重定位表和导入表导出表寻找方法一样，找NT头的数据目录表。

IMAGE\_BASE\_RELOCATION

找到相应节区信息定位到重定位表，重定位表由多个IMAGE\_BASE\_RELOCATION块组成，每个块包含一组需要修改的地址。

typedef struct \_IMAGE\_BASE\_RELOCATION {

DWORD VirtualAddress; *// 这组重定位项的起始RVA*

DWORD SizeOfBlock; *// 整个块的大小(包括这个结构体和后面的重定位项)*

*// 后面跟着变长的WORD类型重定位项数组，以注释形式存在：*

*// WORD TypeOffset[1]; // 高4位是类型，低12位是偏移*

} IMAGE\_BASE\_RELOCATION;

每个重定位块的IMAGE\_BASE\_RELOCATION结构下跟着一个多个16位的WORD（TypeOffset），每个WORD代表一个具体的重定位位置信息。

Bits 15-12（高四位）: 重定位类型

Bits 11-0（低四位）: 相对于VirtualAddress的偏移量

重定位类型有：

类型值 宏定义 描述

0 IMAGE\_REL\_BASED\_ABSOLUTE 空项，仅用于对齐填充，不执行任何修正。

1 IMAGE\_REL\_BASED\_HIGH 修正目标地址的 高16位（用于分段内存架构，现代PE文件极少使用）。

2 IMAGE\_REL\_BASED\_LOW 修正目标地址的 低16位（同样较少使用）。

3 IMAGE\_REL\_BASED\_HIGHLOW 完整修正32位地址（最常见的类型，用于32位PE文件）。

4 IMAGE\_REL\_BASED\_HIGHADJ 修正高16位，并额外调整下一个重定位项的值（罕见）。

5 IMAGE\_REL\_BASED\_MIPS\_JMPADDR MIPS架构专用（已废弃）。

9 IMAGE\_REL\_BASED\_MIPS\_JMPADDR16 MIPS16架构专用（已废弃）。

10 IMAGE\_REL\_BASED\_DIR64 完整修正64位地址（用于64位PE文件）。

由 IMAGE\_BASE\_RELOCATION 结构中的 RVA转换后指向的地址的内容是需要被修正的原始绝对地址值。

重定位过程地址底层转换过程如下：

1. 尝试加载到 ImageBase（预定加载基址）

└─ 成功 → 无需重定位

└─ 失败 → 分配新基址 ActualBase（实际加载基址）

1. 计算 Delta（基址偏移量） = ActualBase - ImageBase
2. 遍历重定位表：

└─ 对每个块：

1. 取 VirtualAddress
2. 对每个重定位项：
3. 计算 TargetRVA = VirtualAddress + Offset
4. 找到内存地址：pTarget = ActualBase + TargetRVA
5. 修正 \*pTarget += Delta（这一步是修正指针，文件的地址在上一步就计算完成了）

4. 完成所有修正后，代码中的绝对地址可正确运行。

关键值：

ImageBase : NT头 → OptionalHeader → ImageBase

VirtualAddress : .reloc 节区 → IMAGE\_BASE\_RELOCATION

TargetRVA : VirtualAddress + (TypeOffset & 0xFFF)

重定位高层原理大致是VA -> RVA -> VA的转换，对应到底层并没有VA -> RVA的转换过程，且RVA由VirtualAddress + Offset隐式表达。RVA -> VA步对应到底层也就是TargetRVA += Delta的过程。