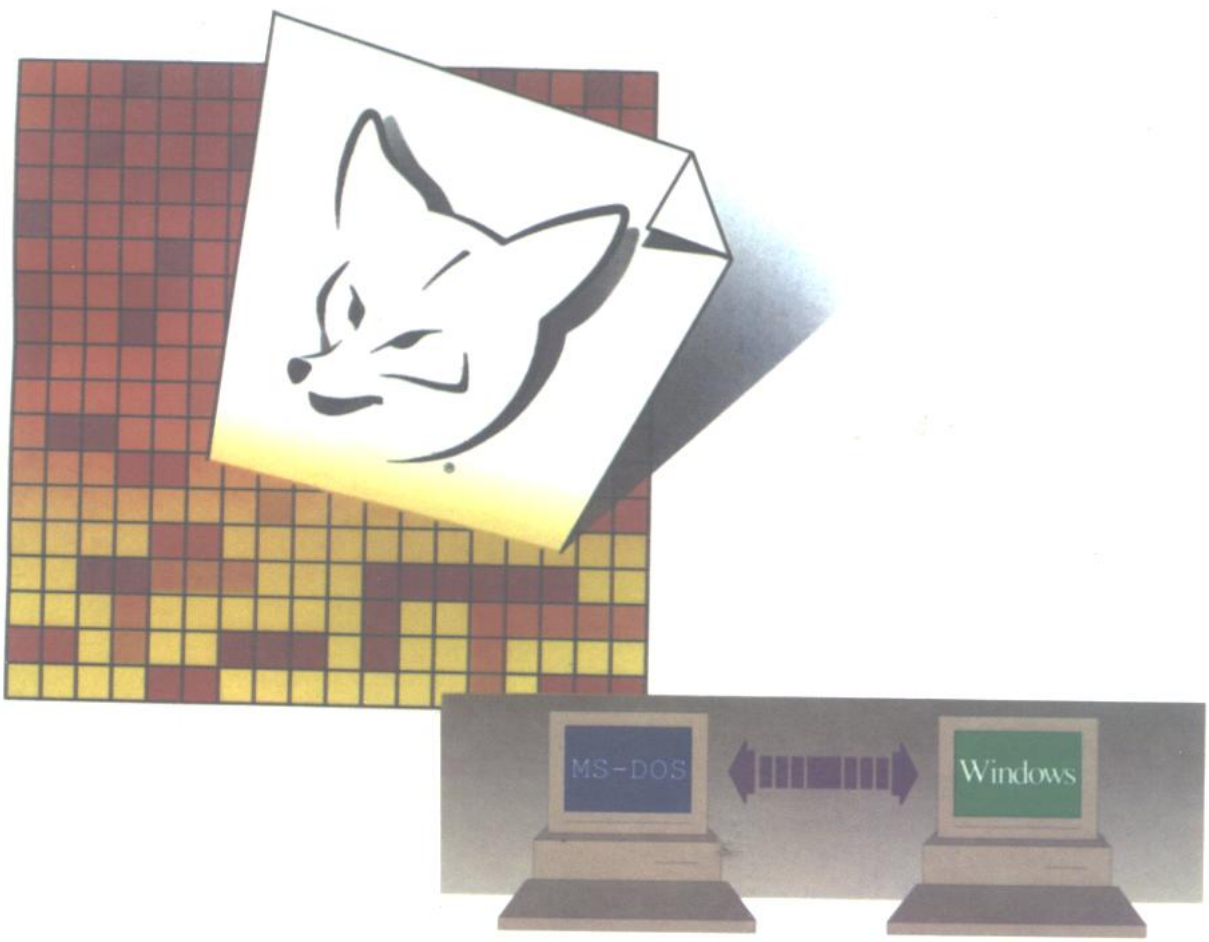
北京科海培训中心

FoxPro 2.5

命令与函数范例解析

(台)章立民　编著
科　培　改编



清华大学出版社

北京科海培训中心

FoxPro 2.5

命令与函数范例解析

（松岗）章立民　编著
科　培　改编

清华大学出版社

FoxPro 2.5 for Windows

命令与函数范例解析

章立民　编著

(京)新登字 158 号

FoxPro 2.5
命令与函数范例解析
(松岗) 章立民　编著
科　培　改编
☆
清华大学出版社出版
北京　清华园
门头沟胶印厂印刷
新华书店总店科技发行所发行
☆
开本：787×1092　1/16　印张：34.5　字数：839 千字
1993 年 12 月第 1 版　1993 年 12 月第 1 次印刷
印数：0001～5000 册
ISBN 7-302-01462-0/TP・577
定价：48.00 元

序

FoxPro 2.5 是目前世界上最快速的 PC 机数据库，其速度虽不敢说是“绝后”，但确实是非常快。FoxPro 2.5 对索引式数据的特定查询之所以会比同级的数据库快百倍甚至千倍，主要是利用名为 Rushmore 的快速数据查询技术。而且数据记录的数目愈多，就愈能感受到 FoxPro 2.5 速度快的优点。

除了在速度上的表现令人惊叹之外，在用户接口的设计上，FoxPro 2.5 for MS-DOS 可以说是文本模式(Text Mode)中的典范。而 FoxPro 2.5 for Windows 更能充分发挥图形界面的特色。用户会发现它不仅美观且颇具智慧！正如您所期望的，OLE(对象连接与嵌入)及 DDE(动态数据交换)在 FoxPro for Windows 中都可发挥得淋漓尽致。此外，如众多程序设计人员所期盼的，一个可产生可执行文件. EXE 的编译与连接程序也包含在 FoxPro 2.5 for MS-DOS/Windows 中，而您所熟悉的用户自定义窗口及用户自定义菜单更作了强化的工作。凡此种种在本书中都将有详细的介绍。

开发 FoxPro 的 Microsoft 公司已表示，未来将陆续推出 FoxPro for Unix 及 FoxPro for Mac，且已着手设计开发，因此可以预见，未来 FoxPro 将普及并得到广泛应用。

本书的习惯用法

为了使读者在阅读本书时能迅速掌握以达事半功倍之效，特将书中各个命令与函数语法的惯用表示式列示如下：

1. 所有冠以中括号([])的命令或函数表示可选择的内容(参数或关键字)。

2. 在一命令中以符号"|"分隔的两个选项(例如 ON|OFF)，表示一次只能使用两者之一，而不能同时使用。

3.

<expC>	:表示字符表达式
<expN>	:表示数值表达式
<expL>	:表示逻辑表达式
<expD>	:表示日期表达式
<expr>与<expression>	:表示不限定类型的表达式

4. 所有命令皆以大写列示，但在使用时大、小写皆可。

5. 每当一个函数允许或需要一个 <alias> 的参数时，可以使用别名(alias name)、工作区英文字母代号(A～J)、工作区的数字编号(1～225)。

6. 若是在一个参数后面加上三个句点"...",表示此参数可以无限重复使用，直到用完计算机所有的内存或是达到单一程序行最多只能有 2048 个字符的限制。

目　录

第一章　命令篇

第二章　函数篇

第一章　命令篇

#DEFINE... #UNDEF

语法： #DEFINE <constant name> <expr>
#UNDEF <constant name>

如果您熟悉C语言，相信对预处理命令(Preprocessor Directives)必定不会陌生。FoxPro 2.5提供的预处理命令#DEFINE与#UNDEF能在程序中建立编译常量。利用#DEFINE建立常量而不使用内存变量，可少占内存，提高程序执行的效率，简化程序并增加程序的可读性。

要使用#DEFINE建立一个常量，只要在<constant name>中指定此常量名称，并在<expr>中指定此常量所代表的值。当程序被编译时，FoxPro便进行正文替换，且每当<constant name>出现在程序中时，FoxPro便将它视为是<expr>。一旦想终止常量的替换，只要在#UNDEF之后指定此常量的名称(即<constant name>)即可。

替换仅仅发生在#DEFINE语句与#UNDEF语句之间的那些程序行中，而且此常量仅适用于建立此常量的程序中。

<constant name>

欲使用#DEFINE建立一个编译时的常量，请在<constant name>中指定此常量的名称，<constant name>必须是一个合法的FoxPro名字。这意味着<constant name>必须以一个英文字母或是下划线开头，而且最长不可超过10个字符。为了提高程序的可读性并简化调试过程，建议在对编译的常量命名时，采用一个标准的命名约定。

注意： 不要使用FoxPro的关键字(Keyword)作为常量的名称。

<expr>

当使用#DEFINE建立一个编译时的常量时，请在<expr>中指定此常量的值。<expr>可以是一个字符、数值、日期或逻辑型的表达式。

程序范例

正如前面所提到的，利用预处理命令#DEFINE建立一个编译常量而不使用内存变量，可少占内存、提高程序执行的效率及简化程序。此外，最直接让程序员感受到的好处是增加程序的易读性。举例来说，我们经常在程序中根据INKEY()或LASTKEY()函数的返回值来判断用户究竟按下哪一个键，以决定后面的程序走向。然而各个按键的INKEY()及LASTKEY()的返回值皆不相同，因此当我们见到类似如下程序代码时，很难了解究竟哪一个数值代表哪一个键：

```
...
...
choice = INKEY(0,"HM")
CLEAR TYPEAHEAD
DO CASE
```

```
    CASE choice = 27
         EXIT
    CASE choice = 1
         GO TOP
    CASE choice = 6
         GO BOTTOM
    ...
    ...
ENDCASE
...
...
```

但是，如果我们能够利用有明确含义的常数来置换每个按键 INKEY()及 LASTKEY()的返回值，将大大增加程序的易读性。如下所示：

```
* 程序文件名称:DEMO1.PRG
#DEFINE Home 1
#DEFINE End  6
#DEFINE PgDn 3
#DEFINE PgUp 18
DOW alex FROM 5,01 TO 24,79:
  FONT "Helv", 12 ;
  STYLE "B";
  TITLE "数据浏览"
mROW = WROWS ("alex") - 1
mCOL = WCOLS ("alex")

ACTIVATE WINDOW alex
@ 01,06 TO 08,55 PEN 2

DO WHILE .NOT. JUDGE
   @ 2,10 SAY."客户编号:" +cust_id
   @ 3,10 SAY "公司名称:" +company
   @ 4,10 SAY "接洽人:" +contact
   @ 5,10 SAY "住址:" +address1
   @ 6,10 SAY "税率:" +STR (taxrate, 4, 2)

   @mROW - 1, 01  SAY "<Home 键 第一项> <End 键 最后一项> ";
                      "<PgUp 键 下一项> <PgDn 键 上一项>";
                      COLOR RGB (225,0,0,0,0,0)
   @mROW, 26  SAY "<Esc 键 退出>";
                      COLOR RGB (225,0,0,0,0,0)

   CHOICE = INKEY(0,"HM")
   CLEAR TYPEAHEAD
   DO CASE
      CASE CHOICE = Home
           GO TOP
      CASE CHOICE = End
           GO BOTTOM
      CASE CHOICE = PgDn
           SKIP
           IF EOF( )
                WAIT "已到达最后一项数据" WINDOW NOWAIT TIMEOUT 5
                SKIP -1
           ENDIF
      CASE CHOICE = PgUp
           SKIP -1
```

```
            IF BOF( )
                WAIT "已到达第一项数据" WINDOW NOWAIT TIMEOUT 5
                GO TOP
            ENDIF
        CASE CHOICE = ESC
            DEACTIVATE WINDOW alex
            RELEASE WINDOWS alex
            #UNDEF    Home
            #UNDEF    End
            #UNDEF    PgDn
            #UNDEF    PgUp
            #UNDEF    Esc
            STORE .T. TO JUDGE
    ENDCASE
ENDDO
```

#IF... #ENDIF

语法：

```
#IF<expN1>|<expL1>
    <statements>
[#ELIF <expN2>|<expL2>
    <statements>
...
...
#ELIF <expNn>|<expLn>
    <statements>]
[#ELSE
    <statements>]
#ENDIF
```

预处理命令#IF... #ENDIF 可进行有条件地编译，即有条件地将原始程序代码嵌入一个已编译的程序中。#IF... #ENDIF 能够提高程序的易读性，使已编译的程序文件减小，在某些情况下甚至能提高程序的执行效率。

当#IF... #ENDIF 结构被编译时，此结构中连续的逻辑或数值表达式便会被计算。而计算结果决定了哪一部分的 ForPro 内容(即程序码)将会被编译并放入编译后的程序文件中。

```
#IF <expN1>|<expL1>
    <statements>
```

FoxPro 会计算紧跟在#IF 之后的数值表达式<expN1> 或逻辑表达式<expL1>。如果此表达式是数值表达式且为非零值，或者此表达式为逻辑表达式且为逻辑真值.T.，FoxPro 将会编译#IF 后下一行开始的程序码(<statements>)，即将它们嵌入已编译的程序中。接着便跳离#IF... #ENDIF 结构，而且#ENDIF 后的程序行将跟着被编译。

反之，如果此表达式是数值表达式且为零值；或者此表达式为逻辑表达式且为逻辑假值.F.，FoxPro 将不会编译 #IF 后的程序代码(<statements>)。但是，所有位于 #ELIF 之后的表达式将会继续被计算。

#ELIF <expN2>|<expL2>
<statements>
...
...
#ELIF <expNn>|<expLn>
<statements>

如果#IF 中 <expN1>为零或<expL1>为逻辑假值 .F.,则#ELIF 后的表达式将会接着被计算。如果存在的话,第一个 #ELIF 表达式<expN2>或<expL2>会被计算。假如<expN2>为非零值或<expL2>为逻辑真值.T.,紧跟在#ELIF 后的程序代码(<statements>)将会被编译而加入到已编译的程序中。接着便跳离#IF...#ENDIF 结构,#ENDIF 后的程序代码将跟着被编译。

假如<expN2>为零或者<expL2>为逻辑假值 .F.,则位于 #ELIF 后的程序代码(<statements>)将不会被编译——不会被嵌入已编译的程序代码中。而如果存在下一个#ELIF,则此#ELIF 后的表达式将继续被计算。其余依此类推。

#ELSE
<statements>

如果并未指定任何#ELIF 命令,或者所有的 #ELIF 命令都被计算为零或逻辑假值 .F.,那么是否存在 #ELSE 命令将决定其他程序代码(<statements>)是否会被编译并嵌入已编译的程序中:

- 假如指定了#ELSE 命令,则#ELSE 后的程序码(<statements>)将会被编译并被嵌入已编译的程序中。
- 假如并未指定#ELSE 命令,则所有位于#IF 与#ENDIF 之间的程序代码(<statements>)将不会被编译——不会被嵌入已编译的程序中。此时 FoxPro 会跳离#IF...#ENDIF 结构,并继续从 #ENDIF 后的第一行程序代码开始编译。

程序范例

```
*程序文件名称:DEMO2.PRG
#IF'WINDOWS' $ UPPER(VERSION())
    ?'This is FoxPro for Windows'
#ELIF'MAC' $ UPPER(VERSION())
    ?'This is FoxPro for Mac'
#ELIF'UNIX' $ UPPER(VERSION())
    ?'This is FoxPro for UNIX'
#ELSE
    ?'This is FoxPro for DOS'
#ENDIF
```

RUN|!

语法: RUN[/N[K]]<MS-DOS 命令|外部程序名>
或