第一章　FoxPro程序设计基础

Foxpro 提供的编程功能是十分完善的。Foxpro 的命令格式类似于英语的日常用语。易读性好，它的每一基本指令又可派生出多条命令。整个命令系统提供了处理大型、复杂数据库系统的能力，利用这些命令你可以开发出大型的数据库管理信息系统。

1.1　FoxPro 命令的基本约定

FoxPro 语言由两部分组成：命令和函数。一条命令执行一个操作，而一个函数则返回一个值。例如：CREATE 命令用于建立一个数据库，函数 DATE()返回当前的系统日期供你使用。函数包含一对括号以便同命令区分开。

命令和函数可以组合在一起组成一条 FoxPro 语句。函数不能单独使用，它们必须同命令一起使用。

例如，问号（?）是一条把输出发送到屏幕的命令，函数 TIME() 执行返回系统时间的功能。它们结合起来就可以完成向屏幕输出系统时间的功能。

```
? TIME()
```

1.1.1　命令和函数

FoxPro 命令和函数由一个或多个成分组成。这些组成决定了如何使用一个命令或函数。

让我们看一个典型的 FoxPro 命令的语法。下面以 REPLACE 命令来举例说明，该命令用于对数据库记录的字段进行更新。表 1-1 列出了各组成部分的说明。

```
REPLACE <field> WITH <expr1> [ADDITIVE]
    [, <field2> WITH <expr2> [ADDITIVE]]...
    [<scope>] [FOR <expL1>] [WHILE <expL2>]
    [NOOPTIMIZE]
```

表 1-1　命名各组成部分的说明

组成部分类型	以 REPLACE 举例说明
关键字	REPLACE, WITH, ADDITIVE, FOR, WHILE, NOOPTIMIZE
表达式	expr1, expr2, expL1, expL2
名　称	field1, field2

每一个 FoxPro 的命令或函数都至少包含一个关键字。它们采用大写形式，FoxPro 采用关键字来标识一条命令或函数。这些特殊字符被 FoxPro 保留供内部使用，因此又被称为保留字。

在命令 REPLACE 中关键字有 REPLACE（FoxPro 用它来标识一条命令），WITH、ADDITIVE、FOR 和 WHILE。关键字 WITH、ADDITIVE、FOR、WHILE 被用于字句中。

关键字也可简写成前四个字符。

< > 角括号及所扩起来的小写字符表示这是需要由用户提供的信息。该信息可以是文件名、表达式和内存变量。

（ ）在所有的 FoxPro 函数中必须包含小括号。小括号以其自身的形式出现在函数的语法说明中。

[ ] 方括号及所扩起来的内容表示 FoxPro 命令或函数的任选部分。在实际输入过程中可不输入它们。例如，REPLACE 命令提供了大量的任选项，你可以选择其中的一项或多项，子句 ADDITIVE,<scope>,FOR<expL1>,WHILE<expr2> 和 NOOPTIMIZE 都被放在方括号中以表示它们是可选项。

| 一个竖杠符号用于分隔任选项，这些并置的任选项一次只能选择其中的一项。

...一个省略符号表明一条命令或函数的一部分可以以类似的格式继续。

1.1.2 表 达 式

一个 FoxPro 表达式由这些部分组成：数据库字段、函数、内存变量、数组元素、常量和操作符。FoxPro 系统的表达式分成四类：字符表达式、数字表达式、日期表达式、逻辑表达式。

在 FoxPro 命令和函数中，表达式以表 1-2 列出的方式给出。

表 1-2 命令和函数中表达式的方式

表达式	表达式类型
<expC>	字符表达式
<expN>	数字表达式
<expD>	日期表达式
<expL>	逻辑表达式
<expr>	字符、数字、日期或逻辑表达式
<expr list>	以逗号分割表达式

当命令或函数中相同类型的表达式出现一次以上时，则在表达式之后加上一个数字以表明它们在命令或函数中的位置。例如，函数 CHRTRAN() 的语法为：

CHRTRAN (<expC1>,<expC2>,<expC3>)

这表明 CHRTRAN() 函数需要三个字符表达式参数：<expC1>,<expC2>,<expC3>

由数据库字段、函数、内存变量和数组元素组合成的表达式各成份应当具有相同的数据类型。若它们的类型不一致，则 FoxPro 产生一条“操作符 / 操作类型不匹配”（Operator / operand type mismatch）的错误信息。

1.1.2.1 字符表达式

字符表达式组成如下：

- 字符类型的数据库字段
- 返回字符值的函数
- 包含字符数据的内存变量和数组元素
- 字符常量或称为字符串

一个字符串是由单引号或双引号扩起来的一串字符。例如："hello"或' hello'。引号必须匹配，也就是说，一单引号开始的字符串不能以双引号结束。

你可以在一种类型的引号中扩起来的文字串中嵌入另外一种引号。例如 " Don't touch!"是一个合法的字符串。表 1-3 列出了字符表达式的所有操作符。

表 1-3 字符表达式的操作符

操作符	操作
+	连接字符串，将两个字符串连接在一起
-	连接字符串，删去第一个字符串和第二个字符串的尾遂空格
$	字符表达式比较

空串可以使用一对二者之间没有空格的引号来表示如：""或''。如果 CHAR()函数的参数为'0'，即 CHAR(0),则它返回为空串。

注意：在所有的 FoxPro 资料中，将会经常提到空串。空串是一个长度为零的字符串，它不包含任何字符。

1.1.2.2 数字表达式

数字表达组成如下：

- 数字类型的数据库字段
- 返回数字值的函数
- 包含数字数据的内存变量和数组元素
- 数字常量

数字常量是由数字组成的变量。表 1-4 列出了数学运算的所有操作符。

表 1-4 数学运算操作符

操作符	操作
()	括号用于改变运算顺序
**|^	幂
*,/	乘、除
%	取模（余数）
+,-	加、减

1.1.2.3 日期表达式

日期表达式组成如下：

- 日期类型的数据库字段
- 返回日期值的函数

●包含日期数据的内存变量和数组元素

●日期常量

日期常量为数字：1、2、3、1981 等等。日期常量表示天数，对于一个给定的日期，你可以减去或加上一个日期常量（天数）。

可以用花括号扩起来以指定日期。如命令：

```
STORE {12/22/85} TO F_DAY
```

建立一个日期型的内存变量 F_DAY 来保存日期“12/22/85”。

1.1.2.4 逻辑表达式

逻辑表达式的值是下列二者之一“真”或“假”，在 FoxPro 系统中分别用“.T.”和“.F.”表示真和假。

逻辑表达式组成如下：

●逻辑类型的数据库字段

●返回逻辑值的函数

●包含逻辑数据的内存变量和数组元素

●由专门的关系操作符分割开的其它表达式（字符、数字、或日期）

下表分别列出了逻辑表达式的操作符和关系操作符。

表 1-5 列出了逻辑表达式的操作符（按优先顺序）。表 1-6 列出了逻辑表达式的关系操作符

表 1-5 逻辑表达式的操作符

操作符	操作
()	括号用于改变运算次序
!|NOT	逻辑非
AND	逻辑与
OR	逻辑或

表 1-6 逻辑表达式的关系操作符

操作符	操作
<	小于
>	大于
=	等于
<>|#|!=	不等于
<=	小于等于
>=	大于等于
==	字符串恒等比较（当 EXTRACT 为 ON 时尾部空格有意义）

1.1.2.5 记录范围：<scope>、FOR 和 WHILE

当命令对数据库中的记录进行操作时，你可以特别指定该命令所起作用的范围。使用

〈scope〉, FOR 和 WHILE 子句确定记录的范围。

范围：当包含一个范围时，命令在数据库中特定的记录范围内起作用。你可以使用下列子句确定记录的范围。

ALL

命令对数据库中的所有记录起作用。

NEXT 〈expN〉

命令在某范围内执行。该命令起始于当前记录并包含以后的 N 个记录，N 是数字表达式 expN 的值。例如：NEXT 1 在当前的记录上操作。

RECORD 〈expN〉

命令对数据库中指定的第〈n〉个记录进行操作，N 是数字表达式 expN 的值。

REST

命令在某范围内执行，该命令起始于当前记录并持续到文件的最后一个记录。

FOR 〈expL〉

一个命令的作用范围也可以使用 FOR 子句和 WHILE 子句来确定。当使用子句时，命令对每一个满足指定逻辑条件的记录产生影响。

WHILE 〈expL〉

从另一角度看，WHILE 表达式命令仅在逻辑表达式为真时，才对每一数据库记录进行处理。如果表达式的值为假，则不管后面的记录如何，都中止命令的执行。这一表达式经常与数据库字段一起使用，并根据 WHILE 表达式中使用的一个或多个字段对数据库进行排序和索引。

范围、FOR 表达式和 WHILE 表达式均可在同一 FoxPro 命令中使用。如果在命令中同时指定了 FOR 和 WHILE，则 WHILE 表达式具有较高的表达功能。

1.2 结构化程序设计

FoxPro 提供了先进的结构化程序设计功能。整个命令语言采用了类似于 PASCAL 语言的方式。体现了结构化程序设计的特点。第一，控制流基于顺序、选择、重复三种方式，不使用 GOTO 语句。第二，大程序要分解成小的模块。结构化程序设计可以使用由上至下、逐步细化的方式进行编程。用这种方法编制的程序易于阅读、易于检验其正确性、便于用户维护等等。

1.2.1 顺序设计

顺序程序设计就是根据事物的处理顺序和要求，将相应的指令按照它们所完成的功能

有机地结合起来的一个指令序列；这些指令的执行是按它们的排列顺序一条接一条的来执行。

如下例：该程序完成将一个数据库打开并显示该数据库的数据结构。

```
SET TALK OFF              &&关闭系统对话
USE  PZKO                     &&打开数据库“PZKO”
CLEAR                         &&清除屏幕
DISPLAY STRUCTURE         &&显示数据库结构
WAIT "请键入任意一键返回"    &&等待用户响应
CLEAR                         &&清除屏幕
SET TALK ON                   &&打开系统对话
RETURN                    &&返回调用程序
```

从上例可以看出顺序程序设计不需要特殊的编程技巧，只要将所需完成的工作一步一步地分析清，再将之翻译成相应的 FoxPro 命令即可。

1.2.2 分支设计

最简单的程序只是一列命令从头执行到尾。但世界上的事物不可能只是一个简单的顺序过程；大多数的时候你需要根据不同的情况采取不同的处理方法。这就需要程序本身具备有控制程序流向的能力，这也是结构程序设计的三大特征之一。FoxPro 系统为我们提供了专门用于选择和判断的命令序列。

1.2.2.1 简单判断语句（IF…ENDIF）

格式：IF（条件表达式）

```
.
语句序列
.
ENDIF
```

其中（条件表达式）可以是各种表达式的组合，但表达式的返回值必须是逻辑值；即真“.T.”或假“.F.”。当条件表达式的值为真时，执行在 IF 和 ENDIF 之间的命令行序列；当表达式的返回值为假时，则跳过在 IF 和 ENDIF 之间的指令序列而去直接执行在 ENDIF 语句后面的命令序列。

下面是一个带简单判断的范例。

```
SET TALK OFF
USE customer
GET EXPR '请输入查询的条件' TO temp;
    TYPE 'L' DEFAULT ' COMPANY=""
LOCATE FOR &temp     &&设定要查询的条件
```

```
IF FOUND()        &&判断是否找到
    ?'Company:'+company &&如果找到则显示客户信息
ENDIF
USE
SET TALK ON
```

1.2.2.2 选择判断语句（IF-ELSE-ENDIF）

如果有时需要判断的问题比较复杂。如上例，我们需要对问题进一步的处理，当未找到时如何处理？那简单的判断语句就显得力量比较单薄，同时使用起来也不是很方便。FoxPro 系统提供了另一种处理这种问题的语句。选择判断语句。其格式如下：

```
IF（条件表达式）
    命令序列一
ELSE
    命令序列二
ENDIF
```

执行此命令时，系统将先判断表达式是否为真，如果为真，系统将执行命令序列一，然后执行 ENDIF 后面的命令；如果表达式的返回值为假，系统将执行命令序列二，再执行 ENDIF 后面的命令。

进一步完善上例，增加当无法找到要查询的公司时，提示无该公司。

```
CLOSE DATABASES
USE customer
GET EXPR'请输入查询的条件' TO temp;
    TYPE 'L' DEFAULT 'COMPANY=""'
LOCATE FOR  &temp&&设定要查询的条件
IFFOUND()&&判断是否找到
    ?'Company:'+company &&如果找到则显示客户信息
ELSE
    ?'Condition'+temp+'无该公司情况' &&显示找不到
ENDIF
USE
```

由该例可见，编制具有逻辑判断能力的程序时，关键是如何使用判断语句。在使用判断语句时，应特别注意正确设置条件表达式，表达式的值决定着程序的执行走向和执行结果的正确与否。

另外：FoxPro 系统还提供了一个逻辑判断函数，其格式如下：

```
IIF(<expL>,<expr1>,<expr2>)
```

expL 为逻辑表达式
expr1 第一个表达式
expr2 第二个表达式

该函数的功能是：当逻辑表达式为真时，IIF 将返回第一个表达式的值；否则返回第二个表达式的值。该函数一般使用在只须作一简单判断的时候。如下例：

```
IF LEN(TRIM(DH1)#0)  &&如果单号一不等于“0”
    DH=TRIM(DH1)     &&将单号一赋给单号变量
ELSE
    DH=TRIM(DH2)       &&将单号二赋给单号变量
ENDIF
```

上面的命令序列就可以用逻辑判断函数代替。

```
DH=IIF(LEN(TRIM(DH1)#0),TRIM(DH1),TRIM(DH2))
```

1.2.2.3 多重选择（IF 语句的嵌套使用）

在实际应用中，经常有不止一种条件比较简单的问题，而面对的是很复杂的实际问题，对这些问题的判断是一个复杂的过程；这就引出了所谓的通过对 IF 语句进行嵌套来进行判断的问题。当使用 IF 语句嵌套时，存在着 IF 与哪个 ELSE 或 ENDIF 配对的问题。

在 FoxPro 系统中是按照下列规则处理的。对于每一个 IF 语句，系统在其后寻找 ELSE 和 ENDIF，若首先找到 ELSE，则继续寻找 ENDIF 与其配对；如果在找到 ENDIF 之前发现了其它的 IF 语句，则挂起当前寻找配对的工作，用同样的方法为 IF 配对。当该 IF 语句配对工作完成后，在它的 ENDIF 之后继续为前面打断的 IF 寻找 ELSE 或 ENDIF，这种配对方法称为“最近搭配方法”。

ELSE 或 ENDIF 不能与其他语句共行，因为系统不理会出现在它们后面的任何内容，利用这一特性可以在它们的后面加上一些注释内容。

下面是一个利用 IF 的嵌套关系来处理大量判断的例子。

```
SET TALK OFF
CLEAR
USE PZKO
XZ="1"
@3,31 SAY "请按下列方式选择排序方式"
@5,31 SAY "1-日期....."
@7,31 SAY "2-凭证编号....."
@9,31 SAY "3-科目代码....."
@11,31 SAY "其它不排序....."
@13,31 SAY "请输入你选择的排序方式" GET XZ
```

```
READ
IF XZ="1"    &&使用复杂判断语句
SORT ON RQ TO PZK1
ELSE
IF XZ="2"
    SORT ON BH TO PZK1
ELSE
    IFXZ="3"
    SORT ON KM TO PZK1
    ELSE
    RETURN
    ENDIF
ENDIF
ENDIF
USE PZK1
BROWSE
USE
RETURN
```

1.2.2.4 情况语句（DOCASE...ENDCASE）

我们从上面的例子可以看出，IF 语句嵌套的层数太多的话，不仅使程序复杂，又很容易出错，使用起来实在是不太方便。因此，FoxPro 系统提供了另一种处理需大量选择的判断语句，DOCASE 语句。

其格式如下：

```
DOCASE
    CASE<条件 1>
        语句序列 1
    CASE<条件 2>
        语句序列 2
    ...
    CASE<条件 N>
        语句序列 N
    [OTHERWISE
        语句序列]
ENDCASE
```

其中条件可以是各种表达式的组合，但表达式的值必须是逻辑值真或假，执行此命令时，系统将依次查看每一个 CASE 的条件，只要某一条件成立，则执行该条件下的命令序列。其它条件下的语句序列都被跳过，接下去执行 ENDCASE 后面的命令。如果所有的条件均不成立，在有选择项 OTHERWISE 的情况下，执行它后面的语句行序列，执行后接着执行 ENDCASE

后面的命令；在没有选择项 OTHERWISE 的情况下，执行 ENDCASE 后面的命令。

在 DOCASE 情况语句中，如果其中条件为真的情况多于一个，则执行第一个满足条件的。DOCASE 和 ENDCASE 必须成对出现。在 DOCASE 与第一个 CASE 之间的任何语句都不被执行。可以将 DOCASE 语句看成是一个一组嵌套的 IF 语句，它们的功能是等效的。

将上面的例子改写如下：

```
SET TALK OFF
CLEAR
USE PZK0
XZ="1"
@3,31 SAY "请按下列方式选择排序方式"
@5,31 SAY "1-日期....."
@7,31 SAY "2-凭证编号....."
@9,31 SAY "3-科目代码....."
@11,31 SAY "其它不排序....."
@13,31 SAY "请输入你选择的排序方式" GET XZ
READ
DOCASE      &&执行多重选择操作
CASE XZ="1"
    SORT ON RQ TO PZK1
CASE XZ="2"
    SORT ON BH TO PZK1
CASE XZ="3"
    SORT ON KM TO PZK1
OTHERWISE
    RETURN
ENDCASE     &&结束操作
USE PZK1
BROWSE
USE
RETURN
```

1.2.3 循环程序设计

循环程序是指按照给出的条件去重复执行一段完成特定功能的程序，直到给定的条件不再成立，才会退出循环体的执行。

1.2.3.1 FOR 语句

FOR 语句功能是 FoxPro 系统新增加的功能。该语句完成在一个指定次数的循环中执行一组指令。循环以 FOR 开头，以 ENDFOR 结束。

其格式如下：

```
FOR <memvar>=<expN1> TO <expN2>
    [STEP<expN3>]
        <statements>
    [EXIT]
    [LOOP]
ENDFOR|NEXT
```

<memvar>内存变量，作为一个计数器来确定循环内的语句执行次数。

<expN1>计数器的初始值

<expN2>计数器的结束值

<expN3>步长

EXIT 在循环结束时，将控制从循环体内转移到 ENDFOR 后面的命令。

LOOP 给定该参数，可以控制程序直接返回到 FOR，而不去执行 LOOP 后面的指令序列，但不影响计数器的正常工作。该命令类似于 BASIC 语言中的循环语句用法。

下面以三个例子来说明它们的用法。

例 1：单纯作为计数器使用

```
CLOSE DATABASES
CLEAR
FOR m=1 TO 10
    ?m
ENDFOR
```

例 2：EXIT 的应用，该程序在执行过程中，判断数据库指针是否已经指向尾部，如果是则退出循环体，而不去理会计数器是否计满。

```
SET TALK OFF
USE khk
GO TOP
STORE 2 TO I
STORE 10 TO J
STORE 2 TO K
FOR m=I TO J STEP K
    IF EOF()
        EXIT
    ENDIF
    GOTO m
```

```
    DISPLAY UPPER(company)
ENDFOR
USE
SET TALK ON
```

例 3：LOOP 的应用，该程序在执行过程中，如果发现为空记录，则不显示信息，且到循环体的第一条指令执行。

```
SET TALK OFF
USE khk
GO TOP
STORE 2 TO I
STORE 10 TO J
STORE 2 TO K
FOR m= I TO J STEP K
    IF NAME=SPACE(20)
        LOOP
    ENDIF
    GOTO m
    DISPLAY UPPER(company)
ENDFOR
USE
SET TALK ON
```

1.2.3.2 DO WHILE 语句

DO WHILE 语句是经常使用的循环命令。它的功能是：当逻辑条件保持为真时，重复执行循环体内的命令序列，直到逻辑表达式返回“.F.”。其命令格式如下：

```
DO WHILE<条件>
    <指令序列>
    [LOOP]
    <指令序列>
    [EXIT]
    <指令序列>
ENDDO
```

整个循环的结构由三部分构成。

（1）循环说明语句，即：DO WHILE 条件

它的作用是判断循环的条件是否满足，如果满足条件则执行循环体；否则跳出循环体。

（2）循环终端语句，即 ENDDO

它表示该循环以该语句为终点。

（3）循环体

包括在 DO WHILE 和 ENDDO 之间的所有语句。它是循环的主体，是需要多次重复执行的部分，一般完成一个独立的功能。

在使用循环时应注意以下几个问题。

- 循环说明语句中的条件可以是各种表达式的组合，但其返回值必须是逻辑值真和假。
- 在循环中 DO WHILE 和 ENDDO 语句必须成对出现。
- 在循环体中 LOOP 是可选的，可以出现在循环体中的任何位置。它的作用只是循环短路，迫使系统不执行 LOOP 后面的命令序列，而回到循环的开始位置。
- 在循环体中 EXIT 是可选的，可以出现在循环体中的任何位置。它的作用是强制循环结束，而不理会循环条件是否为真；只要执行该语句，系统就退出循环去执行 ENDDO 后面的第一条指令。

执行 DO WHILE 语句时，系统先对表达式求值，如果为真，则顺序执行循环体内的各条语句；当执行到 ENDDO 语句时，又判断条件是否为真，如果为真，继续重复执行循环体；如条件为假，则退出循环体执行 ENDDO 后面的语句。

注意：该语句同前面所讲过的 FOR 语句循环有较大的差别。DOWHILE 语句不会自己修复循环执行的条件，因此在使用过程中要考虑到系统结束循环的条件，否则可能会造成死循环。

使用举例：

```
SET TALK OFF
USE jhk
T=0
DO WHILE .T.
    IF EOF()
        EXIT
    ENDIF
    IF CMONTH(DATE())<>"JULY"
        SKIP
        LOOP
    ENDIF
    T=T+1
    SKIP
ENDDO
USE
?"总共有"+STR(T,3)+"条满足条件的记录"
```

```
SET TALK ON
```

1.2.3.3 SCAN...ENDSCAN 语句

SCAN 语句完成依次扫描数据库文件，同时针对满足指定条件的每一个记录执行该结构中包含的命令序列。SCAN 命令自动地把记录指针向下移动，然后再测试指定的条件。其命令格式如下：

```
SCAN
    [<scope>][FOR<expL1>]
    [WHILE<expL2>]
    [<statements>]
    [LOOP]
    [EXIT]
ENDSCAN
```

[<scope>]：用来确定 SCAN 语句的执行范围。如不指定则扫描整个数据库文件。
[FOR<expL1>]：针对指定范围内满足条件<expL1>为真的记录，执行指定的命令。
[WHILE<expL2>]：只有当当前记录使条件<expL2>仍为真时，才执行指定的命令序列。
[LOOP]：同 DO WHILE 命令相同。
[EXIT]：同 DO WHILE 命令相同。

程序举例：

```
SET TALK OFF
USE  jhk
T=0
SCAN FOR CMONTH(DATE())<>"JULY"
    T=T+1
ENDSCAN
USE
?"总共有"+STR(T,3)+"条满足条件的记录"
SET TALK ON
```

1.2.3.4 循环的嵌套使用

循环命令是可以嵌套使用的。在使用嵌套时应注意正确的次序。下面为使用循环的嵌套关系所举的几个例子。

例 1：　正确

```
DO WHILE1
    DO WHILE2
    .       .
```

```
    ENDDO2
ENDDO1
```

例 2： 正确

```
DO WHILE1
    FOR1


ENDFOR1
ENDDO1
```

例 3： 正确

```
DO WHILE1
IF

ELSE

ENDIF
ENDDO1
```

通过上面可以看出，下一个循环体一定要在上一个循环体内，它们之间不允许交叉嵌套。如下面的使用方法是错误的。

例 1：

```
DO WHILE1
    DO WHILE2
    .       .
    .   .   .
    ENDDO1
ENDDO2
```

例 2：

```
    DO WHILE
    FOR1

    ENDDO1
    ENDFOR1
```

例 3：

```
    DO WHILE1
    IF
```

```
    ELSE

    ENDDO1
    ENDIF
```

程序据例：

编写打印乘法口诀，要求打印出一个三角形。

```
SET TALK OFF
?""
FOR X=1 TO 9
Y=1
DO WHILE Y<=X
    S=X*Y
    ??STR(X,1)+"*"+STR(Y,1)+"="+STR(S,2)+""&&不换行显示
    Y=Y+1
ENDDO
?""&&换行显示
END FOR
SET TALK ON
```

1.3 过程调用和自定义函数

对于由上至下的程序设计来讲，最重要的就是：对一个过程的分解逐步求精，这样就可以将一个大的问题逐渐分解成一个个独立的具有一定功能的小的过程，最终达到对问题的求解。因此对过程和子程序的调用是一个重要的环节。调用子程序或过程，就是一个程序调用另一个程序模块。由于它们之间存在着密切的关系，前一过程是后一过程的前提和出发点，后一过程是对前一过程的进一步处理和完善。这样就使得整个系统之间的各个部分的逻辑关系非常明确、结构清晰、易于理解。该方法通常被结构化程序设计所采用。

对过程和子程序的调用也是减少程序代码长度和功能扩充的主要途径。一个过程被定义后，即可被其它程序所调用。它们之间如果有信息传递，可通过全局或局部变量来实现。

1.3.1 过程的概念和调用

在 FoxPro 环境中，过程和主程序基本没有什么分别。不同的是在过程程序的最后一条语句总是一条返回语句“RETURN”。

该命令将终止一个程序或一个用户定义的函数的执行，并把控制返回到调用程序，其格式如下：

RETURN[<expr>|TO MASTER|TO <programname>]

<expr>如果把一个程序作为用户定义的函数，那么该值将返回到调用程序。如果省略该参数，系统将自动返回逻辑值“.T.”到调用程序。

TO MASTER 选择该参数，系统将把控制权交给到最高一级的调用程序。TO <programname>选择该参数，系统将把控制权交给由<programname>指定的程序。

注意：执行 RETURN 命令时，系统将释放局部内存变量。

对于过程的调用非常的简单。只须执行下列命令即可。

DO<文件名>

如下例：

```
SET TALK OFF
TJ="Y"
DO WHILE UPPER(TJ)="Y"
    CLEAR
    ACCE "查找：[D：按日期] [M：按月份] [Y：按年份]" TO XZ
    KM="PZ"+XZ
    USE &KM
    DO EX01
    WAIT "继续查询？"TO TJ
ENDDO
CLOSE DATABASE
RETURN

**EX01.PRG
JX="Y"
DO WHILE JX$"Yy"
    INPUT  "请输入凭证号" TO PZNB0
    LOCATE ALL FOR PZNB=PZNB0
    IF FOUND()
        DISP
    ELSE
        ?"找不到该张凭证"
    ENDIF
    WAIT "继续查询？"TO JX
ENDDO
```