**Windows Shell编程**

**第一章 Windows Shell是什么**

一个操作系统外壳的不错的定义是它是一个系统提供的用户界面，它允许用户执行公共的任务，如访问文件系统，导出执行程序，改变系统设置等。MS-DOS有一个Command.COM扮演着这个角色。然而Windows已经有了图形界面环境，他的外壳程序也就必然是图形方式的。在Windows95以前，默认的Windows Shell就是程序管理器。

         程序管理器是一个中央控制台，从那里你可以启动应用程序，重排和重组图标，执行新任务。换句话说，程序管理器就像他的名字提示的那样管理所有集中在Windows大伞之下的程序。现在对应程序管理器的是文件管理器，它是一个专门为维护文件系统而设计的工具。

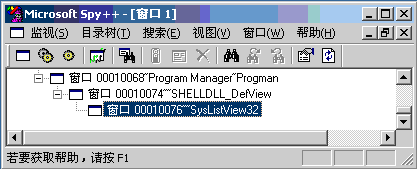
         随着Windows95的出现，探测器取代了这两个老工具，并集成了二者的功能，如果你愿意，你仍能发现文件管理器仍然深深地隐藏在Windows系统目录中。然而，由于用户友善性方面比他的后继者差，现今已经很少使用了。

         一般错误的概念认为，探测器就是一个程序，当你需要通过点击“我的计算机”或右击“开始”按钮来浏览文件系统时这个程序启动。事实上，探测器总是启动和运行着的，从引导开始一直到你关闭计算机。直觉是“探测器”实际上就是新概念下的窗口。探测器是一个可执行模块（explorer.exe），它实现了Windows外壳功能。

         在这一章中，主要是介绍外壳和探测器，更精确地讲是

Shell由许多不同的部分组成，现在我们就从最显而易见的**桌面**和**任务条**开始。从概念上讲，桌面是所有Windows Shell对象的父对象，即宿主对象。就实现方式而言，桌面是一个系统定义窗口类的特殊窗口（命名为#32769），并且是所有窗口的祖先窗口。那些导出应用的顶层窗口一般而言都是桌面的子窗口。在这些子窗口中有一个有趣的窗口子树，它的根是“程序管理器”。

*你可以用CV++带的工具Spy++来检查窗口栈中打开的窗口。*

**

程序管理器保持了兼容性，在图中你可以看到，他的封装结构：程序管理器的直接下级是一个命为SHELLDLL DefView的窗口类，这个窗口包含了一个默认的观察对象，事实上这个窗口负责枚举标准文件夹的内容，它总是从**ListView**控件中导出，这是一个通用的系统控件。SHELLDLL DefView窗口包含了一个ListView(类名为SysListView32)和一个标题控件(类名为SysHeader32)，标题控件仅仅用于ListView的报告观察。

         随着IE4.0的活动桌面和Windows98的发布，默认的观察对象已经有了基于某些浏览器能力的变化。在下一章中我们将更进一步讨论这些观察对象和他们的变化。

就象前面提到的一样，程序管理器窗口是为了兼容性而保留的。它正好演示了一个窗口应用从16位到32位的演变过程。在Win16环境中，与Shell唯一的通讯方式是通过**动态数据交换（DDE）**。这个层面的代码在Windows95 甚至Windows98 中还在维护。为什么呢？，为了兼容性。

         关于DDE接口变成与Shell的详细说明，建议察看资料Internet ClientSDK其中给出了最新的信息。DDE是一款较老的技术，微软已经有大量的资料说明。

主要的Windows Shell部件就是任务条，它实际上就是由探测器进程所拥有的一个窗口。每当你需要终止探测器进程的时候，都将引起任务条的消失和重新显现。每当他重新显现的时候他注册一个具有不同HWND的新窗口。因而，就引用而言，你没有必要保留这个窗口的HWND 。任务条也是也各拥有“开始”按钮的窗口，在托盘区域有时钟和类似按钮的控件，表示正在运行的应用。

         任务条实际上与窗口一样，可以在其上作任何窗口上可以做的操作如移动、隐藏和子类化等。在第七章中我们将说明怎样子类化任务条和“开始”按钮。在第九章中可以找到怎样隐藏任务条和编成重新启动Shell。这后一个特性在编成实现Shell和命名空间扩展时是有用的。

你是否奇怪，桌面上的快捷方式是从哪里来的和属于谁，坦白地讲，开始我也认为探测器模块绘制了这些图标，记录了用户设置，颜色，状态等。这个模块开可能在桌面背景上绘制用户设置的墙纸。

         桌面并不是这样工作的，相反，快捷方式作为图标是由一个普通的ListView显示的。当然这个ListView有了一个不寻常的变异，但是它确实是一个ListView，因此它也就不难从消息中获取处理对象了，将在第九章中给出例程进行说明。

         探测器是一个扮演着系统外壳角色的应用程序。当说到外壳扩展的时候，我们讲的就是有探测器感知的、装入的并最终执行的代码块。

         探测器可以被看作为一个微型的窗口开发环境。想象一下：它有自己的函数和对话框；允许写特殊的与已存在的体系集成的应用程序；能包容应用和文档。他甚至可以解释任何活动的脚本兼容语言的脚本（VBScript，JScript，Perl，等等）。本书中将包含所有这些特征。

随Windows3.1一起发布的文件管理器有着非常好的旦未充分利用的特性，比如：它能够在运行时加载DLL，执行具有特殊原形的已注册函数等。也就是说，在这个资源中有一点，其代码本身能够知晓某些活动是由用户执行的。换句话说，文件管理器支持扩展行为，当执行特定的活动时，查找注册的扩展并装载和运行之。

         我们后面在探测器外壳和命名空间扩展中见到的实际上有相同的原理。不同的完全是细节方面的实现。文件管理器用于加载具有预定义原形的传统的DLL全程函数，而探测器是这一过程更加规范。尤其是它采用COM接口(可以看作预定义和固定函数原型的集合)和进程内服务器(实质上的DLL)

*当然，COM接口和进程内服务器比函数集和DLL更复杂，但是，这也使探测器进程比老的基于DLL的进程更规范和有力。*

         在探测器环境中，基本上有两种类型的扩展：外壳扩展和命名空间扩展。从名字上看有点混淆，探测器就是Windows的外壳，所以两种类型的扩展都可以作为外壳扩展。换句话说，外壳和命名空间扩展二者都扩展了探测器能力。但是在他们之间有一些差别。

         外壳扩展是一种加到给定类型的所有文件上的客户行为，给定类型的文件按照客户要求显示在探测器的观察中。如此，你可以称之为“外壳观察的扩展”。客户的行为，如拖拽，右击关联菜单，绘制图标或显示属性对话框等由一定数量的特殊事件触发。你可以定义这些事件的特殊处理器程序，例如你可以确定显示给定.Bmp文件的图标，为所有Windows 元文件加一个预览页面到属性对话框，甚至可以在关联菜单中增加一个可执行功能。将在十五章中给出例程。

         命名空间扩展有两种形式，这要看你怎样连接。如果你用文件类型连接命名空间扩展，尽管有复杂的代码支持，其功能上仍然等价于关联菜单的扩展。然而，如果你用文件夹连接命名空间扩展，这个文件夹将变成客户文件夹，你的代码将确定文件夹的内容、探测器显示的图标、子文件夹、排序、关联菜单等。

这个问题很有道理，简单的回答就是，为了使我们的应用根号和更丰满。但是这个回答有点过于辞令化，我们这样做是为了使我们的模块与系统集成到一起，或者说更自动化。

         有两种方法对外壳编程，使用API函数和使用COM接口。这两种方法既不互相独立也不相互重叠，它们是两个不同的方向和两种不同领域，这一点我们在下一章中将进一步阐述。现在让我们直接浏览一下个章节的内容。

         你知道Shell编程要求使用API函数和COM接口，API函数使你能够访问Shell的基本功能，如，文件操作，浏览文件夹，执行程序，处理图标和快捷方式等。当你想要增强和精炼Shell的基本行为时，COM方法则触及到了客户化Shell扩展的核心。

         这本书中首先给出所有API函数的解释，进一步探讨函数的原型，资料介绍的差别以及其中的Bugs。通常我的目的是要澄清所有你在资料中遇到的含混之处。第三章到第九章涉及到特殊的APIs分组，其中包含了典型的Shell操作，特别在第三章中讲解SHFileOperation()函数，涉及到文件的拷贝、移动、删除和重命名操作。第四章揭示了神秘的SHGetFileInfo()函数，系统提供了获取文件(属性、图标、类型和显示名)的系统信息和Shell信息的方法。第五章解释文件夹内部组织的叠放过程，包括设置、浏览和象Favorites和SendTo那样的特殊文件夹。

快捷方式和属性在第六章中介绍，其中将介绍建立和解析快捷方式和经常使用的字段。第七章则正式进入探测器地址空间，并且从另一个角度讨论客户化问题：在探测器不可知的情况下什么是你所能安全操作的。特别是我们向你展示一个置换过的“开始”按钮和不同的菜单。一旦你这样做过之后，你就有了完全控制Windows系统的能力了。在余下的第八和第九章中我们将讲述程序的扩张，图标和任务条，我们将说明怎样编程加入新的具有自己的菜单的按钮到任务条中。

这本书的第二部分是基于要求COM接口的探测器特征的。但是直到第十二章之前我们还没有涉及到接口知识，中间的两章作为Shell函数和探测器接口的桥梁。第十章包含了最近更新的Windows的SDK函数。第十一章给出了Shell对象的概览，例如“我的公文包”，“控制面板”和“打印机”，以及客户文件夹的概念。在这一章中还包括了其他Shell对象和RunDLL32使用程序的说明以及全部探测器命令行的解释。

第十二章介绍Shell对象模型，首先致力于把API函数的一个子集移动到对应的COM接口中去，这个特性最少要求系统中要安装“活动桌面”。有趣的是这个对象模型允许你访问系统的某些功能（绝大多数系统对话框）。

第十三章介绍Windows脚本环境，这是一个执行Windows批处理文件的运行时引擎。技术上讲，这并不是一个Shell实体，但是它与Shell有重要的留级关系。Windows脚本环境显露一个对象模型，使你能够使用VBScript，Jscript等任何脚本语言编写程序。我将通过加入有用的新对象来扩展这个模型。

第十四章集中于指导你采用Shell和命名空间扩展的应用和理由方面。我将揭示实际上的Shell集成的应用究竟是什么和为什么说Shell扩展是把应用模块与系统Shell融合的最好方法。第十五章说明怎样写一个Shell扩展来客户化关联菜单、图标和属性，以及怎样排错。第十六章概括了命名空间扩展内容，并且包含一个例子，说明怎样加一个可展开节点到探测器的树观察中，并以文件夹的形式展示了当前窗口的堆栈过程。

       这一章中我们描绘了未来各章中打算作的事情，尤其是我们试图解释：

              各Shell版本之间的差异  
  
**第二章Shell的结构**

  “ Shell  编程”的大伞之下有大量的 API 函数和 COM 接口。这个种类繁多的‘命令’集允许你用不同的方法对 Windows Shell 进行编程。函数和接口并不是两种提供相同功能的等价途径，相反，它们在不同的逻辑层上提供不同的功能。

         API 函数包含了用户想要在 Shell 对象上执行的基本操作，如文件和文件夹。 COM 接口则给出了扩展增强，甚至客户化各种要素对象的机会，包括 Shell 本身标准行为。用面向任务的方法对函数和对象进行分组将给我们一个总体上观察 Shell 的机会，因此，我们仍然能够把它看作一个具有属性和方法的对象。在这一章中我们将努力分出每一个函数和接口究竟属于哪一个功能组。这将有助于我们从大量的功能碎片中寻找出 Shell 编程接口。

在这一章中，将包含有：

          我们在这本书中使用的定义

         Shell API 函数的功能分组

          由 Shell 和其内涵部件实现的 COM 接口的功能分组

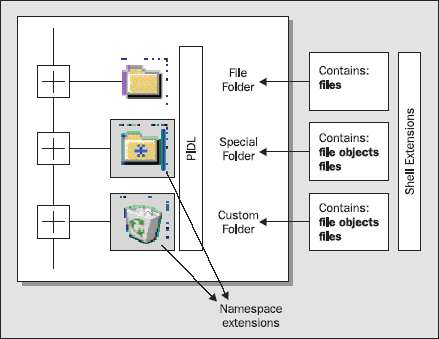
         Shell 结构是怎样随导入的活动桌面而演化的

最后，你将能更好地理解这本书的知识范围，并且作为 Shell 程序员，你将能清晰地勾画出书中的哪些功能是可用的。

**Shell模块**

          实际上到目前为止我们还不能说 Windows Shell 是面向对象的。有一些‘对象’查看结构就能识别它。这些对象也有‘属性’一样的特征，以及象方法一样执行一些活动。但是它们是依赖 API 函数的，一个典型的对象就是文件夹。

          如果 Shell 不是面向对象的，它决不能有一个完全兼容的对象模型。我们能够想象一个看起来象分层对象的体系结构。换言之，有一个对象集合，他们以如下图描述的方法一样工作。



基本上讲， Shell 是由文件夹组成的，文件夹是一个包含有子元素的容器，包括子文件夹，这些元素通常称为**文件夹项**。根文件夹称为‘桌面’，其子项包括‘我的计算机’，‘网上邻居’，‘回收站’，和其他可能的项，所有这些文件夹的集合称之为 Shell 的**命名空间**。

**Shell的命名空间**

          概念上讲，文件夹就类似于文件系统的目录，但是它不一定绑定到一个实际的物理目录上，如果他没有邦定，就称之为虚拟文件夹。我们可以以这种方式区分两种主要的文件夹：正常的文件夹（命名为文件型文件夹或目录）和**客户文件夹**。自然，包含在文件型文件夹中的是文件，其属性是名称、类型、尺寸、最后修改日期等。包含在任何其他文件夹下的项目可以是文件—一般使用其他的扩展特征集—但是也可能是完全不同的东西，如打印机或网络节点。

***文件夹***

文件夹是怎样实现的？文件夹实际上是一个 Shell 对象，它的行为被编码成一个 COM 模块，向 Windows Shell 暴露公共的接口。通过连接，文件夹可以告诉 Shell 怎样设计它的内容，使用什么样的图标显示，采用什么文字来描述，例如‘我的计算机’看起来像一个文件夹。他有一个代码层来感知 PC 上所有可用的驱动器，并且为每个驱动器附加一个子树到探测器的观察中。

          每一种不同的文件夹都有不同类型的层次代码来提供不同的行为，对于文件型文件夹，行为就是扫描文件系统，恢复文件和子文件夹，并且通过列表控件显示它们。而打印机文件夹则记录所遇到的和所安装的打印机，并且为每一个都显示一个图标。你可以设计任何类型的具有任何行为的虚拟文件夹。文件型文件夹（即，目录）只是其中的一种。

          对非文件型文件夹， Shell 的资料相对较少，仅在**特殊文件夹**中有些说明。事实上 Windows Shell 默认提供的是客户文件夹，他们与文件型文件夹的差别是：

          可以包含文件和其他对象

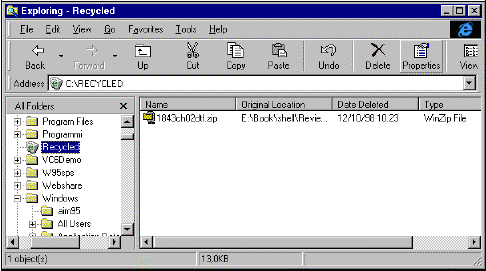
          能够提供对内容的不同观察

          可以有选择地不被邦定到物理目录上

          是系统定义的一部分，这部分由 SDK 提供特殊的函数集。

特殊文件夹列表可以在 Win32 SDK 资料和后面的第五章中找到。就像我原先说过的一样，特殊文件夹是具有自己 COM 模块提供行为的特殊类型的文件夹。由于 COM 模块是新节点被加到 Shell 命名空间的前提，所以它就被称之为**命名空间的扩展**。

          特殊文件夹使用户能够经过适当的接口访问系统信息。也就是说，在大多数情况下，这种文件夹与典型的文件型文件夹提供的内容观察多多少少有些一致的地方。当然，精确的程度依赖于文件夹的类型。



  与普通文件夹一样，特殊文件夹也可以包含文件。然而，通常是以稍微不同的方法表示，显示不同的特征。因此，特殊文件夹给文件赋予了不同的意义，并且，不把它们当作文件系统的正常实体（如果不是这样，它就不是特殊的了）。例如‘回收站’含有正常的而隐含的文件，因为这个文件夹要显示当前被标志为删除的文件列表，所以它把初始位置和删除日期特征显示在前面。

          绝大多数（不是全部）特殊文件夹都依附于磁盘上的物理目录，正常情况下这是一个只读的目录，其内容就是所有需要以最适合的方法显示的信息。

          换一个视角，绝大多数特殊文件夹都需要一个目录来存储它们的数据。这个目录可以被分配到磁盘的任何地方，并且表示为文件夹和 Shell 支持的链接—这个特殊文件夹在命名空间中的位置。目录的内容不必显示为文件列表，相反，关联文件夹的代码可用最适合于它的角色的形式解释和显示它。

          文件夹这个有着包含任何事物能力的东西导出两个重要概念：**文件对象**和 PIDLs ，这些我们将在后面章节中叙述。

***文件对象***

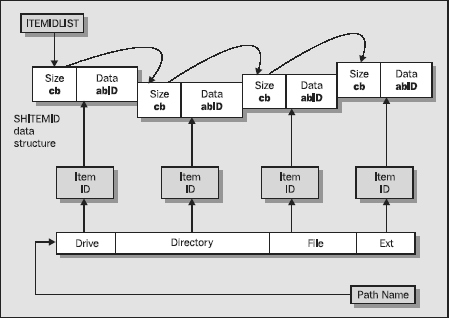
          文件对象是一个包含在普通文件夹中的项—文件、记录、内存块、连接的设备等。‘文件夹项’、‘文件夹元素’和‘文件对象’这些术语是等价的。如果文件夹是一个文件型文件夹则文件对象就是文件。因此这里的‘文件’就比‘文件对象’稍微特殊一点，因为它精确地代表了文件系统中的一个实体。文件是一个文件对象，但是，文件对象不一定是文件。

                   有一个敏感的问题出现在一般的文件夹和文件夹项的概念中，在 Shell 命名空间中我们怎样才能安全并唯一地区分出其中的项。如果 Shell 与文件系统一致（就像 Windows 3.x 一样），则文件的全名就能极好的保证这种唯一性。不可能有两个文件件具有相同的路径和名称。然而当文件夹比文件目录变得更普通的的时候，区分其中的项就需要更普通的方法了。

***PIDLs***

**PIDL** 是一个数据结构，它是唯一地标识包含在文件夹中的项的一种方法。 PIDL —标识符列表指针的缩写（ p ointer to an **id**entifier **l**ist ）—是一种比文件全名更通用的方法，它不仅在文件夹内而且在 Shell 的整个命名空间中保证了项的唯一性。更重要的是，它能透明地处理文件和文件对象。为了理解 PIDLs 的结构和作用，我们来分析一下它的二进制结构并与之所替代的路径名进行比较。

          一个文件全名就是一个字符串，是一个具有非常特殊格式的字符串，是一些子串的串联，每一个子串都在文件系统的层次中表示一个层，有驱动器名，然后是目录名，文件名，最后是扩展名，他们都由反斜线分隔。你所了解的文件全名就是指向这些相连元素的指针—此时指向的是一个字符串。从概念上讲，你可以把它看作是一个数组结构，其中的每一个元素都表示了一个路径名元素。



上图说明了路径名和 PIDL 的关系，同时他也给出了标识符列表在存储器中组织结构的概念。从编程的观点讲， PIDL 是由 LPITEMIDLIST 类型实现的，它是 ITEMIDLIST 结构的指针。

typedef struct \_ITEMIDLIST

{

SHITEMID mkid;

} ITEMIDLIST, \*LPITEMIDLIST;

中间构成路径名各部分的对象映射到 PIDL 的**项目标识**符上。它们存在于整个 SHITEMID 结构中。

typedef struct \_SHITEMID

{

USHORT cb;

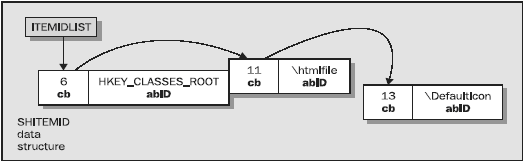
BYTE abID[1];

} SHITEMID, \*LPSHITEMID;

结构的头两个字节指示项目标识符的尺寸—即，相关元素的数据以及用于表示的数据所占用的字节数。 cb 值必须包含它自身的尺寸。对应路径名， cb 应该是所表示的驱动器或目录的长度加上一个 unsigned short 类型变量的长度。随后是这个结构数据的第一个字节。

          一定要记住 PIDL 是一个‘平面’结构，不包含指针。形成 PIDL 的所有数据必须明显地聚集到一起，而不是通过指针连接。这就是说，我们不能使用典型的链表结构方案，使一个元素的最后成员指向链中的下一个元素。还有一点，就像图中所看到的，链表中下一个元素的地址可以通过 cb 相加到当前 SHITEMID 对象计算得出。这是设计规定的，因此要求相连的 SHITEMID s 要连续分配空间。

          定义 PIDLs 的构造规则是约定实现文件夹行为的代码。这些代码也应该确定使用什么样的数据来表示标识符的项。例如，假设想要实现一个文件夹，象文件系统那样显示 Windows 注册表，‘子文件夹’应该是注册表键‘文件对象’应该是注册表值。在这种文件夹中表示每一个元素的可能方法应该是使用相关的键名。这里我们能够看到 PIDL 是怎样使用与前面图中给出相同的方案格式的。注意 HKEY\_CLASSES\_ROOT 是一个长整型值，所以它有四个字节加两个字节的无符号短整数。



  项目链表表示了路径踪迹，从命名空间的根到特定文件夹项。这个标识符链表聚集了链条上的所有元素，说明了一种通过 Shell 唯一地标识一个元素的方法。保证两个项目标识符在内存中连续分配是文件夹对象相关代码的职责。尽管路径名与 PIDLs 类似，他们并不等价，他们也不能交互使用，他们是不同的数据结构。

**Shell观察**

          任何文件夹的内容都是通过一个对象调用 Shell观察 显示在 Windows 探测器中的。每一个文件夹都定义了他自己的 Shell 观察对象，并且所有相关于这个用户接口的任务都指派到这个对象上。对于文件型文件夹 Shell 观察对象是用列表观察控件实现，其中的项就是文件和子文件夹名。默认的 Shell 观察对象在他被调用处理文件时为每一个文件分配图标、**显示名**和**类型名**。

          图标有几种方法确定，这依赖于请求文件的性质。一般使用自身定义的图标显示图标文件（ .ico ），而程序文件则显示其资源中定义的头一个图标。如果没有定义图标，则显示默认的。对于所有其他文件， Shell 通常采用文件归属类所定义的图标。然而正象下面要揭示的那样，这个行为可以被客户化。

          在整个 Shell  环境中，文件都是根据文件扩展名指定的类型分组的，这种根据类型形成的文件集通常称之为**文件类**。它与一个图标和一个描述字符串相连，这个字符串显示在 Windows 探测器观察的详细信息窗口上的类型列上。然而，要置换它们，指定的文件类就需要在注册表中注册， Shell 将从那里读出类型信息和图标。

          一旦定义了文件类，你就可以写代码来影响和修正 Shell 响应某些发生在特定文件类上事件的默认行为，这其中就包括绘制文件图标，弹出关联菜单，和显示属性对话框等。通过定义 Shell扩展 ，你就可以动态地确定这些事件发生时要做些什么。例如，可以在关联菜单中加入新的项，和处理用户的点击，和动态地确定基于每个文件的图标显示。

***钩住Shell***

          一般情况下， Shell 扩展可以看作为钩子，他被设置在整个 Shell 中。 Win32 中，钩子是一段由应用定义的代码，一定事件发生时系统回调这段代码。有许多不同类型的钩子，他们的应用也非常广泛，有一些仅仅影响安装他们的应用程序，而另一些则影响所有系统中运行的应用。

          这其中典型的例子就是键盘钩子，它能够使你在相应消息发送到应用窗口之前得到键盘按下的信息。其他钩子的例子如鼠标活动（移动，点击），窗口管理（建立，析构，活动），和消息处理。更多信息请参见 Win32 SDK 资料。

          从程序员的观点看，钩子是一个具有固定的和预定义语法的**回调函数**，作为回调函数，系统基于已知的原形调用它。 Shell 扩展是 COM 接口，而不是回调函数，但是背后的原理是相同的，二者都允许你指定某些系统将要执行的代码来处理一些预定义的活动。

          这一节特别注意到 Windows 的钩子。通过设置局部钩子，你仅仅能够捕获相关应用内发生的事件。但是设置全程钩子将会导致钩住任何运行的应用所发生的事件。设置全程钩子就是说，你的应用定义了一段代码，它可以被运行中的其他相关进程执行。事实上使用钩子完成 Win32 的跨进程边界和注入代码到其他进程地址空间是最容易的方法。它也是能在所有平台上工作的唯一方法。

**Shell地址空间**

          注入代码到关联的另一个进程是重要的，因为，它允许你访问另一进程没有公开的对象，这对 Shell 编程尤其重要。当你成功地把代码插入到 Shell 地址空间后，你就可以查询 Shell 接口，改变用户接口，甚至置换‘开始’按钮。

          全程钩子是一种使你的代码运行在 Shell 的地址空间中的方法，但是更有力和更灵活的机理是提供**浏览器帮助对象**—一种 COM 对象，探测器和 IE 在启动主窗口时自动加载的对象。

**Shell内存分配器**

          在使用 Shell 时很快你就会接触到内存分配的问题， Shell 提供了一个**存储分配器**，这个封装了 IMalloc 接口的服务可用来代替 New 或 GlobalAlloc() 。

          要获得这个对象的引用，你应该使用 SHGetMalloc() 。它不是返回一个 IMalloc 接口的新指针—由 CoGetmalloc() 函数返回的—而是由系统 Shell 对 IMalloc 对象的一次引用。使用这个指针，你可以安全的释放由 Shell 分配的内存，并且使 Shell 释放这块内存。这可能有点陌生，但是在 Shell 编程中，这是个好习惯。

**Shell任务条**

          任务条窗口作为 Windows 用户接口的一个已知的部件，仅仅是因为它包含了‘开始’按钮。然而我们之所以称之为‘ Windows 任务条’，是因为它实际上是一个窗口系列的特例，称之为‘应用桌面工具条’，最好的例证就是 Office97 的快捷方式杆。有一个特殊的函数和消息集与桌面工具条相关，然有趣的是仅有少量函数和消息影响到 Windows 的任务条。因此，即使资料没有明确地说明，系统任务条和桌面工具条仍然是不同的对象。

          关于任务条的另一个错误观点是它包含了所有运行中应用的按钮，但是有两点原因说明这不是真的：

                    不是所有运行着的应用都显示在任务条上

                    作为按钮，任务条的唯一有的是‘开始’按钮

无论是否相信，作为按钮集出现的实际上是 tab 控件，只是具有特殊的类按钮风格罢了。

          任务条起到了系统控制板的作用，使你能够访问所有运行中的应用。在很多情况下，我们希望能够限制任务条的功能—这是运行在公共 PCs 上应用的一个典型的需求，在那里你不希望用户能够运行其他程序或浏览文件系统。 Win32 API 并没有提供丰富的函数来操作任务条，但是，我们将试图在第九章中对此进行一些补救。

**Shell API 函数**

          在与 VC++6.0 一起提供的 MSDN 库的 Shell 参考一节列出了 100 多函数，然而，其中的大多数都只涉及非常特殊的领域，有时感觉就象是 Windows Shell 的边界领域—这里所说的特殊是关于文件分析和屏幕保护的例程。

          在这本书中，你不能找到关于每一个函数的详尽的说明，然而我们可以集中于文件和文件加操作的核心函数，并试图澄清他们含混不清的资料说明。为了有助于对其进一步分类，我们把它们分作五个不同的函数组。

|  |  |
| --- | --- |
| 组 | 功能 |
| 一般 Windows 函数    Shell 内部函数      任务条函数    文件函数      文件夹函数 | 涉及到屏幕保护，控制面板脚本程序，联机帮助，以及 Shell 拖拽（不是 OLE 拖拽）  访问探测器地址空间的函数，获得 Shell 存储分配器的函数，导出可执行程序的函数以及感觉用户接口改变的函数。  涉及到托盘域的函数和与 Windows 任务条通讯的函数  操作文件的函数，他们执行如‘拷贝’，‘移动’，‘删除’和‘取得信息’等操作的系统活动，和添加文件到特殊的系统文件夹如‘最近文档’等。  操作文件夹的函数，使用这些函数，你可以浏览文件夹，获得系统文件夹的路径，发现文件夹的设置。 |

显示详细信息

根据这个分组结构，可以看到有几个函数作为 Shell 编程接口的一部分并没有被显式引用，但是，他们仍值得出现在这个表中。

|  |  |
| --- | --- |
| 组 | 功能 |
| 图标函数  环境函数  Shell  轻量级 API 函数 | 从执行文件中抽取图标的函数  处理环境变量的函数  容易地访问注册表的函数，读写注册表函数，处理路径名函数，和处理字符串函数。 |

          特别是，有些操作图标和环境变量的函数在 shellapi.h 头文件中，是我们在这里说明他们的主要原因。就像对 Shell 轻量级 API 函数一样（在第 10 章中详细说明），我们说这些函数可以放到任何一类中，但是，他们对 Shell 编程而言有特殊的用途。下一节的表中列出和描述以这种分类方式定义的一些函数。之所以如此，是要你更好地理解函数操作的概念，以及给你提供一个快速查找其中函数的地方。

**一般Windows函数**

          正象标题所提示的那样，这些函数仅仅稍微地涉及到 Windows Shell ，在绝大多数情况下，他们都直接来自于 Windows 3.x 的 API —他们仅处理如帮助文件和拖拽等操作，所有这些函数都很好地支持 32 位 Shell 版本。

|  |  |
| --- | --- |
| 函数 | 描述 |
| DragAcceptFiles() | 标记允许窗口认可拖拽操作。 |
| DragFinish() | 从 Shell 中释放移动文件名列表所分配的内存 |
| DragQueryFile() | 从 Shell 处理拖拽而分配的内存块中抽取文件名 |
| DragQueryPoint() | 获得拖拽发生的点位置 |
| CPlApplet() | 控制面板脚本小程序的主程序 |
| GetMenuContextHelpId() | 返回关联于给定菜单的帮助 ID |
| GetWindowContextHelpId() | 返回关联于给定窗口的帮助 ID |
| SetMenuContextHelpId() | 设置关联于给定菜单的帮助 ID |
| SetWindowContextHelpId() | 设置关联于给定窗口的帮助 ID |
| WinHelp() | 打开帮助文件 |
| ShellAbout() | 显示默认和特定客户化的‘关于’信息框 |
|  |  |

显示详细信息

**Shell内部函数**

          这类函数包括 Shell 底层操作函数，和使你能够进入到 Shell 的地址空间以及可以从一旁操作它并获得对其次年初空间访问的函数。

|  |  |
| --- | --- |
| 函数 | 描述 |
| ShellExecute() | 在指定的文件上执行特殊操作 |
| ShellExecuteEx() | 与上面函数相同，但是有更多的选择 |
| SHChangeNotify() | 通过这个函数程序能够让 Shell 知道什么变化了，以及要求它刷新它所保有的信息 |
| SHGetInstanceExplorer() | 返回探测器 IUnknown 接口指针 |
| SHGetMalloc() | 返回一个指向 Shell 存储分配器的指针 |
| SHLoadInProc() | 装载指定的 COM 对象到探测器地址空间 |

**任务条函数**

         Windows Shell  并没有定义多少操作任务条的函数，所以，控制任务条经常需要自己做很多工作，然而，有两个函数与任务条相关：

|  |  |
| --- | --- |
| 函数 | 描述 |
| Shell\_NotifyIcon() | 显示和管理靠近时钟的托盘区域的图标 |
| SHAppBarMessage() | 发送消息到系统的任务条 |
|  |  |

**文件函数**

          文件是 Windows Shell 最重要的元素之一，图形环境需要文件有许多不同的特性，因此需要特殊的函数来处理。注意，在下表的**版本**列中显示的内容，有些函数是根据最近的 Shell 版本介绍的。

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | 版本 |
| FindExecutable() | 返回指定文件名注册的可执行文件路径 | 所有版本 |
| SHAddToRecentDocs() | 把给定文件的连接加到系统的‘最近文档’文件夹中。 | 所有版本 |
| SHFileOperation() | 用于拷贝、移动、删除或重命名一个或多个文件。 | 所有版本 |
| SHFreeNameMappings() | 释放 SHFileOperation() 函数在特定情况下返回的 存储结构 |  |
| SHGetFileInfo() | 返回给定文件的各种信息块 | 所有版本 |
| SHGetNewLinkInfo() | 建立新的快捷方式名 | 4.71 |

**文件夹(Folder)函数**

就像我们已经讨论的，文件夹比目录更普通一点，它可以包含文件以外的东西，因此文件夹背后的软件就直接涉及到为其中的每一个项返回一个唯一的标识的问题。在活动桌面下，文件夹也可以有它自己的图形特征集。

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | 版本 |
| SHBrowseForFolder() | 显示选择文件夹的对话框 | 所有版本 |
| SHEmptyRecycleBin() | 销毁‘回收站’的内容 | 4.71 |
| SHGetDataFromIDList() | 从标识符表中恢复数据 | 所有版本 |
| SHGetDesktopFolder() | 返回‘桌面’文件夹的 IShellFolder 指针 | 所有版本 |
| SHGetDiskFreeSpace() | 返回指定驱动器的磁盘可用空间 | 4.71 |
| SHGetPathFromIDList() | 返回指定标识符列表的路径名（如果存在） | 所有版本 |
| SHGetSpecialFolder  Location() | 返回特殊的系统文件夹的标识符列表 | 4.71 |
| SHGetSpecialFolderPath() | 返回系统特殊文件夹的路径名（如果存在） | 所有版本 |
| SHGetSettings() | 返回文件夹当前设置的值 | 4.71 |
| SHInvokePrinterCommand() | 向打印机发送命令 | 4.71 |
| SHQueryRecycleBin() | 返回‘回收站’当前占有的空间 | 4.71 |
|  |  |  |

显示详细信息

**图标函数**

          图标是 Windows 图形环境的中心，操作系统外壳最显著的部分。因此，普遍认为，图标是 Windows Shell 编程接口的中心。

|  |  |
| --- | --- |
| 函数 | 描述 |
| ExtractIcon() | 返回可执行文件的图标 Handle |
| ExtractIconEx() | 与上函数相同，但是有更多的选择。 |
| ExtractAssociatedIcon() | 基于文件类，返回指定文件的图标 Handle |

**COM接口**

          我们可以使用 COM 接口就象使用 API 函数那样对 Shell 作类似的操作。再有，使用与 CV++6.0 一同提供的 MSDN 库做为参考，我们可以将涉及到 Shell 相关的 COM 接口分成四类。

|  |  |
| --- | --- |
| 组 | 接口 |
| Shell  扩展 | 涉及到所有 Shell 活动的 COM 接口，从图标到关联菜单，从 UI 活动到文件观察 |
| Namespace  扩展 | 涉及到命名空间扩展的 COM 接口 |
| 钩子 | 能够钩住某些东西的接口，特别是程序执行， URL 转换和建立 Internet 快捷方式 |
| 杂项接口 | 一些零碎接口，如客户化任务条的接口，与打开对话框通讯的接口和对‘我的公文包’编程的接口 |

          对开发者，这些接口并不总是必须实现的—在某些情况下，紧紧需要知道它们，能够适当地调用它们的方法就足够了。下面就更详细点介绍它们。

**Shell接口**

          在开始，我们展示所有 COM 接口，然后利用它们在 Shell 及其扩展上做一点文章。

|  |  |  |
| --- | --- | --- |
| 接口 | 描述 | 版本 |
| IFileViewer ,  IFileViewerSite | 使你能定义对给定类型的文件提供‘快速观察’处理器的模块。 | 所有版本 |
| IInputObject ,  IInputObjectSite | 这两个接口用于处理 UI 活动和对具有接收用户输入的 Shell 对象进行加速操作处理。 | 4.71 |
| IShellIconOverlay ,  IShellIconOverlayIdentifier | 用于发送文件图标重叠消息，使你能够知道用于给定文件的重叠形式。一个图标重叠是 Shell 绘制在图标上的 Bitmap 图像，以便更好地表现它，如，一个手形重叠表示文件夹的共享。 | 4.71 |
| IContextMenu ,  IContextMenu2 | 允许为特殊类型的文件添加新的关联菜单项。  IContextMenu2 处理自绘菜单 | 所有版本 |
| IContextMenu3 | 与 IContextMenu2 相同，但是给出了更好的键盘控制。 | 4.71 |
| IShellExtInit | 执行一个 Shell 扩展的初始化 | 所有版本 |
| IShellChangeNotify | SHChangeNotify() API 函数在 Shell 扩展上的副本，基本上，它允许你写一个模块钩住由 SHChangeNotify() 函数通知的 Shell 层上的变化。 | 4.71 |
| IExtractIcon | 允许你获取任何文件夹项的图标信息。 | 所有版本 |
| IShellIcon | 提供另一种获取任何文件夹项图标信息的方法，在特定情况下，这种方法优于 IExtractIcon 方法。 | 所有版本 |
| IShellLink | 允许建立和解析文件和文件夹的快捷方式 | 所有版本 |
| IShellPropSheetExt | 用于为指定文件类增加属性页到‘属性’对话框。 | 所有版本 |
|  |  |  |

显示详细信息

**命名空间接口**

          要写一个命名空间扩展，本身就需要熟知大量的 COM 接口。这里仅列出最重要的和必须的一些。

|  |  |  |
| --- | --- | --- |
| 接口 | 描述 | 版本 |
| IShellView ,  IShellView2 | 用于定义命名空间扩展的观察对象。 IShellView2 仍然没有文档资料，但是在基于 Web 的观察中有使用。 | 所有版本 |
| IShellBrowser | 显示浏览器，他就是探测器或 Internet 探测器。 | 所有版本 |
| IEnumIDList | 提供 Shell  枚举文件夹内容的方法。 | 所有版本 |
| IShellFolder | 提供令 shell 以标准方式处理客户文件夹的方法。 IShellFolder 对探测器隐藏客户代码。 | 所有版本 |
| IPersistFolder | 使你能初始化某些 Shell 扩展和所有命名空间扩展。 | 所有版本 |
| IPersistFolder2 | 与上相同，加入了一些对基于 Web 的观察更强的支持。 | 4.71 |
| IQueryInfo  Retrieves flags and infotip text for items in a folder. 4.71 | 恢复文件夹项的标志和信息标签文字。 | 4.71 |
|  |  |  |

显示详细信息

**钩子接口**

         Windows Shell 给我们的模块一个机会来感觉一定数量的事件，并使我们可以把客户代码加入其中。

|  |  |  |
| --- | --- | --- |
| 接口 | 描述 | 版本 |
| ICopyHook | 能钩住 Shell 中的所有文件操作 ( 拷贝、移动、删除、重命名 ) 。 | 所有版本 |
| IURLSearchHook | 使你能够探知探测器正在试图转换一个不可知的 URL 协议。 | 4.71 |
| INewShortcutHook | 使你能够探知探测器正在试图建立新的 Internet 快捷方式。 | 4.71 |
| IShellExecuteHook | 能够钩住通过 ShellExecute() 或 ShellExecuteEx() 导出的所有新进程的启动。 | 所有版本 |
|  |  |  |

显示详细信息

**杂项接口**

          覆盖 Shell 编程特殊领域的其它接口统称为杂项接口，如：‘我的公文包’，通用对话框，和任务条等。

|  |  |  |
| --- | --- | --- |
| 接口 | 描述 | 版本 |
| INotifyReplica ,  IReconcilableObject ,  IReconcileInitiator | 所有这些接口都涉及到文件调整过程。最终都产生同一个文档的更新版本。 | 所有版本 |
| ICommDlgBrowser | 当客户文件夹嵌入到通用对话框中时，提供特殊的浏览行为。 | 所有版本 |
| ITaskbarList | 允许在系统任务条中加入新的按钮。 | 4.71 |
|  |  |  |

**为什么又有API，又有COM**

          现在我们已经看到了 Windows Shell 所有的功能，需要花费一点时间才能给出 API 函数和 COM 接口的作用。本质上，整个 Shell 功能可以划分成两个领域，基本功能和扩展功能，从这个观点分析，就很容易区分哪一种方法属于哪一个领域的了。

          现在，大多数由 API 调用提供的功能可以看作调用“ Shell ”的伪对象的‘方法’。这个伪对象允许你移动或拷贝文件，或浏览文件夹。你也可以恢复给定文档的信息，等等。对象模型的头一个特性就是从描述它本身开始的。

          换句话说， Windows 初始是由纯 C 设计的，从没有被考虑过以面向对象的概念进行设计。因此，所有的基本功能都通过直接的 API 调用给出也就不奇怪了。

         COM 技术允许写出部件模块，然后通过选择暴露它的接口来使用它们。使用接口很容易聚集相关函数并提供对给定对象的访问。站在 Shell 的立场上看， COM 接口就是封装的 API 调用—你可以在 ITastbarList 接口中看到，这是头一个 COM 而不是 API 调用暴露的系统部件编程接口的例子。

          这种模式的另一个例子是我们上面提到的钩子接口。在 Win32 SDK 中全部钩子都是通过回调函数而不是 COM 接口编程的。换句话说， Shell 编程接口包含有钩子，这就要求你编写并适当注册一个 COM 服务器来实现。实际上，差别不是很大，但在体系上，他们就不同了。

          有一股变革之风从 Windows Shell 吹来， COM 就是他的源泉。在已经提到的例子中，可以看到，所有 COM 接口都被用于扩展探测器的行为。由于探测器需要设计进程内服务器，因此，他们的技术是平行的， API 调用和 COM 接口技术同样重要。它们可以被看作为一个硬币的两面 ( 这个硬币就是 Shell) ，但是它们确实是有差别的。

**活动桌面有什么变化**

          活动桌面外壳的更新带来一些新的特征，并且使 Windows Shell 产生了几个方面的变化。它在任何可能的地方都鼓励使用 HTML ，引进了 Web 观察的概念，文件夹的客户化，脚本能力，简化而有效的对象模型，以及大把的新函数和 COM 接口。

          上面列出的最后一项应特别引起注意，例如，我们现在有了一个非常原始的 Shell 对象模型，通过 COM ，暴露了一些 Shell 的功能。在大脑中记住这些对于程序员来说是重要的。到目前为止，这个模型还不完善，没有你所期望的灵活性，但这是重要的第一步。

          抛开 Shell API 的变化不谈，活动桌面表现了桌面结构和文件夹的值得注意的演化。特别是：

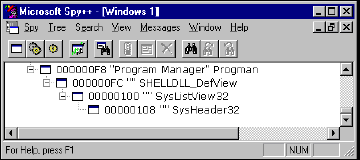
                   Shell 观察对象

                    任务条结构

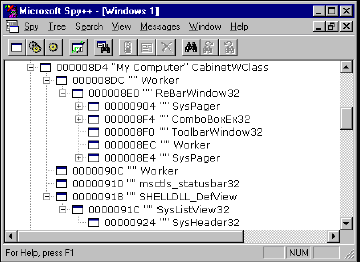
此外还有 Shell 观察对象的增强，以此我们可以在文件夹层上执行脚本代码，以及使用动态 HTML 和脚本程序。

**新的Shell观察对象**

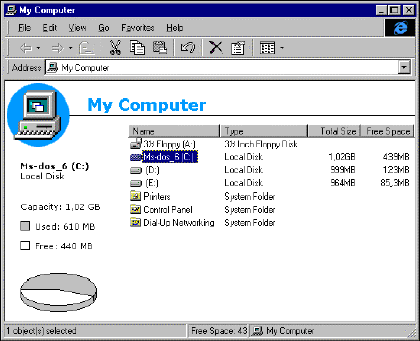
          最初， Shell 观察对象是通过窗口栈顶的 SHELLDLL\_DefView 类来实现和表示的。在第一章中你也已经看到了：



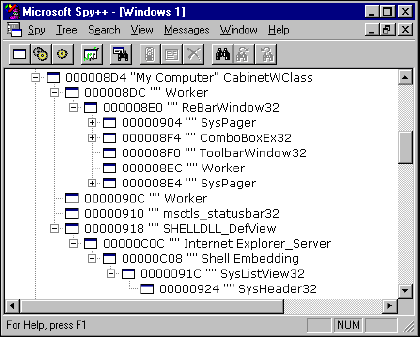
这个截图显示了桌面的观察对象，然而，实际上它对任何文件夹都是一样的。例如下面的图像显示了‘我的计算机’文件夹的窗口堆栈情况：



这里所看到的绝大多数窗口一起合作形成文件夹窗口的框架（窗框，组合框，工具条等）。提供显示实际文件夹内容（即 Shell 观察对象）的总是窗口的 SHELLDLL\_DefView 类与他的后代，列表观察。然而对活动桌面，有另外一种观察对象，这个对象还包含有对 HTML 和脚本的支持，称之为 Web观察 ，并且可以使用文件夹的‘观察 | 作为 Web 页’菜单功能打开和关闭。下面就是在 Web 观察打开时，‘我的计算机’窗口所看到的。



          文件夹的内容以基于 HTML 的模版方式显示，其中的列表观察包容了文件对象详细信息的控件。对应的窗口堆栈为：



应立即注意到的最大差别就是窗口类 Internet Explorer\_Server ，它有一个子窗口类 Shell Embedding ，所有这些形成了通过 WebBrowser 控件显示输出的窗口，而 Shell Embedding 则是一个封装了文件列表控件的列表观察窗口。

*WebBrowser是一个IE3.0以上版本使用的ActiveX控件，用以显示他们的内容：HTML文件，GIF和JPEG图像，和活动文档。*

          概括地讲，如果 Web 观察打开，则

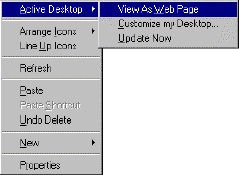
                    文件夹看上去是由 WebBrowser 控件显示的 HTML 页面

                   HTML 页面从 HTML 模版生成，它在必要时可以被客户化。

                    包含文件的列表观察被嵌入到 ActiveX 控件中，并一起并入到 HTML 页面中。

Web 观察也可以在客户文件夹上打开，但是，这种情况下，封装文件夹的命名空间扩展就需要实现特殊附加的接口。

在桌面上事情也是一样的，你可以通过关联菜单打开和关闭 Web 观察：



当这个观察活动时，桌面的观察对象也使用 WebBrowser 控件显示桌面内容。桌面的图标在不同的比背景更高层上绘制，尽管这种‘图标层’在以前的活动桌面上也存在， Web 观察还是加入了一些 HTML ‘墙纸’的东西，其内容总是显示在图标的下面。

*客户化文件加*

          当 Web 观察打开的时候，你所访问的文件夹使用 HTML 模版显示。有一个标准文件夹模版文件 Folder.htt 存储在 Windows 的 Web 子目录下，在没有指定其他的模板之前，它是默认的。如果想要学习它的源码，要注意，他是一个隐藏文件，所以，在打开‘显示所有文件’的设置之前，你不能看到它。

通过右击文件夹，打开一个菜单，如图：

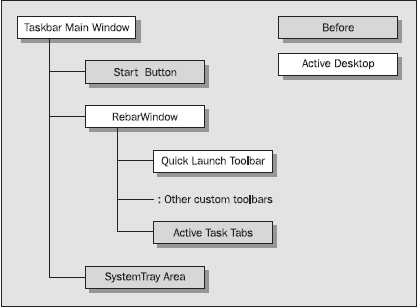


选择‘客户化文件夹’ …  允许你直接运行编辑大师编辑 folder.htt 文件的内容。更精确地说，你实际所编辑的就是在指定文件夹中由编辑大师最初建立的模版。只要你需要，完全可以通过简单地编辑这个 HTML 文件改变文件夹的外观。尽管这个 .htt 扩展是一个完美的 HTML 文件，只要你想，你还可以删除或置换这个文件的列表控件，仅仅显示你想要用户看到的信息。

          由于文件夹模版是一个通过 IE 的 WebBrowser 可观察的 HTML 文件，所以，你可以采用所有 XML 的特征，数据绑定，动态 HTML ，和脚本功能，以及改变一个简单的文件夹使其看起来像一个应用程序。这样的客户化也相当类似于原始的命名空间扩展。

**新任务条的布局**

          与观察对象的改变一样，活动桌面的任务条布局也发生了一定的变化。下图中给出了预期的概念，以及新老结构的比较：



**小结**

          这一章，我们讨论了：

                    从功能上对 Windows Shell 的 API 函数和 COM 接口进行分组

                    这本书的章节布局

                    概述了 Shell 结构及其对象

随着讨论的深入，我们总是试图使 Shell 编程接口的结构清晰印在你的脑海中，因而，在下一章中，将包含那些最关键的 API 函数的详细说明。描述代码主要使用 C++  调用 SDK 函数。