HOPE

HOPE COMPUTER COMPANY LTD.

汉字dBASEⅢ与dBASEⅢPLUS
数据文件修复指南

于 达译
李 平校

本书对dBASEⅢ与dBASEⅢPLUS的数据文件结构做了详尽的分析，以及有关中文数据库文件的修复技巧和方法。

北京希望电脑公司

汉字 dBASE III／dBASE III PLUS
数据文件修复指南

于达　译
李平　校

北京希望电脑公司

一九九一年十月

前　言

dBASE III／Plus是目前在微机上应用十分广泛的关系型数据库管理系统，利用它开发的各类型管理应用系统随处可见．当您在微机上开发了一个非常复杂的dBASE III／Plus应用程序，突然有一天，在您打开数据库文件时，却显示出:"Not a dBASE Database"．这时您的心情就可想而知了．在不知dBASE数据库文件结构的情况下，等于判了这个数据库的死刑．事实上，这类损坏有时仅是第一个字节被破坏而已．了解dBASE数据文件的结构及修复方法，"Not a dBASE Database"就能起死回生．

本书详细讨论了数据文件头及数据文件本身可能出现的损坏情况及特征，并将它们分成了五种常见的损坏类型．又根据数据文件的大小，提出了各种不同的修复方法．

本书对于dBASE III／Plus的数据文件结构做了详尽的分析，如果您在使用dBASE时一直没有遇到数据被损坏的情况，那么，本书对备注域(Memo Field)的剖析，也能帮助您了解备注域结构及如何运用．

由于dBASE应用程序的国际化，在西文和其他文种的环境下修复数据文件的技巧有所不同．译者在此处仅截取了有关中文数据库文件的修复技巧．相信本书一定会对读者有很大的助益．一旦读者利用本书所提供的方法修复了一个十分重要的数据库文件，这本书的价值就真正得到了体现．

译者

1991年10月

目　录

第一部分　在西文系统环境下修复 dBASE 文件

本部分中详细的讨论 dBASE III 及 dBASE III／PLUS 的数据文件结构、数据文件损坏特征及种类、数据文件的"大文件"与"小文件"的区别，以及各类损坏修复方法。

通过仔细阅读本部分提出的概念和方法，读者能掌握常见 dBASE 数据文件的修复方法并能帮助读者自己对其他受损情况进行正确的分析和修复。

第一章　dBASE 数据文件的结构

1.1 dBASEIII / PLUS 数据文件的结构

我们建立一个各类型数据都齐全的数据文件来示范，文件名 TEST3.DBF，文件的结构如下:

```
Structure for database : B:test3.dbf
Number of data records :     3
Date of last update    : 02/12/91
Field      Field name    Type        Width     Dec
    1      NAME          Character   10
    2      AMOUNT        Numeric     8         2
    3      START         Date        8
    4      CONFIRMED     Logical     1
    5      NOTES         Memo        10
** Total **                         38
```

数据文件里有三个记录，而 MEMO 域的数据分别如下:

1. This is the memo field in the record1 of test3.dbf
2. This is the memo field in the record2 of test3.dbf

1.2 dBASE III 数据文件头结构

数据文件的前 32 个字节保存有关文件本身的数据，包括数据文件识别值、日期、记录数、文件头定义长度及每个记录长度(在本书#1 代表第 1 个字节)，分别说明如下:

#1:

数据文件识别值 03 或 83; 如果有 MEMO FIELD，将为 83。

#2～4:

数据文件最近修改日期的十六进制，顺序为年月日。

#5～8:

数据文件记录数，也就是用 Display Structrue 命令所显示 NUMBER OF RECORD 的值。

#9～10:

定义文件头(FILE HEADER)的总字节(高位字节保存低位数，低位字节保存高位数)。例如 C200，定义的文件头是 194 个字节。

#11～12:

定义一个记录的长度。

#13～32:

dBASE 保留。

从#33 到文件头的结束标记(0D)前一个字节，每 32 个字节定义一组数据域(FIELD)，包括名称、数据类型、域的长度及小数位数。我们用一组说明如下:

#1～10:

定义域的名称。

#11

dBASE 保留，值一定是 00。

#12

定义数据类型，共有'C'、'N'、'D'、'L'及'M'等类型的数据。

#13～16

dBASE 保留用。

#17

定义域(field)长度。

#18

定义小数位数。

#19～32

dBASE 保留单元。

所有数据域都定义完毕，紧跟着文件头的结束标记(0D)。在 0D 之后 dBASE 保留一个字节存 00，在 00 之后才是数据的开始。(在 0D 之后保留的字节到了 dBASEII PLUS 版本就被删除)文件头的长度随数据域定义的数目而改变(在 dBASE II 文件头的长度定为 521 字节)。所以最小的样头是 66 字节，只有一个域名；最大的文件头是 128 个域都定义，共 4130 字节。公式如下:

文件长度＝34+32＊数据域数

1.3 dBASE III／PLUS 数据文件头的结构

dBASE III／PLUS 文件的结构基本与与 dBASE／III 是相同的，另外在保留的字节处设置新的用途。文件头结构的前 32 字节分别说明如下:

#1

dBASE 数据文件识别值，03 表示没有.DBT 文件，83 表示有。

#2～4

定义日期，顺序为年月日。

#5～8

保存文件头的长度。

#11～12

定义数据录长度。

#13～15

dBASE 保留。

#16～28

网络系统用。

#29～32

dBASE 保留。

从#33 到文件头结束标记(0D)前一字节，每 32 字节定义一个数据域。包括名称、数据类型、域的定义长度、文件工作区，及网络保留用。我们举一组说明:

#1～10

定义域名称。

#11

dBASE 保留。

#12

定义数据类型:'N'、C'、'N'、'D'、'L'及'M'。

#13～16

数据域在内存的地址，与磁盘内容无关。

#17

定义数据域的长度。

#18

定义数值域小数位数。

#19～20

保留网络系统时用。

#21

文件工作区号码。

#22～23

保留网络系统时用。

#24

SET FIELDS 标志定义。

#25～32

dBASE 保留用。

所有数据域定义完毕后，紧跟着结束标记(0D)，然后在 0D 之后紧跟着数据的开始．数据之间没有任何标记隔开．文件头长度的计算公式如下:

文件头长度＝33+(32＊定义的数据域数)

1.4 dBASE III 文件结构的分析

要更加了解 dBASE 的文件结构，必须从 dBASE 外部来看，我们已经建立了一个 TEST3.DBF 的文件，将这个文件用 DEBUG.COM 装入内存(装入前先检查 DEBUG.COM 是否存在)，装入命令:

```
DEBUG TEST3.DBF
```

按 RETURN 键，装入完毕系统进入'-'提示状态．显示装入的内容用'DUMP'命令:

```
-d
```

按 RETURN 键，显示数据前 8 行的内容．要显示整个文件的内容，用命令:

```
-D 100 23 F
```

如果要显示的内容多于一个屏幕时数据往上滚动，可以用 CTRL S 停止翻滚，要继续显示只要按任一键．

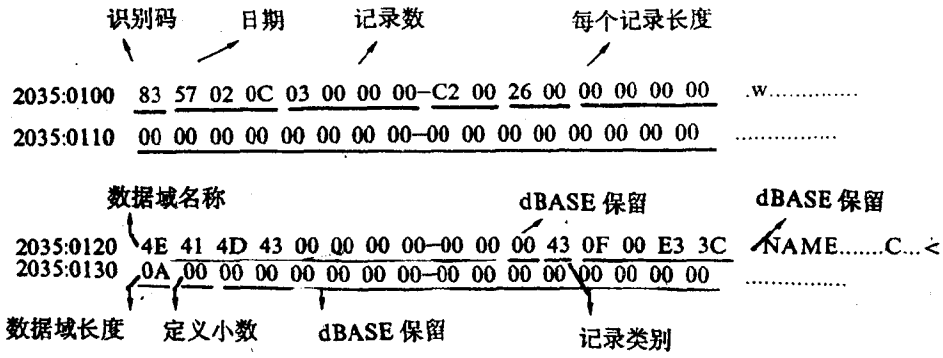
要显示整个文件内容必须知道文件装入的起始地址及结束地址．装入时没有另外声明，起始地址为 100．结束地址，可用如下的方法计算: 装入的起始地址，加上文件总数字节的十六进制值，即是文件的结束地址．

TEST3.DBF DUMP 出来的情况如下:

```
-d100,22f
2035:0100  83 57 02 0C 03 00 00 00-C2 00 26 00 00 00 00 00   .W........&.....
2035:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2035:0120  4E 41 4D 45 00 00 00 00-00 00 00 43 0F 00 E3 3C   NAME.......C...<
2035:0130  0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2035:1040  41 4D 4F 55 4E 54 00 00-00 00 00 4E 19 00 E3 3C   AMOUNT.....N...<
2035:1050  08 02 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2035:0160  53 54 41 52 54 00 00 00-00 00 00 44 21 00 E3 3C   START......D!..<
2035:0170  08 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2035:0180  43 4F 4E 46 49 52 4D 45-44 00 00 4C 29 00 E3 3C   CONFIRMED..L)..<
2035:1090  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

```
2035:01A0  4E 4F 54 45 53 00 00 00-00 00 00 4D 2A 00 E3 3C   NOTES.....M*..<
2035:01B0  0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
2035:01C0  0D 00 20 48 61 72 72 79-20 20 20 20 20 31 32 33   ...Harry     123
2035:01D0  34 35 2E 32 35 31 39 38-35 30 36 31 35 54 20 20   45.25198506151
2035:01E0  20 20 20 20 20 20 20 32-20 52 6F 67 65 72 20 20          2 Roger
2035:01F0  20 20 20 20 20 20 31 32-2E 33 34 31 39 38 35 31        12.3419851
2035:0200  32 31 32 46 20 20 20 20-20 20 20 20 20 33 20 50   212F          3 F
2035:0210  68 69 6C 20 20 20 20 20-20 20 20 20 20 30 2E 32   hil           0.2
2035:0220  35 31 39 38 35 31 32 31-35 54 20 20 20 20 20 20   5198512151T
```

关于数据文件的日期、记录数、每个记录长度、文件头所占字节数，都在前32个字节定义；而后每32字节定义一个数据域。在dBASE所有数据域定义完毕紧跟着0D 00，表示文件头结束标记(dBASE III PLUS的结束标记是0D)，分析如下：



定义 AMOUNT

```
2035:0140  41 4D 4F 35 4E 54 00 00-00 00 00 4E 19 00 E3 3C   AMOUNT.....N...<
2035:0150  08 02 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

定义 START

```
2035:0160  53 54 41 52 54 00 00 00-00 00 00 44 21 00 E3 3C   START.....D!..<
2035:0170  08 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

定义 CONFIRMED

```
2035:0180   43 4F 4E 46 49 52 4D 45-44 00 00 40 29 00 F9 3C    CONFIRMED..4D..<
2035:0190   01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

定义NOTES

```
2035:01A0   4E 4F 54 45 53 00 00 00-00 00 00 4D 2A 00 13 3C    NOTES.....MD..<
2035:01B0   0A 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
```

最后紧跟的二个字节是文件头的结束标志 。

```
2035:01C0   0D 00
```

数据的开始，紧跟在文件头结束标记 0D00 之后(在 dBASE II PLUS 只有 0D)开始字节值是 20 或 2A，记录该个记录的删除与否状态。2A 即 ASCII 字符的 * 号，表示删除在每个记录之间没有任何分隔的标识字节。数据结束有一个'1A'的结束标记，dBASE 就不理会'1A'之后的数据。我们将 TEST3.dbf 的数据用 DEBUG 装入后显示，每个记录的范围用线框起来，便于阅读。如下:

第一个数据记录

```
2035:01C0  0D 00 20 48 61 72 72 79-20 20 20 20 20 31 32 33   .. Harry      123
2035:01D0  34 35 2E 32 35 31 39 38-35 30 36 31 35 54 20 20   45.25198506154
2035:01E0  20 20 20 20 20 20 20 32-20 52 6F 67 65 72 20 20          2 Roger
2035:01F0  20 20 20 20 20 20 31 32-2E 33 34 31 39 38 35 31        12.3419851
2035:0200  32 31 32 46 20 20 20 20-20 20 20 20 20 33 20 50   212F          3 P
2035:0210  68 69 6C 20 20 20 20 20-20 20 20 20 20 30 2E 32   hil           0.2
2035:0220  35 31 39 38 35 31 32 31-35 54 20 20 20 20 20 20   5198512154
2035:0230  20 20 20 34 1A
```

#0100

83 代表本数据文件有 MEMO 域。若沿有 MEMO 域，第一字节的值是 03。

#101～103

文件的时间，它的顺序是年月日的十六进制表示法，转成十进制是 850825.

#104～107

03 表示本数据文件有三个记录.

#108～109

C2 00 是文件头所占的字节换算十进制即 194.

#10A～10B

26 00 是每个记录的长度.

#10C～11F

dBASE 保留用共 20 字节.

#120～129

4E 41 40 45 00 00 00 00 00 00 定义域名 NAME. 没有定义的字节补 00.

#12A

dBASE 保留字节，值一定为 00.

#12B

43 即 ASCII 字符'C'，表示文字型数据.

#130

0A 即十进制 11 表示 NAME 数据域定义的长度.

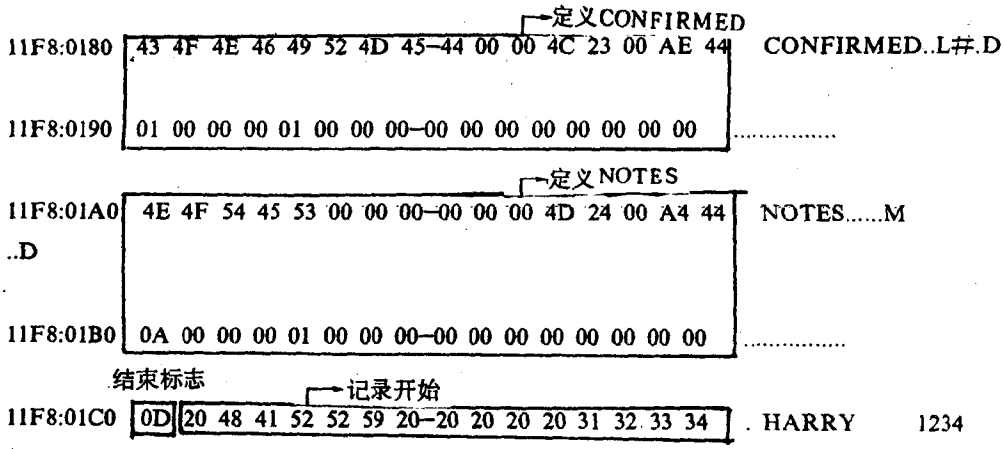
#13A

00 表示没有小数.

1.5 dBASE Ⅲ / PLUS 文件结构的分析

我们在 PLUS 系统下建立一个同样结构及数据的文件 TEST3.DBF1，然后用 DEBUG 装入内存. 显示的结果如下:

```
11F8:0100  83 50 01 01 02 00 00 00-C1 00 26 00 00 00 00 00   .P........ ..
           识别码 日期     记录数  文件头长度 每个记录长度
11F8:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
                                              ┌ 记录类型
11F8:0120  4E 41 4D 45 00 00 00 00-00 00 00 43 09 00 AE 44   NAME.....C...D
                域名称
11F8:0130  0A 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00   ................
           域长度          └ 文件工作区
                                        ┌ 定义 AMOUNT 域
11F8:0140  41 4D 4F 55 4E 54 00 00-00 00 00 4E 13 00 AE 40   AMOUNT.....N...D
11F8:0150  08 02 00 00 01 00 00 00-00 00 00 00 00 00 00 00   ................
                                        ┌ 定义 START
11F8:0160  53 54 41 52 54 00 00 00-00 00 00 44 1B 00 AE 44   START.....D...D
11F8:0170  08 00 00 00 01 00 00 00-00 00 00 00 00 00 00 00   ................
```



很明显的 dBASE III / PLUS 的结构，除了定义新的功能外，与 dBASE III 相同。所以在 PLUS 的环境下，执行 dBASE III 程序，不会有问题。但在 dBASE III 环境下，执行 dBASE III / PLUS 程序则会有问题。

1.6 dBASE III / PLUS 备注域及.DBT 文件

dBASE III / PLUS 数据文件附有一个备注域(MEMO)，提供很方便的记录数据的功能，不需要任何数据格式的限制，也不受域长度固定的困扰。这个备注域不同于其他定义的数据域。它的数据并不存在数据文件本身，在打开文件时 dBASE 如果发现有备注域，会自动打开域(MEMO)在数据文件中保存的只是所存数据的地址指针，所以固定是 10 个字节的长度。

备注域的 10 个字节的长度仅保存备注域的数据在 DBT 文件的位置，保存的方式如何呢?

保存的方式是以 ASCII 的十六进制值，且高位数存高字节，低位数存低字节。没有存的字节补 00。

我们举个例子说明备注域的值保存在 DBF 文件的方式，如 32:

20 20 20 20 20 20 20 20 33 32

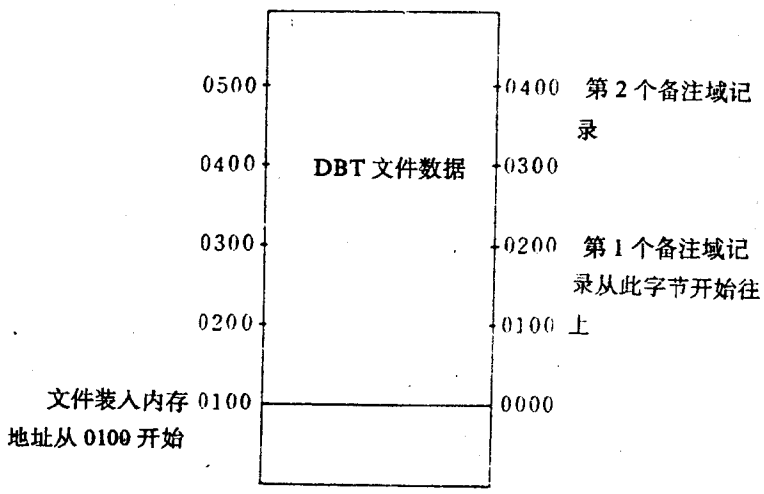
同样的如果是 127，如下:

20 20 20 20 20 20 20 31 32 37

而如果是 0，那么就全部补 20。

当你一开始保存数据到备注域，dBASE 马上预留一块 512 字节的空间，即使你加入的数据仅几个字节，还是占用 512 字节的空间。数据一超过 512 字节，dBASE 马上又预留另一个 512 字节的空间来保存数据，这种阶梯式的增加方式，有时的确浪费磁盘空间。

备注域的数据在 DBT 文件里从第 513 字节开始存数据。而文件的装入地址是从 0100 开始。用图解表示如下:



用十进制观点看这个储存方法会觉得怪异，但从十六进制观点看是合理的。第一个备注域数据从 513 字节开始，结束的字节在 03FF，第二个是从 0400 开始，所以往后推算任何一个备注域的起始位置都是 0200(HEX)的倍数。在内存上的相对地址可用一公式表示如下:

$$\text{DBT 文件数据地址} = 300+(N-1)\times 200$$

N 是备注域数据在 DBT 文件的地址指针，要换算成十六进制。例如:20 20 20 20 20 20 20 20 31 32，是存在 DBF 文件的备注域的 N 值，换算成十六进制数是 0C，把 N 的值带入公式，算出的结果 1900，即是该备注域在 DBT 文件的地址。

知道备注地址指针及数据储存的方式，在修复受损的数据文件时会很有帮助的。如果是备注域的地址指针被损坏，我们可以从 DBT 文件找出该数据的地址，用上述的公式找出 N 值，重建备注地址指针。

更重要的是了解备注域的结构，便于灵活运用备注域的数据。假设你要在不同记录用同一个备注域的数据，又不想重输入数据，只要在 DBF 备注域里设置相同备注域的数据指针地址即可。使用的工具参照本书大小文件的修理工具用法。

在结束本章前，提到最后一点是 dBASE 的 DBT 文件对磁盘空间利用的缺点。当你进入备注域修改数据，dBASE 在 DBT 文件的最上面位置划定新的地址来保存修改过的数据，但并没有删除原来的数据，这种做法是牺牲磁盘空间来换取执行的速度。

第二章　数据文件的分类

2.1 所谓"大文件"与"小文件"的区别

在修复所谓的"大文件""小文件"数据文件，并不是数据在磁盘上所占的大小，而是决定于你硬件设备内存可用的空间有多少．这是相对性的说法，而不是绝对性的定义．

- 可以将整个数据文件用DEBUG装入内存的文件就是"小文件"．
- 整个数据文件无法用DEBUG装入内存的文件就是"大文件"．

我们用 16 位的机器在 PC-DOS 下，判断文件大小的因素如下:

- 硬件设备本身具备的内存多少?
- 是否装入其他的软件如SIDEKICK、SUPERKEY.....等．
- 是否设置了RAM DISK．

由于这三个可变因素的存在，判断文件的大小，只有根据实际情况来判断．

假设你的系统是 640K，没有装入其它的软件，也没有设置 RAM DISK．操作系统占用 300K，剩下 340K 可利用，再扣除 DEBUG 占用的 12K，那么内存可利用的只有 328K．小于 328K 的数据都可装入就是"小文件"．大于 328K 的数据就是"大文件"．

我们修复文件以"大文件"、"小文件"区别修复的步骤．对"小文件"的修复，如果文件大于 64K，除了按照"小文件修复所用的方法外，必须参考修复大于 64K 数据文件应该注意的事项．

"大文件"的修复不容易，需要其他软件辅助，测知数据在磁盘的分布，修改损坏数据又要存回原来的位置，如果对于 DEBUG 应用不熟，可能还会损坏其他的数据．在本书采用容易操作，不容易错的修复方法．在附录里有三个 BASIC 程序，分别为 FILER.BAS、SALVAGE.BAS 及 SEARCH.BAS．我们用这三个程序作辅助以修复大文件．它们的功能分别说明如下:

- FILTER.BAS程序消除dBASE数据文件的结束标记(1A)，并用'20'取代，在修复前都必须执行这个程序．
- SALVAGE.BAS程序将受损文件中的完整数据搬到另一个没有格式的正文文件中．
- SEARCH.BAS程序找寻字符串在文件中的位置．

第三章　受损数据文件的分类

3.1 受损文件特征及分类

受损数据文件依严重程度不同，我们区分为五类，最轻微的为第一类，最严重的为第五类。

我们所谓的数据文件受损修复，并不包括磁盘片物理上的损坏，例如变形等。另外，如果数据被其他的数据覆盖了，也不是这里的修复范围。我们要谈的是数据文件里有ASCII 或非 ASCII 字符的非法侵入，造成数据文件无法正常的使用，如何将这些混入的破坏字符删除。混入的情况有区域的不同及混入字符的多少、分布的区域多广，也就是受损件级别的划分依据。

受损文件的等级:

- **第一类损坏**

文件头(FILE HEADER)完整，有不合法的字符混入好的数据，原来数据的空间，并没有被占用。不合法字符可能是 ASCII 或非 ASCII 的字符，有的是控制字符，有的是可打印字符。

- **第二类损坏**

结束标记混入数据里，使部分数据无法显示出来，好像遗失一样。

- **第三类损坏**

文件头(FILE HEADER)全部或部分被掩盖，但其位置没有被数据占用，数据本身完整无缺。

- **第四类损坏**

文件头完整，但不合法的字符混入数据里，受损文件数据其空间被占用。

- **第五类损坏**

文件头部分或全部损坏，数据也有受损情况散布在数据里，文件头原来的位置及部分损坏数据的位置被占用，这是最严重的损坏，只差数据没有全部遗失。

3.2 数据损坏诊断表

事实上判断受损等级是不容易的事，当你发现数据有异常的现象，先查阅诊断表，选择相近的异常特征，然后按照修复方法进行修复。但必须记住一件事：开始修复前一定要拷贝备份。即使诊断错误或是修复失败，还可以从头再来，不致于有大损失。

dBASE 有时候会自动显示受损特征。如果数据文件头的第一个字符不是 03 或 83，就会有"NOT A dBASE·FILE"的错误信息，同时拒绝打开文件。但并不是所有受损情况都可以自动显示。当结束标记混入数据里，dBASE 并不会显示任何错误信息，必须靠你自己发现。

有多种的损坏混杂在一起，诊断时只能发现最严重的损坏，待最严重的损坏修复后，较轻的损坏才会出现。