1. 简介

1.1 关键sections

那什么是所谓 sections 呢？可以说，sections 是在ELF文件里头，用以装载内容数据的最小容器。

在ELF文件里面，每一个 sections 内都装载了性质属性都一样的内容，比如：

- **.text 里装载了可执行代码；编译后执行语句都编译成机器代码，保存在.text中**
- **.data 里面装载了被初始化的数据；初始化的全局变量和静态局部变量都存储在.data中**
- **.bss 里面装载了未被初始化的数据；未初始化的全局变量和静态局部变量都存储在.bss中。未初始化的全局变量和局部静态变量默认值为0，本来他们也是可以被放在.data中，但是因为他们都是0，所以在.data中分配空间存在数据0是没有必要的。程序在运行时他们在确定要占内存空间，并且可执行文件必须记录所有未初始化的全局变量和局部静态变量的大小总和，即为.bss。所以.bss只是为未初始化的全局变量和局部静态变量预留位置而已，它并没有内容，所以他在文件中也不占据空间。**
- 以 .rel 打头的 sections 里面装载了重定位条目；
- .symtab 或者 .dynsym section 里面装载了符号信息；
- .strtab 或者 .dynstr section 里面装载了字符串信息；
- .rodata只读数据区
- .comment注释信息
- .note.GNU-stack堆栈提示
- 其他还有为满足不同目的所设置的section，比方满足调试的目的、满足动态链接与加载的目的等等。
- .bss：保存未初始化的数据，比如那些未初始化的全局变量。因为是"未初始化"，所以也没必要在文件中占用任何空间去记录其初始值（所以类型为 SHT_NOBITS）。在程序开始运行时，系统会将 .bss 映射的内存区域清零。
- .comment：保存版本控制信息。
- **.data/.data1：保存已初始化的数据。它们会在文件中占用存储空间，这与 .bss 不同。**
- .debug：保存调试相关的信息。
- .fini / .init：分别保存进程退出和初始化时要执行的指令。.init 指令会在程序入口点(main)之前被执行。
- **.got：保存全局偏移量表 (global offset table)。**
- ~~在 Android 中，GOT 分为两部分：.got 和 .got.plt。其中 .got 表用来保存全局变量引用的地址，而 .got.plt 用来保存函数引用的地址。~~
- **.hash：保存符号哈希表，用于快速查找与其对应的符号表中的符号。**
- .interp：保存 ELF 程序解释器（比如 Android 下的动态链接器）的路径名。
- .line：保存用于调试的行号信息。
- .note：保存一些注释信息。
- **.plt：保存过程链接表 (procedure linkage table)。每个外部定义的函数都会在 PLT 中有对应的一项，用于定位外部函数的地址。**

- **.relname/.relaname**：保存重定位表。比如：**.rel.dyn**、**.rel.plt**。
- .rodata / .rodata1：保存程序中的只读数据。
- **.shstrtab**：存储Section名称的字符串。保存一个字符串表，这些字符串都是 **section** 的名字。
- **.strtab**：保存的是普通的字符串。保存字符串表，类似于 **.dynstr**，但 **.dynstr** 中保存的都是那些需要动态链接的符号的名字。
- **.dynstr**：存储的是符号名称的字符串。比如符号表中的每个符号都有一个 **st_name(符号名)**，他是指向字符串表的索引，这个字符串表可能就保存在 **.dynstr**。
- **.dynamic**：保存动态链接信息。
- **.dynsym**：保存需要动态连接的符号表。.dynsym包含.symtab的一个子集, 比如共享库所需要在runtime加载的函数对应的symbols, 它是allocable的. 共享库包含的.dynsym是runtime必需的, 是allocable的。
- **.symtab**：包含大量linker,debugger需要的数据, 但并不为runtime必需, 它是non-allocable的；保存符号表（非动态链接）。

.symtab包含大量linker,debugger需要的数据, 但并不为runtime必需, 它是non-allocable的; .dynsym包含.symtab的一个子集, 比如共享库所需要在runtime加载的函数对应的symbols, 它是allocable的. 共享库包含的.dynsym是runtime必需的, 是allocable的。

- .text：保存可执行的指令代码。

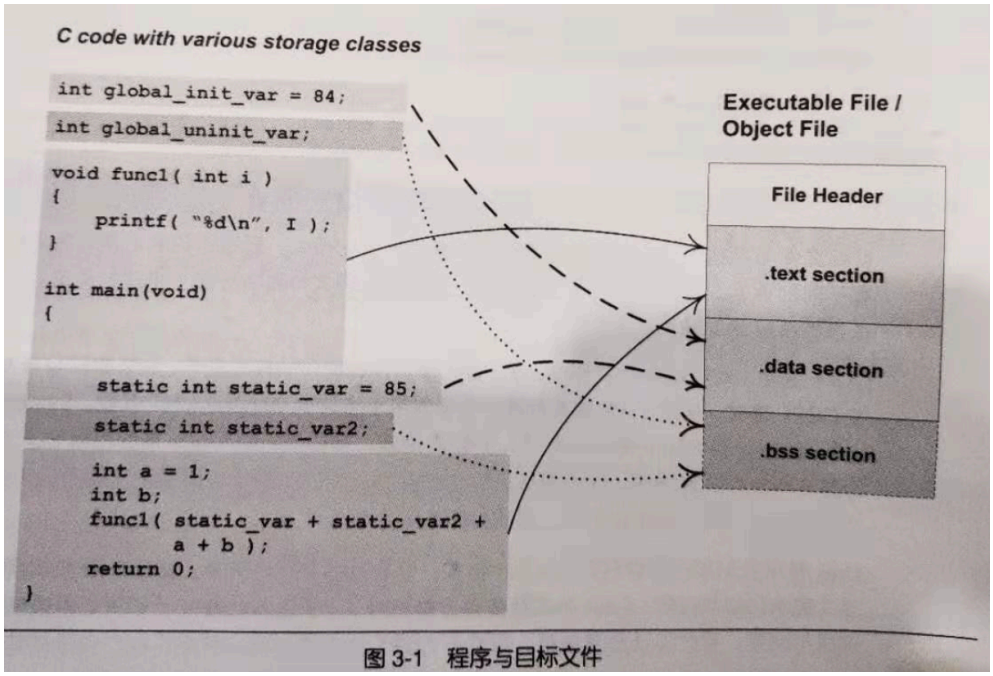

图 3-1 程序与目标文件

1.2 示例

使用**readelf -S *.so**命令查看所有的sections

```
yananhdeMacBook-Pro:armeabi-v7a yananh$ readelf -S libtest.so
There are 25 section headers, starting at offset 0x2284:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .note.android.ide NOTE            00000134 000134 000098 00   A  0   0  4
  [ 2] .note.gnu.build-i NOTE            000001cc 0001cc 000024 00   A  0   0  4
  [ 3] .dynsym           DYNSYM          000001f0 0001f0 000100 10   A  4   1  4
  [ 4] .dynstr           STRTAB          000002f0 0002f0 0000ec 00   A  0   0  1
  [ 5] .hash             HASH            000003dc 0003dc 000054 04   A  3   0  4
  [ 6] .gnu.version      VERSYM          00000430 000430 000020 02   A  3   0  2
  [ 7] .gnu.version_d    VERDEF          00000450 000450 00001c 00   A  4   1  4
  [ 8] .gnu.version_r    VERNEED         0000046c 00046c 000020 00   A  4   1  4
  [ 9] .rel.dyn          REL             0000048c 00048c 000048 08   A  3   0  4
  [10] .rel.plt          REL             000004d4 0004d4 000050 08  AI  3  18  4
  [11] .plt              PROGBITS        00000524 000524 00008c 00  AX  0   0  4
  [12] .text             PROGBITS        000005b0 0005b0 0015a4 00  AX  0   0  4
  [13] .ARM.extab        PROGBITS        00001b54 001b54 00003c 00   A  0   0  4
  [14] .ARM.exidx        ARM_EXIDX       00001b90 001b90 000100 08  AL 12   0  4
  [15] .rodata           PROGBITS        00001c90 001c90 00000a 01 AMS  0   0  1
  [16] .fini_array       FINI_ARRAY      00002ea4 001ea4 000004 04  WA  0   0  4
  [17] .dynamic          DYNAMIC         00002ea8 001ea8 000108 08  WA  4   0  4
  [18] .got              PROGBITS        00002fb0 001fb0 000050 00  WA  0   0  4
  [19] .data             PROGBITS        00003000 002000 000004 00  WA  0   0  4
  [20] .bss              NOBITS          00003004 002004 000000 00  WA  0   0  1
  [21] .comment          PROGBITS        00000000 002004 00012f 01  MS  0   0  1
  [22] .note.gnu.gold-ve NOTE            00000000 002134 00001c 00      0   0  4
  [23] .ARM.attributes   ARM_ATTRIBUTES  00000000 002150 000036 00      0   0  1
  [24] .shstrtab         STRTAB          00000000 002186 0000fe 00      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),
  y (noread), p (processor specific)
```

1.3 SHT和Sections的关系

节区中包含目标文件中的所有信息，除了:ELF 头部、程序头部表格、节区头部表格。节区满足以下条件:

1. 目标文件中的每个节区都有对应的节区头部描述它，反过来，有节区头部不意味着有节区。
2. 每个节区占用文件中一个连续字节区域(这个区域可能长度为 0)。
3. 文件中的节区不能重叠，不允许一个字节存在于两个节区中的情况发生。
4. 目标文件中可能包含非活动空间(INACTIVE SPACE)。这些区域不属于任何

1.4 Sections与ELF head映射

ELF head中出现了SHT的描述

ELF 头部中，**e_shoff** 成员给出从文件头到节区头部表格的偏移字节数;**e_shnum** 给出表格中条目数目;**e_shentsize** 给出每个项目的字节数。从这些信息中可以确切地定 位节区的具体位置、长度。

e_shoff: 节区头部表格(SHT)的偏移量。节区头部表格(Section Header Table)的偏移量(按字节计算)。如果文件 没有节区头部表格，可以为 0。

e_shentsize: 节区头部表格的表项大小。节区头部表格的表项大小(按字节计算)。即Section Header Table中没一项的大小

e_shnum: 节区头部表格的表项数量。节区头部表格的表项数目，可以为0(只能运行时加载，不能用作编译链接，一般用用于so的裁剪)。

e_shstrndx: 节区头部表格中与节区名称字符串表相关的表项的索引。节区头部表格中与节区名称字符串表相关的表项的索引。如果文件没有节区名称字符串表，此参数可以为 SHN_UNDEF。

2 SHT(section head table)结构

一个ELF文件中到底有哪些具体的 sections，由包含在这个ELF文件中的section head table(SHT)决定。

在SHT中，针对每一个section，都设置有一个条目，用来描述对应的这个section，其内容主要包括该 section 的名称、类型、大小以及在整个ELF文件中的字节偏移位置等等。我们也可以在TISCv1.2规范中找到SHT表中条目的C结构定义：

```
typedef struct{
    Elf32_Word sh_name;  //节区名，段名是一个字符串，它位于一个叫做.shstrtab的字符串表。sh_name段名字符串在是".shstrtab"中的偏移。节区头部字符串表节区（Section Header String Table Section）的索引。名字是一个 NULL 结尾的字符串。
    Elf32_Word sh_type;  //为节区类型, 1) PROGBITS,表示此section的内容由程序来解释,.text和.data都是; 2) NOBITS,表示此段不占用文件空间 3) STRTAB是表示字符串表;4) SYMTAB表示符号表等等
    Elf32_Word sh_flags;  //节区标志。可以在该命令的最末看出不同flag标志的含义。其中Alloc，表示该section在进程执行过程中占用内存
    Elf32_Addr sh_addr;  //节区虚拟地址。如果节区可以被加载，此成员为被加载后在进程地址空间中的虚拟地址；否则，此字段为 0。
    Elf32_Off sh_offset;  //节区偏移。节区的第一个字节与文件头之间的偏移。如果该section存在于文件中，则表示再文件中的偏移；否则无意义。比如sh_offset对于BBS Section来说就没有意义。
    Elf32_Word sh_size;  //节区的长度（字节数）。
    Elf32_Word sh_link;  //此成员给出节区头部表索引链接。其具体的解释依赖于节区类型。
    Elf32_Word sh_info;    //此成员给出附加信息，其解释依赖于节区类型。
```

Elf32_Word sh_addralign; //某些节区带有地址对齐约束。有些Section对地址对齐有要求，比如我们假设有个Section起始位置包含一个double变量，因为Intel x86系统要求浮点数的存储地址必须是本身的整数倍，也就是说保存double变量地址的必须是8字节的整数倍。这样对一个Section来说，它的sh_addr必须是8的整数倍。</p>由于地址对齐的数量都是2的指数倍，sh_addralign表示是地址对齐数量中的指数，即sh_addralign==3表示对齐为2的3次方倍，即8倍，依此类推。所以一个段的地址sh_addr必须满足下面的条件，即sh_addr%(2 ** sh_addralign)==0。**表示指数运算。</p>如果sh_addralign为0或1，则表示该Section没有对齐要求。

Elf32_Word sh_entsize; //Section Entry Size 项的长度。有些Section包含了一些固定大小的项，比如符号表，它包含的每个符号所占的大小都一样。对于这种Section，sh_entsize表示每个项的大小。如果为0，则表示该Section不包含固定大小的项。

} Elf32_Shdr;

在010Editor中的展示为：

struct section_header_table		2284h	3E8h
struct section_table_entry32_t section_table_element[0]	SHN_UNDEF	2284h	28h
struct section_table_entry32_t section_table_element[1]	.note.android.ident	22ACh	28h
struct section_table_entry32_t section_table_element[2]	.note.gnu.build-id	22D4h	28h
struct section_table_entry32_t section_table_element[3]	.dynsym	22FCh	28h
struct section_table_entry32_t section_table_element[4]	.dynstr	2324h	28h
struct section_table_entry32_t section_table_element[5]	.hash	234Ch	28h
struct section_table_entry32_t section_table_element[6]	.gnu.version	2374h	28h
struct section_table_entry32_t section_table_element[7]	.gnu.version_d	239Ch	28h
struct section_table_entry32_t section_table_element[8]	.gnu.version_r	23C4h	28h
struct section_table_entry32_t section_table_element[9]	.rel.dyn	23ECh	28h
struct section_table_entry32_t section_table_element[10]	.rel.plt	2414h	28h
struct section_table_entry32_t section_table_element[11]	.plt	243Ch	28h
struct section_table_entry32_t section_table_element[12]	.text	2464h	28h
struct section_table_entry32_t section_table_element[13]	.ARM.extab	248Ch	28h
struct section_table_entry32_t section_table_element[14]	.ARM.exidx	24B4h	28h
struct section_table_entry32_t section_table_element[15]	.rodata	24DCh	28h
struct section_table_entry32_t section_table_element[16]	.fini_array	2504h	28h
struct section_table_entry32_t section_table_element[17]	.dynamic	252Ch	28h
struct section_table_entry32_t section_table_element[18]	.got	2554h	28h
struct section_table_entry32_t section_table_element[19]	.data	257Ch	28h
struct section_table_entry32_t section_table_element[20]	.bss	25A4h	28h
struct section_table_entry32_t section_table_element[21]	.comment	25CCh	28h
struct section_table_entry32_t section_table_element[22]	.note.gnu.gold-versi...	25F4h	28h
struct section_table_entry32_t section_table_element[23]	.ARM.attributes	261Ch	28h
struct section_table_entry32_t section_table_element[24]	.shstrtab	2644h	28h

2.1 三大核心(type, flags, addr)

- type:
  1. PROGBITS,表示此section的内容由程序来解释,.text和.data都是
  2. NOBITS,表示此段不占用文件空间
  3. STRTAB是表示字符串表

4. SYMTAB表示符号表等等

- flags: 可以在该命令的最末看出不同flag标志的含义。其中Alloc，表示该section在进程执行过程中占用内存
- addr : 如果section将出现在进程的内存映像中，此成员给出section的第一个字节应处的位置。

2.2 sh_name: 节区名在节区头部字符串表节区(shstrtab)的索引

shstrtab在哪里：从ELF 头部文件中的e_shstrndx能够获取该section的索引

还是从libtest.so为例，从ELF Head中e_shstrndx为24， 找到该section的内容，发现如下：

struct section_table_entry32_t section_table_element[24]	.shstrtab	2644h	28h	Fg:	Bg:
struct s_name32_t s_name	.shstrtab	2644h	4h	Fg:	Bg:
enum s_name32_e s_name_off	1h	2644h	4h	Fg:	Bg:
string s_name_str[10]	.shstrtab	2187h	Ah	Fg:	Bg:
enum s_type32_e s_type	SHT_STRTAB (3)	2648h	4h	Fg:	Bg:
enum s_flags32_e s_flags	SF32_None (0)	264Ch	4h	Fg:	Bg:
Elf32_Addr s_addr	0x00000000	2650h	4h	Fg:	Bg:
Elf32_Off s_offset	2186h	2654h	4h	Fg:	Bg:
Elf32_Xword s_size	254	2658h	4h	Fg:	Bg:
Elf32_Word s_link	0	265Ch	4h	Fg:	Bg:
Elf32_Word s_info	0	2660h	4h	Fg:	Bg:
Elf32_Xword s_addralign	1	2664h	4h	Fg:	Bg:
Elf32_Xword s_entsize	0	2668h	4h	Fg:	Bg:
char s_data[254]		2186h	FEh	Fg:	Bg:

s_name代表的索引为1，通过**readelf -p 24 libtest.so**获取shstrtab section的内容如下：

```
yananhdeMacBook-Pro:armeabi-v7a yananh$ readelf -p 24 libtest.so

String dump of section '.shstrtab':
  [     1]  .shstrtab
```

*有意思了，shstrtab的名称竟然包含在其属性中，s_name中是shstrtab的第一个，指向的恰好是**.shstrtab***

2.3 sh_type：节区(Section)类型

为节区的内容和语义进行分类，也就是说section name并不进行任何属性绑定，属性绑定的是type

名称	取值	说明
SHT_NULL	0	此值标志节区头部是非活动的，没有对应
SHT_PROGBITS	1	此section的内容由程序来解释 程序段、代码段、数据段都是这种类型的
SHT_SYMTAB	2	该节区内容为符号表 目前目标文件对每种类型的节区都只能包 般，SHT_SYMTAB 节区提供用于链接编 接。
SHT_STRTAB	3	该节区内容为字符串表 目标文件可能包含多个字符串表节区。
SHT_RELA	4	该节区内容为重定位表 其中可能会有补齐内容（addend），例 拥有多个重定位节区。
SHT_HASH	5	此节区包含符号哈希表 所有参与动态链接的目标都必须包含一个 表，不过此限制将来可能会解除。
SHT_DYNAMIC	6	动态链接信息 目前一个目标文件中只能包含一个动态节
SHT_NOTE	7	提示性信息 此节区包含以某种方式来标记文件的信息
SHT_NOBITS	8	该节区再文件中没有内容，比如.bss段 这种类型的节区不占用文件中的 节区不包含任何字节，成员sh_offset 中
SHT_REL	9	该段包含重定位信息 此节区包含重定位表项，其中没有补齐 型。目标文件中可以拥有多个重定位节区
SHT_SHLIB	10	保留 此节区被保留，不过其语义是未规定的。

2.3.1 示例

```
yananhdeMacBook-Pro:armeabi-v7a yananh$ readelf -S libtest.so
There are 25 section headers, starting at offset 0x2284:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .note.android.ide NOTE            00000134 000134 000098 00   A  0   0  4
  [ 2] .note.gnu.build-i NOTE            000001cc 0001cc 000024 00   A  0   0  4
  [ 3] .dynsym           DYNSYM          000001f0 0001f0 000100 10   A  4   1  4
  [ 4] .dynstr           STRTAB          000002f0 0002f0 0000ec 00   A  0   0  1
```

2.4 sh_flag：节区在进程虚拟地址空间的属性

该节区在进程虚拟地址空间中的属性，比如是否可写，是否可执行等。相关常量以SHF开头，定义如下：

名称	取值	含义
SHF_WRITE	0x1	节区包含进程执行过程中将可写的数据。
SHF_ALLOC	0x2	表示该节区再进程空间中需要分配空间 某些指示或控制节区并不出现于目标文件的 此位应设置为 0。像代码段、数据段和.bss.
SHF_EXECINSTR	0x4	该节区再进程空间中可以被执行，一般指代 节区包含可执行的机器指令
SHF_MASKPROC	0xF0000000	所有包含于此掩码中的四位都用于处理器专

2.4.1 示例

```
[yananhdeMacBook-Pro:armeabi-v7a yananh$ readelf -S libtest.so
There are 25 section headers, starting at offset 0x2284:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .note.android.ide NOTE            00000134 000134 000098 00   A  0   0  4
  [ 2] .note.gnu.build-i NOTE            000001cc 0001cc 000024 00   A  0   0  4
  [ 3] .dynsym           DYNSYM          000001f0 0001f0 000100 10   A  4   1  4
  [ 4] .dynstr           STRTAB          000002f0 0002f0 0000ec 00   A  0   0  1
  [ 5] .hash             HASH            000003dc 0003dc 000054 04   A  3   0  4
  [ 6] .gnu.version      VERSYM          00000430 000430 000020 02   A  3   0  2
  [ 7] .gnu.version_d    VERDEF          00000450 000450 00001c 00   A  4   1  4
  [ 8] .gnu.version_r    VERNEED         0000046c 00046c 000020 00   A  4   1  4
  [ 9] .rel.dyn          REL             0000048c 00048c 000048 08   A  3   0  4
  [10] .rel.plt          REL             000004d4 0004d4 000050 08  AI  3  18  4
  [11] .plt              PROGBITS        00000524 000524 00008c 00  AX  0   0  4
  [12] .text             PROGBITS        000005b0 0005b0 0015a4 00  AX  0   0  4
  [13] .ARM.extab        PROGBITS        00001b54 001b54 00003c 00   A  0   0  4
  [14] .ARM.exidx        ARM_EXIDX       00001b90 001b90 000100 08  AL 12   0  4
  [15] .rodata           PROGBITS        00001c90 001c90 00000a 01 AMS  0   0  1
  [16] .fini_array       FINI_ARRAY      00002ea4 001ea4 000004 04  WA  0   0  4
  [17] .dynamic          DYNAMIC         00002ea8 001ea8 000108 08  WA  4   0  4
  [18] .got              PROGBITS        00002fb0 001fb0 000050 00  WA  0   0  4
  [19] .data             PROGBITS        00003000 002000 000004 00  WA  0   0  4
  [20] .bss              NOBITS          00003004 002004 000000 00  WA  0   0  1
  [21] .comment          PROGBITS        00000000 002004 00012f 01  MS  0   0  1
  [22] .note.gnu.gold-ve NOTE            00000000 002134 00001c 00      0   0  4
  [23] .ARM.attributes   ARM_ATTRIBUTES  00000000 002150 000036 00      0   0  1
  [24] .shstrtab         STRTAB          00000000 002186 0000fe 00      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),
  y (noread), p (processor specific)
```

2.5 sh_link 和 sh_info： Section的类型和链接相关

Section的类型和链接相关，比如重定位表、符号表等，那么sh_link和sh_info两个成员包含的意义如下。对于其他节区，这两个成员没有意义。

sh_type	sh_link	sh_info
SHT_DYNAMIC	该节区所使用的字符串表在节区表(Section Header Table)中的下标	0
SHT_HASH	该节区使用的符号表在节区表(Section Header Table)	0

2.5.1 示例

```
yananhdeMacBook-Pro:armeabi-v7a yananh$ readelf -S libtest.so
There are 25 section headers, starting at offset 0x2284:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .note.android.ide NOTE            00000134 000134 000098 00   A  0   0  4
  [ 2] .note.gnu.build-i NOTE            000001cc 0001cc 000024 00   A  0   0  4
  [ 3] .dynsym           DYNSYM          000001f0 0001f0 000100 10   A  4   1  4
  [ 4] .dynstr           STRTAB          000002f0 0002f0 0000ec 00   A  0   0  1
  [ 5] .hash             HASH            000003dc 0003dc 000054 04   A  3   0  4
  [ 6] .gnu.version      VERSYM          00000430 000430 000020 02   A  3   0  2
  [ 7] .gnu.version_d    VERDEF          00000450 000450 00001c 00   A  4   1  4
  [ 8] .gnu.version_r    VERNEED         0000046c 00046c 000020 00   A  4   1  4
  [ 9] .rel.dyn          REL             0000048c 00048c 000048 08   A  3   0  4
  [10] .rel.plt          REL             000004d4 0004d4 000050 08  AI  3  18  4
  [11] .plt              PROGBITS        00000524 000524 00008c 00  AX  0   0  4
  [12] .text             PROGBITS        000005b0 0005b0 0015a4 00  AX  0   0  4
  [13] .ARM.extab        PROGBITS        00001b54 001b54 00003c 00   A  0   0  4
  [14] .ARM.exidx        ARM_EXIDX       00001b90 001b90 000100 08  AL 12   0  4
  [15] .rodata           PROGBITS        00001c90 001c90 00000a 01 AMS  0   0  1
  [16] .fini_array       FINI_ARRAY      00002ea4 001ea4 000004 04  WA  0   0  4
  [17] .dynamic          DYNAMIC         00002ea8 001ea8 000108 08  WA  4   0  4
  [18] .got              PROGBITS        00002fb0 001fb0 000050 00  WA  0   0  4
  [19] .data             PROGBITS        00003000 002000 000004 00  WA  0   0  4
  [20] .bss              NOBITS          00003004 002004 000000 00  WA  0   0  1
  [21] .comment          PROGBITS        00000000 002004 00012f 01  MS  0   0  1
  [22] .note.gnu.gold-ve NOTE            00000000 002134 00001c 00      0   0  4
  [23] .ARM.attributes   ARM_ATTRIBUTES  00000000 002150 000036 00      0   0  1
  [24] .shstrtab         STRTAB          00000000 002186 0000fe 00      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),
  y (noread), p (processor specific)
```

指向索引为4节区，即.dynstr。

指向索引为3节区，即.dynsym

指向索引为3节区，即.dynsym

指向索引为4节区，即.dynstr