也就是说，类似屏显 4-15 的内容，就是通过解读. rela. text 节（或. rela. eh_frame 节）中的多个 Elf64_Rela 结构来完成的。静态重定位表只存在于可重定位目标文件中，可执行文件中是没有静态重定位表的。如果说 objdump 查看重定位表的时候没有显式地指出重定位表所在的节，那么 readelf 则在输出中明确指出重定位节. rela. text 和. rela. eh_frame，如屏显 4-17 所示。

屏显 4-17　readelf 查看重定位表内容及所在的节

```
[root@localhost cs2]#readelf -r main-lib.o

重定位节 '.rela.text' 位于偏移量 0x2f8 含有 8 个条目：
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000000a  000a0000000a R_X86_64_32       0000000000000004 z + 0
00000000000f  000b0000000a R_X86_64_32       0000000000000000 y + 0
000000000014  000c0000000a R_X86_64_32       0000000000000008 x + 0
000000000019  000d00000002 R_X86_64_PC32     0000000000000000 addvec - 4
00000000001f  000a00000002 R_X86_64_PC32     0000000000000004 z + 0
000000000025  000a00000002 R_X86_64_PC32     0000000000000004 z - 4
00000000002a  00050000000a R_X86_64_32       0000000000000000 .rodata.str1.1 + 0
000000000034  000e00000002 R_X86_64_PC32     0000000000000000 printf - 4

重定位节 '.rela.eh_frame' 位于偏移量 0x3b8 含有 1 个条目：
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000020  000200000002 R_X86_64_PC32     0000000000000000 .text + 0
[root@localhost cs2]#
```

注意：虽然前面用“符号”同时指代符号定义和符号引用，但此时读者需要理清符号定义和符号引用。前者是指变量或函数本身（及其地址），而符号引用则是访问这些符号的地址。也就是说，符号表中的符号是指定义，而重定位表中的符号指的是符号的引用。

根据前面分析可知，在完成布局后，链接器会根据. rela. text 节重定位表中记录的所需修正的位置和所引用的符号，查找. symtab 表得到对应符号的地址，然后将地址修正为正确的地址。

4.2.3　符号表

因为符号解析和重定位都针对符号进行，因此需要单独的符号表来管理。符号表需要记录符号名、是否本模块定义、如果本模块定义的则记录其所在位置等，以加快内部的重定位和外部的符号解析（查找）。但符号名字符串长短不一不便于管理，因此无论是编