作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师
https://github.com/drink-cat/Book_Program-Principles

指针的本质

指针的本质是内存地址。
64 位操作系统，内存地址表示为 64 位无符号整数 uint64_t。
64 位指针占用 8 个字节，汇编表示为.quad。

指针读写内存，本质为用内存地址读写内存。
取地址 `movq $int32_arr, %r8` 。
写内存 `movw $0xEEEE, 6(%r8)` 。
读内存 `movl 4(%r8), %edx` 。

指针类型转换，不同类型的指针可以相互转换。指针占用 8 个字节，转换时把值拷贝。

```
int32_t *int32_ptr; // int32_t 类型的指针
double *double_ptr; // double 类型的指针
int64_t *int64_ptr; // int64_t 类型的指针
double_ptr = (double *)int32_ptr;  // 转换
int64_ptr = (int64_t *)double_ptr; // 转换
```

用 C 和汇编查看多种类型的指针

编写代码： type_pointer.c

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

// 整数的指针
int32_t int32 = 200;
int32_t *int32_ptr = &int32;

// 浮点数的指针
double float64 = 555.55;
double *float64_ptr = &float64;

// 数组的指针
int16_t int16_arr[3] = {7, 8, 9};
int16_t *int16_arr_ptr = &(int16_arr[1]);

void func_cat_run(int speed)
{
    printf(" func Cat is running at speed %d \n", speed);
```

```
}
// 函数的指针
void (*func_ptr)(int param) = func_cat_run;

int main()
{
    // 用指针修改变量。
    *int32_ptr = 300;
    *float64_ptr = 666.66;

    // 用指针调用函数。
    func_ptr(77);

    return 0;
}
```

编译代码：

```
gcc type_pointer.c -o type_pointer
gcc type_pointer.c -S -o type_pointer.s
```

运行代码：

```
[root@local pointer]# ./type_pointer
 func Cat is running at speed 77
```

分析结果：

不同类型的指针，汇编类型相同，都为.quad，占用 8 个字节。

分类	C 语言代码	汇编代码
整数的指针	`int32_t *int32_ptr = &int32;`	`int32_ptr:` `.quad int32`
浮点数的指针	`double *float64_ptr = &float64;`	`float64_ptr:` `.quad float64`
数组的指针	`int16_t *int16_arr_ptr = &(int16_arr[1]);`	`int16_arr_ptr:` `.quad int16_arr+2`
函数的指针	`void (*func_ptr)(int param) = func_cat_run;`	`func_ptr:` `.quad func_cat_run`

用 C 和汇编实现用指针遍历内存

编写代码： scan_mem.c

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

struct cat
{
```

```
    int32_t cat_age;
    float cat_height;
};

// 32 位整数
int32_t today = 11;
// 32 位浮点数
float degree = 22.22F;
// 结构体
struct cat cat_tom = {.cat_age = 55, .cat_height = 66.66F};
// 64 位整数
uint64_t level = 0x55667788AABBCCDD;

int main()
{
    // 地址。
    uint64_t addr = (uint64_t)&today;
    printf(" addr  =  %llu \n\n", addr);
    // 指针。指向单个字节。
    char *ptr = (char *)addr;

    // 用指针遍历内存。
    // 地址依次加 4。
    int32_t value_today = *((int32_t *)(ptr + 0));
    float value_degree = *((float *)(ptr + 4));
    int32_t value_cat_age = *((int32_t *)(ptr + 8));
    float value_cat_height = *((float *)(ptr + 12));
    uint32_t value_level_1 = *((uint32_t *)(ptr + 16));
    uint32_t value_level_2 = *((uint32_t *)(ptr + 20));

    printf("     today  =  %d  \n", value_today);
    printf("    degree  =  %.2f  \n", value_degree);
    printf("   cat_age  =  %d  \n", value_cat_age);
    printf(" cat_height =  %.2f  \n", value_cat_height);
    printf("   level_1  =  %#X  \n", value_level_1);
    printf("   level_2  =  %#X  \n", value_level_2);
    return 0;
}
```

编译代码：

```
gcc scan_mem.c -o scan_mem
readelf -a scan_mem > scan_mem.elf.txt
```

运行代码：

```
[root@local pointer]# ./scan_mem
 addr  =  6295608

     today  =  11
    degree  =  22.22
```

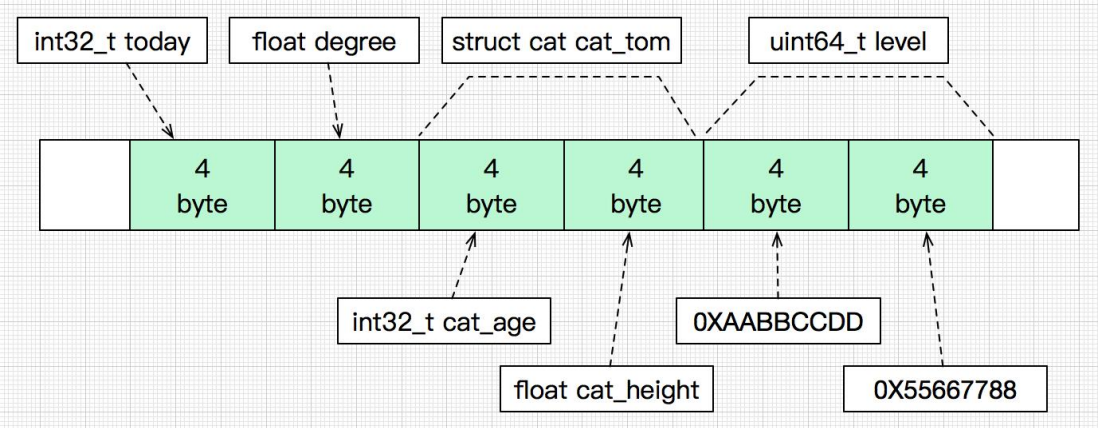
```
   cat_age  =  55
 cat_height =  66.66
   level_1  =  0XAABBCCDD
   level_2  =  0X55667788
```

分析结果：
查看文件 scan_mem.elf.txt，找到符号表。

```
Symbol table '.symtab' contains 67 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
    51: 0000000000601038     4 OBJECT  GLOBAL DEFAULT   24 today
    58: 000000000060103c     4 OBJECT  GLOBAL DEFAULT   24 degree
    63: 0000000000601040     8 OBJECT  GLOBAL DEFAULT   24 cat_tom
    64: 0000000000601048     8 OBJECT  GLOBAL DEFAULT   24 level
```

ELF 中，符号的地址顺序为 today、degree、cat_tom、level。
源码中，依次遍历符号 today、degree、cat_tom、level。
因为顺序相同，所以遍历内存时输出的符号和值对应。

指针的起始值为 today 的地址。用指针依次遍历内存，指针每次增加 4 个字节。
int32_t today 占用 4 个字节。
float degree 占用 4 个字节。
struct cat cat_tom 占用 8 个字节。属性 int32_t cat_age 占用 4 个字节，属性 float cat_height 占用 4 个字节。
uint64_t level 占用 8 个字节。小端表示，前 4 个字节为 0XAABBCCDD，后 4 个字节为 0X55667788。



用汇编实现指针拆分整数

编写代码： split_int.s

```
.data

int32_arr :  # 32 位整数的数组
    .long 0x11111111
    .long 0xAAAAAAAA

str32: .string  " int32 =  %#X  %#X \n"
```

```
.text
.global main

main :
    pushq %rbp
    movq %rsp, %rbp

    movq $int32_arr, %rdi
    callq print_4byte

    movq $int32_arr, %r8           # 取地址
    movq $0x2222222233333333, %r9  # 写值。64 位。
    movq %r9, (%r8)

    movq $int32_arr, %rdi
    callq print_4byte

    movq $int32_arr, %r8    # 取地址
    movw $0xBBBB, 0(%r8)    # 写值。16 位。
    movw $0xCCCC, 2(%r8)    # 写值。16 位。
    movw $0xDDDD, 4(%r8)    # 写值。16 位。
    movw $0xEEEE, 6(%r8)    # 写值。16 位。

    movq $int32_arr, %rdi
    callq print_4byte

    popq %rbp
    retq

print_4byte :       # 每次打印 4 个字节
    pushq %rbp
    movq %rsp, %rbp

    movq %rdi, %r8      # 地址。指针。
    movl 0(%r8), %esi   # 取第 1 个整数。地址+0
    movl 4(%r8), %edx   # 取第 2 个整数。地址+4

    movq $str32, %rdi
    callq printf

    popq %rbp
    retq
```

编译代码：

```
gcc split_int.s -o split_int
```

运行代码：

```
[root@local pointer]# ./split_int
 int32 =  0X11111111  0XAAAAAAAA
 int32 =  0X33333333  0X22222222
 int32 =  0XCCCCBBBB  0XEEEEDDDD
```

分析结果：
数组 int32_arr 包含 2 个 32 位整数。初始值为 `0X11111111  0XAAAAAAAA` 。
把 1 个 64 位整数 0x2222222233333333，写到 int32_arr 之后，输出为 `0X33333333  0X22222222` 。
把 4 个 16 位整数 0xBBBB、0xCCCC、0xDDDD、0xEEEE，写到 int32_arr 之后，输出为 `0XCCCCBBBB  0XEEEEDDDD` 。