



(code、data、resource 存在不同节区，节区头就由上面的结构体组成数组，每个结构体对应一个区)

计算文件偏移

DOS 头

共 64 字节。

```

typedef struct _IMAGE_DOS_HEADER {
    WORD    e_magic; // DOS 签名 "MZ" (0x5A4D), 重要
    WORD    e_cblp; // 字节数 (最后页)
    WORD    e_cp; // 页数
    WORD    e_crlc; // 重定位项数
    WORD    e_cparhdr; // 头部段数
    WORD    e_minalloc; // 最小内存分配

```

```

WORD    e_maxalloc; // 最大内存分配
WORD    e_ss; // 初始 SS 值
WORD    e_sp; // 初始 SP 值
WORD    e_csum; // 校验和
WORD    e_ip; // 初始 IP 值
WORD    e_cs; // 初始 CS 值
WORD    e_lfarlc; // 重定位表偏移
WORD    e_ovno; // 覆盖号
WORD    e_res[4]; // 保留字段
WORD    e_oemid; // OEM 标识符
WORD    e_oeminfo; // OEM 信息
WORD    e_res2[10]; // 保留字段
LONG    e_lfanew; // NT 头偏移 (PE 文件起始位置)，重要
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;

```

00000000:	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....
00000010:	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@
00000020:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030:	00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00

DOS 存根

在 DOS 系统下提示一句话，包含数据和代码，可以修改，但是修改不可覆盖 DOS 头和 NT 头，长度不能随意修改，如果缩短或增长需要修改地址，非常麻烦。

00000040:	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68!..L.!Th
00000050:	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00000060:	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00000070:	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.
00000080:	F3 62 9A E1 B7 03 F4 B2 B7 03 F4 B2 B7 03 F4 B2	.b.....
00000090:	C3 82 F7 B3 BA 03 F4 B2 C3 82 F1 B3 10 03 F4 B2
000000A0:	C3 82 F0 B3 A1 03 F4 B2 C3 82 F5 B3 B4 03 F4 B2
000000B0:	B7 03 F5 B2 E9 03 F4 B2 D1 8B F7 B3 A2 03 F4 B2
000000C0:	D1 8B F0 B3 A6 03 F4 B2 D1 8B F1 B3 FA 03 F4 B2
000000D0:	30 8A F1 B3 B6 03 F4 B2 30 8A F6 B3 B6 03 F4 B2	0.....0.....
000000E0:	52 69 63 68 B7 03 F4 B2 00 00 00 00 00 00 00 00	Rich.....

NT 头

没什么说的。

```

typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature; // PE 签名 "PE\0\0" (0x00004550)
    IMAGE_FILE_HEADER FileHeader; // 文件头
    IMAGE_OPTIONAL_HEADER OptionalHeader; // 可选头 (32/64 位)
}

```

```
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
// 64 位版本 (IMAGE_OPTIONAL_HEADER64)
typedef struct _IMAGE_NT_HEADERS64 {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;
```

NT 头：签名结构体（Signature）

PE 文件核心标识，内容“PE\0\0”。



NT 头：文件头（FileHeader）

固定 20 字节。

```
typedef struct _IMAGE_FILE_HEADER {
    WORD Machine; // 目标 CPU 架构 (如 0x014C=Intel 386)
    WORD NumberOfSections; // 节区数量
    DWORD TimeDateStamp; // 编译时间戳
    DWORD PointerToSymbolTable; // 符号表偏移 (调试用)
    DWORD NumberOfSymbols; // 符号数量
    WORD SizeOfOptionalHeader; // 可选头大小
    WORD Characteristics; // 文件属性 (如可执行/DLL)
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

其中 Characteristics 按 bit 位定义，每一位含义如下：

Bit0	IMAGE_FILE_RELOCS_STRIPPED	0x0001	重定位信息已移除（通常是 EXE）
Bit 1	IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	文件是可执行的
Bit 2	IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	行号信息已移除（已废弃）
Bit 3	IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	符号表已移除（已废弃）
Bit 4	IMAGE_FILE_AGGRESIVE_WS_TRIM	0x0010	优化工作集（已废弃）
Bit 5	IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020	支持 >2GB 地址空间
Bit 7	IMAGE_FILE_BYTES_REVERSED_LO	0x0080	小端字节序（已废弃）

Bit 8	IMAGE_FILE_32BIT_MACHINE	0x0100	32 位架构 (x86)
Bit 9	IMAGE_FILE_DEBUG_STRIPPED	0x0200	调试信息已移除
Bit 10	IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	可从交换设备运行 (已废弃)
Bit 11	IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	可从网络运行 (已废弃)
Bit 12	IMAGE_FILE_SYSTEM	0x1000	系统文件 (如内核驱动)
Bit 13	IMAGE_FILE_DLL	<u>0x2000</u>	<u>这是一个 DLL 文件</u>
Bit 14	IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	仅单处理器运行 (已废弃)
Bit 15	IMAGE_FILE_BYTES_REVERSED_HI	0x8000	大端字节序 (已废弃)

```

000000F0: 50 45 00 00 4C 01 07 00 D1 30 28 68 00 00 00 00 PE..L...0(h...
00000100: 00 00 00 00 E0 00 02 01 0B 01 0E 2C 00 9A 0D 00 .....

```

NT 头：可选头 (OptionalHeader)

长度由文件头里的 `SizeOfOptionalHeader` 确定，32 位 PE 文件通常为 0xE0 (224 字节)，64 位 PE 文件通常为 0xF0 (240 字节)。

```

typedef struct _IMAGE_OPTIONAL_HEADER32 {
    // 标准字段 (所有 PE 文件)
    WORD Magic; // 标识: 0x10B=32 位, 0x20B=64 位
    BYTE MajorLinkerVersion; // 链接器主版本号
    BYTE MinorLinkerVersion; // 链接器次版本号
    DWORD SizeOfCode; // 所有代码段的总大小
    DWORD SizeOfInitializedData; // 已初始化数据的总大小
    DWORD SizeOfUninitializedData; // 未初始化数据 (BSS) 的总大小
    DWORD AddressOfEntryPoint; // 入口点 RVA (相对于 ImageBase)
    DWORD BaseOfCode; // 代码段的起始 RVA
    DWORD BaseOfData; // 数据段的起始 RVA (仅 32 位存在)

    // NT 扩展字段 (Windows 专用)
    DWORD ImageBase; // 进程内存中的优先加载地址

```

```

    DWORD SectionAlignment; // 内存中的节区对齐粒度 (通常 0x1000)
    DWORD FileAlignment; // 文件中的节区对齐粒度 (通常 0x200)
    WORD MajorOperatingSystemVersion; // 要求的最低 OS 主版本
    WORD MinorOperatingSystemVersion; // 要求的最低 OS 次版本
    WORD MajorImageVersion; // 映像主版本号 (用户定义)
    WORD MinorImageVersion; // 映像次版本号 (用户定义)
    WORD MajorSubsystemVersion; // 子系统主版本 (通常 4=Win95)
    WORD MinorSubsystemVersion; // 子系统次版本
    DWORD Win32VersionValue; // 保留 (必须为 0)
    DWORD SizeOfImage; // 映像在内存中的总大小
    DWORD SizeOfHeaders; // 所有头部的总大小 (对齐后)
    DWORD CheckSum; // 校验和 (驱动/DLL 常用)
    WORD Subsystem; // 子系统类型 (1=Native, 2=GUI, 3=CUI)
    WORD DllCharacteristics; // DLL 属性 (如 ASLR/DEP)
    DWORD SizeOfStackReserve; // 初始保留的栈大小
    DWORD SizeOfStackCommit; // 初始提交的栈大小
    DWORD SizeOfHeapReserve; // 初始保留的堆大小
    DWORD SizeOfHeapCommit; // 初始提交的堆大小
    DWORD LoaderFlags; // 保留 (已废弃)
    DWORD NumberOfRvaAndSizes; // 数据目录项数 (通常 16)
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]; // 数据目录表
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;

```

其中的 IMAGE_DATA_DIRECTORY 结构如下:

```

typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD VirtualAddress; // 数据的 RVA (相对虚拟地址)
    DWORD Size; // 数据的大小 (字节数)
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

```

DataDirectory 是数组, 储存 RVA, 下面是数组每一项的含义:

[0] = EXPORT Directory	<u>导出表 (DLL 导出的函数列表)</u> , 重要
[1] = IMPORT Directory	<u>导入表 (依赖的外部 DLL 函数)</u> , 重要
[2] = RESOURCE Directory	资源表
[3] = EXCEPTION Directory	异常处理表
[4] = SECURITY Directory	数字签名
[5] = BASERELOC Directory	重定位表
[6] = DEBUG Directory	调试信息
[7] = COPYRIGHT Directory	架构特定数据
[8] = GLOBALPTR Directory	全局指针寄存器
[9] = TLS Directory	<u>TLS 表</u> , 重要

[A] = LOAD_CONFIG Directory 加载配置表
 [B] = BOUND_IMPORT Directory 绑定导入表
 [C] = IAT Directory 导入地址表 (IAT)
 [D] = DELAY_IMPORT Directory 延迟加载导入表
 [E] = COM_DESCRIPTOR Directory .NET 元数据
 [F] = Reserved Directory 保留, 未使用

00000100:	00 00 00 00 E0 00 02 01	0B 01 0E 2C 00 9A 0D 00,....
00000110:	00 32 02 00 00 00 00 00	C7 11 00 00 00 10 00 002.....
00000120:	00 B0 0D 00 00 00 40 00	00 10 00 00 00 02 00 00@.....
00000130:	06 00 00 00 00 00 00 00	06 00 00 00 00 00 00 00@.....
00000140:	00 10 10 00 00 04 00 00	00 00 00 00 03 00 40 81@.....
00000150:	00 00 10 00 00 10 00 00	00 00 10 00 00 10 00 00@.....
00000160:	00 00 00 00 10 00 00 00	00 00 00 00 00 00 00 00@.....
00000170:	EC 81 0F 00 28 00 00 00	00 00 00 00 00 00 00 00(.....
00000180:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00@.....
00000190:	00 B0 0F 00 B8 4B 00 00	A0 B1 0E 00 38 00 00 00K.....8.....
000001A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00@.....
000001B0:	00 00 00 00 00 00 00 00	B8 B0 0E 00 40 00 00 00@.....
000001C0:	00 00 00 00 00 00 00 00	00 80 0F 00 EC 01 00 00@.....
000001D0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00@.....
000001E0:	00 00 00 00 00 00 00 00	2E 74 65 78 74 00 00 00text...

```

typedef struct _IMAGE_OPTIONAL_HEADER64 {
    // 标准字段 (与 32 位类似)
    WORD Magic; // 标识: 0x20B=64 位
    BYTE MajorLinkerVersion;
    BYTE MinorLinkerVersion;
    DWORD SizeOfCode;
    DWORD SizeOfInitializedData;
    DWORD SizeOfUninitializedData;
    DWORD AddressOfEntryPoint; // 入口点 RVA
    DWORD BaseOfCode; // 代码段起始 RVA
    // BaseOfData 字段在 64 位中不存在!

    // NT 扩展字段
    ULONGLONG ImageBase; // 64 位优先加载地址
    DWORD SectionAlignment;
    DWORD FileAlignment;
    WORD MajorOperatingSystemVersion;
    WORD MinorOperatingSystemVersion;
    WORD MajorImageVersion;
    WORD MinorImageVersion;
    WORD MajorSubsystemVersion;
    WORD MinorSubsystemVersion;
    DWORD Win32VersionValue;
    DWORD SizeOfImage;
  
```



```

    DWORD SizeOfHeaders;
    DWORD CheckSum;
    WORD Subsystem;
    WORD DllCharacteristics;
    ULONGLONG SizeOfStackReserve; // 64 位栈/堆大小
    ULONGLONG SizeOfStackCommit;
    ULONGLONG SizeOfHeapReserve;
    ULONGLONG SizeOfHeapCommit;
    DWORD LoaderFlags;
    DWORD NumberOfRvaAndSizes; // 数据目录项数 (通常 16)
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]; // 数据目录表
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64;

```

节区头

每个节区头固定 40 字节。下面的实际例子里有 7 个节区头，对应的从上面 NT 头中文件头处第二个字可以看到 0x0007（小端序，读的时候倒一下），就是说这里有 7 个节区头。

```

typedef struct _IMAGE_SECTION_HEADER {
    BYTE Name[8]; // 节区名 (如 ".text")，可以填入任何值，只是做参考
    union {
        DWORD PhysicalAddress;
        DWORD VirtualSize; // 内存中的实际大小，重要
    } Misc;
    DWORD VirtualAddress; // 内存中的 RVA，重要
    DWORD SizeOfRawData; // 文件中的大小，重要
    DWORD PointerToRawData; // 文件中的偏移，重要
    DWORD PointerToRelocations; // 重定位表偏移
    DWORD PointerToLinenumbers; // 行号表偏移
    WORD NumberOfRelocations; // 重定位项数
    WORD NumberOfLinenumbers; // 行号项数
    DWORD Characteristics; // 节区属性 (可读/可写/可执行等)，重要
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;

```

000001E0:	00 00 00 00 00 00 00 00	2E 74 65 78 74 00 00 00text
000001F0:	02 98 0D 00 00 10 00 00	00 9A 0D 00 00 04 00 00
00000200:	00 00 00 00 00 00 00 00	00 00 00 00 20 00 00 60
00000210:	2E 72 64 61 74 61 00 00	DA 89 01 00 00 B0 0D 00	.rdata.....
00000220:	00 8A 01 00 00 9E 0D 00	00 00 00 00 00 00 00 00
00000230:	00 00 00 00 40 00 00 40	2E 64 61 74 61 00 00 00	..@.@.data..
00000240:	E4 3C 00 00 00 40 0F 00	00 24 00 00 00 28 0F 00	<..@...\$. (.
00000250:	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 C0@.
00000260:	2E 69 64 61 74 61 00 00	8B 0C 00 00 00 80 0F 00	.idata.....
00000270:	00 0E 00 00 00 4C 0F 00	00 00 00 00 00 00 00 00	...L.....
00000280:	00 00 00 00 40 00 00 40	2E 30 30 63 66 67 00 00	..@.@.00cfg..
00000290:	0E 01 00 00 00 90 0F 00	00 02 00 00 00 5A 0F 00Z.
000002A0:	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 40@.@.
000002B0:	2E 66 70 74 61 62 6C 65	99 01 00 00 00 A0 0F 00	.fptable.....
000002C0:	00 02 00 00 00 5C 0F 00	00 00 00 00 00 00 00 00\.
000002D0:	00 00 00 00 40 00 00 C0	2E 72 65 6C 6F 63 00 00	..@...reloc..
000002E0:	B8 56 00 00 00 B0 0F 00	00 58 00 00 00 5E 0F 00	V.....X..^.
000002F0:	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 42@.B

计算文件偏移

RAW（文件中的物理偏移） = RVA（内存中的相对地址） - VirtualAddress + PointorToRawData

在这个文件截图中以 .text 节区头为例，RVA = 0x1000，VirtualAddress = 0x1000，PointorToRawData = 0xD9A00，算得 RAW = 0xD9A00。这里 VirtualAddress 与 RVA 相同，因为在节区头中 VirtualAddress 字段直接存储的是 RVA 值。但实际上他们的关系是所有 VirtualAddress 都是 RVA，但并非所有 RVA 都是 VirtualAddress。在通用公式中，VirtualAddress 即取值的是节区头中那个 VirtualAddress，但 RVA 取值需要取决于各区域的 VirtualAddress（例如导入表、导出表），只是在节区头中，RVA 需要取自己区域里的 VirtualAddress 值，所以在这里计算时他们的值相等。

程序运行时 CPU 和操作系统访问的都是 VA（虚拟地址）。RAW、RVA、VA 三者转换关系为 RAW（磁盘中的物理偏移，与内存无关）→ RVA（加载到内存，是 PE 文件内部的相对偏移，用于静态分析）→ VA（运行时访问）

磁盘文件：文件地址 = 文件起始位置（0x00）+ 偏移量（RAW）

内存映射：VA = ImageBase + RVA，这里 ImageBase 是 NT 头 IMAGE_OPTIONAL_HEADER 里的。

实际运行：VA = 实际加载基址（随机化）+ RVA

