[Linux ELF文件学习(1)](http://blog.chinaunix.net/uid-21273878-id-1828736.html) *2010-12-01 17:30:16*

分类： LINUX

**ELF头文件学习**

ELF文件原名Executable and Linking Format，译为“可执行可连接格式”。

ELF规范中把ELF文件宽泛的称为“目标文件”，这与我们平时的理解不同。一般的，我们把编译但没有链接的文件(比如Linux下的.o文件)称为目标文件。而ELF文件仅指链接好的可执行文件。在ELF规范中，所用符合ELF规范的文件都成为ELF文件，也成为目标文件，这两个名字意义相同。

经过编译但没有连接的文件则称为“可重定位文件 (relocatable file)”或“待重定位文件 (relocatable file)”。本文采用与ELF规范相同的命名方式，所以当提到可重定位文件时，一般可以理解为惯常所说的目标文件；而提到目标文件时，即指各种类型的 ELF文件。

ELF文件定义如下表格所示：

|  |  |  |
| --- | --- | --- |
| ELF文件 | 可重定位文件(relocatable file) | 可重定位文件(relocatable file)，用于与其它目标文件进行连接以构建可执行文件或动态链接库。可重定位文件就是常说的目标文件，由源文件编译而成，但还没有连接成可执行文件。在 UNIX系统下，一般有扩展名”.o”。之所以称其为“可重定位”，是因为在这些文件中，如果引用到其它目标文件或库文件中定义的符号（变量或者函数）的话，只是给出一个名字，这里还并不知道这个符号在哪里，其具体的地址是什么。需要在连接的过程中，把对这些外部符号的引用重新定位到其真正定义的位置上，所以称目标文件为“可重定位”或者“待重定位”的。 |
| 共享目标文件(shared object file) | 即动态连接库文件。它在以下两种情况下被使用：第一，在连接过程中与其它动态链接库或可重定位文件一起构建新的目标文件；第二，在可执行文件被加载的过程中，被动态链接到新的进程中，成为运行代码的一部分。 |
| 可执行文件(executable file ) | 经过连接的，可以执行的程序文件。 |

ELF文件的作用有两个：一是用于构建程序，构建动态链接库或可执行程序等，主要体现在链接过程。二是用于运行程序。在这两种情况下，我们可以从不同的视角看待同一个目标文件。

从连接的角度和运行的角度，可以分别把目标文件的组成部分做以下划分：



1. ELF头文件

    位于文件最开始处，包含整个文件的结构信息。

2. 节(section)

    是专门用于连接过程而言的，在每个节中包含指令数据、符号数据、重定位数据等等。

3. 程序头表

    在运行过程中是必须的，在链接过程中是可选的，因为它的作用是告诉系统如何创建进程的映像。

4. 节头表

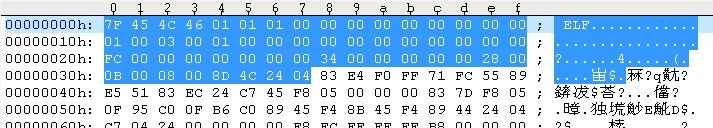
    包含文件中所用节的信息。

下面看一下Linux内核对ELF头文件的定义。

[linux+v2.6.36/include/linux/elf.h](http://lxr.linux.no/linux+v2.6.36/include/linux/elf.h):

|  |
| --- |
| /\* 32-bit ELF base types. \*/ typedef \_\_u32 Elf32\_Addr; typedef \_\_u16 Elf32\_Half; typedef \_\_u32 Elf32\_Off; typedef \_\_s32 Elf32\_Sword; typedef \_\_u32 Elf32\_Word;    #define EI\_NIDENT 16   typedef struct elf32\_hdr{   unsigned char e\_ident[EI\_NIDENT]; //16字节的信息，下文详细解释   Elf32\_Half e\_type;  //目标文件类型    Elf32\_Half e\_machine;  //体系结构类型   Elf32\_Word e\_version;  //目标文件版本   Elf32\_Addr e\_entry; /\* Entry point 程序入口的虚拟地址\*/   Elf32\_Off e\_phoff;  //程序头部表的偏移量   Elf32\_Off e\_shoff;  //节区头部表的偏移量   Elf32\_Word e\_flags;  //   Elf32\_Half e\_ehsize;  //ELF头部的大小   Elf32\_Half e\_phentsize;  //程序头部表的表项大小   Elf32\_Half e\_phnum;  //程序头部表的数目   Elf32\_Half e\_shentsize; //节区头部表的表项大小   Elf32\_Half e\_shnum; //节区头部表的数目   Elf32\_Half e\_shstrndx; //  } Elf32\_Ehdr;  //此结构体一共52个字节 |

我使用Ultraedit打开一个ELF文件，来分析它的前52个字节：



|  |  |  |
| --- | --- | --- |
| e\_ident[0] | 7F | 目标文件最开头的前四个字节。  文件标志，表示是ELF文件  EI\_MAG0 = 0, EI\_MAG1 = 1, EI\_MAG2 = 2, EI\_MAG3 = 3 |
| e\_ident[1] | 45 == ‘E’ |
| e\_ident[2] | 4C == ’L’ |
| e\_ident[3] | 46 == ‘F’ |
| e\_ident[4] | 01 | 文件类别,取值如下：(EI\_CLASS = 4)   |  |  |  | | --- | --- | --- | | ELFCLASSNONE | 0 | 非法目标文件 | | ELFCLASS32 | 1 | 32为目标文件 | | ELFCLASS64 | 2 | 64为目标文件 | |
| e\_ident[5] | 01 | 编码格式,取值如下：(EI\_DATA = 5)   |  |  |  | | --- | --- | --- | | ELFDATANONE | 0 | 非法编码格式 | | ELFDATA2LSB | 1 | 小端编码格式 | | ELFDATA2MSB | 2 | 大端编码格式 | |
| e\_ident[6] | 01 | 文件版本，(EI\_VERSION)，为1表明是目前版本 |
| e\_ident[7]  to  e\_ident[15] | 00 00 00 00 00 00 00 00 00 | 这就个字节暂且不使用，留在以后扩展。 |
| e\_type | 00 01 | 此字段表明文件属于那种类型：   |  |  |  | | --- | --- | --- | | ET\_NONE | 0 | 未知文件类型 | | ET\_REL | 1 | 可重定位文件 | | ET\_EXEC | 2 | 可执行文件 | | ET\_DYN | 3 | 动态链接库文件 | | ET\_CORE | 4 | CORE文件 | | ET\_LOPROC | 0xff00 | 特定处理器文件扩展下边界 | | ET\_HIPROC | 0xffff | 特定处理器文件扩展上边界 | |
| e\_machine | 00 03 | 体系结构类型 : Intel 80386 |
| e\_version | 00 00 00 01 | 目前版本 |
| e\_entry | 00 00 00 00 |  |
| e\_phoff | 00 00 00 00 | 没有程序头部表。 |
| e\_shoff | 00 00 00 FC | e\_ident[0]是文件的第一个字节，从它开始FC(252)字节后就是节的数据信息。 |
| e\_flags | 00 00 00 00 |  |
| e\_ehsize | 00 34 | 52字节的ELF头部结构体 |
| e\_phentsize | 00 00 | 因为没有程序头部表，所以大小为0 |
| e\_phnum | 00 00 | 同上 |
| e\_shentsize | 00 28 | 40字节的节区头部表 |
| e\_shnum | 00 0B | 节区头部表的表项有B(11)个 |
| e\_shstrndx | 00 08 | 节头表中与节名字表相对应的表项的索引。 |

 下面的一段程序用来读ELF文件的头部：readelf\_h.c:

|  |
| --- |
| #include <stdio.h> #include <stdlib.h> #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> #include <unistd.h> #include <errno.h>  #define FALSE 0 #define TURE 1 #define MAX\_SIZE 52  typedef short int     Elf32\_Half; typedef int         Elf32\_Word; typedef Elf32\_Word     Elf32\_Addr; typedef Elf32\_Word     Elf32\_Off; /\*Elf头部文件部分重要数据\*/ typedef struct{         Elf32\_Half e\_type;         Elf32\_Half e\_machine;         Elf32\_Word e\_version;         Elf32\_Addr e\_entry; //程序入口的虚拟地址，如果目标文件没有程序入口，为0           Elf32\_Off e\_phoff; //程序头部表格的偏移量（按字节），如果文件中没有，为0          Elf32\_Off e\_shoff; //节区头部表格的偏移量（按字节），如果文件中没有，为0              Elf32\_Word e\_flags; //          Elf32\_Half e\_ehsize; //ELF头部的大小          Elf32\_Half e\_phentsize; //程序头部表格的表项大小。          Elf32\_Half e\_phnum; //程序头部表格的表项数目。          Elf32\_Half e\_shentsize; //节区头部表格的表项大小。          Elf32\_Half e\_shnum; //节区头部表格的表项数目。          Elf32\_Half e\_shstrndx; }Elf\_lan; static Elf\_lan lan\_elf; int OpenElf(char \*filename) {     int fd;     fd = open(filename, O\_RDONLY);     if(fd == -1){           printf("Open %s Error!\n", filename);           return FALSE;      }     return fd;  } //读取Elf头部表函数 :int ReadElf(int fd);  int ReadElf(int fd) {     char str[MAX\_SIZE];     int num;     memset(str, 0, MAX\_SIZE);     if((num = read(fd, str, 52)) != 52){             perror("File NO ELF!\n");             return FALSE;      }     if((str[0] == 0x7f) && (str[1] == 'E') && (str[2] == 'L') && (str[3] == 'F')){           printf("This is ELF file.\n");           printf("文件类别： ");            switch(str[4]){                         case 0:                             printf("非法目标文件\n");                             break;                         case 1:                             printf("32位目标文件\n");                             break;                         case 2:                             printf("64位目标文件\n");                             break;                         default:                             break;                        }           printf("编码格式: ");           switch(str[5]){                   case 0:                       printf("非法编码格式\n");                       break;                   case 1:                       printf("小端编码格式\n");                       break;                   case 2:                       printf("大端编码格式\n");                           break;                         default:                             break;               }           printf("文件版本： ");           if(str[6] == 1){                   printf("当前版本\n");           }else{                   printf("NULL\n");           }           printf("目标文件类型： ");           lan\_elf.e\_type = \*((Elf32\_Half \*)&str[16]);           printf("e\_type = %d\t", lan\_elf.e\_type);            switch(lan\_elf.e\_type){                   case 0:                       printf("未知文件类型\n");                       break;                   case 1:                       printf("可重定位文件类型\n");                       break;                   case 2:                       printf("可执行文件\n");                       break;                   case 3:                       printf("动态链接库文件\n");                       break;                   case 4:                       printf("CORE文件\n");                       break;                   default:                       break;                                   }           printf("体系结构为：");           lan\_elf.e\_machine = \*((Elf32\_Half \*)&str[18]);           printf("e\_machine = %d\n", lan\_elf.e\_machine);           switch(lan\_elf.e\_machine){                  case 0:                       printf("未知体系结构");                       break;                  case 3:                       printf("Intel 8086");            }           printf("版本信息： ");           lan\_elf.e\_version = \*((Elf32\_Word \*)&str[20]);           if(lan\_elf.e\_version == 1){                   printf("当前版本\n");           }else{                   printf("NULL\n");           }           printf("程序入口的虚拟地址：");           lan\_elf.e\_entry = \*((Elf32\_Word \*)&str[24]);           printf("0x%x\n", lan\_elf.e\_entry);                      printf("程序头部表格的偏移量（按字节）： ");           lan\_elf.e\_phoff = \*((Elf32\_Off \*)&str[28]);           printf("0x%x, %d\n", lan\_elf.e\_phoff, lan\_elf.e\_phoff);                      printf("节区头部表格的偏移量（按字节）： ");           lan\_elf.e\_shoff = \*((Elf32\_Off \*)&str[32]);           printf("0x%x, %d\n", lan\_elf.e\_shoff, lan\_elf.e\_shoff);                      printf("处理器标志位： ");           lan\_elf.e\_flags = \*((Elf32\_Off \*)&str[36]);           printf("%d\n", lan\_elf.e\_flags);                      printf("ELF头文件大小： ");           lan\_elf.e\_ehsize = \*((Elf32\_Half \*)&str[40]);           printf("0x%x, %d\n", lan\_elf.e\_ehsize, lan\_elf.e\_ehsize);                      printf("程序头部表大小： ");           lan\_elf.e\_phentsize = \*((Elf32\_Half \*)&str[42]);           printf("0x%x, %d\n", lan\_elf.e\_phentsize, lan\_elf.e\_phentsize);                      printf("程序头部表的数目：");           lan\_elf.e\_phnum = \*((Elf32\_Half \*)&str[44]);           printf("0x%x, %d\n", lan\_elf.e\_phnum, lan\_elf.e\_phnum);                      printf("节区头部表大小： ");           lan\_elf.e\_shentsize = \*((Elf32\_Half \*)&str[46]);           printf("0x%x, %d\n", lan\_elf.e\_shentsize, lan\_elf.e\_shentsize);                      printf("节区头部表数目： ");           lan\_elf.e\_shnum = \*((Elf32\_Half \*)&str[48]);           printf("0x%x, %d\n", lan\_elf.e\_shnum, lan\_elf.e\_shnum);                      printf("节头表与节名字相对应的表项的索引： ");           lan\_elf.e\_shstrndx = \*((Elf32\_Half \*)&str[50]);           printf("0x%x, %d\n", lan\_elf.e\_shstrndx, lan\_elf.e\_shstrndx);           return TURE;      }else{           perror("File NO ELF!\n");           return FALSE;      }  }  int main(int argc, char \*argv[]) {     int boolen;     if(argc == 2){             boolen = OpenElf(argv[1]);             if(boolen == FALSE){                       return -1;              }             ReadElf(boolen);      }      return 0;  } |

$./readelf\_lan readelf\_lan.o  
This is ELF file.  
文件类别： 32位目标文件  
编码格式:  小端编码格式  
文件版本：  当前版本  
目标文件类型： e\_type = 1       可重定位文件类型  
体系结构为：e\_machine = 3  
Intel 8086版本信息： 当前版本  
程序入口的虚拟地址：0x0  
程序头部表格的偏移量（按字节）： 0x0, 0  
节区头部表格的偏移量（按字节）： 0xc00, 3072  
处理器标志位： 0  
ELF头文件大小： 0x34, 52  
程序头部表大小： 0x0, 0  
程序头部表的数目：0x0, 0  
节区头部表大小： 0x28, 40  
节区头部表数目： 0xc, 12  
节头表与节名字相对应的表项的索引： 0x9, 9