

正点原子 littleVGL 开发指南

littleVGL 移植

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

littleVGL 移植

1. littleVGL 介绍

littleVGL 可以说是这 2 年才刚刚开始流行的一个小型开源嵌入式 GUI 库,具有界面精美,消耗资源小,可移植度高,响应式布局等特点,全库采用纯 c 语言开发,在 2018 年初时,笔者刚开始接触到它,就被它的界面给吸引了,而且 littleVGL 库的更新速度非常快,从当初我刚接触的 V5.1 版本到现在的 V6.0 版本,这两个版本之间功能相差还是很大的,随着 littleVGL 的认知度越来越大,官方资料也逐渐丰富起来,但在国内而言,资源还是比较缺乏的,现在市面上还是缺乏一套完整而易上手的教程,因此我们正点原子团队基于此行情,特意推出 littleVGL 教程.

littleVGL 的官方网址为: <https://littlevgl.com>

littleVGL 的 github 网址为: <https://github.com/littlevgl/lvgl>

littleVGL 的在线文档网址为: <https://docs.littlevgl.com/zh-CN/html/index.html>

littleVGL 的主要特性如下:

- 具有非常丰富的内置控件,像 buttons, charts, lists, sliders, images 等
- 高级图形效果: 动画, 反锯齿, 透明度, 平滑滚动
- 支持多种输入设备,像 touchpad, mouse, keyboard, encoder 等
- 支持多语言的 UTF-8 编码
- 支持多个和多种显示设备, 例如同步显示在多个彩色屏或单色屏上
- 完全自定制的图形元素
- 硬件独立于任何微控制器或显示器
- 可以缩小到最小内存 (64 kB Flash, 16 kB RAM)
- 支持操作系统、外部储存和 GPU (非必须)
- 仅仅单个帧缓冲设备就可以呈现高级视觉特效
- 使用 C 编写以获得最大兼容性(兼容 C++)
- 支持 PC 模拟器
- 为加速 GUI 设计,提供教程,案例和主题,支持响应式布局
- 提供了在线和离线文档
- 基于自由和开源的 MIT 协议

littleVGL 的要求如下:

- 16、32 或 64 位的单片机 (微控制器) 或处理器
- 微处理器的主频最好高于 16MHz
- Flash/ROM:如果只用 littleVGL 核心组件的话,则至少需要 64kB 的容量,如果想完整使用的话,最好保证 180kB 以上的容量
- RAM:
 - 静态 RAM: 大约 8 到 16 kB,这取决于你所用的组件功能和 objects 控件对象类型
 - 栈: 至少为 2kB,一般推荐值为 4kB

- 动态数据(堆): 至少 4kB,如果你用到了多个或多种控件的话,那么最好设置为 16kB 以上,这个是可以通过 `lv_conf.h` 配置文件中的 `LV_MEM_SIZE` 宏来定义的
- 显示缓冲区: 至少要比"水平分辨率像素"要大,一般推介值为 10 倍的"水平分辨率像素",取个例子,假如我们屏幕的水平分辨率为 480 个像素,采用 16 位的颜色深度进行显示,即一个像素占 2 个字节,那么推介的显示缓冲区大小为 $10 \times 480 \times 2 = 9600$ 个字节
- C99 或更新的编译器,如果是用 keil 开发的话,一定得勾选"c99"模式,否则编译会报错的
- 基本的 c(或者 c++)语言知识,如:指针,结构体,回调函数



图 1.1 官方效果图

还有很多界面显示效果,请自行上 littleVGL 官网查看.我这里就不一一列举了

2. littleVGL 移植

先申明一下,本教程适用于正点原子的所有 stm32 开发板,支持正点原子的全系列液晶屏,不同的开发板对液晶屏型号的支持力度大小可能会不同,这个会在相应开发板的配套例程代码中进行说明,为了方便笔者描述,本文档中的图片和素材都以正点原子的“战舰”开发板为参考,不同开发板之间如有区别,笔者将会在文档中进行指出,let's go,让我们现在开始跨进 littleVGL 的大门,那就先从移植开始吧,我把整个移植过程罗列成以下步骤,请按照如下步骤一一操作.

2.1 步骤 1,准备素材

首先我们需要从 https://littlevgl.com/download/lv_pc_simulator.zip 链接上下载到 lv_pc_simulator.zip 压缩包,或者直接使用我们正点原子已经下载好的也行,此压缩包里面包含了 littleVGL 库源代码和 littleVGL 官方演示例程,接着我们还要准备一个 Keil 工程项目,这里我不打算重头新建一个 Keil 工程,因为我觉得这些简单的操作步骤,大家在学正点原子的 stm32 开发板时,肯定已经学会了,我这里就准备在正点原子的 stm32 开发板例程上直接移植,减少一些重复造轮子的工作,那么选择哪一个实验来移植呢,肯定是选择“**标准例程-库函数版本/触摸屏实验**”了,因为此实验里面包含了触摸和液晶屏显示的基本功能,我这里以战舰开发板为参考进行移植演示,其他的开发板操作流程也是一样的,不需要去担心,最后我们拿到了如下图所示的 2 个文件,把它们放到桌面上,方便操作



图 2.1.1 素材

2.2 步骤 2,修改 Keil 工程名

我是一个有强迫症的人,不能允许 Keil 工程名和实际项目含义不一致,于是我们先开始对“触摸屏实验”工程进行整理一下,主要是修改名称,删除无用的代码

我们是移植,所以先就取名为 template 吧,先把顶层目录名改为 template,接着进入到其 USER 子目录中把 TOUCH.uvprojx, TOUCH.uvoptx 相应的改为 template.uvprojx 和 template.uvoptx,接着删除掉 TOUCH.uvguix.Administrator 文件,接着双击 template.uvprojx 文

件打开 Keil 工程,点击  打开对话框,把 Targets 名称从 TOUCH 改为 TEMPLATE,接着点击  魔术棒,切到 Output 选项卡,把 Name of Executable 的内容改为 template,接着打开 main.c 文件,删除掉无用代码,最后只留 main 函数,如下图所示:

```

23
24 int main(void)
25 {
26     delay_init();           //延时函数初始化
27     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
28     uart_init(115200);    //串口初始化为115200
29
30     LED_Init();            //LED端口初始化
31     LCD_Init();
32     KEY_Init();
33     tp_dev.init();
34
35     while(1)
36     {
37
38     }
39 }
```

图 2.2.1 main 函数中的内容

接着我们编译一下,应该是没有任何错误和警告的

2.3 步骤 3,导入 littleVGL 库到 Keil 中

在 template 项目根目录下新建 GUI 和 GUI_APP 俩个子目录,即和 USER 目录是同级别的,GUI 目录是用来存放跟 littleVGL 库相关的所有文件的,而 GUI_APP 是用来放我们自己的 GUI 应用代码的,因为现在才刚刚开始移植,还来不及自己写 GUI 应用,所以 GUI_APP 目录里面先留空,我们的重点是来介绍 GUI 目录.

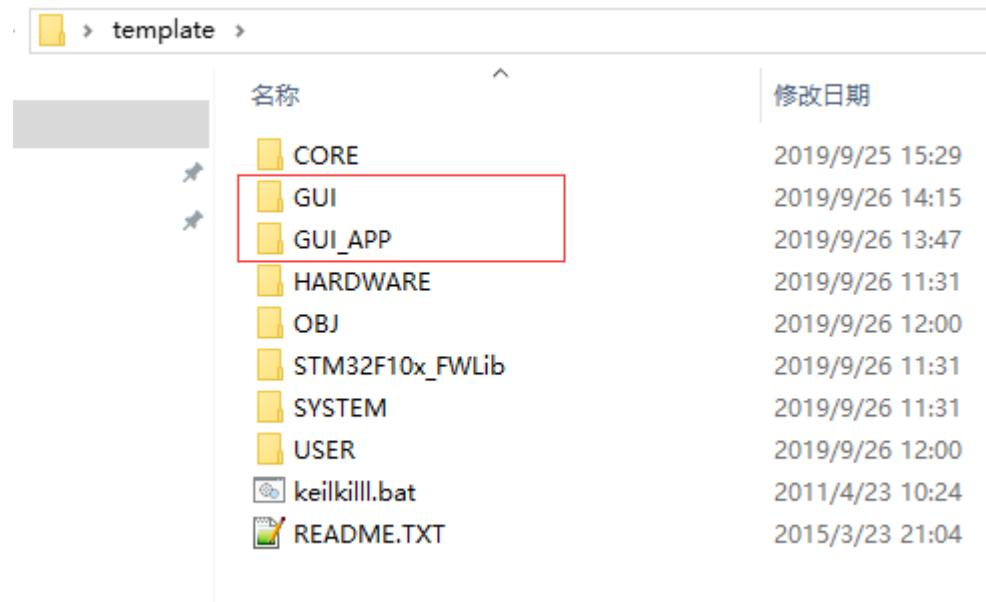


图 2.3.1 新建 GUI 和 GUI_APP 之后的目录结构

接着把 lv_pc_simulator.zip 压缩包里面的 lv_examples.zip 和 lvgl.zip 俩个子压缩包直接拷贝到 template/GUI 目录下,拷贝完成之后,接着分别对 lv_examples.zip 和 lvgl.zip 俩个子压缩包在当前目录下进行解压缩操作,解压缩完成后,可以把 lv_examples.zip 和 lvgl.zip 都删除了,接着把 template/GUI/lvgl/lv_conf_template.h 和 template/GUI/lv_examples/lv_ex_conf_template.h 俩个配置模板文件统统拷贝到 template/GUI 目录下,然后对这个 2 文件分别重命名为 lv_conf.h

和 lv_ex_conf.h,接着还要在 template/GUI 目录下新建一个 lvgl_driver 子目录,这个目录是用来放底层显示驱动和触摸驱动文件的,最后 template/GUI 的目录结构如下图所示.

名称	修改日期
lv_examples	2019/7/2 13:12
lvgl	2019/7/8 19:10
lvgl_driver	2019/9/26 14:26
lv_conf.h	2019/7/8 19:10
lv_ex_conf.h	2019/7/2 13:12

图 2.3.2 GUI 目录结构

注:在实际项目中,可以删除掉 lv_examples 目录来减少项目所占的磁盘空间,因为此目录就是专门用来存放官方的演示 demo,对我们的实际项目没有任何作用

接着我们打开 Keil 工程,点击 图标,打开分组管理面板,在 Groups 栏下新建 GUI 和 GUI_APP 俩个分组,选中 GUI 分组,接着点击 Add Files 按钮,把

```
template\GUI\lvgl\src\lv_core  
template\GUI\lvgl\src\lv_draw  
template\GUI\lvgl\src\lv_font  
template\GUI\lvgl\src\lv_hal  
template\GUI\lvgl\src\lv_misc  
template\GUI\lvgl\src\lv_objx  
template\GUI\lvgl\src\lv_themes
```

所有目录下的所有.c 文件依次全部添加到 GUI 分组下面,最后如下图所示

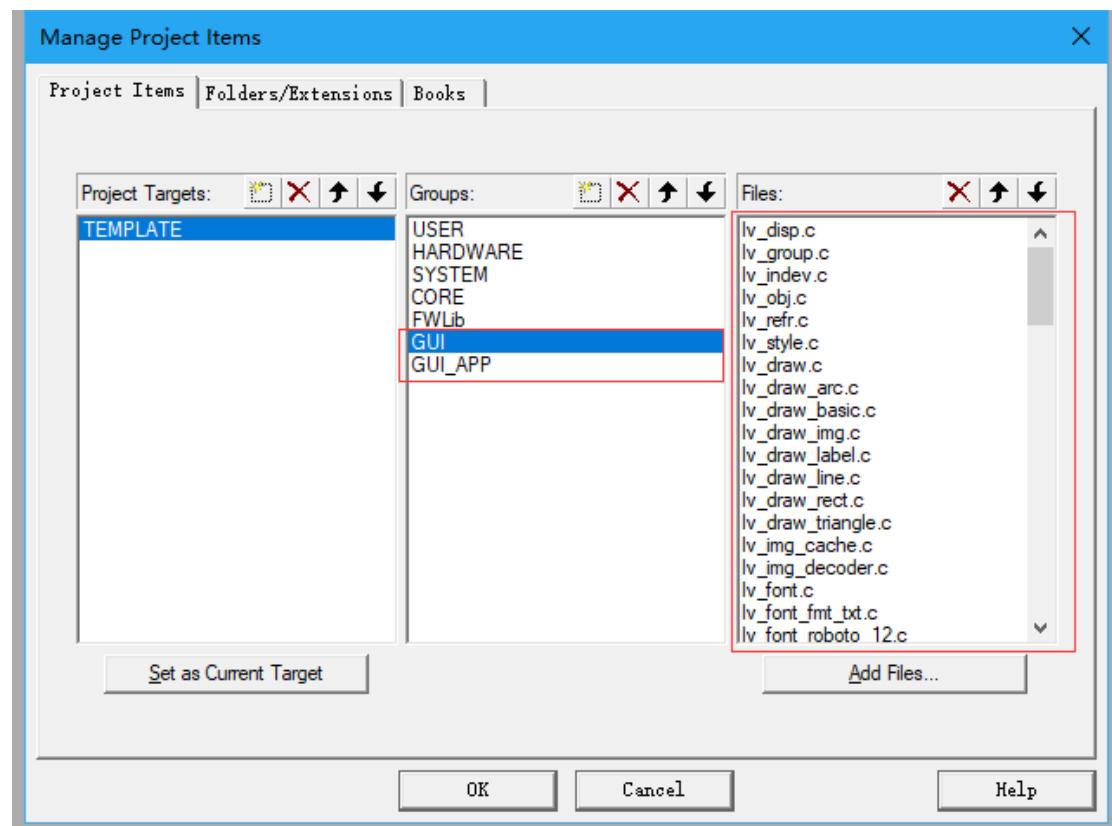


图 2.3.3 GUI 分组

最后点击 OK 按钮保存.

接着我们需要做 2 个比较重要的小操作,加大项目的栈空间到 2KB 和使能 C99 编译器功能,打开 Core 分组下的.s 启动文件,修改 Stack_Size 的值到 2KB(0x00000800),如

Stack_Size EQU 0x00000800 所示,然后点击 图标,打开面板之后,切换到 C/C++面板,选中 C99 Mode 复选框即可,因为 littleVGL 要求 C99 或更新的编译器,否则编译是会报错的。

2.4 步骤 4,修改 lv_conf.h 和 lv_ex_conf.h 配置文件

先打开 template\GUI 下的 lv_conf.h 文件,里面的配置项还是很多的,大家先不要纠结每个配置项到底什么意思,后面我会专门来讲解的,大家先按照我的步骤来,对于关键的配置项,我会先简单解释一下.

打开文件之后,第一个#if 后面的 0 改为 1,使整个文件生效,接着修改 **LV_HOR_RES_MAX** 和 **LV_VER_RES_MAX** 宏的值,这个是告诉 littleVGL 你所用的液晶屏分辨率是多少,请根据自己手头液晶屏的实际分辨率大小相应设置,比如笔者手头就有如下 4 块不同型号的 TFTLCD 液晶屏,分别是 **2.8 寸(320*240)电阻屏**, **3.5 寸(480*320)电阻屏**, **4.3 寸(800*480)电容屏**, **7 寸(800*480)电容屏**,在这个 4 块 MCU 屏中,分辨率最高的为 800*480,因此我这里就设置如下:

```
#define LV_HOR_RES_MAX      (480)
#define LV_VER_RES_MAX      (800)
```

因为这样就可以一套代码兼容 4 块不分分辨率的屏了,原理就是高分辨率可以兼容低分辨率.

接着我们修改 `LV_COLOR_DEPTH` 颜色深度,最常见的设置就是 1 或者 16 了,1 是用于单色屏,而 16 是用于彩色屏,这里我们设置成 16 即可,即如下所示

```
#define LV_COLOR_DEPTH      16
```

再接着我们来设置 `LV_DPI` 的值,默认值为 100,我们把他设置到 60,这个宏是用来调节界面缩放比例的,此值越大,控件分布的就越散,控件自身的间隔也会变大

```
#define LV_DPI              60
```

再接着修改 `LV_MEM_SIZE` 的大小,这个就是控制 littleVGL 中所谓的动态数据堆的大小,是用来给控件的创建动态分配空间的,我们这里设置为 16KB 的大小

```
#define LV_MEM_SIZE    (16U * 1024U)
```

再接着修改 `LV_USE_GPU` 的值,默认值是 1,我们把它设置为 0,即不使能 GPU 功能

```
#define LV_USE_GPU        0
```

再接着修改 `LV_USE_FILESYSTEM` 的值,其默认值为 1,我们把他设置为 0,即不使能文件系统的功能

```
#define LV_USE_FILESYSTEM 0
```

再接着再把 `LV_THEME_LIVE_UPDATE`, `LV_USE_THEME_TEMPL`,
`LV_USE_THEME_DEFAULT`,
`LV_USE_THEME_ALIEN`, `LV_USE_THEME_NIGHT`, `LV_USE_THEME_MONO`,
`LV_USE_THEME_MATERIAL`, `LV_USE_THEME_ZEN`, `LV_USE_THEME_NEMO` 等所有宏的值都设置为 1,即全部使能,这些宏都是跟 littleVGL 自带的主题相关的,因为后面我们要演示官方自带的例程效果,所以这里我们先全部使能,注意,在实际项目中,我们一般最多使能一个,如果我们项目根本就用不到其自带的主题,那么我们应该把这些宏全部禁止,因为这样可以节省 flash 和 ram。

至此 `lv_conf.h` 文件修改就完成了,接下来我们要修改 `lv_ex_conf.h` 文件,这个文件就简单很多了,而且这个文件也不是很重要,只有当我们要演示官方自带的例程时,才会用到,下面就简单做一下说明.

把 `LV_EX_KEYBOARD`, `LV_EX_MOUSEWHEEL`, `LV_USE_TESTS`, `LV_USE_TUTORIALS`,
`LV_USE_BENCHMARK`, `LV_USE_DEMO`, `LV_USE_TERMINAL` 等宏的值全部设置为 1, 其他宏保持默认即可。

2.5 步骤 5,添加定时器,为 littleVGL 提供心跳节拍

这里我打算采用定时器 3,设置其每隔 1ms 进入中断,为 littleVGL 提供 1ms 的心跳节拍,当然你也可以采用其他的定时器,原理都是一样的,定时器的代码可以直接从 stm32 开发板的”定时器中断实验”例程中拷贝过来,然后复制到 `template\HARDWARE` 目录下面,接着打开 Keil 工程,把 `timer.c` 文件添加到 `HARDWARE` 分组下面,接着修改 `timer.c` 文件中定时器中断服务函数,正确的如下所示:

```
#include "lvgl.h"  
void TIM3_IRQHandler(void)  
{  
    if(TIM3->SR&TIM_IT_Update)//溢出中断  
    {  
        lv_tick_inc(1);//lvgl 的 1ms 心跳  
    }  
    TIM3->SR = (uint16_t)~TIM_IT_Update; //清除中断标志  
}
```

然后在 `main` 函数中加入定时器的 `TIM3_Init(arr, psc)` 初始化代码,必须保证中断间隔为

1ms,不同开发板,传入的 arr, psc 参数可能会不同的,请注意!

以战舰开发板为例,此时的 main 函数代码差不多如下所示:

```
int main(void)
{
    delay_init(); //延时函数初始化
    //设置中断优先级分组为组 2: 2 位抢占优先级, 2 位响应优先级
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    uart_init(115200); //串口初始化为 115200
    LED_Init(); //LED 端口初始化
    KEY_Init(); //按键初始化
    TIM3_Int_Init(999,71); //定时器初始化(1ms 中断),用于给 lvgl 提供 1ms 的心跳节拍
    LCD_Init(); //LCD 初始化
    tp_dev.init(); //触摸屏初始化

    lv_init(); //lvgl 系统初始化

    while(1)
    {
        tp_dev.scan(0);
        lv_task_handler();
    }
}
```

如果你的定时器代码是采用标准库或者 HAL 库开发的话,那么你还需要把定时器对应的
标准库或 HAL 库驱动文件添加到 Keil 工程中.

接着我们还需要在 Keil 中配置 littleVGL 的头文件路径,点击  图标,面板打开之后,切

换到 C/C++ 面板,点击  图标,打开 Include Paths 面板, 添加..\\HARDWARE\\TIMER,
..\\GUI , ..\\GUI\\lvgl , ..\\GUI\\lv_examples, ..\\GUI\\lvgl_driver 几个头文件路径,如下图所示:

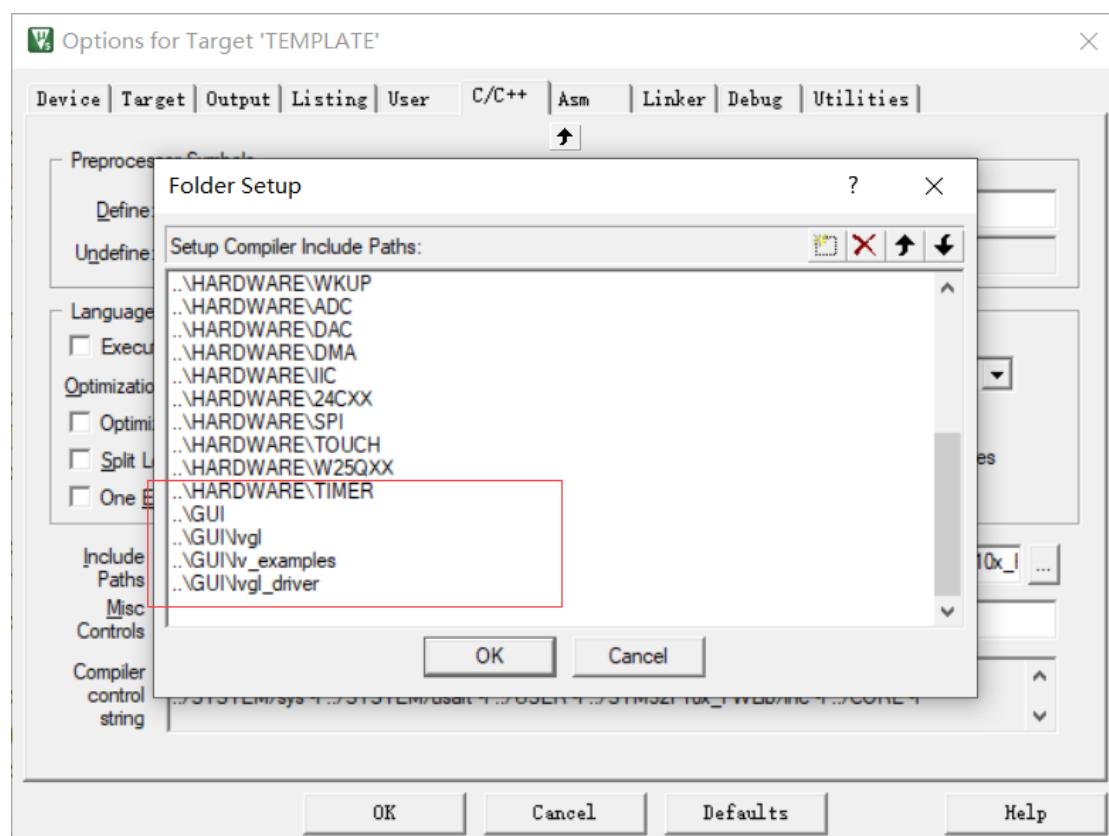


图 2.5.1 头文件路径添加

确认无误之后,哈哈,我们终于迎来了第一次编译,不出意外的话,会报一个错误和一百多个警告,错误提示应该是”..\SYSTEM\uart\uart.c(48): error: #260-D: explicit type is missing (“int” assumed)”,这个很好解决,打开 SYSTEM\uart\uart.c 文件,在 _sys_exit 函数的前面加入 void 返回值,再编译一下,就只剩下警告,没有错误了,接下来我们来解决警告,这一百多个警告中,仔细看其实就只有 68, 111, 550 这三种警告,我可以告诉大家,这个警告对我们项目没有任何影响的,但是强迫症患者看着就是难受,幸亏 Keil 可以通过设置,把某种警告给屏蔽掉,点

击 图标,切换到 C/C++选项卡,在 Misc Controls 中填入 **--diag_suppress=68 --diag_suppress=111 --diag_suppress=550**,如下图所示.

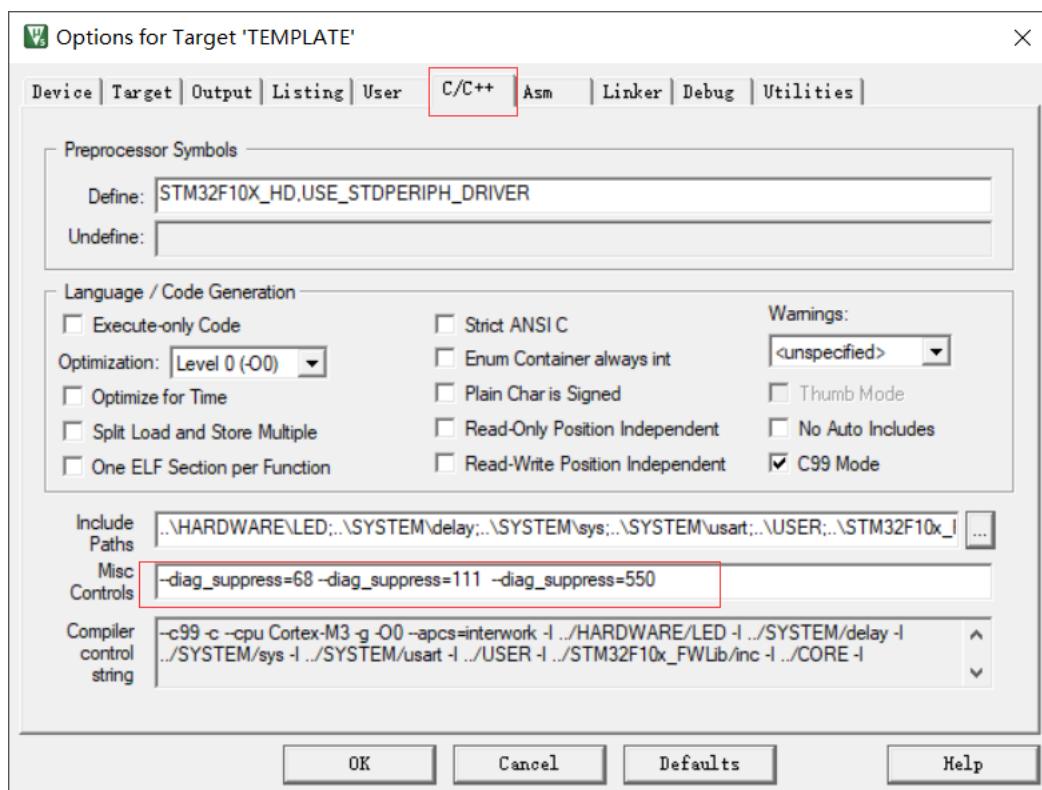


图 2.5.1 屏蔽警告

点击 OK 保存按钮之后,再来编译一下,就会发现没有任何错误和警告了.接下来我们要开始移植底层显示驱动了

2.6 步骤 6,移植底层显示驱动

littleVGL 官方给我们提供了显示驱动,输入驱动,文件系统驱动的模板文件,存放在 template\GUI\lvgl\porting 目录下,这里我们不需要文件系统,所以只需要把 lv_port_disp_template.c, lv_port_disp_template.h, lv_port_indev_template.c, lv_port_indev_template.h 四个文件拷贝到 template\GUI\lvgl_driver 目录下面,并分别重命名为 lv_port_disp.c, lv_port_disp.h, lv_port_indev.c, lv_port_indev.h,前面 2 个文件是跟显示驱动相关的,后面俩个文件是跟触摸驱动相关的,我们先只需要修改 lv_port_disp.c, lv_port_disp.h 俩个文件的内容就可以了,通过看注释,发现并不难移植,我这里直接给出正确的移植代码.

lv_port_disp.h 文件内容如下:

```
#ifndef LV_PORT_DISP_H
#define LV_PORT_DISP_H

#ifndef __cplusplus
extern "C" {
#endif

#include "lvgl/lvgl.h"
```

```
//函数申明
void lv_port_disp_init(void);

#ifndef __cplusplus
}
#endif

#endif
```

lv_port_disp.c 文件内容如下:

```
#include "lv_port_disp.h"
#include "lcd.h"

//函数申明
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p);
#if LV_USE_GPU
    static void gpu_blend(lv_color_t * dest, const lv_color_t * src, uint32_t length, lv_opa_t opa);
    static void gpu_fill(lv_color_t * dest, uint32_t length, lv_color_t color);
#endif

//lvgl 显示接口初始化
void lv_port_disp_init(void)
{
    static lv_disp_buf_t disp_buf;
    static lv_color_t color_buf[LV_HOR_RES_MAX * 10];//显示缓冲区,静态的 sram
    //显示缓冲区初始化
    lv_disp_buf_init(&disp_buf, color_buf, NULL, LV_HOR_RES_MAX * 10);

    //显示驱动默认值初始化
    lv_disp_drv_t disp_drv;
    lv_disp_drv_init(&disp_drv);

    //设置屏幕的显示大小,我这里是为了支持正点原子的多个屏幕,采用动态获取的方式
    //如果你是用于实际项目的话,可以不用设置,那么其默认值就是 lv_conf.h 中
    //LV_HOR_RES_MAX 和 LV_VER_RES_MAX 宏定义的值
    disp_drv.hor_res = lcddev.width;
    disp_drv.ver_res = lcddev.height;
```

```
//注册显示驱动回调
disp_drv.flush_cb = disp_flush;

//注册显示缓冲区
disp_drv.buffer = &disp_buf;

#if LV_USE_GPU
    //可选的,只要当使用到 GPU 时,才需要实现 gpu_blend 和 gpu_fill 接口

    //使用透明度混合俩个颜色数组时需要用到 gpu_blend 接口
    disp_drv.gpu_blend = gpu_blend;

    //用一个颜色填充一个内存数组时需要用到 gpu_fill 接口
    disp_drv.gpu_fill = gpu_fill;
#endif

    //注册显示驱动到 lvgl 中
    lv_disp_drv_register(&disp_drv);
}

//把指定区域的显示缓冲区内容写入到屏幕上,你可以使用 DMA 或者其他的硬件加速器
//在后台去完成这个操作但是在完成之后,你必须得调用 lv_disp_flush_ready()
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p)
{
    //把指定区域的显示缓冲区内容写入到屏幕
    LCD_Color_Fill(area->x1,area->y1,area->x2,area->y2,(u16*)color_p);
    //最后必须得调用,通知 lvgl 库你已经 flushing 拷贝完成了
    lv_disp_flush_ready(disp_drv);
}

//可选的
#if LV_USE_GPU

/*
If your MCU has hardware accelerator (GPU) then you can use it to blend to memories
using opacity
* It can be used only in buffered mode (LV_VDB_SIZE != 0 in lv_conf.h)
*/
static void gpu_blend(lv_disp_drv_t * disp_drv, lv_color_t * dest, const lv_color_t * src,
```

```
uint32_t length, lv_opa_t opa)
{
    /*It's an example code which should be done by your GPU*/
    uint32_t i;
    for(i = 0; i < length; i++) {
        dest[i] = lv_color_mix(dest[i], src[i], opa);
    }
}

/* If your MCU has hardware accelerator (GPU) then you can use it to fill a memory with a
color
 * It can be used only in buffered mode (LV_VDB_SIZE != 0 in lv_conf.h)*/
static void gpu_fill_cb(lv_disp_drv_t * disp_drv, lv_color_t * dest_buf, lv_coord_t dest_width,
                        const lv_area_t * fill_area, lv_color_t color);
{
    /*It's an example code which should be done by your GPU*/
    uint32_t x, y;
    dest_buf += dest_width * fill_area->y1; /*Go to the first line*/

    for(y = fill_area->y1; y < fill_area->y2; y++) {
        for(x = fill_area->x1; x < fill_area->x2; x++) {
            dest_buf[x] = color;
        }
        dest_buf += dest_width;      /*Go to the next line*/
    }
}

#endif
```

我们的 GUI 应用跑的流不流畅,主要是取决 lv_port_disp.c 文件了,这里需要注意一点,为了移植方便,我们的 color_buf 显示缓冲区是采用静态数组的方式定义的,优点是内部 sram 访问速度快,缺点是不能定义的过大,因为内部的 sram 少的可怜,后面的章节中我会介绍怎么把 color_buf 显示缓冲区定义到外部的大容量 sram 中,从而来提升 GUI 的流畅度!

2.7 步骤 7,移植底层触摸驱动

littleVGL 是支持很多种输入设备的,像 Touchpad, Mouse, Keypad, Encoder, Button 等统统支持,而通常情况下,我们用的最多的就是触摸屏了,他属于 Touchpad 类,通过打开 template\GUI\lvgl_driver 目录下的 lv_port_indev.c 和 lv_port_indev.h,文件,不难发现只需要实现 lv_port_indev_init 和 touchpad_read 值个 API 接口就行了,其他的统统可以删除,下面我给出正确的移植代码.

lv_port_indev.h 文件内容如下：

```
#ifndef LV_PORT_INDEV_H
#define LV_PORT_INDEV_H

#ifndef __cplusplus
extern "C" {
#endif

#include "lvgl/lvgl.h"

//函数申明
void lv_port_indev_init(void);

#ifndef __cplusplus
}
#endif

#endif
```

lv_port_indev.c 文件的内容如下：

```
#include "lv_port_indev.h"
#include "touch.h"

//函数申明
static bool touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data);

//lvgl 的输入设备初始化
void lv_port_indev_init(void)
{
    lv_indev_drv_t indev_drv;

    //lvgl 支持很多种输入设备,但是我们一般常用的就是触摸屏,也就是 Touchpad
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
```

```
lv_indev_drv_register(&indev_drv);
}

//将会被lvgl周期性调用,周期值是通过lv_conf.h中的LV_INDEV_DEF_READ_PERIOD
//宏来定义的
//此值不要设置的太大,否则会感觉触摸不灵敏,默认值为30ms
static bool touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
    static uint16_t last_x = 0;
    static uint16_t last_y = 0;

    if(tp_dev.sta&TP_PRES_DOWN)//触摸按下了
    {
        last_x = tp_dev.x[0];
        last_y = tp_dev.y[0];
        data->point.x = last_x;
        data->point.y = last_y;
        data->state = LV_INDEV_STATE_PR;
    }else{//触摸松开了
        data->point.x = last_x;
        data->point.y = last_y;
        data->state = LV_INDEV_STATE_REL;
    }

    //返回 false 代表没有缓冲的数据
    return false;
}
```

2.8 步骤 8,移植官方的演示例程

终于熬到最后一步了,我们只要把官方的演示例程代码添加到 Keil 工程中的 GUI_APP 分组下就可以了,官方的例程源码全部位于 template\GUI\lv_examples 目录下,这里我们先演示 2 个综合一点的例程.

demo 例程,其源码路径为: template\GUI\lv_examples\lv_apps\demo\
theme 例程,其源码路径为: template\GUI\lv_examples\lv_tests\lv_test_theme\
其中 theme 例程里面又包含了 2 个子例程

我们把 demo 和 theme 例程按照上面所描述的源码路径添加到 Keil 中的 GUI_APP 分组下,如图所示:

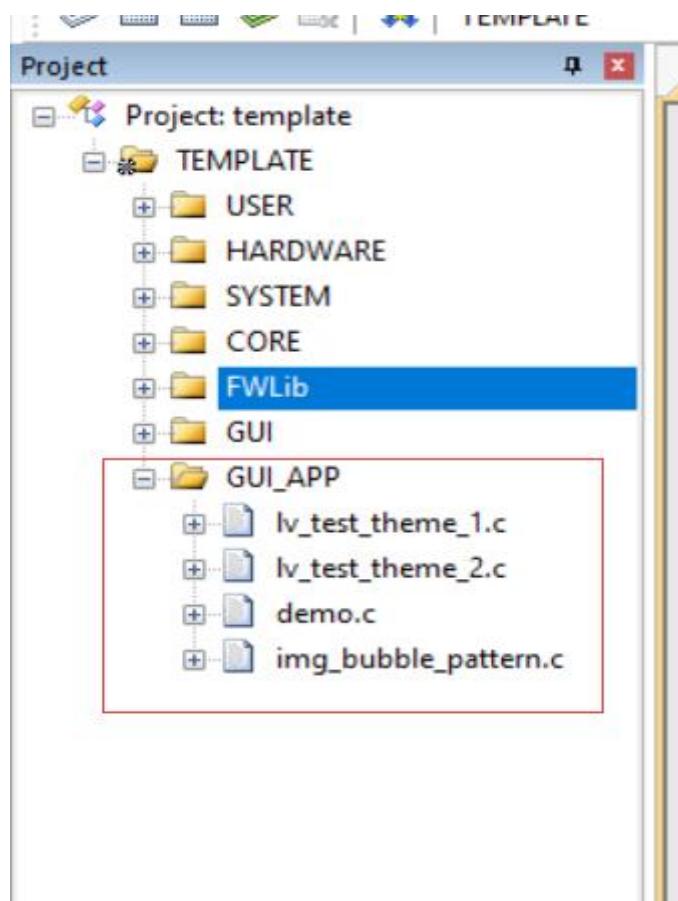


图 2.8.1 GUI_APP 分组

然后在 main 函数中添加显示驱动初始化和触摸驱动初始化的代码, 总之最后 main.c 文件的代码如下:

```
#include "led.h"
#include "delay.h"
#include "key.h"
#include "sys.h"
#include "lcd.h"
#include "uart.h"
#include "touch.h"
#include "timer.h"
#include "lvgl.h"
#include "lv_port_disp.h"
#include "lv_port_indev.h"
#include "lv_apps\demo\demo.h"
#include "lv_tests\lv_test_theme\lv_test_theme_1.h"
#include "lv_tests\lv_test_theme\lv_test_theme_2.h"

#define TEST_NUM      1 //1,2,3 分别对应三个演示例程
```

```
int main(void)
{
    delay_init();           //延时函数初始化
    //设置中断优先级分组为组 2: 2 位抢占优先级, 2 位响应优先级
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    uart_init(115200);      //串口初始化为 115200
    LED_Init();             //LED 端口初始化
    KEY_Init();              //按键初始化
    TIM3_Int_Init(999,71); //定时器初始化(1ms 中断),用于给 lvgl 提供 1ms 的心跳节拍
    LCD_Init();              //LCD 初始化,一定得放在 lv_init()的前面

    tp_dev.init();           //触摸屏初始化

    lv_init();                //lvgl 系统初始化
    lv_port_disp_init();     //lvgl 显示接口初始化,放在 lv_init()的后面
    lv_port_indev_init();    //lvgl 输入接口初始化,放在 lv_init()的后面

    //通过 TEST_NUM 的值来选择运行不同的例程
    #if(TEST_NUM==1)
        demo_create();
    #elif(TEST_NUM==2)
        lv_test_theme_1(lv_theme_night_init(210, NULL));
    #else
        lv_test_theme_2();
    #endif
    while(1)
    {
        tp_dev.scan(0);
        lv_task_handler();
    }
}
```

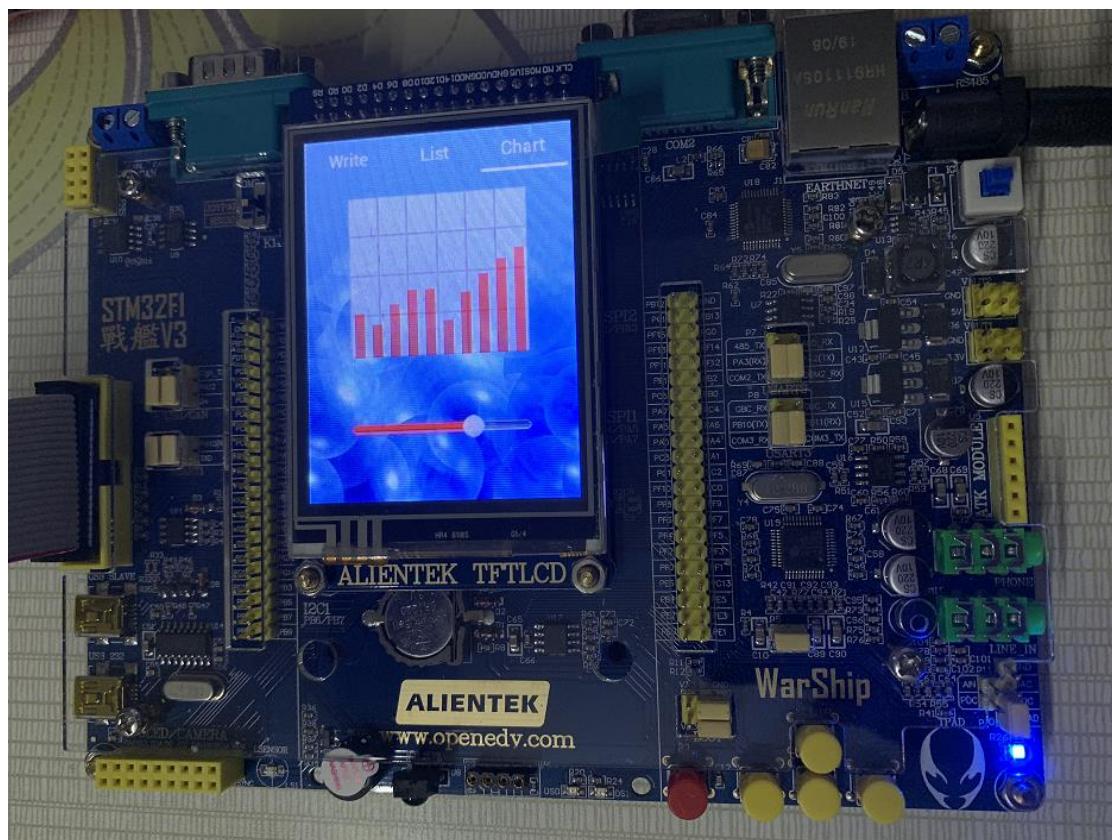
当运行 `demo_create()` 例程时,编译会发现报错的,这是因为官方的 `demo` 例程有一个小疏忽,我们打开 `GUI_APP` 分组下的 `demo.c` 文件,把 `a.user_data = NULL;` 的代码给删除掉就行了,总共有 2 处,分别是在 215 行和 249 行.

因为我们的代码是支持电阻屏的,而电阻屏一般都需要先进行校准操作,所以我在 `touch.c` 文件的 `TP_Init` 函数中加入了一个小操作,那就是在开机之前如果先按住 `KEY0` 键不放,就可以先进入到电阻屏校准程序,校准完成之后,再进入到 littleVGL 演示例程,代码很简单,那就是直接把 `if(TP_Get_Adjdata())` 改成 `if(KEY0==1)&&TP_Get_Adjdata()` 就可以了,其他的地方不需要改动

至此 littleVGL 的移植到此结束了,祝大家旅途愉快!

3. littleVGL 演示效果

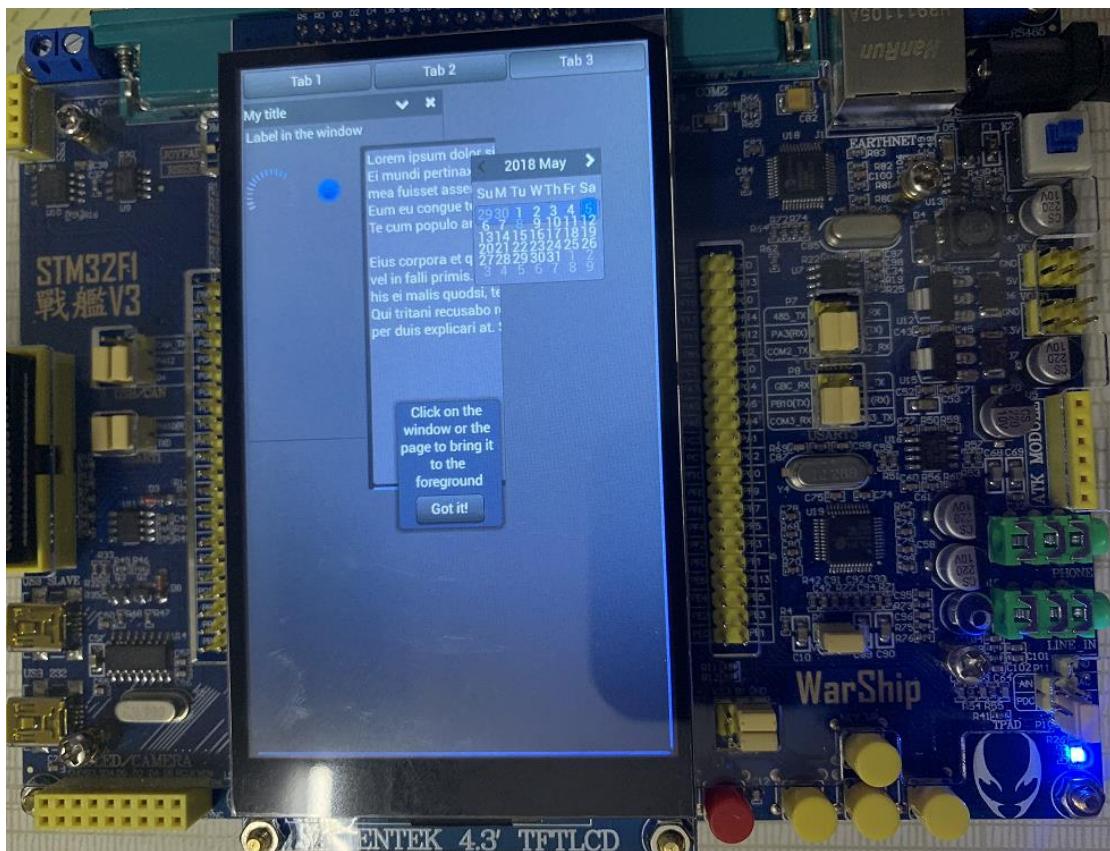
3.1 2.8 寸(320*240) 电阻屏演示效果



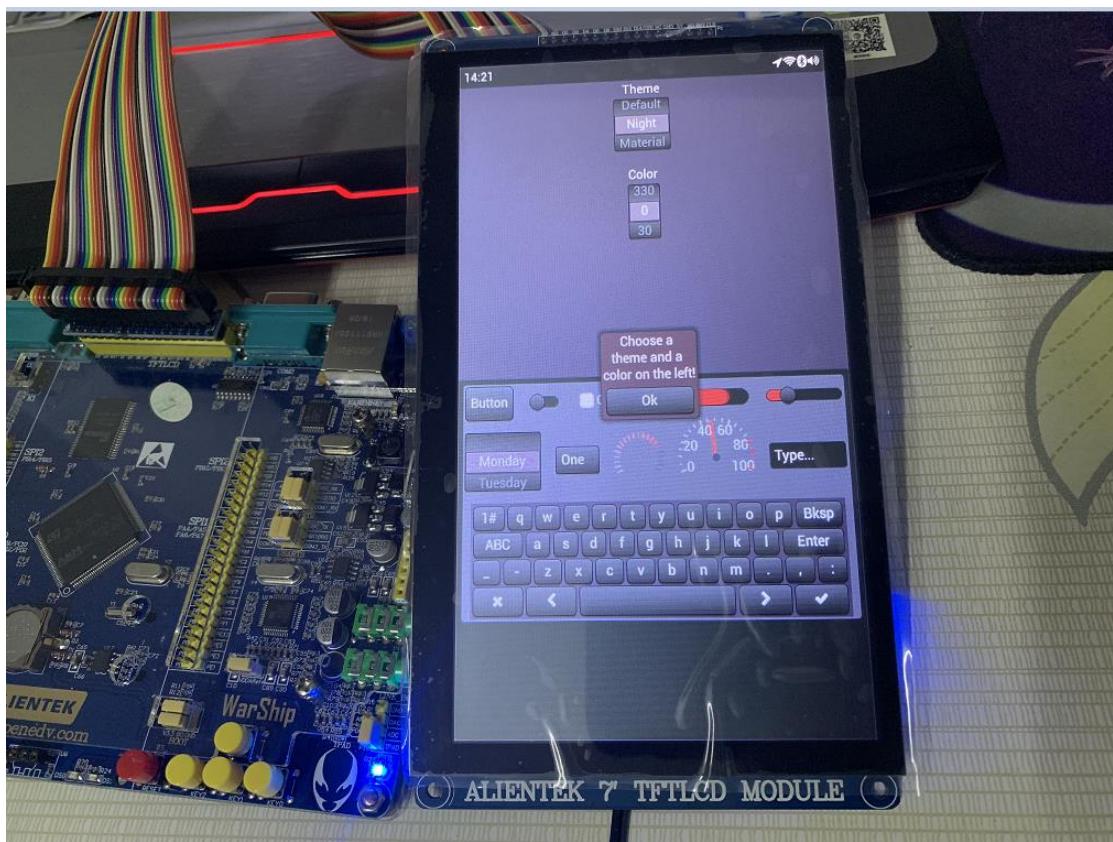
3.2 3.5 寸(480*320) 电阻屏演示效果



3.3 4.3寸(800*480)电容屏



3.4 7寸(800*480)电容屏



3.5 总结

从上面的演示效果可以看出, littleVGL 的响应式布局确实很牛呀, 在不同分辨率的屏幕上跑, 界面不会变形, 当然了 littleVGL 支持的屏幕型号远不止上面列举的 4 种, 不同的正点原子开发板对液晶屏型号的支持力度大小是不同的, 而且运行的流畅度也是不同的, 下一节, 我将给大家介绍如果用外部的大容量 sram 给 littleVGL 增加运行流畅度

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

使用外部 SARM 进行加速

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

使用外部 sram 进行加速

1.介绍

在我们的 littleVGL 移植章节中,为了减少移植的难度,我们是采用处理器内部的 sram 给 littleVGL 的显示缓冲区分配了一个 $10 * LV_HOR_RES_MAX$ 小空间,而在这一节中,我们将通过使用外部的大容量 sram 来给 GUI 应用增加流畅度,当然前提是你的开发板已经配备了外部 sram 模块,为了能让 littleVGL GUI 增加运行流畅度,其影响因素还是很多的,我这里主要列出如下 4 个主要的原因:

1. 使用性能更好的处理器
2. 使用 GPU 或者 DMA 进行 littleVGL 显示缓冲区的拷贝操作
3. 尽量使用处理器内部的 sram
4. 给 littleVGL 的显示缓冲区分配更大的空间,最好全屏大小的空间

在条件有限的情况下,第 1 点和第 2 点我们就不考虑了,我们都知道处理器内部的 sram 访问速度很快,但是容量却少的可怜,当然了如果你的处理器内部 sram 容量很大的话,那么你可以直接用内部 sram 给 littleVGL 分配一个全屏大小的显示缓冲区,就没必要再去使用外部 sram 了,虽然外部 sram 访问速度比内部 sram 慢,但是外部 sram 可以给 littleVGL 分配更大以至全屏的显示缓冲区,在这俩者的 PK 权衡之下,实验证明,使用外部 sram 是可以增加 littleVGL 的运行流畅度的.下面开始详细描述如何去实现这一功能

2.软件设计

本章节的代码改动不大,所以可以在移植那一节的代码基础上进行稍微修改.

2.1 添加外部 sram 驱动

我们可以直接从正点原子相应开发板上的”外部 SRAM 实验”中,把 sram 驱动程序直接拷贝到我们项目的 HARDWARE 目录下,如下图所示:

template > HARDWARE			
名称	修改日期	类型	
24CXX	2019/9/25 14:36	文件夹	
IIC	2019/9/25 14:36	文件夹	
KEY	2019/9/25 14:36	文件夹	
LCD	2019/9/25 14:36	文件夹	
LED	2019/9/25 14:36	文件夹	
SPI	2019/9/25 14:36	文件夹	
SRAM	2019/9/26 9:33	文件夹	
TIMER	2019/9/25 15:38	文件夹	
TOUCH	2019/9/25 19:09	文件夹	
W25QXX	2019/9/25 14:36	文件夹	

图 2.1.1 SRAM 驱动文件的位置

然后打开本项目的 Keil 工程,并在 HARDWARE 分组下添加 sram 的驱动文件,如下图所示:

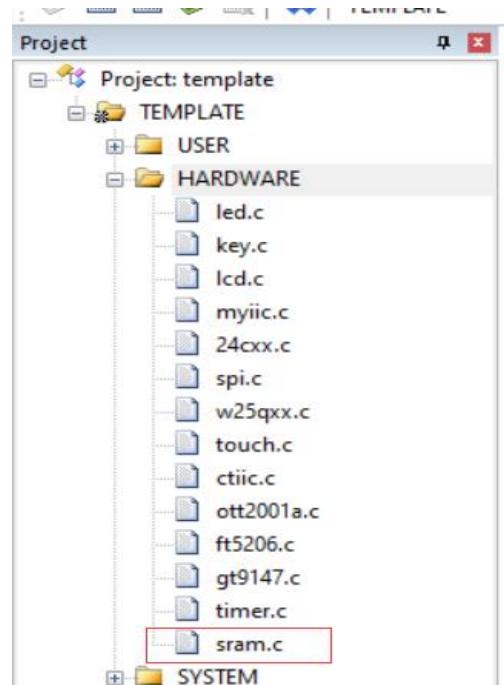


图 2.1.2 分组下添加 sram 的驱动程序

接着也需要注意在 Includes Paths 中添加 sram 驱动的头文件路劲,如下图所示:

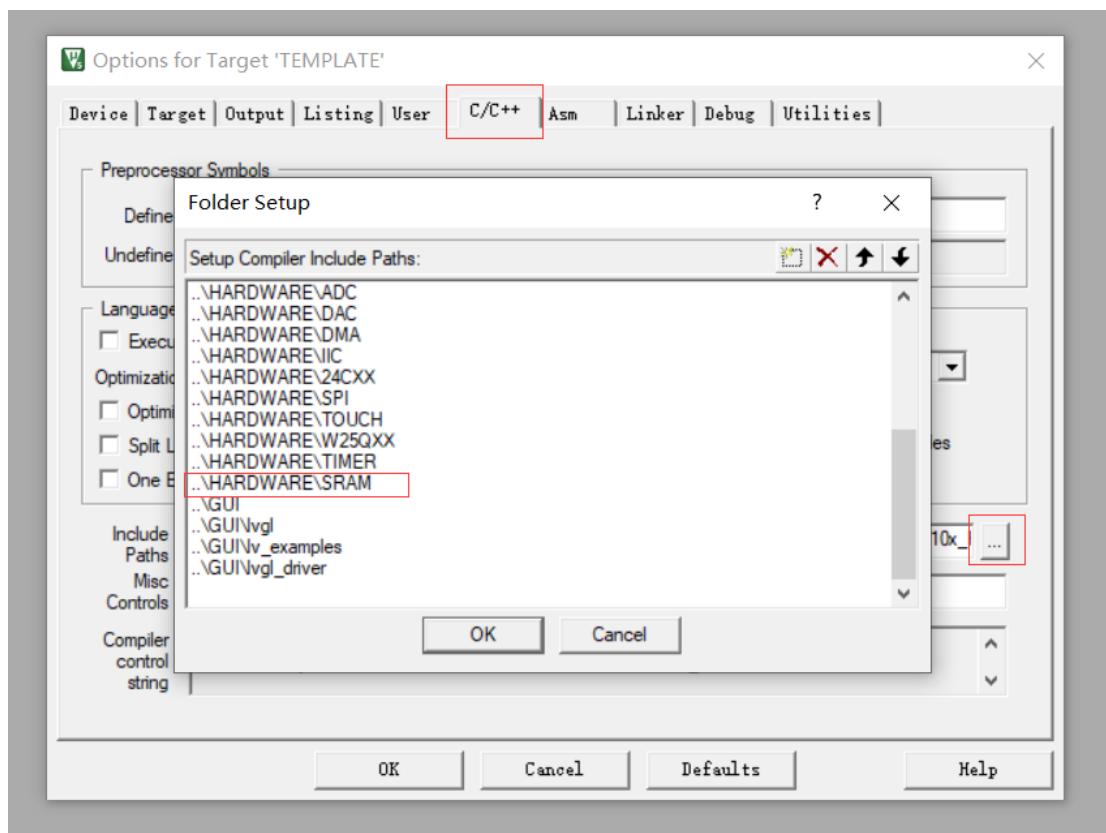


图 2.1.3 添加 sram 驱动的头文件路径

接着再到 main 函数中调用外部 sram 的初始化代码,如下图所示:

```
1 int main(void)
2 {
3     delay_init();          //延时函数初始化
4     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断优先级
5     uart_init(115200);    //串口初始化为115200
6     LED_Init();           //LED端口初始化
7     KEY_Init();           //按键初始化
8     TIM3_Int_Init(999, 71); //定时器初始化(1ms中断),用于给lvg1提供1ms时钟
9     FSMC_SRAM_Init(); //外部1MB的sram初始化
10    LCD_Init();           //LCD初始化
11    tp_dev.init();        //触摸屏初始化
12
13    lvg1_init();           //lvg1系统初始化
14    lvg1_port_disp_init(); //lvg1显示接口初始化
15    lvg1_port_indev_init(); //lvg1输入接口初始化
16 }
```

图 2.1.4 main 函数中调用外部 sram 的初始化代码

注:以后像这种添加硬件驱动的简单操作,我后面的章节中就不再详细描述

2.2 修改 lv_port_disp.c 文件

我们在 keil 中打开 lv_port_disp.c 文件,在这个文件中,我们修改的内容很少,主要就是更换显示缓存区的定义方式,之前是在内部 sram 中分配的,以前的代码是下面这样的:

```
void lv_port_disp_init(void)
{
    ...
    static lv_color_t buf1_1[LV_HOR_RES_MAX * 10];
    lv_disp_buf_init(&disp_buf_1, buf1_1, NULL, LV_HOR_RES_MAX * 10);
    ...
}
```

而现在我们需要把改成如下这样的代码:

```
//全屏的大小
#define COLOR_BUF_SIZE      (LV_HOR_RES_MAX*LV_VER_RES_MAX)
//分配到外部
static lv_color_t color_buf[COLOR_BUF_SIZE] __attribute__((at(0X68000000)));

void lv_port_disp_init(void)
{
    ...
    lv_disp_buf_init(&disp_buf, color_buf, NULL, COLOR_BUF_SIZE);
    ...
}
```

其中为了见名知意,显示缓存区的名字被我从 buf1_1 改为了 color_buf,,上面的 0X68000000 是外部 sram 的地址,这个请你根据你的项目来实际调整,,只要保证外部 sram 剩余的空间大小大于 COLOR_BUF_SIZE*2 个字节就行了,笔者为强迫症患者,所以还删除掉了 lv_port_disp.c 文件中一些无用的函数,比如 disp_init 函数就是没有用的,最后我给出 lv_port_disp.c 文件改完后的一个完整代码:

```
#include "lv_port_disp.h"
#include "lcd.h"

//变量定义
//全屏的大小
#define COLOR_BUF_SIZE      (LV_HOR_RES_MAX*LV_VER_RES_MAX)
//分配到外部 1MB sram 的最起始处
static lv_color_t color_buf[COLOR_BUF_SIZE] __attribute__((at(0X68000000)));

//函数申明
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p);
#if LV_USE_GPU
```

```
static void gpu_blend(lv_color_t * dest, const lv_color_t * src, uint32_t length, lv_opa_t opa);
static void gpu_fill(lv_color_t * dest, uint32_t length, lv_color_t color);
#endif

//lvgl 显示接口初始化
void lv_port_disp_init(void)
{
    static lv_disp_buf_t disp_buf;

    //显示缓冲区初始化
    lv_disp_buf_init(&disp_buf, color_buf, NULL, COLOR_BUF_SIZE);

    //显示驱动默认值初始化
    lv_disp_drv_t disp_drv;
    lv_disp_drv_init(&disp_drv);

    //设置屏幕的显示大小,我这里是为了支持正点原子的多个屏幕,采用动态获取的方式,如果你是用于实际项目的话,可以不用设置,那么其默认值就是 lv_conf.h 中 //LV_HOR_RES_MAX 和 LV_VER_RES_MAX 宏定义的值
    disp_drv.hor_res = lcddev.width;
    disp_drv.ver_res = lcddev.height;

    //注册显示驱动回调
    disp_drv.flush_cb = disp_flush;

    //注册显示缓冲区
    disp_drv.buffer = &disp_buf;

#if LV_USE_GPU
    //可选的,只要当使用到 GPU 时,才需要实现 gpu_blend 和 gpu_fill 接口

    //使用透明度混合俩个颜色数组时需要用到 gpu_blend 接口
    disp_drv.gpu_blend = gpu_blend;

    //用一个颜色填充一个内存数组时需要用到 gpu_fill 接口
    disp_drv.gpu_fill = gpu_fill;
#endif

    //注册显示驱动到 lvgl 中
    lv_disp_drv_register(&disp_drv);
}
```

//把指定区域的显示缓冲区内容写入到屏幕上,你可以使用 DMA 或者其他的硬件加速器在后台去完成这个操作

//但是在完成之后,你必须得调用 lv_disp_flush_ready()

```
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p)
{
    //把指定区域的显示缓冲区内容写入到屏幕
    LCD_Color_Fill(area->x1,area->y1,area->x2,area->y2,(u16*)color_p);
    //最后必须得调用,通知 lvgl 库你已经 flushing 拷贝完成了
    lv_disp_flush_ready(disp_drv);
}
```

//可选的

```
#if LV_USE_GPU
```

```
/* If your MCU has hardware accelerator (GPU) then you can use it to blend to memories
using opacity
```

```
* It can be used only in buffered mode (LV_VDB_SIZE != 0 in lv_conf.h)*/
```

```
static void gpu_blend(lv_disp_drv_t * disp_drv, lv_color_t * dest, const lv_color_t * src,
uint32_t length, lv_opa_t opa)
```

```
{
    /*It's an example code which should be done by your GPU*/
    uint32_t i;
    for(i = 0; i < length; i++) {
        dest[i] = lv_color_mix(dest[i], src[i], opa);
    }
}
```

```
/* If your MCU has hardware accelerator (GPU) then you can use it to fill a memory with a
color
```

```
* It can be used only in buffered mode (LV_VDB_SIZE != 0 in lv_conf.h)*/
```

```
static void gpu_fill_cb(lv_disp_drv_t * disp_drv, lv_color_t * dest_buf, lv_coord_t
dest_width,
```

```
const lv_area_t * fill_area, lv_color_t color);
```

```
{
    /*It's an example code which should be done by your GPU*/
    uint32_t x, y;
    dest_buf += dest_width * fill_area->y1; /*Go to the first line*/

    for(y = fill_area->y1; y < fill_area->y2; y++) {
        for(x = fill_area->x1; x < fill_area->x2; x++) {
            dest_buf[x] = color;
```

```
    }
    dest_buf+=dest_width;      /*Go to the next line*/
}
#endif
```

按照上述改完后,直接编译,应该是无错误无警告的,把代码下载到开发板之后,可以对比发现会比”移植章节” 中的演示效果流畅很多.

3.littleVGL 堆的内存分配

littleVGL 的内存消耗主要体现在 2 个方面,第一个是显示缓冲区,第二个就是我们这里所要讲到的堆,而 littleVGL 堆的内存分配也是有 2 种方式,如下所示:

- 1)采用内部的 sram,原理定义一个静态的局部数组
- 2)和显示缓冲区一样,采用外部的大容量 sram

我们一般情况下都是使用第一种方式的,即采用内部的 sram,因为内部的 sram 速度快,而且 littleVGL 堆的内存消耗一般都不是很大的,对于市面上的大部分微处理器来说,是完全可以满足这个性能的,那什么时候我们去使用第二种方式呢?那就是当你的微处理器的内部 sram 真的是特别少的时候,当然了咯,前提是你的硬件已经具备了外部 sram,如果想要采用第二种方式,针对于软件层面来说是非常简单的,只需要把 lv_conf.h 配置文件中的 LV_MEM_ADR 宏指向我们的外部 sram 地址就行了,其他的什么都不需要改动,具体示例如下:

```
#define LV_MEM_SIZE      (16U * 1024U) //此时是指占用外部 sram 的大小  
#define LV_MEM_ADR       (0X68000000+LV_HOR_RES_MAX*LV_VER_RES_MAX*2)
```

至于 LV_MEM_ADR 的值请根据自己项目实际情况来改动

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_conf 配置文件详解

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

lv_conf 配置文件详解

对于 littleVGL 而言,它的功能和特性基本都是由 lv_conf.h 文件来配置的,所以 lv_conf.h 文件的重要性就不言而喻了,此文件位于项目根目录的 GUI 目录下,给 lv_conf.h 文件进行合理的配置,可以对我们项目的性能和资源起到很大的作用 ,下面让我们对其配置项逐一讲解.

1. 定义最大的液晶屏分辨率

和此功能相关的配置项有 2 个,也就是所谓的 2 个宏定义,分别为: **LV_HOR_RES_MAX**, **LV_VER_RES_MAX**

利用这俩个配置项,再加上 LCD_Display_Dir(dir)函数可以实现屏幕的横屏和竖屏

1.1 LV_HOR_RES_MAX 配置项

LV 是 littleVGL 的缩写,HOR 是 horizontal 的缩写,为水平的意思,RES 是 resolution 的缩写,为分辨率的意思,组合起来之后,就很容易猜到这是用于设置液晶屏水平长度的,单位为像素

1.2 LV_VER_RES_MAX 配置项

LV 是 littleVGL 的缩写,VER 是 vertical 的缩写,为垂直的意思,RES 是 resolution 的缩写,为分辨率的意思,组合起来之后,就很容易猜到这是用于设置液晶屏垂直长度的,单位为像素

2. 定义颜色深度

和此功能相关的配置项有 2 个,为 **LV_COLOR_DEPTH**, **LV_COLOR_16_SWAP**

2.1 LV_COLOR_DEPTH

littleVGL 支持 4 种颜色深度,格式分为为 1 byte per pixel, RGB233, RGB565, ARGB8888, 其对应的配置项值分别为 1, 8, 16, 32,在一般实际项目中,最常用的为 1 和 16,当然如果你的处理器性能和资源都很高的话,那么你可以选择 32,占用 4 个字节,他会给你的项目带来更逼真的颜色效果,保证不失真,如果你选择的是 1,那么它只支持 2 种颜色,占用 1 个字节,在液晶屏上的表现就是显示与不显示的关系,常用于单色屏,比如 lcd12864,oled 等,当你选择的是 16 时,那么他支持 65536 种颜色,占用 2 个字节,显示效果还是很不错的,同时对处理器的要求也不是很高,因此 16 成为了我们实际项目中最常用的设置值.

2.2 LV_COLOR_16_SWAP

这个配置项只有在你选择颜色深度为 16 位时才会生效,在原本颜色值的基础上进行交换高低字节的操作,当你的屏幕有一个 8 位的并行接口时,可能会用到,一般不用理会

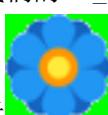
3 定义图片显示时的透明色

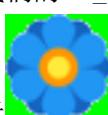
和此功能相关的配置项只有 1 个,为 **LV_COLOR_TRANSP**,它的默认值为 **LV_COLOR_LIME** 宏.而此宏的定义如下:

```
#define LV_COLOR_LIME    LV_COLOR_MAKE(0x00, 0xFF, 0x00)
```

也就是说 **LV_COLOR_LIME** 为纯绿色, **LV_COLOR_TRANSP** 宏只有当我们使用到 image 相关的控件时, 才会有可能起到作用, 注意了, 我这里只是说有可能, 因为 littleVGL 想要显示带有透明效果的图片, 有 2 种方式来实现, 分别为 **Chrome keying** 和 **Alpha byte**, 只有当我们在”**Online image converter**” 图片在线转换工具中选择带有 **Chrome keying** 的颜色格式时, **LV_COLOR_TRANSP** 就会起作用, 具体作用就是当图片中的颜色值和 **LV_COLOR_TRANSP** 指定的颜色值相等时, 此颜色值像素点就不会被绘制出来, 从而达到了透明的效果.

举个例子, 假如我们的 **LV_COLOR_TRANS** 不改动, 即默认采用 **LV_COLOR_LIME** 纯绿色为



透明色, 当我们要显示  这张带有纯绿色的背景图片时, 液晶屏上的实际效果是如下



这样的:  , 因为纯绿色的背景没有被绘制, 从而实现了透明显示的效果, 当然这种方式也是有一定局限性的, 因为它实现不了半透明的效果, 它只能全透明或者完全不透明

4.是否使能抗锯齿功能

和此功能相关的配置项只有 1 个,为 **LV_ANTIALIAS**,它的默认值为 1,即默认是使能抗锯齿的,当设置为 0 时就禁止了抗锯齿功能,实际项目中,一般都是使能的,这样我们 UI 界面上的文字,线条,圆弧等元素就看不到毛刺现象了

5.定义屏幕的刷新周期

和此功能相关的配置项只有 1 个,为 **LV_DISP_DEF_REFR_PERIOD**,它的默认值为 30ms,此值设置的过大的话,就可能会出现卡顿的现象,设置的过小的话,就会有点浪费性能,我们直接采用默认值就可以了,不用过多理会

6.定义界面的缩放比例

和此功能相关的配置项只有 1 个,为 **LV_DPI**,它的默认值为 100,因为 littleVGL 是支持响应式布局的,通过调节此参数的值,可以很好的适应不同分辨率的布局,表面看起来就跟放大缩小了一样,其实本质是调节了 littleVGL 样式的内边距和外边距等,当此值越大时,控件所占的内边距和外边距值也就越大,导致整个控件所占据的位置也就越大,从而表现出了放大的效果,当此值越小时,表现效果与上面相反.

7. 定义 littleVGL 的内存管理方式

和此功能相关的配置项有 8 个,为 `LV_MEM_CUSTOM`, `LV_MEM_SIZE`,
`LV_MEM_ATTR`,`LV_MEM_ATTR`,`LV_MEM_AUTO_DEFRAG`,`LV_MEM_CUSTOM_INCLUDE`,
`LV_MEM_CUSTOM_ALLOC`,`LV_MEM_CUSTOM_FREE`

7.1 LV_MEM_CUSTOM

此配置项是用来选择使用 littleVGL 自带的内存管理还是使用自己自定义的内存管理,当其值为 0 时,是选择内部自带的,当其值为 1 时,是选择自定义的方式,默认值为 0 ,为了减少麻烦,我们直接使用内部自带的就可以了.

7.2 LV_MEM_ATTR

当 `LV_MEM_CUSTOM` 为 0 时,此配置项才生效,当 `LV_MEM_ATTR` 为 0 时,代表所管理的内存是直接定义在处理器内部 sram 的,也就是一个内部静态数组,当 `LV_MEM_ATTR` 为其他地址值时,代表所管理的内存是定义在外部扩展的 sram 上,通过打开 littleVGL 的 `lv_mem.c` 文件,找到 `lv_mem_init` 函数,可以看到如下定义:

```
#if LV_MEM_ATTR == 0
    /*Allocate a large array to store the dynamically allocated data*/
    static LV_MEM_ATTR MEM_UNIT work_mem_int[LV_MEM_SIZE / sizeof(MEM_UNIT)];
    work_mem = (uint8_t *)work_mem_int;
#else
    work_mem = (uint8_t *)LV_MEM_ATTR;
#endif
```

因为所管理的内存池一般不会很大,一般的应用 16KB-32KB 就可以满足了,所以我们一般都把 `LV_MEM_ATTR` 的值设置为 0,即直接定义在处理器内部的 sram 上,这样会比外部的 sram 读写速度快很多

7.3 LV_MEM_SIZE

当 `LV_MEM_CUSTOM` 为 0 而且 `LV_MEM_ATTR` 也为 0 时,此配置项才生效,用来设置所管理内存的大小,单位为字节,至少大于 2KB,通过打开 littleVGL 的 `lv_mem.c` 文件,找到 `lv_mem_init` 函数,可以看到如下定义:

```
static LV_MEM_ATTR MEM_UNIT work_mem_int[LV_MEM_SIZE / sizeof(MEM_UNIT)];
```

work_mem_int 就是函数内部定义的静态数组,也就是所被管理的内存

7.4 LV_MEM_ATTR

当 `LV_MEM_CUSTOM` 为 0 而且 `LV_MEM_SIZE` 也为 0 时,此配置项才生效,是用来修饰所被管理的内存,通过打开 littleVGL 的 `lv_mem.c` 文件,找到 `lv_mem_init` 函数,可以看到如下定义:

```
static LV_MEM_ATTR MEM_UNIT work_mem_int[LV_MEM_SIZE / sizeof(MEM_UNIT)];
```

比如说用来设置字节对齐：

```
#define LV_MEM_ATTR __align(8) //以 8 字节对齐
```

7.5 LV_MEM_AUTO_DEFrag

当 LV_MEM_CUSTOM 为 0 时, 此配置项才生效, 代表在空闲时候, 是否自动进行内存碎片化整理, 1 代表使能, 0 代表不使能, 我们默认使能即可

7.6 LV_MEM_CUSTOM_INCLUDE

当 LV_MEM_CUSTOM 为 1 时, 此配置项才生效, 用来指明你自己所实现的内存管理所在的头文件, 如下所示:

```
#define LV_MEM_CUSTOM_INCLUDE <stdlib.h>
```

7.7 LV_MEM_CUSTOM_ALLOC

当 LV_MEM_CUSTOM 为 1 时, 此配置项才生效, 用来指向你自己所实现的 malloc 函数, 如下所示:

```
#define LV_MEM_CUSTOM_ALLOC malloc
```

7.8 LV_MEM_CUSTOM_FREE

当 LV_MEM_CUSTOM 为 1 时, 此配置项才生效, 用来指向你自己所实现的 free 函数, 如下所示:

```
#define LV_MEM_CUSTOM_FREE free
```

8. 是否使能垃圾回收器

和此功能相关的配置项有 4 个, 为 **LV_ENABLE_GC**, **LV_GC_INCLUDE**,
LV_MEM_CUSTOM_REALLOC, **LV_MEM_CUSTOM_GET_SIZE**

LV_ENABLE_GC 是总开关, 当其值为 1 时, 是使能垃圾回收器, 一般只有当 littleVGL 绑定到其他的高级语言中时才有作用, 此时的 LV_GC_INCLUDE, LV_MEM_CUSTOM_REALLOC, LV_MEM_CUSTOM_GET_SIZE 三个配置项指向相应的头文件和函数, 当其值为 0 时, 是禁止, 我们一般用不到此功能的, 默认禁止就可以了

9. 定义输入设备的轮询周期

和此功能相关的配置项有 1 个, 为 **LV_INDEV_DEF_READ_PERIOD**, 其默认值为 30ms, 代表每隔 30ms 采样一次输入设备(比如键盘, 触摸屏等)的状态

10. 定义 Drag 拖拽动作的阈值

和此功能相关的配置项有 1 个, 为 `LV_INDEV_DEF_DRAG_LIMIT`, 其默认值为 10 个像素, 代表按住不放拖动 10 个像素以上才认为是一次有效的 Drag 拖拽动作

11. 定义 Drag 动作后, 控件停下来的速度

和此功能相关的配置项有 1 个, 为 `LV_INDEV_DEF_DRAG_THROW`, 其默认值为 20%, 此值越大的话, 那么拖动某个控件, 待松手后, 这个控件就会更快的停下来, 类似于汽车紧急刹车后漂移的一种效果

12. 定义长按事件的触发时间

和此功能相关的配置项有 1 个, 为 `LV_INDEV_DEF_LONG_PRESS_TIME`, 其默认值为 400ms, 代表按住某个控件 400ms 以上时, 就会触发这个控件的长按事件

13. 定义长按重复事件的触发时间

和此功能相关的配置项有 1 个, 为 `LV_INDEV_DEF_LONG_PRESS REP TIME`, 其默认值为 100ms, 代表当某个控件的长按事件触发之后, 再继续按住此控件 100ms 以上, 就会触发这个控件的长按重复事件

14. 是否使能动画特性

和此功能相关的配置项有 1 个, 为 `LV_USE_ANIMATION`, 其默认值为 1, 代表使能动画特性, 其值为 0 时, 代表禁止动画特性

15. 是否使能阴影效果

和此功能相关的配置项有 1 个, 为 `LV_USE_SHADOW`, 其默认值为 1, 代表使能动画特性, 其值为 0 时, 代表禁止动画特性

16.是否使能 GROUP 分组功能

和此功能相关的配置项有 1 个, 为 `LV_USE_GROUP`, 其默认值为 1, 代表使能分组功能, 常用于 keyboard 和 encoder 的导航场景, 其值为 0 时, 代表禁止

17.是否使用 GPU

和此功能相关的配置项有 1 个, 为 `LV_USE_GPU`, 其默认值为 1, 代表使用 GPU 进行加速, 那么然后你还要在 `lv_port_disp.c` 文件中实现 `gpu_blend` 和 `gpu_fill` 接口, 请根据自己的处理器是否具有 GPU 功能来相应的设置此配置项, 我们一般设置为 0, 即不使用 GPU 功能

18. 是否使用 littleVGL 自带的文件系统

和此功能相关的配置项有 1 个, 为 `LV_USE_FILESYSTEM`, 当其值为 1 时, 代表使用自带的文件系统, 那么然后你还要移植 `lv_port_fs_template.c` 和 `.h` 文件, 我们一般设置为 0, 即不使用自带的文件系统

19. 是否使用 user_data 子字段

和此功能相关的配置项有 1 个, 为 `LV_USE_USER_DATA`, 当其值为 1 时, 代表在 drivers 和 objects(就是控件)描述结构体上增加一个名为 `user_data` 的子字段, 主要是用来携带用户自己的数据, 当其值为 0 时, 就是不增加此字段, 请根据自己项目的需求来相应的设置, 在 `lv_obj.h` 中可以找到如下定义:

```
typedef struct _lv_obj_t
{
    struct _lv_obj_t * par;
    //中间省略掉...
#ifndef LV_USE_USER_DATA
    lv_obj_user_data_t user_data;
#endif
} lv_obj_t;
```

20.是否使能图片的调色板功能

和此功能相关的配置项有 1 个, 为 `LV_IMG_CF_INDEXED`, 当其值为 1 时, 代表支持图片的调色板方式显示, 也就是先把一张图片的所有颜色值存放到一个数组(调色板)中, 然后通过索引值来映射出颜色值, 好处就是节省图片的存储空间, 缺点就是调色板的元素个数不能太大, 一般最大为 256 色

21. 是否使能图片的 alpha 透明功能

和此功能相关的配置项有 1 个, 为 `LV_IMG_CF_ALPHA`, 其默认值为 1, 代表使能 alpha 透明功能, 0 代表不使能

22. 设置图片缓存的大小

和此功能相关的配置项有 1 个, 为 `LV_IMG_CACHE_DEF_SIZE`, 此值至少大于等于 1, 其默认值为 1, 此功能一般用在从外部存储器(如 SD 卡)显示图片的场景下, 比如用第三方的 PNG 或者 JPG 库来解析存储器上的. png 或. jpg 图片时

23. 定义 lv_tick_inc 函数的修饰属性

和此功能相关的配置项有 1 个, 为 `LV_ATTRIBUTE_TICK_INC`, 其默认值为空, 通过打开 `lv_hal_tick.c` 文件, 可以找到如下定义:

```
LV_ATTRIBUTE_TICK_INC void lv_tick_inc(uint32_t tick_period)
{
    tick_irq_flag = 0;
    sys_time += tick_period;
}
```

24. 定义 lv_task_handler 函数的修饰属性

和此功能相关的配置项有 1 个, 为 `LV_ATTRIBUTE_TASK_HANDLER`, 其默认值为空, 通过打开 `lv_task.c` 文件, 可以找到如下定义:

```
LV_ATTRIBUTE_TASK_HANDLER void lv_task_handler(void)
{
    //省略掉...
}
```

25. 显式定义内存对齐的大小

和此功能相关的配置项有 1 个, 为 `LV_ATTRIBUTE_MEM_ALIGN`, 其默认值为空, 因为当编译器使用-Os 进行优化时, 可能会导致我们的数据不是以 4 或者 8 字节对齐的, 此时我们可以显式定义进行字节对齐, 如下所示:

```
#define LV_ATTRIBUTE_MEM_ALIGN __attribute__((aligned(4)))
```

26. 定义常量关键字

和此功能相关的配置项有 1 个, 为 `LV_ATTRIBUTE_LARGE_CONST`, 其默认值为空, littleVGL 默认自带了 `const` 常量关键字, 如果你发现你的项目中不是用 `const` 的话, 那么你就可以来设置 `LV_ATTRIBUTE_LARGE_CONST` 的值, 如下所示(在 c51 中 `code` 为常量定义的关键字):

```
#define LV_ATTRIBUTE_LARGE_CONST code
```

27. 是否使能 log 日志模块

和此功能相关的配置项有 3 个, 为 `LV_USE_LOG`, `LV_LOG_LEVEL`, `LV_LOG_PRINTF`, 其中 `LV_USE_LOG` 是总开关, 代表是否使能 log 模块, 其实内部就是通过 `printf` 或自实现的 `print` 函数来实现打印日志信息的, 我们一般把 `LV_USE_LOG` 设置为 0, 即不使能

28. 是否使能 littleVGL 自带的主题

和此功能相关的配置项有 9 个, 为 `LV_THEME_LIVE_UPDATE`, `LV_USE_THEME_TEMPL`, `LV_USE_THEME_DEFAULT`, `LV_USE_THEME_ALIEN`, `LV_USE_THEME_NIGHT`, `LV_USE_THEME_MONO`, `LV_USE_THEME_MATERIAL`, `LV_USE_THEME_ZEN`, `LV_USE_THEME_NEMO`

其中 `LV_THEME_LIVE_UPDATE` 是用来是否允许在运行状态下切换主题风格的, 而其他的 8 个配置项表示是否使能其对应的 8 个主题风格

29. 是否使能 littleVGL 自带的字体

和此功能相关的配置项有 5 个, 为 `LV_FONT_ROBOTO_12`, `LV_FONT_ROBOTO_16`, `LV_FONT_ROBOTO_22`, `LV_FONT_ROBOTO_28`, `LV_FONT_UNICODE_8` 分别对应 12 号, 16 号, 22 号, 28 号的 ROBOTO 字体和 8 号的 UNICODE 字体, 这每一个字体都包含了 ASCII 和一些图标字体(Symbols), 而且全都是 4bpp 抗锯齿的, 设置为 1 就代表使能, 设置为 0 就代表不使能

30. 自定义字体申明

和此功能相关的配置项有 1 个, 为 `LV_FONT_CUSTOM_DECLARE`, 默认值为空, 当我们自己自定义了多种字体时, 就可以用此配置项来向外部进行申明, 如下面所示:

```
#define LV_FONT_CUSTOM_DECLARE LV_FONT_DECLARE(my_font_1) \
                                LV_FONT_DECLARE(my_font_2) \
                                LV_FONT_DECLARE(my_font_3)
```

其中 `LV_FONT_DECLARE` 是 littleVGL 用来申明字体的宏, 其定义如下:

```
#define LV_FONT_DECLARE(font_name) extern lv_font_t font_name;
```

31. 定义 littleVGL 默认的字体

和此功能相关的配置项有 1 个, 为 `LV_FONT_DEFAULT`, 当某些控件没有显式指定字体时, 那么此控件使用的就是默字体, 默认字体的定义方式如下:

```
#define LV_FONT_DEFAULT          &lv_font_roboto_16
```

当然了, 前提是你得保证 `LV_FONT_ROBOTO_16` 这个字体已经是使能了的

32. 是否使能大容量字体

和此功能相关的配置项有 1 个, 为 `LV_FONT_FMT_TXT_LARGE`, 其默认值为 0, 即不使能, 当我们自定义的某种字体里面包含了特别多的字符时(大于 10000 个), 如果显示出错了, 那么此时应该使能 `LV_FONT_FMT_TXT_LARGE` 来避免错误.

33. 定义文本的编码方式

和此功能相关的配置项有 1 个, 为 `LV_TXT_ENC`, 此配置项有 2 个可选值, 分别为 `LV_TXT_ENC_UTF8`(默认值) 和 `LV_TXT_ENC_ASCII`, 当我们要显示中文或者其他非 ascii 码字符时, 那么我们一定得设置成 `LV_TXT_ENC_UTF8`, 同时还得保证代码编辑器也是相应的 UTF8 编码

34. 是否使能 lv_obj_realign 函数

和此功能相关的配置项有 1 个, 为 `LV_USE_OBJ_REALIGN`, 其默认值为 1, 即使能 `lv_obj_realign` 函数, 设置为 0 的话, 就是不使能

35. 是否使能 littleVGL 的某个控件

介绍到这里时, `lv_conf.h` 文件基本介绍完了, 后面剩下的所有配置项基本都是一样的功能, 都是用来是否使能某个控件的, 举个例子:

```
#define LV_USE_BTN      1
```

意思就是使能按钮控件, 剩下的其他配置项我就不介绍了, 都是一样的道理

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

正点原子 littleVGL 开发指南

PC 模拟器的使用

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

PC 模拟器的使用

1. 介绍

PC 模拟器的好处就是在于当我们没有实际的硬件开发板时, 我们依然可以测试和调试 GUI 系统, 另外 PC 模拟器对我们的前期界面迭代和后期的产品说明书制作都是可以起到非常大的作用, 作为一款优秀的 GUI 库, littleVGL 也肯定是支持模拟器功能的, 模拟器配合不同的 IDE(Integrated Development Environments) 软件可以运行在不同的操作系统上, 比如 Windows, Linux or OSX, 下面是 littleVGL 所支持的 IDE 软件种类.

Eclipse	CodeBlocks	Visual Studio	PlatformIO	Qt Creator
				
Cross-platform with SDL	Native Windows	Cross-platform with SDL	Cross-platform with SDL	Cross-platform with SDL

图 1.1 littleVGL 所支持的 IDE

其中 littleVGL 官方文档上主要介绍是 Eclipse CDT, 虽然 Eclipse 是跨平台的, 但在笔者亲身体验一番之后, 发现 Eclipse 的环境搭建还是比较麻烦的, 考虑到大部分初学者的感受, 我这里准备打算采用 CodeBlocks 来实现模拟器, 相比 Eclipse 而言, 去掉了很多不必要的步骤, 如果有同学还是执意想用 Eclipse 的话, 请单独联系笔者, 下面让我们正式开始讲述一下 CodeBlocks 模拟器的环境搭建.

2.环境搭建

2.1 材料准备

俗话说的好,工欲善其事,必先利其器,我们先提前把所需要的材料给全部准备好.所需要的材料如下:

1. CodeBlocks 17.12,下载到的文件名为 codeblocks-17.12mingw-setup.exe
2. SDL2,下载到的文件名为 SDL2-devel-2.0.10-mingw.tar.gz
3. littleVGL 模拟器库,下载到的文件名为 lv_pc_simulator.zip

上面所说的材料.笔者已经为大家全部准备好了,可以省去再次下载的过程,但是这个材料的获取过程我还是有必要给大家讲一下的.笔者的搭建环境为 Win10 64 位,建议同学们最好都使用 Windows 系统的电脑来搭建,不管是 Win7 还是 Win8 等,操作步骤是一样的.

2.1.1 CodeBlocks 下载

下载链接为: <http://www.codeblocks.org/downloads/26>

打开链接之后,选择下载 codeblocks-17.12mingw-setup.exe 文件,如下图所示

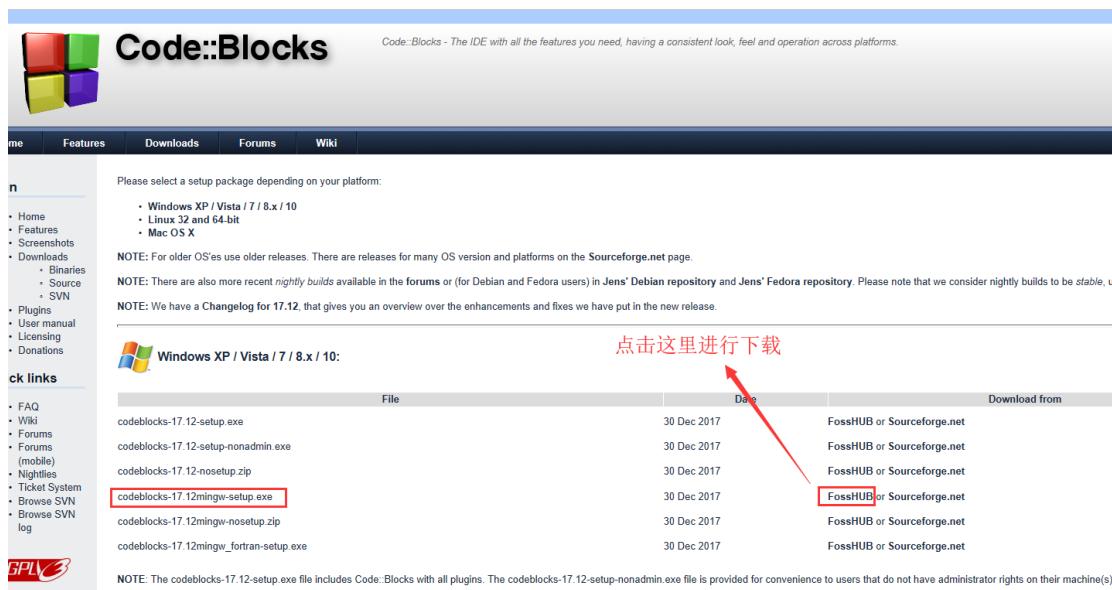


图 2.1.1.1 codeblocks 下载示意

2.1.2 SDL2 下载

下载链接为:<http://www.libsdl.org/download-2.0.php>

打开链接之后,选择下载 SDL2-devel-2.0.10-mingw.tar.gz 文件,如下图所示



图 2.1.2.1 SDL2 下载示意

2.1.3 littleVGL 模拟器库下载

下载链接为: https://littlevgl.com/download/lv_pc_simulator.zip

这个很简单,复制链接地址到浏览器直接进行下载即可

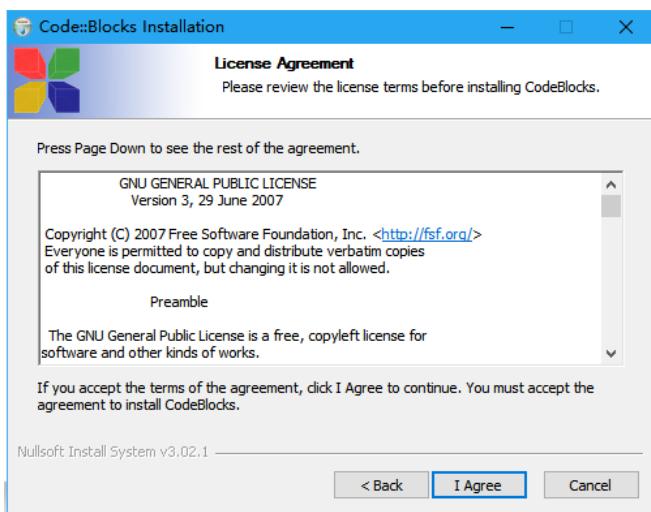
2.2 软件安装和配置

其实这里只需要安装 CodeBlocks 这一个软件就可以了,安装过程也很简单,就是一路 Next 的操作,我这里准备把其安装到我的 D:\baseSoftware 目录下.

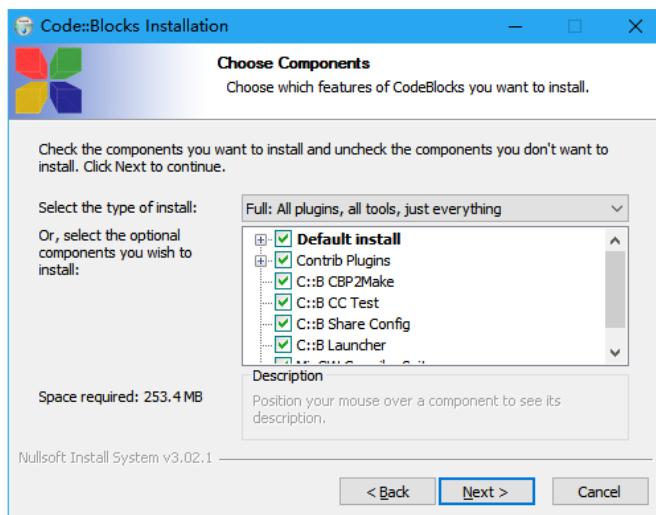
直接双击 codeblocks-17.12mingw-setup.exe 文件,弹出如下对话框:



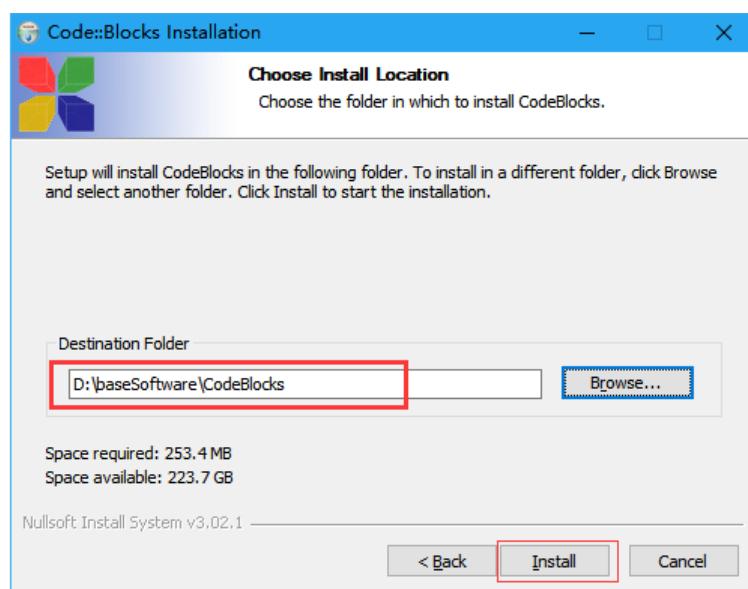
直接点击 Next,接着进入到如下界面:



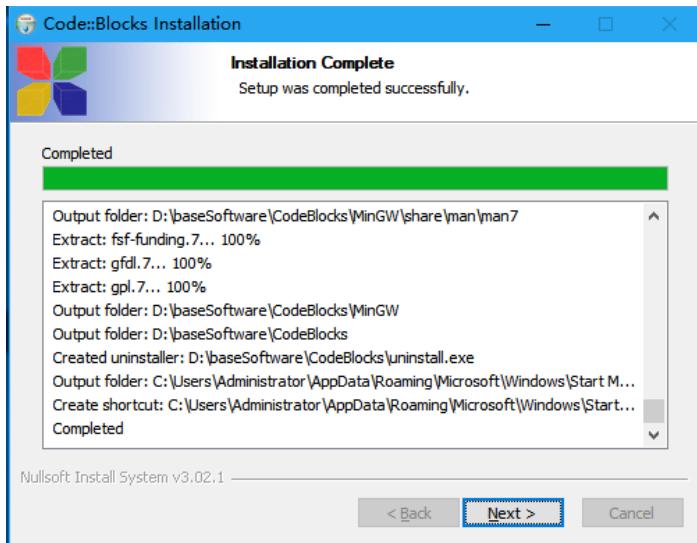
直接点击 I Agree 按钮即可,接着进入到如下界面:



什么都不要去动,直接点击 Next 按钮,接着进入到如下界面

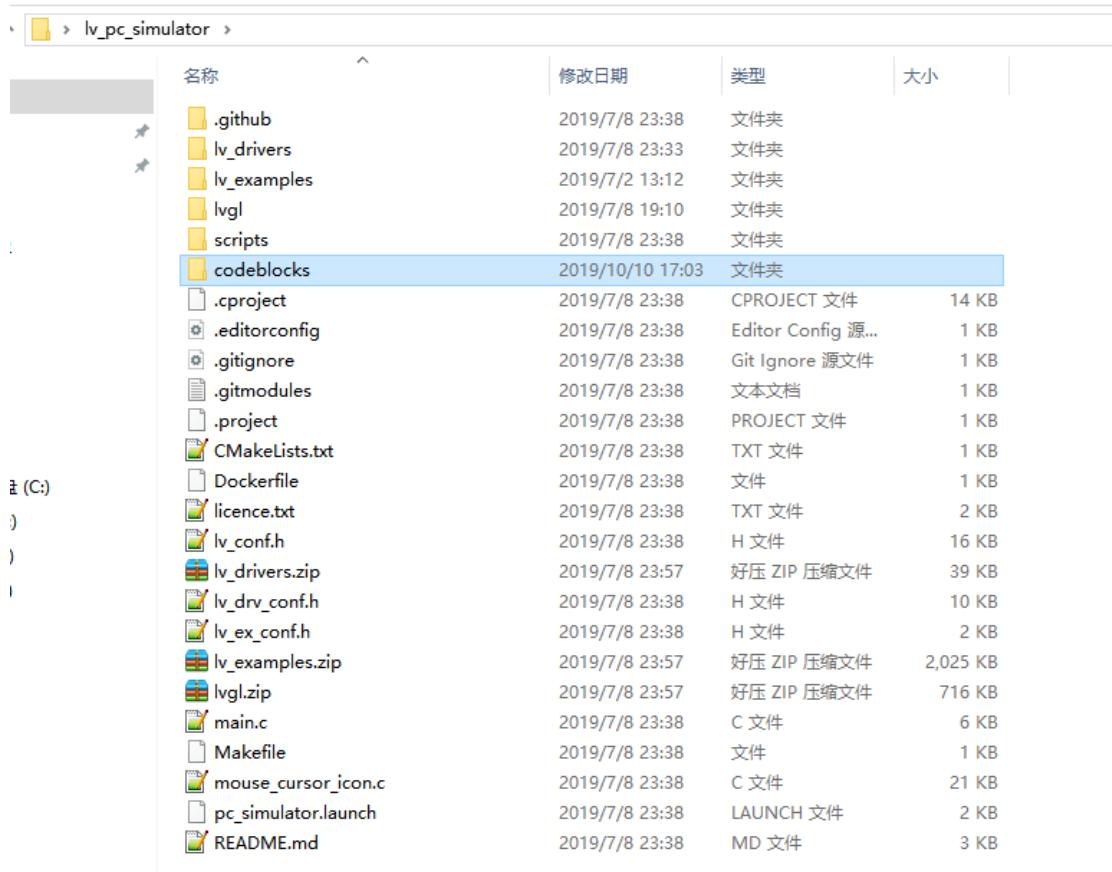


在这里,请根据自己的需求,选择合适的安装路径,选择好之后,直接点击 Install 按钮进行安装,接下来就是等此软件安装完成,在安装过程中,会弹出对话框询问我们,是否立即启动 CodeBlocks,我们先选择否即可,最后软件安装完成之后,如下图所示:



接着点击 Next 按钮,最后再点击 Finish 按钮结束安装.

接着我们需要把 lv_pc_simulator.zip 解压缩到一个英文目录下,我这里先在桌面上新建一个 lv_pc_simulator 目录,然后把 lv_pc_simulator.zip 里的所有文件解压缩到 lv_pc_simulator 目录里面去,接着再把 lv_pc_simulator 目录下的 lv_drivers.zip, lv_examples.zip, lvgl.zip 三个压缩包依次进行在当前目录解压缩的操作,再接着在 lv_pc_simulator 目录下新建一个名为 codeblocks 的子目录,这个子目录主要是用来存放 CodeBlock 项目工程文件的,最后的目录结构如下图所示:



名称	修改日期	类型	大小
.github	2019/7/8 23:38	文件夹	
lv_drivers	2019/7/8 23:33	文件夹	
lv_examples	2019/7/2 13:12	文件夹	
lvgl	2019/7/8 19:10	文件夹	
scripts	2019/7/8 23:38	文件夹	
codeblocks	2019/10/10 17:03	文件夹	
.cproject	2019/7/8 23:38	CPROJECT 文件	14 KB
.editorconfig	2019/7/8 23:38	Editor Config 源...	1 KB
.gitignore	2019/7/8 23:38	Git Ignore 源文件	1 KB
.gitmodules	2019/7/8 23:38	文本文档	1 KB
.project	2019/7/8 23:38	PROJECT 文件	1 KB
CMakeLists.txt	2019/7/8 23:38	TXT 文件	1 KB
Dockerfile	2019/7/8 23:38	文件	1 KB
licence.txt	2019/7/8 23:38	TXT 文件	2 KB
lv_conf.h	2019/7/8 23:38	H 文件	16 KB
lv_drivers.zip	2019/7/8 23:57	好压 ZIP 压缩文件	39 KB
lv_drv_conf.h	2019/7/8 23:38	H 文件	10 KB
lv_ex_conf.h	2019/7/8 23:38	H 文件	2 KB
lv_examples.zip	2019/7/8 23:57	好压 ZIP 压缩文件	2,025 KB
lvgl.zip	2019/7/8 23:57	好压 ZIP 压缩文件	716 KB
main.c	2019/7/8 23:38	C 文件	6 KB
Makefile	2019/7/8 23:38	文件	1 KB
mouse_cursor_icon.c	2019/7/8 23:38	C 文件	21 KB
pc_simulator.launch	2019/7/8 23:38	LAUNCH 文件	2 KB
README.md	2019/7/8 23:38	MD 文件	3 KB

图 2.2.1 lv_pc_simulator 的目录结构

接下来,我们需要打开 CodeBlock 软件来新建工程了,初次打开 CodeBlock 软件,会弹出如下界面:

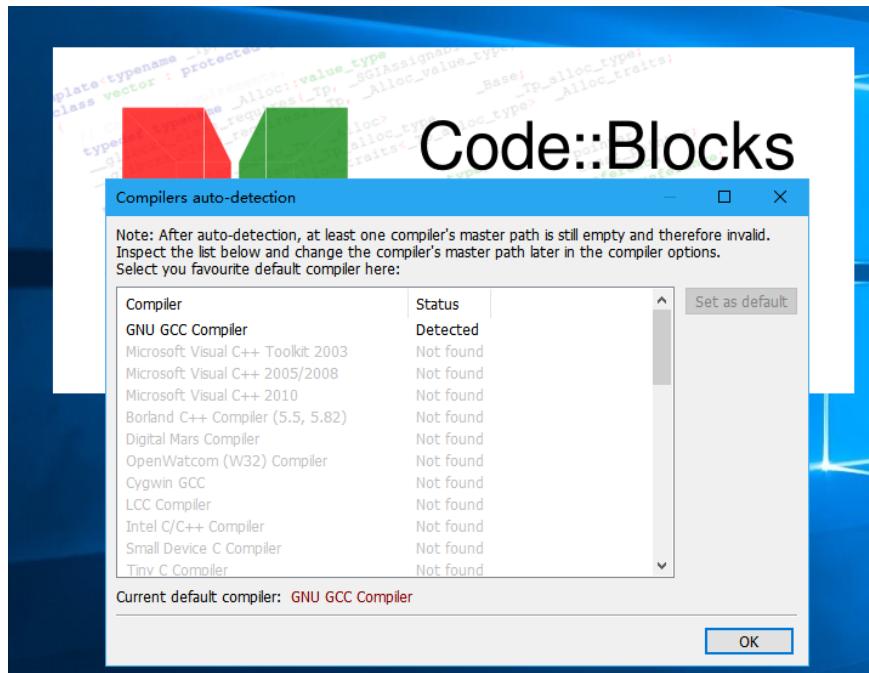


图 2.2.2 CodeBlock 自动检测编译器

当看到这个界面时,不要惊慌,这是在自动检测编译器,我们直接点击 OK 按钮即可.打开软件

之后,点击菜单栏左上角的 File->New->Project...来弹出项目新建向导,我们选择 Console 应用,如下图所示:

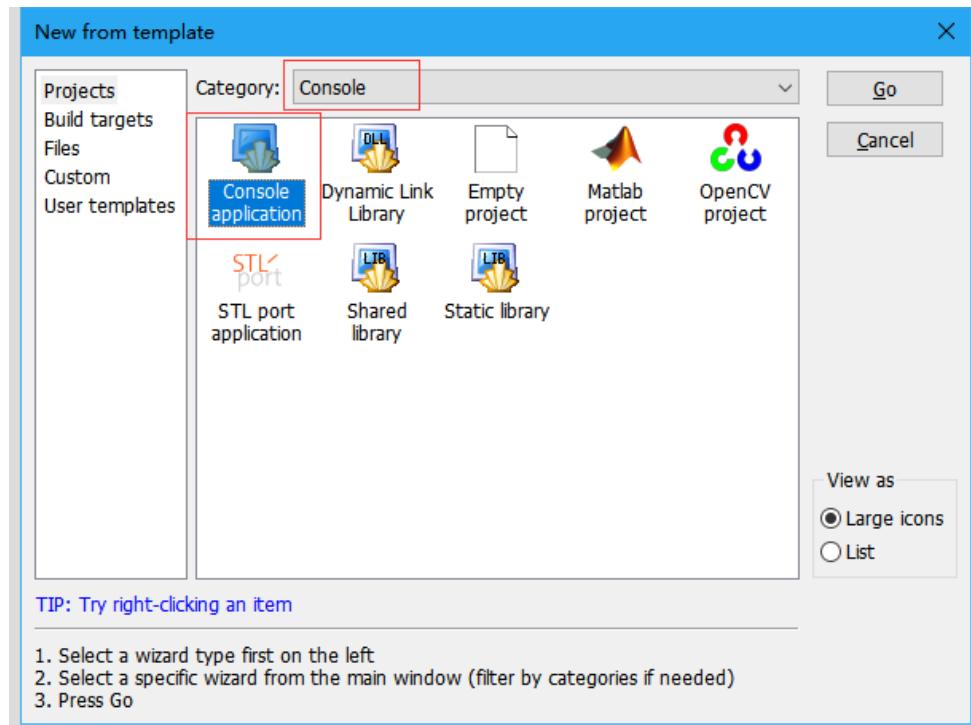


图 2.2.3 选择 Console 应用

接着点击 Go 按钮,又会弹到另外一个对话框,点击 Next 按钮,接着进入到语言选择对话框,如下图所示:

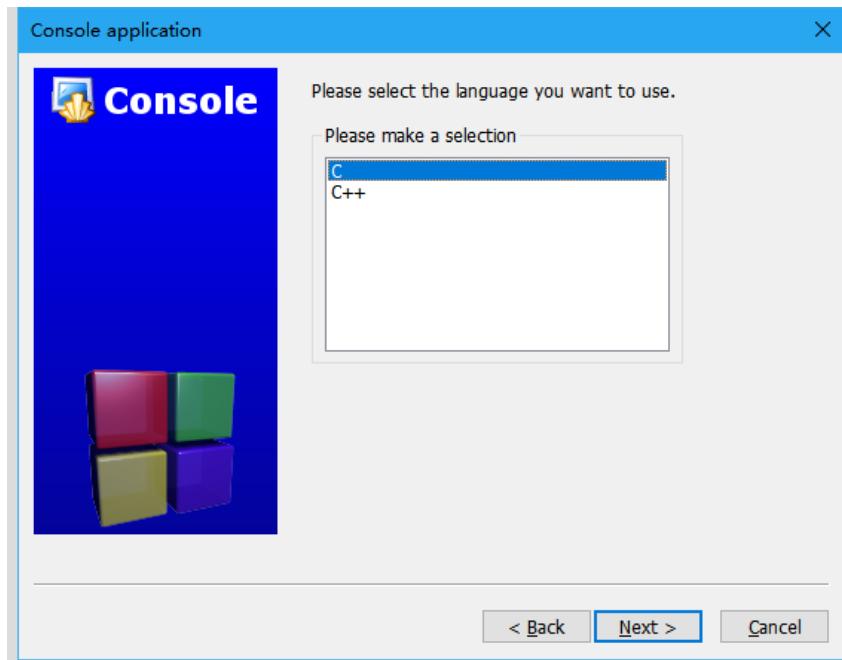


图 2.2.4 语言选择

这里我们选择 C 语言,然后再点击 Next 按钮,又会进入到项目信息输入对话框,如下图所示:

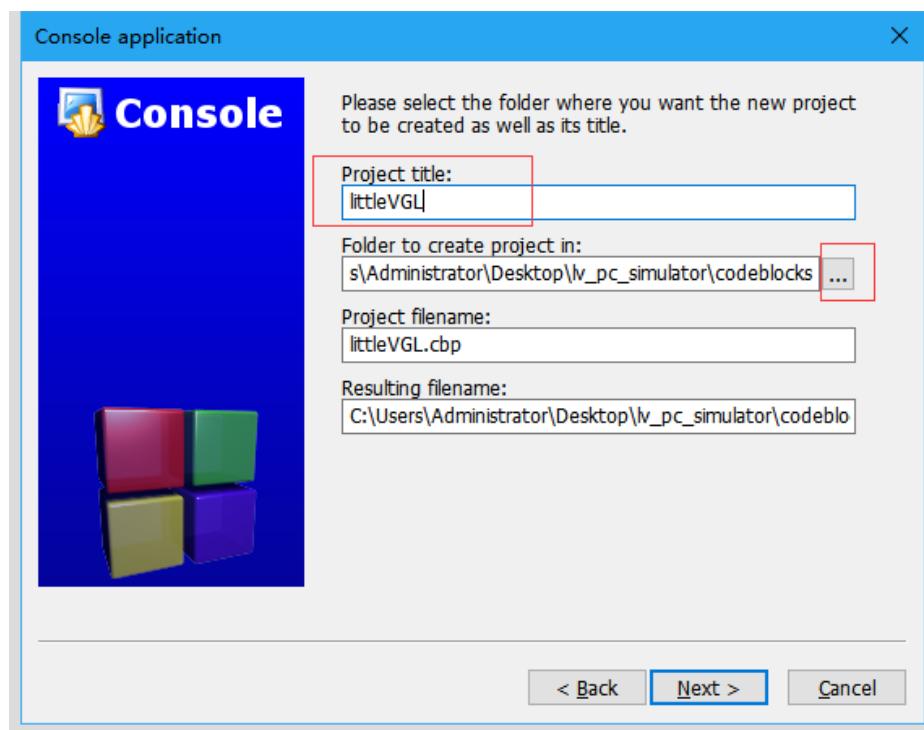


图 2.2.5 项目信息输入

这里我们把项目名输入为 littleVGL,然后选择项目的存放路径到 lv_pc_simulator 目录下的 codeblocks 子目录下,最后就是一路的 Next 操作来完成项目的创建.项目创建完成之后,默认只有一个 main.c 文件,如下图所示:

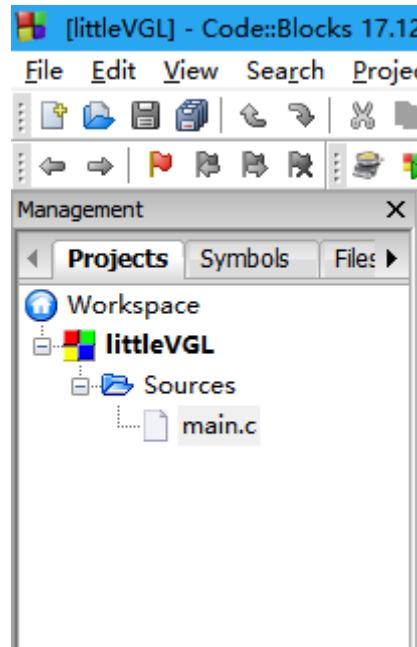


图 2.2.6 项目的默认结构

这里我们不需要默认创建的 main.c 文件,右键 main.c 文件,点击 Remove file from project 来去除此文件,同时最好把 main.c 从磁盘上彻底删除,其位于 lv_pc_simulator\codeblocks\littleVGL 目录下,接下来我们需要把 lv_pc_simulator 模拟器文件递归添加到 CodeBlock 上去,右键



图标,点击 Add files recursively 之后,弹出如下路径选择对话框:

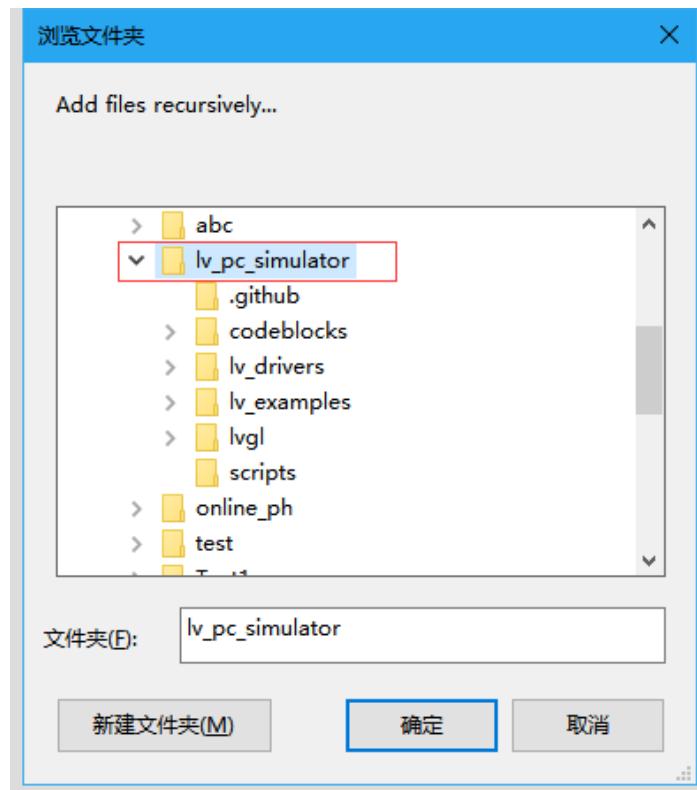


图 2.2.7 路径选择对话框

这里我们一定要选择 `lv_pc_simulator` 顶层目录,然后点击确定按钮,又弹出如下对话框:

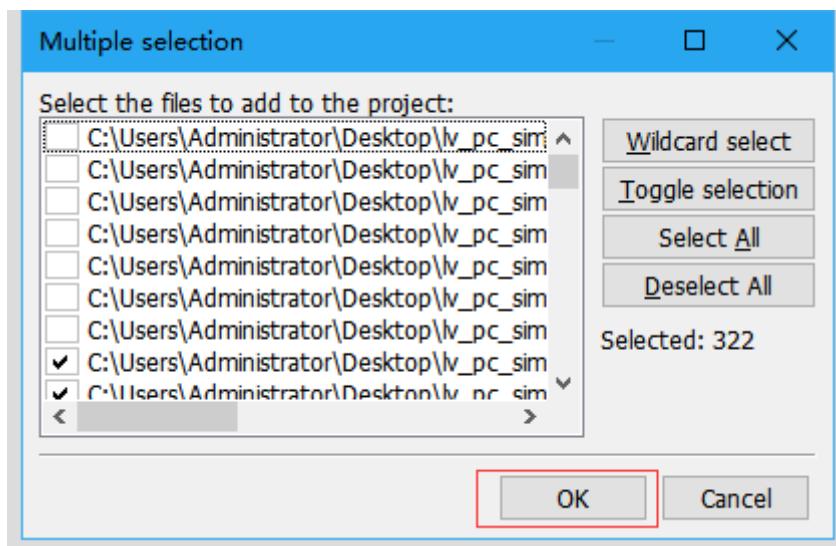


图 2.2.8 具体文件选择

在这里我们什么都不要动,直接点击 OK 按钮即可,接着又弹出

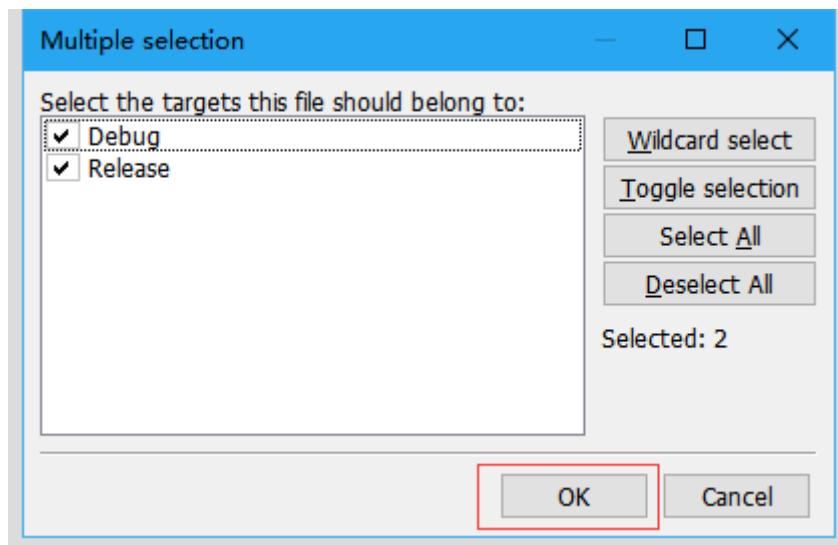


图 2.2.9 targets 选择

同样道理,什么也不动,直接点击 OK 按钮,至此文件的添加操作就完成了,接下来我们需要用到

SDL2-devel-2.0.10-mingw.tar.gz 里面的文件了,我们先可以把其解压缩到桌面,然后做下面 2 点操作:

1. 把 SDL2-2.0.10\i686-w64-mingw32\include 目录下的 SDL2 目录拷贝到 CodeBlock 的安装目录 D:\baseSoftware\CodeBlocks\MinGW\include 下面

2. 把 SDL2-2.0.10\i686-w64-mingw32\lib 下的 7 个.a 库文件拷贝到 CodeBlock 的安装目录 D:\baseSoftware\CodeBlocks\MinGW\lib 下面

做完上面 2 点操作之后,我们需要配置一下 CodeBlock 软件,右键  图标,点击 Build options, 打开对话框之后, 在 Other linker options 下输入 -I mingw32 -ISDL2main -ISDL2, 如下图所示:

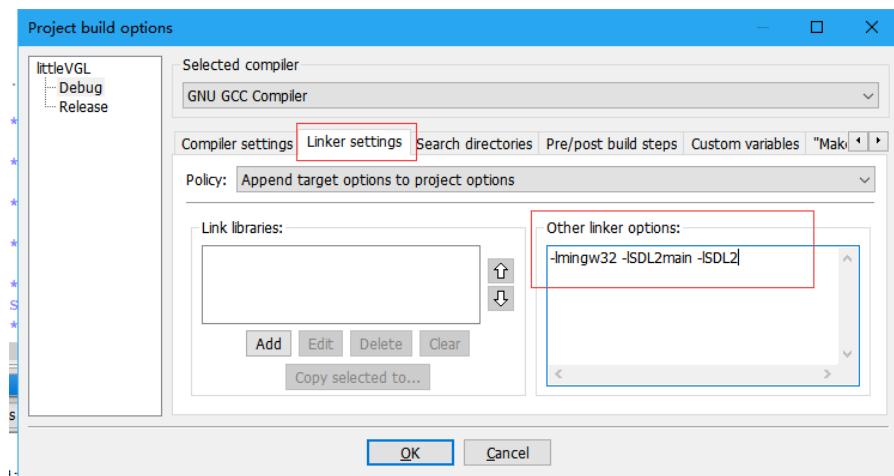


图 2.2.10 添加链接选项

然后我们还需要添加头文件搜索路径,如下图所示:

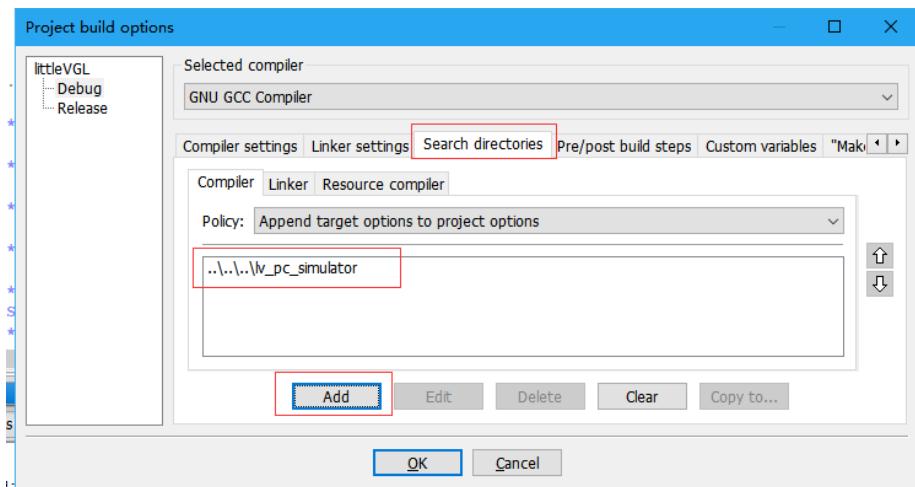


图 2.2.11 添加头文件路径

注意,添加头文件路径时,选择 lv_pc_simulator 相对路径即可,最后点击 OK 按钮保存配置.接着我们点击打开 main.c 文件

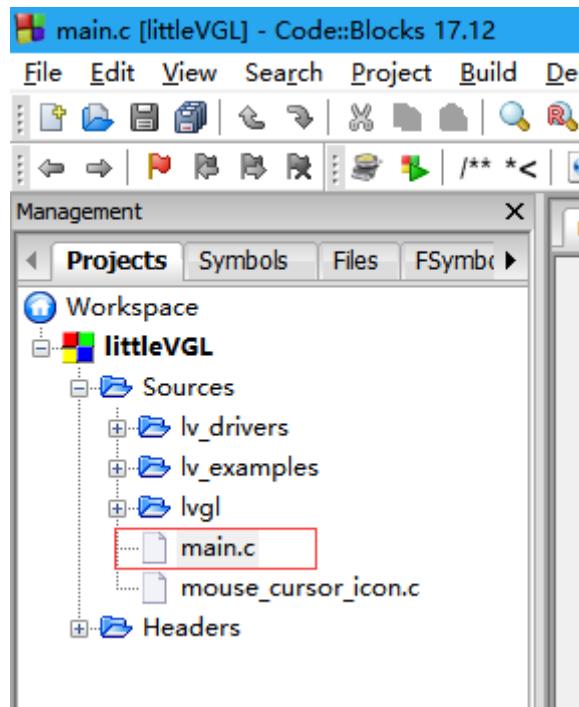


图 2.2.12 main.c 文件

打开之后,我们去掉 71 行处的行注释,即让 demo_create(); 例程得到运行,当然你也可以选择另外 4 个例程中的一个来运行,如下所示:

```
benchmark_create();      在 74 行
lv_test_theme_1(lv_theme_night_init(15, NULL));    在 77 行
lv_test_theme_2();      在 79 行
lv_test_group_1();      在 81 行
```

选择好例程之后,终于迎来了我们的第一次编译,我们点击 图标来编译,不出意外的话,会

报 1 个错误,如下图所示:

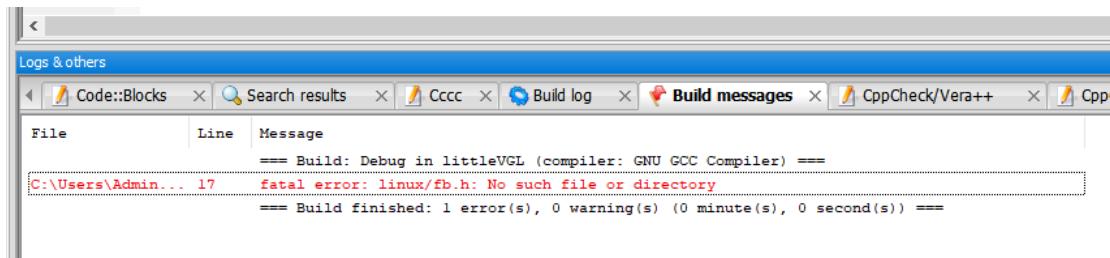


图 2.2.13 初次编译报 1 个错

别慌,这个错误很好解决,这是因为 Windows 系统不支持 Linux 帧缓冲,把宏定义 USE_FBDEV 的值改为 0 就可以了,USE_FBDEV 宏在 lv_pc_simulator\lv_drv_conf.h 头文件中的 186 行处,改成 0 之后,编译不会报错了,但是运行起来后,会报如下图所示的一个错误.

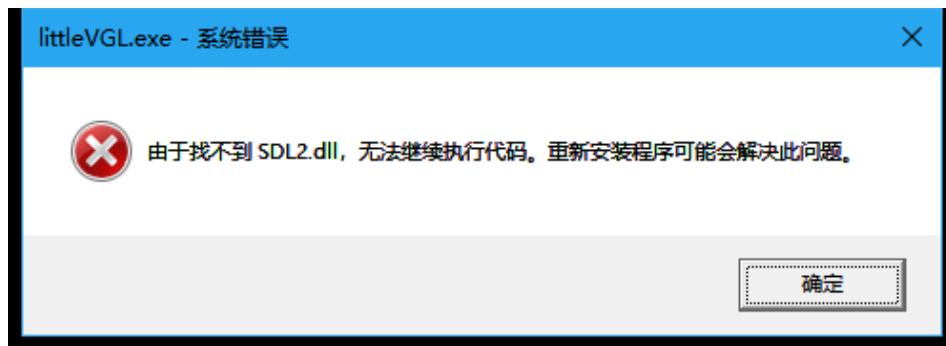


图 2.2.14 运行报错

这个错误也很好解决,是因为找不到 SDL2.dll 库,我们需要把 SDL2-2.0.10i686-w64-mingw32\bin 目录下的 SDL2.dll 文件拷贝到 lv_pc_simulator\codeblocks\littleVGL\bin\Debug 目录下面,如下图所示:

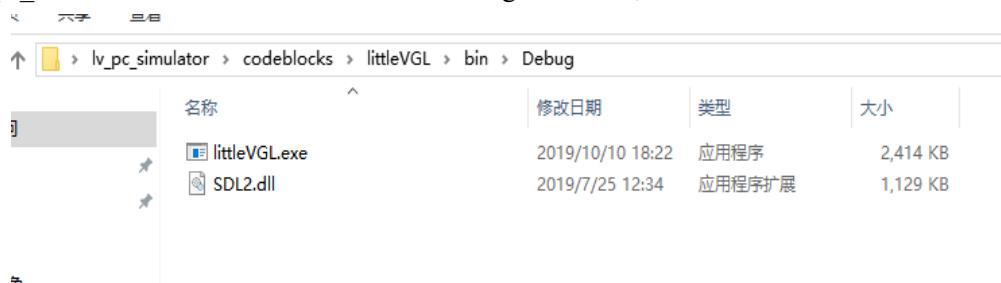


图 2.2.15 拷贝 SDL2.dll 库

拷贝完成之后,再次编译运行就不会有任何错误了,最终的模拟器例程演示效果请看下面

2.3 模拟器演示效果

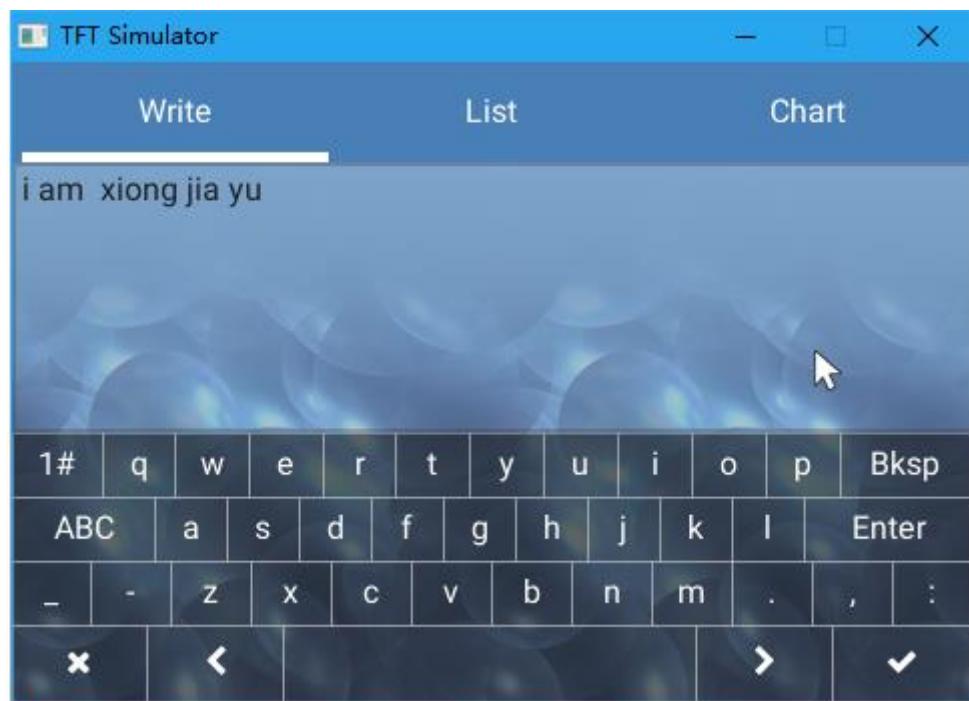


图 2.3.1 Write 界面



图 2.3.2 List 界面



图 2.3.3 Chart 界面

上面的三张演示效果图是由 `demo_create()` 例程产生的，你们自己也可以通过行注释来运行其他的例程。

3.如何添加自己的 GUI 测试代码

怕有的同学不知道怎么添加自己的 GUI 测试代码,特此稍微讲解一下,步骤也是很简单的,

你可以直接在 main.c 文件中添加你自己的测试函数,然后放在 main 函数中的 hal_init();后面进行调用即可,当然你得先把官方自带的几个例程先给注释掉,否则会影响你的测试效果的,直接在 main.c 文件中添加的方式虽然简单,但是我不推荐,因为不符合我们的模块化编程思想,最好的操作步骤应该是这样的,我们在 lv_pc_simulator\lv_examples\lv_apps 目录下新建一个 test 目录,然后在 test 目录下新建 test.c 和 test.h 文件,然后用前面介绍过的方法,把 test 目录递归添加到 lv_apps 分组下,如下图所示:

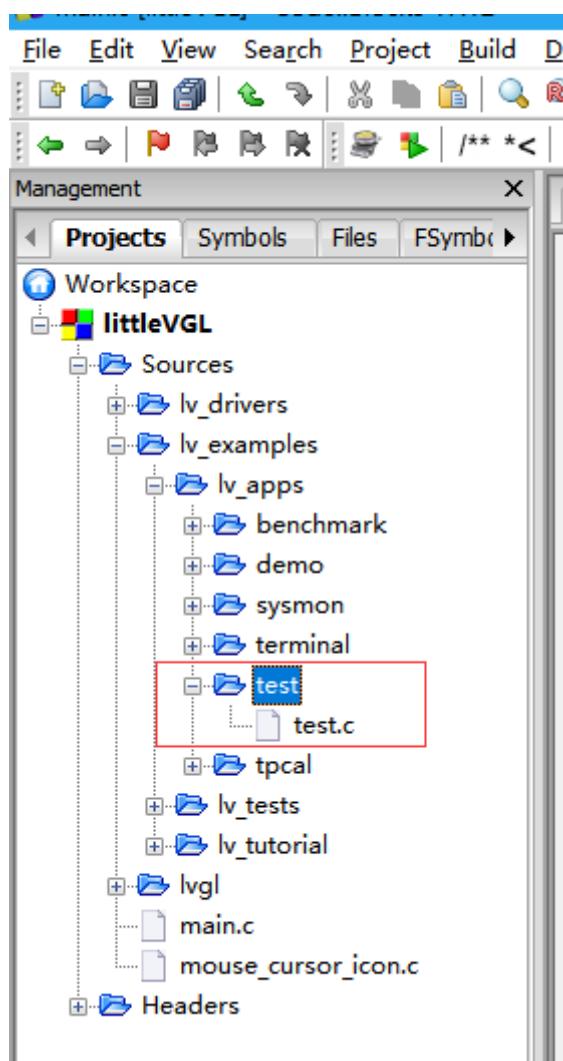


图 3.1 添加 test 目录

test.h 文件的内容可以大致如下:

```
#ifndef __TEST_H__  
#define __TEST_H__
```

```
#ifdef __cplusplus
extern "C" {
#endif

/******************
 *      INCLUDES
 *****************/
#ifndef LV_CONF_INCLUDE_SIMPLE
#include "lvgl.h"
#include "lv_ex_conf.h"
#else
#include "../../lvgl/lvgl.h"
#include "../../lv_ex_conf.h"
#endif

void test_start(void);

#endif /* __cplusplus
 */ /* extern "C" */
#endif // __TEST_H__
```

test.c 文件的内容可以大致如下:

```
#include "test.h"

void test_start()
{
    //获取当前的屏幕对象
    lv_obj_t * scr = lv_disp_get_scr_act(NULL);

    //在屏幕上创建一个 label 控件
    lv_obj_t * label1 = lv_label_create(scr, NULL);

    //设置 label 的文本内容
    lv_label_set_text(label1, "I am xiong jia yu");

    //设置文本和父控件(在这里就是屏幕)居中对齐
    lv_obj_align(label1, NULL, LV_ALIGN_CENTER, 0, 0);
}
```

然后在 main.c 文件中引入 test.h 的头文件,如下所示

```
#include "lv_examples/lv_apps/test/test.h"
```

再接着把 test_start();放在 main 函数中的 hal_init();后面进行调用,最后编译运行可以看到如下效果:

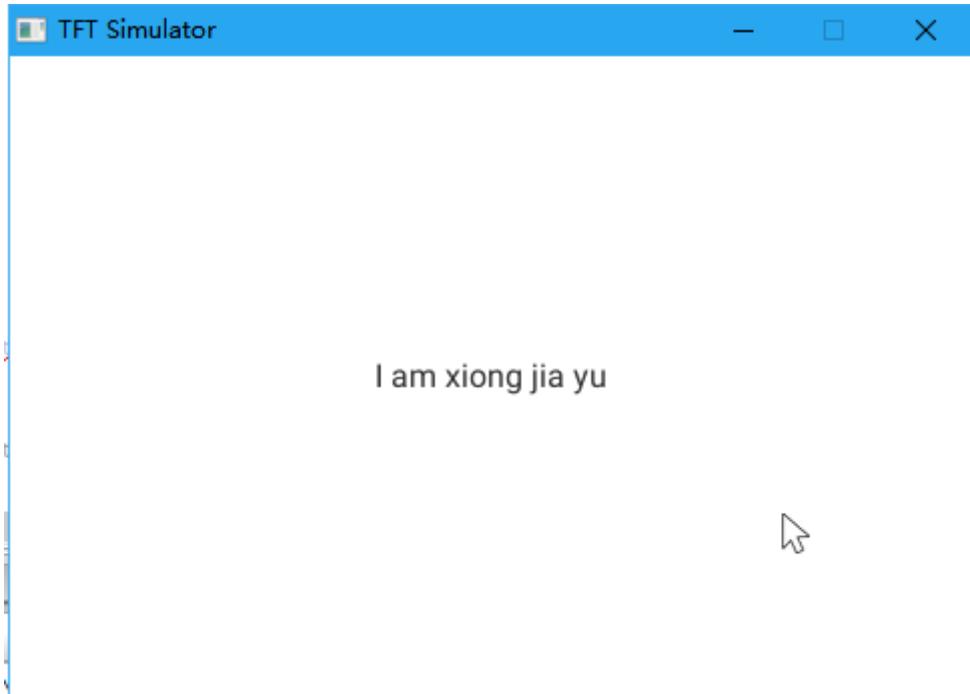


图 3.2 自己的代码测试效果

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

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,所以你也要确保 `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,他们俩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); //设置中断优先级
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lvg1定时
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();             //lvg1系统初始化
39     lv_port_disp_init();  //lvg1显示接口初始化
40     lv_port_indev_init(); //lvg1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(); //事务处理器
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 俩个文件,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:

```
main.c
28 int main(void)
29 {
30     delay_init();          //延时函数初始化
31     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断优先级
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lvg1
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();             //lvg1系统初始化
42     lv_port_disp_init();   //lvg1显示接口初始化
43     lv_port_indev_init();  //lvg1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次信息,如下图所示:



图 3.4.1 每隔 5 秒输出一次信息

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,如下图所示:

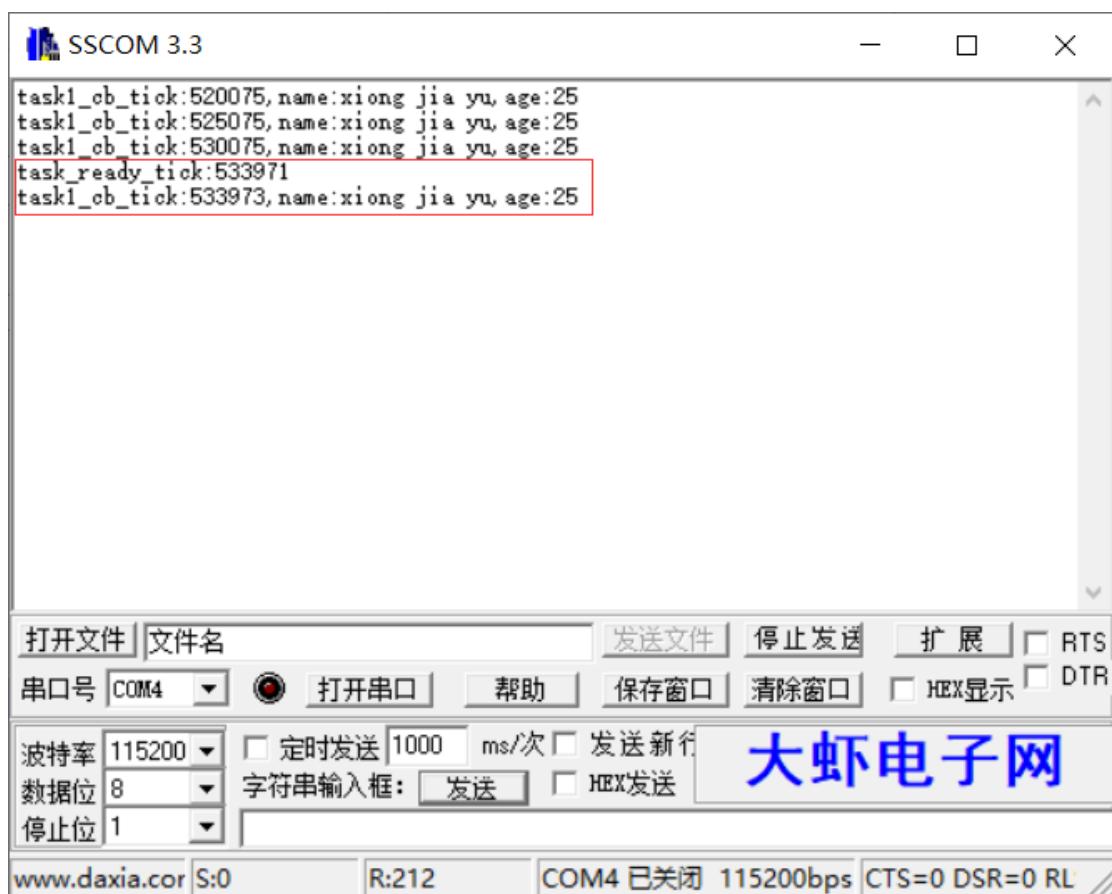


图 3.4.2 按下 KEY1 键时的串口输出

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

正点原子 littleVGL 开发指南

lv_obj 基础对象

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

lv_obj 基础对象

1. 介绍

littleVGL 是以对象为概念的,而其最核心的基础对象是 lv_obj 控件,其他的所有专用控件(比如按钮,标签,列表等)都是在此 lv_obj 对象的基础上衍生出来的,所有的控件对象都具有一些共同的属性,如下所示:

- 1) 位置(Position)
- 2) 大小(Size)
- 3) 父类(Parent)
- 4) 是否可拖拽(Drag enable)
- 5) 是否可点击(Click enable)等等

你可以通过 `lv_obj_set_...` 和 `lv_obj_get_...` 这样的 API 接口对来读写这些属性,就比如下面这样:

```
/*设置对象基本属性*/
lv_obj_set_size(btn1, 100, 50); //按钮大小
lv_obj_set_pos(btn1, 20,30); //按钮位置
```

当然了,除了共同属性外,不同的控件都会有自己的专有属性,这些内容在后面的章节中会具体展开.

下面我们来了解一下对象的工作机制,父对象可以被看作是其子对象的容器,每个对象只有一个父对象(screen 对象没有父对象),父对象可以有无限数量的子对象,同时父对象的类型是没有限制,父对象和子对象之间具有如下 2 点特性:

1) 一起移动

如果父对象的位置更改,则子对象将随父对象一起移动,因此子对象的坐标位置是以父对象的左上角而言的,而不是以屏幕的左上角

2) 子对象只能在父对象的区域内显示

如果子对象的一部分在父对象的外面,那么子对象的这一部分将不会被显示出来

在 littleVGL 中,对象可以动态的被创建和删除,每一种对象都有其专属的 create 创建函数,他需要 2 个参数,为 parent 和 copy 参数,创建函数看起来如下这样:

```
lv_obj_t * lv_<type>_create(lv_obj_t * parent, lv_obj_t * copy);
parent: 父对象,如果想创建一个 screen 对象,那么请传 NULL 值
copy: 此参数可选,表示创建新对象时,把 copy 对象上的属性值复制过来
```

所有对象的删除函数都是相同的,如下面所示:

```
void lv_obj_del(lv_obj_t * obj);
```

`lv_obj_del` 删除对象操作是立即执行的,如果你由于某种原因,不想立即删除的话,你可以使用 `lv_obj_del_async(obj)` 接口来异步删除,它会在下一个 `lv_task_handler` 调用时被执行,另外你可

以通过 `lv_obj_clean(obj);` 接口来清除某个 `obj` 对象下的所有子对象.

我们再来了解另外一个核心概念 Screen 屏幕对象, 屏幕对象是一个特殊的对象, 因为他自己没有父对象, 所以它以这样的方式来创建:

```
lv_obj_t * screen1 = lv_obj_create(NULL, NULL);
```

默认情况下, littleVGL 会为显示器创建一个 `lv_obj` 类型的基础对象来作为它的屏幕, 即最顶层的父类, 可以通过 `lv_scr_act()` 接口来获取当前活跃的屏幕对象, 通过 `lv_scr_load()` 接口来设置一个新的活跃屏幕对象

2.lv_obj 的 API 接口

2.1 主要数据类型

2.1.1 事件相关的数据类型

```
enum {
    LV_EVENT_PRESSED,
    LV_EVENT_PRESSING,
    LV_EVENT_PRESS_LOST,
    LV_EVENT_SHORT_CLICKED,
    LV_EVENT_LONG_PRESSED,
    LV_EVENT_LONG_PRESSED_REPEAT,
    LV_EVENT_CLICKED,
    LV_EVENT_RELEASED,
    LV_EVENT_DRAG_BEGIN,
    LV_EVENT_DRAG_END,
    LV_EVENT_DRAG_THROW_BEGIN,
    LV_EVENT_KEY,
    LV_EVENT_FOCUSED,
    LV_EVENT_DEFOCUSSED,
    LV_EVENT_VALUE_CHANGED,
    LV_EVENT_INSERT,
    LV_EVENT_REFRESH,
    LV_EVENT_APPLY,
    LV_EVENT_CANCEL,
    LV_EVENT_DELETE,
};

typedef uint8_t lv_event_t; //事件类型

typedef void (*lv_event_cb_t)(struct _lv_obj_t * obj, lv_event_t event); //事件回调函数
```

跟事件相关的数据类型主要是一个枚举体和函数指针,枚举体中列出了系统中所支持的所有事件类型

2.1.2 对齐数据类型

```
enum {
    LV_ALIGN_CENTER = 0,
    LV_ALIGN_IN_TOP_LEFT,
    LV_ALIGN_IN_TOP_MID,
    LV_ALIGN_IN_TOP_RIGHT,
    LV_ALIGN_IN_BOTTOM_LEFT,
    LV_ALIGN_IN_BOTTOM_MID,
```

```

LV_ALIGN_IN_BOTTOM_RIGHT,
LV_ALIGN_IN_LEFT_MID,
LV_ALIGN_IN_RIGHT_MID,
LV_ALIGN_OUT_TOP_LEFT,
LV_ALIGN_OUT_TOP_MID,
LV_ALIGN_OUT_TOP_RIGHT,
LV_ALIGN_OUT_BOTTOM_LEFT,
LV_ALIGN_OUT_BOTTOM_MID,
LV_ALIGN_OUT_BOTTOM_RIGHT,
LV_ALIGN_OUT_LEFT_TOP,
LV_ALIGN_OUT_LEFT_MID,
LV_ALIGN_OUT_LEFT_BOTTOM,
LV_ALIGN_OUT_RIGHT_TOP,
LV_ALIGN_OUT_RIGHT_MID,
LV_ALIGN_OUT_RIGHT_BOTTOM,
};

typedef uint8_t lv_align_t;

```

这也是一个枚举体,列出了控件与控件之间的所有可能对齐方式,为了看得明白,请看下面的对齐示意图:

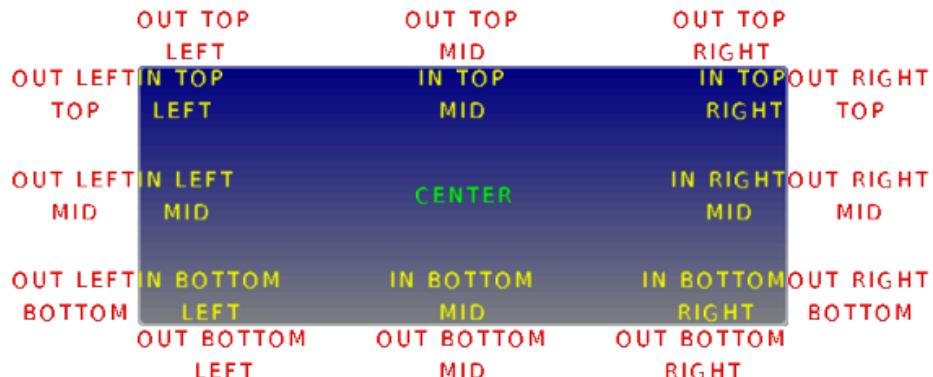


图 2.1.2.1 对齐示意图

2.1.3 拖拽方式数据类型

```

enum {
    LV_DRAG_DIR_HOR = 0x1, //对象只能在水平方向上被拖拽
    LV_DRAG_DIR_VER = 0x2, //对象只能在垂直方向上被拖拽
    LV_DRAG_DIR_ALL = 0x3, //对象可以被随意拖拽
};

typedef uint8_t lv_drag_dir_t;

```

2.1.4 lv_obj 的句柄数据类型

```
typedef struct _lv_obj_t
{
    struct _lv_obj_t * par; //指向父对象的
    lv_ll_t child_ll; //是一个链表,用来保存所有的子对象的
    lv_area_t coords; //此对象的坐标(x1, y1, x2, y2)
    lv_event_cb_t event_cb; //此对象的事件回调函数
    lv_signal_cb_t signal_cb; //归 littleVGL 库内部使用,不用理会
    lv_design_cb_t design_cb; //归 littleVGL 库内部使用,不用理会
    void * ext_attr; //扩展属性,就是利用此字段来衍生出不同的控件
    const lv_style_t * style_p; //此对象的样式

#if LV_USE_GROUP != 0
    void * group_p; //此对象所在的分组
#endif

#if LV_USE_EXT_CLICK_AREA == LV_EXT_CLICK_AREA_TINY
    uint8_t ext_click_pad_hor;
    uint8_t ext_click_pad_ver;
#endif

#if LV_USE_EXT_CLICK_AREA == LV_EXT_CLICK_AREA_FULL
    lv_area_t ext_click_pad;
#endif

    uint8_t click : 1; //启用可通过输入设备点击对象, 如果被关闭点击功能那么对象和它后
                      //面的对象都不能被点击(例如标签控件默认不可点击)
    uint8_t drag : 1; //使能拖拽 (通过输入设备移动)
    uint8_t drag_throw : 1; //使能拖拽的”抛掷(throwing)”,如果对象有冲量(惯性)
    uint8_t drag_parent : 1; //如果被启用那么当拖拽发生时对象的父对象也会被移动。它看
                           //起来像父对象被拖拽了, 这是递归, 因此爷爷对象(grandparents)也会被一定移动
    uint8_t hidden : 1; //隐藏对象。对象将不会被画出并可以视对象为不存在的, 他的子对
                       //象也会被隐藏
    uint8_t top : 1; //如果开启了那么当对象或他的任何子对象被点击那么该对象会被前置
    uint8_t opa_scale_en : 1; //是否使能透明度
    uint8_t parent_event : 1; //是否转发事件给父对象, 递归的, 所以也可以传播给爷爷对象
    lv_drag_dir_t drag_dir : 2; //在某些特定的方向上使能拖拽
    uint8_t reserved : 6; //保留
    uint8_t protect; //保护一些属性, 在特定场合下阻止一些事情的发生
    lv_opa_t opa_scale; //对象的透明度
    lv_coord_t ext_draw_pad;

#if LV_USE_OBJ_REALIGN
    lv_realign_t realign; //保存对象最后一次对齐时的内部参数

```

```
#endif

#if LV_USE_USER_DATA
lv_obj_user_data_t user_data; //每个对象都可以携带用户自定义的数据,其中
//lv_obj_user_data_t 的具体数据类型是可以在 lv_conf.h 中配置
#endif

} lv_obj_t;
```

此数据类型为结构体,其中还参杂了条件编译,有点小复杂,不过大家放心,我们一般都不会直接去操作其属性的,都是通过 API 接口,大家只要先了解了解就可以了,后面在 API 接口中还会具体展开的

2.1.5 保护数据类型

```
enum {
    LV_PROTECT_NONE      = 0x00, //不保护
    LV_PROTECT_CHILD_CHG = 0x01, //关闭子对象改变信号,被库内部使用
    LV_PROTECT_PARENT     = 0x02, //防止父对象自动改变(例如页面在后台移动创建的
                                //子对象使其能够滚动)
    LV_PROTECT_POS        = 0x04, //防止自动定位(例如在容器的布局)
    LV_PROTECT_FOLLOW     = 0x08,
    LV_PROTECT_PRESS_LOST = 0x10,
    LV_PROTECT_CLICK_FOCUS = 0x20, //防止对象自动聚焦,如果他在一个群组(Group)中并
                                //打开了点击聚焦
};

typedef uint8_t lv_protect_t;
```

这也是一个枚举体,主要是用在某些特定场合下阻止一些事情的发生

2.2 API 接口

2.2.1 littleVGL 库初始化

```
void lv_init(void)
```

这个接口必须得在使用 littleVGL 之前先调用一次,我们直接放在 main 函数中调用即可,如下图所示:

```

18 int main(void)
19 {
20     delay_init();           //延时函数初始化
21     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
22     uart_init(115200);    //串口初始化为115200
23     LED_Init();           //LED端口初始化
24     KEY_Init();           //按键初始化
25     BEEP_Init();          //蜂鸣器初始化
26     TIM3_Int_Init(999, 71); //定时器初始化 (1ms中断)
27     FSMC_SRAM_Init();    //外部1MB的sram初始化
28     LCD_Init();           //LCD初始化
29     tp_dev.init();        //触摸屏初始化
30
31     lv_init();             //lvgl系统初始化
32     lv_port_disp_init();  //lvgl显示接口初始化
33     lv_port_indev_init(); //lvgl输入接口初始化
34
35
36
37
38
39
40
41
42
43
44

```

图 2.2.1.1 lv_init 的调用

2.2.2 创建对象

```
lv_obj_t * lv_obj_create(lv_obj_t * parent, const lv_obj_t * copy);
```

参数:

parent: 指向父对象,如果传 NULL 的话,则是在创建一个 screen 屏幕

copy: 此参数可选,表示创建新对象时,把 copy 对象上的属性值复制过来

返回值:

返回新创建出来的对象句柄,如果为 NULL 的话,说明 littleVGL 所管理的堆空间不足了

2.2.3 立即删除对象

```
lv_res_t lv_obj_del(lv_obj_t * obj);
```

参数:

obj: 需要被删除的对象

返回值:

删除成功之后,将会返回 LV_RES_INV,其他值代表删除失败

lv_res_t 是 lv_types.h 中定义的一个枚举体数据类型,如下所示:

```
enum {
    LV_RES_INV = 0,
    LV_RES_OK,
};
```

```
typedef uint8_t lv_res_t;
```

其中 LV_RES_INV 是 littleVGL result invalid 的缩写,通常用于表示删除对象成功了,对象删除之后,也就是所谓的 invalid(无效的)了

2.2.4 异步删除对象

```
void lv_obj_del_async(struct _lv_obj_t *obj);
```

参数:

obj: 需要被删除的对象

此 API 接口是没有返回值的,因为它不是立即删除对象的,所以拿不到删除结果,它会在下一个 lv_task_handler 调用时被执行,其实它内部的实现原理就是开启了一个周期为 0ms 的最高优先级的 Task 任务

2.2.5 清空所有子对象

```
void lv_obj_clean(lv_obj_t * obj);
```

参数:

obj: 对象句柄

当执行此操作之后,obj 对象下的所有子对象全部会被删除

2.2.6 使对象无效化

```
void lv_obj_invalidate(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

使对象无效化之后,在下一个 lv_task_handler 调用时,此对象将会被重绘

2.2.7 更换父对象

```
void lv_obj_set_parent(lv_obj_t * obj, lv_obj_t * parent);
```

参数:

obj: 对象句柄

parent: 新的父对象

更换父对象之后,它的相对位置还是保持不变的

2.2.8 将对象的层级前置

```
void lv_obj_move_foreground(lv_obj_t * obj);
```

参数:

obj: 对象句柄

当多个控件对象在屏幕上发生位置重叠时,如果想调节此对象在 z 轴上的层次,可以调用此 API 接口

2.2.9 将对象的层级后置

```
void lv_obj_move_background(lv_obj_t * obj);
```

参数:

obj: 对象句柄

当多个控件对象在屏幕上发生位置重叠时,如果想调节此对象在 z 轴上的层次,可以调用此 API 接口

2.2.10 设置对象的坐标位置

```
void lv_obj_set_pos(lv_obj_t * obj, lv_coord_t x, lv_coord_t y);
```

参数:

obj: 对象句柄

x: x 坐标

y: y 坐标

lv_coord_t 的具体数据类型是在 lv_conf.h 文件中配置的,一般 2 字节就够了,如:

```
typedef int16_t lv_coord_t;
```

2.2.11 单独设置 x 坐标

```
void lv_obj_set_x(lv_obj_t * obj, lv_coord_t x);
```

参数:

obj: 对象句柄

x: x 坐标

2.2.12 单独设置 y 坐标

```
void lv_obj_set_y(lv_obj_t * obj, lv_coord_t y);
```

参数:

obj: 对象句柄

y: y 坐标

2.2.13 设置对象的大小

```
void lv_obj_set_size(lv_obj_t * obj, lv_coord_t w, lv_coord_t h);
```

参数:

obj: 对象句柄

w: 宽度

h: 高度

2.2.14 单独设置对象的宽度

```
void lv_obj_set_width(lv_obj_t * obj, lv_coord_t w);
```

参数:

obj: 对象句柄

w: 宽度

2.2.15 单独设置对象的高度

```
void lv_obj_set_height(lv_obj_t * obj, lv_coord_t h);
```

参数:

h: 高度

2.2.16 设置对象的对齐方式

```
void lv_obj_align(lv_obj_t * obj, const lv_obj_t * base, lv_align_t align, lv_coord_t x_mod,
lv_coord_t y_mod);
```

参数:

obj: 需要进行对齐的对象

base: 与哪一个对象进行对齐,即为基准或参考对象,如果传 NULL 的话,则默认以父对象为参考

align: 对齐方式,总共有 21 种

x_mod: 指定方式对齐之后,再进行 x 轴方向的偏移修正

Y_mod: 指定方式对齐之后,再进行 y 轴方向的偏移修正

请注意,此接口跟调用时的位置是有关系的,一定得在直接或间接确定对象的大小之后,再来调用此接口进行对齐操作,因为其对齐原理是要利用对象的大小等参数来算出相对对齐之后的坐标,为了便于理解,align 的对齐方式请看下图:

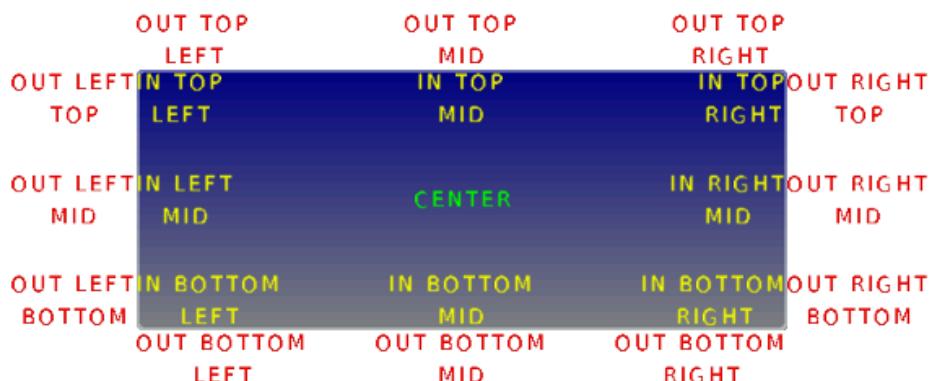


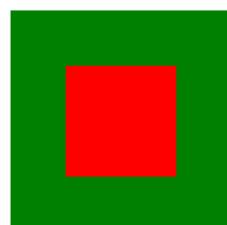
图 2.2.16.1 对齐方式

这里给出所有对齐情况下的图例,先申明一下,下面的所有图例中,  绿色的是代表 base 参考对象,大小为 200x200,而  红色的是代表 obj 对象,大小为 100x100,同时 x_mod 和 y_mod 的值都为 0,代码调用方式如下所示:

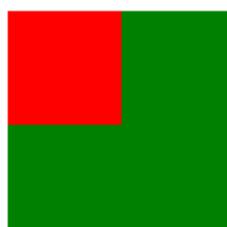
```
lv_obj_align(obj, base, align,0,0);
```

随着 align 取不同的值,得到下面不同的图例:

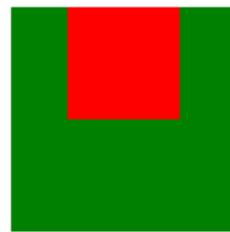
1) LV_ALIGN_CENTER



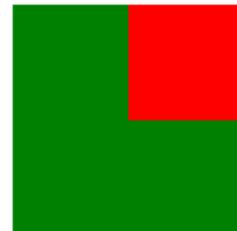
2) LV_ALIGN_IN_TOP_LEFT



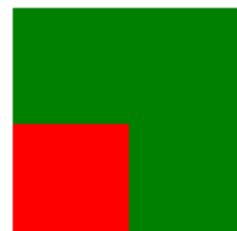
3) LV_ALIGN_IN_TOP_MID



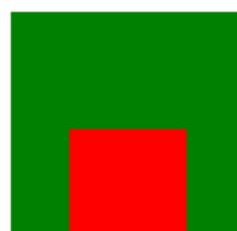
4) LV_ALIGN_IN_TOP_RIGHT



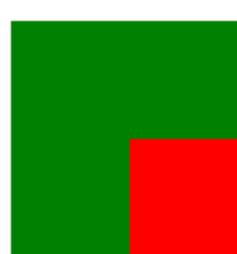
5) LV_ALIGN_IN_BOTTOM_LEFT



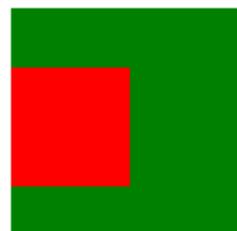
6) LV_ALIGN_IN_BOTTOM_MID



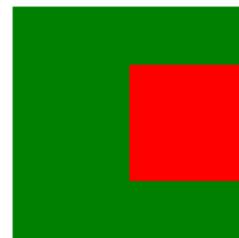
7) LV_ALIGN_IN_BOTTOM_RIGHT



8) LV_ALIGN_IN_LEFT_MID



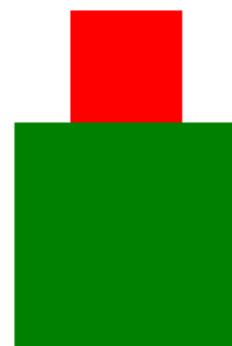
9) LV_ALIGN_IN_RIGHT_MID



10) LV_ALIGN_OUT_TOP_LEFT



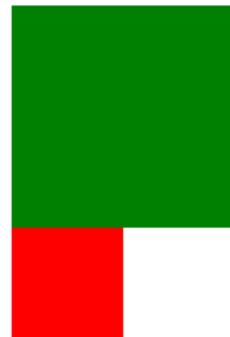
11) LV_ALIGN_OUT_TOP_MID



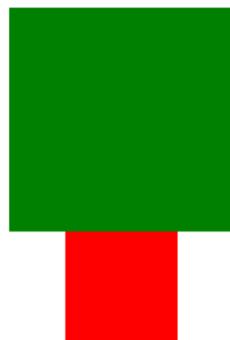
12) LV_ALIGN_OUT_TOP_RIGHT



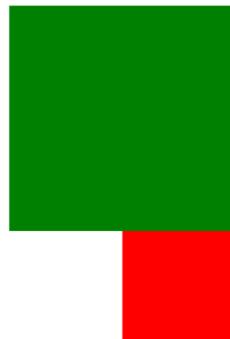
13) LV_ALIGN_OUT_BOTTOM_LEFT



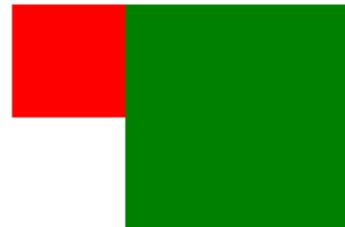
14) LV_ALIGN_OUT_BOTTOM_MID



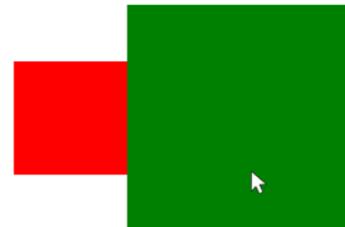
15) LV_ALIGN_OUT_BOTTOM_RIGHT



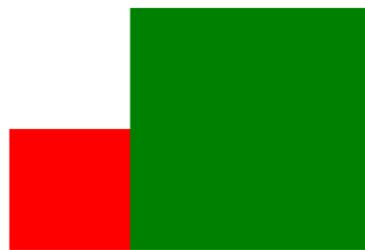
16) LV_ALIGN_OUT_LEFT_TOP



17) LV_ALIGN_OUT_LEFT_MID



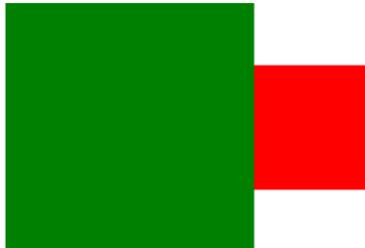
18) LV_ALIGN_OUT_LEFT_BOTTOM



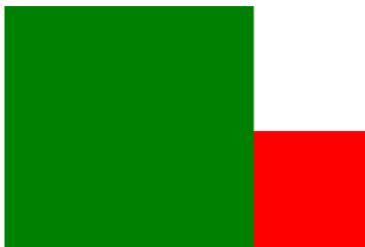
19) LV_ALIGN_OUT_RIGHT_TOP



20) LV_ALIGN_OUT_RIGHT_MID



21) LV_ALIGN_OUT_RIGHT_BOTTOM



2.2.17 设置对象(以其中心点)的对齐方式

```
void lv_obj_align_origo(lv_obj_t * obj, const lv_obj_t * base, lv_align_t align, lv_coord_t x_mod, lv_coord_t y_mod);
```

参数:

obj: 需要进行对齐的对象

base: 与哪一个对象进行对齐,即为基准或参考对象,如果传 NULL 的话,则默认以父对象为参考

align: 对齐方式,总共有 21 种

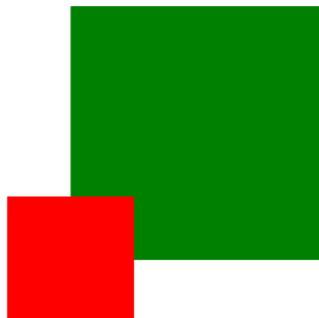
x_mod: 指定方式对齐之后,再进行 x 轴方向的偏移修正

y_mod: 指定方式对齐之后,再进行 y 轴方向的偏移修正

lv_obj_align_origo 和 lv_obj_align 的对齐原理是相同的,只不过 lv_obj_align_origo 接口是
[开发指南](#) www.alientek.com

以 obj 对象的中心点来跟 base 参考对象进行对齐的，下面只以 LV_ALIGN_OUT_BOTTOM_LEFT 对齐方式进行说明,同样绿色是 base 对象,红色是 obj 对象,x_mod 和 y_mode 的值都为 0

1) LV_ALIGN_OUT_BOTTOM_LEFT



2.2.18 重新对齐

```
void lv_obj_realign(lv_obj_t * obj);
```

参数:

obj: 对象句柄

此接口只有在 LV_USE_OBJ_REALIGN 宏使能的前提下才有效，而 LV_USE_OBJ_REALIGN 宏是在 lv_conf.h 文件中配置的,默认是使能了的, lv_obj_realign 的实现原理,就是基于最后一次的对齐参数(此参数保存在对象的 realign 属性中)再来调用一次 lv_obj_align 或者 lv_obj_align_origo 接口,说的再通俗一点那就是重复一下最后一次的对齐操作,这个接口是有应用场景的,比如一个含有 10 个字符大小的 lv_label 标签对象先调用 lv_obj_align 接口来与屏幕保持居中对齐,然后由于某种原因,标签对象含有的字符数增加,导致长度变大了,如果你还想让此标签对象与屏幕保持居中的话,那你只需要调用一次 lv_obj_realign 接口就可以了

2.2.19 是否使能自动重新对齐

```
void lv_obj_set_auto_realign(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: true 是使能自动对齐,false 是禁止自动对齐

此接口只有在 LV_USE_OBJ_REALIGN 宏使能的前提下才有效,如果使能自动重新对齐功能后,当对象的大小发生改变时,我们就不需要去手动调用 lv_obj_realign 接口来再次对齐了,因为 littleVGL 的内部已经帮我们完成了这个操作

2.2.20 设置对象的扩展可点击区域

```
void lv_obj_set_ext_click_area(lv_obj_t * obj, lv_coord_t left, lv_coord_t right, lv_coord_t top, lv_coord_t bottom);
```

参数:

obj: 对象句柄

left: 左边扩展长度

right: 右边扩展长度

top: 上边扩展长度

bottom: 下边扩展长度

2.2.21 设置对象的样式

```
void lv_obj_set_style(lv_obj_t * obj, const lv_style_t * style);
```

参数:

obj: 对象句柄

style: 样式

这里不过多介绍对象的样式,后面会有专门的章节来介绍的

2.2.22 通知对象它的样式发生了改变

```
void lv_obj_refresh_style(lv_obj_t * obj);
```

参数:

obj: 对象句柄

这里不过多介绍对象的样式,后面会有专门的章节来介绍的

2.2.23 向 littleVGL 通知样式发生了改变

```
void lv_obj_report_style_mod(lv_style_t * style);
```

参数:

style: 样式

这里不过多介绍对象的样式,后面会有专门的章节来介绍的

2.2.24 设置对象是否隐藏

```
void lv_obj_set_hidden(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否隐藏对象,true 是使能,false 是不使能

2.2.25 设置对象是否可以点击

```
void lv_obj_set_click(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否可以点击

2.2.26 是否使能对象在层级上置顶

```
void lv_obj_set_top(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否使能自动置顶

2.2.27 设置对象是否可以被拖拽

```
void lv_obj_set_drag(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否可以被拖拽

2.2.28 设置对象只能在指定方向上拖拽

```
void lv_obj_set_drag_dir(lv_obj_t * obj, lv_drag_dir_t drag_dir);
```

参数:

obj: 对象句柄

drag_dir: 指定的拖拽方向

2.2.29 是否使能对象的拖拽惯性滑动功能

```
void lv_obj_set_drag_throw(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否使能

这里讲一下什么叫拖拽惯性滑动功能,当我们在屏幕上按住一个对象,用力朝某个方向划一下,当手松开时,这个对象并不会立即停下来,而是由于惯性作用,会继续往这个方向移动一点距离

2.2.30 是否使能对象的父容器联动拖拽功能

```
void lv_obj_set_drag_parent(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否使能

如果使能了,那么当此对象被拖拽时,此对象的父对象也会被移动,它看起来像是父对象被拖拽了,不过要实现这一功能是有前提的,那就是 obj 的父对象得先使用 lv_obj_set_drag 接口使能了拖拽功能,如果父对象没有使能拖拽功能,而子对象却强行调用 lv_obj_set_drag_parent 接口的话,反而会导致子对象无法被拖拽了的

2.2.31 是否将事件转发给父对象

```
void lv_obj_set_parent_event(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否使能

如果使能了的话,那么当 obj 对象收到某事件时,也会将此事件转发一份给它的父对象

2.2.32 是否使能透明度功能

```
void lv_obj_set_opa_scale_enable(lv_obj_t * obj, bool en);
```

参数:

obj: 对象句柄

en: 是否使能

2.2.33 设置对象的透明度

```
void lv_obj_set_opa_scale(lv_obj_t * obj, lv_opa_t opa_scale);
```

参数:

obj: 对象句柄

opa_scale: 透明度值, 范围为 [0,255], 为 0(LV_OPA_TRANSP) 时是全透明, 为 255(LV_OPA_COVER) 时是完全不透明

2.2.34 设置对象的保护属性

```
void lv_obj_set_protect(lv_obj_t * obj, uint8_t prot);
```

参数:

obj: 对象句柄

prot: 保护属性, 可以是多个 lv_protect_t 类型进行或运算后的取值

2.2.35 清除对象的保护属性

```
void lv_obj_clear_protect(lv_obj_t * obj, uint8_t prot);
```

参数:

obj: 对象句柄

prot: 要被清除的保护属性, 可以是多个 lv_protect_t 类型进行或运算后的取值

2.2.36 设置对象的事件回调函数

```
void lv_obj_set_event_cb(lv_obj_t * obj, lv_event_cb_t event_cb);
```

参数:

obj: 对象句柄

event_cb: 事件回调函数

2.2.37 手动给对象发送事件

```
lv_res_t lv_event_send(lv_obj_t * obj, lv_event_t event, const void * data);
```

参数:

obj: 对象句柄

event: 要发送的事件名

data: 发送时携带的自定义数据, 通常置 NULL 就可以了

返回值:

LV_RES_OK 代表对象在事件中没有被删除, LV_RES_INV 代表对象在事件中被删除了

携带的自定义数据可以在它的事件回调函数中通过 lv_event_get_data 接口来获取, 另外 lv_event_send 接口是有一定的应用场景的, 比如说用来模拟关闭消息对话框, 见如下示意代码:

```
uint32_t btn_id = 0;  
lv_event_send(mbox, LV_EVENT_VALUE_CHANGED, &btn_id);
```

2.2.38 获取当前事件的自定义参数

```
const void * lv_event_get_data(void);
```

返回值:

返回当前事件的自定义参数

此接口一般放在事件回调函数中进行调用

2.2.39 获取对象所在的屏幕

```
lv_obj_t * lv_obj_get_screen(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回所在的屏幕对象

2.2.40 获取对象所在的屏幕

```
lv_disp_t * lv_obj_get_disp(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回所在的屏幕对象

这里稍微解释一下显示器的概念,littleVGL 是支持同时带载多个显示器的,一个显示器就得有一个其对应的显示缓冲区,一个显示器里面又可以有许多个屏幕对象,可以这么说吧,我们以后用 littleVGL 做的所有项目基本都是一个显示器的

2.2.41 获取父对象

```
lv_obj_t * lv_obj_get_parent(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回父对象

2.2.42 获取子对象(从尾端到首端)

```
lv_obj_t * lv_obj_get_child(const lv_obj_t * obj, const lv_obj_t * child);
```

参数:

obj: 对象句柄

child: 上一个子对象,传 NULL 的话,则是代表获取第一个子对象

返回值:

返回下一个子对象,如果返回 NULL,说明已经没有更多的子对象了

一个对象最多只能拥有一个父对象,但是可以拥有许多个子对象,使用 lv_obj_get_child 接口来获取某对象的所有子对象时,并不是一次性把所有的子对象全部返回出来的,调用一次只能返回一个子对象,它是通过多次迭代调用来把所有子对象一一获取出来的,为了更直观的理解,请看下面代码示意:

```
lv_obj_t * child;
child = lv_obj_get_child(parent, NULL); //获取第一个子对象
while(child) {
    //... 使用"子对象" 做一些事情
    child = lv_obj_get_child(parent, child); //然后进行循环迭代,获取下一个子对象
}
```

注: 这里的”首端”是指父对象第一次添加的子对象,”尾端”是指父对象最后一次添加的子对象

2.2.43 获取子对象(从首端到尾端)

```
lv_obj_t * lv_obj_get_child_back(const lv_obj_t * obj, const lv_obj_t * child);
```

参数:

obj: 对象句柄

child: 上一个子对象,传 NULL 的话,则是代表获取第一个子对象

返回值:

返回下一个子对象,如果返回 NULL,说明已经没有更多的子对象了

这个接口和 lv_obj_get_child 接口的使用是完全一样的,只不过就是获取方向不同

2.2.44 获取子对象的总数量

```
uint16_t lv_obj_count_children(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回子对象的总个数

这个子对象的总个数是只限制在儿子辈的

2.2.45 递归获取子对象的总数量

```
uint16_t lv_obj_count_children_recursive(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回子对象的总个数

这个是递归方式获取的,也就是说包括了儿子辈,孙子辈,...一直到最底端

2.2.46 获取对象的坐标

```
void lv_obj_get_coords(const lv_obj_t * obj, lv_area_t * cords_p);
```

参数:

obj: 对象句柄

cords_p: 用来存放坐标的

当然了,你还可以通过 lv_obj_get_x 和 lv_obj_get_y 接口来单独获取 x 或 y 坐标

2.2.47 获取对象的宽度

```
lv_coord_t lv_obj_get_width(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回对象的宽度

2.2.48 获取对象的高度

```
lv_coord_t lv_obj_get_height(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回对象的高度

2.2.49 获取对象的样式

```
const lv_style_t * lv_obj_get_style(const lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回对象的样式

2.2.50 获取对象及其所有祖先的类型

```
void lv_obj_get_type(lv_obj_t * obj, lv_obj_type_t * buf);
```

参数:

obj: 对象句柄

buf: 存放所有的类型,比如 buf.type[0] = "lv_btn", buf.type[1] = "lv_cont", buf.type[2] = "lv_obj"

2.2.51 设置对象的用户自定义数据

```
void lv_obj_set_user_data(lv_obj_t * obj, lv_obj_user_data_t data);
```

参数:

obj: 对象句柄

data: 用户自定义数据

这个接口是得在 LV_USE_USER_DATA 宏使能的前提下才生效的,而此宏是在 lv_conf.h 文件中配置的,默认是不使能的,同时 lv_obj_user_data_t 的具体数据类型也是 lv_conf.h 文件中配置的

2.2.52 获取对象的用户自定义数据

```
lv_obj_user_data_t lv_obj_get_user_data(lv_obj_t * obj);
```

参数:

obj: 对象句柄

返回值:

返回用户自定义数据

这个接口是得在 LV_USE_USER_DATA 宏使能的前提下才生效的,而此宏是在 lv_conf.h 文件中配置的,默认是不使能的,同时 lv_obj_user_data_t 的具体数据类型也是 lv_conf.h 文件中配置的

2.2.53 备注

还有一些比较简单的 API 接口我这里就不列出来了,通过看函数名就能知道大概啥意思了,这些接口基本都是 get 获取函数,用来获取对象的属性的

3.例程设计

3.1 功能简介

主要是先创建 obj1 和 obj2 俩个基本对象,在这里主要演示对象的位置,大小,对齐,样式等基本操作,然后通过按 KEY0 键,让 obj1 对象置顶,这里主要是演示 z 轴层级改变的操作,接着再按 KEY1 键,会在屏幕正中间动态创建一个 obj3 对象,在这里主要是演示对象的拷贝,透明度,拖拽,父对象更换等操作,最后再按 KEY2 键来删除 obj1 对象,这里主要是演示对象的删除操作

3.2 硬件设计

本例程所用到的硬件有:

- 1) 串口 1
- 2) KEY0,KEY1,KEY2 按键
- 3) 液晶屏

3.3 软件设计

我们在 GUI_APP 目录下新建 lv_obj_test.c 和 lv_obj_test.h 文件,然后 lv_obj_test.c 文件的内容如下:

```
#include "lv_obj_test.h"
#include "lvgl.h"
#include "delay.h"
#include "usart.h"
#include "key.h"

lv_obj_t * scr;
lv_obj_t * obj1 = NULL;
lv_obj_t * obj2 = NULL;
lv_obj_t * obj3 = NULL;
lv_style_t red_style;

//例程入口函数
void lv_obj_test_start()
{
    scr = lv_scr_act(); //获取当前活跃的屏幕对象
```

```
//先创建一个红色背景的样式,对于样式相关的代码,看不懂的话,没关系,后面会详  
//细讲解  
  
//先从系统自带的 lv_style_plain 样式拷贝过来,这样就不用对每个属性进行赋值了  
lv_style_copy(&red_style,&lv_style_plain_color);  
red_style.body.main_color = LV_COLOR_RED;  
red_style.body.grad_color = LV_COLOR_RED;  
  
//创建一个基本对象 1  
obj1 = lv_obj_create(scr,NULL);  
lv_obj_set_pos(obj1,20,20);//设置坐标位置  
lv_obj_set_size(obj1,100,100);//设置大小  
  
//创建一个基本对象 2,与对象 1 进行外部底下居中对齐,同时 y 轴向上偏移 10 个像素,  
//目的是为了让 obj2 有一部分压在 obj1 上,方便后面演示 z 轴层级改变的 API 接口  
obj2 = lv_obj_create(scr,NULL);  
lv_obj_set_size(obj2,50,50);  
lv_obj_set_style(obj2,&red_style);//设置新的样式  
lv_obj_align(obj2,obj1,LV_ALIGN_OUT_BOTTOM_MID,0,-20);  
  
}  
  
//按键处理  
//注意:请按照先按 KEY0 键,再按 KEY1 键,最后按 KEY2 键的顺序来观察实验现象  
void key_handler()  
{  
    u8 key = KEY_Scan(0);  
  
    if(key==KEY0_PRES)  
    {  
        //z 轴层级改变演示,将 obj1 对象进行置顶,有三种实现方式  
        #define Z_LAYER_MODE 1 //1,2,3 分别对应三种实现方式  
  
        #if(Z_LAYER_MODE==1)  
            if(obj1)  
                lv_obj_move_foreground(obj1);  
        #elif(Z_LAYER_MODE==2)  
            lv_obj_move_background(obj2);  
        #elif(Z_LAYER_MODE==3)  
            lv_obj_set_top(obj1,true);  
        #endif  
        printf("obj1 is on top layer\r\n");  
    }  
}
```

```
 }else if(key==KEY1_PRES)
{
    //动态创建一个对象 3,与屏幕居中对齐,然后更改对象 2 的父亲为对象 3
    obj3 = lv_obj_create(scr,obj1);//从 obj1 拷贝过来,减少一些属性的赋值
    //传 NULL,那么父对象就是参考对象
    lv_obj_align(obj3,NULL,LV_ALIGN_CENTER,0,0);
    lv_obj_set_drag(obj3,true);//设置 obj3 可以被拖拽
    lv_obj_set_drag_throw(obj3,true);//同时使能 obj3 的拖拽惯性滑动功能
    lv_obj_set_parent(obj2,obj3);//修改 obj2 的父对象为 obj3
    lv_obj_set_pos(obj2,10,10);//obj2 得重新设置一下坐标

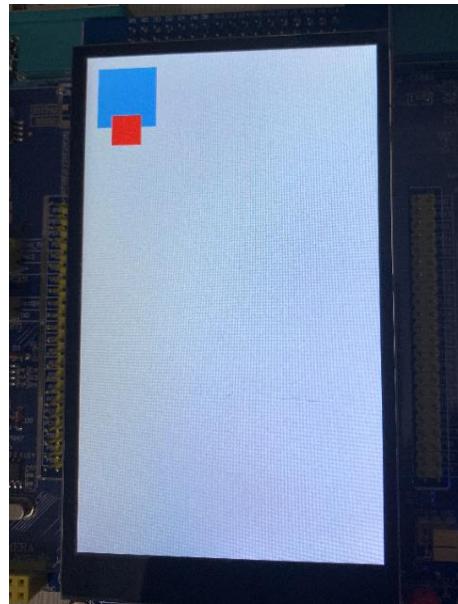
    lv_obj_set_opa_scale_enable(obj1,true);//先使能 obj1 的透明度功能
    lv_obj_set_opa_scale(obj1,100);//再设置 obj1 的透明度为 50

    //接下来,你可以试一试在屏幕上拖拽 obj3
    //设置对象 3 可以被拖拽,注意拖拽时,手指一定得按在 obj3 上,
    //不能按在 obj2 上,否则看不到正确的拖拽现象,虽然 obj2 是 obj3
    //的子对象,但是 obj2 是没有使能拖拽功能的
}else if(key==KEY2_PRES)
{
    if(obj1)
    {
        //删除 obj1 对象,有 2 种实现方式
#define DEL_MODE1

        #if(DEL_MODE==1)
            lv_obj_del(obj1);
        #elif(DEL_MODE==2)
            lv_obj_del_async(obj1);
        #endif
        obj1 = NULL;
        printf("obj1 is deleted\r\n");
    }
}
```

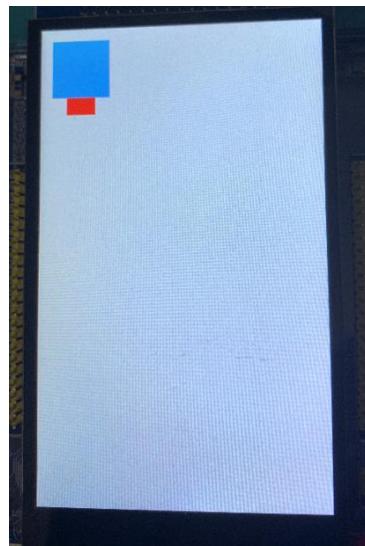
3.4 下载验证

1)开发板刚下载完代码后的状态



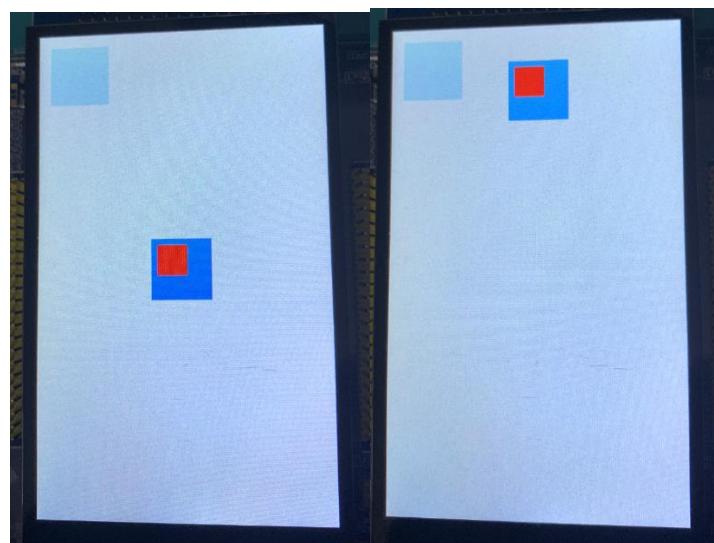
我们可以看到红色的 obj2 对象跟 obj1 对象是进行了 LV_ALIGN_OUT_BOTTOM_MID 对齐后,再接着 y 轴往上偏移 10 个像素,此时 obj2 是压在 obj1 上的

2)按下 KEY0 键后的状态



此时可以看到 obj1 对象是压在了 obj2 对象上

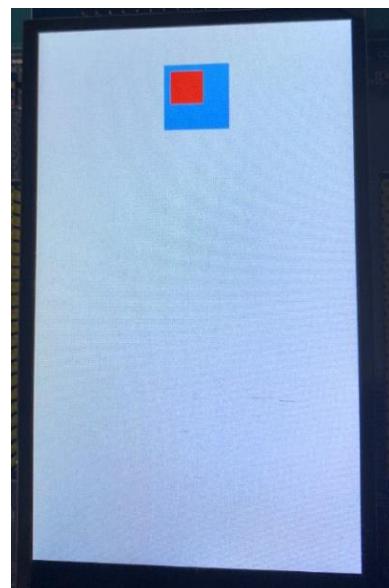
4)按下 KEY1 键后的状态



从左图中可以看到 obj3 对象处于屏幕正中间,同时 obj2 对象处于 obj3 父对象的里面,另外 obj1 对象的背景颜色也变淡了,这是因为 obj1 设置了 50 的透明度

从右图中可以看到 obj3 对象被拖拽了

5)按下 KEY2 键后的状态



从上面可以看到 obj1 对象已经从屏幕上消失了,因为 obj1 对象被删除了

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

正点原子 littleVGL 开发指南

lv_label 标签控件

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

lv_label 标签控件

1. 介绍

lv_label 标签控件可以说是 littleVGL 中使用最频繁的控件了,他的主要作用就是用来显示文本信息的,你可以在运行时的任何时候,使用 **lv_label_set_text(label, "New text")** 接口来动态修改文本内容,littleVGL 内部会重新为这个标签重新分配堆空间,当然了你也可以通过 **lv_label_set_static_text(label, char_array)** 这样的接口来引用一个外部的 `char_array` 文本指针,这样的好处就是 littleVGL 内部不会为这个文本内容分配堆空间,从而可以减少内存的使用,此标签控件支持换行,图标字体,部分文本重绘色等显示功能,同时针对长文本显示,它支持 6 种显示模式.

2. lv_label 的 API 接口

2.1 主要数据类型

2.1.1 长文本模式数据类型

```
enum {
    LV_LABEL_LONG_EXPAND, //自动扩展对象的大小来包裹文本内容

    //保持对象的宽度不变,当文本内容的宽度超过对象的宽度时会
    //自动换行,然后同时自动扩展对象的高度来包裹文本内容的高度
    LV_LABEL_LONG_BREAK,

    //保持对象的大小不变,当文本内容太长显示不下时,
    //会在文本末尾显示...三个点的省略号
    LV_LABEL_LONG_DOT,

    //保持对象的大小不变,当文本内容太长显示不下时,会自动循环向前向后滚动文本
    LV_LABEL_LONG_SCROLL,

    //保持对象的大小不变,当文本内容太长显示不下时,会自动循环环形滚动文本
    LV_LABEL_LONG_SCROLL_CIRC,

    LV_LABEL_LONG_CROP,   //保持对象大小不变,超过的文本内容将会被剪切掉
};

typedef uint8_t lv_label_long_mode_t;
```

2.1.2 文本内容对齐数据类型

```
enum {
    LV_LABEL_ALIGN_LEFT, //文本左对齐
    LV_LABEL_ALIGN_CENTER, //文本居中对齐
    LV_LABEL_ALIGN_RIGHT, //文本右对齐
};

typedef uint8_t lv_label_align_t;
```

2.1.3 标签样式数据类型

```
enum {
    LV_LABEL_STYLE_MAIN,
};

typedef uint8_t lv_label_style_t;
```

2.2 API 接口

2.2.1 创建标签

```
lv_obj_t * lv_label_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

parent: 指向父对象

copy: 此参数可选,表示创建新对象时,把 copy 对象上的属性值复制过来

返回值:

返回新创建出来的标签对象,如果为 NULL 的话,说明堆空间不足了

littleVGL 中的控件都有自己专属的创建 API 接口,虽然名字不同,但是他们的接口命名规范和含义是相同的

2.2.2 设置动态文本(字符串形式)

```
void lv_label_set_text(lv_obj_t * label, const char * text);
```

参数:

label: 标签对象

text: 新的文本内容,文本内容要是'\0'空字符结尾,如果传 NULL 的话,那么代表刷新当前文本内容

这个 API 接口是用来设置文本内容的,但是我特意加了”动态”俩个字来修饰,这是因为当用此 API 接口来设置文本时,它会把之前文本所占用的内存空间先给释放掉,然后为这个新文本内容重新分配一个相应大小的内存空间,所以即使外面的 text 指针被释放了,也不会影响此标签控件的显示

2.2.3 设置动态文本(数组形式)

```
void lv_label_set_array_text(lv_obj_t * label, const char * array, uint16_t size);
```

参数:

label: 标签对象

array: 新的文本内容,不需要是'\0'空字符结尾,如果传 NULL 的话,那么代表刷新当前文本内容

size: 传入的 array 数组的大小,单位为字节

这个 API 接口和 2.2.2 中的 lv_label_set_text 接口功能是一样的,只不过是传入文本的形式不一样,一个是以字符串形式,一个是以数组形式

2.2.4 设置静态文本(字符串形式)

```
void lv_label_set_static_text(lv_obj_t * label, const char * text);
```

参数:

label: 标签对象

text: 新的文本内容,文本内容要是'\0'空字符结尾,如果传 NULL 的话,那么代表刷新当前文本内容

这是以”静态”的方式设置文本内容,所谓的静态就是标签对象内部不会为这个文本内容分配内存空间来保存它,而只是引用了一下这个文本指针,好处就是在某些场合下,可以节省内存,坏处就是外部的 text 内容空间不能随意的被释放,否则会引起标签对象的显示出错

2.2.5 设置静态文本(数组形式)

```
void lv_label_set_array_text(lv_obj_t * label, const char * array, uint16_t size);
```

参数:

label: 标签对象

array: 新的文本内容,不需要是'\0'空字符结尾,如果传 NULL 的话,那么代表刷新当前文本内容

size: 传入的 array 数组的大小,单位为字节

这个 API 接口和 2.2.4 中的 lv_label_set_static_text 接口功能是一样的,只不过是传入文本的形式不一样,一个是以字符串形式,一个是以数组形式

2.2.6 设置长文本模式

```
void lv_label_set_long_mode(lv_obj_t * label, lv_label_long_mode_t long_mode);
```

参数:

label: 标签对象

long_mode: 长文本模式

此 API 接口很重要,而且也很奇妙,它跟调用时的位置有关(请看后面的例子 3),它的取值会影响到标签对象的大小,如果不设置的话,那么标签对象的长文本模式默认为 LV_LABEL_LONG_EXPAND,下面让我们来看一下每一种长文本模式对标签对象大小的影响:

1) LV_LABEL_LONG_EXPAND

当设置为此模式时,用 lv_obj_set_size 接口来设置大小是无效的,标签对象的宽和高只会随着文本的内容进行横向和纵向的扩展,默认是不会自动换行的,但是文本内容中可以插入'\n'字符来进行手动换行,请看下面 2 个例子(只给出关键示意代码)

例子 1:

```
lv_label_set_text(label1,"I am xiong jia yu");
```



图 2.2.6.1 例子 1 效果图

例子 2：

```
lv_label_set_text(label1,"I am xiong jia yu,Who are you?\nCan you tell me?");//使用' \n'换行
```

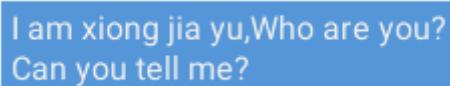


图 2.2.6.2 例子 2 效果图

2)LV_LABEL_LONG_BREAK

当设置为此模式时,用 lv_obj_set_size 接口来设置大小时,只有宽度是有效的,高度会随着文本的内容进行扩展,当文本的内容超过对象的指定宽度时,会进行自动换行的,请看下面的一个例子(只给出关键示意代码)

例子 3(正确的方式):

```
lv_label_set_long_mode(label1,LV_LABEL_LONG_BREAK);
lv_obj_set_size(label1,100,0);//宽度为 100 像素,高度是无效的,随意设置吧
lv_label_set_text(label1,"I amxiong jia yu,Who are you?Can you tell me?");
```

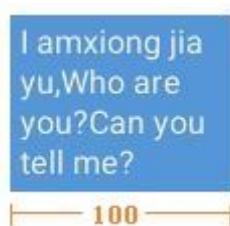


图 2.2.6.3 例子 3 效果图

前面我说过 lv_label_set_long_mode 接口的调用跟位置是有关系的,在这里 lv_label_set_long_mode 的调用必须放在 lv_obj_set_size 调用的前面,否则设置的宽和高是无效的,即下面这种方式是错误:

例子 3(错误的方式):

```
lv_obj_set_size(label1,100,0);//宽度为 100 像素,高度是无效的,随意设置吧
lv_label_set_long_mode(label1,LV_LABEL_LONG_BREAK);
lv_label_set_text(label1,"I amxiong jia yu,Who are you?Can you tell me?");
```

3)LV_LABEL_LONG_DOT

当设置为此模式时,用 lv_obj_set_size 接口来设置大小时,宽和高都是有效的,当文本的内容显示不下时,会在文本的末尾显示一个...省略号,同时具有自动换行的功能,请看下面的一个例子(只给出关键示意代码)

例子 4:

```
lv_label_set_long_mode(label1,LV_LABEL_LONG_DOT);
lv_obj_set_size(label1,100,50);
lv_label_set_text(label1,"I am xiong jia yu,Who are you?Can you tell me?");
```

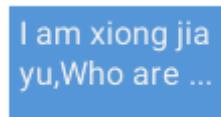


图 2.2.6.4 例子 4 效果图

注:默认情况下是显示...三个点的省略号,不过这个是可以通过 LV_LABEL_DOT_NUM 宏来配置的,此宏在 lv_label.h 头文件中,如下所示:

```
lv_label.c lv_label.h
20 #endif
21
22 #if LV_USE_LABEL != 0
23
24 #include "../lv_core/lv_obj.h"
25 #include "../lv_font/lv_font.h"
26 #include "../lv_font/lv_symbol_def.h"
27 #include "../lv_misc/lv_txt.h"
28 #include "../lv_draw/lv_draw.h"
29
30 ****
31 *      DEFINES
32 ****
33 #define LV_LABEL_DOT_NUM 3
34 #define LV_LABEL_POS_LAST 0xFFFF
35 #define LV_LABEL_TEXT_SEL_OFF 0xFFFF
36
```

图 2.2.6.5 LV_LABEL_DOT_NUM 宏

4)LV_LABEL_LONG_SCROLL

当设置为此模式时,用 lv_obj_set_size 接口来设置大小时,宽和高都是有效的,当文本的内容显示不下时,会进行向前向后的滚动显示,不具有自动换行的功能,请看下面的例子(只给出关键示意代码)

例子 5:

```
lv_label_set_long_mode(label1,LV_LABEL_LONG_SCROLL);
lv_obj_set_size(label1,100,50);
lv_label_set_text(label1,"I am xiong jia yu,Who are you?Can you tell me?");
```



图 2.2.6.5 例子 5 效果图

5) LV_LABEL_LONG_SROLL_CIRC

和 LV_LABEL_LONG_SROLL 模式的基本功能是一样的,唯一区别就是 LV_LABEL_LONG_SROLL_CIRC 不是向前向后滚动,而是环形滚动,请看下面的例子(只给出关键示意代码)

例子 6:

```
lv_label_set_long_mode(label1,LV_LABEL_LONG_SROLL_CIRC);
lv_obj_set_size(label1,100,50);
lv_label_set_text(label1,"I am xiong jia yu,Hello");
```



图 2.2.6.6 例子 6 效果图

6) LV_LABEL_LONG_CROP

当设置为此模式时,用 lv_obj_set_size 接口来设置大小时,宽和高都是有效的,当文本的内容显示不下时,超出的部分会被直接剪切掉,不具有自动换行的功能,请看下面的例子(只给出关键示意代码)

例子 7:

```
lv_label_set_long_mode(label1,LV_LABEL_LONG_CROP);
lv_obj_set_size(label1,100,50);
lv_label_set_text(label1,"I am xiong jia yu,Hello");
```



图 2.2.6.7 例子 7 效果图

2.2.7 设置文本对齐方式

```
void lv_label_set_align(lv_obj_t * label, lv_label_align_t align);
```

参数:

label: 标签对象

align: 水平方向上的文本对齐方式

记住,要想让标签的文本内容具有对齐的效果,那么必须得先保证标签对象具有指定的宽度,请看下面例子

```
lv_obj_t *src = lv_scr_act(); //获取屏幕对象  
lv_obj_t *label1 = lv_label_create(src, NULL); //创建对象  
lv_label_set_long_mode(label1, LV_LABEL_LONG_CROP); //设置长文本模式,不能设置为  
//LV_LABEL_LONG_EXPAND 模式,因为其指定不了宽度  
lv_obj_set_pos(label1, 20, 20); //设置坐标  
lv_obj_set_size(label1, 100, 50); //设置宽和高  
lv_label_set_text(label1, "Hello"); //设置文本内容  
lv_label_set_style(label1, LV_LABEL_STYLE_MAIN, &lv_style_plain_color); //设置具有背景色  
lv_label_set_body_draw(label1, true); //设置绘制背景  
lv_label_set_align(label1, LV_LABEL_ALIGN_CENTER); //设置文本居中对齐
```

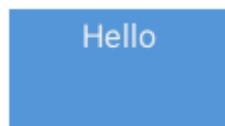


图 2.2.7.1 居中对齐效果

2.2.8 是否使能文本重绘色功能

```
void lv_label_set_recolor(lv_obj_t * label, bool en);
```

参数:

label: 标签对象

en: 是否使能

使能之后,可以让标签的部分文本显示出不同的颜色,即一个标签里可以含有多种不同颜色的文本,这在其他的 GUI 库中,一般是办不到的,使用格式为: #十六进制颜色值 文本#,注意了颜色值和文本之间至少得有一个空格,请看下面例子(只给出关键代码)

```
lv_label_set_recolor(label1,true); //先得使能文本重绘色功能  
lv_label_set_text(label1,"#ff0000 red#,#00ff00 green#,#0000ff blue#"); //使用了 3 次重绘色
```



图 2.2.8 文本重绘色效果

2.2.9 是否使能背景绘制功能

```
void lv_label_set_body_draw(lv_obj_t * label, bool en);
```

参数:

label: 标签对象

en: 是否使能

默认情况下,lv_label 标签对象是没有背景的,即透明的,但是我们可以通过这个接口和样式使标签对象具有背景,请看如下例子

```
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&lv_style_plain_color); //设置主背景的样  
式为  
//lv_style_plain_color  
lv_label_set_body_draw(label1,true); //使能背景绘制  
lv_label_set_text(label1,"Hello world");
```



图 2.2.9.1 带有背景的效果

2.2.10 设置动画时的速度

```
void lv_label_set_anim_speed(lv_obj_t * label, uint16_t anim_speed);
```

参数:

label: 标签对象

anim_speed: 动画速度,单位为 px/sec(一秒多少个像素)

这个接口主要用来设置文本滚动(LV_LABEL_LONG_SCROLL/SCROLL_CIRC 模式下)时的速度

2.2.11 设置样式

```
static inline void lv_label_set_style(lv_obj_t * label, lv_label_style_t type, const lv_style_t * style)
```

参数:

label: 标签对象

type: 哪一个部件的样式,目前就 LV_LABEL_STYLE_MAIN 主背景部件这一个可选值

style: 样式

2.2.12 插入文本

```
void lv_label_ins_text(lv_obj_t * label, uint32_t pos, const char * txt);
```

参数:

label: 标签对象

pos: 插入的位置,0 代表从最前面插入, LV_LABEL_POS_LAST 代表从最后面插入

txt: 要插入的文本

这个接口有一点需要注意,那就是必须得保证之前的文本是动态设置,而不能是静态文本,因为我们知道插入动作是修改内存的操作,而如果之前的文本是通过静态方式设置的话,那么此内存有可能是常量区(用 `const` 关键字修饰),常量区内存是不能修改的

2.2.13 剪切文本

```
void lv_label_cut_text(lv_obj_t * label, uint32_t pos, uint32_t cnt);
```

参数:

label: 标签对象

pos: 剪切的起始位置,从 0 开始

cnt: 要剪切的字符数量

剪切文本也可以理解成删除指定区域的文本,这个接口也同样有一点需要注意,那就是必须得保证之前的文本是动态设置,而不能是静态文本

2.2.14 备注

还有一些比较简单的 API 接口,我这里就不列出来了,通过看函数名就能知道大概啥意思了,这些接口基本都是 get 获取函数,用来获取对象的属性的

3.例程设计

3.1 功能简介

创建 label1 和 label2 俩个标签,label1 标签主要是用来做标题的,显示 label2 标签的当前长文本模式,而 label2 标签主要是用来演示 6 大长文本模式的,为 label2 标签注册了事件回调函数,通过点击 label2 标签,来循环切换 label2 标签的长文本模式,如果按下 KEY0 键的话,则是加大 label2 标签的动画速度,可以在 LV_LABEL_LONG_SCROLL 和 LV_LABEL_LONG_SCROLL_CIRC 模式下观察到明显的动画快慢效果,如果按下 KEY1 键,则是在 label1 文本的最前面插入 OK 字符串,如果按下 KEY2 键,则是删除 label1 文本最前面的 2 个字符

3.2 硬件设计

本例程所用到的硬件有:

- 1) 串口 1
- 2) KEY0,KEY1,KEY2 按键
- 3) 液晶屏

3.3 软件设计

在 GUI_APP 目录下新建 lv_label_test.c 和 lv_label_test.h 俩个文件, lv_label_test.c 文件的内容如下:

```
#include "lv_label_test.h"
#include "lvgl.h"
#include "delay.h"
#include "uart.h"
#include "key.h"

lv_obj_t* label1;
lv_obj_t* label2;
//模式标题
const char * const MODE_STR[] = {"EXPAND","BREAK","DOT","SROLL","SROLL_CIRC","CR
OP"};

//事件处理
void event_handler(lv_obj_t * obj, lv_event_t event)
{
```

```
static lv_label_long_mode_t mode = LV_LABEL_LONG_EXPAND;

if(obj==label2)
{
    if(event==LV_EVENT_RELEASED)//触摸释放事件
    {
        lv_label_set_long_mode(label2,mode);//设置新的长文本模式
        if(mode==LV_LABEL_LONG_EXPAND)//自动扩展模式
        {
            lv_label_set_text(label2,"EXPAND:0123456789ABCDEFGHIJKLMN\nI am xiong jia yu\nWho are you?");
        }
        else if(mode==LV_LABEL_LONG_BREAK)//自动换行模式
        {
            lv_obj_set_width(label2,100);
            lv_label_set_text(label2,"BREAK:Auto to break line");
        }
        else if(mode==LV_LABEL_LONG_DOT)//自动显示省略号模式
        {
            lv_obj_set_size(label2,100,40);
            lv_label_set_text(label2,"DOT:too long,0123456789ABCDEFGHIJKLMN");
        }
    }
    else if(mode==LV_LABEL_LONG_SROLL)//自动前后滚动模式
    {
        lv_obj_set_size(label2,100,40);
        lv_label_set_text(label2,"SROLL:KEY0 to add speed");
    }
    else if(mode==LV_LABEL_LONG_SROLL_CIRC)//自动环形滚动模式
    {
        lv_obj_set_size(label2,100,40);
        lv_label_set_text(label2,"SROLL_CIRC:KEY0 to add speed");
    }
    else if(mode==LV_LABEL_LONG_CROP)//剪切模式
    {
        lv_obj_set_size(label2,100,40);
        lv_label_set_text(label2,"CROP:0123456789ABCDEF");
    }
    lv_obj_realign(label2);//因为 label2 的大小发生了改变,为了让 label2
                           //继续与屏幕保持居中对齐,可以调用重对齐接口
    lv_label_set_text(label1,MODE_STR[mode]);
    printf("current long mode:%d\r\n",mode);
    //切换到下一个模式
    mode++;
    if(mode>LV_LABEL_LONG_CROP)
        mode = LV_LABEL_LONG_EXPAND;
}
}
```

```
//例程入口函数
void lv_label_test_start()
{
    lv_obj_t* scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //创建 label1 标签,用来显示 label2 标签的长文本模式
    label1 = lv_label_create(scr,NULL); //创建标签
    lv_label_set_long_mode(label1,LV_LABEL_LONG_BREAK); //设置长文本模式
    //设置宽度,一定得放在 lv_label_set_long_mode 的后面,否则不起作用的
    lv_obj_set_width(label1,160);
    lv_label_set_recolor(label1,true); //使能文本重绘色功能
    lv_label_set_text(label1,"#ff0000 Title:#mode"); //设置文本,带有颜色重绘
    lv_label_set_align(label1,LV_LABEL_ALIGN_CENTER); //文本居中对齐
    //设置主背景样式
    lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&lv_style_plain_color);
    lv_label_set_body_draw(label1,true); //使能背景绘制
    //注意:设置与屏幕的对齐方式,这个接口也是跟调用位置有关系的,最好放到后面调
    //用,因为放的太前调用的话,
    //在那时标签对象的大小可能还是未知的,此时 lv_obj_align 接口就没办法算出对齐
    //之后的坐标,从而也就
    //达不到我们所要的对齐效果
    lv_obj_align(label1,NULL,LV_ALIGN_IN_TOP_MID,0,20);

    //创建 label2 标签
    label2 = lv_label_create(scr,NULL); //创建标签
    //设置主背景样式
    lv_label_set_style(label2,LV_LABEL_STYLE_MAIN,&lv_style_plain_color);
    lv_label_set_body_draw(label2,true); //使能背景绘制
    lv_obj_set_click(label2,true); //使能点击功能
    lv_obj_set_event_cb(label2,event_handler); //设置事件回调函数
    lv_label_set_text(label2,"Please click me!"); //设置文本
    lv_obj_align(label2,NULL,LV_ALIGN_CENTER,0,0); //设置其与屏幕居中对齐
}

//按键处理
//注意:请按照先按 KEY0 键,再按 KEY1 键,最后按 KEY2 键的顺序来观察实验现象
void key_handler()
{
    u8 key = KEY_Scan(0);

    if(key==KEY0_PRES)
    {
```

```
//调节动画速度
lv_label_set_anim_speed(label2,lv_label_get_anim_speed(label2)+5);
printf("anim speed:%d\r\n",lv_label_get_anim_speed(label2));
}else if(key==KEY1_PRES)
{
    //在 label1 文本的最前面插入 OK 字符串
    lv_label_ins_text(label1,0,"OK");
}else if(key==KEY2_PRES)
{
    //删除 label1 文本最前面的 2 个字符
    lv_label_cut_text(label1,0,2);
}
}
```

3.4 下载验证

请根据功能简介,一一验证实验效果,下面给出一张代码刚下载完后的初始界面效果

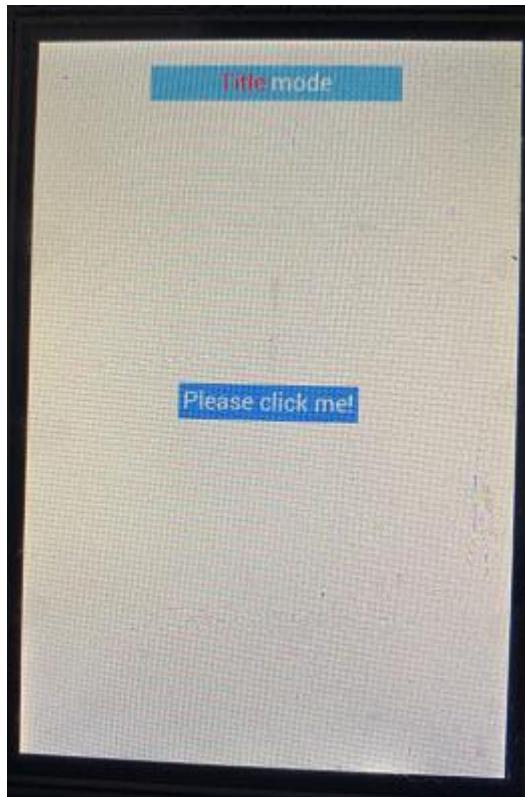


图 3.4.1 初始界面效果

注:如果是电阻屏的话,在开机之前可以先按住 KEY0 键不放,接着在开机(或按复位键也行),可以先进入到电阻屏触摸校准程序,校准完成后会自动进入到演示例程,此注意事项对以后的所有例程同样适用

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

正点原子 littleVGL 开发指南

lv_style 样式

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

lv_style 样式

1. 介绍

所谓的样式就是用来修饰控件 UI 美观性的,随着科技的日新月异,GUI 库的发展也是迅速崛起,传统的老风格界面 UI 已经不再符合我们的审美需求了,littleVGL 正是基于此考虑,推出了样式系统,利用样式可以对 UI 界面进行重绘和重用,利用多个不同的样式来形成 Theme 主题系统,所以样式在 littleVGL 中有着很重要的地位,请大家务必熟练掌握.对于 lv_obj 基础对象而言,每一个对象都会有一个 lv_style 样式,但对于一个其他的控件(比如 lv_btn 按钮控件)而言,可能会拥有多个样式,因为一个稍微复杂点的控件可能是由多个子部件组成的,而每一个子部件可能都需要相应的样式来修饰,所以表面上看来,这个控件拥有了多个样式.

其实 lv_style 样式的结构还是挺复杂的,因为它旨在一个结构来囊括所有的控件对象,所以对于某些控件而言,必定会存在一些冗余而用不到的属性,但 littleVGL 从方便性和实用性去考虑,做出这点牺牲是完全值得的,一个样式主要是由 body 背景,text 文本,image 图片,line 线条等 4 部分组成的,然后其中每一部分又由很多属性组成.对于细节的展开,我们放到下面的“主要数据类型”中来讲解.

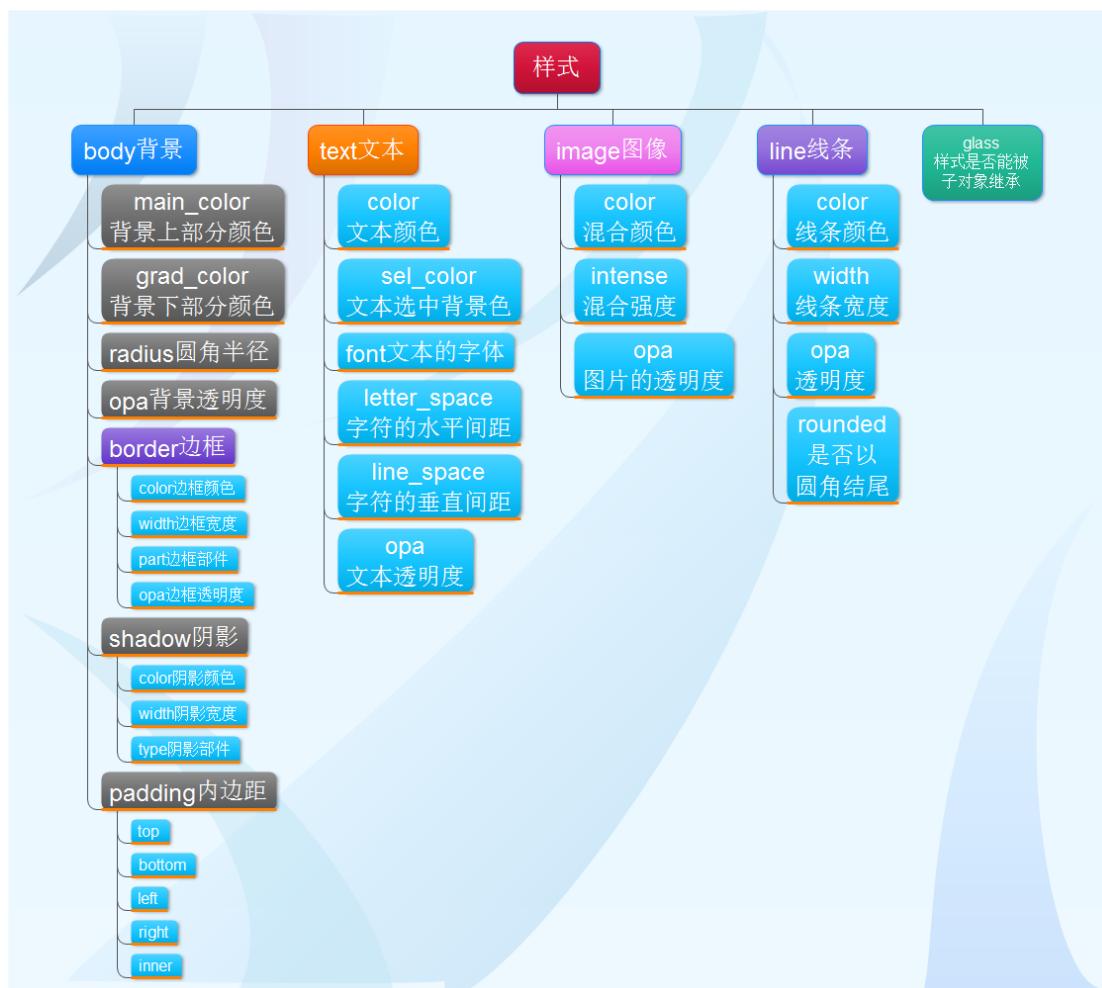


图 1.1 样式的结构

2.lv_style 的 API 接口

2.1 主要数据类型

2.1.1 边框部件数据类型

```
enum {
    LV_BORDER_NONE      = 0x00, //无边框
    LV_BORDER_BOTTOM    = 0x01, //底边框
    LV_BORDER_TOP       = 0x02, //上边框
    LV_BORDER_LEFT      = 0x04, //左边框
    LV_BORDER_RIGHT     = 0x08, //右边框
    LV_BORDER_FULL      = 0x0F, //四条边框
    LV_BORDER_INTERNAL = 0x10, //用于类矩阵的控件,如矩阵按钮
};

typedef uint8_t lv_border_part_t;
```

用于描述到底绘制哪几条边框,这些值可以进行 OR 或操作,进行组合赋值

2.1.2 阴影类型数据类型

```
enum {
    LV_SHADOW_BOTTOM = 0, //只绘制底部阴影
    LV_SHADOW_FULL,      //绘制所有边的阴影
};

typedef uint8_t lv_shadow_type_t;
```

2.1.3 样式句柄数据类型

```
typedef struct
{
    uint8_t glass : 1;
    struct
    {
        lv_color_t main_color;
        lv_color_t grad_color;
        lv_coord_t radius;
        lv_opa_t opa;

        struct
        {
            lv_color_t color;
            lv_coord_t width;
            lv_border_part_t part;
        };
    };
};
```

```
    lv_opa_t opa;
} border;

struct
{
    lv_color_t color;
    lv_coord_t width;
    lv_shadow_type_t type;
} shadow;

struct
{
    lv_coord_t top;
    lv_coord_t bottom;
    lv_coord_t left;
    lv_coord_t right;
    lv_coord_t inner;
} padding;

} body;

struct
{
    lv_color_t color;
    lv_color_t sel_color;
    const lv_font_t * font;
    lv_coord_t letter_space;
    lv_coord_t line_space;
    lv_opa_t opa;
} text;

struct
{
    lv_color_t color;
    lv_opa_t intense;
    lv_opa_t opa;
} image;

struct
{
    lv_color_t color;
    lv_coord_t width;
    lv_opa_t opa;
    uint8_t rounded : 1;
```

```
    } line;  
} lv_style_t;
```

这个结构体是重点,务必搞懂,它的属性成员非常多,通常一般都是通过相应的 API 接口来修改句柄属性的,但这个样式句柄比较特殊,他没有相应的 API 接口,我们就是直接来修改属性的,下面开始讲解一下每个属性的大概含义

2.1.3.1 glass 继承

通常情况下,如果子对象的样式为 NULL 的话,那么此子对象将会从它的父对象那里继承样式,但是如果你设置 my_style.glass = 1;的话,那么 my_style 这个样式就不能被子对象给继承了

2.1.3.2 body 背景

body 属性也是一个结构体,它又由以下属性组成

2.1.3.2.1 main_color

这个是用来设置背景的上半部分颜色的,结合 grad_color 属性可以实现对象的纯色背景 (main_color 和 grad_color 值相等) 和从上到下的渐变色背景 (main_color 和 grad_color 值不相等),请看下面例子(只给出关键示意代码)

```
static lv_style_t my_style;//定义一个样式  
lv_style_copy(&my_style,&lv_style_plain_color);//拷贝样式  
my_style.body.main_color = LV_COLOR_RED;//上半部分为红色  
my_style.body.grad_color = LV_COLOR_GREEN;//下半部分为绿色  
lv_obj_set_style(obj1,&my_style);//设置样式
```

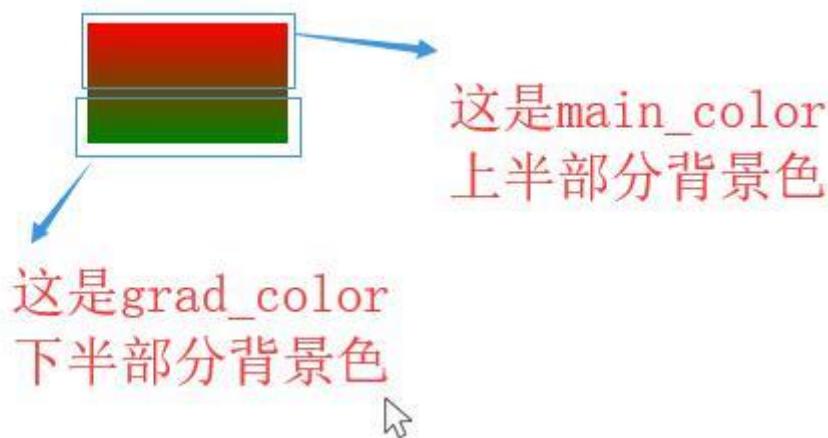


图 2.1.3.2.1.1 渐变色背景效果

2.1.3.2.2 grad_color

grad_color 和 main_color 的作用是差不多的,只不过 grad_color 是指下半部分的背景

2.1.3.2.3 radius

这是是用来设置圆角半径的,请看下面的例子

```
lv_obj_set_size(obj1,100,60);
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color);
my_style.body.radius = 30;//设置圆角半径
lv_obj_set_style(obj1,&my_style);
```



图 2.1.3.2.3.1 圆角矩形效果

2.1.3.2.4 opa

这个是用来设置背景透明度的,请看下面例子(只给出关键示意代码)

```
lv_label_set_text(label1,"I am a label");//设置 label1 的文本
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color);
my_style.body.main_color = LV_COLOR_RED;
my_style.body.grad_color = LV_COLOR_RED;
my_style.body.opa = 160;
lv_obj_set_style(obj1,&my_style);
```



图 2.1.3.4.1 透明度效果

2.1.3.2.5 border

这个是用来设置边框的,其内部结果如下:

```
struct
{
    lv_color_t color; //边框颜色
    lv_coord_t width; //边框宽度
    lv_border_part_t part; //绘制哪几条边框
    lv_opa_t opa; //边框的透明度
} border;
```

请看下面例子(只给出关键示意代码)

```
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color);
my_style.body.border.color = LV_COLOR_RED;
my_style.body.border.width = 5;
my_style.body.border.part = LV_BORDER_FULL;//四条边框都要显示,或者也可以这样
//LV_BORDER_TOP|LV_BORDER_LEFT 只显示上边和左边的边框
my_style.body.border.opa = LV_OPA_COVER;//完全不透明
lv_obj_set_style(obj1,&my_style);
```



图 2.1.3.2.5.1 边框效果

2.1.3.2.6 shadow

用来设置阴影效果的,其内部结构如下:

```
struct
{
    lv_color_t color;//阴影的颜色
    lv_coord_t width;//阴影的宽度
    lv_shadow_type_t type;//阴影的类型,是四周全部阴影还只是底部阴影
} shadow;
```

请看下面例子(只给出关键示意代码)

```
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color);
my_style.body.shadow.color = LV_COLOR_RED;//阴影颜色
my_style.body.shadow.width = 20;//阴影宽度
```

```
my_style.body.shadow.type = LV_SHADOW_FULL;//四周都要有阴影  
lv_obj_set_style(obj1,&my_style);
```



图 2.1.3.2.6.1 阴影效果

2.1.3.2.7 padding

这个是用来设置内边距的,其内部结构如下:

```
struct  
{  
    lv_coord_t top;//上边距  
    lv_coord_t bottom;//下边距  
    lv_coord_t left;//左边距  
    lv_coord_t right;//右边距  
    lv_coord_t inner;//内部之间的间隙  
} padding;
```

请看下面例子(只给出关键示意代码)

```
static lv_style_t my_style;  
lv_style_copy(&my_style,&lv_style_plain_color);  
my_style.body.padding.top = 100;  
my_style.body.padding.left = 80;  
my_style.body.padding.bottom = 60;  
my_style.body.padding.right = 40;  
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&my_style);
```



图 2.1.3.2.7.1 没有设置 padding 时的默认效果

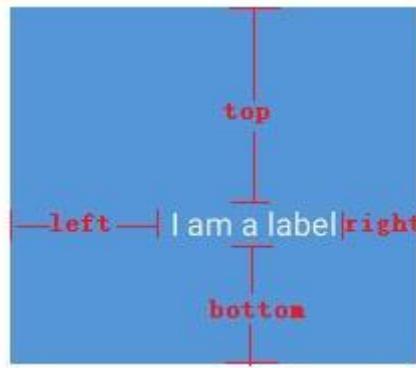


图 2.1.3.2.7.2 设置了 padding 时的效果

2.1.3.3 text 文本

这个是用来设置文本效果的,其内部结构如下:

```
struct
{
    lv_color_t color;//文本颜色
    lv_color_t sel_color;//选中文本的背景颜色
    const lv_font_t * font;//所用的字体
    lv_coord_t letter_space; //字符之间的水平距离
    lv_coord_t line_space; 字符之间的垂直距离
    lv_opa_t opa;//文本的透明度
} text;
```

请看下面例子(只给出关键示意代码)

```
lv_label_set_text(label1,"letter_space\nline_space");
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color);
my_style.text.color = LV_COLOR_RED;//字体为红色
my_style.text.font = &lv_font_roboto_28;//使用大号字体,必须得在 lv_conf.h 中使能相应
//的字体,这里是 LV_FONT_ROBOTO_28 宏
my_style.text.opa = LV_OPA_COVER;//完全不透明
my_style.text.letter_space = 10;//水平间距
my_style.text.line_space = 20;//垂直间距
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&my_style);
```

letter_space
line_space

图 2.1.3.3.1 不设置 text 时的默认效果



图 2.1.3.3.2 设置了 text 时的效果

2.1.3.4 image 图像

这个是用来设置图片样式的,其内部结构如下:

```
struct
{
    lv_color_t color;//混合的颜色
    lv_opa_t intense;//混合的强度
    lv_opa_t opa;//图片的透明度
} image;
```

简单来说就是把图片的每一个像素点数据与指定的混合颜色按照指定的混合强度进行混合,产生新的像素点数据然后显示出来

2.1.3.5 line 线条

```
struct
{
    lv_color_t color;//线条的颜色
    lv_coord_t width;//线条的宽度
    lv_opa_t opa;//线条的透明度
    uint8_t rounded : 1; //线条的末尾是否用圆角来绘制
} line;
```

请看下面的例子:

```
static const lv_point_t p[] = {{0, 0}, {100, 0}};//至少 2 点才能构成一条线
lv_obj_t * line1 = lv_line_create(src, NULL);//创建线对象
lv_obj_set_pos(line1, 50, 50);
lv_line_set_points(line1, p, 2);//设置坐标点
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color);
my_style.line.color = LV_COLOR_RED;//设置线的颜色
my_style.line.width = 10;//设置线的宽度
my_style.line.rounded = 1;//线末尾用圆角来绘制
my_style.line.opa = LV_OPA_COVER;//完全不透明
lv_line_set_style(line1,LV_LINE_STYLE_MAIN,&my_style);
```



图 2.1.3.5.1 非圆角的线条



图 2.1.3.5.2 圆角的线条

2.2 API 接口

2.2.1 样式初始化

```
void lv_style_init(void);
```

这个接口是用来初始化自带的系统样式的,由 littleVGL 内部自行完成调用,我们不需要理会,littleVGL 自带了 13 个系统样式,供其内部使用,当然了也可以供我们用户外部使用,这 13 个样式如下:

```
lv_style_t lv_style_scr;
lv_style_t lv_style_transp;
lv_style_t lv_style_transp_fit;
lv_style_t lv_style_transp_tight;
lv_style_t lv_style_plain;
lv_style_t lv_style_plain_color;
lv_style_t lv_style_pretty;
lv_style_t lv_style_pretty_color;
lv_style_t lv_style_btn_rel;
lv_style_t lv_style_btn_pr;
lv_style_t lv_style_btn_tgl_rel;
lv_style_t lv_style_btn_tgl_pr;
lv_style_t lv_style_btn_ina;
```

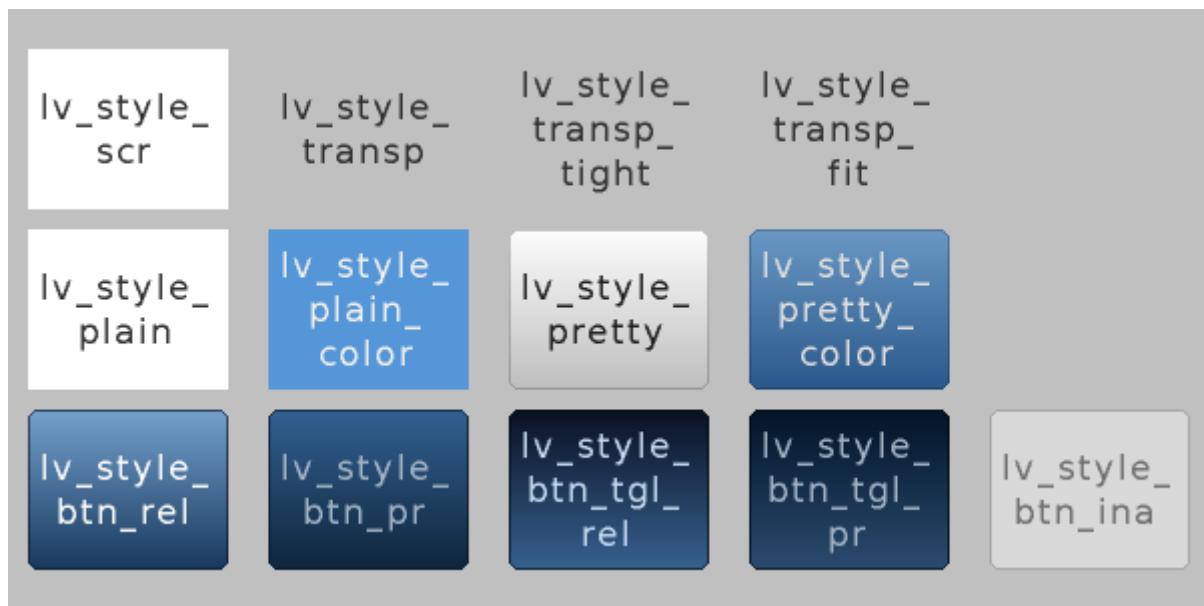


图 2.2.1.1 系统样式效果图

2.2.2 样式拷贝

```
void lv_style_copy(lv_style_t * dest, const lv_style_t * src);
```

参数:

dest: 目的样式,即拷贝给谁

text: 源样式,即从那里拷贝

这个 API 接口见名知意,很简单,目的就是减少相同样式的重复赋值

2.2.3 样式混合

```
void lv_style_mix(const lv_style_t * start, const lv_style_t * end, lv_style_t * res, uint16_t ratio);
```

参数:

start: 起始样式

text: 终止样式

res: 混合后的结果样式

ratio: 混合因子,范围为:[0,256]

这个接口主要是 littleVGL 内部使用,用于 Animations 动画,当然了,我们也可以在外部使用

2.2.4 使用样式

```
//基础对象设置样式  
void lv_obj_set_style(lv_obj_t * obj, const lv_style_t * style);  
//其他控件设置样式,其中 xxx 代表未知的控件名(label,btn 等等)  
void lv_xxx_set_style(lv_obj_t *obj, lv_xxx_style_t type, const lv_style_t * style);
```

参数:

obj: 对象句柄

type: 这个参数对基础对象无效,对其他控件有效,用于设置控件上某个部件的样式

style: 设置的样式,必须得是静态的或全局的或堆上分配的

2.2.5 刷新样式(作用于单个对象)

```
void lv_obj_refresh_style(lv_obj_t * obj);
```

参数:

obj: 对象句柄

通知 obj 对象,其所使用的样式以发生了改变,请及时更新界面

2.2.6 刷新样式(作用于多个对象)

```
void lv_obj_report_style_mod(lv_style_t * style);
```

参数:

style: 要被刷新的样式

通知已使用了 style 样式的所有对象,此样式已发生了改变,请及时更新界面

3.例程设计

3.1 功能简介

利用样式来创建一个自定制的对话框,对话框由一个带有透明度的灰色遮罩层,2 个背景(上面的为黑色,下面的为白色),一个标题,一个内容,一个取消按钮,一个确定按钮等元素组成,当点击取消或者确定按钮时,关闭掉对话框,另外还在屏幕上创建了一个 label,当点击此 label 时打开对话框

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下新建 lv_style_test.c 和 lv_style_test.h 俩个文件, lv_style_test.c 文件的内容如下:

```
#include "lv_style_test.h"
#include "lvgl.h"
#include "key.h"
#include "uart.h"

lv_obj_t* dialog = NULL;
lv_obj_t* title_label,* cancel,* ok;

lv_obj_t* dialog_create(lv_obj_t* parent);

//事件回调函数
static void event_handler(lv_obj_t* obj,lv_event_t event)
{
    if(event==LV_EVENT_PRESSED||event==LV_EVENT_RELEASED||event==LV_EVENT_PRESS_LOST)
    {
        lv_style_t* style = (lv_style_t*)lv_label_get_style(obj,LV_LABEL_STYLE_MAIN);
        if(event==LV_EVENT_PRESSED)
        {
            //处理逻辑
        }
    }
}
```

```
//让按钮看上去有点击的效果
style->body.radius = 10;
style->body.opa = LV_OPA_50;
}else{
    //恢复松手后的状态
    style->body.radius = 0;
    style->body.opa = LV_OPA_COVER;
}

lv_obj_refresh_style(obj);//通知对象,其所使用的样式发生了改变
}

if(obj==title_label)
{
    if(event==LV_EVENT_RELEASED)
    {
        if(dialog==NULL)//再次打开对话框
            dialog = dialog_create(lv_scr_act());
    }
    }else if(obj==cancel||obj==ok)
    {
        //松手事件,其中 LV_EVENT_PRESS_LOST 事件是指侧滑出对象的可视区了
        if(event==LV_EVENT_RELEASED||event==LV_EVENT_PRESS_LOST)
        {
            if(dialog)
            {
                //删除对话框
                lv_obj_del(dialog);
                dialog = NULL;
            }
        }
    }
}

//创建一个自定义的对话框
lv_obj_t* dialog_create(lv_obj_t* parent)
{
    //对话框的宽度
#define DIALOG_WIDTH 180

    //对话框的高度
#define DIALOG_HEIGHT 150
```

```
//对话框的底部按钮栏高度(白色背景区域)
#define DIALOG_BOTTOM_HEIGHT 40

//内边距,也就是按钮与对话框边框的距离
#define DIALOG_BOTTOM_PADDING 10

//1.创建一个带有半透明效果的灰色遮罩层
lv_obj_t* gray_layer = lv_obj_create(parent,NULL);//创建对象
lv_obj_set_pos(gray_layer,0,0);//设置坐标
//与父对象相同大小
lv_obj_set_size(gray_layer,lv_obj_get_width(parent),lv_obj_get_height(parent));
static lv_style_t gray_layer_style;//必须得加 static
lv_style_copy(&gray_layer_style,&lv_style_plain_color);//样式拷贝
gray_layer_style.body.main_color = LV_COLOR_GRAY;//上半部分背景色
gray_layer_style.body.grad_color = LV_COLOR_GRAY;//下半部分背景色
gray_layer_style.body.opa = LV_OPA_80;//透明度
lv_obj_set_style(gray_layer,&gray_layer_style);//设置样式

//2.创建对话框的上面背景(黑色)
//注意,为了保持控件的一体性,这里的父对象应该是传 gray_layer,而不是 parent
lv_obj_t* top_bg = lv_obj_create(gray_layer,NULL);
lv_obj_set_size(top_bg,DIALOG_WIDTH,DIALOG_HEIGHT);
lv_obj_align(top_bg,NULL,LV_ALIGN_CENTER,0,0);//居中对齐
static lv_style_t top_bg_style;
lv_style_copy(&top_bg_style,&lv_style_plain_color);
top_bg_style.body.main_color = LV_COLOR_MAKE(0x39, 0x3C, 0x4A);
top_bg_style.body.grad_color = LV_COLOR_MAKE(0x39, 0x3C, 0x4A);
top_bg_style.body.radius = 10;//圆角半径
// top_bg_style.body.shadow.color = LV_COLOR_BLACK;//阴影效果,加了好像变丑了
// top_bg_style.body.shadow.type = LV_SHADOW_FULL;
// top_bg_style.body.shadow.width = 8;
lv_obj_set_style(top_bg,&top_bg_style);

//3.创建对话框的下面背景(白色)
lv_obj_t* bottom_bg = lv_obj_create(gray_layer,NULL);
lv_obj_set_size(bottom_bg,DIALOG_WIDTH,DIALOG_BOTTOM_HEIGHT);
//与 top_bg 进行对齐
lv_obj_align(bottom_bg,top_bg,LV_ALIGN_IN_BOTTOM_MID,0,0);
static lv_style_t bottom_bg_style;
lv_style_copy(&bottom_bg_style,&lv_style_plain_color);
bottom_bg_style.body.main_color = LV_COLOR_WHITE;
bottom_bg_style.body.grad_color = LV_COLOR_WHITE;
lv_obj_set_style(bottom_bg,&bottom_bg_style);
```

```
//4.创建对话框的标题
lv_obj_t* title = lv_label_create(gray_layer,NULL);
lv_label_set_text(title,"Title");
lv_obj_align(title,top_bg,LV_ALIGN_IN_TOP_MID,0,10);

//5.创建对话框的内容
lv_obj_t* content = lv_label_create(gray_layer,NULL);
lv_label_set_text(content,"This is a dialog\nDo you like it?");
lv_obj_align(content,top_bg,LV_ALIGN_CENTER,0,-12);

//6.创建取消按钮,先用 label 对象来模拟
cancel = lv_label_create(gray_layer,NULL);
//要想有固定的大小,必须得先设置好长文本模式
lv_label_set_long_mode(cancel,LV_LABEL_LONG_CROP);
lv_obj_set_size(cancel,(DIALOG_WIDTH-DIALOG_BOTTOM_PADDING*4)/2,DIALOG_BOTTOM_HEIGHT-DIALOG_BOTTOM_PADDING*2);//设置固定的大小
lv_label_set_text(cancel,"Cancel");//设置文本
lv_label_set_align(cancel,LV_LABEL_ALIGN_CENTER);//设置文本居中对齐
lv_obj_align(cancel,bottom_bg,LV_ALIGN_IN_LEFT_MID,DIALOG_BOTTOM_PADDING,0)
;

lv_label_set_body_draw(cancel,true);
static lv_style_t cancel_style;
lv_style_copy(&cancel_style,&lv_style_plain_color);
cancel_style.body.main_color = LV_COLOR_WHITE;
cancel_style.body.grad_color = LV_COLOR_WHITE;
cancel_style.body.border.width = 1;//边框的宽度
cancel_style.body.border.color = LV_COLOR_MAKE(0xBD,0xBA,0xBD);//边框的颜色
cancel_style.body.border.part = LV_BORDER_FULL;//四条边框全部绘制
cancel_style.text.color = LV_COLOR_MAKE(0x63,0x65,0x63);
lv_label_set_style(cancel,LV_LABEL_STYLE_MAIN,&cancel_style);
lv_obj_set_click(cancel,true);//使能点击
lv_obj_set_event_cb(cancel,event_handler);//注册事件回调函数

//7.创建确定按钮
ok = lv_label_create(gray_layer,cancel);//直接从 cancel 拷贝过来
lv_label_set_text(ok,"Ok");//设置文本
lv_obj_align(ok,bottom_bg,LV_ALIGN_IN_RIGHT_MID,-DIALOG_BOTTOM_PADDING,0);
static lv_style_t ok_style;
lv_style_copy(&ok_style,&lv_style_plain_color);
ok_style.body.main_color = LV_COLOR_MAKE(0x31,0xAA,0xFF);
ok_style.body.grad_color = LV_COLOR_MAKE(0x31,0xAA,0xFF);
ok_style.text.color = LV_COLOR_WHITE;
lv_label_set_style(ok,LV_LABEL_STYLE_MAIN,&ok_style);
```

```
return gray_layer;
}

//例程入口
void lv_style_test_start()
{
    lv_obj_t* src = lv_scr_act(); //获取当前的屏幕对象

    title_label = lv_label_create(src, NULL); //创建一个 label,当点击它时,打开对话框
    lv_label_set_text(title_label, "click to open dialog"); //设置文本
    lv_obj_align(title_label, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);
    lv_label_set_body_draw(title_label, true); //绘制背景
    //设置样式
    lv_label_set_style(title_label, LV_LABEL_STYLE_MAIN, &lv_style_plain_color);
    lv_obj_set_click(title_label, true); //使能点击
    lv_obj_set_event_cb(title_label, event_handler); //注册事件回调函数

    dialog = dialog_create(src); //创建对话框
}
```

3.4 下载验证

代码下载完成之后,即可以看到如下的界面:



图 3.4.1 初始界面效果

当你点击按钮时,会看到按钮有按下的效果,主要是通过监听按钮按下时,修改其样式中的 opa 透明度和 radius 圆角半径,当按钮松手时,再次还原其样式

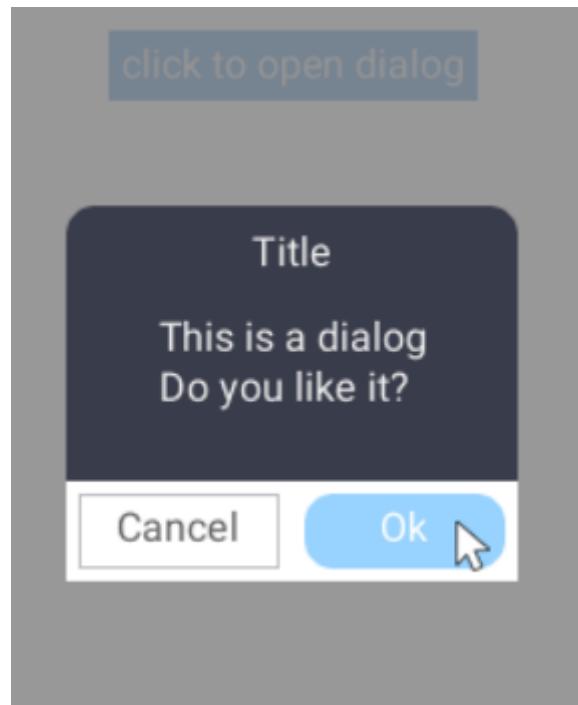


图 3.4.2 按钮按下时的界面效果

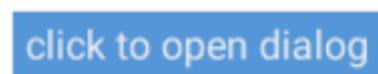


图 3.4.3 关闭对话框时的界面效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_font 字体

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

lv_font 字体

1. 介绍

在 littleVGL 中的字体功能也是非常强大的,支持最高 8bpp(在老版本中存在)的抗锯齿,另外还有 1bpp,2bpp,4bpp 三个值可选,实现灵活性配置,选择越大的 bpp 值时,要求的 flash 存储资源也是成倍的增加的,比如 4bpp 的存储消耗是 1bpp 存储消耗的 4 倍,除了存储消耗变大之外,bpp 值越大,在界面上进行字体渲染时,绘制速度也会越慢,当然了,bpp 值越大,也是有好处的,那就是绘制出来的字体边缘越平滑,没有毛刺,使我们产品的 UI 界面看上去更高大上,所以在实际项目中,我们应该根据硬件平台的实际性能来选出一个比较合适的 bpp 值,一般的项目都是默认采用 4bpp 就可以了,littleVGL 的字体还支持 unicode 编码显示,注意这里所谓的 unicode 编码是一种泛指能显示所有字符的编码概念,并不是指实际的 unicode 编码,而是指 UTF-8 编码,所以大家不要搞混淆了,littleVGL 在 UTF-8 编码的支持下,可以实现显示全球所有的字符.最后要说的一个功能那就是它还支持”图标字体”,可能大家一听,有点懵,这是在”web 前端”中经常用到的一种技术,可以看出来 littleVGL 的作者涉猎范围很广呀,下面让我们开始更进一步的探究它的细节.

1.1 如何启用 UTF-8 编码

littleVGL 自身支持 2 种编码,一种是 ASCII 编码,此编码只支持英文字符的显示,另外一种也就是我们前面说到的 UTF-8 编码,此编码可以支持全球所有字符的显示,像我们要显示中文,显示图标字体等,那我们就必须得选择 UTF-8 编码了,至于选择哪一个编码是由 lv_conf.h 头文件中的 LV_TXT_ENC 宏来配置的,见如下:

```
//启用 UTF-8 编码,LV_TXT_ENC_ASCII 是启用 ASCII 编码  
#define LV_TXT_ENC LV_TXT_ENC_UTF8
```

littleVGL 默认就是启用了 UTF-8 编码的,另外这里有一点需要注意的是当 littleVGL 启用 UTF-8 编码之后,为了我们开发的方便性,最好把我们的集成开发工具的编码也改成 UTF-8,弄成一致之后,我们就可以在编辑器中以字面量的方式直接写入我们想要显示的文本了,如下面代码所示:

```
//想显示啥,就直接写啥,当然了,前提是你的字库中有中国这两个字  
lv_label_set_text(label1, " text:中国" );
```

如果你不想把集成开发工具的编码弄成 UTF-8 也是可以的,只不过就是麻烦了点,那么你想要显示中国这两个字时,你只能以转义字符的形式来书写了,如下面代码所示:

```
#define ZHONG      "\xE4\xB8\xAD"    //中字的 UTF-8 编码  
#define GUO        "\xE5\x9B\xBD"    //国字的 UTF-8 编码  
lv_label_set_text(label1, " text:" ZHONG GUO);
```

这种方式的前提是你必须要知道字符的 UTF-8 编码,这个也简单,因为我们有专门的工具来转换,就在你们买的开发板资料盘中,大致路径为“软件资料\软件\中英文字符编码查询”,另外这种转义字符的形式也是很重要的,像我们后面要介绍的“图标字体”只能通过这种方式来实

现显示,下面我就以 Keil 集成开发环境为例,教大家怎么设置为 UTF-8 编码,先点击  图标,然后弹出如下对话框:

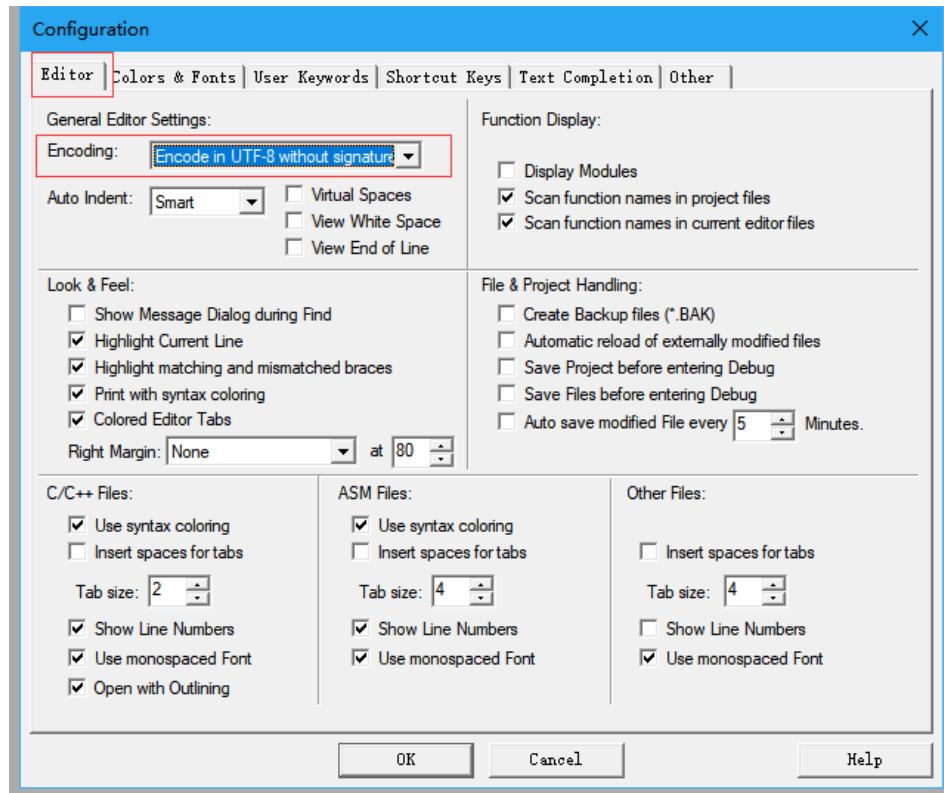


图 1.1.1 Keil 中设置 UTF-8 编码

1.2 啥叫图标字体

这首先是从 web 前端中流行起来的一种技术,见名知意,它是一个图标,按照我们以往的思维,图标一般都是指图片对不对?但是它在 littleVGL 中,即可以当成字体来显示,也可以当成图片来显示,两者都可以喔,不过这种图标有一个限制那就是它只能显示单色,果然是继承了字体的特性呀,littleVGL 系统给我们自带了许多个这样的常用小图标,如下图所示(左边是显示效果,右边是这个图标的名称):

- ♫ LV_SYMBOL_AUDIO
- 📺 LV_SYMBOL_VIDEO
- ☰ LV_SYMBOL_LIST
- ✓ LV_SYMBOL_OK
- ✗ LV_SYMBOL_CLOSE
- ⟳ LV_SYMBOL_POWER
- ⚙ LV_SYMBOL_SETTINGS
- 🗑 LV_SYMBOL_TRASH
- 🏡 LV_SYMBOL_HOME
- ⬇ LV_SYMBOL_DOWNLOAD
- 🖲 LV_SYMBOL_DRIVE
- ⟳ LV_SYMBOL_REFRESH
- ◀ LV_SYMBOL_MUTE
- 🔉 LV_SYMBOL_VOLUME_MID
- 🔊 LV_SYMBOL_VOLUME_MAX
- 🖼 LV_SYMBOL_IMAGE
- ✎ LV_SYMBOL_EDIT
- ◀ LV_SYMBOL_PREV
- ▶ LV_SYMBOL_PLAY
- ⏸ LV_SYMBOL_PAUSE
- ⏹ LV_SYMBOL_STOP
- ▶ LV_SYMBOL_NEXT
- ⏏ LV_SYMBOL_EJECT
- < LV_SYMBOL_LEFT
- > LV_SYMBOL_RIGHT
- ➕ LV_SYMBOL_PLUS
- ➖ LV_SYMBOL_MINUS
- ⚠ LV_SYMBOL_WARNING
- 🔀 LV_SYMBOL_SHUFFLE
- ˄ LV_SYMBOL_UP
- ˅ LV_SYMBOL_DOWN
- 🔁 LV_SYMBOL_LOOP
- 📁 LV_SYMBOL_DIRECTORY
- 📤 LV_SYMBOL_UPLOAD
- 📞 LV_SYMBOL_CALL
- ✂ LV_SYMBOL_CUT
- _COPY LV_SYMBOL_COPY
- 💾 LV_SYMBOL_SAVE
- ⚡ LV_SYMBOL_CHARGE
- 🔔 LV_SYMBOL_BELL
- ⌨ LV_SYMBOL_KEYBOARD
- 📍 LV_SYMBOL_GPS
- 📄 LV_SYMBOL_FILE
- 📶 LV_SYMBOL_WIFI
- 🔋 LV_SYMBOL_BATTERY_FULL
- 🔋 LV_SYMBOL_BATTERY_3
- 🔋 LV_SYMBOL_BATTERY_2
- 🔋 LV_SYMBOL_BATTERY_1
- 🔋 LV_SYMBOL_BATTERY_EMPTY
- Bluetooth LV_SYMBOL_BLUETOOTH

图 1.2.1 littleVGL 自带的图标字体效果

2. 如何使用字体

不论是使用 littleVGL 系统自带的字体,还是使用我们自己创建的字体,他们的使用方法都是一样的,为了大家更容易接受,我先给大家介绍怎么使用字体,后面我再给大家介绍怎么创建字体并使用它,我们就先以系统自带的字体为例来讲解怎么使用它,littleVGL 内置了 5 个字库,如下所示:

```
lv_font_roboto_12
lv_font_roboto_16
lv_font_roboto_22
lv_font_roboto_28
lv_font_unscii_8
```

其中的 `lv_font_roboto_12`, `lv_font_roboto_16`, `lv_font_roboto_22`, `lv_font_roboto_28` 都是 Roboto 字体,只不过是细分出了 4 个不同的字体大小,这四个字库全都是 4bpp 抗锯齿的,这四个字库中每一个字库除了包含所有的可见英文字符外,还都包含了一套相同的图标字符,也就是我们前面所说的图标字体,这套图标字符有一个自己的字体名为 FontAwesome ,然后 `lv_font_unscii_8` 是一种非常好的单色屏字体,是 1bpp 的,相当于无抗锯齿的功能,另外 `lv_font_unscii_8` 字库只包含英文字符,不包含上面所说的图标字符.

为了节省存储空间,这 5 个字库中,默认情况下只有 `lv_font_roboto_16` 字库是被使能了的,如果想使能其他的字体或者想设置默认的字体等,就必须得在 `lv_conf.h` 头文件中进行配置,相关的配置宏如下:

```
#define LV_FONT_ROBOTO_12      0    //是否使能 lv_font_roboto_12 字库
#define LV_FONT_ROBOTO_16      1    //是否使能 lv_font_roboto_16 字库,默认被使能了
#define LV_FONT_ROBOTO_22      0    //是否使能 lv_font_roboto_22 字库
#define LV_FONT_ROBOTO_28      0    //是否使能 lv_font_roboto_28 字库
#define LV_FONT_UNSCII_8       0    //是否使能 lv_font_unscii_8 字库

//这个是用来申明我们自己创建的字体的, 如果没有创建字体,那么可以留空
#define LV_FONT_CUSTOM_DECLARE

//把 lv_font_roboto_16 定义为默认字体
#define LV_FONT_DEFAULT  &lv_font_roboto_16

#define LV_FONT_FMT_TXT_LARGE  0    //当你的字库系统非常庞大时,比如字符数
                                //超过 10000 个,或者遇到某些莫名的错误时,那么你可以使能它
```

这里还有一点是关于系统自带的图标字符,他们全部申明在 `lv_symbol_def.h` 头文件中,相对路径为 `GUI\lvgl\src\lv_font`,可以自行查看图标名,或者你直接查看图 1.2.1 也是没问题的,对于字体的使用,可以概括为如下三步:

1)申明字体

如果是自己创建的字体,那么在使用前,就必须得先申明字体,否则在其他的文件中就会报找不到字体的错误,这跟变量申明是一模一样的原理,如果是系统自带的字体,那么就可以不用申明了,因为 littleVGL 内部已经帮我们申明好了,申明字体是用 LV_FONT_DECLARE 宏来申明的,如下所示:

```
LV_FONT_DECLARE(my_font_1); //申明 my_font_1 字体
```

其实 LV_FONT_DECLARE 宏的定义是下面这样的:

```
#define LV_FONT_DECLARE(font_name) extern lv_font_t font_name;
```

为了避免在每个地方都要去申明一次字体,我们可以直接利用 lv_conf.h 文件中的 LV_FONT_CUSTOM_DECLARE 配置宏来进行全局申明,如下所示:

```
#define LV_FONT_CUSTOM_DECLARE LV_FONT_DECLARE(my_font_1) \
                                LV_FONT_DECLARE(my_font_2)
```

2)在样式中设置要使用的字体

直接给出示意代码,这样大家看的更明白

```
static lv_style_t label_style;
lv_style_copy(&label_style,&lv_style_plain_color); //样式拷贝
label_style.text.font = &lv_font_roboto_16; //设置字体
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&label_style); //给标签对象设置样式
```

3)使用字体中的字符

常见的有下面这几种方式

```
lv_label_set_text(label1,"Hello"); //纯字符方式
lv_label_set_text(label1,"Hello" LV_SYMBOL_OK); //字符和图标组合方式
//多个图标组合方式,注意图标与图标之间至少得需要一个空格
lv_label_set_text(label1, LV_SYMBOL_OK LV_SYMBOL_CLOSE);
```



图 2.1 纯字符方式



图 2.2 字符和图标组合方式



图 2.3 多个图标组合方式

3. 如何创建自己的字体

如果想要创建自己的字体,那么就得借助官方提供的字体在线转换工具,它提供了图形操作界面,使用简单,此转换工具的在线网址为: <https://littlevgl.com/ttf-font-to-c-array>,对于国内用户来说,此在线转换工具的缺点就是打开网页速度和转换速度都太慢了,真的急死人喔,有耐心的朋友可以慢慢等哈,打开网页之后,效果如下:

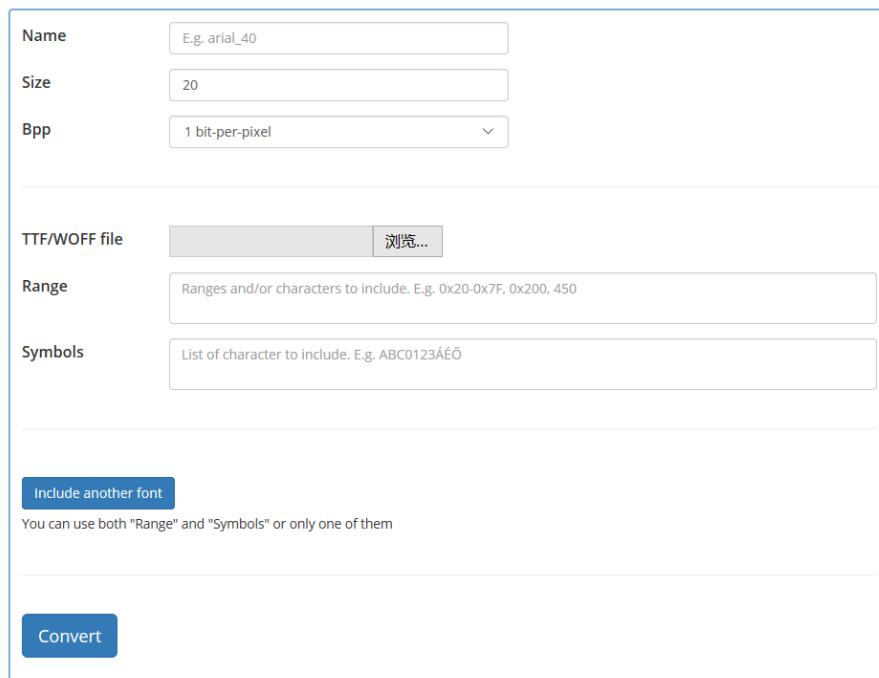


图 3.1 字体在线转换工具

littleVGL 的作者也是考虑到网速慢的原因,因此也推出了离线版本的转换工具,github 链接为: https://github.com/littlevgl/lv_font_conv,这是用 node.js 写的,没有图形化界面,需要敲命令行,比上面的在线转换工具可能要复杂点,但是它的优点非常的明显,那就是转换速度快,因此我打算给大家介绍怎么使用离线版本的转换工具,学会了难的,再去搞定图形化界面的在线转换工具,那是分分钟的事哈

4. 离线字体转换工具的使用

离线工具是用 node.js 开发出来的,所以我们需要先安装 node.js 的运行环境,安装 node.js 的过程我就不细说了,网上有很多关于这方面的帖子,我这里给出一个网上的参考教程:
<https://www.runoob.com/nodejs/nodejs-install-setup.html>,在安装完 node.js(会附带把 npm 包管理器给装好)之后,我们接下来要安装 lv_font_conv 离线转换工具了,操作很简单,先打开 cmd 命令窗口,然后直接输入 `npm i lv_font_conv -g` 命令,回车运行就会把 lv_font_conv 给安装好,装完之后,我们来讲一下 lv_font_conv 的主要命令行参数

--bpp: 抗锯齿大小,可选值为 1-4

--size: 字体的大小,实际就是指字符的高度

-o(或者--output): 输出路径,比如为

```
C:\Users\Administrator\Desktop\my_font_heiti_30.c
```

--format: 输出格式,可选值有 dump,bin,lvg1,我们只用 lvg1 就行了

--font: ttf/woff/woff2 字体文件的路径

-r(或--range): 所需字符的 unicode 编码范围,可选值为单个字符,字符范围,加可选的映射地址,首先你得必须保证你所需要的字符范围在--font 指定的字体文件中能找到,-r 命令行参数可以在一条命令中多次出现,下面例子所示:

```
-r 0xF450 单个字符,十六进制和十进制都行  
-r 0xF450-0xF470 字符范围  
-r '0xF450=>0x81' 单个字符加映射地址  
-r '0xF450-0xF470=>0x81' 字符范围加映射地址  
-r 0xF450 -r 0xF451-0xF470 一下使用 2 次-r 命令行参数  
-r 0xF450,0xF451-0xF470 用单个-r 表示多种可选值
```

上面所说的映射地址只有在多个字体进行合并时才会用得到,比如前面所说的 lv_font_robo_16 字体,那就是 Roboto 字体文件和 FontAwesome 图标字体文件合并而成的

--symbols: 和-r 的作用差不多,都是用来指示所要用到的字符,不过他们的表达形式不一样,

对--symbols 而言,它是接受字符的字面量形式,而-r 是接受字符的编码形式,

--symbols 和-r 可以同时使用,也可以只使用它们其中的一个,请看如下例子:

```
--symbols 0123456789 中 ABCD 国 EFG
```

--no-compress: 禁止进行 RLE 压缩,我们在生成字体时,请禁止进行压缩

上面介绍的都是重要的命令行参数,至于剩下的--autohint-off, --autohint-strong,

--force-fast-kern-format, --no-prefilter, --no-kerning, --full-info

这些命令行参数都是不太重要的,自己可以进入 lv_font_conv 的 github 地址自行了解.

下面给出一条完整的创建命令示例(复制到 cmd 命令窗口运行的):

```
lv_font_conv --no-compress --format lvg1 --font C:\Users\fish\Desktop\heiti.ttf -o  
C:\Users\fish\Desktop\my_font.c --bpp 4 --size 30 --symbols 我是熊家余 -r 0x20-0x7F
```

5. 创建普通字体

创建一个字体之前,我们首先要完成 2 点,第一点就是要获取到相应的 ttf/woff/woff2 字体文件,第二点就是要明确我们需要这个字体文件中的那些字符,对于 ttf/woff/woff2 字体文件我们可以从网上去下载,或者直接从系统自带的字体文件中拷贝过来,对于 Windows 系统而言,它的字体文件一般都存放在 C:\Windows\Fonts 目录下,下面我们以一个例子来进行讲解.

假如我们现在要创建一个 30 像素大小的普通字库(之所以称为普通,是因为它里面不含有图标字体),我准备用黑体来取字符,这个黑体的 ttf 文件我是从 Windows 系统自带的字体目录下拷贝过来的,然后直接放到桌面上,接着改名为 heiti.ttf,假如我们这个字库只需要 0-9 十个数字和正点原子四个汉字就可以了,也就是总共 14 个字符,采用 4bpp 抗锯齿,然后我们构建出用于创建此字库的命令,如下所示:

```
lv_font_conv --no-compress --format lvgl --font C:\Users\fish\Desktop\heiti.ttf -o C:\Users\fish\Desktop\my_font_heiti_30.c --bpp 4 --size 30 --symbols 正点原 -r 0x30-0x39 -r 0x5B50
```

其中 0x5B50 是子字的 unicode 编码,见下图所示:



图 3.2.1 子字的 unicode 编码

此编码转换工具在我们提供的资源当中可以找到,我之所以不把子字和正点原三个汉字写到一起去,是因为我想给你们多演示几种取字符范围的方式,然后复制上面的命令,粘贴到 cmd 命令窗口中去,如下图所示:

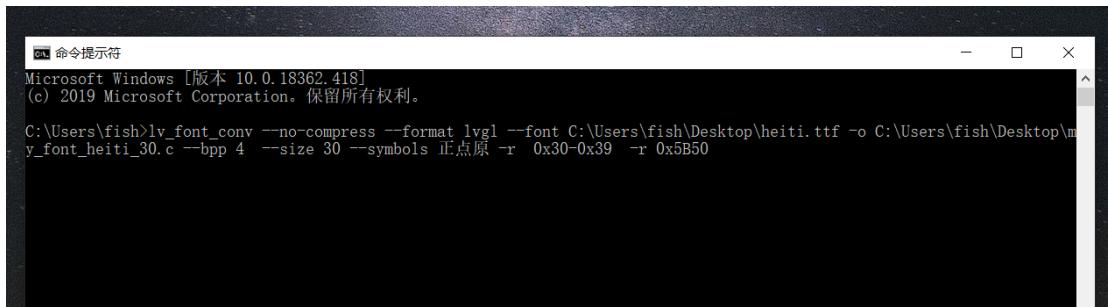


图 3.2.2 命令运行窗口

粘贴完成之后,直接回车运行,几秒钟后,就可以在桌面上看到刚创建出来的 my_font_heiti_30.c 这个字体文件,再接着我们把这个 my_font_heiti_30.c 文件拷贝到我们 Keil 项目的 GUI_APP 目录下面,接着用 Keil 打开项目,把这个.c 文件导入到 GUI_APP 分组下,此时还需要注意 2 个小细节,第一个就是把 Keil 的编码改成 UTF-8,第二个就是 littleVGL 得启用 UTF-8 编码,最后我们再来写一段简单的测试代码,如下所示:

```
LV_FONT_DECLARE(my_font_heiti_30); //申明字体

//测试例程入口
void lv_font_test_start()
{
    lv_obj_t* scr = lv_scr_act();

    lv_obj_t* label1 = lv_label_create(scr, NULL);
    static lv_style_t style1;
    lv_style_copy(&style1, &lv_style_plain_color);
    style1.text.font = &my_font_heiti_30; //在样式中使用这个字体
    lv_label_set_style(label1, LV_LABEL_STYLE_MAIN, &style1);
    lv_label_set_text(label1, "正点原子 123456"); //设置文本
    lv_label_set_body_draw(label1, true);
    lv_obj_align(label1, NULL, LV_ALIGN_CENTER, 0, 0);
}
```

运行起来之后的效果如下图所示:



图 3.2.3 运行效果图

6. 创建图标字体

虽然 littleVGL 自带的字体中给我们提供了一些常用的图标,但是总有时候,我们需要的

图标在 littleVGL 提供的图标中是找不到的,那么这时候我们该怎么办呢?别慌,我们自己动手撸它,只要功夫深,铁棒也得被磨成针呀,现在我们来讲解如何创建一个自己的图标字库,其实这个最大的难点就是如何获取到一个 ttf/woff/woff2 格式的字体文件,只要解决了这个,那么图标字体的创建和上面普通字体的创建就没啥大区别了。

现在我给大家介绍一个免费的图标字体平台,叫”阿里巴巴矢量图标库”,简称 iconfont,没错他是我们国内开发的一个平台,官方网址为:<https://www.iconfont.cn/>,非常的好用,而且操作很简单,我们只需要在上面挑选出我们需要的图标,然后把其添加到我们创建的图标项目中,接着只要点击”下载至本地”按钮,就可以把相应的.ttf 图标字体文件下载到我们的电脑上了,下面我们就开始讲一下这个阿里巴巴矢量图标库平台怎么使用。

在浏览器输入官网网址,打开网页后,如果是第一次使用的话,那么请先注册一下账号,如下图所示:

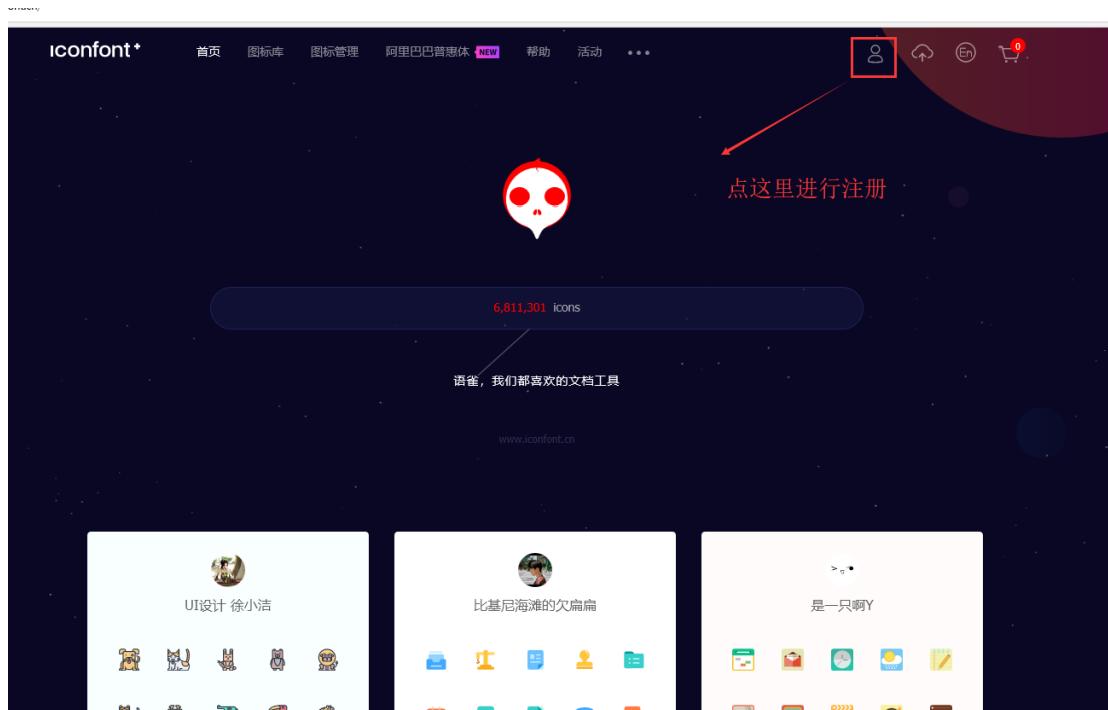


图 3.2.1 注册账号

注册完账号后,我们需要来创建一个图标项目了,先点击”图标管理”->”我的项目”进入项目管理面板,如下图所示:

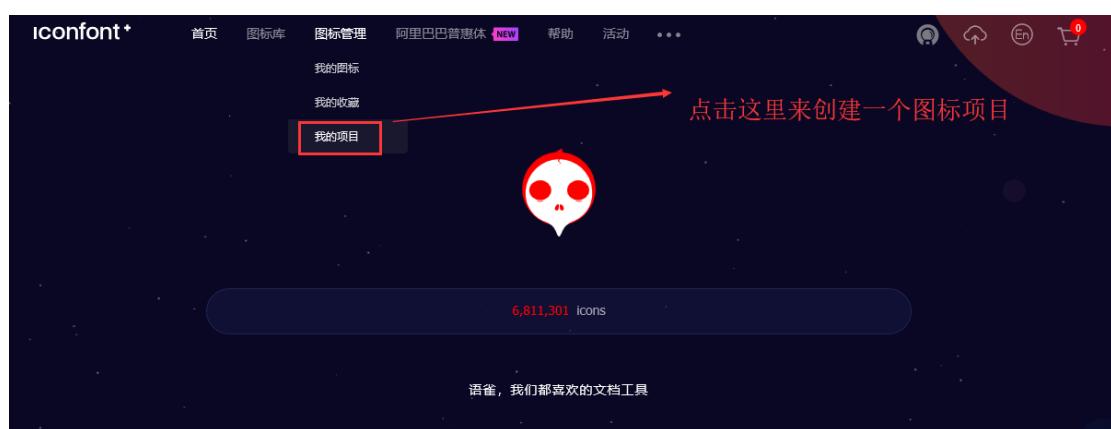


图 3.2.2 点击进入项目管理面板



图 3.2.3 点击进行创建项目

点击之后,会弹出一个创建对话框,里面只需要输入完项目名称就可以了,其他的保持默认就行,如下图所示:

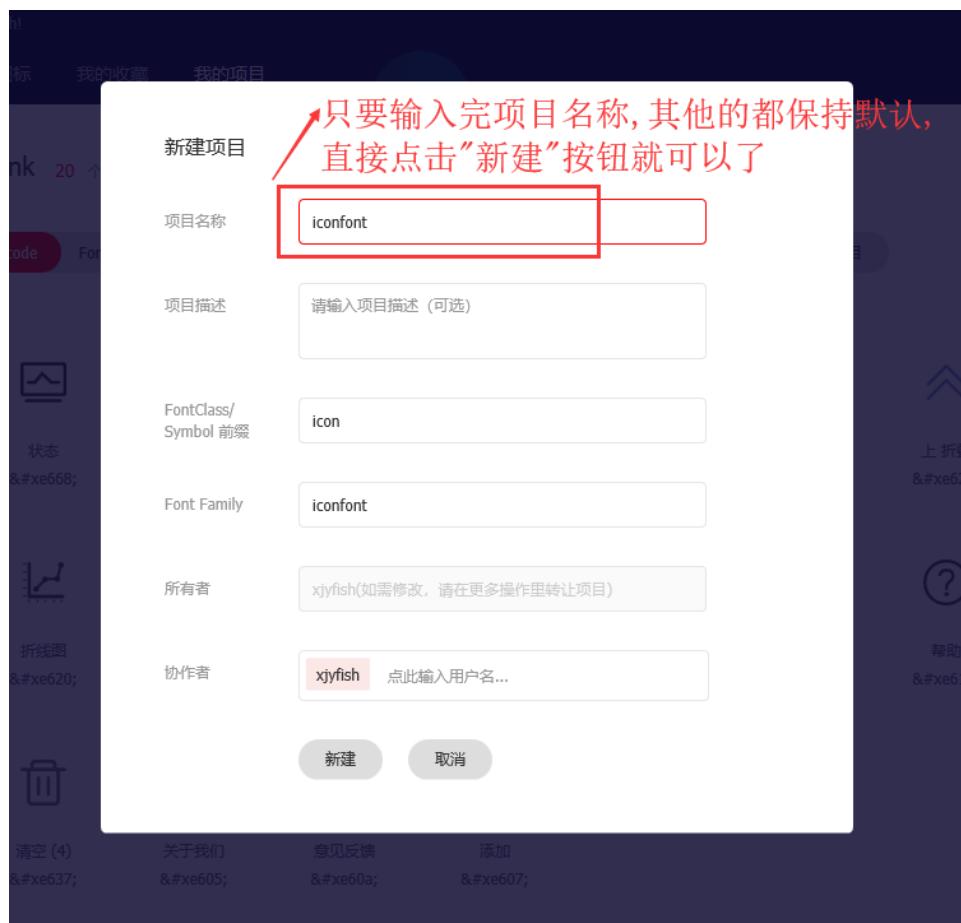


图 3.2.4 创建图标项目

创建完图标项目之后,现在我们要去挑选我们所需要的图标了,这里我以 wifi 和 usb 俩个图标为例进行演示,在搜索框中直接输入我们需要的图标名就可以了,如下图所示:

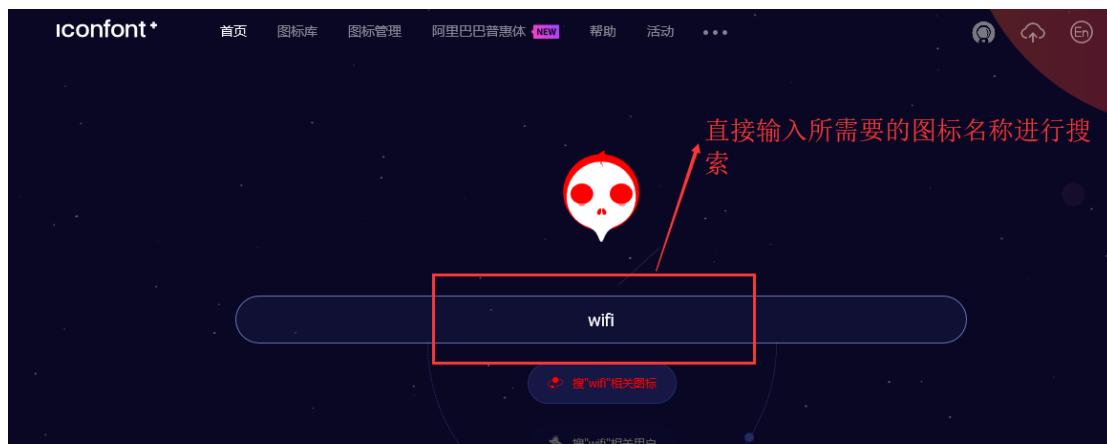


图 3.2.5 搜索图标

搜索之后,会出现一大堆的图标,我们需要把符合我们条件的图标先添加入库,如下图所示:

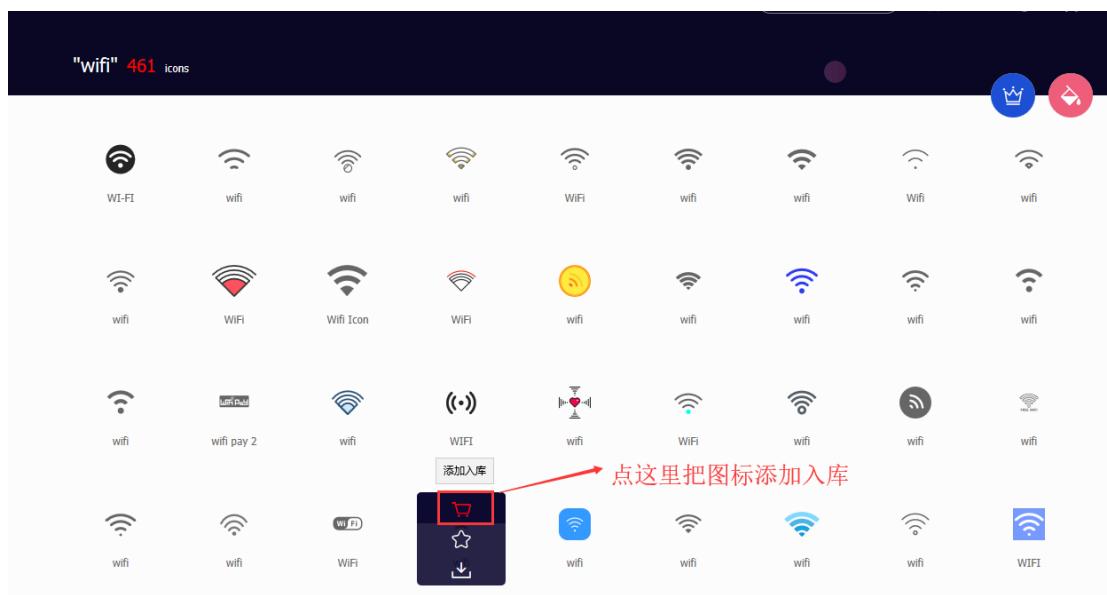


图 3.2.6 把符合条件的图标添加入库

wifi 图标添加入库之后,usb 图标也是同样的方式添加入库,接着我们需要把入库了的图标保存到我们刚刚创建了的图标项目中,这要先点击最右上角的  购物车图标,如下图所示:



图 3.2.7 点击购物车图标

点击之后,会弹出库面板,然后点击“添加至项目”按钮,如下图所示:



图 3.2.8 把图标添加到刚创建的项目中

接下来的最后一步就是把此图标项目下载到我们的本地磁盘上,如下图所示:



图 3.2.9 下载图标项目

下载完后,我们会得到一个 download.zip 的压缩包,我们只需要其里面的 iconfont.ttf 这一个字体文件,我们把 iconfont.ttf 这个文件拷贝到桌面上,接下来我们需要构建一条创建图标字库的命令,命令如下所示:

```
lv_font_conv --no-compress --format lvgl --font C:\Users\fish\Desktop\iconfont.ttf -o C:\Users\fish\Desktop\my_font_icon_30.c --bpp 4 --size 30 -r 0xe7e0,0xed6a
```

其中 0xe7e0 是 wifi 图标的 unicode 编码,0xed6a 是 usb 图标的 unicode 编码,这个是可以在我们刚创建的图标项目中查看到的,如下图所示:

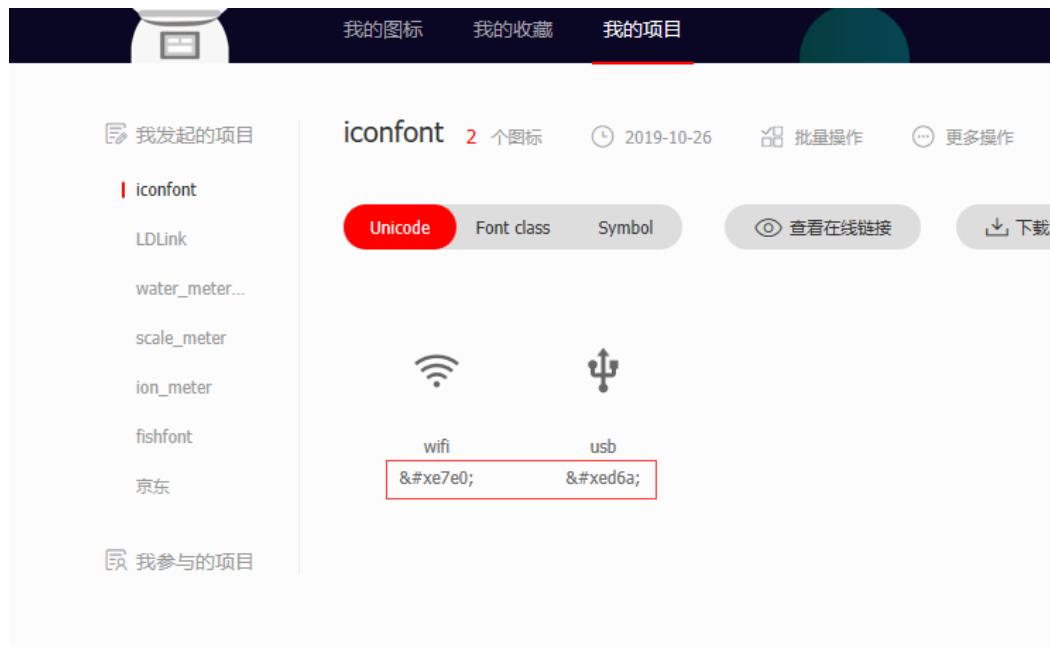


图 3.2.10 查看图标对应的 unicode 编码

最后我们复制上面的命令到 cmd 命令窗口中进行运行,然后把生成的 my_font_icon_30.c 文件拷贝到 GUI_APP 目录下,接着把此文件导入到 Keil 中,写一段简单的测试代码如下:

```

LV_FONT_DECLARE(my_font_icon_30);//申明字体

//定义图标,因为 littleVGL 只接受 utf-8 编码
//所以我们得通过工具把图标的 unicode 编码转化成 utf-8 编码
#define MY_ICON_WIFI "\xEE\x9F\xA0"
#define MY_ICON_USB      "\xEE\xB5\xAA"

//测试例程入口
void lv_font_test_start()
{
    lv_obj_t* scr = lv_scr_act();

    lv_obj_t* label1 = lv_label_create(scr,NULL);
    static lv_style_t style1;
    lv_style_copy(&style1,&lv_style_plain_color);
    style1.text.font = &my_font_icon_30;//使用图标字体
    lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&style1);//设置样式
    lv_label_set_text(label1,MY_ICON_WIFI MY_ICON_USB);//设置文本
    lv_label_set_body_draw(label1,true);//绘制背景
    lv_obj_align(label1,NULL,LV_ALIGN_CENTER,0,0);//标签与屏幕居中对齐
}

```



图 3.2.11 运行效果

上面我们说到需要把图标的 `unicode` 编码转化为 `utf-8` 编码, 这里需要用到转换工具, 总共有 2 个工具, 一个在线的, 一个离线的, 在线转换工具的网址为:

<http://www.ltg.ed.ac.uk/~richard/utf-8.cgi> , 打开之后界面效果如下:

Character	
Character name	
Hex code point	E7E0
Decimal code point	59360
Hex UTF-8 bytes	EE 9F A0
Octal UTF-8 bytes	356 237 240
UTF-8 bytes as Latin-1 characters bytes	i <9F> <A0>

图 3.2.12 unicode 转 utf-8 在线工具

然后离线版本的工具是笔者为了方便大家,简单写的一个 c 控制台应用,简陋之处,请大家多多包涵,如后面有机会,我会带领大家用一种全新的 PC 软件开发方式来开发一款 PC 软件,不是用传统的 MFC, QT, VB, C#等方式,而是用 electron 或 nw.js 前端技术,好处就是做出来的界面很容易美化,开发效率也高,而且一套代码跨 Windows, Linux, Mac os 等系统,废话不多说了,此离线版本的使用界面如下:

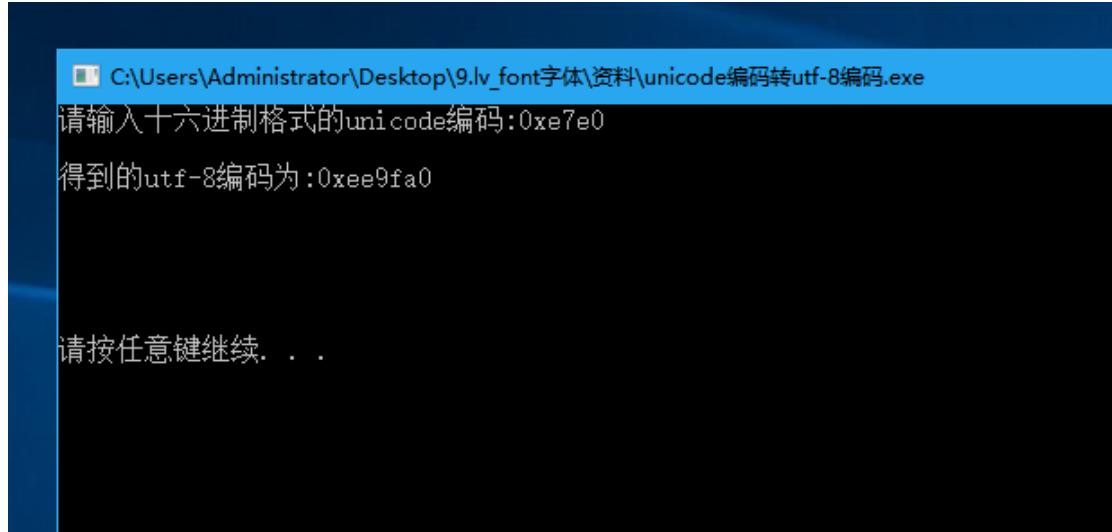


图 3.2.13 unicode 转 utf-8 离线工具

3.4 创建合并字体

有时候,我们可能会有这样的一个需求,那就是需要在一个 `label` 控件上同时显示普通字符和图标,如下面代码所示:

```
lv_label_set_text(label,MY_ICON_WIFI":fish 已连接");
```

一般情况下是做不到的,因为图标和普通字符明显是两个不同的.ttf 字体文件,但是在 littleVGL 中,它做到了,他可以让多个不同的.ttf(.wof 和.woff2 也是可以的)字体文件中的字符合并到一个字库上去,现在我就用 `heiti.ttf` 普通字体文件和 `iconfont.ttf` 图标字体文件以合并的方式来生成一个 `my_font_30.c` 的字库,操作过程跟前面的 3.2 和 3.3 章节差不多是一样的,下面我直接给出创建命令:

```
lv_font_conv --no-compress --format lvgl --font C:\Users\fish\Desktop\heiti.ttf -r 0x20-0x7F  
--symbols 已连接 --font C:\Users\fish\Desktop\iconfont.ttf -r 0xe7e0  
-o C:\Users\fish\Desktop\my_font_30.c --bpp 4 --size 30
```

复制上面的命令到 cmd 命令窗口进行运行,然后把创建出来的 `my_font_30.c` 文件拷贝到 `GUI_APP` 目录下,并添加到 Keil 中,最后写一段测试代码如下:

```
LV_FONT_DECLARE(my_font_icon_30);//申明字体
//定义 wifi 图标
#define MY_ICON_WIFI "\xEE\x9F\xA0"

//测试例程入口
void lv_font_test_start()
{
    lv_obj_t* scr = lv_scr_act();

    lv_obj_t* label1 = lv_label_create(scr,NULL);
    static lv_style_t style1;
    lv_style_copy(&style1,&lv_style_plain_color);
    style1.text.font = &my_font_30;//使用图标字体
    lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&style1);//设置样式
    lv_label_set_text(label1,MY_ICON_WIFI":fish 已连接");//设置文本
    lv_label_set_body_draw(label1,true);//绘制背景
    lv_obj_align(label1,NULL,LV_ALIGN_CENTER,0,0);//标签与屏幕居中对齐
}
```



图 3.4.1 合并字库的运行效果

注: 上面的 `lv_label_set_text(label,MY_ICON_WIFI":fish 已连接")`; 这句设置文本的代码在 Keil 中可能会报如下 2 条错误:

```
error: #8: missing closing quote
error: #18: expected a ")"
```

如果没有报此错误的朋友, 可以直接忽略跳过此注意事项, 因为笔者的电脑上出现了这个情况, 这个应该是由于 Keil 版本差异问题或者 Keil 自身存在这个小 bug 导致的, 什么时候有可能触发这种问题呢? 那就是当你的 Keil 设置为 UTF-8 编码, 而且字符串又是以汉字结尾时, 就有可能会触发这种问题, 解决思路就是让字符串不以汉字结尾, 那么我们可以变相地在字符串末尾添加空格, 或者其他不可见的字符, 如下所示:

```
lv_label_set_text(label,MY_ICON_WIFI":fish 已连接 ");//添加空格方式
lv_label_set_text(label,MY_ICON_WIFI":fish 已连接\x00");//以转义字符的形式
//添加其他不可见字符
```

4.例程设计

4.1 功能简介

我们用 digit.ttf, heiti.ttf, iconfont.ttf 三个字体文件来合成一个名为 my_font_30.c 的字库, 其中 digit.ttf 是数码管式的英文字体, 我们只取其中的 0x20-0x7F 字符范围, heiti.ttf 是黑体的中文字体, 我们只取其中的正点原子四个汉字, iconfont.ttf 是图标字体, 我们只取其中的 wifi 和 usb 图标

4.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

4.3 软件设计

先给出创建 my_font_30.c 字库的命令, 如下所示:

```
lv_font_conv --no-compress --format lvgl --font C:\Users\Administrator\Desktop\digit.ttf  
-r 0x20-0x7F --font C:\Users\Administrator\Desktop\heiti.ttf --symbols 正点原子 --font C:\  
Users\Administrator\Desktop\iconfont.ttf -r 0xe7e0,0xed6a -o C:\Users\Administrator\Desktop\  
my_font_30.c --bpp 4 --size 30
```

生成 my_font_30.c 文件后, 把此文件拷贝到 GUI_APP 目录下, 然后接着在 GUI_APP 目录下创建 lv_font_test.c 和 lv_font_test.h 俩个文件, 其中 lv_font_test.c 文件的内容如下:

```
#include "lv_font_test.h"  
#include "lvgl.h"  
  
LV_FONT_DECLARE(my_font_30); //申明字体  
  
//定义图标  
#define MY_ICON_WIFI "\xEE\x9F\xA0" //wifi 图标  
#define MY_ICON_USB "\xEE\xB5\xAA" //usb 图标  
  
//例程入口函数  
void lv_font_test_start()  
{  
    lv_obj_t* src = lv_scr_act(); //获取当前活跃的屏幕对象
```

```
static lv_style_t my_style;
lv_style_copy(&my_style,&lv_style_plain_color); //样式拷贝
my_style.text.font = &my_font_30; //在样式中使用字体

lv_obj_t* label = lv_label_create(src,NULL); //创建标签控件
lv_label_set_style(label,LV_LABEL_STYLE_MAIN,&my_style); //设置样式
//设置文本
lv_label_set_text(label,"正点原子 Nice\n"MY_ICON_WIFI MY_ICON_USB"\n0123456789");
lv_label_set_body_draw(label,true); //使能绘制背景
lv_obj_align(label,NULL,LV_ALIGN_CENTER,0,0); //标签与屏幕保持居中对齐
}
```

4.4 下载验证

把代码下载进去之后,正常的话,会看到如下所示的界面效果:



图 3.4.1 字体例程界面效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_cont 容器

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

lv_cont 容器

1. 介绍

所谓的容器就是一个载体,用来装东西的,在 littleVGL 中,可以用来存放各种各样的子对象,当子对象的数量越来越多时,子对象们在父容器中的排列方式就显得尤为重要,因此 lv_cont 容器就有一个专门的 Layout 布局属性来约束子对象们的摆放,layout 布局间隙是由样式来控制的,具体表现在 style.body.padding 样式属性上,lv_cont 容器除了 layout 这个重要特性外,还有一个 Auto fit 大小自动适应的特性.只要弄懂了 Layout 和 Auto fit 这两个概念,lv_cont 容器的使用就特别简单了,所以请务必搞懂,因为后面还有许多复杂一点的控件是由 lv_cont 容器构成的.

2. lv_cont 的 API 接口

2.1 主要数据类型

2.1.1 布局数据类型

```
enum {
    LV_LAYOUT_OFF = 0,
    LV_LAYOUT_CENTER,
    LV_LAYOUT_COL_L,
    LV_LAYOUT_COL_M,
    LV_LAYOUT_COL_R,
    LV_LAYOUT_ROW_T,
    LV_LAYOUT_ROW_M,
    LV_LAYOUT_ROW_B,
    LV_LAYOUT_PRETTY,
    LV_LAYOUT_GRID,
    _LV_LAYOUT_NUM //这个值是无意义的,只是用来记录一下有多少种布局方式
};

typedef uint8_t lv_layout_t;
```

这个数据类型就是 lv_cont 容器的 Layout 布局特性,是重点内容,下面我们通过图示来一一介绍每种布局方式.

下面所有的图示中,假定 A,B,C 三个对象都属于 lv_cont 容器的子对象.

1) LV_LAYOUT_OFF

这个值是用来禁止布局功能的,当禁止之后,对于子对象在容器中的摆放位置,子对象必须得手动通过 `lv_obj_set_pos(child,x,y);`这样的接口来设置其具体的位置,如果没有禁止布局功能的话,就算子对象手动调用 `lv_obj_set_pos(child,x,y);`接口,也是无效的.

2) LV_LAYOUT_CENTER

把所有的子对象以从上到下的方式摆放到父容器的正中心,如下图所示:

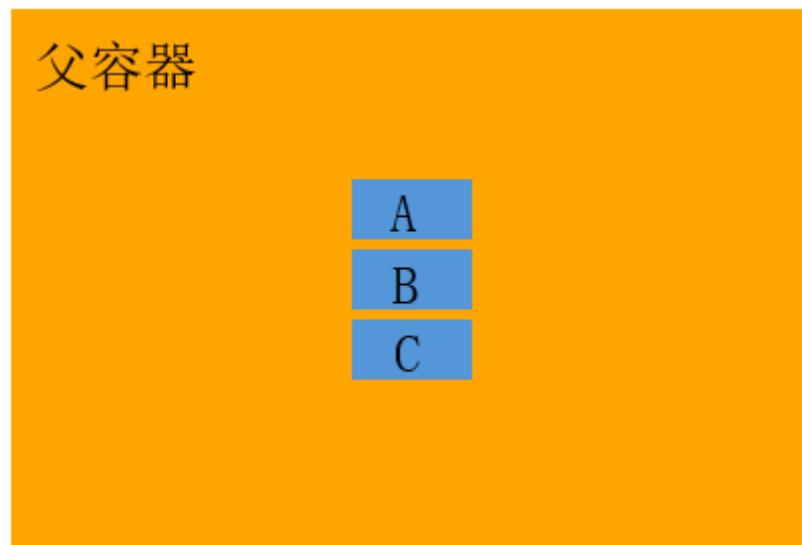


图 2.1.1.1 LV_LAYOUT_CENTER 布局效果

其中 A,B,C 子对象之间的垂直间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改

3) LV_LAYOUT_COL_L

从父容器的左上角开始,把所有的子对象以从上到下的方式进行摆放,如下图所示:



图 2.1.1.2 LV_LAYOUT_COL_L 布局效果

其中 A,B,C 子对象之间的垂直间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改,与父容器左边的距离可以通过 `lv_cont_style.body.padding.left` 样式值来修改,与父容器上边的距离可以通过 `lv_cont_style.body.padding.top` 样式值来修改

4) LV_LAYOUT_COL_M

从父容器的顶部中心点开始,把所有的子对象以从上到下的方式进行摆放,如下图所示:

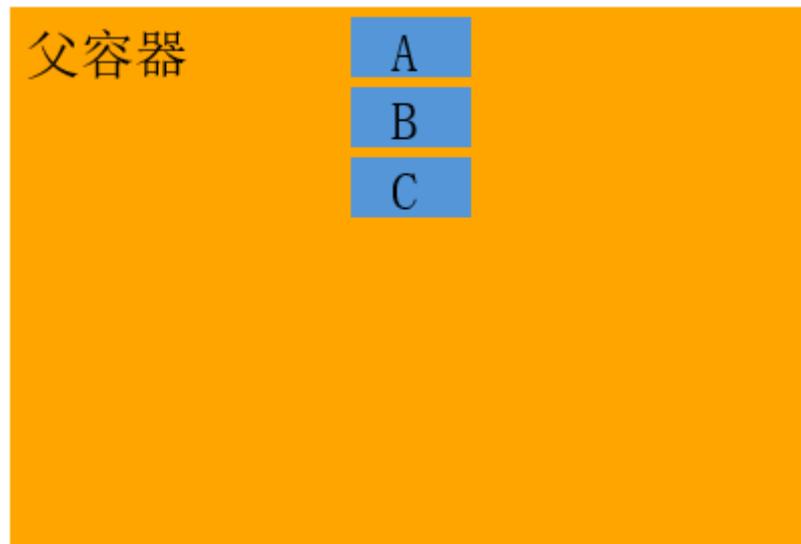


图 2.1.1.3 LV_LAYOUT_COL_M 布局效果

其中 A,B,C 子对象之间的垂直间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改,与父容器上边的距离可以通过 `lv_cont_style.body.padding.top` 样式值来修改

5) LV_LAYOUT_COL_R

从父容器的右上角开始,把所有的子对象以从上到下的方式进行摆放,如下图所示:



图 2.1.1.4 LV_LAYOUT_COL_R 布局效果

其中 A,B,C 子对象之间的垂直间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改,与父容器上边的距离可以通过 `lv_cont_style.body.padding.top` 样式值来修改

6) LV_LAYOUT_ROW_T

从父容器的左上角开始,把所有的子对象以从左到右的方式进行摆放,如下图所示:

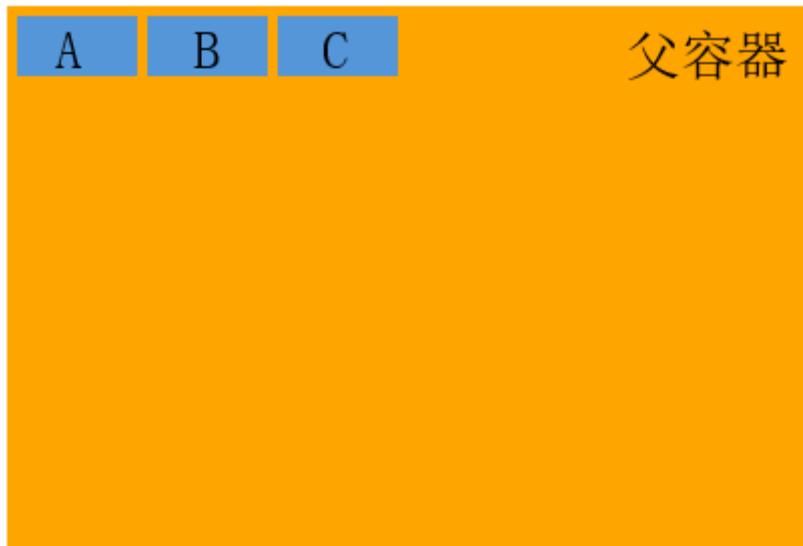


图 2.1.1.5 LV_LAYOUT_ROW_T 布局效果

其中 A,B,C 子对象之间的水平间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改,与父容器左边的距离可以通过 `lv_cont_style.body.padding.left` 样式值来修改,与父容器上边的距离可以通过 `lv_cont_style.body.padding.top` 样式值来修改

7) LV_LAYOUT_ROW_M

从父容器的左边中心点开始,把所有的子对象以从左到右的方式进行摆放,如下图所示:

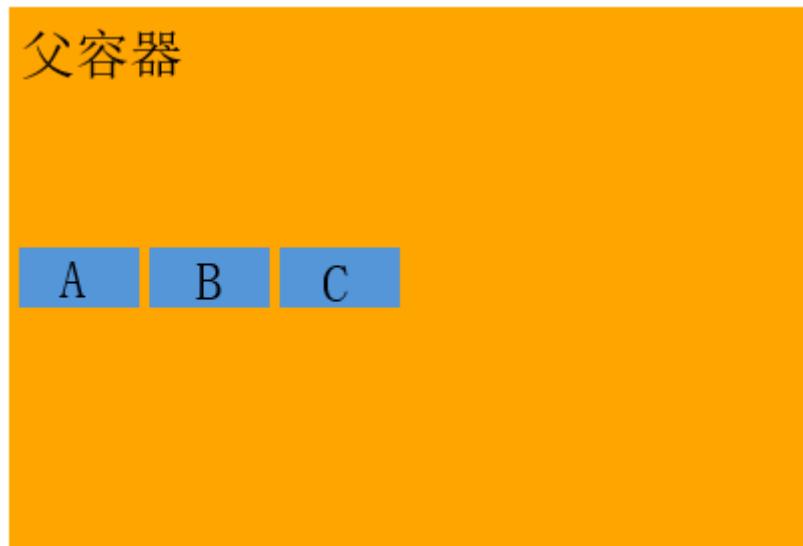


图 2.1.1.6 LV_LAYOUT_ROW_M 布局效果

其中 A,B,C 子对象之间的水平间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改,与父容器左边的距离可以通过 `lv_cont_style.body.padding.left` 样式值来修改

8) LV_LAYOUT_ROW_B

从父容器的左下角开始,把所有的子对象以从左到右的方式进行摆放,如下图所示:

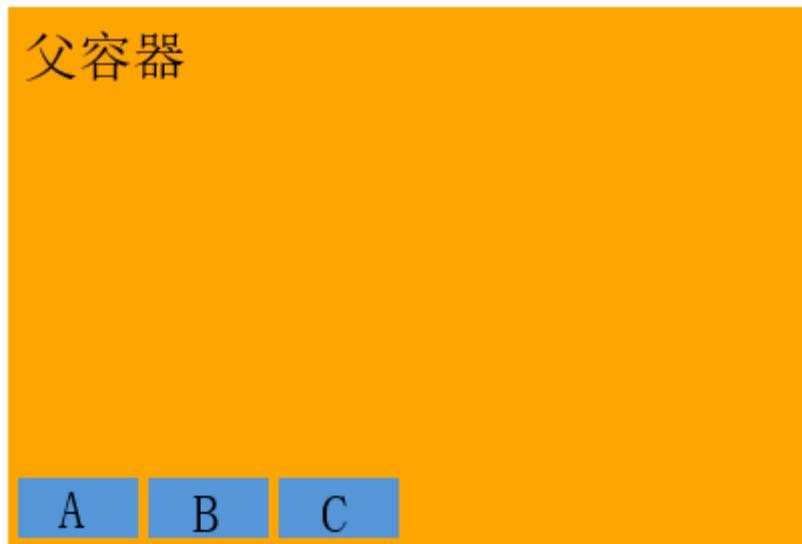


图 2.1.1.7 LV_LAYOUT_ROW_B 布局效果

其中 A,B,C 子对象之间的水平间距可以通过 `lv_cont_style.body.padding.inner` 样式值来修改,与父容器下边的距离可以通过 `lv_cont_style.body.padding.bottom` 样式值来修改

9) LV_LAYOUT_PRETTY

从父容器的左上角开始,把所有的子对象以从左到右,从上到下的方式进行摆放,并且保证每一行都均匀分布,如下图所示:

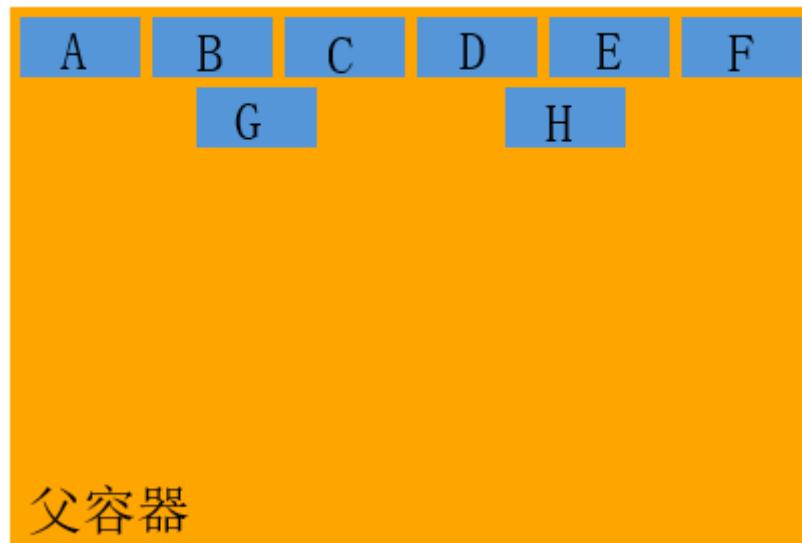


图 2.1.1.8 LV_LAYOUT_PRETTY 布局效果

其中子对象之间的水平和垂直距离都是通过 `lv_cont_style.body.padding.inner` 样式值来

修改的,另外通过 `lv_cont_style.body.padding.left/top/right/bottom` 等样式值来修改其与父容器相应边的距离

10) LV_LAYOUT_GRID

`LV_LAYOUT_GRID` 和 `LV_LAYOUT_PRETTY` 是很相似的,但是 `LV_LAYOUT_GRID` 不要求在每一行均匀分布,而是按照 `lv_cont_style.body.padding.inner` 指定的间隙依次摆放

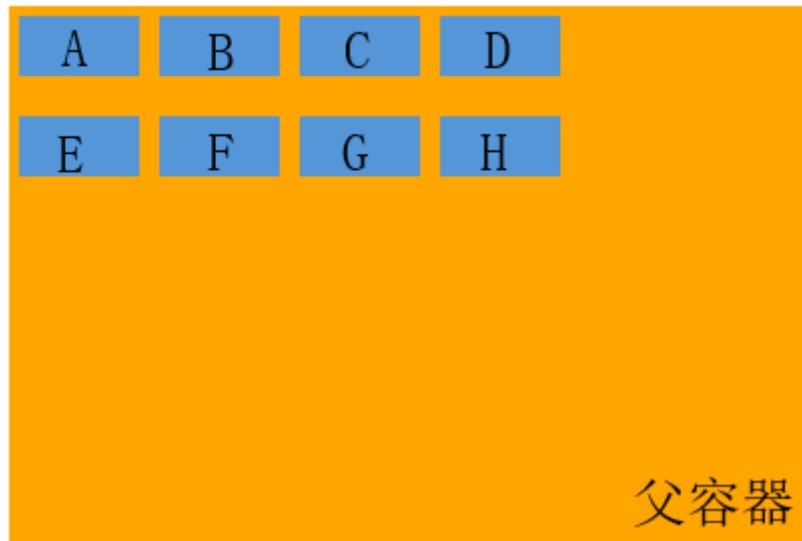


图 2.1.1.9 LV_LAYOUT_GRID 布局效果

2.1.2 大小自适应数据类型

```
enum {
    LV_FIT_NONE,
    LV_FIT_TIGHT,
    LV_FIT_FLOOD,
    LV_FIT_FILL,
    _LV_FIT_NUM //这个值是无意义的,只是用来记录一下有多少种自适应方式
};

typedef uint8_t lv_fit_t;
```

这个数据类型就是 `lv_cont` 容器的大小自适应特性,所谓的自适应就是指 `lv_cont` 容器的大小可以根据它的子对象们或父对象的大小来自动调整,下面详细说一下每一个值的具体作用

1) LV_FIT_NONE

禁止 `lv_cont` 容器使用自适应功能,当禁止之后,那么 `lv_cont` 容器只能通过手动调用 `lv_obj_set_size` 接口来设置容器的高和宽了,如果没有禁止的话,调用 `lv_obj_set_size` 接口是无效的

2) LV_FIT_TIGHT

包裹住所有的子对象,也就是说 lv_cont 容器的大小是随着子对象们的总大小变化而变化的,子对象与父容器四边的距离可以通过 lv_cont_style.body.padding.left/top/right/bottom 样式值来相应的设置

3) LV_FIT_FLOOD

lv_cont 容器平铺它父对象的整个空间,lv_cont 容器与其父对象四边的距离可以通过其父对象样式中的 body.padding.left/top/right/bottom 值来相应的修改

4) LV_FIT_FILL

当 lv_cont 容器比其父对象小时,采用 LV_FIT_FLOOD 方式,当比其父对象大时,采用 LV_FIT_TIGHT 方式

2.2 API 接口

2.2.1 创建容器

```
lv_obj_t * lv_cont_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 指向父对象

copy: 此参数可选,表示创建新对象时,把 copy 对象上的属性值复制过来

返回值:

返回新创建出来的容器对象,如果为 NULL 的话,说明堆空间不足了

2.2.2 设置容器的布局方式

```
void lv_cont_set_layout(lv_obj_t * cont, lv_layout_t layout);
```

参数:

cont: 容器对象

layout: 布局方式,请详看 2.1.1 布局数据类型那一小章节

2.2.3 设置容器 4 个方向上的自适应方式

```
void lv_cont_set_fit4(lv_obj_t * cont, lv_fit_t left, lv_fit_t right, lv_fit_t top, lv_fit_t bottom);
```

参数:

cont: 容器对象

left: 左边的自适应方式,请详看 2.1.2 大小自适应数据类型那一小章节

right: 右边的自适应方式

top: 上边的自适应方式

bottom: 下边的自适应方式

比如说,我设置容器 A 的左边为 LV_FIT_NONE 方式,设置容器 A 的右边为 LV_FIT_TIGHT 方式,假如现在子对象在宽度上增大了,那么容器 A 的宽度也会跟着变大,但是容器 A 只会朝着右边这个方向来把宽度增大的,因为容器 A 的左边是 LV_FIT_NONE 方式,即左边是无自适应的,通过这个例子,我相信大家能够举一反三

2.2.4 设置容器水平和垂直方向上的自适应方式

```
static inline void lv_cont_set_fit2(lv_obj_t * cont, lv_fit_t hor, lv_fit_t ver);
```

参数:

cont: 容器对象

hor: 水平方向上的自适应方式

ver: 垂直方向上的自适应方式

这个接口的内部实现原理很简单,没啥好解释的,直接看下图吧:

```
static inline void lv_cont_set_fit2(lv_obj_t * cont, lv_fit_t hor, lv_fit_t ver)
{
    lv_cont_set_fit4(cont, hor, hor, ver, ver);
}
```

图 2.2.4.1 内部实现原理

2.2.5 设置容器的自适应方式

```
static inline void lv_cont_set_fit(lv_obj_t * cont, lv_fit_t fit);
```

参数:

cont: 容器对象

fit: 四个方向上的共同自适应方式

这个接口的内部实现原理也很简单,没啥好解释的,直接看下图吧:

```
static inline void lv_cont_set_fit(lv_obj_t * cont, lv_fit_t fit)
{
    lv_cont_set_fit4(cont, fit, fit, fit, fit);
}
```

图 2.2.5.2 内部实现原理

2.2.6 设置容器的样式

```
static inline void lv_cont_set_style(lv_obj_t * cont, lv_cont_style_t type, const lv_style_t *
style);
```

参数:

cont: 容器对象

type: 设置容器上的哪部分样式, 目前只有 LV_CONT_STYLE_MAIN 一个可选值

style: 样式

3.例程设计

3.1 功能简介

创建一个 cont 容器和一个 tip 标签,tip 提示标签主要是用来显示 cont 容器当前的自适应方式和布局方式的,在创建容器时,默认先给容器一个 LV_LAYOUT_CENTER 的布局方式和一个 LV_FIT_TIGHT 的自适应方式,然后当用户每按一次 KEY0 按键时,就会往 cont 容器中添加一个序号依次增加的标签子对象,当用户每按一次 KEY1 按键时,就会把 cont 容器最后添加的一个子对象给删除掉,此时通过 KEY0 和 KEY1 键的相互操作,你可以发现容器的高度会相应的增加或者减少,这就演示了容器大小的自适应能力,然后当用户每按一次 KEY2 键时,会切换一次 cont 容器的布局方式,此时再接着利用 KEY0 和 KEY1 键的配合,你可以看到 cont 容器的每一种布局效果

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1,KEY2 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_cont_test.c 和 lv_cont_test.h 两个文件,其中 lv_cont_test.c 文件的内容如下:

```
#include "lv_cont_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_obj_t* cont;
lv_obj_t* tip;
lv_layout_t layout = LV_LAYOUT_CENTER;//起始的布局方式
lv_fit_t fit = LV_FIT_TIGHT;//起始的自适应方式
u8 child_no = 1;
//布局提示内容
const char* const LAYOUT_STR[] = {
    "LV_LAYOUT_OFF","LV_LAYOUT_CENTER","LV_LAYOUT_COL_L","LV_LAYOUT_COL_M","LV_LAYOUT_COL_R",
    "LV_LAYOUT_ROW_T","LV_LAYOUT_ROW_M","LV_LAYOUT_ROW_B","LV_LAYOUT_PERTY","LV_LAYOUT_GRID"
```

```
};

//自适应提示内容
const char* const FIT_STR[] = {"LV_FIT_NONE","LV_FIT_TIGHT","LV_FIT_FLOOD","LV_FIT_FILL"};

//将当前的布局方式和自适应方式通过 label 显示出来
void info_tip()
{
    char buff[80];
    sprintf(buff,"#ff0000 Fit:#%s\n#ff0000 Layout:#%s",FIT_STR[fit],LAYOUT_STR[layout]);
    lv_label_set_text(tip,buff);
    lv_obj_align(tip,NULL,LV_ALIGN_IN_BOTTOM_MID,0,-40);
}

//例程入口函数
void lv_cont_test_start()
{
    lv_obj_t* scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1. 创建容器并初始化
    cont = lv_cont_create(scr,NULL); //创建容器
    lv_obj_set_pos(cont,20,20); //设置坐标
    //设置 cont 容器四个方向上的自适应方式,我们让容器的 left 左边和 top 上边无自适应,即保持不动,因为容器的坐标为(20,20),数值比较小,如果左边和上边不设置为无自适应的话,当子对象数量增多时,容器因高或宽自动变大,就很容易碰到屏幕的左边缘或上边缘
    lv_cont_set_fit4(cont,LV_FIT_NONE,fit,LV_FIT_NONE,fit);
    //先设置容器布局方式
    lv_cont_set_layout(cont,layout);

    static lv_style_t cont_style;
    lv_style_copy(&cont_style,&lv_style_plain_color); //样式拷贝
    //设置纯红色的背景
    cont_style.body.main_color = LV_COLOR_RED;
    cont_style.body.grad_color = LV_COLOR_RED;
    //设置容器的 4 个内边距
    cont_style.body.padding.top = 10;
    cont_style.body.padding.left = 10;
    cont_style.body.padding.right = 10;
    cont_style.body.padding.bottom = 10;
    cont_style.body.padding.inner = 10; //设置容器中子对象之间的间隙

    lv_cont_set_style(cont,LV_CONT_STYLE_MAIN,&cont_style); //给容器设置样式
```

```
//2.创建一个提示用的 label
tip = lv_label_create(scr,NULL);
lv_label_set_recolor(tip,true);//使能颜色重绘
info_tip();
}

//在容器中添加子对象
void cont_add_child()
{
    char no[4];//记录子对象的编号
    lv_obj_t* child;

    child = lv_label_create(cont,NULL);
    lv_label_set_long_mode(child,LV_LABEL_LONG_CROP);//设置长文本模式
    lv_label_set_body_draw(child,true);//使能背景绘制
    lv_obj_set_size(child,50,50);//设置固定的大小
    lv_label_set_style(child,LV_LABEL_STYLE_MAIN,&lv_style_plain_color);//设置样式
    lv_label_set_align(child,LV_LABEL_ALIGN_CENTER);//设置文本居中对齐
    sprintf(no,"%d",child_no++);
    lv_label_set_text(child,no);
}

//删除容器最后添加的一个子对象
void cont_del_child()
{
    lv_obj_t* child = lv_obj_get_child(cont,NULL);
    if(child)//如果为 NULL 的话,说明没有子对象了
    {
        lv_obj_del(child);//删除掉此子对象
        child_no--;
    }
}

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

    if(key==KEY0_PRES)
    {
        //在 cont 容器中添加子对象
        cont_add_child();
    }else if(key==KEY1_PRES)
```

```
{  
    //删除掉 cont 容器中最后被添加进来的子对象  
    cont_del_child();  
}  
else if(key==KEY2_PRES)  
{  
    //为了让布局切换看得更明显,我们先把容器的自适应功能给关掉,然后给容器  
    //一个固定的大小  
    if(fit!=LV_FIT_NONE)//只需要执行一次就可以了  
    {  
        fit = LV_FIT_NONE;  
        lv_cont_set_fit(cont,fit);  
        lv_obj_set_size(cont,200,200);  
    }  
    //循环切换容器的布局  
    layout++;  
    if(layout==_LV_LAYOUT_NUM)  
        layout = LV_LAYOUT_CENTER;  
    lv_cont_set_layout(cont,layout);  
    info_tip();//信息提示  
}  
}
```

3.4 下载验证

把代码下载进去之后,初始的界面效果如下:

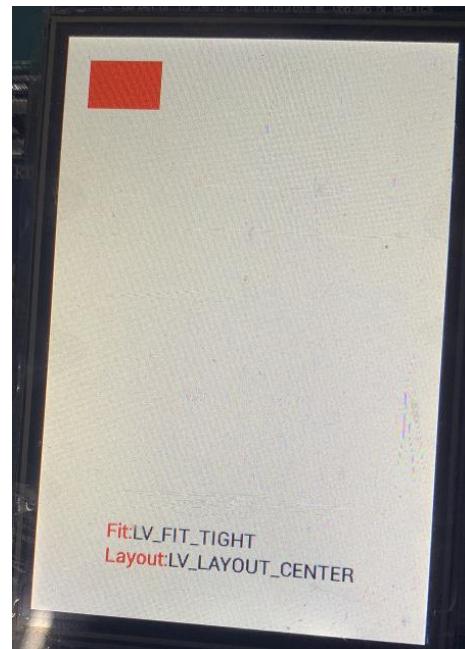


图 3.4.1 初始界面效果

然后按 KEY0 键三次,往 cont 容器里面添加了 3 个子对象之后的界面效果如下:

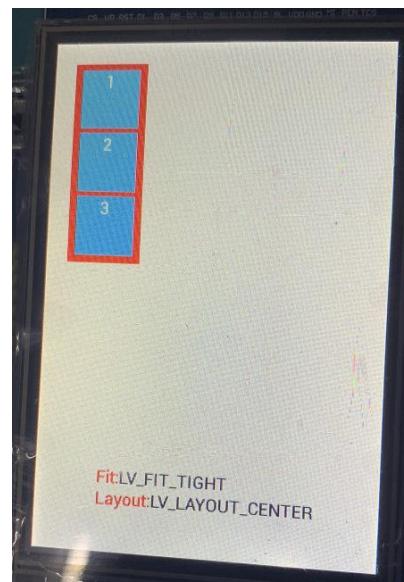


图 3.4.2 添加了 3 个子对象后的界面效果

往容器中添加子对象后,可以发现容器的高度明显增加了,然后我们再按一次 KEY1 键来删除掉 3 号子对象,界面效果如下:

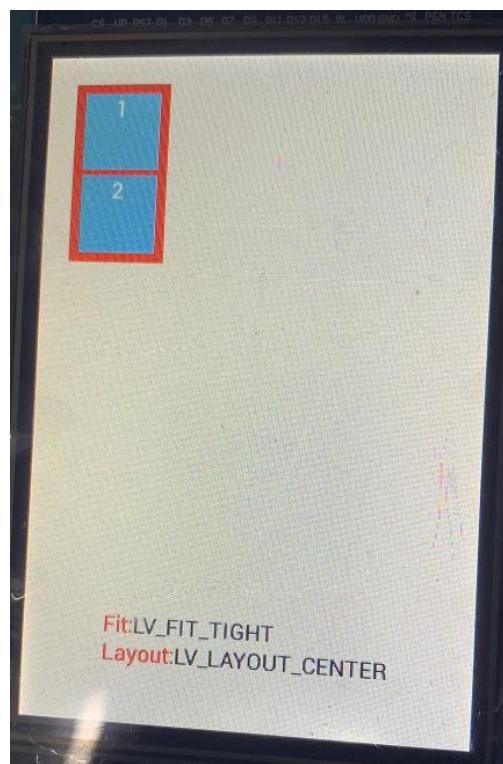


图 3.4.3 删除 3 号子对象后的界面效果

删除一个子对象后,可以发现容器的高度明显减少了,再接着我们按下 KEY2 键来切换容器的布局方式,界面效果如下:

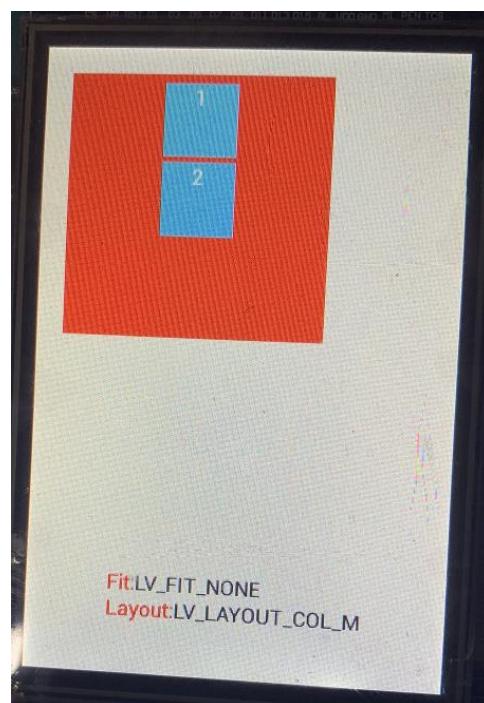


图 3.4.4 LV_LAYOUT_COL_M 布局方式

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_btn 按钮

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

lv_btn 按钮

1. 介绍

说到按钮,咱们都不陌生吧,和 lv_label 标签控件一样重要,除了能显示文本外,它的主要用途是和用户进行交互,lv_btn 按钮其实就是一个 lv_cont 容器的复杂变体,另外加上可选的子对象,最常用的子对象就是 lv_label 标签,在 littleVGL 中,lv_btn 按钮具有 5 种状态,如下所示:

LV_BTN_STATE_REL: 按钮的正常释放状态

LV_BTN_STATE_PR: 按钮的正常按下状态

LV_BTN_STATE_TGL_REL: 按钮的切换(Toggle)释放状态

LV_BTN_STATE_TGL_PR: 按钮的切换(Toggle)按下状态

LV_BTN_STATE_INA: 按钮的禁用无效状态

对于**非切换按钮**,当对其进行一次点击操作时(也就是先按下再松手的操作),它的状态转换如下所示:

LV_BTN_STATE_REL -> LV_BTN_STATE_PR -> LV_BTN_STATE_REL

对于**切换按钮**,当对其进行一次点击操作时(也就是先按下再松手的操作),它的状态转换根据其起始状态不同有 2 种情况,如下所示:

起始状态为正常态(也就是 LV_BTN_STATE_REL 释放子状态):

LV_BTN_STATE_REL -> LV_BTN_STATE_PR -> LV_BTN_STATE_TGL_REL

起始状态为切换态(也就是 LV_BTN_STATE_TGL_REL 释放子状态):

LV_BTN_STATE_TGL_REL -> LV_BTN_STATE_TGL_PR -> LV_BTN_STATE_REL

也就是说每一次点击操作,会使切换按钮在正常态和切换态之间来回循环,正常态下有 LV_BTN_STATE_REL 释放和 LV_BTN_STATE_PR 按下俩种子状态,而切换态下有 LV_BTN_STATE_TGL_REL 释放和 LV_BTN_STATE_TGL_PR 按下俩种子状态,为什么切换按钮的起始状态只可能是 2 种释放子状态呢?不可以为 2 种按下子状态吗?这个很简单咯,这是因为按子状态只有在手按下的时候才会存在,也就是说它是短暂的,不可能永久维持,不管是那一种形式的按钮,你都可以通过调用 lv_btn_set_state(btn,state);接口来手动设置按钮的状态.默认创建的按钮是非切换形式的,如果你想把此按钮变成切换形式的,那么你可以调用 lv_btn_set_toggle(btn, true)接口来使能.既然按钮有 5 种不同的状态,那么相应的,它也有 5 种与之对应的样式,如下所示:

LV_BTN_STYLE_REL: 正常态下释放时的样式,默认值为 lv_style_btn_rel

LV_BTN_STYLE_PR: 正常态下按下时的样式,默认值为 lv_style_btn_pr

LV_BTN_STYLE_TGL_REL: 切换态下释放时的样式,默认值为 lv_style_btn_tgl_rel

LV_BTN_STYLE_TGL_PR: 切换态下按下时的样式,默认值为 lv_style_btn_tgl_pr

LV_BTN_STYLE_INA: 禁用无效时的样式,默认值为 lv_style_btn_ina

你可以通过 lv_btn_set_style(btn, LV_BTN_STYLE_..., &style)接口来修改某种状态下的样式,而不再采用其默认样式值了,之前我们已经学过了 lv_cont 容器,因为 lv_btn 就是 lv_cont 的复杂变体,所以 lv_btn 同样也具有容器的 Layout 布局和 Fit 自适应特性.

2. lv_btn 的 API 接口

2.1 主要数据类型

2.1.1 按钮状态数据类型

```
enum {
    LV_BTN_STATE_REL, //按钮的正常释放状态
    LV_BTN_STATE_PR, //按钮的正常按下状态
    LV_BTN_STATE_TGL_REL, //按钮的切换(Toggle)释放状态
    LV_BTN_STATE_TGL_PR, //按钮的切换(Toggle)按下状态
    LV_BTN_STATE_INA, //禁用无效状态
    _LV_BTN_STATE_NUM, //无意义
};

typedef uint8_t lv_btn_state_t;
```

这个就是对应按钮的 5 种状态,挺简单的,不用过多介绍了

2.1.2 按钮样式数据类型

```
enum {
    LV_BTN_STYLE_REL, //正常态下释放时的样式,默认值为 lv_style_btn_rel
    LV_BTN_STYLE_PR, //正常态下按下时的样式,默认值为 lv_style_btn_pr
    LV_BTN_STYLE_TGL_REL, //切换态下释放时的样式,默认值为 lv_style_btn_tgl_rel
    LV_BTN_STYLE_TGL_PR, //切换态下按下时的样式,默认值为 lv_style_btn_tgl_pr
    LV_BTN_STYLE_INA, //禁用无效时的样式,默认值为 lv_style_btn_ina
};

typedef uint8_t lv_btn_style_t;
```

2.2 API 接口

2.2.1 创建按钮

```
lv_obj_t * lv_btn_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 指向父对象

copy: 此参数可选,表示创建新对象时,把 copy 对象上的属性值复制过来

返回值:

返回新创建出来的按钮对象,如果为 NULL 的话,说明堆空间不足了

2.2.2 设置为 Toggle 切换按钮

```
void lv_btn_set_toggle(lv_obj_t * btn, bool tgl);
```

参数:

btn: 按钮对象

tgl: 是否设置为 toggle 切换按钮

所谓的切换按钮就跟自锁开关的效果一样,当按下去之后,它是不会自动弹上来的,必须的再按一次,才会弹上来

2.2.3 手动设置按钮的状态

```
void lv_btn_set_state(lv_obj_t * btn, lv_btn_state_t state);
```

参数:

btn: 按钮对象

state: 按钮的状态

这是通过手动的方式来设置按钮的状态,用的最多的地方就是把按钮设置为 LV_BTN_STYLE_INA 禁用无效状态,使其不能点击,否则一般都不会用到这个 API 接口

2.2.4 设置按钮的布局和自适应方式

```
static inline void lv_btn_set_layout(lv_obj_t * btn, lv_layout_t layout);
```

参数:

btn: 按钮对象

layout: 布局方式

说的再本质一点,lv_btn 其实就是 lv_cont 容器,无非就是多增加了点击效果,所以 lv_cont 容器拥有的特性,lv_btn 按钮是全部都拥有的,大家脑海中一定要有这个概念,与此类似的 API 接口还有下面几个跟**自适应特性**相关的接口:

:

```
void lv_btn_set_fit4(lv_obj_t * btn, lv_fit_t left, lv_fit_t right, lv_fit_t top, lv_fit_t bottom);
void lv_btn_set_fit2(lv_obj_t * btn, lv_fit_t hor, lv_fit_t ver);
void lv_btn_set_fit(lv_obj_t * btn, lv_fit_t fit);
```

对于这 4 个 API 接口的用法是跟 lv_cont 容器中的用法是一模一样的,如还不懂,请去仔细查看 lv_cont 容器那个章节的文档

2.2.5 设置按钮的波纹点击效果

```
void lv_btn_set_ink_in_time(lv_obj_t * btn, uint16_t time);
void lv_btn_set_ink_wait_time(lv_obj_t * btn, uint16_t time);
void lv_btn_set_ink_out_time(lv_obj_t * btn, uint16_t time);
```

参数:

btn: 按钮对象

time: 时长,单位为 ms

所谓的波纹点击效果就是当点击按钮时,会从按钮的点击处开始出现一个小圆圈,然后慢慢扩大至按钮的整个背景,就跟往静止的水面上扔一个小石头后,泛起的波纹效果一样,在 littleVGL 中,我们可以把这个波纹点击效果分解成 3 个时长,如下所示:

1)波纹效果的入场动画时长

此动画时长由 `lv_btn_set_ink_in_time` 接口来设置

2)波纹效果的维持等待时长

此动画时长由 `lv_btn_set_ink_wait_time` 接口来设置

3)波纹效果的出场动画时长

此动画时长由 `lv_btn_set_ink_out_time` 接口来设置,注意出场动画是一种淡出动画

如果想要实现这种波纹点击效果,那么至少得调用 `lv_btn_set_ink_in_time` 接口来设置入场动画的时长,至于 `lv_btn_set_ink_wait_time` 和 `lv_btn_set_ink_out_time` 接口可调可不调,下面举个简单的例子:

```
lv_obj_t* btn = lv_btn_create(lv_scr_act(),NULL);//创建按钮
lv_obj_set_pos(btn,50,50);//设置坐标
lv_btn_set_ink_in_time(btn,2000);//设置入场动画的时长
lv_btn_set_ink_wait_time(btn,3000);//设置维持等待的时长
lv_btn_set_ink_out_time(btn,1000);//设置出场动画的时长
当点击按钮时,先是做入场动画,耗时 2000 毫秒,接着进入 3000 毫秒的维持等待,最后再做出场动画,耗时 1000 毫秒,下面只给出入场动画过程中的 3 张效果图
```



图 2.2.5.1 入场动画效果图 1



图 2.2.5.2 入场动画效果图 2



图 2.2.5.3 入场动画效果图 3

注:想要实现按钮的波纹点击效果,必须得先把 lv_conf.h 文件中的 LV_BTN_INK_EFFECT 宏给置 1,否则是看不到效果的

2.2.6 设置按钮的样式

```
void lv_btn_set_style(lv_obj_t * btn, lv_btn_style_t type, const lv_style_t * style);
```

参数:

btn: 按钮对象

type: 设置哪一种状态下的样式,可选值有 5 个,分别为:

```
LV_BTN_STYLE_REL //默认值为 lv_style_btn_rel  
LV_BTN_STYLE_PR //默认值为 lv_style_btn_pr  
LV_BTN_STYLE_TGL_REL //默认值为 lv_style_btn_tgl_rel  
LV_BTN_STYLE_TGL_PR //默认值为 lv_style_btn_tgl_pr  
LV_BTN_STYLE_INA //默认值为 lv_style_btn_ina
```

style: 样式

2.2.7 备注

还有一些简单的 get 类型 API 接口,我这里就不一一列举了,请自行查阅 lv_btn.h 头文件

3.例程设计

3.1 功能简介

创建 3 个按钮,第 1 个按钮是默认按钮,同时设置它具有波纹点击效果,第 2 个按钮是 Toggle 切换按钮,为其设置了自定义的样式,并注册事件回调函数,在回调函数中切换它的文本值,第 3 个是宽度自适应大小的按钮,为其设置了自定义的样式,并注册事件回调函数,在回调函数中增加文本的长度,可以明显的看到按钮的宽度也会随之增加

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_btn_test.c 和 lv_btn_test.h 俩个文件,其中 lv_btn_test.c 文件的内容如下:

```
#include "lv_btn_test.h"
#include "lvgl.h"

lv_style_t my_style_btn_release;//按钮释放状态下的样式
lv_style_t my_style_btn_press;//按钮按下时的样式
lv_obj_t * btn1,* btn2,* btn3;
lv_obj_t * btn2_label,* btn3_label;

//事件回调函数
static void btn_event_cb(lv_obj_t * obj,lv_event_t event)
{
    if(event==LV_EVENT_RELEASED)
    {
        if(obj==btn2)
        {
            lv_btn_state_t state = lv_btn_get_state(btn2);//获取切换按钮的当前状态
            //切换态或者正常态
            lv_label_set_text(btn2_label,state==LV_BTN_STATE_TGL_REL?"Toggle":"Normal");
        }else if(obj==btn3)
    }
}
```

```
//当点击时,在最后面增加文本的内容
    lv_label_ins_text(btn3_label, LV_LABEL_POS_LAST, " long ");
}

}

//例程入口函数
void lv_btn_test_start()
{
    lv_obj_t* src = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.先创建 2 种状态下按钮样式
    //1.1 释放状态下的样式
    lv_style_copy(&my_style_btn_release, &lv_style_plain_color);
    //设置纯色的背景
    my_style_btn_release.body.main_color = LV_COLOR_MAKE(0x1E, 0x9F, 0xFF);
    my_style_btn_release.body.grad_color = my_style_btn_release.body.main_color;
    my_style_btn_release.body.opa = LV_OPA_COVER; //设置背景色完全不透明
    my_style_btn_release.body.radius = LV_RADIUS_CIRCLE; //绘制圆角按钮
    my_style_btn_release.body.shadow.color = LV_COLOR_MAKE(0x1E, 0x9F, 0xFF);
    my_style_btn_release.body.shadow.type = LV_SHADOW_FULL; //设置四边都有阴影
    my_style_btn_release.body.shadow.width = 3; //设置阴影的宽度
    my_style_btn_release.text.color = LV_COLOR_WHITE;
    my_style_btn_release.body.padding.left = 10; //设置左内边距
    my_style_btn_release.body.padding.right = 10; //设置右内边距

    //1.2 按下状态下的样式
    lv_style_copy(&my_style_btn_press, &lv_style_plain_color);
    my_style_btn_press.body.opa = LV_OPA_0; //设置背景色透明
    my_style_btn_press.body.radius = LV_RADIUS_CIRCLE; //绘制圆角按钮
    //设置边框的颜色
    my_style_btn_press.body.border.color = LV_COLOR_MAKE(0xC9, 0xC9, 0xC9);
    my_style_btn_press.body.border.part = LV_BORDER_FULL; //四条边框都绘制
    my_style_btn_press.body.border.width = 2; //设置边框的宽度
    my_style_btn_press.body.border.opa = LV_OPA_COVER; //设置边框完全不透明
    my_style_btn_press.text.color = LV_COLOR_BLACK;
    my_style_btn_press.body.padding.left = 10; //设置左内边距
    my_style_btn_press.body.padding.right = 10; //设置右内边距

    //2.创建一个默认的按钮,同时使能其波纹点击效果,任何一个对象创建之后,其默认
    //的宽度为 LV_OBJ_DEF_WIDTH,默认的高度为 LV_OBJ_DEF_HEIGHT 而 LV_OB
    //J_DEF_WIDTH 和 LV_OBJ_DEF_HEIGHT 宏的值是受 lv_conf.h 文件中 LV_DPI
    //配置项的影响,具体关系如下:
    //#define LV_OBJ_DEF_WIDTH (LV_DPI)
    //#define LV_OBJ_DEF_HEIGHT (2 * LV_DPI / 3)
```

```
btn1 = lv_btn_create(src, NULL);
lv_obj_set_pos(btn1,20,20);//设置坐标
lv_btn_set_ink_in_time(btn1, 3000);//入场动画时长
lv_btn_set_ink_wait_time(btn1, 1000);//维持等待时长
//出场动画时长,这个出场动画是淡出效果的,请睁大眼睛观看,否则不容易看出效果
lv_btn_set_ink_out_time(btn1, 600);

//3.创建一个 Toggle 切换按钮
btn2 = lv_btn_create(src, NULL);
lv_obj_set_size(btn2,90,30);//设置大小
lv_obj_align(btn2, btn1, LV_ALIGN_OUT_RIGHT_TOP, 20, 0);//设置对齐方式
lv_btn_set_toggle(btn2,true);//设置为 Toggle 按钮
//设置按钮的起始状态为切换态下的释放状态
lv_btn_set_state(btn2,LV_BTN_STATE_TGL_REL);

//设置按钮切换态下的释放状态样式
lv_btn_set_style(btn2,LV_BTN_STYLE_TGL_REL,&my_style_btn_release);
//设置按钮切换态下的按下状态样式,为了看起来更美观和谐,使其和
//LV_BTN_STYLE_TGL_REL 的样式值保持一致
lv_btn_set_style(btn2,LV_BTN_STYLE_TGL_PR,&my_style_btn_release);
//设置按钮正常态下释放状态样式
lv_btn_set_style(btn2,LV_BTN_STYLE_REL,&my_style_btn_press);
//设置按钮正常态下按下状态样式,为了看起来更美观和谐,使其和
//LV_BTN_STYLE_REL 的样式值保持一致
lv_btn_set_style(btn2,LV_BTN_STYLE_PR,&my_style_btn_press);

btn2_label = lv_label_create(btn2,NULL);//给 btn2 添加 label 子对象
lv_label_set_text(btn2_label,"Toggle");

//设置按钮 2 的布局方式,使 label 处于正中间,当然了如果不设置的话,默认也是正中
//间的
lv_btn_set_layout(btn2,LV_LAYOUT_CENTER);

lv_obj_set_event_cb(btn2,btn_event_cb);//设置 btn2 的事件回调

//3.创建一个宽度自适应的正常按钮
btn3 = lv_btn_create(src, NULL);
lv_obj_align(btn3,btn1,LV_ALIGN_OUT_BOTTOM_LEFT,0,20);//设置对齐
lv_obj_set_height(btn3,30);//只设置高度固定
//设置宽度只在右边自适应
lv_btn_set_fit4(btn3,LV_FIT_NONE,LV_FIT_TIGHT,LV_FIT_NONE,LV_FIT_NONE);

//设置按钮正常态下释放状态样式
lv_btn_set_style(btn3,LV_BTN_STYLE_REL,&my_style_btn_release);
```

```
//设置按钮正常态下按下状态样式  
lv_btn_set_style(btn3, LV_BTN_STYLE_PR, &my_style_btn_press);  
  
btn3_label = lv_label_create(btn3, NULL); //给 btn3 添加 label 子对象  
lv_label_set_text(btn3_label, "This is long text");  
  
lv_obj_set_event_cb(btn3, btn_event_cb); //设置 btn3 的事件回调  
}
```

3.4 下载验证

把代码下载进去之后,正常的话,会看到如下所示的初始界面效果:

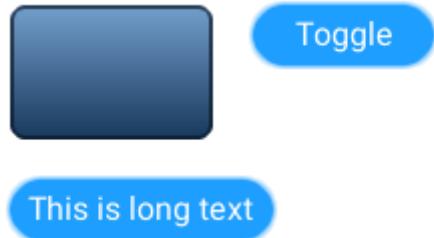


图 3.4.1 初始界面效果

然后点击波纹效果按钮,Toggle 按钮以及长文本按钮后,可以看到如下所示的界面效果:

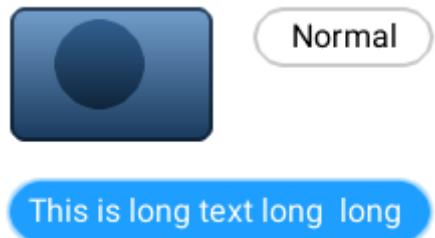


图 3.4.2 点击按钮后的效果

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

正点原子 littleVGL 开发指南

Events 事件

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

Events 事件

1. 介绍

在 littleVGL 中任何对象都可以注册事件,这是在新版本中才加入的特性,分为通用事件和专用事件,总共支持 20 种事件类型,这是一个总和哈,并不是指每一个对象都具有 20 种事件类型,事件可以是由 littleVGL 库自身触发的,也可以是由外部物理操作触发的,比如触摸,点击等等,当然了,我们也可以通过调用 `lv_event_send` 接口来手动发送事件进行触发,同时可以携带用户自定义的数据.

2. Events 的 API 接口

2.1 主要数据类型

2.1.1 事件数据类型

```
enum {
    LV_EVENT_PRESSED,//对象被按下时触发,每次按下时只触发一次
    LV_EVENT_PRESSING,//对象正在被按下中,只要按下不放,就会一直被触发
    LV_EVENT_PRESS_LOST,//在按下的过程中,手指从对象的可视区域划出时被触发

    //在 LV_INDEV_LONG_PRESS_TIME 时间之前松手触发,如果是在被拖拽的话,则不会
    //被触发
    LV_EVENT_SHORT_CLICKED,

    //按下时长超过 LV_INDEV_LONG_PRESS_TIME 值时触发, 只会触发一次,如果是在被
    //拖拽的话,则不会被触发
    LV_EVENT_LONG_PRESSED,

    //在触发 LV_EVENT_LONG_PRESSED 事件之后,接下来按下的时长每超过
    //LV_INDEV_LONG_PRESS_REPEAT 值一次,就会被触发一次,如果是在被拖拽的话,
    //则不会被触发
    LV_EVENT_LONG_PRESSED_REPEAT,

    //只要松手了就会被触发,但是如果触发了 LV_EVENT_PRESS_LOST 事件的话,那么此
    //事件会被触发
    LV_EVENT_CLICKED,
```

```
//和 LV_EVENT_CLICKED 事件一样,没啥区别,位于 LV_EVENT_CLICKED 事件之后触发
LV_EVENT_RELEASED,
LV_EVENT_DRAG_BEGIN,//拖拽开始时触发
LV_EVENT_DRAG_END,//拖拽结束时触发
LV_EVENT_DRAG_THROW_BEGIN,//拖拽漂移开始时触发

//当实体按键被按下时触发,我们一般都是用触摸屏作为输入,所以此事件
//一般用不到
LV_EVENT_KEY,
LV_EVENT_FOCUSED,//当对象在其所在的 group 组获得焦点时触发
LV_EVENT_DFOCUSSED,//当对象在其所在的 group 组失去焦点时触发
LV_EVENT_VALUE_CHANGED,//对象的数值改变时被触发,如 lv_slider 滑动控件

LV_EVENT_INSERT,//有东西插入到对象上时触发,如 lv_ta 文本框控件

//可以说是留给用户使用的一种事件,用户只能通过 lv_event_send 接口来手动发送
//触发此事件
LV_EVENT_REFRESH,

//点击 lv_kb 键盘控件上的”OK”, ”Apply” 等相似词义的按钮时触发
LV_EVENT_APPLY,

//点击 lv_kb 键盘控件上的”Close”, ”Cancel” 等相似词义的按钮时触发
LV_EVENT_CANCEL,
LV_EVENT_DELETE //对象被删除时触发
};

typedef uint8_t lv_event_t;
```

2.1.2 事件回调函数数据类型

```
typedef void (*lv_event_cb_t)(struct _lv_obj_t * obj, lv_event_t event);
```

2.2 API 接口

2.2.1 设置事件回调函数

```
void lv_obj_set_event_cb(lv_obj_t * obj, lv_event_cb_t event_cb);
```

参数:

obj: 对象句柄

event_cb: 事件回调函数

2.2.2 手动发送事件

```
lv_res_t lv_event_send(lv_obj_t * obj, lv_event_t event, const void * data);
```

参数:

obj: 给哪一个对象发送事件

event: 需要发送的事件名

data: 携带的用户自定义数据

返回值:

LV_RES_OK:对象没有被删除

LV_RES_INV:对象在事件中被删除了

这里需要注意 event 参数,系统是自带了 20 种事件类型,其对应的值是从 0 到 19,除了给 event 参数传系统自带的事件外,其实我们还可以传用户自定义的事件的,范围为:[20,255],举个简单的例子如下(只给出示意代码):

```
#define USER_EVENT_START      20
#define USER_EVENT_1           (USER_EVENT_START+1)
static void btn_event_cb(lv_obj_t * obj,lv_event_t event)
{
    if(event==USER_EVENT_1)
    {
        printf("用户自定义事件 1 被触发了\r\n");
    }
}
lv_obj_t* btn1 = lv_btn_create(src, NULL);
lv_obj_set_event_cb(btn1,btn_event_cb);//设置回调函数

//手动发送 USER_EVENT_1 事件,不携带参数
lv_event_send(btn1,USER_EVENT_1,NULL);
```

2.2.3 给任意事件回调函数手动发送事件

```
lv_res_t lv_event_send_func(lv_event_cb_t event_xcb, lv_obj_t * obj, lv_event_t event,
const void * data);
```

参数:

event_xcb: 给哪一个事件回调函数发送事件

obj: 给哪一个对象发送事件

event: 需要发送的事件名

data: 携带的用户自定义数据

返回值:

LV_RES_OK:对象没有被删除

LV_RES_INV:对象在事件中被删除了

其实这个接口也是很简单的,它只是把设置回调函数和发送事件的二步操作变成了一步,所以这个接口其实是等价 lv_obj_set_event_cb 和 lv_event_send 二个接口的调用

2.2.4 获取当前事件的用户自定义参数

```
const void * lv_event_get_data(void);
```

返回值:

返回用户自定义的数据

请注意,这个是返回当前事件的用户数据,是当前已经被触发了的事件,此 API 接口一般的用法就是放在事件回调函数中进行调用,如下所示(只给出示意代码):

```
#define USER_EVENT_START      20
#define USER_EVENT_1           (USER_EVENT_START+1)

//构建一个用户自定义数据结构体,当然了,如果你的用户数据简单,可以不用结构体
typedef struct{
    char name[20];
    u8 age;
}USER_DATA;
USER_DATA user_data = {"xiong jia yu"},25}//初始化一下

//事件回调函数
static void btn_event_cb(lv_obj_t * obj,lv_event_t event)
{
    USER_DATA *data;
    if(event==USER_EVENT_1)//当然了,这里可以是其他的任何事件,
    {
        data = (USER_DATA*) lv_event_get_data(); //获取到用户的自定义数据
        printf("name:%s,age:%d\r\n",data->name,data->age);
    }
}

void user_data_test()
{
    lv_event_send(btn1,USER_EVENT_1,&user_data); //发送事件,同时携带用户自定义参数
}
```

3.例程设计

3.1 功能简介

创建一个默认按钮,用来测试各种事件,比如短按下,长按下,松手,故意在按下时划出按钮的可视区域等等操作,当用户按下 KEY0 键时,手动来发送事件,我这里弄了二种方式,第一种是发送用户自定义事件,并且同时携带用户自定义参数,第二种是发送系统自带的事件,不携带用户自定义参数,当用户按下 KEY1 键时,会使能按钮的拖拽和拖拽惯性漂移功能,使能之后,我们可以拖动按钮来测试拖拽事件,当用户按下 KEY2 键时,会把按钮对象给删除掉,这是用来测试 LV_EVENT_DELETE 事件的

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1,KEY2 按键
- 3) 串口

3.3 软件设计

在 GUI_APP 目录下创建 events_test.c 和 events_test.h 俩个文件,其中 events_test.c 文件的内容如下:

```
#include "events_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

#define USER_EVENT_START      20
#define USER_EVENT_1           (USER_EVENT_START+1)//用户自定义事件 1

//构建一个用户自定义数据结构体,当然了,如果你的用户数据简单,可以不用结构体
typedef struct{
    char name[20];
    u8 age;
}USER_DATA;

USER_DATA user_data = {"xiong jia yu"},25}//初始化一下
```

```
lv_obj_t * btn1;

//事件回调函数
static void btn_event_cb(lv_obj_t * obj, lv_event_t event)
{
    if(event==LV_EVENT_PRESSED)
    {
        //对象被按下时触发,每次按下时只触发一次
        printf("LV_EVENT_PRESSED\r\n");
    }else if(event==LV_EVENT_PRESSING)
    {
        //对象正在被按下中,只要按下不放,就会一直被触发
        printf("LV_EVENT_PRESSING\r\n");
    }else if(event==LV_EVENT_PRESS_LOST)
    {
        //在接下的过程中,手指从对象的可视区域划出时被触发
        printf("LV_EVENT_PRESS_LOST\r\n");
    }else if(event==LV_EVENT_SHORT_CLICKED)
    {
        //在 LV_INDEV_LONG_PRESS_TIME 时间之前松手触发,如果是
        //在被拖拽的话,则不会被触发
        printf("LV_EVENT_SHORT_CLICKED\r\n");
    }else if(event==LV_EVENT_LONG_PRESSED)
    {
        //按下时长超过 LV_INDEV_LONG_PRESS_TIME 值时触发,
        //只会触发一次,如果是在被拖拽的话,则不会被触发
        printf("LV_EVENT_LONG_PRESSED\r\n");
    }else if(event==LV_EVENT_LONG_PRESSED_REPEAT)
    {
        //在触发 LV_EVENT_LONG_PRESSED 事件之后,接
        //下来接下的时长每超过 LV_INDEV_LONG_PRESS_REPEAT_TIME 值一次,就会
        //被触发一次,如果是在被拖拽的话,则不会被触发
        printf("LV_EVENT_LONG_PRESSED_REPEAT\r\n");
    }else if(event==LV_EVENT_CLICKED)
    {
        //只要松手了就会被触发,但是如果触发了 LV_EVENT_PRESS_LOST
        //事件的话,那么此事件将不会被触发
        printf("LV_EVENT_CLICKED\r\n");
    }else if(event==LV_EVENT_RELEASED)
    {
        //和 LV_EVENT_CLICKED 事件一样,没啥区别,位于 LV_EVENT_CLICKED 事
        //件之后触发
        printf("LV_EVENT_RELEASED\r\n");
    }
}
```

```
 }else if(event==LV_EVENT_DRAG_BEGIN)
 {
 //拖拽开始时触发
 printf("LV_EVENT_DRAG_BEGIN\r\n");
 }else if(event==LV_EVENT_DRAG_END)
 {
 //拖拽结束时触发
 printf("LV_EVENT_DRAG_END\r\n");
 }else if(event==LV_EVENT_DRAG_THROW_BEGIN)
 {
 //拖拽漂移开始时触发
 printf("LV_EVENT_DRAG_THROW_BEGIN\r\n");
 }else if(event==LV_EVENT_REFRESH)
 {
 //可以说是留给用户使用的一种事件,用户只能通过 lv_event_send
 //接口来手动发送触发此事件
 printf("LV_EVENT_REFRESH\r\n");
 }else if(event==LV_EVENT_DELETE)
 {
 //对象被删除时触发
 printf("LV_EVENT_DELETE\r\n");
 }else if(event==USER_EVENT_1)
 {
 //用户自定义事件 1
 //获取用户自定义数据
 USER_DATA* data = (USER_DATA*)lv_event_get_data();
 printf("USER_EVENT_1,name:%s,age:%d\r\n",data->name,data->age);
 }
 }

//例程入口函数
void events_test_start()
{
 lv_obj_t* scr = lv_scr_act(); //获取当前活跃的屏幕对象

//3. 创建一个默认按钮,用来测试事件
btn1 = lv_btn_create(scr, NULL);
lv_obj_set_pos(btn1,20,20); //设置坐标
lv_obj_set_size(btn1,150,50); //设置大小
lv_obj_set_event_cb(btn1,btn_event_cb); //设置回调函数
lv_obj_t* label = lv_label_create(btn1,NULL); //给 btn1 添加 label 子对象
lv_label_set_text(label,"Click me"); //设置文本
}
```

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

    if(btn1==NULL)
        return;
    if(key==KEY0_PRES)
    {
        //手动发送事件
        //方式 1:发送用户自定义事件,同时携带用户自定义数据
        lv_event_send(btn1,USER_EVENT_1,&user_data);

        //方式 2:发送系统自带的事件,不携带用户自定义数据
        //这里主要是为了演示一下 lv_event_send_func 接口
        //lv_event_send_func(btn_event_cb,btn1,LV_EVENT_REFRESH,NULL);
    }else if(key==KEY1_PRES)
    {
        //使能之后,主要是用来测试// LV_EVENT_DRAG_THROW_BEGIN 三个事件的
        lv_obj_set_drag(btn1,true);//使能拖拽功能
        lv_obj_set_drag_throw(btn1,true);//使能拖拽惯性漂移功能
    }else if(key==KEY2_PRES)
    {
        //删除对象,主要是用来测试 LV_EVENT_DELETE 事件
        if(btn1)
        {
            lv_obj_del(btn1);
            btn1 = NULL;
        }
    }
}
```

3.4 下载验证

把代码下载进去之后,正常的话,会看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

然后大家可以按照功能简介中的操作,利用串口助手来观看实验现象

图 3.4.2 串口输出信息

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_led 指示灯

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

lv_led 指示灯

1. 介绍

lv_led 控件就是一个简单的指示灯,它的外形可以通过样式来修改,简单的来说一般为一个小圆点或者一个小矩形,它具有一个 0 到 255 级可调的亮度属性,当为 0 时代表最暗,为 255 时代表最亮,基于此亮度属性,又引出了 OFF 和 ON 两种状态的概念,当为 OFF 状态时,其本质就是设置亮度值为 100,当为 ON 状态时,其本质就是设置亮度值为 255,整体上来说,这个控件的使用非常简单.

2. lv_led 的 API 接口

2.1 主要数据类型

因为 lv_led 控件太过于简单,没有什么可以介绍的数据类型

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_led_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置亮度值

```
void lv_led_set_bright(lv_obj_t * led, uint8_t bright);
```

参数:

led: 指示灯对象

bright: 亮度值,范围为[0,255]

2.2.3 设置为 ON 状态

```
void lv_led_on(lv_obj_t * led);
```

参数:

led: 指示灯对象

这个 API 接口的实现本质就是在调用 lv_led_set_bright 接口,如下图所示:

```
122 /**
123  * Light on a LED
124  * @param led pointer to a LED object
125  */
126 void lv_led_on(lv_obj_t * led)
127 {
128     lv_led_set_bright(led, LV_LED_BRIGHT_ON);
129 }
130 
```

图 2.2.3.1 lv_led_on 的实现原理

2.2.4 设置 OFF 状态

```
void lv_led_off(lv_obj_t * led);
```

参数:

led: 指示灯对象

这个 API 接口的实现本质也是在调用 lv_led_set_bright 接口,如下图所示:

```
131 /**
132  * Light off a LED
133  * @param led pointer to a LED object
134  */
135 void lv_led_off(lv_obj_t * led)
136 {
137     lv_led_set_bright(led, LV_LED_BRIGHT_OFF);
138 }
```

图 2.2.4.1 lv_led_off 的实现原理

2.2.5 切换指示灯的 ON 和 OFF 状态

```
void lv_led_toggle(lv_obj_t * led);
```

参数:

led: 指示灯对象

如果指示灯当前为 ON 状态,调用此接口之后,就会变成 OFF 状态,如果当前为 OFF 状态,调用此接口之后,就会变成 ON 状态

2.2.6 设置样式

```
static inline void lv_led_set_style(lv_obj_t * led, lv_led_style_t type, const lv_style_t * style);
```

参数:

led: 指示灯对象

type: 设置那部分的样式,目前就 LV_LED_STYLE_MAIN 这一个可选值

style: 样式

我们可以通过设置样式来给 LED 指示灯不同的外表

2.2.7 获取亮度值

```
uint8_t lv_led_get_bright(const lv_obj_t * led);
```

参数:

led: 指示灯对象

返回值:

返回当前的亮度值

3.例程设计

3.1 功能简介

先创建一个自定义的样式,然后接着创建 2 个 LED 指示灯,给它们设置刚创建好的新样式,让 LED1 默认处于 OFF 状态,让 LED2 具有一个默认的亮度值,当按下 KEY0 键时,我们来回切换 LED1 的 OFF 和 ON 状态,当按下 KEY1 键时,我们来增加 LED2 的亮度值,当按下 KEY2 键时,我们来减少 LED2 的亮度值

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1,KEY2 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_led_test.c 和 lv_led_test.h 俩个文件,其中 lv_led_test.c 文件的内容如下:

```
#include "lv_led_test.h"
#include "lvgl.h"
#include "key.h"

lv_obj_t * led1,* led2;
int16_t led2_bright = 190;//先给个默认亮度

//例程入口函数
void lv_led_test_start()
{
    lv_obj_t* scr = lv_scr_act();//获取当前活跃的屏幕对象

    //1.创建指示灯的样式
    static lv_style_t led_style;
    lv_style_copy(&led_style, &lv_style_pretty_color);//样式拷贝
    led_style.body.radius = LV_RADIUS_CIRCLE;//绘制半圆角
    //主背景的上半部分的颜色
    led_style.body.main_color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
    //主背景的下半部分的颜色
```

```
led_style.body.grad_color = LV_COLOR_MAKE(0x50, 0x07, 0x02);
led_style.body.border.color = LV_COLOR_MAKE(0xfa, 0x0f, 0x00); //边框颜色
led_style.body.border.width = 3; //边框的宽度
led_style.body.border.opa = LV_OPA_30; //透明度
led_style.body.shadow.color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04); //阴影的颜色
led_style.body.shadow.width = 5; //阴影的大小

//2.创建 LED1,并设置其默认为 OFF 状态
led1 = lv_led_create(scr, NULL);
lv_obj_set_pos(led1, 50, 50); //设置坐标
lv_obj_set_size(led1, 30, 30); //设置大小
lv_obj_set_style(led1, &led_style); //设置样式
lv_led_off(led1);

//3.创建 LED2,先直接从 LED1 拷贝过来
led2 = lv_led_create(scr, led1);
lv_obj_align(led2, led1, LV_ALIGN_OUT_BOTTOM_MID, 0, 40); //设置对齐方式
lv_led_set_bright(led2, led2_bright); //设置亮度值
}

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

    if(key==KEY0_PRES)
    {
        lv_led_toggle(led1); //来回切换 LED1 的状态
    }else if(key==KEY1_PRES)
    {
        //增加 LED2 的亮度
        led2_bright += 20;
        if(led2_bright>255)
            led2_bright = 0;
        lv_led_set_bright(led2, led2_bright);
    }else if(key==KEY2_PRES)
    {
        //减少 LED2 的亮度
        led2_bright -= 20;
        if(led2_bright<0)
            led2_bright = 255;
        lv_led_set_bright(led2, led2_bright);
    }
}
```

3.4 下载验证

把代码下载进去之后,正常的话,会看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

然后大家可以按照功能简介中的操作,按下 KEY0,KEY1 或 KEY2 键来看其他的现象

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_arc 弧形

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

lv_arc 弧形

1. 介绍

lv_arc 是一个用来绘制弧形的控件,和 lv_led 控件一样,非常的简单,通过样式中的 line 字段来修改弧形的外观,如下所示:

```
my_style.line.color : 用来设置弧形的颜色  
my_style.line.width : 用来设置弧线的厚度  
my_style.line.rounded : 弧线的末端是否为圆角
```

除了弧形的外观描述外,我们要想把一个弧形给绘制出来,还需要起始角度和终止角度两个重要的参数,这是通过 lv_arc_set_angles(arc, start_angle, end_angle) 接口来设置的,可能有的朋友就郁闷了,难道没有弧形的半径参数嘛?答案是确实没有半径参数,但是在 littleVGL 中,它是通过 lv_obj_set_size(arc, width, height) 接口设置弧形的大小来间接的实现半径效果,在这里传入的 width 宽度和 height 高度值必须相等,而且你可以认为 width/2 就是弧形的半径,不过这里有一点遗憾的是 lv_arc 控件目前还不支持抗锯齿功能.

2. lv_arc 的 API 接口

2.1 主要数据类型

因为 lv_arc 控件太过于简单,没有什么可以介绍的数据类型

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_arc_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置起始角度和终止角度

```
void lv_arc_set_angles(lv_obj_t * arc, uint16_t start, uint16_t end);
```

参数:

arc: 弧形对象

start: 起始角度,范围为[0,360]

end: 终止角度,范围为[0,360]

以对象的底部正中间为 0 度点,然后角度以逆时针的方向往上增加,如下图所示:

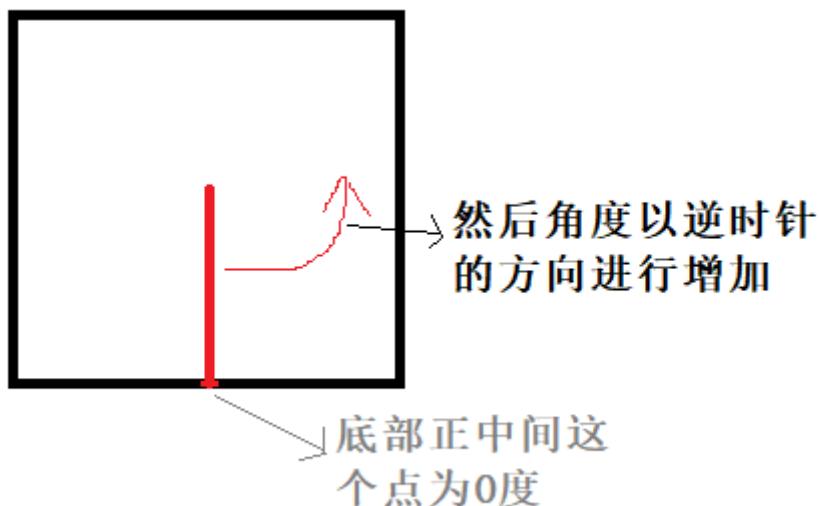


图 2.2.2.1 角度示意图

有了起始角度和终止角度,再加上一个半径,我们就可以绘制出一个弧形了,然后这里所谓的半径就是对象的宽度的一半,对象的宽度和高度必须是保持相等的

2.2.3 设置样式

```
void lv_arc_set_style(lv_obj_t * arc, lv_arc_style_t type, const lv_style_t * style);
```

参数:

led: 指示灯对象

type: 设置那部分的样式,目前就 LV_ARC_STYLE_MAIN 这一个可选值

style: 样式

我们可以利用样式中的 line 属性来设置弧形的厚度和颜色等

3.例程设计

3.1 功能简介

创建一个自定义的样式和 3 个弧形对象,让弧形 1 中的终止角度比起始角度小,让弧形 2 中的终止角度比起始角度大,用弧形 3 和一个 lv_label 标签来实现一个环形进度条,为了方便演示,我们这里创建一个任务来模拟进度的加载过程,然后利用 KEY0 按键来控制是否暂停加载

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_arc_test.c 和 lv_arc_test.h 俩个文件,其中 lv_arc_test.c 文件的内容如下:

```
#include "lv_arc_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_style_t arc_style;
lv_obj_t * arc3;
lv_obj_t * progress_label;
uint8_t progress = 0;//当前的进度,范围[0,100]
uint8_t is_pause = 0;//是否暂停加载过程

//弧形 3 设置进度
//progress:进度值,范围为[0,100]
static void arc3_set_progress(uint8_t progress)
{
    char buff[10];
    lv_arc_set_angles(arc3,0,(uint16_t)(3.6f*progress));

    sprintf(buff,"%d%%",progress);
```

```
lv_label_set_text(progress_label,buff);
lv_obj_realign(progress_label); //重新与 arc3 居中对齐
}

//任务回调函数
static void progress_task(lv_task_t * t)
{
    if(is_pause)
        return;
    progress++;
    if(progress>100)
        progress = 0;
    arc3_set_progress(progress);
}

//例程入口
void lv_arc_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.先创建 1 个样式
    lv_style_copy(&arc_style, &lv_style_plain);
    arc_style.line.color = LV_COLOR_RED; //弧形的颜色
    arc_style.line.width = 8; //弧形的厚度
    arc_style.line.rounded = 1; //末端为圆角,如果为 0 的话,则为直角

    //2.创建弧形 1(让终止角度比起始角度小)
    lv_obj_t * arc1 = lv_arc_create(scr, NULL); //创建弧形对象
    lv_arc_set_style(arc1, LV_ARC_STYLE_MAIN, &arc_style); //设置样式
    lv_arc_set_angles(arc1, 180, 90); //设置角度
    //设置大小,设置的宽度和高度必须得相等,弧形半径等于宽度的一半
    lv_obj_set_size(arc1, 100, 100);
    lv_obj_set_pos(arc1, 20, 20); //设置坐标

    //3.创建弧形 2(让终止角度比起始角度大)
    lv_obj_t * arc2 = lv_arc_create(scr, arc1); //直接从 arc1 拷贝
    //设置与 arc1 的对齐方式
    lv_obj_align(arc2, arc1, LV_ALIGN_OUT_RIGHT_TOP, 10, 0);
    lv_arc_set_angles(arc2, 90, 180); //设置角度

    //4.创建弧形 3,实现环形进度条的效果
    //4.1 创建弧形 3
    arc3 = lv_arc_create(scr, arc1); //直接从 arc1 拷贝
```

```
//设置与 arc1 的对齐方式
lv_obj_align(arc3,arc1,LV_ALIGN_OUT_BOTTOM_LEFT,0,10);

//4.2 创建进度指示 label
progress_label = lv_label_create(scr,NULL);
lv_obj_align(progress_label,arc3,LV_ALIGN_CENTER,0,0);//与 arc3 居中对齐

//4.3 设置一个默认的进度值
arc3_set_progress(progress);

//4.4 创建一个任务来模拟进度的加载过程
lv_task_create(progress_task,800,LV_TASK_PRIO_MID,NULL);
}

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

    if(key==KEY0_PRES)
    {
        is_pause = !is_pause;
    }
}
```

3.4 下载验证

把代码下载进去之后,环形进度条默认会从进度 0 开始慢慢加载到 100,如下图所示:

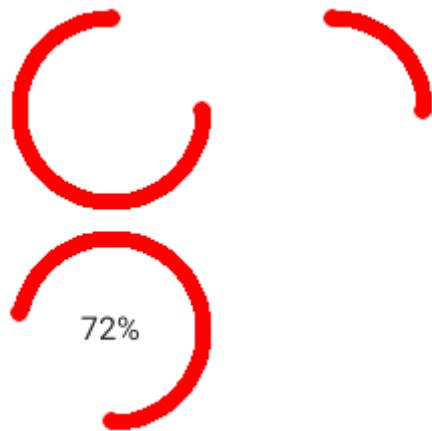


图 3.4.1 弧形演示效果

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

正点原子 littleVGL 开发指南

lv_bar 进度条

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

lv_bar 进度条

1. 介绍

lv_bar 进度条它由背景和指示器俩部分构成,这两部分的样式都可以被单独设置,然后此进度条会根据它的宽和高的大小来自动决定它是水平进度条还是垂直进度条,可以通过 lv_bar_set_range 接口来设置进度条的数值范围,通过 lv_bar_set_value 接口来设置一个新的进度值,与此同时可选一个进度变化的动画效果,整体上来看,这个控件的使用还是比较简单的

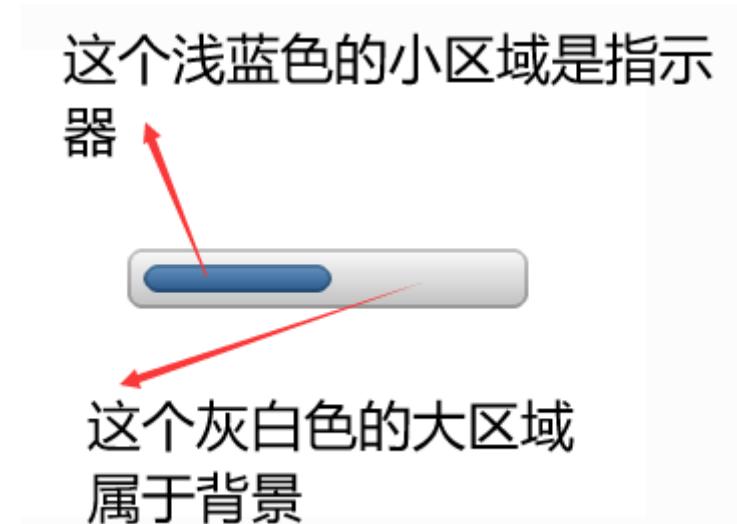


图 1.1 lv_bar 进度条的组成

2. lv_bar 的 API 接口

2.1 主要数据类型

3. 进度条样式数据类型

```
enum {
    LV_BAR_STYLE_BG,
    LV_BAR_STYLE_INDIC,
};

typedef uint8_t lv_bar_style_t;
```

LV_BAR_STYLE_BG: 背景的样式,默认值为 lv_style_pretty

LV_BAR_STYLE_INDIC: 指示器的样式 , 默认值为 lv_style_pretty_color, 它使用 style.body.padding 里面的 left, top, right, bottom 等字段来控制与背景边框之间的距离

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_bar_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置进度范围

```
void lv_bar_set_range(lv_obj_t * bar, int16_t min, int16_t max);
```

参数:

bar: 进度条对象

min: 最小值

max: 最大值

如果不设置的话,则默认范围为[0,100]

2.2.3 设置新的进度值

```
void lv_bar_set_value(lv_obj_t * bar, int16_t value, lv_anim_enable_t anim);
```

参数:

bar: 进度条对象

value: 新的进度值,此值需要在进度范围内

anim: 在切换到新的进度值时,是否使能动画效果,LV_ANIM_ON 代表使能,LV_ANIM_OFF 代表 不使能

2.2.4 设置动画时长

```
void lv_bar_set_anim_time(lv_obj_t * bar, uint16_t anim_time);
```

参数:

bar: 进度条对象

anim_time: 动画时长,单位为 ms

2.2.5 设置样式

```
void lv_bar_set_style(lv_obj_t * bar, lv_bar_style_t type, const lv_style_t * style);
```

参数:

bar: 进度条对象

type: 要设置哪一个部件的样式,有俩个可选值如下:

LV_BAR_STYLE_BG: 是设置背景的样式

LV_BAR_STYLE_INDIC: 是设置指示器的样式

style: 样式

2.2.6 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建2个自定义的样式,一个是用来修饰进度条的背景,一个是用来修饰进度条的指示器,然后再创建2个进度条对象,一个为水平进度条,一个为垂直进度条,给它们设置一个新的进度值,同时使能动画效果

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_bar_test.c 和 lv_bar_test.h 俩个文件,其中 lv_bar_test.c 文件的内容如下:

```
#include "lv_bar_test.h"
#include "lvgl.h"

lv_style_t bar_bg_style;//进度条的背景样式
lv_style_t bar_indic_style;//进度条的指示器样式

//例程入口
void lv_bar_test_start()
{
    lv_obj_t * scr = lv_scr_act();//获取当前活跃的屏幕对象

    //1.创建进度条的背景和指示器样式
    //1.1 创建背景样式
    lv_style_copy(&bar_bg_style,&lv_style_plain_color);
    bar_bg_style.body.main_color = LV_COLOR_MAKE(0xBB,0xBB,0xBB);
    bar_bg_style.body.grad_color = LV_COLOR_MAKE(0xBB,0xBB,0xBB);
    bar_bg_style.body.radius = LV_RADIUS_CIRCLE;//绘制圆角

    //1.2 创建指示器样式
    lv_style_copy(&bar_indic_style,&lv_style_plain_color);
```

```
bar_indic_style.body.main_color = LV_COLOR_MAKE(0x5F,0xB8,0x78);
bar_indic_style.body.grad_color = LV_COLOR_MAKE(0x5F,0xB8,0x78);
bar_indic_style.body.radius = LV_RADIUS_CIRCLE;//绘制圆角
bar_indic_style.body.padding.left = 0;//让指示器跟背景边框之间没有距离
bar_indic_style.body.padding.top = 0;
bar_indic_style.body.padding.right = 0;
bar_indic_style.body.padding.bottom = 0;

//2.创建水平进度条
lv_obj_t * bar1 = lv_bar_create(scr, NULL);//创建进度条
lv_obj_set_size(bar1,180,16);//设置大小,宽度比高度大就是水平的
lv_obj_set_pos(bar1,20,20);//设置坐标
lv_bar_set_style(bar1,LV_BAR_STYLE_BG,&bar_bg_style);//设置进度条背景的样式
//设置进度条指示器的样式
lv_bar_set_style(bar1,LV_BAR_STYLE_INDIC,&bar_indic_style);
lv_bar_set_anim_time(bar1,1000);//设置动画时长
lv_bar_set_value(bar1,100,LV_ANIM_ON);//设置新的进度值,带有动画效果的

//3.创建垂直进度条
lv_obj_t * bar2 = lv_bar_create(scr, bar1);//从 bar1 进行拷贝
lv_obj_set_size(bar2,16,180);//设置大小,宽度比高度小就是垂直的
//设置与 bar1 的对齐方式
lv_obj_align(bar2,bar1,LV_ALIGN_OUT_BOTTOM_LEFT,0,10);
lv_bar_set_range(bar2,100,200);//设置进度范围
lv_bar_set_anim_time(bar2,1000);//设置动画时长
lv_bar_set_value(bar2,180,LV_ANIM_ON);//设置新的进度值,180 正好是 80%的进度
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的演示效果:

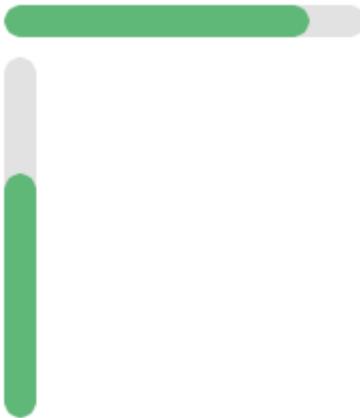


图 3.4.1 进度条演示效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_cb 复选框

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

lv_cb 复选框

1. 介绍

lv_cb 复选框是由 3 部分组成的,第一部分是最外层的背景,第二部分是内部左侧的小方块,这个小方块其实就是一个 lv_btn 按钮,所以这个小方块也具有 5 种状态和相应的 5 个样式,第三部分是内部右侧的文本,这个文本其实就是 lv_label 标签,可以通过 lv_cb_set_text 接口来给复选框设置文本,通过 lv_cb_set_checked 接口来设置复选框是否被选中,当复选框被点击时,它会发送一个 LV_EVENT_VALUE_CHANGED 事件给它的事件回调函数,然后就是复选框的大小是自适应的,也就是说使用 lv_obj_set_size 接口来设置复选框的大小是无效的,它只会根据它内部的文本大小来决定自身的大小.

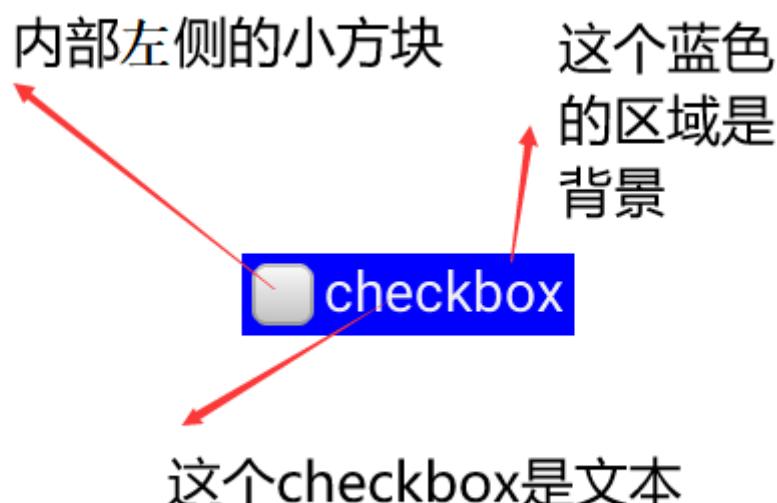


图 1.1 lv_cb 复选框的组成

注:默认创建出来的复选框背景区域是透明的,我这里是为了方便给大家讲解,才把它弄成了蓝色

2. lv_cb 的 API 接口

2.1 主要数据类型

3. 复选框样式数据类型

```
enum {
    LV_CB_STYLE_BG,
    LV_CB_STYLE_BOX_REL,
    LV_CB_STYLE_BOX_PR,
    LV_CB_STYLE_BOX_TGL_REL,
    LV_CB_STYLE_BOX_TGL_PR,
    LV_CB_STYLE_BOX_INA,
};

typedef uint8_t lv_cb_style_t;
```

这上面有 6 种样式,其中只有 `LV_CB_STYLE_BG` 这一种样式是用来修饰背景和文本的,其他的 5 种样式都是用来修饰内部左侧的小方块的,因为我们前面说过这个小方块其实就是 `lv_btn` 按钮,所以这里的 5 种样式就是对应着按钮的 5 种样式,跟前面按钮章节中的内容是一样的

LV_CB_STYLE_BG: 使用 `style.body` 来修饰背景部分,使用 `style.text` 来修饰文本部分,这种样式的默认值为 `lv_style_transp`

剩下的其他 5 种样式这里不多讲了,请看”`lv_btn` 按钮”那一章节的内容,它们的功能是一模一样的

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_cb_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

`par`: 父对象

`copy`: 拷贝的对象,如果无拷贝的话,传 `NULL` 值

返回值:

返回创建出来的对象,如果返回 `NULL` 的话,说明堆空间不够了

2.2.2 设置动态文本

```
void lv_cb_set_text(lv_obj_t * cb, const char * txt);
```

参数:

cb: 复选框对象

txt: 文本内容

这里动态的意思是说 littleVGL 内部会为这个文本内容重新分配相应大小的堆空间

2.2.3 设置静态文本

```
void lv_cb_set_static_text(lv_obj_t * cb, const char * txt);
```

参数:

cb: 复选框对象

txt: 文本内容

这里静态的意思是说 littleVGL 内部不会为这个文本内容分配堆空间,所以你得保证这个 txt 文本资源在外面不会被释放

2.2.4 设置复选框是否选中

```
static inline void lv_cb_set_checked(lv_obj_t * cb, bool checked);
```

参数:

cb: 复选框对象

checked: 是否选中复选框

2.2.5 设置复选框为禁用状态

```
static inline void lv_cb_set_inactive(lv_obj_t * cb);
```

参数:

cb: 复选框对象

当调用此接口把复选框设置为禁用状态后,内部左侧的小方块就会显示出 LV_CB_STYLE_BOX_INA 指定的样式外观,与此同时不管用户怎么点击,复选框的状态都不会再被改变了,但是只要有点击,复选框就会把 LV_EVENT_VALUE_CHANGED 事件发送给它的事件回调函数

2.2.6 设置样式

```
void lv_cb_set_style(lv_obj_t * cb, lv_cb_style_t type, const lv_style_t * style);
```

参数:

cb: 复选框对象

type: 设置那一部分的样式

style: 样式

2.2.7 判断复选框是否被选中了

```
static inline bool lv_cb_is_checked(const lv_obj_t * cb);
```

参数:

cb: 复选框对象

返回值:

返回 true 代表复选框此时被选中了,返回 false 代表复选框此时没有被选中

2.2.8 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式,用来修饰复选框的背景和文本,然后接着创建一个复选框,并为其注册事件回调函数,最后再创建一个标签,用来显示复选框的当前选中状态

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_cb_test.c 和 lv_cb_test.h 俩个文件,其中 lv_cb_test.c 文件的内容如下:

```
#include "lv_cb_test.h"
#include "lvgl.h"

lv_style_t cb_bg_style;
lv_obj_t * label1;

//事件回调函数
static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED)
    {
        lv_label_set_text(label1,lv_cb_is_checked(obj)?"Checked":"Unchecked");
    }
}

//例程入口
void lv_cb_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象
```

```
//1.创建背景样式
lv_style_copy(&cb_bg_style,&lv_style_plain_color);

//1.1 body 字段是用来修饰背景的
cb_bg_style.body.opa = LV_OPA_TRANSP;//设置背景透明

//1.2 text 字段是用来修饰内部右侧的文本
cb_bg_style.text.letter_space = 3;//字符之间的距离
cb_bg_style.text.color = LV_COLOR_RED;//设置文本的颜色

//2.创建复选框
lv_obj_t * cb1 = lv_cb_create(scr, NULL);
lv_cb_set_text(cb1,"checkbox");//设置文本
lv_obj_set_pos(cb1,40,40);//设置坐标
//调用此接口来设置复选框的大小是无效的,因为复选框的大小是自适应的
//lv_obj_set_size(cb1,100,50);
lv_cb_set_style(cb1,LV_CB_STYLE_BG,&cb_bg_style);//设置样式
lv_cb_set_checked(cb1,false);//设置复选框没有被选中
lv_obj_set_event_cb(cb1,event_handler);//设置事件回调函数

//3.创建一个 label 来显示复选框的选中状态
label1 = lv_label_create(scr,NULL);
lv_label_set_text(label1,"Unchecked");
lv_obj_align(label1,cb1,LV_ALIGN_OUT_BOTTOM_LEFT,0,40);
lv_label_set_body_draw(label1,true);
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&lv_style_plain_color);
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

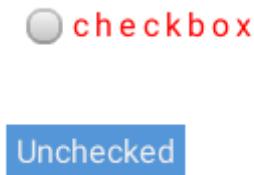


图 3.4.1 复选框未选中时的效果

然后我们点击一下 checkbox 复选框,就可以看到如下所示的效果:



图 3.4.2 复选框选中时的效果

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

正点原子 littleVGL 开发指南

lv_line 线条

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

lv_line 线条

1. 介绍

lv_line 线条简单来说是由多个点连接而成的对象,它可以通过 `lv_obj_set_size` 接口来设置固定的大小,也可以通过 `lv_line_set_auto_size(line, true)` 接口来设置线条对象的大小自适应,它会根据其内部所有点中最大的 x 和 y 坐标来算出自身的大小,默认情况下,大小自适应功能是被使能了的,对于线条对象来说,其内部的所有点坐标默认情况下都是以其左上角为参考原点的,当然了,你可以通过 `lv_line_set_y_invert(line, true)` 接口来反转 y 轴,从而使左下角变为参考原点.

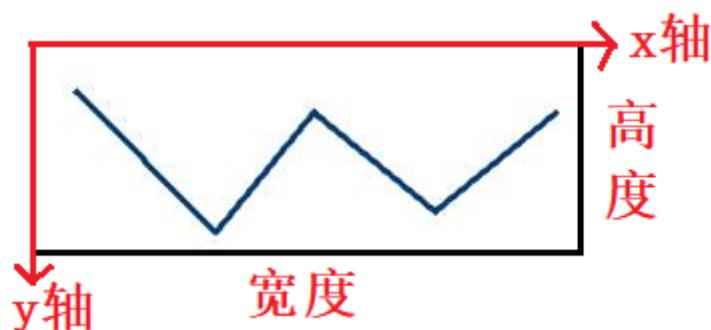


图 1.1 lv_line 对象组成

注:创建出来的 `lv_line` 对象背景是透明的,而且也是没有边框的,上面图片中具有的黑色和红色边框是为了给大家讲解方便,才加上去的

2. lv_line 的 API 接口

2.1 主要数据类型

由于 lv_line 控件过于简单,所以没有啥可讲的数据类型

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_line_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置坐标点集合

```
void lv_line_set_points(lv_obj_t * line, const lv_point_t point_a[], uint16_t point_num);
```

参数:

line: 线条对象

point_a: 坐标点集合,数组的形式,如下所示:

```
const lv_point_t line_points[] = { {20, 30}, {70, 70}, {120, 10} };
```

point_num: 坐标点的个数

线是由点构成的,这里传入 point_num 个坐标点,就可以绘制出 point_num-1 条线

2.2.3 是否使能大小自适应

```
void lv_line_set_auto_size(lv_obj_t * line, bool en);
```

参数:

line: 线条对象

en: 是否使能

默认情况下,线条对象的大小自适应功能是被使能了的,它会根据其内部所有点中最大的 x 和 y 坐标来算出自身的大小,这个大小计算的过程其实是在 lv_line_set_points 接口内部完成的

2.2.4 是否设置 y 轴反转

```
void lv_line_set_y_invert(lv_obj_t * line, bool en);
```

参数:

line: 线条对象

en: 是否使能

默认情况下是以 lv_line 对象的左上角为参考原点的,如果你调用了 lv_line_set_y_invert 接口来反转 y 轴的话,那么参考原点就会变为左下角

2.2.5 设置样式

```
static inline void lv_line_set_style(lv_obj_t * line, lv_line_style_t type, const lv_style_t * style);
```

参数:

line: 线条对象

type: 设置哪一个部件的样式,目前就只有 LV_LINE_STYLE_MAIN 一个可选值

style: 样式

只有 style.line 字段才会被 lv_line 对象使用到

2.2.6 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式,用来修饰线条对象,然后接着创建一个线条对象,并为其设置好坐标点集合,当按下 KEY0 键时,我们让此线条对象进行 y 轴反转

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_line_test.c 和 lv_line_test.h 俩个文件,其中 lv_line_test.c 文件的内容如下:

```
#include "lv_line_test.h"
#include "lvgl.h"
#include "key.h"

//坐标点集合
const lv_point_t line_points[] = { {10, 20}, {70, 50}, {120, 10}, {140, 60}, {180, 10} };
//坐标点的个数
#define LINE_POINTS_NUM           (sizeof(line_points)/sizeof(line_points[0]))

lv_obj_t * line1;

//例程入口
void lv_line_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建自定义样式
    static lv_style_t line_style;
    lv_style_copy(&line_style, &lv_style_plain);
    line_style.line.color = LV_COLOR_RED; //线条的颜色
    line_style.line.width = 4; //线条的厚度
```

```
line_style.line.rounded = 1;//线条的末端是否为圆角

//2.创建线条对象
line1 = lv_line_create(scr, NULL);//创建线条对象
lv_obj_set_pos(line1, 20, 20);//设置坐标
//使能大小自适应,当然了,你也可以不调用,因为默认就是被使能了的
lv_line_set_auto_size(line1, true);
//设置坐标点集合,同时也会在此内部计算出线条对象的大小
lv_line_set_points(line1, line_points, LINE_POINTS_NUM);
lv_line_set_style(line1, LV_LINE_STYLE_MAIN, &line_style);//设置样式
}

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

    if(key==KEY0_PRES)
    {
        lv_line_set_y_invert(line1,!lv_line_get_y_invert(line1));//来回取反
    }
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 y 轴未反转时的效果

然后我们按一下 KEY0 键来反转 y 轴,就可以看到如下所示的效果:



图 3.4.2 y 轴反转后的效果

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_slider 滑块

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

lv_slider 滑块

1. 介绍

lv_slider 滑块是由 lv_bar 进度条对象外加一个类似于旋钮的东西构成的,这个旋钮可以被拖拽来设置 lv_slider 滑块的值,和 lv_bar 进度条一样,lv_slider 也可以被设置成是水平滑块或者是垂直滑块,不仅如此,lv_bar 对象上的大部分特性在 lv_slider 对象上基本都能找到,比如设置进度值,动画时间,设置最小最大范围等等,API 接口的用法基本是一模一样的.最后来说一下它的事件,当滑块被点击或者滑块上面的旋钮被拖拽导致其进度值发生变化时,它就会给它的事件回调函数发送一个 LV_EVENT_VALUE_CHANGED 事件,如果旋钮是在被持续拖拽的话,那么 LV_EVENT_VALUE_CHANGED 事件也将会被持续发送,有时候我们可能不希望持续接受到此事件,那我们只需忽略就行,只监听它的 LV_EVENT_RELEASED 松手事件来获取最后的进度值.

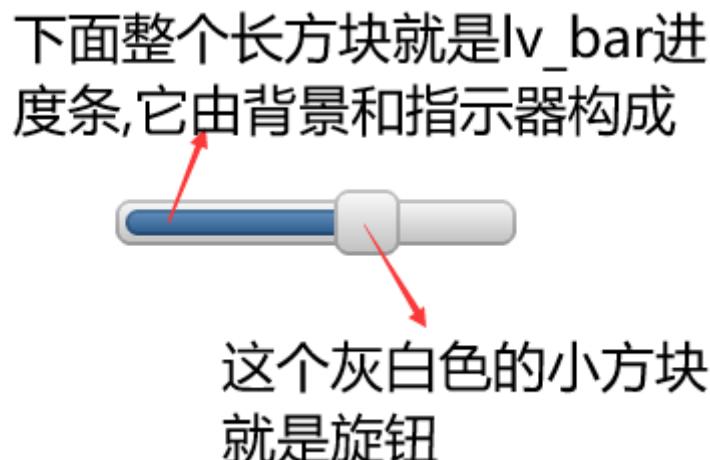


图 1.1 lv_slider 对象组成

2. lv_slider 的 API 接口

2.1 主要数据类型

3. 滑块样式数据类型

```
enum {
    LV_SLIDER_STYLE_BG,
    LV_SLIDER_STYLE_INDIC,
    LV_SLIDER_STYLE_KNOB,
};
```

typedef uint8_t lv_slider_style_t;

LV_SLIDER_STYLE_BG: 滑块的背景样式,其实就是其内部的 lv_bar 进度条的背景样式,使用样式中的 style.body 字段,其中的 padding 字段用来设置背景边框与旋钮边框之间的距离

LV_SLIDER_STYLE_INDIC: 滑块的指示器样式,其实就是其内部的 lv_bar 进度条的指示器样式,使用样式中的 style.body 字段,其中的 padding 字段设置指示器与背景边框之间的距离

LV_SLIDER_STYLE_KNOB: 滑块上旋钮的样式,使用样式中的 style.body 字段,但是其内部的 padding 字段除外

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_slider_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置动画时长

```
static inline void lv_slider_set_anim_time(lv_obj_t * slider, uint16_t anim_time);
```

参数:

slider: 滑块对象

anim_time: 动画时长,单位 ms

注意此 API 接口必须得放在 lv_slider_set_value 接口前面调用,否则无效

2.2.3 设置进度值

```
static inline void lv_slider_set_value(lv_obj_t * slider, int16_t value, lv_anim_enable_t anim);
```

参数:

slider: 滑块对象

value: 新的进度值

anim: 在切换到新的进度值时,是否使能动画效果,有 2 个可选值如下:

LV_ANIM_OFF: 不使能动画效果

LV_ANIM_ON: 使能动画效果

2.2.4 设置进度范围

```
static inline void lv_slider_set_range(lv_obj_t * slider, int16_t min, int16_t max);
```

参数:

slider: 滑块对象

min: 最小值

max: 最大值

2.2.5 设置样式

```
void lv_slider_set_style(lv_obj_t * slider, lv_slider_style_t type, const lv_style_t * style);
```

参数:

slider: 滑块对象

type: 设置哪一个部件的样式,有如下 3 个可选值:

LV_SLIDER_STYLE_BG: 设置背景的样式

LV_SLIDER_STYLE_INDIC: 设置指示器的样式

LV_SLIDER_STYLE_KNOB: 设置旋钮的按钮

style: 样式

2.2.6 获取当前的进度值

```
int16_t lv_slider_get_value(const lv_obj_t * slider);
```

参数:

slider: 滑块对象

返回值:

返回当前的进度值

2.2.7 判断旋钮是否正在被拖拽

```
bool lv_slider_is_dragged(const lv_obj_t * slider);
```

参数:

slider: 滑块对象

返回值:

返回 true 代表正在被拖拽,返回 false 代表没有在被拖拽

2.2.8 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建3个自定义样式,分别用于修饰滑块的背景,指示器和旋钮,然后创建一个水平滑块和一个标签,并为滑块对象设置事件回调函数,在事件回调函数中,把滑块的当前进度值显示在标签上

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_slider_test.c 和 lv_slider_test.h 俩个文件,其中 lv_slider_test.c 文件的内容如下:

```
#include "lv_slider_test.h"
#include "lvgl.h"
#include <stdio.h>

lv_style_t slider_bg_style;//背景的样式
lv_style_t slider_indic_style;//指示器的样式
lv_style_t slider_knob_style;//旋钮的样式

lv_obj_t * label1;

//事件回调函数
static void event_handler(lv_obj_t * obj,lv_event_t event)
{
    char buff[16];
    //lv_slider 的进度值发生了改变或者松手了
    if(event==LV_EVENT_VALUE_CHANGED||event==LV_EVENT_RELEASED)
    {
        //将当前的进度显示在 label1 标签中,如果是松手事件的话,则加上"End:"前缀
        sprintf(buff,event==LV_EVENT_VALUE_CHANGED?"%d%%":"End:%d%%",lv_slider_get_value(obj));
        lv_label_set_text(label1,buff);
    }
}
```

```
lv_obj_realign(label1);
}

//例程入口
void lv_slider_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建用于滑块的 3 个样式
    //1.1 创建背景样式
    lv_style_copy(&slider_bg_style,&lv_style_pretty);
    slider_bg_style.body.main_color = LV_COLOR_BLACK;
    slider_bg_style.body.grad_color = LV_COLOR_GRAY;
    slider_bg_style.body.radius = LV_RADIUS_CIRCLE;
    slider_bg_style.body.border.color = LV_COLOR_WHITE;

    //1.2 创建指示器的样式
    lv_style_copy(&slider_indic_style,&lv_style_pretty_color);
    slider_indic_style.body.main_color = LV_COLOR_MAKE(0x5F,0xB8,0x78);
    slider_indic_style.body.grad_color = LV_COLOR_MAKE(0x5F,0xB8,0x78);
    slider_indic_style.body.radius = LV_RADIUS_CIRCLE;
    slider_indic_style.body.shadow.width = 8;
    slider_indic_style.body.shadow.color = slider_indic_style.body.main_color;
    slider_indic_style.body.padding.left = 3; //设置指示器与背景边框之间的距离
    slider_indic_style.body.padding.right = 3;
    slider_indic_style.body.padding.top = 3;
    slider_indic_style.body.padding.bottom = 3;

    //1.3 创建旋钮的样式
    lv_style_copy(&slider_knob_style,&lv_style_pretty);
    slider_knob_style.body.radius = LV_RADIUS_CIRCLE;
    slider_knob_style.body.opa = LV_OPA_70;

    //2.创建滑块对象
    lv_obj_t * slider1 = lv_slider_create(scr,NULL);

    //设置大小,当宽度比高度大时,是水平滑块,当宽度比高度小时,是垂直滑块
    lv_obj_set_size(slider1,200,20);
    lv_slider_set_range(slider1,0,100); //设置进度范围

    //设置动画时长,必须得放在 lv_slider_set_value 前面调用,否则无效
    lv_slider_set_anim_time(slider1,1000);
    lv_slider_set_value(slider1,70,LV_ANIM_ON); //设置当前的进度值,使能动画效果
```

```
//设置背景样式  
lv_slider_set_style(slider1,LV_SLIDER_STYLE_BG,&slider_bg_style);  
//设置指示器的样式  
lv_slider_set_style(slider1,LV_SLIDER_STYLE_INDIC,&slider_indic_style);  
//设置旋钮的样式  
lv_slider_set_style(slider1,LV_SLIDER_STYLE_KNOB,&slider_knob_style);  
lv_obj_align(slider1,NULL,LV_ALIGN_CENTER,0,0);//设置与屏幕居中对齐  
lv_obj_set_event_cb(slider1,event_handler);//设置事件  
  
//3.创建标签,用来显示滑块的当前进度  
label1 = lv_label_create(scr,NULL);  
lv_obj_align(label1,slider1,LV_ALIGN_OUT_TOP_MID,0,-10);  
lv_label_set_text(label1,"70%");  
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 滑块演示效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_sw 开关

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

lv_sw 开关

1. 介绍

lv_sw 开关看起来像是一个缩小版的 lv_slider 滑块控件,那我可以告诉大家,它的本质就是一个 lv_slider 滑块控件,然后在此上面做一些小的升级,就变成了 lv_sw 开关控件,在前一章中,我们已经学会了 lv_slider 滑块控件的使用,利用上一章的知识,我们再来学习 lv_sw 开关控件,就会觉得顺心应手,lv_sw 控件具有开和关两种状态,我们可以通过如下三种方式来改变它的状态:

- 1) 点击这个开关控件
- 2) 滑动这个开关控件,它会根据你滑动的位置来自动决定状态的
- 3) 使用 lv_sw_on(sw, LV_ANIM_ON/OFF), lv_sw_off(sw, LV_ANIM_ON/OFF), lv_sw_toggle(sw, LV_ANOM_ON/OFF) 等 API 接口来改变其状态

当其状态发生改变后,它会给它的事件回调函数发送一个 LV_EVENT_VALUE_CHANGED 事件.但这里有一点需要注意一下,如果你是通过调用 lv_sw_on, lv_sw_off, lv_sw_toggle 等接口来改变开关状态的话,它是不会发送 LV_EVENT_VALUE_CHANGED 事件的.



图 1.1 lv_sw 对象的组成

2. lv_sw 的 API 接口

2.1 主要数据类型

3. 开关样式数据类型

```
enum {
    LV_SW_STYLE_BG,
    LV_SW_STYLE_INDIC,
    LV_SW_STYLE_KNOB_OFF,
    LV_SW_STYLE_KNOB_ON,
};

typedef uint8_t lv_sw_style_t;
```

LV_SW_STYLE_BG: 开关的背景样式,使用样式中的 style.body 字段,其中的 padding 字段用来设置背景边框与旋钮边框之间的距离

LV_SW_STYLE_INDIC: 开关的指示器样式,使用样式中的 style.body 字段,其中的 padding 字段设置指示器与背景边框之间的距离

LV_SW_STYLE_KNOB_OFF: 开关处于关闭状态时的旋钮样式,使用样式中的 style.body 字段,但是其内部的 padding 字段除外

LV_SW_STYLE_KNOB_ON: 开关处于打开状态时的旋钮样式,使用样式中的 style.body 字段,但是其内部的 padding 字段除外

lv_sw 开关样式的使用和 lv_slider 滑块样式的使用基本是一样的,唯一的区别就是 lv_sw 上的旋钮具有 2 种样式,分别对应开和关两种状态

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_sw_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置为打开状态

```
void lv_sw_on(lv_obj_t * sw, lv_anim_enable_t anim);
```

参数:

sw: 开关对象

anim: 在切换到打开状态时,是否使能动画效果,有 2 个可选值如下:

LV_ANIM_OFF: 不使能动画效果

LV_ANIM_ON: 使能动画效果

注意此 API 接口不会触发 LV_EVENT_VALUE_CHANGED 事件

2.2.3 设置为关闭状态

```
void lv_sw_off(lv_obj_t * sw, lv_anim_enable_t anim);
```

参数:

sw: 开关对象

anim: 在切换到关闭状态时,是否使能动画效果,有 2 个可选值如下:

LV_ANIM_OFF: 不使能动画效果

LV_ANIM_ON: 使能动画效果

注意此 API 接口不会触发 LV_EVENT_VALUE_CHANGED 事件

2.2.4 反转当前的状态

```
bool lv_sw_toggle(lv_obj_t * sw, lv_anim_enable_t anim);
```

参数:

sw: 开关对象

anim: 在反转状态时是否使能动画效果,有 2 个可选值如下:

LV_ANIM_OFF: 不使能动画效果

LV_ANIM_ON: 使能动画效果

返回值:

返回开关反转后的状态,true 代表为打开状态,false 代表关闭状态

注意此 API 接口不会触发 LV_EVENT_VALUE_CHANGED 事件

2.2.5 设置样式

```
void lv_sw_set_style(lv_obj_t * sw, lv_sw_style_t type, const lv_style_t * style);
```

参数:

sw: 开关对象

type: 设置哪一个部件的样式,有如下 4 个可选值:

LV_SW_STYLE_BG: 设置背景的样式

LV_SW_STYLE_INDIC: 设置指示器的样式

LV_SW_STYLE_KNOB_OFF: 设置开关处于关闭状态时的旋钮样式

LV_SW_STYLE_KNOB_ON: 设置开关处于打开状态时的旋钮样式

style: 样式

2.2.6 设置动画时长

```
void lv_sw_set_anim_time(lv_obj_t * sw, uint16_t anim_time);
```

参数:

sw: 开关对象

经过笔者测试,发现此 API 接口好像不起作用

2.2.7 获取开关的状态

```
static inline bool lv_sw_get_state(const lv_obj_t * sw);
```

参数:

sw: 开关对象

返回值:

返回 true 代表开关处于打开状态,返回 false 代表开关处于关闭状态

2.2.8 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建4个自定义样式,分别用于修饰开关的背景,指示器和旋钮,然后创建一个开关对象和一个标签对象,并为开关对象设置事件回调函数,在事件回调函数中,把开关的当前状态显示在标签上,如果用户按下KEY0键的话,则会通过lv_sw_toggle接口来反转开关的当前状态.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_sw_test.c 和 lv_sw_test.h 俩个文件,其中 lv_sw_test.c 文件的内容如下:

```
#include "lv_sw_test.h"
#include "lvgl.h"
#include "key.h"

lv_style_t sw_bg_style;
lv_style_t sw_indic_style;
lv_style_t sw_knob_off_style;
lv_style_t sw_knob_on_style;

lv_obj_t * sw1;
lv_obj_t * label1;

//label1 显示开关的状态
#define LABEL1_DISP_STATE(sta) lv_label_set_text(label1,sta?"State:#65C466 ON#":""
State:#FF0000 OFF#")

//事件回调函数
static void event_handler(lv_obj_t * obj,lv_event_t event)
```

```
{  
    if(event==LV_EVENT_VALUE_CHANGED)  
    {  
        LABEL1_DISP_STATE(lv_sw_get_state(obj));  
    }  
}  
  
//例程入口  
void lv_sw_test_start()  
{  
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象  
  
    //1.创建 4 个样式  
    //1.1 创建背景样式  
    lv_style_copy(&sw_bg_style,&lv_style_plain_color);  
    sw_bg_style.body.main_color = LV_COLOR_MAKE(0xCC,0xCC,0xCC); //灰白色  
    sw_bg_style.body.grad_color = LV_COLOR_MAKE(0xCC,0xCC,0xCC);  
    sw_bg_style.body.radius = LV_RADIUS_CIRCLE; //圆角  
  
    //设置背景边框与旋钮边框之间的距离,当设置为负数时,背景会比旋钮大  
    sw_bg_style.body.padding.left = -3;  
    sw_bg_style.body.padding.right = -3;  
    sw_bg_style.body.padding.top = -3;  
    sw_bg_style.body.padding.bottom = -3;  
  
    //1.2 创建指示器样式  
    lv_style_copy(&sw_indic_style,&lv_style_plain_color);  
    sw_indic_style.body.main_color = LV_COLOR_MAKE(0x65,0xC4,0x66); //浅绿色  
    sw_indic_style.body.grad_color = LV_COLOR_MAKE(0x65,0xC4,0x66);  
    sw_indic_style.body.radius = LV_RADIUS_CIRCLE; //圆角  
    sw_indic_style.body.padding.left = 0; //让指示器与背景边框之间无距离  
    sw_indic_style.body.padding.top = 0;  
    sw_indic_style.body.padding.right = 0;  
    sw_indic_style.body.padding.bottom = 0;  
  
    //1.3 创建关闭状态时,旋钮的样式  
    lv_style_copy(&sw_knob_off_style,&lv_style_plain_color);  
    sw_knob_off_style.body.main_color = LV_COLOR_WHITE; //纯白色  
    sw_knob_off_style.body.grad_color = LV_COLOR_WHITE;  
    sw_knob_off_style.body.radius = LV_RADIUS_CIRCLE; //圆角  
  
    //阴影颜色  
    sw_knob_off_style.body.shadow.color = LV_COLOR_MAKE(0xA0,0xA0,0xA0);  
    sw_knob_off_style.body.shadow.width = 6; //阴影宽度
```

```
//1.4 创建打开状态时,旋钮的样式  
//保持和关闭状态时的样式一样即可  
lv_style_copy(&sw_knob_on_style,&sw_knob_off_style);  
  
//2.创建开关对象  
sw1 = lv_sw_create(scr,NULL);  
lv_obj_set_size(sw1,100,50);//设置大小  
lv_obj_align(sw1,NULL,LV_ALIGN_CENTER,0,0);//设置与屏幕保持居中  
lv_sw_on(sw1,LV_ANIM_ON);//设置为打开状态,并带有动画效果  
lv_sw_set_style(sw1,LV_SW_STYLE_BG,&sw_bg_style);//设置背景样式  
lv_sw_set_style(sw1,LV_SW_STYLE_INDIC,&sw_indic_style);//设置指示器样式  
  
//设置关闭状态时,旋钮的样式  
lv_sw_set_style(sw1,LV_SW_STYLE_KNOB_OFF,&sw_knob_off_style);  
  
//设置打开状态时,旋钮的样式  
lv_sw_set_style(sw1,LV_SW_STYLE_KNOB_ON,&sw_knob_on_style);  
lv_obj_set_event_cb(sw1,event_handler);//设置事件回调函数  
  
//3.创建 label 标签,用来显示开关的当前状态  
label1 = lv_label_create(scr,NULL);  
lv_label_set_recolor(label1,true);//使能重绘色功能  
LABEL1_DISP_STATE(lv_sw_get_state(sw1));  
lv_obj_align(label1,sw1,LV_ALIGN_OUT_BOTTOM_MID,0,20);  
}  
  
void key_handler()  
{  
    u8 key = KEY_Scan(0);  
  
    if(key==KEY0_PRES)  
    {  
        //反转 sw1 开关对象的状态  
        lv_sw_toggle(sw1,LV_ANIM_ON);  
    }  
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



State:ON

图 3.4.1 开关处于打开状态的效果

然后我们可以通过点击,滑动或者按一下 KEY0 键等操作来把开关切换到关闭状态,其效果如下:



State:OFF

图 3.4.2 开关处于关闭状态的效果

图 3.4.1 弧形演示效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_btm 矩阵按钮

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

lv_btm 矩阵按钮

1. 介绍

lv_btm 矩阵按钮对象你可以看作是一系列伪按钮的集合,只不过它是以行和列有序的方式来排列这些子按钮,名称中的 m 就是 matrix(矩阵)的缩写,注意了,我这里故意加了一个伪字来说明它不是真正的 lv_btn 按钮对象,而是 lv_btm 内部纯绘制出来的具有按钮外观的图形,而且这个图形具有和 lv_btn 按钮一样的点击效果,这种伪按钮的好处是它基本不占内存消耗(一个伪按钮大概需要 8 个字节),这属于 littleVGL 的一种优化操作,所以如果你有多个按钮集合的应用场景的话,请最好使用 lv_btm 控件,而不要去使用 lv_cont 容器外加 lv_btn 按钮的方案.

lv_btm 控件可以给它内部的每个伪按钮设置文本标题以及设置相对的大小,然后它还可以通过 lv_btm_set_btn_ctrl 接口来设置其内部按钮的各种控制属性,比如是否可见,是否处于禁用状态等等,除此之外 lv_btm 具有和 lv_label 对象一样的文本重绘色功能,以及自己专有的“One toggle”特性,可以通过 lv_btm_set_one_toggle 接口来使能“One toggle”特性,使能之后,在同一时刻就只能允许最多一个按钮处于切换态了.

当 lv_btm 内部的按钮被点击或者被重复长按下时,它会给它的事件回调函数发送 LV_EVENT_VALUE_CHANGED 事件,于此同时这个按钮的 id 号会作为事件自定义数据传递过去,那我们在事件回调函数中可以通过如下代码的方式来获取按钮的 id 号:

```
uint16_t * btn_id = (uint16_t*)lv_event_get_data();
printf("当前被按下的按钮 id 为:%d", *btn_id);
```

除了这种方式外,我们还可以通过 lv_btm_get_active_btn(btm) 接口来获取按钮的 id 号,注意了,当按钮被重复长按下时,它也是发送 LV_EVENT_VALUE_CHANGED 事件,而不是发送前面章节介绍过的 LV_EVENT_LONG_PRESSED_REPEAT 事件,如果你不想让按钮具有重复长按效果的话,那我们是可以通过控制属性来设置的.

最后我们来说一下 lv_btm 对象内部按钮的排列方式,它是以从左往右,从上到下的方式来排列的,其内部的每一个按钮都具有一个唯一的 id 号,这个 id 号也是按照上面的排列方式来进行分配的,从 0 开始分配,然后依次增 1,整体上来说,lv_btm 控件有点小复杂,但是只要你掌握了,就可以实现很强大的功能,我们后面要讲到的 lv_kb 键盘控件也是利用 lv_btm 控件来实现的.

2. lv_btm 的 API 接口

2.1 主要数据类型

3. 控制属性数据类型

```
enum {
    LV_BTNM_CTRL_HIDDEN      = 0x0008,
    LV_BTNM_CTRL_NO_REPEAT   = 0x0010,
    LV_BTNM_CTRL_INACTIVE     = 0x0020,
    LV_BTNM_CTRL_TGL_ENABLE   = 0x0040,
    LV_BTNM_CTRL_TGL_STATE    = 0x0080,
    LV_BTNM_CTRL_CLICK_TRIG   = 0x0100,
};

typedef uint16_t lv_btm_ctrl_t;
```

LV_BTNM_CTRL_HIDDEN: 设置按钮为隐藏不可见状态

LV_BTNM_CTRL_NO_REPEAT: 设置按钮不具有重复长按的动作效果

LV_BTNM_CTRL_INACTIVE: 设置按钮为禁用状态

LV_BTNM_CTRL_TGL_ENABLE: 设置按钮为 Toggle 切换按钮

LV_BTNM_CTRL_TGL_STATE: 设置按钮当前的状态就是切换态

LV_BTNM_CTRL_CLICK_TRIG: 设置按钮的点击方式为松手触发,如果不设置的话,那默认就是按下时触发,二者只能选择一种

这些值之间是可以进行位或操作的

4. 矩阵按钮样式数据类型

```
enum {
    LV_BTNM_STYLE_BG,
    LV_BTNM_STYLE_BTN_REL,
    LV_BTNM_STYLE_BTN_PR,
    LV_BTNM_STYLE_BTN_TGL_REL,
    LV_BTNM_STYLE_BTN_TGL_PR,
    LV_BTNM_STYLE_BTN_INA,
};

typedef uint8_t lv_btm_style_t;
```

我们在前面已经学过了 lv_btn 按钮的样式,所以再来学习矩阵按钮的样式就非常简单了,只有 LV_BTNM_STYLE_BG 这一个样式是用来修饰 lv_btm 对象背景的,其他的 5 个样式全部是用来修饰其内部的伪按钮.

LV_BTNM_STYLE_BG: 修饰 lv_btm 对象的背景,其中 body.padding 下的 left,top, right, bottom 是用来设置按钮与 lv_btm 对象边框的各种内边距,然后 inner 字段是用来设置按钮与按钮之间的距离

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_btm_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置按钮映射表

```
void lv_btm_set_map(const lv_obj_t * btm, const char * map[]);
```

参数:

btm: 矩阵按钮对象

map: 矩阵按钮对象就是根据这个 map 映射表来确定其内部的按钮个数以及按钮标题的, map 就是一个字符串数组,除了"\n"换行元素和""空串结尾元素外,其中的每一个元素都代表着一个按钮,字符串的内容就是这个按钮的文本标题,比如我想在其内部创建 4 个按钮,分为 2 行,上面 2 个,下面 2 个,那么 map 映射表具体如下:

```
const char * const map[] = {"Btn1", "Btn2", "\n", "Btn3", "Btn4", ""};
```

其中的"\n"元素仅仅是用来换行的,没有其他的实际含义,然后 map 映射表的末尾必须要以一个""空串来作为结束标志,然后其默认样式效果如下:



图 2.2.2.1 map 映射效果

然后这里还有一点需要注意了,那就是 lv_btm 矩阵按钮对象只是在内部简单的引用了 map 映射表,没有做拷贝操作的,所以你得保证这个 map 映射表资源在外部不能被释放了

2.2.3 设置某个按钮的控制属性

```
void lv_btmn_set_btn_ctrl(const lv_obj_t * btmn, uint16_t btn_id, lv_btmn_ctrl_t ctrl);
```

参数:

btmn: 矩阵按钮对象

btn_id: 设置哪一个按钮的控制属性,由此按钮的 id 来指定,注意"\n"换行元素不是按钮,因此它是没有按钮 id 的

ctrl: 控制属性,有如下 6 个可选值:

LV_BTNM_CTRL_HIDDEN:设置按钮为隐藏不可见状态

LV_BTNM_CTRL_NO_REPEAT:设置按钮不具有重复长按的动作效果

LV_BTNM_CTRL_INACTIVE:设置按钮为禁用状态

LV_BTNM_CTRL_TGL_ENABLE:设置按钮为 Toggle 切换按钮

LV_BTNM_CTRL_TGL_STATE:设置按钮当前的状态就是切换态

LV_BTNM_CTRL_CLICK_TRIG:设置按钮的点击方式为松手触发,如果不设置的话,那默认就是按下时触发,二者只能选择一种

这些属性值之间是可以进行位或操作的

2.2.4 给所有按钮设置共同的控制属性

```
void lv_btmn_set_btn_ctrl_all(lv_obj_t * btmn, lv_btmn_ctrl_t ctrl);
```

参数:

btmn: 矩阵按钮对象

ctrl: 控制属性,和 lv_btmn_set_btn_ctrl 接口中的含义相同

2.2.5 清除某个按钮的控制属性

```
void lv_btmn_clear_btn_ctrl(const lv_obj_t * btmn, uint16_t btn_id, lv_btmn_ctrl_t ctrl);
```

参数:

btmn: 矩阵按钮对象

btn_id: 按钮 id

ctrl: 控制属性,和 lv_btmn_set_btn_ctrl 接口中的含义相同

2.2.6 清除所有按钮的控制属性

```
void lv_btmn_clear_btn_ctrl_all(lv_obj_t * btmn, lv_btmn_ctrl_t ctrl);
```

参数:

btmn: 矩阵按钮对象

ctrl: 控制属性,和 lv_btmn_set_btn_ctrl 接口中的含义相同

2.2.7 设置某个按钮的相对宽度

```
void lv_btmn_set_btn_width(const lv_obj_t * btmn, uint16_t btn_id, uint8_t width);
```

参数:

btmn: 矩阵按钮对象

btn_id: 设置矩阵按钮对象中的哪一个按钮的相对宽度,由此按钮的 id 来指定,注意"\n"换行元素不是按钮,因此它是没有按钮 id 的

width: 指定在同一行按钮中此按钮所占的宽度比例,它是一个当前行的相对大小哦,并不是指定具体的宽度数值,此 width 参数的范围为[1,7],如果不设置的话,那么默认值就是 1,此值越大,那么在同一行中,此按钮具有的宽度也就越大,举个简单的例子,比如在某一行中具有 btn1 和 btn2 两个按钮,btn1 的 width 值为 4,而 btn2 的 width 值为 2,那么最后 btn1 的在此行中的宽度是 btn2 的 4/2=2 倍,如下图所示:

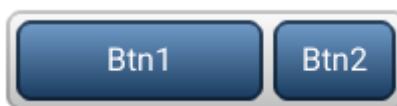


图 2.2.4.1 相对宽度效果

就以上面这个例子为参考,其实这个 Btn1 或者 Btn2 的具体大小是可以推算出来的,假如我们通过 lv_obj_set_size 接口来设置 btmn 对象的大小,宽度为 btmn_w,高度为 btmn_h,而且 btmn 对象的背景样式为 bg_style,那么

Btn1 的宽度 = (btmn_w - bg_style.body.padding.left - bg_style.body.padding.right - bg_style.body.padding.inner)*2/3;

Btn1 的高度 = btmn_h - bg_style.body.padding.top - bg_style.body.padding.bottom;

Btn2 的宽度 = (btmn_w - bg_style.body.padding.left - bg_style.body.padding.right - bg_style.body.padding.inner)*1/3;

Btn2 的高度 = btmn_h - bg_style.body.padding.top - bg_style.body.padding.bottom;

2.2.8 设置控制映射表

```
void lv_btmn_set_ctrl_map(const lv_obj_t * btmn, const lv_btmn_ctrl_t ctrl_map[]);
```

参数:

btmn: 矩阵按钮对象

ctrl_map: 控制映射表,其中的每一个元素控制其相应的按钮,在每一个元素里面,你可以设置按钮的控制属性以及还有按钮的相对宽度,如下所示:

```
ctrl_map[0] = width | LV_BTNM_CTRL_NO_REPEAT | LV_BTNM_CTRL_TGL_ENABLE;
```

这个接口会在内部对 ctrl_map 进行拷贝操作的,所以 ctrl_map 这个资源在外面被释放掉是没有关系的

2.2.9 设置某个按钮为按下状态

```
void lv_btmn_set_pressed(const lv_obj_t * btmn, uint16_t id);
```

参数:

btmn: 矩阵按钮对象

btn_id: 按钮 id

2.2.10 设置样式

```
void lv_btmn_set_style(lv_obj_t * btmn, lv_btmn_style_t type, const lv_style_t * style);
```

参数:

btmn: 矩阵按钮对象

type: 要设置那部分的样式,有如下 6 个可选值

LV_BTNM_STYLE_BG
LV_BTNM_STYLE_BTN_REL
LV_BTNM_STYLE_BTN_PR
LV_BTNM_STYLE_BTN_TGL_REL
LV_BTNM_STYLE_BTN_TGL_PR
LV_BTNM_STYLE_BTN_INA

style: 样式

2.2.11 是否使能文本重绘色功能

```
void lv_btmn_set_recolor(const lv_obj_t * btmn, bool en);
```

参数:

btmn: 矩阵按钮对象

en: 是否使能重绘色,true 是使能,false 是不使能

矩阵按钮对象的文本重绘色功能和 lv_label 标签对象的文本重绘色功能是一样的,格式也是#十六进制颜色 文本#,举个例子如下所示:

```
static const char * const btmn_map[] = {"#ff0000 Red# btn", "#00ff00 Green# btn", ""};
```

2.2.12 是否使能 One toggle 特性

```
void lv_btmn_set_one_toggle(lv_obj_t * btmn, bool one_toggle);
```

参数:

btmn: 矩阵按钮对象

one_toggle: 是否使能 One toggle 特性,true 代表使能,false 代表不使能

使能 One toggle 特性之后,在所有被设置了 LV_BTNM_CTRL_TGL_ENABLE 控制属性的按钮中,同一时刻就只能允许最多一个按钮处于切换态了,所以要想看到正确的效果,这里是有一个前提的,那就是至少得有 2 个按钮设置了 LV_BTNM_CTRL_TGL_ENABLE 控制属性,即至少得有 2 个 Toggle 按钮

2.2.13 获取当前被点击的按钮 id

```
uint16_t lv_btm_get_active_btn(const lv_obj_t * btm);
```

参数:

btm: 矩阵按钮对象

返回值:

返回当前被点击的按钮 id,如果获取失败,则返回 LV_BTNM_BTN_NONE

这里的 active 是指当前活跃的按钮,也就是被点击了的按钮,点击这两个字就包含了 2 个含义,按下时触发的或者松手时触发的,至于到底是哪一种方式触发的,取决于它有没有设置 LV_BTNM_CTRL_CLICK_TRIG 控制属性,此 API 接口一般用在事件回调函数中,用来判断当前事件是被哪一个按钮触发的

2.2.14 获取当前被点击的按钮文本标题

```
const char * lv_btm_get_active_btn_text(const lv_obj_t * btm);
```

参数:

btm: 矩阵按钮对象

返回值:

返回当前被点击的按钮文本标题,获取失败,则返回 NULL

此 API 接口一般用在事件回调函数中,也可以用来判断当前事件是被哪一个按钮触发的

2.2.15 获取当前被按下的按钮 id

```
uint16_t lv_btm_get_pressed_btn(const lv_obj_t * btm);
```

参数:

btm: 矩阵按钮对象

返回值:

返回当前被按下的按钮 id,如果获取失败,则返回 LV_BTNM_BTN_NONE

此 API 接口和 lv_btm_get_active_btn 接口功能差不多,但是此 API 接口更为具体一点,它指的是按下时,不包括松手时的情况

2.2.16 判断某个按钮是否已经设置过了某控制属性

```
bool lv_btm_get_btn_ctrl(lv_obj_t * btm, uint16_t btn_id, lv_btm_ctrl_t ctrl);
```

参数:

btm: 矩阵按钮对象

btn_id: 按钮 id

ctrl: 判断 ctrl 控制属性是否被设置过了

返回值:

返回 true 代表已经被设置过了, 返回 false 代表没有被设置过

2.2.17 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了, 比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰 btmn 的背景,主要是修改 padding 字段来控制内边距,然后创建 btmn1 和 btmn2 俩个矩阵按钮对象, btmn1 是一个类似于主菜单效果的矩阵按钮,同时 btmn1 也用来演示按钮的各种控制属性,而 btmn2 是用来演示 One Toggle 特性的,当按下 KEY0 按键时是来回切换 AUDIO 按钮的 LV_BTNM_CTRL_HIDDEN 控制属性,当按下 KEY1 键时是来回切换 VIDEO 按钮的 LV_BTNM_CTRL_INACTIVE 控制属性,当按下 KEY2 键时是来回切换 HOME 按钮的 LV_BTNM_CTRL_NO_REPEAT 控制属性,当按下 WKUP 按键时是来回切换 SAVE 按钮的 LV_BTNM_CTRL_CLICK_TRIG 控制属性.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0, KEY1, KEY2, WKUP 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_btmn_test.c 和 lv_btmn_test.h 俩个文件,其中 lv_btmn_test.c 文件的内容如下:

```
#include "lv_btmn_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_obj_t * btmn1,* btmn2;
lv_obj_t * label1;
lv_style_t bg_style;

//第二个 const 表示把此映射表放在 flash 存储区,而不是 sram,可以减少内存的开销
static const char * const btmn1_map[] = {           //btmn1 的按钮映射表
    " "LV_SYMBOL_AUDIO"\n#ff0000 AUDIO#",          //第一行放 2 个按钮
    " "LV_SYMBOL_VIDEO"\n#ff0000 VIDEO#",          //换行
    "\n",                                         //第二行放 2 个按钮
    " "LV_SYMBOL_HOME"\n#ff0000 HOME#",          //第三行放 2 个按钮
    " "LV_SYMBOL_SAVE"\n#ff0000 SAVE#",          //空字符串作为结束符
    ""};

};
```

```
static const char * const btm2_map[] = { //btm2 的按钮映射表
    "Btn1",
    "Btn2",
    "Btn3",
    "" //空字符串作为结束符
};

//事件回调函数
static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    static uint32_t cnt = 0; //记录进入 LV_EVENT_VALUE_CHANGED 事件的次数
    char buff[100];
    const char * btn_title;
    uint16_t btn_id;

    if(event==LV_EVENT_VALUE_CHANGED)
    {
        //1.计数器增 1
        cnt++;

        //2.获取当前被点击了的按钮标题
        btn_title = lv_btm_get_active_btn_text(obj);

        //3.获取当前被点击了的按钮 id
        btn_id = *((uint16_t*)lv_event_get_data()); //方式 1
        //btn_id = lv_btm_get_active_btn(obj); //方式 2

        //4.把事件信息显示在标签上
        sprintf(buff,"obj:%s,btn_id:%d,btn_title:%s,cnt:%d,",obj==btm1?"btm1":"btm2",
                btn_id,btn_title,cnt);
        lv_label_set_text(label1,buff);
    }
}

//例程入口
void lv_btm_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建一个自定义样式,用于修饰 lv_btm 的背景
    lv_style_copy(&bg_style,&lv_style_pretty);
```

```
bg_style.body.padding.top = 5;//上内边距
bg_style.body.padding.bottom = 5;//底内边距
bg_style.body.padding.left = 15;//左内边距
bg_style.body.padding.right = 15;//右内边距
bg_style.body.padding.inner = 5;//按钮与按钮之间的距离

//2.创建一个类似于主菜单效果的 btmn1
btmn1 = lv_btmn_create(scr,NULL);
lv_obj_set_size(btmn1,160,160);//设置大小
lv_obj_align(btmn1,NULL,LV_ALIGN_IN_TOP_MID,0,10);//设置与屏幕的对齐方式
lv_btmn_set_map(btmn1,(const char**)btmn1_map);//设置按钮映射表
lv_btmn_set_recolor(btmn1,true);//使能颜色重绘
lv_btmn_set_style(btmn1,LV_BTNM_STYLE_BG,&bg_style);//设置背景样式
lv_obj_set_event_cb(btmn1,event_handler);//设置事件回调函数

//3.创建 btmn2,用来演示 One Toggle 特性
btmn2 = lv_btmn_create(scr,NULL);
lv_obj_set_size(btmn2,220,50);//设置大小
//设置与 btmn1 的对齐方式
lv_obj_align(btmn2,btmn1,LV_ALIGN_OUT_BOTTOM_MID,0,10);
lv_btmn_set_map(btmn2,(const char**)btmn2_map);//设置按钮映射表
lv_btmn_set_btn_width(btmn2,1,2);//设置 Btn2 按钮的宽度是其他按钮的 2 倍

//所有按钮都设置为 Toggle 按钮,至少得有 2 个以上的 Toggle 按钮才能看到 One
//Toggle 效果
lv_btmn_set_btn_ctrl_all(btmn2,LV_BTNM_CTRL_TGL_ENABLE);
lv_btmn_set_one_toggle(btmn2,true);//使能 One Toggle 特性

//让 Btn3 默认就处于 Toggle 状态
lv_btmn_set_btn_ctrl(btmn2,2,LV_BTNM_CTRL_TGL_STATE);
lv_obj_set_event_cb(btmn2,event_handler);//设置事件回调函数

//4.创建一个 label 标签用来显示信息
label1 = lv_label_create(scr,NULL);
lv_label_set_long_mode(label1,LV_LABEL_LONG_BREAK);//设置长文本模式
lv_obj_set_width(label1,220);//设置宽度

//设置与 btmn2 的对齐方式
lv_obj_align(label1,btmn2,LV_ALIGN_OUT_BOTTOM_MID,0,10);
lv_label_set_body_draw(label1,true);//使能背景绘制
lv_label_set_recolor(label1,true);//使能文本重绘色

//设置背景
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&lv_style_plain_color);
```

```
lv_label_set_text(label1,"Event info");
}

//来回切换 btm1 中某个按钮的控制属性
static void btm1_toggle_btn_ctrl(uint16_t btn_id,lv_btm_ctrl_t ctrl)
{
    //判断此按钮是否已经设置了 ctrl 控制属性
    if(lv_btm_get_btn_ctrl(btm1,btn_id,ctrl))
        lv_btm_clear_btn_ctrl(btm1,btn_id,ctrl);//清除掉
    else
        lv_btm_set_btn_ctrl(btm1,btn_id,ctrl);//重新设置
}

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

    if(key==KEY0_PRES)
    {
        //来回切换 AUDIO 按钮的隐藏控制属性
        btm1_toggle_btn_ctrl(0,LV_BTNM_CTRL_HIDDEN);
    }else if(key==KEY1_PRES)
    {
        //来回切换 VIDEO 按钮的禁用控制属性
        btm1_toggle_btn_ctrl(1,LV_BTNM_CTRL_INACTIVE);
    }else if(key==KEY2_PRES)
    {
        //来回切换 HOME 按钮的禁用重复长按控制属性
        btm1_toggle_btn_ctrl(2,LV_BTNM_CTRL_NO_REPEAT);
    }else if(key==WKUP_PRES)
    {
        //来回切换 SAVE 按钮的松手触发控制属性,如果不设置的话,默认是按下时触发
        btm1_toggle_btn_ctrl(3,LV_BTNM_CTRL_CLICK_TRIG);
    }
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

然后我们随便点击一个按钮,比如 Btn2,会看到事件信息显示在底部的标签上,如下所示:



图 3.4.2 点击按钮之后的效果

接着我们分别按一下 KEY0,KEY1,KEY2,WKUP 按键,来设置它们相应的控制属性,效果如下:

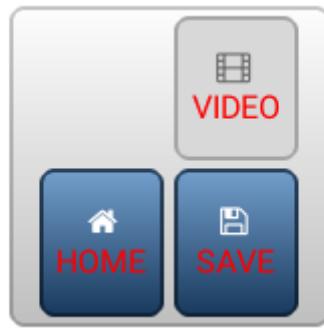


图 3.4.3 控制属性效果

此时我们可以发现 AUDIO 按钮不可见了,VIDEO 按钮处于禁用状态,不能被点击了,通过长按 HOME 按钮可以发现这个长按动作触发不了事件了,通过点击 SAVE 按钮可以发现按下时不能触发事件了,而是变成了松手时触发

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

正点原子 littleVGL 开发指南

lv_Imeter 刻度指示器

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

lv_lmter 刻度指示器

1. 介绍

lv_lmter 控件是一种类似于汽车仪表盘上的弧形刻度,它是由一些径向放射状的刻度线构成,它的主要作用就是进度指示,在所有的刻度线中,可以分为 2 种,一种是活跃刻度线,它利用样式中的 body.main_color 和 body.grad_color 属性,可以让活跃刻度线显示出弧形渐变色,活跃刻度线的作用就是展示当前的进度值,配合渐变色,可以达到更为友好更为直观的提示作用,另外一种就是非活跃刻度线了,它的颜色由 line.color 指定,一般为灰色,用来作为背景线,表示还剩下多少进度未被加载,lv_lmter 的进度范围可以通过 lv_lmter_set_range(lmeter, min, max);接口进行设置,另外其刻度线的数量和角度值可以通过 lv_lmter_set_scale(lmeter, angle, line_num)接口来进行设置.

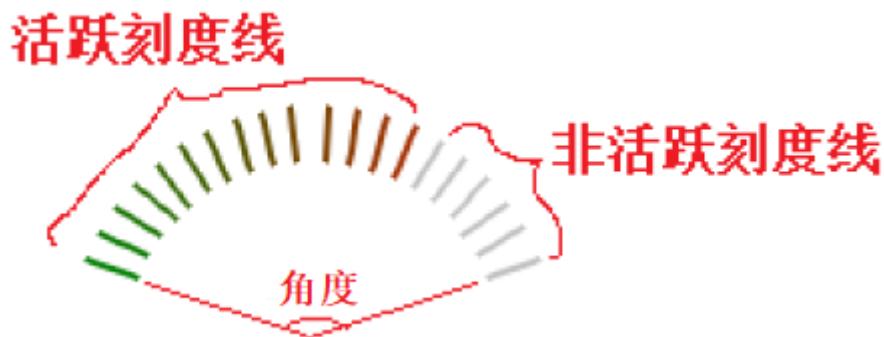


图 1.1 lv_lmter 的构成

注 1:上面的红色线以及红色文字是笔者为了给大家演示方便,故意加上去的,实际的 lv_lmter 控件是没有这些东西的

注 2:lv_lmter 控件永远是垂直对齐的

2. lv_lmter 的 API 接口

2.1 主要数据类型

3. 刻度指示器样式数据类型

```
enum {
    LV_LMETER_STYLE_MAIN,
};

typedef uint8_t lv_lmter_style_t;
```

我们这里主要是讲这种样式里面相关字段的含义,如下所示:

body.main_color: 活跃刻度线的起始颜色

body.grad_color: 活跃刻度线的终止颜色

line.color: 非活跃刻度线的颜色

line.width: 每一条刻度线的宽度

body.padding.left: 每一条刻度线的长度

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_lmter_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置进度值

```
void lv_lmter_set_value(lv_obj_t * lmter, int16_t value);
```

参数:

lmter: 刻度指示器对象

value: 进度值

2.2.3 设置进度范围

```
void lv_lmter_set_range(lv_obj_t * lmter, int16_t min, int16_t max);
```

参数:

lmter: 刻度指示器对象

min: 最小进度值

max: 最大进度值

2.2.4 设置角度和刻度线数量

```
void lv_lmter_set_scale(lv_obj_t * lmter, uint16_t angle, uint8_t line_cnt);
```

参数:

lmter: 刻度指示器对象

angle: 角度,范围为[0,360]

line_cnt: 刻度线的数量

2.2.5 设置样式

```
static inline void lv_lmter_set_style(lv_obj_t * lmter, lv_lmter_style_t type, lv_style_t * style)
```

参数:

lmter: 刻度指示器对象

type: 要设置那部分的样式,目前只有 LV_LMETER_STYLE_MAIN 这一个可选值

style: 样式

2.2.6 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰 lv_lmter 对象,然后创建一个 lmter1 刻度指示器和一个 label1 标签,其中 label1 标签是用来显示 lmter1 的当前进度值,最后再创建一个任务来模拟 lmter1 的加载过程

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_lmter_test.c 和 lv_lmter_test.h 两个文件,其中 lv_lmter_test.c 文件的内容如下:

```
#include "lv_lmter_test.h"
#include "lvgl.h"
#include <stdio.h>

lv_style_t main_style;
lv_obj_t * lmter1;
lv_obj_t * label1;

uint16_t lmter_value = 0;

//任务回调函数
void task_cb(lv_task_t *task)
{
    char buff[10];
    lmter_value += 5;
    if(lmter_value>100)
        lmter_value = 0;
    lv_lmter_set_value(lmter1,lmter_value);
    sprintf(buff,"%d%%",lmter_value);
    lv_label_set_text(label1,buff);
```

```
}

//例程入口
void lv_lmter_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建一个自定义样式
    lv_style_copy(&main_style,&lv_style_plain_color);
    main_style.body.main_color = LV_COLOR_GREEN; //活跃刻度线的起始颜色
    main_style.body.grad_color = LV_COLOR_RED; //活跃刻度线的终止颜色
    main_style.line.color = LV_COLOR_SILVER; //非活跃刻度线的颜色
    main_style.line.width = 2; //每一条刻度线的宽度
    main_style.body.padding.left = 16; //每一条刻度线的长度

    //2.创建一个刻度指示器对象
    lmter1 = lv_lmter_create(scr,NULL);
    lv_obj_set_size(lmter1,180,180); //设置大小
    lv_obj_align(lmter1,NULL,LV_ALIGN_CENTER,0,0); //与屏幕保持居中对齐
    lv_lmter_set_range(lmter1,0,100); //设置进度范围
    lv_lmter_set_value(lmter1,lmter_value); //设置当前的进度值
    lv_lmter_set_scale(lmter1,240,31); //设置角度和刻度线的数量
    lv_lmter_set_style(lmter1,LV_LMETER_STYLE_MAIN,&main_style); //设置样式

    //3.创建一个 label 标签来显示当前的进度值
    label1 = lv_label_create(scr,NULL);
    lv_obj_align(label1,lmter1,LV_ALIGN_CENTER,0,0); //设置与 lmter1 居中对齐
    //使能自动对齐功能,当文本长度发生变化时,它会自动对齐的
    lv_obj_set_auto_realign(label1,true);
    lv_label_set_text(label1,"0%"); //设置文本

    //4.创建一个任务来模拟 lmter 的加载
    lv_task_create(task_cb,500,LV_TASK_PRIO_MID,NULL);
}
```

3.4 下载验证

把代码下载进去之后,刻度指示器会自动进行加载,如下图所示:

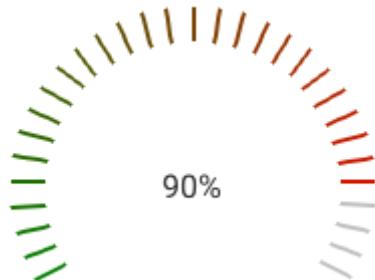


图 3.4.1 刻度指示器加载效果

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_gauge 仪表盘

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

lv_gauge 仪表盘

1. 介绍

我们前一章已经学习过了 lv_lmeter 控件了,再来学习 lv_gauge 仪表盘控件就会简单很多,因为 lv_gauge 控件就是由 lv_lmeter 控件外加一些其他附件构成的,这里的附件有 lv_label 标签,指针,中心圆点等三个元素,其中标签的个数和指针的个数都是可以通过接口来设置的,在 lv_gauge 控件中,它将 lv_lmeter 控件中的非活跃刻度线的概念给转变了一下,现在它叫关键数值点(Critical value),超过此数值点后的所有刻度线可以被样式中 line.color 指定的颜色进行高亮,那么在此关键数值点之前的所有刻度线跟 lv_lmeter 中的活跃刻度线概念差不多,可以由 body.main_color 和 body.grad_color 来形成弧形渐变色.可以通过 lv_gauge_set_critical_value(gauge, value);这个接口来设置关键数值点.

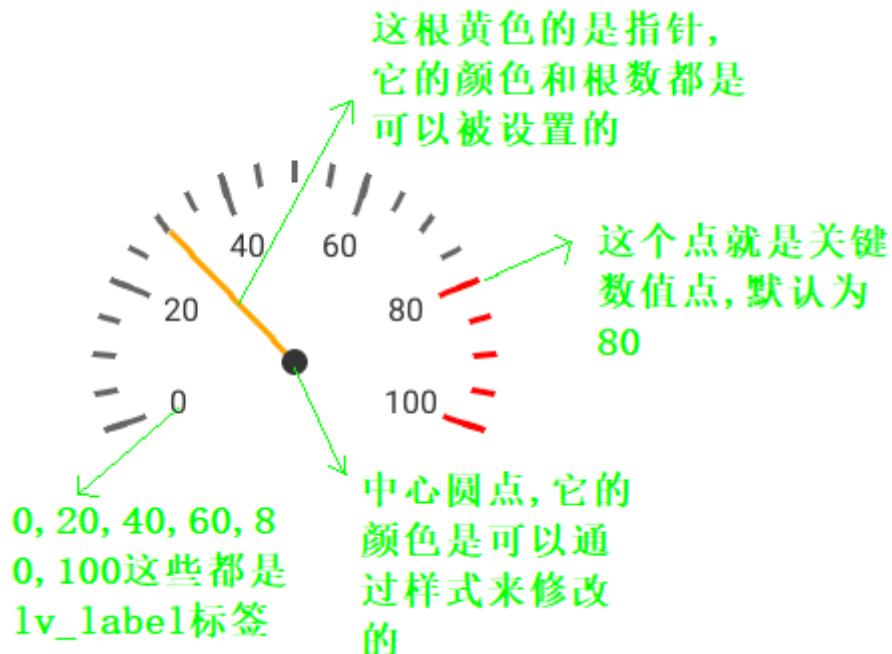


图 1.1 lv_gauge 仪表盘的构成

注:上面的绿色线以及绿色文字是笔者为了给大家演示方便,故意加上去的,实际是没有的

2. lv_gauge 的 API 接口

2.1 主要数据类型

3. 仪表盘样式数据类型

```
enum {
    LV_GAUGE_STYLE_MAIN,
};

typedef uint8_t lv_gauge_style_t;
```

我们这里主要是讲这种样式里面相关字段的含义,如下所示:

body.main_color: 关键数值点之前的刻度线的起始颜色
body.grad_color: 关键数值点之前的刻度线的终止颜色
body.padding.left: 每一条刻度线的长度
body.padding.inner: 数值标签与刻度线之间的距离
body.border.color: 中心圆点的颜色
body.radius: 中心圆点的半径
line.width: 每一条刻度线的宽度
line.color: 关键数值点之后的刻度线的颜色
text.font/color/letter_space: 数值文本的相应样式

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_gauge_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置指针的数量和颜色

```
void lv_gauge_set_needle_count(lv_obj_t * gauge, uint8_t needle_cnt, const lv_color_t colors[]);
```

参数:

gauge: 仪表盘对象

needle_cnt: 指针的数量

colors: 是一个数组,存放每一根指针的颜色,这个 colors 数组必须是静态的或者全局的,即必须得保证这个资源在外部不能被释放,因为这个接口内部没有对 colors 做拷贝操作

2.2.3 设置某根指针的数值

```
void lv_gauge_set_value(lv_obj_t * gauge, uint8_t needle_id, int16_t value);
```

参数:

gauge: 仪表盘对象

needle_id: 指针 id 号,从 0 开始的

value: 指针所指向的数值

2.2.4 设置仪表盘的数值范围

```
static inline void lv_gauge_set_range(lv_obj_t * gauge, int16_t min, int16_t max);
```

参数:

gauge: 仪表盘对象

min: 最小数值

max: 最大数值

如果不设置的话,默认范围为[0,100]

2.2.5 设置关键数值点

```
static inline void lv_gauge_set_critical_value(lv_obj_t * gauge, int16_t value);
```

参数:

gauge: 仪表盘对象

value: 关键数值点对应的数值

如果不设置的话,默认值为 80

2.2.6 设置角度,刻度数量,标签数量

```
void lv_gauge_set_scale(lv_obj_t * gauge, uint16_t angle, uint8_t line_cnt, uint8_t label_cnt);
```

参数:

gauge: 仪表盘对象

angle: 仪表盘的角度,范围为[0,360]

line_cnt: 仪表盘的刻度总数量

label_cnt: 数值标签的数量

2.2.7 设置样式

```
static inline void lv_gauge_set_style(lv_obj_t * gauge, lv_gauge_style_t type, lv_style_t * style);
```

参数:

gauge: 仪表盘对象

type: 设置那一部分的样式,目前就只有 LV_GAUGE_STYLE_MAIN 这一个可选值

style: 样式

2.2.8 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式用来修饰仪表盘,然后创建一个仪表盘控件,设置它具有 2 个指针,我们把其中的一个指针指向仪表盘控件的关键数值点,然后把另外一个指针当做速度仪表,我们另外会创建一个任务来模拟速度指针的变化,最后再创建一个标签来显示当前的速度值,而此标签会根据不同的速度值大小,显示不同的文本颜色,以起到更友好的提示作用

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_gauge_test.c 和 lv_gauge_test.h 两个文件,其中 lv_gauge_test.c 文件的内容如下:

```
#include "lv_gauge_test.h"
#include "lvgl.h"
#include <stdio.h>

lv_style_t gauge_style;
lv_obj_t * gauge1;
lv_obj_t * label1;

//在 keil 中 lv_color_t needle_colors[] = {LV_COLOR_BLUE, LV_COLOR_PURPLE};直接
//赋值报错,这是因为不接受 LV_COLOR_MAKE 形成的
//结构体赋值
lv_color_t needle_colors[2];//每一根指针的颜色,
int16_t speed_val = 0;

//任务回调函数
void task_cb(lv_task_t * task)
{
    static uint8_t is_add_dir = 1;//是否是速度增加的方向
```

```
char buff[40];

if(is_add_dir)
{
    speed_val += 5;
    if(speed_val>=100)
        is_add_dir = 0;
} else
{
    speed_val -= 5;
    if(speed_val<=0)
        is_add_dir = 1;
}
//设置指针的数值
lv_gauge_set_value(gauge1,0,speed_val);
//把此速度显示在标签上,然后根据不同大小的数值显示出不同的文本颜色
if(speed_val<60)
    sprintf(buff,"%5FB878 %d km/h#",speed_val);//显示绿色,代表安全
else if(speed_val<90)
    sprintf(buff,"%FFB800 %d km/h#",speed_val);//显示黄色,代表警告
else
    sprintf(buff,"%FF0000 %d km/h#",speed_val);//显示红色,代表危险
lv_label_set_text(label1,buff);
}
```

```
//例程入口
void lv_gauge_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1. 创建自定义样式
    lv_style_copy(&gauge_style, &lv_style_pretty_color);
    //关键数值点之前的刻度线的起始颜色,为浅绿色
    gauge_style.body.main_color = LV_COLOR_MAKE(0x5F,0xB8,0x78);
    //关键数值点之前的刻度线的终止颜色,为浅黄色
    gauge_style.body.grad_color = LV_COLOR_MAKE(0xFF,0xB8,0x00);
    gauge_style.body.padding.left = 10; //每一条刻度线的长度
    gauge_style.body.padding.inner = 8; //数值标签与刻度线之间的距离
    //中心圆点的颜色
    gauge_style.body.border.color = LV_COLOR_MAKE(0x33,0x33,0x33);
    gauge_style.line.width = 3; //刻度线的宽度
    gauge_style.text.color = LV_COLOR_BLACK; //数值标签的文本颜色
```

```
gauge_style.line.color = LV_COLOR_RED;//关键数值点之后的刻度线的颜色

//2.创建一个 gauge1 仪表盘
gauge1 = lv_gauge_create(scr, NULL);//创建仪表盘
lv_obj_set_size(gauge1,200,200);//设置仪表盘的大小
lv_gauge_set_style(gauge1,LV_GAUGE_STYLE_MAIN,&gauge_style);//设置样式
lv_gauge_set_range(gauge1,0,100);//设置仪表盘的范围
needle_colors[0] = LV_COLOR_BLUE;
needle_colors[1] = LV_COLOR_PURPLE;
//设置指针的数量和其颜色
lv_gauge_set_needle_count(gauge1,sizeof(needle_colors)/sizeof(needle_colors[0]),needle_colors);
//设置指针 1 指向的数值,我们把指针 1 当作速度指针吧
lv_gauge_set_value(gauge1,0,speed_val);
lv_gauge_set_value(gauge1,1,90);//设置指针 2 指向的数值,就让它指向关键数值点吧
lv_gauge_set_critical_value(gauge1,90);//设置关键数值点
lv_gauge_set_scale(gauge1,240,31,6);//设置角度,刻度线的数量,数值标签的数量
lv_obj_align(gauge1,NULL,LV_ALIGN_CENTER,0,0);//设置与屏幕居中对齐

//3.创建一个标签来显示指针 1 的数值
label1 = lv_label_create(scr,NULL);
lv_label_set_long_mode(label1,LV_LABEL_LONG_BREAK);//设置长文本模式
lv_obj_set_width(label1,80);//设置固定的宽度
lv_label_set_align(label1,LV_LABEL_ALIGN_CENTER);//设置文本居中对齐
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&lv_style_pretty);//设置样式
lv_label_set_body_draw(label1,true);//使能背景重绘制
lv_obj_align(label1,gauge1,LV_ALIGN_CENTER,0,60);//设置与 gauge1 的对齐方式
lv_label_set_text(label1,"0 km/h");//设置文本
lv_label_set_recolor(label1,true);//使能文本重绘色

//4.创建一个任务来模拟速度指针的变化
lv_task_create(task_cb,1000,LV_TASK_PRIO_MID,NULL);
}
```

3.4 下载验证

把代码下载进去之后,刻度指示器会自动进行加载,如下图所示:



图 3.4.1 仪表盘加载效果

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_calendar 日历

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

lv_calendar 日历

介绍

lv_calendar 日历控件是用来显示或者设置日期的,只包含年,月,日,星期等信息,是不包含时,分,秒等信息的,此控件主要由上下俩部分构成,上面的部分我们叫做 HEADER,里面有显示年,月文本信息和上下翻月份的箭头按钮,其示意图如下:



图 1.1 HEADER 的构成

对于 HEADER 中的背景颜色,文本颜色,箭头按钮按下时的颜色可以通过 LV_CALENDAR_STYLE_HEADER 和 LV_CALENDAR_STYLE_HEADER_PR 俩种样式来进行修改,然后 lv_calendar 控件的下面一部分我们把它叫做 BODY 主体,其构成稍微复杂一点,包含了星期信息行,我们把此星期信息行叫做 DAY_NAMES,其文本颜色和与 HEADER 部分之间的上边距是可以通过 LV_CALENDAR_STYLE_DAY_NAMES 样式来进行修改的,在星期信息行的下面就是我们的日期信息了,日期信息中包含了当前月份的所有天数,以及上月份的月末天数和下月份月初天数,对于 TODAY 日期(就是今天的意思)会被 LV_CALENDAR_STYLE_TODAY_BOX 样式来单独修饰,以达到高亮显示区分的作用,而对于 TODAY 日期所在的行会被 LV_CALENDAR_STYLE_WEEK_BOX 样式来单独修饰,以达到整个星期中所有天都高亮显示的效果,最后 lv_calendar 控件还支持对任意一个日期进行高亮显示,比如用来标注一些重要的日子就会显得很有作用,这个是通过 LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS 样式来修饰的,最后给出 BODY 主体的示意图:



图 1.2 BODY 主体的构成

我们可以通过 lv_calendar_set_today_date(calendar, &today_date) 接口来设置今天的日期,通过 lv_calendar_set_highlighted_dates(calendar, &highlighted_dates) 接口来设置哪些日期被高亮显示,通过 lv_calendar_set_shown_date(calendar, &shown_date) 接口来显示任意年份任意月份下的日期信息,此接口就相当于一个页跳转的功能,当然了我们也可以通过 HEADER 上的左右箭头来实现上下翻月的功能.最后给出 lv_calendar 控件的整体示意图:



图 1.3 lv_calendar 控件

1. lv_calendar 的 API 接口

2.1 主要数据类型

2. 日历数据类型

```
typedef struct
{
    uint16_t year;
    int8_t month;
    int8_t day;
} lv_calendar_date_t;
```

这个就是一个结构体,包含了年,月,日三个字段

2.1.2 日历样式数据类型

```
enum {
    LV_CALENDAR_STYLE_BG,
    LV_CALENDAR_STYLE_HEADER,
    LV_CALENDAR_STYLE_HEADER_PR,
    LV_CALENDAR_STYLE_DAY_NAMES,
    LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS,
    LV_CALENDAR_STYLE_INACTIVE_DAYS,
    LV_CALENDAR_STYLE_WEEK_BOX,
    LV_CALENDAR_STYLE_TODAY_BOX,
};

typedef uint8_t lv_calendar_style_t;
```

这个日历控件用到的样式高达 8 种,稍微有点多哈,别急,我们下面来一一介绍

LV_CALENDAR_STYLE_BG: 用来修饰日历控件主体部分的背景,使用样式中的 body 字段来修饰背景,使用 text 字段来修饰未被高亮显示的日期的文本,使用 body.padding.left/right/bottom 字段来设置日期与日历控件边框之间的距离

LV_CALENDAR_STYLE_HEADER: 用来修饰 HEADER 部分的,比如背景,文本颜色等等

LV_CALENDAR_STYLE_HEADER_PR: 设置 HEADER 中的左右箭头按钮被按下时的文本样式

LV_CALENDAR_STYLE_DAY_NAMES: 设置星期信息行的样式,用 text 字段来设置文本样式,用 body.padding.top 来设置星期信息行与 HEADER 之间的距离

LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS: 用来修饰被 lv_calendar_set_highlighted_dates 接口选择高亮了的日期,只用 text 字段来修饰文本样式

LV_CALENDAR_STYLE_INACTIVE_DAYS: 用来修饰上一个月的月末日期和下一个月的月初日期,只用 text 字段来修饰其文本样式

LV_CALENDAR_STYLE_WEEK_BOX: 用来修饰 TODAY 日期所在的行

LV_CALENDAR_STYLE_TODAY_BOX: 用来修饰 TODAY 日期

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_calendar_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置 TODAY 日期

```
void lv_calendar_set_today_date(lv_obj_t * calendar, lv_calendar_date_t * today);
```

参数:

calendar: 日历对象

today: TODAY 日期

2.2.3 跳转到某年某月份下的日期界面

```
void lv_calendar_set_showed_date(lv_obj_t * calendar, lv_calendar_date_t * showed);
```

参数:

calendar: 日历对象

showed: 要跳转到哪个年份,哪个月份下的日期界面中去,其中 showed.day 字段不起作用,无任何含义

2.2.4 设置哪一些日期被高亮显示

```
void lv_calendar_set_highlighted_dates(lv_obj_t * calendar, lv_calendar_date_t *
```

```
highlighted, uint16_t date_num);
```

参数:

calendar: 日历对象

highlighted: 是一个数组,存放着需要被高亮显示的日期,得保证此数组是静态的或者全局的

date_num: 需要被高亮显示的日期个数

2.2.5 设置星期信息行的标题

```
void lv_calendar_set_day_names(lv_obj_t * calendar, const char ** day_names);
```

参数:

calendar: 日历对象

day_names: 是一个字符串数组,用来存放星期标题,如下所示:

```
const char * const day_names[7] = {"周日","周一","周二","周三","周四","周五","周六"};
```

同时请保证此数组是静态的或者全局的

2.2.6 设置月份的标题

```
void lv_calendar_set_month_names(lv_obj_t * calendar, const char ** month_names);
```

参数:

calendar: 日历对象

month_names: 是一个字符串数组,用来存放月份的标题,如下所示:
`const char * const month_names[12] = {"一月","二月","三月","四月",...,"十二月"};`同时请保证此数组是静态的或者全局的

2.2.7 设置样式

```
void lv_calendar_set_style(lv_obj_t * calendar, lv_calendar_style_t type, const lv_style_t * style);
```

参数:

calendar: 日历对象

type: 设置哪一部分的样式,有如下 8 个可选值:

```
LV_CALENDAR_STYLE_BG,  
LV_CALENDAR_STYLE_HEADER,  
LV_CALENDAR_STYLE_HEADER_PR,  
LV_CALENDAR_STYLE_DAY_NAMES,  
LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS,  
LV_CALENDAR_STYLE_INACTIVE_DAYS,  
LV_CALENDAR_STYLE_WEEK_BOX,  
LV_CALENDAR_STYLE_TODAY_BOX
```

style: 样式

2.2.8 获取被手指按到的日期

```
lv_calendar_date_t * lv_calendar_get_pressed_date(const lv_obj_t * calendar);
```

参数:

calendar: 日历对象

返回值:

返回被手指按到的日期,如果没有日期被手指按到,则返回 NULL

此 API 接口一般放到事件回调函数中进行调用,用来实现用户选择日期的功能

2.2.9 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建 8 种样式来修饰 lv_calendar 日历控件,然后我们接着创建一个日历控件,设置它的 TODAY 日期,然后设置它具有 2 个高亮显示的日期,还设置了它的星期信息行的标题和月份的标题,最后再来创建一个标签对象,在事件回调函数中,用标签对象把用户选择的日期给显示出来

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_calendar_test.c 和 lv_calendar_test.h 俩个文件,其中 lv_calendar_test.c 文件的内容如下:

```
#include "lv_calendar_test.h"
#include "lvgl.h"
#include <stdio.h>

//周日,周一,周二,...,周六
const char * const day_names[7] = {"Su","Mo","Tu","We","Th","Fr","Sa"};

//一月,二月,三月...,十二月
const char * const month_names[12] = {"January","February","March","April","May",
"June","July","August","September","October","November","December"};

//需要被高亮显示的日期
const lv_calendar_date_t highlighted_days[2] = {{2018,11,9},{2018,11,13}};

//样式
lv_style_t bg_style;
lv_style_t header_style;
lv_style_t header_pr_style;
lv_style_t day_names_style;
```

```
lv_style_t highlighted_style;
lv_style_t inactive_style;
lv_style_t today_style;
lv_style_t today_row_style;
//标签
lv_obj_t * label1;

//事件回调函数
static void event_handler(lv_obj_t * obj, lv_event_t event)
{
    char buff[30];
    //相当于松手事件,所以也可以替换成 LV_EVENT_RELEASED
    if(event==LV_EVENT_CLICKED)
    {
        //获取被按下的日期,或者说是用户选择的日期
        lv_calendar_date_t * selected_date = lv_calendar_get_pressed_date(obj);

        //当用户点击 HEADER 区域和星期信息行区域时,会出现
        //year=0,month=0,day=0 的情况,所以我们得过滤掉这种情况
        if(selected_date&&(selected_date->year!=0)) {

            sprintf(buff,"%d/%d/%d",selected_date->year,selected_date->month,selected_date->day);
            lv_label_set_text(label1,buff);//把用户选择的日期给显示出来

            //将用户选择的日期设置为今天的日期,即 TODAY 日期
            lv_calendar_set_today_date(obj,selected_date);

            //跳转到这个日期所在的界面
            lv_calendar_set_showed_date(obj,selected_date);
        }
    }

//例程入口
void lv_calendar_test_start()
{
    lv_obj_t * scr = lv_scr_act();//获取当前活跃的屏幕对象

    //1.创建 8 种样式
    //1.1 创建背景样式
    lv_style_copy(&bg_style,&lv_style_plain_color);
```

```
bg_style.body.main_color = LV_COLOR_WHITE;//背景颜色  
bg_style.body.grad_color = LV_COLOR_WHITE;  
bg_style.body.border.color = LV_COLOR_MAKE(57,61,73);//边框的颜色  
bg_style.body.border.width = 2;//边框的宽度  
bg_style.body.shadow.color = LV_COLOR_GRAY;//阴影的颜色  
bg_style.body.shadow.width = 6;//阴影的宽度  
bg_style.body.padding.left = 0;//设置日期与左边框的距离  
bg_style.text.color = LV_COLOR_BLACK;//未被高亮显示的日期文本颜色
```

//1.2 创建 HEADER 样式

```
lv_style_copy(&header_style,&lv_style_plain_color);  
header_style.body.main_color = LV_COLOR_MAKE(57,61,73);//背景颜色  
header_style.body.grad_color = LV_COLOR_MAKE(57,61,73);  
header_style.body.padding.left = 15;//设置左箭头按钮与左边框的距离  
header_style.body.padding.right = 15;//设置右箭头按钮与右边框的距离  
header_style.body.padding.top = 8;//设置文本内容与上边框的距离
```

//1.3 创建 HEADER 按下时的样式

```
lv_style_copy(&header_pr_style,&lv_style_plain_color);  
header_pr_style.text.color = LV_COLOR_GRAY;//箭头按钮按下时的文本颜色
```

//1.4 创建星期信息行的样式

```
lv_style_copy(&day_names_style,&lv_style_plain_color);  
day_names_style.text.color = LV_COLOR_MAKE(0,150,136);
```

//1.5 创建高亮日期的样式

```
lv_style_copy(&highlighted_style,&lv_style_plain_color);  
highlighted_style.text.color = LV_COLOR_RED;//用红色进行高亮
```

//1.6 创建上个月的月末日期和下个月的月初日期样式

```
lv_style_copy(&inactive_style,&lv_style_plain_color);  
inactive_style.text.color = LV_COLOR_MAKE(0xAA,0xAA,0xAA); //灰色
```

//1.7 创建 TODAY 日期的样式

```
lv_style_copy(&today_style,&lv_style_plain_color);  
today_style.body.main_color = LV_COLOR_MAKE(85,150,216);  
today_style.body.grad_color = LV_COLOR_MAKE(85,150,216);  
today_style.body.radius = LV_RADIUS_CIRCLE;//圆角  
today_style.text.color = LV_COLOR_WHITE;
```

//1.8 创建 TODAY 日期所在行的日期

```
lv_style_copy(&today_row_style,&lv_style_transp);  
today_row_style.text.color = LV_COLOR_MAKE(85,150,216);
```

```
//2.创建日历对象
lv_obj_t * calendar1 = lv_calendar_create(scr,NULL); //创建日历对象
lv_obj_set_size(calendar1,220,220); //设置大小
lv_obj_align(calendar1,NULL,LV_ALIGN_CENTER,0,0); //设置与屏幕居中对齐
lv_obj_set_event_cb(calendar1,event_handler); //设置事件回调函数
lv_calendar_date_t today = {2018,11,23}; //可以定义为局部的
lv_calendar_set_today_date(calendar1,&today); //设置 TODAY 日期
lv_calendar_set_showed_date(calendar1,&today); //跳转到 TODAY 日期所在的界面
//设置星期信息行的标题,也可以不设置,那么 lv_calendar 会有一个默认的值
lv_calendar_set_day_names(calendar1,(const char **)day_names);
//设置月份的标题,也可以不设置,那么 lv_calendar 会有一个默认的值
lv_calendar_set_month_names(calendar1,(const char **)month_names);
//设置需要被高亮显示的日期
lv_calendar_set_highlighted_dates(calendar1,(lv_calendar_date_t *)highlihted_days,sizeof(highlihted_days)/sizeof(highlihted_days[0]));
//设置日历控件主体部分的背景样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_BG,&bg_style);
//设置 HEADER 样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_HEADER,&header_style);
//设置 HEADER 按下时的样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_HEADER_PR,&header_pr_style);
//设置星期信息行的样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_DAY_NAMES,&day_names_style);
//设置高亮日期的样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS,&highlighted_style);
//设置高亮日期的样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_INACTIVE_DAYS,&inactive_style);
//设置 TODAY 日期的样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_TODAY_BOX,&today_style);
//设置 TODAY 日期所在行的样式
lv_calendar_set_style(calendar1,LV_CALENDAR_STYLE_WEEK_BOX,&today_row_style);

//3.创建标签,用来显示用户选择的日期
label1 = lv_label_create(scr,NULL);
lv_label_set_text(label1,"year/month/day");
lv_obj_align(label1,calendar1,LV_ALIGN_OUT_BOTTOM_MID,0,15);
lv_obj_set_auto_realign(label1,true); //使能自动重新对齐功能
}
```

3.4 下载验证

把代码下载进去之后,我们可以随便选择一个日期,然后就能看到如下所示的界面效果:



图 3.4.1 日历控件演示效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_mbox 消息对话框

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

lv_mbox 消息对话框

1. 介绍

lv_mbox 控件是一个消息对话框,跟我们平常所见到的消息对话框有一点不同,那就是它没有专门的标题部分,不过我们可以通过往消息内容中加入\n 换行符来间接的实现标题部分,其实 lv_mbox 控件是由 lv_cont 容器,lv_label 标签,lv_btmn 矩阵按钮三个控件构成的,我们之前已经学习过了这三个控件的使用,所以再来学习 lv_mbox 就比较简单了,其中 lv_cont 容器是用来充当 lv_mbox 控件的背景,lv_label 标签是用来显示 lv_mbox 的消息内容,而 lv_btmn 矩阵按钮是用来充当 lv_mbox 的底部按钮栏,具体示意图如下所示:



图 1.1 lv_mbox 控件的构成

因为我们的 lv_mbox 内部包含了 lv_cont 容器,而容器是具有大小自适应特性的,所以我们的 lv_mbox 控件在垂直方向上表现出了 LV_FIT_TIGHT 的自适应方式,即其宽度可以设置成固定值,而其高度会随着内部内容的大小而变化,而且其内部的 lv_label 标签固定为 LV_LABEL_LONG_MODE_BREAK 长文本模式,我们可以通过 lv_mbox_set_text(mbox, "Message") 接口来设置 lv_mbox 的消息内容,通过 lv_mbox_add_btns(mbox, btn_str) 接口来设置其底部按钮栏,当其中的按钮被点击时,它会跟 lv_btmn 矩阵按钮一样给它的事件回调函数发送 LV_EVENT_VALUE_CHANGED 事件,被点击按钮的 id 号会被当成事件自定义参数给传递过去,当 lv_mbox 控件被创建出来之后,默认就是处于打开状态的,我们可以通过 lv_mbox_start_auto_close(mbox, delay) 接口在指定的 delay 时间之后自动关闭此消息对话框,而且在关闭时会有一种动画效果,至于动画时间的长短我们是可以通过 lv_mbox_set_anim_time(mbox, anim_time) 接口来设置的.

2. lv_mbox 的 API 接口

2.1 主要数据类型

2.1.1 消息对话框样式数据类型

```
enum {
    LV_MBOX_STYLE_BG,
    LV_MBOX_STYLE_BTN_BG,
    LV_MBOX_STYLE_BTN_REL,
    LV_MBOX_STYLE_BTN_PR,
    LV_MBOX_STYLE_BTN_TGL_REL,
    LV_MBOX_STYLE_BTN_TGL_PR,
    LV_MBOX_STYLE_BTN_INA,
};

typedef uint8_t lv_mbox_style_t;
```

别看它具有 7 种样式,最后面的 6 种样式是用来修饰其内部的 lv_btm 矩阵按钮的,使用方法和 lv_btm 矩阵按钮章节中的内容是一样的,这里不过多介绍了.

LV_MBOX_STYLE_BG: 修饰其背景和消息内容的,即修饰其内部的 lv_cont 容器和 lv_label 标签的,使用样式中的 body 字段来修饰背景,使用样式中的 text 字段来修饰消息内容,默认值为 lv_style_pretty

LV_MBOX_STYLE_BTN_BG: 修饰矩阵按钮的背景,默认值为 lv_style_trans

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_mbox_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置底部按钮栏

```
void lv_mbox_add_btns(lv_obj_t * mbox, const char ** btn_mapaction);
```

参数:

mbox: 消息框对象

btn_mapaction: 按钮映射表,必须以”\n”空字符串结尾,如下所示:

```
const char * const btn_mapaction [] = {"ok", "\n", "close", ""};
```

另外还得保证此 btn_mapaction 是静态的或者全局的

此接口的使用方法和矩阵按钮中的 lv_btm_set_map(const lv_obj_t * btm, const char * map[])接口的使用方法是一样的

2.2.3 设置消息内容

```
void lv_mbox_set_text(lv_obj_t * mbox, const char * txt);
```

参数:

mbox: 消息框对象

txt: 消息内容

我们前面说过此消息对话框是没有专门的标题部分,但是我们可以通过此接口和\n 换行符来间接的实现标题部分,示意代码如下:

```
//我们自己定义一个接口,来设置其内部的消息内容标签是否使能文本重绘色功能  
//因为 littleVGL 没有提供这样的接口,所以我们得自己来实现
```

```
void mbox_set_msg_recolor(lv_obj_t * mbox,bool en)  
{  
    lv_mbox_ext_t * ext = lv_obj_get_ext_attr(mbox);//获取控件的扩展字段  
    lv_label_set_recolor(ext->text,en);//ext->text 就是消息对话框内部的标签对象  
}
```

```
mbox_set_msg_recolor(mbox1,true);//使能消息内容的重绘色功能,这样标题就可以加色
```

```
//设置消息内容,附带设置标题
```

```
lv_mbox_set_text(mbox1," #FF0000 Title#\nThis is a message" );
```

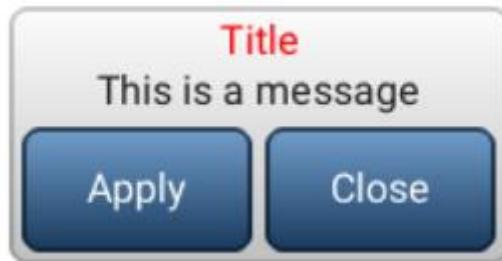


图 2.2.3.1 标题效果

2.2.4 设置动画时长

```
void lv_mbox_set_anim_time(lv_obj_t * mbox, uint16_t anim_time);
```

参数:

mbox: 消息框对象

anim_time: 动画时长,单位为 ms,当设置为 0 时代表无动画效果

这个是设置消息对话框被关闭时所做动画的时长,经笔者测试,发现此 API 接口好像不起作用

2.2.5 设置消息对话框自动关闭

```
void lv_mbox_start_auto_close(lv_obj_t * mbox, uint16_t delay);
```

参数:

mbox: 消息框对象

delay: 定时多少时间之后自动关闭此消息对话框,单位为 ms,当设置为 0 时代表立即关闭

当此消息对话框被关闭时,其所占的资源也会被全部删除,相当于执行了 lv_obj_del(mbox) 操作

2.2.6 取取消消息对话框的自动关闭

```
void lv_mbox_stop_auto_close(lv_obj_t * mbox);
```

参数:

mbox: 消息框对象

在消息对话框还未被自动关闭之前,你可以通过此接口来取消自动关闭功能

2.2.7 设置样式

```
void lv_mbox_set_style(lv_obj_t * mbox, lv_mbox_style_t type, const lv_style_t * style);
```

参数:

mbox: 消息框对象

type: 设置哪一部分的样式,有如下 7 个可选值:

LV_MBOX_STYLE_BG: 用来修饰背景和消息内容的

LV_MBOX_STYLE_BTN_BG //下面这 6 个都是用来修饰其内部矩阵按钮的

LV_MBOX_STYLE_BTN_REL,

LV_MBOX_STYLE_BTN_PR,

LV_MBOX_STYLE_BTN_TGL_REL,

LV_MBOX_STYLE_BTN_TGL_PR,

LV_MBOX_STYLE_BTN_INA,

style: 样式

2.2.8 是否使能底部按钮栏的文本重绘色

```
void lv_mbox_set_recolor(lv_obj_t * mbox, bool en);
```

参数:

mbox: 消息框对象

en: 是否使能

这其实就是在设置其内部的矩阵按钮是否使能文本重绘色功能,说的再直白点,就是底部按钮上的文本内容是否支持文本重绘色,注意此 API 接口必须得放在 lv_mbox_add_btns 接口的后面进行调用

2.2.9 获取当前被点击的按钮 id

```
uint16_t lv_mbox_get_active_btn(lv_obj_t * mbox);
```

参数:

mbox: 消息框对象

返回值:

返回当前被点击的按钮 id,如果获取失败,则返回 LV_BTNM_BTN_NONE

此接口的用法和矩阵按钮中的 lv_btm_get_active_btn 接口的用法是一样,一般放在事件回调函数中进行调用

2.2.10 获取内部的矩阵按钮对象

```
lv_obj_t * lv_mbox_get_btm(lv_obj_t * mbox);
```

参数:

mbox: 消息框对象

返回值:

返回其内部的矩阵按钮对象

拿到此矩阵按钮对象之后,我们就可以通过矩阵按钮对象自己专有的 API 接口来设置其特性了,注意此 API 接口必须得放在 lv_mbox_add_btns 接口的后面进行调用,否则会返回 NULL

2.2.11 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建 4 种样式来修饰消息对话框,然后接着创建一个按钮,此按钮的作用就是用来再次打开消息对话框的,最后接着创建一个消息对话框,为其设置了消息内容,在消息内容中附带实现了标题的功能,然后为其设置了 Apply 和 Close 俩个按钮,当点击 Apply 按钮时,会马上把此消息对话框给关闭掉,当点击 Close 按钮时,会定时 2 秒后把此消息对话框给关闭掉.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_mbox_test.c 和 lv_mbox_test.h 俩个文件,其中 lv_mbox_test.c 文件的内容如下:

```
#include "lv_mbox_test.h"
#include "lvgl.h"

lv_style_t bg_style;
lv_style_t btm_bg_style;
lv_style_t btn_rel_style;
lv_style_t btn_pr_style;
lv_obj_t * open_btn;
lv_obj_t * mbox1;
const char * const btns_map[] = {"#5FB878 Apply#", "\n", "#ff0000 Close#", ""};

lv_obj_t * mbox_create(lv_obj_t * parent); // 函数申明
```

```
//事件回调函数
void event_handler(lv_obj_t * obj, lv_event_t event)
{
```

```
uint16_t btn_id;
if(obj==open_btn)//是打开按钮
{
    if(event==LV_EVENT_RELEASED)
    {
        //重新在创建一个消息对话框
        mbox1 = mbox_create(lv_scr_act());
    }
} else if(obj==mbox1)//是消息对话框
{
    if(event==LV_EVENT_VALUE_CHANGED)
    {
        //获取按钮 id
        //也可以使用 btn_id = *((uint16_t*)lv_event_get_data())的方式
        btn_id = lv_mbox_get_active_btn(obj);

        if(btn_id==0)//Apply 按钮
        {
            //马上关闭消息对话框
            //也可以使用 lv_obj_del(obj);只不过 lv_obj_del 是立即关闭,而且无关
            //闭动画效果
            lv_mbox_start_auto_close(obj,0);
        } else if(btn_id==1)//Close 按钮
        {
            //定时关闭消息对话框
            lv_mbox_start_auto_close(obj,2000);//定时 2 秒
        }
    }
}

//我们自己定义一个接口,来设置其内部的消息内容标签是否使能文本重绘色功能
//因为 littleVGL 没有提供这样的接口,所以我们得自己来实现
void mbox_set_msg_recolor(lv_obj_t * mbox,bool en)
{
    lv_mbox_ext_t * ext = lv_obj_get_ext_attr(mbox);//获取控件的扩展字段
    lv_label_set_recolor(ext->text,en);//ext->text 就是消息对话框内部的标签对象
}

//创建自己封装之后的消息对话框
//parent:父对象
//返回值: 返回创建出来的消息对话框对象
lv_obj_t * mbox_create(lv_obj_t * parent)
{
```

```
#define MBOX_WIDTH          220 //消息对话框的宽度
#define MBOX_BTN_HEIGHT      30  //其内部每个按钮的高度

//其内部含有多少个按钮,我们这里只有 Apply 和 Close 俩个按钮
#define MBOX_BTN_NUM         2

lv_obj_t * mbox = lv_mbox_create(parent,NULL); //创建消息对话框

//使能消息内容的文本重绘色,这样就可以对标题进行加色区分了
mbox_set_msg_recolor(mbox,true);

//设置消息内容,附带设置标题,同时对标题重绘色
lv_mbox_set_text(mbox,"#007AFF Title#\nThis is a message");
lv_mbox_add_btns(mbox,(const char**)btms_map); //设置按钮映射表
lv_mbox_set_recolor(mbox,true); //使能其内部按钮的文本重绘色功能
lv_obj_set_width(mbox,MBOX_WIDTH); //设置固定的宽度,高度会自适应的
lv_obj_align(mbox,NULL,LV_ALIGN_CENTER,0,0); //设置与父对象居中对齐
lv_obj_set_event_cb(mbox,event_handler); //设置事件回调函数

//设置消息对话框的背景样式
lv_mbox_set_style(mbox,LV_MBOX_STYLE_BG,&bg_style);

//设置其内部矩阵按钮的背景样式
lv_mbox_set_style(mbox,LV_MBOX_STYLE_BTN_BG,&btm_bg_style);

//设置按钮释放状态的样式
lv_mbox_set_style(mbox,LV_MBOX_STYLE_BTN_REL,&btm_rel_style);

//设置按钮按下状态的样式
lv_mbox_set_style(mbox,LV_MBOX_STYLE_BTN_PR,&btm_pr_style);

//最好在消息对话框全部初始化完成之后,再来设置其内部矩阵按钮的各种特性,否则有可能得不到预期的效果
lv_obj_t * btm_of_mbox = lv_mbox_get_btm(mbox); //获取其内部的矩阵按钮对象

//设置矩阵按钮的大小
lv_obj_set_size(btm_of_mbox,MBOX_WIDTH,MBOX_BTN_HEIGHT*MBOX_BTN_NUM);

return mbox;
}

//例程入口
void lv_mbox_test_start()
```

```
{  
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象  
  
    //1. 创建 4 种样式  
    //1.1 创建消息对话框的背景样式  
    lv_style_copy(&bg_style, &lv_style_plain_color);  
    bg_style.body.main_color = LV_COLOR_MAKE(250, 250, 250); //背景颜色  
    bg_style.body.grad_color = bg_style.body.main_color;  
    bg_style.body.radius = 10; //圆角半径  
    bg_style.body.border.width = 1; //边框宽度  
    bg_style.body.border.color = LV_COLOR_MAKE(150, 150, 150); //边框颜色  
    bg_style.body.shadow.color = bg_style.body.border.color; //阴影颜色  
    bg_style.body.shadow.width = 6; //阴影的宽度  
  
    //设置其内部的消息内容与消息对话框上边框之间的距离  
    bg_style.body.padding.top = 10;  
  
    //设置其内部的矩阵按钮与消息对话框底边框之间的距离  
    bg_style.body.padding.bottom = 0;  
    bg_style.body.padding.inner = 10; //设置消息内容与矩阵按钮之间的距离  
    bg_style.text.color = LV_COLOR_BLACK; //消息内容的文本颜色  
  
    //1.2 创建矩阵按钮的背景样式  
    lv_style_copy(&btm_bg_style, &lv_style_transp_tight);  
  
    //设置各种内边距全部为 0, 其实 lv_style_transp_tight 样式默认就是各种内边距全为 0 的  
    btm_bg_style.body.padding.top = 0;  
    btm_bg_style.body.padding.left = 0;  
    btm_bg_style.body.padding.right = 0;  
    btm_bg_style.body.padding.bottom = 0;  
    btm_bg_style.body.padding.inner = 0;  
  
    //1.3 创建按钮释放状态的样式  
    lv_style_copy(&btn_rel_style, &lv_style_transp);  
    btn_rel_style.body.border.part = LV_BORDER_TOP; //只绘制上边框  
    btn_rel_style.body.border.width = 1; //边框的宽度  
    btn_rel_style.body.border.color = bg_style.body.border.color; //边框的颜色  
  
    //1.4 创建按钮按下状态的样式  
    lv_style_copy(&btn_pr_style, &btn_rel_style);  
    btn_pr_style.body.opa = LV_OPA_COVER; //完全不透明  
    btn_pr_style.body.border.part = LV_BORDER_FULL; //绘制四边  
    btn_pr_style.body.main_color = LV_COLOR_MAKE(200, 200, 200); //背景颜色  
    btn_pr_style.body.grad_color = btn_pr_style.body.main_color;
```

```
//2.创建一个按钮来打开消息对话框
open_btn = lv_btn_create(scr,NULL);
lv_obj_set_size(open_btn,150,60);
lv_obj_set_event_cb(open_btn,event_handler);//设置按钮的事件回调函数
lv_obj_t * label1 = lv_label_create(open_btn,NULL);
lv_label_set_text(label1,"Open mbox");
lv_obj_align(open_btn,NULL,LV_ALIGN_CENTER,0,0);//与屏幕居中对齐

//3.创建封装之后的消息对话框
mbox1 = mbox_create(scr);

}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

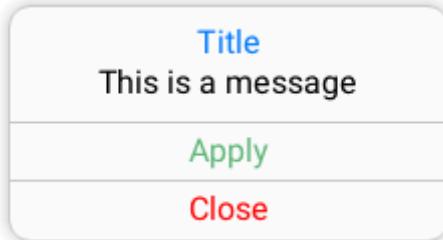


图 3.4.1 初始界面效果

然后我们可以通过点击 Apply 或者 Close 按钮来把此消息对话框给关闭掉,其中 Apply 是马上关闭,而 Close 是定时 2 秒后关闭,消息对话框被关闭时会有动画效果.

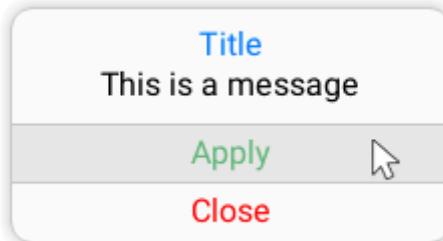


图 3.4.2 按钮点击效果

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

正点原子 littleVGL 开发指南

lv_page 页面

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

lv_page 页面

1. 介绍

lv_page 页面控件是由两个 lv_cont 容器控件构成的,其中一个容器作为 lv_page 页面控件的背景层,另外一个容器作为 lv_page 页面控件的载体,此载体可以用来存放其他任何类型的子对象,算是发挥了它容器自身的本色,如果此载体的大小超过了 lv_page 控件自身的大小,那么此载体就可以在水平或者垂直方向上进行滚动了,通过滚动操作,就能实现在有限的可视区域内跟更多的子对象进行交互操作了,为了方便后面理解,我们把这个容器或者此载体简称为 **scrl**,其实就是 Scroll 单词的缩写。

当进行滚动操作时,lv_page 页面控件上可以出现可选的滚动条进行示意,根据不同的应用场景,此滚动条总共有 6 种模式,我们可以通过 lv_page_set_sb_mode(page, SB_MODE) 接口来设置滚动条的模式,如果我们想要让 lv_page 页面控件中的某一个子对象处于可见状态时,除了用户滚动操作可以实现外,我们还可以调用 lv_page_focus(page, child, LV_ANIM_ON/OFF) 接口来使此 child 子对象获得聚焦,接着 lv_page 页面控件会自动将此 child 子对象滚动到可见状态,在这个过程中,我们可以选择是否具有动画效果,动画时长可以通过 lv_page_set_anim_time(page, anim_time) 接口来进行设置,除此之外,我们还可以通过 lv_page_scroll_hor(page, dist) 和 lv_page_scroll_ver(page, dist) 接口来更为具体的控制页面的滚动,一个是控制水平方向的,另外一个是控制垂直方向的。

最后我们来说一下 lv_page 页面控件的边缘半圆弧动画效果,当我们通过 lv_page_set_edge_flash(page, en) 接口使能此效果之后,如果用户将页面滚动到了某边缘时,那么此边缘上就会出现一个半圆弧动画效果,主要是提醒用户你已经滑到边缘了,还算是一个比较人性化,炫酷的小功能。



图 1.1 lv_page 的构成

2. lv_page 的 API 接口

2.1 主要数据类型

2.1.1 滚动条模式数据类型

```
enum {
    LV_SB_MODE_OFF      = 0x0,
    LV_SB_MODE_ON       = 0x1,
    LV_SB_MODE_DRAG     = 0x2,
    LV_SB_MODE_AUTO     = 0x3,
    LV_SB_MODE_HIDE      = 0x4,
    LV_SB_MODE_UNHIDE   = 0x5,
};

typedef uint8_t lv_sb_mode_t;
```

我们前面说过了 lv_page 的滚动条具有 6 种模式,但主要的就是前面 4 种,最后面的 2 种没啥用处.

LV_SB_MODE_OFF: 不显示任何滚动条

LV_SB_MODE_ON: 不论如何,都要把垂直和水平滚动条显示出来

LV_SB_MODE_DRAG: 当滚动页面时才显示出相应的滚动条,否则不显示

LV_SB_MODE_AUTO: 自动模式,当载体容器的大小超过 lv_page 页面的大小时,就会显示出相应的滚动条

LV_SB_MODE_HIDE: 把所有的滚动条都给隐藏了,跟 LV_SB_MODE_OFF 的作用差不多

LV_SB_MODE_UNHIDE: 把隐藏了的滚动条再次显示出来

2.1.2 页面边缘数据类型

```
enum {
    LV_PAGE_EDGE_LEFT  = 0x1,
    LV_PAGE_EDGE_TOP   = 0x2,
    LV_PAGE_EDGE_RIGHT = 0x4,
    LV_PAGE_EDGE_BOTTOM = 0x8
};

typedef uint8_t lv_calendar_style_t;
```

此数据类型主要是在 bool lv_page_on_edge(lv_obj_t * page, lv_page_edge_t edge); 接口中被使用到了,用来判断页面是否到达了某边缘

2.1.3 页面样式数据类型

```
enum {
    LV_PAGE_STYLE_BG,
    LV_PAGE_STYLE_SCRL,
    LV_PAGE_STYLE_SB,
    LV_PAGE_STYLE_EDGE_FLASH,
};

typedef uint8_t lv_page_style_t;
```

LV_PAGE_STYLE_BG: 用来修饰页面背景的,使用样式中的 body 字段,默认值为 lv_style_pretty_color

LV_PAGE_STYLE_SCRL: 用来修饰载体容器的,使用样式中的 body 字段,默认值为 lv_style_pretty

LV_PAGE_STYLE_SB: 用来修饰水平和垂直滚动条的,使用样式中的 body 字段,默认值为 lv_style_pretty_color,其中 body.padding.right 是用来控制垂直滚动条与页面右边缘的距离,当此值为正数时,是在页面内部,当为负值时,是在页面外部,而 body.padding.bottom 是用来控制水平滚动条与页面底边缘的距离,当此值为正数时,是在页面内部,当为负值时,是在页面外部,而 body.padding.inner 是用来设置滚动条的宽度

LV_PAGE_STYLE_EDGE_FLASH: 用来修饰边缘半圆弧动画效果的,一般只用到里面的 body.main_color, body.grad_color, body.opa 等样式字段,对于上边缘只用 grad_color 来设置半圆弧的颜色,对于下边缘只用 main_color 来设置半圆弧的颜色,而对于左和右边缘,main_color 和 grad_color 都会用到

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_page_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 清空页面内的所有子对象

```
void lv_page_clean(lv_obj_t * obj);
```

参数:

obj: 页面对象

其实就是把页面载体容器内的所有子对象全部删除掉

2.2.3 获取页面的载体容器对象

```
lv_obj_t * lv_page_get_scrl(const lv_obj_t * page);
```

参数:

page: 页面对象

返回值:

返回 page 页面内部的载体容器对象

拿到此载体容器对象之后,我们就可以拿 lv_cont 容器专有的 API 接口来操作其特性了

2.2.4 设置滚动条的模式

```
void lv_page_set_sb_mode(lv_obj_t * page, lv_sb_mode_t sb_mode);
```

参数:

page: 页面对象

sb_mode: 滚动条的模式,有如下 6 个可选值

LV_SB_MODE_OFF: 不显示任何滚动条

LV_SB_MODE_ON: 不论如何,都要把垂直和水平滚动条显示出来

LV_SB_MODE_DRAG: 当滚动页面时才显示出相应的滚动条,否则不显示

LV_SB_MODE_AUTO: 自动模式,当载体容器的大小超过 lv_page 页面的大小时,就会显示出相应的滚动条

LV_SB_MODE_HIDE: 把所有的滚动条都给隐藏了,跟 LV_SB_MODE_OFF 的作用差不多

LV_SB_MODE_UNHIDE: 把隐藏了的滚动条再次显示出来如果不设置的话,则默认就是 LV_SB_MODE_AUTO 模式

2.2.5 设置动画时长

```
void lv_page_set_anim_time(lv_obj_t * page, uint16_t anim_time);
```

参数:

page: 页面对象

anim_time: 动画时长,单位 ms

2.2.6 是否使能页面的滚动特性传递

```
void lv_page_set_scroll_propagation(lv_obj_t * page, bool en);
```

参数:

page: 页面对象

en: 是否使能滚动特性传递

首先说一下,此 API 接口不是很重要,基本用不到,我给大家稍微讲解一下,了解即可,假如当 A 页面对象内部含有一个 B 页面子对象时,而且 B 页面子对象的大小超过了 A 页面的大小,那么当我们想滚动 A 页面时,通常情况下是没办法实现的,因为 B 页面子对象已经把 A 页面给填充满了,A 页面已经没有空间来提供滚动操作了,所有的滚动操作都会作用在 B 页面子对象上,但是如果我们给 B 页面使能了滚动特性传递的话,即如下代码:

```
lv_page_set_scroll_propagation(B, true);
```

那么 B 页面会将它接受到的滚动操作传递给它的父对象,从而使 A 页面能够接受滚动操作了

2.2.7 是否使能边缘半圆弧动画效果

```
void lv_page_set_edge_flash(lv_obj_t * page, bool en);
```

参数:

page: 页面对象

en: true 代表使能,false 代表不使能

当使能之后,用户把页面滚动到自身某边缘时,此边缘上就会出现一个半圆弧的动画效果,主要是提示用户已经划到底,配合 LV_PAGE_STYLE_EDGE_FLASH 样式,可以给用户带来友好的体验,给出一个简单例子(只给出关键代码):

```
lv_style_copy(&edge_flash_style,&lv_style_plain_color);
edge_flash_style.body.main_color = LV_COLOR_RED;//红色
edge_flash_style.body.grad_color = LV_COLOR_GREEN;//绿色
edge_flash_style.body.opa = 150;//透明度
lv_obj_t * page1 = lv_page_create(scr,NULL);//创建 page1 页面
//设置 page1 的样式
lv_page_set_style(page1,LV_PAGE_STYLE_EDGE_FLASH,&edge_flash_style);
lv_page_set_edge_flash(page1,true);//使能 page1 的边缘半圆弧动画特效
```

然后四个边缘的演示效果分别如下:



图 2.2.7.1 上边缘

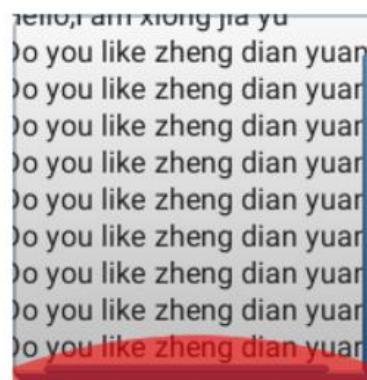


图 2.2.7.2 下边缘



图 2.2.7.3 左边缘



图 2.2.7.4 右边缘

从上面的效果我们可以得出上边缘只会用到 `grad_color` 颜色,而下边缘只会用到 `main_color` 颜色,而左和右边缘会把 `main_color` 和 `grad_color` 颜色都用到

2.2.8 设置载体容器的布局方式

```
static inline void lv_page_set_scrl_layout(lv_obj_t * page, lv_layout_t layout)
```

参数:

`page`: 页面对象

`layout`: 容器的布局方式,我们在 `lv_cont` 容器章节已经介绍过了

对于已经学习过 `lv_cont` 容器章节的朋友来说,,此 API 接口的实现机制其实特别简单,如下图所示:

```
229 | static inline void lv_page_set_scrl_layout(lv_obj_t * page, lv_layout_t layout)
230 |{
231 |     lv_cont_set_layout(lv_page_get_scrl(page), layout);
232 | }
```

图 2.2.8.1 此接口的内部实现原理

2.2.9 设置样式

```
void lv_page_set_style(lv_obj_t * page, lv_page_style_t type, const lv_style_t * style);
```

参数:

page: 页面对象

type: 设置那部分的样式,目前有如下 4 个可选值:

LV_PAGE_STYLE_BG: 修饰背景的

LV_PAGE_STYLE_SCRL: 修饰载体容器的

LV_PAGE_STYLE_SB: 修饰滚动条的

LV_PAGE_STYLE_EDGE_FLASH: 修饰边缘半圆弧动画效果的

style: 样式

2.2.10 判断是否滚动到了某边缘

```
bool lv_page_on_edge(lv_obj_t * page, lv_page_edge_t edge);
```

参数:

page: 页面对象

edge: 哪些边缘,有如下 4 个可选值:

LV_PAGE_EDGE_LEFT,

LV_PAGE_EDGE_TOP,

LV_PAGE_EDGE_RIGHT,

LV_PAGE_EDGE_BOTTOM

这些值之间是可以进行位或操作的

返回值:

如果已经滚动到了 edge 参数所指定的边缘,则返回 true,否则返回 false

2.2.11 使某个子对象获得聚焦

```
void lv_page_focus(lv_obj_t * page, const lv_obj_t * obj, lv_anim_enable_t anim_en);
```

参数:

page: 页面对象

obj: page 页面内的子对象

anim_en: 在获得聚焦的过程中,是否开启动画效果,有如下 2 个可选值

LV_ANIM_ON:开启动画效果

LV_ANIM_OFF:不开启动画效果

所谓的获得聚焦,说得直白一点,就是让 obj 子对象在 page 页面内处于可见状态

2.2.12 让页面在水平方向上滚动指定距离

```
void lv_page_scroll_hor(lv_obj_t * page, lv_coord_t dist);
```

参数:

page: 页面对象

dist: 要滚动的距离,单位为像素,当此值小于 0 时,是往右边滚动,大于 0 时,是往左边滚动

2.2.13 让页面在垂直方向上滚动指定距离

```
void lv_page_scroll_ver(lv_obj_t * page, lv_coord_t dist);
```

参数:

page: 页面对象

dist: 要滚动的距离,单位为像素,当此值小于 0 时,是往下边滚动,大于 0 时,是往上边滚动

2.2.14 获取页面可填充区域的宽度

```
lv_coord_t lv_page_get_fit_width(lv_obj_t * page);
```

参数:

page: 页面对象

返回值:

返回页面可填充区域的宽度

所谓的可填充区域就是指页面中实际能够用来存放子对象的空间,它的宽度会比页面的宽度小一点,因为页面的宽度减去左右内边距之后就是等于它的宽度了,用公式表示如下:

$$\text{fit_width} = \text{lv_obj_get_width}(\text{page}) - \text{bg_style}\rightarrow\text{body.padding.left} - \text{bg_style}\rightarrow\text{body.padding.right} - \text{scrl_style}\rightarrow\text{body.padding.left} - \text{scrl_style}\rightarrow\text{body.padding.right}$$
; 其中 `bg_style` 是背景容器的样式, `scrl_style` 是载体容器的样式



图 2.2.14.1 填充区域示意图

2.2.15 获取页面可填充区域的高度

```
lv_coord_t lv_page_get_fit_height(lv_obj_t * page);
```

参数:

page: 页面对象

返回值:

返回页面可填充区域的高度

和 lv_page_get_fit_width 接口的用途是差不多的

2.2.16 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建 2 种样式来分别修饰页面的滚动条和边缘半圆弧效果,然后创建一个 page1 页面,在其里面添加一个 label1 标签子对象和一个 btn1 按钮子对象,当点击 btn1 子对象时,会清空 page1 页面里的所有子对象,当按下 KEY0 按键时,会让 btn1 按钮获得聚焦而处于可见状态,当按下 KEY1 按键时,会让 page1 页面向下滚动,当按下 KEY2 按键时,会让 page1 页面向右滚动,当按下 WK_UP/KEY_UP 按键时,会来回切换页面的滚动条模式

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1,KEY2,WK_UP/KEY_UP 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_page_test.c 和 lv_page_test.h 俩个文件,其中 lv_page_test.c 文件的内容如下:

```
#include "lv_page_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_style_t sb_style;
lv_style_t edge_flash_style;
lv_obj_t * page1;
lv_obj_t * btn1;
lv_sb_mode_t sb_mode = LV_SB_MODE_AUTO;

//事件回调函数
void event_handler(lv_obj_t * obj,lv_event_t event)
{
    if(event==LV_EVENT_RELEASED)
    {
        lv_page_clean(page1); //清空内部的所有子对象
    }
}
```

```
}

//例程入口
void lv_page_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建 2 个样式
    //1.1 创建滚动条的样式
    lv_style_copy(&sb_style,&lv_style_plain);
    sb_style.body.main_color = LV_COLOR_BLACK;
    sb_style.body.grad_color = LV_COLOR_BLACK;
    sb_style.body.border.color = LV_COLOR_WHITE;
    sb_style.body.border.width = 1;
    sb_style.body.radius = LV_RADIUS_CIRCLE;
    sb_style.body.opa = LV_OPA_60;
    sb_style.body.padding.right = 3; //垂直滚动条的宽度
    sb_style.body.padding.bottom = 3; //水平滚动条的宽度
    sb_style.body.padding.inner = 8; //滚动条距页面边框的距离

    //1.2 创建边缘半圆弧样式
    lv_style_copy(&edge_flash_style,&lv_style_plain);
    edge_flash_style.body.main_color = LV_COLOR_GRAY;
    edge_flash_style.body.grad_color = LV_COLOR_GRAY;
    edge_flash_style.body.opa = 150; //透明度

    //2.创建页面
    //2.1 创建 page1 页面对象
    page1 = lv_page_create(scr,NULL);
    lv_obj_set_size(page1, 200, 200); //设置页面的大小
    lv_obj_align(page1, NULL, LV_ALIGN_CENTER,0,0); //与屏幕居中对齐
    //LV_SB_MODE_AUTO 是默认的滚动条模式
    lv_page_set_sb_mode(page1,sb_mode);
    lv_page_set_edge_flash(page1,true); //使能边缘半圆弧动画效果
    lv_page_set_style(page1, LV_PAGE_STYLE_SB,&sb_style); //设置滚动条的样式

    //设置边缘半圆弧样式
    lv_page_set_style(page1, LV_PAGE_STYLE_EDGE_FLASH,&edge_flash_style);

    //2.2 往 page1 页面中添加一个 label1 标签子对象
    lv_obj_t * label1 = lv_label_create(page1,NULL); //这里的父对象应该是 page1
    lv_label_set_long_mode(label1, LV_LABEL_LONG_BREAK); //设置长文本模式
```

```
//设置与 page1 页面的填充区域宽度相等
lv_obj_set_width(label1,lv_page_get_fit_width(page1));
lv_label_set_text(label1,"Hello zheng dian yuan zi, I am xiong jia yu,\n"
"sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.\n"
"Ut enim ad minim veniam, quis nostrud exercitation ullamco\n"
"laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure\n"
"dolor in reprehenderit in voluptate velit esse cillum dolore\n"
"eu fugiat nulla pariatur.\n"
"Excepteur sint occaecat cupidatat non proident, sunt in culpa\n"
"qui officia deserunt mollit anim id est laborum.");
```

//2.3 往 page1 页面中添加一个 btn1 按钮子对象
btn1 = lv_btn_create(page1,NULL); //这里的父对象应该是 page1
lv_obj_set_size(btn1,80,50);

```
//设置与 label1 的对齐方式
lv_obj_align(btn1,label1,LV_ALIGN_OUT_RIGHT_MID,0,0);
lv_obj_t * btn_label = lv_label_create(btn1,NULL); //给 btn1 按钮添加文本标题
lv_label_set_text(btn_label,"Clean");
```

```
//设置事件回调函数,当点击时清空 page1 内部的所有子对象
lv_obj_set_event_cb(btn1,event_handler);
```

```
}
```

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

    if(key==KEY0_PRES)
    {
        lv_page_focus(page1,btn1,LV_ANIM_ON); //使 btn1 按钮处于可见状态
    }else if(key==KEY1_PRES)
    {
        lv_page_scroll_ver(page1,-10); //让 page1 页面往下滚动
    }else if(key==KEY2_PRES)
    {
        lv_page_scroll_hor(page1,-10); //让 page1 页面往右滚动
    }else if(key==WKUP_PRES)
    {
        sb_mode++;
        if(sb_mode>LV_SB_MODE_UNHIDE)
            sb_mode = LV_SB_MODE_OFF;
```

```
lv_page_set_sb_mode(page1,sb_mode); //设置滚动条的模式  
}  
}
```

3.4 下载验证

把代码下载进去之后,就能看到如下所示的初始界面效果:

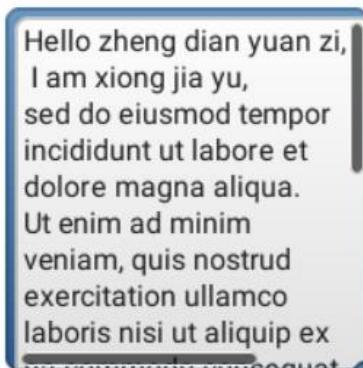


图 3.4.1 初始界面效果

然后我们可以按 WK_UP/KEY_UP 按键,来观看各种滚动条模式下的效果,接着我们来手动滚动页面或者按下 KEY0 键,让内部的按钮处于可见状态,如下所示:

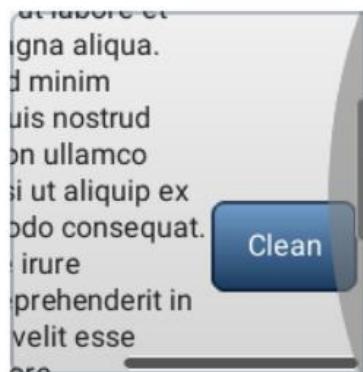


图 3.4.2 按钮处于可见状态

然后我们可以点击 Clean 按钮一下,它会把 page1 页面内的所有子对象全部给删除掉的,最后可以看到如下所示的空白页面效果:



图 3.4.3 空白页面

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_chart 图表

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

lv_chart 图表

1. 介绍

lv_chart 图表控件主要是由背景,水平垂直分割线,数据线(series)三部分构成的,其中数据线是 lv_chart 图表控件中最重要的部分,一条数据线就是一些数据点的集合,一条数据线可以被绘制成折线,柱状,散点,面积等几种主要的图表类型,或者是这几种图表类型共同组拼后的结果,即进行位或操作,而一个 lv_chart 图表控件中又可以同时绘制多条数据线,数量是不受限制的,每条数据线的颜色都可以单独指定,这是通过 `lv_chart_add_series(chart, color)` 添加数据线接口来具体操作的,我们说过了数据线就是数据点的集合,说的再具体一点就是 y 坐标点的集合,至于数据点的个数我们是可以通过 `lv_chart_set_point_count(chart, point_num)` 接口来设置的,对于数据值的修改或者更新,lv_chart 图表控件提供了如下四种方式:

- 1) 直接手动修改数据点的值,如 `ser1->points[2] = 45`,这样修改完后,还得手动调用 `lv_chart_refresh(chart)` 接口来刷新图表
- 2) 使用 `lv_chart_set_next(chart, ser, value)` 接口动态修改,这里又包含了 2 种更新方式,一种为左平移方式,另一种为环形覆盖方式
- 3) 使用 `lv_chart_init_points(chart, ser, value)` 接口把所有的数据点设置为同一个 value 值
- 4) 使用 `lv_chart_set_points(chart, ser, value_array)` 接口,传入数组的方式为数据线中的每一个数据点一一赋值

然后这里有一点需要注意的是,当数据点的值为 `LV_CHART_POINT_DEF` 时,那么此数据点将不会被绘制出来,而是被直接跳过了,会使整条数据线看起来有点断裂了的感觉.

前面说了数据点其实就是 y 坐标点,是不包含 x 坐标信息的,那么我们可以通过 `lv_chart_set_range(chart, y_min, y_max)` 接口来设置 y 轴的一个数值范围,使每一个数据点在 y 轴上都有一个确切的位置,有的同学可能就会疑惑了,数据点中不包含 x 坐标信息,那么 x 轴上如何确定位置呢?在 lv_chart 图表控件中它是这样处理的,它会根据数据线中的数据点总个数(用 n 表示)在 x 轴上做 n 等分处理,也就是说每个数据点之间的水平间距是相等的,而且是等于图表控件的宽度除以 n,我想 littleVGL 这么做的目的是为了减少 lv_chart 图表控件的复杂性和减少内存开销.

为了使图表看起来更美观,更人性化,我们也可以通过 `lv_chart_set_x/y_tick_text`, `lv_chart_set_x/y_tick_length`, `lv_chart_set_margin` 等接口来给图表设置 x 轴或者 y 轴的刻度线和主刻度标题.

2. lv_chart 的 API 接口

2.1 主要数据类型

2.1.1 图表类型数据类型

```
enum {
    LV_CHART_TYPE_NONE          = 0x00,
    LV_CHART_TYPE_LINE           = 0x01,
    LV_CHART_TYPE_COLUMN          = 0x02,
    LV_CHART_TYPE_POINT           = 0x04,
    LV_CHART_TYPE_VERTICAL_LINE   = 0x08,
    LV_CHART_TYPE_AREA             = 0x10,
};

typedef uint8_t lv_chart_type_t;
```

用来指定 lv_chart 图表控件中所有数据线的图表类型,一条数据线可以被绘制成折线图,柱状图,散点图,面积图,或者是这几种主要图表类型组拼之后的结果,因为这些值之间是可以进行位或操作的,比如为 LV_CHART_TYPE_LINE|LV_CHART_TYPE_COLUMN,那么你会看到折线图和柱状图共存的形式

LV_CHART_TYPE_NONE: 不显示任何数据线(series)

LV_CHART_TYPE_LINE: 数据线将会被绘制成折线图

LV_CHART_TYPE_COLUMN: 数据线将会被绘制成柱状图

LV_CHART_TYPE_POINT: 数据线将会被绘制成散点图

LV_CHART_TYPE_AREA: 数据线将会被绘制成面积图

LV_CHART_TYPE_VERTICAL_LINE: 当数据点的个数和图表控件的宽度相等时才会起作用,主要是性能优化了,它会比 LV_CHART_TYPE_AREA 的渲染速度更快,我们了解即可,一般用不到

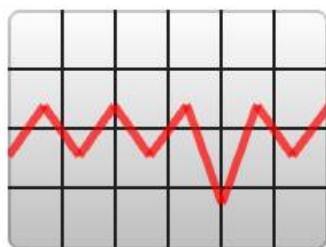


图 2.1.1.1 折线图



图 2.1.1.2 柱状图

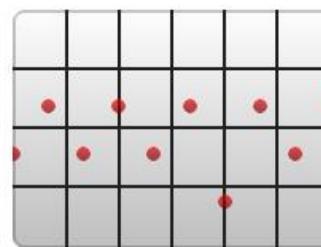


图 2.1.1.3 散点图

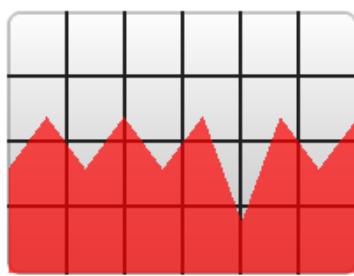


图 2.1.1.4 面积图

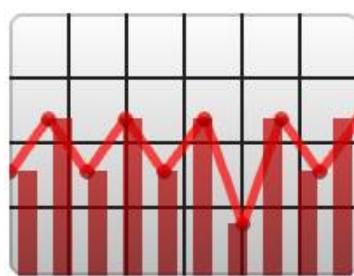


图 2.1.1.1 折线,柱状,散点共存图

2.1.2 数据点更新模式数据类型

```
enum {
    LV_CHART_UPDATE_MODE_SHIFT,
    LV_CHART_UPDATE_MODE_CIRCULAR,
};

typedef uint8_t lv_chart_update_mode_t;
```

此数据类型主要是用在 `lv_chart_set_update_mode(chart, update_mode)` 接口中,用来确定 `lv_chart_set_next` 接口的数据点更新模式的

`LV_CHART_UPDATE_MODE_SHIFT`: 左平移方式,当用 `lv_chart_set_next` 接口来添加一个新的数据点时,它会先把最左边的数据点给删除掉,然后把新的数据点放到最右边,看起来有一种向左平移的效果

`LV_CHART_UPDATE_MODE_CIRCULAR`: 环形覆盖方式,用 `lv_chart_set_next` 接口产生出来的新数据点会从最左边到最右边的方式依次替换掉老数据点,当到了最右边时,又会重新跳到最左边来开始新的循环

2.1.3 轴刻度绘制方式数据类型

```
enum {
    LV_CHART_AXIS_SKIP_LAST_TICK      = 0x00,
    LV_CHART_AXIS_DRAW_LAST_TICK      = 0x01
};

typedef uint8_t lv_chart_axis_options_t;
```

这个数据类型主要是用在 `lv_chart_set_x/y_tick_texts` 接口中的,用来确定是否需要绘制最后面的一个主刻度.

`LV_CHART_AXIS_SKIP_LAST_TICK`: 不绘制最后面的一个主刻度

`LV_CHART_AXIS_DRAW_LAST_TICK`: 绘制最后面的一个主刻度



图 2.1.3.1 不绘制最后面的主刻度效果



图 2.1.3.2 绘制最后面的主刻度效果

通过上面的效果对比,我们可以发现数字 5 那里有一个主刻度线的区别

2.1.4 数据线数据类型

```
typedef struct
{
    lv_coord_t * points;
    lv_color_t color;
    uint16_t start_point;
} lv_chart_series_t;
```

这是一个结构体,用来描述数据线的,其中 `points` 存放着所有的数据点,`color` 指定数据线的颜色,`start_point` 是归 littleVGL 库内部使用,我们用不到的

2.1.5 图表样式数据类型

```
enum {
    LV_CHART_STYLE_MAIN,
};

typedef uint8_t lv_chart_style_t;
```

图表的样式就一种,还算是简单的,我们主要是来介绍此样式中各字段的具体用途.

`LV_CHART_STYLE_MAIN`: 样式中的 `body` 字段用来修饰图表的背景,样式中的 `line` 字段用来修饰水平垂直分割线和 x/y 轴上的刻度线,样式中的 `text` 字段用来修饰 x/y 轴上的文本标题

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_chart_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

`par`: 父对象

`copy`: 拷贝的对象,如果无拷贝的话,传 `NULL` 值

返回值:

返回创建出来的对象,如果返回 `NULL` 的话,说明堆空间不够了

2.2.2 添加数据线

```
lv_chart_series_t * lv_chart_add_series(lv_obj_t * chart, lv_color_t color);
```

参数:

chart: 图表对象

color: 指定被添加数据线的颜色

返回值:

返回被添加的数据线对象

拿到数据线对象之后,是有作用的,因为后面的某些接口会用到此对象

2.2.3 清除某数据线中的所有数据点

```
void lv_chart_clear_serie(lv_obj_t * chart, lv_chart_series_t * serie);
```

参数:

chart: 图表对象

serie: 数据线对象

这个实现原理就是用 LV_CHART_POINT_DEF 给所有的数据点进行赋值,前面说过值为 LV_CHART_POINT_DEF 的数据点是不会被绘制出来的,注意,清除之后界面是不会自动刷新的,此时你可以调用 lv_chart_refresh(chart) 来手动刷新图表

2.2.4 设置水平和垂直分割线的条数

```
void lv_chart_set_div_line_count(lv_obj_t * chart, uint8_t hdiv, uint8_t vdiv);
```

参数:

chart: 图表对象

hdiv: 水平分割线的条数,默认为 3 条

vdiv: 垂直分割线的条数,默认为 5 条

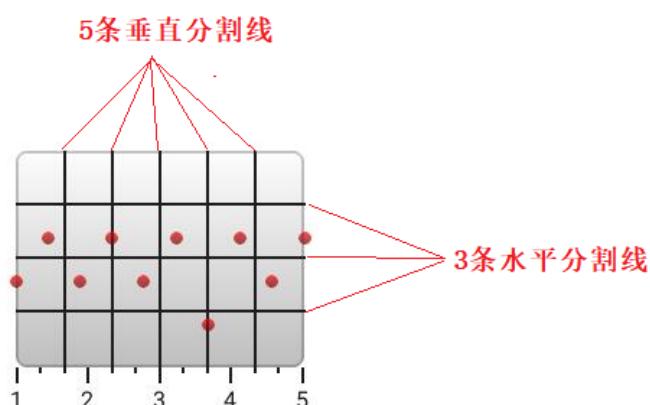


图 2.2.4.1 水平垂直分割线

2.2.5 设置 y 轴的数值范围

```
void lv_chart_set_range(lv_obj_t * chart, lv_coord_t ymin, lv_coord_t ymax);
```

参数:

chart: 图表对象

ymin: 最小值,默认值为 0,对应 y 轴的最底部点

ymax: 最大值,默认值为 100,对应 y 轴的最顶部点

2.2.6 设置图表的类型

```
void lv_chart_set_type(lv_obj_t * chart, lv_chart_type_t type);
```

参数:

chart: 图表对象

type: 图表类型,有如下 6 个可选值:

LV_CHART_TYPE_NONE: 不显示

LV_CHART_TYPE_LINE: 折线图

LV_CHART_TYPE_COLUMN: 柱状图

LV_CHART_TYPE_POINT: 散点图

LV_CHART_TYPE_VERTICAL_LINE: 基本用不到,了解即可

LV_CHART_TYPE_AREA: 面积图

这个接口设置了所有数据线的统一图表类型,我们是不能为某条数据线单独指定图表类型的

2.2.7 设置数据点的个数

```
void lv_chart_set_point_count(lv_obj_t * chart, uint16_t point_cnt);
```

参数:

chart: 图表对象

point_cnt: 数据点的个数

这个接口设置了每条数据线所具有的数据点个数,我们是不能为某条数据线单独指定数据点个数的

2.2.8 设置数据线的透明度

```
void lv_chart_set_series_opa(lv_obj_t * chart, lv_opa_t opa);
```

参数:

chart: 图表对象

opa: 透明度,默认值为 LV_OPA_COVER

2.2.9 设置数据线的宽度

```
void lv_chart_set_series_width(lv_obj_t * chart, lv_coord_t width);
```

参数:

chart: 图表对象

width: 数据线的宽度

当数据线是折线图时,它就是设置线的宽度,当数据线是散点图时,它就是设置点的半径,此 width 宽度值对柱状图无意义,因为柱状图是内部自动计算宽度的

2.2.10 设置数据线的黑阴影效果

```
void lv_chart_set_series_darking(lv_obj_t * chart, lv_opa_t dark_eff);
```

参数:

chart: 图表对象

dark_eff: 黑阴影的透明度,当设为 0 时,相当于没有黑阴影效果

此黑阴影效果只对散点图或者柱状图有效果,设置之后图表更美观一点

2.2.11 给所有数据点设置统一的值

```
void lv_chart_init_points(lv_obj_t * chart, lv_chart_series_t * ser, lv_coord_t y);
```

参数:

chart: 图表对象

ser: 数据线对象

y: 数据点的值

2.2.12 给所有数据点一一赋值

```
void lv_chart_set_points(lv_obj_t * chart, lv_chart_series_t * ser, lv_coord_t y_array[]);
```

参数:

chart: 图表对象

ser: 数据线对象

y_array: 数组形式,存放着每个数据点的值

2.2.13 添加新的数据点

```
void lv_chart_set_next(lv_obj_t * chart, lv_chart_series_t * ser, lv_coord_t y);
```

参数:

chart: 图表对象

ser: 数据线对象

y: 数据点的值

往指定的 ser 数据线中添加值为 y 的新数据点,在添加新数据点时,这里有 2 种数据更新模式,请看 lv_chart_set_update_mode 接口

2.2.14 设置数据更新模式

```
void lv_chart_set_update_mode(lv_obj_t * chart, lv_chart_update_mode_t update_mode);
```

参数:

chart: 图表对象

update_mode: 数据更新模式,有如下 2 个可选值:

LV_CHART_UPDATE_MODE_SHIFT: 左平移方式,当用 lv_chart_set_next 接口来添加一个新的数据点时,它会先把最左边的数据点给删除掉,然后把新的数据点放到最右边,看起来有一种向左平移的效果

LV_CHART_UPDATE_MODE_CIRCULAR: 环形覆盖方式,用 lv_chart_set_next 接口产生出来

的新数据点会从最左边到最右边的方式依次替换掉老数据点,当到了最右边时,又会重新跳到最左边来开始新的循环

2.2.15 设置样式

```
static inline void lv_chart_set_style(lv_obj_t * chart, lv_chart_style_t type, const lv_style_t * style);
```

参数:

chart: 图表对象

type: 设置哪部分的样式,目前就 LV_CHART_STYLE_MAIN 一个可选值

style: 样式

2.2.16 设置 X 或 Y 轴上的刻度线长度

```
void lv_chart_set_x_tick_length(lv_obj_t * chart, uint8_t major_tick_len, uint8_t minor_tick_len);
void lv_chart_set_y_tick_length(lv_obj_t * chart, uint8_t major_tick_len, uint8_t minor_tick_len);
```

参数:

chart: 图表对象

major_tick_len: 主刻度线的长度

minor_tick_len: 次刻度线的长度

为了让大家更容易理解什么是主刻度线和次刻度线,请看下图

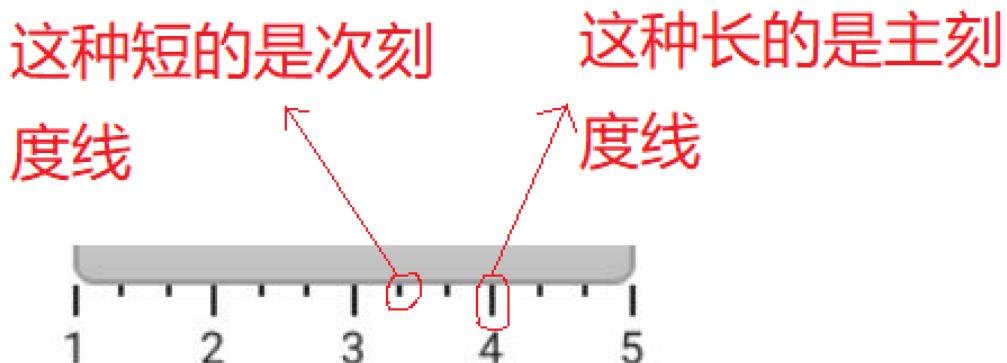


图 2.2.16.1 主次刻度线图解

2.2.17 设置 X 或 Y 轴上的刻度线数量和主刻度线标题

```
void lv_chart_set_x_tick_texts(lv_obj_t * chart, const char * list_of_values, uint8_t num_tick_marks, lv_chart_axis_options_t options);
void lv_chart_set_y_tick_texts(lv_obj_t * chart, const char * list_of_values, uint8_t num_tick_marks, lv_chart_axis_options_t options);
```

参数:

chart: 图表对象

list_of_values: 每个主刻度线对应的文本标题,标题之间需要用\n换行符来作为间隔标志,传入了多少个标题就意味着有多少个主刻度线,如下面例子所示:

```
const char * list_of_values = "1\n2\n3\n4\n5";
```

这是传入了 1, 2, 3, 4, 5 总共五个标题,也意味着有五个主刻度线

num_tick_marks: 相邻的两个主刻度线之间又有 num_tick_marks-1 个次刻度线

options: 决定是否绘制最末尾的一个主刻度线,有如下 2 个可选值:

LV_CHART_AXIS_SKIP_LAST_TICK: 不绘制最后面的一个主刻度

LV_CHART_AXIS_DRAW_LAST_TICK: 绘制最后面的一个主刻度

最后给大家举一个小例子如下:

```
lv_chart_set_x_tick_texts(chart, "1\n2\n3\n4\n5", 3, LV_CHART_AXIS_DRAW_LAST_TICK);
lv_chart_set_x_tick_length(chart, 10, 3);
lv_chart_set_margin(chart, 30); //这个接口后面讲解,必须得设置,否则看不到效果的
```



图 2.2.17.1 刻度数量和主刻度标题演示效果

2.2.18 设置刻度区域的高度

```
void lv_chart_set_margin(lv_obj_t * chart, uint16_t margin);
```

参数:

chart: 图表对象

margin: 刻度区域的高度,单位像素

如果我们的图表在 x 或者 y 轴上设置了刻度线和主刻度标题的话,那么我们必须得调用 `lv_chart_set_margin` 接口来设置刻度区域的高度,否则可能会看不到完整的效果,我们设置的刻度区域高度必须得满足以下要求:

刻度区域高度 $margin > (\text{主刻度线长度 } \text{major_tick_len} + \text{ 主刻度标题的文本高度})$

其中主刻度标题的文本高度取决于样式中所用字体的大小,最后为了方便大家理解,给出一张图解.

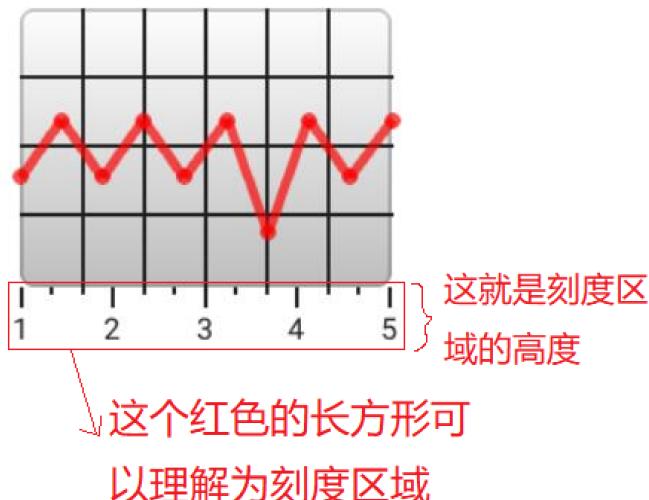


图 2.2.18.1 刻度区域高度的图解

2.2.19 刷新图表

```
void lv_chart_refresh(lv_obj_t * chart);
```

参数:

chart: 图表对象

有些操作之后,图表是不会自动刷新界面的,此时我们可以通过此 API 接口来实现手动刷新,其实此 API 接口内部就是在调用 `lv_obj_invalidate(chart)`;

2.2.20 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰图表控件,然后我们接着创建一个 chart1 图表,设置好它的各种属性,比如透明度,y 轴数值范围,刻度线,数据点个数等等,然后我们往 chart1 图表中添加 series1 和 series2 俩条数据线,series1 是采用数组一一赋值和手动直接修改的方式,而 series2 是采用统一赋值的方式,当按下 KEY0 键时,会来回切换 chart1 的图表类型,当按下 KEY1 键时,会往 series1 数据线上添加新的数据点,当按下 KEY2 键时,会来回切换数据点的更新模式

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1,KEY2 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_chart_test.c 和 lv_chart_test.h 俩个文件,其中 lv_chart_test.c 文件的内容如下:

```
#include "lv_chart_test.h"
#include "lvgl.h"
#include "key.h"

lv_style_t scr_style;

#define POINT_COUNT    10 //每条数据线所具有的数据点个数
lv_style_t main_style;
const lv_coord_t series1_y[POINT_COUNT] = {30,60,40,60,20,60,50,80,60,80};
lv_obj_t * chart1;
lv_chart_series_t * series1;
lv_chart_series_t * series2;

//例程入口
void lv_chart_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象
```

```
lv_style_copy(&scr_style,&lv_style_plain_color);
scr_style.body.main_color = LV_COLOR_CYAN;
scr_style.body.grad_color = scr_style.body.main_color;

//给屏幕设置一个纯色的背景,让我们的图表显示更明显,同时可以解决色点现象
lv_obj_set_style(scr,&scr_style);

//1.创建样式
lv_style_copy(&main_style,&lv_style_pretty);
main_style.body.main_color = LV_COLOR_WHITE;//主背景为纯白色
main_style.body.grad_color = main_style.body.main_color;
main_style.body.border.color = LV_COLOR_BLACK;//边框的颜色
main_style.body.border.width = 3;//边框的宽度
main_style.body.border.opa = LV_OPA_COVER;
main_style.body.radius = 0;
main_style.line.color = LV_COLOR_GRAY;//分割线和刻度线的颜色
main_style.text.color = LV_COLOR_BLUE;//主刻度标题的颜色

//2.创建图表对象
//2.1 创建图表对象
chart1 = lv_chart_create(scr,NULL);
lv_obj_set_size(chart1,180,200);//设置图表的大小
lv_obj_align(chart1,NULL,LV_ALIGN_CENTER,20,0);//设置对齐方式

//设置为散点和折线的组合
lv_chart_set_type(chart1,LV_CHART_TYPE_POINT|LV_CHART_TYPE_LINE);

//设置数据线的透明度,不设置的话,则 LV_OPA_COVER 是默认值
lv_chart_set_series_opa(chart1,LV_OPA_80);
lv_chart_set_series_width(chart1,4);//设置数据线的宽度
lv_chart_set_series_darking(chart1,LV_OPA_80);//设置数据线的黑阴影效果
lv_chart_set_style(chart1,LV_CHART_STYLE_MAIN,&main_style);//设置样式

//设置每条数据线所具有的数据点个数,如果不设置的话,则默认值是 10
lv_chart_set_point_count(chart1,POINT_COUNT);
lv_chart_set_div_line_count(chart1,4,4);//设置水平和垂直分割线
lv_chart_set_range(chart1,0,100);//设置 y 轴的数值范围,[0,100]也是默认值

//设置 y 轴的主刻度线长度和次刻度线长度
lv_chart_set_y_tick_length(chart1,10,3);

lv_chart_set_y_tick_texts(chart1,"100\n80\n60\n40\n20\n0",2,LV_CHART_AXIS_D
RAW_LAST_TICK);
```

```
//设置 x 轴的主刻度线长度和次刻度线长度
lv_chart_set_x_tick_length(chart1,10,3);

//设置 x 轴的刻度数和主刻度标题
lv_chart_set_x_tick_texts(chart1,"0\n2\n4\n6\n8\n10",2,LV_CHART_AXIS_DRAW_LAST_TICK);
lv_chart_set_margin(chart1,40);//设置刻度区域的高度

//2.2 往图表中添加第 1 条数据线
series1 = lv_chart_add_series(chart1, LV_COLOR_RED); //指定为红色
lv_chart_set_points(chart1, series1, (lv_coord_t*)series1_y); //初始化数据点的值
series1->points[1] = 70; //也可以采用直接修改的方式
lv_chart_refresh(chart1); //如果是采用直接修改的方式,请最好调用一下刷新操作

//2.3 往图表中添加第 2 条数据线
series2 = lv_chart_add_series(chart1, LV_COLOR_BLUE); //指定为蓝色
lv_chart_init_points(chart1, series2, 90); //给所有数据点设置统一的值

}

//按键处理
void key_handler()
{
    static u8 type_index = 0; //图表的模式
    static lv_chart_update_mode_t update_mode = LV_CHART_UPDATE_MODE_SHIFT; //数据点更新模式

    u8 key = KEY_Scan(0);

    if(key==KEY0_PRES)
    {
        //为了使数据线的图表类型变化看得更明显,我们把 series2 数据线给隐藏
        //了,防止干扰视线
        lv_chart_clear_serie(chart1, series2);

        //来回切换图表类型
        lv_chart_set_type(chart1, (lv_chart_type_t)(0x01<<type_index));
        type_index++;
        if(type_index==6)
            type_index = 0;
    }else if(key==KEY1_PRES)
    {
```

```
//往 series1 数据线上添加新的数据点
//可以切换到不同的数据点更新模式来观看不同的效果
lv_chart_set_next(chart1,series1,40);
}else if(key==KEY2_PRES)
{
    //来回切换数据点的更新模式
    //如果不设置的话,则默认是 LV_CHART_UPDATE_MODE_SHIFT 左平移模式
    update_mode = !update_mode;
    lv_chart_set_update_mode(chart1,update_mode);

    //为了使效果看起来更明显,我们将 series1 数据线的值初始化到原始状态
    lv_chart_set_points(chart1,series1,(lv_coord_t*)series1_y);
}
}
```

3.4 下载验证

把代码下载进去之后,我们可以看到如下所示的界面效果:

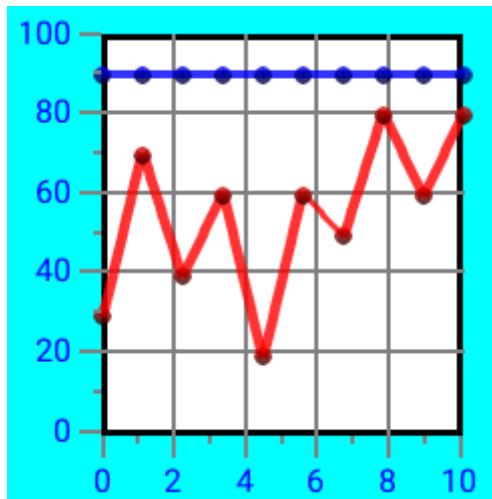


图 3.4.1 初始界面效果

然后我们可以按下 KEY0 键来切换图表类型,可以依次看到如下所示界面效果:

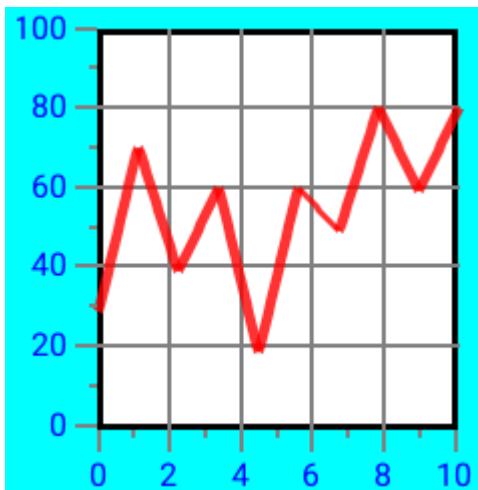


图 3.4.2 折线图

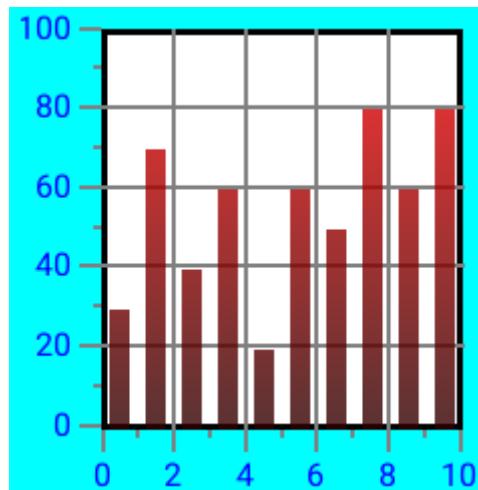


图 3.4.3 柱状图

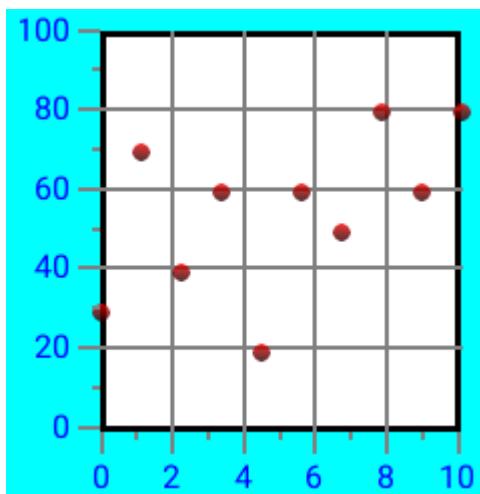


图 3.4.4 散点图

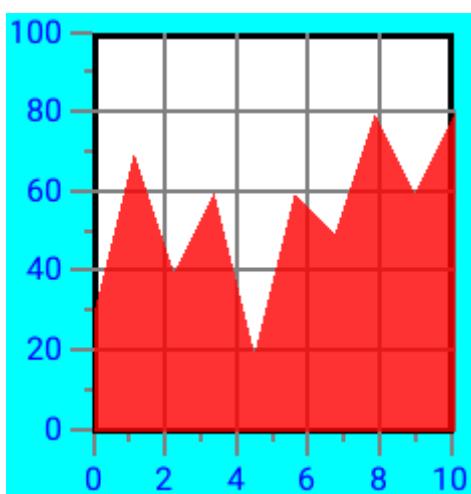


图 3.4.5 面积图

注意上面是为了使图表类型效果观看的更明显,所以在按下 KEY0 键时,我们用代码把 series2 数据线给隐藏了,最后我们还可以按下 KEY1 键来往 series1 数据线中添加新的数据点,以及配合 KEY2 键来改变数据点的更新模式

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

正点原子 littleVGL 开发指南

lv_table 表格

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

lv_table 表格

1. 介绍

lv_table 是 littleVGL 中的表格控件,谈到表格,我相信大家并不陌生,它可以非常友好而又直观的展示数据,它是由行(row),列(col)组成的,说的再直白一点,它就是由一个一个的单元格(cell)组成的,在 littleVGL 中的 lv_table 表格控件是非常轻量化的,经过了专门的优化,因为它的单元格(cell)只能存放文本形式的内容,不支持存放其他任何类型的对象或者控件,然后单元格中的文本内容是一个伪标签对象,即不是真正的 lv_label 标签对象,因为它是 lv_table 控件内部通过绘图函数纯绘制出来的效果,所以可以认为每个单元格基本没有内存消耗.

表格控件可以通过 lv_table_set_row_cnt(table, row_cnt) 接口来设置其所占的行数,通过 lv_table_set_col_cnt(table, col_cnt) 接口来设置其所占的列数,如果不进行单元格合并操作的话,那么此表格所具有的单元格数就等于 $row_cnt * col_cnt$,另外我们可以通过 lv_table_set_col_width(table, col_id, width) 接口来给每一列设置单独的宽度,而表格控件的宽度就是所有列的宽度总和,然后这里有一点需要注意的是,每一行的高度我们是没办法来指定的,因为内部是根据字体大小,内间距等因素来自动计算出每一行的高度,即高度表现为自适应特性.

最后我们要来讲一下单元格,对于表格来说,它是最重要的,因为它是文本数据的载体,我们可以通过 lv_table_set_cell_value(table, row, col, "Content") 接口来给每一个单元格设置文本内容,由 row 和 col 这两个行列参数就可以确定一个单元格的具体位置了,此文本内容在内部会做拷贝操作的,所以不用担心在外部是否被释放了,然后接着可以通过 lv_table_set_cell_align(table, row, col, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT) 接口来设置文本内容在单元格中的水平对齐方式,在 lv_table 表格控件中,每一个单元格都具有四种类型,这是通过 lv_table_set_cell_type(table, row, col, type) 接口来指定单元格具体为那一种类型的,那这四种类型之间有什么区别呢?其实没什么区别,之所以弄出四种类型,主要是为了让单元格在外观样式上能够得到区分,我们可以给每种不同类型的单元格单独指定一种样式,比如说表格的第一行一般都作为标题行,而标题行的外观样式一般都是比较突出醒目的,如果没有单元格类型这个功能,我们就没有办法实现标题行和数据行之间的样式区分了,当然了,作用远不止于此,比如还可以用来实现单元格的高亮显示等等,单元格还有一个重要的特性,那就是合并操作,不过只能实现水平方向上的合并,而不能实现垂直方向上的合并,即上下两个单元格之间是没有办法合并为一个单元格的,对于水平方向上的合并操作是通过 lv_table_set_cell_merge_right(table, col, row, true) 接口来完成的.

Name	Price
Apple	\$7
Banana	\$4
Citron	\$6

图 1.1 lv_table 表格的外观

2. lv_table 的 API 接口

2.1 主要数据类型

2.1.1 表格样式数据类型

```
enum {
    LV_TABLE_STYLE_BG, //表格的背景样式
    LV_TABLE_STYLE_CELL1,//表格的类型 1 单元格样式
    LV_TABLE_STYLE_CELL2,//表格的类型 2 单元格样式
    LV_TABLE_STYLE_CELL3,//表格的类型 3 单元格样式
    LV_TABLE_STYLE_CELL4,//表格的类型 4 单元格样式
};

typedef uint8_t lv_table_style_t;
```

我们说了表格中的单元格具有四种类型,而 LV_TABLE_STYLE_CELL1/2/3/4 就是用来分别修饰这四种类型的单元格.

LV_TABLE_STYLE_BG: 用来修饰表格的背景,只用了样式中的 body 字段,默认值为 lv_style_plain_color

LV_TABLE_STYLE_CELL1/2/3/4: 用来分别修饰四种类型的单元格,只用了样式中的 body 字段,默认值为 lv_style_plain

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_table_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置表格所具有的行数

```
void lv_table_set_row_cnt(lv_obj_t * table, uint16_t row_cnt);
```

参数:

table: 表格对象

row_cnt: 表格所具有的总行数

2.2.3 设置表格所具有的列数

```
void lv_table_set_col_cnt(lv_obj_t * table, uint16_t col_cnt);
```

参数:

table: 表格对象

col_cnt: 表格所具有的总列数,最大值不能超过 LV_TABLE_COL_MAX 的值,而

LV_TABLE_COL_MAX 宏的值是在 lv_table.h 头文件中定义的,默认值为 12,如果你觉得

最大 12 列满足不了你的需求,那么你可以手动修改 LV_TABLE_COL_MAX 宏的值

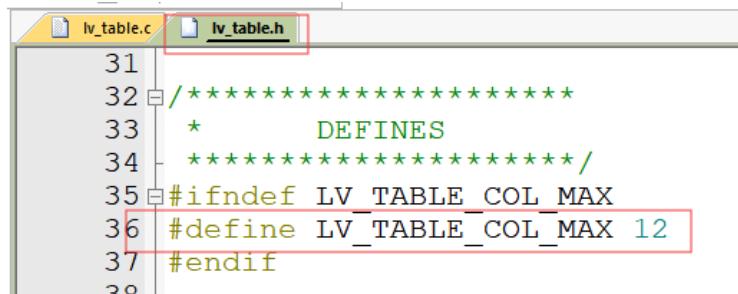


图 2.2.3.1 LV_TABLE_COL_MAX 宏的定义

2.2.4 设置单元格的文本内容

```
void lv_table_set_cell_value(lv_obj_t * table, uint16_t row, uint16_t col, const char * txt);
```

参数:

table: 表格对象

row: 单元格所在的行,从 0 开始

col: 单元格所在的列,从 0 开始

txt: 单元格的文本内容,此文本内容在内部会进行拷贝操作,所以不用担心其在外部是否被释放了

由传入的 row 和 col 两个参数,就可以定位到一个具体的单元格了,当文本内容的宽度大于所在列的宽度时,文本内容会自动进行换行的,从而也会导致此单元格的高度增加,即整行的高度增加,如果你想手动让文本内容换行的话,那么你可以使用\n 转义字符,但这种手动换行的效果有一点特殊,它会在换行的两段文本之间加一根和此单元格等宽的水平横线,请看如下例子:

1) 不换行时的效果

```
lv_table_set_cell_value(table1, 3, 0, "Xiong jia yu");
```

Name	Price
Apple	\$7
Banana	\$4
Xiong jia yu	\$999999

图 2.2.4.1 不换行时的效果

2) 使用\n 手动换行的效果

```
lv_table_set_cell_value(table1, 3, 0, "Xiong \njia yu" );
```

Name	Price
Apple	\$7
Banana	\$4
Xiong jia yu	\$999999

图 2.2.4.2 手动换行时的效果

3) 自动换行时的效果

```
lv_table_set_cell_value(table1, 3, 0, "Xiong jia yu, Are you ok?" );
```

Name	Price
Apple	\$7
Banana	\$4
Xiong jia yu, Are you ok?	\$999999

图 2.2.4.3 自动换行时的效果

2.2.5 设置某一列的宽度

```
void lv_table_set_col_width(lv_obj_t * table, uint16_t col_id, lv_coord_t w);
```

参数:

table: 表格对象

col_id: 列的位置,从 0 开始

w: 列的宽度

如果不设置列的宽度,那么列宽度的默认值为 LV_DPI,表格控件的宽度就是所有列的宽度总和,你用 lv_obj_set_size 接口来给表格控件设置宽度是无效的,然后这里有一点需要注意的是,每一行的高度我们是没办法来指定的,因为内部是根据字体大小,内间距等因素来自动计算出每一行的高度,即高度表现为自适应特性

2.2.6 设置单元格中文本的水平对齐方式

```
void lv_table_set_cell_align(lv_obj_t * table, uint16_t row, uint16_t col, lv_label_align_t align);
```

参数:

table: 表格对象

row: 文本内容所在的行,从 0 开始

col: 文本内容所在的列,从 0 开始

align: 水平对齐方式,有如下三个可选值

LV_LABEL_ALIGN_LEFT:居左对齐

LV_LABEL_ALIGN_CENTER:居中对齐

LV_LABEL_ALIGN_RIGHT:居右对齐

2.2.7 设置单元格的类型

```
void lv_table_set_cell_type(lv_obj_t * table, uint16_t row, uint16_t col, uint8_t type);
```

参数:

table: 表格对象

row: 单元格所在的行,从 0 开始

col: 单元格所在的列,从 0 开始

type: 单元格的类型,可选值为 1, 2, 3, 4 ,分别对应四种类型

如果不设置的话,那么单元格的默认类型为 1,即第一种类型,我们一般把表格的第一行作为标题行,在外观样式上是比较突出醒目的,所以我们可以把第一行中的所有单元格的类型设置为 2 或者其他非 1 值,这样就可以和数据行在外观上有区分了

2.2.8 是否禁止某单元格中文本的自动换行功能

```
void lv_table_set_cell_crop(lv_obj_t * table, uint16_t row, uint16_t col, bool crop);
```

参数:

table: 表格对象

row: 单元格所在的行,从 0 开始

col: 单元格所在的列,从 0 开始

crop: 如果为 true 的话,那么代表禁止自动换行功能,当文本内容超过所在列的宽度时,超出内容将会被截断,如果为 false 的话,那么代表使能自动换行功能,默认值就是为 false 的

2.2.9 合并右边的单元格

```
void lv_table_set_cell_merge_right(lv_obj_t * table, uint16_t row, uint16_t col, bool en);
```

参数:

table: 表格对象

row: 单元格所在的行,从 0 开始

col: 单元格所在的列,从 0 开始

en: 是否把(row,col)单元格和(row,col+1)单元格合并为一个单元格

2.2.10 设置样式

```
void lv_table_set_style(lv_obj_t * table, lv_table_style_t type, const lv_style_t * style);
```

参数:

table: 表格对象

type: 设置哪一部分的样式,有如下 5 个可选值:

LV_TABLE_STYLE_BG: 表格的背景样式

LV_TABLE_STYLE_CELL1: 表格的类型 1 单元格样式

LV_TABLE_STYLE_CELL2: 表格的类型 2 单元格样式

LV_TABLE_STYLE_CELL3: 表格的类型 3 单元格样式

LV_TABLE_STYLE_CELL4: 表格的类型 4 单元格样式

style: 样式

2.2.11 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建 3 个自定义样式,分别用来修饰三种类型的单元格,第一个样式用于修饰普通的数据行,第二个样式用于修饰标题行,即第一行,第三个样式用于对某个单元格进行红色高亮显示,然后接着创建一个表格控件,设置其 4 行 3 列,以及每列的宽度,再接着初始化所有单元格的文本内容,以及文本内容的水平对齐方式,当按下 KEY0 按键时,来设置(1,0)单元格是否禁止自动换行功能,当按下 KEY1 按键时,来设置是否将(3,0)单元格和(3,1)单元格合并为一个单元格,当按下 WKUP 按键时,设置(2,0)单元格进行手动换行操作.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1,WKUP 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_table_test.c 和 lv_table_test.h 俩个文件,其中 lv_table_test.c 文件的内容如下:

```
#include "lv_table_test.h"
#include "lvgl.h"
#include "key.h"

lv_style_t cell1_style;
lv_style_t cell2_style;
lv_style_t cell3_style;
lv_obj_t * table1;
bool is_cell_crop = false;//默认是使能文本自动换行功能的
bool is_cell_merge = false;//是否合并单元格,默认不合并

#define TABLE_COL_CNT    3    //表格的列数
#define TABLE_ROW_CNT    4    //表格的行数

//定义每一行单元格的文本内容
const char * const TABLE_CELL_VALUE[TABLE_ROW_CNT][TABLE_COL_CNT] = {
    {"Name","Work","Math"},//第一行,作为标题行
```

```
{"Zhend dian yuan zi","95","90"},//数据行
 {"Xiong jia yu","56","99"},//数据行
 {"Fish","88","93"}//数据行
};

//每一列的宽度
const uint16_t TABLE_COL_WIDTH[TABLE_COL_CNT] = {100,60,60};

//例程入口
void lv_table_test_start()
{
    uint16_t row,col;

    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建样式
    //1.1 创建第一种单元格样式,用来修饰普通数据单元格
    lv_style_copy(&cell1_style,&lv_style_plain);
    cell1_style.body.border.width = 1;
    cell1_style.body.border.color = LV_COLOR_BLACK;

    //1.2 创建第二种单元格样式,用来修饰标题行中的单元格
    lv_style_copy(&cell2_style,&lv_style_plain_color);
    cell2_style.body.border.width = 1;
    cell2_style.body.border.color = LV_COLOR_BLACK;
    cell2_style.body.main_color = LV_COLOR_SILVER;//银色的背景
    cell2_style.body.grad_color = LV_COLOR_SILVER;
    cell2_style.text.color = LV_COLOR_BLACK;

    //1.3 创建第三种单元格样式,用来修饰特殊的单元格,比如进行数据高亮显示等等
    lv_style_copy(&cell3_style,&lv_style_plain);
    cell3_style.body.border.width = 1;
    cell3_style.body.border.color = LV_COLOR_BLACK;
    cell3_style.text.color = LV_COLOR_RED;//文本颜色为红色,高亮显示

    //2.创建表格
    table1 = lv_table_create(scr,NULL);
    lv_table_set_col_cnt(table1,TABLE_COL_CNT); //设置表格的总列数
    lv_table_set_row_cnt(table1,TABLE_ROW_CNT); //设置表格的总行数
    //设置每一列的宽度以及标题行的文本对齐方式和单元格类型
    for(col=0;col<TABLE_COL_CNT;col++)
    {
        //设置每一列的宽度
        lv_table_set_col_width(table1,col,TABLE_COL_WIDTH[col]);
    }
}
```

```
//标题行的文本内容居中对齐
lv_table_set_cell_align(table1,0,col,LV_LABEL_ALIGN_CENTER);
lv_table_set_cell_type(table1,0,col,2);//设置为第二种单元格类型
}

//给(2,1)单元格设置为第三种类型,具有红色高亮显示的外观
lv_table_set_cell_type(table1,2,1,3);

//设置所有单元格的文本内容
for(row=0;row<TABLE_ROW_CNT;row++)
{
    for(col=0;col<TABLE_COL_CNT;col++)
    {
        lv_table_set_cell_value(table1,row,col,(const char*)TABLE_CELL_VALUE
[row][col]);//设置文本内容
    }
}

//设置表格的背景样式,为透明
lv_table_set_style(table1,LV_TABLE_STYLE_BG,&lv_style_transp_tight);

//设置第一种单元格的样式
lv_table_set_style(table1,LV_TABLE_STYLE_CELL1,&cell1_style);

//设置第二种单元格的样式
lv_table_set_style(table1,LV_TABLE_STYLE_CELL2,&cell2_style);

//设置第三种单元格的样式
lv_table_set_style(table1,LV_TABLE_STYLE_CELL3,&cell3_style);
lv_obj_align(table1,NULL,LV_ALIGN_CENTER,0,0);//设置表格与屏幕居中对齐
}

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

    if(key==KEY0_PRES)
    {
        is_cell_crop = !is_cell_crop;
        //设置(1,0)单元格是否禁止自动换行功能
        lv_table_set_cell_crop(table1,1,0,is_cell_crop);
        lv_obj_invalidate(table1);//得手动刷新 table1 表格
    }else if(key==KEY1_PRES)
```

```
{  
    is_cell_merge = !is_cell_merge;  
  
    //是否将(3,0)单元格和(3,1)单元格合并为一个单元格  
    lv_table_set_cell_merge_right(table1,3,0,is_cell_merge);  
}else if(key==WKUP_PRES)  
{  
    lv_table_set_cell_value(table1,2,0,"Xiong\njia yu");//用\n 转义字符进行手动换行  
}  
}
```

3.4 下载验证

把代码下载进去之后,我们可以看到如下所示的界面效果:

Name	Work	Math
Zhend dian yuan zi	95	90
Xiong jia yu	56	99
Fish	88	93

图 3.4.1 初始界面效果

然后我们按一下 KEY0 按键,禁止掉(1,0)单元格的自动换行功能,即表现为截断效果,如下图所示:

Name	Work	Math
Zhend dian yu	95	90
Xiong jia yu	56	99
Fish	88	93

图 3.4.2 禁止自动换行效果

再接着我们按一下 KEY1 按键,让(3,0)单元格和(3,1)单元格合并为一个单元格,如下图所示:

Name	Work	Math
Zhend dian yuan zi	95	90
Xiong jia yu	56	99
Fish		93

图 3.4.3 单元格合并效果

最后我们按一下 WKUP 按键,让(2,0)单元格进行手动换行操作,如下图所示:

Name	Work	Math
Zhend dian yuan zi	95	90
Xiong jia yu	56	99
Fish	88	93

图 3.4.4 手动换行效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_reload 预加载

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

lv_preload 预加载

1. 介绍

lv_preload 预加载控件可以说是进度条类型中的一种,但它有点特殊,它不能表示出当前事务的具体进度值,它只能示意当前事务正在处理中,所以它有它专门的应用场景,lv_preload 预加载控件是基于 lv_arc 控件衍生出来的,所以在外观上,它们之间是差不多的,lv_preload 是由一个圆环作为背景,然后还有一个小圆弧绕着这个背景圆环在做旋转动画,如下图所示:



2. lv_preload 预加载控件的外观

其中小圆弧所占的角度是通过 `lv_preload_set_arc_length(preload, deg)` 接口来设置的,然后小圆弧的旋转动画方式有俩种,这俩种方式的共同特点是在顶部时速度慢,不同点是,在旋转的过程中,一种是小圆弧对应的角度保持不变,另外一种是小圆弧对应的角度由小变大,再由大变小来回交替,可以通过 `lv_preload_set_type(preload, type)` 接口来设置到底为哪一种旋转方式,还可以通过 `lv_preload_set_dir(preload, dir)` 接口来设置旋转的方向,默认为顺时针旋转.

3. lv_preload 的 API 接口

2.1 主要数据类型

2.1.1 旋转类型数据类型

```
enum {
    LV_PRELOAD_TYPE_SPINNING_ARC,
    LV_PRELOAD_TYPE_FILLSPIN_ARC,
};

typedef uint8_t lv_reload_type_t;
```

我们前面说过了,lv_reload 预加载控件具有 2 种旋转动画方式,具体区别如下

LV_PRELOAD_TYPE_SPINNING_ARC: 在旋转的过程中,小圆弧对应的角度保持不变

LV_PRELOAD_TYPE_FILLSPIN_ARC: 在旋转的过程中,小圆弧对应的角度由小变大,再由大变小来回交替

2.1.2 旋转方向数据类型

```
enum {
    LV_PRELOAD_DIR_FORWARD,//顺时针旋转
    LV_PRELOAD_DIR_BACKWARD,//逆时针旋转
};

typedef uint8_t lv_reload_dir_t;
```

2.1.3 样式数据类型

```
enum {
    LV_PRELOAD_STYLE_MAIN,
};

typedef uint8_t lv_reload_style_t;
```

lv_reload 预加载控件的样式就一种,但我们需要来了解一下此样式中某些字段的具体含义.

line 字段: 用来修饰小圆弧的,比如 main_style.line.color 可以用来修改小圆弧的颜色

body.border 字段: 用来修饰背景圆环的,比如 main_style.body.border.width 可以用来修改圆环的宽度

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_preload_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置小圆弧的角度

```
void lv_preload_set_arc_length(lv_obj_t * preload, lv_anim_value_t deg);
```

参数:

preload: 预加载对象

deg: 角度值,范围为[0,360]

2.2.3 设置旋转动画的速度

```
void lv_preload_set_spin_time(lv_obj_t * preload, uint16_t time);
```

参数:

preload: 预加载对象

time: 小圆弧旋转一圈所用的时间,单位为 ms,此值越大,那么旋转速度就越小

2.2.4 设置旋转动画的方式

```
void lv_preload_set_type(lv_obj_t * preload, lv_preload_type_t type);
```

参数:

preload: 预加载对象

type: 旋转动画的方式,有如下俩个可选值:

LV_PRELOAD_TYPE_SPINNING_ARC: 在旋转的过程中,小圆弧对应的角度保持不变

LV_PRELOAD_TYPE_FILLSPIN_ARC: 在旋转的过程中,小圆弧对应的角度由小变大,再由大变小来回交替

2.2.5 设置旋转方向

```
void lv_preload_set_dir(lv_obj_t * preload, lv_reload_dir_t dir);
```

参数:

preload: 预加载对象

dir: 旋转方向,有如下俩个可选值:

LV_PRELOAD_DIR_FORWARD: 顺时针旋转

LV_PRELOAD_DIR_BACKWARD: 逆时针旋转

2.2.6 设置样式

```
void lv_reload_set_style(lv_obj_t * preload, lv_reload_style_t type, const lv_style_t * style);
```

参数:

preload: 预加载对象

type: 设置哪一部分的样式,目前就 LV_PRELOAD_STYLE_MAIN 这一个可选值

style: 样式

2.2.7 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰预加载控件中的小圆弧和背景圆环,然后接着创建一个预加载对象,设置小圆弧对应的角度,设置旋转方向,设置旋转动画的方式等操作.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_preload_test.c 和 lv_preload_test.h 倘个文件,其中 lv_preload_test.c 文件的内容如下:

```
#include "lv_reload_test.h"
#include "lvgl.h"

lv_style_t main_style;
lv_obj_t *preload1;

//例程入口
void lv_reload_test_start()
{
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1. 创建样式
    lv_style_copy(&main_style, &lv_style_plain);

    //1.1 修饰小圆弧
    main_style.line.width = 10; //小圆弧的宽度
    main_style.line.color = LV_COLOR_GREEN; //小圆弧的颜色

    //1.2 修饰背景圆环
    main_style.body.border.color = LV_COLOR_GRAY; //背景圆环的颜色
    main_style.body.border.width = 10; //背景圆环的宽度
    main_style.body.padding.left = 0; //让小圆弧和背景圆环重合在一起
```

```
//2.创建 preload1 预加载对象
preload1 = lv_reload_create(scr,NULL);
lv_obj_set_size(preload1,120,120); //设置大小
lv_obj_align(preload1,NULL,LV_ALIGN_CENTER,0,0); //与屏幕居中对齐
lv_reload_set_style(preload1,LV_PRELOAD_STYLE_MAIN,&main_style); //设置样式
lv_reload_set_arc_length(preload1,45); //设置小圆弧对应的角度
lv_reload_set_dir(preload1,LV_PRELOAD_DIR_BACKWARD); //设置为逆时针旋转

//设置旋转动画的方式
lv_reload_set_type(preload1,LV_PRELOAD_TYPE_FILLSPIN_ARC);
lv_reload_set_spin_time(preload1,2000); //设置旋转的速度,转一圈所需要的时间

}
```

3.4 下载验证

把代码下载进去之后,我们可以看到绿色小圆弧在不断的旋转,而且其对应的角度值也在由小到大,由大到小地来回交替,如以下时刻所示:

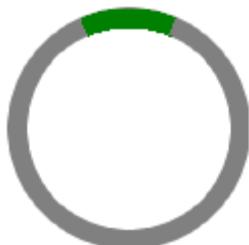


图 3.4.1 某时刻 1 的界面效果



图 3.4.2 某时刻 2 的界面效果

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_tabview 选项卡

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

lv_tabview 选项卡

1. 介绍

lv_tabview 选项卡是一个非常实用的控件,在页面稍微复杂或者页面结构层次较深的 GUI 项目中,使用 lv_tabview 选项卡控件来设计界面,会很得心应手,使我们的界面设计得到简化,简单来说,它是由任意多个内容页面构成的,每一个内容页面都会有一个与之对应的页面选择按钮,然后 lv_tabview 选项卡中还有一个页面指示器,用来指示当前是哪一个页面处于选中显示状态,如下图所示:

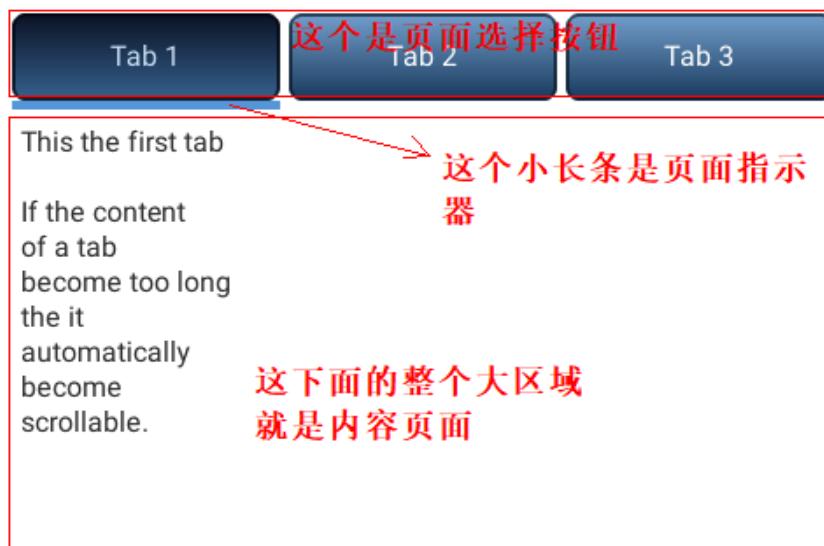


图 1.1 lv_tabview 选项卡组成

我们说了选项卡控件中的内容页面个数是不受限制的,它是通过 lv_tabview_add_tab(tabview, "Tab name")接口来添加一个内容页面的,然后你可以采用如下三种方式来切换内容页面:

- 1) 点击与之对应的页面选择按钮,但有一个前提条件,那就是你没有使用 lv_tabview_set_btns_hidden(tabview, true)这个接口把选择按钮给隐藏了
- 2) 用手指水平滑动页面,但有一个前提条件,那就是你没有使用 lv_tabview_set_sliding(tabview, false)这个接口把滑动功能给禁止了
- 3) 使用 lv_tabview_set_tab_act(tabview, id, LV_ANIM_ON/OFF)接口来选择显示指定 id 页面
同时附带指定页面切换时是否具有动画效果

页面选择按钮的位置默认是处于顶部的,其实它的位置是可以通过 lv_tabview_set_btns_pos(tabview, LV_TABVIEW_BTNS_POS_TOP/BOTTOM/LEFT/RIGHT) 接口来改变的,但是一般用的最多的就是 TOP 顶部和 BOTTOM 底部位置了.

最后来说一下它的事件,当你通过某种方式切换了页面,一旦页面切换完成之后,它就会给它的事件回调函数发送一个 LV_EVENT_VALUE_CHANGED 事件,注意,如果你是通过调用 lv_tabview_set_tab_act 接口来切换的页面,那么它是不会发送 LV_EVENT_VALUE_CHANGED 事件的.

2. lv_tabview 的 API 接口

2.1 主要数据类型

2.1.1 页面选择按钮栏位置数据类型

```
enum {
    LV_TABVIEW_BTNS_POS_TOP,      //顶部位置
    LV_TABVIEW_BTNS_POS_BOTTOM,   //底部位置
    LV_TABVIEW_BTNS_POS_LEFT,     //左边位置
    LV_TABVIEW_BTNS_POS_RIGHT    //右边位置
};

typedef uint8_t lv_tabview_btns_pos_t;
```

一般最常用的是 TOP 顶部和 BOTTOM 底部位置

2.1.2 选项卡样式数据类型

```
enum {
    LV_TABVIEW_STYLE_BG,          //选项卡的整个背景样式
    LV_TABVIEW_STYLE_INDIC,        //页面指示器的样式
    LV_TABVIEW_STYLE_BTN_BG,       //页面选择按钮栏的背景样式
    LV_TABVIEW_STYLE_BTN_REL,      //下面四种就是页面选择按钮的样式
    LV_TABVIEW_STYLE_BTN_PR,
    LV_TABVIEW_STYLE_BTN_TGL_REL,
    LV_TABVIEW_STYLE_BTN_TGL_PR,
};

typedef uint8_t lv_tabview_style_t;
```

虽然上面有 7 种样式,但是最下面的 4 种是用来修饰页面选择按钮的,至于具体的用法我们在 lv_btn 按钮章节中已经讲过了,是一模一样的,我们这里只介绍下面两种.

LV_TABVIEW_STYLE_BG: 选项卡的整个背景样式,使用样式中的 body 字段,默认值为 lv_style_plain

LV_TABVIEW_STYLE_INDIC: 页面指示器的样式,使用样式中的 body 字段,它的高度由 body.padding.inner 字段来指定,默认值为 lv_style_plain_color

LV_TABVIEW_STYLE_BTN_BG: 页面选择按钮栏的背景样式,使用样式中的 body 字段,默认值为 lv_style_transp

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_tabview_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 添加内容页面

```
lv_obj_t * lv_tabview_add_tab(lv_obj_t * tabview, const char * name);
```

参数:

tabview: 选项卡对象

name: 此内容页面的标题,会显示在对应的页面选择按钮上

返回值:

返回被添加的内容页面对象,其实此对象就是 lv_page 页面,所以拿到此对象之后,我们也可以用 lv_page 页面专有的 API 接口来操作它

2.2.3 清空某内容页面中的所有子对象

```
void lv_tabview_clean(lv_obj_t * obj);
```

参数:

obj: 由 lv_tabview_add_tab 接口返回出来的内容页面对象

其实此 API 接口的实现非常简单,它就是利用到了 lv_page 页面控件专有的 API 接口来实现的,如下图所示:

```
193 void lv_tabview_clean(lv_obj_t * obj)
194 {
195     lv_obj_t * scr1 = lv_page_get_scr1(obj);
196     lv_obj_clean(scr1);
197 }
```

图 2.2.3.1 此接口的实现原理

2.2.4 设置哪一个内容页面处于可见状态

```
void lv_tabview_set_tab_act(lv_obj_t * tabview, uint16_t id, lv_anim_enable_t anim);
```

参数:

tabview: 选项卡对象

id: 内容页面的 id, 第一个被添加的内容页面 id 为 0, 后面的页面依次递增 1

anim: 在页面切换时, 是否具有切换动画, 有如下两个可选值

LV_ANIM_OFF: 不开启动画

LV_ANIM_ON: 开启动画

2.2.5 是否使能手指滑页功能

```
void lv_tabview_set_sliding(lv_obj_t * tabview, bool en);
```

参数:

tabview: 选项卡对象

en: 是否使能手指滑页功能

如果不设置的话, 默认是使能的

2.2.6 设置切换动画的时长

```
void lv_tabview_set_anim_time(lv_obj_t * tabview, uint16_t anim_time);
```

参数:

tabview: 选项卡对象

anim_time: 动画时长, 单位 ms

2.2.7 设置样式

```
void lv_tabview_set_style(lv_obj_t * tabview, lv_tabview_style_t type, const lv_style_t * style);
```

参数:

tabview: 选项卡对象

type: 设置哪一部分的样式, 有 7 个可选值, 详情请看 2.1.2 选项卡样式数据类型

style: 样式

2.2.8 设置页面选择按钮栏的位置

```
void lv_tabview_set_btns_pos(lv_obj_t * tabview, lv_tabview_btns_pos_t btns_pos);
```

参数:

tabview: 选项卡对象

btns_pos: 页面选择按钮栏的位置, 有如下 4 个可选值:

LV_TABVIEW_BTNS_POS_TOP: 顶部位置

LV_TABVIEW_BTNS_POS_BOTTOM: 底部位置

LV_TABVIEW_BTNS_POS_LEFT: 左边位置, 基本用不到

LV_TABVIEW_BTNS_POS_RIGHT: 右边位置, 基本用不到

2.2.9 是否隐藏页面选择按钮

```
void lv_tabview_set_btns_hidden(lv_obj_t * tabview, bool en);
```

参数:

tabview: 选项卡对象

en: 是否隐藏页面选择按钮

如果不设置的话,默认是不隐藏的,如果你设置成了隐藏的话,那么页面选择按钮之前占据的空间位置会被释放出来,给内容页面使用

2.2.10 获取内容页面的总个数

```
uint16_t lv_tabview_get_tab_count(const lv_obj_t * tabview);
```

参数:

tabview: 选项卡对象

返回值:

返回 tabview 选项卡中的内容页面总个数

2.2.11 获取当前哪一个内容页面处于可见状态

```
uint16_t lv_tabview_get_tab_act(const lv_obj_t * tabview);
```

参数:

tabview: 选项卡对象

返回值:

返回当前可见内容页面的 id 值

2.2.12 获取某个内容页面对象

```
lv_obj_t * lv_tabview_get_tab(const lv_obj_t * tabview, uint16_t id);
```

参数:

tabview: 选项卡对象

id: 内容页面的 id

返回值:

返回 id 值对应的内容页面对象

拿到此对象之后,我们就可以用 lv_page 页面控件中专有的 API 接口来操作它了

2.2.13 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建 7 个自定义样式来修饰选项卡控件,然后创建一个选项卡控件,设置其大小,以及设置其事件回调函数,在事件回调函数中通过串口打印页面 id,接着给此选项卡添加三个内容页面,在内容页面 1 中,添加一个标签子对象和一个按钮子对象,在内容页面 2 和内容页面 3 中都只添加一个标签子对象,当按下 KEY0 按键时,调用 lv_tabview_set_tab_act 接口来切换内容页面,当按下 KEY1 按键时,来设置是否使能手指滑页功能,当按下 WKUP 按键时,来设置是否隐藏页面选择按钮栏.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0, KEY1, WKUP 按键
- 3) 串口

3.3 软件设计

在 GUI_APP 目录下创建 lv_tabview_test.c 和 lv_tabview_test.h 两个文件,其中 lv_tabview_test.c 文件的内容如下:

```
#include "lv_tabview_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_style_t bg_style;
lv_style_t indic_style;
lv_style_t btn_bg_style;
lv_style_t btn_rel_style;
lv_style_t btn_pr_style;
lv_style_t btn_tgl_rel_style;
lv_style_t btn_tgl_pr_style;
lv_obj_t * tabview1;
bool is_sliding = true;
```

```
bool is_btns_hidden = false;

//事件回调函数
void event_handler(lv_obj_t * obj,lv_event_t event)
{
    if(obj==tabview1&&event==LV_EVENT_VALUE_CHANGED)
    {
        uint16_t page_id = lv_tabview_get_tab_act(tabview1); //获取当前可见的页面 id
        printf("Current page id:%d\r\n",page_id); //串口打印当前页面的 id
    }
}

//例程入口
void lv_tabview_test_start()
{
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1. 创建样式
    //1.1 创建背景样式
    lv_style_copy(&bg_style,&lv_style_plain);
    bg_style.body.main_color = LV_COLOR_MAKE(49,49,49); //纯色背景
    bg_style.body.grad_color = bg_style.body.main_color;
    bg_style.body.border.color = LV_COLOR_MAKE(150,150,150); //边框颜色
    bg_style.body.border.width = 2; //边框宽度
    bg_style.text.color = LV_COLOR_WHITE;

    //1.2 创建页面指示器的样式
    lv_style_copy(&indic_style,&lv_style_plain_color);
    indic_style.body.main_color = LV_COLOR_MAKE(42,212,66); //指示器的颜色,绿色
    indic_style.body.grad_color = indic_style.body.main_color;
    indic_style.body.padding.inner = 3; //设置指示器的高度

    //1.3 创建页面选择按钮栏的背景样式
    //lv_style_transp_tight 样式中的 inner,left,top,right,bottom 等内间距值都为 0,这是为了
    //让页面选择按钮能够紧挨在一起
    lv_style_copy(&btn_bg_style,&lv_style_transp_tight);

    //1.4 创建按钮正常态下的松手样式
    lv_style_copy(&btn_rel_style,&lv_style_plain_color);
    btn_rel_style.body.main_color = LV_COLOR_MAKE(98,98,98);
    btn_rel_style.body.grad_color = btn_rel_style.body.main_color;
    btn_rel_style.body.border.color = LV_COLOR_MAKE(150,150,150); //边框颜色
    btn_rel_style.body.border.width = 1;
```

```
btn_rel_style.text.color = LV_COLOR_WHITE;//字体颜色

//1.5 创建按钮正常态下的按下样式
lv_style_copy(&btn_pr_style,&btn_rel_style);
btn_pr_style.body.main_color = LV_COLOR_GRAY;
btn_pr_style.body.grad_color = btn_pr_style.body.main_color;

//1.6 创建按钮切换态下的松手样式
lv_style_copy(&btn_tgl_rel_style,&btn_rel_style);
btn_tgl_rel_style.body.main_color = bg_style.body.main_color;//和主背景颜色一致
btn_tgl_rel_style.body.grad_color = btn_tgl_rel_style.body.main_color;

//1.7 创建按钮切换态下的按下样式
//保持和 btn_tgl_rel_style 一样就行了
lv_style_copy(&btn_tgl_pr_style,&btn_tgl_rel_style);

//2.创建选项卡
tabview1 = lv_tabview_create(scr,NULL);
//设置选项卡的大小,比屏幕小 16 像素
lv_obj_set_size(tabview1,lv_obj_get_width(scr)-20,lv_obj_get_height(scr)-20);
lv_obj_align(tabview1,NULL,LV_ALIGN_CENTER,0,0);//与屏幕居中对齐
lv_obj_set_event_cb(tabview1,event_handler);//设置事件回调函数
//设置页面选择按钮栏位于顶部
lv_tabview_set_btns_pos(tabview1,LV_TABVIEW_BTNS_POS_TOP);
lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_BG,&bg_style);//设置背景样式
//设置页面指示器的样式
lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_INDIC,&indic_style);
//设置页面选择按钮栏的背景样式
lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_BTN_BG,&btn_bg_style);
//设置按钮正常态下的松手样式
lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_BTN_REL,&btn_rel_style);
//设置按钮正常态下的按下样式
lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_BTN_PR,&btn_pr_style);

lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_BTN_TGL_REL,&btn_tgl_rel_style);//设置按钮切换态下的松手样式

lv_tabview_set_style(tabview1,LV_TABVIEW_STYLE_BTN_TGL_PR,&btn_tgl_pr_style);//设置按钮切换态下的按下样式
//2.1 添加 tab1 内容页面
lv_obj_t *tab1_page = lv_tabview_add_tab(tabview1, LV_SYMBOL_WIFI " Tab1");
//往内容页面 1 中添加标签子对象
lv_obj_t *tmp_obj = lv_label_create(tab1_page,NULL);
lv_label_set_text(tmp_obj, "\nThis is the tab1 page\nIf the content\nof a tab\nbecome too
```

```
long\nthe it \nautomatically\nbecome\nscrollable.");\n    //往内容页面 1 中添加标签子对象\n    tmp_obj = lv_btn_create(tab1_page,NULL);\n    lv_obj_set_pos(tmp_obj,50,300);\n\n    //2.2 添加 tab2 内容页面\n    lv_obj_t *tab2_page = lv_tabview_add_tab(tabview1, LV_SYMBOL_AUDIO" Tab2");\n    //往内容页面 2 中添加标签子对象\n    tmp_obj = lv_label_create(tab2_page,NULL);\n    lv_label_set_text(tmp_obj,"\\nThis is the tab2 page\\nZheng dian yuan zi");\n\n    //2.3 添加 tab3 内容页面\n    lv_obj_t *tab3_page = lv_tabview_add_tab(tabview1, LV_SYMBOL_BELL" Tab3");\n    //往内容页面 3 中添加标签子对象\n    tmp_obj = lv_label_create(tab3_page,NULL);\n    lv_label_set_text(tmp_obj,"\\nThis is the tab3 page\\nXiong jia yu");\n}\n\n//按键处理\nvoid key_handler()\n{\n    static u16 page_id = 0;\n\n    u8 key = KEY_Scan(0);\n\n    if(key==KEY0_PRES)\n    {\n        //使用接口来切换内容页面\n        page_id++;\n        if(page_id==3)\n            page_id = 0;\n        lv_tabview_set_tab_act(tabview1,page_id,LV_ANIM_ON); //带有切换动画效果\n    }else if(key==KEY1_PRES)\n    {\n        //是否使能手指滑页功能\n        is_sliding = !is_sliding;\n        lv_tabview_set_sliding(tabview1,is_sliding);\n    }else if(key==WKUP_PRES)\n    {\n        //是否隐藏页面选择按钮栏\n        is_btns_hidden = !is_btns_hidden;\n        lv_tabview_set_btns_hidden(tabview1,is_btns_hidden);\n    }\n}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的界面效果:

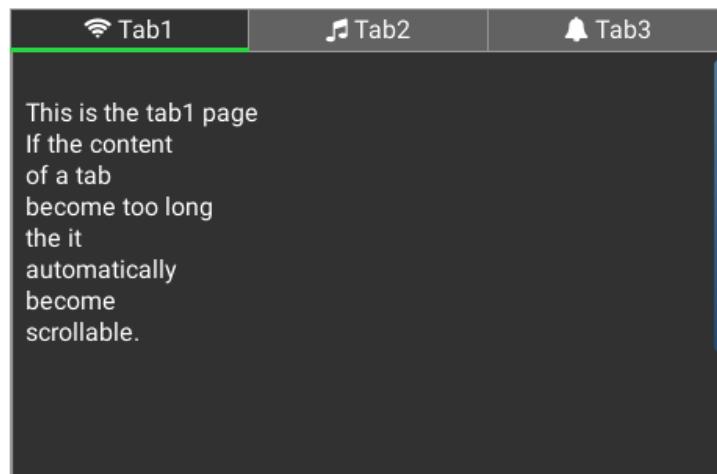


图 3.4.1 初始界面效果

然后我们可以通过点击顶部的页面选择按钮,或者通过手指滑动页面,或者按 KEY0 按键来切换到 Tab2 页面,如下图所示:

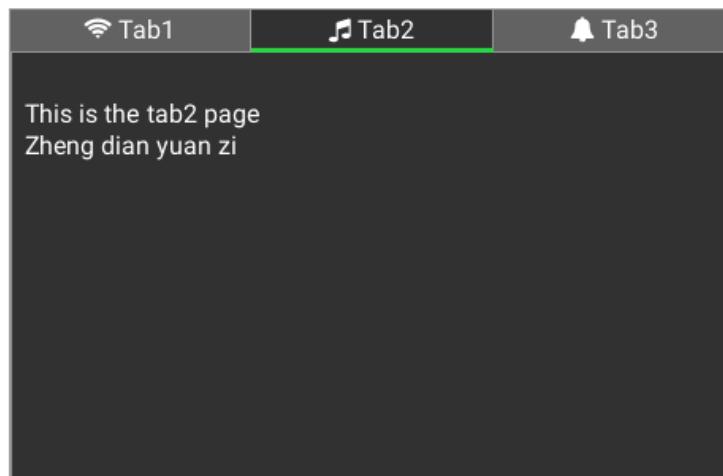


图 3.4.2 页面切换效果

然后我们还可以按下 WKUP 按键来隐藏顶部的页面选择按钮栏,如下图所示:

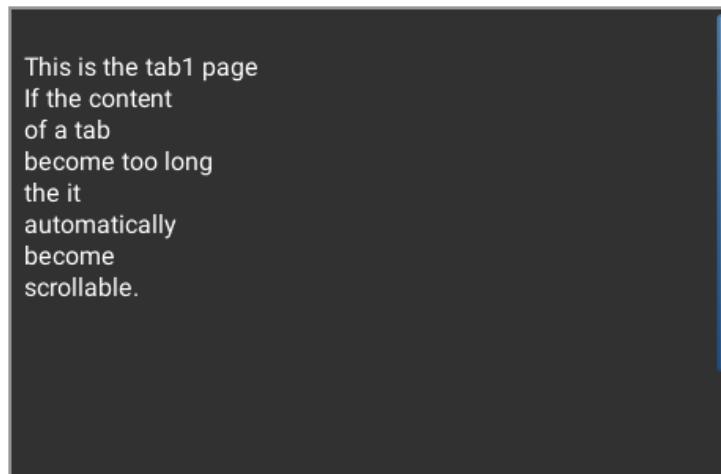


图 3.4.3 隐藏页面选择按钮栏后的效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_ta 文本域

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

lv_ta 文本域

1. 介绍

lv_ta 文本域控件其实是我们通常所看到的文本输入框,配合我们后面将要讲到的 lv_kb 键盘控件可以实现输入某些内容,lv_ta 文本域控件是由一个 lv_page 页面控件,一个 lv_label 标签控件,以及还有一个光标构成的,利用 lv_page 页面控件的特性,lv_ta 文本域控件可以实现垂直和水平滚动,利用 lv_label 标签控件的特性,lv_ta 文本域控件可以实现显示文本的功能,而光标的作用就是用于标记当前所在的文本输入位置.

lv_ta 文本域控件具有 Placeholder 提示文字的功能,可以通过 lv_ta_set_placeholder_text(ta, "Placeholder text")接口来设置想要的提示文字,当 lv_ta 文本域控件中没有输入任何文本内容时,lv_ta 文本域控件就会把 Placeholder 提示文字给显示出来,可以起到提醒用户输入,或者用来自说明此文本域用途等功能,当文本域中一旦含有文本内容时,不管是用户输入的,还是自己通过 API 接口添加的,此 Placeholder 提示文字都会立即消失.

文本域中是可以显示文本内容的,那我们怎么来给其添加文本内容呢?一是通过 API 接口来直接添加,二是通过 lv_kb 键盘来间接输入,当然了 lv_kb 键盘输入的内部实现原理也是利用到了 API 接口,在文本域控件中,一共有如下三个 API 接口可以实现添加文本:

- 1) lv_ta_set_text(ta, "xiong jia yu") 这是直接设置文本内容
- 2) lv_ta_add_char(ta, 'c') 这是添加单个字符
- 3) lv_ta_add_text(ta, "hello world") 这是添加字符串

添加的文本内容会放到或插入到当前光标所在的位置,而光标的位置是可以通过用户点击和 API 接口来改变的,如果是想通过用户点击来改变光标的位置,那么是有一个前提的,那就是必须得调用 lv_ta_set_cursor_click_pos(ta, true)接口来先使能此功能,不过文本域控件默认是使能了此功能的,如果是想通过 API 接口来改变光标的位置,总共有如下 5 种方式:

- 1) lv_ta_set_cursor_pos(ta, pos) 设置任意的光标位置,pos 从 0 开始
- 2) lv_ta_cursor_right(ta) 设置光标向右移动一步
- 3) lv_ta_cursor_left(ta) 设置光标向左移动一步
- 4) lv_ta_cursor_up(ta) 设置光标向上移动一步,文本域中有多行文本时才起作用
- 5) lv_ta_cursor_down(ta) 设置光标向下移动一步,文本域中有多行文本时才起作用

文本域控件有添加文本内容的方式,当然也会有删除文本内容的方式,删除的 API 接口有如下俩个:

- 1) lv_ta_del_char(ta) 删除当前光标左侧的一个字符
- 2) lv_ta_del_char_forward(ta) 删除当前光标右侧的一个字符

在文本域控件中,光标的类型总共有 5 种,分别为无,垂直线,填充矩形块,矩形边框,下划线,它是通过 lv_ta_set_cursor_type(ta, LV_CURSOR_...)接口来设置的.

文本域控件默认情况下是多行模式的,如果你想使能文本域控件的单行模式,可以通过 lv_ta_set_one_line(ta, true)接口来设置,变成单行模式之后,\n换行符和 word wrap 自动换行功能将会被忽略,和 lv_label 标签一样,文本域控件也具有文本对齐的功能,可以通过 lv_ta_set_text_align(ta, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)接口来设置,如果文本域控件是处于单行模式,而且文本是左对齐的,那么当文本内容过长超过文本域的宽度时,文本内容是可以被水平滚动条滑动的.

littleVGL 中的 lv_ta 文本域控件是非常强大的,在小细节处理上也表现得非常出色,比如

有时候我们可能会有这样的一些需求场景,我们希望文本域只能显示数字或者某些英文字母,同时还想限定其最大长度不超过指定值,希望密码输入框有保护显示功能等等,而 lv_ta 文本域控件可以统统满足你,通过 `lv_ta_set_accepted_chars(ta, list)` 接口,你可以设置文本域所能接受的字符列表,比如 `const char * list = "0123456789";` 那么此时文本域就只能显示数字了,不在此列表中的字符将统统不能被显示出来,通过 `lv_ta_set_pwd_mode(ta, true)` 接口,你可以使能文本域的密码输入保护功能,使能之后,它将会用*号来代替每个字符进行显示,通过 `lv_ta_set_max_length(ta, max_char_num)` 接口,你可以指定文本域所能显示的最大字符数,然后这里有一点需要注意的是,当文本域显示的字符数超过 20k 时,文本域控件的渲染速度可能会变得很慢,那么你可以通过将 `lv_conf.h` 中的 `LV_LABEL_LONG_TXT_HINT` 宏设为 1 来加速渲染过程,这个操作会额外增加 12 字节的内存消耗.

最后我们来说一下文本域控件的事件,除了按下,松手等常用事件外,还有 `LV_EVENT_INSERT` 和 `LV_EVENT_VALUE_CHANGED` 两个特殊事件与之相关,当有新的文本内容放入到或者说是插入到文本域中时,`LV_EVENT_INSERT` 事件将会被触发,这个新的文本内容将会被作为事件自定义参数给传递过去,同时请确保此新文本内容在事件回调函数被调用之前不能被释放,当文本域中的文本内容发生改变时,`LV_EVENT_VALUE_CHANGED` 事件将会被触发.

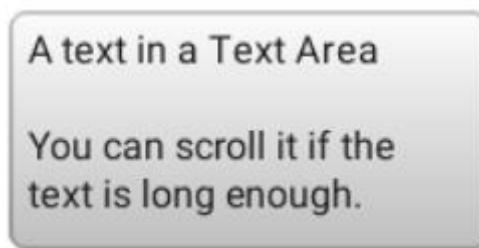


图 1.1 lv_ta 文本域控件的默认外观

2. lv_ta 的 API 接口

2.1 主要数据类型

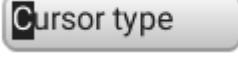
2.1.1 光标样式数据类型

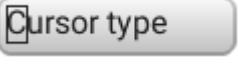
```
enum {
    LV_CURSOR_NONE,
    LV_CURSOR_LINE,
    LV_CURSOR_BLOCK,
    LV_CURSOR_OUTLINE,
    LV_CURSOR_UNDERLINE,
    LV_CURSOR_HIDDEN = 0x08,
};

typedef uint8_t lv_cursor_type_t;
```

LV_CURSOR_NONE: 无光标显示

LV_CURSOR_LINE: 垂直线光标,如  所示

LV_CURSOR_BLOCK: 填充矩形块光标,如  所示

LV_CURSOR_OUTLINE: 矩形边框光标,如  所示

LV_CURSOR_UNDERLINE: 下划线光标,如  所示

LV_CURSOR_HIDDEN: 隐藏光标,和 LV_CURSOR_NONE 的作用差不多,不过

LV_CURSOR_HIDDEN: 它可以和上面的其他值进行位或操作

2.1.2 文本域样式数据类型

```
enum {
    LV_TA_STYLE_BG,
    LV_TA_STYLE_SB,
    LV_TA_STYLE_CURSOR,
    LV_TA_STYLE_EDGE_FLASH,
    LV_TA_STYLE_PLACEHOLDER,
};

typedef uint8_t lv_ta_style_t;
```

LV_TA_STYLE_BG: 用此样式中的 body 字段来修饰文本域的背景,用此样式中的 text 字段来修饰其文本内容,默认值为 lv_style_pretty.

LV_TA_STYLE_SB: 用来修饰水平和垂直滚动条的,使用样式中的 body 字段,默认值为 lv_style_pretty_color,其中 body.padding.right 是用来控制垂直滚动条与页面右边缘的距离,当此值为正数时,是在页面内部,当为负值时,是在页面外部,而 body.padding.bottom 是用来控制水平滚动条与页面底边缘的距离,当此值为正数时,是在页面内部,当为负值时,是在页面外部,而 body.padding.inner 是用来设置滚动条的宽度.

LV_TA_STYLE_CURSOR: 用来修饰光标的,如果不设置的话,则系统会根据文本的字体和颜色来自动设置光标的样式,我们一般不设置.

LV_TA_STYLE_EDGE_FLASH: 用来修饰边缘半圆弧动画效果的,一般只用到里面的 body.main_color, body.grad_color, body.opa 等样式字段,对于上边缘只用 grad_color 来设置半圆弧的颜色,对于下边缘只用 main_color 来设置半圆弧的颜色,而对于左和右边缘,main_color 和 grad_color 都会用到.

LV_TA_STYLE_PLACEHOLDER: 用来修饰 Placeholder 提示文字的,使用此样式中的 text 字段.

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_ta_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 添加字符内容

```
void lv_ta_add_char(lv_obj_t * ta, uint32_t c);
```

参数:

ta: 文本域对象

c: 字符

在当前光标所在的位置处添加一个字符内容

2.2.3 添加字符串内容

```
void lv_ta_add_text(lv_obj_t * ta, const char * txt);
```

参数:

ta: 文本域对象

txt: 字符串文本内容

在当前光标所在的位置处添加字符串文本内容

2.2.4 删除光标左侧的一个字符

```
void lv_ta_del_char(lv_obj_t * ta);
```

参数:

ta: 文本域对象

2.2.5 删除光标右侧的一个字符

```
void lv_ta_del_char_forward(lv_obj_t * ta);
```

参数:

ta: 文本域对象

2.2.6 设置文本内容

```
void lv_ta_set_text(lv_obj_t * ta, const char * txt);
```

参数:

ta: 文本域对象

txt: 文本内容

这是直接设置文本域控件的文本内容

2.2.7 设置 placeholder 提示文字

```
void lv_ta_set_placeholder_text(lv_obj_t * ta, const char * txt);
```

参数:

ta: 文本域对象

txt: 提示文字

当 lv_ta 文本域控件中没有输入任何文本内容时,lv_ta 文本域控件就会把 placeholder 提示文字给显示出来,可以起到提醒用户输入,或者用来说说明此文本域用途等功能,当文本域中一旦含有文本内容时,不管是用户输入的,还是自己通过 API 接口添加的,此 placeholder 提示文字都会立即消失

2.2.8 设置光标的任意位置

```
void lv_ta_set_cursor_pos(lv_obj_t * ta, int16_t pos);
```

参数:

ta: 文本域对象

pos: 光标的位置,从 0 开始,如果为负数的话,则代表以文本末尾为起点开始算,当为 LV_TA_CURSOR_LAST 值时,则是直接将光标定位到末尾

2.2.9 设置光标的类型

```
void lv_ta_set_cursor_type(lv_obj_t * ta, lv_cursor_type_t cur_type);
```

参数:

ta: 文本域对象

cur_type: 光标的类型,有如下 6 个可选值

LV_CURSOR_NONE: 不显示光标

LV_CURSOR_LINE: 垂直线光标

LV_CURSOR_BLOCK: 填充矩形块光标

LV_CURSOR_OUTLINE: 矩形边框光标

LV_CURSOR_UNDERLINE: 下划线光标

LV_CURSOR_HIDDEN: 隐藏光标,和 LV_CURSOR_NONE 的作用差不多

2.2.10 是否使能用户点击改变光标位置

```
void lv_ta_set_cursor_click_pos(lv_obj_t * ta, bool en);
```

参数:

ta: 文本域对象

en: 是否使能

默认是使能的,当使能之后,用户可以点击文本域控件的某处,光标也会相应地跳到手点击的地方

2.2.11 是否使能密码保护模式

```
void lv_ta_set_pwd_mode(lv_obj_t * ta, bool en);
```

参数:

ta: 文本域对象

en: 是否使能

当使能密码保护之后,用户输入的每一个字符在短暂时间显示之后,会被转换成*号来进行最终显示,如下图所示:

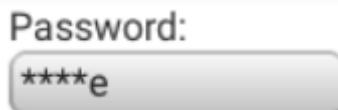


图 2.2.11.1 密码保护效果

注:上面说到的短暂时间是可以通过 lv_ta_set_pwd_show_time 接口来设置的

2.2.12 是否使能单行模式

```
void lv_ta_set_one_line(lv_obj_t * ta, bool en);
```

参数:

ta: 文本域对象

en: 是否使能

使能单行模式之后,\n 换行符和自动换行(word wrap)功能将会被忽略

2.2.13 设置文本对齐方式

```
void lv_ta_set_text_align(lv_obj_t * ta, lv_label_align_t align);
```

参数:

ta: 文本域对象

align: 文本对齐方式

2.2.14 设置文本域可接受的字符列表

```
void lv_ta_set_accepted_chars(lv_obj_t * ta, const char * list);
```

参数:

ta: 文本域对象

list: 字符列表,比如 const char * list = “0123456789”;那么此时文本域就只能显示数字了,在此列表中的字符将统统不能被显示出来

2.2.15 设置文本域能显示的最大字符数

```
void lv_ta_set_max_length(lv_obj_t * ta, uint16_t num);
```

参数:

ta: 文本域对象

num: 能显示的最大字符数

这里有一点需要注意的是,当文本域实际显示的字符数超过 20k 时,文本域控件的渲染速

度可能会变得很慢,那么你可以通过将 lv_conf.h 中的 LV_LABEL_LONG_TXT_HINT 宏设为 1 来加速渲染过程,这个操作会额外增加 12 字节的内存消耗

2.2.16 用新文本内容替换掉正打算插入的文本内容

```
void lv_ta_set_insert_replace(lv_obj_t * ta, const char * txt);
```

参数:

ta: 文本域对象

txt: 新文本内容

此接口必须得在 LV_EVENT_INSERT 事件回调中调用才能看到效果,主要是用来对用户输入的信息进行过滤或者格式控制的,可能这样理解起来比较费劲,请看如下例子(只给出示意代码):

```
lv_obj_t * ta1;
//事件回调函数
void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event==LV_EVENT_INSERT) //当有文本插入时,就会触发此事件
    {
        const char * inserted_txt = lv_event_get_data(); //获取到被插入的文本

        //如果用户插入的文本是 date 字符串的话,那么我们对其进行过滤或者格式控制
        //我这里就把它替换成一个固定的日期时间,当然了,你也可以做其他的操作
        if(strcmp(inserted_txt, "date") == 0)
        {
            //用"2019/12/26 23:45"字符串替换掉正准备插入的" date " 字符串
            lv_ta_set_insert_replace(ta1, "2019/12/26 23:45");
        }
    }
}

ta1 = lv_ta_create(lv_scr_act(), NULL); //创建文本域
lv_ta_set_text(ta1, "now:"); //直接设置文本内容

//模拟用户插入 date 字符串的操作,会触发 LV_EVENT_INSERT 事件
lv_ta_add_text(ta1, "date");
```

最后我们可以看到如下图所示的效果:



now:2019/12/26 23:45

图 2.2.16.1 效果图

从上图的效果中我们可以看到,插入的 date 字符串并不会被显示出来,这是因为它已经被 lv_ta_set_insert_replace 所指定的新文本内容给替换掉了

2.2.17 设置滚动条的模式

```
lv_ta_set_sb_mode(lv_obj_t * ta, lv_sb_mode_t mode);
```

参数:

ta: 文本域对象

mode: 滚动条的模式

此 API 接口的使用方法和 lv_page 页面控件中的 lv_page_set_sb_mode 接口的使用方法是一模一样的,详情请看”lv_page 页面”章节

2.2.18 是否使能边缘半圆弧动画效果

```
static inline void lv_ta_set_edge_flash(lv_obj_t * ta, bool en);
```

参数:

ta: 文本域对象

en: 是否使能

此 API 接口的使用方法和 lv_page 页面控件中的 lv_page_set_edge_flash 接口的使用方法是一模一样的,详情请看”lv_page 页面”章节

2.2.19 设置样式

```
void lv_ta_set_style(lv_obj_t * ta, lv_ta_style_t type, const lv_style_t * style);
```

参数:

ta: 文本域对象

type: 设置那一部分的样式,目前有如下 5 个可选值

LV_TA_STYLE_BG: 修饰文本域的背景和文本内容

LV_TA_STYLE_SB: 修饰水平和垂直滚动条的

LV_TA_STYLE_CURSOR: 修饰光标的,我们一般不设置,系统会自动设置的

LV_TA_STYLE_EDGE_FLASH: 修饰边缘半圆弧动画效果的

LV_TA_STYLE_PLACEHOLDER: 修饰 Placeholder 提示文字的

style: 样式

2.2.20 是否使能文本选中功能

```
void lv_ta_set_text_sel(lv_obj_t * ta, bool en);
```

参数:

ta: 文本域对象

en: 是否使能

使用此 API 接口的前提是先将 lv_conf.h 头文件中的 LV_LABEL_TEXT_SEL 宏给置 1,然后如果将 en 设为 true 使能之后,我们就可以通过手指触摸选中某个区域的文本了,效果如下

图所示：



图 2.2.20.1 文本选中效果

扩展:可以采用如下方法来获取被选中的文本内容

```
lv_obj_t * ta_label = lv_obj_get_ext_attr(ta1)->label;//获取 ta1 文本域中的标签对象  
uint16_t start_pos = lv_label_get_text_sel_start(ta_label); //获取被选中文本的起始位置  
uint16_t end_pos = lv_label_get_text_sel_end(ta_label); //获取被选中文本的终止位置  
拿到了 start_pos 和 end_pos,也就间接地拿到了被选中的文本内容
```

2.2.21 判断文本是否有被选中

```
bool lv_ta_text_is_selected(const lv_obj_t * ta);
```

参数:

ta: 文本域对象

返回值:

如果有文本被选中了,返回 false,没有被选中的话,返回 true

使用此 API 接口的前提是先将 lv_conf.h 头文件中的 LV_LABEL_TEXT_SEL 宏给置 1

2.2.22 清除文本选中状态

```
void lv_ta_clear_selection(lv_obj_t * ta);
```

参数:

ta: 文本域对象

使用此 API 接口的前提是先将 lv_conf.h 头文件中的 LV_LABEL_TEXT_SEL 宏给置 1

2.2.23 将光标移动一步

```
void lv_ta_cursor_right(lv_obj_t * ta); //光标向右移动一步  
void lv_ta_cursor_left(lv_obj_t * ta); //光标向左移动一步  
void lv_ta_cursor_down(lv_obj_t * ta); //光标向下移动一步,文本域中有多行文本时才起作用  
void lv_ta_cursor_up(lv_obj_t * ta); //光标向上移动一步,文本域中有多行文本时才起作用
```

参数:

ta: 文本域对象

2.2.24 设置密码保护时的短暂显示时间

```
void lv_ta_set_pwd_show_time(lv_obj_t * ta, uint16_t time);
```

参数:

ta: 文本域对象

time: 单位 ms

在密码保护显示模式下,用户输入的每个字符经过 time 短暂时间显示之后,会被系统转化为*号进行最终的显示,如果 time 为 0 的话,则代表立即被转化成*号

2.2.25 设置光标的闪烁间隔

```
void lv_ta_set_cursor_blink_time(lv_obj_t * ta, uint16_t time);
```

参数:

ta: 文本域对象

time: 闪烁间隔,单位为 ms

2.2.26 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个文本域对象,设置其大小,与屏幕居中对齐,设置其光标类型,文本内容,注册事件回调函数等等,当按下 KEY0 按键时,往光标所在的位置处添加[add txt]文本,当按下 KEY1 按键时,删除光标左侧的一个字符,当按下 WKUP 按键时,往光标所在的位置处添加 d 字符,这个主要是配合 LV_EVENT_INSERT 事件,用来演示插入替换功能的

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0, KEY1, WKUP 按键
- 3) 串口

3.3 软件设计

在 GUI_APP 目录下创建 lv_ta_test.c 和 lv_ta_test.h 俩个文件,其中 lv_ta_test.c 文件的内容如下:

```
#include "lv_ta_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>
#include <string.h>

lv_obj_t * ta1;

//事件回调函数
void event_handler(lv_obj_t *obj,lv_event_t event)
{
    if(event==LV_EVENT_VALUE_CHANGED)//文本内容改变时,就会触发此事件
    {
        //把当前的文本打印出来
        printf("LV_EVENT_VALUE_CHANGED:%s\r\n",lv_ta_get_text(ta1));
    }else if(event==LV_EVENT_INSERT)//有文本插入时,就会触发此事件
    {
        const char * inserted_txt = lv_event_get_data();//获取被插入的文本
```

```
printf("LV_EVENT_INSERT:%s\r\n",inserted_txt);//把插入的文本打印出来
//如果插入的是 d 字符串的话,就进行过滤或者格式控制
if(strcmp(inserted_txt,"d")==0)
{
    //用"2019/12/26"替换掉正准备插入的"d"
    lv_ta_set_insert_replace(ta1,"2019/12/26");
}
}

//例程入口
void lv_ta_test_start()
{
    lv_obj_t *scr = lv_scr_act();//获取当前活跃的屏幕对象

    //1.创建文本域对象
    ta1 = lv_ta_create(scr,NULL);
    lv_obj_set_size(ta1,200,100);//设置文本域的大小
    lv_obj_align(ta1,NULL,LV_ALIGN_CENTER,0,0);//与屏幕居中对齐
    lv_ta_set_cursor_type(ta1,LV_CURSOR_LINE);//设置光标的类型为竖直线

    //设置提示文字,当文本内容为空时,才会显示出来
    lv_ta_set_placeholder_text(ta1,"Please input");
    lv_ta_set_text(ta1,"This is a text area!"); //设置文本内容
    lv_ta_set_pwd_mode(ta1,false); //不使能密码保护模式,默认就是不使能的
    lv_ta_set_one_line(ta1,false); //不使能单行模式,默认就是不使能的
    lv_ta_set_text_sel(ta1,true); //使能文本选中功能
    lv_ta_set_cursor_click_pos(ta1,true); //使能用户点击改变光标位置,默认就是使能的

    //设置文本居左对齐,默认就是居左对齐
    lv_ta_set_text_align(ta1,LV_LABEL_ALIGN_LEFT);
    lv_obj_set_event_cb(ta1,event_handler); //设置事件回调函数
}

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

    if(key==KEY0_PRES)
    {
        //往光标所在的位置处添加文本
    }
}
```

```
lv_ta_add_text(ta1,"[add txt]");
}else if(key==KEY1_PRES)
{
    //往光标所在的位置处添加字符,用来演示插入替换功能
    lv_ta_add_char(ta1,'d');
}else if(key==WKUP_PRES)
{
    //删除光标左侧的一个字符
    lv_ta_del_char(ta1);
}
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

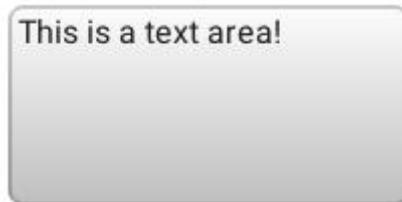


图 3.4.1 初始界面效果

然后我们可以用手触摸来选中文本,如下图所示:

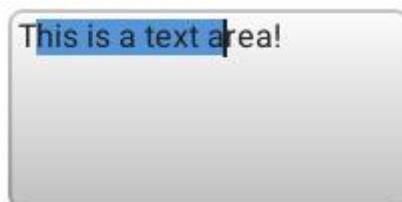


图 3.4.2 文本选中效果

接着我们可以按一下 KEY0 按键,往光标所在的位置处添加[add txt]文本,如下图所示:

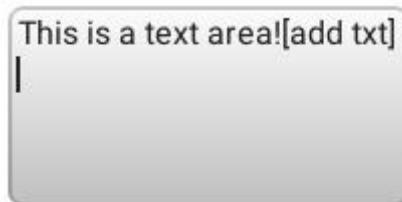


图 3.4.3 添加或者插入文本效果

再接着我们可以多按几下 KEY1 按键,删除掉光标左侧的几个字符,如下图所示:

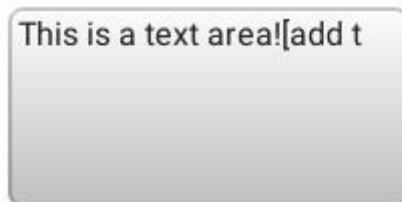


图 3.4.4 删除字符效果

最后我们可以按一下 WKUP 按键,来演示插入替换功能,插入的 d 字符会被替换成 2019/12/26 这个字符串的

This is a text area![add
t2019/12/26

图 3.4.5 插入替换功能效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_kb 键盘

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

lv_kb 键盘

1. 介绍

在上一章节中,我们学会了 lv_ta 文本域控件的使用,这一节我们趁热打铁,来学习它的另外一个小伙伴 lv_kb 键盘,从它的控件实现原理上来讲,lv_kb 键盘其实就是一个复杂而又特殊的 lv_btmn 矩阵按钮对象,而 lv_btmn 矩阵按钮我们在前面的章节中也学习过了,所以借助前面的知识,我们应该能更好的理解 lv_kb 键盘.

lv_kb 键盘有 2 种模式,一种是 LV_KB_MODE_TEXT 文本键盘,另一种是 LV_KB_MODE_NUM 数字键盘,其中文本键盘复杂一些,它里面包含了小写,大写,符号三类键盘,文本键盘中的三类键盘是通过如下方式来切换的:

- 1) 点击  键切换到大写键盘
- 2) 点击  键切换到小写键盘
- 3) 点击  键切换到符号

为了有个感性认识,请直接看如下外观图:



图 1.1 文本键盘-小写



图 1.2 文本键盘-大写



图 1.3 文本键盘-符号

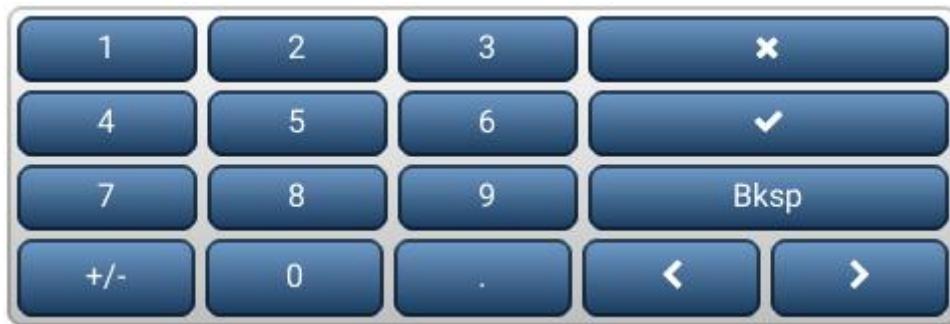


图 1.4 数字键盘

虽然文本键盘-符号中也含有 0123456789 数字字符,但对比数字键盘而言,数字键盘表现的更专一,它只能输入数字,对于创建出来的键盘,如果你不指定它的坐标位置和大小的话,那么它是一个默认的坐标位置和大小的,默认是与父对象底部居中对齐的,默认宽度等于父对象的可适应宽度(即除掉左右内边距后的宽度),默认高度等于父对象的可适应高度(即除掉上下内边距后的高度)的一半,一个 lv_kb 键盘对象可以绑定一个 lv_ta 文本域对象,这是通过 lv_kb_set_ta(kb, ta) 接口来完成绑定的,一旦绑定之后,用户在键盘上按下的字符会相应的输入到绑定的文本域控件中,于此同时键盘对象也可以接管对文本域控件上光标的控制,接管之后的好处就是一致性体验更好,具体表现是当键盘绑定新的文本域控件时,键盘会将之前绑定的文本域控件上的光标给隐藏掉,而让新绑定的文本域控件上显示光标,这个接管光标的操作是通过 lv_kb_set_cursor_manage(kb, true) 接口来完成的,默认是不接管的.

对于 lv_kb 键盘而言,它到底具有哪些字符按键,以及排版顺序如何,这些东西在键盘选定模式之后都是固定的,那要是我们想自定义键盘咋办?比如添加或者删除一个按键,我可以告诉大家这是能实现的,因为 lv_kb 键盘给我们留出了 lv_kb_set_map(kb, map) 和 lv_kb_set_ctrl_map(kb, ctrl_map) 这两个 API 接口,而这两个接口的本质其实就是 lv_btmn 矩阵按钮章节中的 lv_btmn_set_map(kb, map) 和 lv_btmn_set_ctrl_map(kb, ctrl_map) 接口,讲到这里,大家是不是有点恍然大悟了呢?如果叫大家自己去实现,可能还是有点难度,但是我们可以模仿官方的 lv_kb.c 源码来实现呀,大家抄总会吧!在本章的例程设计中,我会给大家演示如何实现自定义键盘,在 lv_kb 键盘中,有些按键是具有特殊功能的,此特殊功能按键上的文本标题是不能随便改动的,如果我们自定义的键盘上也想具有此特殊功能的按键,那么我们也得必须遵守如下特殊按键原则:

LV_SYMBOL_OK: Apply 确认, 对应键盘上的 键, 点击会触发 LV_EVENT_APPLY 事件

 **LV_SYMBOL_CLOSE:** Close 关闭, 对应键盘上的  键, 点击会触发 LV_EVENT_CANCEL 事件

LV_SYMBOL_LEFT: 将光标向左移动一步

LV_SYMBOL_RIGHT: 将光标向右移动一步

“ABC”：切换到大写键盘

“abc”：切换到小写键盘

“Enter”：换行

“Bkps”：删除光标左侧的一个字符

最后我们来讲一下 lv_kb 键盘的事件, 常用的按下, 松手事件就不讲了, 主要说跟它相关的三个特殊事件:

LV_EVENT_VALUE_CHANGED: 当键盘上的按键被按下或者被重复按下时, 会触发此事件, 于此同时, 被按下的按键 id 会作为事件自定义参数给传递过去

 **LV_EVENT_APPLY:** 当键盘上的  键被点击时会触发此事件

 **LV_EVENT_CANCEL:** 当键盘上的  键被点击时会触发此事件

然后这里有一点需要注意的是 lv_kb 键盘控件自身是有一个名为 lv_kb_def_event_cb 的默认事件回调函数, 它处理按键按下, 按键值改变, 管理绑定的文本域等操作, 当然了你完全可以用你自定义的事件回调函数来覆盖掉它, 但是为了减少一些相同事务的处理, 我们可以在我们自定义的事件回调函数里的前面调用一下默认的 lv_kb_def_event_cb 事件回调函数.

2. lv_kb 的 API 接口

2.1 主要数据类型

2.1.1 键盘模式数据类型

```
enum {
    LV_KB_MODE_TEXT,
    LV_KB_MODE_NUM,
};

typedef uint8_t lv_kb_mode_t;
```

LV_KB_MODE_TEXT: 文本键盘模式,里面又包含了小写,大写,符号三类键盘

LV_KB_MODE_NUM: 数字键盘模式,比较专一,只能用来输入数字

2.1.2 键盘样式数据类型

```
enum {
    LV_KB_STYLE_BG,
    LV_KB_STYLE_BTN_REL,
    LV_KB_STYLE_BTN_PR,
    LV_KB_STYLE_BTN_TGL_REL,
    LV_KB_STYLE_BTN_TGL_PR,
    LV_KB_STYLE_BTN_INA,
};

typedef uint8_t lv_kb_style_t;
```

虽然这里面有 6 种样式,但是最后面的 5 种样式是用来修饰按钮的,跟 lv_btn 按钮章节中的使用方法是一样的,这里就不介绍了

LV_KB_STYLE_BG: 用来修饰键盘的背景,使用样式中的 body 字段,默认值为 lv_style_pretty

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_kb_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 绑定文本域控件

```
void lv_kb_set_ta(lv_obj_t * kb, lv_obj_t * ta);
```

参数:

kb: 键盘对象

ta: 文本域对象

2.2.3 设置键盘的模式

```
void lv_kb_set_mode(lv_obj_t * kb, lv_kb_mode_t mode);
```

参数:

kb: 键盘对象

mode: 键盘模式,有如下俩个可选值

LV_KB_MODE_TEXT: 文本键盘模式,里面又包含了小写,大写,符号三类键盘

LV_KB_MODE_NUM: 数字键盘模式,比较专一,只能用来输入数字

2.2.4 是否接管对光标的控制

```
void lv_kb_set_cursor_manage(lv_obj_t * kb, bool en);
```

参数:

kb: 键盘对象

en: 是否使能接管

接管之后的好处就是一致性体验更好,具体表现是当键盘绑定新的文本域控件时,键盘会将之前绑定的文本域控件上的光标给隐藏掉,而让新绑定的文本域控件上显示光标

2.2.5 设置按键映射表

```
static inline void lv_kb_set_map(lv_obj_t * kb, const char * map[]);
```

参数:

kb: 键盘对象

map: 按键映射表

利用此 API 接口是可以实现自定义键盘的,这个接口的使用方法和 lv_btmn 章节中的 lv_btmn_set_map 接口使用方法是一模一样的,详情请看 lv_btmn 章节

2.2.6 设置按键属性控制映射表

```
static inline void lv_kb_set_ctrl_map(lv_obj_t * kb, const lv_btmn_ctrl_t ctrl_map[]);
```

参数:

kb: 键盘对象

ctrl_map: 属性控制映射表

利用此 API 接口是可以实现自定义键盘的,这个接口的使用方法和 lv_btm 章节中的 lv_btm_set_ctrl_map 接口使用方法是一模一样的,详情请看 lv_btm 章节

2.2.7 设置样式

```
void lv_kb_set_style(lv_obj_t * kb, lv_kb_style_t type, const lv_style_t * style);
```

参数:

kb: 键盘对象

type: 设置哪一部分的样式,有如下 6 个可选值

LV_KB_STYLE_BG: 修饰键盘背景的
LV_KB_STYLE_BTN_REL: 下面 5 个都是用来修饰键盘上按钮的
LV_KB_STYLE_BTN_PR,
LV_KB_STYLE_BTN_TGL_REL,
LV_KB_STYLE_BTN_TGL_PR,
LV_KB_STYLE_BTN_INA,

2.2.8 键盘默认的事件回调函数

```
void lv_kb_def_event_cb(lv_obj_t * kb, lv_event_t event);
```

参数:

kb: 键盘对象

event: 当前事件

当我们不设置事件回调函数时,那么此 API 接口就是键盘控件默认的事件回调函数,它会帮我们处理按键按下,按键值改变,管理绑定的文本域等操作,当然了你完全可以用你自定义的事件回调函数来覆盖掉它,但是为了减少一些相同事务的处理,我们可以在我自定义的事件回调函数里的前面调用一下默认的 lv_kb_def_event_cb 事件回调函数.如以下示意代码所示:

```
lv_obj_t * kb1;  
void my_event_handler(lv_obj_t * obj, lv_event_t event) // 自定义的事件回调函数  
{  
    if(obj==kb1) // 是键盘对象  
    {  
        // 调用键盘的默认事件回调函数,帮我们处理掉一些通用的操作  
        lv_kb_def_event_cb(obj, event);  
        /*下面这里可以加你自己的功能代码*/  
    }  
}
```

2.2.9 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

先创建 2 个文本域控件,全部设置为单行模式,然后给第二个文本域控件使能密码保护功能,然后再接着创建一个键盘控件,设置其为文本键盘模式,对其中的小写键盘进行自定义改造,主要是在空格键的右边添加了一个@普通按键和一个 Clear 特殊功能按键,当点击@键时,是往相应的文本域中输入@字符,当点击 Clear 键时,是将相应的文本域中的内容给清空,最后我们给键盘设置自定义的事件回调函数,我们在 LV_EVENT_VALUE_CHANGED 事件中实现 Clear 特殊按键和自定义键盘的功能,在 LV_EVENT_APPLY 和 LV_EVENT_CANCEL 事件中把当前键盘给删除掉,当键盘被删除时,点击上面的文本域控件又可以把键盘给创建出来,当键盘已经存在时,点击某文本域控件就是将某文本域绑定到键盘上.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_kb_test.c 和 lv_kb_test.h 两个文件,其中 lv_kb_test.c 文件的内容如下:

```
#include "lv_kb_test.h"
#include "lvgl.h"
#include "key.h"
#include <string.h>

lv_obj_t * kb1;
lv_obj_t * ta1,* ta2;

/*
实现自定义键盘,可以参考 lv_kb.c 源码中的映射表的格式我们这里在官方自带的小写键盘基础上增加 2 个按键,都放到空格键的右边,一个是普通的@符号按键,另外一个是输入内容清空键,即 Clear 键,这个键是特殊功能键,littleVGL 中不自带此特殊功能的键,所以我们得自定义此特殊键
*/
#define MY_KB_CTRL_BTN_FLAGS (LV_BTNM_CTRL_NO_REPEAT | LV_BTNM_CTRL_CLICK_TRIG) //无重复按下功能,选择松手触发
```

```
static const char * const my_kb_map_lc[] = {  
    "1#", "q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "Bksp", "\n",  
    "ABC", "a", "s", "d", "f", "g", "h", "j", "k", "l", "Enter", "\n",  
    "_", "-", "z", "x", "c", "v", "b", "n", "m", ".", ",", ":" , "\n",  
    LV_SYMBOL_CLOSE,  
    LV_SYMBOL_LEFT, " ",  
    /*自己添加的键*/ "@",  
    /*自己添加的键*/ "Clear",  
    LV_SYMBOL_RIGHT,  
    LV_SYMBOL_OK, ""};
```

```
static const lv_btm_ctrl_t my_kb_ctrl_lc_map[] = {  
    MY_KB_CTRL_BTN_FLAGS | 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 7,  
    MY_KB_CTRL_BTN_FLAGS | 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7,  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
    MY_KB_CTRL_BTN_FLAGS | 2, 2,  
    /*由原来的 6 改为了 3*/ 3,  
    /*自己添加的键的控制属性*/ 1,  
    /*自己添加的键的控制属性*/ MY_KB_CTRL_BTN_FLAGS | 2,  
    2,  
    MY_KB_CTRL_BTN_FLAGS | 2};
```

```
void kb_create(lv_obj_t * parent);  
  
//事件回调函数  
void event_handler(lv_obj_t * obj, lv_event_t event)  
{  
    if(obj==ta1||obj==ta2)  
    {  
        if(event==LV_EVENT_RELEASED)//松手事件  
        {  
            if(kb1==NULL)  
                kb_create(lv_scr_act());//如果键盘不存在的话,则先创建键盘  
            else  
                lv_kb_set_ta(kb1,obj);//重新绑定文本域对象  
        }  
    }else if(obj==kb1)  
    {  
        //获取按下键的文本标题,放到 lv_kb_def_event_cb 的前面调用  
        const char * key_txt = lv_btm_get_active_btn_text(kb1);
```

```
//调用键盘的默认事件回调函数,帮我们处理掉一些通用的操作,如果让我们自己写
//代码来处理,那就太麻烦了
lv_kb_def_event_cb(kb1,event);
//添加自己的功能代码
if(event==LV_EVENT_VALUE_CHANGED)
{
    //uint16_t key_id = *(uint16_t*)lv_event_get_data();//获取按下键的 id,第一个按键
    //的 id 为 0,后面的按键依次增 1
    if(key_txt==NULL)
        return;
    if(strcmp(key_txt,"Clear")==0)//按下的是清空键
    {
        lv_obj_t * bind_ta = lv_kb_get_ta(kb1);//获取当前绑定的文本域对象
        lv_ta_set_text(bind_ta,"");
    }
    else if(strcmp(key_txt,"abc")==0)//按下的是切换小写键盘键
    {
        //重定向到我们自己自定义的小写键盘,而不是系统自带的小写键盘
        lv_kb_set_map(kb1,(const char **)my_kb_map_lc);//设置自定义按键的映射表
        //设置自定义按键的属性控制映射表
        lv_kb_set_ctrl_map(kb1,my_kb_ctrl_lc_map);
    }
}
else if(event==LV_EVENT_APPLY)
{
/*
    这里可以根据用户输入的文本信息,做相应的业务逻辑处理
*/
    lv_obj_del(kb1);//点击√键时把键盘删除掉
    kb1 = NULL;
}
else if(event==LV_EVENT_CANCEL)
{
/*
    这里也可以根据用户输入的文本信息,做相应的业务逻辑处理
*/
    lv_obj_del(kb1);//点击×键时把键盘删除掉
    kb1 = NULL;
}
}

//创建键盘
void kb_create(lv_obj_t * parent)
{
    kb1 = lv_kb_create(parent,NULL);
    lv_kb_set_cursor_manage(kb1,true);//使能对光标的接管
```

```
lv_kb_set_mode(kb1,LV_KB_MODE_TEXT);//设置为文本键盘模式,这也是默认值  
//先默认绑定 ta1 文本域对象,后面可以通过点击某文本域来重新绑定到相应的文本域对象  
lv_kb_set_ta(kb1,ta1);  
lv_kb_set_map(kb1,(const char **)my_kb_map_lc);//设置自定义按键的映射表  
lv_kb_set_ctrl_map(kb1,my_kb_ctrl_lc_map);//设置自定义按键的属性控制映射表  
lv_obj_set_event_cb(kb1,event_handler);//设置自定义的事件回调函数  
}  
  
//例程入口  
void lv_kb_test_start()  
{  
    lv_obj_t *scr = lv_scr_act();//获取当前活跃的屏幕对象  
    lv_obj_set_click(scr,true);  
    lv_obj_set_event_cb(scr,event_handler);  
  
    //1.创建俩个文本域对象  
    //1.1 创建 ta1 文本域对象  
    ta1 = lv_ta_create(scr,NULL);  
    lv_obj_set_width(ta1,200);//设置宽度  
    lv_obj_align(ta1,NULL,LV_ALIGN_IN_TOP_MID,0,15);  
  
    //使能单行模式,在单行模式下,文本域的高度是不能被设置的  
    lv_ta_set_one_line(ta1,true);  
    lv_ta_set_text(ta1,"");//设置为空文本  
    lv_obj_set_event_cb(ta1,event_handler);//设置事件回调函数  
  
    //1.2 创建 ta2 文本域对象  
    ta2 = lv_ta_create(scr,ta1);//直接从 ta1 进行复制  
    lv_ta_set_pwd_mode(ta2,true);//使能密码模式  
  
    //先隐藏掉光标  
    lv_ta_set_cursor_type(ta2,LV_CURSOR_LINE|LV_CURSOR_HIDDEN);  
    lv_obj_align(ta2,ta1,LV_ALIGN_OUT_BOTTOM_MID,0,15);  
  
    //2.创建键盘对象,创建出来的键盘默认是与父对象底部居中对齐的,  
    //默认宽度等于父对象的可适应宽度(即除掉左右内边距后的宽度),  
    //默认高度等于父对象的可适应高度(即除掉上下内边距后的高度)的一半  
    kb_create(scr);  
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

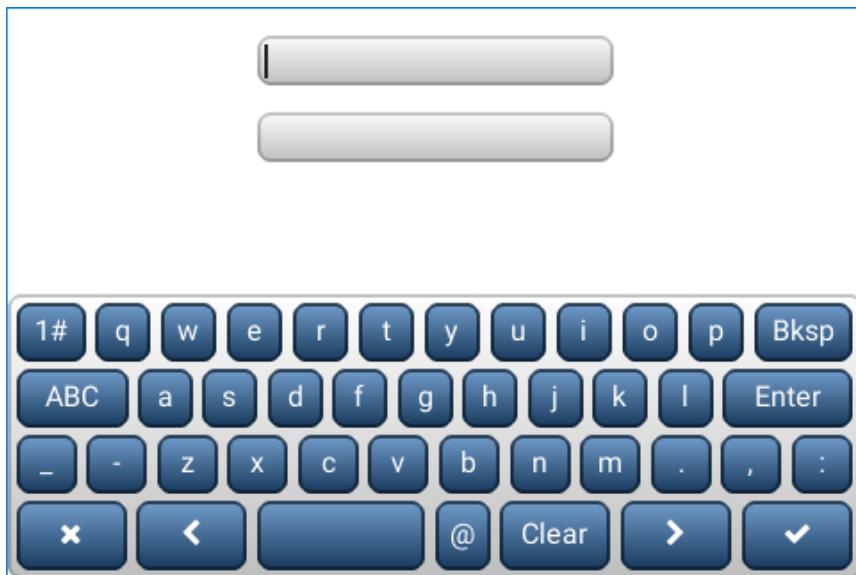


图 3.4.1 初始界面效果

如上图所示,可以看到空格键的右边有我们新增的@普通按键和 Clear 特殊功能按键,当点击@键时,是往相应的文本域中输入@字符,当点击 Clear 键时,是将相应的文本域中的内容

给清空,当点击  键或者点击  键时,会把当前的键盘给删除掉,然后点击上面的文本域控件时又可以把键盘给创建出来了.当键盘已经存在时,点击某文本域控件就是将某文本域绑定到键盘上.

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_spinbox 递增递减

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

lv_spinbox 递增递减

1. 介绍

lv_spinbox 递增递减控件本质上就是一个 lv_ta 文本域控件,只不过在功能上做了某种延伸,主要是用于递增或者递减地调节某数值,它是可以按照某个 step 步值来精确调节的,具体的 step 步值是通过 lv_spinbox_set_step(spinbox, step)接口来设置的,比如 step 设置为 10,那么当你递增一次或者递减一次时,原来的数值将会增加 10 或者减少 10,而一次递增操作是通过 lv_spinbox_increment(spinbox)接口来完成的,递减操作是通过 lv_spinbox_decrement(spinbox)接口来完成的.当递增或者递减之后的值将要超出所设置的范围时,数值将会保持不变,而这个限定范围是通过 lv_spinbox_set_range(spinbox, min, max)接口来设置的.

先给大家上一张 lv_spinbox 的大致外观图,如下所示:



图 1.1 lv_spinbox 大致外观

有了感性认识之后,我们再来说一下 lv_spinbox 在显示方面的格式控制,我们可以把整个显示内容细分成五个小部分,第一部分就是最前面的符号位,当数值是正数时,显示+号,当数值是负数时,显示-号,然后第二部分就是处于符号位和数值之间的空格间隙了,至于有多少个空格,这是通过 lv_spinbox_set_padding_left(spinbox, cnt)接口来设置的,然后第三部分就是数值了,第四部分就是分割符小数点了,然后这里有一点必须要说清楚,那就是分隔符小数点的作用,它仅仅在显示上起到示意作用,它不属于数值部分,比如图 1.1 中显示 13.2,对于用户来说,看上去就是一个 13.2 的小数,但是对于 lv_spinbox 控件自身而言,代码层面而言,它实际是一个 132 的整数,因为在 lv_spinbox 中,数值的存储数据类型是 int32_t,所以它是不可能存储小数的,大家一定要认识到这一点,但是利用这个小数点的显示,你是可以间接实现小数功能的,因为显示的数和实际的数之间就是一个倍数转换关系而已,然后数值的总位数和分割符小数点的位置是通过 lv_spinbox_set_digit_format(spinbox, digit_count, separator_position)接口来设置的,第五部分就是闪烁显示的光标了,这个光标是用来示意当前的 step 步值精度的,比如当 step 小于 10 时,光标就会在个位上闪烁,当 step 小于 100 时,光标就会在十位上闪烁,以此类推.除了通过 lv_spinbox_set_step(spinbox, step)接口改变 step 值来改变光标的位置外,还可以通过 lv_spinbox_step_prev(spinbox)接口或 lv_spinbox_step_next(spinbox)接口来改变光标位置,这两个 API 接口从字面意思上来理解就是将光标左移一位或者右移一位,其实它的本质还是通过修改 step 值来实现的,一个是将当前的 step 值扩大 10 倍,一个是将当前的 step 值缩小 10 倍.

最后来说一下 lv_spinbox 控件的事件,当它的数值改变时,LV_EVENT_VALUE_CHANGED 事件将会被触发.

2. lv_spinbox 的 API 接口

2.1 主要数据类型

2.1.1 样式数据类型

```
enum {
    LV_SPINBOX_STYLE_BG,
    LV_SPINBOX_STYLE_SB,
    LV_SPINBOX_STYLE_CURSOR,
};

typedef uint8_t lv_spinbox_style_t;
```

LV_SPINBOX_STYLE_BG: 修饰递增递减控件的背景,默认值为 lv_style_pretty

LV_SPINBOX_STYLE_SB: 修饰递增递减控件的滚动条,基本上用不到,我们说了 lv_spinbox 其实就是一个 lv_ta 控件,而 lv_ta 是具有滚动条的,所以 lv_spinbox 也具有滚动条,默认值为 lv_style_pretty_color

LV_SPINBOX_STYLE_CURSOR: 修饰递增递减控件的光标

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_spinbox_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置数值范围

```
void lv_spinbox_set_range(lv_obj_t * spinbox, int32_t range_min, int32_t range_max);
```

参数:

spinbox: 递增递减对象

range_min: 最小值

range_max: 最大值

2.2.3 直接设置数值

```
void lv_spinbox_set_value(lv_obj_t * spinbox, int32_t i);
```

参数:

spinbox: 递增递减对象

i: 数值,只能是一个整数,配合分隔符小数点的位置可以实现显示小数,比如你要显示 1.23,那么这里 i 你要传入 123,然后再设置小数点到相应位置就可以了

2.2.4 设置 step 步值

```
void lv_spinbox_set_step(lv_obj_t * spinbox, uint32_t step);
```

参数:

spinbox: 递增递减对象

step: 只能传入一个整数,比如 step 设置为 5,那么当你递增一次或者递减一次时,原来的数值将会增加 5 或者减少 5

如果不设置的话,那么 step 的默认值为 1

2.2.5 设置空格间隙

```
void lv_spinbox_set_padding_left(lv_obj_t * spinbox, uint8_t padding);
```

参数:

spinbox: 递增递减对象

padding: 在符号位和数值之间的空格个数

2.2.6 设置样式

```
static inline void lv_spinbox_set_style(lv_obj_t * spinbox, lv_spinbox_style_t type,  
lv_style_t * style);
```

参数:

spinbox: 递增递减对象

type: 设置哪一部分的样式,有如下 3 个可选值

LV_SPINBOX_STYLE_BG: 修饰背景的

LV_SPINBOX_STYLE_SB: 修饰滚动条的

LV_SPINBOX_STYLE_CURSOR: 修饰光标的

style: 样式

2.2.7 设置显示格式

```
void lv_spinbox_set_digit_format(lv_obj_t * spinbox, uint8_t digit_count,  
                                  uint8_t separator_position);
```

参数:

spinbox: 递增递减对象

digit_count: 设置数值的总位数,当数值的实际位数不够此位数时,会在数值的前面加前导 0 进行补齐

separator_position: 分割符小数点的位置,从数值左边的第一位开始算为 1,后面的依次增 1,当设置为 0 时,代表无小数点

2.2.8 将光标右移一位

```
void lv_spinbox_step_next(lv_obj_t * spinbox);
```

参数:

spinbox: 递增递减对象

此 API 接口的实现本质是将当前值的 step 值缩小 10 倍

2.2.9 将光标左移一位

```
void lv_spinbox_step_prev(lv_obj_t * spinbox);
```

参数:

spinbox: 递增递减对象

此 API 接口的实现本质是将当前值的 step 值扩大 10 倍

2.2.10 将数值进行递增一次

```
lv_spinbox_increment(lv_obj_t * spinbox);
```

参数:

spinbox: 递增递减对象

2.2.11 将数值进行递减一次

```
void lv_spinbox_decrement(lv_obj_t * spinbox);
```

参数:

spinbox: 递增递减对象

2.2.12 获取当前的数值

```
int32_t lv_spinbox_get_value(lv_obj_t * spinbox);
```

参数:

spinbox: 递增递减对象

返回值:

返回当前的数值

2.2.13 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个 spinbox 对象,设置其 step 步值为 10,数值范围为[-1000,1000],设置空格间隙为 3 个空格,设置数值显示格式为 4 位长度,小数点位置为第 3 位,最后给其设置事件回调函数,在 LV_EVENT_VALUE_CHANGED 事件中,通过串口打印当前的数值,当按下 KEY0 按键时,进行一次递增操作,当按下 KEY1 按键时,进行一次递减操作.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1 按键
- 3) 串口

3.3 软件设计

在 GUI_APP 目录下创建 lv_spinbox_test.c 和 lv_spinbox_test.h 俩个文件,其中 lv_spinbox_test.c 文件的内容如下:

```
#include "lv_spinbox_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_obj_t * spinbox1;

//事件回调函数
void event_handler(lv_obj_t * obj,lv_event_t event)
{
    if(event==LV_EVENT_VALUE_CHANGED)
    {
        printf("spinbox value:%d\r\n",lv_spinbox_get_value(spinbox1));
    }
}
```

```
//例程入口
void lv_spinbox_test_start()
{
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建 spinbox 对象
    spinbox1 = lv_spinbox_create(scr, NULL);
    lv_obj_set_size(spinbox1, 80, 30); //设置大小
    lv_obj_align(spinbox1, NULL, LV_ALIGN_CENTER, 0, 0); //与屏幕居中对齐
    lv_spinbox_set_step(spinbox1, 10); //设置 step 步值为 10
    lv_spinbox_set_range(spinbox1, -1000, 1000); //设置数值范围 [-1000, 1000]
    lv_spinbox_set_padding_left(spinbox1, 3); //设置空格间隙

    //设置显示格式, 数值总共 4 位, 小数点在第 3 位
    lv_spinbox_set_digit_format(spinbox1, 4, 3);
    lv_spinbox_set_value(spinbox1, 0); //设置起始值为 0
    lv_obj_set_event_cb(spinbox1, event_handler); //设置事件回调函数
}

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

    if(key==KEY0_PRES)
    {
        lv_spinbox_increment(spinbox1); //进行递增操作
    }else if(key==KEY1_PRES)
    {
        lv_spinbox_decrement(spinbox1); //进行递减操作
    }
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

当按一下 KEY0 按键时,可以看到当前数值会加 10,当按一下 KEY1 按键时,可以看到当前数值会减 10,同时串口也会打印出当前的数值.

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_img 图片

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

lv_img 图片

1. 介绍

1.1 理论介绍

lv_img 就是一个图片控件,它就是根据你传入的图片源来显示你想要的图片,littleVGL 为了提供最大的灵活性,它支持如下三种图片源方式:

- 1) 内部 C 数组,用 lv_img_dsc_t 结构体来进行描述
- 2) 外部存储文件,比如 SD 卡或者 U 盘上的图片文件
- 3) LV_SYMBOL_XXX 形式的图标字体或者文本,此时 lv_img 图片就相当于一个 lv_label 标签控件

如果你确定好图片源之后,就可以通过 lv_img_set_src(img, src) 接口来显示此图片,此接口内部会自动判断出 src 是属于哪一种图片源方式,然后选择相应的解析程序把图片给显示出来.

本章我们主要讲解的是上面的第 1 种和第 3 种图片源方式,对于第 2 种图片源方式我们这里稍微介绍一下,如果想详细研究,请参考 GUIT\lv_examples\lv_tutorial\6_images 目录下的官方例程,我们先来介绍一下第 1 种内部 C 数组图片源方式,为了产生一个图片的 C 数组,我们必须得借助官方提供的图片在线转换工具(<https://littlevgl.com/image-to-c-array>),你可以选择一张 png, jpg 或者 bmp 格式的图片文件,通过此在线图片转换工具的转换,你可以得到一个.c 文件,在这个.c 文件中,它存放的是各个颜色深度下的图片像素数据,它是以数组方式存放的,它内部是通过宏预处理指令来保证只有和当前颜色深度匹配的那个像素数据数组才会生效,这些 C 数组将会保存到微处理器的内部 flash 上,所以请注意微处理器的 flash 容量大小是否能够满足你们的项目需求,对于这样产生的 C 数组,我们是不能直接使用的,littleVGL 会用 lv_img_dsc_t 结构体对其进行一次封装,比如这里以一张红花图片为例,封装之后的格式如下所示:

```
lv_img_dsc_t red_flower = {  
    .header.always_zero = 0,  
    .header.w = 100, //图片的宽度  
    .header.h = 75, //图片的高度  
    .data_size = 7500 * LV_COLOR_SIZE / 8, //C 数组的大小,单位为字节  
    .header.cf = LV_IMG_CF_TRUE_COLOR, //图片的转换格式  
    .data = red_flower_map, //C 数组,也就是图片的核心像素数据  
};
```

对于 red_flower 是不需要去修改的,因为它是根据图片在线转换工具中的相应配置来自动生成的,在有了这个 lv_img_dsc_t 结构体之后,我们就可以直接使用这个图片了,只不过你还需要在使用它的地方先用 LV_IMG_DECLARE(red_flower) 宏来申明一下此图片,具体调用方式如下所示:

```
LV_IMG_DECLARE(red_flower); //先申明此图片  
lv_img_set_src(img, &red_flower); //然后显示此图片
```

接着我们来稍微介绍一下第 2 种外部存储文件图片源方式,它是把图片数据文件放到了外部的储存介质上,比如 SD 卡或者 U 盘上,所以这里你必须得另外用到 littleVGL 的文件系统模块,然后对于这个外部存储文件的格式这里又可以分为俩大类,一类是图片的最原始格式,如.png 文件,它不需要经过任何转换,你只需要把这张.png 图片直接放到外部的储存介质上就可以了,但是对于这种方式,littleVGL 是不能直接支持的,你必须得外加实现 png 图片的解析库,可以参考官方的 https://blog.littlevgl.com/2018-10-05/png_converter 资料,对于另外一类是.bin 文件格式,它是需要经过在线图片转换工具的转换,拿到转换后的.bin 文件后,你就就可以调用 lv_img_set_src(img, "S:folder1/my_img.bin") 接口把图片给显示出来了,不过前提就是你得先实现 littleVGL 的文件系统驱动哦!

最后我们来说一下第 3 种图片源方式,当你使用此方式时,lv_img 控件其实就是一个 lv_label 标签控件,它就是用来显示图标字体或文本的,比如当你想要显示一个 OK 图标字体时,你可以直接调用 lv_img_set_src(img, LV_SYMBOL_OK) 接口进行显示,或者调用 lv_img_set_src(img, LV_SYMBOL_OK "OK") 接口来和文本进行混合显示,当你如果只想显示文本或者显示以文本开头的内容时,内容最前面你必须得加上 LV_SYMBOL_DUMMY 标记符,这个 LV_SYMBOL_DUMMY 宏的作用就是用来让系统能够正确的识别这是第 3 种图片源方式的,无其他任何作用,正确的调用应如下:

```
lv_img_set_src(img,LV_SYMBOL_DUMMY "Hello"); //只显示 Hello 文本  
lv_img_set_src(img,LV_SYMBOL_DUMMY "OK" LV_SYMBOL_OK); //显示 OK 文本和  
OK 图标
```

用一句话来高度总结就是:当你显示的内容是以文本开头时,你就必须得在最前面加上 LV_SYMBOL_DUMMY 宏来作为标记.

接着我们来说一下图片如何实现带有透明度显示,比如实现背景透明显示等,除了采用第 2 种图片源方式中的 png 解析库外,lv_img 中还支持如下俩种透明度方式:

Chrome keying 方式: 当图片的背景色与 lv_conf.h 中的 LV_COLOR_TRANSP 宏所定义的颜色相同时,那么图片中的背景色将不会被绘制出来,相当于实现了背景透明显示,比如把 LV_COLOR_TRANSP 宏的值给配置为纯绿色,然后原始图片如图 1.1 所示,最终的显示效果如图 1.2 所示

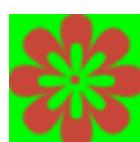


图 1.1.1 纯绿色背景的原始图片



图 1.1.2 最终的背景透明显示效果

Alpha byte 方式: 每一个像素增加一个额外的 alpha 透明度字节,此种方式可以实现任意地方透明显示,但是缺点就是存储空间会增加,而且渲染速度也会变慢

lv_img 图片控件支持在运行时进行图片重绘色,或者说是颜色混合,它是将图片中的每一个像素点与指定的颜色按照指定的强度进行混合操作,这里的颜色是通过样式中的 image.color 字段来指定,强度是通过样式中的 image.intense 字段来指定的,当 image.intense 字段的值为 0 时,代表不使能图片重绘色功能,否则就是使能了重绘色功能.

lv_img 控件还有一个大小自动适配的功能,这是通过 lv_img_set_auto_size(img, true) 接口来使能的,不过默认情况下就是使能的,所谓的大小自动适配就是指图片为多大,那么 lv_img 控件就为多大,当不使能大小自动适配,而且 lv_img 控件的大小比图片的大小还要大时,那么

在剩下的空间处,图片将会被重复的绘制,就跟贴瓦片一样,利用这个特性,我们可以用一张很窄的图片来实现一个全屏的背景壁纸,就如下面的效果所示:



图 1.1.3 图片重复绘制效果

其中上面的一张红花图片的大小为 100x75,而 lv_img 控件的大小为 200*150,正好可以重复绘制 4 张图片,有些时候,我们可能想调整图片在 lv_img 控件中的显示位置,它可以通过 lv_img_set_offset_x(img, x_ofs) 接口和 lv_img_set_offset_y(img, y_ofs) 接口来完成.

1.2 图片在线转换工具的使用

在本章中,学会图片在线转换工具的使用也是非常重要的,我们打开浏览器,输入 <https://littlevgl.com/image-to-c-array> 网址,对于国内用户来说,速度可能有点感人,没办法,大家忍忍吧,打开之后,可以看到如下所示的界面:

 LittlevGL
GET STARTED
LIVE DEMOS
DOCUMENTATION
TOOLS ▾
DOWNLOAD
FORUM
B

How to use the generated file in LittlevGL?

- For C arrays
 1. Copy the result C file into your LittlevGL project
 2. In a C file of your application declare the image as: `LV_IMG_DECLARE(my_image_name);`
 3. Set the image for an `lv_img` object: `lv_img_set_src(img1, &my_image_name);`
- For external binary files (e.g. SD card)
 1. Set up a new driver. To learn more read the [Tutorial](#).
 2. Set the image for an `lv_img` object: `lv_img_set_src(img1, "S:/path/to/image");`

Image file

Name

Color format

Alpha byte Add a 8 bit Alpha value to every pixel
Chroma keyed Make LV_COLOR_TRANSP (lv_conf.h) pixels to transparent

Output format

Dithering Dithering of True color images

Old version for v5.1.1

图 1.2.1 图片在线转换工具的界面

整体上来说,此工具还是比较简单的,我们现在来介绍一下每一个配置项的含义.

Image file: 从你的 PC 电脑上选择你想要转换的图片

Name: 生成的.c 文件名称

Color format: 颜色格式,有如下 14 个选项值,可以分为 4 类

[第 1 类,真彩色格式]

这类格式的好处就是显示出来的图片不失真,但是占用存储空间大

True color: 真彩色格式

True color with alpha: 带有 alpha 透明度的真彩色格式

True color chroma keyed: 带有 chroma keyed 透明度的真彩色格式

[第 2 类,调色板格式]

这类格式的好处就是占用存储空间小,缺点就是不能显示太鲜艳的图片

Indexed 2 colors: 当图片中的颜色种类不超过 2 种时,可以采用此格式

Indexed 4 colors: 当图片中的颜色种类不超过 4 种时,可以采用此格式

Indexed 16 colors: 当图片中的颜色种类不超过 16 种时,可以采用此格式

Indexed 256 colors: 当图片中的颜色种类不超过 256 种时,可以采用此格式

[第 3 类,纯阴影格式]

这类格式中的像素点只有 alpha 通道的数据,没有 R,G,B 等三个颜色通道的数据

Alpha only 2 shades: alpha 通道占 1 位,每一个像素有 2 种阴影等级

Alpha only 4 shades: alpha 通道占 2 位,每一个像素有 4 种阴影等级

Alpha only 16 shades: alpha 通道占 4 位,每一个像素有 16 种阴影等级

Alpha only 256 shades: alpha 通道占 8 位,每一个像素有 256 种阴影等级

[第 4 类,原始数据格式]

当你用到外部 png 解析库时,才会用到此类格式,它可以转换得到 png 图片的原始数据,而且是以 C 数组的方式存储在微处理器的内部 flash 上,如果你是把 png 图片直接放到外部存储介质上,如 SD 卡,U 盘等,那么你就用不到此类格式了

Raw: 原始数据格式

Raw with alpha: 带有 alpha 透明度的原始数据格式

Raw as chroma keyed: 带有 chroma keyed 透明度的原始数据格式

Output format: 输出格式,会根据选择的颜色格式不同出现不同的选项值,但整体上来说,它具有如下 5 个选项值

C array: 输出.c 文件,即 C 数组格式

Binary RGB332: 输出.bin 文件,即二进制格式,颜色为 RGB332 格式

Binary RGB565: 输出.bin 文件,即二进制格式,颜色为 RGB565 格式

Binary RGB565 Swap: 输出.bin 文件,即二进制格式,颜色为 RGB565 Swap 格式

Binary RGB888: 输出.bin 文件,即二进制格式,颜色为 RGB888 格式

Binary: 输出.bin 文件,即二进制格式,颜色格式自动确定

Dithering: 是否使能真彩色图片的颜色抖动显示,使能之后,转换得到的数据是已经经过抖动处理了的,所以它是不会增加运行时开销的,而 Dithering 抖动显示技术的好处就是它欺骗你的眼睛,使用有限的色彩让你看到比实际图象更多色彩的显示方式,通过在相邻像素间随机的加入不同的颜色来修饰图象,通常这种方式被用于颜色较少的情况下

Convert: 当你的所有配置项都设置好后,你可以点击此按钮来进行转换

2. lv_img 的 API 接口

2.1 主要数据类型

2.1.1 图片样式数据类型

```
enum {
    LV_IMG_STYLE_MAIN,
};

typedef uint8_t lv_img_style_t;
```

这里面就一种样式,比较简单,我们主要是来介绍一下此样式里某些字段的含义

image.color 字段: 重绘色时的混合颜色或者图标字体和文本的颜色

image.intense 字段: 重绘色时的混合强度,当此值为 0 时,代表不使能图片重绘色功能

image.opa 字段: 图片的整体透明度

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_img_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置图片源

```
void lv_img_set_src(lv_obj_t * img, const void * src_img);
```

参数:

img: 图片对象

src_img: 图片源,有 3 种方式,详情请看 1.1 理论介绍

2.2.3 设置大小自动适配

```
void lv_img_set_auto_size(lv_obj_t * img, bool autosize_en);
```

参数:

img: 图片对象

autosize_en: 是否使能大小自适应

当使能之后,lv_img 控件的大小将会根据所显示图片的大小来自动确定

2.2.4 设置 x 轴上的偏移量

```
void lv_img_set_offset_x(lv_obj_t * img, lv_coord_t x);
```

参数:

img: 图片对象

x: x 轴上的偏移量

这是设置图片在 lv_img 控件中的 x 轴偏移量

2.2.5 设置 y 轴上的偏移量

```
void lv_img_set_offset_y(lv_obj_t * img, lv_coord_t y);
```

参数:

img: 图片对象

y: y 轴上的偏移量

这是设置图片在 lv_img 控件中的 y 轴偏移量

2.2.6 设置样式

```
static inline void lv_img_set_style(lv_obj_t * img, lv_img_style_t type, const lv_style_t * style);
```

参数:

img: 图片对象

type: 设置哪一部分的样式,目前就只有 LV_IMG_STYLE_MAIN 这一个可选值

style: 样式

默认情况下,lv_img 控件的样式是为 NULL 的,即从父对象上继承样式

2.2.7 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰图片对象,然后接着创建 4 个 lv_img 图片对象,第 1 个图片对象用来显示图标字体和文本,第 2 个图片对象用来显示 True color 颜色格式的图片,第 3 个图片对象用来显示 True color with alpha 颜色格式的图片,第 4 个图片对象用来显示 True color chroma keyed 颜色格式的图片,并设置第 4 个图片对象不使能大小自动适配功能,设置其固定大小为 150*50,它的宽度是当前显示图片的 3 倍,高度相等,目的是为了给大家演示图片重绘现象,当按下 KEY0 按键时,会将样式中的 image. Intense 字段值加 10,当按下 KEY1 按键时,会将样式中的 image.opa 字段值减 10,这样大家就可以清楚地看到图片重绘色现象和透明度变化现象.然后本例程需要三张图片素材,我已经把它们放到了资料目录下,如下图所示:

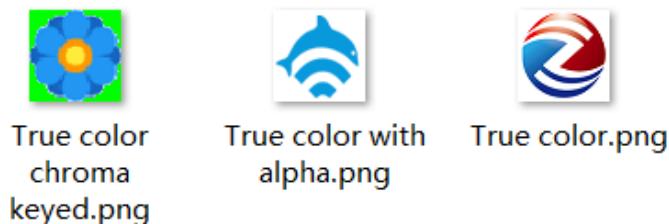


图 3.1.1 图片素材

这三张图片的大小都是 50*50 的,在背景上各有要求,分来用来演示不同的颜色格式效果, True color chroma keyed.png 图片是用来演示 True color chroma keyed 颜色格式的,所以他的背景色为纯绿色,而 True color with alpha.png 图片是用来演示 True color with alpha 颜色格式的,所以它的背景为透明,而 True color.png 图片是用来演示 True color 颜色格式的,所以它的背景不透明,为纯白色,而这三张图片在使用图片在线转换工具时,对于 Color format 配置项,分别选择它们对应的值,对于 Output format 配置项,它们都固定为”C array”选项值,对于 Dithering 配置项,它们都不使能.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) KEY0,KEY1 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_img_test.c 和 lv_img_test.h 俩个文件,其中 lv_img_test.c 文件的内容如下:

```
#include "lv_img_test.h"
#include "lvgl.h"
#include "key.h"

lv_style_t img_style;

//图片申明
LV_IMG_DECLARE(true_color);
LV_IMG_DECLARE(true_color_with_alpha);
LV_IMG_DECLARE(true_color_chroma_keyed);

//例程入口
void lv_img_test_start()
{
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建图片的样式
    lv_style_copy(&img_style,&lv_style_transp);
    img_style.image.color = LV_COLOR_RED; //图片重绘色时的混合颜色或者文本的颜色
    img_style.image.intense = 0; //暂时不使能重绘色功能
    img_style.image.opa = LV_OPA_COVER; //透明度

    //2.创建显示图标字体和文本的图片对象
    lv_obj_t * img1 = lv_img_create(scr,NULL);
    //以文本开头,前面必须得加 LV_SYMBOL_DUMMY
    lv_img_set_src(img1,LV_SYMBOL_DUMMY"Icon font: "LV_SYMBOL_AUDIO" audio");
    lv_img_set_style(img1,LV_IMG_STYLE_MAIN,&img_style); //设置样式
    lv_obj_align(img1,NULL,LV_ALIGN_IN_TOP_MID,0,10);

    //3.创建显示 True color 格式的图片对象
    lv_obj_t * img2 = lv_img_create(scr,NULL);
    lv_img_set_src(img2,&true_color);
    lv_img_set_style(img2,LV_IMG_STYLE_MAIN,&img_style); //设置样式
    lv_obj_align(img2,img1,LV_ALIGN_OUT_BOTTOM_MID,0,10);

    //4.创建显示 True color with alpha 格式的图片对象
    lv_obj_t * img3 = lv_img_create(scr,NULL);
    lv_img_set_src(img3,&true_color_with_alpha);
    lv_img_set_style(img3,LV_IMG_STYLE_MAIN,&img_style); //设置样式
    lv_obj_align(img3,img2,LV_ALIGN_OUT_BOTTOM_MID,0,10);

    //5.创建显示 True color chroma keyed 格式的图片对象
```

```
//同时需要把 lv_conf.h 中 LV_COLOR_TRANSP 宏的值改为 LV_COLOR_GREEN 纯
//绿色,目的就是为了保持和此图片的背景色一致,
//不过 LV_COLOR_TRANSP 的默认值就是纯绿色的
lv_obj_t* img4 = lv_img_create(scr,NULL);
lv_img_set_src(img4,&true_color_chroma_keyed);
lv_img_set_style(img4,LV_IMG_STYLE_MAIN,&img_style);//设置样式
lv_img_set_auto_size(img4,false);//不使能大小自动适配

//设置 lv_img 控件的宽度是 true_color_chroma_keyed 图片宽度的 3 倍,高度相等,可以
//看到在水平方向上有图片重复绘制现象
lv_obj_set_size(img4,150,50);
lv_obj_align(img4,img3,LV_ALIGN_OUT_BOTTOM_MID,0,10);

}

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

    if(key==KEY0_PRES)
    {
        //修改样式中的重绘色强度
        img_style.image.intense += 10;
        if(img_style.image.intense>=250)
            img_style.image.intense = 0;
        lv_obj_report_style_mod(&img_style);//刷新使用了此样式的对象
    }else if(key==KEY1_PRES)
    {
        //修改样式中的透明度
        img_style.image.opa -= 10;
        if(img_style.image.opa<=10)
            img_style.image.opa = LV_OPA_COVER;
        lv_obj_report_style_mod(&img_style);//刷新使用了此样式的对象
    }
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

然后我们可以多按几下 KEY0 键,来观察图片重绘色效果,如下图所示:

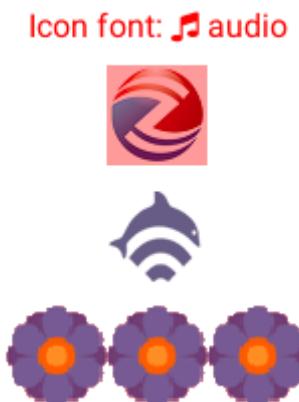


图 3.4.2 图片重绘色效果

当然了,你也可以多按几下 KEY1 键,来观察图片透明度变化的效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_imgbtn 图片按钮

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

lv_imgbtn 图片按钮

1. 介绍

lv_imgbtn 图片按钮控件跟 lv_btn 按钮控件是非常相似的,它只不过是在 lv_btn 按钮控件的每一个状态基础上增加了图片显示的功能,之前我们已经学习过了 lv_btn 按钮章节和 lv_img 图片章节,现在再来学习它们的混合体 lv_imgbtn 图片按钮控件就会简单很多.

我们知道按钮是具有五种状态的,而在 lv_imgbtn 图片按钮中也是同样具有五种状态的,我们可以通过 `lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &img_src)` 接口来给每一种状态设置一个对应的图片,不过这里有一点需要注意的是,这里的 `img_src` 图片源不支持图标字体和文本的方式,当采用这种方式设置图片之后,图片按钮的大小是自动适配图片大小的,用 `lv_obj_set_size` 接口来修改图片按钮的大小是无效的,另外 littleVGL 中还提供了另一种给图片按钮设置图片的接口,它是通过 `lv_conf.h` 中的 `LV_IMGBTN_TILED` 宏来使能的,当设为 1 使能之后,设置图片源的接口就变成了如下所示:

```
lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &src_left,&src_mid,&src_right)
```

从上面可以看出,接口名称是保持不变的,主要是形参发生了变化,之前是一个状态只能设置一个对应的图片,而现在是一个状态能同时设置三个图片,分别位于左边显示,中间显示,右边显示,在这种情况下,lv_imgbtn 图片按钮的宽度是可以设置的,高度是自动适配的,对于 `src_left`,`src_mid`,`src_right` 这三个图片源是可选的,对于不需要的传入 NULL 值就可以了,当 lv_imgbtn 图片按钮中有多余的宽度空间时,它会用 `src_mid` 图片进行重复填充的,默认情况下,`LV_IMGBTN_TILED` 功能是没有被使能的.

当 lv_imgbtn 图片按钮使用 `lv_imgbtn_set_toggle(imgbtn, true)` 接口使能 toggle 功能之后,用户的每一次 toggle 操作都会触发一个 `LV_EVENT_VALUE_CHANGED` 事件,当然了,你可以直接使用 `lv_imgbtn_set_state(imgbtn, LV_BTN_STATE_...)` 接口来改变图片按钮的状态,但是这种方式是不会触发 `LV_EVENT_VALUE_CHANGED` 事件的.

2. lv_imbtn 的 API 接口

2.1 主要数据类型

2.1.1 图片按钮样式数据类型

```
enum {
    LV_IMGBTN_STYLE_REL,
    LV_IMGBTN_STYLE_PR,
    LV_IMGBTN_STYLE_TGL_REL,
    LV_IMGBTN_STYLE_TGL_PR,
    LV_IMGBTN_STYLE_INA,
};

typedef uint8_t lv_imbtn_style_t;
```

这五种样式就是用来修饰其对应的五种状态,每一种样式中都只用到了 image 和 text 字段,含义如下:

image.color: 图片重绘色时的混合颜色

image.intense: 图片重绘色时的混合强度

image.opa: 图片的透明度

text.color: 修饰图片按钮对象中的文本颜色,text 字段中的其他子字段就不介绍了,很简单的

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_imbtn_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置某状态下的图片

```
#if LV_IMGBTN_TILED == 0
void lv_imbtn_set_src(lv_obj_t * imbtn, lv_btn_state_t state, const void * src);
#else
```

```
void lv_imgbtn_set_src(lv_obj_t * imgbtn, lv_btn_state_t state, const void * src_left, const
void * src_mid,const void * src_right);
#endif
```

参数:

imgbtn: 图片按钮对象

state: 按钮的状态

src: 在 LV_IMGBTN_TILED 没有使能时的图片源

src_left: 在 LV_IMGBTN_TILED 使能时的左边位置图片源,不需要的话,传入 NULL 值

src_mid: 在 LV_IMGBTN_TILED 使能时的中间位置图片源,不需要的话,传入 NULL 值

src_right: 在 LV_IMGBTN_TILED 使能时的右边位置图片源,不需要的话,传入 NULL 值

2.2.3 是否使能 toggle 功能

```
static inline void lv_imgbtn_set_toggle(lv_obj_t * imgbtn, bool tgl);
```

参数:

imgbtn: 图片按钮对象

tgl: 是否使能 toggle 功能

这个 API 接口的使用方法和 lv_btn 章节中的 lv_btn_set_toggle 接口的使用方法是一样的

2.2.4 设置按钮的当前状态

```
static inline void lv_imgbtn_set_state(lv_obj_t * imgbtn, lv_btn_state_t state);
```

参数:

imgbtn: 图片按钮对象

state: 按钮的状态

这个 API 接口的使用方法和 lv_btn 章节中的 lv_btn_set_state 接口的使用方法是一样的

2.2.5 进行一次 toggle 切换操作

```
static inline void lv_imgbtn_toggle(lv_obj_t * imgbtn);
```

参数:

imgbtn: 图片按钮对象

这个 API 接口的使用方法和 lv_btn 章节中的 lv_btn_toggle 接口的使用方法是一样的

2.2.6 设置样式

```
void lv_imgbtn_set_style(lv_obj_t * imgbtn, lv_imgbtn_style_t type, const lv_style_t * style);
```

参数:

imgbtn: 图片按钮对象

type: 设置哪一部分的样式,目前就只有 LV_IMG_STYLE_MAIN 这一个可选值

style: 样式

2.2.7 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰图片按钮按下时的效果,然后接着创建一个图片按钮对象,给其设置每个状态下的图片源,并给其添加”img btn”文本标题,然后使能它的 toggle 功能,最后给其设置事件回调函数,本例程中将会用到如下 2 个图片素材:

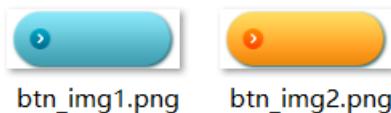


图 3.1.1 图片素材

这两张图片都是背景透明的,我们需要用 littleVGL 官方的图片在线转换工具将其进行转换,Color format 选择”true color with alpha”值,Output format 选择”C array”值即可.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) 串口

3.3 软件设计

在 GUI_APP 目录下创建 lv_imgbtn_test.c 和 lv_imgbtn_test.h 俩个文件,其中 lv_imgbtn_test.c 文件的内容如下:

```
#include "lv_imgbtn_test.h"
#include "lvgl.h"
#include <stdio.h>

//图片申明
LV_IMG_DECLARE(btn_img1);
LV_IMG_DECLARE(btn_img2);

lv_style_t pr_style;
```

```
//事件回调函数
void event_handler(lv_obj_t * obj,lv_event_t event)
{
    if(event==LV_EVENT_VALUE_CHANGED)
    {
        printf("LV_EVENT_VALUE_CHANGED\r\n");
    }
}

//例程入口
void lv_imgbtn_test_start()
{
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建按下时的样式
    lv_style_copy(&pr_style,&lv_style_plain);

    //图片重绘色时的混合色为黑色,这样看上去有按下的效果
    pr_style.image.color = LV_COLOR_BLACK;
    pr_style.image.intense = LV_OPA_50; //混合强度
    pr_style.text.color = LV_COLOR_MAKE(0xAA,0xAA,0xAA); //按下时的文本色

    //2.创建图片按钮对象
    lv_obj_t * imgbtn1 = lv_imgbtn_create(scr,NULL);

    //设置正常态松手时的图片
    lv_imgbtn_set_src(imgbtn1,LV_BTN_STATE_REL,&btn_img1);

    //设置正常态按下时的图片
    lv_imgbtn_set_src(imgbtn1,LV_BTN_STATE_PR,&btn_img1);

    //设置 toggle 切换态松手时的图片
    lv_imgbtn_set_src(imgbtn1,LV_BTN_STATE_TGL_REL,&btn_img2);

    //设置 toggle 切换态按下时的图片
    lv_imgbtn_set_src(imgbtn1,LV_BTN_STATE_TGL_PR,&btn_img2);

    //设置正常态按下时的样式
    lv_imgbtn_set_style(imgbtn1,LV_BTN_STATE_PR,&pr_style);

    //设置 toggle 切换态按下时的样式
    lv_imgbtn_set_style(imgbtn1,LV_BTN_STATE_TGL_PR,&pr_style);
    lv_imgbtn_set_toggle(imgbtn1,true); //使能 toggle 功能
```

```
lv_obj_align(imbtn1,NULL,LV_ALIGN_CENTER,0,0);//与屏幕居中对齐  
lv_obj_set_event_cb(imbtn1,event_handler);//设置事件回调函数  
lv_obj_t * label1 = lv_label_create(imbtn1,NULL); //给图片按钮添加标题  
lv_label_set_text(label1, "img btn");  
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

然后我们可以按住此图片按钮不放,就可以看到如下所示的按下效果:



图 3.4.2 按下效果(忽略掉上面的鼠标)

松开手之后,会进入到 toggle 切换态下的松手状态,如下图所示:



图 3.4.3 toggle 切换态下的松手状态

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_win 窗体

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

lv_win 窗体

1. 介绍

lv_win 窗体是一个最复杂的类似于容器的控件,它主要是由上下俩部分组成,上面主要是一个 header 容器,下面主要是一个存放内容的 page 页面,在 header 容器里的左侧是一个窗体标题,在右侧是一个控制按钮栏,所以换句话来说,它其实就是我们之前学过的 lv_cont 容器控件,lv_label 标签控件,lv_page 页面控件的一个组合使用,具体的构成图如下所示:

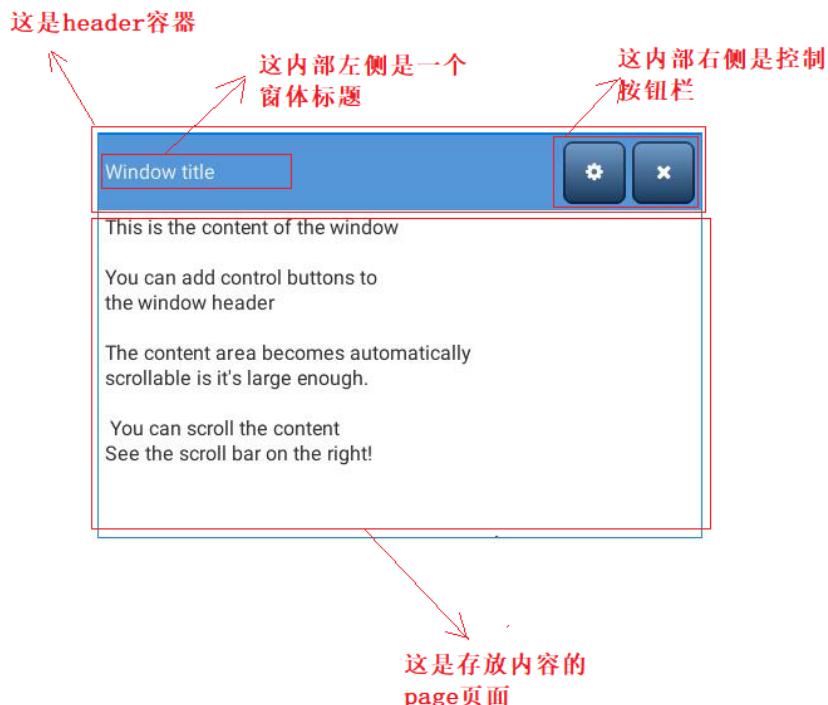


图 1.1 lv_win 窗体的构成

对于上面的窗体标题你是可以通过 lv_win_set_title(win, title) 接口来设置的,当然了,你还可以往右侧的控制按钮栏添加控制按钮,这是通过 lv_win_add_btn(win, &img_src) 接口来完成的,其中的 img_src 就是 lv_img 图片章节中学过图片源,被添加的控制按钮是依次从右往左排的,而且按钮的高度和宽度是相等的,不过你可以用 lv_win_set_btn_size(win, size) 接口来给所有的控制按钮设置统一的大小,然后这里有一点你需要注意的是,header 容器的高度会根据控制按钮的高度来自动确定的.

我们知道 lv_win 窗体的下面是一个用来存放内容的 lv_page 页面控件,所以 lv_win 窗体是具有 lv_page 的某些特性的,比如用 lv_win_set_sb_mode(win, LV_SB_MODE_...) 接口来设置窗体中的滚动条模式,用 lv_win_scroll_hor(win, dist_px) 接口和 lv_win_scroll_ver(win, dist_px) 接口来滚动窗体页面,或者用 lv_win_focus(win, child, LV_ANIM_ON/OFF) 接口来将窗体页面中的某个子对象直接处于可见状态,对于滚动和 focus 操作动画的时长是可以通过 lv_win_set_anim_time(win, anim_time_ms) 接口来设置的,如果你想调整窗体页面的布局方式,你可以调用 lv_win_set_layout(win, LV_LAYOUT_...) 接口来进行设置,上面说到的这些 API 接口都是能够在 lv_page 页面控件章节中找到对应体的.

2. lv_win 的 API 接口

2.1 主要数据类型

2.1.1 窗体样式数据类型

```
enum {
    LV_WIN_STYLE_BG,
    LV_WIN_STYLE_CONTENT,
    LV_WIN_STYLE_SB,
    LV_WIN_STYLE_HEADER,
    LV_WIN_STYLE_BTN_REL,
    LV_WIN_STYLE_BTN_PR,
};

typedef uint8_t lv_win_style_t;
```

LV_WIN_STYLE_BG: 用来修饰窗体背景的,使用样式中的 body 字段,默认值为 lv_style_plain

LV_WIN_STYLE_CONTENT: 用来修饰存放内容的 page 页面,使用样式中的 body 字段,默认值为 lv_style_transp

LV_WIN_STYLE_SB: 用来修饰 page 页面中的滚动条,默认值为 lv_style_pretty_color,具体用法请参考 lv_page 页面章节

LV_WIN_STYLE_HEADER: 用来修饰 header 容器的,默认值为 lv_style_plain_color

LV_WIN_STYLE_BTN_REL: 用来修饰控制按钮的松手状态,使用样式中的 body 字段,默认值为 lv_style_btn_rel

LV_WIN_STYLE_BTN_PR: 用来修饰控制按钮的按下状态,使用样式中的 body 字段,默认值为 lv_style_btn_pr

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_win_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 添加控制按钮

```
lv_obj_t * lv_win_add_btn(lv_obj_t * win, const void * img_src);
```

参数:

win: 窗体对象

img_src: 按钮上显示的图片,此参数就是 lv_img 图片章节中的图片源

返回值:

返回被添加的控制按钮对象

拿到被添加的控制按钮对象之后,你可以用 lv_obj_set_event_cb 接口来给此对象设置你自定义的事件回调函数,在事件回调函数中实现你自己的控制功能,如果你是想点击某控制按钮去关闭整个窗体的话,那么这里有一种快速的办法,如下所示:

```
lv_obj_set_event_cb( close_btn, lv_win_close_event_cb);
```

其中 close_btn 是一个被添加的控制按钮,而 lv_win_close_event_cb 是 littleVGL 系统内部定义的一个事件回调函数,它的作用就是监听 LV_EVENT_RELEASED 事件,监听到了的话,把整个窗体给关闭掉.

2.2.3 设置窗体的标题

```
void lv_win_set_title(lv_obj_t * win, const char * title);
```

参数:

win: 窗体对象

title: 窗体的标题

2.2.4 设置控制按钮的大小

```
void lv_win_set_btn_size(lv_obj_t * win, lv_coord_t size);
```

参数:

win: 窗体对象

size: 按钮的大小,宽=高=size

这是给所有的控制按钮设置统一的大小,当控制按钮的大小发生变化时,header 容器的高度也会自动发生变化,以实现对子对象的包裹

2.2.5 设置窗体页面的布局方式

```
void lv_win_set_layout(lv_obj_t * win, lv_layout_t layout);
```

参数:

win: 窗体对象

layout: 布局方式

这个 API 接口的使用方法和 lv_page 页面章节中的 lv_page_set_scrl_layout 接口的使用方法是一样的

2.2.6 设置窗体页面中的滚动条模式

```
void lv_win_set_sb_mode(lv_obj_t * win, lv_sb_mode_t sb_mode);
```

参数:

win: 窗体对象

sb_mode: 滚动条模式

这个 API 接口的使用方法和 lv_page 页面章节中的 lv_page_set_sb_mode 接口的使用方法是一样的

2.2.7 设置动画时长

```
void lv_win_set_anim_time(lv_obj_t * win, uint16_t anim_time);
```

参数:

win: 窗体对象

anim_time: 滚动和 focus 动画的时长,单位为 ms

2.2.8 设置样式

```
void lv_win_set_style(lv_obj_t * win, lv_win_style_t type, const lv_style_t * style);
```

参数:

win: 窗体对象

type: 设置哪一部分的样式,有如下 6 个可选值

LV_WIN_STYLE_BG: 用来修饰窗体背景的

LV_WIN_STYLE_CONTENT: 用来修饰存放内容的 page 页面

LV_WIN_STYLE_SB: 用来修饰 page 页面中的滚动条

LV_WIN_STYLE_HEADER: 用来修饰 header 容器的

LV_WIN_STYLE_BTN_REL: 用来修饰控制按钮的松手状态

LV_WIN_STYLE_BTN_PR: 用来修饰控制按钮的按下状态

style: 样式

2.2.9 设置窗体是否能够被拖拽

```
void lv_win_set_drag(lv_obj_t * win, bool en);
```

参数:

win: 窗体对象

en: 是否使能窗体的拖拽功能

当使能拖拽之后,要从窗体上面的header容器区域拖拽才有效,从下面的page页面区域拖拽是无效的

2.2.10 将页面中的某个子对象处于可见状态

```
void lv_win_focus(lv_obj_t * win, lv_obj_t * obj, lv_anim_enable_t anim_en);
```

参数:

win: 窗体对象

anim_en: 是否使能动画效果,有如下 2 个可选值

LV_ANIM_OFF:不使能

LV_ANIM_ON:使能

2.2.11 将窗体页面进行水平滚动

```
static inline void lv_win_scroll_hor(lv_obj_t * win, lv_coord_t dist);
```

参数:

win: 窗体对象

dist: 水平滚动距离

这个 API 接口的使用方法和 lv_page 页面章节中的 lv_page_scroll_hor 接口的使用方法是一样的

2.2.12 将窗体页面进行垂直滚动

```
static inline void lv_win_scroll_ver(lv_obj_t * win, lv_coord_t dist);
```

参数:

win: 窗体对象

dist: 垂直滚动距离

这个 API 接口的使用方法和 lv_page 页面章节中的 lv_page_scroll_ver 接口的使用方法是一样的

2.2.13 获取窗体的 page 页面对象

```
lv_obj_t * lv_win_get_content(const lv_obj_t * win);
```

参数:

win: 窗体对象

返回值:

返回窗体中用于存放内容的页面对象,拿到此页面对象之后,你就可以使用 lv_page 页面控件专有的 API 接口来操作它了

2.2.14 获取某控制按钮所在的窗体对象

```
lv_obj_t * lv_win_get_from_btn(const lv_obj_t * ctrl_btn);
```

参数:

ctrl_btn: 控制按钮对象

返回值:

返回此控制按钮所在的窗体对象

2.2.15 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰窗体的背景,然后创建一个窗体对象,设置其窗体标题,设置其控制按钮的大小,然后使能其拖拽功能,接着给其添加 2 个控制按钮,一个是用来关闭窗体的,一个是用来修改窗体标题的,最后往窗体页面中添加一个长文本的标签子对象.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_win_test.c 和 lv_win_test.h 俩个文件,其中 lv_win_test.c 文件的内容如下:

```
#include "lv_win_test.h"
#include "lvgl.h"

lv_style_t bg_style;
lv_obj_t * title_btn;

//事件回调函数
void event_handler(lv_obj_t * obj,lv_event_t event)
{
    if(obj==title_btn)
    {
        if(event==LV_EVENT_RELEASED)
        {
            //获取此控制按钮所在的窗体对象,其实就是 win1 对象
            lv_obj_t * win = lv_win_get_from_btn(title_btn);
            lv_win_set_title(win,"a new title");//修改窗体标题
        }
    }
}
```

```
}

//例程入口
void lv_win_test_start()
{
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建背景样式
    lv_style_copy(&bg_style,&lv_style_plain);
    bg_style.body.border.width = 2;
    bg_style.body.border.color = LV_COLOR_MAKE(0x55,0x96,0xD8);

    //2.创建窗体对象
    lv_obj_t * win1 = lv_win_create(scr,NULL);
    lv_obj_set_size(win1,220,220); //设置窗体的大小
    lv_obj_align(win1,NULL,LV_ALIGN_CENTER,0,0); //与屏幕居中对齐
    lv_win_set_title(win1,"Win title"); //设置窗体的标题
    lv_win_set_btn_size(win1,30); //设置控制按钮的大小
    lv_win_set_drag(win1,true); //使能拖拽功能
    lv_win_set_style(win1,LV_WIN_STYLE_BG,&bg_style); //设置背景样式

    //2.1 往窗体中添加一个控制按钮,用于关闭窗体
    lv_obj_t * close_btn = lv_win_add_btn(win1,LV_SYMBOL_CLOSE);

    //给控制按钮设置事件回调函数,lv_win_close_event_cb 是 littleVGL 系统内部自定义的
    //一个事件回调函数
    //专门用于关闭窗体的
    lv_obj_set_event_cb(close_btn,lv_win_close_event_cb);

    //2.2 再往窗体中添加一个控制按钮,用于修改窗体的标题
    title_btn = lv_win_add_btn(win1,LV_SYMBOL_SETTINGS);
    lv_obj_set_event_cb(title_btn,event_handler);

    //2.3 往窗体页面中添加一个长文本的标签子对象
    lv_obj_t * label1 = lv_label_create(win1,NULL);
    lv_label_set_text(label1, "This is the content of the window\n\n"
                           "You can add control buttons to\n"
                           "the window header\n\n"
                           "The content area becomes automatically\n"
                           "scrollable if it's large enough.\n\n"
                           "You can scroll the content\n"
                           "See the scroll bar on the right!");
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

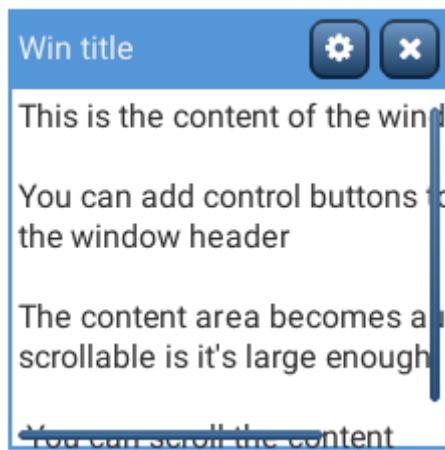


图 3.4.1 初始界面效果



然后我们可以点击一下控制按钮,可以看到窗体的标题发生了变化,如下图所示:

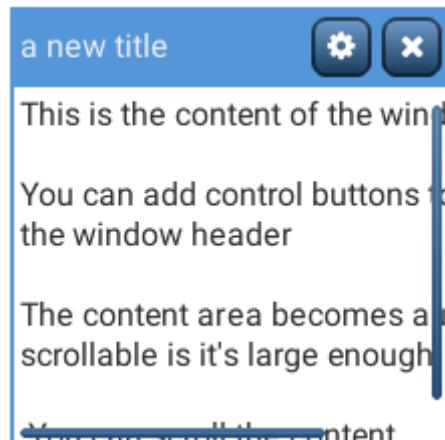


图 3.4.2 窗体标题发生变化



接着你可以从窗体的 header 容器区域测试拖拽效果,最后你可以点击一下控制按钮来把整个窗体给关闭了.

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_list 列表

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

lv_list 列表

1. 介绍

lv_list 列表控件是由一个作为背景的 page 页面和一些作为列表项的按钮构成的,其中的每一个列表项按钮是由可选的左右俩部分构成,左侧是用来放图标的,右侧是用来放文本的,具体的构成示意图如下所示:



图 1.1 lv_list 列表控件的构成

当 lv_list 列表中的列表项足够多时,lv_list 列表就可以被滚动了,你可以通过 lv_list_add_btn(list, &icon_img, text)接口来给 lv_list 列表添加列表项按钮,icon_img 就是按钮内部左侧的图标,跟 lv_img 图片章节中的图片源用法是一样的,text 就是按钮内部右侧的文本,其中 icon_img 图标是可选的,当传入 NULL 值时,代表无图标,那么它占据的位置也会被释放出来留给文本使用,列表项按钮的宽度和 lv_list 列表控件的宽度保持一致,列表项按钮的高度是根据其内部的内容大小来自动确定的,具体值为内容的高度+padding.top+padding.bottom,按钮内部右侧的文本标签默认是具有 LV_LABEL_LONG_SCROLL_CIRC 长文本模式的,也就是说当文本内容足够长时,它是会进行循环滚动显示的,然后被添加的列表项按钮其实也是可以被删除的,直接使用 lv_obj_del(btn)接口来删除就可以了,其中的 btn 就是 lv_list_add_btn 接口返回出来的对象,如果你想把列表中的所有列表项都给清空的话,那么你可以使用 lv_list_clean(list)接口.

然后我们来说一下 lv_list 列表控件的手动导航,前提是列表控件中的列表项能够被滚动,否则手动导航 API 接口是无效的,使用 lv_list_up(list)接口可以将列表底部的一个不可见列表项按钮往上移动一步,使此列表项按钮处于可见状态,或者使用 lv_list_down(list)接口将列表顶部的一个不可见列表项按钮往下移动一步,同样使此列表项按钮处于可见状态,再或者使用 lv_list_focus(btn, LV_ANIM_ON/OFF)接口将任意一个不可见的列表项按钮处于可见状态.

2. lv_list 的 API 接口

2.1 主要数据类型

2.1.1 列表样式数据类型

```
enum {
    LV_LIST_STYLE_BG,//这上面 4 种是用于修饰列表中的背景页面的
    LV_LIST_STYLE_SCRL,
    LV_LIST_STYLE_SB,
    LV_LIST_STYLE_EDGE_FLASH,
    LV_LIST_STYLE_BTN_REL,//这下面 5 种是用于修饰列表中的列表项按钮的
    LV_LIST_STYLE_BTN_PR,
    LV_LIST_STYLE_BTN_TGL_REL,
    LV_LIST_STYLE_BTN_TGL_PR,
    LV_LIST_STYLE_BTN_INA,
};

typedef uint8_t lv_list_style_t;
```

虽然有 9 种样式,但是最上面 4 种是用于修饰列表中的背景页面的,跟 lv_page 页面章节中的使用方法是一样的,最下面的 5 种是用于修饰列表项按钮的,跟 lv_btn 按钮章节中的使用方法是一样的.

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_list_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 清空所有的列表项按钮

```
void lv_list_clean(lv_obj_t * list);
```

参数:

list: 列表对象

2.2.3 添加列表项按钮

```
lv_obj_t * lv_list_add_btn(lv_obj_t * list, const void * img_src, const char * txt);
```

参数:

list: 列表对象

img_src: 按钮内部左侧的图标,跟 lv_img 图片章节中的图片源用法是一样的

txt: 按钮内部右侧的文本

返回值:

把被添加的列表项按钮对象给返回出来,拿到此对象之后,我们可以用 lv_obj_set_event_cb 接口给此列表项按钮设置事件回调函数,以实现我们想要的逻辑功能,或者在后面还可以用 lv_obj_del 接口把此列表项按钮给删除掉

2.2.4 删除某位置上的列表项按钮

```
bool lv_list_remove(const lv_obj_t * list, uint16_t index);
```

参数:

list: 列表对象

index: 列表项按钮对应的索引位置,由上到下,从 0 开始

2.2.5 是否使能列表的单选模式

```
lv_list_set_single_mode(lv_obj_t * list, bool mode);
```

参数:

list: 列表对象

mode: 是否使能单选模式

默认情况下,单选模式是没有被使能的,假如使能之后,我们可以实现单选按钮组的功能,或者实现选中行高亮的功能,如下图所示,Open 列表项按钮就处于选中或者说是高亮状态,这完全取决于你样式修饰的效果.

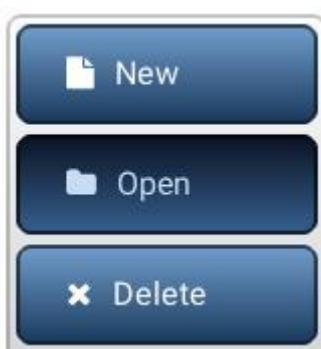


图 2.2.5.1 列表的单选模式效果

2.2.6 让某列表项按钮处于选中状态

```
void lv_list_set_btn_selected(lv_obj_t * list, lv_obj_t * btn);
```

参数:

list: 列表对象

btn: 列表项按钮对象

2.2.7 设置滚动条的模式

```
static inline void lv_list_set_sb_mode(lv_obj_t * list, lv_sb_mode_t mode);
```

参数:

list: 列表对象

mode: 滚动条的模式

此 API 接口的使用方法和 lv_page 页面章节中的 lv_page_set_sb_mode 接口的使用方法是一样的

2.2.8 是否使能边缘半圆弧动画效果

```
static inline void lv_list_set_edge_flash(lv_obj_t * list, bool en);
```

参数:

list: 列表对象

en: 是否使能

此 API 接口的使用方法和 lv_page 页面章节中的 lv_page_set_edge_flash 接口的使用方法是一样的

2.2.9 设置动画时长

```
static inline void lv_list_set_anim_time(lv_obj_t * list, uint16_t anim_time);
```

参数:

list: 列表对象

anim_time: 滚动或者 focus 动画的时长,单位 ms

2.2.10 设置样式

```
void lv_list_set_style(lv_obj_t * list, lv_list_style_t type, const lv_style_t * style);
```

参数:

list: 列表对象

type: 设置哪一部分的样式,详情请看 2.1.1 小节

style: 样式

2.2.11 获取列表项按钮内的文本内容

```
const char * lv_list_get_btn_text(const lv_obj_t * btn);
```

参数:

btn: 列表项按钮对象

返回值:

返回此列表项按钮内的文本内容

2.2.12 获取列表项按钮内的标签对象

```
lv_obj_t * lv_list_get_btn_label(const lv_obj_t * btn);
```

参数:

btn: 列表项按钮对象

返回值:

返回此列表项按钮内的标签对象,拿到此标签对象之后,你就可以用 lv_label 标签控件专有的 API 接口来操作它了

2.2.13 获取列表项按钮内的图片对象

```
lv_obj_t * lv_list_get_btn_img(const lv_obj_t * btn);
```

参数:

btn: 列表项按钮对象

返回值:

返回此列表项按钮内的图片对象,拿到此图片对象之后,你就可以用 lv_img 图片控件专有的 API 接口来操作它了

2.2.14 获取某列表项按钮所在的位置索引

```
int32_t lv_list_get_btn_index(const lv_obj_t * list, const lv_obj_t * btn);
```

参数:

list: 列表对象

btn: 列表项按钮对象

返回值:

返回 btn 列表项按钮对象在列表中的位置索引值

2.2.15 将列表上移一步

```
void lv_list_up(const lv_obj_t * list);
```

参数:

list: 列表对象

此 API 接口的作用是将列表底部的一个不可见列表项按钮往上移动一步,使此列表项按钮处于可见状态

2.2.16 将列表下移一步

```
void lv_list_down(const lv_obj_t * list);
```

参数:

list: 列表对象

此 API 接口的作用是将列表顶部的一个不可见列表项按钮往下移动一步,使此列表项按钮处于可见状态

2.2.17 将某列表项按钮处于可见状态

```
void lv_list_focus(const lv_obj_t * btn, lv_anim_enable_t anim);
```

参数:

list: 列表对象

anim: 是否开启动画效果,有如下俩个可选值

LV_ANIM_OFF: 不开启动画效果

LV_ANIM_ON: 开启动画效果

此 API 接口的作用是可以将任意一个不可见的列表项按钮处于可见状态

2.2.18 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建一个自定义样式来修饰列表控件的背景,然后创建一个列表控件,使能其单选模式,接着设置它的滚动条模式为自动模式,最后往它里面添加 5 个列表项按钮,并为每一个列表项按钮对象设置了事件回调函数,当按下 KEY0 按键时,会将此列表控件往上移动一步,当按下 KEY1 按键时,会将此列表控件往下移动一步,当按下 WKUP 按键时,会将最后一个列表项按钮处于可见状态.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) 串口
- 3) KEY0,KEY1,WKUP 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_list_test.c 和 lv_list_test.h 俩个文件,其中 lv_list_test.c 文件的内容如下:

```
#include "lv_list_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_style_t bg_style;
lv_obj_t * list1;
lv_obj_t * last_item_btn;

//定义列表项按钮
const char * const LIST_ITEM_BTN[][2] = {
    //含有长文本内容,将会看到循环滚动效果
    {LV_SYMBOL_FILE,"This is a long long text!"},
    {NULL,"Open"},//不给图标
    {LV_SYMBOL_CLOSE,"Close"},
```

```
{LV_SYMBOL_EDIT,"Edit"},  
{LV_SYMBOL_SAVE,"Save"}  
};  
//总共有多少个列表项按钮  
#define LIST_ITEM_BTN_NUM (sizeof(LIST_ITEM_BTN)/sizeof(LIST_ITEM_BTN[0]))  
  
//事件回调函数  
static void event_handler(lv_obj_t * obj,lv_event_t event)  
{  
    if(event==LV_EVENT_RELEASED)  
    {  
        printf("list item btn: %s\n",lv_list_get_btn_text(obj));  
    }  
}  
  
//例程入口  
void lv_list_test_start()  
{  
    lv_obj_t *scr = lv_scr_act(); //获取当前活跃的屏幕对象  
  
    //1.创建背景样式  
    lv_style_copy(&bg_style,&lv_style_transp);  
    bg_style.body.border.width = 2;  
    bg_style.body.border.color = LV_COLOR_MAKE(0x36,0x5D,0x85);  
    bg_style.body.radius = 6;  
    bg_style.body.padding.right = 8; //给右侧的滚动条留点空间  
  
    //2.创建列表对象  
    list1 = lv_list_create(scr,NULL);  
    lv_obj_set_size(list1,150,150); //设置大小  
    lv_obj_align(list1,NULL,LV_ALIGN_CENTER,0,0); //与屏幕居中对齐  
    lv_list_set_single_mode(list1,true); //使能列表的单选模式  
    lv_list_set_sb_mode(list1,LV_SB_MODE_AUTO); //设置滚动条模式  
    lv_list_set_style(list1,LV_LIST_STYLE_BG,&bg_style); //设置背景样式  
  
    //设置页面中容器载体的样式  
    lv_list_set_style(list1,LV_LIST_STYLE_SCRL,&lv_style_transp);  
  
    //2.1 添加列表项按钮  
    uint8_t i;  
    lv_obj_t * list_item_btn;
```

```
for(i=0;i<LIST_ITEM_BTN_NUM;i++)
{
    //添加列表项按钮
    list_item_btn = lv_list_add_btn(list1,LIST_ITEM_BTN[i][0],LIST_ITEM_BTN[i][1]);
    lv_obj_set_event_cb(list_item_btn,event_handler); //给列表项按钮设置事件回调函数
}
last_item_btn = list_item_btn; //保存最后一个列表项按钮对象
}

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

    if(key==KEY0_PRES)
    {
        lv_list_up(list1); //让列表向上移动一步
    }else if(key==KEY1_PRES)
    {
        lv_list_down(list1); //让列表向下移动一步
    }else if(key==WKUP_PRES)
    {
        //让最后一个列表项按钮直接处于可见状态
        lv_list_focus(last_item_btn,LV_ANIM_ON);
    }
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

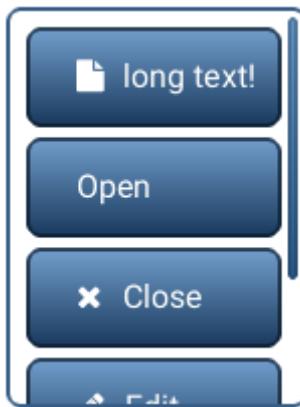


图 3.4.1 初始界面效果

然后我们可以随便滚动列表,并按一下 Close 按钮,使其处于选中状态,如下图所示:

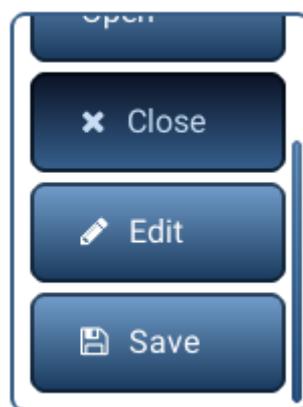


图 3.4.2 列表的单选效果

最后我们可以按下 KEY0,KEY1 或者 WKUP 按键来测试列表的手动导航效果.

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_ddlist 下拉列表框

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

lv_ddlist 下拉列表框

1. 介绍

lv_ddlist 是一个下拉列表框控件,它允许用户从多个选项值中去任意选择一个值,它具有收缩和展开两个状态,默认情况下,它是处于收缩状态的,在收缩状态下,它会将当前选择的值给显示出来,当通过 API 接口或者通过用户点击使其处于展开状态时,它会将所有的选项值都给显示出来,然后用户可以从中选择一个想要的值,一旦选择好之后, lv_ddlist 下拉列表框控件会自动从展开状态回到收缩状态的.如果你想通过 API 接口来手动改变下拉列表框控件的状态,那也是可以的,这是通过 lv_ddlist_open(ddlist) 和 lv_ddlist_close(ddlist) 接口来完成的,收缩和展开这两种状态的外观图如下所示:



图 1.1 下拉列表框控件的收缩状态



图 1.2 下拉列表框控件的展开状态

你可以通过 lv_ddlist_set_options(ddlist, options) 接口来给下拉列表框控件设置所有的选项值,其中 options 是一个字符串,而不是一个数组,所以每一个选项值之间需要用'\n'换行符来作为区分,如 const char * options = "First\nSecond\nThird"; 就定义了 First, Second, Third 这三个选项值,除了通过用户点击来选择选项值之外,我们还可以直接通过 lv_ddlist_set_selected(ddlist, id) 接口来选择某一个选项值,相反的,你可以通过 lv_ddlist_get_selected(ddlist) 接口来获取下拉列表框控件当前选择的选项值,这个 API 接口返回的是选项值在下拉列表框中的索引位置,如果你是想获取它的文本内容的话,那么你可以通过 lv_ddlist_get_selected_str(ddlist, buf, buf_size) 接口来完成.

在图 1.1 中所示的向下箭头默认情况下是没有的,它需要通过调用 lv_ddlist_set_draw_arrow(ddlist, true) 接口来使能绘制向下箭头,然后下拉列表框中文本的水平对齐方式是可以通过 lv_ddlist_set_align(ddlist, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT) 接口来设置的,然后这里需要来讲一下如何设置下拉列表框控件的大小,大家的第一印象可能会想到使用 lv_obj_set_size 接口,但是我告诉大家,这是无效的,因为下拉列表框控件提供了专门的 API 接口来设置其大小,设置宽度的话用 lv_ddlist_set_fix_width(ddlist, width) 接口,设置高度的话用 lv_ddlist_set_fix_height(ddlist, height) 接口,如果不设置宽和高的话,也是没有关系的,因为下拉列表框控件会自动根据选项值的多少和文本内容的长短来自动确定其高和宽的.当你设置的高度不足以展示所有的选项值时,下拉列表框控件就会出现滚动条,你可以通过 lv_ddlist_set_sb_mode(ddlist, LV_SB_MODE...) 接口来设置滚动条的模式.

最后我们来说一下下拉列表框控件的事件,当它的选项值被改变时,它会给它的事件回调函数发送一个 LV_EVENT_VALUE_CHANGED 事件.

2. lv_ddlist 的 API 接口

2.1 主要数据类型

2.1.1 下拉列表框样式数据类型

```
enum {
    LV_DDLIST_STYLE_BG,
    LV_DDLIST_STYLE_SEL,
    LV_DDLIST_STYLE_SB,
};

typedef uint8_t lv_ddlist_style_t;
```

LV_DDLIST_STYLE_BG: 用来修饰下拉列表框控件的背景,使用样式中的 body 和 text 字段,默认值为 lv_style_pretty

LV_DDLIST_STYLE_SEL: 用来修饰被选择的选项值,使用样式中的 body 和 text 字段,默认值为 lv_style_plain_color

LV_DDLIST_STYLE_SB: 用来修饰滚动条的,默认值为 lv_style_plain_color,具体用法请参考 lv_page 页面章节

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_ddlist_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置选项列表

```
void lv_ddlist_set_options(lv_obj_t * ddlist, const char * options);
```

参数:

ddlist: 下拉列表框对象

options: 选项列表,每一个选项值之间需要用'\n'换行符来作为区分,如 const char * options = "First\nSecond\nThird";就定义了 First, Secode, Third 这三个选项值

2.2.3 选中某个选项值

```
void lv_ddlist_set_selected(lv_obj_t * ddlist, uint16_t sel_opt);
```

参数:

ddlist: 下拉列表框对象

sel_opt: 选项索引值,从 0 开始

2.2.4 设置下拉列表框的高度

```
void lv_ddlist_set_fix_height(lv_obj_t * ddlist, lv_coord_t h);
```

参数:

ddlist: 下拉列表框对象

h: 高度值

如果不设置高度的话,也是没有关系的,下拉列表框控件会根据选项的个数来自动确定高度的

2.2.5 设置下拉列表框的宽度

```
void lv_ddlist_set_fix_width(lv_obj_t * ddlist, lv_coord_t w);
```

参数:

ddlist: 下拉列表框对象

w: 宽度值

如果不设置宽度的话,也是没有关系的,下拉列表框控件会根据选项值的最大文本长度来自动确定宽度的

2.2.6 是否使能绘制向下箭头

```
void lv_ddlist_set_draw_arrow(lv_obj_t * ddlist, bool en);
```

参数:

ddlist: 下拉列表框对象

en: 是否使能绘制收缩状态下右边的向下箭头

2.2.7 是否禁止自动返回到收缩状态

```
void lv_ddlist_set_stay_open(lv_obj_t * ddlist, bool en);
```

参数:

ddlist: 下拉列表框对象

en: true 代表禁止自动返回到收缩状态,false 代表不禁止

默认情况下是当用户从展开状态下选择好选项值时,下拉列表框会自动返回到收缩状态的,但是如果你调用 `lv_ddlist_set_stay_open(ddlist,true)` 接口之后,它将会一直保持在展开状态,而不会自动返回到收缩状态了

2.2.8 设置滚动条的模式

```
static inline void lv_ddlist_set_sb_mode(lv_obj_t * ddlist, lv_sb_mode_t mode);
```

参数:

ddlist: 下拉列表框对象

mode: 滚动条的模式

此 API 接口的使用方法和 `lv_page` 页面章节中的 `lv_page_set_sb_mode` 接口的使用方法是一样的

2.2.9 设置动画时长

```
static inline void lv_ddlist_set_anim_time(lv_obj_t * ddlist, uint16_t anim_time);
```

参数:

ddlist: 下拉列表框对象

anim_time: 展开和收缩动画的时长,单位 ms

2.2.10 设置样式

```
void lv_ddlist_set_style(lv_obj_t * ddlist, lv_ddlist_style_t type, const lv_style_t * style);
```

参数:

ddlist: 下拉列表框对象

type: 设置哪一部分的样式,有如下 3 个可选值

LV_DDLIST_STYLE_BG: 修饰下拉列表框控件的背景

LV_DDLIST_STYLE_SEL: 修饰被选择的选项值

LV_DDLIST_STYLE_SB: 修饰滚动条

style: 样式

2.2.11 设置文本的水平对齐方式

```
void lv_ddlist_set_align(lv_obj_t * ddlist, lv_label_align_t align);
```

参数:

ddlist: 下拉列表框对象

align: 水平对齐方式

2.2.12 获取当前选择的文本内容

```
void lv_ddlist_get_selected_str(const lv_obj_t * ddlist, char * buf, uint16_t buf_size);
```

参数:

ddlist: 下拉列表框对象

buf: 用来存放选项值的文本内容

buf_size: buf 缓冲区的大小

2.2.13 获取当前选择的选项值的索引

```
uint16_t lv_ddlist_get_selected(const lv_obj_t * ddlist);
```

参数:

ddlist: 下拉列表框对象

返回值:

返回当前选择的选项值的索引值,从 0 开始的

2.2.14 将下拉列表框处于展开状态

```
void lv_ddlist_open(lv_obj_t * ddlist, lv_anim_enable_t anim);
```

参数:

ddlist: 下拉列表框对象

anim: 是否使能动画效果,有如下俩个可选值

LV_ANIM_ON: 使能动画效果

LV_ANIM_OFF: 不使能动画效果

2.2.15 将下拉列表框处于收缩状态

```
void lv_ddlist_close(lv_obj_t * ddlist, lv_anim_enable_t anim);
```

参数:

ddlist: 下拉列表框对象

anim: 是否使能动画效果,有如下俩个可选值

LV_ANIM_ON: 使能动画效果

LV_ANIM_OFF: 不使能动画效果

2.2.16 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建 2 个自定义样式,分别用来修饰下拉列表框的背景和选项值被选中时的样式,接着创建一个下拉列表框对象,设置它的固定宽度,设置它的所有选项值,使能绘制向下箭头,最后给其设置事件回调函数,当按下 KEY0 按键时,会将下拉列表框强制设为展开状态,当按下 KEY1 按键时,会将下拉列表框强制设为收缩状态,当按下 WKUP 按键时,会禁止掉自动回到收缩状态的功能.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) 串口
- 3) KEY0,KEY1,WKUP 按键

3.3 软件设计

在 GUI_APP 目录下创建 lv_ddlist_test.c 和 lv_ddlist_test.h 俩个文件,其中 lv_ddlist_test.c 文件的内容如下:

```
#include "lv_ddlist_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

//是否将下拉列表框的弹出方向改为向上,默认情况是向下的
#define DDLIST_DIRECTION_UP_EN    0

lv_style_t bg_style;
lv_style_t sel_style;
lv_obj_t * ddlist1;

//事件回调函数
static void event_handler(lv_obj_t * obj,lv_event_t event)
{
```

```
if(event == LV_EVENT_VALUE_CHANGED)
{
    char buf[32];
    uint16_t selected_index = lv_ddlist_get_selected(obj);//获取选项值的索引
    lv_ddlist_get_selected_str(obj,buf,sizeof(buf));//获取选项值的文本内容
    printf("Option index: %d, Option text: %s\r\n",selected_index,buf);
}

//例程入口
void lv_ddlist_test_start()
{
    lv_obj_t *scr = lv_scr_act();//获取当前活跃的屏幕对象

    //1. 创建样式
    //1.1 创建背景样式
    lv_style_copy(&bg_style,&lv_style_plain);
    bg_style.body.main_color = LV_COLOR_WHITE;//纯白色背景
    bg_style.body.grad_color = bg_style.body.main_color;
    bg_style.body.border.width = 1;//边框宽度
    bg_style.body.border.color = LV_COLOR_MAKE(0xAA,0xAA,0xAA);
    //LV_COLOR_MAKE(0x30,0x30,0x30);
    bg_style.body.padding.left = 10;//设置左侧的内边距
    bg_style.text.color = LV_COLOR_BLACK;//文本颜色
    bg_style.body.shadow.color = bg_style.body.border.color;//阴影颜色
    bg_style.body.shadow.width = 4;//阴影宽度

    //1.2 创建选择项被选中时的样式
    lv_style_copy(&sel_style,&lv_style_plain);
    sel_style.body.main_color = LV_COLOR_MAKE(0x5F,0xB8,0x78);//浅绿色背景
    sel_style.body.grad_color = sel_style.body.main_color;
    sel_style.text.color = LV_COLOR_WHITE;//文本为白色

    //2. 创建下拉列表框
    ddlist1 = lv_ddlist_create(scr,NULL);

    lv_ddlist_set_options(ddlist1,"Shanghai\nBeijing\nShenzhen\nGuangzhou\nHangzhou\nNanchang");//设置列表选项
    lv_ddlist_set_selected(ddlist1,4);//设置默认选中值为 Hangzhou
    lv_ddlist_set_fix_width(ddlist1,140);//设置固定宽度
    lv_ddlist_set_draw_arrow(ddlist1,true);//使能绘制向下的箭头
    lv_ddlist_set_style(ddlist1,LV_DDLIST_STYLE_BG,&bg_style);//设置背景样式
```

```
lv_ddlist_set_style(ddlist1,LV_DDLIST_STYLE_SEL,&sel_style); //设置背景样式  
lv_obj_set_event_cb(ddlist1, event_handler); //注册事件回调函数  
lv_obj_align(ddlist1,NULL,LV_ALIGN_IN_TOP_MID,0,20); //设置与屏幕的对齐方式  
#if(DDLIST_DIRECTION_UP_EN)  
  
    //重新设置与屏幕的对齐方式,必须为底部对齐  
    lv_obj_align(ddlist1,NULL,LV_ALIGN_IN_BOTTOM_MID,0,-20);  
    //必须使能自动对齐,只有满足这两点,才能看到向上弹出的效果  
    lv_obj_set_auto_realign(ddlist1,true);  
#endif  
}  
  
//按键处理  
void key_handler()  
{  
    u8 key = KEY_Scan(0);  
  
    if(key==KEY0_PRES)  
    {  
        lv_ddlist_open(ddlist1,LV_ANIM_ON); //把下拉列表框强制设置为展开状态  
    }else if(key==KEY1_PRES)  
    {  
        lv_ddlist_close(ddlist1,LV_ANIM_ON); //把下拉列表框强制设置为收缩状态  
    }else if(key==WKUP_PRES)  
    {  
        lv_ddlist_set_stay_open(ddlist1,true); //禁止自动返回到收缩状态  
    }  
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 初始界面效果

接着我们用手点击一下此下拉列表框,会看到如下展开状态下的效果:



图 3.4.2 展开状态下的界面效果

最后我们还可以按 KEY0, KEY1, WKUP 等按键来测试其他的功能.

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_roller 滚轮

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

lv_roller 滚轮

1. 介绍

lv_roller 是一个滚轮控件,它和我们前面学习过的 lv_ddlist 下拉列表框控件非常的相似,最大的区别在于 lv_ddlist 是以展开和收缩的方式来选择选项值的,而 lv_roller 是以滚动的方式来选择选项值的,你只需要把想要被选择的选项值滚动到中间位置即可.其外观效果图如下所示:

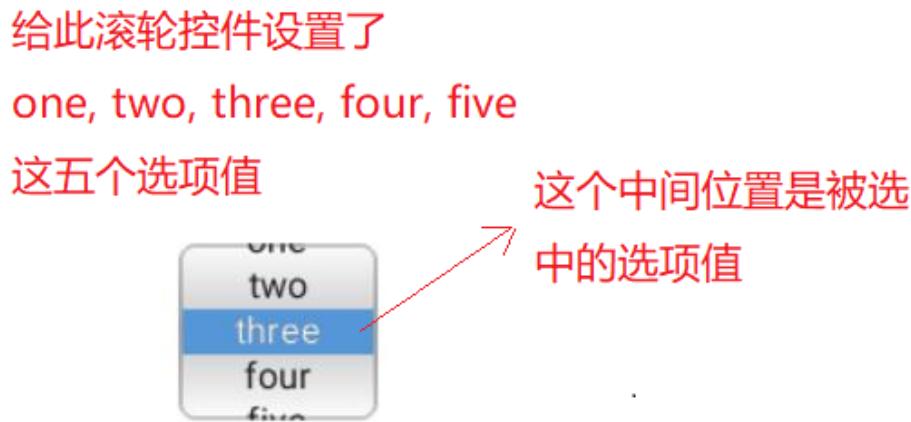


图 1.1 外观效果图

你可以通过 `lv_roller_set_options(roller, options, mode)` 这个 API 接口来给滚轮设置所有的选项值,每个选项值之间需要用\n换行符隔开,如 `const char * options = "one\ntwo\nthree";` 就给此滚动设置了 one, two, three 三个选项值,另外此 API 接口还可以附带指定滚动的模式,总共有 `LV_ROLLER_MODE_NORMAL` 正常滚动和 `LV_ROLLER_MODE_INFINITE` 循环滚动两种模式,你可以通过 `lv_roller_set_selected(roller, id)` 接口来选中一个默认的值,反之同理,通过 `lv_roller_get_selected(roller)` 接口可以获取被选中选项值的位置索引,如果你是想获取被选中的文本内容,那么你可以通过 `lv_roller_get_selected_str(roller, buf, buf_size)` 接口来实现获取.

通过 `lv_roller_set_align(roller, LV_LABEL_ALIGN_LEFT/CENTER/RIGHT)` 接口,你可以设置文本内容的水平对齐方式,通过 `lv_roller_set_visible_row_count(roller, num)` 接口,你可以设置滚轮控件的可见行数,即同一时刻有多少个选项值处于可见状态,设置好可见行数之后,滚轮控件的高度会被自动的确定下来,对于滚轮控件的宽度也是自动确定下来的,当然了,你可以通过 `lv_roller_set_fix_width(roller, width)` 接口来强制给滚轮控件设置一个固定的宽度.

最后来说一下滚轮控件的事件,普通的按下,松手等事件就不细述了,当滚轮的选项值被滚动或者被点击时, `LV_EVENT_VALUE_CHANGED` 事件将会被触发.

2. lv_roller 的 API 接口

2.1 主要数据类型

2.1.1 滚动模式数据类型

```
enum {
    LV_ROLLER_MODE_NORMAL, //正常滚动模式
    LV_ROLLER_MODE_INFINITE, //循环滚动模式
};

typedef uint8_t lv_roller_mode_t;
```

2.1.2 滚轮样式数据类型

```
enum {
    LV_ROLLER_STYLE_BG,
    LV_ROLLER_STYLE_SEL,
};

typedef uint8_t lv_roller_style_t;
```

LV_ROLLER_STYLE_BG: 用来修饰滚轮的背景,此样式中的 body 和 text 字段会被用到,其中 text 字段用来修饰未被选中的文本内容,默认值为 lv_style_pretty

LV_ROLLER_STYLE_SEL: 用来修饰被选中的选项值,此样式中的 body 和 text 字段会被用到,默认值为 lv_style_pretty

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_roller_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置所有的选项值

```
void lv_roller_set_options(lv_obj_t * roller, const char * options, lv_roller_mode_t mode);
```

参数:

roller: 滚轮对象

options: 选项值列表,每一个选项值之间需要用'\n'换行符来作为区分,如 const char * options = "First\nSecond\nThird";就定义了 First, Secode, Third 这三个选项值

mode: 滚动模式,有如下俩个选项值

LV_ROLLER_MODE_NORMAL: 正常滚动模式

LV_ROLLER_MODE_INFINITE: 循环滚动模式

2.2.3 选中某个选项值

```
void lv_roller_set_selected(lv_obj_t * roller, uint16_t sel_opt, lv_anim_enable_t anim);
```

参数:

roller: 滚轮对象

sel_opt: 选项值的索引值,从 0 开始

anim: 是否附带动画效果,有如下俩个可选值

LV_ANIM_OFF: 不附带动画效果

LV_ANIM_ON: 附带动画效果

2.2.4 设置可见行的个数

```
void lv_roller_set_visible_row_count(lv_obj_t * roller, uint8_t row_cnt);
```

参数:

roller: 滚轮对象

row_cnt: 可见行的个数,也是指在同一时刻下可见选项值的个数

设置好可见行的个数之后,滚轮控件的高度也会被自动的确定下来

2.2.5 设置滚轮的宽度

```
static inline void lv_roller_set_fix_width(lv_obj_t * roller, lv_coord_t w);
```

参数:

roller: 滚轮对象

w: 宽度值

如果不设置宽度的话,也是没有关系的,滚轮控件会根据选项值的最大文本长度来自动确定宽度的

2.2.6 设置动画时长

```
static inline void lv_roller_set_anim_time(lv_obj_t * roller, uint16_t anim_time);
```

参数:

roller: 滚轮对象

anim_time: 展开和收缩动画的时长,单位 ms

2.2.7 设置样式

```
void lv_roller_set_style(lv_obj_t * roller, lv_roller_style_t type, const lv_style_t * style);
```

参数:

roller: 滚轮对象

type: 设置哪一部分的样式,有如下 3 个可选值

LV_ROLLER_STYLE_BG: 修饰滚轮控件的背景

LV_ROLLER_STYLE_SEL: 修饰被选中的选项值

style: 样式

2.2.8 设置文本的水平对齐方式

```
void lv_roller_set_align(lv_obj_t * roller, lv_label_align_t align);
```

参数:

roller: 滚轮对象

align: 水平对齐方式

2.2.9 获取当前选择的文本内容

```
static inline void lv_roller_get_selected_str(const lv_obj_t * roller, char * buf, uint16_t buf_size);
```

参数:

roller: 滚轮对象

buf: 用来存放选项值的文本内容

buf_size: buf 缓冲区的大小

2.2.10 获取当前选择的选项值的索引

```
uint16_t lv_roller_get_selected(const lv_obj_t * roller);
```

参数:

roller: 滚轮对象

返回值:

返回当前选择的选项值的索引值,从 0 开始的

2.2.11 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

创建2个自定义样式,一个用来修饰滚轮的背景,一个用来修饰被选中时的选项值,接着创建一个滚轮对象,设置它的固定宽度,设置它的所有选项值,设置它的可见行数,最后给其设置事件回调函数,在事件回调函数中,通过串口打印出当前被选中的文本内容.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏
- 2) 串口

3.3 软件设计

在 GUI_APP 目录下创建 lv_roller_test.c 和 lv_roller_test.h 俩个文件,其中 lv_roller_test.c 文件的内容如下:

```
#include "lv_roller_test.h"
#include "lvgl.h"
#include <stdio.h>

lv_style_t bg_style;
lv_style_t sel_style;

//事件回调函数
void event_handler(lv_obj_t * obj,lv_event_t event)
{
    char txt[32];
    if(event==LV_EVENT_VALUE_CHANGED)
    {
        lv_roller_get_selected_str(obj,txt,sizeof(txt));
        printf("Selected text: %s\r\n",txt);
    }
}
```

```
}

//例程入口
void lv_roller_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1. 创建样式
    //1.1 创建背景样式
    lv_style_copy(&bg_style,&lv_style_plain);
    bg_style.body.main_color = LV_COLOR_WHITE; //纯白色背景
    bg_style.body.grad_color = bg_style.body.main_color;
    bg_style.body.border.width = 1; //边框宽度
    bg_style.body.border.color = LV_COLOR_MAKE(0xAA,0xAA,0xAA);
    //LV_COLOR_MAKE(0x30,0x30,0x30); //边框颜色
    bg_style.body.padding.left = 10; //设置左侧的内边距
    bg_style.text.color = LV_COLOR_BLACK; //文本颜色
    bg_style.body.shadow.color = bg_style.body.border.color; //阴影颜色
    bg_style.body.shadow.width = 4; //阴影宽度

    //1.2 创建选择项被选中时的样式
    lv_style_copy(&sel_style,&lv_style_plain);
    sel_style.body.main_color = LV_COLOR_MAKE(0x5F,0xB8,0x78); //浅绿色背景
    sel_style.body.grad_color = sel_style.body.main_color;
    sel_style.text.color = LV_COLOR_WHITE; //文本为白色

    //2. 创建滚轮对象
    lv_obj_t * roller1 = lv_roller_create(scr,NULL);
    lv_roller_set_options(roller1,"Shanghai\nBeijing\nShenzhen\nGuangzhou\nHangzhou\nNanchang",LV_ROLLER_MODE_INFINITE); //设置所有的选项值，循环滚动模式
    lv_roller_set_selected(roller1,3,LV_ANIM_OFF); //设置默认选中值为 Guangzhou
    lv_roller_set_fix_width(roller1,140); //设置固定宽度
    lv_roller_set_visible_row_count(roller1,4); //设置可见的行数
    lv_roller_set_style(roller1,LV_ROLLER_STYLE_BG,&bg_style); //设置背景样式
    lv_roller_set_style(roller1,LV_ROLLER_STYLE_SEL,&sel_style); //设置背景样式
    lv_obj_set_event_cb(roller1,event_handler); //注册事件回调函数
    lv_obj_align(roller1,NULL,LV_ALIGN_CENTER,0,0); //设置与屏幕居中对齐
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 初始化界面效果

然后我们可以用手滑动滚轮,滑动效果如下所示:



图 3.4.2 滚动效果

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。

请关注正点原子弹公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子弹公众号

正点原子 littleVGL 开发指南

lv_canvas 画布

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

lv_canvas 画布

1. 介绍

lv_canvas 是一个画布控件,你可以在它上面绘制任意图形,以及进行旋转等操作,但是此 lv_canvas 画布控件必须得依赖于一个 buffer 缓冲区,此 buffer 缓冲区必须得是全局的或者静态的,即只要保证在使用此画布控件期间,此 buffer 缓冲区不会被释放掉就可以了,可以通过 lv_canvas_set_buffer(canvas, buffer, width, height, cf) 接口来给画布控件设置一个 buffer 缓冲区,此接口同时指定了画布绘图区域的宽和高,以及画布的颜色格式,注意了,这里传入的 width 和 height 参数是用来指定画布绘图区域的宽和高,并不等价于画布控件自身的宽和高,画布控件的宽和高是通过 lv_obj_set_size 接口来设置的,默认情况下,画布控件自身的大小是等于绘图区域的大小的,当设置画布控件的大小与绘图区域的大小不相等时,就会出现截取显示或重复填充显示等情况,为了方便大家理解,我们假定画布的绘图区域大小为 200*150,里面只绘制了一个”Hello world”文本和一个红色填充矩形,那么三种显示情况如下所示:

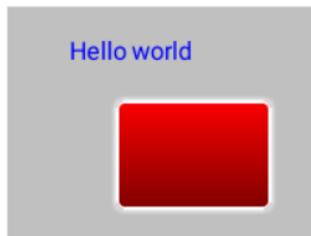


图 1.1 正常显示情况(画布控件的大小默认为 200*150)

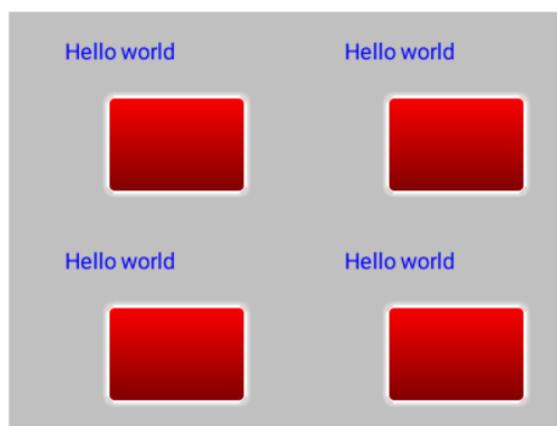


图 1.2 重复填充显示情况(画布控件的大小为 400*300)

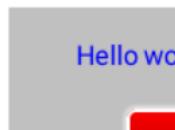


图 1.3 截取显示情况(画布控件的大小为 100*75)

而最后的 cf 参数是用来指定画布颜色格式的,比如 LV_IMG_CF_TRUE_COLOR 等真彩色格式和 LV_IMG_CF_INDEXED_1BIT 等调色板格式,这里的颜色格式和图片在线转换工具中的颜色格式含义是一样的.当画布控件的颜色格式选择为调色板格式时,会比选择真彩色格

式多出一个步骤,因为调色板格式需要先构建出一个调色板,对于 LV_IMG_CF_INDEXED_1 BIT 调色板格式,调色板中有 2 种颜色,对于 LV_IMG_CF_INDEXED_2BIT 调色板格式,调色板中有 4 种颜色,后面的调色板格式以此类推,所谓的构建调色板,就是对其里面的每一种颜色赋具体的颜色值,它是通过 lv_canvas_set_palette(canvas, id, color) 接口来完成的,按理来说,调色板中有多少种颜色,那么此接口就需要被调用多少次,不论是真彩色格式还是调色板格式,最后都可以通过 lv_canvas_set_px(canvas, x, y, color) 接口来设置任意像素点的颜色值,只不过当为调色板格式时,color 参数传入的是某颜色在调色板中的位置索引值,当为真彩色格式时,color 参数传入的是真正的颜色值.

从图 1.1 中,我们可以看到绘图区域的背景颜色是银色的,其实它是通过 lv_canvas_fill_bg (canvas, color) 接口来设置的,默认情况下,绘图区域背景是黑色的,除此之外,画布控件还给我们提供了一些其他的绘图 API 接口,如下所示:

1) 在画布上绘制矩形

```
lv_canvas_draw_rect(canvas, x, y, width, height, &style);
```

2) 在画布上绘制文本内容

```
lv_canvas_draw_text(canvas, x, y, max_width, &style, txt,  
LV_LABEL_ALIGN_LEFT/CENTER/RIGTH);
```

3) 在画布上绘制图片

```
lv_canvas_draw_img(canvas, x, y, &img_src, &style)
```

4) 在画布上绘制线条

```
lv_canvas_draw_line(canvas, point_array, point_cnt, &style)
```

5) 在画布上绘制多边形

```
lv_canvas_draw_polygon(canvas, points_array, point_cnt, &style)
```

6) 在画布上绘制弧形

```
lv_canvas_draw_arc(canvas, x, y, radius, start_angle, end_angle, &style)
```

7) 在画布上绘制一个经过旋转后的图片

```
lv_canvas_rotate(canvas, &img_dsc, angle, x, y, pivot_x, pivot_y)
```

2. lv_canvas 的 API 接口

2.1 主要数据类型

2.1.1 画布样式数据类型

```
enum {
    LV_CANVAS_STYLE_MAIN,
};

typedef uint8_t lv_canvas_style_t;
```

此样式中的 image.color 字段用来描述 LV_IMG_CF_ALPHA_...颜色格式下的基色,除此之外,基本用不到的,了解即可

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_canvas_create(lv_obj_t * par, const lv_obj_t * copy);
```

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 设置缓冲区

```
void lv_canvas_set_buffer(lv_obj_t * canvas, void * buf, lv_coord_t w, lv_coord_t h,
lv_img_cf_t cf);
```

参数:

canvas: 画布对象

buf: 缓冲区,必须得保证在画布使用期间,此缓冲区不能被释放了

w: 画布绘图区域的宽度

h: 画布绘图区域的高度

cf: 画布的颜色格式

请注意,上面的 w 和 h 是指绘图区域的宽和高,和画布控件自身的宽高并不是同一个概念,同时请保证 buf 缓冲区的大小和传入的 w, h 参数保持一致性,假定 buf 缓冲区的大小为 buf_size,那么它应该满足如下公式:

```
buf_size = LV_CANVAS_BUF_SIZE_...(w,h);
```

其中 LV_CANVAS_BUF_SIZE_...是一个和 cf 颜色格式相关的宏,此宏的所有可能值如下:

```
LV_CANVAS_BUF_SIZE_TRUE_COLOR  
LV_CANVAS_BUF_SIZE_TRUE_COLOR_CHROMA_KEYED  
LV_CANVAS_BUF_SIZE_TRUE_COLOR_ALPHA  
LV_CANVAS_BUF_SIZE_INDEXED_1BIT  
LV_CANVAS_BUF_SIZE_INDEXED_2BIT  
LV_CANVAS_BUF_SIZE_INDEXED_4BIT  
LV_CANVAS_BUF_SIZE_INDEXED_8BIT
```

假如 cf 参数为 LV_IMG_CF_TRUE_COLOR 时,那么 buf 缓冲区的定义应该如下所示:

```
static lv_color_t buf[LV_CANVAS_BUF_SIZE_TRUE_COLOR(w, h)];
```

2.2.3 设置某像素的颜色值

```
void lv_canvas_set_px(lv_obj_t * canvas, lv_coord_t x, lv_coord_t y, lv_color_t c);
```

参数:

canvas: 画布对象

x: 像素的 x 坐标,是以画布控件的左上角为参考原点的

y: 像素的 y 坐标,是以画布控件的左上角为参考原点的

c: 像素的颜色值

请注意,当画布选择的 cf 颜色格式为 LV_IMG_CF_INDEXED_1BIT 等调色板格式时,c 参数传入的是某颜色在调色板中的位置索引值,当为 LV_IMG_CF_TRUE_COLOR 等真彩色格式时,c 参数传入的才是真正颜色值.

2.2.4 构建调色板

```
void lv_canvas_set_palette(lv_obj_t * canvas, uint8_t id, lv_color_t c);
```

参数:

canvas: 画布对象

id: 给调色板中的哪一个位置赋颜色值,从 0 开始,id 最大值等于此调色板的颜色种类数减 1

c: 颜色值

只有当画布选择的 cf 颜色格式为 LV_IMG_CF_INDEXED_1BIT(有 2 种颜色),
LV_IMG_CF_INDEXED_2BIT(有 4 种颜色), LV_IMG_CF_INDEXED_4BIT(有 16 种颜色),
LV_IMG_CF_INDEXED_8BIT(有 256 种颜色)中的一个时,此 API 接口才有效,选择不同的调色板格式,其具有的颜色种类数也是不同的,如果想要实现绘制透明的效果,只需要给调色板中的某个位置赋一个 LV_COLOR_TRANS 透明色,然后在想要透明的地方,调用这种颜色就可以了

2.2.5 设置样式

```
void lv_canvas_set_style(lv_obj_t * canvas, lv_canvas_style_t type, const lv_style_t * style);
```

参数:

canvas: 画布对象

type: 设置哪一部分的样式,目前只有 LV_CANVAS_STYLE_MAIN 这一个可选值

style: 样式

2.2.6 获取画布所对应的图片源

```
lv_img_dsc_t * lv_canvas_get_img(lv_obj_t * canvas);
```

参数:

canvas: 画布对象

返回值:

返回画布所对应的图片源

拿到此图片源后,我们可以利用 lv_img 图片章节中的 lv_img_set_src 接口把此图片给显示出来,显示出来的效果和画布中的效果是一模一样的,例子如下所示:

```
lv_obj_t * img = lv_img_create(lv_scr_act(),NULL);//创建图片对象  
lv_img_dsc_t * img_src = lv_canvas_get_img(canvas);//获取画布所对应的图片源  
lv_img_set_src(img, img_src);//设置图片源
```

2.2.7 在画布中绘制矩形

```
void lv_canvas_draw_rect(lv_obj_t * canvas, lv_coord_t x, lv_coord_t y, lv_coord_t w,  
lv_coord_t h, const lv_style_t * style);
```

参数:

canvas: 画布对象

x: 矩形的 x 坐标,是以画布控件的左上角为参考原点的

y: 矩形的 y 坐标,是以画布控件的左上角为参考原点的

w: 矩形的宽度

h: 矩形的高度

style: 用于修饰矩形的样式

是否为填充的矩形,这取决于传入的 style 样式

2.2.8 在画布中绘制文本内容

```
void lv_canvas_draw_text(lv_obj_t * canvas, lv_coord_t x, lv_coord_t y, lv_coord_t max_w,  
const lv_style_t * style, const char * txt, lv_label_align_t align);
```

参数:

canvas: 画布对象
x: 文本内容的 x 坐标,是以画布控件的左上角为参考原点的
y: 文本内容的 y 坐标,是以画布控件的左上角为参考原点的
max_w: 最大宽度,当文本内容的长度超过此大小时,会自动换行的
style: 用于修饰文本内容的样式
txt: 文本内容
align: 文本内容在指定的 max_w 宽度区域内的水平对齐方式

2.2.9 在画布中绘制图片

```
void lv_canvas_draw_img(lv_obj_t * canvas, lv_coord_t x, lv_coord_t y, const void * src,  
const lv_style_t * style);
```

参数:

canvas: 画布对象
x: 图片的 x 坐标,是以画布控件的左上角为参考原点的
y: 图片的 y 坐标,是以画布控件的左上角为参考原点的
src: 图片源
style: 用于修饰图片的样式

2.2.10 在画布中绘制线条

```
void lv_canvas_draw_line(lv_obj_t * canvas, const lv_point_t * points, uint32_t point_cnt,  
const lv_style_t * style);
```

参数:

canvas: 画布对象
points: 用于构成线条的点集合
point_cnt: 点的个数
style: 用于修饰线条的样式

2.2.11 在画布中绘制多边形

```
void lv_canvas_draw_polygon(lv_obj_t * canvas, const lv_point_t * points, uint32_t  
point_cnt, const lv_style_t * style);
```

参数:

canvas: 画布对象
points: 用于构成多边形的点集合
point_cnt: 点的个数

style: 用于修饰多边形的样式

是否为填充的多边形,这取决于传入的 style 样式

2.2.12 在画布中绘制弧形

```
void lv_canvas_draw_arc(lv_obj_t * canvas, lv_coord_t x, lv_coord_t y, lv_coord_t r, int32_t start_angle, int32_t end_angle, const lv_style_t * style);
```

参数:

canvas: 画布对象

x: 弧形的圆心 x 坐标,是以画布控件的左上角为参考原点的

y: 弧形的圆心 y 坐标,是以画布控件的左上角为参考原点的

r: 弧形的半径

start_angle: 弧形的起始角度,范围为[0,360]

end_angle: 弧形的终止角度,范围为[0,360]

style: 用于修饰弧形的样式

是以底边正中间点为 0 度,然后逆时针方向是增加度数,绘制弧形时,也是按逆时针方向从起点绘制到终点的

2.2.13 在画布中绘制旋转后的图片

```
void lv_canvas_rotate(lv_obj_t * canvas, lv_img_dsc_t * img, int16_t angle, lv_coord_t offset_x, lv_coord_t offset_y, int32_t pivot_x, int32_t pivot_y);
```

参数:

canvas: 画布对象

img: C 数组方式的图片源,对于图标字体,外部文件等方式的图片源是不支持的

angle: 顺时针旋转的角度

offset_x: 图片的 x 坐标,是以画布控件的左上角为参考原点的

offset_y: 图片的 y 坐标,是以画布控件的左上角为参考原点的

pivot_x: 旋转点的 x 坐标,是以画布控件的左上角为参考原点的

pivot_y: 旋转点的 y 坐标,是以画布控件的左上角为参考原点的

为了大家能更直观的理解,举一个例子,我们这里就直接把画布的绘图区域当成图片,旋转 60 度后,再绘制到画布中去,当然了,你也可以用图片在线转换工具来生成一张图片进行实验,示意代码如下:

```
#define CANVAS_WIDTH 200
#define CANVAS_HEIGHT 150
lv_color_t buf[LV_CANVAS_BUF_SIZE_TRUE_COLOR(CANVAS_WIDTH, CANVAS_HEIGHT)];
lv_obj_t * canvas = lv_canvas_create(lv_scr_act(), NULL);
lv_canvas_set_buffer(canvas, buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_IMG_CF_TRUE_COLOR);
```

```
lv_canvas_fill_bg(canvas, LV_COLOR_GRAY); //将画布的背景清成灰色  
lv_canvas_draw_rect(canvas, 50, 40, 100, 70, &style); //在画布中绘制一个红色的填充矩形
```

此时得到的效果如下图所示：

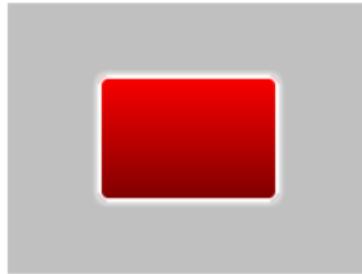


图 2.2.13.1 绘图区域未旋转之前的效果

```
//重点,接着把画布的绘图区域生成一个图片源  
lv_color_t buf_tmp[CANVAS_WIDTH * CANVAS_HEIGHT];  
memcpy(buf_tmp, buf, sizeof(buf_tmp));  
lv_img_dsc_t img_src;  
img_src.data = (void *)buf_tmp;  
img_src.header.cf = LV_IMG_CF_TRUE_COLOR;  
img_src.header.w = CANVAS_WIDTH;  
img_src.header.h = CANVAS_HEIGHT;  
  
lv_canvas_fill_bg(canvas, LV_COLOR_GRAY); //将画布的背景再次清成灰色  
//选择 60 度后,再绘制到画布中去  
lv_canvas_rotate(canvas, & img_src, 60, 0, 0, CANVAS_WIDTH / 2, CANVAS_HEIGHT /  
2);
```

旋转后得到的效果如下图所示：

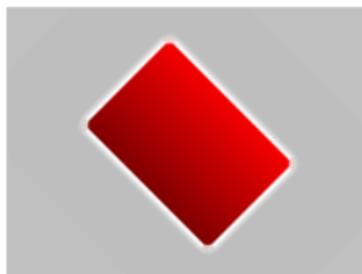


图 2.2.13.2 绘图区域旋转后的效果

2.2.14 将某个绘图缓冲区拷贝到画布中

```
void lv_canvas_copy_buf(lv_obj_t * canvas, const void * to_copy, lv_coord_t x, lv_coord_t  
y, lv_coord_t w, lv_coord_t h);
```

参数：

[开发指南](#)

www.alientek.com

canvas: 画布对象
to_copy: 绘图缓冲区
x: 画布中的 x 坐标
y: 画布中的 y 坐标
w: to_copy 绘图缓冲区对应的宽度
h: to_copy 绘图缓冲区对应的高度

这个 API 接口的作用就是将 to_copy 绘图缓冲区拷贝到 canvas 画布中的(x,y)位置上去。但是有一个前提就是必须得保证 to_copy 绘图缓冲区的颜色格式和 canvas 画布的颜色格式保持一致,下面举一个简单的例子,示意代码如下所示:

```
//画布 1 绘图区域的大小
#define CANVAS1_WIDTH 100
#define CANVAS1_HEIGHT 100
//画布 2 绘图区域的大小
#define CANVAS2_WIDTH 150
#define CANVAS2_HEIGHT 150

//创建画布 1
lv_color_t buf1[LV_CANVAS_BUF_SIZE_TRUE_COLOR(CANVAS1_WIDTH, CANVAS1_HEIGHT)];
lv_obj_t * canvas1 = lv_canvas_create(lv_scr_act(), NULL);
lv_canvas_set_buffer(canvas1,buf1,CANVAS1_WIDTH,CANVAS1_HEIGHT, LV_IMG_CF_TRUE_COLOR);

//在画布 1 上绘制一个 Hello 文本内容
lv_canvas_draw_text(canvas1, 10, 10, CANVAS1_WIDTH, &style, "Hello", LV_LABEL_ALIGN_LEFT);

//创建画布 2
lv_color_t buf2[LV_CANVAS_BUF_SIZE_TRUE_COLOR(CANVAS2_WIDTH, CANVAS2_HEIGHT)];
lv_obj_t * canvas2 = lv_canvas_create(lv_scr_act(), NULL);
lv_canvas_set_buffer(canvas2,buf2,CANVAS2_WIDTH,CANVAS2_HEIGHT, LV_IMG_CF_TRUE_COLOR);

//把画布 1 的绘图缓冲区拷贝到画布 2 中的(20,20)坐标上
lv_canvas_copy_buf(canvas2, buf1,20,20, CANVAS1_WIDTH, CANVAS1_HEIGHT);
```

2.2.15 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3.例程设计

3.1 功能简介

先自定义一个样式,用来修饰画布中的各种绘图形状,然后接着创建2个画布控件,第一个画布控件主要是用来演示各种绘图接口的,比如画矩形,画文本内容,画弧形,画线条等,而第二个画布控件主要是用来演示调色板格式的.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_canvas_test.c 和 lv_canvas_test.h 俩个文件,其中 lv_canvas_test.c 文件的内容如下:

```
#include "lv_canvas_test.h"
#include "lvgl.h"

lv_style_t style;
const lv_point_t points[] = {{50,90},{70,50},{90,90}};//构成线条的点集合

//画布 1,用来演示一些绘图 API 接口
#define CANVAS1_WIDTH    100 //画布 1 的宽度
#define CANVAS1_HEIGHT   100 //画布 1 的高度
//因为内部 sram 不够用,所以我们把画布的缓存区定义在外部 ram 上,注意需要跳过 lvgl
//的帧缓存区
lv_color_t canvas1_buf[LV_CANVAS_BUF_SIZE_TRUE_COLOR(CANVAS1_WIDTH,C
ANVAS1_HEIGHT)] __attribute__((at(0X68000000+LV_HOR_RES_MAX*LV_VER_RES_
MAX*2)));

//画布 2,用来演示调色板格式
#define CANVAS2_WIDTH    40 //画布 2 的宽度
#define CANVAS2_HEIGHT   40 //画布 2 的高度
```

```
//画布 2 比较小,所以它的缓冲区可以直接定义在内部 sram 上
lv_color_t canvas2_buf[LV_CANVAS_BUF_SIZE_INDEXED_1BIT(CANVAS2_WIDTH,
CANVAS2_HEIGHT)];

//例程入口
void lv_canvas_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1.创建样式
    lv_style_copy(&style,&lv_style_plain);
    style.body.main_color = LV_COLOR_RED;
    style.body.grad_color = LV_COLOR_RED;
    style.line.width = 2;
    style.line.color = LV_COLOR_GREEN;
    style.text.color = LV_COLOR_BLUE;

    //2.创建画布 1,用来演示一些绘图 API 接口
    lv_obj_t * canvas1 = lv_canvas_create(scr,NULL);
    lv_canvas_set_buffer(canvas1,canvas1_buf,CANVAS1_WIDTH,CANVAS1_HEIGHT,
LV_IMG_CF_TRUE_COLOR); //设置缓冲区,真彩色格式
    //设置与屏幕的对齐方式
    lv_obj_align(canvas1,NULL,LV_ALIGN_IN_TOP_MID,0,20);
    lv_canvas_fill_bg(canvas1,LV_COLOR_GRAY); //将背景清成灰色
    lv_canvas_draw_rect(canvas1,10,10,60,20,&style); //绘制一个红色的填充矩形
    lv_canvas_draw_text(canvas1,0,35,CANVAS1_WIDTH,&style,"Hello",LV_LABEL_ALIGN_CENTER); //绘制文本内容
    lv_canvas_draw_arc(canvas1,30,60,20,270,90,&style); //绘制弧形
    //绘制线条
    lv_canvas_draw_line(canvas1,points,sizeof(points)/sizeof(lv_point_t),&style);

    //3.创建画布 2,用来演示调色板格式
    lv_obj_t * canvas2 = lv_canvas_create(scr,NULL);
    lv_canvas_set_buffer(canvas2,canvas2_buf,CANVAS2_WIDTH,CANVAS2_HEIGHT,
LV_IMG_CF_INDEXED_1BIT); //设置缓冲区,真彩色格式
    //设置与画布 1 的对齐方式
    lv_obj_align(canvas2,canvas1,LV_ALIGN_OUT_BOTTOM_MID,0,20);
    //构建调色板,对于 LV_IMG_CF_INDEXED_1BIT 颜色格式而言,总共有 2 种颜色
    lv_canvas_set_palette(canvas2,0,LV_COLOR_GREEN); //第一种为绿色
    //第二种为红色,也可以换成 LV_COLOR_TRANSP 透明色看看效果哦
    lv_canvas_set_palette(canvas2,1,LV_COLOR_RED);
    //定义 2 个颜色
    lv_color_t color0;
    lv_color_t color1;
```

```
color0.full = 0;//指向调色板中的第一种颜色  
color1.full = 1;//指向调色板中的第二种颜色
```

```
lv_canvas_fill_bg(canvas2,color0);//把背景填充成绿色  
//在画布 2 的正中间绘制一个 20*20 大小的矩形  
uint16_t x,y;  
for(y=10;y<30;y++)  
{  
    for(x=10;x<30;x++)  
    {  
        lv_canvas_set_px(canvas2,x,y,color1);  
    }  
}  
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

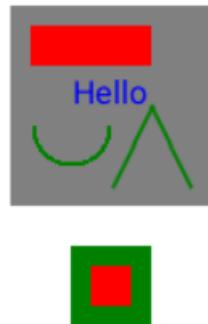


图 3.4.1 初始界面效果

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



扫码关注正点原原子公众号

正点原子 littleVGL 开发指南

lv_theme 主题

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

lv_theme 主题

1. 介绍

lv_theme 主题是 littleVGL 图形库内置的一个特色功能,在我们的项目中直接使用 lv_theme 主题可以使我们的 UI 界面比较美观而且色调统一,最主要的原因是可以大大缩短我们项目的开发周期,因为我们不需要再去重新定义 UI 控件的样式了,这可以使我们更加专注地开发项目的业务逻辑,而对于那些 UI 控件样式,lv_theme 主题已经为我们全部做好了,我们只需要从其中挑选我们满意的主题,然后直接应用到项目中即可.

所谓的主题,就是 littleVGL 为内置的所有控件做了一层 UI 样式的封装,相当于把某个控件创建出来之后,此控件就具有了指定主题中的样式外观,当然了你可以在后续的操作中用自己自定义的样式来覆盖掉其原有的主题样式,littleVGL 从清爽,简约的审美观出发,总共为我们制订了 8 种主题,这些主题的界面效果分别如下图所示:

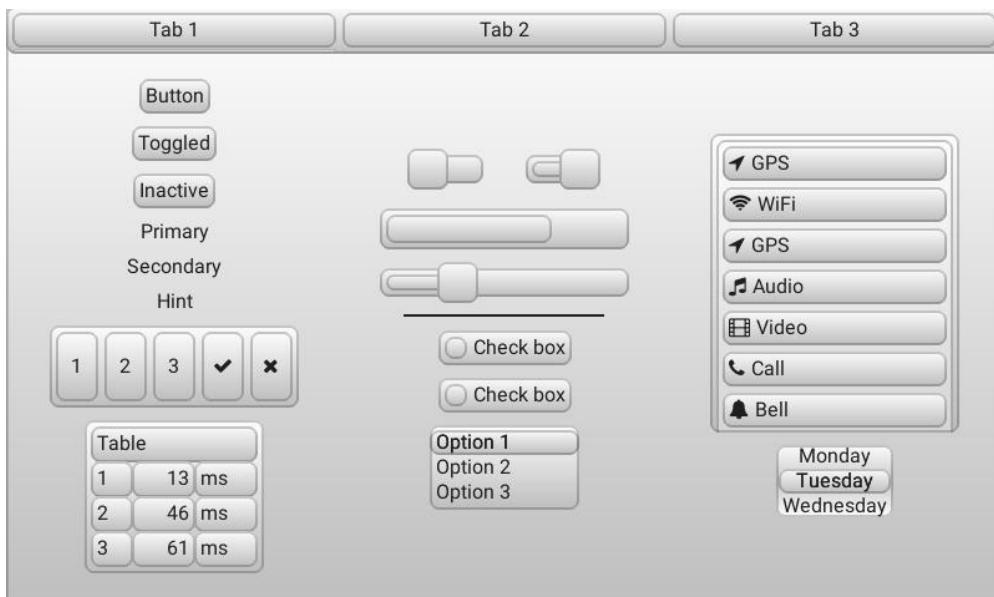


图 1.1 templ 主题

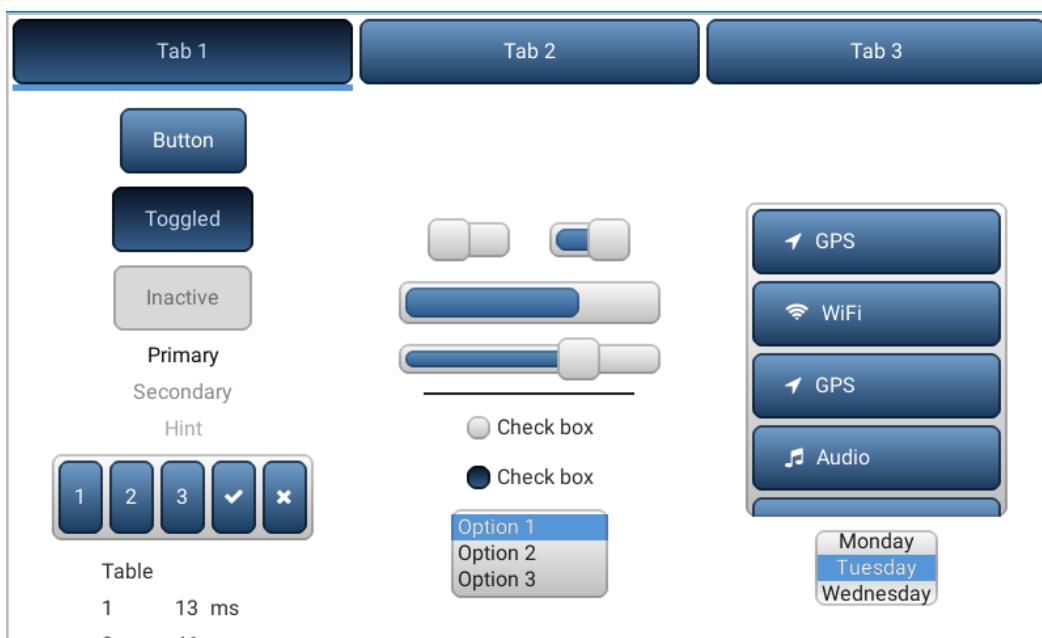


图 1.2 default 主题

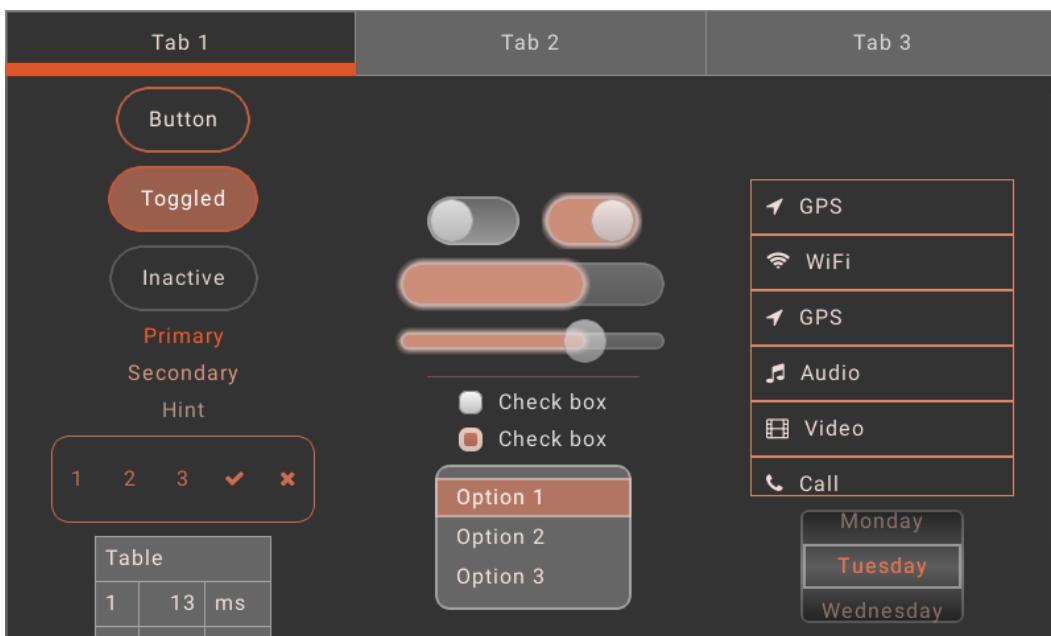


图 1.3 alien 主题



图 1.4 night 主题

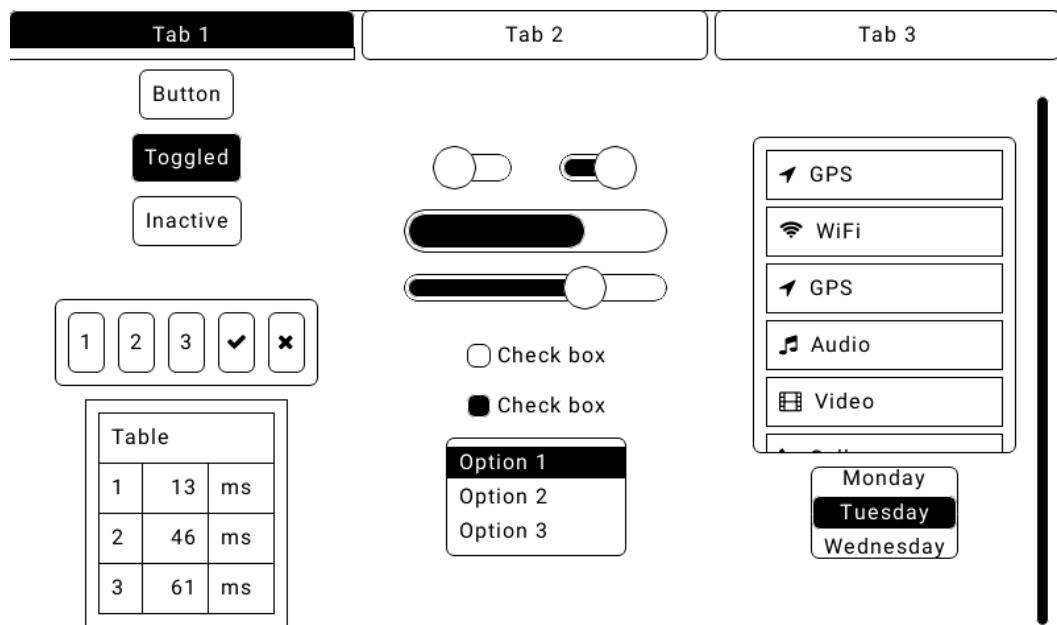


图 1.5 mono 主题

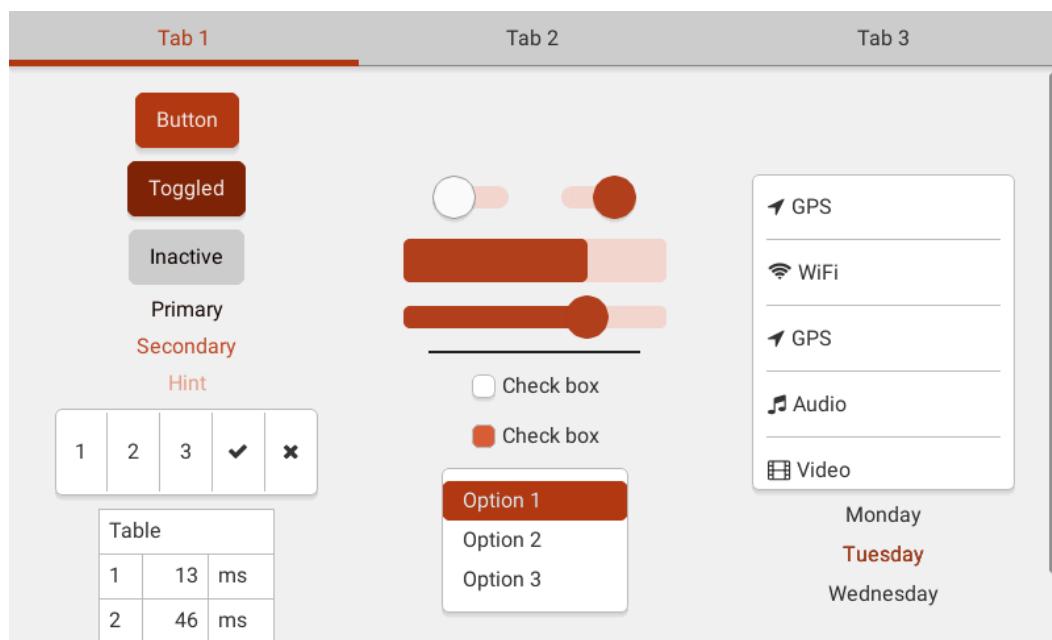


图 1.6 material 主题

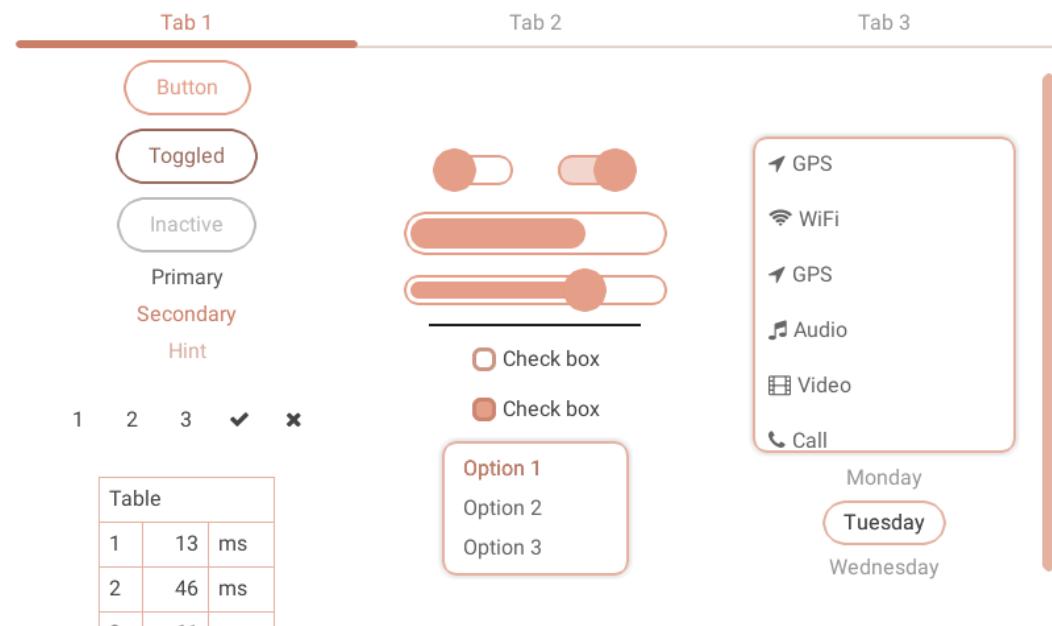


图 1.7 zen 主题

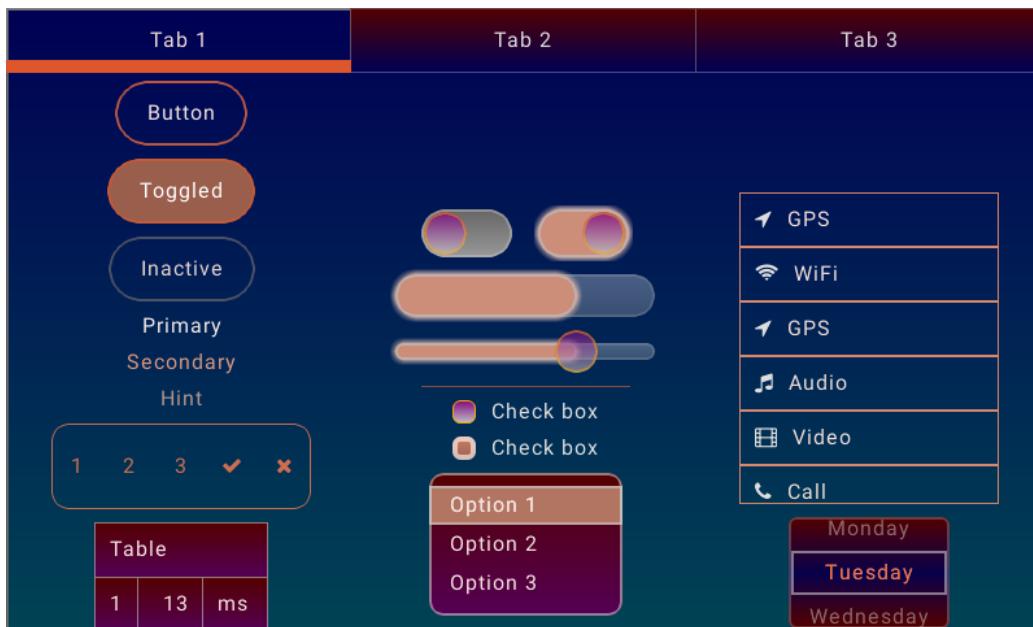


图 1.8 nemo 主题

对于上面的 8 种主题,默认情况下是没有被使能的,你需要在 lv_conf.h 配置文件中,把你所需要的主題给使能了,而不需要的主题应该继续保持失能状态即可,因为这样可以减少 flash 和 ram 容量的消耗,这 8 种主题在 lv_conf.h 配置文件中所对应的配置宏分别如下:

```
#define LV_USE_THEME_TEMPIL          0 //是否使能 templ 主题
#define LV_USE_THEME_DEFAULT          0 //是否使能 default 主题
#define LV_USE_THEME_ALIEN            0 //是否使能 alien 主题
#define LV_USE_THEME_NIGHT             0 //是否使能 night 主题
#define LV_USE_THEME_MONO              0 //是否使能 mono 主题
#define LV_USE_THEME_MATERIAL          0 //是否使能 material 主题
#define LV_USE_THEME_ZEN               0 //是否使能 zen 主题
#define LV_USE_THEME_NEMO              0 //是否使能 nemo 主题
```

除了上面的 8 个配置宏外,还有一个 LV_THEME_LIVE_UPDATE 配置宏,它是用来定义是否允许在运行时动态切换主题的,如果使能了动态切换主题功能的话,它将会增加 8-10KB 左右的 ram 消耗,如下所示:

```
#define LV_THEME_LIVE_UPDATE         0 //是否允许动态切换主题
```

对于上面的 9 个配置宏,都是 1 代表使能,0 代表失能.

我们已经介绍了 littleVGL 中到底具有多少种主题,那么我们应该如何来使用某个主题呢? 这也是很简单的,只需要二个步骤即可,一是先创建主题,二是接着使用此主题,通过如下接口就可以来创建相应的主题:

```
lv_theme_t * theme = lv_theme_tempil_init(hue,font);      //创建 templ 主题
lv_theme_t * theme = lv_theme_default_init(hue,font);      //创建 default 主题
lv_theme_t * theme = lv_theme_alien_init(hue,font);        //创建 alien 主题
lv_theme_t * theme = lv_theme_night_init(hue,font);        //创建 night 主题
lv_theme_t * theme = lv_theme_mono_init(hue,font);         //创建 mono 主题
```

```
lv_theme_t * theme = lv_theme_material_init(hue,font); //创建 material 主题  
lv_theme_t * theme = lv_theme_zen_init(hue,font); //创建 zen 主题  
lv_theme_t * theme = lv_theme_nemo_init(hue,font); //创建 nemo 主题
```

通过上面的某个创建主题接口把 theme 主题创建出来之后,就可以接着通过
`lv_theme_set_current(theme)`接口把此主题给使用起来了.

2. lv_theme 的 API 接口

2.1 主要数据类型

3. 主题数据类型

```
typedef struct
{
    struct
    {
        lv_style_t * scr;
        lv_style_t * bg;
        lv_style_t * panel;

#if LV_USE_CONT != 0
        lv_style_t * cont;
#endif

#if LV_USE_BTN != 0
        struct
        {
            lv_style_t * rel;
            lv_style_t * pr;
            lv_style_t * tgl_rel;
            lv_style_t * tgl_pr;
            lv_style_t * ina;
        } btn;
#endif

... //因为文本内容过长,此处省略掉其他控件的样式结构体
    } style;

#if LV_USE_GROUP
    struct
    {
        lv_group_style_mod_cb_t style_mod_xcb;
        lv_group_style_mod_cb_t style_mod_edit_xcb;
    } group;
#endif
} lv_theme_t;
```

lv_theme_t 是主题所对应的数据类型,它里面就是对所有控件样式的一个封装,了解即可

2.2 API 接口

2.2.1 创建主题

```
lv_theme_t * lv_theme_temp_init(uint16_t hue, lv_font_t * font); //创建 templ 主题  
lv_theme_t * lv_theme_default_init(uint16_t hue, lv_font_t * font); //创建 default 主题  
lv_theme_t * lv_theme_alien_init(uint16_t hue, lv_font_t * font); //创建 alien 主题  
lv_theme_t * lv_theme_night_init(uint16_t hue, lv_font_t * font); //创建 night 主题  
lv_theme_t * lv_theme_mono_init(uint16_t hue, lv_font_t * font); //创建 mono 主题  
lv_theme_t * lv_theme_material_init(uint16_t hue, lv_font_t * font); //创建 material 主题  
lv_theme_t * lv_theme_zen_init(uint16_t hue, lv_font_t * font); //创建 zen 主题  
lv_theme_t * lv_theme_nemo_init(uint16_t hue, lv_font_t * font); //创建 nemo 主题
```

参数:

hue: HSV 颜色格式,用来定义此主题的基色,范围为[0,360]

font: 此主题所使用的默认字体,如果传 NULL 的话,则代表使用 lv_conf.h 文件中的

LV_FONT_DEFAULT 宏所定义的默认字体

返回值:

返回创建出来的主题

2.2.2 使用主题

```
void lv_theme_set_current(lv_theme_t * th);
```

参数:

th: 想要被使用的主题

2.2.3 获取当前正在被使用的主题

```
lv_theme_t * lv_theme_get_current(void);
```

返回值:

返回当前正在被使用的主题

3.例程设计

3.1 功能简介

创建一个 night 主题,并且使用此 night 主题,然后我们在屏幕上创建一个 tabview 选项卡控件来观察主题的界面效果,此 tabview 选项卡控件上有 tab1 和 tab2 两个子选项卡,我们往 tab1 选项卡上添加了按钮,矩阵按钮,开关,进度条,滑块,下拉列表框等 6 种控件,往 tab2 选项卡上添加了列表,滚轮等 2 种控件.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

在 GUI_APP 目录下创建 lv_theme_test.c 和 lv_theme_test.h 两个文件,其中 lv_theme_test.c 文件的内容如下:

```
#include "lv_theme_test.h"
#include "lvgl.h"

const char * const bnm_str[] = {"1", "2", "3", LV_SYMBOL_OK, LV_SYMBOL_CLOSE,
""};

//tab1 选项卡初始化
void tab1_init(lv_obj_t * parent)
{
    //1.设置页面的布局方式
    lv_page_set_scrl_layout(parent,LV_LAYOUT_PRETTY);

    //2.创建一个按钮
    lv_obj_t * btn = lv_btn_create(parent,NULL);
    lv_obj_t * btn_label = lv_label_create(btn,NULL);
    lv_label_set_text(btn_label,"Button");

    //3.创建一个矩阵按钮
    lv_obj_t * bnm = lv_bnm_create(parent,NULL);
    lv_obj_set_size(bnm,LV_HOR_RES_MAX/4, 2*LV_DPI/3);
```

```
lv_btm_set_map(btnm,(const char **)btnm_str);
lv_btm_set_btn_ctrl_all(btnm,LV_BTNM_CTRL_TGL_ENABLE);
lv_btm_set_one_toggle(btnm,true);

//4.创建一个开关
lv_sw_create(parent,NULL);

//5.创建一个进度条
lv_obj_t * bar = lv_bar_create(parent,NULL);
lv_bar_set_value(bar,70,false);

//6.创建一个滑块
lv_obj_t * slider = lv_slider_create(parent,NULL);
lv_slider_set_value(slider,50,false);

//7.创建一个下拉列表框
lv_obj_t * ddlist = lv_ddlist_create(parent,NULL);
//为右侧的箭头腾出点空间
lv_ddlist_set_fix_width(ddlist,lv_obj_get_width(ddlist)+LV_DPI/2);
lv_ddlist_set_draw_arrow(ddlist,true);
}

//tab2 选项卡初始化
void tab2_init(lv_obj_t * parent)
{
    //1.设置页面的布局方式
    lv_page_set_scrl_layout(parent,LV_LAYOUT_PRETTY);

    //2.创建一个列表
    lv_obj_t * list = lv_list_create(parent,NULL);
    lv_obj_set_size(list,LV_HOR_RES_MAX/4,LV_VER_RES_MAX/2);
    lv_list_add_btn(list,LV_SYMBOL_VIDEO,"Video");
    lv_list_add_btn(list,LV_SYMBOL_CALL,"Call");
    lv_list_add_btn(list,LV_SYMBOL_BELL,"Bell");
    lv_list_add_btn(list,LV_SYMBOL_FILE,"File");
    lv_list_add_btn(list,LV_SYMBOL_EDIT,"Edit");
    lv_list_add_btn(list,LV_SYMBOL_CUT,"Cut");
    lv_list_add_btn(list,LV_SYMBOL_COPY,"Copy");

    //3.创建一个滚轮
    lv_obj_t * roller = lv_roller_create(parent,NULL);

    lv_roller_set_options(roller,"Monday\nTuesday\nWednesday\nThursday\nFriday\nSaturday\nSunday",true);
}
```

```
lv_roller_set_selected(roller,1,false);
lv_roller_set_visible_row_count(roller,3);
}

//例程入口
void lv_theme_test_start()
{
    lv_obj_t * scr = lv_scr_act(); //获取当前活跃的屏幕对象

    //1. 使用 night 主题,当然了,你也可以换成其他的主题
    lv_theme_t * theme = lv_theme_night_init(210,NULL); //创建主题
    lv_theme_set_current(theme); //使用主题

    //2. 在屏幕上创建一些控件来查看主题的界面效果
    lv_obj_t * tabview = lv_tabview_create(scr,NULL);
    //2.1 创建 tab1 选项卡
    lv_obj_t * tab1 = lv_tabview_add_tab(tabview,"Tab 1");
    tab1_init(tab1);
    //2.2 创建 tab2 选项卡
    lv_obj_t * tab2 = lv_tabview_add_tab(tabview,"Tab 2");
    tab2_init(tab2);
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:



图 3.4.1 tab1 选项卡的初始界面效果(night 主题)

然后我们点击 tab2 选项卡按钮,可以看到 tab2 选项卡的界面如下所示:

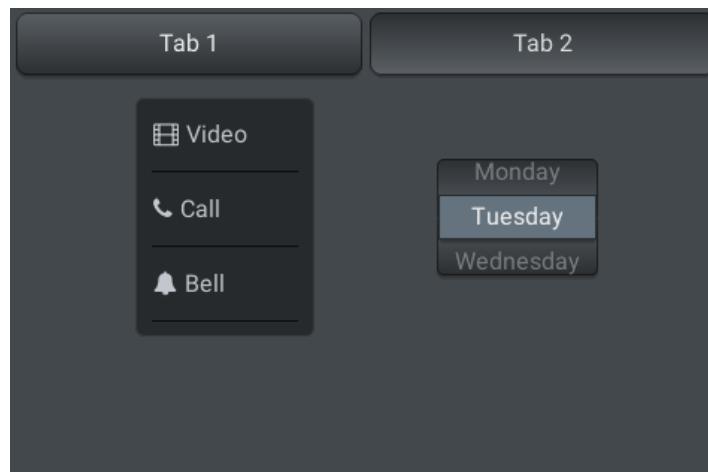


图 3.4.2 tab2 选项卡的初始界面效果(night 主题)

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



扫码关注正点原原子公众号