嵌入式C语言之-

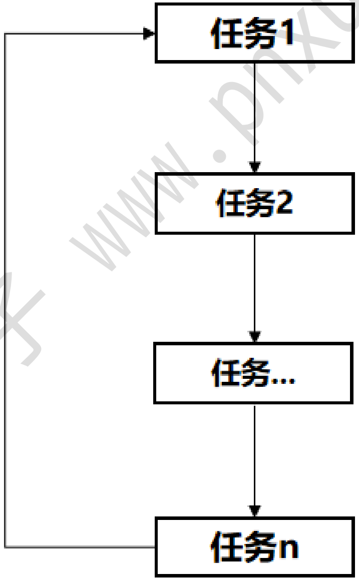
函数指针和回调函数

裸机程序的任务调度

助力你成为优秀的电子工程师！

裸机任务调度方案1，大锅饭

```
int  main(void)
{
    Init();
    while (1)
    {
        SensorTask();
        KeyScanTask();
        DisplayTask();
    }
}
```



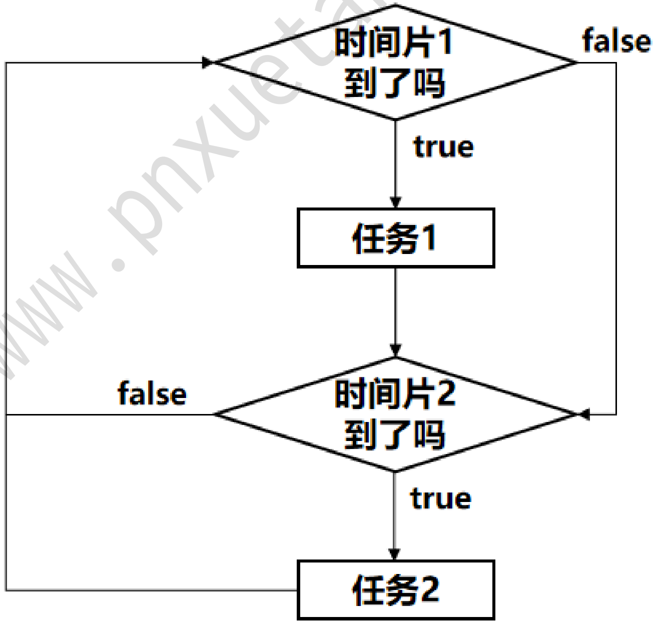
裸机任务调度方案2，按需分配，效率更高

```
int  main(void)
{
        Init();
        while (1)
        {

                if (period1sFlag)
                {
                        SensorTask();
                        period1sFlag = 0;
                }


                if (period20msFlag)
                {
                        KeyScanTask();
                        period20msFlag = 0;
                }
        }

}
```

```
void TimerInterrupt(void)
{
        if (period1sNum)
        {
                period1sNum--;
                 if (period1sNum == 0)
                 {
                            period1sFlag = 1;
                            period1sNum = 1000;
                 }
         }

        if (period20msNum)
        {
                period20msNum--;
                 if (period20msNum == 0)
                 {
                            period20msFlag = 1;
                            period20msNum = 20;
                 }
         }
}
```

裸机任务调度方案2，按需分配，效率更高

```
int  main(void)

{
        Init();
        while (1)
        {

                if (period1sFlag)
                {
                        SensorTask();
                        period1sFlag = 0;
                }


                if (period20msFlag)
                {
                        KeyScanTask();
                        period20msFlag = 0;
                }


}
```



裸机任务调度方案3，按需分配，软件架构更优

```
int main(void)
{
        SYS_Init();

        while (1)
        {
                TaskHandler();
        }
}
```

```
void TaskHandler(void)
{
        uint8_t i;
        for(i=0; i<Tasks_Max; i++)
        {
                if(Task_Comps[i].Run) /* 判断任务时间片标记 */
                {
                        Task_Comps[i].Run = 0;    /* 标记清零 */
                        Task_Comps[i].TaskHook();  /* 执行调度任务 */
                }
        }
}
```

裸机任务调度方案3，按需分配，软件架构更优

```
void TimerInterrupt(void)
{
        uint8_t i;
        for(i=0; i<Tasks_Max; i++)
        {
                if(Task_Comps[i].TIMCount)    /* 判断时间片计数 */
                {
                        Task_Comps[i].TIMCount--;  /* 时间片计数递减 */
                        if(Task_Comps[i].TIMCount == 0)
                        {
                                /*时间片标记为1，并重载计数初值 */
                                Task_Comps[i].TIMCount = Task_Comps[i].TRITime;
                                Task_Comps[i].Run = 1;
                        }
                }
        }
}
```

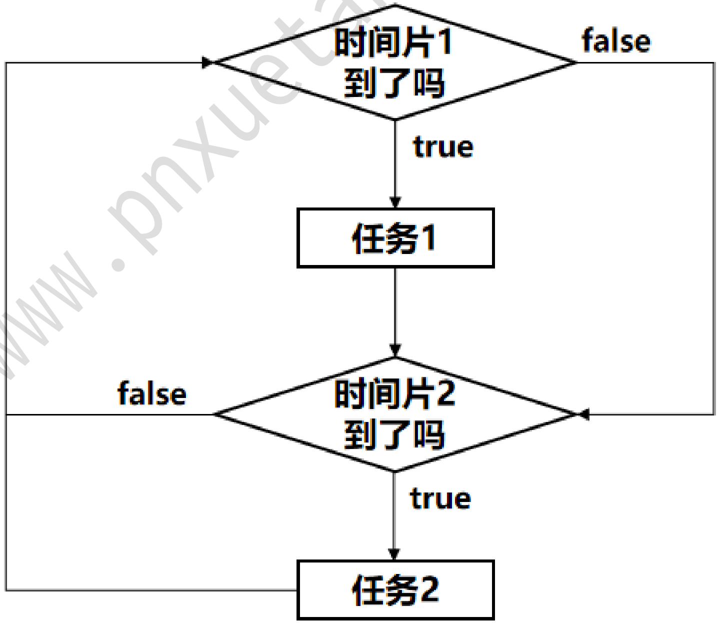
裸机任务调度方案3，按需分配，软件架构更优

```
typedef struct
{
        uint8_t Run;                //任务状态：Run/Stop
        uint16_t TIMCount;          //时间片周期，用于递减计数
        uint16_t TRITime;            //时间片周期，用于重载
        void (*TaskHook) (void);    //函数指针，保存任务函数地址
} TASK_COMPONENTS;
```

```
static TASK_COMPONENTS Task_Comps[]=
{
       //状态  计数  周期  函数
        {0, 1000, 1000, SensorTask},               /* task 1 Period:  1000ms */
        {0, 20, 20, KeyScanTask},                  /* task 2 Period:  20ms*/
        /* Add new task here */
};
```

裸机任务调度方案 VS RTOS

```
int  main(void)

{
        Init();
        while (1)
        {

                if (period1sFlag)
                {
                        SensorTask();
                        period1sFlag = 0;
                }


                if (period20msFlag)
                {
                        KeyScanTask();
                        period20msFlag = 0;
                }


}
```



FreeRTOS应用案例

```
BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,
                        const char * const pcName,
                        const configSTACK_DEPTH_TYPE usStackDepth,
                        void * const pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * const pxCreatedTask )

typedef void (* TaskFunction_t)( void * );
```

```
xTaskCreate( vLCDTask, "LCD", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
```

RT-Thread应用案例

```
rt_thread_t   rt_thread_create(const char *name,
                         void (*entry)(void *parameter),
                         void      *parameter,
                         rt_uint32_t stack_size,
                         rt_uint8_t  priority,
                         rt_uint32_t tick);

rt_err_t rt_thread_init(struct rt_thread *thread,
                     const char       *name,
                     void (*entry)(void *parameter),
                     void            *parameter,
                     void            *stack_start,
                     rt_uint32_t       stack_size,
                     rt_uint8_t       priority,
                     rt_uint32_t       tick);
```

```
   rt_thread_create( "send", led_thread_entry, RT_NULL, 512, 2, 20);
```

应用案例

```
void UserProgram(void)
{
    lv_timer_t * timer = lv_timer_create(RefreshClockUI, 1000, NULL);
}
```

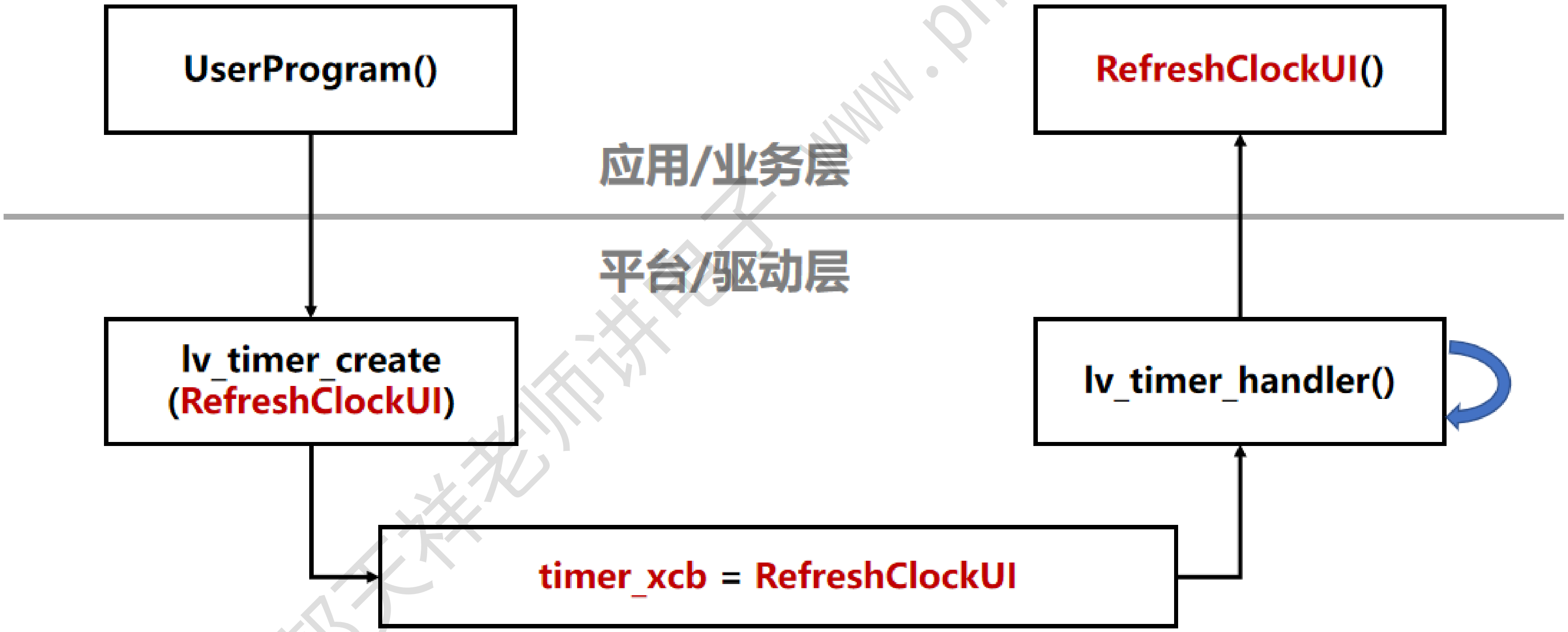
```
void RefreshClockUI(lv_timer_t *timer)

{

    time_t rawtime;

    struct tm *info;

    time(&rawtime);

    info = localtime(&rawtime);

    lv_label_set_text_fmt(label, "%02d:%02d:%02d",

        info->tm_hour, info->tm_min, info->tm_sec);

}
```

应用案例

```
void UserProgram(void)
{
    lv_timer_t * timer = lv_timer_create(RefreshClockUI, 1000, NULL);
}
```

```
void RefreshClockUI(lv_timer_t *timer)

{

    time_t rawtime;

    struct tm *info;

    time(&rawtime);

    info = localtime(&rawtime);

    lv_label_set_text_fmt(label, "%02d:%02d:%02d",

        info->tm_hour, info->tm_min, info->tm_sec);

}
```

```
void UserProgram(void)
{
    lv_timer_t * timer = lv_timer_create(RefreshClockUI, 1000, NULL);
}
```



UserProgram()

RefreshClockUI()

应用/业务层

平台/驱动层

lv_timer_create
(RefreshClockUI)

lv_timer_handler()

timer_xcb = RefreshClockUI

回调函数和函数指针

```
void UserProgram(void)
{
    lv_timer_t * timer = lv_timer_create(RefreshClockUI, 1000, NULL);
}
```

- void RefreshClockUI(lv_timer_t *timer)，称为回调函数，回调函数本身也是普通函数，只是因为调用关系比较特别，它的代码位于上层业务层，却是由下层库代码去调用，所以叫做回调函数；

```
lv_timer_t * lv_timer_create(lv_timer_cb_t timer_xcb, uint32_t period, void * user_data)
```

- lv_timer_cb_t timer_xcb，timer_xcb称为函数指针，它用来保存回调函数的地址，严谨一些，应该称为函数指针类型的**变量**/函数指针变量；

函数指针变量

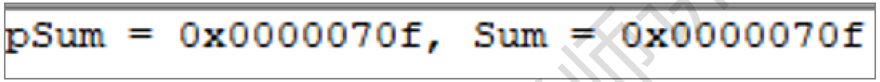
- 格式为：

  函数返回值类型 (* 函数指针变量名) (函数参数列表);

- int32_t (*pSum)(int32_t a, int32_t b);

  函数指针变量pSum，就像int32_t *ptr里的ptr一样;

- 函数名称就像数组名称一样保存了函数地址：



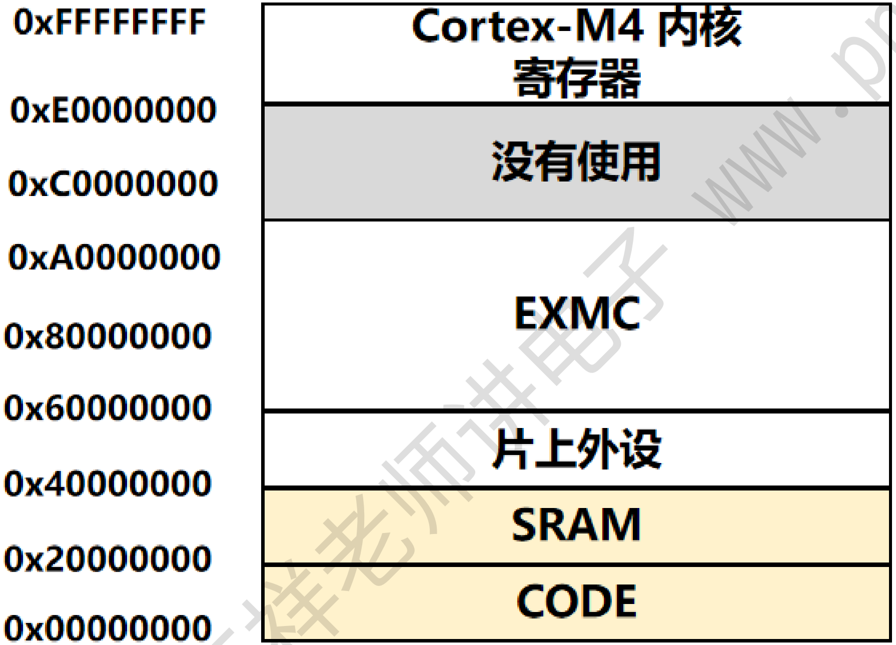
```
pSum = 0x0000070f, Sum = 0x0000070f
```

- (*pSum)(1, 2)，表示间接访问并调用Sum函数。

```
int32_t Sum(int32_t x, int32_t y)
{
        return x + y;
}
int main(void)
{
        int32_t (*pSum)(int32_t a, int32_t b);
        pSum = Sum;
        printf("pSum = 0x%p, Sum = 0x%p\n",
                    pSum, Sum);
        int32_t sum = (*pSum)(1, 2);
        printf("%d\n", sum);
        return 0;
}
```

单片机寻址范围

- 单片机通过地址来访问FLASH、内存和寄存器，ARM寻址范围4GB，分为多个块，FLASH对应地址范围是0x00000000-0x20000000。



0xFFFFFFFF Cortex-M4 内核寄存器

0xE0000000 没有使用

0xC0000000

0xA0000000 EXMC

0x80000000

0x60000000 片上外设

0x40000000 SRAM

0x20000000 CODE

0x00000000

注：基于GD32F303单片机

函数指针和指针函数

- int32_t (*pSum)(int32_t a, int32_t b); 为什么(*pSum)要使用()?

- 如果不使用()，变成了int32_t *pSum(int32_t a, int32_t b)；基于运算符优先级，pSum先结合()再结合*，这种格式被称为指针函数，表示返回值为指针类型的函数，比如常见的：

```
void *malloc(size_t size)

char *strcpy(char *dest, const char *src)
```

- 使用()，基于运算符优先级，pSum先结合*再结合后面的()，这种格式用来定义函数指针变量，变量是pSum。

函数指针

```c
int32_t Sum(int32_t x, int32_t y)
{
    return x + y;
}
int main(void)
{
    int32_t (*pSum)(int32_t a, int32_t b);
    pSum = Sum;
    int32_t sum = (*pSum)(1, 2);
    printf("%d\n", sum);
    return 0;
}
```

```c
int32_t Sum(int32_t x, int32_t y)
{
    return x + y;
}
void RegAndHandle(int32_t (*pSum)(int32_t a, int32_t b));
int main(void)
{
    RegAndHandle(Sum);
    return 0;
}
void RegAndHandle(int32_t (*pSum)(int32_t a, int32_t b))
{
    int32_t sum = (*pSum)(1, 2);
    printf("%d\n", sum);
}
```

函数指针类型和函数指针变量

```
void RegAndHandle(int32_t (*pSum)(int32_t a, int32_t b))
```

- 如果程序中很多地方都需要定义这种函数指针类型的变量，书写起来太繁琐，可以使用typedef重定义：

```
typedef int32_t (*PFUNC)(int32_t a, int32_t b);

void RegAndHandle(PFUNC pSum)
{
    int32_t sum = (*pSum)(1, 2);
    printf("%d\n", sum);
}
```

函数指针类型和函数指针变量

```
typedef int32_t (*PFUNC)(int32_t a, int32_t b);
```

➢ 为什么函数指针类型的变量PFUNC还可以作为数据类型?

1. 这里typedef，和常规用法不太一样：

```
typedef   signed  char    int8_t;
```

2. 当有typedef时，PFUNC表示函数指针类型，PFUNC pSum；当没有typedef时，int32_t (*pSum)(int32_t a, int32_t b)，pSum表示变量，pSum = Sum。

FreeRTOS应用案例

```
BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,
                        const char * const pcName,
                        const configSTACK_DEPTH_TYPE usStackDepth,
                        void * const pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * const pxCreatedTask )

typedef void (* TaskFunction_t)( void * );
```

```
xTaskCreate( vLCDTask, "LCD", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
```

RT-Thread应用案例

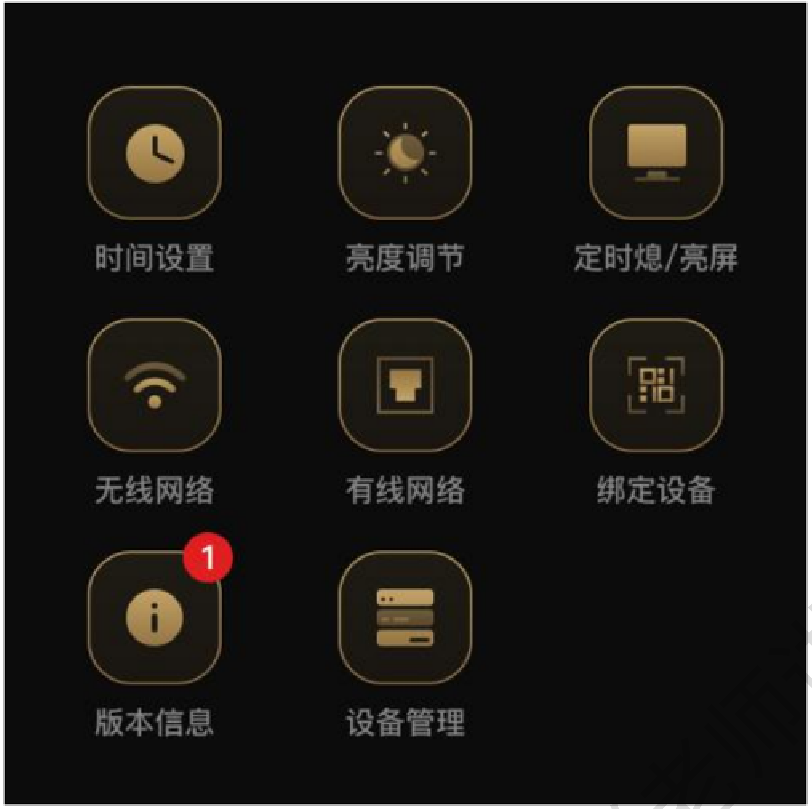
```
rt_thread_t   rt_thread_create(const char *name,
                        void (*entry)(void *parameter),
                        void       *parameter,
                        rt_uint32_t stack_size,
                        rt_uint8_t  priority,
                        rt_uint32_t tick);

rt_err_t rt_thread_init(struct rt_thread *thread,
                     const char       *name,
                     void (*entry)(void *parameter),
                     void             *parameter,
                     void             *stack_start,
                     rt_uint32_t       stack_size,
                     rt_uint8_t        priority,
                     rt_uint32_t       tick);
```

```
   rt_thread_create( "send", led_thread_entry, RT_NULL, 512, 2, 20);
```

函数指针扩展

```
int32_t Sum(int32_t x, int32_t y)
{
    return x + y;
}
int main(void)
{
    int32_t (*pSum)(int32_t a, int32_t b);
    pSum = Sum;
    printf("pSum = 0x%p, Sum = 0x%p\n", pSum, Sum);
    int32_t sum = (*pSum)(1, 2);
    printf("%d\n", sum);
    sum = pSum(1, 2);
    sum = (*Sum)(1, 2);
    return 0;
}
```

函数指针应用案例



```
typedef struct desktop_interface
{
    const lv_img_dsc_t *app_icon;
    const char *app_name;
    void (*app_event_cb)(lv_event_t * event);
} AppInfo_t;

static AppInfo_t appInfo[] =
{
    {&img_set_time, "时间设置", set_time_event_cb},
    {&img_set_backlight, "亮度调节", set_backlight_event_cb},
    {&img_dev_manage, "设备管理", dev_manage_event_cb},
    ...
}
uint8_t APP_MAX = sizeof(appInfo)/sizeof(appInfo[0]);
...
for (uint8_t i = 0; i < APP_MAX; i++)
{
```

THANK YOU！