FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2005 年 3 月刊

来自 VFP 开发团队的 TIPS —微软 VFP 开发团队 著 LQL.NET 译

Page.33

这个月的“来自 VFP 开发团队的 TIPS” 专题继续聚焦在 VFP9 SQL 引擎提供的新功能和速度的增强上。VFP9 于 2004 年 12 月向业界发布，开始零售并提供 MSDN 预订。那么你读到这篇文章的时候里面所展示的内容在零售版里应该可以兑现了。

哦，参数？不，是 oParametrs! —Andy Kramek & Marcia Akins 著 Fbilo 译

Page.37

作为程序员，我们极大的依赖于在对象之间传递参数以操作它们的行为、或者报告执行的结果。然而，当我们需要传递多个参数的时候，这个过程很快就变得不灵活起来，因为，在 Visual FoxPro 中，参数必须以一个指定的顺序被传递。此外，我们只能从一个函数或者方法中返回仅一个值，所以，要传回多个值的问题变得很棘手。在这个月的专栏中，Andy Kramek 和 Marcia Akins 讨论了怎样使用一个参数对象来简化你的生活。

VFP9 来了！接下来是什么？

原著：David Stevenson

翻译：LQL.NET

VFP8 发布将近两年之后，微软于十二月向业界推出了 Visual FoxPro 9.0，并开始零售并在 MSDN 提供预订！自然地，随着 VFP9 的发布，你可以听到来自全球各个 FOX 社区的同一个声音“下一个版本会怎样？”。

听好了宝贝，VFP9 来了，下一步会是什么？哦当然是把它买回来开发了呵呵……

好吧好吧，让我们看得更远一些。微软就“VFP9 以后会发生什么？”这一问题发表的声明到目前为止都还是含糊的而且是有意地含糊。从他们的声明中，我们能看到 VFP 的未来是什么但却不知道具体会怎么样。什么意思？就是他们最终下决心告诉我们，VFP9 将正式被支持到 2014 年，仅此而已。

依我看来，会有很多人心烦意乱仿佛受了谁的欺骗担心天塌下来。我不是说 VFP 程序员会盲目忠于他们钟爱的开发工具而排斥其他任何东西，我们大多数人不会这样。我的意思是我们应该花更多的精力去挖掘这个产品目前已经提供的东西并将其运用到日常工作中去，然后让把你所想的告诉微软让他们知道，告诉他们你还需要哪些目前没有的功能，为什么需要这些功能。

你可以在 Fox wiki 找到 VFP10 的 WISH LIST（功能需求清单），然后你可以把你的建议和说明加到这个清单里并且参与讨论。请点击 http://fox.wikis.com 并搜索“VFP Version 10 Wish List.”。

VFP9 是进化性的还是革命性的？

近来关于 VFP9 的在线讨论大部分都是围绕这个版本提供的新功能展开的。有些人说这个版本只是简单的升级，是进化性的，而另一些人则把她看成是一个革命性的版本。那么，到底是怎么样呢？

我觉得这个问题实际上依赖你的个体情况，这和你现在正在进行的开发工作、你的项目要求以及你原先对新版的构想有相当大的关系。

举个例子，一个以 WEB 开发为主的程序员对新增的 BINDEVENT 允许你绑定 WINDOWS 事件这个功能就没什么感觉；一个用惯了水晶报表的程序员对扩展重写的报表设计器就没什么感觉。那么这些人可能就看不到 VFP9 有什么改变。

进化，做得很好

向下兼容是件好事，VFP9 仍然允许你运行老的代码。VFP 开发团队给了我们一系列的 SET 命令来让我们决定是否启用新功能，比如 SET REPORTBEHAVIOR，SET VARCHARMAPPING，SET ENGINEBEHAVIOR 等。

你可以移植你以前的项目到新版来而不用冒太大的风险并且你可以根据你的需要和喜好启用新功能。

许多新功能是进化性的，他们扩展了现有的类和函数。XMLAdapter 和 CursorAdapter 新增的能力能更好地支持特定的运行环境，比如支持分级的 XML 并且进行控制，比如控制 CursorAdapter CURSOR 如何被刷新等等……

改革部分，也相当棒

有些人说新的“超级 SQL 语法”属于进化部分，但实际上却是一个彻底的“革命部分”，在你近距离领教了她的性能增强和处理复杂子查询的能力之后你就明白了。

依我看来，新的报表系统也是一个“革命”，因为我们现在可以在 ReportListener 的子类中写自己的代码来控制输出。举个例子，你知道么现在你可以用你自己的 HTMLListener 的 XSL 风格页来代替默认的 HTML 输出了？实现这个只需要很少的代码就能获取完全的输出控制(请参考本期 Doug Hennig 的文章)

实际上，你可以派生 XMLDisplayListener（象使用 HTMLListener 那样）并用 XSL 来创建几乎所有你想要的输出格式。这是一项“革命”!

使用她并谈论她

因此，我们为什么不集中精力去试试 VFP9 并参与相关的在线讨论呢？而我们在 FOXTALK2.0 中也将继续提供包括以前版本在内的技术文章。让我们博采众长并激情地生活吧。

案例学习

你的应用程序理解你吗?

原著: Dave Bernard

翻译: Fbilo

如果只要告诉你的应用程序你需要什么数据，应用程序就真的会自己去把数据找来给你的话，是不是很酷？在这个案例学习中，**Dave Bernard** 讲解了一个最近的项目，这个项目整合了语音识别技术、**Microsoft English Query(微软英语查询)**、**SQL Server**、**ASP**、以及 **Visual FoxPro**。

在我按惯例经营我的客户网络的时候，碰到了一个有了大麻烦的家伙。他跟一个信誉很好的高等教育学校签了个合同，内容是去挽救一个已经失败多月了的研究项目。在了解到该项目是关于一个独特的艺术可访问性应用程序的概念论证后，考虑到自己在某些重要的技术领域有一些实际经验，我决定接受这次挑战，我感觉它是一些可以解决的问题。我还注意到其中的部分技术最近已经成熟了。

而且这个项目中看起来有着大量的乐趣。

背景：一个本地博物馆或者画廊

想象一下这个背景：一个美术赞助商走入了一个画廊或者博物馆，她被告知，这里有一个能帮助她参观的无线网络，所以她拿出了她的 **PDA**，并去指定的网站浏览。当她按照自己的方式一个画廊一个画廊的参观的时候，她的 **PDA** 立即感应到最近浏览的内容，并返回任何相关的信息，并通过耳机把这些信息告诉她。**PDA** 会提示她可以提出特定的问题，当她用流利的英语向 **PDA** 提问以后，后者立即告诉她答案。系统持续的跟踪记录她与系统交互的方式、以及在她的访问过程中在哪里停留的最久，并会相应的调整她的经历。

这个项目的目标，是要做一个概念论证来证明做这样一个支持这种类型应用程序的稳定的、花费不多的系统是可行的。

关于语音识别、从语言到文字的转换、以及 **RFID** 部件的内容超出了本文的范围。如果你对项目中

的这些方面感兴趣的话，可以去钻研微软语音识别服务（Microsoft Speech Services，缩写为 MSS）、语音识别应用程序语言标记协议（Speech Application Language Tags protocol，缩写为 SALT）、以及可用于 Microsoft Pocket Internet Explorer 的语音识别插件。

在这篇文章中，我主要谈核心结构的部分，涉及 Visual FoxPro、SQL Server、Microsoft English Query、以及 Web 部署部件。这是一个用 VFP 来将现有的各种不同的技术“粘合”起来以极大的扩展一个应用程序的能力的好例子。

一些背景

有人也许会指出，某些时候我们已经有了“自然语言查询”，毕竟，每天都有数百万人通过 Google 来搜索各种类型的信息和帮助。当我们在搜索的时候，为了满足我们的好奇心，Google 通常会向我们提供成百上千个可能会被我们剔除掉的“匹配项”。虽然这种功能相当的有用，可它毕竟不能跟人与人之间的交互相媲美。

例如，我也许会在 Google 上搜索有关滑雪技术的信息，可如果我想买一张去滑雪胜地的机票，最好还是去找个旅行的网站（或者找个人问问）。从搜索引擎上查找一个特定的主题跟直接问一个预定机票的服务员是相当不同的，例如，“下个星期三晚上有哪一个航班是从亚特兰大到 Aspen 的？”，这样的问题应该得到的是一个独特的、范围有限的答案。

从自然的语言来理解，计算机必须能够理解给出的问题，把问题跟一个已有的知识库联系起来，然后给出有用的答案。自然语言处理（Natural Language Processing，缩写为 NLP）必须理解语法（词语是怎样连接起来的、以及词语的定义之间是怎样的关系）以及所有这些内容与知识库中存储着的信息怎样来关联。

事实上，你在做的，是在定义一个解释你的数据的语义模型。由此，你不需要事先知道将会被问到的问题，在该语义模型内的任何问题都可以被提问。提出的问题会被转换成一个 SQL 查询，在数据存储上执行了该查询后最终返回查询的结果。

当你完成了这样的一个东西以后，情况就变得清楚了。这个结构可以被放在任何定义完好的数据之上，让一个并不熟练的用户（比如一个 CEO 或者 CFO）来提出简单、直接的问题（“过去八个季度我们的利润率怎么样？”）而不需要向工作繁重的 IT 部门提出一大堆的要求。这种功能还可以被用来代替做一大堆的预定义报表并指望这些报表能够满足用户的需求。

思考这个问题一会儿...

计划要做的步骤

目前，我已经做了一个基于 Web 的商业系统，它使用了下列通用的部件：

- 在客户端上的浏览器作为表现层来服务（DHTML、JavaScript、CSS）；
- 位于 Internet 上某个数据服务器上的一个 SQL Server 2000 数据库作为数据存储；
- 位于 Internet 上某个 Web 服务器上的基于 Windows 2000 的 IIS 5.0（或者 Windows 2003 上的 IIS 6.0）上有一个小 ASP 页，它通过访问一个 Visual FoxPro 的 COM+ 部件来提供服务。

如你所见，这是一种相当面向部件的途径，它使得将来缩放该应用程序变得相当的容易。每个部分都存在于一台独立的机器上、或者每个部分都可以被部署到一个不同的硬件上。

需要给这个结构增加的，就是能够生成一个 SQL 查询以返回所需结果的自然语言处理引擎。

Microsoft English Query 能救你的命！

有一种我早就知道、却没有试过的技术，它就是 Microsoft English Query(MSEQ)，SQL Server 从 6.5 版本开始就自带了这种技术。MSEQ 是一个让用户可以用浅白的英语来查询数据库的部件。当我深入到该项目中的时候，逐渐发现 SQL Server 的另一个工具全文本索引将会非常有用，除非结合它和 MSEQ，否则只用 MSEQ 的话，我是无法处理 blob 字段（没有结构的文本，例如一个艺术家的传记）的。

MSEQ 所做的工作是复杂而令人迷惑的，但可以被解释得非常简单。当你在 SQL Server 中定义并设置好了你的数据表以后，还需要设置 MSEQ 以使它理解在你的表和字段们之间的关系，以及不同的怎样按语义来引用它们的途径。

在我们的美术馆示例中，“艺术家制造艺术品”、“艺术家演出”、以及“艺术品被收藏”。当你设置好这些以后，MSDE 就可以接受象这样的一个英语要求：“哪个艺术家在伦敦做过表演？”并为应用程序建立一个适当的 SQL 语句以供执行。（MSEQ 并不真正的执行 SQL 调用，所以你必须用它提供给你的查询单独的执行。）

MSEQ 提供了多个向导来自动化大量的建立英语查询所需的语义模型的过程。我们需要做的，是建立一个定义完善的数据结构，例如我们示例的 SQL Server 数据结构，不过也可以是任何 OLE DB 数据源，包括 OLAP 数据存储。

当你开发完了一个模型以后，**MSEQ** 验证工具（**authoring tool**）允许你用你的用户可能会提出的问题作为英语查询的示例来测试它。如果某个查询测试不能处理，向导会向你提出建议来帮助你手工定义那些关系，这样以后该问题就可能能被正确的处理了。

当你把一切都准备好了以后，你可以生成一个轻便的英语查询域文件，文件中包含有供运行时引擎使用的编译过的 **MSEQ** 模型。你可以轻松的针对不同的数据模式定义出各种大不相同的 **MSEQ** 模式，并用一个程序在运行时切换选择不同的 **MSEQ** 模式。

所以，我们计划中的结构将使用下列工具：

- **运行在 Microsoft Windows 2003 上的 Web/数据服务器**
- **Web 服务器： IIS 6.0**
- **数据库：Microsoft SQL Server 2000 SP4**
- **SQL Server English Query**
- **SQL Server 全文本索引**
- **DHTML、JavaScript、VBScript(ASP)、CSS**
- **Microsoft Visual FoxPro 8.0 SP1**
- **Microsoft 组件服务**
- **Visual Studio**

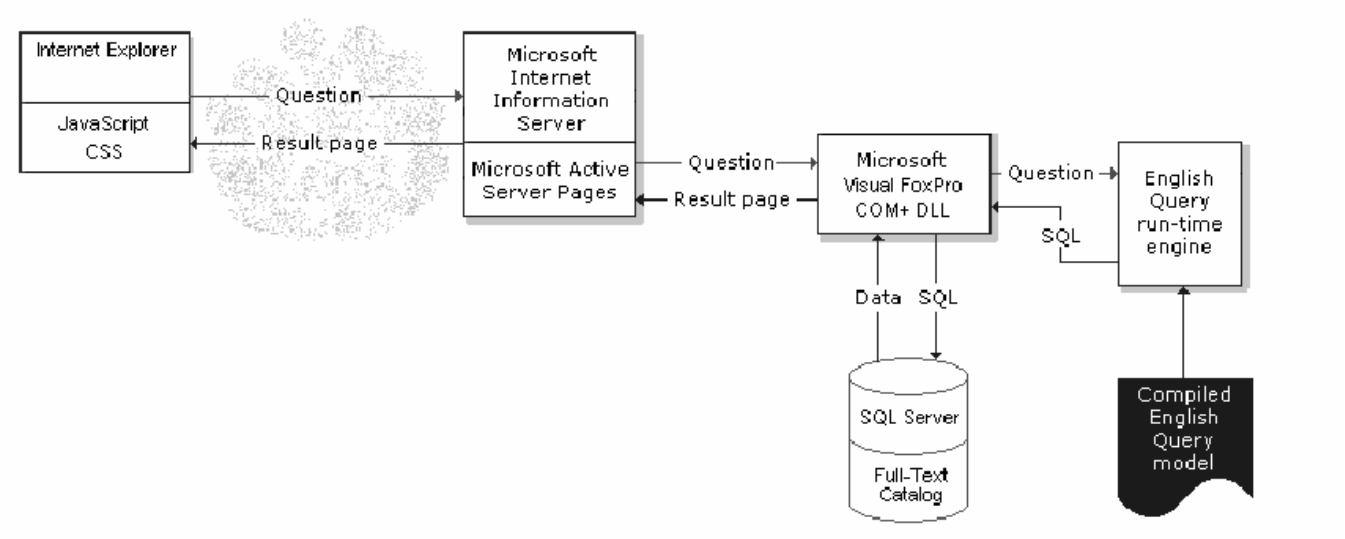

图 **1**、可缩放的、多层软件应用程序结构

在中间层上的 Visual FoxPro

微软已经将 VFP 定位为一个用于建立“集成客户/服务器计算机和 Internet 的可缩放多层应用程序”的重要工具。在过去的几年中，我热切的拥抱了这个魔咒，并用 VFP GUI 真正的做了一点工作。

一个基于 Web 的应用程序要比一个基于桌面的胖客户端容易部署和维护得多。当我部署一个应用程序或对应用程序做了某些改动的时候，一切都发生在一个位置上，而我的客户则立即得到了改动。这就为我减少了大量的工作，并让我可以以更低的代价向客户提供更好的服务。

这个策略的核心是一个 VFP COM+ DLL（COM 对象）。我自己已经有了一个框架类，因此，我在此基础上来建立这个基于 Web 的应用程序。通常，我很少使用 VFP 表，而更愿意信赖 SQL Server 的引擎、工具以及存储策略来处理应用程序所需的数据。在 2GB 的限制内，VFP 表是很好的，可我做过的不少应用程序都很容易会超过这个限制。此外，SQL Server 支持的 SQL 语法要比 VFP 强大不少，尽管 VFP 9.0 已经很大的弥补了差距。

在中间层使用 VFP 最大的好处，是你可以用已知最好的工具来做所有需要做的重活，并且它做得非常快而高效。此外，它是你熟悉的工具，这对我来说也是最好的理由。我的咨询工作中的大部分要求我应尽快的给客户某些结果，所以我通常不会有 6 人/月的时间能花费在赶上 Visual Studio .NET IDE 的脚步上。

做一个 VFP COM DLL 并与它的接口进行交互实在是非常的简单，不过就是从一个 Web 页上取得信息然后交给一个 VFP COM 对象而已。在这个应用程序中，我的 VFP COM 对象的目标是：

1) 为自定义 Web 页提供服务；
2) 与 MSEQ 进行交互；
3) 对 SQL Server 数据库执行 SQL 调用；

在一个 Web 应用程序中使用一个 VFP COM 对象

首先，为什么使用 VFP 来生成 Web 页？对我来说，答案是：这样做很快，并行很灵活。一个 Web 页只是一长串 HTML 标记，而 VFP 在字符串处理方面极其擅长。几年前，我做过一个能进行 XSLT 转换的 VFP COM 对象，它会合并 XML 数据和 XSL 标记来生成一个 Web 页。它运行的很慢，即使在调试和优化以后还是运行的很慢。我转而采用一个老的备用方案，使用 TEXTMERGE，结果，与最好

的 XSLT 方案相比也提高了 6 倍的速度！所以我从来没想过回头。

由于你有一个全面的程序设计语言可用，你就能够整合快速建立一个 Web 页的能力和按你想要的任何方式定制这个 Web 页的能力，因此，你就有了一个用这种方式来建立 Web 页的强力的理由。

为了利用好 IIS（它的目标，是在 Internet 或者 Intranet 上提供 Web 页服务），我使用了一个单薄的“经典”ASP 层来接收来自一个浏览器的请求（例如，用户在一个表单上单击提交），并把请求交给 VFP COM 对象去处理。“经典”ASP 代码的缺点，是它运行起来比较慢，因为它是在运行时被解释执行的，ASP.NET 通过预编译解决了这个问题。由于在这个系统中我只使用了很少一点 ASP 代码，所以影响不大。

这里是一些“经典”的 ASP 代码，它将会与一个名为 MyVFPCOM 的 VFP COM 对象进行交互（为了让你看的清晰些，我去掉了错误和安全性处理代码）：

```
<%
' 声明一些变量
Dim loApp, lcURLAction, lcHTMLStr

' 获得一个对 VFP COM 对象的引用
Set loApp = Server.CreateObject("MyVFPCOM.MyVFPCOM")

' 从一个被传递的查询字符串变量来取得被调用方法的名称 lcURLAction =
Request.QueryString("PRGMethod")

' 生成对 VFP COM 对象引用的调用并运行它，
' 把整个 Request 对象传递给 VFP;
' 将返回的值(VFP 方法建立的一个 HTML 页/字符串）
' 保存到 lcHTMLStr 变量中
lcHTMLStr = Eval("loApp." & lcURLAction _
& "(Request)")

' 把生成的 Web 页回写到浏览器中
Response.Write lcHTMLStr

' 清理对象引用
Set loApp = Nothing
```

```
' 结束在浏览器和 Web 服务器之间的连接
Response.End
%>
```

ASP 代码放在一个简单的扩展名为 ASP 的文本文件中。注意，这里的 Request 对象被传递给了 VFP 方法。Request 对象（它本身其实就是一个 COM 对象）包含着各种对象的集合，这些对象代表着浏览器可以使用的所有 Web 页信息——包括任何表单信息、URL 查询字符串变量、cookies、以及一个关于象浏览器类型、屏幕解析率等等“内部”信息的宿主。

如果你用一个象 MyVFPCOMasp?PRGMethod=GetAPage 这样的 URL 来访问网站，那么，ASP 代码将执行在 VFP COM 对象上的 GetAPage 方法，而后者接着将建立一个应答网页的字符串并将之返回给 ASP 代码，然后 ASP 代码再将之送回给浏览器。这里，ASP 代码在客户端浏览器和 VFP COM 对象之间起着传递数据的中间人的作用。

说清楚点，这个结构本质上是在把 VFP 所有的非界面的能力提供给一个 Web 浏览器使用！如果你是一个想要开发 Web 应用程序的 VFP 程序员的话，你会爱死它的。并且记住：它是 FoxPro，所以它还很快！

建立并使用一个 VFP COM 对象

如果你以前还没做过它，你将会很高兴的知道，建立一个 VFP COM 对象要比你想象的简单得多。要记住的最重要的方面是：1）选择哪个类作为你工作的基础，以及 2）OLEPUBLIC 关键字。从 VFP 6 SP3 开始引入的轻量级、非可视类 Session 是特地为多线程 COM 对象而设计的。它提供了一个私有数据工作期而又不需要一个表单的开销。

Session 类默认还隐藏了所有自带的属性，所以类型库里不会有这些属性存在。EXCLUSIVE、TALK 和 SAFETY 的默认值是 OFF（从 VFP 7 的 Session 类开始），不过你需要去合理的设置其他“SET”命令，因为现在它们的作用范围都在私有数据工作期中了。

```
*=============================================
DEFINE CLASS DemoCOM AS Session OLEPUBLIC
*=============================================
*=============================================
* 初始化 COM 环境
```

```
*================================================
PROCEDURE INIT

SET RESOURCE OFF
SET DELETED ON

* 阻止错误消息和对话框打断程序的运行
SYS(2335, 0)

* 阻止 SQL Server 的登录对话框
SQLSetProp(0, "DispLogin", 3)
SQLSetProp(0, "DispWarnings", .F.)

ENDPROC
*================================================
* 返回一些简单的 HTML
*================================================
PROCEDURE MakeSimpleWebPage
RETURN "<B>Here's a simple web page</B>"
ENDPROC
ENDDEFINE
```

上面这个类的 **INIT** 方法非常重要，因为它被用以设置应用程序的初始化运行环境。记住，一个 **COM** 对象不能有任何用户界面，所以我增加了对 **SYS(2335)**和 **SQLSetProp** 的调用。每当 **CreateObject** 被调用的时候都会触发 **Init** ，因此尽量让其中的代码保持简洁高效！

当在 **IIS** 下运行一个 **COM** 对象的时候，出于缓存的目的，**DLL** 被持久的锁定在内存中，由于 **VFP** 的运行库只会被加载一次，此后会一直留在 **IIS** 的进程中，因此这样一来就会极大的提高性能。不幸的是，在更新一个正在运行中的 **COM** 对象的时候，需要采取几个手动的步骤来释放已有的实例并将之换成一个新的。我将会在后面我们探索组件服务的时候谈到更多关于这个问题的内容。

建立一个名为 **SimpleCOM** 的项目，把上面的代码保存成 **DemoCOM.PRG** 文件，并把它添加给该项目。然后编译一个多线程 **DLL**。编译完以后，你可以在 **VFP** 里这样测试它：

```
oCOM = CREATEOBJECT("SimpleCOM.DemoCOM")
? oCOM.MakeSimpleWebPage
```

为了从 ASP 中使用它，建立一个名为 COMDemo.asp 的文本文件，并把下面内容拷贝过去：

```
<%
Dim loApp, lcHTMLStr
Set loApp = Server.CreateObject("SimpleCOM.DemoCOM")
lcHTMLStr = loApp.MakeSimpleWebPage
Response.Write lcHTMLStr
Set loApp = Nothing
Response.End
%>
```

把这个 ASP 文件保存到你的 IIS 的 wwwroot 文件夹中，并确保 IUSR_MachineName 帐号对该文件夹有执行的权限。

IIS 的安全性会显著的影响 COM 对象的部署。当一个 ASP 页建立对一个 COM 对象的引用的时候，该 COM 对象会继承 IIS 安全性状态，这个状态通常就是 IUSR_MachineName 帐号的权限设置（在某些情况下可能是 IWAM_MachineName 帐号）。需要保证 IUSR_MachineName 或者 Everyone 帐号有对 COM 对象的读和执行权限、以及 COM 对象自己有着对任何 VFP 数据和它要用到的运行库的全部权限。

主要 COM 对象需要有 VFP 的运行时 DLL。在你用来开发的机器上这些运行库已经存在了，所以不需要做什么事情。如果你的产品所运行的服务器上没有安装 VFP，请将 VFP 运行时 DLL 拷贝到 COM 对象所在的目录上，以使组件服务能够在需要的时候找到它们。

在你可以测试以前，COM 对象必须先被安装在组件服务下（根据你使用的是哪个版本的 Windows，准确的步骤可能会有所不同）：

1. 从管理工具中打开组件服务。
2. 展开组件服务节点，找到 COM+ 应用程序。
3. 在 COM+ 应用程序上单击使它高亮显示。
4. 在 COM+ 应用程序上单击鼠标右键，并选择“新建|应用程序”。
5. 单击“下一步”，然后选择“创建空应用程序”。
6. 命名为 SimpleCOM 并单击“下一步”，再“下一步”，最后“完成”。
7. 在左边的面板中，在 SimpleCOM 上单击右键，然后选择“属性”。

8. 在“安全性”页上单击，然后选中“对此应用程序强制进行访问权限检查”。
9. 单击 OK。
10. 在左边的面板中，展开 SimpleCOM 节点。
11. 在“组件”上单击来高亮它。
12. 在“组件”上单击右键，并选择“新建|组件”。
13. 单击“下一步”，然后再“安装新组件”。
14. 找到你放 SimpleCOM.DLL 的文件夹然后单击 SimpleCOM.DLL，再单击打开。
15. 单击下一步，然后完成。

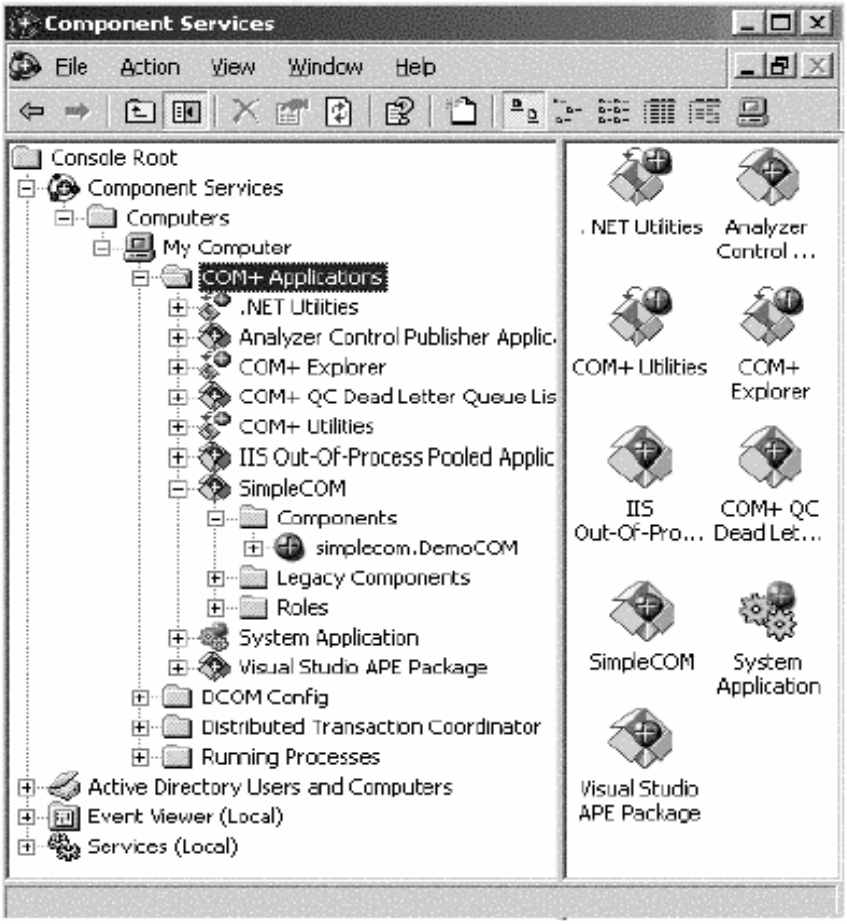

图 2、组件服务管理器

为了测试这个应用程序，打开你的浏览器，并在地址栏中输入 http://localhost/COMDemo.asp 后回车，你将会看到由 VFP COM 对象的 MakeSimpleWebPage 方法生成的简单的 Web 页。

关于组件服务的更多内容

你可以把 COM 看作是让应用程序可以象对象一样被访问的某种东西。而 COM+则以组件服务而闻名，它是从 Windows 2000 开始引入来替代、增强和简化老的 Microsoft Transaction Server(微软事务服务器，简称 MTS)。COM+ 应用程序是通过控制面板中管理工具下的组件服务管理器来管理的。

COM+ 使用基于角色的安全性。基于 Windows 2000 用户和组。角色是用户的类型，特定的 Windows 用户被分配给角色。例如，为了让我们的部件可以被通过 Internet 来访问，我们可以建立一个 Internet 用户角色，并把 IUSR_MachineName 分配给该角色。如果任何不属于该角色的人试图去访问该部件，访问会被拒绝并且一个错误会被返回给客户端。

更新一个 VFP COM 对象

在对一个 COM 对象做了改动以后，如果该 COM 对象正作为一个实例运行在组件服务下的话，你是不能马上重新编译或者覆盖它的。这里是编译和部署你的改动时应采取的步骤：

1. 在组件服务中，在该 COM 应用程序上单击鼠标右键，并选择“关闭”。
2. 重新编译 DLL。如果编译好的 DLL 默认的所在位置不在 Web 服务器上，则将其拷贝过去。下一此当一个应用程序建立对该 COM 对象的一个引用的时候，它会自动被组件服务器所“启动”。

请注意，DLL 在被编译的机器上是自己会注册的。由于注册信息被保存在项目文件中，重新编译 DLL 会受限自动取消对它的注册。总之，如果项目文件被删除然后又重建了，那么直到下一次编译之前，旧的注册表项不会被去掉。

DLL 的编译进程还会建立一个类型库（TLB）文件和一个注册键文件（VBR），这两个文件必须与 DLL 放在一起。类型库包含着说明 DLL 接口的属性和方法信息。简而言之，它列出了 COM 对象的整个对象模型。

测试和调试

建立一个 COM 对象是一个相当简单的任务，但要测试它就完全不同了。测试它最好的办法，是在用 ASP 调用进行系统测试以前，就尽可能多的在 VFP 中进行测试。在我们这个特定的例子中，当在 VFP IDE 中进行测试的时候，很难模拟传递 ASP Request 对象的过程：这只有在真正的 ASP 环境中才能做到，而且你无法在 VFP 的调试器中跟踪一个 VFP COM 对象。你能够跟踪 PRG 类，但无法跟踪编译过的 COM 对象。

使用 VFP 和 MSEQ

现在是时候使用这个简单的结构、并扩展它去处理自然语言查询了。使用我们的 COM 对象示例，还需要增加一个方法以执行下列内容：

1. 从一个 Web 页返回一个用户输入的浅白英语查询。
2. 在将问题发送给 MSEQ 引擎之前，先对问题进行
3. 将从 MSEQ 返回的 SQL 调用语句发送给 SQL Server，并得到一个结果集。
4. 将结果集格式化为一个 Web 页。
5. 将该 Web 页返回给浏览器。

```
*================================================
* 处理英语查询请求
*================================================
PROCEDURE EQResponse(loRequest AS Object) AS Variant

LOCAL lcReturn, lcRest, lcInquiry, lcChar, i,;
lcRespGrid, lnConn, loEQResult, lcSQL, loEQ

WITH THIS
* 从变量获得来自服务器的问题
lcInquiry = loRequest.Form("Question").Item()

* 去掉问题后面的问号，如果有的话
lcInquiry = STRTRAN(lcInquiry, "?", "")

* 生成应答的 Web 页
lcRespGrid = ""

IF NOT EMPTY(lcInquiry)
    loEQ = CREATEOBJECT("MSEQ.Session")

    IF VARTYPE(loEQ) != "O"
        RETURN "错误: MSEQ 不能被初始化"
```

```
ENDIF

loEQ.InitDomain("EQDemo.eqd")
loEQResult = loEQ.ParseRequest(lcInquiry)

loEQ = .NULL.
RELEASE loEQ

IF loEQResult.Type = 0
    * 建立一个对 SQL Server 数据库的连接
    lnConn=SQLStringConnect("DRIVER={SQL Server};"+;
        "SERVER=MyServer;" +;
        "UID=;PWD=;" +;
        "DATABASE=MyDatabase")

    * 获得一个对 MSEQ 结果集的一个对象引用
    loEQCmd = loEQResult.Commands(0)

    * 返回生成的 SQL 调用字符串
    lcSQL = loEQCmd.SQL

    * 在 SQL Server 连接上执行该字符串
    lnTally = SQLEXEC(lnConn, lcSQL)

    IF lnTally <= 0
        RETURN "EQ 应答查询失败！"
    ENDIF

    IF RECCOUNT() > 0
        * 如果 SQL 调用的结果返回了多条记录，
        * 则将它们格式化到单个 Web 页中去
        SELECT DISTINCT Artist, ArtistID;
            FROM SQLResult;
            ORDER BY Artist;
            INTO CURSOR q
```

```
            * 生成一个应答列表
            SCAN
                lcRespGrid = lcRespGrid +;
                IIF(EMPTY(lcRespGrid), [], [<br/>]) +;
                ALLTRIM(Artist)
            ENDSCAN

            * 给应答添加一个前导语句，Add a lead-in statement to the response and
            * 并包含一个 MSEQ 对问题理解后的翻译
            lcRespGrid = ;
                [I understood you to ask: "] +;
                loEQResult.Restatement + ["; ] +;
                [I found ] +;
                ALLTRIM(STR(RECCOUNT())) +;
                [ match] +;
                IIF(RECCOUNT() = 1, [], [es]) +;
                [, which ] +;
                IIF(RECCOUNT() = 1, [is], [are]) +;
                [ listed below.<br/>] +;
                lcRespGrid
            USE IN q
        ELSE
            lcRespGrid = "没有找到匹配的内容。"
        ENDIF

        USE IN SQLResult
    ELSE
        * 有错误发生吗？
        IF loEQResult.Type = 2
            lcRespGrid = loEQResult.Description
            lcRespGrid = STRTRAN(lcRespGrid,;
            [Please capitalize proper names], [])
        ELSE
            * 需要一个澄清
            IF loEQResult.Type = 3
                lcRespGrid = [Spelling clarification ]+;
```

```
                    [needed for ] + lcInquiry
                ENDIF
            ENDIF
        ENDIF
    ENDIF
    ENDWITH
    * 把结果发送回给浏览器
    RETURN lcRespGrid
    ENDPROC
```

代码分解

EQResponse 方法封装了一个 **Web** 页的建立过程，该 **Web** 页中包含着一个 **SQL** 调用的结果集，而这个调用是 **MSEQ** 根据一个浅白英语查询而生成的。如前所述，**Request** 对象从浏览器被传递给 **EQResponse** 方法，这样一来，就可以在这个方法中访问浏览器对象了。在这个案例里，我们只对 **Web** 页上用于让用户输入浅白英语查询的文本框感兴趣，这个文本框将象下面这样在 **HTML** 中指定：

```
<input type="text" name="Question" size="40" value=""/>
```

将输入的查询文本赋值给 **lcInquiry**：

```
lcInquiry = loRequest.Form("Question").Item()
```

下一步，取得对 **MSEQ** 引擎的一个引用：

```
loEQ = CREATEOBJECT("MSEQ.Session")
```

我们必须告诉引擎，该使用哪个编译了的 **MSEQ** 定义：

```
loEQ.InitDomain("C:\MyDemo\EQDemo.eqd")
```

现在，将浅白英语查询传递给 **MSEQ**，并记下返回的对象：

```
loEQResult = loEQ.ParseRequest(lcInquiry)
```

接下来对这个结果对象进行工作，使用它的 **Type** 属性来检查它的状态。一个 **0** 表示 **MSEQ** 返回了一个 **Command** 对象，后者对我们来说就意味着一个 **SQL** 语句的字符串。状态为 **2** 表示发生了一个错误，错误消息被储存在 **Description** 属性中。状态为 **3** 表示 **MSEQ** 用户对查询进行澄清，通常这是因为用户输入的查询中包含着某种拼写错误。

对于得到的应答是 **Command** 的情况，我们可以继续执行下去，建立一个对 **SQL Server** 数据库的连接，并在该连接上执行 **MSEQ** 生成的 **SQL** 语句：

```
lnConn = SQLStringConnect("DRIVER={SQL Server};" +;
    "SERVER=MyServer;" +;
    "UID=;PWD=;" +;
    "DATABASE=MyDatabase")
loEQCmd = loEQResult.Commands(0)
lcSQL = loEQCmd.SQL

lnTally = SQLEXEC(lnConn, lcSQL)
```

最后，我们格式化出一个结果页，并把这个结果页发送回给浏览器。

运行这些代码：

建立一个简单的 **Web** 页来捕捉用户输入的查询：

```
<form action="SubmitQuestion.asp" method="POST">
    Enter a plain-English query:
    <input type="text" name="Question"
        size="40" value=""/>
    <input type="Submit" name="btnSubmit"
        value="Submit"/>
</form>
```

这里是当用户按下 **Submit(提交)**按钮后将被执行的 **ASP** 代码，假定 **EQResponse** 方法已经被加入到 **DemoCOM.PRG** 中了：

```
<%
```

```
        Dim loApp, lcHTMLStr
        Set loApp = Server.CreateObject("SimpleCOM.DemoCOM")
        lcHTMLStr = loApp.EQResponse(Request)
        Response.Write lcHTMLStr
        Set loApp = Nothing
        Response.End
    %>
```

数据库设计指南

对于被尽可能的正规化了、并且最大限度的利用了主键和外键来全面定义内部表间关系键的数据库（见图 3 中用于本示例的数据模型）来说，基于 **MSEQ** 的应用程序工作地最好，并且最容易建立。

计划英语查询模型

在我们可以开始使用英语查询引擎之前，我们需要为数据存储定义它的语义模型。这里是开发一个非常高效的英语查询模型所需的主要步骤提纲：

1、考虑用户们可能会提出的问题。在我的案例中，他们的问题可能是关于艺术家、他们的工作、他们的表演等等。确保你的语义模型能将这些实体联系好。

2、使用英语查询项目向导建立一个模型，该向导会使用数据模型来自动根据这些表、字段、主键、以及连接来建立实体和关系。

3、改进模型来接纳向导不能处理的问题。你可以通过添加向导漏掉的重要实体和关系来完成这个任务。在定义实体之间的关系中有着大量的灵活性，并且它可能会变得非常的复杂。缓慢进行并最大限度的利用建议向导来测试可能的问题，并调整这个模型。

4、建立用于部署的 **EQD** 文件。

建立英语查询模型

1、选择开始|程序|**Microsoft SQL Server**|**English Query**|**Microsoft English Query**。这时会打开 **Visual Studio** 以及一个在英语查询项目下的新项目对话框。确保高亮选中了 **SQL** 项目向导，并设置你所希望的项目名称和路径，然后单击打开。

2、在数据链接属性对话框中，定义一个对你的数据库的连接。

3、当向导显示出一个所有找到的表的列表时，选中所有的东西并单击 **OK**。

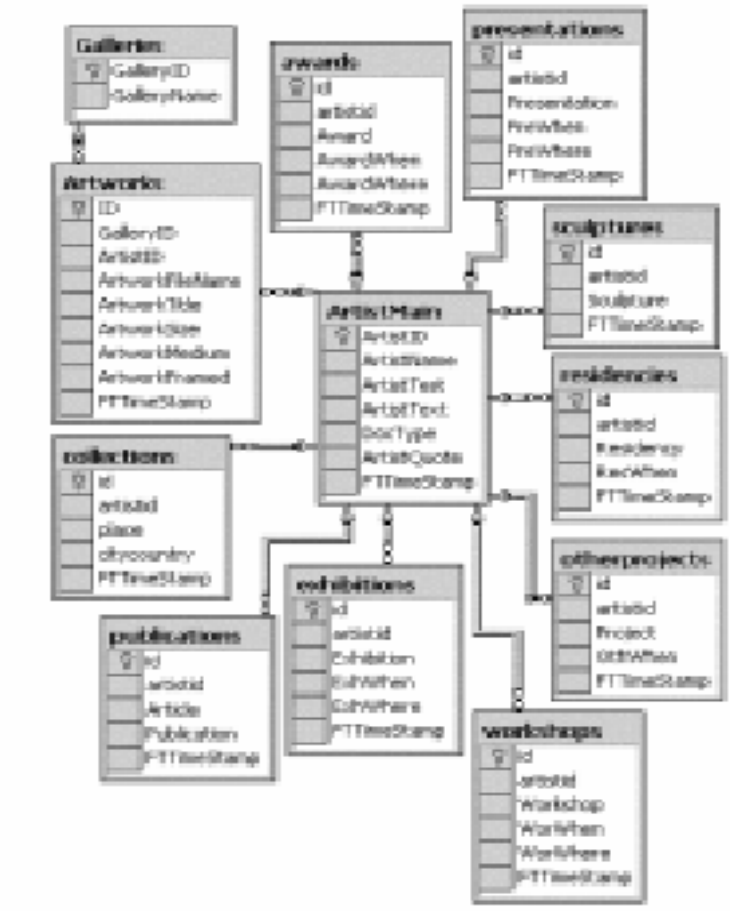

图 **3**、数据库模型

4、下一个对话框显示了向导从模型中找出的一系列标准的实体和关系。你可以接受向导的选择或者取消选中你不想要的项目。单击 **OK**。

5、现在，基本的模型已经被建立起来了，在编译最终的版本之前，你可以改动、测试、调整这个模型。

MSEQ 将模型定义储存在两个文件中：一个英语查询模块（扩展名为 **EQM**）和一个英语查询项目（扩展名为 **EQP**）。这两个其实都是 **XML** 文件，并且可以用任何文本编辑器来编辑它们。**MSEQ** 用这些文件来生成轻便的英语查询域（**EQD**）文件，**COM** 对象将用这个文件来与英语查询引擎进行交互。

测试和调整英语查询模型

英语查询带有一个非常有帮助的测试功能，我已经发现，在确保用户可能提出的问题能够被模型支持方面，这个功能是必不可少的。例如，我可以测试这样一个问题“谁在伦敦做过一个演出？”（见图 **4**），然后会看到来自英语查询引擎的详细应答。这个应答包含着将从数据库返回期望的答案的 **SQL** 语句、以及在数据库上运行该 **SQL** 语句的结果。

如果碰到了一个引擎无法翻译的问题，你可以打开建议向导，这个向导将帮助你调整这个模型。记住，**MSEQ** 能够理解的东西严格依赖于已经存在于模型中的那些关系。

在英语查询中，一个关系将一个或多个实体关联起来，并定义它们之间是怎样的关系。这可以用简单的与实体相关的语句来实现（例如，“画家创作艺术作品”），实体有一些涉及的短句，短句的类型包括名词、形容词、子集、前置词、动词、以及特征短句。

例如，“画家创作艺术作品”是一个动词短句的例子（见图 **5**）。使用不同的短句是扩展一个模型的好办法。完整的解释短句功能超出了本文的范围，不过可以被建立在一个模型中的复杂关系的数量是令人难以想象的。

在你对模型的性能满意了之后，把它编译为一个编译过的英语查询应用程序（**EQD** 文件）。

此外...

我已经讲了一些与英语查询所能实现的功能相关的非常表面的内容。希望我为你激起的兴趣能让你自己走得更远。

下载：**503BERNARD.ZIP**

超链接你的报表

原著：Doug Hennig

翻译：CY

上月，Doug Hennig已经讨论过VFP9里新的ReportListener类，和如何以以前不可能的方法来控制报表输出。本月，他将关注如何从报表生成活动超链接，允许在被点击时以实现某些动作。

如果让VFP在报表里加入超链接是不是很酷？然后用户可以点击超链接以导航到某些相关的信息。比如，一个报表显示客户和他们的网址或邮件地址，并有其活动链接；在网址上点击时可以导航浏览器到URL。更有趣的事是可以在你的应用程序里导航到其他地方。比如，点击报表里的公司名可以从所选定的公司记录带出客户数据。因为VFP里的报表预览窗口并不支持活动，报表里的可点击对象，最容易的实现办法是利用HTML，它可以支持超链接。

超链接报表

VFP所带来的报表监听器可以输出HTML（HTMLListener 类内置于 ReportOutput.APP 里，也包含于 FFC 文件夹下的 _ReportListener.VCX 里），但是我确信，它还需要许多工作才可以支持超链接。可是，我很乐意去发现所需要的努力。

首先，后台的 HTMLListener 是 XMLDisplayListener 的子类，也是 XMLListener 的子类，也是 _ReportListener 的子类，也是上月我所讨论并推荐作为你自己平常使用的类的父类。当你使用 HTMLListener 时，可以直接例化它并把它作为报表监听器，或者通过在 REPORT 命令里指定 OBJECT TYPE 为5，它实际上为报表生成 XML（这是由它的父类实现），然后对 XML 应用XSL 转换以生成 HTML。所使用的 XSLT 是定义在 GetDefaultUserXSLTAsString 方法里的。

HTMLListener 默认使用的XSLT 是非常复杂的，并且还不是有很多的 XSL 专家，我认为找出更改什么可以支持超链接是件不可抗拒的任务。可是，在我开始深入 GetDefaultUserXSLTAsString 后，我发现如下 ：

```
<xsl:when test="string-length(@href) &gt; 0">
<A href="{@href}">
<xsl:call-template name="replaceText"/>
</A>
</xsl:when>
```

如果在XML里的当前元素里有 HREF属性，XSL 加入一个锚标记到 HTML。这是很酷的，这意味着 HTMLListener 已经支持超链接！可是，在 XMLDisplayListener 和 XMLListener 里找不到任何与“HREF”有关的，可是你该如何加入属性到元素里，特别是动态的？

在深入更多后，我发现特定元素的属性是在 XMLListener 的 GetRawFormattingInfo 方法里设置的。于是，我把 HTMLListener 子类化，并且加入了所期望的行为到方法里。

下面的代码，摘取自 HyperlinkListener .PRG，它提供了一个监听器，可以在报表的对象上生成超链接，如果对象的 User 备注里包含有跟随有可作为URL的表达式的“*:URL =”指令。

```
define class HyperlinkListener as HTMLListener ;
of home() + 'ffc\_ReportListener.vcx'
QuietMode = .T.
&& default QuietMode to suppress feedback
dimension aRecords[1]
&& an array of information for each record in FRX
```

* 在我们运行报表前，遍历整个FRX，并把每个在 User 备注有我们所期望的指令的字段信息存储到 aRecords 数给里。

```
function BeforeReport
dodefault()
with This
.SetFRXDataSession()
dimension .aRecords[reccount()]
scan for atc('*:URL', USER) > 0
.aRecords[recno()] = ;
alltrim(strextract(USER, '*:URL =', ;
chr(13), 1, 3))
endscan for atc('*:URL', USER) > 0
.ResetDataSession()
endwith
endfunc
```

* 如果当前字段有指令，加入URL到节点属性。

```
function GetRawFormattingInfo(tnLeft, tnTop, ;
```

```
tnWidth, tnHeight, tnObjectContinuationType)
local lcInfo, ;
lnURL
with This
lcInfo = dodefault(tnLeft, tnTop, tnWidth, ;
tnHeight, tnObjectContinuationType)
lcURL = .aRecords[recno('FRX')]
if not empty(lcURL)
.SetCurrentDataSession()
lcInfo = lcInfo + ' href="' + ;
textmerge(lcURL) + '"'
.ResetDataSession()
endif not empty(lcURL)
endwith
return lcInfo
endfunc
enddefine
```

BeforeReport 事件仅在报表运行前发生。它使用 SetFRXDataSession 方法来选择 FRX 游标所在的数据工作期，然后再扫描 FRX 并把任何有指令的对象的URL表达式存入到数组。在结束时它调用 ResetDataSession 来恢复监听器所在的数据工作期。

GetRawFormattingInfo 方法利用 DODEFAULT()来实现通常的行为，它为 XML 生成属性作为字符串。然后它检查相应的数组元素（FRX 游标的数据工作期是由 XMLListener 里的代码在这个代码执行前来选定），以查看报表里的当前对象是否有指令，如果有，就加入 HREF 属性到 XML 元素。它调用 SetCurrentDataSession 来选择报表数据所使用的数据工作期，并对 URL 表达式使用 TEXTMERGE()，因为对于每个记录，表达式很可能包含有某些特殊字符，比如<<CustomerID>>。最后，它通过调用 ResetDataSession() 来实现某些必要的内部处理以离开数据工作期，正如我们所见到的。

就是如此！现在让我们来看看一些如何使用监听器的示例。

例1：活动链接到URL

Links.FRX 是一个简单的示例，它显示了监听器是如何工作的。它是在 Links 表上的报表，它有一给的公司名和相应的网址。报表里的网址栏在它的 User 备注里有“*:URL = http://<<trim(website)>>”。 Links.PRG 运行这个报表，它使用 HyperlinkListener 来作为报表的监听器，并使用 FFC 文件夹下的 _ShellExecute 类，以在你的默认浏览器里显示 HTML 文件。图1显示的是结果。

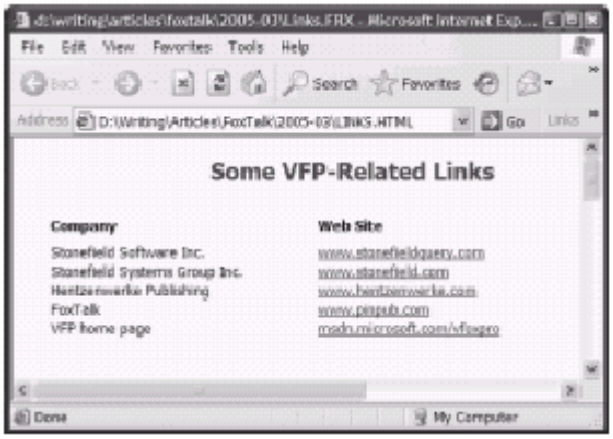

图1：从 Links 报表里生成的 HTML 有活动超链接。

```
loListener = newobject('HyperlinkListener', ;
'HyperlinkListener.prg')
loListener.TargetFileName = fullpath('Links.html')
report form Links object loListener
loShell = newobject('_ShellExecute', ;
home() + 'ffc\_Environ.vcx')
loShell.ShellExecute(loListener.TargetFileName)
```

例2：挖掘式报表

HyperlinkReports.SCX 是一个更复杂的示例。正如你在图2里所见，它呈现出一组的客户信息。可是，HTML 是显示在 嵌入到VFP表单的Web Browser ActiveX 控件里，而不是在浏览器里。当你点击公司名时，VFP将会运行这个客户的订单报表并显示在表单里，如图3所示。订单报表里也有超链接可以返回到显示客户列表。于是，这个表单提供了挖掘式报表。

图2：HyperlinkReports.SCX 显示了客户报表，每个客户超链接到匹配的订单。

图3：点击客户的链接将显示客户订单报表。

表单的 Init 方法使用了 HyperlinkListener 类来从 HyperlinkCustomers 里生成一个超链接的 HTML 文件，然后再调用 ShowReport 方法以在 Web Browser 控件里来显示出来。它也维护一个由表单生成的 HTML 文件集合，以使得表单在关闭时它们可以被删除。

```
with This

* 创建一个我们所创建的 HTML 文件集合，以使得我们可以在关闭时来清除它。

.oFiles = createobject('Collection')

* 创建客户报表

.oListener = newobject('HyperlinkListener', ;
'HyperlinkListener.prg')
.oListener.TargetFileName = ;
fullpath('HyperlinkCustomers.html')
report form HyperlinkCustomers object .oListener
.oFiles.Add(.oListener.TargetFileName)

* 显示报表

.ShowReport()
endwith
```

ShowReport 方法只需要让 Web Browser 控件来装载当前 HTML 文件：

```
local lcFile
```

```
lcFile = This.oListener.TargetFileName
This.oBrowser.Navigate2(lcFile)
```

为更胜于为每个客户创建订单报表并作超链接，我决定在客户名被点击时根据需要来生成报表。为此，我需要截获超链接点击。幸好，这很容易做到：只需要把代码放入到 Web Browser 控件的 BeforeNavigate2 事件里。

为让 BeforeNavigate2 知道这不是普通的超链接，我使用了一个约定“vfps://,”，它表示着“VFP script,”而不是“http://.”。 BeforeNavigate2 里的代码在URL里查找这个字符串，如果有，就执行其余的URL，而不是导航到它。比如，在 HyperlinkCustomers.FRX 的 CompanyName 栏的 User 备注里有如下内容：

```
*:URL = vfps://Thisform.ShowOrdersForCustomer('
<<CustomerID>>')
```

HyperlinkListener 报表监听器对于 CustomerID 为 ALFKI 的客户，将会转换它为一个锚标记如 <a href=“vfps://Thisform .ShowOrdersforCustomer(‘ALFKI’)”> 。当你点击 Web Browser 控件里的超链接时，BeforeNavigate2 将会触发，并且在事件里的代码将剔除“vfps://”部分并执行其余部分。它也通过传递参数来设置 Cancel 参数，.T.表示普通的导航不会发生（类似于在VFP里使用 NODEFAULT）。这里在是 BeforeNavigate2 的代码：

```
LPARAMETERS pdisp, url, flags, targetframename, ;
postdata, headers, cancel
local lcMethod
if url = 'vfps://'
lcMethod = substr(url, 8)
lcMethod = left(lcMethod, len(lcMethod) - 1)
&& strip trailing /
&lcMethod
cancel = .T.
endif url = 'vfps://'
```

当你点击公司名时，ShowOrdersForCustomer 方法被执行，它为特定的客户运行 HyperlinkOrders 报表，并在 Web Browser 控件里显示，然后加入文件名到文件文件集里，以便在表单关闭时被删除。

```
lparameters tcCustomerID
```

```
with This
.oListener.TargetFileName = ;
fullpath(tcCustomerID + '.html')
report form HyperlinkOrders object .oListener ;
for Orders.CustomerID = tcCustomerID
.ShowReport()
.oFiles.Add(.oListener.TargetFileName)
endwith
```

在 HyperlinkOrders 报表里的 CustomerName 栏里有“*:URL = vfps://Thisform.ShowCustomers()”在它的 User 备注里，于是，点击这个报表里的超链接将显示客户列表。

例3：运行VFP表单

CustomerReport.SCX 与 HyperlinkReports.SCX 类似，但它更简单。它也有一个 Web Browser 控件来从报表里显示 HTML，EditCustomers .FRX，它看起来与前面的例子一样。可是，在表单上点击客户名时将显示一个所选定客户的维护表单。

EditCustomers.FRX 是在前面示例里 HyperlinkCustomers 报表的克隆，但是在的 CompanyName的 User 备注里代替的是“*:URL = vfps://Thisform .EditCustomer(‘<<CustomerID>>’)”。当点击客户名时，BeforeNavigate2 调用了表单的 EditCustomer 方法，以运行 Customers 表单，并传递给它选定客户的 CustomerID。Customers 表单是一个简单的对 Customer 表的维护表单，它的控件绑定了每个字段，并有 Save 和 Cancel 按钮。

NavPaneListener

MVP Fabio Vazquez 创建有另一种带超链接的监听器，虽然是出于完全不同的目的。他的 NavPaneListener 可以从 http://ReportListener.com 下载，提供了对报表的带目录表的 HTML 报表预览。正如你在图4里所见到的，每页的略图是显示左边，当前页是显示在右边。点击略图可以导航到对应的页。

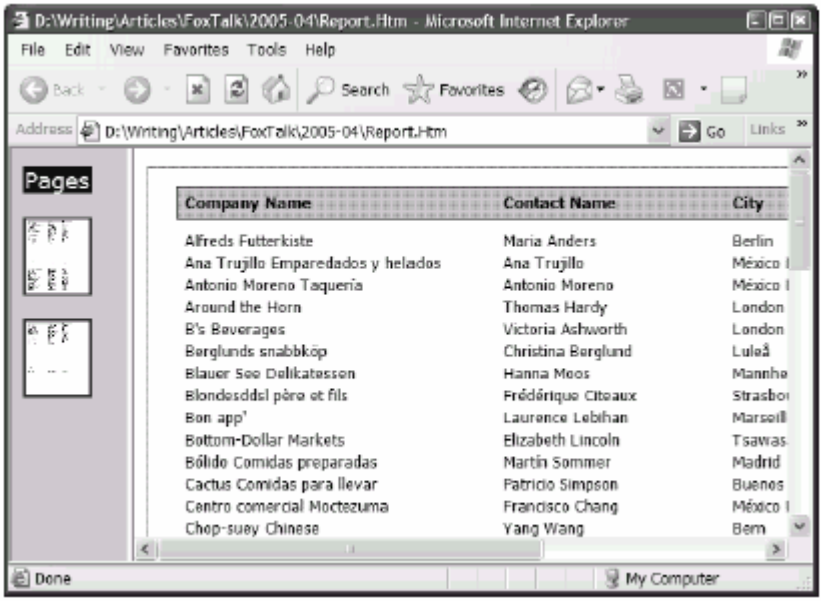

图4：Fabio Vazquez 的 NavPaneListener 创建了一个带目录表的 HTML 报表预览。

就象 HyperlinkListener，NavPaneListener 也是很简单的。它的 OutputPage 事件，对每一页进行调用输出，只需要为每页生成一个GIF文件，通过以适当的参数来调用它自身。tnDeviceType 被初始为 -1，意味着没有输出，因为监听器的 ListenerType 属性是设置为 2。在这种情况下，OutputPage 调用它自身，并传递要生成的GIF文件名和路径（cPath 属性默认为当前目录），和代表着GIF文件的设备类型值。在第二次调用时，除了普通行为外（生成指定的GIF文件），OutputPage 传递文件名和路径给存储在 oNavigator 属性的集合对象的 AddPage 方法。（注：我翻译了 Fabio 的部分代码为英语以便更容易阅读）。

```
procedure OutputPage(tnPageNo, teDevice, ;
tnDeviceType)
local lnDeviceType
with This
do case
case tnDeviceType = -1 && None
lnDeviceType = 103 && GIF
.OutputPage(tnPageNo, .cPath + 'Page' + ;
transform(tnPageNo) + '.gif', lnDeviceType)
case tnDeviceType = 103
.oNavigator.AddPage(teDevice)
endcase
endwith
```

endproc

当报表运行后，导航对象创建了一对的 HTML 文档---一个定义了帧集，在左边帧里带有目录表，而内容显示在右边的帧里，另一个包含有目录表，以GIF文件略图超链接到显示全尺寸GIF文件在内容帧里。导航对象然后自动让IE浏览器来显示帧集文档。

总结

注意在这些示例里的代码没有是很复杂的，也不是很长的。作为结果，它给出一些新的方法来实现带活动超链接的报表，挖掘式报表，运行VFP表单的报表，或是其它动作，或是带导航面板的报表。这真实的显示了报表监听器的强大！

下月，我们将关注一个类似的主题----可以在点击时实现某些动作的报表预览---但是使用了完全不同技术，可以让我们实现诸如文本搜索和文本标签。

来自 VFP 开发团队的 TIPS

原著: 微软 VFP 开发团队

翻译: LQL.NET

这个月的“来自 VFP 开发团队的 TIPS” 专题继续聚焦在 VFP9 SQL 引擎提供的新功能和速度的增强上。VFP9 于 2004 年 12 月向业界发布，开始零售并提供 MSDN 预订。那么你读到这篇文章的时候里面所展示的内容在零售版里应该可以兑现了。

用 VFP9 的 Update 命令快速更新记录

下面的代码展示了 VFP9 Update 命令在速度上的巨大的增强，在看的时候你有必要对比一下以前的程序代码风格。注：所有的样例代码都包含在附带的下载文件里。

```
* Demonstrate correlated UPDATE (UPDATE ... FROM ..)
* UPDATE unitprice of products from mfg_msrp with 10%
* off MSRP
CLOSE DATABASES ALL
SET MULTILOCKS ON
OPEN DATABASE Northwind
CLEAR
? "Procedural REPLACE FOR"
lnStart = SECONDS()
USE Products IN 0
CURSORSETPROP("Buffering",5,"Products")
USE Mfg_msrp IN 0
SELECT Mfg_msrp
SCAN
   SELECT Products
   REPLACE ALL unitprice WITH mfg_msrp.msrp*.90 ;
     FOR Products.ProductID = mfg_msrp.ProductID
```

```
   SELECT mfg_msrp
ENDSCAN
? SECONDS() - m.lnStart
SELECT Products
=TABLEREVERT(.T.)
CLOSE TABLES ALL
? "Correlated UPDATE"
lnStart = SECONDS()
USE Products IN 0
CURSORSETPROP("Buffering",5,"Products")
UPDATE products ;
  SET unitprice = mfg_msrp.msrp*.90 ;
  FROM mfg_msrp ;
  WHERE mfg_msrp.productID = products.productID;
  AND mfg_msrp.discontinu = .F.
? SECONDS() - m.lnStart
SELECT Products
=TABLEREVERT(.T.)
CLOSE DATABASES ALL
```

用 VFP9 的 Derived Table（源表）功能在 SQL FROM 子句使用子查询

```
* Derived table (Sub-select as part of FROM)
* Returns customer ID and product count for all custs
* who have purchased at least 40% of the product line.
CLOSE DATABASES ALL
OPEN DATABASE Northwind
LOCAL lnPct
lnPct = .40
SELECT ;
   C.customerid, ;
   C.companyname, ;
   P.p_count AS Product_Count;
  FROM Customers C, ;
     (SELECT c2.customerid, ;
```

```
        COUNT(DISTINCT D.productID) AS p_count ;
         FROM Customers C2 ;
        INNER JOIN Orders O ;
           ON  C2.customerid = O.customerid ;
        INNER JOIN OrderDetails D ;
           ON O.orderid = D.orderid ;
        GROUP BY c2.customerid) AS P ;
 WHERE C.customerID = p.customerID ;
   AND P.p_count >= ;
       (SELECT (COUNT(*)*m.lnPct) FROM Products ;
        WHERE NOT EMPTY(ProductName)) ;
 ORDER BY p.p_count DESC
CLOSE DATABASES ALL
```

用 VFP9 的 Projection（发射）功能在 SQL 字段列表中使用子查询

```
* Projection (Sub-select as part of field list and
* complex expression)
* Sales total by customer with PCT of All cust sales
CLOSE DATABASES ALL
OPEN DATABASE Northwind
SELECT ;
   C.customerID, ;
   C.companyname, ;
   SUM(D.quantity*D.unitprice) AS CustTotal ,;
   (SUM(D.quantity*D.unitprice) / ;
     (SELECT SUM((quantity*unitprice)-discount) ;
        FROM OrderDetails D2) ;
    )*100 AS PctTotal ;
  FROM Customers C ;
 INNER JOIN Orders O ;
    ON C.customerID = O.customerID ;
 INNER JOIN OrderDetails D ;
    ON O.orderid = D.orderid ;
 GROUP BY C.customerID, C.companyname, O.orderID ;
```

```
ORDER BY pctTotal DESC

CLOSE DATABASES ALL
```

哦，参数？不，是 oParametrs!

原著：Andy Kramek & Marcia Akins

翻译：Fbilo

作为程序员，我们极大的依赖于在对象之间传递参数以操作它们的行为、或者报告执行的结果。然而，当我们需要传递多个参数的时候，这个过程很快就变得不灵活起来，因为，在 Visual FoxPro 中，参数必须以一个指定的顺序被传递。此外，我们只能从一个函数或者方法中返回仅一个值，所以，要传回多个值的问题变得很棘手。在这个月的专栏中，Andy Kramek 和 Marcia Akins 讨论了怎样使用一个参数对象来简化你的生活。

（译者注：哈哈，英雄所见略同！在参数较多、或者参数数量不确定的情况下，我也会通过建立一个 empty 对象、并给它添加参数属性、然后把这个对象当作参数来传递。所谓的参数数量不确定，是指有时候我会让一个中介者对象的方法来接收来自多个不同类型的方法的消息，这样的情况下，不同方法传递过来的参数数量就可能是不同的。）

安迪：这难道不会让你发疯吗？

玛西亚：你说什么？

安迪：试图去记住传递给函数们的参数的正确排列顺序。我最近在一个旧的 **FoxPro 2.6** 的系统上工作，该系统使用了大量的（并且非常隐蔽的）函数调用来处理日常工作，试图去记住正确的参数排列顺序实在是一个麻烦。例如，为了显示图 **1** 中的对话框，必须这样调用函数：

```
ln_resp=f0getdl("Continue anyway",1,1,0,.F.,0,"",;
"2","Yes","No",'','','Errors Found' )
```

而这个函数中竟然包含着 **500** 行的代码。

玛西亚：首先，那是一个老的应用程序。其次，这个函数的大小没什么关系，第三，你已经不再需要为了显示一个对话框而去写函数了，只管调用 **MessageBox()** 函数就是了。

安迪：是的，我明白。但这件事让我想到了关于参数传递的一些常见问题。你认为使用多少参数是合理的？事实上，你能够使用多少参数？

玛西亚：嗯，在帮助文件的 VFP 系统能力表中曾说到，在 6.0 版和 7.0 版中，最多可以传递 27 个参数，但有趣的是，8.0 和 9.0 中却只支持 26 个。

安迪：并且它看起来明显是错的。下面的代码在 6.0、7.0、8.0 和 9.0 中都能正常的工作。事实上，它甚至可以在 FoxPro 2.6 中通过编译而不发生错误，尽管在运行的时候会得到一个“堆栈空间不足”的错误。

```
DO ParmTest WITH 'Test1','Test2','Test3','Test4',;
'Test5','Test6','Test7','Test8','Test9','Test10',;
'Test11','Test12','Test13','Test14','Test15','Test16',;
'Test17','Test18','Test19','Test20','Test21','Test22',;
'Test23','Test24','Test25','Test26','Test27','Test28',;
'Test29','Test30','Test31','Test32','Test33','Test34',;
'Test35','Test36','Test37','Test38','Test39','Test40',;
'Test41','Test42','Test43','Test44','Test45','Test46',;
'Test47','Test48','Test49','Test50','Test51','Test52',;
'Test53','Test54','Test55','Test56','Test57'
FUNCTION ParmTest
LPARAMETERS p01,p02,p03,p04,p05,p06,p07,p08,p09,p10,;
 p11,p12,p13,p14,p15,p16,p17,p18,p19,p20,p21,p22,p23,;
 p24,p25,p26,p27,p28,p29,p30,p31,p32,p33,p34,p35,p36,;
 p37,p38,p39,p40,p41,p42,p43,p44,p45,p46,p47,p48,p49,;
 p50,p51,p52,p53,p54,p55,p56,p57
 *** 有多少参数?
 FOR lnCnt = 1 TO PCOUNT()
 lcVal = 'p'+ PADL( lnCnt,2,'0')
 ? &lcVal
 NEXT
RETURN
```

玛西亚：这太古怪了！你为什么停在了第 57 个参数的地方？是受到限制了吗？

安迪：不是，我只是输入的烦了。同样的代码也可以用在表单里面，所以，看起来不管帮助文件是怎么说的，我们可以使用的参数数量要比我们有理由使用的参数数量多上不少。不过，这并没有回到我的问题：“你究竟应该使用多少参数？”

玛西亚：个人的看法，我总是喜欢使用命名了的参数。

安迪：但 **VFP** 不支持命名参数。

玛西亚：不直接支持。如果你使用一个参数对象，你就能得到真正的命名参数了。通过采用这种办法，我们也回到了“要多少参数”的问题。

安迪：怎么说？

玛西亚：由于我们能够在一个对象中封装任意数量的参数，而且这样一来我们只需要传递一个参数就行了！

安迪：啊，是的！这里，我还可以看到这么做的另一个好处。由于我们只需要一个对象，所以我们可以在任何一个方向上使用它。

玛西亚：此话怎讲？

安迪：哦，既然我们仅能从一个函数或者方法接收到一个结果参数，那么，我们可以返回一个对象作为这个结果参数，并在对象里放入多个结果。

玛西亚：哦，是的，当然可以了。事实上，返回一个“结果”对象的办法是避开“每个函数或方法仅能返回一个结果参数”这种限制最简单的办法。现在，让我们总结一下把参数传入一个方法或者函数的问题。毕竟，这是最常见的情况。首先，我们要建立一个对象，然后把参数的名称作为属性名向该对象添加自定义属性。搞定以后，就可以简单的把参数的值赋值给这些属性了。

安迪：在 **8.0** 版中提供的 **“empty”** 类显然是建立参数对象的第一选择，它没有自己的属性或者方法，因此是极其轻量级的。

玛西亚：它也会是我的首选。当然，为了真正的添加属性，我们必须使用 **AddProperty()** 函数，因为 **empty** 类是没有 **AddProperty()** 方法的（**empty** 基类就是一片空白！）。所以我们看到的基本代码应该是这样子的：

```
oParams = CREATEOBJECT( 'empty' )
IF VARTYPE( oParams ) = "O"
ADDPROPERTY( oParams, <property name>, ;
```

```
    <property value >
    *** 根据需要重复调用 ADDPROPERTY()   ***
ENDIF
```

安迪：当然，这里有一个不言自明的假设：当你建立参数对象的时候，已经知道了那些参数的名称。

玛西亚：这不止是一个假设而已——如果你将要使用命名参数的话，这是必须的。总之，在这种情况下，那些参数的名称肯定是公开的接口的一部分，所以这个不成问题。

安迪：我没那么放心。毕竟，当使用连续的参数们的时候，参数们在源代码中使用什么名称是无关紧要的，因为唯一重要的是参数们在被传递的时候的排列顺序。参数的名称是在方法或者函数的内部用 **Parameters** 关键词指定的，所以在调用代码和被调用代码中使用的参数们之间没有直接的联系。总之，我建议总是使用 **PEMSTATUS()** 来确认参数对象中是否存在着某个期望的属性：

```
IF PEMSTATUS( oParams, 'cMyExpectedParam', 5 )
```

玛西亚：是的，当然了。你还可以使用更多的通用代码来判定被传递进来的有哪些参数。例如，你可以使用 **AMEMBERS()** 函数来生成一个包含着参数对象中所有属性名称的数组，就象这样：

```
lnProps = AMEMBERS( laPropertyNames, oParams, 0 )
FOR lnCnt = 1 TO lnProps
*** 取得参数名称
lcVarName = laPropertyNames[ lnCnt ]
*** 声明一个与属性同名的本地变量
LOCAL &lcVarname
&lcVarname = EVAL( "oParams." + lcVarName )
NEXT
```

毕竟，既然 **empty** 基类没有自带的属性，那么将出现在这个数组中的肯定然是被传递进来的参数了。

安迪：那非常简洁。但它假设我们总是会想要把参数转换成本地变量。通常（尤其是在给一个表单的 **Init()** 传递参数的时候），我们真正想要的是让它们成为（接收到参数的对象的）属性。如果我们为这些属性采用了一种命名转换方式，我们就可以处理不同的类型。例如，以“**p_xxxx**”方式命名的参数应该作为属性被建立在接收到参数的对象上，而以“**v_xxxx**”方式命名的参数则应该被作为变量来处理。所以我们

可以把上面的代码改成这个样子：

```
lnProps = AMEMBERS( laPropertyNames, oParams, 0 )
FOR lnCnt = 1 TO lnProps
*** 取得参数的名称
lcVarName = laPropertyNames[ lnCnt ]
luPropVal = EVAL( "oParams." + lcVarName )
*** 把参数处理成一个属性，或者一个变量
IF LOWER( LEFT( lcVarname, 2 )) == 'p_'
    *** 这是一个属性
    This.AddProperty(SUBSTR( lcVarname, 3 ), luPropVal)
ELSE
    *** 声明一个与参数属性同名的本地变量
    LOCAL &lcVarname
    &lcVarname = luPRopVal
ENDIF
NEXT
```

玛西亚：你猜怎么着，看到这里，我突然想到，其实我们可以把它做的更简单一些。为什么不使用一个基于 **Collection** 基类而不是 **empty** 的对象呢？

安迪：这样做的话，会怎样简化问题呢？

玛西亚：**Collection** 基类已经有了添加和检索一个数据项的方法，并且还有一个 **count** 属性可以告诉我们它有多少个数据项。我们可以使用参数的名称作为 **key**，值作为数据项。现在，用于建立参数对象的代码就会是这样了：

```
oParams = CREATEOBJECT( 'collection' )
IF VARTYPE( oParams ) = "O"
oParams.Add( <属性的值 >, <属性的名称> )
*** 根据需要重复调用 ADD() 方法 ***
ENDIF
```

安迪：对我来说，看起来几乎没什么区别。

玛西亚：在建立参数对象的一端是没什么区别，但是使用这个对象的过程就简化了。看这里：

```
FOR lnCnt = 1 TO oParams.Count
lcVarName = oParams.GetKey( lnCnt )
luPropVal = oParams.Item( lnCnt )
*** 把参数处理成一个属性，或者一个变量
IF LOWER( LEFT( lcVarname, 2 )) == 'p_'
    *** 这是一个参数
    This.AddProperty(SUBSTR( lcVarname, 3 ), luPropVal)
ELSE
    *** 声明一个与参数属性同名的本地变量
    LOCAL &lcVarname
    &lcVarname = luPRopVal
ENDIF
NEXT
```

安迪：啊哈，我明白了。你避免了使用 **AMEMBERS()** 函数的需要，并且还不需要使用 **EVAL()** 来取得被传递的参数的值。不过，这里有一个问题。如果你想传递一个数组作为参数的话会发生什么事情？你不能直接给一个集合添加一个数组，必须得把这个数组转换成一个集合，然后再把这个集合作为一个数据项添加给参数对象。

玛西亚：这个我倒没有想到。如果使用 **empty** 对象的话，你怎么把一个数组作为一个属性添加给它？

安迪：就使用 **ACOPY()** 函数：

```
DIMENSION laFruit[3]
laFruit[1] = 'Apples'
laFruit[2] = 'Oranges'
laFruit[3] = 'Grapes'
oParams = CREATEOBJECT( 'empty' )
AddProperty( oParams, 'a_fruit[1]' )
ACOPY( laFruit, oParams.a_fruit )
```

玛西亚：**OK**，前面我不知道你可以这么做。我看到你在定义数组属性的时候给属性的名称使用了一个“**a_**”作为前缀，接收参数的时候你准备怎样把它取出来？是转换成一个变量呢、还是一个属性？

安迪：我不能确定。我建议扩展我们的转换规则来包含一个“a_”前缀作为数组变量、以及一个“z_”前缀作为数组属性。不管怎么说，我们可以把这个决定留到实际工作中再去实现。

玛西亚：打住。我曾假设这部分代码将是非常通用的。换句话说，我们将会有一个 unPackParameters() 方法，无论何时需要处理一个参数对象我们都可以调用这个方法。而你在谈的则是这个方法的代码将被直接嵌入到接收参数对象的方法中去。

安迪：嗯，是的。毕竟，如果你将要把参数的值放到本地变量中去，那么就没有多少理由把这个任务放在一个次要的方法中去。调用方法将无法获得这些值。

玛西亚：今天早上我肯定误服了愚蠢药了！我没想到这个问题。从那个角度来看，其实有两种不同的情况：

- 第一种是我们只是想要简单的使用命名参数。在这种情况下，没必要使用通用的代码，因为接收参数的方法知道参数的名称，并且可以直接从对象中读取它们并对它们进行相应的处理。
- 第二种情况是参数的数量不确定、或者在运行时参数的名称可能会不同（例如来自多个表中的大量字段名称），这种情况就比较复杂了。在这种情况下，我们得使用通用的代码来处理参数对象，并且总是会为从参数对象中取得的不管什么东西都为之建立属性。然后由实际情况中的方法来根据需要利用这些属性。

安迪：有道理。它还避免了依赖于命名转换来决定怎么处理不同的东西的问题。不管怎么说，通用的处理代码总是需要包含一个检查以确认一个指定的属性是否存在的，但对于它给予我们的灵活性来说，这只是一点非常小的内务操作而已。

玛西亚：我们最好还是区别对待数组和其他属性。幸运的是，VFP 9.0 增加的 TYPE() 函数可以对被测试的数组返回一个“A”值、对集合返回一个“C”值。你只需要传递一个 1 作为第二个参数就行了，象这样：

```
lcVarName = laPropertyNames[ lnCnt ]
lcType = TYPE( lcVarName, 1 )
DO CASE
 CASE lcType = 'A'
      *** 这是一个数组
 CASE lcType = 'C'
      *** 这是一个集合
```

```
    OTHERWISE
        *** 就是属性
    ENDCASE
```

安迪：这很好，我刚刚还在想如果被传递进来的参数中有一个集合的话该怎么办呢！虽然，经过思考发现，处理的办法还是不变的——它还是一个属性。但能够知道我们测试的属性是不是一个集合还是有好处的。那么我们能把所有代码都放在一起吗？

玛西亚：我想可以。我建议应该把它建立成一个全局函数，这样一来我们就不需要向每个基类都添加方法了。

安迪：哦，你可以把它放到一个应用程序对象类里面。

玛西亚：哪种办法都可以，不过，不管在哪种情况下，你都需要给函数传递两个对象引用——一个引用的是将要接收参数的对象，而另一个引用的就是参数对象本身。本专栏下载文件中有作为这段代码作为独立函数的版本，但这里是完整的代码：

```
LPARAMETERS toSource, toParams
LOCAL ARRAY laPropertyNames[1]
LOCAL lnProps, lnCnt, lcVarName, lcType
*** 这里我们需要两个对象！
IF VARTYPE( toSource ) # "O"
 ASSERT .F. MESSAGE ;
 "必须给 UNPACKPARAMOBJ 传递一个对象引用！"
ENDIF

IF VARTYPE( toParams ) # "O"
 ASSERT .F. MESSAGE ;
 "必须给 UNPACKPARAMOBJ 传递一个参数对象"
ENDIF

*** 首先，取得在对象上所有参数的名称
lnProps = AMEMBERS( laPropertyNames, toParams, 0 )

*** 现在处理数组
FOR lnCnt = 1 TO lnProps
```

```
        *** 取得参数的名称
        lcVarName = laPropertyNames[ lnCnt ]

        *** 这里我们得到的是哪种类型的参数？
        lcType = TYPE( "toParams." + lcVarName, 1 )

        *** 检查属性是否存在
        IF NOT PEMSTATUS( toSource, lcVarName, 5 )
            *** 根据参数的类型建立属性
            IF lcType = 'A'
                *** 这是一个数组
                ADDPROPERTY( toSource, lcVarName + "[1]" )
            ELSE
                *** 这是一个集合或者简单的属性
                ADDPROPERTY( toSource, lcVarName )
            ENDIF
        ENDIF

        *** 现在生成这个属性
        IF lcType = 'A'
            *** 这是那个数组
            ACOPY( toParams.&lcVarName, toSource.&lcVarName )
        ELSE
            *** 一个简单的属性或者集合
            toSource.&lcVarName = toParams.&lcVarName
        ENDIF
        NEXT

        *** 不返回值，而是就返回一个常规的 .T.
        RETURN
```

安迪：东西不多！只是有一个问题：为什么这里我们用了宏替换而不是 **EVAL()** 来生成那些属性？

玛西亚：因为在 **ACOPY()** 函数中，**EVAL()**无法工作（在引用一个数组的时候，你不能使用一个名称表达式，因为这么做你只能获得数组的第一个元素）。所以，我们必须得使用宏替换——这是那些没有后

备方案的情况之一。

下载文件：**503KITBOX.ZIP**