交互式 CAD 系统开发基础系列丛书

用 VB.NET 和 VC#.NET 开发交互式 CAD 系统

苏金明　周建斌　编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书主要结合 VB.NET 和 VC#.NET 两种语言介绍了创建交互式 CAD 系统的基本思路和技术，分别给出了两种语言的小系统完整代码，并讨论了技巧实现的其他可能性以及系统代码的改进方法。

本书前 3 章主要介绍语言基础和.NET 框架基础，第 4 章至第 8 章结合一个 CAD 小系统的创建详细地介绍了交互式 CAD 系统的组织思路和基本技术，第 9 章至第 11 章介绍了更多的技巧实现方法和系统优化方法，第 12 章结合 CAD 编程进行了一些设计模式方面的讨论；写作过程中注意了循序渐进的讲解原则，内容适合不同学习阶段的读者。

书中所有示例程序都通过调试，并放在随书的光盘上，以便于学习和交流。

本书可供从事图形学、CAD 技术以及编程技术的有关工程技术人员、程序员、大学生、研究生阅读参考。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

用 VB.NET 和 VC#.NET 开发交互式 CAD 系统/苏金明，周建斌编著. —北京：电子工业出版社，2004.1
(交互式 CAD 系统开发基础系列丛书)
ISBN 7-5053-9443-6

Ⅰ. 用… Ⅱ. ①苏…②周… Ⅲ. ①BASIC 语言—程序设计②C 语言—程序设计③计算机辅助设计
Ⅳ. ①TP312②TP391.72

中国版本图书馆 CIP 数据核字（2003）第 112512 号

责任编辑：王昌铭
印　　刷：北京金特印刷厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱　邮编　100036
经　　销：各地新华书店
开　　本：787×1 092　1/16　印张：22.5　字数：576 千字
印　　次：2004 年 1 月第 1 次印刷
印　　数：5 000 册　　定价：38.00 元　（含光盘 1 张）

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前　　言

交互式 CAD 技术是目前应用十分广泛的一种绘图技术，它的主要特点是可以把创建和编辑图形的动态过程展示出来，因而具有更强的交互性能。为了使更多的人了解和掌握这一技术，我们编写了这套交互式 CAD 系统开发基础系列丛书。丛书一共 3 本，分别结合不同的语言进行介绍。本书是其中的第 3 本，主要介绍用 VB.NET 和 VC#.NET 开发交互式 CAD 系统。

本书的内容

本书结合 VB.NET 和 VC#.NET 两种语言介绍了创建交互式 CAD 系统的基本思路和技术。

第 1 章至第 3 章主要介绍后面各章可能用到的语言基础和.NET 框架基础。其中：第 1 章对.NET 进行了比较简略的介绍；第 2 章介绍了 VB.NET 和 VC#.NET 的面向对象编程方法，需要指出的是，VB.NET 已经是完全面向对象的编程语言了；第 3 章介绍了 GDI+ 编程的一些方法和效果，GDI+ 是 GDI 的改进版本。

第 4 章至第 8 章结合一个 CAD 小系统的创建详细地介绍了交互式 CAD 系统的组织思路和基本技术。这部分的第 4 章介绍了系统中对象的提取、组织及代码实现情况，给出了一些相关的 UML 类图；然后建立通用坐标系，确定它与页面坐标系、设备坐标系之间的转换关系。第 5 章和第 6 章介绍通过建立基本图元类和交互绘图类来实现图元的鼠标交互绘制。第 7 章介绍在基本图元类中添加图元拾取的方法，添加选择图元的交互绘图类，实现图元的拾取、选择和删除。第 8 章讲解了通过对图元的控制点进行几何变换来实现图元变换，包括图元的平移变换、旋转变换、镜像变换和比例变换。

第 9 章介绍了 GDI+提供的一些交互绘图技巧，包括线形图元和区域的拾取、图元的几何变换等。

第 10 章介绍与图元相交关系有关的一些交互技术，给出了一个相交线与直线段相交的实例，讨论了用矩形框拾取图元的思路和方法，最后结合 GDI+提供的对象方法介绍了计算其他图元与曲线图元交点的思路和方法。

第 11 章介绍了对第 4 章至第 8 章建立的小系统进行优化的一些手段，包括建立强键值的集合类、获得 For Each、纠正圆整错误、用 GDI+进行交互绘图、界面美化和数据存盘、加载等。

第 12 章结合 CAD 编程技巧讨论了几种设计模式的应用，主要讨论了状态模式、访问者模式、模板方法模式和策略模式。另外，还简略地介绍了工厂方法模式、观察者模式、命令模式和记事模式。

本书的特色

1．采用了面向对象的编程技术

本书主要的实例都采用了面向对象的编程技术，其中第 4 章至第 8 章建立的 CAD 小系统是一个比较完整的程序。它不仅仅演示了创建 CAD 系统的思路和方法，还是一个很好的

面向对象编程实例。

2．实例丰富

本书实例比较丰富，其特点就是用实例“说话”。全书示例程序分 VB.NET 和 VC#.NET 两种语言，大大小小各有 50 多个，而且所有的示例程序都在 CAD 的语境中进行绘制，基本上没有与 CAD 无关的实例。这样做的好处是可以为读者创造一个学习的氛围，不分散注意力。

3．循序渐进的讲解方式

在章节安排上注意了循序渐进原则。首先介绍一些基础知识，然后是 CAD 系统的创建，最后介绍更多的技巧实现方法和优化方法，并结合 CAD 编程讨论设计模式的应用。如果您对 VB.NET 或 VC#.NET 已经有了比较多的了解，而且熟悉 GDI+ 和互操作，可以直接从第 4 章开始阅读。不过建议您还是从头开始过一遍。

本书的读者

所有关心图形学、CAD 技术的大学生、研究生和其他相关人员都可以是本书的读者。要求读者最好有一定的.NET 编程基础和面向对象编程经验。对于初学者，前面 3 章提供了快速入门的捷径。

为方便您学习本书的内容，我们将所有示例程序都放在随书的光盘上。所有程序都通过调试。

写作过程中，始终得到黄国明的支持，深表感谢！苏华惠做了部分录入工作，刘玉珊帮助搜集资料，一并表示感谢！

由于作者水平有限，书中难免存在谬误和缺点，谨请批评指正。请通过电子邮件与我们联系：

苏金明：s_jm@263.net.cn

周建斌：zjb@cdut.edu.cn

作　者

2003.10.12

目　　录

第 1 章　.NET 基础 ……(1)
1.1　.NET 开发环境 ……(1)
1.2　基本语法 ……(3)
1.2.1　数据类型与转换 ……(3)
1.2.2　变量 ……(5)
1.2.3　数组 ……(5)
1.2.4　过程 ……(6)
1.3　名字空间 ……(9)
第 2 章　面向对象编程 ……(11)
2.1　类 ……(11)
2.1.1　属性 ……(11)
2.1.2　方法 ……(13)
2.1.3　构造函数 ……(16)
2.1.4　重载 ……(17)
2.1.5　Me 和 this ……(18)
2.1.6　应用 Position 类 ……(19)
2.2　继承 ……(20)
2.2.1　基类 ……(20)
2.2.2　派生类 ……(22)
2.2.3　抽象基类 ……(24)
2.2.4　重写 ……(25)
2.2.5　遮蔽 ……(25)
2.2.6　重载 ……(27)
2.2.7　MyBase 和 base ……(27)
2.3　接口 ……(28)
2.3.1　创建 IGElement 接口 ……(28)
2.3.2　实现 IGElement 接口 ……(28)
2.3.3　测试 IGElement 接口 ……(30)
2.4　多态 ……(30)
2.4.1　用继承实现多态 ……(31)
2.4.2　用接口实现多态 ……(32)
2.4.3　两种方式的比较 ……(34)
第 3 章　GDI+ 编程 ……(35)

3.1 Graphics 对象 …… (35)
3.1.1 创建和使用 Graphics 对象 …… (35)
3.1.2 Paint 事件和 OnPaint 方法 …… (40)
3.2 线条绘制 …… (40)
3.2.1 颜色 …… (40)
3.2.2 画笔 …… (41)
3.2.3 线条绘制示例 …… (42)
3.3 文本 …… (46)
3.3.1 FontFamily 类 …… (46)
3.3.2 Font 类 …… (47)
3.3.3 StringFormat 类 …… (47)
3.3.4 刷子 …… (48)
3.3.5 DrawString 方法 …… (49)
3.3.6 文本绘制示例 …… (49)
3.4 路径 …… (50)
3.4.1 GraphicsPath 类 …… (51)
3.4.2 绘制和填充路径 …… (52)
3.4.3 路径定义示例 …… (52)
3.5 区域 …… (53)
3.5.1 Region 类 …… (53)
3.5.2 渐变色填充 …… (55)
3.6 坐标与变换 …… (59)
3.6.1 坐标系统 …… (59)
3.6.2 几何变换 …… (59)
3.6.3 全局坐标与局部坐标 …… (64)
3.7 Alpha 混合 …… (70)
3.8 反走样 …… (71)
3.9 用 API 函数绘图 …… (73)
3.9.1 为什么还要使用 API 函数 …… (73)
3.9.2 API 函数的声明和调用 …… (74)
3.9.3 用 API 函数绘图示例 …… (74)
第 4 章 设计 CAD 小系统的基本思路和技术 …… (78)
4.1 相关类的设计 …… (78)
4.1.1 对象和类 …… (78)
4.1.2 基本图元类设计 …… (78)
4.1.3 交互绘图类设计 …… (79)
4.1.4 类的交互 …… (80)
4.2 坐标系统 …… (80)
4.3 交互技术及其实现 …… (82)

4.3.1 用鼠标绘图 …… (83)
4.3.2 橡皮线 …… (85)
4.4 集合类 …… (89)
4.5 其他技术 …… (92)
4.5.1 数据存盘 …… (92)
4.5.2 界面优化 …… (92)
第 5 章 基本图元类设计 …… (93)
5.1 Win32API 类 …… (93)
5.2 CGElement 类 …… (97)
5.3 CLine 类 …… (101)
5.4 CRectangle 类 …… (106)
5.5 CCircle 类 …… (112)
5.6 CArc 类 …… (117)
5.7 CText 类 …… (125)
第 6 章 交互绘图类设计 …… (133)
6.1 ICommand 接口 …… (133)
6.2 CCreateLine 类 …… (133)
6.3 CCreateRectangle 类 …… (138)
6.4 CCreateCircle 类 …… (142)
6.5 CCreateArc 类 …… (147)
6.6 CCreateText 类 …… (153)
6.7 实现交互绘图 …… (155)
6.7.1 创建程序界面 …… (155)
6.7.2 创建测试代码 …… (156)
第 7 章 图元的编辑 …… (161)
7.1 拾取图元 …… (161)
7.1.1 包围矩形的计算 …… (161)
7.1.2 拾取图元 …… (172)
7.2 选择图元 …… (181)
7.2.1 添加菜单资源 …… (182)
7.2.2 鼠标单选 …… (182)
7.2.3 全选 …… (184)
7.2.4 放弃选择 …… (186)
7.3 删除图元 …… (187)
第 8 章 图元变换 …… (188)
8.1 平移变换 …… (188)
8.1.1 更新图元类 …… (188)
8.1.2 创建 CMove 类 …… (192)
8.1.3 实现平移图元 …… (196)

8.2 旋转变换 (197)
8.2.1 更新图元类 (198)
8.2.2 创建 CRotate 类 (201)
8.2.3 实现旋转图元 (205)
8.3 镜像图元 (206)
8.3.1 更新图元类 (207)
8.3.2 创建 CMirror 类 (211)
8.3.3 实现镜像图元 (215)
8.4 比例缩放图元 (216)
8.4.1 在 CGElement 类中添加 Scale 方法 (216)
8.4.2 在派生类中重写 Scale 方法 (216)
8.4.3 实现比例变换 (219)
第 9 章 GDI+ 提供的交互技巧 (221)
9.1 获取线形图元的包围矩形 (221)
9.2 拾取线形图元 (223)
9.3 区域的拾取 (230)
9.4 图元的复制 (233)
9.5 曲线的拾取 (234)
9.6 图元变换 (239)
第 10 章 相交图元 (245)
10.1 相交线 (245)
10.2 矩形框拾取 (255)
10.3 曲线求交 (268)
第 11 章 优化处理 (274)
11.1 强键值的集合类 (274)
11.1.1 .NET 提供的集合类的缺点 (274)
11.1.2 创建强键值的集合类 (275)
11.2 获得 For Each (283)
11.2.1 以后期绑定方式使用 For Each (283)
11.2.2 以前期绑定方式使用 For Each (285)
11.3 圆整错误 (290)
11.4 使用 GDI+ 交互绘图 (293)
11.5 界面美化 (299)
11.5.1 添加工具栏和状态栏 (299)
11.5.2 启动窗口 (306)
11.5.3 About 窗口 (308)
11.6 数据存储 (309)
11.6.1 序列化与反序列化 (309)
11.6.2 CAD 图形数据的序列化和反序列化 (314)

第 12 章　设计模式讨论 ……（319）
12.1　状态模式 ……（319）
12.2　访问者模式 ……（320）
12.3　模板方法模式 ……（328）
12.4　策略模式 ……（340）
12.5　其他设计模式 ……（348）
12.5.1　工厂方法模式 ……（349）
12.5.2　命令模式 ……（349）
12.5.3　观察者模式 ……（349）
12.5.4　记事模式 ……（349）
参考文献 ……（350）

第 1 章　.NET 基 础

本书不是 VB.NET 和 VC#.NET 方面的入门教程，不会对.NET 环境和语言的特点进行系统的介绍。但是，有一些我们后面要用到的与 VB 6.0 等有显著差别的特点还是需要说明一下。这部分内容包括.NET 的开发环境、基本语法特点和名字空间等。

1.1　.NET 开发环境

VB.NET、VC#.NET 和 VC++.NET 等共用一个.NET 平台，所以安装新的 Visual Studio 版本时它们全部被安装。新建项目时显示如图 1-1 所示的“新建项目”对话框。在“项目类型”列表框中选择语言类型，在“模板”文件列表框中选择具体的应用类型。

图 1-1　“新建项目”对话框

VB.NET 和 VC#.NET 的界面设计窗口如图 1-2 和图 1-3 所示，二者大同小异。VB.NET 和 VC#.NET 中的界面设计方法大致相同，不分别介绍。下面主要介绍 VB.NET 与 VB 6.0 在界面设计方面的差别。与 6.0 版本相比，VB.NET 中主要有以下几方面的不同。

1．窗体的变化

虽然在本书的 VB 篇中曾经介绍过，窗体也是类，但是在 VB 6.0 中，窗体和类基本上是分开来讲的。在 VB.NET 中，窗体是类这个问题就公开了，如果新建的窗体名称为 Form1，则会自动创建一个 Form1 类，它就相当于 6.0 中的窗体模块。另外，添加了一些窗体属性。在 VB 6.0 中，将窗体置于顶部需要调用 API 函数，现在不必了，将它的 Topmost 属性设置为 True 就行了。设置 Visible 属性，可以控制窗体的可见性；利用 Opacity 属性，可以将窗体设置为透明；利用 StartPosition 属性，可以设置窗体启动时在屏幕上的显示位置。现在有了模式对话框的概念，如果用 ShowDialog 方法调用窗体时将该窗体设置为有模式的对话框，则必须关闭该窗体以后才能对其他窗体进行操作。

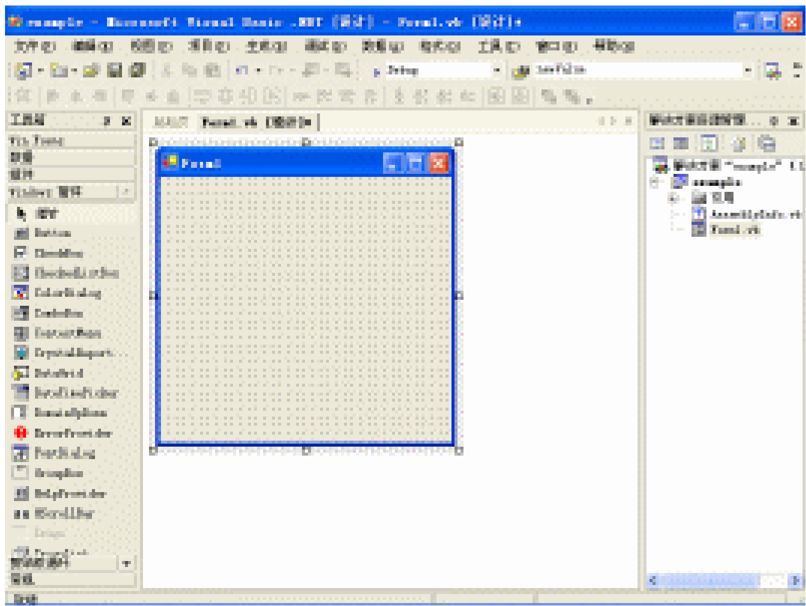

图 1-2　VB.NET 的界面设计窗口

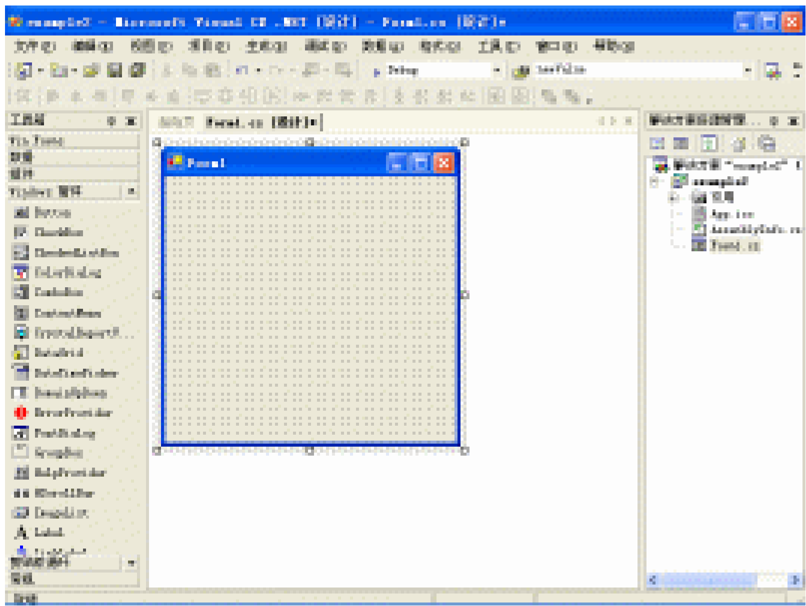

图 1-3　VC#.NET 的界面设计窗口

2．菜单制作方法的变化

VB 6.0 中使用对话框进行菜单制作，现在分别使用 MainMenu 控件和 ContextMenu 控件制作主菜单和弹出式菜单。菜单项的内容直接在窗体上输入，菜单项的属性，如名称、快捷键、单选、核选等都在属性窗口中设定。这样，菜单的制作就与其他控件的制作统一起来了。

3．控件设计的变化

部分控件的功能有了一些变化。比如我们常用的图片框控件，现在也能像图像框那样 Stretch 了。以前为了得到图片框的外框和图像框的 Stretch，我是将它们组合起来用的，现在不必了，设置 SizeMode 属性就行了。在 VB.NET 中，Timer, ImageList 这样一些不在窗体上显示的控件全部被放到窗体下面的一个空白面板中去了。这样，窗体上要干净一些，对这类控件的管理也方便一些。另外还有一些其他的变化，如后面要讲到的工具栏和状态栏等。详细内容需要参考其他专门介绍语言的书。

管理工具方面也有一些变化。如 VB 6.0 中管理窗体、类和公共模块等的窗口是工程窗口，现在用的是解决方案管理器。以前的工程现在称为项目。解决方案是一个比项目更大的概念，它可以包含多个项目。实际上，解决方案和项目是两个容器，它们对创建的类、引用、数据连接和文件等进行管理。

VB 6.0 中有一个对象浏览器，对 VB 中的所有对象进行显示。.NET 中有一个类视图窗口，如图 1-4 所示。它只显示项目中建立的类、接口和模块的信息。

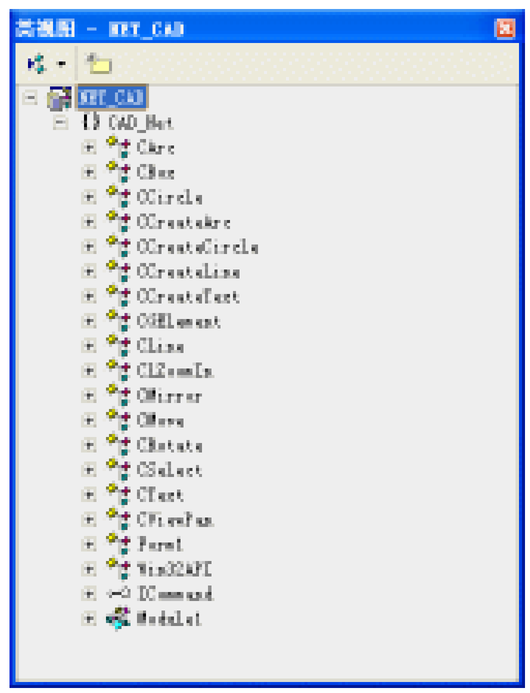

图 1-4　类视图窗口

1.2　基本语法

1.2.1　数据类型与转换

.NET 中，通用数据类型为 Object 型，VB 6.0 中的 Vairant 类型已经取消了。对象类型赋值时不再需要 Set 关键字。例如：

【VB.NET】

```
Dim obj1 As New Object()
Dim obj2 As Object = obj1
```

【VC#.NET】

```
object obj1=new object();
object obj2 = obj1;
```

数据类型的转换有隐式转换和显式转换两种。隐式转换不需要任何转换形式，直接采用 A=B 的形式，系统会自动进行转换。显式转换则通过转换函数或其他的转换形式对数据类型进行强制转换。对于数值类型的转换而言，低精度类型向高精度类型转换影响不大，但高精度类型向低精度类型转换要损失部分信息。在 VB.NET 中，两种类型的转换都可以通过隐式转换完成。但 VC#.NET 是强类型的语言，要求后一种转换必须是显式转换。在 VB.NET 中，可以通过转换函数进行显式转换。如将数值转换为单精度型时使用 CSng 函数，转换为双精度型时使用 CDbl 函数，转换为字符型时使用 CStr 函数等。在 VC#.NET 中则需要进行强类型转换，转换时需要在要转换变量的名称前添加带小括号的类型名。下面的代码演示了数据类型的隐式转换和显式转换。

【VB.NET】

```
Dim i As Integer = 10
Dim j As Double = 20.756
Dim k1 As Single = i
Dim k2 As Single = CSng(j)
Dim k3 As String = j.ToString      或   Dim k3 As String =CStr(j)
Console.WriteLine(("k1={0},k2={1},k3={2}"), k1, k2, k3)
```

【VC#.NET】

```
int i = 10;
double j = 20.756;
float k1 = i;
float k2 = (float)(j);
string k3 = j.ToString();
Console.WriteLine(("k1={0},k2={1},k3={2}"), k1, k2, k3);
```

进行显式转换时还有一种情况，就是自定义转换类型。比如下面的代码中，p2 是 Object(object)类型的变量，它本身并没有 X 属性和 Y 属性。现在将 p2 指定为 p1 的引用，因为 p1 是 Point 类型的数据，有 X 属性和 Y 属性，所以现在 p2 也有这两个属性。但是调用时必须对它进行强制类型转换，并且转换为 Point 型。VB.NET 中使用 CType 函数进行自定义类型转换，VC#.NET 中则进行强类型转换。

【VB.NET】

```
Dim p1 As New Point(10, 10)
Dim p2 As Object = p1
Console.WriteLine("x={0},y={1}", CType(p2, Point).X, CType(p2, Point).Y)
```

【VC#.NET】

```
Point p1=new Point(10, 10);
object p2= new object();
p2 = p1;
Console.WriteLine("x={0},y={1}", ((Point)(p2)).X,  ((Point)(p2)).Y);
```

1.2.2 变量

这里主要讲两个问题，一个是变量的声明，另一个是变量的作用范围。VB.NET 与 VB 6.0 不同，可以采用下面的形式声明变量：

```
Dim a, b, c As Single, d As Double
```

即同一类型的变量可以用逗号间隔，连续输入变量名。不同类型的变量也可以在一行中进行声明。上面的变量在 VC#.NET 中做如下声明：

```
float a, b, c;
double d;
```

在 VB.NET 和 VC#.NET 中，变量的作用范围可以局限于块，如 For 循环块或者 If 块等。在块外使用该变量时，需要重新进行声明。如下面的代码段中，在 For 循环中声明了一个单精度型的 xx 变量，利用它进行求和计算。该变量只在 For 循环内部有效，在外部使用它时会报错，认为它是一个没有声明的变量。

【VB.NET】

```
Dim sum As Single, i As Integer
For i = 0 To 9
    Dim xx As Single=10+i
    sum += xx
Next
sum += xx
```

【VC#.NET】

```
float sum;
for (int i = 0;i<=9;i++)
{
    float xx=10+i;
    sum += xx;
}
sum+=xx;
```

1.2.3 数组

.NET 中，数组的索引值是从 0 开始的，而且必须从 0 开始。因为基数固定，所以声明数组时可以直接像下面这样声明：

【VB.NET】

```
Dim Value(9) As Single
```

【VC#.NET】

```
float[] Value;
```

使用 New(new)关键字创建数组，如下所示：

【VB.NET】

```
Dim Value() As Single = New Single()
```

【VC#.NET】

```
float[] Value=new float[3]
```

在创建数组的同时，可以对数组进行初始化。如下所示，将数组元素的初始化值用大括号括起来，然后直接放在创建数组的代码后面就行了。对数组进行初始化是一个好的编程习惯。

【VB.NET】

```
Dim Value() As Single = New Single() {2, 4, 7}
```

【VC#.NET】

```
float[] Value=new float[10]{2,4,7};
```

数组还有一些高级特性，后面讲述过程的参数传递时将会讲到，过程函数可以传回一个数组。

1.2.4　过程

.NET 中的过程与某些语言中的过程的一个很明显的不同就在于，缺省时它的参数是按值传递的，而其他语言如 VB 6.0 是按地址进行传递的。VB.NET 和 VC#.NET 中的过程一般具有下面的形式。注意，现在函数使用 Return(return)语句返回值。

【VB.NET】

```
Private Sub Proc(ByVal x As Single, ByVal y As Single)
    '功能代码　比如　y=x*x
End Sub
Private Function Func(ByVal x As Single, ByVal y As Single) As Single
    Dim aVal As Single
    '功能代码
    Return aVal
End Function
```

【VC#.NET】

```
private void Proc(float x , float y)
{
    //功能代码　比如　y=x*x;
}
```

```
private float Func(float x, float y)
{
      float aVal;
      //
      //功能代码
      //
      return aVal;
}
```

按值传递参数和按地址传递参数的一个主要区别是，按值传递参数不改变参数在调用过程中的值，而按地址传递参数时要改变。比如，现在要调用上面的 Proc 过程，要利用它返回一个 y 值，y 值是 x 的平方。在调用函数中首先声明一个 a 变量，并赋初值 0。然后给定 x 参数的值，调用 Proc 过程，要求返回 y 参数的值并输出到输出窗口中。即：

【VB.NET】

```
Private Sub CallProc()
    Dim a As Single = 0
    Proc(10, a)
    Console.WriteLine(a)
End Sub
```

【VC#.NET】

```
private float CallProc()
{
    float a = 0;
    Proc(10, a);
    Console.WriteLine(a)；
}
```

按照前面的意图，我们希望得到的值是 10 的平方，即 100，但实际上得到的是 0，是 a 的初值。因为 a 参数是按值传递的，在 CallProc 过程中，它的值不会因为调用 Proc 过程而改变。下面把传值方式改为传址方式就可以解决问题。

VB.NET 中声明使用传值方式时，使用 ByVal 关键字；使用传址方式时，使用 ByRef 关键字。VC#.NET 中使用传值方式时不使用关键字，使用传址方式时使用的是 ref 关键字。需要说明的是，在 VC#.NET 中使用传址方式时，还要在调用过程中声明 ref 关键字。

在 VC#.NET 中，传回数组参数使用的是 out 关键字，也需要在调用过程中声明它。

另外，使用数组的高级特性还可以返回一个数组。可以参见下面的例子。

下面的例子演示了上面所介绍的几种参数传递方法，可以对照上面的文字进行体会。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) Handles MyBase.Load
    Dim c As Single
    Sum(10, 20, c)
```

```
        Console.WriteLine(c)
        Dim cpt(1) As Single
        Compute(10, 20, cpt)
        Console.WriteLine(cpt(0) & "," & cpt(1))
        Dim cpt2(1) As Single
        cpt2 = Compute(50, 20)
        Console.WriteLine(cpt2(0) & "," & cpt2(1))
    End Sub

    Private Sub Sum(ByVal a As Single, ByVal b As Single, ByRef c As Single)
        c = a + b
    End Sub

    Private Sub Compute(ByVal a As Single, ByVal b As Single, ByRef cpt As Single())
        cpt(0) = a + b
        cpt(1) = a - b
    End Sub

    Private Function Compute(ByVal a As Single, ByVal b As Single) As Single()
        Dim cpt(1) As Single
        cpt(0) = a + b
        cpt(1) = a - b
        Return cpt
    End Function
```

【VC#.NET】

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
        float c=0;
        Sum(10, 20, ref c);
        Console.WriteLine(c);
        float[] cpt={0,0};
        Compute(10, 20, out cpt);
        Console.WriteLine(cpt[0] + "," + cpt[1]);
        float[] cpt2;
        cpt2 = Compute(50, 20);
        Console.WriteLine(cpt2[0] + "," + cpt2[1]);
    }

    private void Sum(float a , float b ,ref float c)
    {
```

```
        c = a + b;
    }

    private void Compute(float a,float b, out float[] cpt)
    {
        cpt=new float[2];
        cpt[0] = a + b;
        cpt[1]= a - b;
    }

    private float[] Compute(float a, float b)
    {
        float cpt0,cpt1;
        cpt0 = a + b;
        cpt1 = a - b;
        float[] cpt={cpt0,cpt1};
        return cpt;
    }
```

程序运行结果如图 1-5 所示。

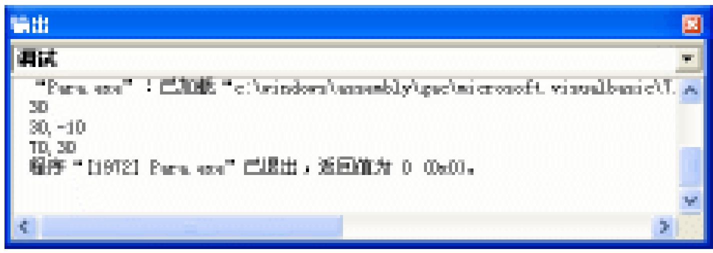

图 1-5　参数传递演示

1.3　名字空间

名字空间（或者名称空间、命名空间等）实际上就是.NET 框架提供的一些类库。这些类库具有严密的组织，有各自的功能。利用它们，可以访问系统功能、构建应用程序、创建组件和控件等。名字空间构成了利用.NET 进行编程的基础。

名字空间具有清晰的层次结构。处于最顶部的是 System 名字空间，它包含一些基本的类和基类，构建基本框架。System 名字空间下面有很多次一级的名字空间，如 Drawing，Math，Collections 等。利用它们提供的类，可以实现绘图、数学计算、集合等比较具体的功能。常用的名字空间及其说明如表 1-1 所示。

表 1-1　常用的名字空间及其说明

名 字 空 间	说　　明
System	包含基本类和基类，它们定义常用的值和引用数据类型、事件和事件处理程序、接口、属性和异常处理
System.Drawing	提供了对 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。此命名空间包含梯度型画刷、Matrix 类（用于定义几何变换）和 GraphicsPath 类
System.Math	为三角函数、对数函数和其他通用数学函数提供常数和静态方法
System.Collections	包含接口和类，这些接口和类定义各种对象的集合
System.Runtime.InterOperations	提供可用于访问 .NET 中的 COM 对象和本机 API 的类的集合。此命名空间中的类型分为下列功能领域：属性、异常、COM 类型的托管定义、包装、类型转换器和 Marshal 类

使用名字空间中的类之前需要首先导入名字空间。在 VB.NET 和 VC#.NET 中分别使用 Imports 和 using 进行导入。假如要在一个名为 CLine 的类中创建一个方法绘制直线段，则需要用到 GDI+中 Graphics 对象的 DrawLine 方法。Graphics 类属于 System.Drawing 名字空间，所以需要在定义类以前导入该名字空间，如：

【VB.NET】

```
Imports System.Drawing

Public Class CLine
End Class
```

【VC#.NET】

```
using System;
using System.Drawing;

namespace GraphDraw
{
    public class CLine
    {
    }
}
```

第 2 章　面向对象编程

虽然面向对象编程已经提出很久了，但是在观念、技术不断更新的程序设计领域，它仍然是主流的编程技术，而且一些更深入、更系统的方法正在提出，讨论仍然热烈。本章主要介绍基本的面向对象编程方法，第 12 章将结合 CAD 编程做一些设计模式方面的讨论。

需要说明的是，VB.NET 的诞生对于 VB 程序员来说，具有非同寻常的意义。所谓“语言一小步，VB 一大步”，主要就体现在对面向对象编程技术的支持上。现在 VB 已经是真正的面向对象编程语言了，构造函数、接口、继承、多态等这样一些必备元素一样也不少，而且它现在和 VC++在一个.NET 平台上工作。

2.1　类

类是描述对象特征的代码。作为对象来说，一方面它有很多自身的特点，有一些固有属性，另一方面，它要与其他对象进行交互，发生联系，会产生一系列的行为。而且，当某种行为发生在它上面时，可能会发生一些事件，产生一定的后果。所以，用代码来描述对象时，就需要考虑这些问题，并进行抽象、概括和组织。本节主要介绍类的相关概念和实现方法。

2.1.1　属性

属性描述对象的特征，它是一种特殊的方法，用于设置和读取对象的值。下面创建一个 CPosition 类，它有一个 x 属性和一个 y 属性，分别定义点的横坐标和纵坐标，定义代码如下：

【VB.NET】

```
Public Class Position
   Private m_x, m_y As Single
   Private m_ID As Integer

   Public Property x() As Single
       Get
           Return m_x
       End Get
       Set(ByVal Value As Single)
           m_x = Value
       End Set
   End Property

Public Property y() As Single
       Get
```

```
            Return m_y
        End Get
        Set(ByVal Value As Single)
            m_y = Value
        End Set
    End Property
```

【VC#.NET】

```
public class Position
{
    private float m_x, m_y;
    private int m_ID;

    public float x
    {
        get{return m_x;}
        set{m_x = value;}
    }

    public float y
    {
        get{return m_y;}
        set{m_y = value;}
    }
```

从上面的代码中可以看出，在 VB.NET 代码中，读写属性的语句是集中在一个过程中的，这与 VB 6.0 中不同。在 VB 6.0 中，读写属性使用各自的属性过程。相比而言，现在的形式更紧凑，更便于阅读。

1．只读属性

有的属性值可以由其他基本属性值获得，是只读的，称为只读属性。在 VB.NET 和 VC#.NET 中，在属性过程中添加 ReadOnly 关键字，并且只实现 Get 块，可以实现只读属性。例如，假设坐标原点为(0,0)，则在同一坐标系中，当前点到原点的距离可以求得，把它设置为 Disto 属性，编码为：

【VB.NET】

```
Public ReadOnly Property Disto() As Double
    Get
        Return Sqrt(m_x * m_x + m_y * m_y)
    End Get
End Property
```

【VC#.NET】

```
public float Disto
{
    get{return (float)(Math.Sqrt(m_x * m_x + m_y * m_y));}
}
```

2．只写属性

与只读属性相对应，有的属性是只写的。在属性过程中添加 WriteOnly 关键字，并且只实现 Set 块，可以实现只写属性。

3．缺省属性

如果是用 VB.NET 编程，用 Default 关键字来标示属性时该属性定义为缺省属性。现在假设创建了一个 CPoints 类，它能够保存 CPosition 对象并能实现索引和删除。为了实现索引，定义了一个 Item 属性，它是一个缺省的只读属性，其中 aID 为索引值。该方法返回一个 CPosition 对象。

【VB.NET】

```
Default Public ReadOnly Property Item(ByVal aID As Integer) As Position
    Get
        ' 返回一个 CPosition 类型的对象
    End Get
End Property
```

如果用 VC#.NET 编程，要定义缺省属性则需要使用索引器，如下所示：

【VC#.NET】

```
public Position this{int aID}{
    get{
        //返回一个 CPosition 类型的对象
    }
}
```

将属性定义为缺省属性以后，可以在类实例名称后面直接跟括号和索引值。例如，现在 clines 是一个 CLines 类实例，其中存储了 10 个 CLine 对象，则可以通过 clines(4)的方式直接调用其中的第 5 个对象。

2.1.2　方法

方法是一些函数或过程。根据需要，可以创建公共方法、受保护的方法和私有方法等，它们具有不同的使用范围。

1．公共方法

公共方法用 Public 或 public 关键字进行标示，在整个项目范围内都可以使用。下面给 Position 类添加一个 GetAngle 方法，它返回当前点与坐标原点间的方位角。

【VB.NET】

```
Public Function GetAngle() As Single
    '坐标原点为 Org，全局变量
    Dim org As New Position(0, 0)
    Dim sita,tanSita,subx As Single
    subx = Abs(m_x – org.x)
    If org.x = m_x Then subx = 0.0001
    tanSita = (Abs(m_x – org.x)) / subx
    sita = Atan(tanSita)
    If m_x >= org.x And m_y <= org.y Then
            Return sita
    ElseIf m_x <= org.x And m_y >= org.y Then
            Return (3.1416 – sita)
    ElseIf m_x <= org.x And m_y >= org.y Then
            Return (3.1416 + sita)
    ElseIf m_x >= org.x And m_y >= org.y Then
            Return (2 * 3.1416 – sita)
    End If
End Function
```

【VC#.NET】

```
public float GetAngle()
{
    //坐标原点为 Org，全局变量
    Position org=new Position(0, 0);
    float sita,tanSita,subx;
    float angle=0;
    subx = Math.Abs(m_x – org.x);
    if (org.x == m_x) {subx = 0.0001f;}
    tanSita = (Math.Abs(m_x – org.x)) / subx;
    sita = checked((float)(Math.Atan(tanSita)));
    if (m_x >= org.x && m_y <= org.y)
    {
        angle=sita;
    }
    else if (m_x <= org.x && m_y >= org.y)
    {
        angle=3.1416f – sita;
    }
    else if (m_x <= org.x && m_y >= org.y)
    {
```

```
            angle=3.1416f + sita;
        }
        else if (m_x >= org.x && m_y >= org.y)
        {
            angle=2 * 3.1416f – sita;
        }

        return angle;
    }
```

2．受保护的方法

受保护的方法用 Protected 或 protected 关键字进行标示，可以用于本类及其派生类。受保护的方法就像那些只在特定系统或部门内适用的法规、规则一样。

3．私有方法

在函数或过程前添加 Private 关键字，将方法设置为私有。私有方法只能被本类中的函数或过程所调用，是为其他函数服务的。比如，在交互绘制圆弧的过程中，拾取圆弧时需要知道拾取点是否落在圆弧所在的范围内，可通过比较拾取点、圆弧起点和终点这三点的方位角的关系来实现。此时，就要用到前面的 GetAngle 函数，它在这里是私有的。因为在外部调用圆弧类 CArc 的方法时不会用到它，它是为 Pick 方法服务的。可用下面的代码将方法设置为私有。

【VB.NET】

```
Public Class CArc
  '......
  Public Function Pick(Byval aPos As PointF,_
                       Byval PickRadius As Single)_
                       As Boolean
    '......
    angBegin=GetAngle(m_Begin)
    angEnd=GetAngle(m_End)
    angCur=GetAngle(aPos)
    '......
  End Function

  Private Function GetAngle(Byval aPos As PointF) As Single
    '......
  End Function

  '......
End Class
```

【VC#.NET】

```
public class CArc
{
    //......
    public bool Pick(PointF aPos,float PickRadius)
    {
        //......
        angBegin=GetAngle(m_Begin);
        angEnd=GetAngle(m_End);
        angCur=GetAngle(aPos);
        //......
    }

    private float GetAngle(PointF aPos)
    {
      //......
     }
      //......
 }
```

2.1.3 构造函数

构造函数是 VB.NET 新增的功能，VB 6.0 里面没有。它的主要作用是提供对象数据的初始化。可以提供一个构造函数，也可以提供多个构造函数。

1．创建无参构造函数

无参构造函数没有参数，每个类都有一个无参构造函数。下面给前面创建的 CPositon 类添加无参构造函数。用 VC#.NET 创建类时，会自动给出一个无参构造函数。

【VB.NET】

```
Public Sub New()
End Sub
```

【VC#.NET】

```
public position() {
}
```

2．带参构造函数

构造函数可以带一个或多个参数。一般而言，类中应该提供尽可能全面的构造函数。下面给 Position 类提供两个构造函数，一个用给定的横、纵坐标定义，另一个用已经存在的点来定义。

【VB.NET】

```
Public Sub New(ByVal xx As Single, ByVal yy As Single)
```

```
        m_x = xx
        m_y = yy
    End Sub

    Public Sub New(ByVal aPos As Position)
        With aPos
            m_x = .x
            m_y = .y
        End With
    End Sub
```

【VC#.NET】

```
    public Position(    float xx,float yy )
    {
        m_x = xx;
        m_y = yy;
    }

    public Position(Position aPos)
    {
        m_x =aPos.x;
        m_y =aPos.y;
    }
```

2.1.4　重载

重载允许我们使用参数不同但名称相同的方法。假如前面定义了多个构造函数，过程名称都相同，但参数个数或类型不同，这实际上也是一种重载。在 VB 6.0 里面不能实现重载，同一个类中不允许有名称相同的方法。

使用重载时可以有几种情况，一种是参数个数不相同，另一种是参数个数相同，但参数类型不全相同。为了演示重载，下面给 Position 类添加一组重载方法 Move。第 1 个 Move 方法使当前点水平移动，第 2 个 Move 方法使当前点可以向平面上任何方向移动。

【VB.NET】

```
    Public Sub Move(ByVal deltaX As Single)
        m_x += deltaX
    End Sub

    Public Sub Move(ByVal deltaX As Single, ByVal deltaY As Single)
        m_x += deltaX
        m_y += deltaY
    End Sub
```

【VC#.NET】

```
public void Move(float deltaX )
{
    m_x += deltaX;
}

public void Move(float deltaX, float deltaY)
{
    m_x += deltaX;
    m_y += deltaY;
}
```

定义重载方法以后，调用时会显示如图 2-1 所示的黄色提示信息框。信息框中，前面是一个上下框，其中给出了重载方法的总个数和当前方法的序号；后面显示的是当前方法的调用格式。单击上下框两侧的箭头按钮，可以进行滚动，显示上一个或下一个方法的调用格式。

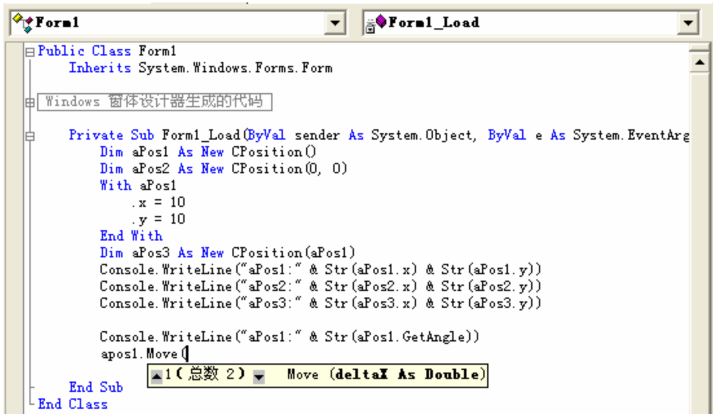

图 2-1　重载方法提示

2.1.5　Me 和 this

在 VB.NET 和 VC#.NET 中，分别用 Me 和 this 调用自身成员。下面在两种语言的 Position 类中分别添加 MeTest 方法和 thisTest 方法，在输出窗口输出当前点的坐标。

【VB.NET】

```
Public Sub MeTest()
    Console.WriteLine("坐标为：{0},{1}", Me.x, Me.y)
End Sub
```

【VC#.NET】

```
public void thisTest()
{
    Console.WriteLine("坐标为：{0},{1}", this.x, this.y);
}
```

2.1.6　应用 Position 类

为了说明类的创建，前面定义了一个 Position 类，包括它的属性、方法和构造函数等。下面对它进行测试，说明如何调用它的成员。在项目的 Form1 类中添加 Form1_Load 事件代码，在载入窗体时创建 3 个 Position 对象，然后在输出窗口中输出它们的坐标值。后面还测试了 GetAngle 方法和重载方法 Move。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
    Dim aPos1 As New Position(10, 10)
    Dim aPos2 As New Position(0, 0)

    Dim aPos3 As New Position(aPos1)

    Console.WriteLine("坐标为：{0},{1}", aPos2.x, aPos2.y)
    Console.WriteLine("坐标为：{0},{1}", aPos3.x, aPos3.y)

    Console.WriteLine(aPos1.GetAngle())

    aPos2.Move(100)
    aPos3.Move(100, 100)
    Console.WriteLine("坐标为：{0},{1}", aPos2.x, aPos2.y)
    Console.WriteLine("坐标为：{0},{1}", aPos3.x, aPos3.y)

    aPos1.MeTest()
End Sub
```

【VC#.NET】

```
private void Form1_Load(object sender, System.EventArgs e)
{
    Position aPos1=new Position(10, 10);
    Position aPos2=new Position(0, 0);

    Position aPos3=new Position(aPos1);

    Console.WriteLine("坐标为：{0},{1}", aPos2.x, aPos2.y);
    Console.WriteLine("坐标为：{0},{1}", aPos3.x, aPos3.y);

    Console.WriteLine(aPos1.GetAngle());
```

```
            aPos2.Move(100);
            aPos3.Move(100, 100);
            Console.WriteLine("坐标为：{0},{1}", aPos2.x, aPos2.y);
            Console.WriteLine("坐标为：{0},{1}", aPos3.x, aPos3.y);

            aPos1.thisTest();
        }
```

运行结果如图 2-2 所示，输出结果显示在输出框中。

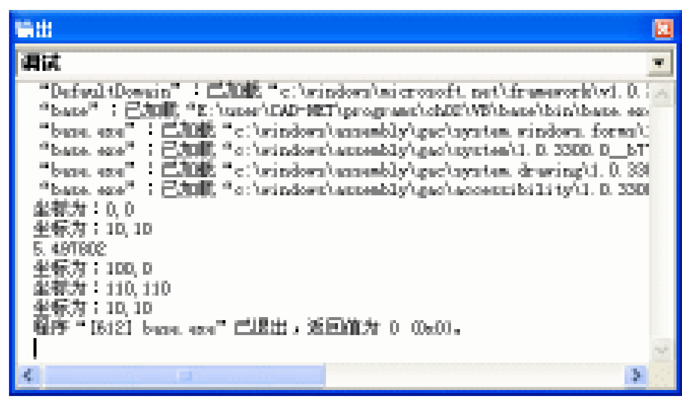

图 2-2　类的测试结果

2.2　继承

继承是 VB 6.0 编程人员盼望已久的面向对象编程特性。在这套书的 VB 篇中，笔者不得不为每种图元都定义了颜色、线宽和线型等属性。有了继承以后，只需要在基类中定义它们，并在派生类中进行继承就行了。

对继承关系的描述，最常用的说法是，是“is-a”的关系，即“是一种”的关系。比如老虎是一种动物，香蕉是一种水果等。继承是一种很有用的面向对象特性，与它有关的概念包括基类、派生类、抽象基类、重写、遮蔽等。

2.2.1　基类

继承是一个体系，是多个对象之间的一种关系。在这个体系里面，相邻的两个继承层次中处于上面一个层次的那个类或那些类称为基类，又称为父类，处于下面一个层次的那个类或那些类称为派生类或子类。基类在它所属的概念层次上描述对象的特征和行为。

下面结合一个例子来说明继承关系。点、直线段、圆和圆弧等都是图元的一种，它们都有编号、颜色、线宽和线型等属性，都有绘图等方法。可以考虑建立一个图元基类 CGElement，然后在它的基础上派生其他图元类。作为演示，下面建立一个简单的 CGElement 类。

【VB.NET】

```
Public Class CGElement
    Private m_ID As Integer

    Public Property ID() As Integer
        Get
            Return m_ID
        End Get
        Set(ByVal Value As Integer)
            m_ID = Value
        End Set
    End Property

    Public Sub New()
    End Sub

    Public Sub Draw()
        Console.WriteLine("绘图元。")
    End Sub
End Class
```

【VC#.NET】

```
public class CGElement
{
    private int m_ID;

    public int ID
    {
        get{return m_ID;}
        set{m_ID = value;}
    }

    public CGElement()
    {
    }

    public void Draw()
    {
        Console.WriteLine("绘图元。");
    }
}
```

2.2.2 派生类

古时候有一则笑话是这么说的：一个人照着别人描述的样子去找千里马，他老想着这千里马眼要突、蹄要怎么的，结果找回来一只大青蛙。对了，是按图索骥。笑话的本意是要我们做事不要生搬硬套，否则要闹笑话。那么这人的问题出在哪里呢？他显然忽略了他要找的首先是马，然后才是千里马这个问题。

切换到语言环境，这个问题就是基类与派生类的关系问题。显然，这里马是基类，千里马是派生类。派生类首先从基类那里继承成员，使自己具有基类的基因，然后才有自己的区别于其他派生类的特征。关于马和千里马，大致还有以下这样一些排列组合。

（1）千里马也是马。

（2）先有马，后有千里马。

（3）想做千里马，先做马。

后面要讲到的重写和遮蔽等都在派生类中实现。在 VB.NET 和 VC#.NET 中分别用 Inherits 关键字和冒号（:）来表示继承关系，如：

【VB.NET】

```
Public Class CLine
    Inherits CGElement
```

【VC#.NET】

```
public class CLine:CGElement
```

派生类中可以继承基类的公共成员和受保护成员。如最常用的 Form1 类就继承了 System.Windows.Forms 名字空间中的 Form 类。当我们在 Form1 内部使用 Me 这个关键字来调用自身的成员时，会显示一个成员列表，如图 2-3 所示。在没有新建成员的情况下，该成员列表中的成员都是从基类中继承来的。

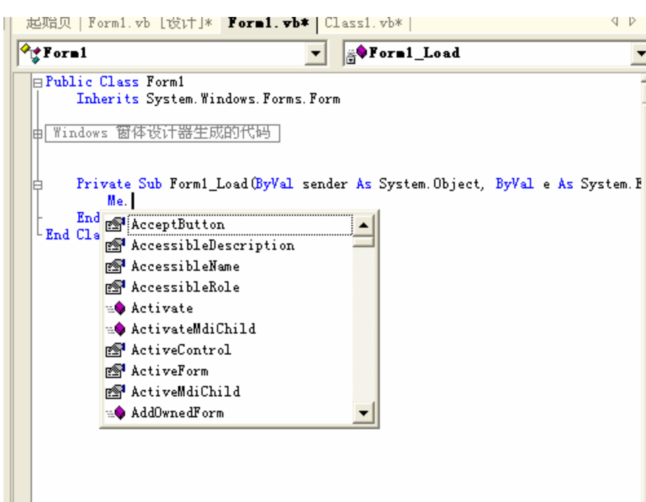

图 2-3 Form1 类的继承成员

下面创建一个 CLine 类，它描述直线段的特征和行为。直线段是一种图元，所以直线段

类 CLine 与图元类 CGElement 之间存在继承关系。定义这种继承关系以后，CLine 类就能继承 CGElement 类的属性和方法。下面从 CGElement 类派生 CLine 类，它继承基类的 ID 属性和 Draw 方法。

【VB.NET】

```
Public Class CLine
    Inherits CGElement

    Private m_Begin, m_End As PointF

    '定义直线段的起点和终点属性
    '......

    '构造函数
    Public Sub New()
    End Sub

End Class
```

【VC#.NET】

```
public class CLine:CGElement
{
    private PointF m_Begin,m_End;

    //定义直线段的起点和终点属性
    //......

    //构造函数
    public CLine()
    {
    }
}
```

在 Form1 类中输入下面的代码，对 CGElement 类和 CLine 类进行测试。代码中首先分别创建一个 CGElement 类实例 ge 和一个 CLine 类实例 line；然后将 line 的继承成员 ID 属性设置为 10，并在输出窗口中输出它；最后绘制直线段 line。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
    Dim line As New CLine()
    line.ID = 10
    Console.WriteLine("直线段的 ID 号为：{0}", line.ID)
```

```
        line.Draw()
    End Sub
```

【VC#.NET】

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
        CLine line=new CLine();
        line.ID = 10;
        Console.WriteLine("直线段的 ID 号为：{0}", line.ID);
        line.Draw();
    }
```

运行程序，输出窗口中的结果如图 2-4 所示。

图 2-4 输出窗口中的结果

2.2.3 抽象基类

前面创建的 CGElement 类作为 CLine 类的基类是可以实例化的，可以用 New 关键字或 new 关键字创建它的实例。但如果现在只想把 CGElement 类作为一个基类使用，因为作为"图元"来讲，它只是一个抽象的概念，并没有可见的形体，所以创建一个它的实例并没有多大的实际意义。

将基类创建为抽象基类，使用 VB.NET 时需要用到 MustInherit 关键字，使用 VC#.NET 时需要用到 abstract 关键字。必须重写的抽象方法需要用 MustOverride 关键字或 abstract 关键字进行标示。

下面将前面的 CGElement 类改成抽象基类。

【VB.NET】

```
    Public MustInherit Class CGElement
        '此处代码与前面的相同

        Public MustOverride Sub Draw()
    End Class
```

【VC#.NET】

```
    public abstract class CGElement
    {
        //此处代码与前面的相同
```

```
        public abstract void Draw();
    }
```

将 CGElement 类改成抽象基类以后，就不能用 New 关键字或 new 关键字进行实例化了。在派生类中必须重写抽象方法 Draw。下面介绍重写。

2.2.4 重写

重写，或者说覆盖，指的是在派生类中改写从基类中继承来的方法，用另外的方式来实现它，以实现个性。

如果基类的方法为抽象方法，或者是用 Overridable（VB.NET 中）或 virtual（VC#.NET）关键字声明了的可以重写的方法，则在派生类中可以重写该方法。比如在基类 CGElement 类中有一个可以重写的方法 Draw，则派生类 CLine 类和 CRectangle 类等中可以重写该方法，分别实现直线段和矩形的绘制。在派生类中重写方法需要使用 Overrides 关键字或 override 关键字。

下面接着 2.2.3 节的例子，在派生类 CLine 类中重写 Draw 方法。

【VB.NET】

```
Public Overrides Sub Draw()
    Console.WriteLine("绘制直线段。")
End Sub
```

【VC#.NET】

```
public override void Draw()
{
    Console.WriteLine("绘制直线段。");
}
```

2.2.5 遮蔽

使用遮蔽也可以重写基类的方法，但遮蔽不仅仅是重写基类的方法，它还可以替代基类的方法。例如，在 VB.NET 中，如果基类的方法有两个参数，使用遮蔽，可以在派生类中用带有 3 个参数的同名方法替代它。另外，使用遮蔽方法重写基类的方法时，不需要在基类中将方法声明为 Overridable 或 virtual。

下面的代码首先创建一个基类 CBase，它有一个 Add 方法，返回两个参数的和；然后创建一个派生类 CDerive，在其中遮蔽基类的 Add 方法，在两个参数的和的基础上再加上 100 以后返回；最后在 Form1 类中调用 CDerive 类实例的 Add 方法。

创建项目以后，首先创建一个 CBase 类。

【VB.NET】

```
Public Class CBase
    Public Function Add(ByVal a As Single, ByVal b As Single) As Single
        Return a + b
    End Function
End Class
```

【VC#.NET】

```
public class CBase
{
    public CBase()
    {
    }

    public float Add(float a,float b)
    {

        return a + b;
    }
}
```

然后创建 CDerive 类，代码如下：

【VB.NET】

```
Public Class CDerive
    Inherits CBase
    Public Shadows Function Add(ByVal a As Single, ByVal b As Single) As Single
        Return a + b + 100
    End Function
End Class
```

【VC#.NET】

```
public class CDerive:CBase
{
    public CDerive()
    {
    }

    public new float Add(float a,float b)
    {
        return a + b + 100;
    }
}
```

在 Form1 类中添加下面的代码，创建一个 CDerive 类实例 sum，然后调用它的 Add 方法，根据给定的参数 10 和 20 求和。此时基类返回的值将是 130，而不是 30。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
    Dim sum As New CDerive()
```

```
    sum.Add(10, 20)
End Sub
```

【VC#.NET】

```
private void Form1_Load(object sender, System.EventArgs e)
{
    CDerive sum=new CDerive();
    sum.Add(10, 20);
}
```

2.2.6　重载

前面在介绍类的时候曾经讲解了重载技术。这里主要介绍在有继承的情况下如何使用重载。如果要重载从基类中继承来的方法，在 VB.NET 中，必须使用 OverLoads 关键字。比如，在上面的例子中，在没有遮蔽的情况下，如果要重载基类的 Add 方法，求两个整型值的和，需要添加下面的代码：

```
Public Overloads Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
    Return a + b
End Function
```

在有遮蔽的情况下重载基类的 Add 方法，也必须使用 Shadows 关键字。比如，2.2.5 节的例子中，遮蔽基类的 Add 方法以后，如果还要重载一个带有 3 个参数的 Add 方法，需要添加下面的代码：

```
Public Shadows Function Add(ByVal a As Single, ByVal b As Single, _
                ByVal c As Single) As Single
    Return a + b + 100
End Function
```

2.2.7　MyBase 和 base

在 VB.NET 中使用 MyBase 或在 VC#.NET 中使用 base，可以在派生类中调用基类的成员。如 2.2.5 节遮蔽了基类的 Add 方法，在派生类中将不能应用该 Add 方法的定义。但是通过 MyBase 或 base 来调用时，仍然可以实现。比如，2.2.5 节中遮蔽后重写的 Add 方法可以重写为：

【VB.NET】

```
Public Shadows Function Add(ByVal a As Single, ByVal b As Single) As Single
    Return MyBase.Add(a, b) + 100
End Function
```

【VC#.NET】

```
public new float Add(float a,float b)
{
    return base.Add(a,b) + 100;
}
```

2.3　接口

VB 6.0 中提供了接口，在没有继承的情况下，它可以前期绑定的方式实现多态。在本套书的 VB 篇中大量使用了接口技术。在.NET 中，任何对象都有一个本地接口，使用 Implements 关键字(VB.NET 中)或冒号(VC#.NET 中)可以实现辅助接口。

2.3.1　创建 IGElement 接口

.NET 中接口的概念与 VB 6.0 中的基本相同，但实现方式不太一样。例如前面的 CGElement 基类，把它重新定义为接口的形式，如下所示：

【VB.NET】

```
Public Interface IGElement
    Property ID() As Integer
    Sub Draw()
End Interface
```

【VC#.NET】

```
public interface IGElement
{
    int ID
    {
        get;
        set;
    }
    void Draw();
}
```

其中必须使用 Interface 关键字或 interface 关键字。在 VB.NET 中定义属性只需要在 Property 后面跟属性名称和类型，不需要属性过程；定义方法只需要在过程关键字后面给出方法名；属性和方法都不需要使用 Private, Public 等指定范围。在 VC#.NET 中定义属性时需要给出 get 和 set，方法只需要给出方法名。

2.3.2　实现 IGElement 接口

在 VB.NET 中实现接口需要使用 Implements 关键字，VC#.NET 中则像指定继承关系那样使用冒号。

【VB.NET】

```
Public Class CLine
    Implements IGElement

    Private m_ID As Integer
```

```
    Private m_Begin,m_End As PointF

    Public Property ID() As Integer Implements IGElement.ID
        Get
            Return m_ID
        End Get
        Set(ByVal Value As Integer)
            m_ID = Value
        End Set
    End Property

    Public Sub Draw() Implements IGElement.Draw
        Console.WriteLine("绘直线段。")
    End Sub

End Class
```

【VC#.NET】

```
public class CLine:IGElement
{
    public CLine()
    {
    }

    private int    m_ID;
    private PointF m_Begin,m_End;

    public int ID
    {
        get{return m_ID;}
        set{m_ID =value;}
    }

    public void Draw()
    {
        Console.WriteLine("绘直线段。");
    }
}
```

2.3.3 测试 IGElement 接口

定义接口并创建实现接口的类以后，在 Form1 类中添加下面的代码进行测试。首先创建一个 IGElement 类型的接口，用它引用 CLine 类实例，然后将它的 ID 属性设置为 1，并调用 Draw 方法。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
    Dim ge As IGElement = New CLine()
    ge.ID = 1
    Console.WriteLine("ID:{0}", ge.ID)
    ge.Draw()
End Sub
```

【VC#.NET】

```
private void Form1_Load(object sender, System.EventArgs e)
{
      IGElement ge= new CLine();
      ge.ID = 1;
      Console.WriteLine("ID:{0}", ge.ID);
      ge.Draw();
}
```

输出窗口中的结果如图 2-5 所示。

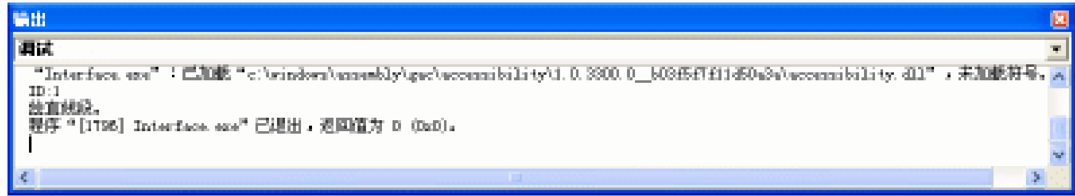

图 2-5 接口测试

2.4 多态

很多人常把多态和继承混淆起来，实际上二者区别较大，一个明显的例证便是，VB 6.0 中没有继承，但有多态。一般的理解是，继承强调的是自上而下的“血缘关系”，是“是一个”、“是一种”的关系；而多态只强调两个对象在行为上的相似性，而不管这两个对象之间是什么关系。有继承关系的对象一般都有多态关系，而有多态关系的对象间未必有继承关系。比如，马有跑的行为，蒙古马和伊犁马都是马的一种，它们都与马有继承关系，都有跑这个行为，在“跑”这个意义上，它们有多态关系。同样，机器猫也能跑，机器猫与马在跑这个行为上有多态关系，但它们没有继承关系。

VB 6.0 中有两种方法可以实现多态，即后期绑定和接口。在本套书的 VB 篇中我们大量采用了接口这种方法来实现多态。在性能上，它比用后期绑定实现要好些。.NET 中还可以通过继承和.NET 反射来实现多态，下面主要介绍继承和接口这两种实现途径。

2.4.1　用继承实现多态

在 2.2 节介绍继承时创建了基类 CGElement 和派生类 CLine。下面再创建一个派生类 CRectangle。CRectangle 类用于定义矩形的特征和方法。为了演示多态，这里只添加一个无参构造函数和一个 Draw 方法。Draw 方法重写了基类的 Draw 方法。

【VB.NET】

```
Public Class CRectangle
    Inherits CGElement

    Public Sub New()
    End Sub

    Public Overrides Sub Draw()
        Console.WriteLine("绘矩形。")
    End Sub
End Class
```

【VC#.NET】

```
public class CRectangle:CGElement
{
    public CRectangle()
    {
    }

    public override void Draw()
    {
        Console.WriteLine("绘矩形。");
    }
}
```

然后在 Form1 类中添加下面的代码，创建两个 CGElement 类型的变量并分别作为一个 CLine 类实例和一个 CRectangle 类实例的引用，然后调用 CGElement 类的 Draw 方法绘制直线段和矩形。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
    Dim ge1 As CGElement = New CLine()
    Dim ge2 As CGElement = New CRectangle()
```

```
        ge1.Draw()
        ge2.Draw()
    End Sub
```

【VC#.NET】

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
        CGElement ge1= new CLine();
        CGElement ge2= new CRectangle();
        ge1.Draw();
        ge2.Draw();
    }
```

程序运行结果如图 2-6 所示。可见，由于实现了多态，可以自动区分是绘制直线段还是绘制矩形。

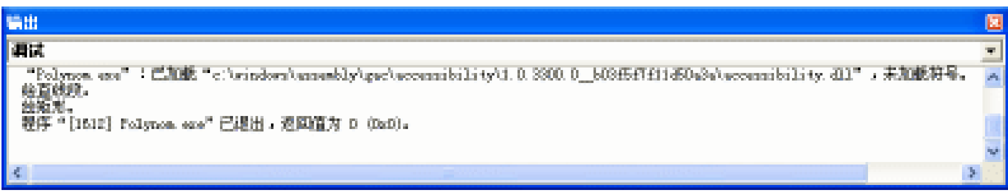

图 2-6　用继承实现多态

2.4.2　用接口实现多态

沿用 2.3 节的例子，仿照 2.4.1 节添加 CRectangle 类，它实现了 IGElement 接口。

【VB.NET】

```
    Public Class CRectangle
        Implements IGElement
        Private m_ID As Integer

        Public Property ID() As Integer Implements IGElement.ID
            Get
                Return m_ID
            End Get
            Set(ByVal Value As Integer)
                m_ID = Value
            End Set
        End Property

        Public Sub Draw() Implements IGElement.Draw
```

```
            Console.WriteLine("绘矩形。")
        End Sub
    End Class
```

【VC#.NET】

```
    public class CRectangle:IGElement
    {
        private int m_ID;

        public int ID
        {
            get{return m_ID;}
            set{m_ID = value;}
        }

        public void Draw()
        {
            Console.WriteLine("绘矩形。");
        }
    }
```

在 Form1 类中添加下面的代码，对实现了 IGElement 接口的类进行测试。

【VB.NET】

```
    Private Sub Form1_Load(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ge1 As IGElement = New CLine()
        Dim ge2 As IGElement = New CRectangle()
        ge1.ID = 1
        ge2.ID = 11
        Console.WriteLine("ID:{0},{1}", ge1.ID, ge2.ID)
        ge1.Draw()
        ge2.Draw()
    End Sub
```

【VC#.NET】

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
        IGElement ge1= new CLine();
        IGElement ge2= new CRectangle();
        ge1.ID = 1;
        ge2.ID = 11;
        Console.WriteLine("ID:{0},{1}", ge1.ID, ge2.ID);
```

```
            ge1.Draw();
            ge2.Draw();
        }
```

程序运行结果如图 2-7 所示。可见，利用接口也能很好地实现多态。

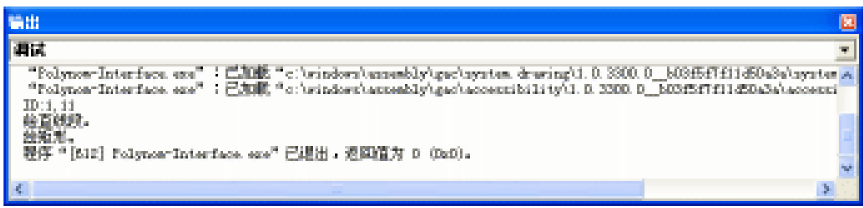

图 2-7　用接口实现多态

2.4.3　两种方式的比较

上面两种方式都能实现多态，而且性能相近。一般来说，当对象存在继承关系时用继承实现多态，不存在继承关系时用接口实现多态。

第 3 章　GDI+ 编 程

GDI+是 GDI 的改进版本，是一个用于在 Windows 中进行二维图形处理的类库。与 GDI 相比，GDI+中添加了一些新的绘图功能，采用了一些新的编程模式。新功能主要包括渐变色画笔、基数样条曲线、路径对象、图形变换、Alpha 混合和对多种图像格式的支持等。新的编程模式包括对画笔、颜色、路径、区域和图像等的不同处理。

3.1　Graphics 对象

3.1.1　创建和使用 Graphics 对象

用计算机语言编写成的程序是对相关事物和关系的高度抽象和概括。为了更好地理解这一节的内容，我们从绘画这件事情说起。我们的画，可以绘在纸上，布上，墙壁上，木板上，或者电脑屏幕上。用计算机语言来描述绘画这件事情的时候，就要进行抽象，这些纸、布、墙壁、木板、电脑屏幕可以全部抽象成一个表面，即一个可以绘图的表面。

GDI+中有绘图表面的概念，在这里，绘图表面可以是窗体、控件、打印机、预览或者图像。当指定绘图表面是图片框以后，绘图行为将在图片框上发生，而不会画到窗体上去。指定绘图表面是窗体时，绘图行为就在窗体上发生，也不会画在图片框上。

但问题是，谁来指定绘图表面呢？想起一个笑话：老鼠们开会，议题是如何对付猫。有鼠提议，给猫脖子上挂个铃铛不就得了。提议近于完美，但问题是谁去给猫脖子上挂铃铛呢？

所以由谁来指定是问题的关键。

这里是由 Graphics 对象来指定。就像 GDI 中的设备上下文一样，Graphics 对象与特定的窗口相关联，指定绘图表面。

虽然把 Graphics 对象与上面的笑话联系在一起不太恰当，但关于谁来指定的问题的确太重要了。

所以，这个 Graphics 对象太重要了，有人甚至说它是最重要的。但本人以为如果把 Graphics 对象比做纸的话，那还得有笔有墨这画才画得成。说哪个最重要就像讨论先有蛋还是先有鸡一样，不是我们研究的范畴。但不管哪个最重要，用 GDI+进行具体的绘图工作以前，必须创建一个 Graphics 对象的实例却是千真万确的事实。

根据绘图表面的不同，有不同的创建 Graphics 对象实例的方法。当指定绘图表面为窗体时，下面的代码介绍了创建 Graphics 对象实例的两种方法。创建实例以后，对于该实例所做的任何绘图行为都作用在窗体上。

【VB.NET】

```
Public Class Form1
    Inherits System.Windows.Forms.Form
```

```
        '……

        Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
            Dim g As Graphics = e.Graphics
            '……
        End Sub

        Private Sub Button1_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button1.Click
            Dim g As Graphics = Me.CreateGraphics()
            '……
        End Sub
    End Class
```

【VC#.NET】

```
    namespace Graph
    {
        public class Form1 : System.Windows.Forms.Form
        {
            //……

            protected override void OnPaint(PaintEventArgs e)
            {
                Graphics g = e.Graphics;
                //……
            }

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

第 1 种方法假设当前过程是绘制窗体的 OnPaint 方法，则可以直接从该方法的参数 e 中获得 Graphics 对象的实例；第 2 种方法假设通过单击一个命令按钮来实现在窗体上绘图，此时调用 Form1 类的 CreateGraphics 方法创建 Graphics 对象实例。

如果指定绘图表面为控件，如图片框，则必须通过调用该控件的 CreateGraphics 方法来创建 Graphics 对象实例。下面的语句创建图片框控件 PictureBox1(pictureBox1)的 Graphics 对象实例。

【VB.NET】

```
Dim g As Graphics = PictureBox1.CreateGraphics()
```

【VC#.NET】

```
Graphics g= pictureBox1.CreateGraphics();
```

创建 Graphics 对象的实例以后，就有了画布，但还要有笔和墨。在 GDI+中，代表笔的 Pen 对象和代表墨的 Color 对象是独立于代表纸的 Graphics 对象的，将在后面的小节中进行介绍。但关于 Graphics 对象的内容还远远没完。试想，我们讲书法要讲究章法，绘画要讲究构图，其实就隐含了坐标参照系的概念；写生素描要涉及到比例问题；用计算机绘图，还要提供组成任何几何图形所需要的基本图形元素如直线段、椭圆、椭圆弧、曲线、区域、文本等的绘制方法，这些都由 Graphics 对象提供。表 3-1 中示出了 Graphics 对象的部分属性，表 3-2 中示出了 Graphics 对象的部分方法。表中的内容将在后续章节中陆续用到，届时再进行详细说明。

表 3-1　Graphics 对象的部分属性

属　　性	说　　明
Clip	获取或设置 Region 对象，该对象限定此 Graphics 对象的绘图区域
ClipBounds	获取 RectangleF 结构，该结构限定此 Graphics 对象的剪辑区域
DpiX	获取此 Graphics 对象的水平分辨率
DpiY	获取此 Graphics 对象的垂直分辨率
PageScale	获取或设置此 Graphics 对象的全局单位和页单位之间的比例
PageUnit	获取或设置用于此 Graphics 对象中的页坐标的度量单位
SmoothingMode	获取或设置此 Graphics 对象的呈现质量
Transform	获取或设置此 Graphics 对象的全局变换

表 3-2　Graphcis 对象的部分方法

方　　法	说　　明
Clear	清除整个绘图面并以指定背景色填充
Dispose	释放由此 Graphics 对象使用的所有资源
DrawArc	绘制一段弧线，它表示由一对坐标、宽度和高度指定的椭圆部分
DrawBezier	绘制由 4 个 Point 结构定义的贝塞尔样条
DrawClosedCurve	绘制由 Point 结构的数组定义的闭合基数样条
DrawCurve	绘制经过一组指定的 Point 结构的基数样条
DrawEllipse	绘制一个由边框（该边框由一对坐标、高度和宽度指定）定义的椭圆
DrawIcon	在指定坐标处绘制由指定的 Icon 对象表示的图像
DrawImage	在指定位置并且按原始大小绘制指定的 Image 对象
DrawLine	绘制一条连接由坐标对指定的两个点的线条
DrawPath	绘制 GraphicsPath 对象
DrawPie	绘制一个扇形，该扇形由一个坐标对、宽度和高度以及两条射线所指定的椭圆定义
DrawPolygon	绘制由一组 Point 结构定义的多边形

续表

方　　法	说　　明
DrawRectangle	绘制由坐标对、宽度和高度指定的矩形
DrawRectangles	绘制一系列由 Rectangle 结构指定的矩形
DrawString	在指定位置并且用指定的 Brush 和 Font 对象绘制指定的文本字符串
FillEllipse	填充边框所定义的椭圆的内部，该边框由一对坐标、一个宽度和一个高度指定
FillPath	填充 GraphicsPath 对象的内部
FillPie	填充由一对坐标、一个宽度、一个高度以及两条射线指定的椭圆所定义的扇形区的内部
FillPolygon	填充 Point 结构指定的点数组所定义的多边形的内部
FillRectangle	填充由一对坐标、一个宽度和一个高度指定的矩形的内部
FillRectangles	填充由 Rectangle 结构指定的一系列矩形的内部
FillRegion	填充 Region 对象的内部
GetHdc	获取与此 Graphics 对象关联的设备上下文的句柄
ResetTransform	将此 Graphics 对象的全局变换矩阵重置为单位矩阵
RotateTransform	将指定旋转应用于此 Graphics 对象的变换矩阵
ScaleTransform	将指定的缩放操作应用于此 Graphics 对象的变换矩阵，方法是将其添加到该对象的变换矩阵前
TranslateTransform	将指定的平移添加到此 Graphics 对象的变换矩阵前

下面举一个例子，介绍 Graphics 对象的创建和使用。该示例在窗体上放置一个图片框 PictureBox1(pictureBox1)和两个命令按钮 Button1(button1)和 Button2(button2)，要求重绘窗体时在图片框周围绘一个红色矩形 单击两个命令按钮时分别在窗体和图片框上绘制蓝色椭圆。本例用到了 Pen 对象、Brushes 对象和 Color 对象。这里暂不做详细介绍，可参见后续章节的内容。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    Dim x As Single, y As Single
    Dim w As Single, h As Single
    With PictureBox1
        x = .Left - 2
        y = .Top - 2
        w = .Width + 4
        h = .Height + 4
    End With
    Dim pen As New Pen(Color.Red, 2)
    g.DrawRectangle(pen, x, y, w, h)
End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles Button1.Click
    Dim g As Graphics = Me.CreateGraphics()
    g.FillEllipse(Brushes.Blue, 10, 20, 50, 100)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles Button2.Click
    Dim g As Graphics = PictureBox1.CreateGraphics()
    g.FillEllipse(Brushes.Blue, 10, 20, 50, 100)
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    float x, y,w, h;
    x = pictureBox1.Left - 2;
    y = pictureBox1.Top - 2;
    w = pictureBox1.Width + 4;
    h = pictureBox1.Height + 4;
    Pen pen=new Pen(Color.Red, 2);
    g.DrawRectangle(pen, x, y, w, h);
}

private void button1_Click(object sender, EventArgs e)
{
    Graphics g= this.CreateGraphics();
    g.FillEllipse(Brushes.Blue, 10, 20, 50, 100);
}

private void button2_Click_1(object sender, EventArgs e)
{
    Graphics g= pictureBox1.CreateGraphics();
    g.FillEllipse(Brushes.Blue, 10, 20, 50, 100);
}
```

运行程序，单击两个命令按钮以后窗体显示结果如图 3-1 所示。

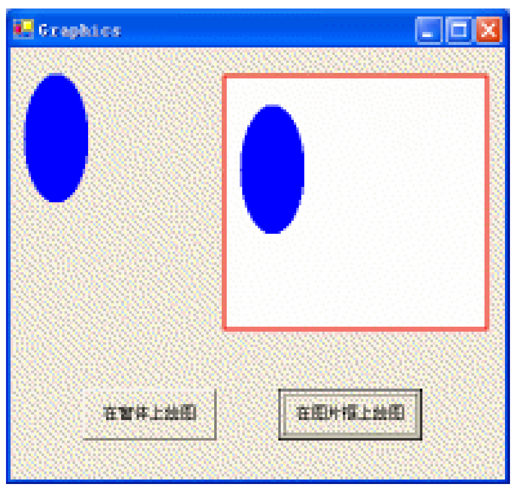

图 3-1　创建和使用 Graphics 对象

3.1.2　Paint 事件和 OnPaint 方法

上一节的例子中我们用到了 OnPaint 方法，它重写基类的 OnPaint 方法，用于绘制窗体。在 VB 6.0 中，绘制窗体用的是窗体的 Paint 事件，每次要求绘制窗体时触发该事件。.NET 中绘制窗体也可以用 Paint 事件。如：

【VB.NET】

```
Private Sub Form1_Paint(ByVal sender As Object, _
                ByVal e As PaintEventArgs) Handles MyBase.Paint
    Dim g As Graphics = e.Graphics
    g.DrawEllipse(Pens.Blue, 50, 50, 100, 100)
End Sub
```

【VC#.NET】

```
private void Form1_Paint(object sender,PaintEventArgs e)
{
    Graphics g=e.Graphics;
    g.DrawEllipse(Pens.Blue,50,50,100,100);
}
```

这样，实现绘制窗体的任务就有两种方法供我们选择，但两种方法是有差别的。用 Paint 事件绘制窗体时，会调用基类的 OnPaint 方法，而用重写的 OnPaint 方法绘制窗体时则不需要调用基类的 OnPaint 方法。所以，前者的消耗要大一些。

3.2　线条绘制

3.2.1　颜色

GDI+中用 Color 结构定义绘图工具的颜色。下面的代码定义一个蓝色对象 c：

【VB.NET】

```
Dim c As Color = Color.Blue
```

【VC#.NET】

```
Color c=Color.Blue;
```

采用这种定义方式，可以定义标准颜色。所谓标准颜色，是系统已经配好的颜色，具有各自的名称。利用 Color 结构的 FromArgb 方法，可以自定义任何颜色。

【VB.NET】

```
Dim c2, c3 As Color
c2.FromArgb(255, 0, 0)
c3.FromArgb(200, 0, 0, 255)
```

【VC#.NET】

```
Color c2, c3;
c2.FromArgb(255, 0, 0);
c3.FromArgb(200, 0, 0, 255);
```

上面代码中，c2 定义为红色，c3 定义为蓝色，且 Alpha 组分值为 200。Alpha 定义颜色的透明度，将在后面介绍。

3.2.2 画笔

常用 Pen 类定义画笔。下面的代码定义一支绿色画笔和一支宽度为 2 的红色画笔。

【VB.NET】

```
Dim p1 As New Pen(Color.Green)
Dim p2 As New Pen(Color.Red, 2)
```

【VC#.NET】

```
Pen p1=new Pen(Color.Green);
Pen p2=new Pen(color.red,2);
```

与标准颜色相对应，GDI+ 提供了若干标准颜色的画笔。标准颜色的画笔用 Pens 类定义，如：

【VB.NET】

```
Dim p As Pen = Pens.BlueViolet
```

【VC#.NET】

```
Pen p= Pens.BlueViolet;
```

通过设置画笔对象的属性，可以定义画笔的颜色、宽度、线条的线型、起点和终点的形状等。Pen 类的常用属性如表 3-3 所示。

表 3-3　Pen 类的常用属性

属　　性	说　　明
Color	获取或设置此 Pen 对象的颜色
DashCap	获取或设置用在短划线终点的帽样式，这些短划线构成通过此 Pen 对象绘制的虚线
DashStyle	获取或设置用于通过此 Pen 对象绘制的虚线的样式
EndCap	获取或设置用在通过此 Pen 对象绘制的直线终点的帽样式
LineJoin	获取或设置通过此 Pen 对象绘制的两条连续直线终点之间的连接样式
StartCap	获取或设置用在通过此 Pen 对象绘制的直线起点的帽样式
Transform	获取或设置此 Pen 对象的几何变换
Width	获取或设置此 Pen 对象的宽度

3.2.3　线条绘制示例

下面举一个绘制线条的例子。该例实现在图片框中绘制直线段、椭圆、圆弧、多义线、贝塞尔曲线和基数样条曲线。如图 3-2 所示，进入编程环境以后，在窗体上放置一个图片框和 7 个命令按钮，各控件的属性设置如表 3-4 所示。

图 3-2　程序界面

表 3-4　控件属性设置

控 件 类 型	名　　称	标　　注
PictureBox	PictureBox1	
Button	line	直线
	elllipse	椭圆
	arc	圆弧
	poly	多义线
	curve	曲线
	clear	清除
	close	关闭

程序界面设计完毕以后，在 Form1 类的代码框架中添加下面的代码：

【VB.NET】

```
Private g As Graphics

  Private Sub line_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles line.Click
      g = PictureBox1.CreateGraphics()
      Dim pen1 As New Pen(Color.Blue, 3)
      pen1.DashStyle = Drawing.Drawing2D.DashStyle.DashDot
      g.DrawLine(pen1, 10, 10, 200, 200)
      Dim pen2 As New Pen(Color.Red)
      pen2.EndCap = Drawing.Drawing2D.LineCap.ArrowAnchor
      g.DrawLine(pen2, 200, 50, 20, 150)
End Sub

Private Sub ellipse_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles ellipse.Click
      g = PictureBox1.CreateGraphics()
      g.DrawEllipse(Pens.Red, 30, 50, 300, 200)
End Sub

  Private Sub arc_Click(ByVal sender As System.Object, _
                 ByVal e As System.EventArgs) Handles arc.Click
      g = PictureBox1.CreateGraphics()
      Dim pen As New Pen(Color.FromArgb(100, 200, 0, 200), 2)
      g.DrawArc(pen, 100, 40, 100, 100, 30, 290)
End Sub

Private Sub poly_Click(ByVal sender As System.Object, _
               ByVal e As System.EventArgs) Handles poly.Click
      g = PictureBox1.CreateGraphics()
      Dim points() As PointF = New PointF() { _
                  New PointF(90, 10), New PointF(300, 100), _
                  New PointF(150, 200), New PointF(100, 80)}
      g.DrawPolygon(Pens.BlueViolet, points)
  End Sub

Private Sub curve_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles curve.Click
      g = PictureBox1.CreateGraphics()
```

```
        Dim pen As New Pen(Color.Cyan, 2)
        pen.DashStyle = Drawing.Drawing2D.DashStyle.DashDotDot
        g.DrawBezier(pen, New PointF(90, 10), New PointF(300, 100), _
                    New PointF(150, 200), New PointF(100, 80))
        Dim points() As PointF = New PointF() { _
                   New PointF(90, 10), New PointF(300, 100), _
                   New PointF(150, 200), New PointF(100, 80)}
        g.DrawCurve(Pens.Blue, points)
    End Sub

    Private Sub clear_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles clear.Click
        g = PictureBox1.CreateGraphics()
        g.Clear(Color.White)
    End Sub

    Private Sub close_Click(ByVal sender As System.Object, _
               ByVal e As System.EventArgs) Handles close.Click
        Me.Dispose()
    End Sub
```

【VC#.NET】

```
    private Graphics g;

    private void line_Click(object sender, System.EventArgs e)
    {
        g = pictureBox1.CreateGraphics();
        Pen pen1=new Pen(Color.Blue, 3);
        pen1.DashStyle = DashStyle.DashDot;
        g.DrawLine(pen1, 10, 10, 200, 200);
        Pen pen2=new Pen(Color.Red);
        pen2.EndCap = LineCap.ArrowAnchor;
        g.DrawLine(pen2, 200, 50, 20, 150);
    }

    private void ellipse_Click(object sender, System.EventArgs e)
    {
        g = pictureBox1.CreateGraphics();
        g.DrawEllipse(Pens.Red, 30, 50, 300, 200);
    }
```

```
private void arc_Click(object sender, System.EventArgs e)
{
    g = pictureBox1.CreateGraphics();
    Pen pen=new Pen(Color.FromArgb(100, 200, 0, 200), 2);
    g.DrawArc(pen, 100, 40, 100, 100, 30, 290);
}

private void poly_Click(object sender, System.EventArgs e)
{
    g = pictureBox1.CreateGraphics();
    PointF[] points= new PointF[] {
    new PointF(90, 10), new PointF(300, 100),
    new PointF(150, 200), new PointF(100, 80)};
    g.DrawPolygon(Pens.BlueViolet, points);
}

private void curve_Click(object sender, System.EventArgs e)
{
    g = pictureBox1.CreateGraphics();
    Pen pen=new Pen(Color.Cyan, 2);
    pen.DashStyle =DashStyle.DashDotDot;
    g.DrawBezier(pen, new PointF(90, 10), new PointF(300, 100),
    new PointF(150, 200), new PointF(100, 80));
    PointF[] points = new PointF[] {
    new PointF(90, 10), new PointF(300, 100),
    new PointF(150, 200), new PointF(100, 80)};
    g.DrawCurve(Pens.Blue, points);
}

private void clear_Click(object sender, System.EventArgs e)
{
    g = pictureBox1.CreateGraphics();
    g.Clear(Color.White);
}

private void close_Click(object sender, System.EventArgs e)
{
    this.Dispose();
}
```

运行程序，绘图效果如图 3-3、图 3-4 所示。图 3-3 中定义了两条不同颜色的直线段，其中一条为虚点线，另一条终点处有箭头。图 3-4 中显示了用 4 个点所确定的多边形、贝塞尔曲线（蓝绿色曲线）和基数样条曲线（蓝色曲线）。

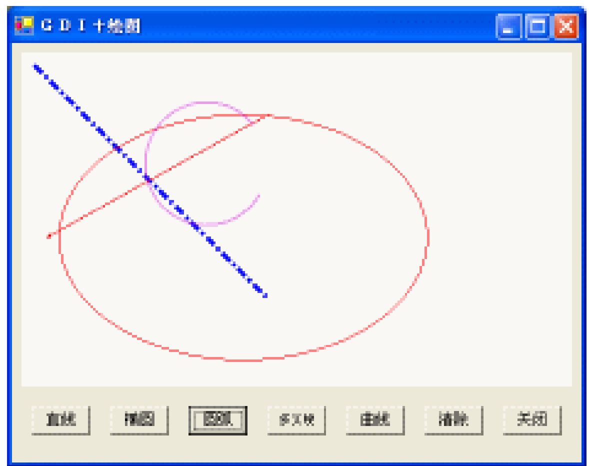

图 3-3　线条绘制效果一

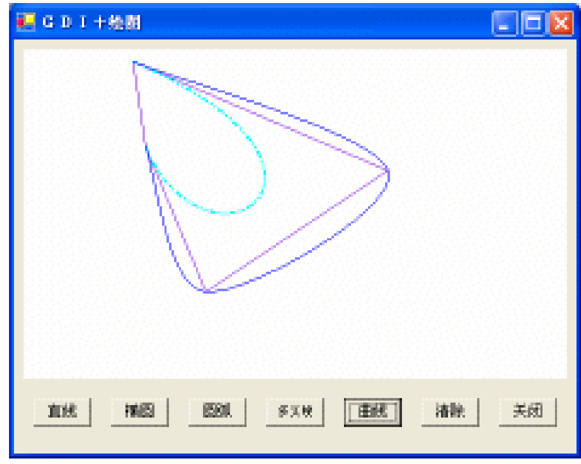

图 3-4　线条绘制效果二

3.3　文本

3.3.1　FontFamily 类

FontFamily 类定义有着相似的基本设计但在形式上有某些差异的一组字样。下面的语句定义一个字体名称为“宋体”的 FontFamily 类实例。

【VB.NET】

```
Dim ff As New FontFamily("宋体")
```

【VC#.NET】

```
FontFamily ff=new FontFamily("宋体");
```

如果不是使用默认字体，或者不是从其他字体创建新字体，则大多数情况下需要创建 FontFamily 类对象，并与 Font 类一起使用来定义新字体。

3.3.2　Font 类

Font 类定义特定的文本格式，包括字体、字号和字形属性。如上面定义了 FontFamily 类实例 ff，下面使用它创建两个 Font 对象：

【VB.NET】

```
Dim f1 As New Font(ff, 20)
Dim f2 As New Font(ff, 20, FontStyle.Bold)
```

【VC#.NET】

```
Font f1=new Font(ff, 20);
Font f2=new Font(ff, 20, FontStyle.Bold);
```

调用 Font 类的属性，可以获取字体的名称、度量单位、大小和样式等信息。常用的 Font 类属性如表 3-5 所示。

表 3-5　常用的 Font 类属性

属　　性	说　　明
Bold	获取一个值，该值指示此 Font 对象是否为粗体
FontFamily	获取与此 Font 对象关联的 FontFamily 对象
Height	获取此 Font 对象的高度
Italic	获取一个值，它指示此 Font 对象是否为斜体
Name	获取此 Font 对象的字体名称
Size	获取此 Font 对象的全身大小，采用设计单位
SizeInPoints	获取此 Font 对象的字号，以磅为单位
Strikeout	获取一个值，该值指示此 Font 对象是否指定通过该字体的横线
Underline	获取一个值，该值指示此 Font 对象是否有下划线
Unit	获取此 Font 对象的度量单位

3.3.3　StringFormat 类

StringFormat 类可以设置文本对齐方式和行距等。下面指定文本格式为行对中，按列显示。

【VB.NET】

```
Dim sf As New StringFormat()
sf.LineAlignment = StringAlignment.Center
sf.FormatFlags = StringFormatFlags.DirectionVertical
```

【VC#.NET】

```
StringFormat sf=new StringFormat();
sf.LineAlignment = StringAlignment.Center;
sf.FormatFlags = StringFormatFlags.DirectionVertical;
```

上面代码中，sf 的 FormatFlags 属性设置为一个 StringFormatFlags 枚举值。StringFormatFlags 枚举值指示文本的显示和布局信息，其成员和说明如表 3-6 所示。

表 3-6 StringFormatFlags 枚举

成员名称	说明
DirectionRightToLeft	指定文本从右到左排列
DirectionVertical	指定文本垂直排列
DisplayFormatControl	导致控制字符（如从左到右标记）随具有代表性的标志符号一起显示在输出中
FitBlackBox	指定任何标志符号的任何部分都不突出边框
LineLimit	在格式化的矩形中只布置整行
MeasureTrailingSpaces	在缺省情况下，MeasureString 方法返回的边框都将删除每一行结尾处的空格。设置此标记以便在测定时将空格包括进去
NoClip	允许显示标志符号的伸出部分和延伸到边框外的未换行文本。在缺省情况下，延伸到边框外侧的所有文本和标志符号部分都被剪裁
NoFontFallback	对于请求的字体中不支持的字符，禁用回退到可选字体
NoWrap	在矩形中进行格式化时禁用文本换行

3.3.4 刷子

注意，GDI+ 里面，文字是用刷子刷的，而不是用笔写的。使用 Brushes 类可以定义标准颜色的刷子，其使用方法与 Pens 类相似。除了 Brushes 类，GDI+ 还提供了表 3-7 所示的 5 个类。利用这 5 个类，可以定义纯色刷子、图像刷子、线性梯度刷子、路径梯度刷子和阴影刷子。

表 3-7 定义刷子的类

类	说明
SolidBrush	纯色刷子，用纯色填充形状内部
TextureBrush	用图像填充形状内部
LinearGradientBrush	线性梯度刷子，用线性渐变的颜色填充形状内部
PathGradientBrush	路径梯度刷子，用沿路径渐变的颜色填充形状内部
HatchBrush	用阴影样式图案填充形状内部

下面的代码创建一把纯色刷子。

【VB.NET】

```
Dim brush As New SolidBrush(Color.Blue)
```

【VC#.NET】

```
SolidBrush brush=new SolidBrush(Color.Blue);
```

3.3.5　DrawString 方法

利用前面介绍的几个类，可以设置文本的字体、显示格式和绘制工具。具体绘制文本时需要用到 Graphics 类的 DrawString 方法。该方法具有下面一些重载：

```
Public Sub DrawString(String,Font,Brush,PointF)
Public Sub DrawString(String,Font,Brush,PointF,StringFormat)
Public Sub DrawString(String,Font,Brush,RectangleF,StringFormat)
Public Sub DrawString(String,Font,Brush,Single,Single)
Public Sub DrawString(String,Font,Brush,Single,Single,StringFormat)
```

下面的例子在前面示例定义的基础上调用 Graphics 对象 g 的 DrawString 方法，在窗体上(100,100)的位置绘制字符串“ 你好 !”。

【VB.NET】

```
Dim s As String = "你好!"
g.DrawString(s, f2, brush, 100, 100)
```

【VC#.NET】

```
String s = "你好!";
g.DrawString(s, f2, brush, 100, 100);
```

3.3.6　文本绘制示例

下面的例子将一首十六字令竖排显示在指定的矩形框中。字体为华文新魏体，蓝色。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim s As String = "离天三尺三              惊回首                        " _
                      & "快马加鞭未下鞍  山"
    Dim fm As New FontFamily("华文新魏")
    Dim f As New Font(fm, _
        20, FontStyle.Bold, _
        GraphicsUnit.Point)
    Dim rectF As New RectangleF(30, 20, 180, 205)
    Dim sf As New StringFormat()
    Dim sbrush As New SolidBrush(Color.FromArgb(255, 0, 0, 255))
    sf.LineAlignment = StringAlignment.Center
    sf.FormatFlags = StringFormatFlags.DirectionVertical
    g.DrawString(s, f, sbrush, rectF, sf)
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    string s= "离天三尺三          惊回首                    "
                        + "快马加鞭未下鞍  山";
    FontFamily fm=new FontFamily("华文新魏");
    Font f =new Font(fm,
        20, FontStyle.Bold,
        GraphicsUnit.Point);
    RectangleF rectF= new RectangleF(30, 20, 180, 205);
    StringFormat sf=new StringFormat();
    SolidBrush sbrush=new SolidBrush(Color.FromArgb(255, 0, 0, 255));
    sf.LineAlignment = StringAlignment.Center;
    sf.FormatFlags = StringFormatFlags.DirectionVertical;
    g.DrawString(s, f, sbrush, rectF, sf);
}
```

程序运行结果如图 3-5 所示。

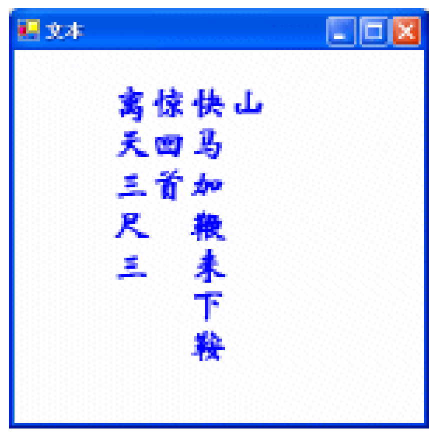

图 3-5　文本绘制

3.4　路径

道路有直有曲，有坦途也有羊肠小道。如果把一组线形几何图形比做一个路线图的话，那么“路径”这个词用来表示这组线形图是恰当的。路径中可以有直线段、圆、圆弧或者曲线，它们按顺序组合，可以相连，也可以不相连。在 GDI+中，路径由 GraphicsPath 类定义。

3.4.1　GraphicsPath 类

GraphicsPath 类提供了一系列属性和方法。利用它们，可以获取路径上的关键点，可以添加直线段、圆等几何元素，可以获取包围矩形，进行拾取测试等。GraphicsPath 类的部分常用方法如表 3-8 所示。

表 3-8　GraphicsPath 类的部分常用方法

方　法	说　明
AddArc	向当前图形追加一段椭圆弧
AddBezier	向当前图形添加一段立方贝塞尔曲线
AddBeziers	向当前图形添加一系列相互连接的立方贝塞尔曲线
AddClosedCurve	向此路径添加一个闭合曲线。由于曲线经过数组中的每个点，因此使用基数样条曲线
AddCurve	向当前图形添加一段样条曲线。由于曲线经过数组中的每个点，因此使用基数样条曲线
AddEllipse	向当前路径添加一个椭圆
AddLine	向此 GraphicsPath 对象追加一条线段
AddLines	向此 GraphicsPath 对象末尾追加一系列相互连接的线段
AddPath	将指定的 GraphicsPath 对象追加到该路径
AddPie	向此路径添加一个扇形轮廓
AddPolygon	向此路径添加多边形
AddRectangle	向此路径添加一个矩形
AddRectangles	向此路径添加一系列矩形
AddString	向此路径添加文本字符串
Clone	创建此路径的一个精确副本
CloseAllFigures	闭合此路径中所有开放的图形并开始一个新图形。它通过连接一条从图形的终结点到起始点的直线，闭合每一开放的图形
CloseFigure	闭合当前图形并开始新的图形。如果当前图形包含一系列相互连接的直线和曲线，则该方法通过连接一条从终结点到起始点的直线，闭合该环回

下面的代码创建一个 GraphicsPath 类实例，然后添加一个矩形到路径中。

【VB.NET】

```
Dim gp As New GraphicsPath()
gp.AddRectangle(New Rectangle(10, 10, 100, 100))
```

【VC#.NET】

```
GraphicsPath gp=new GraphicsPath();
gp.AddRectangle(new Rectangle(10, 10, 100, 100));
```

注意，创建 GraphicsPath 类实例，需要导入 System.Drawing.Drawing2D 名称空间。即在代码窗口顶部加入下面的声明：

【VB.NET】

```
Imports System.Drawing.Drawing2D
```

【VC#.NET】

```
using System.Drawing.Drawing2D;
```

另外，路径不再使用后，需要删除，即：

【VB.NET】

```
gp.Dispose()
```

【VC#.NET】

```
gp.Dispose();
```

3.4.2 绘制和填充路径

定义路径以后，调用 Graphics 类的 DrawPath 方法和 FillPath 方法可以绘制和填充路径。下面把 3.4.1 节定义的路径绘制到窗体，然后平移后进行填充。其中，g 为 Graphics 类实例。

【VB.NET】

```
g.DrawPath(Pens.Blue, gp)
g.TranslateTransform(200, 0)
g.FillPath(Brushes.GreenYellow, gp)
```

【VC#.NET】

```
g.DrawPath(Pens.Blue, gp);
g.TranslateTransform(200, 0);
g.FillPath(Brushes.GreenYellow, gp);
```

3.4.3 路径定义示例

下面结合一个实例来介绍路径的定义和绘制。首先创建一条路径，然后在路径中先后添加椭圆、饼形和矩形各一个，最后在窗体上绘制路径并且平移后填充路径。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim gp As New GraphicsPath()
    gp.AddEllipse(20, 20, 300, 200)
    gp.AddPie(50, 100, 300, 100, 45, 200)
    gp.AddRectangle(New Rectangle(100, 30, 100, 80))
    g.DrawPath(Pens.Blue, gp)
    g.TranslateTransform(200, 20)
```

```
        g.FillPath(Brushes.GreenYellow, gp)
        gp.Dispose()
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        g.FillRectangle(Brushes.White,this.ClientRectangle);
        GraphicsPath gp=new GraphicsPath();
        gp.AddEllipse(20, 20, 300, 200);
        gp.AddPie(50, 100, 300, 100, 45, 200);
        gp.AddRectangle(new Rectangle(100, 30, 100, 80));
        g.DrawPath(Pens.Blue, gp);
        g.TranslateTransform(200, 20);
        g.FillPath(Brushes.GreenYellow, gp);
        gp.Dispose();
    }
```

程序运行结果如图 3-6 所示。

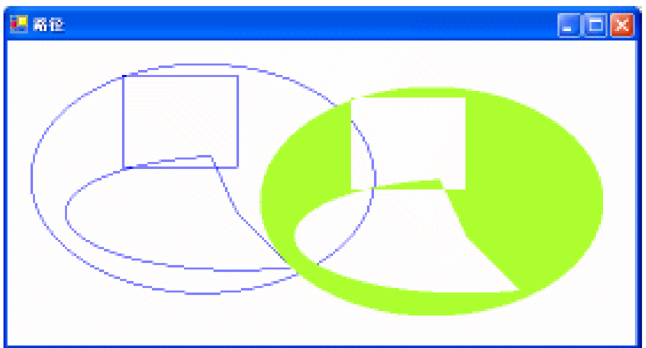

图 3-6　路径绘制和填充

3.5　区域

3.5.1　Region 类

使用 Region 类定义区域。该类的构造函数如下所示：

```
    Public Sub New()
    Public Sub New(GraphicsPath)
    Public Sub New(Rectangle)
    Public Sub New(RectangleF)
    Public Sub New(RegionData)
```

其中，GraphicsPath 参数为一个已经指定的路径，Rectangle 和 RectangleF 分别为整型和浮点型坐标下的矩形，RegionData 为指定 Region 对象的字节数组。可见，从已有的矩形和路径可以创建区域。

区域通过 Graphics 类的 FillRegion 方法进行绘制。

下面的例子讲解了区域的创建和绘制方法。首先创建一个矩形区域，然后创建一个三角形和一个椭圆组成的路径，并根据该路径创建另一个区域。最后绘制这两个区域。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim r1 As New Region(New RectangleF(60, 50, 150, 100))

    Dim gp As New GraphicsPath()
    gp.AddLine(300, 50, 500, 100)
    gp.AddLine(500, 100, 400, 200)
    gp.CloseFigure()
    gp.StartFigure()
    gp.AddEllipse(50, 200, 200, 100)
    Dim r2 As New Region(gp)

    g.FillRegion(Brushes.CornflowerBlue, r1)
    g.FillRegion(Brushes.DarkOrange, r2)
    r1.Dispose()
    r2.Dispose()
    gp.Dispose()
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    Region r1=new Region(new RectangleF(60, 50, 150, 100));

    GraphicsPath gp=new GraphicsPath();
    gp.AddLine(300, 50, 500, 100);
    gp.AddLine(500, 100, 400, 200);
    gp.CloseFigure();
    gp.StartFigure();
    gp.AddEllipse(50, 200, 200, 100);
    Region r2=new Region(gp);
```

```
        g.FillRegion(Brushes.CornflowerBlue, r1);
        g.FillRegion(Brushes.DarkOrange, r2);
        r1.Dispose();
        r2.Dispose();
        gp.Dispose();
    }
```

程序运行结果如图 3-7 所示。

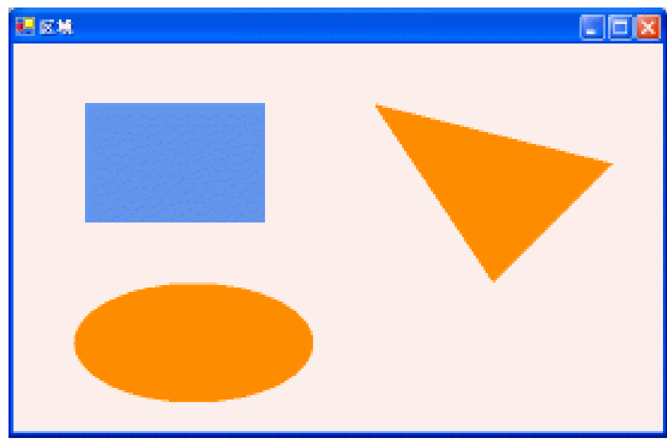

图 3-7　绘制区域

3.5.2　渐变色填充

在介绍文本时我们讲到了刷子，其中有两种刷子分别称为线性梯度刷子和路径梯度刷子。这两种刷子按照一定方向，用渐变的颜色填充区域。例如，下面定义一把线性梯度刷子，其线性渐变的起始点为(10,10)，终止点为(100,10)，起始点的颜色为红色，终止点的颜色为蓝色。

【VB.NET】

```
Dim lGBrush As New LinearGradientBrush( _
    New Point(10, 10), _
    New Point(100, 10), _
    Color.FromArgb(255, 0, 0), _
    Color.FromArgb(0, 0 , 255))
```

【VC#.NET】

```
LinearGradientBrush lGBrush=new LinearGradientBrush(
    new Point(10, 10),
    new Point(100, 10),
    Color.FromArgb(255, 0, 0),
    Color.FromArgb(0, 0, 255));
```

下面的例子使用线性梯度刷子和路径梯度刷子来填充椭圆、矩形和绘制直线段。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics

    '定义一个线性梯度刷子
    Dim lGBrush As New LinearGradientBrush( _
        New Point(0, 10), _
        New Point(150, 10), _
        Color.FromArgb(255, 0, 0), _
        Color.FromArgb(0, 255, 0))
    Dim pen As New Pen(lGBrush)
    '用线性梯度刷效果的笔绘制一条直线段，填充一个矩形
    g.DrawLine(pen, 10, 130, 500, 130)
    g.FillRectangle(lGBrush, 50, 150, 370, 30)

    '定义路径并添加一个椭圆
    Dim gp As New GraphicsPath()
    gp.AddEllipse(10, 10, 200, 100)
    '用该路径定义路径梯度刷子
    Dim brush As New PathGradientBrush(gp)
    '颜色数组
    Dim colors As Color() = { _
       Color.FromArgb(255, 0, 0), _
       Color.FromArgb(100, 100, 100), _
       Color.FromArgb(0, 255, 0), _
       Color.FromArgb(0, 0, 255)}
    ' 定义颜色渐变比率
    Dim r As Single() = { _
       0.0F, _
       0.3F, _
       0.6F, _
       1.0F}
    Dim blend As New ColorBlend()
    blend.Colors = colors
    blend.Positions = r
    brush.InterpolationColors = blend
    '在椭圆外填充一个矩形
    g.FillRectangle(brush, 0, 0, 210, 110)
```

```
        '用添加了椭圆的路径定义第 2 个路径梯度刷子
        Dim gp2 As New GraphicsPath()
        gp2.AddEllipse(300, 0, 200, 100)
        Dim brush2 As New PathGradientBrush(gp2)
        ' 设置中心点的位置和颜色
        brush2.CenterPoint = New PointF(450, 50)
        brush2.CenterColor = Color.FromArgb(0, 255, 0)
        '设置边界颜色
        Dim colors2 As Color() = {Color.FromArgb(255, 0, 0)}
        brush2.SurroundColors = colors2
        '用第 2 个梯度刷填充椭圆
        g.FillEllipse(brush2, 300, 0, 200, 100)
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g=e.Graphics;
        //定义一个线性梯度刷子
        LinearGradientBrush lGBrush=new LinearGradientBrush(
            new Point(0, 10),
            new Point(150, 10),
            Color.FromArgb(255, 0, 0),
            Color.FromArgb(0, 255, 0));
        Pen pen=new Pen(lGBrush);
        //用线性梯度刷效果的笔绘制一条直线段，填充一个矩形
        g.DrawLine(pen, 10, 130, 500, 130);
        g.FillRectangle(lGBrush, 50, 150, 370, 30);

        //定义路径并添加一个椭圆
        GraphicsPath gp=new GraphicsPath();
        gp.AddEllipse(10, 10, 200, 100);
        //用该路径定义路径梯度刷子
        PathGradientBrush brush=new PathGradientBrush(gp);
        //颜色数组
        Color[] colors= {
                Color.FromArgb(255, 0, 0),
                Color.FromArgb(100, 100, 100),
                Color.FromArgb(0, 255, 0),
                Color.FromArgb(0, 0, 255)};
        //定义颜色渐变比率
```

```
        float[] r = {
                0.0F,
                0.3F,
                0.6F,
                1.0F};
        ColorBlend blend=new ColorBlend();
        blend.Colors = colors;
        blend.Positions = r;
        brush.InterpolationColors = blend;
        //在椭圆外填充一个矩形
        g.FillRectangle(brush, 0, 0, 210, 110);

        //用添加了椭圆的路径定义第 2 个路径梯度刷子
        GraphicsPath gp2=new GraphicsPath();
        gp2.AddEllipse(300, 0, 200, 100);
        PathGradientBrush brush2=new PathGradientBrush(gp2);
        //设置中心点的位置和颜色
        brush2.CenterPoint = new PointF(450, 50);
        brush2.CenterColor = Color.FromArgb(0, 255, 0);
        //设置边界颜色
        Color[] colors2 = {Color.FromArgb(255, 0, 0)};
        brush2.SurroundColors = colors2;
        //用第 2 个梯度刷填充椭圆
        g.FillEllipse(brush2, 300, 0, 200, 100);
    }
```

程序运行结果如图 3-8 所示。

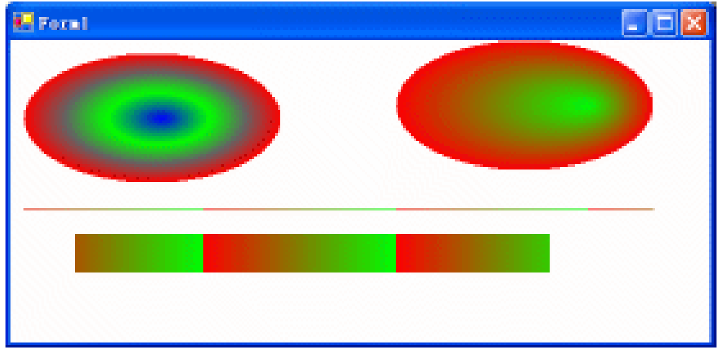

图 3-8　渐变色填充效果

3.6　坐标与变换

3.6.1　坐标系统

在本套书的 VB 篇中，讲到了 Windows 绘图的三套坐标系，即用户坐标系、逻辑坐标系和设备坐标系。与此对应，GDI+中也有三套坐标系，但名称有所不同。GDI+中的三套坐标系称为通用坐标系、页面坐标系和设备坐标系。

通用坐标系是用户自己定义的坐标系，比如按照我们的习惯，把原点放在屏幕左下角或者中央，让纵轴正向朝上，横轴正向向右。页面坐标系与过去说的逻辑坐标系对应，是一个虚拟的坐标系，其原点始终位于页面的左上角，纵轴正向朝下，横轴正向向右。设备坐标系用在具体的输出设备如屏幕中，屏幕坐标系的原点在屏幕的左上角，纵轴正向朝下，横轴正向向右。当页面坐标系和设备坐标系的单位都是像素时，它们相同。

3.6.2　几何变换

在 GDI+中实现几何变换很方便。因为 Graphics 类已经提供了进行各种变换的方法。例如，用 TranslateTransform 方法指定横向上和纵向上的位移量，进行平移变换；用 RotateTransform 方法指定旋转角度，进行旋转变换；用 ScaleTransform 方法指定按比例缩小或放大的比例，进行比例变换。

下面这个例子演示在全局状态下的各种几何变换方法和效果。首先建立项目，在窗体中放置图片框 1 个，命令按钮 6 个，如图 3-9 所示。各控件的属性设置如表 3-9 所示。

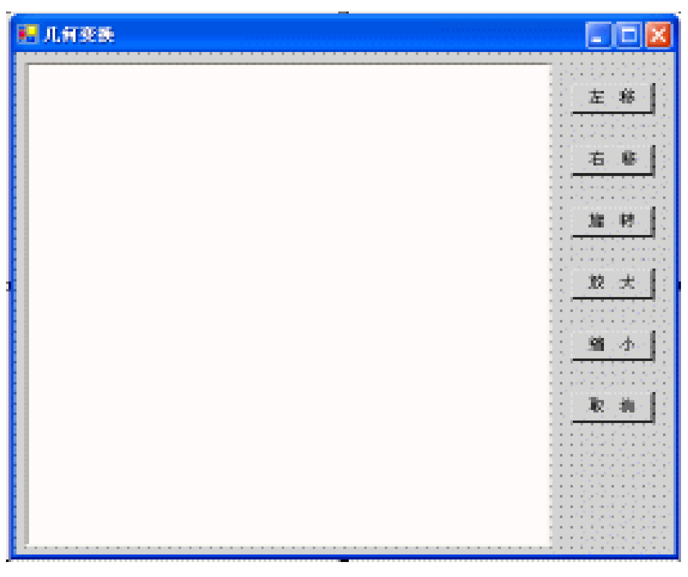

图 3-9　程序界面

表 3-9　控件属性设置

控 件 类 型	名　　称	标　　注
PictureBox	PictureBox1	
Button	Button1	左　　移
	Button2	旋　　转
	Button3	右　　移
	Button4	放　　大
	Button5	缩　　小
	Button6	取　　消

在 Form1 类中添加下面的代码：

【VB.NET】

```
Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    '左移
    g = PictureBox1.CreateGraphics()
    g.Clear(Color.White)
    g.TranslateTransform(-50, 0)
    Draw(g)
End Sub

Private Sub PictureBox1_Paint(ByVal sender As Object, _
         ByVal e As Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint
    g = e.Graphics
    Draw(g)
End Sub

Private Sub Draw(ByVal g As Graphics)
    '绘图
    g.DrawLine(Pens.Black, 10, 10, 100, 100)
    g.DrawEllipse(Pens.Black, 50, 50, 200, 100)
    g.DrawArc(Pens.Black, 100, 10, 100, 100, 20, 160)
    g.FillRectangle(Brushes.Green, 50, 200, 150, 100)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles Button2.Click
    '旋转
    g = PictureBox1.CreateGraphics()
    g.Clear(Color.White)
```

```
        g.RotateTransform(-30)
        Draw(g)
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button4.Click
        '放大
        g = PictureBox1.CreateGraphics()
        g.Clear(Color.White)
        g.ScaleTransform(1.5, 1.5)
        Draw(g)
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button5.Click
        '缩小
        g = PictureBox1.CreateGraphics()
        g.Clear(Color.White)
        g.ScaleTransform(0.8, 0.8)
        Draw(g)
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, _
                 ByVal e As System.EventArgs) Handles Button3.Click
        '右移
        Dim g As Graphics = PictureBox1.CreateGraphics()
        g.Clear(Color.White)
        g.TranslateTransform(50, 0)
        Draw(g)
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles Button6.Click
        '取消运行
        End
    End Sub

    Module Module1
        Public g As Graphics
    End Module
```

【VC#.NET】

```
private System.Windows.Forms.PictureBox pictureBox1;
internal System.Windows.Forms.Button Button6;
internal System.Windows.Forms.Button Button5;
internal System.Windows.Forms.Button Button4;
internal System.Windows.Forms.Button Button3;
internal System.Windows.Forms.Button Button2;
internal System.Windows.Forms.Button Button1;
private Graphics g;

private void Button1_Click(object sender,EventArgs e)
{
    //左移
    g = pictureBox1.CreateGraphics();
    g.Clear(Color.White);
    g.TranslateTransform(-50, 0);
    Draw(g);
}

private void PictureBox1_Paint(object sender, EventArgs e)
{
    g = pictureBox1.CreateGraphics();
    Draw(g);
}

private void Draw(Graphics g)
{
    //绘图
    g.DrawLine(Pens.Black, 10, 10, 100, 100);
    g.DrawEllipse(Pens.Black, 50, 50, 200, 100);
    g.DrawArc(Pens.Black, 100, 10, 100, 100, 20, 160);
    g.FillRectangle(Brushes.Green, 50, 200, 150, 100);
}

private void Button3_Click(object sender, EventArgs e)
{
    //右移
    Graphics g= pictureBox1.CreateGraphics();
    g.Clear(Color.White);
```

```
        g.TranslateTransform(50, 0);
        Draw(g);
    }

    private void Button2_Click_1(object sender, System.EventArgs e)
    {
        //旋转
        g = pictureBox1.CreateGraphics();
        g.Clear(Color.White);
        g.RotateTransform(-30);
        Draw(g);
    }

    private void Button4_Click_1(object sender, System.EventArgs e)
    {
        //放大
        g = pictureBox1.CreateGraphics();
        g.Clear(Color.White);
        g.ScaleTransform(1.2f, 1.2f);
        Draw(g);
    }

    private void Button5_Click_1(object sender, System.EventArgs e)
    {
        //缩小
        g = pictureBox1.CreateGraphics();
        g.Clear(Color.White);
        g.ScaleTransform(0.8f, 0.8f);
        Draw(g);
    }

    private void Button6_Click_1(object sender, System.EventArgs e)
    {
        //取消运行
        this.Dispose();
    }
```

程序运行结果如图 3-10 和图 3-11 所示。

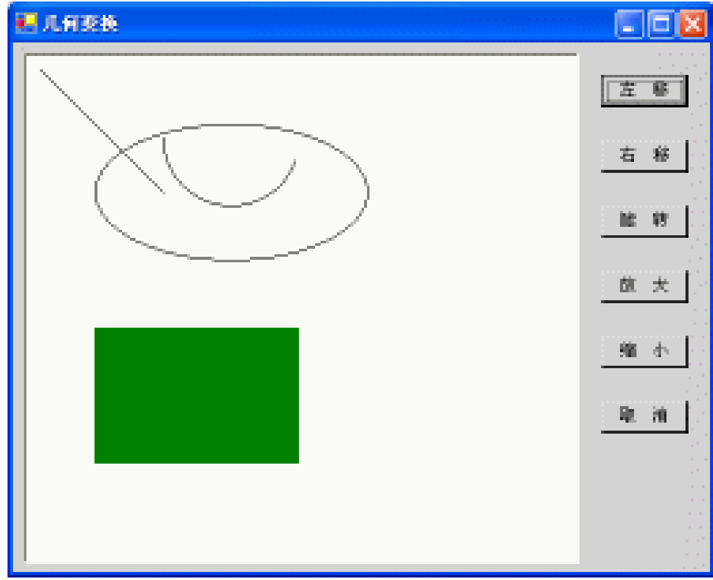

图 3-10　几何变换效果一

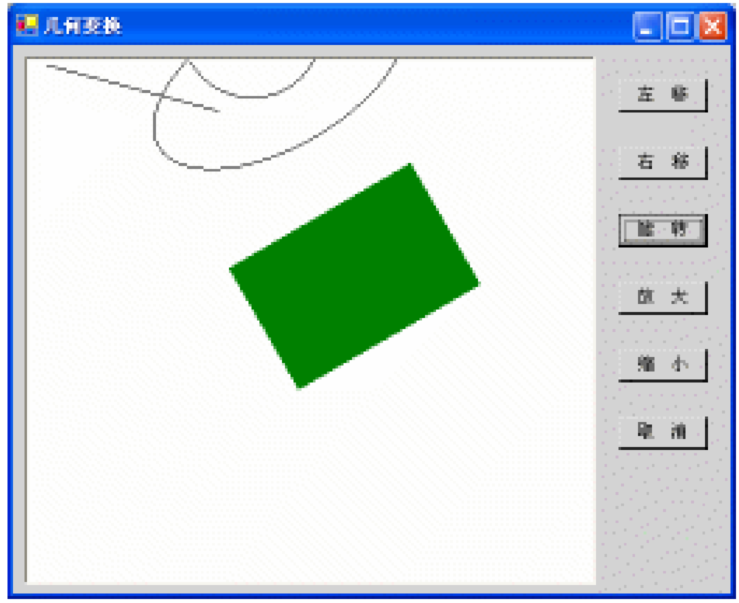

图 3-11　几何变换效果二

3.6.3　全局坐标与局部坐标

前面的各种变换都是在全局坐标系下面进行的，其变换对于绘图表面上的每个图元都会产生影响。全局变换常用于设定通用坐标系，比如重新指定原点位置和坐标轴正向等。由于它对全部图元都产生影响，所以也可以用于实现平移和缩小放大操作。

下面的例子首先将原点移到屏幕中心，并使 Y 轴正向朝上。然后在坐标系下进行一些几何变换和图形绘制工作。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    '本程序建立 X 轴正向向右，Y 轴正向向上，原点在屏幕中心的全局坐标系
    '使 Y 轴朝上，需要做相对于 X 轴的镜像，变换矩阵为［1,0,0,−1,0,0］
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    g.DrawEllipse(Pens.Black, −100, −100, 200, 200)

    '相对于 X 轴进行镜像变换
    Dim mat As New Matrix(1, 0, 0, −1, 0, 0)
    g.Transform = mat
    Dim rect As Rectangle = Me.ClientRectangle
    Dim w As Integer = rect.Width
    Dim h As Integer = rect.Height
    '将原点移到屏幕中心
    g.TranslateTransform(w / 2, −h / 2)

    '以原点为中心，绘一个半径为 100 的圆
    g.DrawEllipse(Pens.Red, −100, −100, 200, 200)

    g.TranslateTransform(100, 100)
    g.DrawEllipse(Pens.Green, −100, −100, 200, 200)

    g.ScaleTransform(2, 2)
    g.DrawEllipse(Pens.Blue, −100, −100, 200, 200)

End Sub

Private Sub Form1_MouseMove(ByVal sender As Object, _
   ByVal e As Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
    '在文本框中显示当前点的坐标。屏幕坐标被转换为工作区坐标
    Dim g As Graphics = CreateGraphics()
    Dim rect As Rectangle = Me.ClientRectangle
    Dim w As Integer = rect.Width
    Dim h As Integer = rect.Height
    TextBox1.Text = " "
    TextBox1.Text = Str(e.X − w / 2) & "    " & Str(− (e.Y − h / 2))
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
```

```
{
    //本程序建立 X 轴正向向右，Y 轴正向向上，原点在屏幕中心的全局坐标系
    //使 Y 轴朝上，需要做相对于 X 轴的镜像，变换矩阵为［1,0,0, -1,0,0］
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    g.DrawEllipse(Pens.Black, -100, -100, 200, 200);

    //相对于 X 轴进行镜像变换
    Matrix mat=new Matrix(1, 0, 0, -1, 0, 0);
    g.Transform = mat;
    Rectangle rect= this.ClientRectangle;
    int w = rect.Width;
    int h= rect.Height;
    //将原点移到屏幕中心
    g.TranslateTransform(w / 2, -h / 2);

    //以原点为中心，绘一个半径为 100 的圆
    g.DrawEllipse(Pens.Red, -100, -100, 200, 200);

    g.TranslateTransform(100, 100);
    g.DrawEllipse(Pens.Green, -100, -100, 200, 200);

    g.ScaleTransform(2, 2);
    g.DrawEllipse(Pens.Blue, -100, -100, 200, 200);
}

private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    //在文本框中显示当前点的坐标。屏幕坐标被转换为工作区坐标
    Graphics g = CreateGraphics();
    Rectangle rect= this.ClientRectangle;
    int w = rect.Width;
    int h= rect.Height;
    textBox1.Text = " ";
    textBox1.Text =e.X - w / 2 + "    " + (- (e.Y - h / 2));
}
```

程序运行结果如图 3-12 所示。文本框中显示了通用坐标系下鼠标的当前位置。

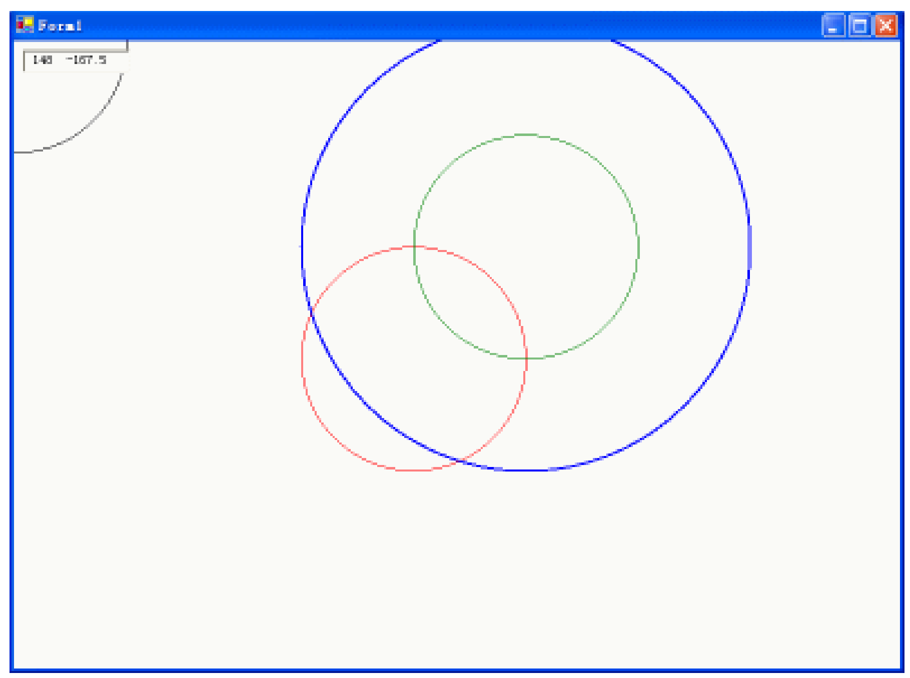

图 3-12　全局变换

现在假设只对某个或某些图形进行变换，而其他图形元素不变，就需要用到局部坐标系。进行局部变换，需要首先创建一个 GraphicsPath 对象，添加图元到该对象以后，创建一个 Matrix 对象，确定变换矩阵，然后将这个 Matrix 对象作为参数传递给 GraphicsPath 对象的 Transform 方法。这样，路径中的所有图元都会做相应的变换，而其他图元不会变化。

下面的程序首先进行全局变换，建立原点在屏幕中心，Y 轴正向朝上的通用坐标系；然后创建一条路径，在路径中添加一个圆，再在该路径所在的局部坐标系中进行平移变换和旋转变换。路径中的圆在变换后的另一个位置显示。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    '把客户区设置为白色
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    '定义相对于 X 轴镜像的转换矩阵，使得 Y 轴朝上
    Dim mat As New Matrix(1, 0, 0, -1, 0, 0)
    g.Transform = mat
    '将坐标原点移动到屏幕中心
    Dim rect As Rectangle = Me.ClientRectangle
    Dim w As Integer = rect.Width
    Dim h As Integer = rect.Height
    g.TranslateTransform(w / 2, -h / 2)

    '在全局坐标下绘制椭圆
```

```
        g.DrawEllipse(Pens.Red, -100, -100, 200, 200)
        g.FillRectangle(Brushes.Black, 100 - 2, 0 + 2, 4, 4)
        g.DrawRectangle(Pens.Blue, 0 - 2, -100 + 2, 4, 4)
        g.DrawRectangle(Pens.Blue, -100 - 2, 0 + 2, 4, 4)
        g.DrawRectangle(Pens.Blue, 0 - 2, 100 + 2, 4, 4)

        '创建一个椭圆，然后在局部坐标系中进行几何变换
        Dim gp As New GraphicsPath()
        gp.AddEllipse(-100, -100, 200, 200)
        Dim mat2 As New Matrix()
        '平移变换
        mat2.Translate(150, 150)
        '旋转变换，缺省时按顺时针方向旋转
        mat2.Rotate(30)
        gp.Transform(mat2)
        g.DrawPath(Pens.Blue, gp)
        Dim p() As PointF = gp.PathPoints()
        g.FillRectangle(Brushes.Black, p(0).X - 2, p(0).Y + 2, 4, 4)
        g.DrawRectangle(Pens.Blue, p(3).X - 2, p(3).Y + 2, 4, 4)
        g.DrawRectangle(Pens.Blue, p(6).X - 2, p(6).Y + 2, 4, 4)
        g.DrawRectangle(Pens.Blue, p(9).X - 2, p(9).Y + 2, 4, 4)
        gp.Dispose()
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        //把客户区设置为白色
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        //定义相对于 X 轴镜像的转换矩阵，使得 Y 轴朝上
        Matrix mat=new Matrix(1, 0, 0, -1, 0, 0);
        g.Transform = mat;
        //将坐标原点移动到屏幕中心
        Rectangle rect= this.ClientRectangle;
        int    w = rect.Width;
        int h= rect.Height;
        g.TranslateTransform(w / 2, -h / 2);

        //在全局坐标下绘制椭圆
        g.DrawEllipse(Pens.Red, -100, -100, 200, 200);
```

```
        g.FillRectangle(Brushes.Black, 100 - 2, 0 + 2, 4, 4);
        g.DrawRectangle(Pens.Blue, 0 - 2, -100 + 2, 4, 4);
        g.DrawRectangle(Pens.Blue, -100 - 2, 0 + 2, 4, 4);
        g.DrawRectangle(Pens.Blue, 0 - 2, 100 + 2, 4, 4);

        //创建一个椭圆，然后在局部坐标系中进行几何变换
        GraphicsPath gp=new GraphicsPath();
        gp.AddEllipse(-100, -100, 200, 200);
        Matrix mat2=new Matrix();
        //平移变换
        mat2.Translate(150, 150);
        //旋转变换，缺省时按顺时针方向旋转
        mat2.Rotate(30);
        gp.Transform(mat2);
        g.DrawPath(Pens.Blue, gp);
        PointF[] p= gp.PathPoints;
        g.FillRectangle(Brushes.Black, p[0].X - 2, p[0].Y + 2, 4, 4);
        g.DrawRectangle(Pens.Blue, p[3].X - 2, p[3].Y + 2, 4, 4);
        g.DrawRectangle(Pens.Blue, p[6].X - 2, p[6].Y + 2, 4, 4);
        g.DrawRectangle(Pens.Blue, p[9].X - 2, p[9].Y + 2, 4, 4);
        gp.Dispose();
    }
```

程序运行结果如图 3-13 所示。图中还显示了圆上的 4 个关键点，可见，变换以后，关键点也随着移动了位置。

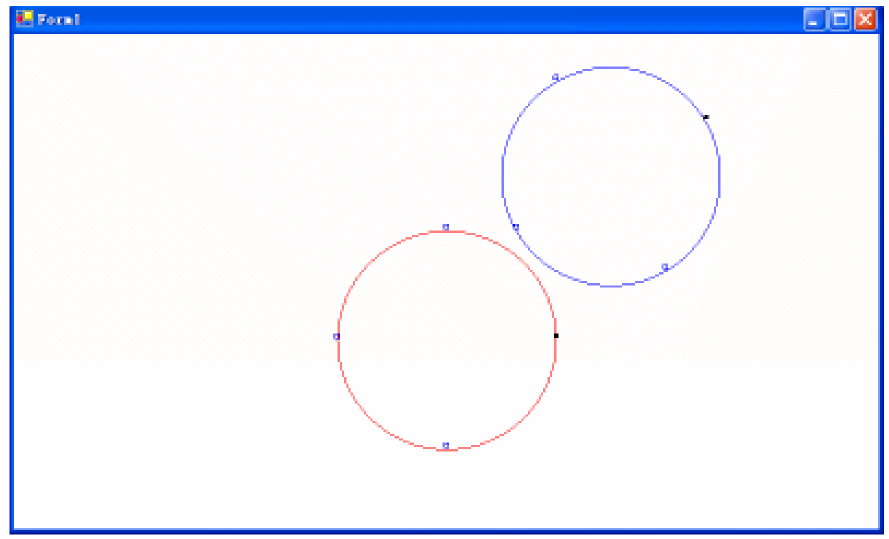

图 3-13　局部变换

3.7 Alpha 混合

前面介绍 Color 类时曾经讲到了 FromArgb 方法。该方法一个最常用的重载为：

```
Overloads Public Shared Function FromArgb(Integer,Integer, _
    Integer,Integer) As Color
```

它有 4 个参数，是当前要设置的颜色的 ARGB 分量（alpha、红色、绿色和蓝色）值，均为 8 位，共 32 位。其中的 alpha 值表示颜色的透明度，即前景色与背景色的混合程度，取值范围为 0 到 255，取 0 时表示完全透明，取 255 时完全不透明。

对于给定的 alpha 值，前景色、背景色和当前色之间具有下面的关系式：

当前色＝前景色 ×alpha / 255 + 背景色×(255 – alpha) / 255

下面的程序首先创建一个蓝紫色的矩形作为背景，然后创建两个位图，每个位图中有两个矩形。第 1 个位图中间的两个矩形具有透明效果，第 2 个位图中间的两个矩形则没有透明效果。有无透明效果是通过将该位图 Graphics 对象的 CompositingMode 属性设置为 CompositingMode.SourceCopy 枚举值来实现的。设置为该枚举值以后，前景色将覆盖背景色，而不是混合。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    '创建一个填充矩形
    Dim brush As New SolidBrush(Color.BlueViolet)
    g.FillRectangle(brush, 180, 70, 200, 150)
    '创建一个位图，其中的两个矩形之间有透明效果
    Dim bm1 As New Bitmap(200, 100)
    Dim bg1 As Graphics = Graphics.FromImage(bm1)
    Dim redBrush As New SolidBrush(Color.FromArgb(210, 255, 0, 0))
    Dim greenBrush As New SolidBrush(Color.FromArgb(210, 0, 255, 0))
    bg1.FillRectangle(redBrush, 0, 0, 150, 70)
    bg1.FillRectangle(greenBrush, 30, 30, 150, 70)
    g.DrawImage(bm1, 100, 100)
    '创建第 2 个位图，其中两个矩形之间没有透明效果
    Dim bm2 As New Bitmap(200, 100)
    Dim bg2 As Graphics = Graphics.FromImage(bm2)
    bg2.CompositingMode = CompositingMode.SourceCopy
    bg2.FillRectangle(redBrush, 0, 0, 150, 70)
    bg2.FillRectangle(greenBrush, 30, 30, 150, 70)
    g.CompositingQuality = CompositingQuality.GammaCorrected
    g.DrawImage(bm2, 300, 100)
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    //创建一个填充矩形
    SolidBrush brush=new SolidBrush(Color.BlueViolet);
    g.FillRectangle(brush, 180, 70, 200, 150);
    //创建一个位图，其中的两个矩形之间有透明效果
    Bitmap bm1=new Bitmap(200, 100);
    Graphics bg1= Graphics.FromImage(bm1);
    SolidBrush redBrush=new SolidBrush(Color.FromArgb(210, 255, 0, 0));
    SolidBrush greenBrush=new SolidBrush(Color.FromArgb(210, 0, 255, 0));
    bg1.FillRectangle(redBrush, 0, 0, 150, 70);
    bg1.FillRectangle(greenBrush, 30, 30, 150, 70);
    g.DrawImage(bm1, 100, 100);
    //创建第 2 个位图，其中两个矩形之间没有透明效果
    Bitmap bm2=new Bitmap(200, 100);
    Graphics bg2= Graphics.FromImage(bm2);
    bg2.CompositingMode = CompositingMode.SourceCopy;
    bg2.FillRectangle(redBrush, 0, 0, 150, 70);
    bg2.FillRectangle(greenBrush, 30, 30, 150, 70);
    g.CompositingQuality = CompositingQuality.GammaCorrected;
    g.DrawImage(bm2, 300, 100);
}
```

程序运行结果如图 3-14 所示。

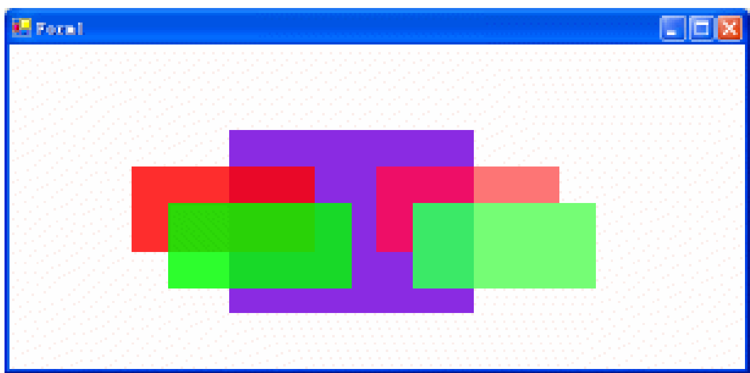

图 3-14　Alpha 混合示例

3.8　反走样

走样是指绘制线条、文本时，出现的锯齿现象。反走样则是通过一定的算法来消除这些

锯齿，或者说使这些锯齿显得不那么明显。GDI+中，将 Graphcis 对象的 SmoothingMode 属性设置为 AntiAlias 可以改善图形的显示效果。

下面的例子比较了反走样前后图形和文本的显示效果。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    '放大 8 倍
    g.ScaleTransform(8, 8)
    '绘制没有反走样效果的图形和文字
    Draw(g)
    '设置反走样
    g.SmoothingMode = SmoothingMode.AntiAlias
    '向右平移 40，绘制有反走样效果的图形和文字
    g.TranslateTransform(40, 0)
    Draw(g)
End Sub

Private Sub Draw(ByVal g As Graphics)
    '绘一条直线段、一个椭圆和一字符串
    g.DrawLine(Pens.Gray, 10, 10, 40, 20)
    g.DrawEllipse(Pens.Gray, 20, 20, 30, 10)
    Dim s As String = "反走样测试"
    Dim font As New Font("宋体", 5)
    Dim brush As New SolidBrush(Color.Gray)
    g.DrawString(s, font, brush, 10, 40)
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics   g= e.Graphics;
    //放大 8 倍
    g.ScaleTransform(8, 8);
    //绘制没有反走样效果的图形和文字
    Draw(g);
    //设置反走样
    g.SmoothingMode = SmoothingMode.AntiAlias;
    //向右平移 40，绘制有反走样效果的图形和文字
    g.TranslateTransform(40, 0);
    Draw(g);
}
```

```
private void Draw(Graphics g)
{
    //绘一条直线段、一个椭圆和一字符串
    g.DrawLine(Pens.Gray, 10, 10, 40, 20);
    g.DrawEllipse(Pens.Gray, 20, 20, 30, 10);
    string s= "反走样测试";
    Font font=new Font("宋体", 5);
    SolidBrush brush=new SolidBrush(Color.Gray);
    g.DrawString(s, font, brush, 10, 40);
}
```

程序运行结果如图 3-15 所示。

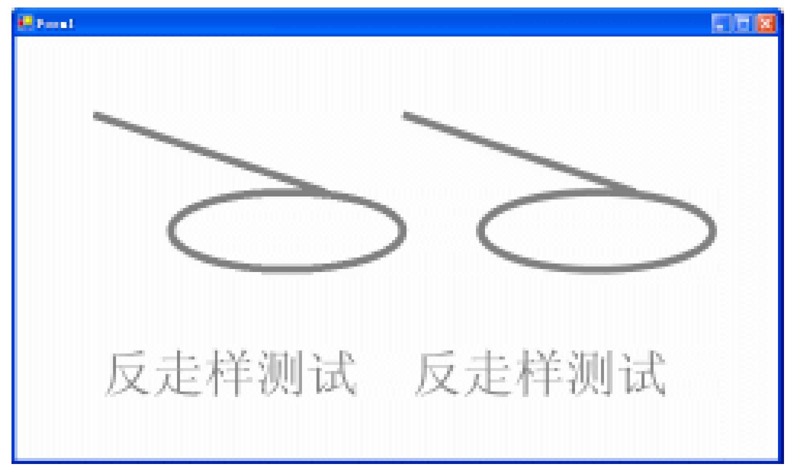

图 3-15　反走样效果测试

3.9　用 API 函数绘图

3.9.1　为什么还要使用 API 函数

对于 VB 6.0 来说，很多任务必须借助 API 函数才能实现。而现在，大部分 API 函数具备的功能都已经纳入到.NET 框架中来了，使用 API 函数本身也存在一些缺陷，所以.NET 并不提倡使用 API 函数。但在某些特定的情况下，使用 API 函数仍然要方便很多。比如 SetROP2 函数。它的功能是设置绘图模式，即将画笔的颜色与背景色进行计算以后，用得到的颜色绘图。

在 VB 6.0 中，通过设置窗体或图片框等的 DrawMode 属性，可以设置绘图模式。但 VB.NET 和 VC#.NET 中却找不到对应的属性或方法。如果从头开始重写一个 SetROP2 函数，比直接调用 API 函数要麻烦得多。

3.9.2　API 函数的声明和调用

声明 API 函数有两种方式。一种是使用 Declare 语句，另一种是使用 DLLImport。下面用 Declare 语句声明 SetROP2 函数。

【VB.NET】

```
Private Declare Auto Function SetROP2 Lib "gdi32.dll" ( _
        ByVal hdc As IntPtr, _
        ByVal nDrawMode As Integer) As Boolean
```

下面用 DLLImport 声明 SetROP2 函数。

【VB.NET】

```
<DllImport("gdi32.dll")> _
 Private Shared Function SetROP2(ByVal hdc As IntPtr, _
        ByVal nDrawMode As Integer) As Boolean
 End Function
```

【VC#.NET】

```
[DllImport("gdi32.dll")]
private static extern bool SetROP2(IntPtr hdc,int nDrawMode);
```

声明 API 函数以后，可以像调用其他函数或方法一样调用它们。

3.9.3　用 API 函数绘图示例

下面的例子声明和调用 API 函数 SetROP2，Rectangle，CreatePen，SelectObject 和 DeleteObject，用它们绘制矩形。

【VB.NET】

```
<StructLayout(LayoutKind.Sequential)> Public Structure LPPOINT
      Public x As Long
      Public y As Long
End Structure

<DllImport("gdi32.dll")> _
 Private Shared Function SetROP2(ByVal hdc As IntPtr, _
        ByVal nDrawMode As Integer) As Boolean
End Function

<DllImport("gdi32.dll")> _
Private Shared Function Rectangle _
      (ByVal hdc As IntPtr, ByVal X1 As Integer, ByVal Y1 As Integer, _
      ByVal X2 As Integer, ByVal Y2 As Integer) As Boolean
End Function
```

```
<DllImport("gdi32.dll")> _
Private Shared Function CreatePen(ByVal nPenStyle As Integer, _
    ByVal nWidth As Integer, ByVal crColor As Long) As Long
End Function

<DllImport("gdi32.dll")> _
Private Shared Function SelectObject(ByVal hdc As IntPtr, _
    ByVal hObject As Long) As Long
End Function

<DllImport("gdi32.dll")> _
Private Shared Function DeleteObject(ByVal hObject As Long) As Long
End Function

'绘直线段
Public Sub Draw(ByVal aGraphics As Graphics, ByVal aColor As Long)
    Dim hdc As IntPtr
    hdc = aGraphics.GetHdc
    SetROP2(hdc, 10)
    Dim p As Long
    Dim oldP As Long
    p = CreatePen(0, 1, aColor)
    oldP = SelectObject(hdc, p)
    Rectangle(hdc, 20, 150, 170, 250)
    SelectObject(hdc, oldP)
    DeleteObject(p)
    aGraphics.ReleaseHdc(hdc)
End Sub

Public Sub Form1_Paint(ByVal sender As Object, _
       ByVal e As Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    '用 GDI+函数绘一个蓝色矩形
    g.DrawRectangle(Pens.Blue, 20, 20, 150, 100)
    '用 API 函数绘一个红色矩形
    Draw(g, RGB(255, 0, 0))
End Sub
```

【VC#.NET】

```
[StructLayout(LayoutKind.Sequential)]
public struct LPPOINT
{
    public long x;
    public long y;
}

[DllImport("gdi32.dll")]
private static extern bool SetROP2(IntPtr hdc,int nDrawMode);

[DllImport("gdi32.dll")]
private static extern bool Rectangle
    (IntPtr hdc,int X1,    int Y1,int X2, int Y2);

[DllImport("gdi32.dll")]
private static extern long CreatePen(int nPenStyle,
    int nWidth, long crColor);

[DllImport("gdi32.dll")]
private static extern long SelectObject(IntPtr hdc,
    long hObject);

[DllImport("gdi32.dll")]
private static extern long DeleteObject(long hObject);

//绘直线段
public void Draw(Graphics g, long aColor)
{
    IntPtr hdc;
    hdc =g.GetHdc();
    SetROP2(hdc, 10);
    long p;
    long oldP;
    p = CreatePen(0, 1, aColor);
    oldP = SelectObject(hdc, p);
    Rectangle(hdc, 20, 150, 170, 250);
    SelectObject(hdc, oldP);
    DeleteObject(p);
    g.ReleaseHdc(hdc);
```

```
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.FillRectangle(Brushes.White, this.ClientRectangle);
            //用 GDI+函数绘一个蓝色矩形
            g.DrawRectangle(Pens.Blue, 20, 20, 150, 100);
            //用 API 函数绘一个红色矩形
            Draw(g,255);
        }
```

程序运行结果如图 3-16 所示。图中蓝色矩形是用 GDI+函数绘的，红色矩形是用 API 函数绘的。

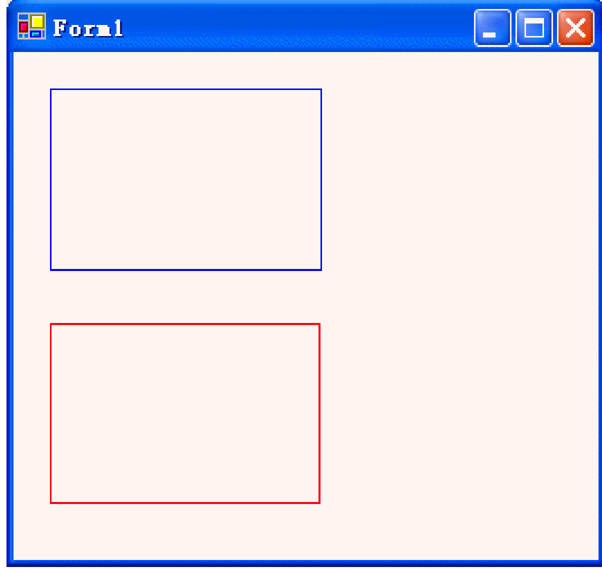

图 3-16　用 API 函数绘图

第 4 章　设计 CAD 小系统的基本思路和技术

为使您对后面将要介绍的交互式 CAD 小系统实例所采用的组织思路和基本技术有一个总体认识和把握，而编写本章。本章从类的设计、坐标系统、交互技术、数据存储等方面进行介绍，是导读，也是总结。

4.1　相关类的设计

4.1.1　对象和类

对象是存在于现实世界中的事物或关系，而类是描述对象的代码集。比如树是对象，Tree 类是描述树这个对象的代码集。

形成类的前提是有可描述的对象。而对象常常隐藏在纷繁复杂的表象中，有经验、有洞察力的人才能发现它，并把它提取出来。对象本身是客观存在的，但对象的提取方法不是惟一的，它与要处理的问题有关，与解决问题的方法有关。

对象被提取出来以后，它不是孤立的，它要与其他对象发生联系。如何处理对象之间的关系，是对象的组织问题。如果把单个的对象比做黑白世界里的棋子，那么对象的组织就好象是布一个局，可以有很多变化。围棋中有定势，对象组织也有定势。人们在长期的编程实践中，发现解决某类问题时可以采用同样的模式来提取和组织类，这就是设计模式。本书第 12 章专门结合 CAD 编程进行讨论。

对象的提取完成以后，用计算机语言描写出来，就是类。而对象的组织相应地要用有组织的类来描述。

所以，进行交互式 CAD 系统设计以前，需要首先提取出必要的对象，包括基本的图元对象和定义交互操作的对象，然后用语言来描写它们。

4.1.2　基本图元类设计

为了描述各种基本的图形元素，必须进行图元类的设计。作为图元来讲，它们有很多共同点，比如都有颜色、样式（线型或文本格式）和大小，交互绘制时都要进行拾取、平移、旋转、镜像、缩放和绘制等。所以有必要创建一个图元基类 CGElement。把该基类定义为抽象类，必须继承；将共有的方法设置为抽象方法，必须在派生类中重写。本书最后提供的 CAD 小系统中包含有直线段、矩形、圆、圆弧和文本等 5 种图元，对应地创建了 CLine 类、CRectangle 类、CCircle 类、CArc 类和 CText 类。这些类都从 CGElement 类继承，其 UML 图如图 4-1 所示。

第 5 章将详细介绍基本图元类的设计。

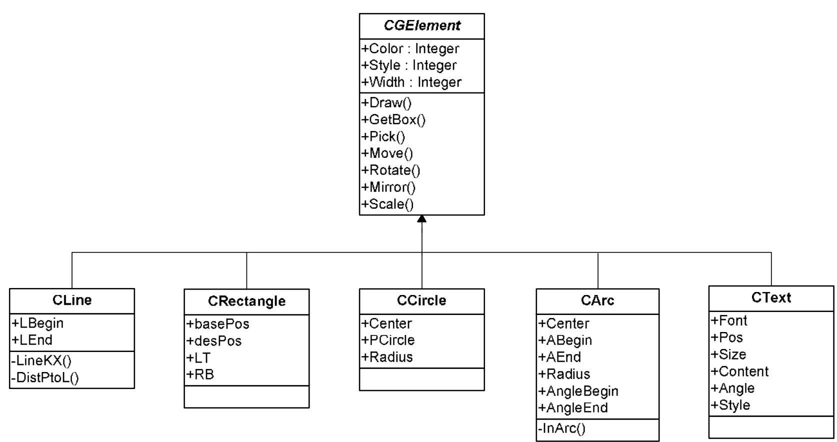

图 4-1　基本图元类的 UML 图

4.1.3　交互绘图类设计

基本图元类对图元的基本特征和行为进行了描述，但仅仅依靠它们还不够，因为在交互式的 CAD 系统中，确定图元形态的数据（如直线段的端点坐标等）不是预先给定的，而是在绘制的过程中通过命令窗口或鼠标即时输入的。所以，还需要对交互绘制图元时的整个过程进行描述。这个过程是一个动态的过程。例如，绘制直线段时，首先在绘图区单击鼠标左键，确定了直线段的起点，然后移动鼠标，移动时要动态地显示直线段的当前状态；再次单击鼠标左键时，把直线段的终点确定下来，完成绘制。有了这样一个描述以后，绘图工作就变得有趣了，也更方便了。

为了描述交互绘图的过程，必须创建交互绘图类。用鼠标绘图时，总是涉及到按键和移动等操作。所以，有必要创建一个接口或抽象的基类来定义这些操作行为，为具体的操作提供模板。本书中创建了一个 ICommand 接口，它有 3 个方法：LbuttonDown, MouseMove 和 RButtonDown，分别定义鼠标左键按下、移动和右键按下时的绘图行为。然后有 9 个交互绘图类实现了 ICommand 接口，即 CCreateLine 类、CCreateRect 类、CCreateCircle 类、CCreateArc 类、CCreateText 类、CMove 类、CRotate 类、CMirror 类、CSelect 类，分别实现直线段、矩形、圆、圆弧、文本的创建，对选定图元的平移变换、旋转变换和镜像变换，以及选择图元。

如图 4-2 所示，各类与 ICommand 接口之间是接口关系。也可以把 ICommand 接口用一个抽象基类 CCommand 替换，然后，后面各类从它那里继承。两种方法在性能上并没有明显的区别，但这里主要强调的是行为的相似性，所以定义为接口。关于交互绘图类的组织，第 12 章还有设计模式方面的讨论，可以参阅。

交互绘图类的设计在第 6 章进行具体介绍。

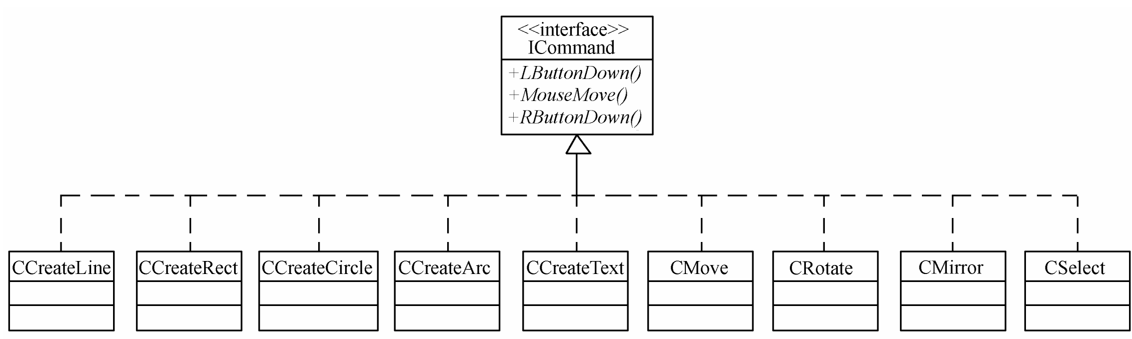

图 4-2 交互绘图类的 UML 图

4.1.4 类的交互

其实，类的交互问题已经比较清楚了。当客户端（菜单项或命令按钮）调用交互绘图类进行绘图时，交互绘图类必然要创建基本图元类的实例并调用它的方法。描述一个一般的操作，如创建一条直线段，选中它，然后平移它，其步骤大致如下。

（1）首先发出创建直线段的指令，调用 CCreateLine 类进行具体实施。第 1 次单击鼠标左键确定直线段的起点以后,移动鼠标时要绘制起点到鼠标光标当前位置的直线段(橡皮线)，绘制橡皮线就需要创建 CLine 类的实例，然后调用该实例的 Draw 方法完成绘制。

（2）选择直线段时使用 CSelect 类，需要计算当前鼠标光标点到直线段的距离，判断直线段是否被拾取。此时需要从保存图元的集合类中通过索引找到该直线段对象，然后调用它的 Pick 方法进行判断。如果被拾取，则用 Draw 方法，用某种其他的颜色或线型重画直线段。

（3）选中图元以后，可以进行平移操作。指令发出以后，用 CMove 类完成。此时要调用被选中的直线段对象的 Move 方法移动直线段并重新显示。

4.2 坐标系统

坐标系统是 CAD 绘图时的参照，所有图元的数据大小都与坐标原点和坐标轴方向有关。所以，绘图以前，必须首先设置坐标系统。

第 3 章比较详细地介绍了.NET 中坐标系统的类型和设置方法，需要明确的有下面几点。

（1）.NET 中有通用坐标系、页面坐标系和设备坐标系等 3 种坐标系。当绘图表面的度量单位为像素时，页面坐标与设备坐标在数值上相同。

（2）页面坐标系中坐标原点在左上角，纵轴向下为正。通过全局变换可以将页面坐标系转换为通用坐标系。比如让坐标原点设置在屏幕中心，纵轴坐标正向向上。

（3）图形元素始终绘制在设备坐标系中，而计算在通用坐标系中完成。

下面，首先进行全局变换，设置坐标原点在屏幕中心，纵轴向上为正的通用坐标系。在 Form1 类中添加下面的代码，载入窗体时就进行变换。

【VB.NET】

```
Private Sub Form1_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
```

```
        Dim g As Graphics = Me.CreateGraphics()
        g.FillRectangle(Brushes.White, Me.ClientRectangle)
        g.PageUnit = GraphicsUnit.Pixel

        '相对于 X 轴进行镜像变换
        Dim mat As New Matrix(1, 0, 0, -1, 0, 0)
        g.Transform = mat
        Dim rect As Rectangle = Me.ClientRectangle
        viewDX = rect.Width / 2
        viewDY = rect.Height / 2
        g.TranslateTransform(viewDX, viewDY)
    End Sub
```

【VC#.NET】

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
        Graphics g= this.CreateGraphics();
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        g.PageUnit = GraphicsUnit.Pixel;

        //相对于 X 轴进行镜像变换
        Matrix mat=new Matrix(1, 0, 0, -1, 0, 0);
        g.Transform = mat;
        Rectangle rect= this.ClientRectangle;
        float viewDX = rect.Width / 2;
        float viewDY = rect.Height / 2;
        g.TranslateTransform(viewDX, viewDY);
    }
```

因为坐标系统要随着计算状态和绘制状态的转换而改变，所以需要首先编写坐标系统的转换函数。下面是没有考虑缩放因素的转换函数。

【VB.NET】

```
    '页面坐标转换为通用坐标
    Public Function PagetoWorld(ByVal ep As PointF) As PointF
        Dim p As New PointF()
        With ep
            p.X = (.X - viewDX)
            p.Y = - (.Y - viewDY)
        End With
        Return p
    End Function
```

```
'通用坐标转换为页面坐标
Public Function WorldtoPage(ByVal pp As PointF) As PointF
    Dim p As New PointF()
    With pp
        p.X = .X + viewDX
        p.Y = -.Y + viewDY
    End With
    Return p
End Function
```

【VC#.NET】

```
//页面坐标转换为通用坐标
public PointF PagetoWorld(PointF ep)
{
    Form1 f=new Form1();
    Rectangle rect=f.ClientRectangle;
    float viewDX=rect.Width/2;
    float viewDY=rect.Height/2;
    PointF p=new PointF();
    p.X = ep.X - viewDX;
    p.Y = viewDY-ep.Y;
    return p;
}

//通用坐标转换为页面坐标
public PointF WorldtoPage(PointF pp)
{
    Form1 f=new Form1();
    Rectangle rect=f.ClientRectangle;
    float viewDX=rect.Width/2;
    float viewDY=rect.Height/2;
    PointF p=new PointF();
    p.X = pp.X + viewDX;
    p.Y = -pp.Y + viewDY;
    return p;
}
```

4.3 交互技术及其实现

常见的交互绘图技术包括用鼠标绘图、橡皮线绘制、拾取、选择、变换、相交线、定位和界面交互等。本套书的 VB 篇中介绍了一些。下面主要介绍最基础的鼠标绘图和橡皮线绘

制技术，后面各章节还要陆续讲到拾取、选择、变换、相交线和界面交互。

4.3.1　用鼠标绘图

在电脑屏幕上，鼠标的光标就像我们的画笔，通过单击和移动鼠标可以实现交互绘图。用鼠标绘图时，要触发一系列鼠标事件。如按下鼠标键时触发 MouseDown 事件，移动鼠标时触发 MouseMove 事件。.NET 中部分鼠标事件及说明如表 4-1 所示。

表 4-1　鼠标事件及说明

事　　件	说　　明
MouseDown	在控件上按下鼠标按钮时触发此事件
MouseEnter	鼠标光标进入控件时触发此事件
MouseMove	鼠标光标在控件上移动时触发此事件
MouseHover	鼠标光标悬停在控件上方时触发此事件
MouseUp	按下的鼠标按钮被释放时触发此事件
MouseWheel	移动鼠标滑轮时触发此事件
MouseLeave	鼠标光标离开控件时触发此事件

同绘制窗体可以使用 Paint 事件或重写 OnPaint 方法一样，使用表 4-1 中的鼠标事件也有两种方法，一种是直接调用事件，另一种是使用重写的 OnMouseEvent 方法（Event 代表 Down, Up 等，下同）。使用前一种方法会调用基类的 OnMouseEvent 方法，而后一种不会。所以后面均采用后一种方法。

下面的程序在窗体上画一个方框，鼠标在方框内外移动，或者在方框内单击时显示相应的说明文字到方框下面的文本框中。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    Dim pen As New Pen(Color.Blue, 3)
    g.DrawRectangle(pen, 100, 20, 200, 200)
End Sub

Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    TextBox1.Text = " "
    If InBox(e.X, e.Y) Then
        TextBox1.Text = "当前鼠标在方框内移动。"
    Else
        TextBox1.Text = "当前鼠标在方框外移动。"
    End If
End Sub

Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)
    If InBox(e.X, e.Y) Then
```

```
            TextBox1.Text = " "
            If e.Button = MouseButtons.Left Then
                TextBox1.Text = "当前鼠标左键在方框内按下。"
            ElseIf e.Button = MouseButtons.Right Then
                TextBox1.Text = "当前鼠标右键在方框内按下。"
            End If
        End If
    End Sub

    Private Function InBox(ByVal x As Single, ByVal y As Single) As Boolean
        If x > 100 And x < 300 And y > 20 And y < 220 Then
            Return True
        Else
            Return False
        End If
    End Function
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        Pen pen=new Pen(Color.Blue, 3);
        g.DrawRectangle(pen, 100, 20, 200, 200);
    }

    protected override void OnMouseMove(MouseEventArgs e)
    {
        textBox1.Text = " ";
        if (InBox(e.X,e.Y ))
        {
            textBox1.Text = "当前鼠标在方框内移动。";
        }
        else
        {
            textBox1.Text = "当前鼠标在方框外移动。";
        }
    }

    protected override void OnMouseDown(MouseEventArgs e)
    {
        if (InBox(e.X,e.Y ))
```

```
        {
            textBox1.Text = " ";
            if (e.Button == MouseButtons.Left)
            {
                textBox1.Text = "当前鼠标左键在方框内按下。";
            }
            else if (e.Button == MouseButtons.Right)
            {
                textBox1.Text = "当前鼠标右键在方框内按下。";
            }
        }
    }

    private bool InBox(float x, float y)
    {
        if (x > 100 && x < 300 && y > 20 && y < 220)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

程序运行结果如图 4-3 所示。

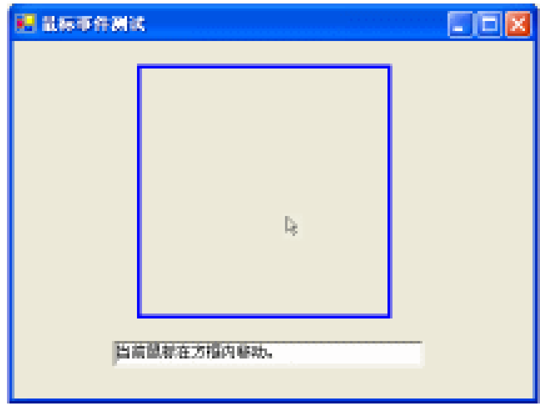

图 4-3　鼠标事件测试

4.3.2　橡皮线

橡皮线技术是交互绘图过程中必不可少的一种技术，它试图实现一种类似于橡皮线的绘

图效果。橡皮线的一端固定以后，牵引另一端，可以随意地改变方向和长短，直到它也固定在某个点上。使用这种技术以后，就可以一种直观的形式描述绘制直线段中定位起点和终点之间的过程。

在计算机上实现橡皮线效果采用了一些“障眼法”。实际上，在线条动态绘制的过程中，屏幕上在不断地擦除原来位置上的图元显示，然后在新位置上绘制图元。此过程中，直线段的起点不变，而终点不断变化。由于计算机的绘制速度很快，肉眼几乎觉察不出擦除图元和显示新图元的过程。

与橡皮线类似的还有橡皮矩形、橡皮圆和橡皮弧等。它们的工作机理与橡皮线大同小异。下面的程序演示了橡皮矩形的实现过程。第 1 次单击鼠标左键时，确定矩形的第 1 个顶点。在移动鼠标光标的过程中，鼠标光标的当前位置作为第 2 个顶点，以橡皮矩形的形式显示矩形。第 2 次单击鼠标左键时，确定矩形中第 1 个顶点的对角顶点，绘制矩形。下面的程序中用到了 API 函数 SetROP2 和 Rectangle，分别设置绘图模式和绘制矩形。

【VB.NET】

```
<DllImport("gdi32.dll")> _
 Private Shared Function SetROP2(ByVal hdc As IntPtr, _
     ByVal nDrawMode As Integer) As Boolean
End Function

<DllImport("gdi32.dll")> _
Private Shared Function Rectangle _
    (ByVal hdc As IntPtr, ByVal X1 As Integer, ByVal Y1 As Integer, _
    ByVal X2 As Integer, ByVal Y2 As Integer) As Boolean
End Function

Private m_StartX, m_StartY As Single
Private m_EndX, m_EndY As Single
Private m_Step As Integer = 0

Protected Overrides Sub OnMouseDown(ByVal e As Windows.Forms.MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics()
    Dim hdc As IntPtr = g.GetHdc()
    m_Step += 1
    SetROP2(hdc, 10)
    If m_Step = 1 Then
        m_StartX = e.X
        m_StartY = e.Y
        m_EndX = e.X
        m_EndY = e.Y
    ElseIf m_Step = 2 Then
        Rectangle(hdc, m_StartX, m_StartY, m_EndX, m_EndY)
```

```
            Rectangle(hdc, m_StartX, m_StartY, e.X, e.Y)
            m_Step = 0
        End If
        g.ReleaseHdc(hdc)
    End Sub

    Protected Overrides Sub OnMouseMove(ByVal e As Windows.Forms.MouseEventArgs)
        Dim g As Graphics = Me.CreateGraphics()
        Dim prex, prey As Single
        If m_Step = 1 Then
            Dim hdc As IntPtr = g.GetHdc()
            prex = m_EndX
            prey = m_EndY
            SetROP2(hdc, 10)
            Rectangle(hdc, m_StartX, m_StartY, prex, prey)
            Rectangle(hdc, m_StartX, m_StartY, e.X, e.Y)
            m_EndX = e.X
            m_EndY = e.Y
            g.ReleaseHdc(hdc)
        End If
    End Sub

    Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
        Dim g As Graphics = e.Graphics
        g.FillRectangle(Brushes.White, Me.ClientRectangle)
    End Sub
```

【VC#.NET】

```
    [DllImport("gdi32.dll")]
    private static extern bool SetROP2(IntPtr hdc,int nDrawMode);

    [DllImport("gdi32.dll")]
    private static extern bool    Rectangle
          (IntPtr hdc, int X1, int Y1,int X2, int Y2);

    private int m_StartX, m_StartY;
    private int m_EndX, m_EndY;
    private int m_Step = 0;

    protected override void OnMouseDown(MouseEventArgs e )
    {
```

```
        Graphics g =this.CreateGraphics();
        IntPtr hdc= g.GetHdc();
        m_Step += 1;
        SetROP2(hdc, 10);
        if (m_Step == 1)
        {
            m_StartX = e.X;
            m_StartY = e.Y;
            m_EndX = e.X;
            m_EndY = e.Y;
        }
        else if (m_Step == 2)
        {
            Rectangle(hdc, m_StartX, m_StartY, m_EndX, m_EndY);
            Rectangle(hdc, m_StartX, m_StartY, e.X, e.Y);
            m_Step = 0;
        }
        g.ReleaseHdc(hdc);
    }

    protected override void OnMouseMove(MouseEventArgs e)
    {
        Graphics g =this.CreateGraphics();
        int prex, prey;
        if (m_Step == 1)
        {
            IntPtr hdc= g.GetHdc();
            prex = m_EndX;
            prey = m_EndY;
            SetROP2(hdc, 10);
            Rectangle(hdc, m_StartX, m_StartY, prex, prey);
            Rectangle(hdc, m_StartX, m_StartY, e.X, e.Y);
            m_EndX = e.X;
            m_EndY = e.Y;
            g.ReleaseHdc(hdc);
        }
    }

    protected override void OnPaint(PaintEventArgs e)
    {
```

```
            Graphics g= e.Graphics;
            g.FillRectangle(Brushes.White, this.ClientRectangle);
        }
```

程序运行结果如图 4-4 所示。

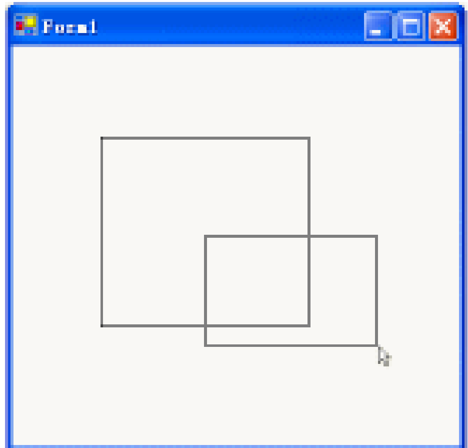

图 4-4　实现橡皮矩形

4.4　集合类

图元被创建以后需要保存起来，以便于后面进行拾取、重画等操作。被选择的图元也要单独保存起来，后面的几何变换、删除等操作将只针对这部分单独保存起来的图元进行。利用.NET 提供的集合类如 ArrayList 可以实现图元对象的保存。ArrayList 类的功能很强大，利用它可以实现不同对象的添加、索引和删除。

具体应用时，需要首先创建 ArrayList 类的实例。如下所示，创建一个 ges 对象和一个 geSels 对象，分别实现所有图元的保存和被选中图元的保存。如果用 VB.NET 编程，可以将它们设置为全局变量。但 VC#.NET 中没有全局变量，可以在 Form1 类中声明，然后作为方法的参数进行传递。

【VB.NET】

```
Public ges As New ArrayList()
Public geSels As New ArrayList()
```

【VC#.NET】

```
private ArrayList ges=new ArrayList();
private ArrayList geSels=new ArrayList();
```

集合类中图元的添加一般在图元创建完毕以后进行。下面是用于创建直线段的 CCreateLine 类中描述左键单击行为的部分代码。第 2 次单击鼠标左键以后，将新创建的 newLine 对象添加到集合类中。

【VB.NET】

```
Public Sub LButtonDown(ByVal g As Graphics, _
                ByVal aPos As PointF) Implements ICommand.LButtonDown
    '记录鼠标左键的单击次数
    m_Step += 1
    Select Case m_Step
        Case 1      '第 1 次单击鼠标左键
            '……
        Case 2     '第 2 次单击鼠标左键
            '……
            ges.Add(newLine)
            '……
    End Select
End Sub
```

【VC#.NET】

```
public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels)
{
    //记录鼠标左键的单击次数
    m_Step+=1;
    switch (m_Step)
    {
        case 1:
            //……
        case 2:
            //……
            ges.Add(newLine);
            //……
    }
}
```

选择图元时，要遍历图元类 ges 中的元素，如果第 i 个元素被拾取，则用选择模式绘制它，并将它放到选择集 geSels 中去。

【VB.NET】

```
Public Sub LButtonDown(ByVal g As System.Drawing.Graphics, _
            ByVal aPos As System.Drawing.PointF) _
            Implements CAD_Net.ICommand.LButtonDown
    Dim i As Integer
    For i = 0 To ges.Count－1
        '如果图元被拾取，则用选择模式绘制图元
        '并将该图元添加到选择集中去
```

```
            If (ges(i).Pick(aPos) = True) Then
                ges(i).Draw(g, CGElement.geDrawMode.Selec)
                geSels.Add(ges(i))
            End If
        Next
    End Sub
```

【VC#.NET】

```
    public void LButtonDown(Graphics g, PointF aPos,ArrayList ges,ArrayList geSels)
     {
         for(int i = 0;i<ges.Count;i++){
              //如果图元被拾取，则用选择模式绘制图元
              //并将该图元添加到选择集中去
              if (((CGElement)(ges[i])).Pick(aPos))
              {
                   ((CGElement)(ges[i])).Draw(g, DrawMode.Selec);
                   geSels.Add(ges[i]);
              }
         }
     }
```

图元被选到选择集中去以后，随后进行的变换操作和删除操作将只针对它们进行。下面的代码段将选择集中的图元进行平移变换。

【VB.NET】

```
    '如果选择集中的图元个数大于 0
    If geSels.Count > 0 Then
        For i = 0 To geSels.Count－1
            '清除图元在当前位置上的图形
            geSels(i).draw(g, CGElement.geDrawMode.Delete)
            '将所有被选中的图元移动到目标位置并进行绘制
            geSels(i).Move(g, m_basePos, m_desPos)
            geSels(i).Draw(g, CGElement.geDrawMode.Selec)
        Next i
     End If
```

【VC#.NET】

```
    //将所有被选中的图元移动到目标位置并进行绘制
    if (geSels.Count>0)
    {
         for (int i=0;i< geSels.Count;i++)
         {
              //清除图元在当前位置上的图形
```

```
                ((CGElement)(geSels[i])).Draw(g,DrawMode.Delete);
                ((CGElement)(geSels[i])).Move(g,m_basePos,m_desPos);
                ((CGElement)(geSels[i])).Draw(g,DrawMode.Selec);
            }
        }
```

4.5　其他技术

4.5.1　数据存盘

为了永久保存所绘制的图形，需要对图形数据进行存盘。.NET 提供了二进制序列化和反序列化方面的类，其中，序列化将对象转换成数据流，而反序列化将数据流转换为对象。利用它们可以实现图元对象的序列化和反序列化，并相应地将图元数据保存到文件或从文件中打开。

4.5.2　界面优化

设计程序界面时需要为用户着想，尽量让它们感觉到舒适和方便。工具栏和状态栏是必不可少的交互工具，前者为用户发出指令提供了一种快捷方式，后者为当前绘图状态提供了必要的信息反馈。另外还要从导航、视图等方面考虑，进行合理的窗口布局。界面设计的技巧多种多样，但总的原则是简明、便利、美观。

第 5 章　基本图元类设计

基本图元包括直线段、矩形、圆、圆弧和文本等，基本图元类用语言代码来描述这些基本图元的特征和行为，包括 CLine 类、CRectangle 类、CCircle 类、CArc 类和 CText 类等，它们从一个抽象的 CGElement 类中继承。

5.1　Win32API 类

因为要使用 SetROP2 函数获得橡皮线效果，系统中使用了较多的 API 函数，除了 SetROP2 外，还包括 CreatePen, CreateSolidBrush, GetStockObject, SelectObject, DeleteObject, MoveToEx, LineTo, Ellipse 和 Arc 等，以及一个 LPPOINT 结构。为了管理这些 API 函数，创建了一个 Win32API 类。关于 API 函数的声明和调用，请参见第 3 章的内容。需要注意的是，由于用到互操作，必须导入 System.Runtime.InteropServices 名字空间。Win32API 类中用到的 API 函数的意义如表 5-1 所示。因为 MoveToEx 等函数中要用到 LPPOINT 类型的参数，所以需要首先创建一个该类型的结构。

表 5-1　Win32API 类中声明的 API 函数的意义

函 数 名 称	说　　明
SetROP2	设置绘图模式
CreatePen	创建画笔
CreateSolidBrush	创建纯色刷子
GetStockObject	获取指定的画笔、刷子、字体或调色板
SelectObject	将对象选入设备上下文
DeleteObject	从设备上下文中删除对象
MoveToEx	将画笔移动到当前点
LineTo	从前一点向给定点画线
Ellipse	画椭圆、圆
Arc	画圆弧

【VB.NET】

```
Imports System.Runtime.InteropServices

Public Class Win32API
    <StructLayout(LayoutKind.Sequential)> Public Structure LPPOINT
        Public x As Long
        Public y As Long
    End Structure
```

```
<DllImport("gdi32.dll")> _
 Public Shared Function SetROP2(ByVal hdc As IntPtr, _
     ByVal nDrawMode As Integer) As Boolean
End Function

<DllImport("gdi32.dll")> _
Public Shared Function CreatePen(ByVal nPenStyle As Integer, _
    ByVal nWidth As Integer, ByVal crColor As Integer) _
    As Long
End Function

<DllImport("gdi32.dll")> Public Shared Function CreateSolidBrush _
    (ByVal crColor As Integer) As Long
End Function

<DllImport("gdi32.dll")> _
Public Shared Function GetStockObject(ByVal nIndex As Integer) As Long
End Function

<DllImport("gdi32.dll")> _
Public Shared Function SelectObject(ByVal hdc As IntPtr, _
    ByVal hObject As Long) As Long
End Function

<DllImport("gdi32.dll")> _
Public Shared Function DeleteObject(ByVal hObject As Long) As Long
End Function

<DllImport("gdi32.dll")> _
Public Shared Function MoveToEx _
    (ByVal hdc As IntPtr, ByVal x As Integer, ByVal y As Integer, _
    ByVal lpPoint As LPPOINT) As Boolean
End Function

<DllImport("gdi32.dll")> _
Public Shared Function LineTo _
    (ByVal hdc As IntPtr, ByVal x As Integer, _
    ByVal y As Integer) As Boolean
End Function

<DllImport("gdi32.dll")> _
```

```
        Public Shared Function Rectangle _
            (ByVal hdc As IntPtr, ByVal X1 As Integer, ByVal Y1 As Integer, _
            ByVal X2 As Integer, ByVal Y2 As Integer) As Boolean
         End Function

        <DllImport("gdi32.dll")> _
        Public Shared Function Ellipse _
          (ByVal hdc As IntPtr, ByVal X1 As Integer, ByVal Y1 As Integer, _
          ByVal X2 As Integer, ByVal Y2 As Integer) As Boolean
        End Function

        <DllImport("gdi32.dll")> _
        Public Shared Function Arc _
          (ByVal hdc As IntPtr, ByVal X1 As Integer, ByVal Y1 As Integer, _
          ByVal X2 As Integer, ByVal Y2 As Integer, ByVal X3 As Integer, _
          ByVal Y3 As Integer, ByVal X4 As Integer, ByVal Y4 As Integer) As Boolean
        End Function

    End Class
```

【VC#.NET】

```
    public class Win32API
    {
         public Win32API()
         {
         }

         [StructLayout(LayoutKind.Sequential)] public struct LPPOINT
         {
             long x;
               long y;
         }

         [DllImportAttribute("gdi32.dll")]
         public static extern bool SetROP2(IntPtr hdc, int nDrawMode);

         [DllImport("gdi32.dll")]
         public static extern bool GetROP2(IntPtr hdc);

         [DllImport("gdi32.dll")]
         public static extern long CreatePen(int nPenStyle,
```

```
            int nWidth,int crColor);

    [DllImport("gdi32.dll")]
    public static extern long CreateSolidBrush(int crColor);

    [DllImport("gdi32.dll")]
    public static extern long GetStockObject(int nIndex);

    [DllImport("gdi32.dll")]
    public static extern long SelectObject(IntPtr hdc,
            long hObject);

    [DllImport("gdi32.dll")]
    public static extern long DeleteObject(long hObject);

    [DllImport("gdi32.dll")]
    public static extern bool MoveToEx(IntPtr hdc,
          int x,int y,LPPOINT lpPoint);

    [DllImport("gdi32.dll")]
    public static extern bool LineTo (IntPtr hdc,
           int x,int y );

    [DllImport("gdi32.dll")]
    public static extern bool Rectangle(IntPtr hdc,
            int X1, int Y1,int X2,int Y2);

    [DllImport("gdi32.dll")]
    public static extern bool Ellipse(IntPtr hdc, int X1,
          int Y1, int X2,int Y2);

   [DllImport("gdi32.dll")]
    public static extern bool Arc
          (IntPtr hdc,int X1,int Y1,
          int X2,int Y2,int X3,
          int Y3,int X4,int Y4);
}
```

5.2　CGElement 类

CGElement 类是图元基类，为抽象类。它定义了图元的公共属性，如颜色、线型、线宽等。geDrawMode 枚举定义了正常、选择、拖曳和删除等 4 种绘图模式，对应于图元不同的显示效果。geStyle 枚举定义图元的线型，有实线、虚线、点线等 5 种。

Draw 方法是必须在派生类中重写的，它有两个参数：g 为 Graphics 对象，指示绘图表面；aGrawMode 为 geDrawMode 枚举类型，表示以何种绘图模式绘图。

DrawSettings 函数确定在不同绘图模式下用何种线条绘制图形。因为函数中用到了 SetROP2 函数，该函数有一个 hdc 参数，指示绘图环境的句柄，所以要传递一个 hdc 参数给它。调用 DrawSettings 函数时，利用 Graphics 对象的 GetHdc 方法可以获得该参数的值。需要注意的是，在这里，hdc 参数为 IntPtr 类型，而不是 VB 6.0 中的 Long 型。以正常模式绘图时，SetROP2 函数的第 2 个参数的值设置为 13（此时，钢笔的颜色覆盖背景色），用宽度为 1 的实线绘制，颜色为黑色；以选择模式绘图时，SetROP2 函数的设置与上面的相同，用宽度为 1 的虚线绘制，颜色为红色；以拖曳模式绘图时，SetROP2 函数的第 2 个参数的值设置为 10，用反转方式画线，此时用宽度为 1 的实线画线，颜色为蓝色；以删除模式绘图时，SetROP2 函数的设置与正常模式的相同，用宽度为 1 的实现绘制，颜色为白色。因为背景色设为白色，用白色重画图元实际上就将图元在屏幕上删除了。在 VB.NET 中，用 RGB 函数设置图元的颜色。该函数的 3 个参数分别为所设置的颜色的红色、绿色和蓝色分量。在 VC#.NET 中没有 RGB 函数，直接指定颜色值。

【VB.NET】

```
'图元基类
Public MustInherit Class CGElement
    '绘图方式
    Public Enum geDrawMode
        Normal = 1
        Selec = 2
        Drag = 3
        Delete = 4
    End Enum

    '图元的线型
    Public Enum geStyle
        Solid = 1
        Dash = 2
        Dot = 3
        DashDot = 4
        DashDotDot = 5
    End Enum
```

```
Private m_Color, m_Style, m_Width As Integer

'颜色属性
Public Property Color() As Integer
    Get
        Return m_Color
    End Get
    Set(ByVal Value As Integer)
        m_Color = Value
    End Set
End Property

'线型属性
Public Property Style() As Integer
    Get
        Return m_Style
    End Get
    Set(ByVal Value As Integer)
        m_Style = Value
    End Set
End Property

'线宽属性
Public Property Width() As Integer
    Get
        Return m_Width
    End Get
    Set(ByVal Value As Integer)
        m_Width = Value
    End Set
End Property

'构造函数
Public Sub New()
    Init()
End Sub

'初始化图元
Protected Sub Init()
    m_Style = 0
    m_Width = 1
```

```
            m_Color = RGB(0, 0, 0)
    End Sub

        '绘制图元
        Public MustOverride Sub Draw(ByVal g As Graphics, _
            ByVal aGrawMode As geDrawMode)

        '根据不同的绘图模式设置不同的绘图参数
        Public Function DrawSettings(ByVal hdc As IntPtr, _
            ByVal aDrawMode As geDrawMode) As Integer()
            Dim penPara(2) As Integer
            Select Case aDrawMode
                Case geDrawMode.Normal
                    Win32API.SetROP2(hdc, 13)
                    m_Style = 0
                    m_Width = 1
                    m_Color = RGB(0, 0, 0)
                Case geDrawMode.Selec
                    Win32API.SetROP2(hdc, 13)
                    m_Style = 1
                    m_Width = 1
                    m_Color = RGB(255, 0, 0)
                Case geDrawMode.Drag
                    Win32API.SetROP2(hdc, 10)
                    m_Style = 0
                    m_Width = 1
                    m_Color = RGB(0, 0, 255)
                Case geDrawMode.Delete
                    Win32API.SetROP2(hdc, 13)
                    m_Style = 0
                    m_Width = 1
                    m_Color = RGB(255, 255, 255)
            End Select

            penPara(0) = m_Style
            penPara(1) = m_Width
            penPara(2) = m_Color

            Return penPara
        End Function
    End Class
```

【VC#.NET】

```
public abstract class CGElement
{
    private int m_Color, m_Style, m_Width;

    public int Color
    {
        get{return m_Color;}
        set{m_Color=value;}
    }

    public int Style
    {
        get{return m_Style;}
        set{m_Style=value;}
    }

    public int Width
    {
        get{return m_Width;}
        set{m_Width=value;}
    }

    public CGElement()
    {
        init();
    }

    //初始化图元
    protected void init()
    {
        m_Style=0;
        m_Width=1;
        m_Color=0;
    }

    //绘制图元
    abstract public void Draw(Graphics g,DrawMode aDrawMode);

    //根据不同的绘图模式设置不同的绘图参数
```

```
public int[] DrawSettings(IntPtr hdc,DrawMode aDrawMode)
{
    switch(aDrawMode)
    {
        case DrawMode.Normal:
            Win32API.SetROP2(hdc, 13);
            m_Style = 0;
            m_Width = 1;
            m_Color =0;
                break;
        case DrawMode.Selec:
            Win32API.SetROP2(hdc, 13);
            m_Style = 1;
            m_Width = 1;
            m_Color =255;
                break;
        case DrawMode.Drag:
            Win32API.SetROP2(hdc, 10);
            m_Style = 0;
            m_Width = 1;
            m_Color =16711680;
                break;
        case DrawMode.Delete:
            Win32API.SetROP2(hdc, 13);
            m_Style = 0;
            m_Width = 1;
            m_Color =16777215;
                break;
    }

    int[] penPara={m_Style,m_Width,m_Color};

    return penPara;
}
```

5.3　CLine 类

CLine 类是 CGElement 类的派生类。它继承 CGElement 类的 Color, Style 和 Width 等属性和 DrawSettings 方法，添加自己的起点和终点属性，重写基类的 Draw 方法。

该类提供了 3 个构造函数，包括一个无参构造函数和两个带参构造函数。使用无参构造

函数创建 CLine 类实例时，用 Init 方法初始化该实例的属性数据。由于基类中也有 Init 方法，所以这里需要使用 Shadows 关键字或 new 关键字遮蔽基类的该方法。遮蔽以后，基类的 Init 方法在 CLine 类中就不能直接使用了，但通过 MyBase 对象或 base 对象可以调用。利用第 2 种构造函数可以根据给定的两个点构造一条直线段，调用 Init 函数进行初始化。第 3 种构造函数根据已知的直线段构造新直线段。

【VB.NET】

```
'直线段类
Imports System.Math
Imports System.Drawing.Drawing2D

Public Class CLine
    Inherits CGElement

    Private m_Begin, m_End As PointF

    '直线段的起点属性
    Public Property LBegin() As PointF
        Get
            Return m_Begin
        End Get
        Set(ByVal Value As PointF)
            m_Begin = Value
        End Set
    End Property

    '直线段的终点属性
    Public Property LEnd() As PointF
        Get
            Return m_End
        End Get
        Set(ByVal Value As PointF)
            m_End = Value
        End Set
    End Property

    '无参构造函数
    Public Sub New()
        Init()
    End Sub
```

```
'构造函数，用已知的两点构造直线段
Public Sub New(ByVal pBegin As PointF, ByVal pEnd As PointF)
    Init()
    m_Begin = pBegin
    m_End = pEnd
End Sub

'构造函数，用已知的直线段构造直线段
Public Sub New(ByVal aline As CLine)
    m_Begin = aline.LBegin
    m_End = aline.LEnd
End Sub

'初始化直线段
Private Shadows Sub Init()
    MyBase.Init()
    With m_Begin
        .X = 0
        .Y = 0
    End With
    m_End = m_Begin
End Sub
```

【VC#.NET】

```
//直线段类
public class CLine:CGElement
{
    protected PointF m_Begin, m_End;
    Module m=new Module();

    public PointF LBegin
    {
        get{return m_Begin;}
        set{m_Begin=value;}
    }

    public PointF LEnd
    {
        get{return m_End;}
        set{m_End=value;}
    }
```

```
public CLine(PointF pBegin,PointF pEnd)
{
    Init();
    m_Begin=pBegin;
    m_End=pEnd;
}

public CLine(CLine aline)
{
    m_Begin=aline.LBegin;
    m_End=aline.LEnd;
}

private new void Init()
{
    base.Init();
    m_Begin=new PointF(0,0);
    m_End=new PointF(0,0);
}
```

然后重写基类的 Draw 方法。绘图工作总是在设备坐标系中完成的，所以绘图以前，需要将直线段的起点坐标和终点坐标从通用坐标转换为设备坐标。在第 3 章中讨论坐标系统时曾经讲到，通用坐标不是直接转换为设备坐标的，而是先转换为页面坐标，然后由页面坐标转换为设备坐标的。以像素作为度量单位时，页面坐标与设备坐标相同，所以，把通用坐标转换为页面坐标时，在数值上实际上也就获得了设备坐标。

由于采用 API 函数绘制图元，所以要先获取绘图环境的句柄 hdc，利用 Graphics 对象的 GetHdc 方法获得它。然后调用 DrawSettings 方法得到当前绘图模式下的画笔参数。因为 CLine 类是 CGElement 类的派生类，它们之间是继承和被继承的关系，所以 CLine 类就自然拥有 DrawSettings 方法，在类代码中不做任何声明就可以直接调用它。根据刚刚获取的画笔参数，用 CreatePen 函数创建一支画笔。用 SelectObject 函数将该画笔选入绘图环境，并返回原来的画笔。现在有笔了，用 MoveToEx 函数把它移到要绘制的直线段的起点处，用 LineTo 函数绘制直线段。直线段绘好以后，做一些善后工作，把原来的画笔重新选入到绘图环境中，并删除刚刚创建的画笔，用 Graphics 对象的 ReleaseHdc 方法释放绘图环境的句柄。

需要注意的是，由于 VC#.NET 是强类型的语言，某些隐式转换是不能完成的，如将 double 型转换为 float 型，或将 float 型转换为 int 型等，这时必须使用显式转换。API 绘图函数中的坐标参数是 int 型的，而给定的坐标值是 float 型的，所以必须进行显式转换。

【VB.NET】

```
'绘直线段
Public Overrides Sub Draw(ByVal g As Graphics, ByVal aDrawMode As geDrawMode)
    Dim aPen, oldP As Long
```

```
        '将直线段的起点和终点坐标转换为页面坐标
        Dim eb As PointF = WorldtoPage(m_Begin)
        Dim ee As PointF = WorldtoPage(m_End)

        '获取绘图环境的句柄
        Dim hdc As IntPtr
        hdc = g.GetHdc()
        '设置画笔参数
        Dim penPara As Integer() = DrawSettings(hdc, aDrawMode)
        '创建画笔
        aPen = Win32API.CreatePen(penPara(0), penPara(1), penPara(2))
        '把画笔选入绘图环境，并返回原来的画笔
        oldP = Win32API.SelectObject(hdc, aPen)
        '把画笔移动到直线段的起点处
        Win32API.MoveToEx(hdc, eb.X, eb.Y, Nothing)
        '绘直线段到终点
        Win32API.LineTo(hdc, ee.X, ee.Y)
        '把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP)
        '删除新创建的画笔
        Win32API.DeleteObject(aPen)
        '释放绘图环境句柄
        g.ReleaseHdc(hdc)
    End Sub

End Class
```

【VC#.NET】

```
    //绘直线段
    override  public void Draw(Graphics g,DrawMode aDrawMode)
    {
        long aPen,oldP;

        //将控制点的坐标由通用坐标转换为页面坐标
        PointF eb=m.WorldtoPage(m_Begin);
        PointF ee=m.WorldtoPage(m_End);

        //获得当前绘图环境的句柄
        IntPtr hdc=g.GetHdc();
```

```
        //设置画笔参数
        int[] penPara=DrawSettings(hdc,aDrawMode);
        //创建画笔
        aPen=Win32API.CreatePen(penPara[0],penPara[1],penPara[2]);
        //把画笔选入绘图环境，并返回原来的画笔
        oldP=Win32API.SelectObject(hdc,aPen);
        Win32API.LPPOINT prePos=new Win32API.LPPOINT();
        //把画笔移动到直线段的起点处
        Win32API.MoveToEx(hdc, (int)(eb.X), (int)(eb.Y), prePos);
        //绘直线段到终点
        Win32API.LineTo(hdc, (int)(ee.X), (int)(ee.Y));
        //把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP);
        //删除新创建的画笔
        Win32API.DeleteObject(aPen);
        //释放绘图环境句柄
        g.ReleaseHdc(hdc);
    }
```

5.4　CRectangle 类

CRectangle 类从 CGElement 类派生。它除了继承 CGElement 类的 Color，Style 和 Width 等属性和 DrawSettings 方法，还要添加绘制矩形时的起点和终点属性，并根据这两个点的坐标得到矩形左上角点和右下角点的坐标，它们是只读的。需要说明的是，用鼠标交互绘制矩形时，可以从左上角向右下角绘，也可以从右下角向左上角绘，所以起点、终点和左上角、右下角的顶点不一定是相同的。根据起点和终点的坐标，用两者横坐标的小值和纵坐标的大值构造左上角点，用两者横坐标的大值和纵坐标的小值构造右下角点。其间用到了求最大值和最小值的函数，在 VB.NET 程序中需要导入 System.Math 名字空间。然后重写基类的 Draw 方法，调用 Win32API 类的 Rectangle 方法绘制矩形。

这里，仍然提供了 3 个构造函数，一个为无参构造函数，另一个用给定的两个点作为创建矩形时的起点和终点构造矩形，第 3 个用已知的矩形构造矩形。用构造函数创建 CRectangle 类的实例时，用 Init 方法进行初始化，将起点和终点的位置初始化为坐标原点。

【VB.NET】

```
Imports System.Math

Public Class CRectangle
    Inherits CGElement

    Private m_basePos, m_desPos As PointF
```

```
'绘矩形时的起点
Public Property basePos() As PointF
    Get
        Return m_basePos
    End Get
    Set(ByVal Value As PointF)
        m_basePos = Value
    End Set
End Property

'绘矩形时的终点
Public Property desPos() As PointF
    Get
        Return m_desPos
    End Get
    Set(ByVal Value As PointF)
        m_desPos = Value
    End Set
End Property

'矩形的左上角点
Public ReadOnly Property LT() As PointF
    Get
        Return (New PointF(Min(m_basePos.X, m_desPos.X), _
              Max(m_basePos.Y, m_desPos.Y)))
    End Get
End Property

'矩形的右下角点
Public ReadOnly Property RB() As PointF
    Get
        Return (New PointF(Max(m_basePos.X, m_desPos.X), _
              Min(m_basePos.Y, m_desPos.Y)))
    End Get
End Property

'无参构造函数
Public Sub New()
    Init()
End Sub
```

```
'构造函数，用已知的两点构造矩形
Public Sub New(ByVal pBase As PointF, ByVal pDes As PointF)
    Init()
    m_basePos = pBase
    m_desPos = pDes
End Sub

'构造函数，用已知的矩形构造矩形
Public Sub New(ByVal arectangle As CRectangle)
    m_basePos = arectangle.basePos
    m_desPos = arectangle.desPos
End Sub

'初始化矩形
Private Shadows Sub Init()
    MyBase.Init()
    With m_basePos
        .X = 0
        .Y = 0
    End With
    m_desPos = m_basePos
End Sub
```

【VC#.NET】

```
public class CRectangle:CGElement
{
    protected PointF m_basePos, m_desPos;
    Module m=new Module();

    //绘矩形时的起点
    public PointF basePos
    {
        get{return m_basePos;}
        set{m_basePos=value;}
    }

    //绘矩形时的终点
    public PointF desPos
    {
        get{return m_desPos;}
        set{m_desPos=value;}
```

```
}

//矩形的左上角点
public PointF LT
{
    get
    {
        return (new PointF(Math.Min(m_basePos.X, m_desPos.X),
       Math.Max(m_basePos.Y, m_desPos.Y)));
    }
}

//矩形的右下角点
public PointF RB
{
    get
    {
        return (new PointF(Math.Max(m_basePos.X, m_desPos.X),
            Math.Min(m_basePos.Y, m_desPos.Y)));
    }
}

public CRectangle()
{
    Init();
}

public CRectangle(PointF pBase,PointF pDes)
{
    Init();
    m_basePos=pBase;
    m_desPos=pDes;
}

public CRectangle(CRectangle aRect)
{
    m_basePos=aRect.basePos ;
    m_desPos=aRect.desPos ;
}

private new void Init()
```

```
{
    base.Init();
    m_basePos=new PointF(0,0);
    m_desPos=new PointF(0,0);
}
```

通过重写基类的 Draw 方法来绘制矩形。绘矩形时，需要首先将矩形的起点和终点坐标转换为页面坐标，然后求取页面坐标下矩形左上角和右下角的坐标。使用 VC#.NET 时，还需要进行显式转换，将计算得到的浮点型坐标值转换为整型值。进行绘图设置以后，用 Rectangle 函数绘制矩形。需要注意的是，因为矩形是一个封闭的图形，用 API 函数 Rectangle 绘图之前还需要将一把空刷子选入绘图环境。

【VB.NET】

```
'绘矩形
Public Overrides Sub Draw(ByVal g As Graphics, ByVal aDrawMode As geDrawMode)
    Dim aPen, oldP As Long

    '将绘矩形时的起点和终点坐标转换为页面坐标
    Dim eb As PointF = WorldtoPage(m_basePos)
    Dim ed As PointF = WorldtoPage(m_desPos)

    Dim minX, minY As Single
    Dim maxX, maxY As Single
    minX = Min(eb.X, ed.X)
    minY = Min(eb.Y, ed.Y)
    maxX = Max(eb.X, ed.X)
    maxY = Max(eb.Y, ed.Y)

    '获取绘图环境的句柄
    Dim hdc As IntPtr
    hdc = g.GetHdc()
    '设置画笔参数
    Dim penPara As Integer() = DrawSettings(hdc, aDrawMode)
    '创建画笔
    aPen = Win32API.CreatePen(penPara(0), penPara(1), penPara(2))
    '把画笔选入绘图环境，并返回原来的画笔
    oldP = Win32API.SelectObject(hdc, aPen)
    '把空刷子选入绘图环境
    Win32API.SelectObject(hdc, Win32API.GetStockObject(5))
    '绘制矩形
    Win32API.Rectangle(hdc, minX, maxY, maxX, minY)
    '把原来的画笔选入绘图环境
```

```
        Win32API.SelectObject(hdc, oldP)
        '删除新创建的画笔
        Win32API.DeleteObject(aPen)
        '释放绘图环境句柄
        g.ReleaseHdc(hdc)
    End Sub
```

【VC#.NET】

```
    //绘矩形
    override    public void Draw(Graphics g,DrawMode aDrawMode)
    {
        long aPen,oldP;

        //将控制点的坐标由通用坐标转换为页面坐标
        PointF eb=m.WorldtoPage(m_basePos);
        PointF ed=m.WorldtoPage(m_desPos);

        int minX, minY,maxX, maxY;
        minX = (int)(Math.Min(eb.X, ed.X));
        minY = (int)(Math.Min(eb.Y, ed.Y));
        maxX = (int)(Math.Max(eb.X, ed.X));
        maxY = (int)(Math.Max(eb.Y, ed.Y));

        //获得当前绘图环境的句柄
        IntPtr hdc=g.GetHdc();

        //设置画笔参数
        int[] penPara=DrawSettings(hdc,aDrawMode);
        //创建画笔
        aPen=Win32API.CreatePen(penPara[0],penPara[1],penPara[2]);
        //把画笔选入绘图环境，并返回原来的画笔
        oldP=Win32API.SelectObject(hdc,aPen);
        Win32API.LPPOINT prePos=new Win32API.LPPOINT();
        //把空刷子选入绘图环境
        Win32API.SelectObject(hdc, Win32API.GetStockObject(5));
        //绘制矩形
        Win32API.Rectangle(hdc, minX, maxY, maxX, minY);
        //把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP);
        //删除新创建的画笔
        Win32API.DeleteObject(aPen);
```

```
        //释放绘图环境句柄
        g.ReleaseHdc(hdc);
    }
```

5.5 CCircle 类

在项目中添加 CCircle 类，它是 CGElement 类的派生类。CCircle 类也要继承 CGElement 类的 3 个属性和 DrawSettings 方法。除此之外，CCircle 类还要提供确定圆本身所需要的圆心和圆上一点两个属性，另外还有一个 Radius 属性，它是只读的，根据圆心坐标和圆上一点的坐标，可以计算出半径 Radius。与 CLine 类一样，CCircle 也提供了 3 个构造函数。其中，根据两个给定的点构造一个圆是最常用的。

由于计算半径 Radius 时要用到数学函数，所以需要导入 System.Math 名字空间。但是，在 VC#.NET 中没有该名字空间，可以直接调用 Math 类。

【VB.NET】

```
'圆类
Imports System.Math

Public Class CCircle
    Inherits CGElement

    Private m_Center, m_PCircle As PointF

    '圆心属性
    Public Property Center() As PointF
        Get
            Return m_Center
        End Get
        Set(ByVal Value As PointF)
            m_Center = Value
        End Set
    End Property

    '圆上一点属性
    Public Property PCircle() As PointF
        Get
            Return m_PCircle
        End Get
        Set(ByVal Value As PointF)
            m_PCircle = Value
        End Set
```

```
End Property

'半径属性，只读
Public ReadOnly Property Radius() As Single
    Get
        Dim r As Single
        r = distPtoP(m_Center, m_pCircle)
        Return r
    End Get
End Property

'无参构造函数
Public Sub New()
    Init()
End Sub

'构造函数，用已知的两个点构造圆
Public Sub New(ByVal pCenter As PointF, ByVal pCircle As PointF)
    Init()
    m_Center = pCenter
    m_PCircle = pCircle
End Sub

'构造函数，用已知的圆构造圆
Public Sub New(ByVal aCircle As CCircle)
    With aCircle
        m_Center = .Center
        m_PCircle = .PCircle
    End With
End Sub

'初始化圆
Private Shadows Sub Init()
    MyBase.Init()
    With m_Center
        .X = 0
        .Y = 0
    End With
    m_PCircle = m_Center
End Sub
```

【VC#.NET】

```
public class CCircle:CGElement
{
        protected PointF m_Center, m_pCircle;
        Module m=new Module();

        public PointF Center
        {
                get{return m_Center;}
                set{m_Center=value;}
        }

        public PointF PCircle
        {
                get{return m_pCircle;}
                set{m_pCircle=value;}
        }

        public float Radius
        {
                get
                {
                        float r=m.DistPtoP(m_Center,m_pCircle);
                        return r;
                }
        }

        public CCircle()
        {
            Init();
        }

        public CCircle(PointF pCenter,PointF pCircle)
        {
                Init();
                m_Center=pCenter;
                m_pCircle=pCircle;
        }

        public CCircle(CCircle aCircle)
```

```
{
    m_Center=aCircle.Center;
    m_pCircle=aCircle.PCircle;
}

private new void Init()
{
    base.Init();
    m_Center=new PointF(0,0);
    m_pCircle=new PointF(0,0);
}
```

【VB.NET】

```
'计算点与点之间的距离
Public Function DistPtoP(ByVal p1 As PointF, ByVal p2 As PointF) As Single
    Dim dx, dy As Single
    dx = p2.X - p1.X
    dy = p2.Y - p1.Y
    Return Sqrt((dx * dx) + (dy * dy))
End Function
```

【VC#.NET】

```
//计算点与点之间的距离
public float DistPtoP(PointF p1, PointF p2){
    float dx,dy;
    dx = p2.X - p1.X;
    dy = p2.Y - p1.Y;
    float dist=(float)(Math.Sqrt((dx * dx) + (dy * dy)));
    return dist;
}
```

在 CCircle 类中重写基类的 Draw 方法，根据不同的绘图模式绘圆。在绘图之前，同样需要将圆心和圆上一点的坐标从通用坐标转换为页面坐标。像素模式下，设备坐标的大小与该页面坐标相同。后面绘图的过程也基本上与绘直线段相同，但是需要选一把空刷子到绘图环境中，否则绘出的是黑色的圆形区域，该操作用 GetStockObject 函数实现，需要从 Win32API 类中调用它。绘圆使用的是 Ellipse 函数。

【VB.NET】

```
'绘圆
Public Overrides Sub Draw(ByVal g As Graphics, ByVal aDrawMode As geDrawMode)
    Dim aPen, oldP As Long

    '将控制点的坐标转换为页面坐标
```

```
        Dim ec As PointF = WorldtoPage(m_Center)
        Dim ep As PointF = WorldtoPage(m_PCircle)

        '获取绘图环境的句柄
        Dim hdc As IntPtr
        hdc = g.GetHdc()
        '设置画笔参数
        Dim penPara As Integer() = DrawSettings(hdc, aDrawMode)
        '创建画笔
        aPen = Win32API.CreatePen(penPara(0), penPara(1), penPara(2))
        '把画笔选入绘图环境，并返回原来的画笔
        oldP = Win32API.SelectObject(hdc, aPen)
        '把空刷子选入绘图环境
        Win32API.SelectObject(hdc, Win32API.GetStockObject(5))
        '绘圆
        Win32API.Ellipse(hdc, ec.X － Radius, ec.Y + Radius, _
            ec.X + Radius, ec.Y － Radius)
        '把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP)
        '删除新创建的画笔
        Win32API.DeleteObject(aPen)
        '释放绘图环境句柄
        g.ReleaseHdc(hdc)
    End Sub

End Class
```

【VC#.NET】

```
    //绘圆
    override public void Draw(Graphics g,DrawMode aDrawMode)
    {
        long aPen,oldP;

        //将控制点的坐标由通用坐标转换为页面坐标
        PointF ec=m.WorldtoPage(m_Center);

        //获得当前绘图环境的句柄
        IntPtr hdc=g.GetHdc();

        //设置画笔参数
        int[] penPara=DrawSettings(hdc,aDrawMode);
```

```
        //创建画笔
        aPen=Win32API.CreatePen(penPara[0],penPara[1],penPara[2]);
        //把画笔选入绘图环境，并返回原来的画笔
        oldP=Win32API.SelectObject(hdc,aPen);
        //绘圆
        Win32API.Ellipse(hdc, (int)(ec.X – Radius),
                    (int)(ec.Y + Radius),
                    (int)(ec.X + Radius),
                    (int)(ec.Y – Radius));
        //把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP);
        //删除新创建的画笔
        Win32API.DeleteObject(aPen);
        //释放绘图环境句柄
        g.ReleaseHdc(hdc);
    }
```

5.6　CArc 类

CArc 类继承了 CGElement 类，定义圆弧的属性和圆心、起点、终点三点绘圆的方法。颜色、线型、线宽等属性以及指定绘图模式下的画笔参数获取函数从基类中继承，另外还要定义圆弧特有的圆心、起点和终点属性。从这 3 个点还可以计算出圆弧所在圆的半径以及起点和终点相对于圆心的方位角，所以可以提供分别定义这 3 个数据的属性，它们是只读的。

CArc 类也有 3 个构造函数。除了无参构造函数和用已知圆弧构造新圆弧的构造函数外，还提供了利用给定的 3 个点构造圆弧的构造函数。其中，这 3 个点分别是指定的圆心、起点和另外一点。这另外一点不一定是圆弧的终点。为什么不直接指定终点呢？因为交互绘图的过程中，第 1 次单击鼠标左键时确定圆心，第 2 次单击鼠标左键时确定圆弧的起点，然后鼠标移动。该移动的过程中，鼠标的光标位置是不确定的，它不可能正好落在圆弧上。但这个点还是有价值的，因为利用它相对于圆心的方位角和根据前两个点得到的半径，可以将这个点对应的终点计算出来。这个问题可以抽象为已知方向和距离，求直线段上一点坐标的问题。下面的代码中给出了解答。

CArc 类用到多个数学函数，用 VB.NET 编程时，需要导入 System.Math 名字空间。

【VB.NET】

```
'圆弧类
Imports System.Math


Public Class CArc
    Inherits CGElement


    Private m_Center, m_Begin, m_End As PointF
```

```
'圆心属性
Public Property Center() As Pointf
    Get
        Return m_Center
    End Get
    Set(ByVal Value As Pointf)
        m_Center = Value
    End Set
End Property

'圆弧起点属性
Public Property ABegin() As Pointf
    Get
        Return m_Begin
    End Get
    Set(ByVal Value As Pointf)
        m_Begin = Value
    End Set
End Property

'圆弧终点属性
Public Property AEnd() As Pointf
    Get
        Return m_End
    End Get
    Set(ByVal Value As Pointf)
        m_End = Value
    End Set
End Property

'圆弧半径属性，只读
Public ReadOnly Property Radius() As Single
    Get
        Dim r As Single
        r = distPtoP(m_Center, m_Begin)
        Return r
    End Get
End Property

'圆弧起点的方位角
```

```
Public ReadOnly Property AngleBegin() As Single
    Get
        Dim aAngle As Single
        aAngle = GetAngle(m_Center, m_Begin)
        Return aAngle
    End Get
End Property

'圆弧终点的方位角
Public ReadOnly Property AngleEnd() As Single
    Get
        Dim aAngle As Single
        aAngle = GetAngle(m_Center, m_End)
        Return aAngle
    End Get
End Property

'无参构造函数
Public Sub New()
    Init()
End Sub

'构造函数，用已知的圆心、起点和另一点进行构造
Public Sub New(ByVal pCenter As PointF, ByVal p1 As PointF, ByVal p2 As PointF)
    Init()
    m_Center = pCenter
    m_Begin = p1
    Dim r As Single = DistPtoP(m_Center, m_Begin)
    Dim angle2 As Single = GetAngle(m_Center, p2)
    m_End.X = r * Cos(angle2) + m_Center.X
    m_End.Y = r * Sin(angle2) + m_Center.Y
End Sub

'构造函数，用已知的一个圆弧进行构造
Public Sub New(ByVal aArc As CArc)
    With aArc
        m_Center = .Center
        m_Begin = .ABegin
        m_End = .AEnd
    End With
End Sub
```

```
'初始化圆弧
Private Shadows Sub Init()
    MyBase.Init()
    With m_Center
        .X = 0
        .Y = 0
    End With
    m_Begin = m_Center
    m_End = m_Center
End Sub
```

【VC#.NET】

```
public class CArc:CGElement
{
    protected PointF m_Center, m_Begin, m_End;
    Module m=new Module();

    public PointF Center
    {
        get{return m_Center;}
        set{m_Center=value;}
    }

    public PointF ABegin
    {
        get{return m_Begin;}
        set{m_Begin=value;}
    }

    public PointF AEnd
    {
        get{return m_End;}
        set{m_End=value;}
    }

    public float Radius
    {
        get
        {
            float r=m.DistPtoP(m_Center,m_Begin);
```

```
            return r;
        }
    }

    public float AngleBegin
    {
        get
        {
            float aAngle=m.GetAngle(m_Center,m_Begin);
            return aAngle;
        }
    }

    public float AngleEnd
    {
        get
        {
            float aAngle=m.GetAngle(m_Center,m_End);
            return aAngle;
        }
    }

    public CArc()
    {
      Init();
    }

    //构造函数，用已知的圆心、起点和另一点进行构造
    public    CArc(PointF pCenter,PointF p1,PointF p2)
    {
        Init();
        m_Center=pCenter;
        m_Begin=p1;
        float r=m.DistPtoP(m_Center,m_Begin);
        float angle2=m.GetAngle(m_Center,p2);
        m_End.X= (float)(r*Math.Cos(angle2))+m_Center.X;;
        m_End.Y= (float)(r*Math.Sin(angle2))+m_Center.Y;
    }

    //构造函数，用已知的一个圆弧进行构造
    public    CArc(CArc aArc)
```

```
{
        m_Center=aArc.Center;
        m_Begin=aArc.ABegin;
        m_End=aArc.AEnd;
}

//初始化圆弧
  private new void Init()
  {
        base.Init();
        m_Center=new PointF(0,0);
        m_Begin=new PointF(0,0);
        m_End=new PointF(0,0);
  }
```

前面用圆心、起点和第 3 点构造圆弧的构造函数用到了 GetAngle 函数。该函数返回两点连线的方位角，即两点连线与 X 轴正向之间的夹角。计算方位角时需要比较两个点的横纵坐标，根据终点相对于起点的位置得到所求的角度。GetAngle 函数的代码如下：

【VB.NET】

```
'计算点间连线与 X 轴正向之间的夹角
Public Function GetAngle(ByVal p1 As PointF, ByVal p2 As PointF) As Single
    Dim tansita, sita, dx As Single
    dx = Abs(p2.X - p1.X)
    If p2.X = p1.X Then dx = 0.0001
    tansita = Abs(p2.Y - p1.Y) / dx
    sita = Atan(tansita)
    '如果终点横坐标大于、等于起点横坐标，并且终点纵坐标大于、等于起点纵坐标
    If p2.X >= p1.X And p2.Y >= p1.Y Then
        Return sita
        '如果终点横坐标小于、等于起点横坐标，并且终点纵坐标大于、等于起点纵坐标
    ElseIf p2.X <= p1.X And p2.Y >= p1.Y Then
        Return PI - sita
        '如果终点横坐标小于、等于起点横坐标，并且终点纵坐标小于、等于起点纵坐标
    ElseIf p2.X <= p1.X And p2.Y <= p1.Y Then
        Return PI + sita
        '如果终点横坐标大于、等于起点横坐标，并且终点纵坐标小于、等于起点纵坐标
    ElseIf p2.X >= p1.X And p2.Y <= p1.Y Then
        Return PI * 2 - sita
    End If
End Function
```

【VC#.NET】

```
//计算点间连线与 X 轴正向之间的夹角
public float GetAngle(PointF p1,PointF p2){
    float tansita,sita,dx;
    float angle=0;
    dx = Math.Abs(p2.X - p1.X);
    if (p2.X == p1.X){dx = 0.0001f;}
    tansita = Math.Abs(p2.Y - p1.Y) / dx;
    sita = (float)(Math.Atan(tansita));
    //如果终点横坐标大于、等于起点横坐标，并且终点纵坐标大于、等于起点纵坐标
    if (p2.X >= p1.X && p2.Y >= p1.Y){angle=sita;}
    //如果终点横坐标小于、等于起点横坐标，并且终点纵坐标大于、等于起点纵坐标
    if (p2.X <= p1.X && p2.Y >= p1.Y){angle=Const.PI - sita;}
    //如果终点横坐标小于、等于起点横坐标，并且终点纵坐标小于、等于起点纵坐标
    if (p2.X <= p1.X && p2.Y <= p1.Y) {angle=Const.PI + sita;}
    //如果终点横坐标大于、等于起点横坐标，并且终点纵坐标小于、等于起点纵坐标
    if (p2.X >= p1.X && p2.Y <= p1.Y) {angle=Const.PI * 2 - sita;}

    return angle;
}
```

CArc 类还需要重写基类的 Draw 方法，根据不同的绘图模式绘制圆弧。实现 Draw 方法的过程与前面各类中的基本相同。绘制圆弧使用的是 API 函数 Arc，可从 Win32API 类中调用该函数。

【VB.NET】

```
'绘图
Public Overrides Sub Draw(ByVal g As Graphics, ByVal aDrawMode As geDrawMode)
    Dim aPen, oldP As Long

    '将控制点的坐标由通用坐标转换为页面坐标
    Dim ec As PointF = WorldtoPage(m_Center)
    Dim eb As PointF = WorldtoPage(m_Begin)
    Dim ee As PointF = WorldtoPage(m_End)

    '获得当前绘图环境的句柄
    Dim hdc As IntPtr
    hdc = g.GetHdc()

    '设置画笔参数
    Dim penPara As Integer() = DrawSettings(hdc, aDrawMode)
    '创建画笔
```

```
        aPen = Win32API.CreatePen(penPara(0), penPara(1), penPara(2))
        '把画笔选入绘图环境，并返回原来的画笔
        oldP = Win32API.SelectObject(hdc, aPen)
        '用刚刚创建的画笔绘圆弧
        Win32API.Arc(hdc, ec.X - Radius, ec.Y + Radius, _
            ec.X + Radius, ec.Y - Radius, eb.X, _
            eb.Y, ee.X, ee.Y)
        '把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP)
        '删除新创建的画笔
        Win32API.DeleteObject(aPen)
        '释放绘图环境句柄
        g.ReleaseHdc(hdc)
    End Sub

End Class
```

【VC#.NET】

```
    //绘图
    override public void Draw(Graphics g,DrawMode aDrawMode)
    {
        long aPen,oldP;

        //将控制点的坐标由通用坐标转换为页面坐标
        PointF ec=m.WorldtoPage(m_Center);
        PointF eb=m.WorldtoPage(m_Begin);
        PointF ee=m.WorldtoPage(m_End);

        //获得当前绘图环境的句柄
        IntPtr hdc=g.GetHdc();

        //设置画笔参数
        int[] penPara=DrawSettings(hdc,aDrawMode);
        //创建画笔
        aPen=Win32API.CreatePen(penPara[0],penPara[1],penPara[2]);
        //把画笔选入绘图环境，并返回原来的画笔
        oldP=Win32API.SelectObject(hdc,aPen);
        //用刚刚创建的画笔绘圆弧
        Win32API.Arc(hdc, (int)(ec.X - Radius), (int)(ec.Y + Radius),
                (int)(ec.X + Radius), (int)(ec.Y - Radius), (int)(eb.X),
                (int)(eb.Y), (int)(ee.X), (int)(ee.Y));
```

```
        //把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP);
         //删除新创建的画笔
        Win32API.DeleteObject(aPen);
        //释放绘图环境句柄
        g.ReleaseHdc(hdc);
    }
```

5.7　CText 类

CText 类从 CGElement 类继承，用于定义文本的基本属性和与交互绘制有关的方法。文本的属性包括字体名称、插入点、内容、大小、旋转角度、颜色、字体样式等。CText 类提供了多个构造函数，可以根据给定的属性值和给定的文本构造 CText 对象。采用 Graphics 对象的 DrawString 方法绘制文本。

【VB.NET】

```
    '文本类
    Public Class CText
        Inherits CGElement

        Private m_Font As Font
        Private m_Pos As PointF
        Private m_Content As String
        Private m_Size, m_Angle As Single
        Private m_Color As Color
        Private m_Style As FontStyle

        '字体属性
        Public Property Font() As Font
            Get
                Return m_Font
            End Get
            Set(ByVal Value As Font)
                m_Font = Value
            End Set
        End Property

        '位置属性
        Public Property Pos() As PointF
            Get
                Return m_Pos
```

```
        End Get
        Set(ByVal Value As PointF)
            m_Pos = Value
        End Set
    End Property

    '大小属性
    Public Property Size() As Single
        Get
            Return m_Size
        End Get
        Set(ByVal Value As Single)
            m_Size = Value
        End Set
    End Property

    '内容属性
    Public Property Content() As String
        Get
            Return m_Content
        End Get
        Set(ByVal Value As String)
            m_Content = Value
        End Set
    End Property

    '旋转角度属性
    Public Property Angle() As Single
        Get
            Return m_Angle
        End Get
        Set(ByVal Value As Single)
            m_Angle = Value
        End Set
    End Property

    '颜色属性
    Public Shadows Property Color() As Color
        Get
            Return m_Color
        End Get
```

```
        Set(ByVal Value As Color)
            m_Color = Value
        End Set
    End Property

    '字体模式
    Public Shadows Property Style() As FontStyle
        Get
            Return m_Style
        End Get
        Set(ByVal Value As FontStyle)
            m_Style = Value
        End Set
    End Property

    '无参构造函数
    Public Sub New()
        Init()
    End Sub

    '构造函数，用已知的字符串进行构造
    Public Sub New(ByVal str As String, ByVal aPos As PointF)
        Init()
        m_Content = str
        m_Pos = aPos
    End Sub

    '构造函数
    Public Sub New(ByVal str As String, ByVal f As Font, _
                   ByVal size As Single, _
                   ByVal aPos As PointF, ByVal ang As Single, _
                   ByVal c As Color, ByVal sty As FontStyle)
        m_Content = str
        m_Font = f
        m_Size = size
        m_Pos = aPos
        m_Angle = ang
        m_Color = c
        m_Style = sty
    End Sub
```

```
    '构造函数，用已知的文本进行构造
    Public Sub New(ByVal text As CText)
        With text
            m_Content = .Content
            m_Font = .Font
            m_Pos = .Pos
            m_Size = .Size
            m_Angle = .Angle
            m_Color = .Color
            m_Style = .Style
        End With
    End Sub

    '初始化文本
    Private Shadows Sub Init()
        m_Content = " "
        m_Size = 20
        m_Font = New Font("宋体", m_Size, m_Style, GraphicsUnit.Pixel)
        Dim format As New StringFormat()
        format.Alignment = StringAlignment.Near
        m_Pos = New PointF(0, 0)
        m_Angle = 0
        m_Color = Color.Black
        m_Style = FontStyle.Regular
    End Sub

    '绘文本
    Public Overrides Sub Draw(ByVal g As Graphics, ByVal aDrawMode As geDrawMode)
        Select Case aDrawMode
            Case CGElement.geDrawMode.Normal
                m_Color = Color.Black
            Case CGElement.geDrawMode.Selec
                m_Color = Color.Red
            Case CGElement.geDrawMode.Delete
                m_Color = Color.White
        End Select
        Dim Pos As PointF = WorldtoPage(m_Pos)
        Dim sf As New StringFormat(StringFormatFlags.NoWrap)
        g.DrawString(m_Content, m_Font, New SolidBrush(m_Color), Pos.X, Pos.Y, sf)
    End Sub

End Class
```

【VC#.NET】

```
public class CText:CGElement
{
    private Font m_Font;
    private PointF m_Pos;
    private string m_Content;
    private float m_Size, m_Angle;
    private Color m_Color;
    private FontStyle m_Style;
     Module m=new Module();

    //字体属性
     public Font Font
     {
          get{return m_Font;}
          set{m_Font = value;}
     }

    //位置属性
     public PointF Pos
     {
          get{return m_Pos;}
          set{m_Pos = value;}
     }

    //大小属性
     public float Size
     {
          get{return m_Size;}
          set{m_Size = value;}
     }

    //内容属性
     public string Content
     {
          get{return m_Content;}
          set{m_Content = value;}
     }

    //旋转角度属性
```

```
 public float Angle
 {
        get{return m_Angle;}
        set{m_Angle = value;}
 }

//颜色属性
 new public Color Color
 {
        get{return m_Color;}
        set{m_Color = value;}
 }

//字体模式
 new public FontStyle Style
 {
        get{return m_Style;}
        set{m_Style = value;}
 }

//无参构造函数
public CText(){
      Init();
}

//构造函数，用已知的字符串进行构造
public CText(string str, PointF aPos){
      Init();
      m_Content = str;
      m_Pos = aPos;
}

//构造函数
public CText(string str, Font f,float size,
                  PointF aPos, float ang,
                  Color c, FontStyle sty){
      m_Content = str;
      m_Font = f;
      m_Size = size;
      m_Pos = aPos;
```

```
        m_Angle = ang;
        m_Color = c;
        m_Style = sty;
    }

    //构造函数，用已知的文本进行构造
    public CText(CText text)
    {
        m_Content =text.Content;
        m_Font =text.Font;
        m_Pos =text.Pos;
        m_Size =text.Size;
        m_Angle =text.Angle;
        m_Color =text.Color;
        m_Style =text.Style;
    }

    //初始化文本
    private new void Init()
    {
        m_Content = " ";
        m_Size = 20;
        m_Font = new Font("宋体", m_Size, m_Style, GraphicsUnit.Pixel);
        m_Pos = new PointF(0, 0);
        m_Angle = 0;
        m_Color = Color.Black;
        m_Style=FontStyle.Regular;
    }

    //绘文本
    override public void Draw(Graphics g,DrawMode aDrawMode)
    {
        switch(aDrawMode){
            case DrawMode.Normal:
                m_Color = Color.Black;
                    break;
            case DrawMode.Selec:
                m_Color = Color.Red;
                    break;
            case DrawMode.Delete:
```

```
                m_Color = Color.White;
                    break;
            }
            PointF Pos= m.WorldtoPage(m_Pos);
            StringFormat sf=new StringFormat(StringFormatFlags.NoWrap);
            g.DrawString(m_Content, m_Font, new SolidBrush(m_Color), Pos.X, Pos.Y, sf);
        }
```

第 6 章　交互绘图类设计

前一章创建了基本的图元类，对图元本身的属性和绘图方法进行了定义。这一章主要介绍进行交互操作的类，即描述鼠标操作过程中程序响应的类。第 4 章已经介绍了进行交互操作的接口和类之间的关系，这里不再重复。下面主要介绍接口和各个类的实现方法。

6.1　ICommand 接口

ICommand 接口有 3 种方法：LButtonDown、RButtonDown 和 MouseMove，分别定义单击鼠标左键、单击鼠标右键和移动鼠标时的绘图行为。VB.NET 和 VC#.NET 两种情况下 LButtonDown 方法的参数有所不同，后者多两个 ArrayList 对象：ges 和 geSels。这种差异与两种语言的特点有关。在 VB.NET 中可以声明全局变量，而 VC#.NET 中不能声明全局变量。由于不能声明全局变量，在 VC#.NET 的程序代码中只好传递它们。ges 保存创建的所有图元，而 geSels 保存所有被选择的图元。另外两个参数是 3 种方法都有的，g 参数是 Graphics 对象，指示绘图表面；aPos 参数是 PointF 对象，为鼠标光标的位置。

【VB.NET】

```
Public Interface ICommand
    Sub LButtonDown(ByVal g As Graphics, ByVal aPos As PointF)
    Sub RButtonDown(ByVal g As Graphics, ByVal aPos As PointF)
    Sub MouseMove(ByVal g As Graphics, ByVal aPos As PointF)
End Interface
```

【VC#.NET】

```
public interface ICommand
{
    void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels);
    void RButtonDown(Graphics g,PointF aPos);
    void MouseMove(Graphics g,PointF aPos);
}
```

6.2　CCreateLine 类

CCreateLine 类实现了 ICommand 接口，用于描述交互绘制直线段时对鼠标事件的响应。类中需要定义 3 个局部变量：m_Step 记录鼠标左键的单击次数，每单击一次，它的值就增加 1；m_Begin 和 m_End 记录直线段的起点和动态变化的终点。

首先是第 1 次单击鼠标左键的操作，将当前鼠标单击点定义为直线段起点，同时将直线

段的终点也初始化为鼠标单击点。

然后是移动鼠标时的操作，在后面 MouseMove 方法中描述。

第 2 次单击鼠标左键时，确定直线段终点的位置，它就是当前单击点 aPos 的位置。此时，需要首先从屏幕上清除最后一条橡皮线的位置，用拖曳模式重绘它就可以了。然后用正常模式绘制当前位置的直线段，并将它添加到图元集合类中去。绘制完成以后，要将 m_Step 的值归 0，准备下一条直线段的绘制。

【VB.NET】

```
'创建直线类
Public Class CCreateLine
    Implements ICommand

    Private m_Step As Integer
    Private m_Begin, m_End As PointF

    Public Sub LButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.LButtonDown
        Dim prePos As New PointF()
        '记录鼠标左键的单击次数
        m_Step += 1
        Select Case m_Step
            Case 1                      '第 1 次单击鼠标左键
                m_Begin = aPos
                m_End = aPos
            Case 2
                '重绘并删除前一位置的直线段
                Dim tempLine As New CLine(m_Begin, m_End)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing
                '绘当前位置的直线段
                m_End = aPos
                Dim newLine As New CLine(m_Begin, m_End)
                newLine.Draw(g, CGElement.geDrawMode.Normal)
                ges.Add(newLine)

                m_Step = 0
        End Select
    End Sub
```

【VC#.NET】

```
public class CCreateLine:ICommand
{
```

```
private int m_Step;
private PointF m_Begin, m_End;

public CCreateLine()
{
}

public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels){

    PointF prePos=new PointF();
    //记录鼠标左键的单击次数
        m_Step+=1;
    switch (m_Step)
    {
        case 1:
            m_Begin=aPos;
            m_End=aPos;
            break;
        case 2:
            //重绘并删除前一位置的直线段
            prePos=m_Begin;
            CLine tempLine=new CLine(m_Begin,m_End);
            tempLine.Draw(g,DrawMode.Drag);
            tempLine=null;
            //绘当前位置的直线段
            m_Begin=prePos;
            m_End=aPos;
            CLine newLine=new CLine(m_Begin,m_End);
            newLine.Draw(g,DrawMode.Normal);

            ges.Add(newLine);

            m_Step=0;
            break;
    }
}
```

第 1 次单击鼠标左键后，鼠标在绘图区移动，寻找直线段终点的位置。这个过程用橡皮线描述。绘制橡皮线需要删除前一位置的直线段，并绘制当前位置的直线段，所以要记录直线段终点在前一位置和当前位置的坐标。前一位置的坐标用 prePos 对象记录，当前位置的坐标用 curPos 对象记录。直线段终点在前一位置的坐标总是由 m_End 指定，而在当前位置的

坐标总是由鼠标移动结束时光标所处的点 aPos 指定。所以，首先要将 m_End 和 aPos 分别赋给 prePos 和 curPos。有了直线段起点坐标 m_Begin 和前一位置的终点坐标 prePos 以后，就可以暂时创建一条直线段，并用拖曳模式绘制它，实际上也就是在屏幕上删除它。由于是暂时使用的直线段，绘完图后注销它，以减少内存消耗。然后绘当前位置的直线段。由于当前直线段的终点在绘制下一位置的直线段时变成了前一位置直线段的终点，所以将 curPos 赋给 m_End。

【VB.NET】

```
Public Sub MouseMove(ByVal g As Graphics, _
        ByVal aPos As PointF) Implements ICommand.MouseMove
    Select Case m_Step
        Case 1
            Dim prePos As New PointF()
            Dim curPos As New PointF()
            prePos = m_End
            curPos = aPos
            '清除前一位置的直线段
            Dim tempLine1 As New CLine(m_Begin, prePos)
            tempLine1.Draw(g, CGElement.geDrawMode.Drag)
            tempLine1 = Nothing

            '绘当前位置的直线段
            Dim tempLine2 As New CLine(m_Begin, curPos)
            m_End = curPos
            tempLine2.Draw(g, CGElement.geDrawMode.Drag)
            tempLine2 = Nothing
    End Select
End Sub
```

【VC#.NET】

```
public void MouseMove(Graphics g,PointF aPos)
{
    switch (m_Step)
    {
        case 1:
            PointF prePos=new PointF();
            PointF curPos=new PointF();
            prePos=m_End;
            curPos=aPos;
            CLine tempLine1=new CLine(m_Begin,prePos);
            tempLine1.Draw(g,DrawMode.Drag);
            tempLine1=null;
```

```
                //绘当前位置的直线段
                CLine tempLine2=new CLine(m_Begin,curPos);
                m_End=curPos;
                tempLine2.Draw(g,DrawMode.Drag);
                tempLine2=null;

                break;
        }
    }
```

第 1 次单击鼠标左键以后，如果发现点的位置有错，或者现在不想绘制它，则通过单击鼠标右键可以取消当前操作。取消操作需要从屏幕上删除最后一条橡皮线，用拖曳模式重绘它，实现删除。完成后将 m_Step 归 0。程序如下：

【VB.NET】

```
    '单击鼠标右键时的绘图行为
    Public Sub RButtonDown(ByVal g As Graphics, _
              ByVal aPos As PointF) Implements ICommand.RButtonDown
        If m_Step = 1 Then
            '清除先前绘制的橡皮线
            Dim tempLine As New CLine(m_Begin, m_End)
            tempLine.Draw(g, CGElement.geDrawMode.Drag)
            tempLine = Nothing
        End If
        m_Step = 0
    End Sub

End Class
```

【VC#.NET】

```
    //单击鼠标右键的绘图行为
    public void RButtonDown(Graphics g,PointF aPos)
    {
        if (m_Step==1)
        {
            CLine tempLine=new CLine(m_Begin,m_End);
            tempLine.Draw(g,DrawMode.Drag);
            tempLine=null;
        }
        m_Step=0;
    }
}
```

6.3　CCreateRectangle 类

CCreateRectangle 类用于描述交互绘制矩形时对鼠标事件的响应，它实现了 ICommand 接口。与 CCreateLine 类中一样，定义了 3 个局部变量：m_Step 记录鼠标左键的单击次数，m_basePos 和 m_desPos 分别记录矩形绘制的起点和终点。

第 1 次单击鼠标左键时，单击点定义为绘制矩形的起点，同时矩形的终点也初始化为鼠标单击点。

第 2 次单击鼠标左键时，确定终点的位置。以拖曳方式重绘最后一个橡皮矩形，将它从屏幕上清除。然后用正常模式绘制当前位置的矩形，并将它添加到图元集合类中。绘制完成以后，将 m_Step 的值归 0。

【VB.NET】

```
Public Class CCreateRectangle
    Implements ICommand

    Private m_Step As Integer
    Private m_basePos, m_desPos As PointF

    Public Sub LButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.LButtonDown
        Dim prePos As New PointF()
        '记录鼠标左键的单击次数
        m_Step += 1
        Select Case m_Step
            Case 1                    '第 1 次单击鼠标左键
                m_basePos = aPos
                m_desPos = aPos
            Case 2
                '重绘并删除前一位置的橡皮矩形
                prePos = m_basePos
                Dim tempRect As New CRectangle(m_basePos, m_desPos)
                tempRect.Draw(g, CGElement.geDrawMode.Drag)
                tempRect = Nothing
                '绘当前位置的矩形
                m_basePos = prePos
                m_desPos = aPos
                Dim newRect As New CRectangle(m_basePos, m_desPos)
                newRect.Draw(g, CGElement.geDrawMode.Normal)

                ges.Add(newRect)
```

```
                m_Step = 0
            End Select
        End Sub
```

【VC#.NET】

```
public class CCreateRectangle:ICommand
{
    private int m_Step;
    private PointF m_basePos, m_desPos;

    public CCreateRectangle()
    {
    }

    public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels)
    {
        PointF prePos=new PointF();
        //记录鼠标左键的单击次数
        m_Step+=1;
        switch (m_Step)
        {
            case 1:
                m_basePos=aPos;
                m_desPos=aPos;
                break;
            case 2:
                //重绘并删除前一位置的直线段
                prePos=m_basePos;
                CRectangle tempRect=new CRectangle(m_basePos,m_desPos);
                tempRect.Draw(g,DrawMode.Drag);
                tempRect=null;
                //绘当前位置的直线段
                m_basePos=prePos;
                m_desPos=aPos;
                CRectangle newRect=new CRectangle(m_basePos,m_desPos);
                newRect.Draw(g,DrawMode.Normal);

                ges.Add(newRect);
```

```
                m_Step=0;
                break;
            }
        }
```

鼠标左键第 1 次单击以后，移动鼠标时触发窗体的 MouseMove 事件。这里用 MouseMove 方法描述移动鼠标时绘图区的响应，它实现了 ICommand 接口的 MouseMove 方法。移动鼠标时，在绘图区绘制橡皮矩形。在绘制过程中，橡皮矩形的起点是不变的，而终点，即起点的对角顶点随鼠标光标位置的变化而变化。使用拖曳绘图模式，不断清除前一位置的橡皮矩形并绘制当前位置的橡皮矩形，然后将当前矩形的终点进行记录，作为绘制下一个橡皮矩形的前一个终点，以便清除该矩形。

【VB.NET】

```
Public Sub MouseMove(ByVal g As Graphics, _
        ByVal aPos As PointF) Implements ICommand.MouseMove
    Select Case m_Step
        Case 1
            Dim prePos As New PointF()
            Dim curPos As New PointF()
            prePos = m_desPos
            curPos = aPos
            '清除前一位置的橡皮矩形
            Dim tempRect1 As New CRectangle(m_basePos, prePos)
            tempRect1.Draw(g, CGElement.geDrawMode.Drag)
            tempRect1 = Nothing

            '绘当前位置的橡皮矩形
            Dim tempRect2 As New CRectangle(m_basePos, curPos)
            m_desPos = curPos
            tempRect2.Draw(g, CGElement.geDrawMode.Drag)
            tempRect2 = Nothing
    End Select
End Sub
```

【VC#.NET】

```
public void MouseMove(Graphics g,PointF aPos)
{
    switch (m_Step)
    {
        case 1:
            PointF prePos=new PointF();
            PointF curPos=new PointF();
```

```
                prePos=m_desPos;
                curPos=aPos;
                CRectangle tempRect1=new CRectangle(m_basePos,prePos);
                tempRect1.Draw(g,DrawMode.Drag);
                tempRect1=null;

                //绘当前位置的矩形
                CRectangle tempRect2=new CRectangle(m_basePos,curPos);
                m_desPos=curPos;
                tempRect2.Draw(g,DrawMode.Drag);
                tempRect2=null;

                break;
        }
    }
```

第 1 次单击鼠标左键并移动鼠标以后，屏幕上留下一个橡皮矩形，此时如果想取消操作，可以用下面的 RButtonDown 方法实现，它实现了 ICommand 接口的 RButtonDown 方法。该方法在屏幕上清除最后一个橡皮矩形，并将 m_Step 设置为 0。

【VB.NET】

```
'单击鼠标右键时的绘图行为
Public Sub RButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.RButtonDown
    If m_Step = 1 Then
        '清除先前绘制的橡皮矩形
        Dim tempRect As New CRectangle(m_basePos, m_desPos)
        tempRect.Draw(g, CGElement.geDrawMode.Drag)
        tempRect = Nothing
    End If
    m_Step = 0
End Sub
```

【VC#.NET】

```
//单击鼠标右键的绘图行为
public void RButtonDown(Graphics g,PointF aPos)
{
    if (m_Step==1)
    {
        CRectangle tempRect=new CRectangle(m_basePos,m_desPos);
        tempRect.Draw(g,DrawMode.Drag);
        tempRect=null;
```

```
        }
        m_Step=0;
    }
```

6.4 CCreateCircle 类

CCreateCircle 类定义交互绘圆时对鼠标事件的响应，它实现了 ICommand 接口。用鼠标绘圆，需要定义圆的控制点的位置。本书采用圆心和圆上一点绘圆，所以需要确定圆心和圆上一点的位置。用鼠标交互绘圆的大体思路是：把鼠标左键的第 1 个单击点作为圆心，把第 2 个单击点作为圆上一点，两次单击之间的过程用橡皮线和橡皮圆描述，单击右键取消操作。

与 CCreateLine 类一样，CCreateCircle 类要实现 ICommand 接口的全部方法。用 m_Step 记录鼠标左键的单击次数并控制不同的绘图操作，用 m_Center 和 m_pCircle 记录圆心和圆上一点的位置。

LButtonDown 方法描述单击鼠标左键时的绘图行为。第 1 次单击鼠标左键时，将当前点作为圆心。第 2 次单击时，确定圆上一点，并删除最后一条橡皮线和最后一个橡皮圆，然后用正常模式绘当前圆。将新创建的圆添加到图元集合类中保存，以备重画和编辑之用。将 m_Step 归 0。

【VB.NET】

```
'创建圆类
Public Class CCreateCircle
    Implements ICommand

    Private m_Step As Integer
    Private m_Center, m_pCircle As PointF

    Public Sub LButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.LButtonDown
        '记录鼠标左键的单击次数
        m_Step += 1
        Select Case m_Step
            Case 1                    '第 1 次单击鼠标左键
                m_Center = aPos
                m_pCircle = aPos
            Case 2
                '清除最后一次拖动时显示的橡皮线
                Dim tempLine As New CLine(m_Center, m_pCircle)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing

                '清除最后一次拖动时显示的橡皮圆
```

```
                Dim tempCircle As New CCircle(m_Center, m_pCircle)
                tempCircle.Draw(g, CGElement.geDrawMode.Normal)
                tempCircle = Nothing

                '绘当前位置的圆
                Dim newCircle As New CCircle(m_Center, aPos)
                newCircle.Draw(g, CGElement.geDrawMode.Normal)

                ges.Add(newCircle)

                m_Step = 0
        End Select
    End Sub
```

【VC#.NET】

```
public class CCreateCircle:ICommand
{
    private int m_Step;
    private PointF m_Center, m_pCircle;

    public CCreateCircle()
    {
    }

    public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels)
    {
        //记录鼠标左键的单击次数
        m_Step+=1;
        switch (m_Step)
        {
            case 1:                 //第 1 次单击鼠标左键
                m_Center=aPos;
                m_pCircle=aPos;
                break;
            case 2:
                //清除最后一次拖动时显示的橡皮线
                CLine tempLine=new CLine(m_Center,m_pCircle);
                tempLine.Draw(g,DrawMode.Drag);
                tempLine=null;

                //清除最后一次拖动时显示的橡皮圆
```

```
                CCircle tempCircle=new CCircle(m_Center,m_pCircle);
                tempCircle.Draw(g,DrawMode.Normal);
                tempCircle=null;

                //绘当前位置的圆
                CCircle newCircle=new CCircle(m_Center,aPos);
                newCircle.Draw(g,DrawMode.Normal);

                ges.Add(newCircle);

                m_Step=0;
                break;
            }
        }
```

MouseMove 方法定义鼠标在绘图区中移动时的绘图行为。第 1 次单击鼠标左键，确定圆心的位置以后，移动鼠标，绘制橡皮线和橡皮圆。橡皮线连接圆心和鼠标光标当前点，橡皮圆为以橡皮线长度为半径的圆，橡皮线和橡皮圆随鼠标光标位置的改变而改变方向、长度或大小。在实际绘制的过程中，需要首先清除上一位置的橡皮线和橡皮圆，然后绘制当前位置的橡皮线和橡皮圆。绘制完毕以后，需要将当前点赋给 m_pCircle，因为圆上一点是动态给定的，当前位置的点在绘制下一个橡皮线和橡皮圆时它就变成了上一个点。所以需要记录它，以便绘下一条橡皮线之前删除橡皮线和橡皮圆。

【VB.NET】

```
    Public Sub MouseMove(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.MouseMove
        Select Case m_Step
            Case 1
                Dim prePos As New PointF()
                Dim curPos As New PointF()

                prePos = m_pCircle
                curPos = aPos

                '清除上一条橡皮线
                Dim tempLine1 As New CLine(m_Center, prePos)
                tempLine1.Draw(g, CGElement.geDrawMode.Drag)
                tempLine1 = Nothing

                '清除上一个橡皮圆
                Dim tempCircle1 As New CCircle(m_Center, prePos)
                tempCircle1.Draw(g, CGElement.geDrawMode.Drag)
```

```
                tempCircle1 = Nothing

                '绘当前位置的橡皮线
                Dim tempLine2 As New CLine(m_Center, curPos)
                tempLine2.Draw(g, CGElement.geDrawMode.Drag)
                tempLine2 = Nothing

                '绘当前位置的橡皮圆
                Dim tempCircle2 As New CCircle(m_Center, curPos)
                tempCircle2.Draw(g, CGElement.geDrawMode.Drag)
                tempCircle2 = Nothing

                m_pCircle = curPos

        End Select
    End Sub
```

【VC#.NET】

```
public void MouseMove(Graphics g,PointF aPos)
{
    switch (m_Step)
    {
        case 1:
            PointF prePos=new PointF();
            PointF curPos=new PointF();
            prePos=m_pCircle;
            curPos=aPos;

            //清除上一条橡皮线
            CLine tempLine1=new CLine(m_Center,prePos);
            tempLine1.Draw(g,DrawMode.Drag);
            tempLine1=null;

            //清除上一个橡皮圆
            CCircle tempCircle1=new CCircle(m_Center,prePos);
            tempCircle1.Draw(g,DrawMode.Drag);
            tempCircle1=null;

            //绘当前位置的橡皮线
            CLine tempLine2=new CLine(m_Center,curPos);
            tempLine2.Draw(g,DrawMode.Drag);
```

```
                tempLine2=null;

                //绘当前位置的橡皮圆
                CCircle tempCircle2=new CCircle(m_Center,curPos);
                tempCircle2.Draw(g,DrawMode.Drag);
                tempCircle2=null;

                m_pCircle=curPos;

                break;
        }
    }
```

第 1 次单击鼠标左键以后，绘图工作已经进行了一半，此时因为某种原因想取消操作，就用 RButtonDown 方法来实现。取消操作需要清除最后一条橡皮线和橡皮圆。然后将 m_Step 归 0。

【VB.NET】

```
    '单击鼠标右键时的绘图行为
    Public Sub RButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.RButtonDown
        If m_Step = 1 Then
            '清除橡皮线
            Dim tempLine As New CLine(m_Center, m_pCircle)
            tempLine.Draw(g, CGElement.geDrawMode.Drag)
            tempLine = Nothing
            '清除橡皮圆
            Dim tempCircle As New CCircle(m_Center, m_pCircle)
            tempCircle.Draw(g, CGElement.geDrawMode.Drag)
            tempCircle = Nothing
        End If
        m_Step = 0
    End Sub

End Class
```

【VC#.NET】

```
    //单击鼠标右键时的绘图行为
    public void RButtonDown(Graphics g,PointF aPos)
    {
        if (m_Step==1)
        {
```

```
            //清除橡皮线
            CLine tempLine=new CLine(m_Center,m_pCircle);
            tempLine.Draw(g,DrawMode.Drag);
            tempLine=null;
            //清除橡皮圆
            CCircle tempCircle=new CCircle(m_Center,m_pCircle);
            tempCircle.Draw(g,DrawMode.Drag);
            tempCircle=null;
        }
        m_Step=0;
    }
```

6.5　CCreateArc 类

CCreateArc 类描述如何交互绘制圆弧。本程序采用圆心、起点和终点三点绘制圆弧，所以用鼠标交互绘图时需要确定这 3 个点的位置。其大体思路是：前两次单击鼠标左键分别定义圆心和起点，这两点定义以后是不会变的；然后定义圆弧的终点。终点不能直接定义，因为在绘图区我们基本上不能直接获得终点的准确位置，但可以间接得到终点坐标。具体方法是根据第 3 点的方位角和由前两点得到的半径，计算出终点的准确位置。所以我们说的用圆心、起点和终点绘制圆弧从本质上讲应该是用圆心、起点和第 3 点绘制圆弧。

与前面的 CCreateLine 类和 CCreateCircle 类一样，CCreateArc 类也实现了 ICommand 接口。类中用到 4 个局部变量，m_Step 定义鼠标左键的单击次数，m_Center、m_Begin 和 m_End 分别记录圆弧的圆心、起点和终点的坐标。

LButtonDown 方法实现了 ICommand 接口的 LButtonDown 方法。第 1 次单击鼠标左键时，定义圆心的位置，将当前点的坐标赋给 m_Center。第 2 次单击鼠标左键时，确定圆弧起点的位置，并将该位置也赋给 m_End 对象，作为终点。所以，在初始情况下，圆弧的起点和终点是重合的。然后代码中用前面定义的 3 个点构造了一个暂时的 CArc 对象，并用拖曳模式绘制了该对象，然后注销。为什么要这样做，将在介绍 MouseMove 方法时介绍。第 3 次单击鼠标左键时，清除圆心到起点的橡皮线、圆心到终点的橡皮线和橡皮弧，以正常模式绘制最后生成的圆弧。将 m_Step 归 0。

【VB.NET】

```
'创建圆弧类
Public Class CCreateArc
    Implements ICommand

    Private m_Step As Integer
    Private m_Center, m_Begin, m_End As PointF

    Public Sub LButtonDown(ByVal g As Graphics, _
```

```
        ByVal aPos As PointF) Implements ICommand.LButtonDown
    '记录鼠标左键的单击次数
    m_Step += 1
    Select Case m_Step
        Case 1                  '第 1 次单击鼠标左键
            m_Center = aPos
        Case 2
            m_Begin = aPos
            m_End = aPos
            Dim tempArc1 As New CArc(m_Center, aPos, aPos)
            tempArc1.Draw(g, CGElement.geDrawMode.Drag)
            tempArc1 = Nothing
        Case 3
            '清除最后一次拖动时圆心到起点的橡皮线
            Dim tempLine1 As New CLine(m_Center, m_Begin)
            tempLine1.Draw(g, CGElement.geDrawMode.Drag)
            tempLine1 = Nothing

            '清除最后一次拖动时圆心到终点的橡皮线
            Dim tempLine2 As New CLine(m_Center, m_End)
            tempLine2.Draw(g, CGElement.geDrawMode.Drag)
            tempLine2 = Nothing

            '清除最后一次拖动时显示的橡皮弧
            Dim tempArc As New CArc(m_Center, m_Begin, m_End)
            tempArc.Draw(g, CGElement.geDrawMode.Drag)
            tempArc = Nothing

            '绘当前位置的圆弧
            Dim newArc As New CArc(m_Center, m_Begin, aPos)
            newArc.Draw(g, CGElement.geDrawMode.Normal)
            ges.Add(newArc)

            m_Step = 0
    End Select
End Sub
```

【VC#.NET】

```
public class CCreateArc:ICommand
{
    private int m_Step;
```

```
private PointF m_Center, m_Begin, m_End;

public CCreateArc()
{
}

public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels)
{
    //记录鼠标左键的单击次数
    m_Step+=1;
    switch (m_Step)
    {
        case 1:                         //第 1 次单击鼠标左键
            m_Center=aPos;
            break;
        case 2:
            m_Begin=aPos;
            m_End=aPos;
            break;
        case 3:
            //清除最后一次拖动时圆心到起点的橡皮线
            CLine tempLine1=new CLine(m_Center,m_Begin);
            tempLine1.Draw(g,DrawMode.Drag);
            tempLine1=null;

            //清除最后一次拖动时圆心到终点的橡皮线
            CLine tempLine2=new CLine(m_Center,m_End);
            tempLine2.Draw(g,DrawMode.Drag);
            tempLine2=null;

            //清除最后一次拖动时显示的橡皮弧
            CArc tempArc=new CArc(m_Center,m_Begin,m_End);
            tempArc.Draw(g,DrawMode.Normal);
            tempArc=null;

            //绘当前位置的圆弧
            CArc newArc=new CArc(m_Center,m_Begin,aPos);
            newArc.Draw(g,DrawMode.Normal);

            ges.Add(newArc);
```

```
                    m_Step=0;
                    break;
            }
        }
```

MouseMove 方法实现了 ICommand 接口的 MouseMove 方法，用于描述鼠标在绘图区移动时的绘图行为。鼠标移动时，需要不断地清除上一条圆心到终点的橡皮线、上一条橡皮弧，并绘制当前的圆心到终点的橡皮线和橡皮弧。需要注意的是，在介绍 LButtonDown 方法时说过，在初始状态下，圆弧的起点和终点是重合的。既然是重合的，那么，绘出的是一个点还是一个圆呢？读者朋友不妨测试一下，结果是一个圆。但绘制橡皮弧时，首先要清除上一个橡皮弧，由于在鼠标第 1 次移动以前并没有绘圆弧，所以没有上一个橡皮弧可供清除。结果如何呢？由于采用拖曳模式绘制，它不但没有清除，实际上还绘了一个圆。这样，后面的橡皮弧绘制将全部反向，实际上是逆时针方向绘制的，显示在屏幕上却是顺时针方向绘制。所以，在前面的 LButtonDown 方法中，在第 2 次单击以后就根据给定的 3 个点以拖曳模式绘了一个圆，等着 MouseMove 方法来清除。当前橡皮线和橡皮弧绘制完毕以后，不要忘了将当前点设置为 m_End。

【VB.NET】

```
    Public Sub MouseMove(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.MouseMove
        Select Case m_Step
            Case 2
                Dim prePos As New PointF()
                Dim curPos As New PointF()

                prePos = m_End
                curPos = aPos

                '清除上一条圆心到终点的橡皮线
                Dim tempLine1 As New CLine(m_Center, prePos)
                tempLine1.Draw(g, CGElement.geDrawMode.Drag)
                tempLine1 = Nothing

                '清除上一个橡皮弧
                Dim tempArc1 As New CArc(m_Center, m_Begin, prePos)
                tempArc1.Draw(g, CGElement.geDrawMode.Drag)
                tempArc1 = Nothing

                '绘当前圆心到终点的橡皮线
                Dim tempLine2 As New CLine(m_Center, curPos)
                tempLine2.Draw(g, CGElement.geDrawMode.Drag)
                tempLine2 = Nothing
```

```
            '绘当前位置的橡皮弧
            Dim tempArc2 As New CArc(m_Center, m_Begin, curPos)
            tempArc2.Draw(g, CGElement.geDrawMode.Drag)
            tempArc2 = Nothing

            m_End = curPos
    End Select
 End Sub
```

【VC#.NET】

```
public void MouseMove(Graphics g,PointF aPos)
{
    switch (m_Step)
    {
        case 2:
            PointF prePos=new PointF();
            PointF curPos=new PointF();
            prePos=m_End;
            curPos=aPos;

            //清除上一条圆心到终点的橡皮线
            CLine tempLine1=new CLine(m_Center,prePos);
            tempLine1.Draw(g,DrawMode.Drag);
            tempLine1=null;

            //清除上一条橡皮弧
            CArc tempArc1=new CArc(m_Center,m_Begin,m_End);
            tempArc1.Draw(g,DrawMode.Drag);
            tempArc1=null;

            //绘当前圆心到终点的橡皮线
            CLine tempLine2=new CLine(m_Center,curPos);
            tempLine2.Draw(g,DrawMode.Drag);
            tempLine2=null;

            m_End=curPos;
            break;
    }
}
```

RButtonDown 方法实现了 ICommand 接口的 RButtonDown 方法，它实现对当前未完成工作的取消。取消操作需要从屏幕上清除圆心到起点的橡皮线、圆心到终点的橡皮线和橡皮弧，并将 m_Step 的值设为 0。

【VB.NET】

```
'单击鼠标右键时的绘图行为
Public Sub RButtonDown(ByVal g As Graphics, _
        ByVal aPos As PointF) Implements ICommand.RButtonDown
    If m_Step = 2 Then
        '清除圆心到起点的橡皮线
        Dim tempLine1 As New CLine(m_Center, m_Begin)
        tempLine1.Draw(g, CGElement.geDrawMode.Drag)
        tempLine1 = Nothing

        '清除圆心到终点的橡皮线
        Dim tempLine2 As New CLine(m_Center, m_End)
        tempLine2.Draw(g, CGElement.geDrawMode.Drag)
        tempLine2 = Nothing

        '清除橡皮弧
        Dim tempArc As New CArc(m_Center, m_Begin, m_End)
        tempArc.Draw(g, CGElement.geDrawMode.Drag)
        tempArc = Nothing
    End If

    m_Step = 0
End Sub
```

【VC#.NET】

```
//单击鼠标右键时的绘图行为
public void RButtonDown(Graphics g,PointF aPos)
{
    if (m_Step==2)
    {
        //清除圆心到起点的橡皮线
        CLine tempLine1=new CLine(m_Center,m_Begin);
        tempLine1.Draw(g,DrawMode.Drag);
        tempLine1=null;

        //清除圆心到终点的橡皮线
        CLine tempLine2=new CLine(m_Center,m_End);
        tempLine2.Draw(g,DrawMode.Drag);
```

```
            tempLine2=null;

            //清除橡皮弧
            CArc tempArc=new CArc(m_Center,m_Center,m_End);
            tempArc.Draw(g,DrawMode.Drag);
            tempArc=null;
        }

        m_Step=0;
    }
```

6.6　CCreateText 类

CCreateText 类实现了 ICommand 接口。在绘图区单击鼠标左键时，如果是用 VB.NET 编程，则用一个输入对话框输入文本内容。如果是用 VC#.NET 编程，则新建一个窗体作为有模式对话框输入文本内容。用 Graphics 对象的 DrawString 方法将对话框中得到的字符串输出到绘图区，将新创建的文本添加到图元集合类中。

【VB.NET】

```
'创建文本类
Public Class CCreateText
    Implements ICommand

    Public Sub LButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.LButtonDown
        Dim strContent As String

        '利用输入对话框输入文本内容
        strContent = InputBox("请输入标注文本：")

        Dim newText As New CText(strContent, aPos)
        newText.Draw(g, CGElement.geDrawMode.Normal)

        ges.Add(newText)

    End Sub

    Public Sub MouseMove(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.MouseMove
    End Sub
```

```
    '单击鼠标右键时的绘图行为
    Public Sub RButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.RButtonDown
    End Sub

End Class
```

【VC#.NET】

```
public class CCreateText:ICommand
{
    public CCreateText()
    {
    }

    public void LButtonDown(Graphics g, PointF aPos,ArrayList ges,ArrayList geSels)
    {
        string strContent;

        Text text=new Text();
        text.ShowDialog(new Form1());
        strContent=text.SetConent();

        CText newText=new CText(strContent, aPos);
        newText.Draw(g, DrawMode.Normal);

        ges.Add(newText);
    }

    public void MouseMove(Graphics g, PointF aPos)
    {
    }

    public void RButtonDown(Graphics g, PointF aPos)
    {
    }
}
```

使用 VC#.NET 时，需要新建一个窗体，在 Form1 类中创建一个它的实例以后，将它作为有模式对话框进行调用。窗体的界面如图 6-1 所示。

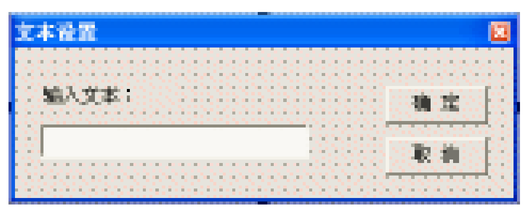

图 6-1　“文本设置”对话框

需要从“文本设置”对话框中返回要输入的文本的内容。在该窗体类中添加下面的代码：

```
private string txt;

private void button1_Click(object sender, System.EventArgs e)
{
    txt=textBox1.Text;
     this.Dispose();
}

public string SetConent()
{
     return txt;
}

private void button2_Click(object sender, System.EventArgs e)
{
     this.Dispose();
}
```

6.7　实现交互绘图

6.7.1　创建程序界面

为了测试上面我们建立的各个类，下面建立一个简单的程序界面。该界面中只有一个“绘图”菜单，其中有 4 个菜单选项，分别实现绘制直线段、圆、圆弧和文本。

首先创建一个新项目“NET_CAD”，将窗体的背景色换成白色，如图 6-2 所示。然后在工具箱中单击“MainMenu”按钮，在窗体上进行拖曳，将在窗体的顶部显示菜单条，在“请在此处输入”字样处输入菜单或菜单项的名称，创建“绘图”菜单，其程序界面如图 6-2 所示。各菜单项的名称、标注和说明如表 6-1 所示。

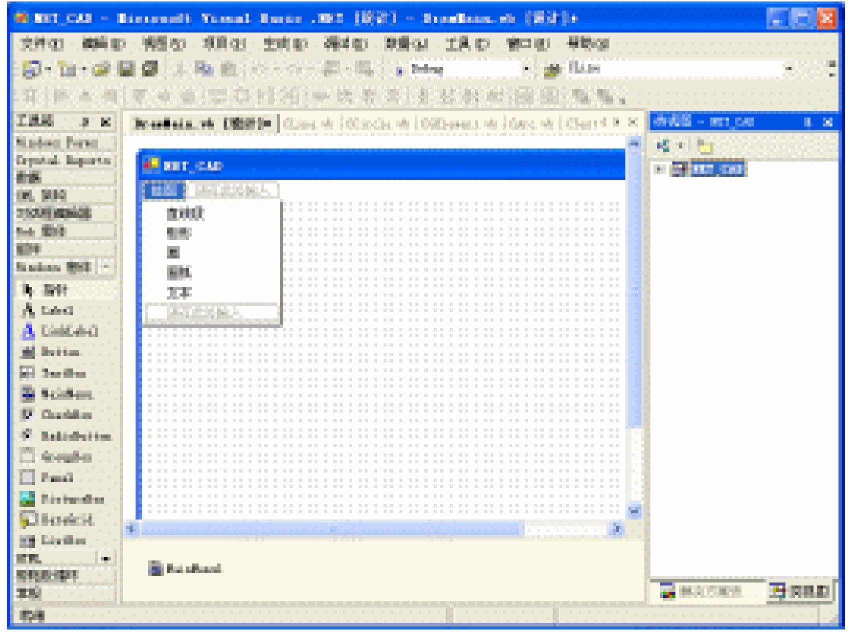

图 6-2　程序界面

表 6-1　菜单项的名称、标注和说明

名　　称	标　　注	说　　明
mnuDraw	绘图	
mnuLine	直线段	绘直线段
mnuRect	矩形	绘矩形
mnuCircle	圆	用圆心和圆上一点绘圆
mnuArc	圆弧	用圆心、起点和终点绘圆弧
mnuText	文本	绘标注文本

6.7.2　创建测试代码

前面已经创建了图元类、交互绘图类和基本的程序界面，可以说万事俱备，只欠东风了。下面创建响应菜单项事件的测试代码，它们在自定义类与前端界面之间建立联系。

在 VB.NET 和 VC#.NET 中，变量的使用范围有所不同。VC#.NET 中是没有全局变量的，所以程序中把这一类变量作为参数进行传递。如果使用 VB.NET 编程，在公共模块 Module1 中声明下面的全局变量。如果使用的是 VC#.NET，则在 Form1 类中声明它们。

【VB.NET】

```
Public viewDX, viewDY As Single
Public viewScale As Single = 1
Public PickRadius As Single = 3
Public aCommand As ICommand
Public ges As New ArrayList()
```

```
Public geSels As New ArrayList()
Public creLine As New CCreateLine()
Public creCircle As New CCreateCircle()
Public creArc As New CCreateArc()
Public creText As New CCreateText()
Private mat As New Matrix()
```

【VC#.NET】

```
private ArrayList ges=new ArrayList();
private ArrayList geSels=new ArrayList();
private ICommand aCommand;
private CCreateLine creLine=new CCreateLine();
private CCreateCircle creCircle=new CCreateCircle();
private CCreateArc creArc=new CCreateArc();
private CCreateText creText=new CCreateText();
private Module m=new Module();
```

然后在 Form1 类中添加下面的代码。其中，OnPaint 方法、OnMouseDown 方法和 OnMouseMove 方法分别重写基类中对应的方法。其他各方法为单击“绘图”菜单中各菜单项时的响应代码。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    DrawAll(g)
    DrawSel(g)
End Sub

Protected Overrides Sub OnMouseDown(ByVal e As Windows.Forms.MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics()
    Dim aPos As PointF = PagetoWorld(New PointF(e.X, e.Y))
    If e.Button = MouseButtons.Left Then
        aCommand.LButtonDown(g, aPos)
    ElseIf e.Button = MouseButtons.Right Then
        aCommand.RButtonDown(g, aPos)
    End If
End Sub

Protected Overrides Sub OnMouseMove(ByVal e As Windows.Forms.MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics()
    Dim aPos As PointF = PagetoWorld(New PointF(e.X, e.Y))
    aCommand.MouseMove(g, aPos)
```

```
End Sub

Private Sub mnuLine_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuLine.Click
    aCommand = creLine
End Sub

Private Sub mnuCircle_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuCircle.Click
    aCommand = creCircle
End Sub

Private Sub mnuArc_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuArc.Click
    aCommand = creArc
End Sub

Private Sub mnuText_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuText.Click
    aCommand = creText
End Sub
```

【VC#.NET】

```
private void mnuLine_Click(object sender, System.EventArgs e)
{
    aCommand=creLine;
}

private void menuItem4_Click(object sender, System.EventArgs e)
{
    aCommand=creCircle;
}

private void mnuArc_Click(object sender, System.EventArgs e)
{
    aCommand=creArc;
}

private void mnuText_Click(object sender, System.EventArgs e)
{
```

```
        aCommand=creText;
}

protected override void OnMouseDown(MouseEventArgs e)
{
        Graphics g=this.CreateGraphics();
        Console.WriteLine("Yes");

        PointF aPos=m.PagetoWorld(new PointF(e.X, e.Y));
        if (e.Button==MouseButtons.Left)
        {
                aCommand.LButtonDown(g,aPos,ges,geSels);
        }
        else if(e.Button==MouseButtons.Right)
        {
                aCommand.RButtonDown(g,aPos);
        }
}

protected override void OnMouseMove(MouseEventArgs e)
{
        Console.WriteLine("Yes");
        Graphics g=this.CreateGraphics();
        PointF aPos=m.PagetoWorld(new PointF(e.X, e.Y));
        aCommand.MouseMove(g,aPos);
}

private void mnuDelete_Click(object sender, System.EventArgs e)
{
        Graphics g = this.CreateGraphics();
        for (int i = 0;i<=geSels.Count - 1;i++)
        {
                ((CGElement)(geSels[i])).Draw(g, DrawMode.Delete);
                ges.Remove(geSels[i]);
        }
        geSels.RemoveRange(0, geSels.Count);
}
```

现在运行程序。图 6-3 是一个交互绘图效果的示例。

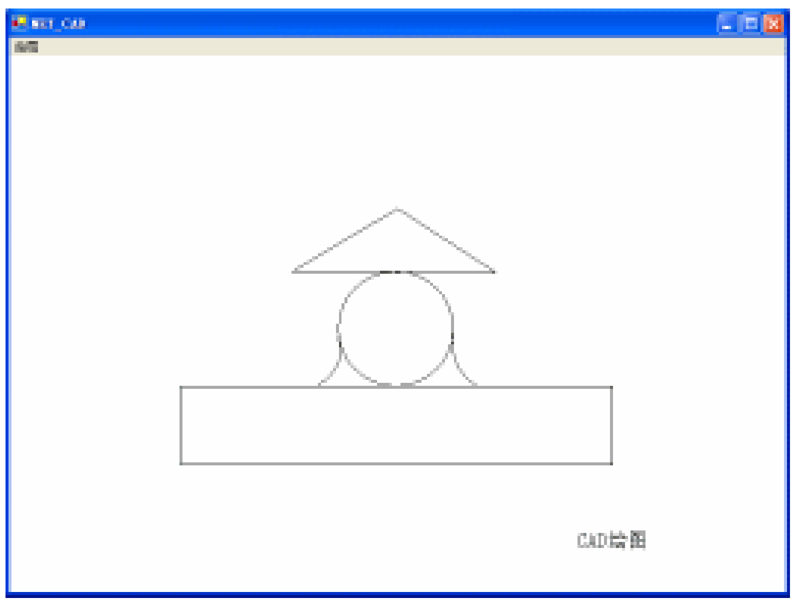

图 6-3　交互绘图示例

第 7 章　图元的编辑

图元绘制到绘图区以后，需要进行各种编辑操作。如拾取、选择、平移、旋转、镜像和比例缩放等。本章主要介绍拾取、选择和删除图元的方法。

7.1　拾取图元

拾取图元，指的是判断鼠标光标（具体地讲是鼠标光标的热点）是否位于要拾取的图元上方。就好像在商店里找一个什么东西，我们的目光就是鼠标光标的热点，这东西就是要拾取的目标图元。如果目光定位在这东西上，就说这个东西被找到了。而对于图元来讲，就说图元被拾取了。拾取和选择容易混淆，实际上是一件事情的两个阶段。就好比这东西我找到了，但不一定把它买回去。如果把它买回去，就可以说我选择了它，我要了。

拾取图元是交互式 CAD 系统中非常关键的一个部分，因为大部分的计算工作都在这部分完成。如果鼠标光标离图元足够近，就说图元被拾取了。所以拾取的过程也就是计算点与图元之间距离的过程。

进行一次拾取操作时，需要对所有图元进行遍历，即对每个图元都进行计算。所以，虽然单个图元的拾取运算比较简单，但当图形比较复杂，由成千上万个基本图元组成时，计算量就相当可观了。人们想了很多办法来减少拾取工作的计算量。目前比较通用的办法是把拾取过程分成两个阶段，首先判断拾取点是否位于图元的包围矩形中，如果不在矩形中，则图元不被拾取；如果在，则计算拾取点与图元之间的距离；如果距离足够小，则图元被拾取，否则不被拾取。由于判断点是否位于矩形中的运算量比计算点与图元之间距离的运算量要小得多,所以采用这样的两步判别法拾取图元比直接通过计算点与图元的距离来拾取要快得多。

包围矩形指的是包围图元的最小矩形。下面一节将介绍包围矩形的计算。

7.1.1　包围矩形的计算

计算包围矩形之前,首先创建一个 CBox 类,定义包围矩形的数据格式。然后在 CGElement 类中添加一个获取图元包围矩形的函数 GetBox，并在各派生类中实现它。

1．创建 CBox 类

用 X 方向上的最大值、最小值和 Y 方向上的最大值、最小值就可以确定一个矩形。在第 5 章的示例程序 NetCAD 的基础上，添加一个 Cbox 类，其代码如下：

【VB.NET】

```
Public Class CBox
    Private m_minX, m_minY As Single
    Private m_maxX, m_maxY As Single
```

```
        Public Property minX() As Single
            Get
                Return m_minX
            End Get
            Set(ByVal Value As Single)
                m_minX = Value
            End Set
        End Property

        Public Property minY() As Single
            Get
                Return m_minY
            End Get
            Set(ByVal Value As Single)
                m_minY = Value
            End Set
        End Property

        Public Property maxX() As Single
            Get
                Return m_maxX
            End Get
            Set(ByVal Value As Single)
                m_maxX = Value
            End Set
        End Property

        Public Property maxY() As Single
            Get
                Return m_maxY
            End Get
            Set(ByVal Value As Single)
                m_maxY = Value
            End Set
        End Property

    End Class
```

【VC#.NET】

```
    public class CBox
    {
```

```
    public CBox()
    {
    }

   private float m_minX, m_minY;
    private float m_maxX,m_maxY;

    public float minX
    {
        get{return m_minX;}
        set{m_minX=value;}
    }

    public float maxX
    {
        get{return m_maxX;}
        set{m_maxX=value;}
    }

    public float minY
    {
        get{return m_minY;}
        set{m_minY=value;}
    }

    public float maxY
    {
        get{return m_maxY;}
        set{m_maxY=value;}
    }
}
```

2．在 CGElement 类中添加 GetBox 方法

定义 CBox 类以后，在 CGElement 类中添加一个 GetBox 方法。该方法返回一个 CBox 对象，它是图元的包围矩形。

【VB.NET】

```
'获取图元的包围矩形
Public MustOverride Function GetBox() As CBox
```

【VC#.NET】

```
//获取图元的包围矩形
abstract public CBox GetBox();
```

3．直线段的包围矩形

直线段的包围矩形是以直线段为对角线的矩形。所以，通过比较直线段起点和终点的横坐标和纵坐标的大小就可以确定包围矩形的参数。但是，还要考虑两个特殊情况，即直线段水平或竖直的情况。在这两种情况下，按照上面定义得到的矩形就是直线段本身。所以，需要在竖向上或横向上扩大拾取范围。

在 CLine 类中实现 GetBox 方法。

【VB.NET】

```
'计算包围矩形
Public Overrides Function GetBox() As CBox
    Dim aBox As New CBox()
    '分竖直、水平和倾斜 3 种情况计算直线段的包围矩形
    If m_Begin.X = m_End.X Then
        aBox.minX = m_Begin.X - PickRadius
        aBox.minY = Min(m_Begin.Y, m_End.Y)
        aBox.maxX = m_Begin.X + PickRadius
        aBox.maxY = Max(m_Begin.Y, m_End.Y)
    ElseIf m_Begin.Y = m_End.Y Then
        aBox.minX = Min(m_Begin.X, m_End.X)
        aBox.minY = m_Begin.Y - PickRadius
        aBox.maxX = Max(m_Begin.X, m_End.X)
        aBox.maxY = m_Begin.Y + PickRadius
    Else
        aBox.minX = Min(m_Begin.X, m_End.X)
        aBox.minY = Min(m_Begin.Y, m_End.Y)
        aBox.maxX = Max(m_Begin.X, m_End.X)
        aBox.maxY = Max(m_Begin.Y, m_End.Y)
    End If
    Return aBox
End Function
```

【VC#.NET】

```
//计算包围矩形
override public CBox GetBox()
{
    CBox aBox=new CBox();
    if (m_Begin.X==m_End.X)
    {
        aBox.minX=m_Begin.X-Const.PickRadius;
        aBox.minY=Math.Min(m_Begin.Y,m_End.Y);
        aBox.maxX=m_Begin.X+Const.PickRadius;
```

```
            aBox.maxY=Math.Max(m_Begin.Y,m_End.Y);
        }
        else if (m_Begin.Y==m_End.Y)
        {
            aBox.minX=Math.Min(m_Begin.X,m_End.X);
            aBox.minY=m_Begin.Y-Const.PickRadius;
            aBox.maxX=Math.Max(m_Begin.X,m_End.X);
            aBox.maxY=m_Begin.Y+Const.PickRadius;
        }
        else
        {
            aBox.minX=Math.Min(m_Begin.X,m_End.X);
            aBox.minY=Math.Min(m_Begin.Y,m_End.Y);
            aBox.maxX=Math.Max(m_Begin.X,m_End.X);
            aBox.maxY=Math.Max(m_Begin.Y,m_End.Y);
        }

        return aBox;
    }
```

4．矩形的包围矩形

矩形的包围矩形就是它本身。在 CRectangle 类中添加一个 GetBox 方法，它重写基类的 GetBox 方法并返回一个 CBox 类型的对象。包围矩形的属性值用 CRectangle 对象左上角顶点和右下角顶点的坐标进行设置。

【VB.NET】

```
'计算包围矩形
Public Overrides Function GetBox() As CBox
    Dim aBox As New CBox()
    With aBox
        .minX = Me.LT.X
        .minY = Me.RB.Y
        .maxX = Me.RB.X
        .maxY = Me.LT.Y
    End With
    Return aBox
End Function
```

【VC#.NET】

```
//计算包围矩形
override public CBox GetBox()
{
```

```
        CBox aBox=new CBox();
        aBox.minX = this.LT.X;
        aBox.minY = this.RB.Y;
        aBox.maxX = this.RB.X;
        aBox.maxY = this.LT.Y;

        return aBox;
    }
```

5．圆的包围矩形

圆的包围矩形是中心为圆心，边长为圆的直径的正方形。在 CCircle 类中重写基类的 GetBox 方法：

【VB.NET】

```
'计算圆的包围矩形
Public Overrides Function GetBox() As CBox
    Dim aBox As New CBox()
    With aBox
        .minX = m_Center.X - Radius
        .minY = m_Center.Y - Radius
        .maxX = m_Center.X + Radius
        .maxY = m_Center.Y + Radius
    End With
    Return aBox
End Function
```

【VC#.NET】

```
//计算圆的包围矩形
override public CBox GetBox()
{
    CBox aBox=new CBox();
    aBox .minX=m_Center.X-Radius;
    aBox .minY=m_Center.Y-Radius;
    aBox .maxX=m_Center.X+Radius;
    aBox .maxY=m_Center.Y+Radius;

    return aBox;
}
```

6．圆弧的包围矩形

圆弧的包围矩形如图 7-1 所示。计算圆弧的包围矩形时，首先假设有一个局部坐标系，它的原点在圆弧圆心的位置上，然后计算圆弧与坐标轴 4 个方向上的相交关系。为什么要得

到这些相交关系呢？因为圆弧与坐标轴正向或负向的交点坐标在数值上代表了这个方向上的最大值。如果相交，则包围矩形在这个方向上的参数值便是该值。如果不相交，最大值就是圆弧起点和终点 X、Y 坐标的值中在这个方向上的最大值。

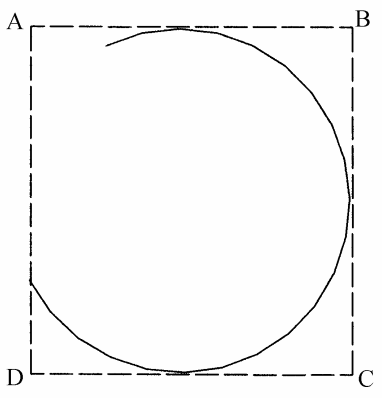

图 7-1　圆弧的包围矩形

圆弧与 4 个轴的相交关系可以通过比较可能的交点与圆弧之间的关系得到。如果圆弧所在的圆与坐标轴某个方向上的交点在圆弧内，则该交点便是圆弧与坐标轴的交点，否则不是。

计算圆弧的包围矩形时，需要比较拾取点相对于圆心的方位角与圆弧起点和终点的方位角之间的关系。下面的程序中用了一个私有的 InArc 函数来实现。如果拾取点相对于圆心的方位角位于圆弧起点和终点的方位角之间，则返回 True，否则返回 False。这里需要考虑一个问题，那就是缺省时系统以逆时针方向画弧。

在 CArc 类中重写基类的 GetBox 方法：

【VB.NET】

```
'计算圆弧的包围矩形
Public Overrides Function GetBox() As CBox
    Dim x1, y1, x2, y2 As Single
    Dim i As Integer
    Dim aBox As New CBox()
    With m_Begin
        x1 = Min(.X, m_End.X)
        y1 = Min(.Y, m_End.Y)
        x2 = Max(.X, m_End.X)
        y2 = Max(.Y, m_End.Y)
    End With
    For i = 0 To 3
        If (InArc(AngleBegin, AngleEnd, PI / 2 * i)) Then
            '如果圆弧与 X 轴正向相交
            If i = 0 Then
                x2 = m_Center.X + Radius
                '如果圆弧与 Y 轴正向相交
```

```
            ElseIf i = 1 Then
                y2 = m_Center.Y + Radius
                '如果圆弧与 X 轴负向相交
            ElseIf i = 2 Then
                x1 = m_Center.X - Radius
                '如果圆弧与 Y 轴负向相交
            ElseIf i = 3 Then
                y1 = m_Center.Y - Radius
            End If
        End If
    Next i

    With aBox
        .minX = x1
        .minY = y1
        .maxX = x2
        .maxY = y2
    End With

    Return aBox
End Function

'判断角度 Angle 对应的点是否在 Angle1 至 Angle2 的圆弧上
Private Function InArc(ByVal Angle1 As Single,  _
       ByVal Angle2 As Single, ByVal Angle As Single) As Boolean
    '如果起始角小于终止角
    If Angle1 < Angle2 Then
        '如果方向角在起始角与终止角之间，则返回 True，
        '否则返回 False
        If Angle >= Angle1 And Angle <= Angle2 Then
            Return True
        Else
            Return False
        End If
        '如果起始角小于终止角
    Else
        '如果方向角大于起始角或小于终止角，则返回 True，
        '否则返回 False
        If Angle >= Angle1 Or Angle <= Angle2 Then
            Return True
        Else
```

```
            Return False
        End If
    End If
End Function
```

【VC#.NET】

```
//计算圆弧的包围矩形
override public CBox GetBox(){
    float x1=Math.Min(m_Begin.X,m_End.X);
    float y1=Math.Min(m_Begin.Y,m_End.Y);
    float x2=Math.Max(m_Begin.X,m_End.X);
    float y2=Math.Max(m_Begin.Y,m_End.Y);
    for (int i=0;i<4;i++)
    {
        if (InArc(AngleBegin,AngleEnd,Const.PI/2*i))
        {
            //如果圆弧与 X 轴正向相交
                if(i==0) { x2=m_Center.X+Radius;}
            //如果圆弧与 Y 轴正向相交
                if(i==1) { y2=m_Center.Y+Radius;}
            //如果圆弧与 X 轴负向相交
                if(i==2) {x1=m_Center.X-Radius;}
            //如果圆弧与 Y 轴负向相交
                if(i==3) { y1=m_Center.Y-Radius;}
        }
    }

    CBox aBox=new CBox();
    aBox.minX=x1;
    aBox.minY=y1;
    aBox.maxX=x2;
    aBox.maxY=y2;

    return aBox;
}

//判断角度 Angle 对应的点是否在 Angle1 至 Angle2 的圆弧上
private bool InArc(float Angle1,float Angle2,float Angle)
{
    //如果起始角小于终止角
    if (Angle1<Angle2)
```

```
        {
            //如果方向角在起始角与终止角之间，则返回 true；
            //否则返回 false
            if (Angle>=Angle1 && Angle<=Angle2)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        //如果起始角小于终止角
        else
        {
            //如果方向角大于起始角或小于终止角，则返回 true
            //否则返回 false
            if (Angle>=Angle1 || Angle<=Angle2)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

7．文本的包围矩形

根据文本的插入点、文本的宽度和高度，可以获得文本的包围矩形。下面我们没有采用这种方法，而是采用了 GDI+的一些方法。首先将文本添加到一个 GraphicsPath 对象中，然后利用该对象的 GetBounds 方法获得路径的包围矩形，该包围矩形也就是文本的包围矩形。

在 CText 类中重写基类的 GetBox 方法：

【VB.NET】

```
'计算文本的包围矩形
Public Overrides Function GetBox() As CBox
    Dim aBox As New CBox()
    Dim gp As New GraphicsPath()
    gp.AddString(m_Content, m_FontFamily, m_Style, m_Size, m_Pos, m_Format)
    Dim rect As RectangleF = gp.GetBounds()
```

```
        With aBox
            .minX = rect.Left
            .minY = rect.Top - rect.Height
            .maxX = rect.Right
            .maxY = rect.Bottom - rect.Height
        End With
        gp.Dispose()
        Return aBox
    End Function
```

【VC#.NET】

```
    //计算文本的包围矩形
    override public CBox GetBox()
    {
         CBox aBox=new CBox();
         GraphicsPath gp=new GraphicsPath();
         FontFamily fm=new FontFamily("宋体");
         StringFormat sf=new StringFormat(StringFormatFlags.NoWrap);
         int i=2;
         switch (m_Style)
         {
             case FontStyle.Bold:
                 i=0;
                 break;
             case FontStyle.Italic:
                 i=1;
                 break;
             case FontStyle.Regular :
                 i=2;
                 break;
             case FontStyle.Strikeout :
                 i=3;
                 break;
             case FontStyle.Underline :
                 i=4;
                 break;
         }
         gp.AddString(m_Content, fm, i, m_Size, m_Pos, sf);
         RectangleF rect= new RectangleF(0,0,0,0);
         rect=gp.GetBounds();
         aBox.minX = rect.Left;
```

```
        aBox.minY = rect.Top - rect.Height;
        aBox.maxX = rect.Right;
        aBox.maxY = rect.Bottom - rect.Height;
        gp.Dispose();
        return aBox;
    }
```

7.1.2　拾取图元

前面说了，如果拾取点位于包围矩形中，则继续计算拾取点与图元之间的距离。如果距离足够小，则图元被拾取，否则不被拾取。在图元拾取的计算中，有 3 种计算是最常用的，即点与点之间的距离计算、点与直线段之间距离的计算和点相对于另一点的方位角的计算。为了提高拾取效率，常常采用距离的近似算法。比如点与点之间的距离计算和点与直线段之间的距离计算都涉及到开方运算，比较费时。所以，常常采用近似计算，把它转化为四则运算。

实现图元拾取，需要在 CGElement 类中添加一个 Pick 方法，然后在各派生类中实现它。

1．在 CGElement 类中添加 Pick 方法

在 CGElement 类中添加一个必须重写的 Pick 方法。该方法有两个参数，一个是 PointF 对象，表示拾取点；另一个是单精度值，表示拾取半径。返回一个布尔值。

【VB.NET】

```
'拾取图元
Public MustOverride Function Pick(ByVal aPos As PointF) As Boolean
```

【VC#.NET】

```
//拾取图元
abstract public bool Pick(PointF aPos);
```

2．直线段的拾取

拾取直线段，需要知道拾取点与直线段之间的距离。CAD 应用中常采用近似计算。一种方案是用拾取点到直线段的水平距离和竖直距离之间的小者来代替真实距离。

在 CLine 类中添加一个计算点到直线段距离的私有函数 distPtoL 和一个计算直线段截距式方程参数的私有函数 LineKX，然后重写基类的 Pick 方法：

【VB.NET】

```
'计算点到直线段的距离
Private Function distPtoL(ByVal aPos As PointF, ByVal pB As PointF, _
    ByVal pE As PointF) As Single
    Dim kc(1) As Single
    Dim px As Single = aPos.X
    Dim py As Single = aPos.Y
    Dim distX, distY, dist As Single
```

```
        '获取直线段的截距式方程，返回斜率和截距
        kc = LineKX(pB, pE)
        '如果为水平直线段
        If kc(0) = 0 Then
            distX = 10000
            distY = Abs(py - pB.Y)
            '如果为竖直直线段
        ElseIf kc(0) = 10000 Then
            distX = Abs(px - pB.X)
            distY = 10000
            '如果为斜线
        Else
            distX = Abs(px - (py - kc(1)) / kc(0))
            distY = Abs(py - (kc(0) * px + kc(1)))
        End If
        '返回水平距离和竖直距离之间的小值
        dist = Min(distX, distY)
        Return dist
    End Function

    '计算直线段的截距式方程
    Private Function LineKX(ByVal pB As PointF, _
                ByVal pE As PointF) As Single()
        Dim kc(1) As Single
        '若直线段不为竖直线段
        If pB.X <> pE.X Then
            kc(0) = (pE.Y - pB.Y) / (pE.X - pB.X)
            '如果是竖直线段
        Else
            kc(0) = 10000
        End If
        '计算截距
        kc(1) = pB.Y - kc(0) * pB.X
        Return kc
    End Function

    '拾取直线段
    Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
        Dim geBox As New CBox()
```

```
        '判断拾取点是否在测试包围矩形中，若不是，
        '则直线段不被拾取
        If (Not InBox(GetBox, aPos)) Then
            Return False
        Else
            '如果拾取点位于包围矩形中，且到直线段的距离小于拾取半径，
            '则直线段被拾取；否则不被拾取
            If distPtoL(aPos, m_Begin, m_End) <= PickRadius Then
                Return True
            Else
                Return False
            End If
        End If
    End Function
```

【VC#.NET】

```
    //计算点到直线段的距离
    private float distPtoL(PointF aPos,CLine aLine)
    {
        float px,py,dist;
        float distX,distY;
        float[] kc;

        px=aPos.X;
        py=aPos.Y;

        //获取直线段的截距式方程，返回斜率和截距
        kc=aLine.LineKX();
        //如果为水平直线段
        if (kc[0]==0)
        {
            distX=10000;
            distY=Math.Abs(py-aLine.LBegin.Y);
        }
        //如果为竖直直线段
        else if (kc[0]==10000)
        {
            distX=Math.Abs(px-aLine.LBegin.X);
            distY=10000;
        }
        //如果为斜线
        else
```

```
        {
            distX=Math.Abs(px- (py-kc[1])/kc[0]);
            distY=Math.Abs(py- (kc[0]*px+kc[1]));
        }
        //返回水平距离和竖直距离之间的小值
        dist=Math.Min(distX,distY);
        return dist;
    }

    //计算直线段的截距式方程
    private float[] LineKX()
    {
        float kc0,kc1;
        //若直线段不为竖直线段
        if (m_Begin.X!=m_End.X)
        {
            kc0=(m_End.Y-m_Begin.Y)/(m_End.X-m_Begin.X);
        }
        //如果是竖直线段
        else
        {
            kc0=10000f;
        }
        //计算截距
        kc1=m_Begin.Y-kc0*m_Begin.X;

        float[] kc={kc0,kc1};
        return kc;
    }

    //拾取直线段
    override public bool Pick(PointF aPos)
    {
        CBox geBox=new CBox();

        //判断拾取点是否在测试包围矩形中，若不是，
        //则直线段不被拾取
        if (!(m.InBox(GetBox(),aPos)))
        {
            return false;
        }
```

```
        else
        {
            if (distPtoL(aPos,this)<Const.PickRadius)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

3．矩形的拾取

拾取矩形的问题可以转化为拾取直线段的问题，当代表矩形任意一条边的直线段被拾取时，矩形被拾取。在 CRectangle 类中添加一个 Pick 方法，它重写了基类的 Pick 方法。代码中，首先创建了 4 个 CLine 类实例，并分别设置为代表矩形 4 条边的直线段，然后给出一个 If 语句，当至少有一条直线段被拾取时，整个矩形被拾取。

【VB.NET】

```
'拾取矩形
Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
    '首先把要拾取的矩形分解为 4 条直线段
    Dim line0 As New CLine()
    Dim line1 As New CLine()
    Dim line2 As New CLine()
    Dim line3 As New CLine()
    line0 = New CLine(Me.LT, New PointF(Me.LT.X, Me.RB.Y))
    line1 = New CLine(New PointF(Me.LT.X, Me.RB.Y), Me.RB)
    line2 = New CLine(Me.RB, New PointF(Me.RB.X, Me.LT.Y))
    line3 = New CLine(New PointF(Me.RB.X, Me.LT.Y), Me.LT)

    '如果有一条直线段被拾取，则整个矩形被拾取
    If line0.Pick(aPos) Or _
        line1.Pick(aPos) Or _
        line2.Pick(aPos) Or _
        line3.Pick(aPos) Then
        Return True
    Else
        Return False
    End If
End Function
```

【VC#.NET】

```
//拾取矩形
override public bool Pick(PointF aPos)
{
    //首先把要拾取的矩形分解为 4 条直线段
    CLine line0=new CLine();
    CLine line1=new CLine();
    CLine line2=new CLine();
    CLine line3=new CLine();
    line0 = new CLine(this.LT, new PointF(this.LT.X, this.RB.Y));
    line1 = new CLine(new PointF(this.LT.X, this.RB.Y), this.RB);
    line2 = new CLine(this.RB, new PointF(this.RB.X, this.LT.Y));
    line3 = new CLine(new PointF(this.RB.X, this.LT.Y), this.LT);

    //如果有一条直线段被拾取，则整个矩形被拾取
    if (line0.Pick(aPos) ||
        line1.Pick(aPos) ||
        line2.Pick(aPos) ||
        line3.Pick(aPos))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

4．圆的拾取

假设拾取半径为 PickRadius，要拾取的圆的半径为 R，则以该圆的圆心为圆心，以 R-PickRadius 和 R+PickRadius 为半径得到一个环带，当拾取点落在该环带内时，圆被拾取。

在 CCircle 类中重写基类的 Pick 方法：

【VB.NET】

```
'拾取圆
Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
    Dim dist As Single
    '如果拾取点不在包围矩形中，则该圆不被拾取
    If (Not InBox(GetBox, aPos)) Then
        Return False
    Else
```

```
            '如果拾取点在包围矩形中，且到圆心的距离在一定的范围内，
            '则圆被拾取，否则不被拾取
            dist = DistPtoP(aPos, m_Center)
            If ((dist > Radius - PickRadius) And _
              (dist < Radius + PickRadius)) Then
                Return True
            Else
                Return False
            End If
        End If
    End Function
```

【VC#.NET】

```
    //拾取圆
    override public bool Pick(PointF aPos)
    {
        float dist;
        if (!(m.InBox(GetBox(),aPos)))
        {
            return false;
        }
        else
        {
            dist=m.DistPtoP(aPos,m_Center);
            if ((dist>Radius-Const.PickRadius) && (dist<Radius+Const.PickRadius))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

5．圆弧的拾取

判断圆弧是否被拾取，需要首先判断拾取点是否落在圆弧所在的圆上；如果在圆上，则继续判断拾取点相对于圆心的方位角是否位于圆弧起点和终点的方位角之间；如果在，则圆弧被拾取，否则不被拾取。编程时需要注意，缺省时 GDI+按逆时针方向绘圆弧。

在 CArc 类中重写基类的 Pick 方法：

【VB.NET】

```
'拾取圆弧
Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
    Dim angle As Single
    Dim dist As Single

    '如果拾取点不在包围矩形中，则该圆弧不被拾取
    If Not InBox(GetBox(), aPos) Then
        Return False
        '否则，进一步判断
    Else
        '计算拾取点与圆心之间的距离
        dist = DistPtoP(aPos, m_Center)
        '计算拾取点的方位角
        angle = GetAngle(m_Center, aPos)

        '如果起始角小于终止角
        If AngleBegin < AngleEnd Then
            '如果拾取点的方位角界于起始角和终止角之间，
            '则拾取点到圆心的距离与圆弧的半径接近，
            '则圆弧被拾取，否则不被拾取
            If (angle >= AngleBegin And angle <= AngleEnd) _
              And Abs(Radius - dist) <= PickRadius Then
                Return True
            Else
                Return False
            End If
            '如果起始角大于终止角
        Else
            '如果拾取点的方位角大于等于起始角或小于等于终止角
            '且拾取点到圆心的距离与圆弧的半径接近，
            '则该圆弧被拾取；否则不被拾取
            If (angle >= AngleBegin Or angle <= AngleEnd) And _
              Abs(Radius - dist) <= PickRadius Then
                Return True
            Else
                Return False
            End If
        End If
    End If
End Function
```

【VC#.NET】

```
//拾取圆弧
override public bool Pick(PointF aPos)
{
    //如果拾取点不在包围矩形中，则该圆弧不被拾取
    if (!(m.InBox(GetBox(),aPos)))
    {
        return false;
    }
    else
    {
        //计算拾取点与圆心之间的距离
        float dist=m.DistPtoP(aPos,m_Center);
        //计算拾取点的方位角
        float angle=m.GetAngle(m_Center,aPos);

        //如果起始角小于终止角
        if (AngleBegin<AngleEnd)
        {
            //如果拾取点的方位角界于起始角和终止角之间，
            //则拾取点到圆心的距离与圆弧的半径接近，
            //则圆弧被拾取；否则不被拾取
            if ((angle>=AngleBegin && angle<=AngleEnd) &&
                (Math.Abs(Radius-dist)<=Const.PickRadius))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        //如果起始角大于终止角
        else
        {
            //如果拾取点的方位角大于等于起始角或小于等于终止角
            //且拾取点到圆心的距离与圆弧的半径接近，
            //则该圆弧被拾取；否则不被拾取
            if ((angle>=AngleBegin || angle<=AngleEnd) &&
                (Math.Abs(Radius-dist)<=Const.PickRadius))
```

```
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
    }
```

6．文本的拾取

拾取文本比较简单，当拾取点位于文本的包围矩形中时，文本被拾取。在 CText **类中重写基类的** Pick **方法。**

【VB.NET】

```
 '拾取文本
Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
    If InBox(GetBox(), aPos) Then
        Return True
    Else
        Return False
    End If
End Function
```

【VC#.NET】

```
//拾取文本
override public bool Pick(PointF aPos)
{
    if (m.InBox(GetBox(), aPos))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

7.2　选择图元

把图元放到单独的选择集中的过程称为选择图元。图元被拾取到以后，如果希望对它进

行某种后续操作，可以先把它放到一个单独的集合类里面，后面的特定操作如几何变换、删除等操作将只针对该集合类中的图元进行。选择图元的方法很多，常见的有鼠标单选、线选、窗选、多边形选择等。被选择的图元可以用不同颜色或线型显示。

7.2.1　添加菜单资源

选择图元，需要首先添加相应的菜单资源，如图 7-2 所示。在第 5 章和第 6 章创建的 CAD 程序的基础上，在窗体主菜单中添加一个“编辑”菜单。菜单中主要包括 4 个选项，即“选择”、“全选”、“放弃选择”和“删除”，分别实现图元的逐个选择、全选、放弃选择和删除。各选项的名称分别为 mnuSelect, mnuSelAll, mnuDeSel 和 mnuDelete。

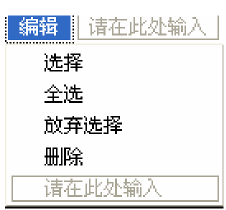

图 7-2　在主菜单中添加“编辑”菜单

7.2.2　鼠标单选

为了实现鼠标单选，首先创建一个 CSelect 类。该类实现了 ICommand 接口。当鼠标左键按下时，如果鼠标下方的图元被拾取，则该图元被添加到选择集 geSels 中。

【VB.NET】

```
Public Class CSelect
    Implements ICommand

    '单击鼠标左键时的绘图行为
    Public Sub LButtonDown(ByVal g As System.Drawing.Graphics, _
            ByVal aPos As System.Drawing.PointF) _
            Implements ICommand.LButtonDown
        Dim i As Integer
        For i = 0 To ges.Count - 1
            '如果图元被拾取，则用选择模式绘制图元
            '并将该图元添加到选择集中
            If (ges(i).Pick(aPos, PickRadius) = True) Then
                ges(i).Draw(g, CGElement.geDrawMode.Selec)
                geSels.Add(ges(i))
            End If
        Next
    End Sub

    Public Sub MouseMove(ByVal g As System.Drawing.Graphics, _
```

```
                ByVal aPos As System.Drawing.PointF) _
                Implements ICommand.MouseMove
        End Sub

        Public Sub RButtonDown(ByVal g As System.Drawing.Graphics, _
                ByVal aPos As System.Drawing.PointF) _
                Implements ICommand.RButtonDown
        End Sub
    End Class
```

【VC#.NET】

```
    public class CSelect:ICommand
    {
        public CSelect()
        {
        }

        //单击鼠标左键时的绘图行为
       public void LButtonDown(Graphics g, PointF aPos,ArrayList ges,ArrayList geSels)
       {
            CLine line=new CLine();
            CCircle circle=new CCircle();
            CArc arc=new CArc();
            CText txt=new CText();
            for(int i = 0;i<ges.Count;i++){
                //如果图元被拾取，则用选择模式绘制图元
                //并将该图元添加到选择集中
                if (((CGElement)(ges[i])).Pick(aPos))
                {
                    ((CGElement)(ges[i])).Draw(g, DrawMode.Selec);
                    geSels.Add(ges[i]);
                }
            }
        }

        public void MouseMove(Graphics g, PointF aPos)
        {
        }

        public void RButtonDown(Graphics g, PointF aPos)
        {
        }
    }
```

创建 CSelect 类以后，如果使用的是 VB.NET，首先在 Module1 模块中创建一个 CSelect 类的实例 selected，即

```
Public selected As New CSelect()
```

如果使用的是 VC#.NET，首先在 Form1 类中创建 CSelect 类实例 sel：

```
private CSelect sel=new CSelect();
```

然后在 Form1 类中添加下面的代码：

【VB.NET】

```
Private Sub mnuSelect_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles sel.Click
    aCommand = selected
End Sub
```

【VC#.NET】

```
private void mnuSelect_Click(object sender, System.EventArgs e)
{
    aCommand=sel;
}
```

这样，在“编辑”菜单中单击菜单项“选择”以后，可以用鼠标单选图元。单选图元的效果如图 7-3 所示。

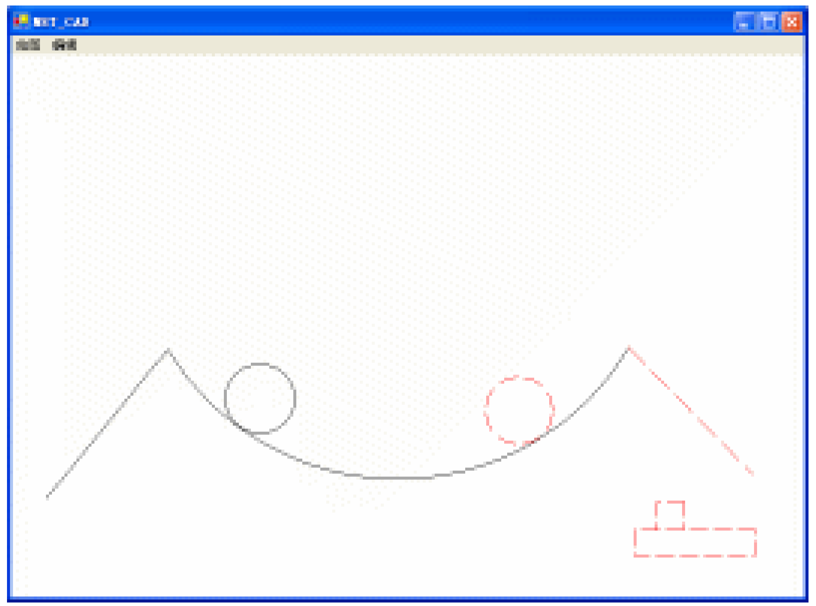

图 7-3　单选图元

7.2.3　全选

全选图元需要将所有图元都放到选择集中去，并且把所有图元都以选择模式重画。

【VB.NET】

```
Private Sub selAll_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles selAll.Click
    Dim g As Graphics = Me.CreateGraphics
    Dim i As Integer
    For i = 0 To ges.Count - 1
        geSels.Add(ges(i))
    Next
    DrawSel(g)
End Sub
```

【VC#.NET】

```
private void mnuSelAll_Click(object sender, System.EventArgs e)
{
    Graphics   g= this.CreateGraphics();
    for (int i = 0;i<ges.Count;i++)
    {
        geSels.Add(ges[i]);
    }
    m.DrawSel(g,geSels);
}
```

上面的代码中，DrawSel 过程将选择集中的所有图元用选择模式重画。

【VB.NET】

```
Public Sub DrawSel(ByVal g As Graphics)
    Dim i As Integer
    For i = 0 To geSels.Count - 1
        geSels(i).draw(g, geDrawMode.Selec)
    Next
End Sub
```

【VC#.NET】

```
public void DrawSel(Graphics g,ArrayList geSels){
    for(int i = 0;i<geSels.Count;i++)
    {
        CGElement ge=(CGElement)(geSels[i]);
        ge.Draw(g, DrawMode.Selec);
    }
}
```

运行程序以后，在“编辑”菜单中单击“全选”菜单项，选择当前绘图区中的所有图元。全选效果如图 7-4 所示。

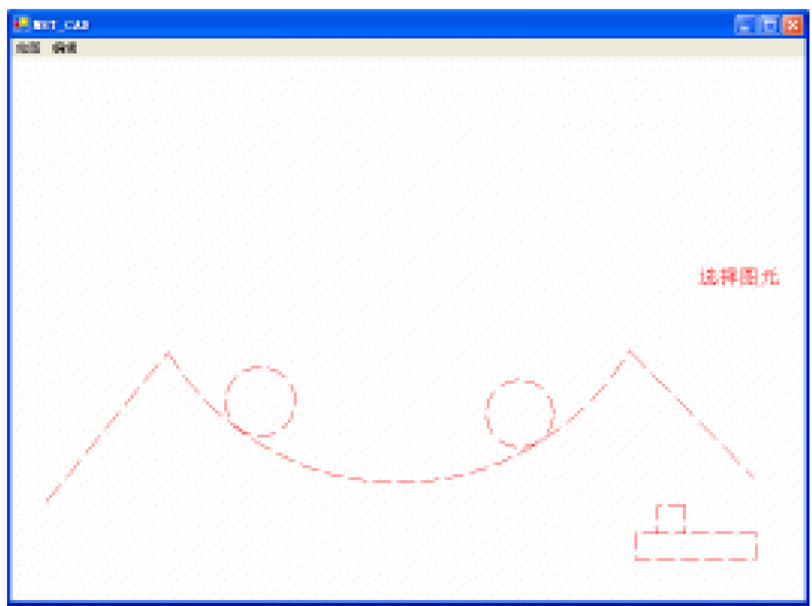

图 7-4　全选图元

7.2.4　放弃选择

放弃选择操作将选择集中的图元以正常模式重绘，然后清空选择集。

【VB.NET】

```
Private Sub mnuDesel_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles desel.Click
    Dim g As Graphics = Me.CreateGraphics
    Dim i As Integer
    For i = 0 To geSels.Count － 1
        geSels(i).Draw(g, geDrawMode.Normal)
    Next i
    geSels.RemoveRange(0, geSels.Count)
End Sub
```

【VC#.NET】

```
private void mnuDesel_Click(object sender, System.EventArgs e)
{
    Graphics   g= this.CreateGraphics();
    for (int i = 0;i<geSels.Count;i++)
    {
        ((CGElement)(geSels[i])).Draw(g, DrawMode.Normal );
    }
    geSels.RemoveRange(0, geSels.Count);
}
```

添加上面的代码以后，运行程序，在“编辑”菜单中单击“放弃选择”选项，当前绘图区中被选中的图形恢复到正常模式的绘制状态。

7.3 删除图元

删除图元操作是针对选择集中的图元的，它需要完成下面的操作：

- 将选择集中的图元从图元集合类中删除；
- 清空选择集；
- 重画图元，将删除的图元从屏幕上删除。

【VB.NET】

```
Private Sub MenuItem19_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles tdelete.Click
    Dim g As Graphics = Me.CreateGraphics
    Dim i As Integer
    For i = 0 To geSels.Count - 1
        geSels(i).Draw(g, geDrawMode.Delete)
        ges.Remove(geSels(i))
    Next i
    geSels.RemoveRange(0, geSels.Count)
End Sub
```

【VC#.NET】

```
private void mnuDelete_Click(object sender, System.EventArgs e)
{
    Graphics g = this.CreateGraphics();
    for (int i = 0;i<=geSels.Count - 1;i++)
    {
        ((CGElement)(geSels[i])).Draw(g, DrawMode.Delete);
        ges.Remove(geSels[i]);
    }
    geSels.RemoveRange(0, geSels.Count);
}
```

添加代码以后，运行程序，单击“选择”菜单中的“删除”菜单项，当前被选中的图形将从内存中和屏幕上被删除掉。

第 8 章　图 元 变 换

采用变换，可以提高绘图的效率和准确性。本章主要介绍对图元进行平移、旋转、镜像和比例缩放等几何变换操作的技巧。图元变换只针对选择集中的图元。在交互式 CAD 系统中，关键的问题是找到变换后图元控制点的位置。因为控制点确定以后，图元也就确定了。

8.1　平移变换

实现图元的平移需要首先在 CGElement 类中添加一个需要重写的 Move 方法，然后在各派生类中重写它。图元类更新以后，创建一个平移交互类 CMove，它定义鼠标事件时的响应。

8.1.1　更新图元类

1．在 CGElement 类中添加 Move 方法

首先在 CGElement 类中添加一个 Move 方法，该方法有 3 个参数：Graphics 对象 g、移动的基点 basePos 和移动的第 2 点 desPos。desPos 点相对于 base 点的移动方向和距离就是图元相对于当前位置的移动方向和距离。

【VB.NET】

```
'平移图元
Public MustOverride Sub Move(ByVal g As Graphics, _
    ByVal basePos As PointF, ByVal desPos As PointF)
```

【VC#.NET】

```
//平移图元
abstract public void Move(Graphics g,PointF basePos,
    PointF desPos);
```

2．在 CLine 类中重写基类的 Move 方法

在 CLine 类中重写基类的 Move 方法。首先计算 basePos 和 desPos 在横向上和纵向上的偏移量，然后利用该偏移量得到平移后直线段端点的坐标。

【VB.NET】

```
'平移变换
Public Overrides Sub Move(ByVal g As Graphics, _
     ByVal basePos As PointF, ByVal desPos As PointF)
    Dim xx, yy As Single
    '计算偏移量
    xx = desPos.X − basePos.X
```

```
    yy = desPos.Y － basePos.Y
    '偏移后的起点和终点坐标
    m_Begin.X += xx
    m_Begin.Y += yy
    m_End.X += xx
    m_End.Y += yy
End Sub
```

【VC#.NET】

```
//平移变换
override public   void Move(Graphics g,PointF   basePos,PointF desPos)
{
    float xx,yy;
    //计算在 X 和 Y 两个方向上的位移量
    xx = desPos.X － basePos.X;
    yy = desPos.Y － basePos.Y;
    m_Begin.X += xx;
    m_Begin.Y += yy;
    m_End.X += xx;
    m_End.Y += yy;
}
```

3．在 CRectangle 类中重写基类的 Move 方法

在 CRectangle 类中重写基类的 Move 方法，计算矩形平移以后起点和终点的坐标。

【VB.NET】

```
'平移变换
Public Overrides Sub Move(ByVal g As Graphics, _
                ByVal basePos As PointF, ByVal desPos As PointF)
    Dim xx, yy As Single
    '计算偏移量
    xx = desPos.X － basePos.X
    yy = desPos.Y － basePos.Y
    '偏移后的起点和终点坐标
    m_basePos.X += xx
    m_basePos.Y += yy
    m_desPos.X += xx
    m_desPos.Y += yy
End Sub
```

【VC#.NET】

```
//平移变换
```

```
override public  void Move(Graphics g,PointF  basePos,PointF desPos)
{
    float xx,yy;
    //计算在 X 和 Y 两个方向上的位移量
    xx = desPos.X - basePos.X;
    yy = desPos.Y - basePos.Y;
    m_basePos.X += xx;
    m_basePos.Y += yy;
    m_desPos.X += xx;
    m_desPos.Y += yy;
}
```

4．在 CCircle 类中重写基类的 Move 方法

在 CCircle 类中重写基类的 Move 方法，利用第 1 点和第 2 点得到的横向和纵向上的偏移量来计算平移后圆心和圆上一点的新坐标值。

【VB.NET】

```
'平移变换
Public Overrides Sub Move(ByVal g As Graphics, _
            ByVal basePos As PointF, ByVal desPos As PointF)
    Dim xx, yy As Single
    '计算偏移量
    xx = desPos.X - basePos.X
    yy = desPos.Y - basePos.Y
    '平移圆心和圆上一点
    With m_Center
        .X += xx
        .Y += yy
    End With
    With m_PCircle
        .X += xx
        .Y += yy
    End With
End Sub
```

【VC#.NET】

```
//平移变换
override public void Move(Graphics g,PointF  basePos,PointF desPos)
{
    float xx,yy;
    //计算在 X 和 Y 两个方向上的位移量
    xx = desPos.X - basePos.X;
```

```
        yy = desPos.Y - basePos.Y;
        m_Center.X += xx;
        m_Center.Y += yy;
        m_pCircle.X += xx;
        m_pCircle.Y += yy;
    }
```

5．在 CArc 类中重写基类的 Move 方法

在 CArc 类中重写基类的 Move 方法，利用给定的两个点的坐标值计算平移后圆弧圆心、起点和终点的坐标值。

【VB.NET】

```
'平移变换
Public Overrides Sub Move(ByVal g As Graphics, _
        ByVal basePos As PointF, ByVal desPos As PointF)
    Dim xx,yy As Single
    '计算在 X 和 Y 两个方向上的位移量
    xx = desPos.X - basePos.X
    yy = desPos.Y - basePos.Y
    With m_Center
        .X += xx
        .Y += yy
    End With
    With m_Begin
        .X += xx
        .Y += yy
    End With
    With m_End
        .X += xx
        .Y += yy
    End With
End Sub
```

【VC#.NET】

```
//平移变换
override public  void Move(Graphics g,PointF  basePos,PointF desPos)
{
    float xx,yy;
    //计算在 X 和 Y 两个方向上的位移量
    xx = desPos.X - basePos.X;
    yy = desPos.Y - basePos.Y;
    m_Center.X += xx;
```

```
        m_Center.Y += yy;
        m_Begin.X += xx;
        m_Begin.Y += yy;
        m_End.X += xx;
        m_End.Y += yy;
    }
```

8.1.2 创建 CMove 类

CMove 类是一个平移交互类，它实现了 ICommand 接口。CMove 类的作用是定义图元平移操作过程中程序对鼠标事件的响应。第 1 次单击鼠标时，确定第 1 点（即基点）的位置。移动鼠标时，显示橡皮线。第 2 次单击鼠标时，确定第 2 点的位置。用这两个点的坐标值来计算横向上和纵向上的偏移量，然后利用该偏移量计算平移后图元控制点的位置。在新位置上绘制图元，并在屏幕上删除图元在原来位置上的显示。当然，上面所说的图元都是指选择集中的图元。

【VB.NET】

```
'平移变换类
Public Class CMove
    Implements ICommand

    Private m_Step As Integer
    Private m_basePos, m_desPos As PointF

    '单击鼠标左键时的绘图行为
    Public Sub LButtonDown(ByVal g As Graphics, _
                ByVal aPos As PointF) Implements ICommand.LButtonDown
        Dim i As Integer
        m_Step += 1
        Select Case m_Step
            Case 1
                m_basePos = aPos
                m_desPos = aPos
            Case 2
                '清除最后一条橡皮线
                Dim tempLine As New CLine(m_basePos, m_desPos)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing

                '如果选择集中的图元个数大于 0
                If geSels.Count > 0 Then
```

```
                For i = 0 To geSels.Count – 1
                    '清除图元在当前位置上的图形
                    geSels(i).draw(g, CGElement.geDrawMode.Delete)
                    '将所有被选中的图元移动到目标位置并进行绘制
                    geSels(i).Move(g, m_basePos, m_desPos)
                    geSels(i).Draw(g, CGElement.geDrawMode.Selec)
                Next i
            End If

            m_Step = 0
    End Select
End Sub

'移动鼠标时的绘图行为
Public Sub MouseMove(ByVal g As Graphics, _
          ByVal aPos As PointF) Implements ICommand.MouseMove
    Select Case m_Step
        Case 1
            Dim prePos As New PointF()
            Dim curPos As New PointF()

            prePos = m_desPos
            curPos = aPos

            '清除上一条橡皮线
            Dim tempLine1 As New CLine(m_basePos, prePos)
            tempLine1.Draw(g, CGElement.geDrawMode.Drag)
            tempLine1 = Nothing

            '绘当前位置的橡皮线
            Dim tempLine2 As New CLine(m_basePos, curPos)
            tempLine2.Draw(g, CGElement.geDrawMode.Drag)
            tempLine2 = Nothing

            m_desPos = aPos
    End Select
End Sub

'单击鼠标右键时的绘图行为
Public Sub RButtonDown(ByVal g As Graphics, _
```

```
                ByVal aPos As PointF) Implements ICommand.RButtonDown
            If m_Step = 1 Then
                '清除上一条橡皮线
                Dim tempLine As New CLine(m_basePos, m_desPos)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing
            End If
        End Sub
    End Class
```

【VC#.NET】

```
public class CMove:ICommand
{
    private int m_Step;
    private PointF m_basePos, m_desPos;

    public CMove()
    {
    }

    public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels)
    {
        m_Step+=1;
        switch (m_Step)
        {
            case 1:
                m_basePos=aPos;
                m_desPos=aPos;
                break;
            case 2:
                //清除最后一条橡皮线
                CLine tempLine=new CLine(m_basePos,m_desPos);
                tempLine.Draw(g,DrawMode.Drag);
                tempLine=null;

                //将所有被选中的图元移动到目标位置并进行绘制
                if (geSels.Count>0)
                {
                    for (int i=0;i< geSels.Count;i++)
                    {
```

```
                    ((CGElement)(geSels[i])).Move(g,m_basePos,m_desPos);
                    ((CGElement)(geSels[i])).Draw(g,DrawMode.Selec);
                }
            }

            m_Step=0;
            break;
    }
}

public void MouseMove(Graphics g,PointF aPos)
{
    switch (m_Step)
    {
      case 1:
            PointF prePos=new PointF();
            PointF    curPos=new PointF();
            prePos =m_desPos;
            curPos =aPos;
            //清除上一条橡皮线
            CLine tempLine1=new CLine(m_basePos,prePos);
            tempLine1.Draw(g,DrawMode.Drag);
            tempLine1=null;

            //绘当前位置的橡皮线
            CLine tempLine2=new CLine(m_basePos,curPos);
            tempLine2.Draw(g,DrawMode.Drag);
            tempLine2=null;

            m_desPos=aPos;
            break;
    }
}

//单击鼠标右键时的绘图行为
public void    RButtonDown(Graphics g,PointF aPos)
{
    if (m_Step==1)
    {
        //清除上一条橡皮线
```

```
                    CLine tempLine=new CLine(m_basePos,m_desPos);
                    tempLine.Draw(g,DrawMode.Drag);
                    tempLine=null;
                }
            }
        }
```

8.1.3　实现平移图元

要实现图元变换，需要首先添加菜单资源。在前面程序的基础上添加一个“几何变换”菜单，如图 8-1 所示。菜单中包括 5 个选项，即“平移”、“旋转”、“镜像”、“比例放大”和“比例缩小”。名称分别为：mnuTMove，mnuTRotate，mnuTMirror，mnuTZoomIn 和 mnuTZoomOut。

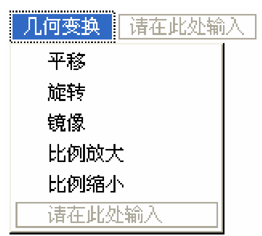

图 8-1　添加“几何变换”菜单

实现选定图元的平移需要首先创建一个 CMove 类实例，然后在 Form1 类中用一个公共的 ICommand 对象引用它，其代码如下：

【VB.NET】

```
Public transMove As New CMove()

Private Sub mnuTMove_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles mnuTMove.Click
    aCommand = transMove
End Sub
```

【VC#.NET】

```
private CMove trMove=new CMove();

private void mnuMove_Click(object sender, System.EventArgs e)
{
    aCommand=trMove;
}
```

运行程序，可以选择绘制的图元并平移它们。图元平移的效果如图 8-2、图 8-3 所示。其中，图 8-2 为平移前的图形，图 8-3 为平移后的图形。

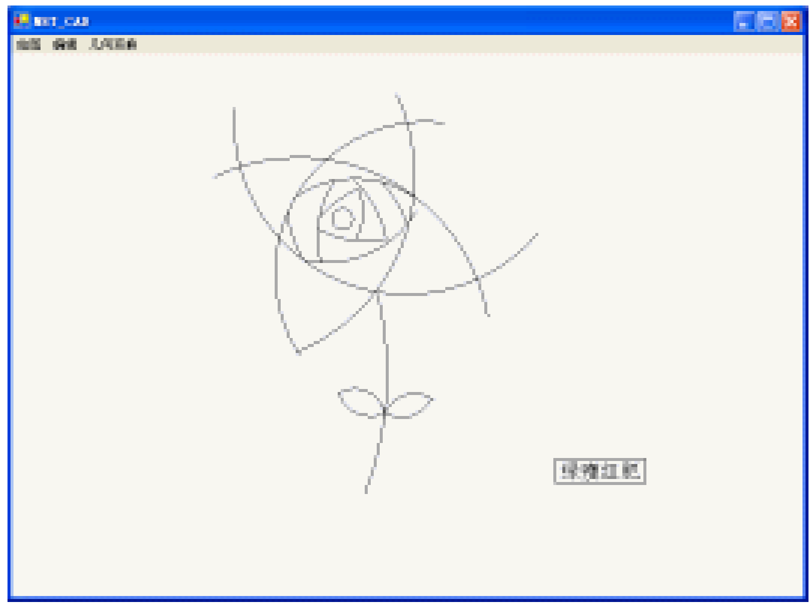

图 8-2　平移前的图形

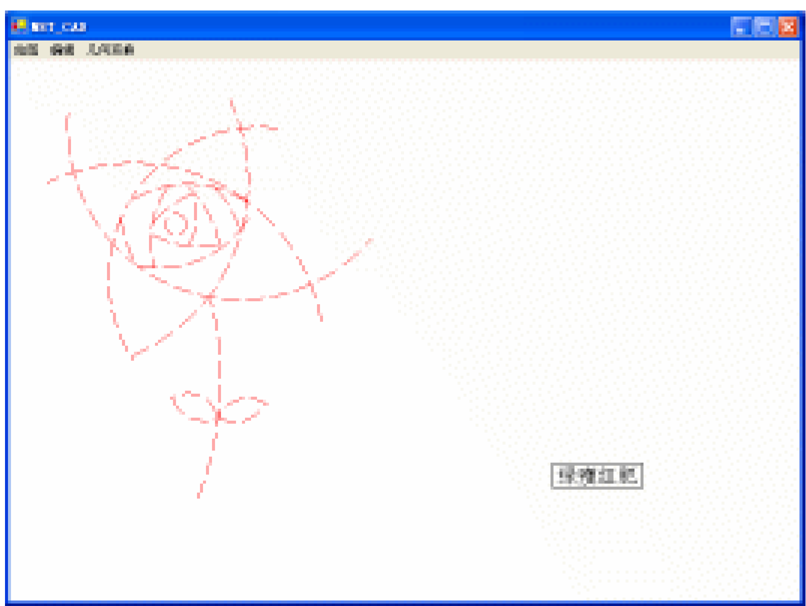

图 8-3　平移后的图形

8.2　旋转变换

图元绕基点旋转以后到达新的位置，称为旋转变换。基点可以是原点，也可以是其他点。本节讨论最通用的情况，即图元绕任意点旋转。绕任意点旋转得到的变换是一个组合变换，需要进行 3 次变换，第 1 次做平移变换，第 2 次做旋转变换，第 3 次做第 1 次变换的逆变换。所以绕任意点（x_0,y_0）旋转的变换矩阵如下所示：

$$\boldsymbol{T}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ -x & -y & 1\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ x & y & 1\end{bmatrix}$$

$$=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ -x_0\cos\theta+y_0\sin\theta+x_0 & -x_0\sin\theta-y_0\cos\theta+y_0 & 1\end{bmatrix}$$

这样，点(x,y)绕任意点（x_0,y_0）旋转以后点的坐标为：

$$x' = x\cos\theta - y\sin\theta - x_0\cos\theta + y_0\sin\theta + x_0$$

$$y' = x\sin\theta + y\cos\theta - x_0\sin\theta - y_0\cos\theta + y_0$$

其中，θ 是点(x,y)和(x_0,y_0)所确定的直线段的方位角。根据上面的计算公式，可以得到图元控制点旋转以后的坐标值。实际操作中，需要首先在 CGElement 类中添加一个需要重写的 Rotate 方法，然后在各派生类中重写它。图元类更新以后，创建一个旋转交互类 CRotate，它定义触发鼠标事件时的绘图行为。

8.2.1　更新图元类

1．在 CGElement 类中添加 Rotate 方法

首先在 CGElement 类中添加一个 Rotate 方法，该方法有 3 个参数：g 参数为 Graphics 对象，指定绘图表面；basePos 为基点；aAngle 为旋转角度，以弧度为单位。

【VB.NET】

```
旋转图元
Public MustOverride Sub Rotate(ByVal g As Graphics, _
     ByVal basePos As PointF, ByVal aAngle As Single)
```

【VC#.NET】

```
//旋转图元
abstract public void Rotate(Graphics g,PointF basePos,
     float aAangle);
```

2．在 CLine 类中重写基类的 Rotate 方法

在 CLine 类中重写基类的 Rotate 方法，计算旋转以后直线段起点和终点的位置。

【VB.NET】

```
旋转变换
Public Overrides Sub Rotate(ByVal g As Graphics, _
          ByVal basePos As PointF, ByVal aAngle As Single)
     m_Begin = pRotate(basePos, m_Begin, aAngle)
     m_End = pRotate(basePos, m_End, aAngle)
End Sub
```

【VC#.NET】

```
//旋转变换
override public void Rotate(Graphics g,PointF basePos, float aAngle)
{
    m_Begin = m.pRotate(basePos, m_Begin, aAngle);
    m_End = m.pRotate(basePos, m_End, aAngle);
}
```

上面的代码中用到了一个 pRotate **函数，该函数返回绕基点旋转以后图元控制点的坐标。函数有** 3 **个参数：**baseP **为基点，**Pos **为要旋转的点，**angle **为旋转角度。返回值为一个** PointF **对象，为旋转以后的** Pos **点对象。旋转以后的坐标根据前面绕任意点旋转以后点的坐标的计算公式得到。**

【VB.NET】

```
'计算旋转变换以后点的坐标
Public Function pRotate(ByVal baseP As PointF, _
            ByVal Pos As PointF, ByVal angle As Single) As PointF
    Dim pr As New PointF()
    Dim sinv,cosv As Single

    cosv = Cos(angle)
    sinv = Sin(angle)
    With pr
        .X = Pos.X * cosv - Pos.Y * sinv + (1 - cosv) * baseP.X + baseP.Y * sinv
        .Y = sinv * Pos.X + cosv * Pos.Y + (1 - cosv) * baseP.Y - sinv * baseP.X
    End With
    Return pr
End Function
```

【VC#.NET】

```
//计算旋转变换以后点的坐标
public PointF pRotate(PointF baseP,PointF Pos,float angle){
    PointF pr=new PointF();
    float sinv,cosv;

    cosv = (float)(Math.Cos(angle));
    sinv = (float)(Math.Sin(angle));
    pr.X = Pos.X * cosv - Pos.Y * sinv + (1 - cosv) * baseP.X + baseP.Y * sinv;
    pr.Y = sinv * Pos.X + cosv * Pos.Y + (1 - cosv) * baseP.Y - sinv * baseP.X;
    return pr;
}
```

3．在 CRectangle 类中重写基类的 Rotate 方法

在 CRectangle 类中重写基类的 Rotate 方法，利用 pRotate 函数计算旋转以后矩形的起点和终点的位置。

【VB.NET】

```
'旋转变换
Public Overrides Sub Rotate(ByVal g As Graphics, _
                ByVal basePos As PointF, ByVal aAngle As Single)
    m_basePos = pRotate(basePos, m_basePos, aAngle)
    m_desPos = pRotate(basePos, m_desPos, aAngle)
End Sub
```

【VC#.NET】

```
//旋转变换
override public void Rotate(Graphics g,PointF basePos, float aAngle)
{
    m_basePos = m.pRotate(basePos, m_basePos, aAngle);
    m_desPos = m.pRotate(basePos, m_desPos, aAngle);
}
```

4．在 CCircle 类中重写基类的 Rotate 方法

在 CCircle 类中重写基类的 Rotate 方法，计算旋转以后圆的圆心和圆上一点的坐标。

【VB.NET】

```
'旋转变换
Public Overrides Sub Rotate(ByVal g As Graphics, _
                ByVal basePos As PointF, ByVal aAngle As Single)
    m_Center = pRotate(basePos, m_Center, aAngle)
    m_PCircle = pRotate(basePos, m_PCircle, aAngle)
End Sub
```

【VC#.NET】

```
//旋转变换
override public void Rotate(Graphics g,PointF basePos, float aAngle)
{
    m_Center = m.pRotate(basePos, m_Center, aAngle);
    m_pCircle = m.pRotate(basePos, m_pCircle, aAngle);
}
```

5．在 CArc 类中重写基类的 Rotate 方法

在 CCircle 类中重写基类的 Rotate 方法，计算旋转以后圆弧的圆心、起点和终点的坐标。

【VB.NET】

```
'旋转变换
Public Overrides Sub Rotate(ByVal g As Graphics, _
            ByVal basePos As PointF, ByVal aAngle As Single)
    m_Center = pRotate(basePos, m_Center, aAngle)
    m_Begin = pRotate(basePos, m_Begin, aAngle)
    m_End = pRotate(basePos, m_End, aAngle)
End Sub
```

【VC#.NET】

```
//旋转变换
override public void Rotate(Graphics g,PointF basePos, float aAngle)
{
    m_Center =m.pRotate(basePos, m_Center, aAngle);
    m_Begin = m.pRotate(basePos, m_Begin, aAngle);
    m_End = m.pRotate(basePos, m_End, aAngle);
}
```

8.2.2 创建 CRotate 类

CRotate 类定义用鼠标进行交互旋转变换操作时的绘图行为。它实现了 ICommand 接口。第 1 次单击鼠标左键时，定义旋转的基点。第 2 次单击鼠标左键时，确定第 2 点并旋转图元，旋转角度为它与基点之间的方位角。进行旋转的图元为选择集中的图元。删除图元旋转前位置上的显示，在新位置上绘制图元。

【VB.NET】

```
'旋转变换
Public Class CRotate
    Implements ICommand

    Private m_Step As Integer
    Private m_basePos, m_desPos As PointF

    '单击鼠标左键时的绘图行为
    Public Sub LButtonDown(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.LButtonDown
        Dim i As Integer
        '鼠标单击次数加 1
        m_Step += 1
        Select Case m_Step
            Case 1
                m_basePos = aPos
```

```
                m_desPos = aPos
            Case 2
                '清除最后一条橡皮线
                Dim tempLine As New CLine(m_basePos, m_desPos)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing

                '如果选择集中的图元个数大于 0
                If geSels.Count > 0 Then
                    Dim aAngle As Single
                    '计算旋转角度
                    aAngle = GetAngle(m_basePos, m_desPos)
                    For i = 0 To geSels.Count － 1
                        '清除图元在当前位置的图形
                        geSels(i).Draw(g, CGElement.geDrawMode.Delete)
                        '将所有被选中的图元旋转到目标位置并进行绘制
                        geSels(i).Rotate(g, m_basePos, aAngle)
                        geSels(i).Draw(g, CGElement.geDrawMode.Selec)
                    Next i
                End If

                m_Step = 0
        End Select
End Sub

'移动鼠标时的绘图行为
Public Sub MouseMove(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.MouseMove
        Select Case m_Step
            Case 1
                Dim prePos As New PointF()
                Dim curPos As New PointF()

                prePos = m_desPos
                curPos = aPos

                '清除上一条橡皮线
                Dim tempLine1 As New CLine(m_basePos, prePos)
                tempLine1.Draw(g, CGElement.geDrawMode.Drag)
                tempLine1 = Nothing
```

```
                    '绘当前位置的橡皮线
                    Dim tempLine2 As New CLine(m_basePos, curPos)
                    tempLine2.Draw(g, CGElement.geDrawMode.Drag)
                    tempLine2 = Nothing

                    m_desPos = aPos
            End Select
        End Sub

        '单击鼠标右键时的绘图行为
        Public Sub RButtonDown(ByVal g As Graphics, _
                    ByVal aPos As PointF) Implements ICommand.RButtonDown
            If m_Step = 1 Then
                '清除上一条橡皮线
                Dim tempLine As New CLine(m_basePos, m_desPos)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing
            End If
        End Sub

    End Class
```

【VC#.NET】

```
    public class CRotate:ICommand
    {
        private int m_Step;
        private PointF m_basePos, m_desPos;
        Module m=new Module();

        public CRotate()
        {
        }

        public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,
                                ArrayList geSels)
        {
            m_Step+=1;

            switch (m_Step)
            {
                case 1:
```

```
                m_basePos=aPos;
                m_desPos=aPos;
                break;
            case 2:
                //清除最后一条橡皮线
                CLine tempLine=new CLine(m_basePos,m_desPos);
                tempLine.Draw(g,DrawMode.Drag);
                tempLine=null;

                //将所有被选中的图元移动到目标位置并进行绘制
                if (geSels.Count>0)
                {
                    float aAngle=m.GetAngle(m_basePos,m_desPos);
                    for (int i=0;i< geSels.Count;i++)
                    {
                        ((CGElement)(geSels[i])).Rotate(g,m_basePos,aAngle);
                        ((CGElement)(geSels[i])).Draw(g,DrawMode.Selec);
                    }
                }

                m_Step=0;
                break;
        }
    }

    public void MouseMove(Graphics g,PointF aPos)
    {
        switch (m_Step)
        {
          case 1:
              PointF prePos=new PointF();
              PointF    curPos=new PointF();
              prePos =m_desPos;
              curPos =aPos;
              //清除上一条橡皮线
              CLine tempLine1=new CLine(m_basePos,prePos);
              tempLine1.Draw(g,DrawMode.Drag);
              tempLine1=null;

              //绘当前位置的橡皮线
              CLine tempLine2=new CLine(m_basePos,curPos);
```

```
                tempLine2.Draw(g,DrawMode.Drag);
                tempLine2=null;

                m_desPos=aPos;
                break;
            }
        }

        //单击鼠标右键时的绘图行为
        public void    RButtonDown(Graphics g,PointF aPos)
        {
            if (m_Step==1)
            {
                //清除上一条橡皮线
                CLine tempLine=new CLine(m_basePos,m_desPos);
                tempLine.Draw(g,DrawMode.Drag);
                tempLine=null;
            }
        }
    }
```

8.2.3　实现旋转图元

8.2.2 节中已经创建了实现旋转图元功能的菜单资源，下面的程序先创建一个 CRotate 类实例，然后用 ICommand 对象 aCommand 引用它。

【VB.NET】

```
Public transRotate As New CRotate()

Private Sub mnuTRotate_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles mnuTRotate.Click
    aCommand = transRotate
End Sub
```

【VC#.NET】

```
private CRotate trRotate=new CRotate();

private void mnuRotate_Click(object sender, System.EventArgs e)
{
    aCommand=trRotate;
}
```

旋转图 8-2 中的那支花，效果如图 8-4 所示。

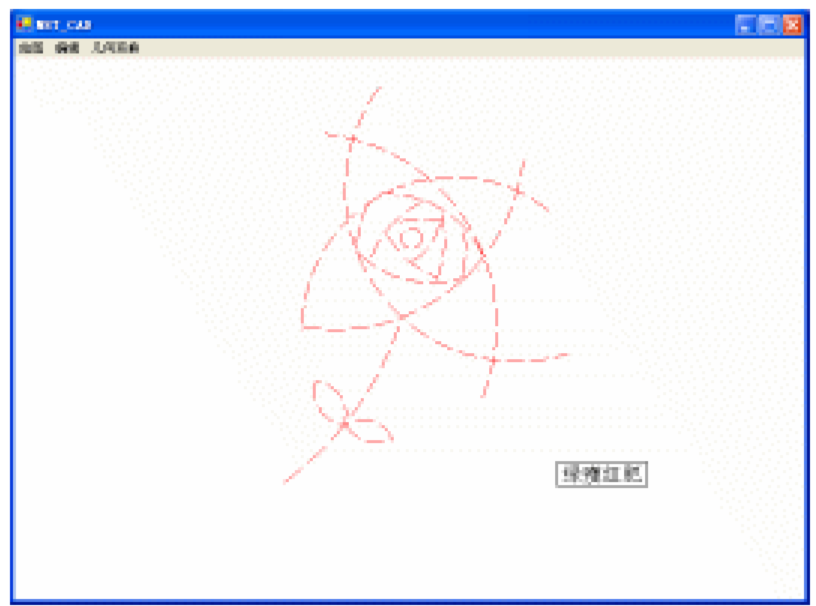

图 8-4 旋转图元

8.3 镜像图元

对图元进行镜像变换是要把图元相对于一个点或一条直线对称显示到另一面。比如在直角坐标系中，第 1 象限中的一个三角形相对于 Y 轴做镜像变换，则将三角形对称显示在第 2 象限中；若相对于原点做镜像变换，则对称显示在第 3 象限中。

本章讨论最通用的情况，即图元相对于任意直线进行镜像。相对于任意直线进行镜像变换时，需要进行 5 次变换。第 1 次进行平移变换，第 2 次进行旋转变换，第 3 次进行镜像变换，然后，先后做第 2 次和第 1 次变换的逆变换。所以，相对于任意直线的镜像变换的变换矩阵如下所示：

$$\boldsymbol{T} = \boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_3\boldsymbol{T}_4\boldsymbol{T}_5$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -a & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & a & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ -a\sin 2\theta & a\cos 2\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & a & 1 \end{bmatrix} = \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ -a\sin 2\theta & a\cos 2\theta + a & 1 \end{bmatrix}$$

故，变换后点的坐标为：

$$x' = x\cos 2\theta + y\sin 2\theta - a\sin 2\theta$$

$$y' = x\sin 2\theta - y\cos 2\theta + a\cos 2\theta + a$$

对于两个点(x_1, y_1)和(x_2, y_2)确定的一条直线段，其截距方程中的常数项 a 可用下式求得：

$$a = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$$

参数θ 是点(x_1,y_1)和(x_2,y_2)所确定的直线段的方位角。

这样，图元控制点旋转以后的坐标值可以求得。实际编程时，需要首先在 CGElement 类中添加一个需要重写的 Mirror 方法，然后在各派生类中重写它。图元类更新以后，创建一个旋转交互类 CMirror，它定义触发鼠标事件时的绘图行为。

8.3.1　更新图元类

1．在 CGElement 类中添加 Mirror 方法

需要首先在 CGElement 类中添加 1 个需要重写的 Mirror 方法。利用该方法求取镜像以后图元控制点的位置。Mirror 方法有 3 个参数，其中 g 为 Graphics 对象，指定绘图表面，pPos1 和 pPos2 分别定义代表镜像对称轴的直线段的起点和终点。

【VB.NET】

```
'镜像图元
Public MustOverride Sub Mirror(ByVal g As Graphics, _
     ByVal pPos1 As PointF, ByVal pPos2 As PointF)
```

【VC#.NET】

```
//镜像图元
abstract public void Mirror(Graphics g,PointF pPos1,
     PointF pPos2);
```

2．在 CLine 类中重写基类的 Mirror 方法

在 CLine 类中重写基类的 Mirror 方法，计算镜像以后直线段起点和终点的坐标值。计算过程是通过 pMirror 函数实现的。pMirror 函数有 3 个参数，Pos1 和 Pos2 表示对称轴的起点和终点，Pos 表示要镜像的点。函数返回 1 个 PointF 对象，为镜像后的点。

【VB.NET】

```
'镜像变换
Public Overrides Sub Mirror(ByVal g As Graphics, _
          ByVal pPos1 As PointF, ByVal pPos2 As PointF)
     m_Begin = pMirror(pPos1, pPos2, m_Begin)
     m_End = pMirror(pPos1, pPos2, m_End)
End Sub

'计算镜像以后点的坐标
Public Function pMirror(ByVal Pos1 As PointF, _
          ByVal Pos2 As PointF, ByVal Pos As PointF) As PointF
     Dim pm As New PointF()
     Dim Angle,cos2v, sin2v As Single
     Dim x1, y1, x2, y2 As Single
```

```
        Dim aa, desX, desY As Single
        With Pos1
            x1 = .X
            y1 = .Y
        End With
        With Pos2
            x2 = .X
            y2 = .Y
        End With
        If x2 = x1 Then
            aa = 10000000
        Else
            aa = (x2 * y1 - x1 * y2) / (x2 - x1)
        End If
        Angle = GetAngle(Pos1, Pos2)
        cos2v = Cos(Angle * 2)
        sin2v = Sin(Angle * 2)
        desX = Pos.X * cos2v + Pos.Y * sin2v - aa * sin2v
        desY = Pos.X * sin2v - Pos.Y * cos2v + aa * cos2v + aa
        With pm
            .X = desX
            .Y = desY
        End With
        Return pm
    End Function
```

【VC#.NET】

```
    //镜像变换
    override public void Mirror(Graphics g,PointF pPos1,PointF pPos2)
    {
        m_Begin = m.pMirror(pPos1, pPos2, m_Begin);
        m_End = m.pMirror(pPos1, pPos2, m_End);
    }

    //计算镜像以后点的坐标
    public PointF pMirror(PointF Pos1,PointF Pos2, PointF Pos){
        PointF pm=new PointF();
        float Angle, cos2v,sin2v;
        float x1,y1, x2,y2;
        float aa, desX,desY;
        x1 =Pos1 .X;
```

```
        y1 =Pos1.Y;
        x2 =Pos2 .X;
        y2 = Pos2.Y;
        if (x2 == x1) {
            aa = 10000000;
        }
        else
        {
            aa = (x2 * y1 - x1 * y2) / (x2 - x1);
        }
        Angle = GetAngle(Pos1, Pos2);
        cos2v = (float)((Math.Cos(Angle * 2)));
        sin2v = (float)((Math.Sin(Angle * 2)));
        desX = Pos.X * cos2v + Pos.Y * sin2v - aa * sin2v;
        desY = Pos.X * sin2v - Pos.Y * cos2v + aa * cos2v + aa;
        pm.X = desX;
        pm.Y = desY;
        return pm;
    }
```

3．在 CRectangle 类中重写基类的 Mirror 方法

在 CRectangle 类中重写基类的 Mirror 方法时，用 pMirror 函数计算镜像以后矩形起点和终点的坐标值。

【VB.NET】

```
'镜像变换
Public Overrides Sub Mirror(ByVal g As Graphics, ByVal pPos1 As PointF, ByVal pPos2 As PointF)
    m_basePos = pMirror(pPos1, pPos2, m_basePos)
    m_desPos = pMirror(pPos1, pPos2, m_desPos)
End Sub
```

【VC#.NET】

```
//镜像变换
override public void Mirror(Graphics g,PointF pPos1,PointF pPos2)
{
    m_basePos = m.pMirror(pPos1, pPos2, m_basePos);
    m_desPos = m.pMirror(pPos1, pPos2, m_desPos);
}
```

4．在 CCircle 类中重写基类的 Mirror 方法

在 CCircle 类中重写基类的 Mirror 方法，获得圆相对于任意直线段镜像以后控制点的坐标值。

【VB.NET】

```
'镜像变换
Public Overrides Sub Mirror(ByVal g As Graphics, _
        ByVal pPos1 As PointF, ByVal pPos2 As PointF)
    m_Center = pMirror(pPos1, pPos2, m_Center)
    m_PCircle = pMirror(pPos1, pPos2, m_PCircle)
End Sub
```

【VC#.NET】

```
//镜像变换
override public void Mirror(Graphics g,PointF pPos1,PointF pPos2)
{
    m_Center =m.pMirror(pPos1, pPos2, m_Center);
    m_pCircle = m.pMirror(pPos1, pPos2, m_pCircle);
}
```

5．在 CArc 类中重写基类的 Mirror 方法

在 CArc 类中重写基类的 Mirror 方法，获得圆弧相对于任意直线段镜像以后圆心、起点和终点的坐标值。需要注意的是，由于缺省时圆弧总是以逆时针方向画圆，所以镜像以后需要交换圆弧起点和终点的位置。

【VB.NET】

```
'镜像变换
Public Overrides Sub Mirror(ByVal g As Graphics, _
            ByVal pPos1 As PointF, ByVal pPos2 As PointF)
    '注意，圆弧的镜像变换要交换起点和终点的坐标
    m_Center = pMirror(pPos1, pPos2, m_Center)
    Dim pt As PointF
    pt = pMirror(pPos1, pPos2, m_Begin)
    m_Begin = pMirror(pPos1, pPos2, m_End)
    m_End = pt
End Sub
```

【VC#.NET】

```
//镜像变换
override public void Mirror(Graphics g,PointF pPos1,PointF pPos2)
{
    //注意，圆弧的镜像变换要交换起点和终点的坐标
    m_Center =m.pMirror(pPos1, pPos2, m_Center);
    PointF pt = m.pMirror(pPos1, pPos2, m_Begin);
    m_Begin = m.pMirror(pPos1, pPos2, m_End);
    m_End = pt;
}
```

8.3.2 创建 CMirror 类

为了实现交互镜像，还需要创建一个 CMirror 类，专门定义触发鼠标事件时的响应行为。该类实现了 ICommand 接口。第 1 次单击鼠标左键时，确定镜像轴的起点，第 2 次单击鼠标左键时，确定镜像轴的终点。对于选择集中的图元，首先在屏幕上用删除模式重画，然后进行变换，并在变换后的位置上用选择模式重画。

【VB.NET】

```
'镜像变换
Public Class CMirror
    Implements ICommand

    Dim m_Step As Integer
    Dim m_Pos1, m_Pos2 As PointF

    '单击鼠标左键时的绘图行为
    Public Sub LButtonDown(ByVal g As Graphics, _
                ByVal aPos As PointF) Implements ICommand.LButtonDown
        Dim i As Integer

        '鼠标单击次数加 1
        m_Step += 1

        Select Case m_Step
            Case 1
                m_Pos1 = aPos
                m_Pos2 = aPos
            Case 2
                '清除最后一条橡皮线
                Dim tempLine As New CLine(m_Pos1, m_Pos2)
                tempLine.Draw(g, CGElement.geDrawMode.Drag)
                tempLine = Nothing

                '如果选择集中的图元个数大于 0
                If geSels.Count > 0 Then
                    For i = 0 To geSels.Count － 1
                        '清除图元在当前位置上的图形
                        geSels(i).draw(g, CGElement.geDrawMode.Delete)
                        '将所有被选中的图元镜像到目标位置并进行绘制
                        geSels(i).Mirror(g, m_Pos1, m_Pos2)
                        geSels(i).Draw(g, CGElement.geDrawMode.Selec)
```

```
                Next i
            End If

            m_Step = 0
    End Select
End Sub

'移动鼠标时的绘图行为
Public Sub MouseMove(ByVal g As Graphics, _
            ByVal aPos As PointF) Implements ICommand.MouseMove
    Select Case m_Step
        Case 1
            Dim prePos As New PointF()
            Dim curPos As New PointF()

            prePos = m_Pos2
            curPos = aPos

            '清除上一条橡皮线
            Dim tempLine1 As New CLine(m_Pos1, prePos)
            tempLine1.Draw(g, CGElement.geDrawMode.Drag)
            tempLine1 = Nothing

            '绘当前位置的橡皮线
            Dim tempLine2 As New CLine(m_Pos1, curPos)
            tempLine2.Draw(g, CGElement.geDrawMode.Drag)
            tempLine2 = Nothing

            m_Pos2 = aPos
    End Select
End Sub

'单击鼠标右键时的绘图行为
Public Sub RButtonDown(ByVal g As Graphics, _
        ByVal aPos As PointF) Implements ICommand.RButtonDown
    If m_Step = 1 Then
        '清除上一条橡皮线
        Dim tempLine As New CLine(m_Pos1, m_Pos2)
        tempLine.Draw(g, CGElement.geDrawMode.Drag)
        tempLine = Nothing
    End If
```

```
        End Sub

    End Class
```

【VC#.NET】

```
    public class CMirror:ICommand
    {
        private int m_Step;
        private PointF m_Pos1, m_Pos2;

        public CMirror()
        {
        }

        public void LButtonDown(Graphics g,PointF aPos,ArrayList ges,ArrayList geSels)
        {
            m_Step+=1;

            switch (m_Step)
            {
                case 1:
                    m_Pos1=aPos;
                    m_Pos2=aPos;
                    break;
                case 2:
                    //清除最后一条橡皮线
                    CLine tempLine=new CLine(m_Pos1,m_Pos2);
                    tempLine.Draw(g,DrawMode.Drag);
                    tempLine=null;

                    //将所有被选中的图元移动到目标位置并进行绘制
                    if (geSels.Count>0)
                    {
                        for (int i=0;i< geSels.Count;i++)
                        {
                            ((CGElement)(geSels[i])).Mirror(g,m_Pos1,m_Pos2);
                            ((CGElement)(geSels[i])).Draw(g,DrawMode.Selec);
                        }
                    }
```

```
                    m_Step=0;
                    break;
            }
        }

        public void MouseMove(Graphics g,PointF aPos)
        {
            switch (m_Step)
            {
                case 1:
                    PointF prePos=new PointF();
                    PointF    curPos=new PointF();
                    prePos =m_Pos2;
                    curPos =aPos;
                    //清除上一条橡皮线
                    CLine tempLine1=new CLine(m_Pos1,prePos);
                    tempLine1.Draw(g,DrawMode.Drag);
                    tempLine1=null;

                    //绘当前位置的橡皮线
                    CLine tempLine2=new CLine(m_Pos1,curPos);
                    tempLine2.Draw(g,DrawMode.Drag);
                    tempLine2=null;

                    m_Pos2=aPos;
                    break;
            }
        }

        //单击鼠标右键时的绘图行为
        public void    RButtonDown(Graphics g,PointF aPos)
        {
            if (m_Step==1)
            {
                //清除上一条橡皮线
                CLine tempLine=new CLine(m_Pos1,m_Pos2);
                tempLine.Draw(g,DrawMode.Drag);
                tempLine=null;
            }
        }
    }
```

8.3.3　实现镜像图元

8.1.3 节创建了用于镜像变换的菜单项 mnuTMirror，下面创建一个 CMirror 类实例 transMirror，然后用 ICommand 对象 aCommand 引用它。

【VB.NET】

```
Public transMirror As New CMirror()

Private Sub mnuTMirror_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles mnuTMirror.Click
    aCommand = transMirror
End Sub
```

【VC#.NET】

```
private CMirror trMirror=new CMirror();

private void mnuMirror_Click(object sender, System.EventArgs e)
{
    aCommand=trMirror;
}
```

使用“几何变换”菜单中的“镜像变换”选项，对图 8-2 中的花进行镜像变换，得到图 8-5 所示的图形。

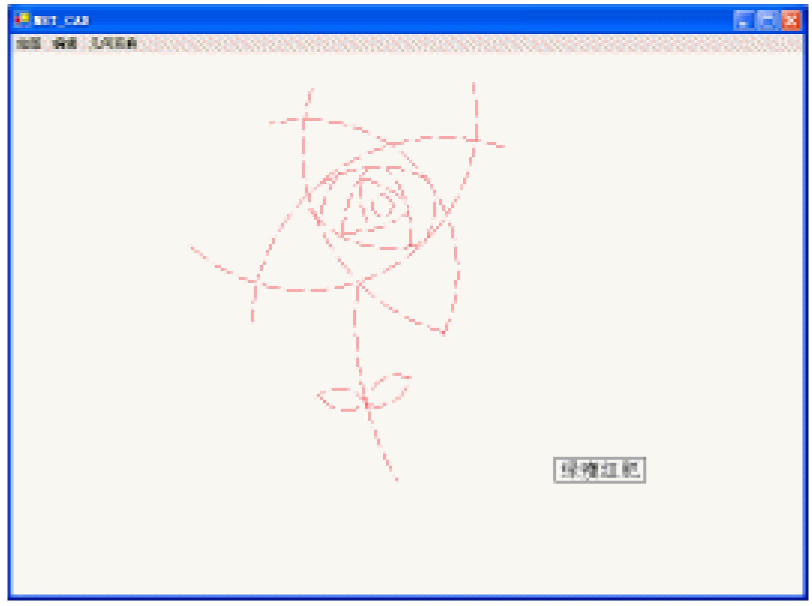

图 8-5　镜像变换以后的图形

8.4　比例缩放图元

在绘图过程中有对某个或某些图元进行比例缩小或放大的要求，通过比例变换可以实现它。比例缩放前需要将要变换的图元放到选择集中，变换只针对选择集中的图元进行。具体操作时，通过单击菜单选项或命令按钮就可以实现比例缩放，所以不必像前面几种变换那样建立专门的交互操作类，但仍然需要在CGElement类中添加相应的方法并在派生类中实现它。

8.4.1　在 CGElement 类中添加 Scale 方法

在 CGElement 类中添加一个 Scale 方法。该方法有 3 个参数：g 为 Graphics 对象，指定绘图表面，scalex 和 scaley 分别为横向和纵向上的缩放比例。

【VB.NET】

```
'比例缩放图元
Public MustOverride Sub Scale(ByVal g As Graphics, _
     ByVal scalex As Single, ByVal scaley As Single)
```

【VC#.NET】

```
//比例缩放图元
abstract public void Scale(Graphics g,float scalex,
     float scaley);
```

8.4.2　在派生类中重写 Scale 方法

在 CGElement 类中添加 Scale 方法以后，在各派生类中重写该方法，确定变换以后图元控制点的坐标。

1．在 CLine 类中重写 Scale 方法

下面在 CLine 类中重写 Scale 方法，计算比例变换以后直线段起点和终点的坐标。

【VB.NET】

```
'比例变换
Public Overrides Sub Scale(ByVal g As Graphics, ByVal sx As Single, ByVal sy As Single)
    With m_Begin
        .X = .X * sx
        .Y = .Y * sy
    End With
    With m_End
        .X = .X * sx
        .Y = .Y * sy
    End With
End Sub
```

【VC#.NET】

```
//比例变换
override public void Scale(Graphics g, float sx,float sy)
{
    m_Begin.X *=sx;
    m_Begin.Y *= sy;
    m_End.X *= sx;
    m_End.Y *=sy;
}
```

2．在 CRectangle 类中重写 Scale 方法

下面在 CRectangle 类中重写基类的 Scale 方法，计算镜像变换以后矩形起点和终点的坐标值。

【VB.NET】

```
'比例变换
Public Overrides Sub Scale(ByVal g As Graphics, ByVal sx As Single, ByVal sy As Single)
    With m_basePos
        .X = .X * sx
        .Y = .Y * sy
    End With
    With m_desPos
        .X = .X * sx
        .Y = .Y * sy
    End With
End Sub
```

【VC#.NET】

```
//比例变换
override public void Scale(Graphics g, float sx,float sy)
{
    m_basePos.X *=sx;
    m_basePos.Y *= sy;
    m_desPos.X *= sx;
    m_desPos.Y *=sy;
}
```

3．在 CCircle 类中重写 Scale 方法

下面在 CCircle 类中重写 Scale 方法，计算变换以后圆的圆心和圆上一点的坐标值。

【VB.NET】

```
'比例变换
Public Overrides Sub Scale(ByVal g As Graphics, ByVal sx As Single, ByVal sy As Single)
```

```
        With m_Center
            .X = .X * sx
            .Y = .Y * sy
        End With
        With m_PCircle
            .X = .X * sx
            .Y = .Y * sy
        End With
    End Sub
```

【VC#.NET】

```
    //比例变换
    override public void Scale(Graphics g, float sx,float sy)
    {
        m_Center.X *= sx;
        m_Center.Y *= sy;
        m_pCircle.X    *=sx;
        m_pCircle.Y *= sy;
    }
```

4．在 CArc 类中重写 Scale 方法

在 CArc 类中重写基类的 Scale 方法，计算变换以后圆弧的圆心、起点和终点的坐标值。

【VB.NET】

```
    '比例变换
    Public Overrides Sub Scale(ByVal g As Graphics, ByVal sx As Single, ByVal sy As Single)
        With m_Center
            .X = .X * sx
            .Y = .Y * sy
        End With
        With m_Begin
            .X = .X * sx
            .Y = .Y * sy
        End With
        With m_End
            .X = .X * sx
            .Y = .Y * sy
        End With
    End Sub
```

【VC#.NET】

```
    //比例变换
```

```
override public void Scale(Graphics g, float sx,float sy)
{
    m_Center.X *= sx;
    m_Center.Y *= sy;
    m_Begin.X    *=sx;
    m_Begin.Y *= sy;
    m_End.X *= sx;
    m_End.Y *=sy;
 }
```

8.4.3　实现比例变换

在前面程序的基础上添加下面的代码，确定在单击“比例放大”菜单项时将图形放大到 1.2 倍，单击“比例缩小”菜单项时将图形缩小到原图的 0.8 倍。

【VB.NET】

```
Private Sub mnuTZoomin_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles mnuTZoomout.Click
    Dim g As Graphics = Me.CreateGraphics()
    ScaleZoom(g, 1.2, 1.2)
End Sub

Private Sub mnuTZoomout_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles mnuTZoomin.Click
    Dim g As Graphics = Me.CreateGraphics()
    ScaleZoom(g, 0.8, 0.8)
End Sub
```

【VC#.NET】

```
private void mnuZoomIn_Click(object sender, System.EventArgs e)
{
     Graphics g = this.CreateGraphics();
     m.ScaleZoom(g, 1.2f, 1.2f,geSels);
}

private void mnuZoomOut_Click(object sender, System.EventArgs e)
{
     Graphics g = this.CreateGraphics();
     m.ScaleZoom(g, 0.8f, 0.8f,geSels);
}
```

运行程序以后，绘图，选择图元，然后进行比例缩放。对图 8-2 中的花进行放大和缩小以后的图形效果如图 8-6 和图 8-7 所示。

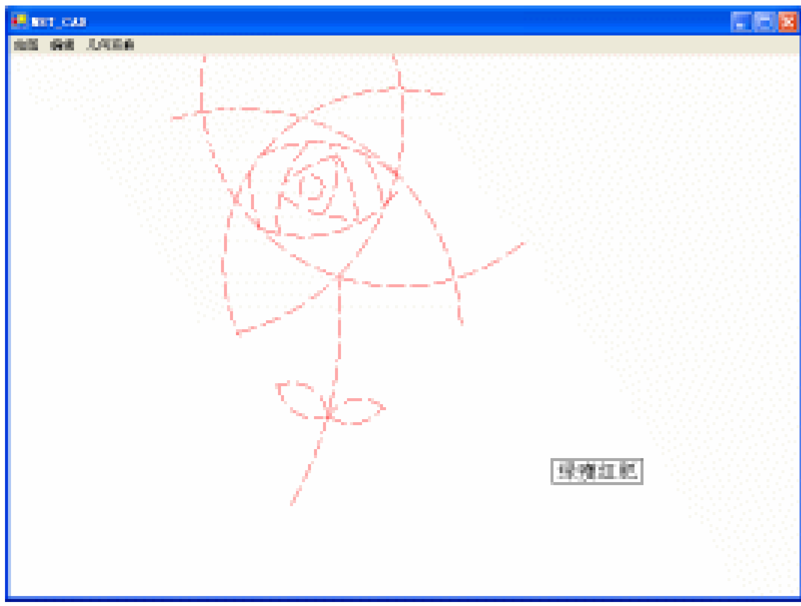

图 8-6　放大选定图元

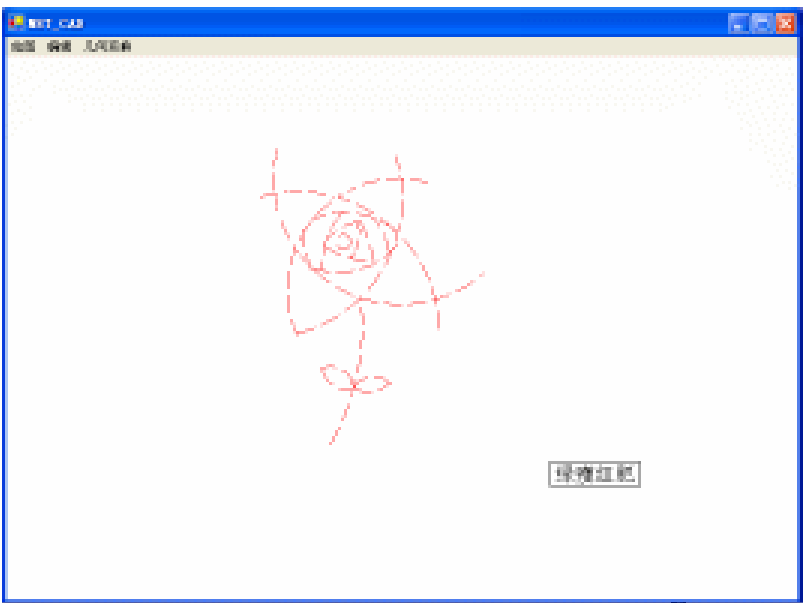

图 8-7　缩小选定图元

第 9 章　GDI+ 提供的交互技巧

前面几章介绍了一个比较完整的小型交互式 CAD 系统。其中拾取和几何变换等技术都是通过自己编写算法代码来实现的。通过自己编写算法代码，可以比较清楚地了解实现相关技术的来龙去脉。实际上，GDI+某些类中封装了一些相关的属性和方法，利用它们，可以比较容易地完成相同的任务。本章将具体介绍利用 GDI+求取图元的包围矩形，拾取图元和进行几何变换。

9.1　获取线形图元的包围矩形

利用 GraphicsPath 类的 GetBounds 属性，可以轻易地获取路径的包围矩形。下面的例子把直线段、圆、圆弧和文本分别放到不同的路径中，然后用 GetBounds 属性获得它们各自的包围矩形并显示该矩形。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    Dim gp1 As New GraphicsPath()
    Dim gp2 As New GraphicsPath()
    Dim gp3 As New GraphicsPath()
    Dim gp4 As New GraphicsPath()
    Dim i As Integer

    '利用路径的 GetBounds 函数获取图元的包围矩形，
    '并且绘出该矩形
    gp1.AddLine(New PointF(10, 10), New PointF(100, 100))
    gp2.AddEllipse(150, 100, 100, 100)
    gp3.AddArc(10, 80, 100, 100, 20, 80)
    Dim s As String = "巴蜀秋山形又瘦，岳麓红枫醉几回"
    Dim sf As StringFormat = New StringFormat(StringFormatFlags.NoWrap)
    Dim f As FontFamily = New FontFamily("隶书")
    gp4.AddString(s, f, 1, 30, New PointF(120, 50), sf)

    Dim gb1 As RectangleF = gp1.GetBounds
    Dim gb2 As RectangleF = gp2.GetBounds
    Dim gb3 As RectangleF = gp3.GetBounds
    Dim gb4 As RectangleF = gp4.GetBounds
    g.DrawPath(Pens.Black, gp1)
```

```
        g.DrawRectangle(Pens.Red, gb1.Left, gb1.Top, gb1.Width, gb1.Height)
        g.DrawPath(Pens.Black, gp2)
        g.DrawRectangle(Pens.Red, gb2.Left, gb2.Top, gb2.Width, gb2.Height)
        g.DrawPath(Pens.Black, gp3)
        g.DrawRectangle(Pens.Red, gb3.Left, gb3.Top, gb3.Width, gb3.Height)
        g.DrawPath(Pens.Blue, gp4)
        g.DrawRectangle(Pens.Red, gb4.Left, gb4.Top, gb4.Width, gb4.Height)
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;

        GraphicsPath gp1=new GraphicsPath();
        GraphicsPath gp2=new GraphicsPath();
        GraphicsPath gp3=new GraphicsPath();
        GraphicsPath gp4=new GraphicsPath();

        //利用路径的 GetBounds 函数获取图元的包围矩形，
        //并且绘出该矩形
        gp1.AddLine(new PointF(10, 10), new PointF(100, 100));
        gp2.AddEllipse(150, 100, 100, 100);
        gp3.AddArc(10, 80, 100, 100, 20, 80);
        string s= "巴蜀秋山形又瘦，岳麓红枫醉几回";
        StringFormat sf = new StringFormat(StringFormatFlags.NoWrap);
        FontFamily f = new FontFamily("隶书");
        gp4.AddString(s, f, 1, 30, new PointF(120, 50), sf);

        RectangleF gb1= gp1.GetBounds();
        RectangleF gb2= gp2.GetBounds();
        RectangleF gb3= gp3.GetBounds();
        RectangleF gb4 = gp4.GetBounds();
        g.DrawPath(Pens.Black, gp1);
        g.DrawRectangle(Pens.Red, gb1.Left, gb1.Top, gb1.Width, gb1.Height);
        g.DrawPath(Pens.Black, gp2);
        g.DrawRectangle(Pens.Red, gb2.Left, gb2.Top, gb2.Width, gb2.Height);
        g.DrawPath(Pens.Black, gp3);
        g.DrawRectangle(Pens.Red, gb3.Left, gb3.Top, gb3.Width, gb3.Height);
        g.DrawPath(Pens.Blue, gp4);
        g.DrawRectangle(Pens.Red, gb4.Left, gb4.Top, gb4.Width, gb4.Height);
    }
```

程序运行结果如图 9-1 所示。图中的红色矩形就是用 GetBounds 属性获得的各图元的包围矩形。

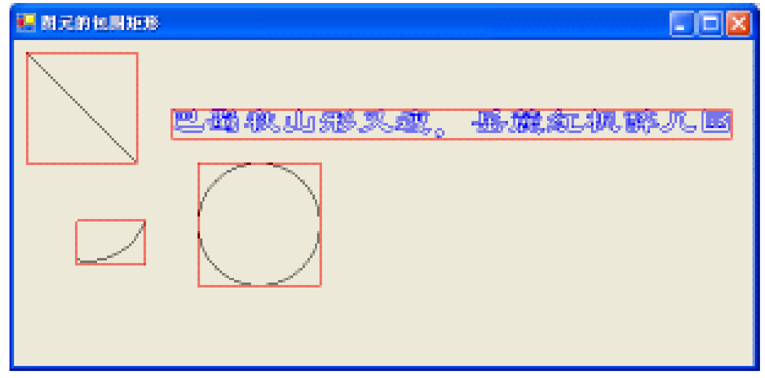

图 9-1 图元的包围矩形

9.2 拾取线形图元

利用 GraphicsPath 类的 IsOutlineVisible 属性，可以拾取线形图元。该属性判断某一点在路径中的可见性。如果可见，则返回 True，否则返回 False。在这里，“可见”的意思也就是说该点位于路径上，即该路径中的图元找到了，被拾取了。

下面这段代码利用 IsOutlineVisible 属性拾取直线段、圆、圆弧和文本等图元。当鼠标光标移动到图元上（即图元被拾取）时，图元用红色虚线表示，否则用黑色实线表示。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics()
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    g.DrawLine(Pens.Black, New PointF(10, 10), New PointF(100, 100))
    g.DrawEllipse(Pens.Black, 150, 100, 100, 100)
    g.DrawArc(Pens.Black, 10, 80, 100, 100, 20, 80)
    Dim s As String = "拾取图元"
    Dim f As Font = New Font("宋体", 20)
    g.DrawString(s, f, Brushes.Black, 120, 50)
End Sub

Private Sub Form1_MouseMove(ByVal sender As Object, _
            ByVal e As MouseEventArgs) Handles MyBase.MouseMove
    Dim aPos As New PointF(e.X, e.Y)
    Dim g As Graphics = CreateGraphics()
    Dim gp1 As New GraphicsPath()
    Dim gp2 As New GraphicsPath()
```

```
Dim gp3 As New GraphicsPath()
Dim gp4 As New GraphicsPath()
gp1.AddLine(New PointF(10, 10), New PointF(100, 100))
gp2.AddEllipse(150, 100, 100, 100)
gp3.AddArc(10, 80, 100, 100, 20, 80)
Dim s As String = "拾取图元"
Dim sf As StringFormat = New StringFormat(StringFormatFlags.NoWrap)
Dim f As FontFamily = New FontFamily("宋体")
gp4.AddString(s, f, 0, 25, New PointF(120, 50), sf)
Dim p1 As New Pen(Color.Black)
Dim p2 As New Pen(Color.Red)
Dim p3 As New Pen(Color.White)
p2.DashStyle = DashStyle.Dash

If gp1.IsOutlineVisible(aPos, Pens.Black) Then
    g.DrawPath(p3, gp1)
    g.DrawPath(p2, gp1)
Else
    g.DrawPath(p1, gp1)
End If
If gp2.IsOutlineVisible(aPos, Pens.Black) Then
    g.DrawPath(p3, gp2)
    g.DrawPath(p2, gp2)
Else
    g.DrawPath(p1, gp2)
End If
If gp3.IsOutlineVisible(aPos, Pens.Black) Then
    g.DrawPath(p3, gp3)
    g.DrawPath(p2, gp3)
Else
    g.DrawPath(p1, gp3)
End If
If gp4.IsOutlineVisible(aPos, Pens.Black) Then
    g.DrawPath(p3, gp4)
    g.DrawPath(p2, gp4)
Else
    g.DrawPath(p1, gp4)
End If

gp1.Dispose()
```

```
        gp2.Dispose()
        gp3.Dispose()
        gp4.Dispose()
        p1.Dispose()
        p2.Dispose()
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        g.DrawLine(Pens.Black, new PointF(10, 10), new PointF(100, 100));
        g.DrawEllipse(Pens.Black, 150, 100, 100, 100);
        g.DrawArc(Pens.Black, 10, 80, 100, 100, 20, 80);
        string s = "拾取图元";
        Font f = new Font("宋体", 20);
        g.DrawString(s, f, Brushes.Black, 120, 50);
    }

    private void Form1_MouseMove(object sender, MouseEventArgs e)
    {
        PointF aPos=new PointF(e.X, e.Y);
        Graphics g = CreateGraphics();
        GraphicsPath gp1=new GraphicsPath();
        GraphicsPath gp2 =new GraphicsPath();
        GraphicsPath gp3=new GraphicsPath();
        GraphicsPath gp4=new GraphicsPath();
        gp1.AddLine(new PointF(10, 10), new PointF(100, 100));
        gp2.AddEllipse(150, 100, 100, 100);
        gp3.AddArc(10, 80, 100, 100, 20, 80);
        string s = "拾取图元";
        StringFormat sf= new StringFormat(StringFormatFlags.NoWrap);
        FontFamily f= new FontFamily("宋体");
        gp4.AddString(s, f, 0, 25, new PointF(120, 50), sf);
        Pen p1=new Pen(Color.Black);
        Pen p2=new Pen(Color.Red);
        Pen p3=new Pen(Color.White);
        p2.DashStyle = DashStyle.Dash;
```

```
if (gp1.IsOutlineVisible(aPos, Pens.Black))
{
        g.DrawPath(p3, gp1);
        g.DrawPath(p2, gp1);
}
else
{
        g.DrawPath(p1, gp1);
}

if (gp2.IsOutlineVisible(aPos, Pens.Black))
{
        g.DrawPath(p3, gp2);
        g.DrawPath(p2, gp2);
}
else
{
        g.DrawPath(p1, gp2);
}

if (gp3.IsOutlineVisible(aPos, Pens.Black))
{
        g.DrawPath(p3, gp3);
        g.DrawPath(p2, gp3);
}
else
{
        g.DrawPath(p1, gp3);
}

if (gp4.IsOutlineVisible(aPos, Pens.Black))
{
        g.DrawPath(p3, gp4);
        g.DrawPath(p2, gp4);
}
else
{
        g.DrawPath(p1, gp1);
}
```

```
        gp1.Dispose();
        gp2.Dispose();
        gp3.Dispose();
        gp4.Dispose();
        p1.Dispose();
        p2.Dispose();
    }
```

程序运行结果如图 9-2(a)、(b)所示。其中图(a)为文本被拾取是的效果，图(b)为圆被拾取时的效果。

(a)

(b)

图 9-2　拾取图元

GraphicsPath 类还有一个 IsVisible 属性，该属性测试点在图元内部的可见性。“图元内部”可以这样理解：比如圆的内部，矩形的内部，或者一段弧，它的弧线和弦围起来的部分，等等。当点在图元内部时，值为 True，否则为 False。

下面的程序测试点在圆和矩形内部的可见性，当点在图元内部时，图元被拾取。被拾取的图元用红色虚线表示。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics()
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim rect As New Rectangle(20, 20, 90, 100)
    g.DrawRectangle(Pens.Black, rect)
    g.DrawEllipse(Pens.Black, 150, 100, 100, 100)
End Sub

Private Sub Form1_MouseMove(ByVal sender As Object, _
        ByVal e As MouseEventArgs) Handles MyBase.MouseMove
    Dim aPos As New PointF(e.X, e.Y)
    Dim g As Graphics = CreateGraphics()
```

```
        Dim gp1 As New GraphicsPath()
        Dim gp2 As New GraphicsPath()
        Dim rect As New Rectangle(20, 20, 90, 100)
        gp1.AddRectangle(rect)
        gp2.AddEllipse(150, 100, 100, 100)
        Dim p1 As New Pen(Color.Black)
        Dim p2 As New Pen(Color.Red)
        Dim p3 As New Pen(Color.White)
        p2.DashStyle = DashStyle.Dash

        If gp1.IsVisible(aPos) Then
            g.DrawPath(p3, gp1)
            g.DrawPath(p2, gp1)
        Else
            g.DrawPath(p1, gp1)
        End If
        If gp2.IsVisible(aPos) Then
            g.DrawPath(p3, gp2)
            g.DrawPath(p2, gp2)
        Else
            g.DrawPath(p1, gp2)
        End If

        gp1.Dispose()
        gp2.Dispose()
        p1.Dispose()
        p2.Dispose()

    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        Rectangle rect=new Rectangle(20, 20, 90, 100);
        g.DrawRectangle(Pens.Black, rect);
        g.DrawEllipse(Pens.Black, 150, 100, 100, 100);
    }
```

```
private void Form1_MouseMove(object sender,MouseEventArgs e)
{
    PointF aPos=new PointF(e.X, e.Y);
    Graphics g = CreateGraphics();
    GraphicsPath gp1=new GraphicsPath();
    GraphicsPath gp2=new GraphicsPath();
    Rectangle rect=new Rectangle(20, 20, 90, 100);
    gp1.AddRectangle(rect);
    gp2.AddEllipse(150, 100, 100, 100);
    Pen p1 =new Pen(Color.Black);
    Pen p2=new Pen(Color.Red);
    Pen p3=new Pen(Color.White);
    p2.DashStyle = DashStyle.Dash;

    if (gp1.IsVisible(aPos))
    {
        g.DrawPath(p3, gp1);
        g.DrawPath(p2, gp1);
    }
    else
    {
        g.DrawPath(p1, gp1);
    }

    if (gp2.IsVisible(aPos))
    {
        g.DrawPath(p3, gp2);
        g.DrawPath(p2, gp2);
    }
    else
    {
        g.DrawPath(p1, gp2);
    }

    gp1.Dispose();
    gp2.Dispose();
    p1.Dispose();
    p2.Dispose();
}
```

程序运行结果如图 9-3 所示。图中鼠标光标位于圆的内部时，圆被拾取，用红色虚线表示。

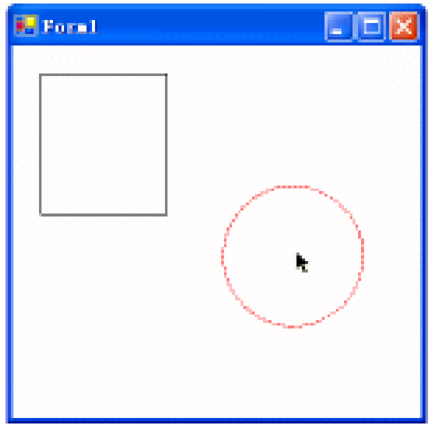

图 9-3 IsVisible 属性测试

9.3 区域的拾取

利用 Region 类的 GetBounds 方法和 IsVisible 属性，可以获取区域的包围矩形和进行可见性测试。

下面这段程序首先创建一个多边形区域，区域内部有一个矩形空洞。用 GetBounds 方法获得它的包围矩形，然后用 IsVisible 属性测试指定的点是否位于区域中。测试结果显示在输出窗口中。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    '定义一个含有多边形的路径
    Dim points As PointF() = { _
        New PointF(10, 10), _
        New PointF(200, 20), _
        New PointF(500, 200), _
        New PointF(300, 250), _
        New PointF(100, 150)}
    Dim gp As New GraphicsPath()
    gp.AddPolygon(points)
    '利用该路径生成多边形区域
    Dim reg As New Region(gp)
    '在该区域中剔除一个矩形区域
    reg.Exclude(New Rectangle(90, 60, 150, 60))
    '绘出最后得到的区域
```

```
        g.FillRegion(Brushes.Blue, reg)
        '得到并绘制多边形区域的包围矩形
        Dim rect As RectangleF = reg.GetBounds(g)
        g.DrawRectangle(Pens.Red, rect.Left, rect.Top, rect.Width, rect.Height)
        '确定下面两个点是否在区域中
        Dim p1 As New PointF(300, 100)
        Dim p2 As New PointF(100, 100)
        InRegion(reg, p1)
        InRegion(reg, p2)
        gp.Dispose()
        reg.Dispose()
    End Sub

    Private Sub InRegion(ByVal reg As Region, ByVal p As PointF)
        '确定某点是否位于指定区域中，并在调试窗口中输出
        If reg.IsVisible(p) Then
            Console.WriteLine("点(" & Str(p.X) & "," & Str(p.Y) & ")在指定区域内")
        Else
            Console.WriteLine("点(" & Str(p.X) & "," & Str(p.Y) & ")不在指定区域内")
        End If
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        //定义一个含有多边形的路径
        PointF[] points= {
                    new PointF(10, 10),
                    new PointF(200, 20),
                    new PointF(500, 200),
                    new PointF(300, 250),
                    new PointF(100, 150)};
        GraphicsPath gp=new GraphicsPath();
        gp.AddPolygon(points);
        //利用该路径生成多边形区域
        Region reg=new Region(gp);
        //在该区域中剔除一个矩形区域
        reg.Exclude(new Rectangle(90, 60, 150, 60));
        //绘出最后得到的区域
        g.FillRegion(Brushes.Blue, reg);
```

```
        //得到并绘制多边形区域的包围矩形
        RectangleF rect= reg.GetBounds(g);
        g.DrawRectangle(Pens.Red, rect.Left, rect.Top, rect.Width, rect.Height);
        //确定下面两个点是否在区域中
        PointF p1=new PointF(300, 100);
        PointF p2=new PointF(100, 100);
        InRegion(reg, p1);
        InRegion(reg, p2);
        gp.Dispose();
        reg.Dispose();
    }

    private void InRegion(Region reg, PointF   p)
    {
        //确定某点是否位于指定区域中，并在调试窗口中输出
        if (reg.IsVisible(p))
        {
            Console.WriteLine("点(" + p.X + "," + p.Y + ")在指定区域内");
        }
        else
        {
            Console.WriteLine("点(" + p.X + "," + p.Y + ")不在指定区域内");
        }
    }
```

程序运行结果如图 9-4 和图 9-5 所示。图 9-4 中，区域为蓝色部分，红色矩形框显示了区域的包围矩形。图 9-5 中显示，点(100,100)不在指定区域内部，点(300,100)在区域内部。

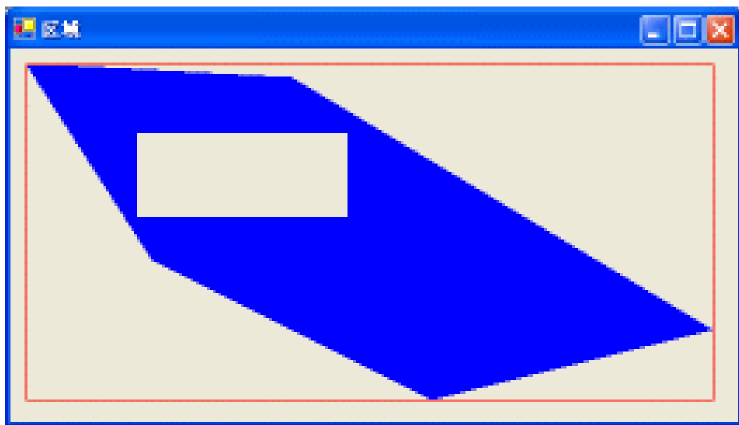

图 9-4　区域的包围矩形

图 9-5　点的可见性测试结果

9.4　图元的复制

利用 GraphicsPath 类的 Clone 方法，可以实现路径图元的复制。下面利用 Clone 方法复制多个矩形区域。

【VB.NET】

```
Protected Overrides Sub OnPaint( _
      ByVal e As System.Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    Dim gp As New GraphicsPath()
    '在路径中添加一个矩形
    gp.AddRectangle(New Rectangle(10, 10, 100, 100))
    '填充矩形
    g.FillPath(Brushes.Blue, gp)
    '复制多个矩形区域
    Dim i As Integer
    For i = 1 To 20
        g.TranslateTransform(10, 10)
        Dim gp2 As New GraphicsPath()
        gp2 = gp.Clone
        g.FillPath(New SolidBrush(Color.FromArgb(150, 10 + i, i * 10, 100 - i * 2)), gp2)
    Next i
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    GraphicsPath gp=new GraphicsPath();
    //在路径中添加一个矩形
    gp.AddRectangle(new Rectangle(10, 10, 100, 100));
    //填充矩形
    g.FillPath(Brushes.Blue, gp);
    //复制多个矩形区域
    for (int i = 1;i<=20;i++)
```

```
        {
            g.TranslateTransform(10, 10);
            GraphicsPath gp2=new GraphicsPath();
            gp2 = (GraphicsPath)gp.Clone();
            g.FillPath(new SolidBrush(Color.FromArgb(150, 10 + i, i * 10,
                          100 - i * 2)), gp2);
        }
    }
```

程序运行结果如图 9-6 所示。

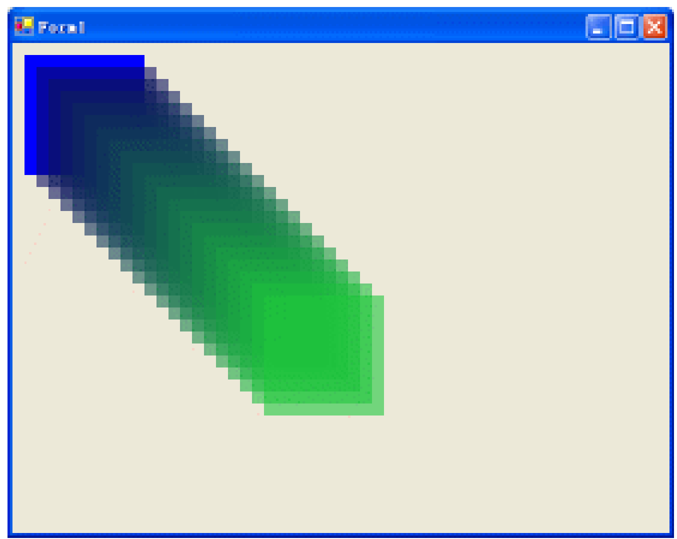

图 9-6　图元的复制

9.5　曲线的拾取

GDI+中提供了绘制贝塞尔曲线和基数样条曲线的方法，下面主要结合基数样条曲线介绍曲线的拾取方法。

基数样条曲线是一种插值拟合曲线，它穿过各个要拟合的数据点。下面的程序绘制一条基数样条曲线。注意，其中用到了 GraphicsPath 类的 PointCount 属性和 PathPoints 属性。这两个属性分别获取路径中控制点的个数和坐标。我们试图用它们找到控制基数样条曲线的控制点，并用红色的圆点来表示它们。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    Curve(g)
End Sub

Public Sub Curve(ByVal g As Graphics)
```

```
        Dim point1 As New Point(10, 100)
        Dim point2 As New Point(80, 20)
        Dim point3 As New Point(150, 300)
        Dim point4 As New Point(200, 100)
        Dim point5 As New Point(300, 200)
        Dim point6 As New Point(500, 150)
        Dim points As Point() = {point1, point2, point3, point4, point5, point6}
        Dim gp As New GraphicsPath()
        gp.AddCurve(points)
        g.DrawPath(New Pen(Color.Black, 2), gp)
        Dim p() As PointF = gp.PathPoints
        Dim i As Integer
        For i = 0 To gp.PointCount - 1
            g.FillEllipse(Brushes.Red, p(i).X - 3, p(i).Y - 3, 6, 6)
        Next
    End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Curve(g);
}

public void Curve(Graphics g)
{
    Point point1=new Point(10, 100);
    Point point2=new Point(80, 20);
    Point point3=new Point(150, 300);
    Point point4=new Point(200, 100);
    Point point5=new Point(300, 200);
    Point point6=new Point(500, 150);
    Point[] points= {point1, point2, point3, point4, point5, point6};
    GraphicsPath gp= new GraphicsPath();
    gp.AddCurve(points);
    g.DrawPath(new Pen(Color.Black, 2), gp);
    PointF[] p= gp.PathPoints;
    for (int i = 0;i<gp.PointCount;i++)
    {
        g.FillEllipse(Brushes.Red, p[i].X - 3, p[i].Y - 3, 6, 6);
    }
}
```

程序运行结果如图 9-7 所示。由图可见，基数样条曲线的控制点都位于曲线的极值点附近，并且每个极值点附近都有 3 个控制点，它们连成一条直线段，该直线段与曲线相切。

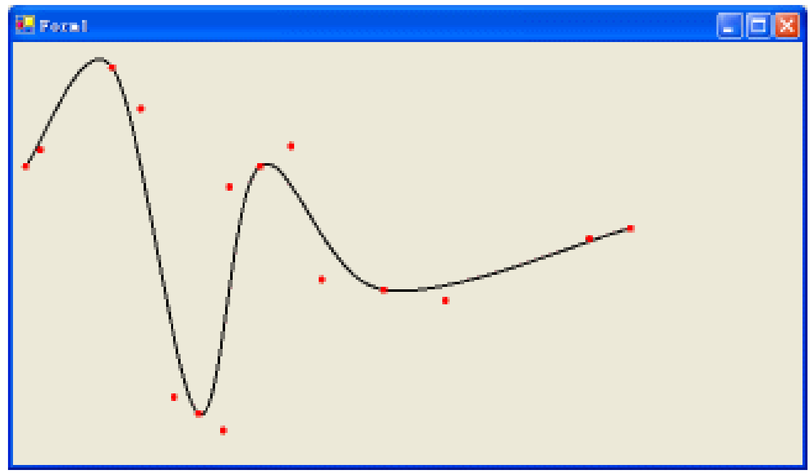

图 9-7　基数样条曲线及其控制点

把曲线添加到一个 GraphicsPath 对象中，然后利用该对象的 GetBounds 方法和 IsOutlineVisible 属性拾取曲线。利用 GetBounds 方法可以首先判断鼠标光标是否位于曲线的包围矩形中，如果在包围矩形中，则进一步用 IsOutlineVisible 属性判断曲线是否被拾取。

下面的示例程序拾取一条基数样条曲线。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim gp As GraphicsPath
    gp = Definegp()
    g.DrawPath(New Pen(Color.Black, 2), gp)
End Sub

Private Sub Form1_MouseMove(ByVal sender As Object, _
        ByVal e As Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
    Dim g As Graphics = CreateGraphics()
    Dim gp As New GraphicsPath()
    gp = Definegp()
    Dim rect As RectangleF = gp.GetBounds
    Dim picked As Boolean = False

    '如果拾取点位于曲线的包围矩形中
    If rect.Contains(e.X, e.Y) Then
        '如果拾取点在曲线的轮廓线上可见
        If gp.IsOutlineVisible(e.X, e.Y, Pens.Black) Then
```

```
                '则曲线被拾取
                picked = True
            End If
        End If
        '曲线被拾取时用红色虚线表示，否则
        '用黑色实线表示
        If picked Then
            g.DrawPath(New Pen(Color.White, 2), gp)
            Dim aPen As New Pen(Color.Red, 2)
            aPen.DashStyle = DashStyle.Dash
            g.DrawPath(aPen, gp)
        Else
            g.DrawPath(New Pen(Color.Black, 2), gp)
        End If
    End Sub

    Private Function Definegp() As GraphicsPath
        Dim gp As New GraphicsPath()
        '定义顶点，添加基数样条曲线到路径
        Dim point1 As New Point(10, 100)
        Dim point2 As New Point(80, 20)
        Dim point3 As New Point(150, 300)
        Dim point4 As New Point(200, 100)
        Dim point5 As New Point(300, 200)
        Dim point6 As New Point(500, 150)
        Dim points As Point() = {point1, point2, point3, point4, point5, point6}
        gp.AddCurve(points)
        Return gp
    End Function
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        GraphicsPath gp;
        gp = Definegp();
        g.DrawPath(new Pen(Color.Black, 2), gp);
    }

    private void Form1_MouseMove(object sender,
            MouseEventArgs e)
```

```
{
    Graphics g = CreateGraphics();
    GraphicsPath gp=new GraphicsPath();
    gp = Definegp();
    RectangleF rect = gp.GetBounds();
    bool picked= false;

    //如果拾取点位于曲线的包围矩形中
    if (rect.Contains(e.X, e.Y))
    {
        //如果拾取点在曲线的轮廓线上可见
        if (gp.IsOutlineVisible(e.X, e.Y, Pens.Black))
        {
            //则曲线被拾取
            picked = true;
        }
    }
    //曲线被拾取时用红色虚线表示，否则
    //用黑色实线表示
    if (picked)
    {
        g.DrawPath(new Pen(Color.White, 2), gp);
        Pen aPen=new Pen(Color.Red, 2);
        aPen.DashStyle = DashStyle.Dash;
        g.DrawPath(aPen, gp);
    }
    else
    {
        g.DrawPath(new Pen(Color.Black, 2), gp);
    }
}

private GraphicsPath Definegp()
{
    GraphicsPath gp=new GraphicsPath();
    //定义顶点，添加基数样条曲线到路径
    Point point1=new Point(10, 100);
    Point point2=new Point(80, 20);
    Point point3=new Point(150, 300);
    Point point4=new Point(200, 100);
    Point point5=new Point(300, 200);
```

```
        Point point6=new Point(500, 150);
        Point[] points= {point1, point2, point3, point4, point5, point6};
        gp.AddCurve(points);
        return gp;
    }
```

程序运行结果如图 9-8 所示。当鼠标光标位于曲线上时，曲线被拾取，且用红色虚线表示。

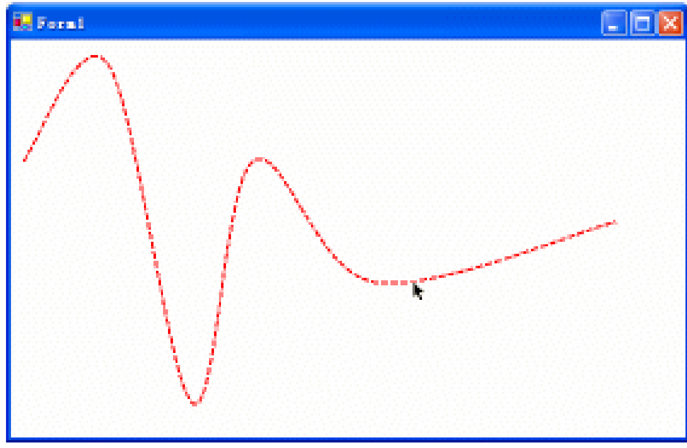

图 9-8　曲线的拾取

9.6　图元变换

在第 3 章已经介绍了利用 Graphics 对象的几何变换方法和 GraphicsPath 对象的一些方法，可以实现图元的几何变换。但是，在进行交互设计时，仅仅把图元画出来是不够的，还要知道变换后图元控制点的位置，并保存它们。

利用 GraphicsPath 类的 PointCount 属性和 PathPoints 属性，可以获得路径中图形控制点的个数和坐标值。下面的程序获取直线段、矩形、圆和圆弧的控制点，并用红色实心圆圈把它们显示出来。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim gp As New GraphicsPath()
    gp.AddLine(50, 50, 100, 100)
    gp.AddRectangle(New Rectangle(190, 50, 100, 100))
    gp.AddEllipse(50, 190, 100, 100)
    gp.AddArc(190, 190, 100, 100, 30, 300)
    g.DrawPath(Pens.Blue, gp)
    Dim p() As PointF = gp.PathPoints
    Dim i As Integer
```

```
        For i = 0 To gp.PointCount - 1
            g.FillEllipse(Brushes.Red, p(i).X - 3, p(i).Y - 3, 6, 6)
        Next
        gp.Dispose()
    End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    GraphicsPath gp=new GraphicsPath();
    gp.AddLine(50, 50, 100, 100);
    gp.AddRectangle(new Rectangle(190, 50, 100, 100));
    gp.AddEllipse(50, 190, 100, 100);
    gp.AddArc(190, 190, 100, 100, 30, 300);
    g.DrawPath(Pens.Blue, gp);
    PointF[] p= gp.PathPoints;
    for (int i = 0;i<=gp.PointCount - 1;i++)
    {
        g.FillEllipse(Brushes.Red, p[i].X - 3, p[i].Y - 3, 6, 6);
    }
    gp.Dispose();
}
```

程序运行结果如图 9-9 所示。从图中可见直线段和矩形的控制点都在顶点处，而圆和圆弧的部分控制点却并不在图元上面。

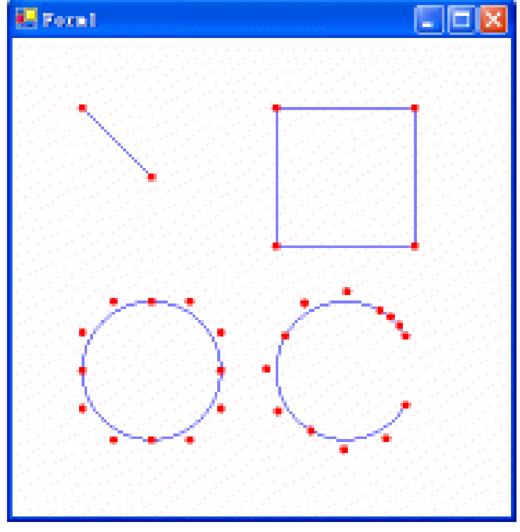

图 9-9　图元的控制点

由于 GraphicsPath 类提供了 PointCount 属性和 PathPoints 属性，所以不管图元如何变换，都能找到图元上的控制点，从而为控制该图元的后续操作提供了可能。

下面的程序采用局部变换的方法，对直线段进行平移、旋转和镜像变换。该程序是 CLine 类的一部分，其他部分与小系统中的基本相同，不再重复。

【VB.NET】

```
'平移变换
Public Overrides Sub Move(ByVal g As Graphics, _
            ByVal basePos As PointF, ByVal desPos As PointF)
    Dim xx, yy As Double
    With desPos
        xx = .X - basePos.X
        yy = .Y - basePos.Y
    End With
    Dim aMatrix As New Matrix()
    Dim gp As New GraphicsPath()
    aMatrix.Translate(xx, yy)
    gp.AddLine(m_Begin.X, m_Begin.Y, m_End.X, m_End.Y)
    gp.Transform(aMatrix)
    Dim p() As PointF = gp.PathPoints
    m_Begin = p(0)
    m_End = p(1)
    gp.Dispose()
End Sub

'旋转变换
Public Overrides Sub Rotate(ByVal g As Graphics, _
           ByVal basePos As PointF, ByVal aAngle As Double)
    Dim gp As New GraphicsPath()
    Dim cossita sinsita, As Double
    cossita = Cos(aAngle)
    sinsita = Sin(aAngle)
    Dim aa, bb As Double
    aa = -basePos.X * cossita + basePos.Y * sinsita + basePos.X
    bb = -basePos.X * sinsita - basePos.Y * cossita + basePos.Y
    Dim aMatrix As New Matrix(cossita, sinsita, -sinsita, cossita, aa, bb)
    gp.AddLine(m_Begin.X, m_Begin.Y, m_End.X, m_End.Y)
    gp.Transform(aMatrix)

    Dim P() As PointF = gp.PathPoints
    m_Begin = P(0)
```

```
        m_End = P(1)

        gp.Dispose()
    End Sub
```

镜像变换

```
    Public Overrides Sub Mirror(ByVal g As Graphics, _
                ByVal pPos1 As PointF, ByVal pPos2 As PointF)
        Dim gp As New GraphicsPath()
        Dim x1, y1, x2, y2 As Double
        x1 = pPos1.X
        y1 = pPos1.Y
        x2 = pPos2.X
        y2 = pPos2.Y
        Dim a, sita As Double
        a = (x2 * y1 - x1 * y2) / (x2 - x1)
        sita = GetAngle(pPos1, pPos2)

        Dim sin2v, cos2v As Double
        sin2v = Sin(sita * 2)
        cos2v = Cos(sita * 2)
        Dim aMatrix As New Matrix(cos2v, sin2v, sin2v, -cos2v, -a * sin2v, a * cos2v + a)
        gp.AddLine(m_Begin.X, m_Begin.Y, m_End.X, m_End.Y)
        gp.Transform(aMatrix)

        Dim p() As PointF = gp.PathPoints()
        m_Begin = p(0)
        m_End = p(1)

        gp.Dispose()
    End Sub
```

【VC#.NET】

```
    //平移变换
    public override void Move(Graphics g, PointF basePos, PointF desPos)
    {
        float xx, yy;
        xx = desPos.X - basePos.X;
        yy = desPos.Y - basePos.Y;
        Matrix aMatrix=new Matrix();
        GraphicsPath gp=new GraphicsPath();
```

```
        aMatrix.Translate(xx, yy);
        gp.AddLine(m_Begin.X, m_Begin.Y, m_End.X, m_End.Y);
        gp.Transform(aMatrix);
        PointF[] p= gp.PathPoints;
        m_Begin = p[0];
        m_End = p[1];
        gp.Dispose();
    }

    //旋转变换
    public override void Rotate(Graphics g, PointF basePos,float aAngle)
    {
        GraphicsPath gp=new GraphicsPath();
        float cossita,sinsita;
        cossita = Math.Cos(aAngle);
        sinsita = Math.Sin(aAngle);
        float aa, bb;
        aa = -basePos.X * cossita + basePos.Y * sinsita + basePos.X;
        bb = -basePos.X * sinsita - basePos.Y * cossita + basePos.Y;
        Matrix aMatrix=new Matrix(cossita, sinsita, -sinsita, cossita, aa, bb);
        gp.AddLine(m_Begin.X, m_Begin.Y, m_End.X, m_End.Y);
        gp.Transform(aMatrix);

        PointF[] P As = gp.PathPoints;
        m_Begin = P[0];
        m_End = P[1];

        gp.Dispose();
    }

    //镜像变换
    public override void Mirror(Graphics g,PointF pPos1, PointF pPos2)
    {
        GraphicsPath gp=new GraphicsPath();
        float x1,y1,x2,y2;
        x1 = pPos1.X;
        y1 = pPos1.Y;
        x2 = pPos2.X;
        y2 = pPos2.Y;
        float a,sita;
        a = (x2 * y1 - x1 * y2) / (x2 - x1);
```

```
        sita = GetAngle(pPos1, pPos2);

        float sin2v, cos2v;
        sin2v = Math.Sin(sita * 2);
        cos2v = Math.Cos(sita * 2);
        Matrix aMatrix=new Matrix(cos2v, sin2v, sin2v, -cos2v, -a * sin2v, a * cos2v + a);
        gp.AddLine(m_Begin.X, m_Begin.Y, m_End.X, m_End.Y);
        gp.Transform(aMatrix);

        PointF[] p = gp.PathPoints;
        m_Begin = p[0];
        m_End = p[1];

        gp.Dispose();

    }
```

第10章　相 交 图 元

在交互式 CAD 程序的设计过程中，要涉及大量图元之间几何关系的计算。比如判断图元是否相交，交点坐标是多少等。与此相关的技术包括相交线、相交弧、图元拾取等。在本套书的 VB 篇中，详细介绍了点、直线段、圆和圆弧之间的几何关系，可以参阅。

10.1　相交线

本节主要讨论一种称为相交线的技术。相交线是一种绘图工具，它经过的图元都被它分割，而且该线本身也被它与所经过的图元的交点所分割。显然，一个功能完整的相交线工具至少需要具备与直线段、椭圆、椭圆弧和区域等的求交能力。

本节讨论最简单的情况，即一条相交线切割一条直线段，只需要计算两条直线段之间的交点就行了。按照上面的定义，一条相交线切割一条直线段以后，两条线会变成 4 条线。因为相交线把直线段分成两部分，而自己也被与直线段的交点分割成两部分。

直线段与直线段之间是否相交，需要首先判断两条直线段所在的直线是否相交，如果相交，则继续判断交点是否位于两条直线段上。如果在直线段上，则该交点是两条直线段的交点，否则不是。

下面的例子首先在窗体上绘两条相交的直线段，然后以交点为分割点，将两条直线段保存为 4 条直线段。当鼠标光标位于这 4 条直线段中的一条上时，该直线段被拾取，用绿色虚线表示。

首先创建 CLline 类，它有一个 Draw 方法和一个 Pick 方法，分别实现直线段绘制和拾取。

【VB.NET】

```
Imports System.Drawing.Drawing2D

Public Class CLine
    Private m_Begin, m_End As PointF

    Public Property LBegin() As PointF
        Get
            Return m_Begin
        End Get
        Set(ByVal Value As PointF)
            m_Begin = Value
        End Set
    End Property
```

```
        Public Property LEnd() As PointF
            Get
                Return m_End
            End Get
            Set(ByVal Value As PointF)
                m_End = Value
            End Set
        End Property

        Public Sub New()

        End Sub

        Public Sub New(ByVal pBegin As PointF, ByVal pEnd As PointF)
            m_Begin = pBegin
            m_End = pEnd
        End Sub

        Public Sub Draw(ByVal g As Graphics, ByVal aPen As Pen)
            g.DrawLine(aPen, m_Begin, m_End)
        End Sub

        Public Function Pick(ByVal p As Point) As Boolean
            Dim gp As New GraphicsPath()
            gp.AddLine(m_Begin, m_End)
            If gp.IsOutlineVisible(p.X, p.Y, Pens.Blue) Then
                Return True
            Else
                Return False
            End If
        End Function
    End Class
```

【VC#.NET】

```
    public class CLine
    {
         private PointF m_Begin, m_End;

         public PointF LBegin
         {
              get{return m_Begin;}
```

```
            set{m_Begin = value;}
        }

        public PointF LEnd
        {
            get{return m_End;}
            set{m_End = value;}
        }

        public CLine()
        {
        }

        public CLine(PointF pBegin, PointF pEnd)
        {
            m_Begin = pBegin;
            m_End = pEnd;
        }

        public void Draw( Graphics g, Pen aPen)
        {
            g.DrawLine(aPen, m_Begin, m_End);
        }

        public bool Pick( Point p)
        {
            GraphicsPath gp=new GraphicsPath();
            gp.AddLine(m_Begin, m_End);
            if (gp.IsOutlineVisible(p.X, p.Y, Pens.Blue))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

然后创建 Module 模块或类，用来提供求取直线段之间交点的运算。该运算用 LineLine 函数实现，其输入为两个 CLline 类对象，返回一个单精度浮点型数组。数组中的第 1 个元素

为交点个数，后面各元素依次为各交点的横、纵坐标。

【VB.NET】

```
Imports System.Math

Module Module1
    Public Function LineLine(ByVal line1 As CLine, ByVal line2 As CLine) As Single()
        Dim reValue(4) As Single
        Dim xx1, yy1, xx2, yy2 As Single
        Dim x1, y1, x2, y2 As Single
        Dim n1, n2, n3, n4 As Single
        Dim k1, k2, c1, c2 As Single

        With line1
            If .LBegin.X > .LEnd.X Then
                xx2 = .LBegin.X
                yy2 = .LBegin.Y
                xx1 = .LEnd.X
                yy1 = .LEnd.Y
            Else
                xx1 = .LBegin.X
                yy1 = .LBegin.Y
                xx2 = .LEnd.X
                yy2 = .LEnd.Y
            End If
        End With

        With line2
            If .LBegin.X > .LEnd.X Then
                x2 = .LBegin.X
                y2 = .LBegin.Y
                x1 = .LEnd.X
                y1 = .LEnd.Y
            Else
                x1 = .LBegin.X
                y1 = .LBegin.Y
                x2 = .LEnd.X
                y2 = .LEnd.Y
            End If
        End With
```

```
        Dim kc1 As Single() = LineKX(New PointF(xx1, yy1), New PointF(xx2, yy2))
        Dim kc2 As Single() = LineKX(New PointF(x1, y1), New PointF(x2, y2))

        '如果两条直线段的斜率相同
        If kc1(0) = kc2(0) Then
            If x1 > xx2 Or x2 < xx1 Then
                reValue(0) = 0
            Else
                n1 = y1 + (-x1) * kc1(0)
                n1 = yy1 + (-xx1) * kc2(0)
                If n1 <> n2 Then
                    reValue(0) = 0
                Else
                    n3 = max(xx1, x1)
                    n4 = min(xx2, x2)
                    reValue(1) = n3
                    reValue(2) = y1 + (n3 - x1) * kc1(0)
                    If (n3 = n4) Then
                        reValue(0) = 1
                    End If
                    reValue(3) = n4
                    reValue(4) = y1 + (n4 - x1) * kc1(0)
                    reValue(0) = 2
                End If
            End If
        Else
            reValue(1) = (kc1(1) - kc2(1)) / (kc2(0) - kc1(0))
            reValue(2) = kc1(1) + reValue(1) * kc1(0)
            '如果交点横坐标在两条直线段的横坐标范围内
            If reValue(1) >= x1 And reValue(1) <= x2 And reValue(1) >= xx1 _
                        And reValue(1) <= xx2 Then
                reValue(0) = 1
            Else
                reValue(0) = 0
            End If
        End If

        Return reValue
    End Function

    '计算直线段的截距式方程
```

```
    Private Function LineKX(ByVal pB As PointF, ByVal pE As PointF) As Single()
        Dim kc(1) As Single
        '若直线段不为竖直线段
        If pB.X <> pE.X Then
            kc(0) = (pE.Y - pB.Y) / (pE.X - pB.X)
            '如果是竖直线段
        Else
            kc(0) = 10000
        End If
        '计算截距
        kc(1) = pB.Y - kc(0) * pB.X
        Return kc
    End Function
End Module
```

【VC#.NET】

```
    public class Module
    {
        public Module()
        {
        }
        public float[] LineLine(CLine line1, CLine line2)
        {
            float[] reValue={0,0,0};
            float xx1, yy1, xx2, yy2;
            float x1, y1, x2, y2;
            float n1, n3, n4;
            float n2=0;

            if (line1.LBegin.X > line1.LEnd.X)
            {
                xx2 =line1.LBegin.X;
                yy2 =line1.LBegin.Y;
                xx1 =line1.LEnd.X;
                yy1 =line1.LEnd.Y;
            }
            else
            {
                xx1 = line1.LBegin.X;
                yy1 = line1.LBegin.Y;
                xx2 = line1.LEnd.X;
```

```
        yy2 = line1.LEnd.Y;
    }

    if (line2.LBegin.X > line2.LEnd.X)
    {
        x2 = line2.LBegin.X;
        y2 = line2.LBegin.Y;
        x1 = line2.LEnd.X;
        y1 = line2.LEnd.Y;
    }
    else
    {
        x1 = line2.LBegin.X;
        y1 = line2.LBegin.Y;
        x2 = line2.LEnd.X;
        y2 = line2.LEnd.Y;
    }

    float[] kc1= LineKX(new PointF(xx1, yy1), new PointF(xx2, yy2));
    float[] kc2= LineKX(new PointF(x1, y1), new PointF(x2, y2));

    //如果两条直线段的斜率相同
    if (kc1[0] == kc2[0])
    {
        if (x1 > xx2 || x2 < xx1)
        {
            reValue[0] = 0;
        }
        else
        {
            n1 = y1 + (-x1) * kc1[0];
            n1 = yy1 + (-xx1) * kc2[0];
            if (n1 != n2)
            {
                reValue[0] = 0;
            }
            else
            {
                n3 = Math.Max(xx1, x1);
                n4 = Math.Min(xx2, x2);
                reValue[1] = n3;
```

```
                    reValue[2] = y1 + (n3 - x1) * kc1[0];
                    if (n3 == n4){reValue[0] = 1;}
                    reValue[3] = n4;
                    reValue[4] = y1 + (n4 - x1) * kc1[0];
                    reValue[0] = 2;
                }
            }
        }
        else
        {
            reValue[1] = (kc1[1] - kc2[1]) / (kc2[0] - kc1[0]);
            reValue[2] = kc1[1] + reValue[1] * kc1[0];
            //如果交点横坐标在两条直线段的横坐标范围内
            if ((reValue[1] >= x1 && reValue[1] <= x2 &&
                reValue[1] >= xx1 && reValue[1] <= xx2))
            {
                reValue[0] = 1;
            }
            else
            {
                reValue[0] = 0;
            }
        }

        return reValue;
    }

    //计算直线段的截距式方程
    private float[] LineKX(PointF pB,PointF pE)
    {
        float[] kc={0,0};
        //若直线段不为竖直线段
        if (pB.X != pE.X)
        {
            kc[0] = (pE.Y - pB.Y) / (pE.X - pB.X);
        }
            //如果是竖直线段
        else
        {
            kc[0] = 10000;
        }
```

```
            //计算截距
            kc[1] = pB.Y - kc[0] * pB.X;
            return kc;
        }
    }
```

最后，在 Form1 类中添加下面的代码。其中，需要创建一个集合类，用来存储分割后的直线段。需要添加重写的 OnMouseMove 方法，当鼠标移动时拾取分割后的直线段。

【VB.NET】

```
Private lines As New ArrayList()

Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Draw(g)
End Sub

Private Sub Draw(ByVal g As Graphics)
    Dim line1 As New CLine(New PointF(10, 10), New PointF(500, 400))
    Dim line2 As New CLine(New PointF(20, 300), New PointF(400, 50))
    line1.Draw(g, Pens.Red)
    line2.Draw(g, Pens.Blue)
    Dim intersec As Single() = LineLine(line1, line2)
    Dim newP As New PointF(intersec(1), intersec(2))

    Dim line3 As New CLine(line1.LBegin, newP)
    Dim line4 As New CLine(newP, line1.LEnd)
    Dim line5 As New CLine(line2.LBegin, newP)
    Dim line6 As New CLine(newP, line2.LEnd)

    lines.Add(line3)
    lines.Add(line4)
    lines.Add(line5)
    lines.Add(line6)
End Sub

Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics
    Dim gp As New GraphicsPath()
    Dim pen1 As New Pen(Color.White)
    Dim pen As New Pen(Color.Green)
```

```
        pen.DashStyle = DashStyle.DashDot
        Dim i As Integer
        For i = 0 To lines.Count - 1
            If lines(i).Pick(New Point(e.X, e.Y)) Then
                lines(i).Draw(g, pen1)
                lines(i).Draw(g, pen)
            End If
        Next
    End Sub
```

【VC#.NET】

```
private ArrayList lines=new ArrayList();

protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    Draw(g);
}

private void Draw(Graphics g)
{
    CLine line1=new CLine(new PointF(10, 10), new PointF(500, 400));
    CLine line2=new CLine(new PointF(20, 300), new PointF(400, 50));
    line1.Draw(g, Pens.Red);
    line2.Draw(g, Pens.Blue);
    float[] intersec= m.LineLine(line1, line2);
    PointF newP=new PointF(intersec[1], intersec[2]);

    CLine line3=new CLine(line1.LBegin, newP);
    CLine line4=new CLine(newP, line1.LEnd);
    CLine line5=new CLine(line2.LBegin, newP);
    CLine line6=new CLine(newP, line2.LEnd);

    lines.Add(line3);
    lines.Add(line4);
    lines.Add(line5);
    lines.Add(line6);
}

protected override void OnMouseMove( MouseEventArgs e)
```

```
{
    Graphics g= this.CreateGraphics();
    GraphicsPath gp=new GraphicsPath();
    Pen pen1=new Pen(Color.White);
    Pen pen=new Pen(Color.Green);
    pen.DashStyle = DashStyle.DashDot;
    for (int i = 0 ;i<=lines.Count - 1;i++)
    {
        if (((CLine)(lines[i])).Pick(new Point(e.X, e.Y)))
        {
            ((CLine)(lines[i])).Draw(g, pen1);
            ((CLine)(lines[i])).Draw(g, pen);
        }
    }
}
```

运行程序，其结果如图 10-1 所示。其中绿色虚线为直线段被拾取以后的显示效果。

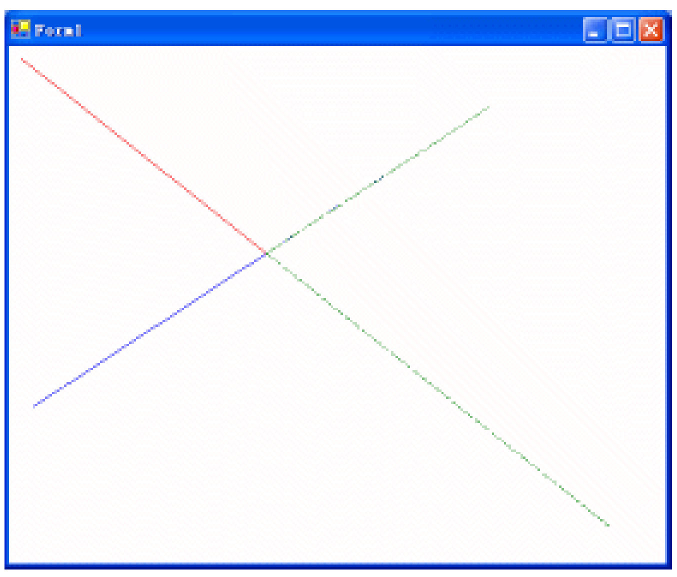

图 10-1　相交直线段

10.2　矩形框拾取

在 AutoCAD 等软件中，我们可以看到用矩形框拾取图元的技巧。当鼠标光标移动时，始终有一个绿色矩形跟着移动，感觉好像鼠标光标上面套了一个框框一样。当矩形框与要拾取的图元相交时，图元被拾取。这种拾取技巧的效果如图 10-2 所示。

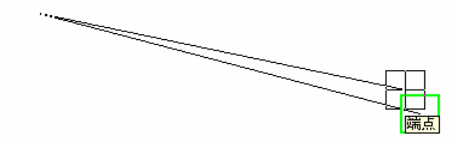

图 10-2　矩形框拾取效果图示

稍一分析就能明白是怎么回事，归根到底还是计算矩形框与其他图元之间的相交关系，相交时图元被拾取，否则不被拾取。

下面我们来实现这个拾取技巧，但只能拾取直线段和圆。首先需要创建一个直线段类 CLline 类和 CCircle 类。CLine 类与前面基本相同，不再重复，可以参见光盘。

下面创建圆类——CCircle 类。

【VB.NET】

```
Public Class CCircle
    Private m_Center, m_PCircle As PointF

    '圆心属性
    Public Property Center() As PointF
        Get
            Return m_Center
        End Get
        Set(ByVal Value As PointF)
            m_Center = Value
        End Set
    End Property

    '圆上一点属性
    Public Property PCircle() As PointF
        Get
            Return m_PCircle
        End Get
        Set(ByVal Value As PointF)
            m_PCircle = Value
        End Set
    End Property

    '半径属性，只读
    Public ReadOnly Property Radius() As Single
        Get
            Dim r As Single= DistPtoP(m_Center, m_PCircle)
            Return r
```

```
            End Get
        End Property

        Public Sub New()
        End Sub

        Public Sub New(ByVal pCenter As PointF, ByVal pCircle As PointF)
            m_Center = pCenter
            m_PCircle = pCircle
        End Sub

        Public Sub Draw(ByVal g As Graphics, ByVal aPen As Pen)
            g.DrawEllipse(aPen, m_Center.X - Radius, m_Center.Y - Radius, _
                          Radius * 2, Radius * 2)
        End Sub

    End Class
```

【VC#.NET】

```
    public class CCircle
    {
        protected PointF m_Center, m_pCircle;
        Module m=new Module();

        public PointF Center
        {
            get{return m_Center;}
            set{m_Center=value;}
        }

        public PointF PCircle
        {
            get{return m_pCircle;}
            set{m_pCircle=value;}
        }

        public float Radius
        {
            get
            {
                float r=m.DistPtoP(m_Center,m_pCircle);
```

```
                    return r;
                }
            }

            public CCircle()
            {
            }

            public CCircle(PointF pCenter,PointF pCircle)
            {
                m_Center=pCenter;
                m_pCircle=pCircle;
            }

            //绘圆
            public void Draw(Graphics g,Pen aPen)
            {
                g.DrawEllipse(aPen, m_Center.X－Radius,
                    m_Center.Y－Radius, Radius * 2, Radius * 2);
            }
        }
```

然后创建 Module 模块或类，在 10.1 节示例代码的基础上添加计算直线段与圆的交点的函数 LineCircle。

【VB.NET】

在 Module 中添加：

```
'圆的一般式方程
Private Function CircleF(ByVal pCenter As PointF, ByVal r As Single) As Single()
    Dim con As Single() = {0, 0, 0}
    con(0) = −2 * pCenter.X
    con(1) = −2 * pCenter.Y
    con(2) = pCenter.X * pCenter.X + pCenter.Y * pCenter.Y－r * r
    Return con
End Function

Public Function LineCircle(ByVal line As CLine, ByVal circle As CCircle) As Single()
    Dim kc As Single() = {0, 0}
    Dim con As Single() = {0, 0, 0}
    Dim interxy As Single() = {0, 0, 0, 0, 0}
    Dim LineMinX, LineMaxX, LineMinY, LineMaxY As Single
    Dim X(1) As Single
```

```
Dim Y(1) As Single
LineMinX = Min(line.LBegin.X, line.LEnd.X)
LineMaxX = Max(line.LBegin.X, line.LEnd.X)
LineMinY = Min(line.LBegin.Y, line.LEnd.Y)
LineMaxY = Max(line.LBegin.Y, line.LEnd.Y)
kc = LineKX(line.LBegin, line.LEnd)
con = CircleF(circle.Center, circle.Radius)
Dim A As Single = 1 + kc(0) * kc(0)
Dim B As Single = 2 * kc(1) * kc(0) + con(1) * kc(0) + con(0)
Dim C As Single = kc(1) * kc(1) + con(1) * kc(1) + con(2)
Dim Root As Single = B * B - 4 * A * C
If Root < 0 Then
    interxy(0) = 0
ElseIf Root = 0 Then
    X(0) = -B / 2 / A
    Y(0) = kc(0) * X(1) + kc(1)
    '如果直线不是竖直线
    If line.LBegin.X <> line.LEnd.X Then
        If X(0) < LineMinX Or X(0) > LineMaxX Then
            interxy(0) = 0
        Else
            interxy(0) = 1
            interxy(1) = X(0)
            interxy(2) = Y(0)
        End If
        '如果直线是竖直线
    Else
        If Y(0) < LineMinY Or Y(0) > LineMaxY Then
            interxy(0) = 0
        Else
            interxy(0) = 1
            interxy(1) = X(0)
            interxy(2) = Y(0)
        End If
    End If
Else
    X(0) = (-B + Sqrt(Root)) / 2 / A
    X(1) = (-B - Sqrt(Root)) / 2 / A
    Y(0) = kc(0) * X(0) + kc(1)
    Y(1) = kc(0) * X(1) + kc(1)
    '如果直线不是竖直线
```

```
If line.LBegin.X <> line.LEnd.X Then
    If (X(0) < LineMinX Or X(0) > LineMaxX) And _
       (X(1) < LineMinX Or X(1) > LineMaxX) Then
        interxy(0) = 0
    ElseIf (X(0) < LineMinX Or X(0) > LineMaxX) And _
           Not (X(1) < LineMinX Or X(1) > LineMaxX) Then
        interxy(0) = 1
        interxy(1) = X(1)
        interxy(2) = Y(1)
    ElseIf Not (X(0) < LineMinX Or X(0) > LineMaxX) And _
           (X(1) < LineMinX Or X(1) > LineMaxX) Then
        interxy(0) = 1
        interxy(1) = X(0)
        interxy(2) = Y(0)
    Else
        interxy(0) = 2
        interxy(1) = X(0)
        interxy(2) = Y(0)
        interxy(3) = X(1)
        interxy(4) = Y(1)
    End If
    如果直线是竖直线
Else
    If (Y(0) < LineMinY Or Y(0) > LineMaxY) And _
       (Y(1) < LineMinY Or Y(1) > LineMaxY) Then
        interxy(0) = 0
    ElseIf (Y(0) < LineMinY Or Y(0) > LineMaxY) And _
           Not (Y(1) < LineMinY Or Y(1) > LineMaxY) Then
        interxy(0) = 1
        interxy(1) = X(1)
        interxy(2) = Y(1)
    ElseIf Not (Y(0) < LineMinY Or Y(0) > LineMaxY) And _
           (Y(1) < LineMinY Or Y(1) > LineMaxY) Then
        interxy(0) = 1
        interxy(1) = X(0)
        interxy(2) = Y(0)
    Else
        interxy(0) = 2
        interxy(1) = X(0)
        interxy(2) = Y(0)
        interxy(3) = X(1)
```

```
                    interxy(4) = Y(1)
                End If
            End If
        End If

        Return interxy
    End Function
```

【VC#.NET】

```
//圆的一般式方程
private float[] CircleF(PointF pCenter,float r)
{
    float[] con = {0, 0, 0};
    con[0] = -2 * pCenter.X;
    con[1] = -2 * pCenter.Y;
    con[2] = pCenter.X * pCenter.X + pCenter.Y * pCenter.Y - r * r;
    return con;
}

//求直线段和圆的交点
public float[] LineCircle(CLine line, CCircle circle)
{
    float[] kc= {0, 0};
    float[] con= {0, 0, 0};
    float[] interxy= {0, 0, 0, 0, 0};
    float LineMinX,LineMaxX,LineMinY,LineMaxY;
    float[] X={0,0};
    float[] Y={0,0};
    LineMinX = Math.Min(line.LBegin.X, line.LEnd.X);
    LineMaxX = Math.Max(line.LBegin.X, line.LEnd.X);
    LineMinY = Math.Min(line.LBegin.Y, line.LEnd.Y);
    LineMaxY = Math.Max(line.LBegin.Y, line.LEnd.Y);
    kc = LineKX(line.LBegin, line.LEnd);
    con = CircleF(circle.Center, circle.Radius);
    float A = 1 + kc[0] * kc[0];
    float B = 2 * kc[1] * kc[0] + con[1] * kc[0] + con[0];
    float C = kc[1] * kc[1] + con[1] * kc[1] + con[2];
    float Root= B * B - 4 * A * C;
    if (Root < 0)
    {
        interxy[0] = 0;
```

```
}
else if (Root == 0)
{
    X[0] = -B / 2 / A;
    Y[0] = kc[0] * X[1] + kc[1];
    //如果直线不是竖直线
    if (line.LBegin.X != line.LEnd.X)
    {
        if (X[0] < LineMinX || X[0] > LineMaxX)
        {
            interxy[0] = 0;
        }
        else
        {
            interxy[0] = 1;
            interxy[1] = X[0];
            interxy[2] = Y[0];
        }
    }
    //如果直线是竖直线
    else
    {
        if (Y[0] < LineMinY || Y[0] > LineMaxY)
        {
            interxy[0] = 0;
        }
        else
        {
            interxy[0] = 1;
            interxy[1] = X[0];
            interxy[2] = Y[0];
        }
    }
}
else
{
    X[0] = (-B + (float)(Math.Sqrt(Root))) / 2 / A;
    X[1] = (-B - (float)(Math.Sqrt(Root))) / 2 / A;
    Y[0] = kc[0] * X[0] + kc[1];
    Y[1] = kc[0] * X[1] + kc[1];
    //如果直线不是竖直线
```

```
if (line.LBegin.X != line.LEnd.X)
{
    if ((X[0] < LineMinX || X[0] > LineMaxX) &&
         (X[1] < LineMinX || X[1] > LineMaxX))
    {
        interxy[0] = 0;}
    else if ((X[0] < LineMinX || X[0] > LineMaxX) &&
        !(X[1] < LineMinX || X[1] > LineMaxX))
    {
        interxy[0] = 1;
        interxy[1] = X[1];
        interxy[2] = Y[1];
    }
    else if (!(X[0] < LineMinX || X[0] > LineMaxX) &&
         (X[1] < LineMinX || X[1] > LineMaxX))
    {
        interxy[0] = 1;
        interxy[1] = X[0];
        interxy[2] = Y[0];
    }
    else
    {
        interxy[0] = 2;
        interxy[1] = X[0];
        interxy[2] = Y[0];
        interxy[3] = X[1];
        interxy[4] = Y[1];
    }
}
//如果直线是竖直线
else
{
    if ((Y[0] < LineMinY || Y[0] > LineMaxY) &&
        (Y[1] < LineMinY || Y[1] > LineMaxY))
    {
        interxy[0] = 0;}
    else if ((Y[0] < LineMinY || Y[0] > LineMaxY) &&
        !(Y[1] < LineMinY || Y[1] > LineMaxY))
    {
        interxy[0] = 1;
        interxy[1] = X[1];
```

```
                    interxy[2] = Y[1];
                }
                else if (!(Y[0] < LineMinY || Y[0] > LineMaxY) &&
                     (Y[1] < LineMinY || Y[1] > LineMaxY))
                {
                    interxy[0] = 1;
                    interxy[1] = X[0];
                    interxy[2] = Y[0];
                }
                else
                {
                    interxy[0] = 2;
                    interxy[1] = X[0];
                    interxy[2] = Y[0];
                    interxy[3] = X[1];
                    interxy[4] = Y[1];
                }
            }
        }
        return interxy;
    }
```

然后在 Form1 **类中添加下面的代码：**

【VB.NET】

```
Private m_preBegin, m_preEnd, m_curBegin, m_curEnd As PointF

Private Sub Draw(ByVal g As Graphics)
    g.DrawEllipse(Pens.Blue, 50, 50, 200, 200)
    g.DrawLine(Pens.Blue, 40, 80, 300, 200)
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Draw(g)
End Sub

Private Sub Form1_MouseMove(ByVal sender As Object, _
        ByVal e As MouseEventArgs) Handles MyBase.MouseMove
    Dim g As Graphics = Me.CreateGraphics()
    Draw(g)
```

```
m_curBegin = New PointF(e.X - 5, e.Y - 5)
m_curEnd = New PointF(e.X + 5, e.Y + 5)
g.DrawRectangle(Pens.White, m_preBegin.X, _
        m_preBegin.Y, 10, 10)
g.DrawRectangle(Pens.Green, m_curBegin.X, _
        m_curBegin.Y, 10, 10)

'把拾取矩形转换为 4 条直线段
Dim line1 As CLine() = {Nothing, Nothing, Nothing, Nothing}
line1(0) = New CLine(m_curBegin, New PointF(m_curBegin.X, m_curEnd.Y))
line1(1) = New CLine(New PointF(m_curBegin.X, m_curEnd.Y), m_curEnd)
line1(2) = New CLine(m_curEnd, New PointF(m_curEnd.X, m_curBegin.Y))
line1(3) = New CLine(New PointF(m_curEnd.X, m_curBegin.Y), m_curBegin)

Dim pickPen As New Pen(Color.Red)
pickPen.DashStyle = DashStyle.Dash

Dim i As Integer
'拾取直线段
For i = 0 To 3
    Dim newline As New CLine(New PointF(40, 80), New PointF(300, 200))
    Dim inter1 As Single()

    inter1 = LineLine(line1(i), newline)
    If inter1(0) > 0 Then
        newline.Draw(g, Pens.White)
        newline.Draw(g, pickPen)
    End If
Next

'拾取圆
For i = 0 To 3
    Dim circle As New CCircle(New PointF(150, 150), New PointF(150, 50))
    Dim inter3 As Single()
    inter3 = LineCircle(line1(i), circle)
    If inter3(0) > 0 Then
        circle.Draw(g, Pens.White)
        circle.Draw(g, pickPen)
    End If
```

```
        Next i

        m_preBegin = m_curBegin
        m_preEnd = m_curEnd
    End Sub
```

【VC#.NET】

```
    private    PointF m_preBegin, m_preEnd;
    private PointF m_curBegin, m_curEnd;
    private Module m=new Module();

    private void Draw( Graphics g)
    {
        g.DrawEllipse(Pens.Blue, 50, 50, 200, 200);
        g.DrawLine(Pens.Blue, 40, 80, 300, 200);
    }

    protected override void OnPaint( PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        Draw(g);
    }

    private void Form1_MouseMove(object sender, MouseEventArgs e )
    {
        Graphics g = this.CreateGraphics();
        Draw(g);
        m_curBegin = new PointF(e.X - 5, e.Y - 5);
        m_curEnd =new PointF(e.X + 5, e.Y + 5);
        g.DrawRectangle(Pens.White, m_preBegin.X,
            m_preBegin.Y, 10, 10);
        g.DrawRectangle(Pens.Green, m_curBegin.X,
            m_curBegin.Y, 10, 10);

        //把拾取矩形转换为 4 条直线段
        CLine[] line1= {null, null, null, null};
        line1[0] = new CLine(m_curBegin, new PointF(m_curBegin.X, m_curEnd.Y));
        line1[1] = new CLine(new PointF(m_curBegin.X, m_curEnd.Y), m_curEnd);
        line1[2] = new CLine(m_curEnd, new PointF(m_curEnd.X, m_curBegin.Y));
```

```
        line1[3] = new CLine(new PointF(m_curEnd.X, m_curBegin.Y), m_curBegin);

        Pen pickPen=new Pen(Color.Red);
        pickPen.DashStyle = DashStyle.Dash;

        //拾取直线段
        for (int i = 0;i<=3;i++)
        {
            CLine newline=new CLine(new PointF(40, 80), new PointF(300, 200));
            float[] inter1;
            inter1 = m.LineLine(line1[i], newline);
            if (inter1[0] > 0)
            {
                newline.Draw(g, Pens.White);
                newline.Draw(g, pickPen);
            }
        }

        //拾取圆
        for (int i = 0;i<=3;i++)
        {
            CCircle circle=new CCircle(new PointF(150, 150), new PointF(150, 50));
            float[] inter3;
            inter3 = m.LineCircle(line1[i], circle);
            if (inter3[0] > 0)
            {
                circle.Draw(g, Pens.White);
                circle.Draw(g, pickPen);
            }
        }

        m_preBegin = m_curBegin;
        m_preEnd = m_curEnd;
    }
```

程序运行结果如图 10-3 所示。鼠标光标移动时，始终有一矩形框随着移动。当矩形框与直线段或圆相交时，直线段或圆被拾取，红色虚线显示。

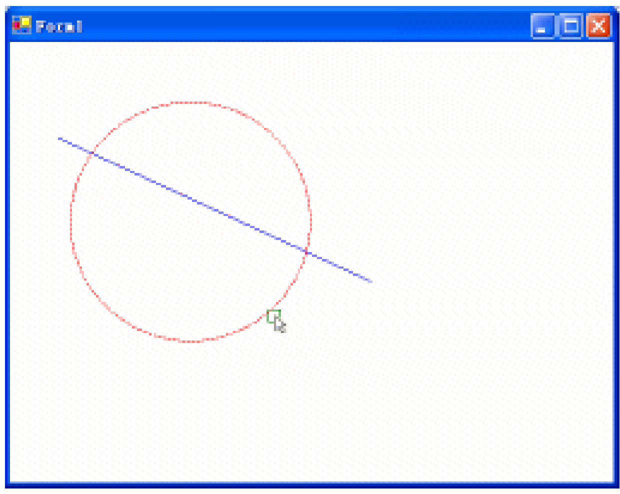

图 10-3 矩形框拾取

10.3 曲线求交

在解决复杂问题时，常常先把复杂问题转化为若干个简单问题，然后进行处理。而关于曲线的问题，常常是首先求出与曲线接近的多义线，然后处理该多义线。

GDI+ 提供了计算与曲线接近的多义线的方法，即使用 GraphicsPath 类的 Flatten 方法可以实现。下面的例子首先获取一段基数样条曲线的近似多义线，然后用红色圆点显示多义线的顶点。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    flatCurve(g)
End Sub

Public Sub flatCurve(ByVal g As Graphics)
    Dim mat As New Matrix()
    mat.Translate(0, 0)
    Dim point1 As New Point(10, 100)
    Dim point2 As New Point(80, 20)
    Dim point3 As New Point(150, 300)
    Dim point4 As New Point(200, 100)
    Dim point5 As New Point(300, 200)
    Dim point6 As New Point(500, 150)
    Dim points As Point() = {point1, point2, point3, point4, point5, point6}
    Dim gp As New GraphicsPath()
    gp.AddCurve(points)
```

```
        g.DrawPath(New Pen(Color.Black, 2), gp)
        gp.Flatten(mat, 10)
        Console.WriteLine(Str(gp.PointCount))
        Dim p() As PointF = gp.PathPoints
        Dim i As Integer
        For i = 0 To gp.PointCount - 1
            g.FillEllipse(Brushes.Red, p(i).X - 3, p(i).Y - 3, 6, 6)
        Next
        g.DrawPath(Pens.Blue, gp)
    End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    flatCurve(g);
}

public void flatCurve(Graphics g)
{
    Matrix mat=new Matrix();
     mat.Translate(0, 0);
    Point point1=new Point(10, 100);
    Point point2=new Point(80, 20);
    Point point3=new Point(150, 300);
    Point point4=new Point(200, 100);
    Point point5=new Point(300, 200);
    Point point6=new Point(500, 150);
    Point[] points= {point1, point2, point3, point4, point5, point6};
    GraphicsPath gp= new GraphicsPath();
    gp.AddCurve(points);
    g.DrawPath(new Pen(Color.Black, 2), gp);
    gp.Flatten(mat, 10);
    Console.WriteLine(gp.PointCount);
    PointF[] p= gp.PathPoints;
    for (int i = 0;i<gp.PointCount;i++)
    {
        g.FillEllipse(Brushes.Red, p[i].X - 3, p[i].Y - 3, 6, 6);
    }
    g.DrawPath(Pens.Blue, gp);
}
```

程序运行结果如图 10-4 所示。

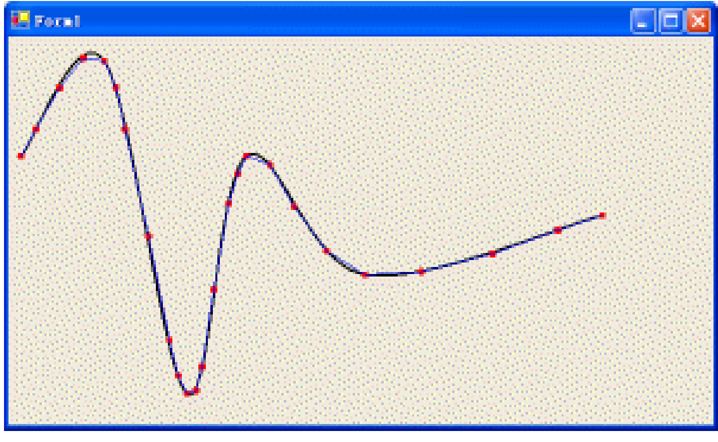

图 10-4 基数样条曲线的近似多义线

把 Flatten 方法的第 2 个参数设置为 5 时，获得的近似多义线的近似程度更高，如图 10-5 所示。

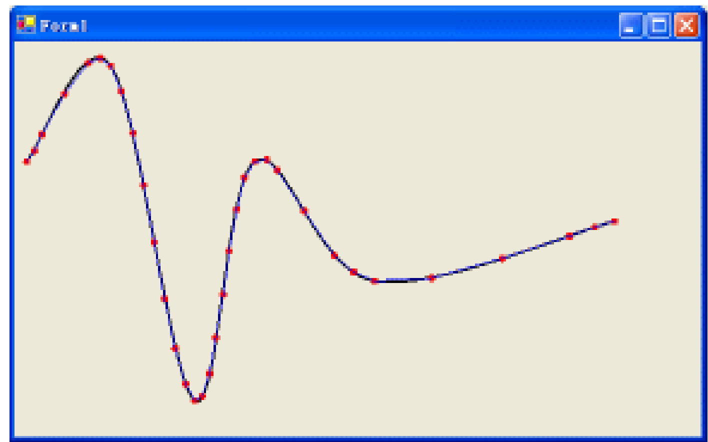

图 10-5 更精确的近似多义线

获得曲线的近似多义线以后，与曲线的求交问题就可以转化为与近似多义线的求交问题。而多义线是由一系列直线段组成的，所以与曲线的求交问题最终转化为与直线段的求交问题。

下面的程序求得曲线与直线段之间的交点以后，用红色圆点进行显示。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
    Dim g As Graphics = e.Graphics
    '定义转换矩阵
    Dim mat As New Matrix()
    mat.Translate(0, 0)
    '定义顶点，添加基数样条曲线到路径
```

```
        Dim point1 As New Point(10, 100)
        Dim point2 As New Point(80, 20)
        Dim point3 As New Point(150, 300)
        Dim point4 As New Point(200, 100)
        Dim point5 As New Point(300, 200)
        Dim point6 As New Point(500, 150)
        Dim points As Point() = {point1, point2, point3, point4, point5, point6}
        Dim gp As New GraphicsPath()
        gp.AddCurve(points)
        g.DrawPath(New Pen(Color.Black, 2), gp)
        '展平曲线
        gp.Flatten(mat, 5)
        '获得路径上的控制点
        Dim p() As PointF = gp.PathPoints
        '给出要与曲线求交的直线段
        Dim line1 As New CLine(New PointF(20, 400), New PointF(400, 50))
        line1.Draw(g, Pens.Blue)
        Dim interNum As Integer = -1
        Dim intersec As Single()
        Dim interPoints(5) As PointF
        Dim i As Integer
        '根据展平以后得到的多条直线段逐条与给定的直线段求交点
        '最终得到的交点即可看成给定直线段与曲线的交点
        For i = 0 To gp.PointCount - 2
            Dim line2 As New CLine(p(i), p(i + 1))
            intersec = LineLine(line1, line2)
            If intersec(0) > 0 Then
                interNum += 1
                interPoints(interNum) = New PointF(intersec(1), intersec(2))
            End If
        Next
        '用红色填充圆表示交点
        For i = 0 To interNum
            g.FillEllipse(Brushes.Red, interPoints(i).X - 3, interPoints(i).Y - 3, 6, 6)
        Next
    End Sub
```

【VC#.NET】

```
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
```

```
//定义转换矩阵
Matrix mat=new Matrix();
mat.Translate(0, 0);
//定义顶点，添加基数样条曲线到路径
Point point1=new Point(10, 100);
Point point2=new Point(80, 20);
Point point3=new Point(150, 300);
Point point4=new Point(200, 100);
Point point5=new Point(300, 200);
Point point6=new Point(500, 150);
Point[] points= {point1, point2, point3, point4, point5, point6};
GraphicsPath gp=new GraphicsPath();
gp.AddCurve(points);
g.DrawPath(new Pen(Color.Black, 2), gp);
//展平曲线
gp.Flatten(mat, 5);
//获得路径上的控制点
PointF[] p= gp.PathPoints;
//给出要与曲线求交的直线段
CLine line1=new CLine(new PointF(20, 400), new PointF(400, 50));
line1.Draw(g, Pens.Blue);
int interNum = -1;
float[] intersec;
PointF pos=new PointF(0,0);
PointF[] interPoints={pos,pos,pos};
//根据展平以后得到的多条直线段逐条与给定的直线段求交点
//最终得到的交点即可看成给定直线段与曲线的交点
Module m=new Module();
for (int i = 0;i<gp.PointCount - 1;i++)
{
    CLine line2=new CLine(p[i], p[i + 1]);
    intersec = m.LineLine(line1, line2);
    if (intersec[0] > 0)
    {
        interNum += 1;
        interPoints[interNum] = new PointF(intersec[1], intersec[2]);
    }
}
//用红色填充圆表示交点
for (int i = 0;i<=interNum;i++)
```

```
            {
                g.FillEllipse(Brushes.Red, interPoints[i].X - 3,
                        interPoints[i].Y - 3, 6, 6);
            }
        }
```

程序运行结果如图 10-6 所示。

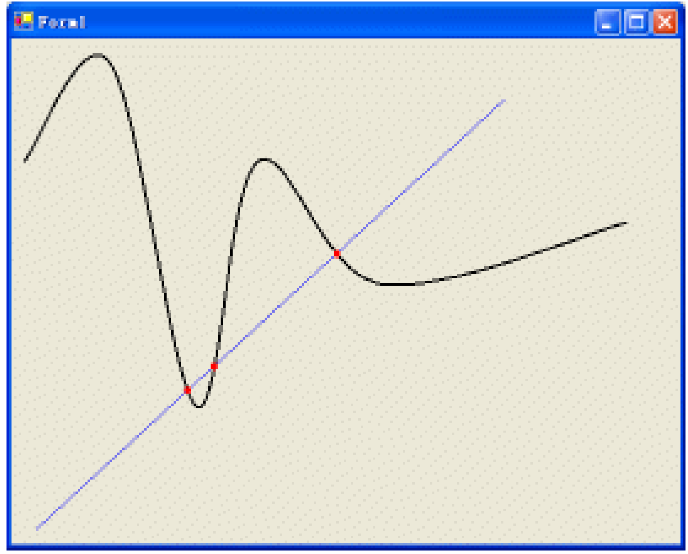

图 10-6　曲线与直线段相交

第 11 章　优 化 处 理

本章讨论一些可以优化前面几章创建的小系统 NET_CAD 的技术和方法，包括创建强键值的集合类、获得以前期绑定方式使用的 For Each、圆整错误及其.NET 修正、用 GDI+函数实现交互绘图、界面美化和数据保存等。

11.1　强键值的集合类

强键值的集合类是这样一种集合类，即它的元素只能是一种对象（这里的键是指可以标示和区分对象的一个属性）。比如一个 CLines 类，规定它只能保存 CLine 对象，即直线段对象，那么它是一个强键值的集合类，它不能保存其他类型的对象。在本套书的 VB 篇中，我们使用的是强键值的集合类，分别创建了各种图元类型的集合类。下面主要介绍为什么要使用强键值的集合类以及如何创建强键值的集合类。

11.1.1　.NET 提供的集合类的缺点

在小系统 NET_CAD 中使用的是 ArrayList 类型的集合类。这种集合类的功能非常强大，可以实现元素的添加、索引和删除，而且任何对象都可以保存。但任何事情都是一分为二的，当我们占据了某一方面的优势并自以为成功时，实际上已在不知不觉中丧失另一方面的优势。ArrayList 就是这样，它对任何对象都来者不拒，拥有足够的灵活性。但是，当要保存的对象类型比较多时，它就可能要付出稳健性方面的代价，就可能出错。

下面略举一例说明 ArrayList 集合类的缺点。创建项目以后，在 Form1 类中添加下面的代码。首先声明一个 ArrayList 类实例 ges，它可以保存各种类型的图元。装载窗体时，创建一个 Point 对象 p1、一个 PointF 对象 p2 和一个 Rectangle 对象 r1，并将它们保存到 ges 集合类中去。然后重写基类的 OnPaint 方法，在窗体上绘制 ges 集合类中的第 3 个图元和第 2 个图元。现在，假设编程者把第 2 个图元的类型记错了，认为它是 Rectangle 类型的，然后使用 Graphics 对象的 DrawRectangle 方法绘制它，结果出错，系统认为无法转换对象类型，或者函数的参数不对。

【VB.NET】

```
Private ges As New ArrayList()

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.DrawRectangle(Pens.Blue, ges(2))
    g.DrawRectangle(Pens.Red, ges(1))
End Sub
```

```
  Private Sub Form1_Load(ByVal sender As System.Object, _
                           ByVal e As System.EventArgs) Handles MyBase.Load
      Dim p1 As New Point(100, 100)
      Dim p2 As New PointF(200, 50)
      Dim r1 As New Rectangle(30, 30, 60, 60)
      ges.Add(p1)
      ges.Add(p2)
      ges.Add(r1)
  End Sub
```

【VC#.NET】

```
private ArrayList ges=new ArrayList();

protected override void OnPaint(PaintEventArgs e)
{
      Graphics g= e.Graphics;
      g.DrawRectangle(Pens.Blue, (Rectangle)(ges[2]));
      g.DrawRectangle(Pens.Red, (ges[1]));
}

private void Form1_Load(object sender,EventArgs   e)
{
      Point p1=new Point(100, 100);
      PointF p2=new PointF(200, 50);
      Rectangle r1=new Rectangle(30, 30, 60, 60);
      ges.Add(p1);
      ges.Add(p2);
      ges.Add(r1);
}
```

11.1.2　创建强键值的集合类

使用强键值的集合类可以避免上面的错误，因为每个存入集合类的对象都要通过身份验证，如果不是专有类型，将不得入内。当然，从集合类中出来的也就都是一种类型的对象了。除了稳健性外，使用 ArrayList 集合类和强键值的集合类中的对象时，在使用方式上还有后期绑定和前期绑定的区别，这在一定程度上会对性能有所影响。这个问题将在后面介绍。

下面首先创建一个 CLine 类，然后创建它的强键值集合类 CLines，最后在 Form1 类中测试它们。

1．创建 CLine 类

下面创建一个简单的 CLine 类，该类具有 5 个属性：ID、Color、Width、LBegin 和 LEnd，

分别表示直线段的编号、颜色、宽度、起点和终点。有一个无参构造函数和一个根据给定参数构造直线段的构造函数。另外还有一个 Draw 方法，实现在绘图表面上绘制直线段。

【VB.NET】

```
Public Class CLine
    Private m_ID,m_Width As Integer
    Private m_Color As Color
    Private m_Begin m_End, As PointF

    Public Property ID() As Integer
        Get
            Return m_ID
        End Get
        Set(ByVal Value As Integer)
            m_ID = Value
        End Set
    End Property

    Public Property Color() As Color
        Get
            Return m_Color
        End Get
        Set(ByVal Value As Color)
            m_Color = Value
        End Set
    End Property

    Public Property Width() As Integer
        Get
            Return m_Width
        End Get
        Set(ByVal Value As Integer)
            m_Width = Value
        End Set
    End Property

    Public Property LBegin() As PointF
        Get
            Return m_Begin
        End Get
        Set(ByVal Value As PointF)
            m_Begin = Value
        End Set
```

```
        End Property

        Public Property LEnd() As PointF
            Get
                Return m_End
            End Get
            Set(ByVal Value As PointF)
                m_End = Value
            End Set
        End Property

        Public Sub New()
        End Sub

        Public Sub New(ByVal aID As Integer, ByVal aColor As Color, _
                ByVal aWidth As Integer, ByVal p1 As PointF, ByVal p2 As PointF)
            m_ID = aID
            m_Color = aColor
            m_Width = aWidth
            m_Begin = p1
            m_End = p2
        End Sub

        Public Sub Draw(ByVal g As Graphics)
            g.DrawLine(New Pen(m_Color, m_Width), m_Begin, m_End)
        End Sub
    End Class
```

【VC#.NET】

```
    public class CLine
    {
        private int m_ID, m_Width;
        private Color m_Color;
        private PointF m_Begin, m_End;

        public int  ID
        {
            get{return m_ID;}
            set{m_ID=value;}
        }

        public Color  Color
        {
```

```
            get{return m_Color;}
            set{m_Color=value;}
        }

        public int    Width
        {
            get{return m_Width;}
            set{m_Width=value;}
        }

        public PointF LBegin
        {
            get{return m_Begin;}
            set{m_Begin=value;}
        }

        public PointF LEnd
        {
            get{return m_End;}
            set{m_End=value;}
        }

        public CLine()
        {
        }

        public CLine(int aID,Color aColor,int aWidth,PointF p1,PointF p2)
        {
            m_ID = aID;
            m_Color = aColor;
            m_Width = aWidth;
            m_Begin = p1;
            m_End = p2;
        }

        public void Draw(Graphics g)
        {
            g.DrawLine(new Pen(m_Color, m_Width), m_Begin, m_End);
        }
    }
```

2．创建 CLines 类

CLines 类继承了 System.Collections 名字空间中的 DictionaryBase 类。DictionaryBase 类

是一个抽象基类，是专门为构建键/值对的强类型集合而提供的。其中，键指的是可以标示和区分对象的一个属性，值指的是对象本身。比如，学号/学生就是一个键/值对，学号是学生这个对象的一个属性，它可以代表和区分学生，而学生本身则是这个值。为了理解 CLines 类，下面我们先好好研究一下 DictionaryBase 类。

打开.NET 类库文档，可以发现 DictionaryBase 类有一个 Count 公共属性和一个受保护的 Dictionary 属性。CLines 类继承了 DictionaryBase 类，所以自动拥有这两个属性。Dictionary 属性的声明如下：

```
Protected ReadOnly Property Dictionary As IDictionary
```

所以它是一个受保护的 IDictionary 类型的只读属性。IDictionary 是一个接口，DictionaryBase 类实现了该接口的 Add 方法、Contains 方法和 Remove 方法。Add 方法将带有指定键和值的元素添加到 DictionaryBase 类中，Contains 方法确定 DictionaryBase 是否包含特定的键，而 Remove 方法则从 DictionaryBase 类中删除指定键值的元素。

下面创建 CLines 类，给它添加一个只读的 Item 属性，其中用到了 Dictionary 属性的 Contains 方法和 Item 属性。Item 属性是 IDictionary 接口的缺省属性，所以可以省略属性名称。添加一个 Add 方法和一个 Remove 方法，它们分别用到 Dictionary 属性的 Add 方法和 Remove 方法。

需要说明的是，CLines 类的 Item 属性值是 CLine 类型的，而 ArrayList 类的 Item 属性值是 Object 型的。所以，如果调用 CLine 类的 Draw 方法，则前者使用的是前期绑定，而后者使用的是后期绑定，代码性能上会有差异。如果是用 VC#.NET，还存在对象类型显式转换的问题。

【VB.NET】

```
Public Class CLines
    Inherits System.Collections.DictionaryBase

    Public Sub New()
    End Sub

    Public Sub Add(ByVal aLine As CLine)
        Me.Dictionary.Add(aLine.ID, aLine)
    End Sub

    Public Sub Remove(ByVal aID As Integer)
        Me.Dictionary.Remove(aID)
    End Sub

    Default Public ReadOnly Property Item(ByVal aID As Integer) As CLine
        Get
            If Me.Dictionary.Contains(aID) Then
                Return CType(Me.Dictionary(aID), CLine)
            Else
                Return Nothing
            End If
```

```
            End Get
        End Property

    End Class
```

【VC#.NET】

```
    public class CLines:System.Collections.DictionaryBase
    {

        public CLines()
        {
        }

        public void Add(CLine aLine)
        {
            this.Dictionary.Add(aLine.ID, aLine);
        }

        public void Remove(int aID)
        {
            this.Dictionary.Remove(aID);
        }

        public CLine this[int aID]
        {
            get{
                if (this.Dictionary.Contains(aID))
                {
                    return (CLine)((this.Dictionary[aID]));
                }
                else
                {
                    return null;
                }
            }
        }
    }
```

3．集合类测试

现在强键值的集合类 CLines 已经有了，它可以实现元素的添加、索引和删除。下面进行测试，看它能不能正常工作。在 Form1 类中，首先创建一个 CLines 类实例 lines，然后生成一个 CreateLines 方法，它创建了 20 条颜色渐变、宽度渐变的直线段，并保存到 lines 中。载

入窗体时，运行 CreateLines **方法，创建并保存直线段。然后重写基类的** OnPaint **方法，遍历** lines **中的对象并把它们绘到窗体上。**

【VB.NET】

```
Private lines As New CLines()
Private aID As Integer
Private pBegin, pEnd As PointF

Private Sub CreateLines()
    Dim line(19) As CLine
    Dim i As Integer
    pBegin.Y = 100
    pEnd.Y = 100
    For i = 0 To 19
        aID = i
        With pBegin
            .X = 0
            .Y += 5 + i + 1
        End With
        With pEnd
            .X = 800
            .Y += 5 + i + 1
        End With
        line(i) = New CLine(aID, Color.FromArgb(200, i * 5, 200, i * 13), _
            i + 1, pBegin, pEnd)
        lines.Add(line(i))
    Next i
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim i As Integer
    For i = 0 To lines.Count － 1
        lines(i).Draw(g)
    Next i
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    CreateLines()
End Sub
```

【VC#.NET】

```
private CLines lines=new CLines();
private int aID;
private PointF pBegin,pEnd;

private void CreateLines()
{
    CLine[] line={null,null,null,null,null,null,null,null,
                  null,null,null,null,null,null,null,null,
                  null,null,null,null};
    pBegin.Y = 100;
    pEnd.Y = 100;
    for (int i = 0;i<=19;i++)
    {
        aID = i;
        pBegin.X = 0;
        pBegin.Y += 5 + i + 1;
        pEnd.X = 800;
        pEnd.Y += 5 + i + 1;
        line[i] = new CLine(aID, Color.FromArgb(200, i * 5, 200, i * 13),
                    i + 1, pBegin, pEnd);
        lines.Add(line[i]);
    }
}

private void Form1_Load(object sender, System.EventArgs e)
{
    CreateLines();
}

protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    for (int i = 0;i<=lines.Count - 1;i++)
    {
        lines[i].Draw(g);
    }
}
```

程序运行结果如图 11-1 所示。

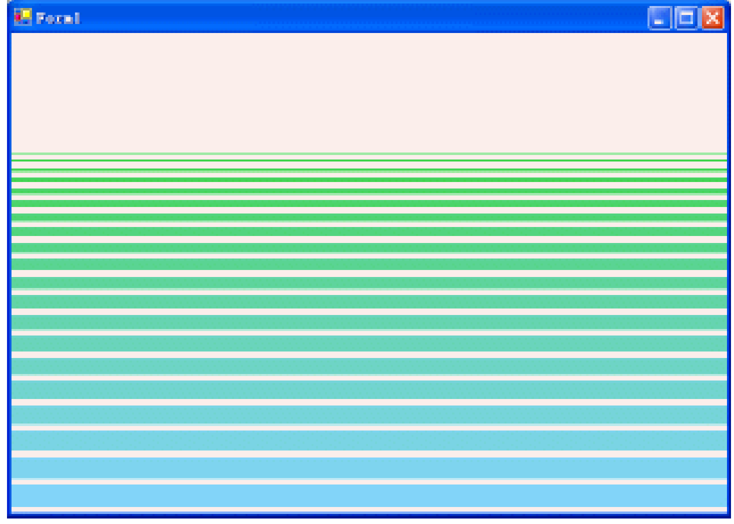

图 11-1　强键值的集合类测试效果

11.2　获得 For Each

11.2.1　以后期绑定方式使用 For Each

如果使用 VB.NET 编程，采用后期绑定方式可以使用 For Each…Next 语句。如下面的代码示例，首先创建一个 CPosition 类和一个 CRectangle 类，然后在 Form1 类中创建一个 ArrayList 类型的实例 ges。将新创建的 CPosition 对象和 CRectangle 对象添加到 ges 中，然后使用 For Each…Next 语句，对 ges 中的每一个对象调用 Draw 方法。因为预先不知道 ges 中对象的类型，将它作为 Object 类型处理，从而采用后期绑定的方式实现绘图。

创建项目，然后添加 CPosition 类，代码如下：

```
Public Class CPosition
    Private m_x, m_y As Single

    Public Property x() As Single
        Get
            Return m_x
        End Get
        Set(ByVal Value As Single)
            m_x = Value
        End Set
    End Property

    Public Property y() As Single
        Get
            Return m_y
```

```
            End Get
            Set(ByVal Value As Single)
                m_y = Value
            End Set
        End Property

        Public Sub New()
        End Sub

        Public Sub New(ByVal xx As Single, ByVal yy As Single)
            m_x = xx
            m_y = yy
        End Sub

        Public Sub Draw(ByVal g As Graphics)
            g.FillEllipse(Brushes.Blue, m_x - 4, m_y - 4, 8, 8)
        End Sub
    End Class
```

然后添加 CRectangle 类，并根据给定的两个点绘矩形。

```
    Imports System.Math
    Public Class CRectangle
        Private m_basePos, m_desPos As CPosition

        Public Property BasePos() As CPosition
            Get
                Return m_basePos
            End Get
            Set(ByVal Value As CPosition)
                m_basePos = Value
            End Set
        End Property

        Public Property DesPos() As CPosition
            Get
                Return m_desPos
            End Get
            Set(ByVal Value As CPosition)
                m_desPos = Value
            End Set
        End Property
```

```
        Public Sub New()
        End Sub

        Public Sub New(ByVal p1 As CPosition, ByVal p2 As CPosition)
            m_basePos = p1
            m_desPos = p2
        End Sub

        Public Sub Draw(ByVal g As Graphics)
            g.DrawRectangle(Pens.Red,  _
                     Min(m_basePos.X, m_desPos.X), Min(m_basePos.Y, m_desPos.Y), _
                     Abs(m_basePos.X - m_desPos.X), Abs(m_basePos.Y - m_desPos.Y))
        End Sub
    End Class
```

定义 CPosition 类和 CRectangle 类以后，在 Form1 类中添加下面的代码，使用 For Each…Next 绘制集合类中的图形。

```
Private ges As ArrayList = New ArrayList()
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim p1 As New CPosition(40, 50)
    Dim p2 As New CPosition(50, 80)
    Dim rect As New CRectangle(New CPosition(20, 10), New CPosition(100, 100))
    ges.Add(p1)
    ges.Add(p2)
    ges.Add(rect)
    Dim obj As Object
    For Each obj In ges
        obj.Draw(g)
    Next
End Sub
```

程序运行结果如图 11-2 所示。

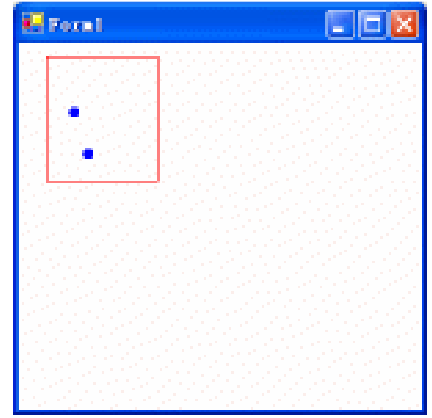

图 11-2　绘制点和矩形

11.2.2　以前期绑定方式使用 For Each

在 11.1 节我们创建了强键值的集合类 CLines，在该节所创建的项目的基础上，添加一个 CLineEnum 类，该类实现了 IEnumerable 接口。IEnumerable 接口只有一个 GetEnumerator 方法，它返回一个实现了 IEnumerator 接口的类的实例。IEnumerator 接口支持在集合上进行简单迭代。它有一个 Current 属性，用

于获取集合中的当前元素；有两个方法，MoveNext 方法将枚举值设置为集合中的下一个元素，Reset 方法将枚举数设置为集合中第 1 个元素的索引值减 1，如果第 1 个元素的索引值为 0，则 Reset 方法将枚举数设置为-1。

为了实现 IEnumerator 接口，需要创建一个内部类 LineEnumerator，它位于 CLineEnum 类的内部，私有。该类实现了 IEnumerator 接口的属性和方法，并有一个用集合类构建实例的构造函数。有了这个内部类，CLineEnum 类就可以实现 IEnumerable 接口的 GetEnumerator 方法，它返回一个参数为 CLines 对象的 LineEnumerator 类实例。其中，作为参数的 CLines 对象是用 CLineEnum 类的公共方法 AddLines 提供的。

CLineEnum 类的代码如下：

【VB.NET】

```
Public Class CLineEnum
    Implements IEnumerable

    Public Sub New()
    End Sub

    Private m_lines As New CLines()
    Public Sub AddLines(ByVal lines As CLines)
        Dim i As Integer
        For i = 0 To lines.Count - 1
            m_lines.Add(lines(i))
        Next
    End Sub

    Private Class LineEnumerator
        Implements IEnumerator

        Dim m_lines As CLines
        Dim m_pos As Integer = -1
        Dim m_Count As Integer

        Public Sub New(ByVal lines As CLines)
            m_lines = lines
            m_Count = lines.Count
        End Sub

        Public Sub Reset() Implements IEnumerator.Reset
            m_pos = -1
        End Sub

    Public Function MoveNext() As Boolean Implements IEnumerator.MoveNext
```

```
                m_pos += 1
                If (m_pos >= m_lines.Count) Then
                    Return False
                Else
                    Return True
                End If
            End Function

            Public ReadOnly Property Current() As Object Implements IEnumerator.Current
                Get
                    Return m_lines(m_pos)
                End Get
            End Property
        End Class

        Public Function GetEnumerator() As IEnumerator Implements IEnumerable.GetEnumerator
            Return New LineEnumerator(m_lines)
        End Function

    End Class
```

【VC#.NET】

```
    public class CLineEnum:IEnumerable
    {
        public CLineEnum()
        {
        }

        private CLines m_lines=new CLines();

        public void AddLines(CLines lines)
        {
            for (int i = 0;i<=lines.Count - 1;i++)
            {
                m_lines.Add(lines[i]);
            }
        }

        private class LineEnumerator:IEnumerator
        {
            CLines m_lines;
            int m_pos = -1;
            int m_Count;
```

```
            public LineEnumerator(CLines lines)
            {
                m_lines = lines;
                m_Count = lines.Count;
            }

            #region Implementation of IEnumerator
            public void Reset()
            {
                m_pos = -1;
            }

            public bool MoveNext()
            {
                m_pos++;
                if (m_pos >= m_lines.Count)
                {
                    return false;
                }
                else
                {
                    return true;
                }
            }

            public object Current
            {
                get
                {
                    return m_lines[m_pos];
                }
            }
            #endregion
        }

        #region Implementation of IEnumerable
        public System.Collections.IEnumerator GetEnumerator()
        {
            return new LineEnumerator(m_lines);
        }
        #endregion
    }
```

添加 CLineEnum 类以后，在 Form1 类中进行测试。键入下面的代码。现在可以使用 For Each…Next 结构了。而且这时进行迭代的对象是已知类型的，调用对象的属性和方法时属于前期绑定方式的用法，效率比较高。

【VB.NET】

```
Private linesE As New CLineEnum()

Private Sub CreateLines()
    Dim line(19) As CLine
    Dim i As Integer
    pBegin.Y = 200
    pEnd.Y = 200
    For i = 0 To 19
        aID = i
        With pBegin
            .X = 0
            .Y += 5 + i + 1
        End With
        With pEnd
            .X = 800
            .Y += 5 + i + 1
        End With
        line(i) = New CLine(aID, Color.FromArgb(200, i * 5, 200, i * 13), _
                    i + 1, pBegin, pEnd)
        lines.Add(line(i))
    Next i
    linesE.AddLines(lines)
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim i As Integer
    Dim line As CLine
    For Each line In linesE
        line.Draw(g)
    Next
End Sub
```

【VC#.NET】

```
private CLineEnum linesE=new CLineEnum();
```

```
private void CreateLines()
{
    CLine[] line={null,null,null,null,null,null,null,null,
                   null,null,null,null,null,null,null,null,
                  null,null,null,null};
    pBegin.Y = 100;
    pEnd.Y = 100;
    for (int i = 0;i<=19;i++)
    {
        aID = i;
        pBegin.X = 0;
        pBegin.Y += 5 + i + 1;
        pEnd.X = 800;
        pEnd.Y += 5 + i + 1;
        line[i] = new CLine(aID, Color.FromArgb(200, i * 5, 200, i * 13),
                    i + 1, pBegin, pEnd);
        lines.Add(line[i]);
    }
    linesE.AddLines(lines);
}

protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    foreach (CLine line in linesE)
    {
        line.Draw(g);
    }
}
```

11.3 圆整错误

前面小系统中为了以简便的方式实现橡皮线效果，大量使用了 API 函数。但 API 绘图函数使用的是整型坐标，在涉及到相交运算等必然要涉及到浮点值的情况下，会导致浮点坐标值向最近的整型坐标值跳跃。这个问题通常称为圆整错误。圆整错误会导致哪些问题呢？比如我们在第 10 章讲到了相交线，一条相交线与一条直线段相交，得到一个交点，然后以这个交点为切割点，将 2 条线变成 4 条线。如果这个交点的坐标值不是整型值（通常情况下都不是），由于使用整型坐标，要圆整，它就可能向离它最近的整型点跳跃，跳跃以后，它就不在原来那个点上了。也就是说，连接该点得到的 4 条直线段的形态与原来的一条相交线与一条

直线段相交的形态相比就有了扭曲，出错了。

使用浮点坐标可以消除圆整错误。由于可以采用浮点值表示坐标，当出现点的坐标值为浮点数时也就不会导致圆整。GDI+中提供了浮点数表示的数据类型，如 PointF, SizeF 和 RectangleF 等。

下面的例子分别在整型坐标和浮点坐标下绘制一条黑色直线段，然后通过计算得到它们的中点并用红色虚线分别连接中点和原直线段的两个端点。为了使它们的差异更明显，将它们放大 15 倍显示。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)

    '整型坐标下的端点
    Dim p1 As New Point(1, 1)
    Dim p2 As New Point(11, 20)
    浮点坐标下的端点
    Dim p3 As New PointF(1, 1)
    Dim p4 As New PointF(11, 20)

    '求以前两点为端点的直线段的中点
    '整形坐标下的中点
    Dim pMiddle1 As New Point((p1.X + p2.X) / 2, (p1.Y + p2.Y) / 2)
    浮点坐标下的中点
    Dim pMiddle2 As New PointF((p3.X + p4.X) / 2, (p3.Y + p4.Y) / 2)

    '放大 15 倍显示
    g.ScaleTransform(15, 15)
    Dim pen1 As Pen = New Pen(Color.Black)
    Dim pen2 As Pen = New Pen(Color.Red)
    '放大后笔宽也放大了，除以放大倍数，保持原宽
    pen1.Width = pen1.Width / 15
    pen2.Width = pen2.Width / 15
    '整型坐标下直接绘制整条直线段
    g.DrawLine(pen1, p1, p2)
    '整型坐标下用虚线分别把直线段的端点与整型坐标下得到的中点相连
    pen2.DashStyle = DashStyle.Dash
    g.DrawLine(pen2, p1, pMiddle1)
    g.DrawLine(pen2, pMiddle1, p2)
    '右移
    g.TranslateTransform(10, 0)
```

```
        浮点坐标下直接绘制整条直线段
        g.DrawLine(pen1, p3, p4)
        浮点坐标下用虚线分别把直线段的端点与整型坐标下得到的中点相连
        pen2.DashStyle = DashStyle.Dash
        g.DrawLine(pen2, p3, pMiddle2)
        g.DrawLine(pen2, pMiddle2, p4)

        pen1.Dispose()
        pen2.Dispose()

    End Sub
```

【VC#.NET】

```
protected override void OnPaint( PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);

    //整型坐标下的端点
    Point p1=new Point(1, 1);
    Point p2=new Point(11, 20);
    //浮点坐标下的端点
    PointF p3=new PointF(1, 1);
    PointF p4=new PointF(11, 20);

    //求以前两点为端点的直线段的中点
    //整形坐标下的中点
    Point pMiddle1=new Point((p1.X + p2.X) / 2, (p1.Y + p2.Y) / 2);
    //浮点坐标下的中点
    PointF pMiddle2=new PointF((p3.X + p4.X) / 2, (p3.Y + p4.Y) / 2);

    //放大 15 倍显示
    g.ScaleTransform(15, 15);
    Pen pen1= new Pen(Color.Black);
    Pen pen2= new Pen(Color.Red);
    //放大后笔宽也放大了，除以放大倍数，保持原宽
    pen1.Width = pen1.Width / 15;
    pen2.Width = pen2.Width / 15;
    //整型坐标下直接绘制整条直线段
    g.DrawLine(pen1, p1, p2);
    //整型坐标下用虚线分别把直线段的端点与整型坐标下得到的中点相连
```

```
        pen2.DashStyle = DashStyle.Dash;
        g.DrawLine(pen2, p1, pMiddle1);
        g.DrawLine(pen2, pMiddle1, p2);
        //右移
        g.TranslateTransform(10, 0);
        //浮点坐标下直接绘制整条直线段
        g.DrawLine(pen1, p3, p4);
        //浮点坐标下用虚线分别把直线段的端点与整型坐标下得到的中点相连
        pen2.DashStyle = DashStyle.Dash;
        g.DrawLine(pen2, p3, pMiddle2);
        g.DrawLine(pen2, pMiddle2, p4);

        pen1.Dispose();
        pen2.Dispose();

    }
```

运行程序，结果如图 11-3 所示。图中两条直线段的差异还是比较明显的。

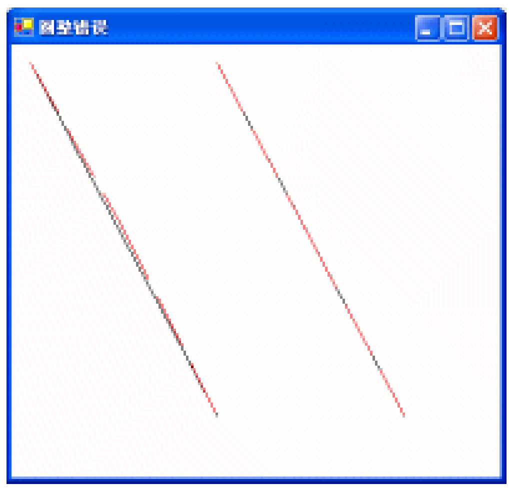

图 11-3 圆整错误演示及修正

11.4 使用 GDI+ 交互绘图

使用 GDI+ 提供的类进行绘图有很多好处，如浮点坐标、反走样等。但要实现交互绘图，橡皮线是一个必不可少的技术。VB.NET 和 VC#.NET 中没有提供设置绘图模式的属性或方法，要实现橡皮线效果必须寻找另外的途径。

利用 Invalidate 方法和 Update 方法可以实现橡皮线效果。橡皮线的绘制过程实际上是一个不断删除前一位置的图形显示和不断绘制当前位置的图形显示的过程。其中关键的问题是只删除前一位置的图形显示，而不删除该图形所经过的已有图元的显示。

使用 Invalidate 方法和 Update 方法实现橡皮线的步骤大致如下。

（1）第 1 次单击鼠标，确定直线段的起点，将直线段的终点也设置为该点，所以初始状态下的直线段是一个点。

（2）第 1 次移动鼠标，调用 Invalidate 方法使移动前直线段（为一个点）包围矩形内的区域失效，用 Update 方法强制重画，清除该点，然后绘制起点到当前点的直线段。

（3）后面各次移动鼠标，调用 Invalidate 方法使移动前直线段包围矩形内的区域失效，用 Update 方法强制重画，清除该直线段，然后绘制起点到当前点的直线段。

（4）第 2 次单击鼠标，确定直线段的终点，绘制直线段并将它保存到集合类中。

已经创建的图元保存在集合类中，并在 OnPaint 方法中进行绘制。使用 Update 方法强制重绘时，将调用 OnPaint 方法，绘制所有已经存在的图元。所以，按照上面的绘图次序，已经存在的图元将总是显示在绘图区内，而不会在删除橡皮线的时候被删除（实际上是被删除后又重绘了）。

下面的代码用 GDI+ 实现了直线段的交互绘制。

1．直线段类

【VB.NET】

```
'直线段类
Imports System.Drawing.Drawing2D

Public Class CLine

    Private m_Begin, m_End As PointF

    '直线段的起点属性
    Public Property LBegin() As PointF
        Get
            Return m_Begin
        End Get
        Set(ByVal Value As PointF)
            m_Begin = Value
        End Set
    End Property

    '直线段的终点属性
    Public Property LEnd() As PointF
        Get
            Return m_End
        End Get
        Set(ByVal Value As PointF)
            m_End = Value
```

```
        End Set
    End Property

    '无参构造函数
    Public Sub New()
    End Sub

    '构造函数，用已知的两点构造直线段
    Public Sub New(ByVal pBegin As PointF, ByVal pEnd As PointF)
        m_Begin = pBegin
        m_End = pEnd
    End Sub

    '绘直线段
    Public Sub Draw(ByVal g As Graphics)
        g.CompositingQuality = CompositingQuality.HighQuality
        g.DrawLine(Pens.Red, m_Begin, m_End)
    End Sub

End Class
```

【VC#.NET】

```
//直线段类
public class CLine
{
    protected PointF m_Begin, m_End;

    public PointF LBegin
    {
        get{return m_Begin;}
        set{m_Begin=value;}
    }

    public PointF LEnd
    {
        get{return m_End;}
        set{m_End=value;}
    }

    public CLine()
    {
```

```
        }

        public CLine(PointF pBegin,PointF pEnd)
        {
            m_Begin=pBegin;
            m_End=pEnd;
        }

        //绘直线段
        public void Draw(Graphics g)
        {
            g.DrawLine(Pens.Red,m_Begin,m_End);
        }
    }
```

2．在 Form1 类中添加下面的代码

【VB.NET】

```
Private m_StartX, m_StartY As Single
Private m_EndX, m_EndY As Single
Private m_Step As Integer = 0
Private lines As New ArrayList()

Protected Overrides Sub OnMouseDown(ByVal e As Windows.Forms.MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics()
    m_Step += 1
    If m_Step = 1 Then
        m_StartX = e.X
        m_StartY = e.Y
        m_EndX = e.X
        m_EndY = e.Y
    ElseIf m_Step = 2 Then
        Dim newline As New CLine(New PointF(m_StartX, m_StartY), _
                New PointF(e.X, e.Y))
        newline.Draw(g)
        lines.Add(newline)
        m_Step = 0
    End If
End Sub

Protected Overrides Sub OnMouseMove(ByVal e As Windows.Forms.MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics()
```

```
        Dim prex, prey As Single
        Dim minX, minY As Single
        Dim maxX, maxY As Single
        If m_Step = 1 Then
            prex = m_EndX
            prey = m_EndY
            minX = Min(m_StartX, m_EndX)
            minY = Min(m_StartY, m_EndY)
            maxX = Max(m_StartX, m_EndX)
            maxY = Max(m_StartY, m_EndY)
            Me.Invalidate(New Rectangle(minX, minY, Abs(m_EndX - m_StartX) + 1, _
                    Abs(m_EndY - m_StartY) + 1))
            Me.Update()
            Dim tempLine As New CLine(New PointF(m_StartX, m_StartY), _
                    New PointF(e.X, e.Y))
            tempLine.Draw(g)
            m_EndX = e.X
            m_EndY = e.Y
        End If
    End Sub

    Protected Overrides Sub OnPaint(ByVal e As Windows.Forms.PaintEventArgs)
        Dim g As Graphics = e.Graphics
        g.FillRectangle(Brushes.White, Me.ClientRectangle)
        Dim i As Integer
        For i = 0 To lines.Count - 1
            lines(i).Draw(g)
        Next
    End Sub
```

【VC#.NET】

```
    private float m_StartX, m_StartY;
    private float m_EndX, m_EndY;
    private int m_Step= 0;
    private ArrayList lines=new ArrayList();

    protected override void OnMouseDown(MouseEventArgs e)
    {
        Graphics g= this.CreateGraphics();
        m_Step += 1;
        if (m_Step == 1)
```

```
        {
            m_StartX = e.X;
            m_StartY = e.Y;
            m_EndX = e.X;
            m_EndY = e.Y;
        }
        else if (m_Step == 2)
        {
            CLine newline=new CLine(new PointF(m_StartX, m_StartY),
                        new PointF(e.X, e.Y));
            newline.Draw(g);
            lines.Add(newline);
            m_Step = 0;
        }
    }

    protected override void OnMouseMove(MouseEventArgs e)
    {
        Graphics g= this.CreateGraphics();
        float prex, prey;
        float minX, minY;
        float maxX, maxY;
        if (m_Step == 1)
        {
            prex = m_EndX;
            prey = m_EndY;
            minX = Math.Min(m_StartX, m_EndX);
            minY = Math.Min(m_StartY, m_EndY);
            maxX = Math.Max(m_StartX, m_EndX);
            maxY = Math.Max(m_StartY, m_EndY);
            this.Invalidate(new Rectangle((int)minX, (int)minY,
                        (int)(Math.Abs(m_EndX - m_StartX)) + 1,
                        (int)(Math.Abs(m_EndY - m_StartY)) + 1));
            this.Update();
            CLine tempLine=new CLine(new PointF(m_StartX, m_StartY),
                         new PointF(e.X, e.Y));
            tempLine.Draw(g);
            m_EndX = e.X;
            m_EndY = e.Y;
        }
    }
```

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    for (int i = 0;i<=lines.Count－1;i++)
    {
        ((CLine)(lines[i])).Draw(g);
    }
}
```

绘图结果如图 11-4 所示。可以和 API 函数的实现效果比较一下，这里绘制时有些闪烁现象，使用 VC#.NET 时更明显。

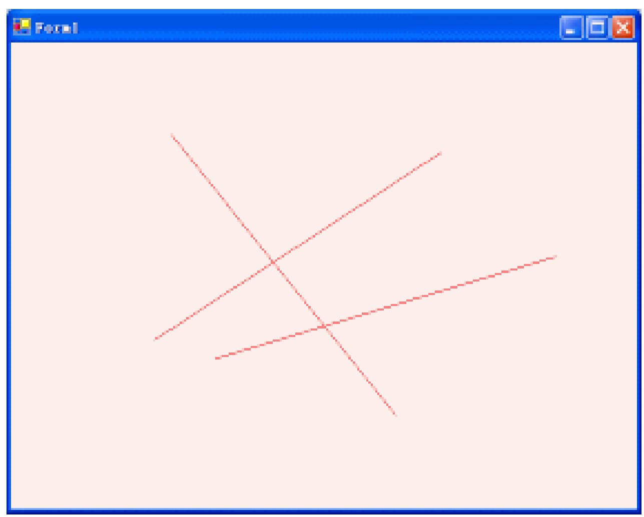

图 11-4　GDI+ 实现直线段的交互绘制

11.5　界面美化

11.5.1　添加工具栏和状态栏

工具栏和状态栏是一般 Windows 程序必不可少的控件。工具栏提供了一系列的命令按钮，这些按钮实际上是部分菜单选项的快捷方式，单击时可以实现指定的任务。状态栏则可以指示程序运行的当前状态，包括鼠标当前位置、运行进度、当前任务、时间等。

1．添加工具栏

打开前面创建的小系统 NET_CAD，在工具箱中单击工具栏命令按钮，然后在窗体中单击或拖曳一下，将在菜单下方显示一个空白的工具栏 ToolBar1。在设计阶段创建工具栏时，如果要在命令按钮上显示图片，还需要创建一个 ImageList 控件，它提供一个图片列表，工具栏与它建立连接以后，从它那里获取图片。

从工具栏中将一个 ImageList 控件的实例 ImageList1 拖曳到窗体中，因为该控件在窗体中不显示，所以它被放到窗体下方的一个空白面板上。在属性编辑器中设置该控件的属性。它的一个关键属性是 Images 属性，单击该属性名所在行中的按钮，将显示“Image 集合编辑器”对话框，如图 11-5 所示。

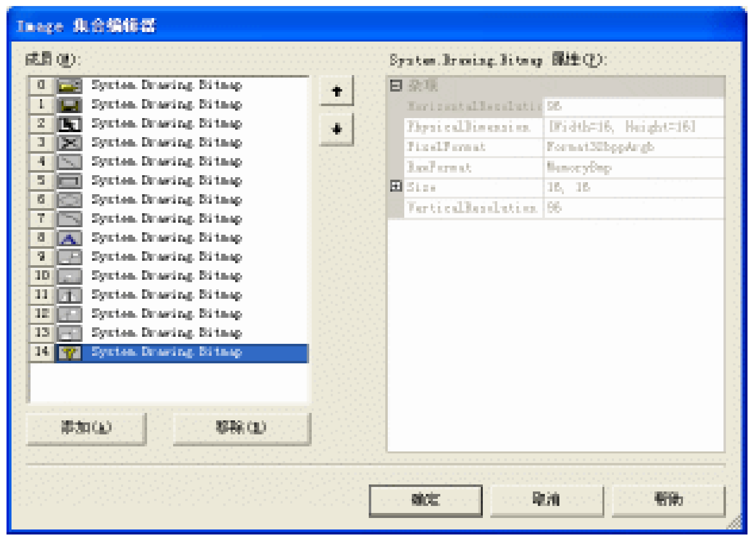

图 11-5　“Image 集合编辑器”对话框

“Image 集合编辑器”的用法一目了然，不必多说。将创建 NET_CAD 程序工具栏所需要的位图创建好以后，逐个添加到“成员”列表框中，单击“确定”按钮。

然后设置工具栏的属性。将 ToolBar1 控件的 ImageList 属性设置为“ImageList1”。单击 Buttons 属性名后面的按钮，打开图 11-6 所示的“ToolBarButton 集合编辑器”添加并设置命令按钮。通过 ImageIndex 属性和 Style 属性分别设置按钮的图片和类型。图片编号与 ImageList1 控件中的图片编号相对应。

图 11-6　ToolBarButton 集合编辑器

单击 Style 属性名后面的按钮，打开一个下拉式列表框，其中列出了命令按钮可以采用的样式，如表 11-1 所示，共 4 种样式。

表 11-1　工具栏中的按钮样式

样 式 名 称	效　果
PushButton	单击以后，按钮会自动弹起
ToggleButton	单击以后，按钮会保持按下或弹起状态
Separator	把不同功能类别的按钮分隔开
DropDownButton	下拉式组合按钮

通过按钮的 ToolTipText 属性，可以设置按钮的功能提示。

在 ToolBar1 控件中添加按钮，按钮名称、图标、功能提示和说明如表 11-2 所示。

表 11-2　工具栏按钮设置和说明

名　称	图　标	提　示	说　明
btnOpen		打开文件	打开 CAD 数据文件
btnSave		保存文件	保存 CAD 数据文件
btnSelect		选择图元	逐个选择图元
btnDelete		删除图元	删除已经选中的图元
btnLine		画直线段	画直线段
btnRectangle		画矩形	画矩形
btnCircle		画圆	画圆
btnArc		画圆弧	画圆弧
btnText	A	文本标注	进行文本标注
btnMove		平移	对选中的图元进行平移变换
btnRotate		旋转	对选中的图元进行旋转变换
btnMirror		镜像	对选中的图元进行镜像变换
btnZoomIn		比例放大	对选中的图元进行比例放大
btnZoomOut		比例缩小	对选中的图元进行比例缩小
btnHelp		关于	显示 About 窗口

另外创建了几个 Separator 型按钮，将几种不同类别的按钮分隔开，可参见光盘。

2．添加状态栏

从工具箱中拖一个 StatusBar 控件的实例到窗体中，缺省名称为 StatusBar1。单击 Panels 属性名后面的按钮，打开 StatusBarPanel 集合编辑器，如图 11-7 所示。在编辑器中进行设置，可以创建和设置状态栏中的面板，如图 11-7 中所示，创建了一个面板。

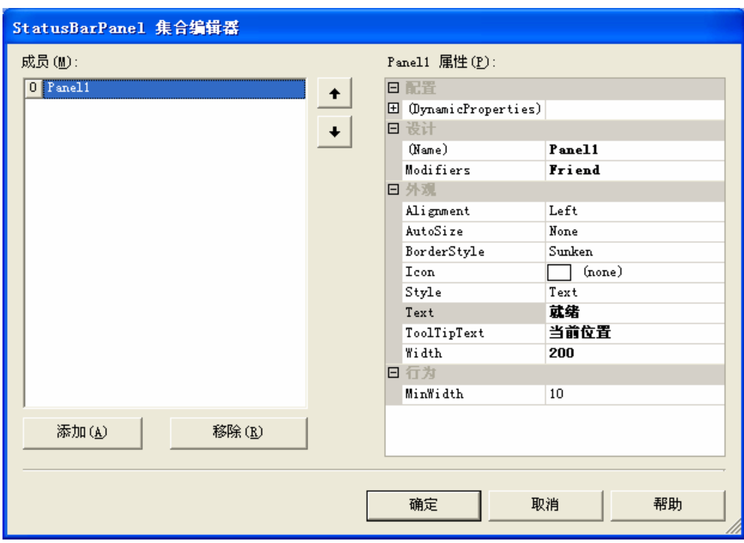

图 11-7　StatusBarPanel 集合编辑器

3．添加事件响应代码

前面已经在项目中创建了一个工具栏和一个状态栏，但它们只有控件的外观，还不能发挥实效，必须用代码把它们与项目中的其他类建立联系才能产生作用。

工具栏的作用是提供菜单项单击事件的快捷方式，首先要做的工作是区分当前鼠标单击了工具栏上的哪个按钮。单击工具栏时，将触发 ButtonClick 事件，根据该事件的 e 参数提供的信息可以得知当前被单击的按钮是哪一个。把各按钮区分开以后，调用对应的菜单项单击事件就行了。

【VB.NET】

```
Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object, _
            ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) _
            Handles ToolBar1.ButtonClick
    If e.Button Is btnOpen Then
        mnuOpen_Click(sender, e)
    ElseIf e.Button Is btnSave Then
        mnuSave_Click(sender, e)
    ElseIf e.Button Is btnSelect Then
        mnuSelect_Click(sender, e)
    ElseIf e.Button Is btnDelete Then
        mnuDelete_Click(sender, e)
    ElseIf e.Button Is btnLine Then
        mnuLine_Click(sender, e)
    ElseIf e.Button Is btnRectangle Then
        mnuRect_Click(sender, e)
    ElseIf e.Button Is btnCircle Then
```

```
            mnuCircle_Click(sender, e)
        ElseIf e.Button Is btnArc Then
            mnuArc_Click(sender, e)
        ElseIf e.Button Is btnText Then
            mnuText_Click(sender, e)
        ElseIf e.Button Is btnMove Then
            mnuTMove_Click(sender, e)
        ElseIf e.Button Is btnRotate Then
            mnuTRotate_Click(sender, e)
        ElseIf e.Button Is btnMirror Then
            mnuTMirror_Click(sender, e)
        ElseIf e.Button Is btnZoomIn Then
            mnuTZoomin_Click(sender, e)
        ElseIf e.Button Is btnZoomOut Then
            mnuTZoomout_Click(sender, e)
        ElseIf e.Button Is btnHelp Then
            mnuAbout_Click(sender, e)
        End If
    End Sub
```

【VC#.NET】

```
private void toolBar1_ButtonClick(object sender,
            System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
    if (e.Button==btnOpen)
    {
        mnuOpen_Click(sender, e);
    }
    else if (e.Button==btnSave)
    {
        mnuSave_Click(sender, e);
    }
    else if (e.Button==btnSelect )
    {
        mnuSelect_Click(sender, e);
    }
    else if (e.Button==btnDelete )
    {
        mnuDelete_Click(sender, e);
    }
    else if (e.Button==btnLine )
```

```
{
    mnuLine_Click(sender, e);
}
else if (e.Button==btnRectangle )
{
    mnuRect_Click(sender, e);
}
else if (e.Button ==btnCircle )
{
    mnuCircle_Click(sender, e);
}
else if (e.Button==btnArc )
{
    mnuArc_Click(sender, e);
}
else if (e.Button==btnText )
{
    mnuText_Click(sender, e);
}
else if (e.Button==btnMove )
{
    mnuMove_Click(sender, e);
}
else if (e.Button==btnRotate )
{
    mnuRotate_Click(sender, e);
}
else if (e.Button==btnMirror)
{
    mnuMirror_Click(sender, e);
}
else if (e.Button==btnZoomIn)
{
    mnuZoomIn_Click(sender, e);
}
else if (e.Button==btnZoomOut)
{
    mnuZoomOut_Click(sender, e);
}
else if (e.Button==btnHelp)
{
```

```
                mnuAbout_Click(sender, e);
            }
        }
```

在 NET_CAD 中，用状态栏显示鼠标的当前位置。在 Form1 类中重写的 OnMouseMove 方法中添加下面的代码：

【VB.NET】

```
    Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
        Dim g As Graphics = Me.CreateGraphics()
        Dim aPos As PointF = PagetoWorld(New PointF(e.X, e.Y))
        StatusBar1.Panels(0).Text = "X=" & Str(aPos.X) & ",    Y=" & Str(aPos.Y)
        aCommand.MouseMove(g, aPos)
    End Sub
```

【VC#.NET】

```
    protected override void OnMouseMove(MouseEventArgs e)
    {
        Graphics g=this.CreateGraphics();
        PointF aPos=m.PagetoWorld(new PointF(e.X, e.Y));
        statusBar1.Panels[0].Text = "X=" + aPos.X + ",    Y=" + aPos.Y;
        aCommand.MouseMove(g,aPos);
    }
```

程序运行结果如图 11-8 所示。

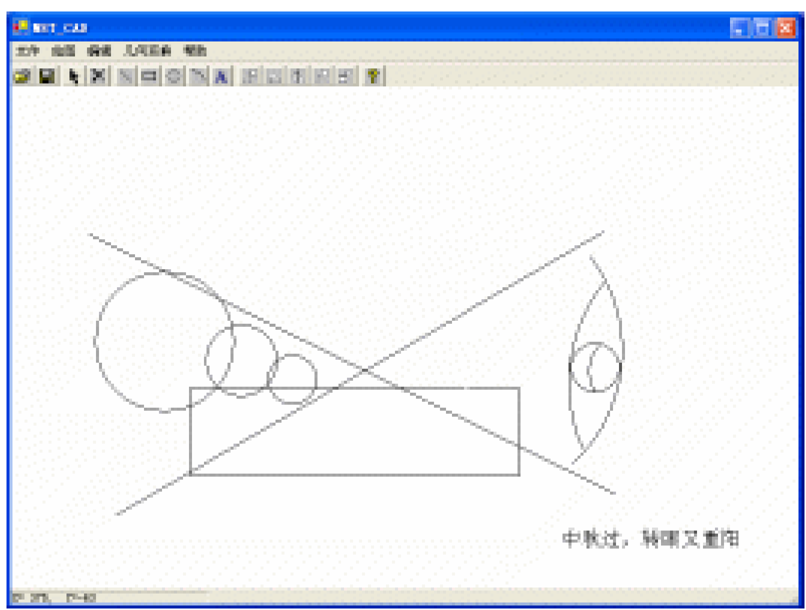

图 11-8　添加工具栏和状态栏以后的程序界面

11.5.2 启动窗口

在很多软件启动时我们都可以看到一个启动窗口，它显示几秒钟以后就消失了。它是一个信息框，但又不是必要的，往往在程序需要比较长的载入时间时添加它。本节在 NET_CAD 中添加一个启动窗口。

添加启动窗口，首先需要在项目中添加一个新的 Windows 窗体，把它命名为 Splash。我们做一个简单的启动窗口，只显示一幅画面，于是需要添加一个图片框。规定窗口显示 2 秒钟，所以需要一个 Timer 控件来定时。另外，窗口还要显示在最上面，并且居中，所以还要设置窗体的属性。Splash 窗体及其控件的属性设置如表 11-3 所示。

表 11-3　Splash 窗体及其控件的属性设置

类　　型	名　　称	属　　性
Form	Splash	FormBorderStyle=None StartPosition=CenterScreen TopMost=True
PictureBox	PictureBox1	Image="我的电脑\共享文档\共享图片\Blue hills.jpg" SizeMode=StretchImage
Timer	Timer1	

Splash 窗体的设计效果如图 11-9 所示。因为将 FormBorderStyle 属性设置为 None，所以窗体没有上面的标题栏。

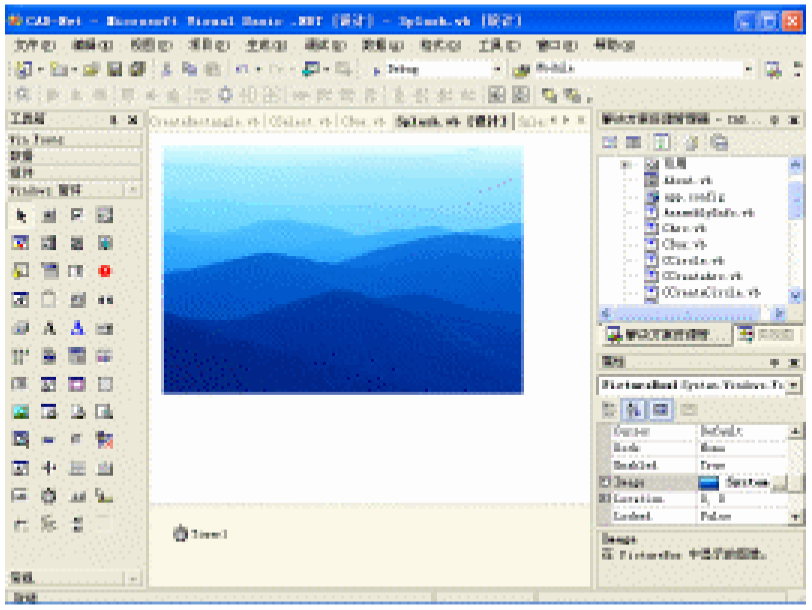

图 11-9　Splash 窗体的设计界面

现在编写代码，在载入窗体时，将 Timer1 控件的时间间隔设置为 2 秒（缺省计时单位为 1/1000s），并开始计时；当触发 Timer 控件的 Tick 事件时关闭窗体。

【VB.NET】

```
Public Sub Splash_Load(ByVal sender As Object, _
                ByVal e As EventArgs) Handles MyBase.Load
    Timer1.Interval = 2000
    Timer1.Start()
End Sub

Public Sub Timer1_Tick(ByVal sender As Object, _
                ByVal e As EventArgs) Handles Timer1.Tick
    Me.Close()
End Sub
```

【VC#.NET】

```
private void Splash_Load (object sender, System.EventArgs e)
{
    timer1.Interval = 2000;
    timer1.Start();
}

private void timer1_Tick (object sender, System.EventArgs e)
{
    this.Close();
}
```

然后在 Form1 类的载入事件中添加代码，引用并显示启动窗口。这里有两个概念要区分：启动窗体和启动窗口。启动窗体指的是程序启动时的主窗体，本例中为 Form1；启动窗口指的是程序运行时首先显示并随即消失的那个窗体。

在 Form1 类中，需要首先创建一个 Splash 类的实例，然后调用它的 ShowDialog 方法，参数为 Me 或 this。表示把 Splash 窗体作为 Form1 窗体的有模式对话框进行显示。有模式对话框关闭以前不能操作其他窗体。

【VB.NET】

```
Dim spForm As New Splash()
spForm.ShowDialog(Me)
```

【VC#.NET】

```
Splash spForm=new Splash();
spForm.ShowDialog(this);
```

现在，运行程序，效果如图 11-10 所示。

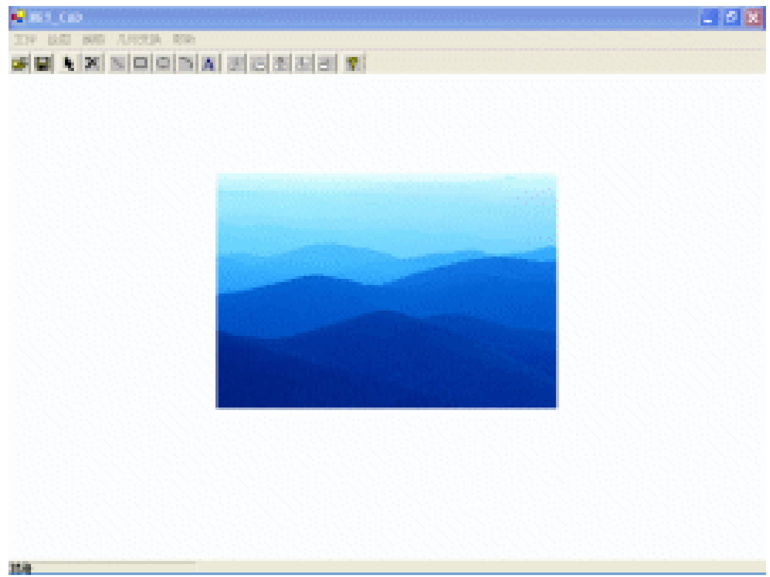

图 11-10　启动窗口

11.5.3　About 窗口

About 窗口常用于进行版权声明，也提供一些其他的信息。同创建启动窗口时一样，也需要首先创建一个新窗体，名称为 About。本例给 NET_CAD 项目中添加一个如图 11-11 所示的窗体，窗体上的控件比较简单，不详细介绍。但需要将窗体的 FormBorderStyle 属性设置为 FixedToolWindow，这样就不能改变窗体的大小了。

图 11-11　About 窗口设计界面

在 About 窗体中单击“确定”按钮时，关闭窗体，其代码如下：

【VB.NET】

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles Button1.Click
    Me.Close()
End Sub
```

【VC#.NET】

```
private void Button1_Click(object sender, System.EventArgs e)
{
    this.Close();
}
```

在 Form1 窗体中单击“帮助”菜单中的“关于”选项时，显示 About 窗口。在 Form1 类中添加下面的代码：

【VB.NET】

```
Private Sub mnuAbout_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles mnuAbout.Click
    Dim about As New About()
    about.ShowDialog(Me)
End Sub
```

【VC#.NET】

```
private void mnuAbout_Click(object sender, System.EventArgs e)
{
    About about=new About();
    about.ShowDialog(this);
}
```

可见，Form1 窗体将 About 窗口同样是作为有模式对话框进行显示的。程序运行效果如图 11-12 所示。

图 11-12 About 窗口

11.6 数据存储

11.6.1 序列化与反序列化

数据通过一定的格式进行存储和传输，而序列化可以将对象的公共字段和私有字段等转换为指定格式的字节流，然后被写入数据流。与序列化相对的是反序列化，它将数据流转换为对象。这两个过程结合起来，就使得数据能够被轻松地存储和传输。

.NET 框架提供的序列化技术包括二进制序列化和 XML 序列化两种，这里主要介绍二进制序列化。缺省条件下，二进制序列化对对象的所有公共字段和私有字段进行转换。实现 ISerializable 接口，可以具体控制转换哪些数据。

下面举一个例子来说明二进制序列化和反序列化。 首先创建一个 CPosition 类，将它声明为可序列化，然后创建一系列随机点，并用一个集合类保存起来。最后对集合类中的点数据进行序列化和反序列化。

1．创建 CPosition 类

创建项目以后，建立一个 CPosition 类，它与前面我们用过的 Position 类等基本相同，不再重复，可以参见光盘。需要指出的是，为了实现序列化，需要在类的定义前指定序列化属性，即

【VB.NET】

```
<Serializable()> Public Class CPosition
    '此处代码省略
End Class
```

【VC#.NET】

```
[Serializable()]public class CPosition
{
    //此处代码省略
}
```

2．创建程序界面

如图 11-13 所示，创建程序界面。在窗体上放置 3 个命令按钮，其名称分别为 btnSave、btnClear 和 btnShow，分别用于保存点数据到文件（序列化）、清屏和显示文件数据所表示的对象（反序列化）。

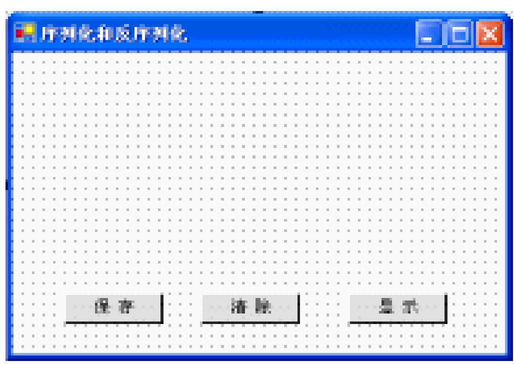

图 11-13　程序界面

3．创建点对象并保存到集合类中

创建一个 ArrayList 类实例 points，用于保存点数据。生成随机点的任务由 AddPoints 方法完成，它创建 50 个随机点。点的横坐标值为小于 500 的正整数，纵坐标值为小于 150 的正

整数。创建的点放到 points 中。

【VB.NET】

```
Private points As New ArrayList()

Private Sub AddPoints()
    Dim i As Integer
    Dim ValueX, ValueY As Single
    For i = 0 To 49
        ValueX = CInt(Int((500 * Rnd()) + 1))
        ValueY = CInt(Int((150 * Rnd()) + 1))
        Console.WriteLine(ValueX)
        Dim p As New CPosition(ValueX, ValueY)
        points.Add(p)
    Next
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    Draw(g)
End Sub

Private Sub Draw(ByVal g As Graphics)
    Dim i As Integer
    For i = 0 To points.Count - 1
        points(i).Draw(g)
    Next
End Sub
```

【VC#.NET】

```
private ArrayList points=new ArrayList();

private void AddPoints()
{
    System.Random rp=new System.Random();
    for (int i = 0;i<=49;i++)
    {
        float ValueX=rp.Next(500);
        float ValueY=rp.Next(150);
        Console.WriteLine(ValueX);
        CPosition p=new CPosition(ValueX, ValueY);
        points.Add(p);
```

```
        }
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        Draw(g);
    }

    private void Draw(Graphics g)
    {
        for (int i = 0;i<=points.Count - 1;i++)
        {
            ((CPosition)(points[i])).Draw(g);
        }
    }

    private void Form1_Load (object sender, System.EventArgs e)
    {
        AddPoints();
        Console.WriteLine(points.Count );
    }
```

4．序列化

首先在 Form1 类中创建一个 FileName 字符串，由应用程序的启动路径和文件名“Points.Pnt”组成。Pnt 为指定的扩展名。然后创建单击“保存”按钮的事件代码。序列化对象要用到 BinaryFormatter 对象的 Serialize 方法。

【VB.NET】

```
Private FileName As String = Application.StartupPath & "\Points.Pnt"

Private Sub btnSave_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnSave.Click
    Dim stream As _
        New FileStream(FileName, System.IO.FileMode.Create)
    Dim binary As New BinaryFormatter()
    binary.Serialize(stream, points)
    stream.Close()
End Sub
```

【VC#.NET】

```
private string FileName = Application.StartupPath + "\\Points.Pnt";
```

```
private void btnSave_Click_1(object sender, System.EventArgs e)
{
      FileStream stream=new System.IO.FileStream(FileName,
                  System.IO.FileMode.Create);
      BinaryFormatter binary=new BinaryFormatter();
      binary.Serialize(stream, points);
      stream.Close();
}
```

5．反序列化

在 Form1 类中添加下面的代码，实现对 Points.Pnt 文件中数据的反序列化。使用 BinaryFormatter 对象的 Deserialize 方法进行反序列化，并将反序列化后得到的对象转换为 ArrayList 对象。

【VB.NET】

```
Private Sub btnShow_Click(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles btnShow.Click
    Dim stream As New System.IO.FileStream(FileName, _
        System.IO.FileMode.Open)
    Dim binary As New BinaryFormatter()
    points = CType(binary.Deserialize(stream), ArrayList)
    stream.Close()
    Dim g As Graphics = Me.CreateGraphics
    Draw(g)
End Sub
```

【VC#.NET】

```
private void btnShow_Click (object sender, System.EventArgs e)
{
      FileStream stream=new FileStream(FileName,
                        System.IO.FileMode.Open);
      BinaryFormatter binary=new BinaryFormatter();
      points = (ArrayList)(binary.Deserialize(stream));
      stream.Close();
      Graphics g = this.CreateGraphics();
      Draw(g);
}
```

6．清屏

调用 Graphics 对象的 Clear 方法，用白色清除窗体上的所有图形。

【VB.NET】

```
Private Sub btnClear_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnClear.Click
    Dim g As Graphics = Me.CreateGraphics()
    g.Clear(Color.White)
End Sub
```

【VC#.NET】

```
private void btnClear_Click (object sender, System.EventArgs e)
{
    Graphics g = this.CreateGraphics();
    g.Clear(Color.White);
}
```

程序运行效果如图 11-14 所示。载入窗体时，在窗体上显示随机点。单击“保存”按钮，将点对象序列化到文件中。单击“清除”按钮，在屏幕上清除随机点。单击“显示”按钮，将文件中的数据反序列化为点对象并显示在窗体上。

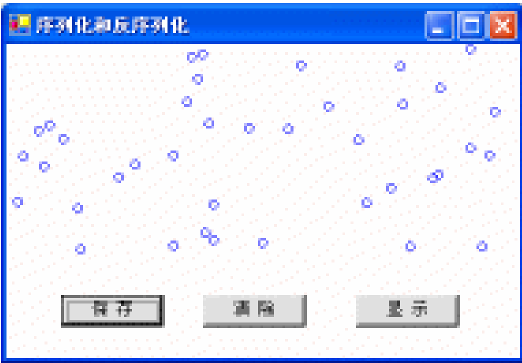

图 11-14　序列化和反序列化演示

11.6.2　CAD 图形数据的序列化和反序列化

对 NET_CAD 中的图元数据进行序列化，首先需要对各个图元类添加可序列化属性，其代码如下：

【VB.NET】

```
<Serializable()> Public MustInherit Class CGElement
    '……
End Class

<Serializable()> Public Class CLine
    Inherits CGElement
    '……
End Class
```

```
<Serializable()> Public Class CRectangle
    Inherits CGElement
    '……
End Class

<Serializable()> Public Class CCircle
    Inherits CGElement
    '……
End Class

<Serializable()> Public Class CArc
    Inherits CGElement
    '……
End Class

<Serializable()> Public Class CText
    Inherits CGElement
    '……
End Class
```

【VC#.NET】

```
[Serializable()] public abstract class CGElement
{
    //……
}

//直线段类
[Serializable()]public class CLine:CGElement
{
    //……
}

//矩形类
[Serializable()]public class CRectangle:CGElement
{
    //……
}

//圆类
[Serializable()]public class CCircle:CGElement
{
```

```
    //……
}

//圆弧类
[Serializable()]public class CArc:CGElement
{
    //……
}

//文本类
[Serializable()]public class CText:CGElement
{
    //……
}
```

下面，在 Form1 类中添加保存数据到文件的事件代码。单击保存数据的菜单项或命令按钮时，打开一个保存文件的对话框，文件的扩展名规定为 .CAD，然后将各对象序列化到指定的文件中。

【VB.NET】

```
Private Sub mnuSave_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles mnuSave.Click
    Dim dlg As New SaveFileDialog()
    dlg.Filter = "CAD 文件 (*.CAD)|*.CAD|所有文件 (*.*)|*.*"
    dlg.ShowDialog(Me)
    Dim FileName As String = dlg.FileName
    Dim stream As New FileStream(FileName, FileMode.Create)
    Dim binary As New BinaryFormatter()
    binary.Serialize(stream, ges)
    stream.Close()
End Sub
```

【VC#.NET】

```
private void mnuSave_Click(object sender, System.EventArgs e)
{
    SaveFileDialog dlg=new SaveFileDialog();
    dlg.Filter="CAD 文件 (*.CAD)|*.CAD|所有文件 (*.*)|*.*";
    dlg.ShowDialog(this);
    string FileName=dlg.FileName;
    FileStream stream=new FileStream(FileName, FileMode.Create);
    BinaryFormatter binary=new BinaryFormatter();
    binary.Serialize(stream, ges);
    stream.Close();
}
```

图 11-15 为保存数据到文件时的对话框，其中保存类型指定为 CAD 格式。

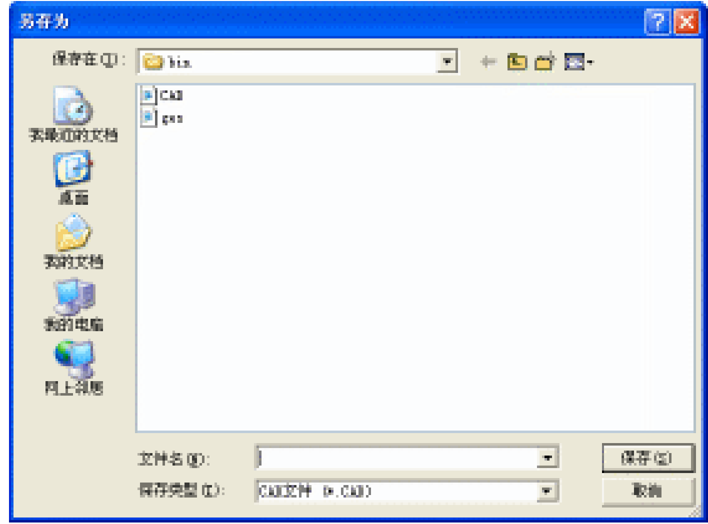

图 11-15 保存文件对话框

打开 CAD 格式的文件时，进行反序列化。单击打开文件的菜单项或命令按钮时，打开一个对话框，要求指定一个文件。然后将该文件中的数据反序列化为图元对象，并绘制到窗体上。

【VB.NET】

```
Private Sub mnuOpen_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles mnuOpen.Click
    Dim g As Graphics = Me.CreateGraphics()
    Dim dlg As New OpenFileDialog()
    dlg.Filter = "CAD 文件 (*.CAD)|*.CAD|所有文件 (*.*)|*.*"
    dlg.ShowDialog(Me)
    Dim FileName As String = dlg.FileName
    Dim stream As New FileStream(FileName, FileMode.Open)
    Dim binary As New BinaryFormatter()
    ges = CType(binary.Deserialize(stream), ArrayList)
    stream.Close()
    DrawAll(g)
End Sub
```

【VC#.NET】

```
private void mnuOpen_Click(object sender, System.EventArgs e)
{
    Graphics g = this.CreateGraphics();
    OpenFileDialog dlg=new OpenFileDialog();
    dlg.Filter="CAD 文件 (*.CAD)|*.CAD|所有文件 (*.*)|*.*";
```

```
            dlg.ShowDialog(this);
            string FileName=dlg.FileName;
            FileStream stream=new FileStream(FileName, FileMode.Open);
            BinaryFormatter binary =new BinaryFormatter();
            ges = (ArrayList)(binary.Deserialize(stream));
            stream.Close();
            m.DrawAll(g,ges);
        }
```

如图 11-16 所示，打开文件时，从对话框中指定一个 CAD 格式的文件，单击“打开”按钮，显示图元。

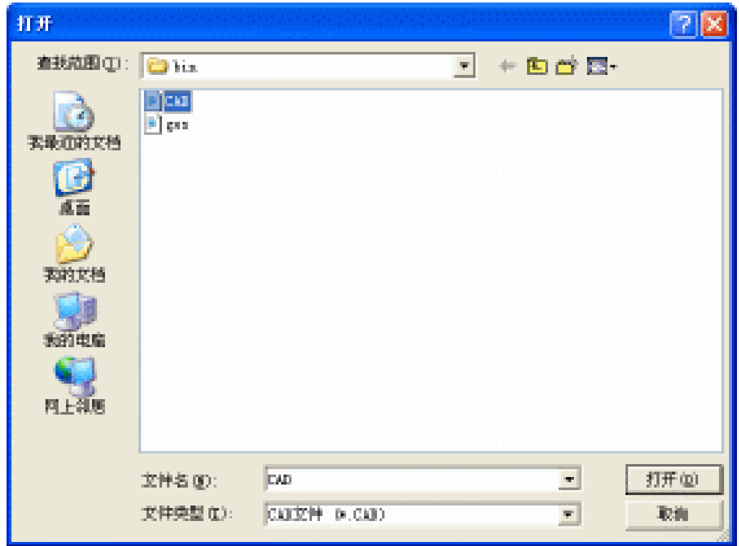

图 11-16　打开文件对话框

第 12 章　设计模式讨论

设计模式是当前讨论得比较多的一种面向对象编程方法，是人们在长期的编程实践中发现很多问题具有相似的解决方法，并搜集、整理和总结这些方法以后慢慢形成的。它是经验的总结，也是创造性的成果。当然，并不是一开始就有模式，是经过坚持不懈的追求和探索，经过千百次的挫折和失败以后逐渐形成的被证明确实可以解决某方面问题的一些成功的样式。灵感来源于生活，很多通灵的东西就附着在、隐藏在我们的生活表象中，当我们偶尔捕捉到它，那一刻我们就是大师！

本章结合 CAD 编程技巧讨论状态模式、访问者模式、模板方法模式、策略模式等几个模式的应用。

12.1　状态模式

在交互式的绘图程序中，当我们单击不同的菜单项或命令按钮以后，同样是单击或移动鼠标会有不同的绘图行为。比如，单击绘制直线段的按钮以后，用鼠标可以绘一条直线段；单击绘制圆弧的按钮以后，用鼠标可以绘制一条圆弧。程序的背后好像有一支无形的手在进行操控。那么这种绘图行为的转换是如何实现的呢？状态模式提供了一种解决方案。

状态模式为每个绘图状态创建一个基本的状态对象子类，并且随着绘图任务的改变在不同的状态之间进行切换。状态模式的结构图如图 12-1 所示。

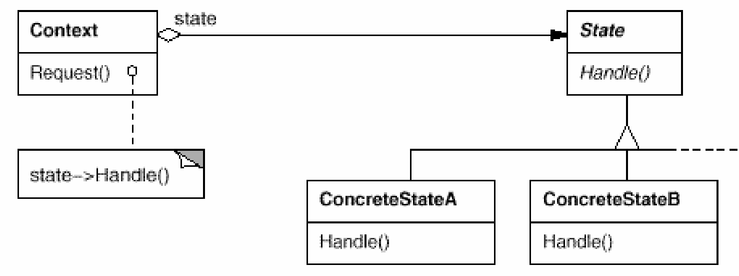

图 12-1　状态模式的结构图

从图 12-1 中可以看出，状态模式主要涉及以下几方面的元素。

（1）State 类：该类是一个抽象类，有一个或多个抽象方法。

（2）ConcreteState 类：该类为 State 类的派生类，有多个。实现了 State 类的各个抽象方法，分别定义应用程序各状态下的行为。

（3）Context 类：该类获得当前的操作状态，它建立了对 State 类实例的引用。

获得当前操作状态以后，就可以按照相应的指令进行工作。

对照图 12-1 我们不难发现，在前面创建的小系统 NET_CAD 中实际上已经应用了状态模

式。该程序中 ICommand 接口的作用就相当于 State 类的作用。通过实现 ICommand 接口，分别建立了 CCreateLine 类、CCreateCircle 类等交互操作类，它们相当于 State 类的派生类。得到基本的类以后，分别创建它们的实例，即

```
Public aCommand As ICommand
Public creLine As New CCreateLine()
Public creCircle As New CCreateCircle()
```

当然，创建实例的工作也可以通过在各类中创建一个返回实例的方法来完成。

获取当前绘制状态的任务由菜单选项或命令按钮的单击事件完成。例如下面的代码分别将当前绘制状态设置为绘制直线段和绘制圆。

```
Private Sub mnuLine_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles mnuLine.Click
    aCommand = creLine
End Sub

Private Sub mnuCircle_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles mnuCircle.Click
    aCommand = creCircle
End Sub
```

获取当前绘制状态以后，用鼠标进行绘图时就能直接调用 aCommand 对象的方法进行绘制。虽然从下面的代码中看不出来要进行什么样的具体工作，但由于多态的缘故，aCommand 对象自会将任务交给合适的派生类完成。

```
Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    Dim g As Graphics = Me.CreateGraphics()
    Dim aPos As PointF = PagetoWorld(New PointF(e.X, e.Y))
    aCommand.MouseMove(g, aPos)
End Sub
```

12.2　访问者模式

在小系统中，实现图元绘制、拾取和变换的 Draw, GetBox, Pick, Move, Rotate, Mirror 和 Scale 等方法是分散在各个图元类中的。实际上，在每个图元类中定义这些方法时都要重复一些代码。例如 Draw 方法，在定义每个图元类的 Draw 方法时，都要获得绘图环境的句柄，并进行创建画笔、选入画笔、删除画笔和释放句柄等操作，大段的代码都是重复的。为解决代码重复的问题，可使用访问者模式。

使用访问者模式，需要首先定义一个与绘图有关的 Visitor 类。Visitor 类是一个外部类，可以对其他数据进行访问。它对其他对象（图元对象）进行访问时是遵守外交礼仪的，它有一个 Visit 方法，负责给“别人”打招呼，说我要给你画个像。那么图元类也不能含糊，得专门设置一个部门搞接待，它就是 Accept 方法。Accept 方法说，欢迎你的访问，给俺来个全身像，于是绘图这个事就成了。

这种关系可以用代码来表示：

```
Public Sub Accept(Byval v As Visitor)
    v.Visit(Me)
End Sub
```

访问者模式的结构图如图 12-2 所示。

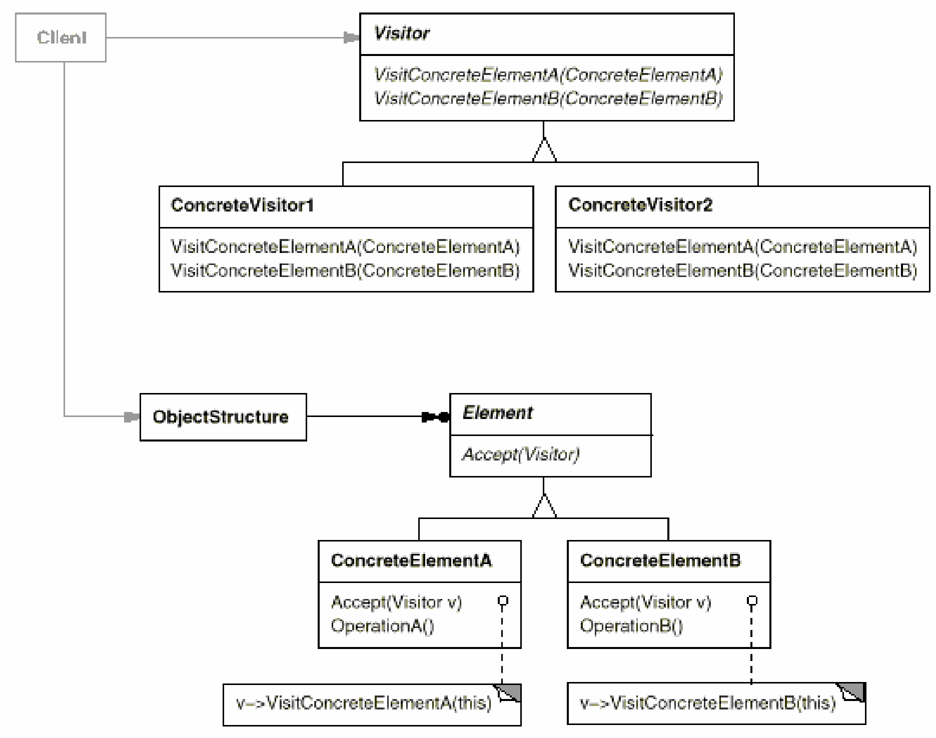

图 12-2　访问者模式的结构图

访问者模式主要涉及下面一些元素。

（1）Visitor：抽象类，定义抽象的 Visit 方法。

（2）ConcreteVisitor 类：为 Visitor 类的一个或多个派生类，实现具体的 Visit 行为。

（3）Element 类：抽象类，是被访问对象的基类，它有一个 Accept 方法。

（4）ConcreteElement 类：为 Element 类的派生类，实现了 Accept 方法。

下面我们用访问者模式解决前面提到的问题。以直线段和圆的绘制为例，首先建立一个抽象的 CGElement 类，然后它派生出一个 CLine 类和一个 CCircle 类。它们就相当于访问者模式中的 Element 类和 ConcreteElement 类。然后创建一个 Visitor 类和它的派生类 Drawer，在 Drawer 类中实现具体的绘图。下面进行详细的介绍。

1．创建 CGElement 类

CGElement 类为抽象类，其中有一个必须重写的抽象方法 Draw。Draw 方法有两个参数：g 参数为 Graphics 对象，指定绘图表面；v 参数为 Drawer 对象，进行绘图。Draw 方法就相当于模型中的 Accept 方法，它把 Visitor 对象作为绘图参数并通过它实现绘图。

【VB.NET】

```
Public MustInherit Class CGElement
    Public MustOverride Sub Draw(ByVal g As Graphics, ByVal v As Drawer)
End Class
```

【VC#.NET】

```
public abstract class CGElement
{
    public CGElement()
    {
    }

    public abstract void Draw( Graphics g,Drawer v);
}
```

2．创建 CLine 类

CLine 类继承 CGElement 类，重写了基类的 Draw 方法，绘制直线段。绘制工作是通过调用 Drawer 对象的 toDraw 方法实现的，它将绘图对象设置为 Me 或 this，相当于说，就画我吧！

【VB.NET】

```
Public Class CLine
    Inherits CGElement

    Public Overrides Sub Draw(ByVal g As Graphics, ByVal v As Drawer)
        v.toDraw(g, Me)
    End Sub
End Class
```

【VC#.NET】

```
public class CLine:CGElement
{
    public CLine()
    {
    }

    public override void Draw(Graphics g, Drawer v)
    {
        v.toDraw(g, this);
    }
}
```

3．创建 CCircle 类

创建 CCircle 类的过程与创建 CLine 类的相似，同样要从 CGElement 类继承，然后重写基类的 Draw 方法，并通过它绘图。

【VB.NET】

```
Public Class CCircle
    Inherits CGElement

    Public Overrides Sub Draw(ByVal g As Graphics, ByVal v As Drawer)
        v.toDraw(g, Me)
    End Sub
End Class
```

【VC#.NET】

```
public class CCircle:CGElement
{
     public CCircle()
     {
     }

     public override void Draw(Graphics g, Drawer v)
     {
          v.toDraw(g, this);
     }
}
```

4．创建 Visitor 类

现在创建 Visitor 类，它就是访问者的总头目。总头目一般都只给宏观规划，所以 Visitor 类只给出了两个抽象的 toDraw 重载方法，一个负责画直线段，另一个负责画圆。但具体绘画的工作总头目是不会做的，具体工作交给它的下属机构"绘图部"Drawer 类完成。如果还有拾取、变换等任务要做，还可以设立相应的"拾取部"Picker 类和"变换部"Transformer 类来完成，总之，总头目不会亲自动手。

【VB.NET】

```
Public MustInherit Class Visitor
    Public MustOverride Sub toDraw(ByVal g As Graphics, ByVal line As CLine)
    Public MustOverride Sub toDraw(ByVal g As Graphics, ByVal circle As CCircle)
End Class
```

【VC#.NET】

```
public abstract class Visitor
{
```

```
        public Visitor()
        {
        }

        public abstract void toDraw(Graphics g, CLine line);
        public abstract void toDraw(Graphics g, CCircle circle);
    }
```

5．创建 Drawer 类

前面说了，Drawer 类是一个职能部门，主要掌管绘图方面的工作，直属 Visitor 类。在 Drawer 类中要重写基类中的重载方法 toDraw。Drawer 类的实现代码罗列在后面。从中可以看出，Drawer 类比较好地解决了本节开始提到的绘图代码重复的问题。在 Drawer 类中，将绘图以前的设置绘图模式、设置画笔参数、创建画笔、选入画笔等工作放到了一个 preDraw 方法中，将选入原来画笔、删除新画笔和释放绘图环境句柄等任务放在一个 postDraw 方法中完成。用 toDraw 方法进行绘图时只需要根据当前要绘制的图元类型选用不同的绘图函数并调用前面两个方法就行了。

当然，这一点不用访问者模式也能做到，可将 preDraw 方法和 postDraw 方法放到公共模块中就行了，但面向对象的特征就差了一些。

【VB.NET】

```
    Public Class Drawer
        Inherits Visitor

        Private m_Style, m_Width, m_Color As Integer
        Private aPen, oldP As Long

        Public Sub preDraw(ByVal hdc As IntPtr)
            '设置画笔参数
            Win32API.SetROP2(hdc, 13)
            m_Style = 0
            m_Width = 3
            m_Color = RGB(255, 0, 0)
            '创建画笔
            aPen = Win32API.CreatePen(m_Style, m_Width, m_Color)
            '把画笔选入绘图环境，并返回原来的画笔
            oldP = Win32API.SelectObject(hdc, aPen)
        End Sub

        Public Sub postDraw(ByVal g As Graphics, ByVal hdc As IntPtr)
            '把原来的画笔选入绘图环境
            Win32API.SelectObject(hdc, oldP)
            '删除新创建的画笔
```

```
        Win32API.DeleteObject(aPen)
        '释放绘图环境句柄
        g.ReleaseHdc(hdc)
    End Sub

    Public Overloads Overrides Sub toDraw(ByVal g As Graphics, ByVal line As CLine)
        Dim hdc As IntPtr = g.GetHdc
        preDraw(hdc)
        '把画笔移动到直线段的起点处
        Win32API.MoveToEx(hdc, 50, 50, Nothing)
        '绘直线段到终点
        Win32API.LineTo(hdc, 200, 200)
        postDraw(g, hdc)
    End Sub

    Public Overloads Overrides Sub toDraw(ByVal g As Graphics, ByVal circle As CCircle)
        Dim hdc As IntPtr = g.GetHdc
        preDraw(hdc)
        '把空刷子选入绘图环境
        Win32API.SelectObject(hdc, Win32API.GetStockObject(5))
        '绘圆
        Win32API.Ellipse(hdc, 100, 80, 300, 400)
        postDraw(g, hdc)
    End Sub

End Class
```

【VC#.NET】

```
public class Drawer
{
     public Drawer()
     {
     }

     private int m_Style,m_Width,m_Color;
     private long aPen,oldP;

     public void preDraw(IntPtr hdc)
     {
          //设置画笔参数
          Win32API.SetROP2(hdc, 13);
          m_Style = 0;
```

```
        m_Width = 3;
        m_Color = 255;
        //创建画笔
        aPen = Win32API.CreatePen(m_Style, m_Width, m_Color);
        //把画笔选入绘图环境，并返回原来的画笔
        oldP = Win32API.SelectObject(hdc, aPen);
    }

    public void postDraw(Graphics g,IntPtr hdc)
    {
        //把原来的画笔选入绘图环境
        Win32API.SelectObject(hdc, oldP);
        //删除新创建的画笔
        Win32API.DeleteObject(aPen);
        //释放绘图环境句柄
        g.ReleaseHdc(hdc);
    }

    public    void toDraw(Graphics g,CLine line)
    {
        IntPtr hdc= g.GetHdc();
        preDraw(hdc);
        Win32API.LPPOINT    p=new Win32API.LPPOINT();
        //把画笔移动到直线段的起点处
        Win32API.MoveToEx(hdc, 50, 50,p);
        //绘直线段到终点
        Win32API.LineTo(hdc, 200, 200);
        postDraw(g, hdc);
    }

    public    void toDraw(Graphics g,CCircle circle)
    {
        IntPtr hdc= g.GetHdc();
        preDraw(hdc);
        //把空刷子选入绘图环境
        Win32API.SelectObject(hdc, Win32API.GetStockObject(5));
        //绘圆
        Win32API.Ellipse(hdc, 100, 80, 300, 400);
        postDraw(g, hdc);
    }
}
```

6．类的测试

至此，用于绘制直线段和圆的访问者模式的框架就搭起来了。现在测试一下，看它能不能正常工作。在 Form1 **类中重写基类的** OnPaint **方法，分别创建一个** CLine **对象** line, CCircle **对象** circle **和** Drawer **对象** v**，然后调用** line **和** circle **的** Draw **方法绘制直线段和圆。**

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim line As New CLine()
    Dim circle As New CCircle()
    Dim v As New Drawer()
    line.Draw(g, v)
    circle.Draw(g, v)
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    CLine line=new CLine();
    CCircle circle=new CCircle();
    Drawer v = new Drawer();
    line.Draw(g, v);
    circle.Draw(g, v);
}
```

程序的运行效果如图 12-3 **所示。**

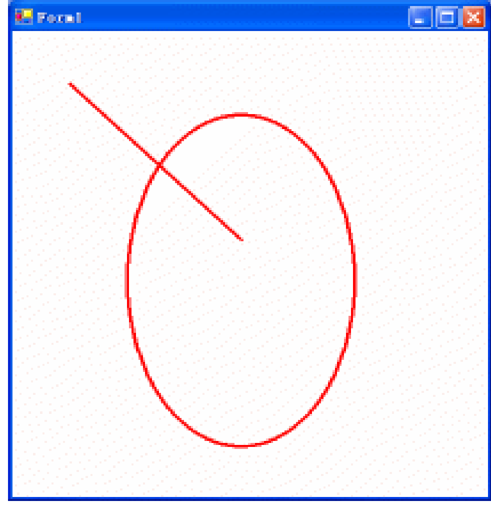

图 12-3　**绘图效果**

12.3　模板方法模式

模板方法模式用于实现一个算法，这个算法的结构是确定的，即先算什么后算什么是确定的，但在某个步骤上具体怎么算可能不一样。使用模板方法模式，就可以在不改变算法结构的情况下重新定义这些步骤。模板方法模式的结构图如图 12-4 所示。

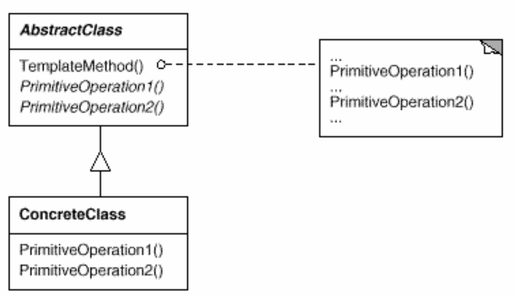

图 12-4　模板方法的结构图

从图中可以看出，模板方法模式主要要实现下面一些元素。

（1）AbstractClass 类：该类为抽象类，其中有一个或多个抽象方法，其功能在各派生类中实现。有一个已经实现的模板方法 TemplateMethod，这个模板方法调用一个或多个抽象方法，确定实现算法的结构。

（2）ConcreteClass 类：该类为 AbstractClass 类的派生类。它(们)实现了基类的抽象方法，并且继承的模板方法调用这些方法，实现对应条件下的算法。

下面的例子用模板方法模式实现一个图元选择器。当我们考虑模板方法模式时，要确定判断图元选中的条件和反馈方式。这里，我们确定在图元上方单击鼠标左键时，如果图元被拾取，则该图元被选中，并用红色虚线表示。

1．创建 GESelecter 类

首先创建一个 GESelecter 类，它是一个抽象类，有两个抽象方法，即 Pick 方法和 Draw 方法。Selected 方法调用这两个抽象方法，如果图元被拾取，就在屏幕上删除它然后用红色虚线重画。DrawSettings 方法返回一个画笔对象，确定正常绘制模式、选中模式和删除模式下使用的画笔的属性。

【VB.NET】

```
Imports System.Collections
Imports System.Drawing.Drawing2D


Public MustInherit Class GESelecter


    Public Enum geDrawMode
        Normal = 1
        Selected = 2
```

```
        Delete = 3
    End Enum

    '绘制图元
    Public MustOverride Sub Draw(ByVal g As Graphics, ByVal drawMode As geDrawMode)

    '拾取图元
    Public MustOverride Function Pick(ByVal aPos As PointF) As Boolean

    '选择图元
    Public Sub Selected(ByVal g As Graphics, ByVal aPos As PointF)
        If Me.Pick(aPos) Then
            Me.Draw(g, geDrawMode.Delete)
            Me.Draw(g, geDrawMode.Selected)
        End If
    End Sub

    Public Function DrawSettings(ByVal aDrawMode As geDrawMode) As Pen
        Dim pen As New Pen(Color.Black)
        Select Case aDrawMode
            Case GESelecter.geDrawMode.Normal
                pen.Color = Color.Black
            Case GESelecter.geDrawMode.Selected
                pen.Color = Color.Red
                pen.DashStyle = DashStyle.Dash
            Case GESelecter.geDrawMode.Delete
                pen.Color = Color.White
        End Select
        Return pen
    End Function
End Class
```

【VC#.NET】

```
public abstract class GESelecter
{
     public GESelecter()
     {
     }

     public enum geDrawMode
     {
```

```
        Normal,Selected,Delete
    }

    //绘制图元
    public abstract void Draw(Graphics g, geDrawMode drawMode);

    //拾取图元
    public abstract bool Pick(PointF aPos);

    //选择图元
    public void Selected(Graphics g, PointF aPos)
    {
        if (this.Pick(aPos))
        {
            this.Draw(g, geDrawMode.Delete);
            this.Draw(g, geDrawMode.Selected);
        }
    }

    //设置画笔
    public Pen DrawSettings(geDrawMode aDrawMode)
    {
        Pen pen=new Pen(Color.Black);
        switch (aDrawMode)
        {
            case GESelecter.geDrawMode.Normal:
                pen.Color = Color.Black;
                break;
            case GESelecter.geDrawMode.Selected:
                pen.Color = Color.Red;
                pen.DashStyle = DashStyle.Dash;
                break;
            case GESelecter.geDrawMode.Delete:
                pen.Color = Color.White;
                break;
        }
        return pen;
    }
```

2．创建 CLine 类

CLine 类继承了 GESelecter 类，并重写了基类的两个抽象方法，分别实现直线段的拾取

和绘制。与小系统中不同的是，这里都是采用 GDI+函数实现的。实现这两个方法以后，从基类继承过来的 Selected 方法就会调用它们，确定直线段的选择方案。

【VB.NET】

```
Imports System.Drawing.Drawing2D

Public Class CLine
    Inherits GESelecter

    Private m_Begin As PointF
    Private m_End As PointF

    '直线段的起点属性
    Public Property LBegin() As PointF
        Get
            Return m_Begin
        End Get
        Set(ByVal Value As PointF)
            m_Begin = Value
        End Set
    End Property

    '直线段的终点属性
    Public Property LEnd() As PointF
        Get
            Return m_End
        End Get
        Set(ByVal Value As PointF)
            m_End = Value
        End Set
    End Property

    '无参构造函数
    Public Sub New()
    End Sub

    构造函数，用已知的两点构造直线段
    Public Sub New(ByVal pBegin As PointF, ByVal pEnd As PointF)
        m_Begin = pBegin
        m_End = pEnd
    End Sub
```

```
    '绘直线段
    Public Overrides Sub Draw(ByVal g As Graphics, _
                                     ByVal aDrawMode As geDrawMode)
        Dim pen As Pen = DrawSettings(aDrawMode)
        g.DrawLine(pen, m_Begin.X, m_Begin.Y, m_End.X, m_End.Y)
    End Sub

    '拾取直线段
    Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
        Dim gp As New GraphicsPath()
        gp.AddLine(m_Begin, m_End)
        If gp.GetBounds.Contains(aPos) Then
            If gp.IsOutlineVisible(aPos, Pens.Black) Then
                gp.Dispose()
                Return True
            Else
                gp.Dispose()
                Return False
            End If
        Else
            gp.Dispose()
            Return False
        End If
    End Function
End Class
```

【VC#.NET】

```
public class CLine:GESelecter
{
    private PointF m_Begin;
    private PointF m_End;

    public PointF LBegin
    {
        get{return m_Begin;}
        set{m_Begin=value;}
    }

    public PointF LEnd
    {
```

```
        get{return m_End;}
        set{m_End=value;}
    }

    public CLine()
    {
    }

    public CLine(PointF pBegin,PointF pEnd)
    {
        m_Begin=pBegin;
        m_End=pEnd;
    }

    //绘直线段
    public override void Draw(Graphics g, geDrawMode aDrawMode)
    {
        Pen pen = DrawSettings(aDrawMode);
        g.DrawLine(pen, m_Begin.X, m_Begin.Y, m_End.X, m_End.Y);
    }

    //拾取直线段
    public override bool Pick( PointF aPos)
    {
        Console.WriteLine("yes");
        GraphicsPath gp=new GraphicsPath();
        gp.AddLine(m_Begin, m_End);
        RectangleF rect=gp.GetBounds();
        if (rect.Contains(aPos))
        {
            if (gp.IsOutlineVisible(aPos, Pens.Black))
            {
                gp.Dispose();
                return true;
            }
            else
            {
                gp.Dispose();
                return false;
            }
        }
```

```
                else
                {
                    gp.Dispose();
                    return false;
                }
            }
        }
```

3．创建 CCircle 类

可以像创建 CLine 类那样创建 CCircle 类。从基类继承的 Selected 方法调用重写后的 Pick 方法和 Draw 方法，确定圆的选择方案。

【VB.NET】

```
Imports System.Drawing.Drawing2D
Imports System.Math

Public Class CCircle
    Inherits GESelecter

    Private m_Center, m_pCircle As PointF

    '圆心
    Public Property Center() As PointF
        Get
            Return m_Center
        End Get
        Set(ByVal Value As PointF)
            m_Center = Value
        End Set
    End Property

    '圆上一点
    Public Property pCircle() As PointF
        Get
            Return m_pCircle
        End Get
        Set(ByVal Value As PointF)
            m_pCircle = Value
        End Set
    End Property
```

```
'半径属性，只读
Public ReadOnly Property Radius() As Single
    Get
        Dim r As Single
        r = DistPtoP(m_Center, m_pCircle)
        Return r
    End Get
End Property

'计算点与点之间的距离
Private Function DistPtoP(ByVal p1 As PointF, _
            ByVal p2 As PointF) As Single
    Dim dx, dy As Single
    dx = p2.X - p1.X
    dy = p2.Y - p1.Y
    Return Sqrt((dx * dx) + (dy * dy))
End Function

'无参构造函数
Public Sub New()
End Sub

'构造函数，用已知的两点构造圆
Public Sub New(ByVal pCenter As PointF, ByVal pCir As PointF)
    m_Center = pCenter
    m_pCircle = pCir
End Sub

'绘圆
Public Overrides Sub Draw(ByVal g As Graphics, _
                    ByVal aDrawMode As geDrawMode)
    Dim pen As Pen = DrawSettings(aDrawMode)
    g.DrawEllipse(pen, m_Center.X - Radius, _
          m_Center.X - Radius, Radius * 2, Radius * 2)
End Sub

'拾取圆
Public Overrides Function Pick(ByVal aPos As PointF) As Boolean
    Dim gp As New GraphicsPath()
    gp.AddEllipse(m_Center.X - Radius, m_Center.X - Radius, _
            Radius * 2, Radius * 2)
```

```
                If gp.GetBounds.Contains(aPos) Then
                    If gp.IsOutlineVisible(aPos, Pens.Black) Then
                        gp.Dispose()
                        Return True
                    Else
                        gp.Dispose()
                        Return False
                    End If
                Else
                    gp.Dispose()
                    Return False
                End If
            End Function
        End Class
```

【VC#.NET】

```
        public class CCircle:GESelecter
        {
            private PointF m_Center, m_pCircle;

            public PointF Center
            {
                get{return m_Center;}
                set{m_Center=value;}
            }

            public PointF PCircle
            {
                get{return m_pCircle;}
                set{m_pCircle=value;}
            }

            public float Radius
            {
                get
                {
                    float r=DistPtoP(m_Center,m_pCircle);
                    return r;
                }
            }
```

```
public CCircle()
{
}

public CCircle(PointF pCenter,PointF pCircle)
{
    m_Center=pCenter;
    m_pCircle=pCircle;
}

//计算点与点之间的距离
private float DistPtoP(PointF p1,PointF p2)
{
    float dx,dy;
    dx = p2.X - p1.X;
    dy = p2.Y - p1.Y;
    return (float)(Math.Sqrt((dx * dx) + (dy * dy)));
}

//绘圆
public override void Draw(Graphics g, geDrawMode aDrawMode)
{
    Pen pen= DrawSettings(aDrawMode);
    g.DrawEllipse(pen, m_Center.X - Radius,
        m_Center.X - Radius, Radius * 2, Radius * 2);
}

//拾取圆
public override bool Pick(PointF aPos)
{
    GraphicsPath gp=new GraphicsPath();
    gp.AddEllipse(m_Center.X - Radius, m_Center.X - Radius,
                Radius * 2, Radius * 2);
    if (gp.GetBounds().Contains(aPos))
    {
        if (gp.IsOutlineVisible(aPos, Pens.Black))
        {
            gp.Dispose();
            return true;
        }
        else
```

```
                    {
                        gp.Dispose();
                        return false;
                    }
                }
                else
                {
                    gp.Dispose();
                    return false;
                }
            }
        }
```

4．类的测试

前面创建了抽象类 GESelecter **和它的两个派生类** CLine **类和** CCircle **类，并试图用一个模板方法** Selected **实现图元的选择判断和反馈。下面在** Form1 **类中添加下面的代码，对我们的模板方法模式进行测试。首先在窗体上绘制两条直线段和两个圆，并把它们添加到一个集合类中，然后重写基类的** OnMouseDown **方法，在窗体上单击鼠标左键时对集合类中的元素进行遍历，并调用模板方法** Selected **进行图元选择。**

【VB.NET】

```
Private ges As New ArrayList()
Private drawm As GESelecter.geDrawMode

Private Sub Draw(ByVal g As Graphics)
    Dim line1 As New CLine(New PointF(10, 20), New PointF(200, 300))
    Dim line2 As New CLine(New PointF(50, 100), New PointF(400, 200))
    Dim circle1 As New CCircle(New PointF(100, 100), New PointF(0, 100))
    Dim circle2 As New CCircle(New PointF(200, 200), New PointF(200, 300))
    line1.Draw(g, drawm)
    line2.Draw(g, drawm)
    circle1.Draw(g, drawm)
    circle2.Draw(g, drawm)
    ges.Add(line1)
    ges.Add(line2)
    ges.Add(circle1)
    ges.Add(circle2)
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
```

```
        g.FillRectangle(Brushes.White, Me.ClientRectangle)
        drawm = GESelecter.geDrawMode.Normal
        Draw(g)
    End Sub

    Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)
        Dim g As Graphics = Me.CreateGraphics
        Dim aPos As New PointF(e.X, e.Y)
        If e.Button = MouseButtons.Left Then
            Dim i As Integer
            For i = 0 To ges.Count - 1
                ges(i).Selected(g, aPos)
            Next
        End If
    End Sub
```

【VC#.NET】

```
    private ArrayList ges=new ArrayList();
    private GESelecter.geDrawMode drawm;

    private void Draw(Graphics g)
    {
        CLine line1=new CLine(new PointF(10, 20), new PointF(200, 300));
        CLine line2=new CLine(new PointF(50, 100), new PointF(400, 200));
        CCircle circle1=new CCircle(new PointF(100, 100), new PointF(0, 100));
        CCircle circle2=new CCircle(new PointF(200, 200), new PointF(200, 300));
        line1.Draw(g, drawm);
        line2.Draw(g, drawm);
        circle1.Draw(g, drawm);
        circle2.Draw(g, drawm);
        ges.Add(line1);
        ges.Add(line2);
        ges.Add(circle1);
        ges.Add(circle2);
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics g= e.Graphics;
        g.FillRectangle(Brushes.White, this.ClientRectangle);
        drawm = GESelecter.geDrawMode.Normal;
```

```
        Draw(g);
    }

    protected override void OnMouseDown(MouseEventArgs e)
    {
        Graphics g = this.CreateGraphics();
        PointF aPos=new PointF(e.X, e.Y);
        if (e.Button == MouseButtons.Left)
        {
            for (int i = 0;i<=ges.Count － 1;i++)
            {
                ((GESelecter)(ges[i])).Selected(g, aPos);
            }
        }
    }
```

程序运行结果如图 12-5 所示。用鼠标左键单击图元时，图元被选中，并用红色虚线表示。

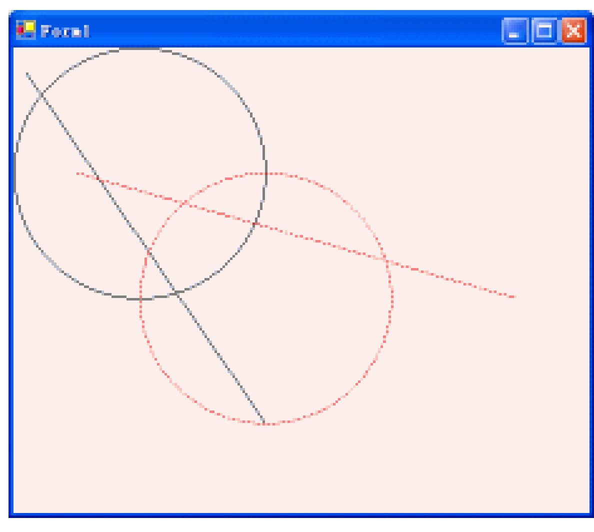

图 12-5　选中图元

12.4　策略模式

在解决同一个问题时，常常有多种方法可供选择。比如 CAD 绘图中就有很多这样的情况，可以采用多种方法绘制圆或圆弧，可以有多种方法在图元被选中以后进行反馈显示等。策略模式提供了一种管理不同算法的途径。它将这些方法一个个地封装起来，并且使它们可以互相替换。策略模式的结构图如图 12-6 所示。

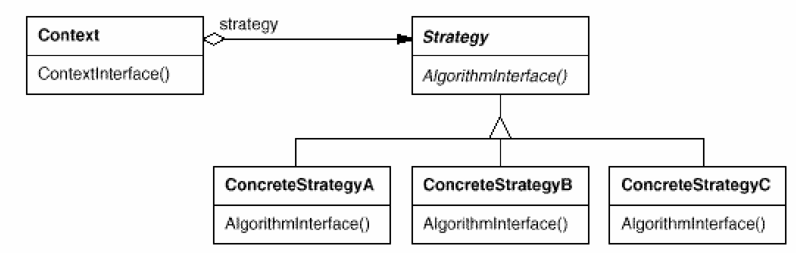

图 12-6　策略模式的结构图

从图 12-6 中可以看出，策略模式主要包括以下几方面的元素。

（1）Strategy 类：该类为抽象类，定义一个抽象的算法方法。

（2）ConcreteStrategy 类：该类为 Strategy 类的多个派生类，分别实现不同的算法。

（3）Context 类：该类建立策略类和客户之间的关系。它有一个 Strategy 类实例的引用，并通过它得到指定的算法。

圆的绘制有多种方法，可以用圆心和圆上一点绘圆、圆心和半径绘圆，也可以用圆上三点绘圆。下面的例子讨论采用第 1 种方式和第 3 种方式绘圆时的策略模式。

1．创建 DrawCircle 类

DrawCircle 类是一个抽象类，相当于模式中的 Strategy 类。它提供了一个抽象的 Draw 方法。该方法负责绘圆，但是在派生类中具体实现。

【VB.NET】

```
Public MustInherit Class DrawCircle
    Public MustOverride Sub Draw(ByVal g As Graphics)
End Class
```

【VC#.NET】

```
public abstract class DrawCircle
{
    public DrawCircle()
    {
    }

    public abstract void Draw(Graphics g);
}
```

2．创建 CPCircle 类

跟后面的 ThreePCircle 类一样，CPCircle 类是 DrawCircle 类的派生类。CPCircle 类实现用圆心和圆上一点绘圆，是本例要讨论的绘圆策略之一。因为代码与 12.3 节中 CCircle 类的代码大同小异，这里不再重复，可参见光盘。

3．创建 ThreePCircle 类

ThreePCircle 类实现用圆上 3 个点绘圆。用圆上的 3 个点可以找到圆的圆心，它是连接 3 个点中不同两个点的连线的中法线的交点。圆心找到了，又知道圆上的点，于是就可以绘出圆来。

【VB.NET】

```
Imports System.Math
Imports System.Windows.Forms

Public Class ThreePCircle
    Inherits DrawCircle

    Private m_Point1, m_Point2 m_Point3, As PointF

    Public Property Point1() As PointF
        Get
            Return m_Point1
        End Get
        Set(ByVal Value As PointF)
            m_Point1 = Value
        End Set
    End Property

    Public Property Point2() As PointF
        Get
            Return m_Point2
        End Get
        Set(ByVal Value As PointF)
            m_Point2 = Value
        End Set
    End Property

    Public Property Point3() As PointF
        Get
            Return m_Point3
        End Get
        Set(ByVal Value As PointF)
            m_Point3 = Value
        End Set
    End Property
```

```
Public ReadOnly Property Center() As PointF
    Get
        Return CircleCenter()
    End Get
End Property

Public ReadOnly Property Radius() As Single
    Get
        Return Sqrt((m_Point1.X - Center.X) * (m_Point1.X - Center.X) + _
                (m_Point1.X - Center.X) * (m_Point1.X - Center.X))
    End Get
End Property

'无参构造函数
Public Sub New()
End Sub

'构造函数，用已知的两点构造圆
Public Sub New(ByVal p1 As PointF, ByVal p2 As PointF, ByVal p3 As PointF)
    m_Point1 = p1
    m_Point2 = p2
    m_Point3 = p3
End Sub

Private Function CircleCenter() As PointF
    Dim x0 As Single, y0 As Single
    Dim x1 As Single = m_Point1.X
    Dim y1 As Single = m_Point1.Y
    Dim x2 As Single = m_Point2.X
    Dim y2 As Single = m_Point2.Y
    Dim x3 As Single = m_Point3.X
    Dim y3 As Single = m_Point3.Y
    Dim A1 As Single = x1 - x3
    Dim B1 As Single = y1 - y3
    Dim A2 As Single = x2 - x3
    Dim B2 As Single = y2 - y3
    Dim C1 As Single = (x1 * x1 - x3 * x3 + y1 * y1 - y3 * y3) / 2
    Dim C2 As Single = (x2 * x2 - x3 * x3 + y2 * y2 - y3 * y3) / 2
    Dim D As Single = A1 * B2 - A2 * B1
    If D = 0 Then
        MessageBox.Show("找不到圆心。")
```

```
        Else
            x0 = (C1 * B2 - C2 * B1) / D
            y0 = (A1 * C2 - A2 * C1) / D
        End If
        Return (New PointF(x0, y0))
    End Function

    '绘圆
    Public Overrides Sub Draw(ByVal g As Graphics)
        g.DrawEllipse(Pens.Red, Center.X - Radius, _
                Center.X - Radius, Radius * 2, Radius * 2)
    End Sub

End Class
```

【VC#.NET】

```
public class ThreePCircle:DrawCircle
{
    private PointF m_Point1, m_Point2, m_Point3;

    public PointF Point1
    {
        get{return m_Point1;}
        set{m_Point1 = value;}
    }

    public PointF Point2
    {
        get{return m_Point2;}
        set{m_Point2 = value;}
    }

    public PointF Point3
    {
        get{return m_Point3;}
        set{m_Point3 = value;}
    }

    public PointF Center
    {
        get{return CircleCenter();}
    }
```

```
public float Radius
{
    get{
        return (float)(Math.Sqrt((m_Point1.X - Center.X) * _
            (m_Point1.X - Center.X) +(m_Point1.X - Center.X) * _
            (m_Point1.X - Center.X)));
    }
}

//无参构造函数
public ThreePCircle()
{
}

//构造函数，用已知的三点构造圆
public ThreePCircle(PointF p1, PointF p2, PointF p3)
{
    m_Point1 = p1;
    m_Point2 = p2;
    m_Point3 = p3;
}

private PointF CircleCenter()
{
    float x0=10000;
    float y0=10000;
    float x1= m_Point1.X;
    float y1= m_Point1.Y;
    float x2= m_Point2.X;
    float y2= m_Point2.Y;
    float x3= m_Point3.X;
    float y3 = m_Point3.Y;
    float A1= x1 - x3;
    float B1 = y1 - y3;
    float A2 = x2 - x3;
    float B2 = y2 - y3;
    float C1 = (x1 * x1 - x3 * x3 + y1 * y1 - y3 * y3) / 2;
    float C2 = (x2 * x2 - x3 * x3 + y2 * y2 - y3 * y3) / 2;
    float D = A1 * B2 - A2 * B1;
    if (D == 0)
```

```
            {
                MessageBox.Show("找不到圆心。");
            }
            else
            {
                x0 = (C1 * B2 - C2 * B1) / D;
                y0 = (A1 * C2 - A2 * C1) / D;
            }
            return (new PointF(x0, y0));
        }

        //绘圆
        public override void Draw( Graphics   g)
        {
            g.DrawEllipse(Pens.Red, Center.X - Radius,
                Center.X - Radius, Radius * 2, Radius * 2);
        }
    }
```

4．创建 Context 类

Context 是上下文的意思。以此命名，说明这个类起联系的作用。它联系绘圆的各个类和客户端。其中客户端可以是菜单选项或命令按钮。Context 类有一个 DrawCircle 对象 drawer，通过它，在 SetCPCircle 方法和 SetThreePCircle 方法中分别指定用圆心、圆上一点方式和圆上三点方式绘圆。然后有一个 Draw 方法，它调用 drawer 的 Draw 方法在当前绘图表面上绘圆，这里用到了多态。

【VB.NET】

```
Public Class Context
    Private drawer As DrawCircle

    Public Sub SetCPCircle(ByVal circle As CPCircle)
        drawer = circle
    End Sub

    Public Sub SetThreePCircle(ByVal circle As ThreePCircle)
        drawer = circle
    End Sub

    Public Sub Draw(ByVal g As Graphics)
        drawer.Draw(g)
    End Sub
End Class
```

【VC#.NET】

```
public class Context
{
    private DrawCircle drawer;

    public void SetCPCircle(CPCircle circle)
    {
        drawer = circle;
    }

    public void SetThreePCircle(ThreePCircle circle)
    {
        drawer = circle;
    }

    public void Draw(Graphics g)
    {
        drawer.Draw(g);
    }
}
```

5．类的测试

现在，这个策略模式的实现框架就建起来了。在 Form1 类中重写基类的 OnPaint 方法，把它测试一下。首先创建一个 Context 类的实例 CircleDrawer、一个 CPCircle 类实例 aCP、一个 ThreePCircle 类实例 aThree。然后调用 Context 类的 SetCPCircle 方法、SetThreePCircle 方法和 Draw 方法，分别设置算法策略以后进行绘图。

【VB.NET】

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.FillRectangle(Brushes.White, Me.ClientRectangle)
    Dim CircleDrawer As New Context()
    Dim aCP As New CPCircle(New PointF(100, 100), New PointF(50, 50))
    Dim aThree As New ThreePCircle(New PointF(100, 50), _
                    New PointF(300, 100), New PointF(80, 150))
    CircleDrawer.SetCPCircle(aCP)
    CircleDrawer.Draw(g)
    CircleDrawer.SetThreePCircle(aThree)
    CircleDrawer.Draw(g)
End Sub
```

【VC#.NET】

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g= e.Graphics;
    g.FillRectangle(Brushes.White, this.ClientRectangle);
    Context CircleDrawer=new Context();
    CPCircle aCP=new CPCircle(new PointF(100, 100), new PointF(50, 50));
    ThreePCircle aThree=new ThreePCircle(new PointF(100, 50),
                    new PointF(300, 100), new PointF(80, 150));
    CircleDrawer.SetCPCircle(aCP);
    CircleDrawer.Draw(g);
    CircleDrawer.SetThreePCircle(aThree);
    CircleDrawer.Draw(g);
}
```

运行程序，结果如图 12-7 所示。

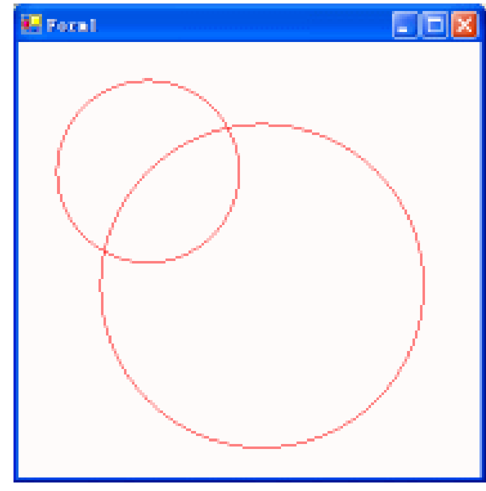

图 12-7　用两种算法绘圆

12.5　其他设计模式

除了上面一些设计模式外，还有很多比较成熟的模式，而且在 CAD 设计中也可以应用，比如工厂方法模式、命令模式、观察者模式、记事模式和组合模式等。当然，它们好，或者说我在这里介绍它们，读者不一定就要用它们。再好的东西也忌讳生搬硬套。比如我们前面介绍的 CAD 小系统，它的结构就比较清晰，代码可读性很好。但也有缺点，比如同一个功能函数分散在各个派生类中，代码分散，且多少都有一些重复代码。可以有办法把功能代码集中管理，代码也不重复，但可读性就要受到损失。所以，任何事物都有成功的一面，也有失败的一面，关键是我们站在什么角度去看它。

下面简单介绍几种设计模式。

12.5.1 工厂方法模式

工厂方法模式属于创建型设计模式，它的作用主要是将实例的创建和分发进行集中管理。具体应用中，首先定义一个创建类实例的抽象基类，然后用派生类来确定到底创建哪个类的实例。基类中要有一个抽象的工厂方法 FactoryMethod，一个调用该方法的可以继承的方法 ConstructObjects。在派生类中重写基类的 FactoryMethod 方法，它返回指定产品类的实例。然后在客户端调用派生类的 ConstructObjects 方法。实际上，我们可以发现，这里 ConstructObjects 方法是一个模板方法。

结合 CAD 小系统来说，我们可以这样做：创建一个工厂抽象基类，它的派生类是各个命令按钮；一个产品抽象基类，它的派生类是各个交互绘图类。用鼠标单击指定的命令按钮时，将创建一个实现该指令的交互类的实例。

12.5.2 命令模式

在小系统中，首先创建一个命令基类或接口，它有一些抽象的方法，然后定义一系列派生类或实现类，通过重写基类或接口的抽象方法来实现具体的任务。这个模式就是命令模式。在介绍状态模式时有过类似的讨论，但按作者的理解，当有多个派生类继承或实现命令基类或接口时，每个派生类对应一个操作状态，状态模式的作用是实现各个状态之间的任务切换，而命令模式则强调命令请求的封装。后面介绍记事模式时还会讲到，使用命令模式，还可以实现操作的取消。

12.5.3 观察者模式

对象之间往往存在依赖关系，当对象主体发生变化时，依赖它的对象将相应地发生变化。观察者模式用于处理对象之间的这种依赖关系。使用观察者模式，需要首先弄清依赖关系中谁是主体，谁是观察者。然后，描述主体的类要有一个 Notify 方法，它通知观察者有事情发生了；描述观察者的类要有一个 Update 方法，用于接收主体的通知并进行更新。

观察者模式的应用很普遍，因为对象间的依赖关系很普遍。比如 CAD 中，绘制三维图形时经常要在另外的视图窗口中显示它的三视图（前视图、左视图和上视图），当主窗口中三维图形发生变化时，显示三个方向视图的窗口中图形也要发生变化。这里主窗口就是主体，显示三视图的窗口就是观察者。

12.5.4 记事模式

绘图操作的取消是绘图系统需要具备的功能，它让用户在犯下错误的时候有“后悔药”可吃。如果说编程者是“后悔药”厂的厂长，那么记事模式为厂长提供了一个安全有效的处方。处方大抵有下面一些内容。

（1）创建一个 Memento 类，它负责保存和获取操作状态。

（2）创建一个 CareTaker 类，它创建 Memento 类实例并用一个集合类保存要取消的项目，它有一个 Undo 方法。

（3）前面介绍命令模式时曾经讲到，使用命令模式的一个好处是可以实现操作的撤消。在命令派生类中创建 CareTaker 类的实例，然后调用它的 Undo 方法实现操作的撤消。

参 考 文 献

1 苏金明编著. 用 Visual Basic 开发交互式 CAD 系统. 电子工业出版社, 2003.5
2 李于剑编著. Visual C++实践与提高—图形图像编程篇. 中国铁道出版社, 2001.2
3 陈建春编著. Visual C++开发 GIS 系统—开发实例剖析. 电子工业出版社, 2001.3
4 Fred Barwell, Richard Blair 等著, 康博译. VB.NET 高级编程. 清华大学出版社, 2002.3
5 Eric White 著, 杨浩, 张哲峰译. GDI+程序设计. 清华大学出版社, 2002.12
6 [美]James W. Cooper 著, 赵会群等译. Visual Basic 设计模式. 清华大学出版社, 2003.4