第 3 章　图形界面应用开发

3.1　基于 GDI 屏幕绘图

3.1.1　GDI+简介

GDI+（Graphics Device Interface Plus）就是图形设备接口，它提供了各种丰富的图形图像处理功能，在 Windows CE 系统中，它充当应用程序和硬件设备之间的中间层，封装与硬件交互所需的低级 API，这些 API 函数可用于绘制图形和文本。过去 GDI 是以 C 语言函数形式提供可以调用的 API，而 GDI+在很大程度上是 GDI 和应用程序之间的一层，提供了更直观、基于继承性的对象模型。它是对 GDI 作了一个面向对象的封装，是 GDI 的一个包装器。随着.NET Compact Framework 的出现，微软将 GDI+非托管的 API 进行包装，形成类库放进.NET 精简版的框架中，这种类库称为 GDI+类库，它是一种托管对象，使得开发人员很方便地完成屏幕绘图任务，同时当托管的 GDI+类对象不再使用时，可由.NET Compact Framework 中的垃圾回收器进行释放，以回收 GDI+绘图所占用的资源。当定制 Windows CE 操作系统时，加入有关.NET Compact Framework 框架的组件，这时就可以使用 GDI+提供的功能来完成有关屏幕绘图任务。

在.NET Compact Framework 中，GDI+API 封装在一组托管代码中，这些类被称为 GDI+的托管类接口，GDI+的类被组织到如表 3-1 所示的命名空间中。

表 3-1　GDI+的命名空间

命名空间	功能
System.Drawing	提供基本的图形功能，定义了用以存储基元自身信息的类、结构和存储基元绘制方式信息的类，以及实际进行绘制的类
System.Drawing.Drawing2D	提供大多数高级 2D 和矢量绘图操作的类，例如消除锯齿、几何转换和图形路径等
System.Drawing.Imaging	提供处理图像（位图、GIF 文件等）的各种类
System.Drawing.Text	提供字体和字体系列操作的类

3.1.2　设备绘图表面

在嵌入式设备上使用 GDI+进行绘图时，得到的图形可以放在两种目标环境中，一种是屏幕上窗口，另一种是内存中的位图。每一个环境都有一个绘图表面，而绘图表面包括两种：矢量表面和光栅表面。

1. 矢量表面

在矢量表面上，图元是用一种实体表示的，该实体具有特定的坐标、方向、大小等属性。

例如线条可以用实体来表示，它从一个定义好的点开始根据指定的长度指向特定的方向，如图 3-1 所示，通过矢量表面可以很方便地对图形进行描述和绘制。

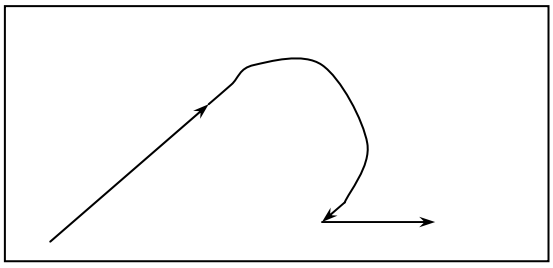

图 3-1　矢量表面线条绘制

2. 光栅表面

在光栅表面上，图元是一组着色的像素集合，如图 3-2 所示，目前所有的嵌入式设备都是基于光栅表面对图形进行显示，在开发绘制图形的应用程序中，可以先假定图形是基于矢量表面绘制的，因为矢量图形更加容易描述图像，然后根据需要采用某种机制，将矢量形式转换为光栅的形式，这种机制就是利用 GDI+的绘图技术来提取矢量图形表达式，然后把它们显示在基于光栅的绘图表面上。

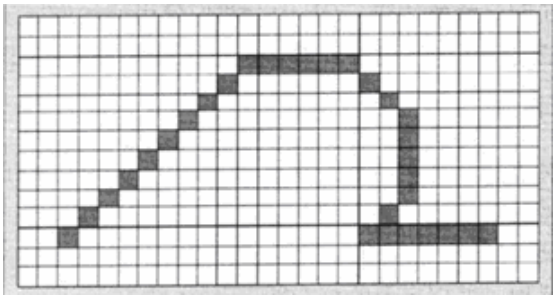

图 3-2　光栅表面的线条绘制

3. 使用 Graphics 类创建绘图表面

在 GDI+中可以利用 Graphics 类创建一个与目标环境相关的绘图表面，即产生对应的 Graphics 对象，这样就可以在该表面上进行图形组件的绘制，在.NET Compact Framework 中，有四种方法可以创建 Graphics 对象，分别如下：

（1）Graphics.FromHdc 方法。该方法用于从一个设备环境句柄中创建一个绘图表面，在编程中用这种方法创建绘图表面可以将 GDI+中的托管对象和非托管对象联系在一起。

（2）Graphics.FromImage 方法。该方法用于从内存中的一个位图对象去创建绘图表面，这样在绘图表面上绘制的图形都将被保存在位图中。

例如下面的代码：

```
Bitmap m_bitmp=new Bitmap(@"\123.bmp");
Graphics g= Graphics.FormImage(m_bitmp);
```

创建上述 Graphics 对象之后，可以调用 Graphics 对象的相应方法来绘制线条、形状或文本。

（3）使用控件类的 CreateGraphics 方法。通过这种方法创建的 Graphics 对象以及进行的

各种绘图操作，都将体现在所指定的窗体控件上，包括窗体本身。用这种方法绘制的图形一旦窗体被刷新，就需要一种重绘机制完成窗体内容的重绘。例如下面的代码：

```
Graphics g= this.CreateGraphics();//通过窗体本身创建 Graphics 对象
```

（4）在窗体或控件的 Paint 事件处理方法中创建 Graphics 对象。在.NET Compact Framework 中可以通过触发窗体的 Paint 事件，产生对应的事件处理方法去获取 Graphics 对象，PaintEventArgs 类的参数中包含图形对象的引用，开发人员只需编写相应代码即可完成图形的绘制及显示操作。

例如下面的代码：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
    {
      Graphics g=e.Graphics;
     }
```

3.1.3　绘图操作工具

1. Pen 类

Pen 类用于绘制直线或曲线对象，通过它本身的属性能够绘制具有指定宽度和样式的直线。另外可以使用 Pen 类的 DashStyle 属性绘制虚线，DashStyle 属性值是由 DashStyle 枚举定义的，具体说明如表 3-2 所示。

表 3-2　DashStyle 枚举的取值

DashStyle 属性值	说明
DashStyle.Solid	画笔绘制出的线条是直线
DashStyle.Dash	画笔绘制出的线条是虚线

例如下面创建 Pen 对象的两种方法：

```
Pen redPen=new Pen(Color.Red)          //创建一个红色，宽度默认为 1 像素的画笔
Pen redPen=new Pen(Color.Red,3)        //创建一个红色，宽度为 3 像素的画笔
```

2. Brush 类

Brush 类定义用于填充图形形状内部的对象，Brush 类是一个抽象基类，不能进行实例化处理。如果需要创建一个画刷对象，则需要 Brush 类的派生类。SolidBrush 类和 TextureBrush 类都是 Brush 类的子类，位于 System.Drawing 命名空间中，SolidBrush 类定义了单色画刷，画刷可以填充图形形状。当实心画刷难以满足要求时，还可以使用 TextureBrush 把位图当作画刷进行位图的填充。例如，下面的代码创建了一个实心画刷和位图画刷。

```
SolidBrush myBrush = new SolidBrush(Color.Blue);// 创建一个实心画刷
TextureBrush tBrush=new TextureBrush(Properties.Resources.Bitmap);// 创建一个位图画刷
```

3. Color 结构

GDI+用 System.Drawing.Color 结构来描述颜色，在.NET CF 类库中 GDI+使用 RGB 颜色，Color 结构以属性的方式定义了大量的有名称的颜色，例如 Color.Red、Color.Blue 和 Color.Pink 等。通过 Color 结构内的 FormArgb 方法可以自定义生成的颜色，通过设置 3 个值来组成创建的颜色。每种值的取值为 0～255 之间。例如，使用下面的代码创建一种颜色。

```
Color mm = Color.FormArgb(0,255,0); //绿色
```

3.1.4 常用图形的绘制

通过 GDI+可以绘制直线、矩形、椭圆、弧线、多边形和基数样条等矢量图形，Graphics 对象除了可以创建绘图表面之外，还提供了一系列方法，用来完成实际的绘图工作，绘制工作可以分为两部分，一部分是绘制线条，另一部分是填充图形。如表 3-3 所示，绘制线条可以用产生的画笔对象绘制具体形状的外形线条，而填充图形可以用画刷对象填充具体形状的封闭区域。

表 3-3 Graphics 类提供的常用绘制方法

方法名称	说明
DrawLine	绘制线条
DrawRectangle	绘制矩形
DrawPolygon	绘制多边形
DrawEllipse	绘制椭圆形
FillEllipse	填充椭圆
FillRectangle	填充矩形

1. 画直线

使用 Graphics 类的 DrawLine 方法。

格式 1 为：DrawLine(画笔,x1,y1,x2,y2)

功能：在起点坐标(x1,y1)和终点坐标(x2,y2)之间画一条直线。

格式 2 为：DrawLine(画笔,point1,point2)

功能：在点 point1 和点 point2 之间绘制一条直线。

例如下面绘制两条直线代码：

```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue ,5);//生成画笔，蓝色，5 个像素
g.DrawLine(Mypen,1,1,30,30);//画线
Point pt1=new Point(1,30);//生成起点
Point pt2=new Point(30,1);//生成终点
g.DrawLine(Mypen,pt1,pt2);//画线
```

2. 画椭圆

使用 Graphics 类的 DrawEllipse 方法。

格式 1 为：DrawEllipse (画笔,矩形结构数据)

功能：绘制一个边界由矩形结构数据定义的椭圆。

格式 2 为：DrawEllipse (画笔,x,y,width,height)

功能：绘制一个由边框（坐标、宽度、高度）定义的椭圆。

例如下面绘制一个椭圆的代码：

```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue,5);//生成画笔，蓝色，5 个像素
g.DrawEllipse(Mypen,1,1,80,40);//画椭圆
Rectangle rect=new Rectangle(10,10,160,80);//生成矩形
g.DrawEllipse (Mypen,rect);//画椭圆
```

3. 画矩形

使用 Graphics 类的 DrawRectangle 方法。

格式 1 为：DrawRectangle (画笔,矩形结构数据)

功能：绘制一个边界由矩形结构数据定义的矩形。

格式 2 为：DrawRectangle (画笔,x,y,width,height)

功能：绘制一个由左上角坐标、宽度、高度定义的矩形。

例如下面绘制一个矩形的代码：

```
Graphics g=this.CreateGraphics();//生成图形对象
Pen Mypen=new Pen(Color.Blue,2);//生成画笔，蓝色，2 个像素
g.DrawRectangle (Mypen,5,5,80,40);//画矩形
Rectangle rect=new Rectangle(85,15,140,50);//生成矩形
g.DrawRectangle (Mypen,rect);//画矩形
```

4. 画多边形

使用 Graphics 类的 DrawPolygon 方法。

格式为：DrawPolygon (画笔,Point[] points)

功能：绘制由一组 Point 结构定义的多边形。

例如下面绘制一个多边形的代码：

```
Pen blackPen = new Pen(Color.Black,3);//生成画笔，黑色，3 个像素
Graphics g=this.CreateGraphics();//生成图形对象
Point point1 = new Point(50,50);
Point point2 = new Point(70,25);
Point point3 = new Point(100,30);
Point point4 = new Point(120,85);
Point point5 = new Point(80,100);
Point[] curvePoints ={point1,point2,point3,point4,point5};//定义 Point 结构的数组
g.DrawPolygon(blackPen, curvePoints);//绘制多边形
```

5. 填充椭圆

使用 Graphics 类的 FillEllipse 方法。

格式 1 为：FillEllipse(Brush F,矩形结构数据)

功能：填充边界由矩形结构数据定义的椭圆。

格式 2 为：FillEllipse(Brush F,int x,int y,int width,int height)

功能：填充一个由边框（坐标、宽度、高度）定义的椭圆。

例如下面填充一个椭圆的代码：

```
Graphics g=this.CreateGraphics();//生成图形对象
SolidBrush BlueBrush = new SolidBrush(Color.Blue);//生成填充用的画刷
//定义外接矩形的左上角坐标、高度及宽度
int x = 0;
int y = 0;
int width = 200;
int height = 100;
Rectangle rect = new Rectangle( x, y,width, height);//定义矩形
g.FillEllipse(BlueBrush,rect);//填充椭圆
```

6. 填充矩形

使用 Graphics 类的 FillRectangle 方法。

格式 1 为：FillRectangle(Brush F,矩形结构数据)

功能：用指定的画刷填充一个矩形。

格式 2 为：FillRectangle (Brush F,int x,int y,int width,int height)

功能：填充一个由边框（坐标、宽度、高度）定义的矩形。

例如下面填充一个矩形的代码：

```
Graphics g=this.CreateGraphics();//生成图形对象
SolidBrush BlueBrush = new SolidBrush(Color.Blue);//生成填充用的画刷
int x = 15;//定义外接矩形的左上角坐标和高度及宽度
int y = 15;
int width = 200;
int height = 100;
Rectangle rect = new Rectangle( x,y,width，height);//定义矩形
g.FillRectangle(BlueBrush,rect);//填充矩形
```

3.1.5 绘制文本

在.NET Compact Framework 中，利用 GDI+库中的 Graphics 类的 DrawString 方法可以实现文本的绘制工作。

1. Graphics 类的 DrawString 方法的四个重载方法

（1）public void DrawString(string str,Font ft,Brush bh,RectangleF rf);

参数说明：str 需要绘制的字符串；ft 绘制字符串使用的字体；bh 绘制字体使用的画刷；rf 绘制字符串的矩形区域。

（2）public void DrawString(string str,Font ft,Brush bh,RectangleF rf,StringFormat sf);

这里参数 sf 代表控制整个字符串的呈现格式，其他参数含义同上所示。

（3）public void DrawString(string str,Font ft,Brush bh,float x,float y);

这里参数 x、y 代表需要绘制字符串左上角的横坐标及纵坐标位置，其他参数含义同上所示。

（4）public void DrawString(string str,Font ft,Brush bh,float x,float y,StringFormat sf);

参数含义同上所示。

2. 控制文本的格式

在上述 Graphics 类的 DrawString 重载方法中，有两个方法用到 StringFormat 类型参数，它们用来控制整个字符串的呈现样式。通过如表 3-4 所示的 StringFormat 类型的主要属性值，开发者可以通过编写代码来控制字符串格式。

表 3-4 StringFormat 类型的主要属性说明

属性	类型	说明
Alignment	StringAlignment 枚举	控制字符串的水平对齐方式
LineAlignment	StringAlignment 枚举	控制字符串的垂直对齐方式
FormatFlags	StringAlignment 枚举	控制字符串的排列

字符串在水平和垂直方向上都是通过 StringAlignment 枚举值控制对齐方式的，具体的取

值可以参见表 3-5 所示。

表 3-5　StringAlignment 枚举值

枚举值	说明
Center	绘制区居中对齐
Far	对于水平对齐，这表示右对齐；对于垂直对齐，表示底端对齐
Near	对于水平对齐，这表示左对齐；对于垂直对齐，表示顶端对齐

例如以下实例代码：

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Font ft = new Font("Arial", 12.0f, FontStyle.Bold);
    SolidBrush sbh = new SolidBrush(Color.Blue);
    StringFormat sf = new StringFormat();
    sf.Alignment = StringAlignment.Near;
    e.Graphics.DrawString("WinCE 编程_水平方向_Near",ft,sbh,
                          new RectangleF(0, 0, 320, 240), sf);
    sf.Alignment = StringAlignment.Far;
    e.Graphics.DrawString("WinCE 编程_水平方向_Far",ft, sbh,
                          new RectangleF(0,30,320,210), sf);
    sf.Alignment = StringAlignment.Center;
    e.Graphics.DrawString("WinCE 编程_水平方向_Center",ft, sbh,
                          new RectangleF(0,60,320,180),sf);
    sf.Alignment = StringAlignment.Far;
    sf.LineAlignment = StringAlignment.Center;
    e.Graphics.DrawString("WINCE 编程_垂直_Center_水平_Far",ft,sbh,
                          new RectangleF(0,0,320,240),sf);
}
```

运行效果如图 3-3 所示。

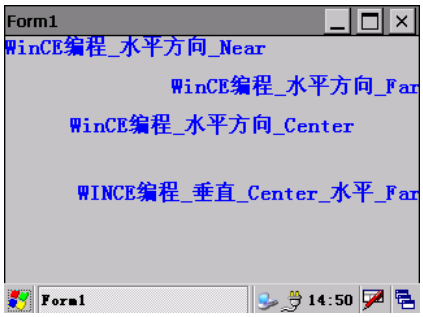

图 3-3　文字格式的控制

3.1.6　绘制图像

1. 用屏幕作绘图表面进行绘图

GDI+使用 Bitmap 类表示一个位图，这个类是抽象类 Image 的子类，尽管 Bitmap 类表示

的是一个位图，并且在内存中也是使用位图的格式来存储图片的，但 Bitmap 类可以支持 bmp、jpeg、gif 等常见的图像格式文件。

在绘制图像时，可以使用 Bitmap 类对象指定为 Image 类型的图像，并作为参数，然后调用 Graphics 类的对象的 DrawImage 方法进行绘制。DrawImage 方法为可重载方法，它主要用来在指定位置绘制指定的 Image 图像，其常用格式有以下两种。

（1）在指定的位置按原始大小绘制指定的 Image 图像。

方法为：public void DrawImage (Image image，int x，int y)

参数说明：image 表示要绘制的 Image 图像；x 和 y 表示所绘制图像的左上角的坐标。

例如下面的代码表示在屏幕上绘制一个位图。

```
private void MainForm_Paint(object sender,EventArgs e)
{
Bitmap bmp=new Bitmap("图片文件.gif");
e.Graphics.DrawImage(bmp,0,0);
}
```

（2）对 Image 图像进行缩放。

GDI+在绘制位图的过程中，还具有对位图进行缩放的功能。

方法：public void DrawImage (Image image,Rectangle destRect,Rectangle srcRect,GraphicsUnit srcUnit)

DrawImage 方法中各参数说明如下：Image 表示要绘制的位图；srcRect 表示 Rectangle 结构，指定 image 图像中规定的矩形区域显示到绘制表面上；destRect 表示指定绘图表面上哪一块矩形区域中显示 srcRect 矩形结构中的位图；srcUnit 表示 GraphicsUnit 枚举类型，用来指定 srcRect 和 destRect 矩形区域的单位，对于.NET CF 中只能取值为 Pixel。

2. 用内存中位图作绘图表面

在从一个位图创建出来的绘图表面上进行的绘图操作不会反映到屏幕上，如果希望看到绘图结果，只需使用前面绘制位图的方法将这个位图绘制出来即可。这样使用的优势就是当所需的图像包含大量复杂的图形，在内存中的 BitMap 上绘图比直接在屏幕上绘图要快很多，因此可以在一个 Bitmap 绘图表面上先完成所有的绘图操作，然后在必要的时候将这个位图直接显示到屏幕上。

例如下面 MyDrawBitMap 方法表示用指定大小的内存位图创建一个绘图表面：

```
private void    MyDrawBitMap()//
{
        Bitmap m_bmp=new Bitmap(width,height);
        Graphics g= Graphics .FromImage(m_bmp);
        g.FillRectangle(参数);
        g.DrawString(参数);
}
private void MainForm_Paint(object sender,EventArgs e)
{
        e.Graphics.DrawImage(m_bmp,0,0);//将内存中绘制完成的位图显示在屏幕上
}
```

3. 将 Image 图像保存为文件

在一个内存位图的绘图表面上完成绘图工作以后，希望将图像结果保存为文件，以备今后浏览或使用，Bitmap 类提供了 Save 方法用于完成这一工作。

方法为：public void Save(string filename,ImageFormat format);

参数 filename 代表包含文件路径的文件名；ImageFormat 属性有四个可选项：bmp、gif、jpeg、png。

例如下面的代码将上面在内存中的生成的位图进行保存。

```
private    void SaveBitmap()
{
    if(m_bmp==null)
    return;
        m_bmp.Save("文件名. Bmp", ImageFormat. Bmp);
}
```

3.2　触摸屏的手写笔程序实例

3.2.1　功能设计

1. 功能描述

程序实现的功能是在 WINCE 设备上用手写笔点下并划动触摸屏时，将记录所有经过的点，并将它们连接在一起，这样整个图形就可以看成是由一个个线条构成的，在编程实现过程中，可以通过响应一个 MouseDown、若干个 MouseMove 和一个 MouseUp 事件的方法去实现线条的绘制。此外，由于要在重绘的时候避免已经绘制完成的线条丢失，程序还必须能够实现所有绘制内容的存储。设计思路可以先在内存中创建一个位图，由位图创建一个绘图表面，所有的绘图操作都是在内存的位图中进行的，如果要显示在屏幕上，再用绘制位图的方法显示出来，这样使用的优势就是当所需的图像包含有大量的基本图形时，在内存中的 Bitmap 上进行绘图比直接在屏幕上绘图要快很多，如图 3-4 所示。

图 3-4　手写笔设计界面

2. 手写笔事件响应处理

由于嵌入式设备的显示屏较小，一般都是通过手写笔来点击触摸液晶屏完成响应的输入

输出处理。在.NET CF 中使用事件驱动机制将手写笔的输入信息传送给应用程序处理，如表3-6 所示是嵌入式设备手写笔程序能够接受的事件，对于开发者而言，所要做的工作就是为事件对应的处理方法添加处理代码，以实现所需的程序功能。

表 3-6　程序手写笔事件处理说明

手写笔事件	程序事件对应的操作
Click 事件	当手写笔单击触摸屏时，响应事件
MouseDown 事件	当手写笔接触到触摸屏时，开启一个新的线条
MouseMove 事件	当手写笔接触到触摸屏并移动手写笔时，陆续连接手写笔经过的一个个连续的点，以便构成一个线条
MouseUp 事件	当手写笔离开触摸屏时，结束连接当前的最后一个点

3.2.2　功能实现

1. 手写笔自定义控件功能实现

（1）在 VS.NET2005 平台上选择 C#语言创建基于 Windows CE 5.0 的手写笔程序工程项目，名称为 Signature，创建完成之后解决方案资源管理器界面如图 3-5 所示。

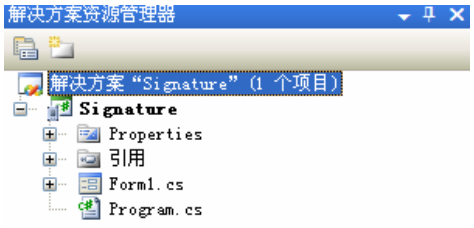

图 3-5　所示项目中解决方案资源管理器

（2）右击“解决方案选择”菜单项中的“添加→新建项目”项，进入“添加新项目”对话框，如图 3-6 所示，选择“控件库”模板，输入名称 SignatureControl，单击“确定”按钮。

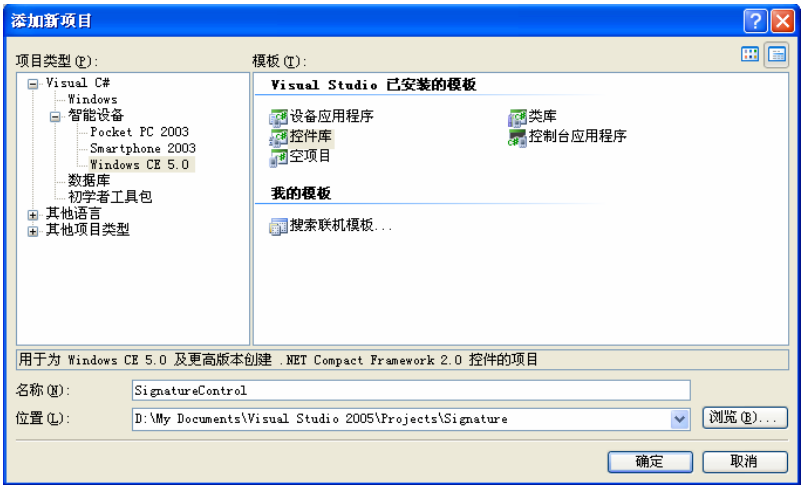

图 3-6　添加控件项目

（3）在项目解决方案资源管理器中出现控件项目 SignatureControl，单击控件项目中的 UserControl1.cs 项，在对应的属性栏中将文件名 UserControl1.cs 改为 SignatureControl.cs，如图 3-7 所示。

图 3-7　SignatureControl 控件项目

（4）控件功能代码的具体实现如下：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;

namespace SinosysSignatureControl
{
    public partial class Signature : UserControl
    {
        private Graphics _signatureSurface;
        private Bitmap _bitmap;
        private bool _stylusCaptured;
        private int _oldX, _oldY;
        public Color signColor=Color.Black;
        public int signWidth = 1;
        public Signature()
        {
            InitializeComponent();
            //初始化一个由内存位图创建的绘图表面
            InitializeGraphics();
        }

        private void InitializeGraphics()
```

```
        {
            this._stylusCaptured = false;
            // 创建指定宽度和高度的内存位图
            this._bitmap = new Bitmap(this.Size.Width, this.Size.Height);
            if (this._bitmap != null)
            {
                //根据内存位图创建绘图表面 graphics 对象
                this._signatureSurface = Graphics.FromImage(this._bitmap);
                if (this._signatureSurface != null)
                {
                    //内存位图的绘图表面填充白色的背景
                    this._signatureSurface.FillRectangle(new SolidBrush(Color.White),
                                                         0,
                                                         0,
                                                         this.Size.Width,
                                                         this.Size.Height);
                }
            }
        }

        //当屏幕窗体大小改变时，重新初始化绘图表面操作
        protected override void OnResize(EventArgs e)
        {
            InitializeGraphics();
            base.OnResize(e);
        }

        //向输出屏幕执行重绘操作
        protected override void OnPaint(PaintEventArgs e)
        {
            if (this._bitmap != null)
            {
              //从指定绘制区的左上角开始向屏幕绘制内存位图
                e.Graphics.DrawImage(this._bitmap, 0, 0);
            }
            base.OnPaint(e);
        }

       //消除屏幕闪烁
        protected override void OnPaintBackground(PaintEventArgs e)
        {
            base.OnPaintBackground(e);
        }

        //当手写笔接触触摸屏时，记录所在位置的 X 和 Y 坐标
        protected override void OnMouseDown(MouseEventArgs e)
```

```
{
    //true 代表手写笔接触屏幕，绘制操作开始
    this._stylusCaptured = true;
    //获取手写笔按下的位置 x 和 y 坐标
    this._oldX = e.X;
    this._oldY = e.Y;
    base.OnMouseDown(e);
}

//当手写笔离开触摸屏时，停止绘制操作
protected override void OnMouseUp(MouseEventArgs e)
{
    // false 表示绘制操作结束
    this._stylusCaptured = false;
    base.OnMouseUp(e);
}

//表示手写笔在屏幕绘制区中划动时，一方面进行内存位图的绘制，另一方面使绘制区无效，
//从而实时地在屏幕绘制区输出绘制的位图
protected override void OnMouseMove(MouseEventArgs e)
{
    int _x, _y;
    if (this._stylusCaptured)
    {
     //获得手写笔在移动过程中的轨迹坐标(X,Y)
        _x = e.X;
        _y = e.Y;
        if (this._signatureSurface != null)
        {
            //创建指定颜色和宽度的画笔对象
            Pen _pen = new Pen(signColor, signWidth);
            // Draw line from the old coordinates to the new coordinates
            this._signatureSurface.DrawLine(_pen, this._oldX, this._oldY, _x, _y);
        }
       //使绘制区无效，激发 Paint 事件
        this.Invalidate(new Rectangle(Math.Min(_x, this._oldX),
                                      Math.Min(_y, this._oldY),
                                      Math.Abs(this._oldX - _x) + 1,
                                      Math.Abs(this._oldY - _y) + 1));
        //获得手写笔接触屏幕的前一个点坐标位置
        this._oldX = _x;
        this._oldY = _y;
    }
    base.OnMouseMove(e);
  }

//清除绘图表面
```

```
        public void Clear()
        {
            if (this._signatureSurface != null)
            {
                //用白色画刷填充绘制区
                this._signatureSurface.FillRectangle(new SolidBrush(System.Drawing.Color.White),
                                                     0,
                                                     0,
                                                     this.Size.Width,
                                                     this.Size.Height);
                this.Invalidate();//在绘制区产生无效区域
            }
        }

    //将绘制完成的位图按照指定的图片格式和路径位置进行保存
        public void Save(string path,System.Drawing.Imaging.ImageFormat pic )
        {
            if (this._bitmap != null && path != null && path != "")
            {
                //Save the bitmap as a Gif file
                this._bitmap.Save(path, pic);
            }
        }
    }
}
```

（5）右击 SinosysSignatureControl 控件项目，选择“设为启动项目”选项，选择“生成”选项，进行控件项目编译，如果编译成功，这时可以在工具箱上方看到如图 3-8 所示的 SignatureControl 组件。

2. 项目窗体功能设计

（1）单击 Signature 工程项目下默认生成的 Form1.cs 窗体，在属性栏上重新命名为 MainFrm.cs 文件名，如图 3-9 所示。

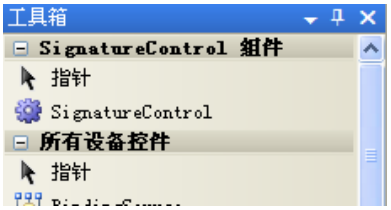

图 3-8　生成的 SignatureControl 控件

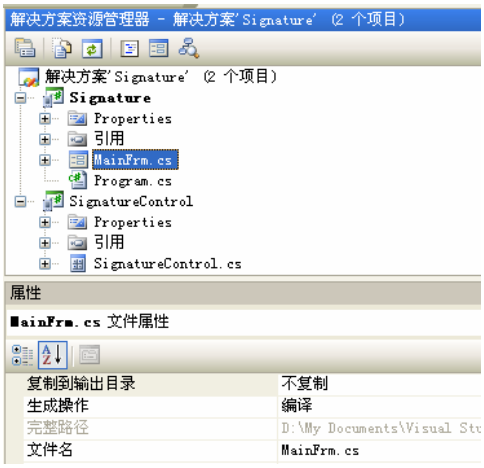

图 3-9　更改 Signature 工程项目 Form1 窗体名称

（2）在 MainFrm 窗体对应的属性栏中输入 Text 属性值为“手写笔程序”，Size 大小输入为 320,240，如图 3-10 所示。

图 3-10　更改 MainFrm 窗体属性值

（3）右击 Signature 项目中的“引用”，选择“添加引用”选项，如图 3-11 所示，这时选择控件项目中刚刚编译生成的 SignatureControl.dll，单击“确定”按钮。

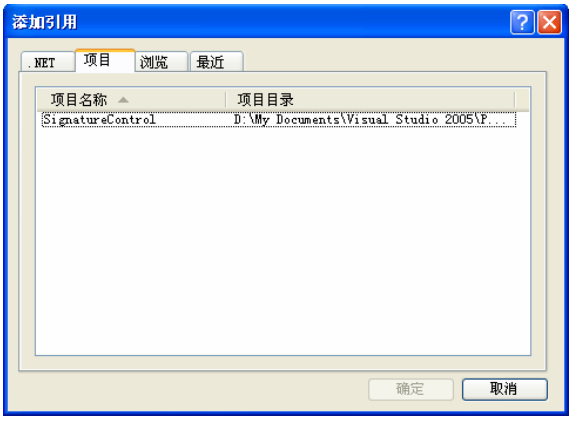

图 3-11　添加 SignatureControl.dll 动态链接库

（4）从工具栏中将 SignatureControl 控件拖放至 MainFrm 窗体设计界面上方中间，调整至合适的高度和宽度，另外添加一个下拉列表 comboBox 控件、四个 pictureBox 控件及两个 Button 按钮，如图 3-12 所示。

图 3-12　MainFrm 窗体设计界面

（5）将图 3-12 中所有控件进行规范命名和设置初始值，如表 3-7 所示进行说明。

表 3-7　项目各项控件说明

控件名称	命名	说明
SignatureControl	SignatureName	自定义的控件
comboBox	CboxWidth	在其 Items 属性中设置一号、二号和三号大小的线条宽度
pictureBox	pictureBoxBlack	设置黑色画笔
pictureBox	pictureBoxBlue	设置蓝色画笔
pictureBox	pictureBoxGreen	设置绿色画笔
pictureBox	pictureBoxRed	设置红色画笔
Button	btnClear	用于清除绘制区
Button	btnSave	用于保存位图文件

3. 项目事件方法处理

（1）窗体加载时执行 Load 事件的处理，这里将下拉列表的画笔宽度选择索引为 0 的值 1。

```
private void MainFrm_Load(object sender, EventArgs e)
{
    this.CboxWidth.SelectedIndex = 0;
}
```

（2）对四个 pictureBox 控件的 Click 事件方法进行处理，分别设置画笔的颜色为蓝色、红色、黑色和绿色。

```
private void pictureBoxBlue_Click(object sender, EventArgs e)
{
    SignatureName.signColor = Color.Blue;//设置为蓝色
}

private void pictureBoxRed_Click(object sender, EventArgs e)
{
    SignatureName.signColor = Color.Red; //设置为红色
}

private void pictureBoxBlack_Click(object sender, EventArgs e)
{
    SignatureName.signColor = Color.Black; //设置为黑色
}

private void pictureBoxGreen_Click(object sender, EventArgs e)
{
    SignatureName.signColor = Color.Green; //设置为绿色
}
```

（3）对下拉列表框选项索引的改变事件进行处理，根据不同的索引值获得对应画笔的宽度。

```
private void CboxWidth_SelectedIndexChanged(object sender, EventArgs e)
{
    switch (CboxWidth.SelectedIndex)
    {
        case 0:
            SignatureName.signWidth = 1;//画笔宽度为 1
            break;
        case 1:
            SignatureName.signWidth = 2; //画笔宽度为 2
            break;
        case 2:
            SignatureName.signWidth = 3; //画笔宽度为 3
            break;
    }
}
```

（4）对屏幕的绘制区执行清屏操作。

```
private void btnClear_Click(object sender, EventArgs e)
{
    SignatureName.Clear();
}
```

（5）对绘制区中完成的图像进行保存，打开“保存”对话框之后，输入文件名称，图片按照指定图片格式和文件路径进行保存。

```
private void btnSave_Click(object sender, EventArgs e)
{
    SaveFileDialog SaveDlg = new SaveFileDialog();
    SaveDlg.Filter = "Jpg 文件|*.Jpg|Bmp 文件|*.Bmp|Png 文件|*.Png|Gif 文件|*.Gif";
    if (SaveDlg.ShowDialog() == DialogResult.OK)
    {
        switch (SaveDlg.FilterIndex)
        {
            case 1:
                SignatureName.Save(SaveDlg.FileName + ".jpg",
                System.Drawing.Imaging.ImageFormat.Jpeg);
                break;
            case 2:
                SignatureName.Save(SaveDlg.FileName + ".bmp",
                System.Drawing.Imaging.ImageFormat.Bmp);
                break;
            case 3:
                SignatureName.Save(SaveDlg.FileName + ".png",
                System.Drawing.Imaging.ImageFormat.Png);
                break;
            case 4:
                SignatureName.Save(SaveDlg.FileName + ".gif",
```

```
                    System.Drawing.Imaging.ImageFormat.Gif);
                    break;
                }
            }
        }
```

3.3 电子相册应用开发

3.3.1 项目分析

1. 项目开发背景

随着数字化技术的不断发展，一系列数码设备（如数码相机及数码摄像机）走入家庭，电子相册遂即诞生。它以欣赏方便、交互性强、易于保存等特点迅速得到现代人的青睐。利用原有的终端 PC 机来管理数字照片已不能满足当前需求，有时可能需要带有照相功能或者摄像头功能的便携式设备在任何地方捕捉一些场景，然后在任何时间利用电子相册来管理这些数字照片。同时电子相册具有传统相册无法比拟的优越性：图、文、声、像并茂的表现手法，随意修改编辑的功能，快速的检索方式，永不褪色的恒久保存特性，以及廉价复制分发的优越手段。

2. 功能结构分析

项目的最大特点就是，可以用手写笔或手指拨动相应的图片索引，拨动的速度越快，则图片翻转得越快，有点像快速翻书的感觉。图片信息采用 XML 文件保存，图片文件单独放置在设定好的文件夹中进行管理。用户如果想添加新的图片，可以通过可视化程序界面添加必要的图片及修改 XML 文件数据，这样就完成了数字相片添加。项目共包括十个类别的数字照片。

程序一旦运行直接进入图片分类界面，如图 3-13 所示，该界面显示当前的三个类别的图片信息（包括类别图片显示、类别标题），通过手写笔或手指拨动相应的图片索引可以进入下一个相邻的三个类别的图片信息，通过手写笔或手指单击中间图片可以进入该类别的所有图片显示界面，如图 3-14 所示，增添图片可以通过单击工具栏图标进入添加图片窗口，如图 3-15 所示。

图 3-13　图片分类运行界面

图 3-14　具体类别的图片运行界面

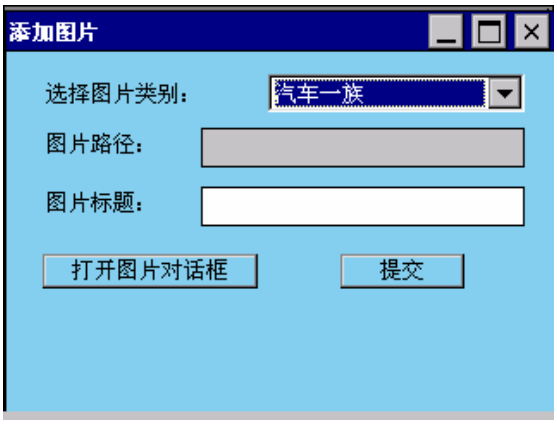

图 3-15　添加图片运行界面

3.3.2　XML 基础

1. XML 简介

当前 Web 上流行的 HTML 是一种标记语言，而不是一种编程语言，主要标记是针对显示，而不是针对文档内容本身结构描述的。对于机器本身而言是不能够解析它的内容的，同时为了存储、传送和交换数据，而不是用来显示数据的，继而就出现了 XML 语言。它是可扩展的标记语言，不像 HTML 那样提供一组事先已经定义好的标记，而是提供一个标准，利用这个标准，可以根据实际的需要，自定义新的标记。XML 具有跨平台、结构清晰、数据内容和表现形式相分离、高度灵活和可扩展等特性，使得它可以被广泛应用于除 Web 应用之外的多种需要数据交换的场合。

在嵌入式环境下，由于硬件存储资源有限以及软硬件平台的多样性，利用 XML 来存储数据量不是很大的数据，是一个很好的选择途径，因为 XML 是与平台无关的一种标记语言，它使用基于文本的、高度结构化的方式对数据进行描述，因此可以在多种平台之间交换 XML 格式的数据。

XML 能够为异构多变环境下的数据交互提供支持，因此在嵌入式环境下应用 XML 具有重要的意义，主要体现在以下几个方面：

（1）提高了设备访问的透明性。XML 文本具有很强的可读性，也很容易被机器处理，对 XML 文本的编辑结果，能直接体现在嵌入式访问中。

（2）统一了设备访问的接口。使用 XML 可以完成大多数对嵌入式系统的访问，这些操作包括对硬件的配置、测试及控制等。访问方式的统一大大简化了对嵌入式设备的访问接口的管理，提高了嵌入式设备与其他平台的交互能力。

（3）平台独立性。XML 是与平台无关的，它使用基于文本的、高度结构化的方式对数据进行描述，因此可以在多种平台间交换 XML 格式的数据。

2. XML 数据存储

XML 文档本身就是以文本文件的形式存在的，利用文本文件来存储 XML 文档数据是最简单的存储方式，它便于管理、适用存储数据量较小的数据，同时能够清楚地反映对象之间的嵌套和所属关系。

例如以下显示一个标准的 XML 文档内容：

```
<?xml version="1.0"?>
  <books>
    <book>
        <author>Jason</author>
        <price>20.65</price>
        <pubdate>06/06/2006</pubdate>
    </book>
    <pubinfo>
        <publisher>MSPress</publisher>
        <state>WA</state>
    </pubinfo>
  </books>
```

3. XML 数据访问

在程序开发过程中，如果要对 XML 文档进行访问和操作，必须通过一种工具按一定的语法来分析和理解存储在文档中的信息，这个工具就是 XML 解析器，它是一个对 XML 文档进行语法分析的 DLL 动态链接库，应用程序正是通过这个 DLL 的接口，实现对 XML 文档的识别和访问。XML 解析器的 DLL 接口分以下两种：基于树或者基于事件的处理器。这两种方式通常都是利用 DOM（Document Object Model）和 SAX（Simple API for XML）提供的编程接口实现应用程序对 XML 数据的访问和存取。如图 3-16 所示是 DOM 和 SAX 在应用程序开发过程中所处的位置。

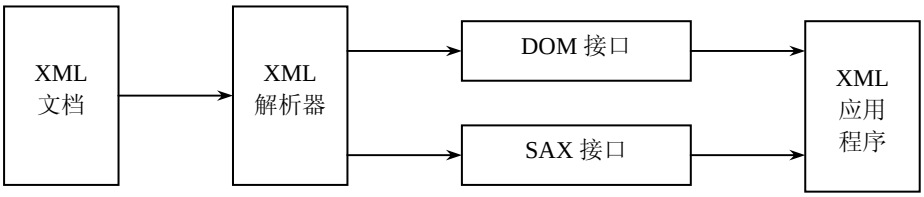

图 3-16　DOM 与 SAX 接口位置

本书电子相册程序涉及到的是 DOM 文档对象模型，故只讨论 DOM。XML 解析器根据 XML 文档生成一个指定平台的 DOM 树结构对象，DOM 用一些类来表示，它以一组非常直观的方式访问和处理 XML 文档。它能够把文档看成是一个有结构的信息树，而不是简单的文本流。这样即使不知道 XML 的语义细节，应用程序也能够方便地操作该结构。DOM 包含两个关键的抽象：一个是树状的层次，另一个是用来表示文档内容和结构的节点集合。树状层次包括所有这些节点，节点本身也可以包含其他的节点，这样的好处使得开发人员可以通过这个层次结构来找到并修改相应的某一个节点信息。另外 DOM 把节点看成是一个通常的对象，这样能够以编程方式先装载一个 XML 文档，然后遍历所有节点，显示和修改节点的信息。这里节点可以有很多种具体的类型，比如元素、属性和文本都可以认为是一个节点。图 3-17 显示了 XML 数据读入 DOM 结构中如何构造文档内存结构。

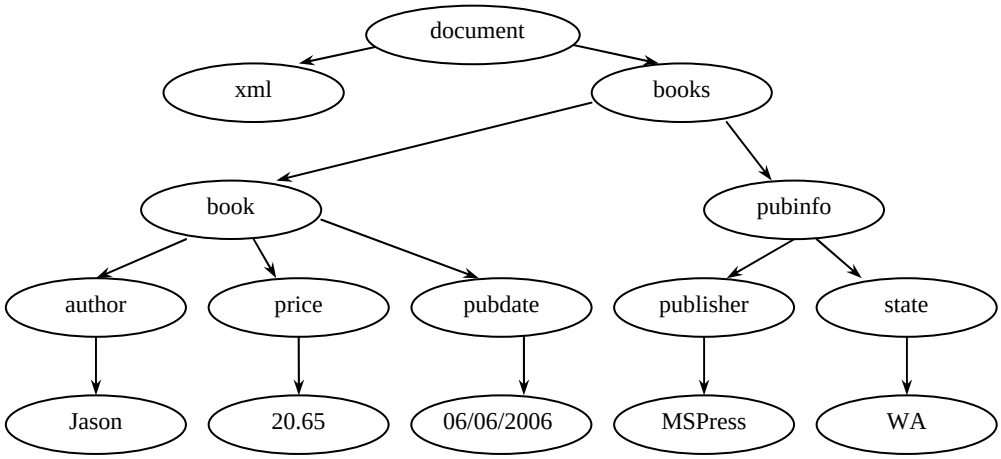

图 3-17　DOM 文档内存结构

4. VS.NET 平台中使用 DOM

对开发人员来说，最重要的是利用已设计好的类去调用它的方法完成相应的功能，在VS.NET中构成DOM的类在命名空间System.XML中，该命名空间包含一些操作XML文档的类，使用System.Xml命名空间中的类有很多优点。

第一，System.Xml 命名空间中类代码是托管代码，它可以确保所有的代码都获得安全保护。

第二，使用托管代码具有一定的类型安全性，同时 System.Xml 命名空间很容易使用，灵活性也非常大。

（1）XmlDocument 对象。通常处理 XML 文件要先从磁盘中加载它，这是 XmlDocument 对象的工作。可以将 XmlDocument 看作磁盘上文件的内存表示。用 XmlDocument 对象把文件加载到内存后，就可以从中获得文档的根节点，开始读取和处理 XML 了。

下面代码为创建一个 XmlDocument 对象并使用 Load 方法加载 XML 数据，然后可以依次遍历每个级次的节点直至找到要找的节点。

```
XmLDocument doc = new XmLDocument();
doc.Load(“Test.xml”);
XMLNode root=doc.FirstChild;
    for (int i=0; i<root.ChildNodes.Count; i++)
{
    YourMethod(root.ChildNodes[i]);
}
```

（2）XmlElement 对象。当文档加载到内存后，就要对它执行一些操作。创建 XmlDocument 对象后，其 DocumentElement 属性会返回一个 XmlElement 对象，它表示 XmlDocument 的根节点。有了它，就可以访问文档中的所有信息。

例如：

```
XmlDocument document = new XmlDocument();//创建 XmlDocument 的新实例
document.Load("demo.xml");//加载 demo.xml 文件
XmlElement element = document.DocumentElement;
```

5. 实现 XML 的序列化及反序列化

序列化是把一个对象以文件流的形式存入磁盘并以 XML 文件进行保存的过程，或者通过网络进行传输到另一个设备端，这样就可以在异构多变环境下进行数据交互。序列化技术在分布式系统的数据传输中得到充分的应用，例如，XML Web Service 利用 XML 序列化实现跨平台，.NET Remoting 则用到了二进制序列化和 SOAP 序列化。.NET Compact Framework 2.0 支持 XML 序列化，不支持二进制序列化和 SOAP 序列化。所以可以在定制 Windows CE 操作系统过程中，添加.NET Compact Framework 2.0 的组件支持，这样就可以开发针对 XML 文件的嵌入式应用程序。

System.Xml.Serialization 命名空间中包含的 XmlSerializer 类可用于把对象序列化为 XML 文档或流。它可以将对象的公共属性和公共字段转换为 XML 元素和属性。

要序列化对象，可以按照以下三个步骤进行：

（1）实例化一个 XmlSerializer 对象，指定要序列化的对象类型。

（2）实例化一个流/写入器对象，把文件写入流/文档。

（3）在 XmlSerializer 上调用 Serializer()方法，给它传送流/写入器对象和要序列化的对象。

对于从 XML 文档中反序列化对象，则应执行上述过程的逆过程，步骤如下：

（1）实例化一个 XmlSerializer 对象，指定要序列化的对象类型。

（2）实例化一个流/写入器对象，把文件写入流/文档。

（3）在 XmlSerializer 上调用 DeSerializer()方法传送该流/读取器对象。该方法返回反序列化的对象，但需要转换为正确的类型。

例如下面的代码演示序列化及反序列化过程：

（1）要序列化的对象的类。

```
public class Person
{
    private string name;
    public string Name
    {
      get
      {return name;}
      set
          {name=value;}
      }
    public string Sex="男";
    public int Age=31;
}
```

（2）进行序列化测试的类。

```
public class Test
{
public void Serialiaze()//序列化
{
Person p=new Person()
p.Name="张三";
XmlSerializer xs=new XmlSerializer(typeof(Person));
```

```
Stream stream = new FileStream("MyXmlFile.xml", FileMode.Create, FileAccess.Write, FileShare.ReadWrite);
xs.Serialize(stream, p);
stream.Close();
}
```

（3）格式化后 Xml 的文档内容为：

```
<?xmlversion="1.0" encoding="utf-8"?>
<Person xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance xmlns:xsd=
                    "http://www.w3.org/2001/XMLSchema">
 <Sex>男</Sex>
  <Age>31</Age>
  <Name>张三</Name>
</Person>
```

（4）进行反序列化测试。

```
public void Deserialize()//反序列化
{
XmlSerializer xs=new XmlSerializer(typeof(Person));
Stream stream = new FileStream("MyXmlFile.xml", FileMode.Open, FileAccess.Read, FileShare.ReadWrite);
Person p=(Person)xs.Deserialize(stream);
}
```

3.3.3　业务逻辑类的设计

1. XML 文档结构设计

项目采用 XML 文件保存必要的文字数据，图片文件单独放置在设定好的文件夹中进行管理。利用 XML 文档来保存数据，可以在跨平台的嵌入式异构环境中获取数据，而不需要专门配置一个数据库存放数据。

DataInfos 根节点包含多个子节点 DataInfo，每个 DataInfo 节点又分别包含图片类别标题、图片数量、类别图片文件路径、小类图片路径、小类图片标题。

下面代码为电子相册中 XML 文档的数据组织结构。

```
<?xmlversion="1.0" encoding="utf-8"?>
<DataInfos xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <DataInfo>
      <Text>大类图片标题值</Text>
      <Count>图片数量值</Count>
      <bmpPath>大类图片文件路径值</bmpPath>
      <mlstImagePath>
            <string>小类图片文件路径值</string>
      </mlstImagePath>
      <mlstTitle>
            <string>小类图片标题值</string>
      </mlstTitle>
  </DataInfo>
  <DataInfo>
      <Text>大类图片标题值</Text>
```

```
        <Count>图片数量值</Count>
        <bmpPath>大类图片文件路径值</bmpPath>
        <mlstImagePath>
            <string>小类图片文件路径值</string>
        </mlstImagePath>
        <mlstTitle>
            <string>小类图片标题值</string>
        </mlstTitle>
    </DataInfo>
</DataInfos>
```

2. DataInfo 类的设计

为了能够按照面向对象的方式获取和处理 XML 文档数据，可以将 XML 文档进行反序列化，生成一个 DataInfo 类型的对象，这样就可以通过调用 DataInfo 类型对象的成员获得 XML 文档中某一 DataInfo 节点中的各项数据。

这里先要建立 DataInfo 类，代码如下：

```
public   class DataInfo
{
    //图片名称
    public string Text = "";
    //信息个数
    public int Count = 0;
    //图片说明
    public string[] mlstTitle = null;
    //图片路径
    public string[] mlstImagePath = null;
    //索引图片路径
    public string bmpPath = "";
    //信息索引
    [XmlIgnore]
    public int Index = 0;
    //索引图片
    [XmlIgnore]
    public Image bmp = null;
    //图片
    [XmlIgnore]
    public List<Image> lstBmp = new List<Image>();
    [XmlIgnore]
    public List<string> lstImagePath
    {
        get
        {
            List<string> mmlstImagePath = new List<string>();
            if (mlstImagePath != null)
            {
                foreach (string str in mlstImagePath)
                {
```

```
                    mmlstImagePath.Add(str);
                }
            }
            return mmlstImagePath;
        }
    }
    [XmlIgnore]
    public List<string> lstTitle
    {
        get
        {
            List<string> mmlstTitle = new List<string>();
            if (mlstTitle != null)
            {
                foreach (string str in mlstTitle)
                {
                    mmlstTitle.Add(str);
                }
            }
            return mmlstTitle;
        }
    }
}
```

3. 实现反序列化的 XMLDB 类的设计

要实现一个 XML 的反序列化，首先根据给定的 XML 文档创建一个文件流（Stream）对象，然后创建一个指定类型（typeof(DataInfo[])）的 XmlSerializer 对象，最后根据文件流（Stream）对象参数调用 DeSerializer()方法，这个方法返回反序列化 DataInfo[]类型对象。

反序列化 XMLDB 类的代码如下：

```
public   class XMLDB
{
    //反序列化
    public DataInfo[] XMLDeserialize(string XmlFile)
   {
        try
        {
        Stream sf = new FileStream(XmlFile, FileMode.Open, FileAccess.Read, FileShare.None);
        XmlSerializer xmls = new XmlSerializer(typeof(DataInfo[]));
        DataInfo[] XmlData = (DataInfo[])xmls.Deserialize(sf);
         if (XmlData == null)
            {
          MessageBox.Show("文件反序列化失败", "", MessageBoxButtons.OK,
          MessageBoxIcon.Hand, MessageBoxDefaultButton.Button1);
          Application.Exit();
            }
            sf.Close();
            return XmlData;
```

```
            }
            catch (Exception e)
            {
               MessageBox.Show("反序列化失败:" + XmlFile + "<" + e.Message + ">");
                  return null;
            }
      }
}
```

4. PhotoDB 类的设计

PhotoDB 类中建立两个方法，一个是 GetImage 方法，其功能是根据给定的图片文件路径，返回一个 Image 类型的图片对象；另一个是 GetDataInfosByImageData 方法，其功能是根据给定的宽度和高度创建一个位图对象和 Graphics 对象。

PhotoDB 类的代码如下：

```
public    class PhotoDB
{
      public Bitmap bitmap;
      public Graphics graphics;
      public enum IndexType { Before, Front, Current, Next, Back };
      public enum MoveType { MoveFrist, MoveUp, MoveDown, MoveLast };
      //获取当前图片
      public Image GetImage(string strFile)
      {
            if (strFile.Length == 0) return null;
            string strPath = @"\Program Files\FilePath" + strFile;
            if (File.Exists(strPath))
            {
                  return new Bitmap(strPath);
            }
            else
            {
                  return null;
            }
      }
      //根据指定高度和宽度获取图片对象和 Graphics 对象
      public void    GetDataInfosByImageData(int width,int height)
      {
            bitmap = new Bitmap(width, height);
            graphics = Graphics.FromImage((System.Drawing.Image)bitmap);
      }
}
```

3.3.4 用户界面设计

1. 创建电子相册工程项目

（1）打开 VS.NET2005 开发平台，创建基于 Windows CE 5.0 的 C#设备应用程序，工程名为 SinoSysPhoto。

（2）右击“项目”，选择“添加→新建”项，打开“添加新项”对话框，选择“类”模板，如图 3-18 所示，分别添加 DataInfo 类、XMLDB 类及 PhotoDB 类文件。

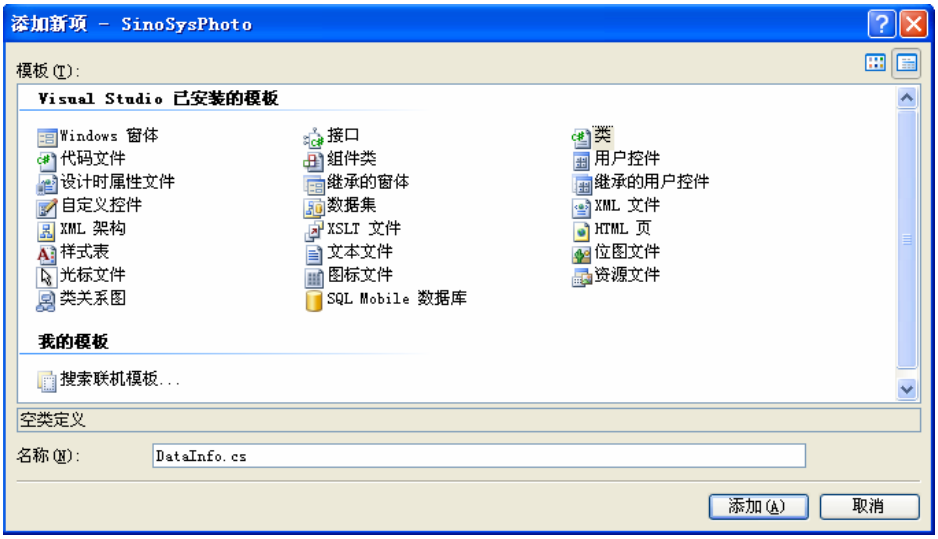

图 3-18　添加新项对话框

（3）右击“添加→新建”项，打开“添加新项”对话框，选择“窗体”模板，分别添加 MainFrm 窗体、InfoFrm 窗体及 AddPhotoFrm 窗体文件，如图 3-19 所示。

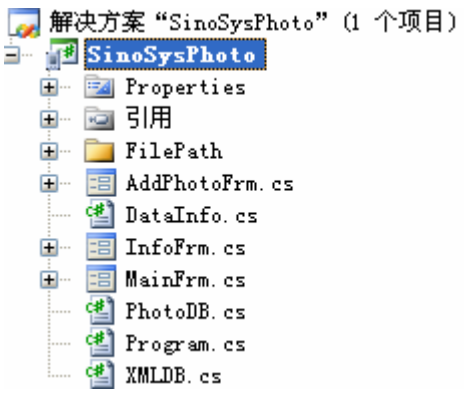

图 3-19　项目解决方案中添加的文件

2. 图片分类窗体（MainFrm）功能的实现

（1）MainFrm 窗体代码文件（MainFrm.cs）结构。

```
public partial class MainFrm : Form
{
    string strInfo = "";
    int intCurrentIndex = 0;
    bool bSelectDown = false;
    public   DataInfo[] DI = null;
    Bitmap bitmap;
    Graphics graphics;
```

```
        PhotoDB pdb = null;
        XMLDB xdb = null;
        //定义手写笔划动的相关变量
        bool MouseFlag = false;
        bool MouseClickFlag = false;
        float fStartX = 0;
        float fWidth = 0;
        int intMoveSpace = 10;
        bool bWay = false;
         public MainFrm()
        {
        }
        private int GetIndex(PhotoDB.IndexType Flag)
         {
         }
        private void MainForm_MouseUp(object sender, MouseEventArgs e)
        {
        }
        private void MainForm_MouseDown(object sender, MouseEventArgs e)
        {
        }
        private void MainFrm_Paint(object sender, PaintEventArgs e)
        {
         }
        private void ShowInfoBar()
        {
        }
    }
```

（2）MainFrm 方法。它是 MainFrm 窗体类的构造函数 。当创建 MainFrm 窗体时被自动调用，它完成三个操作：

第一，实例化 XMLDB 类的对象，然后调用 XMLDeserialize 方法完成 Xml 文档的反序列化。

第二，实例化 PhotoDB 类型对象，然后调用 GetDataInfosByImageData 方法完成位图对象以及 Graphics 对象创建。

第三，在遍历 DataInfo[]数组的过程中，以 DataInfo 对象中的图片文件路径作为参数，调用 GetImage 方法，获取相应的图片对象。

```
    public MainFrm()
    {
        InitializeComponent();
        xdb=new XMLDB();
        DI=xdb.XMLDeserialize(@"\Program Files\FilePath\PicFile.xml");
        pdb = new PhotoDB();
        pdb.GetDataInfosByImageData(this.Width, this.Height);
        bitmap = pdb.bitmap;
        graphics = pdb.graphics;
```

```
        //设置索引图片
        foreach (DataInfo di in DI)
        {
            di.bmp = pdb.GetImage(di.bmpPath);
        }
    }
```

（3）GetIndex(PhotoDB.IndexType Flag)方法。该方法根据给定的 IndexType 类型（包括 Before、Front、Current、Next、Back 五种类型）获得将要显示的图片所在位置索引。

```
    //获得指定的索引值
    private int GetIndex(PhotoDB.IndexType Flag)
    {
        int intIndex = 0;
        switch (Flag)
        {
            case PhotoDB.IndexType.Before:
                intIndex = intCurrentIndex - 2;
                if (intIndex < 0) intIndex = DI.Length - Math.Abs(intIndex);
                break;
            case PhotoDB.IndexType.Front:
                intIndex = intCurrentIndex - 1;
                if (intIndex < 0) intIndex = DI.Length - 1;
                break;
            case PhotoDB.IndexType.Current:
                intIndex = intCurrentIndex;
                break;
            case PhotoDB.IndexType.Next:
                intIndex = intCurrentIndex + 1;
                if (intIndex > DI.Length - 1) intIndex = 0;
                break;
            case PhotoDB.IndexType.Back:
                intIndex = intCurrentIndex + 2;
                if (intIndex > DI.Length - 1) intIndex = intIndex - DI.Length;
                break;
            default:
                break;
        }
        return intIndex;
    }
```

（4）MainForm_MouseDown 事件处理方法。当手写笔在触摸屏上按下时，该方法被调用，以确定触摸笔按下的区域，在后面 MouseUp 方法调用时，用来确定手写笔是划动触摸屏还是点击触摸屏。

```
    //手写笔按下触摸屏
    private void MainForm_MouseDown(object sender, MouseEventArgs e)
    {
        Rectangle rect = new Rectangle(0, 50, 320, 128);
```

```
        if (rect.Contains(e.X, e.Y))
        {
            MouseFlag = true;
            fStartX = e.X;
        }
        rect = new Rectangle(90, 50, 128, 128);
        if (rect.Contains(e.X, e.Y))
        {
            MouseClickFlag = true;
            bSelectDown = true;
            this.Invalidate();
        }
    }
```

（5）MainForm_MouseUp 事件处理方法。如果手写笔在触摸屏上向左或向右划动时，表示要将图片向前或向后切换进行显示，如果手写笔在触摸屏的中间图片区域内单击，表示要进入该类别的图片信息显示窗体。

```
    //手写笔抬起离开触摸屏时
    private void MainForm_MouseUp(object sender, MouseEventArgs e)
    {
        if (MouseFlag)
        {
            MouseFlag = false;
            fWidth = e.X - fStartX;
            bWay = (fWidth > 0);
            fWidth = Math.Abs(fWidth);
            if (fWidth > intMoveSpace)
            {
              //判断手写笔向左还是向右划动
                if (bWay)
                {
                    MoveRight();
                }
                else
                {
                    MoveLeft();
                }
            }
            else
            {
                //手写笔单击事件发生
                if (MouseClickFlag)
                {
                    MouseClickFlag = false;
                    bSelectDown = false;
                    ShowInfoBar();
                }
```

```
            }
        }
    }
```

（6）MoveLeft 方法。当确定手写笔在触摸屏上向左划动时，调用该方法设置新的图片显示位置索引值。

```
    //手写笔向左移动
    private void MoveLeft()
    {
        Thread.Sleep(10);
        intCurrentIndex = GetIndex(PhotoDB.IndexType.Next);
        this.Invalidate();
    }
```

（7）MoveRight 方法。当确定手写笔在触摸屏上向右划动时，调用该方法设置新的图片显示位置索引值。

```
    //手写笔向右移动
    private void MoveRight()
    {
        Thread.Sleep(10);
        intCurrentIndex = GetIndex(PhotoDB.IndexType.Front);
        this.Invalidate();
    }
```

（8）MainFrm_Paint 事件处理方法。当最终要在触摸屏上显示图片时，调用该方法将经过处理的内存位图显示在屏幕上。同时当窗体绘制区无效时，也调用该方法进行窗体重绘。

```
    private void MainFrm_Paint(object sender, PaintEventArgs e)
    {
        Rectangle rect = new Rectangle(0, 0, 128, 128);
        Font font = new Font("宋体", 16, FontStyle.Bold);
        StringFormat fFormat = new StringFormat();
        fFormat.Alignment = StringAlignment.Center;
        fFormat.LineAlignment = StringAlignment.Center;
        //绘制背景
        graphics.FillRectangle(new SolidBrush(Color.White),0,0, this.Width, this.Height);
        //写标题文本
        strInfo = DI[GetIndex(PhotoDB.IndexType.Current)].Text;
        graphics.DrawString(strInfo, font, new SolidBrush(Color.DarkGreen), new RectangleF(0, 10, 320, 20),
                            fFormat);
        //绘制中心图片
        if (!bSelectDown) graphics.FillRectangle(new SolidBrush(Color.Black), new Rectangle(93, 53, 128,
          128));
        graphics.DrawImage(DI[GetIndex(PhotoDB.IndexType.Current)].bmp, 90, 50);
        //绘制左边图片
        graphics.FillRectangle(new SolidBrush(Color.Black), new Rectangle(-7, 77, 90, 80));
        graphics.DrawImage(DI[GetIndex(PhotoDB.IndexType.Front)].bmp, new Rectangle(-10, 74, 90, 80), rect,
                           GraphicsUnit.Pixel);
        //绘制右边图片
```

```
        graphics.FillRectangle(new SolidBrush(Color.Black), new Rectangle(232, 77, 95, 80));
        graphics.DrawImage(DI[GetIndex(PhotoDB.IndexType.Next)].bmp, new Rectangle(228, 74, 95, 80), rect,
                           GraphicsUnit.Pixel);
        //在触摸液晶屏上绘制位图
        e.Graphics.DrawImage(bitmap, 0, 0);
}
```

（9）ShowInfoBar 方法。调用该方法打开某一类别的详细信息对话框窗体。

```
private void ShowInfoBar()
{
    int index=GetIndex(PhotoDB.IndexType.Current);
    InfoFrm frm = new InfoFrm(pdb, DI, index);
    frm.ShowDialog();
}
```

（10）图片分类窗体的运行效果如图 3-20 所示。

图 3-20　图片分类窗体运行界面

3. 图片详细信息 InfoFrm 窗体功能的实现

（1）InfoFrm 窗体代码文件（InfoFrm.cs）结构。

```
public partial class InfoFrm : Form
{
public DataInfo diCurrentlyData = null;
public DataInfo[] DIS = null;
public PhotoDB mypdb = null;
public int Index;
public InfoFrm(PhotoDB pdb,DataInfo[] di,int index)
{
}
private void toolBar_ButtonClick(object sender, ToolBarButtonClickEventArgs e)
{
}
private void Move(PhotoDB.MoveType type)
{
```

```
}
}
private void InfoFrm_Paint(object sender, PaintEventArgs e)
{
}
```

（2）InfoFrm(PhotoDB pdb,DataInfo[] di,int index)方法。当进入图片详细信息窗体时，执行 InfoFrm 构造函数，同时传递 PhotoDB 类型的对象和 DataInfo 类型的数组对象以及将要显示的图片索引。

```
public InfoFrm(PhotoDB pdb,DataInfo[] di,int index)
{
    InitializeComponent();
    mypdb = pdb;
    DIS = di;
    diCurrentlyData =di[index];
    for (int i = 0; i < diCurrentlyData.lstImagePath.Count; i++)
    {
        diCurrentlyData.lstBmp.Add(pdb.GetImage(diCurrentlyData.lstImagePath[i]));
    }
    diCurrentlyData.Index = 0;
}
```

（3）toolBar_ButtonClick 事件处理方法。当单击导航栏上的各个按钮时，根据各个按钮的参数不同，触发不同的事件，以执行对应的操作，如返回图片分类界面、进入该类别的第一张图片、上一张图片、下一张图片、最后一张图片、进入添加图片窗体界面。

```
private void toolBar_ButtonClick(object sender, ToolBarButtonClickEventArgs e)
{
    switch (e.Button.ToolTipText)
    {
        case "返回":
            this.Close();
            break;
        case "最前":
            Move(PhotoDB.MoveType.MoveFrist);
            break;
        case "前一个":
            Move(PhotoDB.MoveType.MoveUp);
            break;
        case "下一个":
            Move(PhotoDB.MoveType.MoveDown);
            break;
        case "最后":
            Move(PhotoDB.MoveType.MoveLast);
            break;
        case "关于":
            MessageBox.Show("电子相册系统\n" );
            break;
        case "添加图片":
            this.Close();
```

```
                AddPhotoFrm af = new AddPhotoFrm(mypdb,DIS);
                af.ShowDialog();
                break;
        }
    }
```

（4）Move(PhotoDB.MoveType type)方法。根据移动的类型不同，获得需要显示的图片索引，同时确定四个导航按钮的可用情况。

```
    private void Move(PhotoDB.MoveType type)
    {
        switch (type)
        {
            case PhotoDB.MoveType.MoveFrist:
                diCurrentlyData.Index = 0;
                tBar1.Enabled = false;
                tBar2.Enabled = false;
                tBar3.Enabled = true;
                tBar4.Enabled = true;
                break;
            case PhotoDB.MoveType.MoveUp:
                diCurrentlyData.Index--;
                tBar3.Enabled = true;
                tBar4.Enabled = true;
                if (diCurrentlyData.Index < 0)
                {
                    diCurrentlyData.Index = 0;
                }
                if (diCurrentlyData.Index < 1)
                {
                    tBar1.Enabled = false;
                    tBar2.Enabled = false;
                }
                else
                {
                    tBar1.Enabled = true;
                    tBar2.Enabled = true;
                }
                break;
            case PhotoDB.MoveType.MoveDown:
                diCurrentlyData.Index++;
                tBar1.Enabled = true;
                tBar2.Enabled = true;
                if (diCurrentlyData.Index > diCurrentlyData.Count - 1)
                {
                    diCurrentlyData.Index = diCurrentlyData.Count - 1;
                }
                if (diCurrentlyData.Index > diCurrentlyData.Count - 2)
                {
                    tBar3.Enabled = false;
```

```
                tBar4.Enabled = false;
            }
            else
            {
                tBar3.Enabled = true;
                tBar4.Enabled = true;
            }
            break;
        case PhotoDB.MoveType.MoveLast:
            diCurrentlyData.Index = diCurrentlyData.Count - 1;
            tBar3.Enabled = false;
            tBar4.Enabled = false;
            tBar1.Enabled = true;
            tBar2.Enabled = true;
            break;
        default:
            break;
    }
    this.Invalidate();
}
```

（5）InfoFrm_Paint(object sender, PaintEventArgs e) 事件处理方法。当最终要在触摸屏上显示图片时，调用该方法将经过处理的内存位图显示在屏幕上。同时当窗体绘制区无效时，也调用该方法进行窗体重绘。

```
private void InfoFrm_Paint(object sender, PaintEventArgs e)
{
    if (diCurrentlyData.Count > 0)
      {
        Font font = new Font("宋体", 12, FontStyle.Bold);
        StringFormat fFormat = new StringFormat();
         fFormat.Alignment = StringAlignment.Near;
          //绘背景
          e.Graphics.FillRectangle(new SolidBrush(Color.White), 0, 0, this.Width, this.Height);
            if (diCurrentlyData.lstTitle[diCurrentlyData.Index].Length > 0)
            {
              e.Graphics.DrawString(diCurrentlyData.lstTitle[diCurrentlyData.Index], font,
                        new SolidBrush(Color.Black), new RectangleF(10, 30, 30, 200), fFormat);
            }
            if (diCurrentlyData.lstBmp[diCurrentlyData.Index] != null)
            {
            e.Graphics.FillRectangle(new SolidBrush(Color.Black),
                                    new Rectangle(49, 32, 230, 180));
            e.Graphics.DrawImage(diCurrentlyData.lstBmp[diCurrentlyData.Index], 45, 28);
          }
    }
}
```

（6）图片详细信息窗体的运行效果如图 3-21 所示。

图 3-21　图片详细信息窗体运行界面

4. 添加图片 AddPhotoFrm 窗体功能的实现

（1）AddPhotoFrm 窗体代码文件（AddPhotoFrm.cs）结构。

```
public partial class AddPhotoFrm : Form
{
    public DataInfo[] DIS = null;
    public PhotoDB mydb = null;
    private string filename = @"\Program Files\FilePath\PicFile.xml";
    private string newthumbfile;
    public AddPhotoFrm(PhotoDB pdb, DataInfo[] dis)
    {
    }
    private void AddPhotoFrm_Load(object sender, EventArgs e)
    {
    }
    private string    GetSmallImageData()
    {
    }
    private void btnAdd_Click(object sender, EventArgs e)
    {
    }
    private void btnOpenDlg_Click(object sender, EventArgs e)
    {
    }
}
```

（2）AddPhotoFrm(PhotoDB pdb, DataInfo[] dis)方法。当进入添加图片窗体时，执行 AddPhotoFrm 构造函数，将传递进 PhotoDB 类型的对象和 DataInfo 类型的数组对象。

```
public AddPhotoFrm(PhotoDB pdb, DataInfo[] dis)
{
    InitializeComponent();
    DIS = dis;
    mydb = pdb;
}
```

（3）AddPhotoFrm_Load 事件处理方法。该方法在执行过程中，循环遍历 DataInfo 对象，以获得显示的图片类别标题信息。

```
private void AddPhotoFrm_Load(object sender, EventArgs e)
{
    foreach (DataInfo dinfo in DIS)
    {
        this.cBoxText.Items.Add(dinfo.Text);
    }
    this.cBoxText.SelectedIndex = 0;
}
```

（4）GetSmallImageData 方法。该方法根据给定的图片生成符合规格要求的缩略图，并将保存在所指定类别的图片文件夹中。

```
private string   GetSmallImageData()
{
    string[] newfilename = this.txtphotoPath.Text.Split('\\');
    //生成缩略图
    Image originalImage = new Bitmap(this.txtphotoPath.Text);
    Image newimage = new Bitmap(230, 180);
    Graphics g = Graphics.FromImage(newimage);
    g.DrawImage(originalImage, new Rectangle(0, 0, 230, 180),
                    new Rectangle(0, 0, originalImage.Width, originalImage.Height), GraphicsUnit.Pixel);
    int indexdirectoy = this.txtphotoPath.Text.IndexOf(newfilename[newfilename.Length - 1]);
    string addfilepath = this.txtphotoPath.Text.Substring(0, indexdirectoy);
    string thumbfilename = addfilepath + "small" + newfilename[newfilename.Length - 1];
    newimage.Save(thumbfilename, System.Drawing.Imaging.ImageFormat.Jpeg);
    //在原图片所在路径下保存一个原图的缩略图，文件名为原图片名加 small 字符串。
    newthumbfile = "\\" + newfilename[newfilename.Length - 2] + "\\small" + newfilename[newfilename.Length - 1];
    return newthumbfile;
}
```

（5）btnOpenDlg_Click 事件处理方法。执行该方法，打开对话框，选择所需图片格式，查找到新增的图片，然后单击“确定”按钮，这时新增的图片文件路径显示在文本框中。

```
private void btnOpenDlg_Click(object sender, EventArgs e)
{
    OpenFileDialog picdlg = new OpenFileDialog();
    picdlg.InitialDirectory = @"\Program Files\FilePath\";
    picdlg.Filter = "jpg 文件|*.jpg|png 文件|*.png|所有文件|*.*";
    picdlg.FilterIndex = 0;
    if (picdlg.ShowDialog() == DialogResult.OK)
    {
        this.txtphotoPath.Text = picdlg.FileName;
    }
}
```

（6）btnAdd_Click 事件处理方法。该方法将生成的缩略图的有关信息插入到 XML 文档

DataInfo 节点中。首先读取 XML 文档，创建一个文档对象，然后在所对应的 DataInfo 节点中创建新的节点及修改原节点的值。

```
private void btnAdd_Click(object sender, EventArgs e)
{
    bool flagBig = false;
    newthumbfile = this.GetSmallImageData();
    //向 XML 文件创建新的节点及修改原节点的数值
    XmlDocument Xmldoc = new XmlDocument();
    Xmldoc.Load(filename);
    //得到 DataInfo 节点
    XmlElement topelement = Xmldoc.DocumentElement;
    //   XmlNodeList topM = Xmldoc.DocumentElement.ChildNodes;
    //得到 Data 节点
    foreach (XmlElement element in topelement.ChildNodes)
    {
        bool flagSmall = false;
        if (flagBig)
        {
            break;
        }
        //得到 DataInfo 节点的各个子节点
        XmlNodeList nodelist = element.ChildNodes;
        //读 DataInfo 子节点中的各个元素值
        foreach (XmlElement e1 in nodelist)
        {
            if (flagSmall)
                break;
                switch (e1.Name)
                {
                 case "Text":
                 if (e1.InnerText != this.cBoxText.SelectedItem.ToString())
                 {
                        flagSmall = true;
                   }
                        break;
                        //修改对应 DataInfo 节点中子节点 Count 的值
                         case "Count":
                        e1.InnerText = (Int32.Parse(e1.InnerText) + 1).ToString();
                        break;
                        //在对应 mlstImagePath 节点中创建子节点 string
                           case "mlstImagePath":
                           XmlElement elem1 = Xmldoc.CreateElement("string");
                           elem1.InnerText= newthumbfile;
                           e1.AppendChild(elem1);
```

```
                    break;
                case "mlstTitle":
                    XmlElement elem2 = Xmldoc.CreateElement("string");
                    elem2.InnerText = this.txtTitle.Text;
                    e1.AppendChild(elem2);
                    flagBig = true;
                    break;
                    }
                }
            }
        Xmldoc.Save(filename);
        MessageBox.Show("新增成功！");
        this.Close();
        MainFrm mfrm = new MainFrm();
        mfrm.ShowDialog();
    }
```

（7）添加图片 AddPhotoFrm 窗体的运行效果如图 3-22 所示。

单击“打开图片对话框”按钮，显示如图 3-23 所示的对话框，选择添加的图片文件。

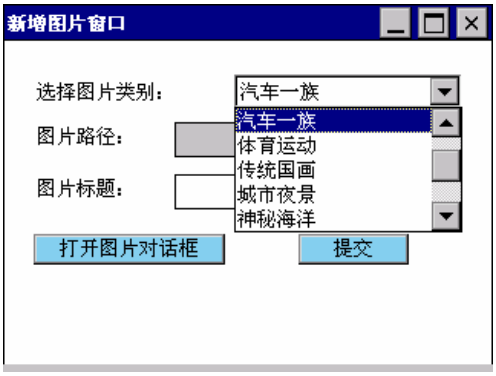

图 3-22　添加图片窗体运行界面

图 3-23　选择添加的图片

当图片选择完成之后，单击“提交”按钮，这时会提示图片添加成功信息，如图 3-24 所示。

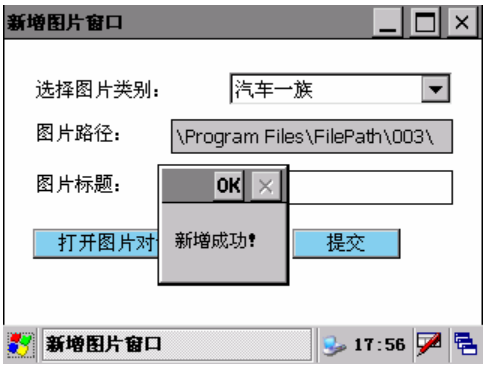

图 3-24　图片添加成功信息

3.3.5 部署安装应用程序

当在桌面计算机上完成 Windows CE 应用程序开发和测试工作之后，就需要将其部署到嵌入式设备中去运行。

1. 部署.NET Compact Framework 2.0

.NET Compact Framwork 是所有基于.NET 技术的嵌入式设备应用程序的运行基础，因此，每一个需要运行.NET 程序的嵌入式设备端，都必须安装.NET Compact Framework。Netcfsetup2.msi 是一个运行于桌面计算机的 Windows Install 数据文件，它可以将.NET Compact Framework 通过 ActiveSync 部署到嵌入式设备上。当使用 VS.NET2005 运行电子相册程序时，.NET Compact Framework 会被自动地部署到嵌入式设备上，另一个方法就是在定制 Windows CE 操作系统时，把.NET Compact Framework 组件选上即可，如图 3-25 所示。

图 3-25 .NET Compact Framework 安装

2. 制作 CAB 安装包

当用户需要在 WINCE 设备的开始菜单项中，通过单击安装程序名的快捷方式就能运行程序，就需要使用 CAB 安装包进行程序的部署。VS.NET2005 对 CAB 安装包制作提供了良好的支持，开发者只需进行一些操作，无须编写一行代码就能完成安装包的制作。

3. 创建 CAB 安装包项目

VS.NET2005 专门为制作安装包提供了项目类型和专用的项目模板，因此，一个 CAB 安装包的制作工作是从创建一个项目开始的，将安装项目添加到现有解决方案之中，开发者可以在解决方案资源管理器窗口中，右击“解决方案”条目，依次选择“添加→新建项目”选项，这样弹出“添加新项目”对话框，依次选择“安装部署”→“智能设备 CAB 项目”，输入项目名称 E_Photo，如图 3-26 所示。

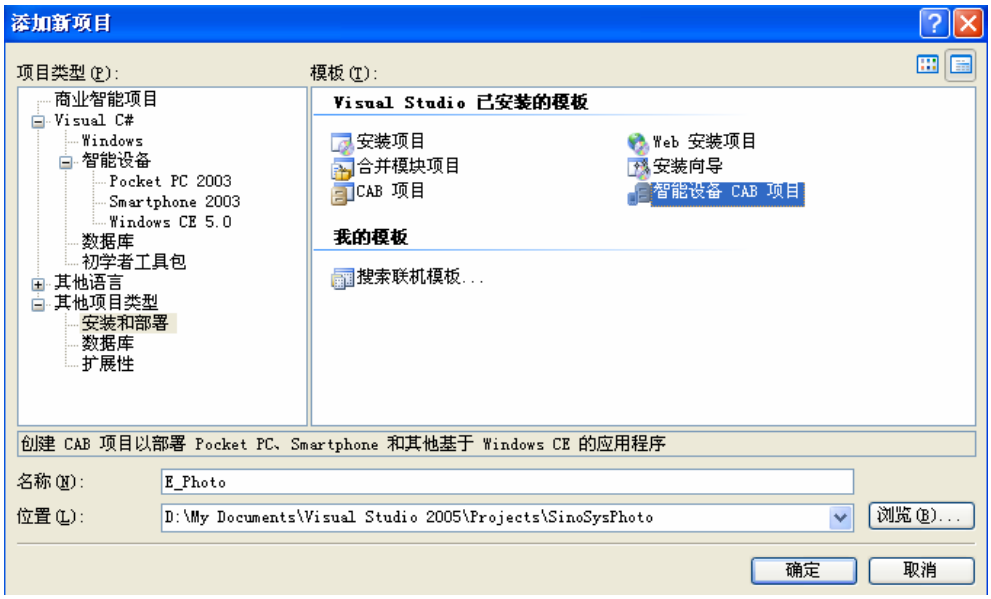

图 3-26　新建 CAB 安装包项目

4. 编辑器

当 CAB 安装包项目创建好之后，VS.NET2005 就会自动打开文件系统编辑器。在 Program Files 文件夹下建立子目录 FilePath 文件夹，再在 FilePath 文件夹下建立子目录 001 至 010 文件夹，分别在各自文件夹下选择相应的打包文件，如图 3-27 所示。

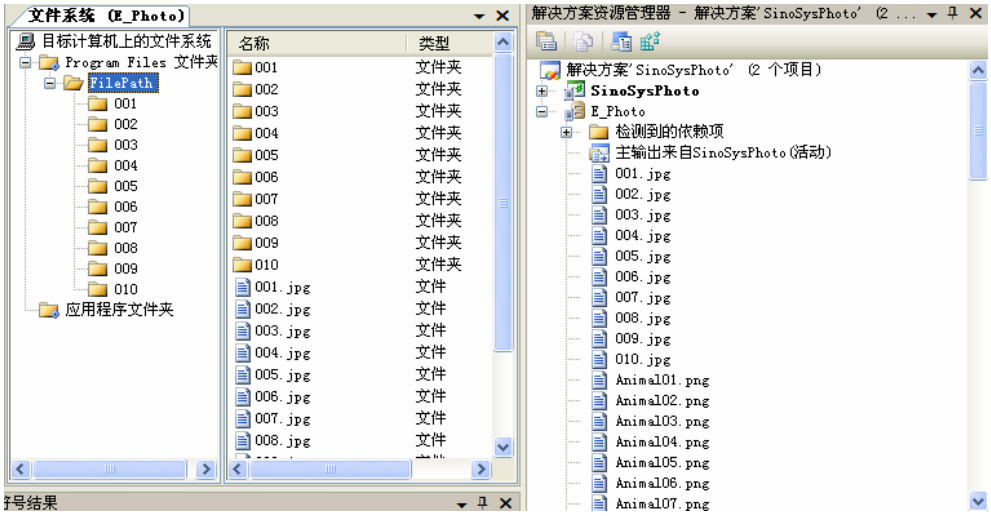

图 3-27　文件系统编辑器

5. 添加可执行文件

可执行文件通常放在“应用程序文件夹”中，通过右击“应用程序文件夹”，依次选择“添加”→“项目输出”菜单项，将弹出“添加项目输出组”对话框，如图 3-28 所示，选中“主输出”单击“确定”按钮，将对应项目输出的可执行文件添加到安装包中。

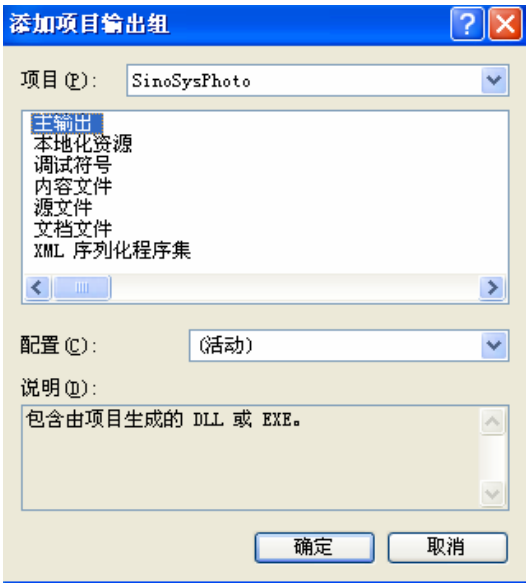

图 3-28 “添加项目输出组”对话框

6. 创建快捷方式

快捷方式通常是应用程序向用户展现的窗口，通过使用 VS.NET2005 为应用程序制作安装包，可以很方便地创建快捷方式。右击“目标计算机上的文件系统”，添加 Start Menu 文件夹，在编辑器右侧的列表视图中，右击鼠标，选择“创建新的快捷方式”菜单项，如图 3-29 所示。

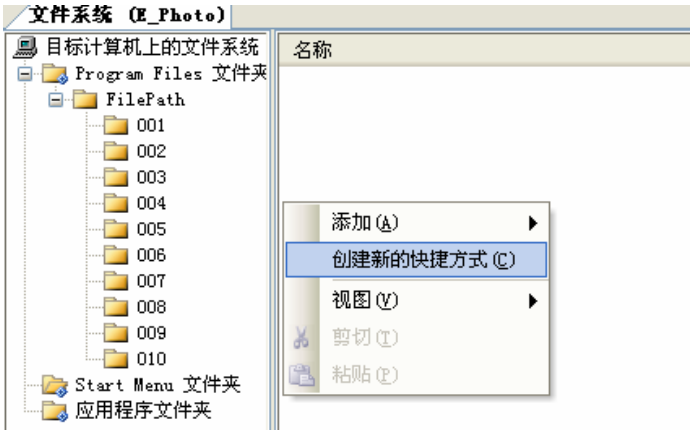

图 3-29 添加新的快捷方式

这时会出现如图 3-30 所示“选择项目中的项”对话框，这里选择应用程序文件夹中的主输出，单击“确定”按钮，新的快捷方式就添加完毕了。

7. 生成项目安装包

在解决方案资源管理器窗口中右击安装包 E_Photo 项目，选择“生成”项，如图 3-31 所示，等待片刻之后，生成扩展名为.CAB 的安装文件。用户可以使用 ActiveSync 分发 CAB 安装包，这时必须确保嵌入式设备与桌面计算机已经通过 ActiveSync 建立了良好的通信连接。

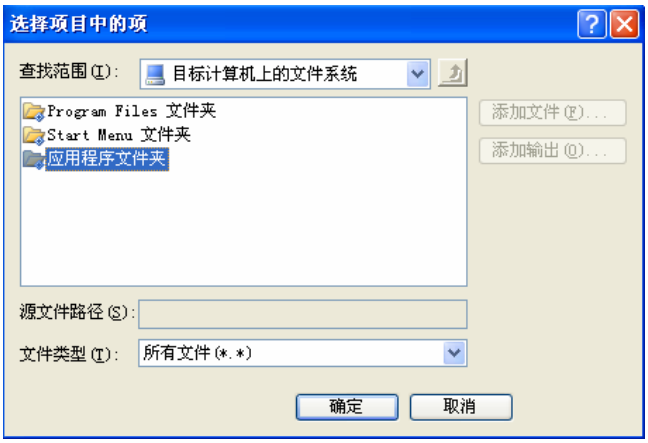

图 3-30　选择快捷方式的目标文件

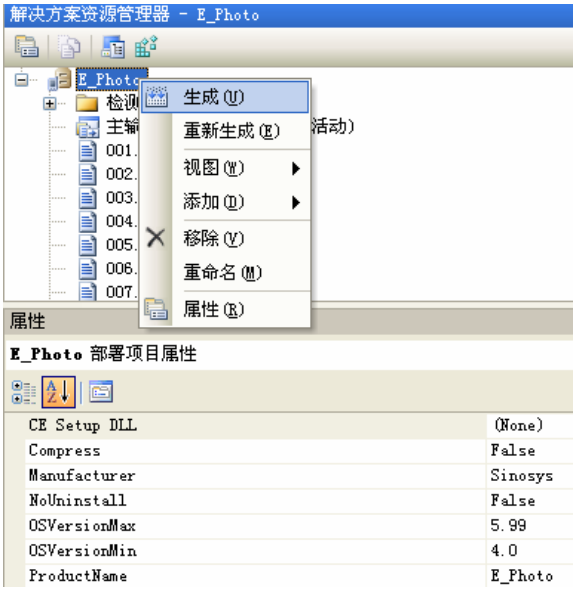

图 3-31　生成项目 CAB 包