第 5 章　Visual FoxPro 程序设计基础

Visual FoxPro 6.0 数据库管理系统既支持结构化程序设计，又支持面向对象的程序设计。结构化程序设计采用模块化程序设计思想，程序由顺序结构、分支结构和循环结构三种基本控制结构组成。程序执行时从主程序开始，按照各模块间的相互调用顺序和控制结构规定的顺序依次执行。

本章讲解结构化程序设计的基本知识，其中包括程序及程序文件的建立、常用的程序命令、基本控制结构、子程序以及程序调试 5 个方面的内容。通过本章的学习，读者应该了解及掌握以下内容。

- 了解程序、程序文件的概念。
- 掌握程序中输入输出语句等常用命令。
- 掌握结构化程序设计中三种基本控制结构。
- 掌握自定义过程、函数的创建和使用。
- 掌握内存变量的作用域。
- 掌握程序调试的基本方法。

5.1　程序及程序文件

Visual FoxPro 数据库管理系统提供了交互式执行和程序执行两种工作方式。交互式方式包括前面介绍的命令操作法和菜单操作法，这种方式适合初学者，或者是完成简单、不需要反复执行的某些操作的用户，效率低下。但在实际应用中，往往有大量的操作需要反复执行，此时就需要使用程序执行方式。

5.1.1　基本概念

使用 Visual FoxPro 6.0 解决实际问题，就是用 Visual FoxPro 6.0 提供的命令来管理和处理数据，完成一些具体的操作。但许多任务是单条命令的一次执行无法完成的，通常需要连续执行一组命令才能达到预期的效果。如果采用命令方式，在命令窗口逐条数据执行，或采用菜单操作方式，不仅操作步骤繁杂，容易出错，并且有些操作是交互方式无法完成的，比如在条件满足的情况下，多次重复执行某条命令，此时就需要使用程序执行方式来完成。

1. 程序

程序是指能够完成一定任务的一组命令的有序集合。这些命令被存储在一个文件中，当需要完成该任务时，运行该文件，系统会按照程序规定的次序自动执行程序中包含的命令序列。

例 5-1 设计一个程序，首先将“学生”表复制到“学生_bak”中，然后删除“学生_bak”表中所有男同学，并显示结果。

根据题目要完成的任务，可设计程序如下：

```
SET TALK OFF                          && 关闭对话,不显示命令结果
CLEAR                                 && 清屏
USE 学生                              && 打开数据表"学生"
COPY TO 学生_bak                      && 将"学生"表复制到"学生_bak"中
USE 学生_bak
DELE ALL FOR 性别="男"                && 逻辑删除所有男同学
PACK                                  && 物理删除
LIST                                  && 显示结果
USE                                   && 关闭表
SET TALK ON                           && 打开会话
```

说明：

① 命令注释语句。程序中可以插入命令注释语句，以提高程序的可读性。以 NOTE 或 * 开头的代码为注释行。命令行后，可以使用 && 作为该行命令的注释。注释并非程序代码，Visual FoxPro 6.0 并不会执行这些注释语句。

② SET TALK ON|OFF 命令。在 Visual FoxPro 6.0 中很多命令在执行时都会返回有关执行状态的信息，这些信息通常显示在主窗口、状态栏或用户自定义窗口内。在 Visual FoxPro 6.0 中可以使用 SET TALK OFF 命令关闭这些交互信息的显示，使用 SET TALK ON 开启。

③ 命令分行。如果程序中有一条命令行过程，就需要转行书写，此时就需要使用续行符号“;”。

与交互执行方式相比，程序执行方式具有以下 4 个特点：

① 程序可被修改并重新运行；

② 程序可在菜单、表单和工具栏下启动；

③ 一个程序可以调用其他程序；

④ 程序一旦建立，可以多次重复执行。

2. 程序文件

在 Visual FoxPro 6.0 中程序是被存放在一个扩展名为.PRG 的文件中的，称为程序文件。程序文件是一个文本文件，其中包含了程序中命令代码，可以通过菜单操作或命令操作方式运行。

5.1.2 程序文件的建立、编辑和运行

在 Visual FoxPro 6.0 中可以使用命令方式、菜单操作方式和项目管理器方式创建和

编辑程序文件。

1. 程序文件的建立和编辑

1）命令方式

命令方式下可以使用 MODIFY COMMAND 命令创建程序文件。

格式：

```
MODIFY COMMAND <prg_file_name>
```

功能：创建或编辑由 prg_file_name 指定的程序文件。

说明：

① 该命令打开一个程序编辑窗口，用于建立、编辑和修改指定的程序文件。

② ＜prg_file_name＞用于指定程序文件的名称，其中可以包括完整的盘符和路径名。若缺省路径，则默认为当前路径。若＜prg_file_name＞缺省程序文件的扩展名，则在存盘时，系统自动添加.PRG 扩展名。

③ 若＜prg_file_name＞缺省时，系统自动为创建的程序文件以此命名为“程序 1”、“程序 2”等。

④ 程序文件编辑完成后，可按下 Ctrl＋W 组合键存盘退出，若需放弃本次编辑的程序文件可以按下 Ctrl＋Q 组合键。

2）菜单操作方式

使用菜单方式打开程序文件编辑窗口，创建程序文件的步骤如下：

① 在 Visual FoxPro 6.0 系统主菜单下，选择“文件”→“新建”命令，或者直接单击常用工具栏上的“新建”按钮，打开“新建”对话框。

② 在“新建”对话框中，单击“文件类型”项目组中的“程序”单选按钮，接着单击“新建文件”按钮，即可打开默认名称为“程序 1”的程序文件编辑框，如图 5-1 所示。

③ 在程序文件编辑窗口中输入程序代码后，按下组合键 Ctrl＋W，或选择菜单项“文件”→“保存”（也可以是“另存为”），或者单击常用工具栏上的“保存”按钮，弹出“另存为”对话框，在其中输入文件名后，单击“保存”按钮，即可为新创建的程序文件命名并保存。

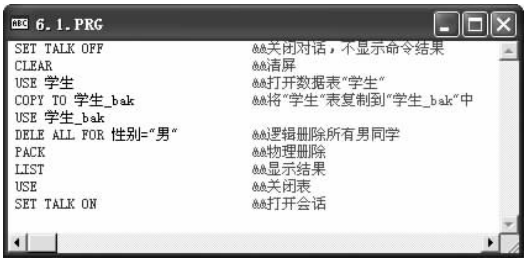

图 5-1　“程序文件”编辑窗口

3）项目管理器方式

使用项目管理器方式创建程序的步骤如下：

① 在 Visual FoxPro 6.0 系统主菜单下，选择“文件”→“新建”命令，打开“新建”对话框。单击“文件类型”项目组中的“项目”单选按钮，接着单击“新建文件”按钮，打开“创建”对话框，输入项目文件名后，单击“保存”按钮，即可打开“项目管理器”对话框。

② 在“项目管理器”对话框中，切换至“全部”选项卡，展开“代码”项目，选择“程序”选项后，单击“新建”按钮，如图 5-2 所示，即可打开程序文件编辑窗口。

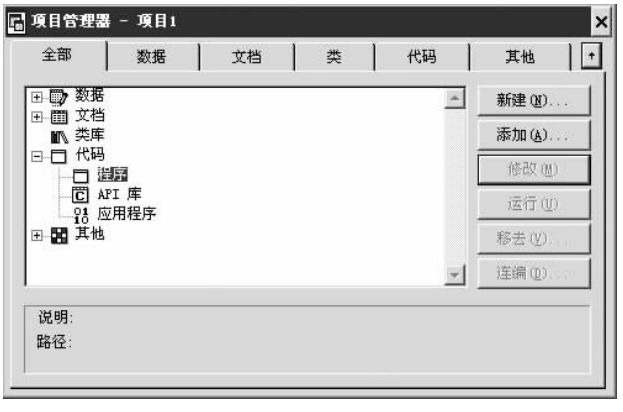

图 5-2　使用“项目管理器”创建程序文件

2. 程序文件的运行

在 Visual FoxPro 6.0 中，可以使用命令方式、菜单操作方式和项目管理器方式来运行程序。

1）命令方式

格式：

```
DO <prg_file_name>
```

功能：运行由＜prg_file_name＞指定的程序文件。

说明：

① ＜prg_file_name＞用于指定被执行的程序文件的名称，其中可包含文件的路径，若省略路径，默认在当前目录下。

② ＜prg_file_name＞指定的程序文件名，可以省略文件的扩展名，此时系统默认为.PRG。

③ 该命令可以在命令窗口中执行，也可以在另一个程序中被调用。

2）菜单操作方式

在 Visual FoxPro 6.0 系统主菜单下，选择“程序”→“运行”命令，或单击常用工具栏上的“运行”按钮，打开“运行”对话框。在其中选择需要运行的程序文件后，单击“运行”按钮，如图 5-3 所示。

3）在项目管理器中执行命令

在“项目管理器”的“代码”选项卡中选中需要执行的程序文件后，单击“运行”按钮

即可。

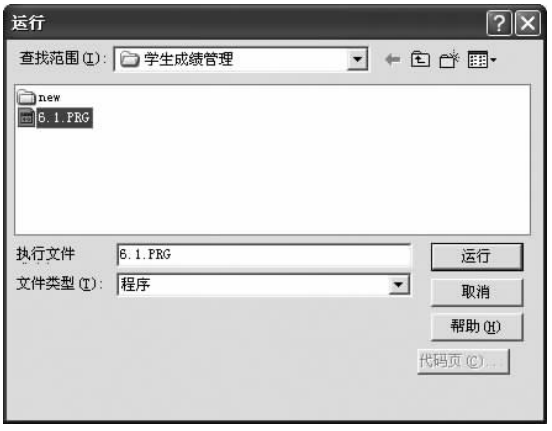

图 5-3　“运行”对话框

5.2　程序中常用的命令语句

在 Visual FoxPro 6.0 程序设计中，经常要给变量赋值，也需要从键盘中接受用户的输入，再将程序执行的结果输出给用户，这些就需要使用输入、输出语句。

5.2.1　赋值语句

在 Visual FoxPro 6.0 中，可以使用两种格式的赋值语句。

格式 1：

```
<var>=<expr>
```

功能：先计算表达式＜expr＞的值，然后赋予左边的变量＜var＞。

说明：

① 赋值语句中的赋值符号“＝”与等号形式相同，但含义完全不同。赋值语句的物理含义是将表达式的值计算出来后赋予变量所代表的存储单元，两者之间不存在相等的概念。例如 $X=X+1$，当“＝”代表赋值符号时，表示先将 X 的值取出后加 1，再赋予 X。若“＝”代表等号，则该表达式无意义。

② 格式 1 的赋值语句一次只能给一个变量赋值。

③ 在 Visual FoxPro 6.0 中，变量的类型随着赋值语句右边表达式值的类型的改变而改变。

例 5-2　演示变量类型的变化。

在命令窗口输入以下语句，查看显示结果：

```
X={^1999-01-01}
?TYPE(X)                              && 显示 D,表示 X 为日期型
```

```
X=DATE()-X
?TYPE(X)                                    && 显示 N,表示 X 为数值型
```

格式 2：

```
STORE <expr> TO <var_list>
```

功能：先计算表达式＜expr＞的值，再将值赋予变量列表＜var_list＞中所有的变量。

说明：

① ＜var_list＞中的多个变量使用逗号隔开。

② STORE 能一次赋值给多个变量。

5.2.2 常用的输入输出语句

1. 输入单字符命令

格式：

```
WAIT [<messages>][ TO <mem_var>] [ WINDOWS [AT <row_num>,<col_num>]] [NOWAIT]
[CLEAR| NOCLEAR][ TIMEOUT <expN>]
```

功能：暂停程序的执行，显示提示信息＜messages＞，等待接受用户从键盘输入的一个字符。

说明：

① ＜messages＞为提示信息，若缺省时，系统默认提示信息为“按任意键继续……”。

② WINDOWS [AT ＜row_num＞,＜col_num＞]用于指定提示信息在主窗口的 row_num 行，col_num 列显示。若缺省，则系统在当前行的最左列显示。

③ TO ＜mem_var＞子句将接收到的用户输入字符保存到内存变量＜mem_var＞中。用户输入时，无须使用字符定界符，否则，系统将把定界符的第一个字符作为输入。若用户直接按 Enter 键或单击鼠标，则内存变量将得到空值。若无 TO ＜mem_var＞子句，系统不保留接受的用户输入。

④ 若无 NOWAIT 子句，系统不等待用户按键，程序直接往下执行。

⑤ TIMEOUT ＜expN＞子句，用于设定系统等待用户输入的时间间隔，单位是秒，超过该间隔，程序不等待用户输入，自动向下执行，mem_var 也得不到输入。

⑥ CLEAR| NOCLEAR 子句，用于设定是否清除 WAIT 语句留在系统主窗口上的提示信息，CLEAR 表示清除，NOCLEAR 不清除。

例如：在命令窗口中输入以下命令。

```
WAIT "请选择是否继续执行(Y/N)" TO X WINDOWS AT 10,20 TIMEOUT 5 NOCLEAR
```

该命令在主窗口的 10 行 20 列处，弹出窗口，显示提示信息“请选择是否继续执行(Y/N)”，等待用户输入，如果 5 秒内输入一个字符，该字符将被存入内存变量 *X* 中，如果 5 秒内用户未输入字符，*X* 变量为空，且提示信息窗口不清除。

2. 输入字符串命令

格式：

```
ACCEPT [ <messages>] TO <mem_var>
```

功能：程序暂停执行，显示提示消息＜messages＞，等待接受用户从键盘输入字符串。

说明：

① ACCEPT 只接受字符串，因此用户从键盘输入时，无须使用字符串定界符，否则会将定界符一起作为用户输入保存到内存变量中。即使用户输入的是纯数字，也将作为字符串保存到内存变量中。

② ＜messages＞为提示信息，可以是字符串，也可以是字符串变量。

③ 如果用户不输入任何字符，直接按下 Enter 键，系统将空串赋予内存变量 mem_var。

例如：若程序中包含以下命令。

```
ACCEPT "请输入需要查找的学生的姓名:" TO sname
&& 显示提示信息"请输入需要查找的学生的姓名:",等待用户输入学生姓名
```

3. 输入任意数据命令

格式：

```
INPUT [<messages>] TO <mem_var>
```

功能：程序暂停执行，显示提示信息，等待用户从键盘输入数据。当用户以 Enter 键结束输入时，内存变量中得到相应的值，程序继续向下执行。

说明：

① ＜messages＞为提示信息。＜mem_var＞为接受输入数据的内存变量。

② 与 WAIT 和 ACCEPT 命令只能接受字符或字符串数据不同，INPUT 命令可以接受任意类型的数据，因此用户在输入数据的时候需要使用定界符。数值数据直接输入；字符数据使用" "、' '、或[]；逻辑性数据使用圆点，如. T. 、. F. 等，日期型和日期时间型数据使用严格格式输入，或使用转换函数，如{^1989-01-12}、CTOT("1989-01-12 04:16:18 a")。

③ 不能不输入数据而直接按 Enter 键。

4. 输出命令

格式 1：

```
?[<expr>]
```

格式 2：

```
??<expr>
```

功能："?"命令先计算表达式 expr 的值，然后换行显示表达式的值。"??"命令先计算表达式 expr 的值，接着在当前光标位置显示表达式的值。

说明：

① 输出命令"?"和"??"的不同在于"?"在输出表达式的值之前先要换行，而"??"不换行。

② 当“?”命令省略其后的表达式时，只打印一个空行。

5. 文本输出命令

格式：

```
TEXT
        <context_txt>
ENDTEXT
```

功能：将文本内容 context_txt 按照输入的格式原样输出。

例如：下面的程序段用于演示 TEXT 命令的用法。

```
SET TALK OFF
CLEAR
TEXT
    1.新建表
    2.打开表
    3.添加表
    4.关闭表
    5.删除表
ENDTEXT
WAIT "请选择需要的操作:" TO ch TIMEOUT 5
SET TALK ON
RETURN
```

6. 格式输出命令

在 Visual FoxPro 6.0 中，当需要在窗口指定的位置显示数据时，可以使用格式输出命令。

格式：

```
@<row,col>SAY <expr>
```

功能：在有 row 和 col 指定的行、列位置显示表达式 expr 的值。

说明：

① row 为数据显示的行位置，col 为列位置，窗口最左上角的行列坐标为<0,0>。

② SAY 后面只能跟一个显示项目，如果希望同时显示字符串常量和其他类型的变量，可以使用字符串连接符“+”或“-”。

例如：@ 10,20 SAY "当前时间为："+TIME()。

执行结果为在主窗口的 10 行 20 列处显示

```
当前的时间为:22:03:15
```

7. 格式输入输出命令

格式：

```
@<row,col>[ SAY <messages>] GET <var>
```

```
READ
```

功能：在 row 和 col 指定的行、列位置建立一个反像显示编辑区，对变量 var 现有的值进行显示，等待用户编辑，并将结果存放到该变量中。

说明：

① row 为数据显示的行位置，col 为列位置，窗口最左上角的行列坐标为＜0，0＞。

② SAY 子句用于显示提示信息 messages。

③ GET 子句中的变量 var 可以是已赋值的内存变量，也可以是字段变量，类型可以是任意类型。

④ READ 子句用于激活 GET 子句的编辑区，所以在 READ 子句前面的所有 GET 子句仅起显示变量值的作用，不能编辑。

⑤ 当 GET 子句后面的变量 var 是备注类型或通用性字段时，双击该变量，或将光标停到该变量上，按下 Ctrl＋Home、Ctrl＋PgUp、Ctrl＋PgDn 组合键才能编辑其内容。其他类型字段的内容可以在指定变量的显示位置单击鼠标或通过键盘直接在编辑域修改，按 Enter 键结束编辑，将编辑区的内容保存到相应的变量中。

例 5-3　编写程序 prg5-3.PRG，使用格式输入输出命令，修改“学生.DBF”中指定记录的内容。

```
*prg5-3.PRG
SET TALK OFF
CLEAR
USE 学生
INPUT "请输入记录号:" TO n
GO n
@3,10 SAY "学号" GET 学号
@3,30 SAY "姓名" GET 姓名
@4,10 SAY "性别" GET 性别
@4,30 SAY "出生日期" GET 出生日期
@5,10 SAY "简历" GET 简历
@5,30 SAY "照片" GET 照片
READ
USE
SET TALK ON
CANCLE
```

程序运行的结果如图 5-4 所示。

图 5-4　例 5-3 运行结果

5.2.3 其他命令

1. 清屏命令

格式：

```
CLEAR
```

功能：清除整个屏幕，光标回到屏幕的左上角。

2. 终止程序命令

格式：

```
CANCLE
```

功能：结束程序运行，返回命令窗口，同时释放所有私有变量，通常在主程序的最后使用该命令。

3. 设置会话状态命令

格式：

```
SET TALK ON|OFF
```

功能：ON 用户打开用户会话状态，OFF 用于关闭会话状态。在 Visual FoxPro 6.0 中，会话是指命令在执行时给用户反馈的命令执行后的状态信息。默认情况下会话状态是打开的，但在程序执行模式下，这种会话状态通常是不需要的，故关闭。

4. 注释语句

在 Visual FoxPro 6.0 中注释语句有三种格式。

格式 1：

```
* <comment_txt>
```

格式 2：

```
NOTE <comment_txt>
```

格式 3：

```
&& <comment_txt>
```

功能：格式 1 和格式 2 可以用于程序的任何位置，对程序块或整个程序进行说明。格式 3 用在某条命令之后，多用于对某行命令进行说明。

5.3 程序控制结构

结构化程序设计方法规定，一个程序只能包含三种基本控制结构。这三种基本控制结构分别是顺序结构、分支结构和循环结构。每一种结构都可以包含若干条命令，但只能