# 随手笔记-PE概述-Object Pascal描述（连载）

作者：Delort

参考：《Windows PE权威指南》，Delphi的Winapi.Windows

## 1.DOS_HEADER

```
IMAGE_DOS_HEADER=record
        e_magic:Word; //exe标志，“MZ”
        e_cblp:Word;  //最后(部分)页中的字节数
        e_cp:Word;    //文件中的全部和部分页数
        e_crlc:Word;  //重定位表的指针数
        e_cparhdr:Word;   //头部尺寸，以段落为单位
        e_minalloc:Word;   //所需的最小附加段
        e_maxalloc:Word;   //所需的最大附加段
        e_ss:Word;         //初始的ss值
        e_sp:Word;         //初始的sp值
        e_csum:Word;       //补码校验值
        e_ip:Word;         //初始的ip值
        e_cs:Word;         //初始的cs值
        e_lfarlc:Word;     //重定位表的字节偏移量
        e_ovno:Word;       //覆盖号
        e_res:array[0..3] of Word; //保留字
        e_oemid:Word;        //OEM标识符
        e_oeminfo:Word;      //OEM信息
        e_res2:array[0..9] of Word;   //保留字
        e_lfanew:LongWord;   //PE文件相对于文件的偏移地址
    end;
```

## 2.NT_HEADER

```
IMAGE_NT_HEADERS=record
        Signature:Cardinal;  //“PE\0\0”
        FileHeader:IMAGE_FILE_HEADER;
        OptionalHeader:IMAGE_OPTIONAL_HEADER;
   end;
```

## 3.FILE_HEADER

```
IMAGE_FILE_HEADER=record
         Machine:Word;   //运行平台
         NumberSizeOfSections:Word; //PE中节的数量
         TimeDateStamp:Cardinal;   //文件创建日期和时间
         PointerToSymbolTable:Cardinal;    //指向符号表（用于调试）
         NumberOfSymbols:Cardinal;        //符号表中的符号数量（用于调试）
         SizeOfOptionalHeader:Word;       //扩展头结构的长度
         Characteristics:Word;            //文件属性
   end;
```

# 4.OPTIONAL_HEADER

```
IMAGE_OPTIONAL_HEADER=record
          Magic:Word;  //魔术字 107h是rom image，10bh是exe image
          MajorLinkerVersion:Byte;  //链接器版本号
          MinorLinkerVersion:Byte;
          SizeOfCode:Cardinal;       //所有含代码节的总大小
          SizeOfInitializedData:Cardinal;     //所有含已初始化数据的节的总大小
          SizeofUninitializedData:Cardinal;   //所有含未初始化数据的节的大小
          AddressOfEntryPoint:Cardinal;        //程序执行入口 RVA
          BaseOfCode:Cardinal;                 //代码的节的起始RVA
          BaseOfData:Cardinal;                 //数据的节的起始 RVA
          ImageBase:Cardinal;                  //程序的建议装载地址
          SectionAlignment:Cardinal;           //内存中节的对齐粒度
          FileAlignment:Cardinal;              //文件中的节的对齐粒度
          MajorOperatingSystemVersion:Word;    //操作系统版本号
          MinorOperatingSystemVersion:Word;
          MajorImageVersion:Word;              //该PE的版本号
          MinorImageVersion:Word;
          MajorSubsystemVersion:Word;          //所需子系统的版本号
          MinorSubsystemVersion:Word;
          Win32VersionValue:Cardinal;          //未用
          SizeOfImage:Cardinal;                //内存中整个PE的映像尺寸
          SzieOfHeaders:Cardinal;              //所有头+节表的总大小
          CheckSum:Cardinal;                   //校验和
          Subsystem:Word;                      //文件的子系统
          DllCharacteristics:Word;             //DLL文件特性
          SizeOfStackReverse:Cardinal;         //初始化的栈大小
          SizeOfStackCommit:Cardinal;          //初始化时实际提交的栈大小
          SizeOfHeapReverse:Cardinal;          //初始化时保留的堆大小
          SizeOfHeapCommit:Cardinal;           //初始化时提交的堆大小
          LoaderFlags:Cardinal;                //与调试有关
          NumberOfRvaAndSizes:Cardinal;        //下面的数据目录结构的项目数量
          DataDirectory:array[0..15] of IMAGE_DATA_DIRECTORY ; //数据目录
    end;
```

# 5.DATA_DIRECTORY

```
IMAGE_DATA_DIRECTORY =record
        virtualAddress:Cardinal;
        Size:Cardinal;
   end;
```

## 6.数据目录入口

| 序号 | 名称 | 解释 | 是否讲解 |
| --- | --- | --- | --- |
| 0 | IMAGE_DIRECTORY_ENTRY_EXPORT | 导出表 | 是 |
| 1 | IMAGE_DIRECTORY_ENTRY_IMPORT | 导入表 | 是 |
| 2 | IMAGE_DIRECTORY_ENTRY_RESOURCE | 资源目录 | 是 |
| 3 | IMAGE_DIRECTORY_ENTRY_EXCEPTION | 异常处理表 | |
| 4 | IMAGE_DIRECTORY_ENTRY_SECURITY | 安全表 | |
| 5 | IMAGE_DIRECTORY_ENTRY_BASERELOC | 重定位表 | 是 |
| 6 | IMAGE_DIRECTORY_ENTRY_DEBUG | 调试表 | |
| 7 | IMAGE_DIRECTORY_ENTRY_ARCHITECTURE | 版权表 | |
| 8 | IMAGE_DIRECTORY_ENTRY_GLOBALPTR | 全局指针表 | |
| 9 | IMAGE_DIRECTORY_ENTRY_TLS | 线程本地存储表 | 是 |
| 10 | IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG | 加载配置表 | 是 |
| 11 | IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT | 绑定导入表 | 是 |
| 12 | IMAGE_DIRECTORY_ENTRY_IAT | 导入函数地址表 | 是 |
| 13 | IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT | 延迟导入表 | 是 |
| 14 | IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR | COM运行时描述符 | |
| 7 | IMAGE_DIRECTORY_ENTRY_COPYRIGHT | 描述字符串 | |

## 7.节表项

NT头后紧跟着节表，由许多节表项组成，记录了每个PE中与特定节有关的信息。

其数据结构如下：

```
TISHMisc = record
    case Integer of
      0: (PhysicalAddress: DWORD);
      1: (VirtualSize: DWORD);//节区尺寸
  end;
_IMAGE_SECTION_HEADER = record
    Name: packed array[0..IMAGE_SIZEOF_SHORT_NAME-1] of Byte;//8字节节名
    Misc: TISHMisc;//如上
    VirtualAddress: DWORD;//节区RVA地址
    SizeOfRawData: DWORD;//在文件中对齐后的尺寸
    PointerToRawData: DWORD;//在文件中的偏移
    PointerToRelocations: DWORD;//OBJ用
    PointerToLinenumbers: DWORD;//行号表位置-调试使用
    NumberOfRelocations: Word;//OBJ用
    NumberOfLinenumbers: Word;//行号表中行号数量
    Characteristics: DWORD;//节的属性
```

```
  end;

```

# 8.导入表

## 8.1导入表描述符

```
IMAGE_IMPORT_DESCRIPTOR = record
    case Byte of
      0: (Characteristics: DWORD);// 0 for terminating null import descriptor
      1: (OriginalFirstThunk: DWORD;//桥1
          TimeDateStamp: DWORD;//时间戳
          ForwarderChain: DWORD;//链表的前一个结构
          Name: DWORD;//指向链接库名字的指针
          FirstThunk: DWORD);//桥2
  end;
```

### 8.1.1OriginalFirstThunk

其中OriginalFirstThunk指向的数组的每个结构为IMAGE_THUNK_DATA，最后一个该结构以全零结束。这个结构定义了一个导入函数信息。

```
IMAGE_THUNK_DATA32 = record
    case Byte of
      0: (ForwarderString: DWORD); // PBYTE
      1: (_Function: DWORD);       // PDWORD Function -> _Function
      2: (Ordinal: DWORD);
      3: (AddressOfData: DWORD);   // PIMAGE_IMPORT_BY_NAME
  end;
```

这个结构是个双字。

最高位为0表示，表示导入符号是一个数值，该数值是一个RVA。

最高位为1表示，表示导入符号是一个名称。

### 8.1.2TimeDateStamp

一般不用，为0.如果该导入表项被绑定，那么绑定后的时间戳就被设置为对应的DLL文件时间戳。操作系统加载时，可以依据此项判断绑定信息是否过时。

### 8.1.3ForwarderChain

链表的前一个结构.

### 8.1.4Name

是个RVA，指向DLL文件的名称，以'\0'结尾。

### 8.1.5FirstThunk

与OriginalFirstThunk相同，它指向的链表为Name这个动态链接库引入的所有导入函数。

桥1指向INT,桥2指向IAT，俩者内容一样，位置不同，最后指向"编号-名称"Hint/Name描述部分。

在内存中，桥1可以帮你找到调用的函数名称或者函数的索引编号，桥2可以帮你找到该函数指令代码在内存空间的地址。

当PE被加载进虚拟地址空间后，IAT的内容会被操作系统改为函数VA，会导致通向H/N的桥2断裂，发生断裂后，如果没有桥1作为参照，我们就无法重新找到该地址到底调用了哪个函数。

Borland的TLink只保留桥2，为单桥结构。

调用外部函数，IMAGE_THUNK_DATA会是个RVA,指向IMAGE_IMPORT_BY_NAME结构。

其结构如下：

```
_IMAGE_IMPORT_BY_NAME = record
    Hint: Word;//函数编号
    Name: array[0..0] of Byte;//函数名字的字符串
  end;
  IMAGE_IMPORT_BY_NAME = _IMAGE_IMPORT_BY_NAME;
```

## 8.2导入函数地址表-IAT

该地址表是数据目录的第13个数据项目录。是个双字数组。

其结构如下：

| **链接库1引入的函数1地址** |
| --- |
| 链接库1引入的函数2地址 |
| 0x00000000 |
| 链接库2引入的函数1地址 |
| 链接库2引入的函数2地址 |
| 链接库2引入的函数3地址 |
| 0x00000000 |
| …… |

## 8.3导入表遍历实验

实验步骤：

1.获取导入表第一个描述符的起始地址

2.获取导入表所处节的位置，同时显示

3.循环，直到到达末尾

4.显示链接库名称和该链接库所有函数和编号。

实验代码：

```

```

实验结果样图：

## 8.4绑定导入

目的：提前进行IAT修正的工作。

### 8.4.1绑定导入的数据结构

```
_IMAGE_BOUND_IMPORT_DESCRIPTOR = record
    TimeDateStamp: DWORD;//时间戳
    OffsetModuleName: Word;//指向DLL的名字
    NumberOfModuleForwarderRefs: Word;//ModuleForwarderRef的数目
    // Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
  end;
```

### 8.4.2TimeDateStamp

该字段必须与引用的DLL文件头IMAGE_FILE_HEADER.TimeDateStamp字段值吻合。否则加载器重新计算IAT。发生在DLL版本不同或者DLL映像被重定位时。

### 8.4.3OffsetModuleName

存储了以IMAGE_BOUND_IMPORT_DESCRIPTOR作为基址，DLL名字的字符串的偏移。

### 8.4.4NumberOfModuleForwarderRefs

描述了在IMAGE_BOUND_IMPORT_DESCRIPTOR结构后的IMAGE_BOUND_FORWARDER_REF的数组元素个数。

其结构为：

```
_IMAGE_BOUND_FORWARDER_REF = record
    TimeDateStamp: DWORD;//时间戳
    OffsetModuleName: Word;//指向DLL名字
    Reserved: Word;//预留
  end;
```

### 8.4.5总体结构

| **IMAGE_BOUND_IMPORT_DESCRIPTOR（2个REF)** |
| --- |
| IMAGE_BOUND_FORWARDER_REF |
| IMAGE_BOUND_FORWARDER_REF |
| IMAGE_BOUND_IMPORT_DESCRIPTOR（1个REF) |
| IMAGE_BOUND_FORWARDER_REF |
| IMAGE_BOUND_IMPORT_DESCRIPTOR（3个REF) |
| IMAGE_BOUND_FORWARDER_REF |
| IMAGE_BOUND_FORWARDER_REF |
| IMAGE_BOUND_FORWARDER_REF |
| .... |

# 9.导出表

作用：

1.可以通过导出表分析不认识的动态链接库文件所提供的功能。

2.向调用者提供输出函数指令在模块中的起始地址。

## 9.1导出表数据结构

```
_IMAGE_EXPORT_DIRECTORY = record
     Characteristics: DWord;//标志，未用
     TimeDateStamp: DWord;//时间戳
     MajorVersion: Word;//未用
     MinorVersion: Word;//未用
     Name: DWord;//指向该导出表的文件名字符串
     Base: DWord;//导出函数的起始序号
     NumberOfFunctions: DWord;//所有导出函数的个数
     NumberOfNames: DWord;//以函数名导出的函数的个数
     AddressOfFunctions: DWORD;//导出函数地址表RVA
     AddressOfNames: DWORD;//函数名称地址表RVA
     AddressOfNameOrdinals: DWORD;//函数序号地址表
 end;
```

### 9.1.1Name

一个字符串地址，记录了导出表所在文件的最初文件名。

### 9.1.2NumberOfFunctions

记录了文件中导出函数的总个数。

### 9.1.3NumberOfNames

记录了所有定义了名字的导出函数，有的没有名字。

### 9.1.4AddressOfFunctions

该指针指向了全部导出函数的入口地址的起始。从入口地址开始为双字数组，数组的个数由NumberOfFunctions决定。

导出函数每一个地址按函数编号依次往后排开。

### 9.1.5Base

导出函数编号的起始值。函数地址=Base+AddressOfFunctions。

### 9.1.6AddressOfNames

指向一连串的双字值，这些双字值均指向了对应的定义了函数名的函数的字符串地址。个数为NumberOfNames。

### 9.1.7AddressOfNameOrdinals

与AddressOfNames一一对应，不同的是AddressOfNames指向字符串指针数组，而AddressOfNameOrdinals则指向了该函数在AddressOfFunctions中索引值。索引值=编号-Base。是一个字大小。

# 10.重定位表

重定位表指针指向的是重定位项数组

## 10.1重定位表数据结构

```
IMAGE_BASE_RELOCATION=record
     VirtualAddress:DWord;
     SizeofBlock:DWord;
end;
```

### 10.1.1VirtualAddress

重定位块RVA，是页面起始RVA

因为地址用=2*n+4+4个字节来表达。此处n为n个重定位项，第一个4为页面起始RVA，第二个4为本页重定位项个数。

为何有2呢？因为一页1000h大小，4096字节，及2**12,12位可以用2个字节表达

### 10.1.2SizeofBlock

重定位表项数目，第二个4.

### 10.1.3重定位项

每个重定位项大小为2字节，高四位说明类型，其余12位为地址。

常用的俩种类型：

0：无意义，对齐作用。

3：双32位都需要修正。

### 10.1.4重定位表结构