Windows Shell 编程指南与实例

萧秋水　文　娟　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.com.cn

Windows Shell 编程指南与实例

Windows Shell
编程指南与实例

萧秋水　文　娟　编著

電子工業出版社
Publishing House of Electronics Industry

内 容 简 介

本书的全部内容适合于 Windows 9X，部分内容适合于 Windows NT 和 Windows 2000，涉及到 Shell 编程的各个侧面，包括名字空间、Shell 扩展处理程序、任务条通知区、应用程序任务条、屏幕保护程序等众多有趣话题。

本书的一大特点是编程实例丰富，所附光盘中给出了许多源代码，同时书中给出了讲解。

本书适合于进行 Windows 编程的中高级程序员，尤其是正在对 Windows Shell 进行扩展的程序员、VC6 和 Delphi 的中高级用户及相关的爱好者。

图书在版编目(CIP)数据

Windows Shell 编程指南与实例/萧秋水，文娟编著．北京：电子工业出版社，2000.10
ISBN 7-5053-6273-9

Ⅰ．W… Ⅱ．①萧… ②文… Ⅲ．操作系统，Windows-程序设计 Ⅳ．TP316.7

中国版本图书馆 CIP 数据核字(2000)第 75182 号

书　　名：Windows Shell 编程指南与实例
编 著 者：萧秋水　文　娟
责任编辑：宋　漪
特约编辑：刘　瀚
排版制作：电子工业出版社计算机排版室
印 刷 者：北京兴华印刷厂
装 订 者：三河市双峰装订厂
出版发行：电子工业出版社　URL：http://www.phei.com.cn
　　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：787×1092　1/16　印张：20　字数：510 千字
版　　次：2000 年 10 月第 1 版　2000 年 10 月第 1 次印刷
书　　号：ISBN 7-5053-6273-9/TP·3385
印　　数：5000 册　　定价：30.00 元

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。如何在自己的应用程序中利用 Windows 漂亮的外壳，并在其中加入自己的个性化元素（注意，不是换几块桌布这种简单的事情），使 Windows 按照应用程序指定的方式运行从而给用户提供最大的方便，是一个颇为值得探讨的问题。这个问题涉及到对 Windows 外壳进行编程，也就是本书要讨论的内容。本书的全部内容适合于 Windows 9x，部分内容也适合于 Windows NT 和 Windows 2000。

写作本书的缘由说来也颇为简单，笔者在进行 Windows Shell 扩展的开发过程中，本想找几本书作为参考，结果发现市面上很难找到一本合意的。大部分编程的指导书中都未曾提起，偶尔提起，也是语焉不详，三言两语带过。于是笔者便萌生了自己写一本此类书籍的冲动，这便有了本书。希望读者能从中获益，在进行一些 Windows 的高级开发时，不要再走笔者曾经走过的弯路。

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，下面对本书的内容做一个简单的概述。

第 1 章“从 COM 说起”介绍了组件对象模型的一些基本概念和实现方法，以及它与 Windows Shell 的关系。不熟悉 COM 的读者一定要仔细地阅读这一章，为后面的学习打下基础。

第 2 章“Windows Shell 名字空间”介绍了 Windows 的名字空间和在其中进行漫游的方法，并在介绍中创建了一个 Windows 资源浏览器的框架，读者可以使用它在名字空间自由地漫游。

第 3 章“Windows Shell 扩展”介绍了 Windows Shell 扩展处理程序的基本概念及其概貌（如分类等）。

从第 4 章“拷贝钩子处理程序”到第 10 章“数据处理程序”，尽可能地拓展了对 Windows Shell 扩展处理程序的说明，分别介绍了七种 Shell 扩展处理程序的使用机制及其编程方法，这七种 Shell 扩展处理程序包括拷贝钩子处理程序、上下文相关菜单处理程序、拖放处理程序、图标处理程序、属性表处理程序、放置处理程序和数据处理程序，是 Shell 编程中极为重要的一块。

第 11 章“任务条通知区”介绍了 Windows 桌面任务条通知区在应用程序中的使用方法，是当今应用程序开发中极为流行的一项技术。

第 12 章“应用程序桌面任务条”介绍了 Windows 为应用程序提供的任务条服务，应用程序可以使用这项服务生成与 Windows 的任务条类似的应用程序任务条。

第 13 章“Windows Shell 链接”介绍了 Shell 链接（也就是桌面上常见的快捷方式）的创建方法。

第 14 章“屏幕保护程序”讲解了屏幕保护程序的运行机理以及编程实现自己的屏幕保护程序所必须遵循的规则。

各章的最后大都附有“常用的参考信息”，这些参考信息是笔者从相关文档中精选出来的，读者在进行自己的开发时可以查阅这些信息。

本书的一大特点是编程实例丰富，不仅在所附光盘中给出了作者在平时编码过程中积累的许多源代码，而且在书中给出了必要的、详细的讲解，引导读者逐步了解 Shell 编程的奥妙。此外，所附源码的可重用性强，对于有的主题，如任务条通知区、应用程序任务条等，读者甚至不用从头编写与 Shell 交互的部分代码，而只要重用例程中给出的代码即可。

本书适用于进行 Windows 编程的中高级程序员尤其是正在对 Windows Shell 进行扩展的程序员。由于笔者特别酷爱 Visual C++ 和 Delphi，所以书中的许多程序给出了 VC 6 和 Delphi 5 的两种实现，以供 VC 和 Delphi 的中高级用户参考。

限于编者水平，难免在内容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。

编者

2000 年 9 月

目　录

第 1 章　从 COM 说起 …… (1)
1.1　为什么要使用 COM 技术 …… (1)
1.2　COM 的几个基本概念 …… (2)
1.3　COM 接口初探 …… (3)
1.4　说说 GUID、CLSID、IID …… (4)
1.5　COM 服务器的形式 …… (5)
1.6　用 MFC 实现一个 COM 服务器的简单步骤 …… (6)
1.6.1　创建一个 MFC AppWizard(DLL)项目 …… (7)
1.6.2　声明组件和接口的 GUID …… (7)
1.6.3　声明组件和接口 …… (8)
1.6.4　声明组件类 CTestSvr …… (8)
1.6.5　实现类厂和接口映射 …… (10)
1.6.6　为嵌套类实现 IUnknown 接口 …… (10)
1.6.7　实现 ITest 接口的方法 TestMethod …… (11)
1.6.8　管理全局对象计数 …… (11)
1.6.9　实现一个客户应用程序 …… (11)
1.7　Delphi 对 COM 的包装 …… (13)
1.7.1　创建一个 ActiveX Library 项目 …… (14)
1.7.2　为项目添加一个 COM 类 …… (14)
1.8　Windows Shell 扩展的实质 …… (17)
第 2 章　Windows Shell 名字空间 …… (19)
2.1　Shell 名字空间的使用 …… (19)
2.1.1　Shell 名字空间的单一结构 …… (19)
2.1.2　名字空间中元素的标识 …… (20)
2.1.3　Shell 名字空间的漫游机制 …… (20)
2.1.4　Shell 名字空间提供的接口 …… (21)
2.2　搭建一个真正的资源浏览器的框架 …… (22)
2.2.1　使用 MFC AppWizard(exe)创建一个项目 …… (22)
2.2.2　修改程序的界面架构 …… (22)
2.2.3　将树型视图的根节点绑定到名字空间的根 …… (26)
2.2.4　介绍几个重要的辅助函数 …… (30)
2.2.5　当用户选中树型视图节点时,实现漫游 …… (35)
2.2.6　看看我们的框架 …… (39)
2.3　常用的参考信息 …… (40)
第 3 章　Windows Shell 扩展 …… (55)

3.1 Shell 扩展的基本概念 …… (55)
3.2 Shell 扩展的分类 …… (56)
3.3 Windows Shell 如何访问 Shell 扩展 …… (57)
3.4 编写 Shell 扩展的基本步骤 …… (58)
3.4.1 手工实现 Shell 扩展(无需编程) …… (59)
3.4.2 需要编程实现的 Shell 扩展 …… (64)
第 4 章 拷贝钩子处理程序 …… (67)
4.1 拷贝钩子处理程序的使用 …… (67)
4.2 用 VC 6 实现一个拷贝钩子处理程序 …… (67)
4.2.1 创建一个空的 DLL 项目 …… (68)
4.2.2 为项目添加一个类 CCopyHook …… (68)
4.2.3 为项目添加另一个类 CCopyHookClassFactory …… (70)
4.2.4 给 DLL 添加两个全局变量 …… (70)
4.2.5 给 DLL 添加几个需要实现的标准函数 …… (71)
4.2.6 为类 CCopyHook 添加实现代码 …… (71)
4.2.7 为类 CCopyHookClassFactory 添加实现代码 …… (74)
4.2.8 实现 DLL 中的几个全局函数 …… (76)
4.2.9 为 DLL 添加一个 DEF 文件 …… (78)
4.2.10 编译链接该 DLL 项目生成 CopyHook.dll …… (78)
4.2.11 编辑该拷贝钩子处理程序的注册文件 …… (78)
4.2.12 测试该拷贝钩子处理程序 …… (79)
4.3 用 Delphi 5 来实现这个拷贝钩子 …… (80)
4.3.1 创建一个 ActiveX Library 项目 …… (80)
4.3.2 为项目添加一个 COM 类 TMyCopyHook …… (81)
4.3.3 实现 TMyCopyHook …… (83)
4.3.4 实现服务器的注册与反注册 …… (84)
4.3.5 编译链接并测试该拷贝钩子处理程序 …… (85)
4.4 常用的参考信息 …… (86)
第 5 章 上下文相关菜单处理程序 …… (89)
5.1 上下文相关菜单处理程序的使用 …… (89)
5.2 用 VC 6 实现一个上下文相关菜单处理程序 …… (90)
5.2.1 创建一个空的 DLL 项目 …… (91)
5.2.2 为项目添加一个类 CContextMenuExt …… (91)
5.2.3 为类 CContextMenuExt 添加实现代码 …… (93)
5.2.4 为类 CContextMenuExt 实现一个类工厂 …… (101)
5.2.5 添加并实现 DLL 服务器的框架 …… (102)
5.2.6 编译链接该 DLL 项目生成 ContextMenuExt.dll …… (102)
5.2.7 编辑上下文相关菜单处理程序的注册文件 …… (102)
5.2.8 测试该上下文相关菜单处理程序 …… (103)
5.3 Delphi 5 的实现 …… (103)

5.3.1 创建一个 ActiveX Library 项目 …… (103)
5.3.2 为项目添加完成处理功能的 COM 类 TContextMenu …… (104)
5.3.3 实现 TContextMenu …… (104)
5.3.4 实现服务器的注册与反注册 …… (109)
5.3.5 编译链接并测试该上下文相关菜单处理程序 …… (110)
5.4 常用的参考信息 …… (111)
第 6 章 拖放处理程序 …… (119)
6.1 拖放处理程序的使用 …… (119)
6.2 用 VC 6 实现一个拖放处理程序 …… (119)
6.2.1 为项目添加一个类 CDrapDropExt …… (119)
6.2.2 为类 CDrapDropExt 添加实现代码 …… (121)
6.3 常用的参考信息 …… (128)
第 7 章 图标处理程序 …… (131)
7.1 图标处理程序的使用 …… (131)
7.2 用 VC 6 实现一个图标处理程序 …… (132)
7.2.1 创建一个空的 DLL 项目 …… (133)
7.2.2 为项目添加一个类 CIconHandler …… (133)
7.2.3 为项目添加需要的资源 …… (136)
7.2.4 为类 CIconHandler 添加实现代码 …… (136)
7.2.5 为类 CIconHandler 实现一个类工厂 …… (140)
7.2.6 添加并实现 DLL 服务器的框架 …… (140)
7.2.7 编译链接该 DLL 项目生成 IconHandler.dll …… (141)
7.2.8 编辑该图标处理程序的注册文件 …… (141)
7.2.9 测试该图标处理程序 …… (142)
7.3 Delphi 5 的实现 …… (143)
7.3.1 创建一个 ActiveX Library 项目 …… (143)
7.3.2 为项目添加一个 COM 类 TMyIconHandler …… (143)
7.3.3 为 DLL 项目添加图标资源 …… (145)
7.3.4 实现 TMyIconHandler …… (146)
7.3.5 实现服务器的注册与反注册 …… (148)
7.3.6 编译链接并测试该图标处理程序 …… (149)
7.4 常用的参考信息 …… (149)
第 8 章 属性表处理程序 …… (153)
8.1 属性表处理程序的使用 …… (153)
8.2 用 VC 6 实现一个属性表处理程序 …… (155)
8.2.1 创建一个空的 DLL 项目 …… (155)
8.2.2 为项目添加一个类 CPropSheetHandler …… (155)
8.2.3 为项目添加需要的资源 …… (156)
8.2.4 为类 CPropSheetHandler 添加实现代码 …… (157)
8.3 常用的参考信息 …… (165)

第 9 章　放置处理程序 …… (171)
9.1　放置处理程序的使用 …… (171)
9.2　用 VC 6 实现一个支持特定文件类型拖放的程序 …… (171)
9.2.1　使用 MFC AppWizard 创建一个简单编辑器 …… (172)
9.2.2　分析 AppWizard 生成的程序框架 …… (173)
9.3　常用的参考信息 …… (177)
第 10 章　数据处理程序 …… (181)
10.1　数据处理程序的使用 …… (181)
10.2　常用的参考信息 …… (181)
第 11 章　任务条通知区 …… (189)
11.1　任务条通知区的管理机制 …… (189)
11.2　用 VC 6 更新任务条通知区 …… (190)
11.2.1　创建一个 MFC AppWizard(exe)项目 …… (191)
11.2.2　为项目添加类 CTrayNotifyIcon 并对其进行分析 …… (191)
11.2.3　为项目添加一个辅助类 CTrayRessurectionWnd …… (193)
11.2.4　为类 CTrayRessurectionWnd 添加实现代码 …… (194)
11.2.5　为项目添加另一个辅助类 CTrayTimerWnd …… (195)
11.2.6　为类 CTrayTimerWnd 添加实现代码 …… (196)
11.2.7　为类 CTrayNotifyIcon 添加实现代码 …… (198)
11.2.8　为类 CTrayNotifyIcon 添加测试代码 …… (205)
11.3　Delphi 5 的实现 …… (206)
11.3.1　组件的类别 …… (206)
11.3.2　建立组件的准备工作 …… (207)
11.3.3　组件的创建步骤 …… (207)
11.3.4　在 Delphi 中安装自定义的组件 …… (209)
11.3.5　创建 TTrayIcon 组件并为其添加成员 …… (209)
11.3.6　为 TTrayIcon 组件添加实现代码 …… (212)
11.3.7　安装 TTrayIcon 组件并在应用程序中进行测试 …… (221)
11.4　常用的参考信息 …… (221)
第 12 章　应用程序桌面任务条 …… (223)
12.1　应用程序桌面任务条的管理机制 …… (223)
12.1.1　桌面任务条的注册与删除 …… (224)
12.1.2　桌面任务条的状态 …… (224)
12.1.3　桌面任务条的外观 …… (225)
12.1.4　系统发给任务条的通知消息 …… (225)
12.2　用 VC 6 创建应用程序桌面任务条 …… (226)
12.2.1　创建一个 MFC AppWizard(exe)项目 …… (226)
12.2.2　为项目添加类 CAppBar 并对其进行分析 …… (227)
12.2.3　为类 CAppBar 添加实现代码 …… (228)
12.2.4　为类 CAppBar 添加测试代码 …… (246)

12.3 Delphi 5 中更为漂亮的实现 …… (246)
12.3.1 创建 TAppBar 类并为其添加成员 …… (246)
12.3.2 为 TAppBar 类添加实现代码 …… (251)
12.3.3 从 TAppBar 类派生自己的任务条 …… (270)
12.4 常用的参考信息 …… (270)
第 13 章 Windows Shell 链接 …… (273)
13.1 Shell 链接的使用 …… (273)
13.1.1 .lnk 文件 …… (274)
13.1.2 IShellLink 接口 …… (274)
13.2 用 VC 6 实现 Shell 链接 …… (276)
13.2.1 创建一个以对话框为基础的应用程序项目 …… (276)
13.2.2 为项目添加必要的资源 …… (277)
13.2.3 添加核心代码 …… (277)
13.3 Delphi 5 的实现 …… (279)
13.3.1 创建一个普通的应用程序项目 …… (279)
13.3.2 创建程序自身的 Shell 链接 …… (280)
13.3.3 获取 Shell 链接存储的信息 …… (281)
13.3.4 测试应用程序 …… (284)
13.4 常用的参考信息 …… (284)
第 14 章 屏幕保护程序 …… (285)
14.1 屏幕保护程序的运行机制 …… (285)
14.2 用 VC 6 创建一个屏幕保护程序 …… (287)
14.2.1 创建一个普通的应用程序项目并调整其配置 …… (287)
14.2.2 为屏幕保护窗口创建一个基类 CScreenSaverWnd …… (287)
14.2.3 为配置对话框创建一个基类 CScreenSaverDlg …… (293)
14.2.4 从 CScreenSaverWnd 派生一个可用的屏保类 CMySaver …… (295)
14.2.5 实现屏幕保护程序的输出函数 …… (299)
14.2.6 添加需要的资源 …… (302)
14.2.7 编译链接并安装测试该屏幕保护程序 …… (302)
14.3 用 VC6 创建一个屏幕保护程序的管理程序 …… (302)
14.3.1 开始屏保 …… (303)
14.3.2 禁止/允许屏保 …… (304)
14.3.3 运行屏保 …… (305)
14.3.4 打开显示属性 …… (306)
14.4 用 Delphi 5 创建屏幕保护程序 …… (307)

第 1 章　从 COM 说起

1.1　为什么要使用 COM 技术

计算机技术无疑是当今世界最为引人注目的潮流，计算机的软硬件每天都在飞速发展，真可谓是日新月异。硬件技术的发展，以 PC 机为例，其 CPU 从 286、386、486、奔腾，一直到今天的奔腾 II 、奔腾III处理器，早就令人眼花缭乱、目不暇接了。而软件技术也经历了一个日益进步的发展过程，这就是我们现在要介绍的内容。

在软件发展的早期，当编译器生成一个应用程序的二进制文件后，在对下一版本重新编译并发行新的版本之前，应用程序一般不会发生任何变化。即使用户需求、操作系统或者硬件环境发生了改变，应用程序的升级也必须等到整个应用程序被重新编译后才能够得以认可。整个软件业就是这样随着已发行软件的日益“老化”、逐渐更新而奔向未来的。

后来，这种状况发生了变化，这就是组件思想在软件业中的出现。人们可以将单个的应用程序分割成多个独立的部分（即组件）来分别进行处理。这种做法的好处是：可以随技术的不断发展而用新的组件取代已有的组件。此时的应用程序将不再是像以前那样的一个在发行前就已注定过时的实体，而是可以随着新组件对旧组件的不断取代而逐步趋于完善。

软件业发展到今天，面向对象的编程语言已经日趋成熟。使用组件编程可以减少代码的重复开发，降低软件的制作成本。但是，由于编程语言的多样化，用一种语言开发的组件往往不能够被另外一种语言所使用。例如在 Visual C++的编程环境中，你就不能使用 Delphi 的 VCL 控件。

COM（Component Object Model，即组件对象模型）组件的出现，解决了这个令许多程序员无奈的问题。使用 COM 技术给软件带来的影响是巨大的。如果你接手过别人编写的程序，你一定会了解在改动程序前花费大量时间去理解源程序的痛苦，每天面对的都是枯燥而又多层嵌套的函数，以及使人眼花缭乱的全局变量等等。而 COM 则提供了一种二进制标准的、面向对象、可扩充、可重用的通信协议。二进制标准这一点非常重要，因为在这个层次上，你就无需关心别人的组件是用什么编程语言实现的了。一个符合 COM 标准的组件，即一个 COM 组件，对编程人员来说好比一个“黑箱”，它有以下几点好处：

1. 编程语言无关性

一个真正的 COM 组件，对外界公开的仅仅是其属性、方法等规范，至于内部的具体实现，使用它的程序员不需要知道。所以，无论用什么语言编程，COM 组件都可以像该编程语言的自带组件一样使用。

2. 组件的易更新性

组件可以独立于使用它的程序而单独更新，只要它对外提供的接口不变，就不会影响使用该组件的客户程序代码。

3. 可重用性

一个组件可能被多个程序同时使用，而且可以与其他组件组合使用。像 Windows 的 DLL 一样，COM 组件的代码第一次被使用时，才会被装入内存，而以后其他程序使用这个 COM 组件，只会增加它的引用计数。这种调用机制，减少了系统资源的浪费。

1.2 COM 的几个基本概念

COM 技术本质上仍然是基于客户/服务器模式的。COM 客户通常是可执行程序（exe），也可能是动态链接库（DLL），甚至就是 Windows 自己（这一点对本书来说非常重要，将在后面进一步说明）。COM 客户一般应独立于 COM 服务器，因为 COM 客户并不知道 COM 服务器在哪儿，怎样把它唤醒，甚至有没有这样的 COM 服务器都不知道。客户应用程序请求创建 COM 对象并通过 COM 对象的接口操纵 COM 对象，服务器根据客户的请求创建并管理 COM 对象。客户和服务器这两种角色并不是绝对的。组件对象与一般意义上的对象既相似也有区别，一般意义上的对象是一种把数据和操纵数据的方法封装在一起的数据类型的实例，但组件对象使用接口（Interface）而不是方法来描述自己和提供服务。

所谓接口，其精确定义是“基于对象的一组语义上相关的功能”，实际上就是一个纯虚类，也就是说，接口本身只有定义，没有实现。真正实现接口的是接口对象（Interface Object）。一个 COM 对象可以只实现一个接口，也可以实现任意多个接口。

接口是客户与服务器通信的惟一途径，客户只能通过接口访问接口对象中的方法，而不能访问该对象中的数据，客户应用程序是不可能获得整个 COM 对象的引用的。如果一个组件对象有多个接口，通过一个接口不能直接访问其他接口，但是，客户应用程序需要访问其他接口的服务怎么办？COM 模型提供了一种机制，即允许客户调用 COM 接口中的 QueryInterface 函数去查询组件对象所支持的其他接口。当客户调用 QueryInterface 函数后，如果组件对象正好支持要查询的接口，函数就返回该接口的指针，这样，客户就可以访问别的接口提供的服务了。如果组件对象不支持该接口，函数就返回一个出错信息。所以，QueryInterface 函数是很有用的，它可以使客户程序动态了解组件对象所支持的接口。

关于 COM 组件，值得一提的另一个重要概念就是引用计数。引用计数是一种机制，使组件对象具有一定的“智能性”，知道如何管理自己的生存期，它的工作原理是这样的：

当接口对象被第一次询问时，它被创建并赋予引用计数的初始值 1。然后如果有别的客户请求获得该接口对象的指针时，引用计数就加 1。当某客户不再需要组件对象的服务时，它应当释放其获得的接口指针。注意，这导致引用计数减 1,但并不真正释放接口对象，因为可能还有其他客户正在使用，只有当引用计数正好减为零时，接口对象才被删除。

举例来说，假设客户甲向服务器请求 ITest 接口，服务器收到请求后，首先看该接口对象是否已经存在，如果存在就将其引用计数加 1 并返回该接口对象的指针，否则，就创建一个接口对象，并赋予引用计数的初始值 1，同时把该接口对象的指针传递给客户甲。如果这时候客户乙也启动了，并且也向服务器请求 ITest 接口，由于此时 ITest 接口对象已存在，服务器只是简单地返回一个指针，并且将引用计数加 1 使之变为 2。当客户甲不再需要 ITest 接口时，它就释放这个接口。显然，这时候不能删除 ITest 接口对象，因为客户乙还在用着呢。正是有了引用计数这种机制，才使得服务器知道怎样管理自己的接口，何时该删除接

口对象。引用计数这种机制也带来一个问题，就是 COM 程序员在获取和释放接口时丝毫马虎不得，否则一旦出现混乱，可能导致接口对象永远不被删除或者过早地被删除。

1.3 COM 接口初探

通常，描述一个对象时是将其划分为若干的方法和属性，并通过继承这一机制来获取对方法的使用权以及对属性的访问权，从而达到数据处理的目的。而 COM 模块不再允许客户直接调用对象的方法（包括属性及其构造函数，析构函数），COM 只提供一个标准的全局函数来访问对象。从而最大限度地将客户与服务器分离。在客户与服务器之间完全通过接口来连结，使用虚拟函数表（vtbl）的内存结构，实现了一组指向虚拟函数实现的指针。接口本身纯粹只是一种规范，它并不实现自己，接口的方法都是纯虚的。真正实现接口的是另外一个类，这就是所谓的 COM 类或者接口对象。

COM 规范中规定，所有组件对象都必须实现一个叫做 IUnknown 的接口，而且所有其他接口都需要继承 IUnknown 接口。实现 IUnknown 接口的目的是为了找到其他功能性的接口。IUnknown 接口提供一个 QueryInterface 函数，该函数使用一个接口 ID，返回一个指向该对象接口的指针。因为所有其他接口都从 IUnknown 中继承，因此，所用接口中都有 Query Interface 函数。

IUnknown 接口的定义如下：

```
IUnknown
{
public:
    BEGIN_INTERFACE
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject) = 0;

    virtual ULONG STDMETHODCALLTYPE AddRef( void) = 0;

    virtual ULONG STDMETHODCALLTYPE Release( void) = 0;

    END_INTERFACE
};
```

其中，AddRef() 和 Release()正是用来跟踪前面介绍过的接口的引用计数。

IUnknown 提供了 COM 库的核心服务，包括引用计数和查询其他接口的功能。任何一个接口，除了它自身的成员外，还继承了它的祖先接口中的成员，当然肯定包括 IUnknown 接口中的成员。要说明的是，在 COM 中，继承并不是代码重用的手段，因为虽然一个接口从它的祖先接口中继承了方法，但是并没有继承实现祖先接口的代码，也就是说，该接口必须重新实现祖先接口中的方法，就好像祖先接口中的方法都是纯虚的一样。声明了接口

后，还要具体实现接口才能被外部代码访问。

接口实际上是面向对象编程思想的一种体现，它隐藏了 COM 对象实现服务的细节，COM 对象可以完全独立于访问它的客户，只要接口本身保持不变。如果实在需要修改和更新接口方式，只要重新增加一个新的接口，这样，对于只知道老接口的客户来说，就得到了最大程度的保护和兼容。

在 COM 中，除了 IUnknown 接口之外，还有另一个特殊的接口也十分重要，它就是 IClassFactory 接口。当然，IClassFactory 也是从 IUnknown 派生的，但除了继承 IUnknown 接口的三个方法之外，它还定义了两个重要的成员函数 CreateInstance 和 LockServer：

```
IClassFactory : public IUnknown
{
public:
    virtual /* [local] */ HRESULT STDMETHODCALLTYPE CreateInstance(
        /* [unique][in] */ IUnknown __RPC_FAR *pUnkOuter,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject) = 0;

    virtual /* [local] */ HRESULT STDMETHODCALLTYPE LockServer(
        /* [in] */ BOOL fLock) = 0;

};
```

之所以说 IClassFactory 接口在 COM 中十分重要，是因为在 COM 中无法直接调用构造函数，于是只好让组件服务器来决定如何构造接口对象，为此组件服务器提供了类工厂（Class Factory），从而将具体的创建过程封装起来。所谓类工厂，其实就是实现了 IClassFactory 接口的特殊 COM 类。

1.4 说说 GUID、CLSID、IID

首先让我们增加一点感性认识。在 Windows 注册表中，有一个键叫做 HKEY_CLASSES_ROOT，它有一个子键叫做“CLSID”，使用 Regedit 实用程序打开它，可以看见所有的 COM 类注册信息，如图 1-1 所示。

在一个复杂的系统中，可能充斥着大量的组件对象，每个组件对象可能又有大量的接口，为了保证这些接口彼此不会冲突，Microsoft 规定用 GUID（GUID 是 Globally Unique Identifier 的缩写，意为全局惟一标识符）来标识组件对象。GUID 可以标识组件对象的类，这时候 GUID 称为 CLSID（ClassIdentifier 的缩写）。GUID 也可以标识组件对象的接口，这时候 GUID 则被称为 IID（Interface Identifier 的缩写）。实质上，GUID 是一个 128 位的整数（也就是 16 个字节），可以绝对保证每一个组件对象及其接口都具有惟一的 GUID。GUID 的定义如下：

图 1-1　注册表中的组件注册信息

```
typedef struct _GUID {          // size is 16
    DWORD Data1;
    WORD   Data2;
    WORD   Data3;
    BYTE  Data4[8];
} GUID;
```

1.5　COM 服务器的形式

简单地说，服务器就是一个堆放 COM 组件的“仓库”。前面已经介绍过，一个 COM 组件可以被不同的应用程序同时使用，每个应用程序在使用这个 COM 组件之前，都会对服务器发出请求，服务器根据需要检索“库存”，如果一切顺利，就会产生一个 COM 组件的实例，即 COM 对象。

COM 服务器可以分为如下三种类型：

- 进程内服务器（in-process server）
- 本地服务器（local server 或者 out-process server）
- 远程服务器（remote server）

进程内服务器指一个 COM 服务器在被某个客户应用程序调用时运行在该客户应用程序的进程之内，一般来说进程内服务器以动态链接库的形式存在。进程内服务器被客户应用程序使用的原理图如图 1-2 所示。

本地服务器是指一个以 exe 可执行文件方式存储的服务器，它包含有客户应用程序需要的 COM 组件库。当它被某个客户应用程序调用时，服务器运行在自己的进程空间之内，而

在客户应用程序的进程空间之外。本地服务器被客户应用程序使用的原理图如图 1-3 所示。

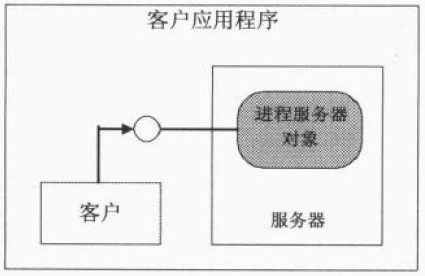

图 1-2　进程内服务器的使用模式

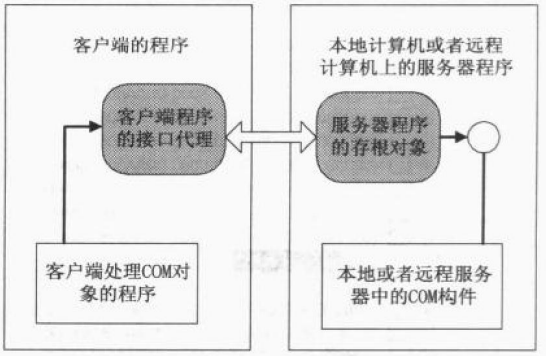

图 1-3　本地服务器的使用模式

从图 1-3 可以看出，使用本地服务器的时候，要用到一个接口代理对象和一个接口存根对象。这是怎么回事呢？其实，当客户应用程序使用 COM 对象的时候，所有该 COM 对象的成员都好像客户应用程序的进程内函数一样被调用，而不管服务器所在的位置。这叫做 COM 的“位置透明性”。通过“位置透明性”，进程外服务器就变得像进程内服务器一样易于使用了。但是，一个进程中的函数入口地址是无法被另外一个进程知道的（这是 Win32 的规范，进程之间是相互独立的，各自拥有虚拟的 4GB 地址空间）。于是，需要通过接口代理对象和接口存根对象来保证不同进程之间的通信。工作原理大致是这样的：

一个 COM 服务器中 COM 组件的成员（包括属性、方法、事件等）会在这个服务器所在的计算机系统中注册，建立存根。当一个客户端应用程序申请使用一个 COM 对象时，它的接口代理对象（这个接口代理对象往往由编译客户端应用程序的编程工具给定，对程序员是透明的）将获得服务器应用程序中 COM 组件的成员函数列表。客户端应用程序是通过访问接口代理对象中的函数列表来调用 COM 组件成员的。接口代理对象和存根对象的工作是交给操作系统处理的，不需要程序员编写什么代码。而操作系统的解决方法就是使用 GUID 在注册表中进行登记。

进程内服务器和本地服务器与客户应用程序都运行在同一台计算机上，而远程服务器则有所不同。远程服务器运行在其他的计算机上，它既可以以动态链接库的形式存在，也可以以可执行文件的方式存储。这种分布式的 COM 技术被称做 DCOM（Distributed Component Object Model）。远程服务器被客户应用程序使用的原理与本地服务器的使用模式类似。

1.6　用 MFC 实现一个 COM 服务器的简单步骤

如何实现 COM 中的接口，方案之一是通过多重继承来实现。但多重继承存在多个接口方法符号名冲突的问题，即多个接口可能含有同名的方法，但是不同接口的方法实现却各不相同。这时编译器仅允许程序员提供一个方法的实现。显然这么实现是有问题的。

而 MFC 则是通过嵌套类而不是多重继承来实现 COM 接口的，MFC 通过接口映射机制将接口和实现该接口的嵌套类关联起来，MFC 又提供了一套简明的宏来实现嵌套类的定义。

此外，MFC 通过 CCmdTarget 类实现了 IUnknown 接口。

下面介绍在 VC 中使用 MFC 创建 COM 服务器的简单步骤。在服务器中，将要创建一个叫做 TestSvr 的 COM 组件。

1.6.1　创建一个 MFC AppWizard（DLL）项目

首先需要创建一个 MFC AppWizard（DLL）项目，注意在向导的第一步中选择支持 Automation，如图 1-4 所示。事实上，本程序不一定是自动化服务器，但选择支持 Automation 后，VC 生成的代码框架自动实现了几个必要的输出函数如 DllGetClassObject，DllRegisterServer 等，这样就可以集中精力干我们感兴趣的事。

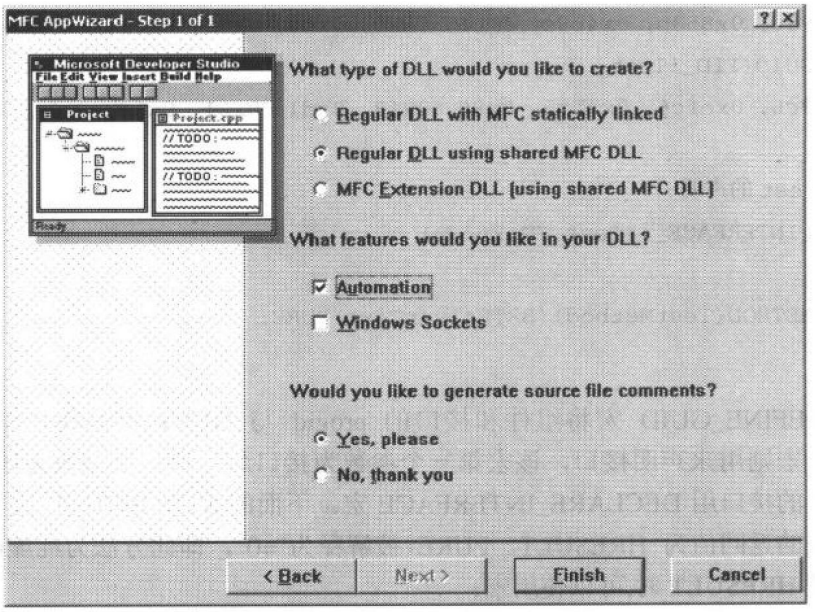

图 1-4　让生成的 DLL 支持 Automation

1.6.2　声明组件和接口的 GUID

为项目添加一个空的头文件，存为 GUIDdef.h，在 GUIDdef.h 中声明组件和接口的 GUID：

```
//声明 CLSID {a9dd5990-853b-4b9c-b689-e845beab5a25}
//DEFINE_GUID(CLSID_TestSvr,
//0xa9dd5990, 0x853b, 0x4b9c, 0xb6, 0x89, 0xe8, 0x45, 0xbe, 0xab, 0x5a, 0x25);
static const IID CLSID_TestSvr =
{0xa9dd5990, 0x853b, 0x4b9c, 0xb6, 0x89, 0xe8, 0x45, 0xbe, 0xab, 0x5a, 0x25}};
// 声明 IID{e51db9c6-efc6-47f5-85f3-d1216d472fa1}
//DEFINE_GUID(IID_ITest,
//0xe51db9c6, 0xefc6, 0x47f5, 0x85, 0xf3, 0xd1, 0x21, 0x6d, 0x47, 0x2f, 0xa1);
static const IID IID_ITest =
```