Visual C++ Visual C++

VC 篇

◉清华天则工作室 / 编著

编程技巧及故障排除即时通

控件编程常见问题及解答

核心编程常见问题及解答

网络和通信常见问题及解答

图形和打印常见问题及解答

组件技术常见问题及解答

其他常见问题及解答

界面编程常见问题及解答

兵器工业出版社

编程技巧及故障排除即时通

——VC 篇

清华天则工作室　编著

兵器工业出版社

前　言

Windows这个优秀的图形化操作系统，如今已经深入人心，成了PC市场的绝对主流操作系统。相应地，Windows程序员的队伍也日渐壮大。而在Windows应用程序的开发领域中，使用最多的开发环境莫过于微软自己推出的Visual C++和Visual Basic了。

关于Visual C++，目前市面上的书已是汗牛充栋，但笔者在使用Visual C++进行各种各样的Windows应用程序的开发过程中，却发现很难找到能及时为各种技术问题提供帮助指导的索引式编程参考书，这便是本书成书的一大理由。笔者在年深日久的编程过程中积累起来许多资料，从中取其精华，去其糟粕，编成本书。希望读者能从中获益，在进行各种各样的Windows应用程序开发时，能随时从本书中发现自己所遇到的问题的答案。

本书具有如下几个特点：

知识速查：针对编程过程可能遇到的各种问题，提供索引式的检取方案，以备读者速查。

循序渐进：在编写本书过程中，针对每一个大类的问题（即书中的每一章），尽量按照“由浅入深”的顺序来回答各种问题，以便于读者的学习。

深入浅出：作者站的角度较高，能深入探讨编程原理及技巧，而且又是本国作者，能以流利、规范、通俗易懂的语言和详尽的实例来说明。

实例详尽：书中的难点、疑点尽量利用实例来进行说明，以便于理解掌握。同时在每个细节中都尽可能地给出详尽的实例，从而避免由于一些细微的错误而导致不能掌握某个知识点。

内容全面：本书所包含的内容比较全面，涉及到Visual C++编程的各个侧面，可以适应不同层次读者开发不同应用程序的需要。

在本书的编著过程中，笔者尽量注意减少冗长无味的说明，代之以具体实用的实例演示。通过实例，引导读者把握精髓所在。本书注重开发实例、开发经验、开发技巧和Windows高级特性的开发，适应用户的急需。

读者在使用本书时，既可以循序渐进地阅读，也可以直接查阅自己感兴趣的话题，进行索引式的阅读。下面对本书的内容做一个简单的概述：

“第一章 界面编程常见问题及解答”介绍了使用Visual C++进行界面编程时经常遇到的一些问题及相关的解决方案，其中具体的界面编程问题涉及到窗口和程序实例、菜单、对话框等各个方面。

“第二章 控件编程常见问题及解答”介绍了在Visual C++中使用Windows控件

时经常遇到的一些问题及相关的解决方案，其中具体的控件编程问题涉及到普通控件、工具条、状态条等。

“第三章 核心编程常见问题及解答”介绍了使用 Visual C++ 进行深层次的 Windows 应用程序编程时经常遇到的一些问题及相关的解决方案，其中具体的核心编程问题涉及到消息处理、钩子函数、进程、线程、资源、DLL 和 VxD 等各个方面。

“第四章 图形和打印常见问题及解答”介绍了使用 Visual C++ 进行编程时经常遇到的图形绘制和打印方面的问题及相关的解决方案。

“第五章 网络和通信常见问题及解答”介绍了使用 Visual C++ 进行网络编程时经常遇到的一些问题及相关的解决方案，其中具体的网络编程问题涉及到 WinSock、Http、Ftp、Gopher、串口和并口等各个方面。

“第六章 组件技术常见问题及解答”介绍了使用 Visual C++ 进行组态软件编程时经常遇到的一些问题及相关的解决方案，其中重点包括当今流行的 COM 和 DCOM。

“第七章 其他常见问题及解答”覆盖了 Visual C++ 编程的其他各个侧面，主要包括数据库编程、Windows 外壳程序、汉字技术、MSDEV 集成环境等。

本书适用于使用 Visual C++ 进行 Windows 编程的各个层次的程序员。限于作者水平，难免在选材和叙述上有不当之处。欢迎广大读者对本书提出批评和建议。

目　录

第一章　界面编程常见问题及解答

第一节　窗口和程序实例的操作

第二节 菜单

第三节 对话框

第二章 控件编程常见问题及解答

第一节 普通控件

第二节　工具条和状态条

第三章　核心编程常见问题及解答

第一节　消息处理及钩子函数

第二节　进程和线程

第三节　资源

第四节 通用非窗口 MFC 类

第五节 SDI/MDI 程序结构

第六节 DLL 和 VxD

第四章 图形和打印常见问题及解答

第一节 窗口绘图

第二节　OpenGL 编程

第三节　打印输出

第五章　网络和通信常见问题及解答

第一节　WinSock 编程

第二节　HTTP、FTP、Gopher

第三节　串口和并口

第四节　进程间通信

第五节　其它

第六章　组件技术常见问题及解答

第一节　COM

第二节　DCOM

第七章　其他常见问题及解答

第一节　MSDEV 集成环境

第二节　文件及目录操作

第三节　汉字技术及其它

第四节　Windows 外壳程序

第五节　数据库技术

第六节　一些非技术问题

第一章　界面编程常见问题及解答

第一节　窗口和程序实例的操作

一　如何访问桌面窗口

静态函数 CWnd::GetDesktopWindow 返回桌面窗口的指针。下例说明了 MFC 函数 CFrameWnd::BeginModalStae 是如何使用该函数进入内部窗口列表的。

```
void CFrameWnd::BeginModalState()
{
    …
    //first count all windows that need to be disabled
    UINT nCount = 0;
    HWND hWnd = ::GetWindow(::GetDesktopWindow(),GW_CHILD);
    while(hWnd! = NULL)
    {
    if(::IsWindowEnabled(hwnd)&&
    CWnd::FromHandlePermanent(hWnd)! = NULL &&
    AfxIsDescendant(pParent - > m_hWnd,hWnd)&&
    ::SendMessage(hWnd,WM_DISABLEMODAL,0,0) = = 0)
    {
    + + nCount;
    }
    hWnd = ::GetWindow(hWnd,GW_HWNDNEXT);
    }
    …
```

```
}
```

二　怎么修改主窗口的风格

AppWizard生成的应用程序框架的主窗口具有缺省的窗口风格，比如在窗口标题条中自动添加文档名、窗口是迭加型的、窗口大小可改变等。要修改窗口的缺省风格，需要重载CWnd::PreCreateWindow(CREATESTRUCT& cs)函数，并在其中修改CREATESTRUCT型参数cs。

CWnd::PreCreateWindow函数先于窗口创建函数执行。如果该函数被重载，则窗口创建函数将使用CWnd::PreCreateWindow函数返回的CREATESTRUCT cs参数所定义的窗口风格来创建窗口；否则使用预定义的窗口风格。

CREATESTRUCT结构定义了创建函数创建窗口所用的初始参数，其定义如下：

```
typedef struct tagCREATESTRUCT {
LPVOID lpCreateParams;//创建窗口的基本参数
HANDLE hInstance;//拥有将创建的窗口的模块实例句柄
HMENU hMenu;//新窗口的菜单句柄
HWND hwndParent;//新窗口的父窗口句柄
int cy;//新窗口的高度
int cx;//新窗口的宽度
int y;//新窗口的左上角Y坐标
int x;//新窗口的左上角X坐标
LONG style;//新窗口的风格
LPCSTR lpszName;//新窗口的名称
LPCSTR lpszClass;//新窗口的窗口类名
DWORD dwExStyle;//新窗口的扩展参数
} CREATESTRUCT;
```

CREATESTRUCT结构的style域定义了窗口的风格。比如，缺省的MDI主窗口的风格中就包括FWS_ADDTOTITLE(在标题条中显示当前的工作文档名)、FWS_PREFIXTITLE(把文档名放在程序标题的前面)、WS_THICKFRAME(窗口具有可缩放的边框)等风格。由于多种风格参数由逻辑或("|")组合在一起的，因此如果想要添加某种风格，只需用"|"把对应的参数加到CREATESTRUCT结构的style域中；而删除已有的风格，则只需用"&"连接CREATESTRUCT结构的style域与该风格的逻辑非值。

CREATESTRUCT结构的x、y、cx、cy域分别定义了窗口的初始位置和大小，因此，在CWnd::PreCreateWindow函数中给它们赋值，将能定义窗口的初始显示位置和大小。

下例中的代码将主框窗口的大小固定为1/4屏幕，标题条中仅显示窗口名，不显示文档名。

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    //TODO:Modify the Window class or styles here by modifying
    //the CREATESTRUCT cs
    //修改主窗风格
    cs.style &= FWS_ADDTOTITLE;//去除标题条中的文档名
    cs.style &= WS_THICKFRAME;//去除可改变大小的边框
    cs.style |= WS_DLGFRAME;//增加不能改变大小的边框
    //确定主窗的大小和初始位置
    int cxScreen = ::GetSystemMetrics(SM_CXSCREEN);//获得屏幕宽
```

```
    int cyScreen = ::GetSystemMetrics(SM_CYSCREEN);//获得屏幕高
    cs.x = 0;//主窗位于左上角
    cs.y = 0;
    cs.cx = cxScreen/2;//主窗宽为 1/2 屏幕宽
    cs.cy = cxScreen/2;//主窗高为 1/2 屏幕高
    return CMDIFrameWnd::PreCreateWindow(cs);
}
```

三　如何改变窗口标题

调用 CWnd::SetWindowText 可以改变任何窗口(包括控件)的标题。

```
//Set title for application's main frame window.
AfxGetMainWnd() - > SetWindowText(_T("Application title"))
//Set title for View's MDI child frame window.
GetParentFrame() - > SetWindowText("_T("MDI Child Frame new title"))
//Set title for dialog's push button control.
GetDigitem(IDC_BUTTON) - > SetWindowText(_T("Button new title"))
```

如果需要经常修改窗口的标题(注:控件也是窗口),应该考虑使用半文档化的函数 AfxSetWindowText。该函数在 AFXPRIV.H 中说明,在 WINUTIL.CPP 中实现,在联机帮助中找不到它,它在 AFXPRIV.H 中半文档化,在以后发行的 MFC 中将文档化。

AfxSetWindowText 的实现如下:

```
voik AFXAPI AfxSetWindowText(HWND hWndCtrl,LPCTSTR IpszNew)
{
    itn nNewLen = Istrlen(Ipaznew)
    TCHAR szOld [256]
    //fast check to see if text really changes(reduces
    flash in the
    controls)
    if(nNewLen〉_contof(szOld)
    ||::GetWindowText(hWndCrtl,szOld,_countof(szOld)! = nNewLen
    || Istrcmp(szOld,IpszNew)! = 0
    {
    //change it
    ::SetWindowText(hWndCtrl,IpszNew)
    }
}
```

四　如何单击除了窗口标题栏以外的区域使窗口移动

当窗口需要确定鼠标位置时 Windows 向窗口发送 WM_NCHITTEST 信息,程序可以处理该信息使 Windows 认为鼠标在窗口标题上。对于对话框和基于对话的应用程序,可以使用 ClassWizard 处理该信息并调用基类函数,如果函数返回 HTCLIENT 则表明鼠标在客房区域,返回 HTCAPTION 表明鼠标在 Windows 的标题栏中。

```
UINT CSampleDialog::OnNcHitTest(Cpoint point)
{
    UINT nHitTest = Cdialog::OnNcHitTest(point)
    return(nHitTest = = HTCLIENT)? HTCAPTION:nHitTest
}
```

上述技术有两点不利之处:其一是在窗口的客户区域双击时,窗口将极大;其二,它不适合包含几个窗口的主框窗口。

还有一种方法,用户按下鼠标左键使主框窗口认为鼠标在其窗口标题上,使用 ClassWizard 在窗口中处理 WM_LBUTTODOWN 信息,并向主框窗口发送一个 WM_NCLBUTTONDOWN 信息和一个单击测试 HTCAPTION。

```
void CSampleView::OnLButtonDown(UINT nFlags,Cpoint point)
{
    CView::OnLButtonDow(nFlags,point)
    //Fool frame window into thinking somene clicked onits caption bar.
    GetParentFrame() - > PostMessage(
    WM_NCLBUTTONDOWN,
    HTCAPTION,MAKELPARAM(poitn.x,point.y))
}
```

该技术也适用于对话框和基于对话的应用程序,只是不必调用 CWnd::GetParentFrame。

```
void CSampleDialog::OnLbuttonDown(UINT nFlags,Cpoint point)
{
    Cdialog::OnLButtonDow(nFlags,goint)
    //Fool dialog into thinking simeone clicked on itscaption bar.
    PostMessage(WM_NCLBUTTONDOWN,HTCAPTION,MAKELPARM(point.x,point. y))
}
```

五　如何让窗口和 MDI 窗口一启动就最大化或者最小化

1. 普通窗口

在 InitStance 函数中设定 m_nCmdShow 的取值。

```
m_nCmdShow = SW_SHOWMAXMIZED//最大化
m_nCmdShow = SW_SHOWMINMIZED//最小化
m_nCmdShow = SW_SHOWNORMAL//正常方式
```

2. MDI 窗口

如果是创建新的应用程序,可以用 MFC AppWizard 的 Advanced 按钮,并在 MDI 子窗口风格组中选取最大化或最小化,还可以重载 MDI Window 的 PreCreateWindow 函数,设置 WS_MAXMIZE or WS_MINMIZE。

如果从 CMDIChildWnd 派生,调用 OnInitialUpdate 函数中的 CWnd::Show Window 来指定 MDI Child Window 的风格。

六　如何创建一个不规则形状的窗口

标准的 Windows 窗口是矩形的,但在有些时候我们需要非矩形的窗口,比如圆形的、甚至是不规则

的。借助 CWnd 类的 SetWindowRgn 函数可以创建不规则形状窗口。

CWnd::SetWindowRgn 的函数原型如下：

```
int SetWindowRgn(HRGN hRgn,//窗口区域句柄
BOOL bRedraw);//是否重画窗口
```

CRgn 类封装了关于区域的数据和操作。通过(HRGN)强制操作可以从 CRgn 类中取得其 HRGN 值。

CRgn 提供了 CreateRectRgn、CreateEllipticRgn 和 CreatePolygonRgn 成员函数，分别用以创建矩形、(椭)圆形和多边形区域。

创建非矩形窗口的方法如下：

首先，在窗口类中定义区域类成员数据(如 CRgn m_rgnWnd)；其次，在窗口的 OnCreate 函数或对话框的 OnInitDialog 函数中调用 CRgn 类的 CreateRectRgn、CreateEllipticRgn 或 CreatePolygonRgn 函数创建所需的区域，并调用 SetWindowRgn 函数。

下例将生成一个椭圆窗口。

① 在 Developer Studio 中选取 File 菜单中的 New 命令，在出现的 New 对话框中选择创建 MFC AppWizard(exe)框架应用程序，并输入项目名为 EllipseWnd。设定应用程序类型为基于对话框(Dialog based)，其它选项按缺省值创建项目源文件。

② 使用资源编辑器从主对话框(ID 为 IDD_ELLIPSEWND_DIALOG)删除其中的所有控制，并从其属性对话框(Dialog Properties)中设定其风格为 Popup、无标题条和边框。

③ 在 EllipseWndDlg.h 源文件中给主对话框类 CEllipseWndDlg 增加一个 CRgn 类型的保护型数据成员 m_rgnWnd，它将定义窗口的区域。

④ 在 EllipseWndDlg.cpp 源文件中修改主对话框类 CEllipseWndDlg 的 OnInitDialog()函数，增加 m_rgnWnd 的创建过程，并将其定义为窗口区域。

```
BOOL CEllipseWndDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    //Add"About……"menu item to system menu.
    //IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) = = IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX<0xF000);
    CMenu * pSysMenu = GetSystemMenu(FALSE);
    if(pSysMenu ! = NULL)
    {
    CString strAboutMenu;
    strAboutMenu.LoadString(IDS_ABOUTBOX);
    if(! strAboutMenu.IsEmpty())
    {
    pSysMenu - > AppendMenu(MF_SEPARATOR);
    pSysMenu - > AppendMenu(MF_STRING,IDM_ABOUTBOX,
    strAboutMenu);
    }
    }
    //Set the icon for this dialog. The framework does this automatically
    //when the application's main window is not a dialog
    SetIcon(m_hIcon,TRUE);//Set big icon
```

```
    SetIcon(m_hIcon,FALSE);//Set small icon
    //设置窗口标题为"椭圆窗口",虽然对话框没有标题条,
    //但在任务条的按钮中仍需要标题
    SetWindowText(_T("椭圆窗口"));
    //取得屏幕宽、高
    int cxScreen = ::GetSystemMetrics(SM_CXSCREEN);
    int cyScreen = ::GetSystemMetrics(SM_CYSCREEN);
    //设置椭圆 X、Y 方向的半径
    int nEllipseWidth = cxScreen/8;
    int nEllipseHeight = cyScreen/8;
    //将窗口大小设为宽 nEllipseWidth,高 nEllipseHeight
    //并移至左上角
    MoveWindow(0,0,nEllipseWidth,nEllipseHeight);
    //创建椭圆区域 m_rgnWnd
    m_rgnWnd.CreateEllipticRgn(0,0,nEllipseWidth,nEllipseHeight);
    //将 m_rgnWnd 设置为窗口区域
    SetWindowRgn((HRGN)m_rgnWod,TRUE);
    return TRUE;//return TRUE unless you set the focus to a control
}
```

除了借助 CWnd 类的 SetWindowRgn 函数之外,还可以使用新的 SDK 函数 SetWindowRgn。该函数将绘画和鼠标消息限定在窗口的一个指定的区域,实际上使窗口成为指定的不规则形状。使用 AppWizard 创建一个基于对话的应用程序,并使用资源编辑器从主对话资源中删除所在的缺省控件、标题以及边界。

给对话类增加一个 CRgn 数据成员,以后要使用该数据成员建立窗口区域。

```
Class CRoundDlg:public CDialog
{
    ...
    private:
    Crgn m_rgn://window region
    ...
}
```

修改 OnInitDialog 函数建立一个椭圆区域并调用 SetWindowRgn 将该区域分配给窗口:

```
BOOL CRoundDlg::OnInitDialog()
{
    CDialog::OnInitDialog()
    //Get size of dialog.
    CRect rcDialog
    GetClientRect(rcDialog)
    //Create region and assign to window.
    m_rgn. CreateEllipticRgn(0,0,rcDialog.Width(),rcDialog.Height())
    SetWindowRgn(GetSafeHwnd(),(HRGN)m_rgn,TRUE)
    return TRUE
}
```

通过建立区域和调用 SetWindowRgn,已经建立一个不规则形状的窗口,下面的例子程序修改了 On-