PE文件结构补充

1.整体结构图

相对虚拟地址RVA与虚拟地址VA

当一个 PE 文件被加载到内存中以后，我们称之为 " 映象 "（image），**一般来说，PE 文件在硬盘上和在内存里是不完全一样的，被加载到内存以后其占用的虚拟地址空间要比在硬盘上占用的空间大一些，这是因为各个节在硬盘上是连续的，而在内存中是按页对齐的，所以加载到内存以后节之间会出现一些 “空洞” 。**

因为存在这种对齐，所以在 PE 结构内部，表示某个位置的地址采用了两种方式：

1. 针对在硬盘上存储文件中的地址，称为 **原始存储地址** 或 **物理地址，表示 距离文件头的偏移。**
2. 针对加载到内存以后映象中的地址，称为 **相对虚拟地址（RVA），表示相对内存映象头的偏移**。

数据目录

DataDirectory[] 数据目录数组：数组中的每一项对应一个特定的数据结构，包括导入表，导出表等等，第一个双字表示RVA，第二个双字表示大小

第一个是导出表

第二个是导入表

倒数第三个是IAT表

```
'#define IMAGE_DIRECTORY_ENTRY_EXPORT          0   // Export Directory '
'#define IMAGE_DIRECTORY_ENTRY_IMPORT          1   // Import Directory '
#define IMAGE_DIRECTORY_ENTRY_RESOURCE        2   // Resource Directory
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3   // Exception Directory
#define IMAGE_DIRECTORY_ENTRY_SECURITY        4   // Security Directory
#define IMAGE_DIRECTORY_ENTRY_BASERELOC       5   // Base Relocation Table
#define IMAGE_DIRECTORY_ENTRY_DEBUG           6   // Debug Directory
//      IMAGE_DIRECTORY_ENTRY_COPYRIGHT       7   // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7   // Architecture Specific Data
#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   // Import Address Table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT   13   // Delay Load Import Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14   // COM Runtime descriptor
```

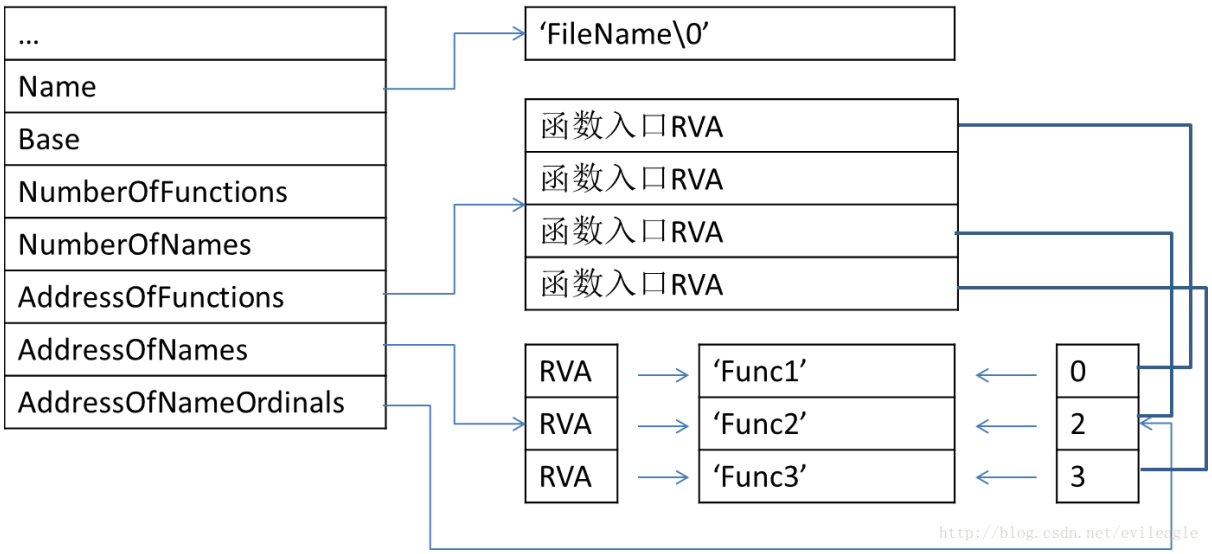
函数导出表

由数据目录的第二项指示

具体每一项的含义如下：

1. Characteristics：现在没有用到，一般为0。
2. TimeDateStamp：导出表生成的时间戳，由连接器生成。
3. MajorVersion，MinorVersion：看名字是版本，实际貌似没有用，都是0。
4. Name：模块的名字。
5. Base：序号的基数，按序号导出函数的序号值从Base开始递增。
6. NumberOfFunctions：所有导出函数的数量。
7. NumberOfNames：按名字导出函数的数量。
8. AddressOfFunctions：一个RVA，指向一个DWORD数组，数组中的每一项是一个导出函数的RVA，顺序与导出序号相同。
9. AddressOfNames：一个RVA，依然指向一个DWORD数组，数组中的每一项仍然是一个RVA，指向一个表示函数名字。

10. AddressOfNameOrdinals：一个RVA，还是指向一个WORD数组，数组中的每一项与AddressOfNames中的每一项对应，表示该名字的函数在AddressOfFunctions中的序号。



函数导入表

- OriginalFirstThunk 保存一个 RVA，指向一个 IMAGE_THUNK_DATA 的数组，这个数组中的每一项表示一个导入函数。

• FirstThunk：也是一个 RVA，也指向一个 IMAGE_THUNK_DATA 数组。

IMAGE_THUNK_DATA结构书中已有说明。

OriginalFirstThunk 指向的 IMAGE_THUNK_DATA 数组包含导入信息，在这个数组中只有 Ordinal 和 AddressOfData 是有用的，因此可以通过 OriginalFirstThunk 查找到函数的地址。也就是**指向INT（导入名称表），**该表内容指向函数名。

FirstThunk则略有不同，在PE文件加载以前或者说在导入表未处理以前，他所指向的数组与 OriginalFirstThunk 中的数组虽不是同一个，但是内容却是相同的，都包含了导入信息，而在加载之后，FirstThunk 中的 Function 开始生效，他指向实际的函数地址，因为FirstThunk 实际上指向 IAT 中的一个位置，IAT 就充当了 IMAGE_THUNK_DATA 数组，加载完成后，这些 IAT 项就变成了实际的函数地址，即 Function 的地址。

这个OriginalFirstThunk 和 FirstThunk明显是亲家，两家伙首先名字就差不多哈。那他们有什么不可告人的秘密呢？来，我们看下面一张图（画的很辛苦，大家仔细看哈）：