FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2004 年 10 月刊

从 XSouce 中淘金，第一部分 — Doug Hennig 著 Fbilo 译

Page.3

Visual FoxPro 销售的产品还带有大多数“Xbase”工具的源代码，其中包括类浏览器、代码引用、Toolbox 和 Task Pane 等等。阅读由顶尖的 VFP 专家们编写的源代码经常可以学习到新的、强大的编程技术。在这个系列的头一篇文章中，Doug Hennig 将会带着我们阅读在 XSource 中的大量文件，并向大家演示可以用在自己的应用程序中的好主意和代码，以将自定义设置保存到一个资源文件中和建立面向对象的菜单。

在 VFP 9.0 中用 FFC 类库来使用 GDI+，第三部分 —Walter Nicholls 著 Fbilo 译

Page.14

在 8 月初，Microsoft 给可下载的 Visual FoxPro 9.0 测试版增加了一些文件，包括一套新的用于 GDI+图形工作的基础类库（请查看位于 http://msdn.com/vfoxpro 的微软 VFP 网站以寻找下载信息）。在这个系列文章的第三部分中，Walter Nicholls 提供了一个使用由 _GDIPLUS.VCX 提供的图像和位图类技术的“烹调手册”。

来自 VFP 开发组的 Tips — 微软 VFP 开发团队 著 LQL.NET 译

Page.23

每个月，VFP 开发团队成员都会给你带来有趣的 VFP 技术 TIPS。这个月带给你的是如何让 BINDEVENT 执行 WINDOWS 系统消息。注意：这个功能不能在 VFP9 公开测试版上运行，只有 VFP9 正式版才有这个功能。

拆分工具（第一部分） — Lauren Clarke & Randy Pearson 著 CY 译

Page.26

这是文本处理技术和策略系列文章的第四部分，也是最后部分。上月 Lauren Clarke 和 Randy Pearson 已经开发了一个新的强大的文本处理函数，它无缝地组合了强大的规则表达式和 VFP 内置函数和自定义函数。在这个文章里，他们将通过组装基于文本处理的类群的模型，为开发拆分工具创建一个框架。（注意：如果你刚刚加入我们，请不要担心。先前文章里的所有示例代码都包含在这篇文章的下载文件里。）

“哚、哚、哚”有人在家吗？（第二部分） — Andy Kramek & Macia Akins 著 Fbilo 译

Page.37

上个月，Andy Kramek 和 Marcia Akins 推定了实现应用程序安全性最佳的途径是采用一种“基于角色”的模型。他们将“角色”定义为一个标志符，它代表一种可以被应用程序中的一个或多个用户使用的特定的访问模式。在这个月的专栏中他们选定的任务，是设计和实现一个类来处理这种多层基于角色的安全性的管理任务。

使用 Inno Setup 安装程序 — Rick Borup 著 YASUR 译

Page.46

在这个系列的第一部分（见 2004 7 月刊），Rick Borup 向你推荐了 Inno Setup 并就配置 VFP 应用程序讲述了基础知识。在第二部分里，Rick 创建了一个样例，向你演示如何用 Inno Setup 脚本来增强她的功能，包括如果安装数据库文件到应用程序以外的路径。

从 XSouce 中淘金，第一部分

原著: Doug Hennig

翻译: Fbilo

在 Visual FoxPro 销售的产品还带有大多数“Xbase”工具的源代码，其中包括类浏览器、代码引用、Toolbox 和 Task Pane 等等。阅读由顶尖的 VFP 专家们编写的源代码经常可以学习到新的、强大的编程技术。在这个系列的头一篇文章中，Doug Hennig 将会带着我们阅读在 XSource 中的大量文件，并向大家演示可以用在自己的应用程序中的好主意和代码，以将自定义设置保存到一个资源文件中和建立面向对象的菜单。

从 1998 年 12 月开始，我在 Foxtalk 上写了一个由三个部分组成的系列文章“挖掘 FFC 基础类库中的金子”，讨论了从 VFP 主目录下的 FFC (FoxPro 基础类库)子目录中的源代码中寻找有用的内容。然而，VFP 还带有更多的源代码！

事实上，VFP 自带的许多工具都是用 VFP 而不是 C++（用于开发 VFP 的语言）写就的。我们通常把这些工具称为“Xbase”工具，以将它们与用 C++ 写的内部部件相区别。部分工具可以从“工具”菜单来打开，例如类浏览器、IntelliSense 管理器、Toolbox、Task Pane 管理器、以及报表向导。其它的一些则要根据所处的相应位置来打开，例如参照完整性生成器、CursorAdapter 生成器、还有 VFP 9 中的 ReportBuilder.APP，它提供了报表设计器的新对话框。

这篇文章是这个探索 XSource 系列文章中的第一篇，讲的是我们可以用在自己的应用程序中的有趣的技术和源代码。我并不准备在这个系列的文章中探索 XSource 中的每样东西，那会需要整本书来讨论。我将讨论的是一些我认为很有用的东西，包括：

- Outlook 风格的 Toolbox 是怎样工作的；
- 怎样使用 FOXUSER 资源文件以代替 INI 文件或者 Windows 注册表；
- 在 VFP 表单上显示 AVI 文件；
- 建立面对对象的快捷菜单；

寻找 XSource

从前几个版本开始，VFP 销售时就带上了几乎所有 Xbase 工具的源代码（Beautify.APP 是个例外，它提供了工具菜单下的“修饰”菜单项的功能）。由于代码太多，所以被打包在一个 ZIP 文件中：VFP 主目录下 Tools\XSource 目录下的 XSource.ZIP。把这个文件解压后会在 Tools\XSource 目录下建立一个叫做 VFPSource 的子目录，这个目录里面就包含有 Xbase 工具的源代码。表 1 中列出了 XSource 中各个路径的用途以及由其中的源代码生成的 APP 文件所在的位置。

表 1、由 XSource 源代码生成的各个应用程序的用途和位置

XSource 目录	用途	位置
AddLabel	AddLabel 应用程序	Tools\AddLabel\AddLabel.APP
Browser	类浏览器和组件管理库	Browser.APP 和 Gallery.APP
Builders	主生成器程序（各个生成器 APP 的代理）	Builder.APP 和 Wizard.APP
	和主向导程序（各个向导的代理）	
Builders\Builders	用于生成器中的常用文件	
Builders\CmdBldr	命令按钮组生成器	Wizards\CmdBldr.APP
Builders\EditBldr	EditBox 生成器	Wizards\EditBldr.APP
Builders\FormBldr	表单生成器	Wizards\FormBldr.APP
Builders\GridBldr	Grid 生成器	Wizards\GridBldr.APP
Builders\ListBldr	ListBox 生成器	Wizards\ListBldr.APP
Builders\RIBuilder	参照完整性生成器	Wizards\RIBuildr.APP
Builders\StylBldR	样式生成器	Wizards\StylBldr.APP
Convert	转换器（将文件转换到新版本）	Convert.APP
Coverage	代码范围分析器	Coverage.APP
DataExplorer(VFP 9)	数据浏览器（在 Task Pane 管理器中可用）	DataExplorer.APP
EnvMgr	环境管理器（在 Task Pane 管理器中可用）	EnvMgr.APP
FoxCode	IntelliSense 管理器	FoxCode.APP
FoxRef	代码引用	FoxRef.APP
oBrowser	对象浏览器	ObjectBrowser.APP
ReportBuilder(VFP 9)	报表设计器的对话框和其它特性	ReportBuilder.APP
ReportOutput(VFP 9)	Report listeners(例如 XML 和 HTML)	ReportOutput.APP
ReportPreview(VFP 9)	新的报表预览对话框	ReportPreview.APP
TaskList	任务列表	TaskList.APP
TaskPane	Task Pane 管理器	TaskPane.APP
ToolBox	ToolBox	ToolBox.APP

VFPClean	恢复 VFP 用户文件和注册表设置的工具	VFPClean.APP
WebService	Web Services 生成器	Wizards\WebService.APP
Wizards\Automate	用于图形、邮件合并和交叉表向导的通用文件	-
Wizards\DEBuilder	CursorAdapter 和数据环境生成器	Wizards\DEBuilder.APP
Wizards\WZApp	应用程序向导	Wizards\WZApp.APP
Wizards\WZCommon	用于向导和生成器的通用文件	-
Wizards\WZForm	表单向导	Wizards\WZForm.APP
Wizards\WZFoxDoc	文档向导	Wizards\WZFoxDoc.APP
Wizards\WZGraph	图形向导	Wizards\WZGraph.APP
Wizards\WZImport	导入向导	Wizards\WZimport.APP
Wizards\WZIntNet	Web 搜索页向导	不再作为工具包含
Wizards\WZMail	邮件合并向导	Wizards\WZMail.APP
Wizards\WZPivot	交叉表向导	Wizards\WZPivot.APP
Wizards\WZQuery	查询向导	Wizards\WZQuery.APP
Wizards\WZReport	报表和标签向导	Wizards\WZReport.APP
Wizards\WZTable	表和数据库向导	Wizards\WZTable.APP 和 Wizards\WZDBC.APP
Wizards\WZUpsize	升迁向导	Wizards\WZUpsize.APP
Wizards\WZWeb	Web 发布向导	Wizards\WZWeb.APP

注意：Tools 目录的其它一些子目录——Analyzer、CPZero、Filer、GenDBC、HexEdit、MSAA 和 Test——中包含有在这些目录中工具的源代码。而这些工具是无法从 VFP 的菜单中访问的，它们是独立的工具，可以通过运行在相应目录中的文件来访问它们。在这个系列文章中我不会讨论它们。

XSource 代码的好处

拥有这些工具的源代码有什么好处？至少有两样：

- 如果这些工具不能满足你的需要，你可以自己修改它并生成一个新的 APP 文件来代替。
- 我一向喜欢利用别人已经完成的佳作，而 VFP 的 XBase 工具则是由一些 VFP 世界最出色的明星写就的（其中不少工具是由非微软雇员写就的，例如 Markus Egger，他写出了对象浏览器）。在某些工具中使用了非常有趣的技术，我在源代码的探索中学到了许多东西。更重要的是，其中某些类、图像和 PRG 在其它应用程序中非常有用。

如果你决定要在自己的应用程序中使用 XSource 源代码中的某些部分，请注意某些类是位于不同

VCX 文件中多个类的子类，而某些 VCX 中则有着大量的类。即使你只需要其中的一个，当你编译你的项目的时候，所有相关的 VCX、PRG 和图形文件都会自动被包含进去。

你可以做的是去适应它（唯一的负面效果是 EXE 文件的体积增大了，不过反正目前磁盘空间便宜）或者把你想要的类从 VCX 中拔出来放到你自己的小 VCX 中去。象类浏览器和白光计算机的 HackCX（以前由 Geeks 和 Gurus 写的一个工具，我曾在 2003 年 12 月的文章《Tools for and by Geeks and Gurus》中讨论过，强烈推荐，详情请见：www.whitelightcomputing.com）这样的工具可以帮上忙，不过还是会有一些杂务是需要你自己去手工修正的。

注意：VFP 的帮助文件在重用来自 XSouce 的代码的主题上有一点模糊，但 Microsoft 最近表示，直接使用或者轻微修改后使用 Xbase 的源代码是可能的，甚至我们可以使用整个类库——如果这些类库被包含在项目中并且被编译进应用程序内的话。这种情况类似于使用来自于 Samples 目录中的源代码和类库，后者在帮助主题“Visual FoxPro 功能和文件的发布和限制”中有专门的论述。

一个丰富的图像资源

当我搜索用于按钮的图片或者用于表单的图标的时候，其中一个我会搜索的位置是 XSource，尤其是那些新的 Xbase 工具（在 VFP 8 或者更新的版本中增加的工具）的目录。我经常会找到我正好想要的东西，或者某些近似的东西，我会用象画图这样的工具对它稍做改动。图 1 是我在 XSource 目录中找到的部分有用的图片。

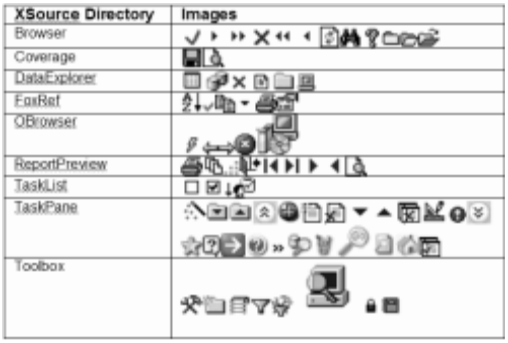

图 1、在 Xsource 目录中一些有用的图像

在资源文件中保存设置

一个专业的应用程序会保存当前的设置，这样，当用户下一次运行的时候，这些同样的设置就会再次被使用。例如，这样的情况会很不错：一个应用程序会“记住”表单的大小和位置、你上次导入某个文件时所用的目录、上次输出一个文件时用的是什么文件类型，等等。有几种应用程序可以用来保存这

样的设置的位置，包括 INI 文件、Windows 注册表、一个 XML 文件、或者一个表。

VFP 交互式开发环境（IDE）在记住一些设置方面做得相当好，比如在代码文件的大小、位置、选中的文本，在一个表单或者类中最近一次打开的代码窗口等等。它把这些设置保存在你的资源文件中，通常是你的 HOME(7) 目录下的 FoxUser.DBF。用这个资源文件来保存你自己的设置应该不错吧？

这就是在 FoxResource.PRG 中的 FoxResource 类的目标。这个类可以在包括 Toolbox 这样的多个 XSource 目录中找到，它提供了读取和写入到一个资源文件的方法。资源文件并非一定要是当前的那个（指用 SYS(2005)返回的那个）；你可以在 ResourceFile 属性中指定要使用的文件的名称。

一个资源文件应该包含以下字段：

- TYPE——这里记录字段的类型。大多数记录的这个字段中的内容会是“PREFW”，但你可以通过设置 FoxResource 类的 ResourceType 属性来使用你需要的任何值（这个属性的默认值是“PREFW”）。
- ID——当前记录的 ID。可以是你想要的任何值，但它应该是唯一的。
- NAME——当前记录的名称。如果需要的话，可以保留为空。
- READONLY——如果记录不应被改动则为 .T.。
- CKVAL——DATA 字段的检查和（checksum）。
- DATA——设置。FoxResource 用 SAVE TO MEMO DATA 来把你的设置保存为一个数组。
- UPDATED——记录上次被更新时的日期。

FoxResource 类使用 Load 方法来从资源文件的一条记录中加载一系列的设置。给这个方法传递 ID 和 Name（与文件中的 ID 和 Name 字段匹配），它将把这些设置加载到它内部的“名称—值”数据项集合中去。设置加载好以后，你就可以通过给 Get 方法传递设置的名称来返回你所需要的设置的值。

不用担心数据类型的问题；由于设置是作为一个数组被保存到资源文件中去的，返回的值也会保持当它们被保存时同样的数据类型。与使用 INI 文件的方法相比，这是一个很好的改动，因为在 INI 文件里面一切都是字符串。如果你要求返回的设置不存在，则 Get 方法返回的是 .NULL.，所以你也许应该先用 NVL() 对返回的值进行一下检查。

这里是一个如果 Top 设置不存在则忽略它的例子：

```
This.Top = nvl(oFoxResource.Get('Top'), This.Top)
```

你可以通过使用 **OptionExists** 方法来检查一个设置是否存在。向这个方法传递传递设置的名称，如果存在的话则它会返回一个 **.T.**。

要保存一个设置请使用 **Set** 方法，给该方法传递设置的名称和值。当你已经保存了一系列设置以后，你可以使用 **Save** 方法来将这些设置保存到资源文件中去。这个方法象 **Load** 方法一样需要 **ID** 和 **Name** 参数。**Save** 还同时更新 **UPDATED** 和 **CKVAL** 字段，并且它支持资源文件中的只读记录，如果当前记录的 **READONLY** 字段为 **.T.**，则它会拒绝更新该记录。

还有一些用处较少的方法。**Clear** 清除集合，**GetData** 返回指定记录的备注字段 **DATA** 的内容，**SaveTo** 保存到一个指定的备注字段而不是默认的备注字段 **DATA**，**RestoreFrom** 从指定的备注字段中恢复出内容。

在这个月的下载文件中的 **TestMenu.SCX** 演示了 **FoxResource** 的用法。它的 **Init** 方法会建立 **FoxResource** 的实例，为名称为 **TESTMENU** 的设置恢复设置，调用 **SetFormPosition** 来恢复表单的大小和位置，然后使用 **FoxResource** 的 **Get** 方法来恢复记录指针和过滤设置。

```
local lnRecno, ;
 lcFilter


with This
     .oResourceOptions = newobject('FoxResource', ;
          home() + 'Tools\XSource\VFPSource\Toolbox\' + ;
          'FoxResource.prg')
     .oResourceOptions.Load('TESTMENU')
     .SetFormPosition()
     lnRecno = nvl(.oResourceOptions.Get('RecNo'), 0)

     if between(lnRecno, 1, reccount())
          go lnRecno
     endif between(lnRecno, 1, reccount())

     lcFilter = nvl(.oResourceOptions.Get('Filter'), '')
     .SetFilter(lcFilter)
endwith
```

这里的 **SetFormPosition** 方法的代码与在 **Toolbox** 目录中的 **ToolboxCtrls.VCX** 类库中的 **cFoxForm** 类上同名的方法的代码是一样的。这些代码所做的远不止是恢复表单的 **Top**、**Left**、**Width** 和 **Height** 而已，它还能确保这些值不会导致表单脱离屏幕的范围，例如如果表单原来被放在远离一个高解析率系统中屏幕的右边的位置，而现在却在一个低解析率系统中被打开的情况。

```
LOCAL nTop, nLeft, nWidth, nHeight

IF !ISNULL(THIS.oResourceOptions)
m.nHeight = THIS.oResourceOptions.Get("HEIGHT")
IF VARTYPE(m.nHeight) == 'N' AND m.nHeight >= 0
    THIS.Height = m.nHeight
ENDIF

m.nWidth = THIS.oResourceOptions.Get("WIDTH")
IF VARTYPE(m.nWidth) == 'N' AND m.nWidth >= 0
    THIS.Width = m.nWidth
ENDIF

m.nTop = THIS.oResourceOptions.Get("TOP")
IF VARTYPE(m.nTop) == 'N'
    * 确保可见
    IF !THIS.Desktop
        IF m.nTop > _SCREEN.Height
            m.nTop = MAX(_SCREEN.Height - THIS.Height, 0)
        ELSE
            IF m.nTop + THIS.Height < 0
                m.nTop = 0
            ENDIF
        ENDIF
    ENDIF
    THIS.Top = m.nTop
ENDIF

m.nLeft = THIS.oResourceOptions.Get("LEFT")
IF VARTYPE(m.nLeft) == 'N'
```

```
        * 确保可见
        IF !THIS.Desktop
            IF m.nLeft > _SCREEN.Width
                m.nLeft = MAX(_SCREEN.Width - THIS.Width, 0)
            ELSE
                IF m.nLeft + THIS.Width < 0
                    m.nLeft = 0
                ENDIF
            ENDIF
        ENDIF
        THIS.Left = m.nLeft
    ENDIF
ENDIF
```

Destory 方法把我们想要记录的设置保存起来，然后将它们写入到资源文件中的 **TESTMENU** 设置中去。

```
with This
    .oResourceOptions.Set('Left', .Left)
    .oResourceOptions.Set('Top', .Top)
    .oResourceOptions.Set('Height', .Height)
    .oResourceOptions.Set('Width', .Width)
    .oResourceOptions.Set('Filter', .cFilterName)
    .oResourceOptions.Set('RecNo', recno())
    .oResourceOptions.Save('TESTMENU')
endwith
```

要测试一下这个程序，请运行 **TestMenu.SCX**，在屏幕上四处移动一下表单。在表单上单击鼠标右键，从快捷菜单的 **Set Filter** 子菜单中选择一个过滤设置，然后单击鼠标右键，选择几次 **Next**（后面我会谈到快捷菜单是怎样建立的）。关闭表单然后重新运行它，这个表单将以与你关闭它的时候同样的位置、过滤设置、当前记录来显示。

面对对象的快捷菜单

VFP 的菜单系统是少数仍然保持为面向过程而不是面向对象的功能之一。虽然你可以使用各种与菜

单相关的命令（例如 **DEFINE POPUP** 和 **DEFINE BAR**）来建立你自己的菜单，但几乎没有人这么做，因为 **VFP** 已经包含了一个菜单设计器，它让我们可以可视化的建立菜单。不过，使用由菜单设计器生成的菜单之最大的缺点，是不能在运行时修改它们。那就意味着你不能轻松的定位它们，或者在某些特定的条件下显示或者隐藏特定的菜单项。

ContextMenu 能解决问题！这个定义在 **Toolbox** 目录下的 **FoxMenu.PRG** 文件中的类让你可以动态建立一个菜单而不需要写丑陋的 **DEFINE POPUP** 和 **DEFINE BAR** 命令。简单的建立这个类的实例，根据需要设置菜单项（这里可以使用条件码来定位菜单或者决定要包含哪些菜单项），然后告诉它去显示菜单。如果需要的话，你甚至可以以数据来驱动菜单。

要向菜单中添加菜单项，请调用 **AddMenu** 方法。这个方法接受六个参数（后面的四个是可选的）：菜单项的标题、当菜单被选中的时候要被执行的一个表达式、用于菜单项的一个图像文件的名称，如果为 **.T.** 则菜单的选中标记被打开，如果为 **.T.** 则菜单可用，如果为 **.T.** 则菜单以粗体显示。

由于 **VFP** 的菜单系统存在于它的对象系统之外，在要被执行的表达式中使用 **This** 或者 **Thisform** 这样的对象引用是无法工作的，所以要把对表单或者对象的引用放入到一个私有变量（**PRIVATE**）中去，并在表达式中使用这个变量。在下载文件中的示例代码演示了这种情况。

AddMenu 会返回对一个包含着关于菜单项属性（标题、图片、选中标记、操作代码、是否可用、以及粗体）的对象引用。更有趣的是，它还包含一个 **SubMenu** 属性，该属性包含着对另一个 **ContextMenu** 对象的引用。这样一来，如果要为一个菜单项建立一个子菜单，只要这个菜单项的 **SubMenu** 对象的 **AddMenu** 方法就行了。

要显示这个菜单，请调用 **Show** 方法。在 **VFP 8** 中，你可以选择传递两个参数：菜单要显示位置的行和列坐标。**VFP 9** 中增加了第三个可选参数：菜单所显示于的表单的名称（在这种情况下，**ContextMenu** 使用 **DEFINE POPUP** 命令的 **IN WINDOW** 子句）。如果你不带参数的调用 **Show**，那么菜单就不会出现在准确的位置上。所以，向 **Show** 传递 **MROW('')** 和 **MCOL('')**并忽略表单的名称、或者忽略行和列而为表单的名称传递 **This.Name**。

在 **VFP 9** 测试版的这个类中有一个 **Bug**。在下面这行代码的 **“m.nCol”**中漏了一个分号。请自己把它添加到你的 **FoxMenu.PRG** 中去（注意：这个 **Bug** 在 **VFP 9** 的正式版中应该会被修正了）。

```
DEFINE POPUP (m.cMenuName) SHORTCUT ;
    RELATIVE FROM m.nRow, m.nCol
    IN WINDOW (m.cFormName)
```

ContextMenu 有几个属性。**MenuBarCount** 包含着菜单中菜单项的数量。**ShowInScreen** 指定菜单要被显示在 **_SCREEN** 中，但所有引用这个属性的代码都被注释掉了，所以你可以忽略它。

图 2

这里是从 **TestMenu.SCX**（见图 **2**）的 **ShowMenu** 方法中取来的一个例子。它建立 **ContextMenu** 类的实例，然后调用 **AddMenu** 方法来建立要显示的各个菜单项。这些代码建立向导菜单项（**First**、**Next**、**Previous** 和 **Last**），传递图片和 **enabled** 参数，这样，菜单项就拥有了图片并只在合适的条件下才可用。**Set Filter** 菜单项有一个子菜单，它包含着可以被设置的各种过滤条件，匹配当前过滤条件的菜单项上会显示选中标记。

```
local loMenu, ;
 loMenuItem
private poForm

* 建立菜单对象
loMenu = newobject('ContextMenu', ;
     home() + ;
     'Tools\XSource\VFPSource\Toolbox\FoxMenu.prg')

* 建立对这个表单的私有变量引用
* 以便菜单操作生效
poForm = This

* 定义菜单项
loMenu.AddMenu('First', 'poForm.FirstRecord()', ;
     'frsrec_s.bmp', .F., not This.lFirstRecord)
loMenu.AddMenu('Next', 'poForm.NextRecord()', ;
     'nxtrec_s.bmp', .F., not This.lLastRecord)
loMenu.AddMenu('Previous', 'poForm.PreviousRecord()', ;
     'prvrec_s.bmp', .F., not This.lFirstRecord)
```

```
loMenu.AddMenu('Last', 'poForm.LastRecord()', ;
      'lstrec_s.bmp', .F., not This.lLastRecord)

loMenu.AddMenu('\-')
loMenuItem = loMenu.AddMenu('Set Filter')

loMenuItem.SubMenu.AddMenu('North American Customers', ;
      'poForm.SetFilter("NA")', '', This.cFilterName = 'NA')
loMenuItem.SubMenu.AddMenu('European Customers', ;
      'poForm.SetFilter("EU")', '', This.cFilterName = 'EU')
loMenuItem.SubMenu.AddMenu('South American Customers', ;
      'poForm.SetFilter("SA")', '', This.cFilterName = 'SA')

loMenuItem.SubMenu.AddMenu('\-')
loMenuItem.SubMenu.AddMenu('Clear Filter', ;
      'poForm.SetFilter("")', '', empty(This.cFilterName))

loMenu.AddMenu('\-')
loMenu.AddMenu('Close', 'poForm.Release()')

* 显示菜单
loMenu.Show(, , This.Name)
```

总结

XSource 并不只是 **VFP** 自带的 **Xbase** 工具的源代码而已，它还是一个丰富的思想、资源和可重用代码的宝藏。下个月，我将谈谈在一个 **VFP** 表单上显示 **Video** 文件、在一个很长的进程中显示一个进程栏、给你的应用程序添加一个 **OutLook** 式的 **Listbar** 的技术和代码。

下载：410HENNIG.ZIP

在 VFP 9.0 中用 FFC 类库来使用 GDI+，第三部分

原著：Walter Nicholls

翻译：Fbilo

在 8 月初，Microsoft 给可下载的 Visual FoxPro 9.0 测试版增加了一些文件，包括一套新的用于 GDI+图形工作的基础类库（请查看位于 http://msdn.com/vfoxpro 的微软 VFP 网站以寻找下载信息）。在这个系列文章的第三部分中，Walter Nicholls 提供了一个使用由 _GDIPLUS.VCX 提供的图像和位图类技术的“烹调手册”。

这个系列已经这么久了，我所讲述的也已经覆盖了用 _GDIPLUS.VCX 类库来画图、以及使用这些类在一个 VFP 表单和报表上显示一个饼状图的基础。到现在为止，有两个类我只是简要的提了下而已。

GpImage 和 GpBitmap 类

与前面讲过的类不同，前面那些类提供的是各种对 GDI+ 进行操作来渲染表面的功能，而这两个则是提供完成的图片。

从技术上来说，GpImage 是一个抽象类：你不能直接建立它的实例，但是从它可以派生出许多图形类（比如 GpBitmap）。虽然如此，它还是有自己的一项专长。你可以建立一个 GpImage 对象，并从一个文件来加载一个图片，而它能够适用于任何类型的图片（位图，或者矢量图）。表 1 中展示的是 GpImage 的属性和方法。

表 1、GpImage 类的属性和方法

属性/方法名称	说明
CreateFromFile(sFilename[,lUseICM]	从磁盘上读取一个文件
GetBounds(units)	取得矩形的边界，以被要求的单位来度量（返回一个 GpRectangle）
GetEncoderCLSID(sMimeType)	取得给定图像类型的已注册编码器的 CLSID
GetDecoderCLSID(sMimeType)	取得给定图像类型的已注册解码器的 CLSID
GetThumbnailImage(width,height)	建立一个指定大小位图的缩略图
SaveToFile(sFilename,sMimeType[,options])	把一个图像保存为一个磁盘文件
RotateFlip(nRotateFileType)	旋转或者翻转图像

ImageWidth,ImageHeight	以一个图形自己的单位表示的图像大小
HorizontalResolution,VerticalResolution	以每英寸多少象素表示的解析率
RawFormat	图像格式的 GUID
Flags	图像属性标记
Init(nGDIPlusImage)	根据 GDI+ 图像句柄建立
Init(sFileName[,lUseCM])	直接从一个磁盘文件建立

GpBitmap 用于以象素数据显示的图像。与它的父类相比，还多了几个属性和方法，如表 2 所示。

表 2、GpBitmap 类增加的属性和方法

属性/方法名称	说明
Create(width,height[,pixelformat])	以指定的大小和象素格式建立一个位图（位深）。
CreateFromFile(sFilename[,lUseICM])	从磁盘上读取一个文件——注意这个文件并非必须是一个位图，但它将被当作是一个位图来读取
CreateFromGraphics(graphics,width,height)	建立自一个 GDI+ 图形对象
GetPixel(x,y),SetPixel(x,y,color)	读取和写入单个象素
Init(width,height[,pixelformat])	以指定的大小和象素格式建立
PixelFormat	该位图对象的象素格式

将一个图像文件加载到内存里

下面的代码使用了一个包含在本期杂志下载文件中的图像文件：zoephoto.jpg(我不得不在这个系列的文章放入至少一张我的孩子之一的照片！)。

```
oPhoto = newobject('GpImage', 'ffc/_gdiplus.vcx')
oPhoto.CreateFromFile( 'zoephoto.jpg' )
```

在这里，使用 GpImage 而不是一个 GpBitmap 对象是因为它们有一个微妙的区别。如果你正在加载一个矢量格式的图形文件——比如一个 Windows 位元文件（WMF 格式）——那么它内部的矢量格式的独立解析率被保留。如果你建立了一个 GpBitmap，却用它来加载一个矢量文件，那么将在内存里被转换成光栅格式，并且不能平滑缩放。不过在这个示例中问题不大，因为 JPEG 本身就是光栅格式的。

建立缩略图——方法 1

建立缩略图最简单的办法是使用 **GetThumbnailImage()**方法。如果该图像文件自带有一个嵌入的缩略图，该方法就会使用它；否则，它将重新对主图像进行调整，然后返回一个指定格式的位图。例如，要从前面加载的照片来建立一个 **80** 象素宽、**100** 象素高的缩略图：

```
oThumbnail = oPhoto.GetThumbnailImage( 80, 100 )
```

虽然这种办法简单的只要执行一个方法调用就行了，但是它不够灵活。新的位图总是会填满指定的区域（某些时候会让图像变得扭曲不堪），而你除了象素格式以外不能指定其它的属性。稍后我将向你展示怎样来增强它。

保存为一个文件

要保存为一个文件，你应该以一个文件名和希望的格式来调用 **SaveToFile** 方法。

```
oThumbnail.SaveToFile( 'zoethumb1.jpg', 'image/jpeg' )
```

对于某些图像格式，你可以指定额外的选项参数。例如，对于 **JPEG** 格式，你可以调整压缩率：

```
oThumbnail.SaveToFile( 'zoethumb2.jpg', ;
 'image/jpeg', 'quality=100' )
```

表 **3** 中是部分常用的图像格式和可选的参数（完整的列表取决于安装在你系统上的编码器和解码器）。

表 **3**、常用图像格式

MIME 类型	扩展名	SaveToFile() 选项参数
image/jpeg	jpg,jpeg	quality=百分比
		transformation=
image/gif	gif	
image/tiff	tif,tiff	compression=
		colordepth=位的数量
image/png	png	Saveflag=0 或 1
image/wmf	wmf	

从头开始建立一个位图

建立一个新的位图很简单——只要使用 **Create()** 方法。例如，要建立一个 **320** 象素宽、**200** 象素高的位图，你可以象下面这样做：

```
oNewBitmap = newobject('GpBitmap', 'ffc/_gdiplus.vcx')
oNewBitmap.Create(320,200)
```

该位图的象素格式是默认的——**32** 位每象素（红、绿、蓝和透明各 **8** 位）。你也可以指定不同的格式，例如单色调、灰度等级或者调色板索引。要改动这幅图像的格式，你可以使用 **GetPixel** 和 **SetPixel** 方法。不过，不用说，这是相当麻烦的事。

```
* 使用 SetPixel 画一个黑色的边框
for x = 0 to 319
    oNewBitmap.SetPixel(m.x,0,0xFF000000)
    oNewBitmap.SetPixel(m.x,199,0xFF000000)
endfor
for y = 1 to 198
    oNewBitmap.SetPixel(0,m.y,0xFF000000)
    oNewBitmap.SetPixel(319,m.y,0xFF000000)
endfor
```

从一个新的位图建立一个 **GpGraphics** 对象的办法要有用的多，这样我们就可以用上类似的画图技术了。下面的代码使用一个 **GpSolidBrush** 对象来以黄色填充一个新的位图，并在其上画一个红色的三角形：

```
oBmpGraphics = newobject('GpGraphics', ;
    'ffc/_gdiplus.vcx')
oBmpGraphics.CreateFromImage(m.oNewBitmap)

* 以黄色填充
oBmpGraphics.Clear( 0xFFFFFF00)

* 画红色的三角形
oBrush = newobject('GpSolidBrush', ;
```

```
        'ffc/_gdiplus.vcx','',0xFF990000)
    local aTriangle[3,2]
    aTriangle[1,1] = 160
    aTriangle[1,2] = 30
    aTriangle[2,1] = 80
    aTriangle[2,2] = 170
    aTriangle[3,1] = 240
    aTriangle[3,2] = 170
    oBmpGraphics.FillPolygon(m.oBrush,@aTriangle)
```

这里可以象前面那样保存为一个文件。结果如图 1 所示。

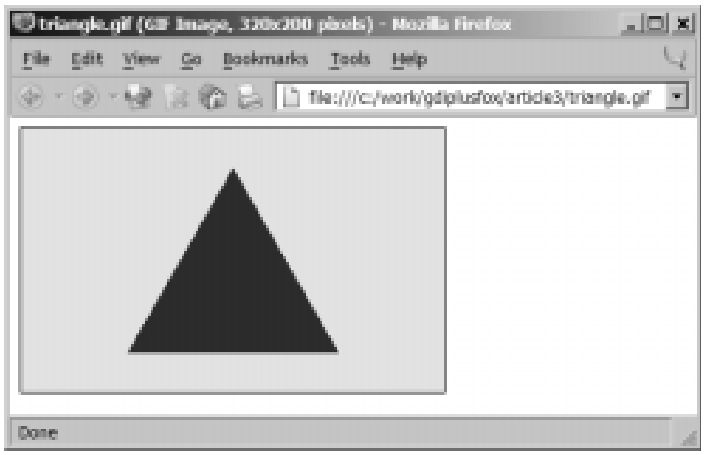

图 1、一个从头开始绘制的位图

建立缩略图——方法 2

做一个缩略图的另一种办法是建立一个所需大小的新位图，并在其中划入第一个图像。这种办法你可以控制位置、高宽比、以及其它的装饰品——比如边界和背景色。

我将会演示一个这种技术的例子，但是就像编程中经常出现的情况那样，这里有许多你可以尝试的变数和扩展。

在这个案例里，我将建立一个能从磁盘读取一个图像文件的函数，然后建立一个指定大小的缩略图。图像将在保持它的长宽比的前提下被缩放为合适的大小。如果缩略图的形状与原来的不同，那么它的边界将以单色被填充。结果如图 2 所示。

```
    function CreateThumbnail( ;
        cSourceFilename as String ;
```

```
        , cThumbFilename as string ;
        , nThumbWidth as Integer ;
        , nThumbHeight as Integer ;
        , nBackgroundColor as Integer ;
        )
```

这个函数的实现是象前面的从磁盘加载一个图像文件那样开始的。

```
    oSrcImage = newobject('GpImage','ffc/_gdiplus')
    oSrcImage.CreateFromFile( cSourceFilename )
```

然后，以指定的大小建立一个新的缩略图，并为它填充背景色。

```
    oThumb = newobject('GpBitmap', 'ffc/_gdiplus.vcx')
    oThumb.Create(m.nThumbWidth,m.nThumbHeight)
    oGraphics = newobject('GpGraphics','ffc/_gdiplus.vcx')
    oGraphics.CreateFromImage(m.oThumb)
    oGraphics.Clear( m.nBackgroundColor )
```

深入研究图形以后，就不可避免的要使用某些数学计算。在这个案例里，它是非常简单的。为了维持原来的长宽比，我需要用同一个值来缩放宽度和高度。这样一来，图像可以是被水平填充入位图而在顶部和底部留下空白、也可以是垂直填充在两边留下空白——只看它们哪个较小。

```
    nScaleFactor = min( ;
            m.nThumbWidth/ oSrcImage.ImageWidth ;
            , m.nThumbHeight/oSrcImage.ImageHeight)
```

注意，源图像并非必须是以象素来衡量的！在这种算法中是没关系的，因为被使用的值只是用来判定最终的图像大小用的：

```
    nDrawnWidth = oSrcImage.ImageWidth*m.nScaleFactor
    nDrawnHeight = oSrcImage.ImageHeight*m.nScaleFactor
```

然后，我们就开始计算位置以便它被画在目标位图的中央：

```
    nDrawX = (m.nThumbWidth-m.nDrawnWidth)/2
```

```
nDrawY = (m.nThumbHeight-m.nDrawnHeight)/2
```

最后，是有趣的部分：把图像画在位图上面！

```
oGraphics.DrawImageScaled( oSrcImage ;
      , m.nDrawX ;
      , m.nDrawY ;
      , m.nDrawnWidth ;
      , m.nDrawnHeight )
```

最后一步是将结果图像保存为一个文件。

```
oThumb.SaveToFile( m.cThumbFilename, 'image/jpeg' )
```

除了这里使用的 **DrawImageScaled()** 方法以外，还有个 **DrawImageAt()** 方法，它不会缩放图像；另外一堆方法则只画出源图像的一部分。这里是我用于生成图 **2** 的图像的代码：

```
CreateThumbnail( ;
      'zoephoto.jpg', 'zoethumb3.jpg' ;
      , 120, 100 ; && 120 象素 x 100 象素
      , 0xFF66FF66 ; && 亮绿色
      )
```

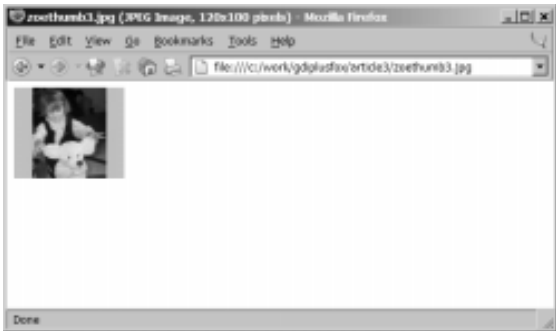

图 **2**、使用第二个办法建立的缩略图

错误处理的建议

这个系列的文章写了这么久，我只提供了假设一切都 **OK** 的代码。在实际的编程中当然没有这么顺利了，尤其是在调试的时候！

几乎所有在 _GDIPLUS 中的类都会返回一个表示是否成功的逻辑值(.T. 表示成功,.F.表示失败),不然,则返回某种无效值,比如 0 或者 NULL。

为了找到发生了什么错误,每个 _GDIPLUS 对象都有几个属性,你可以从中读取 GDI+ 状态码,如果发生了后台的 Win32 错误的话,还可以读取到 Win32 错误代码(见表 4)。

表 4、 重要的 GDI+ 状态码

以 #define 定义的常量符号	值	解释
GDIPLUS_STATUS_OK	0	成功!
GDIPLUS_STATUS_GenericError	1	不明错误时使用的状态码
GDIPLUS_STATUS_OutOfMemory	3	内存溢出,GDI+有个讨厌的趋向,即使发生了别的什么错误也会报这个错
GDIPLUS_STATUS_Win32Error	7	查看 Win32LastError 属性以了解真正的错误原因
GDIPLUS_STATUS_GdiplusNotInitialized	18	GDI+ 没有被初始化(例如 createobject('ReportListener')时出错)。注意,这种情况通常会触发一个在 _GDIPLUS 中的错误

例如,下面的代码试图去保存到一个写保护的文件。它会去检查 SaveToFile() 方法是否成功,如果否,则读取一个相应的状态码。

```
if not oImage.SaveToFile('protected.jpg','image/jpeg')
 if oImage.GetStatus() = GDIPLUS_STATUS_Win32Error
      messagebox( "Win32 error occurred: " ;
           + ltrim(str(oImage.win32LastError)) )
 else
      messagebox( "GDI+ error occurred: " ;
           + ltrim(str(oImage.GetStatus())) )
 endif
 * 返回或者其它处理
endif
```

你也许不会惊讶的发现 5 是 Win32 的错误码“访问被拒绝”(见图 3)。

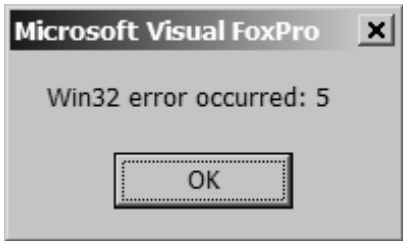

图 3、写入到一个只读文件

如果你自己做了一个 _GDIPLUS 类库的拷贝，可以通过改动在 GDIPLUS.H 头文件顶部的某些 #define 值来修改它的错误处理行为。例如，默认的内建错误处理器会自行捕捉所有的错误而并不把错误传递给你的应用程序错误处理器。你可以通过打开 GDIPLUS_ERRHANDLER_RETHROW 开关标志来改变这种行为：

```
#define GDIPLUS_ERRHANDLER_RETHROW .T.
```

下个月

我有了太多的关于怎样结束这个系列文章的念头，以至于无法决定到底该选用哪个。静候更多 GDI+ FFC 类的极酷用法吧！

下载：410NICHOLLS.ZIP

来自 VFP 开发团队的 TIPS

原著: 微软 VFP 开发团队

翻译: LQL.NET

每个月，VFP 开发团队成员都会给你带来有趣的 VFP 技术 TIPS。这个月带给你的是如何让 BINDEVENT 执行 WINDOWS 消息。注意：这个功能不能在 VFP9 公开测试版上运行，只有 VFP9 正式版才有这个功能。

绑定 VFP 代码到 WINDOWS 消息

BINDEVENT 函数有了新功能（仅在 VFP9 正式版中有效），VFP 程序员现在可以在收到 WINDOWS 消息时执行 VFP 代码。一个 WINDOWS 应用程序从 WINDOWS 操作系统收到消息然后事件被触发。

举个例子，“鼠标移动”消息发出，计算机被告知进入 休眠 或 准备 或 一个 USB 设备插入 状态。你可以在 Visual Studio "Platform SDK\include"目录下的 WINUSER.H 头文件中找到许多 WINDOWS 应用程序可以接收的消息。

使用 BINDEVENT 的这个新功能，你需要注意 4 个参数：

? hWnd - WINDOWS 消息句柄

? nMessage - 消息挂钩，整数值

? oEventHandler - 指向消息的对象引用

? cDelegate - 收到消息后要执行的方法

下面的代码向你展示了当 WINDOWS XP 桌面主题改变时如何执行 VFP 代码。你可以用这个样例来作为你探索这一重要新功能的切入点。

```
* from Mike Stewart
*-- DEFINE from winuser.h for theme changes
#define WM_THEMECHANGED 0x031A
*-- For getting the address of the VFP window
```

```
*-- process
#define GWL_WNDPROC -4
PUBLIC oHandler
*-- Create the object that will be handling this
*-- Windows message
oHandler=CREATEOBJECT("IDEHandler")
*-- Bind to the Windows message
*-- using new BINDEVENT syntax
BINDEVENT(_screen.hWnd, WM_THEMECHANGED, ;
  oHandler, "HandleEvent")
DEFINE CLASS IDEHandler AS Custom
*-- Placeholder for the VFP window process
nOldProc = 0
PROCEDURE Destroy
  UNBINDEVENT(_SCREEN.hWnd, WM_THEMECHANGED)
ENDPROC
PROCEDURE Init
  *-- Use this API to get the address
  *-- of VFP window process. Refer to
  *-- MSDN for usage.
  DECLARE integer GetWindowLong IN WIN32API ;
    integer hWnd, ;
    integer nIndex
  *-- Use this API call to pass the message and
  *-- parameters to the VFP window process
  *-- after we are done with it.
  DECLARE integer CallWindowProc IN WIN32API ;
    integer lpPrevWndFunc, ;
    integer hWnd,integer Msg,;
    integer wParam,;
    integer lParam
  *-- Get the address for the VFP window process
  *-- to use later.
  *--
  *-- From MSDN, when GetWindowLong is passed
```

```
  *-- GWL_WNDPROC: "Retrieves the address of the
  *-- window procedure, or a handle representing
  *-- the address of the window procedure."
  THIS.nOldProc=GetWindowLong(_VFP.HWnd,GWL_WNDPROC)
ENDPROC
PROCEDURE HandleEvent(hWnd as Integer, ;
  Msg as Integer, wParam as Integer, lParam as Integer)
  lResult=0
  *-- Note: for WM_THEMECHANGED, MSDN indicates the
  *-- wParam and lParam are reserved so can't use them.
  DO CASE
    CASE msg= WM_THEMECHANGED
      MESSAGEBOX("Theme Changed!")
    ENDCASE

  *-- Pass the message to the VFP window process
  lResult=CallWindowProc(this.nOldProc, ;
    hWnd,msg,wParam,lParam)
  RETURN lResult
ENDPROC
```

拆分工具（第四部分）

原著: Lauren Clarke 、Randy Pearson

翻译: CY

这是文本处理技术和策略系列文章的第四部分，也是最后部分。上月Lauren Clarke 和 Randy Pearson已经开发了一个新的强大的文本处理函数，它无缝地组合了强大的规则表达式和VFP内置函数和自定义函数。在这个文章里，他们将通过组装基于文本处理的类群的模型，为开发拆分工具创建一个框架。（注意：如果你刚刚加入我们，请不要担心。先前文章里的所有示例代码都包含在这篇文章的下载文件里。）

在经过简短的对VFP内置函数的变换和加入以规则表达式类(vbscript.regExp)到VFP文本处理指令后，我们在上月的文章里已经完成一个带规则表达式的VFP函数strtranx()。通过strtranx()你可以做任何在规则表达式所赋予strtran()函数的。

```
*-- replace digits with *
*-- 替换数字为*
? strtranx("ABC123 123ABC", "/d", "*" )
** prints "ABC***"

*-- remove repeated words
*-- 移去被誉为替换的字符
? strtranx( "this text has has repeated words ",;
"\b(\w+)\s\1\b","$1")
* prints "this text has repeated words"
```

可是，强大的strtranx()函数的出现和实现动态替换的一一对应的替换值由独立的VFP代码来决定的。

```
*-- add 1 to all digits
*-- 给所有数字加1
? strtranx( [ABC123], "(\d)", ;
[=TRANSFORM(VAL($1)+1)] )
** prints "ABC234"
```

```
*-- convert all dates to long format
*-- 转换所有日期为长格式
? strtranx( [the date is: 7/18/2004 ] ,;
[(\d{1,2}/\d{1,2}/\d{4})],;
[=TRANSFORM(CTOD($1),"@YL")])
** prints "the date is: Sunday, July 18, 2004"
```

在这最后的示例里，strtranx()函数在字符串里查找日期模型并把它传入CTOD()函数和GTRANSFORM()函数，以产生替换值。通过一次简单调用strtranx()函数，文档中所有的日期都将被转换为长格式！

设计拆分模型

现在，我们已经有VFP内置字符处理函数和一些特别的“规则表达式赋予的”自定义函数用于处理。我们可以对原始的资料来进行某些非常复杂的字符处理。可是，我们缺乏一个系统来组织拆分处理。
我们更想把拆分认为是相容的系统级服务，可以在需要的时候进行调用。拆分和在复杂拆分任务中所涉及的语义学，作为我们结构体系中的不相关内容，更甚于那些我们整合到日常商务处理程序里的。如果我们使用这个方法，我们的拆分过程将会是可重用的并且会由于重用变得更强健。
虽然，我们想灵活控制拆分任务的步骤，如交换特定拆分任务处理的输入和输出，或者改变其实现的顺序。以数据驱动方式来执行这些，将会减轻配置问题，并且向着重用拆分组件更是前进一大步。
因为某些公共设计模型使得它更有利于拆分任务，我们可以在我们的拆分类群里使用基于模型的方法。

响应链

在*设计模型：面对对象软件可重用原理*（第223页），**Gamma 和 Helm**讲述了响应链的动作模型如下 ：

> *避免请求的发送者对接收者一个对象超过一个以上处理请求的耦合度。连接接收对象并沿着链传递一直到有对象处理它。*

响应链（COR）模型就是软件结构的一大群猴子。就象是玩具，可以让塑料猴子以手相连，连接成任意长度的链，设计模型等同于让开发者连接对象成任意长度的链。在它的典型形象上，加入COR的对象沿着一个链传递请求，一直到某个处理者（猴子）处理它。对于我们的分析程序，我们略微归纳了这个模型并允许每个处理者（分析程序）来对“请求”进行操作（在我们的情况是，拆分字符串）。典型COR的结构基于一个自引用的类，它的子类被用于实现每个任务。我们将对抽象的L7CORLink类作子类以支持对字符串的拆分操作。
典型的类图示如图1所示。图2表示了一个运行COR对象结构。

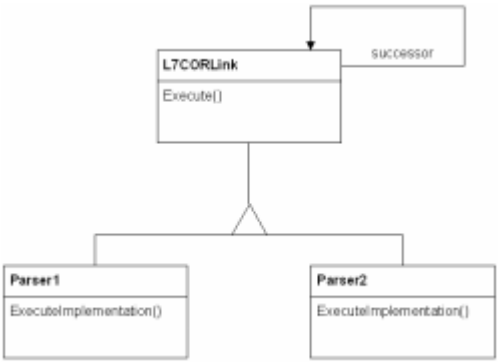

图1:基于响应链的分析程序的典型的类图示。

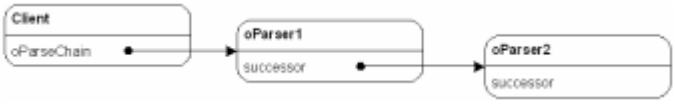

图2:运行时拆分链。

现在来看看创建链的实际代码。基类L7CORLink有一个setSuccessor()方法，可以用来钩住(hook)链里的新连接。

```
FUNCTION setSuccessor( tvSuccessor )
LOCAL loSuccessor

   *-- we can pass an object or a class name
   *-- 我们可以传递一个对象或对象名
   IF VARTYPE( m.tvSuccessor ) = "C"
   loSuccessor = CREATEOBJECT( m.tvSuccessor )
   ELSE
   loSuccessor = m.tvSuccessor
   ENDIF

   ASSERT (VARTYPE( m.loSuccessor ) = "O") ;
   MESSAGE this.Name + ;
   [:: setSuccessor() ] + ;
   [ requires an object or class name ]

   *-- if already have a successor pass down the chain
   *-- 如果沿着链传递已经有一个后续
```

```
    IF THIS.hasSuccessor()
    RETURN THIS.oSuccessor.setSuccessor(m.loSuccessor)
    ELSE
    *-- return a reference to the new link
    *-- 返回一个对新连接的引用
    THIS.oSuccessor = m.loSuccessor
    RETURN m.loSuccessor
    ENDIF
ENDFUNC && setSuccessor
```

给出的setSuccessor()方法如上所示，你可以按如下组装链：

```
lcText = "some text to be parsed"
*-- instantiate the first link
*-- 示例第一个连接
loParseChain = createobject("Parser1")
*-- add a link
*-- 加入一个连接
loParseChain.setSuccessor( createobject("Parser2") )
*-- or, just pass the class name to add another link
*-- 或者，仅传递一个类名以加入到其他连接
loParseChain.setSuccessor( "Parser3" )
*-- at this point, a 3-link chain is assembled
*-- 此时，已经组装一个3个连接的链
loParseChain.execute( @lcText )
*-- lcText has been operated on
* by Parser1, Parser2 and Parser 3
*-- lcText已经处理为：Parser1,Parser2 和 Parser3
```

注意前面的代码，在链里的第一个“连接”，loParseChain只是对Parser1作引用。Parser1加入了对Parser2的连接，并自动在execute()操作时传递字符串给它；Parser2加入对Parser3的连接；等等。为了明白前面的示例代码，我们是在客户端组装链；用于组装链的组件（并且要求连接的知识和它们处理的先后顺序）预先调用了execute()方法。在通常的实际中，客户端并不需要知道这些的任何东西，但是可以简单调用被某些其他对象组装和管理的链的execute()命令。

在这个情况下，我们以传址方式传递字符串（带@前缀）。这并不是COR模型所要求的，但是它可以

识别这个情况。由于字符串可能是很长的并且可能也很少操作的，这在传递时按址传递会比复制它可以会得到较大的内存碎片。

基类L7CORLink的Execute()方法是一个模板（参见*设计模型*第325页），它着重于沿着链传递消息。实际的拆分工作是用executeImplementation()来完成的，它在每一个类都是不同的（在实际情形是，在L7Parser和L7CORLink类间有一个更独立的层，implementation()方法实际上是“Parse()”，但是结果通常是一样的）。

```
FUNCTION execute( tvParam )
* The main invoker, template method
* 主请求者，模板方法
* tvParam must always be passed by reference
* tvParam必须是按址传递

   this.lError = .f. && innocent until proven guilty

   IF this.beforeExecute( @m.tvParam )
   IF this.isActive() AND ;
   NOT (this.lerror and this.lhaltonerror)
   this.lError = ;
   this.executeImplementation( @m.tvParam )
   ENDIF
   ENDIF

   this.afterExecute( @m.tvParam )
   *- Check for a successor link
   *-- 检查后续连接
   *- we may block execution after the first error
   *-- 我们可以在第一次错误后中止执行
   IF NOT (this.lerror AND this.lhaltonerror)
   IF THIS.hasSuccessor()
   THIS.oSuccessor.execute( @m.tvParam )
   ENDIF
   ENDIF

   THIS.onEndOfChain( @m.tvParam )
   RETURN NOT this.lError
ENDFUNC && execute()
```

任何时候都有一系列的多步骤（可配置）任务要执行，一个COR模型可以是很有用的。让我们来看看具体的示例。

拆分链示例

你可以想想对字符串进行拆分的序列操作任务。比如，考虑当接收一个上传到在线消息板的新消息时所发生的下列的拆分步骤：

1. 清除或编码字符中可能有的恶意字符。
2. 查找所有的VFP参考字并替换为对www.FoxCentral.net的连接。
3. 编码所有的e-mail地址使得它不再被垃圾邮件所捕获。

当然，这里还有其他任务需要执行，但是这只是一个典型的示例。注意到可以很容易想到在这个网站的其他地方或其他应用里需要一个或多个这样的操作。比如，“编码所有的e-mail地址”任务对前面上传到网站上的静态网页可以是很方便的处理。出于这个原因（还有其他），我们分解拆分任务为基本部分然后再组装到一个链。这个示例里的所有类，还有其他一些示例，都包含在伴随于本文章的下载文件里。可是，我们还要详细看看这两个分析程序。

在列表里以#2开始，对我们来说都很容易。L7BaseParser类为我们做了许多规则表达式的工作，所以只需要设置某些属性：

```
DEFINE CLASS VFPLinker AS L7BaseParser
    cpattern = [visual fox ?pro]
    cReplacement = ;
        [<a href="http://www.foxCentral.net">]+;
        [Visual FoxPro</a>]
ENDDEFINE

SET PROCEDURE TO L7Parsersamples, L7Utilsamples
loParser = CREATEOBJECT("VFPLinker")
lcText = [We just love visual fox pro!]
loParser.execute(@lcText)
? lcText
```

运行代码显示如下：

```
We just love <a href="http://www.foxCentral.net">
Visual FoxPro</a>!
```

这个示例事实上是利用了L7BaseParser有内置支持基本规则表达式，并且默认parse()方法授权replacex()方法来执行（基本上等同于我们的strtranx()）。

```
FUNCTION parse( tcText )
THIS.replacex( @m.tcText , THIS.cPattern, ;
            THIS.cReplacement )
ENDFUNC
```

基本分析程序的默认行为是利用cPattern和cReplacement属性来对目标字符串进行操作。回调的cPattern将会是个规则表达式，并且cReplacement会是一个简单规则表达式或是一个用来捕获子匹配的VFP表达式。
可是，我们并没有强制在拆分中使用规则表达式。比如，如果我们想让一个分析程序提供一个标准完全VFP的proper()转换，下列可以实现：

```
DEFINE CLASS ProperParser AS L7BaseParser
   FUNCTION parse( m.tcText )
      *-- tcText is passed by reference
      tcText = PROPER( m.tcText )
   ENDFUNC
ENDDEFINE
```

下面是带恶意字符分析程序的这个过程：

```
DEFINE CLASS EncodeSpecialChars AS L7BaseParser
   *-- encodes potentially malicious characters
   *-- 编码可能的恶意字符
   cBadChars = [&<>"'%;()+{}!;]

   FUNCTION parse( tcText )
LOCAL lnK, lcBad

      lcBad = this.cBadChars
      FOR lnK = 1 to LEN( this.cBadChars )
```

```
            lcChar = SUBSTR( lcBad, lnK, 1 )
            tctext = STRTRAN( tcText, lcChar, "&#" + ;
                        transform(asc(lcChar)))
        ENDFOR
ENDFUNC
ENDDEFINE
```

这个拆分并没有使用规则表达式来完成。它也可以这样做，但是在这个情况下使用完全VFP代码会更有效。我们所做的是以等同的HTML编码替代特定字符。如果有人上传如下内容到我们的消息板上：

```
<script>
DoSomethingBad();
</script>
```

它将会被EncodeSpecialChars分析程序转换为下列文本：

```
&#60script&#62
DoSomethingBad&#40&#41&#59
&#60/script&#62
```

这个编码的文本不会被解释为脚本，并且不会在客户端被执行。取而代之的是它只会显示为原始内容。

组合两个方法

到目前为止，我们创建了使用和不使用规则表达式的分析程序。我们的第三个也是最后的分析程序将会合并这两个。我们将在一个名为MailManger的类里利用L7Parser类的默认paser()行为以替换匹配的内容为自定义方法buildMailLink()动态的生成结果。

```
DEFINE CLASS MailMangler AS L7BaseParser
* converts email addresses into
* encoded mailto: links
* 转换邮件地址为编码化
* http://www.cdt.org/speech/spam/030319spamreport.shtml

lencodeEmail = .T.
```

```
*-- email address pattern
*-- 邮件地址模型
cpattern = ;
"([\w\.!#\$%\-+]+@[A-Za-z0-9\-]+(\.[A-Za-z0-9\-]+)+)"
cReplacement = [=this.buildMailLink($1)]

    FUNCTION buildMailLink( tcAddress )
    *-- convert someone@someplace.com to
    * <a href="mailto:someone@someplace.com">
    * someone@someplace.com</a>
    * with optional encoding
    * 带可选项编码的转换邮件地址

        LOCAL lcHref, lcAddress

        lcAddress = ALLTRIM( m.tcAddress)
        lcHref = "mailto:" + m.lcAddress

        *-- shall we encode the href?
        *-- 是否需要编码引用地址
        IF THIS.lencodeEmail
        lcHref = THIS.convertToHtmlCodes(m.lcHref)
        lcAddress = THIS.convertToHtmlCodes(m.lcAddress)
        ENDIF
        RETURN [<a href="]+m.lcHref+[">]+m.lcAddress+[</a>]
    ENDFUNC

    FUNCTION convertToHtmlCodes( tcString )
        LOCAL lcHTMLEncoded , lnChar
        lcHTMLEncoded = ""
        FOR lnChar = 1 TO LEN( tcString )
            lcHTMLEncoded = m.lcHTMLEncoded + ;
                "&#" + ;
                TRANSFORM(ASC(;
                SUBSTR(m.tcString,m.lnChar,1)))+";"
```

```
        ENDFOR
        RETURN m.lcHTMLEncoded
    ENDFUNC
ENDDEFINE
```

关于这个类的最重要的就是cReplacement属性：

```
cReplacement = [=this.buildMailLink($1)]
```

这意味着：替换所有匹配模型为从这个类的buildMailLink()方法生成的结果，从模型里传递第一个捕捉子匹配作为参数。在内部，L7BasePaser将会见到以“=”开始的replacement属性，实现当前一一对应处理的需要，并且遍历所有匹配集合，创建并执行指定匹配命令为在源文本里每个匹配生成替换字符串。我们将钩住(hook)规则表达式的替换命令并用自己的基于VFP的方法代码来扩展它。
于是，现在我们有了三个分析程序。让我们把它们一起加入到一个链再作个测试。

```
SET PROCEDURE TO L7Parsersamples, L7Utilsamples
loParser = createobject("EncodeSpecialChars")
loParser.setSuccessor("VFPLinker")
loParser.setSuccessor("MailMangler")

lcText = [We just love our visual fox pro! ]+;
       chr(13)+chr(10)+;
       [Why don't you email me about it: ]+
       [foxfan@somedomain.com.]

loParser.execute( @lcText )
```

在调用execute()后，lcText将会如下：

```
We just love our <a href="http://www.foxCentral.net">
Visual FoxPro</a>&#33
Why don&#39t you email me about it:
<a href="&#109;&#97;&#105; ... More encoding here...
&#109;</a>.
```

它看起来很奇怪，但浏览器会表示为：

```
We just love our Visual FoxPro! Why don't you email me
about it: foxfan@somedomain.com.
```

顺序的重要性

注意，拆分链的顺序是非常重要的。如果我们把VFPLinker放在第一个，我们将会被某些东西缠绕成这样：

```
We just love our &#60a href=&#34http://
www.foxCentral.net&#34&#62Visual FoxPro&#60/a&#62&#33
Why don&#39t you ...
```

注意到<>字符将会被编码为VFP连接。这并不会在浏览器里得到希望的结果。
执行顺序在一个分析程序与其他分析程序相互影响时也是很重要的。我们有两个方法来减轻不必要的分析程序影响：按序组装链和使用“堆栈”。
L7BasePaser类有一个“堆栈”，它是作为转换匹配的临时存储空间。如果lPushToStack属性为真(TRUE)，分析程序将会执行它的通常工作，但是代替于在匹配位置加入转换文本，它将会把它压入堆栈并在源文本里加入占位符。然后在链被遍历和所有拆分已经完成时，堆栈里的项目将会被回填到文本里。
这样的方法，一个分析程序可以在链里后续的分析程序隐藏起它的工作。我们发现这两个方法可以让我们完全控制整个拆分过程。一个更多详细顺序控制的方法，我们引导你看Steven Black的*The Hooks and Anchors Design Pattern*文章（参见文章结尾内容）。
为了总结，我们把拆分任务作为独立的类来实现，这些可以串成一个“链”。我们在运行时通过元数据或其他，来排序和增加链的任务。同样，注意到客户端并不知道在链里有多少连接；它只需要引用第一个连接，并且CORLink类会对链自行反复，返回链里所有的操作结果。

处理链群

我们可以组装任意长度的链，对setSuccess()的调用仍然有少许工作要做。在这篇文章伴随的下载文件里，你可以找到一个类L7PaserChainAnchor。L7BaseParser的子类是个特别的CORLink，它可以提供对链的基本管理服务。
它允许所有的连接共享一个规则表达式对象和公共的堆栈对象。同样，L7PaserChainAnchor类也提供一个forgeChainImplement()钩子(hook)以在运行时从代码或元数据创建一个链。
我们可以利用这个类来包装我们先前的线程任务为单个的类，它将会在例化时扩展为三个连接的链。

```
DEFINE CLASS ThreadParser AS L7ParseChainAnchor
   FUNCTION forgeChainImplementation()
      this.setSuccessor( "EncodeSpecialChars" )
      this.setSuccessor("VFPLinker")
      this.setSuccessor("MailMangler")
   ENDFUNC
ENDDEFINE
```

利用这里的这个类，我们对链的使用可以减少为下面两行代码，不再需要知道拆分细节的有关知识和拆分顺序：

```
loParser = createobject("ThreadParser")
loParser.execute(@lcText)
```

最后，在运行中创建这些链是昂贵的，所以它需要有一个好办法来缓存这些链并加以重用。同样，也要有一个好办法来完整的提炼拆分任务为一个应用级的服务。L7PaserChainManager可以完全做到：

```
DEFINE CLASS MyParseChainManager AS L7ParseChainManager
   FUNCTION init()
   *-- chains can be added on init(), or more likely,
   * configured from meta-data
      this.addChain("ThreadParser","ThreadParser")
      this.addChain("SQLTypeFix","SQLTypeConverter")
      this.addChain("TextToHTML","TextToHTML" )
      this.addChain("TextToPDF","TextToPDF" )
      this.addChain("TextToRTF","TextToRTF" )
   ENDFUNC
ENDDEFINE
```

调用addChain的第二个可选参数为当前拆分任务创建一个普通的IDs，以便不同的链的实例类名不再需要嵌入到你的代码里。这为应用方法打开了一个用以配置和创建链的门（参见*设计模型*，第107页）。
Typically, a chain manager would be hung off a persistent object and called as follows:
典型的，链管理器将会变成为一个持久对象并可以调用如下：

```
goApp.oParseManager.parseByName("ThreadParser",@tcText)
```

下载文件包含有四个拆分支持类（参见表1），并附加一些示例分析程序。

表1:拆分支持类

类	描述
L7BaseParser	分析程序的工蜂，这个类是响应链的成员。
L7MultiParser	为快速处理元素拆分任务的次轻量级的分析程序
L7ParseChainAnchor	L7BasePaser的子类，相对于后续连接作为COR的第一个连接。对象为链里的连接提供了某些共享服务。
L7ParseChainManager	一个为你的整个应用程序缓存和管理拆分链的类。

结论

VFP是一个强大字符串操作者，但是它缺少对规则表达式的支持。通过附加的规则表达式引擎，我们可以增加VFP的性能并且创建一些非常强大的模型搜索和替换工具。最后，通过利用COR设计模型，我们可以为我们的应用创建一个健壮、可重用的拆分服务。想想把拆分作为一个应用级服务，它利用可配置的链的基本拆分对象多过一次性的自定义函数库，可以带来一些优美的对字符操作挑战的数据驱动解决方案。

感谢

我们衷心感谢Steven Black开辟的在本文章里所讲述的拆分链的设计理念。

Resources

• *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (Addison-Wesley, 1994)

• *Mastering Regular Expressions*, by Jeffrey E.F. Friedl (O'Reilly Media, 2002)

• *The Hooks and Anchors Design Pattern*, by Steven Black (paper available at http://stevenblack.com/HooksAndAnchorsDesignPattern.ASP)

“哚、哚、哚”
有人在家吗？（第二部分）

原著：Andy Kramek & Marcia Akins

翻译：Fbilo

上个月，Andy Kramek 和 Marcia Akins 推定了实现应用程序安全性最佳的途径是采用一种“基于角色”的模型。他们将“角色”定义为一个标志符，它代表一种可以被应用程序中的一个或多个用户使用的特定的访问模式。在这个月的专栏中他们选定的任务，是设计和实现一个类来处理这种多层基于角色的安全性的管理任务。

玛西亚：现在我们已经决定采用基于角色的安全性了。我认为我们应该开始设计支持它的各个表了。

安迪：我同意。我们需要的第一件东西是一个用于储存用户名和密码的表。在表 1 中的结构应该不错。

表 1、Login_user 表的结构

字段名称	数据类型	字段说明
user_id	I	主关键字
login_nme	C(10)	用于登录到系统中的名字
pwd	C(10)	加密了的密码
fname	C(10)	用户的姓
lname	C(10)	用户的名

玛西亚：打住！你没漏掉什么东西吗？对角色表的外键在哪儿？

安迪：我想我们最好使用一个分配表来为一个指定的用户所属的每个角色存储用户 ID 和角色 ID。毕竟，在一个组织中，单个用户可能有多种不同的角色身份。

玛西亚：你还有点记性吗，安迪？上个月我们已经解决这个问题了。就此而论，一个角色只不过是在应

用程序中的一种“访问模式”。它本身跟在应用程序中的是一个还是多个角色没什么关系。还是你改变主意了？

译者注：

这一段以我的理解是这样的：在上一期的专栏中，玛西亚提到了一般角色和特殊角色的概念，所谓的“单个用户可能有多个不同的角色身份”其实质上不过是用户拥有的权限与一般角色相比要多上那么几项，那么，完全可以给该用户专门建立一个特殊的角色以指定他所拥有的全部权限，而不是让他拥有多个角色。

安迪：不，我没忘。只是要花我一分钟来回忆而已。毕竟，从我们上次讨论开始已经过了一个月了。你是对的，给每个唯一的用户名和密码的组合分配一个角色这种办法要简单的多（并且好用）。

玛西亚：通过这种办法，我们还可以保证那些需要超过一种访问模式的人必须对每种模式使用不同的用户名和密码。这样的话，表 2 中的结构将是我们真正采用的那种。

表 2、Login_user 表最终的结构

字段名称	数据类型	字段说明
user_id	I	主关键字
role_id	I	对角色表的外键
login_nme	C(10)	用于登录到系统中的用户名
pwd	C(10)	加密了的密码
fname	C(10)	用户的姓
lname	C(10)	用户的名

安迪：事实上，这个多角色的问题完全是一盘熏青鱼！为什么一个用户要在同一个应用程序中使用两种不同的访问模式？这根本没有意义。所以，让我们继续。

玛西亚：那么，下一步就是建立角色表。这非常简单——它就只有一个 Key 和一个说明而已（见表 3）。

表 3、logion_role 表的结构

字段名称	数据类型	字段说明
role_id	I	主关键字
role_dsc	C(30)	表示角色的名称

安迪：是的，在 login_user 表和 login_role 表之间的关系是一个很好的多对一关系的真实例子。在这两个表的数据项之间没有有意义的占有关系：用于并不由角色所“占有”，用户也不“占有”角色。

login_role 所起的主要作用只是作为一个参照表，还有作为“一系列为每个指定的角色定义了访问方式的相关表的”入口。

玛西亚：现在我们必须去定义余下的那些表了。我们将需要为每个要求安全性的应用程序实体使用一套表了。

安迪：嗯？我不大明白，请解释的清楚些。

玛西亚：记住，我们将要用来处理安全性的基本原则是“根据例外”。对于任何给定的角色，我们必须能够识别它对应用程序的哪些部分的访问是受限制的。因此，在 **login_role** 表和那些定义应用程序的各个部分——模块、菜单、表单和字段——的表之间将会是一种多对多关系。

安迪：这样就清楚了。那么，你说的那“一套”表实际上指的是分配表和应用程序部件的参照表吗？

玛西亚：正确。这些表的逻辑模型见图 **1**。

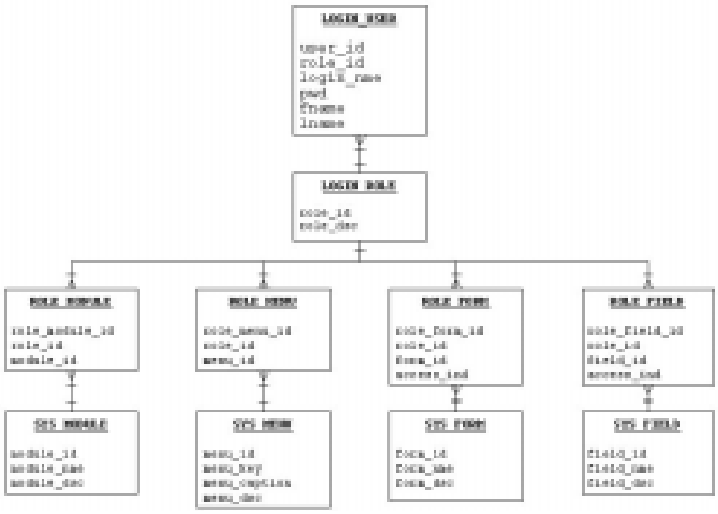

图 **1**、安全性参考数据的逻辑模型

安迪：我注意到那些以“**sys_**”开头的表都有着同样的结构，那么只要再增加一个在标准的应用程序参照表的类型标志符，我们就可以用一个表来实现它们。毕竟，物理模型并非必须要完全按照逻辑模型来做。

玛西亚：我也认为可以做到。但是，如果它们都在一个表里面，那么，不论是出于维护还是真正的实现安全性的目标，我们都必须要建立一些能够与这些逻辑表非常相似的视图。此外，这些表的结构虽然很类似，但是也并非相同的。这么做你还有问题吗？

安迪：没有了。但是既然你一直大力支持将所有的参照表都集中做成一个表，所以我有点疑惑为什么你在这里要用不同的实现方法。

玛西亚：好的，它们是非常专用的表，目的仅是为了处理安全性。它们并非是应用程序查找数据的真实部分。

安迪：这个理由不错。**Okay**，我们将按照逻辑模型来实现这些表。那么下一步呢？

玛西亚：我们要设计使用这些数据来控制对限制项目的访问的安全性管理类（**SecurityMgr**）。技巧是在主程序一启动就建立这个对象的实例（作为“私有”变量），这样的话，在该应用程序中的所有其它对象就都可以访问它了。此外，如果我们让它继承 **Session** 基类，它还可以有它自己的数据工作期。然后，当用户登录到这个应用程序的时候，**SecurityMgr** 能够返回所有为用户的角色定义了限制的数据，并且保有这些信息直到需要更多的安全服务的时候。

安迪：对我来说，看起来是可以理解的了。那么，基类的定义是怎么样的？

玛西亚：好的，由于我们使用 **Session** 基类来获得一个私有数据工作期，所以我们必须做的第一件事情是设置环境。我们在 **SetEnvironment()** 方法中做这件工作，该方法由 **Init()** 来调用，并为我们设置需要的默认值。

安迪：请告诉我这方面的内容！对数据工作期的设置问题困扰了我一次又一次。至少我现在知道了在帮助文件中完整的列表是在“设置数据工作期”主题中，所以我以前没有在这上面浪费太多时间。

玛西亚：当 **SecurityMgr** 实例化的时候，它有两个可见的属性。一个（**formDS**）握有对另一个数据工作期的引用；这只在处理表单和字段级别安全性的时候才需要。另一个（**user_id**）握有当前登录用户的 **id**。这个属性必须在登录成功后明确的设置好。它有一个 **Assign** 方法，该方法调用 **GetSecurityData()** 方法来返回与该用户的角色关联的限制项列表。这里是有关的代码：

```
DEFINE CLASS SecurityMgr AS Session
 user_id = -1
 formDS = 1

 ********************************************************
FUNCTION user_id_assign( tiUserID )
 ********************************************************
     WITH This
```

```
        IF VARTYPE( tiUserID ) = "N" ;
            AND NOT EMPTY( tiUserID )
            *** 我们有了一个可能有效的用户 id
            .user_id = tuValue
            .GetSecurityData()
        ENDIF
        RETURN
        ENDWITH
    ENDFUNC

    ************************************************************
    FUNCTION GetSecurityData()
    ************************************************************
    LOCAL vp_user_id

        *** 为登录进来的用户取得所有的限制项
        vp_user_id = This.user_id

        *** 首先取得受限制的模块
        USE lv_restricted_module IN 0

        *** 接着取得受限制的菜单
        USE lv_restricted_menu IN 0

        *** 还要取得表单访问限制
        USE lv_restricted_form IN 0

        *** 和受限制的敏感字段
        USE lv_restricted_field IN 0

    ENDFUNC
```

安迪：稍等一下。前面你曾说过你想让所有的参照表保持独立，单独作为一个表，因为这样做要比把它们放在一个表里然后分别为之做视图要容易，对吧？可这里你却又在为每个表使用本地视图了，为什么？

玛西亚：方便。由于视图是不可更新的，并且只包含一个或两个列，所以我可以直接在这个方法中写 **SQL** 查询并且使用游标。这么做没什么本质的区别。

安迪：那么表单和控件呢？我们是否需要添加什么专门的属性或方法来允许它们与安全性管理器进行交互？

玛西亚：这是一个好问题。记得我们在 **2003** 年 **8** 月刊上的专栏（“这到底是谁的活？”）吗？在那篇文章里，我们讨论了关于每个表单控件应该怎样根据表单的模式（添加、编辑或者查看）来对使能或者禁止自己负责。我们将通过给我们所有的可视化类添加一个自定义属性 **cSecurityMode** 来处理表单和字段级别的安全性问题。

安迪：并且我们可以使用控件的 **Refresh()** 方法来检查安全性设置并相应的修改它拥有的行为特性。**Good**！这就定义了表单和控件级别的安全性机制，那么模块和菜单级别的呢？

玛西亚：事实上它们非常简单。尤其是它是全是或者全否的：你或者能访问或者不能。不过，为了保证我们的思路一致，我先问你一下，你是怎么定义一个模块的？

安迪：一个模块是由一系列相关的功能组成的一组部件。举例来说，一个帐单系统，应付账款是一个模块，应收账款则是另一个。其它的模块包括库存、工资和成本。可以想象，所有最终用户都将拥有这些模块，但只能访问他们需要的部分特定的模块。如果被分配的角色不包括一个特定的模块，那么这个用户就不允许进入它。

玛西亚：足够了。因此，这里的问题就是，用户要怎样访问这些不同的模块？如果它是通过主菜单来的，那么我们根本不需要有一个单独的模块级别安全性功能。它可以在菜单级别上处理。不过，我确实有一个客户，他的应用程序是自动为每个已经购买了的模块在用户的桌面上安装快捷方式的。在那种情况下，我们就需要额外的模块级别安全性了，这样它才能够在应用程序启动的时候检查用户的访问许可。

安迪：我觉得这两种情况我们都要有所准备。如果我们不需要模块级别的安全性，把它留在里面也不会有什么坏处，但如果需要这种模块级别的安全性的话，它就起作用了。那么，让我们开始模块级别的代码吧！

玛西亚：这很简单！安全性管理器有一个叫做 **ChkModule()** 的方法，它接受一个模块的名称作为参数。然后它会检查它的本地视图来找出该名称是否在那些受限制的项目名单内，然后返回一个表示用户是否被允许访问的逻辑值。记住，我们只定义例外。所以如果这个模块的名字没有找到，那么这个用户就是被允许使用模块的。

```
LOCAL llRetVal
*** 看看用户对这个模块是否受限制
SELECT lv_restricted_module
LOCATE FOR ALLTRIM( module_nme ) == ;
    LOWER( ALLTRIM( tcModuleName ) )
llRetVal = NOT( FOUND() )
RETURN llRetVal
ENDFUNC
```

安迪：那么，调用这个方法的位置应该是在用户成功的登录到系统之后、但还没做其它任何事情之前了。这就意味着应用程序的启动代码应该是象这样的东西：

```
*** 声明和初始化变量
LOCAL lnUserID
lnUserID = 0
PRIVATE poSecMgr
poSecMgr = NULL

*** 在这里调用登录表单
DO loginform TO lnUserID

***直接建立安全性管理器
IF NOT EMPTY( lnUserID )
    *** 建立对象的实例
    poSecMgr = ;
        NEWOBJECT( 'SecurityMgr', 'SecurityMgr.prg' )
    *** 设置用户的 ID (触发 Assign 方法)

    poSecMgr.user_id = lnUserID
    IF NOT poSecMgr.ChkModule( "this_module_name" )
        *** 显示消息或者什么——不能进入!
        QUIT
    ENDIF
*** 在这里是余下的启动代码
```

玛西亚：正确！下一步我们要处理菜单级别的安全性了。要做的第一件事情是在安全性对象上建立一个叫做 **ChkMenu()** 的方法，并传递给它一个菜单的 **key**。

安迪：什么是菜单的 **Key**？

玛西亚：这只是我们在 **Sys_Menu** 表中为每个数据项（不论它是一个 **pad** 还是一个 **bar**）设置的一个唯一标志符，它也是我们在为 **GenMenuX** 定义规则的时候使用的东西。

安迪：为什么不是直接使用菜单的标题？

玛西亚：因为那在菜单中可能不是唯一的。例如，你也许在多个顶层菜单的 **Pad** 上有一个“报表”菜单项。

安迪：啊哈，是的。我没有想到这一条。那么在 **ChkMenu()** 方法中有什么内容呢？

玛西亚：没什么出奇的。它非常类似于 **ChkModule()**：

```
LOCAL llRetVal
*** 检查菜单对该用户是否受限制
SELECT lv_restricted_menu
LOCATE FOR ALLTRIM( menu_key ) == ;
     LOWER( ALLTRIM( tcMenuKey ) )
llRetVal = NOT( FOUND() )
RETURN llRetVal
ENDFUNC
```

安迪：非常类似？简直除了视图名称和字段名称以外就是一样的。这可以用一个带参数的方法来处理。

玛西亚：有什么意义？它们之所以会相似是因为在这个例子中我们做的活相似，可并非总是如此。让它去，就这样吧！

安迪：**Okay**，**Okay**！顺便问一下，你是怎么让 **GenMenuX** 阻止一个菜单项的显示的？

玛西亚：你必须在菜单设计器的特定菜单项的选项对话框中的注释（**comment**）编辑框中放入一个格式

化的表达式。**GenMenuX** 使用“***:IF**”来将文本指定为一个 **GenMenuX** 命令。不管 **IF** 后面跟的是什么，它的作用就像系统的 **Skip For** 命令一样，返回一个逻辑值。这个命令将在菜单生成的时候被运算，为特定条件进行测试的代码将被添加到 **MPR** 文件中的 **CleanUp** 部分。如果该条件运算的结果为 **.F.**，则指定的菜单项将从菜单中被删除。

安迪：清楚！那么为了让安全性管理器检查一个 **Key** 值为“**Employee**”菜单项是否受限，我只要把下面这行代码添加到菜单项的注释里面去就行了：

```
*:IF poSecMgr.ChkMenu( "Employee" )
```

玛西亚：是的，那就是所有你需要做的了。现在，这一切都很简单，但是当我们进入表单和字段级别的时候，情况就变得复杂得多了。不幸的是，我们用完了本期的版面了，那么就只能等到下个月了。

下载：**410KITBOX.ZIP**

用 Inno Setup 制做安装包（第二部分）

原著：Rick Borup

翻译：YASUR

在这个系列的第一部分（见 2004 7 月刊），Rick Borup 向你推荐了 Inno Setup 并就配置 VFP 应用程序讲述了基础知识。在第二部分里，Rick 创建了一个样例，向你演示如何用 Inno Setup 脚本来增强她的功能，包括如果安装数据库文件到应用程序以外的路径。

在这个系列的第一部分里，我讲述了如何制做 VFP 安装包的基础知识，里面有个很简单的脚本样例叫 myVFPApp，内容是这样的：

```
[Setup]
AppName=MyVFPApp
AppVerName=MyVFPApp version 1.0.0
DefaultDirName={pf}\MyVFPApp
DefaultGroupName=MyVFPApp
[Files]
Source: MyVFPApp.exe; DestDir: {app}
Source: Readme.txt; DestDir: {app}; Flags: isreadme
#include "VFP8Runtimes.txt"
[Icons]
Name: {group}\MyVFPApp; Filename: {app}\MyVFPApp.exe
```

这个脚本，麻雀虽小，五脏俱全。当然，实际情况可能要复杂很多，某些选项需要考虑：

? 安装前需要用户确认协议许可

? 指定一个最低的操作系统要求

? 创建注册表键

? 执行安装向导时显示的图片

? 配置安装所需的组件

? 安装数据库文件到应用程序以外的路径

此外，你还要做些事情使你的安装程序更易用。

[Setup]标识增强

默认情况下，Inno Setup 脚本中的文件引用都在脚本所在文件夹下，如果文件在其他文件夹，你可以为每个文件加上完全路径名称，但这样你的脚本会变得又长又笨。那，你可以象我这样做，加一个 SourceDir 指向到[Setup]标识，这样就可以把安装的文件引用都指到 SourceDir 指定的文件夹中去

```
SourceDir=C:\FoxTalk\MyVFPApp
```

默认情况下，编译后的输出文件被命名为 SETUP.EXE 并被放到一个叫 OUTPUT 的子文件夹下。你可以用 OutputBaseFilename 和 OutputDir 指定一个不同的输出文件名和路径。下面的例子告诉 Inno Setup 编译后输出文件名“myVFPApp 1.0.0. Setup.exe”并存到 C:\VFP8Distrib\MyVFPApp 中去。

```
OutputBaseFilename=myVFPApp 1.0.0 Setup
OutputDir=C:\FoxTalk\MyVFPApp\Distrib
```

如果你的应用程序对 WINDOWS 版本有最低要求，你可以指定 MinVersion。下面的例子假设你的应用程序最低要求是 WIN98 或 WIN2000。WINDOWS 版本号对应的参数请参考 Inno Setup 的帮助文件。

```
MinVersion=4.1.2222,5.0.2195
```

如果安装前你需要用户确认协议许可，你只需在脚本中简单地加入 LicenseFile 参数即可。协议许可文件可以是 TXT 文件或 RTF 文件。

```
LicenseFile=license.rtf
```

安装后的版本更新

当你更新你的应用程序版本时，Inno Setup 会检查前一个版本是否安装过。应用程序的版本依赖

AppID 参数，她是一个内部值，你可以设置她。如果 AppID 省略，Inno Setup 会使用 AppName 参数，但 AppID 不象 AppName 那样会在安装过程中显示出来，所以使用 AppID 你可以想设什么值就设什么值。

不管显式还是隐式，这个共同的 AppID 值都将绑定在应用程序的每个版本。这样做，一个重要的原因便是共享卸载信息。如果没有这个共同的 AppID，每个版本都将会被加入 WINDOWS 的 添加/删除程序中。在这个例子中，AppID 被设为和 AppName 值相同。

```
AppID=MyVFPApp
```

当你安装更新应用程序版本时，即使你用 DefaultDirName 参数指定到不同的目录，Inno Setup 默认安装目录仍然是前个版本所在的目录。这个行为被 UsePreviousAppDir 参数所控制，它的默认值是 Yes。

当你安装更新应用程序版本时，Inno Setup 不会警告用户目标文件夹已经存在，但当你安装一个全新的应用程序时发生这种情况，她就会警告说目标文件夹已存在。这个行为被 DirExistsWarning 参数所控制，它的默认值是 Auto（自动）。

如果你要使用这两个参数的默认值，那你可以不用将她们包含在脚本里。但不管什么情况我都喜欢把她们包含进去，这样我可以使我的脚本更易读。

```
UsePreviousAppDir=yes
DirExistsWarning=auto
```

定制安装界面

你可以用图片来打扮你的安装向导界面。有很多 Inno Setup 图片可供下载，或者你也可以自己制做。你需要一个大图和一个小图，大图最大可以是 164x314，小图最大可以是 55x55。然后你可以象这样在脚本中把图片加到向导中：

```
WizardImageFile=compiler:images\WizModernImage13.bmp
WizardSmallImageFile=compiler:images\WizModernSmallImage13.bmp
```

"compiler:"选项告诉 Inno Setup 图片所在路径，默认参考路径是脚本所在目录。

Creating Registry entries

Registry entries can be created on the target machine by including a Registry section in the setup script. A typical use for Registry entries might be to store the location where the application is installed, as illustrated in the following script.

创建注册表键

在脚本中用[Registry]标识可以在目标电脑上创建注册表键。比如在注册表中记录应用程序安装路径等等，如下例所示：

```
[Registry]
Root: HKCU; Subkey: Software\ITA; Components: workstation;
Flags: uninsdeletekeyifempty
Root: HKCU; Subkey: Software\ITA\myVFPApp; Components: workstation;
Flags: uninsdeletekey
Root: HKCU; Subkey: Software\ITA\myVFPApp\Settings; ValueType: string;
ValueName: AppDir; ValueData: {code:GetAppDir}; Components: workstation
Root: HKCU; Subkey: Software\ITA\myVFPApp\Settings; ValueType: string;
ValueName: DataDir; ValueData: {code:GetDataDir}; Components: database
```

其中参数"code:"和"Components:"等下会解释。

[Types]标识和[Components]标识

Inno Setup 允许你用组件的方式来组织你的安装包，每个组件都和一个或多个不同的安装类型关联。系统提供的安装类型是 完全安装、简单安装和定制安装，你还可以自己创建安装类型。文件和其他资源，比如注册表键值等，被标注在一个或多个组件中。当用户安装时选择了一个安装类型，Inno Setup 就安装相关组件所指定的文件和资源。如果某个文件不属于任何一个组件，除非有其他原因，否则它将被无条件安装。

很多 VFP 应用程序以数据为中心，因此很自然需要定义一个工作站安装组件和一个数据库文件安装组件。[Types]标识和[components]标识
的例子如下：

```
[Types]
Name: full; Description: Full installation
Name: workstation; Description: Workstation installation
Name: database; Description: Database installation
Name: custom; Description: Custom installation; Flags: iscustom
[Components]
Name: workstation; Description: Workstation files;
```

```
Types: full workstation
Name: database; Description: database files; Types: full database
```

你看这个[Components]标识，你可以看到 workstation 组件和 full、workstation 两个安装类型关联，而 database 组件则和 full 、database 两个组件关联。

在安装的时候，用户可以任选 4 个安装类型之一。安装向导会显示所选类型关联的组件清单。清单里每个组件后面都跟着选择框，因此用户可以看到哪些组件会被安装。如果用户改动组件选择，定制安装就会被调用。

个别组件相关文件用 Components 参数加入[Files]标识中，后面跟着组件的名字，如下例：

```
Source: MyVFPApp.exe; DestDir: {app}; Components: workstation
Source: Readme.txt; DestDir: {app}; Components: workstation;
Flags: isreadme
```

使数据独立与应用程序

在多用户环境下，有个共同点，就是要把应用程序安装到某个地方比如本地的 Program Files 文件夹，而要把数据库文件安装到另一个地方比如文件服务器以让所有用户共享。第一步就是将应用程序的工作站、数据库文件组件分开，这前面解释过。然后，你还得想个办法为数据库组件创建一个独立的目录，把它和应用程序安装目录分开。

在[Files]标识里，每个文件的 DestDir 参数值决定这个文件要装在哪儿。DestDir 参数值通常传给{app}常量，安装时供用户选择路径，或者传给其他 Inno Setup 常量，比如{pf}（Program Files 目录）或{cf}（Common Files 目录）。如果你要把数据库安装到应用程序无关的路径，你没法把数据库文件组件的 DestDir 参数传到{app}或其他常量中。你可以用一个固定的值比如 F:\Data\myVFPApp，但这样会让你的安装程序显得呆板不灵活。

为了使数据库安装完全脱离应用程序路径，且又不使用固定路径，则应在用户安装时让用户来指定数据库安装路径。要达到这个目的，你需要在安装过程中插入一个定制页面。Inno Setup 为此提供了一个方法让你写入脚本。

Pascal 脚本

Inno Setup 中的自定义脚本是采用 Pascal 脚本，这些脚本被放在[Code]标识下。Pascal 脚本可能对大多数 VFP 程序员不熟悉，但别让这影响你。毕竟，代码就是代码，对吧？

样例中的代码检查安装时数据库组件是否被选中，如果是则应当插入一个自定义页面让用户为这个数据库文件选择目标路径。用户指定的这个值被存在一个叫 DataDir 的变量中。然后一个叫 GetDataDir

的函数返回 DataDir 值。

我建立这段代码的目的是让各位熟悉一下 Inno Setup 的代码风格。由于文章篇幅所限，我不能在这里贴上所有的代码，但是这里提供的一小段可以让大家看看 Pascal 脚本到底是什么玩意儿。[Code]标识下完全的代码我把她放在附带的下载文件里了。

```
[Code]
var
  DataDir: String;
function GetDataDir(S: String): String;
begin
  { Return the selected DataDir }
  Result := DataDir;
end;
```

这段代码建立后，你便可以在其他标识中用 GetDataDir 函数调用她。在这段代码中，你要为数据库文件指定 DestDir 参数。[File]标识中要作如下处理：

```
Source: Data\customers.DBF; DestDir: {code:GetDataDir};
Components: database
Source: Data\customers.CDX; DestDir: {code:GetDataDir};
Components: database
```

这两个文件也要被标识成属于安装包中的数据库组件，这样数据库组件被选中时她们才会被安装。

Inno Setup 处理数据、应用程序无关性的能力非常强大，因为她可以让你创建一个非常灵活的安装包来适应各种多变的场合。当然这里只提供了一个 Inno Setup 的 Pascal 脚本样例。你可以发挥想象用[Code]标识来达到你的目的；更多细节和方法请参考 Inno Setup 帮助文件。

总结

在这篇文章中，我向你展示了一些东西，把你从这个系列第一部分讲述的基础样例引申到一个实战的安装脚本中去。尽管这里讲的都还比较表面化，但这里有许多灵活的知识有助于你以后掌握更多。我建议你看看 Inno Setup 的帮助文件以及附带的样例，也可以到 Inno Setup 的网站（www.jrsoftware.org/isinfo.php）去转转，上面应该有些链接，指向很多新闻组，你可以试试。