

C语言面向对象(针对较大型软件)

宋宝华

CSDN直播时间:2017.11.16 20:00 - 21:30

报名地址:http://edu.csdn.net/huiyiCourse/detail/594?utm_source=wx2

麦当劳喜欢您来，喜欢您再来



扫描关注

Linuxer



大纲

- 1.面向对象的封装、继承和多态
- 2.C语言实现封装、函数指针
- 3.从一个命令解析器的案例开始
 - a. 过程化程序设计
 - b. 采用封装思想设计
4. 含很多模块的软件初始化
- 5.一个嵌入式图形框架的C面向对象
- 6.C语言实现继承和多态，模拟虚函数
- 7.提炼软件的中间层
- 8.综合案例，对象式思维锻炼

面向对象解释

封装、继承、多态

人:中年人, 青年人, 婴儿

吃饭:筷子吃饭, 奶瓶吃饭

大象进冰箱

大象打开冰箱门。大象走进去，大象关上门？

冰箱
大象

类

```
#ifndef C_Class
#define C_Class struct
#endif

C_Class A
{
    C_Class A *A_this;      /* this指针 */
    void (*Foo)(C_Class A *A_this); /* 行为: 函数指针 */
    int a;                  /* 数据 */
    int b;
};
```

提炼数据结构

WAVE文件格式说明表

	偏移地址	字节数	数据类型	内 容
文件头	00H	4	Char	"RIFF"标志
	04H	4	int32	文件长度
	08H	4	Char	"WAVE"标志
	0CH	4	Char	"fmt"标志
	10H	4		过渡字节 (不定)
	14H	2	int16	格式类别
	16H	2	int16	通道数
	18H	2	int16	采样率 (每秒样本数), 表示每个通道的播放速度
	1CH	4	int32	波形音频数据传送速率
	20H	2	int16	数据块的调整数 (按字节算的)
	22H	2		每样本的数据位数
	24H	4	Char	数据标记符 "data "
	28H	4	int32	语音数据的长度



```
typedef struct tagWaveFormat
{
    char cRiffFlag[4];
    UIN32 nFileLen;
    char cWaveFlag[4];
    char cFmtFlag[4];
    char cTransition[4];
    UIN16 nFormatTag;
    UIN16 nChannels;
    UIN16 nSamplesPerSec;
    UIN32 nAvgBytesperSec;
    UIN16 nBlockAlign;
    UIN16 nBitNumPerSample;
    char cDataFlag[4];
    UIN16 nAudioLength;
} WAVEFORMAT;
```

命令解析器（过程式设计）

cmd.c

```
switch(命令码)
{
case GPIO高:
    gpio_high();
case GPIO低:
    gpio_low();
case SPI发:
    spi_send();
case I2C收:
    i2c_recv();
...
}
```

gpio.c

```
gpio_high()
{
}

gpio_low()
{
}
```

spi.c

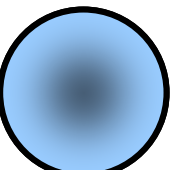
```
spi_send()
{
}
```

i2c.c

过程设计引起的问题：

- 1.命令增减后引起跨模块修改,很low的switch case
- 2.大量的外部函数，如gpio_high(),高耦合

1. 增加或者减少命令不影响cmd.c:
消除很low的switch case
2. 命令的处理函数要便成为static:
去耦合




命令解析器（对象式设计）

cmd.h

```
struct cmd
{
    命令码
    *ops
}
```

```
register_cmds
(
    struct cmd cmds[],
    int num
);
```

cmd.c

```
register_cmds(struct cmd cmds[], int num)
{
    
}
```

```
dispatch_cmds(输入码)
{
    for(遍历cmds)
        if(node[i].命令码 == 输入码)
            node[i]->ops();
}
```

gpio.c

```
static gpio_high()
{
}

static gpio_low()
{
}
```

```
struct cmd gpio_cmds[] =
{
    {"gpio_H", gpio_high},
    {"gpio_L", gpio_low},
};
```

```
register_cmds(gpio_cmds, 2);
```

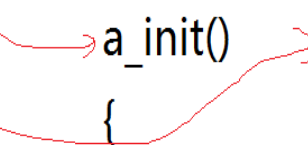
ARRAY_SIZE(gpio_cmds)

```
#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
```

软件初始化过程

各个模块的初始化函数 应该是内部函数而不是外部函数

```
main.c      a.c      b.c
{
a_init();   a_init()
b_init();   {
c_init();   }
...
}
```

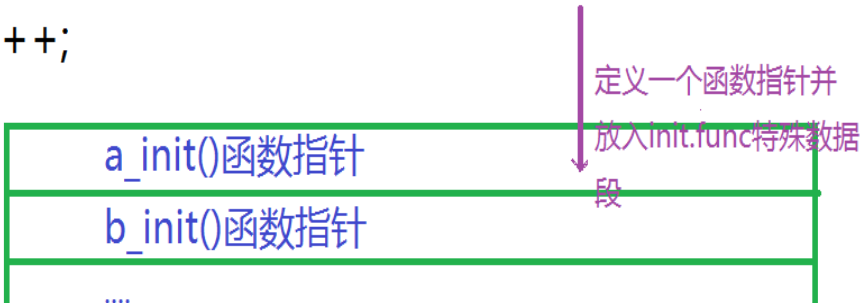


缺点：所有的初始化
函数都是外部函数

```
main.c      a.c
{
ops=init.func.s
for(所有初始化函数指针) {
ops();
ops++;
}
}

static a_init()
{
}

DECLARE_INIT(a_init)
```



封装(1)

我读研的时候写的代码



```
/* 按下OK键 */
void onOkkey()
{
    /* 判断在什么焦点菜单上按下ok键，调用相应处理函数 */
    switch(currentFocus)
    {
        case MENU1:
            menu1onOk();
            break;
        case MENU2:
            menu2onOk();
            break;
        ...
    }
}

/* 按下Cancel键 */
void onCancelkey()
{
    /* 判断在什么焦点菜单上按下Cancel键，调用相应处理函数 */
    switch(currentFocus)
    {
        case MENU1:
            menu1onCancel();
            break;
        case MENU2:
            menu2onCancel();
            break;
        ...
    }
}
```

封装(2)

```
typedef struct tagSysMenu
{
    char *text; /* 菜单的文本 */
    BYTE xPos; /* 菜单在LCD上的x坐标 */
    BYTE yPos; /* 菜单在LCD上的y坐标 */
    void (*onOkFun)(); /* 在该菜单上按下ok键的处理函数指针 */
    void (*onCancelFun)(); /* 在该菜单上按下cancel键的处理函数指针 */
}SysMenu, *LPSysMenu;
```

```
static SysMenu menu[MENU_NUM] =
{
    {
        "menu1", 0, 48, menu1OnOk, menu1OnCancel
    },
    {
        " menu2", 7, 48, menu2OnOk, menu2OnCancel
    },
    {
        " menu3", 7, 48, menu3OnOk, menu3OnCancel
    },
    {
        " menu4", 7, 48, menu4OnOk, menu4OnCancel
    }
};

/* 按下OK键 */
void onOkKey()
{
    menu[currentFocusMenu].onOkFun();
}

/* 按下Cancel键 */
void onCancelKey()
{
    menu[currentFocusMenu].onCancelFun();
}
```

封装(3)



1. 将不同的画面类比为WIN32中不同的窗口，将窗口中的各种元素（菜单、按钮等）包含在窗口之中；
2. 给各个画面提供一个功能键“消息”处理函数，该函数接收按键信息为参数；
3. 在各画面的功能键“消息”处理函数中，判断按键类型和当前焦点元素，并调用对应元素的按键处理函数。

继承和多态的模拟：软件分层

中间层三大功能：

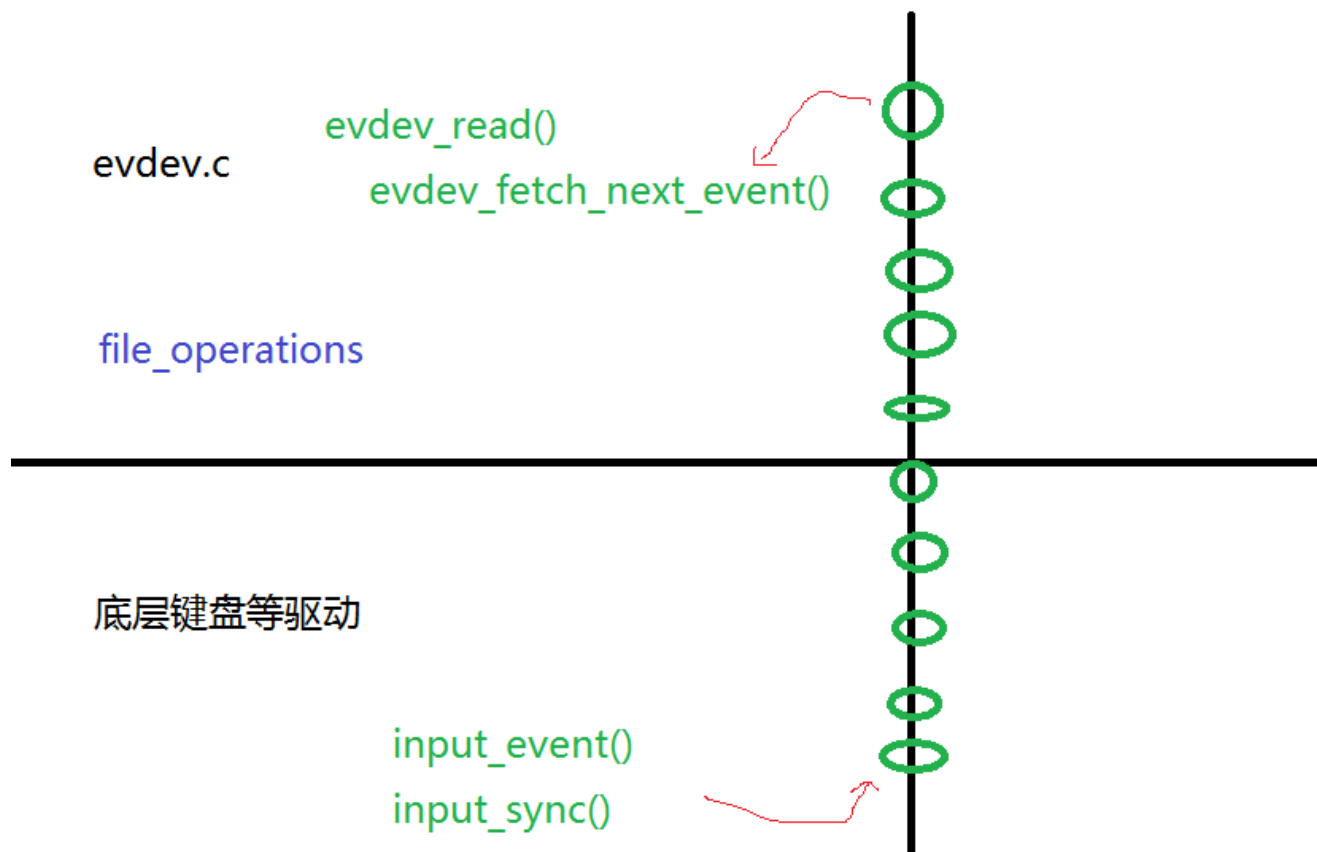
- ✓ 对上提供接口；
- ✓ 中间层实现通用逻辑；
- ✓ 对下定义框架。

```
return_type core_funca(xxx_device * bottom_dev, param1_type param1, param1_type param2)
{
    if (bottom_dev->funca)
        return bottom_dev->funca(param1, param2);
    /* 核心层通用的funca代码 */
    ...
}

return_type core_funca(xxx_device * bottom_dev, param1_type param1, param1_type param2)
{
    /*通用的步骤代码A */
    ...
    bottom_dev->funca_ops1();
    /*通用的步骤代码B */
    ...
    bottom_dev->funca_ops2();
    /*通用的步骤代码C */
    ...
    bottom_dev->funca_ops3();
}
```

核心层案例：Linux的input子系统

Linux的input核心层，最大程度地减少了每个input驱动的工作



类似“虚函数”

通过函数指针为NULL或者Not NULL来区分底层是否覆盖上层

```
static ssize_t
fb_write(struct file *file, const char __user *buf, size_t count, loff_t
{
    unsigned long p = *ppos;
    struct fb_info *info = file_fb_info(file);
    u8 *buffer, *src;
    u8 __iomem *dst;
    int c, cnt = 0, err = 0;
    unsigned long total_size;

    if (!info || !info->screen_base)
        return -ENODEV;

    if (info->state != FBINFO_STATE_RUNNING)
        return -EPERM;

    if (info->fbops->fb_write)
        return info->fbops->fb_write(info, buf, count, ppos);

    total_size = info->screen_size;

    if (total_size == 0)
        total_size = info->fix.smem_len;

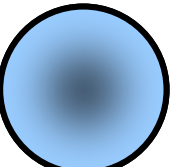
    if (p > total_size)
```

底层LCD想盖掉中间层

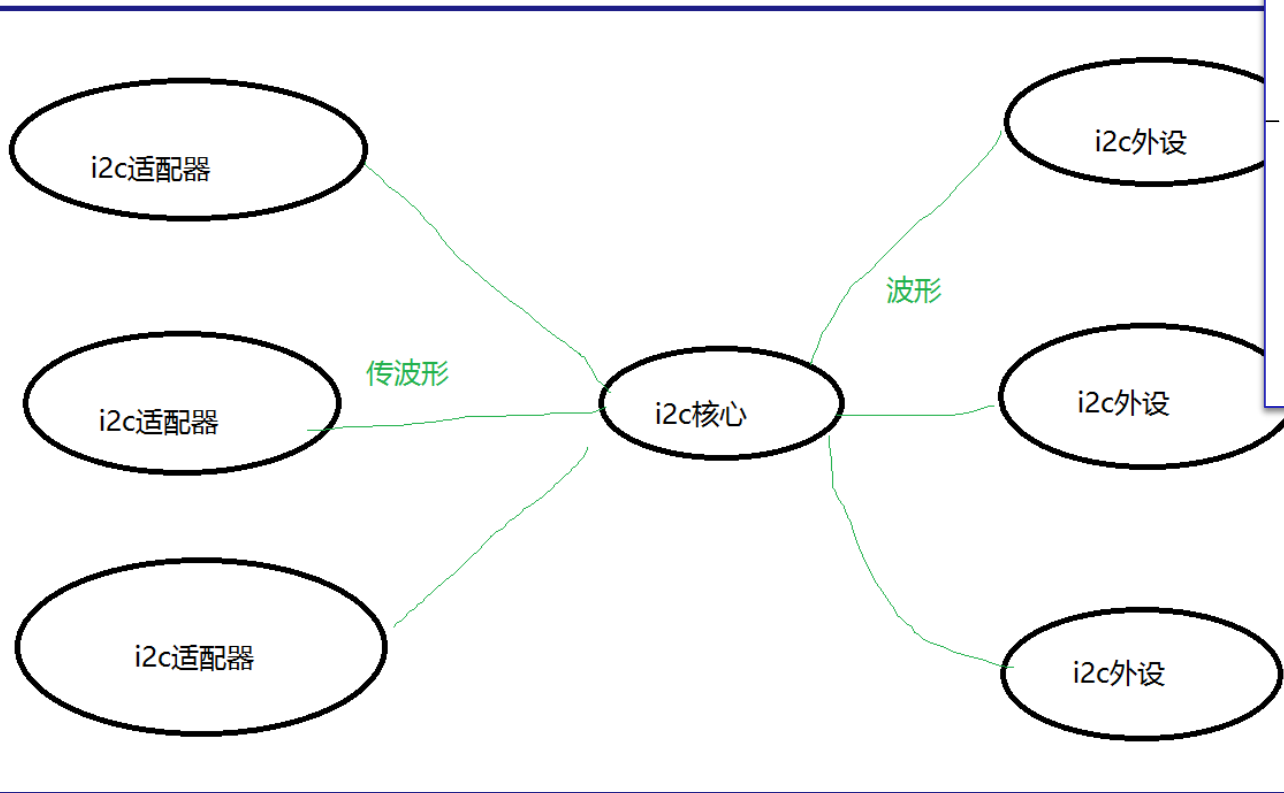
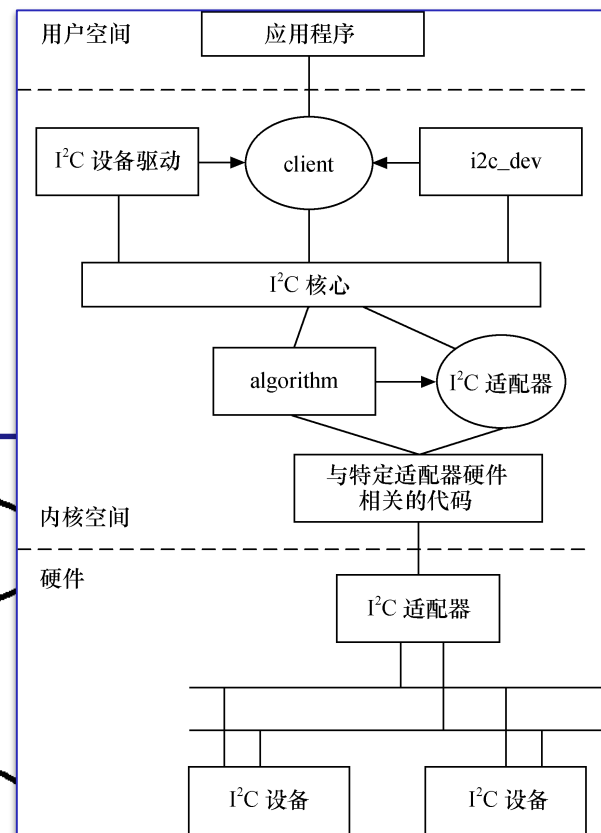
走中间层代码

先想出对象，然后从2个角度思维

1. 需要什么？ - 别人给你的API
2. 提供什么？ - 你给别人的API



综合案例：Linux的I2C/SPI/USB等子系统



宋宝华.2005年系列文章. 《C语言嵌入式系统编程修炼之道》:

<http://soft.yesky.com/lesson/188/2023188.shtml>

《Linux总线、设备、驱动模型》视频:

<http://edu.csdn.net/course/detail/5329>

《深入探究Linux的设备树》视频:

<http://edu.csdn.net/course/detail/5627>

《Linux进程、线程和调度》系列视频:

<http://edu.csdn.net/course/detail/5995>

谢谢!

