E-BO 会议
Visual FoxPro 商业对象

作者：*Kevin McNeish*， *Oak Leaf* 企业, 有限公司
译者：*fbilo.*

内容

许多程序员开始认识到，他们需要把应用程序的逻辑放在商业对象上，而不是用户界面上，以实现从桌面应用转向 Internet 应用所需的缩放性　　但是，他们却不知道该如何去做！这次会议演示了对象建模的机制，并回答了如下一些问题：你应该为你的应用程序建立哪些商业对象？你该怎样决定哪个对象应该被赋予什么特定的责任？你该怎样建立可以访问不同的后台、并且能与各种前端协同工作的商业对象？在回答这些问题的过程众，这次会议将会帮助你理解设计合理的商业对象的结构体系。

你是一个 **21** 世纪的程序员吗？

你的编程风格是已经跨入 21 世纪了呢还是仍然停留在 FoxPro 2.x 的时代里？在 FoxPro 的时代里我们建立的是整体式应用程序　　应用程序的主要逻辑是放在用户界面里的　　在命令按钮的 Click 事件里、在文本框的 Valid 方法里面等等。在那时候，这种应用程序开发的途径应用的不错。

不过，一种新的技术在 1990 年露出了水面（客户/服务器数据库、Internet、以及最近出现的 Web Services）。为了保证能用上这些新技术而又不需要完全重写你的软件，很明显，需要一种开发软件的不同途径。

1995 年，Visual FoxPro 3.0 向 Fox 社群引入面对对象编程的概念，并提供了建立可重用的、基于部件的应用程序所需的工具，这样的应用程序可以很容易的实现那些软件开发的新技术。然而，大多数的程序员仍然采用 FoxPro 2.x 中的老办法来建立应用程序　　他们通过将应用程序的主要逻辑放在用户界面上来建立整体式的软件。尽管他们有目的的将他们的应用程序设计为最终能够访问客户/服务器的数据、或者能够通过 Internet 来访问，但是，当他们真正想要将程序升迁为使用这些技术的时候，往往发现，他们软件中的大部分都要重写。

那么，如果你不把应用程序逻辑放在用户界面上的话，要放在哪里呢？答案就是：商业对象。

用商业对象来摆脱困境

你找遍 Visual FoxPro 基础类也不会找到一个商业对象类。如果你想要使用商业对象，只有自己去建立、或者买一个内含有建好的商业对象类的应用程序框架。

> 注意: 这次会议的示例代码提供了一个简单的商业对象类，可以帮助你熟悉和理解基于部件的软件开发的概念。

从概念上来看，商业对象是表示一个人、位置、事件或者商业进程的一个较高层次的抽象对象。

通常，商业对象表示真实世界中的实体。考虑这一点的时候，可以想象一下，数据表也是这么回事。以一个销售点的应用程序为例，它包含发票、发票细节、付款和客户表　　这就是对真实世界的映射。在这些表中的数据反映了真实世界中实体的属性、或者特征。商业对象要比这个概念更进一步，它们不仅仅是映射那些实体的特征而已，还为它们的行为建立了模型。

一个发票商业对象将会包含所有对发票进行操作的逻辑（包括数据检索和操作）。一个客户商业对象将会包含所有对客户进行操作的逻辑，等等。

这种途径是怎样对你的软件产生帮助的？

你应该使用商业对象的四个重要原因

理由 4：建立和构想复杂的软件系统

当你的应用程序逻辑被四处散布在应用程序的各个角落中、并且深入的隐藏在用户界面控件中的时候，要理解程序整体的构思是很困难的，因为，你的应用程序逻辑是“藏在杂草里”的。

比较起来，当你把你的应用程序逻辑放在更高层次的商业对象中的时候，你可以以这样一种更能帮助你理解“商业进程的实际工作的复杂过程“的途径，来查看和操作这些对象。这个概念起初可能有些难以掌握，但是一旦你理解了它以后你就绝不会再走回头路！商业对象的封装，使得分析、设计和软件软件构造的工作的容易程度远远超过了面向过程的开发技术。

理由 3：正规化你的应用程序逻辑

Visual FoxPro 的开发人员们通常都能通过排除冗余而在正规化一个应用程序的数据上做得很好。不幸得是，这个正规化的概念没有被应用到应用程序逻辑上去　　尤其是在团队开发的环境中，你经常会看到同样的代码在一个应用程序中重复了许多次。

比较来看，当你使用商业对象的时候，在单个商业对象上重复建立两个任务相同的方法这种可能性几乎不存在。这就意味着最终你将会写更少的代码、调试更少的代码和维护更少的代码！

理由 2：帮助解决“代码放在哪里了？”的问题

当你把应用程序逻辑四处散布在你的应用程序的各个角落里的时候，你会花上大量的时间试图去找到代码。这可能是极其令人沮丧的。

比较起来，当你使用商业对象的时候，通常你立刻就可以找到代码所处的位置。例如，如果发票有什么问题，你完全可以确定它就在发票对象里。如果税率计算有什么问题，那么问题可能就在税率计算对象里，等等。

理由 1：升迁到 Internet 和客户/服务器数据的能力

正如我们前面已经提到的，商业对象给了你的应用程序可伸缩性。如果你把你的应用程序逻辑放在一个 Visual FoxPro 用户界面中，那么，界面换成了一个 Web 浏览器的时候怎么办？回答是，没有办法，因为你不能访问那些已经困在用户界面中了的代码。

比较起来，当你把你应用程序逻辑放在商业对象中的时候，你可以从你的 Visual FoxPro 用户界面上、也可以从 Internet 上访问这些对象。它之所以可能，是因为商业对象可以被编译为 COM 服务器，而 COM 服务器是可以被那些设计为知道怎样与 COM 对话的 Internet 产品（例如 ASP 和新的 ASP.NET）访问的。此外，象 West Wind Web Connection 和 Active FoxPro pages 这样的 Visual FoxPro Web 工具也可以“有机地”访问它们，而不需要把它们编译到一个 COM 服务器里面去。

此外，当所有的数据访问和操作都通过你的商业对象来发生的时候，它还让你的应用程序可以升迁到客户/服务器模式。当用户界面要求商业对象去检索或者保存数据的时候，用户界面根本不知道在后台的是哪种类型的数据（VFP vs. 客户/服务器）、或者数据放在什么地方。只有在商业对象上一个薄薄的数据层知道关于数据的信息。这个数据层可以很容易的切换，这样就让你的应用程序变得可以缩放数据。

怎样去实现它?

如果你有一个 Visual FoxPro 应用程序，想让它从整体式结构转换到基于部件的架构的话会怎么样？你要怎样去做到它？应用程序逻辑必须被从用户界面从提取出来放到商业对象中去。

最好选择你的应用程序的一部分来开始。通常，应用程序的某些部分需要首先通过 Internet 来访问。你可以建立商业对象，并开始将那些数据检索、数据操作、和商业逻辑从用户界面转移到商业对象的方法中去。

鉴别商业对象

你要怎样鉴别出那些存在于应用程序中的、应该为之建立商业对象的实体呢？这牵涉到面对对象的分析和设计。有许多不同的面对对象方法论来挑选它们。在 Oak Leaf 企业，我们使用 UML 语言以及 Rational Software 的统一过程（关于 UML 语言的更多信息请参见我在 FoxPro Advisor 99 年 1 月、2 月刊上的专栏）。不管你选用那种方法论，最终都会以对你的商业对象们的一个特征定义、责任分工和协作而结束。

当制作商业对象的时候，通常会给应用程序中的主要的每个表建立一个相应的商业对象。通常，给每个后台的表一个操作它的商业对象会是一种好架构，客户商业对象应该只操作客户记录而不管发票或者库存清单。一个订单商业对象应该只操作订单记录，等等。

建立一个设计良好的对象模型

意识到应该在应用程序中采用商业对象只是较为容易的部分。困难的是决定要建立哪些商业对象以及给它们分配哪些责任。对于怎样设计你的对象模型而言，参考手边能够找到的的产品 例如 Microsoft 的 Office 会是一个好主意。图 1 是在 VFP7 的对象浏览器中看到的 MicroSoft 的对象层次。

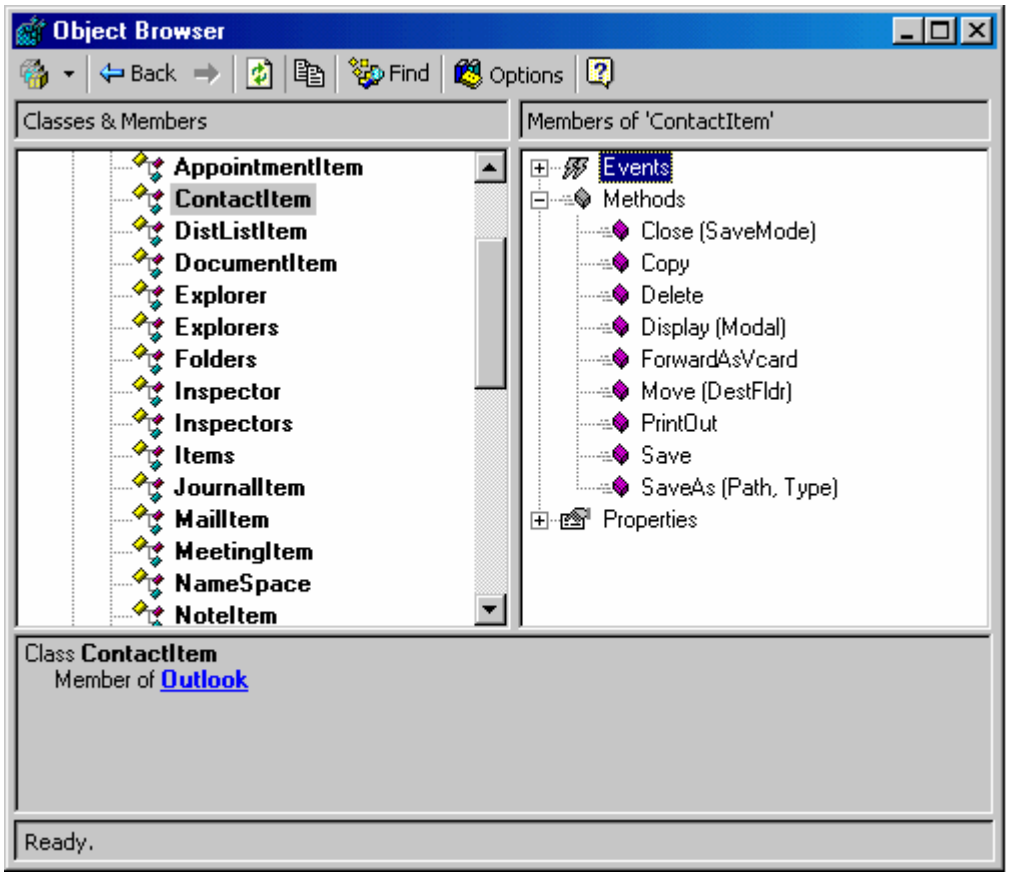


图 **1**。 **The Microsoft Outlook o** 对象模型提供了一个商业对象设计的好范例

要学习更多关于这些对象的属性、方法的内容，在 Visual FoxPro 7 中执行以下操作：

1. 从【工具】菜单中选择【**Object Browser**】（对象浏览器）。
2. 在对象浏览器中，单击【**Open Type Library**】（打开类型库）按钮。
3. 在打开对话框中选择【**COM Libraries**】（COM 库）页。
4. 从列表中选择【**Microsoft Outlook 9.0 Object Library**】，然后单击 OK。

在【Classes & Members】（类和成员）面板上选择不同的对象，然后花一点时间检查每个对象的那些方法。这会让你对“ 商业对象的方法是怎样被用来检索和操作数据、以及执行大量的商业逻辑 ”产生一个良好的映象。

建立一个商业对象的时候要注意，你正在建立的是一个你自己或者其它程序员未来将会使用的 API。

商业对象和用户界面

商业对象的定义不包含任何用户界面逻辑。它们只是显示一下消息或者与用户界面进行互动。这就让商业对象可以缩放到更大范围的平台上　　包括 Internet。如果你的商业对象包含着“需要某种特殊用户界面”的逻辑，这会在第一层上给你的灵活性造成障碍。如果你试图硬要让用户界面与一个进程内 DLL 进行互动，在运行时就会发生一个错误。

商业对象的结构

如下面图中所示，一个设计良好的商业对象是由多个部分组成的。这个示例架构包含：

- 商业对象使者（emissary）；
- 数据环境和数据访问对象；
- 商业规则；
- 商业对象实行者（只适用于分布式应用程序）；

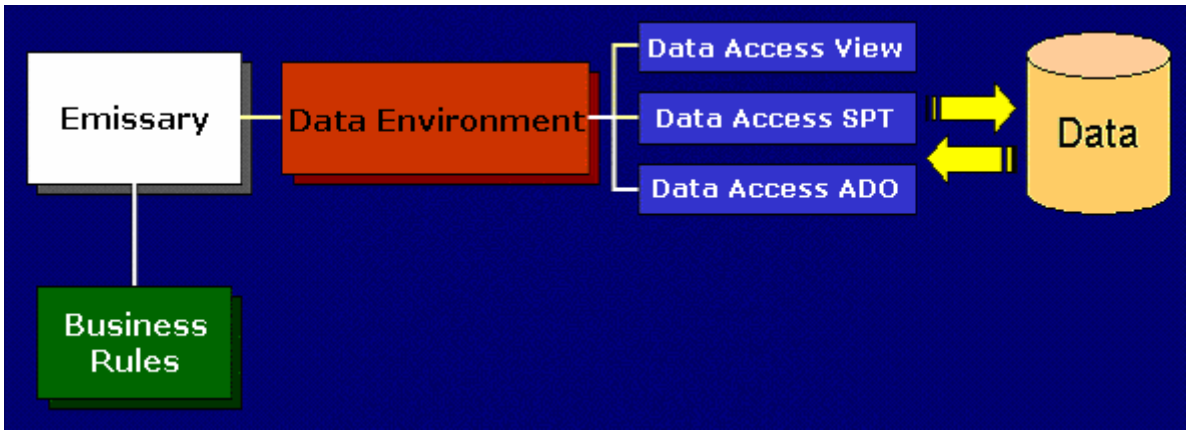


图 **2**：一个由多个对象组成的商业对象架构提供了可缩放性

商业对象使者（**Business Object Emissary**）

使者是商业对象结构中的主要元素。它是商业对象中与客户端代码直接通讯的部分。它包含者属于某个特定实体的应用程序逻辑。例如，一个发票商业对象包含着属于发票的逻辑；一个客户商业对象包含着属于客户的逻辑。

商业规则

商业对象的一个主要角色是去强制商业规则。商业规则对象负责为相关的商业对象维护和校验商业规则。商业规则在数据被发送到后台前对它们进行检查。

数据环境和数据访问对象

数据环境为商业对象指定数据访问。它为每个由商业对象实现的数据访问的实例包含一个数据访问对象。（It contains a data access object for each instance of data access implemented by the business object.）例如，如果一个商业对象需要把数据加载如三个不同的 Cursor 中，在数据环境中就将会有三个不同的数据访问对象。一个数据环境类应该在两个重要的方面改进 Visual FoxPro 的数据环境：

1. 允许数据环境在一个表单或者报表的范围外结束。
2. 允许你随便的加载本地或者远程数据，并在运行时能动态切换。

数据环境和数据工作期

数据工作期是由一个表单、表单集或者报表用来加载和操作数据的工作环境。每个数据工作期包含它自己的一套工作区。这些工作区包含打开的表、它们的索引和关系。寄宿在一个 Visual FoxPro 表单上的商业对象将 Cursor 加载入单个表单的数据工作期。例如，入图 3，ProjectsForm 包含三个不同的商业对象，它们将三个不同的视图加载进了同一个数据工作期。

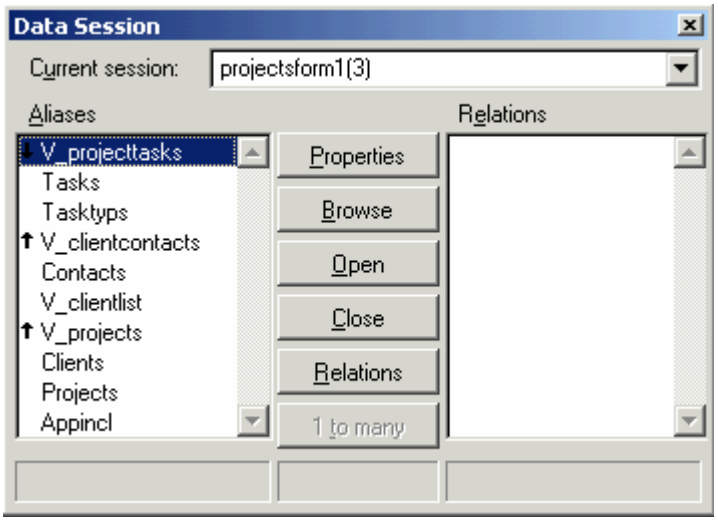


图 **3**：**Cursor** 可以被加载入单个数据工作期内

你可以通过挂起应用程序，然后在命令窗口中输入 **SET** 来打开一个表单的数据工作期窗口。

将界面控件与商业对象视图进行数据绑定

当某个商业对象已经将一个 Cursor 加载到表单的数据工作期了以后，就可以通过将 Cursor/字段的名称指定为某个用户界面控件的 ControlSource 来将该控件绑定到这个 Cursor 了。

例如，如图 4 所示，在 Projects 表单上的 Project Name 文本框已经被绑定到由 Projects 商业对象加载的 v_Projects 视图上了。

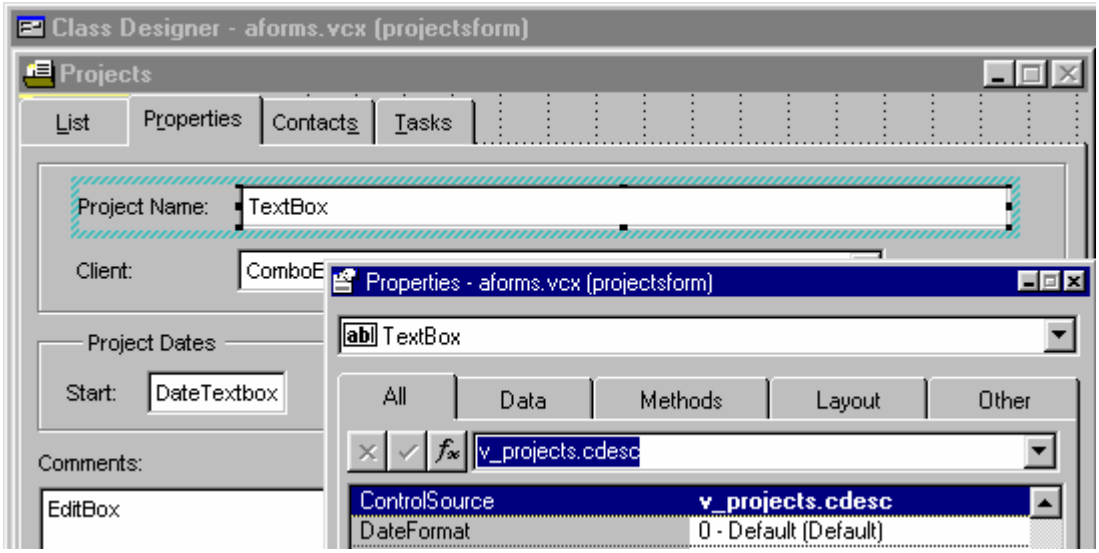


图 **4**：一个文本框可以绑定到由某个商业对象加载的一个 **Cursor** 的字段上

加载到某个表单数据工作期中的 Cursor 可以被位于表单任何页上的任何用户界面控件所访问。

使用 **Data Session** 类

Visual FoxPro 7 帮助文件推荐使用 Session 类来建立你自己的 COM 服务器商业对象。这是因为使用 Session 类会自动为你的商业对象提供它们自己的私有数据工作期。这样就防止了商业对象的多个实例踏入彼此的数据内。更好的是，VFP 7 会自动排除 Session 类原有的 PEMs(属性、事件、方法)而只将自定义属性和方法写到类型库里面去。

不过，如果你准备在桌面应用和 Web 环境应用中使用同一套商业对象，那么你可能会想要采用一种不同的途径。尽管让一个位于 COM 服务器中的商业对象拥有自己的私有工作期是可取的，但是，当你在桌面应用中使用同一个商业对象的时候，就会希望让这个商业对象使用那个建立这个商业对象的表单的默认工作期。由于 Session 对象的 DataSession 在运行时是只读的，所以，没有办法可以动态的切换私有和默认数据工作期。

不过，如果你采用了类似于本文示例代码中那个商业对象那样的途径，你就能两全其美了。KbizObj 类（在 Kbizness.PRG 文件中）是基于 VFP 的 Custom 类的。这里不是直接搞一个 Session 对象，KbizObj 有一个 Session 对象，可是只有当商业对象是从一个 COM 服务器中建立的时候才会在运行时建立这个 Session 对象的实例。商业对象使用 IsCOMServer 方法来判断它自己是怎么实例化的。

理解商业规则

商业对象的主要角色之一是：强制商业规则。当商业规则与商业对象相关联的时候，不管商业对象被用在哪里　　在一个表单或者应用程序中、在一个 ASP 页上、在一个 MicroSoft Office 产品中、或者是从 Visual FoxPro DBC 的存储过程中建立的　　都可以应用上商业规则。

商业规则通常粗略的分成两大类：

- 强制数据完整性
- 强制域规则

数据完整性规则

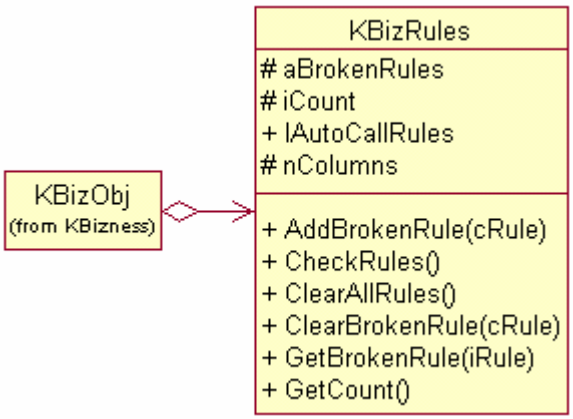
数据完整性规则确保你的数据的生命力。例如，大多数的 Cursor 在保存一个记录前必须有至少一个字段不为空…其它一些字段也可能需要包含唯一值。商业规则对象可以被用来强制这些规则。

域规则

域规则包含那些专用于某个特定的域的规则。例如，当一个客户的信用额度超过$10,000 时可能就不允许再为他声称任何新的帐单，又或者一个帐单上的最大折率只能是 10%等等。

理解商业规则对象

将商业规则分解成它自己的对象这种办法为我们提供了灵活性和可缩放性。这种结构使得商业规则可以在运行时被动态的切换。



如右面的 UML 图所示，商业规则对象强制商业规则，并跟踪所有违反规则情况的发生。商业规则对象为每一条被强制的规则增加了一个方法。如果某条规则被违反了，那么这条被违反的规则就会被添加到 aBrokenRules 数组中去。如果规则没有被违反，那么这条规则就会被从数组中清除。

商业规则方法可以独立的被调用。例如，如果某个字段必须包含一个唯一值，那么，就可以从它相应的界面控件的 Valid 方法中去调用商业规则对象上的一个方法，以检查用户输入的值是否唯一。

商业规则的方法们还可以成群的被调用。例如，当用户单击保存按钮的时候，在记录被保存前可以调用商业规则上的一个方法（例如 CheckRules），该方法会调用所有相关的商业规则。

为了维护可缩放性，商业规则对象不能显示错误消息。象表单这样的客户端可以借助于商业规则对象的 GetBrokenRule 这样的方法来得知违反了哪些商业规则，然后自己去显示错误消息。

从 Visual FoxPro 中使用商业对象

可以直接从一个 Visual FoxPro 类库中建立商业对象的实例。例如：

```
oClient = CREATEOBJ('Clients')
```

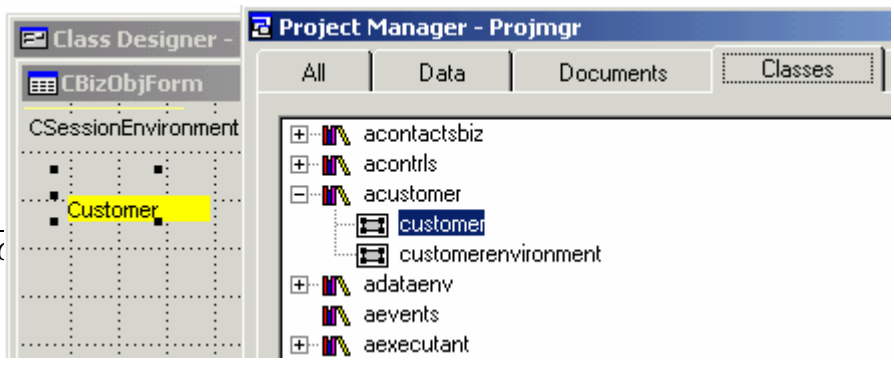
当你需要在一个 Visual FoxPro 表单上使用一个商业对象的时候，可以直接从商业对象类库中将某个类拖放到表单上，然后给表单添加调用商业对象上的方法的代码。

将界面控件绑定到 VFP Cursors 会破坏三层模式吗？

将一个商业对象拖放到一个表单上难道不会破坏三层模式吗？不会！需要专门指出这个问题，是因为在 Visual FoxPro 社群中这个问题一直在造成着困扰。为了说清楚，我们需要谈一些基础内容。

在表单上的商业对象“ 实例 ”

当你将一个商业对象从类库拖放到某个表单上的时候，你得到了什么？你很有效率的建立了一个商业对象类的实例。例



如，如右面图中所示，Customer 商业对象已经被从 aCustomer 类库中被拖放到表单上了。在表单上的 Customer 对象是 Customer 类的一个实例。

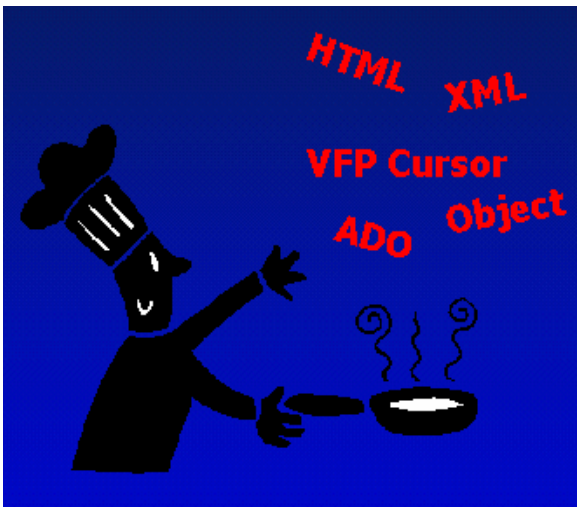
可能你会改动了表单上 Customer 对象的某些属性的值、给它添加了某些方法代码，但是，你在操作的并非类本身 你是在修改该类的一个实例。这意味着你仍然拥有着一个“纯种”的商业对象 Customer 类，你仍然可以把这个类用于 COM 服务器中，而完全不受任何 Visual FoxPro 用户界面逻辑的影响。

绑定到商业对象数据

如果你把用户界面控件直接绑定到由一个商业对象加载的数据上会怎么样？难道这不会导致将第一层直接绑定到第三层吗？答案还是不会！

关键之处在于：是商业对象在负责加载和保存数据 而不是表单。表单完全不清楚数据是怎么来的、或者数据是怎么被保存到后台的。数据可能来自与一个 SQL Server、Oracle 或者 Visual FoxPro 数据源。不管商业对象从后台拿来的是什么数据源， 它交给表单的都是一个它自己生成的 Visual FoxPro 的 Cursor（呵呵，好像…几乎…完全就是 VFP8 的 CursorAdapter 了）。当你要保存数据的时候情况也一样 虽然 Visual FoxPro 的用户界面控件可能被用来操作 Cursor 中的数据，但是，本质上是商业对象在负责用商业规则校验数据、并将数据保存到后台去。

用最适合的格式向客户端供应数据



尽管如此，一些人还是争辩说，用户界面控件的 ControlSource 属性应该绑定到商业对象的一些属性而不是一个 Visual FoxPro 的 Cursor。他们的做法是：在商业对象上为商业对象的 Cursor 中的每个字段建立一个属性，然后把用户界面控件的 ControlSource 直接绑定到这些属性。

这种机制有一些问题。首先，它不适用于 General 字段。General 字段不能被拷贝到一个对象的属性，所以你最终将不得不绕过“规则”以显示 General 字段。其次，它不适用于那些显示多条记录的控件。如果你想要在一个 Grid 上显示几百条记录的话（这种情况即使在客户/服务器环境下也是可以接受的），你就会不得不通过拷贝商业对象的属性们来动态生成一个 Cursor，然后不断地向商业对象要求下一条记录。第三，不能优化这种办法运行起来非常的慢。

这种机制是在自讨苦吃 为什么？Visual FoxPro 与 Cursor 一起工作是最好的 轻而易举。它既然没有一种基于对象的数据访问模型，我们为什么又要强做冯妇呢？商业对象不应该强行要求客户端接受某种特殊格式的数据类型，而应该由商业对象来适应客户端的需要！

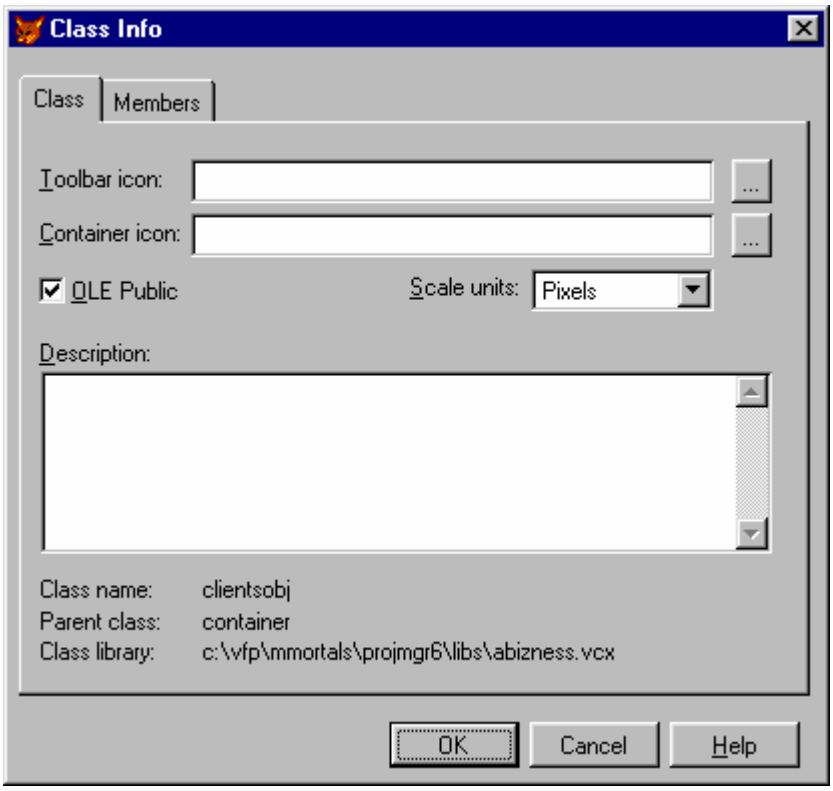
商业对象应该能向客户端提供适合客户端需要的来自于各种格式 XML、ADO 记录集、HTML 对象或 Visual FoxPro 的 Cursor 的数据。

分布式架构

关于分布式架构有什么问题呢？如果商业对象已经被拖放到某个表单上了，那么你怎样才能将你的工作站们与数据断开，并通过一个应用程序服务器来路由所有的数据访问呢？答案是商业对象已经被分解成两个部分了　　一个使者和一个执行者。查看《建立真正的 VFP 分布式应用程序》会议以了解更多的细节。

COM Business Objects

如果你将主要的应用程序逻辑都放在了商业对象类里，那么就可以很轻松的让象 Visual Basic、C++、J++、MicorSoft Office 和 ASP 这样的其它产品和技术将你的应用程序作为一个自动化服务器（COM 部件）来访问。



你需要做的，只是在 Visual FoxPro 中将一个包含着被定义为 OLEPUBLIC 的商业对象类的项目建立为一个自动化服务器。在这个项目中，你可以根据需要拥有任意多个 OLEPUBLIC 类。

你可以指定到底要有多少个 OLE Public 类。例如，你可能会希望只向外提供几个 OLE Public 商业对象，只在幕后建立其它的商业对象的实例。

要将一个类指定为 OLE Public 的做法如下：

1. 在 Visual FoxPro 中打开这个类；
2. 从类菜单中选择类信息；
3. 选中 **OLE Public** 复选框；
4. 单击 **OK** 保存改动。

编译自动化服务器

当你编译一个带有 OLE Public 类的项目时，Visual FoxPro 会建立三个文件：

- DLL 或 EXE ；
- 一个类型库 (TLB) 文件；
- 一个注册 (VBR) 文件。

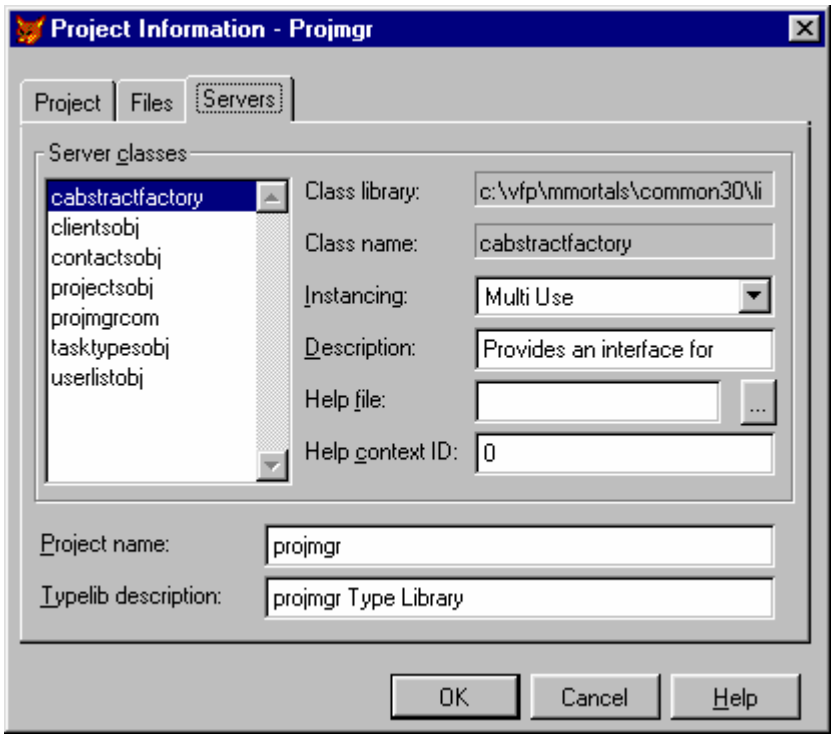
默认情况下，DLL 或 EXE 文件的文件名与你的项目名相同，扩展名是 DLL 或 EXE。

项目信息

在编译完了项目以后，单击项目菜单然后选择项目信息以打开项目信息对话框。单击服务器页来看一下所有的服务器类列表。

当你单击每个服务器类的时候，你可以为每个类改动实例、描述和 **Help context ID**。帮助文件、项目名称和类型库描述用于所有的类。

这些信息会被放到类型库和 Windows 注册表里面去。



类型库

类型库是一个二进制文件，它列出了你的自动化服务器中的所有发布的类，以及它们的属性、方法和时间。Visual FoxPro 的类浏览器可以读取类型库，并用容易理解的形式将它们出来。

所有的自定义、用户指定的属性、用户指定的方法都列在 Visual FoxPro 类型库中，并且它们都被标记为 Public。对于方法，Visual FoxPro 还包含一个返回值的类型（variant）和一个从方法中分析出来的参数（variant）列表。来自.vcx 类文件中属性的描述也记录在了类型库中。

在编译过程中生成的类型库保存在 TLB 文件里。

在运行 Windows 2000 或 Windows NT 的机器上，Visual FoxPro 编译生成的 DLL 或 EXE 文件中也带有类型库以作为一个绑定的资源。这样实际上就不需要再另外生成一个 TLB 文件了，不过为了远程发布的需要，这个文件还是会与 VBR 文件一起被生成。对于远程发布，你可以只带上 TLB 和 VBR 文件，并用 CLIREG32.EXE 将 VBR 文件注册为指向远程服务器。