一、基本数据分布

可以看出，FAT1和FAT2紧紧接着引导扇区，每个FAT占9扇区，根目录又紧随其后，从第19扇区开始，但根目录区的大小则不固定，依赖于Directory Entry的数目，最多有BPB_RootEntCnt个。

每个Directory Entry占32字节，其结构如下：

名称	开始字节	长度	内容
DIR_Name	0	0xB	文件名8字节，扩展名3字节
DIR_Attr	0xB	1	文件属性
保留位	0xC	10	保留
DIR_WrtTime	0x16	2	最后一次写入的时间
DIR_WrtDate	0x18	2	最后一次写入的日期
DIR_FstClus	0x1A	2	此文件在数据区和FAT表中的开始簇号
DIR_FileSize	0x1C	4	文件大小

DIR_FstClus：对应了文件的第一个簇号，但需要注意的是：数据区的开始簇号是2，不是0和1，为此，FAT中的前两个FAT项（0项和1项）不被使用，从第2个FAT项开始有效。

每个FAT项（FATEntry）长度位12Bit，即一个半字节，FAT表中每3个字节存放了2个FATEntry。

3 Bytes：

FATEntry代表文件的下一个簇号，但如果其值大于等于0xFF8，则表示当前簇是文件的最后一个簇；如果是0xFF7，则表示这是一个坏簇。

上面已经提到，FAT中第一个有效的FATEntry是2号，对应了数据区的开始簇号。

－－

可以这样理解：FATEntry就是数据区对应簇的next字段，它使一个文件以链表结构存放在数据区各个不连续的簇中，而把“索引”放在FAT中。

主要介绍以3.5英寸的1.44M标准格式化的FAT12文件系统的软盘为介绍对象。这里强调那么多是因为：1.44M的软盘格式化可以不是1.44M，可以大于也可以小于；格式化的文件系统也可以不是FAT12。

为什么会出现正常的1.44M软盘格式化后可大可小的情况呢？从软盘及软盘驱动器原理出发，软盘的寻址方式（可以认为是读取数据的方式）是：CHS，C = Cylinder（柱面）,H = Header（磁头）,S = Sector（扇区）。标准地格式化后，磁盘将被格式化为 每面80磁道（80个同心圆，柱面），每个磁道有18个扇区，每个扇区是 512字节，那么高密3.5英寸软盘的容量为：2×80×18×512 ＝ 1474560 Byte = 1440 KB = 1.44 MB。然而，软盘可以不格式为80磁道，每个磁道也可以不是18扇区，这是题外话，如果您有兴趣，可以用古老的HDCopy试试。

文件存储到磁盘上时至少要占用1个扇区，即使这个文件只有1个字节，如果文件有513字节，那就得占用2个扇区，下一个文件就不能用这只使用了一个字节的扇区。即软盘以扇区为单位存储文件。现在用下面的假设来说明本文的目的：

假设只有18个扇区的磁盘，以 0 - 17 编址，如果一个文件保存在 1 - 6扇区，另一文件保存在 7 - 16扇区，如果我们对第一个文件增加了内容，又需要一个扇区来保存它，但由于文件连续存储， 7号扇区是第二个文件的，我们当然不能用它，只有最后留有一个扇区可用，我们会不会把第二个文件先挪到8-17扇区以腾出一个扇区来给第

一个文件使用呢？当只有少数两个文件的时候可以，但有很多文件的时候会变得麻烦起来。如果我们用一个表来表示有一个文件占用了 1-6扇区 和 17扇区，那事情就简单了——我们不必为文件不连续而烦恼。这个表就叫它：文件分配表(File Allocation Table)。那怎样才能知道这个文件存储的文件名和文件存放的起始扇区？再建一个表，用于存放文件名、起始扇区、文件创建时间、文件实际大小等等资料，这个表叫：文件目录表(File Directory Table)。将这两个表放在磁盘指定的位置，以便操作系统使用，磁盘的其它扇区全都用来存放文件的实际内容，这就构成了有文件系统的磁盘。

磁盘上，0面0磁道第1扇区用于存放引导程序，如果这512字节最后两个字节分别是0x55,0xAA（一个字是0xAA55)，称为可引导标志，BIOS会将这512字节读取出来执行，操作系统便是利用这里来实现引导的。标识软盘是不是FAT12并不是没有根据的，在这512字节中，还有一个设备头用于标识这个软盘（设备），例子如下：

```
;==================================================================================
; 程序执行的第一条指令必须是跳转（如果你想使用FAT12这类文件系统的磁盘）
; 必须占用3字节
;==================================================================================
jmp SHORT main ; 2 bits，跳转到主程序执行
nop            ; 1 bit
;==================================================================================
; FAT12 文件系统头，从NYAOS 借过来的，可以参考相关的文档以获得更多细节
; 这个块会让 Winimage 认出编译后的二进制文件为有效的引导文件
; 如果不使用这个块，Winimage将不会将其作为引导程序处理
; 但我们可以借助其它方法和工具处理，比如DEBUG
;==================================================================================
bsOEM       db "ExOS0.02"             ; OEM String,任意你喜欢的8字节ASCII码
bsSectSize  dw 512                    ; Bytes per sector
bsClustSize db 1                      ; Sectors per cluster
bsRessect   dw 1                      ; # of reserved sectors
bsFatCnt    db 2                      ; # of fat copies
bsRootSize  dw 224                    ; size of root directory
bsTotalSect dw 2880                   ; total # of sectors if < 32 meg
bsMedia     db 0xF0                   ; Media Descriptor
bsFatSize   dw 9                      ; Size of each FAT
bsTrackSect dw 18                     ; Sectors per track
bsHeadCnt   dw 2                      ; number of read-write heads
bsHidenSect dd 0                      ; number of hidden sectors
bsHugeSect  dd 0                      ; if bsTotalSect is 0 this value is
                                      ; the number of sectors
bsBootDrv   db 0                      ; holds drive that the bs came from
bsReserv    db 0                      ; not used for anything
bsBootSign  db 29h                    ; boot signature 29h
bsVolID     dd 0                      ; Disk volume ID also used for temp
                                      ; sector # / # sectors to load
bsVoLabel   db "NO NAME    "          ; Volume Label
bsFSType    db "FAT12   "             ; File System type <- FAT 12文件系统
;==================================================================================
; Main start here
;==================================================================================
main:
#至于如何引导计算机，可以参考我Blog里的more.asp?name=xemean&id=2
```

0面0道第2扇区到第10扇区的9个扇区是FAT表的存放位置，为了预防，0面0道的第11扇区到1面0道第1扇区的9个扇区是第2个FAT表的存放位置，这第2个FAT是备用的，当第一个FAT出了问题里，可以用第2个FAT。1面0道的第2扇区起到1面0道的第15扇区（共14个扇区）用于存放 FDT。FDT没有备份，所以没有第二个FDT。这里要注意的是，磁盘为了读写的速度，0面0道的18个扇区接下来的是 1面0道的扇区，而不是0面1道，因为0面0道跟1面0道同在一个柱面上（同心圆），只是用的磁头不同。

FAT12 中，每个文件分配表项只占12位(bit)，即1.5字节(byte)，每个表项代表一个扇区，在这里，磁盘只有扇区的概念，磁盘里所有扇区都被类似于上一段提到的磁盘读写方式线性地编址（LBA），不再有CHS。这里还要提一提簇的概念：DOS会把2个扇区作为一簇，那么文件就要以簇为单位读写。簇的大小通常根据磁盘的大小设定，以尽可能少浪费磁盘空间为本。

FAT12每个表项的值指出文件存放的下一个扇区号，同时也是表项入口。比如如果文件的存放的第一个扇区是002，那系统首先找FAT的002，在002处得到一个值003，表示文件下一个扇区是003号，再接着003表项找，得到006...，表项的值含义如下：
000 - 此簇未用；FF8 - FFF 该簇为文件的最后一簇；FF0 - FF7，此簇为坏，不可用；其它值表示文件下一簇的簇号。
下面的图来说明FAT的基本原理：
表项编号 值(16位) 备注
000 | FF0 | <- 000 项 001项为表头，1字节 0xF0表示存储介质
001 | FFF | <- 2、3字节为 0xFFFF ，固定值，FAT标志
002 | 003 | <- 文件下一簇为003
003 | 005 | <- 下一簇：005
004 | FF7 | <- 坏簇，不可用
005 | 011 | <- 下一簇：011
.....
011 | FF8 | <- 该文件结束
012 | 000 | <- 可用簇
.....
根据上表，我们可以知道，一个文件占用了 002,003,005,011 这4个簇。
簇号 + 31 = 逻辑扇区号 //// 31 ＝ 保留扇区数 + 隐藏扇区数 + FAT数×每个FAT所占扇区数 + FDT所占扇区数 - 2 = 1 + 0 + 2*18 + 14 -2
LBA = 逻辑扇区号 - 1
扇区 = (LBA MOD 每道扇区数) + 1
磁道 = (LBA / 每道扇区数) / 磁头数
磁头 = (LBA / 每道扇区数) MOD 磁头数

根据上面的公式，得到以下计算值：
002: S = (32 MOD 18) + 1 = 15
002: C = (32 / 18) / 2 = 0
002: H = (32 / 18) MOD 2 = 1

011: S = ((11+31-1) MOD 18) + 1 = 6
011: C = ((11+31-1) / 18) / 2 = 1
011: H = ((11+31-1) / 18) MOD 2 = 0

就此，我们已经可心根据簇号得到物理CHS了，那怎样才能得到一个文件的关系首簇号呢？前面我们提到了FDT。下面说说FDT的结构：
每个FDT项占32字节，分配如下：
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
0 - 7 ： 8字节，文件名
8 - 10： 3字节，文件扩展名

11： 1字节，文件的属性
12 - 15：4字节，保留
16 - 21：6字节，保留
22 - 23：2字节，文件最后修改时间（时分秒，5:6:5）
24 - 25：2字节，文件最后修改日期（年月日，7:4:5，年取0-119对应 1980 - 2099）
26 - 27：2字节，文件首簇号，我们可以根据这个值在FAT中找到文件的存储位置
28 - 31：4字节，文件的长度，以字节为单位
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
0 - 7 文件名含义：0 - 目录项为空，可用；E5 - 此文件已经被删除
7 - 10： 文件名和扩展名为8.3格式，如果不够，必需用空格填充，即文件名如果只有6个字节，那剩下的2个字节必须以空格填充。文件名和扩展名都是大写。
11属性字节含义：00 - 普通文件；01 - 只读；02 - 隐藏；04 - 系统文件；10(1x) - 该文件是目录。

就此，本文已经基本介绍完了软盘的结构，下面介绍如何读一个文件：

给出一个文件名，比如“KERNEL.SYS”
将文件名扩展为“KERNEL SYS”，即去掉点并为文件名和扩展名补充空格
读FDT到内存中（用BIOS INT 13）
在FDT中查找到符合的文件名
可选，判断在FDT中找到的是否是目录
在符合的FDT中取出文件首簇号
读入FAT，可以选择读入两个FAT表，以检查是否有效
将簇号转换为CHS，将扇区读入内存
根据簇号在FAT中查找下一簇，并判断是否是文件最后一簇
如果是文件最后一簇，则文件读取完毕；如果不是，则转第8步
如果在引导程序中读指定的内核文件，则可以省略1、2步，直接给出内核文件名即可。

如果给出的文件是带目录的，那这里还有必要介绍一下：实际上，目录也是一个文件，只不过这个文件是一个FDT，FDT指出该目录下其它文件或目录。因此，给出如下路径：/EXOS/KERNEL.BIN ，则先是在根目录中，将“EXOS ”这个“文件”读出来，然后在读出的FDT中找“KERNEL BIN”。

fat12 文件系统 源代码分析

```
/* The FAT file system is difficult to trace through FAT table. */
/* There are two kinds of FAT's, 12 bit and 16 bit. The 16 bit */
/* FAT is the easiest, since it is nothing more than a series   */
/* of UWORD's. The 12 bit FAT is difficult, because it packs 3 */
/* FAT entries into two BYTE's. These are packed as follows:    */
/*                                          */
/*     0x0003 0x0004 0x0005 0x0006 0x0007 0x0008 0x0009 0x0010...   */
/*                                          */
/*     are packed as                              */
/*                                          */
/*     0x03 0x40 0x00 0x05 0x60 0x00 0x07 0x80 0x00 0x09 0x00 0x01...   */
```

```
/*请注意上面的0x0009和0x0010的fat12的表示     */
/*                                          */
/*     12 bytes are compressed to 9 bytes                */
/* (fat文件系统是很难遍历的，有两种fat，一个是12位的，一个是16位的*/
/* 16位的是很简单的，但是12位的是不同的，因为它把3个字节压缩到两个字节里了，）*/
/* 而且数据是有交叉的，第二个字节的高4位是下个簇号的低四位，第二个字节的*/
/* 第四位是第一个字节的高四位,但是仍旧是12个位表达一个文件目录*/

/*假设就是fat12文件系统，这时调用为
*
* find_free_fat()-->next_cluster(dpbp,idx)->link_fat(dpbp,idx,READ_CLUSTER)
*
* idx是当前簇索引值，每次都是以一递增,是簇号索引。
*   struct dpb *dpbp其实就是内部的驱动器的参数表，就是在每个盘最前扇区
*  的内容，来对这个驱动盘进行描述的内容结构体。
*
*
* 有个规定：簇是从2开始用的，前两个没有用，保留。
*
* */
CLUSTER link_fat(struct dpb FAR * dpbp, CLUSTER Cluster1,
            REG CLUSTER Cluster2)
{
struct buffer FAR *bp;
unsigned idx;
unsigned secdiv;
unsigned char wasfree = 0;
CLUSTER clussec = Cluster1/*这么多簇有多少扇区*/;
CLUSTER max_cluster = dpbp->dpb_size;/*最大簇号*/

#ifdef WITHFAT32
if (ISFAT32(dpbp))
   max_cluster = dpbp->dpb_xsize;
#endif

/*当前索引的簇值肯定是要在当前盘的空间内的，如果超过了（包括两边），就返回，并报错*/
if (clussec <= 1 || clussec > max_cluster)
{
   put_string("run CHKDSK: trying to access invalid cluster 0x");
#ifdef WITHFAT32
   put_unsigned((unsigned)(clussec >> 16), 16, 4);
#endif
/*put_unsigned()函数是将一个十进制的字符串数转化成为2进制或者是16进制的表示方法，非常巧妙*/
   put_unsigned((unsigned)(clussec & 0xffffu), 16, 4);
   put_console('/n');
   return 1;
}

/*dpb->dpb_secsize是当前驱动盘的扇区大小,为什么要用secdiv做变量我还不是很清楚,可能是secctor divide
*因为下面做除法了
* */
secdiv = dpbp->dpb_secsize;
if (ISFAT12(dpbp))
```

```
{
/*这好像要看官方文档，看 fat12的簇大小是多少的,一个簇＝一个扇区＝512字节，*/
/*
* 这里一个2，一个3，注意的是1.5＝3／2，就是一个fat要占用的字节数，
* 我突然想到baidu知道上不知道是谁说fat12的簇含有4个扇区!!!!!!!
* */
   clussec = (unsigned)clussec * 3;
   secdiv *= 2;
}
/*暂且只讨论fat12*/
else /* FAT16 or FAT32 */
{
   secdiv /= 2;
#ifdef WITHFAT32
   if (ISFAT32(dpbp))
    secdiv /= 2;
#endif
}

/* idx is a pointer to an index which is the nibble offset of the FAT
    entry within the sector for FAT12, or word offset for FAT16, or
    dword offset for FAT32 */
/*我认为下面的代码就是计算偏移，化整为零*/
idx = (unsigned)(clussec % secdiv);/*没有构成一个扇区*/
clussec /= secdiv;/*还有多少个整数扇区*/
/*曾经做过一个计算题，好像fat表项可以是很大的，可以有几兆的*/
clussec += dpbp->dpb_fatstrt;/*绝对扇区地址*/
#ifdef WITHFAT32
if (ISFAT32(dpbp) && (dpbp->dpb_xflags & FAT_NO_MIRRORING))
{
   /* we must modify the active fat,
     it's number is in the 0-3 bits of dpb_xflags */
   clussec += (dpbp->dpb_xflags & 0xf) * dpbp->dpb_xfatsize;
}
#endif

/* Get the block that this cluster is in            */
/*读一个block到一个缓冲区，返回的是缓冲区的首地址,但是比较特殊的是*/
  /* 它返回的是一个512字节长度的fat表项 */
bp = getFATblock(dpbp, clussec);

/*没有读成功，自然要返回,也就是读fat表项没有成功，可能是别的错误之类的*/
if (bp == NULL)
   return 1; /* the only error code possible here （他是说这是这个函数里面唯一的错误信息,不过好像确实是的)*/

/*针对fat12特别处理，可以看到下面每种文件系统都做了处理，我在想，如果我要加入ntfs怎么办？ */
if (ISFAT12(dpbp))
{
   REG UBYTE FAR *fbp0, FAR * fbp1;
   struct buffer FAR * bp1;
   unsigned cluster, cluster2;

   /* form an index so that we can read the block as a    */
```

```
   /* byte array                              */
/*idx上面操作得到的在簇里面的第几个扇区
* fat12文件系统一个簇＝一个扇区
*
* 第n个fat目录项:
* n为偶数:低四位3*n/2，高四位3*n/2+1
* n为奇数:低四位
* 一个簇＝？？？
* 一个扇区＝512字节
* 一个fat表项＝12位＝1.5位
*
* idx是在原有的簇的基础上还要移动几个扇区的索引
* 一个buffer的大小就是一个扇区的大小
* */
   idx /= 2;

   /* Test to see if the cluster straddles the block. If   */
   /* it does, get the next block and use both to form the */
   /* the FAT word. Otherwise, just point to the next      */
   /* block.(测试这个簇是否跨越了这个块，如果是这样的话，得到下个块，
* 用这两个簇形成一个fat字)                                 */

/*sizeof(b_buffer)=512B,PRI(&)>PRI(->)*/
   fbp0 = &bp->b_buffer[idx];

   /* pointer to next byte, will be overwritten, if not valid */
/*如果是跨越的话，就是越界的访问了吧,我的意思没有意思的访问*/
   fbp1 = fbp0 + 1;

/*如果这时候是跨越了的话，那么就要读下一个块，那么fbp1是无效的，而是需要重新赋值*/
/*因为是必须要读2个字节的，idx只是开始字节地址*/
   if (idx >= (unsigned)dpbp->dpb_secsize - 1)
   {
    /* blockio.c LRU logic ensures that bp != bp1 (这话我还没懂，希望懂的人能解释下)*/
    bp1 = getFATblock(dpbp, (unsigned)clussec + 1);
    if (bp1 == 0)
      return 1; /* the only error code possible here */

/*我看到现在只要看过READ_CLUSTER*/
     if ((unsigned)Cluster2 != READ_CLUSTER)
       bp1->b_flag |= BFR_DIRTY | BFR_VALID;

  /*fbp1指向下个byte*/
    fbp1 = &bp1->b_buffer[0];
   }

/*fbp1只是unsigned char阿？总共也就8位的，左移8位还有吗？？*/
/*高高低低原则*/
   cluster = *fbp0 | (*fbp1 << 8);
   {
    unsigned res = cluster;

    /* Now to unpack the contents of the FAT entry. Odd and */
    /* even bytes are packed differently.                   */
```

```
/*如果要寻找的idx是奇数的话，则要*/
    if (Cluster1 & 0x01)
      cluster >>= 4;
    cluster &= 0x0fff;

/*因为我只要看获取一个空闲fat，所以我就看到了这里
*
* 当fat项是0时候，说明是空闲的
*
* */
    if ((unsigned)Cluster2 == READ_CLUSTER)
    {
      if (cluster >= MASK12)
        return LONG_LAST_CLUSTER;
      if (cluster == BAD12)
        return LONG_BAD;
      return cluster;
    }

  /*如果不是在寻找空闲的fat,那么可以计算空闲的fat*/
    if (cluster == FREE)
      wasfree++;
    cluster = res;
  }

  /* Cluster2 may be set to LONG_LAST_CLUSTER == 0x0FFFFFFFUL or 0xFFFF */
  /* -- please don't remove this mask!                                  */
  cluster2 = (unsigned)Cluster2 & 0x0fff;

  /* Now pack the value in                          */
  if (Cluster1 & 0x01)
  {
    cluster &= 0x000f;
    cluster2 <<= 4;
  }
  else
  {
    cluster &= 0xf000;
  }
  cluster |= cluster2;
  *fbp0 = (UBYTE)cluster;
  *fbp1 = (UBYTE)(cluster >> 8);
}
else if (ISFAT16(dpbp))
{
  /* form an index so that we can read the block as a    */
  /* byte array                                 */
  /* and get the cluster number                         */
  UWORD res = fgetword(&bp->b_buffer[idx * 2]);
  if ((unsigned)Cluster2 == READ_CLUSTER)
  {
    if (res >= MASK16)
```

```
      return LONG_LAST_CLUSTER;
    if (res == BAD16)
      return LONG_BAD;

    return res;
   }
   /* Finally, put the word into the buffer and mark the   */
   /* buffer as dirty.                                     */
   fputword(&bp->b_buffer[idx * 2], (UWORD)Cluster2);
   if (res == FREE)
    wasfree++;
}
#ifdef WITHFAT32
else if (ISFAT32(dpbp))
{
   /* form an index so that we can read the block as a     */
   /* byte array                                           */
   UDWORD res = fgetlong(&bp->b_buffer[idx * 4]) & LONG_LAST_CLUSTER;
   if (Cluster2 == READ_CLUSTER)
   {
    if (res > LONG_BAD)
      return LONG_LAST_CLUSTER;

    return res;
   }
   /* Finally, put the word into the buffer and mark the   */
   /* buffer as dirty.                                     */
   fputlong(&bp->b_buffer[idx * 4], Cluster2 & LONG_LAST_CLUSTER);
   if (res == FREE)
    wasfree++;
}
#endif
else
   return 1;

/* update the free space count                             */
bp->b_flag |= BFR_DIRTY | BFR_VALID;
if (Cluster2 == FREE || wasfree)
{
   int adjust = 0;
   if (!wasfree)
    adjust++;
   else if (Cluster2 != FREE)
    adjust--;
#ifdef WITHFAT32
   if (ISFAT32(dpbp) && dpbp->dpb_xnfreeclst != XUNKNCLSTFREE)
   {
    /* update the free space count for returned     */
    /* cluster                                      */
    dpbp->dpb_xnfreeclst += adjust;
    write_fsinfo(dpbp);
   }
```

```
    else
#endif
    if (dpbp->dpb_nfreeclst != UNKNCLSTFREE)
      dpbp->dpb_nfreeclst += adjust;
}
return SUCCESS;
}

/* Given the disk parameters, and a cluster number, this function
   looks at the FAT, and returns the next cluster in the clain. */
CLUSTER next_cluster(struct dpb FAR * dpbp, CLUSTER ClusterNum)
{
return link_fat(dpbp, ClusterNum, READ_CLUSTER);
}
```