FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2004 年 9 月刊

拆分工具（第三部分） — Lauren Clarke & Randy Pearson 著 CY 译

Page.21

这是用 VFP 处理文本的系列文章的第三部分，先前的文章已经重现了 VFP 纯文本处理性能，并给出了如何对规则表达式进行性能补充——强大的模型描述语言。这个月 Lauren Clarke 和 Randy Pearson 将创建一个令人惊奇的函数，它可以用纯 VFP 表达式无缝地配合了规则表达式。这个函数，它是很有用的，将会作为下个月的拆分基类的基本方法。（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 VFP9.0 中通过 FFC 类库使用 GDI+(第二部分) — Walter Nicholls 著 Yasur 译

Page.34

在八月份的早些时候，微软已经预先发布了一些伴随着 VFP9 Beta 版的额外的文件，这其中就包括了一个强劲的为 GDI+图形所服务的功能类库(检查一下 VFP 官方网站 http://msdn.com/vfoxpro 上的下载信息)。上个月， Walter Nicholls 为大家引入并解释了 GDI+ FFC 类库，在这第二篇(此系列共包含四篇)文章中，它将说明如何使用 GDI+类库来扩展 VFP9 的报表系统。

国内原创文章

VFP 联姻 Flash(第一部分) — Yasur 著

Page.44

用 Flash 可以做出非常漂亮的界面和动画效果，而 vfp 在这方面的确是先天不足，将两者的优点结合起来肯定是一件令人激动的事情，用美妙的 Flash 界面、互动功能结合 Vfp 的强大的数据处理及自身的数据库功能将会大大提高应用程序的感官、功能和竞争力。我按照这个思路，试着做了几个例子，结果非常满意。我想这应该是一件很有意义的事情，无限畅想。。。

“哚、哚、哚”
有人在家吗？（第一部分）

原著：Andy Kramek & Marcia Akins

翻译：Fbilo

开始开发一个应用程序时总是会出现的一个问题是“安全性”。在这个月的专栏中，Andy Kramek 和 Marcia Akins 讨论了多种应用程序安全模式。尽管他们以之上手的是一个相当简单的概念，可他们很快发现，概念的实现将会变得非常复杂。

安迪：我被要求在一个应用程序中实现一个用户等级安全机制，我想问你一下是否有什么类适合我用的（这样我就可以省下重新发明轮子的工夫了）。

玛西亚：恐怕这得取决于你说的“用户等级”的准确含义了。

安迪：我不是很确定。我们还没涉及细节问题，不过，用户的需求看来应该是并非所有的用户都能够访问应用程序的每个模块；并且，即使他们能用每个模块，也不一定能够看到或者使用指定表单上的每样东西。

玛西亚：你要做的第一件事情是必须定下来怎样识别一个用户。

安迪：这个简单——我们有一个登陆界面，用户要输入一个用户名和密码，这样我们就知道谁在访问系统了。

玛西亚：听起来很有道理。下一步是定义你有多少种类型的用户。

安迪：什么？你说的“用户的类型”是什么意思？

玛西亚：我想我实际上的意思是：“你到底需要多少种不同类型的访问方式？”你看，如果只是任何一个特定用户能否访问某个特定的功能这样简单的问题，你可以在菜单中使用自带的 **SKIP FOR** 功能来实现它。

安迪：但我不想象这样在代码中写死。

玛西亚：是什么让你认为必须要在代码中把某样东西写死？只要做一个名为 **AllowUserAccess()**的全局函数，给它传送菜单项名称和用户 **ID**，这样就可以让这个函数来判定该用户是否能访问那个菜单项了。

安迪：这个我可以做。事实上，我甚至把这个写成了一个类而不是一个函数，也许它将成为我的表单管理器的一个部分。

玛西亚：但并非每个菜单项都是用来打开一个表单的。做个类起来似乎有点象是拿着重磅榔头打核桃，只要做个函数就够了。

安迪：我猜你是对的，但这样我还是有个问题。当你使用 **SKIP FOR** 的时候，在菜单栏上的菜单项虽然被禁用了但还是能看得到，而有些用户看到这样的情况时就觉得很不舒服。

玛西亚：这很容易处理：用 **GenMenuX**（由 **Andrew Ross MacNeill** 做的一个工具，可以从 **www.meistermacneill.com/genmenux.htm** 下载）来生成你的菜单。它允许你从菜单中完全删除一个菜单项而不是仅仅禁用它。

安迪：那太棒了！不过，再多考虑一下的话，我发现这样做还不够。假定我们有一个用户，他被允许浏览某些数据，但没有权限去修改它。或者某个用户被允许修改某些已有的数据，但是不能添加或者删除任何东西。

玛西亚：这意味着我们除了要在菜单级别以外，还要在表单级别上也实现安全机制。如果我们明智的设计我们的类和元数据的话，这应该问题不大。问题是：“我们要怎么定义用户的访问？”最简单的办法是赋予每个用户一个从 **0** 到 **10** 的访问级别。级别 **0** 表示没有访问权限，级别 **10** 表示允许访问所有东西。

安迪：这个办法不好，因为它首先假定任何特定的用户在应用程序的所有部分中都有着完全一样的访问级别。我想要根据每个用户要使用哪个表单来赋予他不同的访问级别。

玛西亚：我理解你的意思。**AllowUserAccess()**函数对菜单工作，以判定用户是否能访问某个表单。我们还需要对表单本身添加第二层控制。对于一个表单，有三种基本的模式：

- 只读——用户可以看表单，并使用导航和查询功能，但不能对数据做任何改动。
- 仅编辑——除了只读的权限以外，用户只能对已有的数据进行改动，但不能添加或删除数据。
- 无限制——用户拥有访问该表单可能提供的所有功能的权限。

安迪：在某些特殊的情况下，还可能有一些其它的选项（比如“可以添加但不能删除”），不过大致上都是这样三种了。

玛西亚：没错，是这样的。无论如何，记住无限制选项意味着它允许用户访问表单能做的任何事情。如果表单上没有删除功能，那么即使用户有着无限制访问的权限，他/她也不能删除记录。

安迪：但我还是看不出用这种办法怎么能让不同的用户用不同的方法访问同一个表单。毕竟，如果要在菜单上看到某个表单需要访问等级“4”，那么，所有要访问这个表单的用户的访问等级都必须是“4”了。你的意思是否是我们需要添加一些其它值来控制表单是怎样被访问的？

玛西亚：这个不重要。要访问某个表单，一个用户必须具有等于或大于“4”的访问等级。你可以在表单上写入代码以当表单初始化的时候检查用户的访问等级。如果用户的访问等级就等于能在菜单上看到这个表单的访问等级，那么表单就把自己设置未只读模式。如果用户的等级大于这个最小值，那么表单再相应的改变它的模式。

安迪：嗯？我没听明白。

玛西亚：表 1 说明了我的想法。假定在菜单上看到某个表单的访问等级定义为等级 4（或更大的值）。

表 1、基于用户的访问等级设置表单的模式

用户访问等级	表单模式
等级 0～3	菜单项不可用
等级 4	只读
等级 5	仅编辑
等级 6～10	无限制

安迪：啊！现在我明白了。这样做很聪明，因为它意味着在表单上的代码是完全通用的。

玛西亚：这种办法只有一个问题。它是假定应用程序的所有部分都使用同样的访问等级。假使我们有三个模块：客户关系管理（CRM）、销售订单处理（SOP）、人际关系（HR）。如果我们允许 10 个等级，那么，为了让一个用户拥有对 CRM 的完全访问权限、对 SOP 的只读权限、并且不能访问 HR，我们必须象表 2 中那样给他分配等级。

表 2、单访问等级的问题

访问等级	CRM	SOP	HR
只读	3	4	8
仅编辑	4	5	9
无限制	5	7	10

安迪：我知道你的意思了。为了满足这种类型的需求，模块必须被按等级来设置。结果是：任何能够访问 SOP 的人都会自动获得对 CRM 的无限制权限，并且任何能够访问 HR 的人都会自动获得对 CRM 和 SOP 的无限制权限。不过，我的程序里面与此不同。我无法保证我的模块们是否会具有这样的关系。

玛西亚：这样的话，你必须决定是采用一个基于用户的模型还是基于角色的模型。

安迪：有什么区别？

玛西亚：在一个基于用户的模式里，你必须为每个用户专门指定可以访问项目。这样做便于编程，但对于你的最终用户来说就麻烦了，因为每次添加一个新用户或者规则发生了变动，就必须为每个用户指定一遍权限。这意味着系统管理员将不得不去维护一个象表 3 那种类型的表，在该表里，“X”表示没有访问权限，“R”表示只读，“E”表示编辑，而“U”表示无限制。

安迪：这种办法可以很好的进行控制，但对于拥有大量用户的大型应用程序来说，维护起来的确也是件痛苦的事情。不管怎么说，为它写代码是简单。它就是一个基于表单名称和用户 ID 的查询。基于角色的安全机制的情况如何？

表 3、基于用户的安全模式

	CRM1	CRM2	CRM3	SOP1	SOP2	SOP3	HR1	HR2	HR3
User1	U	U	E	R	R	R	X	X	X
User2	R	R	X	E	E	U	X	X	X
User3	X	X	X	X	X	X	U	E	U
User4	U	U	E	R	R	R	X	X	X

玛西亚：它基于这样一个事实：在分析不同的用户怎样访问应用程序的时候，其模型与基于用户的模型是一样的。见表 3，用户 1 和用户 4 有着完全一样的访问权限，所以，代替把所有的选项定义两次（如果这样做的话就是基于用户的模型了）的是：我们为每种唯一的情况建立一个角色，然后把角色分配给用户（见表 4 和表 5）。

表 4、基于角色的安全模型

CRM1	CRM2	CRM3	SOP1	SOP2	SOP3	HR1	HR2	HR3

角色 1	U	U	E	R	R	R	X	X	X
角色 2	R	R	X	E	E	U	X	X	X
角色 3	X	X	X	X	X	X	U	E	U

表 5、给用户分配角色

用户 ID	角色
1	1
2	2
3	3
4	1

注意：这里说的“角色”主要指的是一种特定的访问样式——跟是否专用于一个特定的工作或者用户没什么关系。

安迪：是的，我可以看出这种办法将会较为容易管理和维护。不过，它实际上依赖于一个用户只有一个角色的假定。

玛西亚：同意，但那是一个非常可以理解的假定。毕竟，可以被定义的角色的数量没有什么限制，即使碰到那种最糟的每个用户都需要一个唯一的角色的情况，与基于用户的模型相比也没什么坏处。事实上，对大多数人只需要定义很少几个角色。特殊情况总是会有的，可这样也很容易用特殊角色来处理。

安迪：好，我看到现在已经讲到了所有我曾经想到过的问题，但我刚刚又想到了一个。前面我们都是假设一个用户在访问表单的时候将会访问表单上的每一件东西。事情并非总是如此，我刚刚意识到这个模型不能处理字段级别的安全性。

玛西亚：这下麻烦了。你真的想要那种根据登录进来的用户是谁来决定表单上字段的使能和禁止（或者甚至可见和不可见）的东西吗？

安迪：我可以举个例子，尤其是当你在处理那些存储象薪水、身份信息、或者医疗记录这样的敏感信息的应用程序的时候。使得只有特定的用户才能看到这样的信息这一点是非常有意义的，并且，在某些情况下，这样的功能也是法律所要求的。我们或者要为同一个表单建立几个不同的版本，或者就必须在运行时处理这个问题。

玛西亚：我猜你是对的，考虑到你在处理多个模块，这看上去也不是没有道理。我认为我们还是可以用角色来处理它——它只需要对元数据做一些小扩展和几个假定。

安迪：是哪几个？

玛西亚：好，首先我们必须假设，在默认情况下，一个拥有对一个表单的访问权限的用户，自然也拥有对表单上每一件东西的访问权限。其次，同样在默认情况下，每个控件都从表单上继承了模式。换句话说，如果表单处于只读模式，那么表单上的每个控件都将处于只读模式，即使当前用户拥有编辑它们的权限也是这样。

安迪：啊，我明白了。表单还是老板，没有哪个控件可以超越它的设置。这就是说，我们实际上要实现的是那些例外的情况。我喜欢这样，因为所有我们需要管理的是那些一定会被利用的特殊条件，我猜想这会在元数据中被处理，是吗？

玛西亚：是的，我们需要四个查找表——一个用于模块，一个用于菜单项，第三个用于表单，最后一个用于限制字段。

安迪：看起来要做大量的工作。

玛西亚：是的，我看不出你怎么能避免一些设置工作。不过，记住：只有当要加上某些特殊限制的时候你才需要在这些表中添加记录。默认情况下，所有的用户都被假定为拥有对任何东西的无限制访问权限。

安迪：打住，这是否意味着我们根本不需要角色了呢？

玛西亚：不，我们还是需要给某些事情分配限制的，不管是一个用户还是一个角色。如前所见，角色是最好的。

安迪：好吧！我开始以为我们是要走遍所有的房子只是为了找出一条完全不同的路线。那么，开始写代码吧。

玛西亚：我觉得它可以被封装入一个在应用程序启动时建立、并可以在任何需要的时候调用的“安全性管理器”类里。不过，具体的实现细节将要等到下个月的专栏了。

用 MenuHit 和 MenuContext 扩展 VFP 9 的 IDE

原著: Doug Hennig

翻译: Fbilo

在 VFP 9 中的一个关键主题是扩展性。你可以通过报表事件扩展报表设计器、通过新的 Reportlistener 基类扩展报表引擎。而现在，你甚至可以通过跟踪系统菜单和快捷菜单的点击来扩展 IDE 界面。这个月，Doug Hennig 向你展示了怎样用你所无法想象的方式去定制 VFP 9 的 IDE 界面。

长久一来，VFP 一直在提供多种挂钩（Hook）到交互式开发环境的各个方面的途径，并且每个版本都做的更好。FoxPro 2.x 允许我们通过将应用程序的名称改为指向象 _GENSCRN 和 _GENMENU 这样的系统变量来替换特定的 Xbase 部件的功能。VFP 3 给了我们生成器，用它，我们可以自动化或简化在类和表单上的工作。

VFP 6 添加了 Project Hook，让我们可以在当用户在项目管理器中执行象添加或删除文件这样的操作的时候接收事件。IntelliSense 是 VFP 7 中一个主要的新功能，它带有一种我们可以轻松扩展的数据驱动的途径。VFP 8 提供了可以自定义的 Toolbox 和其它一些非常有用的 IDE 增强。

而在 VFP 9 中，Microsoft 已经揭开了 IDE 的面纱。由于报表设计器会引发报表事件，我们能够完全改变它的对话框的表现和行为。新的 ReportListener 基类允许我们接收象一个报表运行这样的事件，提供象旋转标签、动态格式化、以及自定义如 Grapha 这样的着色对象的功能。我们还能够通过对 BindEvent()的增强来对 Windows 事件做出反应——当我写这篇文章的时候，这个功能在 Beta 版中还不能用，但它将给予我们对 Windows 系统事件更好的访问途径，包括那些与 VFP IDE 相关的会调用带 hWnds 的窗口的系统事件。

总之，这篇文章关注的焦点，是挂钩（Hook）到系统菜单和系统快捷菜单（那些大量在属性窗口或数据库设计器这样的地方单击鼠标右键时会显示的菜单）的 hits（指用户从 VFP 系统菜单标题中选择了一项）。我将从怎样做到这一点开始，然后再举一些实际的例子。

译者注：

在本文中，经常提及几个菜单术语 PAD、BAR、ITEM，这几个词在各种翻译文献中译法各不相同，比如 PAD，有菜单标题、菜单板、选择页各种译法，在本文中，暂且统一翻译为菜单标题，按我个人的理解，一个 PAD 就是一个 POPUP 的标题。而 BAR 和 ITEM 则都可以翻译成菜单项：通常一个 POPUP 下的每一项被称作 BAR，如果其中某一个是有子

菜单的，那么它的子菜单中的各项就叫做 ITEM。本文根据它们所要表达的意思将它们都翻译做菜单项。

MENUHIT

为了跟踪一个系统的 hit，需要在 IntelliSense 表中建立一个脚本记录，其中，TYPE = "S"（表示“脚本”），ABBREV = "MENUHIT"，要运行的代码则放在备注型字段 DATA 中（IntelliSense 表由系统变量 _FOXCODE 指定，默认是在 HOME(7) 目录中的 FOXCODE.DBF）。代码需要接收一个对象作为参数。这个对象有几个属性，对于 MENUHIT 来说最重要的部分如表 1 所示：

表 1、被传递给 MENUHIT 脚本的参数对象的属性提供了关于菜单 hit 的信息

属性	解释
UserTyped	菜单 hit 所来自的菜单标题（pad）的标题
MenuItem	用户选择的菜单项的标题
ValueType	给 VFP 的一个返回值——值为空表示继续执行默认的行为，若值为“V”或“L”则表示阻止默认的行为（类似于在类代码中使用 NODEFAULT）

这里是一个简单的示例（取自这个月下载文件中的 SimpleMenuHit.PRG）

```
text to lcCode noshow
lparameters toParameter
wait window 'Pad: ' + toParameter.UserTyped + ;
     chr(13) + 'Bar: ' + toParameter.MenuItem
endtext

delete from (_foxcode) where TYPE = 'S' and ABBREV = 'MENUHIT'
insert into (_foxcode)   (TYPE, ABBREV, DATA) ;
     values ('S', 'MENUHIT', lcCode)
```

这段代码只是简单的显示被选中菜单项所在的菜单项和菜单标题的标题，然后继续执行菜单项默认的行为。这只是用于测试，但对于“真实”的代码来说，你可能会需要禁止默认的行为而代之以你自己的。在那种情况下，请将参数对象的 ValueType 属性设置为“V”或者“L”（你也许会问为什么有两个可用的值，答案是：这些值是由 IntellSense 管理器用于其它目的的，而 VFP 开发组决定让 MENUHIT 使用同样的机制。事实上，不管你用哪个都没关系）。

这里是一个取自 DisableExit.PRG 的示例。它会禁止文件菜单中的 Exit 功能（也许你可以用它来给同事开一个愚人节玩笑。）：

```
text to lcCode noshow
lparameters toParameter
if toParameter.UserTyped = 'File' and ;
      toParameter.MenuItem = 'Exit'
      toParameter.ValueType = 'V'
endif toParameter.UserTyped = 'File' ...
endtext

delete from (_foxcode) where TYPE = 'S' and ABBREV = 'MENUHIT'
insert into (_foxcode)   (TYPE, ABBREV, DATA) ;
      values ('S', 'MENUHIT', lcCode)
```

这段代码虽然看起来没错，可是当你从文件菜单中选择退出的时候，VFP 还是会退出。看来，除了将 ValueType 设置为“ V”或者“ L”以外，某些功能还要求 MENUHIT 的代码返回一个 .T. 来阻止默认的行为。请在 EndIF 语句前面加上 Return .T. 来阻止 Exit 去从 VFP 退出。

另一个 MENUHIT 问题：如果你使用的是一个本地化版本的 VFP，比如德语版，会如何？在那种情况下，用“File”和“Exit”来检查是无效的，因为它们不是出现在菜单中的标题。这是一件你必须要留心的问题，如果你把一个 MENUHIT 脚本发布给其它 VFP 程序员的话，你也必须要让你的最终用户留心这个问题。

MENUHIT 不断出现

既然我们谈到了发布一个 MENUHIT 脚本给别人的话题，那么顺便再来谈谈其它相关的话题。如果你有个做某些事情非常酷的脚本、而别的什么人也有这么一个他自己的脚本，而你想把两个脚本都用起来——这样的话该怎么办？问题在于：一共只能有一个 MENUHIT 记录存在（即使 IntelliSense 表中有多条记录，VFP 也只会使用其中的第一条）。基于这个原因，我认为，最好让 MENUHIT 记录将执行期望的自定义 IDE 任务委托给别的什么东西而不是自己来做。

这么做最简单的办法是给 IntelliSense 添加一些记录，并让 MENUHIT 记录使用它们来执行真正的任务。虽然有多种办法，不过我认为一个 TYPE = "M" 的记录更适合现在这种情况，因为“ M”可以代表菜单，而且目前的 IntelliSense 表中还没有哪个记录的 Type 是 M 的。例如，要处理 Exit 功能，你可以添加一条 TYPE = "M"、ABBREV = "Exit"、要执行的代码放在 DATA 字段中的记录。

为了完成这个任务，我们需要添加一个“标准”的 MENUHIT 记录。这个记录的代码会从表中搜索一条 TYPE = "M"、ABBREV 设置为用户选择的菜单项标题的记录，如果找到了，则运行该记录的备注字段 DATA 中的代码。这里是处理这个任务的代码：

```
lparameters toParameter
local lnSelect, ;
      lcCode, ;
      llReturn


try
      lnSelect = select()
      select 0
      use (_foxcode) again shared order 1
      if seek('M' + padr(upper(toParameter.MenuItem), ;
            len(ABBREV)))
            lcCode = DATA
      endif seek('M' ...


      use
      select (lnSelect)
      if not empty(lcCode)
            llReturn = execscript(lcCode, toParameter)
            if llReturn
                  toParameter.ValueType = 'V'
            endif llReturn
      endif not empty(lcCode)
catch
endtry
return llReturn
```

运行 **StandardMenuHit.PRG** 来把这段代码安装到 **IntellsSense** 表中去。关于这段代码，有一些有趣的东西。首先，我通常是用 **ORDER <Tag** 名**>** 而不是 **ORDER <n>** 来给表设定排序的。然而，**IntelliSense** 表有些特殊：如果你打开了这个表，并使用 **ATAGINFO()** 来从中返回这个表的索引信息的话，你会看到那里有两个 **Tag**，它们都被标记为主索引，并且都没有索引标识的名称。所以，你必须使用 **ORDER 1** 或者 **ORDER 2** 来给这个表设定排序顺序。

第二件事情是这段代码被封装再一个 **TRY** 结构中以防止任何错误——例如象打开表出错、或者也许存在于其它记录中的代码中的错误。

第三个问题是：这段代码不检查用户选择的菜单项所出自的菜单标题的标题，只检查了菜单项的标题。这是因为我基于性能的原因决定执行一个 SEEK，并且使用的索引标识是 UPPER(TYPE + ABBREV)。由于这是一个小表，你也可以通过将菜单标题标题放入 EXPANDED 字段、使用 LOCATE FRO TYPE = "M" AND ABBREV = toParameter.MemuItem AND EXPANDED = toParameter.UserTyped 来保证找到准确的记录。

这个版本的 VFP 9 Beta 发布的时候，Microsoft 还没有决定默认情况下 IntelliSense 表中是否要存在这么一个“标准”的 MENUHIT 记录。如果（正式版中）没有的话，他们可能会提供一种简单的途径来添加这么一个记录，例如通过 Solution Sample（Solution Sample 是一些演示大量 VFP 功能的示例，通过 Task Pane 管理器可以很容易的访问它们）。

MENUHIT 的最后一个问题：如果备注字段 DATA 中的代码有任何编译错误，你不会看到错误消息——VFP 会忽略错误代码而使用默认的行为。这是令人头疼的事情，因为这样的话你就无法知道你的代码到底错在哪里了。

有什么好处？

OK，现在我们可以挂钩到 VFP 的菜单项了。那么我们能用它来干什么？

我想到的头一件事情，是换掉新建属性和新建方法对话框。既然我们现在已经可以保留自定义属性和方法的大小写方式了（参见 Foxtalk 2.0 2004 年 6 月我的专栏），那么我当然希望 VFP 使用我在新建属性或者新建方法对话框中输入的大小写方式，以避免那些打开 MemberData 编辑器去修改大小写方式的额外步骤了。此外，它令我恼火的还有总是得按下“添加”按钮来增加一个新属性和方法，然后再按下“关闭”按钮来退出对话框。因为我通常总是一次只要添加一个属性或方法，我当然希望能够看到有一个同时添加了成员并关闭这个对话框的按钮了。

由于现在用 MENUHIT 的办法让我们可以跟踪表单和类菜单中的新建属性和新建方法菜单项，我们将能够自建一个有着我们需要的行为的对话框代替品。

图 1 显示了这么一个对话框，并有着下列功能：

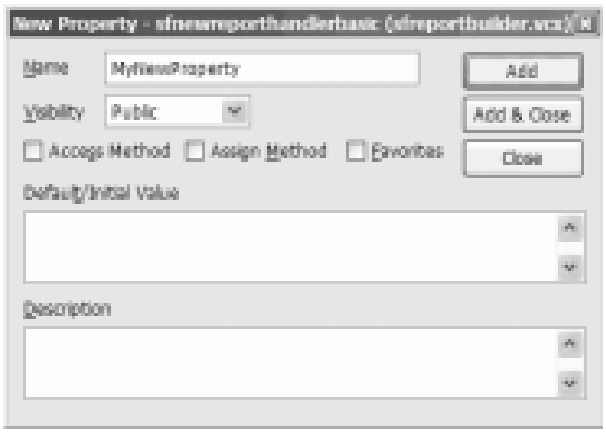
- 它会自动更新 _MemberData 属性（根据情况添加这个属性），这样，为新成员所输入的大小写将被使用（即使是对新建的 Access 和 Assign 方法也有效），并且如果选中了对话框中的 Favorites 选项，那么这个成员将显示在属性窗口的 Favorites 页上。
- 它是非模式对话框。这样一来，你就可以让它总是开在那里，添加一些属性或方法、切换到其它工作、然后

再切换回来添加更多的成员。

- 它是可停靠的：试一下把它停靠到属性窗口上。它非常酷！
- 它是可以缩放的。缩放后的大小和位置会保存到你的资源文件（**FOXUSER**）中去。
- 它有一个 **Add & Close** 按钮来一次执行两个任务。
- 属性的默认值会根据属性的数据类型自动被设置为一个相应的值（需要你采用匈牙利记法给属性命名）。例如，**lTest** 会是逻辑型，所以它的默认值是 **.F.**。对于 **nTest**，其默认值是 **0**。
- 它会隐藏而不是禁止不可用的控件。由于这个对话框同时用于类和表单设计器的新建属性和新建方法，对于某个特定的实例来说，某些选项可能不可用。
- 如果你输入的是非法的名称，当你输入的时候就会被禁止，而不是等到单击“**Add**”或者“**Add & Close**”按钮的时候才去检查。
- 只有你输入了一个名称以后，**Add** 按钮才会被启用。

这里我们就不看组成这个 **NewPropertyDialog.APP** 应用程序的代码了。它并不复杂，其核心原理是：它调用在类或表单设计器中被编辑对象的 **AddProperty** 和 **WriteMethod** 方法来添加一个新属性或者方法。这两个方法都将接收 **VFP** 中的两个新参数：**Visibility**（**1** 表示 **Public;2** 表示 **Protected**，**3** 表示 **hidden**）和用于新属性和新方法的 **description**（描述、解释）。

图 **1**、这个新建属性和方法的对话框比原来那个多出了许多的功能。



只要简单的 **DO NewPropertyDialog.APP** 一下，就可以把这个对话框的替代品注册到 **IntelliSense** 表中去，然后你就可以使用它了。它会向表中添加一条前面提到的 **MENUHIT** 记录以及另两条记录（一条用于“新建属性”，另一条用于“新建方法”），这两条记录的 **DATA** 字段中有着同样的代码以使用 **NewPropertyDialog** 类。在这段代码中，“**<path>**”表示在你的系统上 **NewPropertyDialog.APP** 所在的路径（当你运行这个 **APP** 的时候，它会自动在 **DATA** 字段的代码中插入正确的路径）。

```
lparameters toParameter
local llReturn, ;
      llMethod, ;
      llClass

try
      llMethod = toParameter.MenuItem = 'New Method'
      llClass = toParameter.UserTyped = 'Class'
      release _oNewProperty
      public _oNewProperty
      _oNewProperty = newobject('NewPropertyDialog', ;
           'NewProperty.vcx', ;
           '<path>NewPropertyDialog.app', llMethod, llClass)
      _oNewProperty.Show()
      llReturn = .T.
catch
endtry
return llReturn
```

现在，当你从表单或者类菜单中选择“新建属性”或“新建方法”的时候，新的对话框就会出现了。

MENUCONTEXT

除了挂钩到 VFP 的系统菜单的选中操作以外，你还可以跟踪系统快捷菜单（比如当你在属性窗口中单击鼠标右键时会出现的那个）。这是通过使用与 MENUHIT 同样的机制来实现的，除了在 IntelliSense 表中的 ABBREV 字段中包含的将是“MENUCONTEXT”而不是“MENUHIT”以外。

跟 MENUHIT 一样，MENUCONTEXT 记录中的代码要接收一个对象参数。在这种情况下该对象有三个有趣的属性：Item，一个显示在快捷菜单中所有标题的数组；ValueType，跟 MENUHIT 中的那个的用途相同；还有 MenuItem，是快捷菜单的 ID。这里是 MenuItems 的部分有效值：在命令窗口快捷菜单中的是“24446”，项目管理器快捷菜单的是“24447”，属性窗口快捷菜单的是“24456”。我是怎么找出这些值来的？就是建立一条 MENUCONTENT 记录，其代码简单的显示 toParameter.MenuItem 的值，然后我在各种地方到处点击鼠标右键...。象对付 MENUHIT 一样，你应该把 ValueType 设置为“ V”或者“ L”并返回一个 .T. 来阻止默认的行为（快捷菜单的显示）。

出于各种原因，MENUCONTENT 没有 MENUHIT 那么容易使用。首先，改动 Items 数组的内

容不会反应到菜单的显示上。例如，下面这段代码看起来对于快捷菜单根本无效：

```
lparameters toParameter
toParameter.Items[1] = 'My Bar'
```

同样，下面的代码对于菜单中新的菜单项也不起作用：

```
lparameters toParameter
lnItems = alen(toParameter.Items) + 1
dimension toParameter.Items[lnItems]
toParameter.Items[lnItems] = 'My Bar'
```

其次，没有办法改变当一个菜单条被选中的时候会发生的事情。例如，你可能想让命令窗口快捷菜单的属性菜单项显示你自己的对话框而不是系统的那个。问题在于：没有一个参数对象的属性能指定当一个菜单项被选中的时候该怎样去做——它们是建立在菜单自身内部的。

所以，看来我们可以改变菜单的唯一途径是：使用我们自己的快捷菜单并阻止默认的行为。不过，这样做还有一个困难：由于你必须将整个菜单换成你自己的菜单，当你的某些菜单项被选中的时候要怎样告诉 **VFP** 去使用它本身自建的行为？当我写这篇文章的时候，看起来还没有办法解决这个问题，所以，我猜想愿意使用 **MENUCONTEXT** 的人会很少。

替换项目管理器快捷菜单

虽然有这些问题，我们还是来做个例子体验一下吧。在我的 **Foxtalk 2000** 年 **5** 月的文章“**Zip it, Zip it Good**”中，我提供了一个能够把一个项目中所有基于表的文件（例如 **VCX**、**SCX**、以及 **FRX** 文件）打包的工具，另一个工具则能将所有在一个项目中引用到的文件全部压缩到一个 **ZIP** 文件中去。在那篇文章中，这些工具是从一个由项目专用的 **Project hook** 显示的工具栏来调用的。现在我们事实把它们放到任何项目都能用的快捷菜单中试试。

MENUCONTEXT 有着与 **MENUHIT** 同样的潜在冲突问题，所以我们将采用同一种解决办法：采用一条“标准”记录，它简单的根据各个特定的快捷菜单来将任务委托给另一些记录。事实上，这条标准记录的代码与 **MENUHIT** 标准记录的代码是一样的（运行 **StandardMenuContext.PRG** 来建立 **MENUCONTEXT** 记录）。不同的是，在 **ABBREV** 字段中放入的不是菜单项的标题而是 **MenuID(**你也许会想要将菜单的目标——例如“命令窗口”——放入 **EXPANDED** 字段以标明这条记录是用于什么目的的**)**。

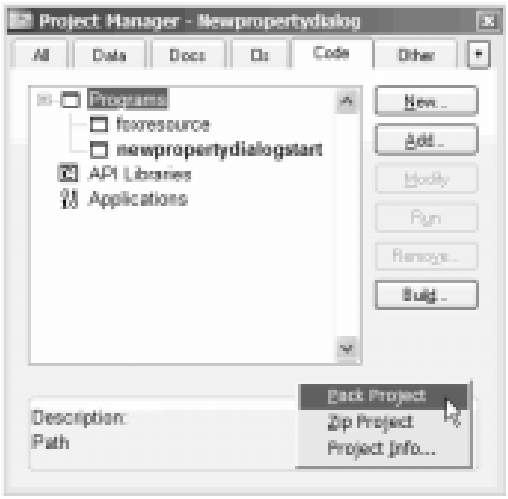
在建立 **MENUCONTEXT** 记录以后，我建立了一条 **TYPE = "M"**、**ABBREV = "24447"**（项目管

理器快捷菜单的 ID）的记录，在 DATA 字段中放入的是以下代码：

```
lparameters toParameter
local loMenu
loMenu = newobject('_ShortcutMenu', home() + 'ffc\_menu.vcx')
loMenu.AddMenuBar('\<Pack Project', 'do PACKPROJ with _vfp.ActiveProject.Name')
loMenu.AddMenuBar('\<Zip Project',  'do ZIPPROJ with _vfp.ActiveProject.Name')
loMenu.AddMenuBar('Project \<Info...', 'keyboard "{CTRL+J}" plain')
loMenu.ShowMenu()
```

这段代码使用了 FFC（FoxPro 基础类库)中的 _ShortcutMenu 类来提供快捷菜单。我在 Foxtalk 1999 年 2 月我的文章《A Last Look at the FFC》中讨论了这个类。运行 ProjectMenu.PRG 来将这条记录建立在 IntelliSense 表中。

当你在项目管理器中单击鼠标右键的时候，新的自定义菜单会显示三个菜单条（见图 2）：Pack Project，它会用当前项目的名称作为参数调用 PACKPROJ.PRG；Zip Project，它会用当前项目的名称作为参数调用 ZIPPROJ.PRG；还有 Project Infor，它会使用 KEYBOARD 命令来从项目菜单中调用项目信息对话框（后者由于在系统菜单中有这样的功能，因此可以这样调用，以减少自己的工作量）。



总结：

新的 MENUHIT 和 MENUCONTEXT 功能让我们可以前所未有紧密得挂钩到 VFP IDE 中去。除了这篇文章中我用到过的以外，我至少还能够替换“编辑属性/方法”对话框和文件菜单中的“新建”、“导入”、“导出”功能。我相信你也会乐意看到不同的 VFP 9 的 IDE 功能的实现。如果你有什么利用这个很酷的新功能的好电子，请一定告诉我。

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，微软 VFP 团队成员都会向你展示一些有趣的 VFP 技术。这个月的 tips 包括 通过 OLEDB 驱动运行 PRG 程序 和 利用智能感知技术来处理控件的注册。

通过 OLEDB 驱动运行 PRG 程序

在 VFP8 或 VFP9 中，通过给连接字符串指定一个 PRG 文件路径，VFP OLEDB 驱动便可以运行这个 PRG，而且可以用 PRG 文件名作为过程来运行（不用加上.PRG 扩展名）。这样就可以在本机运行 OLEDB 驱动触及的程序并且可以被任何支持 OLEDB 的应用程序调用。注意：被调用的 PRG 不能包含帮助文件中指定的“OLEDB 驱动不支持的 VFP 命令”。

这是一个简单的例子，表明了你可以通过 OLEDB 驱动运行程序并返回一个 ADO 记录集：

```
* from Aleksey Tsingauz
LOCAL oConn as ADODB.Connection
TEXT TO cPRG NOSHOW
RETURN "Here is the result"
ENDTEXT
STRTOFILE(cPRG,"c:\testPRG.prg")
oConn=CREATEOBJECT("ADODB.Connection")
oConn.Open("Provider=VFPOLEDB.1;Data Source=c:\")
oRS=oConn.Execute("testPRG",,4)
?oRS.Fields(0).Value && prints "Here is the result"
DELETE FILE c:\testPRG.prg
RETURN
```

利用智能感知技术来处理控件的注册

你每次在 VFP 中新建一个 COM 组件（或重建一次并重新生成组件 IDs），都要重新在注册表里注册一次。为了控制注册表的
膨胀，一个好习惯就是在做这些事的时候将其从注册表中移去。下面的 IntelliSens 脚本将帮你做这件事。

为了加入这个 IntelliSense 脚本，你必须指定一个字符串的键入来访问这个脚本，改掉脚本顶部代

码中的 lcKeys 变量，然后运行代码。一旦脚本被加入 IntelliSense，键入你指定的字符串并在对话框中选择一个 EXE 或 DLL，选中的对象将被反注册！

```
* from Trevor Hancock
LOCAL lcISense AS STRING, ;
  lcMyEndText AS STRING, ;
  lcKeys AS STRING
lcMyEndText = SYS(2015)
* Change this to preferred IS access string
lcKeys = 'unreg'
TEXT TO lcISense NOSHOW TEXTMERGE
LPARAMETERS oFoxcode
IF !INLIST(oFoxcode.Location,0)
  RETURN '<>'
ENDIF
oFoxcode.valuetype = 'V'
LOCAL lcFCCode AS STRING
TEXT TO lcFCCode NOSHOW
LOCAL lcFOXCOM AS STRING
lcFOXCOM = ;
  GETFILE('Fox COM Object:DLL,EXE', ;
  'COM Object:', ;
  'Select', 0, ;
  'Select Com DLL/EXE')
IF EMPTY(lcFOXCOM) OR ;
    !INLIST(LOWER(JUSTEXT(lcFOXCOM)), 'dll', 'exe')
  RETURN .F.
ENDIF
LOCAL llHadErr AS Boolean
llHadErr = .F.
IF LOWER(JUSTEXT(lcFOXCOM)) == 'dll'
  DECLARE INTEGER DllUnregisterServer IN &lcFOXCOM
  TRY
    DllUnregisterServer()
  CATCH TO loErr
    llHadErr = .T.
  ENDTRY
  CLEAR DLLS DllUnregisterServer
ELSE
  LOCAL lcUnregCmd
  lcUnregCmd = [RUN /N "] + lcFOXCOM + [" /unregserver]
  TRY
    &lcUnregCmd
  CATCH TO loErr
```

```
    llHadErr = .T.
  ENDTRY
ENDIF
IF llHadErr
  WAIT WINDOW 'Error unregistering ' + ;
    JUSTFNAME(lcFOXCOM) + ": " + loErr.MESSAGE ;
  TIMEOUT 2
ELSE
  WAIT WINDOW 'Unreg command for ' + ;
  JUSTFNAME(lcFOXCOM) + ' completed.'  ;
  TIMEOUT 2
ENDIF
<>
EXECSCRIPT(lcFCCode)
RETURN '~'
ENDTEXT
USE (_FOXCODE) AGAIN SHARED IN 0 ALIAS FC
INSERT INTO FC VALUES('U', lcKeys, '', '{}', '', ;
  STRTRAN(lcISense, lcMyEndText,'ENDTEXT'), ;
  'U', .T., DATETIME(), 'FoxTips', '', '')
USE IN SELECT('FC')
```

在 VFP9 中从缓冲区取出 SELECT 结果

当你对一个缓冲区 CURSOR 执行 SELECT 并返回结果到缓冲区的时候，往往会得到不正确的数据。VFP9 中 SQL SELECT 命令新的[WITH BUFFERING = .T.|.F.|Expression]子句正好是你所需的。

拆分工具（第三部分）

原著: Lauren Clarke 、Randy Pearson

翻译: CY

这是用VFP处理文本的系列文章的第三部分，先前的文章已经重现了VFP纯文本处理性能，并给出了如何对规则表达式进行性能补充——强大的模型描述语言。这个月 Lauren Clarke 和 Randy Pearson 将创建一个令人惊奇的函数，它可以用纯VFP表达式无缝地配合了规则表达式。这个函数，它是很有用的，将会作为下个月的拆分基类的基本方法。（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。）

如果你已经看过这个系列的前两篇文章，现在该是兴奋的时候了。这个月，我们将通过混合VFP的某些可爱特性与规则表达式，以支撑起规则表达式的强大力量，产生一个小而强大的文本处理函数。警告：这篇文章并不适合于心脏不好的人，如果你对于规则表达式是个新手，或者对VFP很一般，你或许需要自行去回顾前面两篇文章。

上个月，我们提供了一个函数StrtranRegExp()，它是对VFP函数STRTRAN()的不完全扩展。扩展这个函数以支持STRTRAN()所有的六个参数，是作为留给读者的练习。

这个函数strtranx1()包含在本月的下载文件里，是个局部的解决方案。

```
FUNCTION strtranx1(tcSearched, tcSearchFor, ;
tcReplacement, tnStart, tnNumber, tnFlag )
```

在那个函数里它并没有处理所有的nFlag设置情况，但是它通过设置vbscript.regexp对象的ignorecase 属性，处理了重要的“大小写敏感查找”情况。

```
.ignorecase = IIF(VARTYPE(m.tnFlag)=[N], ;
m.tnFlag = 1,.F.)
```

通过检查strtranx1()的代码，你会注意到loMatches集合是通过调用vbscript.regexp的execute()方法返回的。

```
loMatches = loRE.execute( m.lcText )
```

LoMatches集合包含了所有匹配于给出的模型字符串（通过tcSearchFor传递），规则表达式引擎在文本(tcSearched)里进行查找。我们然后重复这个集合，根据tnStart 和 tnNumber 参数以控制要执行的开始位置和替换数量。

这样，我们可以完成loRE.Replace()方法所作的，但是我们可以控制每次替换过程的开始和结束。函数strtranx1()的第二个重要部分是下面段落：

```
*-- build the replacement string from the value
*-- 从值创建替换字符串
lcReplacement = loRE.REPLACE( loMatch.VALUE, ;
m.tcReplacement )
*-- stuff the replacement text into the source text
*-- 填充替换文本到原文本
lcText = STUFF( m.lcText, ;
loMatch.FirstIndex + m.lnShift, ;
m.loMatch.length, m.lcReplacement)
*-- keep track of the string position dynamics
*-- 动态记录字符串位置
lnShift = m.lnShift + LEN( m.lcReplacement ) - ;
m.loMatch.length
```

第一行并没有真正的完成在目标文本里的替换。更正确的是，它只对匹配值进行操作，并且创建了随后将要填充入目标文本里的替换字符串(lcReplacement)。这个版本允许规则表达式引擎生成基于规则表达式的替换(它可以包括捕捉的类似于$1和$2的子匹配参考）。稍后，我们自己将用VFP代码来完成这个工作以更进一步的控制这个过程。

下面两行插入替换字符串以代替匹配的字符串，并处理替换字符串可能与匹配字符串长度不相同的情况。这样，

我们可以一个接一个周而复始的找出所有的匹配字符串并加以替换。

这时所有我们所做的是vbRegExp.replace()函数为我们所做的，但是我们现在用参数tbStart和tnNumber来参数化这个过程。

投入使用

利用这个函数，我们可以做如下的事情：

```
*-- replace first and second lower-case "a"
*-- 替换第一个和第二个小写的[a]
? strtranx1([Abracadabra],[a],[*],1,2)
** prints Abr*c*dabra
** 打印 *Abr*c*dabra
*-- replace first and second "a" case-insensitive
*-- 替换第一个和第二个[a],忽略大小写
? strtranx1([Abracadabra],[a],[*],1,2,1)
** prints *br*cadabra
** 打印 *br*cadabra
```

并且，更有趣的是，我们可以利用规则表达式来定位替换目标。

```
*-- case-sensitive, prints Abracadabr*
*-- 大小写敏感，打印 Abracadabr*
? strtranx([Abracadabra],[^a|a$],[*],1,2,0)
*-- case-insensitive, prints *bracadabr*
*-- 大小写不敏感，打印 *bracadabr*
? strtranx([Abracadabra],[^a|a$],[*],1,2,1)
```

规则表达式 ^a|a$ 在处理代码里的意思是，“在行首或者行尾的字母’a’”。在第二个例子里，我们发出了“大小写不敏感”标识，于是我们也替换了在行首的大写字母’A’。

请稍候，还有更多！（或者至少会是如此）

在这一点上，我们有一个混合的STRTRAN()函数，可以让我们来指定查找和替换为普通文本或作为一个规则表达式。非常棒！我们会满意吗？当然不。这里是一个上月的示例，它转换数字为欧洲格式：

```
lcText = "The cost, is $123,345.75. "
*-- convert the commas
*-- 转换逗号
lcText = strtranx1( m.lcText, ;
"(\d{1,3})\,(\d{1,}) ","$1 $2" )
*-- convert the decimals
*-- 转换小数点
? strtranx1( m.lcText, "(\d{1,3})\.(\d{1,})", "$1,$2" )
** prints "The cost, is $123 345,75."
** 打印 “This cost, is $123 345,75.”
```

注意前述的示例里我们是如何捕捉匹配的部分（在标点符号两边的数字），并利用$1和$2构造替换的，参见第一个和第二个捕捉的子匹配。这是有意义的，因为它意味着替换可以取决于某些范围内的匹配字符串。没有这个能力，我们将不得不多次的查找不同的数字组合。可是，还有很多机会，更强大和可更多控制是有益的。如果我们可以利用VFP代码对替换作出判断，难道不好吗？考虑到这个函数的强大，它可以让我们调用VFP函数（传递捕捉的子匹配作为参数）以生成替换文本。让我们先停留在这里。或许你还没有认识到你所需要的，但是你已经做到了。

示例的期望列表

让我们来把一些示例放入到期望列表上，再看看我们是否可以找到一个方法，以通过混合规则表达式和VFP函数来进行处理。

1. 单步欧洲格式转换

再回头看看先前的欧洲格式示例，我们转换逗号和小数点的位置。我们所做的是两次非常相似的查找和替换的操作，但是用到两次对strtranx1()函数的调用才完成这个工作。第一次调用替换逗号为空格，第二次调用替

换句点为逗号（注意相关的处理顺序）。它可以完美的压缩这个记法为单次的函数调用，如下：

```
lcText = [The cost, is $123,345.75.]
? strtranx( lcText, ;
"(\d{1,3})([,\.])(\d{1,3})",;
"=$1+IIF($2=",",space(1),",")+$3" )
** prints "The cost, is $123 345,75."
** 打印 “The cost, is $123 345,75.”
```

注意到这里的查找模型包含一个捕捉模型[,\.]，它是通过替换模型里的点占位符“$2”来表示的。你可能已经明白我们所要做的。我们重复进行匹配，子匹配将会在对VFP表达式进行计算前插入到替换表达式。然后VFP表达式计算的结果将会用来替换当前匹配字符串。

```
=$1+IIF($2=",",space(1),",")+$3
```

你可以描述先前的表达式为如下：第一个子匹配（标点符号前的数字），如果标点符号是个逗号那么紧跟着一个空格（或者如果是个句点那就是紧跟一个逗号），紧跟着第三个子匹配（数字跟着标点符号）。

2. Web应用的周末装饰

假想你正在写一个Web应用，它在周末时要把任何自由格式文本被标记为黑体。如果我们使用VFP代码来作这样的替换，我们可以如下所示：

```
lcText = strtranx( ;
[6/12/2004 is a weekend, 6/17/2004 is not. ] ,;
[(\d{1,2}/\d{1,2}/\d{4})],;
[=IIF(INLIST(DOW(CTOD($1)),1,7),;
"<strong>"+$1+"</strong>",$1)])
```

在它调用我们的梦幻版的strtranx1()函数后，变量lcText就会包含：

6/12/2004 is a weekend, 6/17/2004 is not.

我们使用了规则表达式来查找日期，并且利用VFP函数根据此周内的某天来处理相应的替换值。

3.转换CamelCaps字为连接

[译注:从示例来看，似乎“CamelCaps”可以理解为“以大写字母打头单词的”]

这里是另一个兴奋的示例。假设你正在创建一个wiki类型Web网站（参见http://fox.wikis.com为例），并且想要改变CamelCaps字为连接。利用前面的第一次传递strtranx1()函数，创建连接可以变得很容易。

```
? strtranx1( [If text is CamelCaps, link it.] ,;
"(\b\w*[A-Z]+[a-z]+[A-Z]+\w+\b)",;
[<a href="someurl~topic=$1">$1</a>])
** prints: If text is
** <a href="someurl~topic=CamelCaps">CamelCaps</a>,
** link it.
** 打印: If text is <a href="someurl~topic=CamelCaps">CamelCaps</a>,  link it.
```

这个工作不需要你确定什么样的CamelCaps字要变为连接。可是，你或许需要返回取决于有多个规则的不同标记，比如：是否有主题存在？用户对此主题是否有权阅读？主题的最后编辑时间？主题是否定向到当前网站外的资源？所有的情况都会影响你所要为给定的CamelCapss字放置的连接类型。如果你可以插入VFP代码到替换模型，你将可以通过简单的方法调用以处理所有支持的VFP代码的任意类型，当传递一个主题时将返回相应的标记。

```
? strtranx(lcTopicText,
"(\b\w*[A-Z]+[a-z]+[A-Z]+\w+\b)",;
"=oWikiProcess.linkBuilder($1) ")
```

4.自动家庭作业处理

假设你的小孩想要你来帮他做家庭作业。用一个（目前虚构的）strtranx1() 函数调用来做难道不好吗？

```
TEXT TO lcText NOSHOW FLAGS 1
A. 4+7=
B. 7-2=
C. 2^2=
D. 7*(3+1)=
ENDTEXT
? strtranx( lcText,"([A-Z]\. )(.+)=", ;
[=$1+$2+"="+TRANSFORM(EVALUATE($2))] )
```

这将打印为：

```
A. 4+7=11
B. 7-2=5
C. 2^2=4
D. 7*(3+1)=28
```

这最后一个示例，有利于使人理解“大主意”。我们谈论着取决于匹配值（或子匹配）的动态替换。Perl 用户有Perl的 “S”函数的这样能力。多个文本编辑器，比如WIM和Emacs，允许这个类别的替换。现在就是我们要在VFP里实现的时候了。

在这一点上，我们所显示的四个示例仅仅只是虚构的想像。这篇文章的剩余部分将会扩展strtranx1()函数以处理这些动态规则表达式。

创建动态置换

咋看来，扩展strtranx1()函数以提供基于VFP的替换表达式，可能看起来很难。Vbscript.regexp类丝毫不会识别VFP，并且我们确实没有任何办法通过回调或其他交互的方法来训练它。可是，vbscript.regexp对象模型允许我们在给定的匹配内编址每一个单独的子匹配：

```
loRegExp = CREATEOBJECT([vbscript.regexp])
```

```
loRegExp.pattern = " (\d{1,3})([,\.])(\d{1,2})"
loRegExp.global = .t.
loMatches=loRegExp.execute([The cost is $123,345.75.])
FOR EACH loMatch IN loMatches
FOR EACH lcSubmatch IN loMatch.subMatches
? lcSubmatch
ENDFOR
ENDFOR
```

处理代码将会为第一个匹配产生出罗列在表1内的子匹配（注意在被处理的字符串里的标记为红色的位置）。

表1:包含在第一个匹配中的子匹配

输出	字符串中的位置
123	$**123**,345.75
,	$123**,**345.75
34	$123,**34**5.75

第二个匹配所包含的子匹配如表2所示。

表2:包含在于第二个匹配里的子匹配

输出	字符串中的位置
5	$123,34**5**.75
.	$123,345**.**75
75	$123,345.**75**

我们可以通过产生几个非期望的捕捉来改变这个行为（已在上月的文章里讨论），但是现在这可以满足了。让我们再来看看熟悉的替换字符串：

```
=$1+IIF($2=",",space(1),",")+$3
```

要把这变成有效的VFP代码，我们需要填充相应的子匹配到$1、$2、$3占位符，然后EVALUVATE()计算整个替换字符串。下面是从我们新版的函数里摘录的关键部分（strtranx2()在下载文件里）：

1.通过调用vbscript.regexp::Execute()生成匹配集合：

```
loMatches = loRE.execute( m.lcText )
```

2. 重复匹配，并对每一个匹配，重复子匹配：

```
FOR lnK = m.lnStart TO m.lnNumber
loMatch = loMatches.ITEM(m.lnK-1) && zero based
lcCommand = m.lcReplacement
FOR lnI= 1 TO loMatch.submatches.COUNT
```

3.填充子匹配到给定的替换表达式：

```
lcSubMatch = loMatch.submatches(m.lnI-1) && zero based
lcCommand = STRTRAN(m.lcCommand, "$" + ;
ALLTRIM( STR( m.lnI ) ) , m.lcSubMatch)
```

4. EVALUATE()计算替换表达式以产生VFP代码：

```
lcReplacementText = EVALUATE( m.lcCommand )
```

5. 填充结果字符串到目标文本以替换匹配：

```
lcText = STUFF( m.lcText, ;
loMatch.FirstIndex + m.lnShift, ;
m.loMatch.LENGTH, m.lcReplacementText)
```

6. 返回匹配集合里的下一匹配的第二步。

strtranx2()函数可以（几乎是）可以解决我们先前的“期望列表”示例。可是，在这里还有一些基本问题。注意那个循环，我们创建了命令并填充子匹配到替换字符串中的占位符：

```
*-- build the replacement string
FOR lnI= 1 TO loMatch.submatches.COUNT
lcSubMatch=loMatch.submatches(m.lnI-1) && zero based
lcCommand=STRTRAN(m.lcCommand, "$" + ;
ALLTRIM( STR( m.lnI ) ) , m.lcSubMatch)
ENDFOR
```

在货币转换示例（$123,345.75）中，占位符替代将会在这个表达式里产生，它却不能被VFP正确计算：

```
123+IIF(,='',',space(1),',')+3
```

很明显，在计算时这将会导致错误，于是我们必须要把子匹配的占位符包括在引号内，以确保字符串可以被创建。为使用strtranx2()来完成这个工作，我们将不得不设立一个如下的表达式：

```
"$1"+IIF("$2"='',',space(1),',')+"$3"
```

可是，这并不满意，因为它意味着我们不能在子匹配里有引号字符，没有额外的防护性的代码就无法在创建命令前先处理引号字符。同样，如果计算表达式超过255个字符，EVALUATE()函数将会失败。并且，如果回车符换行出现在表达式里，EVALUATE()调用也会出错。

解决问题

所有的这些问题，都是严重的，但还不是我们最糟糕的问题。当用户提交字符串进行处理时，由于插入恶意代码的可能，Strtranx2()函数会给应用程序打开一个巨大的安全漏洞。如果不给予关注，那么它可能被用户通过精巧的文本以运行任意的代码，它将通过执行EVALUATE()来返回一个恶意的表达式。你可以在下载文件的示例里见到这个攻击类别的示例。

在我们函数的最终版本里，现在简单的调用strtranx()函数，我们解决了这些所有问题（参见下载文件里的代

码）。我们的策略非常简单。代替插入子匹配为文字，我们把它剔除到一个集合内：

```
loCol = CREATEOBJECT([collection])
* several lines omitted here to save space
* 这里省略多行以节省空间
lcKey = ALLTRIM(TRANSFORM(m.lnK)+[_]+TRANSFORM(m.lnI))
loCol.ADD( m.lcSubMatch, m.lcKey )
```

然后我们插入一个相应的对集合的item()方法调用，以检索子匹配文本：

```
lcSubMatch = [loCol.item(']+m.lcKey+[')]
lcCommand = STRTRAN( m.lcCommand, "$" + ;
ALLTRIM( STR( m.lnI ) ) , m.lcSubMatch)
```

这样，EVALUATE()函数就再也不会遇到任何可能有害的匹配文本。我们也排除了在子匹配外面加引号的必要。再次，来看看那个欧洲格式示例表达式：

```
$1+IIF($2=",",space(1),",")+$3
```

对于第一个匹配，在最终版本的strtranx()函数里它将会变成为一个有效的VFP表达式，然后它也可以被计算：

```
loCol.item('1_1')+ ;
IIF(loCol.item('1_2')=",",space(1),",")+ ;
loCol.item('1_3')
```

通过对集合关键字采用一个[匹配数]_[子匹配数]的转换，我们可以避免重装集合的代价，或者从它里面移去我们重复匹配的元素。剔除子匹配到一个集合里，可以避免定界问题，大大地减轻插入的风险，并且，对于大多数情况下，排除了行长度的违规。错误仍然会出现，但是他们主要应该是动态替换字符串的问题。那些错误通常可以被程序员修复。这个strtranx()函数可以通过TRY/CATCH结构来捕捉并报告这些错误。同样注意到我们并没有直接计算任何的匹配文本，开发人员仍然需要在对替换表达式格式化时不得不加以关注。

再访家庭作业

在前面的家庭作业示例时，我们明确地把EVALUATE()函数放入到替换模型里。很明显，我们通过放置用户提供的内容到集合元素，我们小心堵塞的漏洞又会被重新打开。绕过这个问题的一个方法是，把家庭作业表达式分解为操作符和数字，然后在EVALUATE()计算前重建。这将会剥除任何可能的恶意文本：

```
*-- break the math expression down to
*-- number, operator, number components
*-- 分解数学表达式为数字、操作符、数字部件
? strtranx( lcText,;
"([A-Z]\. )(\d+)(.+)(\d+)=",;
[=$1+$2+$3+$4+"="+TRANSFORM(EVALUATE($2+$3+$4))] )
```

这个方法限制了可以被计算的表达式类别。比如，7*(3+1)= ，如果不做额外的工作，将无法被更多限制性的模型所识别。另一个选择就是传递整个表达式给一个设计用来更安全拆分数学表达式的函数或方法。

当然，最好的选择是让你的孩子他自己来做家庭作业。

注意，当替换模型它自身通过调用EVALUATE()、EXECSCRIPT()等等来计算子匹配时，还需要额外的小心。通常，这并不需要关注。事实上，我们在实际中从未有过这样的机会，但是，为这个强大的工具指出建议性的安全指导方针还是很重要的。

更重要的是注意到恶意代码的插入问题，虽然是非常重要的，但还并不是这里最主要的。在strtranx( )函数里，我们现在已经有了一个非常强大的函数，它支持用以查找和替换的规则表达式，并且可以使用基于VFP的表达式按照一一对应的原则以判断替换文本。由于代码插入是个问题，strtranx()函数通过在EVALUATE()计算替换表达式前剔除子匹配字符串以保护自己。开发人员要利用这个强大的工具仍然需要特别关注那些包含运行时执行命令的精巧的替换表达式。

结论

在这个系列的前三个部分里我们已经“装备”起来了。现在在工作台上我们已经有数个强大的工具，但是我们还没有统一的结构。下个月，我们将开始整合这些工具（特别是strtranx()函数）为一个类群，它将形成一个

对所有方式的拆分任务可重用的框架。

在VFP9.0中通过FFC类库使用GDI+

第二部分

原著：Walter Nicholls

翻译：YAUR

在八月份的早些时候，微软已经预先发布了一些伴随着VFP9 Beta版的额外的文件，这其中就包括了一个强劲的为GDI+图形所服务的功能类库(检查一下VFP官方网站http://msdn.com/vfoxpro 上的下载信息)。上个月， Walter Nicholls为大家引入并解释了GDI+ FFC类库，在这第二篇(此系列共包含四篇)文章中，它将说明如何使用GDI+类库来扩展VFP9的报表系统。

在上月发表的这个系列的第一篇文章中，我为您说明了FFC _GDIPLUS类库中的主要类以及在一个VFP的表单上绘制出一个饼状图的实战方法，现在我将来说明如何用同样的代码在VFP的报表中加入一个饼图。我将改进这个饼图实例，让它包含一个图例和被扯出来的饼图块。

回来了—一个饼图绘制类

让我们从找出上个月的饼图代码并将它转换为几个单独的泪，就像图1那样的。

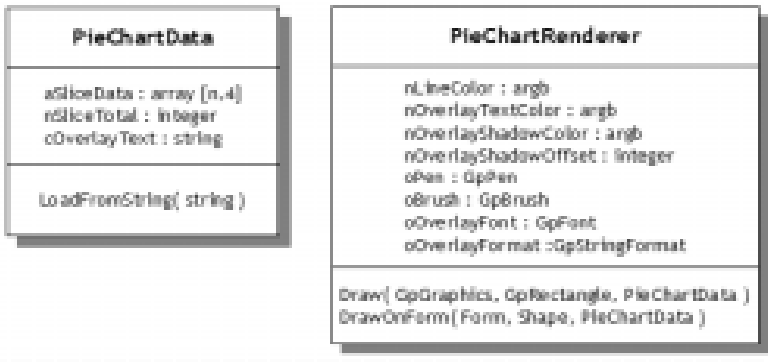


图1

如果你读过上篇文章，你应该能搞定这些，为了节省篇幅，我就不把所有的修改过的代码贴出来了，不过你可以在附带的下载文件中看到完整的源代码。**PieChartData**类是用来处理绘制饼图所需要的数据的，我会分离出与上个月所作的表单中相同的代码并且为这个数组增加两列，如表1

表1. PieChartData类的aSliceData数组的列

Column#	#define symbol	Type	Description
1	SLICE_COL_LABEL	String	标签/标题
2	SLICE_COL_VALUE	Numeric	数值
3	SLICE_COL_COLOUR	ARGB color	图例的颜色填充值
4	SLICE_COL_FLAGS	Integer	标记(0x0001 =抽出饼图区域)

PieChartRenderer类是基于上个月的表单中的Paint方法的，但是也有它独到的地方，DrawOnForm()方法是用来创建GpGraphics对象并且来设置一些属性(例如阴影的便宜量)以使绘制在标单上显得非常合适。Draw()方法则不依赖于绘图界面而且是不可改变的—无论是在表单、报表上还是其它任何地方。

使用这些类，相应的上个月的示例表单中的Paint()方法就变成这样：

```
This.oPieChart.DrawOnForm( Thisform ;
Thisform.shpPlaceHolder, This.oPieData )
```

这个在下载文件的gpintro2_ex1.scx中，运行后的结果如图2所示。

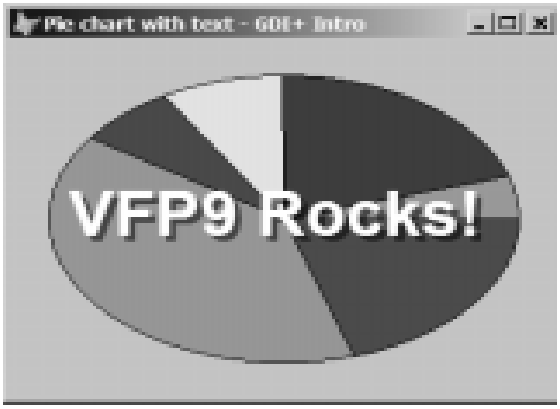


图2

跃然纸上

有了上边的路子，我们就可以更好的在报表上绘制图形，不过在标单上绘图和在报表上绘图还是有一些不同。

在表单上绘图

- 绘图是发生在 Form.Paint()事件中
- 坐标距离是用像素来计量的
- 屏幕分辨率通常是72-120dpi
- 必须首先初始化GDI+
- 通过表单的HWND值来创建GDI+图像对象

在报表上绘图

- 绘图是发生在 ReportListener.Render()事件中

•坐标距离是用一英寸的1/960(0.265毫米)来计量的

•打印分辨率可以达到600dpi甚至更高，预览(100%的时候)则是和屏幕分辨率是一样的

•GDI+自动为你初始化

•GDI+图像对象是由ReportListener基类来创建的

*在报表上绘制饼图

```
function DrawOnReport( ;
tnGdiplusGraphics as integer ; && GDI+ handle
, tnX, tnY, tnWidth, tnHeight ; && bounding box
, toPieData as PieChartData ; && Pie chart data
)
* Create a rectangle object storing the bounding
* box of our drawing
local oBounds as GpRectangle of ffc/_gdiplus.vcx
oBounds = newobject( ;
'GpRectangle','ffc/_gdiplus.vcx','' ;
, m.tnX, m.tnY ;
, m.tnWidth, m.tnHeight )
* Associate FFC object with the graphics handle
* (Don't create one - it already exists!)
local oGr as GpGraphics of ffc/_gdiplus.vcx
oGr = newobject('GpGraphics','ffc/_gdiplus.vcx')
oGr.SetHandle( m.tnGdiplusGraphics )
* Set the drop-shadow to about 0.05in (1.3mm)
This.nOverlayShadowOffset = 48 && == 0.05in/960
return This.Draw( m.oGr, m.oBounds, @toPieData )
endfunc && DrawOnReport
```

使用这种方法似乎有点难度，对于第一次尝试，还是让我们先从简单的东东着手吧：创建一个报表，在它的细节带区绘制一个简单的形状(看图3，gpintro2_rpt1.frx文件)。现在用如下列出这种个试创建一个简单的ReportListener。

```
define class PieChartListener as ReportListener
oPieData = null
oPieChart = null
function Init
* Add the pie chart data object
This.oPieData = newobject( "PieChartData", ;
"gpintro2_pie2.prg" )
This.oPieChart = newobject( "PieChartRenderer", ;
"gpintro2_pie2.prg" )
* Create some data to fill the chart with
local lcPieInfo
text to m.lcPieInfo noshow flags 1
Auckland, 33, 0xFFFF0000
```

```
Hamilton, 71, 0xFF00FF00
Dunedin, 12, 0xFF3333FF
Christchurch, 21, 0xFFFFFF00
Wellington, 40, 0xFF990099
Chathams, 6, 0xFF999999
endtext
This.oPieData.LoadFromString( m.lcPieInfo )
This.oPieData.cOverlayText = "VFP9 Rocks!"
endfunc && Init
function Render
lparameters nFRXRecno ;
, nLeft, nTop, nWidth, nHeight ;
, nObjectContinuationType ;
, cContentsToBeRendered, GDIPlusImage
This.oPieChart.DrawOnReport( ;
This.GDIPlusGraphics ;
, m.nLeft, m.nTop, m.nWidth, m.nHeight ;
, This.oPieData )
endfunc && Render
enddefine && PieChartListener
```

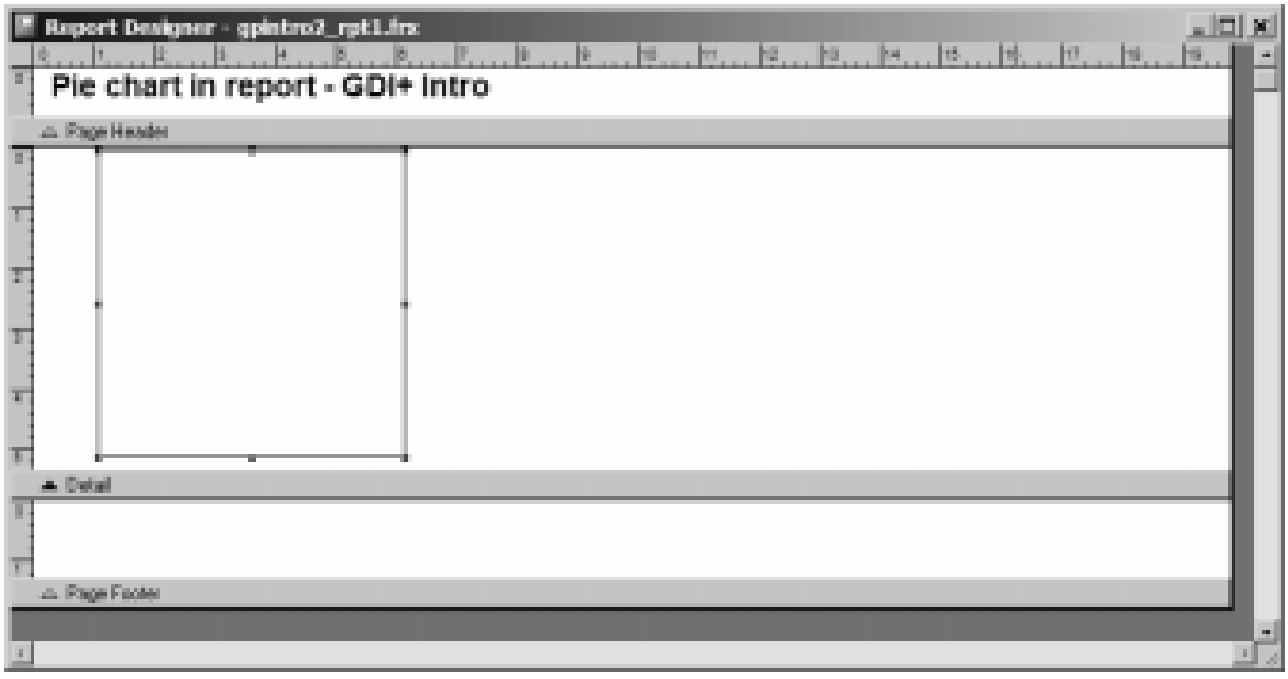


图3

现在运行它，或许可以使用如下这段程序来运行。

```
local oListener as PieChartListener
oListener = newobject('PieChartListener')
```

```
oListener.ListenerType = 1 && preview
create cursor csrReport ( dummy c(10) )
append blank
report form gpintro2_rpt1 object m.oListener
```

小贴士：你可以考虑ReportListener用包含TO PRINT 或 TO PREVIEW的方式来输出。一个将ListenerType 属性设置为0的ReportListener对象和上边的输出方式的效果是相同的，上例中使用的ListenerType = 1就等同于REPORT ... TO PREVIEW，还有其它一些输出方式，包括输出为XML、HTML，而且你也可以加入自己的。上例的运行效果如图4

很明显，我们希望如此简单的代码能够在报表中的任何对象中绘图，而不仅仅是形状(shape)对象。

不过，要调整这些也还比较容易，我们需要一些方法来决定是否当前的图形将被描绘成饼图，如果是这样的话，我们可以自己绘制或者执行NODEFAULT来阻止ReportListener基类的描绘动作。如果我们假定它不是一个饼图，那在我们专门的ReportListener中的Render方法会什么也不干，允许这个对象正常的去描绘。

我们似乎走了一些捷径，然后这是却因为我们通过迅速的解决一些问题而获得了一个很好的解决方案。

重要的ReportListener任务

当我们建立一个类似于此来执行插件对象的这样一个ReportListener对象的时候，有几个任务我们需要完成：

•确定哪个报表对象将被描绘、哪一个vfp的ReportListener基类(或者另一个ReportListener)将被操作。

•分析并记住任何存储与FRX.中的设置信息。

•获得生成饼图的数据

•掌握这些在我们所调用的代码中能够被共享的信息。

•与其它的ReportListener相互结合。

有很多方法可以判断哪一个报表对象将被处理，而且它是依赖于ReportListener 服务的目的。 例如，一个将其中的负数用红色来显示的ReportListener可能会在计算结果为负数的对象中查找这些类型为8(字段/表达式)的对象。在这种情形下，我们还是会继续干活而不去考虑关于这些对象类型更多的细节。

为了判定一个饼图对象，我们将是用VFP9的一个新的特性，就是在对象的" Execute When"表达式中加入"PIECHART"字符串，这个在新的报表设计器中很容易编辑处理(除非你通过_reportbuilder变量禁止了默认的报表生成器程序)。你将会看到"Execute When"是在字段属性对话框的Other选项卡中的"Run-time Extensions"项里(图5)。

" Run-time Extensions"也是用来处理其它报表需要的数据的地方， 它甚至可以用来扩展报表设计器从而使之包含用户自己的界面和设置信息，不过这个难度颇大。

现在，我们将用字段表达式对象替换形状(Shape)对象来在报表上绘制饼图，我们会把饼图自身的数据存储在一个备注字段中(csrReport.piedata)以便在字段表达式中引用。我这样建议是因为它比较容易而且可以简化代码—vfp计算表达式并且通过它来响应ReportListener事件。我们已经有了LoadFromString()方法，在实际过程中，数据极有可能不是最初的格式，这这种情况下你或许要将一个键值存储在子表当中。

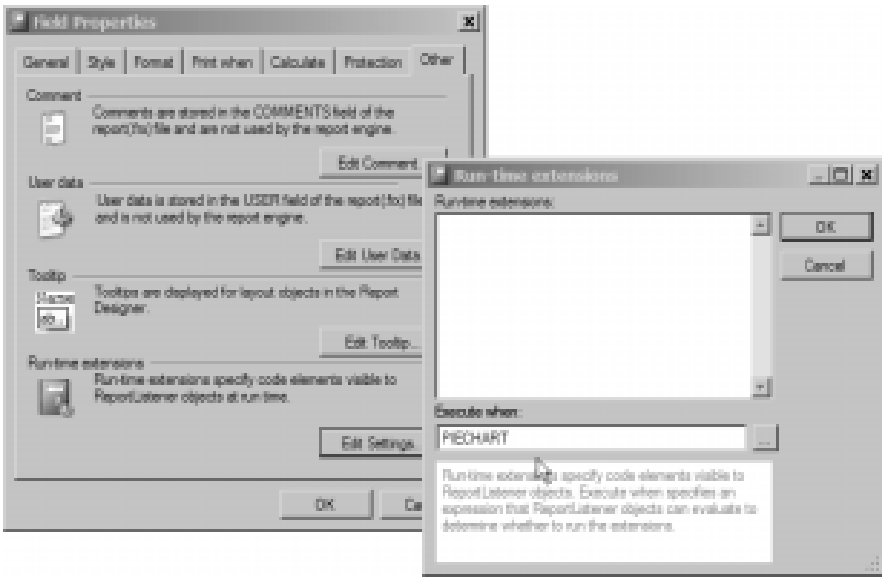


图5

在报表中(gpintro2_rpt2.frx),注意报表对象已经自动拉伸而且超出范围了。这点很重要：就是说我们可以确信当数据是文本的时候VFP不会去计算它(从而会改变字段尺寸)。为了获得一个自动尺寸的图表，你必须使用一个形状(Shape)对象和ReportListener的AdjustObjectSize()方法。

我给这个ReportListener增加了一个数组属性来存储FRX设置信息(一个饼图的情况)，加入一个PieChartData对象的实例。你也可以使用临时表或者其它的数据结构来替换这个，只要这些设置信息是依托于FRX的记录数,而且必须保证能够唯一识别这些对象。两选其一，你可能用其它方法来标识对象，不过，我觉得Recno('frx')这种方法是既简单又安全的。

APieObjects数组是在ReportListener的BeginReport()方法中定义的(部分代码如下)：

code shown):

```
This.setFRXDataSession()
lnSaveArea= select()
select frx
This.nPieObjects = 0
scan for not empty(frx.style)
xmltocursor(frx.style,'xmltemp')
if atc('PIECHART',xmltemp.ExecWhen)>0
* It's a pie object
This.nPieObjects = This.nPieObjects + 1
dimension This.aPieObjects[This.nPieObjects, 2]
This.aPieObjects[This.nPieObjects,1] = ;
recno('frx')
This.aPieObjects[This.nPieObjects,2] = ;
newobject( "PieChartData" )
endif
endscan
use in select('xmltemp')
```

在之后的任何时候我们都可以通过**ASCAN()**来判断是否一个对象是饼图并且取得索引存入数组中。现在，当报表引擎调用**EvaluateContents**函数的时候，我们可以定位这个**PieChartData**对象。

```
function EvaluateContents(nFRXRecno,oObjProperties)
local lnIndex
lnIndex = This.FindPieChartObject(m.nFRXRecno)
if m.lnIndex > 0 && pie chart
* Gather pie chart data
This.aPieobjects[m.lnIndex,2].LoadFromString( ;
oObjProperties.Text )
return .F. && we've taken over processing
else && normal report object
return dodefault()
endif
endfunc && EvaluateContents
```

然后，当最后要描绘饼图的时候，我们就可以使用存储的数据了。

```
function Render
lparameters nFRXRecno ;
, nLeft, nTop, nWidth, nHeight ;
, nObjectContinuationType ;
, cContentsToBeRendered, GDIPlusImage
local lnIndex
lnIndex = This.FindPieChartObject(m.nFRXRecno)
if m.lnIndex > 0
* Pie chart - use the data we got previously
This.oPieChart.DrawOnReport( ;
```

```
This.GDIPlusGraphics ;
, m.nLeft, m.nTop, m.nWidth, m.nHeight ;
, This.aPieobjects[m.lnIndex,2] )
nodefault && we've taken over processing
endif
endfunc && Render
```

将这些汇至一处，并且提供适当的数据，我们就能得到图6，我已经在gpintro2_ex3.prg中提供了完整的代码。要留心的是我已经去掉了其中的概括性文字，因为它的使命已经完成了。

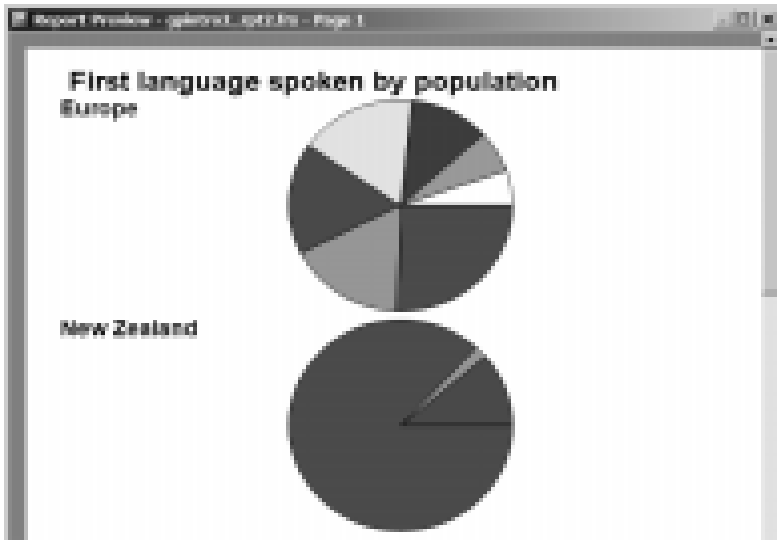


图6

加入图例

到目前为止，圆形分格统计图表主要是美观，事实上不是很有用，因为它没有告诉你每一块表示的是什么！为了弥补这个问题,我们应该在上面加个图例。

在这里我不写出所有的代码（它在下载文件 gpintro2_pie2.prg中用到,及一个程序用到它:gpintro2_ex4.prg). 我正在将报告域的矩形的范围分成两部分。圆形分格图标将继续在左边，而图例将在右边，当每个圆形块画好后，我也将在右边画个同样的图例。图7展示了不同的部分如何对应。

这里的代码有几个我没有说明怎样修改的问题，很难用代码将字体编写成10pt arial，但这是不难处理的：你即可以在报表设计器中设置字体也可以添加我前面所提起的设置数据。

比较严重的是，如果这里有很多块以至于图例变得比圆形本身还高，图标将开始溢出盒子的范围并且开始覆盖报告的其它部分。对于这个有多种方法，包括减小字体的大小，使用更多的圆柱，限制被加页眉

的数据项目的数字，或切换到一个形状(Shape)对象然后执行AdjustObjectSize()方法来拉伸图表。

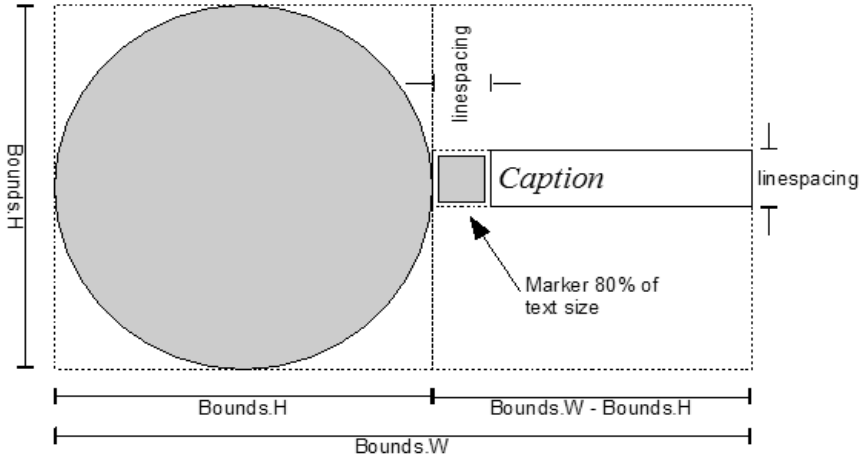


图7

撕出图块

现在让我门关注图表的“其它”块，将它拉出圆形的中心，我们将用一个叫做“变形”的相当聪明的办法来做。

如果一片要被扯出，我们保存GpGraphics object 的当前的状态并且移动同等空间覆盖：

```
if m.lCallOut
* Move coordinate space over to pull the slice out
toGraphics.Save( @nSavedGraphicsState )
toGraphics.TranslateTransform( ;
  CALLOUT_SLICE_OUTDENT * ;
cos(dtor((aAngles[m.iSlice]+aAngles[m.iSlice+1])/2)) ;
  , CALLOUT_SLICE_OUTDENT * ;
sin(dtor((aAngles[m.iSlice]+aAngles[m.iSlice+1])/2)) ;
  )
endif
```

然后，我们画的圆形块就好像没有改变

```
toGraphics.FillPie(This.oBrush, m.oPieBounds ;
  , aAngles[m.iSlice] ;
  , aAngles[m.iSlice+1] - aAngles[m.iSlice])
toGraphics.DrawPie(This.oPen, m.oPieBounds ;
  , aAngles[m.iSlice] ;
  , aAngles[m.iSlice+1] – aAngles[m.iSlice])
```

最后，我们将同等空间移回：

```
if m.lCallOut
  * Restore graphics state
```

```
    toGraphics.Restore( m.nSavedGraphicsState )
  endif
```

最后的代码在gpintro2_ex5.prg中可以下载，你可以在图8中看到其效果。

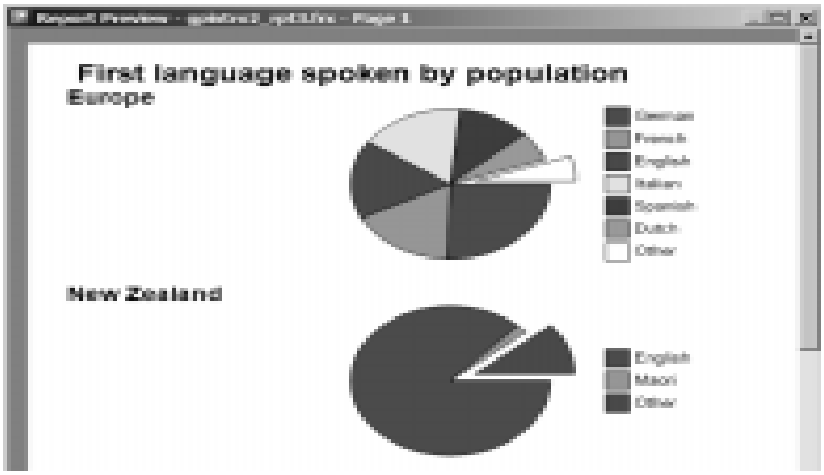


图8

变形是非常有用的。象这样的移动，你可以旋转和改变角度，并且你甚至可以把变形串连到一起。（在文章之初的水印加密展示的就是通过角度及转化完成的）

我没有隐秘的东东

可能我要尽力避免的问题就是处理可伸缩的对象和跨带区或跨页的对象。不是相应的我们这个系列中所关注的GDI+，可伸缩对象倒不是很困难，可是它们仅限于Shape对象(这个限制存在于Beta版中，可能正式版中会有所改变)。

这就意味着你执行AdjustObjectSize()方法的时候要替换EvaluateContents()函数，你也不得不使用不同的技术来获得饼图数据，产生跨页或跨带区问题对于图表来说不是一件好事情，尽管其它的GDI+的使用者不得不去处理它。

即将到来的下个月

图像和位图

----结束

VFP 联姻 Flash(第一部分)

原著:Yasur

用 Flash 可以做出非常漂亮的界面和动画效果，而 vfp 在这方面的确是先天不足，将两者的优点结合起来肯定是一件令人激动的事情，用美妙的 Flash 界面、互动功能结合 Vfp 的强大的数据处理及自身的数据库功能将会大大提高应用程序的感官、功能和竞争力。我按照这个思路，试着做了几个例子，结果非常满意。我想这应该是一件很有意义的事情，无限畅想。。。

当然，这其中包含两方面的工作，首先就是 Flash 可以控制或指挥 VFP，这样才可以让 Flash 中的界面元素发挥作用。另一方面，VFP 程序根据 Flash 的指挥在完成数据处理等工作后必须将数据或指令返回 Flash，这样才可以达到完全的互动。所以我们的工作从两个方面来开展：[1]用 Flash 控制 Vfp [2] Flash 和 VFp 的相互控制---数据通信。

一．用 Flash 控制 Vfp

看个界面图先。。

图 1：打开表的操作

具体实现步骤：

下面就来说说具体的步骤：

[1] 制作 flash

在 flash 中制作 4 个按钮，分别为打开，最大化，提示，关闭。

代码分别为(都是在鼠标释放的时候触发)：

打开：

```
on(release)
{
Fscommand("open","Opentable");
}
```

最大化：

```
on(release)
{
   Fscommand("max","windowsmax");
}
```

提示：

```
on(release)
{
Fscommand("msg","msgbox");
}
```

关闭：

```
on(release)
{
Fscommand("exit","close");
}
```

以上代码中 fscommand()是 flash 中用来执行外部命令的方法。参数是用来标识所点击的按钮及其值。

这就做好了 flash 发布为 swf 文件，保存到你所知道的地方。

[2] 编写 vfp 程序

首先制作一个简单的表单。在表单中插入 flash 组件，注意：这个组件是 macromedia 的官方组件，在 flash5.0 之前为 swflash.ocx，之后为 flash.ocx 文件，如果安装了 flash 这个文件在 c:\windows\system32\macormed\flash\中。注册该组件(该组件在 VFP 的组件列表中的名称为 shock wave Flash)，然后插入到表单中。

将 Flash 对象插入到表单以后首先设置它的 movie 属性，这个属性是指定此 Flash 的文件位置。也可以在程序中动态指定，指定后在程序运行的时候默认自动播放此 Flash。

在表单中加入一个表格，名称为 ygrid1.然后在 flash 对象的 fscommand()方法中写入如下代码(一看这个方法传递过来的参数，你就应该明白怎么回事了)：

```
DO CASE
  CASE command="open"
     LOCAL FileName
     FileName=GETFILE("dbf")
     IF !EMPTY(FileName)
```

```
            USE (FileName)
            thisform.ygrid1.RecordSource=""
            thisform.ygrid1.RecordSource=ALIAS()
            thisform.Refresh
        ENDIF
    CASE command="max"
        thisform.WindowState= 2
    CASE command="msg"
        MESSAGEBOX("Flash+vfp 可能会让您的程序增色不少。有无限的拓展空间","YASUR")
    CASE command="exit"
        THISFORM.Release
ENDCASE
```

好了，执行表单，小功告成。。。

图 2 提示按钮的执行效果

通过上边的例子和 **Fscomand()**方法的参数形式就可以明白他的原理了。。

二．Flash 和 VFp 的相互控制---数据通信。

上节主要说明了 **Flash** 来控制 **vfp**，是单向的。这篇则是说明了他们之间的数据交互，这样才算真正的能在实际应用中大量应用。。。

看个界面先。。。

一看界面就明白了。

大致思路：Flash 发出打开命令---vfp 响应打开命令并打开表----将表的相关信息传入 Flash 这一节主要来说明后半程的实现。

主要步骤：

[1]在 flash 中加入几个动态文本域，将其分别命名(即设置变量名)。

[2]在 vfp 中使用 flash 对象的 setvariable(name,value)来将值传入对应的 Flash 对象中。。

Flash 会自动刷新该值。

部分代码：

```
this.setVariable("T_name",ALLTRIM(DBF(ALIAS()))) &&表名
this.setVariable("T_recount",ALLTRIM(STR(RECCOUNT()))) && 记录数
this.setVariable("T_fcount",ALLTRIM(STR(FCOUNT()))) && 字段数
this.setVariable("T_curecno",ALLTRIM(STR(recno()))) && 当前记录
```

完了，就这么简单。。。

也可以使用 getvariable()函数来获取 Flash 中对象的值，当然这些对象应该实现定义为变量。。

以下是引用 vfp 迷所提出的问题：

我准备用 Flash 来制作一个奶牛场，鼠标点击牛舍就可知道这个牛舍的相关资料（比如有多少牛，有几头已怀孕，总产奶多少.....）

这个就是我最理想中的奶牛场"实景"查询系统

（呵呵，国外已经有了，但不知是否用 Flash+数据库实现的）

顺便来谈一下 vfp 迷所提出的这个应用的解决方案。

当点击牛场中的某个牛以后，将参数传入 vfp 的查询方法中，将查询出的结果比如奶牛年龄、身体状况、产奶量等传入 Flash 中，然后在相应的奶牛参数展示窗口中显示，或者是不用点击，当鼠标移动到某个奶牛的对象上时。参数自动传递，进行查询返回。。

继续发现。。。

VFP 和 Flash 的联姻无疑是给 Foxer 们打开另一扇大门，虽然应用过程中可能还有许多的问题和不便，不过就像 VFp 迷所提出的问题一样，在一些特殊的展现场合中，这种应用非常有用。

在下一篇文章中，我们还会介绍一些更高级的应用，其中会涉及到 Flash 中一些内置组件的应用等。

资源：

文章位置：飞弧堂 blog：http://blog.vfp.cn