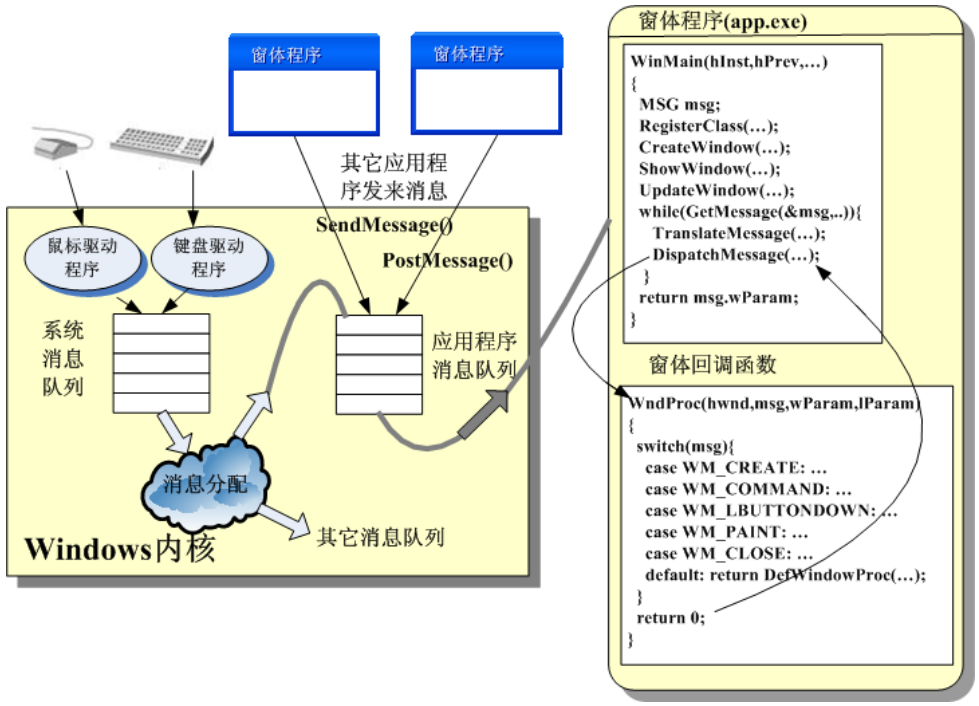

图 8-1 Windows 消息机制与窗体对象

8.3 基本窗体程序

VS12 工具支持创建基础窗体应用程序，在本实验中新建项目，选择项目类型为 Visual C++ 下的 Windows 窗体应用程序，设位置为 D:\xue，名称为 sample，如图8-2所示，进入 Win32 应用程序向导，在应用程序设置页选择应用程序类型为 Windows 应用程序，如图8-3所示，生成 C++ 语言编写的基本窗体程序。

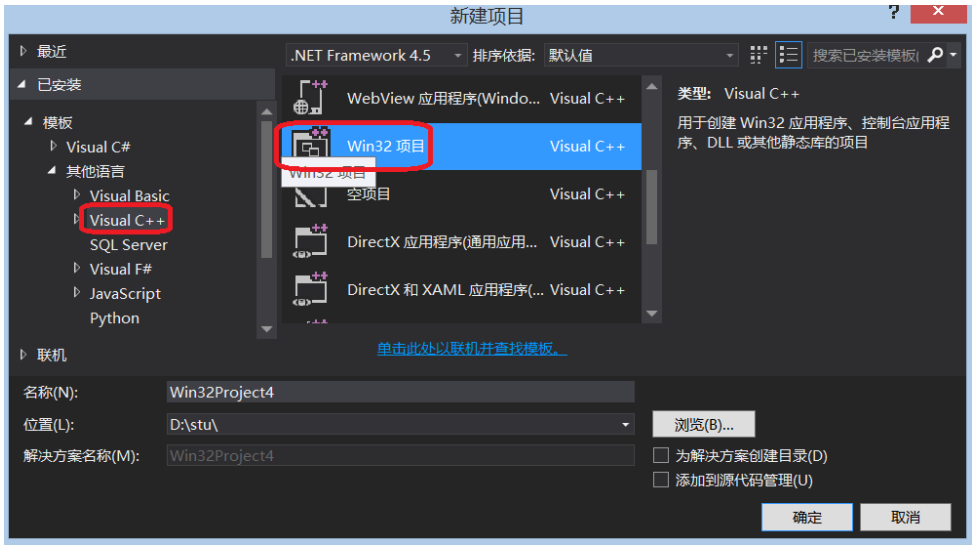

图 8-2 创建 Windows 应用程序

主程序是 sample.cpp 文件，sample.cpp 文件中的 About 函数无实际用处可直接去除，图8-4描绘的 _tWinMain 是程序入口函数并由操作系统调用。窗体程序包括几个主要部分，MyRegisterClass 函数用于窗体资源的注册，最重要的功能是设置窗体的回调函数为 WndProc，InitInstance 函数使用 CreateWindow 方法创建运行中的窗体资源，ShowWindow 和 UpdateWindow 则使窗体开始刷新显示。_tWinMain 函数中最重要的是消息循环结构，在运行期间由消息队列获取消息，根据消息值执行分支处理，来实现接收输入和结果显示。

```
// 主消息循环:
while (GetMessage(&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }
}
```

以上代码是窗体程序的基本结构，程序反复从消息队列中取出消息并调用函数 WndProc，WndProc 函数中则对消息的值进行匹配后执行相应任务，WndProc 函数函数也称为窗体消息处理回调函数。用户虽然可编译运行基本窗体程序，但是在基本的窗体程序源程序上编写应用程序工作量非常大。微软曾采用 MFC 技术生成窗体程序，由于 MFC 技术存在很多问题，现在.NET 平台使用了新的窗体封装技术，并使用 C# 等语言开发窗体程序。窗体程序中程序入

图 8-3 Windows 应用程序类型

口，窗体类注册，消息循环结构，窗体消息匹配流程大量重复性结构代码被隐藏。窗体程序的运行结构是不变的，它仍然接收消息结构，还可以重载窗体消息处理函数响应自定义消息值。

重载后主窗体消息处理函数仍然是由消息循环调用的，而不是只执行一次。

启动窗体应用程序时 OS 创建进程对象，为进程分配运行资源。启动进程程序时首先创建主窗体线程，主窗体线程再创建工作线程，每个窗体创建一个主线程，称为主窗体线程，每个具有窗体的线程会有一个线程相关消息队列，而非窗体线程在初始化时是没有消息队列的。操作系统工作线程后台方式运行不与用户发生交互，工作线程默认是没有消息队列和消息循环机制的。

然后为窗体程序采用消息循环机制负责用户输入，对收到的消息进行匹配，根据匹配结果执行相应的分支代码，并将绘制运算结果出来。windows 程序对用户的活动作出反应，提供丰富多彩的用户体验。

微软把消息分为两种，一种是系统预先定义的消息，一种是应用程序自定义消息；系统预定义消息由系统产生，代表的是机器共有的事件，比如用户按下键盘，鼠标移动等，键盘与鼠标的各种操作事件对应不同的消息值，应用程序也可以模拟发送这些系统消息到目标窗体。每个系统定义消息都具有固定值，这个常量值被定义在 windows.h 头文件中；用户自定义消息则由用户程序定义，并且由用户程序对消息作出响应完成约定的工作。系统消息的取值范围是 0x0000 到 0x03FF，用户定义的消息值取值范围是 0x0400 到 0x7FFF，用户自定义消息不能与系统消息取值相混淆。系统消息与自定义消息最主要的区别是，系统消息对每个程序的意义是一样的，而用户自定义消息代表的意义完全由用户设定，值相同的自定义消息在不同程序中可以代表不同的意义。

```
int APIENTRY _tWinMain(_In_ HINSTANCE hInstance,
                       _In_opt_ HINSTANCE hPrevInstance,
                       _In_ LPTSTR    lpCmdLine,
                       _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
```

图 8-4 tWinMain 入口函数

8.4 向指定窗体发送消息

本小节实现向特定目标窗体发送消息。使用 VS2008 工具创建一个窗体应用程序，设位置为 D:\xue，名称为 winop；窗体添加文本标签，文本框和按钮，文本框设为多行显示，每行文本值表示要查找的窗体标题文本。界面设计如图8-5。

图 8-5 窗体与消息

查找指定的窗体是通过 EnumWindows 内核函数实现，函数将枚举系统中存在的窗体，并比较窗体的字符串的值。参考代码如下：

```
//EnumWindows 函数
[DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]
public static extern int EnumWindows(EnumWndCallBack ew_call, int y);
```

添加字符串数组定义：

```
public static string[] strList;
```

在" 查找窗体" 按钮事件中添加代码：

```
strList = new string[textBox1.Lines.Length];
for (int j = 0; j < textBox1.Lines.Length; j++)
```

```
{
  strList[j] = textBox1.Lines[j];
}
EnumWindows(new EnumWndCallBack(FrmMain.EnumWindowsProc), (int)0);
```

定义回调函数的代理类

```
public delegate bool EnumWndCallBack(IntPtr hwnd, uint lParam);
```

定义窗体查找的回调函数 EnumWindowsProc

```
private static bool EnumWindowsProc(IntPtr hwnd, uint lParam)
{
  //hwnd 是被传入的窗体句柄
  StringBuilder s_b = new StringBuilder(512);
  GetWindowText(hwnd, s_b, s_b.Capacity);
  for (int i=0; i < strList.Length; i++)
  {
    // getting an instance of listbox from its handle.
    if (s_b.ToString().IndexOf(strList[i], StringComparison.OrdinalIgnoreCase) != -1)
    {
      //找到目标窗体
      dest_hwnd = hwnd;
      break;
    }
  }

  return true;
}
```

内核函数 GetWindowText 根据窗体句柄获取窗体的文本值，参考代码如下。

```
//GetWindowText 函数
[DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]
public static extern int GetWindowText(IntPtr hwnd, StringBuilder lptrString, int nMax-
Count);
```

定义目标窗体句柄变量：

```
public static IntPtr dest_hwnd;
```

上面代码即可完成查找窗体的功能，查找到的窗体句柄值记录在 dest_hwnd 变量中。

如是要使窗口关闭，只须向其发送 WM_CLOSE 消息即可，这需要用到 SendMessage 内核函数，最大化对应的消息值为 SW_SHOWMAXIMIZED，最小化对应消息值为 SW_SHOWMINIMIZED，由用户自行实现，代码从略。

```
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(
IntPtr hWnd, // handle to destination window
int Msg, // message
```

```
int wParam, // first message parameter
int lParam // second message parameter
);
```

在上述功能基础目添加适当代码让窗体能够跳舞，也就是让窗体在较短时间间隔出现在桌面不同位置，SetWindowPos 函数可以设置窗体的位置，ShowWindow 函数和 SetForegroundWindow 函数将窗体置前，参考代码如下：

```
private const int SW_SHOW = 0x005;
private const int WM_CLOSE = 0x0010;
private const int SW_SHOWMINIMIZED = 0x0002;
private const int SW_SHOWMAXIMIZED = 0x0003;
[DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]
public static extern int ShowWindow(IntPtr hwnd, int lParam);
[DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]
public static extern int SetForegroundWindow(IntPtr hwnd);
[DllImport("user32.dll")]
public static extern bool SetWindowPos(IntPtr hWnd, int hWndInsertAfter, int X, int Y, int cx, int cy, uint uFlags);
```

在" 窗体 dance" 按钮中添加启动线程的代码如下：

```
ThreadStart danceStart = new ThreadStart(windance);
Thread danceThread = new Thread(danceStart);
danceThread.IsBackground = true;
danceThread.Start();
```

线程函数的参考代码：

```
private const int HWND_TOP = 0x00;
private const int SWP_SHOWWINDOW = 0x40;
public void windance()
{
  //POINT center_Point;
  ShowWindow(dest_hwnd, SW_SHOW);
  SetForegroundWindow(dest_hwnd);
  Thread.Sleep(5000);
  for (int i = 0; i < 10; i++)
  {
    Thread.Sleep(200);
    SetWindowPos(dest_hwnd, HWND_TOP, i * 5, i*10, 300, 200, SWP_SHOWWINDOW);
  }
}
```

调试运行一下，窗体是不是在自己移动呢？mouse_event 内核函数可模拟 Windows 程序的鼠标输入，在" 鼠标模拟" 按钮中编写代码启动线程 auto_click，这部分代码同" 窗体 dance"，线程 auto_click 的参考如下：

```
public void auto_click()
{
    POINT center_Point;
    center_Point.x = Screen.PrimaryScreen.Bounds.Width / 2;
    center_Point.y = Screen.PrimaryScreen.Bounds.Height / 2;
    ShowWindow(dest_hwnd, SW_SHOW);
    SetForegroundWindow(dest_hwnd);
    //发送鼠标点击消息，一共发 3 次
    //将鼠标移动屏幕中间
    Thread.Sleep(5000);
    for (int i = 0; i < 10;i++ )
    {
        //设鼠标位置
        SetCursorPos(center_Point.x + 80 + GetTickCount() % 200,
                center_Point.y + 130 + GetTickCount() % 200);
        //右键点击一次
        mouse_event(MouseEventFlags.RightDown, 0, 0, 0, IntPtr.Zero);
        Thread.Sleep(200 + GetTickCount() % 100);
        mouse_event(MouseEventFlags.RightUp, 0, 0, 0, IntPtr.Zero);
        Thread.Sleep(500 + GetTickCount() % 400);
    }
}
```

其中用到的 POINT 结构，鼠标动作结构，SetCursorPos 函数用于设置鼠标位置，mouse_event 函数用于产生鼠标消息，和 GetTickCount 内核函数声明如下：

```
[Flags]
public enum MouseEventFlags
{
    Move = 0x0001,
    LeftDown = 0x0002,
    LeftUp = 0x0004,
    RightDown = 0x0008,
    RightUp = 0x0010,
    MiddleDown = 0x0020,
    MiddleUp = 0x0040,
    Wheel = 0x0800,
    Absolute = 0x8000
}
```

```
[DllImport("user32.dll")]
static extern bool SetCursorPos(int X, int Y);
[DllImport("user32.dll")]
static extern void mouse_event(MouseEventFlags flags, int dx, int dy,
        int data, IntPtr extraInfo);
[DllImport("kernel32.dll")]
static extern int GetTickCount();
public struct POINT
{
  public int x;
  public int y;
}
```

运行一下，是不是鼠标在自动点击呢？

8.5 作业

1. 完善本实验程序，补全所缺代码。
2. 编写一个程序实现对一个网页窗体特定区域自动点击功能。
3. 使用 WM_COPYDATA 消息进行窗体间字符串传送。