readelf

用于显示elf格式文件的信息

补充说明

readelf命令 用来显示一个或者多个elf格式的目标文件的信息，可以通过它的选项来控制显示哪些信息。这里的elf-file(s)就表示那些被检查的文件。可以支持32位，64位的elf格式文件，也支持包含elf文件的文档（这里一般指的是使用ar命令将一些elf文件打包之后生成的例如lib*.a之类的“静态库”文件）。

这个程序和objdump提供的功能类似，但是它显示的信息更为具体，并且它不依赖BFD库(BFD库是一个GNU项目，它的目标就是希望通过一种统一的接口来处理不同的目标文件)，所以即使BFD库有什么bug存在的话也不会影响到readelf程序。

运行readelf的时候，除了-v和-H之外，其它的选项必须有一个被指定。

ELF文件类型

种类型的ELF文件：

1. 可重定位文件:用户和其他目标文件一起创建可执行文件或者共享目标文件,例如lib*.a文件。
2. 可执行文件：用于生成进程映像，载入内存执行,例如编译好的可执行文件a.out
3. 共享目标文件：用于和其他共享目标文件或者可重定位文件一起生成elf目标文件或者和执行文件一起创建进程映像，例如lib*.so文件。

ELF文件作用：

ELF文件参与程序的连接(建立一个程序)和程序的执行(运行一个程序)，所以可以从不同的角度来看待elf格式的文件：

1. 如果用于编译和链接（可重定位文件），则编译器和链接器将把elf文件看作是节头表描述的节的集合,程序头表可选。
2. 如果用于加载执行（可执行文件），则加载器则将把elf文件看作是程序头表描述的段的集合，一个段可能包含多个节，节头表可选。
3. 如果是共享文件，则两者都含有。

ELF文件总体组成：

elf文件头描述elf文件的总体信息。包括：系统相关，类型相关，加载相关，链接相关。

- 系统相关表示elf文件标识的魔术数，以及硬件和平台等相关信息，增加了elf文件的移植性,使交叉编译成为可能。
- 类型相关就是前面说的那个类型。
- 加载相关：包括程序头表相关信息。
- 链接相关：节头表相关信息。

选项

```
-a
--all 显示全部信息,等价于 -h -l -S -s -r -d -V -A -I.

-h
```

```
--file-header 显示elf文件开始的文件头信息.

-l
--program-headers
--segments 显示程序头（段头）信息(如果有的话)。

-S
--section-headers
--sections 显示节头信息(如果有的话)。

-g
--section-groups 显示节组信息(如果有的话)。

-t
--section-details 显示节的详细信息(-S的)。

-s
--syms
--symbols 显示符号表段中的项（如果有的话）。

-e
--headers 显示全部头信息，等价于: -h -l -S

-n
--notes 显示note段（内核注释）的信息。

-r
--relocs 显示可重定位段的信息。

-u
--unwind 显示unwind段信息。当前只支持IA64 ELF的unwind段信息。

-d
--dynamic 显示动态段的信息。

-V
--version-info 显示版本段的信息。

-A
--arch-specific 显示CPU构架信息。

-D
--use-dynamic 使用动态段中的符号表显示符号，而不是使用符号段。

-x <number or name>
--hex-dump=<number or name> 以16进制方式显示指定段内内容□number指定段表中段的索引,或字符串指定文件中的段名。

-w[liaprmfFsoR] or
--debug-dump[=line,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges] 显示调试段中指定的内容。
```

```
-I
--histogram 显示符号的时候，显示bucket list长度的柱状图。

-v
--version 显示readelf的版本信息。

-H
--help 显示readelf所支持的命令行选项。

-W
--wide 宽行输出。

@file 可以将选项集中到一个文件中，然后使用这个@file选项载入。
```

实例

先给出如下例子：

1.对于可执行文件形式的elf格式文件：

1)查看可执行程序的源代码如下：

```
root@localhost [test]$ cat main.cpp
#include <iostream>
using std::cout;
using std::endl;
void my_print();

int main(int argc, char *argv[])
{
        my_print();
        cout<<"hello!"<<endl;
        return 0;
}

void  my_print()
{
        cout<<"print!"<<endl;
}
```

2)编译如下：

```
[root@localhost test]$ g++ main.cpp -o main
[root@localhost test]$ g++ -g main.cpp -o main.debug
```

3)编译之后，查看生成的文件：

```
[root@localhost test]$ ls -l
总计 64
-rwxr-xr-x 1 quietheart quietheart  6700 07-07 18:04 main
-rw-r--r-- 1 quietheart quietheart   201 07-07 18:02 main.cpp
```

```
-rwxr-xr-x 1 quietheart quietheart 38932 07-07 18:04 main.debug
```

这里，main.debug是带有调试信息的可执行文件，main是一般的可执行文件。

2.对于库文件形式的elf格式文件：

1)查看库的源代码如下：

```
//myfile.h
#ifndef __MYFILE_H
#define __MYFILE_H
void printInfo();
#endif

//myfile.cpp
#include "myfile.h"
#include <iostream>
using std::cout;
using std::endl;
void printInfo()
{
    cout<<"hello"<<endl;
}
```

2)编译如下：

```
[root@localhost test]$ g++ -c myfile.cpp
[root@localhost test]$ g++ -shared -fPCI -o libmy.so myfile.o
[root@localhost test]$ ar -r libmy.a myfile.o
ar: creating libmy.a
```

3)编译之后，查看生成的文件：

[root@localhost test]$ ls -l

总计 44

```
-rw-r--r-- 1 quietheart quietheart 2154 07-08 16:14 libmy.a
-rwxr-xr-x 1 quietheart quietheart 5707 07-08 16:08 libmy.so
-rwxr-xr-x 1 quietheart quietheart  117 07-08 16:06 myfile.cpp
-rwxr-xr-x 1 quietheart quietheart   63 07-08 16:08 myfile.h
-rw-r--r-- 1 quietheart quietheart 2004 07-08 16:08 myfile.o
libmy.a  libmy.so  myfile.cpp  myfile.h  myfile.o
```

这里，分别生成目标文件myfile.o，共享库文件libmy.so，和静态库文件libmy.a。

基于以上可执行文件和库，这里给出一些常用的命令。

读取可执行文件形式的elf文件头信息：

```
[root@localhost test]$ readelf -h main
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:                              exec (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048580
  Start of program headers:          52 (bytes into file)
  Start of section headers:          3232 (bytes into file)
  Flags:                             0x0
  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:         29
  Section header string table index: 26
```

这里，可见可执行文件的elf文件，其类型为EXEC(可执行文件)。另外，含调试信息的"main.debug"和不含调试信息的"main"除了一些大小信息之外，其内容是一样的。并且由此可见文件的体系结构为Intel 80386

读取目标文件形式的elf文件头信息：

```
[root@localhost test]$ readelf -h myfile.o
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:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          516 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         15
  Section header string table index: 12
```

这里，可见目标文件的elf文件，其类型为REL(可重定位文件)。

读取静态库文件形式的elf文件头信息：

```
[root@localhost test]$ readelf -h libmy.a
File: libmy.a(myfile.o)
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:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          516 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         15
  Section header string table index: 12
```

这里，可见静态库文件的elf文件，其类型为REL(可重定位文件)。

读取动态库文件形式的elf文件头信息：

```
[root@localhost test]$ readelf -h libmy.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:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x550
  Start of program headers:          52 (bytes into file)
  Start of section headers:          2768 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         5
  Size of section headers:           40 (bytes)
  Number of section headers:         27
  Section header string table index: 24
```

这里，可见动态库文件的elf文件，其类型为DYN(共享目标文件)。

查看可执行的elf文件程序头表信息：

```
[root@localhost test]$ readelf -l main
Elf file type is EXEC (Executable file)
Entry point 0x8048580
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      Requesting program interpreter: /lib/[ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00970 0x00970 R E 0x1000
  LOAD           0x000970 0x08049970 0x08049970 0x00130 0x001c8 RW  0x1000
  DYNAMIC        0x000988 0x08049988 0x08049988 0x000e0 0x000e0 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00020 0x00020 R   0x4
  GNU_EH_FRAME   0x000820 0x08048820 0x08048820 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
.eh_frame_hdr .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag
   06     .eh_frame_hdr
   07
```

这里，含调试信息的"main.debug"和不含调试信息的"main"其内容是一样的。

查看目标文件的elf文件程序头表信息：

```
[root@localhost test]$ readelf -l myfile.o
There are no program headers in this file.
```

这里可知，可重定位的目标文件，它没程序头表。

查看静态库文件的elf文件程序头表信息：

```
[root@localhost test]$ readelf -l libmy.a
File: libmy.a(myfile.o)
There are no program headers in this file.
```

这里可知，可重定位的静态库文件，它没程序头表。

查看动态库文件的elf文件程序头表信息：

```
[root@localhost test]$ readelf -l libmy.so
Elf file type is DYN (Shared object file)
Entry point 0x550
There are 5 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x00000000 0x00000000 0x007f4 0x007f4 R E 0x1000
  LOAD           0x0007f4 0x000017f4 0x000017f4 0x0011c 0x00128 RW  0x1000
  DYNAMIC        0x000810 0x00001810 0x00001810 0x000e0 0x000e0 RW  0x4
  GNU_EH_FRAME   0x000738 0x00000738 0x00000738 0x0002c 0x0002c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

Section to Segment mapping:
  Segment Sections...
   00     .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn
.rel.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
   01     .ctors .dtors .jcr .data.rel.ro .dynamic .got .got.plt .bss
   02     .dynamic
   03     .eh_frame_hdr
   04
```

这里可知，做为共享目标文件的动态库，它程序头表。

查看一个可执行的elf文件的节信息：

```
[root@localhost test]$ readelf -S main
There are 29 section headers, starting at offset 0xca0:
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0
0  0
  [ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0
0  1
  [ 2] .note.ABI-tag     NOTE            08048148 000148 000020 00   A  0
0  4
  [ 3] .gnu.hash         GNU_HASH        08048168 000168 000030 04   A  4
0  4
  [ 4] .dynsym           DYNSYM          08048198 000198 0000d0 10   A  5
1  4
  [ 5] .dynstr           STRTAB          08048268 000268 000183 00   A  0
0  1
  [ 6] .gnu.version      VERSYM          080483ec 0003ec 00001a 02   A  4
0  2
  [ 7] .gnu.version_r    VERNEED         08048408 000408 000060 00   A  5
2  4
  [ 8] .rel.dyn          REL             08048468 000468 000010 08   A  4
0  4
  [ 9] .rel.plt          REL             08048478 000478 000048 08   A  4
11  4
```

```
  [10] .init             PROGBITS        080484c0 0004c0 000017 00  AX  0
0  4
  [11] .plt              PROGBITS        080484d8 0004d8 0000a0 04  AX  0
0  4
  [12] .text             PROGBITS        08048580 000580 000268 00  AX  0
0 16
  [13] .fini             PROGBITS        080487e8 0007e8 00001c 00  AX  0
0  4
  [14] .rodata           PROGBITS        08048804 000804 00001a 00   A  0
0  4
  [15] .eh_frame_hdr     PROGBITS        08048820 000820 000044 00   A  0
0  4
  [16] .eh_frame         PROGBITS        08048864 000864 00010c 00   A  0
0  4
  [17] .ctors            PROGBITS        08049970 000970 00000c 00  WA  0
0  4
  [18] .dtors            PROGBITS        0804997c 00097c 000008 00  WA  0
0  4
  [19] .jcr              PROGBITS        08049984 000984 000004 00  WA  0
0  4
  [20] .dynamic          DYNAMIC         08049988 000988 0000e0 08  WA  5
0  4
  [21] .got              PROGBITS        08049a68 000a68 000004 04  WA  0
0  4
  [22] .got.plt          PROGBITS        08049a6c 000a6c 000030 04  WA  0
0  4
  [23] .data             PROGBITS        08049a9c 000a9c 000004 00  WA  0
0  4
  [24] .bss              NOBITS          08049aa0 000aa0 000098 00  WA  0
0  8
  [25] .comment          PROGBITS        00000000 000aa0 000114 00      0
0  1
  [26] .shstrtab         STRTAB          00000000 000bb4 0000e9 00      0
0  1
  [27] .symtab           SYMTAB          00000000 001128 000510 10     28
53  4
  [28] .strtab           STRTAB          00000000 001638 0003f4 00      0
0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

这里□main是可执行文件，不含调试信息。

查看一个包含调试信息的可执行的elf文件的节信息：

```
[root@localhost test]$ readelf -S main.debug
There are 37 section headers, starting at offset 0x88c8:

Section Headers:
```

```
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0
0  0
  [ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0
0  1
  [ 2] .note.ABI-tag     NOTE            08048148 000148 000020 00   A  0
0  4
  [ 3] .gnu.hash         GNU_HASH        08048168 000168 000030 04   A  4
0  4
  [ 4] .dynsym           DYNSYM          08048198 000198 0000d0 10   A  5
1  4
  [ 5] .dynstr           STRTAB          08048268 000268 000183 00   A  0
0  1
  [ 6] .gnu.version      VERSYM          080483ec 0003ec 00001a 02   A  4
0  2
  [ 7] .gnu.version_r    VERNEED         08048408 000408 000060 00   A  5
2  4
  [ 8] .rel.dyn          REL             08048468 000468 000010 08   A  4
0  4
  [ 9] .rel.plt          REL             08048478 000478 000048 08   A  4
11  4
  [10] .init             PROGBITS        080484c0 0004c0 000017 00  AX  0
0  4
  [11] .plt              PROGBITS        080484d8 0004d8 0000a0 04  AX  0
0  4
  [12] .text             PROGBITS        08048580 000580 000268 00  AX  0
0 16
  [13] .fini             PROGBITS        080487e8 0007e8 00001c 00  AX  0
0  4
  [14] .rodata           PROGBITS        08048804 000804 00001a 00   A  0
0  4
  [15] .eh_frame_hdr     PROGBITS        08048820 000820 000044 00   A  0
0  4
  [16] .eh_frame         PROGBITS        08048864 000864 00010c 00   A  0
0  4
  [17] .ctors            PROGBITS        08049970 000970 00000c 00  WA  0
0  4
  [18] .dtors            PROGBITS        0804997c 00097c 000008 00  WA  0
0  4
  [19] .jcr              PROGBITS        08049984 000984 000004 00  WA  0
0  4
  [20] .dynamic          DYNAMIC         08049988 000988 0000e0 08  WA  5
0  4
  [21] .got              PROGBITS        08049a68 000a68 000004 04  WA  0
0  4
  [22] .got.plt          PROGBITS        08049a6c 000a6c 000030 04  WA  0
0  4
  [23] .data             PROGBITS        08049a9c 000a9c 000004 00  WA  0
0  4
  [24] .bss              NOBITS          08049aa0 000aa0 000098 00  WA  0
```

```
0  8
  [25] .comment          PROGBITS        00000000 000aa0 000114 00      0
0  1
  [26] .debug_aranges    PROGBITS        00000000 000bb4 000020 00      0
0  1
  [27] .debug_pubnames   PROGBITS        00000000 000bd4 000028 00      0
0  1
  [28] .debug_info       PROGBITS        00000000 000bfc 0067aa 00      0
0  1
  [29] .debug_abbrev     PROGBITS        00000000 0073a6 000726 00      0
0  1
  [30] .debug_line       PROGBITS        00000000 007acc 0003e1 00      0
0  1
  [31] .debug_frame      PROGBITS        00000000 007eb0 00009c 00      0
0  4
  [32] .debug_str        PROGBITS        00000000 007f4c 000735 00      0
0  1
  [33] .debug_loc        PROGBITS        00000000 008681 0000f3 00      0
0  1
  [34] .shstrtab         STRTAB          00000000 008774 000151 00      0
0  1
  [35] .symtab           SYMTAB          00000000 008e90 000590 10     36
61  4
  [36] .strtab           STRTAB          00000000 009420 0003f4 00      0
0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

可见，相对非调试版本的可执行文件，多了".debug*"段的信息。

查看一个目标文件的elf文件的节信息：

```
[root@localhost test]$ readelf -S myfile.o
There are 15 section headers, starting at offset 0x204:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0
0  0
  [ 1] .text             PROGBITS        00000000 000034 00009e 00  AX  0
0  4
  [ 2] .rel.text         REL             00000000 000744 000060 08     13
1  4
  [ 3] .data             PROGBITS        00000000 0000d4 000000 00  WA  0
0  4
  [ 4] .bss              NOBITS          00000000 0000d4 000001 00  WA  0
0  4
  [ 5] .ctors            PROGBITS        00000000 0000d4 000004 00  WA  0
```

```
0  4
  [ 6] .rel.ctors        REL             00000000 0007a4 000008 08     13
5  4
  [ 7] .rodata           PROGBITS        00000000 0000d8 000006 00   A  0
0  1
  [ 8] .eh_frame         PROGBITS        00000000 0000e0 00008c 00   A  0
0  4
  [ 9] .rel.eh_frame     REL             00000000 0007ac 000028 08     13
8  4
  [10] .comment          PROGBITS        00000000 00016c 00002e 00      0
0  1
  [11] .note.GNU-stack   PROGBITS        00000000 00019a 000000 00      0
0  1
  [12] .shstrtab         STRTAB          00000000 00019a 00006a 00      0
0  1
  [13] .symtab           SYMTAB          00000000 00045c 000180 10     14
14  4
  [14] .strtab           STRTAB          00000000 0005dc 000166 00      0
0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

查看一个静态库文件的elf文件的节信息：

```
[root@localhost test]$ readelf -S libmy.a
File: libmy.a(myfile.o)
There are 15 section headers, starting at offset 0x204:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0
0  0
  [ 1] .text             PROGBITS        00000000 000034 00009e 00  AX  0
0  4
  [ 2] .rel.text         REL             00000000 000744 000060 08     13
1  4
  [ 3] .data             PROGBITS        00000000 0000d4 000000 00  WA  0
0  4
  [ 4] .bss              NOBITS          00000000 0000d4 000001 00  WA  0
0  4
  [ 5] .ctors            PROGBITS        00000000 0000d4 000004 00  WA  0
0  4
  [ 6] .rel.ctors        REL             00000000 0007a4 000008 08     13
5  4
  [ 7] .rodata           PROGBITS        00000000 0000d8 000006 00   A  0
0  1
  [ 8] .eh_frame         PROGBITS        00000000 0000e0 00008c 00   A  0
0  4
```

```
  [ 9] .rel.eh_frame     REL             00000000 0007ac 000028 08     13
8  4
  [10] .comment          PROGBITS        00000000 00016c 00002e 00      0
0  1
  [11] .note.GNU-stack   PROGBITS        00000000 00019a 000000 00      0
0  1
  [12] .shstrtab         STRTAB          00000000 00019a 00006a 00      0
0  1
  [13] .symtab           SYMTAB          00000000 00045c 000180 10     14
14  4
  [14] .strtab           STRTAB          00000000 0005dc 000166 00      0
0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

查看一个动态库文件的elf文件的节信息：

```
[root@localhost test]$ readelf -S libmy.so
There are 27 section headers, starting at offset 0xad0:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk
Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0
0  0
  [ 1] .gnu.hash         GNU_HASH        000000d4 0000d4 00003c 04   A  2
0  4
  [ 2] .dynsym           DYNSYM          00000110 000110 000120 10   A  3
1  4
  [ 3] .dynstr           STRTAB          00000230 000230 000199 00   A  0
0  1
  [ 4] .gnu.version      VERSYM          000003ca 0003ca 000024 02   A  2
0  2
  [ 5] .gnu.version_r    VERNEED         000003f0 0003f0 000050 00   A  3
2  4
  [ 6] .rel.dyn          REL             00000440 000440 0000b0 08   A  2
0  4
  [ 7] .rel.plt          REL             000004f0 0004f0 000010 08   A  2
9  4
  [ 8] .init             PROGBITS        00000500 000500 000017 00  AX  0
0  4
  [ 9] .plt              PROGBITS        00000518 000518 000030 04  AX  0
0  4
  [10] .text             PROGBITS        00000550 000550 0001c4 00  AX  0
0 16
  [11] .fini             PROGBITS        00000714 000714 00001c 00  AX  0
0  4
  [12] .rodata           PROGBITS        00000730 000730 000006 00   A  0
0  1
```

```
  [13] .eh_frame_hdr     PROGBITS        00000738 000738 00002c 00   A  0
0  4
  [14] .eh_frame         PROGBITS        00000764 000764 000090 00   A  0
0  4
  [15] .ctors            PROGBITS        000017f4 0007f4 00000c 00  WA  0
0  4
  [16] .dtors            PROGBITS        00001800 000800 000008 00  WA  0
0  4
  [17] .jcr              PROGBITS        00001808 000808 000004 00  WA  0
0  4
  [18] .data.rel.ro      PROGBITS        0000180c 00080c 000004 00  WA  0
0  4
  [19] .dynamic          DYNAMIC         00001810 000810 0000e0 08  WA  3
0  4
  [20] .got              PROGBITS        000018f0 0008f0 00000c 04  WA  0
0  4
  [21] .got.plt          PROGBITS        000018fc 0008fc 000014 04  WA  0
0  4
  [22] .bss              NOBITS          00001910 000910 00000c 00  WA  0
0  4
  [23] .comment          PROGBITS        00000000 000910 0000e6 00      0
0  1
  [24] .shstrtab         STRTAB          00000000 0009f6 0000da 00      0
0  1
  [25] .symtab           SYMTAB          00000000 000f08 000410 10     26
48  4
  [26] .strtab           STRTAB          00000000 001318 000333 00      0
0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```