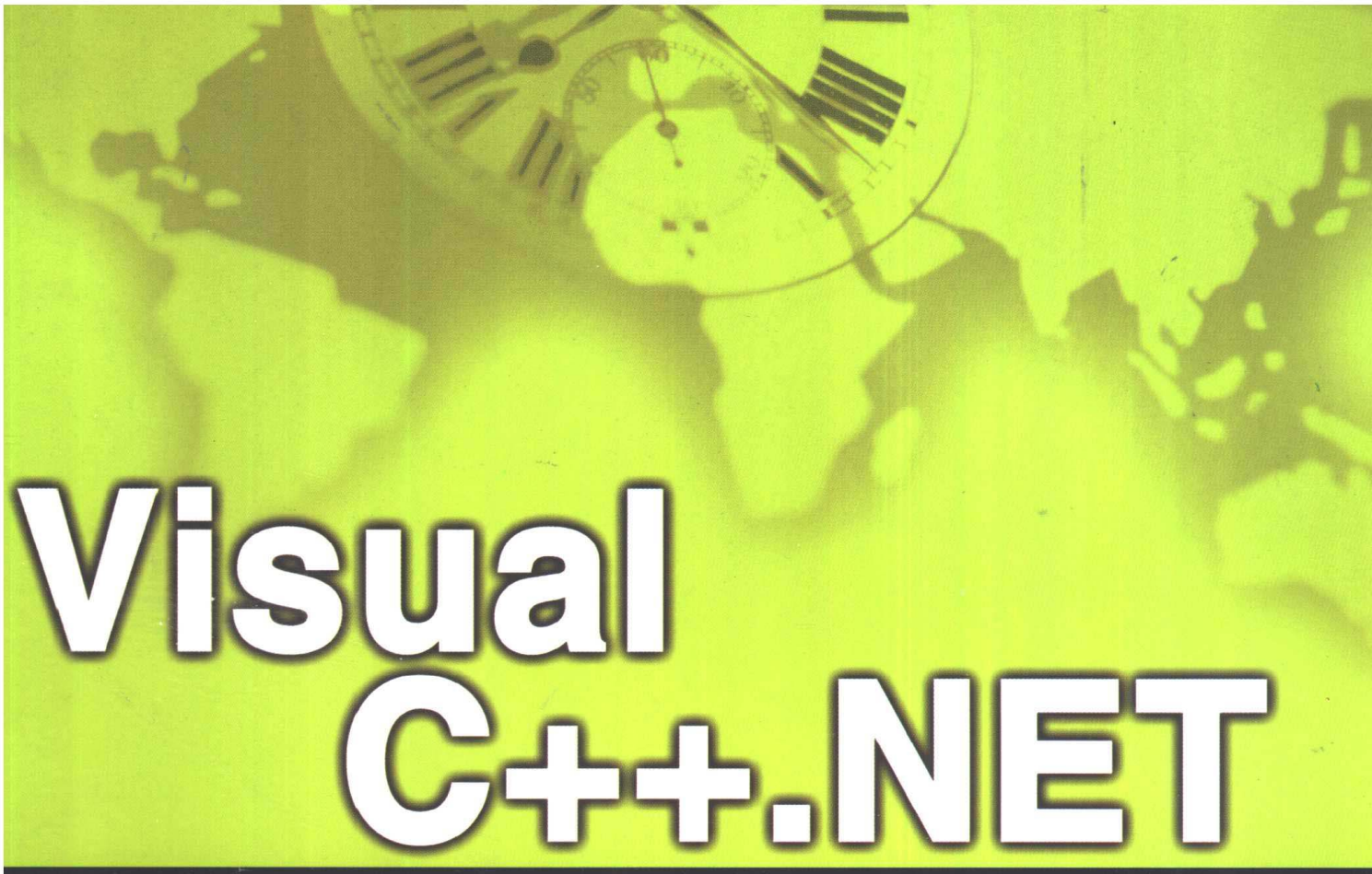

Visual C++.NET 实用编程技术

精锐创作组　编著

人民邮电出版社
www.pptph.com.cn

Visual C++ .NET 实用编程技术

精锐创作组　　编著

人民邮电出版社

图书在版编目（CIP）数据

Visual C++ .NET 实用编程技术 / 精锐创作组编著. —北京：人民邮电出版社，2001.12

ISBN 7-115-09844-1

I. V... II. 精... III. C 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2001）第 081153 号

内 容 提 要

本书从 Visual C++ .NET 的编程基础开始，以示例的形式由浅入深地介绍了在 Visual C++ .NET 中开发 Windows 应用程序的技术，包括文档与视、对话框编程、ActiveX 技术、线程与消息、数据库与网络等。

本书通俗易懂，讲解细致，并对大量示例进行了透彻的分析，适合中级程序开发人员学习使用。对于从事 Visual C++ .NET 开发与应用的广大科研人员和高校相关专业的师生而言，本书也不失为一本有价值的自学读物或教学参考书。

Visual C++ .NET 实用编程技术

◆ 编　　著　精锐创作组
责任编辑　张瑞喜　姚予疆
执行编辑　牛　磊

◆ 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@ pptph.com.cn
网址　http://www.pptph.com.cn
读者热线　010－67129212　010－67129211(传真)
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：19.5
字数：473 千字　　2001 年 12 月第 1 版
印数：1－5 000 册　　2001 年 12 月北京第 1 次印刷

ISBN 7-115-09844-1/TP·2593

定价：28.00 元

前　言

Visual C++ .NET 是运行于 Windows 平台上的交互式的可视化集成开发环境，它是美国 Microsoft 公司开发的 Microsoft Visual Studio.net 套件的一部分。像其他的可视化集成开发环境（如 Visual Basic、Delphi、C++ Builder）一样，Visual C++ .NET（为了叙述方便，以下简称为 VC）集程序的代码编辑、编译、连接和调试等功能于一体，为编程人员提供了一个完整方便的开发界面和许多有效的辅助开发工具。VC 的应用程序向导可以为很大一部分类型的程序提供框架代码，用户不用书写程序代码，只需单击几个按钮就可以生成一些完整的可以运行的程序。

本书通过丰富的示例程序向读者介绍如何使用 VC 开发Windows 应用程序，这些示例程序都是作者根据自己使用和开发 VC 程序时的体会精心编写的，是针对程序员在开发过程中最迫切需要、使用频率最高的内容特意定制的，可以说符合了中级程序员的需求。另外，本书中所有示例程序的代码都经过了严格的调试和测试，读者只要按照书中的步骤往下做，最终一定能够圆满地完成程序。

本书章节内容安排如下：

MFC 框架中最容易被程序员和用户两者都见到的部分就是文档和视。利用应用框架编写程序时的大部分工作是编写视类和文档类。因此，我们有必要详细了解文档和视。第 1 章我们结合单文档、多文档以及多窗口的应用来详细阐述这些内容。

几乎所有的 Windows 应用程序都要通过对话框和用户打交道，这种对话框可以是简单的消息框，也可以是复杂的属性控制对话框。对话框通过其上面的各种控件能够产生丰富的界面，实现各种各样的功能。第 2 章讲述了基于对话框的应用程序的基本编程思路。首先用一个实例程序说明对话框编程的基本过程，顺便介绍了基于对话框的应用程序结构（类、头文件、源文件以及资源）和 MFC 程序的工作流程，然后从属性设置、类的成员函数以及窗口事件处理等角度介绍了几种经常用于对话框的简单控件。

对 Windows 应用程序而言，设计一个优秀的程序界面并不是一件太困难的事，第 3 章中主要对增强工具条和菜单功能做了一些介绍，同时结合例子程序 AdvBar 介绍了一些 MCI 的基本知识。

ActiveX 是 OLE 技术在 Internet 上的扩展，它由某些与 Internet 相关的技术和控件所构成。ActiveX 控件是 ActiveX 体系中很重要的一个组成部分。第 4 章在简要介绍 ActiveX 体系的基础上，重点通过实例介绍了如何创建和调试一个 ActiveX 控件。

以创建线程、线程间的通信和线程同步等内容开始，第 5 章首先介绍了程序中如何启动和管理辅助线程的执行，以及如何同步化线程的活动。然后介绍了如何进行消息映射，如何手工添加消息映射入口，以及如何实现消息处理函数的内容。同时，本章中还介绍了如何实现用户界面对象的更新显示等内容。

利用 VC 可以很方便地设计出功能强大的数据库和网络应用程序，不但可以访问本地数据库，也能够访问远程大型数据库。另外，Windows 提供了许多 API 函数来支持 Internet 的编程，但是这需要较多的网络知识，因此，第 6 章着重介绍了使用 MFC 中的 WinInet 类进行编程的

方法，并举例说明了如何建立自己的网络应用程序。

本书由精锐创作组编写，参加工作的人员还有尹立民、管蒙蒙、金海涛、王传新、苗娜、鲍栋、郝天婷、李红、李立杰、李涛、崔晓东、崔汇枝、吕君、王春玲、刘志红、张国梁、钱艳霞、吕红荣、吕波、张意如、张红梅、杨华、杨振、王婧、叶莹莹、孔立、展小涛等编写。

由于编写时间紧，作者水平有限，书中难免存在一些不足之处，恳请读者批评指正。

E-mail：pptph@263.net

编著者

目　录

第 1 章
文档与视

MFC 框架中最容易被程序员和用户见到的部分就是文档和视。利用应用框架编写程序时的大部分工作是编写视类和文档类。因此，我们有必要详细了解文档和视结构。

文档和视结构是基于 MFC 库应用的一个重要特性。它的实质就是将数据本身与用户对数据的观察和操作分离开来。所有的数据变化都在文档类中进行管理，同时它为视的访问提供一个接口。视调用两者间的接口来响应用户的操作，进行数据的修改，并且不断更新对文档的显示。这样就允许对同一数据可以有多个视。了解这一特性对于我们的应用开发有着极为重要的意义。

文档、关联的视以及包含视的主框架窗口由一个文档模板创建。文档模板有责任创建和管理一个文档类的所有文档对象。我们将结合单文档以及多文档的应用来详细阐述这些内容。

分析一个应用程序

我们首先分析由 MFC Application Wizard 自动创建的一个 SDI 应用程序，了解有关文档和视的一些最基本的东西。

1.1.1 建立一个应用程序

首先我们利用 MFC Application Wizard 建立一个基于单文档应用的项目 SingleDoc1。具体的选项如下：

- 新建一个 MFC Application，在 Project Name 框中键入 SingleDoc1。
- 选择基于单文档的应用，选取资源语言为英语，选中文档／视结构支持。
- 不要选中 ActiveX Control。

这样，我们就建立了一个基于单文档应用的项目。

接下来我们将结合 MFC Application Wizard 产生的代码讲述一些基本的知识。

1.1.2 程序运行的流程

当应用程序开始运行时，它会首先查找程序中的静态和全程变量。其中最重要的就是 CWinApp 类所创建的实例。由 MFC Application Wizard 创建的代码如下所示：

```
// The one and only CSingleDoc1App object
CSingleDoc1App theApp;
```

该代码首先通过初始化数据段来建立全局变量，以及建立一些 MFC 内部使用的对象，

然后执行 CWinApp 类的构造函数。

当执行 CWinApp 类的构造函数时，应用实例并没有进入正常的运行，所以不要在这些构造函数中做重要的操作，但是可以在这里初始化变量。如果在这里进行 CWnd 类的操作，可能会出现错误，因为实际的 CWnd 类对象尚未创建完毕。

在所有的静态对象的构造工作完成以后，运行时间库就会调用 WinMain 函数。该函数使我们想起 Dos 环境下程序中的 main 函数。的确，两者有许多类似之处。它是每一个 Windows 应用程序都必须具备的。我们无法在 MFC Application Wizard 产生的源代码当中找到它，因为它被隐藏在应用框架内部了。

WinMain 函数首先调用 CWinApp 类的 InitApplication 函数和 InitInstance 函数进行应用的初始化。InitInstance 函数会完成一系列的初始化工作，其中最重要的是完成主框架窗口的构造和显示工作。

在工作完成后，它会调用 CWinApp 类的 Run 函数（缺省状态下为 CWinThread::Run()）来进入应用程序的消息循环。这时，它开始等待消息并对接收到的消息调用进行分类，然后将它传给相应的消息处理函数做相应的处理。

当接收到表示程序终止的 WM_QUIT 消息时，MFC 会调用 CWinApp 类的 ExitInstance 函数。随后调用静态对象的析构函数，删除包括 CWinApp 对象在内的全局和静态对象。最后将控制权交还给操作系统。

1.1.3 框架窗口类

在 MFC Application Wizard 产生的文件中，可以找到 MainFrm.cpp 和 MainFrm.h 文件。在其中定义了一个 CMainFrame 的类。

这个 CMainFrame 类定义了程序的主框架窗口。它是由框架窗口类 CFrameWnd 派生出来的，而 CFrameWnd 又是由窗口类 CWnd 派生出来的。对于单文档应用来讲，主框架窗口定义了一切的应用程序界面的特性。它是程序与用户之间交换信息的主要界面。

当程序的一个新的实例运行时，系统根据文档模板生成新的对象，这些对象包括一个新的 CMainFrame 对象、一个新的 CSingleDoc1Doc 对象、一个新的 CSingleDoc1View 对象。屏幕上出现的是标准框架窗口，接着生成文档，接着生成视。

由 MFC Application Wizard 产生的主框架窗口包含了菜单项、工具栏、状态栏和一个用户区。视窗被作为子窗口附着在框架窗口上，覆盖了框架窗口的用户区。用户主要在用户区中工作，但是用户可以选择框架窗口的菜单命令对视窗产生影响。

当应用程序初始化主框架窗口时，会向主框架窗口发送一个 WM_CREATE 消息，要求创建主框架窗口。然后主框架窗口调用 OnCreate 函数来处理该消息。OnCreate 函数主要对主框架窗口进行一些初始化。在调用了该函数后，系统再调用 ShowWindow 函数，

就可以将主框架窗口显示在屏幕上。值得指出的是，本函数必须在主框架窗口对象创建以后才能使用。

下面列出了 MFC Application Wizard 缺省产生的 OnCreate 函数：

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    if (!m_wndToolBar.CreateEx(this,TBSTYLE_FLAT,WS_CHILD | WS_VISIBLE
|CBRS_TOP|CBRS_GRIPPER|CBRS_TOOLTIPS|CBRS_FLYBY|
CBRS_SIZE_DYNAMIC)||!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }
    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
    return 0;
}
```

可以看出，在函数中，创建了工具栏、状态条。框架窗口负责为程序处理菜单、状态条和工具栏。而这些菜单命令都已经被链接到了对应的函数上，因此，用户不用再去处理各个菜单，只要将精力集中在文档和视上就可以了。

我们这里介绍的是 SDI 的框架窗口，将来我们还会讲到 MDI 的框架窗口，它和 SDI 的框架窗口有较大的不同。

1.1.4 文档模板

文档模板被用于指明程序的一个新的实例、一个框架窗口、一个文档，以及一个视之间的某种联系。它通常在 InitInstance 函数中被创建，然后由它来创建文档对象和主框架窗口。

在用 MFC Application Wizard 产生的 SDI 程序的代码中，已经创建了一个单文档模板并将它加入到应用程序中，如下所示：

```
CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CSingleDoc1Doc),
        RUNTIME_CLASS(CMainFrame),          // main SDI frame window
        RUNTIME_CLASS(CSingleDoc1View));
    AddDocTemplate(pDocTemplate);
```

我们对该段代码解释如下：

- 声明了一个指向 CSingleDocTemplate 的指针，使用 new 在堆中产生模块对象。只要应用程序存在模块对象，即使在调用 AddDocTemplate 后不直接使用指针，也应保留在内存中。
- 用四个参数调用构造函数，包括资源 ID、文档、框架和视类的类信息结构，后三个参数由 RUNTIME_CLASS 宏提供。
- 调用 AddDocTemplate，用 MFC 指定模板，并将该模板放入一个由系统保持的可用文件模板列表中。

构造函数的声明如下：

```
CSingleDocTemplate( UINT nIDResource, CRuntimeClass* pDocClass,
CRuntimeClass* pFrameClass, CRuntimeClass* pViewClass );
```

其中：

- nIDResource：用于该文档类型的资源标识符。
- pDocClass：指向文档类的指针。
- pFrameClass：框架类的指针。
- pViewClass：视类的指针。

函数动态地生成一个文档模板的对象。

在加入了单文档模板对象后，我们就在 CSingleDoc1Doc、CMainFrame、CSingleDoc1View 之间建立起了某种联系。应用程序将使用由 MFC 为产生新文档和视提供的缺省功能。这些工作不必过问，只要让 MFC 去做就行了。

1.1.5 文档类

文档就是指一组数据的集合。用户常常使用 File 菜单中的 Open 命令来打开它，用 File 菜单中的 Save 命令来保存它。在 MFC 中，由文档类 CDocument 来实现文档操作。派生文档类一般用来完成对文档对象数据的读和写工作。

MFC Application Wizard 为本项目生成了一个文档类 CSingleDoc1Doc，它派生于 MFC 的文档类 CDocument。它在 SingleDoc1Doc.h 中被如下声明：

```
// SingleDoc1Doc.h : interface of the CSingleDoc1Doc class
//
/////////////////////////////////////////////////////////////////////////////
class CSingleDoc1Doc : public CDocument
{
protected: // create from serialization only
    CSingleDoc1Doc();
    DECLARE_DYNCREATE(CSingleDoc1Doc)
// Attributes
public:
// Operations
public:
// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CSingleDoc1Doc)
public:
virtual BOOL OnNewDocument();
virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL
// Implementation
public:
virtual ~CSingleDoc1Doc();
#ifdef _DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
protected:
    CString m_string;
```

```
// Generated message map functions
protected:
//{{AFX_MSG(CSingleDoc1Doc)
// NOTE - the ClassWizard will add and remove member functions here.
// DO NOT EDIT what you see in these blocks of generated code !
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};
```

现在它只包含了一个类定义的基本框架。注意宏 DECLARE_DYNCREATE 的包含，它使得该类可以动态地生成。要在文档中存储数据，只需要简单地向类的定义中加入数据成员即可。

1.1.6 视类

视就是附属于某个文档的一个窗口，它扮演着用户和文档之间交换信息媒体的角色。视在屏幕上显示一个文档，并且解释用户的输入来对文档进行相应的操作。视也显示图像以用于打印和打印预览。

视在 MFC 中由 CView 类来支持。该类通常表示一个依附于框架窗口并占据它的用户区的子窗口。根据自己的需要，我们也可以选择一些由 CView 类派生出来的类作为基类，它们都有着自己的特性。例如，使用 CScrollView 类可以使视支持滚动条，使用 CFormView 支持表格风格，使用 CEditView 能够方便地编辑文本。

MFC Application Wizard 为本项目生成了一个文档类 CSingleDoc1Doc，它派生于 MFC 的文档类 CDocument。它在 SingleDoc1View.h 中如下声明：

```
// SingleDoc1View.h : interface of the CSingleDoc1View class
//
/////////////////////////////////////////////////////////////////////////////
class CSingleDoc1View : public CView
{
protected: // create from serialization only
    CSingleDoc1View();
    DECLARE_DYNCREATE(CSingleDoc1View)
// Attributes
public:
    CSingleDoc1Doc* GetDocument();
// Operations
public:
```

```
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSingleDoc1View)
        public:
        virtual void OnDraw(CDC* pDC);   // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CSingleDoc1View();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
        //{{AFX_MSG(CSingleDoc1View)
        afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
#ifndef _DEBUG  // debug version in SingleDoc1View.cpp
inline CSingleDoc1Doc* CSingleDoc1View::GetDocument()
 { return (CSingleDoc1Doc*)m_pDocument; }
#endif
```

现在它只包含了一个类定义的基本框架。我们可以看到，其中定义了两个很重要的函数：OnDraw 和 GetDocument。前者用于重绘窗口，后者用于获得指向视对应的文档的指针。我们将在后面详细介绍这两个函数。我们将来要做的工作主要是向视中加入各种有用的函数。