

FoxTalk 2.0

Solutions for Microsoft® Visual FoxPro® Developers



- 2004 年 5 月刊 -

编辑语：新一代的诞生

David Steven 刚刚抱上孙子，欣喜之下，撰文谈论自己的爱孙和 FoxTalk 2.0、VFP 9，透露了一点消息：第 6 期的 Foxtalk 将会用 100% 的内容来讲述 VFP9 的新功能。

建立 Web 页部件

Page.3

Doug Hennig 一时兴起，重新看了一下自己的网站主页的代码，发现绝大部分都是 Copy & Paster 来的——于是，就想到做一些能生成 HTML/ASP 页面的类，这样一来，生成网页文件的代码就“可重用”了。

来自 VFP 开发组的 Tips

Page.x

雷德蒙得微软开发园区的 41 号楼这些日子里非常的繁忙，Visual FoxPro 开发组正在日以继夜的忙于 VFP 9 的新功能、修正 Bug、写文档和示例代码、以及穷举测试。百忙之中，他们也偷空在 Foxtalk 2.0 上开了一个专栏，每期介绍一些 VFP 新功能的使用方法，这一期讲的就是关于 VFP 9 中 SQL Select 语句的一些新功能...

RSS: 通过 XML 发布新闻

Page.x

BLOG 现在是一个非常热门的话题了，连 VFP 开发组也搞了个 Blog 玩玩。这篇文章就是介绍怎样用 VFP 的命令来快速生成 Blog 的 Rss 文件的。

一个类的接触

Page.x

VFP 的代码引用工具可以找到大多数基于 Prg 的类，但是没法找到基于 VCX 的类到底位于哪个类库中。AVCXClasses()函数可以完成这个任务。可惜的是，没有哪个工具能够两者兼备。于是作者做了一个实现这两个目的的小工具。

生成 Web 页 部件

原著: Doug Hennig

翻译: Fbilo

因为其拥有一批快速而强大的字符串函数，VFP 是一个生成 HTML 的极好的工具。这个月，Doug Hennig 演示了你可以怎样的将 Web 页当作是一个可重用的 HTML "部件"的集合，以轻松的生成静态和动态的网站。

最近，当我在维护我的网站（www.stonefield.com 和 www.stonefieldquery.com）的时候，发现我自己干的绝大多数都是 Copy & Paster 的活。当我写代码的时候，因为 Copy & Paster 而涉及的所有潜在的问题、以及额外的工作不断的引起我的警惕。因为使用的是象 VFP 这样一种面对对象的语言，我们总是努力通过继承以及其它的机制来建立可重用的代码。而现在我需要的是能够生成 HTML 页的类似的东西：在其它文档中能够重用某个文档里面的不同部分的能力。我把这些部分叫做“部件”。最理想的情况是，能够简单的按次序组合一些不同的部件就能建立一个 Web 页。

在阅读了 Marcia Akins、Andy Kramek 和 Rick Schummer 合著的精彩作品《MegaFox: 扩展 Visual FoxPro 需要知道的 1002 件事》中的一章以后，我大受启发，从而开发了一个 VFP 工具，这个月的文章就是成果。

数据驱动的过程

在 MegaFox 一书描述从 VFP 生成 HTML 的那一章中认为最好的办法是以数据来驱动这个过程，生成 HTML 的代码不仅要从中读取内容，还要读取这些内容的排列顺序、甚至是要执行生成工作的类。

我决定为此建立三个表：

- **PAGES**：指定要生成的 Web 页；
- **Content**：包含关于要将在一个网站中用到的部件的信息；
- **PageContent**：它起着作为其它两个表的连接表的作用，指定哪个部件要出现在哪个 Web 页上、以及它们应该怎样去格式化（为了能够真正的可重用，一个部件的格式应该与它的内容相分离。）。这些表的内容见表 1、表 2 和表 3。

表 1.Pages.dbf 指定要生成的 Web 页。

字段	类型	用途
ID	I(自动增长)	主关键字
PAGE	C(30)	页面的名称

TITLE	C(60)	页面的标题
TEMPLATE	C(60)	用于静态内容的模板页的名称（例如<HEAD>部分）

表 2.Content.dbf 包含关于要在一个网站中用到的部件的信息。

字段	类型	用途
ID	I(自动增长)	主关键字
HEADER	C(60)	部件的 header
CONTENT	M	部件的 BODY
FOOTER	C(60)	部件的 FOOTER
HTML	L	如果内容包含不应被格式化的 HTML，则为 .T.

表 3.PageContent.dbf 指定哪个部件显示在哪个页上

字段	类型	用途
ID	I(自动增长)	主关键字
PAGEID	I	PAGE 表中某个记录的 ID
CONTENTID	I	CONTENT 表中某个记录的 ID
ORDER	I	部件应该在特定某个网页上显示的顺序
PARENT	I	如果本部件是另一个部件的子部件，则记录下那个部件在 PageContent 表中的 ID
CLASS	C(30)	用于交付这个部件的类的名称
LIBRARY	C(30)	在 CLASS 中指定的类包含在哪个类库中
PROPERTIES	M	给在 CLASS 中指定的类设置属性的代码

有些东西需要解释一下。首先，我决定不是给每个特定的 Web 页都生成全部的 HTML，而是只生成每个页上彼此不同的部分。例如，大多数的网站都有一个统一的样式，因为它们在每一页上都使用了许多相同的部件，比如公司的 logo、或者一个向导工具栏。为每个页都生成这样的静态 HTML 是没有意义的，所以 PAGES.DBF 中有一个 TEMPLATE 字符，它指定用于该页的一个 HTML 模板页。在这个文件中的一个占位符将被替换为生成的 HTML。这样一来，要改变整个网站的样式，只需要修改这个模板文件、或者换一个新的模板文件就可以了。

其次，为了表现的统一性，大多数布局都将使用定义在一个 CSS(cascading style sheet，层叠样式表)文件中的样式。象 Microsoft Word 中的样式那样，CSS 样式用于将象字体和颜色这样的统一格式应用在 HTML 元素上。稍候我们将讨论到的某些类上拥有一些属性，你可以把 CSS 的样式名称保存在这些属性中。

还有一个要解释的小问题是：我的 Web 页有一些小段的文本，它们将被放在一个表格的某些列中。为了方便称呼，我把这些部件叫做“Snippets”(片断)。在图 1 的网页中，GoldMine Speiific Feature、Ordering 和 Download 部分就是一些 Snippets。

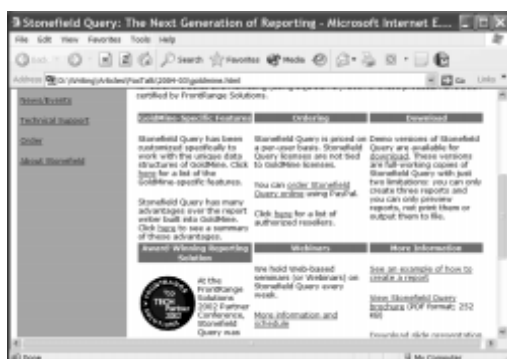


图 1

这些表的设计不仅允许可重用部件（一个指定的部件可以出现在多个页面上，只要在 **PageContent** 表中放入任意多条相同 **ContentID** 值的记录就可以了。），而且一个部件在多个不同的页面上的表现也可以大不相同，因为用于交付一个部件的类是指定在 **PageContent** 表而不是 **Content** 表中的。

HTML 交付类

交付在 **Contents** 表中指定部件的类被定义在 **SFHTML.VCX** 类库中。**SFHTML** 是该类库中所有其它类的父类。它是一个 **Custom** 基类的子类，只有一个方法 **Render**，该方法抽象在这个类里。

SFHTMLComponent 是 **SFHTML** 的一个子类，用于为一个部件交付 **HTML**。它拥有一些用于保存某个部件的 **Header**、**Body** 和 **footer** 的属性（**cHeader**、**cBody** 和 **cFooter**），还有一些是用于这些项目的 **CSS** 样式的类（**cHeaderClass**、**cBodyClass** 和 **cFooterClass**）。尽管在 **HTML** 中空格是无关紧要的，或许你还是想要为某个部件的 **HTML** 设置一些缩进之类的设置，以便增加文档的可读性。**nIndent** 属性指定 **HTML** 的缩进量，而 **nChildIndent** 属性则指定特定的结构（例如在一个 **<TABLE>** 中的 **<TR>** 和 **<TD>** 标志）还要再缩进多少。**ICRAfterContent** 属性决定是否要在 **HTML** 的每一行后面加上一个回车和换行。

Render 方法为部件生成和返回 **HTML**。其实大多数情况下它只是将任务转交给其它的方法，这样，从这个类派生子类就变得轻松了。由于版面的限制，我们就不讨论其它方法了，但这些方法都很类似。注意 **Render** 方法将 **IcHTML** 变量声明为 **PRIVATE** 而不是 **LOCAL**，这是因为这个类将使用 **TEXTMERGE** 来将部件的 **HTML** 输出给这个变量，而我們希望能避免变量的有效范围问题。

```
private IcHTML
```

```
local IoException as Exception
```

```
with This
```

```
    IcHTML = "
```

* 设置部件的文本合并，并输出它的各个部分

```

try
    .SetupTextMerge()
    .RenderHeader()
    .RenderBody()
    .RenderFooter()

```

* 不处理错误，只将错误抛出给下一层的错误处理器

```

catch to loException
    loException.Procedure = This.Name + '.' + ;
    loException.Procedure
    throw

```

* 关闭文本合并，并清理环境

```

finally
    .CleanupTextMerge()
endtry

```

* 在 HTML 的尾部添加一个中止行

```

    lcHTML = lcHTML + .cLineBreak
endwith

```

```

return lcHTML

```

SFHTMLContainer 类为一个或多个部件提供了一个容器。所有的部件都储存在该类的一个集合 **oContentCollection** 属性中（这个集合在 **INIT** 方法中建立实例）。**SFHTMLContainer** 类的 **Render** 方法简单的调用各个部件的 **Render** 方法，然后把它们的结果合并在一起。

```

local lcHTML, ;
    loContent
with This
    lcHTML = "
    for each loContent in .oContentCollection
        loContent.nIndent = .nIndent + .nChildIndent

```

```

        lchTML = lchTML + loContent.Render()
    next loContent
endwith
return lchTML

```

SFHTMLSnippetContainer 类是 **SFHTMLContainer** 类的一个子类, 用于存储部件的片断。它有一个 **nColumns** 属性, 用于指定这些片断在该页上要填充多少个列。它的 **Render** 方法要比 **SFHTMLContainer** 类的 **Render** 复杂的多, 因为它必须要去建立一个 **HTML** 表来存储这些片断, 并将每个片断的 **HTML** 放到表格的一个单元格中去。**cSnippetContainerClass** 和 **cSnippetCellClass** 属性指定用于表格和单元格的 **CSS** 样式。

```

local lIndent1, ;
    lIndent2, ;
    lIndent3, ;
    lIndent4, ;
    lchTML, ;
    lRows, ;
    lRow, ;
    lColumn, ;
    lTopic, ;
    loContent, ;
    loException as Exception

```

with This

* 指定每一层前的缩进量以及每一行末尾要加上的字符串

```

try
    lIndent1 = space(.nIndent)
    lIndent2 = space(.nIndent + .nChildIndent)
    lIndent3 = space(.nIndent + .nChildIndent * 2)
    lIndent4 = .nIndent + .nChildIndent * 3
    .cLineBreak = .GetLineBreak()

```

* 开始处理放置片断的 **HTML** 表格。

* 弄清楚我们到底需要多少行。

```

lcHTML = lcIndent1 + ;
'<table class="' + .cSnippetContainerClass + ;
"">' + .cLineBreak

```

```

InRows = ceiling(.oContentCollection.Count/.nColumns)

```

- * 处理每一行。处理时，先建立一个行，然后通过输出一个单元格、
- * 并让相应的 Topic 对象将它的输出交付给单元格来处理每一个列

```

for InRow = 1 to InRows
    lcHTML = lcHTML + lcIndent2 + '<tr>' + .cLineBreak
    for InColumn = 1 to .nColumns
        InTopic = (InRow - 1) * .nColumns + InColumn

        if InTopic <= .oContentCollection.Count
            lcHTML = lcHTML + lcIndent3 + ;
            '<td class="' + .cSnippetCellClass + ' "' + ;
            'width="' + transform(int(100/.nColumns)) + ;
            '%">' + .cLineBreak
            loContent = .oContentCollection.Item(InTopic)
            loContent.nIndent = InIndent4
            lcHTML = lcHTML + loContent.Render()
            lcHTML = lcHTML + lcIndent3 + '</td>' + ;
            .cLineBreak
            endif InTopic <= .oContentCollection.Count
        next InColumn
        lcHTML = lcHTML + lcIndent2 + '</tr>' + .cLineBreak
    next InRow

lcHTML = lcHTML + lcIndent1 + '</table>' + .cLineBreak

```

- * 不处理错误，只将错误抛出给下一层的错误处理器。

```

catch to loException
    loException.Procedure = This.Name + '.' + ;
    loException.Procedure

```



```
        throw
    endtry
endwith
return lcHTML
```

单个的片断的标题是与 **SFHTMLSnippetTopic** 类一起给出的，**SFHTMLSnippetTopic** 类是 **SFHTMLComponent** 类的一个子类，它的 **cHeaderClass**、**cBodyClass** 和 **cFooterClass** 属性都已经用恰当的 CSS 样式填充好了。

SFHTMLCalendar 是 **SFHTMLComponent** 类的一个更复杂的子类，它用于提供 **Calendar**(日历)。它的 **nMonth** 和 **nYear** 属性包含着日历的月份和年份。其它的许多属性，例如 **cCalendarTableClass** 和 **cCalendarDayClass** 属性，包含着用于日历的相应部分要使用的 CSS 样式。为每一天显示的内容来自于一个表，该表的名字在 **cTable** 属性中指定。这个表中必须有一个保存发生事件的日期字段，这个日期字段的名称会被放在 **cDataField** 属性中；还必须有一个保存对事件的描述的字段，该字段的名称在 **cContentField** 属性中指定。你还可以在 **cFilter** 属性中指定一个过滤表达式，以对该表进行过滤（例如，如果你有一个包含所有的培训类型的表，而又只想显示在指定的日程内的那些类型）。由于版面的限制，我就不讨论这个类的代码了，请自行从附带的源代码中研究。

我们要讨论的最后一个类是 **SFHTMLPage**。这个基于 **SFHTML** 的类负责给出整个 HTML 页。它的 **Render** 方法是数据驱动的，因为它从我们前面提到的 **CONTENT**、**PAGES** 和 **PAGECONTENT** 表中去读取数据然后进行输出。它首先搞清楚需要哪些部件、并把这些部件添加到一个集合中，然后遍历集合中的每个对象，让每个对象给出自己的 HTML 结果。最后，它把来自所有对象的 HTML 结果组合起来，插入到由 **Pages** 表的 **TEMPLATE** 字段指定的模板文件中合适的位置。详情见附带的源代码。

根据你自己的需求，还有许多你可以建立的其它类。例如，你也许想要在一个表格或者一个 **Chart** 图形中显示某些数据（这些数据来自一个 **VFP** 或者 **SQL Server** 表）。通过派生 **SFHTMLComponent** 类的子类，你能够建立起能交付你需要的任何形势内容的类。

检查

现在我们已经看了许多代码了，再让我们来看看它们是如何工作的。为了给我的网站生成一个页面，我给 **PAGES** 表添加了一条记录，以指定要生成的页面的名称；同时，我还在 **CONTENT** 表中建立了几个标题，每一个表示一个将要在这个页面上显示的部件。然后，我向 **PAGECONTENT** 表添加了一些记录，这些记录为那些标题和这个页面建起了联系，并指定了用于每个标题的部件的类型。最后，我做了一个名为 **TestPage.prg** 的小测试程序，以生成这个页面，并把它显示在我默认的浏览器中。图 1 中显示的就是从这个程序中生成的页面。

```
set classlib to SFHTML
```

```

loPage = createobject('SFHTMLPage')
lcHTML = loPage.Render('goldmine.html')
lcFile = sys(5) + curdir() + 'goldmine.html'
strtofile(lcHTML, lcFile)
declare integer ShellExecute in SHELL32.DLL ;
    integer nWinHandle, string cOperation, ;
    string cFileName, string cParameters, ;
    string cDirectory, integer nShowWindow
ShellExecute(0, "", lcFile, "", "", 0)

```

用几乎一样的代码（区别仅在于生成页面的名字不同），**TestCalendar.prg** 生成了图 2 中的日历。

生成一个网站

通过使用在 **SFHTML.VCX** 中的类，你至少有两种方法可以生成一个网站的页面。首先，你可以通过遍历 **PAGES** 表，并生成表中指定的每一页来生成一个静态的网站。**GenerateWebSite.prg** 就是干这个的：

```

local loPage, ;
    lcFile, ;
    lcHTML
set classlib to SFHTML
loPage = createobject('SFHTMLPage')
use PAGES
scan
    lcFile = trim(PAGE)
    lcHTML = loPage.Render(lcFile)
    strtofile(lcHTML, lcFile)
endscan
use

```

另一种更有趣的办法是动态生成那些页面。用户的浏览器将访问一个 **ASP** 或者 **ASP.NET** 页面，该页面会建立某个 **VFP COM DLL** 的实例，由该实例来提供需要的 **HTML** 内容。这里是一个会为指定在 **Pages.dbf** 表中的每个页生成 **ASP** 页的程序（**GenerateASP.prg**）。每个 **ASP** 页有着几乎一致的内容（当然这也是可以改变的）：它建立一个在 **WebPages.DLL** 中的 **WebPages** 类的实例，并用在 **Pages.dbf** 中的某个页面的名字作为参数调用类实例的 **GeneratePage** 方法。

```
local lcHTML, ;  
    loPage, ;  
    lcFile
```

* 生成 ASP 内容。

```
set textmerge off  
text to lcHTML noshow  
    <%Language="VBScript"%>  
    <html>  
    <head>  
    <title><<trim(TITLE)>></title>  
    <link rel="stylesheet" type="text/css"  
    href="stonefield.css">  
    </head>  
    <body>  
    <%  
    dim WebApp  
    set WebApp = Server.CreateObject("WebPages.WebPages")  
    Response.Write(WebApp.GeneratePage("<<trim(PAGE)>>"))  
    %>  
    </body>  
    </html>  
endtext
```

* 现在建立 ASP 页

```
set classlib to SFHTML  
loPage = createobject('SFHTMLPage')  
use PAGES  
scan  
    lcFile = forceext(trim(PAGE), 'asp')  
    strtofile(textmerge(lcHTML), lcFile)  
endscan  
use
```

WebPages.DLL 是由 **WebPages.PJX** 编译而来的。这个项目包含 **SFHTML.VCX** 和 **WebPages.PRG**，这个 **PRG** 中有着 **WebPages** 类的定义。**WebPages** 类简单的建立 **SFHTMLPage** 的实例，并用指定的页名称参数调用它的 **Render** 方法。

```
define class WebPages as Session olepublic
    function GeneratePage(PageName as String) as String
        local loPage, ;
        lcHTML, ;
        loException as Exception

    try
        set path to justpath(_vfp.ServerName)
        set classlib to SFHTML
        loPage = createobject('SFHTMLPage')
        lcHTML = loPage.Render(PageName)
    catch to loException
        comreturnerror('WebPages', 'Error #' + ;
            transform(loException.ErrorNo) + ;
            ' occurred in line ' + ;
            transform(loException.LineNo) + ' of ' + ;
            loException.Procedure + ': ' + ;
            loException.Message)
    endtry
    return lcHTML
endfunc
enddefine
```

要测试这个程序，请运行 **GenerateASP.PRG**，然后把生成的 **ASP** 页拷贝到你的 **IIS** 网站的根目录（例如 **C:\inetpub\wwwroot**）或者其它的虚拟路径里面去。打开 **WebPages.PJX** 项目，并从中编译一个多线程 **DLL**。然后打开你的浏览器并输入 "**http://localhost/goldmine.asp**"。你将看到图 1 中显示的那个 **Web** 页面。

总结

这个月，我们看了一套作为一个可重用的 **HTML** 部件的集合、让你可以生成一个 **Web** 页的类。不过，当你自己动手做的时候，你很快会发现手工在 **Browse** 窗口中生成用于驱动这些类的表并不是一件愉快的工作。在以后的文章中，我将会讲述一个用于这些表的前台用户界面，以更轻松的完成这些任务。

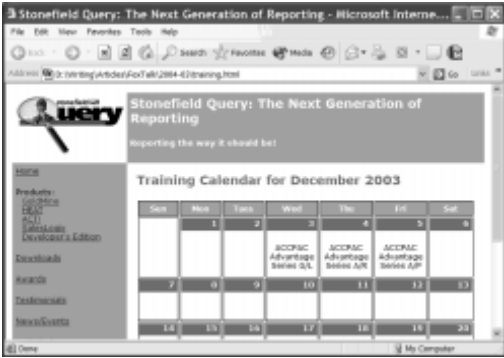


图 2

附件：405HENNIG.ZIP

来自 VFP 开发团队的 Tips

原著:微软开发团队成员

翻译: LQL.NET

雷德蒙得微软开发园区的 41 号楼这些日子里非常的繁忙, Visual FoxPro 开发组正在日以继夜的忙于 VFP 9 的新功能、修正 Bug、写文档和示例代码、以及穷举测试。百忙之中, 他们也偷空在 Foxtalk 2.0 上开了一个专栏, 每期介绍一些 VFP 新功能的使用方法, 这一期讲的就是关于 VFP 9 中 SQL Select 语句的一些新功能...

以下是来自 Microsoft Visual FoxPro 开发组月刊的第一部分。如需 Visual FoxPro 更多信息和资料可以访问 <http://msdn.com/vfoxpro>。如要有关 Visual FoxPro 9.0 的更多信息, 可以参考各种月刊在 <http://msdn.com/vfoxpro/letters>。

在 Visual FoxPro 9.0 里有许多新增的内容, 包括 SQL 语言的重大改变。不久, 你就可以下载 Visual FoxPro 9.0 公开测试版去体验下新的功能。如果要更详细地了解 VFP9 Beta, 就请关注 Foxtalk 6 月刊吧。

看, 在 6 月到来之前, 为吊起你们的胃口, 我们提供了一些 VFP9 SQL 语言方面样例, 以及 VFP9 对前期版本的一些改动。

Joins

VFP9: 无限制

VFP8: 限 9 个

子查询

VFP9: 无限制

VFP8: 限 9 个

UNIONs

VFP9: 无限制

VFP8: 限 9 个

连接表数量

VFP9: 无限制

VFP8: 限 30 个

IN()

VFP9: 无限制

VFP8: 有 24 位限制。实际上仍然受限于 **SYS(3055)** 的设置，**SYS(3055)** 设多少就限多少。

VFP9: 允许 **UNION** 子句中包含 **OrderBy** 使用的字段

在 **VFP8** 中这是不行的，你必须用列号代替。

优化 TOP N 的性能

在 **VFP8** 或更早的版本中，**TOP N [PERCENT]** 是这样运行的：所有记录先排序，然后由 **TOP N [PERCENT]** 萃取。在 **VFP9** 中，我们从排序进程中尽早地除掉记录，而不让她进入 **TOP N [PERCENT]** 的萃取进程。

优化 LIKE “sometext%” 的性能

要查找 **VFP** 本地数据某个字段由 “sometext” 开头，我们可以用 **Rushmore** 优化的查询：

```
SELET * FROM table1 WHERE somefield = “sometext”
```

然而，许多 **SQL** 后端并不支持这样的句子。 **VFP9** 现在使用更通用的表达式：

```
SELECT * FROM table1 WHERE somefield LIKE “sometext%”
```

FROM 子句支持 子 SELECT

我们经常提到衍生表。衍生表就是 **FROM** 子句中的 **SELECT** 段生成的别名。

```
SELECT ... FROM (SELECT ...) ...
```

允许 无关联子查询 使用 TOP N/ORDER BY

VFP9 现将允许 无关联的子查询 使用 **TOP N/ORDER BY**。如果使用 **TOP N**，同时也该使用 **ORDER BY**。这是这一许可的唯一条件。

```
SELECT ... WHERE ... ;
```

```
(SELECT ... TOP nExpr ... FROM ... ORDER BY ... ) ...
```

允许 无关联子查询 使用 GROUP BY

许多查询可以通过“执行一次子查询、将子查询的一个或多个结果替换入外部查询的 **Where** 子句中”来运算出结果。在包含一个相关的子查询的查询（以“重复查询”而闻名）中，子查询的值依靠外部查询提供。这就意味着子查询是被重复执行的，对外部查询的每一行都要执行一次子查询。

```
SELECT ... WHERE ... ;
```

```
(SELECT ... WHERE ... GROUP BY ...) ...
```

允许多个子查询嵌套

语法:

```
SELECT ... WHERE ... ;  
    (SELECT ... WHERE ... (SELECT) ... ) ...
```

允许在 **SELECT** 字段列表中使用子查询

VFP9 中 **SELECT** 字段列表可以包含子查询产生的结果, 比如:

```
SELECT T1.f1, ;  
(SELECT f2 FROM foo2 T2 WHERE T2.f1=T1.f1) AS foo2 ;  
FROM foo1 T1
```

允许在 **INSERT INTO ...SELECT** 中使用 **UNION**

我们将允许在 **INSERT INTO** 的 **FROM** 段中使用 **UNION** :

```
INSERT INTO ... ;  
(SELECT ... FROM ... UNION SELECT...) ...
```

允许在 **UPDATE SET** 列表中使用 子查询

```
UPDATE foo1 SET f2=100+(SELECT f2 FROM foo2 ;  
WHERE foo2.f1=foo1.f1) WHERE f1>5
```

对 **DELETED()**标记 进行 **Rushmore** 优化

我们将优化含有 **DELETED()/NOT DELETED()/FOR DELETED()**表达式的索引, 还有, **DELETED** 标记中含有 **MAX()/MIN()** 时, 我们也将优化她。

不久我们会提供更多的源码例程

在下一期 **Foxtalk**, 我们将提供许多 提示/技巧, 包括源码, 让你马上可以在 **VFP9 Beta** 中使用。

类的亲密接触

原著: Andy Kramek / Marcia Akins

翻译: CY

VFP8.0 为我们提供了一个代码参考工具 (**Code References**)，它可以巧妙地对你的代码进行字符查找。然而，它还是无法帮助你完整找出某个指定的类 (你已经知道其类名) 的所在。**AVCX()**类函数可以对以 **VCX** 定义的类正常运行，而代码参考却可以运行于大多数基于 **PRG** 的类，伸是没有任何工具可以同时完成这两件事。在本月的栏目里，**Andy Kramek** 和 **Marcia Akins** 将着手一项任务：创建一个可以列示所有类的小工具，无论它们是怎么定义的，并可以存储于一个表内，并可以用于查找和打开类，不论是任何类名。

Marcia: 我已经忘记如何去理解别人写的代码的困难性，这些所使用类的类名并没有表示出其所在的位置。

Andy: 你说的是什么意思？我从未见过根据类所处的位置来命名的，除非你说的是那些如**CodeBook**模型，它的首个字母表示在体系里的层次级别 (**aXxxx**=应用层，**iXxxx**=隔离层，等等)。但是同样它也未告知所在的类库，或所驻留的磁盘位置。那**VFP8.0**的代码参考工具又是怎么样？

Marcia: 它并不是很有帮助，因为我已经知道类名。我想知道的是在这**40**个类库和**35**个程序里所包含定义的类，它们分散在半打的目录里，有那些是做过补丁的。理想的，我喜欢可以直接输入类名，并可以有多种办法找到对应的类库。

Andy: 对了，我们有一个**Steven Black**于**1999**年在我们和他一起工作时设计的程序。它在一个表内存储类的有关信息，并利用存储的数据来作查找和打开类。

Marcia: 是的，但是它使用了**AVCX()**类来对表填充必要的信息。这是对于大多数的可视类，但是对于以程序文件来定义的种类是没有用处的。

Andy: 我相信我们可以为此做些什么。让我们从最容易的资料开始。**AVCX()**类使我想起了，从我使用它已经一段时间了。

Marcia: 你可以传入数据组名和可视类库名。函数将以指定类库中类的信息填充数组。联机帮助里的主题已经解释得非常好，**表1**已经列出主要内容。

Andy: 我已经说过的，非常全面。我见不到我们需要的其他东西。其实，我们不需要这里面的**第5、6、7**栏。然而，我们需要一个表来存储这些信息。

Marcia: 注意，所有的路径信息都是以相对路径方式返回的。如果你假定所有的内容都是相关于单个目录，那么这是可行的，但是我认为如果我们存储完整的指定路径和类库的文件名将会是更有益。然后你可以在磁盘的任何位置来使用这

个表，并且它也可以正常运行。

Andy: 听起来有道理。那我该为路径栏设定多少长度？100个字符？

Marcia: 对我来说我们最好是使用备注字段来存储完整的指定路径。毕竟，它可以得到更好的长度——特别是当你是使用默认的MS安装位置时。

Andy: 听起来我有计划了。让我调用那个表ClassInfo并包含入数据库ClassInfo。

表1: AVCX()类返回的信息内容

Col	Contains (内容)	VCX field name (VCX字段名)
1	Class name 类名	OBJNAME
2	Base class of the class 该类的基类	BASECLASS
3	Parent class name 父类名	CLASS
4	Relative path and file name of the parent class library 父类库的相对路径和文件名	CLASSLOC
5	Relative path and file name of the bitmap for a custom class icon 定制类图标的位图的相对路径和文件名	RESERVED4
6	Relative path and file name for a custom Project Manager or Class Browser class icon 定制项目管理器或类浏览器类的图标的相对路径和文件名	RESERVED5
7	ScaleMode of the class, Pixels or Foxels 类的度量单位(ScaleMode): Pixels(点阵)或Foxels	RESERVED6
8	Description of the class 类的描述	RESERVED7
9	Relative path and file name for the #INCLUDE file for the class 类的头文件(#INCLUDE)的相对路径和文件名	RESERVED8
10	User-defined information for the class 类的用户自定义信息	USER
11	Logical true (.T.) if the class is OLEPUBLIC, otherwise logical false (.F.) 如果类是OLEPUBLIC就为真(.T.)，否则就是否(.F.)	RESERVED2

表2列出我们所需要的东西。

Marcia: 好象都有了，现在，这些东西是怎么运行的？

Andy: 现在，我假设你已经指向一个起始目录，随后它将会创建一个所有子目录的列表，并按序对每一个进行处理。然后，再定位到那个目录后，它将利用ADIR()来获取所有的VCX和PRG文件列表，并按序对每一个获取数据。我能发现的唯一问题就是如何实现必要的递归以获取目录列表。

Marcia: 等一下，我们是怎么创建它的？是类，表单，还是什么？

Andy: 好的，让我们来创建一个小表单。你可以使用基类来创建，并且把保存在VFP的主目录，以使得一直都可用。现在，回到它怎么运行上来。

Marcia: 好的，我推测首先是指定一个你想要查找的目录。

Andy: 并且自动处理所有的子目录吗？

Marcia: 是的，我是这样认为的。如果我们把指定的目录作为“根”目录，我们可以利用ADIR()来找出所有的文件，也可以找出所有的子目录。然后我们有办法让它对每一个子目录调用自身。很容易！

Andy: 如果象你说的这样，可是我却不得不说，我一直产生递归错乱。首先，我们需要有办法来确定是覆盖已存在的表还是只是作简单追加。

Marcia: 为什么？我是假设我们一直都是覆盖表。

Andy: 那么，如果想要获取一个项目的所有类，但是也有部分类库是与其他项目共享的时候会发生什么？你该如何只获取相关的类进入表？

Marcia: 哈，我不能肯定，我听听你的。

Andy: 请见图1里的目录树，你就会明白我的意思。

Marcia: 哈，现在我明白了。如果我所要的那个项目里的类，它也可能是公共的资料，我将不得不以公共目录作为“根”目录，然后为此项目再次运行。你是对的，我们需要允许两种可能：要么追加到已存在的数据集里，要么覆盖整个表。

Andy: 为此我们需要一个属性，几个自定义属性如表3的所示。

Marcia: 为什么我们需要错误日志？

Andy: 因为所有我们所做的事都有可能出错。比如，VCX文件可能是损坏的，或是VCT文件已经丢失。我们需要知道是怎么回事，为什么不？

Marcia: 我猜的。现在，我们需要一个控件方法来运行整个过程。我们从“创建列表(Build List)”按钮（见图2）里调用它，并叫它为Setup()。

Andy: 这个方法程序将会做这些，对吗？

Marcia: 首先，它将会确认我们存在指定的起始目录。接着，它将会处理在必要的时候覆盖ClassInfo.dbf。然后，它将会改变到指定的起始目录并调用FindClasses()方法，传送起始目录作为参数。最后，它将会检查错误日志，刷新表格并作整理。

表2: ClassInfo.dbf的结构

Field name	Type	Size	Description(描述)	Source(来源)
iClassPK	I	4	Primary key 主键	Auto-Incrementing Column 自增量字段
cClass	C	50	Name of the class 类名	Array Column 1 数组列1
cBaseClass	C	30	VFP base class from which the class derives 类所派生自的VFP基类	Array Column 2 数组列2
cClassLib	C	50	Current library file name 当前类库名	Input parameter 输入参数
cLibPath	M	4	Fully qualified path to the current library file 当前类库文件的完整路径	Current working directory 当前工作目录
cParClass	C	50	Name of the parent class 父类名	Array Column 3 数组列3
cParLibPath	M	4	Fully qualified path to the parent library file 父类库文件的完整路径	Array Column 4 数组列4
cIncFile	C	50	Name of the include file (if any) 头文件名（如果有）	Array Column 9 数组列9
cIncFilePath	M	4	Fully qualified path to the include file (if any) 头文件的完整路径（如果有）	Array Column 9 数组列9
cDescrip	M	4	Class description 类描述	Array Column 8 数组列8
IOlePublic	L	1	OLEPUBLIC flag OLEPUBLIC标识	Array Column 11 数组列11

表3: 自定义属性

Name	Type	Purpose（目的）
IOverWrite	Logical	Flag to indicate whether to overwrite ClassInfo.DBF or append to it 标识，以表明是覆盖ClassInfo.dbf还是追加它
cHomeDir when finished)	Char	Path to the directory from which the form was started (so we can return there 表单启动时的目录路径（以使得我们可以在完成时得以返回）
cStartDir	Char	Path to the directory to use as root 作为“根”目录的目录路径
cErrorLog HOME()+“ClassErr.txt”)	Char	Fully qualified path and file name of the error log (default to 错误日志文件的完整路径和文件名（默认为Gome()+“ClassErr.txt”）

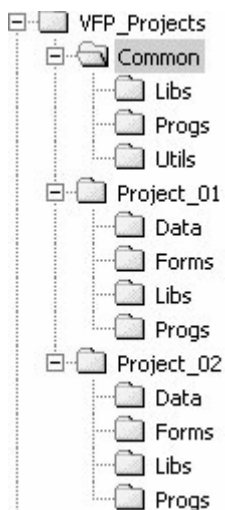


图1：示例的目录结构

Andy: 看起来很简单。FindClasses()是什么？

Marcia: FindClasses()是个关键方法程序，并且它将会递归调用。它会在我们所要处理的每一个目录调用，并对于每一个目录，它将会使用ADIR()三次。首先，它将会检查在当前目录下的所有VCX文件。如果它有找到，它将会对每一个文件调用AddVCX()方法程序。接着，它将会检查PRG文件。如果有找到，也对每个文件调用AddVCX()方法程序。最后，它将会检查子目录。如果有找到，它将会对每个子目录调用它自身，并传递子目录名作为参数。

Andy: 我明白了。这里有几点需要考虑。我希望你已经确认在方法程序里的所有变量都被明确的申明为LOCAL。我仍然记得试图在递归调用象这样的方法程序里为找出错误时，使用到了计数器。然而发生的却是计数器没有被申明为LOCAL并导致可怕的错误。

Marcia: 当然每个都被申明为LOCAL——你以为和你谈话的是谁？你的第二点是什么？

Andy: 这里有一个限制，象你这样可以调用方法程序的次数。我知道我们并没有使用直接的DO命令，但是DO命令的嵌套限制也同样适用于方法调用。

Marcia: 是的，但是限制是128层。我并不认为我们在递归遍历一个目录结构时将会达到这样的层数。

Class Name	Class Library	Parent Class
object	MSOutils.VCX	None
webctrl	MSOutils.VCX	None
webbuilder	MSOutils.VCX	webbasebuilder
webbuilder1	MSOutils.VCX	webbasebuilder
webbuilder2	MSOutils.VCX	webbasebuilder
webbuilder4	MSOutils.VCX	webbasebuilder
webbuilder7	MSOutils.VCX	webbasebuilder
webbasebuilder	MSOutils.VCX	webbasebuilder
webpage	MSOutils.VCX	webbasebuilder
column	BASE.PRG	column
cursor	BASE.PRG	cursor
databaseenvironment	BASE.PRG	databaseenvironment
header	BASE.PRG	header
webctrl	BASE.PRG	webctrl
webpagecontrol	BASE.PRG	webpagecontrol
application	BASE.PRG	application
page	BASE.PRG	page
webctrl	BASE.PRG	webctrl
webpage	BASE.PRG	webpage
exception	BASE.PRG	exception

图2：类信息表单

Andy: 128?真的吗?我当然同意128层是完全足够的。我疑惑的是为什么我会认为限制是32。

Marcia: 谁这么说?至少从VFP6.0开始就已经是128。不管怎样,这是FindClasses()方法的代码:

```
LPARAMETERS tcDirectoryName
LOCAL ARRAY laLibs[1], laDirs[1]
LOCAL lcCurDir, lnLibs, lnCnt, lnSubDirs

*** Check that the directory is valid
*** Files without extensions may exist!
*** 检查目录是否有效,无扩展名的文件也可能存在
IF NOT DIRECTORY( tcDirectoryName )
RETURN
ENDIF

*** Save the current location
*** 保存当前位置
lcCurDir = FULLPATH( CURDIR() )

*** Change to the required directory
*** 改变到要求的目录
CD ( tcDirectoryName )

*** Find all the vcx's in this folder
*** 在这个文件下查找所有的VCX文件
lnLibs = ADIR( laLibs, '*.vcx' )
IF lnLibs > 0
*** Process each vcx in the array
*** 处理数组内的每一个VCX
FOR lnCnt = 1 TO lnLibs
ThisForm.AddVCX( ALLTRIM( laLibs[lnCnt,1] ))
ENDFOR
ENDIF

*** now see if we have any prgs to process
*** 现在检查是否有PRG文件需要处理
lnLibs = ADIR( laLibs, '*.prg' )
IF lnLibs > 0
*** Process each program in the array
*** 处理数组内的每一个程序
FOR lnCnt = 1 TO lnLibs
ThisForm.AddPRG( ALLTRIM( laLibs[lnCnt,1] ))
```

```

ENDFOR
ENDIF

*** Finally, go through any subdirectories
*** 最后，转到处理所有子目录
InSubDirs = ADIR( laDirs, '.*', 'D' )
FOR InCnt = 1 TO InSubDirs
IF NOT( LEFT( ALLTRIM( laDirs[InCnt,1] ), 1 ) == '.' )
*** Here is the recursive call!
*** 这里是递归调用
Thisform.FindClasses( ALLTRIM( laDirs[InCnt,1] ) )
ENDIF
ENDFOR

*** When done change back to the current directory
*** 当完成后改回到当前目录
CD ( lcCurDir )

```

Andy: 这是令人惊讶的小代码，真的，考虑到遍历目录树的每一个分支。

Marcia: 递归调用的一个好处就是你可以直接重用它自己。

Andy: 我想AddVCX()方法程序看起来很简单。因为你传递类库名作为参数，我推测它是利用AVCXClasses()来获取信息。

Marcia: 是的。我们有类库文件名，并已经转到类库所在的目录，所以我们对文件名和位置没有困难。我们从由AVCXClasses()创建的数组里别的任何东西，都列在表1里。我所感兴趣的是你怎么建议从PRG定义的类里获取同样的信息。AddPRG()方法程序里是什么？

Andy: 正如它所运行的，VFP的纯字符处理函数可以使得这非常容易。首先，我们利用FILETOSTR()和ALINES()来获取程序代码并放入数组。然后，我们必须要做的是循环处理数组以查找DEFINE CLASS语句。

Marcia: 但是附加信息呢？比如父类等等。

Andy: 事实上，那是相当简单的。可以考虑DEFINE CLASS命令的语法。基本上它可以有四种格式：

- DEFINE CLASS myclass AS parentclass
- DEFINE CLASS myclass AS parentclass OLEPUBLIC
- DEFINE CLASS myclass AS parentclass OF parentclassfile
- DEFINE CLASS myclass AS parentclass OF parentclassfile OLEPUBLIC

Marcia: 我想我明白你所说的意思。所以你可以检查“DEFINE CLASS”的每行头12个字符，因为它对于任何以代码定义的类都是如此的。但是如果有人没有对他或她的代码作象我们所要那样的格式化，那将会发生什么？我的意思，猜想类的定义代码格式为如下这样：

```
DEFINE CLASS MyClass ;  
AS ParentClass ;  
OF ParentClassFile ;  
OLEPUBLIC
```

Andy: 写这样代码的人应该感到羞愧！不管怎样，我想这里的关键是你可以修改你的代码以测试分号(;)以作为每行代码的末个字符，并追加行一直到我们找到不是以分号作为终止符。然后，用CHRTAN()去掉分号，你就可以继续了。但是，我必须说我从未见到如你上面所写的DEFINE CLASS代码。

Marcia: 每件事都有第一次，但是我趋向于认同你的观点。但我并不认为有必要现在就把这代码加入到我们的小表单里。

Andy: 好，因为我并不计划这样做。不管怎样，从DEFINE CLASS行里我们可以提取出类名。因为我们的代码转换了整行为小写，类名一直都是在单词“class”和“as”两者之间。STREXTRACT()函数使这变得非常容易：

```
lcClassName = ALLTRIM(STREXTRACT(lcLine, 'class ', ' as '))
```

Marcia: 非常好。我发现你在“as”前后加上空格以避免任何可能的混淆。现在，父类和父类文件名该怎么样了？


Andy: 好的，如果在行内有“of”，就必须把父类同它的文件名区分开来。于是我们为此测试并再次使用STRTEXTACT()提取我们想要的信息：

```
lcParentDef = STREXTRACT( lcLine, ' as ', ' of ' )  
lcParentClass = JUSTSTEM( lcParentDef )  
lcParentFile = STREXTRACT( lcLine, ' of ', '' )  
lcParentPath = JUSTPATH( lcParentFile )
```

当然，如果没有“of”，我们知道我们对父类文件没有任何的明确参考，并且只能从跟随“as”后面任何东西作为父类名。

Marcia: 非常灵巧。当然，测试OLEPUBLIC定义也非常容易——要么行内有关键字要么没有。但是基类又是该怎么样？

Andy: 这里实在不能太肯定的事是我们可以得到它。我可以提出的唯一的事情是检查以确认父类是否为基类。IsbaseClass()方法程序使用ALANGUAGE()函数以所有的基类来填充数据组属性，并检查父类是否出现于其中。

Marcia: 我想你可以运行一个后处理以跟踪任何未填写基类的类的层次，但是我们把它留给读者作为练习。不管怎样，完成的表单如所示。注意，当你对基于PRG的类点击“修改类(Modify Class)”按钮时，程序编辑器将会打开并停留在包含有DEFINE CLASS命令的行上。

Andy: 巧妙的技巧！你是怎么做的？

Marcia: 这就是MODIFY COMMAND部分内容。很容易忘记这个命令的选项RANGE子句。它可以让你指定一个从文件头开始的偏移范围，并且另一个是结束的偏移。当编辑器打开时两者间的所有内容都将被高亮显示。

Andy: 是的，我含糊地知道这些。你是怎么样取得那些偏移的？

Marcia: 我只是使用了FILETOSTR()和AT()函数。这是表单ModifyClass()方法程序的代码:

```
LOCAL lcClass, lcLib, lcPrg, lnStartPos, lcCmd
WITH ThisForm
*** Hide the form
*** 隐藏表单
.Hide()
*** Grab the Class and Library Name
*** 提取类名和库名
lcClass = ALLTRIM( ClassInfo.cClassName )
lcLib = ADDBS( ALLTRIM( ClassInfo.cLibPath ) ) ;
+ ALLTRIM( ClassInfo.cClassLib )
*** See if we have a vcx or a prg
*** and use the appropriate command
*** 检查是否有VCX或PRG文件, 并使用相应的命令
IF LOWER( JUSTEXT( lcLib ) ) == 'vcx'
*** Modify the Class
*** 修改类
MODIFY CLASS ( lcClass ) OF ( lcLib )
ELSE
*** We got a prg
*** See if we can determine where to position
*** the cursor in the file
*** 找到PRG文件, 检查是否可以确定文件中光标的位置
lcPrg = FILETOSTR( lcLib )
lnStartPos = AT( 'DEFINE CLASS ' + UPPER( lcClass ), ;
UPPER( lcPrg ) )
lnStartPos = IIF( lnStartPos = 0, 1, lnStartPos )
lcCmd = [MODIFY COMMAND ] + lcLib + [ RANGE ] ;
+ TRANSFORM( lnStartPos ) + [, ] ;
+ TRANSFORM( lnStartPos )
&lcCmd
ENDIF
*** Show Form
*** 显示表单
.Show()
ENDWITH
```

Andy: 为什么你传递相同的开始和结束位置值?

Marcia: 因为你要么传递两个参数要不然就会出现语法错误。但是我们并不想高亮显示代码, 因为这会很容易意外删除代码。通过传递两个相同的值给开始和结束位置, 我们可以插入光标的位置, 但并没有任何东西被高亮显示。

Andy: 我明白了。这是明智的——总比说对不起更安全。表单的全部代码包含在随附的下载里。我知道我要立即使用它了。

Marcia: 希望我们的读者也能发现它有用。

Andy Kramek是一名长期的**FoxPro**开发者，**FoxPro MVP**，独立顾问，和**Tightline Computer**公司的共同拥有人，在俄亥俄州的阿克伦城。一名老练的会议发言人，他发表众多，并可以在**CompuServe**论坛 (<http://go.compuserve.com/msdevapps>) 和 **Virtual FoxPro Users Group** (www.vfug.org) 联机找到。
`andykr@tightlinecomputers.com`

Marcia Akins是一名**FoxPro MVP**，独立顾问，和**Tightline Computers**公司的共同拥有人，在俄亥俄州的阿克伦城。一名老练的会议发言人，他发表众多，并且以他的投稿闻名于**CompuServe**论坛(<http://go.compuserve.com/msdevapps>) 和 **Universal Thread** (www.universalthread.com) 。
`marcia@tightlinecomputers.com`