目　录

第一章　Visual FoxPro 编程简介

1.1　为什么要编程

Visual FoxPro 将面向过程的程序设计与面向对象的程序设计结合在一起，用户可以用它创建出功能强大、灵活多变的应用程序。程序就是为了完成某一具体任务而编写一系列指令。

通常，在程序中能够完成的工作都可以通过人工操作来完成，那么为什么还要编程呢？请看下面的例子对比。例如，某公司要在雇员“employee”表中查找一个特定雇员（陈智勇）的信息，可以通过人工执行一系列指令来实现。

人工操作步骤：

(1) 选择“文件”菜单中的“打开”命令。

(2) 在弹出的“打开”对话框的“文件类型”列表框中选择“表”。

(3) 打开相应的表。

(4) 在“显示”菜单中选择“浏览”命令。

(5) 滚动浏览表，查找 employee 字段为“陈智勇”的记录。

若要通过编程实现上述目的，可以在“命令”窗口中键入下列命令：

```
USE ...\...\employee
LOCATE FOR employee= "陈智勇"
BROWSE
```

在表中找到这个雇员后，就可以执行特定的操作（如将工资提高百分之十）。

若要通过人工操作提高该雇员的工资，步骤如下：

(1) 选定 salary 字段。

(2) 将 salary 字段值乘以 0.1，再加上原来的字段值，然后在该字段处键入得到的结果。

若要通过编程实现以上结果，可以在“命令”窗口中键入以下命令：

```
REPLACE salary WITH salary * 1.1
```

如果要修改某个雇员的工资，用人工方式或在“命令”窗口中键入指令都很容易实现，但如果要把所有的雇员的工资都提高百分之十，恐怕就不是一件轻而易举的事情。若还是用人工方式来做，不但费时费力，而且还容易出错。解决此类问题的更好办法是编写一个可执行的程序文件，用该程序可以轻松无误地完成这一工作。

增加所有雇员工资的示例程序：

代码	注释
USE ...\...\employee	打开 employee 表。
SCAN	浏览表中所有记录，针对每条记录执行 SCAN 与 ENDSCAN 之间的所有指令。

```
REPLACE salary WITH ;
salary * 1.1                    将工资增加百分之十。（分号(;)表示续行。）
ENDSCAN                         结束由 SCAN 开始、针对表中所有记录执
                                行的指令序列。
```

和“命令”窗口中单独键入每条命令相比，运行程序有如下优点：

程序可被修改并重新运行。

可从菜单、表单和工具栏启动程序。

一个程序可调用其他程序。

1.2 VisualFoxPro 编程的特点

大多数编程语言具有相似的基本特征。如果熟悉一种编程语言就可以很容易地将这种编程语言的知识应用到其它编程语言中。下面讲述 Visual FoxPro 同几种常用编程语言的比较。

(1) Visual FoxPro 编程中不区分大小写。与流行的 C/C++不同， Visual FoxPro 编程中不区分大小写，但为了程序易读易维护，建议用户对标识符使用固定的大小写规则。

(2) Visual FoxPro 不需要明确声明变量。当把值保存到变量而该变量不存在时，Visual FoxPro 隐含地声明它。这种情况对于需要明确声明变量的编程语言来说会产生错误。

(3) 在 Visual FoxPro 中，不能为变量指定数据类型（虽然在命名变量时建议您使用表示其数据类型的前缀）。其他语言在声明变量时，需要为其指定数据类型。

(4) 每种编程语言都有它自己的在程序中添加注释的格式。 Visual FoxPro 的注释格式是：

```
*           整行注释
USE         && 结束行
NOTE        多行注释
```

(5) 赋值语句。 Visual FoxPro 中的赋值方式是： nVal = 1 或者 STORE 1 to nVal 。在 Visual FoxPro 中，也可以使用 REPLACE 命令给表中的字段赋值。

设计程序时，就是用一系列指令存储数据并操作这些数据。程序设计的原材料是数据和数据的存储容器，而处理这些原材料的工具是命令、函数和操作符。

(6) 存储数据。用户使用的数据可能包括时间、货币数量以及日期、名称和说明等。每个数据都有其数据类型，属于同一类型的数据可以按相似的方法进行处理。您当然可以直接处理数据而不加以存储，但这样做会失去很大的灵活性，而且 Visual FoxPro 提供的许多功能没有发挥作用。为增强处理数据的能力， Visual FoxPro 提供了多种数据存储容器。

(7) 数据类型。数据类型决定了数据的存储方式和使用方式。两个实数可以做乘法运算，但两个字符型数据不能做乘法运算。同样道理，字符可以用大写方式打印，而数字就不存在大小写的问题。下表列出了 Visual FoxPro 的主要数据类型。

数据类型	类型示例
数值型	123 .31415 -71

字符型	"West String"	"23" "1/01/95"
逻辑型	.T. .F.	
日期型	{02/01/98}	
日期时间型	{01/01/98 12:32:00 am}	

(8) 数据容器。数据容器允许在多个数据上进行相同的操作。例如，将一个雇员工作的小时数加起来，再乘以每小时工作应付的工资，扣除税款后，便可知道一个雇员应得的报酬。若对每个雇员都进行这样的操作将非常麻烦，但若将这些信息保存在数据容器中并对数据容器进行操作，那么，通过运行程序就能实现数据的更新。下面列出了 Visual FoxPro 中主要的数据容器：

数据容器的类型	说明
变量	在随机存储器 (RAM) 中的单个数据元素
表记录	多行预定义字段，每个字段包含一条预定义数据，表存储在磁盘上。
数组	随机存储器中的多元素数据。

(9) 处理数据。数据容器和数据类型构成了处理数据的基础，而对数据的处理最终要通过操作符、函数和命令来实现。

1）使用操作符。Visual FoxPro 中最常用的操作符如下：

=、+、！或 NOT、*, /

注意，操作符应与数据类型相匹配。下面的语句将两个数值赋给两个变量，因为变量名的首写字母为 n，所以可马上知道该变量保存的是数值型数据，但这样的规定只是约定的，实际上，您可以使用字母、数字及下划线的任意组合为变量命名。

```
nSalary = 123
nBonus = 456
```

下面的语句将两个字符串赋值给两个变量，变量名首写字母 c 表明该变量保存的是字符型数据。

```
cName = "Tom"
cCompany = "Macrohard"
```

因为 cName 是字符型数据，而 nBonus 是数值型数据，所以执行下面操作时将出现数据类型不匹配的错误：

```
? cName + nBonus
```

2）使用函数。函数可返回特定类型的数据。可以通过这些函数所带的文档来了解其返回值类型。

可以用以下五种方法调用 Visual FoxPro 函数：

a.将函数的返回值赋给某个变量。

下面这行代码使用变量 dToday 保存当前系统日期。

```
dToday = DATE( )
```

b.在 Visual FoxPro 命令中包含函数调用。

下面的命令使用 GETDIR( ) 函数的返回值设置默认路径。

```
CD GETDIR( )
```

c.在活动输出窗口中输出返回值。

下面这行代码在 Visual FoxPro 主窗口中输出当前系统时间。

? TIME( )

d.调用函数但不保存其返回值。

下面的函数调用将关闭临时表。

SYS(2002)

e.函数嵌套。

下面这行程序输出今天是星期几。

? DOW(DATE( ))

3）使用命令。一条命令即完成一个特定动作的指令。每条命令都有自己特定的语法，用来说明为实现该命令的功能所必须包含的东西。此外，与命令有关的还有一些可选子句，这些关键字可进一步指导命令干些什么。

例如，USE 命令可以打开表或关闭表：

USE 语法	说明
USE	关闭当前工作区中的表。
USE employee	在当前工作区中打开 employee 表，同时关闭当前工作区中所有已打开的表。
USE employee IN 0	在下一个可用工作区中打开 employee 表。
USE employee IN 0 ALIAS sonny	在下一个可用工作区中打开 employee 表，并且指定该工作区的别名为 sonny 。

(10) 程序流的控制。Visual FoxPro 中有一类特殊的命令，它们可以反复执行其他命令或函数，并决定这些命令或函数何时执行以及执行次数。这类特殊命令可用来实现两种主要的程序结构：条件分支和循环，它们在程序设计过程中作用很大。将在下一章详细讲述这些命令。

掌握了一些基本概念后，程序设计就变成了一个不断重复的过程。您需要多次重复某些步骤，并在这个过程中不断对代码进行优化。从编写第一行程序开始，便不断进行测试，完成每个“试验—查错”的过程。对语言越熟悉，则编程速度越快，而且更多的初步测试工作将在头脑中进行。

程序设计的基本步骤包括：对问题进行说明；分解问题；编制各模块；测试并完善各模块；组装全部模块；整体测试。

开始程序设计之前，请注意以下几个问题：

（1）在解决问题之前，必须把问题说明清楚，否则会不断进行修改，丢弃已编好的代码并从头再来，而且最终也不可能得到满意的结果。

（2）将问题分解成可单独处理的几个步骤，而不是一下子解决全部问题。

（3）在开发过程中不断测试和调试已编好的代码。通过测试检查代码是否能实现所需的功能；调试是找出代码在哪里出错并纠正这些错误。

（4）精炼数据和数据存储方式，便于程序对其进行处理。这需要正确构造表格的结构。

1.3 Visual FoxPro 中设计程序的步骤

Visual FoxPro 程序由代码组成，代码包括以命令形式出现的指令、函数或 Visual FoxPro 可以理解的任何操作。这些指令可以包含在：“命令”窗口中、程序文件中、“表单设计器”或“类设计器”的事件或方法程序代码窗口中、“菜单设计器”的过程代码窗口中，或者“报表设计器”的过程代码窗口中。下面分别介绍：

1. 在“命令”窗口中的代码指令

可以在“命令”窗口中键入 Visual FoxPro 命令并按 ENTER 键执行。若要重新执行该命令，还可以将光标移到此命令所在行并按 ENTER 键。也可以在“命令”窗口中将多行代码象独立程序一样执行，方法是：选择要运行的代码行，然后按 ENTER 键。因为“命令”窗口是一个编辑窗口，所以在编辑命令时可以使用 Visual FoxPro 提供的编辑工具。在“命令”窗口中可以编辑、插入、删除、剪切、复制和粘贴命令正文。在“命令”窗口中执行命令的优点是：能够立即执行被键入的命令，不需要将其保存为文件。

此外，在菜单和对话框中所作的选择可以马上转化为“命令”窗口中的命令。用户可以将这些命令复制并粘贴至 Visual FoxPro 程序，然后重复执行这些程序。这样做很容易重复执行成百上千条的命令。

2. 编制程序

Visual FoxPro 程序是包含一系列命令的文本文件。在 Visual FoxPro 中，可以通过以下步骤建立程序：

（1）在“项目管理器”中，选择“代码”标签中的“程序”项。

（2）选择“新建”命令。

也可以按以下步骤进行：

（1）在“文件”菜单中选择“新建”命令。

（2）在弹出的“新建”对话框中选择“程序”。

（3）选择“新建文件”按钮。

还可以这样做：

在“命令”窗口中键入“ MODIFY COMMAND ”， Visual FoxPro 就会打开一个称为“程序 1 ”的新窗口，这时就可以键入编写的应用程序了。

3. 保存程序

程序编写好并输入后，请注意保存。若要保存程序，可以通过在“文件”菜单中选择“保存”命令来保存程序文件。若用户要关闭一个没有保存的程序，则会弹出相应对话框，提示用户是保存还是放弃已做出的修改。若用户保存了一个由“项目管理器”创建的程序则保存的程序被加入到当前项目中。若用户保存一个尚未命名的程序，则会打开“另存为...”对话框，提示用户为程序指定程序名。程序保存后，用户可运行或修改它。

4. 修改程序

程序创建后可以修改。要修改程序，先要打开程序文件。若要打开程序文件，按下述方法进行操作：

若程序包含在某个项目中，则在“项目管理器”中选择它，然后选择“修改”命令。也可以在“文件”菜单中选择“打开”命令，弹出“打开”对话框。在“文件类型”列表框中选择“程序”，然后在“文件名”列表中选择要修改的程序的文件，单击“确定”按钮，就可以打开要修改的程序文件了。此外，在“命令”窗口中键入以下命令，也可以打开要修改的程序文件：

MODIFY COMMAND program-file-name

如果记不清程序文件的名字，可以在“命令”窗口中键入：

MODIFY COMMAND ?

可以弹出“打开”对话框，然后在文件列表中选择要修改的程序，单击“打开”按钮打开要修改的文件。

打开文件之后便可进行修改，修改完毕后不要忘记保存。

5. 运行程序

程序创建完毕后便可以运行。运行程序的步骤是：

若程序包含在一个项目中，则在“项目管理器”中选择该程序，然后单击“运行”按钮。或者在“程序”菜单中选择“运行”菜单项。然后在弹出的程序列表中，选择想要运行的程序，单击“运行”按钮。也可以在“命令”窗口中键入以下命令：

DO program-file-name

就可以运行程序了。

6. 使用 Visual FoxPro 设计工具编写代码

通过“表单设计器”、“类设计器”和“菜单设计器”，程序员可以很容易地把程序代码与用户界面连接起来，这样应用程序便可响应用户的输入并执行相应的代码。同样，“报表设计器”将程序代码与报表文件联系起来，以此定制结构复杂并且符合用户要求的报表。

使用上面提到的设计工具可以充分发挥 Visual FoxPro 的强大功能。

1.4 本书特色和内容提要

本书循序渐进，由浅入深地讲述了 Visual FoxPro 中的编程技术，在介绍了 Visual FoxPro 编程简介后，第二章从程序结构讲起，使用户了解程序的组成，初步掌握程序设计的步骤。然后在第三章中介绍 Visual FoxPro 的数据库操作命令，在程序设计中要经常用到这些命令。上面讲过，在 Visual FoxPro 的程序设计中还可以使用函数，第四章介绍了 Visual FoxPro 的常用函数的用法。Visual FoxPro 支持面向对象的程序设计方法，利用面向对象的程序设计方法可以编制出功能强大、灵活多变的应用程序。在讲述了基本的程序设计知识后，本

书将介绍面向对象的程序设计（ OOP ）方法，是用户编制的程序可以发挥面向对象的威力。这之后将综合前面所讲述的程序设计技术，通过实例讲述用面向对象的方法快速创建应用程序。最后还要介绍通过对象链接与嵌入（ OLE ）实现 Visual FoxPro 与其他应用程序交流的方法以及网络环境下的开发技术。

程序的开发是离不开测试与调试的， Visual FoxPro5.0 提供了一个强大的调试工具，通过它，用户可以轻松地调试自己的程序，迅速排除错误，优化代码。本书详细讲述了该调试器的使用。

第二章　Visual FoxPro 的程序结构

同其他程序设计语言一样，Visual FoxPro 的编程语言中也有专门控制程序流程的语句。它们可以反复执行其他命令或函数，并决定这些命令或函数何时执行以及执行次数。这类特殊命令可用来实现两种主要的程序结构：条件分支和循环，它们在程序设计过程中作用很大。为了更好地理解示例程序，先讲述几个最常用的命令。

2.1　数据的输入输出命令

2.1.1　?/??

这两个命令求出表达式的值并显示表达式的结果，使用的语法是：

? | ?? Expression1

[PICTURE cFormatCodes] | [FUNCTION cFormatCodes] | [VnWidth]

[AT nColumn]

[FONT cFontName [, nFontSize] [STYLE cFontStyle | Expression2]]

[, Expression3] ...

参数为：

? Expression1　求出表达式 Expression1 的值并在表达式的结果前插入回车换行符。结果显示在 Visual FoxPro 主窗口或者用户定义的活动窗口的下一行，并且靠窗口的左边显示。除非功能代码 cFormatCodes 或者系统变量_ALIGNMENT 指定别的显示方式。

如果省略表达式，则显示一个空行。当包括多个表达式时，在显示的每个表达式之间加入一个空格。

?? Expression1　求出表达式 Expression1 的值并把表达式的结果显示在 Visual FoxPro 主窗口或者用户定义的活动窗口，或者打印机的当前行的当前位置。在表达式的结果前不插入回车换行符。

PICTURE cFormatCodes　指定显示 Expression1 结果所使用的图形格式。cFormatCodes 可以包含功能代码、图形代码，或者两者都包括。用户可以使用与 Format 和 InputMask 属性中可用的相同的代码。

功能代码影响结果的整个格式，而图形代码则只影响结果中的单个字符。如果在 cFormatCodes 中使用功能代码，则它们必须出现在图形代码的前面。而且它们的前面必须有一@符号。无内嵌空格的多个功能代码可以直接跟随在一个@符号后面。最后一个功能代码后面必须有一个或者多个空格。这些空格标志着功能代码的结束和图形代码的开始。

FUNCTION cFormatCodes　指定? 和 ??输出中的功能代码。如果包含功能子句，则不要在功能代码前键入@符号。在功能代码包含在 PICTURE 中时前面必须有一个@符号。

VnWidth 指定一特殊的功能代码，这个代码使字符表达式的结果垂直拉伸有限列。

nWidth 指定输出中拉伸的列数。

AT nColumn 指定显示输出所在的列号，这个可选的功能使用户可以对齐输出的列从而制作出表格。数值表达式 nColumn 可以是一个用户定义的返回数值的函数。

FONT cFontName [, nFontSize] 指定 ?! ?? 所输出的字体。cFontName 指定字体的名称，nFontSize 指定字号的大小。例如，下面的命令用 16 号字显示系统时间。

? DATE( ) FONT 'Courier',16

如果命令中包括 FONT 子句但忽略了 nFontSize ，则使用 10 号字体。

STYLE cFontStyle 指定?! ??输出的字体风格。如果省略 STYLE 子句，则使用 Normal 字体风格。如果不能得到指定的字体风格，则用最接近的字体风格代替。

说明：

? 和?? 求出表达式的值并把结果送到 Visual FoxPro 主窗口，或者活动的用户定义窗口，或者打印机。

?! ??命令示例：

? 15 * (10+10)　　显示结果如图 2.1 。

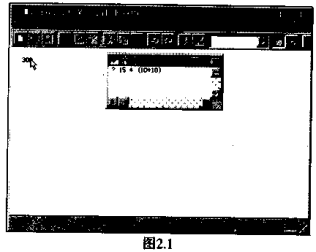


图2.1

? 'Welcome to ' PICTURE '@!'　　显示结果如图 2.2 。

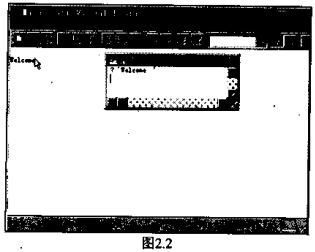


图2.2

?? 'Visual FoxPro'　　显示结果如图 2.3 。

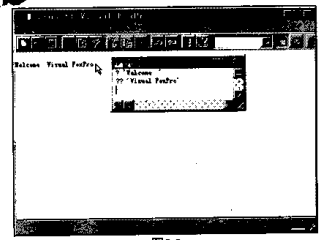


图2.3

2.1.2 wait

这个命令显示一条信息并暂停 Visual FoxPro 的执行，直到用户按下一个键或者单击一下鼠标。

语法：

WAIT

[cMessageText]

[TO VarName]

[WINDOW [AT nRow, nColumn]]

[NOWAIT]

[CLEAR | NOCLEAR]

[TIMEOUT nSeconds]

参数

cMessageText 指定要显示的用户信息。如果省略 cMessageText，则 Visual FoxPro 显示缺省信息。如果 cMessageText 是一个空字符串（""），则不显示任何信息，Visual FoxPro 等待按下一个键或者单击一下鼠标才继续执行程序。

TO VarName 把按下的键存储到一个变量或者一个数组元素中。如果 VarName 指定 的变量或数组元素不存在，则创建它。如果按下 ENTER 键或者其他不可打印字符键组合或者单击鼠标，则 VarName 中存储一个空字符串。

WINDOW 把信息显示在一个位于 Visual FoxPro 主窗口右上角的系统信息窗口中。这个窗口可以通过按下 CTRLor SHIFT 键来暂时隐藏。

AT nRow, nColumn 指定信息窗口在屏幕上显示的位置。

NOWAIT 指示在显示信息之后立即继续执行程序，不等待到信息从主窗口消失。

CLEAR 从程序的 Visual FoxPro 主窗口中删除一个 Visual FoxPro 系统窗口或者 WAIT 信息窗口。

NOCLEAR 指示将 WAIT 信息窗口保留在 Visual FoxPro 主窗口中，直到提出 WAIT CLEAR 或另一个 WAIT WINDOW 命令，或者显示一条 Visual FoxPro 系统信息。

TIMEOUT nSeconds 指定在结束 WAIT 前等待从键盘或鼠标输入的秒数。nSeconds 指定等待的秒数(允许分数秒)。这个子句必须出现在 WAIT 命令的最后。否则 Visual FoxPro 产生一条语法错误信息。

说明：当 WAIT 信息显示在 Visual FoxPro 中时，如果按下 Shift 或者 CTRL 键将隐藏包括 WAIT 信息窗口在内的所有窗口。

2.1.3 TextBox 控件

该控件创建一个文本框。

语法：

TextBox

说明：该控件创建一个文本框，在文本框中用户可以编辑内存变量、数组元素或者域的内容。所有的标准 Visual FoxPro 编辑特性，如剪切、拷贝和粘贴都在文本框中得到支持。如果文本框被用于编辑一个日期或日期时间值，则可以通过按“+”或者“-”来增减值。

用 InputMask 和 Format 属性来指定如何在文本框中输入和显示值。

2.2 内存变量

内存变量用于储存数据和进行数学运算。在 Visual FoxPro 中有三种不同的内存变量：一般变量、数组和系统变量。一般变量和数组由用户定义，系统变量是 Visual FoxPro 内建，不用再定义，可以直接使用。

2.2.1 内存变量的声明和赋值

1. 变量的声明

一般变量又可以分为私有变量、局部变量和全局变量。

一般变量可以不经定义直接使用，通过赋值来指定变量的类型。例如：

```
date={98/3/3}    && date 为日期型变量。
sex=.T.          && sex 为逻辑型变量。
```

为了封装数据，Visual FoxPro 支持把变量声明为私有变量。被声明为私有的变量只在声明它的过程中存在，当声明该变量的过程结束时，私有变量所占的内存被释放。外部过程无法访问私有变量。这在大型程序中对防止数据冲突起积极的作用。声明私有变量的语法是：

PRIVATE variable

局部变量类似于私有变量，但它只在声明它的过程中有效，在声明它的过程之外无法访问局部变量。声明局部变量的语法是：

LOCAL variable

全局变量在整个程序中有效，当需要在各个过程中传递数据时，使用全局变量比较方便，但是当程序比较大时，全局变量往往会造成极大的麻烦，产生不易发现的错误。声明

全局变量的语法是：

PUBLIC variable

Visual FoxPro 把声明的全局变量设为逻辑型的并赋给初值 .F.。

2. 变量的赋值

在 Visual FoxPro 中，有两种给变量赋值的方法：

（1） =

用“=”直接给变量赋值，例如：

salary=2000.00

sex=.T.

birthday={74/10/08}

在用“=”赋值的同时也就指定了变量的类型。

（2） STORE...TO

STORE...TO 命令与“=”功能类似，只是 STORE...TO 可以同时给多个变量赋值，例如：

STORE 2000.00 TO salary

STORE .T. To sex， married

2.2.2 数组的声明和赋值

1. 声明数组

使用数组可以很方便地操作大量的同一类型的数据。但是 Visual FoxPro 只支持一维和二维数组。与一般变量不同，数组在使用前必须先声明。声明数组的语法是：

DECLARE ArrayName1 （ nRows1 [,nColumns1]） [， ArrayName2 （ nRows2 [,nColumns2]）...]

2. 数组的赋值

给数组赋值其实就是给各个数组元素赋值，给数组元素赋值的语法同一般变量：

```
DECLARE  ABC （10）     &&  把 ABC 声明为一维数组，
STORE 30 TO ABC （3）   &&  把 30 赋给 ABC （3）
ABC （4）=40            &&  把 40 赋给 ABC （4）
```

2.2.3 内存变量的操作命令

对内存变量的操作非常丰富，包括：用内存变量保存数据、把内存变量中的数据保存在备注字段中、把内存变量保存在文件中、从内存中移去内存变量、恢复内存变量、将当前记录复制到内存变量中、输出内存变量的内容以及显示内存变量的内容等。

用内存变量保存数据： GATHER

GATHER 命令用一个数组或一组变量或一个对象的数据替换当前选中表中的当前记录中的数据。

语法：

GATHER FROM ArrayName | MEMVAR | NAME ObjectName
[FIELDS FieldList | FIELDS LIKE Skeleton | FIELDS EXCEPT Skeleton]
[MEMO]

参数：

FROM ArrayName 以指定数据替换当前记录的数据的数组。

MEMVAR 指定从中拷贝数据到当前记录的变量或者数组。数据从变量传送到与变量同名的字段中，如果不存在与字段同名的变量，则字段的内容保持不变。

NAME ObjectName 指定具有与表中的字段同名的属性的对象。各个字段的内容被同名属性的值替换。如果不存在与字段同名的属性，则字段的内容保持不变。

FIELDS FieldList 指定由数组元素或者变量内容替换内容的字段。

FIELDS LIKE Skeleton | FIELDS EXCEPT Skeleton 用户可以有选择地替换某些字段。这通过包含一个 LIKE 或 EXCEPT 子句来实现。如果包含一个 LIKE Skeleton，则 Visual FoxPro 替换匹配 Skeleton 的字段。如果包含 EXCEPT Skeleton，则 Visual FoxPro 替换所有不匹配*Skeleton 的字段。 Skeleton 支持通配符(* and ?)。例如，要替换所有以 c 开头的字段，只需使用以下命令：

GATHER FROM myarray FIELDS LIKE c*

MEMO 指定用变量或者数组元素的内容替换备注字段的内容。如果省略 MEMO，则在替换时跳过备注字段。在 GATHER 命令中总是忽略通用字段与图形字段。

范例：

本范例使用 GATHER 把数据拷贝到表的一个新字段中。在创建表 Test 之后，用 SCATTER 命令在该表中的字段的基础上创建一组变量。然后给每个字段赋值，并在表中添加一个空记录。

```
CREATE TABLE Test FREE ;
    (Object C(10), Color C(16), SqFt n(6,2))
SCATTER MEMVAR BLANK
m.Object="Box"
m.Color="Red"
m.SqFt=12.5
APPEND BLANK
GATHER MEMVAR
BROWSE
```

把内存变量中的数据保存在备注字段中或文件中： SAVE TO

SAVE TO 命令把当前的内存变量或数组保存在文件或备注字段中。

语法：

SAVE TO FileName | MEMO MemoFieldName[ALL LIKE Skeleton | ALL EXCEPT Skeleton]

参数：

FileName 指定存储变量或数组的文件。缺省的文件扩展名是.MEM.

MEMO MemoFieldName 指定存储变量或数组的备注字段。

ALL LIKE Skeleton 指示保存所有匹配 Skeleton 的变量和数组。Skeleton 可以包含通配符?和*。

ALL EXCEPT Skeleton 指示保存所有不匹配 Skeleton 的变量和数组。Skeleton 可以包含通配符?和*。

范例：

本范例创建两个变量，把它们保存在一个变量文件中，并在不清除已存在的变量的情况下恢复它们。

```
gnVal1 = 50
gcVal2 = 'Hello'
SAVE TO temp
CLEAR MEMORY
gdVal3 = DATE( )
RESTORE FROM temp ADDITIVE
CLEAR
DISPLAY MEMORY LIKE g*
```

从内存中移去内存变量：RELEASE

RELEASE 命令从内存中移去变量和数组。

语法：

```
RELEASE MemVarList
```

或者

```
RELEASE ALL [EXTENDED]
[LIKE Skeleton | EXCEPT Skeleton]
```

参数：

RELEASE MemVarList 指定从要内存中移去的变量和数组。变量与数组名之间用逗号隔开。

RELEASE ALL 从内存中移去所有的变量和数组。

EXTENDED 在程序中使用 RELEASE 命令时，该参数指示释放所有的全局变量。在程序中 RELEASE ALL, RELEASE ALL LIKE, 和 RELEASE ALL EXCEPT 不释放全局变量。

LIKE Skeleton | EXCEPT Skeleton 从内存中释放所有的匹配 skeleton ，或者不匹配 skeleton 的变量和数组。Skeleton 可以包含通配符?和*。

恢复内存变量：RESTORE FROM

RESTORE FROM 命令取回存放在变量文件或备注字段中的变量和数组并把它们放回内存。

语法：

```
RESTORE FROM FileName | MEMO MemoFieldName[ADDITIVE]
```

参数：

FileName 指定从中恢复变量和数组的变量文件。变量文件具有扩展名.MEM 。

MEMO MemoFieldName 指定从中恢复变量和数组的备注字段。

ADDITIVE 用来防止覆盖掉当前内存中的变量和数组。如果 ADDITIVE 加上已经存在的变量的数目超过了变量的限制，Visual FoxPro 将从变量文件或者备注字段中恢复尽可能多的变量和数组。

说明：

当在程序中使用 RESTORE FROM 命令时，所有的 PUBLIC 和 PRIVATE 变量和数组被恢复为 PRIVATE；所有的 LOCAL 变量和数组被恢复为 LOCAL。如果是在命令窗口中使用 RESTORE，则 PUBLIC 和 PRIVATE 变量和数组被恢复为 PUBLIC；LOCAL 变量与数组被恢复为 LOCAL。

如果不包括 ADDITIVE 参数，RESTORE FROM 命令清楚内存中所有的变量和数组。RESTORE FROM 不影响系统变量。

值得指出的是，对象类型的变量不能从变量文件或备注字段中恢复。

范例：

本范例创建两个变量，把它们保存在一个变量文件中，并在不清除已存在的变量的情况下恢复它们。

```
gnVal1 = 50
gcVal2 = 'Hello'
SAVE TO temp
CLEAR MEMORY
gdVal3 = DATE( )
RESTORE FROM temp ADDITIVE
CLEAR
DISPLAY MEMORY LIKE g*
```

将当前记录复制到内存变量中：SCATTER

SCATTER 命令把数据从当前记录中拷贝到一组变量或一个数组中。

语法：

SCATTER [FIELDS FieldNameLisl FIELDS LIKE Skeleton | FIELDS EXCEPT Skeleton] [MEMO]TO ArrayName | TO ArrayName BLANK | MEMVAR | MEMVAR BLANK| NAME ObjectName

参数：

FIELDS FieldNameList 指定将被复制到变量或数组中的字段。如果省略这一参数，则将所有的字段都保存。如果在该参数后有 MEMO 关键字，则字段表中可以有备注字段。SCATTER 命令总是忽略通用字段和图形字段。

FIELDS LIKE Skeleton | FIELDS EXCEPT Skeleton 用户可以通过在 SCATTER 命令中包含 LIKE 或 EXCEPT 子句来有选择地保存字段。如果包含 LIKE 子句，则所有匹配 LIKE 子句的字段被保存。如果包含 EXCEPT 子句，则所有不匹配 EXCEPT 子句的字段被保存。

MEMO 指示在字段列表中包含备注字段。

TO ArrayName 指定保存字段内容的数组。

TO ArrayName BLANK 创建一个与表中的字段具有相同的大小与类型的空元素的数组。

MEMVAR 把数据保存到一组变量中而不是保存到数组中。

MEMVAR BLANK 创建一组空的变量。每个变量被赋予与它的字段相同的名字、相同的数据类型和同样的大小。

NAME ObjectName 创建一个对象。该对象具有与表中的字段具有相同的名字的属性。对象的每个属性的值就是相应字段的内容。

说明：

SCATTER 和 COPY TO ARRAY 很相似。COPY TO ARRAY 命令把多个记录复制到一个数组。而 SCATTER 命令则把一个记录复制到一个数组或者一组变量中。如果指定的变量或数组不存在，则 SCATTER 命令自动创建它们

范例：

例 1：本范例用 SCATTER 命令基于 test 表中的字段创建一组变量。然后为每个字段赋一个值并在表中加入一个空记录。然后用 GATHER 命令把数据复制到表中。

```
CREATE TABLE Test FREE ;
    (Object C(10), Color C(16), SqFt n(6,2))
SCATTER MEMVAR BLANK
m.Object="Box"
m.Color="Red"
m.SqFt=12.5
APPEND BLANK
GATHER MEMVAR
BROWSE
```

例 2：本范例用带有 NAME 子句 SCATTER 命令创建一个对象，该对象具有与表中的字段相对应的属性。然后给这个对象的属性赋值，并在表中增加一个空记录，然后用带有 NAME 子句的 GATHER 命令把对象中的数据复制到表的新记录中。

```
CREATE TABLE Test FREE ;
    (Object C(10), Color C(16), SqFt n(6,2))
SCATTER NAME oTest BLANK
oTest.Object="Box"
oTest.Color="Red"
oTest.SqFt=12.5
APPEND BLANK
GATHER NAME oTest
RELEASE oTest
BROWSE
```

输出内存变量的内容以及显示内存变量的内容 ： TEXT...ENDTEXT 与 DISPLAY MEMORY

TEXT...ENDTEXT 命令输出内存变量的内容。此外，该命令还可用于输出文本行和表达式与函数的结果。

语法

```
TEXT
TextLines
ENDTEXT
```

参数：

TextLines 指定送到当前输出设备的文本。TextLines 可以由文本、内存变量、数组元素、表达式、函数等组成。

说明：

本命令把处于 TEXT 和 ENDTEXT 之间的文本输出到 Visual FoxPro 主窗口、用户定义的窗口、打印机或文本文件。

范例：

本范例展示 SET TEXTMERGE, SET TEXTMERGE DELIMITERS, TEXT ... ENDTEXT 等命令以及系统变量_TEXT 的使用和设置。本范例创建一个名为 NAMES.TXT 的文件，并把这个文件的句柄保存在系统变量_TEXT 中。如果不能创建 NAMES.TXT 文件则程序终止。如果成功创建 NAME.TXT 文件，则程序打开 customer 表，并把表中的前 10 个合同的名字和函数的结果输出到 NAMES.TXT 文件。然后用 MODIFY FILE 命令打开该文件。

```
CLEAR
CLOSE DATABASES
SET TALK OFF
SET TEXTMERGE ON
STORE FCREATE('names.txt') TO _TEXT
IF _TEXT = -1
    WAIT WINDOW 'Cannot create an output file; press a key to exit'
    CANCEL
ENDIF
CLOSE DATABASES
OPEN DATABASE (HOME( ) + 'samples\data\testdata')
USE customer   && Opens Customer table
TEXT
            CONTACT NAMES
    <<DATE( )>>          <<TIME( )>>
ENDTEXT
WAIT WINDOW 'Press a key to generate the first ten names'
SCAN NEXT 10
    TEXT
        <<contact>>
    ENDTEXT
ENDSCAN
CLOSE ALL   && Close the text file and the table
MODIFY FILE names.txt
```

ERASE names.txt

2.2.4 系统内存变量的操作使用

系统变量是 Visual FoxPro 自动创建和维护的内建变量。它们缺省地是 PUBLIC，但用户可以把它们声明为 PRIVATE。

通过对系统内存变量的操作（改变其值）可以在程序中轻松地实现复杂的功能。下表列出了 Visual FoxPro 中的 5 种类型的系统内存变量和它们的形式特点。

表2.1 Visual FoxPro中的5种系统内存变量

变量类型	描述	形式
C	字符型	cExpression
D	日期型	dExpression
L	逻辑型	lExpression
N	数值型	nExpression
O	对象型	oExpression

2.3 程序控制结构

最一般的情况下，程序从头执行到尾，但是这种程序的功能是十分有限的。Visual FoxPro 中有一类特殊的命令，它们可以反复执行其他命令或函数，并决定这些命令或函数何时执行以及执行次数。这类特殊命令可用来实现两种主要的程序结构：条件分支和循环，它们在程序设计过程中作用很大。

2.3.1 分支判断结构：IF...ELSE...ENDIF

IF...ELSE...ENDIF 结构在逻辑表达式为真时执行一组命令。

语法：

IF lExpression [THEN]Commands[ELSECommands]ENDIF

参数：

lExpression 指示被求值的逻辑表达式。如果 lExpression 的值为真(.T.)，则所有处于 IF 或者 THEN 之后并在 ELSE 或者 ENDIF 之前的命令被执行。如果 lExpression 为假，并且分支结构中包含 ELSE，则所有处于 ELSE 和 ENDIF 之间的命令被执行。如果 lExpression 为假，并且分支结构中不包含 ELSE，则所有处于 IF 和 ENDIF 之间的命令被忽略，程序在 ENDIF 之后的第一条命令处继续执行。

说明：IF ... ENDIF 块可以嵌套。

范例：

本范例假设一位有 10,000 个雇员的老板要给年薪大于或等于 $30,000 的雇员涨 3% 的工资，给年薪低于 $30,000 的雇员涨 6% 的工资。下面的示例程序将完成这一任务。本程序假定在当前工作区中已打开了一个表，此表有一个名为 salary 的数值型字段。

```
SCAN
IF salary >= 30000.00
REPLACE salary WITH salary * 1.03
ELSE
    REPLACE salary WITH salary * 1.06
ENDIF
ENDSCAN
```

2.3.2 多重判断结构

若要判断多种可能的情况， DO CASE ... ENDCASE 结构将比用多个 IF 语句更有效，并且易于跟踪调试。

DO CASE ... ENDCASE 结构执行其条件表达式为真的第一个命令。

语法：

```
DO CASE
    CASE lExpression1
    Commands
[CASE lExpression2
    Commands
...
CASE lExpressionN
    Commands]
[OTHERWISE
    Commands]
ENDCASE
```

参数：

CASE lExpression1 Commands ... 当遇到第一个为真的 CASE 表达式时，跟随在该表达式后的命令组被执行，直到遇到下一个 CASE 或 ENDCASE 。

如果一个 CASE 表达式为假，则该表达式后的命令被忽略。

在 CASE 结构中，只有第一个为真的 CASE 表达式后的命令被执行，其后的为真的 CASE 表达式后的命令也被忽略。

OTHERWISE Commands 如果所有的 CASE 表达式都为假，则由 OTHERWISE 决定是否执行一组附加的命令；如果包含 OTHERWISE ，则 OTHERWISE 后的命令被执行；然后程序跳到 ENDCASE 后继续执行。如果省略 OTHERWISE ，则程序直接跳到 ENDCASE 后的第一条命令继续执行。

说明： DO CASE 按照逻辑表达式的值来执行一组 Visual FoxPro 命令。在执行 DO CASE 结构时，连续对逻辑表达式求值，逻辑表达式的只决定哪一组命令被执行。

范例：

在本范例中，Visual FoxPro 对每个 CASE 子句求值，直到找到包含 MONTH 变量的列表。然后把相应的字符串存放在变量 rpt_title 中并退出 DO CASE 结构。

```
STORE CMONTH(DATE( )) TO month   && The month today
DO CASE   && Begins loop
    CASE INLIST(month,'January','February','March')
        STORE 'First Quarter Earnings' TO rpt_title
    CASE INLIST(month,'April','May','June')
        STORE 'Second Quarter Earnings' TO rpt_title
    CASE INLIST(month,'July','August','September')
        STORE 'Third Quarter Earnings' TO rpt_title
    OTHERWISE
        STORE 'Fourth Quarter Earnings' TO rpt_title
ENDCASE   && Ends loop
WAIT WINDOW rpt_title NOWAIT
```

2.3.3 循环结构

循环结构可以按照需要任意次地重复执行一行或多行代码。在 Visual FoxPro 中有三种循环语句：

SCAN ... ENDSCAN

FOR ... ENDFOR

DO WHILE ... ENDDO

1.SCAN ... ENDSCAN

若要对表中全部记录执行某一操作，可以使用 SCAN。随着记录指针的移动，SCAN 循环允许对每条记录执行相同的代码块。

SCAN ... ENDSCAN 结构从选中的表中从头至尾移动记录的指针并对满足条件的记录执行一组命令。

语法：

```
SCAN [NOOPTIMIZE]
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    [Commands]
    [LOOP]
    [EXIT]
ENDSCAN
```

参数：

NOOPTIMIZE 禁止对 SCAN 使用 Rushmore 优化。

Scope 指定被扫描的记录的范围。只有处在指定的范围内的记录才被扫描。范围子句包括：ALL, NEXT nRecords, RECORD nRecordNumber 和 REST。默认的范围是 ALL。

FOR lExpression1 只对那些 lExpression1 为真的记录执行命令。用户可以通过包含

FOR 子句来过滤掉不想被扫描的记录。

WHILE lExpression2 指定一个条件，只要 lExpression2 为真，就执行命令。

Commands 指定要执行 Visual FoxPro 命令。

LOOP 直接把控制返回到 SCAN 。 LOOP 可以放置在 SCAN 和 ENDSCAN 之间的任何地方。

EXIT 把程序控制从 SCAN ... ENDSCAN 循环中转移到 ENDSCAN 后的第一条命令。EXIT 可以放置在 SCAN 和 ENDSCAN 之间的任何地方。

ENDSCAN 指示 SCAN 过程的结束。

说明： SCAN 自动地把记录指针增加到下一个满足特定条件的记录并执行命令块。

范例：

本范例用 SCAN ... ENDSCAN 循环显示所有总部在法国的公司：

```
CLOSE DATABASES
OPEN DATABASE (HOME( ) + 'samples\data\testdata')
USE customer   && Opens Customer table
CLEAR
SCAN FOR UPPER(country) = 'France'
    ? contact, company, city
ENDSCAN
```

2.FOR...ENDFOR

若知道循环次数，则可以使用 FOR 循环。

语法：

```
FOR Var = nInitialValue TO nFinalValue [STEP nIncrement]
Commands
    [EXIT]
    [LOOP]
ENDFOR | NEXT
```

参数：

Var 指定作为计数器的变量或者数组元素。

nInitialValue TO nFinalValue nInitialValue 给出计数器的初始值， nFinalValue 给出计数器的终值。

STEP nIncrement nIncrement 给出计数器增加或者减少的步长。

Commands 指定被执行的 Visual FoxPro 命令。

EXIT 把控制从 FOR ... ENDFOR 循环内转移到 I ENDFOR 后的第一条命令。可以把 EXIT 放在 FOR 和 ENDFOR 之间的任何地方。

LOOP 把控制返回到 FOR 子句，忽略掉 LOOP 和 ENDFOR 之间的命令。计数器照常增减。 LOOP 可以放在 FOR 和 ENDFOR 之间的任何地方。

3.DO WHILE...ENDDO

若要在某一条件满足时结束循环，可以使用 DO WHILE 语句。使用 DO WHILE 结构可以不清楚循环的次数，但应知道什么时候结束循环的执行。

语法：

```
DO WHILE lExpression
    Commands
    [LOOP]
    [EXIT]
ENDDO
```

参数：

lExpression 指定一个逻辑表达式，该逻辑表达式的值决定是否执行 DO WHILE 和 ENDDO 之间的命令。只要 lExpression 的值为真，就执行命令。

Commands 指定被执行的 Visual FoxPro 命令。

LOOP 把程序控制直接返回到 DO WHILE 。LOOP 可以放在 DO WHILE 和 ENDDO 之间的任何地方。

EXIT 把程序控制转移到 ENDDO 后的第一条命令。EXIT 可以放在 DO WHILE 和 ENDDO 之间的任何地方。

说明：只要表达式 lExpression 为真，DO WHILE 和 ENDDO 之间的命令就被执行。每个 DO WHILE 语句必须具有一个相应的 ENDDO 语句。

范例：

本范例假定一张表包含员工的姓名和姓名缩写，需要使用姓名缩写来查找某些人员的信息。当向表中加入一个姓名缩写时，如果表中已有了一个相同的姓名缩写，就会遇到问题。

通过给姓名缩写加一个数字的方法，可以解决这个问题。例如，Michael Susan 的缩写为 MS，Mary Sun 的姓名缩写与之相同，则将其替换为 MS1 。以后，Mike Smith 的姓名缩写便是 MS2，依次类推。这个过程可以用一个 DO WHILE 循环实现。

```
nHere = RECNO()
cInitials = LEFT(firstname,1) + LEFT(lastname,1)nSuffix = 0
LOCATE FOR person_id = cInitials
DO WHILE FOUND( )
    nSuffix = nSuffix + 1
    cInitials =LEFT(cInitials,2)+ ALLTRIM(STR(nSuffix))
    CONTINUE
ENDDO
GOTO nHere
REPLACE person_id WITH cInitials
```

因为无法预知可能匹配的标识代码会被找到多少次，因此这里要用 DO WHILE 循环。

2.4 命令文件的建立与执行

Visual FoxPro 程序是包含一系列命令的文本文件，由以下 5 个部分组成：

命令：操作数据库的各种指令

变量：用于存放数据。

函数： Visual FoxPro 内建或者用户定义的，可以完成一定功能的命令集。

流程控制语句：控制程序的流程的语句。

预处理器：给编译器提供信息的语句。

2.4.1 命令文件的建立与编辑

1. 建立命令文件

在 Visual FoxPro 中，可以通过以下途径创建程序：

(1)在“项目管理器”中，选定“代码”标签中的“程序”项。

(2)选择“新建”命令。

或者

(1)在“文件”菜单中选择“新建”命令。

(2)在“新建”对话框中选择“程序”。

(3)单击“新建文件”按钮。

或者

在“命令”窗口中，键入：

MODIFY COMMAND

Visual FoxPro 将打开一个称为“程序 1 ”的新窗口，这时就可以键入应用程序了（见图 2.4 ）。

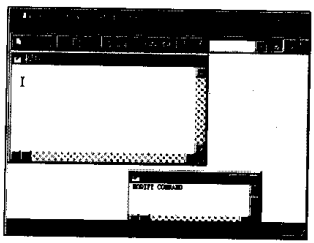


图2.4 程序1窗口

创建程序后，请注意保存，保存文件的方法是：在“文件”菜单中选择“保存”命令。

2. 编辑命令文件

要编辑命令文件，首先打开想要修改的程序；

若程序包含在一项目中，则在“项目管理器”中选定它并选择“修改”命令。

或者

在“文件”菜单中选择“打开”命令，这时弹出一个包含文件列表的对话框。在“文件类型”列表框中选择“程序”，然后在“文件名”列表中选定要修改的程序，按下“确

定”按钮。

或者

在“命令”窗口中按如下方式键入要修改的程序名。

MODIFY COMMAND myprogram

或者

在“命令”窗口中键入：

MODIFY COMMAND ?

将弹出打开文件列表对话框，在文件列表中选择要修改的程序，选择“打开”。

打开文件之后便可进行编辑，编辑完后不要忘记保存。

2.4.2 命令文件的执行

程序创建之后便可运行。

（1）若程序包含在一个项目中，则在“项目管理器”中选定它并选择“运行”命令。

（2）或者在“程序”菜单中选择“运行”菜单项。在程序列表中，选择想要运行的程序，然后单击“运行”按钮。

或者在“命令”窗口中，按如下方式键入 DO 以及要运行的程序名：

DO myprogram 都可以运行命令文件。

2.5 过程和自定义函数

2.5.1 过程和函数概述

过程和函数可以将常用代码集中在一起，供应用程序在需要时调用。这样做提高了程序代码的可读性和可维护性，使您可以不必对程序进行多次修改，只变动一个地方就足够了。

Visual FoxPro 的过程如下所示：

PROCEDURE myproc

ENDPROC

习惯上，过程是为完成某个操作而编写的代码，函数用来计算并返回一个值。在 Visual FoxPro 中，这二者区别不大。

FUNCTION myfunc

```
ENDFUNC
```

可以将过程和函数保存在单独的程序文件中，也可放在一般程序的结尾。不能把可执行的主程序代码放在过程或函数之后。

若将过程或函数放在单独的程序文件中，可以在应用程序中使用 SET PROCEDURE TO 命令访问它们。例如，保存过程或函数的文件名为 FUNPROC.PRG，可在“命令”窗口中使用下面的命令调用它们：

```
SET PROCEDURE TO funproc.prg
```

2.5.2 过程和函数的执行

在程序中有两种调用过程或函数的方式：

（1）使用 DO 命令，例如：

```
DO myproc
```

（2）在函数名后加上一对小括号，例如：

```
myfunc( )
```

如果向过程或函数发送值或接收它们的返回值，则两种方式都可以加以扩展。若要向过程或函数传递值，可以使用参数。例如，下面的过程接收一个参数：

```
PROCEDURE myproc( cString )
    MESSAGEBOX ("myproc" + cString)
ENDPROC
```

在过程或函数定义行的括号中包含参数，表明该参数的作用范围仅为该过程或函数，例如 PROCEDURE myproc (cString)。也可以使用 LPARAMETERS，让函数或过程来接收局部作用域的参数。

在函数中，参数以同样方式工作。要向函数或过程传递参数值，可以使用字符串或包含字符串的变量。例如：

```
DO myproc WITH cTestString
DO myproc WITH "test string"
myfunc("test string")
myfunc( cTestString )
```

若在调用过程或函数时不是用 Do 命令，则 UDFPARMS 设置控制如何传递参数。在默认条件下，UDFPARMS 设置为 VALUE，即传递参数的副本。在使用 DO 命令调用过程或函数时，实际的参数被传递（以引用方式传递参数）。这时，在过程或函数运行过程中，参数值的任何变化都将反映到原始数据中，而与 UDFPARMS 状态无关。可以向过程或函数传递多个参数，参数之间用逗号分开。例如，下面的过程有三个参数：日期、字符串和数字。

```
PROCEDURE myproc( dDate, cString, nTimesToPrint )
    FOR nCnt = 1 to nTimesToPrint
```

```
        ? DTOC(dDate) + " " + cString + " " + STR(nCnt)
    ENDFOR
ENDPROC
```

下面的代码将调用这个过程：

```
DO myproc WITH DATE(), "Hello World", 10
```

函数默认的返回值是“真”(.T.)，但可以使用 RETURN 命令返回任意值。例如，下面的函数返回比参数值晚一周的日期。

```
FUNCTION plus1weeks
PARAMETERS dDate
    RETURN dDate + 7
ENDFUNC
```

下面的代码将此函数的返回值保存在一个变量中：

```
dDeadLine = plus1weeks(DATE())
```

要保存和显示函数的返回值，可以仿照以下代码编写：

```
var = myfunc( )
? myfunc( )
```

检验发送的参数是否为过程或函数所要接收的数据是一种良好的设计习惯，可以使用 TYPE( ) 和 PARAMETERS( )函数来检验参数的类型和个数。

例如，上面提到的函数需要接收一个日期型参数，可以用 TYPE( ) 函数检验所接收的参数类型是否正确。

```
FUNCTION plus1weeks( dDate )
    IF TYPE("dDate") = "D"
        RETURN dDate + 7
    ELSE
        MESSAGEBOX( "必须传递一个日期型数据！" )
        RETURN {}
    ENDIF
ENDFUNC
```

当过程所接收的参数多于所需要的个数时， Visual FoxPro 将产生一个错误信息。例如，如果您只列出了两个参数，却使用三个参数调用它，这时将会出错。但如果过程接收的参数个数小于所要求的数目，则 Visual FoxPro 仅将余下的参数赋初值为“假”(.F.)，而不产生出错信息。下面的过程展示确认参数个数是否正确的方法。

```
PROCEDURE SaveValue( cStoreTo, cNewVal, llsInTable )
    IF PARAMETERS( ) < 3
        MESSAGEBOX( "Too few parameters passed." )
        RETURN .F.
    ENDIF
    IF llsInTable
        REPLACE (cStoreTo) WITH (cNewVal)
```

```
    ELSE
        &cStoreTo = cNewVal
    ENDIF
    RETURN .T.
ENDPROC
```

2.6 项目管理器的使用

“项目管理器”是 Visual FoxPro 中处理数据和对象的主要组织工具。所谓“项目”是指文件、数据、文档和 Visual FoxPro 对象的集合，项目被保存在带有 .PJX 扩展名的文件中。

“项目管理器”是 Visual FoxPro 的灵魂。在建立表、数据库、查询、表单、报表以及应用程序时，可以用“项目管理器”来组织和管理文件。

通过把已有的 Visual FoxPro .DBF 文件添加到一个新的项目中，可以创建一个项目。“项目管理器”为数据提供了一个组织良好的分层结构视图。若要处理项目中某一特定类型的文件或对象，可选择相应的标签。

在建立表和数据库，以及创建表单、查询、视图和报表时，用户所要处理的主要是“数据”和“文档”标签中的内容。

“项目管理器”中的“数据”标签包含了一个项目中的所有数据：数据库、自由表、查询和视图。

数据库是表的集合，一般通过公共字段彼此关联。使用“数据库设计器”可以创建一个数据库，数据库文件的扩展名为 .DBC 。

自由表存贮在以 .DBF 为扩展名的文件中，它不是数据库的组成部分。

查询是检查存贮在表中的特定信息的一种结构化方法。利用“查询设计器”，可以设置查询的格式，该查询将按照您输入的规则从表中提取记录。查询被保存为带 .QPR 扩展名的文件。

视图是特殊的查询，通过更改由查询返回的记录，可以用视图访问远程数据或更新数据源。视图只能存在于数据库中，它不是独立的文件。

“项目管理器”中的“文档”标签中包含了处理数据时所用的全部文档：输入和查看数据所用的表单，以及打印表和查询结果所用的报表及标签。

表单用于显示和编辑表的内容。

报表是一种文件，它告诉 Visual FoxPro 如何设置查询来从表中提取结果，以及如何将它们打印出来。

其余标签（如“类”、“代码”及“其他”）主要用于为最终用户创建应用程序。

“项目管理器”中的项是以类似于大纲的结构来组织的，可以将其展开或折叠，以便查看不同层次中的详细内容。

如果项目中具有一个以上某一类型的项，其类型符号旁边会出现一个 + 号。单击符号旁边的 + 号可以显示项目中该类型项的名称，单击项名旁边的 + 号可以看到该项的组件。

例如，单击“自由表”符号旁边的 + 号，可以看到项目中自由表的名称；单击表名旁边的 + 号，可以看到表中的字段名和索引名（见图 2.5）。

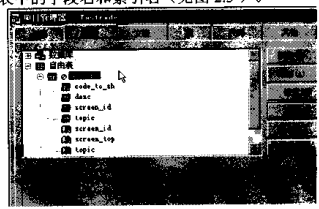


图2.5

大纲显示了项目中不同层次内的详细内容。若要折叠已展开的列表，可单击列表旁边的 - 号。

要想使用“项目管理器”，必须在其中添加已有的文件或者用它来创建新的文件。例如，如果想把一些已有的扩展名为 .DBF 的表添加到项目中，只需在“数据”标签中选择“自由表”，然后用“添加”按钮把它们添加到项目中。

若要在项目中加入文件，选择要添加项的类型，然后选择“添加”，弹出“打开”对话框，在“打开”对话框中，选择要添加的文件名，然后选择“确定”。

若要从项目中移去文件，选定要移去的内容，然后选择“移去”。

如果要从计算机中删除文件，请选择“删除”。

“项目管理器”简化了创建和修改文件的过程。只需选定要创建或修改的文件类型，然后选择“新建”或“修改”按钮，Visual FoxPro 将显示与所选文件类型相应的设计工具。

若要创建添加到“项目管理器”中的文件，先选定要创建的文件类型。然后单击“新建”按钮。对于某些项，可以利用向导来创建文件。

若要修改文件，先选定要修改的文件，然后单击“修改”按钮。

例如，要修改一个表，先选定表的名称，然后选择“修改”按钮，该表便显示在“表设计器”中（见图 2.6）。

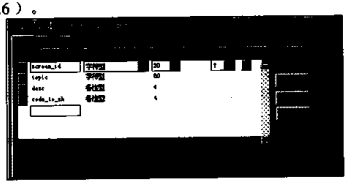


图2.6

创建或添加新的文件时，可以为文件加上说明。文件被选定时，说明将显示在“项目管理器”的底部。

要为文件添加说明，按如下步骤进行：

（1）在“项目管理器”中选定文件。

（2）从“项目”菜单中选择“编辑说明”。

（3）在“说明”对话框中键入对文件的说明。

（4）单击“确定”按钮。

从“项目管理器”中可以浏览项目中表的内容。若要浏览表，先选择“数据”标签。然后选中要浏览的表，最后单击“浏览”按钮，就可以浏览选中的表了。

通过“项目管理器”还可以实现项目间共享文件。通过与其他项目共享文件，可以重用在其它项目开发上的工作成果。被共享的文件并未复制，项目只储存了对该文件的引用。文件可同时与不同的项目连接。

若要在项目之间共享文件，步骤如下：

（1）在 Visual FoxPro 中，打开要共享文件的两个项目。

（2）在包含该文件的“项目管理器”中，选择该文件。

（3）拖动该文件到另一个的项目容器中。

定制“项目管理器”

用户可以定制可视工作区域，方法是改变“项目管理器”的外观或设置在“项目管理器”中双击运行的文件。

“项目管理器”显示为一个独立的窗口。可以移动它的位置、改变它尺寸或者将它折叠起来只显示标签。

要移动“项目管理器”，将鼠标指针指向标题栏，然后将“项目管理器”拖到屏幕上的其他位置。

要改变“项目管理器”窗口的大小，将鼠标指针指向“项目管理器”窗口的顶端、底端、两边或角上，拖动鼠标即可扩大或缩小它的尺寸。

要折叠“项目管理器”，单击右上角的上箭头。在折叠情况下只显示标签。

折叠后，可以很容易地将“项目管理器”还原为通常大小。要还原“项目管理器”，单击右上角的下箭头。

折叠“项目管理器”后，可以拖开标签，并根据需要重新安排它们的位置。拖下某一标签后，它可以在 Visual FoxPro 的主窗口中独立移动。若要拖开某一标签，先折叠“项目管理器”。然后选定要拖开的标签，将它拖离“项目管理器”（见图 2.7）。

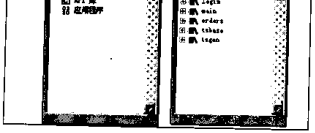


图2.7

当标签处于浮动状态时，通过在标签中单击鼠标右键可以访问“项目”菜单中的选项。

如果希望标签始终显示在屏幕的最表层，可以单击标签上的图钉图标，这样，该标签就会一直保留在其他 Visual FoxPro 窗口的上面。再次单击图钉图标可以取消标签的“顶层显示”设置。

若要还原标签，单击标签上的“关闭”按钮，或者将标签拖回到“项目管理器”。

还可以停放“项目管理器”，使它象工具栏一样显示在 Visual FoxPro 主窗口的顶部。若要停放“项目管理器”，将“项目管理器”拖到 Visual FoxPro 主窗口的顶部就可以了。

停放“项目管理器”后，它就变成窗口工具栏区域的一部分（图 2.8 ）。“项目管理器”处于停放状态时，不能将其展开，但是可以单击每个标签来进行相应的操作。对于停放的“项目管理器”，同样可以从中拖开标签。

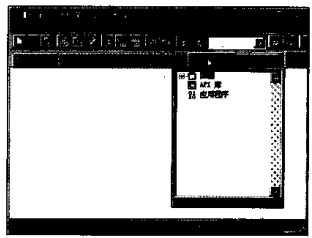


图2.8

第三章　FoxPro 的数据库操作命令

如果你曾经使用过 DOS 环境下的数据库 xBase 或早期版本的 FoxPro 的话，相信你对命令该是再熟悉不过了。在 VFP 中，我们似乎已经抛弃了纯粹的命令操作方式，而代之以方便的菜单操作。同时 VFP 提供了丰富的向导和工具，用户几乎可以在系统的帮助下完成任何操作。

但同时我们看到在 VFP 中，依然保留了“命令”窗口。这一保留并不止是为了和以前版本的兼容，还更是以为命令方式有其自身的明显优势。尤其对于较为熟练的程序员来说，命令方式更是其一种方便的选择。

VFP 的命令往往带有许多参数，使用起来更具威力和弹性。使得很多在菜单及工具中需很多时间和操作才能完成的要求，以命令方式可以方便地实现。例如我们要筛选出数据表“成绩表”中所有“语文”成绩大于 80 分的记录项，应用菜单、工具要有以下步骤：

（1）在工程管理器中选取“成绩表”，按“浏览”按钮；

（2）选取菜单项“表/属性”；

（3）按“数据过滤器”右方按钮，设置过滤条件；

（4）浏览结果。

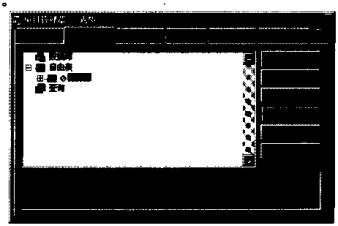


图 3.1　在工程管理器中选取“成绩表”

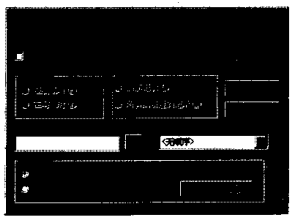


图 3.2　选取菜单项“表/属性”

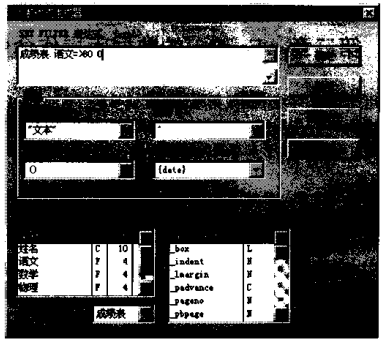


图 3.3 设置筛选条件

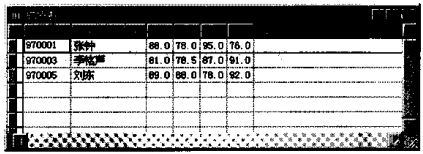


图 3.4 浏览结果

对于以上同样的要求，用命令方式只要在“命令”窗口中，键入以下命令：

USE 成绩表

SET FILTER TO 成绩表.语文>=80.0

BROWSE

执行了以上命令，我们可以得到同样的结果。整个过程更方便更快捷。可见命令方式有其独特的优势。

将 VFP 的命令联系起来，加上适当的程序语法控制，就可以设计出 VFP 的程序了。可见 VFP 的命令还是其程序的基本组成要素，因此对 VFP 命令的熟悉和了解是 VFP 程序设计的基础。

可见若单单学习菜单、工具和向导的操作，还只能是一个简单的使用者。要想成为 VFP 高超的使用者和程序设计者，就需要学习 VFP 的各种命令的使用。以此为基础，我们才能设计出 VFP 的应用程序。在本章中，我们将分别介绍数据库、数据表和记录的操作命令，以此使读者对各种操作命令有一个全面的了解。

3.1 数据库的操作命令

数据库的操作一般包括数据库的创建、打开数据库、设定当前数据库、关闭数据库和删除数据库操作。在下面的一节中，我们对数据库的一般操作做一个较为全面的说明。希望读者可以在参阅本节的同时，自己亲自尝试一下各个命令的应用，这样可以尽快掌握数据库的操作。

3.1.1 创建数据库

CREATE DATABASE [DatabaseName | ?]

该命令用于创建并打开一个数据库。

参数 DatabaseName 指定新建数据库的名称。若 SAFETY 设置为 ON，则当用户指定的数据库已经存在时，VFP 将显示一个警告。

参数？将弹出一个对话框，要求用户输入所要创建的数据库名称。

不带任何参数的命令也弹出对话框，要求用户输入所要创建的数据库名称。

图 3.5 数据库已经存在

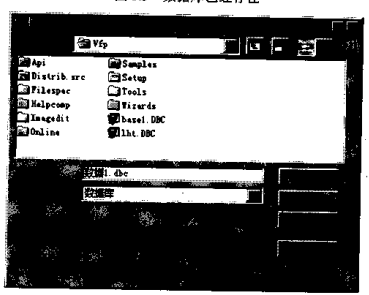


图 3.6 提示输入要创建的数据库名称

创建的数据库文件 VFP 自动将其保存于用户指定的路径处，其文件名后缀为 .DBC；同时该数据库中的备注字段内容将单独保存在与数据库文件同名，后缀为 .DCT 的文件中；该数据库对应的索引文件后缀为 .DCX。

一个数据库的打开可以有独占方式（EXCLUSIVE）和共享方式（SHARED）。独占方式的打开时，其他用户对该数据库的访问请求将被拒绝并报错；而以共享方式打开的数据库允许其他用户同时对其进行访问。当一个数据库被创建后，同时被以独占方式打开，用户不必再应用 OPEN DATABASE 命令重新打开。

例子：

以下的例子创建一个名为 Base 的数据库。DISPLAY DATABASES 命令用来显示有关数据库的信息。

```
CREATE DATABASE   Base                    && 创建数据库
CLEAR                                     && 清除屏幕
DISPLAY DATABASES                         && 显示数据库信息
```

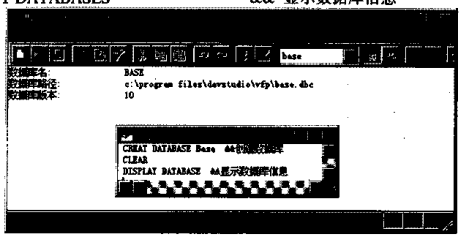


图 3.7 显示数据表和数据库信息

3.1.2 打开数据库

经 CREAT DATABASE 创建的数据库同时就被打开。但对于其他已经存在的数据库，用户还经常需要显式的打开。

```
OPEN DATABASE [FileName | ?]
    [EXCLUSIVE | SHARED]
    [NOUPDATE]
    [VALIDATE]
```

该命令用语打开一个已经存在的数据库。

参数 FileName 指定要打开的数据库名称。缺省的文件名后缀为 .DBC 。若指定一个不存在的数据库或者路径错误，VFP 都将给出出错信息。

参数 ？将打开一个对话框，用户可以输入文件名。若该文件存在将被打开，若不存在该文件将给出出错信息。

没有指定 FileName 时，VFP 将询问要打开的文件名。

参数 EXCLUSIVE | SHARED 指定数据库打开的方式。以 EXCLUSIVE（独占方式）打开的数据库其他用户便无法再次打开。以 SHARED（共享方式）打开的数据库其他用户也可以在此打开。该参数在缺省时，打开的方式有 SET EXCLUSIVE 的值决定。

参数 NOUPDATE 指定数据库以只读方式打开。在此种方式下，用户无法对数据库做任何修改。缺省时数据库以读写方式打开。

注意：数据库中的表并不受该参数的影响。用户可以通过在 USE 命令中 NOUPDATE 参数将表设置为只读方式。

参数 VALIDATE 使 VFP 在打开数据库时确认引用的有效性。

图 3.8 打开数据库时的出错信息

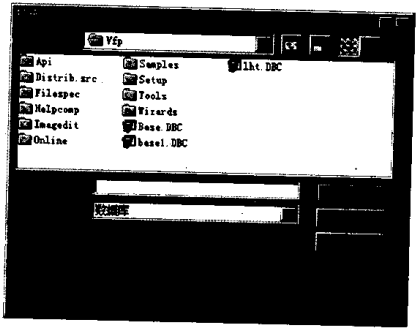


图 3.9 打开数据库对话框

当数据库被打开时，数据库中所有的表都成为可用的。但这些表并没有被打开，用户可以用 USE 命令打开数据库中的表。

用户无法应用 OPEN DATABASE 命令打开别人正以独占方式打开的数据库。

例子：

下面的命令打开一个名为 Base 的数据库。DISPLAY DATABASE 命令显示该数据库的信息。

```
CLOSE DATABASES                 && 关闭数据库
OPEN DATABASE Base              && 打开数据库
DISPLAY DATABASE                && 显示数据库信息
```

3.1.3 设定当前数据库

我们可以同时打开许多数据库，但同时只能指定一个数据库为当前数据库。VFP 的许多操作命令都是对设置为当前数据库的数据库进行操作的。

SET DATABASE TO [DatabaseName]

该命令用来设定当前数据库。

参数 DatabaseName 指定一个处于打开状态的数据库为当前数据库。若该参数被缺省，则没有数据库被指定为当前数据库。此后的操作将没有缺省的数据库设置。如以下命令将导致 VFP 询问数据库名称：

```
OPEN DATABASE Base                  && 打开数据库 Base
SET DATABASE TO                     && 没有数据库被设置为当前数据库
DISPLAY DATABASE                    && 显示数据库信息，此时 VFP 将产生询问
```

VFP 可以通过 OPEN DATABASE 命令同时打开多个数据库，但同时只有一个数据库可以是当前数据库（通常最后打开的数据库为当前数据库）。通过该命令可以将指定的数据库设置为当前数据库。以后的各种操作命令在缺省时均对该数据库作用。

例子：

下面的例子创建两个数据库，通过 SET DATABASE TO 命令来改变当前数据库，通过 DISPLAY DATABASE 命令来显示数据库信息。

```
CLEAR                               && 清屏
CREAT DATABASE Dbc1                 && 创建数据库 Dbc1
CREAT DATABASE Dbc2                 && 创建数据库 Dbc2
DISPLAY DATABASE                    && 显示数据库信息，此时 Dbc2 为当前数据库
SET DATABASE TO Dbc1                && 设置 Dbc1 为当前数据库
DISPLAY DATABASE                    && 显示数据库信息，此时 Dbc1 为当前数据库
SET DATABASE TO Dbc2                && 设置 Dbc2 为当前数据库
DISPLAY DATABASE                    && 显示数据库信息，此时 Dbc2 为当前数据库
```

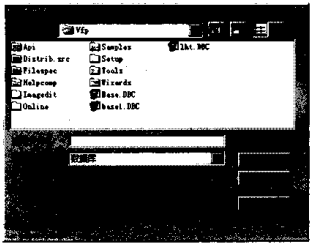


图 3.10 询问要显示的数据库

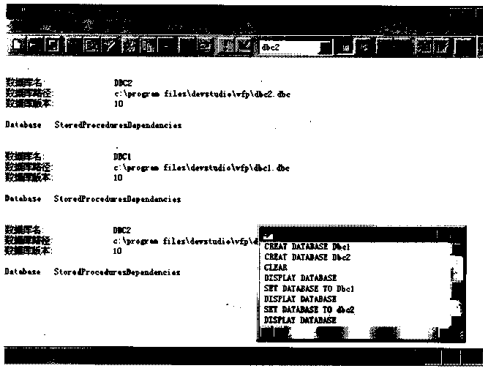


图 3.11 改变设置当前数据库

3.1.4 关闭数据库

CLOSE DATABASE[ALL]

该命令用于关闭设置为当前的数据库。

参数 ALL 指定关闭所有打开的数据库。

详细的使用用户可以参考命令 CLOSE 。这里限于篇幅，只列出其语法：

CLOSE

[ALL | ALTERNATE | DATABASES [ALL] | DEBUGGER

| FORMAT | INDEXES | PROCEDURE | TABLES [ALL]]

3.1.5 删除数据库

当一个数据库已经不再使用时，用户应及时将其删除。

DELETE DATABASE DatabaseName | ?

[DELETETABLES] [RECYCLE]

该命令用于从磁盘上删除一个数据库的操作。

参数 DatabaseName 指定要删除的数据库名称，这个名称可以是包含该数据库文件存储路径的全文件名。要删除的数据库应处于非打开方式，否则 VFP 将报错。

参数 ？将弹出对话框提示用户输入要删除的数据库名称。.

参数 DELETETABLES 从磁盘上删除该数据库中包含的表文件。

参数 RECYCLE 指定要删除的文件并不是立即从磁盘上删除，而是将要删除的文件放入 Windows 95 的回收站（ Recycle Bin ），用户可以应用回收站命令恢复删除的文件。由

此可以防止删除了本不想删除的文件。

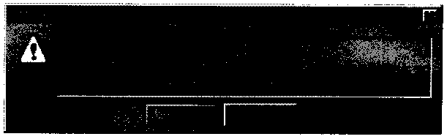


图 3.12 删除处于打开状态的数据库

在进行删除操作时，若 SET SAFETY 的值是 ON， VFP 将询问用户是否真的要删除该数据库；值若是 OFF 则 VFP 将不做任何提示而直接删除该数据库。

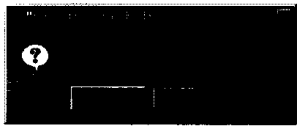


图 3.13 询问是否确认删除

例子：

应用下面的例子，我们将创建一个名位 TempDbc 的数据库，在该数据库中创建一个名为 TempTab 的表，显示该数据库和表的信息后将它们删除。

```
CLOSE ALL                                    && 关闭所有数据库
CREATE DATABASE TempDbc                      && 创建数据库
CREATE TABLE TempTab(Name C(20), Age         && 创建数据表
N(4))
CLEAR                                        && 清屏
DISPLAY TABLES                               && 显示表信息
DISPLAY DATABASES                            && 显示数据库信息
CLOSE ALL                                    && 关闭数据库
DELETE DATABASE TempDbc DELETETABLES         && 删除数据库
```

3.2 数据表的操作命令

在 VFP 中，数据表的一般操作包括创建数据表、将一个自由表加入到数据库中、将一个数据库中的表移出该数据库；对于已经存在的数据表，一般操作包括数据表的打开、数据表的关闭、删除数据表、显示数据表信息和对数据表定义的修改等。

3.2.1 创建数据表

CREATE TABLE TableName1 [NAME LongTableName] [FREE]
(FieldName1 FieldType [(nFieldWidth [, nPrecision])]……

该命令用于创建一个带字段的数据表。

参数 TableName1 指定要创建的数据表名称。

参数 NAME LongTableName 可以给该表指定一个长名称（long name）。长名称只能应用于数据库中的表。长名称可以包含 128 个字符。

参数 FREE 指明该表为自由表，不加入任何数据库中。若创建表时，没有数据库处于打开状态，该参数是不必要的。以为此时该表自动为自由表，而不加入任何数据库中。

参数(FieldName1 FieldType [(nFieldWidth [, nPrecision])]分别指定字段名、字段类型、字段宽度和精度。单个表中可以包含 255 个字段，当一个或者更多字段允许空值（Null）时，则允许的字段数减少到 254 个。

下表给出了字段名、字段类型、字段宽度和精度的描述：

表 3.1 字段描述

字段类型	字段宽度	精度	描述
C	n	—	宽度为 n 的字符型
D	—	—	日期型
T	—	—	日期时间型
N	n	d	宽度为 n，小数位为 d 的数值型
F	n	d	宽度为 n，小数位为 d 的浮动型
I	—	—	整型
B	—	d	双精度型
Y	—	—	货币型
L	—	—	逻辑型 1
M	—	—	备注型
G	—	—	通用型

对于 N 或 F 类型来说，当 nPrecision 的值没有显示给出时，缺省值为 0（没有小数位）。对于 B 类型来说，当 nPrecision 的值没有显示给出时，缺省值由 SET DECIMAL 的设置决定。

该命令的参数较多，如：

[NULL | NOT NULL]

[CHECK lExpression1 [ERROR cMessageText1]]

[DEFAULT eExpression1]

[PRIMARY KEY | UNIQUE]

[REFERENCES TableName2 [TAG TagName1]]
[NOCPTRANS]
……

限于篇幅这里就不一一介绍了，读者可以具体查看有关“命令、函数”部分，得到详细的语法描述。

新建的表被放置在最低的可用工作区中，用户还可以通过别名（Alias）访问它。新建的表一律为独占方式打开，而不管当前的 SET EXCLUSIVE 的值如何。

若在创建新表时，某个数据库处于打开状态，而且没有带 FREE 参数时，该表将被自动加入到该数据库中。但在一个数据库中的表不能重名。

例子：

在下面的两个例子中，第一个创建一个自由表，而第二个在数据库中创建一个表。

例 1：

```
CLOSE ALL                                    && 关闭所有数据库
CREAT TABLE MyTable （Name C(10)，Age N(4)）  && 创建自由表
```

例 2：

```
CREAT DATABASE MyDbc                         && 创建数据库
CREAT TABLE MyTable （Name C(10)，Age N(4)）  && 创建数据库中的表
```

3.2.2 将数据表加入到数据库

一个数据表可以是在某个数据库中的表，也可以是一个不属于任何数据库的自由表。用户可以应用 ADD TABLE 命令，将一个自由表加到某个数据库中，使该表成为数据库中的数据表。

ADD TABLE TableName | ?
[NAME LongTableName]

该命令将一个自由表加入到当前数据库中。

参数 TableName 指定要加入到数据库中的数据表名称。

参数 ？将弹出询问框，要求用户选择要加入到数据库中的数据表名称。

参数 NAME LongTableName 为该数据表指定一个长名称。该名称可以最多包含 128 个字符，可以用来替代来后缀.DBF 的文件名。

一旦某个数据表被命令 ADD TABLE 加入到某个数据库中，该表就不再是自由表。当然，用户可以通过命令 REMOVE TABLE 将该表从数据库中移出，使之从新成为自由表。有关 REMOVE TABLE 命令，下面还将介绍到。

对于要加入的数据表，VFP 对其是有一些规定的：

（1）必须是一个有效的 .DBF 文件；

（2）不能与数据库中已经存在的数据表重名，除非为该表分配一个唯一的长名称；

（3）一旦一个数据表属于一个数据库，该表就不能再成为其他数据库中的表。

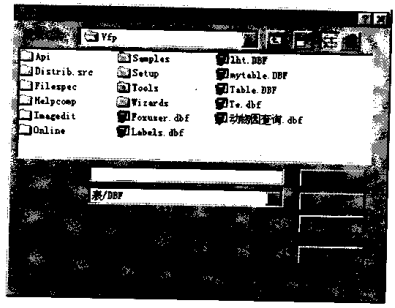


图 3.14　添加表对话框

例子：

在下面的例子中，创建两个数据库 MyDbc1 和 MyDbc2，创建一个数据表 MyTab，该表在创建时被自动加到 MyDbc2 中。我们将其从 MyDbc2 中移出加入到 MyDbc1 中。

```
CREATE DATABASE MyDbc1                          && 创建数据库
CREATE DATABASE MyDbc2                          && 创建数据库
CREATE TABLE MyTab(Name C(10), Age C(4))        && 创建数据表，并假如到 MyDbc
CLOSE TABLES                                    && 关闭数据表
REMOVE TABLE MyTab                              && 从数据库中移出数据表
SET DATABASE TO MyDbc1                          && 设定 MyDbc1 为当前数据库
ADD TABLE MyTab                                 && 将数据表加入到 MyDbc1 中
```

3.2.3　将数据表移出数据库

如同上面我们将到的一样，一个自由表可以加到数据库中，反之，我们可以应用 REMOVE TABLE 命令将一个数据表从其所属的数据库中删除。

REMOVE TABLE TableName | ?

[DELETE] [RECYCLE]

该命令从当前数据库中移出数据表。

参数 TableName 指定要移出的数据表的名称。

参数 ? 显示移出对话框，用户可以在当前数据库中选择要移出的表。

参数 DELETE 指定该表被移出的同时，从磁盘删除。

注意：通过参数 DELETE 而从磁盘上删除的数据表，不能被恢复。

参数 RECYCLE 指定删除时只是将其放入 Windows 95 的回收站，而不是立即进行物理删除。

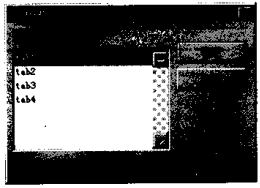


图 3.15 从数据库中选择要移出的表

一个数据表从某个数据库中移出后就成为一个自由表，该自由表还可以应用 ADD TABLE 命令加入到另外的数据库中。

当一个数据表被应用 REMOVE TABLE 命令从某个数据库中移出的同时，所有与之有关的主索引（primary indexes），缺省值（default values）和有效性规则（validation rules）都将被删除。当 SET SAFETY 的值为 ON 时，VFP 会显示确认框。

图 3.16 是否从数据库中移出表

注意：当一个数据表从某个数据库中被移出时，该表与数据库中其他表之间的关系将同时被删除。

例子：

在下面的例子中，我们创建一个名为 Base 的数据库，在其中创建三个数据表 Tab1、Tab2 和 Tab3。显示数据库信息后，移出 Tab1，然后再次显示数据库信息。

```
CREAT DATABASE Base                          && 创建数据库
CREAT TABLE Tab1(Name C(10), Age N(4))       && 创建表，并加入到数据库中
CREAT TABLE Tab1(Name C(10), Age N(4))       && 创建表，并加入到数据库中
CREAT TABLE Tab1(Name C(10), Age N(4))       && 创建表，并加入到数据库中
CLEAR                                        && 清屏
DISPLAY DATABASE                             && 显示数据库信息
CLOSE TABLES                                 && 关闭所有表
REMOVE TABLE Tab1                            && 移出表 Tab1
CLEAR                                        && 清屏
DISPLAY DATABASE                             && 显示数据库信息
```

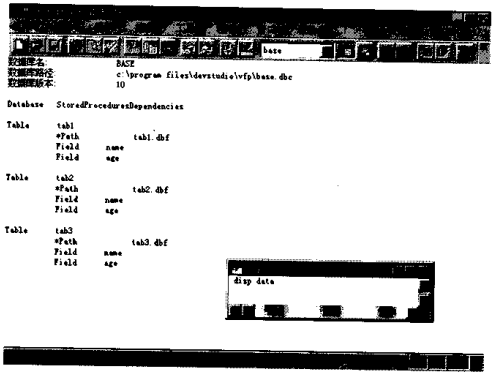


图 3.17　移出表前的数据库信息

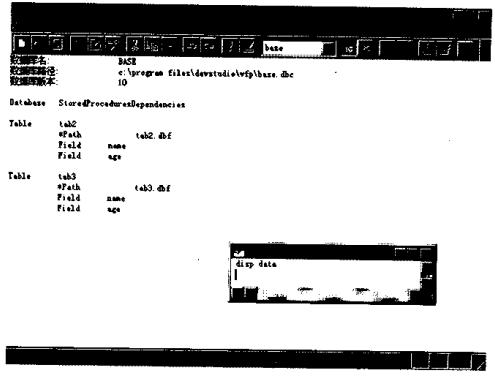


图 3.18　移出表后的数据库信息

3.2.4　打开数据表

一个数据表的打开并不是应用 OPEN 命令，而是要用到命令 USE 。 VFP 同时可以打开许多数据表，所以在打开数据表时还要涉及到工作区的选择。

```
USE [[DatabaseName.]Table | ?]
    [IN nWorkArea | cTableAlias]
    ……
```

该命令用于打开数据表，以供操作其中数据之用。

参数[DatabaseName.]TableName 指定了要打开的数据表名称。若要打开的数据表在当前数据库中，则不必带数据库名称（DatabaseName）；否则用户应指明数据表所处的数据库名称。两者之间用下圆点（.）隔开。

参数 ? 将显示一个使用对话框，用户可以从中选择要打开的数据表。

参数 IN nWorkArea 将该数据表在指定的工作区中打开。

在 IN 从句中，用户可以将工作区号指定为 0 。此时 VFP 会自动将该数据表在当前可用的最低的工作区中打开。

参数 IN cTableAlias 指定将该数据表打开在以 cTableAlias 为别名打开的数据表的工作区中。

若用户没有指定 nWorkArea 或 cTableAlias，该数据表将在当前工作区中打开。

在一个工作区中，只能打开一个数据表。当一个新的数据表在某个工作区中被打开的同时，该工作区中原有的数据表将被关闭。若使用 USE 命令而没有带数据表名称，则在当前工作区中打开的数据表将被关闭（下面数据表的关闭中还要介绍到）。

USE 命令是 VFP 中的一个非常常用的一个命令，其参数也十分丰富，涉及到数据表打开的属性、是否允许修改、索引的应用、排序等等许多性质。限于篇幅，我们无法一一做详细的介绍。在此我们将其语法完整的列在下面，需要时用户可以参考“命令、函数”部分。

```
USE [[DatabaseName.]Table | SQLViewName | ?]
    [IN nWorkArea | cTableAlias]
    [ONLINE]
    [ADMIN]
    [AGAIN]
    [NOREQUERY [nDataSessionNumber]]
    [NODATA]
    [INDEX IndexFileList | ?
    [ORDER [nIndexNumber | IDXFileName
    | [TAG] TagName [OF CDXFileName]
    [ASCENDING | DESCENDING]]]]
    [ALIAS cTableAlias]
    [EXCLUSIVE]
[SHARED]
    [NOUPDATE]
```

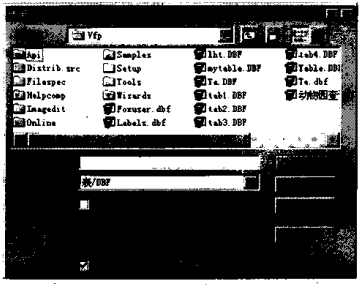


图 3.19　使用对话框

例子：

在下面的例子中，我们打开数据库 Base 中的三个数据表 Tab1 、 Tab2 和 Tab3 。因为我们在 USE 命令中使用了 IN 0 从句，这三个数据表被打开在不同的工作区中。

我们通过“数据工作期”窗口来观察数据表的打开和工作区的变化情况。在此应注意到各个数据表打开后工作区的变化，以体会 IN 0 从句的使用。注意在“数据工作期”窗口下部工作区的变化。

```
CLOSE DATABASES                 && 关闭数据库
OPEN DATABASE Base              && 打开数据库 Base
ACTIVATE WINDOW View            && 激活“数据工作期”窗口
USE Tab1 IN 0                   && 在工作区 1 中打开数据表
USE Tab2 IN 0                   && 在工作区 2 中打开数据表
USE Tab3 IN 0                   && 在工作区 3 中打开数据表
```

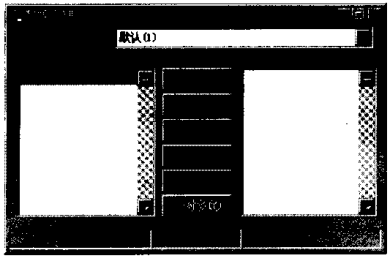


图 3.20　在工作区 1 中打开数据表

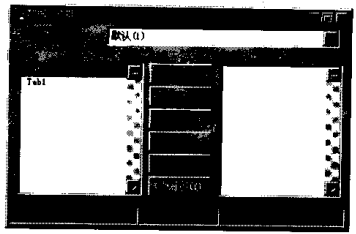


图 3.21 在工作区 2 中打开数据表

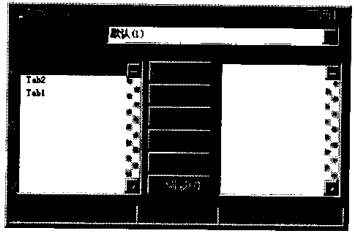


图 3.22 在工作区 3 中打开数据表

3.2.5 关闭数据表

USE

在上面的打开数据表命令中，我们已经介绍过，若使用 USE 命令而没有带数据表名称，则在当前工作区中打开的数据表将被关闭。因此，若用户想要关闭数据表，就需要首先切换的欲关闭数据表所在的工作区，然后使用 USE 命令来关闭数据表。

在 VFP 中切换工作区的方法有很多。我们可以使用命令 SELECT，也可以应用“数据工作期”窗口。当然最简单的方法还是在 USE 命令中直接带上 IN 从句。这样用户就可以简单的关闭一个数据表。

例子：

在下面的例子中，我们应用 SELECT 命令和直接在 USE 命令中带上 IN 从句的方法，关闭数据表。在创建数据表的过程中，VFP 自动将三个数据表分别打开在工作区 1、2 和 3 中。

```
CREAT TABLE Tab1(Name C(10),Age N(4))      && 创建数据表
CREAT TABLE Tab2(Name C(10),Age N(4))      && 创建数据表
CREAT TABLE Tab3(Name C(10),Age N(4))      && 创建数据表
```

```
ACTIVATE WINDOW View          && 激活“数据工作期”窗口
SELECT 1                      && 选择工作区 1
USE                           && 关闭工作区 1 中的 Tab1 表
USE IN 3                      && 关闭工作区 3 中的 Tab3 表
```

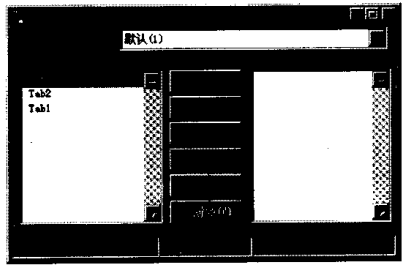


图 3.23 关闭数据表前

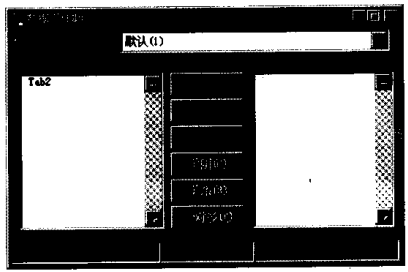


图 3.24 关闭数据表后

3.2.6 删除数据表

DELETE FILE [FileName | ?] [RECYCLE]

该命令可以从磁盘上删除一个文件。

参数 FileName 指定要删除的文件。我们也可以在文件名中使用通配符（*、?）。

参数 ? 显示一个对话框要求用户选择要删除的文件。

参数 RECYCLE 指定将要删除的文件放入回收站，而不是立即删除。

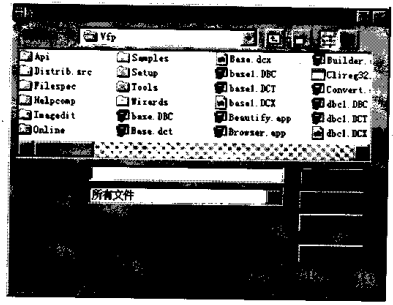


图 3.25 删除对话框

要删除的文件名应包括该文件的存储路径和扩展名，否则 VFP 将无法找到要删除的文件。删除时要删除的文件应处于关闭状态，否则 VFP 将报“该文件正被使用”。

当用户要删除一个数据库中的数据表时，应先将该表应用 REMOVE TABLE，从数据库中移出。若要删除的数据表包含备注字段时，因为备注字段的内容将被分别存在于后缀为 .FPT 的文件中，所以用户应同时删除 .FPT 文件。

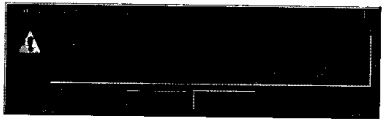


图 3.26 删除处于打开状态的文件

例子：

在下面的例子中，我们删除一个数据表和其备注文件。

```
CLOSE ALL                          && 关闭所有打开的文件
DELETE FILE MyTab.dbf              && 删除数据表文件
DELETE FILE MyTab.fpt              && 删除相关的备注文件
```

3.2.7 显示数据表的字段定义

```
LIST STRUCTURE
    ……
    [IN nWorkArea | cTableAlias]
```

该命令用于显示当前工作区或指定工作区中的数据表的字段定义。

参数 IN nWorkArea | cTableAlias 用于指定要显示的工作区号或者数据表别名。当该项

缺省时，VFP 会显示当前工作区中的数据表字段定义。

为达到相同的目的，用户还可以应用 DISPLAY STRUCTURE 命令。有关 LIST STRUCTURE 的参数较多，在这里只列出其语法。LIST 和 DISPLAY 命令也有一些差别，需要全面掌握其用法的用户可以参考“命令、函数”有关的部分。，

例子：

在下面的例子中，我们创建一个数据表，分别应用 LIST STRUCTURE 和 DISPLAY STRUCTURE 命令来显示该表中字段的定义。我们看到，LIST STRUCTURE 和 DISPLAY STRUCTURE 在此时的应用完全一样。

```
CLOSE ALL                                  && 关闭所有数据库
CREAT TABLE MyTab(Name C(10),Age N(4))     && 创建数据表
CLEAR                                      && 清屏
LIST STRUCTURE                             && 显示数据表字段定义
DISPLAY STRUCTURE                          && 显示数据表字段定义
```

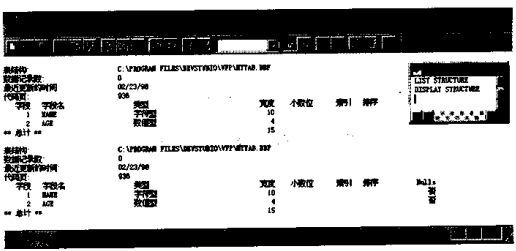


图 3.27 显示数据表字段定义

3.2.8 修改数据表的定义

```
ALTER TABLE TableName1
    ADD | ALTER [COLUMN] FieldName1
        FieldType [(nFieldWidth [, nPrecision])]
    ……
–或者–
ALTER TABLE TableName1
    ALTER [COLUMN] FieldName2
    ……
–或者–
ALTER TABLE TableName1
```

[DROP [COLUMN] FieldName3]
……

该命令可以在编程中用做修改数据表定义。

参数 bleName1 指定将要修改的数据表名称。

该命令有几种格式：

ADD [COLUMN] FieldName1 指定在数据表中增加一个字段，字段名为 FieldName1 。

ALTER [COLUMN] FieldName1 指定将现有的字段 FieldName1 按新的定义进行修改。

以上两种格式中，都要有字段类型 FieldType [(nFieldWidth [, nPrecision])]，关于具体的字段类型定义，读者可以参考 CREAT TABLE 命令中的讲述。

ALTER [COLUMN] FieldName2 指定要改变的字段名。我们可以对该字段是否允许空值、缺省值、有效性检查等从新定义。

DROP [COLUMN] FieldName3 删除指定的字段 FieldName3 。同时删除和该字段相关的缺省值、有效性检查等。

以上命令中还包含许多参数，限于篇幅无法一一详细介绍。读者可以参考“命令、函数”部分，以得到详细的使用说明。

另外，用户也可以使用命令 MODIFY STRUCTURE 来进行数据表结构的修改。命令 MODIFY STRUCTURE 命令可以激活“表设计器”，在表设计器中，用户可以进行各种修改。当然在编程中的修改，还应该有 ALTER TABLE 命令来实现表结构的修改。

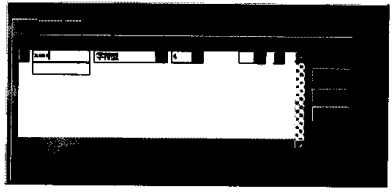


图 3.28 由 MODIFY STRUCTURE 激活的表设计器

例子：

在下面的例子中，我们创建一个数据表。然后应用 ALTER TABLE 命令对其中的字段进行修改。通过 LIST STRUCTURE 来显示修改后的情况。

```
CREAT TABLE MyTab(Name C(10),Age C(4))    && 创建数据表
CLEAR                                     && 清屏
LIST STRUCTURE                            && 显示表结构
ALTER TABLE MyTab ADD Adr C(20)           && 增加字段 Adr
CLEAR                                     && 清屏
LIST STRUCTURE                            && 显示表结构
ALTER TABLE MyTab ALTER Age N(4)          && 将字段 Age 该为数值型
```

```
CLEAR                                    && 清屏
LIST STRUCTURE                           && 显示表结构
```

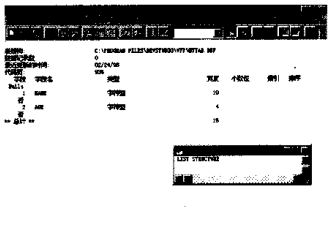


图 3.29　改变前的表结构

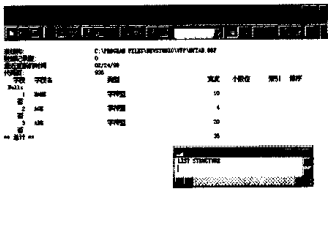


图 3.30　增加字段后的表结构

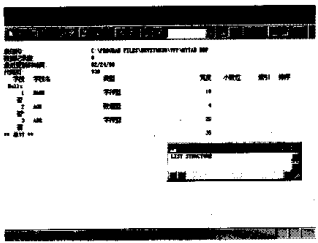


图 3.31　修改字段类型后的表结构

3.3 记录数据的操作命令

在建立了数据表的结构后，我们进一步要对数据表中的记录进行操作。记录的操作一般包括记录的显示、记录的定位、记录的查找、记录的增加、修改、编辑；另外比较常用的操作还有记录的索引、记录的筛选、查询和统计。我们将在下面的一节中，对以上的操作逐一介绍。

3.3.1 记录的显示

```
DISPLAY
    [[FIELDS] FieldList]
    [Scope] [FOR lExpression1] [WHILE lExpression2]
……
—或者—
LIST
    [FIELDS FieldList]
    [Scope] [FOR lExpression1] [WHILE lExpression2]
……
```

以上两个命令都可以用来显示当前数据表的记录。

参数 FIELDS FieldList 指定要显示的字段。缺省时将显示数据表中所有的字段。但备注字段只有在显式包含在 FieldList 中时，才会被显示。

参数 Scope 指定显示记录的范围。这个范围可以是：ALL（全部记录）、NEXT nRecords（下面 n 条记录）、RECORD nRecordNumber（第 n 条记录）和 REST（其余记录）。

参数 FOR lExpression1 指定满足表达式 lExpression1 的记录才会被显示。

参数 WHILE lExpression2 指定只有使得表达式 lExpression2 值为真（.T.）的记录才被显示。

以上这些参数都是用来筛选记录，以显示满足用户要求的记录。另外，这两个命令都还有一些参数，在此就不一一介绍。读者可以参考“命令、函数”部分的有关内容。

以上 DISPLAY 和 LIST 命令在使用上有很多相同之处，但又有一些差别：

（1）显示范围在缺省时 LIST 命令为全部记录（ALL）而 DISPLAY 命令为下一条记录（Next 1）；

（2）当 VFP 的窗口或者用户自定义的用来显示的窗口写满时，LIST 命令没有停顿而一直显示到完成，但 DISPLAY 命令会停下来，等待用户击键；

（3）当 SET DELETED 的值为 ON 时，LIST 命令不显示标记为删除的记录。

图 3.32 停顿提示

例子：

在下面的例子中，我们分别用 LIST 和 DISPLAY 命令来显示表中的记录。其中 LIST 命令显示全部记录，DISPLAY 命令显示一条记录和其余记录。

```
CLEAR                          && 清屏
LIST                           && 显示全部记录
DISPLAY RECORD 4               && 显示第四条记录
DISPLAY REST                   && 显示其余记录
```

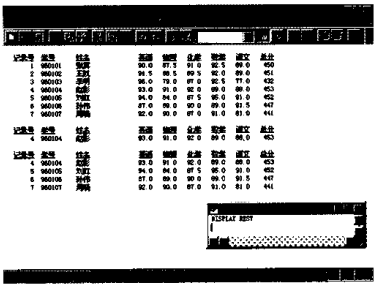


图 3.33　显示记录

3.3.2　记录的定位

在显示命令中，我们发现在数据表中经常需要定位记录。如在 DISPLAY 命令中缺省时只显示下一条记录，定位到要显示的记录就变得十分必要了。

VFP 提供了许多定位记录的方法，下面我们分别介绍：

```
GO [RECORD] nRecordNumber [IN nWorkArea | IN cTableAlias]
-或者-
GO TOP | BOTTOM [IN nWorkArea | IN cTableAlias]
-或者-
GOTO [RECORD] nRecordNumber [IN nWorkArea | IN cTableAlias]
-或者-
GOTO TOP | BOTTOM [IN nWorkArea | IN cTableAlias]
```

参数 RECORD nRecordNumber 指定将记录指针移到指定的位置。该位置是指记录的物理记录号。

参数 IN nWorkArea 指定移动某个工作区中的记录指针，由此可以改变不同的表中的记录指针。

参数 IN cTableAlias 指定一个表的别名，移动该表中的记录指针到指定的位置。

参数 TOP 将记录指针移到表的第一条记录。值得注意的是若该表有一个升序的索引，则第一条记录是指索引值最低的记录。反之若索引为降序的话，第一条记录为索引值最高的记录。

参数 BOTTOM 将记录指针移到表的最后一条记录。关于索引数据记录与 TOP 中正好相反。

例子：

在显示的例子中，我们用到 DISPLAY 的 Scope 范围选项，下面我们应用 GO 命令直接定位记录指针。

```
CLEAR                    && 清屏
DISPLAY ALL              && 显示所有记录，此时记录指针在表尾
GOTO 4                   && 将记录指针移到第四个记录
DISPLAY                  && 显示第四条记录
GOTO TOP                 && 将记录指针移到表头
DISPLAY                  && 显示第一条记录
```

SKIP

[nRecords]

[IN nWorkArea | cTableAlias]

在上面的 GO|GOTO 命令中，记录指针直接移到用户指定的位置。而应用 SKIP 命令可以从当前位置起，方便的将记录指针移到相对于当前位置的前或者后面几个记录。

参数 nRecords 指定从当前位置起要移动的记录数。当缺省时，将记录指针移到下一个记录。该值可以为正或者为负，若值为正时，记录指针往后移动（向表尾），若值为负，记录指针向前移动（向表头）。

若数据表已经建立了索引，则以上的移动均指按索引值排序。

参数 IN nWorkArea | cTableAlias 指定一个表所在的工作区，或其别名。

例子：

在下面的例子中，我们已经打开了“成绩表”。我们使用 SKIP 命令来定位记录指针，通过 DISPLAY 命令显示当前指针处的记录。

```
CLEAR                    && 清屏
DISPLAY ALL              && 显示全部记录，此时记录指针在表尾
GOTO 1                   && 将记录指针移到表头
DISPLAY                  && 显示第一条记录
SKIP 5                   && 将记录指针后移 5 个记录，到第六个记录
DISPLAY                  && 显示该条记录
SKIP -3                  && 将记录指针前移 3 个记录，到第三条记录
DISPLAY                  && 显示该条记录
```

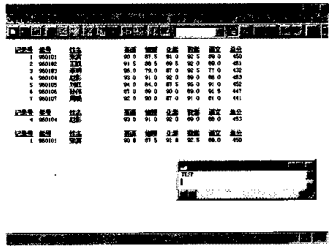


图 3.34 应用 GO 命令定位

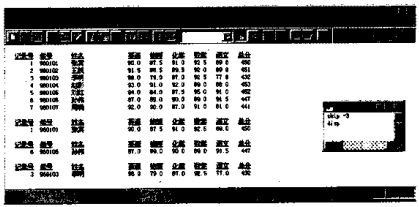


图 3.35 应用 SKIP 命令移动记录指针

3.3.3 记录的查找

在上面的内容里我们讲述了移动记录指针的命令 GO\GOTO 和 SKIP。这些命令在使用时，要求用户知道所要移动记录指针到达的记录号。如我们应用 GOTO 1 命令将记录指针移到第一条记录，应用 SKIP -1 命令将记录指针移到相对与当前记录指针处的前一条记录。

在很多时候，我们并不知道我们要定位的记录在数据表中的具体位置。比如我们要显示姓名为“张红”的记录，此时通过移动记录指针的 GO\GOTO 和 SKIP 命令，就无法达到目的，此时就需要应用查找记录的命令。反过来，就是在数据表中查找满足用户要求的记录，并将记录指针移到该条记录处。

经常用到的用于查找记录的命令也有一些，我们将分别做一些简单的介绍：

LOCATE FOR lExpression1

[Scope]

[WHILE]

……

该命令用于查找符合要求的记录，并将指针定位到第一条匹配的记录。

参数 FOR lExpression1 指定一个逻辑表达式，在表中查找与之匹配的记录。

参数 Scope 指定查找的范围。其选项与前面范围选项完全相同。（ ALL 、 NEXT nRecords 、 RECORD nRecordNumber 和 REST ）。缺省的范围是 ALL 。

参数 WHILE lExpression2 与前面讲到的 WHILE 从句一样，也是指定一个记录应满足的逻辑表达式。也就是说，记录应使该表达式值为真。

应用 LOCATE 查找一个记录，该数据表不必经过索引。

应用 LOCATE 命令找到一个符合要求的记录后，记录指针就定位在该记录处。如果用户想知道该条记录的记录号，可以通过函数 RECNO( )返回当前的记录号。一个数据表中，符合查找条件的记录往往不止一条，而应用 LOCATE 命令在找到第一条记录后，就将记录指针停在该条记录处，当用户需要查找下一条匹配的记录时，可以通过 CONTINUE 命令继续查找。

当找到匹配记录时，函数 FOUND( )返回真（.T.）、 RECNO( )返回当前（匹配记录）的记录号；若没有找到匹配的记录， FOUND( )返回假（.F.）、 RECNO( )返回为数据表全部记录数再加 1 的值，并且 EOF( )返回真（.T.）。

例子：

在下面的例子中，我们继续使用上面已经打开的“成绩表”，通过 LOCATE 命令查找英语成绩成绩为 96 分的记录。在找到满足条件的记录后，我们可以通过 ？：：命令查看函数 FOUND( )、 RECNO( )和 EOF( )在执行了 LOCATE 命令后的值。在下面的图中，我们可以看到函数 FOUND( )的值为真（.T.），表示找到了满足要求的记录；函数 RECNO( )的值为 3 ，表示匹配的记录为数据表中的第三条记录；函数 EOF( )的值为假（.F.），表示记录指针未到文件结尾，通过此值也可以反映出找到了一条满足条件的记录。

```
CLEAR                          && 清屏
DISPLAY ALL                    && 显示所有记录
LOCATE FOR 英语=96             && 查找英语成绩为 96 的记录
DISPLAY                        && 显示该条记录
?FOUND( )                      && 显示函数 FOUND( )的值
?RECNO( )                      && 显示函数 RECNO( )的值
?EOF( )                        && 显示函数 EOF( )的值
```

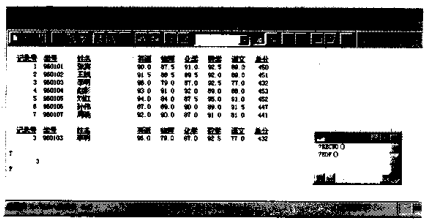


图 3.36 应用 LOCATE 命令查找记录

SEEK eExpression
　[ORDER nIndexNumber | IDXIndexFileName
　| [TAG] TagName [OF CDXFileName]
　[ASCENDING | DESCENDING]]
　[IN nWorkArea | cTableAlias]

SEEK 命令与 LOCATE 命令都是查找一个匹配的记录，并将记录指针移到相应的记录处。但在具体使用中，SEEK 命令与 LOCATE 命令有又不同。SEEK 命令要求要查找的数据表经过索引排序。

如下面的例子中，因为数据表没有索引排序，VFP 将报错：

```
CLOSE ALL
USE 成绩表
SEEK “960001”
```

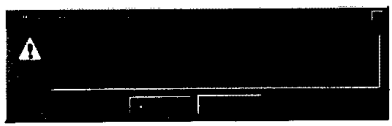


图 3.37　未排序的表

与 LOCATE 命令一样，当找到匹配记录时，函数 FOUND( )返回真（.T.）、RECNO( )返回当前(匹配记录)的记录号；若没有找到匹配的记录，FOUND( )返回假(.F.)、RECNO( )返回为数据表全部记录数再加 1 的值，并且 EOF( )返回真（.T.）。

有关 SEEK 命令的具体使用，限于篇幅这里就不一一讲解了。读者可以参考“命令、函数”部分。在此，我们给出一个例子。

例子：

在本例中，我们打开的“成绩表”，并按“学号”字段排序，通过 SEEK 命令查找“学号”为“960104”的记录，显示该条记录后，再次应用 SEEK 命令查找“总分”为 450 分的记录，并显示。

```
CLOSE ALL                    && 关闭所有文件
USE 成绩表 ORDER 学号        && 打开数据表，按学号排序
CLEAR                        && 清屏
DISPLAY ALL                  && 显示所有记录
SEEK “960104”                && 查找学号为 960104 的记录
DISPLAY                      && 显示该条记录
SEEK 450 ORDER 总分          && 查找总分为 450 的记录
DISPLAY                      && 显示该条记录
```

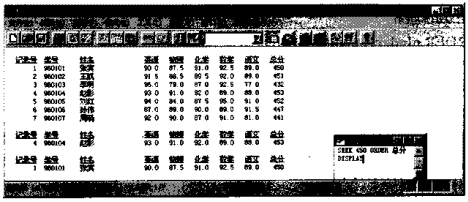


图 3.38 应用 SEEK 命令

另外，还有一些用于查找匹配记录的命令，如 FIND 命令等。在此不再介绍，读者可以参考“命令、函数”部分的有关内容。

在上面的讲述中，我们经常用到记录指针的概念。记录指针表明数据表中当前记录的位置。我们的很多操作都是针对当前即记录指针指向的记录的。

当打开一个数据表时，记录指针指向表头第一条记录。应用以上介绍的命令，都可以移动记录指针。例如当我们知道要找的记录的记录号时，可以应用 GOIGOTO 将记录指针直接移动目的地；若是想将记录指针移动到相对于当前位置的前几个记录或者是后几个记录，就可以应用 SKIP 命令来实现；如果用户知道要查找记录的某些特征，更可以使用 LOCATE 、 SEEK 等命令，将记录指针移动到想要查找的记录。

我们在此只是对其进行一些介绍，所有这些操作，只有用户在使用中，根据实际情况灵活使用才能达到很好的效果。

3.3.4 记录的增加

APPEND [BLANK]

[IN nWorkArea | cTableAlias]

……

该命令用于在数据表中增加一条或者几条记录。一个数据表在创建时还只有一个“空架子”，只有在其中增加了一些记录数据后，该数据表才有意义。

参数 BLANK 在当前数据表的表尾处增加一条空记录。

参数 IN nWorkArea 指定要增加记录的数据表所处的工作区。缺省时为当前工作区。

参数 IN cTableAlias 指定要增加记录的数据表的别名。

若当我们使用 APPEND 命令时，当前工作区中没有打开的数据表， VFP 会询问要打开的数据表。

另外，对参数 BLANK 的使用与否， VFP 也将有不同的处理。当使用 APPEND 而没有带 BLANK 参数时， VFP 将打开一个编辑窗口，用户可以在其中直接增加记录数据。但若使用 APPEND BLANK 命令时， VFP 只是在数据表表尾增加一条空记录，而并不打开编辑窗口供用户输入数据。用户可以在以后的操作中应用 BROWS 、 EDIT 或 CHANGE 命令来

加入数据，也可以应用 REPLACE 修改记录的命令加入数据。

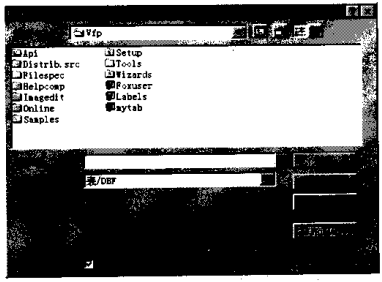


图 3.39　APPEND 命令激活的打开窗口

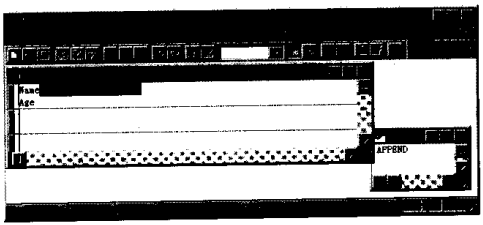


图 3.40　不带 BLANK 参数的 APPEND 命令

例子：

在下面的例子中，我们创建一个数据表，此时数据表中还没有任何记录。然后用 APPEND 命令增加两条空记录。为了向数据表中加入数据，我们用到了 REPLACE 命令（REPLACE 命令下面将要介绍到），应用该命令向第一条空记录加入数据。

```
CLOSE ALL                                  && 关闭所有文件
CREAT TABLE MyTab(Name C(10),Age N(4))     && 创建数据表
APPEND BLANK                               && 增加一条空记录
APPEND BLANK                               && 增加一条空记录
GOTO 1                                     && 回到第一条记录
REPLACE Name WITH "张红", Age WITH 25      && 加入数据
CLEAR                                      && 清屏
LIST                                       && 显示结果
```

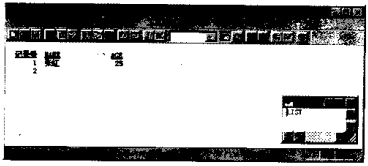


图 3.41 APPEND 的例子

3.3.5 记录的修改

```
REPLACE FieldName1 WITH eExpression1 [ADDITIVE]
    [, FieldName2 WITH eExpression2 [ADDITIVE]] ...
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    [IN nWorkArea | cTableAlias]
    ……
```

在 APPEND 命令中，我们已经提到了 REPLACE 命令。当用 APPEND BLANK 命令为数据表增加一条空记录后，我们就可以应用 REPLACE 命令将数据加入到空记录中。REPLACE 命令的功能就是修改数据记录。

从句 FieldName1 WITH eExpression1 [, FieldName2 WITH eExpression2 ...]指定应用 eExpression1 的值代替 FieldName1 字段中现有的值，用 eExpression2 的值代替 FieldName2 字段中现有的值……，这样我们就可以将该记录中的原有值用新的值代替。

注意：当表达式的值比字段的宽度大时， VFP 将做一些强制的转化。

参数 ADDITIVE 只应用于修改备注字段。带上 ADDITIVE 参数的 REPLACE 命令将新值加到原有内容的末尾，而并不改变原有的内容。若没有 ADDITIVE 参数，新值将覆盖原有的备注字段内容。

参数 Scope 、 FOR lExpression1 和 WHILE lExpression2 与前面介绍到的命令完全一样，也是用来指定范围。 Scope 的缺省值为下一条（ NEXT 1 ）。

参数 IN nWorkArealIN cTableAlias 也与前面的介绍完全一样，用于指定要修改记录的数据表。缺省时为当前选定的工作区。

例子：

在 APPEND 命令的例子中，我们已经用 REPLACE 命令将数据表中的第一条记录加入了数据，下面我们将第二条记录也加入数据。

```
CLOSE ALL                                  && 关闭所有文件
USE MyTab                                  && 打开数据表
GOTO 2                                     && 将记录指针移到第二条记录
REPLACE Name WITH "王铭",Age WITH 30       && 修改该条记录
CLEAR                                      && 清屏
LIST                                       && 显示
```

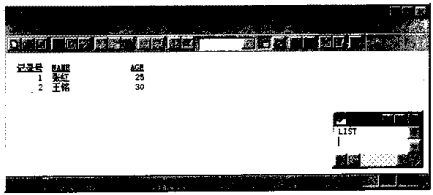


图 3.42　REPLACE 命令

3.3.6　记录的编辑

EDIT

[FIELDS FieldList]

[Scope] [FOR lExpression1] [WHILE lExpression2]

……

该命令用来激活编辑窗口，用户可以在该窗口中编辑数据表中的记录。

参数 FIELDS FieldList 用来指定显示在编辑窗口中的字段名。也就是要编辑的字段名。用户可以改变字段名的顺序，编辑窗口中将按用户给出的顺序显示。缺省时为全部字段。

例如下面的不同命令，将有不同的显示结果：

（1）EDIT。

（2）EDIT FIELDS Name。

（3）EDIT FIELDS Age,Name。

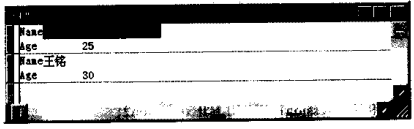


图 3.43　EDIT 命令

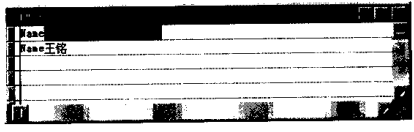


图 3.44　EDIT FIELDS Name 命令

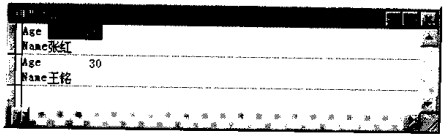


图 3.45 EDIT FIELDS Age,Name 命令

字段名还有许多修饰，如 :R 使得该字段处于只读状态，用户不能对其进行修改。限于篇幅，这里只给出语法结构，具体应用读者可以参考“命令、函数”部分：

```
FieldName1
    [:R]
    [:nColumnWidth]
    [:V = lExpression1 [:F] [:E = cMessageText]]
    [:P = cFormatCodes]
    [:B = eLowerBound, eUpperBound [:F]]
    [:H = cHeadingText]
    [:W = lExpression2]
    [, FieldName2 [:R]...]
```

当用户按 ESC 键推出编辑窗口时，用户所做的最后一条修改将丢失。用户在编辑窗口中，将光标从一条记录移动到另一条记录时，该条记录中所做的修改就将被保存。

EDIT 命令的参数还有很多，我们无法一一介绍到。读者可以在查阅“命令、函数”部分的同时，自己做一些练习。

在 VFP 中，还有一些命令用于记录数据的编辑，如命令 CHANGE；另外，有些命令虽然本身不是编辑命令，但用户同样可以应用其对记录进行编辑操作，如浏览命令 BROWSE 命令。我们在此不再一一介绍，仅将其简单的语法列出，用户可以自己做一个练习。

```
CHANGE
    [FIELDS FieldList]
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    ......
```

CHANGE 命令的用法与功能与 EDIT 命令完全一样，保留该命令是为了与以前版本的 FoxPro 兼容。

```
BROWSE
    [FIELDS FieldList]
    [FONT cFontName [, nFontSize]]
    [STYLE cFontStyle]
    [FOR lExpression1 [REST]]
    ......
```

BROWSE 命令是常用的用于浏览数据表的命令，相信读者对其再熟悉不过了。应用

BROWSE 命令同样可以编辑数据表中的记录。有关 BROWSE 的具体应用，这里就不再介绍。

总之，我们可以应用 EDIT 、 CHANGE 或者 BROWSE 激活编辑窗口，也可以将该窗口打开在用户自己定义的窗口中。在编辑窗口中，用户可以交互的编辑已有的记录。以上介绍到的三个用于编辑的命令，都有丰富的参数，从显示的字段、字体、范围到有效性检查等等，用户都可以自己方便地设定，以满足用户的不同需要。

很多用户在用到上面的命令时，总是仅仅应用到一些基本的参数，而没有真正用到该命令的丰富设定。建议用户参照用户手册中对该命令的描述，或者 VFP 的联机帮助，将其丰富的功能应用到自己的程序中，你一定可以有不同的感受。

3.3.7 记录的删除

在 VFP 的数据表中，删除一个记录并不是直接将该记录从表中删除，而是先进行标记，也就是将该记录加上删除标记。当确认删除后，用户可以再将该记录彻底删除。在标记为删除后，用户还可以将该删除标记去处，也就是取消该记录的删除，恢复到删除前的状态。但当一个记录被彻底删除后，该记录的数据就被从表中去掉了，而无法恢复。相应与以上的操作， VFP 提供了相应的命令 DELETE （标记删除）、 RECALL （取消删除标记）、 PACK （将标记为删除的记录彻底删除）和 ZAP （删除数据表中所有记录）。

DELETE 命令——标记记录为删除

DELETE

[Scope] [FOR lExpression1] [WHILE lExpression2]

[IN nWorkArea | cTableAlias]

……

参数 Scope 、 FOR lExpression1 和 WHILE lExpression2 与前面将到的用法一样，用来设定 DELETE 命令作用的范围。

Scope 参数的缺省值为当前记录（ NEXT 1 ）。

参数 IN nWorkArea | IN cTableAlias 指定进行删除标记的数据表所在的工作区或者别名。缺省时为当前工作区。

例子：

在下面的例子中，我们应用数据表 Tab ，通过命令 SKIP 将记录指针移到第二条记录，应用命令 DELETE 将数据表中的第二条记录标记为删除。再次显示该数据表中记录时，我们看到第二条记录已经被标记为删除（记录旁标记了星号）。

```
CLOSE ALL          && 关闭所有文件
USE MyTab          && 打开数据表
SKIP               && 将记录指针移到第二条记录
DELETE             && 标记该条记录为删除
CLEAR              && 清屏
```

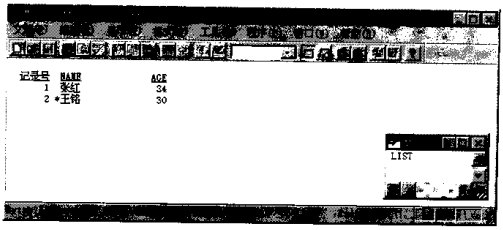


图 3.46 标记删除

另外，我们还可以用 DELETE — SQL 命令来标记删除记录。限于篇幅，我们只列出其语法，读者可以进一步参考“命令、函数”部分：

DELETE FROM [DatabaseName!]TableName

[WHERE FilterCondition1 [AND | OR FilterCondition2 ...]]

除了以上的命令方式外，标记删除的操作还可以方便的在编辑或者浏览窗口中，直接用鼠标在该记录旁标记。

RECALL 命令——去掉删除标记

RECALL

[Scope] [FOR lExpression1] [WHILE lExpression2]

.....

参数 Scope 、 FOR lExpression1 和 WHILE lExpression2 与前面将到的用法一样，用来设定 DELETE 命令作用的范围。

Scope 参数的缺省值为当前记录（ NEXT 1 ）。

当某个记录被标记为删除后，也就是被作用与 DELETE 命令后，尚没有被作用 PACK 或 ZAP 命令，也就是还没有物理删除时，可以用 RECALL 命令去掉删除标记，恢复到标记以前状态。

另外，我们也可以在编辑窗口或者浏览窗口中，用鼠标方便地去掉删除标记。

例子：

在下面的例子中，我们接着上面 DELETE 命令的例子，将标记为删除的记录去掉删除标记，恢复到标记前的状态。

```
CLOSE ALL              && 关闭所有文件
USE MyTab              && 打开数据表
CLEAR                  && 清屏
LIST                   && 显示该数据表中记录，第二条已经被标记为删除
GOTO 2                 && 将记录指针移到第二条记录
RECALL                 && 去掉删除标记
```

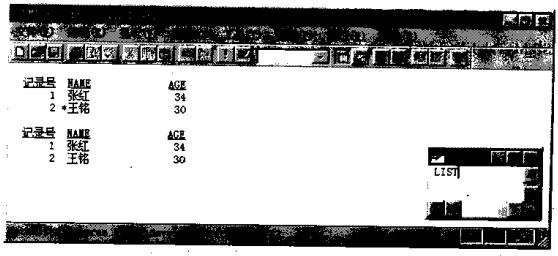


图 3.47 RECALL 命令

PACK 命令——删除标记的记录

在一个数据表中，标记删除的记录只是在该记录处做一个删除标记，而并没有从数据表中真正删除，依然占据磁盘空间。当这种记录过多时，不但会浪费磁盘空间，还会影响到数据表处理的速度。所以当用户真正要删除某些记录时，除了标记外，还应该将其彻底删除。

PACK [MEMO] [DBF]

该命令可以将数据表中标记为删除的记录从磁盘上彻底删除，还可以删除备注文件中相应的部分，减少备注文件的大小。

参数 MEMO 指示该命令并不删除数据表中标记为删除的记录，而是整理该数据表相应的备注文件，删除其中没有用到的部分。

参数 DBF 正好相反，指示该命令只删除数据表中标记为删除的记录，而不影响相应的备注文件。

而不带任何参数的 PACK 命令，既删除数据表中标记为删除的记录，又整理该数据表相应的备注文件，删除其中没有用到的部分。

当使用 PACK 命令删除记录时，数据表应为独占方式打开。 PACK 命令同时重建索引。

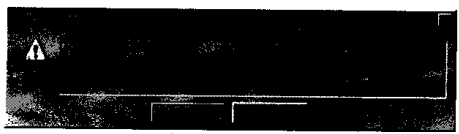


图 3.48 删除时文件应为独占方式打开

例子：

在下面的例子中，我们将第二条记录标记为删除，然后用 PACK 命令将其彻底删除。在不同的阶段， LIST 命令将显示该数据表中相应的情况。

```
CLOSE ALL                    && 关闭所有文件
USE MyTab EXCLUSIVE          && 打开数据表
CLEAR                        && 清屏
LIST                         && 显示数据表信息
GOTO 2                       && 将记录指针移到第二条记录
DELETE                       && 标记该条记录为删除
LIST                         && 显示数据表信息
PACK                         && 彻底删除记录
LIST                         && 显示数据表信息
```

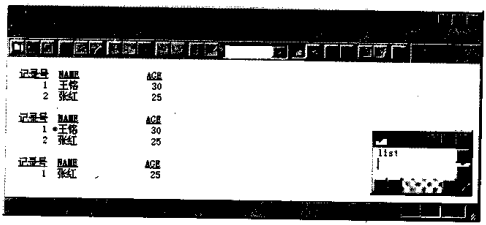


图 3.49 应用 PACK 命令

ZAP 命令——删除表中所有记录

ZAP

[IN nWorkArea | cTableAlias]

该命令可以删除一个数据表中的所有记录，使该数据表成为一个空表。

参数 IN nWorkArea | cTableAlias 指定要删空的数据表所在的工作区或者别名。

当缺省时，为当前工作区中的数据表。

ZAP 命令用于彻底删除一个数据表中所有记录，相当于以下命令，但 ZAP 命令执行得更快。

```
DELETE ALL
PACK
```

由于 ZAP 命令为彻底删除，删除后便无法恢复，所以用户应十分小心使用。另外若 SET SAFETY 的值为 ON 时，使用 ZAP 命令时， VFP 会询问是否确实要删除所有记录。

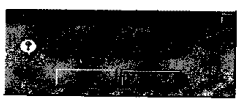


图 3.50 是否确实要删除所有记录

3.3.8 记录的索引

当我们往数据表中加入数据时，往往并没有按照一定的顺序。即使按照一定的顺序，数据表在使用过程中记录的增减也会改变这种顺序。另一方面，在一个数据表中，用户往往需要按照不止一个的字段值排序。这时，我们就需要建立索引文件。索引的建立不仅可以实现按不同字段排序的需要，更可以大大加快查询记录的速度。

索引的建立，并不改变记录在数据表中的原有位置，而仅仅是建立一个记录的排序顺序。

下面我们就介绍一下如何建立和使用索引。

INDEX 命令——建立索引

INDEX ON eExpression TO IDXFileName | TAG TagName [OF CDXFileName]

……

[ASCENDING | DESCENDING]

……

该命令用于建立索引文件，按该索引文件建立数据表中记录的排列顺序。

参数 eExpression 指定一个索引表达式，该表达式可以包含字段名。例如我们可以在“成绩表”中建立按“英语”成绩排序的索引（INDEX ON 英语……），也可以建立按“英语+化学”成绩排序的索引（INDEX ON 英语+化学……）。

参数 TO IDXFileName 指定建立该索引在名为 IDXFileName 的索引文件中。该索引文件的后缀缺省为（.IDX）。

参数 TAG TagName [OF CDXFileName]产生一个复索引文件（Compound Index File）。复索引文件缺省的文件后缀为（.CDX）。

参数 ASCENDING | DESCENDING 用于指定建立索引的排序。缺省时为升序（ASCENDING）。

例子：

在下面的例子中，我们将为数据表 Tab 中的 Name 字段建立一个索引，通过命令 DISPLAY STATUS 我们可以看到该索引建立后，数据表的状态。

```
CLOSE ALL                          && 关闭所有文件
USE MyTab                          && 打开数据表
INDEX ON Name TO MyIdx             && 建立索引
CLEAR                              && 清屏
DISPLAY STATUS                     && 显示数据表状态
```

SET INDEX TO ——打开 | 关闭索引

SET INDEX TO [IndexFileList | ?]

[ORDER nIndexNumber | IDXIndexFileName

| [TAG] TagName [OF CDXFileName] [ASCENDING | DESCENDING]]

[ADDITIVE]

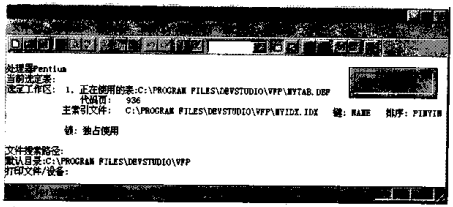


图 3.51 索引文件

该命令用于打开索引文件。

参数 IndexFileList 指定要打开的索引文件名。各个索引文件名之间用逗号隔开。

参数 ? 显示对话框，用户可以从中选择索引文件。

参数 ORDER nIndexNumber 指定主控索引号。这个顺序号是按照上面打开索引文件名列表中的顺序分配的。

当该号为 0 时，按照数据表中记录的物理顺序排列。当该号大于现有的索引文件数时，VFP 将报错。

参数 ORDER IDXIndexFileName 指定一个索引文件为主控索引。

参数 ORDER [TAG] TagName [OF CDXFileName]指定一个复索引文件中的标识为主控索引。

其他参数的用法同前面的 INDEX 命令，详细的用法读者可以参考“命令、函数”部分。

当一个 SET INDEX TO 命令不带任何参数时，关闭所有打开的索引文件。（除了相应于该数据表的结构索引文件）

SET ORDER TO ——使用索引排序

一个数据表可以有许多索引，我们可以通过该命令，设定按照某个索引排序数据表数据。

SET ORDER TO

[nIndexNumber | IDXIndexFileName | [TAG] TagName [OF CDXFileName]

[IN nWorkArea | cTableAlias]

[ASCENDING | DESCENDING]]

一个数据表可以同时打开许多索引文件，但只有一个单索引文件（.IDX ）或者一个复索引文件中的标识（ Tag ）决定着该数据表中记录的排列顺序。该文件或者标识称为主控索引（ Controlling Index ）。应用该命令可以改变数据表的主控索引，为以后的显示、查询等操作做好记录的排序工作。

一个数据表相应的复索引文件在该数据表打开时自动打开，而其他索引文件可以在 USE 命令中加入 INDEX 从句打开。当数据表已经打开后，可以通过 SET INDEX 打开或者关闭索引文件。

参数 nIndexNumber 指定随 USE 命令同时打开的多个索引文件中，哪个为主控索引。

参数 IDXIndexFileName 指定索引文件名。

参数[TAG] TagName [OF CDXFileName]指定标识名。

其他参数和以上参数的具体应用可以参考“命令、函数”部分的相应内容，这里限于篇幅无法一一介绍。

例子：

在下面的例子中，我们通过 SET ORDER TO 命令改变主控索引，在 DISPLAY STATUS 命令中，我们看到主索引的变化。在 LIST 命令中，我们看到数据的排序发生了变化。

```
CLOSE ALL                          && 关闭所有文件
USE 成绩表                         && 将“学号”字段设为主控索引
CLEAR                              && 清屏
LIST                               && 列出记录
DISPLAY STATUS                     && 显示数据表信息
SET ORDER TO 总分 ASCENDING        && 将“总分”字段设为主控索引，按升序
CLEAR                              && 清屏
LIST                               && 列出记录
DISPLAY STATUS                     && 显示数据表信息
```

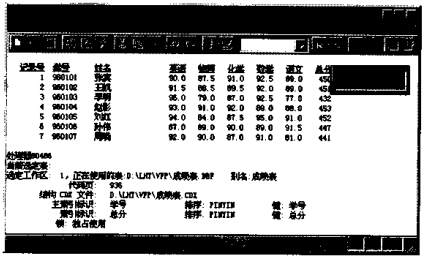


图 3.52　“学号”字段设为主控索引

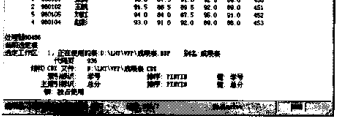


图 3.53　“总分”字段设为主控索引

REINDEX ——重新索引

当用户打开数据表文件时，有些索引文件可能没有同时打开。这样在对该数据表的记录进行修改（增减记录）后，未打开的索引文件因为无法及时更正将产生错误的索引。此时应从新打开该索引文件，从新索引。

REINDEX [COMPACT]

该命令用于重建打开的索引文件。

参数 COMPACT 将通常的单索引文件转换为压缩的索引文件。

例子：

下面的命令用于对过时的索引文件重新索引：

```
CLOSE ALL                                  && 关闭所有文件
USE TableName INDEX OutdatedIndexNames     && 打开数据表和过时的索引文件
REINDEX                                    && 重新索引
```

DELETE TAG ——删除索引

DELETE TAG TagName1 [OF CDXFileName1]
[, TagName2 [OF CDXFileName2]] ...

或

DELETE TAG ALL [OF CDXFileName]

该命令用于删除索引标识。

参数 TagName1 [OF CDXFileName1] [, TagName2 [OF CDXFileName2]] ...指定要删除的标识。这里可以是一个标识的列表，各个标识名用逗号分隔，用于删除多个标识。

参数 ALL [OF CDXFileName]删除某个索引文件中的所有标识。

当一个索引文件中的所有标识都被删除后，该索引文件也被删除。

当 SET SAFETY 的值是 ON 时，若要删除主索引或者候选索引时， VFP 会询问用户是否确实要删除。

3.3.9 记录的筛选

当我们应用 LIST 或者 DISPLAY 时，我们列出数据表中的所有记录。我们还可以通过命令 SET FILTER TO 设定一些筛选条件，这样在显示记录时， VFP 就将仅显示符合条件的记录。

SET FILTER TO [lExpression]

参数 lExpression 指定筛选条件。

当用户使用 SET FILTER TO 而没有带任何参数， VFP 会关闭这个过滤条件。

注意： SELECT - SQL 命令不受该过滤条件的限制。

例子：

我们加入筛选条件，使用 LIST 命令列出的记录将与以前不同：

```
CLOSE ALL                                  && 关闭所有文件
USE 成绩表                                 && 打开数据表
```

```
CLEAR                                  && 清屏
LIST                                   && 显示记录
SET FILTER TO 总分>450                 && 设定过滤条件
LIST                                   && 显示记录
```

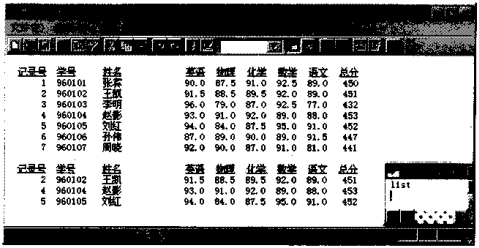


图 3.54　设置过滤条件

3.3.10　记录的查询

```
SELECT — SQL
SELECT [ALL | DISTINCT] [TOP nExpr [PERCENT]]
    [Alias.] Select_Item [AS Column_Name]
    [, [Alias.] Select_Item [AS Column_Name] ...]
FROM [FORCE][DatabaseName!]Table [Local_Alias]
    [[INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER] JOIN
        DatabaseName!]Table [Local_Alias]
        [ON JoinCondition ...
[[INTO Destination]
    | [TO FILE FileName [ADDITIVE] | TO PRINTER [PROMPT]
| TO SCREEN]]
[PREFERENCE PreferenceName]
[NOCONSOLE]
[PLAIN]
[NOWAIT]
[WHERE JoinCondition [AND JoinCondition ...]
    [AND | OR FilterCondition [AND | OR FilterCondition ...]]]
[GROUP BY GroupColumn [, GroupColumn ...]]
[HAVING FilterCondition]
```

[UNION [ALL] SELECTCommand]

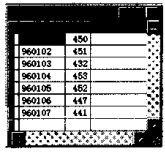
[ORDER BY Order_Item [ASC | DESC] [, Order_Item [ASC | DESC] ...]]

通过 SELECT — SQL 可以建立一个查询。其应用十分灵活，参数也有许多。上面仅列出了其语法。在这里限于篇幅，无法一一介绍。在此我们结合例子，给出一些 SELECT-SQL 命令的简单应用。读者可以通过下面的例子，掌握一些 SELECT-SQL 语句的基本应用，详细的情况，读者可以参考“命令、函数”中的有关部分。

例 1：

在本例中，我们通过 SELECT — SQL 命令选出数据表中的两个字段：

```
CLOSE ALL                          && 关闭所有文件
SELECT 学号,总分 FROM 成绩表        && 建立查询
```



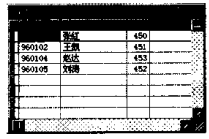
	450
960102	451
960103	432
960104	453
960105	452
960106	447
960107	441

图 3.55 建立查询例 1

例 2：

在本例中，我们从“学生表”中，选择“学号”和“姓名”字段，从“成绩表”中选择“总分”字段，两个表有条件“学生表.学号=成绩表.学号”联接。另外两个表分别以别名“a、b”打开。

```
CLOSE ALL                          && 关闭所有文件
SELECT a.学号,a.姓名,b.总分 FROM 学生表 a,成绩表 b WHERE a.学号=b.学号
```



	姓名	总分
960102	王勇	451
960104	赵达	453
960105	刘晓	452

图 3.56 建立查询例 2

例 3：

和上例相同，只是我们增加了要求，按总分排序：

```
CLOSE ALL                          && 关闭所有文件
SELECT a.学号,a.姓名,b.总分 FROM 学生表 a,成绩表;
    b WHERE a.学号=b.学号 ORDER BY 总分 DESENDING
```

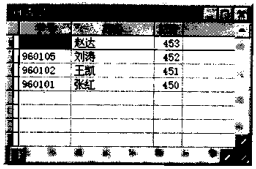


图 3.57　建立查询例 3

例 4：

和上例相同，只是我们将查询结果存在另一个表中：

```
CLOSE ALL                                          && 关闭所有文件
SELECT a.学号,a.姓名,b.总分 FROM 学生表 a,成绩表;
  b WHERE a.学号=b.学号;
  ORDER BY 总分 DESENDING;
  INTO TABLE Temp.Dbf
USE Temp                                           && 打开存储结果的表
CLEAR                                              && 清屏
LIST                                               && 列出表中记录
```

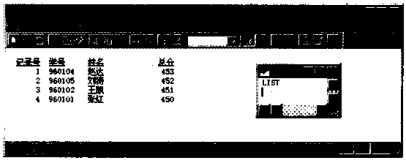


图 3.58　建立查询例 4

3.3.11　记录数据的统计计算

以上我们已经介绍了一些记录的操作，还有一些很有用的命令就是记录数据的统计和计算。通过这些命令我们可以查看数据表中的记录数，对数据表中的记录数据进行求平均、求和、求最大最小值等计算。这样，利用数据表中的原有记录数据，用户可以得到更多的信息。

COUNT ——统计数据表中的记录

```
COUNT
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    [TO VarName]
```

……

有时候，我们需要知道数据表中的记录数，尤其是符合一定条件的记录数。这时，我们就可以应用 COUNT 命令。

参数 Scope 、 FOR 、 WHILE 和以前介绍到的限定范围的参数用法完全一致，这里，我们不再重复介绍。需要提一下的是 Scope 的缺省值是全部记录（ ALL ）。

参数 TO VarName 指定一个变量或变量数组用于存储统计结果。用户在此可以指定一个不存在的变量， VFP 会自动创建。

当 SET TALK 的值为 ON 时，统计结果将被显示。

例子：

在下面的例子中，我们统计一下“总分”在 450 分以上的记录，将结果存储于变量“ total ”中。

```
CLOSE ALL                          && 关闭文件
USE 成绩表                         && 打开数据表
COUNT FOR 总分>450 TO tota         && 统计记录
?total                             && 显示结果
```

AVERAGE ——计算数值字段的算术平均值

```
AVERAGE [ExpressionList]
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    [TO VarList | TO ARRAY ArrayName]
    ……
```

该命令用于计算数值字段或者表达式的算术平均值。

参数 ExpressionList 指定一些要计算的表达式。该表达式可以是一些用逗号分隔的数值字段，也可以是包含表中字段的数值表达式。

若用户省略该表达式，则表所有的数值字段将被求平均值。

其他参数的用法与前面介绍到的相同， Scope 的缺省值为全部记录（ ALL ）。

结果的显示与否，显示时是否有标题取决于 SET TALK 和 SET HEADINGS 的值。

例子：

在下面的例子中，我们计算成绩表中各个字段的平均值：

```
CLOSE ALL                          && 关闭所有文件
USE 成绩表                         && 打开数据表
CLEAR                              && 清屏
AVERAGE                            && 计算所有数值字段的平均值
AVERAGE 英语 TO Eng                && 计算“英语”的平均值存入变量 Eng
?Eng                               && 显示 Eng 的值
```

SUM ——计算数值字段的和

```
SUM [eExpressionList]
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    [TO MemVarNameList | TO ARRAY ArrayName]
    ……
```

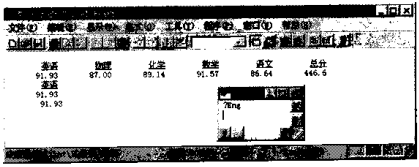


图 3.59　用 AVERAGE 命令计算平均值

该命令用于计算全部或者指定的数值字段的和。

SUM 命令的各个参数与 AVERAGE 命令完全相同，其用法也一致。在此我们就不再重复了。

例子：

还是让我们看一个 SUM 命令的例子，进一步了解该命令的使用。在本例中，我们计算各科成绩的总和。虽然在此这种计算也许是没有意义的，我们借此只是想说明 SUM 命令的用法：

```
CLOSE ALL                    && 关闭所有文件
USE 成绩表                   && 打开数据表
CLEAR                        && 清屏
SUM                          && 计算所有数值字段和
SUM 英语 TO Eng              && 计算“英语”的和存入变量 Eng
?Eng                         && 显示 Eng 的值
```

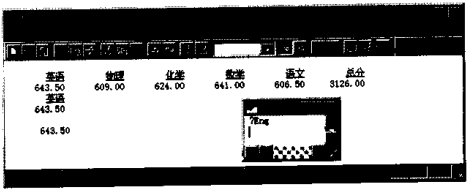


图 3.60 用 SUM 命令计算和

CALCULATE ——记录数据的统计和计算

以上我们分别介绍了用于统计的 COUNT 命令，用于计算数值字段的 AVERAGE 、SUM 命令。用户一定发现还有许多功能我们没有介绍到。下面我们就介绍一个通用的用于统计和计算的 VFP 命令。

CALCULATE eExpressionList
[Scope] [FOR lExpression1] [WHILE lExpression2]
[TO VarList | TO ARRAY ArrayName]

……

该命令用于记录数据的统计和计算，可以根据要求实现不同的目的。

参数 eExpressionList 指定一个要计算的表达式，可以包含许多函数。包含不同的函数也就实现不同的计算。这些函数包括：

AVG(nExpression)

CNT( )

MAX(eExpression)

MIN(eExpression)

NPV(nExpression1, nExpression2 [, nExpression3])

STD(nExpression)

SUM(nExpression)

VAR(nExpression)

以上函数的具体功能介绍如下：

AVG(nExpression)计算表达式的算术平均值，用该函数可以实现 AVERAGE 命令的功能。

CNT( )返回该数据表中的记录数，其功能相当于 COUNT 命令。

MAX(eExpression)返回记录中最大或者最后一个记录值。

MIN(eExpression) 返回记录中最小或者最前一个记录值。

NPV(nExpression1, nExpression2 [, nExpression3])计算以一个固定的折扣率计算的资金流动当前的净值。

nExpression1 指定一个利率值。

nExpression2 指定一个字段或者数值表达式，代表一个资金流。

nExpression3 指定一个初始投资额。

STD(nExpression)计算表达式的标准差。

SUM(nExpression)计算表达式的和。相当于 SUM 命令的功能。

VAR(nExpression)计算表达式的偏差。相当于标准差的开方值。

参数 Scope 、 FOR 、 WHILE 与前面介绍的相同。用于限定计算记录的范围。只有落在这个范围内（符合 FOR 的条件限制）的记录才参与计算。

参数 TO VarList | TO ARRAY ArrayName 指定将计算结果存储于变量或变量数组。

例子：

我们应用 CALCULATE 命令来计算“成绩表”的数据：

```
CLOSE ALL                                   && 关闭所有文件
USE 成绩表                                  && 打开数据表
CLEAR                                       && 清屏
CALCULAT AVG(英语),MAX(英语),MIN(英语);      && 计算
  TO EngAvg,EngMax,EngMin
?EngAvg,EngMax,EngMin                       && 显示结果
```

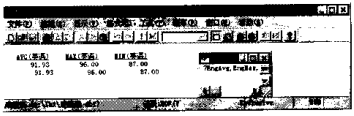


图 3.61　应用 CALCULAT 命令

3.4　数据库操作命令的应用

以上我们初步介绍了有关数据库的一些操作命令，对这些命令的了解和熟练使用是 VFP 的编程基础。

下面先让我们对以上介绍的命令进行一个简单的回顾和总结，使得用户对其有个整体的总结：

下面的表格中，我们总结了数据库操作的一些常用命令。读者既可以把它看作一个总结，也可以当作一个备查的手册。

要想掌握以上命令的使用最好的方法是实践。计算机本身就是一门实践性很强的科学，编程更是要不断实践，不断练习，才能从陌生到熟悉，从熟悉到熟练。

表 3.2　数据库操作命令

命　令	功　能
CREATE DATABASE	创建并打开一个数据库
OPEN DATABASE	打开一个已经存在的数据库
SET DATABASE TO	设定当前数据库
CLOSE DATABASE	关闭设置为当前的数据库
DELETE DATABASE	从磁盘上删除一个数据库

表 3.3　数据表操作命令

命　令	功　能
CREATE TABLE T	创建数据表
ADD TABLE	将自由表加入到数据库中
REMOVE TABLE	从当前数据库中移出数据表
USE [DatabaseName.]Table	打开数据表
USE	关闭数据表
DELETE FILE	删除文件
LIST STRUCTURE	显示数据表的字段定义
ALTER TABLE	修改数据表定义

表 3.4 记录的操作命令

命 令	功 能
DISPLAY	显示当前数据表的记录
LIST	显示当前数据表的记录
GO|GOTO	将记录指针移到指定的位置
SKIP	将记录指针移到相对于当前位置的前或者后几个记录
LOCATE	查找符合要求的记录
SEEK	查找一个匹配的记录
FIND	查找一个匹配的记录
APPEND	增加记录
REPLACE	修改数据记录
EDIT	激活编辑窗口
CHANGE	激活编辑窗口
BROWSE	浏览数据表
DELETE	标记记录为删除
RECALL	去掉删除标记
PACK	删除标记的记录
ZAP	删除表中所有记录
INDEX	建立索引
SET INDEX TO	打开 \ 关闭索引
SET ORDER TO	使用索引排序
REINDEX	重新索引
DELETE TAG	删除索引
SET FILTER TO	设定筛选条件
SELECT — SQL	建立一个查询
COUNT	统计数据表中的记录
AVERAGE	计算数值字段的算术平均值
SUM	计算数值字段的和
CALCULATE	记录数据的统计和计算

很多学习过编程的人都有这样一个体会，一个恰当的例子对于学习编程是非常有益的。往往花费很多言语无法表达清楚的情况，一个简单的例子就能达到预期效果。所以在讲解中，我们基本上每个命令都配了一些简单的例子。读者在学习的基础上，实践以下这些例子是很有帮助的。

VFP 的程序，就是一些命令加上一些流程控制的组合。当然还有我们下一章要涉及到的函数。所以，在了解了各种操作命令后，我们将各种命令组合起来，完成一个例子。

在下面的例子中，我们建立一个存放学生情况的数据库“学生库”，在该数据库中，我们建立两个数据表“学生情况”和“学生成绩”。并在这两个表中建立一些索引，输入

一些数据。完成建立过程后，我们还要进行一些查询操作。所有这些功能的实现，我们也许用的方法并不是最好、最简捷的方法，同样的功能也许用不同的命令实现。所有这些都是为了熟悉上面讲到的各种命令。相信用户在使用中自己可以比较其优劣。

例子：

```
&& 建立数据库“学生库”，在 D:\STUDENTS\目录下，该目录已经存在
   CREAT DATABASE D:\STUDENTS\学生库
&& 建立数据表“学生情况”，在数据库“学生库”中，并为其按学号索引
   CREAT TABLE D:\STUDENTS\学生情况(No C(6),Name C(10),Birth D)
   INDEX ON No TO TAG NoIndex
&& 为数据表“学生情况”输入数据
   APPEND BLANK
   REPLACE No WITH “980101”,Name WITH “张红”,Birth WITH {03/04/80}
   APPEND BLANK
   REPLACE No WITH “980102”,Name WITH “王刚”,Birth WITH {05/06/80}
&& 下面通过编辑窗口继续输入一些数据
   EDIT
&& 显示数据库信息，我们可以看到在 D:\STUDENTS\学生库.dbc 已经建立（图 3.62）
```

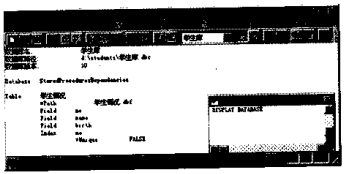


图 3.62　显示数据库信息

```
   DISPLAY DATABASE
&& 显示数据表中的记录（图 3.63）
```

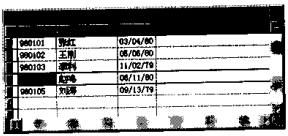
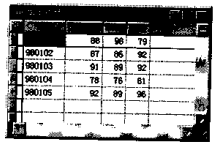


图 3.63　显示数据表中的记录

```
   BROWS
&& 至此，数据表“学生情况”已经建立
   CLOSE ALL
```

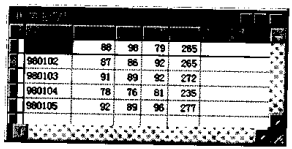
```
&& 现在建立数据表“学生成绩”，该表现在为自由表
   CREAT TABLE D:\STUDENTS\学生成绩;
   ( No C(6),Chinese N(4),Math N(4),Eng N(4))
   INDEX ON No TAG NoIndex
   USE
&& 我们将该表加入到数据库“学生库”中
   OPEN DATABASE D:\STUDENTS\学生库
   ADD TABLE D:\STUDENTS\学生成绩
&& 在数据表“学生成绩”中加入数据（见图 3.64）
```



	88	96	79	
980102	87	86	92	
980103	91	89	92	
980104	78	76	81	
980105	92	89	96	

图 3.64 加入数据

```
   EDIT
&& 在数据表“学生成绩”中，加入总分字段
&& 由于我们加入了缺省值，该字段自动计算出总分（见图 3.65）
```



	88	96	79	265
980102	87	86	92	265
980103	91	89	92	272
980104	78	76	81	235
980105	92	89	96	277

图 3.65 加入总分字段

```
   ALTER TABLE 学生成绩 ADD Total N(4) DEFAULT Chinese+Math+Eng
   INDEX ON Tatal TAG TalIndex
&& 显示数据库信息，并关闭数据库
   CLEAR
   DISPLAY DATABASE
   CLOSE ALL
&& 至此，我们完成了创建数据库和数据表的工作（见图 3.66）
&& 建立的数据表难免会出错，我们下面进行修改
&& 我们删除学号为“980105”的记录，然后重新输入该记录
   OPEN DATABASE D:\STUDENTS\学生库
   USE 学生成绩
```

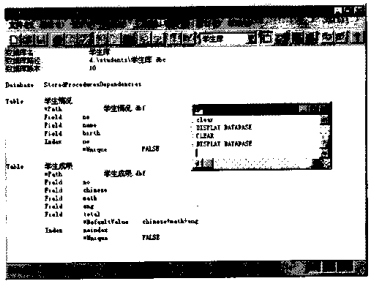


图 3.66　完成创建数据库和数据表

```
LOCATE FOR NO="980105"
DELETE
PACK
GOTO BOTTOM
APPEND BLANK
REPLACE No WITH "980105",Chinese WITH 88,Math WITH 92,Eng WITH 91;
,Total WITH Chinese+Math+Eng
CLOSE ALL
```

&& 我们完成了修改工作，下面要进行一些统计计算工作（见图 3.67）

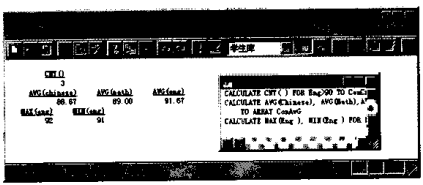


图 3.67　统计计算

```
&& 我们进行一项统计，看看共有多少学生
OPEN DATABASE D:\STUDENTS\学生库
USE D:\STUDENTS\学生情况 IN 1
COUNT TO TalStud
&& 下面我们计算以下各科的平均分
USE D:\STUDENTS\学生成绩 IN 2
SELECT 2
```

```
    AVERAGE Chinese,Math,Eng TO AveChin,AveMath,AveEng
&& 我们换一个命令，统计计算一下英语（Eng）成绩在90分以上的记录
    CALCULATE CNT( ) FOR Eng>90 TO ConCnT
        CALCULATE AVE(Chinese ), AVE(Math),AVE(Eng ) FOR Eng>90;
        TO ARRAY ConAve
    CALCULATE MAX(Eng ), MIN(Eng ) FOR Eng>90 TO ConMax,ConMin

&& 我们完成了统计计算工作，下面要进行一些查询工作
&& 应用 SELECT — SQL 语句建立查询
    SELECT a.No,a.Name,a.Birth,b.Chinese,b.Math,b.Eng,b.Total;
        FROM 学生情况 a,学生成绩 b WHERE a.No=b.No;
        TO FILE D:\STUDENTS\QurFile
&& 我们打开该查询结果文件（见图 3.68）
```

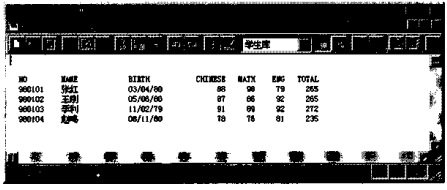


图 3.68 查询结果

```
&& 我们完成了所有工作，可以关闭数据库了
    CLOSE ALL
```

上面我们给出了一个比较完整且较为贴近实用的数据库操作的例子，在这个例子中，我们用到了许多很有用的命令。希望读者可以自己尝试一下以上的命令。还可以变换一些方式，实现更多的功能。当然这仅仅是学习 VFP 编程的第一步，但确是 VFP 编程的基础。

该例子中的命令我们都是在“命令”窗口中调试的，因此还不能算得上一个程序。我们只是希望通过这个例子，读者可以对数据库的操作命令有一个更加深刻的了解，能更熟练的使用。当然在上面的例子中，加入一些输入输出的控制后，就可以完成一定的功能了。

在上面的例子中，有很多输出，我们在下面的图中给出。读者可以参考这些输出，进一步体会以上命令的功能和用法。

3.5 总　　结

本章中，我们介绍了一些常用的数据库命令。这些命令不仅在数据库操作中可以方便的使用，其应用更是 VFP 编程的基础。

我们介绍了这些命令的主要语法、参数和功能，通过一定的例子，读者可以对其有更好的了解。但限于篇幅，我们在此无法将所有的命令，所有的功能和详细的参数描述都呈现给大家。需要时，读者可以参考“命令、函数”的有关部分。另外也可以参考 VFP 提供的联机帮助文件。

第四章　FoxPro 的常用函数的用法

进行过程序开发的用户，不管使用的是 VC++还是 VFP，都会对函数的作用有很深的体会。甚至于无法想象没有了系统提供的数以百计的函数，程序将如何完成。

VFP 中包含了数百个系统提供的内建函数。这些函数构成了 VFP 程序开发的核心。用户同样也可以定义自己的函数，以完成独特的功能。

4.1　函数的使用方法

VFP 中的函数使用方法与其他程序设计语言调用函数的方法是一样的。所以对于程序设计者来说，使用 VFP 的函数不会有任何的不适应。

在 VFP 中，调用函数只需简单的在函数名后加上一对小括号，例如：MyFunc( )。

传递参数只要在括号中填入参数即可，例如：MyFunc(Value)。

当要传递的参数不止一个时，各个参数之间要有逗号隔开，例如：MyFunc(Val1 , Val2)。函数的返回值可以保存在变量中，其用法如下：

```
Var=MyFunc( )          && 将函数的返回值保存于 Var 变量中
```

也可以将返回值直接显示在屏幕上：

```
? MyFunc( )            && 在活动输出窗口中打印函数的返回值
```

关于函数的使用，经常会有一些常见的错误。如函数名键入错误，系统将找不到该函数，会报错。

图 4.1　函数名键入错误

还有一类常见错误是函数的输入参数类型与函数要求输入的参数类型不匹配。从而造成了 VFP 报错。

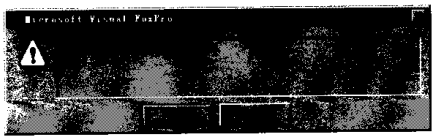


图 4.2　输入参数类型不匹配

以上我们简单介绍了 VFP 中函数的使用方法，相信对于有过编程经验（任何高级语言）的用户来说，函数的使用没有任何困难。若用户是初次接触编程，可以根据以上的介绍反复试验。只有经过自己的操作和练习，才能真正掌握。

我们已经提到，VFP 系统提供给用户的内建函数有数百个之多。学习和掌握这些函数的功能和使用，是学习 VFP 编程的基础。也只有掌握了其使用方法，用户才能比较轻松地开发出满意的程序。

限于篇幅，我们无法对 VFP 提供的函数都做详细介绍，而只能选择其中较为常用的函数，做一个初步的介绍。希望以此为契机，使用户了解 VFP 提供的功能强大的内建函数。

4.2　字符处理函数

VFP 作为一个数据库系统，字符的处理毫无疑问的是其主要功能。因此字符类型在 VFP 中有着重要的作用，有关字符的处理函数是 VFP 的内建函数中十分重要的部分。VFP 中的字符类型指单个字符或者一个字符型表达式，我们通常说的字符串也包括在其中。

由于其独特的重要地位，VFP 提供了许多处理字符的函数，从字母大小写的判断、转换到字符表达式的匹配。限于篇幅，我们在此只能介绍一部分常用的函数。其余的函数，用户可以参考“命令、函数”部分的有关章节。

4.2.1　去掉字符表达式中的头尾空格

ALLTRIM( ) | L TRIM( ) | RTRIM( )

在输入输出过程中，字符表达式中头尾的空格往往并不是我们所需要的。以上三个函数都用来处理函数头尾的空格，但其用法和功能又有一些不同之处。

语法：　ALLTRIM(cExpression)
　　LTRIM(cExpression)
　　RTRIM(cExpression)

参数：　cExpression 为要处理的字符表达式。

返回：　去掉了头尾空格的字符表达式。

以上函数去掉字符型表达式 cExpression 中的头和尾的空格字符。主要可以用于在输入时，确认输入的字符中，没有头尾的无用的空格字符。但其使用上是有差别的：

- 函数 LTRIM( )去掉头的空格
- 函数 RTRIM( )去掉尾的空格
- 函数 ALLTRIM( )去掉头尾的空格，相当与以上两个函数的联合使用。

例子：

```
? ALLTRIM （“  HELLO WORLD！  ”） && 返回为“HELLOE WORLD！”
? LTRIM （“  HELLO WORLD！  ”）  && 返回为“HELLOE WORLD！”
? RTRIM （“  HELLO WORLD！  ”）  && 返回为“  HELLOE WORLD！”
```

4.2.2 查找字符函数

AT( ) | ATC( ) | AT_C( ) | ATCC( )

字符型表达式的处理中，我们经常需要在一个给定的字符型表达式中，查找与指定字符表达式相匹配的一段字符串的位置。也就是在给定的字符型表达式中，查找另一个字符串。另外该函数还可以在备注字段的查找中给定的字符表达式。

语法： AT(cSearchExpression, cExpressionSearched [, nOccurrence])
ATC(cSearchExpression, cExpressionSearched [, nOccurrence])
AT_C(cSearchExpression, cExpressionSearched [, nOccurrence])
ATCC(cSearchExpression, cExpressionSearched [, nOccurrence])

参数： cSearchExpression 指定字符表达式，函数将搜索此字符表达式。
cExpressionSearched 指定在其中进行搜索的字符表达式。
nOccurrence 指定搜寻 cSearchExpression 在 cExpressionSearched 中的第几次（第一、第二、第三次等等）出现。默认情况下，为首次出现（ nOccurrence=1)。使用 Occurrence 参数可以搜索 cSearchExpression 在 cExpressionSearched 中其他的出现，如果参数 nOccurrence 的值大于 cExpressionSearched 中包含 cExpressionSearched 的数目，函数将返回 0 。

返回： 数值型，表示匹配处的位置。

以上四个函数的用法有一些差别：

- 函数 AT( )对大小写敏感。
- 函数 ATC( ) 对大小写不敏感。
- 函数 AT_C( )专为双字节字符而设计，对大小写敏感。
- 函数 ATCC( ) 专为双字节字符而设计，对大小写不敏感。

当字符中只有单字节时，函数 AT_C( ) | ATCC( )与函数 AT( ) | ATC( )等价。

例子：

```
STORE 'Now we want to test this function'
TO gcString
STORE 'we' TO gcFindString
CLEAR
? AT(gcFindString,gcString)                    && 显示 5
STORE 'WANT' TO gcFindString
? AT(gcFindString,gcString)                    && 显示 0，该函数对大小写敏感
? ATC(gcFindString,gcString)                   && 显示 8，该函数对大小写不敏感
```

ATLINE( ) | ATCLINE( )

有时，我们不光要知道位置，匹配处的行号对于定位也十分重要。这两个函数就是返回一个字符表达式或备注字段在另一个字符表达式或备注字段中首次出现的行号。

语法： ATLINE(cSearchExpression, cExpressionSearched)

ATCLINE(cSearchExpression, cExpressionSearched)

参数： cSearchExpression 指定字符表达式，函数查找此表达式。

cExpressionSearched 指定在其中进行搜索的表达式。

返回： 数值型，表示匹配处的行号。

该函数与函数 AT( )功能相似，用于查找匹配的第一次字符。返回该匹配出现的行号。这两个函数的用法有一些差别：

- 函数 ATLINE( )对大小写敏感。
- 函数 ATCLINE( )对大小写不敏感。

OCCURS( )

有时我们关心的是一个字符表达式在另一个字符表达式中出现的次数。以下的函数实现这一功能。

语法： OCCURS(cSearchExpression, cExpressionSearched)

参数： cSearchExpression 指定字符表达式，函数在 cExpressionSearched 中查找该表达式。

cExpressionSearched 指定一字符表达式， OCCURS( )在其中查找 cSearchExpression 字符表达式。

返回： 该函数返回出现的次数，为一数值型变量。

该函数用于查找一个字符表达式 cSearchExpression 在另一个表达式 cExpressionSearched 中出现的次数。

例子：

```
? OCCURS("a","abcaabccc")          && 显示 3
? OCCURS("b","abcaabccc")          && 显示 2
? OCCURS("c","abcaabccc")          && 显示 4
```

INLIST( )

判断一个表达式是否与一组表达式中的某一个相匹配

语法： INLIST(eExpression1, eExpression2 [, eExpression3 ...])

参数： eExpression1 指定 INLIST( ) 函数要在表达式组中搜索的表达式。

eExpression2 [, eExpression3 ...] 指定要搜索的表达式组。表达式组中必须至少包含一个表达式(eExpression2)，最多可包含 24 个。表达式组中的所有表达式必须具有相同的数据类型。

返回： 逻辑型或 null 值，如 INLIST( )函数在表达式组中找到了要搜索的表达式，就返回“真”(.T.)；否则，返回“假”(.F.)。如果 eExpression1 为 null 值，则 INLIST( )函数返回 null 值；如果 eExpression1 与表达式组中的任何表达式都不匹配，或者表达式组中有一个表达式为 null 值， INLIST( ) 函数也返回 null 值。

例子：

```
? INLIST("hello","hello","world")          && 显示.T.
```

```
? INLIST("ello","hello","world")          && 显示.F.
```

4.2.3 替换字符函数

CHRTRAN( ) | CHRTRANC( )

以上我们介绍了用于查找的函数，找到后进行替换是用户经常需要的功能。VFP 专门提供了查找并替换的函数。

语法： CHRTRAN(cSearched, cSearchFor, cReplacement)
CHRTRANC(cSearched, cSearchFor, cReplacement)

参数： cSearched 指定字符表达式，函数替换其中的字符。
cSearchFor 指定字符表达式，函数在其中寻找该表达式的字符。
cReplacement 指定包含替换字符的表达式。

返回： 字符型，替换后的字符表达式。

该函数查找匹配的字符，并将其进行替换。

当 cSearchFor 中的一个字符在 cSearched 中被发现时， cSearchedFor 中的字符被 cReplacement 中的字符替换。当替换字符比匹配字符数小时，多出的字符被删除；当替换字符比匹配字符多时，多出的字符被忽略。

函数 CHRTRANC( )用来对含有双字节的字符进行替换。当字符表达式中只有单字节时，其功能与函数 CHRTRAN( )相同。

例子：

```
? CHRTRAN('ABCDEF', 'ACE', 'XYZ')          && 显示 " XBYDZF "
? CHRTRAN('ABCD', 'ABC', 'YZ')             && 显示 " YZD "
? CHRTRAN('ABCDEF', 'ACE', 'XYZQRST') && 显示 " XBYDZF "
```

4.2.4 截取字符表达式中的一段

LEFT( ) | RIGHT( ) | LEFTC( ) | RIGHTC( )

我们可以在给定的一个字符表达式中，截取一段字符。这种截取可以是从最左或者最右开始的任意个字符的一段。

语法： LEFT(cExpression, nExpression)
RIGHT(cExpression, nExpression)
LEFTC(cExpression, nExpression)
RIGHTC(cExpression, nExpression)

参数： cExpression 指定字符表达式，函数从中返回字符。
nExpression 指定从字符表达式中返回的字符个数。若 nExpression 的值大于 cExpression 的长度，则返回字符表达式的全部字符。如果 nExpression 为负值或 0，则返回空字符串。

返回： 字符型，字符表达式 cExpression 中的最左（最右）的 nExpression 个字符。

其中 LEFT( )从最左开始， RIGHT( )从最右开始。

而 LEFTC( ) | RIGHTC( )专为双字节字符表达式设计。

例子：

```
? LEFT ( " HELLO WORLD " ,  2 )        && 显示 " HE "
? RIGHT( " HELLO WORLD " ,  2 )        && 显示 " LD "
```

SUBSTR( ) | SUBSTRC( )

用户可能已经注意到，在上面的两个函数中，我们只能从最左或者最右开始，截取字符表达式。而该函数可以任意截取。

语法：　SUBSTR(cExpression, nStartPosition [, nCharactersReturned])

SUBSTRC(cExpression, nStartPosition [, nCharactersReturned])

参数：　cExpression 指定要从其中返回字符串的字符表达式或备注字段。

nStartPosition 指定返回的字符串在字符表达式或备注字段 cExpression 中的位置，cExpression 的第一个字符是位置 1。

nCharactersReturned 从 cExpression 中返回的字符数目。如果省略了 nCharactersReturned 参数，那么返回字符表达式结束前的全部字符。

返回：　字符型，截取的字符表达式。

注意：如果 TALK 设置为 ON，并且 nStartPosition 大于 cExpression 中的字符数目，那么 VFP 产生错误信息；如果 TALK 设置为 OFF，那么返回一个空字符串。

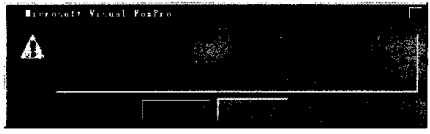


图 4.3　错误信息

显然这个函数更具一般性。同样的 SUBSTRC( )专为双字节设计。

例子：

```
? SUBSTR( " HELLO WORLD " ,  2 ,  5)     && 显示 " ELLO "
? SUBSTR( "你好，世界" ,  3 ,  5)          && 显示 "好，"
? SUBSTRC( "你好，世界" ,  3 ,  5)         && 显示 "，世界"
```

4.2.5　计算字符表达式长度

LEN( ) | LENC( )

以下的函数用来计算一个字符表达式的长度，并返回。

语法：　LEN(cExpression)

LENC(cExpression)

参数：　cExpression 指定字符表达式，LEN( ) 函数返回其字符数目。

返回： 数值型，为字符表达式的长度。

同样的 LENC( )专为双字节字符表达式设计。

例子：

```
?LEN(“ HELLO,WORLD ”)          && 显示 11
?LEN(“你好，世界”)             && 显示 10
?LENC(“你好，世界”)            && 显示 5
```

4.2.6 判断字符类型的函数

对于一个字符型表达式，我们经常要知道其是字母还是数字。

ISALPHA( )

该函数用于判断字符表达式中，最左一个字符是否为字母。当然如果该表达式仅为一个字符，该函数可以判断该字符是否为字母。

语法： ISALPHA(cExpression)

参数： cExpression 函数所要判断的字符表达式。 cExpression 中第一个字符之后的所有字符都将被忽略。

返回： 逻辑型（ Logical ），当为字母时返回真（.T.），否则返回假（.F.）。

该函数事实上只判断最左的一个字符，其他字符则被忽略。

例子：

```
?ISALPHA （“ Test ”）          && 返回值为.T.
?ISALPHA （“ test ”）          && 返回值为.T.
?ISALPHA （“ 1test ”）         && 返回值为.F.
```

ISDIGIT( )

该函数判断字符表达式 cExpression 的最左边一个字符是否为数字（ 0 到 9 ）。其返回值类型为逻辑型。

语法： ISDIGIT(cExpression)

参数： cExpression 指定函数要判断的字符表达式。 cExpression 中第一个字符之后的所有字符都被忽略。

返回： 逻辑型，如果字符表达式的最左边一个字符是数字（ 0 到 9 ），则 ISDIGIT( ) 函数返回“真”(.T.)；否则， ISDIGIT( ) 函数返回“假”(.F.)。

该函数事实上只判断最左的一个字符，其他字符则被忽略。

例子：

```
?ISDIGIT(“6HELLO”)             && 显示.T.
?ISDIGIT(“HELLO”)              && 显示.F.
```

4.2.7 判断字符大小写的函数

ISLOWER( ) | ISUPPER( )

该函数用于判断字符表达式 cExpression 中，最左一个字符的大小写。当然如果该表达

式仅为一个字符，该函数可以判断该字符的大小写。

语法： ISLOWER(cExpression)
ISUPPER(cExpression)

参数： cExpression 指定函数要判断的字符表达式。 cExpression 中第一个字符之后的所有字符都被函数忽略。

返回： 该函数的返回值为一个逻辑值（ Logical ）。

- ISLOWER( ) 当为小写字符时返回真（.T.），否则返回假（.F.）。
- ISUPPER( ) 当为大写字符时返回真（.T.），否则返回假（.F.）

该函数事实上只判断最左的一个字符，其他字符则被忽略。

例子：

```
?ISLOWER （“ Test ”）          && 返回值为.F.
?ISLOWER （“ test ”）          && 返回值为.T.
?ISUPPER （“ Test ”）          && 返回值为.T.
?ISUPPER （“ test ”）          && 返回值为.F.
```

4.2.8 字符大小写转换函数

LOWER( ) | UPPER( )

以下的函数用来改变字符型表达式 cExpression 的大小写。

语法： LOWER(cExpression)
UPPER(cExpression)

参数： cExpression 指定要由函数转换的字符表达式。

返回： 字符型，改变后的字符。

该函数只改变字符表达式中的字母，而其他字符保持不变。

例子：

```
?LOWER(“ HELLO WORLD ”)        && 返回“ hello world ”
?UPPER(“ hello world ”)        && 返回“ HELLO WORLD ”
```

4.2.9 其他关于字符的函数

ASC( )

返回字符表达式中最左边字符的 ANSI 值

语法： ASC(cExpression)

参数： cExpression 指定字符串表达式。函数仅返回 cExpression 中第一个字符的 ANSI 值，忽略其他字符。

返回： 数值型，返回字符表达式中最左边字符的 ANSI 值。

例子：

```
?ASC("C")                      && 显示 67
?ASC("c")                      && 显示 99
```

CHR( )

返回指定的 ANSI 数值代码对应的字符

语法： CHR(nANSICode)

参数： nANSICode 指定一个介于 0 和 255 之间的数值，CHR( ) 返回与之对应的 ANSI 字符。

该函数的功能与 ASC( ) 正好相反。

例子：

```
?CHR （ ASC("C")）                    && 显示 " C "
?CHR （ 99 ）                         && 显示 " c"
```

REPLICATE( )

该函数将指定字符表达式重复指定次数后，形成新的字符型表达式。

语法： REPLICATE(CExpression, nTimes)

参数： cExpression 指定要重复的字符表达式。

nTimes 指定字符表达式的重复次数。

返回： 字符型，返回新形成的字符型表达式。

该函数将指定的字符型表达式 cExpression 重复 nTimes 次，而形成新的字符型表达式。

例子：

```
?REPLICATE(" HELLO ", 3)              && 显示 " HELLO HELLO HELLO "
```

SPACE( )

该函数由指定个数的空格组成字符串。

语法： SPACE(nSpaces)

参数： nSpaces 指定函数返回的空格数目。

返回： 字符型，返回由空格组成的字符串。

该函数将产生 nSpaces 个空格，并返回。

例子：

```
? " HELLO "+SPACE （3）+ " WORLD "&& 显示 " HELLO    WORLD "
```

4.3 数值处理函数

数值类型是数据库中又一十分重要且常用的类型。VFP 提供了常用的用于数值处理的函数，可以完成用户对数值计算的要求。VFP 提供的数值函数与一般用户常用的数学函数名基本一致，用户在使用时十分方便。

下面我们分别介绍一下 VFP 提供的用于数值计算的函数。读者可以看到这些函数的使用十分方便。

4.3.1 三角函数

SIN( )
正弦函数
语法： SIN(nExpression)
参数： nExpression 函数返回正弦值的角度。nExpression 可以取任何值，并且函数的返回值在 -1 和 1 之间。但要注意的是，这里使用弧度作为输入值。用户可以使用 VFP 同时提供的函数 DTOR( )将角度转换为弧度。
返回： 数值型，返回表达式 nExpression 的正弦值。
例子：

```
? SIN(PI( )/2)                    && 显示1.00
? SIN(DOTOR(90 ))                 && 显示1.00
```

COS( )
余弦函数
语法： COS(nExpression)
参数： nExpression 指定一个需要返回余弦值的数值表达式。nExpression 可以是任意值。
返回： 数值型，返回表达式 nExpression 的余弦值。
这里使用弧度作为输入值，VFP 同时提供函数 DTOR( )将角度转换为弧度。
例子：

```
? COS(PI( ))  && 显示              1.00
? COS(DOTOR(180 ))  && 显示        1.00
```

TAN( )
正切函数
语法： TAN(nExpression)
参数： nExpression 指定一个需要返回正切值的数值表达式。
返回： 数值型，返回数值型表达式 nExpression 的正切值。
这里使用弧度作为输入值，VFP 同时提供函数 DTOR( )将角度转换为弧度。
例子：

```
? TAN(PI( )/4)                    && 显示1.00
? TAN(DOTOR(45 ))                 && 显示1.00
```

4.3.2 反三角函数

ASIN( )
反正弦函数
语法： ASIN(nExpression)
参数： nExpression 指定数值表达式，ASIN( ) 函数返回此表达式的反正弦值。

nExpression 的取值范围为 -1 到 +1，ASIN( ) 函数返回值的值域范围为 -π/2 到 +π/2 (-1.57079 到 1.57079)。显示结果中小数点的位置可以用 SET DECIMAL 命令指定。

返回：数值型，返回数值型表达式 nExpression 的反正弦值。

这里使用弧度作为输出值，VFP 同时提供函数 RTOD( )将弧度转换为角度。

例子：

```
? RTOD(ASIN(0))                    && 显示 0.00
? RTOD(SIN(SQRT(2)/2 ))            && 显示 45.00
```

ACOS( )

反余弦函数

语法：ACOS(nExpression)

参数：nExpression 指定一个数值表达式，ACOS( ) 函数返回此表达式的反余弦值。nExpression 参数的取值范围从 -1 到 +1，ACOS( ) 函数的值域从 0 到 π (3.141592)。ACOS( ) 函数返回数值的小数位数由 SET DECIMALS 指定。

返回：数值型，返回数值型表达式 nExpression 的反余弦值。

这里使用弧度作为输出值，VFP 同时提供函数 RTOD( )将弧度转换为角度。

例子：

```
? RTOD(COS(0))                     && 显示 90.00
? RTOD(ACOS(SQRT(2)/2 ))           && 显示 45.00
```

ATAN( )

反正切函数

语法：ATAN(nExpression)

参数：nExpression 指定一个数值表达式，ATAN( ) 函数返回此数值表达式的反正切值。nExpression 可为任意值。ATAN( ) 返回值的值域范围从 -π/2 到 +π/2 (-1.57079 到 1.57079)。返回值显示到小数点后第几位数由 SET DECIMALS 决定。

返回：数值型，返回数值型表达式 nExpression 的反正切值。

这里使用弧度作为输出值，VFP 同时提供函数 RTOD( )将弧度转换为角度。

例子：

```
? RTOD(ATAN(1.0))                  && 显示 45.00
? RTOD(ATAN(0 ))                   && 显示 0.00
```

4.3.3 指数函数

EXP( )

计算 e^x 的值

语法：EXP(nExpression)

参数：nExpression 指定指数表达式 e^x 中的指数 x。

返回：数值型，返回自然指数 e 的 nExpression 次方。

这里，自然对数的底 e 约等于 2.71828。EXP( ) 函数返回的小数位数由 SET

DECIMALS 命令指定。

例子：

```
? EXP(0)                    && 显示 1.00
? EXP(1)                    && 显示 2.72
```

SQRT( )

平方根函数

语法： SQRT(nExpression)

参数： nExpression 函数要计算的数值表达式，由平方根的定义， nExpression 不能是负值。

返回： 数值型，返回将 nExpression 开方后的值。

例子：

```
? SQRT(100)                 && 显示 10.00
```

4.3.4 对数函数

LOG( )

自然对数（底数为 e 的对数）

语法： LOG(nExpression)

参数： nExpression 指定数值表达式， LOG( )函数返回方程 e^x = nExpression 中的 x 值。由对数的定义， nExpression 必须大于 0 。

返回： 数值型，返回以自然指数 e 为底的对数值。

例子：

```
? LOG(1)                    && 显示 0.00
? LOG(EXP(2))               && 显示 2.00
```

LOG10( )

常用对数（以 10 为底的对数）

语法： LOG10(nExpression)

参数： nExpression 指定数值表达式， LOG10( ) 函数返回方程 10^x = nExpression 的 x 值。由对数的定义， nExpression 必须大于 0 。

返回： 数值型，返回以 10 为底的对数值。

例子：

```
? LOG10(1)                  && 显示 0.00
? LOG10(100)                && 显示 2.00
```

4.3.5 截断函数

对于一个要截断的数来说，截断的方法有许多种：向上截断、向下截断、四舍五入截断等。各种截断方式可以满足不同的要求。 VFP 对应于这些截断的方法，提供了不同的函数，以实现用户的不同需要。

CEILING( )
向上取整函数
语法： CEILING(nExpression)
参数： nExpression 指定一个数值， CEILING( ) 返回其后续的整数。
返回： 数值型，返回大于等于数值 nExpression 的整数，也就是说将通过将小数部分进位而得到最接近的整数。
例子：

```
? CEILING(12)                    && 显示 12
? CEILING(12.3)                  && 显示 13
? CEILING(12.5)                  && 显示 13
? CEILING(-12.5)                 && 显示-12
```

FLOOR( )
向下截断
语法： FLOOR(nExpression)
参数： nExpression 指定数值型表达式， FLOOR( ) 函数返回小于或等于此数值型表达式值的最大整数。
返回： 数值型，返回小于等于数值 nExpression 的整数，也就是说将通过将小数部分舍去而得到最接近的整数。
例子：

```
? FLOOR(12)                      && 显示 12
? FLOOR(12.3)                    && 显示 12
? FLOOR(12.5)                    && 显示 12
? FLOOR(-12.5)                   && 显示-13
```

INT( )
取整函数
语法： INT(nExpression)
参数： nExpression 指定 INT( ) 要计算的数值表达式。
返回： 数值型，返回数值 nExpression 的整数部分，也就是说将通过将小数部分舍弃而得到最接近的整数。
例子：

```
? INT(12)                        && 显示 12
? INT(12.3)                      && 显示 12
? INT(12.5)                      && 显示 12
? INT(-12.5)                     && 显示-12
```

ROUND( )
四舍五入到指定的小数位

语法： ROUND(nExpression, nDecimalPlaces)

参数： nExpression 指定要圆整的数值表达式。

nDecimalPlaces 指定 nExpression 圆整到的小数位数。

返回： 数值型，截断后的数值。

如果 nDecimalPlaces 为负数，则 ROUND( ) 返回的结果在小数点左端包含 nDecimalPlaces 个零。例如，如果 nDecimalPlaces 为 -2 ，那么小数点左端的第一和第二个数字(个位和十位)均为 0 。

例子：

```
? ROUND(1234.1962, 3)          && 显示 1234.1960
? ROUND(1234.1962, 2)          && 显示 1234.2000
? ROUND(1234.1962, 0)          && 显示 1234.0000
? ROUND(1234.1962, -1)         && 显示 1230.0000
? ROUND(1234.1962, -2)         && 显示 1200.0000
? ROUND(1234.1962, -3)         && 显示 1000.0000
```

4.3.6 绝对值函数

ABS( )

绝对值函数

语法： ABS(nExpression)

参数： nExpression 指定需由 ABS( )返回绝对值的数值表达式。

返回： 数值型，返回一个数值型表达式的绝对值。

例子：

```
? ABS(-10)                     && 显示 10
? ABS(4-10)                    && 显示 6
? ABS(10)                      && 显示 10
```

4.3.7 最大最小值函数

MAX( )

求几个表达式中的最大值

语法： MAX(eExpression1, eExpression2 [, eExpression3 ...])

参数： eExpression1, eExpression2 [, eExpression3 . . .]指定表达式，MAX( ) 返回其中具有最大值的表达式。所有表达式必须为同一数据类型。输入的参数可以为多种类型：字符型、数值型、货币型、双精度、浮动型、日期型或者日期时间型。

返回： 字符型、数值型、货币型、双精度型、浮点型、日期型或日期时间型。返回各个表达式中最大的一个。

例子：

```
? MAX(1,2,3,4)                 && 显示 4
```

```
? MAX("D","A","B")                    && 显示"D"
```

MIN( )

求几个表达式中的最小值

语法： MIN(eExpression1, eExpression2 [, eExpression3 ...])

参数： eExpression1, eExpression2 [, eExpression3 ...] 指定一组表达式，MIN( ) 返回其中具有最小值的表达式。所有表达式应该是同一数据类型。输入的参数可以为多种类型：字符型、数值型、货币型、双精度、浮动型、日期型或者日期时间型。

返回： 字符型、数值型、货币型、双精度型、浮点型、日期型或日期时间型。返回各个表达式中最小的一个。

例子：

```
? MIN(1,2,3,4)                        && 显示 1
? MIN("D","A","B")                    && 显示"A"
```

4.3.8 求余函数

MOD( )

求两个数相除的余数

语法： MOD(nDividend, nDivisor)

参数： nDividend 指定被除数。nDividend 中的小数位数决定了返回值中的小数位。

nDivisor 指定除数。若 nDivisor 为正数，返回值为正；若 nDivisor 为负数，返回值为负。

返回： 数值型，返回两个数相除而得的余数。

例子：

```
? MOD(12,5)                           && 显示 2
```

4.3.9 其他常用函数

PI( )

返回常数 π

语法： PI( )

返回： 数值型，返回圆周率常数。其结果中的小数点位置由 SET DECIMALS 命令决定。

例子：

```
? PI( )                               && 显示 3.14
```

RAND( )

随机数函数

语法：　RAND([nSeedValue])

参数：　nSeedValue 指定种子数值，它决定 RAND( ) 函数返回的数值序列。

在第一次发出 RAND( ) 函数时用种子数 nSeedValue ，然后再使用不带 nSeedValue 参数的 RAND( ) 函数，将得到一个随机数序列。如果第三次发出 RAND( ) 函数时使用同样的种子数值 nSeedValue ，那么 RAND( )返回同样的随机数序列。

如果第一次发出 RAND( ) 时使用的 nSeedValue 参数是负数，那么将使用来自系统时钟的种子值。若要获得随机程度最大的数字序列，可以最初用一个负的参数发出 RAND( ) 函数，然后再不带参数发出 RAND( ) 函数。

如果省略了 nSeedValue 参数，RAND( ) 函数使用默认的种子数值 100001 。

返回：　数值型，0 ～ 1 的一个随机数。

例子：

```
? RAND( )                              && 显示结果不定
```

SIGN( )

符号函数

语法：　SIGN(nExpression)

参数：　nExpression 指定用 SIGN( ) 函数进行求值的数值表达式。

返回：　数值型，当该数为正时返回 1 ；为负时返回-1 ；为 0 时返回 0 。

例子：

```
? SIGN(10 )                            && 显示 1
? SIGN(-10 )                           && 显示-1
? SIGN(0 )                             && 显示 0
```

VAL( )

由数字组成的字符表达式返回数字值

语法：　VAL(cExpression)

参数：　cExpression 指定字符表达式。

返回：　数值型，将字符型表达式 cExpression 转换为数值型表达式并返回。

注意：　VAL( ) 函数从左到右返回字符表达式中的数字，直至遇到非数值型字符（忽略前面的空格）时为止。若字符表达式的第一个字符不是数字，也不是加、减号，则 VAL( ) 函数返回 0 。

例子：

```
? VAL("12"+"12")                       && 显示 1212.00
? VAL("12"+"."+"12")                   && 显示 12.12
```

4.4　日期和时间处理函数

在数据库中，由于经常要处理一些与时间相关的资料，所以时间和日期也是 VFP 中十分重要的数据类型。 VFP 系统提供了许多处理时间和日期的函数。

4.4.1 系统日期时间

DATE( )
系统日期
语法： DATE( )
返回： 日期型，返回当前的系统日期。
例子：
? DATE( ) && 显示当前日期，如 03/04/98 （显示根据具体日期而不同）

DATETIME( )
系统日期时间
语法： DATETIME( )
返回： 日期时间型，返回当前的系统日期和时间。
例子：
? DATETIME( ) && 显示当前日期时间。（显示根据具体日期时间而不同）

4.4.2 日期时间函数

YEAR( )
返回年份
语法： YEAR(dExpression | tExpression)
参数： dExpression 指定日期表达式， YEAR( ) 从该表达式返回年份。参数 dExpression 可以是返回值为日期或日期型内存变量、数组元素或字段的函数；也可以是日期字符串。
tExpression 指定日期时间表达式。YEAR( ) 从该表达式返回年份。
返回： 数值型，从指定的日期表达式或日期时间表达式中返回年份。
例子：
? YEAR(DATE()) && 显示 1998

MONTH( )
返回月份
语法： MONTH(dExpression | tExpression)
参数： dExpression 指定一日期表达式，用 MONTH( )函数返回其月份值。
tExpression 指定一日期时间表达式，用 MONTH( ) 函数返回其月份值。
返回： 数值类型，从指定的日期表达式或日期时间表达式中返回月份。
例子：
? MONTH(DATETIME()) && 显示 3，当前为三月

WEEK( )

返回星期

语法：　WEEK(dExpression | tExpression [, nFirstWeek] [, nFirstDayOfWeek])

参数：　dExpression | tExpression 指定日期或日期时间表达式，WEEK( )函数从这个表达式中返回一年中的周的序号。如果省略可选参数 nFirstWeek 和 nFirstDayOfWeek，则 WEEK( ) 函数把“星期日”作为每周的第一天。

nFirstWeek 指定一年中的第一周。

nFirstDayOfWeek 指定每周的第一天。

nFirstWeek 和 nFirstDayOfWeek 的取值如下表所示：

表 4.1　nFirstWeek 取值说明

nFirstWeek	说　明
0	WEEK( )函数返回当前选定的周，选定值在“选项”对话框中“国际”选项卡中的“一年的第一周”列表中
1	第一周包含1月1日，省略nFirstWeek时为默认值
2	第一周后半部分（四天）在当前年内
3	第一周有7天

表 4.2　nFirstDayOfWeek 取值说明

nFirstDayOfWeek	说　明
0	WEEK( ) 函数返回当前选定的日，选定值在“选项”对话框中“国际”选项卡内“星期开始于”列表内。
1	星期日。省略 nFirstDayOfWeek 时的默认值，并且是早期 FoxPro 版本中每周的第一天。
2	星期一
3	星期二
4	星期三
5	星期四
6	星期五
7	星期六

返回：数值型，函数返回 1 到 53 之间的一个数，该数代表一年中周的序号。例如，WEEK( )返回 1，表明是一年中的第一周；返回 2，为一年中的第二周，依此类推。

注意：一周可以分在两年之中；一年的第一周可以在当前年和前一年之中。

例子：

```
? WEEK(DATE())                    && 显示今天所在周的序号
```

DAY( )

返回日期

语法：　DAY(dExpression | tExpression)

参数： dExpression 指定一个日期， DAY( ) 返回该日期是某月中的第几天。dExpression 可以是一个日期字符串，或是一个日期型内存变量、数组元素或字段。

tExpression 指定一个日期时间， DAY( ) 返回该日期时间是某月中的第几天。tExpression 可以是一个日期时间字符串，或是一个日期时间型内存变量、数组元素或字段。

返回：数值型，返回日期。

例子：

```
? DAY({03/04/98})                    && 显示日期，本例为 4
```

HOUR( )

返回小时

语法： HOUR(tExpression)

参数： tExpression 指定日期时间表达式， HOUR( ) 函数将返回它的小时部分。

返回： 数值型，返回日期时间表达式 tExpression 的小时部分。

HOUR( ) 函数返回的数值是以 24 小时制为基础的，并且不受 SET HOURS 当前设置的影响。

例子：

```
? HOUR({1:00pm})                     && 显示 13
```

MINUTE( )

返回分钟

语法： MINUTE(tExpression)

参数： tExpression 指定日期时间表达式， HOUR( ) 函数将返回它的分钟部分。

返回： 数值型，返回日期时间表达式 tExpression 的分钟部分。

例子：

```
? MINUTE({1:50pm})                   && 显示 50
```

SEC( )

返回秒钟

语法： SEC(tExpression)

参数： tExpression 指定日期时间表达式， HOUR( ) 函数将返回它的秒钟部分。

返回： 数值型，返回日期时间表达式 tExpression 的秒钟部分。

例子：

```
? SEC({1:00:45pm})                   && 显示 45
```

CDOW( )

返回星期值

语法： CDOW(dExpression | tExpression)

参数： dExpression 指定日期， CDOW( ) 返回其星期值。

tExpression 指定日期时间，CDOW( )返回其星期值。

返回： 字符型，返回星期值的字符型表达式。

该函数输入一个日期型或者日期时间型表达式，给出该日期是星期几。

例子：

```
? CDOW(DATE())                    && 显示今天的星期，这里是“ Thursday ”
```

DOW( )

返回该日期是一周的第几天

语法： DOW(dExpression | tExpression [, nFirstDayOfWeek])

参数： dExpression 指定日期表达式。

tExpression 指定日期时间表达式。

nFirstDayOfWeek 指定一周的第一天。 nFirstDayOfWeek 可以是下列某个值：

表 4.3 nFirstDayOfWeek 取值说明

nFirstDayOfWeek	说 明
0	DOW( ) 使用当前在“星期开始于”列表框中选定的日期，该列表框出现在“选项”对话框中的“国际”选项卡上
1	星期日。当省略 nFirstDayOfWeek 时这是默认值。它是早期 FoxPro 版本使用的一周的第一天
2	星期一
3	星期二
4	星期三
5	星期四
6	星期五
7	星期六

例子：

```
? DOW(DATE())
```

CMONTH( )

返回月份名称

语法： CMONTH(dExpression | tExpression)

参数： dExpression 指定日期表达式，CMONTH( ) 返回其月份名称。

tExpression 指定日期时间表达式，CMONTH( ) 返回其月份名称。

返回： 字符型，返回给定日期 dExpression 或日期时间 tExpression 表达式的月份名称。

例子：

```
? CMONTH(DATE())                  && 显示“ March ”
```

```
? CMONTH({01/02/98})                    && 显示 " January "
```

4.4.3 转换函数

CTOD( )

把字符表达式转换成日期表达式

语法： CTOD(cExpression)

参数： cExpression 指定一个字符表达式， CTOD( )把它转换成日期型值。其求值结果必须是从 1/1/100 至 12/31/9999 范围内的一个有效日期。默认格式是 mm/dd/yy 。可用 SET DATE 和 SET CENTURY 来更改默认格式。如果输入日期时没指定是哪个世纪（如字符表达式 1/1/95 ），则假定为二十世纪。

返回： 日期型，返回字符表达式 cExpression 转换成的日期表达式。

例子：

```
? CTOD("03/04/98")                      && 显示日期型表达式 03/04/98
```

CTOT( )

把字符表达式转换成日期时间值

语法： CTOT(cCharacterExpression)

参数： cCharacterExpression 指定要返回日期时间值的字符表达式。根据 SET DATE 、 SET HOURS 和 SET MARK 的当前设置确定 cCharacterExpression 的正确格式。 cCharacterExpression 的求值结果必须是一个可识别的日期时间值，否则 VFP 将产生错误。

下面是有效的字符表达式的示例：

02/16/95 2:00:00pm

02/16/95 2:00pm

02/16/95 14:00

02/16/95

14:00

如果在 cCharacterExpression 中只指定了日期部分或时间部分， VFP 自动给 cCharacterExpression 加上默认时间，即午夜时间(12:00:00 A.M.)或默认日期 12/30/1899 。

可在 cCharacterExpression 中包含世纪。但只有在 SET CENTURY 是 ON 的时候返回世纪。如果不包含世纪，则指定为二十世纪。

返回： 日期时间型，返回从字符表达式 cCharacterExpression 转换成的日期时间型表达式。

例子：

```
? CTOT("03/04/98")                      && 显示日期时间型表达式 03/04/98 12:00:00AM
? CTOT("03/04/98 11:00PM")              && 显示日期时间型表达式 03/04/98 11:00:00PM
```

DMY( )

转换成“日-月-年”格式的字符表达式

语法： DMY(dExpression | tExpression)

参数： dExpression 指定日期型表达式，DMY( ) 函数从该表达式返回“日-月-年”格式的字符串。

tExpression 指定日期时间表达式，DMY( ) 函数从该表达式返回“日-月-年”格式的字符串。

返回： 字符型，返回转换成的字符表达式。

该函数从一个日期型或日期时间型表达式返回一个“日-月-年”格式的字符表达式（例如，31 May 1995）。月名不缩写。

如果 SET CENTURY 设置为 OFF，DMY( ) 函数以 dd Month yy 格式（例如，16 February 95，不带世纪）返回一个字符串。如果 SET CENTURY 设为 ON，格式为“日-月-年”（例如，16 February 1995，带世纪）。

例子：

```
? DMY(DATE())                    && 显示“ 05 March 98 ”
? DMY(DATETIME())                && 显示“ 05 March 98 ”
```

MDY( )

转换成“月-日-年”格式的字符表达式

语法： MDY(dExpression | tExpression)

参数： dExpression 指定要返回的日期表达式。表达式用“月-日-年”格式表示。

tExpression 指定要返回的日期时间表达式。表达式用“月-日-年”格式表示。

返回： 字符型，返回转换成的字符表达式。

该函数与函数 DMY( )相似，只是其返回值为“月-日-年”格式的日期或日期时间表达式，其中月份名不缩写。

例子：

```
? MDY(DATE())                    && 显示“ March 05, 98 ”
? MDY(DATETIME())                && 显示“ March 05, 98 ”
```

DTOC( )

把日期或日期时间表达式转换成返回字符型日期

语法： DTOC(dExpression | tExpression [, 1])

参数： dExpression 指定日期型内存变量、数组元素或字段，DTOC( ) 由此返回字符型日期。

tExpression 指定日期时间型内存变量、数组元素或字段，DTOC( ) 由此返回字符型日期。

1 以适合作为索引的格式返回日期。对于按时间顺序维护表的记录特别有用。

返回： 字符型，返回转换成的字符表达式。

其中，日期格式由 SET CENTURY 和 SET DATE 确定。

例子：

```
? DTOC(DATE())                    && 显示 " 03/05/98 "
? DTOC(DATE(), 1)                 && 显示 " 19980305 "
```

TTOC( )

把日期时间表达式中转换成字符型表达式

语法： TTOC(tExpression [, 1])

参数： tExpression 指定一个日期时间表达式， TTOC( ) 函数从该表达式中返回一个字符值。tExpression 必须是一个合法的日期时间值。如果 tExpression 中只包含时间，VFP 把默认的日期 " 12/30/1899 "添加到 tExpression 中；如果 tExpression 中只包含日期，VFP 将默认的午夜时间 " 12:00:00A.M." 添加到 tExpression 。

1 指定 TTOC( ) 函数按适合于进行索引的格式返回一个字符串。该字符串有 14 个字符，格式为 " yyyy:mm:dd:hh:mm:ss " ，此格式不受 SET CENTURY 或 SET SECONDS 中当前设置的影响。

返回： 字符型，返回转换成的字符表达式。

例子：

```
? TTOC(DATE())                    && 显示 " 03/05/98 12:00:00AM "
? TTOC(DATETIME())                && 显示 " 03/05/98 11:34:30AM "
? TTOC(DATETIME(), 1)             && 显示 " 19980305113454 "
```

DTOS( )

把日期或日期时间表达式转换成 yyyymmdd 格式的字符串日期。

语法： DTOS(dExpression | tExpression)

参数： dExpression 指定日期表达式， DTOS( ) 将其转换为八位字符串。

tExpression 指定日期时间表达式， DTOS( ) 将其转换为八位字符串。

返回： 字符型，返回转换成的字符表达式。

该函数的功能与包含参数 1 的 DTOC( ) 相同。

例子：

```
? DTOS(DATE())                    && 显示 " 19980305 "
```

DTOT( )

把日期型表达式转换成日期时间型

语法： DTOT(dDateExpression)

参数： dDateExpression 指定要返回日期时间型值的日期型表达式。

返回： 日期时间型，返回转换成的日期时间型表达式。

DTOT( ) 返回的日期时间型值的格式由 SET DATE 和 SET MARK 当前设置值决定。若未提供世纪值，则假定为二十世纪。

DTOT( ) 向日期中加上午夜 (12： 00： 00 A.M.) 的默认时间来生成有效的日期时间。

例子：

```
? DTOT(DATE())                    && 显示{03/05/98 12:00:00AM}
```

TTOD( )

把日期时间表达式转换成日期型。

语法： TTOD(tExpression)

参数： tExpression 指定一个日期时间表达式， TTOD( ) 函数从该表达式中返回一个日期。tExpression 必须是一个合法的日期时间值。如果 tExpression 只包含时间，VFP 将默认的日期 " 12/30/1899 " 添加到 tExpression 中，并返回这个默认的日期。

例子：

```
? TTOD(DATETIME())                && 显示{03/05/98}
```

4.5 其他常用函数

还有一些比较常用的函数，我们介绍如下：

MTON( )

把货币型表达式转换成数值型表达式

语法： MTON(mExpression)

参数： mExpression 指定一个货币型表达式， MTON( ) 函数返回它的值，mExpression 计算结果必须是有效的货币值，否则 VFP 产生错误。

返回： 数值型，返回转换成的数值型值。

注意：在数值型值前面加一个美元符号 ($) 的前缀，就可以创建一个货币型值。

例子：

```
? MTON($1000)                     && 显示 1000.0000
```

NTOM( )

把数值表达式转换成含有四位小数的货币值

语法： NTOM(nExpression)

参数： nExpression 指定一个数值表达式， NTOM( ) 返回其货币值。如果 nExpression 的小数位数大于四位，则将其四舍五入至四位；如果少于四位，则添加 0 补齐四位。

该函数的功能与 MTON( ) 正好相反。

例子：

```
? NTOM(100)                       && 显示 100.0000
? NTOM(MTON($100))                && 显示 100.0000
```

STR( )

返回与指定数值表达式对应的字符。

语法： STR(nExpression [, nLength [, nDecimalPlaces]])

参数： nExpression STR( ) 要计算的数值表达式。

nLength STR( ) 返回的字符串长度。该长度包括小数点所占的字符和小数点右边每个数字所占的字符。如果指定长度大于小数点左边数字位数， STR( ) 用前导空格填充返回的字符串；如果指定长度小于小数点左边的数字位数， STR( ) 返回一串星号，表示数值溢出。

nDecimalPlaces 由 STR( ) 返回的字符串中的小数位数。若要指定小数位数，必须同时包含 nLength 。如果指定的小数位数小于 nExpression 中的小数位数，则截断多余的数字。

例子：

```
? STR(100.123,7,3)          && 显示“ 100.123 ”
? STR(100.123,3)            && 显示“ 100 ”
? STR(100.123,2)            && 显示“**”
```

TYPE( )

返回表达式的数据类型。

语法： TYPE(cExpression)

参数： cExpression 指定备注型字段的名称或字符表达式， TYPE( ) 函数将对其中内容求值，并返回适当的数据类型。

返回： 字符型，其返回的字符代表不同的类型：

表 4.4 TYPE( )返回的类型

数据类型	返回的字符
字符型	C
数值型(或者整数、单精度浮点数和双精度浮点数)	N
货币型	Y
日期型	D
日期时间型	T
逻辑型	L
备注型	M
对象型	O
通用型	G
未定义的表达式类型	U

例子：

```
? TYPE("123")               && 显示N，表示为数值型
? TYPE("$123")              && 显示Y，表示为货币型
```

BETWEEN( )

判断一个表达式的值是否在另外两个相同数据类型的表达式的值之间。

语法： BETWEEN(eTestValue, eLowValue, eHighValue)

参数： eTestValue 指定 BETWEEN( ) 函数所测试的表达式。

eLowValue 指定 BETWEEN( ) 计算范围的下界。

eHighValue 指定 BETWEEN( ) 计算范围的上界。

返回： 逻辑型或 Null 值，当 eTestValue 大于等于 eLowerValue 而小于等于 eHighValue 时，BETWEEN( ) 返回逻辑值“真”(.T.)，否则返回逻辑值“假”(.F.)。如果 eLowerValue 或 eHighValue 为 Null 值，则返回 Null 值。

例子：

```
? BETWEEN("B","A","C")          && 显示.T.
? BETWEEN(20,30,40)              && 显示. F.
```

EMPTY( )

确定表达式是否为空值。

语法： EMPTY(eExpression)

参数： eExpression 指定 EMPTY( ) 函数作用的表达式。可包含字符、数值、日期或逻辑表达式，也可以是已打开表的备注字段或通用字段的名称。当表达式取下列值时，EMPTY( ) 函数返回“真”(.T.)。

表 4.5 EMPTY( )函数返回真的定义

表达式类型	取 值
字符型	空字符串、空格、制表符、回车、换行符或以上各字符的任意组合。
数值型	0
货币型	0
浮点型	0
整型	0
双精度型	0
日期型	空（例如 CTOD('')）
日期时间型	空（例如 CTOT('')）
逻辑型	“假”(.F.)
备注字段	空（没有内容）
通用字段	空（没有 OLE 对象）
图片	空（没有图片）

返回： 逻辑型，返回是否为空值。

例子：

```
CLOSE ALL
CREAT TABLE MyTab(Name C(10))
```

```
APPEND BLANK
? EMPTY(Name)                    && 显示为.T.
```

ISBLANK( )

判断表达式是否为空值

语法： ISBLANK(eExpression)

参数： eExpression 函数要判断的表达式。 eExpression 可以是表中的一个字段、一个内存变量或数组元素，也可以是一个表达式。

对一个字段来说，如果该字段包含下述值，则 ISBLANK( ) 函数将返回“真”(.T.)。

表 4.6 空值的定义

类 型	内 容
字符型	空字符串、空格或无值（新追加的空记录或用 BLANK 命令清除后的记录）
数值型	无值（新追加的空记录或用 BLANK 命令清除后的记录）
浮点型	无值（新追加的空记录或用 BLANK 命令清除后的记录）
日期型	空日期（{ / / }）或无值（新追加的空记录或用 BLANK 命令清除后的记录）
日期时间型	空日期时间（{ / / : : }）或无值（新追加的空记录或用 BLANK 命令清除后的记录）
逻辑型	无值（新追加的空记录或用 BLANK 命令清除后的记录）
备注型	空（无备注内容）
通用型	空（无 OLE 对象）
图片	空（无图片）

返回： 逻辑型。

APPEND BLANK 和 BLANK 命令可用来创建空记录。 BLANK 命令还可以用来清除一个记录内某些字段中的数据。用 ISBLANK( ) 函数则可以判断一个字段是否为空值。

注意：货币型、整型、双精度型表达式永远不可能为空值，因而对这些表达式 ISBLANK( ) 函数总是返回“假”(.F.)。

ISBLANK( ) 函数与 EMPTY( ) 和 ISNULL( ) 函数不同。例如，当字符表达式是 null 值、空格、Tab 字符、回车、换行符的某种组合时， EMPTY( ) 函数将返回“真”(.T.)；但只有在字符表达式中仅包含空字符串或空格时， ISBLANK( ) 函数才返回“真”(.T.)。

例子：

在下面的例子中，我们为表加入一条空记录，用该函数判断其是否为空：

```
CLOSE ALL
CREAT TABLE MyTab(Name C(10))
APPEND BLANK
? ISBLANK(Name)                  && 显示为.T.
```

ISNULL( )

判断一个表达式是否为 Null 值

语法： ISNULL(eExpression)

参数：　eExpression 指定要计算的表达式。

该函数判断一个表达式的计算结果是否为 null 值，是则返回“真”(.T.)；否则返回“假”(.F.)。

ISNULL( ) 函数可用于判断字段、内存变量或数组元素是否为 null 值，也可以判断表达式的计算结果是否为 null 值。

例子：

```
STORE Null TO Temp
? ISNULL(Temp)                    && 显示.T.
```

4.6 总　结

以上我们简单地介绍了一些常用函数的功能和用法，为了便于用户查找了和使用，我们将以上的函数总结一下。

表 4.7　字符处理函数

函　数	功　能
LTRIM( )	去掉字符表达式头的空格
RTRIM( )	去掉字符表达式尾的空格
ALLTRIM( )	去掉字符表达式头尾的空格
AT( ) | AT_C( )	查找匹配的字符串，大小写敏感
ATC( ) | ATCC( )	查找匹配的字符串，大小写不敏感
ATLINE( ) | ATCLINE( )	查找匹配的字符串的行号
OCCURS( )	计算一个字符表达式在另一个字符表达式中出现的次数
INLIST( )	判断一个表达式是否与一组表达式中的某一个相匹配
CHRTRAN( ) | CHRTRANC( )	查找并替换匹配的字符
LEFT( ) | LEFTC( )	从最左截取一段字符
RIGHT( ) | RIGHTC( )	从最右截取一段字符
SUBSTR( ) | SUBSTRC( )	任意截取一段字符
LEN( ) | LENC( )	计算一个字符表达式的长度
ISALPHA( )	判断是否为字母
ISDIGIT( )	判断是否为数字
ISLOWER( ) | ISUPPER( )	判断字符大小写
LOWER( ) | UPPER( )	改变字符大小写
ASC( )	返回字符的ANSI值
CHR( )	返回ANSI数值代码对应的字符
REPLICATE( )	重复指定次数后，形成新的字符型表达式
SPACE( )	由指定个数的空格组成字符串

表 4.8 数值处理函数

函 数	功 能
SIN( )	正弦函数
COS( )	余弦函数
TAN( )	正切函数
ASIN( )	反正弦函数
ACOS( )	反余弦函数
ATAN( )	反正切函数
EXP( )	计算e^x 的值
SQRT( )	平方根函数
LOG( )	自然对数（底数为 e 的对数）
LOG10( )	常用对数（以 10 为底的对数）
CEILING( )	向上取整函数
FLOOR( )	向下截断
INT( )	取整函数
ROUND( )	四舍五入到指定的小数位
ABS( )	绝对值函数
MAX( )	求几个表达式中的最大值
MIN( )	求几个表达式中的最小值
MOD( )	求两个数相除的余数
PI( )	返回常数 π
RAND( )	随即数函数
SIGN( )	符号函数
VAL( )	由数字组成的字符表达式返回数字值

表 4.9 日期时间处理函数

函 数	功 能
DATE( )	返回系统日期
DATETIME( )	返回系统日期时间
YEAR( )	返回年份
MONTH( )	返回月份
WEEK( )	返回星期
DAY( )	返回日期
HOUR( )	返回小时
MINUTE( )	返回分钟
SEC( )	返回秒钟
CDOW( )	返回星期值
DOW( )	返回该日期是一周的第几天

（续表）

函　数	功　能
CMONTH( )	返回月份名称
CTOD( )	把字符表达式转换成日期表达式
CTOT( )	把字符表达式转换成日期时间值
DMY( )	转换成“日-月-年”格式的字符表达式
MDY( )	转换成“月-日-年”格式的字符表达式
DTOC( )	把日期或日期时间表达式转换成返回字符型日期
TTOC( )	把日期时间表达式中转换成字符型表达式
DTOS( )	把日期或日期时间表达式转换成yyyymmdd格式的字符串日期
DTOT( )	把日期型表达式转换成日期时间型
TTOD( )	把日期时间表达式转换成日期型

表 4.10　其他常用函数

函　数	功　能
MTON( )	把货币型表达式转换成数值型表达式
NTOM( )	把数值表达式转换成含有四位小数的货币值
STR( )	返回与指定数值表达式对应的字符
TYPE( )	返表达式的数据类型
BETWEEN( )	判断一个表达式的值是否在另外两个相同数据类型的表达式的值之间
ISBLANK( )	判断表达式是否为空值
ISNULL( )	判断一个表达式是否为 Null 值

第五章　Visual FoxPro 5.0 的调试器

程序的调试是指查找并确定计算机程序中发生错误或问题的原因。任何程序在编制初期，无论编制者多么精明，都难免出现各式各样的错误、失误或者疏忽。错误的种类很多，不同程序员的习惯错误又有不同，通常错误可分为以下几种：

（1）编码错误：编码错误包括语法错误、逻辑错误和运行环境错误。语法错误是最低级的因而也是最易发现的错误，同时也是最常见的错误。对代码进行编译可迅速查知并及时纠正语法错误，比如某些拼写错误等。

（2）过程错误：过程错误是由于程序或过程的代码过长，程序员在把握所有详细细节上的疏忽而产生的矛盾和错误。

（3）参数错误：过程调用时传给过程的参数数目、次序不正确或者变量使用发生错误导致程序错误。

（4）结构错误：由于程序中有多个 RETURN 的多点出口语句，可能引发潜在的错误。

（5）文件错误：在文件的处理过程中如果发生意外关机没有正确退出程序，因而无法关闭表格等会导致文件出错。

针对不同类型的错误，纠正和排除错误的方法有很多。首先，程序员应当养成良好的编程习惯。在可能出错处使用错误处理例程，针对自己常有的错误对程序重点检查。另外一个常用的重要手段是对程序段或过程进行调试。本章的目的正在于此。调试程序的重要工具是调试器。

5.1　FoxPro 最新版本的调试器

调试器是 Visual FoxPro 5.0 区别于其他各 FoxPro 版本的新功能，它对以前的调试工具进行了补充和集成。在命令窗口中输入 DEBUG 命令可以打开调试器（如图 5.1）。

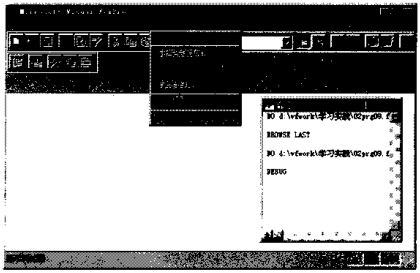


图 5.1　打开调试器

也可以从主菜单中选择菜单项“工具”——>“调试器”。调试器主要包括菜单条、工具栏和窗口显示区。菜单条提供调试器的各菜单选项；工具栏中包含一些常用的工具；窗口显示区可根据需要选择显示“跟踪”、“监视”、“局部”、“调用堆栈”和“输出”等窗口中的一个或者几个窗口。

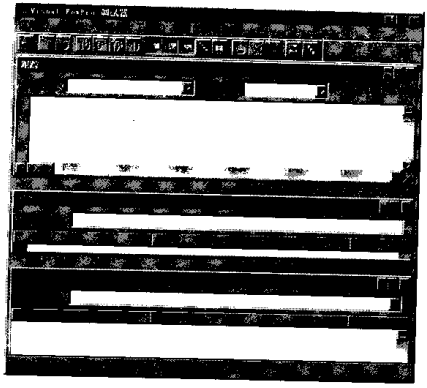


图 5.2　调试器

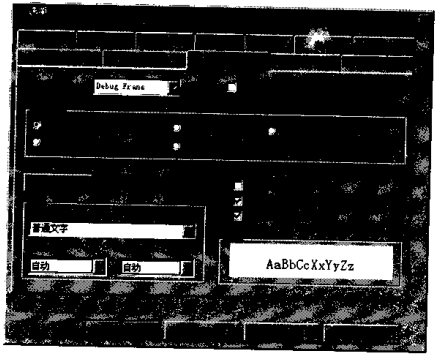


图 5.3　调试器选项

使用“调试”选项卡可定制 Visual FoxPro 5.0 的调试器窗口，其方法是：

（1）选择菜单项“工具”——>“选项”，即可弹出选项卡窗口；

（2）用鼠标单击选中“调试”选项卡；

（3）定制调试器窗口。

可通过设置选项来定制 Visual FoxPro 调试器窗口的特性有：

（1）字体及颜色；

（2）是否跟踪 Timer 事件代码；

（3）是否显示行号；

（4）是否把“调试输出”窗口的内容输出到一个文件中。

当选择对话框中每个选项卡上的“设置为默认值”按钮时，Visual FoxPro 在注册表（系统 Windows 注册数据库）中保存选项设置。

下面来说明“调试”选项卡的各选项以及如何定制。

1. 环境

通过该选项可指定调试环境，即指定调试工作基于什么窗口进行，有两种可供选择的调试环境：

（1）Debug Frame。 所有调试器的窗口出现在一个大页框中。这个页框在 Visual FoxPro 主窗口之外，有自己的菜单和工具栏。这样，可以使调试器更少地介入应用程序的运行环境。如果选择了该选项，可以通过选择“工具”菜单上的“调试器”来打开调试器。上面图中所示的调试器窗口就是采用 Debug Frame 环境的例子。

（2）FoxPro Frame。 调试器窗口出现在 Visual FoxPro 主窗口中。如果只需要单个的窗口打开（如“监视”窗口），FoxPro 环境最有用。如果选择了该选项，Visual FoxPro 用打开单个窗口的命令代替“工具”菜单上的“调试器”命令，在“工具”菜单中不出现“调试器”选项，同时单个调试窗口的工具栏也贴在 Visual FoxPro 主窗口中，如图 5.4。值得注意的是调试器打开时“环境”选项为灰色，此时不能更改该选项。

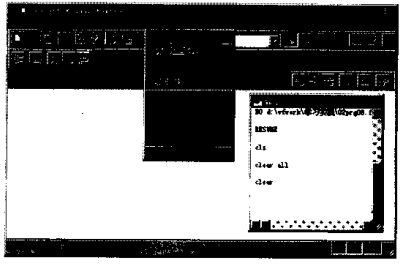


图 5.4 试窗口的工具栏贴在 Visual FoxPro 主窗口中

2. 显示计时器事件

在达到间隔值时，如果希望计时器控件的 Timer 事件显示在“跟踪”窗口中，选定该选项。

如果清除了这个选项，Timer 事件仍然发生，但不显示；就好象已经为那个事件指定了“跳出”一样对待。

3.指定窗口

选定一个选项，指定是为哪一个调试器窗口设置选项。在这里选定的窗口会影响下列可用的选项：

（1）调用堆栈。其中可指定的附加选项决定了是否在“调用堆栈”窗口中显示调用顺序、当前行指示器或堆栈调用指示器。“显示堆栈调用顺序”复选框选项确定是否在“调用堆栈”窗口列出的每一个程序旁边显示一个编号，其中最大的编号指示当前执行的程序；“显示当前行的指示器”复选框选项确定是否将当前行指示符显示在“调用堆栈”窗口中；“显示堆栈调用指示器”复选框选项确定是否在“调用堆栈”窗口中显示一个箭头来指示在“跟踪”窗口中显示的过程。如果当前行与调用堆栈过程相同，Visual FoxPro 只显示当前行指示符。

（2）局部。该窗口没有可用的附加选项。

（3）输出。显示“记录调试输出”区域，它包含“记录调试输出”选项、输出的日志文件名的输入框和“追加”和“改写”选项。在选中“记录调试输出”选项时才能设置输出的日志文件名以及“追加”和“改写”选项。“记录调试输出”选项把写入“调试输出”窗口中的值复制到文本文件中。如果选择记录输出值，需要指定一个日志文件，日志文件的默认扩展名是 .LOG 。只有当“调试输出”窗口显示时，Visual FoxPro 才会把输出记录到一个文件中。可以在程序中使用命令 DEBUGOUT 或 SET PRINTER TO DEBUG 显示“调试输出”窗口中的信息（并把它发送到日志文件中）。当退出 Visual FoxPro 时，该选项被清除。这样下次启动 Visual FoxPro 时，您就不会无意中改写日志文件。“追加”复选框可以指定在现有文件当前内容的后面写入调试输出，保留初始的内容；而“改写”复选框指定调试输出改写（或覆盖）指定文件的内容。

（4）跟踪。在“跟踪”选项下显示该窗口的下列三个选项：

“显示行号”选项决定是否在“跟踪”窗口中的代码行左侧显示行号。“跟踪断点之间的部分”选项以调节速度执行断点之间的代码。调节速度意味着在每行程序的执行之间有数秒的停顿。如果清除了该选项，则以正常速度执行断点之间的代码。“运行每行时暂停”选项指定执行每行代码之间停顿的秒数。该选项允许减慢程序的执行，这样可以看到每行代码，而不需要单独找每一行。该选项也可以在调试器窗口中设置。

（5）监视。该窗口没有可用的附加选项。

上面是针对不同窗口所独有的一些选项。还有一些选项是所有“调试器”窗口都可用的选项，包括设置窗口的字体、颜色等内容。

4.字体

为指定的调试器窗口选择一个字体和样式。例如，若要设置调试器“跟踪”窗口的字

体，在“指定窗口”下选择“跟踪”，然后选定“字体”来选择字体、大小及字体样式。

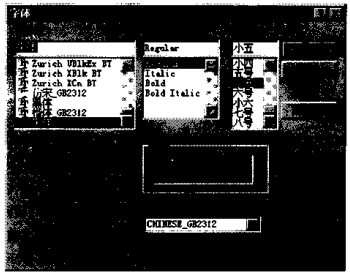


图 5.5 选择字体

5.颜色

（1）区域。选择准备为其指定颜色的文本元素。例如，若要指定更改的值所用文本的颜色，选定“更改的值”。“区域”列表中的选择取决于“指定窗口”下指定的窗口。

（2）前景。为选定范围的文本选择颜色。要使用 Windows 控制面板中建立的默认颜色，选择“自动”。

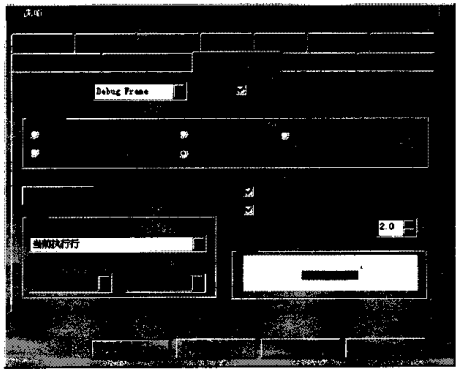


图 5.6 设置环境

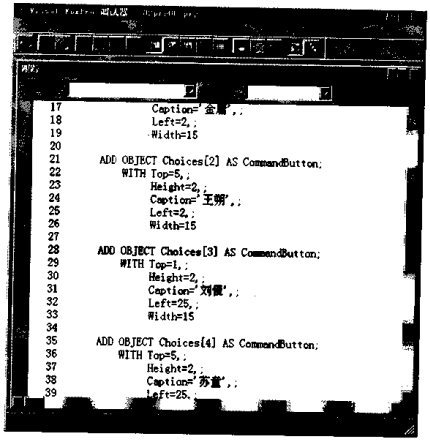


图 5.7 跟踪窗口

（3）背景。为选定范围选择一个背景颜色。例如，如果要用绿底红字显示当前执行行，可以在“前景”框中选择红色，然后在“背景”框中选择绿色。如果要使用 Windows 控制面板中建立的默认颜色，选择“自动”。

一个设置字体为宋体，并要求显示行号的自动上色的跟踪窗口如图 5.7。利用 Visual FoxPro 5.0 的强大功能，用户完全可以根据自己的爱好设置这些环境。

5.2 调试器菜单

调试器窗口的菜单条包含有文件、编辑、调试、工具、窗口和帮助等六个菜单，菜单中含有相应的菜单命令。但如果在“环境”选项设置为 FoxPro Frame 时，Visual FoxPro 系统把这些命令附加到 Visual FoxPro 主菜单中去，但其功能与用法是一样的。

5.2.1 文件菜单

文件菜单中包含了操作调试文件的命令项和退出调试器的选项。利用这些选项，可以打开程序，进行调试设置，也可以将调试输出保存到文件中。

1.打开

显示“打开”对话框，指定一个显示在“跟踪”窗口中的源程序。打开程序并不运行该程序。在对话框中显示的文件类型有：

（1）PRG 文件，可用于调试的源程序文件。

（2）FXP 文件，编译后的程序文件。该文件不能单独用来调试，必须有同名的 PRG 文件存在时才可用。

（3）MPR 文件，生成的菜单程序文件。

（4）APP 文件，生成的应用程序文件。选定时弹出对话框，可从中选择一个程序用来调试。

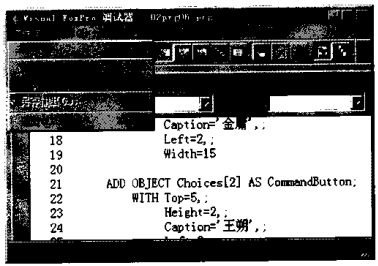


图 5.8 文件菜单

2.加载配置

从配置文件（.DBG 文件）中加载要追踪的断点、监视及事件。选中时显示“打开”对话框，在对话框中您可以指定一个文件。

3.保存配置

将当前追踪的断点、监视及事件保存在配置文件（.DBG 文件）中，以便以后恢复它们。选中时显示“打开”对话框，在对话框中可以指定一个文件。

4.另存输出

将“调试输出”窗口中显示的文本写入“另存为”对话框中指定的文件。

5.退出

关闭“调试器”窗口，返回 Visual FoxPro 主窗口。选中该项时系统退出并关闭调试器窗口。

5.2.2 编辑菜单

编辑菜单的选项帮助完成一些与编辑相关的功能。

1.剪切

把选中的文本删除并放到剪贴板上。只有对于可编辑文本的输入框，该功能才有效。

2.复制

复制选中文本并放在剪贴板上。可以利用该选项可将跟踪窗口中选中的内容复制到剪

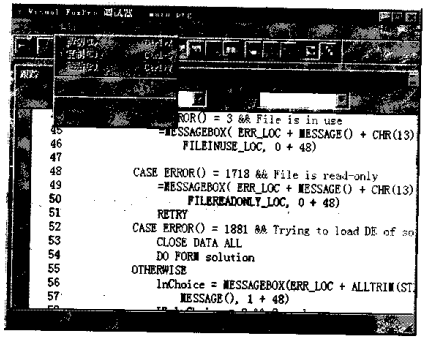


图 5.9 编辑菜单

切板中。比如，要想将某一段程序在命令窗口运行，可采用下述方法：

（1）在需要的程序行开始处按下鼠标左键，然后拖动鼠标至到选中所需程序段；

（2）选择菜单项“编辑”——>“复制”，将所选部分复制到剪切板；

（3）在命令窗口中单击鼠标，使光标位于命令窗口的末尾；

（4）在 Visual FoxPro 主窗口中选择菜单项“编辑”——>“粘贴”。

这样就可以在命令窗口执行程序段了。但是，并非所有程序段都能执行。对于有些程序模块，可以先拷贝到一个新文件中，然后在命令窗口中使用 DO 命令执行。

熟练使用键盘进行剪切复制和粘贴工作也是可取的。因为这些操作在许多应用软件中是几乎相同的。下面帮助不太熟练的使用者回顾一下。

（1）将光标移到所需程序段的开始处，在按下 Shift 键的同时按任意箭头键，此时可以看到一部分程序段处于高亮选中状态；

（2）通过按向上或者向下箭头键可以选中或者放弃选中一行；同理，通过按向左或者向右箭头键可以选中或者放弃选中一行的一个字符；

（3）按 Ctrl+c 可将选中部分拷贝到剪切板；在许多应用软件编辑时，多数情况下按 Ctrl+x 可剪切，按 Ctrl+v 可粘贴。但由于跟踪区不能修改，剪切和粘贴功能不可用。

（4）在命令窗口获得焦点时，按 Ctrl+v 可将剪切板的内容复制到命令窗口。

3.粘贴

把剪下或复制的文本放至插入点。

4.全部选定

单击该菜单选项时，选中“调试器”活动窗口中的所有文本，以便进行上述操作。

5.查找

显示“查找”对话框，用来查找文本。如果调试的代码较多，在查看和定位时采用滚动条会很麻烦，此时可用编辑菜单的“查找”菜单命令。选中该命令后，屏幕上弹出“查找”对话框。“查找”对话框包括四个区：

（1）查找内容区。

查找内容的输入框为一组合框。该组合框的下拉列表框中存入了查找的历史记录。用户可从组合框的编辑框部分输入需要查找的字符串，也可以从列表框中选择曾经查找过的字符串。

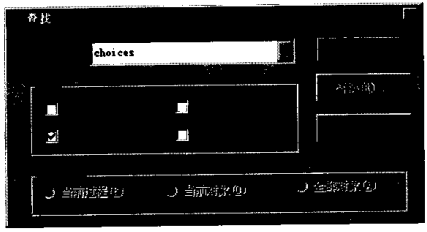


图 5.10 查找对话框

（2）选项区。

在选项区中可设置四个选项。用户可决定查找内容是否区分大小写，是否要求全字匹配。如果选取了“回绕”选项，当查找到文件的末尾或者开始处能根据是否选择“向后搜索”选项进行向前或者向后继续搜索。向后搜索选项指定按行号递减的方向进行搜索，不选中时则按行号递增的方向搜索。

（3）作用范围区。

在作用范围选项区可设置在什么范围查找字符串。默认时在当前过程中查找。

（4）按钮区。

按钮区含有三个按钮：“查找下一个”按钮用来继续查找；“替换”按钮此时不可用，因为跟踪窗口此时不可修改；单击“取消”按钮可退出查找窗口。

6.再次查找

“再次查找”菜单命令以最近一次设置的功能选项和范围查找最近一次查找过的字符串。它同样依设置情况决定是否区分大小写和进行全字匹配、是否回绕和查找的方向。

5.2.3　调试菜单

调试菜单中集中了调试操作的主要功能选项。运用这些选项可以运行或者退出调试，选择调试方式，设置执行速度等。

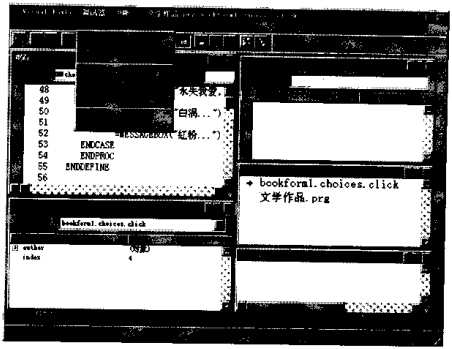


图 5.11　调试菜单

1.执行（或继续执行）

开始执行在“跟踪”窗口中打开的程序。若“跟踪”窗口没有打开的程序，显示“执行”对话框，让您能够指定要跟踪的程序（或表单）。指定的程序（或表单）与在可执行代码第一行挂起的执行程序一起执行。当把一个程序调入跟踪窗口后，可在程序的适当位置设置断点然后执行程序，在执行停顿后对程序加以检查。执行程序挂起时，“继续执行”命令有效。该命令在代码当前行继续执行“跟踪”窗口中的程序。读者可尝试调试下面一段程序。

```
*************************************************************
* 利用对象数组引用对象
*************************************************************
author=CREATEOBJECT('BookForm')
author.Show(1)
*************************************************************
DEFINE CLASS BookForm AS FORM
DIMENSION CHOICES[4]
   Caption="我喜欢的文学作品"
   ScaleMode=0
```

```
        Width =50
        Height =8
        PARAMETER INDEX
        DO CASE
           CASE INDEX=1
              =MESSAGEBOX("笑傲江湖，鹿鼎记...")
           CASE INDEX=2
              =MESSAGEBOX("永失我爱，过把瘾...")
           CASE INDEX=3
              =MESSAGEBOX("白涡...")
           CASE INDEX=4
     ****在下面一行设置断点************
●    =MESSAGEBOX("红粉...")
        ENDCASE
        ENDPROC
     ENDDEFINE
```

上述程序段的详细说明将在其他章节中介绍，这里仅以此为例说明调试器的有关操作方法。当程序运行到有断点的一行时，程序停止运行，此时菜单的第一条菜单项变成“继续执行”。按下述步骤可以体会程序的跟踪调试情况。

（1）将上述程序段在 Visual FoxPro 中输入并保存为程序“文学作品.prg”；

（2）在调试器窗口中选择菜单“文件”——>“打开”，打开程序“文学作品.prg”；

（3）在程序的适当地方设置断点；比如：

● =MESSAGEBOX("红粉...")

（4）选择菜单“调试”——>“执行”，此时屏幕上弹出窗口如图 5.12。

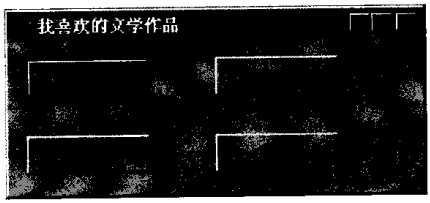


图 5.12 执行程序

（5）此时若单击左上角的按钮，将弹出消息框显示“笑傲江湖，鹿鼎记...”，单击“确定”按钮，运行光标回到程序开始并重新执行。

图 5.13　消息框显示“笑傲江湖，鹿鼎记...”

注意到我们设置的断点是在如下位置：

```
CASE INDEX=4
****在下面一行设置断点**************************************
```

- =MESSAGEBOX("红粉...")

因此程序没有执行到有断点处而停止下来。

（6）在 Visual FoxPro 主窗口中显示的“我喜欢的文学作品”窗口中用鼠标单击右下角的按钮。此时，程序在断点处暂停。菜单命令“执行”变成“继续执行”。

2.取消

关闭和终止“调试”窗口中的执行程序或表单。选择取消菜单命令可以终止对当前跟踪窗口的程序调试。但此时系统不退出调试器。选择打开文件可继续对加载文件调试。

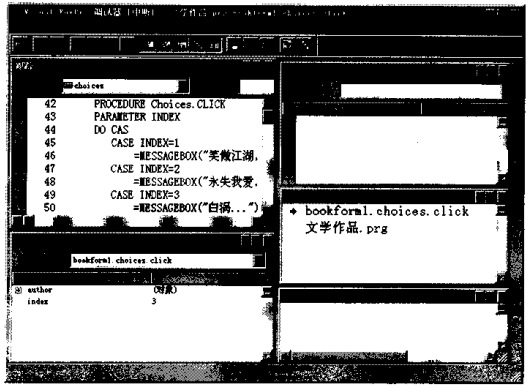


图 5.14　定位修改

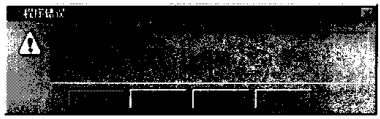


图 5.15 程序错误信息

图 5.16 消息框

3.定位修改

在 Visual FoxPro 5.0 版本的调试器中是不能对所调试的程序进行修改的，要修改程序可使用定位修改功能。执行程序挂起时，该命令有效。如果您正在跟踪程序，“固定”命令会提示您终止程序，然后在编辑窗口内打开它，将光标放在“跟踪”窗口内同样的位置上。如果您正在表单中跟踪代码，“固定”命令会使您终止执行，并且从内存中清除该表单对象，然后在“表单设计器”内打开编辑窗口，将光标放在“跟踪”窗口内同样的位置上。比如说，若把第 44 行的 CASE 误写为 CAS，当程序执行到第 44 行时，屏幕上弹出出错信息对话框。若选择“取消”按钮，则直接定位到该行并高亮显示。如果希望先到调试器中查看相关信息再修改错误，可选择“挂起”。此时调试器中的执行光标指向出错处。若选择菜单“调试”——>“定位修改”，屏幕上弹出消息框，从消息框中选择“取消程序”则可回到 Visual FoxPro 主菜单中修改程序。图 5.17 说明定位修改操作使程序的错误行得以高亮显示。当程序代码较多时，从程序中查找对应的出错程序段十分烦琐，该功能非常方便和迅速的找到错误，减少了麻烦。

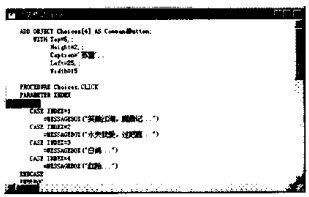


图 5.17 在程序中定位

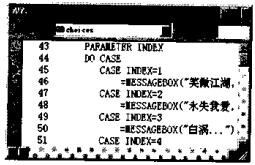


图 5.18 断点跟踪

4.跳出

继续执行一个过程中的代码，但并不是逐行地单步执行。在调用程序中，当走到过程调用的下边一行代码时，程序执行重新被挂起。当程序因为断点而停止运行时，用“跳出”命令可使程序继续运行，直到一个遇到新断点或者需要处理其他事件。

5.单步

逐行执行代码。如果下一行代码调用了函数、方法程序或者过程，那么该函数、方法程序或过程在后台执行。当程序因为断点而停止运行时，用“单步”命令可使程序继续运行，每一次命令只能使程序运行一步。从执行光标的停顿处不难看出“跳出”与“单步”之间的不同之处。也就是说，如果选择的按钮为“苏童”使得 INDEX = 4，而断点设在第 45 和 49 行处，在光标停在 45 行时如果选择“单步”命令，则执行光标停顿在第 47 行，如图 5.19 所示。但如果选择了“跳出”命令，则执行光标停顿在第 49 行，如图 5.20 所示。

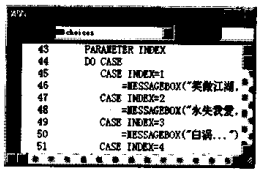


图 5.19 单步命令

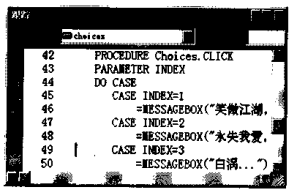


图 5.20 跳出

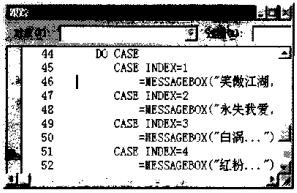


图 5.21 运行到光标处

6.单步跟踪

逐行执行代码。使用“单步跟踪”可以跟踪程序执行的详细过程。

7.运行到光标处

执行从当前行指示器到光标所在行之间的代码。在您要挂起执行的代码上单击，从而把光标移到这一行上。选择该项之前应当把光标置于适当的位置，选择“运行到光标处”命令可使程序执行到光标处暂停。比如，当程序执行到在第 44 行设置的断点时，如果选择的按钮为“苏童”使得 INDEX = 4，当光标在第 49 行处并选择了“运行到光标处”命令，则当程序执行到 49 行时暂停。程序不能停止在执行不到的语句行。这就是说，如果选择的按钮为“苏童”使得 INDEX = 4，将光标置于 46 行处并不能使程序在该处停止。实际上，程序将一直执行到满足 INDEX = 4 从而在屏幕上弹出消息框。

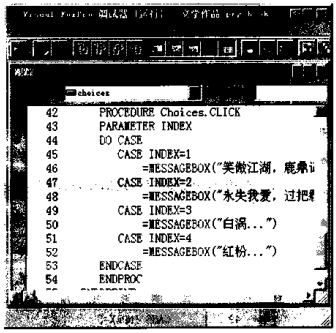


图 5.22 程序不能停止在执行不到的语句行

8.调速

打开“调整执行速度”对话框，您可以指定代码行之间的执行延迟时间(以秒计)。调试器调试菜单的调速选项控制了每个被执行程序之间的暂停时间长度。默认值 0 表示没有延迟，但是这并非指跟踪窗口执行程序的速度和平常程序执行的速度相同。实际上这也是一个大大降低了的速度，这是因为显卡驱动程序比 Visual FoxPro 平常执行代码的速度。通常可将各命令之间的延迟增加为 0~5 秒。另外，在追踪运行时，也可通过按 Ctrl 键、 Shift 键或者一个鼠标按钮将执行速度减半。当选择调速命令项时，屏幕上弹出“调整运行速度”窗口，从中可输入减速值或者用鼠标单击向上或者向下按钮设置适当的值，每次单击按钮使延迟时间值改变 0.1 秒。修改系统内存变量_THROTTLE 的值也可以设置减速值，比如：

_Throttle = 1.0

图 5.23　设置跟踪的延迟时间

9.设置下一条语句

将当前行指示器放在光标所在的代码行上，然后当单步执行或恢复执行时，即执行该行。将该命令与“单步”命令联用可以调试跟踪到特定的程序段中去，但用其他方法可能不会进入到该程序段。这就是说，如果选择的按钮为“苏童”使得 INDEX = 4，可以将光标置于 48 行，然后选择“设置下一条语句”菜单项，可以观察到，程序执行到 48 行处暂停下来。此时选择“单步”或者“继续执行”，将执行语句：

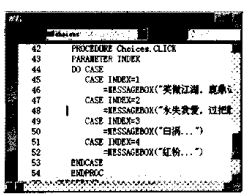


图 5.24　设置下一条语句

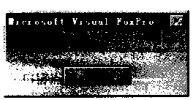


图 5.25　消息框

=MESSAGEBOX("永失我爱，过把瘾...")

从而显示如图 5.25 消息框。

5.2.4 工具菜单

工具菜单中包含三个菜单选项：断点、事件跟踪和编辑日志。

1. 断点

打开“断点”对话框，可以添加、删除、启用或废止断点。选择该项后屏幕上出现“断点”窗口，如图 5.26 所示，在该窗口中可以设置：

（1）断点类型，包括“在定位处中断”、“如果表达式值为真则在定位处中断”、“当表达式值为真时中断”以及“当表达式值改变时中断”。

（2）选择定位。

（3）设置断点的文件。

（4）运行次数。

（5）表达式。

（6）断点。

下面示例性的设置断点来具体加以说明。

例 1. 在定位处中断

类型：在定位处中断

定位： bookform::choices.click, 7

文件： d:\vfwork\学习实践\文学作品.prg

运行次数： 5

意义：在名字为“ d:\vfwork\学习实践\文学作品.prg ”程序的 bookform::choices.click 过程第 7 行设置断点，要求第五次执行该语句时在该处中断。

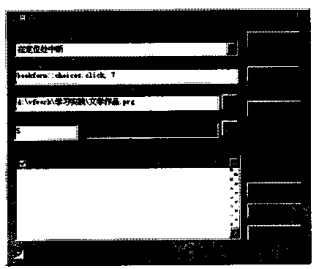


图 5.26 断点窗口

实施方法：

（1）在“跟踪”窗口中，找到想要设置断点的那一行；

（2）将光标放置在该代码行上；

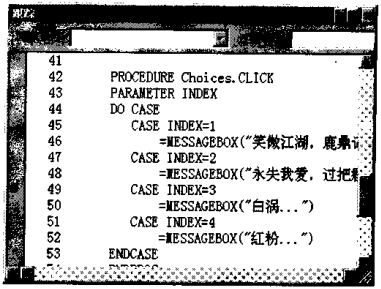


图 5.27 设置断点

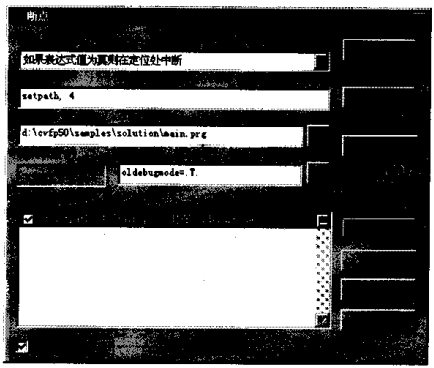


图 5.28 如果表达式值为真则在定位处中断

（3）按下 F9 键或者单击“调试器”工具栏上的“切换断点”按钮或者双击该行代码左边的灰色区域，该代码行左边的灰色区域中会显示一个实心红点，这表明该行已经设置了一个断点。如果正在调试对象，那么通过从对象列表中选择该对象，从过程列表中选择所需的方法程序或事件，就可以在“跟踪”窗口中找到特定的代码行。

（4）在“断点”窗口中对断点进行进一步设置。

断点设置结果显示如图 5.27 所示。从图上可以看出，中断点设在了第 49 行。这是为什么呢？原来定位的数字 7 是相对值，它是相对于过程 Choices.CLICK 的第一行而言的。也就是说第 43 行定位值为 1，依此类推。

运行次数为 5 表示在程序最初的 4 次执行该语句并不能中断，而是在第 5 次运行到该语句时才中断。实际执行程序有助于对这点的理解。这是非常有用的，因为有时我们需要检查程序在运行了一定次数之后的情况。

例 2. 如果表达式值为真则在定位处中断

类型：如果表达式值为真则在定位处中断

定位： setpath, 4

文件： d:\cvfp50\samples\solution\main.prg

表达式： oldebugmode=.T.

意义：当执行 main 程序到 oldebugmode=.T.时，在定位处中断。

实施方法：

（1）在“调试器”窗口中，从“工具”菜单上选择“断点”，打开“断点”对话框；

（2）从“类型”列表中，选择“如果表达式值为真则在定位处中断”；

（3）在“位置”框中，输入适当的位置；

（4）在“表达式”对话框中，输入相应的表达式；

（5）选择“添加”，将该断点添加到“断点”列表中；

（6）选择“确定”。

有时在“跟踪”窗口找到某个代码行，设置一个断点，然后在“断点”对话框中编辑该断点会更容易一些。为此，可将断点设置的“类型”从“在定位处中断”改成“如果表达式值为真则在定位处中断”，然后添加该表达式。

例 3. 当表达式值为真时中断

类型：当表达式值为真时中断

表达式： nreturnvalue=6

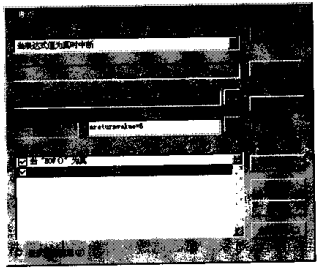


图 5.29 当表达式值为真时中断

意义：如果一个信息框的返回值存储在 nReturnValue 中，当用户在该信息框上选择“确定”的时候，将程序停止。

实施方法：

（1）在“调试器”窗口中，从“工具”菜单上选择“断点”，打开“断点”对话框；

（2）从“类型”列表中，选择“当表达式值为真时中断”；

（3）在“表达式”对话框中，输入相应的表达式；

（4）选择“添加”，将断点添加到“断点”列表中。

例 4. 当表达式值改变时中断

类型：当表达式值改变时中断

表达式：PROGRAM()

意义：在任意一个新的程序、过程、方法程序或事件的第一行上，将程序停止。

实施方法：

（1）在“调试器”窗口中，从“工具”菜单上选择“断点”，打开“断点”对话框；

（2）从“类型”列表中，选择“当表达式值改变时中断”；

（3）在“表达式”对话框中，输入相应的表达式。

另外，如果要了解何时一个变量或者属性的值发生了变化，或者想知道何时运行条件改变了，那么可以对一个表达式设置断点。比如，如果希望当该属性的值由于用户交互或程序运行而发生了改变时将程序停止，可将表达式设置为：

MyForm.MyText.Value

上述是一些设置断点或中断的范例和方法，有时常常消除断点。在“断点”对话框中，不用删除断点，就能使断点无效。在“跟踪”窗口中，可以删除“在当前位置停止”类型的断点。若要将断点从某个代码行中删除在“跟踪”窗口中，找到该断点，然后进行如下操作：将光标置于该行代码处，然后从“调试器”工具栏上选择“切换断点”按钮，或者双击该代码行左边的灰色区域，或者按回车键或空格键。

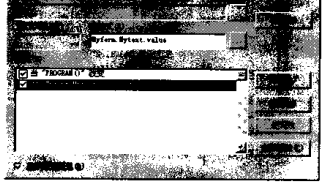


图 5.30　当表达式值改变时中断

2.事件跟踪

查看与其他事件有关的每个事件的发生时间。使用该对话框可以确定加入代码的最有效位置。在“调试器”窗口的“工具”菜单内选择“事件跟踪”，将出现此对话框。对话框可设置如下选项：

（1）开启事件跟踪。

切换事件跟踪的状态。启用事件跟踪时，每一次“事件跟踪”列表中的系统事件发生时，事件的名字都会显示在“调试输出”窗口中或者写入文件。

（2）可用的事件。

显示未选为跟踪的可用事件。若没有专门处理鼠标移动的操作，请将“MouseMove”事件从“跟踪事件”列表中删除。因为“MouseMove”事件通常会频繁发生。

（3）跟踪事件。

显示要跟踪的所有事件。

（4）输出到。

该控件组可设置以下几项：

调试输出窗口，用来切换跟踪事件在“调试输出”窗口中的显示状态。如果“调试输出”窗口未打开，即使已选择了“调试输出窗口”复选框，也不会列出事件。

文件，用来指定用来写入跟踪事件的文件。单击“文件”右侧的对话按钮，弹出“打开”对话框，可以从中选择文件。

追加，用来在指定文件的当前内容之后写入跟踪事件，保存初始内容。

改写，用来指定跟踪事件替换指定文件的内容。

激活事件跟踪后，每当“跟踪事件”列表中的一个系统事件发生时，该事件名字就会显示在“调试输出”窗口中，或者写到一个文件里。如果选择将事件显示在“调试输出”窗口中，仍可将它们保存在文件中。如果没有打开“调试输出”窗口，那么尽管已经将调试输出窗口设置好了，事件也不会列出来。

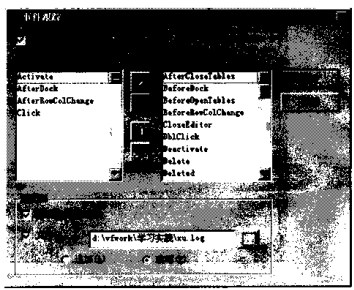


图 5.31 事件跟踪设置

3.编辑日志

打开“编辑日志”对话框。可以使用覆盖范围记录，也可为其指定一个文件。在此可以指定记录程序执行的选项。使用这个对话框检索信息，如，每行代码执行了多少次，在测试时哪些行的代码未执行，以及每行代码的执行时间等等。从“调试器”窗口的“工具”菜单中选择“编辑日志”，将出现该对话框。对话框具有以下内容：

（1）文件。指定写入覆盖范围记录的文件。对话按钮显示“打开”对话框，可以利用这个对话框，选择写入覆盖范围记录的文件。

（2）追加。指定在某个现存文件的当前内容之后写入覆盖范围记录，保存初始的内容。

（3）改写。用该覆盖范围记录代替指定文件的内容。

（4）生成。调用系统变量 _COVERAGE 指定的程序，以一种可读性更强的方式提供原始范围信息。

图 5.32 指定写入覆盖范围记录的文件

使用 SET COVERAGE TO 命令，也能够通过程序将代码覆盖范围功能打开或关闭。例如，在应用程序中，只要在想要研究的那部分代码前面包含如下的命令：

SET COVERAGE TO mylog.log

在这些要记录覆盖范围信息的代码后面，使用如下命令：

SET COVERAGE TO

可将代码覆盖范围功能关闭。

当为代码覆盖范围信息指定了一个文件后，可以切换到 Visual FoxPro 主窗口，然后运行程序、表单或者应用程序。对每一行执行过的代码，日志文件中记录下列信息：

（1）以秒为单位，计算该行代码运行的时间。

（2）该代码行所属的类（如果有的话）。

（3）该代码行所在的方法程序或事件。

（4）代码的行号。

（5）代码所在的文件。

要将信息从日志文件中提取，最简单方法是，将该文件转换为表，然后即可对该表设置筛选条件，运行查询和报表，执行命令，或者使用其他方法操作表。

例：下面的程序将代码覆盖范围的日志文件转换为表

```
*************************************************************
*  将日志文件转换成表
*************************************************************
cFileName = GETFILE('DBF')
```

```
IF EMPTY(cFileName)
    RETURN
ENDIF
CREATE TABLE (cFileName) ;
    (duration n(5,2), ;
    class c(10), ;
    procedure c(15), ;
    line i(5), ;
    file c(30))
APPEND FROM GETFILE('log') TYPE DELIMITED
*************************************************************
```

上述代码首先打开或者新建一个数据表文件（.DBF），然后将已经存在但未被使用的日志文件（.LOG）的记录内容提炼成各字段存入数据表。比如，将日志文件 xu.log 转换成数据表文件 logfile.dbf，然后可打开该文件并选择 Microsoft Visual FoxPro 主窗口的菜单项“显示”——>“浏览”。

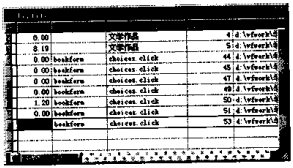


图 5.33 日志文件 xu.log 转换成的数据表文件 logfile.dbf

5.2.5 窗口菜单

从调试器菜单中选择“窗口”可打开下拉窗口菜单，通过“窗口”菜单可以访问活动的“调试器”窗口。

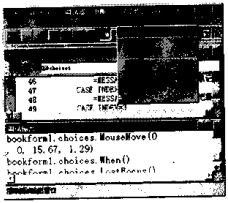


图 5.34 窗口菜单

窗口菜单中包含如下菜单命令：

1.清除输出窗口

删除显示在“调试输出”窗口中的所有文本。只有在显示了“调试输出”窗口并且窗口中有输出文本时，该命令才有效。

2.层叠

允许将所有打开的“调试器”窗口以阶梯方式进行显示，将一个窗口放在另一个窗口上边。

3.还原为默认值

在“调试器”窗口中，按各窗口初始位置显示，并清除所有的监视表达式。同时，删除保存在调试器配置文件中的全部数据。

4.跟踪

激活“跟踪”窗口。

5.监视

激活“监视”窗口。

6.局部

激活“局部”窗口。

7.调用堆栈

激活“调用堆栈”窗口。

8.输出

激活“调试输出”窗口。

如果某窗口被停放，尽管相关的菜单项显示该窗口是活动的，但也无法看到。

5.2.6 帮助菜单

“帮助”菜单包括一些选项，可以打开 Visual FoxPro 的联机帮助，引导找到技术支持，并能够显示计算机的配置信息。该菜单项与 Visual FoxPro 主窗口的“帮助”菜单相同，它所包含的菜单命令有：

1.Microsoft Visual FoxPro 帮助主题

显示一个对话框，可以按关键字查找帮助主题。

2.文档

显示 Visual FoxPro CD 文档。

3.示例应用程序

显示 Visual FoxPro 示例应用程序帮助主题。

4.Microsoft on the Web

显示“Microsoft on the Web”子菜单，可以访问有用的 Microsoft Web 页面。

5.技术支持

显示有关 Microsoft 产品支持服务的信息，并回答一些 Visual FoxPro 的一般问题。

6.关于 Microsoft Visual FoxPro

显示 Visual FoxPro 和机器系统的有关信息。

5.3 调试器的工具栏、快捷键和快捷菜单

调试器中的有些菜单命令可以用调试器工具栏、快捷键和快捷菜单的相应选项实现。而且，使用按钮、快捷键和快捷菜单选项有时也比选择调试器菜单选项方便些，前提是要熟悉这些功能。

5.3.1 调试器的工具栏

表中列出了调试器的可用工具栏的按钮项。

表 5.1 调试器的可用工具栏按钮

按 钮	命 令	说 明
	打开	显示“打开”对话框，指定一个显示在“跟踪”窗口中的源程序。打开程序并不运行该程序
	执行/继续执行	开始执行在“跟踪”窗口中打开的程序。若“跟踪”窗口没有打开的程序，显示“执行”对话框，让您能够指定要跟踪的程序（或表单）。指定的程序（或表单）与在可执行代码第一行挂起的执行程序一起执行。执行程序挂起时，可选该按钮继续执行。在代码当前行继续执行“跟踪”窗口中的程序
	取消	关闭和终止“调试”窗口中的执行程序或表单
	跟踪	逐行执行代码
	单步	逐行执行代码。如果下一行代码调用了函数、方法程序或者过程，那么该函数、方法程序或过程在后台执行
	跳出	继续执行一个过程中的代码，但并不是逐行地单步执行。在调用程序中，当走到过程调用的下边一行代码时，程序执行重新被挂起
	运行到光标所在的位置	执行从当前行指示器到光标所在行之间的代码。在您要挂起执行的代码上单击，从而把光标移到这一行上

（续表）

按钮	命令	说明
	“跟踪”窗口	激活“跟踪”窗口
	“监视”窗口	激活“监视”窗口
	“局部”窗口	激活“局部”窗口
	“调用堆栈”窗口	激活“调用堆栈”窗口
	“调试输出”窗口	激活“调试输出”窗口
	切换断点	光标所在行的断点从有变无，从无变有
	清除所有断点	清除所有断点
	“断点”对话框	打开“断点”对话框，可以添加、删除、启用或废止断点
	切换范围覆盖记录	打开“编辑日志”对话框。您可以使用覆盖范围记录，也可为其指定一个文件
	切换事件记录	打开“事件跟踪”对话框。可以指定事件产生时，哪些事件写到“调试输出”窗口或文件中

5.3.2 调试器的快捷键

下表的键盘快捷键适用于“调试器”窗口或 FoxPro 主窗口中的调试工具。

表 5.2 调试器的快捷键

键盘	操作
F5	继续执行
Esc	取消
F8	跟踪
F6	单步
SHIFT+F7	跳出
F7	运行到光标所在的位置
ALT+8	“跟踪”窗口
ALT+3	“监视”窗口

（续表）

键 盘	操 作
ALT+2	“调试输出”窗口
F9	切换断点
CTRL+SHIFT+F9	清除断点
CTRL+B	断点
CTRL+O	打开文件
ALT+S	保存配置
ALT+F4	退出“调试器”

5.3.3 调试器的快捷菜单

或许你已经发现，用鼠标右键单击调试器的不同区域，会弹出不同的菜单，这样的菜单被称为快捷菜单，取意于其使用的方便快捷。其中标题栏的快捷菜单属于 Windows 95 的标准快捷菜单，在 Windows 95 界面的有关章节介绍，这里不再赘述。

1. “调试器工具栏”快捷菜单

该菜单包含“调试器”或者 FoxPro 主窗口的一些常用命令的快捷方式。用鼠标右键单击“调试器”窗口工具栏或“调试器”主窗口或者各窗口（比如“跟踪”窗口）的标题栏可以显示这些命令。下表详细列出了这些命令及其功能。

表 5.3 “调试器工具栏”快捷菜单

快捷菜单选项	功 能
跟踪	显示或隐藏“跟踪”窗口
监视	显示或隐藏“监视”窗口
局部	显示或隐藏“局部”窗口
调用堆栈	显示或隐藏“调用堆栈”窗口
输出	显示或隐藏“调试输出”窗口
菜单	显示或隐藏 Visual FoxPro “调试器”窗口菜单
工具栏	显示或隐藏 Visual FoxPro “工具栏”窗口菜单
状态	显示或隐藏 Visual FoxPro “状态”窗口菜单

2. “跟踪窗口”快捷菜单

“跟踪窗口”快捷菜单包含“调试器”的“跟踪”窗口中的一些命令的快捷方式。用鼠标右按钮单击“跟踪”窗口可以显示这些命令，参见下表。

表 5.4 “跟踪窗口”快捷菜单

快捷菜单选项	功 能
打开	显示“打开”对话框，该对话框允许您选择一个文件
继续执行	允许在调用的程序中执行下一行代码或移到下一可执行的代码行
跟踪	允许在调用的程序中执行下一行代码或移到下一可执行的代码行
单步	允许您执行下行代码。若下行调用了函数、方法程序或过程，被调用的函数、方法程序或过程在后台执行
跳出	允许您连续地执行过程中的代码(而不是一行一行地执行)，在调用程序调用的下一行处，程序执行被重新挂起
设置下一条语句	将当前行标志放在光标所在行，单步执行或恢复执行时该行即被执行
运行到光标处	允许您从当前指示行执行到光标所在行。若在您想挂起执行的代码行上单击鼠标，可将光标置于此行
在两个断点之间跟踪	切换在断点间跟踪执行
停放视图	允许您指定窗口的停放功能是打开还是关闭。若不选定，窗口不能停放
隐藏	隐藏“跟踪”窗口
字体	显示“字体”对话框。在对话框中，您可以设定字型、样式和字体大小。此命令对应于“格式”菜单的“字体”命令
帮助	显示选中项的“帮助”窗口

3.“监视窗口”快捷菜单

包含“调试器”的“监视”窗口中的一些命令的快捷方式。用鼠标右按钮单击“监视”窗口可以显示这些命令，参见下表。

表 5.5 “监视窗口”快捷菜单

快捷菜单选项	功 能
插入监视	插入新的空白监视表达式
删除监视	删除高亮度的监视表达式
停放视图	允许您指定是否打开停放功能。若不选定，窗口不能停放
隐藏	隐藏“监视”窗口
字体	显示“字体”对话框，您可以设定字体类型、风格和字体大小。此命令对应于“格式”菜单的“字体”命令
帮助	显示选中项的“帮助”窗口

4.“局部窗口”快捷菜单

“局部窗口”快捷菜单包含“调试器”的“局部”窗口中的一些常用命令的快捷方式。用鼠标右按钮单击“局部”窗口可以显示这些命令，参见下表。

表 5.6 “局部窗口”快捷菜单

快捷菜单选项	功 能
公有	在“局部”窗口显示公共变量
局部	在“局部”窗口显示局部变量
常用	在“局部”窗口显示常用变量
对象	在“局部”窗口显示对象
停放视图	允许您指定是否打开停放功能。若不选定，窗口不能停放
隐藏	隐藏“局部”窗口
字体	显示“字体”对话框，您可以设定字体类型、风格和字体大小。此命令对应于“格式”菜单的“字体”命令
帮助	显示选中项的“帮助”窗口

5.“调用堆栈窗口”快捷菜单

该菜单包含“调试器”的“调用堆栈”窗口中的一些常用命令的快捷方式。用鼠标右按钮单击“调用堆栈”窗口，可以显示这些命令，参见下表。

表 5.7 “调用堆栈窗口”快捷菜单

快捷菜单选项	功 能
原位置	允许您显示或隐藏原位置指示器
当前过程	允许您显示或隐藏当前过程指示器
调用堆栈指针	允许您显示或隐藏调用堆栈指示器
停放视图	允许您指定是否打开停放功能。若不选定，窗口不能停放
隐藏	隐藏“调用堆栈”窗口
字体	显示“字体”对话框，允许您设置字型、样式及字体大小。此命令对应于“格式”菜单的“字体”命令
帮助	显示选中项的“帮助”窗口

6.“调试输出窗口”快捷菜单

包含“调试器”的“调试输出”窗口中的一些常用命令的快捷方式。用鼠标右按钮单击“调试输出”窗口，可以显示这些命令，参见下表。

表 5.8 “调试输出窗口”快捷菜单

快捷菜单选项	功 能
另存为	显示“另存为”对话框，保存“调试输出”窗口中的内容
清除	清除“调试输出”窗口中的内容
停放视图	允许您指定是否打开停放功能。若不选定，窗口不能停放
隐藏	隐藏“调试输出”窗口
字体	显示“字体”对话框，允许您设置字体类型，风格及字体大小。此命令对应于“格式”菜单的“字体”命令
帮助	显示选中项的“帮助”窗口

5.4 调试器窗口

前面已经提到，Visual FoxPro 5.0 的“调试器”包含“调试器”菜单、“调试器”工具栏以及几个调试窗口。通过对调试器菜单的详细讲解，我们对调试器的使用方法有了大体的了解。调试窗口用来查看储存的值，在“调试程序”各窗口中，很容易在窗口里看到变量、数组元素、属性和表达式的运行值：“跟踪”窗口中显示正在执行的程序行；“局部”窗口可以查阅程序、过程或方法程序中可见的变量、数组、对象以及对象成员；利用“监视”窗口可以显示表达式以及它们的当前值，并可以在一个表达式上设置断点；如果希望显示正在执行的过程、程序以及方法程序可打开“调用堆栈”窗口，还可以通过“调试输出”窗口查看程序中指定调试的输出。

表 5.9 调试器窗口的基本操作

若 要...	请...
打开窗口	从窗口菜单中选择相应菜单命令或者单击与该窗口相关联的工具栏按钮
调整相邻停放窗口的大小	拖动它们之间的分隔栏
浮动一个窗口	从快捷菜单中选择“解除停放”
查看一个打开的隐含窗口	单击与该窗口相关联的工具栏按钮两次：第一次关闭窗口，第二次重新打开窗口

5.4.1 “调用堆栈”窗口

从“调试器”窗口的“窗口”菜单上选择“调用堆栈”，可以激活该窗口。通过这个窗口可以显示正在执行的过程、程序和方法程序。第一个程序运行时，该程序名列在“调用堆栈”窗口中。如果调用了第一个程序中的子程序或子过程，同时，又在执行第二个程序，两个程序的名字均显示在“调用堆栈”窗口中，依此类推。“调用堆栈”窗口中包含两项内容：

1.输出窗格

显示正在执行的过程、程序和方法程序。

2.调用堆栈指示器

指向正在运行的程序、过程和方法程序。

在如图 5.35 所示“调用堆栈”窗口的例子中，每个程序左边有一个指示堆栈调用顺序的编号，当前执行程序具有最大的编号。本例中先执行程序“文学作品.prg”，再执行方法“ bookform1.choices.click ”。图中箭头指向当前程序，三角形箭头指示在“跟踪”窗口中显示的过程。如果当前行与调用堆栈过程相同，Visual FoxPro 只显示当前行指示符。用鼠标左键双击“调用堆栈”窗口中的程序“文学作品.prg”，可以发现该行和“跟踪”窗口的第 5 行的左边出现了三角形箭头，因此可从“跟踪”窗口中查知相应的源程序行。

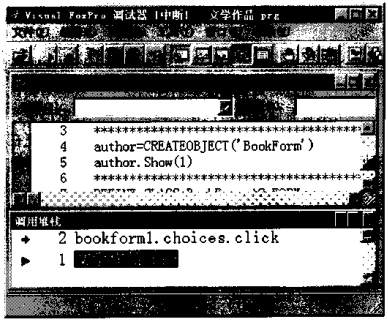


图 5.35 调用堆栈窗口

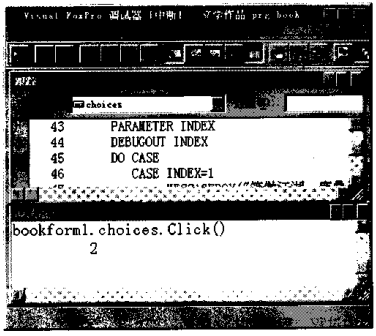


图 5.36 调试输出窗口

5.4.2 “调试输出”窗口

从“调试器”窗口的“窗口”菜单中选择“输出”，可以激活“调试输出”窗口。该窗口可以显示活动程序、过程或方法程序代码的输出。既可以在“调试输出”窗口的输出窗格显示由 DEBUGOUT 命令指定的字符串，同时如果启用了事件跟踪，当系统事件发生时，也可显示事件的名称。若要保存窗口中的文本，在“调试器”窗口中，从“文件”菜单选择“另存输出”。

比如为了在选中某选项时能够输出该选项对应的按钮索引号，可在方法程序 Choices.CLICK 中加入命令：

```
DEBUGOUT INDEX
```

从而原先的 Choices.CLICK 方法程序变成：

```
************************************************************
*  Choices.CLICK 方法程序
************************************************************
PROCEDURE Choices.CLICK
PARAMETER INDEX
DEBUGOUT INDEX
DO CASE
    CASE INDEX=1
        =MESSAGEBOX("笑傲江湖，鹿鼎记...")
    CASE INDEX=2
        =MESSAGEBOX("永失我爱，过把瘾...")
    CASE INDEX=3
        =MESSAGEBOX("白涡...")
    CASE INDEX=4
        =MESSAGEBOX("红粉...")
ENDCASE
ENDPROC
```

图 5.37　执行表单

此时，可以从“输出”窗口中输出以下结果：

```
...
bookform1.choices.Click()
        4
bookform1.Activate()
bookform1.choices.Click()
        2
```

输出中有方法程序如 bookform1.choices.Click()，bookform1.Activate()等，表明跟踪事件中选入了 Click 和 Activate 事件并开启了事件跟踪。输出的“4”和“2”是 DEBUGOUT INDEX 命令的执行结果，分别表示单击了按钮“苏童”和“王朔”从而使得 INDEX=4 和

INDEX=2 。

5.4.3 “局部”窗口

选择“调试器”窗口中“窗口”菜单上的“局部”，可以激活“局部”窗口。使用该窗口可以显示给定的程序、过程或方法程序中的所有变量、数组、对象以及对象成员。该窗口包含以下窗口元素：

1.位置

指明在“局部”窗口显示其变量、数组和对象的过程。

2.名称

显示可视局部变量的名称。

3.值

显示可视局部变量的当前值。

4.类型

显示代表可视局部变量的数据类型的字符。
使用该快捷菜单可以控制显示的变量类型可参见下表。

表 5.10 可视局部变量的数据类型

类 型	说 明
公用	用 PUBLIC 关键字说明的变量
局部	用 LOCAL 关键字说明的变量
常用	在“位置”框中显示的过程作用域内的所有变量
对象	对象变量

如图 5.38 是调试程序“文学作品.prg ”时在“局部”窗口中显示的内容。若从“位置”下拉列表框中选择 bookform1.choices.click 过程，可显示该过程的变量、数组和对象的值和类型，比如 INDEX 的值为 4，为数值型（ NUMERIC ）。还可以查看对象（如 author ）和数组（如 choices ）的类型和值。若要查看数组的元素值或对象的属性值，请单击名称左边的加号展开数组或对象（如数组 choices ），展开后名称左边显示减号（如对象 author ）。通过“局部”窗口可以方便地查看容器层次中所有对象的属性值。

5.4.4 “跟踪”窗口

选择“调试器”窗口中“窗口”菜单的“跟踪”选项，可激活该窗口。代码执行时，可以利用该窗口观察到这些代码。该窗口包含以下内容：

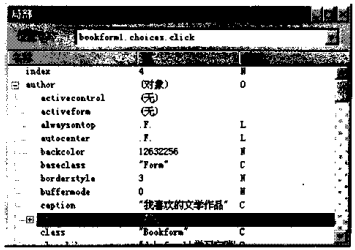


图 5.38　局部窗口

1.对象

在运行对象代码时可用。“跟踪”窗口中的代码与“对象”列表中显示的对象相关。对于那些代码正在执行的对象，它们的最高级父容器所包含的所有对象均包括在“对象”列表中。

2.过程

在运行对象代码时可用。“跟踪”窗口中的代码与“过程”列表中显示的方法程序或事件相关。“过程”列表包括“对象”列表中对象（指有相关代码的对象）的所有方法程序。“跟踪”窗口的左部分灰色区可包括实心圆点、箭头、空心圆点和三角形四种符号，其意义参见下表。

表 5.11　“跟踪”窗口的符号

符　号	名　称	说　明
	箭头	正在执行的代码行
	实心圆点	活动断点
	空心圆点	不活动的断点
	三角形	若检查不同层次上的代码（非当前执行代码），调用堆栈位置

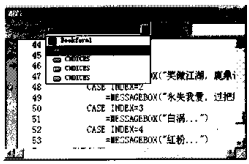


图 5.39　跟踪窗口

5.4.5 “监视”窗口

从“调试器”窗口的“窗口”菜单中选择“监视”，可以激活该窗口，用以显示表达式及它们的当前值，并能够在表达式上设置断点。

1.监视

待查的表达式或值。在此处键入表达式，可以把它们添加到下面的列表框内。

2.名称

显示当前监视表达式的名称。

3.值

显示当前监视表达式的值。

4.类型

显示代表当前监视表达式的数据类型的字符（数据类型的第一个字母）。

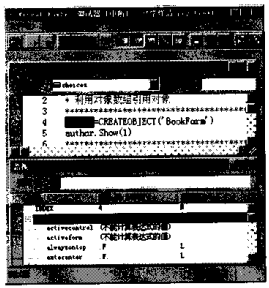


图 5.40 监视窗口

除了在“名称”输入框中直接输入表达式外，还可以选定 Visual FoxPro 任一窗口内的文本，并将其拖至“监视”窗口来创建一个新的监视表达式。若要修改一个监视表达式，请在表达式所在行处于选中状态时单击表达式或者直接双击表达式。

第六章　发挥面向对象的威力

面向对象（Object-Oriented Programming，简称 OOP）编程是程序设计在方法和思维上的巨大进步。它引进近乎完美的抽象机制，用人们易于理解的方式提炼万事万物，解释自然现象、社会现象以及处理日常生活中的各种问题。在面向对象的世界中，人们通常主要考虑“做什么”而不是“怎么做”的问题。因此，若将面向过程设计方法比作士兵的亲临作战，则面向对象方法好比将军的调兵遣将。

6.1　OOP 简介

Visual FoxPro 5.0 提供了完全面向对象编程的功能，用户可以使用几乎所有面向对象设计的特性，这些特性包括基类、子类、继承性、多态性以及封装等等。本节旨在建立面向对象的一些常用的基本概念，使读者对此有一定理解。

6.1.1　OOP 术语快速入门

概括地说，面向对象是一种系统化的程序设计方法，允许抽象化、模块化的分层结构，具有多态性、继承性和封装性。

1.类（Class）

类是定义了客观事物基本特征以及事物外观和行为的模板。比如，自然界的生物是一类事物，其基本特征是：

（1）有生命，生命有长短；
（2）能进行新陈代谢；
（3）由细胞组成，细胞有多少分别；
（4）能繁衍后代，但繁衍方式不同；
（5）能遗传变异.....

每一种性质都描述了生物，使它严格的区别于非生物从而更加确定。在 Visual FoxPro 中，能帮助很好理解类的意义的例子是表单（Form）。在屏幕上画一个确定的表单，要使之独一无二，下列信息必不可少：

（1）宽度（Width）；
（2）高度（Height）；
（3）左边距（Left）；
（4）上边距（Top）；
（5）图标（Icon）；
（6）标题（Caption）；
（7）背景（BackColor）；
（8）前景（ForeColor）.....

也正因为具有上述限制条件，该事物才成其为事物。这些对事物进行描述的性质称为事物的属性。事物还需要完成一定的动作，这些动作称为方法。限制条件不同，形成不同的事物。但所有这些事物都统称为一类，即表单类。同理，在 Visual FoxPro 中还有命令按钮类、标签类、文本框类、微调控件类等等。类按范围的大小可以划分为基类（ Base Class ），子类（ SubClass ）和用户自定义类（ User-Defined Class ）。基类是在 Visual FoxPro 内部定义的类，可用作其他用户自定义类的基础。例如， Visual FoxPro 表单和所有控件就是基类，用户可以在此基础上创建新类，增添自己需要的功能。子类是以其他类定义为起点，对某一对象所建立的新类。新类将继承任何对父类所做的修改。用户自定义类与 Visual FoxPro 基类相似，但由用户定义，并且可用来派生子类。这种类没有可视表示形式。这是容易理解的，因为在自然界中也并非所有的事物都是可见的，比如气体、声音、电磁场等。实际上， Visual FoxPro 中除了可视类外，还存在非可视类。

2.实例（ Instance ）

实例即事例，是类的具体应用。比如用一个具有一定大小、背景、前景和字体等特性的表单创建加工成的应用程序表单就是表单的一个实例（如图 6.1 ）。

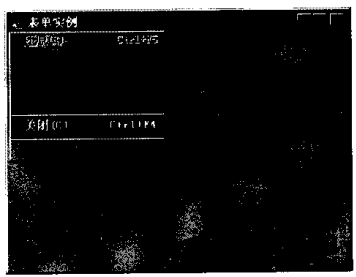


图 6.1 表单实例

该实例在具体运用表单类时，主要规定了以下属性：

（ 1 ） Caption=‘表单实例’

（ 2 ） Icon=‘ d:\vfwork\note17.ico ’

（ 3 ） Width=352

（ 4 ） Height=247

（ 5 ） Top=24

（ 6 ） Left=101

.....

另外，双击图标可以关闭该表单，单击最大化最小化按钮可以改变表单大小等所有这些都说明表单具有了一定的方法过程。

3.对象

所谓对象，就是这样一个实例，包括了数据和过程。在 Visual FoxPro 中，创建的对象也具有属性，这些属性由对象所基于的类决定。属性值既能在设计时也可在运行时进行设置。比如，复选框可能有的属性如下表：

表 6.1　复选框的可用属性

属性	说　明
Caption	复选框旁边的说明性文字
Enabled	复选框能否被用户选择
ForeColor	标题文本的颜色
Left	复选框左边的位置
MousePointer	在复选框内鼠标指针的形状
Top	复选框顶边的位置
Visible	复选框是否可见

使用 Windows 时，用户通常通过下列动作来操作计算机：单击鼠标、移动鼠标和按键。这些能为系统接收的动作效果称为事件。 在 Visual FoxPro 中，每个对象都可以对一个事件的动作进行识别和响应。事件是一种预先定义好的特定动作，由用户或系统激活。在多种情况下，事件是通过用户的交互操作产生的。事件集合虽然范围很广，但却是固定的。用户不能创建新的事件，然而方法程序集合却可以无限扩展。下表列出了与复选框相关联的一些事件。

表 6.2　复选框的可用事件

事件	说　明
Click	用户单击复选框
GotFocus	用户选择复选框
LostFocus	用户选择其他控件

方法程序是与对象相关联的过程，但又不同于一般的 Visual FoxPro 过程。方法程序紧密地和对象连接在一起，并且与一般 Visual FoxPro 过程的调用方式也有所不同。事件可以具有与之相关联的方法程序。例如，为 Click 事件编写的方法程序代码将在 Click 事件出现时被执行。方法程序也可以独立于事件而单独存在，此类方法程序必须在代码中被显式地调用。调用方法为：

容器对象.控件对象.方法

例如：

Form1.CommandButton1.Click

对象具有与之相关联的方法程序，例如与复选框关联的一些方法程序如下表。

表 6.3 复选框的可用方法程序

方法程序	说 明
Refresh	复选框中的值被更新，以反映隐含数据源的数据变化
SetFocus	焦点被置于复选框，好象用户刚使用 TAB 键选中复选框

4.容器类（ Container Classes ）

Visual FoxPro 中的基类可以分为容器类和控件类。容器类是包容其他相似类的 Visual FoxPro 基类，它可以容纳别的对象。例如，一个表单类中可以包含一组控件类，将这些类作为一个整体进行操作，另外，还可以放入复选框、编辑框、直线和文本框等。因此表单属于容器类。在基类中有如下容器类：

（ 1 ）列（ Column ） 可以容纳标头等对象，但不能容纳表单集、表单、工具栏和计时器。

（ 2 ）命令按钮组（ Command Button Group ）只能容纳命令按钮。

（ 3 ）表单（ Form ）可容纳页框、任意控件、容器或自定义对象。

（ 4 ）表单集（ FormSet ）可容纳表单、工具栏。

（ 5 ）表格（ Grid ）只能容纳表格列。

（ 6 ）选项按钮组（ Option Button Group ）只能容纳选项按钮。

（ 7 ）页面（ Page ） 可容纳任意控件、容器和自定义对象。

（ 8 ）页框（ PageFrame ）只能容纳页面。

（ 9 ）工具栏（ ToolBar ）可容纳任意控件、页框和容器。

5.控件类（ Control Classes ）

控件类是可以包含在容器类中并由用户派生的 Visual FoxPro 基类。控件类不能容纳其他对象，比如不能把命令按钮放入编辑框中。当将一个控件对象放入容器中，需要引用对象时必须经过容器。引用容器中控件对象的语法为：

容器对象.控件对象.属性

例如：

Form1.CommandButton1.Caption="关闭"

命令按钮和文本框就属于控件类。下面列出了基类中常用的控件类：

（ 1 ）复选框控件（ CheckBox ）

（ 2 ）组合框控件（ ComboBox ）

（ 3 ）命令按钮控件（ CommandButton ）

（ 4 ）编辑框控件（ EditBox ）

（5）图象控件（Image）
（6）标签控件（Label）
（7）线条控件（Line）
（8）列表框控件（ListBox）
（9）选项按钮控件（OptionButton）
（10）形状控件（Shape）
（11）微调控件（Spinner）
（12）文本框控件（TextBox）
（13）计时器控件（Timer）

6.继承性（Inheritance）

在面向对象程序设计方法中，继承性用来说明子类延用父类特征的能力。这有点类似于生物的遗传，黄种人继承了祖先的控制生成黄肤色的基因，而白种人继承了祖先的控制生成白肤色的基因，这里的基因就相当于所谓的属性。如果父类特征发生改变，则子类将继承这些新特征。例如，如果为一个编辑控件添加了一个新属性为 IsBold，那么以此控件为基础派生的子类也将拥有 IsBold 属性。

7.多态性（Polymorphism）

多态性也是面向对象程序设计专用术语。主要是指一些关联的类包含同名的方法程序，但方法程序的内容可以不同。在运行时根据对象的类确定具体调用哪种方法程序。例如，相关联的几个对象可以同时包含 Draw 方法程序，当某个过程将其中一个对象作为参数传递时，它不必知道该参数是何种类型的对象，只需调用 Draw 方法程序即可。因此，多态性允许每个对象对公共消息格式作出响应，但不同对象接收相同的函数调用可以导致完全不同的行为结果。利用多态性，用户不用考虑函数的实现细节，只在程序中进行一般形式的函数调用。

8.封装性（Encapsulation）

另外一个非常重要的面向对象程序设计专用术语是封装性，它用来说明包含和隐藏对象信息（如内部数据结构和代码）的能力。封装将操作对象的内部复杂性与应用程序的其他部分隔离开来。例如，当用户对一个命令按钮设置 Caption 属性时，不必了解标题字符串是如何存储的。

6.1.2　OOP 的优点

Visual FoxPro 5.0 的面向对象编程是建立在一系列类的基础上的。用户可以在已有基类的基础之上创建自己的新类或者完全使用自定义的类，然后进一步创建对象。各类可通过对象关联属性和方法，对象的属性和方法是类的相应属性方法的具体体现和应用。在已有类的基础上建立对象这样一种对象机制具有如下特点和优点：

1.类、对象机制从原理和方法上有利于程序设计

利用类可以创建多个对象。在需要创建相似的对象之前，将对象的相似性加以归纳总结，进而提炼出一些特性，建立能用来创建对象的类。然后创建任意数目的对象，同一个类创建时产生的对象是相同的，但在修改属性值和方法程序之后，就赋予了具体不同对象自己的特征。对象不仅具有定义其物理性质的属性，还具有定义对象与周围事物和事件进行联系的方法。在完成了类的定义工作后，类属性的定义和方法过程的实现等繁多的过程都封装在类定义的内部，编程的主体工作集中在考虑如何实现对象与程序其他部分的联系。引入多态性的概念，不同的类和对象可以具有相同的属性和方法名。引入继承性的概念，使子类可以继承父类的全部属性、事件和方法。用户可以修改父类的属性和方法，形成更适合自己实际应用的子类。同时类的继承性降低了修改程序的难度，使程序代码易于维护。

2.程序具有很高的重复利用率

正因为引入了面向对象的编程机制，Visual FoxPro 中允许用户对经常使用的对象类按照自己的习惯创建类库。在以后需要在应用程序中创建相同或相似的对象时，可以借用存在的类或对已有的类稍微加以修改。习惯使用类库是 Visual FoxPro 程序开发人员的基本素质，它能避免在许多过程代码上重复开发的浪费。同时，在分发自己的开发成果时通过类库，提供代码与接受代码双方也容易交接。

3.代码更加紧凑

面向对象的编程方法是一种模块化的方法，使用模块方法可以使代码更加紧凑。

4.降低了问题的复杂性

在面向过程的编程方法中，不仅需要考虑对象表象之间的联系以及各对象在系统中的作用，还需要考虑对象的内部实现和各对象在细节上的关系。甚至各对象之间错综复杂，并不能明确分开，以致于没有严格意义上的对象。面向对象的编程机制提供对象作为接口，将无论多么复杂的问题都可以简化为两个方面的问题：对象和对象间的联系。

6.2 可 视 类

在 Visual FoxPro 中具有基类和自定义类。在程序运行时，基于有些基类的子类具有可视对象或者可视元件，这一些类称为可视类。基类中的可视类很多，容器类的如表单、表格、页面，控件类的如命令按钮、组合框、复选框等。

用面向对象的方法创建表单，具体有两种实现方法：一是利用表单设计器，二是用编程的方法，这两种方法是相通的。下面先观察一个表单应用的例子，然后我们分别用两种方法实现。

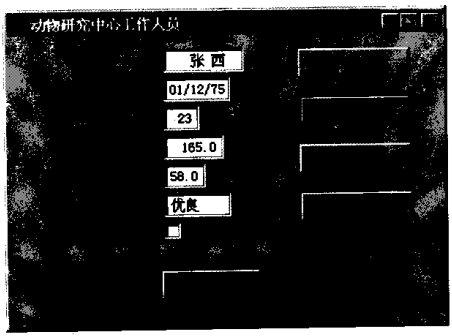


图 6.2　表单应用实例

图 6.2 所示是一个管理动物研究中心工作人员基本情况的表单，用户可按记录查看工作人员的姓名、生日、年龄、身高、体重健康状况和婚姻状况等。用鼠标单击“上一条记录”按钮和“下一条记录”按钮可移动记录指针，查看不同记录。单击“新增加记录”按钮可添加记录，用“删除记录”按钮可删除当前记录。表单下方有“保护环境”和“匹夫有责”的字样。另外，单击“退出”按钮可退出表单。下面就来实现上述功能。

6.2.1　通过表单设计器

1.创建表单对象

用表单设计器创建表单的方法为：

（1）打开表单设计器。在命令窗口中输入命令：

create form

或者从菜单中选择“文件”——>“新建”，并从“新建”对话框中选择“表单”类型，然后单击“新建文件”按钮。

（2）设置表单属性。在表单中单击鼠标右键可弹出快捷菜单（如图 6.3），选中“属性”选项可打开属性窗口（如图 6.4），在属性窗口中主要设置以下属性。

Top = 23

Left = 57

DoCreate = .T.

Caption = "动物研究中心工作人员"

Name = "Form1"

设置完以上属性后的表单如图 6.5 。

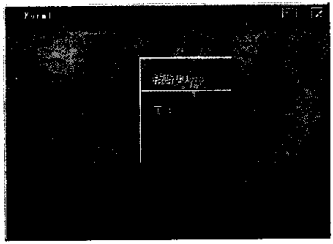


图 6.3 表单的属性菜单

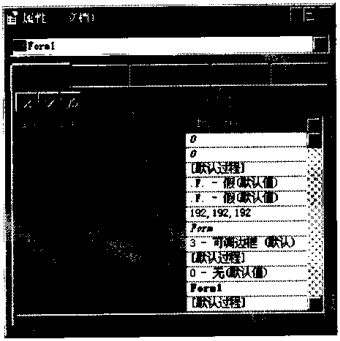


图 6.4 表单的属性窗口

图 6.5 表单的初步设计

（3）以上已经完成了初步的工作，先将文件存档。从“文件”菜单中选择“保存”，这里将此表单保存为“人员管理.scx”。

2.利用数据环境添加控件

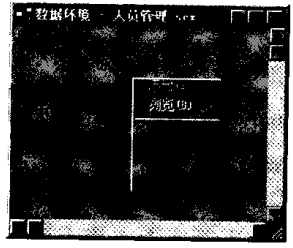


图 6.6　数据环境设计器

图 6.7　将表添加到数据环境

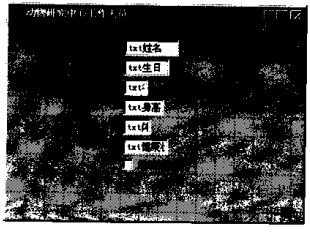


图 6.8　引入字段的表单

（1）加载数据环境。从菜单中选择“显示”——>“数据环境”，或则在表单中单击鼠标右键，从快捷菜单中选取“数据环境”，可打开“数据环境”对话框。在对话框中单击右键打开快捷菜单，选取“添加”项，可加入数据表。这里加入表“工作人员.dbf”。

（2）利用数据环境添加控件。将数据环境中“工作人员”表的字段拖到表单中放在适当的位置，如图 6.7。系统根据各字段的类型添加适当的控件，并将字段名用标签表示在字段的旁边。本例中添加了文本框和复选框两种类型。文本框的大小是根据字段的长度自动确定的，也可以在属性窗口修改。

（3）打开属性窗口，可以检查所加控件的属性设置。利用数据环境添加控件的一个简单的拖放操作，Visual FoxPro 系统在内部实现了许多具体设置。从属性窗口可以看到刚才添加的控件的详细设置。下面用程序语言格式列出了这些设置。

1）姓名文本框的属性设置：

```
ControlSource = "工作人员.姓名",;
Height = 20,;
Left = 135,;
TabIndex = 2,;
Top = 12,;
Width = 69,;
Comment = "",;
Name = "txt 姓名"
```

2）姓名标签的属性设置：

```
AutoSize = .T.,;
BackStyle = 0,;
Caption = "姓名",;
Left = 77,;
Top = 12,;
TabIndex = 1,;
Name = "lbl 姓名"
```

3）生日文本框的属性设置：

```
ControlSource = "工作人员.生日",;
Height = 20,;
Left = 135,;
TabIndex = 4,;
Top = 37,;
Width = 57,;
Comment = "",;
Name = "txt 生日"
```

4）生日标签的属性设置：

```
AutoSize = .T.,;
BackStyle = 0,;
```

Caption = "生日", ;
Left = 77, ;
Top = 37, ;
TabIndex = 3, ;
Name = "lbl 生日"

5）年龄文本框的属性设置：

ControlSource = "工作人员.年龄", ;
Height = 20, ;
Left = 135, ;
TabIndex = 6, ;
Top = 62, ;
Width = 30, ;
Comment = "", ;
Name = "txt 年龄"

6）年龄标签的属性设置：

AutoSize = .T., ;
BackStyle = 0, ;
Caption = "年龄", ;
Left = 77, ;
Top = 62, ;
TabIndex = 5, ;
Name = "lbl 年龄"

7）身高文本框的属性设置：

ControlSource = "工作人员.身高", ;
Height = 20, ;
Left = 135, ;
TabIndex = 8, ;
Top = 87, ;
Width = 51, ;
Comment = "", ;
Name = "txt 身高"

8）身高标签的属性设置：

AutoSize = .T., ;
BackStyle = 0, ;
Caption = "身高", ;
Left = 77, ;
Top = 87, ;
TabIndex = 7, ;
Name = "lbl 身高"

9）体重文本框的属性设置：

```
ControlSource = "工作人员.体重", ;
Height = 20, ;
Left = 135, ;
TabIndex = 10, ;
Top = 112, ;
Width = 35, ;
Comment = "", ;
Name = "txt 体重"
```

10）体重标签的属性设置：

```
AutoSize = .T., ;
BackStyle = 0, ;
Caption = "体重", ;
Left = 77, ;
Top = 112, ;
TabIndex = 9, ;
Name = "lbl 体重"
```

11）健康状况文本框的属性设置：

```
ControlSource = "工作人员.健康状况", ;
Height = 20, ;
Left = 135, ;
TabIndex = 12, ;
Top = 137, ;
Width = 57, ;
Comment = "", ;
Name = "txt 健康状况"
```

12）健康状况标签的属性设置：

```
AutoSize = .T., ;
BackStyle = 0, ;
Caption = "健康状况", ;
Left = 77, ;
Top = 137, ;
TabIndex = 11, ;
Name = "lbl 健康状况"
```

13）婚姻状况复选框的属性设置：

```
Top = 162, ;
Left = 135, ;
Height = 16, ;
Width = 69, ;
```

Caption = "婚姻状况",;
ControlSource = "工作人员.婚姻状况",;
Comment = "",;
Name = "chk 婚姻状况"

（4）现在运行表单。用鼠标右键单击表单弹出快捷菜单，从中选择第一项“执行表单”。如果尚未存盘，应在出现的消息框中选择“是”。此时表单已经具备了一定的功能。如图 6.9，表单实际上显示了“工作人员”表中的当前记录即第一条记录。

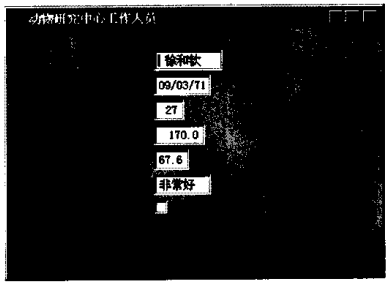


图 6.9　表单的初步运行

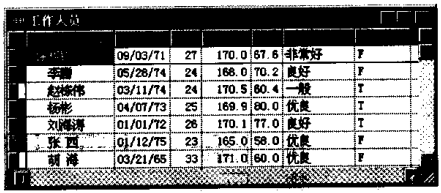


图 6.10　数据表

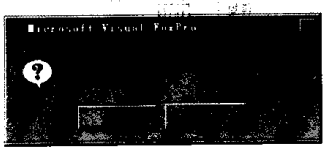


图 6.11　消息框

3.添加命令按钮

很显然，只能显示第一条记录的表单是不能让人满意的。用命令按钮可实现浏览所有记录。还可以添加和删除记录。

（1）使用表单控件工具栏添加命令按钮。如果没有打开控件工具栏，可从菜单中选择“显示”——>“工具栏”，然后选取“表单控件”。如图 6.12 所示。

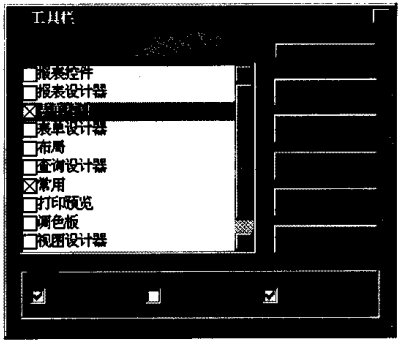


图 6.12 从工具栏中选取“表单控件”

从工具栏中选取“命令按钮”，然后用鼠标在表单的适当位置拖动形成一定尺寸的命令按钮 Command1，如图 6.13 所示。

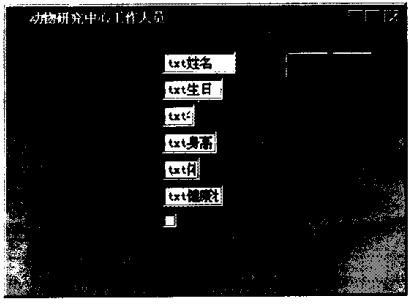


图 6.13 在表单中添加命令按钮

（2）设置按钮属性。用鼠标右键单击该按钮弹出快捷菜单，从中选择“属性”选项打开属性窗口。在属性窗口中设置如下属性：

Top = 13, ;
Left = 251, ;
Height = 25, ;
Width = 96, ;
Caption = "上一条记录", ;
MousePointer = 1, ;
Name = "Command1"

（3）执行表单，此时的命令按钮如图 6.14 。单击“上一条记录”按钮，按钮能被按下，但是除此之外没有任何其他反应。这不奇怪，因为我们只是给命令按钮设置了属性，还没有添加任何方法。关闭该表单，咱们继续进行下面更有意思的工作！

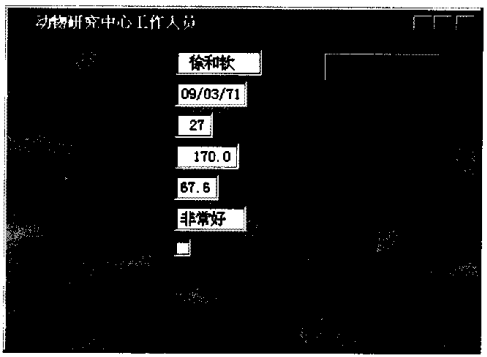


图 6.14　执行表单

4.给命令按钮添加方法程序

正如按钮名所表示的意义，该按钮要实现的功能是：单击该按钮，能使表的记录指针向后退一条记录，并显示当前指针所指的记录。当指针已经指在表的第一条记录时，应使该按钮无效。下面来实现这些功能。

（1）在表单设计器中双击“上一条记录”按钮打开代码输入框。确信此时选取了“ Command1 ”对象和“ Click ”过程（如图 6.15 ）。如果不是，则需要从相应列表框中选取。

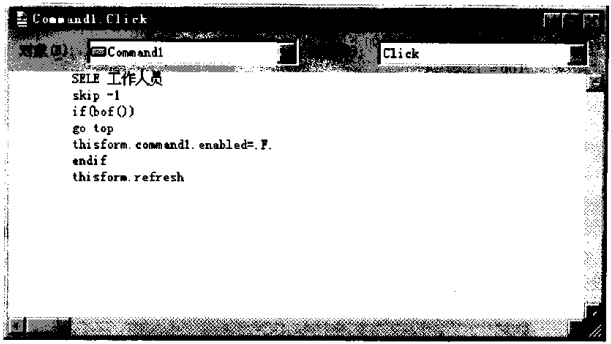


图 6.15 给命令按钮连入代码

（2）给 Click 事件联入如下代码：

```
*************************************************************************
        thisform.command2.enabled=.T.
        SELE 工作人员
        recall
        if(bof())
        MessageBox("已经到了文件第一条记录！", 48)
        thisform.command1.enabled=.F.
        else
        skip -1
        endif
        thisform.refresh
*************************************************************************
```

这里先介绍本例中要用到几个命令、函数和方法：

1）SELE 命令。该命令激活指定工作区。其用法：

SELECT nWorkArea | cTableAlias

其中参数 nWorkArea 指定要激活的工作区。如果 nWorkArea 为零，则激活尚未使用的工作区中编号最小的那一个。参数 cTableAlias 指定要激活的、包含打开表的工作区。cTableAlias 是打开表的别名，也可以用从 A 到 J 中的一个字符作为 cTableAlias 来激活前十个工作区中的一个。

默认情况下，启动 Visual FoxPro 时打开编号为 1 的工作区。在任何工作区中打开的表字段均可以在 Visual FoxPro 的命令和函数中使用。访问非当前工作区中打开表的字段时，应使用如下格式：alias.field 或者 alias -> field 。

这里使用的命令：

```
SELE 工作人员
```

激活了包含打开表“工作人员.dbf ”的工作区。

2 ） SKIP 命令。该命令使记录指针在表中向前移动或向后移动。其用法为：

```
SKIP
    [nRecords]
    [IN nWorkArea | cTableAlias]
```

其中参数 nRecords 指定记录指针需要移动的记录数。使用不带 nRecords 参数的 SKIP 命令将使记录指针走到下一个记录。如果 nRecords 为正数，记录指针向文件尾移动 nRecords 个记录；如果 nRecords 为负数，记录指针将向文件头移动 nRecords 个记录。如果记录指针指向表的最后一个记录，并且执行不带参数的 SKIP 命令时，RECNO( ) 函数返回值比表中记录总数大 1 ，EOF( ) 函数返回“真”(.T.)；如果记录指针指向表的第一个记录，并且执行 SKIP 1 命令，则 RECNO( ) 函数返回 1 ，BOF( ) 函数返回“真”(.T.)。参数 IN nWorkArea | cTableAlias 在指定工作区的表中移动记录指针。nWorkArea 指定工作区编号，cTableAlias 指定一个表或工作区的别名。

这里使用的命令：

```
skip -1
```

使记录指针后退一条记录。

3 ） GO 命令。用该命令可将记录指针移动到指定记录上。其用法为：

```
GO [RECORD] nRecordNumber [IN nWorkArea | IN cTableAlias]
— 或者 —
GO TOP | BOTTOM [IN nWorkArea | IN cTableAlias]
— 或者 —
GOTO [RECORD] nRecordNumber [IN nWorkArea | IN cTableAlias]
— 或者 —
GOTO TOP | BOTTOM [IN nWorkArea | IN cTableAlias]
```

其中参数 RECORD nRecordNumber 指定一个物理记录号，记录指针将移至该记录。可以省略 GO 或 GOTO 命令而只指定记录号，但如果仅指定记录号，则只能在当前工作区中移动记录指针。参数 IN nWorkArea 指定表所在的工作区，记录指针在此表中移动。参数 IN cTableAlias 指定表的别名，记录指针在此表中移动。参数 TOP 将记录指针定位在表的第一个记录上。如果此表使用升序索引，则第一个记录是关键字值最小的记录；如果使用降序索引，则第一个记录是关键字值最大的记录。参数 BOTTOM 将记录指针定位在表的最后一个记录上。如果此表使用升序索引，则最后一个记录是关键字值最大的记录；如果使用降序索引，则最后一个记录是关键字值最小的记录。可以互换使用 GO 和 GOTO 命令。除非在 IN 子句中指定了另一个工作区，否则这两个命令都对当前工作区中的表进行操作。

这里正是使用命令：

```
go top
```

来将指针停留在第一条记录。

4 ） BOF() 函数。使用该函数可确定当前记录指针是否在表头。其用法为：

```
BOF([nWorkArea | cTableAlias])
```

其中参数 nWorkArea 指定在非当前工作区中打开的表的工作区号。参数 cTableAlias 指定在非当前工作区中打开的表别名。如果表是在非当前选定工作区中打开的，那么使用这些可选的参数为表指定别名或所在的工作区号。若此表未在指定的工作区中打开，则 BOF( ) 返回“假”(.F.)。BOF( ) 函数可以用来测试一个表的记录指针是否位于文件头。如果用户尝试将记录指针移动到表的第一条记录之前，则 BOF( ) 返回“真”(.T.)。

5 ） EOF() 函数。该函数用来确定记录指针位置是否超出当前表或指定表中的最后一个记录。其用法为：

```
EOF([nWorkArea | cTableAlias])
```

其中参数 nWorkArea 指定表所在的工作区号，参数 cTableAlias 指定表的别名。如果指定工作区中没有打开的表，则 EOF( ) 函数返回“假”(.F.)。如果没有指定工作区或别名，则检查当前选定工作区中打开的表，看是否到达了表的最后一个记录。如果记录指针已指向表文件的末尾 (EOF)，则 EOF( ) 返回“真”(.T.)。当记录指针超过表中的最后一个记录时，即到达表的末尾。例如，当 FIND 、 LOCATE 或 SEEK 命令不成功时， Visual FoxPro 将把记录指针移动到最后一个记录之后， EOF( ) 函数返回“真”(.T.)。当记录指针不在表的末尾时， EOF( ) 函数返回“假”(.F.)。

6 ） REFRESH 方法。该方法用来重画表单或控制，并刷新所有值。其用法为：

```
[[FormSet.]Object.]Refresh
Object.Refresh
```

一般地，画表单或控制是在没有事件发生时自动处理的。需要立刻更新表单或控制时可使用 Refresh 方法。可使用 Refresh 方法强制地完全重画表单或控制，并更新控制的值。若要在加载另一个表单的同时显示某个表单，或更新控制的内容时， Refresh 方法很有用。要更新组合框或列表框的内容，请使用 Requery 方法。刷新表单的同时，也刷新表单上所有的控制；但在刷新页框时，只刷新活动的页。

这里我们用：

```
thisform.refresh
```

来刷新表单。

7 ） MessageBox() 函数。该函数可显示一个用户自定义对话框。其用法为：

```
MESSAGEBOX(cMessageText [, nDialogBoxType [, cTitleBarText]])
```

其中参数 cMessageText 指定在对话框中显示的文本。在 cMessageText 中包含回车符 (CHR(13)) 可以使信息移到下一行显示。对话框的高度和宽度根据 cMessageText 适当增大，以包含全部信息。参数 nDialogBoxType 指定对话框中的按钮和图标、显示对话框时的默认按钮以及对话框的行为。在下面的表中，对话框按钮值从 0 到 5 指定了对话框中显示的按钮。图标值 16 、 32 、 64 指定了对话框中的图标。默认值 0 、 256 、 512 指定对话框中哪个按钮为默认按钮。当显示对话框时选中此默认按钮，当省略 nDialagBoxType 时，等同于指定 nDialagBoxType 值为 0 。

表 6.4　消息框的按钮设置

数　值	对话框按钮
0	仅有“确定”按钮
1	“确定”和“取消”按钮
2	“放弃”、“重试”和“忽略”按钮
3	“是”、“否”和“取消”按钮
4	“是”、“否”按钮
5	“重试”和“取消”按钮

表 6.5　消息框的图标设置

数　值	图　标
16	“停止”图标
32	问号
48	惊叹号
64	信息 (i) 图标

表 6.6　消息框的默认按钮

数　值	默认按钮
0	第一个按钮
256	第二个按钮
512	第三个按钮

nDialogBoxType 可以是三个值的和从上面每个表中选一个值。例如 nDialogBoxType 为 290（2+32+256），则指定的对话框含有如下特征：“放弃”、“重试”或“忽略”按钮；消息框显示问号图标；第二个按钮，“重试”为默认按钮。

参数 cTitleBarText 指定对话框标题栏中的文本。若省略 cTitleBarText，标题栏中将显示“Microsoft Visual FoxPro”。MESSAGEBOX( ) 的返回值标明选取了对话框中的哪个按钮。在含有取消按钮的对话框中，如果按下 ESC 键退出对话框，则与选取“取消”按钮一样，返回值(2)。注意本函数的最短缩写为 MESSAGEB( )。下表列出了 MESSAGEBOX( ) 对应每个按钮的返回值。

表 6.7　消息框的返回值

返回值	按钮
1	确定
2	取消
3	放弃

（续表）

返回值	按钮
4	重试
5	忽略
6	是
7	否

8）SetFocus() 方法。用该方法为一个控制指定焦点。其用法为：

```
Control.SetFocus
```

如果控制的 Enabled 或 Visible 属性设置为“假”(.F.)，或者控制的 When 事件返回“假”(.F.)，则不能给一个控制指定焦点；如果 Enabled 或 Visible 属性已设置为“假”(.F.)，则控制在使用 SetFocus 方法接受焦点之前，必须首先把它们设置为“真”(.T.)。一旦控制获得了焦点，用户的任何输入都针对这个控制。比如在按钮“新增加记录”的 Click 方法中使用了语句：

```
thisform.txt 姓名.setfocus
```

将焦点置于姓名的文本框中以便输入。

9）DELETE 命令。该命令给要删除的记录做标记，但并不实际删除记录。起其用法：

```
DELETE
    [Scope] [FOR lExpression1] [WHILE lExpression2]
    [IN nWorkArea | cTableAlias]
    [NOOPTIMIZE]
```

其中参数 Scope 指定要做删除标记的记录范围，Scope 子句有：ALL、NEXT nRecords、RECORD nRecordNumber 和 REST。DELETE 的默认范围是当前记录(NEXT 1)。参数 FOR lExpression1 指定一个条件，仅给满足逻辑条件 lExpression1 的记录做删除标记。如果 lExpression1 是一个可优化表达式，且表在 DELETED() 上建立索引时，可以用 Rushmore 优化 DELETE ... FOR 创建的查询。要得到最佳性能，可在 FOR 子句中使用一个可优化表达式。

10）PACK 命令。该命令从当前数据库中删除标有删除标记的记录。用法为：

```
PACK DATABASE
```

当从数据库中删除一个表或视图，或者修改数据库中一个表的结构时，数据库中就会包含做过删除标记的记录。必须以独占方式打开数据库，并且数据库中不能打开任何表或视图。

11）APPEND。该命令在表的末尾添加一个或多个新记录。其用法为：

```
APPEND [BLANK]
    [IN nWorkArea | cTableAlias]
    [NOMENU]
```

其中参数 BLANK 在当前表的末尾添加一个空记录。Visual FoxPro 在发出 APPEND BLANK 命令时并不打开编辑窗口。可以使用 BROWSE、CHANGE 或 EDIT 命令编辑

新记录。参数 IN nWorkArea 指定要添加新记录的表所在的工作区。参数 IN cTableAlias 指定要添加新记录的表的别名。如果省略 nWorkArea 和 cTableAlias，新记录将添加到当前选定工作区的表中。如果发出 APPEND 命令，空记录将添加到由 nWorkArea 或 cTableAlias 指定的工作区的表中，并且自动选定该表；如果发出 APPEND BLANK 命令，空记录将添加到指定的 nWorkarea 或 cTableAlias 工作区的表中，但不选定表。

（3）关闭代码输入框，保存当前表单。

（4）执行表单。因为此时还没有添加命令按钮 Command2，运行前要将 Click 方法程序中的第一行注释掉，即：

```
* thisform.command2.Enabled=.T.
```

单击“上一条记录”，由于当前记录指针指在第一条记录，无上一条记录，因此按钮变成表示无效的灰色。这是因为在过程中使用了语句：

```
thisform.command1.Enabled=.F.
```

设置属性 Enabled 的值为假，从而按钮不可用。另一个值得注意的类似属性是 Visible，用它可以显示或者隐藏命令按钮。隐藏一个命令按钮也包含了使该按钮无效。可以在过程中用语句：

```
thisform.command1.Visible=.F.
```

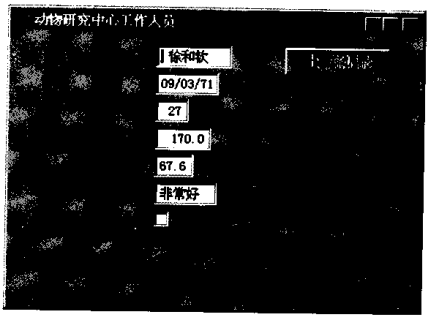


图 6.16　运行有命令按钮的表单

设置按钮的 Visible 属性。

（5）关闭表单。用类似方法，添加以下几个控件，同时设置按钮的属性并给按钮连接相应的代码。

1）添加“下一条记录”按钮控件。该按钮的属性设置为：

```
Top = 54, ;
Left = 252, ;
Height = 25, ;
```

```
Width = 96, ;
Caption = "下一条记录", ;
Name = "Command2"
```

与 Click 事件相连的方法程序为：

```
thisform.command1.enabled=.T.
SELE 工作人员
recall
if(eof())
MessageBox("已经到了文件最后一条记录！",48)
thisform.command2.enabled=.F.
else
skip 1
endif
thisform.refresh
```

这里注意方法程序的第一条语句：

```
thisform.command1.enabled=.T.
```

它在用户单击“下一条记录”按钮时，能使“上一条记录”按钮成为有效。

2）添加“新增加记录”按钮控件。该按钮的属性设置为：

```
Top = 96, ;
Left = 252, ;
Height = 25, ;
Width = 96, ;
Caption = "新增加记录", ;
Name = "Command4"
```

与 Click 事件相连的方法程序为：

```
go bott
append blank
go bott
thisform.refresh
thisform.txt 姓名.setfocus
```

3）添加“删除记录”按钮控件。该按钮的属性设置为：

```
Top = 138, ;
Left = 253, ;
Height = 25, ;
Width = 96, ;
Caption = "\<D 删除记录", ;
Name = "Command4"
```

与 Click 事件相连的方法程序为：

```
if(not eof())
```

```
delete
pack
else
MessageBox("没有记录可删除！",48)
endif
thisform.refresh
```

在 Caption 属性中使用了三个字符“\<D”定义了一个快捷键 D，表示按 D 键将产生与单击该按钮相同的结果。运行时按钮上显示“D 删除记录”。

4）添加标签控件。从工具栏中选取“标签”，然后用鼠标在表单的适当位置拖动形成一定尺寸的标签 Label1，在属性窗口将标签属性设置如下：

```
AutoSize = .T., ;
FontBold = .T., ;
FontItalic = .T., ;
FontSize = 20, ;
FontCondense = .T., ;
FontExtend = .T., ;
BackStyle = 0, ;
ForeColor=RGB(0,128,0),;
Caption = "保护环境", ;
Height = 33, ;
Left = 0, ;
Top = 204, ;
Width = 115, ;
Name = "Label1"
```

用同样的方法，添加标签 Label2，并设置如下属性：

```
AutoSize = .T., ;
FontBold = .T., ;
FontItalic = .T., ;
FontSize = 20, ;
FontCondense = .T., ;
FontExtend = .T., ;
BackStyle = 0, ;
ForeColor=RGB(0,128,0),;
Caption = "匹夫有责", ;
Height = 33, ;
Left = 230, ;
Top = 204, ;
Width = 115, ;
Name = "Label2"
```

5）添加“退出”命令控件。一个友好的界面应使用户操作方便，因此我们用以上添加命令按钮的拖动方法，给表单添加“退出”命令按钮。并在属性窗口将其属性设置如下：

```
Top = 204, ;
Left = 132, ;
Height = 25, ;
Width = 85, ;
Caption = "\<E 退出", ;
Name = "Command1"
```

“退出”按钮与 Click 事件相关联的方法程序：

```
thisform.release
```

（6）保存表单，然后再次执行表单。表单具有了浏览记录的功能，并能够添加和删除记录。如果在数据表中已经没有记录时单击删除按钮，则屏幕上弹出消息框提醒用户没有记录可以删除（如图 6.18）。在显示第一条记录或者最后一条记录时单击“上一条记录”按钮或者“下一条记录”按钮也会出现相似的消息框。消息框中显示的叹号图标是 MessageBox()函数的第二个参数取 48 的结果。

5.添加错误处理程序

为了处理表单执行过程中的错误，可在表单中添加错误处理程序。方法是在表单中没有控件的地方双击鼠标左键打开代码输入窗口，确信选中“Form1”对象和“Error”过程。然后输入如下代码：

```
*************************************************************************
LOCAL lnChoice
#DEFINE CR CHR(13)
DO CASE
```

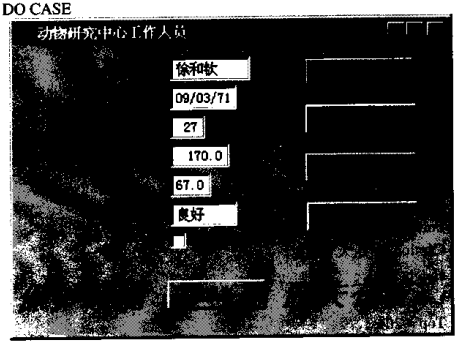


图 6.17 运行表单

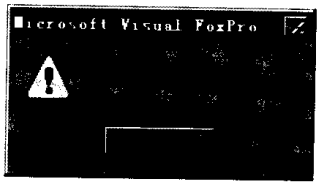


图 6.18　消息框

```
        CASE nError = 1545
            #DEFINE MSG1_LOC "Do you want to save your changes?"
            #DEFINE MSG2_LOC "Uncommitted Changes"
            lnChoice = MESSAGEBOX(MSG1_LOC, 4+48+0, MSG2_LOC)
            DO CASE
                CASE lnChoice = 6 && yes
                    =TABLEUPDATE(.T., .T.)
                CASE lnChoice = 7 && no
                    =TABLEREVERT(.T.)
            ENDCASE
            OTHERWISE   && Unanticipated error
            #DEFINE NUM_LOC "Error Number: "
            #DEFINE PROG_LOC "Program: "
            #DEFINE CAP_LOC "ERROR"
            lcMsg = NUM_LOC + ALLTRIM(STR(nError)) + CR + CR + ;
                    MESSAGE()+ CR + CR + PROG_LOC + PROGRAM(1)
            lnChoice = MESSAGEBOX(lcMsg, 2+48+512, CAP_LOC)
            DO CASE
                CASE lnChoice = 3 &&Abort
                    CANCEL
                CASE lnChoice = 4 &&Retry
                    RETRY
                CASE lnChoice = 5 &&Ignore
                    RETURN
            ENDCASE
    ENDCASE
ENDPROC
*************************************************************************
```

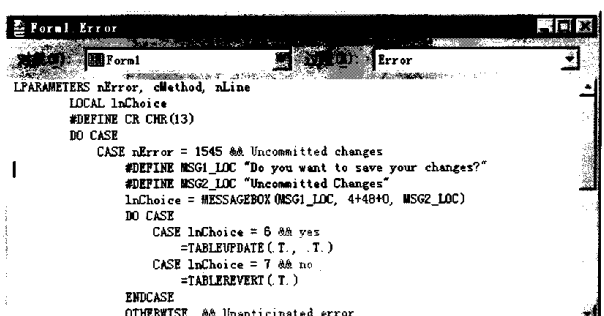


图 6.19 错误处理程序

Error 过程具有参数，Visual FoxPro 按下列顺序向 Error 事件传送三或四个参数。其中参数 nIndex 唯一标识控制数组中的控制，参数 nError 表示 Visual FoxPro 的错误编号，参数 cMethod 存放造成此错误的方法。但是，如果方法调用了用户自定义函数，并且正是在此函数中发生了错误，则 cMethod 包含的是这个用户自定义函数名，而不是调用此函数的方法名。参数 nLine 存放方法中或自定义函数中造成此错误的程序行号。Error 事件使得对象可以对错误进行处理。此事件忽略当前的 ON ERROR 例程，并允许各个对象在内部俘获并处理错误。值得注意的是只有错误发生在代码中时，才调用 Error 事件。如果正在处理错误时，Error 事件过程中又发生了第二个错误，Visual FoxPro 将调用 ON ERROR 例程。如果 ON ERROR 例程不存在，Visual FoxPro 将挂起程序并报告错误，如同 Error 事件和 ON ERROR 例程不存在一样。

6.保存表单类

从文件菜单中选择“另存为类”，然后输入：

类名：RecordForm

文件名：d:\vfwork\学习实践\myclass.vcx

说明：用表单管理数据表记录

单击“确定”按钮将当前表单保存为类。

7.应用表单类

（1）SET CLASSLIB TO 命令，

该命令用来打开包含类定义的 .VCX 可视类库。其用法为：

SET CLASSLIB TO ClassLibraryName [ADDITIVE]

图 6.20　“另存为类”对话框

[ALIAS AliasName]

其中参数 TO ClassLibraryName 指定要打开的 .VCX 可视类库的名称。如果在所给参数 ClassLibraryName 中没有包含完整的路径名，Visual FoxPro 首先在默认的 Visual FoxPro 目录中查找可视类库，然后在 Visual FoxPro 路径下各目录中查找。默认的 Visual FoxPro 目录由 SET DEFAULT 指定，Visual FoxPro 查找路径由 SET PATH 指定。如果不带 ClassLibraryName 执行 SET CLASSLIB TO 命令，则 Visual FoxPro

将关闭所有的可视类库。用 RELEASE CLASSLIB 命令也可以关闭一个可视类库。参数 ADDITIVE 打开 .VCX 可视类库时，不关闭任何当前打开的 .VCX 可视类库。如果省略这一子句，将关闭所有 .VCX 可视类库。参数 ALIAS AliasName 指定可视类库的别名，可以通过它的别名来引用可视类库。如果找不到包含对象的类定义，则 Visual FoxPro 产生错误信息。

（2）CREATEOBJECT() 函数。

该函数从类定义或支持 OLE 的应用程序中创建对象。其用法为：

CREATEOBJECT(ClassName [, eParameter1, eParameter2, ...])

其中参数 ClassName 指定用来创建新对象的类或 OLE 对象。可对 ClassName 使用下面语法创建 OLE 对象：

ApplicationName.Class

例如，若要创建 Microsoft Excel 工作表（支持 OLE 自动化），可用下面的语法：

Sheet1= CREATEOBJECT('Excel.Sheet')

当运行这个代码时，启动 Microsoft Excel（如果还没有运行），并创建一个新的工作表。一个类库有一个别名。若要用别名来指定类库中的对象，可包含类库别名，后接一个句点号和对象名。函数的可选参数 eParameter1, eParameter2, ...用来把值传给类的 Init 事件过程。当发出 CREATEOBJECT( )时执行 Init 事件，并进行对象初始化工作。可用

CREATEOBJECT( ) 从类定义或支持 OLE 自动化的应用程序中创建对象，并将对象引用赋给内存变量或数组元素。由用户自定义类创建对象之前，用户自定义类必须先用 DEFINE CLASS 来创建，或者必须是用 SET CLASSLIB 打开的 .VCX 可视类库中的类。可用 = 或 STORE 来将对象引用赋给内存变量或数组元素。如果释放指定给内存变量或数组元素的对象，则内存变量或数组元素包含 Null 值。可用 RELEASE 从内存中移去内存变量或数组元素。

（3） Show() 方法。

Show() 方法用来显示一个表单，并且确定是模式表单还是无模式表单。其用法为：

[FormSet.]Object.Show([nStyle])

其中参数 nStyle 确定如何显示表单。当 nStyle 值取 1 时显示模式表单。只有隐藏或释放模式表单之后，用户的输入（键盘或鼠标）才能被其他表单或菜单接收。在接收其他用户输入之前，程序必须隐藏或释放模式表单（通常是响应某些用户动作）。当应用程序中的模式表单显示时，虽然其他表单被废止，但是其他应用程序并不被废止。当 nStyle 值取 2（默认值）时为无模式表单。遇到 Show 方法之后出现的代码时，就执行它们。

如果 nStyle 省略，表单按 WindowType 属性指定的样式显示。Show 方法把表单或表单集的 Visible 属性设置为“真”(.T.)，并使表单成为活动的对象。如果表单的 Visible 属性已经设置为“真”(.T.)，则 Show 方法使它成为活动对象。如果激活的是表单集，则表单集中最近一个活动表单成为活动表单；如果没有活动表单，则第一个添加到表单集类定义中的表单成为活动表单。表单集中包含的表单保留 Visible 属性设置。如果表单的 Visible 属性设置为“假”(.F.)，表单集的 Show 方法不显示这个表单。所有表单集中的表单采取表单集的形式。例如，如果表单集为模式表单集，则所有的表单都为模式表单。下列命令打开一个名为 myclass，别名为 MyForm 的可视类库，然后创建一个名为 MyFormOne 的控制。

```
SET CLASSLIB ;
    TO D:\VFWORK\学习实践\myclass.vcx  ;
    ALIAS MyForm
MyFormOne= CREATEOBJ('MyForm.RecordForm')
MyFormOne.show(1)
```

6.2.2 完全通过编程的方法

下例中创建一个表单类，并示例性的创建表单对象说明表单类的具体应用。通过下面程序的实践，相信读者会进一步深刻体会 Visual FoxPro 中面向对象编程的特点。确实，由于程序的对象化和模块化，程序显得简单明了。

```
**********************************************************************
*  程序清单：用表单来管理数据表记录
**********************************************************************
use "d:\vfwork\学习实践\工作人员.dbf"
myrecord = CreateObject("record")
```

```
myrecord.show(1)
*****************************************************************
*-- ParentClass:    form
*-- BaseClass:      form
*-- 动物研究中心工作人员管理表
*****************************************************************
DEFINE CLASS record AS form
    Top = 23
    Left = 57
    DoCreate = .T.
    Caption = "动物研究中心工作人员"
    Name = "Form1"
    ADD OBJECT txt 姓名  AS textbox WITH ;
        ControlSource = "工作人员.姓名", ;
        Height = 20, ;
        Left = 135, ;
        TabIndex = 2, ;
        Top = 12, ;
        Width = 69, ;
        Comment = "", ;
        Name = "txt 姓名"
    ADD OBJECT lbl 姓名  AS label WITH ;
        AutoSize = .T., ;
        BackStyle = 0, ;
        Caption = "姓名", ;
        Left = 77, ;
        Top = 12, ;
        TabIndex = 1, ;
        Name = "lbl 姓名"
    ADD OBJECT txt 生日  AS textbox WITH ;
        ControlSource = "工作人员.生日", ;
        Height = 20, ;
        Left = 135, ;
        TabIndex = 4, ;
        Top = 37, ;
        Width = 57, ;
        Comment = "", ;
        Name = "txt 生日"
    ADD OBJECT lbl 生日  AS label WITH ;
```

```
    AutoSize = .T., ;
    BackStyle = 0, ;
    Caption = "生日", ;
    Left = 77, ;
    Top = 37, ;
    TabIndex = 3, ;
    Name = "lbl 生日"
ADD OBJECT txt 年龄 AS textbox WITH ;
    ControlSource = "工作人员.年龄", ;
    Height = 20, ;
    Left = 135, ;
    TabIndex = 6, ;
    Top = 62, ;
    Width = 30, ;
    Comment = "", ;
    Name = "txt 年龄"
ADD OBJECT lbl 年龄 AS label WITH ;
    AutoSize = .T., ;
    BackStyle = 0, ;
    Caption = "年龄", ;
    Left = 77, ;
    Top = 62, ;
    TabIndex = 5, ;
    Name = "lbl 年龄"
ADD OBJECT txt 身高 AS textbox WITH ;
    ControlSource = "工作人员.身高", ;
    Height = 20, ;
    Left = 135, ;
    TabIndex = 8, ;
    Top = 87, ;
    Width = 51, ;
    Comment = "", ;
    Name = "txt 身高"
ADD OBJECT lbl 身高 AS label WITH ;
    AutoSize = .T., ;
    BackStyle = 0, ;
    Caption = "身高", ;
    Left = 77, ;
    Top = 87, ;
```

```
        TabIndex = 7, ;
        Name = "lbl 身高"
    ADD OBJECT txt 体重 AS textbox WITH ;
        ControlSource = "工作人员.体重", ;
        Height = 20, ;
        Left = 135, ;
        TabIndex = 10, ;
        Top = 112, ;
        Width = 35, ;
        Comment = "", ;
        Name = "txt 体重"
    ADD OBJECT lbl 体重 AS label WITH ;
        AutoSize = .T., ;
        BackStyle = 0, ;
        Caption = "体重", ;
        Left = 77, ;
        Top = 112, ;
        TabIndex = 9, ;
        Name = "lbl 体重"
    ADD OBJECT txt 健康状况 AS textbox WITH ;
        ControlSource = "工作人员.健康状况", ;
        Height = 20, ;
        Left = 135, ;
        TabIndex = 12, ;
        Top = 137, ;
        Width = 57, ;
        Comment = "", ;
        Name = "txt 健康状况"
    ADD OBJECT lbl 健康状况 AS label WITH ;
        AutoSize = .T., ;
        BackStyle = 0, ;
        Caption = "健康状况", ;
        Left = 77, ;
        Top = 137, ;
        TabIndex = 11, ;
        Name = "lbl 健康状况"
    ADD OBJECT chk 婚姻状况 AS checkbox WITH ;
        Top = 162, ;
        Left = 135, ;
```

```
    Height = 16, ;
    Width = 69, ;
    Caption = "婚姻状况", ;
    ControlSource = "工作人员.婚姻状况", ;
    Comment = "", ;
    Name = "chk 婚姻状况"
ADD OBJECT command5 AS commandbutton WITH ;
    Top = 204, ;
    Left = 132, ;
    Height = 25, ;
    Width = 85, ;
    Caption = "\<E 退出", ;
    Name = "Command5"
ADD OBJECT label1 AS label WITH ;
    AutoSize = .T., ;
    FontBold = .T., ;
    FontItalic = .T., ;
    FontSize = 20, ;
    FontCondense = .T., ;
    FontExtend = .T., ;
    BackStyle = 0, ;
    ForeColor=RGB(0,128,0),;
    Caption = "保护环境", ;
    Height = 33, ;
    Left = 0, ;
    Top = 204, ;
    Width = 115, ;
    Name = "Label1"
ADD OBJECT label2 AS label WITH ;
    AutoSize = .T., ;
    FontBold = .T., ;
    FontItalic = .T., ;
    FontSize = 20, ;
    FontCondense = .T., ;
    FontExtend = .T., ;
    BackStyle = 0, ;
    ForeColor=RGB(0,128,0),;
    Caption = "匹夫有责", ;
    Height = 33, ;
```

```
        Left = 230, ;
        Top = 204, ;
        Width = 115, ;
        Name = "Label2"
    ADD OBJECT command1 AS commandbutton WITH ;
        Top = 13, ;
        Left = 251, ;
        Height = 25, ;
        Width = 96, ;
        Caption = "上一条记录", ;
        MousePointer = 1, ;
        Name = "Command1"
    ADD OBJECT command2 AS commandbutton WITH ;
        Top = 54, ;
        Left = 252, ;
        Height = 25, ;
        Width = 96, ;
        Caption = "下一条记录", ;
        Name = "Command2"
    ADD OBJECT command3 AS commandbutton WITH ;
        Top = 96, ;
        Left = 252, ;
        Height = 25, ;
        Width = 96, ;
        Caption = "新增加记录", ;
        Name = "Command3"
    ADD OBJECT command4 AS commandbutton WITH ;
        Top = 138, ;
        Left = 253, ;
        Height = 25, ;
        Width = 96, ;
        Caption = "\<D 删除记录", ;
        Name = "Command4"
**************************************************************************
    PROCEDURE command5.Click
        ThisForm.Release
    ENDPROC
    PROCEDURE command1.Click
        thisform.command2.enabled=.T.
```

```
    SELE 工作人员
    recall
    if(bof())
    MessageBox("已经到了文件第一条记录！",48)
    thisform.command1.enabled=.F.
    else
    skip -1
    endif
    thisform.refresh
ENDPROC
PROCEDURE command2.Click
    thisform.command1.enabled=.T.
    SELE 工作人员
    recall
    if(eof())
    MessageBox("已经到了文件最后一条记录！",48)
    thisform.command2.enabled=.F.
    else
    skip 1
    endif
    thisform.refresh
ENDPROC
PROCEDURE command3.Click
    go bott
    append blank
    go bott
    thisform.refresh
    thisform.txt 姓名.setfocus
ENDPROC
PROCEDURE command4.Click
    if(not eof())
    delete
    pack
    else
    MessageBox("没有记录可删除！",48)
    endif
    thisform.refresh
ENDPROC
PROCEDURE Error
```

```
        LPARAMETERS nError, cMethod, nLine
        LOCAL lnChoice
        #DEFINE CR CHR(13)
        DO CASE
            CASE nError = 1545 && Uncommitted changes
            *-----------------------------------
                #DEFINE MSG1_LOC "Do you want to save your changes?"
                #DEFINE MSG2_LOC "Uncommitted Changes"
                lnChoice = MESSAGEBOX(MSG1_LOC, 4+48+0, MSG2_LOC)
                DO CASE
                    CASE lnChoice = 6 && yes
                        =TABLEUPDATE(.T., .T.)
                    CASE lnChoice = 7 && no
                        =TABLEREVERT(.T.)
                ENDCASE
                OTHERWISE   && Unanticipated error
            *-----------------------------------
                #DEFINE NUM_LOC "Error Number: "
                #DEFINE PROG_LOC "Program: "
                #DEFINE CAP_LOC "ERROR"
                lcMsg = NUM_LOC + ALLTRIM(STR(nError)) + CR + CR + ;
                        MESSAGE()+ CR + CR + PROG_LOC + PROGRAM(1)
                lnChoice = MESSAGEBOX(lcMsg, 2+48+512, CAP_LOC)
                DO CASE
                    CASE lnChoice = 3 &&Abort
                        CANCEL
                    CASE lnChoice = 4 &&Retry
                        RETRY
                    CASE lnChoice = 5 &&Ignore
                        RETURN
                ENDCASE
        ENDCASE
    ENDPROC
ENDDEFINE
*-- EndDefine: record
**********************************************************************
```

6.3 不可视类

基于自定义类的子类没有可视元件，这样的类称为不可视类。不可视类完全由属性和方法组成，用 Custom 基类可创建不可视对象。

创建一个用户自定义类或子类，并为创建的类或子类指定属性、事件和方法。可使用如下语句：

```
DEFINE CLASS ClassName1 AS ParentClass
    [[PROTECTED PropertyName1, PropertyName2 ...]
        PropertyName = eExpression ...]
    [ADD OBJECT [PROTECTED] ObjectName AS ClassName2 [NOINIT]
        [WITH cPropertylist]]...
    [[PROTECTED] FUNCTION | PROCEDURE Name
        [NODEFAULT]
        cStatements
    [ENDFUNC | ENDPROC]]...
ENDDEFINE
```

其中参数 ClassName1 指定要创建的类名，参数 AS ParentClass 指定派生类或子类的父类，父类可以是一个 Visual FoxPro 基类（例如 Form 类）或者是另一个用户自定义的类或子类。通过指定 ParentClass 为 Custom，可以创建一个非可视的用户自定义类。在

```
[PROTECTED PropertyName1, PropertyName2 ...]
    PropertyName = eExpression ...
```

中用 PropertyName = eExpression 创建一个类属性或子类属性，并给属性赋默认值。这些属性是类的命名属性，它们为类定义了特性和行为。类和子类可以有多个属性。可使用“=”为属性赋值。下面示例创建了一个名为“书店”的用户自定义类，并创建了相关属性，并将属性初始化为相应的值。

```
DEFINE CLASS 书店 AS CUSTOM
  编号=11101
  书店名="新华书店"
  地区="北京"
  出版社="水利水电出版社"
  书名="《Visual FoxPro 5.0 编程宝典》"
  出版日期="03/20/98"
  PROTECT 定价=45.7
  份数=0
  折价率=1.0
  批发=.T.
  总金额=0
  InvValue=0
```

当用 CREATEOBJECT( ) 函数创建对象后，可以在类或子类的定义之外访问属性，访问属性的语法如下：

ObjectName.Property

例如创建名为 shop1 的对象，可用下面的语句：

```
shop1=CREATEOBJECT("书店")
```

要访问“书店”类的属性，可通过下述语句实现：

```
shop1.编号=11456
shop1.书店名="北图"
shop1.份数=265
shop1.批发=.T.
shop1.计算总金额
shop1.打印售书清单
```

子句 ADD OBJECT 从 Visual FoxPro 的一个基类、用户自定义类（子类）或从 OLE 自定义控制向一个类定义或子类定义中添加一个对象。参数 PROTECTED 禁止从类定义或子类定义的外部访问和修改对象的属性，PROTECTED 关键字必须直接放在 ObjectName 之前，否则 FoxPro 将产生语法错误。参数 ObjectName 在类定义或子类定义创建一个对象后指定对象名，并用来从类定义或子类定义内部引用对象。参数 AS ClassName2 指定类或子类名，该类或子类中包含添加到类定义中的对象。参数 NOINIT 说明当添加对象时，不执行对象的 Init 方法。WITH cPropertyList 为添加到类定义或子类定义中的对象指定一列属性和属性值。

用户自定义类是放在程序文件中的一组命令，类似一个过程。与过程一样，这些类定义或子类定义后的命令并不执行，它们只定义类或子类的属性、事件和方法。要根据一个类定义或子类定义创建对象，应发出包含该类名或子类名的 CREATEOBJECT( )。不能在结构化编程命令中使用 DEFINE CLASS 创建类定义或子类定义（例如 IF … ENDIF 或 DO CASE … ENDCASE），它们也不能放在循环语句中（例如 DO WHILE … ENDDO 或 FOR … ENDFOR）。

下面先创建类名为“书店”的不可视类，在类的属性定义中包含了有关书销售的一些基本信息，在类的方法定义中计算了销售金额和输出了销售信息的清单。创建的对象 shop1 是对自定义类的一个应用。读完该例之后，相信读者就逐渐熟悉 Visual FoxPro 这样一种从类到对象的面向对象的编程方法了。

```
* 创建不可视类实例程序清单
shop1=CREATEOBJECT("书店")
Clear
shop1.编号=11456
shop1.书店名="北图"
shop1.份数=265
shop1.批发=.T.
shop1.计算总金额
shop1.打印售书清单
```

```
DEFINE CLASS 书店 AS CUSTOM
  编号=11101
  书店名="新华书店"
  地区="北京"
  出版社="水利水电出版社"
  书名="《Visual FoxPro 5.0 编程宝典》"
  出版日期="03/20/98"
  定价=45.7
  份数=0
  折价率=1.0
  批发=.T.
  总金额=0
  InvValue=0
PROCEDURE 计算总金额
    if(This.批发)
    This.折价率=0.75
    else
    This.折价率=0.95
    endif
    This.总金额=This.定价*This.份数*This.折价率
  ENDPROC
```

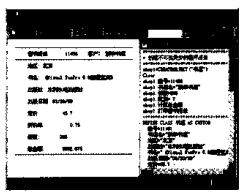


图 6.21 运行自定义类的一个实例

```
PROCEDURE 打印售书清单
  ?"  ----------------------------------------------"
  ? "  售书清单:", this.编号          ,"    客户: ",this.书店名
  ?"  ----------------------------------------------"
  ?"  地区:",this.地区
  ?
  ?"  书名:",this.书名
  ?
```

```
    ?"  出版社:",this.出版社
    ?
    ?"  出版日期:",this.出版日期
    ?
    ?"  定价:",this.定价
    ?
    ?"  折价率:",this.折价率
    ?
    ?"  份数:",this.份数
    ?
    ?"  总金额:",this.总金额
    ?"  ----------------------------------------"
ENDPROC
ENDDEFINE
```

6.4　Visual FoxPro 类的层次

前两节分别通过程序实例针对可视类和不可视类的特点和编程方法做了介绍，其中已经或多或少的用到了有关类层次的概念。前面已经提到，表单是一种容器类。当我们在表单中添加控件时，就产生了一种层次结构。

1.构造 Visual FoxPro 类的层次结构

在 Visual FoxPro 中，用 AddObject()方法程序可以在运行时，给容器对象中添加对象。其用法为：

Object.AddObject(cName, cClass [, cOLEClass] [, aInit1, aInit2 ...])

其中参数 cName 指定引用新对象的名称，参数 cClass 指定添加对象所在的类，参数 cOLEClass 指定添加对象的 OLE 类。用 aInit1, aInit2 ...还可以指定传给新对象的 Init 事件的参数。调用 AddObject 方法时，将触发新添加对象的 Init 事件。在表单集中加入表单时，Load 事件在 Init 事件之前发生。当用 AddObject 方法往容器中加入对象时，对象的 Visible 属性设置为“假”(.F.)。因此可以设置对象的属性，而不看更改对象外观时的一些中间效果。

下面语句创建了表单并在表单容器中加入了命令按钮控件。

```
myForm=CREATEOBJECT("FORM")
myForm.Caption="带退出命令按钮的表单"
myForm.ScaleMode=0
myForm.Top=3
myForm.Left=5
myForm.Width=40
myForm.Height=10
```

```
myForm.AddObject("MyExit","CommandButton")
exitButton=myForm.MyExit
exitButton.Top=7
exitButton.Left=15
exitButton.Width=10
exitButton.Height=2
exitButton.Visible=.T.
myForm.Show(1)
```

如图 6.22 是以上语句定义的结果。这里形成了一种从表单到命令按钮的层次结构。我们已经知道，引用控件的属性和方法等的语句必须包括控件所在的容器对象，因此知道控件的层次结构是非常重要的。

2.引用容器对象中控件

我们在引用控件时，可以使用下面的记法操作表单的控件属性：

myForm.MyExit.Caption="退出"

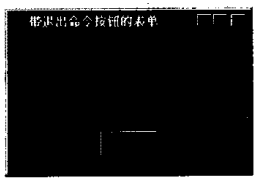


图 6.22 执行表单

但有时这种记法需要输入的内容较长，因此还可以使用其他方法。

（1）This 对象引用。

This 对象引用可在创建对象之前提供对对象的引用。其用法为： THIS.PropertyName | ObjectName 。

THIS 允许在一个类定义中引用属性或对象。类定义块中的方法可以使用 THIS 来指定类创建时存在的属性或对象。因为对象的多个实例共享同样的方法代码，因此 THIS 总是指代码正在执行的实例。如果存在一个对象的多个实例，并且调用了对象的方法， THIS 会引用正确的对象。

（2）THISFORM 对象引用。

ThisForm 在创建之前，提供对表单的引用。其用法为： THISFORM.PropertyName | ObjectName 。

THISFORM 提供了在方法中对对象所在表单或表单属性的引用。THISFORM 允许引用表单上的对象或表单的属性，而不需要使用多个 Parent 属性。

（3）Parent 属性。

用该属性可引用一个控制的容器对象。用法为：Control.Parent 。

使用 Parent 属性，可以访问一个控制所属的容器对象的属性、方法或控制。还可以使用 Parent 属性访问一个页面或表单的容器对象。在应用程序中，当把控制作为参数传递时，Parent 属性很有用。例如，可以把一个控制变量传递给一个通用过程并且应用 Parent 属性访问此控制的容器对象。如果一个控制类中的对象被放置在未知的表单对象上，也可以用 Parent 属性来引用。

6.5　类库操作入门

我们已经体会到，面向对象的程序设计方法与编程技术不同于标准的过程化程序设计。程序设计人员在进行面向对象的程序设计时，不再是单纯地从代码的第一行一直编到最后一行，而是考虑如何创建对象，利用对象来简化程序设计，提供代码的可重用性。对象可以是应用程序的一个自包含组件，一方面具有私有的功能，供自己使用；另一方面又提供公用的功能，供其他用户使用。在这样一种编程方法中，类库是一个非常强有力的助手。Visual FoxPro 5.0 中可以使用“类设计器”可视化地创建并修改类。创建完了类后，可以用“类浏览器”显示类库。“类浏览器”还能浏览表单中的类，也能够显示.TLB 、.OLB 或.EXE 文件中的类型库信息。可用“类浏览器”显示类库或表单中的表，以及查看、使用和管理类及其用户定义成员。

6.5.1　使用类设计器

类设计器可以设计和修改类，并将类存于指定的类库中。使用类设计器创建类库的方法如下：

（1）在命令窗口中输入命令 CREATE CLASS 。

可打开创建新类的对话框，如图 6.23 。

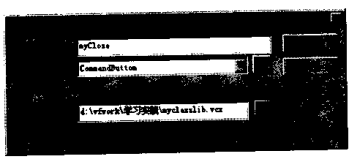


图 6.23　创建新类

（2）比如要创建一个类名为“myClose”的关闭按钮类，并将该类存于名为“d:\vfwork\学习实践\myClassLib.vcx”的可视类库文件中，就在“类名”处输入“myClose”，在“存储于”处输入“d:\vfwork\学习实践\myClassLib.vcx”并确定。这样就打开了类设计器，如图 6.24 。此时，从主菜单中选择“显示”菜单，可以看到该菜单增添了一些与设计类相关的菜单选项。

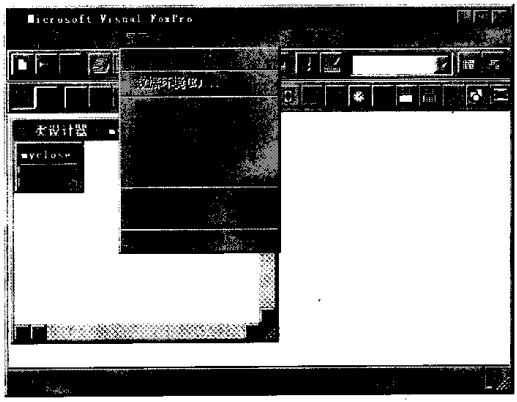


图 6.24 类设计器

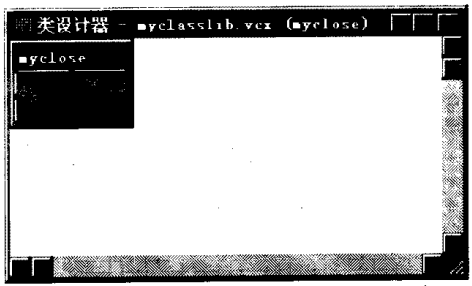


图 6.25 用类设计器设计类

（3）设计 myClose 关闭按钮类的属性。用鼠标右键单击类设计器中的命令按钮弹出快捷菜单，从中选择“属性”项可打开属性窗口，也可以从“显示”菜单中选择“属性”项打开属性窗口。在属性窗口中设置如下属性：

```
Caption="\<C 关 闭"
Picture=d:\vfwork\学习实践\control.bmp
FontSize=12
```

ForeColor=RGB(0,128,0)

其中 Picture 属性可根据喜好随意指定一个位图文件（.BMP）。

（4）给 myClose 关闭按钮类添加方法。从“显示”菜单中选择“代码”，或者用鼠标右键单击命令按钮，从快捷菜单中选择“代码”，或者直接用鼠标左键双击命令按钮都可以打开代码窗口。确信选择了 myclose 对象的 Click 事件。在代码窗口中加入如下代码：

```
ThisForm.Release
```

然后关闭代码窗口。

（5）关闭类设计器，并注意保存设计的类库文件。

（6）用相似的方法，从 Label 类派生一个名为“ mySlogan ”的类保存到同一个库文件“ d:\vfwork\学习实践\myClassLib.vcx ”中。“ mySlogan ”类的属性设置如下：

```
Alignment=2
AutoSize=.T.
Caption="保护环境，人人有责"
FontItalic=.T.
FontName="宋体"
FontSize=20
ForeColor=RGB(0,128,0)
```

（7）若要修改一个类库中的类定义，可以在命令按钮中输入如下命令：

```
MODIFY CLASS
```

然后从弹出的文件对话框（如图 6.26）中选择所需的可视库文件。比如要修改类库文件“ d:\vfwork\学习实践\myClassLib.vcx ”中类 mySlogan 的定义，则选择该文件并从类名中选择“ mySlogan ”。按“打开”按钮后打开类设计器，就可以修改类定义了。

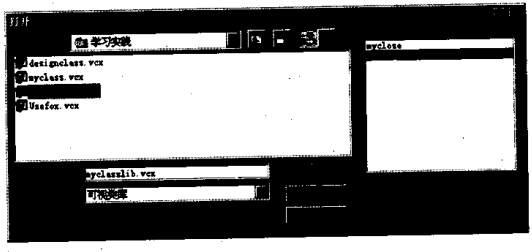


图 6.26　打开可视类库中的类

6.5.2　使用类浏览器

上节介绍了类设计器的使用方法并创建了自己的类库。当需要管理和使用类库时，我

们应当使用类浏览器。在命令窗口中输入如下命令：

```
DO BROWSER
```

可以打开类浏览器。也可以从“工具”菜单中选择“类浏览器”打开。在发出命令或者选中菜单选项后，屏幕上弹出“文件”对话框，从中可以选择所需的类库文件。比如选择上节创建的类库文件“ d:\vfwork\学习实践\myClassLib.vcx ”，则打开的类浏览器如图 6.27 。在命令窗口中输入如下命令可改变类浏览器的属性：

```
_OBROWSER.Caption = "我的类浏览器"
_OBROWSER.Left = 20
```

可在运行时改变类浏览器的属性。当然，也可以改变其他属性设置。类浏览器中包括以下内容：

（1）“类浏览器”按钮。执行各种“类浏览器”命令。将鼠标移到按钮上可以显示按钮名，利用这些按钮可以创建新类、给类重命名或者重定义、删除选中的类等等。“类浏览器”快捷菜单可以提供附加的功能。

（2）类型框。允许选定或输入一个类的类型或字符串来筛选类列表。下拉列表显示基类，并保留类的类型历史记录，同时为“类浏览器”的当前实例筛选所选定的或输入的内容。

（3）类图标。显示代表选定类的图标。可以将该图标拖动到 Visual FoxPro 主窗口，来创建类的一个实例。一个用户定义的位图可以作为类图标。

（4）类列表。按大纲形式列出包含在类库（.VCX）或表单（.SCX）中的类及子类，文件夹图标出现在每个类和子类旁。类旁边的标识 (<<) 表示其父类位于当前“类列表”中未显示的文件之中。要查看父类，选择快捷菜单上的“选择父类”；要修改“类设计器”中的某个类，请双击类名。

（5）成员列表。列出了从“类”列表中选中的类或表单的用户自定义属性和方法程序。从“成员”列表快捷菜单中选择选项，可以筛选这个列表。

（6）“保护”过滤器。在“保护”过滤器快捷菜单项的旁边如果显示了复选标记，则在“成员”列表中将显示被保护的成员。被保护的成员用星号 (*) 标识。

（7）“隐藏”过滤器。在“隐藏”过滤器快捷菜单项的旁边如果显示了复选标记，则在“成员”列表中将显示被隐藏的成员。被隐藏的成员用尖顶号 (^)标识。

（8）“空”过滤器。在“空”过滤器快捷菜单项的旁边如果显示了复选标记，则在“成员”列表中将显示空的方法程序。空的方法程序用浪线号 (~) 标识。

（9）类说明框。显示对选定类的说明，该框位于“类浏览器”窗口的左下部，可以在框中对说明进行编辑。

（10）成员说明框。显示类中所选定成员的信息。该框位于“类浏览器”窗口的右下部。对于对象成员，该框为类及基类的只读显示。对于属性或方法程序成员，该框显示可以进行编辑的说明。例如，说明框显示一个只读说明，说明包括变量范围（公共的或隐藏的）、成员名和属性值等等。

下面具体说明如何使用类浏览器。

例一、使用类浏览器将类库中定义的控件添加到表单中。

上节我们创建了一个类库“ d:\vfwork\学习实践\myClassLib.vcx ”，其中定义了类

myClose 和类 mySlogan 。现在来具体运用这些类。

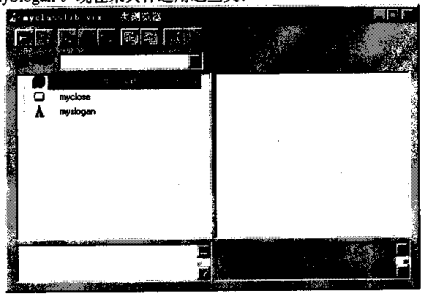


图 6.27　类浏览器

（1）打开类浏览器，并选择浏览文件为“ d:\vfwork\学习实践\myClassLib.vcx ”。

（2）从“文件”菜单中选择“新建”，然后选择新建表单文件，打开表单设计器。拖动表单形成适当的大小，然后在属性窗口中将 Caption 属性设置为“使用类浏览器添加表单控件实例”。

（3）在类浏览器中选择 myClose 类，使对应图标显示在类型框左旁。

（4）将该图标拖放到表单中的适当位置。

（5）用同样的方法将 mySlogan 类图标拖放到表单中，如图 6.28 。

（6）执行表单，注意将刚才的操作保存到文件“表单.scx ”。用鼠标右键单击表单并从弹出的快捷菜单中选择“执行表单”。运行时的表单如图 6.29 。

（7）单击“关闭”按钮可关闭表单，回到表单设计器中。可以看出，在没有写任何代码的情况下也能完成一定的操作（比如关闭表单）。

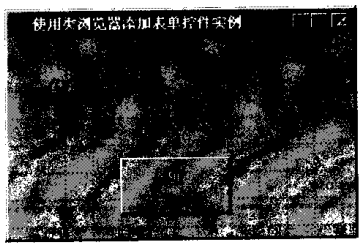


图 6.28　使用类浏览器将类库中定义的控件添加到表单中

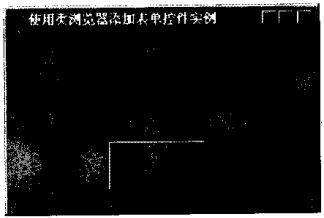


图 6.29 运行时的表单

（8）在表单设计器中双击“关闭”按钮打开代码窗口，可以发现，命令按钮的 Click 方法程序中没有任何代码！

（9）为了深刻地体会这之间的奥妙，我们重新回到类浏览器窗口。在选中 myClose 类后单击“查看类代码”按钮（类浏览器按钮中左起第三个）打开查看类代码的窗口。注意在查看类代码窗口中不能修改类定义。myClose 类的定义如下：

```
**********************************************************************
*-- Class:          myclose (d:\vfwork\学习实践\myclasslib.vcx)
*-- ParentClass:   commandbutton
*-- BaseClass:     commandbutton
**********************************************************************
DEFINE CLASS myclose AS commandbutton
    AutoSize = .F.
    Height = 38
    Width = 96
    FontSize = 12
    Picture = "control.bmp"
    Caption = "\<C 关 闭"
    ForeColor = RGB(0,128,0)
    Name = "myclose"
    PROCEDURE Click
        ThisForm.Release
    ENDPROC
ENDDEFINE
*-- EndDefine: myclose
**********************************************************************
```

（10）进一步理解继承性的概念。保存表单后关闭表单设计器，然后在类浏览器窗口中双击类列表中 myClose 类的图标，打开类设计器，在属性窗口中将“关闭”按钮的 Picture 属性改为默认值（无）。如果不关闭表单设计器，系统将不允许修改类定义。修改完后存

盘并关闭类设计器。重新打开表单并运行，发现在类定义中所做的修改被类的对象继承下来。

例二、创建和派生新类，在类库间移动类定义。

（1）要创建新类，可在类浏览器中选中类库文件名后单击“新类”按钮。

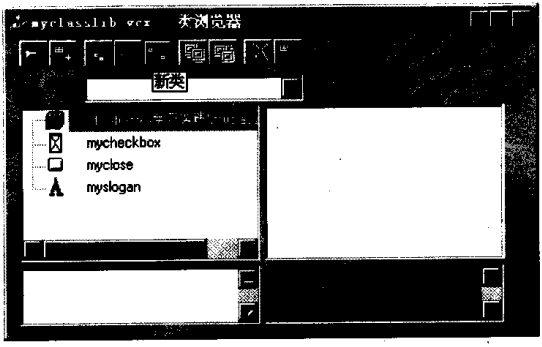


图 6.30　在类浏览器中创建新类

添加新类 myCheckBox 后进入类设计器，我们在类设计器中定义该类的属性：

```
Caption="显示标语"
Value=.T.
...
```

并将 Click 事件的方法定义为：

```
ThisForm.MySlogan1.Visible = This.Value
```

然后保存定义，关闭类设计器回到类浏览器。用前面说过的方法可以使用该类定义：打开表单“表单.scx”，将类浏览器中 myCheckBox 类的图标拖放到表单中的适当位置（如图 6.31）。执行该表单可以发现，单击“显示标语”复选框显示或者隐藏标语。

（2）派生新类。比如要在 myClose 类下派生 myExit 子类，可以在类表中选中 myClose 类后单击“新类”按钮，然后输入子类名。打开类设计器设计子类，将 myExit 类的 Caption 属性设置为：

```
Caption="\<C 退 出"
```

然后保存定义。读者可自己练习将该类用在前面设计的表单中代替 myClose 类。回到类浏览器中，可以发现 myExit 类是 myClose 类的子类。

（3）在类库间移动类。为了便于讲解，我们用前面介绍过的方法创建一个新的类库文件，读者可随意加入一个类。比如，又设计了一个新的类库文件“designclass.vcx”，

下面要做的是将“ myClassLib.vcx ”中的类 myClose 移到类库“ designclass.vcx ”中。具体操作方法为：

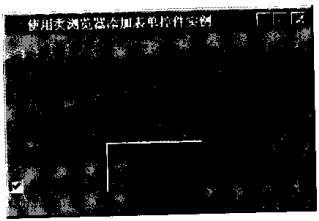


图 6.31 运行有“显示标语”复选框的表单

1）从“工具”菜单中选择“类浏览器”，打开“ myClassLib.vcx ”文件。

2）用同样的方法打开“ designclass.vcx ”文件，这时屏幕上出现两个浏览窗口（注意不能用浏览器中的“打开”按钮打开多个类浏览器）。

3）选中 myClose 类，将类列表上方的类图标拖到“ designclass.vcx ”中放置。这样就完成了类的移动工作。如图 6.33，当 myClose 类被移走后， myExit 类前出现了“<”符号，表示其父类在别的文件中。此时，在“类浏览器”右下方的成员说明框中显示：

类: myexit
父类: myclose (d:\vfwork\学习实践\designclass.vcx)
基类: CommandButton
时间戳: 03/07/98 12:00:00 PM

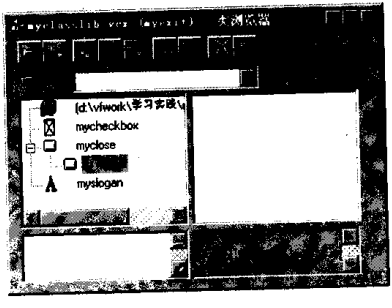


图 6.32 用类浏览器派生子类

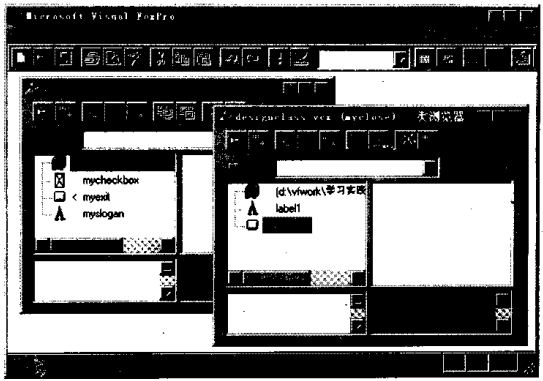


图 6.33　使用类浏览器在类库间移动类

这一节只是简单的介绍了类设计器和类浏览器的使用，相信随着学习的逐步深入，涉及的程序规模更大，您会发现这些工具使原来令人烦恼的工作变得轻松多了。

第七章　用面向对象的方法快速创建应用程序

随着面向对象语言的产生和运用，面向对象的编程方法已经深入到了每个程序员的工作当中，尤其是对于 Windows 应用程序的开发，面向对象的方法更是得到了广大用户的关注。在 Windows 应用程序的开发当中，关于界面的代码大约要占源代码的 40%左右，而面向对象的方法可以快速地使您掌握关于友好用户界面的开发。从而节省了您宝贵的工作时间，也提高了工作效率。

7.1　开发应用程序的过程

基于面向对象方法来开发应用程序的过程比较简单。简单地说，就是从基类中派生出自己的类，利用自己派生类中的添加的方法和事件来完成对于不同事件的响应。

7.1.1　需求分析

在设计数据库程序之前，认真地规划数据库可以节省时间、精力和资金。所以在设计数据库之前一定要进行需求分析。在规划阶段，应该让用户更多地参与进来。无论多么仔细的规划，在项目实施过程中也需要不断修改，并听取最终用户的反馈信息。

在开发之前所做的设计方案往往会对最终结果产生很大的影响。许多问题都应在深入开发之前加以考虑，例如：这个应用程序面向的用户是谁，用户的主要操作是什么，其处理的数据集合有多大，是否需要使用后台数据服务器，以及是单用户还是网络上的多用户等等。对于一般的用户操作，可能需要处理顾客、订单或各种产品，他们处理信息的方式将决定应用程序如何进行数据操作。例如，象\VFP\SAMPLES\TASTRADE.APP 程序所使用的订单输入表单也许对一些应用程序有用，但这个表单可能并不适用于管理货物清单或追踪销售记录。

在数据库的分析中，包括有三个部分：初始调查、可行性分析和需求分析。在这里，在这里将讨论需求分析阶段的目标。

需求分析阶段的目标是确定申请系统的功能和物理需求。需求分析阶段分成 11 个主要任务。

(1)　审查可行性研究数据。

该任务的目标是审查档案室保存的可行性研究数据。

(2)　实施需求定义座谈。

该任务的目标是搜集数据，该数据是对可行性研究期间搜集数据的扩充。

(3)　定义面向对象的系统需求。

该任务的目标是定义由系统执行的功能。

(4)　确定对象。

该任务的目标是确定应用情况中指定的每个对象。

(5)　定义客户机和服务器需求（如果有网络的话）。

该任务的目标是定义建立客户机/服务器环境的各种需求，这种环境将支持该系统的功能需求和体系结构系统。

(6)　定义信息网络服务需求。

该任务的目标是定义信息网络的服务级需求。

(7)　定义实现需求。

该任务的目标是定义在实现申请系统中必须满足的物理需求。

(8)　鉴定可供选择的各个方案。

该任务的目标是鉴定每个实际的设计系统可供选择的方案。

(9)　建立评价可供选择方案的标准。

该任务的目标是建立评价和选择从计算机硬件厂商和第三方软件公司出租或购买的软件包的标准。

(10)　构造需求模型。

该任务的目标是构造模型，该模型能从需求定义期间执行任务的结果中推断各种需求。该模型应考虑面向对象和客户机/服务器计算的各种需求。

(11)　把需求定义文献上交。

该任务的目标是把从需求分析期间执行任务获得的数据上交，以便检索和维护。

7.1.2　数据库组织

数据库的组织包括两个方面：外部设计和内部设计。外部设计是由需求模型向一组面向用户的设计面向的过渡。其中主要包括了9个任务。

(1)　审查需求定义的数据。

该任务的目标是审查保存的需求定义数据。

(2)　定义子系统。

设计子系统并给每个子系统分配任务和对象。该任务又包括：

1）定义接口对象功能

2）定义实体对象功能

3）定义控制对象功能

4）定义功能子系统

(3)　定义客户机/服务器功能并给处理器分配任务。

该任务的目标是定义客户机/服务器并给处理器分配任务。该任务分为：

1）确定处理方式（基于主机，客户机还是协同处理）

2）分配由客户机执行的功能

3）分配由服务器执行的功能

(4)　定义子系统接口。

该任务的目标是定义将出现通信的独立系统或模块之间的公共界面。

(5)　定义系统的各种安全控制。

该任务目标是定义各种系统控制。

(6) 定义各种设计约束。

该任务的目标是定义各种可能以下系统设计的物理约束、部门约束和工作约束。

(7) 重复需求设计模型。

该任务的目的是重复每一种需求和设计过程以便把分析和设计过程期间获得的理解溶入到系统模型中。

(8) 评价可供选择的各种设计方案。

(9) 确定初步设计模型。

而内部设计建立系统实现和测试框架。内部设计包括 8 个部分:

(1) 设计系统结构。

该任务的目标是设计系统设计中必须结合在一起的各个技术部分。

(2) 完成对象设计。

该任务的目标是根据实际的实现环境完成对象设计。

(3) 确定数据结构。

该任务的目标是确定数据说明以便给系统提供输入和输出。该任务包括:

1) 确定事务处理格式。

2) 确定显式格式。

3) 确定报告格式。

4) 确定输出格式。

(4) 设计数据库。

该任务的目标是设计数据库,数据库是实现系统设计所必需的。该任务包括:

1) 把若干简单的对象转化为数据库设计。

2) 把若干组合对象转变为数据库设计。

3) 把若干复合对象转变为数据库设计。

4) 把若干关联对象转变为数据库设计。

5) 把若干混合对象转变为数据库设计。

(5) 确定各种人工控制过程。

该任务的目标是确定与系统设计有关的特殊考虑。

(6) 确定各种人工控制过程。

该任务的目标是确定系统的各个控制过程。该任务包括:

1) 确定 I/O 控制过程。

2) 确定用户控制过程。

3) 确定访问控制过程。

4) 确定文件维护过程。

5) 确定恢复控制过程。

6) 准备过程手册。

(7) 确定实现和测试方案。

(8) 构造详细的设计模型。

7.2　创建应用程序的骨架——事件循环

在 Windows 下的应用程序的设计思想和以前的自顶向下的编程思路已经完全不同了。从以往 DOS 下基于顺序执行的单任务系统编程方法，到现在基于 Windows 桌面多任务系统的消息响应编程方法，其中的改变是观念上的。

Windows 系统下的程序都是根据不同窗体的事件响应函数来执行用户的请求，并通过桌面来反应给用户。用户的任何动作都会有相应的响应程序来对应。例如：当用户的鼠标在窗体上移动，则窗体的 MouseMove 事件响应程序将响应该事件。这里讲到的是对象的事件响应函数，而 Windows 系统还可以使用菜单系统来要求系统对他的命令作出响应。下面，我们将详细地讲到菜单系统的创建。

7.2.1　菜单系统

菜单系统对任何一个 Windows 应用程序来说，都是很重要的一部分，在 Windows 应用程序中，用户有很多的事件是通过菜单来向应用程序发起的。例如，我们经常用的文件打开菜单。所以，对应 Windows 应用程序的编程人员，都会设计出一个良好的、满足用户要求的菜单系统。

菜单为用户提供了一个结构化的、可访问的途径，便于使用应用程序中的命令和工具。适当的计划并设计菜单和工具栏，将使应用程序的主要功能得以体现，用户也不会在使用应用程序时受挫。在 Visaul FoxPro 中菜单的设计已经变得很简单，所以，读者不用担心，您会在很短的时间内学会菜单的简单设计和操作。当然，很多技巧还需要您在应用程序的编制中去亲身体会。

下面，请读者来学习菜单的定制。

首先，进入 Visual FoxPro 系统，新建一个文件，这样将弹出一个对话框，如图 7.1 所示：

图 7.1　新建文件对话框

选中菜单选项后确定，出现了菜单编辑选择窗体，选择菜单，如图 7.2 所示：

图 7.2 菜单向导选择

然后，读者会看到下面的窗体，如图 7.3 所示：

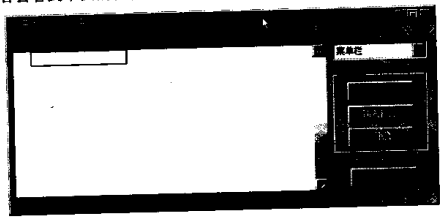


图 7.3 空的菜单设计器

其中的菜单名称是将来您设计菜单在窗体中的显示，结果是指该菜单属性，当用户用鼠标点击它时，其对于用户的反应是什么。当其弹出一个子菜单时，该属性应设置为“子菜单”，当该菜单调用一个程序时，其属性为“过程”，当其执行一个 Visual FoxPro 的命令时，该属性为“命令”等等。

如图 7.4 所示，这是一个设计好的菜单：

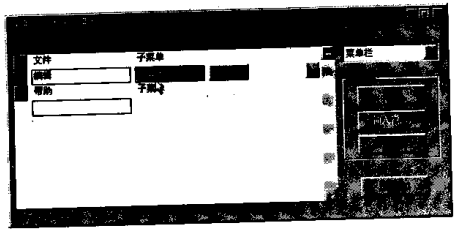


图 7.4 菜单设计器中设计的菜单选项

通过控制“结果”中的选项来创建下一级菜单或子程序和命令，请看文件中的子菜单设计，如图 7.5 所示：

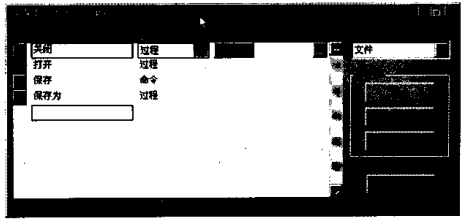


图 7.5　文件中的子菜单设计

当读者需要创建过程时，请用鼠标单击该过程的“创建”按钮，这样将弹出一个过程代码的编辑框，如图 7.6 所示：

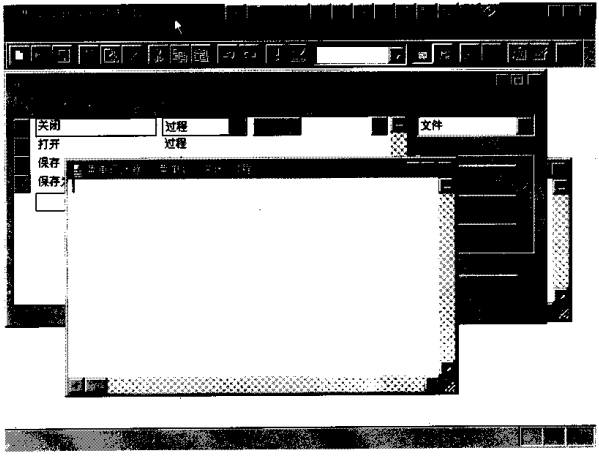


图 7.6　弹出的过程代码编辑框

请在该过程设计窗体中录入您的代码。

当您需要回到上一级菜单的话，请使用菜单级下拉列表，如图 7.7 所示：

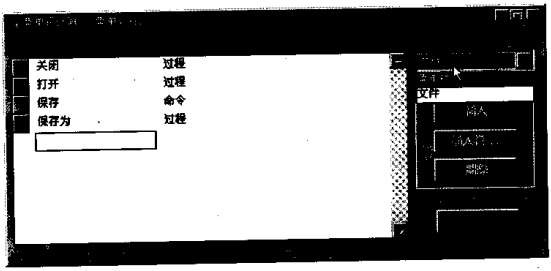


图 7.7 如何回到上一级菜单

通过上图中的预览按钮，读者就可以看到您设计的菜单。如图 7.8 、图 7.9 、图 7.10 所示：

图 7.8 文件菜单的子菜单

图 7.9 编辑菜单的子菜单

图 7.10 帮助菜单的子菜单

7.2.2 事件循环（事件驱动）

Visual FoxPro 的消息机制是基于事件驱动的。即：对于任何的对象都含有很多对应于不同事件（ Windows 系统的标准事件）的响应程序。例如：在窗体上进行鼠标的单击或鼠标的移动都会产生系统激活该对象的 MouseClick 和 MouseMove 事件。并完成用户定义的

任务。

在面向对象的编程中，不同的对象都封装了自己的事件和方法。而对于消息的响应，仅仅是对象的事件响应程序来反应用户的事件。下面看看 EditBox 控件的事件类型，它的事件很多，如下表 7.1 所示：

表 7.1　编辑框对象的事件

Click	DblClick	Destroy
DragDrop	DragOver	Error
ErrorMessage	GotFocus	Init
InteractiveChange	KeyPress	LostFocus
Message	MiddleClick Event	MouseDown
MouseMove	MouseUp	MouseWheel
ProgrammaticChange	RightClick	UIEnable
Valid	When	

不同的事件，该编辑框都会作出相应的响应。例如，当用户在该编辑框中用鼠标单击，则由系统引起对该控件的 Click 事件响应程序的调用；而当在该编辑框中用鼠标双击时，系统会调用该对象的 DblClick 事件响应程序；而当用户的鼠标在该控件上移动时，系统会调用该对象的 MouseMove 事件响应程序。

事件驱动使用户处理 Windows 消息变得更加简单、直接。以上的事件响应程序所响应的事件类型请读者参考 Visual FoxPro 的帮助。

7.3　创建应用程序的血肉——表单和报表

在 Windows 应用程序中，和用户交流和交互的是表单，而可以和打印机作用的是报表，用户通过使用报表可以用最短时间获得需要的数据，并可以在打印机中打印出来。实际上，表单也就是我们经常见到的对话框，其在 Windows 应用程序占了界面设计的主要部分。无论是和用户的交互上，还是从用户界面的设计上，表单都占有了绝对的地位。所以，表单和报表是应用程序的血肉、主体。

7.3.1　创建基类

在 Visual FoxPro 中，包含了很多 Microsoft 定义的类，而这些类都是 Windows 窗体的标准类，用户可以使用这些类来简化自己的程序设计。

在 Visual FoxPro 的应用程序设计中，我们可以使用 CREATEOBJECT 从类中定义和创建一个对象实体。Visual FoxPro 提供了 27 个预定义的基类，用户可以从中创建新的对象。

1. Visual FoxPro5.0 中的基类

下表 7.2 列出了 Visual FoxPro 的基类：

表 7.2 Visual FoxPro 的基类：

Check Boxes	ListBox	ComboBox
ToolBar	CommandButton	OptionButton
Spinner	TextBox	EditBox
Column	CommandGroup	Container
Control	Cursor	Custom
Form	FormSet	Grid
Header	Image	Label
Line	OLE Container	OLE Bound
OptionGroup	Page	PageFrame
Separator	Shape	Timer

基类又可以划分为二种：包容器类和控件类。包容器类是指，在这些控件的内部可以再次加入其他的控件，而且容器内部的控件是附属于容器的。例如，一个表单是一个容器，其可以容纳其他的控件；又如 Frame 控件，其可以创建从属于自身的组件。如图 7.11 所示：

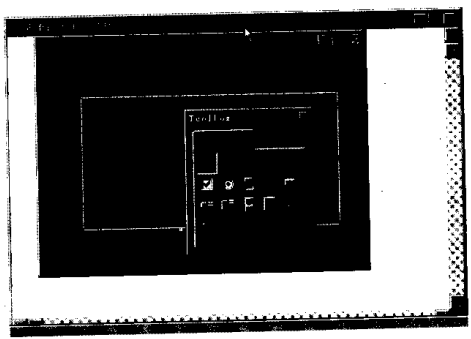


图 7.11 Frame 控件（容器类），可以创建从属于自身的组件

而控件类就不一样，控件类只能对不同的事件产生响应，而不能包容其他的控件。就象一个 EditBox 控件不可能包含一个图形一样。以下列出了属于包容器的控件：

Column
Command Button Group
Form
FormSet

Grid
Option Button Group
Page
PageFrame
ToolBar

为了引用存放在包容器中的控件属性，必须采用以下的方式：

[ContainerObjectName].[ControlName].[Property]

该语句说明了包容器中控件对象的属性。

2.从基类中派生子类

OOP（面向对象编程）方法的优点在于派生类可以从基类中继承基类的属性、方法和事件。而且可以自己在派生类中加入属于派生类的方法、属性和事件。

下面的代码是创建了用户定义的一个 Form 类。

该 Form 类中具有二个 Button 控件，并可以是用户自己定义 Click 事件响应程序，同时，添加了用户自定义的二个属性。

```
*************************************************
DEFINE CLASS formbutton AS form
Top = 0
Left = 0
Height = 165
Width = 297
DoCreate = .T.
Caption = "Form2"
Name = "formbutton"
用户定义 = .T.    &&用户定义为新添加的属性 1
用户定义 2 = .T.   &&用户定义 2 为新添加的属性 2
&&添加 Command1 （ Button 按钮 1 ）
ADD OBJECT command1 AS commandbutton WITH ;
    Top = 108, ;
    Left = 192, ;
    Height = 36, ;
    Width = 84, ;
    Caption = "确定", ;
    Name = "Command1"
&&添加 Command2 （ Button 按钮 2 ）
ADD OBJECT command2 AS commandbutton WITH ;
    Top = 108, ;
    Left = 96, ;
    Height = 37, ;
```

```
    Width = 85, ;
    Caption = "忽略", ;
    Name = "Command2"
PROCEDURE command1.Click
    [
    &&用户程序处理过程 2
    &用户自定义过程
    ]
ENDPROC
PROCEDURE command2.Click
    [
    &&用户代码
    &&用户程序处理过程
    ]
ENDPROC
ENDDEFINE
*************************************************
```

（1）从基类中派生的定义语法：

如上的代码，读者已经看出了如何去从一个基类中派生出其子类。定义一个子类需要用 DEFINE CLASS 命令来创建子类。下面给出了 DEFINE CLASS 命令完整的语法。

```
DEFINE CLASS ClassName1 As cParentClass[OLEPUBLIC]
    [[PROTECTED|HIDDEN PropertyName1,PropertyName2...]
     [object.]PropertyName = eExpression
[ADD OBJECT [PROTECTED] ObjectName AS ClassName [NOINIT]
 [WITH cPropertylist]]...
[ [PROTECTED|HIDDEN] FUNCTION|PROCEDURE Name
[NODEFAULT]
    cStatements
ENDDEFINE
```

但对于一般简单的定义，以下的语法格式就已经可以了。

```
DEFINE CLASS ClassName1 As cParentClass
    PropertyName = eExpression
[FUNCTION | PROCEDURE Name
    cStatements
[ENDFUNC | ENDPROC ]]...
ENDDEFINE
```

（2）添加新属性。

如上程序中的一段：

```
DEFINE CLASS formbutton AS form
```

```
Top = 0
Left = 0
Height = 165
Width = 297
DoCreate = .T.
Caption = "Form2"
Name = "formbutton"
用户定义 = .T.    &&用户定义为新添加的属性 1
用户定义 2 = .T.    &&用户定义 2 为新添加的属性 2
.....
```

用户定义了二个新属性：“用户定义”和“用户定义 2”

（3）定义新的事件。

在该类中的 Command1 按钮中的 Click 事件中加入用户自定义代码。

```
PROCEDURE command1.Click
    [
    &&用户程序处理过程 2
    &用户自定义过程
    ]
ENDPROC
```

（4）定义新方法。

```
PROCEDURE opendialog    &&创建一个新的 Opendialog 方法
    [
        用户自定义方法...
    ]
ENDPROC
```

3. 利用基类派生子类的例程

下面的例程是以 Container 容器类为基类来派生出一个用户定义的子类，如图 7.12 所示：

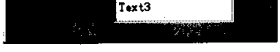


图 7.12　以 Container 容器类为基类来派生出一个用户定义的子类

代码如下：

```
*************************************************
*-- Class:          containertemp (containertemp.vcx)
*-- ParentClass:   container
*-- BaseClass:     container
*
DEFINE CLASS containertemp AS container

Width = 250
Height = 128
Name = "containertemp"

ADD OBJECT label1 AS label WITH
    FontSize = 12, ;
    Alignment = 2, ;
    Caption = "姓名", ;
    Height = 25, ;
    Left = 33, ;
    Top = 8, ;
    Width = 53, ;
    BackColor = RGB(0,255,255), ;
    Name = "Label1"

ADD OBJECT text1 AS textbox WITH ;
    Height = 25, ;
    Left = 100, ;
    Top = 8, ;
    Width = 113, ;
    Name = "Text1"

ADD OBJECT label2 AS label WITH ;
    FontSize = 12, ;
    Alignment = 2, ;
    Caption = "ID", ;
    Height = 25, ;
    Left = 33, ;
    Top = 47, ;
    Width = 53, ;
    BackColor = RGB(0,255,255), ;
    Name = "Label2"

ADD OBJECT text2 AS textbox WITH ;
    Height = 25, ;
```

```
    Left = 100, ;
    Top = 47, ;
    Width = 113, ;
    Name = "Text2"

ADD OBJECT label3 AS label WITH ;
    FontSize = 12, ;
    Alignment = 2, ;
    Caption = "口令", ;
    Height = 25, ;
    Left = 33, ;
    Top = 88, ;
    Width = 53, ;
    BackColor = RGB(0,255,255), ;
    Name = "Label3"

ADD OBJECT text3 AS textbox WITH ;
    Height = 25, ;
    Left = 100, ;
    Top = 88, ;
    Width = 113, ;
    PasswordChar = "*", ;
    Name = "Text3"

ENDDEFINE
*
*-- EndDefine: containertemp
```

7.3.2　创建表单

表单在 Windows 应用程序中的应用是最广的。无论是用 Visual C++，Visual Basic 或是 Visual FoxPro 编制的 Windows 应用程序，其图形界面的接口都是表单。也就是说，任何一个应用程序（Windows 系统下）都至少有一个表单，而且用户和计算机的交互都是通过表单来实现的。在 Visual FoxPro 中，将对象的定义完全的溶入了表单。本节将介绍简单表单的设计。在下一章中将讲述利用 ActiveX 控件来设计复杂的表单。

在面向对象的程序中，可视对象的类型被划分为二种：容器类和控件类。容器类允许其他控件放置在自身的内部，并构成一个完整的整体。表单是一个容器类。它可以容纳其他的，几乎所有的控件。例如：文本框、标签、复选框和图像框等等。

表单和以前的 Visual FoxPro 窗口已经不再一样，因为它已经被完全地面向对象。其具有自身的属性、方法和事件。例如，用户可以通过它自身的属性来定义表单的背景色、前景色、字体和表单名字等等。

创建表单的方法有以下几种：

系统菜单中的 File 、 New Form 。（使用表单设计器）

NewForm = CREATEOBJECT("FORM")

利用 Form Wizard

下面，我们将主要地讲述一下使用表单设计器和 Form Wizard 来设计表单。

1. 表单设计器

系统菜单中的 File 、 New Form 可以激活表单设计器，如图 7.13 所示：

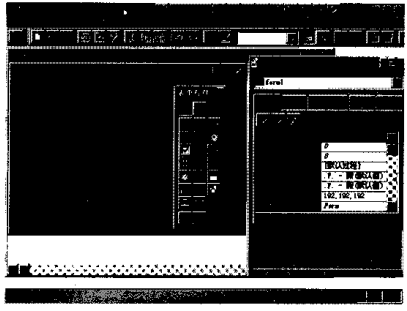


图 7.13 激活的表单设计器

Visual FoxPro 使用控件工具条来向表单中加入对象。可以在上图中看到表单控件和属性窗口。表单控件窗口中包含了表单可以使用的控件。属性窗口是当前激活控件的属性，可以在该窗口中改变控件的属性值。下面将讨论一下简单的表单的设计。

示例（1）：在表单中加入 EditBox 控件

如图 7.14 所示，在表单中加入标签、文本编辑框，并将文本编辑框和一个数据表相连接。

图 7.14 表单中加入标签、文本编辑框

示例（2）：一个比较复杂的表单。如图 7.15 所示，该例程创建了这样的一个表单。

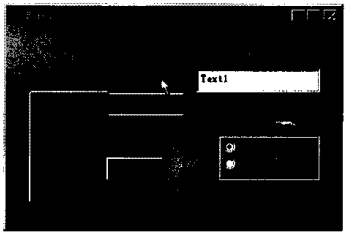


图 7.15　较复杂的表单设计窗体

代码如下：

```
**************************************************
*-- Form:              form1 (表单 1.scx)
*-- ParentClass:   form
*-- BaseClass:      form
*
DEFINE CLASS form1 AS form

Top = 3
Left = 7
Height = 219
Width = 358
DoCreate = .T.
Caption = "Form1"
Name = "form1"

ADD OBJECT optiongroup1 AS optiongroup WITH ;
    ButtonCount = 2, ;
    Value = 1, ;
    Height = 48, ;
    Left = 228, ;
    Top = 120, ;
    Width = 109, ;
    Name = "Optiongroup1", ;
    Option1.Caption = "Option1", ;
    Option1.Value = 1, ;
```

```
    Option1.Height = 16, ;
    Option1.Left = 5, ;
    Option1.Top = 5, ;
    Option1.Width = 63, ;
    Option1.Name = "Option1", ;
    Option2.Caption = "Option2", ;
    Option2.Height = 16, ;
    Option2.Left = 5, ;
    Option2.Top = 23, ;
    Option2.Width = 63, ;
    Option2.Name = "Option2"

ADD OBJECT text1 AS textbox WITH ;
    Height = 25, ;
    Left = 204, ;
    Top = 48, ;
    Width = 133, ;
    Name = "Text1"

ADD OBJECT label1 AS label WITH ;
    Caption = "使用表单设计器来设计自己的表单界面", ;
    Height = 25, ;
    Left = 48, ;
    Top = 12, ;
    Width = 265, ;
    Name = "Label1"

ADD OBJECT label2 AS label WITH ;
    Caption = "选项提示", ;
    Height = 24, ;
    Left = 228, ;
    Top = 96, ;
    Width = 109, ;
    Name = "Label2"

ADD OBJECT pageframe1 AS pageframe WITH ;
    ErasePage = .T., ;
    PageCount = 2, ;
    ActivePage = 1, ;
```

```
    Top = 72, ;
    Left = 24, ;
    Width = 169, ;
    Height = 121, ;
    Name = "Pageframe1", ;
    Page1.Caption = "Page1", ;
    Page1.Name = "Page1", ;
    Page2.Caption = "Page2", ;
    Page2.Name = "Page2"

ADD OBJECT form1.pageframe1.page1.label1 AS label WITH ;
    Caption = "第一页", ;
    Height = 25, ;
    Left = 23, ;
    Top = 10, ;
    Width = 121, ;
    Name = "Label1"

ADD OBJECT form1.pageframe1.page1.command1 AS commandbutton WITH ;
    Top = 46, ;
    Left = 83, ;
    Height = 25, ;
    Width = 61, ;
    Caption = "确定", ;
    Name = "Command1"

ADD OBJECT form1.pageframe1.page2.text1 AS textbox WITH ;
    Height = 25, ;
    Left = 35, ;
    Top = 22, ;
    Width = 97, ;
    Name = "Text1"

ADD OBJECT form1.pageframe1.page2.command1 AS commandbutton WITH ;
    Top = 58, ;
    Left = 59, ;
    Height = 25, ;
    Width = 61, ;
    Caption = "返回", ;
    Name = "Command1"

PROCEDURE optiongroup1.Click
```

```
    &&用户定义过程...
ENDPROC

PROCEDURE text1.KeyPress
    LPARAMETERS nKeyCode, nShiftAltCtrl

    &&用户定义过程...
ENDPROC

PROCEDURE pageframe1.Click
    if pageframe1.activepage =1 then
        &&用户定义过程...
    else
        &&用户定义过程 2...
    endif
ENDPROC
ENDDEFINE
*
*-- EndDefine: form1
```

2. 使用表单向导（ Form Wizard ）

(1) 创建一对一表单。

一对一表单是使用 Form Wizard 来创建的。如图 7.16 所示，通过表单向导的向导选取选择一对一表单。

第一步：表单的第一页是用于选取字段的。

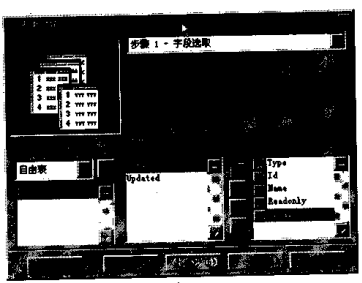


图 7.16 使用 Form Wizard 来创建一对一表单

请注意：选定字段的顺序决定了向导在表达式中安排字段的顺序，以及表单的缺省标签顺序。如果读者需要进一步的设计由向导产生的表单，则该顺序对用户的影响不大，否

则，请小心排列选定字段的字段顺序，然后进行下一步。

第二步：定义表单的整体风格。Visual FoxPro 定义了五种基本风格：标准式，凹陷式，阴影式，边框式和浮雕式。

当选中了不同的风格，位于对话框左边的图像将显示该风格的一个小示例，如图 7.17 所示：

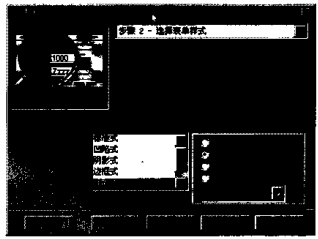


图 7.17　定义表单的整体风格，Visual FoxPro 定义了五种基本风格

按钮类型也如图 7.17 中所示，请读者自己尝试。

第三步：表单向导第三页。如图 7.18 所示，为记录定义了排列顺序。因为表单存储了数据环境，因此，可以在向导的第一页定义一个所选列表的顺序视图。可以将排列顺序定义为升序或降序。而且该选择将作用于整个排序。

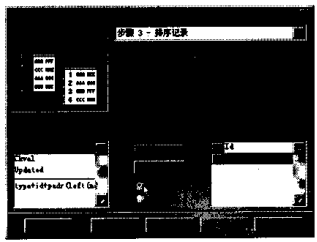


图 7.18　表单向导第三页

请读者注意排序和索引的关系。

第四步：表单设计的最后一页。如图 7.19 所示，该页面仅仅提供了表单的标题，该标题将显示在完成表单的标题栏中。

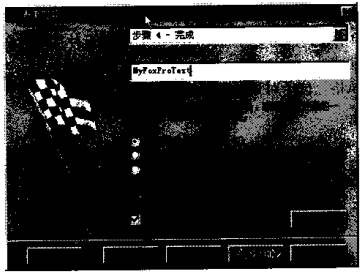


图 7.19 表单设计的最后一页

该页面中有三个存储选项。第一个选项是存储表单以便以后使用，然后将返回到用户启动该向导的状态。第二种选项是存储表单并立即执行它。第三种选项是存储表单并打开表单设计器，从而对表单进行进一步的设计。请看第三种选项出现的表单设计器，如图 7.20 所示：

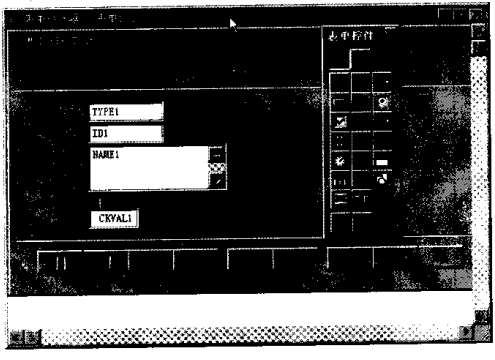


图 7.20 存储表单并打开表单设计器

对于复杂表单的设计，我们将在下一章讲到 ActiveX 控件时详细讨论。

在一对一表单向导的最后一页中有个预览按钮。如果单击该按钮，Visual FoxPro 将根据用户提供的定义来显示该表单。下图 7.21 就是通过使用预览得到的表单运行结果。而上图 7.20 是在表单设计器中的表单结构。

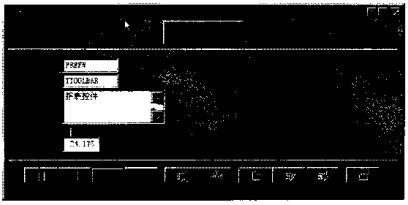


图 7.21　使用预览得到的表单运行结果

在表单的最顶部中间有一个按钮，按钮上标明了“返回向导”。不用担心该按钮会影响您的应用程序的界面，它仅仅是在使用表单向导时才会出现的。而通过单击该按钮，用户可以返回到表单向导的环境中。

当用户在表单中发现有不满意的地方，用户可以返回到设计它的特定步骤中进行修改，只要用户不退出向导，用户就可以在这些步骤和预览模式间不停的反复，重复地修改用户界面，直到满意为止。用户也可以是该向导来测试不同的风格。

在最后一页中，还需要用户注意的是位于页面底部的复选框。当被选定时，该选项告诉 Visual FoxPro 使用多页的页框架来显示字段。如果所有的字段可以在一个页面中显示的话，Visual FoxPro 忽略该项；而当字段较多时，不能在一屏显示完所有的数据，Visual FoxPor 就会忽略多余的字段。在建立页框架时，Visual FoxPro 将根据需要建立任意多页，同时均衡每页上的可利用空间，以便使得每页空间都使用得不会太挤或太空。图 7.22 显示了由该选项建立的表单，并利用预览显示出来。

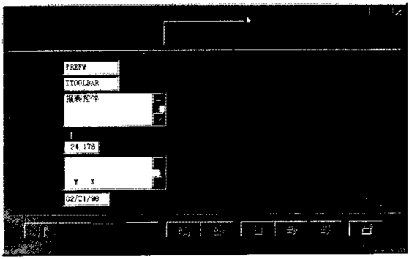


图 7.22　使用预览选项来浏览自己定义的表单

当按了“确定”按钮，表单向导将完成其任务，并提示用户存储该表单后退出表单向导。如果用户发现了表单上的不合适和不满意之处，请使用表单设计器来对表单进行修改。而不能再使用表单向导来修改，也不用重新使用表单向导来重新建立表单。

(2) 一对多表单

使用一对多表单向导（One-To-Many Form Wizard）和一对一表单向导有一定的关系。它们的主要区别在于一对多表单向导使用了两张表格来创建表单。尤其是它可以创建一个表单来显示个别双亲记录和它们所有的子记录。双亲记录总是显示在表单的上半部分，而且使用单独的控件。子记录通常在表单的下半部分的滚动网格中显示。以前，该类型的表单很难创建，要求使用很多的代码，并难以使其正常工作。而一对多表单向导使用户可以通过简单的六个步骤就可以创建这种表单。

第一步：从主或双亲表格中选择字段。该不和常规表单向导的第一步没有区别。但对于选定的数据有一定的要求：该数据要求是作为双亲表格的表。

第二步：和第一步相似。只是该页提示从相关表格或子表格选择字段。同样，必须先选中一个表格，然后从中选取字段。如图 7.23 所示。

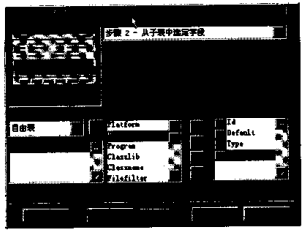


图 7.23 该页提示从相关表格或孩子表格选择字段

注意：窗体左上角的箭头指向了表单的下半部分。请记住，所有子记录的数据都将出现在表单的下半部分，而且是在一个类浏览器控件当中。当所有的字段已经在一行中显示不下，则向导在该控件中加入一个水平滚动条。

第三步：创建双亲表格和子表格之间的关系。即：定义两个表格之间的相关字段。如图 7.24 所示。

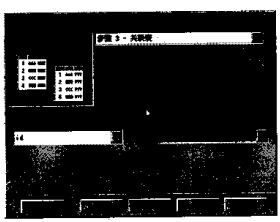


图 7.24 创建双亲表格和子表格之间的关系

该例中定义了两个表格中的 ID 字段为相关字段。在一个关系中，所选字段不需要相同的名字，但它们必须有相匹配的类型。

第四步：定义表格的风格。和一对一表单向导相同。

第五步：定义用于显示主表格或双亲表格中记录的顺序。而其操作和常规表单的操作相同。

第六步：要求用户给出一个表单的标题。其余的工作和常规表单的设计步骤，操作相同。

同样，当表单中存在问题时，通过表单设计器来恢复和改变。不用再次进行设计。

7.4　创建报表

使用报表向导来设计表的报表。只要经过一系列简洁的步骤，通过回答向导提示您的几个简单的问题，您就可以创建报表，过程中您可以指定用以创建报表中控制的表和字段。

Visual FoxPro 有三种类型的报表向导：

通用的用于单个表格报表的报表向导；

一对多报表向导，使用两个表格间的双亲和孩子关系来创建报表；

分组/总计报表向导，在一组记录的基础上创建总计数字型字段的报表。

7.4.1　创建一对一报表

创建一对一报表需要五个步骤：

第一步：一对一报表和一对一表单的第一步是一样的，也是选择字段。在此选择数据库、表格以及需要包含在报表中的个别字段。如图 7.25 所示：

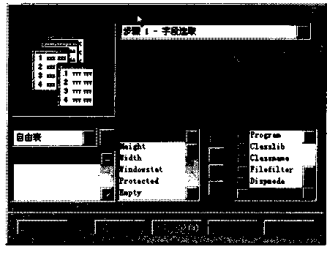


图 7.25　使用向导来设计一对一报表

当需要多张表中的字段时，建议先创建一个视图，并将这些字段保存在数据库中。

第二步：选择三种标准报表风格。如图 7.26 所示：

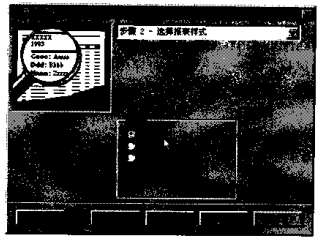


图 7.26 选择三种标准报表风格

该样式有：经营式、账务式和简报式。选中了不同的样式，窗体的左上角将显示了该样式的小示例。

第三步：定义报表的整体布局。如图 7.27 所示。

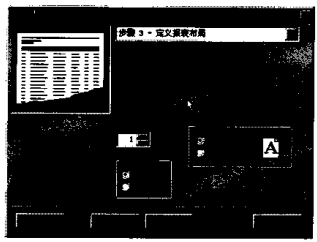


图 7.27 定义报表的整体布局

该窗体中可以定义报表显示的三个方面：第一选项定义了字段布局。两个可选项为“列”和“行”。当报表中的所有字段可以在一页中水平地排满时，可以使用“列”风格来设计报表，这样可以在一个页面中显示更多的数据；而当每个记录都有很多的字段时，即：一行中已经容纳不下所有的字段，那么就的考虑使用“行”风格的报表布局设计。

列数决定可以在一页内显示的重复数据的列数。例如，如果使用“行”风格布局，则数据不能超过页的中间。此时，可以考虑在一页内打印两列以加倍在每页上能输出的信息量。如图 7.27 显示了该选项。

读者一定使用过 Windows 系统下的打印机，并使用过纸张的打印风格，即：打印的方向。该处的纸张选项和它们是很相似的。对于不同的纸张，都有自己的高度和宽度，而且不同类型的纸张是各不不同的。当使用“横向”将意味着纸张的打印方向是横向；而纵向指的是我们一般使用的纸张方向。其实，在图 7.27 中，读者可以选中“横向”或“纵向”，

注意到右边的小图标的变化，这就是不同的纸张方向的效果。

第四步：该步定义记录的排列顺序，如图 7.28 所示。

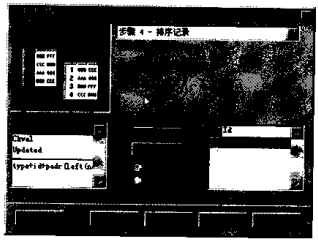


图 7.28　定义记录的排列顺序

可以选用任意一个在报表中的字段为排序字段。在选定字段列表框中的字段决定了排序的等级。排序方向可以是升序或降序。（当您需要三个以上的字段来排序的话，请在报表设计器中对报表进行修改，但一般的应用程序也用不了三个排序。）

第五步：为报表定义一个标题来存储该报表。如图 7.29 所示。

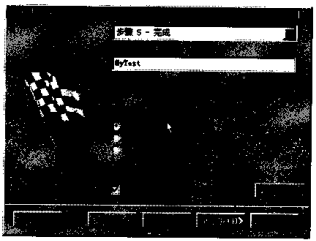


图 7.29　为报表定义一个标题来存储该报表

当字段长度超过所想的列宽度时，还可以选择使字段换行的选项。最后页面上比较重要的按钮是预览按钮。请在保存报表和退出前先预览一下。因为在预览了报表以后还可以退回到设计时的状态。当用户想要对报表进行修改时，可以重新会到设计阶段在向导中进行修改。不过，读者也可以保存后在报表设计器中进行修改工作。

7.4.2　创建一对多报表

一对多报表向导的前两个步骤分别用于选择表和用于主表和相关的字段。该两步和一

对多表单向导的前两步是一样的。

第三步是建立双亲和子表格之间的相连关系。这和一对多表单向导的第三步也是一样的。读者请参考一对多表单向导的第三步。

第四步定义了输出建立的排列顺序。该顺序只能用于主表格上。该向导中没有提供在双亲表格中排序子记录的方法。如果读者想要这样做，请先在该向导外建立一个合适的视图并将其用于报表的定义。

第五步：定义布局设计选项。如图 7.30 所示。

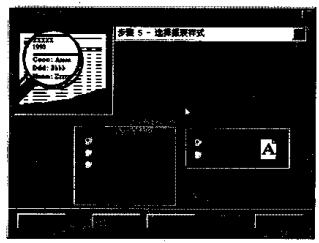


图 7.30 一对多报表定义布局设计选项

该选项比简单报表所支持的选项要少。用户参考一对一报表的该步骤来设计一对多报表的报表样式。

第六步：输入报表标题等选择。和一对一报表相似。请读者在保存设计的报表之前先进行预览。

7.4.3 创建分组/总计报表

前两种报表类型目的是打印出报表的数据，它们不作总计，也不按指定的字段地记录分组。如果选择一个字段排序，则报表中不会包含任何标题行或脚注行来区分这一组和下一组。如图用户希望并且需要确正是这种报表，那么就可以使用分组/总计报表。

第一步：字段选取。（参考一对一表单向导设计）

第二步：分组记录。如图 7.31 所示，这是该向导中第一个和以前向导不同的地方。可以使用“数据分组”将记录分类和排序，这样可以很容易地读取它们。当在“分组依据”框选择一个字段时，可以选择“修改”，进一步更改分组。可以选择适当的“总计”与“小计”。

第三步：排序记录。该向导和以前一对一表单的向导的排序记录步骤相同。请按照每组中记录的排序顺序选择字段。第二步中选择的用来分组的字段在这一步不可用。

第四步：选择报表样式。当单击任何一种样式时，向导都更新放大镜中显示的字体、布局、报表格式的示例。

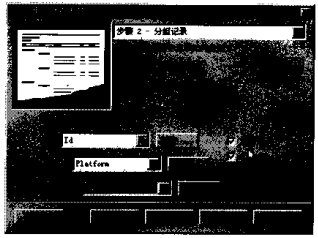


图 7.31　分组/总计报表的分组记录

第五步：完成。生成报表和预览报表。如果选择的字段数与报表的一行的宽度不一致，这些字段会放在下一行。如果不想自动换行，删除“对不能容纳的字段进行折行处理”选项的选定。单击“预览”按钮，在向导关闭前显示报表布局。向导保存报表之后，可以象其它报表一样，在“报表设计器”中打开并修改它。

7.5　创建应用程序的皮肤——完善

当创建完的表单、菜单、表报和部分的应用过程后，我们需要进行的工作是将这些分开的部件组装起来，从而使其成为一个真正的应用程序，并可以用来发布。很多用户还是运用着以前的 Command 命令窗口，但当您使用了 Visual FoxPro 的项目管理器（Project Manager）后，相信您也会习惯于使用该产品来结构化您的应用程序，并通过它来完善您的应用程序。

7.5.1　项目管理器

如图 7.32 所示的项目管理器，其中包括了很多的部件。

图 7.32　“项目管理器”以类似于大纲的形式组织各项

在“项目管理器”中，以类似于大纲的形式组织各项，可以展开或折叠它们。在项目中，如果某类型数据项有一个或多个数据项，则在其标志前有一个加号。单击标志前的加号可查看此项的列表，单击减号可折叠展开的列表。

7.5.2 “项目管理器”按钮

新建：

创建一个新文件或对象。此按钮与“项目”菜单的“新建文件”命令作用相同。新文件或对象的类型与当前选定项的类型相同。

附注：从“文件”菜单中创建的文件不会自动包含在项目中。使用“项目”菜单的“新建文件”命令（或“项目管理器”上的“新建”按钮）创建的文件自动包含在项目中。

添加：

把已有的文件添加到项目中。此按钮与“项目”菜单的“添加文件”命令作用相同。

修改：

在合适的设计器中打开选定项。此按钮与“项目”菜单的“修改文件”命令作用相同。

浏览：

在“浏览”窗口中打开一个表。此按钮与“项目”菜单的“浏览文件”命令作用相同，且仅当选定一个表时可用。

关闭：

关闭一个打开的数据库。此按钮与“项目”菜单的“关闭文件”命令作用相同，且仅当选定一个表时可用。如果选定的数据库已关闭，此按钮变为“打开”。

打开：

打开一个数据库。此按钮与“项目”菜单的“打开文件”命令作用相同，且仅当选定一个表时可用。如果选定的数据库已打开，此按钮变为“关闭”。

移去：

从项目中移去选定文件或对象。Visual FoxPro 会询问您是仅从项目中移去此文件，还是同时将其从磁盘中删除。此按钮与“项目”菜单的“移去文件”命令作用相同。

连编：

连编一个项目或应用程序，在专业版中，还可以连编一个可执行文件。此按钮与“项目”菜单的“连编”命令作用相同。

预览：

在打印预览方式下显示选定的报表或标签。当选定“项目管理器”中的一个报表或标签时可用。此按钮与“项目”菜单的“预览文件”命令作用相同。

运行：

执行选定的查询、表单或程序。当选定项目管理器中的一个查询、表单或程序时可用。此按钮与“项目”菜单的“运行文件”命令作用相同。

7.5.3　将自己的文件加入到项目中

一个 Visual FoxPro 项目有可能包含若干独立的组件，这些组件作为单独的文件保存。例如，一个简单的项目可以包括表单（*.FRX 文件）、报表（*.SCX 文件）和程序（*.PRG 和*.FXP 文件）。除此之外，一个项目经常包含一个或者多个数据库（*.DBC 文件）、索引（*.CDX 和*.IDX 文件）及表（保存在*.DBF 和*.FPT 文件中）。一个文件若要被包含在一个应用程序中，必须要添加到项目中。这样，在编译应用程序时，Visual FoxPro 会在最终的产品中将该文件作为组件包含进来。

下面的方法，可以方便地向一个项目中添加文件：

使用应用程序向导，可以建立项目和添加文件。

如果要自动向一个项目中添加新的文件，请打开该项目，然后在“项目管理器”中建立新的文件。

要向一个项目中添加已存在的文件，请打开项目，使用“项目管理器”。

下面，我们将介绍一下如何使用“项目管理器”来手工添加文件到项目中。

如果用户需要向项目中加入自己定义的文件，请：

（1）在“项目管理器”中，选择“添加”按钮。

（2）在“打开”对话框中，选择要添加的文件。

如果在一个程序中或者表单中引用了某些文件，那么 Visual FoxPro 会将它们添加到项目中。例如，在一个项目中，如果某程序包含如下的命令，那么 Visual FoxPro 会将 ORDERS.SCX 文件添加到项目中：

```
DO FORM ORDERS.SCX
```

如果使用这种方法引用一个文件，该文件不会马上包含在项目中。当连编此项目时，Visual FoxPro 将分析所有文件的引用，并自动把所有的隐式文件包含在项目中。此外在一个新的文件中，如果通过用户自定义的代码引用任何一个其他文件，项目连编也会分析所有包含及引用的文件。当在下一次查看该项目时，引用的文件会出现在“项目管理器”中。

如图 7.33 所示，这是一个项目中包含的一部分文件。

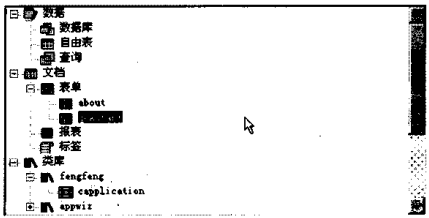


图 7.33　项目管理器中包含的一部分文件

当用户需要修改文件、对象时，也可以在“项目管理器”中直接修改。即通过点亮用

户需要修改的对象（文件），单击修改按钮。例如：通过“项目管理器”来修改一个表单。如图 7.34 所示。

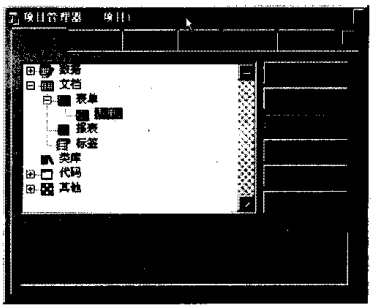


图 7.34 通过“项目管理器”来修改一个表单

7.5.4 为一个项目建立应用程序

一个项目设计的最后一步是连编它。此过程的最终结果是将所有在项目中引用的文件合成为一个应用程序文件。可以将应用程序文件和数据文件一起发布给用户，用户可执行该文件运行应用程序。从项目建立应用程序的具体步骤如下：

1. 测试项目

为了对程序中的引用进行校验，同时检查所有的程序组件是否可用，需要对项目进行测试。所以，需要重新连编项目。这样 Visual FoxPro 会分析文件的引用，然后重新编译过期的文件。测试一个项目需要以下的步骤：

（1）在“项目管理器”中，选择“连编”；

（2）在“连编选项”对话框中，选择“重新连编项目”；

（3）选择任意所需的其他选项，选择“确定”。

如果在连编过程中发生错误，这些错误将收集在当前目录下，一个名字为项目的名称，扩展名为*.ERR 的文件中。编译错误的数量显示在状态栏中，也可以立刻查看错误文件。

2. 将项目连编为一个应用程序文件

一个项目可以编译成应用程序文件（*.APP）或者可执行文件(*.EXE)。如果用户已经安装了 Visual FoxPro，则可以运行一个*.APP 文件。另外一个选择方案是建立一个可执行文件。该可执行文件需要和两个 Visual FoxPro 动态连接库（VFP500.DLL 和 VFPxxx.DLL）联接，这两个库和应用程序一起构成了 Visual FoxPro 所需要的运行环境。VFPxxx.DLL 是

用来指定应用程序开发的地区版本。

连编一个应用程序需要以下步骤：

(1) 在“项目管理器”中，选择“连编”按钮。

(2) 在“连编选项”对话框中，选择“连编应用程序”，生成*.APP 文件；或者“连编可执行程序”来建立一个*.EXE 文件。

(3) 选择所需的其他选项并选择“确定”按钮。

通过以上讲述的“项目管理器”的功能和使用方法，用户可以编译和运行项目程序或者其中的表单、报表等等。总之，“项目管理器”是一个完善、集成应用程序的优良工具。

7.6 小　结

本章讲述了面向对象的程序设计方法（OOP）。从 Visual FoxPro 中的 27 个基类中派生出自己定义的用户定义类。并利用了较为明了的例程说明了如何从基类中派生子类的方法。

通过设计一个简单的菜单给读者介绍了菜单的设计方法和步骤。

表单在 Windows 应用程序中的应用太广，本章也用了大量的例程和代码说明了简单的表单的设计方法和使用一对一表单向导以及一对多表单向导来设计一对一表单和一对多表单。

报表是数据库设计的结果表示，可以用来打印。本章介绍了三种报表的设计：一对一报表、一对多报表和分组/总计报表。

最后在本章中讲到的是如何使用项目管理器来使用户定义的表单、报表、菜单和过程构成一个整体的应用程序。

第八章　高级技术 OLE ActiveX

8.1　Activex 控件的使用

目前，在开发 Windows 32 位应用程序中的热门话题就是 ActiveX 。那么到底什么是 ActiveX 呢？其实， ActiveX 是为了开发基于 Microsoft Web 技术的应用才提出来的。它就是这样的一个产品： Microsoft 公司把 AcitveX 提供给 Windows 32 位应用程序的开发者，使在保护了 Windows 技术中现有开发的同时，又把计算机桌面环境与现今正处在热潮中的 Internet 网络开发资源集成起来。

通过对象的链接与嵌入（ OLE ）和 ActiveX 控件，您可以借助其他 Windows 应用程序来扩展您的 Visual FoxPro 应用程序的功能。在应用程序的表单或通用型字段中，可以包含其他应用程序的数据，例如文本数据、声音数据、图片数据或视频数据。您可以使用创建这些数据的应用程序，以可视方式查看或操作这些数据，或者以编程方式控制这些应用程序，虽然不可视，但自动地操作数据。其他应用程序也可以通过自动化开发 Visual FoxPro 的功能。您甚至还可以在 Visual FoxPro 中创建 OLE 服务程序（ ActiveX 组件），使您的应用程序或其他应用程序可以进行本地或远程访问。

ActiveX 包括了两个关键的技术： Windows 32 位编程接口和 COM （组件对象模型）。为了开发 ActiveX 控件， Microsfot 提供了专有的开发工具， ActiveX 控件是灵活多变的，因为可以通过生成它们的子类来创建其他控件，可以通过使用与 ActiveX 控件有关的事件、方法程序和属性来控制它们。遗憾的是不能使用 Visual FoxPro 创建 ActiveX 控件，但您可以使用 Microsoft Visual C++4.0 或 5.0 提供的 Microsoft OLE Custom Control Developer's Kit 来创建它们,在这里就不一一叙述了。

在本书中主要讲述的是关于 ActiveX 在 Virsual FoxPro 中的应用。对于 ActiveX 应用来说， ActiveX 控件和以前 Virsual FoxPro 中的可视控件在使用上几乎是一样的，但是其功能增加了很多，主要也就在于它们的属性和方法。以后读者也会看到，在控件的使用说明中，主要讲述的也是控件的主要属性和方法的使用。

当初次进入 Virsual Foxpro 环境而设计表单时（见上一章的“创建表单”），读者将发现 Virsual Foxpro 的工作环境是这样的（图 8.1 ）：

在表单中加入一个控件时的开发环境：

而将 ActiveX 控件加入到图中的表单控件中的过程是这样的：

第一步：选中菜单中的工具项，如图 8.2 ；

第二步：在图 8.3 中选定控件选项并且在单选框中选定 ActiveX 选项；

第三步：在图 8.4 中选定想要加入的 ActiveX 控件，例如： ImageList ， ListView 等等，然后确定。

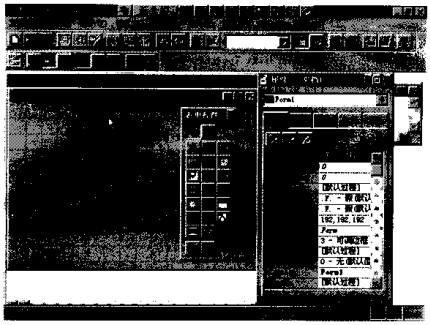


图 8.1　Visual FoxPro5.0 的开发环境（没有 ActiveX 控件）

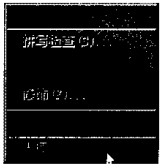


图 8.2　菜单选项

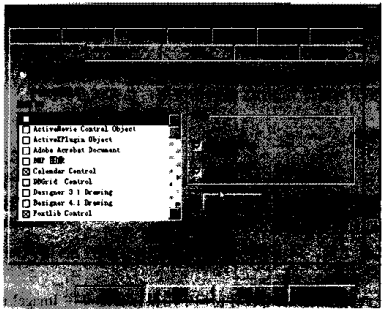


图 8.3　选项菜单列表

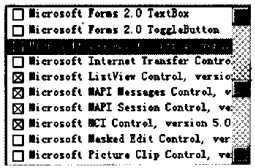


图 8.4 OLE ActiveX 控件列表

第四步：表单控件窗体中如图 8.5 所示，选择 AcitveX 控件选项，则表单控件窗体将变为图 8.6 所示。

图 8.5 选项菜单列表

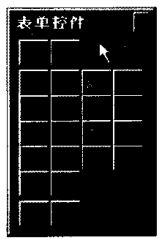


图 8.6 表单上的 ActiveX 控件

这样，读者就在设计表单的表单控件中加入了 ActiveX 选项，然后，就可以使用这些控件来完成表单的界面设计。

以后，我们将要学习部分 ActiveX 控件的应用，并有部分的例子。

8.1.1 ImageList 控件

ImageList 控件是一个存储 Image(图像)的数组，它必须和其他控件或应用相结合，才能表现出强大的功能。为了很好地说明 ImageList 控件的使用，让我们先看以下 ImageList 控件中的部分属性和这些属性的意义和使用方法。

在说明 ImageList 之前，请读者先看看 ImageList 的属性窗口的 General 属性页以及页面上部件的名称。

General 属性页面的弹出：表单设计窗口中用鼠标右键点住 ImageList 控件，将弹出图 8.7 所示的弹出菜单：

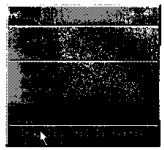


图 8.7　ImageList General 属性页面菜单

选择 ImageListCtrl Porperties，然后将弹出图 8.8 所示的窗体：

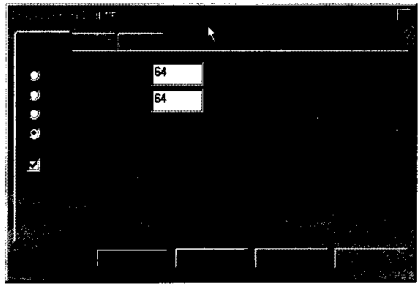


图 8.8　ImageList General 属性窗体

为了解释清楚 ImageList 控件的用法，不得不说明 ImageList 控件的各个属性和初始化 ListImage 对象的过程。如图 8.9：

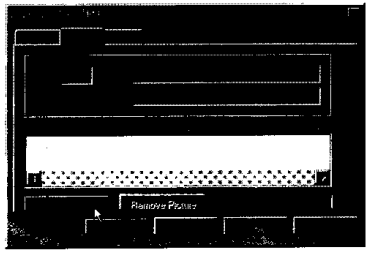


图 8.9　ImageListCtrl 属性窗口

前面说过，ImageList 是一个图像的存储数组，如何在设计阶段来初始化 ImageList 控件中的 ListImages 对象呢？注意到图 8.9 中的 Insert Picture...按钮，这就是关键的所在。如图 8.10 所示，这是增加图像之后的显示：

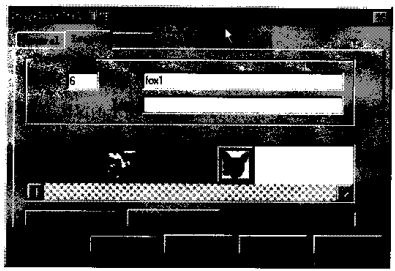


图 8.10 加入图像后的 ImageListCtrl 属性

前面讲的是在设计阶段时初始化 ImageList 控件，而在运行阶段，您可以通过使用 ListImages 包集的 Add 方法来给 ImageList 增加图像。例如：

```
&& 向 ListImages 包集中增加图像
imgtest = ImageList1. ListImages.Add(,"rocket",LoadPicture("icons\industry\rocket.ico"))
```

1. ImgaeList 属性

下面是 ImageList 关键的几个属性：

Name 属性

ImageHeight 属性和 ImageWidth 属性

ListImages 属性

Index 属性

第一个重要的属性就是图 8.11 所示的 Name 属性：

该属性对于任何控件来说都是至关重要的，它主要是在运行阶段来控制该控件、对象，用来区分不同的 ImageList 控件。对于该例子来说，其 Name 属性是 MyTest 。

ImageList 控件中的图像是有一定大小的，而控制图像大小的是 ImageHeight 属性和 ImageWidth 属性。可以在如图 8.8 所示的那样在 Width 和 Heigth 中初始化图像的大小，也可以在程序中通过改变以上两个属性来达到控制的效果。

不同尺寸的图像也可以增加到一个 ImageList 控件中，但是它们反映出来的仍然是具有同样的大小。这取决于第一个加入的图像的尺寸大小。

为了得到 ImageList 中存储的图像或把图像存储在 ImageList 中，必须使用 ListImages 属性和其 Index 属性，例如：您可以通过使用 Picture 对象的 Picture 属性来控制 ImageList 控件的显示。以下的代码将在 ListImages 包集中通过 Picture 属性，创建一个新的 StatusBar

控件中的 ListImage 对象。

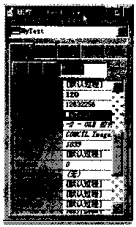


图 8.11　ImageList 属性窗口

&& panel 是一个 Panel 对象

panel = StatusBar1.Panels.Add() && 增加一个新的 Panel 对象

panel.Picture = ImageList1.ListImages(1).Picture　&&图像附值

ImageList 控件除了存储一些图像外，它还可以在图像传递过程时进行图像的操作，例如：用 ImageList 控件的 Overlay 方法可以创建一个由两个不同图像复合的结果图像。

此外，您还可以绑定一个或更多的 ImageList 控件到特定其他的 Windows95 公共控件中，这样可以保存系统的资源。这些公共控件包括 ListView 控件、ToolBar 控件、TabStrip 控件和 TreeView 控件。而为了做到这些，您必须通过该控件合适的属性和 ImageList 控件建立一个必要的联系。例如：对于 ListView 控件来说，您必须设置其 Icon 和 SmallIcon 属性，而对于 TreeView 控件、TabStrip 控件和 ToolBar 控件来说，您必须设置其 ImageList 属性。

对于这些控件，您可以在设计阶段通过使用属性对话框来详细地说明 ImageList 属性，而在运行阶段，您也可以通过设置该控件的 ImageList 属性来实现。以下是一个简短的例子（对 TreeView 而言）。

&& TreeView1 是一个 TreeView 对象变量

&& ImageList1 是一个 ImageList 对象变量

TreeView1.ImageList = ImageList1　&&附值

当您绑定了一个 ImageList 和其他一个控件后，您可以在应用程序中通过 ImageList 控件所包含的 ListImages 合集对象的 Index 属性和 Key 属性来操纵 ImageList 中的图像。以下的例子就是通过 ImageList 控件 ListImage 对象中的 Index 属性和 Key 属性来对 TreeView 控件的第一个 Node 对象的 Image 属性赋值（关于 TreeView 的细节以后讲述）。

&& 使用 Index 属性控制

TreeView1.Nodes(3).Image = 1

&& 使用 Key 属性控制

TeeView1.Nodes(3).Image = "image 1"

2. ImageList 例程

关于其他控件的详细说明，我们将分别在讲该控件时加以说明。而 ImageList 控件的例程也将在以后章节中和其他控件例子结合起来讲（参考 SysInfo 控件和 TreeView 控件）。

3. 安装提示

ImageList 控件是 ActiveX 控件组 COMCTL32.OCX 中的一部分，如果您的应用程序中采用的 ImageList 控件，您必须增加 COMCTL32.OCX 文件到您的目标文件中。当使用或出售您的应用程序时，必须把 COMCTL32.OCX 文件安装在 Microsoft Windows 的 System 目录或 System32 目录下。

以下是 ActiveX 大部分*.OCX 文件和控件的对应表格，请读者和开发人员注意使用的控件和需要注册的.OCX 文件。如表 8.1 所示。

表 8.1 不同控件的所属文件

文 件	控 件
	ImageList 控件
	ListView 控件
	ProgressBar 控件
COMCTRL32.OCX	Slider 控件
	StatusBar 控件
	TabStrip 控件
	Toolbar 控件
	TreeView 控件
COMDLG32.OCX	Common Dialogs 控件
DBLIST32.OCX	MSDataCombo 控件
	MSDataList 控件
FOXHWND.OCX	Visual FoxPro HWND 控件
FOXTLIB.OCX	Visual FoxPro Foxtlib 控件
GRID32.OCX	Grid 控件
MCI32.OCX	Microsoft Multimedia 控件
MSACAL70.OCX	Calendar 控件
MSCOMM32.OCX	Microsoft Comm 控件
MSMAPI32.OCX	Microsoft MAPI Message 控件
	Microsoft MAPI Session 控件
MSOUTL32.OCX	Outline 控件
PICCLP32.OCX	PicClip 控件
RICHTX32.OCX	Rich Textbox 控件
SYSINFO.OCX	SysInfo 控件

（续表）

文　件	控　件
TABCTL32.OCX	SSTab 控件
	Threed Checkbox 控件
	Threed Command Button 控件
THREED32.OCX	Threed Frame 控件
	Threed Group Push Button 控件
	Threed Option Button 控件
	Threed Panel 控件

8.1.2 ProgressBar 控件

ProgressBar 控件主要的目的是显示操作过程的进度，它在 Windows 桌面的表现形式是通过从左到右用颜色填充一个矩形区域。

1. ProgressBar 控件的属性

一个 ProgressBar 控件有以下几个关键的属性：

Max 属性和 Min 属性

Height 属性和 Width 属性

BorderStyle 属性

ProgressBar 控件具有一个范围和一个当前位置。范围反映了整个操作的全部过程，而当前位置表示过程的进展。ProgressBar 控件的 Max 属性和 Min 属性用来设置这个范围，而当前的位置是由 ProgressBar 控件的 Value 属性决定的。注意，Value 的值不能位于 Min 的值和 Max 的值之外。

ProgressBar 控件的 Height 属性和 Width 属性决定填充控件小块的尺寸和数目。小块的数量越多，操作过程的进展描述的就越精确。通过减小控件的 Height 属性或增加控件的 Width 属性可以增加显示的小块的数量。

PorgressBar 控件的 BorderStyle 属性也是一个比较关键的属性，对应 Windows 的编程，有很大的一部分代码是在绘制 Windows 的图形窗口，而 BorderStyle 属性恰恰是可以给该控件一个外观上的选择。

2. ProgressBar 控件的例子

以上讲述了 PorgressBar 控件的几个关键的属性，下面让我们通过设计一个简单的表单来说明如何使用 ProgressBar 控件。

该例子是通过一个时钟来模拟一个程序的进展，随着时间的增加，ProgressBar 控件将演示其反映过程发展情况的功能。

第一步：从表单控件中选中 ProgressBar 控件，绘制在表单上如图 8.12 所示。

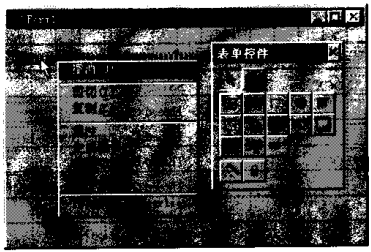


图 8.12 窗体中的 ProgressBar 控件

第二步：和 ImageList 控件的 General 属性的设置一样，利用鼠标的右键打开 ProgCtrl 属性窗口，如图 8.13 所示。

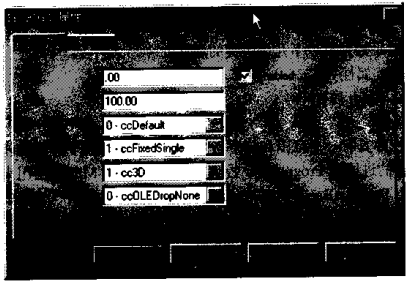


图 8.13 ProgressBar General 属性窗口

设置完该控件的关键属性后，主要的还是在程序的代码中来改变控件的 Value 属性，从而得到一个变化的、动态的过程条（ ProgressBar ）。

第三步：在表单中加入一个时钟控件和二个按钮控件，如图 8.13 所示，并分别给它们命名，如表 8.2 所示。

表 8.2 在表单中加入控件

时钟控件	Timer1
“开始”按钮	Start
“结束”按钮	End

其布局如图 8.14 所示。

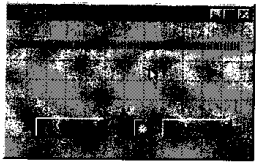


图 8.14　一个关于 ProgressBar 简单的例子

第四步：用鼠标双击 ProgressBar 控件，出现图 8.15 所示的代码编辑窗体，在该窗体中进行代码的录入，通过它使我们对事件响应程序编制代码变的更加容易、简单和可靠。

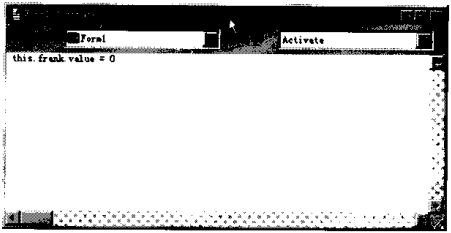


图 8.15　ProgressBar 例子的代码编辑框

图 8.15 中的“对象”列表框拉开后如图 8.16 所示。通过它来选取本表单中的不同对象；而“过程”列表框拉开后可以选择对应于不同事件的响应，如图 8.17 所示。

图 8.16　对象

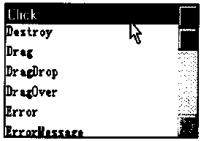


图 8.17　事件响应

第五步：在对象中选择 Start 对象，而在事件响应程序中选择 Click 事件，如图 8.17 所示。然后就可以在图 8.15 所示的代码编辑窗体中编辑程序的代码了。或者在表单的窗体上对“开始”按钮双击，同样会弹出对 Start 对象的 Click 响应程序进行编辑的窗体。

第六步：

编辑如下代码。

Start 按钮对 Click 的响应代码：

```
thisform.timer1.enabled=.t.   &&使 Timer1 时钟工作
```

End 按钮对 Click 的响应代码：

```
thisformt.timer1.enabled=.f.   &&使 Timer1 时钟停止工作
```

Timer1 时钟的 Timer 响应代码：

```
if thisform.frank.value <= 90.00
    thisform.frank.value=this.parent.frank.value + 10
else
    thisform.frank.value = 0.0
endif
```

Form1 窗体加载时 Load 的响应代码：

```
this.frank.value = 0
```

这样就基本上完成了该例子。请对 Timer1 要设置其 Interval 属性，其代表的意思是：间隔多长时间后响应 Timer 时钟的 Timer 事件。工作完成后，可以进行测试了，工具条中的“感叹号”和菜单“程序”中的“运行”都可以测试您 PorgressBar 应用程序的正确与否。结果呢？如图 8.18 所示。

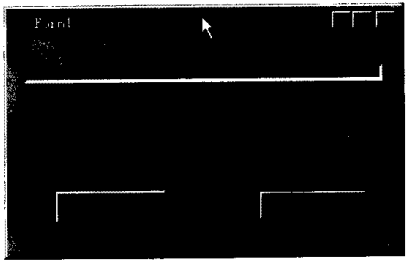


图 8.18 运行结果

8.1.3 Slider 控件

滑块（ Slider ）控件是一个滑道和一个相对滑标构成的窗体，您可以通过鼠标拖动滑标、用鼠标单击滑道的二侧或键盘来改变滑标的数值。

滑块控件可以是您很方便的选择在一个连续数值范围中的数值大小。例如，读者可以通过改变滑标的位置来控制一个图像在屏幕上显示的大小，而不用去录入一个数值，做到了 Windows 界面的直观性。滑块控件可以横向放置也可以纵向放置。

1. 滑块的属性

Name 属性
Min 属性和 Max 属性
SmallChange 属性和 LargeChange 属性
SetStart 属性
Value 属性

(1) Name 属性：

语法：

StatusBarName.Name = String

说明：对象用于区别其他对象的特定属性。这对任何一个控件来说，都是一个很关键的属性，建议用户最好给自己的控件创建一个有意义的名字。以后就不再叙述 Name 属性。

(2) Min 属性和 Max 属性：

语法：

StatusBarName.Min = *Value*

其中，Value 是一个整数。

说明：和 ProgressBar 控件相似，Min 属性控制着滑标的最小值，而 Max 属性控制滑标的最大值。滑标的 Value 属性值不能处在有 Min 属性和 Max 属性构成的数值范围之外。

(3) SmallChange 属性：

语法：

StatusBarName.Smallchange = Value

其中，Value 是一个整数。

说明：通过键盘来控制滑标时，每按一次 Right （Left）键或 Up （Down）键所要改变的滑标数值的大小。

(4) LargeChange 属性：

语法：

StatusBarName.LargeChange = *Value*

其中，Value 是一个整数。

说明：通过鼠标在滑道的二侧单击来改变滑标大小时，每单击一次滑标增大或减小的数值。

(5) SetStart 属性：

语法：

StatusBarName.SetStart = Value

其中，Value 是一个整数。

说明：设置滑标在初始时处在滑道的位置，即：滑标的初始值。

(6) Value 属性：

语法：

StatusBarName.Value = Value

其中的 Value 是一个整数。

滑标在滑道上的位置或滑标的数值。

以上的属性可以通过 Slider 控件的 General 属性页面来在设计阶段进行设置，如图 8.19 所示。

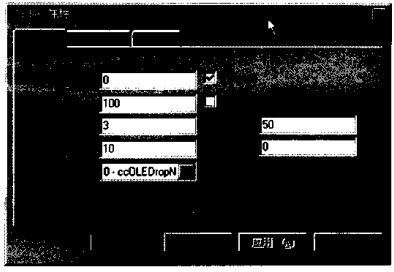


图 8.19 Slider 属性窗体

2. Slider 控件的一般响应事件

Scroll 事件：当您移动了滑块控件的滑标时，或用键盘控制滑块动作时，则 Windows 负责将该消息发送给 Scroll 事件响应程序的接口。

语法：

```
PROCEDURE ScrollName_Scroll( )
    [
        用户程序...
    ]
ENDPROC
```

需要注意的是：滑块响应事件是很多的，其中还有一个是 Click 响应事件，而这二个事件响应的顺序是 Scroll 响应事件先于 Click 响应事件。读者在以后的程序设计当中请注意这二种事件发生的顺序。

Scroll 响应事件将连续地返回 Slider 控件的 Value 属性数值。

3. Slider 控件例子

Slider 控件的例子，将在后面和 StatusBar 一起创建。请参考下一节（StatusBar）。

8.1.4 StatusBar（状态条）控件

状态条控件一般的位置是在一个窗口的最底部，而其提供的是多个小显示“窗口”，

通过它可以显示一些运行阶段有用的数据和用户关心的信息。StatusBar 控件最多可以在窗体底部显示 16 个消息“窗体”，其实，这种“窗体”是一个 Panels 的合集。

状态条控件是由 Panel 对象组成的，每个 Panel 对象都有其 Text 属性和 Picture 属性，通过它们可以控制单个 Panel 对象显示的宽度、布局方式（Text 和 Picture）。而且，系统提供了直接的方法，让读者可以很简单的在 StatusBar 控件中显示 7 种系统公用数据，例如：日期、时间和键盘的状态等等。而这些设置都是通过选择 StatusBar 控件中的 Style 属性来实现的。

在设计阶段，读者可以创建 Panel 对象并且通过设置 StatusBar 控件属性中的 Panel 表来定义 StatusBar 控件在应用程序中的表现，如图 8.20 所示。

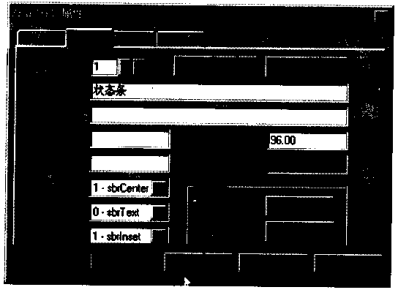


图 8.20　SBarCtrl Panel 属性窗体

在运行阶段，Panel 对象可以由一些不同的函数重新设置。详细的情况请参考 Panel 对象的说明。

StatusBar 控件一般显示窗体中对象的信息、对象的部件或和对象操作有关的、有前后关系的信息。它和 ToolBar 控件的组合，将使您创建 Windows 应用程序界面变得更加简单、精简、快速和实效。

1. StatusBar 控件的属性

StatusBar 控件的常用属性有以下几个：

Style 属性

Panels 属性

(1) Style 属性：

说明：　该属性返回一个 StatusBar 控件的类型

语法：

StatusBarName.Style = Value

表 8.3 StatusBar 类型的种类只有二种

常 量	数 值	说 明
sbrNormal	0	(缺省)正常。StautsBar 控件显示所有的 Panel
sbrSimple	1	简单。StatusBar 控件仅仅显示一个大的 Panel

状态条可以在二种模式 Normal 和 Simple 下转换，当在 Simple 模式下，StatusBar 控件仅仅显示一个 Panel 。StatusBar 控件的显示方式是可以改变的：Bevel 属性可以改变使显示成为浮雕形式而不具有边界。

当 StatusBar 控件的 Style 属性被设置为 Simple 时，可以用 SimpleText 属性来设置显示在 StatusBar 控件中的字符串。如图 8.21 所示的 StatusBar 控件的一般属性设置窗口。

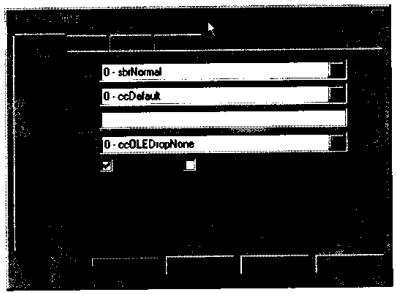


图 8.21 StatusBar 控件的一般属性

以上的二图（图 8.20 、图 8.21 ）分别是 StatusBar 控件的 General 属性窗体和其 Panels 属性窗体，通过这二个窗体，读者可以在设计阶段对 StatusBar 控件进行初始化。

(2) StatusBar 控件的 General 窗体就不再做讨论了。下来看看 StatusBar 控件的 Panels 窗体的一些属性：如图 8.20 所示。

1) Insert Panel 按钮：

通过它可以向 StatusBar 控件中加入需要的、可以反映必要信息的 Panel 。

2) Remove Panel 按钮：

删除不需要的 Panel 。

3) Index 属性：

反映了 StatusBar 控件中 Panels 合集中 Panel 的序列号，通过该属性，就可以对该 Index 对应的 Panel 进行操作。

4) Text 属性：

在 StatusBar 控件的 Panel 中显示的字符信息。

5)　Picuture 属性：

通过 Browser 按钮向 StatusBar 控件中加入一个图像， No Picture 按钮用来删除图像。

6)　ToolTipText 属性：

当鼠标移动到该 Panel 上，所要弹出的字符串。

7)　Style 属性：（这是 Panels 的 Style 属性，而不是 StatusBar 控件的 Style 属性）

该属性有 7 个选项，用来产生 7 个系统的信息，同时反映在 Panel 上。如图 8.22 所示。

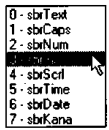


图 8.22　StatusBar 控件 Panels 窗体中的 Style 属性

2. StatusBar 控件的响应事件

StatusBar 控件的响应事件很多，在这里讲比较重要的二个： PanelClick 事件响应程序和 PanelDblClick 事件响应程序。

1.PanelClick 事件响应程序：

PanelClick 事件响应程序和一般正常的 Click 事件响应程序是一样的，也是在鼠标移动到 StatusBar 控件的 Panel 对象上，并按下、释放鼠标按键后产生的响应。

语法：

```
PROCEDURE StatusBarName_PanelClick(panel)
[
    用户定义程序...
]
ENDPROC
```

其中的参数 panel 是对一个 Panel 对象。

注意：正常的 Click 事件响应程序是在 Panel 对象被鼠标单击情况下发生的，当 StatusBar 控件的 Style 属性设置成为 Simple 时， Panel 将被隐藏，并且 PanelClick 事件响应程序不再对鼠标的按下而作出反映。

可以通过使用 Panel 对象的引用来设置该 Panel 对象的属性。下面的例子是用来改变被鼠标单击的 Panel 对象的 Bevel 属性。

例子：

```
PROCEDURE StatusBarName_PanelClick(Panel)
    Select Case Panel.Key
    Case "DisplayFileName"   && Key="DisplayFileName"
    Panel.Bevel = sbrRaised && 重新设置 Bevel 属性
    添加其他的用户操作过程...
```

```
    End Select
ENDPROC
```

2. PanelDblClick 事件响应程序：

PanelClick 事件响应程序和一般正常的 DbClick 事件响应程序是一样的，也是在鼠标移动到 StatusBar 控件的 Panel 对象上，并双击鼠标按键后产生的响应。

语法：

```
PROCEDURE StatusBarName_PanelDblClick(panel)
[
    用户定义过程...
]
ENDPROC
```

注意：参阅 PanelClick 事件响应程序。

3. StatusBar 控件和 Slider 控件的例程

下面给出 StatusBar 控件和 Slider 控件的组合例子，该例子的运行结果如图 8.23 所示。

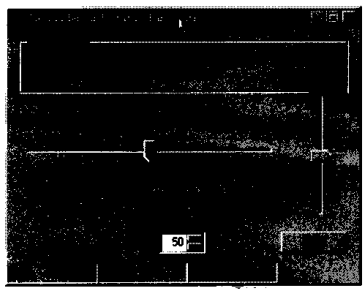


图 8.23 StatusBar 控件和 Slider 控件组合例子的运行结果

该例子中的 StatusBar 控件由 3 部分组成，如图 8.23 所示的窗体的最底部的 3 个 Panel 对象，而窗体中间和右侧的 Slider 控件分别为横向和纵向排列。窗体中还有一个 Spinner 对象，可以将 Slider 对象 Value 属性的数值反映给用户。下面，将给出该程序的设置和代码生成过程。

(1) 生成图 8.23 所示的窗体。

分别加入 Shape 控件、二个 Label 控件、一个 Botton 控件、二个 Slider 控件和一个 StatusBar 控件。

(2) 设置参数。

Label 控件 1：改变其 Caption 属性为“提示信息”。

Label 控件 2：更改 Caption 属性为“通过一个 Slider 控件来改变 Spinner 控件中的数据和另外一个 Slider 控件的数据显示”。

按图 8.19 中的设置改变您应用中 Slider 控件窗体的各个选项中的数据。

因为 StatusBar 控件的设置比较复杂，下面主要讲述关于 StatusBar 控件的技术：

1) 弹出图 8.20 所示的 Panels 设置窗体，按 Insert Panel 按钮，增加三个 Panel 对象，当 Index 属性为三时停止增加。

2) 在 Index 属性为 1 时的 Panel 对象的 Text 编辑框中输入“状态条”；在按下 Picture 属性框中的 Browser 按钮后，我们可以找到想要增加的图像，即：图 8.23 中的狐狸头。

3) 改变 Index 属性的数值为 2，把 Style 属性设置为 sbr-Num，则在 StatusBar 控件中会出现有关 NumLock 键盘的信息。

4) 改变 Index 属性的数值为 3，把 Style 属性设置为 sbr-Time，则在 StatusBar 控件中会出现有关系统时钟的信息。

(3) 编写代码。

1) Form 对象 Init 事件的代码：

```
* 判断是否 OCX 被安装和加载
IF TYPE("THIS.oleH") # "O" OR ISNULL(THIS.oleH)
    RETURN .F.
ENDIF

* 判断是否 OCX 被安装和加载
IF TYPE("THIS.oleV") # "O" OR ISNULL(THIS.oleV)
    RETURN .F.
ENDIF

* 判断是否 OCX 被安装和加载
IF TYPE("THIS.OleStatBar") # "O" OR ISNULL(THIS.OleStatBar)
    RETURN .F.
ENDIF
```

2) Spinner 对象 Init 事件代码。

```
THIS.Value = 50
```

3) Spinner 对象 InteractiveChange 事件代码。

```
#DEFINE TXT_LOC "slider value: "
THISFORM.OleStatBar.Panels(2).Text = TXT_LOC + ALLTRIM(STR(THIS.value))
THIS.Parent.oleH.Value = THIS.Value
THIS.Parent.oleV.Value = THIS.Value
```

4) Slider 对象 1 代码。

```
*** OLE 控件事件响应 ***
#DEFINE TXT_LOC "slider value: "
```

```
THISFORM.OleStatBar.Panels(2).Text = TXT_LOC + ALLTRIM(STR(THIS.value))
THIS.Parent.spn1.Value = THIS.Value
THIS.Parent.oleV.Value = THIS.Value
```

5) Slider 对象 2 代码。

```
*** OLE 控件事件响应 ***
#DEFINE TXT_LOC "slider value: "
THISFORM.OleStatBar.Panels(2).Text = TXT_LOC + ALLTRIM(STR(THIS.value))
THIS.Parent.spn1.Value = THIS.Value
THIS.Parent.oleH.Value = THIS.Value
```

(4) 结果。

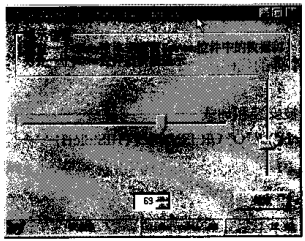


图 8.24 结果

8.1.5 Toolbar 控件

迄今为止，在 Windows 系统下的所有例程中都包含了工具条和菜单条，Visual FoxPro5.0 提供给广大用户一个方便有效的设计方法。本节将介绍如何为应用程序创建适合于自己的工具条。

Toolbar 对象是一种窗体，其中一些水平安排的图形按钮，这些图形按钮可以以成组的方式存在，并且可以由程序来设计成组方式。下图 8.25 是一个 Toolbar 控件的示意图。

图 8.25 ToolBar 控件窗体示意图

一个 Toolbar 控件包含了一个 Button 控件对象或其他控件对象的合集，而这些 Button 对象将和应用程序建立联系。Toolbar 控件的作用主要是对那些使用频率高的应用程序而言的，它通过使用小的图形界面使用户访问应用程序更加方便、直接。而且，Toolbar 控件一般都是和应用程序的菜单相联系的。

创建 Toolbar 控件对象的方法一般是这样的：给一个 Button 合集增加 Button 对象，然

后将这些 Button 合集传递给 Toolbar 控件。每一个 Button 对象有自己的文本和图形，而这些和我们第一节中的 ImageList 控件有很大的关系。读者可以通过使用 ImageList 控件的 Image 属性来给 Button 对象增加图形，或者，可以使用 Button 对象的 Caption 属性来设置显示给用户的文本。在设计阶段，您可以从 Toolbar 控件的属性窗口中增加 Button 对象到 Toolbar 控件之中。而在运行阶段，读者也可以增加或删除 Button 对象（对于 Buttons 合集而言）。

为了使 Toolbar 可以和应用程序结合起来，可以通过 ButtonClick 事件响应程序来对所选的 Button 对象作出反映。您可以通过 Style 属性来设置每个 Button 对象的行为和表现形式，例如：一个 ButtonGroup 中包含有四个 Button 对象，可以使得在任何时间有且仅有一个 Button 对象被按下。

有时候，可以在 Toolbar 控件对象上为别的控件预留空间，等到用户使用时再加入对象，该方法是通过在 Toolbar 控件对象上安置一个 PlaceHolder 形式的 Button 对象。以后将再用别的 Button 对象覆盖在该 Button 对象上。例如：在设计阶段时，在 Toolbar 控件对象上放置一个下拉式列表框，加入一个 PlaceHolder 形式的 Button 对象，该对象和列表框的尺寸一样，然后，在运行阶段或用户需要的情况下将列表框加载进来。

在运行阶段双击 Toolbar 对象可以激活 Customize Toolbar 对话框，通过它可以使用户隐藏、显示和重新排列 Toolbar 对象上的 Button 。为了满足不同用户的目的，可以使用 ALlowCustomize 方法来禁止该对话框。如果您想要存储和恢复 Toolbar 控件对象的状态或允许应用程序的用户这样做，那么 SaveToolbar 方法和 RestoreToolbar 方法可以满足您的要求。当 Toolbar 控件对象的形状改变时将引起 Change 事件响应程序，而该程序一般将调用 SaveToolbar 事件响应程序。

Customize 对话框包含了 Help 按钮，当用户需要设置 Help 文件时，请用 HelpFile 或 HelpContextID 来和您提供的帮助文件建立联系。 ToolTipText 属性将描述每个 Button 对象的功能，为了显示 ToolTipText 属性，必须设置 ShowTips 属性为.T.（真）。

1. Toolbar 控件属性和属性窗口

Toolbar 控件和别的控件一样，也有其属性窗口：如图 8.26 所示。

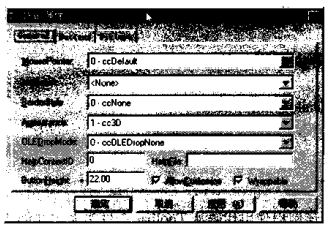


图 8.26　Toolbar 控件属性 General 窗口

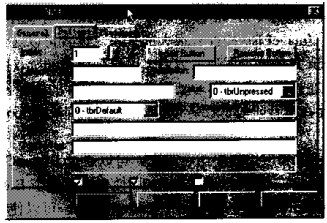


图 8.27 Toolbar 控件属性 Button 窗口

从 Toolbar 控件属性 General 窗体中可以看到 ImageList 对象和 Toolbar 对象的联系，可以在 ImageList 单选框中选中您的 ImageList 对象，然后用 Button 窗口的 Image 输入框中输入 ImageList 对象中图像的 Index 。参考 ImageList 控件章节。

2. Tollbar 控件的事件响应

ButtonClick 事件响应程序：
当用户鼠标单击 Toolbar 控件对象上的 Button 对象时将会例行该事件响应程序。
语法：

```
PROCEDURE ToolbarName_ButtonClick(button)
[
用户自定义程序...
]
```

以下是用 Button 对象的 Key 属性来控制应用程序的执行的例程：

```
PROCEDURE Toolbar1_ButtonClick(Button)
    Select Case Button.Key
    Case "Open"
        CommonDialog1.ShowOpen
    Case "Save"
        CommonDialog1.ShowSave
    End Select
ENDPROC
```

3. Toolbar 控件例程

例程（1）说明：该例程使用代码从 Toolbar 控件类派生并创建一个工具条。

```
PUBLIC tbrDesktop
tbrDesktop = CREATEOBJ('mytoolbar')
```

```
tbrDesktop.SHOW

DEFINE CLASS myToolBar      AS Toolbar
    ADD OBJECT btnBold      AS CommandButton
    ADD OBJECT sep1             AS Separator
    ADD OBJECT btnItalics AS CommandButton

    btnBold.HEIGHT = 20
    btnBold.WIDTH = 50
    btnBold.Caption = "Bold"
    btnItalics.HEIGHT = 20
    btnItalics.WIDTH = 50
    btnItalics.Caption = "Italic"
    btnItalics.FontBold = .F.

    LEFT    = 1
    TOP = 1
    WIDTH = 25

    CAPTION = "Desktop Attributes"

    PROCEDURE Activate
    this.btnBold.FontBold = _SCREEN.FONTBOLD
    this.btnItalics.FontItalic = _SCREEN.FONTITALIC
    ENDPROC

    PROCEDURE btnBold.CLICK
    _SCREEN.FONTBOLD = !_SCREEN.FONTBOLD
    This.FontBold =_SCREEN.FONTBOLD
    ENDPROC

    PROCEDURE btnItalics.CLICK
    _SCREEN.FONTITALIC = !_SCREEN.FONTITALIC
    This.FontItalic = _SCREEN.FONTITALIC
    ENDPROC
ENDDEFINE
```

例程（2）说明：

例程是用类设计器来设置一个 Toolbar 的子类，并通过调用子类来完成一个系统工具条

的设计。

(1) 使用类设计器设计可视类窗体

如图 8.28 所示，通过新建一个文件来创建一个可视类。

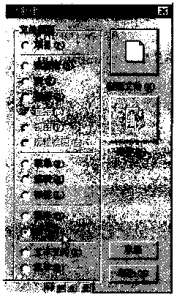


图 8.28 创建可视类

确定后出现如图 8.29 所示。

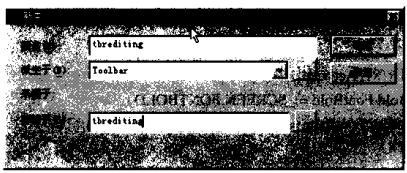


图 8.29 创建新类的类名和文件名

确定后将出现类设计器的窗体，如图 8.30 所示。

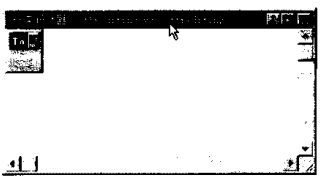


图 8.30 类设计器

在 Toolbar 控件中加入您想要加入的控件，如图 8.31 所示，本例程将加入三个 ComboBox

控件、和三个 CheckBox 控件。

图 8.31　该例程的 Toolbar 设计

(2)　使用类设计器设计可视类代码。

用鼠标双击 cboFonts 控件，在其 InteractiveChange 事件之中加入以下代码，如图 8.32 所示。

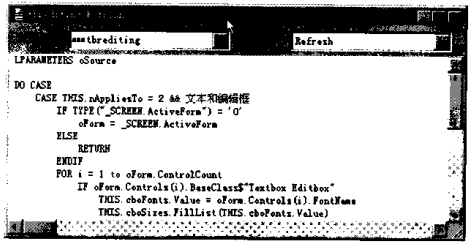


图 8.32　编辑程序代码

```
LOCAL oForm, oControl
IF TYPE("_SCREEN.ActiveForm") = "O" AND
TYPE("_SCREEN.ActiveForm.ActiveControl") = "O"
    oForm = _SCREEN.ActiveForm
    oControl = oForm.ActiveControl
ELSE
    RETURN
ENDIF

DO CASE
    CASE THIS.Parent.nAppliesTo = 1     &&当前控件
        oControl.FontName = This.Value

    CASE THIS.Parent.nAppliesTo = 2     &&对于所有的 EditBox 对象和 TextBox 控件
        oForm.SetAll('FontName', This.Value, 'TEXTBOX')
        oForm.SetAll('FontName', This.Value, 'EDITBOX')
```

```
    CASE THIS.Parent.nAppliesTo = 3     &&所有控件
        oForm.SetAll('FontName', This.Value)

ENDCASE

THIS.Parent.cboSizes.FillList(THIS.Value)
THIS.Parent.Refresh(oControl)
```

Toolbar 控件的 Refresh 事件：

```
LPARAMETERS oSource
DO CASE
    CASE THIS.nAppliesTo = 2 &&  文本和编辑框
        IF TYPE("_SCREEN.ActiveForm") = 'O'
            oForm = _SCREEN.ActiveForm
        ELSE
            RETURN
        ENDIF
        FOR i = 1 to oForm.ControlCount
            IF oForm.Controls(i).BaseClass$"Textbox Editbox"
                THIS.cboFonts.Value = oForm.Controls(i).FontName
                THIS.cboSizes.FillList(THIS.cboFonts.Value)
                THIS.cboSizes.Value = STR(oForm.Controls(i).FontSize)
                THIS.chkBold.Value = oForm.Controls(i).FontBold
                THIS.chkItalic.Value = oForm.Controls(i).FontItalic
                THIS.chkUnderline.Value = oForm.Controls(i).FontUnderline
                EXIT
            ENDIF
        ENDFOR
    OTHERWISE
        IF TYPE("oSource") != 'O'
            RETURN
        ENDIF
        THIS.cboFonts.Value = oSource.FontName
        THIS.cboSizes.FillList(THIS.cboFonts.Value)
        THIS.cboSizes.Value = STR(oSource.FontSize)
        THIS.chkBold.Value = oSource.FontBold
        THIS.chkItalic.Value = oSource.FontItalic
        THIS.chkUnderline.Value = oSource.FontUnderline
```

```
ENDCASE
```

在 Destroy 事件中加入以下代码：

```
This.Visible = .F.
```

cobSize 控件 Error 事件中加入以下代码：

```
    LPARAMETERS nError, cMethod, nLine
#define NUM_LOC "Error Number: "
#define PROG_LOC "Procedure: "
#define MSG_LOC "Error Message: "
#define CR_LOC CHR(13)
#define BADSIZE_LOC "The control doesn't support the selected fontsize."

DO CASE
    CASE nError = 1881 && 字体对于当前控件失效
        WAIT WINDOW BADSIZE_LOC TIMEOUT 2
        IF TYPE("Application.ActiveForm.ActiveControl") = "O"
            THIS.Parent.Refresh(Application.ActiveForm.ActiveControl)
        ENDIF

    OTHERWISE
        lcMsg = NUM_LOC + ALLTRIM(STR(nError)) + CR_LOC + CR_LOC + ;
                MSG_LOC + MESSAGE( )+ CR_LOC + CR_LOC + ;
                PROG_LOC + PROGRAM(1)
        lnAnswer = MESSAGEBOX(lcMsg, 2+48+512)
        DO CASE
            CASE lnAnswer = 3 &&放弃
                CANCEL
            CASE lnAnswer = 4 &&重试
                RETRY
            OTHERWISE
                RETURN
        ENDCASE
ENDCASE
```

在 InteractiveChange 事件中加入以下代码：

```
    IF TYPE("_SCREEN.ActiveForm") = 'O'
    oForm = _SCREEN.ActiveForm
ELSE
```

```
    RETURN
ENDIF

DO CASE
    CASE THIS.Parent.nAppliesTo = 1     && 当前控件
        oForm.ActiveControl.FontSize = VAL(THIS.Value)

    CASE THIS.Parent.nAppliesTo = 2     && 所有的文本和编辑框对象
        oForm.SetAll('FontSize', VAL(THIS.Value), 'TEXTBOX')
        oForm.SetAll('FontSize', VAL(THIS.Value), 'EDITBOX')

    CASE THIS.Parent.nAppliesTo = 3     &&所有控件
        oForm.SetAll('FontSize', VAL(THIS.Value))

ENDCASE
```

在 CheckboxBlod 对象中加入代码：

```
IF TYPE("_SCREEN.ActiveForm") = 'O'
    oForm = _SCREEN.ActiveForm
ELSE
    RETURN
ENDIF

DO CASE
    CASE THIS.Parent.nAppliesTo = 1     && 当前控件
        oForm.ActiveControl.FontBold = THIS.Value

    CASE THIS.Parent.nAppliesTo = 2     &&  所有的文本和编辑框对象
        oForm.SetAll('FontBold', THIS.Value, 'TEXTBOX')
        oForm.SetAll('FontBold', THIS.Value, 'EDITBOX')

    CASE THIS.Parent.nAppliesTo = 3     && 所有控件
        oForm.SetAll('FontBold', THIS.Value)
ENDCASE
```

在 CheckBoxItalic 对象中加入代码：

```
IF TYPE("_SCREEN.ActiveForm") = 'O'
    oForm = _SCREEN.ActiveForm
ELSE
```

```
    RETURN
ENDIF

DO CASE
    CASE THIS.Parent.nAppliesTo = 1    && 当前控件
        oForm.ActiveControl.FontItalic = THIS.Value

    CASE THIS.Parent.nAppliesTo = 2    && 所有的文本和编辑框对象
        oForm.SetAll('FontItalic', THIS.Value, 'TEXTBOX')
        oForm.SetAll('FontItalic', THIS.Value, 'EDITBOX')

    CASE THIS.Parent.nAppliesTo = 3    && 所有控件
        oForm.SetAll('FontItalic', THIS.Value)

ENDCASE
```

在对象 CheckBoxUnderLine 中加入代码：

```
IF TYPE("_SCREEN.ActiveForm") = 'O'
    oForm = _SCREEN.ActiveForm
ELSE
    RETURN
ENDIF

DO CASE
    CASE THIS.Parent.nAppliesTo = 1    && 当前控件
        oForm.ActiveControl.FontUnderline = THIS.Value

    CASE THIS.Parent.nAppliesTo = 2    && 所有的文本和编辑框对象
        oForm.SetAll('FontUnderline', THIS.Value, 'TEXTBOX')
        oForm.SetAll('FontUnderline', THIS.Value, 'EDITBOX')

    CASE THIS.Parent.nAppliesTo = 3    && 所有控件
        oForm.SetAll('FontUnderline', THIS.Value)
ENDCASE
```

在 CboColor 对象中加入代码：

```
IF TYPE("_SCREEN.ActiveForm") = 'O'
    oForm = _SCREEN.ActiveForm
    oControl = oForm.ActiveControl
```

```
ELSE
    RETURN
ENDIF
DO CASE
    CASE This.Value = 1
        RETURN
    CASE This.Value = 2 && 得到前景色
        nForeColor = GETCOLOR()
        IF nForeColor > -1
        DO CASE
            CASE THISFORM.nAppliesTo = 1 && 当前控件
                IF TYPE("oControl.ForeColor") = "N"
                    oControl.ForeColor = nForeColor
                ELSE
                    IF TYPE("oControl.ItemForeColor") = "N"
                        oControl.ItemForeColor = nForeColor
                    ENDIF
                ENDIF
            CASE THISFORM.nAppliesTo = 2 && 文本和编辑框
                oForm.SetAll('ForeColor', nForeColor, 'TEXTBOX')
                oForm.SetAll('ForeColor', nForeColor, 'EDITBOX')

            CASE THISFORM.nAppliesTo = 3 && 所有控件
                oForm.SetAll('ForeColor', nForeColor)
        ENDCASE
        ENDIF

    CASE This.Value = 3 && 得到背景色
        nBackColor = GETCOLOR()
        IF nBackColor > -1
        DO CASE
            CASE THISFORM.nAppliesTo = 1 && 当前控件
                IF TYPE("oForm.ActiveControl.BackColor") = "N"
                    oForm.ActiveControl.BackColor = nBackColor
                ELSE
                    IF TYPE("oForm.ActiveControl.ItemBackColor") = "N"
                        oForm.ActiveControl.ItemBackColor = nBackColor
                    ENDIF
                ENDIF
```

```
            CASE THISFORM.nAppliesTo = 2 &&  文本和编辑框
                oForm.SetAll('BackColor', nBackColor, 'TEXTBOX')
                oForm.SetAll('BackColor', nBackColor, 'EDITBOX')

            CASE THISFORM.nAppliesTo = 3 &&  所有控件
                oForm.SetAll('BackColor', nBackColor)
        ENDCASE
        ENDIF
ENDCASE
THIS.Value = 1
```

这样就完成了定义的新的 Toolbar 类，请保存。

(3) 设计表单程序。

在您的表单初始化时加入以下代码：

```
THIS.tbrEditing.Left = THIS.ShortForm.Left
THIS.tbrEditing.Top = THIS.ShortForm.Top - (THIS.tbrEditing.Height + 25)
THIS.ShortForm.Show
```

表单中 TextBox 对象的 GotFocus 事件中加入代码：

```
THISFORMSET.tbrediting.Refresh(THIS)
```

(4) 运行结果。

运行结果如图所示。出现下图的工具条。

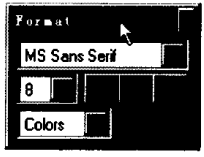


图 8.33　工具条例程结果

8.1.6 CommonDialog 控件

CommonDialog 控件提供了 Windows 系统的标准对话框，例如，打开文件、保存文件、设置打印机、选择字体和选择颜色。如图 8.34 和图 8.35 。该控件还可以在运行 Windows 帮助引擎情况下显示帮助信息。

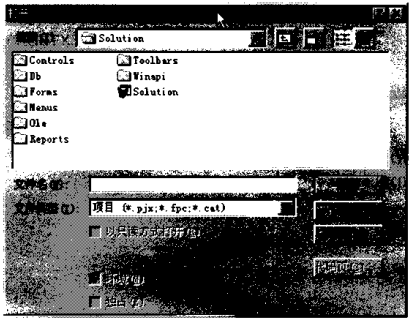


图 8.34 打开文件对话框(CommonDialog 对话框之一)

在 Microsoft Windows 动态连接库 Commdlg.dll 中为 CommonDialog 控件提供了 Windows 的编程界面。用该控件创建对话框时， Commdlg.dll 文件必须在您的 Microsoft Windows 的 System 目录或 System32 目录下。

读者可以在您的应用程序中提供加入 CommonDialog 控件来产生以上所说的几种标准对话框，但是，必须设置它的部分属性。并不是加入了 CommonDialog 控件后就可以显示想象中的对话框，这是需要调用控件的不同方法来实现的。在运行阶段，只有调用适当的方法，才可能出现您想要的对话框和运行帮助的引擎；在设计阶段， CommonDialog 控件以一个 Icon 图标显示在窗体上，而且该 Icon 是不能改变尺寸的。

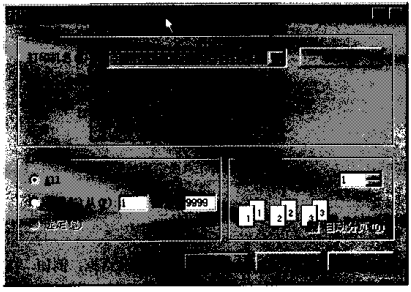


图 8.35 打印机设置对话框(CommonDialog 对话框之一)

CommonDialog 控件在采用不同的方法时可以显示以下几种对话框：

表 8.4　公用对话框的种类

方法	对话框类型
ShowOpen	显示打开文件 Dialog Box
ShowSave	显示保存文件 Dialog Box
ShowColor	显示颜色设置 Dialog Box
ShowFont	显示字体设置 Dialog Box
ShowPrinter	显示打印和打印机设置 Dialog Box
ShowHelp	激活 Windows 帮助引擎

1. 公用对话框的属性

公用对话框的几个一般属性：

Action 属性

DialogTitle 属性

DefaultExt 属性

Filter 属性

InitDir 属性

(1) Action 属性：

语法：

CommonDialogName.Action [= value]

说明：该属性在设计阶段不能使用，只能在运行阶段才能用来设置公用对话框的显示形式。所以，公用对话框的显示方式可以通过二种方法来实现：

1)Action 属性的设置。如表 8.5 所示。

2)通过调用不同的方法来显示不同的对话框。

表 8.5　Aciton 属性

数值	描　述
0	没有 Action 属性
1	文件打开对话框.
2	文件保存对话框.
3	颜色对话框.
4	字体对话框.
5	打印机对话框.
6	运行 WINHLP32.EXE.

(2) DialogTitle 属性：

语法：

CommonDialogName.DialogTitle [= title]

说明：返回和数值对话框标题栏中的字符串。

注意：当显示的对话框为 Color 对话框、Font 对话框和 Print 对话框时，CommonDialog 控件将忽略该设置。Open 对话框、Save As 对话框的缺省值分别为：Open 和 Save As 。

(3) DefaultExt 属性：

语法：

CommonDialogName.DefaultExt [= string]

说明：为对话框设置缺省的扩展文件名。

(4) Filter 属性：

语法：

CommonDialogName.Filter [= description1 |filter1 |description2 |filter2...]

description　显示表达文件类型的字符串

filter　特定的文件扩展名

说明：返回和设置对话框中文件类型列表框中的过滤条件。

例如：

CommonDlg1.Filter = Text (*.txt)|*.txt|Pictures (*.bmp;*.ico)|*.bmp;*.ico

(4) InitDir 属性：

语法：

CommonDialogName.InitDir [= string]

说明：返回和设置初始文件目录。

该属性被用在文件打开和文件保存对话框中，用来指定文件的目录，在没有给定该属性时，当前目录将被采用。

2. 公用对话框的方法

公用对话框的关键方法是：ShowOpen 方法，ShowSave 方法，ShowColor 方法，ShowPrinter 方法、ShowFont 方法和 ShowHelp 方法。这些方法很直观，这里不在叙述。

3. 公用对话框的例程

以下将要以文件打开对话框为例程讲述公用对话框的使用方法。

(1) 创建窗体。

如图 8.36 所示的窗体，其中有一个公用对话框控件，其余的都是基本的控件，包括有：四个 Check Box 控件、一个 Edit 控件、二个 Shape 控件和二个 Button 控件。

图 8.36　例程的外观设计

(2) 创建代码。

以下是文件代码，请按不同的子程序录入到合适的地方。（其中包括了窗口中控件的说明和属性）

```
**************************************************
*
DEFINE CLASS form1 AS form

DataSession = 2
Top = 3
Left = 4
Height = 226
Width = 423
DoCreate = .T.
Caption = "用户定义的文件打开对话框"
MaxButton = .F.
HelpContextID = 197
Name = "Form1"

ADD OBJECT shape1 AS shape WITH ;
    Top = 69, ;
    Left = 10, ;
    Height = 54, ;
    Width = 404, ;
    BackStyle = 0, ;
    SpecialEffect = 0, ;
    Name = "Shape1"

ADD OBJECT label2 AS label WITH ;
    AutoSize = .T., ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    Caption = "选项", ;
    Height = 15, ;
    Left = 18, ;
    Top = 63, ;
    Width = 44, ;
    TabIndex = 1, ;
    Name = "Label2"
```

```
ADD OBJECT behindscenes1 AS behindscenes WITH ;
    Top = 194, ;
    Left = 11, ;
    TabIndex = 8, ;
    Name = "Behindscenes1"

ADD OBJECT cmdclose2 AS cmdclose WITH ;
    Top = 194, ;
    Left = 342, ;
    TabIndex = 10, ;
    Name = "Cmdclose2"

ADD OBJECT chkread AS checkbox WITH ;
    Top = 79, ;
    Left = 115, ;
    Height = 15, ;
    Width = 118, ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    AutoSize = .T., ;
    Caption = "只读选项", ;
    Value = .T., ;
    TabIndex = 3, ;
    Name = "chkRead"

ADD OBJECT chkmulti AS checkbox WITH ;
    Top = 79, ;
    Left = 18, ;
    Height = 15, ;
    Width = 75, ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    AutoSize = .T., ;
    Caption = "打开多文件", ;
    Value = .F., ;
    TabIndex = 2, ;
    Name = "chkMulti"

ADD OBJECT olecommdlog AS olecontrol WITH ;
```

```
        Top = 191, ;
        Left = 78, ;
        Height = 100, ;
        Width = 100, ;
        Name = "oleCommDlog"

ADD OBJECT edtfiles AS editbox WITH ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        Height = 51, ;
        Left = 10, ;
        ReadOnly = .T., ;
        TabIndex = 7, ;
        Top = 134, ;
        Width = 404, ;
        Name = "edtFiles"

ADD OBJECT cmdfiles AS commandbutton WITH ;
        Top = 194, ;
        Left = 264, ;
        Height = 23, ;
        Width = 72, ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        Caption = "运行", ;
        TabIndex = 9, ;
        Name = "cmdFiles"

ADD OBJECT chkhelp AS checkbox WITH ;
        Top = 100, ;
        Left = 18, ;
        Height = 15, ;
        Width = 73, ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        AutoSize = .T., ;
        Caption = "帮助", ;
        Value = .F., ;
        TabIndex = 5, ;
```

```
    Name = "chkHelp"

ADD OBJECT chkfox AS checkbox WITH ;
    Top = 79, ;
    Left = 272, ;
    Height = 15, ;
    Width = 125, ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    AutoSize = .T., ;
    Caption = "Visual FoxPro 文件过滤器", ;
    Value = .T., ;
    TabIndex = 4, ;
    Name = "chkFox"

ADD OBJECT shape2 AS shape WITH ;
    Top = 12, ;
    Left = 10, ;
    Height = 46, ;
    Width = 404, ;
    BackStyle = 0, ;
    SpecialEffect = 0, ;
    Name = "Shape2"

ADD OBJECT label1 AS label WITH ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    WordWrap = .T., ;
    Caption = "用公用对话框来实现文件打开对话框 ", ;
    Height = 32, ;
    Left = 18, ;
    Top = 23, ;
    Width = 394, ;
    TabIndex = 0, ;
    Name = "Label1"

ADD OBJECT label5 AS label WITH ;
    AutoSize = .T., ;
    FontName = "MS Sans Serif", ;
```

```
    FontSize = 8, ;
    Caption = "提示信息", ;
    Height = 15, ;
    Left = 18, ;
    Top = 6, ;
    Width = 62, ;
    TabIndex = 0, ;
    Name = "Label5"

PROCEDURE Init
    *判断 OCX 是否被安装并加载
    IF TYPE("THIS.oleCommDlog") # "O" OR ISNULL(THIS.oleCommDlog)
        RETURN .F.
    ENDIF
ENDPROC

PROCEDURE cmdfiles.Click
    LOCAL nFlags
    nFlags = 0

    * 只读标记
    IF !thisform.chkRead.Value
        m.nFlags = m.nFlags + 4
    ENDIF

    * 多文件标记
    IF thisform.chkMulti.Value
        m.nFlags = m.nFlags + 512
    ENDIF

    * 帮助按钮标记
    IF thisform.chkHelp.Value
        m.nFlags = m.nFlags + 16
    ENDIF

    thisform.oleCommDlog.Flags = m.nFlags

    * Fox 过滤标记
    IF thisform.chkFox.Value
```

```
            thisform.oleCommDlog.FileName = "*.*"
            thisform.olecommdlog.filter="Allfiles(*.*)|*.*|Forms(*.scx)|
    &&*.scx|Reports(*.frx)|*.frx"
        ELSE
            thisform.oleCommDlog.FileName = "*.*"
            thisform.olecommdlog.filter="Allfiles(*.*)|*.*|Text*.txt)|*.txt|
        &&Pictures(*.bmp;*.ico)|*.bmp;*.ico"
        ENDIF

        * 显示对话框
        thisform.oleCommDlog.ShowOpen()

        IF thisform.oleCommDlog.FileName = "*.*"
            thisform.edtFiles.Value = ""
        ELSE
            thisform.edtFiles.Value = thisform.oleCommDlog.FileName
        ENDIF
    ENDPROC

    ENDDEFINE
    *
    *-- EndDefine: form1
    **************************************************
```

(3) 运行结果。

我们可以通过一个图像来看看例程的结果。如图 8.37 所示。我们看到了 Windows 系统下经常见到的文件打开对话框（选中例程所有的 Check Box 选项）。

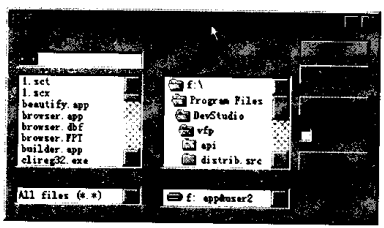


图 8.37 文件打开对话框例程结果

8.1.7 SysInfo 控件

为了使用户的应用程序能够更好的适应于操作系统，Microsoft 公司提供了 SysInfo 控件来将操作系统的信息传递给用户的应用程序，而用户应用程序可以根据不同的信息作出不同的响应。

采用 SysInfo 控件时，读者可以监控操作系统提供的信息，并且还可以响应系统产生的事件。该控件和系统的属性和方法组是有对应关系的：

(1) 和系统联系在一起的事件。如：DisplayChanged，TimeChanged，和 SettingChanged 事件。

(2) Power 状态事件和属性。如：PowerSuspend，PowerResume 事件和 ACStatus 以及 BatteryStatus 属性。

(3) Plug and Play 事件。如：DeviceArrival，DeviceRemoveComplete 事件。

(4) 操作系统属性。如：OSVersion 和 WorkAreaHeight 属性。

SysInfo 控件和 CommonDialog 对话框一样，在表单设计时表现出来的仅仅是一个 Icon 图标。SysInfo 控件的例程将放置在下一章（请参考 TreeView 控件）。

8.1.8 TreeView 控件

TreeView 控件显示了 Node 对象的分层目录结构，每个 Node 对象均由一个 Label 对象和其相关的位图组成。一个 TreeView 控件一般用来显示文件和目录结构、文档中的类层次、索引中的层次和其他具有分层目录结构的信息。如图 8.38 所示。

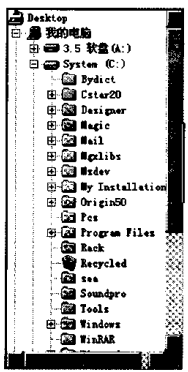


图 8.38　SysInfo 例图

当读者已经创建了一个 TreeView 控件后，您可以通过设置属性和调用方法来对 Node 对象进行增加、删除和其他操作。您还可以通过编程来对 Node 对象进行展开和折叠、显示或隐藏子节点。而这些操作是通过以下的三个函数实现的：

(1) Collapse 事件函数。

(2) Expand 事件函数。

(3) NodeClick 事件函数。

您可以通过代码来浏览 TreeView 控件对象的树形结构，而具体的实现方法是得到所要的 Node 对象，所有这些可以通过使用 Root 属性、Parent 属性、Child 属性、FirstSibling 属性、Next 属性等等。用户也可以通过使用键盘的 UP 和 DOWN 键来控制该操作...

TreeView 控件和 ImageList 控件结合使用，必须要详细的说明 TreeView 控件的 ImageList 属性。而 ImageList 控件的作用正如本章前面诉述，为了存储位图和 Icon 。一个 TreeView 控件一次只能用一个 ImageList 控件。这意味着 TreeView 控件中的每个项目都具有相同的图像尺寸。

1. TreeView 控件属性

TreeView 控件关键的属性有：

Nodes 属性

LineStyle 属性

Style 属性

(1) Nodes 属性：

语法：

TreeViewName.Nodes

说明：返回 TreeView 控件 Node 对象合集一个引用。

下面的例子是在 TreeView 中增加节点。

&& nodX 为 Node 控件对象

nodX = TreeView1.Nodes.Add(,,"R","Root") &&增加根节点

nodX = TreeView1.Nodes.Add("R", tvwChild,"C1","Child 1") &&增加子节点

(2) LineStyle 属性：

语法：

TreeViewName.LineStyle [= number]

说明：返回和设置相邻 Node 对象之间的联线类型。 number 数据的取值为表 8.6 所示，并且在设置该属性前必须设置 Style 属性为 TreeLine （三线）。

表 8.6 LineStyle 属性数值和意义

系统常量	数 值	描 述
tvwTreeLines	0	(缺省) 三线显示兄弟节点和父子节点之间的连线
tvwRootLines	1	根(Root)线。除了上述的连线外，还显示根目录之间的连线

(3) Style 属性：

语法：

TreeViewName.Style [= number]

说明：返回和设置表现在 TreeView 控件对象中 Node 对象的图形类型和文本类型。下表 8.7 为 Style 的 8 个类型：

表 8.7　Style 属性类型

数　值	描　述
0	文本
1	图像和文本
2	+/-以及文本
3	+/-、图像以及文本
4	连线和文本
5	连线、图像和文本
6	连线、+/-和文本
7	(缺省) 连线、+/-、图像和文本

以下是 TreeView 控件的属性控制窗口，如图 8.39 所示。

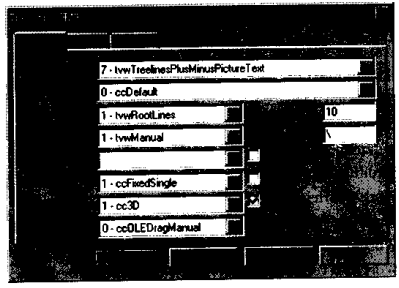


图 8.39　TreeView 控件属性窗体

通过该属性窗体将上述的属性更改，然后加入以下的代码。

2. 例程代码

请在不同控件对象的过程中加入相应的代码。

sysinfo.h 代码：

```
#DEFINE Yes_LOC                 [Yes]
#DEFINE No_LOC                  [No]
#DEFINE Unknown_LOC             [Unknown]
#DEFINE NA_LOC                  [Not Applicable]
#DEFINE High_LOC                [High]
#DEFINE Low_LOC                 [Low]
#DEFINE Critical_LOC            [Critical]
#DEFINE Charging_LOC            [Charging]
#DEFINE DeviceTypeOEM_LOC       [OEM-defined device type]
#DEFINE DeviceTypeDevNode_LOC   [Devnode number (Windows 95)]
#DEFINE DeviceTypeVolume_LOC    [Logical volume (disk drive)]
#DEFINE DeviceTypePort_LOC      [Serial or parallel port]
#DEFINE DeviceTypeNet_LOC       [Network resource]
#DEFINE PowerInfo_LOC           [AC/DC Power Information]
#DEFINE SystemInfo_LOC          [System Information]
#DEFINE DeskTopInfo_LOC         [Desktop Information]

#DEFINE ACStatus_LOC            [Using AC Power]
#DEFINE BatteryFullTime_LOC     [Full-Charge Battery Lifetime]
#DEFINE BatteryLifePercent_LOC  [Percentage Remaining Power]
#DEFINE BatteryStatus_LOC       [Status Of Battery]

#DEFINE OSPlatform_LOC          [Operating System]
#DEFINE OSVersion_LOC           [Version]
#DEFINE OSVerString_LOC         [Version Information]

#DEFINE ScrollBarSize_LOC       [Size Of Scrollbar]
#DEFINE WorkAreaHeight_LOC      [Desktop Work Area Height]
#DEFINE WorkAreaWidth_LOC       [Desktop Work Area Width]
#DEFINE WorkAreaLeft_LOC        [Left Edge Of Desktop]
#DEFINE WorkAreaTop_LOC         [Top Edge Of Desktop]

#DEFINE ConfigChanged_LOC       [System Configuration Has Changed]
#DEFINE ConfigChangeCancel_LOC  [Windows Has Notified Us That The Configuration
Change Has Been Canceled]
#DEFINE DeviceArrival_LOC       [A New Device Has Been Added To The System]
#DEFINE DeviceEventOther_LOC    [An Unhandled Device Event Occurred]
#DEFINE DeviceQueryRemove_LOC   [A Device Is About To Be Removed From The
System]
```

```
#DEFINE DeviceRemoveFailed_LOC  [The Device About To Be Removed Failed]
#DEFINE DeviceRemoveComp_LOC  [The Device Has Been Removed Successfully]
#DEFINE DeviceRemovePend_LOC  [The Device Is Being Removed]
#DEFINE DevModeChanged_LOC  [A Device Has Changed It's Mode]
#DEFINE DisplayChanged_LOC  [The Screen Resolution/Color Depth Has Changed]
#DEFINE PowerQuerySuspend_LOC  [The System Is Requesting The Suspension Of
Power]
#DEFINE PowerResume_LOC  [Power Has Been Resumed]
#DEFINE PowerStatusChanged_LOC  [Power Status Has Changed]
#DEFINE PowerSuspend_LOC  [Power Is Being Suspended]
#DEFINE QueryChangeConfig_LOC  [Windows Has Notified Us It Is About To Change
The Hardware Profile Configuration]
#DEFINE SettingChanged_LOC  [A System Wide Setting Has Changed]
#DEFINE SysColorsChanged_LOC  [A System Color Has Changed]
#DEFINE TimeChanged_LOC  [The System Date/Time Has Been Changed]
```

程序代码列表：

```
**************************************************
*
#INCLUDE "sysinfo.h"
*
DEFINE CLASS main AS form

DataSession = 2
Height = 355
Width = 427
DoCreate = .T.
ShowTips = .T.
AutoCenter = .T.
BorderStyle = 3
Caption = "显示系统信息"
MaxButton = .F.
MinButton = .F.
Icon = "..\"
HelpContextID = 186
Name = "main"

ADD OBJECT info AS olecontrol WITH ;
```

```
        Top = 49, ;
        Left = 9, ;
        Height = 268, ;
        Width = 408, ;
        Name = "Info"

    ADD OBJECT images AS olecontrol WITH ;
        Top = 316, ;
        Left = 101, ;
        Height = 37, ;
        Width = 37, ;
        Name = "Images"

    ADD OBJECT sysinfo AS olecontrol WITH ;
        Top = 316, ;
        Left = 57, ;
        Height = 37, ;
        Width = 37, ;
        Name = "SysInfo"

    ADD OBJECT shape3 AS shape WITH ;
        Top = 15, ;
        Left = 11, ;
        Height = 30, ;
        Width = 405, ;
        BackStyle = 0, ;
        SpecialEffect = 0, ;
        Name = "Shape3"

    ADD OBJECT label12 AS label WITH ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        WordWrap = .T., ;
        Caption = "该窗体使用 SysInfo OLE 控件来显示系统的信息", ;
        Height = 18, ;
        Left = 21, ;
        Top = 24, ;
        Width = 390, ;
        TabIndex = 0, ;
```

```
    Name = "Label12"

ADD OBJECT label13 AS label WITH ;
    AutoSize = .T., ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    Caption = " 提示信息 ", ;
    Height = 15, ;
    Left = 19, ;
    Top = 8, ;
    Width = 62, ;
    TabIndex = 0, ;
    Name = "Label13"

ADD OBJECT cmdclose1 AS cmdclose WITH ;
    Top = 323, ;
    Left = 343, ;
    Name = "Cmdclose1"

ADD OBJECT behindscenes1 AS behindscenes WITH ;
    Top = 323, ;
    Left = 12, ;
    Name = "Behindscenes1"

ADD OBJECT status AS label WITH ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    Caption = "", ;
    Height = 17, ;
    Left = 46, ;
    Top = 328, ;
    Width = 289, ;
    Name = "status"

PROCEDURE checkstatus
    ThisForm.Info.Nodes.Clear

    ThisForm.Info.Nodes.Add( , , [RootPower], PowerInfo_LOC, 2, 2)
    DO CASE
```

```
    CASE ThisForm.SysInfo.ACStatus = 0
        ThisForm.Info.Nodes.Add([RootPower], 4, [ACStatus], ACStatus_LOC + [ - ]
        + No_LOC, 2, 2)
    CASE ThisForm.SysInfo.ACStatus = 1
        ThisForm.Info.Nodes.Add([RootPower], 4, [ACStatus], ACStatus_LOC + [ - ]
        + Yes_LOC, 2, 2)
    CASE ThisForm.SysInfo.ACStatus = 255
        ThisForm.Info.Nodes.Add([RootPower], 4, [ACStatus], ACStatus_LOC + [ - ]
        + Unknown_LOC, 2, 2)
ENDCASE

IF ThisForm.SysInfo.BatteryFullTime =  -1
        ThisForm.Info.Nodes.Add([RootPower],      4,      [BatteryFullTime],
        BatteryFullTime_LOC + [ - ] + ;
        Unknown_LOC, 2, 2)
ELSE
        ThisForm.Info.Nodes.Add([RootPower],      4,      [BatteryFullTime],
        BatteryFullTime_LOC + [ - ] + ;

        ALLTRIM(STR(ThisForm.SysInfo.BatteryFullTime)), 2, 2)
ENDIF

IF ThisForm.SysInfo.BatteryLifePercent = 255
        ThisForm.Info.Nodes.Add([RootPower],      4,      [BatteryLifePercent],
        BatteryLifePercent_LOC + ;
        [ - ] + Unknown_LOC, 2, 2)
ELSE
        ThisForm.Info.Nodes.Add([RootPower],      4,      [BatteryLifePercent],
        BatteryLifePercent_LOC + ;
        [ - ] + ALLTRIM(STR(ThisForm.SysInfo.BatteryLifePercent)) + [%], 2, 2)
ENDIF

DO CASE
    CASE ThisForm.SysInfo.BatteryStatus = 1
        ThisForm.Info.Nodes.Add([RootPower],      4,      [BatteryStatus],
        BatteryStatus_LOC + ;
        [ - ] + High_LOC, 2, 2)
    CASE ThisForm.SysInfo.BatteryStatus = 2
        ThisForm.Info.Nodes.Add([RootPower],                              4,
```

```
        [BatteryStatus],BatteryStatus_LOC + ;
        [ - ] + Low_LOC, 2, 2)
    CASE ThisForm.SysInfo.BatteryStatus = 4
        ThisForm.Info.Nodes.Add([RootPower],        4,        [BatteryStatus],
        BatteryStatus_LOC + ;
                                    [ - ] + Critical_LOC, 2, 2)
    CASE ThisForm.SysInfo.BatteryStatus = 8
        ThisForm.Info.Nodes.Add([RootPower],        4,        [BatteryStatus],
        BatteryStatus_LOC + ;
        [ - ] + Charging_LOC, 2, 2)
    CASE ThisForm.SysInfo.BatteryStatus = 128
        ThisForm.Info.Nodes.Add([RootPower],        4,        [BatteryStatus],
        BatteryStatus_LOC + ;
        [ - ] + NA_LOC, 2, 2)
    CASE ThisForm.SysInfo.BatteryStatus = 255
        ThisForm.Info.Nodes.Add([RootPower],        4,        [BatteryStatus],
        BatteryStatus_LOC + ;
        [ - ] + Unknown_LOC, 2, 2)
ENDCASE

ThisForm.Info.Nodes.Add( , , [RootSystemInfo], SystemInfo_LOC, 1, 1)
DO CASE
    CASE ThisForm.SysInfo.OSPlatform = 0
        ThisForm.Info.Nodes.Add([RootSystemInfo],   4,        [OSPlatform],
        OSPlatform_LOC + [ - Win32s], 1, 1)
    CASE ThisForm.SysInfo.OSPlatform = 1
        ThisForm.Info.Nodes.Add([RootSystemInfo],   4,        [OSPlatform],
        OSPlatform_LOC + [ - Windows 95], 1, 1)
    CASE ThisForm.SysInfo.OSPlatform = 2
        ThisForm.Info.Nodes.Add([RootSystemInfo],   4,        [OSPlatform],
        OSPlatform_LOC + [ - Windows NT], 1, 1)
ENDCASE
        ThisForm.Info.Nodes.Add([RootSystemInfo],   4,        [OSVersion],
        OSVersion_LOC + [ - ] + ;
        TRANSFORM(ThisForm.SysInfo.OSVersion, [998.99]), 1, 1)

*! All this stuff is in TWIPS... I believe the conversion would be # / 15 to get Pixels
ThisForm.Info.Nodes.Add( , , [RootDesktopInfo], DeskTopInfo_LOC, 1, 1)
```

```
            ThisForm.Info.Nodes.Add([RootDesktopInfo], 4, [ScrollBarSize],
            ScrollBarSize_LOC + [ - ] + ;
            TRANSFORM(ThisForm.SysInfo.ScrollBarSize/15, [999]), 1, 1)
            ThisForm.Info.Nodes.Add([RootDesktopInfo], 4, [WorkAreaHeight],
            WorkAreaHeight_LOC + [ - ] + ;
            TRANSFORM(ThisForm.SysInfo.WorkAreaHeight/15, [99999]), 1, 1)
            ThisForm.Info.Nodes.Add([RootDesktopInfo], 4, [WorkAreaWidth],
            WorkAreaWidth_LOC + [ - ] + ;
            TRANSFORM(ThisForm.SysInfo.WorkAreaWidth/15, [99999]), 1, 1)
            ThisForm.Info.Nodes.Add([RootDesktopInfo], 4, [WorkAreaLeft],
            WorkAreaLeft_LOC + [ - ] + ;
            TRANSFORM(ThisForm.SysInfo.WorkAreaLeft/15, [99999]), 1, 1)
            ThisForm.Info.Nodes.Add([RootDesktopInfo], 4, [WorkAreaTop],
            WorkAreaTop_LOC + [ - ] + ;
            TRANSFORM(ThisForm.SysInfo.WorkAreaTop/15, [99999]), 1, 1)
ENDPROC

PROCEDURE QueryUnload
    _SCREEN.Visible = .T.
ENDPROC

PROCEDURE Init
    * 判断是否 OCX 控件被加载
    IF TYPE("THIS.Info") # "O" OR ISNULL(THIS.Info)
        RETURN .F.
    ENDIF

    * 判断是否 OCX 控件被加载
    IF TYPE("THIS.Images") # "O" OR ISNULL(THIS.Images)
        RETURN .F.
    ENDIF

    *判断是否 OCX 控件被加载
    IF TYPE("THIS.SysInfo") # "O" OR ISNULL(THIS.SysInfo)
        RETURN .F.
    ENDIF
```

```
    ThisForm.CheckStatus
ENDPROC

PROCEDURE Activate
    THIS.c_solutions1.saveHelp
ENDPROC

PROCEDURE Deactivate
    IF TYPE("THIS.c_solutions1") = "O" THEN
        THIS.c_solutions1.restoreHelp
    ENDIF
ENDPROC

PROCEDURE info.LostFocus
    ON KEY LABEL F1
ENDPROC

PROCEDURE info.GotFocus
    ON KEY LABEL F1 HELP ID _SCREEN.ActiveForm.HelpContextID
ENDPROC

PROCEDURE sysinfo.ConfigChangeCancelled
    *** OLE 控件事件响应程序***
    ThisForm.Status.Caption = ConfigChangeCancel_LOC
ENDPROC

PROCEDURE sysinfo.ConfigChanged
    *** OLE 控件事件响应程序 ***
    LPARAMETERS oldconfignum, newconfignum

    ThisForm.Status.Caption = ConfigChanged_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DeviceArrival
    *** OLE 控件事件响应程序 ***
    LPARAMETERS devicetype, deviceid, devicename, devicedata
```

```
    ThisForm.Status.Caption = DeviceArrival_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DeviceOtherEvent
    *** OLE 控件事件响应程序 ***
    LPARAMETERS devicetype, eventname, datapointer

    ThisForm.Status.Caption = DeviceEventOther_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DeviceQueryRemove
    *** OLE 控件事件响应程序 ***
    LPARAMETERS devicetype, deviceid, devicename, devicedata, cancel

    ThisForm.Status.Caption = DeviceQueryRemove_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DeviceQueryRemoveFailed
    *** OLE 控件事件响应程序 ***
    LPARAMETERS devicetype, deviceid, devicename, devicedata

    ThisForm.Status.Caption = DeviceRemoveFailed_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DeviceRemoveComplete
    *** OLE 控件事件响应程序***
    LPARAMETERS devicetype, deviceid, devicename, devicedata

    ThisForm.Status.Caption = DeviceRemoveComp_LOC
    ThisForm.CheckStatus
ENDPROC
PROCEDURE sysinfo.DeviceRemovePending
    *** OLE 控件事件响应程序***
    LPARAMETERS devicetype, deviceid, devicename, devicedata
```

```
    ThisForm.Status.Caption = DeviceRemovePend_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DevModeChanged
    *** OLE 控件事件响应程序 ***
    LPARAMETERS devicename

    ThisForm.Status.Caption = DevModeChanged_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.DisplayChanged
    *** OLE 控件事件响应程序***
    ThisForm.Status.Caption = DisplayChanged_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.PowerQuerySuspend
    *** OLE 控件事件响应程序 ***
    LPARAMETERS cancel

    ThisForm.Status.Caption = PowerQuerySuspend_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.PowerResume
    *** OLE 控件事件响应程序 ***

    ThisForm.Status.Caption = PowerResume_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.PowerStatusChanged
    *** OLE 控件事件响应程序***
    ThisForm.Status.Caption = PowerStatusChanged_LOC
    ThisForm.CheckStatus
ENDPROC
```

```
PROCEDURE sysinfo.PowerSuspend
    *** OLE 控件事件响应程序***

    ThisForm.Status.Caption = PowerSuspend_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.QueryChangeConfig
    *** OLE 控件事件响应程序 ***
    LPARAMETERS cancel

    ThisForm.Status.Caption = QueryChangeConfig_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.SettingChanged
    *** OLE 控件事件响应程序 ***
    LPARAMETERS item

    ThisForm.Status.Caption = SettingChanged_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.SysColorsChanged
    *** OLE 控件事件响应程序***

    ThisForm.Status.Caption = SysColorsChanged_LOC
    ThisForm.CheckStatus
ENDPROC

PROCEDURE sysinfo.TimeChanged
    *** OLE 控件事件响应程序***

    ThisForm.Status.Caption = TimeChanged_LOC
    ThisForm.CheckStatus
ENDPROC
ENDDEFINE
```

3. 运行结果

下图 8.40 是该例程运行的结果。

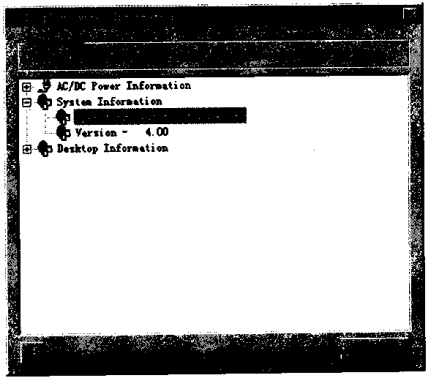


图 8.40　SysInfo 和 TreeView 控件例程运行结果

8.1.9 Multimedia MCI 控件

多媒体 MCI 控件控制着媒体控制接口设备上的多媒体文件的记录和回放。该控件具有一系列的按钮来向多媒体设备发送 MCI 命令，以下将要学习一个通过 MCI 控件来访问 Windows(*.avi)文件的例子：

当读者在设计阶段向您设计的窗体中增加多媒体 MCI 控件后，您会看到如图 8.41 所示的多媒体命令按钮。

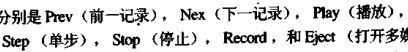


图 8.41　多媒体控件目录按钮

以上的按钮分别是 Prev（前一记录），Nex（下一记录），Play（播放），Pause（暂停），Back（后退），Step（单步），Stop（停止），Record，和 Eject（打开多媒体设备）。

而且可以由用户定义按钮组的构成，这是通过 Multimedia MCI 控件属性的 Control 窗体来定义的，如图 8.42 所示。

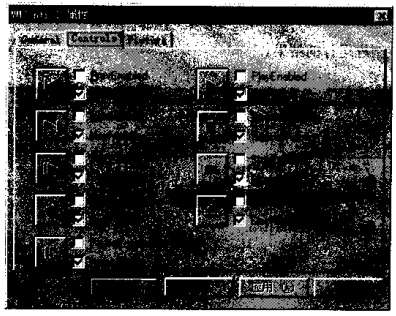


图 8.42 Multimedia MCI 控件属性的 Control 窗体

对于不同的媒体，MCI 的设备是不同的，该属性是由 Multimedia MCI 控件的 General 属性窗体来设置的，如图 8.43 所示。

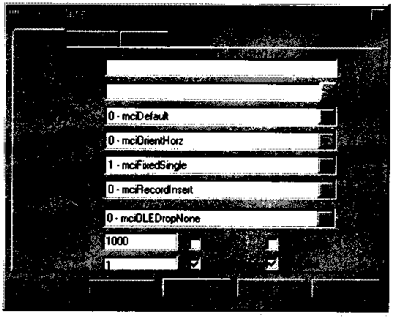


图 8.43 Multimedia MCI 控件的 General 属性窗体

多媒体 MCI 控件常用属性：

(1) hWndDisplay 属性：

语法：

[FromName.]MMControl.hWndDisplay

说明：详细的指出了 MCI 多媒体电影和其他设备的输出窗口，该属性只能在运行阶段使用。

(2) DeviceType 属性：

说明：明确指出了 MCI 的类型。

语法：

[FormName.]MMControl.DeviceType[= device$]

其中，device$可以是以下的几种 MCI 设备：AVIVideo，CDAudio，DAT，DigitalVideo，MMMovie，Other，Overlay，Scanner，Sequencer，VCR，Videodisc 或 WaveAudio。

多媒体 MCI 控件的例程：

该例程将和 FoxHWND 控件例程一起在下一节讲到。

8.1.10 HWND 控件

HWND ActiveX 控件提供了一个窗口，其他 ActiveX 控件（例如 Microsoft Multimedia ActiveX 控件）可以在这个窗口中显示图像。与 Visual FoxPro 图像控件不同，HWND 控件可以用来显示图标和 Windows 元文件。

1. HWND 控件的属性

(1) Appearance 属性：

说明：指定 HWND 控件的外观（在设计时和运行时可用）。

语法：

HWNDName.Appearance = Value

Appearance 属性具有下列值：0 -（默认）平面

1 - 三维

(2) BorderStyle 属性：

说明：指定 HWND 控件的边框样式（在设计时和运行时可用）。

语法：

HWNDName.BorderStyle = Value

BorderStyle 属性具有下列值：0 -（默认）无边框

1 - 单线边框

(3) Picture 属性：

说明：指定在 HWND 控件中所显示图像的图片对象引用。（在设计时和运行时可用）使用 LOADPICTURE( ) 可以创建位图、图标或 Windows 元文件的对象引用。

语法：

HWNDName.Picutre = Image

其中，Image 是一个图像的引用。

(4) HWnd 属性：

说明：包含 HWND 控件窗口的句柄。仅在运行时只读。

语法：

HWNDName.HWnd = Object.HWnd

2. HWND 控件的事件

HWND 控件的事件响应程序：

(1) Click 事件：当鼠标指针放在 HWND 控件上，并且用户按下并释放左键时发生。

(2) DblClick 事件：当鼠标指针放在 HWND 控件上，并且用户连续两次快速按下并释放左键时发生。

(3) KeyDown 事件和 KeyUp 事件：当 HWND 控件获得焦点，并且用户按下或释放一个键时发生。

(4) KeyPress 事件：当 HWND 控件获得焦点，并且用户按下并释放一个键时发生。

(5) MouseDown 事件和 MouseUp 事件：当鼠标指针位于 HWND 控件上，并且用户按下或释放一个鼠标键时发生。

(6) MouseMove 事件：当用户在 HWND 控件上移动鼠标时发生。当鼠标指针在 HWND 控件上经过时，MouseMove 事件是连续触发的。

(7) Paint 事件：当 HWND 控件需要重画时发生。在 Paint 事件代码中执行 RestoreDisplay 方法程序。

3. HWND 控件的方法

(1) RestoreDisplay 方法程序：恢复图像，该图像使用 SaveDisplay 方法程序保存在内存中。

(2) SaveDisplay 方法程序：将当前显示的图像保存在内存中。注意，如果一个工具栏或表单的 AlwaysOnTop 属性设置为“真”(.T.)，则当执行 SaveDisplay 方法程序时，该工具栏或表单位于 HWND 控件之上。工具栏或表单的图像与 HWND 控件中的图像一起保存。

4. HWND 控件和 Multimedia MCI 控件例程

第一步：设计窗体的布局。如图 8.44 所示。

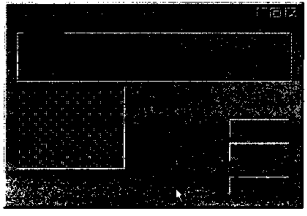


图 8.44 窗体的布局

其中鼠标所指的地方为 MCI 控件。以下为不同的事件响应程序的代码，请读者将它们录入的不同的事件响应程序内部。

第二步：编写代码。代码如下所示。

mulitmedia.h 文件代码：

```
#DEFINE MMEvent_Click_LOC        "Click Event Called"
#DEFINE MMEvent_DblClick_LOC     "DblClick Event Called"
#DEFINE MMEvent_KeyDown_LOC          "KeyDown Event Called"
#DEFINE MMEvent_KeyPress_LOC     "KeyPress Event Called"
#DEFINE MMEvent_KeyUp_LOC            "KeyUp Event Called"
#DEFINE MMEvent_MouseDown_LOC      "MouseDown Event Called"
#DEFINE MMEvent_MouseMove_LOC      "MouseMove Event Called"
#DEFINE MMEvent_MouseUp_LOC        "MouseUp Event Called"
#DEFINE MMEvent_Paint_LOC        "Paint Event Called"
```

程序代码：

```
*****************************************************
*
#INCLUDE " mulitmedia.h"
*
DEFINE CLASS form1 AS form

DataSession = 2
Height = 216
Width = 346
DoCreate = .T.
AutoCenter = .T.
BorderStyle = 3
Caption = "Play an AVI file in an ActiveX control "
MaxButton = .F.
HelpContextID = 202
Name = "Form1"

ADD OBJECT command2 AS commandbutton WITH ;
    Top = 114, ;
    Left = 263, ;
    Height = 23, ;
    Width = 72, ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
```

```
        Caption = "播放", ;
        Name = "Command2"

    ADD OBJECT vcr AS olecontrol WITH ;
        Top = 182, ;
        Left = 35, ;
        Height = 24, ;
        Width = 204, ;
        Visible = .F., ;
        Name = "VCR"

    ADD OBJECT command3 AS commandbutton WITH ;
        Top = 142, ;
        Left = 263, ;
        Height = 23, ;
        Width = 72, ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        Caption = "停止播放", ;
        Name = "Command3"

    ADD OBJECT shape3 AS shape WITH ;
        Top = 12, ;
        Left = 10, ;
        Height = 58, ;
        Width = 326, ;
        BackStyle = 0, ;
        SpecialEffect = 0, ;
        Name = "Shape3"

    ADD OBJECT label7 AS label WITH ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        WordWrap = .T., ;
        BackStyle = 0, ;
        Caption = "FoxHWND 控件和多媒体控件的应用范例", ;
        Height = 17, ;
        Left = 19, ;
        Top = 32, ;
```

```
    Width = 307, ;
    TabIndex = 0, ;
    Name = "Label7"

ADD OBJECT label8 AS label WITH ;
    AutoSize = .T., ;
    FontName = "MS Sans Serif", ;
    FontSize = 8, ;
    Caption = "提示信息", ;
    Height = 15, ;
    Left = 18, ;
    Top = 6, ;
    Width = 50, ;
    TabIndex = 0, ;
    Name = "Label8"

ADD OBJECT cmdclose AS cmdclose WITH ;
    Top = 182, ;
    Left = 263, ;
    Caption = "关闭", ;
    Name = "Cmdclose"

ADD OBJECT paper AS olecontrol WITH ;
    Top = 77, ;
    Left = 11, ;
    Height = 96, ;
    Width = 128, ;
    Name = "Paper"

PROCEDURE Init
    *Multimedia MCI 控件例程
    *用 MultiMedia Player 控件播放 AVI 文件

    *判断是否 OCX 控件被安装和加载
    IF TYPE("THIS.VCR") # "O" OR ISNULL(THIS.VCR)
        RETURN .F.
    ENDIF
    IF TYPE("THIS.PAPER") # "O" OR ISNULL(THIS.PAPER)
        RETURN .F.
```

```
    ENDIF

    ThisForm.VCR.hWndDisplay = ThisForm.Paper.hWnd
    ThisForm.VCR.UpdateInterval = 200
    ThisForm.VCR.TimeFormat = 3
    ThisForm.VCR.DeviceType = 'AVIVideo'
    cFileName = '..\'
    ThisForm.VCR.FileName = cFileName + 'FoxRain.avi' &&FoxRain.avi 所在目录
    ThisForm.VCR.Shareable = .F.
    ThisForm.VCR.Command = 'Open'
    SET PALETTE OFF

    Application.AutoYield = .F.
ENDPROC

PROCEDURE Destroy
    Application.AutoYield = .T.
ENDPROC

PROCEDURE command2.Click
    *播放 AVI 文件
    ThisForm.VCR.Command = "PLAY"
ENDPROC

PROCEDURE vcr.Done
    *** OLE 控件事件响应程序 ***
    LPARAMETERS notifycode

        IF notifycode = 1
        This.Object.From = 0
        This.Object.Command = "SEEK"
        This.Object.Command = "PLAY"
    ENDIF
ENDPROC

PROCEDURE command3.Click
    ThisForm.VCR.Command = "STOP"
    ThisForm.Paper.SaveDisplay
ENDPROC
```

```
PROCEDURE paper.Paint
    *** OLE 控件响应程序***
    This.RestoreDisplay
ENDPROC

ENDDEFINE
```

第三步：运行程序

程序运行结果如图 8.45 所示。

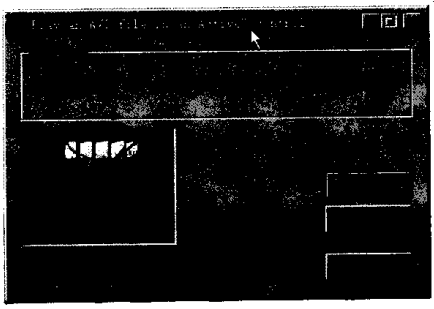


图 8.45　程序运行结果

8.1.11　RichTextBox 控件

Microsoft 在给出 TextBox 控件后遇到了一个问题：TextBox 不能解决文件大于 64k 的问题。但是现在，RichTextBox 提供了解决的方法。

RichTextBox 控件中打开和保存的文件将被按照 RTF 文件格式和 ASCII 格式保存。您可以使用控件的方法：LoadFile 和 SaveFile 去读写文件，或通过控件的 SelRTF 和 TextRTF 属性来实现。

RichTextBox 控件支持对象的连接和嵌入，所有您可以在其中插入 Word 文档和 Excel 表格。打印所有文件或部分文件时，可以使用 RichTextBox 控件的 SelPrint 方法。由于 RichTextBox 控件是一个可以和数据绑定的控件，您可以和一个 Data 控件的 Binary 和 Memo 字段联系起来。

RichTextBox 控件在 Windows 窗体中的显示如图 8.46 所示。

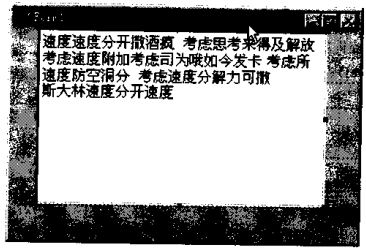


图 8.46 RichTextBox 控件在窗体中的形式

图中所示和 TextBox 控件很相象，的确，TextBox 控件的所有属性、方法和事件都可以用于 RichTextBox 控件中。

1. RichTextBox 控件的属性

如图 8.47 所示，该图是 RichTextBox 控件的属性窗体，由于其属性比较多，所以其含有五个 Group，下图是 General 窗体。

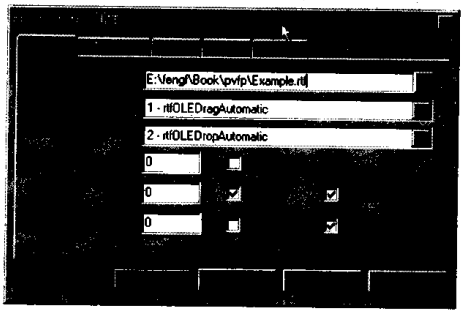


图 8.47 RichTextBox 控件属性的 General 窗体

(1) FileName 属性：指定了 RichTextBox 控件中加载的全文件名（包括路径和文件名）。
(2) MultiLine 属性：允许多行输入。
(3) OLEDragMode 属性：指定该控件是否允许对象的拖/拉操作。
在 RichTextBox 控件属性的 Apperance 窗体中，如图 8.48 所示。

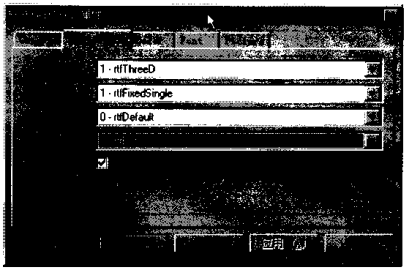


图 8.48　RichTextBox 控件属性的 Apperance 窗体

其 ScroollBars 属性是给该 RichTextBox 控件窗体加上滚动条。比较图 8.49 和图 8.46 的区别。

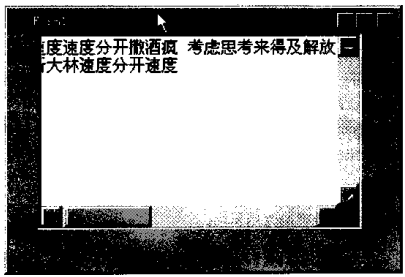


图 8.49　加入 ScrollBars 属性的 RichTextBox 例子

RichTextBox 控件的例程，简单的将 TextBox 控件相象，复杂的请参考 Visual FoxPro5.0 的例子。

8.1.12　Grid 控件

网格是一个容器对象，网格能包含列。这些列除了包含标头和控件外，每一个列还拥有自己的一组属性、事件和方法程序。网格对象能在表单中显示并操作行和列中的数据。使用网格控件的特别有用的应用程序是创建一对多表单，其中文本框显示父网格中的数据，而在子网格中显示子表的多个数据，如图 8.50 所示。

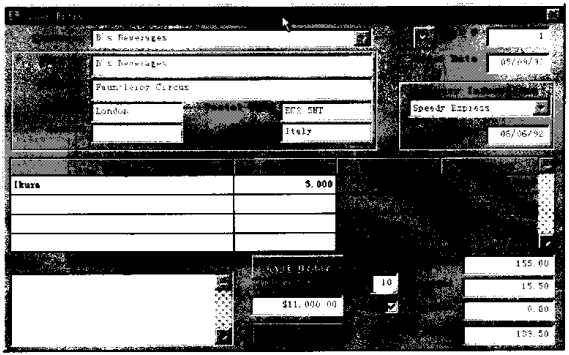


图 8.50 用网格显示父网格和子网格

读者可用从 Visual FoxPro 安装目录 Samples\Tastrade\找到该例程。

如何将网格控件添加到表单中？在“表单控件”工具栏中选择“网格”按钮，并在“表单”窗口中拖成所期望的大小。如果没有指定网格的 RecordSource 属性，但在当前工作区中有一个打开的表，那么网格将显示这个表的所有字段。

现在，我们将用 Visual FoxPro5.0 的网格生成器来生成一个网格，这样对不喜欢编程的人员来说，是一件很轻快的事情。

1. 网格的常用属性

下面表 8.8 列出了 Grid 控件的常用属性：

表 8.8 网格的常用属性

ColumnCount	决定显示的列数
ColumnOrder	设置列在网格中的顺序
RowHeigth	指出每行的高度
ChildOrder	列出于父网格相连的子网格的关键字
LinkMaster	列出主网格或父网格
RecordSource	为网格命名是数据源网格

2. 用网格生成器完成一个网格

第一步：如图 8.51 所示，在窗体中加入一个网格，并放大到用户感到合适的尺寸。

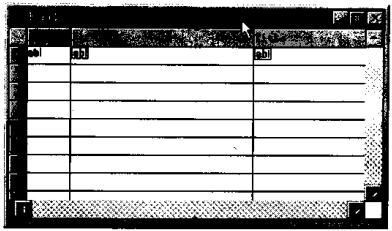


图 8.51　在窗体中加入网格

第二步：用鼠标右键单击 Grid 控件，在弹出菜单中选择“生成器”如图 8.52 所示。

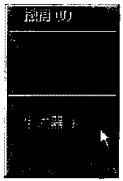


图 8.52　网格生成器弹出菜单

这将导致网格生成器窗体的弹出，如图 8.53 所示。

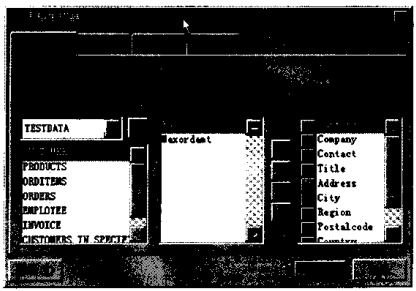


图 8.53　网格生成器窗体

其中，数据库和表选项提供给用户一个简单的方法去连接一个数据源，上图的表用的是 testdata.dbf 文件，数据源下面的列表框中为该数据源中的所有字段。可用字段为读者没有选中的字段，而选定字段就是读者选中在表中使用的字段。

读者也看到了该窗体有四个页组成，在“样式”中的内容是什么呢？如图 8.54 所示。

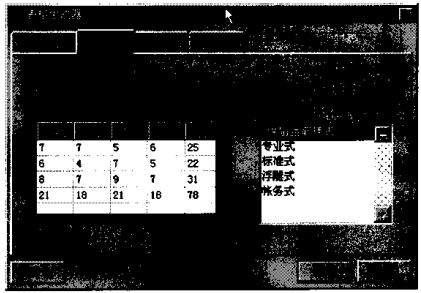


图 8.54 网格生成器样式窗体

该页面给出了网格的表现形式，读者可用逐一尝试，但一般来说，还是使用浮雕式，而且该样式被作为缺省值。

在样式旁边的布局页面中也同样有我们关心的属性设置，如图 8.55 所示。

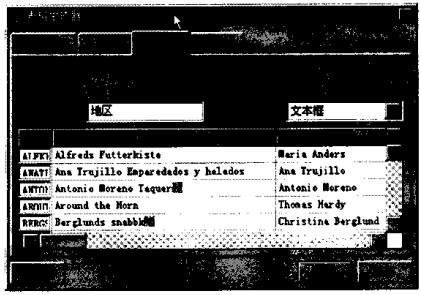


图 8.55 网格生成器布局窗体

该页面主要改变了在网格上的抬头标题，而这使数据库中的 Grid 控件给客户是一个醒目的表现。在关系页面中，主要建立了表和父表、子表之间的关系。

现在，让我们看一看生成的 Grid 控件，如图 8.56 所示。

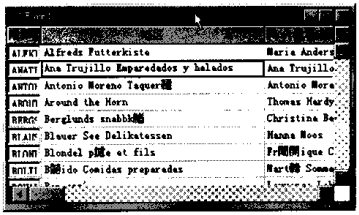


图 8.56　生成的 Grid 控件

很方便，而且很简单地就创建了 Grid 控件。但是，要在程序中使这些数据改变的话，如何操作呢？

3. 在网格中加入控件

读者到目前为止，看到的仅仅是显示数据的网格。在应用程序中，有可能会在网格中加入控件，这是怎么样实现的呢？让我们先看看具有控件网格的例子吧。运行 Visual FoxPro5.0 安装目录 Samples\solutions\Controls\Grid \下的 Grid 表单，可以看到如图 8.57 所示的网格，可以通过选定复选框使每一列可视或不可视。为了查看该表单的代码，可以在 Using Visual FoxPro Controls 表单中单击 See Code 框。

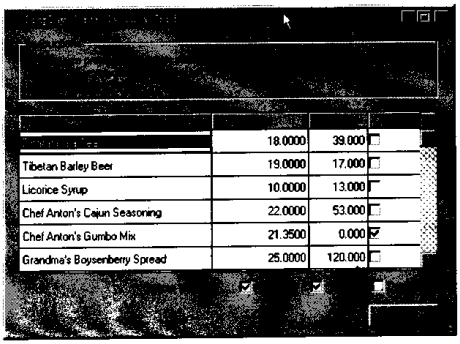


图 8.57　网格中加入控件

选定网格，并查看 Properties 窗口。Init 方法所包含的用户过程在第二列和第三列创建了微调控件，在第四列中创建了一个复选框。表单左下角的复选框用于决定这些控件是否是可视的。该过程的代码如下：

```
THIS.cloPrice.AddObject ('spnUnitPrice','SPINNER')
THIS.colStocked.AddObject ('spnStocked','SPINNER')
THIS.colDropped.AddObject ('chkDropped',"CHECKBOX')

THIS.colPrice.CurrentControl = 'spnUnitPrice'
THIS.colStocked.CurrentControl = 'spnStocked'
THIS.colDropped.CurrentControl = 'chkDropped'

THIS.colDropped.chkDropped.Visible = .T.
THIS.colDropped.chkDropped.Caption = ""

THIS.colPrice.spnUnitPrice.Visible = .T.
THIS.colPrice.spnUnitPrice.FontSize = 8

THIS.colStocked.spnStocked.Visible = .T.
THIS.colStocked.spnStocked.FontSize = 8
```

而下面的代码是各个控件的响应事件程序：

```
PROCEDURE Destroy
    THIS.DataChecker1.VerifyChanges
ENDPROC

PROCEDURE chkpricesparse.Click
    THISForm.Grid1.colPrice.Sparse = THIS.Value
ENDPROC

PROCEDURE chkstockedsparse.Click
    THISForm.Grid1.colStocked.Sparse = THIS.Value
ENDPROC

PROCEDURE chkdroppedsparse.Click
    THISForm.Grid1.colDropped.Sparse = THIS.Value
ENDPROC
```

4. 代码来改变 Grid 控件的数据

其实，这样的问题很好解决，当读者熟悉了采用网格生成器后，您会注意到，Grid 控件打开的表是和 Grid 控件的 RecordSource 属性相联系的。所以，只要改变 RecordSource 属性为指定的表；然后通过改变不同 Column 对象中联系的字段，就可用使用代码轻松地管理 Grid 控件中的数据了。

8.1.13　TabStrip 控件

TabStrip 控件象是一组文件夹，通过使用 TabStrip 控件，我们可以定义一个在相同区域内显示的对话框或窗体，如图 8.58 所示。

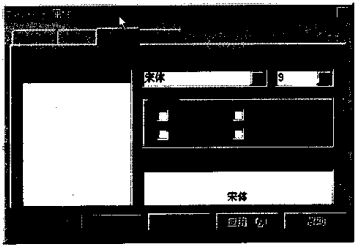


图 8.58　TabStrip 控件

该窗体中含有 General 页、Tabs 页（图 8.59 ）、Font 页和 Picture 页。读者对此已经很熟悉了，因为我们在前面用过很多次这样的窗体来设置控件的属性。

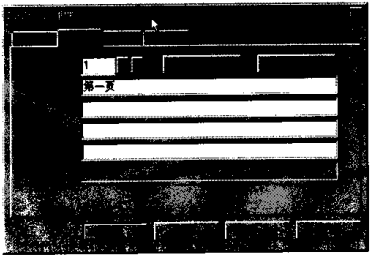


图 8.59　TabStrip 控件 Tabs 页

您可以在设计阶段和程序运行阶段来设置该控件的属性，从而来影响窗体的表现形式。该控件是由一个或多个在 Tabs 合集中的 Tab 对象组成的。在设计阶段，您可以通过 TapStrip 属性页来增加和删除 Tab 对象。同样，您可以在程序运行阶段通过使用合适的方法来添加和删除。

1. TabStrip 控件的属性

(1) Style 属性：

语法：

TabStripName.Style = Value

说明：Style 属性有二种：tabTabs 和 tabButtons，缺省的是 tabTabs。用 Style 属性可以决定 TabStrip 控件的显示形式，其显示的形式有：

1) 以 Button 形式显示出来，如图 8.60 所示。

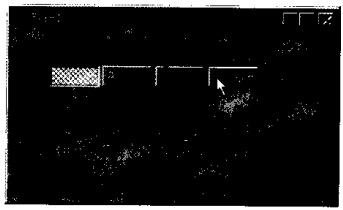


图 8.60 Button 形式显示出来的 TabStrip 控件

2) 以 Tabs 形式显示出来：如图 8.61 所示。

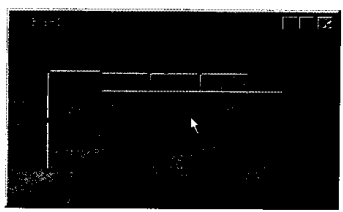


图 8.61 Tabs 形式显示出来的 TabStrip 控件

二图比较，我们可以发现，以 Button 类型显示的窗体中没有边框，而以 Tabs 类型显示的有自己的边框。

(2) MultiRow 属性：

语法：

TabStripName.MultiRow [= boolean]

其参数是一个布尔型变量，当为.T.时，允许多行显示，当为.F.时，不允许多行显示，如图 8.62 所示。

说明：MultiRow 属性设置是否 TabStrip 控件允许显示多行的 Tabs 。如图 8.62 所示。

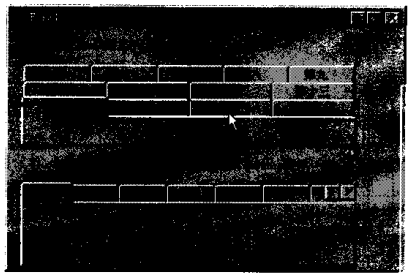


图 8.62　多行和非多行显示

当可以多行显示的话，行数将由 Tabs 的数目和 TabStrip 控件的宽度来自动调节。如果 TabStrip 控件的尺寸改变，则 Tabs 的行数将相应的要改变；当不允许多行显示时，在 Tabs 的右侧会出现向左和向右的箭头，这样也可以浏览所有的 Tabs 页面。

在设计决定，通过 TabStrip 属性的 General 页面来改变 MultiRow 属性。如图 8.63 所示。

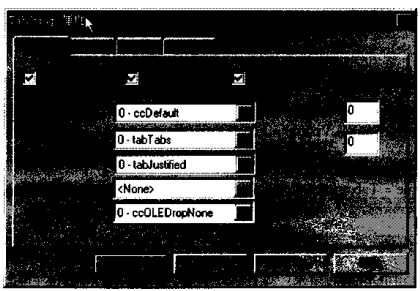


图 8.63　General 页面来改变 MultiRow 属性

在程序中也可以控制该属性，如下代码：

TabStripName.MultiRow = .T.

(3) TabWidthStyle 属性：

语法：

TabStripName.TabWidthStyle = Value

说明： TabWidthStyle 属性决定每一行中的显示，如果 TabWidthStyle 属性被设置为 tabFixed ，您可以用 TabFixedHeight 和 TabFixedWidth 属性将 TabStrip 控件中的所有 Tabs 设置为相同的高度和宽度。

(4) Tabs 属性：

语法：

TabStripName.Tabs(index)

其中 Index 将指出 Tab 对象在 Tabs 合集中的位置。

说明：返回一个对 TabStrip 控件的 Tab 合集的引用。每个 Tab 对象都有其当前状态和表现的属性。例如，您可以有 TabStrip 控件和一个 ImageList 控件相连系。而后，可以给每个 Tabs 对象分配一个图像。

(5) SelectedItem 属性

语法：

TabStripName.SelectedItem

说明：返回所选择的 ListItem 对象、 Node 对象或 Tab 对象的引用。

2. TabStrip 控件的说明

TabStrip 控件并不是一个容器类。所以，为了包含每个页面中属于它们的对象，您必须用 Frame 控件或其他容器类控件来包含这样不同页面中的对象。如果您使用的是容器类对象的数组，您可以为每个 Tab 对象分配一个对应于其的部分，请参考以下代码：

```
PROCEDURE Tabstrip1.Click
If Tabstrip1.SelectedItem.Index = mintCurFrame  Then EXIT PORC

&& 不需要改变 Frame ， 否则，隐藏 Frame ，显示另外一个
Frame1(Tabstrip1.SelectedItem.Index).Visible = True
Frame1(mintCurFrame).Visible = False

&& 设置 mintCurFrame 为另一新值
mintCurFrame = Tabstrip1.SelectedItem.Index
ENDPROC
```

3. TabStrip 例程

以下是 TabStrip 例程的结果，过程如上所述，通过改变属性和一部分代码即可以解决问题，请读者自己尝试。

例程的结果如图 8.64 和图 8.65

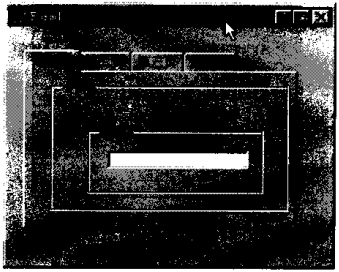


图 8.64　例程结果之一

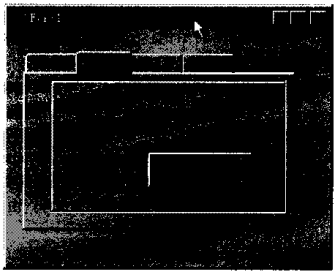


图 8.65　例程结果之二

8.1.14　Visual FoxPro Foxtlib 控件

Visual FoxPro5.0 中包含有 Foxtlib 控件（FOXTLIB.OCX），读者可以将这些控件添加到您的应用程序中，并一起发布。您可以使用 OLE 容器控件将 ActiveX 控件添加到应用程序中的表单中。

Foxtlib 控件允许您访问 Visual FoxPro5.0 应用程序中的类库信息。类库用来保存 OLE 服务程序的类信息（属性、方法程序，等等），您可以在 Visual FoxPro5.0 中创建 OLE 务程序。类库通常是通过“对象浏览器”查看的，并在其他应用程序（例如：Visual Basic 或 Visual C++）中引用。在这些应用程序中，允许进行直接的对象引用，并允许把类库编译到代码中。

当您在“项目管理器”中，如图 8.66 所示。

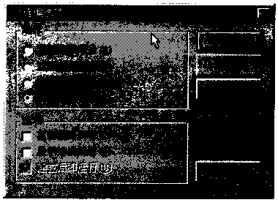


图 8.66 项目管理器中的项目联编窗体

使用创建连编 OLE DLL 或连编可执行程序命令创建一个 OLE 服务程序时，会生成一个类库（*.TLB ）文件。类库文件包含每个类的信息，例如：该类需要的参数个数以及数据类型，控件的返回值类型，对帮助主题或帮助文件的引用（可以提供有关类的更多信息），等等。

TYPELIB.VCX 是一个可视类库的示例，它使用 Foxtlib ActiveX 控件，安装在 SAMPLES\CLASSES 目录中。

下面列出了 Foxtlib ActiveX 控件可用的方法程序。

方法程序说明：

(1) TLLoadTypeLib(cTypeLibName)

该函数加载 cTypeLibName 指定的类型库文件。 cTypeLibName 必须是完全合法的路径和文件名。返回该类型文件的正整数句柄；如果不能加载类型库文件，则返回 0 值或一个负数。而 TLRelease(nTypeLibHandle) 函数释放加载的类型库，由 nTypeLibHandle 指定该类型库的句柄。

(2) TLGetTypeInfoCount(nTypeLibHandle)

该函数返回类型库中类信息的数目，由 nTypeLibHandle 指定该类型库的句柄。

(3) TLGetDocumentation(nTypeLibHandle,aDocArray, nMemberID, nType)

该函数创建一个具有三个元素的数组，该数组包含有关类型库的信息，由 nTypeLibHandle 指定该类库的句柄。由 ADocArray 指定数组名称。该数组包含成员的名称、文档字符串和帮助文件的内容，由 nMemberID 指定数组成员。 nType 指定成员类型：0 -TypeInfo 或 1 - Member 。

(4) TLGetTypeInfo(nTypeLibHandle,nTypeInfoNum)

该函数返回一个整数索引，该索引用于 TLGetTypeAttr( )、 TLGetTypeInfo( ) 和 TLGetFuncDesc( ) 方法程序。如果方法程序失败，则 TLGetTypeInfo( ) 返回 0 值。使用 TLGetTypeInfo( ) 可以逐一访问(从 0 开始)类型库中的所有类型信息，此时 nTypeLibHandle 指定该类型库的句柄， NTypeInfoNum 指定索引返回的类型信息。

(5) TLGetTypeAttr(nTypeInfoIndex,aTypeArray)

该函数创建一个一维数组，该数组包含有关的类型信息。 nTypeInfoIndex 指定类型信息的索引号。 aTypeArray 指定所创建数组的名称。下表 8.7 列出了数组中每个元素包含的信息：

表 8.9　数组中每个元素包含的信息

1	GUID
2	LocaleID
3	保留
4	Constructor ID
5	Destructor ID
6	保留
7	实例大小
8	类型
9	函数个数
10	变量/成员个数
11	完成界面的个数
12	类虚拟函数表大小
13	此类型实例的字节对齐
14	标识
15	主版本号
16	次版本号

(6) TIGetFuncDesc(nTypeInfoIndex,aTypeArray, nFunctionIndex, aParmsArray)

该函数返回在类型信息中指定函数的信息。并由 nTypeInfoIndex 指定类型信息。nFunctionIndex 指定函数的索引，该函数的信息被返回。nFunctionIndex 的取值在 0 和函数总数之间，TLGetTypeAttr( )方法程序的aTypeArray[9] 返回函数总数。TIGetFuncDesc( ) 创建两个数组，该数组的名称由 aTypeArray 和 aParmsArray 两参数指定。aTypeArray 包含有关函数的信息。aParmsArray 数组包含该函数的参数列表。下表 8.10 列出了 aTypeArray 中每个元素包含的信息：

表 8.10　aTypeArray 中每个元素包含的信息

1	ID
2	Func 类型 - 指定函数是虚拟的、静态的，或者是只用于发送的
3	Invoke 类型 - 指定是否是一个属性数组，如果是，是什么类型
4	Callconv - 指定函数的调用约定
5	参数的总数
6	可选参数的数目
7	对于 FUNC_VIRTUAL，指定虚拟函数表的偏移量
8	允许的 Scodes 数目
9	标识

下表 8.11 给出了 Func 类型：

表 8.11 Func 类型

FUNC_PUREVIRTUAL	该函数是通过虚拟函数表访问的，并且具有隐藏的“ this ”指针
FUNC_VIRTUAL	该函数具有 implementation，除此之外该函数的访问方式与 PUREVIRTUAL 相同
FUNC_NONVIRTUAL	该函数是通过静态地址访问的，并且具有隐藏的“ this ”指针
FUNC_STATIC	该函数是通过静态地址访问的，并且不具有隐藏的“ this ”指针
FUNC_DISPATCH	该函数只能通过 IDispatch 访问

下表给出了 Invoke 类型：

表 8.12 Invoke 类型

INVOKE_FUNC	该成员是使用普通的函数激活语法调用的
INVOKE_PROPERTYGET	该函数是使用普通的属性访问语法激活的
INVOKE_PROPERTYPUT	该函数是使用属性值指定语法激活的
INVOKE_PROPERTYPUTREF	该函数是使用属性引用指定语法激活的

下表列出了标识：

表 8.13 标识

FUNCFLAG_FRESTRICTED = 1	该函数不能使用宏语言访问。该标识用于系统级的函数或者用于类型浏览器不能显示的函数。
FUNCFLAG_FSOURCE = 0x2	该函数返回一个对象，该对象是一个事件源。
FUNCFLAG_FBINDABLE = 0x4	该函数支持数据绑定。
FUNCFLAG_FDISPLAYBIND=0x10	该函数对用户显示为可绑定的，也就是说，FUNC_FBINDABLE 也必须设置。
FUNCFLAG_FDEFAULTBIND=0x20	该函数最适于代表对象。在类型信息中只有一个函数具有该属性。
FUNCFLAG_FHIDDEN= 0x40	虽然该函数存在并且是可绑定的，但是也不能显示给用户。

(7) TIGetNames(nTypeInfoIndex,aNameArray, aTypeArray[1])

该函数返回类型信息中函数的个数，并创建一个数组。由 nTypeInfoIndex 指定类型信息， aNameArray 指定所创建数组的名称。 aTypeArray[1]指定 TIGetFuncDesc( )方法程序中 aTypeArray 数组返回的 ID 。

8.1.15 Calendar 控件

Calendar控件显示了一个每月的日历，您可以将它加入到您的窗体中。其所属的 ActiveX

控件的文件为 Mscal.ocx，请加入到您的安装程序中。

通过您的 Calendar 控件的属性，您可以设置系统的时钟和查询 1990~2100 年的日期，星期等等。Calendar 控件的形式如图 8.66 所示。

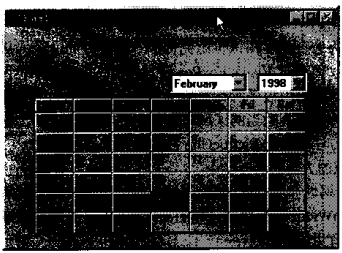


图 8.67　Calendar 控件

1. Calendar 控件的属性

Calendar 控件的属性的列表 8.14 如下：

表 8.14　Calendar 控件的属性列表

Day	当前所选择的日期
FirstDay	在当前 Calendar 控件中所选周的第一天
Month	Calendar 控件显示的当前月
Value	Calendar 控件中选定日期的相关日期数据
ValueIsNull	选择的当前日期是否被高亮度化
Year	当前年

2. Calendar 控件事件

Calendar 控件事件响应主要有：

表 8.15　Calendar 控件事件响应

AfterUpdate	在用户改变了日历的时间并且在 Calendar 控件重画窗体后发生。
BeforeUpdate	在用户改变了日历的时间并且在 Calendar 控件重画窗体前发生。
Click	用户用鼠标单击 Calendar 控件时发生
NewMonth	当 Calendar 控件的月被改变
NewYear	当 Calendar 控件的年被改变

读者可以轻松地通过改变该控件的属性来实现自己的程序，这里就不在叙述。

8.1.16 PictureClip 控件

PictureClip 控件可以使您选择源位图文件中的一个区域，然后将该部分的图像显示在窗体中或一个 Picture 控件中。PictureClip 控件在运行阶段是不可见的。在您的程序中加入该控件，您必须将文件 PICCLP32.OCX 安装在您的系统目录下。

PictureClip 控件为您存储多图像资源提供了一个有效的途径，从而代替了传统的多位图和多图标（Icon），但对于 Visual FoxPor 而言，仅仅只适用于位图。您可以通过它来为您的应用程序创建包含您应用程序所有 Icon 位图的图像。当您需要使用其中的一个时，您可以用 PictureClip 控件来选择适当的区域来获得您的图标位图。

您可以通过两种方法来指定 PictrueClip 控件选中的的位图：

(1) 指定选择区域的大小。

用控件的 ClipX 属性和 ClipY 属性详细说明剪切区域左上角的坐标，并使用控件的 ClipHeight 属性和 ClipWidth 属性指定区域的高度和宽度。当您想很随机的得到不同大小的图像时，该方法是很有用的。

(2) 用网格来分割源位图。

该方法给出了统一的图像单元，它们具有同样的大小，并且它们按照 0，1，2 的顺序排列。当您想要显示任意位图时，您可以用位图单元 GraphicCell 属性来获得该单元的位图。当您的源位图是很有规律的 Icon 调色板时，该方法对您很有用处。

当您需要向一个 PictureClip 控件中加入源位图时，您可以使用 PictureClip 控件的 Picture 属性来设置。此时，该源位图必须是*.bmp 文件。

PictureClip 控件的部分属性

(1) Rows 属性和 Cols 属性：

语法：

```
PictureClipName.Rows = number
```

说明：该属性将源位图分割为统一大小的图像单元。

(2) GraphicCell 属性：

语法：

```
[FormName.]PictureClip.GraphicCell (Index)
```

该语法中的 Index 为整数，从 0 开始，而且按照从左到右，从上到下的顺序增加。

说明：该属性用来获得给定位置的存储图像单元一维数组中的位图。该属性只能在运行阶段使用，而且该属性只能是只读的。

(3) Picture 属性：

语法：

```
object.Picture [= picture]
```

说明：返回和设置一个控件中显示的图像。对于 OLE 容器控件来说，仅仅在运行阶段是只读的。设计阶段不能使用。

表 8.16　Picture 属性可以是以下的取值

None	缺省。没有图像
Bitmap（位图）	指定图像。在设计阶段，您可以从属性窗口直接将图像加载进来。在运行阶段，您也可以使用该属性，但必须通过 LoadPicture 函数来装载位图。

(4) ClipHeight 属性和 ClipWidth 属性：

语法：

[FormName.]PictureClip.ClipHeight[= Height]

说明：指定位图部分的高度。 ClipWidth 属性用来指定位图部分的宽度。

(5) ClipX 属性和 ClipY 属性：

语法：

[FormName.]PictureClip.ClipX[= X]

属性 3 、 4 的类型是整数，描述的是象素点的个数。

说明：指定位图部分左上角的 x 坐标。 ClipY 属性指定位图部分左上角的 y 坐标。

(6) StretchX 属性和 StretchY 属性：

语法：

[FormName.]PictureClip.StretchX[= X]

[FormName.]PictureClip.StretchY[= Y]

该属性是用来定义剪切位图的尺寸。当该位图被拷贝时，位图将按照给出的尺寸来拉伸和压缩位图。该属性也是按照象素点来定义大小的。

说明：指定由 Clip 属性创建的目标位图尺寸。

8.1.17　MAPISession 控件和 MAPIMessages 控件

信息应用程序接口（ MAPI ）控件可以使您很方便地创建可以收发 E-mail 的应用程序。微软公司提供了二种 MAPI 控件：

（ 1 ） MAPISession 控件

（ 2 ） MAPIMessages 控件

1. MAPISession 控件

MAPISession 控件用来建立网络连接和创建会话。它也提供了 MAPI 会话服务的断连。MAPIMessage 控件提供给用户的是一系列的处理消息函数。

当使用 MAPISession 控件建立的网络连接，其 SessionID 属性包含一个指向 MAPI 会话的句柄。而该句柄必须传递给 MAPIMessage 控件，否则，使用 MAPIMesage 控件时会产生一个出错信息。

MAPISession 控件在运行阶段是不可视的。而且，该控件没有对于的事件响应程序。使用时，您必须指定合适的属性和方法。

MAPISession 操作常量：

mapSignOn	1	注册
mapSignOff	2	结束信件发送

2. MAPIMessage 控件

使用 MAPIMessage 控件，您可以做以下事情：

访问当前接收信箱（Inbox）中的信息。
创建一个新信件。
添加和删除收件人和信件所附带的文件。
发送信件。
保存、拷贝和删除信件。
显示地址薄对话框。
显示更详细的对话框。
访问附加件，包括 OLE 附加件。
解析收件人地址。
回信。

MAPIMessage 控件的许多属性可以被归纳为四类：

(1) 地址薄；
(2) 附加文件；
(3) 信息；
(4) 收件人属性。

而附加文件、信息和收件人属性是由控件的 AttachmentIndex 属性、MsgIndex 属性和 RecipIndex 属性分别控制的。

3. Message 缓冲区

当您使用 MAPIMessages 控件时，您需要保留二个缓冲区：写缓冲和读缓冲。读缓冲由从用户收件箱得到信件的索引组成。而 MsgIndex 属性用来访问在这个信件集合中的单个信件。信件的索引是从 0 开始按步长为 1 增加到信箱结尾。

信件集合是由 Fetch 方法来创建的。该集合包括了所有的 FetchMsgType 类型的信息，而且由 FetchSorted 属性来指定信件的排列顺序。在信件集中可以包括已经读过的信件或不包括它们，这样的工作是由 FetchUnreadOnly 顺序来决定的。在读缓冲区中的信件不能被用户改变，但用户可以将它拷贝到写缓冲区中来更改。

信件是由写缓冲区创建的。当 MsgIndex 属性设置为-1 时，写缓冲区是处于激活状态的。而且，很多的信件操作仅仅在写缓冲区处于激活状态时才可以使用，例如：发送信件、用对话框发送信件、保存信件、删除收件人信息和删除附加文件。

4. MAPIMessage 控件的属性和常量

(1) MAPIMessages 删除常量：

表 8.17　MAPIMessage 删除变量的定义

mapMessageDelete	0	删除当前信件
mapRecipientDelete	1	删除当前接收信件
mapAttachmentDelete	2	删除当前接收的附加文件

(2) MAPI 控件的出错常量：

表 8.18　MAPI 控件的出错常量

常　量	值	描　述
mapSuccessSuccess	32000	操作正常完成
mapUserAbort	32001	用户取消进程
mapFailure	32002	未指定错
mapLoginFail	32003	注册失败
mapDiskFull	32004	磁盘满
mapInsufficientMem	32005	内存不够
mapAccessDenied	32006	访问不允许
mapGeneralFailure	32007	一般错
mapTooManySessions	32008	过多的会话连接
mapTooManyFiles	32009	文件太多
mapTooManyRecipients	32010	接收件太多
mapAttachmentNotFound	32011	附加文件没找到
mapAttachmentOpenFailure	32012	附加文件打开出错
mapAttachmentWriteFailure	32013	附加文件写出错
mapUnknownRecipient	32014	未知的接收者
mapBadRecipType	32015	不正确的类型
mapNoMessages	32016	没有信件
mapInvalidMessage	32017	不正当的消息
mapTextTooLarge	32018	文本过大
mapInvalidSession	32019	不正当的会话连接
mapTypeNotSupported	32020	类型不匹配
mapAmbiguousRecipient	32021	不明确的接收者
mapMessageInUse	32022	信件正在使用
mapNetworkFailure	32023	网络失效
mapInvalidEditFields	32024	不正确的编辑区域
mapInvalidRecips	32025	不正确的接收者
mapNotSupported	32026	当前操作不支持

表 8.18 MAPI 控件的出错常量 （续表）

常 量	值	描 述
mapSessionExist	32050	Session ID 已经存在
mapInvalidBuffer	32051	读缓冲区只读
mapInvalidReadBufferAction	32052	仅在写缓冲区中有效
mapNoSession	32053	不正确的 session ID
mapInvalidRecipient	32054	初始信息无效
mapInvalidComposeBufferAction	32055	写缓冲区中的存在不正确
mapControlFailure	32056	没有信件
mapNoRecipients	32057	没有收件者
mapNoAttachment	32058	没有附加文件

(3) MAPIMessages Reciptype 常量

mapOrigList 0 信件的始发者
mapToList 1 信件中包含基本的接收者
mapCcList 2 信件中包含所有的接收者信息
mapBccList 3 信件中不包含所有接收者的信息

(4) MAPIMessages Attachtype 常量

mapData 0 附加文件是一个数据文
mapEOLE 1 附加文件是一个 OLE 大小文件
mapSOLE 2 附加文件是一个静态的 OLE 文件

以上讲述了 MAPI 的二个控件的属性和一些关键的常量。下面讨论一下 Visual FoxPor5.0 的一个发送邮件的例程。

5. 例程

在 Visual FoxPor5.0 中带有使用这二种控件的例子。就象在 Toolbar 控件一章中一样，其是用作类生成器。下来让我们学习一下该控件的使用方法。

(1) 使用类生成器来建立一个包容器。

打开类生成器，新创建一个可视类 Mailbtn，其派生于一个包容器类，即：Container。如图 8.68 所示。

(2) 在包容器中加入可视控件，如图 8.69 所示，其中包括：一个 Button 控件、一个 MAPIMessage 控件和一个 MAPISession 控件。并通过使用 Button 控件的 Picture 属性来向 Button 对象增加一个 Picture，同时消除 Button 对象的 Caption 属性。

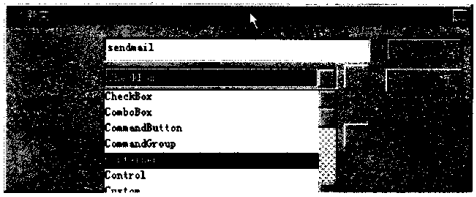


图 8.68　创建基于 Container 类的可视子类

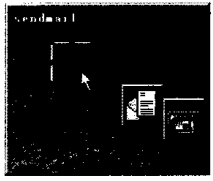


图 8.69　在可视子类中加入控件

(3) 关于本例程控件的补充说明。

该 Mailbtn 控件使用 MAPISession 控件来建立 MAPI 会话连接，用 MAPIMessage 控件来进行信息操作。

该控件包括了二个用户定义的方法： AddTabs 方法和 StripPath 方法。这二个方法是为了格式化从表中收集的信息，并且插入到邮件的信息中。同时，该类使用了用户另外定义的一个方法和属性： Signon 方法和 Logsession 属性（初始时，该属性被置为假）。当用户用鼠标单击 CmdMail 按钮时，则 CmdMail 按钮的 Click 事件将作出响应，并在该例程中调用 Signon 方法，同时设置 Logsession 为真。

如下：

```
this.logsession = .T.
this.OLEMSess.signon
```

当 Signon 失败时，该类的出错事件响应程序将被调用，而且同时属性 Logsession 将被设置为假。

(4) 例程代码。

1) 新建类代码：

```
**************************************************
*-- ParentClass:   container
*-- BaseClass:     container
*-- MAPI button to send the current record.
```

```
*
DEFINE CLASS mailbtn AS container

Width = 25
Height = 25
BorderWidth = 0
TabIndex = 1
BackColor = RGB(192,192,192)
Name = "mailbtn"
logsession = .F.

ADD OBJECT cmdmail AS commandbutton WITH ;
    Top = 0, ;
    Left = 0, ;
    Height = 25, ;
    Width = 25, ;
    Picture = "smmail.bmp", ;
    Caption = "", ;
    TabIndex = 1, ;
    Name = "cmdMail"

ADD OBJECT olemmess AS olecontrol WITH ;
    Top = -1000, ;
    Left = -1000, ;
    Height = 100, ;
    Width = 100, ;
    Name = "oleMmess"

ADD OBJECT olemsess AS olecontrol WITH ;
    Top = -1000, ;
    Left = -1000, ;
    Height = 100, ;
    Width = 100, ;
    Name = "olemSess"

PROCEDURE addtabs
    parameters tcString, tnMaxLength
    #DEFINE TABSPACES 8     && Number of characters that will equal 1 TAB
    local i, lnAdd, lnMaxTabs
```

```
        lnMaxTabs=int(tnMaxLength/TABSPACES)+1
        lnAdd = lnMaxTabs - INT(len(tcString)/TABSPACES)
        for i = 1 to lnAdd
            tcString = tcString + chr(9)
        endfor
        return tcString
    ENDPROC

    PROCEDURE strippath
        parameters tcString
        IF RAT( "\", tcString) > 0
            tcString = SUBSTR( tcString, RAT( "\", tcString) + 1 )
        ENDIF
        return tcString
    ENDPROC

    PROCEDURE signon
        #DEFINE ERR_NOMAPI_LOC   "It does not appear that you have MAPI installed.
Mail could not be run."

        this.logsession = .T.

        IF !FILE(GETENV("WINDIR")+"\SYSTEM32\MAPI32.DLL");
            AND !FILE(GETENV("WINDIR")+"\SYSTEM\MAPI32.DLL")
            MESSAGEBOX(ERR_NOMAPI_LOC)
            RETURN .F.
        ENDIF

        this.OLEMSess.signon
    ENDPROC

    PROCEDURE Error
        LPARAMETERS nError, cMethod, nLine
        =messageb(message(),48)
        this.logsession = .F.
    ENDPROC

    PROCEDURE Init
        this.logsession = .F.
```

```
ENDPROC

PROCEDURE cmdmail.Error
    LPARAMETERS nError, cMethod, nLine
    =messageb(message(),48)
    IF this.parent.logsession
        this.parent.OLEMSess.signoff
    ENDIF
    this.parent.logsession = .F.
ENDPROC

PROCEDURE cmdmail.Click
*:*****************************************************************
*:
*:       Class file: \samples\ole\mapibtn.vcx
*:
*:            System: OLE
*:            Author: Microsoft Corporation
*:           Created: 01/04/95
*:   Last modified: 04/13/95
*:
*:
*:*****************************************************************
* 该部分演示 MAPI 控件的使用
*
* 本例程创建新的 Mail 会话连接，从当前记录中收集数据
* 而且当数据组装成信件后，弹出 SendMail 对话框。
*
*****************************************************************

local j, lnMaxLength, i, lcMessageText, lvFieldValue
** j 和 i 为计数器

private array paDBFields

*** 局部化字符串
#DEFINE DBF_NOT_FOUND_LOC   "No table is open in the current work area."
#DEFINE GEN_UNSUPPORT_LOC   "General fields are not supported in this example
and will be skipped."
```

```
#DEFINE _FALSE_LOC              "FALSE"
#DEFINE _TRUE_LOC               "TRUE"
#DEFINE _NULL_LOC               "NULL"
#DEFINE _DOLLARSIGN_LOC         "$"
#DEFINE FLD_NO_PRINT_LOC "Field could not be printed."
#DEFINE RECORDNUM_LOC           "Record #"

* 校验是否当前工作区有表打开
if empty(dbf())
=messagebox(DBF_NOT_FOUND_LOC,48)
return
else
IF !this.parent.signon()              && 使用用户定义的方法
    RETURN
ENDIF
IF this.parent.LogSession     && 检测是否用户可以注册
    this.parent.OleMMess.sessionid=this.parent.OleMSess.sessionid

    * 得到当前表中字段个数
    =afields(paDBFields)

    **** 找到最长的字符串
    lnMaxLength = 0
    for j = 1 to alen(paDBFields,1)
        if len(paDBFields(j,1))+2 > lnMaxLength
            lnMaxLength = len(paDBFields(j,1))+2
        endif
    endfor

    * 开始新邮件并建立文本
    this.parent.OleMMess.compose
    lcMessageText=""
    for i = 1 to alen(paDBFields,1)
        lvFieldValue=alltrim(upper(paDBFields(i,1)))
        lcMessageText=lcMessageText+this.parent.addtabs((lvFieldValue+":
"),lnMaxLength)
        if !isnull(&lvFieldValue)
            do case
                case    paDBFields(i,2)=    "N"    or    paDBFields(i,2)=    "B"    or
```

```
paDBFields(i,2)= "F"
                        lcMessageText = lcMessageText +
alltrim(str(&lvFieldValue))+chr(13)
                    case paDBFields(i,2) = "Y"
                        lcMessageText =
lcMessageText+_DOLLARSIGN_LOC+alltrim(str(&lvFieldValue,10,2))+chr(13)
                    case paDBFields(i,2)= "C" or paDBFields(i,2) = "M"
                        lcMessageText=lcMessageText + alltrim(&lvFieldValue)+chr(13)
                    case paDBFields(i,2)= "G"

    lcMessageText=lcMessageText+GEN_UNSUPPORT_LOC+chr(13)
                    case paDBFields(i,2) = "D"
                        lcMessageText=lcMessageText +
alltrim(DTOC(&lvFieldValue))+chr(13)
                    case paDBFields(i,2) = "T"
                        lcMessageText = lcMessageText +
alltrim(TTOC(&lvFieldValue))+chr(13)
                    case paDBFields(i,2) = "L"
                        if &lvFieldValue
                            lcMessageText = lcMessageText+_TRUE_LOC+chr(13)
                        else
                            lcMessageText = lcMessageText+_FALSE_LOC+chr(13)
                        endif
                    otherwise
                        lcMessageText = lcMessageText+FLD_NO_PRINT_LOC+chr(13)

                endcase
            else
                lcMessageText=lcMessageText+_NULL_LOC
            endif
        endfor
        this.parent.OleMMess.msgnotetext=lcMessageText
        this.parent.OleMMess.msgsubject=this.parent.strippath(alltrim(dbf()))+":
"+RECORDNUM_LOC+alltrim(str(recno()))
        this.parent.OleMMess.send(1)
        IF this.parent.logsession
            this.parent.OleMSess.signoff
        ENDIF   && 会话句柄测试
    ENDIF       && 注册测试
```

```
endif                    && 表测试
ENDPROC

ENDDEFINE
*
*-- EndDefine: mailbtn
**************************************************
```

2）主程序代码：

```
**************************************************
*-- ParentClass:   form
*-- BaseClass:     form
*
DEFINE CLASS form1 AS form

DataSession = 2
Height = 212
Width = 463
DoCreate = .T.
AutoCenter = .T.
Caption = "发送信件"
MaxButton = .F.
MinButton = .F.
HelpContextID = 196
Name = "Form1"

ADD OBJECT cmdclose2 AS cmdclose WITH ;
    Top = 179, ;
    Left = 381, ;
    TabIndex = 8, ;
    Name = "Cmdclose2"

ADD OBJECT ctrmapibtn AS mailbtn WITH ;
    Top = 100, ;
    Left = 380, ;
    Width = 77, ;
    Height = 66, ;
    BackStyle = 0, ;
    TabIndex = 5, ;
    Name = "ctrMapiBtn", ;
```

```
        cmdMail.Top = 21, ;
        cmdMail.Left = 0, ;
        cmdMail.Height = 40, ;
        cmdMail.Width = 72, ;
        cmdMail.FontName = "MS Sans Serif", ;
        cmdMail.FontSize = 8, ;
        cmdMail.Picture = "..\classes\smmail.bmp", ;
        cmdMail.Caption = "发送记录", ;
        cmdMail.Name = "cmdMail", ;
        OleMMess.Top = -1000, ;
        OleMMess.Left = -1000, ;
        OleMMess.Height = 100, ;
        OleMMess.Width = 100, ;
        OleMMess.Name = "OleMMess", ;
        OleMSess.Top = -1000, ;
        OleMSess.Left = -1000, ;
        OleMSess.Height = 100, ;
        OleMSess.Width = 100, ;
        OleMSess.Name = "OleMSess"

    ADD OBJECT shape3 AS shape WITH ;
        Top = 13, ;
        Left = 10, ;
        Height = 57, ;
        Width = 443, ;
        BackStyle = 0, ;
        SpecialEffect = 0, ;
        Name = "Shape3"

    ADD OBJECT label7 AS label WITH ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        WordWrap = .T., ;
        Caption = "ActiveX 控件的 MAPI 控件应用", ;
        Height = 25, ;
        Left = 15, ;
        Top = 36, ;
        Width = 422, ;
        TabIndex = 0, ;
```

```
            Name = "Label7"

        ADD OBJECT label8 AS label WITH ;
            AutoSize = .T., ;
            FontName = "MS Sans Serif", ;
            FontSize = 8, ;
            Caption = "提示信息", ;
            Height = 15, ;
            Left = 18, ;
            Top = 7, ;
            Width = 50, ;
            TabIndex = 0, ;
            Name = "Label8"

        ADD OBJECT text1 AS textbox WITH ;
            Height = 25, ;
            Left = 128, ;
            Top = 120, ;
            Width = 181, ;
            Name = "Text1"

        ADD OBJECT label1 AS label WITH ;
            FontSize = 10, ;
            Caption = "Mail Message:", ;
            Height = 25, ;
            Left = 24, ;
            Top = 120, ;
            Width = 92, ;
            Name = "Label1"

        PROCEDURE Init
            * 测试 OCX 是否安装和加载
            IF TYPE("THIS.ctrMapiBtn.oleMMess") # "O" OR
ISNULL(THIS.ctrMapiBtn.oleMMess)
                RETURN .F.
            ENDIF

            * 测试 OCX 是否安装和加载
            IF TYPE("THIS.ctrMapiBtn.oleMSess") # "O" OR
```

```
ISNULL(THIS.ctrMapiBtn.oleMSess)
            RETURN .F.
        ENDIF

        use sys(2004)+"samples\data\testdata!products"
    ENDPROC

    PROCEDURE Error
        LPARAMETERS nError, cMethod, nLine
        =messageb(message(),48)
        cancel
    ENDPROC

    PROCEDURE Activate
        THIS.c_solutions1.saveHelp
    ENDPROC

    PROCEDURE Deactivate
        IF TYPE("THIS.c_solutions1") = "O" THEN
            THIS.c_solutions1.restoreHelp
        ENDIF
    ENDPROC

    ENDDEFINE
    *
    *-- EndDefine: form1
    **************************************************
```

8.1.18 MSComm 控件

MSComm 控件为您的应用程序提供了串口通信的功能，它允许您通过串口来发送和接收数据。

MSComm 控件提供了以下的二种处理通讯的手段：

(1) 事件驱动通讯是一个处理和串口交互强有力的方法。在很多情况下，您想要一个事件发生的时间，例如：从 CD （ Carried Detect ）或 RTS （ Request To Send ）线到达一个字符或发生了改变。在这种情况下，采用 MSComm 控件的 OnComm 事件来捕捉和处理这种通讯事件。 OnComm 事件也可以检测和处理通讯时发生的错误。对于那些可能事件和通讯的错误，请参考 CommEvent 属性。

(2) 您也可以通过检测 CommEvent 属性的值来发现您应用程序中的事件和出错。

每个 MSComm 控件可以控制您和一个串口进行通讯。如果您需要通过应用程序访问更多的串口，您必须使用更多的 MSComm 控件。端口地址和中断号可以由 Windows 中的控制面板来改变。

虽然 MSComm 控件具有很多的属性，但对于您急需掌握的是以下的几个：

表 8.19　MSComm 控件的主要属性

CommPort	设置和返回通讯端口号
Settings	通过字符串来设置和返回波特比、奇偶校验、数据位和停止位
PortOpen	设置和返回通讯端口的状态，同时打开和关闭端口
Input	返回和删除从接收缓冲区中得到的字符
Output	向发送缓冲区中写字符或字符串

1. MSComm 控件属性

(1) Input 属性。

说明：返回和设置从接收缓冲区中得到的数据流。在允许阶段该属性为只读。不可更改。

语法：

MSCommName.Input

Input 属性、InputLen 属性和 InputMode 属性是有关系的。InputLen 属性决定读取的字符个数。当 InputLen 属性为 0 时，用 Input 属性来读取数据时将读取整个缓冲区中的数据。InputMode 属性是设置读取数据的类型。当 InputMode 属性被设置为 comInputModeText 时，数据被读取并按照文本的形式存储在变量中；而当 InputMode 属性设置为 comInputModeBinary 时，则接收来的数据按照二进制形式存储在一个二进制数组中。

(2) CommEvent 属性。

说明：返回最近的通讯事件和错误。

语法：

MSCommName.CommEvent

虽然 OnComm 事件在通讯错误和通讯事件发生时都响应，但 CommEvent 属性为这些通讯事件和通讯错误保留出错数字码。当要决定导致 OnComm 事件发生的具体错误和通讯事件时，必须依靠 CommEvent 属性。当通讯错误产生和通讯事件发生，CommEvent 属性将返回一个数字码。出错码如下表所示。

表 8.20　CommEvent 属性返回的出错码

常　量	值	描　述
comEventBreak	1001	受到中断信号
comEventCTSTO	1002	CTS 超时

表 8.20 CommEvent 属性返回的出错码 （续表）

常 量	值	描 述
comEventDSRTO	1003	DSR 超时
ComEventFrame	1004	体制错
ComEventOverrun	1006	端口溢出。在一个字符没有接收之前，有另一个字符发送过来，并且第一个字符丢失
ComEventCDTO	1007	CD 超时
ComEventRxOver	1008	接收缓冲区溢出
ComEventRxParity	1009	奇偶校验出错
ComEventTxFull	1010	发送缓冲区溢出
ComEventDCB	1011	异常出错产生（DCB）

通讯事件包括以下的设置常量：

表 8.21 通讯事件包括的常量

事件	值	描 述
comEvSend	1	在发送缓冲区中的字符的数量少于门槛值
ComEvReceive	2	接收的字符个数。当您从接收缓冲区中删除数据时，该事件由 Input 属性而连续地产生
ComEvCTS	3	CTS 线改变
comEvDSR	4	当 DSR 从 0 变到 1 时，该事件发生
comEvCD	5	CD 线上改变
comEvRing	6	检测到 Ring
comEvEOF	7	接收文件尾

(3) CommPort 属性。

说明：设置和返回通讯端口号。

语法：

MSCommName.CommPort[= value]

您使用 PortOpen 属性打开端口时，当该端口不存在，则 MSComm 控件将产生 68 号错误码。而在设计阶段，端口号可以设置为 1~16 。如果您要打开一个端口，您必须设置属性。

(4) Settings 属性。

说明：设置和返回波特比、奇偶校验、数据位和停止位参数。

语法：

MSCommName.Settings [= value]

其中， Value 表示通讯端口的设置。当端口被打开，而且其值不正确时， MSComm 控件产生一个 380 号错误。 Value 是由如下的格式组成的：

"BBBB， P， D， S"

其分为四个部分，BBBB 代表波特比，P 是奇偶校验位。D 是数据位，S 表示停止位，而其缺省值为“9600，N，8，1”。

以下是正确的波特比数：110、300、600、1200、2400、9600（缺省值）、14400、19200、28800、38400（保留值）、56000（保留值）、128000（保留值）、256000（保留值）。

以下是正确的奇偶校验位：

表 8.22　奇偶校验位

E	偶校验
M	标志位校验
N	（缺省）无校验
O	奇校验
S	空格校验

以下是正确的数据位值：4、5、6、7、8（缺省）。

以下是正确的停止位值：1（缺省）、1.5、2。

(5) PortOpen 属性。

说明：设置和返回通讯端口的状态（打开、关闭）。

语法：

MSCommName.PortOpen [= value]

其中 Value 是一个布尔型数，来指定通讯端口的状态。当 Value 为真（True）时，端口是打开的，当 Value 为假（False）时，端口是关闭的。

通过设置 PortOpen 属性为 True 来打开端口；通过设置该属性为 False 来关闭端口并且清空接收缓冲区和发送缓冲区。当您的应用程序结束时，MSComm 控件自动关闭串口。

当您的 CommPort 属性设置为不正确的端口号，而且试图打开该端口时，MSComm 控件产生一般的 68 号错。所以您必须确信 CommPort 属性在打开端口时的设置是正确的。

附加说明一下，您的串口设备必须支持 Settings 属性的当前值。如果您的硬件设备不支持控件的 Settings 属性的设置，您的硬件可能工作得不稳定。

如果在端口被打开之前，DTREnable 属性或 RTSEnable 属性被设置为真；同样当端口被关闭时，这些属性将被设置为假。否则，DTR 和 RTS 线将保持它们以前的状态。

(6) Output 属性。

说明：对发送缓冲区写数据流。

语法：

MSCommName.Output [= value]

其中：Value 值为输出到流中的字符。

用 Output 属性可以传送文本数据和二进制数据。发送文本数据时，您必须指定接收变量包含一个字符串。发送二进制代码时，您必须在接收变量中包含一个二进制数组。正常情况，如果您发送 ANSI 字符串到您的应用程序，您可以发送文本数据；如果您需要传送

的数据中包含可嵌入控制的特性，即没有字符，那么，您最好还是使用二进制数据来传送数据。

(7) InBufferSize 属性。

说明：设置和返回接收缓冲区的尺寸。（按比特计）

语法：

MSCommName.InBufferSize[= value]

InBufferSize 指出总体的接收缓冲区的尺寸。缺省值为 1024 比特。

当您的接收缓冲区越大，您应用程序的可使用内存就越少。但是，您的缓冲区太小，有可能会发生内存溢出的现象。当出现该情况时，您需要增加缓冲区大小或处理您的应用程序的传送速率。

(8) OutBufferSize 属性。

说明：设置和返回发送缓冲区的大小。（按比特计）

语法：

MSCommName.OutBufferSize [= value]

OutBufferSize 指定了总共的发送缓冲区的尺寸。缺省值为 512 比特。

2. MSComm 控件事件

OnComm 事件

说明： OnComm 事件指示通讯事件的发生和通讯错误的产生，并不管 CommEvent 属性的值是否改变。

语法：

MSCommName.OnComm ()

CommEvent 属性包括了由 OnComm 事件产生的实际错误码和事件。可以通过设置 RThreshold 属性和 SThreshold 属性为 0 来放弃捕捉 comEvReceive 事件和 comEvSEnd 事件。

OnComm 事件例程：

以下的例程演示了控件如何控制通讯的错误和事件。您可以插入代码到相关联的 Case 语句中来处理特定的错误和事件。

```
PROCEDURE MSCommName.OnComm ()
Select Case MSComm1.CommEvent
&&为了处理不同的事件和错误
&&请加入代码到相应的地方

&&错误
    Case comEventBreak    &&Break 产生
    Case comEventCDTO     && CD (RLSD) 超时
    Case comEventCTSTO    && CTS 超时
    Case comEventDSRTO    &&DSR 超时
    Case comEventFrame    &&Frame 错
    Case comEventOverrun  &&数据丢失
```

```
        Case comEventRxOver  &&接收缓冲溢出
        Case comEventRxParity &&奇偶校验错
        Case comEventTxFull   &&发送缓冲区满
        Case comEventDCB      &&异常产生
    &&事件
        Case comEvCD     &&CD 线改变
        Case comEvCTS    &&CTS 线改变
        Case comEvDSR    &&DSR 线改变
        Case comEvRing   &&Ring 指示器改变
        Case comEvReceive      &&接收字符数小于门槛值
        Case comEvSend   &&发送字符数小于门槛值
        Case comEvEof    &&到达文件末尾
    End Select
    ENDPROC
```

下面的例子显示了基本的 modem 串口通讯

Form.Init ()函数体：

```
&&使用 COM1 端口
MSComm1.CommPort = 1
&&9600 波特，没有校验，8 位数据和 1 位停止位
MSComm1.Settings = "9600,N,8,1"
&&读取整个缓冲区
MSComm1.InputLen = 0
&&打开端口
MSComm1.PortOpen = True
&&给 Modem 发送 AT 命令
MSComm1.Output = "AT" + Chr$(13)
&&等待串口数据的返回

Do
    &&转让系统控制函数...

Loop Until MSComm1.InBufferCount >= 2
&&从串口中读取“OK”响应信号
Instring = MSComm1.Input
&&关闭串口
MSComm1.PortOpen = False
```

8.2 OLE（自动化技术） Automation

OLE 自动化技术是指：可以作为服务程序并具有 OLE 功能的应用程序和 ActiveX 组件。其可以向另一个应用程序提供对象。其可以是一个*.DLL 文件或一个应用程序，并将作为 OLE 对象被嵌入和连接。也可以称之为 ActiveX 控件。例如： Visual FoxPor 中可以包含一个 Word 文档，这种情况下， Microsoft Word 文档就称之为 OLE 服务程序，从而也实现了 OLE 自动化技术。其实， OLE 自动化服务器就是通过给其他应用程序一些服务的对象。

8.2.1 链接或嵌入 OLE 对象

可插入的 OLE 对象来自于支持 OLE 的应用程序，例如： Microsoft Excel 和 Word 。这样的对象包括 Word 文档和 Excel 工作表。在表单上您可以使用 OLE 容器控件链接或嵌入这些对象，并且您可以在表的通用字段中保存这些对象，使用 OLE 绑定型控件在您的表单上显示它们。

嵌入和链接的不同在于数据的存储方面。嵌入将数据存储到表或表单中，而链接却不是这样。例如，当您把一个 Excel 电子表格嵌入到一个表单上时，表单将包含该电子表格的复本，其他应用程序不能更改此复本。然而对于链接，表单仅包含一个对电子表格的引用，而不是电子表格本身。这意味着如果其他人更改了电子表格，则这些更改将自动地在您的表单上反映出来。如图 8.70 所示。

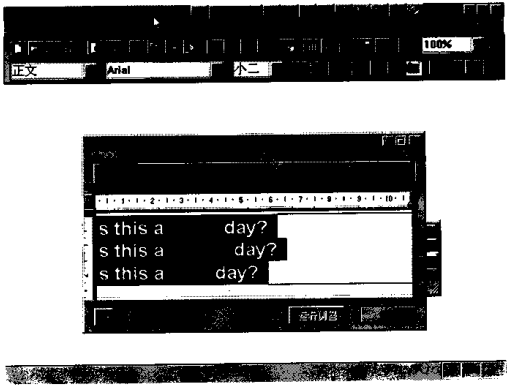


图 8.70 Word 文档在 Visual FoxPor 中

图中就是一个具有 Microsoft Word 文档的 Visual FoxPro 应用程序，注意图中的菜单，其不仅包括了 Visual FoxPro 中的菜单，而且也包括了 Microsoft Word 的菜单。在这里可以直接改变其中的 Word 文档，如图 8.71 所示。

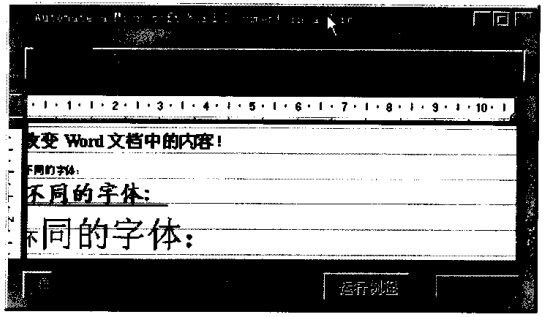


图 8.71　改变后的 Word 文档

8.2.2　创建服务程序

在 Visual FoxPro 中创建一个 OLE 服务程序，只需要在一个项目中包含定义为 OLEPUBLIC 的类。可以在项目中包含任意多的 OLEPUBLIC 类，可以在程序文件（*.PRG）或类库（*.VCX）中定义它们。

例如，下面的程序文件中的类定义创建一个自定义的 OLE 公共类：

```
DEFINE class person AS CUSTOM OLEPUBLIC
FirstName = SPACE(30)
LastName = SPACE(45)

PROCEDURE GetName
    RETURN THIS.FirstName + " " + This.LastName
ENDPROC
ENDDEFINE
```

当您在“类设计器”中设计类时，在“类信息”对话框中选择“OLE Public”，指明该类是 OLEPUBLIC 类。下图 8.72 是一个“类信息”对话框。

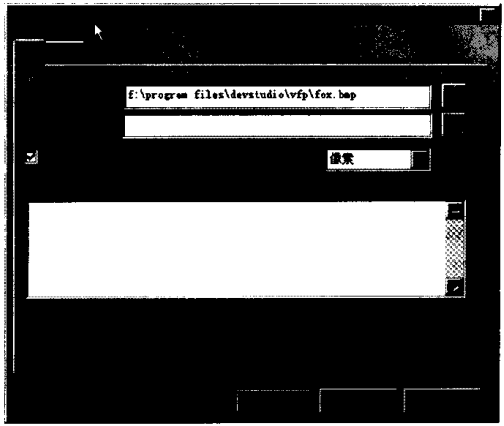


图 8.72 “类信息”对话框

8.2.3 编译服务程序

在 Visual FoxPro 中，可以创建外部过程或内部过程的 OLE 服务程序。内部过程组件是一个动态链接库（*.DLL ），它运行时与客户调用它的地址空间相同。外部过程组件是一个可执行文件（*.EXE 文件），它在自己的过程中运行。客户应用程序和外部过程服务程序之间的通信就称为过程间通信。

内部过程组件和外部过程组件各有好处。内部过程的服务程序更快一些，因为它没有过程间通信的负担。但外部过程可以进行远程调度，内部过程却不能。另外，因为内部过程服务程序和客户共享一个过程地址空间，*.DLL 中任何严重的错误都会中止客户的程序，而当外部过程中发生错误时，*.EXE 中只中止服务程序。

当使用 OLE 公共类创建可执行文件时，*.EXE 文件仍具有正常的功能。您仍可以运行这个可执行文件，提供一个用户界面，以及所有在应用程序中包含的正常功能。但是，其通过允许其他应用程序访问公开的某些功能，可以扩展您的应用程序。当不只一个用户访问 OLE 服务程序时，可能会发生冲突。如果您为应用程序功能提供了自动化访问以及一个用户界面，请在界面上提供用于一致性检查的附加层次，以确保您的环境没有被更改。若要编译 OLE 服务程序，请按照以下的步骤进行：

(1) 在“项目管理器”中，选择“连编”。如图 8.73 所示。

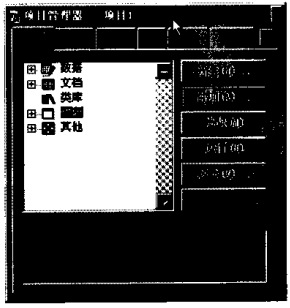


图 8.73　项目管理器

(2) 在“连编选项”对话框中，选择“连编可执行程序”或者“连编 OLE DLL ”。如图 8.74 所示的“连编选项”对话框：

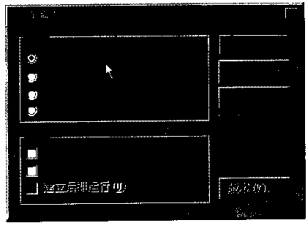


图 8.74　重新连编项目

(3) 选择“确定”或使用 BUILD DLL 或 BUILD EXE 命令。

当您连编项目完成，就可以在“项目信息”对话框中看到服务程序类。在该对话框中还可以为每个类指定帮助文件和一个帮助主题 ID 。该帮助文件可以在大多数普通对象浏览器中打开。如图 8.75 所示。

您可以在“项目信息”对话框中选择特定于类的实例选项。这些实例选项有：

(1) 不可创建：即使类被标识为 OLE 公共类，也不能被其他应用程序使用。例如，可以有一个标准 OLE 公共类库，这个 OLE 公共类被多个应用程序使用，但是对于这些应用程序中的其中某个应用程序来说，可以废止使用该类库中的一个或多个类的自动化功能。

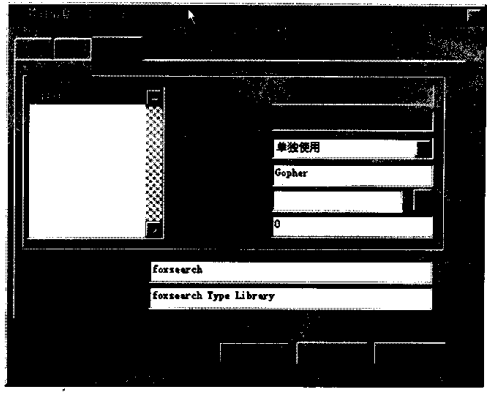


图 8.75 服务程序类

(2) 单独使用：每个客户应用程序使用服务程序时，都各自创建服务程序类的实例。每个实例都有一个线程。虽然独立的实例需要更多的内存，但选择“单独使用”可以允许操作系统执行抢先式多任务。

(3) 多重使用：在该选项下创建的服务程序可以使其他的应用程序使用相同服务器类的实例。

您如果在“项目信息”对话框中的“服务程序”选项页面中进行了修改，请重新连编*.DLL 或*.EXE，这样才能使新的设置生效。

8.2.4 使用 OLE 服务程序

如何在应用程序中使用您自己的OLE的服务程序呢？可以创建Automation对象的应用程序都能创建基于您的 OLE 服务程序对象，设置非隐藏或被保护的属性，调用方法程序。例如，假设您的服务程序名为 Foxole，并且包含了一个名为 person 的类，它具有 GetName 方法程序，下面的代码可以在 Visual FoxPro5.0 中运行：

```
oTest = CREATEOBJECT("foxole.person")
cName = oTest.GetName()
```

类似的代码可以在 Microsoft Excel 或 Visual Basic 应用程序中运行，代码如下：

```
Set oTest = CreateObject("foxole.person")
cName$ = oTest.GetName()
```

8.2.5　使用远程自动化

在标准的自动化方案中，客户和服务器在同一个计算机上运行，并且共享相同的资源，例如内存和处理器。当为自动化创建了本地服务程序，也可以在远程调度它们。通过一个网络，远程自动化提供与本地自动化相同的灵活性、可扩展性和功能。

远程自动化允许：服务程序使用独自的资源，即：许多不同的用户同时访问相同的服务程序。

您可以使用“远程自动化连接管理器”为远程自动化来配置其服务器和本地客户计算机，该管理器保存注册时的设置。在服务器计算机上运行的“自动化管理器”管理自动化，这样，与操作本地对象相同的代码也可以自动操作远程对象。

其过程是这样的：

(1) 通过客户进程的对象引用，使客户计算机上的远程自动化代理和远程计算机的远程自动化监听程序交互；

(2) 通过自动化管理器远程监听程序得到客户请求，通过其代理和服务器进程监听程序交互；

(3) 服务器进程监听程序和服务器的自动化程序交互；

(4) 按照逆向将信息返回客户进程。

8.2.6　服务器的配置

当然，启动远程自动化是要求对服务器进行配置的，使客户可以在“远程自动化连接管理器”中访问服务器。如下过程给出了服务器的配置步骤和方法：

(1) 将 OLE Server 的执行文件（*.EXE ）复制到服务器上，然后运行该程序以便在“ Windows 注册”中注册。

(2) 在服务器计算机上运行“远程自动化连接管理器”（ Racmgr32.exe ）。

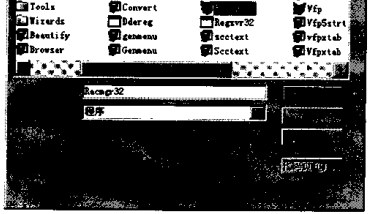


图 8.76　运行 Racmgr32.exe

该应用程序安装在 VFP 目录下。请使用 Explore 运行 Racmgr32.exe 程序或在运行对话框中输入：安装目录\racmgr32.exe，确定后会出现以下的对话框：如图 8.77 所示。

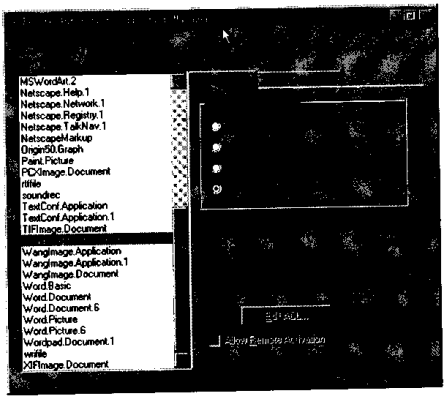


图 8.77 Racmgr32.exe 界面

其中在系统安全策略选项（System Security Policy）中选项的含义（见表 8.23）。

表 8.23 系统安全策略选项含义

名 称	值	描 述
Disallow All Remote Creates	0	不允许创建任何对象
Allow Remote Creates by Key	2	只有当选中“允许远程激活”复选框时才可以创建对象。下面的设置可以改变在 Windows 注册表中对象的 CLSID：AllowRemoteActivation = Y
Allow Remote Creates by ACL	3	只有当 Windows 注册表的 CLSID 的“访问控件列表”包含用户时，一个用户才可以创建对象。只适用于 Windows NT
Allow All Remote Creates	1	允许创建任何对象。但不要在开发环境之外使用

在 COM（组件对象模型）类中选择 Visual FoxPro Application 选项，并如图 8.77 所示的在 Client Access（客户访问权限）中选中 Allow All Remote Creater 选项。同时，在 Server 中选择网络协议和其他一些选项，如图 8.78 所示。

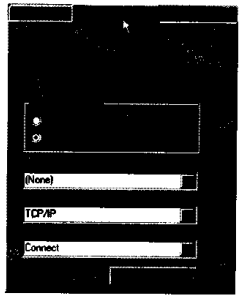


图 8.78　Server Connection 选项

请注意，您需要在远程自动连接管理器（Remote Automation Connection Manager）的注册菜单中（Register）选中 Remote 选项。

(3) 在“远程自动化连接管理器”中允许客户访问之后，请运行服务器计算机上“自动化管理器” Autmgr32.exe。Autmgr32.exe 安装在 Windows 95 下的 SYSTEM 文件夹中或者安装在 Windows NT 下的 SYSTEM32 文件夹中。这样将启动与其他计算机的远程自动化连接。

8.2.7　配置客户计算机

当服务器配置完之后，就可以配置本地的客户机了。以下是为远程自动化而配置本地的客户计算机的步骤：

(1) 复制*.vbr 文件，该文件在向客户机创建 OLE 服务程序时创建。

(2) Clireg32.vbr 命令。例如，如果文件为 MyServer.vbr，可在“命令”提示处运行如下命令：

```
Clireg32 myServer.vbr
```

(3) 在打开的对话框中，输入服务器的网络地址并选择网络协议（通常使用 TCP/IP 协议）。

8.2.8　OLE 自动化服务器的建立（例程和说明）

例程：自动在窗体中增加一个 Word 文档。如图 8.79 所示。

例程位置：File: SAMPLES\SOLUTION\OLE\OLEWORD.SCX

该例程演示了如何在窗体中增加一个嵌入的 Word 文档，并插入一些 RichText 文本（由文件产生）。典型的 Word 是由“Word Basic”对象和 CREATOBJECT()函数交互而设计的。比如象以下的代码：

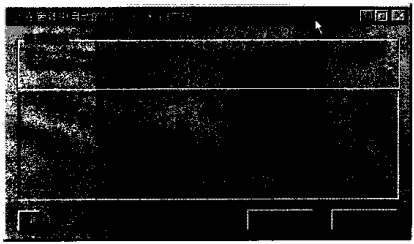


图 8.79 OLE 自动化技术--自动在窗体中增加一个 Word 文档

```
oForm = THISFORM
oForm.AddObject('oWordDoc','OleControl','WordDocument')
oForm.oWordDoc.Visible =  .t.
oForm.oWordDoc.DoVerb(0)
```

(1) 创建窗体。

如图 8.79 所示地创建窗体，其中位于中间的控件是一个 TextBox 控件。

(2) 创建代码。

该例程的代码如下所示。

```
**************************************************
*-- ParentClass:   form
*-- BaseClass:     form
*
DEFINE CLASS form1 AS form
DataSession = 2
Height = 228
Width = 433
DoCreate = .T.
AutoCenter = .T.
Caption = "在窗体中自动的加入一个 Word 文档"
MaxButton = .F.
HelpContextID = 117
Name = "Form1"

ADD OBJECT behindscenes1 AS behindscenes WITH ;
    Top = 199, ;
    Left = 12, ;
```

```
        TabIndex = 1, ;
        Name = "Behindscenes1"

    ADD OBJECT command3 AS commandbutton WITH ;
        Top = 199, ;
        Left = 260, ;
        Height = 23, ;
        Width = 72, ;
        FontName = "MS Dialog Light", ;
        FontSize = 8, ;
        Caption = "运行例程", ;
        TabIndex = 2, ;
        Name = "Command3"

    ADD OBJECT cmdclose1 AS cmdclose WITH ;
        Top = 199, ;
        Left = 351, ;
        Height = 23, ;
        Width = 72, ;
        Caption = "关闭", ;
        TabIndex = 3, ;
        Name = "Cmdclose1"

    ADD OBJECT shape2 AS shape WITH ;
        Top = 14, ;
        Left = 11, ;
        Height = 55, ;
        Width = 413, ;
        BackStyle = 0, ;
        SpecialEffect = 0, ;
        Name = "Shape2"

    ADD OBJECT label5 AS label WITH ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        WordWrap = .T., ;
        Caption = "嵌入一个 Microsoft Word 文档到一个 Visual FoxPor OLE 包容控件中，
并通过程序来控制 Word 文档的改变。", ;
        Height = 41, ;
```

```
        Left = 20, ;
        Top = 23, ;
        Width = 398, ;
        TabIndex = 0, ;
        Name = "Label5"

    ADD OBJECT label6 AS label WITH ;
        AutoSize = .T., ;
        FontName = "MS Sans Serif", ;
        FontSize = 8, ;
        Caption = "提示信息", ;
        Height = 15, ;
        Left = 19, ;
        Top = 8, ;
        Width = 50, ;
        TabIndex = 0, ;
        Name = "Label6"

    ADD OBJECT txtframe AS textbox WITH ;
        BackStyle = 1, ;
        Enabled = .F., ;
        Height = 116, ;
        Left = 12, ;
        ReadOnly = .T., ;
        TabIndex = 0, ;
        Top = 72, ;
        Width = 411, ;
        Name = "txtFrame"
    ENDPROC

    PROCEDURE command3.Click
        #DEFINE CRLF    CHR(13)+CHR(10)
        #DEFINE C_MESS1_LOC         "Word 文档测试?"
        #DEFINE C_MESS2_LOC         "Microsoft Word 文档测试?"
        #DEFINE C_MESS3_LOC   "Word 文档测试?"
        lWord97 = .T.
        nMouseRow = MROW()
        nMouseCol = MCOL()
        oForm = THISFORM
```

```
oForm.addobject('oWordDoc','olecontrol','word.document')
oForm.oWordDoc.Height =   THISFORM.txtFrame.height
oForm.oWordDoc.Width =   THISFORM.txtFrame.width
oForm.oWordDoc.Top =   THISFORM.txtFrame.top
oForm.oWordDoc.Left =   THISFORM.txtFrame.left
oForm.oWordDoc.Visible =   .t.
oForm.Show
oForm.oWordDoc.DoVerb(0)
IF TYPE("oForm.oWordDoc.object.name") # "C"
    lWord97 = .F.
    oWordRef = GetObject('','word.basic')
ENDIF
MOUSE CLICK AT 1,1
MOUSE AT m.nMouseRow,m.nMouseCol
IF m.lWord97
    oForm.oWordDoc.object.content = C_MESS1_LOC+CRLF
    oForm.oWordDoc.object.content.insertafter(C_MESS2_LOC+CRLF)
    oForm.oWordDoc.object.content.insertafter(C_MESS3_LOC)
    oForm.oWordDoc.object.content.Font.Bold = .T.
    oForm.oWordDoc.object.content.Font.Size = 18
    oForm.oWordDoc.object.content.Font.Name = "Arial"
    oForm.oWordDoc.object.content.Font.ColorIndex = 5
ELSE
    oWordRef.Insert(C_MESS1_LOC+CRLF)
    oWordRef.editselectall
    oWordRef.Font("Arial",18)
    oWordRef.Bold
    oWordRef.EditGoTo("\EndofDoc")
    oWordRef.WordLeft(4)
    oWordRef.SelectCurWord
    oWordRef.CharColor(2)
    oWordRef.EditGoTo("\EndofDoc")
    oWordRef.Insert(C_MESS2_LOC)
    oWordRef.WordLeft(3)
    oWordRef.SelectCurWord
    oWordRef.CharColor(4)
    oWordRef.Bold(0)
    oWordRef.EditGoTo("\StartofDoc")
    oWordRef.Insert(C_MESS3_LOC+CRLF)
```

```
            oWordRef.WordLeft(4)
            oWordRef.SelectCurWord
            oWordRef.CharColor(6)
            oWordRef.Bold(0)
            oWordRef.Italic
            oWordRef.editselectall
            oWordRef.shadingpattern(2)
            oWordRef.borderoutside()
            oWordRef.borderlinestyle(7)
        ENDIF
        THIS.ENABLED = .F.
    ENDPROC
    ENDDEFINE
    *
    *-- EndDefine: form1
    **************************************************
```

8.3 小　结

本章第一节主要解释了 AcitveX 控件在 Visual Foxpro5.0 中的部分主要的属性、方法和部分控件的使用，并使用了较多的例程来使读者更加清晰地掌握各个控件的使用方法。如仍有问题，请参考 Microsoft 公司的 ActiveX 控件的使用说明或帮助。

在第二节中讲述了关于 OLE 自动化技术的概念和使用了 Word 文档和 Visual FoxPro 进行交互的例程。同时讲述了如何进行远程自动化服务器控制的概念，并说明了远程自动化服务器的配置方法和本地客户机的配置方法。

第九章　网络环境下的数据库开发

过去几年已看到将个人计算机转化组织成局域网从而同大型计算机集成在一起的可编程工作站。现今局域网和广域网（Internet）的广泛应用，促使数据库迫切地需要向网络化发展。Oracle、Sybsae、Informix、DB2 都是现今流行的大型数据库，并支持 SQL 语言（Standard Query Language）。Microsoft 公司也进一步地改善了 Visual FoxPro，使其也可以支持 SQL 语言，并完全支持客户/服务器模式。

网络环境下的数据库开发是由于网络的发展而产生的，从而其带来了很多网络上的问题。例如：CCR（提交、并发和恢复--Commit，Concurrency，Recovery）。而这些问题必须在网络数据库开发的应用程序中得到完好的解决方案。Visual FoxPro5.0 中为程序开发人员提供了记录和表格锁定函数、命令以及新的数据缓冲方案，该方案能够更方便地锁定和更新数据。而利用文件和表格的锁定功能，当有多个用户要求访问表格和记录时，锁定就会禁止其他用户进入该数据区域，直到当前占用数据的用户完成其编辑活动，并安全地退出该区域为止。而使用缓冲区可以保存原始数据、记录，这样就可以恢复这些记录。

本章主要围绕网络环境下的程序开发来说明如何使多个用户访问共享数据。

9.1　多用户管理

在网络数据库开发中经常遇见的问题是数据库的多个用户访问。而对多个用户同时来访问数据库时的处理方法和在本地计算机上的处理方法是不一样的。它必须考虑到在网络文件共享的环境中，表格在网络驱动器中和网络用户进行交互时的某些概率很大的事件。例如：当有一个用户正在修改数据表中的数据时，另外一个用户需要读取该数据表或也要修改该数据表中的内容，那么 Visual FoxPro 是如何控制这样的事件的发生和处理呢？

9.1.1　多用户环境所必须考虑的问题——并发控制

在多用户的环境中，必须慎重而且考虑的最多的问题就是并发控制。同时还有恢复等和原子事务相关联的事件。

1. 记录锁定和文件锁定

开发多用户应用程序时将用到两种类型的锁定：记录锁定和文件锁定。本节将讨论它们之间的差异，并给出如何将它们应用在程序中的方法。

(1) 记录和文件锁定。

在多用户的访问中，文件和记录都是可以被锁定的。当使用记录锁定（Record Lock）时，它就只允许执行记录锁定的用户来访问它，而对于其他的用户，只有等待。使用了文件锁定（File Lock）意味着该用户将独占整个表格，同时禁止其他用户向表格中写数据，而只允许锁定文件的用户独自编辑表格中的记录。

执行对文件和记录的锁定后，并不能完全地限制用户对文件和数据的访问，而只是部分地限制了用户对文件和记录的写权限，并没有限制其对文件和数据的读取权限的。当执行完文件和记录的锁定后，文件和记录的写权限将被限制，直到占有数据的用户将该文件从锁定的状态释放。

文件和记录锁定的最大区别在于其锁定数据的数目。文件锁定类型将锁定整个表格数据，表格中的所有数据将同时被锁定；而记录锁定类型仅仅锁定了用户指定的记录。在网络环境中，建议程序员最好使用记录锁定，这样可以给其他用户一个比较大的访问权限空间。

1 ）记录锁定的特点：从锁定数据的量来说，锁定记录比锁定表的破坏性小。当一个用户仅锁定一个记录时，其他用户仍然可以操作其他记录，例如：添加或删除表中的其他记录，而当用户锁定整个表时，其他用户就只能等待第一个锁定表的用户释放其权力。如果记录或表已经被一个用户锁定，或者表已经被其他用户用独占方式打开，则执行该锁定记录或表的操作将导致失败。许多的数据库命令，如：BROWSE 、CHANGE 、EDIT 和 MODIFY MEMO 等命令只有在编辑记录时才锁定记录。如果您正在编辑来自相关表记录中的字段，那么该相关记录可能会被锁定。如果当前记录或相关记录已被其他用户锁定，则锁定失败。如果锁定成功，则可以编辑该记录；在移到另一个记录或激活其他窗口时，记录将解锁并释放其权限。

2 ）表锁定的特点：Visual FoxPro 的部分命令执行后将锁定整个表，而有些命令则仅仅锁定表头。表锁定命令比表头锁定命令使用和涉及的范围更广。当锁定表头时，其他用户将不能添加或删除记录，但可以修改字段内的数据。

当使用 APPEND BLANK 命令时，用户可以共享表而不发生冲突。当有两个或多个用户在同时执行 APPEND BLANK 命令，当一个用户先使用了该命令，而其他用户也同样向该表追加一个空白记录时，这样就会导致系统出错。用户如果无法锁定一个表时，则锁定整个表的命令将返回错误信息：“其他用户正在使用文件”，用户可以按 ESC 键来取消正在进行的锁定操作。

(2) 自动锁定和手工锁定。

在对文件和记录锁定时，我们可以采用不同的函数和方法。而有些命令和函数可以实现自动的锁定，有些则不能，还要使用其他的锁定方法。

1 ）自动锁定：下表 9.1 列出了可以自动锁定的命令：

表 9.1 自动锁定命令表一览

命令	锁定范围
ALTER TABLE	整个表
APPEND	整个表
APPEND BLANK	表头
APPEND FROM	整个表
APPEND FROM ARRAY	表头

（续表）

命令	锁定范围
APPEND MEMO	当前记录
BLANK	当前记录
BROWSE、CHANGE 和 EDIT	一旦开始编辑字段，锁定对象为当前记录和相关表中别名字段的所有记录
CURSORSETPROP( )	取决于参数
DELETE	当前记录
DELETE NEXT 1	当前记录
DELETE RECORD n	记录 n
DELETE 删除多个记录	整个表
DELETE - SQL	当前记录
GATHER	当前记录
INSERT	整个表
INSERT - SQL	表头
MODIFY MEMO	编辑开始时，锁定当前记录
READ	当前记录和别名字段的所有记录
RECALL	当前记录
RECALL NEXT 1	当前记录
RECALL RECORD n	记录 n
RECALL 恢复多个记录	整个表
REPLACE	当前记录和别名字段的所有记录
REPLACE NEXT 1	当前记录和别名字段的所有记录
REPLACE RECORD n	记录 n 和别名字段的所有记录
REPLACE 替换多个记录	整个表和别名字段的所有记录
SHOW GETS	当前记录和别名字段引用的所有记录
TABLEUPDATE( )	取决于缓冲
UPDATE	整个表
UPDATE - SQL	整个表

下面给出了一个自动锁定表的示例：

在下面的示例中，即使 Test 表是以共享方式打开的，用户为了从其他表追加记录也将会自动锁定整个表。

```
SET EXCLUSIVE OFF
USE Test
APPEND FROM AnotherTable FOR status = "OPEN"
```

其中 Test 和 AnotherTable 为两个不同的表。

2）手工锁定：用户可以通过使用锁定命令和函数来手工锁定一个记录或一个表，例如 RLOCK()、LOCK()、FLOCK()。

RLOCK( )函数和 LOCK( )函数的作用几乎是一样的，都可以用来锁定一个或多个记录；FLOCK( )可以锁定一个文件。同时，LOCK( )和 RLOCK( )函数也可以用来锁定一个表头，从而给用户一个更大的选择余地。在使用 LOCK( )或 RLOCK( )函数时，如果把 0 作为记录编号，而且测试时表头未被锁定，则该函数将锁定表头并返回一个“真”(.T.)。

当用户锁定了一个记录或一个表并进行操作后，一定要尽快解锁，这样可以方便其他用户访问记录或表。解锁时可使用 UNLOCK 命令。

以上的手工锁定函数都具有以下的功能：

测试记录或表的锁定状态：当测试表明该记录未被加锁时，锁定该记录或表并返回“真”(.T.)。当该记录或表不能锁定，则根据 SET REPROCESS 的当前设置，再次进行锁定。判断返回“真”(.T.)或“假”(.F.)，来反映锁定尝试是否成功。当用户不想锁定记录，而仅仅测试该记录的锁定状态，可以使用 ISRLOCKED( )或 ISFLOCKED( )函数。如果用户锁定记录或表的操作失败，则 SET REPROCESS 命令和您的当前错误处理例程将确定是否再次尝试锁定。用户可以利用 SET REPROCESS 来控制锁定尝试的次数以及时间。

下面是手工锁定记录的示例：

该示例以共享方式打开 Test 表，并用 FLOCK( )函数锁定该表。如果该表被成功地锁定，则 REPLACE ALL 将更新该表中的所有记录，UNLOCK 对该文件解锁；如果其他用户已经锁定该文件或文件中的记录，则显示一条信息，表明当时锁定表失败。

```
SET EXCLUSIVE OFF
SET REPROCESS TO 0
USE Test     && 打开共享表 Test
IF FLOCK()
   REPLACE ALL contact ;     && 替换并解锁
      WITH UPPER(contact)
   UNLOCK
ELSE      && 输出信息
WAIT "文件正被使用!" WINDOW NOWAIT
ENDIF
```

(3) 记录和表的解锁。

在许多的情况下，如果用户显式地锁定了表格或记录，那么其他用户必须等待该记录或文件的锁定被解锁后才能使用。所以，在共享环境下当您锁定了记录或文件并完成了相应的数据操作之后，应该及时解锁，以便其他用户可以访问该表或记录。要解锁被自动锁定的记录，用户只需移动一下记录指针，甚至在设置 MULTILOCKS ON 的情况下也是如此。但对于用户的手工锁定记录，则必须使用明确的命令对记录解锁，仅仅移动记录指针是不够的。

在表 9.2 对释放记录和文件锁定的命令进行了说明。

表 9.2　解锁命令一览表

命　令	效　果
UNLOCK	解锁当前工作区内的记录和文件
UNLOCK ALL	对所有工作区解锁
SET MULTILOCKS OFF	建立新锁定的同时自动解锁当前锁定
FLOCK( )	在锁定文件之前解锁文件中的所有记录
CLEAR ALL 、 CLOSE ALL 、 USE 、 QUIT	对所有记录和文件解锁
END TRANSACTION	对所有的自动锁定项解锁
TABLEUPDATE( )	在更新表之前解锁所有锁定项

2. 缓冲编辑

Visual FoxPro5.0 引入了表单和程序化的数据缓冲。这一改进使在网络中对数据的保护比显式锁定更加容易。缓冲数据的使用能够更好地使用本地资源，从而限制了对网络及服务器资源的竞争。如果希望在更新时保护数据，可以使用缓冲技术。在多用户环境下，Visual FoxPro 的记录缓冲和表缓冲技术可以保护对单个记录或多个记录所做的数据更新以及数据维护操作。缓冲区可以自动测试、锁定以及解锁记录或表。借助缓冲技术，可以容易地检测并解决数据更新操作过程中所遇到的冲突。例如，当前记录被复制到一个由 Visual FoxPro 进行管理的内存或磁盘区域，而其他用户仍然可以同时访问原来的记录。当离开该记录或以编程方式更新该记录时，Visual FoxPro 将准备锁定该记录，确认其他用户没有做修改，然后写入编辑结果。在试图更新数据时，还必须解决冲突，若有冲突则应防止编辑结果写入原来的表。Visual FoxPro 提供了两种缓冲方法：记录缓冲和表缓冲。

（1） 记录缓冲。

当用户希望访问、修改或向磁盘中写入一条记录时，应该使用记录的缓冲。该技术允许适当的对过程进行有效性检查，并且对于多用户环境中其他用户的数据更新操作影响最小。

（2）表格缓冲。

表格缓冲用在需要将更新缓冲到一个表格中的多条记录的情况下。该方法很适合于位于一个表格中的记录，或者存在于一对多关系中的子记录。

记录缓冲和表格缓冲可以用于两种不同的锁定模式：悲观锁定（pessimisti clocking）和乐观锁定（optimistic locking）。这两种方式决定了一个或多个记录何时被锁定，又是何时、怎样被解锁的。

1）悲观锁定。悲观锁定方式非常类似于 FoxPro 以前版本的标准锁定机制。当用户对某一特定记录或表进行修改时，悲观锁定能防止其他用户访问它。悲观锁定为单个记录的修改提供最安全的工作环境，但是会降低用户的操作速度。但是该锁定方法有其内在数据缓冲等更多的好处。

2）乐观锁定。乐观锁定是更新记录的有效方法，也就是，锁定只在对记录进行写操作时才生效。这样就使用户独占系统的时间达到最小。当在一个远程表上使用记录或表缓冲时，Visual FoxPro 将强制使用乐观锁定。

这两种锁定方式的选择是由函数 CURSORSETPROP( )设置的 Buffering 属性值决定的。下表 9.3 列出了 Buffering 属性的有效值：

表 9.3 表格缓冲值

缓冲及锁定类型	对应的缓冲
无缓冲（默认值）	1
悲观记录锁定。锁定当前记录，当记录指针移动或发出 TABLEUPDATE( )命令后更新	2
乐观记录锁定。直到记录指针移动后，才锁定并更新	3
悲观表锁定。锁定当前记录，执行 TABLEUPDATE( )命令之后更新。	4
乐观表锁定。执行 TABLEUPDATE( )命令后，才锁定并更新编辑的记录	5

示例：

（1）悲观记录锁定。

用户可以通过如下函数和参数来实现悲观记录锁定：

```
CURSORSETPROP("Buffering", 2, "Test")
```

Visual FoxPro 锁定指针指向的当前记录。如果锁定成功，Visual FoxPro 将该当前记录放入缓冲区 Buffering 中并允许编辑。当用户移动记录指针或执行 TABLEUPDATE( )命令时，Visual FoxPro 将主动地把缓冲记录写入原来的表。

（2）乐观记录锁定。

用户可以通过如下函数和参数来实现乐观记录锁定：

```
CURSORSETPROP("Buffering", 3, "Test")
```

Visual FoxPro 将指针指向的当前记录写入缓冲区并允许编辑。当移动记录指针或执行 TABLEUPDATE( )命令时，Visual FoxPro 对该记录进行锁定。如果锁定成功，Visual FoxPro 将对磁盘上的记录的当前值与原来的缓冲区值进行比较。如果两值相同，则编辑结果写入原来的表；如果两值不同，则 Visual FoxPro 发出错误信息。

（3）悲观表锁定。

用户可以通过如下函数和参数来实现悲观表锁定：

```
CURSORSETPROP("Buffering", 4, "Test")
```

Visual FoxPro 锁定指针指向的当前记录。如果锁定成功，Visual FoxPro 将把该记录放入缓冲区并允许编辑。执行 TABLEUPDATE( )命令将向原来的表中写入缓冲记录数据。

（4）乐观表锁定。

用户可以通过如下函数和参数来实现乐观表锁定：

```
CURSORSETPROP("Buffering", 5,"Test")
```

在执行 TABLEUPDATE( )命令之前，Visual FoxPro 将记录写入缓冲区并允许编辑，而后，Visual FoxPro 将对缓冲区内的每一记录执行锁定操作，即，试图对每一个已编辑记录锁定。而该操作会有一系列的结果：

如果锁定成功，则对磁盘上的每一记录的当前值与原来的缓冲区值进行比较；

如果比较相同，则 Visual FoxPro 将编辑结果写入原来的表中。

如果比较不同，Visual FoxPro 将给出错误信息。

注意：启用表缓冲情况下，只有执行 TABLEUPDATE( )命令之后，Visual FoxPro 才更新表数据。

3. *更新数据*

当用户对数据库中的数据进行了修改之后，必须对数据更新，这种情况下，用户必须知道使用了哪种缓冲方法和锁定机制。如果知道了这些，我们就可以使用缓冲、事务或视图更新数据。

(1) 使用缓冲进行更新

在选择缓冲方法和锁定类型之后，就可启用记录或表缓冲。启用缓冲有两种方法：

1）在“表单设计器”中，设置表单数据环境中临时表的 BufferModeOverride 属性。如图 9.1 所示：

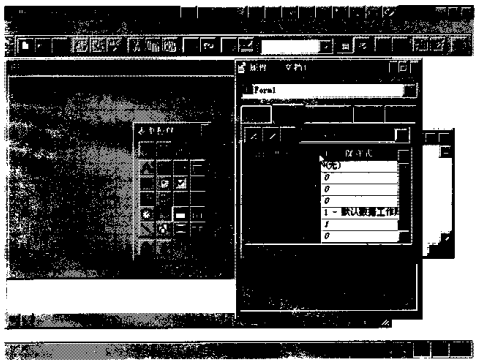


图 9.1　设置表单数据环境中临时表的 BufferModeOverride 属性

2）在代码中设置 Buffering 属性。例如，将下列代码放进表单的 Init 过程中，可以启用保守式记录缓冲：

```
CURSORSETPROP('Buffering', 2)
```

然后将进行更新操作的代码放在控件的适当方法程序代码中。用户可以使用 TABLEUPDATE( )把编辑结果写入原来的表。由于规则限制，造成对表的更新操作失败之后，要取消编辑结果，可以使用 TABLEREVERT( )命令。TABLEREVERT( )即使在没有明确启用表缓冲的情况也都有效。

下面的示例说明在启用保守式记录缓冲的情况下，如何更新记录。

代码如下：

```
OPEN DATABASE testdata
&&在表单的 Init 代码中，打开表并启用保守式记录缓冲
USE customers= CURSORSETPROP('Buffering', 2)

&&遍历字段，检查是否有字段进行了修改。
&&此代码可以在“保存”或“更新”命令按钮的 Click 事件中
lModified = .F.
FOR nFieldNum = 1 TO FCOUNT()
    IF GETFLDSTATE(nFieldNum) = 2
        lModified = .T.
        EXIT
    ENDIF
ENDFOR

&&定位下一个已修改的记录
IF lModified
    nResult = MESSAGEBOX;
        ("Record has been modified. Save?", 4+32+256, "Data Change")

    && 提交当前值并为用户提供选项
    &&询问是否还原对当前字段所做的修改
    IF nResult = 7
        = TABLEREVERT (.F.)
    ENDIF
ENDIF

&&SKIP 确保最后一个修改内容也已写入
SKIP
IF EOF()
    =MESSAGEBOX( "already at bottom")
    SKIP -1
ENDIF
THISFORM.Refresh
```

(2) 管理、检测和消除冲突。

在一个多用户环境中，通过选择打开、缓冲并锁定数据的时间和方式，可以更高效地进行数据更新操作。应该避免和减少访问记录或表时发生冲突的时间，同时必须预测到不

可避免的冲突将导致什么样的后果，并对这些冲突进行管理。冲突一般在一个用户试图锁定当前正被其他用户锁定的记录或表时发生。应用程序中应包含管理冲突的例程。如果没有冲突检测和管理，系统将会发生死锁现象。死锁通常发生在这样的情况下：第一个用户已经锁定了一个记录或表，现在试图去锁定已经被第二个用户锁定的记录，而第二个用户又反过来试图锁定第一个用户锁定的记录。尽管这种情况很少发生，但记录或表被锁定的时间越长，这种死锁的可能性就会越来越大。

设计一个多用户的应用程序和为一个单用户系统添加网络支持服务，都要求找到错误并对其进行处理。使用 Visual FoxPro 记录缓冲和表缓冲可以部分地简化这种工作。如果试图锁定一个已被其他用户锁定的记录或表， Visual FoxPro 将返回出错信息。可使用 SET REPROCESS 自动处理不成功的锁定操作。此命令与 ON ERROR 例程和 RETRY 命令组合，可以继续或取消锁定的操作。

在数据更新操作过程中，您可能希望确定哪些字段已经更改，确定已更改字段的原有值或当前值。 Visual FoxPro 的缓冲和 GETFLDSTATE( )、 GETNEXTMODIFIED( )、OLDVAL( )及 CURVAL( )函数可以提供这样的功能。通过它们来确定哪些字段已经更改；查找已更改数据；比较当前值、原有值及已编辑的值。这样，可以决定如何处理错误或冲突。

1）管理冲突。下面的示例使用 SET REPROCESS 命令自动地重处理一个失败操作。

代码如下：

```
    ON ERROR DO ErrorName WITH ERROR(),MESSAGE() &&出错时本例程运行
SET EXCLUSIVE OFF       &&以非独占方式打开文件
SET REPROCESS TO AUTOMATIC  &&自动重处理不成功的锁定
USE customertemp        &&打开表

    &&必要时创建 APPEND FROM 表
IF !FILE('Customer.dbf')
    COPY TO Customer
ENDIF

    DO app_blank        &&主例程从此处开始
DO rep_next
DO rep_all
DO rep_curr
DO add_Records
    ON ERRORNAME &&主例程在此处结束

&&追加空白记录的例程
    PROCEDURE app_blank
    APPEND BLANK
RETURN
```

```
ENDPROC

    && 替换当前记录数据的例程
    PROCEDURE rep_next
    REPLACE NEXT 1 contact WITH ;
        PROPER(contact)
RETURN
ENDPROC

    &&      替换所有记录数据的例程
    PROCEDURE rep_all
    REPLACE ALL contact WITH ;
        PROPER(contact)
    GO TOP
RETURN
ENDPROC

    &&      替换当前记录数据的例程
    PROCEDURE rep_curr
    REPLACE contact WITH PROPER(contact)
RETURN
ENDPROC

    && 从其他文件中追加记录的例程
    PROCEDURE add_Records
    APPEND FROM Customer
RETURN
ENDPROC
```

2）检测和处理冲突。以下讲述检测和处理冲突的函数和命令。

• GETFLDSTATE( )可以对非缓冲数据进行操作，但在启用了记录缓冲的情况下，此函数更有效。比如，将 GETFLDSTATE( )用于一个表单的“跳过”按钮代码中。当您移动记录指针时， Visual FoxPro 将检查记录中所有字段的状态，如下例所示：

```
lModified = .F.
FOR nFieldNum = 1 TO FCOUNT( )    && 检查所有字段
if GETFLDSTATE(nFieldNum) = 2   && 记录已被修改
lModified = .T.
EXIT    &&    可以在此处插入一个记录
ENDIF
ENDFOR
```

• GETNEXTMODIFIED( )以0作为参数，查找第一个已修改的记录。如果其他用户对缓冲表进行了修改，则您的缓冲区中 TABLEUPDATE( )命令遇到任何修改都将导致冲突。可以用 CURVAL( )、 OLDVAL( )和 MESSAGEBOX( )计算冲突值并解决冲突，CURVAL( )返回磁盘上记录的当前值，而 OLDVAL( )返回记录缓冲时的值。

• 使用 OLDVAL( )函数返回一个缓冲字段的值。

• CURVAL( )将返回一个在编辑操作执行前磁盘缓冲字段的当前值。在共享环境中，可以创建一个出错处理过程，比较当前值与原有值，并决定是接受当前的修改还是更早一些的修改。

下面给出一个示例，使用 GETNEXTMODIFIED( )、 CURVAL( )和 OLDVAL( )函数，在更新操作过程中给用户提供选择。

```
DO WHILE GETNEXTMODIFIED(nCurRec) <> 0    && 循环遍历缓冲区
GO nCurRec
= RLOCK( )  &&锁定已修改的记录

FOR nField = 1 TO FCOUNT(cAlias)  && 查找冲突
    cField = FIELD(nField)

&& 比较原有值和磁盘上的当前值，请用户指示如何处理冲突
    IF OLDVAL(cField) <> CURVAL(cField)
        nResult = MESSAGEBOX("Data was ;
            changed by another user. ;
            Keep changes?", 4+48+0, ;
            "Modified Record")

        IF nResult = 7  && 如果用户选择 'No'，则还原此记录，然后解锁
            = TABLEREVERT(.F.)
            UNLOCK RECORD nCurRec
        ENDIF
    ENDIF
ENDFOR

&& 查找下一个已修改的记录
nCurRec = GETNEXTMODIFIED(nCurRec)
ENDDO
= TABLEUPDATE(.T., .T.)        && 对所有记录更新
```

4. 事务管理更新（原子事务）

在数据库的操作当中，即便使用了缓冲技术，事情有时也会出错。如果希望保护更新

操作，并从一整段代码中还原回来，可以使用事务概念。在应用程序中添加事务所提供的保护机制是很强大的，其超过了记录缓冲和表缓冲提供的保护功能，它将整段代码作为一个受保护的、可恢复的单元。而且事务是可以嵌套的，用户可以嵌套使用事务，并用它保护已操作的缓冲更新。事务的概念在银行系统和一些大的安全级别较高的数据库系统中经常被采用。Visual FoxPro 事务只能用于数据库中的表和视图。

事务类似一层外包装，其用来缓冲应用程序对内存或硬盘的数据更新操作，而不是直接对数据库进行更新。实际的数据库更新在事务完全结束之后进行。如果由于某种原因，系统不能执行对数据库的更新操作，就结束整个事务，而不执行任何操作，即：任何动作将被结束，进程回到了初始调用的位置，就象从未发生任何事情一样。

(1) 控制事务的命令。

Visual FoxPro 提供了三个命令和一个函数来控制事务处理，如表 9.4 所示：

表 9.4 控制事务的函数和命令

BEGIN TRANSACTION	初始化一个事务
TXNLEVEL( )	确定当前事务的等级
ROLLBACK	取消最近一条 BEGIN TRANSACTION 语句以来所做的全部修改
END TRANSACTION	锁定记录，将最近一条 BEGIN TRANSACTION 语句以来对数据库中表所做的全部修改写入磁盘，然后解锁记录

可以用事务缓冲可以对以下文件修改，如：表、结构化的*.CDX 文件以及与数据库表相关的备注文件。涉及内存变量和其他对象的操作不属于事务，因此不能完全复原这些操作。

当使用远程表存储的数据时，事务命令只更新视图临时表的本地备份的数据；对远程基表的更新不起作用。要对远程表启用人工事务，请使用 SQLSETPROP( )，然后用 SQLCOMMIT( )和 SQLROLLBACK( )控制事务处理。

一般来说，除非采用 TABLEUPDATE( )包装调用，事务最好和记录缓冲一起使用，而不与表缓冲一起使用。如果在事务中使用了 TABLEUPDATE( )命令，那么可以恢复失败的更新操作，找到失败的原因，然后重试 TABLEUPDATE( )命令而不丢失任何数据，这样，就确保了更新操作一定是所谓的“全都做或全都不做”的操作。

尽管在正常情况下，简单的事务处理能够提供安全的数据更新操作，但是它并不能提供对系统失败的完全保护。如果在处理 END TRANSACTION 命令期间断电或产生其他系统中断，则数据更新仍将失败。当然，这不是系统和软件设计上的问题。

(2) 使用事务。

1) 一个事务以 BEGIN TRANSACTION 命令开始，以 END TRANSACTION 或 ROLLBACK 命令结束。只有 END TRANSACTION 语句，而没有 BEGIN TRANSACTION 与之相匹配将会出错。

2) ROLLBACK 语句前必须有 BEGIN TRANSACTION 。

3 ）除非应用程序中止（这将导致回滚操作），事务一旦开始，到遇到相应的 END TRANSACTION 语句这一期间，将保持有效，在调用程序、调用函数的情况下也是如此。

4 ）对于涉及事务数据的查询，Visual FoxPro 在使用磁盘数据前先使用在事务缓冲区内的缓冲数据，确保使用最近的数据。

5 ）如果在事务处理过程中应用程序中止，则所有操作恢复到最初的情况。

6 ）事务处理只有在数据库容器中进行。

7 ）如果 INDEX 命令改写了一个已有的索引文件，或者任一索引文件已打开，则不能使用 INDEX 命令。

8 ）事务只在数据工作期中起作用。

(3) 事务处理完成的锁定工作。

1 ）当一个命令直接或间接调用事务时，Visual FoxPro 将强制锁定。任何系统或用户的直接或间接命令将被缓冲，直到 ROLLBACK 或 END TRANSACTION 命令结束事务。

2 ）如果在事务处理中使用了锁定命令，如 FLOCK( )或 RLOCK( )，则 END TRANSACTION 语句将不会解锁。因此必须显式地解除任何在事务处理中进行的显式锁定。应该使包含 FLOCK( )或 RLOCK( )命令的事务尽量简短。否则，用户会长时间不能访问被锁定的记录。

(4) 嵌套事务处理。

在一些情况下，用户可能需要使用嵌套事务处理（ nesting transaction ）。嵌套事务处理是一些表格更新操作的逻辑组，它们可以从并行处理中独立出来。在使用嵌套事务处理时，命令 BEGIN TRANSACTION 和 END TRANSACTION 并不需要出现在同一函数或过程中。下列规则适用于嵌套事务处理：

1 ）可以嵌套五层 BEGIN TRANSACTION 和 END TRANSACTION 。

2 ）直到最外层的 END TRANSACTION 被调用时，才执行嵌套事务中的更新。

3 ）在嵌套事务中，END TRANSACTION 只对最近出现的 BEGIN TRANSACTION 所引起的事务进行操作。

4)在嵌套事务中，ROLLBACK 语句只对最近出现的 BEGIN TRANSACTION 所引起的事务进行操作。

5 ）在嵌套事务中对同一数据的更新操作，最内层的更新优先于同一段中的其他任何设计更新操作。

嵌套事务处理示例：

示例 1 ：

该示例清除以前进行的事务出来操作。然后将 Multilock 设置为 ON ，将 Exclusive 设置为 OFF ，建立缓冲环境。接着使能乐观表格缓冲，同时程序修改一系列的记录。如果数据更新失败，那么整个事务都将恢复，直到找出失败的原因并纠正。

程序代码如下：

```
DO WHILE TXNLEVEL( ) > 0    && 脱离其他事务
ROLLBACK
ENDDO
```

```
    CLOSE ALL  && 建立缓冲环境
SET MULTILOCKS ON
SET EXCLUSIVE OFF

    PEN DATABASE test
USE mrgtest1
=CURSORSETPROP('buffering',5)     && 启用开放式表缓冲
GO TOP
REPLACE fld1 WITH "changed"  && 更改一个记录
SKIP
REPLACE fld1 WITH "another change"     &&更改另一个记录
=MESSAGEBOX("modify first field of both records on another machine")

    BEGIN TRANSACTION     &&开始事务 1，并且非强制的更新所有已修改记录
lSuccess = TABLEUPDATE(.T.,.F.)
    IF lSuccess = .F.          &&如果更新失败， RollBack 事务
    ROLLBACK

    &&由 AERROR( ) 获得错误信息
    &&确定失败原因

    =AERROR(aErrors)
    DO CASE

    && 如果是触发器失败，进行相应处理
    CASE aErrors[1,1] = 1539

    && 如果是字段不接受 null 值，进行相应处理
    CASE aErrors[1,1] = 1581
    ...
    && 如果是违反了字段规则，进行相应处理.
    CASE aErrors[1,1] = 1582

    && 如果一个记录被另一个用户更改，则定位第一个已修改的记录
    CASE aErrors[1,1] = 1585
        nNextModified = getnextmodified(0)

        && 循环遍历所有已修改的记录
        DO WHILE nNextModified <> 0
```

```
&& 从第一个记录开始
    GO nNextModified

    && 锁定每个记录以确保可以进行更新
    =RLOCK()

    && 检查每个字段的修改
    FOR nField = 1 to FCOUNT()
        cField = FIELD(nField)

        && 比较缓冲值和磁盘上的值，然后出现对话框
        if OLDVAL(cField) <> ;
            CURVAL(cField)
            nResult = MESSAGEBOX("Data has been changed ;
            by another user. Keep changes?",4+48,"Modified Record")

            && 如果用户响应“ No ”，则还原一个记录
            IF nResult = 7
                =TABLEREVERT(.F.)  &&解锁
                UNLOCK record ;
                    nNextModified
            ENDIF
            EXIT          &&脱离“ FOR nField... ”循环
        ENDIF
    ENDFOR
ENDDO       &&获得下一个已修改的记录

&& 开始事务 2 并强制性地更新所有未还原记录
BEGIN TRANSACTION
=TABLEUPDATE(.T.,.T.)
END TRANSACTION        &&结束事务 2
UNLOCK      && 解锁

&& 如果记录正被其他用户使用，进行相应处理
CASE aErrors[1,1] = 1700
...
&& 如果违反行规则，进行相应处理
CASE aErrors[1,1] = 1583
...
```

```
        && 如果违反唯一索引规则，进行相应处理
        CASE aErrors[1,1] = 1884
        ...
        && 否则，出现对话框
        OTHERWISE
            =MESSAGEBOX( " Unknown errormessage: " + STR(aErrors[1,1]))
        ENDCASE
 ELSE

    && 结束事务 1
    END TRANSACTION
ENDIF
```

示例 2：

下面的示例使用了事务包装对远程表的数据写入操作。代码如下：

```
    hConnect = CURSORGETPROP('connecthandle')
SQLSETPROP(hConnect, 'transmode',DB_TRANSMANUAL)
lSuccess = TABLEUPDATE(.T.,.F.)
IF lSuccess = .F
    =SQLROLLBACK (hConnect)
    =AERROR(aErrors)
    DO CASE
    CASE aErrors[1,1] = 1539
    ...
    CASE aErrors[1,1] = 1581
    ...
    CASE aErrors[1,1] = 1582
    ...
    CASE aErrors[1,1] = 1585
        nNextModified = GETNEXTMODIFIED(0)
        DO WHILE nNextModified <> 0
            GO nNextModified
            FOR nField = 1 to FCOUNT()
                cField = FIELD(nField)
                IF OLDVAL(cField) <> ;
                    CURVAL(cField)
                    nResult = MESSAGEBOX("数据已经被其他用户；
                                修改。保留修改吗?",4+48,"被修改的记录")
                    IF nResult = 7
```

```
                    =TABLEREVERT(.F.)
                ENDIF
                EXIT
            ENDIF
        ENDFOR
        nNextModified = ;
        GETNEXTMODIFIED(nNextModified)
    ENDDO
    =TABLEUPDATE(.T.,.T.)
    =SQLCOMMIT(hConnect)
CASE aErrors[1,1] = 1700
...
CASE aErrors[1,1] = 1583
...
CASE aErrors[1,1] = 1884
...
OTHERWISE
    =MESSAGEBOX( "Unknown error message: " + STR(aErrors[1,1]))
ENDCASE
ELSE
=SQLCOMMIT(hConnect)
ENDIF
```

9.1.2　加密算法

加密是一种强有力的算法编码技术。加密能保护您的计算机文件和传输，使其他用户、窃听者和未授权者无法访问它们。它通过将数据变换为仅能通过使用密钥才可读的形式来做到这一点。

任何良好的数据安全系统都必须包括加密。加密通过以下各点使安全风险减到最低限度：

机密性：用户用以确保其秘密的手段。加密可以使用户们可以在不安全的信道上拥有安全的通信。

整体性：验证数据流的完整性。通过加密可以确保病毒未影响数据的健康。

访问控制：限制访问资源的手段。

证实：用于验证发送数据的当局，并证实数据在发送阶段未被修改的机制。

1. 加密的工作方法

对于加密，原始信息或文件（明文）及密钥首先被编码算法扰乱或加密。从而产生密文。利用该密码和一解码算法，此加密算法可以被逆转。

例如：当读者想发送一个消息给您的朋友 A 时，您希望除了 A 之外没有人可以读取您的消息。您可以使用密钥将您的消息（明文）加密，从而构成密文。A 使用解密密钥将该密文解密，从而阅读该消息。攻击者可能试图或者获得该密钥或者不使用密钥而恢复明文。在安全密码系统中，明文不可能从密文中恢复出来，除非使用解密密钥。

2. 公开加密技术和秘密加密技术

传统的加密技术以一消息的发送者和接收者知道并使用相同的秘密密钥为基础；该发送者使用秘密密钥将消息加密，而接收者则使用相同的秘密密钥将该消息解密。这种方法被称为秘密加密技术或对称加密技术。其主要问题是要在没有任何其他人发现的情况下，发送者和接收者协商好一个密钥。当他们位于不同的地方时，则必须委托一信使，一电话相同或其他的传输系统，以使他们正在联系的秘密密钥不被泄露。听到或截取到传输中的该秘密密钥的任何人以后都可以读懂使用该密钥加密的所有信息。对秘密密钥来说，密钥的管理是很难保证的。

公开加密技术就不可能出现这种问题。

公开加密主要是为了解决密钥管理的问题。在该系统中，每个人都得到一对密钥：一个是公共密钥；一个是私有密钥。每人都公布其公用密钥，但保留其私有密钥。这就消除了发送者和接收者共享秘密信息的必要性：所有通信都仅涉及到公开密钥而无秘密密钥的传输与共享。使用者无需再相信某些通信信道的安全。任何人都可以使用公开信息发送机密信息；该机密消息只能被指定的接收者所独有的秘密密钥解密。

读者也许会问道："它的工作机理是什么呢？"

其实，使用公开密钥进行数据加密的机理在于数学中数论应用。详细的加密技术是一门学科，这里我们讲一下关于如何使用公用密钥加密技术来实现数字签名。

数字签名的技术是使用了双重的加密，其过程是这样的：

(1) 使用接收者的公用密钥进行加密；
(2) 使用发送者的私有密钥进行加密；
(3) 发生密文；
(4) 使用发送者的公用密钥进行解密；
(5) 使用接收者的私有密钥进行解密。

于是就完成了数字签名工作。

9.2 客户/服务器模式

客户／服务器应用程序将本地计算机上 Visual FoxPro 的功能，以及远程服务器所提供的存储和安全优点结合在一起。可以在本地建立应用程序的原型，然后使用升迁向导把应用程序迁移到客户／服务器环境。

随着 Visual FoxPro 的发布，Microsoft 已经把 FoxPro 定位在客户/服务器结构的前端（客户端）。Visual FoxPro 包含了大量访问后端信息的途径，加上相当快的本地数据库引擎和数据字典，它提供了真正的客户/服务器开发所需的最好选择。

9.2.1　什么是客户/服务器模式

客户/服务器系统的体系结构有以下两个特点：一是集合智能用户工作站作为有效平台使用；二是平台和软件之间的互操作性。客户/服务器结构包括连接在一个网络上的多台计算机。那些处理应用程序、请求来自另一台计算机的服务的计算机称为客户机（Client）。而处理数据库的计算机称为服务器（Server）。所有用户都拥有自己的计算机来处理应用程序。客户机可以是大型机、小型机或PC机。但是由于PC机的成本比较低，所以建议使用PC机为客户机。同样地，服务器通常一台PC机，但当需要很强大的能力时，也可以考虑使用大型机或小型机。

(1) 客户机。

客户机运行那些是用户能阐明其服务请求的程序并将这些请求传送到服务器。由客户机执行的计算称为前端处理。前端处理具有所有与提供操作和信使数据的相关功能。

客户软件由网络接口软件、支持用户需求的应用程序以及实现网络功能的实用程序。

(2) 服务器。

在服务器上执行的操作称为后端处理。后端硬件是一台管理数据库资源和数据库引擎功能（如：存储、操作和保护数据）的计算机。在大型机环境下，后端网络提供后端大型计算机至大容量存储设备、控制器以及文件服务器的连接。

(3) 中间件。

中间件是一个软件层。它保护应用程序开发人员避免受到各种通信协议、操作系统以及数据库管理系统的影响。它为建立可与以前沿袭下来的应用程序并存的新应用程序大下了基础。

中间件有好几种类型。它们包括应用程序设计接口（API）、远程过程调用（RPC）、网络通信、数据库访问以及计算机辅助软件工程（CASE）工具。

9.2.2　如何访问服务器的数据

Visual FoxPro为创建功能强大的客户/服务器应用程序提供了一些专用工具。Visual FoxPro客户/服务器应用程序将Visual FoxPro功能强、速度快、图形化用户界面以及高级的查询、报表和处理等优点与多用户访问、内置安全性、可靠的事务处理和日志以及ODBC数据源等功能紧密地结合在一起。Visual FoxPro和服务器之间的协作可以为用户提供功能强大的客户/服务器解决方案。

1. 设计客户机/服务器应用程序

用Visual FoxPro创建一个快速、高性能的客户/服务器应用程序涉及到如何利用Visual FoxPro高速引擎的问题。可以考虑利用一些新技术来达到这一目的，例如，使用基于集合的数据访问技术；创建参数化查询而且仅仅下载所需的数据；合理安排各数据表的位置；协调使用Visual FoxPro存储过程和远程存储过程等。但在使用新技术之前，必须首先对将要使用的系统进行大致的分析。在设计一个本地应用程序或文件服务器应用程序时，必须确定应用程序需要使用或创建的查询、表单、菜单和报表。在设计一个客户/服

务器应用程序时，除了常规系统分析外，还需要对客户／服务器应用程序的特点进行分析，详细地考虑查询、表单、菜单和报表所使用的数据存放在何处，如何访问这些信息等。

当确定了客户／服务器应用程序的基本组成部分以后，就可以开始着手于应用程序的访问和更新数据设计了。

(1) 下载数据。

1)下载有用的数据. 为了使客户／服务器应用程序快速、高效，必须考虑一个重要的因素：尽量减少从服务器上下载的数据量。由于客户／服务器应用程序需要访问远程服务器上大量的数据，为了提高执行速度和减少下载的数据量，可以使用基于集合的数据访问技术来筛选将要下载的数据。

基于集合的数据是通过网络的传输，将客户机上的请求发送给服务器，在服务器上经过对数据的选择后传回到客户机上的数据信息流。要访问远程数据，必须使用 SQL SELECT 语句从一个大的数据存储地点中选择一个数据集合并传回客户机。

2 ）使用标准化查询。访问服务器上数据的有效方法就是每次仅下载所需要的数据，需要得到其他记录或新记录时，再重新查询。可以使用标准化的 SELECT 语句下载数据集合，需要访问新记录或新的数据集合时，再使用 REQUERY( )函数。

不能对远程服务器数据发出 GO BOTTOM 命令，这样做有以下缺点：

- 下载的数据量巨大将对网络资源造成不必要的负担。
- 处理不需要的数据会使用应用程序的性能降低。
- 因为远程数据的改变只有在重新查询的时候才能反映到本地临时表中，所以降低了本地临时表中数据的准确性。

例如，若要建立一个客户／服务器应用程序用来查阅客户的订单，可创建一个远程视图来访问 Customer 表。然后创建另一个远程视图来访问 Orders 表，并基于 cust_id 字段将该视图参数化。这样，就可以把当前的顾客记录做为 Orders 表的视图的参数。

3 ）为所有的远程数据创建远程视图。好的客户／服务器方案是为所有的远程数据创建远程视图。在 Customer 表的远程视图中，使用 SELECT 语句选择某个地区的客户，这样就可以限制下载到远程视图中的客户记录的数量。然后再对关系中多方 Orders 表创建一个远程视图，并以客户 ID 值为参数。该方案可以取得更少量的记录。使用 SKIP 命令可以在关系的“一”方(即 Customer 视图)中移动。使用 REQUERY( )函数可以访问多方(即 Orders ）中的新数据。这种方案中，既限制（或筛选）了关系中的“一”方又筛选了关系中的多方，但仍能使用 SKIP 命令在筛选过的数据之中定位。如果关系中的“一”方经过筛选以后，仍然能够提供足够的信息满足后续的一些查询，而不需要再次向远程服务器发出请求，那么推荐使用这种方案。

4 ）使用主关键字访问一对多关系。效率最高的客户／服务器方案是放弃使用开销大的 SKIP 命令。通过创建一个表单，让用户输入或选择一个客户 ID，然后把它既做为 Customer 表的远程视图的参数，又做为 Orders 表的远程视图的参数。例如，可以创建一个一对多表单，其中“一”方显示客户信息，而“多”方的订货信息在表格中显示。该表格可以与表单“一”方中选定的客户 ID 相联系。然后可以把 CURSORSETPROP( )的 MaxRow 属性设置为 1，使用下列代码填充表单的“一”方：

SELECT * FROM customer WHERE customer.cust_id = ?cCust_id

当用户想查看另一个客户的记录时，必须输入一个新 ID 。表单用新 ID 值重新查询数据中的订单记录，并用新的记录数据刷新表格控件。使用这些技术，可以使应用程序仅仅下载需要的数据。从而减少了下载的数据量，提高了网络的响应速度，而且，在显示所请求的信息之前重新查询数据，提供最新信息。

该方法在想要用任意主关键字随机访问一对多关系时使用。在打开一个表单时，还可以把一些主关键字下载到控件中，比如一个下拉列表，并且提供一个控件，让用户在需要的时候可以选择用来刷新主关键字值的列表。

5 ）视图。视图是开发客户 / 服务器应用程序的核心方法。远程视图是一种强有力的技术，使用它可以从一个远程服务器中选择所需要的数据并下载到本地 Visual FoxPro 的临时表中，同时还可以查看并更新远程数据。一个视图基本上是使用一个 SQL SELECT 语句所得到的一个结果集合。因为视图定义保存在数据库中，所以视图具有永久性。视图定义具有可以设置的属性，借助其属性可以对活动视图临时表做进一步定制。视图是创建可更新结果集合的最佳数据定义工具。使用本地视图建立一个本地原型，然后通过使用“升迁向导”把本地视图转换为远程视图。

6 ）SQL Pass-Though 。SQL Pass-Through 技术使用 Visual FoxPro SQL Pass-Through 函数对远程服务器进行直接访问。这些函数提供了视图所不能及的服务器访问和控制能力。例如，可以在远程服务器上设置服务器属性，执行数据定义，并且访问服务器存储过程等。

SQL Pass-Through 是创建只读结果集合和使用其他服务器上 SQL 语法的最好工具。视图是一个 SQL SELECT 语句的结果集合， SQL Pass-Through 与此不同，它可以使用 SQLEXEC( )函数向服务器发送符合服务器语法的任何命令。即：通过 SQL Pass-Through 函数可以对服务器的进行操作，就象在服务器上直接运行一样。

下表 9.5 列出了 Visual FoxPro 的 SQL Pass-Through 函数：

表 9.5　Pass-Through 函数

SQLCANCEL( )	SQLCOLUMNS( )	SQLCOMMIT( )
SQLCONNECT( )	SQLDISCONNECT( )	SQLEXEC( )
SQLGETPROP( )	SQLMORERESULTS( )	SQLPREPARE( )
SQLROLLBACK( )	SQLSETPROP( )	SQLSTRINGCONNECT( )
SQLTABLES( )		

读者还可以使用视图和 SQL Pass-Through 函数相结合的方法来访问服务器上的数据。由于视图创建比较容易，而且能够提供自动缓冲和更新功能，因此可以使用视图完成大部分的数据管理任务。同时使用 SQL Pass-Through 执行远程服务器上的特定任务。例如数据定义和服务器存储过程的创建和执行等。所以建立一个 Visual FoxPro 客户 / 服务器应用程序的最有效办法就是组合使用视图和 SQL Pass-Through 技术。

2. 实现客户/服务器应用程序

除了通过升迁本地原型程序以外，还可以利用远程视图开发客户／服务器应用程序，从而访问远程服务器数据库中存储的数据信息，利用远程服务器上的安全性管理和事务处理能力。在用远程视图完成数据管理任务时，可以使用 SQL Pass-Through 技术在服务器上创建对象，运行服务器存储过程，使用服务器上的语法执行各种命令，从而大大提高应用程序的性能。

(1) SQL pass-through 技术。

SQL Pass-Through 技术可以直接把 SQL 语句发送给服务器。由于被传递的 SQL 语句在后台服务器上运行，所以能够在很大程度上提高客户／服务器应用程序的性能。

1) Pass-Through 的优点。SQL Pass-Through 技术具有如下优点：

- 可以使用服务器的特有功能。例如，存储过程和基于服务器的内部函数。
- 可以使用服务器所支持的 SQL 扩展功能，而且可以进行数据定义、服务器管理和安全性管理。
- 对 SQL Pass-Through 的更新、删除和插入语句拥有更多控制权。对远程事务拥有更多控制权。

2) Pass-Through 的缺点。SQL Pass-Through 技术具有如下缺点：

- 一个 SQL Pass-Through 查询可以快速得到远程数据结果，但是在默认情况下得到的结果不能更新，而它保存在一个活动视图的临时表中。要使得此临时表可更新，必须使用 CURSORSETPROP( )函数设置它的属性。相反地，一个可更新的远程视图在更新远程数据之前不需要设置属性。因为属性设置保存在数据库的视图定义中。
- 不能使用图形化的视图设计器，而必须在命令窗口或程序中直接输入 SQL 命令。

(2) SQL Pass-Through 函数。

SQL Pass-Through 函数在表 9.5 中已经列出，下表 9.6 将列出它们的使用目的：

表 9.6 SQL Pass-Through 函数

任 务	函 数	目 的
	SQLCONNECT( )	为 SQL Pass-Through 操作联接一个数据源
联接管理	SQLSTRINGCONNECT( )	使用 ODBC 联接字符串语法联接数据源
	SQLDISCONNECT( )	断开一个对 ODBC 数据源的联接，使指定的联接句柄失效
	SQLCANCEL( )	在一个活动联接上，取消异步执行的 SQL 查询
	SQLEXEC( )	在一个活动联接上执行 SQL Pass-Through 查询；或者返回已生成的结果集合的数目，如果 SQLEXEC( )仍在执行（异步处理），则返回 0 值
SQL 语句的执行和控制	SQLMORERESULTS( )	把第二个结果集合放到临时表中。如果创建结果集合的语句仍在执行，则返回 0 值

（续表）

任　务	函　数	目　的
	SQLPREPARE( )	预编译数据源上的 SQL 语句，同时带有 Visual FoxPro 参数。即：在 SQL 命令中，将所有参数保存为实际的参数表达式
	SQLCOMMIT( )	申请提交一个事务
	SQLROLLBACK( )	申请恢复一个事务
	SQLCOLUMNS( )	将列名及每列的信息存入一个临时表。如果成功，则返回 1；如果仍在执行，则返回 0
数据源信息	SQLTABLES( )	将数据源中的表名存入一个临时表。如果成功，则返回 1。如果仍在执行，则返回 0
其他控制	SQLGETPROP( )	获得活动联接的属性
	SQLSETPROP( )	设置活动联接的属性

注意：如果 SET ESCAPE 设置为 ON，则 SQLEXEC( )、SQLMORERESULTS( )、SQLTABLES( )和 SQLCOLUMNS( )语句可以通过按 ESC 键以同步方式取消。此外，也可以通过发出 SQLCANCEL( )命令以异步方式取消这些语句。其他 SQL Pass-Through 语句都以同步方式工作，不能被中断。

(3) 使用 SQL Pass-Through 函数。

1）使用前提。

- 保证系统能与数据源相连。使用 ODBC Test 工具进行检查。
- 用 SQLSTRINGCONNECT( )或 SQLCONNECT( )函数建立与数据源的联接。

例如：如果要把 Visual FoxPro 联接到 SQL Server 数据源 sqlserver 上，那么必须以系统管理员（用户标识 userid）身份注册，并给出密码 password，即，使用如下命令：

```
nConnectionHandle = SQLCONNECT('sqlserver','userid','password')
```

2）访问服务器存储过程。使用 Visual FoxPro SQL 技术，可以建立和执行在远程服务器上的存储过程。存储过程可以增强 SQL 的有效性、灵活性和改善其性能，也能提高 SQL 语句和批处理的性能。许多的服务器都提供了存储过程，以便于用户对服务器的数据库对象进行定义和操作，而且也允许执行服务器系统和用户管理的任务。

可以使用带有该存储过程名称的 SQLEXEC( )函数调用一个服务器存储过程。例如，以下代码将显示 SQL Server 上名为 who 的存储过程的调用结果。该 SQL Server 使用一个活动的联接与名字为 sqlserver 的数据源相联。

```
nConnectionHandle = SQLCONNECT('sqlserver')
? SQLEXEC(nConnectionHandle, 'userpub')
? SQLEXEC(nConnectionHandle, 'who')
BROWSE
```

3）返回多个结果集。如果执行了一个存储过程，该存储过程包含了服务器上的语法，即 SELECT 语句，则每个结果集将分别返回到一个独立的 Visual FoxPro 临时表中。使用这些临时表，可以从一个服务器存储过程向 Visual FoxPro 客户机上返回值或者参数。

用 SQLEXEC( )函数选择使用服务器语法并返回多个结果集。例如，下面的代码将建立和执行了一个 SQL 服务器存储过程 store_procedure，该过程返回三个 Visual FoxPro 的临时表 sqlresult0, sqlresult1 和 sqlresult2 。

```
=SQLEXEC(nConnectionHandle,'create procedure store_procedure as ;
    select * from sales; select * from authors;
    select * from titles')
=SQLEXEC(nConnectionHandle,'execute store_procedure')
```

9.2.3 示例 s

以下示例是有关 Client/Server 模式下的应用程序的一部分，由于篇幅的关系，只能录入以下的代码，而这些代码的作用在于帮助读者处理事务、锁定和使用 SQL Pass-Through 技术实现数据库的各种功能单元。

代码如下：（仅包含部分过程）详细情况请读者参考 Visaul FoxPor 的例子。

```
PROCEDURE refreshstatus
    LOCAL llOffline
    this.pageframe1.page1.edtCursorInfo.value = CS_CURSOR_INFO_LOC
    this.getStatus
    *刷新三种游标的状态
    DO CASE
        CASE this.pageframe1.page1.opgCursorType.value = 1
            * 刷新表信息
            this.pageframe1.page1.lblCursorName.caption = CS_STATTBL_LOC +
UPPER(CS_TABLE)
            this.pageframe1.page1.cmdCreateView.Enabled = .F.
            this.pageframe1.page1.cmdDropView.Enabled = .F.
            this.pageframe1.page1.chkOnView.Enabled = .F.
            IF oEngine.DatabaseIsOpened AND INDBC(CS_TABLE,'TABLE')
                this.pageframe1.page1.cmdOpen.Enabled = .T.
                this.pageframe1.page1.chkExcl.Enabled = .T.
                this.pageframe1.page1.cmdUpsize.Enabled =
INDBC(CS_CONNECTION,'CONNECTION')
                this.pageframe1.page1.cmdReset.Enabled =
INDBC(CS_INITIALTABLE,'TABLE')
            ELSE
                this.pageframe1.page1.cmdOpen.Enabled = .F.
                this.pageframe1.page1.chkExcl.Enabled = .F.
                this.pageframe1.page1.cmdUpsize.Enabled = .F.
                this.pageframe1.page1.cmdReset.Enabled = .F.
```

```
            ENDIF
        CASE this.pageframe1.page1.opgCursorType.value = 2
            * 刷新本地视信息
            this.pageframe1.page1.lblCursorName.caption = CS_STATLV_LOC +
UPPER(CS_LOCAL_VIEW)
            this.pageframe1.page1.cmdUpsize.Enabled = .F.
            this.pageframe1.page1.cmdReset.Enabled = .F.
            IF oEngine.DatabaseIsOpened AND INDBC(CS_LOCAL_VIEW,'VIEW')
                this.pageframe1.page1.cmdOpen.Enabled = .T.
                this.pageframe1.page1.chkExcl.Enabled = .T.
                llOffline = !EMPTY(DBGETPROP(CS_LOCAL_VIEW, 'view',
'offlinepath'))
                this.pageframe1.page1.cmdCreateView.Enabled = !llOffline
                this.pageframe1.page1.cmdDropView.Enabled = llOffline
                this.pageframe1.page1.chkOnView.Enabled = llOffline
            ELSE
                this.pageframe1.page1.cmdOpen.Enabled = .F.
                this.pageframe1.page1.chkExcl.Enabled = .F.
                this.pageframe1.page1.cmdCreateView.Enabled = .F.
                this.pageframe1.page1.cmdDropView.Enabled = .F.
                this.pageframe1.page1.chkOnView.Enabled = .F.
            ENDIF
        CASE this.pageframe1.page1.opgCursorType.value = 3
            * 刷新远程视信息
            this.pageframe1.page1.lblCursorName.caption = CS_STATRV_LOC +
UPPER(CS_REMOTE_VIEW)
            this.pageframe1.page1.cmdUpsize.Enabled = .F.
            this.pageframe1.page1.cmdReset.Enabled = .F.
            IF oEngine.DatabaseIsOpened AND INDBC(CS_REMOTE_VIEW, 'VIEW')
AND INDBC(CS_CONNECTION, 'CONNECTION')
                this.pageframe1.page1.cmdOpen.Enabled = .T.
                this.pageframe1.page1.chkExcl.Enabled = .T.
                llOffline = !EMPTY(DBGETPROP(CS_REMOTE_VIEW, 'view',
'offlinepath'))
                this.pageframe1.page1.cmdCreateView.Enabled = !llOffline
                this.pageframe1.page1.cmdDropView.Enabled = llOffline
                this.pageframe1.page1.chkOnView.Enabled = llOffline
            ELSE
                this.pageframe1.page1.cmdOpen.Enabled = .F.
```

```
                this.pageframe1.page1.chkExcl.Enabled = .F.
                this.pageframe1.page1.cmdCreateView.Enabled = .F.
                this.pageframe1.page1.cmdDropView.Enabled = .F.
                this.pageframe1.page1.chkOnView.Enabled = .F.
            ENDIF
        ENDCASE
    ENDPROC
    PROCEDURE getstatus
        LOCAL lcStatus, lcView
        lcStatus = ""
        oEngine.SetErrorOff = .T.
        oEngine.HadError = .F.
        IF this.pageframe1.page1.opgCursorType.value = 1
            * 表
            USE CS_TABLE EXCLUSIVE
            IF oEngine.HadError
                oEngine.HadError = .F.
                USE CS_TABLE SHARED
                IF oEngine.HadError
                    lcStatus = CS_STATOPENEX_LOC
                ELSE
                    lcStatus = CS_STATOPENSH_LOC
                ENDIF
            ELSE
                lcStatus = CS_STATNOOPEN_LOC
            ENDIF
            USE
        ELSE
            * 远程和本地视
            lcView = IIF(this.pageframe1.page1.opgCursorType.value = 2,
CS_LOCAL_VIEW, CS_REMOTE_VIEW)
            IF EMPTY(DBGETPROP(lcView, 'view', 'offlinepath'))
                lcStatus = CS_STATONLINE_LOC
                oEngine.OnlineStatus = lcStatus
            ELSE
                lcStatus = CS_STATOFFLINE_LOC
                oEngine.OnlineStatus = lcStatus
                USE (lcView) EXCLUSIVE
                IF oEngine.HadError
```

```
                    oEngine.HadError = .F.
                    USE (lcView) SHARED
                    IF oEngine.HadError
                        lcStatus = m.lcStatus + ", " + CS_STATOPENEX_LOC
                    ELSE
                        lcStatus = m.lcStatus + ", " + CS_STATOPENSH_LOC
                    ENDIF
                ELSE
                lcStatus = m.lcStatus + ", " + CS_STATNOOPEN_LOC
                ENDIF
                USE
            ENDIF
        ENDIF
        oEngine.SetErrorOff = .F.
        this.pageframe1.page1.lblStatus.caption = CS_STATUS_LOC  + m.lcStatus
    ENDPROC
    PROCEDURE offline
        PARAMETERS lcMode
        LOCAL lcView
        oEngine.SetErrorOff = .T.
        oEngine.HadError = .F.
        lcView = IIF(this.pageframe1.page1.opgCursorType.value = 2, CS_LOCAL_VIEW,
CS_REMOTE_VIEW)
        IF lcMode = "create"
            =createoffline(lcView)
        ELSE
            =dropoffline(lcView)
        ENDIF

        IF oEngine.HadError
            oEngine.HadError = .F.
        ENDIF
        oEngine.SetErrorOff = .F.
        this.RefreshStatus
    ENDPROC
    PROCEDURE QueryUnload
        CLEAR EVENTS
    ENDPROC
    PROCEDURE Destroy
```

```
        CLEAR EVENTS
    ENDPROC
    PROCEDURE Activate
        READ EVENTS
    ENDPROC
    PROCEDURE Load
        PUBLIC aErrArray
        DIMENSION aErrArray[5]
        IF TYPE("m.oEngine") # "O" OR ISNULL(m.oEngine)
            RETURN .F.
        ENDIF
        *设置缓冲中的多重锁定
        SET MULTILOCK ON
    ENDPROC
    PROCEDURE pageframe1.Page1.Activate
        * 刷新表、视信息
        thisform.RefreshStatus
    ENDPROC
    PROCEDURE pageframe1.Page2.Activate
        PUBLIC lRowBuffering, lForce, lTableBuffering, nUpdateType, aErrors, lnPrevPage
        DIMENSION aErrors[1]
        this.edtFunctionsInfo.value = CS_FUNCTIONS_INFO_LOC
        *设置打开的游标名字
        DO CASE
            CASE oEngine.CursorType = 1
                * 本地表
                this.lblcursorActive.caption = CS_STATTBL_LOC  + oEngine.CursorAlias
            CASE oEngine.CursorType = 2
                *本地视
                this.lblcursorActive.caption = CS_STATLV_LOC  + oEngine.CursorAlias
+ " " + oEngine.OnlineStatus
            CASE oEngine.CursorType = 3
                *远程视
                this.lblcursorActive.caption = CS_STATRV_LOC  + oEngine.CursorAlias
+ " " + oEngine.OnlineStatus
        ENDCASE
        * 在冲突后不用初始化
        lnPrevPage = oEngine.CurrentPage
        oEngine.Currentpage = 2
```

```
        IF lnPrevPage = 3
            RETURN
        ENDIF
        this.cmdRequery.enabled = !(oEngine.CursorType = TABLE_CURSOR)
        * 初始化缓冲
        this.opgBuffer.value = CURSORGETPROP('Buffering')
        IF Oengine.CursorType = TABLE_CURSOR
            this.opgBuffer.option1.enabled = .T.
            this.opgBuffer.option2.enabled = .T.
            this.opgBuffer.option4.enabled = .T.
            this.cmdUpdate.enabled = !(this.opgBuffer.value = 1)
            this.cmdRevert.enabled = !(this.opgBuffer.value = 1)
            this.cmdRequery.enabled = !(oEngine.CursorType = TABLE_CURSOR)
        ELSE
            this.opgBuffer.option1.enabled = .F.
            this.opgBuffer.option2.enabled = .F.
            this.opgBuffer.option4.enabled = .F.
        ENDIF
        this.opgUpdate.value = 1      && 行更新
        this.chkForce.value = .F.     && 乐观冲突锁定
        this.chkConflicts.value = .T. && 检测冲突设置为开
        *本地事务检测
        this.chkRules.value=      !(TYPE('oEngine.oServer')      <>      'O'      OR
ISNULL(oEngine.oServer))
        this.opgBuffer.interactivechange
        this.grid1.RecordSource = oEngine.CursorAlias
        this.grid1.refresh
    ENDPROC
    PROCEDURE cmdopen.Click
        LOCAL lcExclusive
        lcExclusive = IIF(this.parent.chkExcl.value = 1, 'EXCLUSIVE', 'SHARED')
        *重新设置网格
        This.Parent.Parent.Page2.Grid1.ColumnCount = -1
        DO CASE
            CASE thisform.pageframe1.page1.opgCursorType.Value = 1
                * 本地表
                SELECT 0
                USE CS_TABLE &lcExclusive
                oEngine.CursorAlias = TRIM(ALIAS())
```

```
            oEngine.CursorType = TABLE_CURSOR
        CASE thisform.pageframe1.page1.opgCursorType.Value = 2
            *本地视
            SELECT 0
            USE CS_LOCAL_VIEW   &lcExclusive
            oEngine.CursorAlias = TRIM(ALIAS())
            oEngine.CursorType = LOCAL_VIEW_CURSOR
        CASE thisform.pageframe1.page1.opgCursorType.Value = 3
            *远程视
            SELECT 0
            USE CS_REMOTE_VIEW   &lcExclusive
            oEngine.CursorAlias = TRIM(ALIAS())
            oEngine.CursorType = REMOTE_VIEW_CURSOR
    ENDCASE
    IF EMPTY(ALIAS())
        * 打开数据源失败
        RETURN
    ENDIF
    thisform.pageframe1.ActivePage = 2
ENDPROC
PROCEDURE cmdupsize.Click
    LOCAL lnConnHandle, lcSQL, lnRetVal, llRetVal, lcErrMsg, lnServerError, lcMsg,
aErrArray
    DIMENSION aErrArray[6]
    *连接 SQL 服务器
    ConnHandle = SQLCONNECT(CS_CONNECTION)
    DO CASE
        CASE ConnHandle = -1
            * 连接失败
            lcMsg = message()
            OEngine.Alert(lcMsg, MB_ICONEXCLAMATION + MB_OK)
            RETURN .F.
        CASE ConnHandle = -2
            lcMsg = STRTRAN(SQLCONN_FAIL_LOC, "!1", lcDataSourceName)
            OEngine.Alert(lcMsg, MB_ICONEXCLAMATION + MB_OK)
            RETURN .F.
    ENDCASE
    lcSQL = "DROP TABLE " + CS_TABLE
    lnRetVal = SQLEXEC(ConnHandle, lcSQL)
```

```
=AERROR(aErrArray)
IF lnRetVal <>1
    lnServerError = aErrArray[5]
    lcErrMsg = aErrArray[2]
    IF lnServerError <> 3701
        OEngine.Alert(lcErrMsg, MB_ICONEXCLAMATION + MB_OK)
        RETURN .F.
    ENDIF
ENDIF
* 创建远程表
lcSQL = "CREATE TABLE " + CS_TABLE + " (emp_id char(6) NULL, last_name
char(20)NULL, first_name char(10)"
lcSQL = lcSQL + " NULL, title char(30) NULL, salary int NULL, birth_date datetime
NULL, hire_date datetime NULL"
lcSQL = lcSQL + ", address char(60) NULL, city char(15) NULL, region char(15)
NULL, postalcode "
lcSQL = lcSQL + "char(10) NULL, country char(15) NULL, home_phone char(24)
NULL, extension char(4)"
lcSQL = lcSQL + " NULL, photo image NULL, notes text NULL, reports_to char(6)
NULL)"
lnRetVal = SQLEXEC(ConnHandle, lcSQL)
=SQLDISCONN(ConnHandle)
IF lnRetVal <> 1
    DO CASE
        * 服务器错误产生
        CASE lnRetVal = -1
            =AERROR(aErrArray)
            lnServerError = aErrArray[1]
            lcErrMsg = aErrArray[2]
            IF lnServerError = 1526 AND !ISNULL(aErrArray[5])THEN
            lnServerError = aErrArray[5]
            ENDIF
        * 连接错误产生
        CASE lnRetVal = -2
cErrMsg=STRTRAN(CONNECT_FAILURE_LOC,"l1",LTRIM(STR(lnServerErr)))
            OEngine.Alert(lcErrMsg, MB_ICONEXCLAMATION + MB_OK)
    ENDCASE
    RETURN .F.
ENDIF
```

```
    * 打开雇员表
    SELECT 0
    USE CS_TABLE SHARED
    lcAlias = TRIM(ALIAS())
    IF lcAlias = ''
        OEngine.Alert(ERR_NOOPENEMP_LOC,MB_ICONEXCLAMATION+ B_OK)
        RETURN .F.
    ENDIF

    * 打开本地视
    SELECT 0
    USE CS_REMOTE_VIEW
    =CURSORSETPROP('Buffering', 5)
    * 使用远程视相雇员表中添加信息
    APPEND FROM &lcAlias
    llResult = TABLEUPDATE(.T.)
    IF !llResult
        OEngine.Alert(CONNECT_FAILURE_LOC, MB_ICONEXCLAMATION)
        TABLEREVERT(.T.)
    ELSE
        wait window WAIT_UPSIZEOK_LOC timeout CS_TIMEOUT
    ENDIF
    * 清除
    USE     && 视
    SELECT
    USE     && 表
ENDPROC
PROCEDURE cmdreset.Click
    LOCAL lcSave
    * 检测 CS_INITIALTABLE
    * 检测写权限
    SELECT 0
    USE CS_INITIALTABLE
    lcSave = SET('SAFETY')
    SET SAFETY OFF
    REMOVE TABLE CS_TABLE DELETE
    COPY TO CS_TABLE
    USE
    ADD TABLE CS_TABLE
```

```
        SET SAFETY &lcSave
    ENDPROC
    PROCEDURE cmdcreateview.Click
        thisform.offline("create")
    ENDPROC
    PROCEDURE cmddropview.Click
        thisform.offline("drop")
    ENDPROC
    PROCEDURE cmdclose.Click
        IF oEngine.ServerIsStarted()
            oEngine.ServerStop
        ENDIF
        CLEAR EVENTS
        RELEASE THISFORM
    ENDPROC
    PROCEDURE cmddatabase.Click
        oStartForm = createobj('OpenDBC')
        IF TYPE('oStartForm')#'O' OR ISNULL(m.oStartForm)
            RETURN
        ENDIF
        oStartForm.Show
        THISFORM.RefreshStatus
    ENDPROC
    PROCEDURE opgcursortype.Click
        thisform.RefreshStatus
    ENDPROC
    PROCEDURE cmdhelp.Click
        HELP ID HELP_SAMPLE
    ENDPROC
    PROCEDURE grid1.AfterRowColChange
        LPARAMETERS nColIndex
    ENDPROC
    PROCEDURE grid1.Error
        LPARAMETERS nError, cMethod, nLine
        oEngine.Alert(message(), MB_ICONEXCLAMATION + MB_OK)
    ENDPROC
    PROCEDURE cmdinsert.Click
        append blank
        this.parent.grid1.refresh
```

```
ENDPROC
PROCEDURE cmddelete.Click
    delete
    this.parent.grid1.refresh
ENDPROC
PROCEDURE cmdrequery.Click
    LOCAL m.nResult
    m.nResult = -2          && 检测错误事件
    m.nResult = REQUERY()
    DO CASE
    CASE m.nResult == 1
        wait window WAIT_REQUERYOK_LOC timeout CS_TIMEOUT
    CASE m.nResult == 0
        wait window WAIT_REQUERYFAIL_LOC timeout CS_TIMEOUT
    ENDCASE
ENDPROC
PROCEDURE cmdclose.Click
    * 恢复修改的行
    IF USED(oEngine.CursorAlias)
        SELECT (oEngine.CursorAlias)
        IF CURSORGETPROP('Buffering') > 1
            =TABLEREVERT(.T.)
        ENDIF
        USE
    ENDIF
    IF oEngine.CursorType = LOCAL_VIEW_CURSOR
            IF USED(CS_TABLE)
            SELECT CS_TABLE
            USE
        ENDIF
    ENDIF
    * 清除
    oEngine.CursorAlias = ""
    oEngine.CursorType = NO_CURSOR
    thisform.lockscreen = .T.
    thisform.pageframe1.ActivePage = 1
    thisform.lockscreen = .F.
ENDPROC
PROCEDURE opgbuffer.InteractiveChange
```

```
        * 游标应当在当前工作区中打开
        oEngine.HadError = .F.
        oEngine.SetErrorOff = .T.
        llResult = CURSORSETPROP('Buffering', this.value)
        IF oEngine.HadError OR !llResult THEN
            this.value = CURSORGETPROP('Buffering')
            oEngine.SetErrorOff = .T.
            =MESSAGEBOX(CS_ERR_SET_BUFFERING,        ICON_EXCLAMATION,
CS_TITLE_TEXT)
        ENDIF
        this.parent.cmdUpdate.enabled = !(this.value = 1)
        this.parent.cmdRevert.enabled = !(this.value = 1)
        this.parent.opgUpdate.enabled = !(this.value = 1)
        this.parent.opgUpdate.option1.enabled = !(this.value = 1)
        this.parent.opgUpdate.option2.enabled = !(this.value = 1)
        this.parent.opgUpdate.option3.enabled = !(this.value = 1)
        this.parent.chkForce.enabled = !(this.value = 1)
        this.parent.chkConflicts.enabled = !(this.value = 1)
        this.parent.chkRules.enabled = !(this.value = 1)
        oEngine.SetErrorOff = .F.
    ENDPROC
    PROCEDURE cmdupdate.Click
        LOCAL lnUpdateType, llForce, llUpdate, lnRow, lcEmployee, frmConflicts

        lnUpdateType = this.parent.opgUpdate.value - 1
        llForce = this.parent.chkForce.value
        * 检测事务
        * 行更新
        IF oEngine.ServerIsStarted() AND lnUpdateType = 0
            IF !oEngine.ServerValidateRow()
                RETURN
            ENDIF
        ELSE
            *表更新
            IF oEngine.ServerIsStarted() AND lnUpdateType <> 0
                lnRow = 0
                lnRow = GetNextModified(lnRow)
                DO WHILE lnRow<>0
                    GO lnRow
```

```
                IF !oEngine.ServerValidateRow()
                    RETURN
                ENDIF
                lnRow = GetNextModified(lnRow)
            ENDDO
        ENDIF
    ENDIF
    oEngine.HadError = .F.
    oEngine.SetErrorOff = .T.
    llUpdate = TABLEUPDATE(lnUpdateType, llForce)
    IF oEngine.HadError
        =MESSAGEBOX(Oengine.cMessage,ICON_EXCLAMATION,
CS_TITLE_TEXT)
        llUpdate = .F.
    ENDIF
    oEngine.SetErrorOff = .F.
    IF llUpdate
        * 无冲突
        wait window WAIT_UPDATEOK_LOC timeout CS_TIMEOUT
    ELSE
        * 更新冲突，处理冲突
        IF this.parent.chkConflicts.value
            frmConflicts = createobject('Conflicts')
            frmConflicts.show
        ELSE
            wait window WAIT_UPDATEFAIL2_LOC timeout CS_TIMEOUT
        ENDIF
    ENDIF
ENDPROC
PROCEDURE cmdrevert.Click
    LOCAL llRevertType, lnRows
    llRevertType = (this.parent.opgUpdate.value <> 1)
    oEngine.HadError = .F.
    oEngine.SetErrorOff = .T.
    lnRows = TABLEREVERT(llRevertType)
    IF oEngine.HadError
            GEBOX(Oengine.cMessage,ICON_EXCLAMATION,CS_TITLE_TEXT)
        lnRows = 0
    ENDIF
```

```
        oEngine.SetErrorOff = .F.
        * 无冲突
        wait window ALLTRIM(STR(lnRows)) + WAIT_REVERT2_LOC timeout
CS_TIMEOUT
    ENDPROC
    PROCEDURE opgupdate.InteractiveChange
        *测试在行更新和表更新中的单行冲突
        * 在表更新中测试所有的表冲突
        oEngine.RowConflict = (this.value <> 3)
    ENDPROC
    PROCEDURE chkforce.Click
        lForce = this.value
    ENDPROC
    PROCEDURE chkrules.Click
        IF !oEngine.ServerIsStarted()
            oEngine.ServerStart
        ELSE
            oEngine.ServerStop
        ENDIF
        this.value = IIF(oEngine.ServerIsStarted(), 1, 0)
    ENDPROC
    PROCEDURE cmdhelp.Click
        HELP ID HELP_SAMPLE
    ENDPROC
```

9.3　使用升迁向导

在完成了客户／服务器的设计工作之后，我们就可以开始构造、并升迁一个本地数据库原型。本地数据库原型是应用程序的一种工作模型，它使用 Visual FoxPro 的表、视图和数据库来表示以后将被放在远程服务器上的数据。可以借助“升迁向导”将这些数据库、表和视图从本地系统迁移到一个远程 SQL Server 或 Oracle Server 上。

Visual FoxPro 为用户提供了两个升迁向导：“SQL Server 升迁向导”和“Oracle 升迁向导”。这两个升迁向导可以为用户创建一个 SQL Server 数据库或 Oracle Server 数据库，实现 Visual FoxPro 数据库中各表的功能。还可以重定向 Visual FoxPro 视图，使其可以使用新建的远程数据而不是本地数据。利用“升迁向导”可以实现以下功能：

（1）将本地数据转移到远程服务器上。

（2）将本地基表和本地视图转换为远程基表和远程视图。

（3）将本地应用程序移植为客户／服务器应用程序。

以下将介绍 Oracle 数据库升迁向导的使用，SQL Server 升迁和 Oracle 升迁几乎一样。

注意：尽管“升迁向导”只访问 SQL Server 或 Oracle Server，但是实际上可以为任何远程 ODBC 数据源创建客户 / 服务器应用程序。对于非 SQL Server 和非 Oracle Server，可以使用 SQL pass-through 函数创建远程表，然后使用 Visual FoxPro 来创建远程视图访问这些服务器上的表。

9.3.1 Oracle 升迁向导

使用 Oracle 升迁向导建立 Oracle 服务器数据库。可以最大限度地重现 Visual FoxPro 数据库的功能。并且 Oracle 升迁向导可以做到：

（1）将本地数据移到远程服务器上。

（2）将本地数据库和本地视图转换成远程数据库和远程视图。

（3）将本地应用程序移到客户 / 服务机应用程序上。

Oracle 升迁向导共有 10 个步骤：

1. 选择本地数据库

在该步骤中将选择要升迁的本地数据库。如果没有打开数据库，选择“打开”按钮来选择并且打开一个数据库。

2. 选择数据源

该步中，可以通过指定选项来选择 Oracle ODBC 数据源或需要使用的命名连接。

注意：在使用 ODBC 数据源连接时，必须通过 Windows 系统中的 Control Panel（控制面板）来设置 ODBC 的数据源连接。如下步骤：（以下已 SQL Server 为例）

1) 打开“开始”中的“控制面板”，使用“32 bit ODBC”选项，如图 9.2 所示：

2) 双击该图标，将出现 ODBC 数据源管理器窗体，如图 9.3 所示。

3) 用户通过“添加”按钮来创建新的数据源，并选用 ODBC Driver for Oracle 或 SQL Sever 选项。如图 9.4 所示。

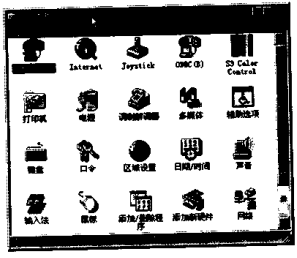


图 9.2 通过控制面板设置 ODBC 数据源连接

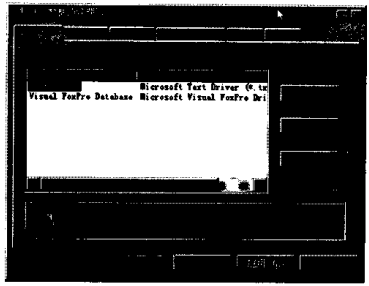


图 9.3　ODBC 数据源管理器

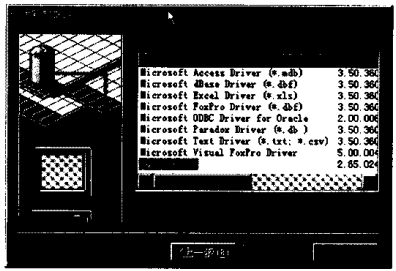


图 9.4　创建 SQL Server 数据源

4) 单击“完成”，来设计数据源和连接。如图 9.5 所示。

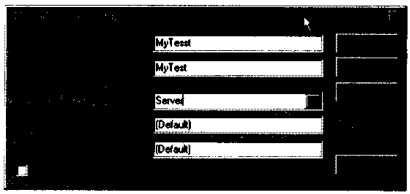


图 9.5　SQL Server 设置

5) 单击“ OK ”，设置好数据源 MyTest，如图 9.6 所示。

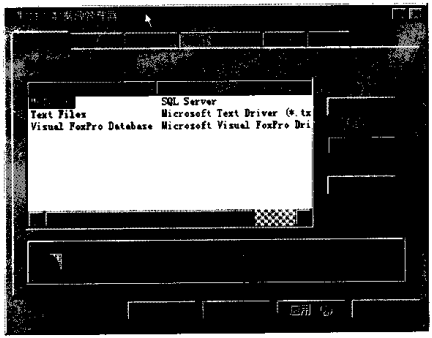


图 9.6 完成“ MyTest ”数据源的创建

3. 选择表

选择需要升迁的该数据库的表。

4. 匹配字段数据类型

Visual FoxPro 显示默认的数据类型映射，将本地数据升迁为远程数据，也就是将本地数据库的表升迁到服务器成为远程数据表。

（1）表。

从所有被选择为升迁的表中，选择需要映射字段的表。

（2）字段名称。

选择需要映射字段名。

（3） FoxPro 类型。

选择字段的 Visual FoxPro 数据类型。

（4）服务器类型。

选择服务器的数据类型。服务器的数据类型是服务器上字段的数据类型，由向导自动映射。

（5）宽度。

指定字段的宽度。

（6）精度。

需要时，可指定字段中小数点位置。

5. 选择表空间

表空间是 Oracle 服务器用来存放远程数据库及其索引。可以选择 Oracle 服务器上现有的表空间，或者创建新的表空间。

（1） FoxPro 表大小。

显示需要升迁表所占的空间大小，以 KB 为单位。

（2）表空间大小。

显示 Oracle 服务器上表空间的剩余空间大小，以 KB 为单位。

（3） FoxPro 索引大小。

显示需要升迁索引所占的空间大小，以 KB 为单位。

（4）索引的表空间大小。

显示 Oracle 服务器上表空间可用于索引的剩余空间大小，以 KB 为单位。

（5）现有的表空间。

显示已有的表空间的列表。

（6）新建的表空间。

创建新的表空间。

6. 选择表空间

扩展在步骤 5 选择的表空间的容量。

（1）表空间。

选择想要扩展的 Oracle 服务器上表空间。

（2）新的数据文件。

输入数据文件的名称。

（3）现有数据文件。

显示与表空间相关的现有文件名，可以避免在指定新数据文件名称时与现有文件名相重。显示的现有文件名不包含新的数据文件。

（4）表空间上的剩余空间。

显示已命名的表空间的剩余空间。

（5） FoxPro 数据的大小。

显示选择为升迁到指定 Oracle 服务器上表空间中的本地 FoxPro 表和索引所占的空间，以 KB 为单位。

（6）数据文件的大小。

该数值代表所分配给的新数据文件的大小，以 KB 为单位。

7. 指定磁盘簇

该步骤用于创建了簇表。如果选择了创建 Oracle 簇表，升迁向导会很容易的完成步骤 7、8 和 9。如果选择不创建簇表，升迁向导自动跳到步骤 10。本步骤中，可以添加新簇，编辑簇的名称，指定簇类型，指定索引名或者 hash 关键字，也可以指定簇行的大小。

（1）簇名称。

显示选中簇的名称。创建一个新簇时，在该文本框中输入新簇的名称。

（2）默认。

显示默认簇列表。默认簇列表基于已打开的数据库中的父一子关系列表。簇名基于父表名称。每个簇包括父表的相关子表。在默认的簇列表中，每个父表对应于一个特殊的簇，这个父表不是以前关系中的子表。

（3）索引。

指定簇的索引。创建了簇之后，必须基于簇关键字创建索引。

（4）Hash。

指定一个 hash 簇。

（5）Hash 关键字。

为 hash 簇指定 hash 关键字。

（6）大小。

在相同的簇关键字或相同的 hash 关键字中指定存储所有记录所需的空间，用 KB 字节表示。如果省略了这个值。Oracle 7 为每一个簇关键字或 hash 关键字保留一个数据块。

（7）添加。

向簇列表中添加簇信息。

（8）移去。

从簇列表中移去选定的簇。

8. 指定簇表

在本步骤中，可以指定每个簇所包含的表。每个簇应该至少包含一个表。如果某个簇中没有表，则必需返回到步骤 7，移去该簇。

（1）簇。

显示簇名称。要选择一个不同的簇。

（2）可用的表。

显示一列表，表中不显示已经属于其他簇的表，在这个表中选择该簇包含的表。

（3）已选中的表。

显示该簇已经选中的表。

（4）验证。

测试被选中的表，确保公共列的数据类型和大小都是相对应的。

9. 指定簇关键字

簇列必须与簇表中每列的数据类型和大小相对应；但数值不需要对应。

（1）簇。

选择一个不同的簇和可扩展列表，并选择一个簇名。

（2）表。

选择簇所包含的表，第一个表应该是簇的主表。

（3）可用的字段。

显示一个所指定的簇和表的字段列表，簇关键字可以包含列表中的任何字段。

（4）选中的字段。

显示已经选中的包含在簇关键字中的字段列表。这个列表中自动包含主关键字，可以移去它们，或者从可用字段列表中选择包含其他字段。主表中被选中的列成为簇关键字的样本；列的名称成为簇中列的名称。

（5）验证。

测试被选中的字段，确保它们的数据类型和大小都是相对应的。

10. 设置升迁向导

指定“Oracle 升迁向导”对本地数据库所需要进行的修改。可以创建升迁报表，重新定向视图到远程数据上，基于升迁报表创建新的远程视图，并保存视图的密码。

（1）升迁表的属性。

索引　升迁 Visual FoxPro 的*.CDX 索引。

默认　升迁表字段的默认值。

关系　升迁保存在升迁过程数据库中的关系。

使用声明的参照完整性　建立关系，以加强参照完整性。

有效性规则　升迁字段和表的有效性规则。

只升迁结构，不带有数据　升迁空的表结构，不将表中的数据复制到 Oracle 数据源。

（2）本地修改。

创建升迁报表　创建一系列报表用来记录升迁过程的结果。

重新定向视图到远程数据　将升迁数据库中的本地视图的定义更改为远程视图，这样查询、表单和报表会使用 Oracle 新数据源中的数据，而不是使用原来的 Visual FoxPro 数据。

在表上创建远程视图　当升迁一个本地表时，需要一个远程视图用于访问远程服务器上已升迁的表。在升迁时，“Oracle 升迁向导”会创建新的远程视图；“Oracle 升迁向导”创建新的远程视图时，重新命名本地表的版本，在表名中加一个后缀“_local”。

与视图一起保存口令　默认情况下，在 Visual FoxPro 的以后打开一个远程视图时，必须输入密码和注册的 ID。可以在本地的数据库视图定义中保存密码。

（3）创建升迁报表。

Visual FoxPro 会生成升迁报表，该报表记录了“Oracle 升迁向导”在 Oracle 上创建的表、视图、字段、索引和参照完整性约束的信息。“Oracle 升迁向导”将使用下列报表名称将报表放在新项目中：RptErrors、RptField、RptIndex、RptRels、RptTable、RptViews。

升迁报表包含有关创建的设备和数据库的信息，还包含在升迁过程中遇到的错误，以及对每个 Visual FoxPro 对象映射为 Oracle 对象方式的完整解释。在升迁之后，可以查看或打印这些报表。

11. 完成

在该步骤中可以有三个选择：

数据库升迁，但不产生 SQL 代码。

升迁数据库，并存储所需的 SQL 代码。

[General Information]
书名=Visual FoxPro 5.0中文版编程宝典
作者=周予滨
页数=383
SS号=10011761
出版日期=1998年6月第1版