深入剖析 Visual C++ 编程技术及应用实例

◆ 求是科技　王洪涛　编著

人民邮电出版社
POSTS & TELECOMMUNICATIONS PRESS

深入剖析 Visual C++ 编程技术及应用实例

◆ 求是科技　王洪涛　编著

人民邮电出版社

图书在版编目（CIP）数据

深入剖析 Visual C++编程技术及应用实例/求是科技，王洪涛编著.
—北京：人民邮电出版社，2003.3
ISBN 7-115-11011-5

I. 深... II. ①求... ②王... III. C 语言—程序设计 IV. TP312
中国版本图书馆 CIP 数据核字（2003）第 011393 号

内容提要

本书通过主流开发领域内的若干具有代表性的实例，向读者深入细致地讲解了利用 Visual C++ 6.0 进行 Windows 高级软件开发的相关技术。

本书涉及的领域包括用户界面、文件系统、图形图像技术、多媒体技术、网络、COM、ActiveX 技术、系统底层开发技术等。

本书可作为有志于用 Visual C++进行 Windows 高级软件开发的人员阅读，也可作为相关人员的参考用书。

深入剖析 Visual C++编程技术及应用实例

◆ 编　　著　求是科技　王洪涛
责任编辑　张立科

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www. ptpress.com.cn
读者热线　010-67132692
北京汉魂图文设计有限公司制作
北京鸿佳印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：33.5
字数：1 045 千字　　2003 年 3 月第 1 版
印数：1-5 000 册　　2003 年 3 月北京第 1 次印刷

ISBN7-115-11011-5/TP・3311

定价：52.00 元（附光盘）

前　言

美国微软公司出品的 Visual C++ 6.0 是著名的集成开发工具，其凭借强大的功能、丰富的类库占据了 Windows 平台下 C++开发工具的绝大部分市场份额。据不完全统计世界上 70%的 C++商用软件是采用 Visual C++开发的，可见其强大的功能已经得到了全世界软件工程师的认可。

本书就是一本讲解利用 Visual C++进行高级软件开发的图书，全书用“由浅入深”、“技术原理与典型实例代码相结合”的方式，向每一位有志于从事 Windows 高级软件开发的读者介绍热门领域的高级开发技术。

本书选取了主流开发领域内的若干具有代表性的实例，包含了在各领域中广泛应用的技术。例如：书中介绍了数字图像处理中常用的算法，这些算法都是计算机前沿科学中经常用到的；网络部分的 HTTP 服务器实例不仅为读者介绍了 HTTP 协议的具体细节，而且还为读者解答了服务器是怎么运行 CGI 程序的，相信很多读者都会对这个问题感兴趣；多媒体技术部分为读者介绍了视频捕捉技术，这是时下非常流行的热门技术；文件捆绑是很多木马程序经常采用的技术，本书的文件系统部分将为读者介绍这一技术。

全书共分为 8 章，分别涉及到 7 个领域，包括用户界面的开发与应用、文件系统的相关开发实例、图形图像技术、网络技术与 IE 开发、多媒体技术、COM 技术的应用、操作系统的底层开发技术，最后为读者介绍了作者在软件开发过程中的一些心得体会。

第 1 章　控件编程。不仅向读者介绍了便于开发通用用户界面的基本控件的使用方法，，而且还介绍了如何通过重载控件的基类来创建特色控件的方法，开发新颖的软件界面。

第 2 章　文件系统。主要介绍了基于文件系统的多种实用技术，包括文件的分割合并、文件监控、文件捆绑等在实际开发中有广泛应用的技术。

第 3 章　图形图像。不仅介绍了计算机图形学、数字图像处理学中的相关算法等理论方面的知识，而且还介绍了在实际工作中进行图形图像开发的具体步骤，这部分的内容对有志于从事 Windows 图形图像开发工作的人员有很高的参考价值。

第 4 章　网络。介绍了如何利用 Windows Socket 进行各种网络协议的开发。IE 浏览器是目前 Windows 系统下最为广泛使用的浏览器软件，由于采用了 COM 组件技术，IE 提供了很多扩展接口供程序员调用。本章将详细介绍这方面的内容。

第 5 章　音频与视频处理。主要讨论音频控制技术、视频捕捉技术、混音技术、Flash SDK 开发技术等时下流行的多媒体技术，该部分内容对于从事多媒体软件开发的人员具有很高的参考价值。

第 6 章　COM 技术。COM 技术在 Windows 软件开发中所占的地位越来越重要，整个 Windows 系统已经全部组件化，而且绝大部分软件都已经提供了自动化 COM 接口，如 Office、OutLook、CuteFtp 等，它们都支持脚本语言（VBScript、JavaScript）的调用，本章将为读者介绍这方面的内容。

第 7 章　Windows 系统编程。主要介绍了 Windows 系统底层开发过程中经常用到的技术，主要有注册表技术、HOOK 技术、多任务编程和动态链接库技术等。

第 8 章　调试及应用技巧。主要介绍了一些程序开发过程中的心得体会，主要包括调试技巧、内存监测、动态链接库的调试技巧等，这些内容对开发人员的实际开发工作具有很高的指导意义。

由于 Windows 编程所涉及的知识面极为广泛，而作者的知识又很有限，因此尽管作者对本书中所涉及的内容一再推敲和仔细调试，仍有可能出现错误和纰漏，希望广大读者批评指正。

编者

目　录

第 1 章　控件编程

本章不仅向读者介绍了便于开发通用用户界面的基本控件的使用方法，而且还介绍了如何通过重载控件的基类来创建特色控件的方法，开发新颖的软件界面。本章介绍的实例主要包括以下内容。

- 复杂形状按钮的制作实例

介绍了如何制作各种几何形状的按钮，甚至是“@、$”等字符形状的按钮。

- 数字编辑控件

介绍了如何制作只能接受数字输入的编辑控件。

- 对鼠标敏感的编辑控件

介绍了如何制作随着鼠标动作的变化而外观作相应变化的编辑控件。

- 带有图标的菜单的制作实例

介绍了如何制作带有各种图标的菜单。

- 类似 IE 浏览器的工具条

介绍了如何制作出类似 IE 的工具条。

- 内部嵌入其他控件的状态栏

介绍了如何在状态栏中嵌入其他的控件。

- 一个超级链接控件的制作实例

介绍了一个超级链接控件的制作过程。

下面将为读者作具体的介绍。

1.1　开发特色控件

现在软件界面越来越朝着新颖、时尚的方向发展，新颖的按钮、漂亮的图片菜单、复杂的工具条，这些界面元素都为软件增色不少。

本节将介绍这些特色控件的开发技巧和应用实例，读者可以将这些实例直接应用到实际的软件项目中去，也可以在这些内容的基础上自行发挥创建出个性化的用户界面。

1.1.1　基础知识

开发特色控件首先需要理解一个概念——“自绘制”。MFC 支持自绘制（Owner-Draw）概念，自绘制的控制类通过调用 DrawItem()函数实现控件的绘制，由于控件绘制、消息检测和消息比较代码是在控件中实现的（不是在拥有控件的窗口中实现），因而叫自绘制。用户可以通过重载 DrawItem(LPDRAWITEMSTRUCT)函数来控制控件的外观和行为，实现控制所需要的参数都包含在 LPDRAWITEMSTRUCT 结构中。

下面介绍 DrawItem 函数的使用方法。

函数原型如下：

virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)

Windows 的绝大部分标准控件中都有 DrawItem 这个虚函数，例如 CButton、CListBox、CListCtrl 等。该函数只有一个参数——LPDRAWITEMSTRUCT，它是一个指向 DRAWITEMSTRUCT 类型的指针。

当一个自绘制控件的外观发生变化的时候，DrawItem 函数将会被操作系统自动调用。可以把这个函数同 OnDraw 函数进行比较，如果读者理解 OnDraw 函数，那么就能想象到 DrawItem 函数的调用方式

了。

在 DrawItem 函数中，用户可以利用 GDI 函数进行图形的绘制，随心所欲地控制控件的外观。

> **注意：绘制完毕，应该将相应的 GDI 对象保存到 LPDRAWITEMSTRUCT 结构中去，以便系统进行图形的恢复，否则会发生逻辑错误。**

下面向读者介绍 DRAWITEMSTRUCT 这个数据结构。

DRAWITEMSTRUCT 的定义如下：

```
typedef struct tagDRAWITEMSTRUCT {
    UINT    CtlType;
    UINT    CtlID;
    UINT    itemID;
    UINT    itemAction;
    UINT    itemState;
    HWND    hwndItem;
    HDC     hDC;
    RECT    rcItem;
    DWORD   itemData;
} DRAWITEMSTRUCT;
```

DRAWITEMSTRUCT 结构中包含了实现控制所需要的参数，下面具体介绍结构中各成员的含义和用法。

- CtlType

指定控件的类型。可以选择的值有 ODT_BUTTON（自绘制风格的按钮）、ODT_COMBOBOX（自绘制风格的下拉框）、ODT_LISTBOX（自绘制风格的列表框）、ODT_MENU（自绘制风格的菜单）、ODT_LISTVIEW（自绘制风格的表单视图控件）、ODT_STATIC（自绘制风格的静态控件）、ODT_TAB（自绘制风格的 TAB 控件）。

- CtlID

控件的 ID。

> **注意：对于菜单对象，不需要这个参数。**

- ItemID

对于菜单，指的是菜单项的 ID；对于列表框、下拉框，指的是列表项的 ID。

- ItemAction

指定绘制动作。可以选的值有 ODA_DRAWENTIRE（整个控件需要自行绘制）、ODA_FOCUS（控件获得、失去输入焦点）、ODA_SELECT（控件选中的相应项发生改变）。

- ItemState

当前的绘制动作发生后，指示控件的状态。可以选择的值有 ODS_CHECKED（菜单项被选中）、ODS_DISABLED（当前项被绘制成非激活状态）、ODS_FOCUS（当前项获得输入焦点）、ODS_GRAYED（当前项成为灰度状态）、ODS_SELECTED（当前项成为选中状态）、ODS_COMBOBOXEDIT（绘制下拉框的文本编辑选中区域）、ODS_DEFAULT（当前项为默认项）。

- HwndItem

自绘制控件的窗口句柄。

- Hdc

绘制控件时使用的绘制设备句柄。

- RcItem

绘制设备的矩形区域结构，指明了控件的绘制区域。

- ItemData

控件的列表项数据。

下面给出一段简单的 DrawItem 函数代码，这是从一个自绘制的图像按钮类中摘录出来的，读者可以重点理解 DrawItem 函数本身，其他的相关问题在后面的特色控件实例中会加以介绍。

```
void CMybtn::DrawItem(LPDRAWITEMSTRUCT lpdis)
{
    HBITMAP hbitmap=NULL;
    ASSERT(lpdis!=NULL);
    //lpdis- >hdc 是设备环境的句柄

    CDC *pdc=CDC::FromHandle(lpdis- >hDC);
    CRect r1;
    //得到控件的矩形范围

    r1.CopyRect(&lpdis- >rcItem);
    //得到控件的状态

   UINT state=lpdis- >itemState;
   //selected 时，用 COLOR_3DDKSHADOW 画左上部
   if((state & ODS_SELECTED))
        //COLOR_HILIGHT 画右下部，表现为凹陷
        pdc- >Draw3dRect(r1,GetSysColor
        (COLOR_3DDKSHADOW),
        GetSysColor(COLOR_3DHILIGHT));
   else
        //正常时，用 COLOR_3DHILIGHT 画左上部
        pdc- >Draw3dRect(r1,GetSysColor(COLOR_3DHILIGHT),
        GetSysColor(COLOR_3DDKSHADOW));
   COLOR_3DDKSHADOW 画右下部，表现为突起
        // TODO: Add your code to draw the specified item
        //如有图像，则装载图像
        if(m_bitmapid)
             hbitmap=(HBITMAP)LoadImage(AfxGetInstanceHandle(),
             MAKEINTRESOURCE(m_bitmapid),IMAGE_BITMAP,0,0,
             LR_DEFAULTCOLOR);
        CString s1;
        //得到 BUTTON 的 CAPTION
        GetWindowText(s1);
        if(!s1.IsEmpty())
        {
             int mode1=pdc->SetBkMode(TRANSPARENT);
             //如没有图像，则在整个 BUTTON 范围输出文字
```

```
            if(!hbitmap)
                pdc- >DrawText(s1,r1,
                DT_CENTER|DT_VCENTER|DT_SINGLELINE);
            else
            {
                CRect r2=r1;
                r2.DeflateRect(2,2);
                CDC memdc;
                CBitmap bmp;
                CBitmap *oldbitmap;
                bmp.Attach(hbitmap);
                BITMAP  bitmap;
                //由 BITMAP 结构可以得出图像的高、宽
                bmp.GetBitmap(&bitmap);
                memdc.CreateCompatibleDC(pdc);
                oldbitmap=memdc.SelectObject(&bmp);
                //把图像从内存压缩拷贝到 BUTTON 范围
                pdc- >StretchBlt(r2.left,r2.top,r2.Width()/2,
                    r2.Height(),&memdc,0,0,bitmap.bmWidth,
                    bitmap.bmHeight,SRCCOPY);
                memdc.SelectObject(oldbitmap);
                bmp.Detach();
                CRect r3=r2;
                r3.left=r2.left+r2.Width()/2;
                pdc- >DrawText(s1,r3,
                    DT_CENTER|DT_VCENTER|DT_SINGLELINE);
            }
        }
    }
```

1.1.2 特色按钮

很多流行的软件界面中常有一些独具特色的按钮，如圆形、三角形按钮等，这些按钮能够充分体现软件的个性化特点。下面将向读者介绍如何通过重载、自绘技术来制作特色按钮。

1. 圆形按钮制作实例

本例向读者介绍通过自绘制技术创建个性化的圆形按钮。读者可以将本例提供的代码直接应用到实际的软件项目中去。程序运行的结果如图 1-1 所示的方法。

上一节介绍了“自绘制”技术的基本概念，本节的圆形按钮控件就是采用该技术实现的。下面简要地列出算法步骤（具体的细节可以参考后面的具体说明）：

- 从类 Cbutton 中继承生成一个子类——CCircleBtn。
- 重载虚函数——DrawItem，在其中进行 GDI 图形绘制，绘制出圆形。
- 嵌入到控件的容器中（本例的容器是对话框）。

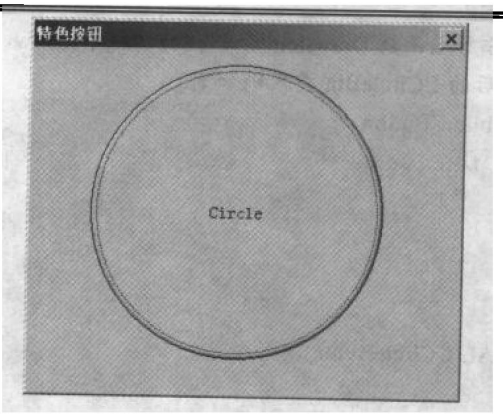

图 1-1　圆形按钮效果图

注意：有两种方式可以选择。一种是事先在资源编辑器中添加一个按钮，这样在设计的时候就可以控制按钮的位置了；另一种方式是在函数 PreSubclassWindow 中动态决定按钮控件放置的位置。

控件的创建、绘制过程如下：

- 用户或系统调用 CCircleBtn::Create 函数来创建一个按钮对象。
- 系统调用 CCircleBtn::PreSubclassWindow 函数来子类化按钮，使得该按钮嵌入到对话框容器中，同时该对象通知系统需要“自绘制”。
- 系统调用 CCircleBtn::DrawItem 进行绘制。

每当该按钮对象接收到 OnPaint 消息的时候都会进行重新绘制，直到系统将按钮销毁为止。

下面介绍具体的制作过程。

- 启动 VC++，新建一个基于对话框的项目。
- 编辑对话框资源，在对话框上放置一个按钮，如图 1-2 所示。

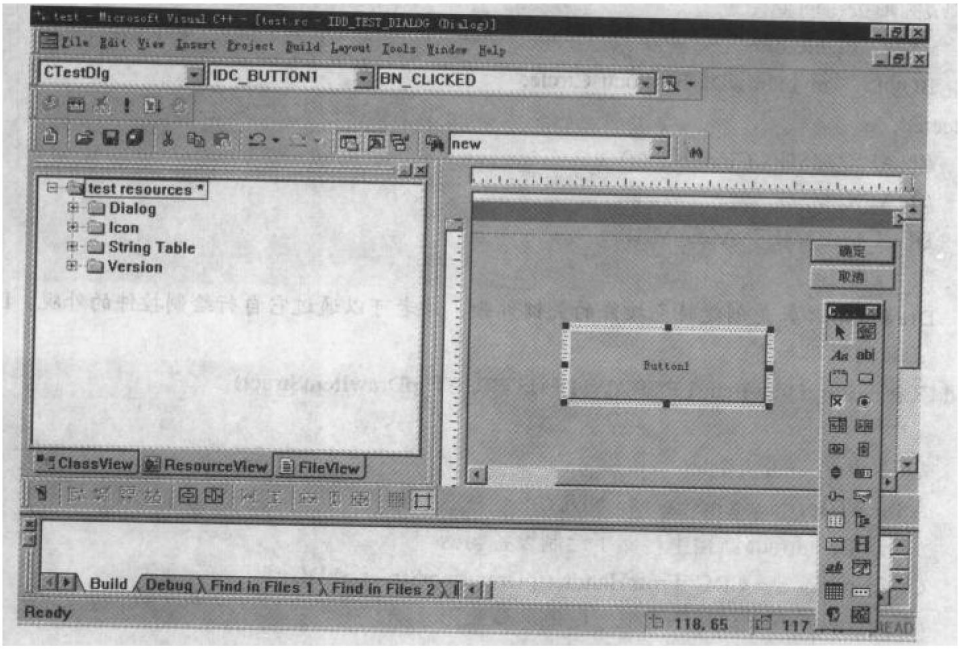

图 1-2　编辑对话框资源

- 启动 ClassWizard 新建一个类，名字为 CCircleBtn，继承于 CButton 类。

- 为 CCircleBtn 类添加虚函数 DrawItem 和 PreSubClassWindow，分别用于自绘制图形按钮和子类化控件。添加后的 CCircleBtn 类代码如下：

```
class CCircleBubtn : public CButton
{
    // 构造函数
public:
    CCircleBubtn();
public:
    //{{AFX_VIRTUAL(CCircleBubtn)
public:
    //绘制控件的外观
    virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
protected:
    //子类化
    virtual void PreSubclassWindow();
    //}}AFX_VIRTUAL

    // Implementation
public:
    virtual ~CCircleBubtn();
    //用于在 DrawItem 函数中进行区域控制
    CRgn    m_rgn;
    //圆形按钮的中心点坐标
    CPoint m_ptCentre;
    //圆形按钮的半径
    int     m_nRadius;
    BOOL    m_bDrawDashedFocusCircle;
protected:
    //{{AFX_MSG(CCircleBubtn)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

- DrawItem 函数是创建特色控件的关键部分，读者可以通过它自行绘制控件的外观。DrawItem 函数的代码如下：

```
void CCircleBtn::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
    //检验指针是否为空
    ASSERT(lpDrawItemStruct != NULL);
    //DrawItemStruct 结构中存储了绘制设备句柄
    CDC* pDC    = CDC::FromHandle(lpDrawItemStruct->hDC);
    //DrawItemStruct 结构中存储了绘制区域矩形
    CRect rect = lpDrawItemStruct->rcItem;
    //DrawItemStruct 结构中存储了控件状态
    UINT state = lpDrawItemStruct->itemState;
```

```
    //获取按钮的风格（通过 VC 资源编辑器设置的）
    UINT nStyle = GetStyle();
    //获取圆形半径
    int nRadius = m_nRadius;
    //保存当前绘制设备
    int nSavedDC = pDC->SaveDC();
    //选择画刷
    pDC->SelectStockObject(NULL_BRUSH);
    //填充矩形
    pDC->FillSolidRect(rect, ::GetSysColor(COLOR_BTNFACE));
    //根据按钮在资源编辑器中的设置状态，进行不同的外观绘制
    if ((state & ODS_FOCUS) && m_bDrawDashedFocusCircle)
        DrawCircle(pDC, m_ptCentre, nRadius--, RGB(0,0,0));
    if (nStyle & BS_FLAT) {
        DrawCircle(pDC, m_ptCentre, nRadius--, RGB(0,0,0));
        DrawCircle(pDC, m_ptCentre, nRadius--, ::GetSysColor(COLOR_3DHIGHLIGHT));
    } else {
        if ((state & ODS_SELECTED))  {
            DrawCircle(pDC, m_ptCentre, nRadius--,
                ::GetSysColor(COLOR_3DDKSHADOW),
                ::GetSysColor(COLOR_3DHIGHLIGHT));
            DrawCircle(pDC, m_ptCentre, nRadius--,
                ::GetSysColor(COLOR_3DSHADOW),
                ::GetSysColor(COLOR_3DLIGHT));
        } else {
            DrawCircle(pDC, m_ptCentre, nRadius--,
                ::GetSysColor(COLOR_3DHIGHLIGHT),
                ::GetSysColor(COLOR_3DDKSHADOW));
            DrawCircle(pDC, m_ptCentre, nRadius--,
                ::GetSysColor(COLOR_3DLIGHT),

                ::GetSysColor(COLOR_3DSHADOW));
        }
    }

    CString strText;
    //获取设置时输入的标题文字
    GetWindowText(strText);
    if (!strText.IsEmpty())
    {
        CRgn rgn;
        //创建圆形区域
        rgn.CreateEllipticRgn(m_ptCentre.x-nRadius, m_ptCentre.y-nRadius,
            m_ptCentre.x+nRadius,
```

```
            m_ptCentre.y+nRadius);
        pDC->SelectClipRgn(&rgn);
        //获取文字的大小
        CSize Extent = pDC->GetTextExtent(strText);
        CPoint pt = CPoint( m_ptCentre.x - Extent.cx/2, m_ptCentre.x - Extent.cy/2 );
        if (state & ODS_SELECTED) pt.Offset(1,1);
        //设置文字背景透明化
        pDC->SetBkMode(TRANSPARENT);
        if (state & ODS_DISABLED)
            pDC->DrawState(pt, Extent, strText, DSS_DISABLED, TRUE, 0, (HBRUSH)NULL);
        else
            pDC->TextOut(pt.x, pt.y, strText);

        pDC->SelectClipRgn(NULL);
        rgn.DeleteObject();
    }
    //绘制按钮虚线
    if ((state & ODS_FOCUS) && m_bDrawDashedFocusCircle)
        DrawCircle(pDC, m_ptCentre, nRadius-2, RGB(0,0,0), TRUE);
    //恢复绘制设备句柄
    pDC->RestoreDC(nSavedDC);
}
```

- 为了使新创建的控件能够工作，还需要处理 Windows 消息，即重载 PreSubClassWindow 函数，代码如下：

```
void CCircleBtn::PreSubclassWindow()
{
    CButton::PreSubclassWindow();
    //将控件的风格改为自绘制
    ModifyStyle(0, BS_OWNERDRAW);
    CRect rect;
    //获取客户区的矩形
    GetClientRect(rect);
    rect.bottom = rect.right = min(rect.bottom,rect.right);
    m_ptCentre = rect.CenterPoint();
    m_nRadius  = rect.bottom/2-1;
    m_rgn.DeleteObject();
    SetWindowRgn(NULL, FALSE);
    m_rgn.CreateEllipticRgnIndirect(rect);
    //将控件窗口区域重新设置
    SetWindowRgn(m_rgn, TRUE);
    ClientToScreen(rect);
    CWnd* pParent = GetParent();
    //移动到正确的位置
```

```
    if (pParent) pParent->ScreenToClient(rect);
    MoveWindow(rect.left, rect.top, rect.Width(), rect.Height(), TRUE);
}
```

更为详细的代码，读者可以参考配书光盘中的完整代码。

2. 复杂形状的按钮实例

上一节向读者介绍了如何通过重载 DrawItem 函数来创建圆形的按钮，本节在前面的基础上介绍创建更为复杂的按钮的方法。程序运行的结果如图 1-3 所示。

图 1-3　复杂形状按钮程序运行效果

这个例子也是基于“自绘制”技术创建的，不同的地方在于按钮的形状不是普通的几何图形而是字符。其实现的过程与前面介绍的圆形按钮基本类似，这里就不做过多的说明。下面重点介绍一下字符图形的绘制方法。

Windows GDI 提供了将任意路径转换为区域的函数——PathToRegion，该函数能够将绘制设备 HDC 上的连续路径转化为 HRGN 对象。而路径的绘制需要配合函数 BeginPath 和 EndPath 使用，使用 BeginPath(HDC hDC) 函数开始路径的选择，使用 EndPath(HDC hdc) 函数结束选择路径的选择。可以通过选择不同的 GDI 函数实现，如选择用 TextOut 函数来输出字符，则可以实现字符形状的按钮。可以用来绘制路径的 GDI 函数如下：

AngleArc，绘制扇形，适用于 Windows NT/2000。
Arc，绘制圆弧，适用于 Windows NT/2000。
ArcTo，绘制圆弧，适用于 Windows NT/2000。
Chord，绘制弦，适用于 Windows NT/2000。
CloseFigure，封闭图形，适用于 Windows 95/98 和 Windows NT/2000。
Ellipse，绘制椭圆，适用于 Windows NT/2000。
ExtTextOut，绘制文字，适用于 Windows 95/98 和 Windows NT/2000。
LineTo，绘制直线，适用于 Windows95/98 和 Windows NT/2000。
MoveToEx，绘制直线，适用于 Windows 95/98 和 Windows NT/2000。
Pie，绘制圆饼，适用于 Windows NT/2000。
PolyBezier，绘制毕塞尔曲线，适用于 Windows 95/98 和 Windows NT/2000。
PolyBezierTo，绘制毕塞尔曲线，适用于 Windows 95/98 和 Windows NT/2000。
PolyDraw，绘制多边形，适用于 Windows NT/2000。
Polygon，绘制多边形区域，适用于 Windows 95/98 和 Windows NT/2000。
Polyline，绘制多边，适用于 Windows 95/98 和 Windows NT/2000。

PolylineTo，绘制多边，适用于 Windows 95/98 和 Windows NT/2000。
PolyPolygon，绘制多边，适用于 Windows 95/98 和 Windows NT/2000。
PolyPolyline，绘制多边，适用于 Windows 95/98 和 Windows NT/2000。
Rectangle，绘制矩形，适用于 Windows NT/2000。
RoundRect，绘制圆角矩形，适用于 Windows NT/2000。
TextOut，绘制文字，适用于 Windows 95/98 和 Windows NT/2000。
PathToRegion(HDC hDC)返回由 BeginPath、EndPath 选中的路径所包围的区域。
具体的制作过程如下。

- 启动 VC++，新建对话框项目，去掉对话框上的所有默认选项。
- 新建一个按钮类——CAdvButton，继承于 CButton。
- 本例比较复杂，需要处理大量的 Windows 鼠标消息，因此需要添加许多消息处理函数，它们都存放在 CAdvButton 类的头文件中，代码如下：

```
class CAdvButton : public CButton
{
public:
    //{{AFX_VIRTUAL(CAdvButton)
public:
    //自绘制函数
    virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
protected:
    //子类化
    virtual void PreSubclassWindow();
    //Windows 消息处理
    virtual LRESULT DefWindowProc(UINT message, WPARAM wParam, LPARAM lParam);
    //}}AFX_VIRTUAL

protected:
    //{{AFX_MSG(CAdvButton)
    //Windows 消息处理-背景擦除
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    //Windows 消息处理-鼠标移动
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    //Windows 消息处理-创建
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //鼠标左键按下
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    //鼠标左键弹起
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
private:
    //按钮边界框的宽度
    UINT m_nBorder;
```