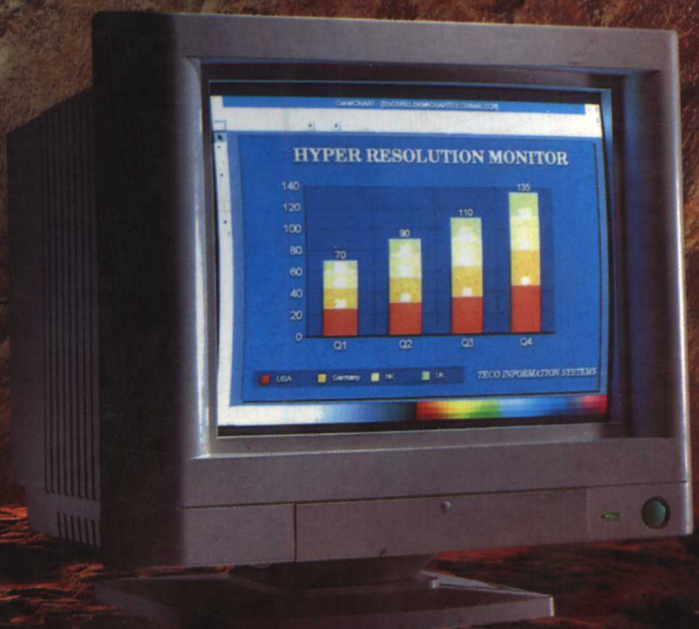
Clipper 5.0 程序设计

王 强 等编著



電子工業出版社

Clipper 5.0 程序设计

王　强等编著

电子工业出版社

（京）新登字 055 号

内 容 提 要

Clipper 是 dBASE 语言的编译型版本。本书是有关 Clipper 的一本程序设计教程。全书共十章，从各种不同角度对 Clipper 程序设计技巧作了系统介绍。此外，本书还提供了丰富的编程实例，是软件开发人员和数据库管理人员的助手和朋友。

本书可适用于大专院校师生、培训班师生及各种不同层次的软件开发人员和数据库管理人员阅读。

Clipper 5.0 程序设计

王 强等编著

责任编辑 王庆育（特约） 张丽华

*

电子工业出版社出版

电子工业出版社发行 各地新华书店经销

北京市燕山联营印刷厂印刷

*

开本：787×1092 毫米 1/16 印张：25.625 字数：626 千字

1995 年 1 月第 1 版 1995 年 1 月第 1 次印刷

印数：1-5000 册 定价：26.00 元

ISBN 7-5053-2669-4/TP・823

前　言

一九八四年，美国一群开发 dBASE 系统的软件工程师离开 Ashton Tate 公司自立门户，创建了 Nantucket 公司。随后推出了一系列兼容 dBASE Ⅲ 的编译型数据库管理系统。

Clipper 是 Nantucket 公司推出的 dBASE 最新编译型数据库管理系统，是一种开发工具。它使用 dBASE Ⅲ Plus 作为它的标准命令集。

Clipper 主要用于编制数据库的应用程序，着眼于为高级程序员提供书写大程序的工作平台和工具。

Clipper 的命令及函数是 dBASE Ⅲ Plus 的超集。Clipper 语言本身就是 Clipper 的开发结构之一，它是第一个提供用户定义函数的 dBASE 语言产品。而 Clipper 5.0 则是第一个提供用户定义命令的 dBASE 语言产品。

Clipper 的扩充系统中已加强了更易于访问 C 语言或汇编语言程序的功能，这使得用户可以随心所欲地向 Clipper 中添加自己的 C 语言或汇编语言例程和函数。例如，书写独特的界面、图形、图表、设备驱动程序等。

此外，Clipper 还包括一个强有力的，用户可自由存取的编译预处理程序和一个新的功能强大的调试程序。

本书不同于一般的使用手册，着重于程序设计技巧和实例。全书共包括十章和两个附录，从各个不同角度对应用 Clipper 进行程序设计作了系统介绍。本书含有大量的实例程序，所有这些程序作者都作了精心调试。

本书由北京航空航天大学管理学院王强主编，参加本书编写工作的还有廖彬山、刘晓林、王彬、刘志伟、林彬、林荣生、朱海东、丁为民、曾志强、强卫、连红兵、解晓东。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，在此对他们表示感谢。

由于时间仓促，不当之处在所难免，敬请读者批评指正。

编著者

1994 年 10 月

目　录

第一章　Clipper 语言基础

1.1　关于 Clipper

一九八四年，美国一群开发 dBASE 系统的软件工程师离开了 Ashton_Tate 公司自立门户，创建了同样属于开发数据库管理系统的 Nantucket 公司。随后一系列以兼容 dBASE Ⅲ 为基础的编译型数据库相继问世。

从 dBASE Ⅲ 最初上市到更新版本 dBASE Ⅳ 问世期间，尽管 Nantucket 公司一再更新其 Clipper 的版本及功能，但仍未能摆脱 dBASE Ⅲ .plus 相容编译器的阴影。直到一九九 O 年推出全新的 Clipper 5.0 版本后，才稍微改变了公司的窘况。自此以后，Nantucket 公司大大松了一口气，Clipper 5.0 新产品不断问世，使用户又恢复了对 Clipper 的信心。

Clipper 是一种开发工具，它使用 dBASE Ⅲ plus 的扩充作为它的标准命令集。它是一个没有圆点提示符或控制中心来为用户提供一个交互界面的软件环境。相反，Clipper 侧重于编制数据库的应用程序工作，着眼于为高级程序员提供书写大程序的工作平台和工具。因此，这些应用程序是从最底层开始设计并提交给用户的。程序员编写应用程序，并将它们编译、连接成可执行的文件(.EXE)，然后将它提交给用户。这是一个可直接在 DOS 操作系统下，不需要数据库管理系统支持的可执行文件，因此，对于最终用户而言，他没有必要去熟悉和掌握诸如 dBASE 的操作及管理知识，他唯一所需的就是如何使用程序本身。这就大大降低了用户的计算机能力要求，可以使用户专心致力于具体的业务工作。

Clipper 的命令及函数是 dBASE Ⅲ plus 的超集。Clipper 语言本身就是 Clipper 的开发结构之一，它是第一个提供用户定义函数的 dBASE 语言产品。而 Clipper 5.0 则是第一个提供用户定义命令的 dBASE 语言产品，因此，Clipper 程序员现在已经拥有了根据自己需要来改造 Clipper 语言的能力。就此而言，这无疑可称作是 Clipper 程序员的幸运之处，他完全可以定义自己独特风格的 Clipper 语言。

这似乎还不够好，因此，在 Clipper 的扩充系统(Clipper Extend System)中已加强了更易于访问 C 语言或汇编语言程序的功能，这使得用户可以随必所欲地向 Clipper 中添加自己的 C 语言或汇编语言例程和函数。这样，用户可以在 Clipper 环境中访问自己的 C 或汇编语言程序库。例如，自己书写独特的界面，图形、图表、设备驱动程序等。

而且，在 Clipper 5.0 以上版本中，还具有使用 dBASE Ⅲ plus 兼容文件(.DBF)之外的数据文件的能力。

除了上述这些功能外，Clipper 还包括一个强有力的，用户可自由存取的编译预处理程序和一个新的功能强大的调试程序。

所有这些功能都是为了一个目标：增强系统的功能。而且，Clipper 是由用户来决定如何使用的。已经有许多的 Clipper 程序员开始使用和喜欢这一软件。因为他们在向用户提交应用程序时无须向用户再提供一个数据库管理系统软件(如 dBASE)的副本，而且，对于用户而言也不需要再花精力去学习 dBASE，在阅读完本书之后，也许您也会高兴作为一个 Clipper 的程序员，并为此而庆幸吧！

1.2 Clipper 的编译与连接

1.2.1 解释器与编译器

众所周知，dBASE Ⅲ plus 和 Clipper 使用相同的语言，而且，Clipper 的命令与函数集是 dBASE Ⅲ plus 的超集，它们都把程序存在 PRG 文件中。不同之处在于 dBASE Ⅲ plus 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形式，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，执行效率太低而不切实际了。

为了解决 dBASE 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文件，再执行。它的作法是，先将 PRG 程序转成机器可接受的中间——文件目标(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。

完成连接功能的程序称为连接器(linker)。Clipper 软盘所提供的连接器是由 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。

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。

Plink86 利用覆盖解决了程序比存储器大的问题。在 Summer'87 以上版的手册中，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。

1.2.2 编译

最简单的编译方法为：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。

当然，也可以选用 Clipper 提供的参数来控制编译器的运行，这些参数如下：

· －m 参数　指示编译器不要包含其它文件。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 RE-

PORT，然后再用连接器产生可执行文件。

- －o 参数　指示编译器将产生的 OBJ 文件放到其它子目录内。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。
- －l 参数　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 参数允许用户删除 EXE 文件中的行号，这样每行可以节省三个字节。
- －t 参数　为了使 Clipper 更快，它提供－t 参数，让用户能将编译器产生的临时文件放入 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，一次编译多个文件。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。

1.2.3 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，它使用内定值，但是程序库文件一定要完全列出且不可省略。假如 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。

大部分程序设计者会将某些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 也可以识别在 PATH 中定义的文件名，它在 DOS 的环境变量 OBJ 所设置的目录中，寻找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\OBJS 及\Clipper 两个子目录中寻找。如此可以将目标文件及程序库放入相应的子目录中。

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
```

1.2.4 程序库

程序库是用来存储公用例程和函数的地方，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，连接时就不会被放入 EXE 文件内，当一个 OBJ 文件被使用到，它才会被装入 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。

可以使用 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，在如此细分下，即使如下列只有一行的程序：

```
x=1
```

也会产生 160K 的可执行文件。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，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。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。

1.2.5 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，而只在连接时才被装入 EXE 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 中时，编译器便无法知道它们的存在，也就无法明确地指明它们是外部函数。所以在连接时连接器无法正确的将它们放入 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 HARDER.FRM，如果程序没有使用到就必须用 EXTERNAL 声明。

1.2.6 使用 MAKE

尽管 Summer'87 以上版的 Clipper 比以前的版本执行效率提高了很多。但是重复编译，连接整个应用程序，仍然要花掉很多时间。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 以上版提供了一个能自动记载程序修改的时间与日期的工具，称为 MAKE。Clipper 的 MAKE 比较类似于 Microsoft 公司所提供的 MAKE。

Make 的执行利用了程序依赖性原理，也就是一个目标文件依赖于一个特定的程序，而一个可执行文件依赖于某些目标文件及程序库。用户只要将这些依赖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存的目标文件晚，它就依照用户指定的处理命令重新编译；同样，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
        clipper screen -m
report.obj: 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
```

1.3 Clipper 软件工具

1.3.1 Clipper 的本质

Clipper 是一个由 Nantucket 公司出版的编译程序和连接程序的混合体，是专门为 dBASE 语言应用程序而设计的软件。它建立的可执行文件能运行在 IBM PC 及其兼容机的 DOS 操作系统之上，此外，Clipper 还提供了对 dBASE Ⅲ plus 语言的一系列扩充，可以执行一些 dBASE Ⅲ plus 难于完成的任务。

当用 Clipper 语言或 dBASE 语言书写完一个应用程序之后，就可以用 Clipper 将它转换为一个能被 DOS 接受的可直接执行的独立文件，而且不必再为这些文件准备一个 dBASE Ⅲ plus 的备份。对于开发者而言，也许需要一份 dBASE Ⅲ plus 或 dBASE IV 的备份软件，以便能够为用户开发和维护数据文件和应用程序文件，但却省下了为每一个用户机器安装一个 dBASE 备份的开销。因此，开发者只需要提交一个 Clipper 应用程序，然后编译并连接它们。一个 Clipper 应用程序是完全自含的。

而且 Clipper 的开发环境也是自含的。在 Nantucket 的产品中，还提供了编写报表及创立数据文件的实用程序，对于程序员而言，他所需要添加的仅是一个好的习惯的文本编辑器。

对应用程序的源文件进行编译的另一个原因是为了源代码的保密。存放源代码的 ASCII 文件很容易打开和读出，并且易于修改。对于用户而言，他们需要一个相对稳定的应用程序系统，不希望经常变更或者不希望被人随意查阅源文件时，尤其需要进行编译。

Clipper 狭义说来是 dBASE 语言的一种版本。Clipper 编译程序接受的命令和函数是 dBASE 语言的超集。对于 Clipper 程序员而言，许多其它数据库管理系统软件才具备的新功能，如用户自定义函数、数组等，早已是运用很久很熟炼的功能。而且，Clipper 5.0 版又推出了嵌入式的预处理程序，使得用户可以建立用户自定义的命令和条件编译，以使能完全控制 Clipper 的环境。此外，它推出的一个新连接程序，能动态分配内存，使多个大程序能安全地共享内存空间。这个新的连接程序的功能之一是能够建立预连接库(Pre-Linked Libary)，PLL 是已连接好的程序代码文件，如果需要时，它可以被装入内存，而且它本身几乎可以是无限大的，仅受到存放它的磁盘空间所限。

1.3.2 Clipper 开发环境

Clipper 有许多重要的开发工具。事实上，Clipper 是非常完善的，它可以完全不用 dBASE Ⅲ plus 或 dBASE Ⅳ 就可以开发和测试应用程序，而 Clipper 的全新调试程序就更容易进行调试和错误定位。

在 Clipper 的软件包中，还包括 DBU.PRG，一个极为有用的用于建立和修改数据和索引文件的工具，DBU 很象一个缩减了的 dBASE Ⅲ plus 中的 ASSISTANT。由于 DBU 是用 Clipper 语言直接书写的，用户可以很容易在磁盘中找到并查阅它，可以了解它是如何实现其强有力的功能的。另外，DBU 还向程序员展示了一种编写 Clipper 程序时应追求的良好风格。

此外，我们在前面介绍的 Make 工具，也是 Clipper 的一个强有力工具，它可以实现对程序模块的修改追踪，提高编译的效率。

Clipper 功能的最重要特殊之一是它的开放式结构。程序员可以用 C 语言和汇编语言编写自己的例程和函数，并将它们挂在 Clipper 应用程序上，并象 Clipper 代码的一部分那样起作用。而且还可以购入其他程序员书写的 Clipper 例程和函数，并作为自己系统的一部分而使用。同时，这些例程和函数还可以再形成程序库，使编译连接时更为方便。

1.3.3 Clipper 的安装

当拿到原版 Clipper 磁盘时，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。

1.4 Clipper 的技术说明

1.4.1 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)字节字段——最多 32K(1K 为 1024 字节)；

(2)日期字段——8 字节；

(3)逻辑字段——1 字节；

(4)备注字段——最多 64K(1K 为 1024 字节)；

(5)数值字段——最多 19 位数。

C. 文件操作

(1)每一工作区最多可同时打开 15 个活动索引文件；

(2)在 DOS3.3 下最多可同时打开 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. 内存变量

(1)内存变量最多可达 2048 个；

(2)内存变量字类型可用的字节数有 64K(1K 为 1024 字节)；

(3)内存变量数值类型可达 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。

1.4.2 Clipper 的文件类型及用途

Clipper 文件名称分为文件名及扩展名两部分。

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符。文件名最大长度为 8 个字符。文件扩展名为三个字符。

A. 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;备注字段则除外,因为它的信息存储在一个备注文件中。

B. 备注字段文件(.DBT)

备注(memo)字段文件存放了某一数据库文件的所有备注字段的内容,备注字段文件与其所属的数据库文件拥有相同的文件名,但其扩展名为.DBT。

C. 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几个索引文件。Clipper 的索引文件有两种扩展名:Clipper 格式的索引文件.NTX 或 dBASE Ⅲ PLUS 兼容的索引文件.NDX。

D. 内存变量文件(.MEM)

内存变量文件是 Clipper 程序执行过程中,将内存变量的名称与数据值存放到磁盘时的存储场所。

E. 标签格式文件(.LBL)

标签格式文件存放 LABEL FORM 命令。打印标签时所需要用到的定义。

F. 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定义。

G. 格式文件(.FMT)

格式文件中存放的是 SET FORMAT TO 命令所调用的屏幕格式命令。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。

H. 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。

I. 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语句,语句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。

J. 程序文件和过程文件(.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。它是一个标准的 ASCII 文本文件,可用文本编辑程序进行建立或修改。

1.5 Clipper 的有关规定

1.5.1 Clipper 的数据库文件结构

A. 数据库字段名称

字段名称最大长度是 10 个字符，可由字母、数字和下划线组成，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符必须为字母。例如：

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

B. 数据库字段的类型

(1)字符字段类型(C)

字符字段中的数据可任意含有字母、数字、空白字符及各种特殊符号，其最大长度可达 32K(1K 为 1024 字节)。

(2)数值字段类型(N)

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

(3)日期字段类型(D)

日期数据字段用于存储日期型数据，它可依日期的各种格式进行输入和显示，其宽度为 8 个字符，即使 CENTURY 设置为 ON 时也是如此，日期数据字段不但可以与一个数值型数据进行加减运算，而且，日期数据字段之间也可以作加减运算。

(4)逻辑字段类型(L)

逻辑字段自动设置为一个字符的数据宽度，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 表示。不过实际数据库文件存储时，逻辑字段中只存放 T 或 F。

(5)备注字段类型(M)

备注字段中存放的是一系列文本数据；所有记录的备注数据均存放到另一个".DBT"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实际长度，其最大的数据容量为 64K(1K 为 1024 字节)。

1.5.2 内存变量

内存变量利用计算机的主存储器来暂时存储数据值；除非使用 SAVE TO 命令将的内存变量存入磁盘上，否则当执行 RETURN、RELEASE ALL 或 CLEAR MEMORY 命令之后，所有的内存变量及其暂时存储的数据均将全部消失。内存变量名称最长不可超过 10 个字符，其名称可由字母、数字及下划线符号组成，第一个字必须为字母，其间不能含有空白字符。

Clipper 内存变量也有四种类型：字符、日期、数值和逻辑；允许一次使用 2048 个内存变量，可使用 STORE 命令或等号(=)作为指定运算符来创建内存变量。例如：

```
STORE 0 TO B_F,QUANTITY,AMT
PRE_HOUSE=500
```

内存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→(MEMVAR)
```

A. 全局与局部的内存变量

内存变量可区分为全局(PUBLIC)及局部(PRIVATE)变量两种。在程序或程序中建立的内存变量，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，所有的格式文件及程序文件即可取用该变量，而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)。

B. 建立局部变量

在子程序中，可利用 PRIVATE 命令声明一个局部变量，而无需担心是否会不小心重新定义名称相同的变量。请参考下面的例子：在被调用程序中声明的局部变量“MEM1”，在程序执行之后并没有任何改变。

```
SET TALK OFF
MEM1="ONE"
MEM2="TWO"
DO CAPITALS                    && CALL PROCEDURE.
? MEM1                         && RESULTS:"ONE"
? MEM2                         && RESULTS:"TWO"
RETURN
PROCEDURE CAPITALS
PRIVATE MEM1                   && HIDE MEM1 ABOVE
MEM1="ONE"                     && CHANGE LOCAL MEM1 ONLY
? MEM1                         && RESULTS:"ONE"
? MEM2                         && RESULTS:"TWO"
RETURN
```

C. 内存数组变量

内存变量也可被声明为一维数组，此时整个数组即为一个内存变量，所能声明的数组数目及每个数组所能声明的元素数目仅受限于主存储器的容量。下例中的 DECLARE 命令即用来声明一个拥有 10 个元素的数组。

```
DECLARE MEMVAR [10]
```

在括弧中填入一个数目，即可访问数组中的某一元素，如下例所示：

```
FOR n=1 TO 10
? MEMVAR[n]
NEXT
```

1.5.3 表达式

在 Clipper 中，表达式由操作数和操作符构成，组成表达式操作数的可以是字段、内存变量、常数、函数等，但是组成具体的每一个表达式中的操作数应当具有相同的数据类型。

Clipper 中有四种不同类型的表达式：

- 算术表达式　在算术运算中执行各种计算工作，它可以由数值型字段、数值型内存变量、数值常数、数值型函数等组成，它的计算值一般也为数值型；
- 关系表达式　比较两个表达式的结果，并用一个逻辑真值或假值来表示这两个表达式之间的关系。关系表达式中的表达式可以是数值型、日期型、字符