

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面这样:

```
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_DEFOCUSED,
    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:

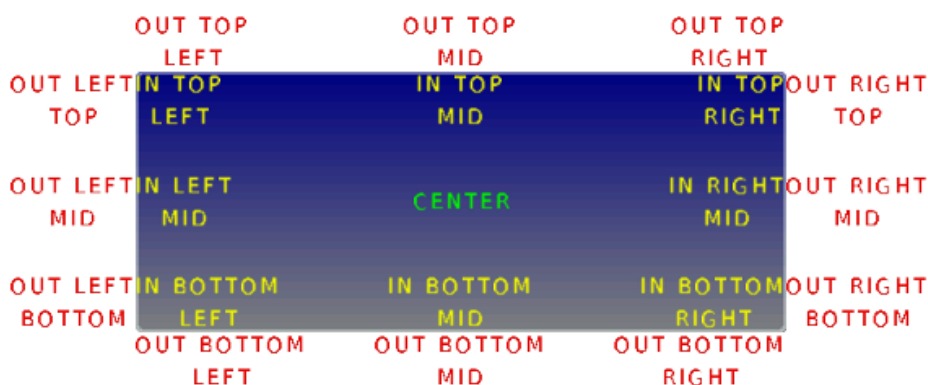


图 2.1.2.1 对齐示意图

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
}
```

```
#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

```

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应对齐之后的坐标,为了便于理解,align 的对齐方式请看下图:

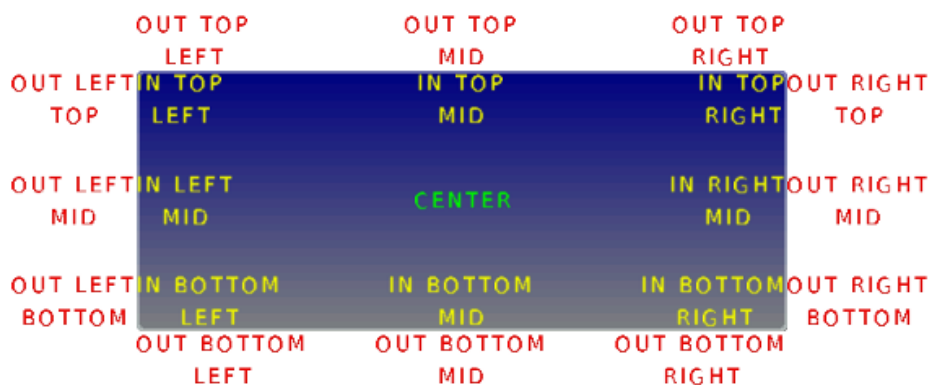




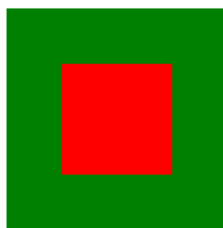
图 2.2.16.1 对齐方式

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:

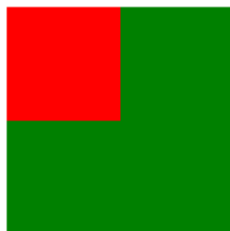
```
lv_obj_align(obj, base, align,0,0);
```

随着 align 取不同的值,得到下面不同的图例:

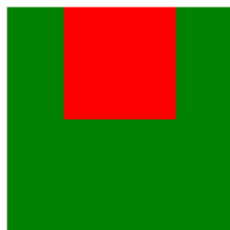
1) LV_ALIGN_CENTER



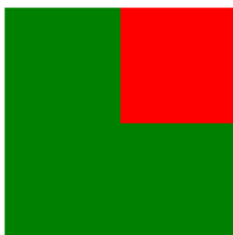
2) LV_ALIGN_IN_TOP_LEFT



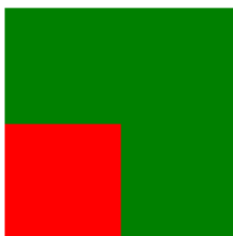
3) LV_ALIGN_IN_TOP_MID



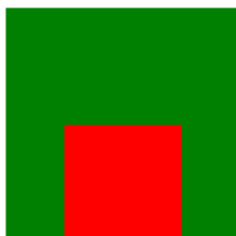
4) LV_ALIGN_IN_TOP_RIGHT



5) LV_ALIGN_IN_BOTTOM_LEFT



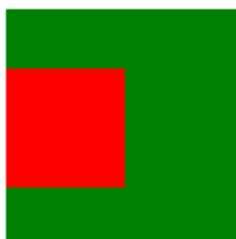
6) LV_ALIGN_IN_BOTTOM_MID



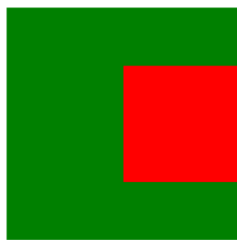
7) LV_ALIGN_IN_BOTTOM_RIGHT



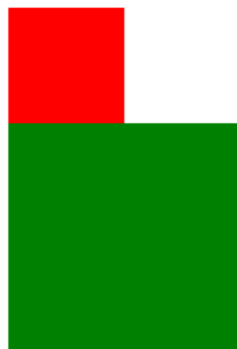
8) LV_ALIGN_IN_LEFT_MID



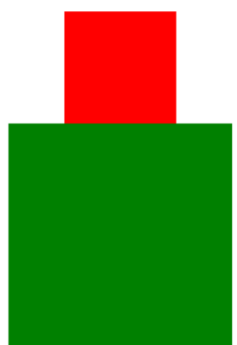
9) LV_ALIGN_IN_RIGHT_MID



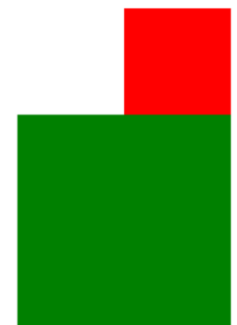
10) LV_ALIGN_OUT_TOP_LEFT



11) LV_ALIGN_OUT_TOP_MID



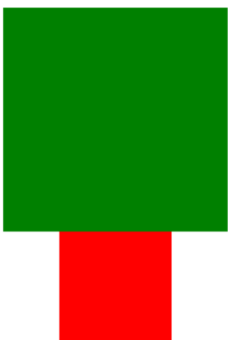
12) LV_ALIGN_OUT_TOP_RIGHT



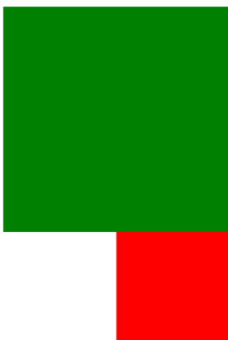
13) LV_ALIGN_OUT_BOTTOM_LEFT



14) LV_ALIGN_OUT_BOTTOM_MID



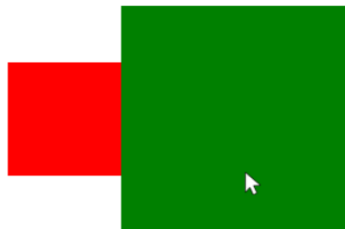
15) LV_ALIGN_OUT_BOTTOM_RIGHT



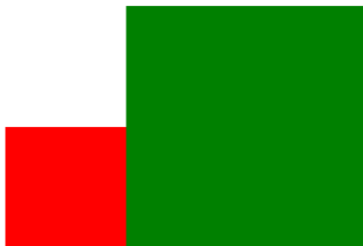
16) LV_ALIGN_OUT_LEFT_TOP



17) LV_ALIGN_OUT_LEFT_MID



18) LV_ALIGN_OUT_LEFT_BOTTOM



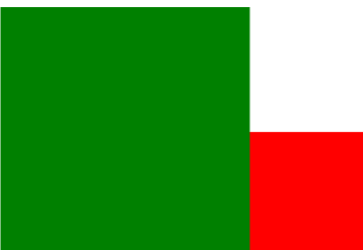
19) LV_ALIGN_OUT_RIGHT_TOP



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

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 对齐方式来进行说明,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 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

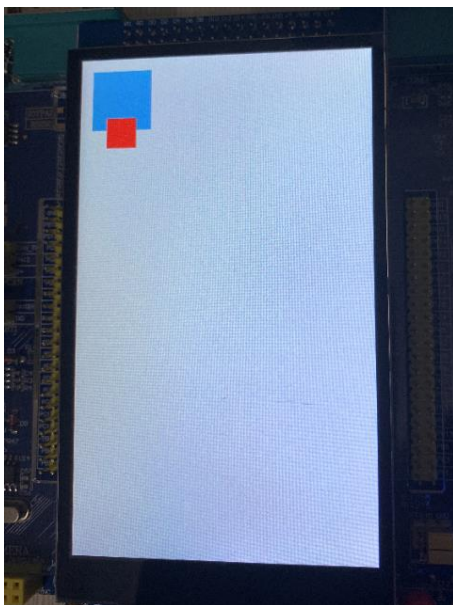
    lv_obj_set_opa_scale_enable(obj1,true);//先使能 obj1 的透明度功能
    lv_obj_set_opa_scale(obj1,100);//再设置 obj1 的透明度为 50

    //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");
    }
}
}
```

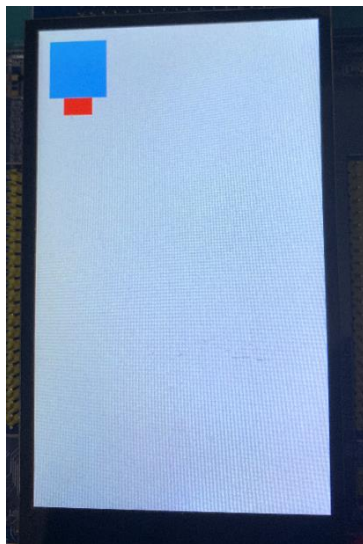
3.4 下载验证

1)开发板刚下载完代码后的状态



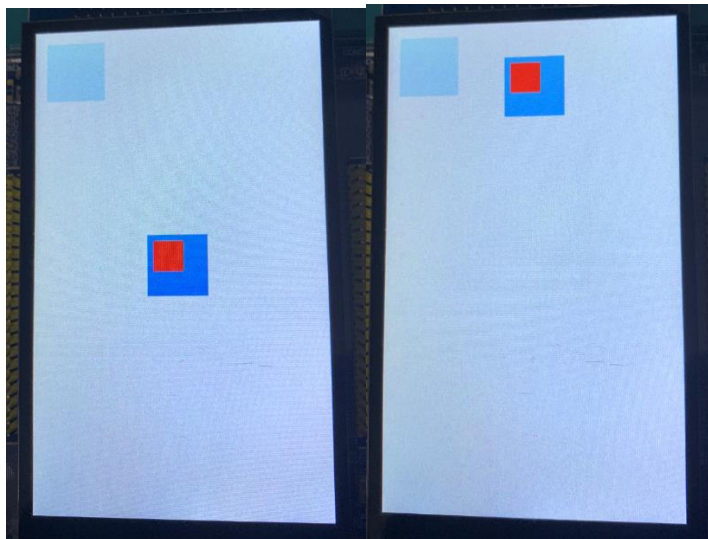
我们可以看到红色的 obj2 对象跟 obj1 对象是进行了 LV_ALIGN_OUT_BOTTOM_MID 对齐后,再接着 y 轴往上偏移 10 个像素,此时 obj2 是压在 obj1 上的

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

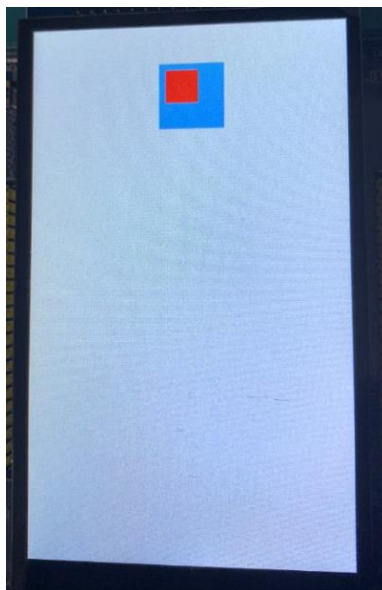
4)按下 KEY1 键后的状态



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