PE文件学习笔记（三）：导出表（Export Table）解析

数据目录（Data Directory）有16个_IMAGE_DATA_DIRECTORY结构体元素，该结构体数组是可选PE头中最后一个成员。这十六个元素分别存储了不同信息，分别是：**导入表**、**导出表**、资源、异常信息、安全证书、**重定位表**、调试信息、版权所有、全局指针、TLS、加载配置、**绑定导入**、**IAT**、延迟导入、COM信息、最后一个保留未使用。和程序运行时息息相关的表有：导出表、导入表、重定位表、IAT表等，这几种也是PE解析中重点研究的几张表。

```
typedef struct _IMAGE_DATA_DIRECTORY{
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

该结构体标记了各个表（元素）在内存中的VirtualAddress与Size。VirtualAddress是内存中的偏移地址，我们要直接在文件中根据VirtualAddress找到对应的表，就需要进行判断。判断VirtualAddress在哪个节，并且计算在节中的偏移量，即RVA->FOA的转换。

1、导出表基本结构：

根据_IMAGE_DATA_DIRECTORY结构体数组的**第1个元素索引处导出表**。一般情况下，dll的函数导出供其他人使用，exe将别人的dll的函数导入运行。
所以，一般.exe没有导出表（但是并非说.exe一定没有导出表）。

导出表结构：

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;    //未使用
    DWORD   TimeDateStamp;      //时间戳
    WORD    MajorVersion;       //未使用
    WORD    MinorVersion;       //未使用
    DWORD   Name;               //指向该导出表文件名字符串
    DWORD   Base;               //导出表的起始序号
    DWORD   NumberOfFunctions;  //导出函数的个数(更准确来说是AddressOfFunctions的元素数，而不是函数个数)
    DWORD   NumberOfNames;      //以函数名字导出的函数个数
    DWORD   AddressOfFunctions;     //导出函数地址表RVA:存储所有导出函数地址(表元素宽度为4，总大小NumberOfFunctions * 4)
    DWORD   AddressOfNames;         //导出函数名称表RVA:存储函数名字符串所在的地址(表元素宽度为4，总大小为NumberOfNames * 4)

```

```
    DWORD   AddressOfNameOrdinals;  //导出函数序号表RVA:存储函数序号(表元素宽度为2，总大小为NumberOfNames * 2)
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

注意：

里面的地址均是RVA，而如果我们不想转换成ImageBuffer就一定要进行RVA->FOA转换，根据FOA直接在FileBuffer中寻找导出表。每个dll都有一个导出表。而每个导出表有三个子导出表(地址AddressOfFunctions、名字AddressOfNames、序号AddressOfOrdinals)。NumberOfFunctions是函数序号最大值与最小值之间的差值，NumberOfNames是函数以名字导出的个数，二者可以不一样大。一个函数必定有地址，但不一定有名字(如果是以无名字的方式导出，eg：func @12 NONAME)。

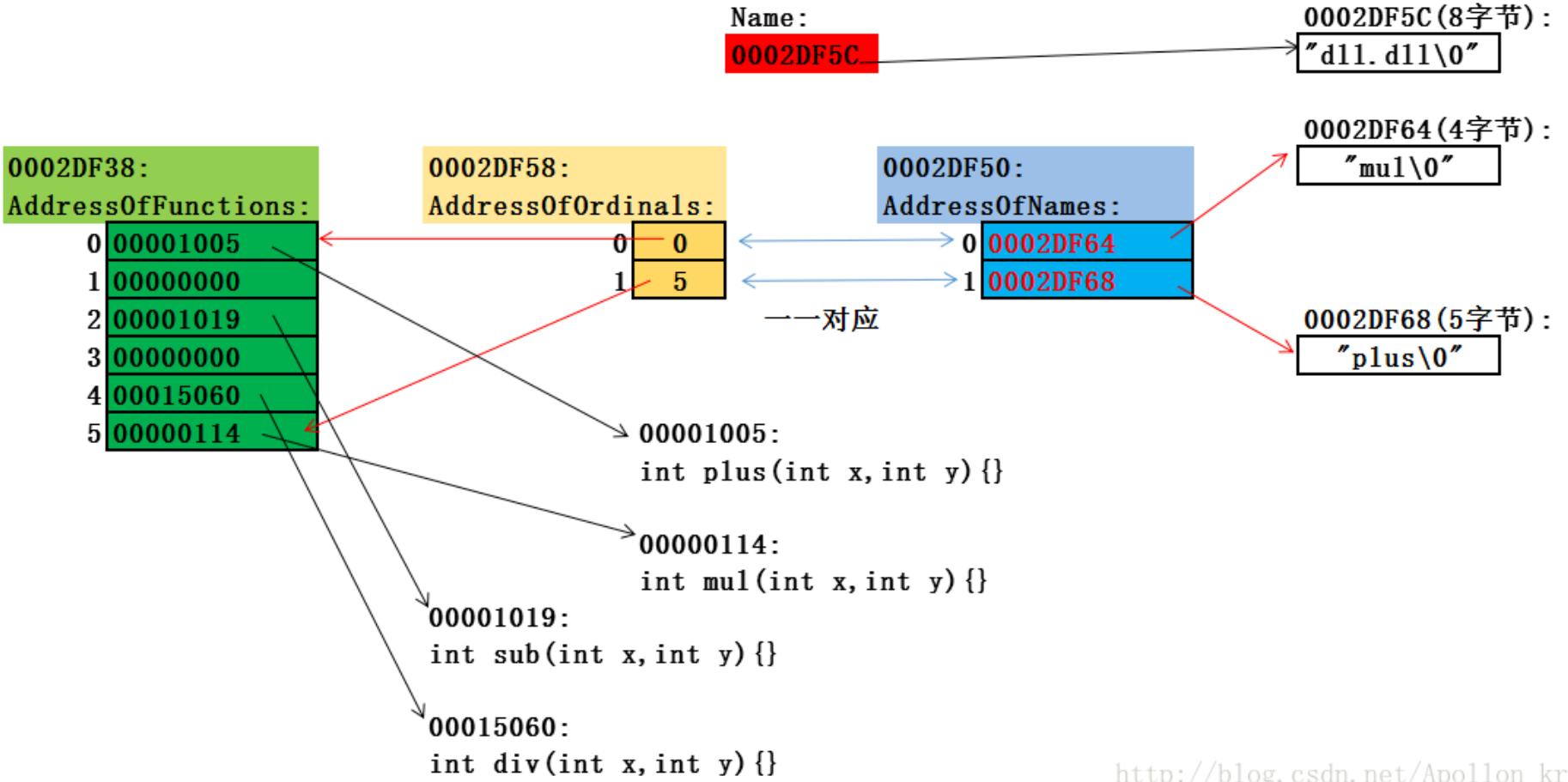
我们自定义一个dll库，在导出时采用.def文件的方式导出：

```
//dll.def
EXPORTS
Plus    @1
Sub     @3 NONAME
div     @5 NONAME
mul     @6
```

则即解析得到的信息如下（“———”代表没有名字，Ordina显示的值与内存里的值差一个Base，因为在用代码进行解析时已经把Base算过了，而内存中没有动）：

```
Offset to Export Table:[0002DF10]
    Characteristics:       [00000000]
    TimeDateStamp:         [59945264]
    MajorVersion:          [0000]
    MinorVersion:          [0000]
    NameAddr:              [0002DF5C]
    NameString:            [dll.dll]
    Base:                  [00000001]
    NumberOfFunctions:     [00000006]
    NumberOfNames:         [00000002]
    AddressOfFunctions:    [0002DF38]
    AddressOfNames:        [0002DF50]
    AddressOfNameOrdinals: [0002DF58]
    Ordina  func_FOA    name_FOA    FunctionName
    0001    00001005    0002DF68    plus
    0003    00001019    --------    --------
    0005    00015060    --------    --------
    0006    00001014    0002DF64    mul
```

导出的函数有两个函数我们声明为noname，故其在导出表中不存在名字。则其导出表的NumberOfNames = 2，NumberOfFunctions = （6-1+1） = 6。即地址表长度为6宽度为4，Size为24；名字表长度为2，宽度为4，Size为8；序号表长度为2，宽度为2，Size为4。对应的内存图如下（名字表的地址是按照从小到大排的，地址表有与我们.def中指定了序号，因此是乱序的，如果不指定，编译器自动分配的一般也是有序的）：



虽然导出时div、sub没有序号与名字，但是二者是有地址的。并且由于序号计算地址表时，地址表中有些值没有映射，则填充为0。而文件中如下所示（导出表在文件中开始

地址0002DF10）：

```
0002df10h: 00 00 00 00 64 52 94 59 00 00 00 00 5C DF 02 00 ;....dR攩....\?.
0002df20h: 01 00 00 00 06 00 00 00 02 00 00 00 38 DF 02 00 ;............8?.
0002df30h: 50 DF 02 00 58 DF 02 00 05 10 00 00 00 00 00 00 ;P?.X?.........
0002df40h: 19 10 00 00 00 00 00 00 60 50 01 00 14 10 00 00 ;........`P......
0002df50h: 64 DF 02 00 68 DF 02 00 05 00 00 00 64 6C 6C 2E ;d?.h?.....dll.
0002df60h: 64 6C 6C 00 6D 75 6C 00 70 6C 75 73 00 00 00 00 ;dll.mul.plus....
0002df70h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ;................
```


①导出表中AddressOfFunction指向的地址表大小根据 NumberOfFunctions 决定:**地址表大小 = NumberOfFunctions * 4**;
②而AddressOfNames指向的名字表大小不由 NumberOfFunctions 决定，而由NumberOfNames决定:**名字表大小 = NumberOfNames * 4**;
③AddressOfNameOrdinals指向的序号表中的值是非准确的，应该均加上Base才是真正的序号(Base等于序号表中最小的值)。**而序号表大小 = NumberOfNames * 2。**
④**地址表可能大于等于名字表，也有可能小于名字表，因为一个函数可能没有名字，也可能有多个名字。**但是一般情况下，名字表均不会大于地址表。并且一个函数必然有地址，不一定有名字，名字表和序号表一一对应。

2、导出表解析：

知道一个函数名字func，如何找到其在PE文件中的地址？步骤（根据三张子表查找）：

①在名字表遍历RVA地址，转换成FOA地址，然后根据FOA比较FOA指向的字符串与func是否相等，不相等则判断下一个。
②如果相等则获取到其在名字表中的索引(下标)，根据该索引获取对应的序号表中同一下标索引到的序号值value。
③value作为地址表的索引，索引到的值即为func()的地址。
也就是我们上面图中所描述的。

RVA->FOA的转换函数如下：

```
//输入RVA（内存相对偏移地址），返回FOA（文件偏移地址）
DWORD PETool::RVAToFOA(DWORD imageAddr)
{
    /*
     * 相对虚拟地址转文件偏移地址
     * ①获取Section数目
     * ②获取SectionAlignment
     * ③判断需要转换的RVA位于哪个Section中(section[n]),
     * offset = 需要转换的RVA-VirtualAddress，计算出RVA相对于本节的偏移地址
```

```
     * @section[n].PointerToRawData + offset就是RVA转换后的FOA
    */

    if(imageAddr > imageSize){
        printf("RVAToFOA in_addr is error!%08X\n",imageAddr);
        exit(EXIT_FAILURE);
    }
    if(imageAddr < section_header[0].PointerToRawData){
        return imageAddr;//在头部（包括节表与对齐）则直接返回
    }
    IMAGE_SECTION_HEADER * section = section_header;
    DWORD offset = 0;
    for(int i = 0; i < sectionNum; i++){
        DWORD lower = section[i].VirtualAddress;//该节下限
        DWORD upper = section[i].VirtualAddress+section[i].Misc.VirtualSize;//该节上限
        if(imageAddr >= lower && imageAddr <= upper){
            offset = imageAddr - lower + section[i].PointerToRawData;//计算出RVA的FOA
            break;
        }
    }
    return offset;
}
```

导出表的解析实现如下（省略文件到内存的读入）：

```
void PETool::print_ExportTable()
{
    fprintf(fp_peMess, "导出表(export table):\n");
    if(dataDir[0].VirtualAddress == 0){
        fprintf(fp_peMess, "\t不存在导出表!\n");
        return;
    }
    DWORD offset = RVAToFOA(dataDir[0].VirtualAddress);
    IMAGE_EXPORT_DIRECTORY * exportTb = (IMAGE_EXPORT_DIRECTORY * )(pFileBuffer + offset);


```

```
    fprintf(fp_peMess, "\tOffset to Export Table:[%08X]\n",dataDir[0].VirtualAddress);
    fprintf(fp_peMess, "\tCharacteristics:       [%08X]\n", exportTb->Characteristics);
    fprintf(fp_peMess, "\tTimeDateStamp:         [%08X]\n", exportTb->TimeDateStamp);
    fprintf(fp_peMess, "\tMajorVersion:          [%04X]\n", exportTb->MajorVersion);
    fprintf(fp_peMess, "\tMinorVersion:          [%04X]\n", exportTb->MinorVersion);
    fprintf(fp_peMess, "\tNameAddr:              [%08X]\n", exportTb->Name);
    fprintf(fp_peMess, "\tNameString:            [%s]\n", pFileBuffer + RVAToFOA(exportTb->Name));
    fprintf(fp_peMess, "\tBase:                  [%08X]\n", exportTb->Base);
    fprintf(fp_peMess, "\tNumberOfFunctions:     [%08X]\n", exportTb->NumberOfFunctions);
    fprintf(fp_peMess, "\tNumberOfNames:         [%08X]\n", exportTb->NumberOfNames);
    fprintf(fp_peMess, "\tAddressOfFunctions:    [%08X]\n", exportTb->AddressOfFunctions);
    fprintf(fp_peMess, "\tAddressOfNames:        [%08X]\n", exportTb->AddressOfNames);
    fprintf(fp_peMess, "\tAddressOfNameOrdinals: [%08X]\n", exportTb->AddressOfNameOrdinals);

    //打印导出表
    fprintf(fp_peMess, "\n\tOrdina\tfunc_FOA\tname_FOA\tFunctionName\n");
    DWORD * addrFunc = (DWORD *)(pFileBuffer + RVAToFOA(exportTb->AddressOfFunctions));
    DWORD * addrName = (DWORD *)(pFileBuffer + RVAToFOA(exportTb->AddressOfNames));
    WORD * addrOrdi = (WORD *)(pFileBuffer + RVAToFOA(exportTb->AddressOfNameOrdinals));

    //Base--->NumberOfFunctions
    DWORD i, j;
    for(i = 0; i < exportTb->NumberOfFunctions; i++){//导出时序号有NumberOfFunctions个
        if(addrFunc[i] == 0){
            continue;//地址值为0代表该序号没有对应的函数，是空余的
        }
        for(j = 0; j < exportTb->NumberOfNames; j++){//序号表序号有NumberOfNames个
            if(addrOrdi[j] == i){//序号表的值为地址表的索引
                fprintf(fp_peMess, "\t%04X\t%08X\t%08X\t%s\n", i + exportTb->Base, addrFunc[i], addrName[j], pFileBuffer + addrName[j]);
                break;
            }
        }
        //存在addrOrdi[j]时,i(索引)等于addrOrdi[j](值),不存在，则i依旧有效,i+Base依旧是序号
        if(j != exportTb->NumberOfNames){
            continue;//在序号表中找到
        }
        else{//如果在序号表中没有找到地址表的索引，说明函数导出是以地址导出的，匿名函数
            fprintf(fp_peMess, "\t%04X\t%08X\t%s\t%s\n", i + exportTb->Base, addrFunc[i], "--------", "--------");
```

```
48
49
50            }
51        }
    }
```