1

Lvgl 应用编程说明

Version:v1.0

Date: 2023.12

FREQCHIP
富芮坤

修订版本

版本	日期	更新内容
V1.0	2023.12.21	首版

目录

1. lv_obj_t 对象简介

LVGL 采用面向对象的编程思想，他的基本构造块(类)是对象(实例)，也就是我们所说的部件(Widgets)就是一个个部件，比如 button、label、image 等等。

obj 可以被看作是一个基类或父类，它提供了部件通用的属性和方法。具体的部件（例如按钮、标签等）被认为是子类，它们继承了 lv_obj_t 的属性和方法，并且可以进一步具体化、扩展或修改它们。这种继承关系使得子类可以保留基类的通用行为，同时根据具体的需求进行自定义。

例如基础对象 lv_obj_t 可能定义了一些通用的属性，如位置、大小、颜色等，并提供了一些通用的方法，如显示、隐藏、设置属性等。然后，具体的部件可以继承这些通用的属性和方法，框图如下图所示。

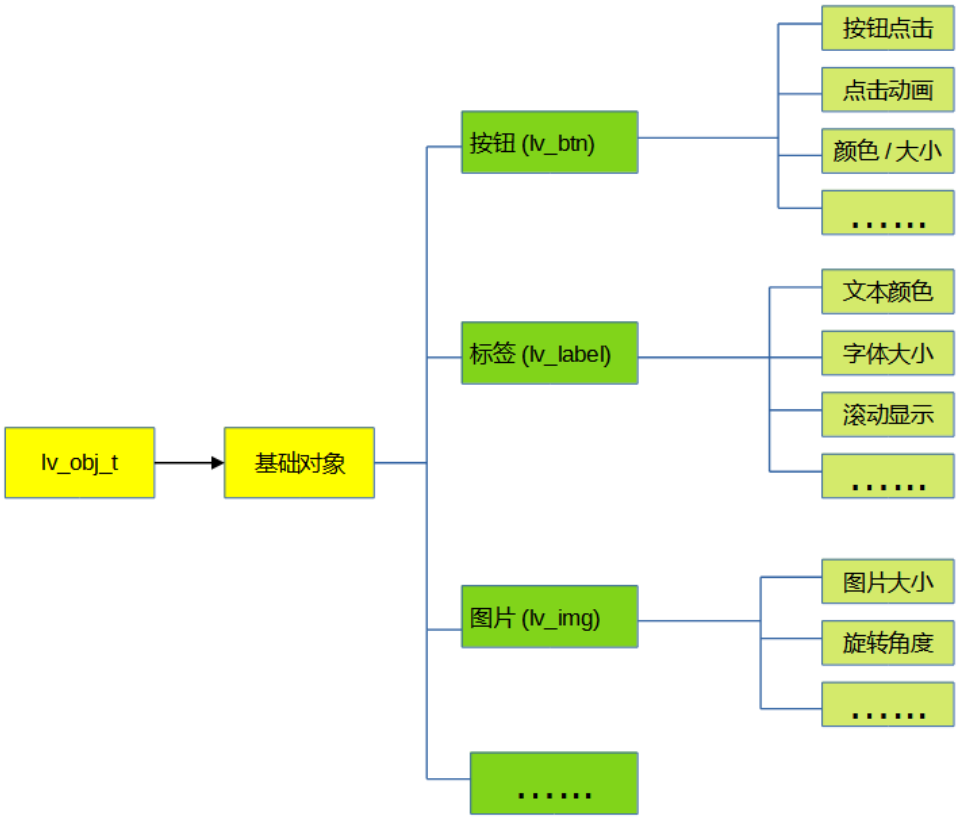
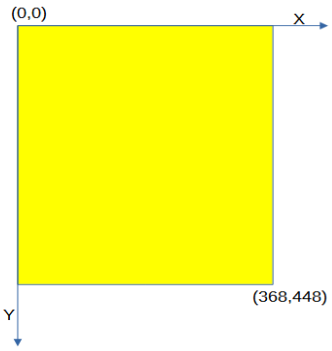


图 1-1 lv_obj_框图

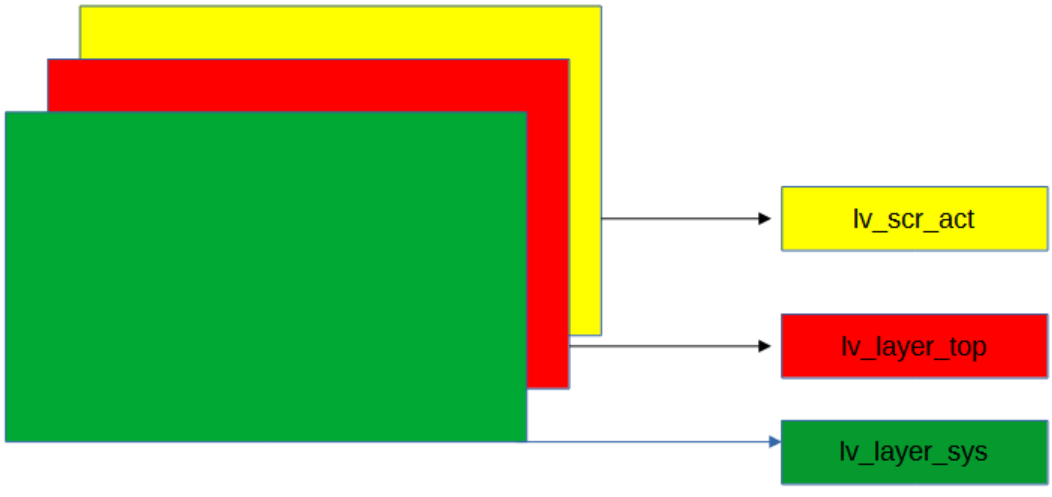
2. 屏幕的对象创建过程

```
lv_init
    _lv_ll_init(&LV_GC_ROOT(_lv_disp_ll),sizeof(lv_disp_t));// 初始化显示器链表
lv_disp_drv_register
   _lv_ll_ins_head(&LV_GC_ROOT(_lv_disp_ll));// 注册显示器到链表
   disp->act_scr   = lv_obj_create(NULL);// 在显示器上创建一个默认屏幕
   lv_obj_class_create_obj
   obj->coords.x1 = 0;
   obj->coords.y1 = 0;
   obj->coords.x2=lv_disp_get_hor_res(NULL)-1;// 设置屏幕的水平宽度
   obj->coords.y2 = lv_disp_get_ver_res(NULL)-1;//  设置屏幕的垂直高度
```



2.1. 图层简介

LVGL 中引入了图层（Layer）的概念，其中默认规则是最后创建的图层位于最上层。通常，我们在 lv_scr_act 层上创建各种控件（widgets），每个控件实际上相当于一个小小的图层，展示在屏幕上。除了 lv_scr_act 层外，还存在两个特殊的层，分别是 lv_layer_top 和 lv_layer_sys 这三个函数涉及到屏幕和图层管理，用于获取或操作不同的屏幕和图层。



2.2. lv_scr_act(void)活动屏幕

作用：该函数用于获取当前活动的屏幕。活动屏幕是用户当前正在交互的屏幕，

也就是当前显示在最顶部的屏幕。该函数通过返回 disp->act_scr 来获取活动屏幕。

2.3. lv_layer_top(void)顶层图层

作用:该函数用于获取顶层图层。LVGL 中的图层是用于管理屏幕上的不同元素(对象)的结构。

顶层图层是图层的一种，用于放置在最顶部的元素，即位于所有其他图层之上。通过返回 disp->top_layer,该函数获取了顶层图层的指针。

2.4. lv_layer_sys(void) 系统层

作用：该函数用于获取系统层。用于放置一些系统级别的元素，例如弹出框、通知等。通过返回 disp->sys_layer 该函数获取了系统层的指针。

3. LVGL 事件管理

事件管理涉及处理用户交互、输入、以及图形渲染和自定义事件。以下是说明如何在 LVGL 中进行事件管理。

3.1. 事件类型

输入设备相关事件

```
LV_EVENT_ALL = 0,
/** Input device events*/
LV_EVENT_PRESSED,             /**< The object has been pressed*/
LV_EVENT_PRESSING,            /**< The object is being pressed (called continuously while pressing)*/
LV_EVENT_PRESS_LOST,          /**< The object is still being pressed but slid cursor/finger off of the object */
LV_EVENT_SHORT_CLICKED,       /**< The object was pressed for a short period of time, then released it. Not called if scrol
LV_EVENT_LONG_PRESSED,        /**< Object has been pressed for at least `long_press_time`.  Not called if scrolled.*/
LV_EVENT_LONG_PRESSED_REPEAT, /**< Called after `long_press_time` in every `long_press_repeat_time` ms.  Not called if scro
LV_EVENT_CLICKED,             /**< Called on release if not scrolled (regardless to long press)*/
LV_EVENT_RELEASED,            /**< Called in every cases when the object has been released*/
LV_EVENT_SCROLL_BEGIN,        /**< Scrolling begins. The event parameter is a pointer to the animation of the scroll. Can b
LV_EVENT_SCROLL_END,          /**< Scrolling ends*/
LV_EVENT_SCROLL,              /**< Scrolling*/
LV_EVENT_GESTURE,             /**< A gesture is detected. Get the gesture with `lv_indev_get_gesture_dir(lv_indev_get_act()
LV_EVENT_KEY,                 /**< A key is sent to the object. Get the key with `lv_indev_get_key(lv_indev_get_act());`*/
LV_EVENT_FOCUSED,             /**< The object is focused*/
LV_EVENT_DEFOCUSED,           /**< The object is defocused*/
LV_EVENT_LEAVE,               /**< The object is defocused but still selected*/
LV_EVENT_HIT_TEST,            /**< Perform advanced hit-testing*/
```

图 3-1 输入设备相关事件宏定义示例

渲染相关事件

```
LV_EVENT_COVER_CHECK,         /**< Check if the object fully covers an area. The event parameter is `lv_cover_
LV_EVENT_REFR_EXT_DRAW_SIZE, /**< Get the required extra draw area around the object (e.g. for shadow). The e
LV_EVENT_DRAW_MAIN_BEGIN,     /**< Starting the main drawing phase*/
LV_EVENT_DRAW_MAIN,           /**< Perform the main drawing*/
LV_EVENT_DRAW_MAIN_END,       /**< Finishing the main drawing phase*/
LV_EVENT_DRAW_POST_BEGIN,     /**< Starting the post draw phase (when all children are drawn)*/
LV_EVENT_DRAW_POST,           /**< Perform the post draw phase (when all children are drawn)*/
LV_EVENT_DRAW_POST_END,       /**< Finishing the post draw phase (when all children are drawn)*/
LV_EVENT_DRAW_PART_BEGIN,     /**< Starting to draw a part. The event parameter is `lv_obj_draw_dsc_t *`. */
LV_EVENT_DRAW_PART_END,       /**< Finishing to draw a part. The event parameter is `lv_obj_draw_dsc_t *`. */
```

图 3-2 渲染相关事件宏定义示例

其他系统事件

```
/** Other events*/
LV_EVENT_DELETE,              /**< Object is being deleted*/
LV_EVENT_CHILD_CHANGED,       /**< Child was removed, added, or its size, position were changed */
LV_EVENT_CHILD_CREATED,       /**< Child was created, always bubbles up to all parents*/
LV_EVENT_CHILD_DELETED,       /**< Child was deleted, always bubbles up to all parents*/
LV_EVENT_SCREEN_UNLOAD_START, /**< A screen unload started, fired immediately when scr_load is called*/
LV_EVENT_SCREEN_LOAD_START,   /**< A screen load started, fired when the screen change delay is expired
LV_EVENT_SCREEN_LOADED,       /**< A screen was loaded*/
LV_EVENT_SCREEN_UNLOADED,     /**< A screen was unloaded*/
LV_EVENT_SIZE_CHANGED,        /**< Object coordinates/size have changed*/
LV_EVENT_STYLE_CHANGED,       /**< Object's style has changed*/
LV_EVENT_LAYOUT_CHANGED,      /**< The children position has changed due to a layout recalculation*/
LV_EVENT_GET_SELF_SIZE,       /**< Get the internal size of a widget*/
```

图 3-3 其他系统事件宏定义示例

用户自定义事件

```
LV_EVENT_CALL_IN,
LV_EVENT_CALL_REJECT,
LV_EVENT_CALL_OK,
LV_EVENT_SCROLL_PREDICT,
LV_EVENT_ANCS_MSG_IN,
LV_EVENT_MSUIC_CONTEXT_IN,
LV_EVENT_CALL_HANDUP,
LV_EVENT_SCREENSAVER_IN,
LV_EVENT_FLEXIBLE_IN,
```

图 3-4 用户自定义事件宏定义示例

事件是整个 LVGL 是其中不可或缺的一部分，它可以将事件有效的通知到组件，应用层可以通过注册事件回调，来监测并响应事件。LVGL 内部的绘制流程，也会通过 Event 机制，通知到各个组件，进行绘制等操作。

事件主要分为以下几类，如下表所示：

表格 3-1 事件分类

事件名称	事件描述
输入事件	点击，长按，滚动，焦点等
渲染事件	绘制开始，绘制中，完成绘制，Post 绘制等
特殊事件	刷新事件等
其他事件	对象被删除，子对象被删除等
自定义事件	来电，消息等

3.2. 事件处理

在对象初始化时添加相关的事件通过函数 lv_obj_add_event_cb 添加，事件回调函数中，添加对相关事件的处理，示例代码如下所示。

```
static void my_event_cb(lv_event_t * e)
{
    lv_event_code_t event = e->code;
    void * user_data = lv_event_get_user_data(e);
    lv_obj_t * obj = lv_event_get_target(e);
    if(!lv_obj_is_valid(obj))
        return;
    switch(event)
    {
        case LV_EVENT_CLICKED:
        printf("LV_EVENT_CLICKED\r\n");
        break;
        case LV_EVENT_LONG_PRESSED:
        printf("LV_EVENT_LONG_PRESSED\r\n");
        break;
        case LV_EVENT_PRESSING:
        printf("LV_EVENT_PRESSING\r\n");
        break;
    }
}
```

图 3-5 事件处理示例代码

通过事件结构体定义，能够知道事件回调中可以获取到当前的对象，事件代码，用户自定义数据等，这些在事件处理的时候需要。

4. lv_obj 基础对象属性

LVGL 是面向对象的, lv_obj_t 是 LVGL 中所有对象的基类、是整个框架的核心概念、对象的一般属性包括位置、大小、样式，父对象和事件等。

创建和设置 lv_obj_t 对象的简单示例请参考如下章节。

4.1. 创建对象

表格 4-1 创建对象

用法	使用 lv_obj_create 函数创建 lv_obj_t 对象
示例	lv_obj_t *my_obj = lv_obj_create(lv_scr_act());
说明	创建一个对象，并将其附加到当前激活的屏幕（lv_scr_act()）。lv_scr_act()是获取当前活动的屏幕对象。屏幕对象也是用户界面的根对象，用于承载整个用户界面。

4.2. 位置

表格 4-2 位置

用法	使用 lv_obj_set_pos 设置对象的位置
示例	lv_obj_set_pos(my_obj, 20, 30);
说明	这将把对象的左上角定位在坐标（20，30）处。

4.3. 大小

表格 4-3 大小

用法	使用 lv_obj_set_size 设置对象的大小
示例	lv_obj_set_size(my_obj, 100, 50);
说明	这将设置对象的宽度为 100 像素，高度为 50 像素。

4.4. 父类与子类（Parents and children）

父类和子类之间的关系是通过对象树的形式建立的。每个 lv_obj_t 对象都可以包含其他对象，形成一个层次结构的树状组织。在这个树中，对象之间的关系是父子关系。每个对象都可以有一个父对象和多个子对象，可以轻松地组织和操纵对象。通过这种关系，可以在整个对象树中导航，找到和操作特定的对象，对于构建复杂的用户界面交互时非常有用。通过子类设置父类信息详见表格 4-4。

表格 4-4 父类与子类

用法	使用 lv_obj_create 函数创建父对象和子对象，通过子类设置父类信息

示例	lv_obj_t *parent_obj = lv_obj_create(lv_scr_act());
	lv_obj_t *child_obj1 = lv_obj_create(parent_obj);
	lv_obj_t *child_obj2 = lv_obj_create(parent_obj);

说明	在这个例子中，child_obj1, child_obj2 都是 parent_obj 的子对象，比如我们应用中想通过子对象去操作父对象的一些属性可以调用接口 lv_obj_get_parent(child_obj1)，就可以获取到当前父对象指针。当我们想通过父类对象去获取子类对象时也可以调用接口 lv_obj_get_child(parent_obj,0)就能获取到当前第一个子类对象指针。

4.5. 事件（Event）

表格 4-5 事件

用法	通过给对象添加点击事件，可以使对象在被点击时触发对应的回调事件

示例	lv_obj_add_event_cb(my_obj, my_obj_click_event_handler,LV_EVENT_CLICKED,NULL);
	// 定义点击事件的回调函数
	static void my_obj_click_event_handler(lv_event_t *event) {
	  if (event->code == LV_EVENT_CLICKED) {
	  printf("LV_EVENT_CLICKED\r\n");
	  // 在这里执行点击事件触发时的操作
	  // 例如，可以在这里切换页面、改变对象属性等
	  }
	}

说明	也支持其他事件的回调，在添加事件时设置即可。

4.6. 其他属性和方法

表格 4-6 其他属性和方法

示例	说明
lv_obj_get_x(obj)	获取对象的 X 坐标。
lv_obj_get_y(obj)	获取对象的 Y 坐标。
lv_obj_get_width(obj)	获取对象的宽度。
lv_obj_get_height(obj)	获取对象的高度。
lv_obj_set_width(obj)	设置对象的宽度。
lv_obj_set_height(obj)	设置对象的高度。

4.7. 相对布局

相对布局是一种在 LVGL 中常用的布局方式，允许定义对象相对于其父对象或其他对象的位置。相对布局的主要优势在于，当父对象的大小或位置发生变化时，子对象的位置可以相对应地调整，使得界面的布局更加灵活和适应性更强。在 LVGL 中，可以使用以下函数来设置对象的位置，对齐类型有如下图：

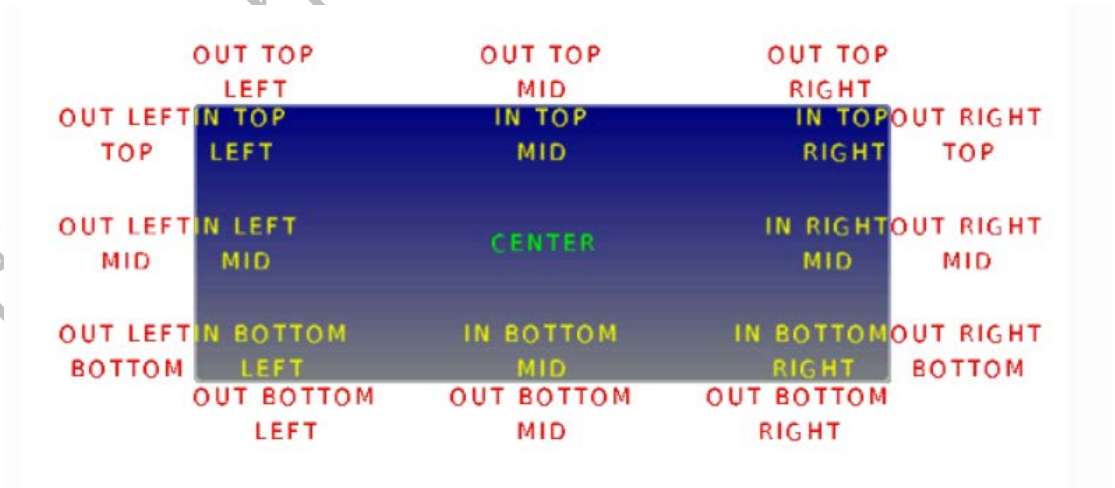


图 4-1 对齐类型图

表格 4-7 相对布局

示例	说明
lv_obj_set_pos(obj, x, y)	设置对象的绝对位置，即对象左上角相对于父对象或屏幕的坐标。
lv_obj_align((obj, align, x_ofs, y_ofs)	使用相对布局 相对于其父对象对齐
lv_obj_align_to(obj, base,align, x_ofs, y_ofs)	使用相对布局将对象相对于基础对象（base）对齐，并可通过 x_ofs 和 y_ofs 进行微调。

lv_obj_t *parent_obj =
lv_obj_create(lv_scr_act());
lv_obj_set_size(parent_obj, 200, 150);
	创建两个子对象（按钮）并设置相对布局

lv_obj_t *btn1 = lv_btn_create(parent_obj); lv_obj_set_pos(btn1, 10, 20);	设置 btn1 的绝对位置
lv_obj_t *btn2 = lv_btn_create(parent_obj); lv_obj_align_to(btn2, btn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);	设置 btn2 相对于 btn1 的位置 将 btn2 相对于 btn1 的底部中间对齐，y 偏移 20 个像素
lv_obj_t *btn3= lv_btn_create(parent_obj); lv_obj_align(btn3, LV_ALIGN_CENTER, 0, 10);	将 btn3 相对于其父对象居中对齐，垂直方向上偏移 10 个像素。

4.8. 样式设置

基本样式设置：

样式是通过 lv_style_t 结构体来定义的。这个结构体包含了一系列字段，用于描述对象不同部分的样式，包含各种属性，例如颜色、字体、边框等。以下是一些基本常用的样式设置。

表格 4-8 样式设置

示例	说明
lv_obj_set_style_bg_color	设置背景颜色
lv_obj_set_style_border_width	设置对象边缘宽度
lv_obj_set_style_border_color	设置对象边缘颜色

lv_obj_set_style_text_color	设置文本颜色
lv_obj_set_style_text_font	设置字体
lv_obj_set_style_radius	设置圆角半径
lv_style_set_bg_opa	设置背景透明度

4.9. 标志（Flag）

这些标志常常与对象的属性状态相关联，以便在运行时决定对象的行为, 属性可以启用/禁用，通过接口 lv_obj_add_flag，lv_obj_clear_flag 来设置。

表格 4-9 标志

示例	说明
lv_obj_add_flag(obj, LV_OBJ_FLAG_CLICKABLE);	使对象可以通过输入设备点击
lv_obj_clear_flag(obj, LV_OBJ_FLAG_CLICKABLE);	使对象不能通过输入设备点击
lv_obj_add_flag(obj, LV_OBJ_FLAG_HIDDEN);	使对象隐藏，就好像它根本不存在一样。

4.10. 组

Group 是一种对象组织结构，用于管理一组相关联的对象。这些对象可以是按钮、标签、图像等 LVGL 元素。通过将对象组织成组，可以在它们之间进行统一的管理和操作。

4.10.1. 创建 Group:

表格 4-10 创建/移除 Group

示例	说明
lv_group_t *my_group = lv_group_create();	创建一个新的 Group。
lv_group_add_obj(my_group, my_button);	将对象添加到 Group，将一个对象添加到 Group 中。

4.10.2. 从 Group 中移除对象

示例	说明
lv_group_remove_obj(my_group, my_button);	从 Group 中移除对象，将对象从 Group 中移除。

4.10.3. Group 的事件处理

Group 通过管理对象的事件来协调对象之间的交互。

表格 4-11 Group 事件处理

示例	说明
lv_group_set_default(my_group, my_button);	设置 Group 的默认焦点对象 当 Group 接收到焦点时，它将焦点设置为默认对象。
lv_obj_t *focused_obj = lv_group_get_focused(my_group);	获取当前焦点对象 返回当前在 Group 中拥有焦点的对象。
lv_group_set_focus_cb(my_group, my_focus_cb);	用户可以设置 Group 的事件处理回调，以便在事件发生时执行特定的操作。 此处是设置焦点变化的回调函数

4.10.4. Group 的导航（向前/向后移动焦点）

表格 4-12 Group 导航

示例	说明
lv_group_focus_next(my_group);	移动焦点到下一个对象
lv_group_focus_prev(my_group);	移动焦点到上一个对象
lv_group_set_wrap(my_group, true);	设置 Group 的导航链: 允许循环导航，即从最后一个对象导航到第一个对象

4.10.5. Group 的状态（启用/禁用 Group）

表格 4-13 启用/禁用 Group

示例	说明
lv_group_set_editing(my_group, true);	启用 Group 的编辑模式
lv_obj_t *edited_obj = lv_group_get_editing(my_group);	在编辑模式下获取当前正在编辑的对象
lv_group_set_editing(obj_to_edit);	将对象设置为正在编辑的对象

5. img 对象

img 对象是 LVGL 中用于显示图像的对象，可以通过设置图像源、位置、大小等参数来控制图像的显示。通过设置透明度、旋转、缩放等属性，可以实现更多的效果。图像对象的创建和设置都可以根据具体的应用场景进行调整。

5.1. 创建 img 对象

表格 5-1 创建 img 对象

示例	说明
lv_obj_t *img_obj = lv_img_create(parent);	在指定的 parent 对象下创建一个 img 对象。

5.2. 设置图像源

表格 5-2 设置图像源

示例	说明
lv_img_set_src(img_obj, &image_data);	设置图像数据 image_data 是图像的数据，可以是存储在内存中的数组或者是外部 flash 等。
lv_obj_set_pos(img_obj, x, y);	设置图像位置
lv_obj_set_size(img_obj, width, height);	设置图像大小 默认不用设置这个，通过图像源数据中设置

5.3. 设置图像数据

表格 5-3 设置图像数据

示例	说明
lv_img_set_src(img_obj, &image_data);	这里的 image_data 可以是一个数组，表示图像的像素数据。

lv_img_set_src(img_obj, "S:/path/image.png");	设置图像路径，如果图像存储在外部文件系统中，可以通过文件系统路径来设置图像源。

5.4. 设置图像的透明度

表格 5-4 设置图像透明度

示例	说明
lv_obj_set_style_img_opa(img_obj, LV_OPA_50, LV_STATE_PRESSED);	这将设置图像的透明度。LV_OPA_50 表示 50%的不透明度。

5.5. 设置图像的旋转

表格 5-5 设置图像旋转

示例	说明
lv_img_set_angle(img_obj, 900);	这将使图像旋转 90 度。可以通过设置不同的角度来实现旋转。旋转角度在刻度上为 0.1 度。

5.6. 设置图像的缩放

表格 5-6 设置图像的缩放

示例	说明
lv_img_set_zoom(img_obj, LV_ZOOM_IMG_NONE);	设置 LV_ZOOM_IMG_NONE 表示不缩放，也可以设置其他的缩放值。

5.7. 设置图像对象的旋转中心点

表格 5-7 设置图像对象的旋转中心点

示例	说明
lv_img_set_pivot(img_obj, 75, 75);	对图像进行旋转时，旋转中心是一个重要的参数，决定了图像围绕哪个点进行旋转，调用 lv_img_set_pivot 函数，用户可以在图像对象上设置一个新的旋转中心，从而改变图像围绕哪个点进

	行旋转。这个函数通常用于在旋转图像之前设置旋转中心的位置，这个函数在制作指针表盘中很有用。lv_img_set_pivot(img_obj, 75, 75);设置后，当旋转这个图片时，就以图像的中心点 (75, 75) 为旋转中心。这样当旋转图像时，它会以中心点为基准进行旋转，而不是默认的左上角。

5.8. 设置图像 img 着色效果

表格 5-8 设置图像着色效果

示例	说明
lv_obj_set_style_img_recolor(img_obj, color, 0);	这个函数的作用是改变图像的颜色，使其具有一种着色效果。

5.9. 图像 img 事件

表格 5-9 图像 img 事件

示例	说明
lv_obj_add_flag(img_obj, LV_OBJ_FLAG_CLICKABLE)	img 图片默认不支持点击，所以直接添加点击事件是不会响应，通过 lv_obj_add_flag(img_obj, LV_OBJ_FLAG_CLICKABLE);添加后才能够触发点击事件。

6. 文本对象（label）

用于显示文本内容的基本对象，文本对象可用于显示单行或多行文本，支持对文本进行样式设置、对齐方式调整等操作。

6.1. 创建文本对象

表格 6-1 创建文本对象

示例	说明
lv_obj_t *label = lv_label_create(parent);	在指定的 parent 对象下创建一个文本对象。

6.2. 设置文本内容

表格 6-2 设置文本内容

示例	说明
lv_label_set_text(label, "Hello, World!");	设置文本对象要显示的文本内容。
lv_label_set_text_fmt(label,"%d%%",100);	支持带格式化参数的显示函数。

6.3. 设置文本字体

表格 6-3 设置文本字体

示例	说明
lv_obj_set_style_text_font(label,LV_FONT_OSD_MEDIUM_32PX,0);	设置文本字体。

6.4. 设置文本的样式

表格 6-4 设置文本样式

示例	说明
lv_obj_add_style(label, &my_style);	可以通过样式系统设置文本的外观，包括颜色、字体、大小等。

6.5. 多行文本

表格 6-5 多行文本

示例	说明
lv_label_set_long_mode(label, LV_LABEL_LONG_WRAP);	设置为自动换行模式
lv_obj_set_width(label, 200);	设置文本对象的宽度，以触发自动换行 通过设置自动换行模式和指定文本对象的宽度，可以实现多行文本的显示。

6.6. 文本对齐方式

表格 6-6 文本对齐方式

示例	说明
lv_obj_set_style_text_align(label, LV_ALIGN_CENTER, 0);	设置文本居中对齐 通过设置对齐方式，可以调整文本在文本对象内的位置。

6.7. 滚动文本

通过设置宽度、滚动模式和文本不透明度，可以实现文本的滚动显示效果。

表格 6-7 滚动文本

示例	说明
lv_obj_set_width(label, 100);	设置文本对象的宽度
lv_label_set_long_mode(label, LV_LABEL_LONG_SCROLL);	设置为循环滚动模式
lv_obj_set_style_text_opa(label, LV_OPA_COVER, 0);	设置文本不透明度

6.8. 实时更新文本内容

表格 6-8 实时更新文本内容

示例	说明
lv_label_set_text(label, "Updated Text");	可以通过 lv_label_set_text 函数在运行时更新文本内容。

7. LVGL 定时器组件

定时器（timer）是一种机制，用于在嵌入式图形界面中进行时间管理。通过定时器，可以在特定的时间间隔内执行回调函数，实现一些周期性的任务或者延时操作。

以下是关于 LVGL 定时器的详细使用。

7.1. 创建/删除定时器

表格 7-1 创建/删除定时器

示例	说明
lv_timer_t *my_timer = lv_timer_create(timer_callback, 1000, NULL);	这个例子中，timer_callback 是定时器到期时要执行的回调函数，1000 表示定时器的周期为 1 秒，NULL 是传递给回调函数的参数。
lv_timer_del(my_timer);	删除定时器，通过 lv_timer_del 函数可以删除一个定时器。

7.2. 定时器的启动/停止

表格 7-2 定时器的启动/停止

示例	说明
lv_timer_start(my_timer);	启动定时器
lv_timer_stop(my_timer);	使用 lv_timer_stop 函数可以停止定时器。停止后，定时器将不再触发回调函数。

7.3. 定时器的回调函数

定时器的回调函数是在定时器到期时执行的函数。可以通过下面的方式定义一个简单的回调函数：

表格 7-3 定时器的回调函数

示例	说明
void timer_callback(lv_timer_t *timer){}	在函数体里执行定时器到期时要进行的操作

7.4. 定时器的参数传递

可以通过 user_data 参数在创建定时器时传递额外的参数给回调函数，这个当我们要修改对象的某些参数时可以将对象通过这个参数传递过去。

表格 7-4 定时器的参数传递

示例	说明
lv_timer_t *my_timer = lv_timer_create(timer_callback, 1000, (void *)user_data);	在回调函数中可以通过 timer->user_data 获取传递的参数。

7.5. 动态调整定时器的周期

可以使用 lv_timer_set_period 函数动态地调整定时器的周期。

表格 7-5 动态调整定时器的周期

示例	说明
lv_timer_set_period(my_timer, 2000);	将定时器的周期调整为 2 秒。

7.6. 实时更新定时器的回调函数

表格 7-6 实时更新定时器的回调函数

示例	说明
lv_timer_set_cb(my_timer, updated_callback);	可以通过 lv_timer_set_cb 函数在运行时更新定时器的回调函数。

8. LVGL 应用界面编程规范

通过上面章节的介绍，对 lvgl 有了初步了解，现在介绍写一个计时器界面来讲解代码结构和规范。

8.1. 功能界面函数定义

首先我们要将每个界面都单独创建一个 c 文件和一个 func 函数来管理，每个界面的应用逻辑也都在当前的 c 文件中实现。功能界面函数定义如下：

表格 8-1 定时器函数说明

函数	void fr_app_timer_func(lv_obj_t *parent, lv_point_t *top)
说明	定义两个形参传递参数进来，一个是当前界面的父对象指针，一个是坐标。 函数里面首先要对父对象有效性进行检查，防止出现野指针问题。

```
    }
}

void fr_app_timer_func(lv_obj_t *parent, lv_point_t *top)
{
    if(!lv_obj_is_valid(parent))
        return;
    UI_PARENT_INIT(parent);
    lv_obj_t *label = lv_label_create(parent);
    lv_obj_set_style_text_color(label, lv_color_hex(0xffffff),0);
    lv_label_set_text_fmt(label,"%02d:%02d",0,0);
    lv_obj_align(label,LV_ALIGN_CENTER,0,-80);
    lv_obj_set_style_text_font(label,LV_FONT_BIG_NUMBER_62PX,0);
    lv_obj_t *start_btn = lv_btn_create(parent);
    lv_obj_set_style_pad_all(start_btn, 0, 0);
    lv_obj_set_style_radius(start_btn, 30, 0);
    lv_obj_set_size(start_btn,120,90);
    lv_obj_set_style_bg_color(start_btn,lv_color_hex(0xFD9405),0);
    lv_obj_align(start_btn,LV_ALIGN_LEFT_MID,30,30);
    lv_obj_t *stop_btn = lv_btn_create(parent);
    lv_obj_set_style_pad_all(stop_btn, 0, 0);
    lv_obj_set_style_radius(stop_btn, 30, 0);
    lv_obj_set_size(stop_btn,120,90);
    lv_obj_set_style_bg_color(stop_btn,lv_color_hex(0xFD9405),0);
    lv_obj_align(stop_btn,LV_ALIGN_RIGHT_MID,-30,30);
    lv_obj_t *start_label = lv_label_create(start_btn);
    lv_obj_set_style_text_color(start_label, lv_color_hex(0xffffff),0);
    lv_label_set_text_fmt(start_label,"开始");
    lv_obj_align(start_label,LV_ALIGN_CENTER,0,0);
    lv_obj_t *stop_label = lv_label_create(stop_btn);
    lv_obj_set_style_text_color(stop_label, lv_color_hex(0xffffff),0);
    lv_label_set_text_fmt(stop_label,"停止");
    lv_obj_align(stop_label,LV_ALIGN_CENTER,0,0);
    cnt_timer = lv_timer_create(timer_callback, 1000, (void*)label);
    lv_timer_pause(cnt_timer);
    lv_obj_add_event_cb(start_btn,click_event_cb, LV_EVENT_CLICKED, NULL);
    lv_obj_add_event_cb(stop_btn, click_event_cb, LV_EVENT_CLICKED, NULL);
    lv_obj_add_event_cb(label,label_delete_cb, LV_EVENT_DELETE, NULL);
}
```

图 8-1 定时器函数示例

以上代码对 label 和按钮进行初始化，同时添加了按钮的点击回调事件。通过创建一个定时器来实现周期性计时，开始计时后如要对时间进行显示，则需要实时修改 label 的内容。创

建定时器可以通过 user_data 传递参数，这样在事件回调中就可以使用我们的 label 对象,当需要改变当前界面多个对象时，可以传递其 parent 对象过去。

运行效果如下图所示。

图 8-2 运行效果图

8.2. 页面逻辑实现

当前界面逻辑，在两个按钮的点击的事件回调里去处理开始/停止定时器，定时器回调里去更新时间，这里需要特别注意的是，在定时器回调中操作 obj 对象时，一定要先进行有效性检查，如果当前对象被其他地方删除了就不能对其进行操作否则会出现异常。可以通过

lv_obj_is_valid(obj)进行检查。还有当界面中有用到定时器时也需要注意，当前界面被删除时要将定时器也删除，不然会出现野指针问题。

这个实例中定时器用于更新 label 的内容，所以给 label 添加了删除事件，当这个界面删除时，label 子对象删除时会触发相应的回调函数，我们在回调函数中将定时器进行删除。总之，当前界面有用到定时器组件的，都需要添加一个子对象的删除事件，来触发删除当前这个定时器，源码如下图所示。

```
static lv_timer_t *cnt_timer=NULL;
static uint32_t cnt_val=0;

static void click_event_cb(lv_event_t *e)
{
    lv_event_code_t event_code = e->code;
    lv_obj_t * target = lv_event_get_target(e);
    if(!lv_obj_is_valid(target))
    return;
    if(event_code ==  LV_EVENT_CLICKED)
    {
        uint8_t id = lv_obj_get_child_id(target);
        if(id==1)
        {
            lv_timer_resume(cnt_timer);
        }else if(id==2)
        {
            lv_timer_pause(cnt_timer);
        }
    }
}
static void timer_callback(lv_timer_t *timer)
{
    lv_obj_t * label = timer->user_data;
    if(!lv_obj_is_valid(label))
        return;
    cnt_val++;
    lv_label_set_text_fmt(label,"%02d:%02d",cnt_val/60%60, cnt_val%60);
}
static void label_delete_cb(lv_event_t * e)
{
    if(cnt_timer!=NULL)
    {
        lv_timer_del(cnt_timer);
        cnt_timer=NULL;
    }
}
```

图 8-3 函数示例

8.3. 显示大分辨率图片实例:

由于打包工具会将分辨率大于切图大小时会切成多张图片，应用端显示时需要创建多个 img 对象来显示，可以使用如下接口来显示。

```
lv_obj_t* lv_img_big_create(lv_obj_t* parent, const lv_img_dsc_t **src , lv_coord_t x, lv_coord_t y,uint8_t img_num,uint8_t start)
{
    lv_obj_t *cont1 = lv_obj_create(parent);
    lv_obj_remove_style_all(cont1);
    lv_obj_clear_flag(cont1, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_scrollbar_mode(cont1, LV_SCROLLBAR_MODE_OFF);
    lv_obj_set_style_bg_color(cont1,lv_color_black(),0);
    lv_obj_set_size(cont1,LV_SIZE_CONTENT,LV_SIZE_CONTENT);
    lv_obj_t *old_obj = NULL;
    for(uint8_t i = 0; i < img_num; i++)
    {
        lv_obj_t *img = lv_img_create(cont1);
        lv_img_set_src(img,src[start+i]);
        if(i == 0)
            lv_obj_set_pos(img,0,0);
        else
            lv_obj_align_to(img,old_obj,LV_ALIGN_OUT_BOTTOM_LEFT,0,0);
        old_obj = img;
    }
    return cont1;
}
```

图 8-4 显示大分辨率函数实现示例

```
LV_ATTRIBUTE_LARGE_CONST static const lv_img_dsc_t* clock_bg_src[6] =
{
    IMG_I280002_D9025001_4_BG_P0001_1_1 ,
    IMG_I280002_D9025001_4_BG_P0001_1_2 ,
    IMG_I280002_D9025001_4_BG_P0001_1_3 ,
    IMG_I280002_D9025001_4_BG_P0001_1_4 ,
    IMG_I280002_D9025001_4_BG_P0001_1_5 ,
    IMG_I280002_D9025001_4_BG_P0001_1_6 ,
};
lv_obj_t* clock_bg_img = lv_img_big_create(parent,clock_bg_src,0,0,6,0);
```

图 8-5 显示大分辨率函数调用示例

8.4. 动画显示实例:

显示 gif 动画时，我们要先将 gif 动画转换成多张静态图片，然后软件里面通过定时器播放显示图片的方式实现 gif 的播放，如下代码创建一个 img 对象，同时创建一个定时器。

```c
void fr_app_anim_func(lv_obj_t *parent, lv_point_t *top)
{
    if(!lv_obj_is_valid(parent))
        return;
    UI_PARENT_INIT(parent);
    lv_obj_t* anim_img = lv_img_create(parent);
    lv_img_set_src(anim_img,gif_img[0]);
    lv_obj_align(anim_img,LV_ALIGN_CENTER,0,0);
    anim_timer = lv_timer_create(anim_timer_callback, 100, (void*)anim_img);
    lv_obj_add_event_cb(anim_img,img_delete_cb, LV_EVENT_DELETE, NULL);

}
```

图 8-6 创建 img 对象示例

```c
LV_ATTRIBUTE_LARGE_CONST static const lv_img_dsc_t* gif_img[10]={
    IMG_RESPIRATORY_RATE_GIF_01,
    IMG_RESPIRATORY_RATE_GIF_02,
    IMG_RESPIRATORY_RATE_GIF_03,
    IMG_RESPIRATORY_RATE_GIF_04,
    IMG_RESPIRATORY_RATE_GIF_05,
    IMG_RESPIRATORY_RATE_GIF_06,
    IMG_RESPIRATORY_RATE_GIF_07,
    IMG_RESPIRATORY_RATE_GIF_08,
    IMG_RESPIRATORY_RATE_GIF_09,
    IMG_RESPIRATORY_RATE_GIF_10,
};

static lv_timer_t *anim_timer=NULL;
```

图 8-7 静态图片示例

定时器回调里面处理图片的切换显示，通过对 img 对象的图像源修改，实现动画的显示。

```c
static void anim_timer_callback(lv_timer_t *timer)
{
    static uint8_t play_index=0;
    lv_obj_t * img = timer->user_data;
    if(!lv_obj_is_valid(img))
        return;
    play_index++;
    play_index%=10;
    lv_img_set_src(img,gif_img[play_index]);
}

static void img_delete_cb(lv_event_t * e)
{
    if(anim_timer!=NULL)
    {
        lv_timer_del(anim_timer);
        anim_timer=NULL;
    }
}
```

图 8-8 定时器回调示例

联系方式

欢迎大家针对富芮坤产品和文档提出建议。

反馈：doc@freqchip.com.

网站：www.freqchip.com

销售：sales@freqchip.com

电话：+86-21-5027-0080