3-目标文件内容

简单目标文件

- 源文件

```c
int printf(const char* format, ...);

int global_init_var = 84;
int global_uninit_var;

void func1(int i) {
    printf("%d", i);
}

int main(void) {
    static int static_var = 85;
    static int static_uninit_var;

    int a = 1;
    int b;

    func1(static_var + static_uninit_var + a + b);

    return a;
}
```

- 显示反汇编内容 objdump -x -s -d *.o

```
SimpleSection.o:     file format elf64-x86-64
SimpleSection.o
architecture: i386:x86-64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000

# 目标文件的Section信息
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000062  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
```

```
  1 .data         00000008  0000000000000000  0000000000000000  000000a4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000008  0000000000000000  0000000000000000  000000ac  2**2
                  ALLOC
  3 .rodata       00000003  0000000000000000  0000000000000000  000000ac  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      0000002c  0000000000000000  0000000000000000  000000af  2**0
                  CONTENTS, READONLY
  5 .note.GNU-stack 00000000  0000000000000000  0000000000000000  000000db
2**0
                  CONTENTS, READONLY
  6 .note.gnu.property 00000020  0000000000000000  0000000000000000  000000e0
 2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .eh_frame     00000058  0000000000000000  0000000000000000  00000100  2**3
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA

# 符号表
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 SimpleSection.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000004 l     O .data  0000000000000004 static_var.1
0000000000000004 l     O .bss   0000000000000004 static_uninit_var.0
0000000000000000 g     O .data  0000000000000004 global_init_var
0000000000000000 g     O .bss   0000000000000004 global_uninit_var
0000000000000000 g     F .text  000000000000002b func1
0000000000000000         *UND*  0000000000000000 printf
000000000000002b g     F .text  0000000000000037 main

# Section内容
Contents of section .text:
 0000 f30f1efa 554889e5 4883ec10 897dfc8b  ....UH..H....}..
 0010 45fc89c6 488d0500 00000048 89c7b800  E...H......H....
 0020 000000e8 00000000 90c9c3f3 0f1efa55  ...............U
 0030 4889e548 83ec10c7 45f80100 00008b15  H..H....E.......
 0040 00000000 8b050000 000001c2 8b45f801  .............E..
 0050 c28b45fc 01d089c7 e8000000 008b45f8  ..E...........E.
 0060 c9c3                                 ..
Contents of section .data:
 0000 54000000 55000000                    T...U...
Contents of section .rodata:
 0000 256400                               %d.
Contents of section .comment:
 0000 00474343 3a202855 62756e74 75203131  .GCC: (Ubuntu 11
```

```
 0010 2e342e30 2d317562 756e7475 317e3232  .4.0-1ubuntu1~22
 0020 2e303429 2031312e 342e3000           .04) 11.4.0.
Contents of section .note.gnu.property:
 0000 04000000 10000000 05000000 474e5500  ............GNU.
 0010 020000c0 04000000 03000000 00000000  ................
Contents of section .eh_frame:
 0000 14000000 00000000 017a5200 01781001  .........zR..x..
 0010 1b0c0708 90010000 1c000000 1c000000  ................
 0020 00000000 2b000000 00450e10 8602430d  ....+....E....C.
 0030 06620c07 08000000 1c000000 3c000000  .b..........<...
 0040 00000000 37000000 00450e10 8602430d  ....7....E....C.
 0050 066e0c07 08000000                    .n......

Disassembly of section .text:

0000000000000000 <func1>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 83 ec 10             sub    $0x10,%rsp
   c:   89 7d fc                mov    %edi,-0x4(%rbp)
   f:   8b 45 fc                mov    -0x4(%rbp),%eax
  12:   89 c6                   mov    %eax,%esi
  14:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # 1b <func1+0x1b>
                        17: R_X86_64_PC32       .rodata-0x4
  1b:   48 89 c7                mov    %rax,%rdi
  1e:   b8 00 00 00 00          mov    $0x0,%eax
  23:   e8 00 00 00 00          call   28 <func1+0x28>
                        24: R_X86_64_PLT32      printf-0x4
  28:   90                      nop
  29:   c9                      leave
  2a:   c3                      ret

000000000000002b <main>:
  2b:   f3 0f 1e fa             endbr64
  2f:   55                      push   %rbp
  30:   48 89 e5                mov    %rsp,%rbp
  33:   48 83 ec 10             sub    $0x10,%rsp
  37:   c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp)
  3e:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 44 <main+0x19>
                        40: R_X86_64_PC32       .data
  44:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 4a <main+0x1f>
                        46: R_X86_64_PC32       .bss
  4a:   01 c2                   add    %eax,%edx
  4c:   8b 45 f8                mov    -0x8(%rbp),%eax
  4f:   01 c2                   add    %eax,%edx
  51:   8b 45 fc                mov    -0x4(%rbp),%eax
```

```
  54:   01 d0                   add    %edx,%eax
  56:   89 c7                   mov    %eax,%edi
  58:   e8 00 00 00 00          call   5d <main+0x32>
                        59: R_X86_64_PLT32      func1-0x4
  5d:   8b 45 f8                mov    -0x8(%rbp),%eax
  60:   c9                      leave
  61:   c3                      ret
```

- func1函数参数，将int和long类型相加，将Int类型隐式提升为long类型，所以分配8字节空间

```c
int global_init_var = 84;
int global_init_var2 = 88;

void func1(int i) {
    printf("%d", i);
}

int main(void) {
    static int static_var4 = 86;
    static long static_var = 89;
    static long static_var3 = 85;

    static int static_uninit_var;

    int a = 1;
    int b;

    func1(static_var4 + static_uninit_var + a + b);

    return a;
}

Contents of section .data:
 0000 54000000 58000000 56000000 00000000  T...X...V.......
 0010 55000000 00000000 59000000 00000000  U.......Y.......

int global_init_var = 84;
int global_init_var2 = 88;

void func1(int i) {
    printf("%d", i);
}

int main(void) {
    static int static_var4 = 86;
```

```
    static long static_var = 89;
    static long static_var3 = 85;

    static int static_uninit_var;

    int a = 1;
    int b;

    func1(static_var + static_uninit_var + a + b);

    return a;
}
 Contents of section .data:
 0000 54000000 58000000 59000000 00000000  T...X...Y.......
 0010 55000000 00000000 56000000           U.......V...
```

自定义段

GCC提供了__attribute__((section("FOO")))机制

```
__attribute__((section("FOO"))) int global = 42;
__attribute__((section("FOO"))) void foo();
```

ELF文件头

- Magic：DEL控制符，E，L，F，64位程序(02)，小端(01)，ELF版本号(01)
- e_type：
  - ET_REL：可重定位文件，一般为.o文件
  - ET_EXEC：可执行文件
  - ET_DYN：共享目标文件，一般为.so文件

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
```

```
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1192 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         14
  Section header string table index: 13
```

段表

- readelf -S *.o

```
There are 14 section headers, starting at offset 0x4a8:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000062  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000388
       0000000000000078  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000a8
       0000000000000020  0000000000000000  WA       0     0     8
  [ 4] .bss              NOBITS           0000000000000000  000000c8
       0000000000000008  0000000000000000  WA       0     0     4
  [ 5] .rodata           PROGBITS         0000000000000000  000000c8
       0000000000000003  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS         0000000000000000  000000cb
       000000000000002c  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  000000f7
       0000000000000000  0000000000000000           0     0     1
  [ 8] .note.gnu.pr[...] NOTE             0000000000000000  000000f8
       0000000000000020  0000000000000000   A       0     0     8
  [ 9] .eh_frame         PROGBITS         0000000000000000  00000118
       0000000000000058  0000000000000000   A       0     0     8
  [10] .rela.eh_frame    RELA             0000000000000000  00000400
       0000000000000030  0000000000000018   I      11     9     8
  [11] .symtab           SYMTAB           0000000000000000  00000170
       0000000000000180  0000000000000018          12    10     8
  [12] .strtab           STRTAB           0000000000000000  000002f0
```

```
       0000000000000093  0000000000000000           0     0     1
  [13] .shstrtab         STRTAB           0000000000000000  00000430
       0000000000000074  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), l (large), p (processor specific)
```

```
----------
ELF Header
(size 0x40)
---------- 0x40
.text
---------- 0xa2
---------- 0xa8
.data
---------- 0xc8
.bss
---------- 0xcb
.comment
---------- 0xf7
.note.GNU-stack
---------- 0xf8
.note.gnu.pr[...]
---------- 0x118
.eh_frame
---------- 0x170
.symtab
---------- 0x2f0
.strtab
---------- 0x383
---------- 0x388
.rela.text
---------- 0x400
.rela.eh_frame
---------- 0x430
.shstrtab
---------- 0x4a4
---------- 0x4a8
Section Table
----------
```

重定位表

.rela.text是对.text重定位的表

符号表

nm *.o

```
0000000000000000 T func1
0000000000000000 D global_init_var
0000000000000004 D global_init_var2
0000000000000000 B global_uninit_var
000000000000002b T main
                 U printf
0000000000000004 b static_uninit_var.2
0000000000000018 d static_var.0
0000000000000010 d static_var3.1
0000000000000008 d static_var4.3
```

特殊符号

- __executable_start，程序起始地址
- __etext, _etext, etext，代码段结束地址
- _edata, edata，数据段结束地址
- _end, end，程序结束地址

extern "C"

使用__cplusplus宏来判断是否是C++代码

```
#ifdef __cplusplus
extern "C" {
#endif
    void *memset (void*, int ,size_t);
#ifdef __cplusplus
}
#endif
```

强符号和弱符号

- 不允许强符号被多次定义
- 有强符号和弱符号，选择强符号
- 一个符号在所有目标文件都是弱符号，选择其中占据空间最大的一个

强引用和弱引用

- 强引用：如果未找到符号定义，直接报错
- 弱引用：未定义弱引用，编译器不报错，默认为0或是特殊值，与COMMON块联系紧密

```
1   __attribute__((weakref)) void foo();
```