第 7 章　图像与文字处理

本章重点：

- 画布的属性和方法的使用
- 坐标系统和映射模式

在前面的几章中，我们已经使用了画布属性（Canvas），实际上，应用程序的窗体可以看成一个画布（Canvas），窗体上的显示组件不过是画在它上面的图形，而完成图形绘制工作的就是 Canvas 属性。并不是只有窗体才有画布属性，很多组件都有 Canvas 属性，例如图像和面板组件等。程序员若能充分利用画布属性的强大功能，就能很容易地设计出更具有人性化的用户界面。本章将向读者介绍使用画布和 Windows GDI（Graphics Device Interface）函数显示图像和文本的基本方法。

7.1　画布的属性（Canvas）

画布是用于绘制图形和显示文字信息的区域，但并不是每个组件都有画布属性，不过由 TForm、TImage 和 TGraphic 等类生成的对象都有画布属性（Canvas）。利用这个属性，我们可以在相应的组件上完成图形绘制和文本显示等操作。为了在画布上绘制用户满意的图形，通常需要设置画布的画笔、像素、画刷、字体和复制模式等属性，下面分别讲解这些属性的用法。

7.1.1　画笔（Pen）

画笔是画布属性的子对象，通过画笔对象可设置画线的颜色（Color）、线型（Style）、宽度（Width）和模式（Mode）等。

1. 画笔的颜色（Color）

Color 是画笔的颜色属性，用于指定画笔画线的颜色，此属性可使用 Delphi 预定义的颜色值，如 clRed（红色）和 clYellow（黄色）等。将画笔的颜色设置为蓝色的代码如下：

```
Canvas.Pen.Color:＝clBlue;
```

如果不使用 Delphi 预定义的颜色值，那么如何使用代码设置画笔的颜色呢？其实 Delphi 已经为我们准备好了解决办法，即用长整数表示颜色，由于屏幕上显示的颜色是由红、绿、蓝三种颜色组合而成的，颜色的组合工作可使用 Delphi 的 RGB 函数来完成，它的使用格式是：

RGB(R，G，B)

其中，参数 R、G 和 B 分别表示红、绿、蓝 3 个颜色值，这 3 种颜色值的取值范围是 0~255，其中 RGB(255，0，0)、RGB(0，255，0)和 RGB(0，0，255)分别返回红色、绿色和蓝色。

Delphi 组件的颜色属性是 TColor 类型的，而由 RGB 函数生成的颜色值却是一个长整型值，因此，当使用 RGB 为组件的颜色属性赋值时，必须将它强制转换成 TColor 类型。例如，将画笔的颜色设为随机颜色的代码如下：

```
Canvas.Pen.Color:=TColor(RGB(Random(255)，Random(255)，Random(255)))；
```

2. 画笔的线型（Style）

Style 属性用于设置画笔画线的线型，表 7-1 列出了画笔线型的可能取值及其功能。

表 7-1　画笔的线型

线 型 值	功　　能	线 型 值	功　　能
PsClear	不画线	psDot	点线
PsDash	画虚线	psInsideFrame	内框线
PsDashDot	画点划线	psSolid	实线
PsDashDotDot	画双点划线		

下面代码将画笔线型改变成点划线：

```
Canvas.Pen.Style:＝psDashDot；
```

图 7-1 显示了使用不同画笔线型在窗体画布上的画线情况。

图 7-1　画笔的画线线型

3. 画笔宽度（Width）

Width 属性用于设置画笔画线的粗细(以像素为单位)。此属性值越大，画笔画出来的线就越粗。

注意：点划线只适用于宽度为 1 的画笔。当宽度设为大于 1 的整数时，则画出的点划线是实线。

4. 画笔的模式（Mode）

画笔的 Mode 属性用于设置画线颜色与画布颜色相互作用的方式。表 7-2 中列出了 Win32 预定义的 16 种模式。

表 7-2　画笔的模式及其作用

模 式 值	含　　义	操作结果
PenBlack	总是黑色	0
pmWhite	总是白色	1
pmNOP	不改变画笔的颜色	D
pmNOT	将画布上的颜色反转	not D
PmCopy	使用画笔的颜色	S
pmNotCopy	使用画笔的反转颜色	not S
PmMergePenNot	将画笔颜色与画布颜色反转后的颜色进行或操作	S or not D

续 表

模 式 值	含 义	操作结果
pmMaskPenNot	将画笔颜色和画布颜色反转后的颜色进行与操作	S and not D
pmMergeNotPen	将屏幕颜色和画笔颜色反转后的颜色进行或操作	not S or D
pmMaskNotPen	将屏幕颜色和画笔颜色反转后的颜色进行与操作	(not S) and D
PmMerge	将画笔颜色和屏幕颜色进行或操作	S or D
PmNotMerge	将 pmMerge 反转	not(S or D)
pmMask	将画笔和屏幕颜色进行与操作	S and D
pmNotMask	将 pmMask 反转	not(S and D)
pmXor	将画笔颜色与画布颜色进行异或操作	S XOR D
PmNotXor	将 pmXor 反转	not(S XOR D)

注：表中的 D 表示画布上的颜色，S 表示画笔的颜色。

在一般情况下，画笔模式使用 pmCopy 作为默认模式。为了让读者更好地理解画笔的相关属性及其用法，下面给出一个较综合的应用实例（见清单 7-1），在这个例子中使用了本节上面所讲的各个属性，请读者认真体会它们的用法。

清单 7-1 画笔的颜色、宽度、线型和模式的演示程序（参见图 7-2、图 7-3）

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure  Button1Click(Sender: TObject);
    procedure  Button2Click(Sender: TObject);
    procedure  Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    procedure ClearScreen;       //清屏
    procedure SetPenDefault;     //设置画笔
```

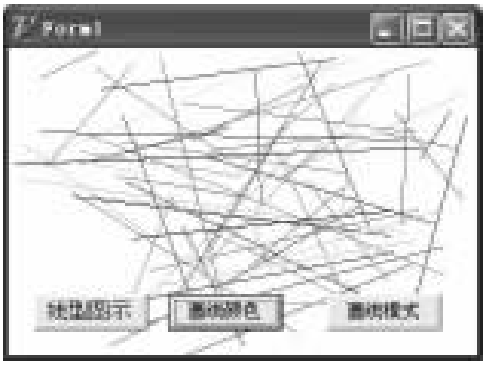

图 7-2 画线的颜色

```
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

Procedure  TForm1.Button1Click(Sender: TObject);
var
  yPos: integer;
  PenStyle: TPenStyle;       //定义枚举变量
begin
  ClearScreen;               //清除屏幕
  SetPenDefault;             //设置画笔
  YPos := 20;                //设置初始纵坐标
  with Canvas do
  begin
    for PenStyle := psSolid to psInsideFrame do
    begin
      Pen.Color := clBlack;
      Pen.Style := PenStyle;
      brush.Color:=clred;
      pen.Width:=4;              //设置画笔的宽度为 4
      MoveTo(100, yPos);
      LineTo(ClientWidth-20, yPos);
      inc(yPos, 20);
    end;
    //显示每种线型的标题
    TextOut (10, 10,'psSolid');
    TextOut(10, 30, 'psDash ');
    TextOut(10, 50, 'psDot');
    TextOut(10, 70, 'psDashDot ');
    TextOut(10, 90, 'psDashDotDot ');
    TextOut(10, 110, 'psClear');
    TextOut(10, 130, 'psInsideFrame');
  end;
```

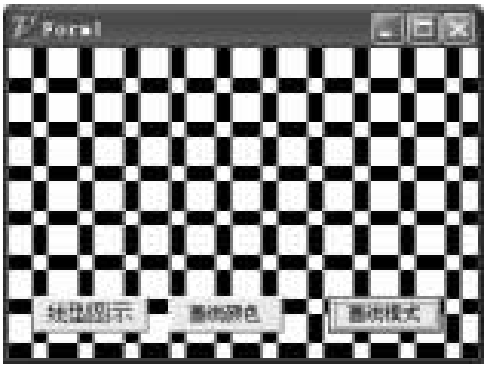

图 7-3　画笔模式设置为 PmNot 的图示

```
end;
procedure TForm1.ClearScreen;     //清除屏幕
begin
   with Canvas do
   begin
      Brush.Style := bsSolid;
      Brush.Color := clWhite;
      FillRect(ClientRect);              //这里的 ClientRect 为窗体的用户可设置区
   end;
end;

procedure TForm1.SetPenDefault;
begin
   with Canvas.Pen do
   begin
      Width := 1;
      Mode := pmCopy;
      Style := psSolid;
      Color := clBlack;
   end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
   i: integer;
begin
   ClearScreen;
   SetPenDefault;
   with Canvas do
   begin
      for i:= 1 to 60 do
      begin
         {使用随机产生的颜色值画线，这里的 ClientWidth 和 ClientHeight
           为窗体用户可设置区的宽度和高度}
         Pen.Color := RGB(Random(256),Random(256), Random(256));
         MoveTo(random(ClientWidth), Random(ClientHeight));
         LineTo(random(ClientWidth), Random(ClientHeight));
      end
   end;
end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
var
  x,y: integer;
begin
  ClearScreen;
  SetPenDefault;
  y := 10;
  canvas.Pen.Width := 20;
  canvas.pen.Mode:= pmNot;          //设置画线模式
  while y < ClientHeight do
  begin
    canvas.MoveTo(0, y);
    canvas.LineTo(ClientWidth, y);
    inc(y, 30);                     //增加画线垂直间距
  end;
  x:= 5;
  while x < ClientWidth do
  begin
    Canvas.MoveTo(x, 0);
    canvas.LineTo(x, ClientHeight);
    inc(x, 30);                     //增加画线水平间距
  end;
end;
end.
```

说明：在上述程序的窗体上有三个按钮，用于完成画笔的线型、颜色和模式的设置。为了增强程序的可阅读性，在程序中定义了 ClearScreen 和 SetPenDefault 两个子程序，用于画布刷新和画笔的初始化。此外，程序中还使用了画布方法，如 LineTo 和 TextOut 等，用于画线和显示文本，这两个方法的具体用法将在第 7.2 节详细说明。

7.1.2 像素（Pixels）

画布的像素属性是一个二维数组，用于表示窗体某个像素的颜色值。其使用格式为：

Canvas.Pixels[x,y]

其中 x、y 分别表示窗体上的横、纵坐标。

例如，将窗体左上角的像素设为红色、右下角的像素设为蓝色，其代码如下：

```
Canvas.Pixels[0,0]:=clRed;
Canvas.Pixels[ClientWidth，ClientHeight]:=clBlue;
```

在一般情况下，很少访问窗体的单个像素，因为访问它的速度太慢了。

7.1.3 画刷（Brush）

画布的画刷用于填充画布上的区域和图形。与画笔不同的是：画笔用于在画布上画线，而画刷则是使用不同的颜色、样式和形状来填充画布的指定区域。

Brush 属性是画布的子对象，它有多个子属性，如 Color、Style 和 Bitmap 等，其中，Color 用于设置画刷的颜色；Style 用于设置画刷刷出的图形样式；Bitmap 则用于指定画刷刷出的图形。

画刷的 Style 属性有八个可能的值，分别为 bsSolid、bsClear、bsHorizontal、bsVertical、bsFDiagonal、bsBDiagonal、bsCross 和 bsDiagCross，它们对应的样式图案如图 7-4 所示。在默认情况下，画刷的颜色为 clWhite，样式为 bsSolid，没有位图。

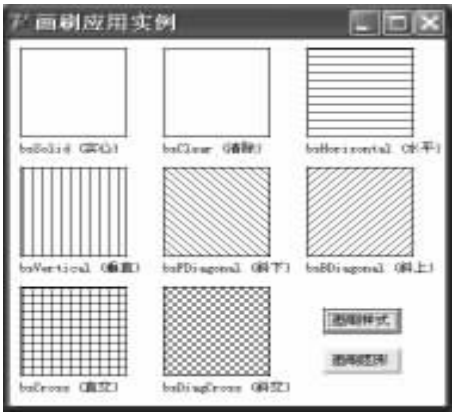

图 7-4 画刷样式图示

图 7-5 画刷位图属性应用实例

下列程序是画刷属性的应用实例（参见图 7-5），见清单 7-2。

清单 7-2 画刷属性的应用实例

```
unit Brush1;

interface

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

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
     procedure ClearScreen;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var x,y,i,wh:word;
    BrushStyleName:variant;
Begin
  //创建动态变体数组
  BrushStyleName:=VarArrayOf(['bsSolid (实心)','bsClear (清除)',
                              'bsHorizontal (水平)','bsVertical (垂直)',
                              'bsFDiagonal (斜下)', 'bsBDiagonal (斜上)',
                              'bsCross (直交)', 'bsDiagCross (斜交)']);
  ClearScreen;
  with Canvas do
  begin
    x:=10;y:=10;wh:=90;
    Brush.Color:=clwhite;
    for i:=0 to 7 do
    begin
      if (i>0) and (i mod 3=0) then
      begin
        y:=y+wh+30;
        x:=10;
      end;
      Brush.Style := TBrushStyle(i);          //设置画刷样式
      if i>1 then
```

```
            Brush.Color:=clblue;
          Rectangle(x,y,x+wh,y+wh);                    //按指定的样式画矩形
          TextOut(x, y+wh+5, BrushStyleName[i]);      //显示每个样式的标签
          x:=x+wh+30;
        end;
    end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var     Picture: TBitmap;
begin
    ClearScreen;
    Picture:=tBitmap.Create;
    Picture.LoadFromFile('C:\Windows\Soap Bubbles.bmp');
    Canvas.Brush.Bitmap := picture;
    try
      Canvas.Rectangle(0, 0, ClientWidth, ClientHeight);          //使用画刷中的位图填充窗
                                                                    体客户区
    finally
      Canvas.Brush.Bitmap := nil;
    end;
     Picture.Free;
end;

procedure TForm1.ClearScreen;     //清屏
begin
    with Canvas do
    begin
      Brush.Style := bsSolid;
      Brush.Color := clWhite;
      Rectangle(0, 0, ClientWidth, ClientHeight);
    end;
end;
end.
```

说明：上述程序中使用了画布的 Rectangle 方法，其功能是画一个矩形，它的定义格式是：Rectangle(X1, Y1, X2, Y2: Integer)；其中（X1,Y1）和（X2,Y2）分别表示所画矩形左上角和右下角的坐标。

为了使用图形文件中的数据，程序中还定义了名为 Picture 的位图变量。

注意：使用位图变量前必须先实例化，使用后还必须释放它所占的内存。Bitmap 属性用

于指定画刷使用 Bitmap 属性中的图案填充绘画区域。用位图作为画刷图案，不仅适用于窗体上的画布，也适用于所有具有 Canvas 属性的组件。

7.1.4 字体(Font)

Font 属性用于设置画布上显示的字体，包括字体的颜色（Color）、名称(Name)、字号(Size)、和字型(Style)等。

1. 设置字体颜色（Color）

设置画布字体的颜色可使用的格式为：

Canvas.Font.Color:=颜色值;

这里的“颜色值”既可以使用预定义的颜色值，如 clRed、clBlue 和 clGreen 等，也可以使用由 RGB 函数生成的颜色值。

例如：将画布字体的颜色设置为黄色的程序代码是：Canvas.Font.Color:=clYellow;

2. 设置字体名称（Name）

字体名用于设置要显示的字体，常用的中文字体名有“宋体”、“楷体 2312”、“黑体”、“新宋体”和“隶书”等；常用的英文字体有“Times New Roman”等。

需要指出的是：当中文字体名前有一个“@”符号时，表示使用的是沿逆时针方向旋转 90°的字体，这种字体在定义 Chart 组件的标题时特别有用。设置画布字体的格式为：

Canvas.Font.Name:= ‘ 字体名’;

例如：设置 Form1 窗体画布的字体为“宋体”，程序代码为：

Form1.Canvas.Font.Name:=‘宋体’;

3. 设置字号（Size）

Size 属性用于设置字体的字号，字号的单位是磅。需要注意的是：字号和字体的高度有一个对应关系，关系式为：Font.Height = －Font.Size * Font.PixelsPerInch / 72。其中，Font.PixelsPerInch 表示屏幕（或打印机）每英寸的像素数，常用有 96（显示器的常用尺寸）和 120（大尺寸）两种，它是字体的只读属性（在 Windows 下通过屏幕显示属性→设置→高级→常规→DPI 可设置此属性的大小），因此，若设置字体的字号为正值，则其高度值就为负值，反之亦然。

例如：若屏幕的 DPI（dots per inch）为 96，设置 Form1 窗体画布的字体为宋体五号字的代码如下：

```
With Form1.Canvas.Font do
begin
Name:=‘宋体’;
Size:=11;
Height:=−14;
end;
```

注意：尽管字体的高度与大小有计算公式，但由于字体属性使用的是计算结果的整数值，因此，仅设置字体的Size属性值还不能精确确定字体的字号。如上例中，若仅设置Size为 11，系统将使用通过计算得到的Height值−14，而这个值并不是五号字的准确高度，因此设置字体的Height属性值也是必要的。

4. 设置字型（FontStyle）

对字型的设置有四种，分别是 fsBold、fsItalic、fsUnderline 和 fsStrikeOut，它们的功能如下：

fsBold 表示粗体。
fsItalic 表示斜体。
fsUnderline 表示下划线字体。
fsStrikeOut 表示有水平删除线的字体。

由于字型是一个集合类型的属性，因此可用上述四个值构成的集合作为字型属性的值，例如设置 Form1 窗体的字体为粗斜体，设置方法为：

```
Form1.Canvas.Font.Style:=[fsBold, fsItalic];
```

7.1.5 复制模式（CopyMode）

画布的复制模式属性用于设置将其他图像复制到当前画布的方式。例如，若将 CopyMode 的值设为 cmSrcCopy，则表示把源图像完全复制到当前画布上，如果将画布模式设为 cmSrcInvert，则复制时将把源图像和目标画布上的图像按位进行 XOR 运算。在 Windows 标准单元中定义了常用的复制模式值，这些值的功能见表 7-3。

表 7-3 画布的复制模式及其功能

模 式 值	功 能
cmBlackness	使用黑色填充目标画布
cmDstInvert	将画布中的图像反转
cmMergeCopy	使用 AND 运算，将源位图合并到目标画布中
cmMergePaint	使用 OR 运算，将反转后的源位图合并到目标画布中
cmNotSrcCopy	将源位图反转后复制到画布中
cmNotSrcErase	使用 OR 运算，将源位图与目标画布中的图像合并，并将合并后的结果再反转
cmPatCopy	复制源图案到目标画布中
cmPatInvert	使用 XOR 运算，将源图案与目标画布中的图像合并
cmPatPaint	使用 OR 运算，将源位图反转后与源图案合并，将合并后的结果再与目标画布上的图像按 OR 运算再合并
cmSrcAnd	使用 AND 运算，将画布图像与源位图合并
cmSrcCopy	将源位图复制到画布
cmSrcErase	使用 AND 运算，将画布的反转图像与源位图合并
cmSrcInvert	使用 XOR 运算，将画布的图像与源位图合并
cmSrcPaint	使用 OR 运算，将画布的图像与源位图合并
cmWhiteness	用白色填充画布

清单 7-3 是不同的复制模式的一个应用实例，图 7-6 则显示的是含有两个椭圆的窗体。其中 TComboBox 组件中保存了用户选择的复制模式值，单击“复制源图像”按钮，则程序将源图像按照用户设置的模式复制到目标区中。

清单 7-3　不同的复制模式应用实例（参见图 7-6）

```
unit CopyModeUnit1;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,Dialogs,StdCtrls,
ExtCtrls;

type
  TForm1 = class(TForm)
    SImage: TImage;
    DImage: TImage;
    ComboBox1: TComboBox;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
  private
        { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  CM:integer=cmBlackness ;          //定义保存图像模式的变量
implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);          //在图片组件上画椭圆
begin
  with Simage.Canvas do          //画源图像
  begin
    Brush.Style := bsSolid;
    Brush.Color := clWhite;
    Rectangle(0,0,Simage.Width,Simage.height);
```

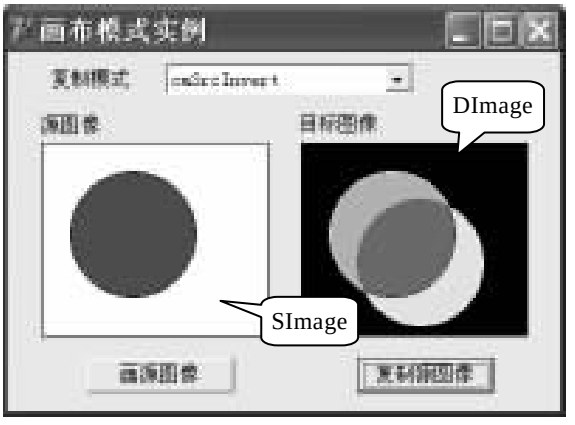

图 7-6　画布模式演示程序主窗体

```
    Brush.Color := clRed;
    Ellipse(20, 20, 110, 110);
  end;
  with Dimage.Canvas do          //画目标图像
  begin
    Brush.Style := bsSolid;
    Brush.Color := clWhite;
    Rectangle(0,0,Dimage.Width,Dimage.height);
    Brush.Color := clBlue;
    Ellipse(40, 40, 130, 130);
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);          //将 Simage 中的图像复制到
                                                           Dimage 中
var
  DRect,SRect: TRect;
begin
  Dimage.Canvas.CopyMode :=CM;        //设置复制模式
  DRect:=Dimage.Canvas.ClipRect;      //设置目标剪切区
  SRect:=Simage.Canvas.ClipRect;      //设置源剪切区
  Dimage.Canvas.CopyRect(Drect, Simage.Canvas, Srect);        //图像复制
end;

procedure TForm1.ComboBox1Change(Sender: TObject);          //将用户的选择转化成复制
                                                             模式值
begin
  Case combobox1.ItemIndex Of
    0:   CM:=cmBlackness;
    1:   CM:=cmDstInvert;
    2:   CM:=cmMergeCopy;
    3:   CM:=cmMergePaint;
    4:   CM:=cmNotSrcCopy;
    5:   CM:=cmNotSrcErase;
    6:   CM:=cmPatCopy;
    7:   CM:=cmPatInvert;
    8:   CM:=cmPatPaint;
    9:   CM:=cmSrcAnd;
    10: CM:=cmSrcCopy;
    11: CM:=cmSrcErase;
    12: CM:=cmSrcInvert;
```

```
        13: CM:=cmSrcPaint;
        Else
          CM:=cmWhiteness;
      end;
  end;

  end.
```

说明：

- 在上述程序中使用了画布的 Ellipse 方法，用于在指定画布上画椭圆，此方法的定义格式为：procedure TCanvas.Ellipse(X1, Y1, X2, Y2：Integer); 其中，(X1,Y1)和(X2,Y2) 分别表示所画椭圆在矩形区域左上角和右下角的坐标。
- 由于画布的模式值是一个长整数，本程序将模式名放在 ComboBox 组件的 Items 属性中，然后通过此组件的 OnChange 事件将用户选择模式值自动转化为程序可用的模式值，另外，ComboBox 组件的 Style 属性最好设置成 csDropDownList，表示是下拉选择框；设置 ComboBox 组件显示初始值为 Items 属性中的第一行内容（即设置 ComboBox 组件的 Itemindex 属性值为 0）。
- 程序中还使用了画布的 ClipRect 方法，它表示画布的剪切矩形区，只有绘在剪切区中的图画才能显示。（在默认情况下，画布的剪切区为整个画布，而且它可通过 CreateRectRgn 和 SelectClipRgn 来设置。）

习题：

1. 在窗体上放入一个 Image 组件，请用 C:\Windows\Soap Bubbles.bmp 中保存的图片填充此组件的图片显示区。

2. 图像模式可在视频压缩中起到独特作用，假设有两帧非常相似的画面，请你设计一个保存它们的优化方案，并说明为什么。若考虑网络视频传输问题，你能更进一步分析图像模式的独特作用吗？

7.2 画布的方法

画布对象提供了大量与 Windows 图形驱动接口（GDI）函数相对应的方法，使用这些方法，可以在画布上画线、画图形、显示文字等操作。

7.2.1 画线

画布的画线方法有很多，例如在第 7.1 节例子中使用的 LineTo 就是画布的画直线方法，LineTo 的使用格式为：

LineTo(X,Y)；

上述格式的功能是从画笔的当前位置开始向画布的（X,Y）坐标画一条直线。

画笔的当前位置可由画布的 MoveTo 方法设定，其使用格式为：

MoveTo(X,Y)

此格式完成的功能是设置（X,Y）坐标为画笔的开始位置。

例如：在 Form1 窗体上，从（0,0）点向(100,100)点画一条直线的程序代码如下：

```
Form1.Canvas.MoveTo(0,0);          //设置画笔的开始位置
Form1.Canvas.LineTo(100,100);      //从画笔的当前位置开始向（100,100）处画直线
```

注意：画线的粗细由画布的 Pen.Width 来确定。

常用的画线方法及其使用格式见表 7-4。

表 7-4 常用的画线方法及其使用格式

画线方法	功 能	使 用 格 式
Arc	画椭圆弧	Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer)
Chord	画扇形	Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer)
Draw	画图片	Draw(X, Y: Integer; Graphic: TGraphic)
Ellipse	画椭圆	Ellipse(X1, Y1, X2, Y2: Integer)
FrameRect	画矩形边框	FrameRect(Const Rect: TRect)
Pie	画饼图	Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
PolyBezier	画 Bezier 曲线	PolyBezier(const Points: Array of TPoint)
Polygon	画多边形	Polygon(Points:Array of TPoint)
Polyline	画折线	Polyline(Points:Array of TPoint)
Rectangle	画矩形	Rectangle(X1, Y1, X2, Y2: Integer)
RoundRect	画圆角矩形	RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer)
StretchDraw	画可伸展图片	StretchDraw(Const Rect: TRect; Graphic: TGraphic)

说明：

- 在画线使用的格式中，参数（X1, Y1, X2, Y2, X3, Y3, X4, Y4:Integer）用于设置画线的范围。其中，前四个参数用于限定画线的矩形区域，（X3,Y3）和（X4,Y4）坐标用于确定画线的开始点和结束点。确定起始和结束点的方法是从椭圆的中心到（X3,Y3）或(X4,Y4)点连线，连线与椭圆的交点即为画线的开始点和结束点。
- 画线方法使用的格式中，参数（X1, Y1, X2, Y2, X3, Y3: Integer）也用于确定画线的范围。其中，前四个参数用于限定画线的矩形区域，X3 和 Y3 分别表示要画椭圆的宽和高，圆角矩形的圆角就是此椭圆的一部分。

Points: Array of TPoint 表示由点构成的数组。

下面通过一个实例（见清单 7-4）来说明画布各个方法的具体应用。

清单 7-4 画布的属性和方法应用实例（参见图 7-7）

```
unit DrawLineUnit1;

interface

uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Spin;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Image1: TImage;
    Label1: TLabel;
    SpinEdit1: TSpinEdit;
    ColorBox1: TColorBox;
    Label2: TLabel;
    Button9: TButton;
    Button10: TButton;
    Button11: TButton;
    procedure Button1Click(Sender: TObject);
    procedure SpinEdit1Change(Sender: TObject);
    procedure ColorBox1Change(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
    procedure Button11Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

图 7-7 画线实例的主窗体

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);   //画直线
begin
  Image1.Canvas.MoveTo(20,20);
  Image1.Canvas.LineTo(200,100);
end;

procedure TForm1.SpinEdit1Change(Sender: TObject);   //修改画笔的宽度
begin
  Image1.Canvas.Pen.Width:=SpinEdit1.Value;
end;

procedure TForm1.ColorBox1Change(Sender: TObject);   //改变画刷的样式和颜色
begin
  Image1.Canvas.Brush.Style:=bsSolid;
  Image1.Canvas.Brush.Color:= ColorBox1.Selected;
end;

procedure TForm1.Button2Click(Sender: TObject);    //画圆弧
begin
  Image1.Canvas.Arc(30,40,300,120,10,80,220,80);
end;

procedure TForm1.Button9Click(Sender: TObject);    //清屏
begin
  with Image1.Canvas do
  begin
    Brush.Style := bsSolid;
    Brush.Color := clWhite;
    FillRect(ClientRect);
    Brush.Color:=ColorBox1.Selected
  end;
end;
```