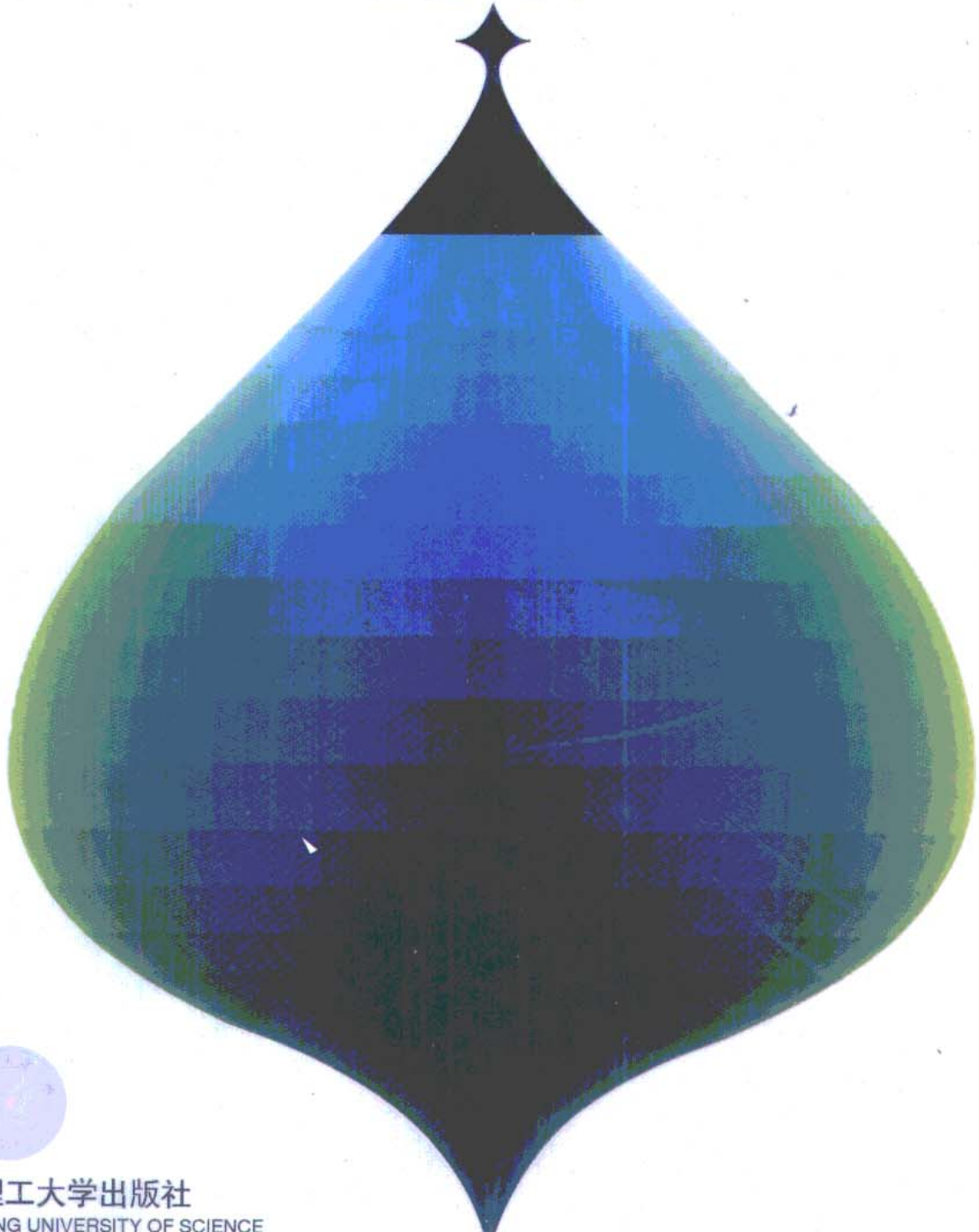
计算机程序设计与软件开发系列丛书

MFC深入浅出

——从MFC设计到MFC编程

李久进 编著



理工大学出版社
HUAZHONG UNIVERSITY OF SCIENCE

计算机程序设计与软件开发系列丛书

MFC 深入浅出

——从 MFC 设计到 MFC 编程

李久进　编著

华中理工大学出版社

前　言

现在流行的 Windows 下的编程语言实在不少，所以在 BBS 上常常有人会问：我应该使用什么编程语言呢？其中，有一个大家认可的答案：真正的程序员使用 Visual C++。

的确，Visual C++是一个功能强大、灵活、方便的编程工具，可以完成其他编程语言所无法完成的任务，可以让程序员方便地实现自己的设计，尽情地发挥自己的创造性。

Visual C++的强大无比的功能除了得益于 C++的特性之外，更重要的是由于它具有体系完整、机制灵活、功能丰富的 MFC 类库。

所以，要讲 Visual C++，必须讲 MFC 类库。

MFC 类库可以分两个层次，首先是实现 MFC 编程框架体系的核心 MFC 类库，然后是建立在核心 MFC 类库基础之上的扩展类库，例如，支持 COM 的类库，实现网络功能的类库，等等。随着 Visual C++的不断升级，MFC 类库的功能越来越丰富，越来越强大，但是，MFC 核心类库是相对稳定的，特别是从 Visual C++ 4.2 开始到现在的 Visual C++ 6.0。

本书的中心就是深入浅出地解析 MFC 类库，分析怎么使用 MFC 类库以及 MFC 类库的内部实现，揭开 MFC 复杂、深奥的面纱，让读者对 MFC 有一个全面、透彻、清晰的理解。关于 MFC 的核心实现，主要有以下几个方面。

首先，MFC 采用 C++的面向对象的特征封装了 Windows 的对象和 Win32 函数，一定程度上隐蔽了底层 Win32 的复杂性。

其次，MFC 采用消息映射的方法来处理 Windows 消息和事件，隐藏了 Windows 窗口的窗口过程，简化了消息处理的复杂性和烦琐性。

还有，MFC 提供了一个以文档-视为中心的编程模式，并实现了以文档-视为中心的编程框架，简化了数据处理的过程。

而且，MFC 提供了模块状态、线程状态、模块线程状态来支持多线程的编程设计和 DLL 的编程。

本书分别从使用 MFC 的角度和 MFC 内部设计及实现的角度讨论了上述内容，分析了 MFC 核心的设计和实现；然后，在此基础上，进一步讨论了 MFC 对一些常用类的实现。有关章节的内容如下：

第一章，MFC 概述。

第二章，解释 MFC 对 Win32 API 和 Windows 对象的封装，讨论各类 MFC 对象的使用，分析 MFC 对象和 Windows 对象的关系。

第三章，讨论 CObject 的特性及其实现，包括动态类信息、动态创建、序列化的实现等内容。

第四章，讨论 MFC 的消息映射机制，分析 MFC 对各类消息的处理，例如对 Windows 消息、控制通知消息、命令消息、状态更新消息、反射消息等的处理；并揭示了 MFC 通过消息映射手段实现 C++虚拟函数机制的原理。

第五章和第六章，分析 MFC 编程框架启动和关闭一个应用程序的过程，揭示 MFC 框架的内幕，剖析以文档模板为核心创建基于文档-视的应用程序的过程，展示 MFC 框架处理消息和调用虚拟函数的时机和位置。

第七、八、九章，介绍 MFC 的动态链接库、进程、线程等概念，以及 MFC 动态链接库的种类和使用，讨论 MFC 下多线程编程的问题。并且进一步阐述 MFC 的核心概念之一：状态（模块状态、线程状态、模块线程状态），揭示 MFC 对多线程的支持机制，MFC 实现规则 DLL 和扩展 DLL 的内幕。

第十章，阐述 MFC 下的调试手段。

第十一章，讨论 CFile 类，主要分析了 CFile 的使用和它对 Win32 文件函数的封装。

第十二章，讨论模式和无模式对话框，分析 MFC 如何设计和实现这两种对话框的功能，分析 CDialog 和 CFormView 为实现有关功能而设计的虚拟函数、消息处理函数等。

第十三章，讨论 MFC 工具栏和状态栏的设计及其实现，分析 MFC 是如何以 CControlBar 为基础，派生出 CStatusBar、CToolBar、CDialogBar 等子类，实现 MFC 工具栏和状态栏标准处理。

第十四章，讨论 MFC 的 Socket 类。

第一章到第十章介绍了 MFC 的核心概念以及实现。在此基础上，第十一章到第十四章讨论了 MFC 一些常用类的实现。

本书的内容对 MFC 的初学者（最好对 Visual C++和 Windows 有所了解）和提高者都是很有帮助的。

如果您是一个初学者，可以读第一至第六章。主要目的是建立对 MFC 的全面理解，了解 MFC 框架是如何支持程序员编程的。如果有读不懂的地方，可以跳过，直接阅读有关分析的结论。特别是第五和第六章，可以重点阅读，了解 MFC 是怎样来处理有关消息、调用有关虚拟函数的。

然后，还可以读第十章，第十一至第十四章。特别第十二章，可以重点阅读，它是 MFC 从 CWnd 或者 CView 派生出特定的类实现特定功能的例子，可以帮助您进一步理解 MFC，并且学习如何设计和实现一个特定的类。

如果您对 MFC 有一定的掌握，可以进一步阅读第八和第九章，了解 MFC 处理 DLL 和线程的知识。对于第一至第六章、第十至第十四章，应该把重点放在 MFC 的设计和实现的分析上。这样，可以深化您对 MFC 和 Windows 编程的理解与掌握。

如果您可以较熟练地使用 MFC，建议您进一步阅读第九章，并且对所有有关章节的设计和实现分析作重点阅读，这样，不仅可以帮助您深入地理解和掌握 MFC，而且，从 MFC 的有关内部设计和实现上，必然可以提高您的程序设计和编写能力。

由于成书仓促，书中可能存在一些缺点和错误，恳请您不吝赐教！作者的电子邮箱：ljjin@public.szonline.net。

编者

1999 年 6 月

MFC 概述

1.1 MFC 是一个编程框架

MFC(Microsoft Foundation Class Library)中的各种类结合起来构成了一个应用程序框架，它的目的就是让程序员在此基础上来建立 Windows 下的应用程序，这是一种相对 SDK 来说更为简单的方法。因为总体上，MFC 框架定义了应用程序的轮廓，并提供了用户接口的标准实现方法，程序员所要做的就是通过预定义的接口把具体应用程序特有的东西填入这个轮廓。Microsoft Visual C++提供了相应的工具来完成这个工作：AppWizard 可以用来生成初步的框架文件(代码和资源等)；资源编辑器用来帮助直观地设计用户接口；ClassWizard 用来协助添加代码到框架文件；编译则通过类库实现应用程序特定的逻辑。

1.1.1 封装

构成 MFC 框架的是 MFC 类库，MFC 类库是 C++类库。这些类或者封装了 Win32 应用程序编程接口，或者封装了应用程序的概念，或者封装了 OLE 特性，或者封装了 ODBC 和 DAO 数据访问的功能，等等，分述如下。

(1) 对 Win32 应用程序编程接口的封装

用一个 C++ Object 来包装一个 Windows Object。例如：class CWnd 是一个 C++ window object，它把 Windows window(HWND)和 Windows window 有关的 API 函数封装在 C++ window object 的成员函数内，后者的成员变量 m_hWnd 就是前者的窗口句柄。

(2) 对应用程序概念的封装

使用 SDK 编写 Windows 应用程序时，总要定义窗口过程，登记 Windows Class，创建窗口，等等。MFC 把许多类似的处理封装起来，替程序员完成这些工作。另外，MFC 提供了以文档–视图为中心的编程模式，MFC 类库封装了对它的支持。文档是用户操作的数据对象，视图是数据操作的窗口，用户通过它处理、查看数据。

(3) 对 COM/OLE 特性的封装

OLE 建立在 COM 模型之上，由于支持 OLE 的应用程序必须实现一系列接口(Interface)，因而相当繁琐。MFC 的 OLE 类封装了 OLE API 大量的复杂工作，这些类提供了实现 OLE 的更高级接口。

(4) 对 ODBC 功能的封装

以少量的能提供与 ODBC 之间更高级接口的 C++类，封装了 ODBC API 的大量的复

杂的工作，提供了一种数据库编程模式。

1.1.2 继承

首先，MFC 抽象出众多类的共同特性，设计出一些基类作为实现其他类的基础。这些类中，最重要的类是 CObject 和 CCmdTarget。CObject 是 MFC 的根类，绝大多数 MFC 类是其派生的，包括 CCmdTarget。CObject 实现了一些重要的特性，包括动态类信息、动态创建、对象序列化、对程序调试的支持，等等。所有从 CObject 派生的类都将具备或者可以具备 CObject 所拥有的特性。CCmdTarget 通过封装一些属性和方法，提供了消息处理的架构。在 MFC 中，任何可以处理消息的类都从 CCmdTarget 派生。

针对每种不同的对象，MFC 都设计了一组类对这些对象进行封装，每一组类都有一个基类，从基类派生出众多更具体的类。这些对象包括以下种类：窗口对象，基类是 CWnd；应用程序对象，基类是 CWinThread；文档对象，基类是 CDocument，等等。

程序员将结合自己的实际，从适当的 MFC 类中派生出自己的类，实现特定的功能，达到自己的编程目的。

1.1.3 虚拟函数和动态约束

MFC 以“C++”为基础，自然支持虚拟函数和动态约束。但是作为一个编程框架，有一个问题必须解决：如果仅仅通过虚拟函数来支持动态约束，必然导致虚拟函数表过于臃肿，消耗内存，效率低下。例如，CWnd 封装 Windows 窗口对象时，每一条 Windows 消息对应一个成员函数，这些成员函数为派生类所继承。如果这些函数都设计成虚拟函数，由于数量太多，实现起来不现实。于是，MFC 建立了消息映射机制，以一种富有效率、便于使用的手段解决消息处理函数的动态约束问题。

这样，通过虚拟函数和消息映射，MFC 类提供了丰富的编程接口。程序员在继承基类的同时，把自己实现的虚拟函数和消息处理函数嵌入 MFC 的编程框架。MFC 编程框架将在适当的时候、适当的地方来调用程序的代码。本书将充分地展示 MFC 调用虚拟函数和消息处理函数的内幕，让读者对 MFC 的编程接口有清晰的理解。

1.1.4 MFC 的宏观框架体系

如前所述，MFC 实现了对应用程序概念的封装，把类、类的继承、动态约束、类的关系和相互作用等封装起来。这样封装的结果对程序员来说，是一套开发模板(或者说模式)。针对不同的应用和目的，程序员采用不同的模板。例如，SDI 应用程序的模板，MDI 应用程序的模板，规则 DLL 应用程序的模板，扩展 DLL 应用程序的模板，OLE/ActiveX 应用程序的模板，等等。

这些模板都采用了以文档-视为中心的思想，每一个模板都包含一组特定的类。典型的 MDI 应用程序的构成将在下一节具体讨论。

为了支持对应用程序概念的封装，MFC 内部必须作大量的工作。例如，为了实现消息映射机制，MFC 编程框架必须要保证首先得到消息，然后按既定的方法进行处理。又如，

为了实现对 DLL 编程的支持和多线程编程的支持，MFC 内部使用了特别的处理方法，使用模块状态、线程状态等来管理一些重要信息。虽然，这些内部处理对程序员来说是透明的，但是，懂得和理解 MFC 内部机制有助于写出功能灵活而强大的程序。

总之，MFC 封装了 Win32 API、OLE API、ODBC API 等底层函数的功能，并提供更高一层的接口，简化了 Windows 编程。同时，MFC 支持对底层 API 的直接调用。

MFC 提供了一个 Windows 应用程序开发模式，对程序的控制主要是由 MFC 框架完成的，而且 MFC 也完成了大部分的功能，预定义或实现了许多事件和消息处理，等等。“框架”或者由其本身处理事件，不依赖程序员的代码；或者调用程序员的代码来处理应用程序特定的事件。

MFC 是 C++类库，程序员就是通过使用、继承和扩展适当的类来实现特定的目的。例如，继承时，应用程序特定的事件由程序员的派生类来处理，不感兴趣的由基类处理。实现这种功能的基础是 C++对继承的支持，对虚拟函数的支持，以及 MFC 实现的消息映射机制。

1.2　MDI 应用程序的构成

本节解释一个典型的 MDI 应用程序的构成。

用 AppWizard 产生一个 MDI 工程 t (无 OLE 等支持)，AppWizard 创建了一系列文件，构成了一个应用程序框架。这些文件分四类：头文件(.h)，实现文件(.cpp)，资源文件(.rc)，模块定义文件(.def)等。

1.2.1　构成应用程序的对象

图 1-1 解释了该应用程序的结构，箭头表示信息流向。

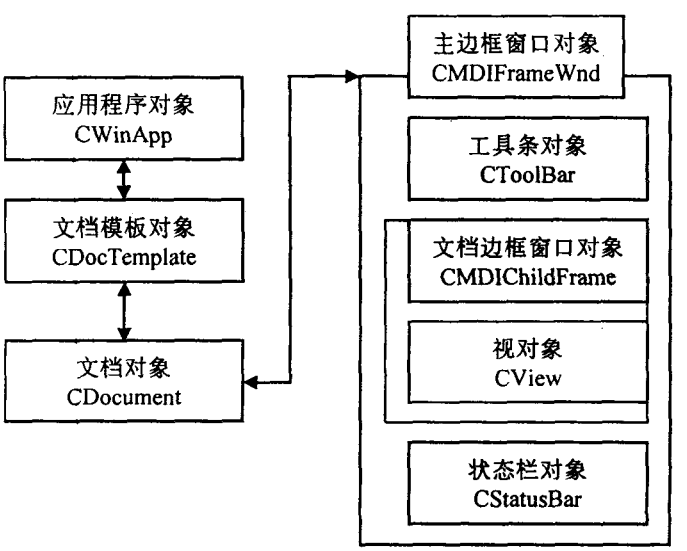


图 1-1　一个 MDI 应用程序的构成

从 CWinApp、CDocument、CView、CMDIFrameWnd、CMDIChildWnd 类对应地派生

出 CTApp、CTDoc、CTView、CMainFrame、CChildFrame 五个类，这五个类的实例分别是应用程序对象、文档对象、视对象、主框架窗口对象和文档边框窗口对象。主框架窗口包含了视窗口、工具条和状态栏。对这些类或者对象解释如下。

(1) 应用程序

应用程序类派生于 CWinApp。基于框架的应用程序必须有且只有一个应用程序对象，它负责应用程序的初始化、运行和结束。

(2) 边框窗口

如果是 SDI 应用程序，从 CFrameWnd 类派生边框窗口类，边框窗口的客户子窗口(MDIClient)直接包含视窗口；如果是 MDI 应用程序，从 CMDIFrameWnd 类派生边框窗口类，边框窗口的客户子窗口(MDIClient)直接包含文档边框窗口。

如果要支持工具条、状态栏，则派生的边框窗口类还要添加 CToolBar 和 CStatusBar 类型的成员变量，以及在一个 OnCreate 消息处理函数中初始化这两个控制窗口。

边框窗口用来管理文档边框窗口、视窗口、工具条、菜单、加速键等，协调半模式状态(如上下文的帮助(Shift+F1 模式)和打印预览)。

(3) 文档边框窗口

文档边框窗口类从 CMDIChildWnd 类派生，MDI 应用程序使用文档边框窗口来包含视窗口。

(4) 文档

文档类从 CDocument 类派生，用来管理数据，数据的变化、存取都是通过文档实现的。视窗口通过文档对象来访问和更新数据。

(5) 视

视类从 CView 或它的派生类派生。视和文档联系在一起，在文档和用户之间起中介作用，即视在屏幕上显示文档的内容，并把用户输入转换成对文档的操作。

(6) 文档模板

文档模板类一般不需要派生。MDI 应用程序使用多文档模板类 CMultiDocTemplate；SDI 应用程序使用单文档模板类 CSingleDocTemplate。

应用程序通过文档模板类对象来管理上述对象(应用程序对象、文档对象、主边框窗口对象、文档边框窗口对象、视对象)的创建。

1.2.2 构成应用程序的对象之间的关系

这里，用图的形式可直观地表示所涉及的 MFC 类的继承或者派生关系，如图 1-2 所示。

图 1-2 所示的类都是从 CObject 类派生出来的；所有处理消息的类都是从 CCmdTarget 类派生的。如果是多文档应用程序，文档模板使用 CMultiDocTemplate，主框架窗口从 CMDIFarmeWnd 派生，它包含工具条、状态栏和文档框架窗口。文档框架窗口从 CMDIChildWnd 派生，文档框架窗口包含视，视从 CView 或其派生类派生。

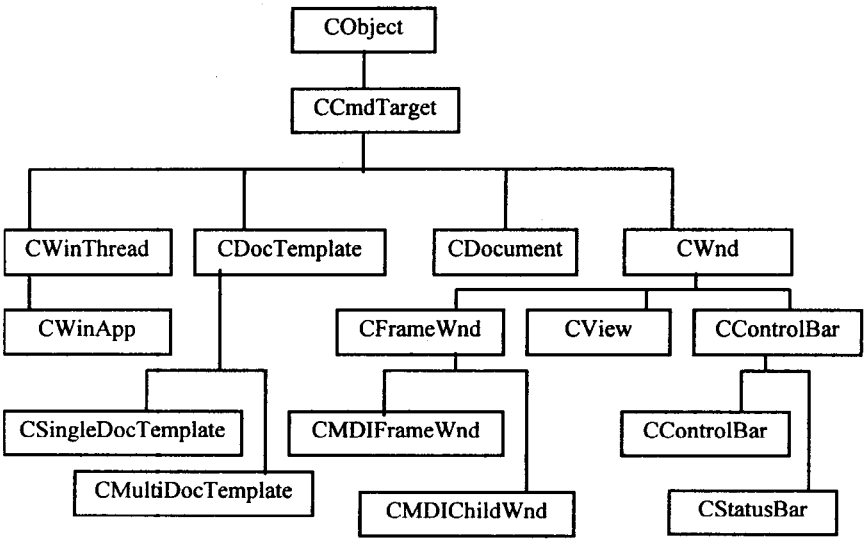


图 1-2　一些 MFC 类的层次

1.2.3　构成应用程序的文件

通过上述分析，可知 AppWizard 产生的 MDI 框架程序的内容、所定义和实现的类。下面，从文件的角度来考察 AppWizard 生成了哪些源码文件，这些文件的作用是什么。表 1-1 列出了 AppWizard 所生成的头文件，表 1-2 列出了 AppWizard 所生成的实现文件及其对头文件的包含关系。

表1-1　AppWizard所生成的头文件

头文件	用　途
stdafx.h	标准 AFX 头文件
resource.h	定义了各种资源 ID
t.h	#include "resource.h" 定义了从 CWinApp 派生的应用程序对象 CTApp
childfrm.h	定义了从 CMDIChildWnd 派生的文档框架窗口对象 CTChildFrame
mainfrm.h	定义了从 CMDIFrameWnd 派生的框架窗口对象 CMainFrame
tdoc.h	定义了从 CDocument 派生的文档对象 CTDoc
tview.h	定义了从 CView 派生的视图对象 CTView

从表 1-2 中的包含关系一栏可以看出：

CTApp 的实现用到所有的用户定义对象，包含了他们的定义；CView 的实现用到 CTDoc；其他对象的实现只涉及自己的定义。

当然，如果增加其他操作，引用其他对象，则要包含相应的类的定义文件。

对预编译头文件说明如下：

表1-2 AppWizard所生成的实现文件

实现文件	所包含的头文件	实现的内容和功能
stdafx.cpp	#include "stdafx.h"	用来产生预编译的类型信息
实现文件	所包含的头文件	实现的内容和功能
t.cpp	# include "stdafx.h" # include "t.h" # include "mainfrm.h" # include "childfrm.h" #include "tdoc.h" #include "tview.h"	定义 CTApp 的实现，并定义 CTApp 类型的全局变量 theApp
childfrm.cpp	#include "stdafx.h" #include "t.h" #include "childfrm.h"	实现了类 CChildFrame
childfrm.cpp	#include "stdafx.h" #include "t.h" #include "childfrm.h"	实现了类 CMainFrame
tdoc.cpp	# include "stdafx.h" # include "t.h" # include "tdoc.h"	实现了类 CTDoc
tview.cpp	# include "stdafx.h" # include "t.h" # include "tdoc.h" # include "tview.h"	实现了类 CTView

所谓头文件预编译，就是把一个工程(Project)中使用的一些 MFC 标准头文件(如 Windows.h、Afxwin.h)预先编译，以后该工程编译时，不再编译这部分头文件，仅仅使用预编译的结果．这样可以加快编译速度，节省时间．

预编译头文件通过编译 stdafx.cpp 生成，以工程名命名，由于预编译的头文件的后缀是“pch”，所以编译结果文件是 projectname.pch．

编译器通过一个头文件 stdafx.h 来使用预编译头文件．stdafx.h 这个头文件名是可以在 project 的编译设置里指定的．编译器认为,所有在指令#include "stdafx.h"前的代码都是预编译的，它跳过#include "stdafx. h"指令,使用 projectname.pch 编译这条指令之后的所有代码．

因此，所有的 CPP 实现文件第一条语句都是：#include "stdafx.h"．另外，每一个实现文件 CPP 都包含了如下语句：

```
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = _FILE_;
#endif
```

这是表示，如果生成调试版本，要指示当前文件的名称．_FILE_是一个宏，在编译器编译过程中给它赋值为当前正在编译的文件名称．

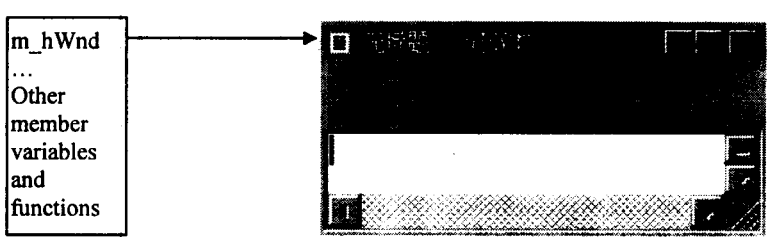
MFC 和 Win32

2.1 MFC Object 和 Windows Object 的关系

MFC 中最重要的封装是对 Win32 API 的封装，因此，理解 Windows Object 和 MFC Object (C++对象，一个 C++类的实例)之间的关系是理解 MFC 的关键之一．所谓 Windows Object(Windows 对象)是 Win32 下用句柄表示的 Windows 操作系统对象；所谓 MFC Object (MFC 对象)是 C++对象，是一个 C++类的实例，这里(本书范围内)MFC Object 是有特定含义的，指封装 Windows Object 的 C++ Object，并非指任意的 C++ Object．

MFC Object 和 Windows Object 是不一样的，但两者紧密联系．以窗口对象为例：

一个 MFC 窗口对象是一个 C++ CWnd 类(或派生类)的实例，是程序直接创建的．在程序执行中它随着窗口类构造函数的调用而生成，随着析构函数的调用而消失．而 Windows 窗口则是 Windows 系统的一个内部数据结构的实例，由一个“窗口句柄”标识，Windows 系统创建它并给它分配系统资源．Windows 窗口在 MFC 窗口对象创建之后，由 CWnd 类的 Create 成员函数创建，“窗口句柄”保存在窗口对象的 m_hWnd 成员变量中．Windows 窗口可以被一个程序销毁，也可以被用户的动作销毁．MFC 窗口对象和 Windows 窗口对象的关系如图 2-1 所示．其他的 Windows Object 和对应的 MFC Object 也有类似的关系．



MFC 窗口对象(CWnd)　　Windows 窗口

图 2-1　MFC 窗口对象和 Windows 窗口对象的关系

下面，对 MFC Object 和 Windows Object 作一个比较．有些论断对设备描述表(MFC 类是 CDC，句柄是 HDC)可能不适用，但具体涉及到时会指出．

(1) 从数据结构上比较

MFC Object 是相应的 C++类的实例，这些类是 MFC 或者程序员定义的；Windows Object 是 Windows 系统的内部结构，通过一个句柄来引用．

MFC 给这些类定义了一个成员变量来保存 MFC Object 对应的 Windows Object 的句柄。对于设备描述表 CDC 类，将保存两个 HDC 句柄。

(2) 从层次上比较

MFC Object 是高层的，Windows Object 是低层的；MFC Object 封装了 Windows Object 的大部分或全部功能，MFC Object 的使用者不需要直接应用 Windows Object 的 Handle(句柄)使用 Win32 API，代替它的是引用相应的 MFC Object 的成员函数。

(3) 从创建上比较

MFC Object 通过构造函数由程序直接创建；Windows Object 由相应的 SDK 函数创建。在 MFC 中，使用这些 MFC Object，一般分两步：

首先，创建一个 MFC Object，或者在 STACK 中创建，或者在 HEAP 中创建，这时，MFC Object 的句柄实例变量为空，或者说不是一个有效的句柄。

然后，调用 MFC Object 的成员函数创建相应的 Windows Object，MFC 的句柄变量存储一个有效句柄。

CDC(设备描述表类)的创建有所不同，在 2.3 节会具体说明 CDC 及其派生类的创建和使用。

当然，可以在 MFC Object 的构造函数中创建相应的 Windows 对象，MFC 的 GDI 类就是如此实现的，但从实质上讲，MFC Object 的创建和 Windows Object 的创建是两回事。

(4) 从转换上比较

可以从一个 MFC Object 得到对应的 Windows Object 的句柄；一般使用 MFC Object 的成员函数 GetSafeHandle 得到对应的句柄。

可以从一个已存在的 Windows Object 创建一个对应的 MFC Object；一般使用 MFC Object 的成员函数 Attach 或者 FromHandle 来创建，前者得到一个永久性对象，后者得到的可能是一个临时对象。

(5) 从使用范围上比较

MFC Object 对系统的其他进程来说是不可见、不可用的；而 Windows Object 一旦创建，其句柄是整个 Windows 系统全局的。一些句柄可以被其他进程使用。典型地，一个进程可以获得另一进程的窗口句柄，并给该窗口发送消息。

对同一个进程的线程来说，只可以使用本线程创建的 MFC Object，不能使用其他线程的 MFC Object。

(6) 从销毁上比较

MFC Object 随着析构函数的调用而消失；但 Windows Object 必须由相应的 Windows 系统函数销毁。

设备描述表 CDC 类的对象有所不同，它对应的 HDC 句柄对象可能不是被销毁，而是被释放。

当然，可以在 MFC Object 的析构函数中完成 Windows Object 的销毁，MFC Object 的 GDI 类等就是如此实现的，但是，应该看到：两者的销毁是不同的。

每类 Windows Object 都有对应的 MFC Object，下面用表格的形式列出它们之间的对应关系，如表 2-1 所示。

表 2-1 中的 Object 分以下几类：Windows 对象，设备上下文对象，GDI 对象(BITMAP，

BRUSH，FONT，PALETTE，PEN，RGN)，菜单，图像列表，网络套接字接口。

表 2-1　MFC Object 和 Windows Object 的对应关系

描　　述	Windows 句柄	MFC Object
窗口	HWND	CWnd and CWnd-derived classes
设备上下文	HDC	CDC and CDC-derived classes
菜单	HMENU	CMenu
笔	HPEN	CGdiObject 类，CPen 和 CPen-derived classes
刷子	HBRUSH	CGdiObject 类，CBrush 和 CBrush-derived classes
字体	HFONT	CGdiObject 类，CFont 和 CFont-derived classes
位图	HBITMAP	CGdiObject 类，CBitmap 和 CBitmap-derived classes
调色板	HPALETTE	CGdiObject 类，CPalette 和 CPalette-derived classes
区域	HRGN	CGdiObject 类，CRgn 和 CRgn-derived classes
图像列表	HIMAGELIST	CImageList 和 CImageList-derived classes
套接字	HSOCKET	CSocket,CAsynSocket 及其派生类

从广义上来看，文档对象和文件可以看作一对 MFC Object 和 Windows Object 分别用 CDocument 类和文件句柄描述。

后续几节分别对前四类作一个简明扼要的论述。

2.2　Windows Object

用 SDK 的 Win32 API 编写各种 Windows 应用程序，有其共同的规律：首先是编写 WinMain 函数，编写处理消息和事件的窗口过程 WndProc，在 WinMain 里头注册窗口 (Register Window)，创建窗口，然后开始应用程序的消息循环。

MFC 应用程序也不例外，因为 MFC 是一个建立在 SDK API 基础上的编程框架。对程序员来说所不同的是：一般情况下，MFC 框架自动完成了 Windows 登记、创建等工作。下面，简要介绍 MFC Window 对 Windows Window 的封装。

2.2.1　Windows 的注册

一个应用程序在创建某个类型的窗口前，必须首先注册该“窗口类”(Windows Class)。注意，这里不是 C++类的类。Register Window 把窗口过程、窗口类型，以及其他类型信息和要登记的窗口类关联起来。

(1) “窗口类”的数据结构

“窗口类”是 Windows 系统的数据结构，可以把它理解为 Windows 系统的类型定义，而 Windows 窗口则是相应“窗口类”的实例。Windows 使用一个结构来描述“窗口类”，

其定义如下：

```
typedef struct _WNDCLASSEX {
    UINT      cbSize;  //该结构的字节数
    UINT      style;   //窗口类的风格
    WNDPROC lpfnWndProc;  //窗口过程
    int       cbClsExtra;
    int       cbWndExtra;
    HANDLE    hInstance;  //该窗口类的窗口过程所属的应用实例
    HICON     hIcon;  //该窗口类所用的像标
    HCURSOR hCursor;  //该窗口类所用的光标
    HBRUSH    hbrBackground;  //该窗口类所用的背景刷
    LPCTSTR lpszMenuName;  //该窗口类所用的菜单资源
    LPCTSTR lpszClassName;  //该窗口类的名称
    HICON     hIconSm;  //该窗口类所用的小像标
} WNDCLASSEX;
```

从“窗口类”的定义可以看出，它包含了一个窗口的重要信息，如窗口风格、窗口过程、显示和绘制窗口所需要的信息，等等。关于窗口过程，将在后面消息映射等有关章节作详细论述。

Windows 系统在初始化时，会注册(Register)一些全局的“窗口类”，例如通用控制窗口类。应用程序在创建自己的窗口时，首先必须注册自己的窗口类。在 MFC 环境下，有几种方法可以用来注册“窗口类”，下面分别予以讨论。

(2) 调用 AfxRegisterClass 注册

AfxRegisterClass 函数是 MFC 全局函数。AfxRegisterClass 的函数原型：

```
BOOL AFXAPI AfxRegisterClass(WNDCLASS *lpWndClass);
```

参数 lpWndClass 是指向 WNDCLASS 结构的指针，表示一个“窗口类”。

首先，AfxRegisterClass 检查希望注册的“窗口类”是否已经注册，如果是，则表示已注册，返回 TRUE，否则，继续处理。

接着，调用::RegisterClass(lpWndClass)注册窗口类。

然后，如果当前模块是 DLL 模块，则把注册“窗口类”的名字加入到模块状态的域 m_szUnregisterList 中。该域是一个固定长度的缓冲区，依次存放模块注册的“窗口类”的名字(每个名字是以“\n\0”结尾的字符串)。之所以这样做，是为了 DLL 退出时能自动取消(Unregister)它注册的窗口类。至于模块状态将在第 9 章详细讨论。

最后，返回 TRUE 表示成功注册。

(3) 调用 AfxRegisterWndClass 注册

AfxRegisterWndClass 函数也是 MFC 的全局函数。AfxRegisterWndClass 的函数原型：

```
LPCTSTR AFXAPI AfxRegisterWndClass(UINT nClassStyle,
    HCURSOR hCursor, HBRUSH hbrBackground, HICON hIcon)
```

参数 1 指定“窗口类”风格；参数 2、3、4 分别指定该窗口类使用的光标、背景刷、

像标的句柄，缺省值是 0。

此函数根据窗口类属性动态地产生窗口类的名字，然后，判断是否该类已经注册，是则返回窗口类名；否则用指定窗口类的属性(窗口过程指定为缺省窗口过程)，调用 AfxRegisterCalss 注册窗口类，返回类名。

动态产生的窗口类名字由以下几部分组成(包括冒号分隔符)。

如果参数 2、3、4 全部为 NULL，则由三部分组成：

“Afx” + “:” + “模块实例句柄” + “:” + “窗口类风格”，

否则，由六部分组成：

“Afx” + “:” + “模块实例句柄” + “:” + “窗口类风格” + “:” + “光标句柄” + “:” + “背景刷句柄” + “:” + “像标句柄”。比如：“Afx:400000:b:13de:6:32cf”。

该函数在 MFC 注册主边框或者文档边框“窗口类”时被调用。具体怎样用 5.3.3 节会指出。

(4) 隐含地使用 MFC 预定义的窗口类

MFC 4.0 以前的版本提供了一些预定义的窗口类，4.0 以后不再预定义这些窗口类。但是，MFC 仍然沿用了这些窗口类，例如：

用于子窗口的“AfxWnd”；

用于边框窗口(SDI 主窗口或 MDI 子窗口)或视的“AfxFrameOrView”；

用于 MDI 主窗口的“AfxMDIFrame”；

用于标准控制条的“AfxControlBar”。

这些类的名字就是“AfxWnd”、“AfxFrameOrView”、“AfxMDIFrame”、“AfxControlBar”加上前缀和后缀(用来标识版本号或是否调试版等)。它们使用标准应用程序像标、标准文档像标、标准光标等标准资源。为了使用这些“窗口类”，MFC 会在适当的时候注册这些类：这些类或者在创建该类的窗口时注册，或者在创建应用程序的主窗口时注册，等等。

MFC 内部使用了函数

BOOL AFXAPI AfxEndDeferRegisterClass(short fClass)

来帮助注册上述原 MFC 版本的预定义“窗口类”。参数 fClass 区分了那些预定义窗口的类型。根据不同的类型，使用不同的窗口类风格、窗口类名字等填充 WndClass 的域，然后调用 AfxRegisterClass 注册窗口类。并且注册成功之后，通过模块状态的 m_fRegisteredClasses 记录该窗口类已经注册，这样该模块在再次需要注册这些窗口类之前可以查一下 m_fRegisteredClasses，如果已经注册就不必浪费时间了。为此，MFC 内部使用宏

AfxDeferRegisterClass(short fClass)

来注册“窗口类”，如果 m_fRegisteredClasses 记录了注册的“窗口类”，返回 TRUE，否则，调用 AfxEndDeferRegisterClass 注册。

注册这些窗口类的例子：

MFC 在加载边框窗口时，会自动地注册“AfxFrameOrView”“窗口类”。在创建视窗口时，就会使用该“窗口类”创建视窗口。当然，如果创建视窗口时，该“窗口类”还没有注册，MFC 将先注册它然后使用它创建视窗口。

不过，MFC 并不使用“AfxMDIFrame”来创建 MDI 主窗口，因为在加载主窗口时一般都指定了主窗口的资源，MFC 使用指定的像标注册新的 MDI 主窗口类(通过函数

AfxRegisterWndClass 完成，因此“窗口类”的名字是动态产生的)。

MDI 子窗口类似于上述 MDI 主窗口的处理。

在 MFC 创建控制窗口时，如工具栏窗口，如果“AfxControlBar”类还没有注册，则注册它。注册过程很简单，就是调用::InitCommonControl 加载通用控制动态连接库。

(5) 调用::RegisterWndClass

直接调用 Win32 的窗口注册函数::RegisterWndClass 注册“窗口类”有一个缺点：如果是 DLL 模块，这样注册的“窗口类”在程序退出时不会自动地取消注册(Unregister)。所以，必须记得在 DLL 模块退出时取消它所注册的窗口类。

(6) 子类化

子类化(Subclass)一个“窗口类”，可自动地得到它的“窗口类”属性。

2.2.2 MFC 窗口类 CWnd

在 Windows 系统里，一个窗口的属性分两个地方存放：一部分放在“窗口类”里头，如上所述的在注册窗口时指定；另一部分放在 Windows Object 本身，如窗口的尺寸，窗口的位置(X，Y 轴)，窗口的 Z 轴顺序，窗口的状态(ACTIVE，MINIMIZED，MAXMIZED，RESTORED...)，和其他窗口的关系(父窗口，子窗口，...)，窗口是否可以接收键盘或鼠标消息，等等。

为了表达所有这些窗口的共性，MFC 设计了一个窗口基类 CWnd。有一点非常重要，那就是 CWnd 提供了一个标准而通用的 MFC 窗口过程，MFC 下所有的窗口都使用这个窗口过程。至于通用的窗口过程却能为各个窗口实现不同的操作，那就是 MFC 消息映射机制的奥秘和作用了。这些，将在后面有关章节详细论述。

CWnd 提供了一系列成员函数，或者是对 Win32 相关函数的封装，或者是 CWnd 新设计的一些函数。这些函数大致如下。

(1) 窗口创建函数

这里主要讨论函数 Create 和 CreateEx。它们封装了 Win32 窗口创建函数::CreateWindowEx。Create 的原型如下：

```
BOOL CWnd::Create(LPCTSTR lpszClassName,
    LPCTSTR lpszWindowName, DWORD dwStyle,
    const RECT& rect, CWnd* pParentWnd, UINT nID,
    CCreateContext* pContext)
```

Create 是一个虚拟函数，用来创建子窗口(不能创建桌面窗口和 POP UP 窗口)。CWnd 的基类可以覆盖该函数，例如边框窗口类等覆盖了该函数以实现边框窗口的创建，视类则使用它来创建视窗口。

Create 调用了成员函数 CreateEx。CWnd::CreateEx 的原型如下：

```
BOOL CWnd::CreateEx(DWORD dwExStyle, LPCTSTR lpszClassName,
    LPCTSTR lpszWindowName, DWORD dwStyle,
    int x, int y, int nWidth, int nHeight,
    HWND hWndParent, HMENU nIDorHMenu, LPVOID lpParam)
```

CreateEx 有 11 个参数，它将调用::CreateWindowEx 完成窗口的创建，这 11 个参数对应地传递给::CreateWindowEx。这些参数指定了窗口扩展风格、“窗口类”、窗口名、窗口大小和位置、父窗口句柄、窗口菜单和窗口创建参数。

CreateEx 的处理流程将在 4.4.1 节讨论窗口过程时分析。

窗口创建时发送 WM_CREATE 消息，消息参数 lParam 指向一个 CreateStruct 结构的变量，该结构有 11 个域，其描述见后面 4.4.1 节对窗口过程的分析，Windows 使用和 CreateEx 参数一样的内容填充该变量。

(2) 窗口销毁函数

例如：

DestroyWindow 函数；销毁窗口。

PostNcDestroy( )；销毁窗口后调用，虚拟函数。

(3) 用于设定、获取、改变窗口属性的函数

例如：

SetWindowText(CString tiltle)；设置窗口标题。

GetWindowText()；得到窗口标题。

SetIcon(HICON hIcon, BOOL bBigIcon)；设置窗口像标。

GetIcon(BOOL bBigIcon)；得到窗口像标。

GetDlgItem(int nID)；得到窗口类指定 ID 的控制子窗口。

GetDC()；　得到窗口的设备上下文。

SetMenu(CMenu *pMenu)；设置窗口菜单。

GetMenu()；得到窗口菜单。

…

(4) 用于完成窗口动作的函数

用于更新窗口，滚动窗口，等等。一部分成员函数设计成或可重载(Overloaded)函数，或虚拟(Overridden)函数，或 MFC 消息处理函数。这些函数或者实现了一部分功能，或者仅仅是一个空函数。如：

- 有关消息发送的函数

SendMessage(UINT message,WPARAM wParam = 0, LPARAM lParam = 0)；给窗口发送消息，立即调用方式。

PostMessage(UINT message,WPARAM wParam = 0, LPARAM lParam = 0)；给窗口发送消息，放进消息队列。

…

- 有关改变窗口状态的函数

MoveWindow(LPRECT lpRect, BOOL bRepaint = TRUE)；移动窗口到指定位置。

ShowWindow(BOOL)；显示窗口，使之可见或不可见。

…

- 实现 MFC 消息处理机制的函数

virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam)；窗口过程，虚拟函数。