第5章

图形图像编程

对于传统的程序设计语言来说，图形图像方向的程序设计是相对复杂和困难的。Microsoft 图形设备接口(Graphics Device Interface，GDI)解决方案提供了丰富的图形功能，方便了图形应用程序的开发。.NET 中使用了其升级版本 GDI＋，为便于充分利用 Windows 图形库，提供了一个新的接口。GDI＋以继承类的方式来完成图形处理，是一个完全面向对象的 2D 图形系统。GDI＋中的“＋”表示相对于 GDI 来说有了很大的改进，允许编写与设备无关的图形应用程序，如游戏、计算机辅助设计和绘图程序等。

本章将介绍如何在.NET 中建立基于 C＃的 GDI＋图形应用程序。

5.1 GDI＋绘图基础

5.1.1 GDI＋概述

图形设备接口(GDI)是一个可执行程序，它接受 Windows 应用程序的绘图请求(表现为 GDI 的方法调用)，并且将它们传给相应的设备驱动程序。GDI＋是对图形设备接口的一个扩展，它所提供的类可用于创建二维矢量图形、操纵字体以及插入图像。

1. GDI＋图形输出类型

使用 GDI＋可以创建 3 种类型的图形输出：矢量图形输出、光栅图形输出和文本输出。

① 矢量图形输出。矢量图形输出指的是创建线条和填充图形，包括点、线、多边形、扇形和矩形的绘制。

② 光栅图形输出。光栅图形输出是指用光栅图形方法对以位图形式存储的数据进行操作。在屏幕上表现为对若干行和列的像素的操作，它是直接从内存到显存的复制操作，在打印机上则是若干行和列的点阵的输出。Windows 在绘制界面时使用了大量的光栅输出。

③ 文本输出。Windows 是按图形方式输出文本。用户可以通过调用各种 GDI＋方法，创造出各种文本输出效果，包括加粗、斜体、设置颜色等。

程序员使用 GDI＋时不需要考虑 GDI＋内部是如何实现的，直接使用其提供的类进行编程即可。GDI＋在 System.Drawing.dll 动态链接库中定义，与其相关的名称空间如表 5-1 所示，其中最常用的名称空间是 System.Drawing，主要有 Graphics、Pen、Brush、Image、Bitmap 等类。

表 5-1　GDI＋相关名称空间

命 名 空 间	功　　能
System.Drawing	提供对 GDI＋基本图形功能的访问
System.Drawing.Drawing2D	提供高级的二维和矢量图形功能
System.Drawing.Imaging	提供高级 GDI＋图像处理功能
System.Drawing.Text	允许用户创建和使用多种字体

2. 编程步骤

通过 GDI＋进行图形图像编程的一般步骤为：构造画布、建立绘图对象、调用绘图方法以及释放绘图对象。

（1）构造画布

绘图前，先要准备好“画布”，可以使用 Graphics 类创建画布对象。通常情况下，直接在窗体上或在图像框中绘图，可以在相应的控件对象上构造画布。例如：

```
Graphics g = this.CreateGraphics();                    //以窗体为画布
```

表示在窗体上构造画布。又如：

```
Graphics g = pictureBox1.CreateGraphics();             //以图像框为画布
```

表示在图像框中构造画布。

（2）建立绘图对象

有了“画布”后，还需要准备“画笔”和“画刷”。通常情况下，画笔通过 Pen 类创建，可以用来绘制直线、曲线或闭合图形的外轮廓；画刷通过 Brush 类创建，可以用来填充闭合图形内部或输出文字。

① 建立画笔时，可以定义画笔的颜色和粗细，详见 5.2.1 节的介绍。例如：

```
Pen p = new Pen(Color.Blue, 2);                        //构造一支线宽为 2 的蓝色画笔
```

② 建立画刷时，需要根据绘图需求定义不同的画刷类对象（有单色刷、网格刷、纹理刷和渐变刷等多种），详见 5.2.2 节的介绍。例如：

```
SolidBrush sb = new SolidBrush(Color.Pink);            //构造一支粉红色的单色刷
```

③ 若要利用画刷输出文字，还需要创建字体对象，声明要输出文字的字体、字号和格

式等，详见 1.3.1 节的介绍。例如：

```
Font f = new Font("宋体", 10, FontStyle.Bold);          //创建字体对象
```

(3) 调用绘图方法

绘图工具(画布、画笔、画刷等)准备齐全后，就可以开始画图。

① 要绘制直线、曲线或闭合图形的外轮廓，使用 Graphics 类中以 Draw 开头的方法并结合画笔对象实现，详见 5.2.1 节的介绍。例如：

```
g.DrawEllipse(p, 10, 10, 100, 60);                      //用画笔 p 绘制椭圆
```

表示在左上角坐标为(10,10)、宽和高分别为 100 和 60 的矩形中，绘制其内切椭圆外轮廓。

② 要填充闭合图形，使用 Graphics 类中以 Fill 开头的方法并结合画刷对象实现，详见 5.2.2 节的介绍。例如：

```
g.FillEllipse(sb, 10, 10, 100, 60);                     //用画刷 sb 填充椭圆
```

表示在左上角坐标为(10,10)、宽和高分别为 100 和 60 的矩形中，填充其内切椭圆。

③ 要输出文本，使用 Graphics 类的 DrawString()方法，并且结合字体和画刷对象实现，详见 5.2.3 节的介绍。例如：

```
g.DrawString("GDI+绘图", f, sb, 50, 35);            //用画刷 sb 输出字体为 f 的文字
```

表示在坐标为(50,35)的位置输出文字“GDI+绘图”，文字颜色和字体样式等根据画刷 sb 和字体 f 指定。

(4) 释放绘图对象

图形绘制完成后，需要调用绘图对象的 Dispose()方法，释放内存资源，提高程序运行效率。例如：

```
p.Dispose();                                            //释放画笔对象
sb.Dispose();                                           //释放画刷对象
f.Dispose();                                            //释放字体对象
g.Dispose()                                             //释放画布对象
```

可以用一个较为形象的方法去理解.NET 中的图形绘制。“画布”就是手工绘图时用到的“纸”，“画笔”在绘图时可以用来画线条、勾轮廓，“画刷”在绘图时用来填充颜色、写字。

想一想，为什么文字的输出用的是画刷而不是画笔？因为在输出文字时，可以指定不同的字体，尤其是输出汉字时，可以是宋体、楷体、隶书等不同的形体。为完成这些形体的书写，在实际生活中，我们会选择使用毛笔，而这里用到的“画刷”是不是更像一支毛笔呢？

例 5.1 在窗体上绘制椭圆并填充，同时输出一行文字，程序运行界面如图 5-1 所示。

分析：

① 按钮单击时触发绘图程序，并且使用 Graphics 类在窗体上创建画布。

② 需要声明一个画笔对象来绘制椭圆外轮廓，声明两个画刷对象分别用来填充椭圆和输出文字，声明一个字体对象来指定输出文字的字体信息。

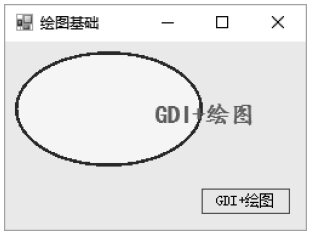

图 5-1 例 5.1 程序运行界面

③ 绘制椭圆外轮廓用 DrawEllipse()方法，填充椭圆用 FillEllipse()方法，输出文字用 DrawString()方法。

④ 图形绘制完成后记得释放各种绘图对象，节省内存空间。

程序源代码如下。

```
private void button1_Click(object sender, EventArgs e)
{
    //在窗体上构建画布
    Graphics g= this.CreateGraphics();

    //建立画笔对象,调用绘图方法绘制图形
    Pen p = new Pen(Color.Blue, 5);
    g.DrawEllipse(p, 10, 10, 150, 90);

    //建立画刷对象,调用绘图方法填充图形
    SolidBrush sb1 = new SolidBrush(Color.Yellow);
    g.FillEllipse(sb1, 10, 10, 150, 90);

    //建立画刷和字体工具,调用 DrawString()方法绘制文字
    SolidBrush sb2 = new SolidBrush(Color.Red);
    Font f = new Font("楷体", 15, FontStyle.Bold);
    g.DrawString("GDI+绘图", f, sb2, 120, 50);

    //调用各绘图对象的 Dispose()方法释放系统资源
    p.Dispose();
    sb1.Dispose();
    sb2.Dispose ();
    f.Dispose();
    g.Dispose();
}
```

5.1.2 坐标系

2D 图形的绘制需要一个可绘图的对象，如窗体、图形框。为了能定位图形，需要一个二维坐标系。在 GDI＋中，对象坐标系以像素为单位。像素是指屏幕上的亮点，即显示器

能分辨的最小单元，每个像素都有一个坐标点与之对应，如图 5-2 所示。默认的坐标原点为对象的左上角，横向向右为 X 轴的正向，纵向向下为 Y 轴的正向。例如，如果在窗体上绘画，那么默认坐标原点就在它的左上角(标题栏的左下方)。

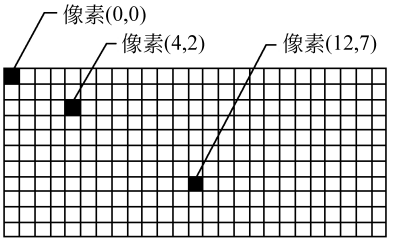

图 5-2　屏幕像素与坐标

GDI＋默认的坐标系统与数学中的坐标系统并不一样，在绘制数学函数 y＝f(x)的图形时，要使所画的图产生与数学坐标系相同的效果，需要在默认坐标的基础上进行坐标变换，如旋转、平移等。Graphics 对象提供了坐标变换方法，常用的坐标变换方法如表 5-2 所示。

表 5-2　Graphics 对象常用的坐标变换方法

方　法　名	功能	使 用 范 例	
TranslateTransform	平移	TranslateTransform(40,30)	将(40,30)设为原点
RotateTransform	旋转	RotateTransform(－30)	将坐标系逆时针方向旋转 30°
ScaleTransform	缩放	ScaleTransform(2,3)	按目前的宽的 2 倍和高的 3 倍放大
ResetTransform	还原	ResetTransform()	还原为默认坐标

例如，坐标系经过平移→旋转→缩放后的效果如图 5-3 所示。其变换参数如下。

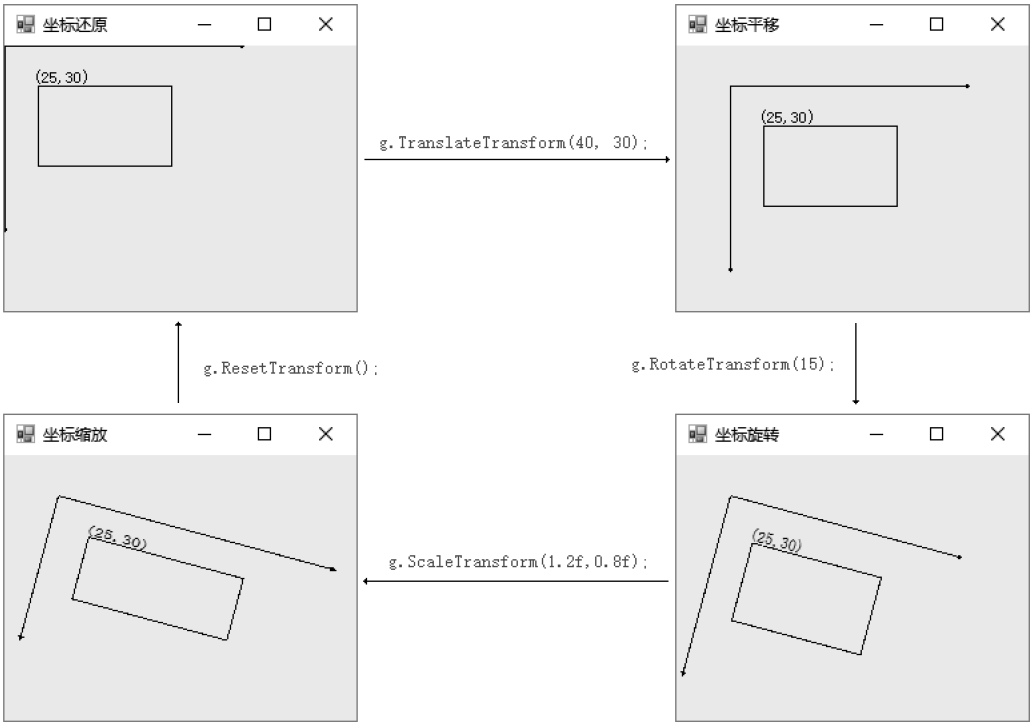

图 5-3　坐标变换后的效果

① 坐标原点平移至(40,30)。

② 坐标系顺时针方向旋转 15°。

③ 坐标系水平放大 20%,垂直缩小 20%。

注意,每次坐标变换后,再次变换坐标是在前一次变换的基础上进行的。图 5-3 中的矩形左上角坐标始终位于(25,30),宽和高分别为 100 和 60。

例 5.2 将例 5.1 中绘制的椭圆和文字经过如图 5-3 所示的坐标变换后,绘制在窗体上,程序运行界面如图 5-4 所示。

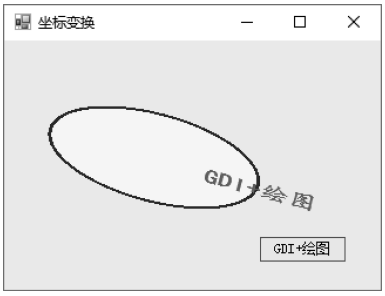

图 5-4 例 5.2 程序运行界面

分析:

① 按钮单击时触发绘图程序,并且使用 Graphics 类在窗体上创建画布。

② 使用如表 5-2 所示的 Graphics 对象常用的坐标变换方法,实现坐标系的平移、旋转和缩放。

③ 在窗体上建立画笔、画刷和字体对象,用相应的绘图方法绘制椭圆和文字。

程序源代码如下。

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();     //构建画布

    g.TranslateTransform(40, 30);           //坐标平移,原点平移至(40,30)
    g.RotateTransform(15);                  //坐标旋转,顺时针方向旋转 15°
    g.ScaleTransform(1.2f,0.8f);            //坐标缩放,水平 1.2 倍、垂直 0.8 倍

    ...                                     //省略图形绘制源代码,详见例 5.1
}
```

注意,用于坐标平移、旋转和缩放的 TranslateTransform()、RotateTransform()和 ScaleTransform()函数,其参数应为 float 数据类型。在调用这些方法时,若输入的实参为整数,则能自动转换成 float 类型;若输入的实参为小数,程序会默认视其为 double 数据类型,此时需要人工地将数据转换成 float 类型。例如,本例中有一个语句。

```
g.ScaleTransform(1.2f,0.8f);
```

通过在小数常量后加上类型符 f,实现了从 double 到 float 的数据类型转换。

5.1.3 Graphics 类

在图形图像应用程序开发过程中,最常用到的类有 Graphics、Pen、Brush 和 Font,如表 5-3 所示。这些类被包含在 System.Drawing 名称空间中。当建立 Windows 窗体应用程序后,系统会默认地将该名称空间通过 using 关键字引用进来。

```
using System.Drawing;
```

表 5-3　GDI＋常用类

类　　名	功　　能
Graphics 类	包含完成绘图的基本方法，如直线、曲线、矩形等
Pen 类	处理图形的轮廓部分
Brush 类	对图形进行填充处理
Font 类	字体功能，如字体样式、旋转等

用 GDI＋绘图时，必须先创建 Graphics 类的画布对象实例。只有创建了 Graphics 的实例后，才可以调用 Graphics 类的绘图方法。窗体和所有具有 Text 属性的控件都可以构成画布。创建 Graphics 画布对象有以下几种方法。

(1) 使用 CreateGraphics()方法

通过窗体或控件的 CreateGraphics()方法来获取对 Graphics 对象的引用，需要先定义一个 Graphics 类的对象，再调用 CreateGraphics()方法。这种方法一般应用于对象已经存在的情况，语法格式如下。

```
Graphics 画布对象;
画布对象 = 窗体或控件名.CreateGraphics();
```

上述语句也可以合成一句。

```
Graphics 画布对象 = 窗体或控件名.CreateGraphics();
```

(2) 利用 PaintEventArgs 参数传递 Graphics 对象

窗体或控件的 Paint 事件可以直接完成图形绘制。在编写 Paint 事件处理程序时，参数 PaintEventArgs 提供了图形对象，格式如下。

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;              //声明对象 g 并获取对 Graphics 对象的引用
    ...                                   //在画布 g 上绘制图形
}
```

将绘图程序放在 Paint 事件处理程序中，该事件在控件需要重新绘制时触发。在 Paint 事件结束时，会自动释放 Graphics 对象所占用的系统资源。因此，不必特地使用 Dispose()方法来释放绘图资源。

(3) 使用 Graphics.FromImage()方法从 Image 对象创建

该方法适用于需要处理已经存在的图像的场合。例如，利用图像文件 mypic.bmp 创建 Graphics 对象。

```
Bitmap b = new Bitmap(@"c:\mypic.bmp");   //根据图像文件声明 Bitmap 对象 b
Graphics g = Graphics.FromImage(b);       //将图像 b 作为画布对象 g
...                                       //在画布 g 上进行绘制,相当于修改图像 b
```

注意：在 C# 中，为了写文件路径时不加转义符“\”，可以在路径前面加上@标识符。例如，在以上代码中，通过用@"c:\mypic.bmp"代替路径字符串"c:\\mypic.bmp"，就能忽略转义字符。

此外，由于 Graphics 类的构造函数是私有的，因此不能直接实例化，即不能使用类似下面的语句来创建 Graphics 类的一个实例。

```
Graphics g = new Graphics();
```

绘图时，还会经常用到一些方法，如清理画布、刷新绘图控件、释放绘图对象等，这些方法如表 5-4 所示。

表 5-4　绘图时的常用方法

方法	说　明
Clear()	功能：清理画布对象 格式：画布对象.Clear(颜色) 范例 1：g.Clear(Color.Pink);　　//将画布对象 g 清理为粉色 范例 2：g.Clear(this.BackColor);　//将画布对象 g 清理为绘图控件(窗体对象)的原底色
Refresh()	功能：刷新绘图控件 格式：对象.Refresh() 范例：pictureBox1.Refresh();　　//刷新绘图控件 pictureBox1
Dispose()	功能：释放绘图对象 格式：绘图对象.Dispose 范例 1：g.Dispose();　　//释放画布 g 范例 2：sb.Dispose();　　//释放画刷 sb

例 5.3　编写一个简单的图像编辑程序，要求将图像文件 fruit.jpg 载入窗体后，以该图像作为画布，对其进行适当修改(即圈出水果篮中的葡萄)，并且能对编辑后的图像进行保存。程序运行界面如图 5-5 所示。

分析：

① 窗体载入时，通过 Bitmap 类创建图像对象，然后将其装入窗体背景图，调整窗体大小，使之与图像大小一致。

② 为实现图像的简单编辑，需要首先将图像文件作为画布，然后在画布上对该图像进行修改。可以使用上述第 3 种创建画布对象的方法，即使用 Graphics.FromImage()方法从 Image 对象创建；然后再声明绘图工具，进行图形绘制等后续工作。

③ 为实现图像文件的保存，需要建立一个“保存文件”对话框，获取保存路径；并且使用 Bitmap 类的 Save()方法来实现保存。格式如下。

```
图像对象.Save("保存路径");
```

(a) 图像编程前　　(b) 图像编辑后

图 5-5　例 5.3 程序运行界面

程序源代码如下。

```
Bitmap b;

private void Form1_Load(object sender, EventArgs e)
{
    b = new Bitmap("fruit.jpg");              //建立图像对象 b
    this.BackgroundImage = b;                 //图像装入窗体背景
    this.ClientSize = b.Size;                 //调整窗体大小
}

private void button1_Click(object sender, EventArgs e)
{
    Graphics g = Graphics.FromImage(b);       //在图像对象 b 上创建画布 g
    Pen p = new Pen(Color.Red, 5);            //创建画笔 p
    g.DrawEllipse(p, 250, 150, 140, 80);      //绘图,圈出葡萄
    this.Refresh();                           //刷新窗体,更新修改
    p.Dispose();
    g.Dispose();
}

private void button2_Click(object sender, EventArgs e)
{
    saveFileDialog1.FileName = "fruit";
    saveFileDialog1.Filter = "JPEG 图像|*.jpg|PNG 图像|*.png|BMP 图像|*.bmp";
    if (saveFileDialog1.ShowDialog() == DialogResult.OK) b.Save
(saveFileDialog1.FileName);                   //保存图像
}
```

```
private void button3_Click(object sender, EventArgs e)
{
    Form1_Load (sender,e);
}
```

思考：在以上程序源代码中，语句 this.Refresh()；起到了什么作用？是否能将其去除？

由于原始图像在编辑前已经装入窗体显示，然后通过画布 g 对图像的编辑实际上是对内存中创建的 Bitmap 对象 b 的修改。当修改完成后，为了使窗体上的显示也得到同步的更新，需要使用语句 this.Refresh()；刷新窗体，同步显示图像的更新效果，因此该语句不能去除。

注意：本程序在调试时，若选择的另存位置和图像载入的路径相同，则无法覆盖保存，原因和解决方法可以参考 5.3.4 节的介绍及例 5.11 中的程序源代码。

5.1.4 GDI+中常用的数据类型

进行图形图像编程时，需要使用相关基础类型与结构类型来表示位置、大小、点、矩形等，表 5-5 列出了常用的数据类型。

表 5-5 常用数据类型

类 型 名	说　　明
Point	功能：表示一个二维坐标点(X,Y) 声明方法：Point pt = new Point(X,Y)　　//X、Y 为 int 型 范例：Point pt = new Point(10, 20);　　//定义坐标 pt 为(10, 20)的点
PointF	与 Point 相似，坐标点 X、Y 为 float 型
Size	功能：用 W(宽度)和 H(高度)两个属性来表示尺寸大小 声明方法：Size s = new Size(W, H);　　//W、H 为 int 型 范例：Size s = new Size(30, 50);　　//定义宽为 30 和高为 50 的尺寸大小
SizeF	与 Size 相似，W 和 H 均为 float 型

Rectangle	功能：定义一个矩形区域，以(X,Y)或 PT 为左上角坐标，以 W 为宽和 H 为高或者以 SZ 为大小
	声明方法 1：Rectangle rect = new Rectangle(X, Y, W, H)　//X、Y、W、H 均为 int 型
	声明方法 2：Rectangle rect = new Rectangle(PT,SZ)　//PT 为 Point 型，SZ 为 Size 型
	范例 1：Rectangle rect = new Rectangle(20, 30, 10, 15);
	//创建一个左上角坐标为(20,30)、宽度为 10、高度为 15 的矩形区域 rect
	范例 2：Rectangle rect = new Rectangle(PT,SZ);//PT 和 SZ，前已声明，表示位置和大小
	//创建一个以 PT 为左上角坐标、SZ 为大小的矩形区域 rect

RectangleF	与 Rectangle 相似，X、Y、W、H 均为 float 型，或者 PT 为 PointF 型、SZ 为 SizeF 型