**offsetof和container\_of两个宏**

功能很强大。offsetof是用来判断结构体中成员的偏移位置，

container\_of宏用来根据成员的地址来获取结构体的地址。两个宏设计的很巧妙，值得学习。linux内核中有着两个宏的定义，并在链表结构中得到应用。不得不提一下linux内核中的链表，设计的如此之妙，只需要两个指针就搞定了。

1、offsetof宏

　　使用offsetof宏需要包含stddef.h头文件，实例可以参考：<http://www.cplusplus.com/reference/cstddef/offsetof/>。

offsetof宏的定义如下：

#define offsetof(type, member) (size\_t)&(((type\*)0)->member)

　　巧妙之处在于将地址0强制转换为type类型的指针，从而定位到member在结构体中偏移位置。编译器认为0是一个有效的地址，从而认为0是type指针的起始地址。

2、container\_of宏

　　使用container\_of宏需要包含linux/kernel.h头文件，container\_of宏的定义如下所示：

#define container\_of(ptr, type, member) ({ \

const typeof( ((type \*)0)->member ) \*\_\_mptr = (ptr); \

(type \*)( (char \*)\_\_mptr - offsetof(type,member) );})

container\_of宏分为两部分，

第一部分：**const typeof( ((type \*)0)->member ) \*\_\_mptr = (ptr);**

通过typeof定义一个member指针类型的指针变量\_\_mptr，（即\_\_mptr是指向member类型的指针），并将\_\_mptr赋值为ptr。

第二部分**： (type \*)( (char \*)\_\_mptr - offsetof(type,member) )**，通过offsetof宏计算出member在type中的偏移，然后用member的实际地址\_\_mptr减去偏移，得到type的起始地址，即指向type类型的指针。

第一部分的目的是为了将统一转换为member类型指针。

3、测试程序

#include <stdio.h>

#include <stdlib.h>

#define NAME\_STR\_LEN 32

#define offsetof(type, member) (size\_t)&(((type\*)0)->member)

#define container\_of(ptr, type, member) ({ \

const typeof( ((type \*)0)->member ) \*\_\_mptr = (ptr); \

(type \*)( (char \*)\_\_mptr - offsetof(type,member) );})

typedef struct student\_info

{

int id;

char name[NAME\_STR\_LEN];

int age;

}student\_info;

int main()

{

size\_t off\_set = 0;

off\_set = offsetof(student\_info, id);

printf("id offset: %u\n",off\_set);

off\_set = offsetof(student\_info, name);

printf("name offset: %u\n",off\_set);

off\_set = offsetof(student\_info, age);

printf("age offset: %u\n",off\_set);

student\_info \*stu = (student\_info \*)malloc(sizeof(student\_info));

stu->age = 10;

student\_info \*ptr = container\_of(&(stu->age), student\_info, age);

printf("age:%d\n", ptr->age);

printf("stu address:%p\n", stu);

printf("ptr address:%p\n", ptr);

return 0;

}

//====================================

MODULE\_DEVICE\_TABLE一般用在热插拔的设备驱动中。

上述xx\_driver\_ids结构，是此驱动所支持的设备列表。

作用是：将xx\_driver\_ids结构输出到用户空间，这样模块加载系统在加载模块时，就知道了什么模块对应什么硬件设备。

用法是：MODULE\_DEVICE\_TABLE（设备类型，设备表），其中，设备类型，包括USB,PCI等，也可以自己起名字，上述代码中是针对不同的平台分的类；设备表也是自己定义的，它的最后一项必须是空，用来标识结束。