FAT文件系统原理

——http://www.sjhf.net

文本结构索引：

图片表格索引：

一、硬盘的物理结构：

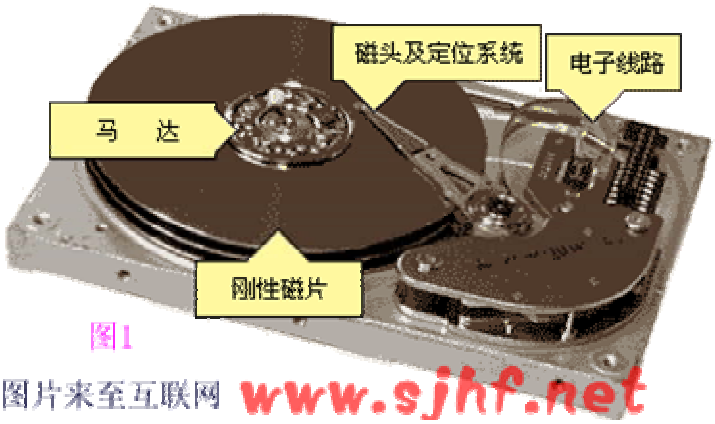


图1

硬盘存储数据是根据电、磁转换原理实现的。硬盘由一个或几个表面镀有磁性物质的金属或玻璃等物质盘片以及盘片两面所安装的磁头和相应的控制电路组成(图 1)，其中盘片和磁头密封在无尘的金属壳中。

硬盘工作时，盘片以设计转速高速旋转，设置在盘片表面的磁头则在电路控制下径向移动到指定位置然后将数据存储或读取出来。当系统向硬盘写入数据时，磁头中“写数据”电流产生磁场使盘片表面磁性物质状态发生改变，并在写电流磁场消失后仍能保持，这样数据就存储下来了；当系统从硬盘中读数据

时，磁头经过盘片指定区域，盘片表面磁场使磁头产生感应电流或线圈阻抗产生变化，经相关电路处理后还原成数据。因此只要能将盘片表面处理得更平滑、磁头设计得更精密以及尽量提高盘片旋转速度，就能造出容量更大、读写数据速度更快的硬盘。这是因为盘片表面处理越平、转速越快就能越使磁头离盘片表面越近，提高读、写灵敏度和速度；磁头设计越小越精密就能使磁头在盘片上占用空间越小，使磁头在一张盘片上建立更多的磁道以存储更多的数据。

[返回索引]

二、硬盘的逻辑结构。

硬盘由很多盘片(platter)组成，每个盘片的每个面都有一个读写磁头。如果有N个盘片。就有 2N个面，对应 2N个磁头(Heads)，从 0、1、2 开始编号。每个盘片被划分成若干个同心圆磁道(逻辑上的，是不可见的。)每个盘片的划分规则通常是一样的。这样每个盘片的半径均为固定值R的同心圆再逻辑上形成了一个以电机主轴为轴的柱面(Cylinders)，从外至里编号为 0、1、2……每个盘片上的每个磁道又被划分为几十个扇区(Sector)，通常的容量是 512byte，并按照一定规则编号为 1、2、3……形成Cylinders×Heads×Sector个扇区。这三个参数即是硬盘的物理参数。我们下面的很多实践需要深刻理解这三个参数的意义。

[返回索引]

三、磁盘引导原理。

3.1 MBR(master boot record)扇区：

计算机在按下power键以后，开始执行主板bios程序。进行完一系列检测和配置以后。开始按bios中设定的系统引导顺序引导系统。假定现在是硬盘。Bios执行完自己的程序后如何 把执行权交给硬盘呢。交给硬盘后又执行存储在哪里的程序呢。其实，称为mbr的一段代码起着举足轻重的作用。MBR(master boot record),即主引导记录，有时也称主引导扇区。位于整个硬盘的 0 柱面 0 磁头 1 扇区(可以看作是硬盘的第一个扇区)，bios在执行自己固有的程序以后就会jump到mbr中的第一条指令。将系统的控制权交由mbr来执行。在总共 512byte的主引导记录中，MBR的引导程序占了其中的前 446 个字节(偏移 0H~偏移 1BDH)，随后的 64 个字节(偏移 1BEH~偏移 1FDH)为DPT(Disk PartitionTable，硬盘分区表)，最后的两个字节“55 AA”(偏移 1FEH~偏移 1FFH)是分区有效结束标志。

MBR不随操作系统的不同而不同，意即不同的操作系统可能会存在相同的MBR，即使不同，MBR也不会夹带操作系统的性质。具有公共引导的特性。
我们来分析一段mbr。下面是用winhex查看的一块希捷 120GB硬盘的mbr。

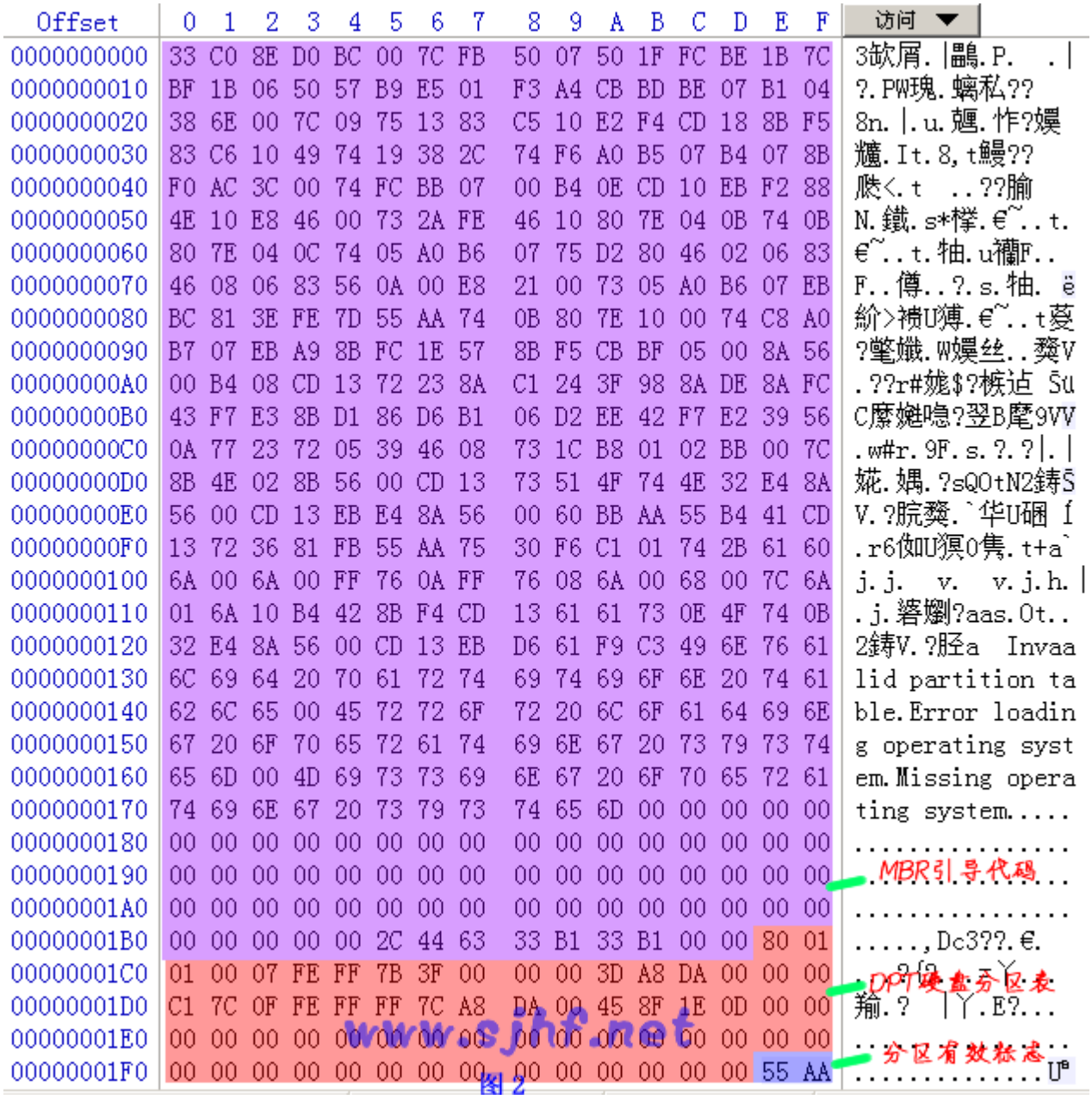


图 2

[返回索引]

你的硬盘的 MBR 引导代码可能并非这样。不过即使不同，所执行的功能大体是一样的。

我们看 DPT 部分。操作系统为了便于用户对磁盘的管理。加入了磁盘分区的概念。即将一块磁盘逻辑划分为几块。磁盘分区数目的多少只受限于 C～Z 的英文字母的数目，在上图 DPT 共 64 个字节中如何表示多个分区的属性呢?microsoft 通过链接的方法解决了这个问题。在 DPT 共 64 个字节中，以 16 个字节为分区表项单位描述一个分区的属性。也就是说，第一个分区表项描述一个分区的属性，一般为基本分区。第二个分区表项描述除基本分区外的其余空间，一般而言，就是我们所说的扩展分区。 这部分的大体说明见表 1。

表 1　图 2 分区表第一字段			
字节位移	字段长度	值	字段名和定义
0x01BE	BYTE	0x80	引导指示符(Boot Indicator)　指明该分区是否是活动分区。
0x01BF	BYTE	0x01	开始磁头(Starting Head)
0x01C0	6 位	0x01	开始扇区(Starting Sector) 只用了 0~5 位。后面的两位(第 6 位和第 7 位)被开始柱面字段所使用
0x01C1	10 位	0x00	开始柱面(Starting Cylinder)　除了开始扇区字段的最后两位外，还使用了 1 位来组成该柱面值。开始柱面是一个 10 位数，最大值为 1023
0x01C2	BYTE	0x07	系统 ID(System ID) 定义了分区的类型，详细定义，请参阅图 4
0x01C3	BYTE	0xFE	结束磁头(Ending Head)
0x01C4	6 位	0xFF	结束扇区(Ending Sector)　只使用了 0~5 位。最后两位(第 6、7 位)被结束柱面字段所使用
0x01C5	10 位	0x7B	结束柱面(Ending Cylinder) 除了结束扇区字段最后的两位外，还使用了 1 位，以组成该柱面值。结束柱面是一个 10 位的数，最大值为 1023
0x01C6	DWORD	0x0000003F	相对扇区数

			(Relative Sectors) 从该磁盘的开始到该分区的开始的位移量，以扇区来计算
0x01CA	DWORD	0x00DAA83D	总扇区数(Total Sectors) 该分区中的扇区总数

[返回索引]

注：上表中的超过 1 字节的数据都以实际数据显示，就是按高位到地位的方式显示。存储时是按低位到高位存储的。两者表现不同，请仔细看清楚。以后出现的表，图均同。

也可以在winhex中看到这些参数的意义：

Master Boot Record, 基础偏移量：0

Offset 偏移	标题	数值
0	Master bootstrap loader code 引导代码	33 C0 8E D0 BC 00 7C FB
	Partition Table Entry #1	
1BE	80 = active partition 活动分区标志	80 80表示活动，00表示非
1BF	Start head 开始磁头	1
1C0	Start sector 开始扇区	1
1C0	Start cylinder 开始柱面	0
1C2	Operating system indicator (hex	07 分区类型标志
1C3	End head 结束磁头	254
1C4	End sector 结束扇区	63
1C4	End cylinder 结束柱面	891
1C6	Sectors preceding partition 1	63 本分区之前的扇区数
1CA	Length of partition 1 in sector	14329917 本分区的扇区数数
	Partition Table Entry #2	
1CE	80 = active partition	00
1CF	Start head	0
1D0	Start sector	1
1D0	Start cylinder	892
1D2	Operating system indicator (hex	0F
1D3	End head	254
1D4	End sector	63
1D4	End cylinder	1023
1D6	Sectors preceding partition 2	14329980
1DA	Length of partition 2 in sector	220106565
	Partition Table Entry #3	
1DE 0H	80 = active partition	00
1DF 1H	Start head	0
1E0 2H	Start sector	0
1E0 2H	Start cylinder	0
1E2 4H	Operating system indicator (hex	00
1E3 5H	End head	0
1E4 6H	End sector	0
1E4 6H	End cylinder	0
1E6 8H	Sectors preceding partition 3	0
1EA CH	Length of partition 3 in sector	0
	Partition Table Entry #4	
1EE	80 = active partition	00
1EF	Start head	0
1F0	Start sector	0
1F0	Start cylinder	0
1F2	Operating system indicator (hex	00
1F3	End head	0
1F4	End sector	0
1F4	End cylinder	0
1F6	Sectors preceding partition 4	0

图3

[返回索引]

说明： 每个分区表项占用 16 个字节，假定偏移地址从 0 开始。如图 3 的分区表项 3。分区表项 4 同分区表项 3。

1、0H偏移为活动分区是否标志，只能选 00H和 80H。80H为活动，00H为非活动。其余值对microsoft而言为非法值。

2、重新说明一下(这个非常重要)：大于 1 个字节的数被以低字节在前的存储格式格式(little endian format)或称反字节顺序保存下来。低字节在前的格式是一种保存数的方法，这样，最低位的字节最先出现在十六进制数符号中。例如，相对扇区数字段的值 0x3F000000 的低字节在前表示为 0x0000003F。这个低字节在前 的格式数的十进制数为 63。

3、系统在分区时，各分区都不允许跨柱面，即均以柱面为单位，这就是通常所说的分区粒度。有时候我们分区是输入分区的大小为 7000M，分出来却是 6997M，就是这个原因。 偏移 2H和偏移 6H的扇区和柱面参数中,扇区占 6 位(bit)，柱面占 10 位(bit)，以偏移 6H为例，其低 6 位用作扇区数的二进制表示。其高两位做柱面数 10 位中的高两位，偏移 7H组成的 8 位做柱面数 10 位中的低 8 位。由此可知，实际上用这种方式表示的分区容量是有限的，柱面和磁头从 0 开始编号,扇区从 1 开始编号,所以最多只能表示 1024 个柱面×63 个扇区×256 个磁头×512byte=8455716864byte。即通常的 8.4GB(实际上应该是 7.8GB左右)限制。实际上磁头数通常只用到 255 个(由汇编语言的寻址寄存器决定),即使把这 3 个字节按线性寻址，依然力不从心。 在后来的操作系统中，超过 8.4GB的分区其实已经不通过C/H/S的方式寻址了。而是通过偏移CH～偏移FH 共 4 个字节 32 位线性扇区地址来表示分区所占用的扇区总数。可知通过 4 个字节可以表示 2^32 个扇区，即 2TB=2048GB，目前对于大多数计算机而言，这已经是个天文数字了。在未超过 8.4GB的分区上，C/H/S的表示方法和线性扇区的表示方法所表示的分区大小是一致的。也就是说，两种表示方法是协调的。即使不协调，也以线性寻址为准。(可能在某些系统中会提示出错)。超过 8.4GB的分区结束C/H/S一般填充为FEH FFH FFH。即C/H/S所能表示的最大值。有时候也会用柱面对 1024 的模来填充。不过这几个字节是什么其实都无关紧要了。

虽然现在的系统均采用线性寻址的方式来处理分区的大小。但不可跨柱面的原则依然没变。本分区的扇区总数加上与前一分区之间的保留扇区数目依然必须是柱面容量的整数倍。(保留扇区中的第一个扇区就是存放分区表的MBR或虚拟MBR的扇区，分区的扇区总数在线性表示方式上是不计入保留扇区的。如果是第一个分区，保留扇区是本分区前的所有扇区。

附：分区表类型标志如图 4

图4

[返回索引]

3.2 扩展分区：

扩展分区中的每个逻辑驱动器都存在一个类似于MBR的扩展引导记录（Extended Boot Record, EBR），也有人称之为虚拟mbr或扩展mbr，意思是一样的。扩展引导记录包括一个扩展分区表和该扇区的标签。扩展引导记录将记录只包含扩展分区中每个逻辑驱动器的第一个柱面的第一面的信息。一个逻辑驱动器中的引导扇区一般位于相对扇区 32 或 63。但是，如果磁盘上没有扩展分区，那么就不会有扩展引导记录和逻辑驱动器。第一个逻辑驱动器的扩展分区表中的第一项指向它自身的引导扇区。第二项指向下一个逻辑驱动器的EBR。如果不存在进一步的逻辑驱动器，第二项就不会使用，而且被记录成一系列零。如果有附加的逻辑驱动器，那么第二个逻辑驱动器的扩展分区表的第一项会指向它本身的引导扇区。第二个逻辑驱动器的扩展分区表的第二项指向下一个逻辑驱动器的EBR。扩展分区表的第三项和第四项永远都不会被使用。

通过一幅 4 分区的磁盘结构图可以看到磁盘的大致组织形式。如图 5：

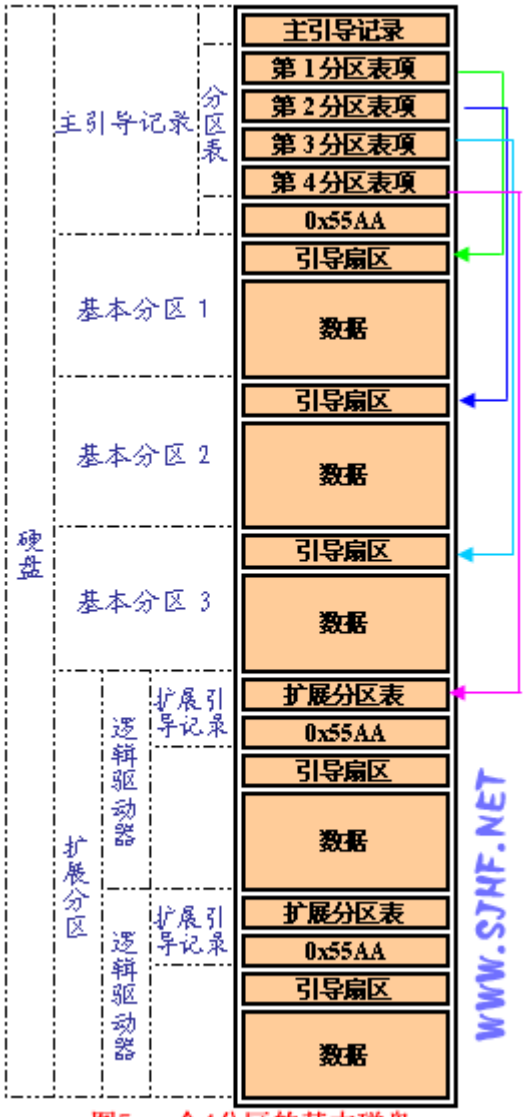


图5 一个4分区的基本磁盘

[返回索引]

关于扩展分区，如图 6 所示，扩展分区中逻辑驱动器的扩展引导记录是一个连接表。该图显示了一个扩展分区上的三个逻辑驱动器，说明了前面的逻辑驱动器和最后一个逻辑驱动器之间在扩展分区表中的差异。

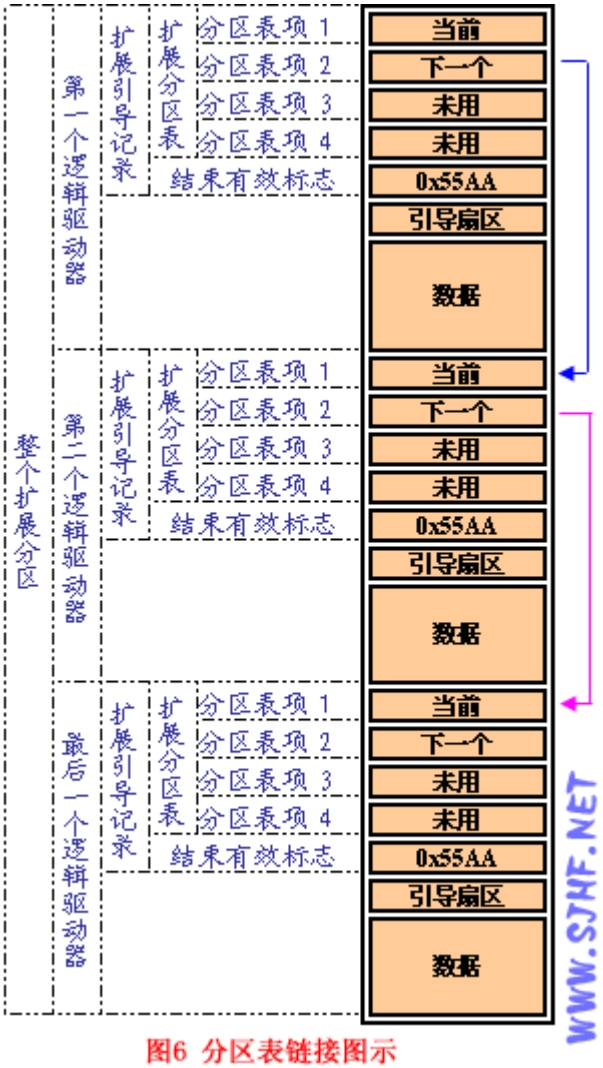


图6 分区表链接图示

[返回索引]

除了扩展分区上最后一个逻辑驱动器外，表 2 中所描述的扩展分区表的格式在每个逻辑驱动器中都是重复的：第一个项标识了逻辑驱动器本身的引导扇区，第二个项标识了下一个逻辑驱动器的 EBR。最后一个逻辑驱动器的扩展分区表只会列出它本身的分区项。最后一个扩展分区表的第二个项到第四个项被使用。

表 2　扩展分区表项的内容

扩展分区表项	分区表项的内容
第一个项	包括数据的开始地址在内的与扩展分区中当前逻辑驱动器有关的信息
第二个项	有关扩展分区中的下一个逻辑驱动器的信息，包括包含下一个逻辑驱动器的 EBR 的扇区的地址。如果不存在进一步的逻辑驱动器的话，该字段不会被使用

第三个项	未用
第四个项	未用

[返回索引]

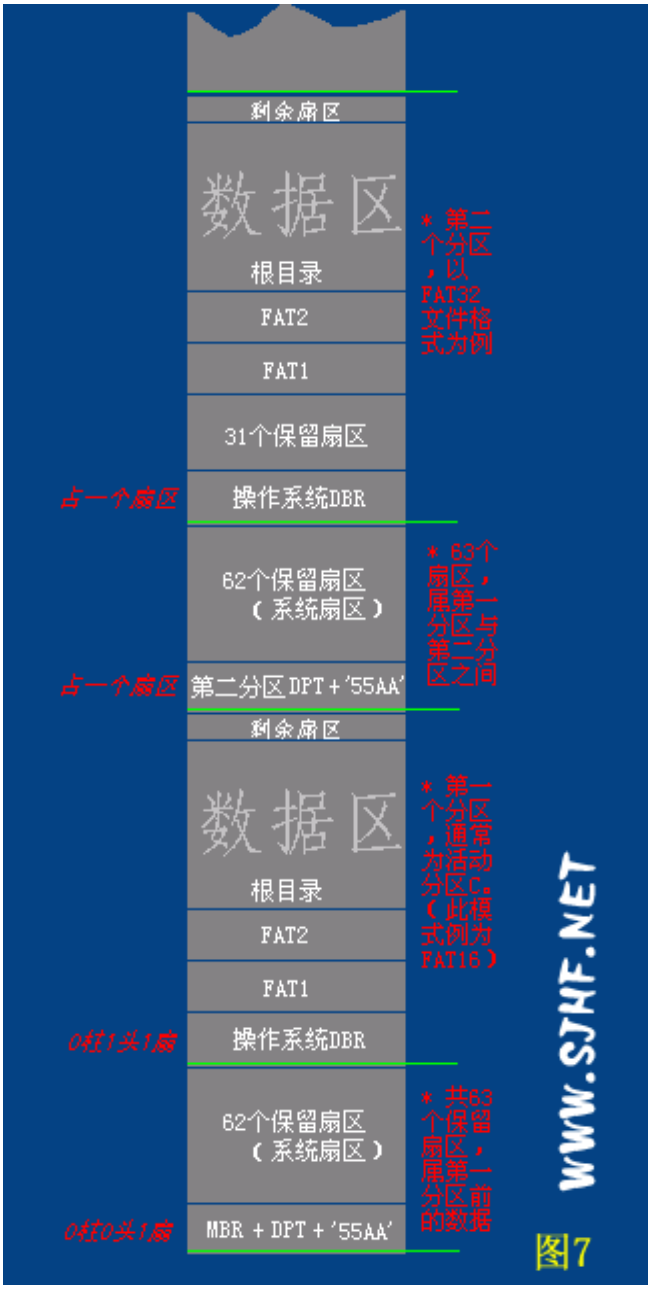
扩展分区表项中的相对扇区数字段所显示的是从扩展分区开始到逻辑驱动器中第一个扇区的位移的字节数。总扇区数字段中的数是指组成该逻辑驱动器的扇区数目。总扇区数字段的值等于从扩展分区表项所定义的引导扇区到逻辑驱动器末尾的扇区数。

有时候在磁盘的末尾会有剩余空间，剩余空间是什么呢？我们前面说到，分区是以 1 柱面的容量为分区粒度的，那么如果磁盘总空间不是整数个柱面的话，不够一个柱面的剩下的空间就是剩余空间了，这部分空间并不参与分区，所以一般无法利用。照道理说，磁盘的物理模式决定了磁盘的总容量就应该是整数个柱面的容量，为什么会有不够一个柱面的空间呢。在我的理解看来， 本来现在的磁盘为了更大的利用空间，一般在物理上并不是按照外围的扇区大于里圈的扇区这种管理方式，只是为了与操作系统兼容而抽象出来CHS。可能其实际空间容量 不一定正好为整数个柱面的容量吧。关于这点，如有高见，请告知http://www.sjhf.net或zymail@vip.sina.com。

[返回索引]

四、FAT分区原理。

先来一幅结构图：



[返回索引]

现在我们着重研究 FAT 格式分区内数据是如何存储的。FAT 分区格式是 MICROSOFT 最早支持的分区格式，依据 FAT 表中每个簇链的所占位数(有关概念，后面会讲到)分为 fat12、fat16、fat32 三种格式"变种"，但其基本存储方式是相似的。

仔细研究图 7 中的 fat16 和 fat32 分区的组成结构。下面依次解释 DBR、FAT1、FAT2、根目录、数据区、剩余扇区的概念。提到的地址如无特别提示均为分区内部偏移。

[返回索引]

4.1 关于DBR.

DBR区(DOS BOOT RECORD)即操作系统引导记录区的意思，通常占用分区的第 0 扇区共 512 个字节(特殊情况也要占用其它保留扇区，我们先说第 0 扇)。在这 512 个字节中，其实又是由跳转指令，厂商标志和操作系统版本号，BPB(BIOS Parameter Block)，扩展BPB，os引导程序，结束标志几部分组成。以用的最多的FAT32 为例说明分区DBR各字节的含义。见图 8。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	对应字符
00000000	EB	58	90	4D	53	57	49	4E	34	2E	31	00	02	08	20	00	隔恙 SWIN4.1... .
00000010	02	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00	.....?.?. .?...
00000020	3F	04	7D	00	32	1F	00	00	00	00	00	00	02	00	00	00	?.}.2..........
00000030	01	00	06	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000040	80	00	29	FE	1C	39	33	4E	4F	20	4E	41	4D	45	20	20	€.)?93NO NAME
00000050	20	20	46	41	54	33	32	20	20	20	FA	33	C9	8E	D1	BC	FAT32 ?庵鸭
00000060	F8	7B	8E	C1	BD	78	00	C5	76	00	1E	56	16	55	BF	22	鹆幜絰.舢..V.U?
00000070	05	89	7E	00	89	4E	02	B1	0B	FC	F3	A4	8E	D9	BD	00	.塺.塭.? 俳.
00000080	7C	C6	45	FE	0F	8B	46	18	88	45	F9	38	4E	40	7D	25	|羒?婩.圖?N@}%
00000090	8B	C1	99	BB	00	07	E8	97	00	72	1A	83	EB	3A	66	A1	婆櫥..钑.r.泮:f?
000000A0	1C	7C	66	3B	07	8A	57	FC	75	06	80	CA	02	88	56	02	.|f;.奧黷.€?均.
000000B0	80	C3	10	73	ED	BF	02	00	83	7E	16	00	75	45	8B	46	€?s 砜..儈..uE 婩
000000C0	1C	8B	56	1E	B9	03	00	49	40	75	01	42	BB	00	7E	E8	.媧.?.I@u.B?~?
000000D0	5F	00	73	26	B0	F8	4F	74	1D	8B	46	32	33	D2	B9	03	_.s&傍 Ot.婩 23 夜.
000000E0	00	3B	C8	77	1E	8B	76	0E	3B	CE	73	17	2B	F1	03	46	.;萋.嫩.;蝧.+?F
000000F0	1C	13	56	1E	EB	D1	73	0B	EB	27	83	7E	2A	00	77	03	..V.胙 s.?儈*.w.
00000100	E9	FD	02	BE	7E	7D	AC	98	03	F0	AC	84	C0	74	17	3C	荤.繟}瑯.赻勒 t.<
00000110	FF	74	09	B4	0E	BB	07	00	CD	10	EB	EE	BE	81	7D	EB	t.??.?腩緛}?
00000120	E5	BE	7F	7D	EB	E0	98	CD	16	5E	1F	66	8F	04	CD	19	寰 }豚樛.^.f??
00000130	41	56	66	6A	00	52	50	06	53	6A	01	6A	10	8B	F4	60	AVfj.RP.Sj.j.嬝`
00000140	80	7E	02	0E	75	04	B4	42	EB	1D	91	92	33	D2	F7	76	€~..u.謈?憭 3 吟 v
00000150	18	91	F7	76	18	42	87	CA	F7	76	1A	8A	F2	8A	E8	C0	.咔 v.B 轉鱳.嫉婠?
00000160	CC	02	0A	CC	B8	01	02	8A	56	40	CD	13	61	8D	64	10	?.谈..獘@?a 肓.
00000170	5E	72	0A	40	75	01	42	03	5E	0B	49	75	B4	C3	0D	0A	^r.@u.B.^.Iu 疵..
00000180	49	6E	76	61	6C	69	64	20	73	79	73	74	65	6D	20	64	Invalid system d
00000190	69	73	6B	0D	0A	44	69	73	6B	20	49	2F	4F	20	65	72	isk..Disk I/O er
000001A0	72	6F	72	0D	0A	52	65	70	6C	61	63	65	20	74	68	65	ror..Replace the
000001B0	20	64	69	73	6B	2C	20	61	6E	64	20	74	68	65	6E	20	disk, and then
000001C0	70	72	65	73	73	20	61	6E	79	20	6B	65	79	0D	0A	00	press any key...
000001D0	6B	65	79	0D	0A	00	00	00	49	4F	20	20	20	20	20	20	key.....IO
000001E0	53	59	53	4D	53	44	4F	53	20	20	20	53	59	53	7E	01	SYSMSDOS SYS~.
000001F0	00	57	49	4E	42	4F	4F	54	20	53	59	53	00	00	55	AA	.WINBOOT SYS..U

图8

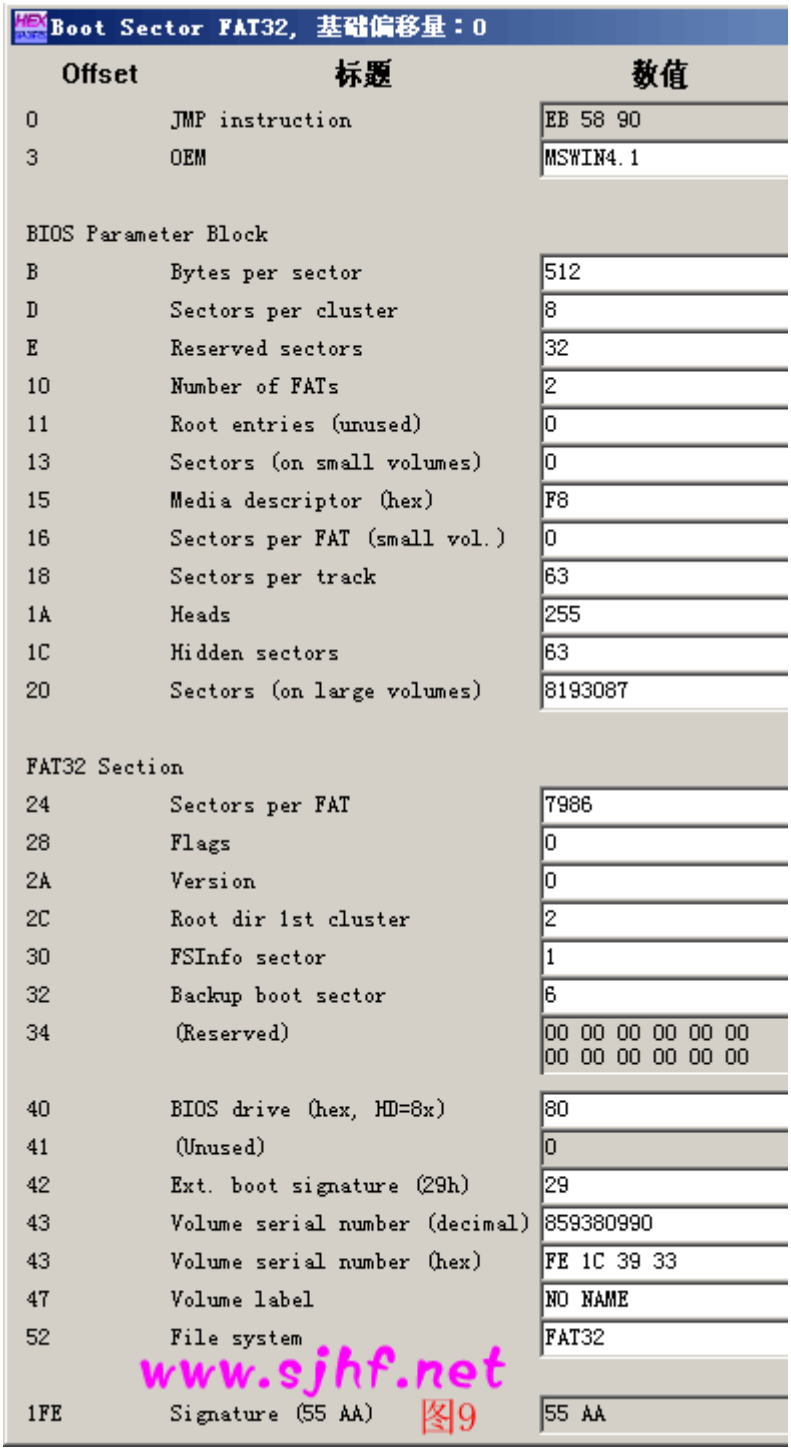
[返回索引]

图 8 的对应解释见表 3

表 3　FAT32 分区上 DBR 中各部分的位置划分			
字节位移	字段长度	字段名	对应图 8 颜色
0x00	3 个字节	跳转指令	
0x03	8 个字节	厂商标志和 os 版本号	
0x0B	53 个字节	BPB	
0x40	26 个字节	扩展 BPB	
0x5A	420 个字节	引导程序代码	
0x01FE	2 个字节	有效结束标志	

[返回索引]

图 9 给出了winhex对图 8 DBR的相关参数解释：



Boot Sector FAT32, 基础偏移量：0

Offset	标题	数值
0	JMP instruction	EB 58 90
3	OEM	MSWIN4.1
	BIOS Parameter Block	
B	Bytes per sector	512
D	Sectors per cluster	8
E	Reserved sectors	32
10	Number of FATs	2
11	Root entries (unused)	0
13	Sectors (on small volumes)	0
15	Media descriptor (hex)	F8
16	Sectors per FAT (small vol.)	0
18	Sectors per track	63
1A	Heads	255
1C	Hidden sectors	63
20	Sectors (on large volumes)	8193087
	FAT32 Section	
24	Sectors per FAT	7986
28	Flags	0
2A	Version	0
2C	Root dir 1st cluster	2
30	FSInfo sector	1
32	Backup boot sector	6
34	(Reserved)	00 00 00 00 00 00 00 00 00 00 00 00
40	BIOS drive (hex, HD=8x)	80
41	(Unused)	0
42	Ext. boot signature (29h)	29
43	Volume serial number (decimal)	859380990
43	Volume serial number (hex)	FE 1C 39 33
47	Volume label	NO NAME
52	File system	FAT32
1FE	Signature (55 AA)	55 AA

图9

[返回索引]

根据上边图例，我们来讨论DBR各字节的参数意义。

MBR将CPU执行转移给引导扇区，因此，引导扇区的前三个字节必须是合法的可执行的基于x86 的CPU指令。这通常是一条跳转指令，该指令负责跳过接下来的几个不可执行的字节(BPB和扩展BPB)，跳到操作系统引导代码部分。

跳转指令之后是 8 字节长的OEM ID，它是一个字符串， OEM ID标识了格式化该分区的操作系统的名称和版本号。为了保留与MS-DOS的兼容性，通常 Windows 2000 格式化该盘是在FAT16 和FAT32 磁盘上的该字段中记录了“MSDOS 5.0”，在NTFS磁盘上(关于ntfs，另述)，Windows 2000 记录的是“NTFS”。

通常在被Windows 95格式化的磁盘上OEM ID字段出现“MSWIN4.0”，在被Windows 95 OSR2和Windows 98格式化的磁盘上OEM ID字段出现“MSWIN4.1”。

接下来的从偏移0x0B开始的是一段描述能够使可执行引导代码找到相关参数的信息。通常称之为BPB(BIOS Parameter Block)，BPB一般开始于相同的位移量，因此，标准的参数都处于一个已知的位置。磁盘容量和几何结构变量都被封在BPB之中。由于引导扇区的第一部分是一个x86跳转指令。因此，将来通过在BPB末端附加新的信息，可以对BPB进行扩展。只需要对该跳转指令作一个小的调整就可以适应BPB的变化。 图9已经列出了项目的名称和取值，为了系统的研究，针对图8，将FAT32分区格式的BPB含义和扩展BPB含义释义为表格，见表4和表5。

表4 FAT32分区的BPB字段

字节位移	字段长度(字节)	图8对应取值	名称和定义
0x0B	2	0x0200	扇区字节数(Bytes Per Sector) 硬件扇区的大小。本字段合法的十进制值有512、1024、2048和4096。对大多数磁盘来说，本字段的值为512
0x0D	1	0x08	每簇扇区数(Sectors Per Cluster),一簇中的扇区数。由于FAT32文件系统只能跟踪有限个簇(最多为4 294 967 296个)，因此，通过增加每簇扇区数，可以使FAT32文件系统支持最大分区数。一个分区缺省的簇大小取决于该分区的大小。本字段的合法十进制值有1、2、4、8、16、32、64和128。Windows 2000的FAT32实现只能创建最大为32GB的分区。但是，Windows 2000能够访问由其他操作系统(Windows 95、OSR2及其以后的版本)所创建的更大的分区
0x0e	2	0x0020	保留扇区数(Reserved Sector) 第一个FAT开始之前的扇区数，包括引导扇区。本字段的十进制值一般为32
0x10	1	0x02	FAT数(Number of FAT) 该分区上FAT的副本数。本字段的值一般为2
0x11	2	0x0000	根目录项数(Root Entries)只

			有 FAT12/FAT16 使用此字段。对 FAT32 分区而言,本字段必须设置为 0
0x13	2	0x0000	小扇区数(Small Sector)(只有 FAT12/FAT16 使用此字段)对 FAT32 分区而言，本字段必须设置为 0
0x15	1	0xF8	媒体描述符(Media Descriptor)提供有关媒体被使用的信息。值 0xF8 表示硬盘，0xF0 表示高密度的 3.5 寸软盘。媒体描述符要用于 MS-DOS FAT16 磁盘，在 Windows 2000 中未被使用
0x16	2	0x0000	每 FAT 扇区数(Sectors Per FAT)只被 FAT12/FAT16 所使用,对 FAT32 分区而言，本字段必须设置为 0
0x18	2	0x003F	每道扇区数(Sectors Per Track) 包含使用 INT13h 的磁盘的“每道扇区数”几何结构值。该分区被多个磁头的柱面分成了多个磁道
0x1A	2	0x00FF	磁头数(Number of Head) 本字段包含使用 INT 13h 的磁盘的“磁头数”几何结构值。例如，在一张 1.44MB 3.5 英寸的软盘上，本字段的值为 2
0x1C	4	0x0000003F	隐藏扇区数(Hidden Sector) 该分区上引导扇区之前的扇区数。在引导序列计算到根目录的数据区的绝对位移的过程中使用了该值。本字段一般只对那些在中断 13h 上可见的媒体有意义。在没有分区的媒体上它必须总是为 0
0x20	4	0x007D043F	总扇区数(Large Sector) 本字段包含 FAT32 分区中总的扇区数
0x24	4	0x00001F32	每 FAT 扇区数(Sectors Per FAT)(只被 FAT32 使用)该分区每个 FAT 所占的扇区数。计算机利用这个数和 FAT 数以及隐藏扇区数(本表中所描述的)来决定根目录从哪里开始。该计算机还可以从目录中的项数决定该分区的用户数据区从哪里开始
0x28	2	0x00	扩展标志(Extended Flag)(只被 FAT32 使用)该两个字节结

			构中各位的值为： 位 0-3：活动 FAT 数(从 0 开始计数，而不是 1). 只有在不使用镜像时才有效 位 4-6：保留 位 7：0 值意味着在运行时 FAT 被映射到所有的 FAT 1 值表示只有一个 FAT 是活动的 位 8-15：保留
0x2A	2	0x0000	文件系统版本(File ystem Version)只供 FAT32 使用,高字节是主要的修订号，而低字节是次要的修订号。本字段支持将来对该 FAT32 媒体类型进行扩展。如果本字段非零，以前的 Windows 版本将不支持这样的分区
0x2C	4	0x00000002	根目录簇号(Root Cluster Number)(只供 FAT32 使用) 根目录第一簇的簇号。本字段的值一般为 2，但不总是如此
0x30	2	0x0001	文件系统信息扇区号(File System Information SectorNumber)(只供 FAT32 使用) FAT32 分区的保留区中的文件系统信息(File System Information, FSINFO)结构的扇区号。其值一般为 1。在备份引导扇区(Backup Boot Sector)中保留了该 FSINFO 结构的一个副本，但是这个副本不保持更新
0x34	2	0x0006	备份引导扇区(只供 FAT32 使用) 为一个非零值，这个非零值表示该分区保存引导扇区的副本的保留区中的扇区号。本字段的值一般为 6，建议不要使用其他值
0x36	12	12 个字节均为 0x00	保留(只供 FAT32 使用)供以后扩充使用的保留空间。本字段的值总为 0

[返回索引]

表 5　FAT32 分区的扩展 BPB 字段			
字节位移	字段长度	图 8 对应取值	字段名称和定义

	(字节)		
0x40	1	0x80	物理驱动器号(Physical Drive Number) 与 BIOS 物理驱动器号有关。软盘驱动器被标识为 0x00，物理硬盘被标识为 0x80，而与物理磁盘驱动器无关。一般地，在发出一个 INT13h BIOS 调用之前设置该值，具体指定所访问的设备。只有当该设备是一个引导设备时，这个值才有意义
0x41	1	0x00	保留(Reserved) FAT32 分区总是将本字段的值设置为 0
0x42	1	0x29	扩展引导标签(Extended Boot Signature) 本字段必须要有能被 Windows 2000 所识别的值 0x28 或 0x29
0x43	4	0x33391CFE	分区序号(Volume Serial Number) 在格式化磁盘时所产生的一个随机序号，它有助于区分磁盘
0x47	11	"NO NAME"	卷标(Volume Label) 本字段只能使用一次，它被用来保存卷标号。现在，卷标被作为一个特殊文件保存在根目录中
0x52	8	"FAT32"	系统 ID(System ID) FAT32 文件系统中一般取为"FAT32"

[返回索引]

DBR的偏移 0x5A开始的数据为操作系统引导代码。这是由偏移 0x00 开始的跳转指令所指向的。在图 8 所列出的偏移 0x00~0x02 的跳转指令"EB 58 90"清楚地指明了OS引导代码的偏移位置。jump 58H加上跳转指令所需的位移量，即开始于 0x5A。此段指令在不同的操作系统上和不同的引导方式上，其内容也是不同的。 大多数的资料上都说win98,构建于fat基本分区上的win2000,winxp所使用的DBR只占用基本分区的第 0 扇区。他们提到，对于fat32，一般的 32 个基本分区保留扇区只有第 0 扇区是有用的。实际上，以FAT32 构建的操作系统如果是win98,系统会使用基本分区的第 0 扇区和第 2 扇区 存储os引导代码；以FAT32 构建的操作系统如果是win2000 或winxp,系统会使用基本分区的第 0 扇区和第 0xC扇区(win2000 或winxp,其第 0xC的位置由第 0 扇区的 0xAB偏移指出)存储os引导代码。所以，在fat32 分区格式上，如果DBR一扇区的内容正确而缺少第 2 扇区(win98 系统)或第 0xC扇区(win2000 或winxp系统)，系统也是无法启动的。如果 自己手动设置NTLDR双系统，必须知道这一点。

DBR扇区的最后两个字节一般存储值为 0x55AA的DBR有效标志，对于其他

的取值，系统将不会执行DBR相关指令。 上面提到的其他几个参与os引导的扇区也需以 0x55AA为合法结束标志。

[返回索引]

FAT16 DBR：

FAT32 中DBR的含义大致如此，对于FAT12 和FAT16 其基本意义类似，只是相关偏移量和参数意义有小的差异，FAT格式的区别和来因，以后会说到，此处不在多说FAT12 与FAT16。我将FAT16 的扇区参数意义列表。感兴趣的朋友自己研究一下，和FAT32 大同小异的。

表 6 一个 FAT16 分区上的引导扇区段

字节位移	字段长度(字节)	字段名称
0x00	3	跳转指令(Jump Instruction)
0x03	8	OEM ID
0x0B	25	BPB
0x24	26	扩展 BPB
0x3E	448	引导程序代码(Bootstrap Code)
0x01FE	4	扇区结束标识符(0x55AA)

[返回索引]

表 7 FAT16 分区的 BPB 字段

字节位移	字段长度(字节)	例值	名称和定义
0x0B	2	0x0200	扇区字节数(Bytes Per Sector) 硬件扇区的大小。本字段合法的十进制值有 512、1024、2048 和 4096。对大多数磁盘来说，本字段的值为 512
0x0D	1	0x40	每簇扇区数(Sectors Per Cluster) 一个簇中的扇区数。由于 FAT16 文件系统只能跟踪有限个簇(最多为 65536 个)。因此，通过增加每簇的扇区数可以支持最大分区数。分区的缺省的簇的

			大小取决于该 分区的大小。本字段合法的十进制值有 1、2、4、8、16、32、64 和 128。导致簇大于 32KB(每扇区字节数*每簇扇区数)的值会引起磁盘错误和软件错误
0x0e	2	0x0001	保留扇区数(Reserved Sector) 第一个 FAT 开始之前的扇区数，包括引导扇区。本字段的十进制值一般为 1
0x10	1	0x02	FAT 数(Number of FAT)该分区上 FAT 的副本数。本字段的值一般为 2
0x11	2	0x0200	根目录项数(Root Entries)能够保存在该分区的根目录文件夹中的 32 个字节长的文件和文件夹名称项的总数。在一个典型的硬盘上，本字段的值为 512。其中一个项常常被用作卷标号(Volume Label)，长名称的文件和文件夹每个文件使用多个项。文件和文件夹项的最大数一般为 511，但是如果使用的长文件名，往往都达不到这个数
0x13	2	0x0000	小扇区数(Small Sector)该分区上的扇区数，表示为 16 位(<65536)。对大于 65536 个扇区的分区来说，本字段的值为 0，而使用大扇区数来取代它
0x15	1	0xF8	媒体描述符(Media Descriptor)提供有关媒体被使用的信息。值 0xF8 表示硬盘，0xF0 表示高密度的 3.5 寸软盘。媒体描述符要用于 MS-DOS FAT16 磁盘，在 Windows 2000 中未被使用
0x16	2	0x00FC	每 FAT 扇区数(Sectors Per FAT) 该分区上每个 FAT 所占用的扇区数。计算机利用这个数和 FAT 数以及隐藏扇区数来决定根目录在哪里开始。计算机还可以根据根目录中的项数(512)决定该 分区的用户数据区从哪里开始

0x18	2	0x003F	每道扇区数(Sectors Per Trark)
0x1A	2	0x0040	磁头数(Number of head)
0x1C	4	0x0000003F	隐藏扇区数(Hidden Sector) 该分区上引导扇区之前的扇区数。在引导序列计算到根目录和数据区的绝对位移的过程中使用了该值
0x20	4	0x003EF001	大扇区数(Large Sector) 如果小扇区数字段的值为 0，本字段就包含该 FAT16 分区中的总扇区数。如果小扇区数字段的值不为 0，那么本字段的值为 0

[返回索引]

表 8　FAT16 分区的扩展 BPB 字段

字节位移	字段长度(字节)	图 8 对应取值	字段名称和定义
0x24	1	0x80	物理驱动器号(Physical Drive Number) 与 BIOS 物理驱动器号有关。软盘驱动器被标识为 0x00，物理硬盘被标识为 0x80，而与物理磁盘驱动器无关。一般地，在发出一个 INT13h BIOS 调用之前设置该值，具体指定所访问的设备。只有当该设备是一个引导设备时，这个值才有意义
0x25	1	0x00	保留(Reserved) FAT16 分区一般将本字段的值设置为 0
0x26	1	0x29	扩展引导标签(Extended Boot Signature) 本字段必须要有能被 Windows 2000 所识别的值 0x28 或 0x29
0x27	2	0x52368BA8	卷序号(Volume Serial Number) 在格式化磁盘时所产生的一个随机序号，它有助于区分磁盘
0x2B	11	"NO NAME"	卷标(Volume Label) 本字段只能使用一次，它被用来保存卷标号。现在，卷标被

			作为一个特殊文件保存在根目录中
0x36	8	"FAT16"	文件系统类型(File System Type) 根据该磁盘格式，该字段的值可以为 FAT、FAT12 或 FAT16

[返回索引]

4.2 关于保留扇区

在上述 FAT 文件系统 DBR 的偏移 0x0E 处，用 2 个字节存储保留扇区的数目。所谓保留扇区(有时候会叫系统扇区，隐藏扇区)， 是指从分区 DBR 扇区开始的仅为系统所有的扇区，包括 DBR 扇区。在 FAT16 文件系统中，保留扇区的数据通常设置为 1，即仅仅 DBR 扇区。而在 FAT32 中，保留扇区的数据通常取为 32，有时候用 Partition Magic 分过的 FAT32 分区会设置 36 个保留扇区，有的工具可能会设置 63 个保留扇区。

FAT32 中的保留扇区除了磁盘总第 0 扇区用作 DBR，总第 2 扇区(win98 系统)或总第 0xC 扇区(win2000,winxp)用作 OS 引导代码扩展部分外，其余扇区都不参与操作系统管理与磁盘数据管理，通常情况下是没作用的。操作系统之所以在 FAT32 中设置保留扇区，是为了对 DBR 作备份或留待以后升级时用。FAT32 中，DBR 偏移 0x34 占 2 字节的数据 指明了 DBR 备份扇区所在，一般为 0x06，即第 6 扇区。当 FAT32 分区 DBR 扇区被破坏导致分区无法访问时。可以用第 6 扇区的原备份替换第 0 扇区来找回数据。

[返回索引]

4.3 FAT表和数据的存储原则。

FAT 表(File Allocation Table 文件分配表)，是 Microsoft 在 FAT 文件系统中用于磁盘数据(文件)索引和定位引进的一种链式结构。假如把磁盘比作一本书，FAT 表可以认为相当于书中的目录，而文件就是各个章节的内容。但 FAT 表的表示方法却与目录有很大的不同。

在 FAT 文件系统中，文件的存储依照 FAT 表制定的簇链式数据结构来进行。同时，FAT 文件系统将组织数据时使用的目录也抽象为文件，以简化对数据的管理。

[返回索引]

★存储过程假想：

我们模拟对一个分区存储数据的过程来说明FAT文件系统中数据的存储原则。

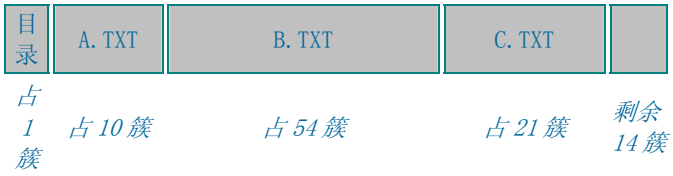
假定现在有一个空的完全没有存放数据的磁盘，大小为 100KB，我们将其想象为线形的空间地址。为了存储管理上的便利，我们人为的将这 100KB的空间均分成 100 份，每份 1KB。我们来依次存储这样几个文件：A.TXT(大小 10KB),B.TXT(大小 53.6KB)，C.TXT(大小 20.5KB)。

最起码能够想到，我们可以顺序的在这 100KB空间中存放这 3 个文件。同时不要忘了，我们还要记下他们的大小和开始的位置，这样下次要用时才能找

的到，这就像是目录。为了便于查找，我们假定用第 1K的空间来存储他们的特征(属性)。还有，我们设计的存储单位是 1KB，所 以，A.TXT我们需要 10 个存储单位(为了说明方便，我们把存储单位叫做“簇”吧。也能少打点字，呵呵。)，B.TXT需要 54 个簇，C.TXT需要 21 个簇。可能有人会说B.TXT和C.TXT不是各自浪费了不到 1 簇的空间吗？干嘛不让他们紧挨着，不是省地方吗？我的回答是，如果按照这样的方式存储，目录中原本只需要记下簇号，现在还需要记下簇内的偏移，这样会增加目录的存储量，而且存取没有了规则，读取也不太方便，是得不偿失的。

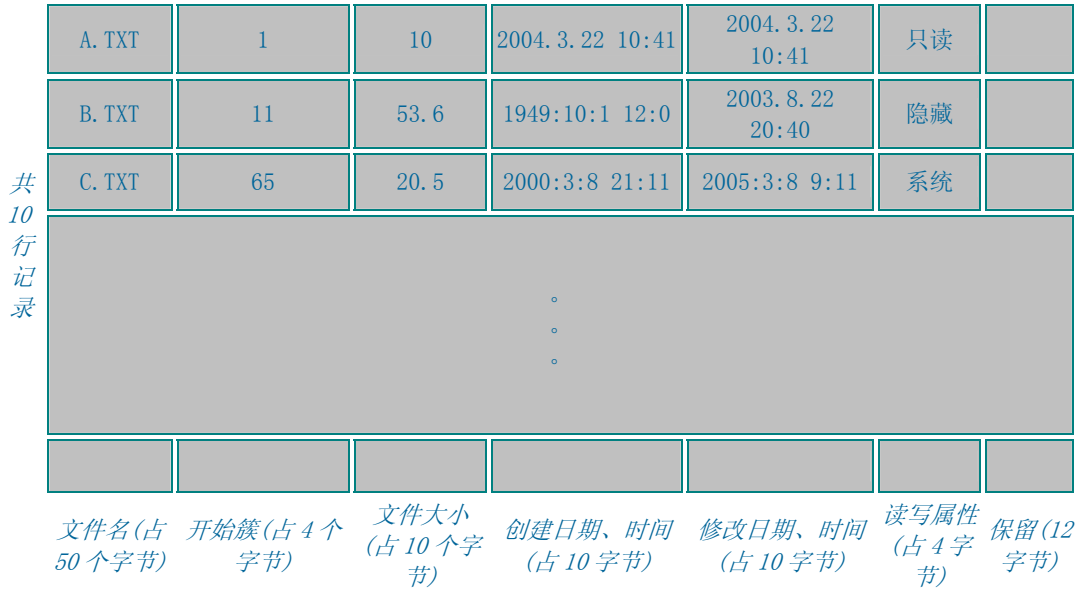
根据上面所说的思想，我们设计了这样的图 4.3.1 所示的存储方式。

图 4.3.1 整个 100KB 空间



我们再考虑如何来写这三个文件的目录。对于每个文件而言，一定要记录的有：文件名，开始簇，大小，创建日期、时间，修改日期、时间，文件的读写属性等。这里大小能不能用结束簇来计算呢？一定不能，因为文件的大小不一定就是整数个簇的大小，否则的话像 B.TXT 的内容就是 54KB 的内容了，少了固然不行，可多了也是不行的。那么我们怎么记录呢？可以想象一下。为了管理上的方便，我们用数据库的管理方式来管理我们的目录。于是我把 1KB 再分成 10 份，假定开始簇号为 0，定义每份 100B 的各个位置的代表含义如图 4.3.2

图 4.3.2 每行 100B 共 10 行(这是例子，非 Fat 系统)



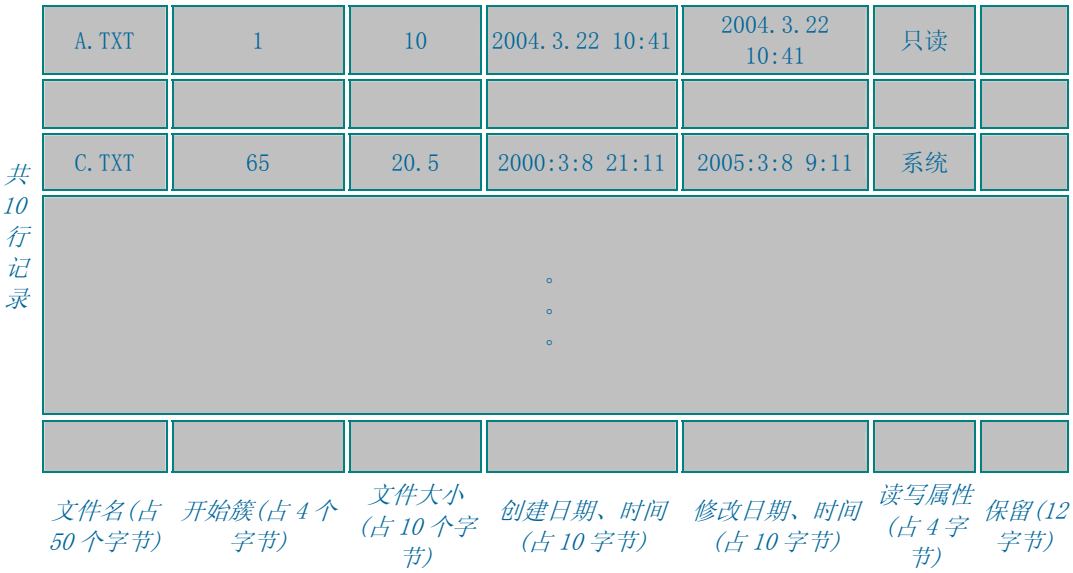
这样设计的结构绝对可以对文件进行正确的读写了。接着让我们设计的文件系统工作吧。先改动个文件，比如 A.TXT，增加点内容吧！咦？增加后往哪里放呀，虽然存储块的后面有很多空间，但紧随其后 B.TXT 的数据还顶着呢？要是把 A.TXT 移到后边太浪费 处理资源，而且也不一定解决问题。这个问题看来暂

时解决不了。

那我们换个操作，把 B.txt 删了，b.txt 的空间随之释放。这时候空间如图 4.3.3，目录如图 4.3.4

图 4.3.3　整个 100KB 空间

图 4.3.4　每行 100B 共 10 行(这是例子，非 Fat 系统)



这个操作看来还可以，我们接着做，在存入一个文件 D.txt(大小为 60.3KB),总共 100 簇的空间只用了 31 簇，还有 68 簇剩余，按说能放下。可是？往那里放呢？没有 61 个连续的空间了，目录行没办法写了，看来 无连续块存储暂时也不行。

你一定能够想到我们可以在连续空间不够或增加文件长度的时候转移影响我们操作的其他文件，从而腾出空间来，但我要问你，那不是成天啥也不要干了，就是倒腾东西了吗？

看来我们设计的文件系统有致命的漏洞，怎么解决呢？。。。。。。。。。。。

其实可以这样解决：

首先我们允许文件的不连续存储。目录中依然只记录开始簇和文件的大小。那么我们怎么记录文件占用那些簇呢，以文件映射簇不太方便，因为文件名是不固定的。我们换个思想，可以用簇来映射文件，在整个存储空间的前部留下几簇来记录数据区中数据与簇号的关系。对于上例因为总空间也不大，所

以用前部的 1Kb 的空间来记录这种对应，假设 3 个文件都存储，空间分配如 图 4.3.5，同时修改一下目录，如图 4.3.6

图 4.3.5 整个 100KB 空间

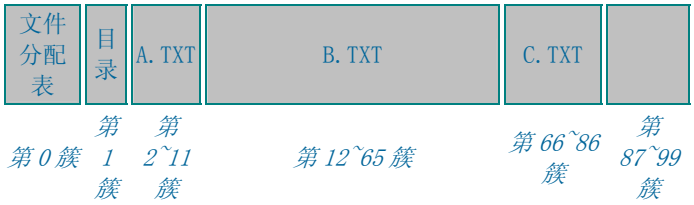
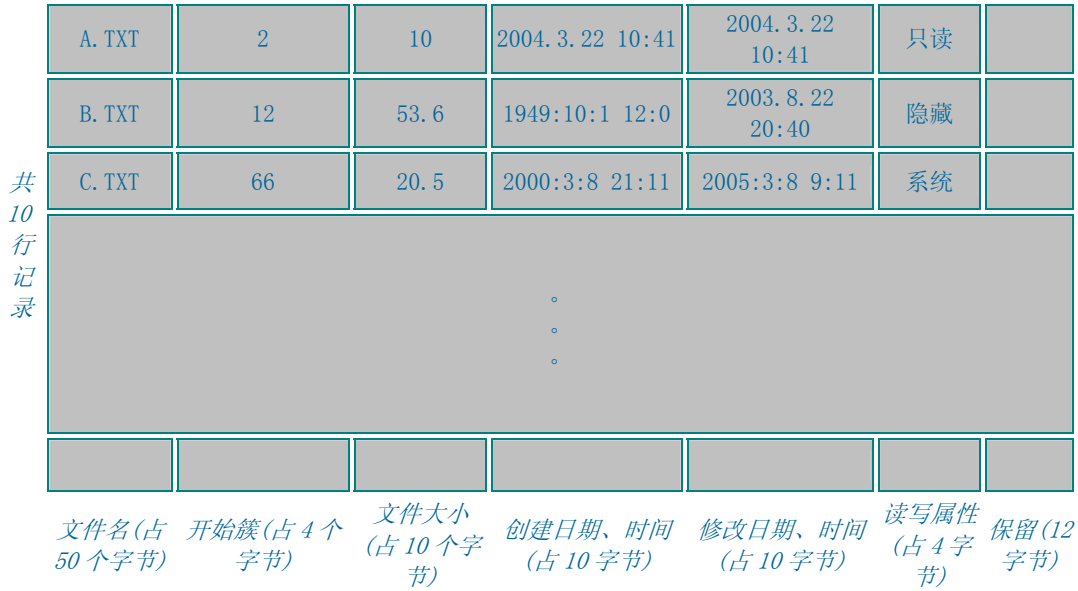
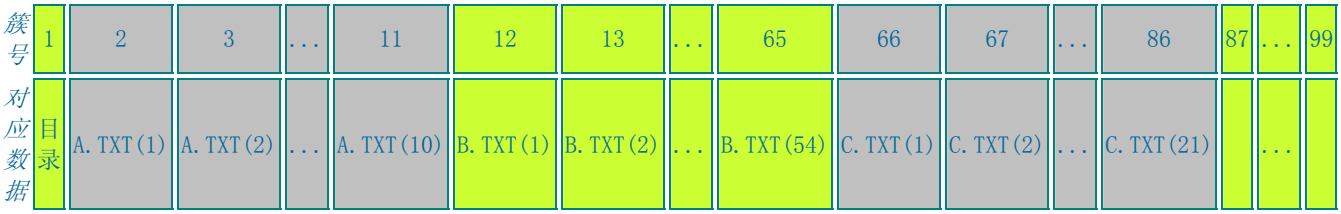


图 4.3.6 每行 100B 共 10 行(这是例子，非 Fat 系统)



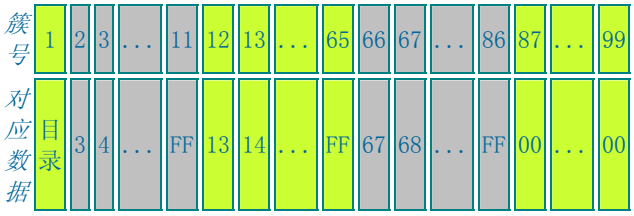
第一簇用来记录数据区中每一簇的被占用情况，暂时称其为文件分配表。结合文件分配表和文件目录就可以达到完全的文件读取了。我们想到，把文件分配表做成一个数据表，以图 4.3.7 的形式记录簇与数据的对应。

图 4.3.7 文件分配表



用图 4.3.7 的组织方式是完全可以实现对文件占有簇的记录的。但还不够效率。比如文件名在文件分配表中记录太多，浪费空间，而实际上在目录中已经记录了文件的开始簇了。所以可以改良一下，用链的方式来存放占有簇的关系，变成 图 4.3.8 的组织方式。

图 4.3.8　改良后的文件分配表



参照图 4.3.8 来理解一下文件分配表的意义。如文件 a.txt 我们根据目录项中指定的 a.txt 的首簇为 2，然后找到文件分配表的第 2 簇记录，上面登记的是 3，我们就能确定下一簇是 3。找到文件分配表的第 3 簇记录，上面登记的是 4，我们就能确定下一簇是 4.....直到指到第 11 簇，发现下一个指向是 FF，就是结束。文件便丝毫无误读取完毕。

我们再看上面提到的第三种情况，就是将 b.txt 删除以后，存入一个大小为 60.3KB 的 d.txt。利用簇链可以很容易的实现。实现后的磁盘如 图 4.3.9　4.3.10　4.3.11

图 4.3.9 整个 100KB 空间划分

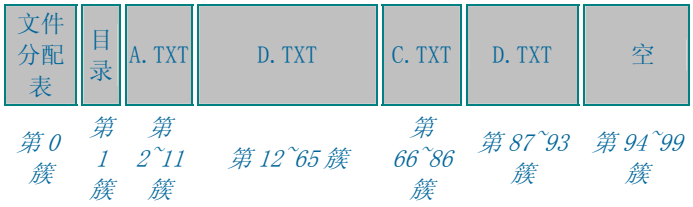


图 4.3.10 文件分配表

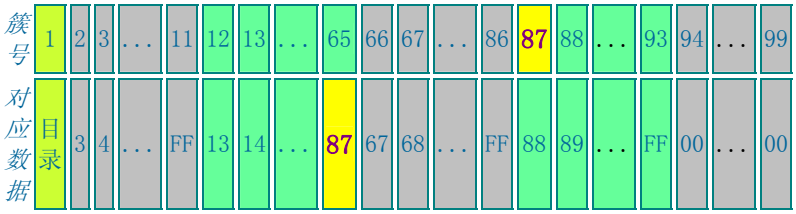
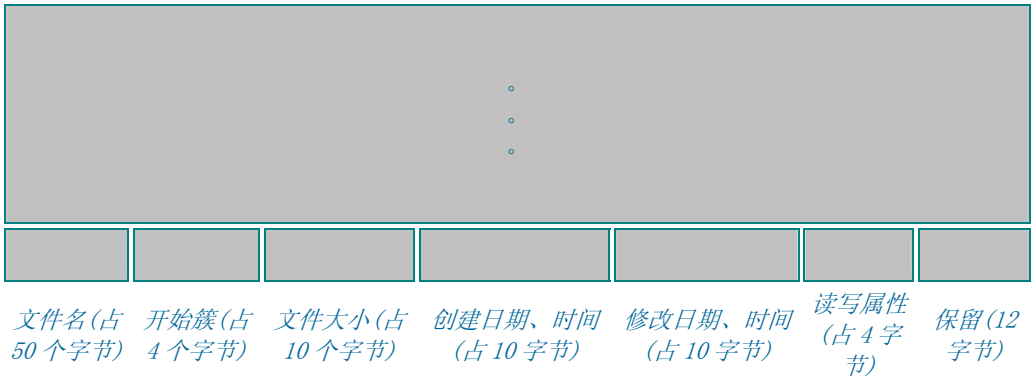


图 4.3.11　目录 每行 100B 共 10 行(这是例子，非 Fat 系统)

共 10 行记录							
	A.TXT	2	10	2004.3.22 10:41	2004.3.22 10:41	只读	
	C.TXT	66	20.5	2000:3:8 21:11	2005:3:8 9:11	系统	
	D.TXT	12	60.3	1999:5:1 8:00	2003:3:20 14:0	存档	



文件名(占50个字节)	开始簇(占4个字节)	文件大小(占10个字节)	创建日期、时间(占10字节)	修改日期、时间(占10字节)	读写属性(占4字节)	保留(12字节)

上面是我们对文件存储的一种假设，也该揭开谜底的时候了。上面的思想其实就是 fat 文件系统的思想的精髓(但并不是，尤其像具体的参数的意义与我们所举的例子是完全不同的。请忘掉上边细节，努力记忆下边)。

[返回索引]

★FAT16 存储原理:

当把一部分磁盘空间格式化为fat文件系统时，fat文件系统就将这个分区当成整块可分配的区域进行规划，以便于数据的存储。一般来讲，其划分形式如图 7 所示。我们把FAT16 部分提取出来， 详细描述一下：

FAT16 是Microsoft较早推出的文件系统，具有高度兼容性，目前仍然广泛应用于个人电脑尤其是移动存储设备中，FAT16 简单来讲由图 4.3.12 所示的 6 部分组成(主要是前 5 部分)。引导扇区(DBR)我们已经说过,FAT16 在DBR之后没有留有任何保留扇区，其后紧随的便是FAT表。FAT表是FAT16 用来记录磁盘数据区簇链结构的。像前面我们说过的例子一样，FAT将磁盘空间按一定数目的扇区为单位进行划分，这样 的单位称为簇。通常情况下，每扇区 512 字节的原则是不变的。簇的大小一般是 2^n (n为整数)个扇区的大小，像 512B,1K,2K,4K,8K,16K,32K，64K。实际中通常不超过 32K。 之所以簇为单位而不以扇区为单位进行磁盘的分配，是因为当分区容量较大时，采用大小为512b的扇区管理会增加fat表的项数，对大文件存取增加消耗，文件系统效率不高。分区的大小和簇的取值是有关系的，见表 9

图 4.3.12 Fat16 的组织形式

引导扇区	FAT1	FAT2(重复的)	根文件夹	其他文件夹及所有文件	剩余扇区
1扇区	实际情况取大小	同FAT1	32个扇区	开始簇编号(从2开始)	不足一簇

[返回索引]

表 9 FAT16 分区大小与对因簇大小		
分区空间大小	每个簇的扇区	簇空间大

		小
0MB-32MB	1	512 个字节
33MB-64MB	2	1k
65MB-128MB	4	2k
129MB-225MB	8	4k
256MB-511MB	16	8k
512MB-1023MB	32	16k
1024MB-2047MB	64	32k
2048MB-4095MB	128	64k

[返回索引]

注意：少于 32680 个扇区的分区中，簇空间大小可最多达到每个簇 8 个扇区。不管用户是使用磁盘管理器来格式化分区，还是使用命令提示行键入 format命令格式化，格式化程序都创建一个 12 位的FAT。少于 16MB的 分区，系统通常会将其格式化成 12 位的FAT，FAT12 是FAT的初始实现形式，是针对小型介质的。FAT12 文件分配表要比FAT16 和FAT32 的文件分配表小，因为它对每个条目使用的空间较少。这就给数据留下较多的空间。所有用FAT12 格式化的 5.25 英寸软盘以及 1.44MB的 3.5 英寸软盘都是由FAT12 格式化的。除了FAT表中记录每簇链结的二进制位数与FAT16 不同外，其余原理与FAT16 均相同，不再单独解释。。。

格式化 FAT16 分区时，格式化程序根据分区的大小确定簇的大小，然后根据保留扇区的数目、根目录的扇区数目、数据区可分的簇数与 FAT 表本身所占空间 来确定 FAT 表所需的扇区数目，然后将计算后的结果写入 DBR 的相关位置。

FAT16 DBR 参数的偏移 0x11 处记录了根目录所占扇区的数目。偏移 0x16 记录了 FAT 表所占扇区的数据。偏移 0x10 记录了 FAT 表的副本数目。系统在得到这几项参数以后，就可以确定数据区的开始扇区偏移了。

FAT16 文件系统从根目录所占的 32 个扇区之后的第一个扇区开始以簇为单位进行数据的处理，这之前仍以扇区为单位。对于根目录之后的第一个簇，系统并不编号为第 0 簇或第 1 簇 (可能是留作关键字的原因吧)，而是编号为第 2 簇，也就是说数据区顺序上的第 1 个簇也是编号上的第 2 簇。

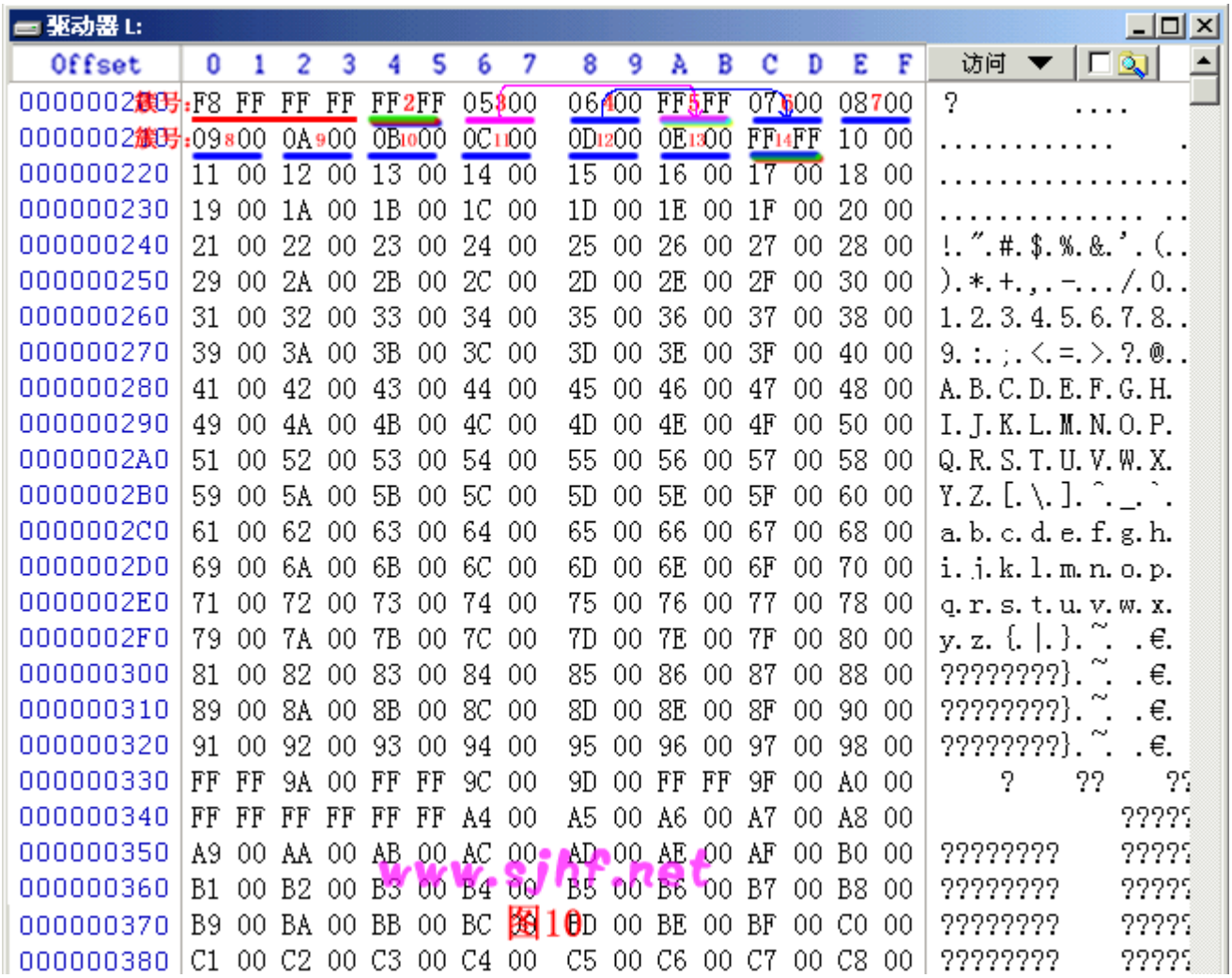
FAT 文件系统之所以有 12，16，32 不同的版本之分，其根本在于 FAT 表用来记录任意一簇链接的二进制位数。以 FAT16 为例，每一簇在 FAT 表中占据 2 字节(二进制 16 位)。所以，FAT16 最大可以表示的簇号为 0xFFFF(十进制的 65535)，以 32K 为簇的大小的话，FAT32 可以管理的最大磁盘空间为：32KB×65535=2048MB,这就是为什么 FAT16 不支持超过 2GB 分区的原因。

FAT 表实际上是一个数据表，以 2 个字节为单位，我们暂将这个单位称为 FAT 记录项，通常情况其第 1、2 个记录项(前 4 个字节)用作介质描述。从第三个记录项开始记录除根目录外的其他文件及文件夹的簇链情况。根据簇的表现情况 FAT 用相应的取值来描述，见表 10

表 10 FAT16 记录项的取值含义(16 进制)	
FAT16 记录项的取值	对应簇的表现情况
0000	未分配的簇
0002~FFEF	已分配的簇
FFF0~FFF6	系统保留
FFF7	坏簇
FFF8~FFFF	文件结束簇

看一幅在 winhex 所截 FAT16 的文件分配表，图 10：

[返回索引]



[返回索引]

如图，FAT表以"F8 FF FF FF" 开头，此 2 字节为介质描述单元，并不参与FAT表簇链关系。小红字标出的是FAT扇区每 2 字节对应的簇号。

相对偏移 0x4~0x5 偏移为第 2 簇(顺序上第 1 簇)，此处为FF,表示存储在第 2 簇上的文件(目录)是个小文件，只占用 1 个簇便结束了。

第 3 簇中存放的数据是 0x0005，这是一个文件或文件夹的首簇。其内容为第 5 簇，就是说接下来的簇位于第 5 簇——〉 FAT表指引我们到达FAT表的第 5 簇指向，上面写的数据是"FF FF",意即此文件已至尾簇。

第 4 簇中存放的数据是 0x0006，这又是一个文件或文件夹的首簇。其内容为第 6 簇，就是说接下来的簇位于第 6 簇——〉FAT表指引我们到达FAT表的第 6 簇指向，上面写的数据是 0x0007，就是说接下来的簇位于第 7 簇——〉FAT表指引我们到达FAT表的第 7 簇 指向……直到根据FAT链读取到扇区相对偏移 0x1A~0x1B，也就是第 13 簇，上面写的数据是 0x000E，也就是指向第 14 簇——〉14 簇的内容为"FF FF"，意即此文件已至尾簇。

后面的FAT表数据与上面的道理相同。不再分析。

FAT表记录了磁盘数据文件的存储链表，对于数据的读取而言是极其重要的，以至于Microsoft为其开发的FAT文件系统中的FAT表创建了一份备份，就是我们看到的FAT2。FAT2 与FAT1 的内容通常是即时同步的，也就是说如果通过正

常的系统读写对FAT1 做了更改，那么FAT2 也同样被更新。如果从这个角度来看，系统 的这个功能在数据恢复时是个天灾。

FAT 文件系统的目录结构其实是一颗有向的从根到叶的树，这里提到的有向是指对于 FAT 分区内的任一文件(包括文件夹)，均需从根目录寻址来找到。可以这样认为：目录存储结构的入口就是根目录。

FAT 文件系统根据根目录来寻址其他文件(包括文件夹)，故而根目录的位置必须在磁盘存取数据之前得以确定。FAT 文件系统就是根据分区的相关 DBR 参数与 DBR 中存放的已经计算好的 FAT 表(2 份)的大小来确定的。 格式化以后，跟目录的大小和位置其实都已经确定下来了：位置紧随 FAT2 之后，大小通常为 32 个扇区。根目录之后便是数据区第 2 簇。

FAT 文件系统的一个重要思想是把目录(文件夹)当作一个特殊的文件来处理，FAT32 甚至将根目录当作文件处理(旁：NTFS 将分区参数、安全权限等好多东西抽象为文件更是这个思想的升华)，在 FAT16 中，虽然根目录地位并不等同于普通的文件或者说是目录，但其组织形式和普通的目录(文件夹)并没有不同。FAT 分区中所有的文件夹(目录)文件，实际上可以看作是一个存放其他文件(文件夹)入口参数的数据表。所以目录的占用空间的大小并不等同于其下所有数据的大小，但也不等同于 0。通常是占很小的空间的，可以看作目录文件是一个简单的二维表文件。其具体存储原理是：

不管目录文件所占空间为多少簇，一簇为多少字节。系统都会以 32 个字节为单位进行目录文件所占簇的分配。这 32 个字节以确定的偏移来定义本目录下的一个文件(或文件夹)的属性，实际上是一个简单的二维表。

这 32 个字节的各字节偏移定义如表 11：

表 11　FAT16 目录项 32 个字节的表示定义

字节偏移(16 进制)	字节数	定义	
0x0~0x7	8	文件名	
0x8~0xA	3	扩展名	
0xB	1	属性字节	00000000(读写)
			00000001(只读)
			00000010(隐藏)
			00000100(系统)
			00001000(卷标)
			00010000(子目录)
			00100000(归档)
0xC~0x15	10	系统保留	
0x16~0x17	2	文件的最近修改时间	
0x18~0x19	2	文件的最近修改日期	
0x1A~0x1B	2	表示文件的首簇号	
0x1C~0x1F	4	表示文件的长度	

[返回索引]

对表 11 中的一些取值进行说明：

(1)、对于短文件名，系统将文件名分成两部分进行存储，即主文件名+扩展名。0x0~0x7 字节记录文件的主文件名，0x8~0xA记录文件的扩展名，取文件名中的ASCII码值。不记录主文件名与扩展名之间的"."　主文件名不足 8 个字符以空白符(20H)填充，扩展名不足 3 个字符同样以空白符(20H)填充。0x0 偏移处的取值若为 00H，表明目录项为空；若为E5H，表明目录项曾被使用，但对应的文件或文件夹已被删除。(这也是误删除后恢复的理论依据)。文件名中的第一个字符若为“.”或“..”表示这个簇记录的是一个子目录的目录项。“.”代表当前目录；“..”代表上级目录(和我们在dos或windows中的使用意思是一样的，如果磁盘数据被破坏，就可以通过这两个目录项 的具体参数推算磁盘的数据区的起始位置，猜测簇的大小等等，故而是比较重要的)

(2)、0xB的属性字段：可以看作系统将 0xB的一个字节分成 8 位，用其中的一位代表某种属性的有或无。这样，一个字节中的 8 位每位取不同的值就能反映各个属性的不同取值了。如 00000101 就表示这是个文件，属性是只读、系统。

(3)、0xC~0x15 在原FAT16 的定义中是保留未用的。在高版本的WINDOWS系统中有时也用它来记录修改时间和最近访问时间。那 样其字段的意义和FAT32 的定义是相同的，见后边FAT32。

(4)、0x16~0x17 中的时间=小时*2048+分钟*32+秒/2。得出的结果换算成 16 进制填入即可。也就是：0x16 字节的 0~4 位是以 2 秒为 单位的量值；0x16 字节的 5~7 位和 0x17 字节的 0~2 位是分钟；0x17 字节的 3~7 位是小时。

(5)、0x18~0x19 中的日期=(年份-1980)*512+月份*32+日。得出的结果换算成 16 进制填入即可。也就是：0x18 字节 0~4 位是日期数；0x18 字节 5~7 位和 0x19 字节 0 位是月份；0x19 字节的 1~7 位为年号，原定义中 0~119 分别代表 1980~2099，目前高版本的Windows允许取 0~127，即年号最大可以到 2107 年。

(6)、0x1A~0x1B存放文件或目录的表示文件的首簇号，系统根据掌握的首簇号在FAT表中找到入口，然后再跟踪簇链直至簇尾，同时用 0x1C~0x1F处字节判定有效性。就可以完全无误的读取文件(目录)了。

(7)、普通子目录的寻址过程也是通过其父目录中的目录项来指定的，与数据文件(指非目录文件)不同的是目录项偏移 0xB的第 4 位置 1，而数据文件为 0。

对于整个FAT分区而言，簇的分配并不完全总是分配干净的。如一个数据区为 99 个扇区的FAT系统，如果簇的大小设定为 2 扇区，就会有 1 个扇区无法分配给任何一个簇。这就是分区的剩余扇区，位于分区的末尾。有的系统用最后一个剩余扇区备份本分区的DBR，这也是一种好的备份方法。

早的FAT16 系统并没有长文件名一说，Windows操作系统已经完全支持在 FAT16 上的长文件名了。FAT16 的长文件名与FAT32 长文件名的定义是相同的，关于长文件名，在FAT32 部分再详细作解释。

[返回索引]

★FAT32 存储原理：

FAT32 是个非常有功劳的文件系统，Microsoft成功地设计并运用了它，直

到今天NTFS铺天盖地袭来的时候，FAT32 依然占据着Microsoft Windows文件系统中重要的地位。FAT32 最早是出于FAT16 不支持大分区、单位簇容量大以致空间急剧浪费等缺点设计的。实际应用中，FAT32 还是成功的。

FAT32 与FAT16 的原理基本上是相同的，图 4.3.13 标出了FAT32 分区的基本构成。

图 4.3.13 Fat32 的组织形式

[返回索引]

FAT32 在格式化的过程中就根据分区的特点构建好了它的DBR，其中BPB参数是很重要的，可以回过头来看一下表 4 和表 5。首先FAT32 保留扇区的数目默认为 32 个，而不是FAT16 的仅仅一个。这样的好处是有助于磁盘DBR指令的 长度扩展，而且可以为DBR扇区留有备份空间。上面我们已经提到，构建在FAT32上的win98 或win2000、winXP，其操作系统引导代码并非只占一个扇区了。留有多余的保留扇区就可以很好的拓展OS引导代码。在BPB中也记录了DBR扇区的备份扇区编号。 备份扇区可以让我们在磁盘遭到意外破坏时恢复DBR。

FAT32 的文件分配表的数据结构依然和FAT16 相同，所不同的是，FAT32 将记录簇链的二进制位数扩展到了 32 位，故而这种文件系统称为FAT32。32 位二进制位的簇链 决定了FAT表最大可以寻址 2T个簇。这样即使簇的大小为 1 扇区，理论上仍然能够寻址 1TB范围内的分区。但实际中FAT32 是不能寻址这样大的空间的，随着分区空间大小的增加，FAT表的记录数会变得 臃肿不堪，严重影响系统的性能。所以在实际中通常不格式化超过 32GB的FAT32 分区。WIN2000及之上的OS已经不直接支持对超过 32GB的分区格式化成FAT32， 但WIN98 依然可以格式化大到 127GB的FAT32 分区，但这样没必要也不推荐。同时FAT32 也有小的限制，FAT32 卷必须至少有 65527 个簇，所以对于小的分区，仍然需要使用FAT16 或FAT12。

分区变大时，如果簇很小，文件分配表也随之变大。仍然会有上面的效率问题存在。既要有效地读写大文件，又要最大可能的减少空间的浪费。FAT32 同样规定了相应的分区空间对应的簇的大小，见表 12：

表 12 FAT32 分区大小与对因簇大小

分区空间大小	每个簇的扇区	簇空间大小
<8GB	8	4k
>=8GB 且<16GB	16	8k
>=16GB 且<32GB	32	16k
>=32GB	64	32k

[返回索引]

簇的取值意义和FAT16 类似，不过是位数长了点罢了，比较见表 13：

表 13 FAT 各系统记录项的取值含义(16 进制)

FAT12 记录项的取值	FAT16 记录项的取值	FAT32 记录项的取值	对应簇的表现情况
000	0000	00000000	未分配的簇
002~FFF	0002~FFEF	00000002~FFFFFFEF	已分配的簇
FF0~FF6	FFF0~FFF6	FFFFFFF0~FFFFFFF6	系统保留
FF7	FFF7	FFFFFFF7	坏簇
FF8~FFF	FFF8~FFFF	FFFFFFF8~FFFFFFFF	文件结束簇

FAT32 的另一项重大改革是根目录的文件化，即将根目录等同于普通的文件。这样根目录便没有了FAT16 中 512 个目录项的限制，不够用的时候增加簇链，分配空簇即可。而且，根目录的位置也不再硬性 地固定了，可以存储在分区内可寻址的任意簇内，不过通常根目录是最早建立的(格式化就生成了)目录表。所以，我们看到的情况基本上都是根目录首簇占簇区顺序上的第 1 个簇。在图 4.3.12 中也 是按这种情况制作的画的。

FAT32 对簇的编号依然同FAT16。顺序上第 1 个簇仍然编号为第 2 簇，通常为根目录所用(这和FAT16 是不同的，FAT16 的根目录并不占簇区空间，32 个扇区的根目录以后才是簇区第 1 个簇)

FAT32 的文件寻址方法与FAT16 相同，但目录项的各字节参数意义却与FAT16 有所不同，一方面它启用了FAT16 中的目录项保留字段，同时又完全支持

长文件名了。

对于短文件格式的目录项。其参数意义见表 14：

[返回索引]

表 14　FAT32 短文件目录项 32 个字节的表示定义			
字节偏移(16 进制)	字节数	定义	
0x0~0x7	8	文件名	
0x8~0xA	3	扩展名	
0xB*	1	属性字节	00000000(读写)
			00000001(只读)
			00000010(隐藏)
			00000100(系统)
			00001000(卷标)
			00010000(子目录)
			00100000(归档)
0xC	1	系统保留	
0xD	1	创建时间的 10 毫秒位	
0xE~0xF	2	文件创建时间	
0x10~0x11	2	文件创建日期	
0x12~0x13	2	文件最后访问日期	
0x14~0x15	2	文件起始簇号的高 16 位	
0x16~0x17	2	文件的最近修改时间	
0x18~0x19	2	文件的最近修改日期	
0x1A~0x1B	2	文件起始簇号的低 16 位	
0x1C~0x1F	4	表示文件的长度	

＊ 此字段在短文件目录项中不可取值 0FH,如果设值为 0FH，目录段为长文件名目录段

[返回索引]

说明：

(1)、这是FAT32 短文件格式目录项的意义。其中文件名、扩展名、时间、日期的算法和FAT16 时相同的。

(2)、由于FAT32 可寻址的簇号到了 32 位二进制数。所以系统在记录文件(文件夹)开始簇地址的时候也需要 32 位来记录，FAT32 启用目录项偏移 0x12~0x13 来表示起始簇号的高 16 位。

(3)、文件长度依然用 4 个字节表示，这说明FAT32 依然只支持小于 4GB的文件(目录)，超过 4GB的文件(目录),系统会截断处理。

FAT32 的一个重要的特点是完全支持长文件名。长文件名依然是记录在目录项中的。为了低版本的 OS 或程序能正确读取长文件名文件，系统自动为所有长文件名文件创建了一个对应的短文件名，使 对应数据既可以用长文件名寻

址，也可以用短文件名寻址。不支持长文件名的 OS 或程序会忽略它认为不合法的长文件名字段，而支持长文件名的 OS 或程序则会以长文件名为显式项来记录和编辑，并隐藏起短文件名。

当创建一个长文件名文件时，系统会自动加上对应的短文件名，其一般有的原则：

(1)、取长文件名的前 6 个字符加上"~1"形成短文件名，扩展名不变。

(2)、如果已存在这个文件名，则符号"~"后的数字递增，直到 5。

(3)、如果文件名中"~"后面的数字达到 5，则短文件名只使用长文件名的前两个字母。通过数学操纵长文件名的剩余字母生成短文件名的后四个字母，然后加后缀"~1"直到最后(如果有必要，或是其他数字以避免重复的文件名)。

(4)、如果存在老 OS 或程序无法读取的字符，换以"_"

长文件名的实现有赖于目录项偏移为 0xB 的属性字节，当此字节的属性为：只读、隐藏、系统、卷标，即其值为 0FH 时，DOS 和 WIN32 会认为其不合法而忽略其存在。这正是长文件名存在的依据。将目录项的 0xB 置为 0F，其他就任由系统定义了，Windows9x 或 Windows 2000、XP 通常支持不超过 255 个字符的长文件名。系统将长文件名以 13 个字符为单位进行切割，每一组占据一个目录项。所以可能一个文件需要多个目录项，这时长文件名的 各个目录项按倒序排列在目录表中，以防与其他文件名混淆。

长文件名中的字符采用 unicode 形式编码(一个巨大的进步哦)，每个字符占据 2 字节的空间。其目录项定义如表 15。

表 15　FAT32 长文件目录项 32 个字节的表示定义

字节偏移(16 进制)	字节数	定义		
0x0	1	属性字节位意义	7	保留未用
			6	1 表示长文件最后一个目录项
			5	保留未用
			4	顺序号数值
			3	
			2	
			1	
			0	
0x1~0xA	10	长文件名 unicode 码①		
0xB	1	长文件名目录项标志，取值 0FH		
0xC	1	系统保留		
0xD	1	校验值(根据短文件名计算得出)		
0xE~0x19	12	长文件名 unicode 码②		
0x1A~0x1B	2	文件起始簇号(目前常置 0)		
0x1C~0x1F	4	长文件名 unicode 码③		

[返回索引]

系统在存储长文件名时，总是先按倒序填充长文件名目录项，然后紧跟其对应的短文件名。从表 15 可以看出，长文件名中并不存储对应文件的文件开始簇、文件大小、各种时间和日期属性。文件的这些属性还是存放在短文件名目录项中，一个长文件名总是和其相应的短文件名一一对应，短文件名没有了长文件名还可以读，但长文件名如果没有对应的短文件名，不管什么系统都将忽略其存在。所以短文件名是至关重要的。在不支持长文件名的环境中对短文件名中的文件名和扩展名字段作更改(包括删除，因为删除是对首字符改写E5H)，都会使长文件名形同虚设。 长文件名和短文件名之间的联系光靠他们之间的位置关系维系显然远远不够。其实，长文件名的 0xD字节的校验和起很重要的作用，此校验和是用短文件名的 11 个字符通过一种运算方式来得到的。系统根据相应的算法来确定相应的长文件名和短文件名是否匹配。这个算法不太容易用公式说明，我们用一段c程序来加以说明。

假设文件名 11 个字符组成字符串shortname[],校验和用chknum表示。得到过程如下：

```
int i, j,chknum=0;
for (i=11; i>0; i--)
    chksum = ((chksum & 1) ? 0x80 : 0) + (chksum >> 1) +
shortname[j++];
```

如果通过短文件名计算出来的校验和与长文件名中的 0xD 偏移处数据不相等。系统无论如何都不会将它们配对的。

依据长文件名和短文件名对目录项的定义，加上对簇的编号和链接，FAT32 上数据的读取便游刃有余了。

五、结束。

[返回索引]