

1 数据结构

1.1 ELF文件头格式如下

1.2 e_ident

1.3 e_type: elf文件的类型

1.4 e_machine: 目标体系结构类型

1.5 e_version: 目标文件版本

1.6 e_entry:程序入口的虚拟地址

1.7 e_phoff: 程序头部表格(PHT)的偏移量

1.8 e_shoff: 节区头部表格(SHT)的偏移量

1.9 e_flags

1.10 e_ehsize: ELF头部大小

1.11 e_phentsize: 程序头部表格的表项大小

1.12 e_phnum: 程序头部表格的表项数量

1.13 e_shentsize: 节区头部表格的表项大小

1.14 e_shnum: 节区头部表格的表项数量

1.15 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

2 实例

2.1 readelf -f

2.2 010格式化

1 数据结构

定义了ELF魔数、硬件平台等

入口地址、程序头入口和长度

段表的位置和长度及段的数量

段表字符串表（.shstrtab）所在的段在段表中的下标。

可以在"/usr/include/elf.h"中找到它的定义（Elf32_Ehdr）。

1.1 ELF文件头格式如下

```
#define EI_NIDENT 16
```

```
typedef struct{
```

```
    unsigned char e_ident[EI_NIDENT]; //目标文件标识信息
```

```
    Elf32_Half e_type;                //目标文件类型
```

```
    Elf32_Half e_machine;             //目标体系结构类型
```

```
    Elf32_Word e_version;             //目标文件版本
```

```
    Elf32_Addr e_entry;               //程序入口的虚拟地址,若没有,可为0
```

```
    Elf32_Off e_phoff;               //程序头部表格（Program Header Table）的  
    偏移量（按字节计算）,若没有,可为0
```

```
    Elf32_Off e_shoff;               //节区头部表格（Section Header Table）的偏  
    移量（按字节计算）,若没有,可为0
```

```
    Elf32_Word e_flags;               //保存与文件相关的,特定于处理器的标志。  
    标志名称采用 EF_machine_flag的格式。
```

```
    Elf32_Half e_ehsize;              //ELF 头部的大小（以字节计算）。
```

```
    Elf32_Half e_phentsize;           //程序头部表格的表项大小（按字节计算）。
```

```
    Elf32_Half e_phnum;               //程序头部表格的表项数目。可以为0。
```

```
    Elf32_Half e_shentsize;           //节区头部表格的表项大小（按字节计算）。
```

```
    Elf32_Half e_shnum;               //节区头部表格的表项数目。可以为0。
```

```
    Elf32_Half 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。如果文件没有节区名称字符串表,此参数可以为 SHN_UNDEF。
```

```
} Elf32_Ehdr;
```

1.2 e_ident

e_ident[EI_MAG0]~e_ident[EI_MAG3]即e_ident[0]~e_ident[3]被称为魔数（Magic Number）,其值一般为0x7f,'E','L','F'。

e_ident[EI_CLASS] (即e_ident[4]) 识别目标文件运行在目标机器的类别, 取值可为三种值:

1. ELFCLASSNONE (0) 非法类别;
2. ELFCLASS32 (1) 32位目标;
3. ELFCLASS64 (2) 64位目标。

e_ident[EI_DATA] (即e_ident[5]): 给出处理器特定数据的数据编码方式。即大端还是小端方式。取值可为3种:

1. ELFDATANONE (0) 非法数据编码;
2. ELFDATA2LSB (1) 高位在前;
3. ELFDATA2MSB (2) 低位在前。

其它数组元素就不作介绍了。

1.3 e_type: elf文件的类型

e_type表示elf文件的类型, 如下定义:

名称	取值	含义
ET_NONE	0	未知目标文件格式
ET_REL	1	可重定位文件
ET_EXEC	2	可执行文件
ET_DYN	3	共享目标文件
ET_CORE	4	Core 文件 (转储格式)
ET_LOPROC	0xff00	特定处理器文件
ET_HIPROC	0xffff	特定处理器文件
ET_LOPROC~ET_HIPROC		0xff00~0xffff 特定处理器文件

1.4 e_machine: 目标体系结构类型

e_machine表示目标体系结构类型:

名称	取值	含义
EM_NONE	0	未指定
EM_M32	1	AT&T WE 32100
EM_SPARC	2	SPARC
EM_386	3	Intel 80386
EM_68K	4	Motorola 68000
EM_88K	5	Motorola 88000
EM_860	7	Intel 80860
EM_MIPS	8	MIPS RS3000
others	9~	预留

1.5 e_version: 目标文件版本

目标文件版本

名称	取值	含义
EV_NONE	0	非法版本
EV_CURRENT	1	当前版本

1.6 e_entry: 程序入口的虚拟地址

程序入口的虚拟地址。如果目标文件没有程序入口，可以为 0。

1.7 e_phoff: 程序头部表格(PHT)的偏移量

程序头部表格(Program Header Table)的偏移量(按字节计算)。如果文件没有程序头部表格，可以为 0。

1.8 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。

1.9 e_flags

保存与文件相关的，特定于处理器的标志。标志名称采用 EF_machine_flag 的格式。

1.10 e_ehsize: ELF头部大小

ELF 头部的大小(以字节计算)。一般32位so位十进制的52

1.11 e_phentsize: 程序头部表格的表项大小

程序头部表格的表项大小(按字节计算)。即Program Header Table中每一项的大小

1.12 e_phnum: 程序头部表格的表项数量

程序头部表格的表项数目，可以为 0(只能用作编译链接，不能运行加载)。

1.13 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

1.14 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，一般用于so的裁剪)。

1.15 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 实例

2.1 readelf -f

readelf -h libtest.so

```
yananhdeMacBook-Pro:armeabi-v7a yananh$ readelf -h libtest.so
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF32
  Data:                               2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                        0
  Type:                               DYN (Shared object file)
  Machine:                            ARM
  Version:                            0x1
  Entry point address:                0x0
  Start of program headers:           52 (bytes into file)
  Start of section headers:          8836 (bytes into file)
  Flags:                              0x5000200, Version5 EABI, soft-float ABI
  Size of this header:                52 (bytes)
  Size of program headers:            32 (bytes)
  Number of program headers:          8
  Size of section headers:            40 (bytes)
  Number of section headers:          25
  Section header string table index: 24
```

网上有更详细的示例

```
@<file>          Read options from <file>
-H --help         Display this information
-v --version      Display the version number of readelf

D:\>readelf.exe -h android_server
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF32
  Data:                               2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                        0
  Type:                               DYN (Shared object file)
  Machine:                            ARM
  Version:                            0x1
  Entry point address:                0xbcf0
  Start of program headers:           52 (bytes into file)
  Start of section headers:          522440 (bytes into file)
  Flags:                              0x50000000, Version5 EABI
  Size of this header:                52 (bytes)
  Size of program headers:            32 (bytes)
  Number of program headers:          8
  Size of section headers:            40 (bytes)
  Number of section headers:          26
  Section header string table index: 25
```

El PAD, 为保留字
节的开始, 默认为0

e_ident[16]: 目标文件标识

e_type: 目标文件类型
e_machine: 文件的目标体系结构类型
e_version: 目标文件版本
e_entry: 程序入口的虚拟地址
e_phoff: Program Header Table偏移量
e_shoff: Section Header Table偏移量
e_flags: 处理器相关标识
e_ehsize: ELF Header的大小
e_phnum: Program Header Entry数目
e_shnum: Section Header Entry数目

2.2 010格式化

对libtest.so使用010Editor模板打开后如下:

Name	Value	Start	Size	Color	Comment
▼ struct file		0h	250h	Fg: Bg:	
▼ struct elf_header		0h	34h	Fg: Bg:	The main elf header basically tells us where everything is located i...
▶ struct e_ident.t e_ident		0h	10h	Fg: Bg:	Magic number and other info
enum e_type32.e e_type	ET_DYN (3)	10h	2h	Fg: Bg:	Object file type
enum e_machine32.e e_machine	EM_ARM (40)	12h	2h	Fg: Bg:	Architecture
enum e_version32.e e_version	EV_CURRENT (1)	14h	4h	Fg: Bg:	Object file version
Elf32_Addr e_entry.START_ADDRESS	0x00000000	18h	4h	Fg: Bg:	Entry point virtual address
Elf32_Off e_phoff.PROGRAM_HEADER_OFFSET_IN_FILE	52	1Ch	4h	Fg: Bg:	Program header table file offset
Elf32_Off e_shoff.SECTION_HEADER_OFFSET_IN_FILE	8836	20h	4h	Fg: Bg:	Section header table file offset
Elf32_Word e_flags	83886592	24h	4h	Fg: Bg:	Processor-specific flags
Elf32_Half e_ehsize.ELF_HEADER_SIZE	52	28h	2h	Fg: Bg:	ELF Header size in bytes
Elf32_Half e_phentsize.PROGRAM_HEADER_ENTRY_SIZE_IN_FILE	32	2Ah	2h	Fg: Bg:	Program header table entry size
Elf32_Half e_phnum.NUMBER_OF_PROGRAM_HEADER_ENTRIES	8	2Ch	2h	Fg: Bg:	Program header table entry count
Elf32_Half e_shentsize.SECTION_HEADER_ENTRY_SIZE	40	2Eh	2h	Fg: Bg:	Section header table entry size
Elf32_Half e_shnum.NUMBER_OF_SECTION_HEADER_ENTRIES	25	30h	2h	Fg: Bg:	Section header table entry count
Elf32_Half e_shtrndx.STRING_TABLE_INDEX	24	32h	2h	Fg: Bg:	Section header string table index

2.3 节区头部表格中与节区名称字符串表相关的表项的索引

libtest.so的索引项为24，通过 **readelf -p 24 libtest.so**

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

String dump of section '.shstrtab':

```
[ 1] .shstrtab
[ b] .note.android.ident
[1f] .note.gnu.build-id
[32] .dynsym
[3a] .dynstr
[42] .hash
[48] .gnu.version
[55] .gnu.version_d
[64] .gnu.version_r
[73] .rel.dyn
[7c] .rel.plt
[85] .text
[8b] .ARM.extab
[96] .ARM.exidx
[a1] .rodata
[a9] .fini_array
[b5] .dynamic
[be] .got
[c3] .data
[c9] .bss
[ce] .comment
[d7] .note.gnu.gold-version
[ee] .ARM.attributes
```

能够得到所有的section的描述，通过 **readelf -S libtest.so**

```
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.id	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),
 v (noread), p (processor specific)

可以看到大致可以很好的对应起来

备注：突然发现字符串的处理于java class的常量字符串池惊人的相似，都是将符号等方法到单独的地方，增加复用性