管理 VFP 类库

作者： Steven Black
译者： fbib

你开发的应用程序越多、开发这些应用程序的团队越大、软件修订的次数越多，掌握管理类库的技巧就越重要。面对现实吧：没有把类库管理好，你就不可能发布一个大的应用程序。当程序变的越来越大的时候，结构正确的重要性就成指数的上升。这次会议讲述的是一个有经验的 Visual FoxPro 程序员应该怎样更有效的管理类库、更聪明的使用可视的、以及代码式的类库。此外，我们还将讨论类和类库的概念和物理的管理、VFP 类浏览器和组件管理库在类库的管理上是怎样的起着互补的作用的、以及类库结构与部件架构之间的关系。时间允许的话，我们还将讲述一下针对类库设计、文档、信息传递、以及版本控制策略方面的工具。

我写下这份文档来向大家共享一些处理 VFP 类库明智的实务。注意，如果你是独自工作的、或者你在一个小项目中工作，那么我在这里向你推荐的材料的用处，要比你在开发一个大系统、并且与其它程序员合作中的用处要小的多。因为经验证明，管理 Visual FoxPro 类和类库的复杂性会随着项目的大小、开发活动的激烈程度、项目类设计的复杂性、以及涉及的程序员数的变化而变化。

俗话说：要建一座 1 米高的塔，你只需要一些未损坏的啤酒罐。而要建一座 10 米、100 米高的塔的话，你就需要一个计划、还需要去注意过程和结构。这篇文章讲的就是这个内容：怎样建立你的类库的结构以提高它们的可缩放性。

好吧，现在我就不再罗嗦了。

首要原则：在你的应用程序中使用你自己的类作为控件的基类。

图 1 展示的，是一个直接从一个 VFP 基类中派生出来的类层次结构。这是一个经典的菜鸟所犯的错误。这里的错误是：在类层次的顶端是一个只读的类、一个 Visual FoxPro 自带的基类。由于这个顶层类是只读的，因此，当你需要对这个层次结构做一个全局的改动的时候，你就不得不去改动所有的第一层子类，这样，在类层次结构把中被我们称作“继承”的好处就几乎全部丧失了。

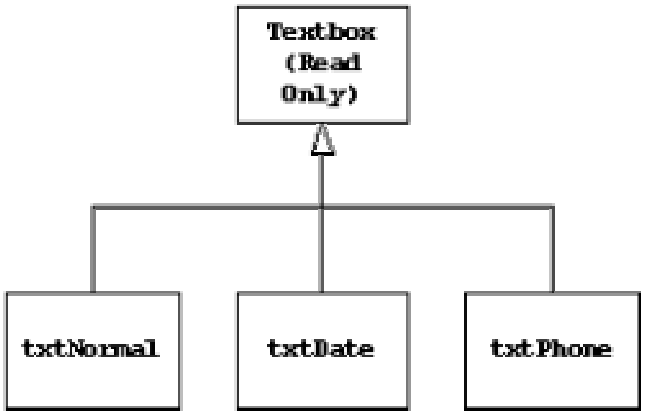

图 1、一个直接从一个 VFP 基类派生出来的类层次结构

将图 1 与图 2 比较一下，图 2 的层次结构中多了一个你自己建的顶层类。要对这个层次结构做一个全局的改动的话，只要对这个顶层类（图 2 中的 _Textbox 类）做改动就可以了，只要那些子类不出毛病，它们就会继承这个改动。

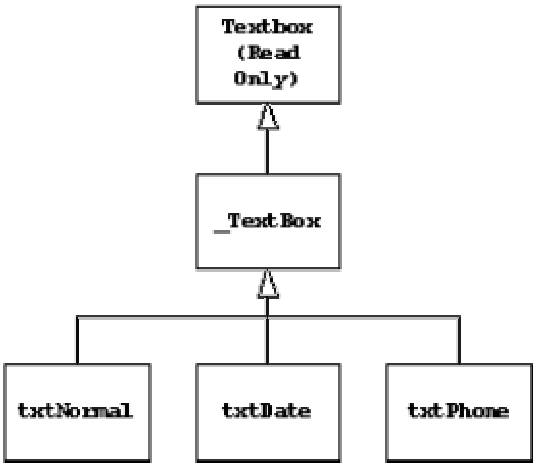

图 2、从你自己的基类中派生出来的一个类层次

在你的工作类与 VFP 的只读基类之间用上你自己的基类，可以给你的类和 VFP 基类之间提供一个缓冲，让你的类可以免受 VFP 基类的特性改动的影响。它还给了你一种将你自己的标准集成倒你所有类的根上去的途径。

次要原则：使用隔离的类以免受版本更动的影响

由第三方提供的类应该被看作是只读的因素，即使你拥有它们的源代码也一样。第三方的类的改动可能会非常频繁、而且难以预测。如果你对第三方类起初的源代码进行了改动，就会碰到这样一个问题：怎样把你的改动可靠的集成到新版本的事件中去？此外，还要考虑并行问题：怎样同时管理好在企业范围内的/应用程序专用的变动而不触及第三方的源代码。

隔离的层（见图 3），有时也被称作“ I 层 ”，它是一个让我们可以对应用程序框架和其它易变的可派生结构进行集中访问的伪顶点。在图 3 中向上看，应用程序专用的类可以根据

项目的不同而不同，而隔离的层通常则作为一个企业级别的、与框架隔离的层来服务。

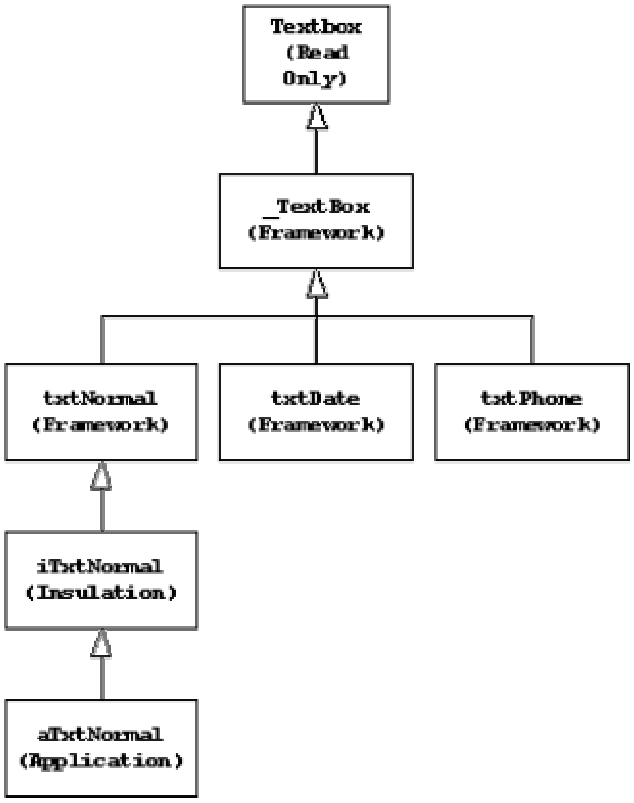

图 3、隔离层让你可以免受应用程序框架变动的影响，代价是增加一个额外的继承层。

隔离的层并非没有先例，重要的是：它们给你的类继承体系增加了一些层，这会让象调试这样的事情变得更加的枯燥乏味。

如果你在为多个客户开发软件，可以考虑为每个公司建立一个层。这种办法只会在某个软件公司一再的为一批固定客户开发软件时才有意义。基本的结构如下：

VFP 基类
你的 VFP 隔离类
客户 1 类
应用程序 1 类
应用程序 2 类
客户 2 类
应用程序 3 类
应用程序 4 类

所以，隔离类是一个较好的结构设想，付出的代价只是让你的类继承层次略微加深一点。现在，继续看下去的话，你很快就会看到我们将马上让这些类各自定位在不同的层上。那么怎么最佳的处理它们呢？

最佳措施：了解 VFP 类浏览器和组件管理库，并理解它们各自的优点。

类库是类的一个物理包。要管理可视的类库，你可以使用类浏览器。但是，类能够也应该 用多种不同的形式抽象出来。这方面，我们可以使用组件管理库来组织它们，包括 VCX 或者 PRG,还可以包括软件的其它元素。组件管理库在以一种符合实际情况的方式为相关的项目建立虚拟和抽象的组方面是很棒的。

在使用类浏览器和组建管理库方面的白皮书可以从以下位置下载：

http://www.msdn.microsoft.com/vfoxpro.

最佳措施：使用 Ken Levy 的 superclass 工具。它让你可以准确的看到你正在继承什么。

你的类层次越深，你就会越重视象 SuperCls.prg 这样的工具，它现在已经随着 VFP 一起销售了（译者注：我在 VFP 的目录下找不到这个 SuperCls.prg，去 MSDN 的网站发现过去这个 PRG 是要买了正版的用户才能在线下载的。难道作者指的是 VFP8 中可以查看父类代码的功能？）SuperCls 给了你在类设计器的代码编辑窗口中查看和编辑超类代码的能力。

最佳措施：打开工具|选项|字段映射，把你自己的基类设置为默认的字段映射类，这会把你的类集成到 VFP 的开发环境（IDE）中。

你可以通过工具|选项|字段映射来建立一套默认类，在从数据环境中拖放字段到表单上的时候，VFP 会用这些默认类来建立控件。John Petersen 有一个工具，可以帮助你管理针对不同的项目使用不同的一套默认类。对于每一个不同的项目，你也可以使用 Regedt32 程序来把下面的键保存到项目文件夹的一个文件中去：

HKCU\Software\Microsoft\VisualFoxPro\5.0\Options\IntelliDrop

当要切换到另一个项目的时候，你只需要在资源管理器中双击这个文件就可以了。

次要原则：保存在每个类库中类的数量要尽量的少。

Visual SourceSafe 在同一时间一次只允许一个人签出一个类库。单个的类由于储存在类库中，因此不能单独的被签出。所以，签出一个类就等于签出该类所储存于的那个类库。

此外，发现某个特定的类库不能马上就可以被独占的使用的概率，也会随着开发过程的激烈程度、同时工作的程序员数量、以及每个类库中类的数量的上升而上升。所有这些因素

都促使你应该尽可能让类库保持精干。

记住，可视类库的 VCX 文件是发布的物理单元。把应该被一起打包的类们用类库打包是一个好主意。因为这样做了以后，每次要对那些相互依赖的类进行独占的编辑的时候，只要签出一个类库就可以了。因此，在设计类库的时候，放在一个类库中的类应该以类之间的相关性而不是类的类型作为标准。从这个角度来说，除非你是独自工作或者工作在一个很小的团队中，否则建立一个“ 按钮 ”或者“ 表单 ”类库是毫无意义的。

次要原则：将类按逻辑性进行存储，以最大限度的减少耦合

乍一看，把类们分门别类的分配到各个类库中去似乎很简单：把所有第一层的子类放在一个 VCXes 里面，把所有的表单类放在一个 VCX 里、所有的好控件都放在另一个 VCX 里，还有一些其它的类库，比如混合类的类库、表单类的类库、工具栏类的类库等等。一切看起来都那么直观，而且便于查找。

但是，类库划分的指导原则是耦合。排列你的类的时候应该按照自然的规律，以防止在发布和重用它们的时候不会因为对外部的依赖性而受到影响。那些彼此依赖的类应该被存储在一起，或者在最近的相关联的类库中。

次要原则：不要把自定义的类与“ 基类 ”放在一个类库中

如果你遵守了所有的类应该从可读/写的顶点类派生出来的通用设计原则，而这些顶点类的存在，对你应用程序中所有其它类来说是必需的。那么，为了日后可以重用这些类，超类的类库里面不相干的的东西应该尽量的减少，超类类库应该保持相当的精干。

可以考虑把你的基础类库分成可视的、分可视的基类类库，或者其它拆分方式也可以。这样的话，当你在其它什么地方需要重用一个抽象的非可视类的时候，你就不需要链接那些只是放在一个超类类库里的不相干的可视类库了。

最佳措施：使用尽量短的类名　　_txt vs _textbox

Visual FoxPro 的编译器会对语法进行检查。可是它无法检查类、字段或者变量的名称是否正确。你的类的名称越长，就越容易出现输入错误。

最佳措施：不要按照类的层次来给类命名

虽然按照一种（或多种）容易理解的命名转换方式在类名称中加上前缀或者后缀通常是一个好主意，不过我还是不推荐按照类的层次来给类命名。换句话来说，就是避免按照下面

这样的方式给类命名：

Txt
    txtLevel1
        txtLevel1Level2
            txtLevel1Level2Level3

最佳措施：知道什么时候要把类储存在 PRG 中，什么时候储存在 VCX 中。

PRG 类库的优点：

PRG 只有一个文件，VCX 有两个（包括 VCT）。

即使 PRG 文件是只读的，也可以被编译。

使用 PRG 可以让多个程序员对同一个类进行工作。不过在签入的过程中合并的时候还是需要小心，因为 VSS 的合并对话框并不清楚被合并文件的意图。

使用 PRG 类库让程序员可以很轻松的看到在不同版本之间的区别。

支持通过编译指令根据情况排除部分方法（及属性） 尤其在调试方面很有帮助，因为它可以随意取消一个错误处理器。

可以轻松的给个别属性添加注释。

PRG 类库的缺点：

由于不能使用属性窗口和类浏览器，使得类结构变得难以理解。

不少程序员觉得代码方式的类令人不舒服。

不能得到类浏览器的任何服务。

自动分析 VCX 远远要比分析 PRG 容易的多，所以许多高级的维护功能用于 VCX 要容易的多。

最佳措施：了解 VFP 的元数据结构

VFP 的 VCX 就是一个 DBF 文件，只是换了个扩展名。从这个角度来看，所有的 VFP 设计器（类设计器、表单设计器、菜单设计器、报表编辑器、以及项目管理器）都只不过是用于向 VCX、SCX、MNX、FRX 和 PJX 中填充数据的程序。

有时候，最有效率的干活方式是直接把类库当作是一个表来打开。意思就是 USE xxx.vcx EXCLUSIVE，然后直接对象 ClassLoc 字段这样的内容进行编辑后再关闭文件。最后执行一下“ COMPILE FORM xxxx ”(如果你是在 hack 一个 .SCX 文件) 或者“ COMPILE

CLASSLIB XXX ”（如果你是在 hack 一个 VCX 文件）。

在你这么干以前最好先备份一下　　如果修改出错的话，文件结构可能会被破坏掉。

有一件很棒的事情：你可以写批处理程序来一次就对某个满是表单的目录进行操作。比如下面这样的代码，它可以为一个目录中所有的表单重新指定类库的路径。

```
lnFormCount = ADIR(laForms, '*.SCX')
FOR lnCnt = 1 TO lnFormCount
    USE (laForms[lnCnt]) EXCLUSIVE
    REPLACE ALL classlibWITH <newpath> ;
        FOR classlib = <oldpath>
    USE
    COMPILE FORM (laForms[lnCnt])
NEXT
```

重要原则：一个 VFP 混合类的“类”只与其容器有关。

Visual FoxPro 支持大量的容器抽象。表单集可以包含表单、表单可以包含控件、表格可以包含自己也包含控件的列、选择按钮组可以包含选项按钮，而容器和 Custom 类们几乎可以包含任何东西。

例如，你可以有一个用于显示地址的地址可视类，它也许包含一些文本框和标签，甚至还有一些可以与用户进行交互的特性。但不要被这些表象迷住了你的眼睛：关于一个对象的容器，唯一高级的东西就是那个容器。那些在容器中的对象们只不过是其它类的实例而已。

当你把一个文本框类放到另一个容器类里的时候，你并非是在建立一个文本框的子类，只是在建立文本框类的一个实例而已。这个容器类的一个子类会包含那个文本框，但是，该文本框是从你的 textbox 基类继承来的，而不是从你在容器类中添加的那个文本框继承来的。

请阅读 Drew Speedies 的文章《Dangers of Composite Classes in Development（混合类在开发中的危险）》（FoxPro Adivsor，1998 年 2 月刊）。通常的规则是：只建立你想要实例化的混合类，而不要去从这个混合类来派生。

次要原则：在你的 Grid 和 Commandgroups 中使用你自己的类。

把你的文本框类添加到 Grid 的列，并删除原有的 VFP textbox 基类。就是把你的文本框从表单控件工具栏中多方到列里面，然后选中 VFP 的 textbox 基类再按下 delete 以根除原有的文本框。

次要原则：理解何时应该或不该将 VCX 包含进你的 app 里面。

你难道必须要把类库包含在可执行文件里面一起发布吗？不 !你可以把任何文件从一个项目中排除，只要这些文件还在 SET PATH 指定的路径里面，那么运行时 VFP 就能找到它。当然，你将不得不自己去记住要正确的发布哪些文件，不过这个策略让你可以极大的提高你更新和定制的灵活性。

主要原则：使用源代码控制来保护你的类库

从现在起，我谈到源代码控制的时候就指的是 Visual Source Safe，不过我所说的同样也适用于 PVCX 或其它任何你可能使用的主力源代码控制系统。

最佳措施：如果你是在做实验的话，从让你心跳的位置开始使用源代码控制安全网。

我经常会发现我想要能拷贝一个类，以便我可以试验一些新的想法又不会对原有的类产生影响。这些就是 Visual SourceSafe 能为你效劳的地方了。你可以在任何时间恢复到一个旧的版本。

指导方针：VFP 的源代码控制集成能力的体会

VFP 自带了源代码控制集成，但你并非必须要用那种方式来做。事实上，我所尊敬的一些程序员们都有意的避开了源代码控制集成，而采用交互式的源代码控制产品。有许多这样做的理由。最重要的是，你将会主意到，随着 SourceSafe 的源代码数据库变得越来越大，它的反应也变得越来越慢。其次，通过使用交互式使用 Source Safe 客户端的 VFP 的集成只提供了很少几个功能和可控制的地方。第三，令人恼火的 Source Safe 对话框经常出现，让你的 VFP 桌面变得混乱不堪，而且集成还被强制通过糟糕的中间文件文档来管理，当你在开发环境中拷贝文件的时候很容易忘了或者删除了这些文件。

指导方针：定时清理你的 Source Safe 项目

源代码控制是需要开销的。如果你使用了源代码控制集成，而且发现打开一个文件的反应时间变得很慢了的时候，就该考虑一下清理你的源代码控制数据库了。我猜测，不过不确定，Source Safe 的反应时间与它的数据库的大小成反比，而每当你签入某样东西的时候数据库就会变大。

最佳措施：注意交互式源代码控制的某些常见问题

如果什么时候你打开一个项目的时候碰到了这样的消息：“ 发生了一个不明的注册错误 ”,这就意味着这个项目是由 Visual SourceSafe 控制着的,而你却没有在 VFP 中激活 VSS，或者根本没有安装它。如果你真的不需要链接 SourceSafe 的话，可以清除 PJX 文件中的 SCCDATA 字段，并把 LOCAL 字段设置为 .T.。

指导方针：VFP 需要读/写的权限以编译它的结构。

如果你有一个不允许编译的项目，并且碰到了错误消息“ 不能对一个只读游标进行写操作 ”，通常就是碰到了一个只读文件。你可能是有一些只读的 SCX 或者 VCX 文件，又没有在项目中把它们标记为签入 SourceSafe。

为了能够编译表单和类，这些文件必须是可读写的。这是因为 VFP 把编译了的代码放在 VCT 或者 SCT 里面。你可能没有签出某些 VCX 或者 SCX 文件，因此它们就仍然是只读的。

如果你是采用交互式的 Source Safe，当你在一个项目中执行一个取得所有文件的最新版本操作的时候，请确保选中了“ 设为可写 ”选项。由于对象代码是作为元数据的一部分被储存的，因此 VFP 要求它的不少结构是可写的。

(全文完)