DOS头部

```
#define IMAGE_DOS_SIGNATURE                 0x5A4D      // MZ

typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;     // // Magic number（需被设置值0X5A4D ,ASCII值为"MZ"）
    WORD   e_cblp;      // Bytes on last page of file
    WORD   e_cp;        // Pages in file
    WORD   e_crlc;      // Relocations
    WORD   e_cparhdr;   // Size of header in paragraphs
    WORD   e_minalloc;  // Minimum extra paragraphs needed
    WORD   e_maxalloc;  // Maximum extra paragraphs needed
    WORD   e_ss;        // Initial (relative) SS value
    WORD   e_sp;        // Initial SP value
    WORD   e_csum;      // Checksum
    WORD   e_ip;        // Initial IP value
    WORD   e_cs;        // Initial (relative) CS value
    WORD   e_lfarlc;    // File address of relocation table
    WORD   e_ovno;      // Overlay number
    WORD   e_res[4];    // Reserved words
    WORD   e_oemid;     // OEM identifier (for e_oeminfo)
    WORD   e_oeminfo;   // OEM information; e_oemid specific
    WORD   e_res2[10];  // Reserved words
    LONG   e_lfanew;    // File address of new exe header（指出PE头的文件偏移位置）
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

PE头

```
#define IMAGE_NT_SIGNATURE                  0x00004550  // PE00

typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;                        //此字段被设置为0X00004550 ，ASCII码是"PE00"
    IMAGE_FILE_HEADER FileHeader;           //一个IMAGE_FILE_HEADER结构
    IMAGE_OPTIONAL_HEADER32 OptionalHeader; //一个IMAGE_OPTIONAL_HEADER结构
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;
    WORD    NumberOfSections;   //区块的数量
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader;   //跟在此结构后面的数据的大小，即 IMAGE_OPTIONAL_HEADER的大小
    WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

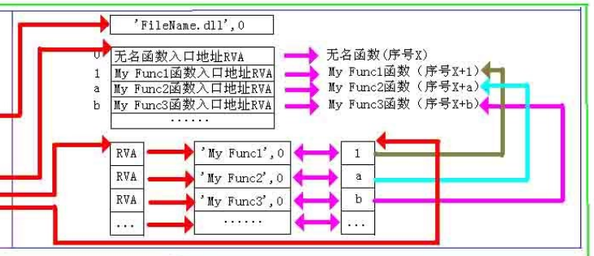
```
typedef struct _IMAGE_OPTIONAL_HEADER {

    // Standard fields.
    WORD    Magic;
    BYTE    MajorLinkerVersion;
    BYTE    MinorLinkerVersion;
    DWORD   SizeOfCode;
    DWORD   SizeOfInitializedData;
    DWORD   SizeOfUninitializedData;
    DWORD   AddressOfEntryPoint;    //程序执行入口RVA
    DWORD   BaseOfCode;     //代码区块的起始RVA
    DWORD   BaseOfData;     //数据区块的起始RVA

    // NT additional fields.
    DWORD   ImageBase;      //文件在内存中的首选装入地址
    DWORD   SectionAlignment;   //被装入内存中时的区块对齐大小，一般是0X1000（4KB）
    DWORD   FileAlignment;      //PE文件内的区块对齐大小，一般是0X200或0X1000
    WORD    MajorOperatingSystemVersion;
    WORD    MinorOperatingSystemVersion;
    WORD    MajorImageVersion;
    WORD    MinorImageVersion;
    WORD    MajorSubsystemVersion;
    WORD    MinorSubsystemVersion;
    DWORD   Win32VersionValue;
    DWORD   SizeOfImage;    //映像装入内存后的总大小
    DWORD   SizeOfHeaders;  //MS-DOS头部+PE头部+区块表的总大小
    DWORD   CheckSum;       //映像的校验和
    WORD    Subsystem;
    WORD    DllCharacteristics;
    DWORD   SizeOfStackReserve;
    DWORD   SizeOfStackCommit;
    DWORD   SizeOfHeapReserve;
    DWORD   SizeOfHeapCommit;
    DWORD   LoaderFlags;
    DWORD   NumberOfRvaAndSizes;    //数据目录表的个数
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];   //数据目录表（重点↑↑）
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT          0   // 导出表
#define IMAGE_DIRECTORY_ENTRY_IMPORT          1   // 导入表
#define IMAGE_DIRECTORY_ENTRY_RESOURCE        2   // 资源
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3   //异常
#define IMAGE_DIRECTORY_ENTRY_SECURITY        4   // 安全
#define IMAGE_DIRECTORY_ENTRY_BASERELOC       5   // 重定位表
#define IMAGE_DIRECTORY_ENTRY_DEBUG           6   // 调试信息
//  IMAGE_DIRECTORY_ENTRY_COPYRIGHT   7 // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7   // 版权信息
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR       8   // RVA of GP
#define IMAGE_DIRECTORY_ENTRY_TLS             9   // TLS Directory
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG    10   // Load Configuration Directory
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT   11   // Bound Import Directory in headers
#define IMAGE_DIRECTORY_ENTRY_IAT            12   // 导入函数地址表
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT   13   // Delay Load Import Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14   // COM Runtime descriptor
```

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress; //数据的RVA
    DWORD   Size;       //   数据的大小
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

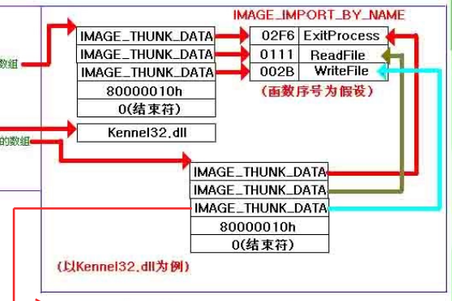
输出表

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;  //输出表创建的时间
    WORD    MajorVersion;   //输出表的主版本号。未使用，设置为0
    WORD    MinorVersion;   //输出表的次版本号。未使用，设置为0
    DWORD   Name;       //指向一个与导输出函数关联的文件名的RVA
    DWORD   Base;       //导出函数的起始序号（假设值为X）
    DWORD   NumberOfFunctions;  //导出函数的总数
    DWORD   NumberOfNames;  //以名称导出的函数总数
    DWORD   AddressOfFunctions;     // 指向导出函数地址表的RVA
    DWORD   AddressOfNames;         // 指向函数名地址表的RVA
    DWORD   AddressOfNameOrdinals;  // 指向函数名序号表的RVA
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```



输入表

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;        // 0 for terminating null import descriptor
        DWORD   OriginalFirstThunk;     // 包含指向IMAGE_THUNK_DATA（输入名称表）结构的数组
    };
    DWORD   TimeDateStamp;      //当可执行文件不与被输入的DLL进行绑定时，此字段为0
    DWORD   ForwarderChain;     //第一个被转向的API的索引
    DWORD   Name;   //指向被输入的DLL的ACII字符串的RVA
    DWORD   FirstThunk;     // 指向输入地址表（IAT）的RVA IAT是一个IMAGE_THUNK_DATA结构的数组
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```



TLS

```
typedef struct _IMAGE_TLS_DIRECTORY32 {
    DWORD   StartAddressOfRawData;//一定范围内存的起始地址
    DWORD   EndAddressOfRawData;//一定范围内存的终止地址
    PDWORD  AddressOfIndex;//索引地址
    PIMAGE_TLS_CALLBACK *AddressOfCallBacks;//函数指针数组的地址
    DWORD   SizeOfZeroFill;//初始化数据的大小
    DWORD   Characteristics;
} IMAGE_TLS_DIRECTORY32;
typedef IMAGE_TLS_DIRECTORY32 * PIMAGE_TLS_DIRECTORY32;
```

资源

```
typedef struct _IMAGE_RESOURCE_DIRECTORY {
    DWORD   Characteristics;    //资源的属性
    DWORD   TimeDateStamp;  //资源的产生时间
    WORD    MajorVersion;   //资源的主版本
    WORD    MinorVersion;   //资源的次版本
    WORD    NumberOfNamedEntries;   //以名称命名的入口数量
    WORD    NumberOfIdEntries;  //以ID命名的入口数量
// IMAGE_RESOURCE_DIRECTORY_ENTRY DirectoryEntries[];
} IMAGE_RESOURCE_DIRECTORY, *PIMAGE_RESOURCE_DIRECTORY;
```

重定位表

```
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress; //重定位内存页的起始RVA
    DWORD   SizeOfBlock;    //重定位块的大小
// WORD  TypeOffset[1];  //重定向位数组
} IMAGE_BASE_RELOCATION;
typedef IMAGE_BASE_RELOCATION UNALIGNED * PIMAGE_BASE_RELOCATION;
```

TypeOffset 是一个数组，数组每项大小为两个字节（16位），它由高 4位和低 12位组成，高4位代表重定位类型，低 12位是重定位地址，它与 VirtualAddress 相加即是指向PE 映像中需要修改的那个代码的地址。

调试信息

```
typedef struct _IMAGE_DEBUG_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;  //DEBUG信息的时间
    WORD    MajorVersion;   //DEBUG的主版本
    WORD    MinorVersion;   //DEBUG的次版本
    DWORD   Type;       //DEBUG信息的类型
    DWORD   SizeOfData; //DEBUG数据的大小
    DWORD   AddressOfRawData;   //当被映射到内存时DEBUG数据的大小
    DWORD   PointerToRawData;   //DEBUG数据的文件偏移
} IMAGE_DEBUG_DIRECTORY, *PIMAGE_DEBUG_DIRECTORY;
```

```
IMAGE_THUNK_DATA STRUC
    union u1
        ForwarderString   DWORD ?   ;指向一个转向者字符串的RVA
        Function          DWORD ?   ;被输入的函数的内存地址
        Ordinal           DWORD ?   ;被输入的API 的序数值
        AddressOfData     DWORD ?   ;指向
        IMAGE_IMPORT_BY_NAME
    ends
```

区块

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];  //块名
    union {
            DWORD   PhysicalAddress;//不用关心，始终是NULL
            DWORD   VirtualSize;    //指出实际的、被使用的区块的大小
    } Misc;                         //（也就是区块的数据没有对齐处理前的实际大小）
    DWORD   VirtualAddress; //该块装载到内存中的RVA
    DWORD   SizeOfRawData;  //该块在磁盘文件中所占的大小
    DWORD   PointerToRawData;   //该块在磁盘文件中的偏移
    DWORD   PointerToRelocations;   //在EXE文件中无意义
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;    //由PointerToRelocations指向的重定位的数目
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;    //块属性
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```