

正点原子 littleVGL 开发指南

Task 任务系统

开发指南

正点原子
广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.00	2020/05/01	第一次发布

Tasks 任务系统

1. 介绍

littleVGL 自带了一个任务管理系统,此任务系统除了给 littleVGL 内部使用外,还开放出来给我们用户使用,这给我们的应用程序设计带来了极大的便利,它支持 6 个任务优先级,高优先级的任务可以抢占低优先级的任务,注意,此任务管理系统是非实时的,因为任务的时效性是取决于 `lv_task_handler()` 函数的调用,而 `lv_task_handler()` 函数一般放在 `main` 函数主循环中进行调用,所以你也确保 `main` 函数主循环中不能有其他过大的延时存在,满足上述要求之后,对于一般的应用来说,此任务系统的时效性误差可以忽略不计.对于我们用户而言, littleVGL 的任务管理系统主要是由 3 个数据类型和 13 个 API 接口组成的,下面我们来详细介绍.

注: littleVGL 的任务管理系统类似于软件定时器,他们两者之间具有许多相同的特性

2.任务系统的 API 接口

2.1 主要数据类型

跟任务管理系统相关的数据类型主要有 3 个,分别为任务回调函数数据类型,任务优先级数据类型,任务管理句柄数据类型.

2.1.1 任务回调函数数据类型

```
typedef void (*lv_task_cb_t)(struct _lv_task_t *);
```

上面的申明是在定义一种函数指针数据类型,如看不懂的同学,请先去熟悉一下 c 语言指针方面的知识,此数据类型就是用来约束任务回调函数的申明,保证我们写的任务回调函数只能有一个 struct _lv_task_t *形参和返回值必须是 void 的

2.1.2 任务优先级数据类型

```
enum {  
    LV_TASK_PRIO_OFF = 0,  
    LV_TASK_PRIO_LOWEST,  
    LV_TASK_PRIO_LOW,  
    LV_TASK_PRIO_MID,  
    LV_TASK_PRIO_HIGH,  
    LV_TASK_PRIO_HIGHEST,  
    _LV_TASK_PRIO_NUM,  
};  
typedef uint8_t lv_task_prio_t;
```

任务优先级数据类型的本质就是一个 enum 枚举体,这很简单,总共具有 LV_TASK_PRIO_OFF 到 LV_TASK_PRIO_HIGHEST 之间的 6 个优先级,这里需要注意, _LV_TASK_PRIO_NUM(值为 6)对于我们来说没有实际意义,它是归 littleVGL 任务管理内部使用的,用来记录总共有 6 个优先级,然后还可以用作优先级形参的合法性判断,比如

```
if(prio>=_LV_TASK_PRIO_NUM)  
    printf(“prio 是非法的优先级形参”);
```

另外还需要注意一下 LV_TASK_PRIO_OFF 优先级,它是停止的意思,也就是说当任务设置为此优先级时,实际上就是不运行或者停止运行

2.1.3 任务管理句柄数据类型

```
typedef struct _lv_task_t
{
    uint32_t period; //任务的回调周期
    uint32_t last_run; //最近一次的运行时间点
    lv_task_cb_t task_cb; //任务回调函数

    void * user_data; //用户自定义数据

    uint8_t prio : 3; //任务的优先级,占 3 位
    uint8_t once : 1; //记录此任务是否只运行一次
} lv_task_t;
```

lv_task_t 是任务句柄数据类型,也可以说是任务对象数据类型,俩者的意思是一样的,只是不同的叫法,以后我统统以句柄为例,我们一般是不直接修改 lv_task_t 句柄的属性的,都是通过其他的 API 接口来操作的

2.2 API 接口

跟任务管理系统相关的 API 接口有 13 个,API 接口也就是所谓的函数或者方法,以后统统以 API 接口为称呼.

2.2.1 核心初始化

```
void lv_task_core_init(void);
```

想要使用 littleVGL 的任务管理系统,就必须得调用 lv_task_core_init 进行初始化一下,但是好处在于不需要我们自己去手动调用了,littleVGL 内部已经帮我们完成了初始化调用,如下图所示:

```
72  /**
73   * Init. the 'lv' library.
74   */
75  void lv_init(void)
76  {
77      /* Do nothing if already initialized */
78      if(lv_initialized) {
79          LV_LOG_WARN("lv_init: already initied");
80          return;
81      }
82
83      LV_LOG_TRACE("lv_init started");
84
85      /*Initialize the lv_misc modules*/
86      lv_mem_init();
87      lv_task_core_init();
88  }
```

图 2.2.1.1 lv_task_core_init 的内部调用

在 lv_init 函数中可以看到 lv_task_core_init 函数的调用,而我们又在 main 函数中调用了 lv_init 函数,所以我们完全可以不用去理会 lv_task_core_init 这个 API 接口

2.2.2 事务处理器

```
LV_ATTRIBUTE_TASK_HANDLER void lv_task_handler(void);
```

LV_ATTRIBUTE_TASK_HANDLER 是在 lv_conf.h 配置文件中定义的一个宏,默认情况下是为空的,不用理会,lv_task_handler 这个 API 接口是非常重要的,littleVGL 内部的所有事务都是通过这个接口来处理的,所以我称它为事务处理器,它需要不断的被周期性调用,我们通常把它放到 main 函数的主循环中去,如下图所示:

```

26 int main(void)
27 {
28     delay_init();           //延时函数初始化
29     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断优
30     uart_init(115200);      //串口初始化为115200
31     LED_Init();             //LED端口初始化
32     KEY_Init();             //按键初始化
33     TIM3_Int_Init(999,71);  //定时器初始化(1ms中断),用于给lvgl
34     FSMC_SRAM_Init();       //外部1MB的sram初始化
35     LCD_Init();             //LCD初始化
36     tp_dev.init();          //触摸屏初始化
37
38     lv_init();               //lvgl系统初始化
39     lv_port_disp_init();     //lvgl显示接口初始化
40     lv_port_indev_init();    //lvgl输入接口初始化
41
42     //demo_create();         //开始运行demo
43
44     while(1)
45     {
46         tp_dev.scan(0);
47         lv_task_handler();
48     }
49 }

```

图 2.2.2.1 lv_task_handler 的调用位置

2.2.3 创建最基本的任务

```
lv_task_t * lv_task_create_basic(void);
```

这个函数是用来创建最基本的任务对象,创建完成之后,还需要通过其他的 API 接口来设置其属性,这个 API 接口归 littleVGL 内部使用,我们一般不调用这个接口来创建任务,因为太麻烦了

2.2.4 创建任务

```
lv_task_t * lv_task_create(lv_task_cb_t task_xcb, uint32_t period, lv_task_prio_t prio, void * user_data);
```

参数:

task_xcb: 任务回调函数

period: 任务回调周期,单位 ms,默认是按此值周期性的执行任务回调函数

prio: 任务优先级

user_data: 用户自定义数据,可以传递到任务回调函数中,如不使用的话,赋 NULL 值

返回值:

返回创建出来的任务句柄指针,后面可以通过此句柄指针来修改其属性

我们创建任务一般都是通过这个 API 接口来进行的,可以一步到位,其实 lv_task_create 的实现原理是先通过调用 lv_task_create_basic 来创建最基本的任务,然后再通过其他的 API 接口来对其进行属性设置

2.2.5 删除任务

```
void lv_task_del(lv_task_t * task);
```

参数:

task: 任务句柄指针

当我们不需要某任务时,需要通过此接口来删除任务,释放在堆上占用的资源

2.2.6 设置任务回调函数

```
void lv_task_set_cb(lv_task_t * task, lv_task_cb_t task_cb);
```

参数:

task: 任务句柄指针

task_cb: 任务回调函数

这个接口,我们一般不常用,除非你的项目有动态修改回调函数的需求

2.2.7 设置任务优先级

```
void lv_task_set_prio(lv_task_t * task, lv_task_prio_t prio);
```

参数:

task: 任务句柄指针

prio: 任务优先级

这个接口,我们一般不常用,除非你的项目有动态修改任务优先级的需求

2.2.8 设置任务回调周期

```
void lv_task_set_period(lv_task_t * task, uint32_t period);
```

参数:

task: 任务句柄指针

period: 任务回调周期,单位 ms

这个接口,我们一般不常用,除非你的项目有动态修改任务回调周期的需求

2.2.9 使任务立即准备就绪

```
void lv_task_ready(lv_task_t * task);
```

参数:

task: 任务句柄指针

通过调用此 API 接口,我们可以使刚创建出来的任务立即处于就绪状态,然后在下一个 lv_task_handler 调用时,使任务回调函数立即得到运行,而不用去等它的第一个运行周期,注意了这里说的是第一个周期不用等了,但是后面的周期还是要等的

2.2.10 使任务回调函数只运行一次

```
void lv_task_once(lv_task_t * task);
```

参数:

task: 任务句柄指针

通过调用此 API 接口,我们可以让任务的回调函数只运行一次,而非周期性调用,在一次调用完成之后,littleVGL 内部会通过 lv_task_del 接口来自动删除此任务的

2.2.11 复位任务

```
void lv_task_reset(lv_task_t * task);
```

参数:

task: 任务句柄指针

举个例子来方便理解,假如任务 A 的回调周期是 1000ms,现在已经等待了 300ms,按理来说还需等待 700ms,任务 A 的回调函数才会被运行,但是如果此时使用 lv_task_reset 接口来复位任务 A 的话,那么之前等待的 300ms 会被作废,而是重新开始等待一个 1000ms 的完整周期

2.2.12 是否使能任务管理系统

```
void lv_task_enable(bool en);
```

参数:

en: true 是使能任务管理系统,false 是禁止任务管理系统

littleVGL 默认是使能了任务管理系统的,如下图所示:

```
51  /**
52   * Init the lv_task module
53   */
54  void lv_task_core_init(void)
55  {
56      lv_ll_init(&LV_GC_ROOT(_lv_task_ll), sizeof(lv_task_t));
57
58      /*Initially enable the lv_task handling*/
59      lv_task_enable(true);
60  }
61
```

图 2.2.12.1 默认使能了任务管理系统

按理来说,任务管理系统是必须得使能的,否则 littleVGL 的内部事务将会失效,比如界面就不会刷新了,控件的点击事件也会失效,所以千万不要使用这个 API 接口,除非你有特殊的需求

2.2.13 获取任务的空闲百分比

```
uint8_t lv_task_get_idle(void);
```

返回值:

返回任务管理系统的空闲百分比

注意这个空闲百分比仅仅只能代表 littleVGL 任务管理系统的一个空闲情况,并不能代表我们整个应用的空闲情况,这个 API 接口也没啥大作用,了解即可

3. 例程设计

3.1 功能简介

创建一个 task1 任务,设置此任务的回调周期为 5000ms,并携带一个用户自定义参数 user_data,在任务回调函数中,让蜂鸣器鸣叫提示,同时通过串口 1 打印信息,当按下 KEY0 键时,通过 lv_task_reset 接口让 task1 任务进行复位操作,当按下 KEY1 键时,通过 lv_task_ready 接口让 task1 任务立即处于就绪状态,当按下 KEY2 键时,通过 lv_task_del 接口删除掉 task1 任务

3.2 硬件设计

本例程所用到的硬件有:

- 1) 蜂鸣器
- 2) 串口 1
- 3) KEY0,KEY1,KEY2 按键

这些硬件模块的驱动程序可以从相应开发板上的其他实验中拷贝过来,以后除了特殊硬件驱动程序外,其他硬件模块的驱动程序不再解释说明

3.3 软件设计

我们在 GUI_APP 目录下新建 task_test.c 和 task_test.h 两个文件,然后把 task_test.c 添加进 Keil 项目的 GUI_APP 分组下,并同时 C/C++ 面板下的 Include Paths 中引入其 GUI_APP 的头文件搜索路径,以后除特殊情况外,像这种简单的文件创建,导入,添加头文件路径的操作,我就不过多叙述了,接下来我们直接看 task_test.c 的代码.

```
#include "task_test.h"
#include "lvgl.h"
#include "key.h"
#include "beep.h"
#include "usart.h"
#include "delay.h"
```

```
typedef struct{
    char name[20];
    u8 age;
}USER_DATA;
```

```
lv_task_t *task1 = NULL;//任务句柄指针
USER_DATA user_data = { //user_data 为用户自定义参数,数据结构一般为结构体
    .name = {"xiong jia yu"},
    .age = 25
```

```
};

//任务回调函数
void task1_cb(lv_task_t* task)
{
    USER_DATA* dat = (USER_DATA*)(task->user_data);//获取用户的自定义参数

    //打印 tick 时间(一个 tick 为 1ms)和用户自定义参数
    printf("task1_cb_tick:%d,name:%s,age:%d\r\n",lv_tick_get(),dat->name,dat->age);

    //蜂鸣器鸣叫 20ms 进行提示
    BEEP = 1;
    delay_ms(30);
    BEEP = 0;
}

//例程入口函数
void task_test_start()
{
    //创建 task1 任务
    task1 = lv_task_create(task1_cb,5000,LV_TASK_PRIO_MID,&user_data);
}

//按键处理
void key_handler()
{
    u8 key = KEY_Scan(0);

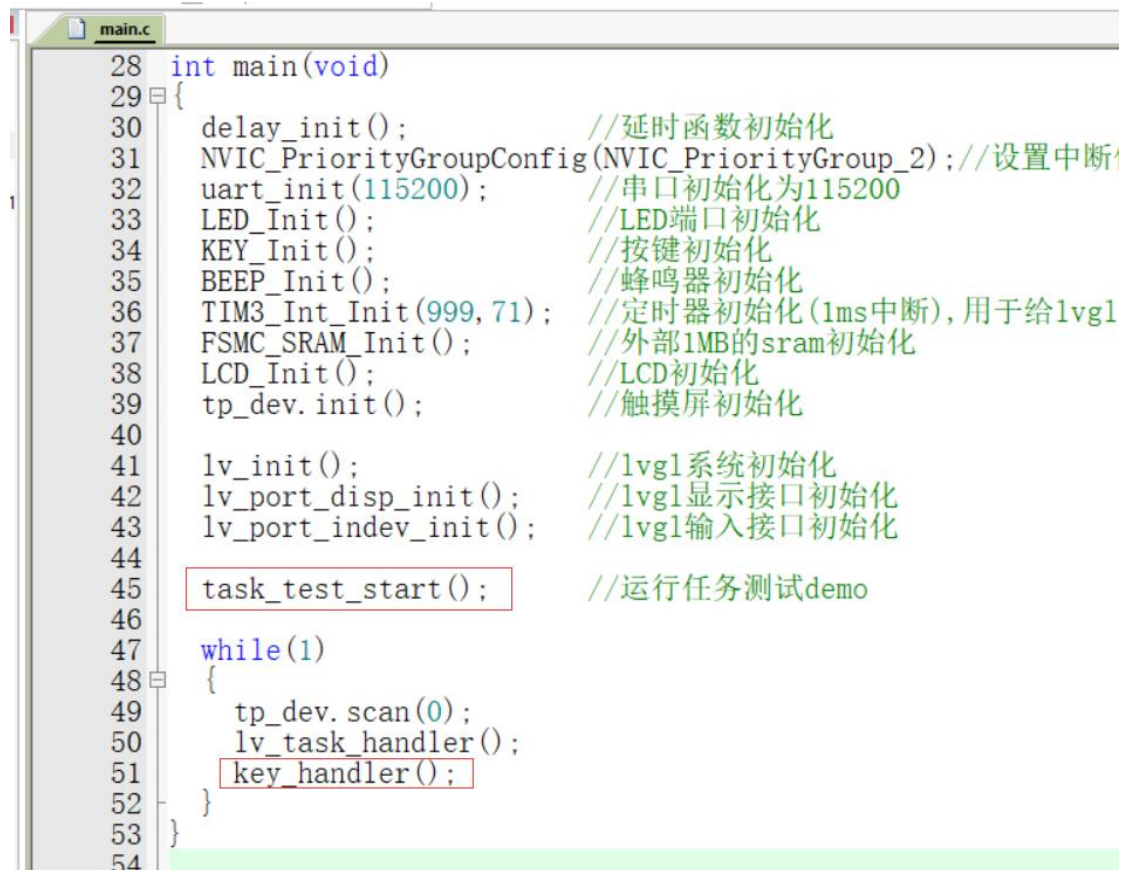
    if(task1==NULL)
        return;
    if(key==KEY0_PRES)
    {
        //使任务复位,如果你以固定的时间间隔不断的重复按下 KEY0 键(间隔时间要小
        //于 5000ms 的回调周期),你会发现 task1_cb 回调函数再也得不到运行了,因为
        //task1 任务在被重复性的复位,每一次复位将会导致重新等待一个完整的回调周期
        lv_task_reset(task1);
        printf("task_reset_tick:%d\r\n",lv_tick_get());
    }else if(key==KEY1_PRES)
    {
        //使任务立即准备就绪,当你按下 KEY1 键时,你会发现 task1_cb 回调函数会在下
        //一个 lv_task_handler 调用时被立即运行,通过串口打印,你会发现
        //task_ready_tick 的值比 task1_cb_tick 的值只小几个数
        lv_task_ready(task1);
        printf("task_ready_tick:%d\r\n",lv_tick_get());
    }
}
```

```

}else if(key==KEY2_PRES)//删除任务
{
//删除任务,当你按下 KEY2 键后,你会发现 task1_cb 回调函数将永远不会再被执行了
lv_task_del(task1);
task1 = NULL;
printf("task_del_tick:%d\r\n",lv_tick_get());
}
}

```

然后在 main 函数中调用 task_test_start 函数和 key_handler,如下图所示:



```

28 int main(void)
29 {
30     delay_init(); //延时函数初始化
31     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断
32     uart_init(115200); //串口初始化为115200
33     LED_Init(); //LED端口初始化
34     KEY_Init(); //按键初始化
35     BEEP_Init(); //蜂鸣器初始化
36     TIM3_Int_Init(999, 71); //定时器初始化(1ms中断),用于给lvgl
37     FSMC_SRAM_Init(); //外部1MB的sram初始化
38     LCD_Init(); //LCD初始化
39     tp_dev.init(); //触摸屏初始化
40
41     lv_init(); //lvgl系统初始化
42     lv_port_disp_init(); //lvgl显示接口初始化
43     lv_port_indev_init(); //lvgl输入接口初始化
44
45     task_test_start(); //运行任务测试demo
46
47     while(1)
48     {
49         tp_dev.scan(0);
50         lv_task_handler();
51         key_handler();
52     }
53 }
54

```

图 3.3.1 task_test_start 和 key_handler 调用

3.4 下载验证

当例程运行起来之后,如果什么按键也不按下的话,那么会听到蜂鸣器每隔 5 秒鸣叫一次,同时串口也会输出一信息,如下图所示:



图 3.4.1 每隔 5 秒输出一信息

当你以固定的时间间隔不断的重复按下 KEY0 键时(间隔时间要小于 5000ms 的回调周期),你会发现 task1_cb 回调函数再也得不到运行了,即串口无输出了,蜂鸣器也不鸣叫了,因为 task1 任务在被重复性的复位,而每一次复位都将会导致重新等待一个完整的回调周期

当你按下 KEY1 键时,你会发现 task1_cb 回调函数会在下一个 lv_task_handler 调用时立即运行,通过串口打印,你会发现 task_ready_tick 的值比 task1_cb_tick 的值只小几个数,如下图所示:

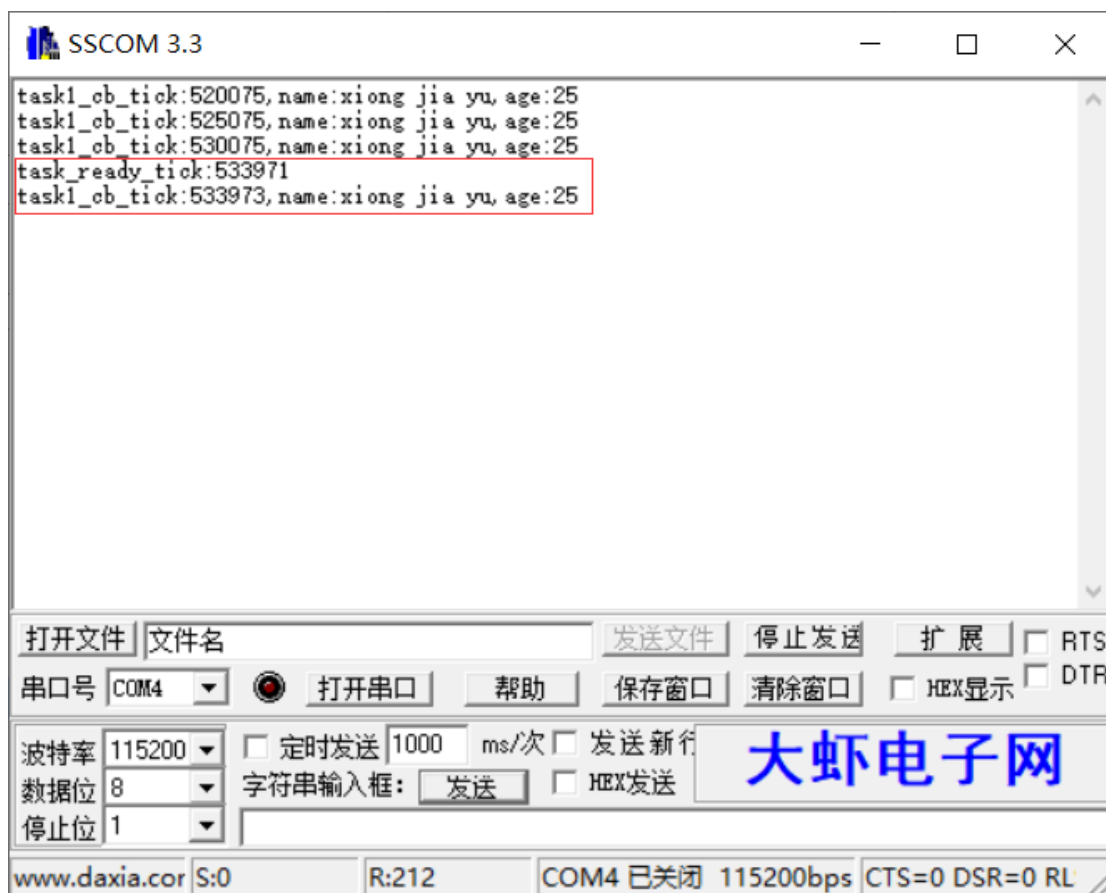


图 3.4.2 按下 KEY1 键时的串口输出

当你按下 KEY2 键后,你会发现 task1_cb 回调函数将永远不会再被执行了,因为任务被删除了

4. 资料下载

正点原子公司名称：广州市星翼电子科技有限公司

LittleVGL 资料连接：www.openedv.com/thread-309664-1-1.html

原子哥在线教学平台：www.yuanzige.com

正点原子淘宝店铺：<https://openedv.taobao.com>

正点原子官方网站：www.alientek.com

正点原子 B 站视频：<https://space.bilibili.com/394620890>

电话：020-38271790 传真：020-36773971

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

请关注正点原子公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子公众号