

PE文件学习笔记（一）：DOS头与PE头解析

原创 Apollon_krj 于 2017-08-10 21:01:32 发布 阅读量1.8w 收藏 85 点赞数 27

版权

分类专栏: COFF PE/ELF 文章标签: dos windows 移植

在Windows下所谓PE文件即Portable Executable，意为可移植的可执行的文件。常见的.EXE、.DLL、.OCX、.SYS、.COM都是PE文件。PE文件有一个共同特点：前两个字节为4D 5A（MZ）。如果一个文件前两个字节不是4D 5A则其肯定不是可执行文件。比如用16进制文本编辑器打开一个“.xls”文件其前两个字节为：0XD0 0XCF；打开一个“.pdf”其前两个字节为：0X25 0X50。

PE文件结构：**DOS头+PE头+节表+.data/.rdata/.text**。而今天我们就来具体了解一下PE文件的DOS头和PE头的结构成员与部分成员的作用。**注意**：一个exe文件本身是一个PE文件，但是由于包含dll库，所以一个exe文件也是许多PE文件组成的(包含多个dll)一个PE文件。

1、DOS头：共40H(64字节)

DOS头中声明用的寄存器(我们可以看到e_ss、e_sp、e_ip、e_cs还是16位的寄存器)，所以在32位/64为系统中用到的只有两个成员了（第一个和最后一个）：

①**e_magic**：判断一个文件是不是PE文件；

②**e_lfanew**：相对于文件首的偏移量，用于找到PE头；

具体结构如下（前面的**十六进制**数表示该成员相对于结构的偏移量，WORD2字节变量、DWORD4字节变量）：

```
1 //注释掉的不需要重点分析
2 struct _IMAGE_DOS_HEADER{
3     0x00 WORD e_magic;        //※Magic DOS signature MZ(4Dh 5Ah):MZ标记:用于标记是否是可执行文件
4     //0x02 WORD e_cblp;      //Bytes on last page of file
5     //0x04 WORD e_cp;        //Pages in file
6     //0x06 WORD e_crlc;      //Relocations
7     //0x08 WORD e_cparhdr;    //Size of header in paragraphs
8     //0x0A WORD e_minalloc;   //Minimun extra paragraphs needs
9     //0x0C WORD e_maxalloc;   //Maximun extra paragraphs needs
10    //0x0E WORD e_ss;         //intial(relative)SS value
11    //0x10 WORD e_sp;         //intial SP value
12    //0x12 WORD e_csum;       //Checksum
13    //0x14 WORD e_ip;         //intial IP value
14    //0x16 WORD e_cs;         //intial(relative)CS value
15    //0x18 WORD e_lfarlc;     //File Address of relocation table
16    //0x1A WORD e_ovno;       //Overlay number
17    //0x1C WORD e_res[4];     //Reserved words
18    //0x24 WORD e_oemid;      //OEM identifier(for e_oeminfo)
19    //0x26 WORD e_oeminfo;    //OEM information;e_oemid specific
```

```

20 //0x28 WORD e_res2[10]; //Reserved words
21 0x3C DWORD e_lfanew; //※Offset to start of PE header:定位PE文件, PE头相对于文件的偏移量
22 };

```

我们查看下面所示unins000.exe文件的结构信息:

unins000.exe x																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	4D	5A	50	00	02	00	00	00	04	00	0F	00	FF	FF	00	00	; MZP.....
00000010h:	B8	00	00	00	00	00	00	00	40	00	1A	00	00	00	00	00	; ?.....@.....
00000020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000030h:	49	6E	55	6E	00	00	00	00	00	00	00	00	00	01	00	00	; InUn.....
00000040h:	BA	10	00	0E	1F	B4	09	CD	21	B8	01	4C	CD	21	90	90	; ?...???L?悖
00000050h:	54	68	69	73	20	70	72	6F	67	72	61	6D	20	6D	75	73	; This program mus
00000060h:	74	20	62	65	20	72	75	6E	20	75	6E	64	65	72	20	57	; t be run under W
00000070h:	69	6E	33	32	0D	0A	24	37	00	00	00	00	00	00	00	00	; in32.. \$7.....
00000080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000a0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000b0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000c0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000d0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000e0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
000000f0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000100h:	50	45	00	00	4C	01	08	00	29	E0	BC	56	00	00	00	00	; PE..L....)嗦V.....

64字节 (共4行) 的DOS头, 第一个成员2个字节是可执行文件的标志信息; 最后一个成员4字节是PE头的偏移地址为00000100H, 我们可以根据00000100H来获取PE头的地址。而DOS头和PE头中间的空余位置是一些垃圾值以及编译器填充的一些“is program cannot be run in DOS mode.”或“This program must be run under Win32”等信息。

2、PE头:

PE头分为标准PE头和可选PE头, 其同为NT结构的成员:

```

//NT头
//pNTHHeader = dosHeader + dosHeader->e_lfanew;
1 struct _IMAGE_NT_HEADERS{
2     0x00 DWORD Signature; //PE文件标识:ASCII的"PE\0\0"
3     0x04 _IMAGE_FILE_HEADER FileHeader;
4
5

```

```

6
7
0x18 _IMAGE_OPTIONAL_HEADER OptionalHeader;
};

```

根据DOS头的e_lfanew成员我们就可以找到NT头，NT头的第一个成员是“PE\0\0”（0X50 0X45 0X00 0X00四字节的签名，可以在上图00000100H地址处观察），后两个成员则分别是标准PE头（_IMAGE_FILE_HEADER）和可选PE头（_IMAGE_OPTIONAL_HEADER）。

(1)、_IMAGE_FILE_HEADER:

```

1 //标准PE头:最基础的文件信息,共20字节
2 struct _IMAGE_FILE_HEADER{
3     0x00 WORD Machine;           //※程序执行的CPU平台:0X0:任何平台, 0X14C:intel i386及后续处理器
4     0x02 WORD NumberOfSections; //※PE文件中区块数量
5     0x04 DWORD TimeDateStamp;    //时间戳:连接器产生此文件的时间距1969/12/31-16:00P:00的总秒数
6     //0x08 DWORD PointerToSymbolTable; //COFF符号表格的偏移位置。此字段只对COFF除错信息有用
7     //0x0c DWORD NumberOfSymbols;    //COFF符号表格中的符号个数。该值和上一个值在release版本的程序里为0
8     //0x10 WORD SizeOfOptionalHeader; //IMAGE_OPTIONAL_HEADER结构的大小(字节数):32位默认E0H,64位默认F0H(可修改)
9     0x12 WORD Characteristics;    //※描述文件属性,eg:
10                                     //单属性(只有1bit为1): #define IMAGE_FILE_DLL 0x2000 //File is a DLL.
11                                     //组合属性(多个bit为1,单属性或运算):0X010F 可执行文件
12 };

```

我们依旧来看unins000.exe的文件信息:

```

00000100h: 50 45 00 00 4C 01 08 00 29 E0 BC 56 00 00 00 00 ; PE..L...)嗦V....
00000110h: 00 00 00 00 E0 00 8F 81 0B 01 02 19 00 F8 0F 00 ; .....?弦.....?_krj

```

首先四字节是NT第一个签名成员“PE\0\0”。接着便是2字节的CPU平台信息：014C，即X86平台（其它平台可见下图）：

Value	Meaning
IMAGE_FILE_MACHINE_I386 0x014c	x86
IMAGE_FILE_MACHINE_IA64 0x0200	Intel Itanium
IMAGE_FILE_MACHINE_AMD64 0x8664	x64

我们可以查看一下其它的other.exe文件，如下所示，其CPU平台信息为8664即X64：

```
00000080h: 50 45 00 00 64 86 0C 00 00 00 00 00 00 00 00 00 ; PE..d?.....
00000090h: 00 00 00 00 F0 00 2E 02 0B 02 02 1C 00 3A 00 00 ; .....?.....
```

然后是区块数量0008H即8个块；第三个成员为时间戳：**56BCE029H**(即十进制1455218729)，转换成北京时间为**2016-02-12 03:25:29** ([时间戳在线转换](#))；第四个和第五个成员各占4字节且均为0；第六个成员占2字节为默认值E0H（即可选PE头的大小为224字节,other.exe中为X64默认值F0H）；最后一个属性成员占2字节为“818F”，该成员是按bit位来看的，属性值也是多个属性的组合（或运算）。每一位的信息如下：

#define IMAGE_FILE_RELOCS_STRIPPED	0x0001	Relocation info stripped from file. (可重定位信息被移去)
#define IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	File is executable (i.e. no unresolved external references). (文件可执行)
#define IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	Line numbers stripped from file. (行号被移去)
#define IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	Local symbols stripped from file. (符号被移去)
#define IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010	Agressively trim working set. (主动调整工作区)
#define IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020	App can handle >2gb addresses. (高地址警告)
#define IMAGE_FILE_BYTES_REVERSED_LO	0x0080	Bytes of machine word are reversed. (处理机的低字节是相反的)
#define IMAGE_FILE_32BIT_MACHINE	0x0100	32 bit word machine. (32位机器)
#define IMAGE_FILE_DEBUG_STRIPPED	0x0200	Debugging info stripped from file in .DBG file (DBG的调试信息被移去)
#define IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	If Image is on removable media, copy and run from the swap file. (如果映像文件在可移动媒体中，则复制到交换文件后再运行)
#define IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	If Image is on Net, copy and run from the swap file. (如果映像文件在网络上，则复制到交换文件后再运行)
#define IMAGE_FILE_SYSTEM	0x1000	System File. (系统文件)
#define IMAGE_FILE_DLL	0x2000	File is a DLL. (文件是DLL文件)
#define IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	File should only be run on a UP machine (文件只能运行在单处理机上)
#define IMAGE_FILE_BYTES_REVERSED_HI	0x8000	Bytes of machine word are reversed. (处理机的高位字节是相反的)

818F = 8000 | 0100 | 0080 | 0008 | 0004 | 0002 | 0001即：

```
IMAGE_FILE_BYTES_REVERSED_HI |
IMAGE_FILE_32BIT_MACHINE |
IMAGE_FILE_BYTES_REVERSED_LO |
IMAGE_FILE_RELOCS_STRIPPED |
IMAGE_FILE_EXECUTABLE_IMAGE |
IMAGE_FILE_LINE_NUMS_STRIPPED |
IMAGE_FILE_LOCAL_SYMS_STRIPPED
```

(2)、_IMAGE_OPTIONAL_HEADER:

可选PE头紧接着标准PE头，其大小在标准PE头中给出：大小为E0H即224字节。_IMAGE_OPTIONAL_HEADER结构如下所示：

```

1  //可选PE头
2  struct _IMAGE_OPTIONAL_HEADER{
3      0x00 WORD Magic;                //※幻数(魔数), 0x0107:ROM image,0x010B:32位PE, 0x020B:64位PE
4      //0x02 BYTE MajorLinkerVersion; //连接器主版本号
5      //0x03 BYTE MinorLinkerVersion; //连接器副版本号
6      0x04 DWORD SizeOfCode;          //所有代码段的总和大小,注意:必须是FileAlignment的整数倍,存在但没用
7      0x08 DWORD SizeOfInitializedData; //已经初始化数据的大小,注意:必须是FileAlignment的整数倍,存在但没用
8      0x0c DWORD SizeOfUninitializedData; //未经初始化数据的大小,注意:必须是FileAlignment的整数倍,存在但没用
9      0x10 DWORD AddressOfEntryPoint; //※程序入口地址OEP,这是一个RVA(Relative Virtual Address),通常会落在.textsection,此字段对于DLLs/EXEs都适用。
10     0x14 DWORD BaseOfCode;          //代码段起始地址(代码基址),(代码的开始和程序无必然联系)
11     0x18 DWORD BaseOfData;          //数据段起始地址(数据基址)
12     0x1c DWORD ImageBase;           //※内存镜像基址(默认装入起始地址),默认为4000H
13     0x20 DWORD SectionAlignment;    //※内存对齐:一旦映像到内存中,每一个section保证从一个「此值之倍数」的虚拟地址开始
14     0x24 DWORD FileAlignment;       //※文件对齐:最初是200H,现在是1000H
15     //0x28 WORD MajorOperatingSystemVersion; //所需操作系统主版本号
16     //0x2a WORD MinorOperatingSystemVersion; //所需操作系统副版本号
17     //0x2c WORD MajorImageVersion;          //自定义主版本号,使用连接器的参数设置,eg:LINK /VERSION:2.0 myobj.obj
18     //0x2e WORD MinorImageVersion;          //自定义副版本号,使用连接器的参数设置
19     //0x30 WORD MajorSubsystemVersion;      //所需子系统主版本号,典型数值4.0(Windows 4.0/即Windows 95)
20     //0x32 WORD MinorSubsystemVersion;      //所需子系统副版本号
21     //0x34 DWORD Win32VersionValue;         //总是0
22     0x38 DWORD SizeOfImage;               //※PE文件在内存中映像总大小,sizeof(ImageBuffer),SectionAlignment的倍数
23     0x3c DWORD SizeOfHeaders;             //※DOS头(64B)+PE标记(4B)+标准PE头(20B)+可选PE头+节表的总大小,按照文件对齐(FileAlignment的倍数)
24     0x40 DWORD CheckSum;                  //PE文件CRC校验和,判断文件是否被修改
25     //0x44 WORD Subsystem;                 //用户界面使用的子系统类型
26     //0x46 WORD DllCharacteristics;        //总是0
27     0x48 DWORD SizeOfStackReserve;         //默认线程初始化栈的保留大小
28     0x4c DWORD SizeOfStackCommit;         //初始化时实际提交的线程栈大小
29     0x50 DWORD SizeOfHeapReserve;          //默认保留给初始化的process heap的虚拟内存大小
30     0x54 DWORD SizeOfHeapCommit;          //初始化时实际提交的process heap大小
31     //0x58 DWORD LoaderFlags;              //总是0
32     0x5c DWORD NumberOfRvaAndSizes; //目录项数目:总为0x00000010H(16)
33     0x60 _IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];//#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES 16
34 };

```

继续接着标准PE头来分析：

00000110h: 00 00 00 00 E0 00 8F 81 0B 01 02 19 00 F8 0F 00

00000120h: 00 94 03 00 00 00 00 00 6C 05 10 00 00 10 00 00

00000130h: 00 10 10 00 00 00 40 00 00 10 00 00 00 02 00 00

00000140h: 05 00 00 00 06 00 00 00 05 00 00 00 00 00 00 00

00000150h: 00 50 14 00 00 04 00 00 3F 17 14 00 02 00 40 81

00000160h: 00 00 10 00 00 40 00 00 00 00 10 00 00 10 00 00

00000170h: 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

00000180h: 00 C0 10 00 40 38 00 00 00 20 11 00 14 25 03 00

00000190h: 00 00 00 00 00 00 00 00 00 90 13 00 90 16 00 00

000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

000001b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

000001c0h: 00 10 11 00 18 00 00 00 00 00 00 00 00 00 00 00

000001d0h: 00 00 00 00 00 00 00 00 80 CA 10 00 8C 08 00 00

000001e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

000001f0h: 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

前
30
个
成
员
60H

最后一个
成员：结
构体数组
128字节
(16*8)

第1个成员（Magic，2Byte）：幻数010B，表示该文件为32位PE，其它情况如下：

#define IMAGE_NT_OPTIONAL_HDR32_MAGIC	0x10b	The file is an executable image. (32位)
#define IMAGE_NT_OPTIONAL_HDR64_MAGIC	0x20b	The file is an executable image. (64位)
#define IMAGE_ROM_OPTIONAL_HDR_MAGIC	0x107	The file is a ROM image.

第4个成员（SizeOfCode，4Byte）：代码段总大小为000FF800H；

第5个成员（SizeOfInitializedData，4Byte）：已初始化数据大小为00039400H；

第6个成员（SizeOfUninitializedData，4Byte）：未初始化数据大小为0，即均已初始化；

第7个成员（AddressOfEntryPoint，4Byte）：程序入口地址OEP=0010056CH；

第8个成员（BaseOfCode，4Byte）：代码段基址 = 00001000H；

第9个成员（BaseOfData，4Byte）：数据段基址 = 00101000H；

第10个成员（ImageBase，4Byte）：内存镜像基址 = 00400000H，这是一个默认的值；

第11个成员（SectionAlignment，4Byte）：内存对齐 = 00001000H，即4096字节；

第12个成员（FileAlignment，4Byte）：文件对齐 = 00000200H，即512字节（文件对齐和内存对齐的目的是提高效率）；

第20个成员（SizeOfImage，4字节）：PE映像在内存中总大小 = 00145000H，是SectionAlignment的325倍（整数倍）；

第21个成员（SizeOfHeaders，4字节）：所有头+节表总大小 = 00000400H；

第22个成员（Checksum，4字节）：PE文件CRC校验和 = 0014173FH

第25个成员（SizeOfStackReserve，4字节）：为线程初始栈保留虚拟内存的默认值 = 00100000H = 1MB；

第26个成员（SizeOfStackCommit，4字节）：为线程的初始栈提交的实际虚拟内存大小 = 00004000H = 16KB；

第27个成员（SizeOfHeapReserve，4字节）：为进程的初始堆保留虚拟内存的默认值 = 00100000H = 1MB；

第28个成员 (SizeOfHeapCommit, 4字节) : 为进程的初始堆提交的实际虚拟内存大小 = 00001000H = 4KB;

第30个成员 (NumberOfRvaAndSizes, 4字节) : 目录项总数默认 = 00000010H = 16个。

最后一个成员为_IMAGE_DATA_DIRECTORY的结构体数组 (本次只分析DOS头和PE头基本成员, 该结构体数组以后重点分析), 该结构体成员如下:

```
1 struct _IMAGE_DATA_DIRECTORY{
2     DWORD   VirtualAddress;
3     DWORD   Size;
4 };
5 //占用16*8 = 128Byte = 80H = E0H(可选PE头默认大小) - 60H(前面所有成员固定占用大小)
```

官方对于Image  结构的解释: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms680198\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680198(v=vs.85).aspx)

3、几个重点的数据成员分析:

(1)、文件对齐 (FileAlignment) 和内存对齐 (SectionAlignment) :

一个PE文件加载进内存中可能大于在硬盘上的大小, 并且无论是在内存中还是硬盘上, 都是分块管理 (分节), 一块和一块存储空间之间是空隙。在硬盘上空隙有可能小于内存中空隙; 在内存中空隙较大 (相较于硬盘)。而存在间隙的原因则是分块管理。

分块的一个原因是节省硬盘: 比如notepad.exe, 由于是早期的程序, 当时硬盘容量比较小, 编译器在生成可执行文件时, 不仅要考虑效率问题使得内存对齐/文件对齐, 还需要设计成节省硬盘空间的结构。所以这种结构遵循的对齐原则: 内存对齐(1000H)和硬盘对齐(200H), 对齐的补充数据(0X0000)便是间隙。硬盘的对齐值较小, 补充间隙自然小, 因此同一个可执行程序在内存中可能比在硬盘上大。但是现今的硬盘空间更大, 所以编译器生成的可执行程序在硬盘上与内存中对齐方式都是1000H。统一对齐为1000H的目的依旧是提高效率。

而分块的另一个目的是节省内存空间, 比如同时在电脑上运行登录多个QQ账号, 就需要运行多次QQ可执行程序。而代码段为只读数据需要一份即可, 数据段则需要为每个账号均开辟一份, , 多个QQ程序共享代码块, 单独使用数据块, 这样就节省了多份代码块的内存。(这些块是使用结构体来维护的, 分块即创建结构体)。

(2)、镜像地址/基址ImageBase的作用:

FileBuffer是磁盘上.exe文件在内存中的一份拷贝, 但是FileBuffer无法直接在内存中运行, 必须经过PE loader(装载器)装载以后成为ImageBuffer。ImageBuffer是FileBuffer的"拉伸"。即".exe→FileBuffer→ImageBuffer"

- ①.exe首地址 (基址) 为0
- ②FileBuffer首地址也为0
- ③ImageBuffer首地址为ImageBase
- ④而真正的程序入口地址是: ImageBase + AddressOfEntryPoint(OEP)

一个exe文件默认镜像地址为400000H（有可能不是，总之有一个默认值），如果一个exe文件中用到了多个dll，而dll文件作为一个PE文件，其默认镜像地址也均是400000H，操作系统不会修改exe的镜像基址。因为.exe先被加载，在.exe中才加载的dll库，由于400000已经被.exe占用，所以装载器会修改dll的镜像基址。而采用ImageBase + OEP的目的也就是：采用偏移地址的方式可以更方便地修改基址，使得任何一个dll文件基址修改后程序依旧不会出错。比如：dll和exe基址有冲突，本只需要将冲突的dll的文件基址修改为600000H（假设是编译器为其分配的是600000H）；如果不采用“基址+偏移地址”的方式，而采用绝对地址，那么要修改的就不是一个基址为600000H了，而是dll中所有的地址统一加上200000H(因为原来默认为400000H)。