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

反汇编的含义

反汇编是逆向工程，把二进制程序文件反汇编为汇编文件。
工具 objdump，可以查看段、符号地址和值、函数定义和调用、汇编指令等。
objdump 的常用参数包括
-D　反汇编 section 的具体内容。
-h　反汇编 section 的头部列表。
-t　输出符号表。
-r　输出重定位表。
-R　输出动态重定位表。

用 C 和汇编分析反汇编

编写代码： dump.c

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

// 变量
int32_t bird_color = 0x51525354;
int64_t bird_speed = 0x7172737475767778;

// 结构体
struct cat
{
    int32_t color;
    int64_t speed;
};
struct cat tom = {
    .color = 0xC1C2C3C4,
    .speed = 0xD1D2D3D4D5D6D7D8};

// 函数
int32_t func_num(int32_t n)
{
    int32_t tmp = (n << 8) + 7;
    return tmp;
}

// 函数。打印地址。
void func_print_addr(char *desc, void *addr)
```

```
{
    printf(" %12s  addr = %p \n", desc, addr);
}

int main()
{
    // 变量地址
    func_print_addr("bird_color", &bird_color);
    func_print_addr("bird_speed", &bird_speed);
    func_print_addr("tom", &tom);
    func_print_addr("tom.color", &(tom.color));
    func_print_addr("tom.speed", &(tom.speed));

    // 函数地址
    func_print_addr("func_num", &(func_num));

    // 调用函数
    int32_t before = 0xA1A2A3A4;
    int32_t after = func_num(before);
    printf(" func_num  before = %#X  after = %#X \n", before, after);
    return 0;
}
```

编译代码：

```
gcc dump.c -o dump
objdump -D dump > dump.asm.all.txt
```

运行代码：

```
[root@local dump]# ./dump
   bird_color  addr = 0x601040
   bird_speed  addr = 0x601048
          tom  addr = 0x601050
    tom.color  addr = 0x601050
    tom.speed  addr = 0x601058
     func_num  addr = 0x40052d
 func_num  before = 0XA1A2A3A4  after = 0XA2A3A407
```

分析结果：

首先用 dump.c 构建可执行文件 dump，然后用 dump 构建反汇编文件 dump.asm.all.txt。

变量 bird_color 的地址为 0x601040，值为 0x51525354，反汇编的代码为

```
0000000000601040 <bird_color>:
  601040:   54                      push   %rsp
  601041:   53                      push   %rbx
  601042:   52                      push   %rdx
  601043:   51                      push   %rcx
```

汇编代码中，符号地址为 0000000000601040，连续的 4 个字节为 54、53、52、51，表示小端整数 0x51525354。

变量 bird_speed 的地址为 0x601048，值为 0x7172737475767778，反汇编的代码为

```
0000000000601048 <bird_speed>:
  601048:   78 77                   js     6010c1 <_end+0x59>
  60104a:   76 75                   jbe    6010c1 <_end+0x59>
  60104c:   74 73                   je     6010c1 <_end+0x59>
  60104e:   72 71                   jb     6010c1 <_end+0x59>
```

汇编代码中，符号地址为 0000000000601048，连续的 8 个字节为 78、77、76、75、74、73、72、71，表示小端整数 0x7172737475767778。

变量 tom 的地址为 0x601050，属性 color 的值为 0xC1C2C3C4，属性 speed 的值为 0xD1D2D3D4D5D6D7D8，反汇编的代码为

```
0000000000601050 <tom>:
  601050:   c4 c3 c2 c1             (bad)
  601054:   00 00                   add    %al,(%rax)
  601056:   00 00                   add    %al,(%rax)
  601058:   d8 d7                   fcom   %st(7)
  60105a:   d6                      (bad)
  60105b:   d5                      (bad)
  60105c:   d4                      (bad)
  60105d:   d3 d2                   rcl    %cl,%edx
  60105f:   d1                      .byte 0xd1
```

汇编代码中，符号地址为 0000000000601050。地址 601050 有 4 个字节表示 0xC1C2C3C4，地址 601054 有 4 个字节对齐填充，地址 601058 有 8 个字节表示 0xD1D2D3D4D5D6D7D8。

函数 func_num 的地址为 0x40052d，反汇编的代码为

```
000000000040052d <func_num>:
  40052d:   55                      push   %rbp
  40052e:   48 89 e5                mov    %rsp,%rbp
  400531:   89 7d ec                mov    %edi,-0x14(%rbp)
  400534:   8b 45 ec                mov    -0x14(%rbp),%eax
  400537:   c1 e0 08                shl    $0x8,%eax
  40053a:   83 c0 07                add    $0x7,%eax
  40053d:   89 45 fc                mov    %eax,-0x4(%rbp)
  400540:   8b 45 fc                mov    -0x4(%rbp),%eax
  400543:   5d                      pop    %rbp
  400544:   c3                      retq
```

汇编代码中，符号地址为 000000000040052d。 `int32_t tmp = (n << 8) + 7;` 对应汇编代码 `shl $0x8,%eax` 、 `add $0x7,%eax` 。

调用普通函数 func_num， `int32_t after = func_num(before);` 对应汇编代码

```
  4005d3:   c7 45 fc a4 a3 a2 a1    movl   $0xa1a2a3a4,-0x4(%rbp)
  4005da:   8b 45 fc                mov    -0x4(%rbp),%eax
  4005dd:   89 c7                   mov    %eax,%edi
  4005df:   e8 49 ff ff ff          callq  40052d <func_num>
```

第一行包含字节 `a4 a3 a2 a1` ，表示小端整数 0xa1a2a3a4。

调用动态库函数 printf， `printf(" %12s  addr = %p \n", desc, addr);` 对应汇编代码

```
  40055d:   48 89 c6                mov    %rax,%rsi
  400560:   bf a0 06 40 00          mov    $0x4006a0,%edi
  400565:   b8 00 00 00 00          mov    $0x0,%eax
```

```
  40056a:   e8 a1 fe ff ff          callq  400410 <printf@plt>
```
符号 printf@plt 的地址为 0000000000400410，在 PLT 段。
```
Disassembly of section .plt:

0000000000400400 <.plt>:
  400400:   ff 35 02 0c 20 00       pushq       0x200c02(%rip)                    #    601008
<_GLOBAL_OFFSET_TABLE_+0x8>
  400406:   ff 25 04 0c 20 00       jmpq          *0x200c04(%rip)                 #    601010
<_GLOBAL_OFFSET_TABLE_+0x10>
  40040c:   0f 1f 40 00             nopl   0x0(%rax)

0000000000400410 <printf@plt>:
  400410:   ff 25 02 0c 20 00       jmpq   *0x200c02(%rip)        # 601018 <printf@GLIBC_2.2.5>
  400416:   68 00 00 00 00          pushq  $0x0
  40041b:   e9 e0 ff ff ff          jmpq   400400 <.plt>
```