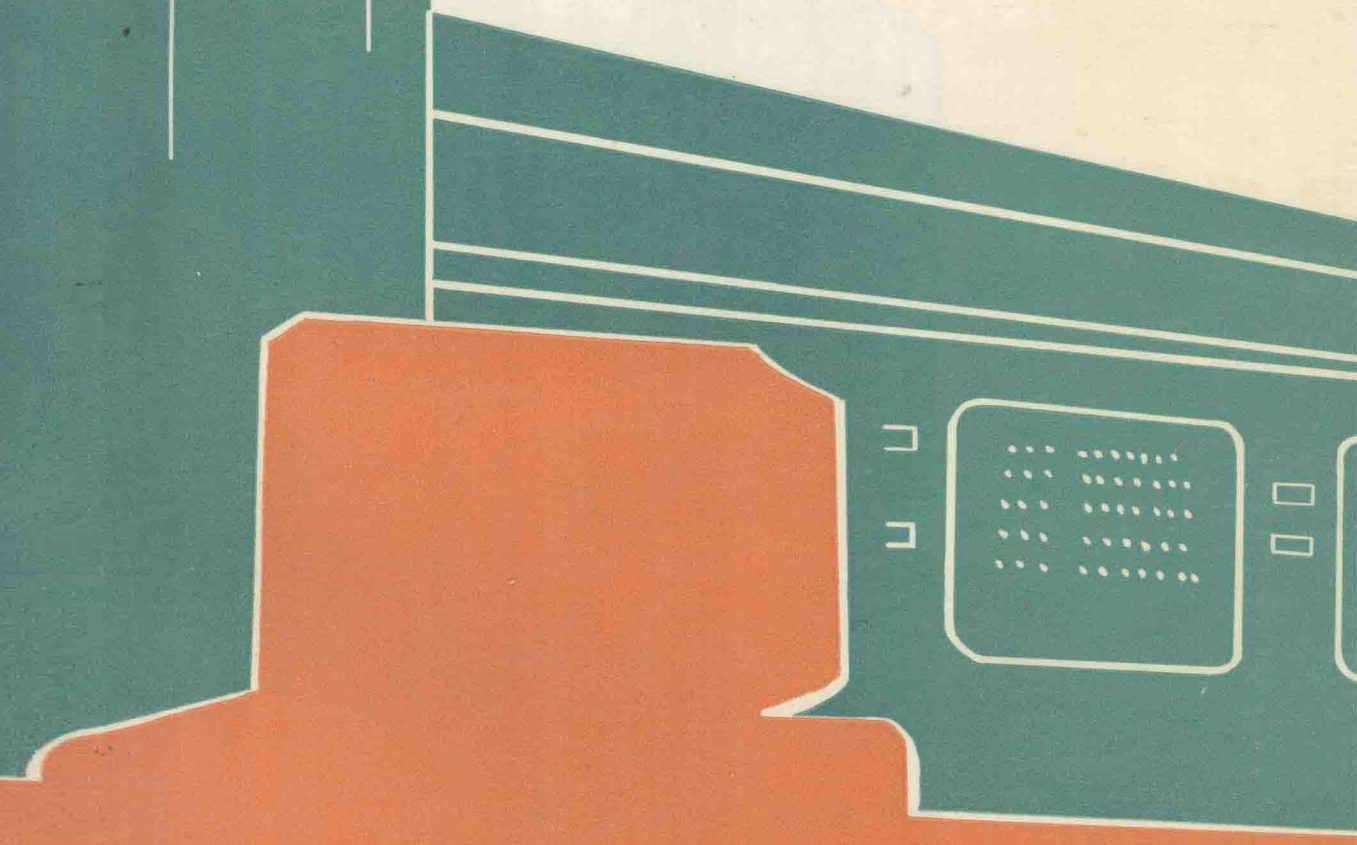
最新 dBASE 编译数据库

Clipper5.0 使用指南

甘登岱　木林森　彭建军　编著



航空工业出版社

最新 dBASE 编译数据库

Clipper 5.0 使用指南

甘登岱　木林森　彭建军　编著

友　元　审

航空工业出版社

1993

(京)新登字161号

内 容 提 要

本书全面系统地介绍了运用 Clipper 进行程序设计的基础知识。全书共分十二章，主要内容包括：Clipper 语言及环境，Clipper 程序编译、连接和调试，数组、备注字段处理及操作，文件处理与查询，与 C 和汇编语言接口，覆盖及用户界面管理，网络应用程序等。本书在附录部分汇集了所有 Clipper 编译及连接程序错误信息和 Clipper 命令与函数。此外，作者还精心设计了大量的编程实例。

本书适用于大专院校师生，培训班师生及数据库管理应用软件开发人员。

最新 dBASE 编译数据库

Clipper 5.0 **使用指南**

甘登岱　木林森　彭建军　编著

友　元　审

航空工业出版社出版发行

（北京市安定门外小关东里14号）

—邮政编码：100029—

全国各地新华书店经售

煤炭出版社印刷厂印刷

1993年8月第1版　　1993年8月第1次印刷

开本：787×1092毫米　1/16　　印张：25.75

印数：1—4100　　字数：640　千字

ISBN　7—80046—554—3/TP·039

定　价：19.50元

序　言

Clipper 是 dBASE 语言的最新编译型版本，它自问世以来，对微机数据库管理系统产生了深远影响。Clipper 是一个开发工具，它利用 dBASE III Plus 的扩充作为其标准命令集，其命令及函数是 dBASE III Plus 的超集。Clipper 的扩展系统则允许用户存取用 C 与汇编语言编制的例程和函数。除此之外，Clipper 还提供了强有力的编译预处理程序、连接程序和调试程序。

鉴于微机数据库管理信息系统已在人们的日常生活中起着日益重要的作用，而 Clipper 作为其强有力的开发工具也已在国内日益普及，为此作者特收集整理了国内外最新资料，加上自己长期使用和研究心得，编著了此书。

本书全面系统地介绍了运用 Clipper 进行程序设计的基础知识。全书共分十二章，主要内容包括：Clipper 语言及环境，Clipper 程序编译、连接和调试，数组、备注字段处理及操作，文件处理与查询，与 C 和汇编语言接口，覆盖及用户界面管理，网络应用程序等。本书在附录部分汇集了所有 Clipper 编译及连接程序错误信息和 Clipper 命令与函数。此外，作者还精心设计了大量的编程实例并将这些实例调试通过后放在软盘中，读者如有兴趣，可与作者联系。

本书不同于一般的使用手册，注重编程与实例，集知识性与技术性于一体，是一本不可多得的参考书。

本书适用于大专院校师生，培训班师生及数据库管理应用软件开发人员。

本书由北京航空航天大学 304 教研室甘登岱和计算机系木林森与彭建军主编，参加本书编写工作的还有：王建平、苏为民、章强、宋岩葵、林海、高木林、朱东、曹志林、丁建民、刘连生、苏伟。本书的录入排版工作由章群山、苏海东和刘威负责，他们为本书的出版付出了辛勤的劳动，在此对他们表示由衷的感谢。

全书由友元工作室审校，在此对友元工作室的全体同仁表示谢意。友元工作室虽成立时间尚短，但成绩已很显赫，它们开发的通用数据库生成器、经济型字幕机、彩色人像打印系统、工业控制软件等已相继推出。有关这方面的问题，读者如有兴趣的话，可直接与他们联系。他们的通信地址是：100083　北京航空航天大学 304 信箱　友元收。最后，我们预祝友元工作室今后能推出更多更好的为社会所需的产品。

尽管作者在编撰本书时已竭尽努力，但由于时间仓促，加之水平有限，不当之处在所难免，敬请读者批评指正。

编著者

1993 年 6 月

目　录

第一章　Clipper 概述

1.1　解释器与编译器

众所周知，dBASE Ⅲ和 Clipper 使用相同的语言，并把程序存在 PRG 文件中，不同之处在于 dBASE Ⅲ是个解释器，而 Clipper 是个编译器。为了解 Clipper 存在的原因及其价值，首先我们看 dBASE 如何执行一个程序，每当程序在 dBASE 中执行时，dBASE 系统会逐一将每个命令转换成计算机可执行的格式，然后再执行。这样在程序开发过程或程序经常需要修改时，是非常有弹性的。但当程序开发完成，不再需要修改时，像 dBASE 这样一次次地转换命令，就显得速度太慢，不切实际了。

为了解决 dBASE 了速度慢的缺陷，Clipper 采用了编译器的结构——一次就将所有命令转换成计算机的可执行格式，再执行。它的作法是，先将 PRG 程序转成机器可接受的中间格式——目标(OBJ)文件，再将目标文件单独或与其他目标文件(可以为 Clipper，C 或汇编语言所产生)连接成可执行文件(EXE 文件)。以后不论 dBASE 或 Clipper 是否存在于计算机中，只要计算机使用 DOS 系统，就可以执行这个 EXE 文件。

上面所谈到完成连接目的的程序称为连接器(linker)。Clipper 软盘所提供的连结器是由 Phoenix 公司提供的 Plink86 连接器。尽管 Plink86 执行稍慢，但提供了覆盖(overlay)功能。所谓覆盖功能是指，它能使比存储器大的程序也能通过执行码的转移，而在小存储器上执行。

若不需要覆盖功能也可使用其他的连接器；如 DOS 的 link，或 Borland 编译器所提供的 Tlink，……，等。它们的速度比 Plink86 快，但当使用其他连接器时，要注意它们所提供的格式不一定与 Plink86 相同。

当使用 EXE 文件时，可能会发现两件事，一是它的速度较快，另一是它非常大。速度快的原因是它利用编译器及连接器将解释器所需不断转换命令的时间完全省掉了，所以速度较快。同时，因为可执行文件是个计算机可执行的格式，人几乎无法读懂，所以它还具有一定的保密功能。

程序会因为各种需求而愈来愈大，所以不论如何管理存储器，它总有一天会不够用。因此，Plink86 利用覆盖解决了程序比存储器大的问题。在 Summer' 87 版的手册中，提供了覆盖文件。

有两种方式可以使用覆盖，一是用 OVL 文件详细地将要覆盖的模块分配好，另外是将这些模块附在 EXE 文件之后。这两种方式都可以正常运行，就算程序超过 640K，计算机也会自动在需要时输入被覆盖的部分。

接下来让我们看看产生 EXE 文件的两个步骤：编译与连接。

1.2 Clipper 的编译与连接

1.2.1 编译

最简单的编译法为:将程序名交给编译器,然后 Clipper 自动将程序内所有 DO 所调用的 PRG 文件包含进来,一起编译产生 OBJ 文件。

假设一个程序 ACCOUNTS 利用 DO 调用 SCREEN 和 REPORT 两个子程序。则

```
Clipper  ACCOUNTS
```

将像编译 ACCOUNTS 一样编译 SCREEN 和 REPORT,并且在当前目录下产生 ACCOUNTS. OBJ 文件。若 SCREEN 和 REPORT 中任何一个不存在,它会显示一个错误信息"cannot open assumed external",然后继续往下执行。

当程序调用子程序或用户自定义函数时,可以用 SET PROCEDURE TO 告诉 Clipper 到何处找它们。编译器可以自动找到并编译它们。和 dBASE 解释器不同的是,Clipper 不需要清楚地列出文件与子程序的关系。

也可以选用 Clipper 提供的参数来控制编译器的运行,详列如下:

- -m 参数指示编译器不要包含其他文件。所以当上例改成:

  ```
  Clipper  ACCOUNTS  -m
  ```

  它将产生只含 ACCOUNTS 的 OBJ 文件,可以另外编译 SCREEN 及 REPORT,然后再用连接器产生可执行文件。
- 也可以用-o 参数指示编译器将产生的 OBJ 文件放到其他子目录内。下例:

  ```
  Clipper  ACCOUNTS  -m,  -o  d:\objdir
  ```

  将 ACCOUNTS. OBJ 文件放入 d:\objdir 目录中,等待连接。
- -s 参数指示编译器只检查语法,不产生 OBJ 文件。
- Clipper 的编译器以行号显示语法错误的行,而调试器(debugger)对程序的追踪与状态显示也以行号为主。-l 参数允许用户除去 EXE 文件中的行号,如此每行可以节省三个字节。
- 为了使 Clipper 更快,它提供-t 参数,让用户能将编译器产生的暂存文件放入 RAM 磁盘中。如下:

  ```
  Clipper  ACCOUNTS  -t  d
  ```

  使用 D 磁盘暂存文件。

Clipper 也允许用户将文件名放入 CLP 的批处理文件中,一次编译许多文件。批处理文件的使用方法是在 CLP 文件前加上一个@,例如,COMP. CLP 文件包含下面两行:

```
ACCOUNTS
REPORT
```

可以用下行编译它

```
Clipper  @  COMP
```

如此将产生 COMP. OBJ 文件。

上面的编译方式看起来就像选用-m 参数一样。所以 COMP. OBJ 文件不包含 SCREEN.

PRG。Clipper 编译器在编译时，程序所使用的变量与常数如果超过一定数量，就可以使用上述方法产生不同的模块，再让连接器连接各模块。

1.2.2 连接

连接器的功能是将各 OBJ 文件组合起来，形成一个可执行的 EXE 文件。

Nantucket 公司特别为 Clipper 修改了 Plink86 的功能，使它能自动连接 Clipper. LIB。Clipper 磁盘上所提供的 Plink86 就是这个修订版。

Tlink 与 link 具有相同的格式，先列出 OBJ 文件，然后是 EXE 文件，MAP 文件最后才是程序库。当省略 EXE 文件及 MAP 文件时，它会自动有个内定值，但是程序库文件一定 要 完 全 列出，且不可省略。假设前述 accounts. obj，report. obj 及 screen. obj 都已存在，下列命令就是使用 link 产生可执行文件 acct. exe 的方法：

```
link  accounts  report  screen,acct,\clipper\clipper
```

Tlink 的格式完全与 link 相同。

若使用到扩展程序库的函数，则必须将它加到命令行中，如下所示：

```
link  accounts  report  screen,acct,\clipper\clipper\clipper\extend
```

Plink86 需要的格式与它们不同。它需要三个命令指定各种资源：1. files，详列所有的 OBJ 文件，2. library，指示编译器到何处寻找程序库，及 3. output，如果可执行文件的名称与第一个 OBJ 文件不同，则用来指示可执行文件的名称，这些命令也可缩写成 fi，lib，out。它们所提供的信息与其他两个连接器相同。

在 DOS 下使用 Plink86 有两种方法，一是直接在 DOS 提示符下打入 Plink86，它会出现＝号，提示输入 OBJ 文件，程序库及可执行文件名。另一种更常用的方式，是将 Plink86 与 OBJ 文件，程序库及可执行文件名打成一行。例如，将含有 report 及 screen 的目标文件 account 连接成可执行文件，可以打入：

```
Plink86  file  ACCOUNTS  library  EXTEND
```

它将连接 ACCOUNTS. OBJ，EXTEND. LIB 及 CLIPPER. LIB，并产生 ACCOUNTS. EXE

下一行：

```
Plink86  out  ACCT  fi  ACCOUNTS  lib  EXTEND
```

将产生 ACCT. EXE。

大部分的程序设计者会将某个特定用途的所有目标文件放在一个子目录中。因此最方便的方式是将包含 Plink86 的子目录放入 PATH 中。

Plink86 也可以认出在 PATH 中定义的文件名，它会在 DOS 的环境变量 OBJ 内所设置的目录中，找寻它所需要的目标文件及程序库。例如，下面的 DOS 命令：

```
SET  OBJ＝\OBJS;\CLIPPER
```

指示 Plink86，如果在当前目录下找不到某些 OBJ 文件，可以到\OBJ 及\CLIPPER 两个子目录中找。如此可以将目标文件及程序库放入所喜欢的子目录中。

在程序开发过程中，子程序会愈来愈多。因此，在编译及除错时，指令会愈来愈长，且愈来愈复杂。Plink86 提供一个方式，编译时可以像使用 CLP 文件一样使用连接器。它是 LNK 文件。

例如，可以建立 ACCOUNTS. LNK。它的内容如下：

```
fi   ACCOUNTS
out   ACCT
lib   EXTEND
```

然后连接时打入下述命令即可：

```
Plink86  @  ACCOUNTS
```

如果想多了解如何使用 LNK 文件及覆盖，请参考 Nantucket 公司所提供的文件。

1.3 程序库

程序库是用来存储共用程序的地方，它包含 OBJ 文件及一个指示各 OBJ 文件内有那些程序的索引。使用程序库的好处是，程序库内不被使用的 OBJ 文件，不会被放入 EXE 文件内，当一个 OBJ 文件被使用到，它才会被引入 EXE 文件中。如此，使用到某个程序库内提供的功能时，只要连接一个很小的 OBJ 文件，便可节省很多空间。

在此推荐一个程序制作的好技巧，就是使用 Microsoft 公司提供的工具 LIB 来建立自己的程序库。可以将 Clipper 编译器产生的 OBJ 文件，结合成一个 LIB 文件。

如 CLIPPER. LIB 是一个程序库，它包含了如索引、网络、排序等各种不同功能的 OBJ 文件及各文件内的相关函数。那么读者可能会问，在如此的细分下，即使如下列只有一行的程序：

```
x=1
```

为什么也会产生 160K 的可执行文件呢？为什么 Clipper 不产生一个只包含所需模块的 EXE 文件？这是一个很难回答的问题。首先，必须提示一点：这个例子是经过精心设计的。我们都知道 Clipper 是一个处理数据库的语言，所以就算是最简单的程序最少也会处理一个数据库，也就是说它必须具有数据库处理的函数，甚至必须有索引处理的函数，还有一些函数也是必须的，包括内部堆栈的处理及存储器管理。这些函数是 EXE 文件所必备的。(也就是说，Clipper 所产生的 EXE 文件必须包含一个 dBASE Ⅲ PLUS 系统)。更进一步说，如果表达式含有一个宏，编译器在执行时并无法确定它将调用那一个函数，所以必须将所有可能的函数都串入 EXE 文件中。一个实际的程序并不是像从 1 算到 10 那么简单，所以必须把大部分程序库都放进去。

毋庸置疑，Clipper 可以将可执行文件的大小设置到某个极限。譬如，可用标号指定宏不可用于函数调用，或让自定义函数是外部或内含的。但在实际应用中，程序大小并不会有大的差别。对基本所需的 160K 而言，再加入的程序码实在很小。

虽然编译器所产生的 OBJ 文件可能没有使用到程序库中所有的文件及数据，但是 Clipper 程序库中大部分的函数还是要放入 EXE 文件中。这不是和前面所述程序库的特性不同吗？Clipper 是如何完成这一点的呢？它的解决方式是，在 OBJ 文件中放入一个特殊的函数，此函数在程序库中包含了所有必须的 OBJ 文件。当然，程序库中还是有一些视需要才放入 EXE 文件的 OBJ 文件。例如，SORT，REPORT 和 LABEL FORM 就不一定要放入 EXE 文件内。

1.4 外部函数

Clipper 将 PRG 文件中没有定义的函数及子程序的名称都放入 OBJ 文件中，称为“外部函

数”。它们可能存在于其他 OBJ 文件或程序库中，而在连接时才被放入文件中。若连接时没有这些子程序，连接器会指示错误的地方。

当函数或子程序出现在宏，REPORT 或 LABEL　FORM 时，编译器便无法知道它的存在，也就无法明确地指明它是外部函数。所以在连接时连接器无法正确的将它放入 EXE 文件中。因此在执行时，如果调用这个函数就会产生执行错误。如果它存在于某个 OBJ 文件中，可以在连接时放入 EXE 文件。如果它是 LIB 文件中某个未被引用的 OBJ 文件，可以用 EXTERNAL 命令来连接它。

对宏，REPORT 或 LABEL　FORM 中未被连接的函数，可以用 EXTERNAL 将它包含到文件中。最常见的例子是 extend. lib 提供的各函数。例如，常被称为 HARDER 的 HARDER. FRM，如果程序没有使用到就必须用 EXTERNAL 声明。

1.5　使用 MAKE

Summer’ 87 版的 Clipper 比以前的版本快了很多。但是重复编译，连接整个应用程序，仍旧要花掉很多时间。大量节省时间的办法是记录下哪些程序有修改，哪些没有，然后只编译那些修改的程序。

但追踪存储哪些程序有修改，哪些没有，是件很乏味的事。所以 Summer’ 87 版提供一个能自动记载程序修改的时间与日期的工具，称为 MAKE。对于熟悉 Boarland 或 Unix 上的 MAKE 的人而言，学习 Clipper 可能有些困难，因为它们有些不同。Clipper 的 MAKE 比较类似于 Microsoft 公司所提供的 MAKE。

Make 执行的原理是利用了依存性的概念，也就是一个目标文件依存于一个特定的程序，而一个可执行文件依存于某些目标文件及程序库。用户只要将这些依存关系与处理方式放入一个文件中，Make 即会自动处理它们。

因此如果任何程序文件的修改日期比它相依的目标文件晚，它就依照用户指定的处理命令重新编译；相同的，如果目标文件或程序库的产生日期比它们产生的可执行文件晚，也一样要依指定命令重新连接。本章前面所述的例子要写成如下的 MAKE 文件：

```
accounts. obj:  accounts. prg
        clipper  accounts－m
screen. obj:  screen. prg
        cliper  screen－m
reportobj:  report. prg
        clipper  report－m
accts. exe:accounts. obj  screen. obj  report. obj
```

第一行指示 accounts. obj 依存于 accounts. prg，若 PRG 文件比 OBJ 文件还新，则执行

```
clipper  accounts－m
```

下面二个命令指示 screen. prg 与 report. prg 的依存关系与产生方式完全与 accounts. prg 相同。

最后的命令指定 acct. exe 依存于 accounts. obj，screen. obj 及 reprot. obj。若任何一个 OBJ 文件比 acct. exe 新，则执行

```
link  accounts  screen  report，acct，\clipper\clipper\clipper\extend
```

1.6 Clipper 软件工具

和其他语言所提供的精密模块一样，Clipper 也提供了一套完整且丰富的数据库处理工具。而本书及 Clipper 的目的就在协助写所需要的应用程序。

下面是 Nantucket Summer' 87 版的 Clipper 软盘中所提供的工具。

第一张软盘是 Clipper 的重心，它包含了编译器 Clipper. exe；连接器 Plink86. exe；还有调试器 debug. obj；包含 Clipper 特殊功能的程序库 extend. lib；覆盖处理程序库 overlay. lib；及一个介绍各软盘内容及特殊功能的文件 READ—ME. 1ST。

软盘 2 的大部分空间被每个应用程序都需要用到的程序库 Clipper. lib 所占据，另外还有两个非常重要的工具。一是 me. prg，它提供一个功能类似 Wordstar 编辑器的原始程序。另一个是允许用户使用 dBASE 索引的工具，ndx. obj。

软盘 3 内包含了 3 个功能很强的工具：

1. DBU：全功能数据库处理器；2. RL：报表及标识编辑器；3. SWITCH：可执行文件连接器。

软盘 4 内包含 Addendum. doc——详列使用手册没有完全介绍的新功能及许多展示 Clipper 程序技巧的有用程序、子程序与函数。

1.7 安装 CLIPPER

当拿到原版 CLIPPER 磁盘，绝不可直接用于编译或连接工作。首先，应该将 CLIPPER 拷贝到硬盘中，接着将它拷贝到一片软盘上作为备份。然后；在根目录下建立一个子目录以备存放 CLIPPER 文件，例如：下面这个指令将建立一个叫“CLIPPER”的子目录

```
C>MD  CLIPPER
```

一旦子目录建立好之后，请依照下列指令进入该子目录

```
C>CD  CLIPPER
```

现在请将 CLIPPER 系统磁盘插入 A 驱中，并请键入“A：”以转换到 A 驱上，接着请键入下列指令：

```
A>CLIPCOPY(<drive>)
```

上面指令中的(drive)是磁盘的识别字。若拿到的 CLIPPER 磁盘，没有 CLIPCOPY. BAT 的文件，则将 CLIPPER 系统磁盘插入 A 驱中，接着请键入下列指令：

```
C>COPY  A:*.*
```

上面的指令是将 A 盘中的所有文件复制到硬盘中。

1.8 CLIPPER 的系统规格

A. **数据库文件**

(1) 记录项数——最大值为 10 亿项

(2) 记录结构——顺序式，记录长度固定

(3) 记录大小——RAM(随机存取存储器)所允许的范围

(4) 记录字段数——RAM(随机存取存储器)所允许的范围

B. **字段大小**

(1) 字节段——最多 32K(1K 为 1024 字节)

(2) 日期字段——最多 8 字节

(3) 逻辑字段——最多 1 字节

(4) 附注字段——最多 64K(1K 为 1024 字节)

(5) 数值字段——最多 19 位数

C. **文件操作**

(1) 每一工作区最多可打开 15 个活动索引文件

(2) 在 DOS3.3 下最多可打开 255 个文件

(3) 每一文件没有限制程序数

(4) 每一个索引文件最多 250 个字节

D. **数值精确度**

(1) 数值精确度可达 18 位数(不包括小数点)

E. **存储器变量**

(1) 存储器变量最多可达 2048 个

(2) 存储器变量字类型可用的字节数有 64K(1K 为 1024 字节)

(3) 存储器变量数值类型可达 19 位

F. **数组**

(1) 每一个数组最多 2048 个变量均视为一个存储器变量

(2) 每一个数组最多 2048 个元素

(3) 元素大小与存储器变量相同

1.9 CLIPPER 所使用的文件

CLIPPER 文件名称分为文件名及附加名两部分。

文件名中含有字母、数字及键盘上的任何符号(连字号"—"除外),文件名一定要以字母开头,并且不得含有任何空白字。文件名最大长度为 8 个字。

数据库文件(.DBF)

数据库文件包含了数据库的记录,以及每一个字段的数据;附注字段则例外,因为它的信息存储在一个辅助文件中。

附注字段文件(.DBT)

附注(memo)字段文件存放了某一数据库文件的所有附注字段,附注字段文件与其所属的数据库文件拥有相同的文件名,但其附加名为".DBT"。

索引文件(NTX)

建立索引文件的目的是要建立数据库中记录的排列次序,同一个数据文件可同时拥有好几个索引文件。CLIPPER 的索引文件有两种附加名:"CLIPPER 格式的索引文件.NTX"或"DBASE Ⅲ PLUS 兼容的索引文件.NDX"。

存储器变量文件(.MEM)

存储器变量文件是CLIPPER程序执行过程中,将存储器变量的名称与数据值存放到磁盘时的存储处所。

标签格式文件(.LBL)

标签格式文件存放LABEL FORM命令,打印标签时所会用到的指令与定义。

报表格式文件(.FRM)

报表格式文件存放REPORT FORM命令,打印报表时需要用到的指令与定义。

格式文件(.FMT)

格式文件中存放的是SET FORMAT TO命令所将调用的屏幕画面命令。格式文件是一个标准的ASCII文本文件,因此可用文本编辑程序来产生或修改它。

文本文件(.TXT)

文本文件被用来记录操作过程中所有在屏幕上出现的信息,它是用SET ALTERNATE TO命令建立的,SET ALTERNATE ON则启动记录工作,将各命令所产生的屏幕输出结果存放到指定的文件中。

命令文件(.PRG)

命令文件中存放了CLIPPER存取文件、数据处理、屏幕处理及程序控制等各种命令,命令文件也是标准的ASCII文本文件,因此可用文本编辑程序建立或修改。

程序文件(.PRG)

程序文件中存放了一个或若干个由PROCEDURE或FUNCTION所定义的程序。除非编译时设置了-m选择项,否则只要使用了SET PROCEDURE TO命令指定了程序文件名,该程序文件就会自动纳入到指定的应用程序中,它是标准的ASCII文本文件,可用文本编辑程序修改或建立。

1.10 CLIPPER数据库文件的结构

数据库字段名称

字段名称最大长度是10个字,可由字母、数值和底线组成,但不能含有空白字符或其他诸如逗号、分号等特殊符号。字段名称的第一个字必须为字母。例如:

CUSTOMER	正确
DESCRIPTION	正确
BK_MEN	正确
NA200	正确
Y;M	不正确
F,NAME	不正确

数据库字段的类型

字符字段类型(C)

字符字段中的数据可任意含有字母、数字、空白字及特殊符号,其最大长度可达32K(1K

为 1024 字节)。

数值字段类型(N)

数值字段只供存放数字,它的数据值可供算术运算用,数值字段中最多只能容纳 19 个字,其中只有 18 个位置可供存放数字,另一个位置存放小数点。

日期字段类型(D)

日期数据字段只能存储日期数据,它可依各种格式输入和显示,其宽度为 8 个字,即使 CENTURY 设置为 ON 时也是如此,日期数据字段不但可以加减一个数值,同时日期数据字段之间也可以作加减运算。

逻辑字段类型(L)

逻辑字段自动设置为一个字的数据宽度,该项数据只为一个逻辑上的真值或假值,真值(.T.)以输入 T、t、Y、y 表示;假值(.F.)则以输入 F、f、N、n。不过实际数据库文件存储时,逻辑字段中只存放 T 或 F 这个字母。

附注字段类型(M)

附注字段中存放的是一堆文本数据;所有记录的附注数据均存放到另一个".DBT"的文件中。一开始 CLIPPER 的.DBT 文件的字段长度为 10,在真正的数据登录进来以后,系统才会计算这些字段的真实长度,其最大的数据容量为 64K(1K 为 1024 字节)。

1.11 存储器变量

存储器变量利用计算机的主存储器暂时存储数据值;除非用 SAVE TO 命令存储器变量至磁盘上,否则当执行 RETURN、RELEASE ALL 或 CLEAR MEMORY 命令之后,所有的存储器变量及其暂时存储的数据均将全部消失不见。存储器变量名称最长不可超过 10 个字,其名称可由字母、数字及底线符号组成,第一个字必须为字母,其间不能含有空白字符。

CLIPPER 存储器变量也有四种类型:字、日期、数值和逻辑;允许一次使用到 2048 个存储器变量,可使用 STORE 命令或等号(=)作为指定运算符来产生存储器变量。例如:

```
STORE  0  TO  B-F,QUANTITY,AMT
PRE-HOUSE=500
```

存储变量名称可以和数据库中的字段名称相同,不过当使用该名称时,系统会先假设要找的是数据库字段而非存储器变量。如果数据库字段与存储器变量同名而欲先执行存储器变量时,请加上别名"M",例如:

```
M->
```

全局与局部的存储器变量

存储器变量可区分为全局(PUBLIC)及局部(PRIVATE)变量两种。在程序或程序中建立的存储器变量,若不特别指明即为局部变量(PRIVATE),否则就得用 PUBLIC 命令声明其为全局变量(PUBLIC)。

当声明某变量为全局(PUBLIC)时,所有的格式及程序即可取用该变量,局部变量则只能在其声明的程序及被该程序调用的子程序中使用。当返回(RETURN)较高级的程序时,局部变量即被释放(RELEASE)。