FoxTalk 2.0
Solutions for Microsoft® Visual FoxPro® Developers

2004 年 11 月刊

“哚、哚、哚”有人在家吗？（第三部分） — Andy Kramek & Marcia Akins 著 Fbilo 译

Page.30

在他们九月和十月的专栏中，Andy Kramek 和 Marcia Akins 讨论了一个“用于在应用程序中实现基于角色的安全性”的类的结构和初始化。在上个月的专栏中，他们以展示了该控件是怎样在模块和表单级别上被实现的来作为结尾。这个月，他们将通过展示这个类是怎样被用来管理表单级别和字段级别的安全性来结束这个系列的文章。

半手工的生成器—更简单，更好 — Dragan Nedeljkovich 著 CY 译

Page.41

生成器是个强大的工具，但是有时它并没有完全按照你想要的来做。或许你想编写你自己的生成器，但是发现这对你来说是件大事，你需要一个 GUI，在 Builder.dbf 里注册，为最终用户做说明...，算了，忘了它吧。Dragan Nedeljkovich 为你展示了如何编写你自己的简单、瘦身、半手工的生成器。

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

Page.44

每个月,VFP 开发团队成员都会给你带来有趣的 VFP 技术 TIPS。这个月带给你的是 VFP9 中 XMLAdapter 的新特性：支持多层 XML 数据

玩转 MSDE （一） — Fbilo 原创文章

Page.47

MSDE 是 Microsoft SQLServer 2000 Desktop Engine 的缩写，翻译过来就是“微软 SQL Server 2000 桌面引擎”。对于用户数量不多、资金有限的中小型企业来说，用 MSDE 作为他们的数据存储是最合适不过的了。而在这个领域，尚没有任何其它软件能与 MSDE 竞争。现在，FBILO 将通过本系列文章教你如何玩转这个优秀的数据引擎。

使用集合(第一部分)

原著: Lauren Clarke and Randy Pearson

翻译: YASUR

当 Lauren Clarke 和 Randy Pearson 第一次听说 VFP8 将要包含一个新的集合基类的时候，他们笑了。作为一个工具和框架的设计者，他们羡慕那些支持集合类的开发语言已经有些年头了。现在，在使用 VFP 强劲的集合两年后，他们回过头来整理出一些文章以便指导那些正在拼命发现 VFP 新特征的人们。

这篇文章是 3 篇关于集合开发系列文章中的第一篇，在这部分，介绍了我们在使用集合的时候学到的东西，之所以这篇文章先来是因为它给我们提供了在使用之初避免错误的丰富经验。在第二篇文章中，我们将提供一些在系统设计过程中很有代表性的实例，包括对象模型的构建。第三篇文章将会定位于更高一级的主题：集合类的子类。

动机

如果你使用过一些象 **Microsoft Office** 应用程序这样的自动化部件，你应该已经注意到了它们自带的结构化“语言”，可以用来访问文档或者文档内的对象。例如，要用 **VFP** 打开一个名为**“LetterToTom”**（给 **TOM** 的信）的 **WORD** 文档，并从中取出第三段的内容，可以使用象下面这样的代码：

```
loWord = CREATEOBJECT("Word.Application")
loDoc = loWord.Documents.Open("c:\docs\LetterToMom.doc")
lcText = loDoc.Paragraphs(3).Range.Text
```

这里除了 **Range** 对象可能会有点难解以外，您应该能很容易理解其它的代码。甚至一个非程序员也能猜出其中的大部分内容。**office** 自动化程序既简单又强大，因为每个 **OFFICE** 应用程序都有一个功能强劲的对象模型。集合在把各种不同的类集成为一个简明形象的结构(一个"对象模型")中起到了黏合剂的作用。在上述的代码段中，同时用到了 **Word** 的 **documents** 和 **paragraphs** 两个集合。

VFP 早已经有了它自己的内部集合，例如页框(**PageFrame**)有一个 **Pages** 集合。但是在 **vfp8.0** 之前，开发人员不能创建他们自己的集合。不过随着 **Collection** 基类的引入，这种情况发生了改变。除了其它的用途以外，它使得我们可以为我们的应用程序、框架或工具构建它们自己的对象模型，并为开发人员提供了更灵活的接口。

已经有各种书籍和文章讨论了集合（参见本文末尾的参考部分）。在开始讲之前，我们先假定大家对这个类已经有了一点基本的了解——它的接口相对来说还是比较简单的，然后我们将以详述从经验中学习到的内容开始讲解。

我们所学的内容可以分为三个部分：

- 键值的使用
- 对象间的关系，引用，以及包含。
- 错误和调试

键的使用

大多数使用集合的方案都会在向集合中添加 Item(数据项)的时候为数据项指定一个键。键是 ADD 方法的第二个参数。尽管某些关键的查询功能会要求使用键，不过部分开发者也许会对这件事情感兴趣：键的使用并非是必须的。

不过，大部分使用集合的案例下都要用到键，而且对使用键的某些方面的理解是非常重要的，这甚至可以决定对于一个特定的设计方案来说使用集合是不是一个合适的选择。

第一：键值必须是字符型的，在一个使用了集合的数据驱动模型中，把从一个后台数据库实体中取来的主关键字或候选关键字当作用于集合的键使用，是非常自然的事情。这种办法对于字符型的键(包括 GUID)来说是不错，但是如果键值是整型的就麻烦了，因为集合的键必须是字符型的。这个问题的原因在于集合 Item 方法的限制。

调用集合的 Item 方法的时候可以给它传递键值或索引。如果键值也可以是整型的话，那么就分不清楚你是在用索引还是键值来使用 Item 方法了。例如：loCol.item(9)指的是索引为 9 的数据项还是一个键值为 9 的数据项？解决这个“问题”的办法用不着引起大家的热烈鼓掌：在这种情况下，你可以在把数据项添加到集合的时候，用 TRANSFORM()把整型的值转换成数据项的键。

第二：键值是大小写敏感的。这意味着如果你增加了一个键值为大写的数据项(例如用 afield()得到的)，然后试图用小写的或混合大小写的方式去引用它的话，那么，结果不是出错(如果你未经检查就调用了 Item())就是找不到这个项目(如果你先用 GetKey()进行了检查)

与那些象 ASCAN 和各种 AT 函数之类的字符串函数不同，对于集合来说，没有可以指定它不对大小写敏感的参数。所以，你应该尝试着去通过一些方法来应付这个问题——比如强制全部大写——但是，我们建议你还是接受现实，并调整你书写代码的风格来正确的操作 VFP 内建的行为。

第三：键值必须是唯一的，下面的代码证明了这一点：

```
loCol = CREATEOBJECT("Collection")
loCol.Add("something", "MyKey")
loCol.Add("something else", "MyKey")
* Error 2062: 指定的键值已经存在
```

这种处理方式是非常正确的，如果允许重复的键值，那么，如果你要用 loCol.item("MyKey")来返回值，VFP 怎么知道要返回的是哪一个数据项的值，如此岂不天下大乱？在设计分析的时候就注意到这种情况，这非常重要，因为在开发的时候通常都会用没有重复键值的数据进行设计和测试，但是当你的

应用变成产品的时候就会发现这种设计有问题。

举一个例子，当你使用一个集合，并在其上建立一个基于某些不能保证唯一性的条件——比如日期——的排序顺序时，就会发生这个问题。如果集合中有两个日期相同的项目，那么它还能继续好好干活吗？

难道这意味着集合不能排序吗？当然不是，但是你得采取措施来保证键值唯一，也许通过使用一些唯一值比如主关键字或 GUID,作为集合的键值,在这个系列文章的后续部分中我们将使用这样一种技术。

调用 item()方法： 隐式 VS 显式

为了从集合中返回一个项目，可以使用 item()方法，可以给它传递一个 key，也可以传递给它一个整型的索引值。但是集合有一个不同于任何其它 Visual FoxPro 基类的功能，那就是 item()是一个“默认的方法”。这就是说，你可以使用精短的方式来直接引用这个对象本身而不用调用 item()方法。换句话说，下面的两行代码实际上是一样的：

```
loForm = MyApp.OpenForms.Item("myForm")
loForm = MyApp.OpenForms("myForm")
```

通过忽略这种对 item()方法的显式调用，你就可以得到这种许多其它对象模型所使用的严谨代码，当你在一个这样的对象模型中多深入几层的时候，这种情况尤其明显。有了这等好事，你是不是再也不想显示的调用 item()方法了呢？然而，请考虑一下这种情况：当有一个内存变量引用了集合并且你想取得集合中的一个项目，比如：

```
? loPages("Home Page")
```

如果你正在命令窗口中测试这段代码是没问题的，但是当你将代码加入到项目中进行编译的时候就会出错。原因是 vfp 项目管理器会检测代码行，并且会认为应该有一个名为"loPages()"的自定义函数，所以就会出现"找不到不明的 loPages"。

这个问题类似于我们引用数组的实际情况;然而,与数组不同的是,没有专门用于集合的 EXTERNAL 标志符。甚至在"loPage"前加个"m."前缀那也是白搭。（注释：你可以对一个集合使用 EXTERNAL ARRAY 来欺骗 Visual FoxPro 以避免它弹出这个错误）所以解决办法之一，就是显式的调用 item()：

```
? loPages.Item("Home Page")
```

遍历技术

到目前为止，我们所举的示例代码都是随机访问的：使用 Item()方法来从集合中取得单个数据项(Item)。对于一个已经生成的集合，另一种主要的用法是遍历它所有的成员，这在报表和分析中很常见，不过也是其它许多领域的基础，例如商业对象设计。有两种技术可以遍历集合中的所有成员，第一种技术使用了 FOR/NEXT 语句和 COUNT 属性：

```
LOCAL lnIndex, loObj
FOR lnIndex = 1 TO TheCollection.Count
  loObj = TheCollection.Item(m.lnIndex)
  loObj.DoWhatever()
```

```
NEXT
```

第二中技术使用了 **FOR EACH** 结构语句：

```
LOCAL loObj
FOR EACH loObj IN TheCollection
  loObj.DoWhavever()
NEXT
```

第二种方法使用了更少的代码，但是使用它的更重要的理由是因为它的灵活性比较强。第一种技术是通过数据项被添加入集合的时间顺序来进行的排序，而第二种技术则让用户可以根据 **KeySort** 属性的设置来进行选择。**KeySort** 的默认值为 **0**，它的结果是用 **Index** 进行升序排列，不过，可用的选项 **(0-3)**支持通过 **Index** 或 **key** 排序，而且可以指定升序或降序。这种灵活性使得第一种技术看起来应该被放弃，但是在这篇文章的后续部分，我们将会看到一个 **VFP8** 的 **BUG**，所以有时候也要使用第一种方法。

对象的对象关系、引用和包含

为了使用集合，你必须理解它们是什么、它们又不是什么。集合与大多数 **VFP** 基类不同，它没有太多的接口（指属性、事件、方法）。此外，我们已经发现，为了扩展集合的功能，我们更倾向于为使用集合的对象定义其行为，而不是建立集合的子类。

集合更像一个新的数据类型，而且在某些方面与 **vfp** 的数组倒颇为相似。尽管集合与数组不同，它是有属性的对象，但其能力还是非常有限的。这个因素也就导致从这里我们所能学到的主要知识：

知识点：集合应该被当作其它对象的成员来部署，任何添加的行为应该被封装在其父对象（容器）中。

我们委婉的这一经验申明为：使用一个集合要比本身是一个集合更好。你会在这篇文章的其它经验中读到其它一些理由。现在让我们再深入的研究一下其中的区别。

就像大多数 **vfp** 基类一样，一个集合既可以作为一个单独的对象，也可以作为一个容器对象——比如一个表单，或者 **Custom** 控件——的成员。要给容器添加一个作为容器成员的集合，可以在类的构造代码中完成：

```
DEFINE CLASS FooWithCollection AS Container
    ADD OBJECT SomeList AS Collection
ENDDEFINE
```

或者，如果你希望用你自己的集合垃圾回收程序来作主导控制一切，可以把“作为任何对象们的聚集体”的集合通过一个简单的对象引用来添加：

```
DEFINE CLASS FooWithCollection AS Container
    SomeList = NULL
    FUNCTION Init()   && 后者以后什么时候...
        THIS.SomeList = CREATEOBJECT("Collection")
```

```
        ENDFUNC
    ENDDEFINE
```

集合和容器

也许要理解的一个最重要的概念就是一个集合和它的数据项（Item）之间的关系。对于集合中的数据项来说，重要的一点是：它没有对集合本身的链接，并且事实上也不知道它们与集合有任何关系。为了更好的明白这一点，下面的为一个集合增加四个集合项的代码实例可以说明一些问题：

```
loCol = CREATEOBJECT("Collection")
* 一个字符
loCol.Add("abc")
* 一个日期
loCol.Add(DATE())
* 一个存在的对象
loCol.Add(_VFP.Application.Forms.Item(1).txtName)
* 一个新产生的对象
loCol.Add(CREATEOBJECT("Custom"))
```

注意这些被添加到集合中去的数据项之间是大不相同的。前两个甚至不是对象，第三个是另一个容器(表单)的成员，第四个是动态生成的对象。从这些数据项的角度来看，它们自身并没有发生什么变化，只是在这个集合中被引用了而已。

知识点：集合了解它的数据项，可是数据项却对集合一无所知。

这里的关键，是要认识到集合和容器没有什么相同的地方。尽管在某种设计目标下可能可以任选它们之中的一种，但经验表明这样的情况很少出现。集合的候补队员通常就是数组、游标或者其它能够管理数据项列表的语言元素。

Visual FoxPro 有多种基类可以作为容器，包括表单、页框、custom 和标准容器，大多数容器对象的一个主要目的，就是作为一个被其它一些对象居住的宿主或父对象来服务。

被包含在容器中对象可以通过构造代码(add object)或者借助容器的 Addobject() 或 NewObject() 方法来添加。任何被这样添加的对象都是作为一个子对象存在的。一个子对象只可以是一个容器对象的子对象，可以通过使用它的 Parent 属性来引用这个容器，而且不能被移动到其它容器。这些特性里没有一个是和集合一样的。

一个集合对象不拥有任何东西。实际上，vfp 里的一个集合对象从来不会扮演象一个容器那样的角色(调用 collection.AddObject 会产生一个错误)。相反，当一个集合与一些对象一起使用的时候，集合也只是简单的维护一个对这些对象的引用的列表。

这些对象可以存在于任何地方，它们可以是相似的，也可以完全不同，甚至可以是其他容器的成员。显然，一个集合中对象没有办法去这个集合，在设计时缺少了相关联的资源，甚至不知道它是否被一个

集合所引用。而另一方面，一个对象都可以很容易的成为多个集合的一部分。

容器还是集合？

也许你已经看出来了，你必须从容器和集合中选择一个。动力在于看一个次要对象是否应该从属于一个主对象，以及是否能够引用其父对象的属性等，等等。但是集合可以提供更多的灵活性，尤其是它的能够在叠代期间控制内部对象的排序的能力。解决方案是：如果你需要一个容器但又需要集合的，那么就两个都用上。或者，说的更准确一点，把你的容器设计为通过部署一个集合来扩展它的基本特性。

举个例子，让我们想象一下一个商业对象的设计，现在有一个发票对象，它是一个容器，其中包含着一个由许多成员行数据项组成的数组。这个数组是使用一个中间层服务从数据库生成的，中间层服务总是根据主关键字来对数组进行排序。假定你的 **Web** 应用程序要生成一个 **HTML** 表，表中的行数据项是使用类似于象下面的 **ListLineItems** 方法那样的代码生成：

```
DEFINE CLASS Invoice AS Container
* 数组由独立的 BizObj 按照固定的顺序填充
DIMENSION aLineItems[1]
FUNCTION ListLineItems
  FOR EACH loItem IN THIS.aLineItems
    * 对数据项进行操作的代码
  ENDFOR
ENDFUNC
```

但是你的客户现在想让这些行数据项能够以类别号而不是默认的方式来排序。一个解决方法，就是不要去在 **ListLineItems()**方法中遍历所有的成员，而是使用集合，代码如下：

```
FUNCTION ListLineItems
  LOCAL loCol
  loCol = CREATEOBJECT("Collection")
  * 遍历所有的成员，并把它们放入临时集合:
  FOR EACH loItem IN THIS.aLineItems
    loCol.Add(m.loItem, m.loItem.Catalog_No)
  ENDFOR
  * 现在使用这个集合来输出:
  loCol.KeySort = 2  && 设置为"使用 key，升序"排序
  FOR EACH loItem IN m.loCol
    * 对数据项进行操作的代码
  ENDFOR
ENDFUNC
```

实际上，你可以通过参数化排序的条件来使得这种方法更具灵活性，而不是把代码写死了使用 **Catalog_No** 排序。此外，还要注意到前面的代码中如果出现了一张发票中存在着多个相同类别号的数

据项的话是会出错的，因为键值必须是唯一的。我们可以通过在第六行中把键值与某些唯一的东西——比如主关键字——组合起来来解决这个问题。上面的第六行可以换成如下代码：

```
loCol.Add(m.loItem, m.loItem.Catalog_No + ;
  TRANSFORM(m.loItem.PK))
```

这种部署一个集合以实现不同的排序顺序的技术甚至可以用在那些已经使用了一个集合、但是数据项却没有安装希望的方式排序的设计上。只要再建立一个集合，让它使用会生成你想要的排序方式的键值，就行了。

集合需要手动去处理对象垃圾回收吗？

一般来说，**vfp** 很擅长于在对象已经不再被使用的情况下去释放已经被它们消耗了的资源。如果没有释放资源的话，随着渐渐的内存“泄漏”的发生，将导致你的应用程序慢慢地停下来。会导致这种情况发生的一种常见设计错误是：除了建立从属对象的主对象以外，还有另外一个对象中也有着对从属对象的一个对象引用。由于集合天生不拥有（包含）任何其它对象，那么，如果你不小心的话，集合所拥有的任何对象引用都有可能造成垃圾回收问题。

关于 **vfp** 的垃圾收集的问题的主题在这里有比较详细的说明：

http://fox.wikis.com/wc.dll?Wiki~ManualGarbageCollection~VFP

对于集合来说，要注意的是，当向其中加入一个数据项，而该数据项是对某个特定对象的引用的时候，你就是正在增加这个特定对象的引用计数。然后，当这个对象不再被需要的时候，如果释放了该对象而集合没有被释放，那么，只要你的应用程序还在运行，你就会碰到一个可能会导致内存泄漏的垃圾回收问题，也可能出现其它的问题。这个问题的解决办法，就是写一段代码来处理当对象不再被需要的时候从集合中删除这些引用。

像工厂方法这样的设计模式(工厂模式)可以通过提供对对象的建立和释放时的管理来帮助我们减少这类的问题。这类代码的本质就像下面的代码段，你可以想象有一个持久性的管理对象（通过私有变量 **WorkerManager** 引用）存在，它有着象下面的 **CreateWorker** 和 **ReleaseWorker** 那样的方法。下面我们还提供了从这个应用程序的别的什么地方调用该对象及其方法的示例代码：

```
  * 工厂方法
FUNCTION CreateWorker(tcClass, tcKey)
  LOCAL loObj
  * 建立对象
  loObj = CREATEOBJECT(m.tcClass)
  * 将对象添加给一个集合
  THIS.oWorkerCollection.Add(m.loObj, m.tcKey)
  * 把对象返回给客户端代码使用
  RETURN m.loObj
ENDFUNC
```

```
FUNCTION ReleaseWorker(tcKey)
  WITH THIS.oWorkerCollection
    LOCAL lnKey
    * 检查数据项是否还在集合中
    lnKey = .GetKey(m.tcKey)
    IF m.lnKey > 0
      .Remove(m.lnKey)
    ENDIF
  ENDWITH
  RETURN
ENDFUNC

* 客户端调用代码:
lcWorkerKey = SYS(2015)  && 唯一键
loWorker = WorkerManager.CreateWorker( ;
  "SomeWorkerClass", m.lcWorkerKey)
* ...执行某些工作，然后当完成的时候:
WorkerManager.ReleaseWorker(m.lcWorkerKey)
```

这段代码不是一个完整的例子，确切的说，它更倾向于去演示如何去系统的创建和删除对象以避免出现垃圾回收问题。

知识点：涉及向集合添加对象引用的设计可能会需要手动进行垃圾回收。

错误和调试

在我们使用集合进行设计之前，有一些经常会遭遇到只有集合才有的技巧、陷阱、和错误调试等的情况，需要我们多加注意。其中的一些在前面的内容中我们已经提到了。

也许最常见的新错误就是**#2061(**集合中没有与索引或表达式相匹配的一个成员**)**，这种情况通常发生在当一个键并不存在于集合中的情况下，你使用了该键试图去从集合中返回一个数据项。这可能是由大小写敏感、书写错误、或者时间**(**在集合被载入之前去访问一个数据项**)**等问题引起的。为了对付这些问题，或者使用 **TRY/CATCH** 结构语句，或者象下面这样在代码中先使用 **GetKey()** 进行一下检查：

```
lnKey = loColl.GetKey(m.lcKey)
    IF m.lnKey > 0  && 数据项存在
      RETURN loColl.Item(m.lnKey)
    ELSE
      RETURN .F.  && 假定这是调用代码的需要!
    ENDIF
```

集合项在调试器中不能被展开是比较头疼的问题，而且很难去解决，但是出现这种的问题却很充分：访问集合项的唯一方法就是通过 **item()**，而这是一个方法，不是一个可以被轻易展开的数组属性。在 **FoxPro Advisor** 的 **2003** 年 **7** 月刊中，**Randy** 提供了对该问题的一个调试“解决办法”，其中包括添加了一个 **Ravel()** 方法，该方法会将集合的数据项们的值拷贝到一个数组属性中去，而这是在调试器中是完全可以看到的。

Visual FoxPro 8 在使用 **FOR EACH** 遍历集合、然后使用 **AMEMBERS()** 或者 **BINDEVENT()** 的时候有一个 **bug**。**FOR EACH** 方法遍历集合的时候有一个很成问题的行为，会导致一个非常难于判断的问题。在这里 **Randy** 对这个情况有非常详细的介绍：

www.west-wind.com/wwthreads/ShowMsg.wwt?MsgId=0Y40USMEN

简要的说一下，**FOR EACH** 结构使用了一个能提供一个对象引用的枚举器，它提供的对象引用并非是对真正幕后的 **Visual FoxPro** 对象的引用，而是一个不在我们控制之下的内部代理。在每一天的使用中，这个内部代理传递所有的属性、事件和方法，所以我们从来没有注意到其中的真相。但是如果你把这个对象传递给 **AMEMBERS()**，你得到的就会是可能有着严重问题的不期望的结果。如果这个问题影响了你的设计，解决办法就是使用这篇文章前面部分所提到过的技术：使用 **Count** 属性而不是 **FOR EACH** 来遍历集合。

当你象前面那样使用 **FOR EACH** 结构来遍历集合、并调用一个 **BINDEVENT** 的时候，也会出现同样的问题，如下面的代码所示：

```
CREATEOBJECT("ShowBug")
DEFINE CLASS ShowBug AS CUSTOM
  ADD OBJECT Textboxes AS Collection
  FUNCTION INIT
    loTxt = CREATEOBJECT("Textbox")
    loTxt.Value = "Some text"
    THIS.Textboxes.Add(m.loTxt)
    FOR EACH loObj IN THIS.Textboxes
      BINDEVENT(m.loObj, "Value", THIS, "OnValueChange")
      * Error 1734, "属性值没有找到"
    ENDFOR
  ENDFUNC
  FUNCTION OnValueChange
    * 这里是事件处理...
  ENDFUNC
ENDDEFINE
```

无法想象，你刚刚建立并使之显示的一个对象——它的属性竟然不存在！你可以通过使用 **COUNT** 属性而不是 **FOR EAC H** 结构来遍历集合以绕开这个问题。然而，你必须要使用由 **FOR EACH** 提供的

排序顺序功能的话，这个解决办法就帮不上忙了。不过，你可以通过在 BINDEVENT 代码之前插入下面这样的“有意义”的代码行来骗得 Visual FoxPro 去寻找 Value 属性：

```
* 不知是什么原因，这会让 Value 属性变得可以使用
= loObj.Value
```

不用说，这种方法在使用的时候极为不爽。VFP9 Beta 版已经提供了解决方法，就是给 FOR EACH 命令添加一个可选的 FOXOBJECT 子句。在这个例子中，你可以使用：

```
FOR EACH loObj IN THIS.MyText FOXOBJECT
```

与采取补救办法相比，我们更推荐使用这种方法，因为你必须了解内部的复杂情况并且知道什么时候去插入这个新的子句。

知识点：遍历集合的代码产生了一些新的 VFP 错误。

下期再见：

这个系列文章的第一篇的目的，是提供一些我们在使用 Visual FoxPro 集合的过程中所掌握的经验。下一期，我们将应用这些知识点到一些设计理念中，包括如何创建一个类似于在 MS OFFICE 自动化中的对象模型。

从 XSouce 中淘金，第二部分

原著：Doug Hennig

翻译：Fbilo

在这个系列文章的第一部分中，Doug Hennig 探索了 XSource 中一些有趣的部件，XSource 指的是大多数 VFP 自带的 “Xbase”工具的源代码。这个月的文章将继续挖掘可以用在你自己的应用程序中的代码。

如我上个月文章中所述，VFP 在一个 ZIP 文件 XSource.ZIP 中包含了几乎所有自带的 Xbase 工具的源代码，该文件位于 VFP 主目录下的 Tools\XSource 子目录。把这个文件解压到 Tools\XSource 下将会生成一个名为 VFPSource 的子目录，其中就包含了 Xbase 工具的源代码。在第一部分中，我谈到了包含在 XSource 中的一些图像，以及你可以用于将设置保存到 VFP 资源文件中去、建立面对对象的快捷菜单的部件。这个月，我将谈谈你可以用到自己应用程序中去的更多部件。

显示视频文件

当你删除或者移动大量文件的时候，Windows 资源管理器会显示一个很好的对话框，它以动画的形式来告诉你正在做什么事情。虽然它没有让整个处理过程变得更快些，但至少让你知道你的系统正在工作而不是死机了。

从某个版本开始，VFP 发行的时候就在它的 Graphics\Videos 子目录下自带了一些动画文件。你可以使用这些动画文件来提供与 Windows 资源管理器一样的进程对话框。在 FoxRef.VCX (位于 XSource 的 FoxRef 子目录下) 类库中的 cAnimation 类可以让我们轻松的做到这一点。简单的把它拖放到一个表单上并调用它的 Load 方法，给该方法传递一个要播放的视频文件名来进行播放。如果你想要加载播放文件但却不想马上播放，就把 lAutoPlay 属性设置为.F.，然后在要播放的时候调用它的 Play 方法就可以了。cAnimation 是 Microsoft Animation 控件的一个子类，所以，在发布你的应用程序的时候需要带上它的 ActiveX 控件（MSCOMCT2.OCX）。

为了让它更容易使用，我在 SFXSouce.VCX 中建立了一个它的子类，名为 SFAnimation。这个子类有一个 cFileName 属性，它包含着视频文件的文件名，而这个类的 Init 方法会调用它自己的 Load 方法，并把 This.cFileName 传递给 Load 。所以，只要把 SFAnimation 控件拖放到一个表

单上，设置它的 cFileName 属性，你就搞定了。

运行包含在这个月的下载文件中的 TestAnimation.SCX，可以看该控件的一个示例。它会播放 VFP 自带视频文件中的 6 个文件，显示出类似于你在 WIndows 资源管理器的进程对话框中看到过的动画。

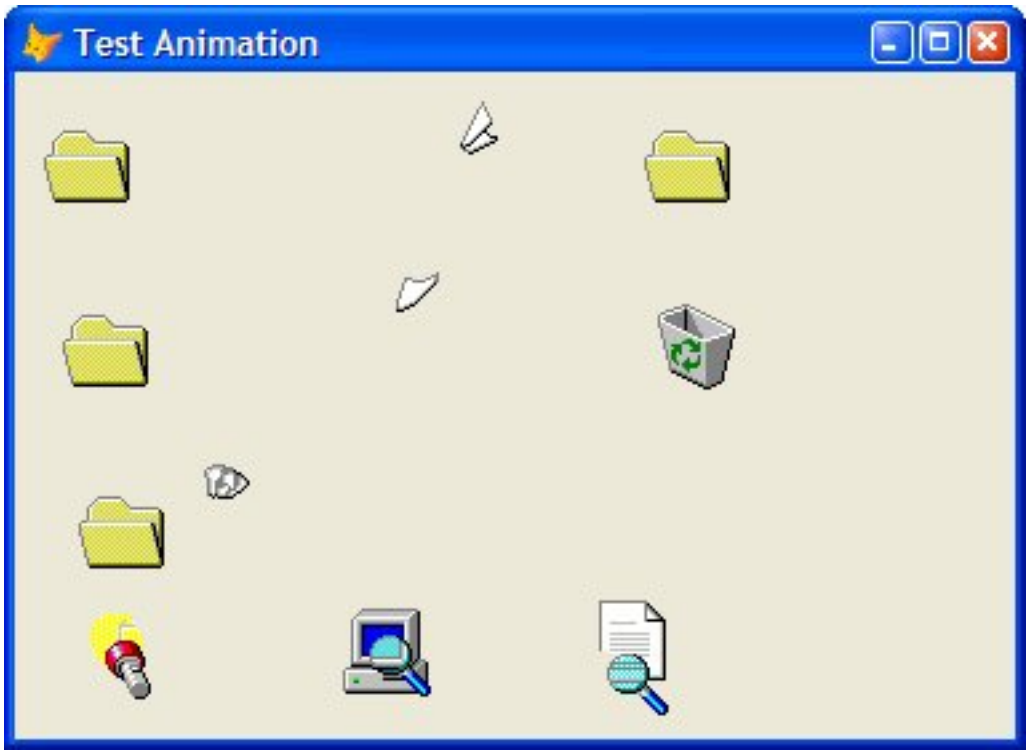

图 1

进程对话框

显示一个动画并不总是能给予用户关于一个运行时间过长的进程的足够信息。许多时候，用户想要知道一个多步骤进程的进度，或者想要看到一个指示出进程进度多少的进度条。在 XSouce 的 Toolbox 子目录下的 FoxToolbox.VCX 类库中的 cProgressForm 类向你提供了一个不错的能够用于这种情况的进程对话框（在 FoxRef.VCX 中有另一个版本的这个类，不过 Toolbox 中的这个版本的能力要强一些）。

如你在图 2 中所见，cProgressForm 提供了几个功能：

- 一个动画图像--计算机上面有一把放大镜，它表示正在发生某些事情（这不是一个视频文件，它是一个带动画的 GIF）。

- 一个整个进程的说明；
- 一个表示进程的进度的进度条；
- 一个状态消息，表示进程当前执行到哪一步；
- 一个 **Cancel** 按钮，让用户可以中止这个进程；

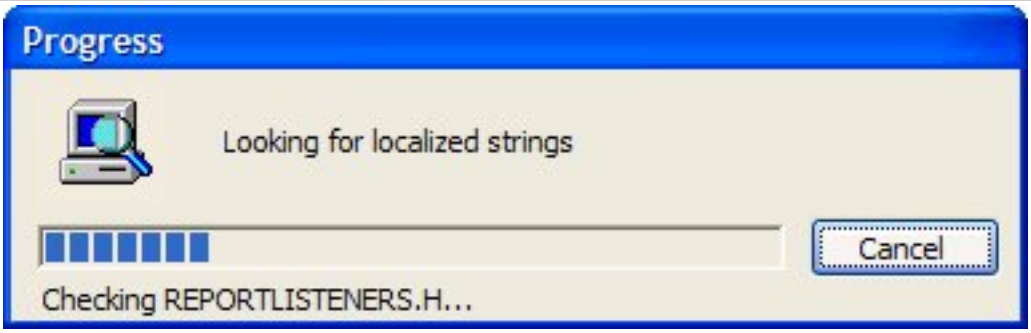

图 2

当你建立 **cProgressForm** 的实例的时候，你可以选择传递给它进程的说明（例如“取得数据”）来设置说明标签的标题。你也可以通过调用 **SetDescription** 方法来设置这个说明。在你的进程执行过程中，你也许会想要更新进度条下面的状态标签的标题，以显示当前正在运行哪一步；把希望设置的字符串传递给 **SetStatus** 方法就能做到这一点。

如果你希望进度条显示一个绝对值--例如相对于记录总数的当前记录号--而不是完成百分比，可以通过调用 **SetMax** 方法来设置最大值。如果你不想显示 **Cancel** 按钮，可以把 **lShowCancel** 属性设置为 **.F.**。如果你不想显示出进度条，可以把 **lShowThermometer** 属性设置为 **.F.**。

cProgressForm 有一对方法与我的设想不一样。首先，你需要手动的调用它的 **Show** 方法来显示这个表单，而我更希望在需要的时候只要我一调用 **SetProgress** 方法就自动显示它。其次，它在 **INIT** 事件中把一个自定义属性 **nSecnods** 设置为 **SECONDS()**，这个属性可以被用来判定一个进程所需的时间总数，可我觉得在 **Init** 中就设置这个属性是不是太早了点。所以，我在 **SFXSource.VCX** 中建立了一个名为 **SFProgressForm** 的子类，在该子类的 **INIT** 方法中先 **DODEFAULT()**然后再把 **nSeconds** 设置为 **0**，而 **SetProgress** 方法的内容则是：如果表单不可见则把它的 **Visible** 设置为 **.T.**，如果 **nSeconds** 为 **0** 则把这个属性设置为 **SECONDS()**，然后使用 **DODEFAULT()**来执行默认的行为。

附带的示例文件中的 **TestProgress.PRG** 是演示这个功能的一个示例，它会从 **VFP** 主目录下的 **FFC** 子目录的文件中搜索本地化字符串（在名称中包含“_LOC”的常量）。它使用 **SetMax** 来指定被搜索的文件的总数，并在每一个文件被搜索过以后更新进度条。你可以试试取消对设置 **lShowCancel** 和 **lShowThermometer** 属性的注释来看看在这种情况下这个进程对话框的表现。

```
* 搞清楚我们要处理多少个文件
lnFiles = adir(laFiles, home() + 'ffc\*.*')

* 建立进程对话框的实例，
* 并设置一些属性
loProgress = newobject('cProgressForm', 'SFXSource.vcx')
loProgress.SetDescription('Looking for localized ' + ;
  'strings')
loProgress.SetMax(lnFiles)
*loProgress.lShowCancel        = .F.
*loProgress.lShowThermometer = .F.

* 在文件们中搜索本地化字符串，
* 并在每次搜索完以后更新进度条
lnFound = 0
for lnI = 1 to lnFiles
  lcFile       = laFiles[lnI, 1]
  lcFullFile = forcepath(lcFile, home() + 'ffc')
  lcExt        = justext(lcFile)
  do case
    case inlist(lcExt, 'H', 'PRG')
      lcContents = filetostr(lcFullFile)
      lnFound     = lnFound + occurs('_LOC', ;
        upper(lcContents))
    case lcExt = 'VCX'
      use (lcFullFile) again shared
      lnFound = lnFound + occurs('_LOC', upper(METHODS))
  endcase
  if not loProgress.SetProgress(lnI, 'Checking ' + ;
    lcFile + '...')
    exit
  endif not loProgress.SetProgress ...
next lnI
* 显示结果
messagebox(transform(lnFound) + ' instances found ' + ;
```

```
        'in ' + transform(seconds() - loProgress.nSeconds) + ;
        ' seconds.')
```

注意 cProgressForm 使用了一个 ActiveX 控件 Microsoft ProgressBar，所以你需要在发布应用程序的时候带上 MSCOMCTL.OSC。它还使用了一个 GIF 动画文件，就是 XSource 下 Toolbox 子目录中的 FindComp.GIF，所以当你编译你的项目的时候该文件会被添加到你的项目中去。

Outlook Bar

即便在发布多年以后的今天，Outlook 仍然在影响不少应用程序的表现和行为。其中被抄袭最多的一个界面元素就是 Outlook bar——在 Outlook 界面中左边的一套控件。这些控件提供了一系列可折叠的类别，每个类别中都包含有一些独特的项目，可以让你通过选择其中一个来指定在 Outlook 的右边要显示什么东西。

目前有大量能够模仿出 Outlook bar 的 ActiveX 控件可供你购买，其中的某些能够在 VFP 中使用，某些则不行。然而，现在 VFP 中也有个应用程序用上了一个类似的控件，而且我们在 XSource 中还有它的源代码，这就是 Toolbox。

在 VFP 8 中增加的 Toolbox 是为了提供一种类似于 Visual Studio .NET 中选择控件和其它常用项目的界面。由于它并非完全与 Outlook 相同，所以我还是把这个能维护类别以及每个类别下面所有子项目的控件叫做“Toolbox”。

如图 3 可见，Toolbox 包含 4 个有趣的区域（不计入底部的帮助文本）。一个类别能提供一系列项目。在 Toolbox 中，类别显示为一系列的条。一次只能有一个类别被展开，其余的则只显示类别条。当即一个条可以展开该类别，同时以前展开的类别则会被关闭。

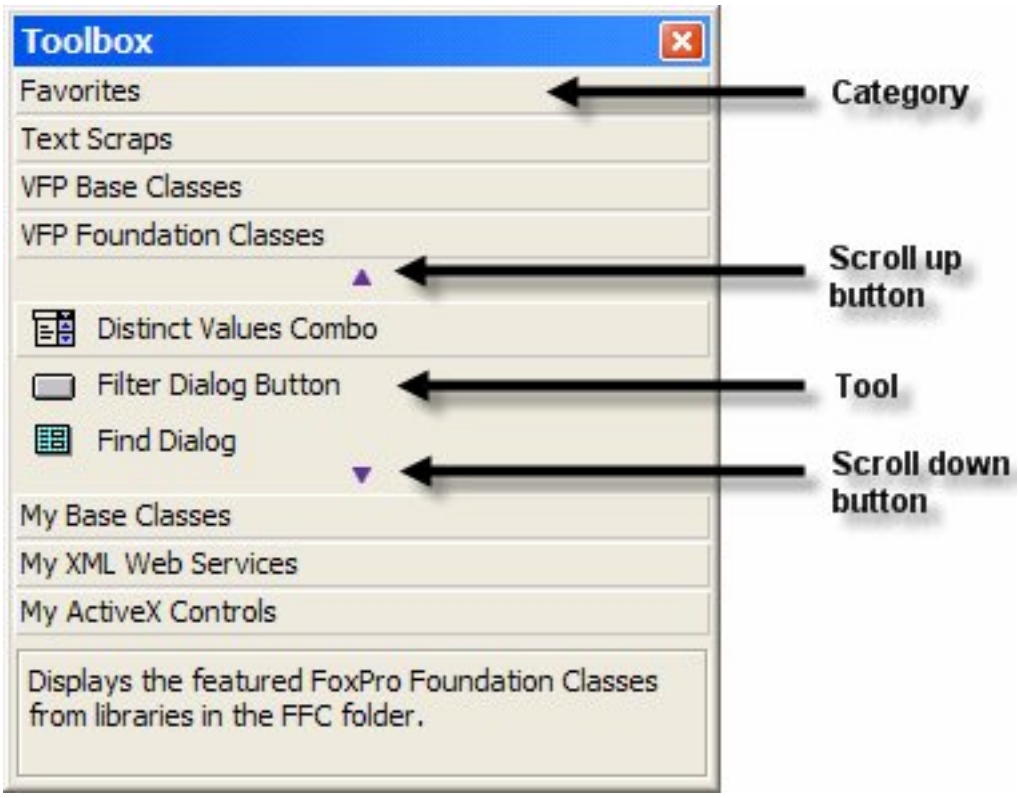

图 3

一个工具（**tool**）是在类别中的一个项目。工具有一个图片、一个名称、以及一个 **ToolTip**（提示）。你可以在这些工具上执行各种操作：单击、右键单击、拖放等等。此外，在一个类别中还各有一个向上滚动和向下滚动按钮，你不需要在这两个按钮上单击，只要把鼠标指针移动到其中一个上面停留几秒，它就会自动开始向相应的方向滚动。

组成 **Toolbox** 的类们存在于 **XSource** 下的 **Toolbox** 子目录中的 **FoxToolbox.VCX** 和 **_Toolbox.VCX** 中。在 **FoxToolbox.VCX** 中有趣的类是 **ToolContainer**，它包含着一个类别的控件和逻辑。有多少类别，用于显示 **Toolbox** 的表单就会包含多少个 **ToolContainer** 对象。

每个 **ToolContainer** 的 **oToolItem** 属性中含有对一个来自 **_Toolbox.VCX** 类库中的 **_Category** 对象的引用（或是它的一个子类；不同的子类用于在 **VFP** 的 **Toolbox** 应用程序中各种不同类型的类别）。**_Category** 对象有着用于类别的一系列属性和方法。基于我们的用途来说，**_Category** 并非那么重要，但某些 **ToolContainer** 方法要求有它存在。

每个 **ToolContainer** 还维护着一个 **Tools** 的集合，集合中的每一项都代表一个来自 **_Toolbox.VCX** 类库的 **_Tool** 对象（或是它的一个子类；不同的子类用于在 **VFP** 的 **Toolbox** 应用程序中各种不同类型的工具）。**_Tool** 对象有着用于一个工具的一系列属性和方法。

在 **Toolbox** 应用程序中的 **main** 表单是基于在 **FoxToolbox.VCX** 类库中的 **ToolboxForm** 类的，本来，我想建立一个这个类的子类，但是不幸的是：它严重的依赖于执行与 **Toolbox** 应用程序而不是一个通用的 **toolbox** 相关的任务。所以，我建立了 **SFToolboxForm (**在 **SFXSource.VCX)** 来作为使用 **Toolbox** 控件的表单的基类。

从它的 **Init** 调用的 **ShowToolbox** 方法负责建立 **Toolbox**。

```
local loCategories, ;
  lnI, ;
  loCategory, ;
  lcCategory, ;
  loContainer, ;
  loTools, ;
  lnJ, ;
  loTool
with This
* 取得一个类别的集合，并对每一个类别进行处理
  loCategories = .GetCategories()
  .nCategories = loCategories.Count
  for lnI = 1 to .nCategories
    loCategory = loCategories.Item(lnI)
* 建立一个 ToolContainer 对象，并设置它的属性
    lcCategory = 'Category' + loCategory.cID
    .NewObject(lcCategory, 'ToolContainer', ;
      'FoxToolbox.vcx')
    loContainer = evaluate('.' + lcCategory)
    with loContainer
      .SetFont(This.FontName, This.FontSize, '')
      .oToolItem   = newobject('_Category', ;
        '_Toolbox.vcx')
      .CategoryID = loCategory.cID
      .Caption     = loCategory.cName
      .Width       = This.nToolboxWidth
      .Visible     = .T.
    endwith
```

```
    * 如果这是第一个类别，就将所有类别的标准高度设置为这个类别的高度
    * 并使它成为默认被展开的那个
        if lnI = 1
          .nStandardHeight   = loContainer.Height
          .cCurrentCategory = loCategory.cID
        endif lnI = 1
    * 为当前的类别取得一个工具的集合
    * 对于每一个工具，将其 OnClick 方法绑定到我们的 ToolSelected 方法
    * toolbox 类要求它的 oEngine 属性要引用一个带有一个 DblClickToOpen 属性的对象
    * 所以这里建立一个。
    * 把 tool 对象添加给类别容器
        loTools = .GetToolsForCategory(loCategory.cID)
        for lnJ = 1 to loTools.Count
          loTool = loTools.Item(lnJ)
          bindevent(loTool, 'OnClick', This, 'ToolSelected')
          loTool.oEngine = createobject('Empty')
          addproperty(loTool.oEngine, 'DblClickToOpen', .F.)
          loContainer.AddTool(loTool)
        next lnJ
      next lnI
    * 调整类别的外形大小
      .ResizeToolbox()
    endwith
```

这段代码由调用 **GetCategories** 方法来返回一个带有关于每个类别信息的对象集合开始。**GetCategories** 方法被封装在这里，是因为真正的填充集合的机制将会是不一样的。然后这段代码遍历这个集合，为每个类别向表单添加一个 **ToolContainer** 对象。**ToolContainer** 的宽度被设置为表单的 **nToolboxWidth** 属性，所以如果你建立了一个这个表单的子类，请确保设置这个属性。

对于第一个类别，这段代码会将它自己的 **nStandardHeight** 属性设置为当前容器的高度，该属性包含着每个类别的标准高度。它还会把 **cCurrentCategory** 属性设置为该类别的 **ID**，这样一来，该类别就会在稍候我们将谈到的代码中自动被打开。

下一步，代码会调用 **GetToolsForCategory** 方法，封装在这个类中的该方法会返回一个 **_Tool** 对象的集合，这些 **_Tool** 对象代表属于某个特定类别的工具。当用户在一个工具上单击的时候，**toolbox**

将调用相应 **tool** 对象的 **OnClick** 方法。我决定在表单级别上处理单击，所以使用了 **BINDEVENT()** 来将每个 **tool** 对象的 **OnClick** 方法绑定到表单的 **ToolSelected** 方法上。由于各种 **toolbox** 方法要求一个 **tool** 对象的 **oEngine** 属性应该要引用一个带有 **DblClickToOpen** 属性的对象，所以这段代码就建立了那个一个对象。然后，**tool** 对象被添加到类别容器的工具集合中去。

最后，**ShowToolbox** 调用 **ResizeToolbox** 方法来打开选定的类别（在这里，就是第一个类别），并关闭其它的类别。基于版面的缘故，这里就不写出它的代码了，请自行查看。

当用户在一个类别上单击的时候，**SetCategory** 方法被从 **toolbox** 代码中调用。这个方法的代码所做的就是将 **cCurrentCategory** 属性设置为被选中类别的 **ID**，然后调用 **ResizeToolbox** 方法来处理新的选择。

SFToolboxForm 包含三个封装的方法：**OnRightClick**，当用户在一个工具上单击鼠标右键的时候被调用；**ResetHelpFile**，将帮助文件设置为默认的文件；**SetHelpText**，它在 **Toolbox** 应用程序中是用来将关于被选中工具的一些信息放入到表单底部的帮助区域中去的。尽管在这个类中它们没有起到什么作用，不过你可以在一个子类中实现适合的自定义行为。

TestToolbox.SCX 是一个基于 **SFToolboxForm** 的示例表单。它使用了一个 **toolbox** 来显示在一个帐单系统中的不同模块。它从一个 **MODULES(模块)**表中读取这些模块，该表包含着类别的记录（例如“应收款”和“订单输入”等等）、以及每个类别下面的模块（例如“客户”、“订单”等等）。它的 **GetCategories** 方法中拥有以下代码，该方法调用 **ShowToolbox** 来填充一个类别的集合：

```
local loCollection, ;
  loCategory
loCollection = createobject('Collection')
select NAME, ID ;
  from MODULES ;
  where PARENTID = 0 and ACTIVE ;
  into cursor _CATEGORIES
scan
  loCategory = createobject('Empty')
  addproperty(loCategory, 'cName', trim(NAME))
  addproperty(loCategory, 'cID',    transform(ID))
  loCollection.Add(loCategory)
endscan
use
```

```
    return loCollection
```

这段代码以一些拥有来自于在 **MODULES** 表中各个类别（那些 **PARENTID = 0** 的记录）的 **cName** 和 **cID** 属性的对象来填充一个集合。

也是从 **ShowToolbox** 方法中被调用的 **GetToolsForCategory** 方法负责为某个指定的类别填充一个工具的集合。

```
    lparameters tcID
    local loCollection, ;
      loCategory
    loCollection = createobject('Collection')
    select NAME, ID, CLASS, LIBRARY, IMAGEFILE ;
      from MODULES ;
      where PARENTID = val(tcID) and ACTIVE ;
      into cursor _MODULES
    scan
      loModule = newobject('_Tool', '_Toolbox.vcx')
      with loModule
        .ToolName   = trim(NAME)
        .UniqueID   = transform(ID)
        .ParentID   = tcID
        .ImageFile = textmerge(trim(IMAGEFILE))
        addproperty(loModule, 'cClass',    trim(CLASS))
        addproperty(loModule, 'cLibrary', trim(LIBRARY))
      endwith
      loCollection.Add(loModule)
    endscan
    use
    return loCollection
```

这段代码是从 **MODULES** 表中选择那些 **PARENTID** 为指定类别的 **ID** 的记录来填充。在集合中的每个对象都是一个 **_Tool** 对象，其属性也相应的被设置好了。此外，这段代码还增加了一对自定义属性：当工具被选中的时候，要被显示在表单右面的控件容器的类和类库。

感谢 BINDEVENT(),当用户在一个工具上单击的时候,ToolSelected 方法被调用。ToolSelected 给表单的右面添加一个对象，使用被选中工具的 cClass 和 cLibrary 属性来指定为这个新添加的对象使用什么类。

```
local llLockScreen, ;
  laEvents[1], ;
  loModule, ;
  lcClass, ;
  lcLibrary, ;
  llHaveIt, ;
  lnI, ;
  loControl, ;
  llDesiredModule
with This
* 锁定屏幕，直到完成后再刷新
  llLockScreen = .LockScreen
  .LockScreen  = .T.
* 取得用于选定模块的类和类库
  aevents(laEvents, 0)
  loModule  = laEvents[1]
  lcClass   = lower(loModule.cClass)
  lcLibrary = lower(loModule.cLibrary)
* 当选中一个模块的时候，隐藏其它的所有模块
  llHaveIt = .F.
  for lnI = 1 to .ControlCount
    loControl = .Controls[lnI]
    if pemstatus(loControl, 'cAlias', 5)
      llDesiredModule   = ;
        lower(loControl.Class) == lcClass
      llHaveIt          = llHaveIt or llDesiredModule
      loControl.Visible = llDesiredModule
    endif pemstatus(loControl, 'cAlias', 5) ...
  next lnI
* 根据需要建立相应类的实例
  if not llHaveIt
    try
```

```
            .NewObject(lcClass, lcClass, lcLibrary)
            loControl = evaluate('.' + lcClass)
            with loControl
              .Left    = 210
              .Width   = This.Width - .Left
              .Height = This.Height
              if version(5) >= 900
                .Anchor = 15
              else
                .OnResize()
              endif version(5) >= 900
              .Visible = .T.
              .SetupControls()
            endwith
          catch to loException
            messagebox('This module has not been defined.')
          endtry
        endif not llHaveIt
      * 刷新屏幕
        .LockScreen = llLockScreen
      endwith
```

图 4 显示的是当表单运行时候的情况。

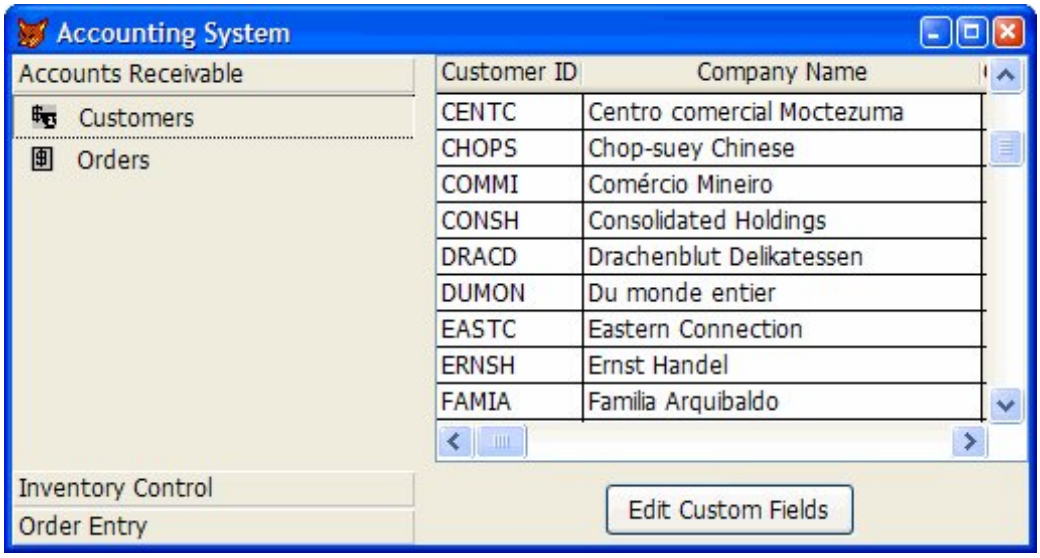

图 4

总结

当你利用好在 **VFP** 自带的 **XSource** 中的部件的时候，在 **VFP** 表单中显示视频文件、在一个较长时间的进程中显示一个进程计、给你的应用程序添加一个 **Outlook bar** 都很容易。下个月，我将探讨一下怎样在一个表单中建立一个滚动的区域、以及怎样使用 **XSource** 部件来建立你自己的生成器。

下载：**411HENNIG.ZIP**

这到底是谁的活?

原著: Andy Kramek & Marcia Akins

翻译: Fbilo

（这是 FOXTALK 以前的文章，安排在这里是为了可以更好地理解《哚哚哚有人在家吗》系列文章）

根据一个用户的安全性访问权限来使能和禁止表单上的控件是一个常见的任务。尽管这么常见，解决这个问题的办法——象往常一样，我们说的是在 Visual FoxPro 里——有很多种。这个月，Andy Kramek 和 Marcia Akins 讨论了一种有效的实现方式，但是由于其责任的分配是错的因而显得太死板了。最后他们设计了一种更灵活的方案，因为它将责任分配正确了。

玛西亚：最近，我在一个应用程序上花了不少时间，该程序有一个相当简单的安全模式。根据一个用户的安全等级来决定他或她是否能查看、编辑/删除、或者添加新数据。无论用户何时要求使用一个表单，他们的安全性等级就受到检查，并且（假定他们拥有对表单的访问权限）表单被显示，而表单上的控件则相应的被使能或者禁止(Enabled or Disabled)。

安迪：听起来很标准。我假定你所说的“控件”不仅包括用于显示数据的那些控件（文本框之类的等等），还包括功能控件（命令按钮和工具栏选项）。有什么问题呢？

玛西亚：最根本的问题在于：该应用程序使用了它自己的、自定义生成器生成的框架，并且它是通过给商业对象分配编辑模式来处理工作的。每个控件都有一个名为 cMode 的自定义属性，在该属性里存储着一个表示控件是否应该被使能的字符串。

安迪：打住——你的意思是否是说，如果我想让一个文本框在添加和编辑模式中可用，我就必须显式的设置这个属性？就像这样：

```
cMode = "ADD,EDIT"
```

玛西亚：事实上就是那样！当然，在基类层次上的类们把这个字符串设置为一个默认值。默认情况下，商业对象一开始总是被设置为查看模式，当一个表单的实例被建立的时候，所有的控件都是被禁止的（disabled）。让一个表单进入添加或者编辑模式要求用户去单击相应的按钮。不过，如果某个用户没有对于一个特定表单的权限，那么一个安全性对象就会把那些用户没有权限使用的按钮的 cMode 属性替换为单词“NONE”，这样，这些按钮就自动被禁止了。

安迪：**Okay**，假定你拥有适当的权限，那么，当你单击添加按钮的时候会发生什么事情？从你前面已经讲述的来看，我们有一个商业对象来控制模式。但是控件还是必须要将它们自己的状态设置为使能或者禁止，那么是什么把这两者关联在一起的？

玛西亚：这就是问题的起源。该应用现在使用的办法是：表单有一个方法 **SetMode()**，无论何时某个表单上的控制按钮（例如添加、编辑、删除、或者查找）被单击，该方法就会被调用。所以当添加按钮被单击的时候，表单的 **New()** 方法被调用，而它会去调用商业对象的 **New()** 方法，这会导致商业对象的 **cMode** 属性被设置为**"ADD"**，并完成其它为了给表单提供一条空记录所需的事情。然后表单的 **New()** 方法调用表单的 **SetMode()** 并传递一个**"ThisForm"**参数给它。这个 **SetMode()** 方法递归的调用自己，在这个过程中它会叠代的遍历表单上所有的对象，并根据商业对象的当前模式是否包含在各个对象的 **cMode** 属性中来使能或者禁止它们。

安迪：是的，我了解你的问题了。这种处理这个问题的复杂办法是重要的，因为控制这个模式的责任和设置控件状态的责任都没有分配正确。首先，商业对象不应该知道关于控件的事情，甚至根本不应该在乎控件们处于什么样的状态。它的任务是处理用户界面与数据库之间的交互**——**让一个商业对象有一个添加或者编辑模式根本没有意义。

玛西亚：虽然我赞同你关于商业对象责任的观点，但是我不完全同意它不应该有一个添加或者编辑模式。例如，如果我们当前绑定到了一些 **cursor**，我们使用 **GetFldState()** 来判定当前记录是一个新添加的记录还是一个已有的记录、是否已经对记录做了任何改动。为了判断是否需要一个插入（在添加模式中）或者一个更新**/**删除（在编辑模式中），商业对象必须先知道当前的状态。

安迪：没错，但是你只想到了其中的一部分。事实上，商业对象只对数据中发生了什么改动感兴趣。判断数据中是否有任何改动的责任可以被指定为必须由用户界面来处理。这就是我为什么说我们在这里的责任分配出了错。

玛西亚：**Okay**，我想我可以接受这种观点。但是真正困扰我的问题是表单是在显式的操作每个控件的 **Enabled** 属性。我觉得控件们应该自己负责管理它们的状态。

安迪：我完全同意，而且你现在已经把你的手指放在这里出错的第二件事情上了。如果情况是整个表单被使能或者禁止的，那么确保所有的控件都被相应的设置好就是表单的责任了**——**毕竟，那就是 **SetAll()** 方法的目的了。然而，在这个案例里，实际上却是要根据每个独立的控件的属性设置来决定它是否应该被使能或者禁止。很清楚，这不是表单的责任，那么表单当然就不应该去搞那些控件的属性了。相反，应该是调用控件上一个能处理这个问题的方法。

玛西亚：这与我所理解的 **OOP** 原则是一致的。但是，不这么做还有一个现实的理由。象它的做法，表单必须递归的处理每个对象（以及所有它作为容器时所包含的对象）而不管该对象是否能够被使能或者禁止。为了这个目的，它必须首先检查当前对象是否有一个 **cMode** 属性，然后必须去判定商业对象的当前模式是否被包含在该属性中，而最后它还得相应的去设置 **Enabled** 属性。这本来应该是一个简单的工作，而它这么做起来却让人觉得绕了老大的弯子。

安迪：我确信你肯定故意的漏了一个步骤。**SetMode()** 方法还应该要在目的对象的 **Enabled** 属性之间先检查它到底是否有这个属性！非可视类是根本没有这个属性的，而你的表单上很可能存在着一个 **Collection** 对象、或者某些基于 **Custom** 基类的其它东西。

玛西亚：首先，我先申明我是从别人那里继承来这种机制的，而看起来很明显的是：原来的设计师假定他们永远也不会给任何没有 **Enabled** 属性的东西添加一个自定义 **cMode** 属性。

安迪：但是这样的假定还意味着可能的模式只有 **Enabled = .T.** 和 **Enabled = .F.** 这样两种。那么如果你需要根据表单的模式来显示或者隐藏某些东西、或者根据模式给一个列表使用不同的过滤时怎么办？对我来说这是这个问题的要点——以及为什么这个实现办法是错的（可能也是你为什么觉得它是错的得地方）。实际上它应该是表单来拥有“模式”，而且应该是每个控件自己负责根据这个模式来对自己进行设置。

玛西亚：这样才有道理。那么，为了修正这个问题，我们需要给表单一个 **cMode** 属性，并让表单把这个属性的改动广播发送给所有的控件。幸运的是，这个表单上已经有一个这么做的 **Refresh()** 方法了。我们需要做的唯一一件事情是，给每个控件类的 **Refresh()** 方法添加代码以检查表单的 **cMode** 属性，并做出相应的反应。我们甚至可以使用该控件自己的 **cMode** 属性来定义一个控件应该怎么做出反应。

安迪：正确。使用自带的 **Refresh()** 方法而不是一个自定义的迭代器还解决了你前面办法中的性能问题。**Refresh()** 已经被限定于只有那些当前活动的、并且有一个 **Refresh()** 方法的对象上了——这就意味着象 **Label**、**Shape**、**Image** 之类的东西就被简单的忽略了。

玛西亚：在 **VFP 8.0** 中不一样！**Label** 和 **Shape** 基类现在都有自己的 **Refresh()** 方法了——不过这个问题不大。还有一件事情需要我们去做。

安迪：是什么？

玛西亚：当前的实现办法还假定 **cMode** 是静态的。目前还没有根据数据中的某些值，动态的修改控件

状态的机制或准备。

安迪：我不能确定理解了你的意思——请给我一个例子吧！

玛西亚：**Okay**，现在有一个客户/服务器应用程序，那么当一个表单启动的时候，它还没有数据，商业对象正处于查看模式，而所有的控件都被 **disabled** 了。用户的第一个任务是单击查找或者添加按钮以获得数据。然而，为了保证当商业对象处于查看模式时编辑按钮是可以被访问的，这个按钮的 **cMode** 属性就必须包含单词**"VIEW"**。这就意味着当表单第一次显示出来的时候，这个编辑按钮也是可用的，即使当前并没有数据可以被编辑。很明显，这是没有意义的。我真正需要的，就是一种仅当商业对象处于查看模式、并且它还包含数据的时候，才使能这个按钮的途径。

安迪：**Okay**，我明白你的意思了。为什么不就建立一个新的模式（**"EMPTY"**）、并只把这个模式添加给添加和查找按钮的 **cMode** 属性呢？然后保证商业对象从 **Empty** 模式开始启动，并且仅当它包含数据的时候才被设置为查看模式。

玛西亚：我不怎么喜欢这个主意。虽然它在这个特殊的案例中起到了作用，可它实际上不过是增加了另一个静态值而已。我想要某些真正动态的东西。

安迪：在那种情况下，每个控件都需要另一个可以被运算从而设置它的 **cMode** 属性的属性。你可以把它叫做**"cModeExp"**，并使用一个 **IIF()** 表达式来判定需求的模式。

玛西亚：这个办法我喜欢。那么，如果我改动了这个应用程序框架，以便给表单添加我们已经讨论过的模式以及属性，那么就可以象下面这样设置编辑按钮的 **cModeExp** 属性了：

```
This.cMode = ;
   IIF(EMPTY(Thisform.oBizObj.KeyValue), 'NONE', 'VIEW')
```

而在按钮类的 **Refresh()** 方法中我可以使用象下面这样的代码：

```
*** 检查我们是否有一个要运算的条件
lcExp = This.cModeExp
IF NOT EMPTY( lcExp )
&lcExp
ENDIF

*** 现在相应的设置模式
This.Enabled = ( ThisForm.cMode $ This.cMode )
```

安迪：这看起来能够解决问题。现在，我们就有了各种工作分配正确的对象啦！表单对模式进行定义，而控件们则通过设置它们自己的状态来对这个模式做出反应。

玛西亚：那么商业对象的情况呢？

安迪：它的情况？这个案例里面没有它的角色。如果商业对象需要根据被请求的操作是否执行成功来相应的设置它自己的状态的话，那还好说。可是在使能或者禁止控件的前提下，跟它没有直接的关系。

玛西亚：那么你刚刚说的意思是指：商业对象应该根据被请求的操作是否成功来设置它自己的模式，而不是把设置自己的模式作为一个请求的结果吗？然后表单就可以检查商业对象的模式来相应设置它自己的模式？这个办法应该有效。

安迪：是的，我想应该是这样。这篇文章附带的代码包含一个表单以及一些相关的类，它使用标准的 **VFP** 示例数据来演示了这种途径是怎样被实现的。

下载：**08KITBOX.ZIP**

“哚、哚、哚”
有人在家吗？（第三部分）

原著：Andy Kramek & Marcia Akins

翻译：Fbilo

在他们九月和十月的专栏中，Andy Kramek 和 Marcia Akins 讨论了一个“用于在应用程序中实现基于角色的安全性”的类的结构和初始化。在上个月的专栏中，他们以展示了该控件是怎样在模块和表单级别上被实现的来作为结尾。这个月，他们将通过展示这个类是怎样被用来管理表单级别和字段级别的安全性来结束这个系列的文章。

安迪：上个月，我们讨论了当安全性管理类与 **GenMenuX** 的能力相结合的时候，是怎样在模块和菜单的级别上提供了一种简便的管理基于角色的安全性的。但是最后，当我们开始试着在表单或字段级别上去进行控制的时候，你说事情正在变得“复杂得多”。我考虑关于这个类的整个设想是只做“例外”的事情以便将复杂性减少到最低，或者我遗漏了什么东西？

玛西亚：是的。实现在模块或者菜单级别上的安全性是或者全是或者全否的。你或者能访问某些东西，或者不能。而当我们深入到表单级别内部的时候事情就不那么简单了，因为在这种情况下，用户至少有三种途径可以获得对任何表单的访问权限：

- 只读--用户可以查看表单，但不能做任何改动；
- 只编辑--用户可以对任何已有的数据进行改动，但不能添加或者删除；
- 无限制--用户拥有全部的添加/编辑/删除权限。

安迪：我明白了。那么，只是检查一个分配给一个用户的角色是否拥有访问的权限已经不够了。我们必须要判定该角色拥有哪种类型的访问权限。

玛西亚：是的，而且字段级别的安全性也是如此。安全性管理器为表单和字段使用的视图将因此而同时

包括项目名称和其访问级别。就是这个访问级别来定义在表单上的那些控件是否应该向客户显示。

安迪：**Okay**。那么你说的意思是指表单级别的安全性事实上变成了要设置表单上各种控件的状态了吗？

玛西亚：就是这个意思！基本的途径是：当一个用户要求访问一个表单的时候，安全性管理器检查它的元数据，并告诉表单使用哪种安全性模式。它是通过设置表单上所有控件的 **cSecurityMode** 属性来做到这一点的。此外，如果一个用户获得的对一个特定表单的权限是只读，安全性管理器就把所有它包含的控件的 **cSecurityMode** 属性设置为“**V**”。如果用户有只编辑权限，那么这个属性就被设置为“**E**”。

安迪：如果在元数据表中没有为用户的角色和当前表单定义任何东西的话会怎么样？

玛西亚：安全性管理器让所有控件的 **cSecurityMode** 保持其默认值**--**就是一个空的字符串。安全性是根据例外来处理的，所以一个空的 **cSecurityMode** 就可以被翻译为“无限制”。记住，如上个月所述，安全性管理器通过返回为当前用户定义下的角色的所有数据来生成它的元数据。在这个案例里，它只需要判定对于当前表单是否存在着一个约束。这都是在安全性管理器的 **ChkForm()** 方法中处理的：

```
FUNCTION ChkForm( toForm )
LOCAL lcFormName, lcAccessMode, loObject
lcFormName = LOWER( toForm.Name )
lcAccessMode = ""
*** 保存传递进来的 DataSession
*** 如果我们要为在一个带有私有数据工作期的表单上的 Grid 设置安全性，
*** 我们就需要这个值了
This.FormDS = toForm.DataSessionID
SELECT lv_restricted_form
LOCATE FOR LOWER( ALLTRIM( form_nme ) ) == lcFormName
lcAccessMode = ALLTRIM(lv_restricted_form.access_ind)
*** 有效的访问模式是：
*** V = 只读
*** E = 只编辑
*** 如果 lv_restricted_form 表里面没有当前表单的记录，
*** 就意味着用户拥有无限制的访问
*** 如果表单的访问等级是受限制的，
*** 就要设置在表单上的控件们的属性
IF NOT EMPTY( lcAccessMode )
  toForm.SetAll( 'cSecurityMode', lcAccessMode )
```

```
ENDIF
```

安迪：打住！我们所有的控件都使用一个叫做 **cMode** 的自定义属性、并根据表单级别的 **cMode** 属性来设置它们自己的状态（要了解详细情况，请查看我们在 **2003** 年 **8** 月的专栏“**Whose Job is it, Anyway?**”）。为什么它们突然又需要一个 **cSecurityMode** 属性了？

玛西亚：原因是这里涉及到两个独立的问题。首先，我们需要根据表单的当前模式处理控件们的使能和禁止，这是通过在 **Refresh()** 中的代码读取 **cMode** 和 **cModeExp** 属性来处理的。然而，考虑到当我们真正用上安全性的时候也许会需要在运行时能够选择覆盖默认的行为，这时就需要 **cSecurityMode** 属性了。

安迪：请解释一下它。

玛西亚：可以想象一下：我们有一个添加记录按钮。很明显，该按钮所在的表单处于查看模式的时候，无论何时，该按钮都必须是 **enabled** 的，而当表单处于添加或者编辑模式的时候则应 **disabled**。

安迪：是的，看上去有点道理。

玛西亚：现在我们把两种安全性合起来。假定一个用户对于一个包含着添加记录按钮的表单拥有只读属性。由于该用户没有被允许添加新记录，因此该按钮必须被禁止，即使表单处于查看模式也一样。由于我们有两件要考虑的事情，所以我们需要两个参数。

安迪：我知道默认的行为是根据按钮自己的 **cMode** 属性与表单的 **cMode** 属性比较的结果简单的来使能或者禁止这个按钮，就象是这样：

```
LOCAL lcExpr
lcExp = This.cModeExp
IF NOT EMPTY( lcExp )
  &lcExp
ENDIF
This.Enabled = Thisform.cMode $ This.cMode
```

但是在你的例子中，我们难道需要能够覆盖这种默认的行为吗？

玛西亚：是的。当处理安全性的时候，在允许默认的行为发生前，我们必须先检查在 **cSecurityMode** 属

性里面有什么东西。记住，这是一个添加记录按钮，所以这个按钮唯一被使能的时候就是当用户被允许添加新记录的时候。这就意味着该用户必须拥有无限制访问权限，而这个无限制权限就意味着这个 **cSecurityMode** 是空的。所以，我们将 **Refresh()** 方法修改如下：

```
WITH This
  *** Check for security setting
  IF NOT EMPTY( .cSecurityMode )
   *** Ensure this control is always disabled
   .cModeExp = [This.cMode = 'NONE']
  ENDIF
 ENDWITH
DODEFAULT()
```

安迪：我还剩下一个问题。实际情况下，我们要怎样调用所有这些安全性管理器相关的内容？

玛西亚：这是个好问题，而答案也许会让你惊讶。它是从表单基类的 **Show()** 方法代码中被调用的，而该表单只是简单的把自己丢给安全性管理器：

```
IF VARTYPE( oSecurityMgr ) = [O]
  oSecurityMgr.ChkForm( This )
ENDIF
```

安迪：这的确让我惊讶！那么我就要问为什么是从 **Show()** 而不是从 **Init()** 中调用？

玛西亚：因为只有当 **Init()** 完成并且返回 **.T.** 的时候表单才真正的存在，所以表单在自己的 **Init()** 中是不能把对自己的引用传递给任何其它对象的。

安迪：这么想的话的确是有道理。毕竟，如果 **Init()** 返回了 **.F.**，表单就不存在了，那么任何外部对它的引用就都落空了。我注意到我们还没有看过在控件 **Refresh()** 方法中的真正代码。

玛西亚：好的，我们已经谈了相当多的关于表单级别实现的问题，但我们仍然需要对付字段级别的问题。由于需要对控件的 **Refresh()** 代码做很多的改动，所以我们讲到这部分的时候就直接展示这些代码最终的修改结果了。

安迪：那么字段级别的问题是怎样的？

玛西亚：好，最大的问题就是我们不再仅仅讨论一个表单是否使能或者禁止了。现在我们需要决定它对用户是否可见。

安迪：哦，但我很讨厌那种控件在表单上一会儿出现一会儿又消失的事情。

玛西亚：一般说来我同意你的观点。不过，在这个案例里情况有所不同，因为控件并不真正一会儿出现一会儿又消失。对于一个指定的角色，在所有的表单上它们的状态都是一致的。

安迪：再说一遍？

玛西亚：考虑一下一个总是“敏感”的字段--例如薪水。对于这个字段。财务经理可能拥有无限制的访问权限。但是，一个部门领导也许拥有只读的权限，而一个办事员则根本无法看到它。

安迪：是的，这个我能理解。让我疑惑的是“在所有表单上”这个部分。

玛西亚：啊，我这么设置是为了让字段的访问权限仅仅基于字段的名称来决定，而不考虑表的名称或者表单的名称。这对我来说特别容易理解，因为如果一个指定的用户角色没有对于在个人数据表单里面的一个名为“薪水”的字段的访问权限，那么，在其它的表单里，该角色也未必会有对这个字段的访问权限。

安迪：这个理由足够了，并且我发现这个假设竟然还戏剧性的减少了需要设置的数量。只是很难提出象你设想那样的现实中的案例：字段有着同样的名字，但是来自不同的来源，它们都是受限的，但却又要通过不同的途径来处理。

玛西亚：我也提不出来另一个了，所以那就是我要这么做的原因。另一个大问题是与 **Grid** 的合成。因为一个 **Grid** 有多个列，所以我们需要给 **Grid** 的基类增加一个数组属性，以给 **Grid** 中的每个列单独分配一个安全性等级。

安迪：当然了--因为 **Grid** 有多个列，而每个列都有着自己的 **ControlSource**，所以必须对它们逐个进行处理。但为什么只说起 **Grids**？其它那些多列的控件--比如 **List** 和 **Combo**--怎么办？

玛西亚：它们是参照控件，并且通常是由某些形式的 **Selection** 进程生成的。如果你需要从一个列表中排除一项，就不要把它包含在这个列表的数据源中。

安迪：好的，我知道了。那么我们应该怎样处理受限的字段呢？

玛西亚：安全性管理器的 **ChkForm()**所做的最后一件事情，是将一个对表单的引用传递给它的 **SetRestrictedFields()** 方法。这是一个递归方法，它会钻入表单上任何容器的内部，直到它找到一个拥有 **ControlSource** 属性的 **Grid** 或者控件为止。

安迪：打住，标签（**label**）的情况如何？如果你要让某个文本控件不可见，那么任何与它相关的标签都不应该还留在表单上。

玛西亚：好想法。幸运的是，我也已经想到这一点了。我给所有的文本控件添加了一个名为 **cLableName** 的属性，并给控件的 **Refresh** 方法添加了代码以当需要的时候删除任何相关的标签。你可以在 **textbox** 的 **Refresh()** 方法中看到这些代码（在本文稍后部分展示）。

安迪：那么实际上 **SetRestrictedFields()** 方法是如何工作的呢？

玛西亚：最简单的案例是处理所有拥有一个 **ControlSource** 属性的控件。

```
lcField = ''
*** 首先检查它是否拥有一个 cSecurityMode 属性
IF PEMSTATUS( toObject, 'cSecurityMode', 5 )
  IF PEMSTATUS( toObject, 'ControlSource', 5 )
    IF NOT EMPTY( toObject.ControlSource )
      lcField = LOWER(ALLTRIM(CHRTRAN(JUSTEXT( ;
              toObject.ControlSource ), ')', '' ) ) )
    ENDIF
  ENDIF
  *** 如果取得了 lcField，再对它进行检查
  IF NOT EMPTY( lcField )
    lcAccessMode = This.GetFieldRestrictions( lcField )
    IF NOT EMPTY( lcAccessMode )
      toObject.cSecurityMode = lcAccessMode
    ENDIF
  ENDIF
ENDIF
```

安迪：我明白了--你在取出字段的名称，然后通过调用 **GetFieldRestrictions()** 方法来返回任何与之相关的约束。那么，返回空的字符串就是“无限制”吗？

玛西亚：是的。象我告诉过你的那样，我们的原则是处理例外。只有存在一个约束的时候才是唯一需要我们※※※※※※。现在，在这个案例里，可能存在的约束如下：

- I--不可见（字段的内容不应向用户显示）；
- V--只读（用户没有被允许编辑字段的内容）；

安迪：啊，我看到控件的 **cSecurityMode** 又被设置了一遍。这很明显会覆盖从表单级别上继承来的任何设置。这样很整洁。但那些 **grid** 呢？

玛西亚：**Grid** 类有一个名为 **aSecurityMode** 的数组属性，初始化时会为 **grid** 中的每一列在数组中建立一行。在 **SetRestrictedFields()** 方法中，我们首先建立一个在 **Grid** 的 **RecordSource** 中字段的列表，然后检查它们是否有相关的约束。对于任何有一个约束的字段，我们将从 **Grid** 中查找与该字段相关的列的编号，如果某个列的 **ControlSource** 中包含该字段，就将字段的约束保存到数组中相关的行内去。

安迪：听起来相当麻烦。为什么你要同时做这些事情？

玛西亚：因为在 **Grid** 列的 **ControlSource** 中使用表达式是很平常的事情，尤其是在要显示参照值的时候，比如象这样：

```
(IIF(SEEK(Alias.field,[LookupTable],[LookupTag]), ;
    LookupTable.Description, [])
```

所以如果“**Alias.Field**”是一个约束字段，我们探测到它的唯一办法是在列的 **ControlSource** 上使用一个 **substring**。

安迪：真的，如果“**LookupTable.Description**”是一个约束字段的时候这种办法也有效（这是我在你提到 **ControlSource** 中包含字段名的时候想到的头一个念头）。

玛西亚：是的，它两种情况都包含到了。无论如何，结果是：最后，对于表单中的每一个控件（包括可能存在的任何 **grid** 中的每一列），我们都会取得一个空字符串或者一个有效的约束。这些代码用在 **Refresh()** 里面。

安迪：对于 **grid** 和普通控件，代码都是一样的吗？

玛西亚：不，**gird** 有一些不适用于普通控件的额外检查。那么，让我们先来处理简单的案例，这里是一个文本框的 **refresh** 代码：

```
LOCAL lcExp, loLabel
WITH This
*** 应用安全性设置
IF .cSecurityMode == 'I'
  *** 确保控件不可见
  .Visible = .F.
  *** 如果有一个相关的标签，
  *** 确保该标签也不可见
  IF NOT EMPTY( .cLabelName )
    loLabel = EVAL( 'This.Parent.' + .cLabelName )
    loLabel.Visible = .F.
  ENDIF
ELSE
  IF .cSecurityMode == 'V'
    .cModeExp = [This.cMode = 'NONE']
  ENDIF
ENDIF
IF .Visible
  *** 首先检查在 cModeExp 中是否有什么要运算的东西，
  *** 并使用它来设置 cMode
  lcExp = .cModeExp
  IF NOT EMPTY( lcExp )
    &lcExp
  ENDIF
  *** 现在检查模式
  IF .cMode == 'ALL'
    *** 在所有表单模式中可用
    .Enabled = .T.
  ELSE
    IF .cMode == 'NONE'
```

```
        *** 从不可用
        .Enabled = .F.
      ELSE
        *** 根据表单的模式决定是否可用
        .Enabled = ThisForm.cMode $ .cMode
        ENDIF
      ENDIF
    ENDIF
  ENDWITH
```

安迪：这看起来相当直观。通过首先检查 **cSecurityMode** 设置并覆盖正常的 **cMode** 值，你为所有可能的情况都做好了准备。那么 **Grid** 如何？

玛西亚：唯一真正的不同，是我们需要在应用了安全性模式以后要确保至少有一个可见的列，否则最后我们得到的可能是一个所有列都不可见的 **grid**，这样的话看上去就太愚蠢了。下面为 **grid** 基类的 **Refresh()** 代码做了改动：

```
LOCAL lcExp, lnColumn, lnVisibleCount, loLabel
*** 只对可见的列做出调整
WITH This
lnVisibleCount = .ColumnCount
*** 在列的层次上应用安全性设置
FOR lnColumn = 1 TO .ColumnCount
  IF .aSecurityMode[ lnColumn ] == 'I'
    *** 确保控件不可见
    .Columns[ lnColumn ].Visible = .F.
    lnVisibleCount = lnVisibleCount - 1
  ELSE
    IF .aSecurityMode[ lnColumn ] == 'V'
      .Columns[ lnColumn ].ReadOnly = .T.
    ENDIF
  ENDIF
ENDFOR
*** 检查是否遗漏了什么事情
IF lnVisibleCount = 0
  .Visible = .F.
```

```
        IF NOT EMPTY( .cLabelName )
          loLabel = EVALUATE( 'This.Parent.' + .cLabelName )
          loLabel.Visible = .F.
        ENDIF
      ELSE
        *** 检查是否需要设置控件的 cMode
        lcExp = .cModeExp
        IF NOT EMPTY( lcExp )
          &lcExp
        ENDIF
          *** 首先检查 all 或者 none
          IF .cMode == 'ALL'
            .Enabled = .T.
          ELSE
            IF .cMode == 'NONE'
              .Enabled = .F.
            ELSE
              .Enabled = Thisform.cMode $ .cMode
            ENDIF
          ENDIF
        ENDIF
      ENDWITH
```

安迪：我们有所有这些内容都用上的示例吗？

玛西亚：当然了。附带的下载文件中包含一个简单有效的 **demo** 以及全部的源代码。只要运行 **SHOWSAMPLE.EXE** 并选择一个用户。项目里已经有了三个用户，并为它们分配了不同的角色。根据你选择了哪个用户，表单将以表 **1** 中那样不同的形式显示。

表 **1**、简单用户和它们基于角色的安全性设置

用户	角色	表单访问	字段访问	字段访问
login_user.dbf	**login_role.dbf**	**role_form.dbf (添加/编辑按钮)**	**role_field.dbf (客户编号)**	**role_field.dbf （电话号码）**
Andy Kramek	超级用户	无限制	无限制	无限制
Marcia Akins	编辑用户	仅编辑	仅查看	仅编辑
Justin Trouble	查看用户	只读	不可见	不可见

安迪：我看到了。这非常酷！现在我们所需要只是一些维护表单以设置各个表--不过我们将把这作为练习留给读者了。

下载：**411KITBOX.ZIP**

半手工的生成器—更简单，更好

原著: Dragan Nedeljkovich

翻译: CY

生成器是个强大的工具，但是有时它并没有完全按照你想要的来做。或许你想编写你自己的生成器，但是发现这对你来说是件大事，你需要一个 GUI，在 Builder.dbf 里注册，为最终用户做说明...，算了，忘了它吧。Dragan Nedeljkovich 为你展示了如何编写你自己的简单、瘦身、半手工的生成器。

VFP 帮助文件有下列内容讲述了关于编写自制生成器的必要步骤：

1. 编写一个可提供接口并可修改选定的控件的应用。
2. 保存应用到向导文件夹。
3. 打开 Builder.dbf 表并追加一个新刻录。
4. 输入 Name, Descript, Type, 和 Program 字段。

按照我的经验，除了“可修改”部分以外都不是必需的。那么第 2-4 步骤是什么目的？它是为了在 Builder.dbf 里注册生成器，使得当你在编辑可视类或表单时右击选定控件时会出现。我不想这样麻烦，因为这次是完全不必要的。
可是，如果生成器没有注册那我该怎么运行它？进入命令窗口里。我是从这里调用我的生成器的。或者，当某一天我变得特别懒时，我将会为我所想要使用的每个生成器在工具栏目里创建按钮，并从命令容窗口里运行工具栏。

它是如何工作的？

每个生成器的核心都是 ASELOBJ()函数，它返回一个指向选定对象的数组。稍后我将抛弃“(s)”并只引用数组的第一个元素。我们的第一个半手工生成器只有两行，但是却会给你带来很大的力量。这里是 curform.prg:

```
* curform.prg
ASELOBJ(aa,1)
RETURN aa[1]
```

什么样的生成器将会对这些对象引用进行处理？我是从命令窗口或是从命令窗口里调用的 PRG 或表单来运行

的。我是不是忘记了提及命令窗口的重要性？
试试这样，仅创建一个空的表单，然后在上面放置一对的标签。选定标签，然后在命令窗口里输入：

```
ox=curform()
```

（实际上，我在我的系统里已经有一个智能感知扩展，所以我只需要输入“ox”再加上空格，我将会在命令窗口里自动插入命令行。）
现在我已经有一个有效变量 ox，并且它指向你当前所编辑的对象的父件（指向父件是因为在 ASELOBJ()调用里的第二个参数）。我发现这很方便，于是我通常在页框、容器或表单上选定一个控件，然后当我在命令窗口里输入处理代码时，我将会得到对页框的指向，并且可以访问同属的控件。
我该如何访问表单的属性或表单上的其他的控件？通过使用 ox 变量，就象是一个完全盛开的、闪亮的生成器。我可以在命令窗口里简捷操作如下：

```
ox.caption="New caption for this form"
ox.label1.name="lblFirst"
ox.lblfirst.caption="The first"
ox.label2.fontbold=.t.
```

当然，我可以在 PEM 窗口（如同属性窗口）做所有的这些事。但是，半手工成器对我来说好象更快。虽然在属性窗口里列出的属性有助于你来寻找你所需要的属性，但是当你在命令窗口里输入时智能感知却会帮助你做到这些。并且你不需要滚动窗口，并且你不需要下拉任何组合框来查找你的成员控件（甚至不需要触动鼠标滚轮）。坦率的说，有许多属性我从未使用过，或者很少使用，我长久地保持 PEM 窗口为“仅非缺省属性”，于是我只见到我所感兴趣的。
那代码如何？欲编辑方法代码，你需要打开编辑窗口，下拉组合框并滚动可用的方法，或者在属性窗口里双击对应方法，这对我无效，因为空的方法代码在“非缺省”状态下的 PEM 窗口里是不可见的。取而代之的是，我是从命令窗口里编写方法代码的。让我们作个假设，我想对我的标签编辑 Click 方法，如下所示：

```
ox.lblfirst.writemethod("click","* add some code here")
```

现在我的方法已经是非缺省的，并且我也为自己在这里发送了注解。
创建一个完整的新的方法，如果没有生成器是很困难的。我将不得不在对话窗口里输入它的名，然后在代码窗口或属性窗口里查找，我的感觉是使用鼠标狂找。注意 WriteMethod()方法的第三个参数，它意味着“按需创建”。

```
ox.writemethod("NewUselessMethod","* whatever",.t.)
```

有时，我会先在小小的 PRG 文件里写出代码，然后再决定把它包含入作为方法：

```
ox.writemethod("SecondHandMethod",;
   filetostr("MyLittle.prg"),.t.)
```

注意你现在可能想为一个包含的对象加入新的方法，VFP 却并未留意，下次当你保存时它却可能没了。
或许我将会在 Foxtalk2.0 的今后期刊里展示更多的我的喜好和更复杂的半手工生成器。期间，你可以利用这时提供的示例来尝试不同的方法进行 VFP 开发，以使你自己从传统的开发环境里释放出来。

来自 VFP 开发团队的 TIPS

原著：微软 VFP 开发团队

翻译：LQL.NET

每个月,VFP 开发团队成员都会给你带来有趣的 VFP 技术 TIPS。这个月带给你的是 VFP9 中 XMLAdapter 的新特性：支持多层 XML 数据

VFP9 中 XMLAdapter 的新特性：支持多层 XML 数据

本月微软提供的支持多层 XML 数据的 TIPS，包括使用新的 PEMs：Nest(), NestedInto, NextSiblingTable, FirstNestedTable, 和 RespectNesting。例程中要用到 3 个 CURSOR：Foo, Foo1, 和 Foo2，它们是一对多对多关联。首先，我们生成嵌套的 XML 并显示，然后，我们从这个嵌套的 XML 中重建 CURSORs。

演示这个样例前，你可以把代码中的第一行改为 lUseNesting=.F.，运行一下看看没有 VFP9 的新特性是怎么工作的，然后再把它改回 lUseNesting=.T.，看看在新特性下是怎么工作的。

```
lUseNesting=.T.
CLEAR
CLOSE DATABASES ALL
LOCAL lcSafety
lcSafety=SET("SAFETY")
SET SAFETY OFF
CREATE CURSOR foo (f1 I)
INSERT INTO foo VALUES (1)
INSERT INTO foo VALUES (3)
BROWSE FONT "Courier New",14
CREATE CURSOR foo1 (f1 I, f11 I, f12 c(10))
INDEX ON f1 TAG f1
INSERT INTO foo1 VALUES (1,11,"foo1.1.11")
INSERT INTO foo1 VALUES (1,12,"foo1.1.12")
INSERT INTO foo1 VALUES (3,33,"foo1.3.33")
INSERT INTO foo1 VALUES (3,34,"foo1.3.34")
```

```
BROWSE FONT "Courier New",14
SELECT foo
SET RELATION TO f1 INTO foo1
SET
CREATE CURSOR foo2 (f11 I,f12 c(10))
INDEX ON f11 TAG f11
INSERT INTO foo2 VALUES (11,"foo2.11.1")
INSERT INTO foo2 VALUES (12,"foo2.12.1")
INSERT INTO foo2 VALUES (33,"foo2.33.1")
INSERT INTO foo2 VALUES (34,"foo2.34.1")
INSERT INTO foo2 VALUES (11,"foo2.11.2")
INSERT INTO foo2 VALUES (12,"foo2.12.2")
INSERT INTO foo2 VALUES (33,"foo2.33.2")
INSERT INTO foo2 VALUES (34,"foo2.34.2")
BROWSE FONT "Courier New",14
SELECT foo1
SET RELATION TO f11 INTO foo2
LOCAL oXA as XMLAdapter, oXT as XMLTable
oXA=CREATEOBJECT("XMLAdapter")
IF !lUseNesting
   oXT=oXA.AddTableSchema("foo")
   oXT=oXA.AddTableSchema("foo1")
   oXT=oXA.AddTableSchema("foo2")
ELSE
   oXA.RespectNesting=.T.
   oXT=oXA.AddTableSchema("foo")
   oXT.Nest(oXA.AddTableSchema("foo1"))
   oXT.FirstNestedTable.Nest(oXA.AddTableSchema("foo2"))
ENDIF
oXa.ToXML("hierxml.xml",,.T.)
* Let's show XML now.
loApp=CREATEOBJECT("internetexplorer.application")
loApp.Visible=.t.
lcFile=FULLPATH("hierxml.xml")
loApp.Navigate(lcFile)
```

```
IF !lUseNesting
   RETURN
ENDIF
* Now, we'll close everything and go backwards
* to recreate tables
CLOSE TABLES all
WAIT WINDOW "Let's close all cursors and load XML"
oXA.LoadXML("hierxml.xml",.T.)
FOR EACH oXT IN oXA.Tables
   oXT.ToCursor()
ENDFOR
FOR EACH oXT IN oXA.Tables
   ?oXT.Alias
   IF NOT ISNULL(oXT.NestedInto)
      ??" nested into",oXT.NestedInto.Alias
   ENDIF
NEXT
* Set relations
SELECT foo2
INDEX ON f11 TAG f11
BROWSE FONT "Courier New",14 NOWAIT
SELECT foo1
SET RELATION TO f11 INTO foo2
INDEX ON f1 TAG f1
BROWSE FONT "Courier New",14 NOWAIT
SELECT foo
SET RELATION TO f1 INTO foo1
GO TOP
BROWSE FONT "Courier New",14 NOWAIT
SET SAFETY &lcSafety
```

下载：411TEAMTIPS.ZIP

玩转 MSDE （一）

Fbilo

MSDE 是 Microsoft SQLServer 2000 Desktop Engine 的缩写，翻译过来就是“微软 SQL Server 2000 桌面引擎”。对于用户数量不多、资金有限的中小型企业来说，用 MSDE 作为他们的数据存储是最合适不过的了。而在这个领域，尚没有任何其它软件能与 MSDE 竞争。现在，FBILO 将通过本系列文章教你如何玩转这个优秀的数据引擎。

MSDE 介绍

什么是 MSDE？MSDE 是 Microsoft SQLServer 2000 Desktop Engine 的缩写，翻译过来就是“微软 SQL Server 2000 桌面引擎”。咬文嚼字的理解一下，“桌面”两字说明 MSDE 适用于那些用户数量不多、但又对数据仓库的稳定性、安全性有较高要求的用户，因此，其适用范围可以说是介于 VFP DBC 、MDB 等文件型数据库与 SQL Server、Oracle 等大型数据库之间。“引擎”则说明 MSDE 的实质就是一个 SQL Server 数据引擎，与 SQL Server 相比，它缺少了企业管理器、事件探察器、查询分析器等用户界面，并且受到了一定的限制。

虽然如此，但是，最重要的是，MSDE 是完全免费的！

你可以从微软网站上下载，也可以从 OFFICE 2000/XP、MS Visio、MS Visual Studio 的光盘上找到它，你可以自由的使用它，也可以把它嵌入到你的应用程序内一起发布给你的客户，而这一切都是免费的。

对于用户数量不多、资金有限的中小型企业来说，用 MSDE 作为他们的数据存储是最合适不过的了。而在这个领域，尚没有任何其它软件能与 MSDE 竞争。

兼容性

既然 MSDE 使用的是 SQL Server 的数据引擎，那么它们的数据格式就是完全兼容的。你可以将一个 SQL Server MDF 数据库附加给 MSDE，反过来也没问题。而通常我们开发的时候，都是在 SQL Server 下面通过企业管理器来开发数据库，然后在发布时再从 SQL Server 上将准备发布的数据库分离

下来，添加到安装包里面去。此外，MSDE 也可以通过作为 InstallShield 项目中的合并模块嵌入到安装包中间去。这样一来，就可以一次完成你的应用程序的安装了。

要使用 MSDE 2000，必须已安装下列操作系统之一：

- Windows Server 2003 Standard Edition、Windows Server 2003 Enterprise Edition、Windows Server 2003 Datacenter Edition。
- Windows 2000 Server、Windows 2000 Advanced Server、Windows 2000 Datacenter Server。
- Windows NT Server 4.0（SP5 或更高版本）；Windows NT Server 4.0 Enterprise Edition（SP5 或更高版本）；Windows NT Workstation 4.0（SP5 或更高版本）。
- Windows XP Professional Edition、Windows XP Home Edition。
- Windows 2000 Professional。
- Windows Millennium Edition。
- Windows 98。如果计算机没有网卡，则要求 Windows 98 第二版。

硬件方面，MSDE 要求奔腾 166 以上、32M 内存、44MB 的硬盘。同时，MSDE 也支持双处理器的小型服务器。

限制

虽然 MSDE 是免费的，对微软对它的性能进行了限制。如果限制的太厉害，不适合我们客户的需要，那么我们可能也不得不放弃 MSDE 了。那么这些限制到底是怎么样的呢？

关于 MSDE 的用户数量限制，是最含糊不清的问题。有的说只能同时有 5 个客户或者连接，有的说可以有 25 个，还有人说，经过测试，100 个客户也没问题，只是速度稍微慢了点。造成这种混乱情况的原因，是微软本身就在这个问题上就说的含糊不清，玩了猫腻——毕竟，如果太多的人使用免费的 MSDE 的话，谁来买昂贵的 SQL Server？微软开发 MSDE 的目标是占领文件数据库与大型数据库中间的那块市场，可不是让它去抢 SQL Server 的市场，那不是搬起石头砸自己的脚吗？

那么，事实到底是怎样的呢？按照微软的《认识 SQL Server 2000 工作负载控制器》白皮书的说法是，MSDE 和 SQL Server 2000 个人版一样，使用了一种叫做“工作负载控制器”的技术，当 MSDE 的一个数据库引擎实例接收到超过 8 个（MSDE 主页的产品介绍上说的是 5 个，白皮书中又说是 8 个，真是乱啊！）并发的活动批处理操作的时候，即使这 8 个操作是在使用不同的数据库，工作负载控制器也将被激活，它会对每个要求进行逻辑读/写数据的连接延迟“几毫秒”的反应时间。

这里的“8 个并发活动批处理操作”并不代表 MSDE 只能支持 8 个并发连接——同时连接到一个

MSDE 实例的活动连接可以有 32,767 个，只有那些当前正在向 MSDE 发送 T-SQL 批处理命令的连接才被计入并发活动操作内。但是，当工作负载控制器被激活以后，受到性能降低影响的却不只是那 8 个并发活动操作所在的连接而已，在当前实例上的所有连接都会受到影响，只要它们试图去逻辑读/或者写数据，就会受到延迟。

这里说的“逻辑读/写数据”讲的是 SQL Server 数据引擎的一种缓冲机制，SQL Server 从物理的数据页上读取了一条或多条记录以后，不会立刻释放它，而是将它保存在缓冲区中。当一个 T-SQL 语句要使用一些数据的时候，数据库引擎会先从缓冲区中查找这些数据，如果没有，则再去找物理文件。这种向缓冲区读/写数据的办法就叫做逻辑读、或者逻辑写。

“操作”指的是客户端应用程序通过连接发送给数据库引擎的 T-SQL 批处理。不幸的是，建立连接并登录、关闭连接也要列入操作的名单内。因为，当客户端应用与数据库引擎建立连接并登录一个用户的时候，数据库引擎需要校验这个用户的用户 ID 和密码，并在内部为管理这个连接分配内部数据结构；而在关闭连接时，数据库引擎会为这个连接放弃所有未决的事务，并释放管理连接的内部数据结构；所以，这些操作无法避免的要用到 T-SQL 语句对数据进行逻辑的读写。这样一来，建立和断开连接的操作就都加入到可能激活工作负载控制器的操作的黑名单中了。

但是，存储在 SQL Server 数据库中的内部对象——视图、触发器、存储过程、用户自定义函数——却不会对工作负载控制器产生影响，因为对它来说，这些对象并没有跨越连接，它们纯粹是在数据库引擎的内部执行的。所以，尤其在你开发基于 MSDE 的应用程序的时候，尽量多用它们！

在实际环境中，MSDE 可以同时对上千个连接进行服务，那些连接到 MSDE 的客户端应用程序大多数时间里其实都处于一种非活动状态，等待着应用程序生成和发送一个新的批处理。只有很小一部分的连接会在同一时间点上传送命令。

> “大多数用户都在做着不与数据库引擎进行交互的事情，比如在一个网页中上下滚动、输入新的数据、或者给他们的主管回电话等等。只有少量用户正在执行一个象是按下“输入”按钮给数据库引擎的实例发送一个命令之类的操作。”
>
> 引用自《认识 SQL Server 2000 工作负载控制器》白皮书

因此，对 MSDE 的性能限制远没有想象中的那么严厉，对于中小型企业的案例大可以放心使用。

对于 MSDE 的另一个限制是，由于 MSDE 被设计为以后台方式运行，因此，它是没有用户界面（UI）的。这就是说，SQL Server 2000 中我们最常用的企业管理器、查询分析器、事件探察器、客户端工具等等工具都没有了，管理起来就有些不方便了——在 Myf1 上，流行的办法是采用命令行形式的 oSql 工具或者自己写程序访问 SQL DMO 来操作。其实还有两种少见一些的办法，只要客户端安装了 Access2000/XP 或者 InterDev，也可以用它们来完成大部分的操作。事实上，Access 可以象操作一个自己的 MDB 一样操作 MSDE 的数据库。

MSDE 2000 最多可在单台计算机上同时支持 16 个数据库服务器实例。在一个实例上，最多可以有

32,767 个数据库，最多可为每个数据库提供 2 GB 存储空间。这一限制条件是以数据库而非服务器为单位的，即每个数据库最大不能超过 2GB。

版本

虽然我们说的 MSDE 通常指的就是 MSDE 2000，但它并非就是从 2000 开始的。最早的 MSDE 出现于 1999 年，那时候的 SQL Server 还是 7.0 版，因此，似乎那个版本的 MSDE 应该叫做 MSDE 7.0 才对。

最初的 MSDE 还没有安装包，只能通过命令行的方式执行 MSDEx86.exe 文件来手工安装。而命令行的参数则隐藏在一个名为 UNATTEND.ISS 的文件中，这个文件其实是一个 INI 文件，你需要手工去改动它以满足自己的需要。更糟糕的是，安装命令采用的是一种“安静”的方式运行的，也就是说，即使安装出错，也得不到任何的提示。

SQL Server 2000 发布以后，在 SQL Server 2000、OFFICE 2000/XP/2003、Visio 2000、Visual Studio 6.0/.NET、Visual FoxPro 7.0 的产品光盘上都带有一个新版本的 MSDE 2000，这个版本的 MSDE 终于有了会调用 Windows Installer 的安装包。但是，你别指望 M$对于免费的东西会太热心，实际使用中发现，这个版本的 MSDE 安装包有着不少的问题，在 Win98 下安装的时候，经常会出现不能安装、安装后 SA 密码为空、无法指定实例名之类的问题。

目前最新的版本，是 SQL Server 2000 SP3 发布以后出现的 MSDE 2000 Release A 和 SQL Server 2000 Desktop engine SP3a，这才终于解决了以上的问题。新的版本还自带 MDAC 2.7 SP1a，除了 Windows XP 系统以外，如果当前系统中的 MDAC 版本小于 2.7 SP1a，则安装 Release A 时也会为你自动升级 MDAC。

Release A 是单独安装版，它已经包含了 SP3 的内容，但它只能用于安装新的 MSDE 2000 实例，不能用于升级以前的 MSDE。所以，要升级已有的 MSDE 数据库引擎的实例的话，必须选用 SP3a。此外，与 SP3a 相比，Release A 还缺了合并模块。

MSDE 2000 Release A （简体中文版）下载地址：

http://www.microsoft.com/downloads/details.aspx?FamilyId=413744D1-A0BC-479F-BAFA-E4B278EB9147&displaylang=zh-cn

SQL Server 2000 DeskTop Engine SP3a （简体中文版）下载地址：

http://www.microsoft.com/downloads/details.aspx?FamilyId=90DCD52C-0488-4E46-AFBF-ACACE5369FA3&displaylang=zh-cn

需要注意的是，对于 MSDE 2000 Release A 的安装，如果客户的系统是 Windows Me，则必须从

上面的地址下载另一个专门的版本。

此外，由于 Release A 和 SP3a 发布于 2003 年 10 月 29 日，因此，它们还不具备对冲击波蠕虫之类病毒的抵抗性，最好还要安装来自 Microsoft TechNet 的最新热修复程序。

2004 年 9 月 4 日，微软在 TechED 大会上宣布 MSDE 的下一代产品 SQL Server 2005 Express Edition，这个产品将连同 SQL Server 2005，以及 VisualStudio2005 于年底正式推出。为了更符合.Net 技术，Express 版本与 MSDE 相比增加了 CLR(Common Language Runtime)以及 XML、XQuery 支持。而有了 CLR，开发者可以使用任何语言来开发。让我们拭目以待吧！

MSDE 的安装

我们安装 MSDE 通常有两种目的：为用户安装最终产品、和自己构建一个测试环境。这两种情况下的安装也不尽相同。

自己构建一个测试环境的时候，由于 MSDE 不能安装在已经安装了 SQL Server 的机器上，所以一般我们会用 VMWare 之类的工具建立一个或多个虚拟机，然后在虚拟机上直接运行 MSDE 的安装包来安装。

而为用户安装最终产品的时候，除了直接安装 MSDE 以外，我们还可能会选择另一种办法：将 MSDE 安装模块嵌入到最终产品的安装包中去。这种办法可以让用户只运行一次安装包就装完客户端应用程序和 MSDE 实例，对于大多数没有多少电脑知识的用户来说，这种办法当然更方便——当然，制作这样的安装包，对于程序员来说也比较复杂了。

由于后一种方法主要的工作是制作安装包，因此，我们现在先讲第一种办法，后一种方法将在后面的“将 MSDE 作为合并模块嵌入到安装包”部分中讲述。

安装前的注意事项

在安装 MSDE 2000 之前，还有一些与操作系统的设置有关的问题需要注意：

- 操作系统中应安装有 Microsoft Internet Explorer 5.0 或更高版本；
- 除 Windows 98 和 Window Millennium Edition 外，必须启用文件和打印机共享；

对于第二点，请采用以下方法确认：

- 在“控制面板”中，双击“网络连接”。

- 在“高级”菜单中，单击“高级设置”。
- 在“适配器和绑定”选项卡上，确定选中了“Microsoft 网络的文件和打印机共享”。

在 MSDE 2000 Release A 的 Readme 中，还提到 Windows XP 的本地安全策略“设备：未签名驱动程序的安装操作”和 Windows 2000 的本地安全策略“未签名非驱动程序的安装操作”两项不能被设为“禁止安装”，但事实上，这两项根本就不能被设为禁止，因此无需理会。

安装完 MSDE 以后，一般系统会要求你重新启动。可以采用以下办法来避免：在安装 MSDE 前，先打开控制面板——管理工具——服务，然后在服务列表里面停止以下服务：

- Microsoft 分布式事务处理协调器 (MS DTC)、Microsoft 搜索服务以及 MSSQLServerOLAPService 服务。
- Microsoft 组件服务、Microsoft 消息队列和 Microsoft COM 事务集成器。
- 包括“控制面板”在内的所有应用程序。（可选）

然后安装 MSDE，安装完毕以后，就可以使用 Net Start MSSQLServer 命令来不重启系统就启动 MSDE。如果不先停止这些服务安装 MSDE，那么安装完以后，如果不重新启动系统，这些服务就无法启动，因而也无法启动 MSDE 服务。

安装 MSDE 的参数设置

双击下载来的压缩包，它会提示你将程序解压到一个指定的目录。单击“确定”等待其解压缩完成后，进入该目录，默认直接双击 Setup.exe 就可执行安装，但是我们不建议这么做。

基于我们开发的要求，还有一些要特别考虑的问题：比如，指定安装实例名称、安全验证模式、SA 密码等等。下面的表 1 是用于设定各个选项的命令行参数。我们可以直接使用带下列参数的命令行来安装，也可以将参数写在一个 Setup.INI 文件中，指定命令行使用这个 INI 文件来安装。

表 1、常用 MSDE 安装命令 Setup.exe 的可选命令行参数

参数名称	可用值	说明
SAPWD	自定	指定 SA 帐号的密码，密码两端要加上引号。
INSTANCENAME	自定	指定实例的名称，名称两端要加上引号。
DISABLENETWORKPROTOCOLS	1—默认，不启用网络支持；	指定实例是否接受在其他计算机上运行的应用

	0—启用网络支持；	程序的网络连接。
SECURITYMODE	SQL—使用混合验证模式	默认为 Windows 验证，使用 SQL 后采用混合验证。
/Settings	自定	指定一个 setup.ini 文件
/L*v	自定	指定一个安装日志(log)

下面是各项参数的解释：

- INSTANCENAME——指定安装实例名称

客户的机器上可能已经安装了 MSDE，并且有了一个或多个服务器的实例，为了避免与可能已存在的实例名称相重复，我们需要在安装时能够指定实例的名称。还要记住的是：MSDE 最多只支持 16 个实例。示例：

```
D:\MSDE2000\>Setup INSTANCENAME = “MSDE_TEST”
```

- SECURITYMODE——指定数据库引擎实例采用哪种安全验证模式

虽然微软极力推荐使用它的 Windows 安全验证模式，但我们在编程工作中最常用的还是混合验证模式，因此，如果你决定采用混合验证的话，最好在安装 MSDE 时就先设置好。示例：

```
D:\MSDE2000\>Setup INSTANCENAME = “MSDE_TEST” SECURITYMODE=SQL
```

- SAPWD——指定 SA 密码

默认的安装方式下 SA 密码为空——众所周知，这会留下多大的安全隐患。那么，指定 SA 密码就是一件必须的工作了。也许你会想要删除默认的管理员帐号 SA 而自建另一个，但是 SQLServerAgent 是使用 SA 帐号的，所以你最好别这么做，否则的话，就无法使用计划任务功能了。示例：

```
D:\MSDE2000\>Setup SAPWD=”mypassword” INSTANCENAME = “MSDE_TEST” SECURITYMODE=SQL
```

- DISABLENETWORKPROTOCOLS——指定是否打开网络支持

MSDE 的意思是微软“桌面”数据引擎，顾名思义，它默认是为单机版用户安装的，不支持网络上的其它用户的应用程序连接到 MSDE。因此，如果你开发的是小型多用户程序的话，那么应该指定打开 MSDE 的网络支持。打开网络支持示例：

```
D:\MSDE2000\>Setup SAPWD=”mypassword” INSTANCENAME = “MSDE_TEST” DISABLENETWORKPROTOCOLS=0
```

- /L*v——指定是否记录安装日志

可以打开详细日志记录选项以备进行故障排查。例如，您可以提供一个 L*v 命令行参数和提供一个日志记录的文件名。例如：

```
D:\MSDE2000\>setup.exe /L*v C:\MSDE_setup.log
```

- /Settings——指定一个设置参数的 INI 文件

可以将前述参数写在一个 INI 文件中，用本参数指定 Setup.exe 使用该 INI 文件中的参数来执行安装。示例：

D:\MSDE2000\setup.exe /Settings D:\MSDE2000\setup.ini

使用命令行安装

打开开始菜单中的“运行...”对话框，输入：

```
C:\MSDERel\Setup.exe SAPWD="ABC" INSTANCENAME="MSDE_TEST"
DISABLENETWORKPROTOCOLS=0 SECURITYMODE=SQL
```

以上命令表示指定安装程序安装一个实例名为 MSDE_TEST、SA 密码为 ABC、使用混合验证模式、并打开网络支持的 MSDE 数据库引擎实例。单击确定后，按提示一步步安装。

使用 INI 文件安装

在解压开的下载文件目录下，会有一个几乎全空的 Setup.ini 文件，我们可以通过修改这个文件来指定参数。如图 1：

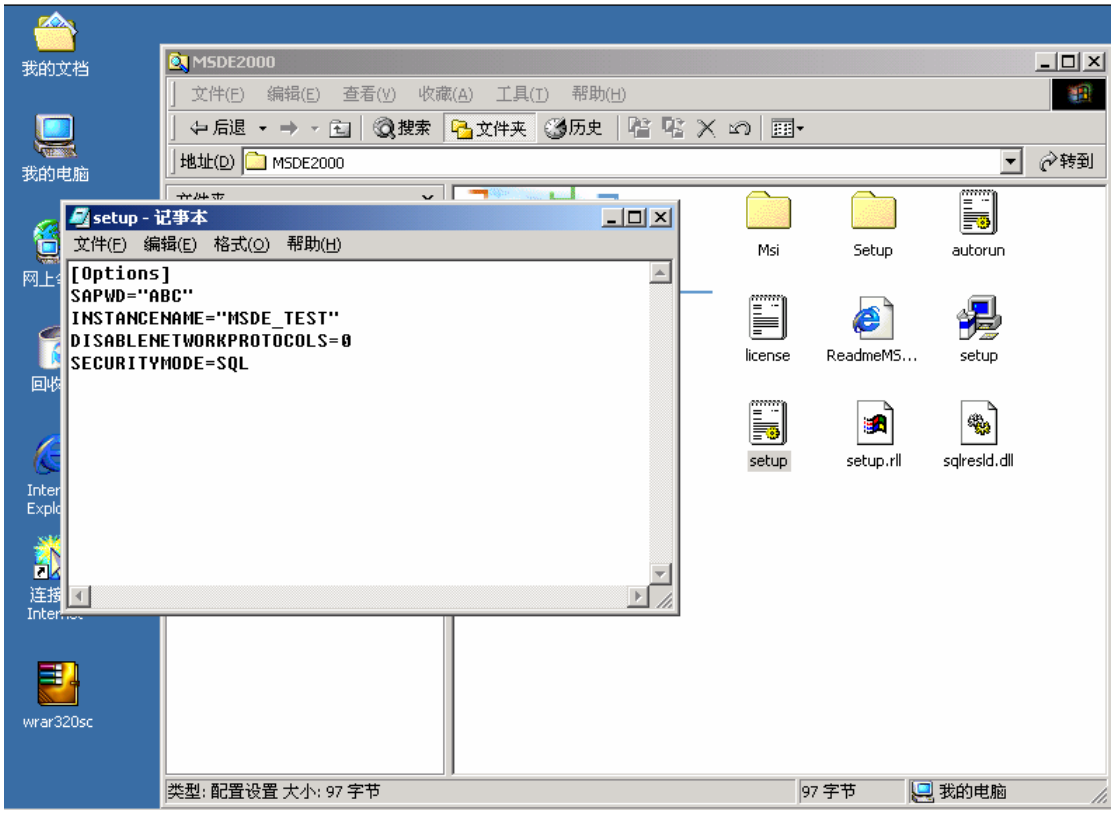

图 1、修改 Setup.ini 文件以指定安装参数

然后，从开始菜单中的“运行...”对话框中执行以下命令：

```
D:\MSDE2000\Setup.exe /Settings D:\MSDE2000\Setup.ini
```

单击确定后，按提示一步步安装。

安装日志

Return Code Meaning

1 Success

2 User cancelled

3 Unrecoverable errors

4 Install suspended waiting for reboot

启动 MSDE 数据库引擎

完成安装后，可能会提示您重新启动系统。如前面“安装前的注意事项”一节所述，如果你在安装前停止了一系列相关的服务，则可以不重启系统而通过执行 Net Start MSSQLServer 命令来启动 MSDE 数据库引擎。否则的话，则必须重启系统。

注意这个命令里的 MSSQLServer，它并非是一个参数常量，而是出现在系统服务里面的名字。如果你为 MSDE 或者 SQL Server 安装的是默认的实例，那么在系统服务里面出现的会是“MSSQLServer”；但是，如果你在安装时指定了自己的实例名称，那么，出现在系统服务里面的，将会是“MSSQL$NewName”这样的名称，后面的 NewName 就是你指定的实例名。例如，如果我在安装 MSDE 里指定了实例名为“MSDE_TEST”，那么系统服务里面出现的名称就会是“MSSQL$MSDE_TEST”，在这种情况下，启动服务的命令就应改为：

```
NET Start MSSQL$MSDE_TEST
```

在 MSDE 2000 Release A 的 Readme 中提到，在系统重新启动（或者完成安装但未请求重新启动）之后，应确保以下服务正在运行：

- MS DTC
- Microsoft 搜索
- MSSQLServer
- MSSQLServerOLAPService
- SQLServerAgent

尤其是 SQLServerAgent 服务，安装完成、并重启系统后，默认都是没有启动的，而且被设置为“手动”，需要手工在控制面板——管理工具——服务里面将它改成自动启动才行。

另外，其中提到 MSSQLServerOLAPService 服务有点莫名其妙，安装了 SQL Server 的机器是无法再安装 MSDE 的，所以，如果有这个服务存在，就说明当前机器上安装了 SQL Server，那么 MSDE 就不可能安装成功。而 MSDE 自己又是不带 MSSQLServerOLAPService 服务——所以这个服务根本就不可能出现！倒是还有个没提到的 MSSQLServerADHelp 存在，从名字上来看，它似乎是 Active Directory 服务的帮助，也是默认不启动、需要手动启动的。

检查 MSDE 安装是否成功

我们现在用 oSql 工具来检查一下是否安装成功、并且设置正确。请打开开始菜单中的“运行...”

对话框，输入“commmand”(Win98)或“cmd”(WinXP/2000)，然后按下回车打开 Dos 窗口。

现在，在窗口内输入：

```
oSql -L
```

L 参数能列出当前网络上的所有服务器列表。注意：L 必须是大写。执行的结果见图 2 所示：

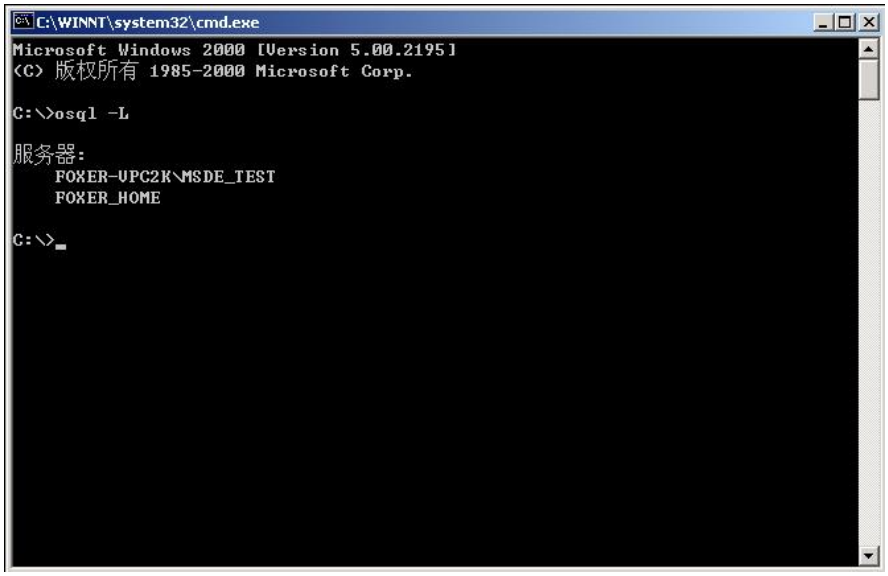

图 2、用 oSql 工具检查安装了的实例

从图中，我们可以看到，刚才我们安装的 MSDE_TEST 实例已经在服务中了，接下来我们使用以下格式的命令试着登录到服务器：

```
oSql -U uid -P pwd -S servername
```

参数解释：

-U 参数——指定输入用户名 uid；

-P 参数——指定输入登录密码 pwd；

-S 参数——指定输入服务器名称

这里的-U、-P、-S 都必须是大写。-P 参数可以先不写，执行后会提示你输入密码。

本来，在安装了 SQL Server 的情况下，是可以忽略 -S servername 参数的，微软 MSDE 的 Readme 里面也没有提到 MSDE 有什么特殊要求。但是经作者测试，在 MSDE 的情况下，忽略这个参数会出现“SQL Server 不存在或者访问被拒绝”的错误，无法登录到服务器，而且这里的 Servername 必须是带完整路径的名称才行，也就是要完全按照 oSql –L 命令列出的服务器列表中的样式写。免费的东西都是二娘养的，受点小气也是没办法的事啦！见下图：

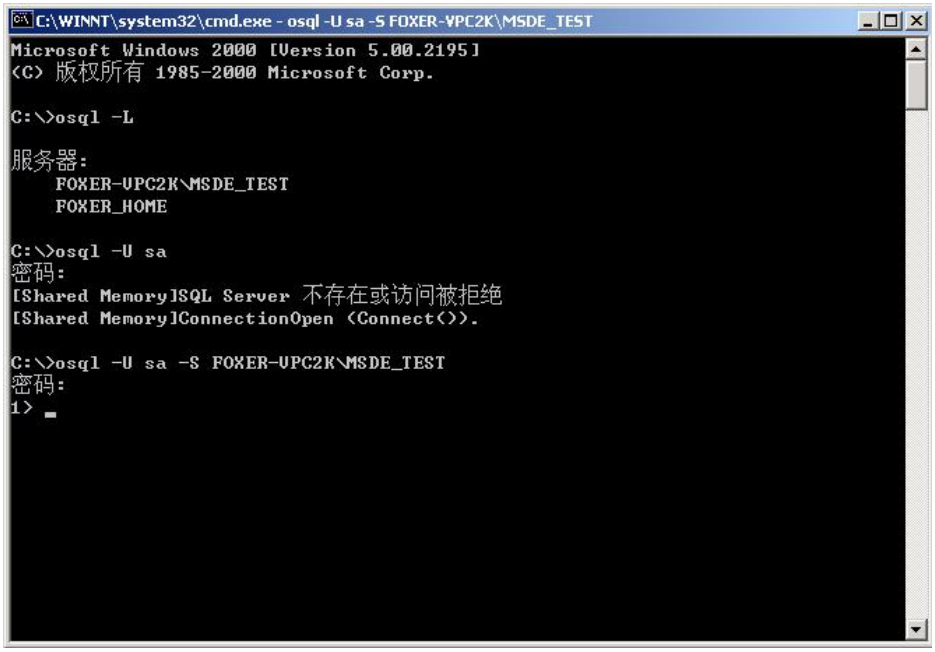

图 3、登录到 MSDE 的实例