# 第3章

# 基本操作及其预备知识

本章知识点如图 3-0 所示。

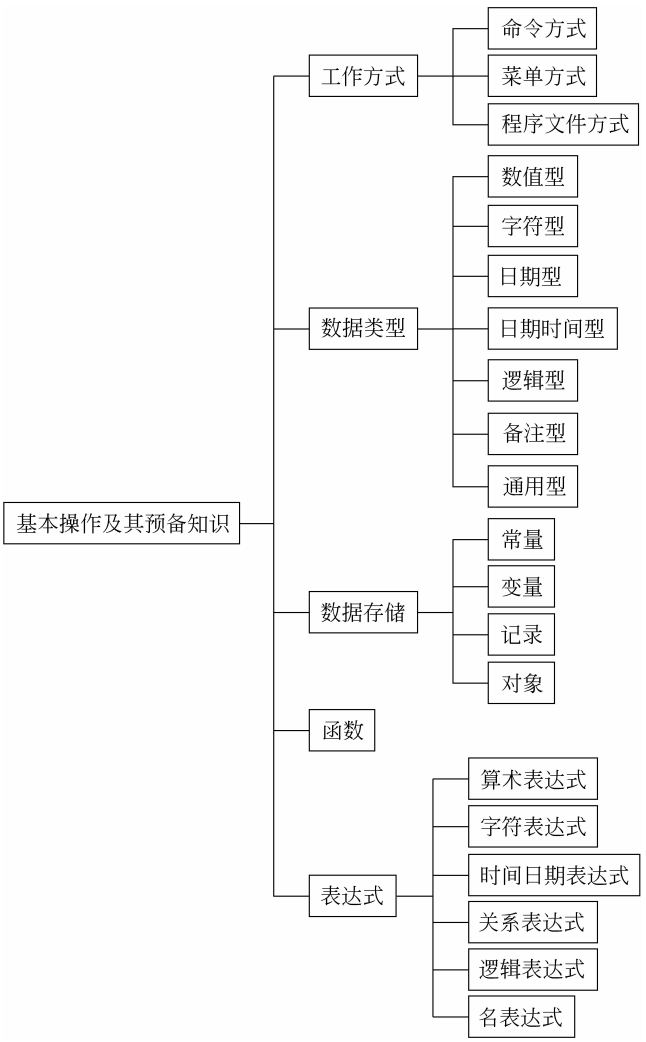


图 3-0　第 3 章知识点

## 3.1 Visual FoxPro 工作方式

Visual FoxPro 不但拥有大量的交互式数据库管理工具，而且还有一整套功能完善的程序语言系统及面向对象可视化程序编写工具。为此，Visual FoxPro 系统给用户提供了三种工作方式，它们是命令方式、菜单方式及程序文件方式。

### 1. 命令方式

命令方式通过用户在命令窗口输入操作命令，立即执行操作的交互式工作方式。

在命令窗口中，可以输入单个的操作命令和系统命令，从而完成对数据库的操作管理、系统程序的设计、各种对象的创建和维护，以及系统环境的设置等。命令窗口不仅是 Visual FoxPro 命令的执行窗口，也是 Visual FoxPro 命令文件的编辑窗口。

如果用户比较熟悉 Visual FoxPro 系统命令及数据库操作命令，用命令方式要比菜单方式来得更方便快捷。但是，由于 Visual FoxPro 系统命令格式较为复杂，命令种类繁多，初学者难于掌握，也不妨考虑采用菜单方式。

### 2. 菜单方式

菜单方式是用户通过对系统菜单提供的菜单选项选择，完成对数据库的操作管理、系统程序的设计、各种对象的创建和维护，以及系统环境设置的操作。

菜单方式也是一种交互式工作方式，只要选择一个菜单选项，系统便完成一个对应的操作。这种直观、简捷的界面操作，不要求熟悉操作命令，只要能够把握操作流程，选择合适的菜单选项，就能够完成操作任务。

菜单方式比命令方式容易掌握，即学即会，但其操作步骤过多，任务的执行时间要长。建议对于一些经常性的操作，还是使用命令方式进行更妥。

### 3. 程序文件方式

程序文件（简称程序）是为解决某一特定问题命令序列的集合。因此，程序文件也称为命令文件，程序文件方式也称为命令文件方式。

程序文件方式是先通过程序文件编辑工具，将对数据库进行的各种操作管理的命令或对系统环境进行设置的命令集中在一个以.PRG 为扩展名的程序文件中；其次，通过菜单方式或命令方式或程序文件方式运行这一程序文件；最后，系统将根据程序文件中的每一个命令完成操作任务。

用程序文件方式进行数据库管理，是把许多操作命令集中使用，不仅运行效率高，还可重复操作。但是，若想编写一个解决实际问题的应用程序，不是一日之功，需要对 Visual FoxPro 命令和语句进行系统的学习，同时还要掌握一定的程序设计方法。

有关程序设计的方法将在后面分别介绍。

## 3.2 数据类型

数据是反映客观事物属性的记录，它的类型决定了它的存储和使用方式。在许多软件环境下，数据通常只分为数值型和字符型两种基本类型，而 Visual FoxPro 系统为了使用户建立和使用数据库更加方便，除了上述两种数据类型外，又细化分出了更多的类型。

### 1. 数值型数据

数值型数据是由数字(0～9)、小数点和正负号组成的。

对数值型数据，由于它们表达或存储形式的不同，又被分为数值型(Numeric)、浮点型(Float)、双精度型(Double)和整型(Integer)。

### 2. 字符型数据

字符型数据描述的是不具有计算能力的文字数据类型，它是最常用的数据类型之一。

字符型数据(Character)是由汉字和 ASCII 字符集中可打印字符(英文字符、数字字符、空格及其他专用字符)组成的。

字符型数据的最大长度为 254 个字符。

### 3. 日期型

日期型数据(Date)是用于表示日期的数据。

其格式为{mm/dd/yyyy}。

其中 mm 代表月，dd 代表日，yyyy 代表年。

长度固定为 8 位。

### 4. 日期时间型

日期时间型数据(Date Time)是描述日期和时间的数据。

其格式为{mm/dd/yyyy hh:mm:ss}。

其中 yyyy 代表年，前两个 mm 代表月，dd 代表日，hh 代表小时，后两个 mm 代表分，ss 代表秒。

长度固定为 8 位。

### 5. 逻辑型

逻辑型数据(Logic)是描述客观事物真假的数据，用于表示逻辑判断结果。逻辑型数据只有真(.T.或.Y.)和假(.F.或.N.)两种值。

长度固定为 1 位。

### 6. 备注型

备注型数据(Memo)是用于存放较长的字符型数据的数据类型。

备注型数据是字符型数据的特殊形式，备注型数据没有数据长度限制，仅受限于现有的磁盘空间。它只用于数据表中的字段类型的定义，其字段长度固定为 4 位，而实际数据被存放在与数据表文件同名的备注文件中，长度根据数据的内容而定。

**7. 通用型**

通用型数据(General)是用于存储 OLE 对象的数据。

通用型数据中的 OLE 对象可以是电子表格、文档、图片等。

OLE 对象的实际内容、类型和数据量取决于连接或嵌入 OLE 对象的操作方式。如果采用连接 OLE 对象方式，则数据表中只包含对 OLE 对象的引用说明，以及对创建该 OLE 对象的应用程序的引用说明；如果采用嵌入 OLE 对象方式，则数据表中除包含对 OLE 对象的引用说明及对创建该 OLE 对象的应用程序的引用说明外，还包含 OLE 对象中的实际数据。

通用型数据只用于数据表中的字段类型的定义。其长度固定为 4 位，实际数据长度仅受限于现有的磁盘空间。

## 3.3 数据存储

数据输入、输出是通过数据的存储设备完成的。通常我们都是将数据存入到常量、变量、数组中，而在数据库系统环境下，还可以把数据存入到字段、记录和对象中。我们把这些供数据存储的常量、变量、数组、字段、记录和对象统称为数据存储容器。

对数据进行加工处理时，数据存储容器的不同，决定了数据的类型和使用方法的不同，同时也限定了数据的使用范围。

以下我们介绍常用的几种数据存储容器。

### 3.3.1 常量

常量是一个命名的数据项，是在命令或程序中直接引用的实际值，其特征是它在所有的操作中其值不变。

常量有数值型、浮点型、字符型、逻辑型、日期型和时间日期型 6 种。

**1. 数值型常量**

由数字(0～9)、小数点和正负号组成。如 8713.25、－12368、＋3325.67。

**2. 浮点型常量**

由数字(0～9)、小数点和正负号组成的浮点格式。如－987e＋15、＋123456e－79。

**3. 字符型常量**

由汉字和 ASCII 字符集中可打印字符组成，并由定界符(“ ”、‘ ’、[])括起来的字符串。如“STRING”、“数据库的应用系统”。

**4. 逻辑型常量**

由表示逻辑判断结果的“.T.”或“.F.”符号组成。如.t.或.T.、.f.或.F.。

**5. 日期常量**

由按其规定格式{mm/dd/yyyy}表示的符号组成。如{20/12/2011}、{01/01/2011}。

**6. 时间日期型常量**

由按其规定格式{mm/dd/yyyy hh:mm:ss}表示时间日期的符号组成。如{12/12/2011 11:50:00}。

### 3.3.2 内存变量

内存变量是一般意义上的简单变量。

每一个内存变量都必须有一个固定的名称,以标识它在该内存单元中的存储位置,用户可以通过变量标识符来存取常量。

**1. 内存变量的命名**

内存变量名是由字母、数字和下划线组成的,不能与 Visual FoxPro 系统提供的命令、语句专用符号相同,也不能和 Visual FoxPro 系统提供的函数名同名。

如果内存变量与数据表中的字段变量同名,用户在引用内存变量时,要在其名字前加一个前缀(m.),用以强调这一变量是内存变量。

**2. 内存变量的类型**

用户可以根据需要定义内存变量类型,它的类型取决于首次接收的常量的类型。

也就是说,内存变量类型的定义是通过赋值语句来完成的。

它的类型有数值型、浮点型、字符型、逻辑型、日期型、时间日期型 6 种。

**3. 内存变量的赋值**

内存变量是内存中的临时存储单元,可以用来在程序的执行过程中保留中间结果和最后结果,或用来保留对数据库进行某种分析处理后得到的结果。

给内存变量赋值的命令有 STORE 和=。

STORE 命令格式:

```
STORE <表达式>  TO  <内存变量表>
```

命令功能:

先计算<表达式>的值,然后将该值赋给<内存变量表>中每一个变量。

“=”命令格式:

```
<内存变量>=<表达式>
```

命令功能：

是先计算＜表达式＞的值，然后将该值赋给以＜内存变量＞为名的内存变量。

特别要注意，除非用内存变量文件来保存内存变量值；否则，当退出 Visual FoxPro 系统后，内存变量的值也会与系统一起消失。

**例 3.1** 给内存变量 A1，A2 赋值。

```
A1=123+456
STORE  "Visual FoxPro"  TO  A2
```

#### 4. 内存变量值的输出

输出内存变量值可以使用“?”命令。

“?”命令的格式：

```
?<表达式>
```

“?”命令的功能是先计算＜表达式＞的值，然后将该值输出到显示器上。

**例 3.2** 输出内存变量 A1，A2 的值。

```
?A1
?A2+"系统"
```

#### 5. 内存变量的清除

在系统程序开始运行时或程序运行过程中，有时或经常对内存变量进行一下清理，会提高程序的运行速度和质量。

使用 RELEASE 命令可以清除不再使用的内存变量或所有内存变量。

RELEASE 命令格式：

```
RELEASE [<内存变量表>] [ALL]
```

**例 3.3** 清除 A1，A2 两个内存变量。

```
RELEASE A1,A2
```

**例 3.4** 清除所有的内存变量。

```
RELEASE ALL
```

#### 6. 内存变量的作用域

内存变量的作用域就是它的作用范围。根据内存变量的作用范围分为全局变量、局部变量和本地变量。

使用 LOCAL、PRIVATE、PUBLIC 命令可以定义内存变量的作用域，也可以使用系统默认的范围作为内存变量的作用域。

1）定义全局变量

用 PUBLIC 命令定义的内存变量可视为全局变量。

全局变量在全部程序、过程和自定义函数，以及它所调用的程序、过程和自定义函数中都有效。即使整个程序结束，全局变量也不被释放，它们的值仍然保存在内存中。如要释放，要用 RELEASE 命令进行操作。

全局变量必须先定义后赋值。已经定义成全局变量的内存变量还可以在下级程序中进一步定义成局部变量。但已经定义成局部变量的内存变量却不可以再定义成全局变量。除了在程序中将内存变量定义成全局变量外，在命令窗口使用的所有内存变量都视为全局变量。

PUBLIC 命令格式：

```
PUBLIC 〈内存变量表〉
```

命令功能：

定义〈内存变量表〉中指定的内存变量为全局变量。

**例 3.5** 定义 P1，P2 为全局变量。

```
PUBLIC P1,P2
```

2）定义局部变量

用 PRIVATE 命令定义的内存变量可视为局部变量。

局部变量在定义它的程序以及被该程序调用的程序、过程和局部变量函数中有效。一旦定义它的程序运行完毕，局部变量将从内存中释放；如果定义它的程序再调用其他子程序，则该变量在子程序中继续有效；如果它在子程序中改变了值，则返回调用程序时也带回新值，并在程序中继续使用；如果另一个程序调用它的程序，调用程序中与定义它的程序同名的内存变量将被屏蔽，而一旦调用程序运行完毕，所有被屏蔽的内存变量便又恢复原来的状态。

PRIVATE 命令格式：

```
PRIVATE〈内存变量表〉
```

命令功能：

定义〈内存变量表〉中指定的内存变量为局部变量。

**例 3.6** 定义 P11，P12 为局部变量。

```
PRIVATE P11,P12
```

3）定义本地变量

用 LOCAL 命令定义的内存变量可视为本地变量。

本地变量只在定义它的程序中有效。一旦定义它的程序运行完毕，本地变量将从内存中释放。需要注意的是，无论是被定义它的程序调用的程序，还是调用定义它的程序都不能使用这些内存变量。

LOCAL 命令格式：

```
LOCAL〈内存变量表〉
```

命令功能：

定义〈内存变量表〉中指定的内存变量为本地变量。

**例 3.7** 定义 L1,L2 为本地变量。

```
LOCAL L1,L2
```

### 3.3.3 数组变量

数组是一组有序内存变量的集合。或者说，数组是由同一个名字组织起来的简单内存变量的集合。其中每一个内存变量都是这个数组的一个元素，它是由一个以行和列形式表示的数组元素的矩阵。

所有的数组元素是用一个变量名命名的一个集合体，而且每一个数组元素在内存中独占一个内存单元。为了区分不同的数组元素，每一个数组元素都是通过数组名和下标来访问的(如 A[1,2]、B[5])。

数组是内存变量的一种特殊形式，使用时也同样要注意其作用域。

#### 1. 定义数组

在 Visual FoxPro 系统环境下，可以通过 DIMENSION 或 DECLARE 定义只有一个下标的一维数组或有两个下标的二维数组。数组一旦定义后，它的初始值为逻辑值.F.，下标的起始值是 1。

DIMENSION 命令格式：

```
DIMENSION〈数组名 1〉(〈下标 1〉[,〈下标 2〉])
[,〈数组名 2〉(〈下标 1〉[,〈下标 2〉])]…
```

DECLARE 命令格式：

```
DECLARE 〈数组名 1〉(〈下标 1〉[,〈下标 2〉])
[,〈数组名 2〉(〈下标 1〉[,〈下标 2〉])]…
```

以上两个命令功能：定义一个或多个数组，同时又定义了下标的个数及下标的上界。

在 Visual FoxPro 系统中，数组也可以重新定义，并能动态地“放大缩小”，如果改变原数组的维数和容量，原数组中每个元素的值不变。因为数组是内存变量，在新定义或重新定义数组时，要特别注意内存空间的大小，避免出现内存不够的现象。

**例 3.8** 定义一个一维数组 SZ1 和一个二维数组 SZ2。

```
DIMENSION  SZ1(10),SZ2(5,2)
```

上面这一定义一旦完成，系统就允许使用 SZ1 和 SZ2 两个数组。

SZ1 是一维数组，SZ1 下标的上界为 10，下界为 1。数组元素分别为 SZ1(1)，SZ1(2)，SZ1(3)，…，SZ1(10)。

SZ2 是二维数组，SZ2 第一个下标为行标，上界为 5，下界为 1，SZ2 第二个下标为列

标，上界为 2，下界为 1。数组元素分别为 SZ2(1,1)，SZ2(1,2)，SZ2(2,1)，SZ2(2,2)，…，SZ2(5,1)，SZ2(5,2)。

**2. 数组类型**

数组类型是指数组元素的类型。因为每一个数组元素又是一个内存变量，所以它的类型同样由它接收的数据的类型所决定。

在 Visual FoxPro 系统环境下，同一个数组元素在不同时刻可以存放不同类型的数据，在同一个数组中，每个元素的值可以是不同的数据类型。

**3. 数组赋值**

给数组赋值，就是分别给每个数组元素赋值，与给内存变量赋值操作完全相同。

**例 3.9** 定义一个一维数组 X，给所有数组元素赋值并输出其值。

```
DIMENSION  X(4)
X(4)="12345"
STORE  0  TO  X(1),X(2),X(3)
?X(1),X(2),X(3),X(4)
```

**4. 数组传递**

使用命令 SCATTER、COPY TO ARRAY、GATHER 可以实现数据表与数组间数据的传递。

COPY TO ARRAY 命令格式：

```
COPY TO ARRAY〈数组名〉[FIELDS〈字段名表〉]
[〈范围〉][FOR〈条件 1〉][WHILE〈条件 2〉]
```

SCATTER 命令格式：

```
SCATTER [FIELDS〈字段名表〉]TO〈数组名〉
```

以上两个命令功能：将当前数据表中的数据传递到〈数组名〉指定的数组中。

GATHER 命令格式：

```
GATHER FROM〈数组名〉[FIELDS〈字段名表〉]
```

命令功能：将〈数组名〉指定的数组中的数据传递到当前数据表中。

**5. 数组特性**

在 Visual FoxPro 数据库管理系统下，对数据库进行操作时引用数组，会使数据操作更方便。

数组和数据表相比，它有如下优点：

(1) 数组可以不像数据表一样有一个固定的结构。

(2) 因为数组中的数据存放在内存中，数据表中的数据存放在磁盘上，所以对数组中

数据的访问比对数据表中的数据访问速度要快。

(3) 数组可以在原有的内存空间进行数据排序,不需要额外的内存和磁盘空间。

### 3.3.4 字段变量

字段变量是数据库管理系统中的一个重要概念。它与记录一纵一横构成了数据表的基本结构。通过前面的学习,我们已经知道,一个数据库是由若干相关的数据表组成的,一个数据表是由若干个具有相同属性的记录组成的,而每一个记录又是由若干个字段组成的。

字段变量就是指数据表中已定义的任意一个字段。

我们可以这样理解,在一个数据表中,同一个字段名下有若干个数据项,而数据项的值取决于该数据项所在记录行的变化,所以称它为字段变量,也有人把字段变量称为字段名变量。

字段变量的数据类型与该字段定义的类型一致。

字段变量的类型有数值型、浮点型、整型、双精度型、字符型、逻辑型、日期型、时间日期型、备注型和通用型等。

使用字段变量首先要建立数据表,建立数据表时首先定义的就是字段变量属性(名字、类型和长度)。字段变量的定义及字段变量数据的输入输出需要在表设计器和表浏览、编辑窗口中进行。有关这方面的内容将在第 4 章详细介绍。

### 3.3.5 记录

记录是数据表中一组数据项的集合。

在同一个数据表中可以有若干个记录,每一个记录具有相同的字段个数。

在 Visual FoxPro 系统中,许多操作都是通过记录操作来完成的。

有关这方面的内容将在第 4 章详细介绍。

### 3.3.6 对象

对象是数据存储器的一种。对象是类的实体,是任何具有属性和方法的信息的集合。

对象的建立可以通过设计器和 CREATE OBJECT()函数实现。

在 Visual FoxPro 系统中,引用对象是可视化编程的重要手段。

有关这方面的内容,将在第 10 章详细介绍。

## 3.4 函数

Visual FoxPro 系统提供了一批具有特定功能的标准函数。这些函数实际上是系统提供的固定的程序,用户可直接引用,从而完成某些特定的操作。

#### 1. 标准函数的类型

根据每一个函数的功能,可将标准函数大致分为以下类型。

(1) 数值型函数。

(2) 字符型函数。