User Guide

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.

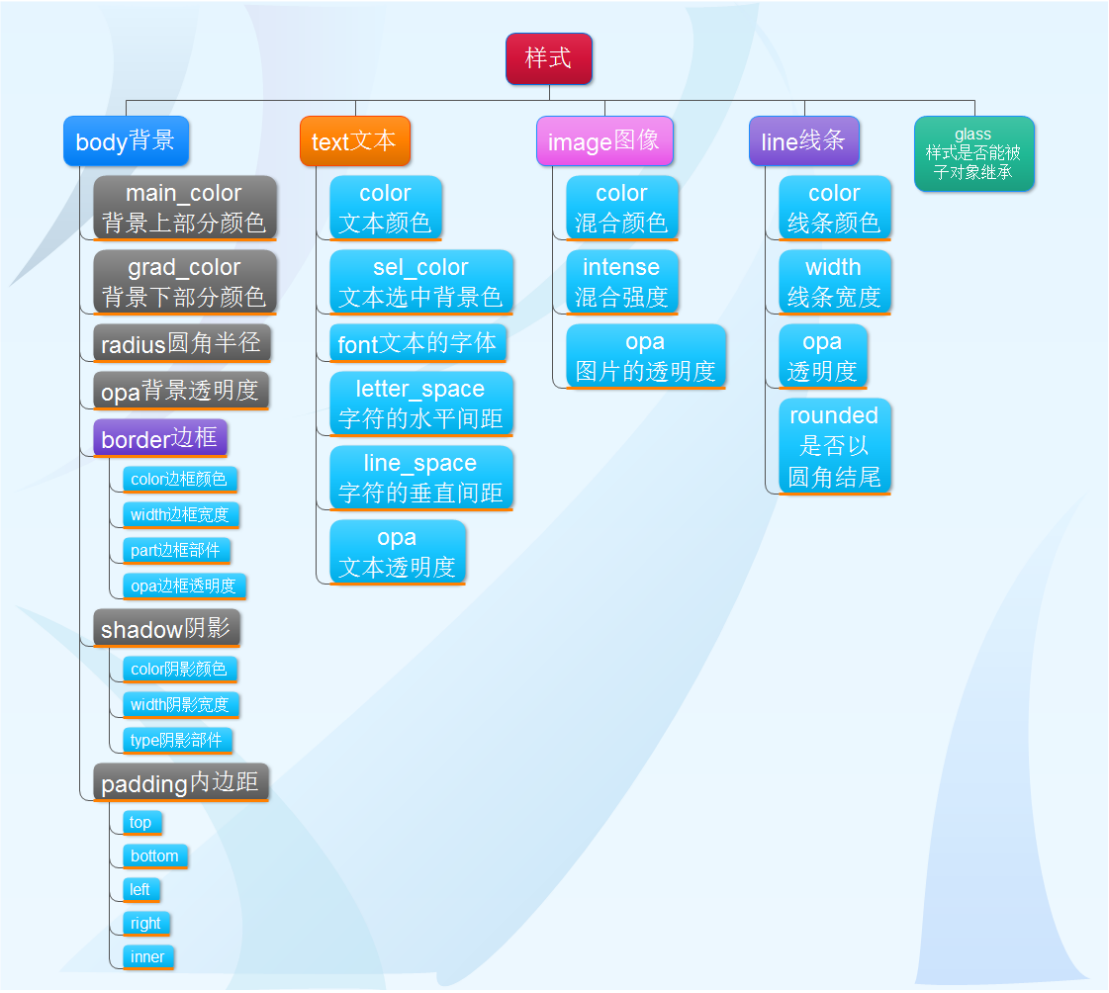

图 1.1 样式的结构

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
```

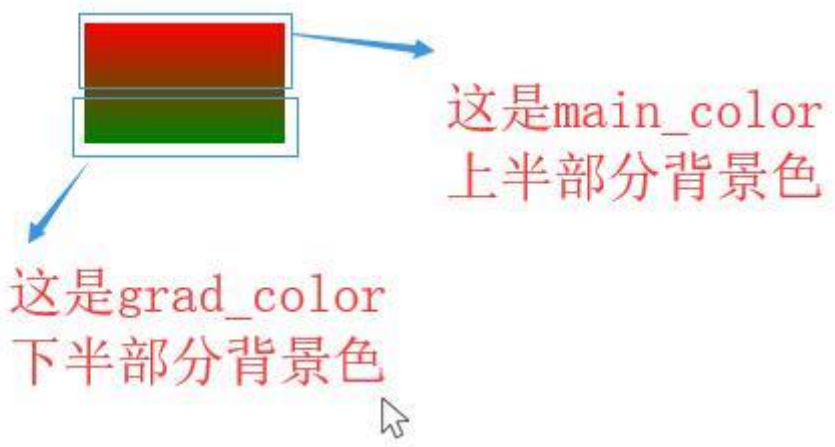

图 2.1.3.2.1.1 渐变色背景效果

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

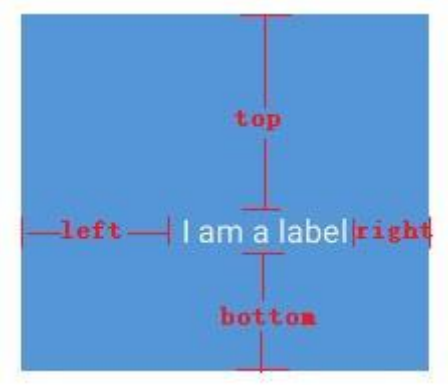

图 2.1.3.2.7.2 设置了 padding 时的效果

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);
```

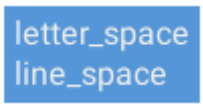

图 2.1.3.3.1 不设置 text 时的默认效果

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;
```

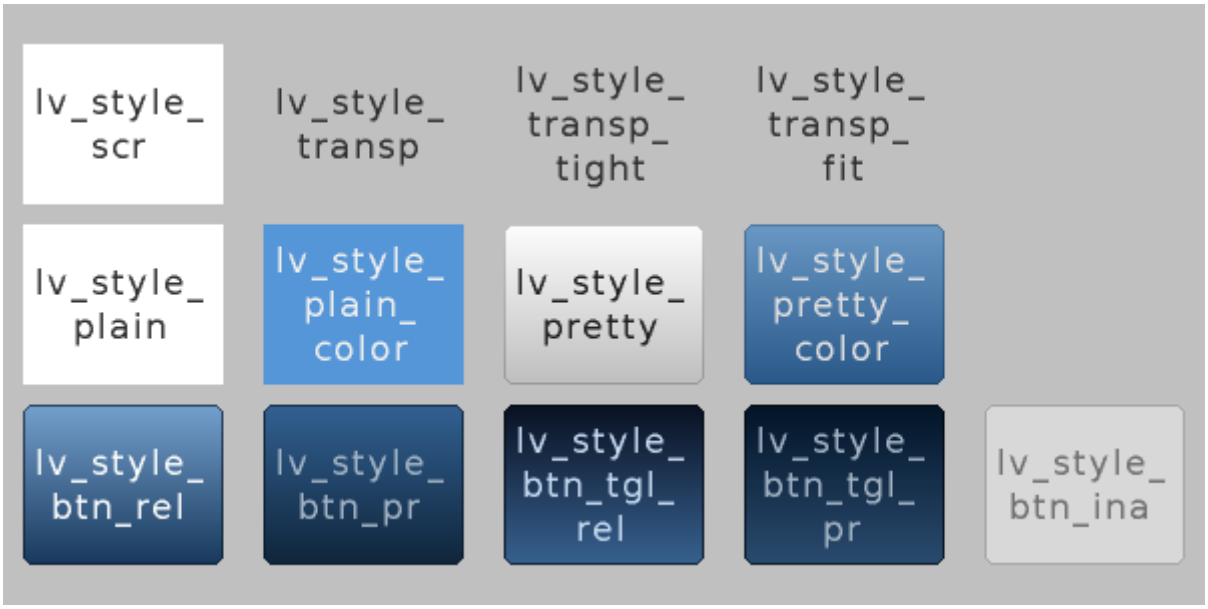

图 2.2.1.1 系统样式效果图

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属性的重复赋值

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 "usart.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)
        {
```

```
            //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_BO
TTOM_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