重定位表

调试版本、动态链接库、发布版的`/FIXED:NO`链接选项会产生重定位表

在数据目录中的大小是有意义的

记录需要根据实际装载地址修正的位置的数据结构，就是重定位表

修正公式：`A + (实际地址 - 原基址)`

IMAGE_BASE_RELOCATION

```
#define IMAGE_DIRECTORY_ENTRY_BASERELOC 5

typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress;
    DWORD   SizeOfBlock;
//  WORD    TypeOffset[1];
} IMAGE_BASE_RELOCATION;
typedef IMAGE_BASE_RELOCATION UNALIGNED * PIMAGE_BASE_RELOCATION
```

- `VirtualAddress`
  重定位表的相对虚拟地址
- `SizeOfBlock`
  重定位表的块大小，表示后面`TypeOffset`的大小
- `TypeOffset`
  第二个重定位表与第一个之间的数据，用来标明那些地址需要更改（类型为**WORD**， 低***12bit***用来标明相对`VirtualAddress`中数据的偏移量）

说明

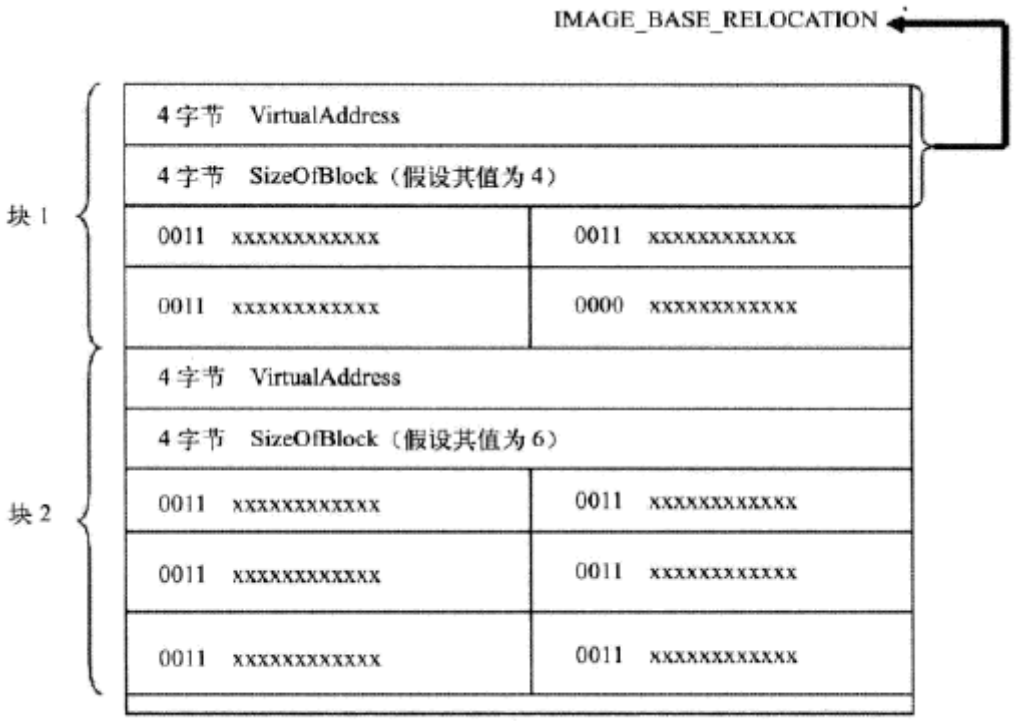

图 6-6　重定位表结构

重定位表有一个`IMAGE_BASE_RELOCATION`结构体和后面跟随的重定位块区域构成，块区域由**WORD**数组构成重定位项。

重定位项的**高4位**表示重定位的类型，如下图

表 6-6　IMAGE_BASE_RELOCATION.SizeOfBlock 高四位含义表

值	常量符号	含　义
0	IMAGE_REL_BASED_ABSULUTE	无意义，仅作对齐用
1	IMAGE_REL_BASED_HIGH	双字中，仅高十六位被修正
2	IMAGE_REL_BASED_LOW	双字中，仅低十六位被修正
3	IMAGE_REL_BASED_HIGHLOW	双字 32 位都需要修正
4	IMAGE_REL_BASED_HIGHADJ	进行基地址重定位时将差值的高十六位加到指定偏移处的一个 16 位域上。这个 16 位域是一个 32 位字的高半部分，而这个 32 位字的低半部分被存储在紧跟在这个 Type/Offset 位域后面的一个 16 位字中。也就是说，这一个基地址重定位项占了两个 Type/Offset 位域的位置
5	IMAGE_REL_BASED_MIPS_JMPADDR	对 MIPS 平台的跳转指令进行基地址重定位
6		保留，必须为 0
7		保留，必须为 0
9	IMAGE_REL_BASED_MIPS_JMPADDR16	对 MIPS16 平台的跳转指令进行基地址重定位
10	IMAGE_REL_BASED_DIR64	进行基地址重定位时将差值加到指定偏移处的一个 64 位域上

而低**12位**表示相对于`VirtualAddress`的偏移量，故需要重定位的地址的`RVA = VirtualAddress + 重定位项的低12位`

例如：

```
bool fix_relocation(void *pe_memory, IMAGE_NT_HEADERS *nt_ptr)
{
    DWORD relocation_table_rva = nt_ptr-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAddress;
    if(relocation_table_rva == 0) {
        // 没有重定位表
        return true;
    }
```

```
    IMAGE_BASE_RELOCATION *relocation = (IMAGE_BASE_RELOCATION
*)rva2va(pe_memory, relocation_table_rva);

    while(relocation->VirtualAddress) {
        printf("[+]修正重定位，命中分页: %p\n", rva2va(pe_memory, relocation-
>VirtualAddress));
        PWORD block_item = (PWORD)((BYTE *)relocation +
sizeof(IMAGE_BASE_RELOCATION));
        DWORD item_count = (relocation->SizeOfBlock - sizeof(relocation-
>VirtualAddress)) / sizeof(WORD);

        for(DWORD i = 0; i < item_count; i++, block_item++) {
            if((*block_item >> 12) == IMAGE_REL_BASED_HIGHLOW) {
                // 需要重定位
                PDWORD rel_addr = (PDWORD)rva2va(pe_memory, (relocation-
>VirtualAddress + (*block_item & 0x0fff))); // 需要修复的位置
                *rel_addr = *rel_addr + ((DWORD)pe_memory - nt_ptr-
>OptionalHeader.ImageBase);
                printf("  [+]修正重定位[%p] -> %p\n", rel_addr, *rel_addr);
            }
        }

        relocation = (IMAGE_BASE_RELOCATION *)((BYTE *)relocation + relocation-
>SizeOfBlock);   // 指向下一个重定位表块
    }

    return true;
}
```