导入表

`IAT`：Import Address Table，导入地址表

编译器和操作系统约定在一个地址上填写外部导入接口的地址，这张表就是IAT（进程装载的时候填写）

动态库与函数属于`1:M`

IMAGE_IMPORT_DESCRIPTOR

```
#define IMAGE_DIRECTORY_ENTRY_IMPORT    1

typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;      √     // 0 for terminating null import
descriptor
        DWORD   OriginalFirstThunk;   √     // RVA to original unbound IAT
(PIMAGE_THUNK_DATA)
    } DUMMYUNIONNAME;
    DWORD   TimeDateStamp;                  // 0 if not bound,
                                            // -1 if bound, and real date\time
stamp
                                            //     in
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT (new BIND)
                                            // O.W. date/time stamp of DLL bound
to (Old BIND)

    DWORD   ForwarderChain;                 // -1 if no forwarders
    DWORD   Name;                   √
    DWORD   FirstThunk;               √     // RVA to IAT (if bound this IAT has
actual addresses)
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```

存储动态库信息，每个被加载的模块对应一个`IMAGE_IMPORT_DESCRIPTOR`结构，形成结构数组，全0结构结尾

- `DUMMYUNIONNAME`
  联合结构，大小**DWORD**，成员：
  - `Characteristics`
    此值为0时，表示表示此为结构数组的结尾
  - `OriginalFirstThunk`
    为相对虚拟地址，指向`IMAGE_THUNK_DATA`结构数组(与`FirstThunk`指向的结构的内存地址不同)。（数组中的每一项在加载前与加载后都**一样**）
- `Name`
  为相对虚拟地址，指向导入模块的名字，所以一个`IMAGE_IMPORT_DESCRIPTOR`描述一个导入的模块
- `FirstThunk`
  为相对虚拟地址，指向`IMAGE_THUNK_DATA`结构数组(与`OriginalFirstThunk`指向的结构的内存

地址不同)。（数组中的每一项在加载前与加载后**不一样**）

IMAGE_THUNK_DATA32

```
#define IMAGE_DIRECTORY_ENTRY_IAT   12

typedef IMAGE_THUNK_DATA32              IMAGE_THUNK_DATA;
typedef PIMAGE_THUNK_DATA32             PIMAGE_THUNK_DATA;

typedef struct _IMAGE_THUNK_DATA32 {
    union {
        DWORD ForwarderString;      // PBYTE
        DWORD Function;             // PDWORD
        DWORD Ordinal;
        DWORD AddressOfData;        // PIMAGE_IMPORT_BY_NAME
    } u1;
} IMAGE_THUNK_DATA32;
typedef IMAGE_THUNK_DATA32 * PIMAGE_THUNK_DATA32;
```

`u1`
联合结构，大小为**DWORD**，成员：

- `ForwarderString`
  转发用的，暂不考虑
- `Function`
  如果是序号导入，则值为函数的相对虚拟地址
- `Ordinal`
  判断是否是序号导入。最高位为1，则是序号导入。为0，则是名字导入（序号为低**16bit**）
- `AddressOfData`
  如果为名字导入，则值为一个相对虚拟地址，指向一个`IMAGE_IMPORT_BY_NAME`结构数组，用来保存名字信息（低**31bit**）

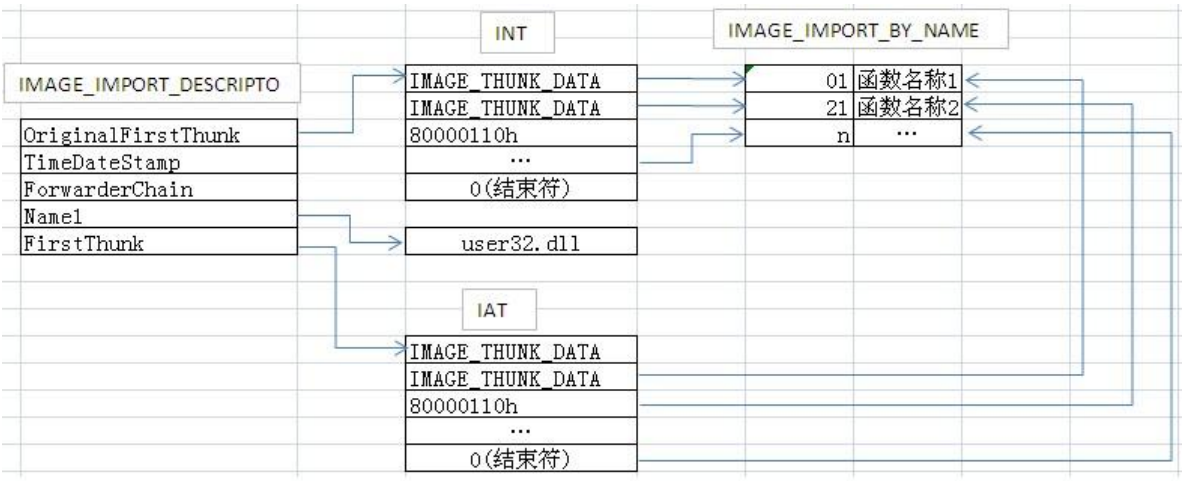
IMAGE_IMPORT_BY_NAME

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    CHAR    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME
```

- `Hint`
  指示本函数在其所驻留DLL的引出表中的索引号。该域被PE装载器用来在DLL的引出表里快速查询函数。该值不是必须的，一些连接器将此值设为0。
- `Name[1]`
  含有引入函数的函数名。函数名是一个ASCIIZ字符串。注意这里虽然将Name1的大小定义成字节，其实它是可变尺寸域。

装载细节

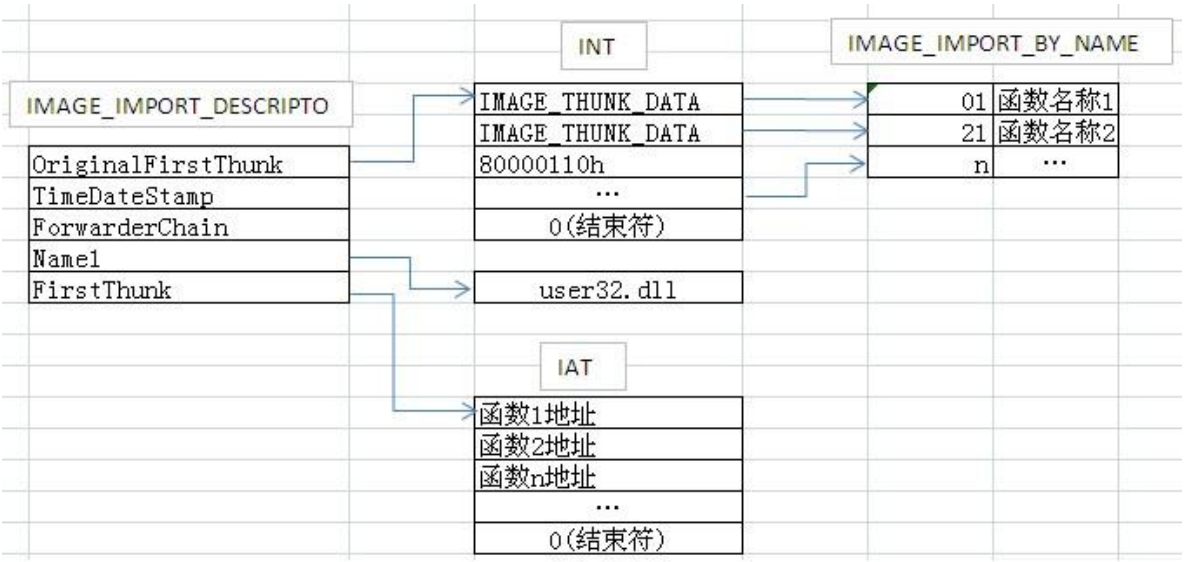
1. 加载前



在文件中**INT**和**IAT**的内容一般是一样的，但所在位置一般不一样

- 当**INT**的指针`OriginalFirstThunk`为空，则找**IAT**中的数据，并在**IAT**中取得函数名（或序号）获取到函数地址再次填入**IAT**

2. 加载后



- 无效导入表项：当**IAT**的指针`FirstThunk`所指向的目标为空（即IAT空），则该目标无效，继续下一次
  - 即如果此时**IAT**的指针`FirstThunk`所指向的目标不是空，则此项导入表`IMAGE_IMPORT_DESCRIPTOR`有效，反之无效

加载时操作系统的行为：

拿到一个导入表项后，

1. 检查`FirstThunk`指向的位置即**IAT**是否为空
    - 为空：此表项无效，继续下一个表项
    - 非空：此表项有效
2. 访问库名称，拿到库名称后`LoadLibrary`，成功则继续
3. 访问名称表`OriginalFirstThunk`，`GetProcessAddress`拿到函数地址填写**IAT**

总结