作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师

# ELF 的含义和组成

ELF 的全称为 Executable and Linkable Format，可执行可链接的文件格式。
ELF 是通用格式，支持可执行程序文件、静态库文件、动态库文件、目标对象文件等。

ELF 的核心概念：
符号 symbol
symbol 是基本元素。全局变量、方法等都会转为符号。还有许多附加的符号，比如_init、_start、_end。
symbol 的属性主要包括，名称、类型、大小、 所属 section、位置。
symbol 的类型，包括 OBJECT、FUNC、SECTION、FILE 等。OBJECT 表示对象、全局变量，FUNC 表示函数，SECTION 表示段。

段 section
section 是分组。把同一类型的 symbol 组织在一起。
section 的属性主要包括，名称、类型、权限、地址、大小。
section 的类型，包括 PROGBITS、RELA、SYMTAB、STRTAB 等。PROGBITS 表示程序二进制数据，包括代码段、数据段、GOT 段、PLT 段等。RELA 表示重定位段。SYMTAB 表示符号表。STRTAB 表示字符串表。

段 segment
segment 是分组。把类型相同、权限相同的多个 section，合并为一个 segment。目的是页对齐，提高内存使用率，减少内存浪费。
segment 的属性主要包括，类型、权限、地址、大小、对齐。

重定位 relocation
程序来源于多个文件。每个文件可以独立编译为目标文件，其中的符号可能是文件内的相对地址。程序的源码，可能引用外部目标文件的符号。程序的编译和运行，依赖进程的虚拟地址空间。重定位把引用的符号的地址修正为进程的虚拟地址空间的地址。
relocation 的属性主要包括偏移、类型、符号名、补充值。
relocation 的类型包括 R_X86_64_32、R_X86_64_PC32 等。R_X86_64_32 表示绝对重定位，地址用 32 位表示。R_X86_64_PC32 表示相对重定位，偏移值用 32 位表示。

# ELF 在 linux 源码的定义

ELF 定义，和具体的 CPU、编译器、OS 版本有关，这里展示部分文件。

```
linux-5.6.3/include/linux/elf.h
linux-5.6.3/include/uapi/linux/elf.h
linux-5.6.3/include/uapi/linux/elf-em.h
linux-5.6.3/arch/x86/include/asm/elf.h
```

基本类型：

```
/* 64-bit ELF base types. */
typedef __u64 Elf64_Addr;
```

```
typedef __u16 Elf64_Half;
typedef __s16 Elf64_SHalf;
typedef __u64 Elf64_Off;
typedef __s32 Elf64_Sword;
typedef __u32 Elf64_Word;
typedef __u64 Elf64_Xword;
typedef __s64 Elf64_Sxword;
```

文件类型：

```
/* These constants define the different elf file types */
#define ET_NONE   0
#define ET_REL    1
#define ET_EXEC   2
#define ET_DYN    3
#define ET_CORE   4
#define ET_LOPROC 0xff00
#define ET_HIPROC 0xffff
```

section 段的类型：

```
/* sh_type */
#define SHT_NULL    0
#define SHT_PROGBITS    1
#define SHT_SYMTAB  2
#define SHT_STRTAB  3
#define SHT_RELA    4
#define SHT_HASH    5
#define SHT_DYNAMIC 6
#define SHT_NOTE    7
#define SHT_NOBITS  8
#define SHT_REL     9
#define SHT_SHLIB   10
#define SHT_DYNSYM  11
#define SHT_NUM     12
#define SHT_LOPROC  0x70000000
#define SHT_HIPROC  0x7fffffff
#define SHT_LOUSER  0x80000000
#define SHT_HIUSER  0xffffffff
```

文件头：

```
typedef struct elf64_hdr {
  unsigned char e_ident[EI_NIDENT]; /* ELF "magic number" */
  Elf64_Half e_type;
  Elf64_Half e_machine;
  Elf64_Word e_version;
  Elf64_Addr e_entry;       /* Entry point virtual address */
  Elf64_Off e_phoff;        /* Program header table file offset */
  Elf64_Off e_shoff;        /* Section header table file offset */
  Elf64_Word e_flags;
  Elf64_Half e_ehsize;
```

```
  Elf64_Half e_phentsize;
  Elf64_Half e_phnum;
  Elf64_Half e_shentsize;
  Elf64_Half e_shnum;
  Elf64_Half e_shstrndx;
} Elf64_Ehdr;
```

section 段头：

```
typedef struct elf64_shdr {
  Elf64_Word sh_name;       /* Section name, index in string tbl */
  Elf64_Word sh_type;       /* Type of section */
  Elf64_Xword sh_flags;     /* Miscellaneous section attributes */
  Elf64_Addr sh_addr;       /* Section virtual addr at execution */
  Elf64_Off sh_offset;      /* Section file offset */
  Elf64_Xword sh_size;      /* Size of section in bytes */
  Elf64_Word sh_link;       /* Index of another section */
  Elf64_Word sh_info;       /* Additional section information */
  Elf64_Xword sh_addralign; /* Section alignment */
  Elf64_Xword sh_entsize;   /* Entry size if section holds table */
} Elf64_Shdr;
```

program 程序头：

```
typedef struct elf64_phdr {
  Elf64_Word p_type;
  Elf64_Word p_flags;
  Elf64_Off p_offset;       /* Segment file offset */
  Elf64_Addr p_vaddr;       /* Segment virtual address */
  Elf64_Addr p_paddr;       /* Segment physical address */
  Elf64_Xword p_filesz;     /* Segment size in file */
  Elf64_Xword p_memsz;      /* Segment size in memory */
  Elf64_Xword p_align;      /* Segment alignment, file & memory */
} Elf64_Phdr;
```

symbol 符号：

```
typedef struct elf64_sym {
  Elf64_Word st_name;       /* Symbol name, index in string tbl */
  unsigned char st_info;    /* Type and binding attributes */
  unsigned char st_other;   /* No defined meaning, 0 */
  Elf64_Half st_shndx;      /* Associated section index */
  Elf64_Addr st_value;      /* Value of the symbol */
  Elf64_Xword st_size;      /* Associated symbol size */
} Elf64_Sym;
```

重定位：

```
typedef struct elf64_rel {
  Elf64_Addr r_offset;  /* Location at which to apply the action */
  Elf64_Xword r_info;   /* index and type of relocation */
} Elf64_Rel;
```

重定位：

```
typedef struct elf64_rela {
  Elf64_Addr r_offset;  /* Location at which to apply the action */
  Elf64_Xword r_info;   /* index and type of relocation */
  Elf64_Sxword r_addend;    /* Constant addend used to compute value */
} Elf64_Rela;
```

## 用 C 和汇编分析 ELF 格式

编写多种类型的变量、方法，查看符号地址、汇编指令、ELF 信息。

编写代码： cat.h

```
#ifndef _CAT_H_
#define _CAT_H_

// 变量
extern char cat_name[16];

// 变量
extern int cat_speed;

// 函数
extern void cat_run(int plus);

#endif
```

编写代码： cat.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

char cat_name[16] = {'T', 'o', 'm', '\0'};

int cat_speed = 0x71727374;

void cat_run(int plus)
{
    printf(" Cat %s is running at speed %#X \n", cat_name, cat_speed);
}
```

编写代码： main.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "cat.h"
```

```
int main_int = 0x61626364;

void main_print_param(char *name, void *addr, int value)
{
    printf(" %15s  addr = %p  value = %#X \n", name, addr, value);
}

void main_print_func(char *name, void *addr)
{
    printf(" %15s  addr = %p   \n", name, addr);
}

int main()
{
    // 重置变量。
    main_int = 0xF1F2F3F4;

    // 调用函数。
    cat_run(3);

    // 查看变量的地址、值。
    printf("\nparam addr and value : \n");
    main_print_param("cat_speed", &cat_speed, cat_speed);
    main_print_param("main_int", &main_int, main_int);

    // 查看方法的地址。
    printf("\nfunc addr :  \n");
    main_print_func("cat_run", cat_run);
    main_print_func("main", main);

    // 休眠。方便查看内存布局。
    sleep(90000);
    return 0;
}
```

编译代码：

```
gcc cat.c -c -o cat.o
gcc main.c cat.o -o main

readelf -a cat.o > cat.o.elf.txt
readelf -a main > main.elf.txt

objdump -D cat.o > cat.o.dump.txt
objdump -D main > main.dump.txt
```

生成 elf 文件：

| 分类 | 源文件 | 结果文件 | ELF 文件 | ELF 文件类型 |
|---|---|---|---|---|
| 普通目标文件 | cat.c | cat.o | cat.o.elf.txt | Type:  REL (Relocatable file) |
| main 程序 | main.c | main | main.elf.txt | Type:  EXEC (Executable file) |

运行代码：

```
[root@local base]# ./main
 Cat Tom is running at speed 0X71727374

param addr and value :
       cat_speed  addr = 0x601060  value = 0X71727374
        main_int  addr = 0x601044  value = 0XF1F2F3F4

func addr :
         cat_run  addr = 0x4006a4
            main  addr = 0x40061b
```

查看进程的内存布局：

```
[root@local base]# ps aux | grep ./main
root      75157  0.0  0.0   4216   352 pts/3    S+   18:41   0:00 ./main
root      75258  0.0  0.0 112812   992 pts/4    S+   18:41   0:00 grep --color=auto ./main
[root@local base]# cat /proc/75157/maps
00400000-00401000               r-xp               00000000               08:03               707619
/root/code/x86-asm/common2/elf2/base/main
00600000-00601000               r--p               00000000               08:03               707619
/root/code/x86-asm/common2/elf2/base/main
00601000-00602000               rw-p               00001000               08:03               707619
/root/code/x86-asm/common2/elf2/base/main
7ff0b0ec6000-7ff0b108a000 r-xp 00000000 08:03 15928                      /usr/lib64/libc-2.17.so
7ff0b108a000-7ff0b1289000 ---p 001c4000 08:03 15928                      /usr/lib64/libc-2.17.so
7ff0b1289000-7ff0b128d000 r--p 001c3000 08:03 15928                      /usr/lib64/libc-2.17.so
7ff0b128d000-7ff0b128f000 rw-p 001c7000 08:03 15928                      /usr/lib64/libc-2.17.so
7ff0b128f000-7ff0b1294000 rw-p 00000000 00:00 0
7ff0b1294000-7ff0b12b6000 r-xp 00000000 08:03 611075                     /usr/lib64/ld-2.17.so
7ff0b14aa000-7ff0b14ad000 rw-p 00000000 00:00 0
7ff0b14b3000-7ff0b14b5000 rw-p 00000000 00:00 0
7ff0b14b5000-7ff0b14b6000 r--p 00021000 08:03 611075                     /usr/lib64/ld-2.17.so
7ff0b14b6000-7ff0b14b7000 rw-p 00022000 08:03 611075                     /usr/lib64/ld-2.17.so
7ff0b14b7000-7ff0b14b8000 rw-p 00000000 00:00 0
7fff12ed4000-7fff12ef5000 rw-p 00000000 00:00 0                          [stack]
7fff12f0f000-7fff12f11000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

文件合并：

把多个文件的内容重组，整合在一起，生成 1 个文件。原始文件为 cat.h、cat.c、main.c，最终文件为 main 可执行文件。符号 cat_speed、main_int、cat_run、main 等，原本分散在 cat.c、main.c，最后都在 main 文件。文件合并之后，符号的绝对地址、相对地址发生变化，需要修正符号的地址。

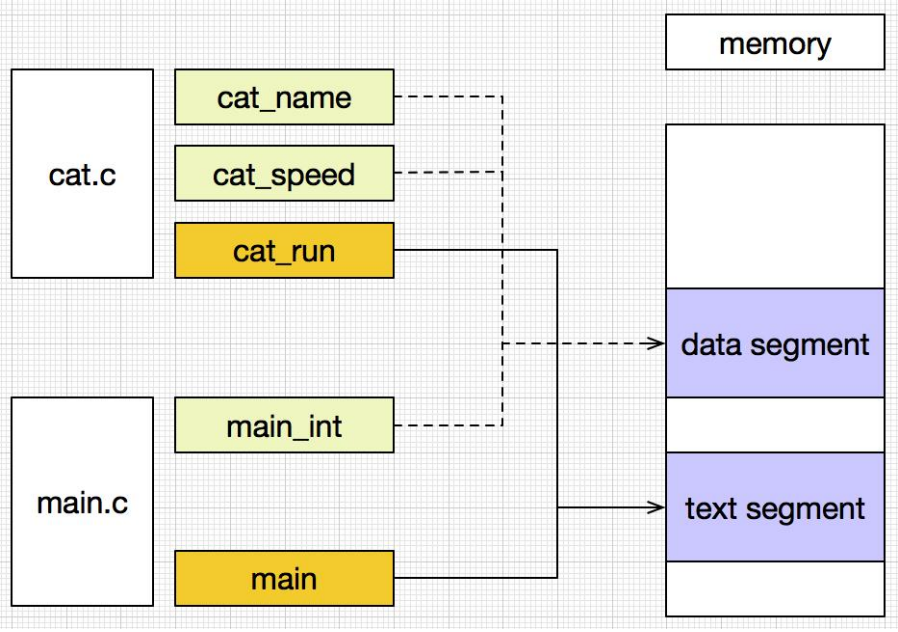
查看符号的内存布局：

符号 cat_speed、main_int，地址前缀为 0x6010，在数据区 `00601000-00602000 rw-p` 。

符号 cat_run、main，地址前缀为 0x4006，在代码区 `00400000-00401000 r-xp` 。

查看 ELF 文件，符号的地址的变化。

| 分类 | 自身的 ELF 文件 | main 的 ELF 文件 |
| --- | --- | --- |
| 普通目标文件 | 0000000000000000 cat_name<br>0000000000000010 cat_speed<br>0000000000000000 cat_run | 0000000000601050 cat_name<br>0000000000601060 cat_speed<br>00000000004006a4 cat_run |
| main 程序 | 这里没有给 main 生成单独的 ELF 文件 | 000000000040061b main<br>0000000000601044 main_int |




符号重定位：
修正符号的绝对地址、相对地址，保证符号的引用关系。
函数 cat_run，引用变量 cat_name、cat_speed 。分析符号之间的关系。

| 文件 | 代码 |
| --- | --- |
| cat.c<br>源码 | char cat_name[16] = {'T', 'o', 'm', '\0'};<br><br>int cat_speed = 0x71727374;<br><br>void cat_run(int plus)<br>{<br>    printf(" Cat %s is running at speed %#X \n", cat_name, cat_speed);<br>} |
| cat.o.elf.txt<br>目标文件的 ELF 信息 | Relocation section '.rela.text' at offset 0x278 contains 4 entries:<br>  Offset  Info  Type  Sym. Value  Sym. Name + Addend<br>00000000000d  000a00000002 R_X86_64_PC32  0000000000000010 cat_speed - 4<br>000000000014  00090000000a R_X86_64_32  0000000000000000 cat_name + 0<br>000000000019  00050000000a R_X86_64_32  0000000000000000 .rodata + 0<br>000000000023  000c00000002 R_X86_64_PC32  0000000000000000 printf - 4 |
| main.elf.txt<br>main 程序的 ELF 信息 | Symbol table '.symtab' contains 72 entries:<br>  Num:  Value  Size Type  Bind  Vis  Ndx Name<br>  58: 0000000000601050  16 OBJECT  GLOBAL DEFAULT  24 cat_name |

| | |
|---|---|
| | 63: 0000000000601060 4 OBJECT GLOBAL DEFAULT 24 cat_speed<br>66: 00000000004006a4 41 FUNC GLOBAL DEFAULT 13 cat_run |
| cat.o.dump.txt<br>目标文件的 dump 信息 | 0000000000000000 <cat_run>:<br>0: 55 push %rbp<br>1: 48 89 e5 mov %rsp,%rbp<br>4: 48 83 ec 10 sub $0x10,%rsp<br>8: 89 7d fc mov %edi,-0x4(%rbp)<br>b: 8b 05 00 00 00 00 mov 0x0(%rip),%eax # 11 <cat_run+0x11><br>11: 89 c2 mov %eax,%edx<br>13: be 00 00 00 00 mov $0x0,%esi<br>18: bf 00 00 00 00 mov $0x0,%edi<br>1d: b8 00 00 00 00 mov $0x0,%eax<br>22: e8 00 00 00 00 callq 27 <cat_run+0x27><br>27: c9 leaveq<br>28: c3 retq |
| main.dump.txt<br>main 程序的 dump 信息 | 00000000004006a4 <cat_run>:<br>4006a4: 55 push %rbp<br>4006a5: 48 89 e5 mov %rsp,%rbp<br>4006a8: 48 83 ec 10 sub $0x10,%rsp<br>4006ac: 89 7d fc mov %edi,-0x4(%rbp)<br>4006af: 8b 05 ab 09 20 00 mov 0x2009ab(%rip),%eax # 601060 <cat_speed><br>4006b5: 89 c2 mov %eax,%edx<br>4006b7: be 50 10 60 00 mov $0x601050,%esi<br>4006bc: bf e0 07 40 00 mov $0x4007e0,%edi<br>4006c1: b8 00 00 00 00 mov $0x0,%eax<br>4006c6: e8 c5 fd ff ff callq 400490 <printf@plt><br>4006cb: c9 leaveq<br>4006cc: c3 retq<br>4006cd: 0f 1f 00 nopl (%rax) |

R_X86_64_32 表示绝对重定位。计算公式为 `abs_addr = symbol_addr + addend`，addend 一般是 0。
cat.o.dump.txt 中，`13: be 00 00 00 00 mov $0x0,%esi` 表示 printf 的第 2 个参数 cat_name，传入 cat_name 的内存地址，先用 0 占位。
main.dump.txt 中，`4006b7: be 50 10 60 00 mov $0x601050,%esi` 表示 `00 00 00 00` 被替换为 `50 10 60 00`，小端表示为 0x601050。
main.elf.txt 中，cat_name 的内存地址为 0000000000601050，和上方的 0x601050 一致。

R_X86_64_PC32 表示相对重定位。计算公式为 `offset = symbol_addr + addend - offset_addr = symbol_addr - PC`，PC 是程序计数器，使用寄存器 rip 保存下一个指令的地址。offset 占用 4 个字节，从 offset 的开头地址到下一个指令的地址间隔 4 个字节,所以 addend 一般是-4。offset_addr 是 offset 的开头位置,`offset_addr - addend = PC`。
cat.o.dump.txt 中，`b: 8b 05 00 00 00 00 mov 0x0(%rip),%eax` 表示用 rip 相对寻址，把 cat_speed 的值写入寄存器 eax。先用 0 占位。
main.dump.txt 中，`4006af: 8b 05 ab 09 20 00 mov 0x2009ab(%rip),%eax` 表示把 `00 00 00 00` 替换为 `ab 09 20 00`，小端表示为 0x2009ab。

main.elf.txt 中，cat_speed 的内存地址为 0000000000601060。
下一个指令为 `4006b5:    89 c2    mov  %eax,%edx` ，PC 即下一个指令的地址 = 4006b5。
计算 `offset = 0000000000601060 - 4006b5 = 0x2009ab` ，和上方的 0x2009ab 一致。