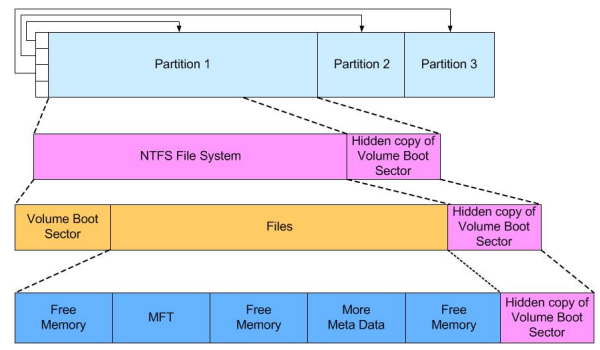
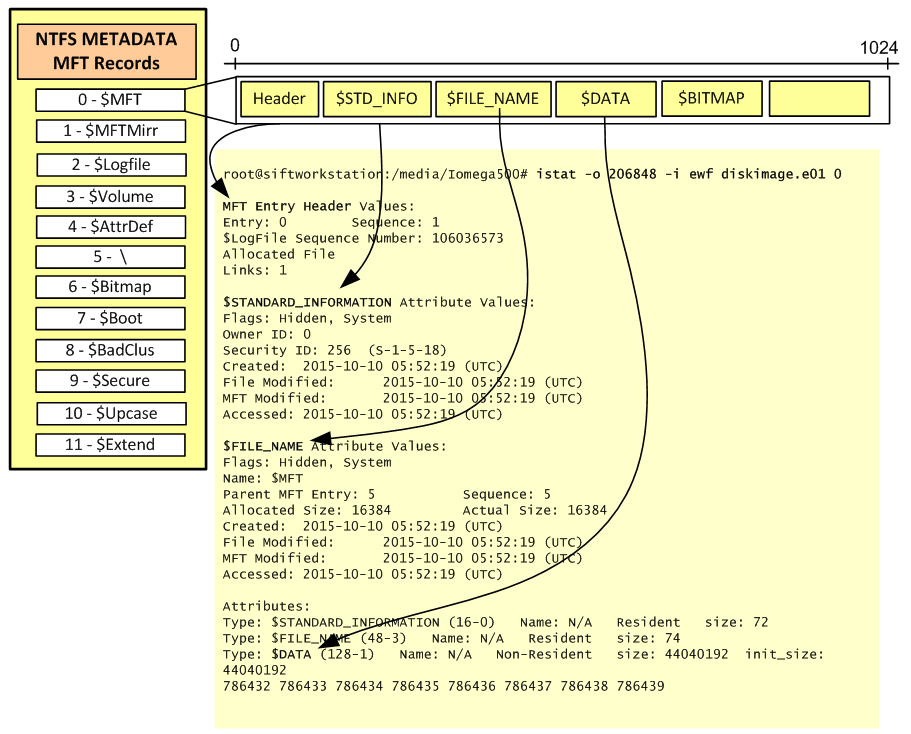
1. NTFS文件系统的整体布局



1. MFT布局

MFT的每条记录都包含一个头部和一个或多个属性（按属性ID升序），并以四个字节的0xFFFFFFFF结束。



1. 属性

3.1 ATTRIBUTE\_TYPE

MFT的每条记录包含一个或多个属性，下表为每条记录可能包含的属性，这些数据在元文件 $AttrDef 中。

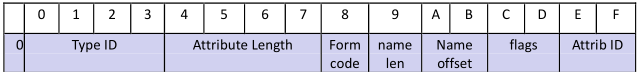
表1 MFT记录中可能存在的属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **属性名** | **OS** | **类型** | **备注** | |
| **$STANDARD\_INFORMATION** |  | **0x10** | 常驻，文件标准信息 | |
| **$ATTRIBUTE\_LIST** |  | **0x20** | 非常驻，文件需要多条MFT记录存放 | |
| **$FILE\_NAME** |  | **0x30** | 常驻，文件名 | |
| **$VOLUME\_VERSION** | NT | **0x40** |  | |
| **$OBJECT\_ID** | 2K | **0x40** | 常驻，文件ID | |
| **$SECURITY\_DESCRIPTOR** |  | **0x50** | 安全描述符 | |
| **$VOLUME\_NAME** |  | **0x60** | 卷名 | |
| **$VOLUME\_INFORMATION** |  | **0x70** | 卷信息 | |
| **$DATA** |  | **0x80** | 文件数据 | |
| **$INDEX\_ROOT** |  | **0x90** | 根目录 | 又称 $I30 属性 |
| **$INDEX\_ALLOCATION** |  | **0xA0** | 索引分配 |
| **$BITMAP** |  | **0xB0** | 位图 |
| **$SYMBOLIC\_LINK** | NT | **0xC0** |  | |
| **$REPARSE\_POINT** | 2K | **0xC0** | 重解析点 | |
| **$EA\_INFORMATION** |  | **0xD0** |  | |
| **$EA** |  | **0xE0** |  | |
| **$PROPERTY\_SET** | NT | **0xF0** |  | |
| **$LOGGED\_UTILITY\_STREAM** | 2K | **0x100** |  | |

|  |
| --- |
| **/\* 属性类型 \*/**  **typedef enum {**  **AttributeStandardInformation = 0x10,**  **AttributeAttributeList = 0x20,**  **AttributeFileName = 0x30,**  **AttributeObjectId = 0x40,**  **AttributeSecurityDescriptor = 0x50,**  **AttributeVolumeName = 0x60,**  **AttributeVolumeInformation = 0x70,**  **AttributeData = 0x80,**  **AttributeIndexRoot = 0x90,**  **AttributeIndexAllocation = 0xA0,**  **AttributeBitmap = 0xB0,**  **AttributeReparsePoint = 0xC0,**  **AttributeEAInformation = 0xD0,**  **AttributeEA = 0xE0,**  **AttributePropertySet = 0xF0,**  **AttributeLoggedUtilityStream = 0x100**  **} ATTRIBUTE\_TYPE, \*PATTRIBUTE\_TYPE; // 4 bytes** |

3.2 ATTRIBUTE

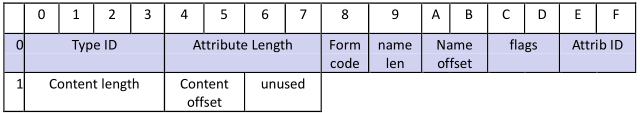
属性大小视该属性是否有名称和是否是常驻属性而定，每个属性头部的前16字节结构相同：



|  |
| --- |
| **/\* 属性头部前 16 个字节 \*/**  **typedef struct {**  **ATTRIBUTE\_TYPE AttributeType; // e.g. 0x10, 0x80. 4@0x00**  **ULONG Length; // 该属性常驻部分的字节大小, 4@0x04**  **BOOLEAN Nonresident; // true = 非常驻属性, 1@0x08**  **UCHAR NameLength; // 属性名的字符大小, 1@0x09**  **USHORT NameOffset; // 属性名的内部偏移，属性名为Unicode, 2@0x0A**  **USHORT Flags; // 0x0001 = 属性被压缩, 2@0x0C**  **USHORT AttributeNumber; // 该属性实例的数字标识符, 2@0x0E**  **} ATTRIBUTE, \*PATTRIBUTE; // 16 bytes** |

3.3 RESIDENT\_ATTRIBUTE

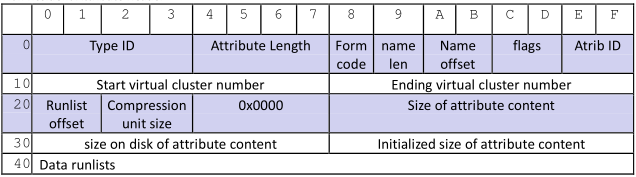
常驻属性头部（如果有属性名，属性名位置@0x18，长度为2\*ATTRIBUTE.NameLength，属性名后接属性值）：



|  |
| --- |
| **/\* 常驻属性头部 \*/**  **typedef struct {**  **ATTRIBUTE Attribute; // 16@0x00**  **ULONG ValueLength; // 属性大小， 4@0x10**  **USHORT ValueOffset; // 属性值的内部偏移， 2@0x14**  **USHORT Flags; // 0x0001 = The attribute is Indexed, 2@0x16**  **} RESIDENT\_ATTRIBUTE, \*PRESIDENT\_ATTRIBUTE; // 24 bytes** |

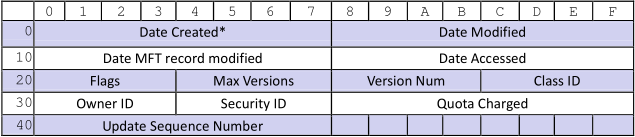
3.4 NONRESIDENT\_ATTRIBUTE

非常驻属性头部（如果有属性名，属性名位置@0x40，长度为2\*ATTRIBUTE.NameLength，属性名后接属性值，所以data run的偏移为0x40+2\*ATTRIBUTE.NameLength）：



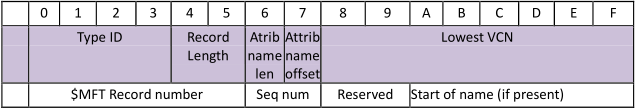
|  |
| --- |
| **/\* 非常驻属性头部 \*/**  **typedef struct {**  **ATTRIBUTE Attribute; // 16@0x00**  **ULONGLONG StartVCN; // 8@0x10**  **ULONGLONG LastVCN; // 8@0x18**  **USHORT RunlistOffset; // 2@0x20**  **UCHAR CompressionUnit; // 若该成员为0，该属性未被压缩，2@0x22**  **UCHAR AlignmentOrReserved[4]; // 4@0x24**  **ULONGLONG AllocatedSize; // 分配了多少磁盘空间用于容纳该属性值，8@0x28**  **ULONGLONG DataSize; // 属性值的真实大小，8@0x30**  **ULONGLONG InitializedSize; // 属性值的初始大小（比如压缩前），8@0x38**  **ULONGLONG CompressedSize; // 压缩后的大小，被压缩才有该成员，8@0x40**  **} NONRESIDENT\_ATTRIBUTE, \*PNONRESIDENT\_ATTRIBUTE; // 72 bytes** |

3.5 $STANDARD\_INFORMATION（0x10）



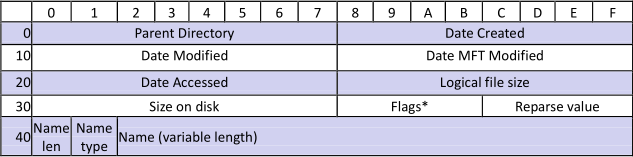
|  |
| --- |
| **/\* 文件标准信息属性 \*/**  **typedef struct {**  **ULONGLONG CreationTime; // 8@0x00**  **ULONGLONG ChangeTime; // 8@0x08**  **ULONGLONG LastWriteTime; // 8@0x10**  **ULONGLONG LastAccessTime; // 8@0x18**  **ULONG FileAttributes; // 文件的属性，包括只读/归档等，4@0x20**  **ULONG MaxVersions; // 4@0x24**  **ULONG VersionNumber; // 4@0x28**  **ULONG ClassId; // 4@0x2C**  **ULONG QuotaId; // NTFS 3.0，4@0x30**  **ULONG SecurityId; // NTFS 3.0，4@0x34**  **ULONGLONG QuotaCharge; // NTFS 3.0，8@0x38**  **USN Usn; // NTFS 3.0，update sequence number，8@0x40**  **} STANDARD\_INFORMATION, \*PSTANDARD\_INFORMATION; // 72 bytes** |

3.6 $ATTRIBUTE\_LIST（0x20）



|  |
| --- |
| **/\* 文件属性列表属性 \*/**  **typedef struct {**  **ATTRIBUTE\_TYPE AttributeType; // 4@0x00**  **USHORT Length; // 属性列表条目的大小，2@0x04**  **UCHAR NameLength; // 如果有属性名，属性名的大小，1@0x06**  **UCHAR NameOffset; // 属性名的内部偏移，1@0x07**  **ULONGLONG LowVcn; // starting vcn，8@0x08**  **ULONGLONG FileReferenceNumber; // $MFT record num + sequence num，8@0x10**  **USHORT AttributeNumber; // 该属性实例的数字标识符，2@0x18**  **USHORT AlignmentOrReserved[3]; // 6@0x1A**  **} ATTRIBUTE\_LIST, \*PATTRIBUTE\_LIST; // 32 bytes** |

3.7 $FILE\_NAME（0x30）



|  |
| --- |
| **/\* 文件名属性 \*/**  **typedef struct {**  **ULONGLONG DirectoryFileReferenceNumber; // 父节点的FRN 8@0x00**  **ULONGLONG CreationTime; // 文件创建时间，8@0x08**  **ULONGLONG ChangeTime; //最近一次被修改的时间，8@0x10**  **ULONGLONG LastWriteTime; // 最近一次被写的时间， 8@0x18**  **ULONGLONG LastAccessTime; // 最近一次被访问的时间，8@0x20**  **ULONGLONG AllocatedSize; // 分配多少磁盘空间用于容纳该属性值，8@0x28**  **ULONGLONG DataSize; // 属性值的真实大小，8@0x30**  **ULONG FileAttributes; // 文件的属性，4@0x38**  **ULONG ReparseValue; // 4@0x3C**  **UCHAR NameLength; // 文件名长度 L，1@0x40**  **UCHAR NameType; // 文件名类型 0x01=长, 0x02=短（小于512字节），1@0x41**  **WCHAR Name[1]; // 文件名，2 \* L @0x42**  **} FILENAME\_ATTRIBUTE, \*PFILENAME\_ATTRIBUTE; // 66 + 2L bytes** |

3.8 $OBJECT\_ID（0x30）

每条MFT记录都有唯一的GUID，是Birth Volume Id, Birth Object Id和Domain Id的统称。该属性最大256字节。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | GUID Object Id | | | | | | | | | | | | | | | |
| 10 | GUID Birth Volume Id | | | | | | | | | | | | | | | |
| 20 | GUID Birth Object Id | | | | | | | | | | | | | | | |
| 30 | GUID Domain Id | | | | | | | | | | | | | | | |

|  |
| --- |
| **/\* 对象ID属性 \*/**  **typedef struct {**  **GUID ObjectId; // 分配给文件的唯一标识符, 16@0x00**  **union {**  **struct {**  **GUID BirthVolumeId; // 文件第一次被创建，所在卷ID**  **GUID BirthObjectId; // 文件第一次被创建时，分配的ID**  **GUID DomainId; // 保留**  **};**  **UCHAR ExtendedInfo[48];**  **};**  **} OBJECTID\_ATTRIBUTE, \*POBJECTID\_ATTRIBUTE; // 64 bytes** |

3.9 $VOLUME\_INFORMATION（0x70）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | 0x0000 | | | | | | | | Major Version | Minor Version | Flags | | 0x0000 | | | |

|  |
| --- |
| **/\* 卷信息 \*/**  **typedef struct {**  **ULONG Unknown[2]; // 8@0x00**  **UCHAR MajorVersion; // NTFS主要版本号**  **UCHAR MinorVersion; // NTFS次要版本号**  **USHORT Flags; // 0x0001 = VolumeIsDirty**  **} VOLUME\_INFORMATION, \*PVOLUME\_INFORMATION; // 12 bytes** |

3.10 $INDEX\_ROOT（0x90）

|  |
| --- |
|  |
| Index node header |
| Index entry |
| Next index entry |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | Attribute type | | | | Collation rule | | | | Bytes/index record | | | | Clus/index record | Unknown | | |
| 10 | First entry offset | | | | IndexBlockLength | | | | AllocatedSize | | | | Sub-node Flags | Padding | | |
| 20 | FileReferenceNumber | | | | | | | | Entry length | | Stream length | | LastEntry Flags | Padding | | |
| 30 | Stream (If last entry flag is not set) | | | | | | | | | | | | | | | |
| S0 | VCN of the sub-node in the index allocation (if sub-node flag is set) | | | | | | | | | | | | | | | |
| X0 | Next Index Entry(Y bytes) | | | | | | | | | | | | | | | |

|  |
| --- |
| **/\* 根目录 \*/**  **typedef struct {**  **ATTRIBUTE\_TYPE Type; // 4@0x00**  **ULONG CollationRule; // 4@0x04**  **ULONG BytesPerIndexBlock; // 每个索引块的字节数, 4@0x08**  **UCHAR ClustersPerIndexBlock; // 每个索引块的簇数, 1@0x0C**  **UCHAR Unknown[3]; // 3@0x0D**  **DIRECTORY\_INDEX DirectoryIndex; // 16@0x10**  **} INDEX\_ROOT, \*PINDEX\_ROOT; // 16 + 16 bytes** |

|  |
| --- |
| **/\* 目录索引 DIRECTORY\_INDEX => Index Node Header\*/**  **typedef struct {**  **ULONG EntriesOffset; // 第一条记录（DIRECTORY\_ENTRY）的内部字节偏移量**  **ULONG IndexBlockLength; // Total size of the Index Entries**  **ULONG AllocatedSize; // 分配了多少磁盘空间给这个索引块**  **ULONG Flags; // Non-leaf node Flag, 0x00 = Small directory, 无子节点, 0x01 = Large directory, 有子节点**  **} DIRECTORY\_INDEX, \*PDIRECTORY\_INDEX; // 16 bytes** |

|  |
| --- |
| **/\* 目录条目 DIRECTORY\_ENTRY => Index entry \*/**  **typedef struct {**  **ULONGLONG FileReferenceNumber; // 该条目描述的文件的FRN**  **USHORT Length; // 该索引条目大小**  **USHORT AttributeLength; // 被索引的属性的字节大小**  **/\* 0x01 = Has trailing VCN, 0x02 = 目录块中最后一个条目**  **若flags = 0x01,那么该 entry 最后8个字节表示包含此条目前一个条目的目录块的VCN \*/**  **ULONG Flags;**  **} DIRECTORY\_ENTRY, \*PDIRECTORY\_ENTRY;** |

3.11 $REPARSE\_POINT (0xC0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | Reparse type | | | | Reparse data len | | Padding | | Reparse data |  |  |  |  |  |  |  |

|  |
| --- |
| **/\* 重解析点 \*/**  **typedef struct {**  **ULONG ReparseTag;**  **USHORT ReparseDataLength;**  **USHORT Reserved;**  **UCHAR ReparseData[1];**  **} REPARSE\_POINT, \*PREPARSE\_POINT; // 9+ bytes** |

3.12 $EA\_INFORMATION (0xD0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | EaLength | | | | EaQueryLength | | | |  |  |  |  |  |  |  |  |

|  |
| --- |
| **/\* extended attribute information \*/**  **typedef struct {**  **ULONG EaLength;**  **ULONG EaQueryLength;**  **} EA\_INFORMATION, \*PEA\_INFORMATION; // 8 bytes** |

3.13 $EA (0xE0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | Next Entry Offset | | | | Flag | EaName Length | EaValue Length | | Ea Name |  |  |  |  |  |  |  |

|  |
| --- |
| **/\* EA属性 \*/**  **typedef struct {**  **ULONG NextEntryOffset;**  **UCHAR Flags;**  **UCHAR EaNameLength;**  **USHORT EaValueLength;**  **CHAR EaName[1];**  **// UCHAR EaData[];**  **} EA\_ATTRIBUTE, \*PEA\_ATTRIBUTE; // 9+ bytes** |

1. 文件

NTFS中的一切都是文件，文件类型有两种，即元文件和普通文件，元文件存NTFS信息，普通文件存用户数据。文件的序列号（sequence number）总是和其所在MFT记录号（record number）相同。

表1 NTFS元文件

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Inode** | **文件名** | **OS** | **属性** | **描述** |
| **0** | **$MFT** |  | 0x10，0x30，0x80，0xB0 | MFT |
| **1** | **$MFTMirr** |  | 0x10，0x30，0x80 | MFT至少前4条记录的备份（视簇大小） |
| **2** | **$LogFile** |  | 0x10，0x30，0x80 | 日志 |
| **3** | **$Volume** |  | 0x10，0x30，0x50，0x60，0x70，0x80 | Serial number，创建时间，dirty标识 |
| **4** | **$AttrDef** |  | 0x10，0x30，0x50，0x80 | 属性定义 |
| **5** | **. (dot)** |  | 0x10，0x30，0x50，0x80，0x90，0xA0，0xB0 | 磁盘根目录 |
| **6** | **$Bitmap** |  | 0x10，0x30，0x80 | 标识簇是否空闲的位图 |
| **7** | **$Boot** |  | 0x10，0x30，0x50，0x80 | 启动扇区 |
| **8** | **$BadClus** |  | 0x10，0x30，0x80，0x80 | 坏簇 |
| **9** | **$Quota** | NT |  |  |
| **9** | **$Secure** | 2K | 0x10，0x30，0x80，0x90，0x90，0xA0，0xA0，0xB0，0xB0 | 卷所使用的安全标识符 |
| **10** | **$UpCase** |  | 0x10，0x30，0x80 |  |
| **11** | **$Extend** | 2K | 0x10，0x30，0x90 |  |
| **12-15** | **<Unused>** |  |  | 标为使用中但是为空 |
| **16-23** | **<Unused>** |  |  | 标为未使用 |
| **Any** | **$ObjId** | 2K | 0x10，0x30，0x90，0xA0，0xB0 | 为每个文件分配的唯一ID |
| **Any** | **$Quata** | 2K | 0x10，0x30，0x90，0x90，0xA0，0xA0，0xB0，0xB0 | Quata信息 |
| **Any** | **$Reparse** | 2K | 0x10，0x30，0x90，0xA0，0xB0 |  |
| **Any** | **$UsnJrnl** | 2K | 0x10，0x30，0x80，0x80 |  |
| **>24** | **A file** |  |  | An ordinary file |
| **>24** | **A dir** |  |  | An ordinary directory |
| **……** | **……** |  |  | …… |

4.1 $AttrDef (4)

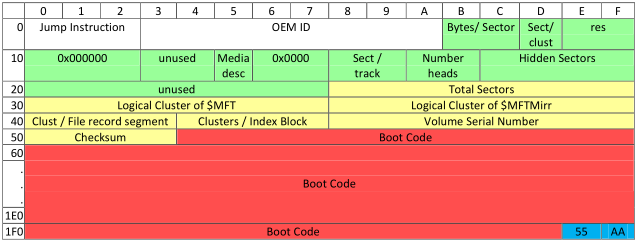
表1 NTFS 元文件AttrDef 布局

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | AttributeName | | | | | | | | | | | | | | | |
| … |
| 80 | AttributeNumber | | | | Display rule | | | | Collation rule | | | | Flags | | | |
| 90 | Minimum Size | | | | | | | | Maximum Size | | | | | | | |

|  |
| --- |
| **/\* $AttrDef, 在 MFT 中的 Record number = 4 \*/**  **typedef struct {**  **WCHAR AttributeName[64]; // 128@0x00**  **ULONG AttributeNumber; // 4@0x80**  **ULONG Unknown[2]; // 8@0x84**  **ULONG Flags; // 4@0x8C**  **ULONGLONG MinimumSize; // 8@0x90**  **ULONGLONG MaximumSize; // 8@0x90**  **} ATTRIBUTE\_DEFINITION, \*PATTRIBUTE\_DEFINITION; // 160 bytes** |

4.2 $Boot（7）

表1 NTFS 元文件Boot sector 布局



Boot sector 数据结构（字节@偏移量）：

|  |
| --- |
| **/\* boot sector \*/**  **typedef struct {**  **UCHAR Jump[3]; // 3@0x00**  **UCHAR Format[8]; // NTFS, 8@0x03**  **USHORT BytesPerSector; // 2@0x0B**  **UCHAR SectorsPerCluster; // 1@0x0C**  **UCHAR Unused1[7]; // 7@0x0E**  **UCHAR MediaType; // 介质描述符，硬盘为0xf8, 1@0x15**  **USHORT Unused2; // 2@0x16**  **USHORT SectorsPerTrack; // 每道扇区数，一般为0x3f, 2@0x18**  **USHORT NumberOfHeads; // 磁头数, 2@0x1A**  **ULONGLONG Unused3; // 8@0x1C**  **ULONG Unused4; // unused,总是80 00 80 00，4@0x24**  **ULONGLONG TotalSectors; // 该分区总扇区数, 8@0x28**  **ULONGLONG MftStartLcn; // MFT表的起始簇号LCN, 8@0x30**  **ULONGLONG MftMirrStartLcn; // MFT备份表的起始簇号LCN, 8@0x38**  **ULONG ClustersPerFileRecord; // 4@0x40**  **ULONG ClustersPerIndexBlock; // 4@0x44**  **ULONGLONG VolumeSerialNumber; // 卷序列号, 8@0x48**  **UCHAR Code[0x1AE]; // 0x50 ~ 0x1FD**  **USHORT BootSignature; // 2@0x1FE**  **} BOOT\_BLOCK, \*PBOOT\_BLOCK; // 512 bytes** |

1. 记录

5.1 Index Record

Index record的大小在Index Root中定义，默认4k。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | INDX | | | | UsaOffset | | UsaCount(S) | | $LogFile sequence number | | | | | | | |
| 10 | IndexBlockVcn | | | | | | | | Index Node Header | | | | | | | |
| 20 | Index Node Header | | | | | | | | Up seq | | Up seq arr(2S-2) | | | | | |
| X0 | Index entry | | | | | | | | | | | | | | | |
| Y0 | Next index entry | | | | | | | | | | | | | | | |

|  |
| --- |
| **/\* NTFS RECORD HEADER \*/**  **typedef struct {**  **ULONG Type; // NTFS记录的类型，FILE/BAAD/INDX/CHKD/HOLE**  **USHORT UsaOffset; // Update Sequence Array 在该结构中的偏移量**  **USHORT UsaCount; // Size in words of the Update Sequence (S)**  **USN Usn; // Update Sequence Number of the record, $LogFile sequence number**  **} NTFS\_RECORD\_HEADER, \*PNTFS\_RECORD\_HEADER; // 16 bytes** |

|  |
| --- |
| **/\* 索引块头部 \*/**  **typedef struct {**  **NTFS\_RECORD\_HEADER Ntfs; // 类型为 INDX 的 NTFS\_RECORD\_HEADER**  **ULONGLONG IndexBlockVcn; // VCN of this Index record in the Index Allocation**  **DIRECTORY\_INDEX DirectoryIndex; // Index Node Header**  **} INDEX\_BLOCK\_HEADER, \*PINDEX\_BLOCK\_HEADER;** |

5.2 MFT File Record

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 00 | FILE | | | | UsaOffset | | UsaCount(S) | | $LogFile sequence number | | | | | | | |
| 10 | Seq no | | Hardlink count | | 1st attrib offset | | Flags | | Used  Size of the record | | | | Allocated size of the record | | | |
| 20 | File reference to the base file record | | | | | | | | Nextattrib ID | | Win XP | | Win XP | | | |
| X0 | usn | | usa | | | | | | | | | | | | | |
| Y0 | usa | | | | | | | | Fist attribute | | | | | | | |

|  |
| --- |
| **/\* NTFS RECORD HEADER \*/**  **typedef struct {**  **ULONG Type; // NTFS记录的类型，FILE/BAAD/INDX/CHKD/HOLE**  **USHORT UsaOffset; // Update Sequence Array 在该结构中的偏移量**  **USHORT UsaCount; // Size in words of the Update Sequence (S)**  **USN Usn; // Update Sequence Number of the record, $LogFile sequence number**  **} NTFS\_RECORD\_HEADER, \*PNTFS\_RECORD\_HEADER; // 16 bytes** |

|  |
| --- |
| **/\* FILE RECORD\_HEADER \*/**  **typedef struct {**  **NTFS\_RECORD\_HEADER Ntfs; // when Type = FILE**  **USHORT SequenceNumber; // MFT entry 被重用次数**  **USHORT LinkCount; // The number of directory links to the MFT entry**  **USHORT AttributesOffset; // 第一个属性在此 MFT entry 中的偏移**  **USHORT Flags; // 0x0001 = InUse, 0x0002 = Directory**  **ULONG BytesInUse; // 已被该 MFT entry 使用的字节数**  **ULONG BytesAllocated; // 分配给该 MFT entry 的字节数**  **ULONGLONG BaseFileRecord; /\*If the MFT entry contains attributes that overflowed a base MFT entry, this member contains the file reference number of the base entry; otherwise , it contains zero.\*/**  **USHORT NextAttributeNumber; // number assigned to next attribute**  **} FILE\_RECORD\_HEADER, \*PFILE\_RECORD\_HEADER; // 42 bytes** |