

LittelvGL 使用教程

一款极炫的开源 GUI



版权：青岛千亿电子科技有限公司

作者：官庆灿

版本：V0.1

目录

第一章：LittelVGL 介绍.....	7
1.1 图形控件.....	7
1.1.1 控件属性.....	8
1.1.2 控件的工作机制.....	9
1.1.3 创建 - 删除对象.....	11
1.1.4 图层.....	11
1.2 样式.....	13
1.2.1 样式属性.....	13
1.2.2 使用样式.....	14
1.2.3 内置样式.....	15
1.2.4 样式动画.....	16
1.2.5 样式栗子.....	16
1.2.6 主题.....	17
1.3 颜色.....	18
1.4 字体.....	20
1.4.1 内置字体.....	21
1.4.2 Unicode 支持.....	21
1.4.3 符号字体.....	22
1.4.4 添加新的字体.....	23
1.4.5 字体示例.....	24
1.5 动画.....	24
1.6 输入设备.....	26
1.7 触摸导航.....	27
1.8 绘图和渲染.....	28
1.8.1 缓冲和无缓冲绘图.....	28
1.8.2 抗锯齿.....	29
第二章：控件介绍.....	29
2.1 基础控件.....	29
2.1.1 相关宏定义.....	32
2.1.2 API 函数.....	32

2.1.3 控件使用	45
2.2 label	45
2.2.1 相关宏定义	46
2.2.2 API 函数	46
2.2.3 控件使用	52
2.3 image	52
2.3.1 相关宏定义	53
2.3.2 API 函数	53
2.3.2 控件使用	55
2.4 Line	55
2.4.1 相关宏定义	55
2.4.2 API 函数	55
2.4.3 控件使用	58
2.5 Container	58
2.5.1 相关宏定义	59
2.5.2 API 函数	59
2.5.3 控件使用	61
2.6 Page.....	61
2.6.1 相关宏定义.....	62
2.6.2 API 函数	62
2.6.3 控件使用	66
2.7 Window	67
2.7.1 相关宏定义.....	68
2.7.2 API 函数	68
2.7.3 控件使用	72
2.8 Tab View	73
2.8.1 相关宏定义.....	74
2.8.2 API 函数	74
2.8.3 控件使用	77
2.9 Bar	80
2.9.1 相关宏定义	80
2.9.2 API 函数	80

2.9.3 控件使用	82
2.10 Line Meter	84
2.10.1 相关宏定义	84
2.10.2 API 函数	84
2.10.3 控件使用	87
2.11 Gauge	88
2.11.1 相关宏定义	88
2.11.2 API 函数	88
2.11.3 控件使用	91
2.12 Chart	93
2.12.1 相关宏定义	94
2.12.2 API 函数	95
2.12.3 控件使用	99
2.13 LED	101
2.13.1 相关宏定义	102
2.13.2 API 函数	102
2.13.3 控件使用	103
2.14 Message Box	105
2.14.1 相关宏定义	105
2.14.2 API 函数	106
2.14.3 控件使用	109
2.15 Text Area	111
2.15.1 相关宏定义	112
2.15.2 API 函数	112
2.15.3 控件使用	117
2.16 Button	117
2.16.1 相关宏定义	118
2.16.2 API 函数	118
2.16.3 控件使用	121
2.17 Button Matrix	124
2.17.1 相关宏定义	125
2.17.2 API 函数	125

2.17.3 控件使用.....	127
2.18 Keyboard.....	129
2.18.1 相关宏定义	130
2.18.2 API 函数.....	130
2.18.3 控件使用.....	133
2.19 List	134
2.19.1 相关宏定义	134
2.19.2 API 函数.....	135
2.19.3 控件使用.....	138
2.20 Drop Down List	141
2.20.1 相关宏定义	141
2.20.2 API 函数.....	142
2.20.3 控件使用.....	146
2.21 Roller.....	147
2.21.1 相关宏定义	148
2.21.2 API 函数.....	148
2.21.3 控件使用.....	151
2.22 Check Box.....	152
2.22.1 相关宏定义	153
2.22.2 API 函数.....	153
2.22.3 控件使用.....	155
2.23 Slider	157
2.23.1 相关宏定义	157
2.23.2 API 函数.....	158
2.23.3 控件使用.....	161
2.24 Switch	162
2.24.1 相关宏定义	163
2.24.2 API 函数.....	163
2.24.3 控件使用.....	165
第三章：将 LittlevGL 移植到微控制器.....	166
3.1 系统架构.....	166
3.2 硬件需求.....	167

3.3 工程设置	168
3.3.1 获取库	168
3.3.2 配置文件	168
3.3.3 初始化	168
3.4 移植库	168
3.4.1 显示接口	169
3.4.2 输入设备接口	170
3.4.3 滴答时钟接口	172
3.4.4 任务处理	172
3.4.5 移植示例	172
版权说明	173

第一章：LittelvGL 介绍

➤ LittelvGL 介绍

LittelvGL 是一个开源免费的 GUI，支持触摸屏操作，移植简单方便，开发者一直在不断完善更新。LittelvGL 自带了丰富的控件：窗口、按键、标签、list、图表等，还可以自定义控件；支持很多特效：透明、阴影、自动显示隐藏滚动条、界面切换动画、图标打开关闭动画、平滑的拖拽控件、分层显示、反锯齿、仅耗少量内存的字体等等。

LittelvGL 常见于 MCU 级别的设备，支持各类输入输出接口与芯片，支持使用 GPU，以 C 编写。

官网地址：<https://littlevgl.com/>

➤ 硬件平台介绍

酷客 Coidea972 开发板是一款基于新唐芯片 NUC972 设计的一款开发板，开发板上集成超过 20 个外围设备，涉及工业通讯、影音处理、数字存储、现场总线、无线通讯、红外通讯、图形界面、电机控制、环境感知、姿态感知等诸多方面，涵盖了单片机开发板和 ARM 开发板两种开发板的绝大多数外设。

核心板和底板均采用沉金工艺，稳定可靠。核心板采用四层板，独立的电源层和地层，并预留屏蔽罩焊接位置，抗干扰能力极强，可应用于相对恶劣的环境。板对板连接器使用东芝进口接口，稳定可靠，杜绝出现氧化接触不良等现象。

芯片 300M 主频，片上集成 64MB SDRAM 自带 LCD 控制器，多达 11 个串口，特别适合工业 HMI 类应用和通信类应用。

开发板详情：[点我](#)



1.1 图形控件

在 Littlev 图形库中，用户界面的基本构建块是对象。 例如：

- Button
- Label
- Image
- List
- Chart
- Text area

1.1.1 控件属性

基本属性

对象具有与其类型无关的基本属性：

- 位置
- 尺寸
- 句柄
- 启用拖动
- 启用点击等

您可以使用 `lv_obj_set_...` 和 `lv_obj_get_...` 函数设置/获取此属性。例如：

```
1. /*Set basic object attributes*/
2. lv_obj_set_size(btn1, 100, 50);           /*Button size*/
3. lv_obj_set_pos(btn1, 20,30);              /*Button position*/
```

要查看所有可用功能，请访问 [Base 对象的文档](#)

特定属性

对象类型具有特殊属性。例如 `slider` 具有：

- 最大值、最小值
- 当前值
- 回调函数
- 样式

对于这些属性，每个对象类型都具有唯一的 API 函数 例如 `slider`：

```
1. /*Set slider specific attributes*/
2. lv_slider_set_range(slider1, 0, 100);      /*Set min. and max. values*/
3. lv_slider_set_value(slider1, 40);          /*Set the current value (position)*/
4. lv_slider_set_action(slider1, my_action);  /*Set a callback function*/
```

1.1.2 控件的工作机制

父-子结构

父对象可以被视为其子对象的容器。 每个对象只有一个父对象（屏幕除外），但父对象可以拥有无限数量的子对象。 父类型没有限制，但通常有父（例如按钮）和典型的子（例如标签）对象

屏幕 - 最基本的父对象

屏幕是一个没有父对象的特殊对象。 总是有一个活动的屏幕。 默认情况下，库会创建并加载一个。 要获取当前活动的屏幕，请使用 `lv_scr_act()` 函数。 可以使用任何对象类型创建屏幕，例如基本对象或用于制作壁纸的图像。

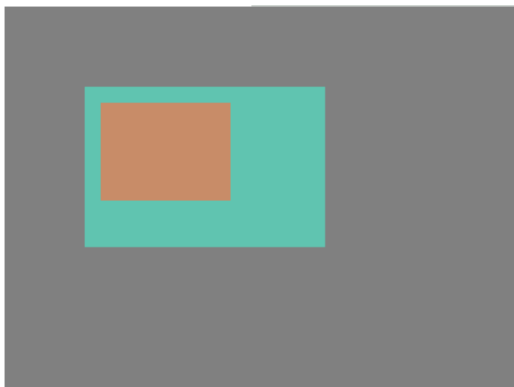
一起移动

如果更改了父项的位置，则子项将与父项一起移动。 因此，所有位置都与父母相关。 因此（0; 0）坐标意味着对象将独立于父对象的位置保留在父对象的左上角



```
1. lv_obj_t * par = lv_obj_create(lv_scr_act(), NULL);    /*Create a parent ob
   object on the current screen*/
2. lv_obj_set_size(par, 100, 80);                        /*Set the size of the pa
   rent*/
3.
4. lv_obj_t * obj1 = lv_obj_create(par, NULL);           /*Create an object on th
   e previously created parent object*/
5. lv_obj_set_pos(obj1, 10, 10);                          /*Set the position of th
   e new object*/
```

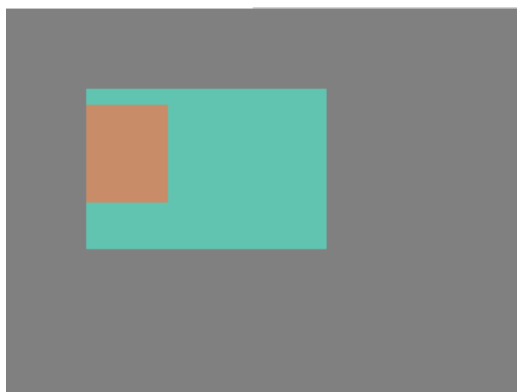
修改父对象的位置:



```
1. lv_obj_set_pos(par, 50, 50);                          /*Move the parent. The child will move w
   ith it.*/
```

仅在父对象上可见

如果子对象部分或完全脱离其父对象，那么外面的部分将不可见。



```
1. lv_obj_set_x(obj1, -30);           /*Move the child a little bit of the parent*/  
   /
```

1.1.3 创建 - 删除对象

在图形库中，可以在运行时动态创建和删除对象。这意味着只有当前创建的对象占用 RAM。例如，如果您需要图表，则只能在需要时创建图表，并在使用图表后删除。

每个对象类型都有自己的创建函数和一个统一的原型。它需要两个参数：指向父对象的指针，以及可选的指向具有相同类型的其他对象的指针。如果第二个参数不为 NULL，则此对象将复制到新对象。要创建屏幕，请将 NULL 设置为父级。create 函数的返回值是指向创建的对象指针。独立于对象类型，使用公共变量类型 lv_obj_t。稍后可以使用此指针来设置或获取对象的属性。创建函数如下所示：

```
lv_obj_t * lv_type_create(lv_obj_t * parent, lv_obj_t * copy);
```

所有对象类型都有一个通用的删除功能。它删除对象及其所有子对象。

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

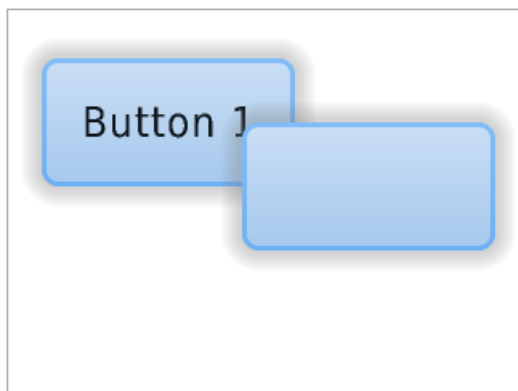
您只能删除对象的子项，但保持对象本身“活动”：

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

1.1.4 图层

较早创建的对象（及其子对象）将更早绘制（更接近背景）。换句话说，最后创建的对象将在其兄弟姐妹中位于顶部。非常重要的一点是，在同一级别的对象（“兄弟姐妹”）之间计算顺序。

通过创建 2 个对象（可以是透明的），首先是“A”，然后是“B”，可以轻松添加图层。'A'和它上面的每个物体都在背景中，可以被'B'和它的孩子覆盖。



```

/*Create a screen*/
lv_obj_t * scr = lv_obj_create(NULL, NULL);
lv_scr_load(scr);                                     /*Load the screen*/

/*Create 2 buttons*/
lv_obj_t * btn1 = lv_btn_create(scr, NULL);           /*Create a
button on the screen*/
lv_btn_set_fit(btn1, true, true);                     /*Enable to
automatically set the size according to the content*/
lv_obj_set_pos(btn1, 60, 40);                         /*Set the
position of the button*/

lv_obj_t * btn2 = lv_btn_create(scr, btn1);           /*Copy the
first button*/
lv_obj_set_pos(btn2, 180, 80);                       /*Set the
position of the button*/

/*Add labels to the buttons*/
lv_obj_t * label1 = lv_label_create(btn1, NULL);      /*Create a
label on the first button*/
lv_label_set_text(label1, "Button 1");                /*Set the text
of the label*/

lv_obj_t * label2 = lv_label_create(btn2, NULL);      /*Create a
label on the second button*/
lv_label_set_text(label2, "Button 2");                /*Set the
text of the label*/

/*Delete the second label*/
lv_obj_del(label2);

```

1.2 样式

可以使用设置对象样式的外观。 样式是一个结构变量，具有颜色，填充，可见性等属性。 有一个共同的样式类型：lv_style_t。

通过设置 lv_style_t 结构的字段，您可以使用该样式影响对象的外观。

对象仅存储指向样式的指针，因此样式不能是在函数存在后销毁的局部变量。您应该使用静态，全局或动态分配的变量。

1.2.1 样式属性

样式有 5 个主要部分：普通，身体，文字，图像和线条。 对象将使用与其相关的字段。 例如 *Lines* 不关联 *letter_space*。 要查看对象类型使用哪些字段，请参阅其文档。

样式结构的字段如下：

共同属性

- **glass 1:** 不要继承这种风格（见下文）

主体样式属性

由类似矩形的对象使用

- **body.empty** 不要填充矩形（只是绘制边框和/或阴影）
- **body.main_color** 主色（顶色）
- **body.grad_color** 渐变色（底色）
- **body.radius** 转角半径。（设置为 LV_RADIUS_CIRCLE 绘制圆圈）
- **body.opa** 不透明度（0..255 或 LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 LV_OPA_COVER）
- **body.border.color** 边框颜色
- **body.border.width** 边框宽度
- **body.border.part** 边框部分（LV_BORDER_LEFT / RIGHT / TOP / BOTTOM / FULL 或 'OR'ed 值）
- **body.border.opa** 边界不透明
- **body.shadow.color** 阴影颜色
- **body.shadow.width** 阴影宽度
- **body.shadow.type** 阴影类型（LV_SHADOW_BOTTOM 或 LV_SHADOW_FULL）

- **body.padding.hor** 水平填充
- **body.padding.ver** 垂直填充
- **body.padding.inner** 内部填充

文本样式属性

由显示文本的对象使用

- **text.color** 文本颜色
- **text.font** 指向字体的指针
- **text.opa** 文本不透明度 (0..255 或 LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)
- **text.letter_space** 字空间
- **text.line_space** 线空间

图像样式属性

由类似图像的对象或对象上的图标使用

- **image.color** 基于像素亮度的图像重新着色的颜色
- **image.intense** 重色强度 (0..255 或 LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 LV_OPA_COVER)
- **image.opa** 图像不透明度 (0..255 或 LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)

线条样式属性

由包含线条或线状元素的对象使用

- **line.color** 线条颜色
- **line.width** 线条宽度
- **line.opa** 线条透明度 (0..255 或 LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)

1.2.2 使用样式

每种对象类型都有一个独特的功能来设置其样式。

如果对象只有一个样式 - 比如标签 - 可以使用

`lv_label_set_style(label1, &style)` 函数来设置新样式。

如果对象具有更多样式（如按钮每个状态有 5 个样式），则可以使用 `lv_btn_set_style(obj, LV_BTN_STYLE_..., &rel_style)` 函数来设置新样式。

对象类型使用的样式和样式属性在其文档中描述。

如果修改由一个或多个对象使用的样式，则必须通知对象有关样式的更改。您有两种选择：

```
void lv_obj_refresh_style(lv_obj_t * obj);      /*Notify an
object about its style is modified*/
void lv_obj_report_style_mod(void * style);     /*Notify all
object if a style is modified.(NULL to notify all objects)*/
```

如果对象的样式为 **NULL**，则其样式将从其父样式继承。它使创建一致的设计变得更容易。不要忘记一种风格同时描述了很多属性。因此，例如，如果您设置按钮的样式并使用 **NULL** 样式在其上创建标签，则将根据按钮样式呈现标签。换句话说，按钮确保其子对象看起来很好。

设置 *glass* 样式属性将阻止继承该样式。如果样式是透明的，则应使用它，以便其子项使用颜色和其父项中的其他颜色。

1.2.3 内置样式

库中有几种内置样式：



正如您所看到的，有一种用于屏幕，按钮，普通和漂亮样式以及透明样式的样式。*lv_style_transp*, *lv_style_transp_fit* 和 *lv_style_transp_tight* 仅在填充方面有所不同：对于 *lv_style_transp_tight* 所有填充都为零，对于 *lv_style_transp_fit* 只有 *hor* 和 *ver* 填充为零。

内置样式是全局 `lv_style_t` 变量，因此您可以使用它们：

```
lv_btn_set_style(obj, LV_BTN_STYLE_REL, &lv_style_btn_rel)
```

您可以修改内置样式，也可以创建新样式。在创建新样式时，建议首先复制内置样式，以确保使用适当的值初始化所有字段。

`lv_style_copy(&dest_style, &src_style)` 可用于复制样式。

1.2.4 样式动画

您可以使用 `lv_style_anim_create(&anim)` 为样式设置动画。在调用此函数之前，您必须初始化 `lv_style_anim_t` 变量。动画会将 `style_1` 淡化为 `style_2`

```
lv_style_anim_t a; /*Will be copied, can be local variable*/
a.style_anim = & style_to_anim; /*Pointer to style to animate*/
a.style_start = & style_1; /*Pointer to the initial style (only pointer saved) */
a.style_end = & style_2; /*Pointer to the target style (only pointer saved) */
a.act_time = 0; /*Set negative to make a delay*/
a.time = 1000; /*Time of animation in milliseconds*/
a.playback = 0; /*1: play the animation backward too*/
a.playback_pause = 0; /*Wait before playback [ms]*/
a.repeat = 0; /*1: repeat the animation*/
a.repeat_pause = 0; /*Wait before repeat [ms]*/
a.end_cb = NULL; /*Call this function when the animation ready*/
```

1.2.5 样式栗子

以下示例演示了上述样式用法



```
1. /*Create a style*/
2. static lv_style_t style1;
3. lv_style_copy(&style1, &lv_style_plain); /*Copy a built-in style to initialize the new style*/
4. style1.body.main_color = LV_COLOR_WHITE;
5. style1.body.grad_color = LV_COLOR_BLUE;
```



```
6. style1.body.radius = 10;
7. style1.body.border.color = LV_COLOR_GRAY;
8. style1.body.border.width = 2;
9. style1.body.border.opa = LV_OPA_50;
10. style1.body.padding.hor = 5;           /*Horizontal padding, used by the bar
      r indicator below*/
11. style1.body.padding.ver = 5;           /*Vertical padding, used by the bar
      indicator below*/
12. style1.text.color = LV_COLOR_RED;
13.
14. /*Create a simple object*/
15. lv_obj_t *obj1 = lv_obj_create(lv_scr_act(), NULL);
16. lv_obj_set_style(obj1, &style1);       /*Apply the created
      style*/
17. lv_obj_set_pos(obj1, 20, 20);          /*Set the position*/

18.
19. /*Create a label on the object. The label's style is NULL by default*/
20. lv_obj_t *label = lv_label_create(obj1, NULL);
21. lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0); /*Align the label to
      the middle*/
22.
23. /*Create a bar*/
24. lv_obj_t *bar1 = lv_bar_create(lv_scr_act(), NULL);
25. lv_bar_set_style(bar1, LV_BAR_STYLE_INDIC, &style1); /*Modify the indicat
      or's style*/
26. lv_bar_set_value(bar1, 70);            /*Set the bar's valu
      e*/
```

1.2.6 主题

为 GUI 创建样式很有挑战性，因为您需要更深入地了解库，并且需要具备一些设计师技能。此外，创建如此多的样式需要花费大量时间。

为了加速设计部分主题的介绍。主题是样式集合，其中包含每种对象类型所需的样式。例如，按钮的 5 种样式用于描述它们的 5 种可能状态。检查现有主题。

更具体地说，主题是一个结构变量，它包含许多 `lv_style_t` * 字段。对于按钮：

```
1. theme.btn.rel      /*Released button style*/
2. theme.btn.pr       /*Pressed button style*/
3. theme.btn.tgl_rel  /*Toggled released button style*/
4. theme.btn.tgl_pr   /*Toggled pressed button style*/
```

```
5. theme.btn.ina      /*Inactive button style*/
```

主题可以通过以下方式初始化: `lv_theme_xxx_init(hue, font)`。其中 `xxx` 是主题的名称, `hue` 是 HSV 颜色空间 (0..360) 的色调值, 字体是主题中应用的字体 (使用 `LV_FONT_DEFAULT` 默认字体为 `NULL`)

初始化主题时, 其样式可以像这样使用:



```
1. /*Create a default slider*/
2. lv_obj_t *slider = lv_slider_create(lv_scr_act(), NULL);
3. lv_slider_set_value(slider, 70);
4. lv_obj_set_pos(slider, 10, 10);
5.
6. /*Initialize the alien theme with a redish hue*/
7. lv_theme_t *th = lv_theme_alien_init(10, NULL);
8.
9. /*Create a new slider and apply the themes styles*/
10. slider = lv_slider_create(lv_scr_act(), NULL);
11. lv_slider_set_value(slider, 70);
12. lv_obj_set_pos(slider, 10, 50);
13. lv_slider_set_style(slider, LV_SLIDER_STYLE_BG, th->slider.bg);
14. lv_slider_set_style(slider, LV_SLIDER_STYLE_INDIC, th->slider.indic);
15. lv_slider_set_style(slider, LV_SLIDER_STYLE_KNOB, th->slider.knob);
```

在创建新对象时, 可以强制库应用主题中的样式。为此, 请使用 `lv_theme_set_current(th)`;

1.3 颜色

颜色模块处理所有颜色相关的功能, 如: 改变颜色深度, 从十六进制代码创建颜色, 颜色深度之间的对话, 混合颜色等。

以下变量类型由颜色模块定义:

- `lv_color1_t` 存储单色。为了兼容性, 它还有 `R`, `G`, `B` 字段但它们总是相同的 (1 字节)
- `lv_color8_t` 用于存储 8 位颜色 (1 字节) 的 `R` (3 位), `G` (3 位), `B` (2 位) 分量的结构

- **lv_color16_t** 用于存储 16 位颜色（2 字节）的 R（5 位），G（6 位），B（5 位）分量的结构
- **lv_color24_t** 存储用于 16 位 cA 结构的 R（5 位），G（6 位），B（5 位）分量以存储用于 24 位的 R（8 位），G（8 位），B（8 位）分量的结构 颜色（4 字节）olors（2 字节）
- **lv_color_t** 根据颜色深度设置等于 color1/8/16 / 24_t
- **lv_color_int_t** uint8_t, uint16_t 或 uint32_t 根据颜色深度设置。 用于从普通数字构建颜色数组。
- **lv_opa_t** 一个简单的 uint8_t 类型来描述不透明度。

lv_color_t, *lv_color1_t* *lv_color8_t*, *lv_color16_t* 和 *lv_color24_t* 类型有四个字段:

- **red** 红色通道
- **green** 绿色通道
- **blue** 蓝色通道
- **full** 红色+绿色+蓝色为一个数字

您可以通过将 *LV_COLOR_DEPTH* 定义设置为 1（单色），8,16 或 24 来设置 *lv_conf.h* 中的当前颜色深度。

您可以将颜色从当前颜色深度转换为另一种颜色深度。 转换器函数返回一个数字，因此您必须使用完整字段：

```

1. lv_color_t c;
2. c.red    = 0x38;
3. c.green  = 0x70;
4. c.blue   = 0xCC;
5.
6. lv_color1_t c1;
7. c1.full = lv_color_to1(c);    /*Return 1 for light colors, 0 for dark colors*/
8.
9. lv_color8_t c8;
10. c8.full = lv_color_to8(c);    /*Give a 8 bit number with the converted color*/
11.
12. lv_color16_t c16;
13. c16.full = lv_color_to16(c);    /*Give a 16 bit number with the converted color*/
14.
15. lv_color24_t c24;
```

```
16. c24.full = lv_color_to24(c);    /*Give a 32 bit number with the converted color*/
```

您可以使用 **LV_COLOR_MAKE** 宏创建具有当前颜色深度的颜色。它需要 3 个参数（红色，绿色，蓝色）作为 8 位数。例如，要创建浅红色：

```
my_color = COLOR_MAKE(0xFF, 0x80, 0x80)
```

也可以从 HEX 代码创建颜色：

```
my_color = LV_COLOR_HEX(0xFF8080) 或 my_color = LV_COLOR_HEX3(0xFF88)
```

混合两种颜色可以使用

```
mixed_color = lv_color_mix(color1, color2, ratio)
```

定量可以是 0..255。0 结果完全 color2, 255 结果完全 color1。

为了描述不透明度，*lv_opa_t* 类型被创建为 *uint8_t* 的包装器。还介绍了一些定义：

- **LV_OPA_TRANSP** 值：0 表示不透明度使颜色完全透明
- **LV_OPA_10** 值：25，表示颜色仅覆盖一点
- **LV_OPA_20 ... OPA_80** 以此类推
- **LV_OPA_90** 值：229，表示完全覆盖附近的颜色
- **LV_OPA_COVER** 值：255，表示颜色完全覆盖

您还可以使用 *lv_color_mix()* 中的 *LV_OPA_** 定义作为比率。

颜色模块定义了最基本的颜色：

```
LV_COLOR_BLACK, LV_COLOR_GRAY, LV_COLOR_SILVER, LV_COLOR_WHITE,
LV_COLOR_RED, LV_COLOR_MARRON,
LV_COLOR_LIME, LV_COLOR_GREEN, LV_COLOR_OLIVE,
LV_COLOR_BLUE, LV_COLOR_NAVY, LV_COLOR_TAIL, LV_COLOR_CYAN,
LV_COLOR_AQUA,
LV_COLOR_PURPLE, LV_COLOR_MAGENTA,
LV_COLOR_ORANGE, LV_COLOR_YELLOW,
```

1.4 字体

在 LittlevGL 中，字体是位图和其他描述符，用于存储字母（字形）的图像和一些附加信息。字体存储在 *lv_font_t* 变量中，可以在样式的 *text.font* 字段中设置。

字体具有 **bpp** (Bit-Per-Pixel) 特性。它显示了用于描述字体中像素的位数。为像素存储的值确定像素的不透明度。这样，字母的图像（特别是在边缘上）可以是平滑和均匀的。可能的 **bpp** 值为 1,2,4 和 8（值越高意味着更好的质量）。**bpp** 还支持存储字体所需的内存大小。例如。与 **bpp = 1** 相比，**bpp = 4** 使字体的内存大小增加 4 倍。

1.4.1 内置字体

有几种内置字体可以通过 `USE_LV_FONT_...` 定义在 `lv_conf.h` 中启用。有不同大小的内置字体：

- 10 px
- 20 px
- 30 px
- 40 px

您可以启用具有 1,2,4 或 8 值的字体来设置其 **bpp**（例如 `USE_LV_FONT_DEJAVU_20 4`）。

内置字体在每个大小中都有多个字符集：

- ASCII (Unicode 32..126)
- Latin supplement (Unicode 160..255)
- Cyrillic (Unicode 1024..1279)

内置字体使用 *Dejavu* 字体。

内置字体是全局变量，名称如下：

- `lv_font_dejavu_20` (20 px ASCII font)
- `lv_font_dejavu_20_latin_sup` (20 px Latin supplement font)
- `lv_font_dejavu_20_cyrillic` (20 px Cyrillic font)

1.4.2 Unicode 支持

LittlevGL 支持 UTF-8 编码字符的 Unicode 字母。您需要配置编辑器以将代码/文本保存为 UTF-8（通常是默认值）并在 `lv_conf.h` 中启用 `LV_TXT_UTF8`。如果未启用 `LV_TXT_UTF8`，则只能使用 ASCII 字体和符号（请参阅下面的符号）

之后，文本将被解码以确定 Unicode 值。要显示字体需要包含字符图像（字形）的字母。

您可以指定更多字体来创建更大的字符集。为此,请选择基本字体(通常是 **ASCII** 字体)并向其添加扩展名: `lv_font_add(child, parent)`。只能分配具有相同高度的字体。



















































内置字体已添加到相同大小的 **ASCII** 字体中。例如,如果在 `lv_conf.h` 中启用了 `USE_LV_FONT_DEJAVU_20` 和 `USE_LV_FONT_DEJAVU_20_LATIN_SUP`,则在使用 `lv_font_dejavu_20` 时可以呈现“abcÁÖÜ”文本。

1.4.3 符号字体

符号字体是包含符号而不是字母的特殊字体。还有内置的符号字体,它们也被分配给具有相同大小的 **ASCII** 字体。在文本中,可以引用符号,如 `SYMBOL_LEFT`, `SYMBOL_RIGHT` 等。您可以将这些符号名称与字符串混合使用: `lv_label_set_text(label1, "Right "SYMBOL_RIGHT);`

符号也可以在没有 **UTF-8** 支持的情况下使用。(`LV_TXT_UTF8 0`)

列表显示现有符号:

	SYMBOL_AUDIO		SYMBOL_PLUS
	SYMBOL_VIDEO		SYMBOL_MINUS
	SYMBOL_LIST		SYMBOL_WARNING
	SYMBOL_OK		SYMBOL_SHUFFLE
	SYMBOL_CLOSE		SYMBOL_UP
	SYMBOL_POWER		SYMBOL_DOWN
	SYMBOL_SETTINGS		SYMBOL_LOOP
	SYMBOL_TRASH		SYMBOL_DIRECTORY
	SYMBOL_HOME		SYMBOL_UPLOAD
	SYMBOL_DOWNLOAD		SYMBOL_CALL
	SYMBOL_DRIVE		SYMBOL_CUT
	SYMBOL_REFRESH		SYMBOL_COPY
	SYMBOL_MUTE		SYMBOL_SAVE
	SYMBOL_VOLUME_MID		SYMBOL_CHARGE
	SYMBOL_VOLUME_MAX		SYMBOL_BELL
	SYMBOL_IMAGE		SYMBOL_KEYBOARD
	SYMBOL_EDIT		SYMBOL_GPS
	SYMBOL_PREV		SYMBOL_FILE
	SYMBOL_PLAY		SYMBOL_WIFI
	SYMBOL_PAUSE		SYMBOL_BATTERY_FULL
	SYMBOL_STOP		SYMBOL_BATTERY_3
	SYMBOL_NEXT		SYMBOL_BATTERY_2
	SYMBOL_EJECT		SYMBOL_BATTERY_1
	SYMBOL_LEFT		SYMBOL_BATTERY_EMPTY
	SYMBOL_RIGHT		SYMBOL_BLUETOOTH

1.4.4 添加新的字体

如果要向库中添加新字体，可以使用[在线字体转换器工具](#)。它可以从 TTF 文件创建一个 C 数组，可以将其复制到项目中。您可以指定高度，字符范围和 bpp。

（可选）您可以枚举字符以仅将它们包含在最终字体中。要使用生成的字体，请使用 `LV_FONT_DECLAER(my_font_name)` 声明它。之后，字体可以用作内置字体。

1.4.5 字体示例

aeuois
äéüöíß

Right ➤

```
1. /*Create a new style for the label*/
2. static lv_style_t style;
3. lv_style_copy(&style, &lv_style_plain);
4. style.text.color = LV_COLOR_BLUE;
5. style.text.font = &lv_font_dejavu_40; /*Unicode and symbol fonts already assigned by the library*/
6.
7. lv_obj_t *label;
8.
9. /*Use ASCII and Unicode letters*/
10. label = lv_label_create(lv_scr_act(), NULL);
11. lv_obj_set_pos(label, 20, 20);
12. lv_label_set_style(label, &style);
13. lv_label_set_text(label, "aeuois\n"
14.                        "äéüöíß");
15.
16. /*Mix text and symbols*/
17. label = lv_label_create(lv_scr_act(), NULL);
18. lv_obj_set_pos(label, 20, 100);
19. lv_label_set_style(label, &style);
20. lv_label_set_text(label, "Right "SYMBOL_RIGHT);
```

1.5 动画

您可以使用带有 `void func(void * var, int32_t value)` 原型的动画函数自动更改变量在开始值和结束值之间的值（动画）。通过使用相应的值参数定期调用动画函数来进行动画。

要创建动画，您必须初始化 `lv_anim_t` 变量（`lv_anim.h` 中有一个模板）：

```
1. lv_anim_t a;
2. a.var = button1; /*Variable to animate*/
3. a.start = 100; /*Start value*/
```



```

4. a.end = 300;                                /*End value*/
5. a.fp = (anim_fp_t)lv_obj_set_height;        /*Function to be used to animate*/
6. a.path = lv_anim_path_linear;               /*Path of animation*/
7. a.end_cb = NULL;                            /*Callback when the animation is ready*/
8. a.act_time = 0;                             /*Set < 0 to make a delay [ms]*/
9. a.time = 200;                               /*Animation length [ms]*/
10. a.playback = 0;                            /*1: animate in reverse direction too when the normal is ready*/
11. a.playback_pause = 0;                      /*Wait before playback [ms]*/
12. a.repeat = 0;                              /*1: Repeat the animation (with or without playback)*/
13. a.repeat_pause = 0;                       /*Wait before repeat [ms]*/
14.
15. lv_anim_create(&a);                        /*Start the animation*/

```

`anim_create(&a)`将注册动画并立即应用起始值，而不管设置的延迟。

您可以确定动画的路径。在大多数简单情况下，它是线性的，这意味着开始和结束之间的当前值线性变化。路径是根据动画的当前状态计算要设置的下一个值的函数。目前有两条内置路径：

- `lv_anim_path_linear` 线性动画
- `lv_anim_path_step` 最后一步改变

默认情况下，您可以设置动画时间。但在某些情况下，动画速度更实用。

`lv_anim_speed_to_time(speed, start, end)`函数计算从具有给定速度的起始值到达结束值所需的时间（以毫秒为单位）。速度以单位/秒维度解释。例如，`anim_speed_to_time(20, 0, 100)`将给出 5000 毫秒

您可以同时也在同一个变量上应用多个不同的动画。（例如，使用 `lv_obj_set_x` 和 `lv_obj_set_y` 为 x 和 y 坐标设置动画）。但是只有一个动画可以与给定的变量和函数对一起存在。因此 `lv_anim_create()` 函数将删除已存在的变量函数动画。

您可以通过 `lv_anim_del(var, func)` 删除动画，同时提供动画变量及其动画制作功能。

1.6 输入设备

要使用创建的对象进行迭代，需要输入设备。例如触摸板，鼠标，键盘甚至编码器。要了解如何添加输入设备，请阅读移植指南。

注册输入设备驱动程序时，库会向其添加一些额外信息，以更详细地描述输入设备的状态。当发生用户动作（例如，按下按钮）并且触发动作（回调）功能时，总是存在触发该动作的输入设备。您可以使用

`lv_indev_t *indev = lv_indev_get_act()` 获取此输入设备。当您需要了解有关输入设备的某些特殊信息（如当前按下的点）或拖动对象时，这可能很重要。

输入设备有一个非常简单的 API：

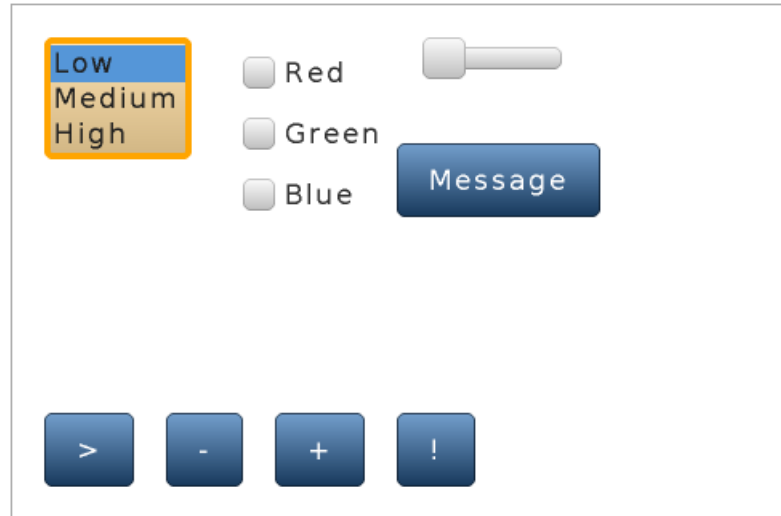
```
1. /*Get the last point on a display input*/
2. void lv_indev_get_point(lv_indev_t * indev, point_t * point);
3.
4. /*Check if there is dragging on input device or not */
5. bool lv_indev_is_dragging(lv_indev_t * indev);
6.
7. /*Get the vector of dragging on a input device*/
8. void lv_indev_get_vect(lv_indev_t * indev, point_t * point);
9.
10. /*Do nothing until the next release*/
11. void lv_indev_wait_release(lv_indev_t * indev);
12.
13. /*Do nothing until the next release*/
14. void lv_indev_wait_release(lv_indev_t * indev);
15.
16. /*Reset one or all (use NULL) input devices*/
17. void lv_indev_reset(lv_indev_t * indev);
18.
19. /*Reset the long pressed state of an input device*/
20. void lv_indev_reset_lpr(lv_indev_t * indev);
21.
22. /*Set a cursor for a pointer input device*/
23. void lv_indev_set_cursor(lv_indev_t * indev, lv_obj_t * cur_obj);
24.
25. /*Set a destination group for a keypad input device*/
26. void lv_indev_set_group(lv_indev_t * indev, lv_group_t * group);
```

1.7 触摸导航

可以对对象进行分组，以便在没有触摸板或鼠标的情况下轻松控制它们。它允许你使用

- 键盘
- 硬件按钮
- 编码器

在对象之间导航。



首先，您必须使用 `lv_groupt_t *group = lv_group_create()` 创建一个对象组，并使用 `lv_group_add_obj(group, obj)` 向其添加对象。在一个小组中总是有一个专注的对象。所有按钮按下将被“发送”到当前关注的对象。

要在组中的对象之间导航（更改聚焦对象）并与它们交互，需要 `LV_INDEV_TYPE_KEYPAD` 类型的输入设备。在其读取功能中，您可以告诉库按下或释放哪个键。要了解如何添加输入设备，请阅读移植指南。

此外，您必须使用 `lv_indev_set_group(indev, group)` 将组分配给输入设备

在读取功能中可以使用一些特殊的控制字符：

- **LV_GROUP_KEY_NEXT** 聚焦于下一个对象
- **LV_GROUP_KEY_PREV** 聚焦于前一个对象
- **LV_GROUP_KEY_UP** 递增值，向上移动或单击聚焦对象（向上移动意味着例如选择上部列表元素）
- **LV_GROUP_KEY_DOWN** 减小值或向下移动焦点对象（向下移动意味着例如选择较低的列表元素）
- **LV_GROUP_KEY_RIGHT** 增加值或单击焦点对象
- **LV_GROUP_KEY_LEFT** 减少焦点对象的值
- **LV_GROUP_KEY_ENTER** 单击焦点对象或选定元素（例如：列表元素）
- **LV_GROUP_KEY_ESC** 关闭对象（例如下拉列表）

在某些情况下（例如，当出现弹出窗口时），将焦点冻结在对象上是有用的。这意味着将忽略 `LV_GROUP_KEY_NEXT/PREV`。您可以使用 `lv_group_focus_freeze(group, true)` 执行此操作。

焦点对象的样式由函数修改。默认情况下，它会使对象的颜色变得橙色，但您也可以使用 `void lv_group_set_style_mod_cb(group, style_mod_cb)` 在每个组中指定自己的样式更新程序功能。`style_mod_cb` 需要一个 `lv_style_t *` 参数，该参数是焦点对象样式的副本。在回调中，您可以将一些颜色混合到当前颜色，并修改参数但不允许设置修改大小的属性（如 `letter_space`, `padding` 等）

1.8 绘图和渲染

在 LittlevGL 中，您可以在图形对象中思考，而不关心绘图是如何发生的。您可以设置对象的大小，位置或任何属性，库将刷新旧（无效）区域并重新绘制新区域。但是，您应该知道基本的绘图方法，以了解您应该选择哪一个。

1.8.1 缓冲和无缓冲绘图

无缓冲的绘图

无缓冲绘图将像素直接放到显示器（帧缓冲区）。因此，在绘制过程中可能会出现一些闪烁，因为首先必须绘制背景，然后绘制其上的对象。因此，在使用滚动，拖动和动画时，此类型不适用。另一方面，它具有最小的内存占用，因为不需要额外的图形缓冲区。

在 `lv_conf.h` 中使用无缓冲的绘图集 `LV_VDB_SIZE` 为 0 并注册 `disp_map` 和 `disp_fill` 函数。在这里您可以了解有关移植的更多信息

缓冲绘图

当使用两个屏幕大小的缓冲区时（一个用于渲染，另一个用于显示最后一个就绪帧），缓冲绘图类似于双缓冲。然而，LittlevGL 的缓冲绘图算法仅使用一个帧缓冲区和一个称为虚拟显示缓冲区（VDB）的小型图形缓冲区。对于 VDB 尺寸 $\sim 1/10$ 的屏幕尺寸通常就足够了。对于具有 16 位颜色的 320×240 屏幕，它意味着仅额外的 15 kB RAM。

使用缓冲绘图时不会出现闪烁，因为图像首先在内存（VDB）中创建，因此可以使用滚动，拖动和动画。此外，它还可以使用其他图形效果，如抗锯齿，透明度（不透明度）和阴影。

要在 `lv_conf.h` 中将缓冲绘图集 `LV_VDB_SIZE` 用于 $> LV_HOR_RES$ 并注册 `disp_flush` 函数。

在缓冲模式下，您可以使用双 VDB 并行执行渲染到一个 VDB，并将像素从另一个复制到帧缓冲区。副本应该使用 DMA 或其他硬件加速在后台工作，让 CPU 做其他事情。在 `lv_conf.h` 中，`LV_VDB_DOUBLE 1` 启用此功能。

缓冲与无缓冲绘图

请记住，它不确定无缓冲的绘图是否更快。在渲染过程中，像素被多次重写（例如，背景，按钮，文本在彼此之上）。这种方式在无缓冲模式下，库需要多次访问外部存储器或显示控制器，这比写入/读取内部 RAM 要慢。

下表总结了两种绘图方法之间的差异：

	Unbuffered drawing	Buffered drawing
Memory usage	No extra	>~1/10 screen
Quality	Flickering	Flawless
Antialiasing	Not supported	Supported
Transparency	Not supported	Supported
Shadows	Not supported	Supported

1.8.2 抗锯齿

在 lv_conf.h 中，您可以使用 `LV_ANTIALIAS 1` 启用抗锯齿功能。仅在缓冲模式（`LV_VDB_SIZE > LV_HOR_RES`）中支持抗锯齿。

抗锯齿的 `algorithm` 使一些半透明像素（具有不透明度的像素）使线条和曲线（包括具有半径的角落）平滑且均匀。因为它只需要一些额外的像素，所以抗噪声只需要一些额外的计算能力（与非抗噪声配置相比，额外时间约为 1,1x）

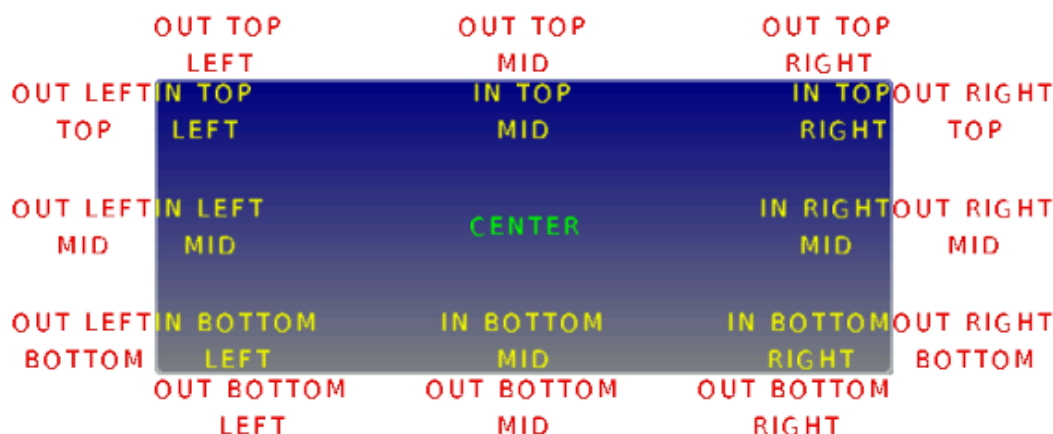
如字体部分所述，可以使用具有更高 `bpp`（Bit-Per-Pixel）的不同字体来防止字体消失。这样，字体的像素不仅可以是 0 或 1，而且可以是半透明的。支持的 `bpp-s` 分别为 1,2,4 和 8.请记住，`bpp` 较高的字体需要更多的 ROM。

第二章：控件介绍

2.1 基础控件



- 基础对象包含对象的最基本属性：
 - 坐标
 - 父对象
 - 子系
 - 样式
 - 点击使能、拖动使能等属性
- 您可以使用 `lv_obj_set_x(obj, new_x)` 和 `lv_obj_set_y(obj, new_y)` 或使用 `lv_obj_set_pos(obj, new_x, new_y)` 在一个函数中设置相对于父级的 x 和 y 坐标
- 可以使用 `lv_obj_set_width(obj, new_width)` 和 `lv_obj_set_height(obj, new_height)` 或使用 `lv_obj_set_size(obj, new_width, new_height)` 在一个函数中修改对象大小
- 您可以使用 `lv_obj_align(obj1, obj2, LV_ALIGN_TYPE, x_shift, y_shift)` 将对象与另一个对齐。最后两个参数表示对齐后的 x 和 y 偏移。第二个参数是另一个对象，其中第一个对齐（`NULL` 表示：与父对齐）。第三个参数是对齐类型：



- 对齐类型构建如下： `LV_ALIGN_OUT_TOP_MID`。例如，对齐图像下方的文本：`lv_obj_align(text, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 10)`。或者在其父级中间对齐文本：`lv_obj_align(text, NULL, LV_ALIGN_CENTER, 0, 0)`
- 您可以使用 `lv_obj_set_parent(obj, new_parent)` 为对象设置新父级。
- 要获取对象的子节点，请使用 `lv_obj_get_child(obj, child_prev)`（从 last 到 first）或 `lv_obj_get_child_back(obj, child_prev)`（从第一个到最后一个）。要获取第一个子节点，将 `NULL` 作为第二个参数传递，然后传递给前一个子节点（返回值）。带 `NULL` 的函数返回不再是子节点
- 当您创建了像 `lv_obj_create(NULL, NULL)` 这样的屏幕时，可以使用 `lv_scr_load(screen1)` 加载它。`lv_scr_act()` 函数为您提供指向当前屏幕的指针。
- 自动生成两层图层：

- 顶层
- 系统层
- 它们独立于屏幕，因此对象创建了一个层，将在每个屏幕上显示。顶层位于屏幕上的每个对象上方，系统层也位于顶层之上。您可以自由添加任何弹出窗口顶层。但是系统层限于系统级事物（例如鼠标光标将在这里移动）。
`lv_layer_top()`和 `lv_layer_sys()`函数提供指向顶层或系统层的指针
- 您可以使用 `lv_obj_set_style(obj, &new_style)`函数为对象设置新样式。如果将 `NULL` 设置为样式，则对象将继承其父样式。如果修改样式，则必须通知正在使用已修改样式的对象。您可以使用 `lv_obj_refresh_style(obj)`或使用给定样式 `lv_obj_report_style_mod(&style)`通知所有对象。将 `lv_obj_report_style_mod`的参数设置为 `NULL` 以通知所有对象。
- `lv_obj_set...(obj, true/false)`可以启用/禁用一些属性：
 - **hidden** 隐藏对象。它不会被绘制，也不会占据空间，它的子系也会被隐藏起来
 - **click** 已启用通过输入设备（例如触摸板）单击对象。如果禁用，则在输入设备单击处理期间将检查此对象后面的对象（对于通常不可点击的对象（如标签）很有用）
 - **top** 如果启用，则单击此对象或其任何子对象时，此对象将显示在前台
 - **drag** 启用拖动（通过 `n` 输入设备移动）
 - **drag_throw** 通过拖动启用“投掷”，就像对象会有动量一样
 - **drag_parent** 如果启用，则在拖动期间将移动对象的父级
- 在库中会自动执行一些特定操作。要防止一种或多种此类操作，您可以保护对象不受其影响。存在以下保护：
 - **LV_PROTECT_NONE** 没有保护
 - **LV_PROTECT_POS** 防止自动定位（例如 `lv_cont` 中的布局）
 - **LV_PROTECT_FOLLOW** 防止在自动排序中遵循对象（例如 `lv_cont` 中的布局）
 - **LV_PROTECT_PARENT** 防止自动更改父级
 - **LV_PROTECT_CHILD_CHG** 禁用子系更改信号。由库使用
- `lv_obj_set/clear_protect(obj, LV_PROTECT...)`设置/清除保护。您也可以使用保护类型的“或”值。
- 有对象的内置动画。存在以下动画类型：
 - **LV_ANIM_FLOAT_TOP** 从顶部浮动/浮动到顶部
 - **LV_ANIM_FLOAT_LEFT** 从左边浮动/浮动到左边

- **LV_ANIM_FLOAT_BOTTOM** 从底部浮动/浮动到底部
- **LV_ANIM_FLOAT_RIGHT** 从右边浮动/浮动到右边
- **LV_ANIM_GROW_H** 水平增长/收缩
- **LV_ANIM_GROW_V** 垂直增长/缩小
- **lv_obj_animate(obj, anim_type, time, delay, callback)** 在 *obj* 上应用动画。使用动画类型确定动画 **OR ANIM_IN** 或 **ANIM_OUT** 的方向。如果未指定，则默认为 **ANIM_IN**。您可以了解有关动画的更多信息。

2.1.1 相关宏定义

2.1.2 API 函数

函数	描述
lv_obj_create ()	创建一个基本对象
lv_obj_del()	删除'obj'及其所有子项
lv_obj_clean()	删除对象的所有子项
lv_obj_invalidate()	将对象标记为无效，因此其当前位置将由 'lv_refr_task' 重绘
lv_scr_load()	加载新屏幕
lv_obj_set_parent()	为对象设置新父级。它的相对位置是一样的
lv_obj_set_pos()	设置相对对象的位置
lv_obj_set_x()	设置对象的 x 坐标
lv_obj_set_y()	设置对象的 y 坐标
lv_obj_set_size()	设置对象的大小
lv_obj_set_width()	设置对象的宽度
lv_obj_set_height()	设置对象的高度
lv_obj_align()	将对象与其他对象对齐
lv_obj_set_style()	为对象设置新样式
lv_obj_refresh_style()	通知对象有关其样式的修改
lv_obj_report_style_mod()	如果修改了样式，则通知所有对象
lv_obj_set_hidden()	隐藏对象。它不可见且可点击
lv_obj_set_click()	启用或禁用单击对象
lv_obj_set_top()	如果单击此对象或其任何子对象，则启用此对象
lv_obj_set_drag()	启用拖动对象
lv_obj_set_drag_throw()	启用拖动后继续拖出对象
lv_obj_set_drag_parent()	允许使用父项进行与拖动相关的操作。如果尝试拖动对象，则将移动父项
lv_obj_set_protect()	在保护字段中设置一个或多个位
lv_obj_clear_protect()	清除保护字段中的一个或多个位

lv_obj_set_signal_func()	设置对象的信号功能。一直调用新的前一个信号功能
lv_obj_set_design_func()	为对象设置新的设计功能
lv_obj_allocate_ext_attr()	分配新的分机.对象的数据
lv_obj_refresh_ext_size()	向对象发送 'LV_SIGNAL_REFR_EXT_SIZE'信号
lv_obj_set_free_num()	为对象设置特定于应用程序的编号
lv_obj_set_free_ptr()	为对象设置特定于应用程序的指针
lv_scr_act()	返回一个指向活动屏幕的指针
lv_layer_top()	返回顶层
lv_layer_sys()	返回系统层
lv_obj_get_screen()	返回对象的屏幕
lv_obj_get_parent()	返回对象的父级
lv_obj_get_child_back()	遍历对象的子对象
lv_obj_count_children()	计算对象的子项
lv_obj_get_coords()	将对象的坐标复制到某个区域
lv_obj_get_x()	获取对象的 x 坐标
lv_obj_get_y()	获取对象的 y 坐标
lv_obj_get_width()	获取对象的宽度
lv_obj_get_height()	获取对象的高度
lv_obj_get_ext_size()	获取对象的扩展大小属性
lv_obj_get_style()	获取对象的样式指针
lv_obj_get_hidden()	获取对象的隐藏属性
lv_obj_get_click()	获取对象的单击启用属性
lv_obj_get_top()	获取对象的置顶启用属性
lv_obj_get_drag()	获取对象的拖动启用属性
lv_obj_get_drag_throw()	获取对象的拖动启用属性
lv_obj_get_drag_parent()	获取对象的拖动父属性
lv_obj_get_protect()	获取对象的保护字段
lv_obj_is_protected()	检查给定保护位域的至少一位被设置
lv_obj_get_signal_func()	获取对象的信号功能
lv_obj_get_design_func()	获取对象的设计功能
lv_obj_get_ext_attr()	获取 ext 指针
lv_obj_get_type()	获取对象及其祖先类型
lv_obj_get_free_num()	获取空闲数量
lv_obj_get_free_ptr()	获得空闲指针
lv_obj_get_group()	获取对象的组

lv_obj_create()

创建一个基本对象

lv_obj_t * lv_obj_create(lv_obj_t * parent, lv_obj_t * copy)

参数	描述
----	----

parent	指向父对象的指针。如果为 NULL，则将创建一个屏幕
copy	指向页面对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向新对象的指针

lv_obj_del()

删除'obj'及其所有子项

lv_res_t lv_obj_del(lv_obj_t * obj)

参数	描述
obj	指向要删除的对象的指针

返回值：LV_RES_INV 因为该对象被删除

lv_obj_clean()

删除对象的所有子项

void lv_obj_clean(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：无

lv_obj_invalidate()

将对象标记为无效，因此其当前位置将由'lv_refr_task'重绘

void lv_obj_invalidate(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：无

lv_scr_load()

加载新屏幕

void lv_scr_load(lv_obj_t * scr)

参数	描述
scr	指向屏幕的指针

返回值：无

lv_obj_set_parent()

为对象设置新父级。它的相对位置是一样的

void lv_obj_set_parent(lv_obj_t * obj, lv_obj_t * parent)

参数	描述
obj	指向对象的指针
parent	指向新父对象的指针

返回值：无

lv_obj_set_pos()

设置相对对象的位置（相对于父对象）

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

参数	描述
obj	指向对象的指针
x	距离父对象左侧的新距离
y	距离父对象顶部的新距离

返回值：无

`lv_obj_set_x()`

设置对象的 x 坐标

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

参数	描述
obj	指向对象的指针
x	距离父对象左侧的新距离

返回值：无

`lv_obj_set_y()`

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

参数	描述
obj	指向对象的指针
y	距离父对象顶部的新距离

返回值：无

`lv_obj_set_size()`

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	描述
obj	指向对象的指针
w	新宽度
h	新高度

返回值：无

`lv_obj_set_width()`

设置对象的宽度

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

参数	描述
obj	指向对象的指针
w	新宽度

返回值：无

`lv_obj_set_height()`

设置对象的高度

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

参数	描述
obj	指向对象的指针
h	新高度

返回值：无

lv_obj_align()

将对象与其他对象对齐

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

参数	描述
obj	指向对象的指针
base	新高度
align	对齐类型（参见'lv_align_t'枚举）
x_mod	对齐后的 x 坐标移位
y_mod	对齐后的 y 坐标移位

返回值：无

lv_obj_set_style()

为对象设置新样式

void lv_obj_set_style(lv_obj_t * obj, lv_style_t * style)

参数	描述
obj	指向对象的指针
style	指向新样式的指针

返回值：无

lv_obj_refresh_style()

通知对象有关其样式的修改

void lv_obj_refresh_style(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：无

lv_obj_report_style_mod()

如果修改了样式，则通知所有对象

void lv_obj_report_style_mod(lv_style_t * style)

参数	描述
style	样式指针的样式。 仅通知具有此样式的对象（NULL 以通知所有对象）

返回值：无

lv_obj_set_hidden()

隐藏对象。 它不可见且可点击

void lv_obj_set_hidden(lv_obj_t * obj, bool en)

参数	描述
obj	指向对象的指针
en	true: 隐藏对象

返回值：无

lv_obj_set_click()

启用或禁用单击对象

void lv_obj_set_click(lv_obj_t * obj, bool en)

参数	描述
obj	指向对象的指针
en	true: 使对象可单击

返回值：无

lv_obj_set_top()

如果单击此对象或其任何子对象，则启用此对象

void lv_obj_set_top(lv_obj_t * obj, bool en)

参数	描述
obj	指向对象的指针
en	true: 启用自动置顶功能

返回值：无

lv_obj_set_drag()

启用拖动对象

void lv_obj_set_drag(lv_obj_t * obj, bool en)

参数	描述
obj	指向对象的指针
en	true: 使对象可拖动

返回值：无

lv_obj_set_drag_throw()

启用拖动后继续拖出对象

void lv_obj_set_drag_throw(lv_obj_t * obj, bool en)

参数	描述
obj	指向对象的指针
en	true: 启用拖动投掷

返回值：无

lv_obj_set_drag_parent()

允许使用父项进行与拖动相关的操作。如果尝试拖动对象，则将移动父项

void lv_obj_set_drag_parent(lv_obj_t * obj, bool en)

参数	描述
obj	指向对象的指针
en	true: 为对象启用“拖动父级”

返回值：无

lv_obj_set_protect()

在保护字段中设置一个或多个位

void lv_obj_set_protect(lv_obj_t * obj, uint8_t prot)

参数	描述
obj	指向对象的指针
prot	来自 lv_obj_prot_t 的'OR'ed 值

返回值：无

lv_obj_clear_protect()

清除保护字段中的一个或多个位

void lv_obj_clear_protect(lv_obj_t * obj, uint8_t prot)

参数	描述
obj	指向对象的指针
prot	来自 lv_obj_prot_t 的'OR'ed 值

返回值：无

lv_obj_set_signal_func()

设置对象的信号功能。一直调用新的前一个信号功能

void lv_obj_set_signal_func(lv_obj_t * obj, lv_signal_func_t fp)

参数	描述
obj	指向对象的指针
fp	新的信号功能

返回值：无

lv_obj_set_design_func()

为对象设置新的设计功能

void lv_obj_set_design_func(lv_obj_t * obj, lv_design_func_t fp)

参数	描述
obj	指向对象的指针
fp	新的设计功能

返回值：无

lv_obj_allocate_ext_attr()

分配新的分机.对象的数据

void * lv_obj_allocate_ext_attr(lv_obj_t * obj, uint16_t ext_size)

参数	描述
obj	指向对象的指针
ext_size	新分机的大小。 数据

返回值：指向分配的 ext 的正常指针

lv_obj_refresh_ext_size()

向对象发送'LV_SIGNAL_REFR_EXT_SIZE'信号

void lv_obj_refresh_ext_size(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：无

lv_obj_set_free_num()

为对象设置特定于应用程序的编号

它可以帮助识别应用程序中的对象

void lv_obj_set_free_num(lv_obj_t * obj, LV_OBJ_FREE_NUM_TYPE free_num)

参数	描述
obj	指向对象的指针
free_num	新的空闲编号

返回值：无

lv_obj_set_free_ptr()

为对象设置特定于应用程序的指针

它可以帮助识别应用程序中的对象

void lv_obj_set_free_ptr(lv_obj_t * obj, void * free_p)

参数	描述
obj	指向对象的指针
free_p	新的空闲指针

返回值：无

lv_obj_animate()

为对象设置动画

void lv_obj_animate(lv_obj_t * obj, lv_anim_builtin_t type, uint16_t time, uint16_t delay, void (*cb) (lv_obj_t *))

参数	描述
obj	指向对象的指针
type	来自'lv_anim_builtin_t'的动画类型。 'OR'与 ANIM_IN 或 ANIM_OUT
time	动画时间，以毫秒为单位
delay	动画之前的延迟，以毫秒为单位
cb	动画准备就绪时调用的函数

返回值：无

lv_scr_act()

返回一个指向活动屏幕的指针

lv_obj_t * lv_scr_act(void)

参数	描述
无	

返回值：指向活动屏幕对象的指针（由'lv_scr_load（）'加载）

lv_layer_top()

返回顶层。（在每个屏幕上都相同，它在正常屏幕层之上）

lv_obj_t * lv_layer_top(void)

参数	描述
无	

返回值：指向顶层对象的指针（透明屏幕大小为 lv_obj）

lv_layer_sys()

返回系统层。（在每个屏幕上都相同，它在所有其他图层之上）

例如由光标使用

lv_obj_t * lv_layer_sys(void)

参数	描述
无	

返回值：指向系统层对象的指针（透明屏幕大小为 lv_obj）

lv_obj_get_screen()

返回对象的屏幕

lv_obj_t * lv_obj_get_screen(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：指向屏幕的指针

lv_obj_get_parent()

返回对象的父级

lv_obj_t * lv_obj_get_parent(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：指向'obj'父项的指针

lv_obj_get_child()

遍历对象的子对象（从“最年轻的”开始）

lv_obj_t * lv_obj_get_child(lv_obj_t * obj, lv_obj_t * child)

参数	描述
obj	指向对象的指针
child	第一次调用时为 NULL 以获取下一个子项以及之后的返回值

返回值：'act_child'之后的孩子，如果没有子项，则为 NULL

lv_obj_get_child_back()

遍历对象的子对象（从“最老的”开始）

lv_obj_t * lv_obj_get_child_back(lv_obj_t * obj, lv_obj_t * child)

参数	描述
obj	指向对象的指针
child	第一次调用时为 NULL 以获取下一个子项以及之后的返回值

返回值: 'act_child'之后的孩子, 如果没有子项, 则为 **NULL**

lv_obj_count_children()

计算对象的子项 (仅直接在'obj'上的子项)

uint16_t lv_obj_count_children(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: 'obj'子项的数量

lv_obj_get_coords()

将对象的坐标复制到某个区域

void lv_obj_get_coords(lv_obj_t * obj, lv_area_t * cords_p)

参数	描述
obj	指向对象的指针
cords_p	指向存储坐标的区域的指针

返回值: 无

lv_obj_get_x()

获取对象的 x 坐标

lv_coord_t lv_obj_get_x(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: 'obj'距其父项左侧的距离

lv_obj_get_y()

获取对象的 y 坐标

lv_coord_t lv_obj_get_y(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: 'obj'距其父项顶部的距离

lv_obj_get_width()

获取对象的宽度

lv_coord_t lv_obj_get_width(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: 宽度

lv_obj_get_height()

获取对象的高度

lv_coord_t lv_obj_get_height(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：高度

lv_obj_get_ext_size()

获取对象的扩展大小属性

lv_coord_t lv_obj_get_ext_size(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：扩展大小属性

lv_obj_get_style()

获取对象的样式指针（如果 NULL 获取父项的样式）

lv_style_t * lv_obj_get_style(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：指向一种样式

lv_obj_get_hidden()

获取对象的隐藏属性

bool lv_obj_get_hidden(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：true：隐藏对象

lv_obj_get_click()

获取对象的单击启用属性

bool lv_obj_get_click(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：true：对象是可点击的

lv_obj_get_top()

获取对象的置顶启用属性

bool lv_obj_get_top(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：true：启用自动置顶功能

lv_obj_get_drag()

获取对象的拖动启用属性

bool lv_obj_get_drag(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: true: 对象是可拖动的

lv_obj_get_drag_throw()

获取对象的拖动启用属性

bool lv_obj_get_drag_throw(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: true: 启用拖动抛出

lv_obj_get_drag_parent()

获取对象的拖动父属性

bool lv_obj_get_drag_parent(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: true: 启用拖动父级

lv_obj_get_protect()

获取对象的保护字段

uint8_t lv_obj_get_protect(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: 保护字段 ('或'lv_obj_prot_t 的值)

lv_obj_is_protected()

检查给定保护位域的至少一位被设置

bool lv_obj_is_protected(lv_obj_t * obj, uint8_t prot)

参数	描述
obj	指向对象的指针
prot	保护要测试的位 ('或'lv_obj_prot_t 的值)

返回值: false: 没有设置给定的位,true: 至少设置一位

lv_obj_get_signal_func()

获取对象的信号功能

lv_signal_func_t lv_obj_get_signal_func(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值: 信号功能

lv_obj_get_design_func()

获取对象的设计功能

lv_design_func_t lv_obj_get_design_func(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：设计功能

lv_obj_get_ext_attr()

获取 ext 指针

void * lv_obj_get_ext_attr(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：ext 指针但不是动态版本使用它作为 ext-> data1，而不是 da(ext)->data1

lv_obj_get_type()

获取对象及其祖先类型。 将他们的名字放在以当前类型开头的`type_buf`中

例如:buf.type [0] = “lv_btn”， buf.type [1] = “lv_cont”， buf.type [2] = “lv_obj”

void lv_obj_get_type(lv_obj_t * obj, lv_obj_type_t * buf)

参数	描述
obj	指向对象的指针
buf	指向`lv_obj_type_t`缓冲区的指针，用于存储类型

返回值：无

lv_obj_get_free_num()

获取空闲数量

lv_obj_get_free_num(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：空闲数量

lv_obj_get_free_ptr()

获得空闲指针

void * lv_obj_get_free_ptr(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：空闲指针

lv_obj_get_group()

获取对象的组

void * lv_obj_get_group(lv_obj_t * obj)

参数	描述
obj	指向对象的指针

返回值：指向对象组的指针

2.1.3 控件使用

2.2 label



- 标签是显示文本的基本对象。文本大小没有限制，因为它是动态存储的。您可以随时使用 `lv_label_set_text()` 在运行时修改文本
- 您可以使用 `\n` 进行换行。例如: `"line1\nline2\n\nline4"`
- 标签对象的大小可以自动扩展到文本大小，或者可以根据几个长模式策略操作文本：
 - `LV_LABEL_LONG_EXPAND`: 将对象大小扩展为文本大小
 - `LV_LABEL_LONG_BREAK`: 保持对象宽度，打破（换行）太长的线条并展开对象高度
 - `LV_LABEL_LONG_DOTS`: 保持对象大小，打破文本并在最后一行写入点
 - `LV_LABEL_LONG_SCROLL`: 展开对象大小并滚动父对象上的文本（移动标签对象）
 - `LV_LABEL_LONG_ROLL`: 保持大小并滚动文本（而不是对象）
- 您可以使用以下命令指定长模式: `lv_label_set_long_mode(label, long_mode)`
- 标签能够显示静态数组中的文本。使用: `lv_label_set_static_text(label, char_array)`。在这种情况下，文本不存储在动态存储器中，而是使用给定的数组。保留在我的数组中不能是函数退出时销毁的局部变量
- 您还可以使用原始字符数组作为标签文本。数组不必是 `\0` 终止。在这种情况下，文本将保存到动态内存中。要设置原始字符数组，请使用 `lv_label_set_array_text(label, char_array)` 函数
- 标签的文本可以与左侧或中间对齐: `lv_label_set_align(label, LV_LABEL_ALIGN_LEFT/CENTER)`
- 您可以使用 `lv_label_set_body_draw(label, draw)` 为标签绘制背景

- 在文本中,您可以使用命令重新着色部分文本。例如: "Write a #ff0000 red# word"。可以通过 `lv_label_set_recolor()` 函数为每个标签单独启用此功能
- 标签可以显示除字母之外的符号。

2.2.1 相关宏定义

长模式行为(<code>lv_label_long_mode_t</code>)	
<code>LV_LABEL_LONG_EXPAND</code>	将对象大小扩展为文本大小
<code>LV_LABEL_LONG_BREAK</code>	保持对象宽度, 打破太长的线条并扩展对象高度
<code>LV_LABEL_LONG_SCROLL</code>	展开对象大小并滚动父对象上的文本 (移动标签对象)
<code>LV_LABEL_LONG_DOT</code>	如果文字太长, 请保持大小并在末尾写点
<code>LV_LABEL_LONG_ROLL</code>	保持大小并无限滚动文本

2.2.2 API 函数

函数	描述
<code>lv_label_create ()</code>	创建标签控件
<code>lv_label_set_text()</code>	为标签设置新文本
<code>v_label_set_array_text()</code>	从字符数组中为标签设置新文本
<code>lv_label_set_static_text()</code>	设置静态文本
<code>lv_label_set_long_mode()</code>	使用较长的文本设置标签的行为, 然后设置对象大小
<code>lv_label_set_align()</code>	设置标签的对齐方式(左侧或中间)
<code>v_label_set_recolor()</code>	通过内联命令启用重新着色
<code>lv_label_set_no_break()</code>	将标签设置为忽略(或接受)'\ n'上的换行符
<code>lv_label_set_body_draw()</code>	设置标签以绘制(或不绘制)其样式主体中指定的背景
<code>lv_label_set_anim_speed()</code>	在 <code>LV_LABEL_LONG_ROLL</code> 和 <code>SCROLL</code> 模式下设置标签的动画速度
<code>lv_label_get_text()</code>	获取标签文本
<code>lv_label_get_long_mode()</code>	获得标签的长模式
<code>lv_label_get_align()</code>	获取对齐属性
<code>lv_label_get_recolor()</code>	获取重新着色属性
<code>lv_label_get_no_break()</code>	获取 <code>no break</code> 属性
<code>lv_label_get_body_draw()</code>	获取主体绘制属性
<code>lv_label_get_anim_speed()</code>	获取标签在

	LV_LABEL_LONG_ROLL 和 SCROLL 模式的动画速度
lv_label_get_letter_pos()	获取字母的相对 x 和 y 坐标
lv_label_get_letter_on()	获取标签相对点上的字母索引
lv_label_ins_text()	在标签上插入文字。 标签文本不能是静态的
lv_label_cut_text()	从标签中删除字符。 标签文本不能是静态的

lv_label_create()

创建标签控件

lv_obj_t * lv_label_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新标签的父控件
copy	指向标签对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向创建标签的指针

lv_label_set_text()

为标签设置新文本。 将分配内存以按标签存储文本

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

参数	描述
label	指向标签控件的指针
text	'\0'终止字符串。 使用当前文本刷新 NULL

返回值：无

lv_label_set_array_text()

从字符数组中为标签设置新文本。 数组不必是'\0'终止

将分配内存以按标签存储数组

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

参数	描述
label	指向标签控件的指针
array	字符数组或 NULL 以刷新标签
size	'array'的大小，以字节为单位

返回值：无

lv_label_set_static_text()

设置静态文本。 它不会被标签保存，因此当标签存在时，'text'变量必须是'alive'

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

参数	描述
----	----

label	指向标签控件的指针
text	指向文本的指针。 使用当前文本刷新 NULL

返回值：无

lv_label_set_long_mode()

使用较长的文本设置标签的行为，然后设置对象大小

void lv_label_set_long_mode(lv_obj_t * label, lv_label_long_mode_t long_mode)

参数	描述
label	指向标签控件的指针
long_mode	来自 lv_label_long_mode 枚举的新模式

返回值：无

lv_label_set_align()

设置标签的对齐方式（左侧或中间）

void lv_label_set_align(lv_obj_t *label, lv_label_align_t align)

参数	描述
label	指向标签控件的指针
align	LV_LABEL_ALIGN_LEFT 或者 LV_LABEL_ALIGN_RIGHT

返回值：无

lv_label_set_recolor()

通过内联命令启用重新着色

void lv_label_set_recolor(lv_obj_t * label, bool recolor_en)

参数	描述
label	指向标签控件的指针
recolor_en	true: 启用重新着色, false: 禁用

返回值：无

lv_label_set_no_break()

将标签设置为忽略（或接受）'\n'上的换行符

void lv_label_set_no_break(lv_obj_t * label, bool no_break_en)

参数	描述
label	指向标签控件的指针
no_break_en	true: 忽略换行符, false: 在'\n'上设置换行符

返回值：无

lv_label_set_body_draw()

设置标签以绘制（或不绘制）其样式主体中指定的背景

void lv_label_set_body_draw(lv_obj_t *label, bool body_en)

参数	描述
label	指向标签控件的指针
body_en	true: 绘制 false:不绘制

返回值： 无

lv_label_set_anim_speed()

在 LV_LABEL_LONG_ROLL 和 SCROLL 模式下设置标签的动画速度

void lv_label_set_anim_speed(lv_obj_t *label, uint16_t anim_speed)

参数	描述
label	指向标签控件的指针
anim_speed	以 px / sec 为单位的动画速度

返回值： 无

lv_label_set_style()

设置文本样式

static inline void lv_label_set_style(lv_obj_t *label, lv_style_t *style)

参数	描述
label	指向标签控件的指针
style	样式指针

返回值： 无

lv_label_get_text()

获取标签文本

char * lv_label_get_text(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值： 标签的文字

lv_label_get_long_mode()

获得标签的长模式

lv_label_long_mode_t lv_label_get_long_mode(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值： 长模式

lv_label_get_align()

获取对齐属性

lv_label_align_t lv_label_get_align(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值： LV_LABEL_ALIGN_LEFT 或 LV_LABEL_ALIGN_CENTER

lv_label_get_recolor()

获取重新着色属性

bool lv_label_get_recolor(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值: true: 启用重新着色, false: 禁用

lv_label_get_no_break()

获取 no break 属性

bool lv_label_get_no_break(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值: true: no_break_enabled (忽略'\n'换行符); false: 在'\n'上换行

lv_label_get_body_draw()

获取主体绘制属性

bool lv_label_get_body_draw(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值: true: 绘制; false: 不绘制

lv_label_get_anim_speed()

获取标签在 LV_LABEL_LONG_ROLL 和 SCROLL 模式的动画速度

uint16_t lv_label_get_anim_speed(lv_obj_t * label)

参数	描述
label	指向标签控件的指针

返回值: 以 px / sec 为单位的动画速度

lv_label_get_letter_pos()

获取字母的相对 x 和 y 坐标

void lv_label_get_letter_pos(lv_obj_t * label, uint16_t index, lv_point_t * pos)

参数	描述
label	指向标签控件的指针
index	字母索引[0 ...文本长度]。用字符索引表示, 而不是字节索引 (UTF-8 不同)
pos	将结果存储在此处 (例如, 索引= 0 给出 0; 0 坐标)

返回值: 无

lv_label_get_letter_on()

获取标签相对点上的字母索引

uint16_t lv_label_get_letter_on(lv_obj_t * label, lv_point_t * pos)

参数	描述
label	指向标签控件的指针
pos	指向标签上带坐标的点

返回值: 'pos_p'点上字母的索引（例如 0; 0 是 0 字母）用字符索引表示而不是字节索引（UTF-8 不同）

lv_label_ins_text()

在标签上插入文字。 标签文本不能是静态的

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

参数	描述
label	指向标签控件的指针
pos	要插入的字符索引。 用字符索引表示而不是字节索引（UTF-8 不同） 0: 在第一个 char 之前。 LV_LABEL_POS_LAST: 在最后一个 char 之后
txt	指向要插入的文本的指针

返回值: 无

lv_label_cut_text()

从标签中删除字符。 标签文本不能是静态的

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

参数	描述
label	指向标签控件的指针
pos	要插入的字符索引。 用字符索引表示而不是字节索引（UTF-8 不同） 0: 在第一个 char 之前
cnt	要剪切的字符数

返回值: 无

lv_label_get_style()

获取文本样式

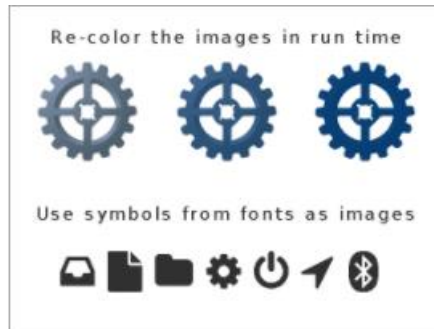
static inline lv_style_t* lv_label_get_style(lv_obj_t *label)

参数	描述
label	指向标签控件的指针

返回值: 样式指针

2.2.3 控件使用

2.3 image



- 图像控件是显示图像的基本对象。 为了提供最大的灵活性，图像的来源可以是：
 - 代码中的变量（带有像素的 C 数组）
 - 外部存储的文件（如在 SD 卡上）
 - 带符号的文字
- 要设置图像的源，可以使用 `lv_img_set_src` 函数
- 要从 PNG, JPG 或 BMP 图像生成像素阵列，请使用[在线图像转换器工具](#)，并使用其指针设置转换后的图像：`lv_img_set_src(img1, &converted_img_var)`
- 要使用外部文件，您还需要使用在线转换器工具转换图像文件，但现在应选择二进制输出格式。 要了解如何处理 LittlevGL 的外部图像文件，请查看[教程](#)
- 您也可以从 `lv_symbol_def.h` 设置符号。 在这种情况下，图像将根据样式中指定的字体呈现为文本。 它可以使用轻量级单色“字母”而不是真实图像。 您可以像这样设置符号：`lv_img_set_src(img1, SYMBOL_OK)`
- 内部（可变）和外部图像支持 2 种透明度处理方法：
 - **Chrome keying** `LV_COLOR_TRANSP` (`lv_conf.h`) 将是透明的
 - **Alpha byte** 为每个像素添加一个 alpha 字节
- 可以在在线字体转换器中选择这些选项
- 根据像素的亮度，可以在运行时将图像重新着色为任何颜色。 显示图像的不同状态（选定，非活动，按下等）非常有用，而不存储同一图像的更多版本。 通过在 `LV_OPA_TRANSP`（无重新着色，值：0）和 `LV_OPA_COVER`（完全重新着色，值：255）之间设置 `img.intense`，可以在样式中启用此功能。 默认值为 `LV_OPA_TRANSP`，因此禁用此功能

- 如果由 `lv_img_set_auto_size(image, true)` 函数启用，则可以自动将图像对象的大小设置为图像源的宽度和高度。如果启用了自动大小，则在设置新文件时，会自动更改对象大小。稍后您可以手动修改大小。如果物体尺寸大于任何方向上的图像尺寸，则图像将像马赛克一样重复。如果图像不是屏幕，则默认启用自动尺寸
- 图像的默认样式为 `NULL`，因此它们继承父项的样式

2.3.1 相关宏定义

2.3.2 API 函数

函数	描述
<code>lv_img_create ()</code>	创建图像控件
<code>lv_img_set_src()</code>	设置像素图以显示图像
<code>lv_img_set_auto_size()</code>	启用自动调整大小功能
<code>lv_img_get_src_type()</code>	获取图像源的类型
<code>lv_img_get_file_name()</code>	获取图像文件集的名称
<code>lv_img_get_auto_size()</code>	获取自动大小启用属性

`lv_img_create()`

创建图像控件

`lv_obj_t * lv_img_create(lv_obj_t * par, lv_obj_t * copy)`

参数	描述
par	指向控件的指针，它将是新图像的父控件
copy	指向图像对象的指针，如果不是 <code>NULL</code> ，则将从该控件复制新控件

返回值：指向创建图像的指针

`lv_img_set_src()`

设置像素图以显示图像

`void lv_img_set_src(lv_obj_t * img, const void * src_img)`

参数	描述
img	指向图像控件的指针
src_img	图像数据

返回值：无

`lv_img_set_auto_size()`

启用自动调整大小功能

如果启用，则对象大小将与图片大小相同

`void lv_img_set_auto_size(lv_obj_t * img, bool autosize_en)`

参数	描述
img	指向图像控件的指针
src_img	true: 自动大小启用, false: 自动大小禁用

返回值: 无

lv_img_set_style()

设置图像样式

static inline void lv_img_set_style(lv_obj_t *img, lv_style_t *style)

参数	描述
img	指向图像控件的指针
style	样式指针

返回值: 无

lv_img_get_src_type()

获取图像源的类型

lv_img_src_t lv_img_get_src_type(const void * src)

参数	描述
src	指向图像源的指针: - 指向'lv_img_t'变量的指针 (内部存储的图像并编译到代码中) - 文件的路径 (例如 "S:/folder/image.bin") - 或符号 (例如 SYMBOL_CLOSE)

返回值: 图像源的类型 **LV_IMG_SRC_VARIABLE** / **FILE** / **SYMBOL** / **UNKNOWN**

lv_img_get_file_name()

获取图像文件集的名称

const char * lv_img_get_file_name(lv_obj_t * img)

参数	描述
img	指向图像控件的指针

返回值: 文件名

lv_img_get_auto_size()

获取自动大小启用属性

bool lv_img_get_auto_size(lv_obj_t * img)

参数	描述
img	指向图像控件的指针

返回值: **true:** 启用自动大小, **false:** 禁用自动大小

lv_img_get_style()

获取图像样式

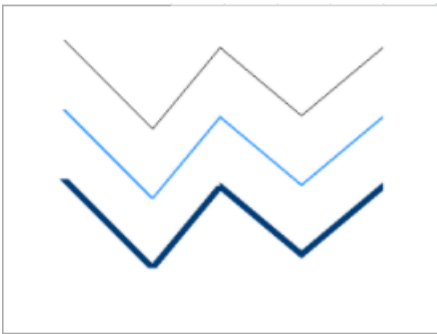
static inline lv_style_t* lv_img_get_style(lv_obj_t *img)

参数	描述
img	指向图像控件的指针

返回值：样式指针

2.3.2 控件使用

2.4 Line



继承：

- 继承自 **Base** 对象，所以每个 `lv_obj_...` 函数都可以用来设置位置，大小，样式和其他基本属性

概述：

- 线对象能够在一组点之间绘制直线。这些点必须存储在 `lv_point_t` 数组中，并通过 `lv_line_set_points(lines, point_array, point_num)` 函数传递给对象
- 可以根据线点自动设置线对象的大小。 您可以使用 `lv_line_set_auto_size(line, true)` 函数启用它。 如果启用，那么当设置点时，对象的宽度和高度将根据最大值进行更改。 `x` 和 `max. y` 点之间的坐标。 默认情况下启用自动大小
- 基本上 `y == 0` 点位于对象的顶部，但您可以使用 `lv_line_set_y_invert(line, true)` 反转 `y` 坐标。 之后，`y` 坐标将从对象的高度中减去

2.4.1 相关宏定义

2.4.2 API 函数

函数	描述
<code>lv_line_create ()</code>	创建线控件
<code>lv_line_set_points()</code>	设置一个点数组

lv_line_set_auto_size()	启用（或禁用）自动调整大小选项
lv_line_set_y_invert()	启用（或禁用）y 坐标反转
lv_line_get_auto_size()	获取自动大小属性
lv_line_get_y_inv()	获取 y 反转属性

lv_line_create()

创建线控件

lv_obj_t * lv_line_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新线的父控件
copy	指向线对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向创建线的指针

lv_line_set_points()

设置一个点数组。 线对象将连接这些点

void lv_line_set_points(lv_obj_t * line, const lv_point_t * point_a, uint16_t point_num)

参数	描述
line	指向线控件的线指针
point_a	一系列的点。 只保存地址，因此数组不能是将被销毁的局部变量
point_num	'point_a'中点的数量

返回值：无

lv_line_set_auto_size()

启用（或禁用）自动调整大小选项。对象的大小适合其点（将宽度设置为 x max，将高度设置为 y max）

void lv_line_set_auto_size(lv_obj_t * line, bool autosize_en)

参数	描述
line	指向线控件的线指针
autosize_en	true: 启用自动大小, false: 禁用自动大小

返回值：无

lv_line_set_y_invert()

启用（或禁用）y 坐标反转

如果启用，则将从对象的高度中减去 y，因此 y = 0 坐标将位于底部

void lv_line_set_y_invert(lv_obj_t * line, bool yinv_en)

参数	描述
line	指向线控件的线指针

autosize_en	true: 启用 y 反转, false: 禁用 y 反转
--------------------	---

返回值: 无

lv_line_set_style()

设置线样式

static inline void lv_line_set_style(lv_obj_t *line, lv_style_t *style)

参数	描述
line	指向线控件的线指针
style	样式指针

返回值: 无

static inline void lv_line_set_upscale(lv_obj_t * line, bool upscale)

lv_line_get_auto_size()

获取自动大小属性

bool lv_line_get_auto_size(lv_obj_t * line)

参数	描述
line	指向线控件的线指针

返回值: **true:** 启用自动大小, **false:** 禁用

lv_line_get_y_inv()

获取 y 反转属性

bool lv_line_get_y_inv(lv_obj_t * line)

参数	描述
line	指向线控件的线指针

返回值: **true:** 启用 y 反转, **false:** 禁用

lv_line_get_style()

获取线样式

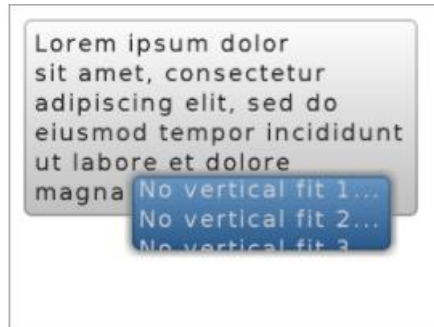
static inline lv_style_t* lv_line_get_style(lv_obj_t *line)

参数	描述
line	指向线控件的线指针

返回值: 样式指针

2.4.3 控件使用

2.5 Container



- 容器是类似矩形的对象，具有一些特殊功能
- 您可以在容器上应用布局以自动订购其子项。 布局间距来自 `style.body.padding.hor / ver / inner` 属性。 可能的布局选项：
 - `LV_CONT_LAYOUT_OFF`: 不对齐
 - `LV_CONT_LAYOUT_CENTER`: 将子项与列中心对齐，并在它们之间保持 `pad.inner` 空间
 - `LV_CONT_LAYOUT_COL_L`: 在左对齐列中对齐子项。 保持左侧的 `pad.hor` 空间，顶部的 `pad.ver` 空间和子项之间的 `pad.inner` 空间。
 - `LV_CONT_LAYOUT_COL_M`: 将中心列中的子项对齐。 将 `pad.ver` 空间保留在顶部，将 `pad.inner` 空间保留在子项之间
 - `LV_CONT_LAYOUT_COL_R`: 在正确对齐的列中对齐子项。 保持右侧的 `pad.hor` 空间，顶部的 `pad.ver` 空间和子项之间的 `pad.inner` 空间
 - `LV_CONT_LAYOUT_ROW_T`: 将子项对齐在一个顶部对齐的行中。 保持左侧的 `pad.hor` 空间，顶部的 `pad.ver` 空间和子项之间的 `pad.inner` 空间
 - `LV_CONT_LAYOUT_ROW_M`: 将子项对齐居中的行。 保持左侧的 `pad.hor` 空间和子项之间的 `pad.inner` 空间
 - `LV_CONT_LAYOUT_ROW_B`: 将子项对齐在底部对齐的行中。 保持左侧的 `pad.hor` 空间，底部的 `pad.ver` 空间和子项之间的 `pad.inner` 空间
 - `LV_CONT_LAYOUT_PRETTY`: 尽可能将对象放在一行中（至少有 `pad.inner` 空间和 `pad.hor` 空间）并开始一个新行。 在子项之间平均划分每条线的空间。 保持 `pad.ver` 空间在顶部和 `pad.inner` 空间在行之间
 - `LV_CONT_LAYOUT_GRID`: 类似于 `PRETTY LAYOUT` 但不能平均分割水平空间，只需让 `pad.hor` 空间

- 您可以启用自动调整功能，该功能会自动设置容器大小以包括所有子项。它会在左侧和右侧保持 `pad.hor` 空间，在顶部和底部保持 `pad.ver` 空间。可以使用 `lv_cont_set_fit(cont, true, true)` 函数水平，垂直或双向启用自动调整。第二个参数是水平，第三个参数是垂直拟合启用

2.5.1 相关宏定义

滚动条模式(lv_layout_t)	
LV_LAYOUT_OFF	
LV_LAYOUT_CENTER	
LV_LAYOUT_COL_L	列左对齐
LV_LAYOUT_COL_M	列中心对齐
LV_LAYOUT_COL_R	列右对齐
LV_LAYOUT_ROW_T	行顶部对齐
LV_LAYOUT_ROW_M	行中心对齐
LV_LAYOUT_ROW_B	行底部对齐
LV_LAYOUT_PRETTY	在行中放置尽可能多的对象并开始新行
LV_LAYOUT_GRID	将相同大小的对象对齐到网格中

2.5.2 API 函数

函数	描述
lv_cont_create()	创建容器对象
lv_cont_set_layout()	在容器上设置布局
lv_cont_set_fit()	启用水平或垂直拟合
lv_cont_get_layout()	获取容器的布局
lv_cont_get_hor_fit()	获取容器的水平拟合启用属性
lv_cont_get_ver_fit()	获取容器的垂直适合启用属性

`lv_cont_create()`

创建容器对象

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

参数	描述
par	指向控件的指针，它将是新容器的父控件
copy	指向容器对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向创建容器的指针

lv_cont_set_layout()

在容器上设置布局

void lv_cont_set_layout(lv_obj_t * cont, lv_layout_t layout)

参数	描述
cont	指向容器对象的指针
layout	来自 lv_cont_layout_t 的布局

返回值：无

lv_cont_set_fit()

启用水平或垂直拟合

容器大小将设置为水平或垂直与子项

void lv_cont_set_fit(lv_obj_t * cont, bool hor_en, bool ver_en)

参数	描述
cont	指向容器对象的指针
hor_en	true: 启用水平拟合
ver_en	true: 启用垂直拟合

返回值：无

lv_cont_set_style()

设置容器样式

static inline void lv_cont_set_style(lv_obj_t * cont, lv_style_t * style)

参数	描述
cont	指向容器对象的指针
style	样式指针

返回值：无

lv_cont_get_layout()

获取容器的布局

lv_layout_t lv_cont_get_layout(lv_obj_t * cont)

参数	描述
cont	指向容器对象的指针

返回值：来自 [lv_cont_layout_t](#) 的布局

lv_cont_get_hor_fit()

获取容器的水平拟合启用属性

bool lv_cont_get_hor_fit(lv_obj_t * cont)

参数	描述
cont	指向容器对象的指针

返回值：true: 启用水平拟合; false: 禁用

lv_cont_get_ver_fit()

获取容器的垂直适合启用属性

bool lv_cont_get_ver_fit(lv_obj_t * cont)

参数	描述
----	----

cont

指向容器对象的指针

返回值: **true**: 启用垂直拟合; **false**: 禁用

lv_cont_get_style()

获取容器样式

static inline lv_style_t * lv_cont_get_style(lv_obj_t *cont)

返回值: 样式指针

2.5.3 控件使用

2.6 Page



- 页面由两个彼此组成的容器组成: 底部是背景 (或底部), 顶部是可滚动的。如果您在页面上创建子项, 它将自动移动到可滚动容器。如果可滚动容器变大, 则可以通过拖动滚动背景 (如智能手机上的列表)
- 默认情况下, 可垂直启用可滚动的自动调整属性, 因此其高度将增加以包括其所有子项。可滚动的宽度自动调整为背景宽度 (减去背景的水平填充)
- 后台对象可以作为页面本身引用, 如: **lv_obj_set_width(page, 100)**
- 可滚动对象可以使用: **lv_page_get_scrl** (页面)
- 可以根据四个策略显示滚动条:
 - **LV_SB_MODE_OFF**: 永远不显示滚动条
 - **LV_SB_MODE_ON**: 始终显示滚动条
 - **LV_SB_MODE_DRAG**: 拖动页面时显示滚动条
 - **LV_SB_MODE_AUTO**: 当可滚动容器足够大以便滚动时显示滚动条
- 您可以通过以下方式设置滚动条显示策略: **lv_page_set_sb_mode(page, SB_MODE)** 默认值为 **LV_PAGE_SB_MODE_ON**
- 您可以将子项粘贴到页面上。在这种情况下, 您可以通过拖动子对象来滚动页面。它可以通过 **lv_page_glue_obj(child, true)** 启用
- 您可以使用以下命令聚焦到页面上的对象: **lv_page_focus(page, child, anim_time)**。它将移动可滚动容器以显示子项。如果最后一个参数不为零, 则页面将随动画移动

- 可以使用 `lv_page_set_rel_action(page, my_rel_action)` 和 `lv_page_set_pr_action(page, my_pr_action)` 为页面分配发布和按下操作。该操作也可以从背景和可滚动性对象触发
- 有些函数可以直接设置/获取可滚动的属性：
 - `lv_page_set_scrl_fit()`
 - `lv_page_set_scrl_width()`
 - `lv_page_set_scrl_height()`
 - `lv_page_set_scrl_layout()`

2.6.1 相关宏定义

滚动条模式(lv_sb_mode_t)	
LV_SB_MODE_OFF	不显示滚动条
LV_SB_MODE_ON	一直显示滚动条
LV_SB_MODE_DRAG	拖动页面是显示滚动条
LV_SB_MODE_AUTO	容器过大时显示滚动条

样式属性	
LV_PAGE_STYLE_BG	背景样式
LV_PAGE_STYLE_SCRL	滚动样式
LV_PAGE_STYLE_SB	滚动条样式

2.6.2 API 函数

函数	描述
<code>lv_page_create()</code>	创建页面控件
<code>lv_page_set_rel_action()</code>	设置页面的释放事件
<code>lv_page_set_pr_action()</code>	设置页面的按下动作
<code>lv_page_set_sb_mode()</code>	在页面上设置滚动条模式
<code>lv_page_set_style()</code>	设置页面样式
<code>lv_page_get_scrl()</code>	获取页面的可滚动对象
<code>lv_page_get_sb_mode()</code>	在页面上设置滚动条模式
<code>lv_page_get_style()</code>	获得页面样式
<code>lv_page_glue_obj()</code>	将对象粘到页面上
<code>lv_page_focus()</code>	聚焦控件，它确保控件在页面上可见

`lv_page_create()`

创建页面控件

lv_obj_t * lv_page_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新页的父控件
copy	指向页面对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向创建页面的指针

lv_page_set_rel_action()

设置页面的释放事件

void lv_page_set_rel_action(lv_obj_t * page, lv_action_t rel_action)

参数	描述
page	页面控件指针
rel_action	当页面被释放时调用的函数

返回值：无

lv_page_set_pr_action()

设置页面的按下动作

void lv_page_set_pr_action(lv_obj_t * page, lv_action_t pr_action)

参数	描述
page	页面控件指针
pr_action	当页面被按下时调用的函数

返回值：无

lv_page_set_sb_mode()

在页面上设置滚动条模式

void lv_page_set_sb_mode(lv_obj_t * page, lv_sb_mode_t sb_mode)

参数	描述
page	页面控件指针
sb_mode	来自 lv_sb_mode_t 枚举里面的新模式

返回值：无

lv_page_set_scr_fit()

设置页面可滚动部分的自适应属性

这意味着它可以自动设置其大小以涉及所有子项

（可以水平和垂直单独设置）

static inline void lv_page_set_scr_fit(lv_obj_t * page, bool hor_en, bool ver_en)

参数	描述
page	页面控件指针
hor_en	使能水平方向自适应
ver_en	使能垂直方向自适应

返回值：无

lv_page_set_scrl_width()

设置页面可滚动部分的宽度

static inline void lv_page_set_scrl_width(lv_obj_t *page, lv_coord_t w)

参数	描述
page	页面控件指针
w	可滚动的新宽度（水平适应开启时无效）

返回值：无

lv_page_set_scrl_height()

设置页面可滚动部分的高度

static inline void lv_page_set_scrl_height(lv_obj_t *page, lv_coord_t h)

参数	描述
page	页面控件指针
h	可滚动的新高度（垂直适应开启时无效）

返回值：无

lv_page_set_scrl_layout()

设置页面可滚动部分的布局

static inline void lv_page_set_scrl_layout(lv_obj_t *page, lv_layout_t layout)

参数	描述
page	页面控件指针
layout	见 lv_layout_t

返回值：无

lv_page_set_style()

设置页面样式

void lv_page_set_style(lv_obj_t *page, lv_page_style_t type, lv_style_t *style)

参数	描述
page	页面控件指针
type	要设置的风格类型
style	风格指针

返回值：无

lv_page_get_scrl()

获取页面的可滚动对象

lv_obj_t * lv_page_get_scrl(lv_obj_t *page)

参数	描述
page	页面控件指针

返回值：指向容器的指针，它是页面的可滚动部分

lv_page_get_sb_mode()

在页面上设置滚动条模式

`lv_sb_mode_t lv_page_get_sb_mode(lv_obj_t * page)`

参数	描述
page	页面控件指针

返回值: `lv_sb_mode_t` 枚举里面的模式

`lv_page_get_scl_width()`

获取页面的可滚动部分的宽度

`static inline lv_coord_t lv_page_get_scl_width(lv_obj_t *page)`

参数	描述
page	页面控件指针

返回值: 可滚动的宽度

`lv_page_get_scl_height()`

获取页面的可滚动部分的高度

`static inline lv_coord_t lv_page_get_scl_height(lv_obj_t *page)`

参数	描述
page	页面控件指针

返回值: 可滚动的高度

`lv_page_get_scl_layout()`

获取页面可滚动部分的布局

`static inline lv_layout_t lv_page_get_scl_layout(lv_obj_t * page)`

参数	描述
page	页面控件指针

返回值: 见 `lv_layout_t`

`lv_page_get_scl_hor_fit()`

获取页面可滚动部分的水平自适应属性

`static inline bool lv_page_get_scl_hor_fit(lv_obj_t * page)`

参数	描述
page	页面控件指针

返回值: **true**: 启用水平自适应; **false**: 禁用

`lv_page_get_scl_fit_ver()`

获取页面可滚动部分的垂直自适应属性

`static inline bool lv_page_get_scl_fit_ver(lv_obj_t * page)`

参数	描述
page	页面控件指针

返回值: **true**: 启用垂直自适应; **false**: 禁用

`lv_page_get_style()`

获得页面样式

`lv_style_t * lv_page_get_style(lv_obj_t *page, lv_page_style_t type)`

参数	描述
page	页面控件指针
type	要获取的样式类型

返回值：样式指针

lv_page_glue_obj()

将对象粘到页面上。之后，页面也可以用这个对象移动（拖动）

void lv_page_glue_obj(lv_obj_t * obj, bool glue)

参数	描述
obj	页面上的控件指针
glue	true :使能粘贴， false :不使能粘贴

返回值：无

lv_page_focus()

聚焦控件，它确保控件在页面上可见

void lv_page_focus(lv_obj_t * page, lv_obj_t * obj, uint16_t anim_time)

参数	描述
page	页面控件指针
obj	指向要聚焦的对象的指针(必须在页面上)
anim_time	以毫秒为单位滚动动画时间(0: 没有动画)

返回值：无

2.6.3 控件使用

```

1.  /*创建一个简单的界面*/
2.  lv_obj_t * page1 = lv_page_create(lv_scr_act(), NULL);
3.
4.  /*重新调整界面*/
5.  lv_obj_t * page2 = lv_page_create(lv_scr_act(), NULL);
6.  lv_obj_set_size(page2, LV_DPI, LV_DPI * 2);
7.  lv_obj_align(page2, page1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20);
8.
9.  /*添加一些可以滚动的文本*/
10. lv_obj_t * page3 = lv_page_create(lv_scr_act(), page2);
11. lv_obj_set_size(page3, LV_DPI, LV_DPI * 2);
12. lv_obj_align(page3, page2, LV_ALIGN_OUT_RIGHT_TOP, 20, 0);
13.
14. lv_obj_t * label = lv_label_create(page3, NULL);
15. lv_label_set_text(label, "First line of a text\n"
16.                          "Second line of a text\n"
17.                          "Third line of a text\n"

```

```

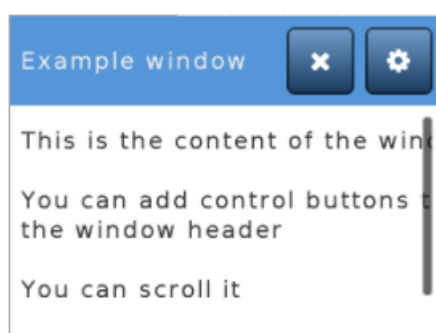
18.         "Forth line of a text\n"
19.         "Fifth line of a text\n"
20.         "Sixth line of a text\n"
21.         "Seventh line of a text\n"
22.         "Eight line of a text\n"
23.         "Ninth line of a text\n"
24.         "Tenth line of a text\n");
25.
26. /*设置水平和垂直两个方向可以滚动*/
27. lv_obj_t * page4 = lv_page_create(lv_scr_act(), page3);
28. lv_obj_align(page4, page3, LV_ALIGN_OUT_RIGHT_TOP, 20, 0);
29. lv_page_set_scrl_fit(page4, true, true);
30. label = lv_label_create(page4, label);

```

运行结果：



2.7 Window



- 窗口是最复杂的容器类对象之一。它们由两个主要部分构成：顶部的标题容器和标题下面的内容的页面。
- 在标题上有标题，可以通过以下方式修改：`lv_win_set_title(win, "New title")`。标题始终继承标题的样式。
- 您可以使用以下命令在标题的右侧添加控制按钮：`lv_win_add_btn(win, "U:/close", my_close_action)`。第二个参数是图像文件路径，第三个参数是释放按钮时要调用的函数。您可以将符号用作图像，如：`lv_win_add_btn(win, SYMBOL_CLOSE, my_close_action)`

- 您可以使用 `lv_win_set_btn_size(win, new_size)` 函数修改控件按钮的大小。
- 滚动条行为可以通过 `lv_win_set_sb_mode(win, LV_SB_MODE_...)` 设置。
- 要设置内容的布局，请使用 `lv_win_set_layout(win, LV_LAYOUT_...)`

2.7.1 相关宏定义

样式属性(lv_win_style_t)	
LV_WIN_STYLE_BG	主背景样式，它使用所有 style.body 属性(default: lv_style_plain)
LV_WIN_STYLE_CONTENT_BG	内容页面的背景，它使用所有 style.body 属性(default: lv_style_transp)
LV_WIN_STYLE_CONTENT_SCRL	内容页面的可滚动部分，它使用所有 style.body 属性(default: lv_style_transp)
LV_WIN_STYLE_SB	滚动条的样式，它使用所有 style.body 属性。 hor / ver * padding 分别设置滚动条的填充，内部填充设置滚动条的宽度(default: lv_style_pretty_color)
LV_WIN_STYLE_HEADER	header 的样式，它使用所有 style.body 属性(default: lv_style_plain_color)
LV_WIN_STYLE_BTN_REL	释放按钮的样式(在标题上)，它使用所有 style.body 属性(default: lv_style_btn_rel)
LV_WIN_STYLE_BTN_PR	释放按钮的样式(在标题上)，它使用所有 style.body 属性(default: lv_style_btn_pr)

2.7.2 API 函数

函数	描述
lv_win_create ()	创建 window 控件
lv_win_add_btn()	将控制按钮添加到窗口的标题
lv_win_close_action()	可以分配给窗口控制按钮以关闭它的释放操作
lv_win_set_title()	设置窗口的标题
lv_win_set_btn_size()	设置窗口的控制按钮大小
lv_win_set_layout()	设置窗口的布局
lv_win_set_sb_mode()	设置窗口的滚动条模式
lv_win_set_style()	设置窗口的样式
lv_win_get_title()	获取窗口的标题
lv_win_get_btn_size()	获取窗口的控件按钮大小

lv_win_get_layout()	获取窗口的布局
lv_win_get_sb_mode()	获取窗口的滚动条模式
lv_win_get_width()	获取窗口的内容区域（页面可滚动）的宽度
lv_win_get_from_btn()	从其中一个控制按钮获取窗口的指针
v_win_get_style()	获得窗口的风格
lv_win_focus()	聚焦一个对象，它确保对象在窗口中可见

lv_win_create ()

创建 window 控件

lv_obj_t * lv_win_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新窗口的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向创建的窗口的指针

lv_win_add_btn()

将控制按钮添加到窗口的标题

lv_obj_t * lv_win_add_btn(lv_obj_t * win, const void * img_src, lv_action_t rel_action)

参数	描述
win	窗口控件指针
img_src	一个图像源
rel_action	释放按钮时调用的函数指针

返回值：指向创建的按钮对象的指针

lv_win_close_action()

可以分配给窗口控制按钮以关闭它的释放操作

lv_res_t lv_win_close_action(lv_obj_t * btn)

参数	描述
btn	指向已释放按钮的指针

返回值：始终是 LV_ACTION_RES_INV，因为该按钮随窗口一起被删除

lv_win_set_title()

设置窗口的标题

void lv_win_set_title(lv_obj_t * win, const char * title)

参数	描述
win	窗口控件指针
title	新标题的字符串

返回值：无

lv_win_set_btn_size()

设置窗口的控制按钮大小

void lv_win_set_btn_size(lv_obj_t * win, lv_coord_t size)

参数	描述
win	窗口控件指针
size	控制按钮大小

返回值：无

lv_win_set_layout()

设置窗口的布局

void lv_win_set_layout(lv_obj_t *win, lv_layout_t layout)

参数	描述
win	窗口控件指针
layout	从 lv_layout_t 布局

返回值：无

lv_win_set_sb_mode()

设置窗口的滚动条模式

void lv_win_set_sb_mode(lv_obj_t *win, lv_sb_mode_t sb_mode)

参数	描述
win	窗口控件指针
sb_mode	来自 lv_sb_mode_t 的新滚动条模式

返回值：无

lv_win_set_style()

设置窗口的样式

void lv_win_set_style(lv_obj_t *win, lv_win_style_t type, lv_style_t *style)

参数	描述
win	窗口控件指针
type	要设置的样式类型
style	样式指针

返回值：无

lv_win_get_title()

获取窗口的标题

const char * lv_win_get_title(lv_obj_t * win)

参数	描述
win	窗口控件指针

返回值：窗口的标题字符串

lv_win_get_btn_size()

获取窗口的控件按钮大小

lv_coord_t lv_win_get_btn_size(lv_obj_t * win)

参数	描述
win	窗口控件指针

返回值：控制按钮大小

lv_win_get_layout()

获取窗口的布局

lv_layout_t lv_win_get_layout(**lv_obj_t** *win)

参数	描述
win	窗口控件指针

返回值：窗口的布局(来自 **lv_layout_t**)

lv_win_get_sb_mode()

获取窗口的滚动条模式

lv_sb_mode_t lv_win_get_sb_mode(**lv_obj_t** *win)

参数	描述
win	窗口控件指针

返回值：窗口的滚动条模式(来自 **lv_sb_mode_t**)

lv_win_get_width()

获取窗口的内容区域（页面可滚动）的宽度

lv_coord_t lv_win_get_width(**lv_obj_t** * win)

参数	描述
win	窗口控件指针

返回值：content_bg 区域的宽度

lv_win_get_from_btn()

从其中一个控制按钮获取窗口的指针(它在控制按钮的操作中非常有用，其中只有按钮是已知的)

lv_obj_t * lv_win_get_from_btn(**lv_obj_t** * ctrl_btn)

参数	描述
ctrl_btn	窗口控制按钮的指针

返回值：指向 ctrl_btn 窗口的指针

lv_win_get_style()

获得窗口的风格

lv_style_t * lv_win_get_style(**lv_obj_t** *win, **lv_win_style_t** type)

参数	描述
win	窗口控件指针
type	样式类型

返回值：样式的指针

lv_win_focus()

聚焦一个对象，它确保对象在窗口中可见

void lv_win_focus(**lv_obj_t** * win, **lv_obj_t** * obj, **uint16_t** anim_time)

参数	描述
win	窗口控件指针
obj	指向要聚焦的对象的指针（必须在窗口中）
anim_time	以毫秒为单位滚动动画时间（0：无动画）

返回值：无

2.7.3 控件使用

```

1. lv_obj_t *win1 = lv_win_create(lv_scr_act(), NULL);
2. lv_obj_set_size(win1, LV_HOR_RES / 2 - LV_DPI / 20, LV_VER_RES / 2 - LV_DPI / 20);
3.
4. lv_obj_t *win2 = lv_win_create(lv_scr_act(), win1);
5. lv_obj_align(win2, NULL, LV_ALIGN_IN_TOP_RIGHT, 0, 0);
6. lv_win_set_title(win2, "Random title");
7. lv_win_add_btn(win2, SYMBOL_CLOSE, NULL);
8. lv_win_add_btn(win2, SYMBOL_OK, NULL);
9. lv_win_add_btn(win2, SYMBOL_EDIT, NULL);
10.
11. lv_obj_t *label = lv_label_create(win2, NULL);
12. lv_obj_set_pos(label, 10, 10);
13. lv_label_set_text(label, "Long\n\nntext\n\nnto\n\nnsee\n\nnthe\n\nnscrollbars");
14.
15.
16. static lv_style_t header;
17. lv_style_copy(&header, &lv_style_plain);
18. header.body.main_color = LV_COLOR_RED;
19. header.body.grad_color = LV_COLOR_MARRON;
20. header.body.padding.inner = 0;
21. header.text.color = LV_COLOR_WHITE;
22.
23. lv_obj_t *win3 = lv_win_create(lv_scr_act(), win2);
24. lv_obj_align(win3, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 0, 0);
25. lv_win_set_style(win3, LV_WIN_STYLE_HEADER, &header);
26. lv_win_set_style(win3, LV_WIN_STYLE_BTN_REL, &lv_style_transp);
27. lv_win_set_style(win3, LV_WIN_STYLE_CONTENT_BG, &lv_style_plain_color);
28. lv_win_set_style(win3, LV_WIN_STYLE_CONTENT_SCRL, &lv_style_plain);
29. lv_win_set_style(win3, LV_WIN_STYLE_BG, &lv_style_plain_color);
30. lv_win_set_btn_size(win3, LV_DPI / 3);
31.

```

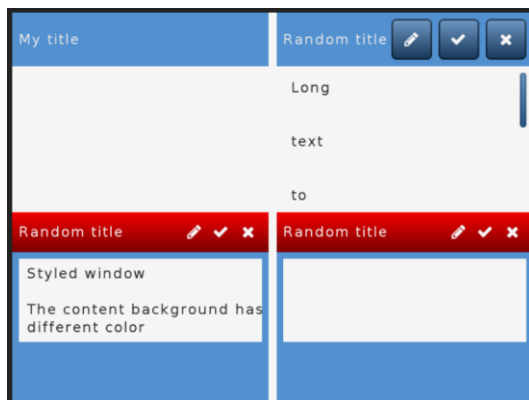


```

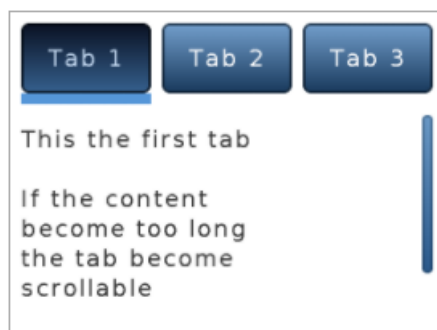
32. label = lv_label_create(win3, NULL);
33. lv_obj_set_pos(label, 10, 10);
34. lv_label_set_text(label, "Styled window\n\nThe content background has\ndifferent color");
35.
36. lv_obj_t *win4 = lv_win_create(lv_scr_act(), win3);
37. lv_obj_align(win4, NULL, LV_ALIGN_IN_BOTTOM_RIGHT, 0, 0);

```

运行结果:



2.8 Tab View



- 选项卡视图对象可用于组织选项卡中的内容。 你可以使用 `lv_tabview_add_tab(tabview, "Tab name")` 添加新选项卡。 它将返回一个指向 `Page` 对象的指针，你可以在其中添加选项卡的内容。
- 要选择标签，您可以：
 - 在标题部分单击它
 - 水平滑动
 - 使用 `lv_tabview_set_tab_act(tabview, id, anim_en)` 函数
- 可以使用 `lv_tabview_set_sliding(tabview, false)` 禁用手动滑动
- 动画时间由 `lv_tabview_set_anim_time(tabview, anim_time)` 调整
- 可以使用 `lv_tabview_set_tab_load_action(tabview, action)` 将回调函数分配给选项卡加载事件。 回调函数需要具有以下原型： `void callback(lv_obj_t * tabview, uint16_t act_id)` 其中 `act_id` 表示将加载

的选项卡。 在操作中, `lv_tabview_get_tab_act(tabview)`将给出旧选项卡的 id

2.8.1 相关宏定义

样式属性(<code>lv_tabview_style_t</code>)	
LV_TABVIEW_STYLE_BG	使用所有 <code>style.body</code> 属性的主要背景 (default: <code>lv_style_plain</code>)
LV_TABVIEW_STYLE_INDIC	顶部有一个细长矩形表示当前标签。使用所有 <code>style.body</code> 属性。 它的高度来自 <code>body.padding.inner</code> (default: <code>lv_style_plain_color</code>)
LV_TABVIEW_STYLE_BTN_BG	标签按钮背景的样式。 使用所有 <code>style.body</code> 属性。 考虑到 <code>body.padding.ver</code> , 将自动设置标题高度(default: <code>lv_style_transp</code>)
LV_TABVIEW_STYLE_BTN_REL	已发布的标签按钮的样式。 使用所有 <code>style.body</code> 属性(default: <code>lv_style_tbn_rel</code>)
LV_TABVIEW_STYLE_BTN_PR	已发布的标签按钮的样式。 使用所有 <code>style.body</code> 属性(default: <code>lv_style_tbn_rel</code>)
LV_TABVIEW_STYLE_BTN_TGL_REL	切换释放的标签按钮的样式。 使用所有 <code>style.body</code> 属性(default: <code>lv_style_tbn_rel</code>)
LV_TABVIEW_STYLE_BTN_TGL_PR	切换按下的标签按钮的样式。 使用所有 <code>style.body</code> 属性(default: <code>lv_style_btn_tgl_pr</code>)

2.8.2 API 函数

函数	描述
lv_tabview_create ()	创建选项卡视图控件
lv_tabview_add_tab()	添加具有给定名称的新选项卡
lv_tabview_set_tab_load_action()	设置加载选项卡时要调用的操作（仅在需要时才能创建内容）
lv_tabview_set_sliding()	使用触摸板启用水平滑动
lv_tabview_set_anim_time()	加载新选项卡时, 设置选项卡视图的动画时间
lv_tabview_set_style()	设置选项卡视图的样式
lv_tabview_get_tab_act()	获取当前活动选项卡的索引
lv_tabview_get_tab_count()	获取选项卡数量

lv_tabview_get_tab()	获取选项卡的页面（内容区域）
lv_tabview_get_tab_load_action()	获取选项卡加载操作
lv_tabview_get_sliding()	是否启用水平滑动
lv_tabview_get_anim_time()	加载新选项卡时，获取选项卡视图的动画时间
lv_tabview_get_style()	获取选项卡视图的样式

lv_tabview_create()

创建选项卡视图控件

lv_obj_t * lv_tabview_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新窗口的父控件
copy	指向选项卡对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：指向创建的选项卡的指针

lv_tabview_add_tab()

添加具有给定名称的新选项卡

lv_obj_t * lv_tabview_add_tab(lv_obj_t * tabview, const char * name)

参数	描述
tabview	选项卡视图指针
name	选项卡按钮上的文本

返回值：指向创建的页面对象（**lv_page**）的指针。 您可以在此处创建内容

lv_tabview_set_tab_act()

设置新选项卡

void lv_tabview_set_tab_act(lv_obj_t * tabview, uint16_t id, bool anim_en)

参数	描述
tabview	选项卡视图指针
id	要加载的选项卡的索引
anim_en	true : 设置滑动动画; false : 立即设置

返回值：无

lv_tabview_set_tab_load_action()

设置加载选项卡时要调用的操作（仅在需要时才能创建内容）

[lv_tabview_get_act\(\)](#) 仍然提供当前（旧）选项卡（从此处删除内容）

void lv_tabview_set_tab_load_action(lv_obj_t * tabview, lv_tabview_action_t action)

参数	描述
tabview	选项卡视图指针
action	指向加载 btn 时要调用的函数的指针

返回值：无

lv_tabview_set_sliding()

使用触摸板启用水平滑动

void lv_tabview_set_sliding(lv_obj_t * tabview, bool en)

参数	描述
tabview	选项卡视图指针
en	true: 启用滑动; false: 禁用滑动

返回值: 无

lv_tabview_set_anim_time()

加载新选项卡时, 设置选项卡视图的动画时间

void lv_tabview_set_anim_time(lv_obj_t * tabview, uint16_t anim_time)

参数	描述
tabview	选项卡视图指针
anim_time	动画时间, 以毫秒为单位

返回值: 无

lv_tabview_set_style()

设置选项卡视图的样式

void lv_tabview_set_style(lv_obj_t * tabview, lv_tabview_style_t type, lv_style_t * style)

参数	描述
tabview	选项卡视图指针
type	要设置的样式类型
style	指向新样式的指针

返回值: 无

lv_tabview_get_tab_act()

获取当前活动选项卡的索引

uint16_t lv_tabview_get_tab_act(lv_obj_t * tabview)

参数	描述
tabview	选项卡视图指针

返回值: 活跃的 btn 索引

lv_tabview_get_tab_count()

获取选项卡数量

uint16_t lv_tabview_get_tab_count(lv_obj_t * tabview)

参数	描述
tabview	选项卡视图指针

返回值: btn 数量

lv_tabview_get_tab()

获取选项卡的页面 (内容区域)

lv_obj_t * lv_tabview_get_tab(lv_obj_t * tabview, uint16_t id)

参数	描述
----	----

tabview	选项卡视图指针
id	btn 的索引(>= 0)

返回值：指向页面（lv_page）对象的指针

lv_tabview_get_tab_load_action()

获取选项卡加载操作

lv_tabview_action_t lv_tabview_get_tab_load_action(lv_obj_t *tabview)

参数	描述
tabview	选项卡视图指针

返回值：返回当前的 btn 加载动作

lv_tabview_get_sliding()

是否启用水平滑动

bool lv_tabview_get_sliding(lv_obj_t * tabview)

参数	描述
tabview	选项卡视图指针

返回值：true：启用滑动; false：禁用滑动

lv_tabview_get_anim_time()

加载新选项卡时，获取选项卡视图的动画时间

uint16_t lv_tabview_get_anim_time(lv_obj_t * tabview)

参数	描述
tabview	选项卡视图指针

返回值：动画时间，以毫秒为单位

lv_tabview_get_style()

获取选项卡视图的样式

lv_style_t * lv_tabview_get_style(lv_obj_t *tabview, lv_tabview_style_t type)

参数	描述
tabview	选项卡视图指针
type	样式类型

返回值：样式指针的样式

2.8.3 控件使用

```

1. /* Default object. It will be an empty tab view*/
2. lv_obj_t *tv1 = lv_tabview_create(lv_scr_act(), NULL);
3. lv_tabview_add_tab(tv1, "First");
4. lv_tabview_add_tab(tv1, "Second");
5. lv_obj_set_size(tv1, LV_HOR_RES / 2 - 10, LV_VER_RES / 2 - 10);
6.
7. /*Copy the first tabview and add some texts*/
8. lv_obj_t *tv2 = lv_tabview_create(lv_scr_act(), tv1);

```

```

9. lv_obj_align(tv2, NULL, LV_ALIGN_IN_TOP_RIGHT, 0, 0);
10. lv_obj_t *tab = lv_tabview_get_tab(tv2, 0);
11. lv_obj_t *label = lv_label_create(tab, NULL);
12. lv_label_set_text(label, "This is\n\n\nA long text\n\n\ntext\n\n\non the\n\n\nsecond\n\n\ntab\n\n\nto see\n\n\nthe scrolling");
13.
14. tab = lv_tabview_get_tab(tv2, 1);
15. label = lv_label_create(tab, NULL);
16. lv_label_set_text(label, "This is the second tab");
17.
18.
19. /*Create styles*/
20. static lv_style_t bg;
21. static lv_style_t sb;
22. static lv_style_t btns_bg;
23. static lv_style_t tab_bg;
24. static lv_style_t rel;
25. static lv_style_t pr;
26. static lv_style_t tgl_rel;
27. static lv_style_t tgl_pr;
28. static lv_style_t indic;
29.
30. lv_style_copy(&btns_bg, &lv_style_plain_color);
31. lv_style_copy(&tab_bg, &lv_style_pretty_color);
32. lv_style_copy(&bg, &lv_style_pretty_color);
33. lv_style_copy(&sb, &lv_style_pretty);
34. lv_style_copy(&btns_bg, &lv_style_transp_fit);
35. lv_style_copy(&rel, &lv_style_plain);
36. lv_style_copy(&pr, &lv_style_plain);
37. lv_style_copy(&tgl_rel, &lv_style_plain);
38. lv_style_copy(&tgl_pr, &lv_style_plain);
39. lv_style_copy(&indic, &lv_style_plain);
40.
41. rel.body.main_color = LV_COLOR_SILVER;
42. pr.body.main_color = LV_COLOR_GRAY;
43. tgl_rel.body.main_color = LV_COLOR_RED;
44. tgl_pr.body.main_color = LV_COLOR_MARRON;
45. indic.body.main_color = LV_COLOR_ORANGE;
46. indic.body.grad_color = LV_COLOR_ORANGE;
47. indic.body.padding.inner = LV_DPI / 16;
48. tab_bg.body.main_color = LV_COLOR_SILVER;
49. tab_bg.body.grad_color = LV_COLOR_GREEN;
50. tab_bg.text.color = LV_COLOR_YELLOW;
51.

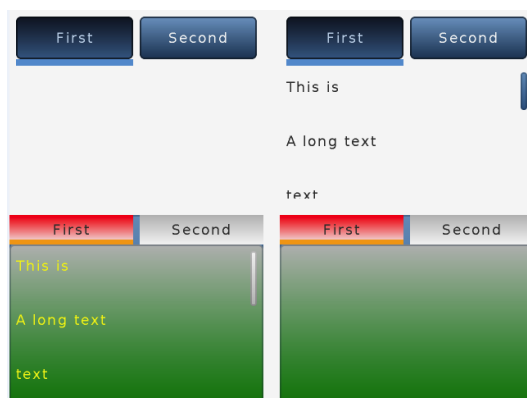
```

```

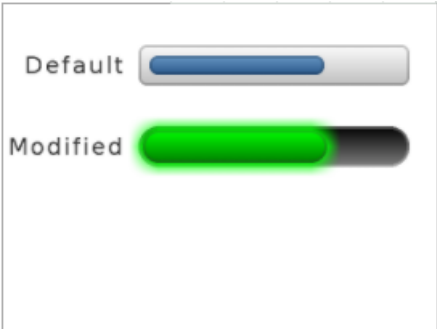
52. /*Apply the styles*/
53. lv_obj_t *tv3 = lv_tabview_create(lv_scr_act(), tv2);
54. lv_obj_align(tv3, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 0, 0);
55. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_BG, &bg);
56. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_BTN_BG, &btns_bg);
57. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_BTN_REL, &rel);
58. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_BTN_PR, &pr);
59. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_BTN_TGL_REL, &tgl_rel);
60. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_BTN_TGL_PR, &tgl_pr);
61. lv_tabview_set_style(tv3, LV_TABVIEW_STYLE_INDIC, &indic);
62.
63. tab = lv_tabview_get_tab(tv3, 0);
64. lv_page_set_style(tab, LV_PAGE_STYLE_BG, &tab_bg);
65. lv_page_set_style(tab, LV_PAGE_STYLE_SB, &sb);
66. label = lv_label_create(tab, NULL);
67. lv_label_set_text(label, "This is\n\nA long text\n\nntext\n\n\non the\n\n\nsecond\n\n\ntab\n\n\nto see\n\n\nthe scrolling");
68.
69. tab = lv_tabview_get_tab(tv3, 1);
70. label = lv_label_create(tab, NULL);
71. lv_label_set_text(label, "This is the second tab");
72.
73. /*Copy the styles tab view*/
74. lv_obj_t *tv4 = lv_tabview_create(lv_scr_act(), tv3);
75. lv_obj_align(tv4, NULL, LV_ALIGN_IN_BOTTOM_RIGHT, 0, 0);

```

运行结果：



2.9 Bar



- 进度条对象有两个主要部分：一个是对象本身的背景，一个与背景相似的形状的指示器，但它的宽度/高度可以调整
- 根据宽度/高度比，杆的方向可以是垂直的或水平的。 在水平条上逻辑上指示器宽度，在垂直条上可以改变指示器高度
- 可以通过以下方式设置新值：`lv_bar_set_value(bar, new_value)`。 该值在范围（最小值和最大值）中解释，可以使用以下值进行修改：`lv_bar_set_range(bar, min, max)`。 默认范围是：1..100。
- 使用从当前值到所需的动画可以设置新值。 在这种情况下，使用 `lv_bar_set_value_anim(bar, new_value, anim_time)`。

2.9.1 相关宏定义

样式属性(lv_bar_style_t)	
LV_BAR_STYLE_BG	它使用了一个 Base 对象的样式元素其默认样式为：LV_STYLE_PRETTY
LV_BAR_STYLE_INDIC	与背景类似。 它的样式可以通过以下方式设置 <code>lv_bar_set_style_indic(bar,&style_indic)</code> 。 它使用 hpad 和 vpad 样式元素来保留背景空间。 其默认样式为：LV_STYLE_PRETTY_COLOR

2.9.2 API 函数

函数	描述
lv_bar_create ()	创建进度条控件
lv_bar_set_value()	设置进度条的值
lv_bar_set_value_anim()	在进度条上设置带动画的新值
lv_bar_set_range()	设置进度条的最小值和最大值

lv_bar_set_style()	设置进度条的样式
lv_bar_get_value()	获取进度条的数值
lv_bar_get_min_value()	获取进度条的最小值
lv_bar_get_max_value()	获取进度条的最大值
lv_bar_get_style()	获取进度条样式

lv_bar_create()

创建进度条控件

lv_obj_t * lv_bar_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新进度条的父控件
copy	指向窗口对象的指针，如果不是NULL，则将从该控件复制新控件

返回值：创建的进度条指针

lv_bar_set_value()

设置进度条的值

void lv_bar_set_value(lv_obj_t * bar, int16_t value)

参数	描述
bar	进度条指针
value	新值

返回值：无

lv_bar_set_value_anim()

在进度条上设置带动画的新值

void lv_bar_set_value_anim(lv_obj_t * bar, int16_t value, uint16_t anim_time)

参数	描述
bar	进度条指针
value	新值
anim_time	动画时间，以毫秒为单位

返回值：无

lv_bar_set_range()

设置进度条的最小值和最大值

void lv_bar_set_range(lv_obj_t * bar, int16_t min, int16_t max)

参数	描述
bar	进度条指针
min	最小值
max	最大值

返回值：无

lv_bar_set_style()

设置进度条的样式

`void lv_bar_set_style(lv_obj_t *bar, lv_bar_style_t type, lv_style_t *style)`

参数	描述
bar	进度条指针
type	要设置的样式类型
style	样式指针

返回值：无

`lv_bar_get_value()`

获取进度条的数值

`int16_t lv_bar_get_value(lv_obj_t * bar)`

参数	描述
bar	进度条指针

返回值：进度条的数值

`lv_bar_get_min_value()`

获取进度条的最小值

`int16_t lv_bar_get_min_value(lv_obj_t * bar)`

参数	描述
bar	进度条指针

返回值：进度条的最小值

`lv_bar_get_max_value()`

获取进度条的最大值

`int16_t lv_bar_get_max_value(lv_obj_t * bar)`

参数	描述
bar	进度条指针

返回值：进度条的最大值

`lv_bar_get_style()`

获取进度条样式

`lv_style_t * lv_bar_get_style(lv_obj_t *bar, lv_bar_style_t type)`

参数	描述
bar	进度条指针
type	要获取的样式类型

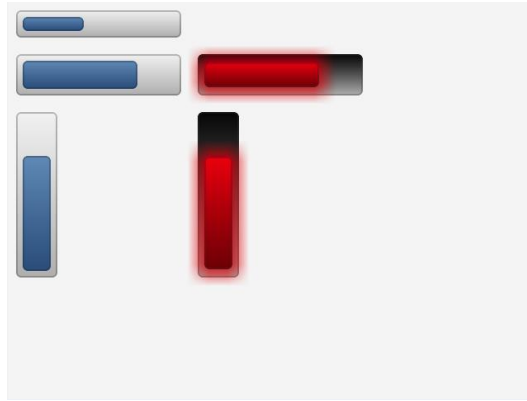
返回值：样式指针

2.9.3 控件使用

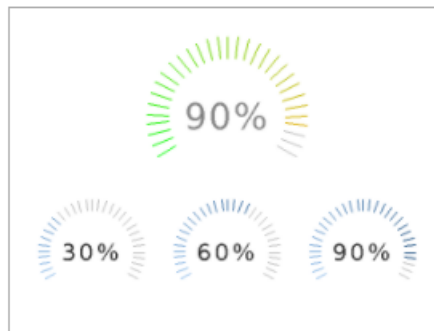
```
1.  /* Create a default object*/
2.  lv_obj_t * bar1 = lv_bar_create(lv_scr_act(), NULL);
3.  lv_obj_set_pos(bar1, 10, 10);
```

```
4. lv_bar_set_value(bar1, 40);
5.
6. /* Modify size and position, range and set to 75 % */
7. lv_obj_t * bar2 = lv_bar_create(lv_scr_act(), NULL);
8. lv_obj_set_size(bar2, 200, 50);
9. lv_obj_align(bar2, bar1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20);
10. lv_bar_set_range(bar2, -50, 50);
11. lv_bar_set_value(bar2, 25);
12.
13. /* Copy 'bar2' but set its size to be vertical (indicator at 75%)*
14. lv_obj_t * bar3 = lv_bar_create(lv_scr_act(), bar2);
15. lv_obj_set_size(bar3, 50, 200);
16. lv_obj_align(bar3, bar2, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20);
17.
18.
19. /* Copy 'bar2' and set new style for it
20. * (like 'bar2' on its left but dark bg, thin red indicator with big light)*
   /
21. static lv_style_t bar_bg;
22. lv_style_copy(&bar_bg, &lv_style_pretty);
23. bar_bg.body.main_color = LV_COLOR_BLACK;
24.
25. static lv_style_t bar_indic;
26. lv_style_copy(&bar_indic, &lv_style_pretty);
27. bar_indic.body.main_color = LV_COLOR_RED;
28. bar_indic.body.grad_color = LV_COLOR_MARRON;
29. bar_indic.body.shadow.color = LV_COLOR_RED;
30. bar_indic.body.shadow.width = 20;
31. bar_indic.body.padding.ver = 10;      /*Make the indicator thinner*/
32.
33. lv_obj_t * bar4 = lv_bar_create(lv_scr_act(), bar2);
34. lv_obj_align(bar4, bar2, LV_ALIGN_OUT_RIGHT_MID, 20, 0);
35. lv_bar_set_style(bar4, LV_BAR_STYLE_BG, &bar_bg);
36. lv_bar_set_style(bar4, LV_BAR_STYLE_INDIC, &bar_indic);
37.
38. /* Copy 'bar4' but set its size to be vertical*/
39. lv_obj_t * bar5 = lv_bar_create(lv_scr_act(), bar4);
40. lv_obj_set_size(bar5, 50, 200);
41. lv_obj_align(bar5, bar4, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20);
```

运行结果:



2.10 Line Meter



- 线型仪表对象包含一些绘制比例的径向线。 使用 `lv_lmeter_set_value(lmeter, new_value)` 设置新值时，刻度的比例部分将重新着色。
- `lv_lmeter_set_range(lmeter, min, max)` 函数设置线路表的范围。
- 您可以通过以下方式设置刻度的角度和线的数量：
`lv_lmeter_set_scale(lmeter, angle, line_num)`。 默认角度为 240，默认行号为 31

2.10.1 相关宏定义

2.10.2 API 函数

函数	描述
<code>lv_lmeter_create ()</code>	创建线型仪表控件
<code>lv_lmeter_set_value()</code>	设置线型仪表的新值
<code>lv_lmeter_set_range()</code>	设置线型仪表的最小值和最大值
<code>lv_lmeter_set_scale()</code>	设置线型仪表的刻度设置
<code>lv_lmeter_get_value()</code>	获取线型仪表的值
<code>lv_lmeter_get_min_value()</code>	获取线型仪表的最小值

lv_lmeter_get_max_value()	获取线型仪表的最大值
lv_lmeter_get_line_count()	获取线型仪表的刻度数
lv_lmeter_get_scale_angle()	获取线型仪表的刻度角

lv_lmeter_create()

创建线型仪表控件

lv_obj_t * lv_lmeter_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新线型仪表的父控件
copy	指向窗口对象的指针，如果不是NULL，则将从该控件复制新控件

返回值：创建的线型仪表指针

lv_lmeter_set_value()

设置线路仪表的新值

void lv_lmeter_set_value(lv_obj_t *lmeter, int16_t value)

参数	描述
lmeter	线型仪表指针
value	新值

返回值：无

lv_lmeter_set_range()

设置线型仪表的最小值和最大值

void lv_lmeter_set_range(lv_obj_t *lmeter, int16_t min, int16_t max)

参数	描述
lmeter	线型仪表指针
min	最小值
max	最大值

返回值：无

lv_lmeter_set_scale()

设置线型仪表的刻度设置

void lv_lmeter_set_scale(lv_obj_t * lmeter, uint16_t angle, uint8_t line_cnt)

参数	描述
lmeter	线型仪表指针
angle	刻度角度（0..360）
line_cnt	行数

返回值：无

lv_lmeter_set_style()

设置线型仪表的样式

static inline void lv_lmeter_set_style(lv_obj_t *lmeter, lv_style_t *bg)

参数	描述
lmeter	线型仪表指针
bg	设置线型仪表的样式

返回值：无

lv_lmeter_get_value()

获取线型仪表的值

int16_t lv_lmeter_get_value(lv_obj_t *lmeter)

参数	描述
lmeter	线型仪表指针

返回值：线型仪表的值

lv_lmeter_get_min_value()

获取线型仪表的最小值

int16_t lv_lmeter_get_min_value(lv_obj_t *lmeter)

参数	描述
lmeter	线型仪表指针

返回值：线型仪表的最小值

lv_lmeter_get_max_value()

获取线型仪表的最大值

int16_t lv_lmeter_get_max_value(lv_obj_t *lmeter)

参数	描述
lmeter	线型仪表指针

返回值线型仪表的最大值

lv_lmeter_get_line_count()

获取线型仪表的刻度数

uint8_t lv_lmeter_get_line_count(lv_obj_t *lmeter)

参数	描述
lmeter	线型仪表指针

返回值：线型仪表的刻度数

lv_lmeter_get_scale_angle()

获取线型仪表的刻度角

uint16_t lv_lmeter_get_scale_angle(lv_obj_t *lmeter)

参数	描述
lmeter	线型仪表指针

返回值：线型仪表的刻度角

lv_lmeter_get_style_bg()

获得线型仪表的风格

static inline lv_style_t * lv_lmeter_get_style_bg(lv_obj_t *lmeter)

参数	描述
----	----

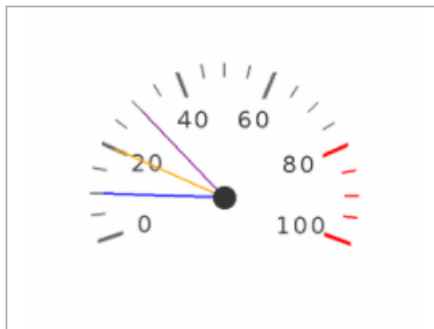
lmeter**线型仪表指针**

返回值：线型仪表的样式

2.10.3 控件使用

```
1. /* Create a default object*/
2. lv_obj_t *lmeter1 = lv_lmeter_create(lv_scr_act(), NULL);
3. lv_obj_set_pos(lmeter1, 10, 10);
4. lv_lmeter_set_value(lmeter1, 60);
5.
6. /*Copy the previous line meter and set smaller size for it*/
7. lv_obj_t *lmeter2 = lv_lmeter_create(lv_scr_act(), lmeter1);
8. lv_obj_set_size(lmeter2, LV_DPI / 2, LV_DPI / 2);
9. lv_obj_align(lmeter2, lmeter1, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
10.
11. /*Create a styled line meter*/
12. static lv_style_t style3;
13. lv_style_copy(&style3, &lv_style_pretty);
14. style3.body.main_color = LV_COLOR_GREEN;
15. style3.body.grad_color = LV_COLOR_RED;
16. style3.body.padding.hor = 4;
17. style3.body.border.color= LV_COLOR_GRAY;          /*Means the needle middle*/
18. style3.line.width = 2;
19. style3.line.color = LV_COLOR_SILVER;
20.
21. lv_obj_t *lmeter3 = lv_lmeter_create(lv_scr_act(), lmeter1);
22. lv_obj_align(lmeter3, lmeter1, LV_ALIGN_OUT_RIGHT_MID, 20, 0);
23. lv_obj_set_style(lmeter3, &style3);
24. lv_lmeter_set_scale(lmeter3, 270, 41);
25. lv_lmeter_set_range(lmeter3, -100, 100);
26. lv_lmeter_set_value(lmeter3, 50);
27.
28.
29. /*Copy the modified 'lmeter3' and set a smaller size for it*/
30. lv_obj_t *lmeter4 = lv_lmeter_create(lv_scr_act(), lmeter3);
31. lv_obj_set_size(lmeter4, 60, 60);
32. lv_obj_align(lmeter4, lmeter3, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
```

2.11 Gauge



- 仪表是带刻度标签和针头的仪表。您可以使用 `lv_gauge_set_scale(gauge, angle, line_num, label_cnt)` 函数来调整缩放角度以及缩放线和标签的数量。默认设置为：220 度角，6 个刻度标签和 21 行
- 仪表可以显示多个针头。使用 `lv_gauge_set_needle_count(gauge, needle_num, color_array)` 函数设置针数和每个针的颜色数组。（数组必须是静态或全局变量）
- 您可以使用 `lv_gauge_set_value(gauge, needle_id, value)` 来设置针的值。
- 要设置临界值，请使用 `lv_gauge_set_critical_value(gauge, value)`。在 `ti` 值之后，缩放颜色将变为 `line.color`。（默认值：80）
- 仪表的范围可以通过 `lv_gauge_set_range(gauge, min, max)` 指定

2.11.1 相关宏定义

2.11.2 API 函数

函数	描述
<code>lv_gauge_create()</code>	创建仪表控件
<code>lv_gauge_set_needle_count()</code>	设定针数
<code>lv_gauge_set_value()</code>	设置针的值
<code>lv_gauge_set_scale()</code>	设置仪表的刻度设置
<code>lv_gauge_get_value()</code>	获得针的数值
<code>lv_gauge_get_needle_count()</code>	获取仪表上的针数
<code>lv_gauge_get_label_count()</code>	获取标签数量

`lv_gauge_create()`

创建仪表控件

`lv_obj_t * lv_gauge_create(lv_obj_t * par, lv_obj_t * copy)`

参数	描述
<code>par</code>	指向控件的指针，它将是新仪表的父

	控件
copy	指向窗口对象的指针，如果不是NULL，则将从该控件复制新控件

返回值：创建的仪表指针

lv_gauge_set_needle_count()

设定针数

void lv_gauge_set_needle_count(lv_obj_t * gauge, uint8_t needle_cnt, const lv_color_t * colors)

参数	描述
gauge	仪表指针
needle_cnt	新的针数
colors	针的颜色数组（带'num'元素）

返回值：无

lv_gauge_set_value()

设置针的值

void lv_gauge_set_value(lv_obj_t * gauge, uint8_t needle_id, int16_t value)

参数	描述
gauge	仪表指针
needle_id	针的 id
value	新值

返回值：无

lv_gauge_set_range()

设置仪表的最小值和最大值

static inline void lv_gauge_set_range(lv_obj_t * gauge, int16_t min, int16_t max)

参数	描述
gauge	仪表指针
min	最小值
max	最大值

返回值：无

lv_gauge_set_critical_value()

在比例上设置临界值。在此值'line.color'之后将绘制比例线

static inline void lv_gauge_set_critical_value(lv_obj_t * gauge, int16_t value)

参数	描述
gauge	仪表指针
value	临界值

返回值：无

lv_gauge_set_scale()

设置仪表的刻度设置

void lv_gauge_set_scale(lv_obj_t * gauge, uint16_t angle, uint8_t line_cnt, uint8_t label_cnt)

参数	描述
gauge	仪表指针
angle	刻度角（0..360）
line_cnt	比例线的数量
label_cnt	刻度标签数量

返回值：无

lv_gauge_set_style()

设置仪表的样式

static inline void lv_gauge_set_style(lv_obj_t * gauge, lv_style_t * bg)

参数	描述
gauge	仪表指针
bg	设置仪表的样式

返回值：无

lv_gauge_get_value()

获得针的数值

int16_t lv_gauge_get_value(lv_obj_t * gauge, uint8_t needle)

参数	描述
gauge	仪表指针
needle	针的 id

返回值：针的数值

lv_gauge_get_needle_count()

获取仪表上的针数

uint8_t lv_gauge_get_needle_count(lv_obj_t * gauge)

参数	描述
gauge	仪表指针

返回值：仪表上的针数

lv_gauge_get_min_value()

获取仪表最小值

static inline int16_t lv_gauge_get_min_value(lv_obj_t * lmeter)

参数	描述
gauge	仪表指针

返回值：仪表的最小值

lv_gauge_get_max_value()

获取仪表最大值

static inline int16_t lv_gauge_get_max_value(lv_obj_t * lmeter)

参数	描述
gauge	仪表指针

返回值：仪表的最大值

`lv_gauge_get_critical_value()`

获取刻度盘上的临界值

`static inline int16_t lv_gauge_get_critical_value(lv_obj_t * gauge)`

参数	描述
gauge	仪表指针

返回值：临界值

`lv_gauge_get_label_count()`

获取标签数量

`uint8_t lv_gauge_get_label_count(lv_obj_t * gauge)`

参数	描述
gauge	仪表指针

返回值：标签数量

`lv_gauge_get_line_count()`

获取仪表的刻度数

`static inline uint8_t lv_gauge_get_line_count(lv_obj_t * gauge)`

参数	描述
gauge	仪表指针

返回值：比例单位的数量

`lv_gauge_get_scale_angle()`

获得仪表的刻度角

`static inline uint16_t lv_gauge_get_scale_angle(lv_obj_t * gauge)`

参数	描述
gauge	仪表指针

返回值：比例角度

`lv_gauge_get_style()`

获得仪表的风格

`static inline lv_style_t * lv_gauge_get_style(lv_obj_t * gauge)`

参数	描述
gauge	仪表指针

返回值：指向仪表样式的指针

2.11.3 控件使用

基本使用流程

➤ 创建仪表控件

```
1. lv_obj_t *gauge1 = lv_gauge_create(lv_scr_act(), NULL);
```

➤ 设置控件显示坐标

```
1. lv_obj_set_pos(gauge1, 10, 10);
```

➤ 设置控件尺寸大小

```
1. lv_obj_set_size(gauge1,100,100);
```

➤ 设置仪表新值

```
1. lv_gauge_set_value(gauge1, 0, 75);
```

➤ 仪表刻度设置

```
1. lv_gauge_set_scale(gauge1, 270, 41, 5);
```

示例：

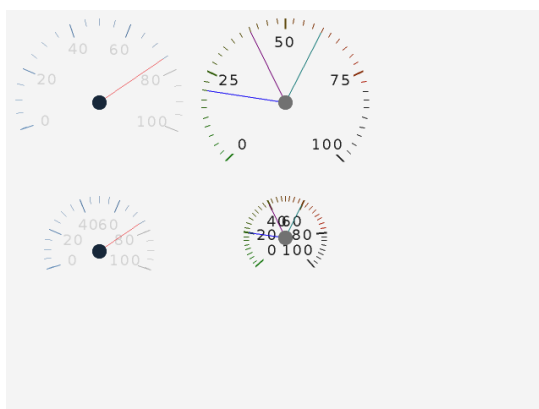
```
1. /* Create a default object*/
2. lv_obj_t *gauge1 = lv_gauge_create(lv_scr_act(), NULL);
3. lv_obj_set_pos(gauge1, 10, 10);
4. lv_gauge_set_value(gauge1, 0, 75);
5.
6.
7. /*Copy the previous gauge and set smaller size for it*/
8. lv_obj_t *gauge2 = lv_gauge_create(lv_scr_act(), gauge1);
9. lv_obj_set_size(gauge2, 2 * lv_obj_get_width(gauge1) / 3, 2 * lv_obj_get_height(gauge1) / 3);
10. lv_obj_align(gauge2, gauge1, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
11.
12. /*Copy the first gauge add more needles and set new style*/
13. lv_color_t needle_colors[3] = {LV_COLOR_BLUE, LV_COLOR_PURPLE, LV_COLOR_TEAL};
14.
15. /*Create a styled gauge*/
16. static lv_style_t style3;
17. lv_style_copy(&style3, &lv_style_pretty);
18. style3.body.main_color = LV_COLOR_GREEN;
19. style3.body.grad_color = LV_COLOR_RED;
20. style3.body.padding.hor = 6;
21. style3.body.padding.inner = 10;
22. style3.body.padding.ver = 8;
23. style3.body.border.color= LV_COLOR_GRAY;
```

```

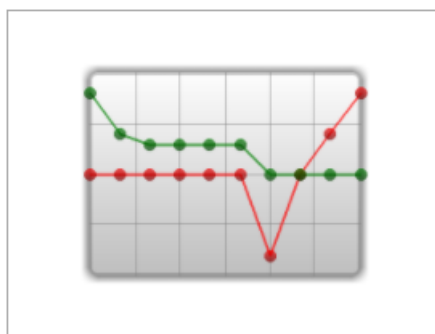
24. style3.line.width = 2;
25.
26. lv_obj_t *gauge3 = lv_gauge_create(lv_scr_act(), gauge1);
27. lv_obj_align(gauge3, gauge1, LV_ALIGN_OUT_RIGHT_MID, 20, 0);
28. lv_obj_set_style(gauge3, &style3);
29. lv_gauge_set_scale(gauge3, 270, 41, 5);
30. lv_gauge_set_needle_count(gauge3, 3, needle_colors);
31. lv_gauge_set_value(gauge3, 0, 20);
32. lv_gauge_set_value(gauge3, 1, 40);
33. lv_gauge_set_value(gauge3, 2, 60);
34.
35. /*Copy the modified 'gauge3' and set a smaller size for it*/
36. lv_obj_t *gauge4 = lv_gauge_create(lv_scr_act(), gauge3);
37. lv_obj_set_size(gauge4, 100, 100);
38. lv_obj_align(gauge4, gauge3, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);

```

运行结果：



2.12 Chart



图表具有类似矩形的背景，具有水平和垂直分割线。您可以通过 `lv_chart_add_series(chart, color)` 向图表添加任意数量的系列。它为 `lv_chart_series_t` 结构分配数据，该结构包含所选颜色和数据数组。您有几个选项来设置系列数据：

- 1.在数组中手动设置值，如 `ser1->points[3] = 7`，并使用 `lv_chart_refresh(chart)` 刷新图表。
- 2.使用 `lv_chart_set_next(chart, ser, value)` 函数将所有数据移至左侧，并在最右侧位置设置新数据。
- 3.使用以下命令将所有点初始化为给定值： `lv_chart_init_points(chart, ser, value)`
- 4.使用以下命令设置数组中的所有点： `lv_chart_set_points(chart, ser, value_array)`

有四种数据显示类型：

- `LV_CHART_TYPE_NONE`: 不显示描点。如果您想添加自己的绘制方法，可以使用它。
- `LV_CHART_TYPE_LINE`: 在描点之间画线
- `LV_CHART_TYPE_COL`: 绘制列
- `LV_CHART_TYPE_POINT`: 绘制描点

您可以使用 `lv_chart_set_type(chart, TYPE)` 指定显示类型。

`LV_CHART_TYPE_LINE` | `LV_CHART_TYPE_POINT` 类型也可用于绘制线和点

您可以指定最小值和最大值。使用 `lv_chart_set_range(chart, y_min, y_max)` 设定在 y 方向上的值。点的数值将按比例缩放。默认范围是：0..100。

数据行中的点数可以通过 `lv_chart_set_point_count(chart, point_num)` 进行修改。默认值为 10。

可以通过 `lv_chart_set_div_line_count(chart, hdiv_num, vdiv_num)` 修改水平和垂直分割线的数量。默认设置为 3 个水平线和 5 个垂直分割线。

要设置线宽和点半径，请使用 `lv_chart_set_series_width(chart, size)` 函数。默认值为：2。

数据行的不透明度可以由 `lv_chart_set_series_opa(chart, opa)` 指定。默认值为：`OPA_COVER`。

您可以通过 `lv_chart_set_series_darking(chart, effect)` 功能在列的底部和点上应用深色淡入淡出。默认的暗级是 `OPA_50`。

2.12.1 相关宏定义

图表类型(<code>lv_chart_type_t</code>)	
<code>LV_CHART_TYPE_LINE</code>	在描点之间画线
<code>LV_CHART_TYPE_COLUMN</code>	绘制列
<code>LV_CHART_TYPE_POINT</code>	绘制描点

2.12.2 API 函数

函数	描述
lv_chart_create ()	创建图表控件
lv_chart_add_series()	分配数据系列并将其添加到图表中
lv_chart_set_div_line_count()	设置水平和垂直分割线的数量
lv_chart_set_range()	设置最小和最大 y 值
lv_chart_set_type()	为图表设置新类型
lv_chart_set_point_count()	设置图表上数据行的点数
lv_chart_set_series_opa()	设置数据系列的不透明度
lv_chart_set_series_width()	设置数据系列的线宽或点半径
lv_chart_set_series_darking()	在点或列的底部设置暗效果
lv_chart_init_points()	使用值初始化所有数据点
lv_chart_set_points()	设置数组中点的值
lv_chart_set_next()	正确移位所有数据并在数据线上设置最正确的数据
lv_chart_get_type()	获取图表的类型
lv_chart_get_point_cnt()	获取图表上每条数据线的的数据点数
lv_chart_get_series_opa()	获取数据系列的不透明度
lv_chart_get_series_width()	获取数据系列宽度
lv_chart_get_series_darking()	获取点或列底部的暗效果级别
lv_chart_refresh()	如果数据行已更改，刷新图表

lv_chart_create()

创建图表控件

lv_obj_t * lv_chart_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新图表的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的图表指针

lv_chart_add_series()

分配数据系列并将其添加到图表中

lv_chart_series_t * lv_chart_add_series(lv_obj_t * chart, lv_color_t color)

参数	描述
chart	图表指针
color	数据系列的顏色

返回值：指向已分配数据系列的指针

lv_chart_set_div_line_count()

设置水平和垂直分割线的数量

void lv_chart_set_div_line_count(lv_obj_t * chart, uint8_t hdiv, uint8_t vdiv)

参数	描述
chart	图表指针
hdiv	水平分割线的数量
vdiv	垂直分割线的数量

返回值：无

lv_chart_set_range()

设置最小和最大 y 值

void lv_chart_set_range(lv_obj_t * chart, lv_coord_t ymin, lv_coord_t ymax)

参数	描述
chart	图表指针
ymin	最小 y 值
ymax	最大 y 值

返回值：无

lv_chart_set_type()

为图表设置新类型

void lv_chart_set_type(lv_obj_t * chart, lv_chart_type_t type)

参数	描述
chart	图表指针
type	新类型的图表（来自 lv_chart_type_t 枚举）

返回值：无

lv_chart_set_point_count()

设置图表上数据行的点数

void lv_chart_set_point_count(lv_obj_t * chart, uint16_t point_cnt)

参数	描述
chart	图表指针
point_cnt	数据线上的新点数

返回值：无

lv_chart_set_series_opa()

设置数据系列的不透明度

void lv_chart_set_series_opa(lv_obj_t * chart, lv_opa_t opa)

参数	描述
chart	图表指针
opa	数据系列的不透明度

返回值：无

lv_chart_set_series_width()

设置数据系列的线宽或点半径

void lv_chart_set_series_width(lv_obj_t * chart, lv_coord_t width)

参数	描述
chart	图表指针
width	新的宽度

返回值：无

lv_chart_set_series_darking()

在点或列的底部设置暗效果

void lv_chart_set_series_darking(lv_obj_t * chart, lv_opa_t dark_eff)

参数	描述
chart	图表指针
dark_eff	暗效果等级（LV_OPA_TRANSP 关闭）

返回值：无

lv_chart_init_points()

使用值初始化所有数据点

void lv_chart_init_points(lv_obj_t * chart, lv_chart_series_t * ser, lv_coord_t y)

参数	描述
chart	图表指针
ser	指向“图表”上数据系列的指针
y	对于所有的点新值

返回值：无

lv_chart_set_points()

设置数组中点的值

void lv_chart_set_points(lv_obj_t * chart, lv_chart_series_t * ser, lv_coord_t * y_array)

参数	描述
chart	图表指针
ser	指向“图表”上数据系列的指针
y_array	'lv_coord_t'点数组（带'点数'元素）

返回值：无

lv_chart_set_next()

正确移位所有数据并在数据线上设置最正确的数据

void lv_chart_set_next(lv_obj_t * chart, lv_chart_series_t * ser, lv_coord_t y)

参数	描述
chart	图表指针
ser	指向“图表”上数据系列的指针
y	最合适数据的新值

返回值：无

lv_chart_get_type()

获取图表的类型

lv_chart_type_t lv_chart_get_type(lv_obj_t * chart)

参数	描述
chart	图表指针

返回值：图表的类型（来自 [lv_chart_type_t](#) 枚举）

lv_chart_get_point_cnt()

获取图表上每条数据线的数据点数

uint16_t lv_chart_get_point_cnt(lv_obj_t * chart)

参数	描述
chart	图表指针

返回值：每条数据线上的点数

lv_chart_get_series_opa()

获取数据系列的不透明度

lv_opa_t lv_chart_get_series_opa(lv_obj_t * chart)

参数	描述
chart	图表指针

返回值：数据系列的不透明度

lv_chart_get_series_width()

获取数据系列宽度

lv_coord_t lv_chart_get_series_width(lv_obj_t * chart)

参数	描述
chart	图表指针

返回值：数据系列的宽度（线或点）

lv_chart_get_series_darking()

获取点或列底部的暗效果级别

lv_opa_t lv_chart_get_series_darking(lv_obj_t * chart)

参数	描述
chart	图表指针

返回值：暗效果等级（LV_OPA TRANSP 关闭）

lv_chart_refresh()

如果数据行已更改，请刷新图表

void lv_chart_refresh(lv_obj_t * chart)

参数	描述
chart	图表指针

返回值：无

2.12.3 控件使用

基本使用流程

➤ 创建控件

```
1. lv_obj_t * chart1 = lv_chart_create(lv_scr_act(), NULL);
```

➤ 设置控件显示坐标

```
1. lv_obj_set_pos(chart1, 10, 10);
```

➤ 设置控件尺寸大小

```
1. lv_obj_set_size(chart1, 140, 100);
```

➤ 添加显示数据

```
1. lv_chart_series_t * dl1_1 = lv_chart_add_series(chart1, LV_COLOR_RED);
2. dl1_1->points[0] = 0;
3. dl1_1->points[1] = 25;
4. dl1_1->points[2] = 0;
5. dl1_1->points[3] = 50;
6. dl1_1->points[4] = 0;
7. dl1_1->points[5] = 75;
8. dl1_1->points[6] = 0;
9. dl1_1->points[7] = 100;
10. dl1_1->points[8] = 0;
11. lv_chart_refresh(chart1);
```

示例:

```
1. /* Create a default object*/
2. lv_obj_t * chart1 = lv_chart_create(lv_scr_act(), NULL);
3.
4. lv_chart_series_t * dl1_1 = lv_chart_add_series(chart1, LV_COLOR_RED);
5. dl1_1->points[0] = 0;
6. dl1_1->points[1] = 25;
7. dl1_1->points[2] = 0;
8. dl1_1->points[3] = 50;
9. dl1_1->points[4] = 0;
10. dl1_1->points[5] = 75;
11. dl1_1->points[6] = 0;
12. dl1_1->points[7] = 100;
13. dl1_1->points[8] = 0;
```

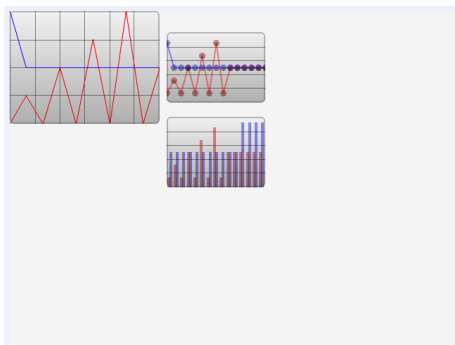
```

14.
15. lv_chart_series_t * dl1_2 = lv_chart_add_series(chart1, LV_COLOR_BLUE);
16. dl1_2->points[0] = 100;
17.
18. lv_chart_refresh(chart1);
19.
20.
21. /* Create a chart with the same data and modify all appearance-like attributes
22.  * also modify the number of points, range, and type*/
23. lv_obj_t * chart2 = lv_chart_create(lv_scr_act(), NULL);
24. lv_obj_set_size(chart2, 140, 100);
25. lv_obj_align(chart2, chart1, LV_ALIGN_OUT_RIGHT_MID, 10, 0);
26. lv_chart_set_series_darking(chart2, LV_OPA_90);
27. lv_chart_set_series_opa(chart2, LV_OPA_40);
28. lv_chart_set_series_width(chart2, 4);
29. lv_chart_set_type(chart2, LV_CHART_TYPE_POINT | LV_CHART_TYPE_LINE);
30. lv_chart_set_range(chart2, -20, 120);
31. lv_chart_set_div_line_count(chart2, 4, 0);
32.
33. lv_chart_series_t * dl2_1 = lv_chart_add_series(chart2, LV_COLOR_RED);
34. dl2_1->points[0] = 0;
35. dl2_1->points[1] = 25;
36. dl2_1->points[2] = 0;
37. dl2_1->points[3] = 50;
38. dl2_1->points[4] = 0;
39. dl2_1->points[5] = 75;
40. dl2_1->points[6] = 0;
41. dl2_1->points[7] = 100;
42. dl2_1->points[8] = 0;
43.
44. lv_chart_series_t * dl2_2 = lv_chart_add_series(chart2, LV_COLOR_BLUE);
45. dl2_2->points[0] = 100;
46.
47. lv_chart_refresh(chart2);
48.
49. lv_chart_set_point_count(chart2, 15);
50.
51.
52. /*Copy the previous chart, set COLUMN type and test lv_chart_set_next()*/
53. lv_obj_t * chart3 = lv_chart_create(lv_scr_act(), chart2);
54. lv_obj_align(chart3, chart2, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);
55. lv_chart_set_type(chart3, LV_CHART_TYPE_COLUMN);
56. lv_chart_series_t * dl3_1 = lv_chart_add_series(chart3, LV_COLOR_RED);

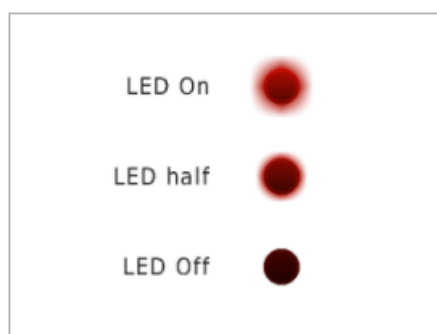
```

```
57. dl3_1->points[0] = 0;
58. dl3_1->points[1] = 25;
59. dl3_1->points[2] = 0;
60. dl3_1->points[3] = 50;
61. dl3_1->points[4] = 0;
62. dl3_1->points[5] = 75;
63. dl3_1->points[6] = 0;
64. dl3_1->points[7] = 100;
65. dl3_1->points[8] = 0;
66.
67. lv_chart_series_t * dl3_2 = lv_chart_add_series(chart3, LV_COLOR_BLUE);
68. dl3_2->points[0] = 100;
69.
70. lv_chart_refresh(chart2);
71.
72. lv_chart_set_next(chart3, dl3_2, 110);
73. lv_chart_set_next(chart3, dl3_2, 110);
74. lv_chart_set_next(chart3, dl3_2, 110);
75. lv_chart_set_next(chart3, dl3_2, 110);
```

运行效果：



2.13 LED



LED 是矩形（或圆形）物体。 您可以使用 `lv_led_set_bright(led, bright)` 设置亮度。 亮度应介于 0（最暗）和 255（最亮）之间。

使用 `lv_led_on(led)` 和 `lv_led_off(led)` 将亮度设置为预定义的 ON 或 OFF 值。
`lv_led_toggle(led)` 在 ON 和 OFF 状态之间切换

2.13.1 相关宏定义

2.13.2 API 函数

函数	描述
lv_led_create ()	创建 LED 控件
lv_led_set_bright()	设置 LED 对象的亮度
lv_led_on()	点亮 LED
lv_led_off()	点灭 LED
lv_led_toggle()	切换 LED 的状态
lv_led_get_bright()	获取 LED 对象的亮度

`lv_led_create()`

创建 LED 控件

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

参数	描述
par	指向控件的指针，它将是新 LED 的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的 LED 指针

`lv_led_set_bright()`

设置 LED 对象的亮度

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

参数	描述
led	LED 控件指针
bright	0（最暗）... 255（最亮）

返回值：无

`lv_led_on()`

点亮 LED

`void lv_led_on(lv_obj_t * led)`

参数	描述
led	LED 控件指针

返回值：无

`lv_led_off()`

点灭 LED

`void lv_led_off(lv_obj_t * led)`

参数	描述
led	LED 控件指针

返回值：无

`lv_led_toggle()`

切换 LED 的状态

`void lv_led_toggle(lv_obj_t * led)`

参数	描述
led	LED 控件指针

返回值：无

`lv_led_set_style()`

设置 LED 样式

`static inline void lv_led_set_style(lv_obj_t *led, lv_style_t *style)`

参数	描述
led	LED 控件指针
style	样式指针

返回值：无

`lv_led_get_bright()`

获取 LED 对象的亮度

`uint8_t lv_led_get_bright(lv_obj_t * led)`

参数	描述
led	LED 控件指针

返回值：0（最暗）... 255（最亮）

`lv_led_get_style()`

获取 LED 样式

`static inline lv_style_t* lv_led_get_style(lv_obj_t *led)`

参数	描述
led	LED 控件指针

返回值：样式指针

2.13.3 控件使用

基本使用流程

➤ 创建控件

```
1. lv_obj_t * led1 = lv_led_create(lv_scr_act(), NULL);
```

➤ 设置控件显示坐标

```
1. lv_obj_set_pos(led1, 20, 20);
```

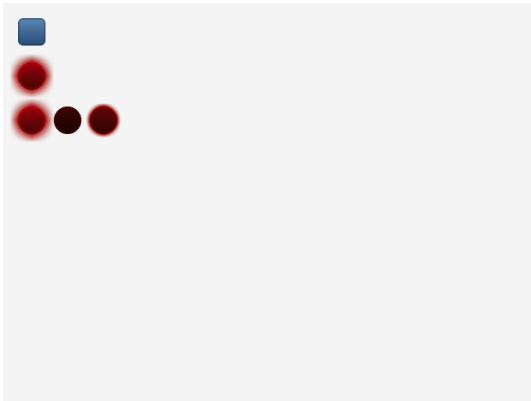
➤ 点亮 LED

```
1. lv_led_on(led1);
```

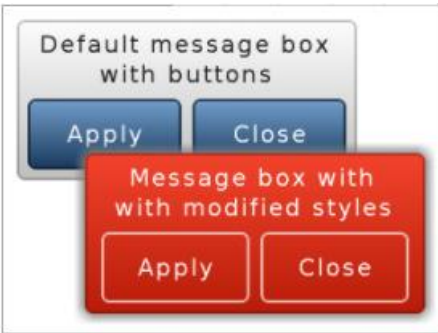
示例:

```
1. /* Create a default object*/
2. lv_obj_t * led1 = lv_led_create(lv_scr_act(), NULL);
3. lv_obj_set_pos(led1, 20, 20);
4.
5. /*Create styles LED*/
6. static lv_style_t style;
7. lv_style_copy(&style, &lv_style_pretty_color);
8. style.body.shadow.width = 10;
9. style.body.radius = LV_RADIUS_CIRCLE;
10. style.body.border.width= 3;
11. style.body.border.opa = LV_OPA_30;
12. style.body.main_color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
13. style.body.grad_color = LV_COLOR_MAKE(0x50, 0x07, 0x02);
14. style.body.border.color = LV_COLOR_MAKE(0xfa, 0x0f, 0x00);
15. style.body.shadow.color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
16.
17. lv_obj_t * led2 = lv_led_create(lv_scr_act(), NULL);
18. lv_led_set_style(led2, &style);
19. lv_obj_align(led2, led1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20);
20.
21. /*Turned ON LED*/
22. lv_obj_t * led_on = lv_led_create(lv_scr_act(), led2);
23. lv_obj_align(led_on, led2, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20);
24. lv_led_on(led_on);
25.
26. /*Tuned OFF LED*/
27. lv_obj_t * led_off = lv_led_create(lv_scr_act(), led_on);
28. lv_obj_align(led_off, led_on, LV_ALIGN_OUT_RIGHT_MID, 10, 0);
29. lv_led_off(led_off);
30.
31. /*Arbitrary brightness*/
32. lv_obj_t * led_x = lv_led_create(lv_scr_act(), led_off);
33. lv_obj_align(led_x, led_off, LV_ALIGN_OUT_RIGHT_MID, 10, 0);
34. lv_led_set_bright(led_x, 170);
```


运行结果



2.14 Message Box



- 消息框充当弹出窗口。它们是由背景，文本和按钮构建的。背景是一个 `Container` 对象，启用垂直拟合以确保文本和按钮始终可见。
- 要设置文本，请使用 `lv_mbox_set_text(mbox, "My text")` 功能。
- 按钮是按钮矩阵。要添加按钮，请使用 `lv_mbox_add_btns(mbox, btn_str, action)` 函数。在这里你可以指定按钮文本，例如 (`const char * btn_str[] = {"btn1", "btn2", ""}`)并添加一个释放按钮时调用的回调。有关更多信息，请访问[Button matrix(/object-types/button-matrix-lv_btnm)]的文档
- 使用 `lv_mbox_start_auto_close(mbox, delay)`，可以在延迟毫秒后使用长动画自动关闭消息框。`lv_mbox_stop_auto_close(mbox)` 函数将停止开始自动关闭
- 关闭动画时间可以通过 `lv_mbox_set_anim_time(mbox, anim_time)` 进行调整

2.14.1 相关宏定义

信息框样式(lv_mbox_style_t)	
LV_MBOX_STYLE_BG	指定背景容器的样式。 style.body 为背景和 style.label 为文本外观。 默认值：lv_style_pretty
LV_MBOX_STYLE_BTN_BG	按钮（按钮矩阵）背景的样式。 默认值：lv_style_transp

LV_MBOX_STYLE_BTN_REL	释放按钮的样式。 默认值: lv_style_btn_rel
LV_MBOX_STYLE_BTN_PR	按下按钮的样式。 默认值: lv_style_btn_pr
LV_MBOX_STYLE_BTN_TGL_REL	切换释放按钮的样式。 默认值: lv_style_btn_tgl_rel
LV_MBOX_STYLE_BTN_TGL_PR	切换按钮的样式。 默认值: lv_style_btn_tgl_pr
LV_MBOX_STYLE_BTN_INA	非活动按钮的样式。 默认值: lv_style_btn_ina

2.14.2 API 函数

函数	描述
lv_mbox_create()	创建消息框
lv_mbox_add_btns()	在消息框中添加按钮
lv_mbox_set_text()	设置消息框的文本
lv_mbox_set_action()	停止按钮释放时要调用的操作
lv_mbox_set_anim_time()	设置动画持续时间
lv_mbox_start_auto_close()	在给定时间后自动删除消息框
lv_mbox_stop_auto_close()	停止自动,关闭消息框
lv_mbox_set_style()	设置消息框的样式
lv_mbox_get_text()	获取消息框的文本
lv_mbox_get_from_btn()	从其中一个按钮获取消息框对象
lv_mbox_get_anim_time()	获取动画持续时间 (关闭动画时间)
lv_mbox_get_style()	获得一个消息框的样式

lv_mbox_create()

创建消息框

lv_obj_t * lv_mbox_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新消息框的父控件
copy	指向窗口对象的指针，如果不是NULL，则将从该控件复制新控件

返回值：创建的消息框指针

lv_mbox_add_btns()

在消息框中添加按钮

void lv_mbox_add_btns(lv_obj_t * mbox, const char **btn_map,
lv_btnm_action_t action)

参数	描述
----	----

mbox	消息框控件指针
btn_map	按钮描述符（按钮矩阵图）。 *例如 <code>const char * txt [] = { “ok” , “close” , “ ” }</code> （不能是局部变量）
action	释放按钮时将调用的函数

返回值：无

`lv_mbox_set_text()`

设置消息框的文本

`void lv_mbox_set_text(lv_obj_t * mbox, const char * txt)`

参数	描述
mbox	消息框控件指针
txt	一个'\0'终止的字符串，消息框文本

返回值：无

`lv_mbox_set_action()`

停止按钮释放时要调用的操作

`void lv_mbox_set_action(lv_obj_t * mbox, lv_btm_action_t action)`

参数	描述
mbox	消息框控件指针
action	指向 <code>lv_btm_action_t</code> 动作的指针

返回值：无

`lv_mbox_set_anim_time()`

设置动画持续时间

`void lv_mbox_set_anim_time(lv_obj_t * mbox, uint16_t anim_time)`

参数	描述
mbox	消息框控件指针
anim_time	动画长度，以毫秒为单位（0：无动画）

返回值：无

`lv_mbox_start_auto_close()`

在给定时间后自动删除消息框

`void lv_mbox_start_auto_close(lv_obj_t * mbox, uint16_t delay)`

参数	描述
mbox	消息框控件指针
delay	删除消息框之前等待的时间（以毫秒为单位）

返回值：无

`lv_mbox_stop_auto_close()`

停止自动,关闭消息框

`void lv_mbox_stop_auto_close(lv_obj_t * mbox)`

参数	描述
----	----

mbox	消息框控件指针
-------------	---------

返回值：无

lv_mbox_set_style()

设置消息框的样式

void lv_mbox_set_style(lv_obj_t *mbox, lv_mbox_style_t type, lv_style_t *style)

参数	描述
mbox	消息框控件指针
type	要设置的样式类型
style	样式指针

返回值：无

lv_mbox_get_text()

获取消息框的文本

const char * lv_mbox_get_text(lv_obj_t * mbox)

参数	描述
mbox	消息框控件指针

返回值：指向消息框文本的指针

lv_mbox_get_from_btn()

从其中一个按钮获取消息框对象。

它在按钮释放操作中很有用，其中只有按钮是已知的

lv_obj_t * lv_mbox_get_from_btn(lv_obj_t * btn)

参数	描述
mbox	消息框控件指针

返回值：指向按钮消息框的指针

lv_mbox_get_anim_time()

获取动画持续时间（关闭动画时间）

uint16_t lv_mbox_get_anim_time(lv_obj_t * mbox)

参数	描述
mbox	消息框控件指针

返回值：动画长度，以毫秒为单位（0：无动画）

lv_mbox_get_style()

获得一个消息框的样式

lv_style_t * lv_mbox_get_style(lv_obj_t *mbox, lv_mbox_style_t type)

参数	描述
mbox	消息框控件指针
type	要获取的样式类型,见 lv_mbox_style_t

2.14.3 控件使用

基本使用流程

➤ 控件创建

```
1. lv_obj_t *mbox1 = lv_mbox_create(lv_scr_act(), NULL);
```

➤ 设置控件显示坐标

```
1. lv_obj_set_pos(mbox1, 10, 10);
```

➤ 添加按钮

```
1. static const char * btns1[] = {"Ok", "Cancel", ""};
2. lv_mbox_add_btns(mbox1, btns1, NULL);
```

➤ 设置消息框显示文本

```
1. lv_mbox_set_text(mbox1, "Message");
```

示例:

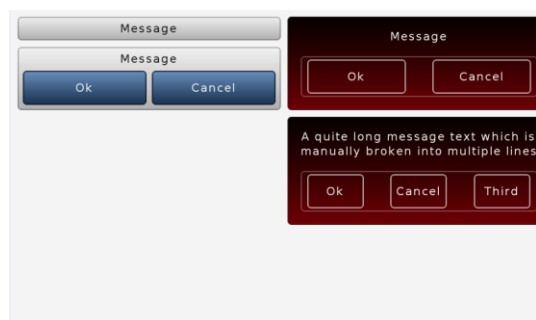
```
1. /* Default object */
2. lv_obj_t *mbox1 = lv_mbox_create(lv_scr_act(), NULL);
3. lv_obj_set_pos(mbox1, 10, 10);
4.
5. /*Add buttons and modify text*/
6. static const char * btns2[] = {"Ok", "Cancel", ""};
7. lv_obj_t *mbox2 = lv_mbox_create(lv_scr_act(), NULL);
8. lv_mbox_add_btns(mbox2, btns2, NULL);
9. lv_mbox_set_text(mbox2, "Message");
10. lv_obj_set_width(mbox2, LV_HOR_RES / 2);
11. lv_obj_align(mbox2, mbox1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 10);
12.
13. /*Add styles*/
14. static lv_style_t bg;
15. static lv_style_t btn_bg;
16. lv_style_copy(&bg, &lv_style_pretty);
17. lv_style_copy(&btn_bg, &lv_style_pretty);
18. bg.body.padding.hor = 20;
19. bg.body.padding.ver = 20;
20. bg.body.padding.inner = 20;
21. bg.body.main_color = LV_COLOR_BLACK;
22. bg.body.grad_color = LV_COLOR_MARRON;
```

```

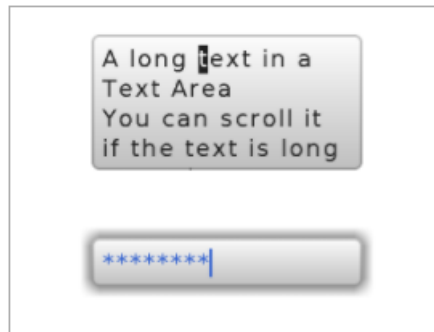
23. bg.text.color = LV_COLOR_WHITE;
24.
25. btn_bg.body.padding.hor = 10;
26. btn_bg.body.padding.ver = 5;
27. btn_bg.body.padding.inner = 40;
28. btn_bg.body.empty = 1;
29. btn_bg.body.border.color = LV_COLOR_WHITE;
30. btn_bg.text.color = LV_COLOR_WHITE;
31.
32. static lv_style_t btn_rel;
33. lv_style_copy(&btn_rel, &lv_style_btn_rel);
34. btn_rel.body.empty = 1;
35. btn_rel.body.border.color = LV_COLOR_WHITE;
36.
37. lv_obj_t *mbox3 = lv_mbox_create(lv_scr_act(), mbox2);
38. lv_mbox_set_style(mbox3, LV_MBOX_STYLE_BTN_REL, &btn_rel);
39. lv_mbox_set_style(mbox3, LV_MBOX_STYLE_BTN_BG, &btn_bg);
40. lv_mbox_set_style(mbox3, LV_MBOX_STYLE_BG, &bg);
41. lv_obj_align(mbox3, mbox1, LV_ALIGN_OUT_RIGHT_TOP, 10, 0);
42. lv_mbox_set_action(mbox3, mbox_action);
43.
44. /*Copy with styles and set button width*/
45. lv_obj_t *mbox4 = lv_mbox_create(lv_scr_act(), mbox3);
46. lv_mbox_set_text(mbox4, "A quite long message text which is\n"
47.                      "manually broken into multiple lines");
48.
49. static const char * btns3[] = {"Ok", "Cancel", "Third", ""};
50. lv_mbox_add_btns(mbox4, btns3, mbox_action);
51. lv_obj_align(mbox4, mbox3, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 10);

```

运行结果



2.15 Text Area



- 文本区域是一个带有标签和光标的页面。 您可以使用以下方法将文本或字符插入当前光标位置
 - `lv_ta_add_char(ta, 'c');`
 - `lv_ta_add_text(ta, "insret this text");`
- `lv_ta_set_text(ta, "New text")`更改整个文本。
- 要删除当前光标位置左侧的字符，使用 `lv_ta_del()`
- 可以像 `lv_ta_set_cursor_pos(ta, 10)`一样直接修改光标位置，也可以单步执行：
 - `lv_ta_cursor_right(ta)`
 - `lv_ta_cursor_left(ta)`
 - `lv_ta_cursor_up(ta)`
 - `lv_ta_cursor_down(ta)`
- 有几种游标类型。 您可以使用以下命令设置： `lv_ta_set_cursor_type(ta, LV_CURSOR_...)`
 - LV_CURSOR_NONE
 - LV_CURSOR_LINE
 - LV_CURSOR_BLOCK
 - LV_CURSOR_OUTLINE
 - LV_CURSOR_UNDERLINE
- 你可以'OR'LV_CURSOR_HIDDEN 到任何类型来隐藏光标。
- 文本区域可以配置为内衬 `lv_ta_set_one_line(ta, true)`
- 文本区域支持密码模式。 它可以使 `lv_ta_set_pwd_mode(ta, true)`启用。

2.15.1 相关宏定义

光标样式(lv_cursor_type_t)	
LV_CURSOR_NONE	无光标
LV_CURSOR_LINE	
LV_CURSOR_BLOCK	
LV_CURSOR_OUTLINE	
LV_CURSOR_UNDERLINE	
LV_CURSOR_HIDDEN	

2.15.2 API 函数

函数	描述
lv_ta_create ()	创建文本区域控件
lv_ta_add_char()	将字符插入当前光标位置
lv_ta_add_text()	将文本插入当前光标位置
lv_ta_del_char()	从当前光标位置删除左侧字符
lv_ta_set_text()	设置文本区域的文本
lv_ta_set_cursor_pos()	设置光标位置
lv_ta_set_cursor_type()	设置光标类型
lv_ta_set_pwd_mode()	启用/禁用密码模式
lv_ta_set_one_line()	将文本区域配置为一行或恢复正常
lv_ta_set_sb_mode()	将文本区域配置为一行或恢复正常
lv_ta_set_style()	设置文本区域的样式
lv_ta_get_text()	获取文本区域的文本
lv_ta_get_label()	获取文本区域的标签
lv_ta_get_cursor_pos()	获取字符索引中的当前光标位置
lv_ta_get_cursor_type()	获取光标类型
lv_ta_get_pwd_mode()	获取密码模式属性
lv_ta_get_one_line()	获取一行配置属性
lv_ta_get_sb_mode()	获取文本区域的滚动条模式
lv_ta_get_style()	获得文本区域的样式
lv_ta_cursor_right()	将光标向右移动一个字符
lv_ta_cursor_left()	将光标向左移动一个字符
lv_ta_cursor_down()	将光标向下移动一行
lv_ta_cursor_up()	将光标向上移动一行

lv_ta_create()

创建文本区域控件

lv_obj_t * lv_ta_create(lv_obj_t * par, lv_obj_t * copy)

参数

描述

par	指向控件的指针，它将是新文本区域的父控件
copy	指向窗口对象的指针，如果不是NULL，则将从该控件复制新控件

返回值：创建的文本区域指针

lv_ta_add_char()

将字符插入当前光标位置

void lv_ta_add_char(lv_obj_t * ta, char c)

参数	描述
ta	文本区域控件指针
c	要插入的新字符

返回值：无

lv_ta_add_text()

将文本插入当前光标位置

void lv_ta_add_text(lv_obj_t * ta, const char * txt)

参数	描述
ta	文本区域控件指针
txt	要插入的新字符串

返回值：无

lv_ta_del_char()

从当前光标位置删除左侧字符

void lv_ta_del_char(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：无

lv_ta_set_text()

设置文本区域的文本

void lv_ta_set_text(lv_obj_t * ta, const char * txt)

参数	描述
ta	文本区域控件指针
txt	指向文本的指针

lv_ta_set_cursor_pos()

设置光标位置

void lv_ta_set_cursor_pos(lv_obj_t * ta, int16_t pos)

参数	描述
ta	文本区域控件指针
pos	字符索引中的新光标位置 <0: 文本末尾的索引 LV_TA_CURSOR_LAST: 追踪最后

	一个字符
--	------

返回值：无

lv_ta_set_cursor_type()

设置光标类型

void lv_ta_set_cursor_type(lv_obj_t * ta, lv_cursor_type_t cur_type)

参数	描述
ta	文本区域控件指针
cur_type	见 lv_ta_cursor_type_t

返回值：无

lv_ta_set_pwd_mode()

启用/禁用密码模式

void lv_ta_set_pwd_mode(lv_obj_t * ta, bool pwd_en)

参数	描述
ta	文本区域控件指针
pwd_en	true: 启用, false: 禁用

返回值：无

lv_ta_set_one_line()

将文本区域配置为一行或恢复正常

void lv_ta_set_one_line(lv_obj_t * ta, bool en)

参数	描述
ta	文本区域控件指针
en	true: 一行, false: 正常

返回值：无

lv_ta_set_sb_mode()

设置文本区域的滚动条模式

static inline void lv_ta_set_sb_mode(lv_obj_t * ta, lv_sb_mode_t mode)

参数	描述
ta	文本区域控件指针
mode	见 lv_sb_mode_t

返回值：无

lv_ta_set_style()

设置文本区域的样式

void lv_ta_set_style(lv_obj_t * ta, lv_ta_style_t type, lv_style_t * style)

参数	描述
ta	文本区域控件指针
type	要设置的样式类型
style	样式指针

返回值：无

lv_ta_get_text()

获取文本区域的文本

const char * lv_ta_get_text(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：文本指针

lv_ta_get_label()

获取文本区域的标签

lv_obj_t * lv_ta_get_label(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：标签指针

lv_ta_get_cursor_pos()

获取字符索引中的当前光标位置

uint16_t lv_ta_get_cursor_pos(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：光标位置

lv_ta_get_cursor_type()

获取光标类型

lv_cursor_type_t lv_ta_get_cursor_type(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：光标类型

lv_ta_get_pwd_mode()

获取密码模式属性

bool lv_ta_get_pwd_mode(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：true：启用密码模式，false：禁用

lv_ta_get_one_line()

获取一行配置属性

bool lv_ta_get_one_line(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：true：启用一行配置，false：禁用

lv_ta_get_sb_mode()

获取文本区域的滚动条模式

static inline lv_sb_mode_t lv_ta_get_sb_mode(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：见 [lv_sb_mode_t](#)

lv_ta_get_style()

获得文本区域的样式

lv_style_t * lv_ta_get_style(lv_obj_t *ta, lv_ta_style_t type)

参数	描述
ta	文本区域控件指针
type	要火器的样式类型

返回值：样式指针

lv_ta_cursor_right()

将光标向右移动一个字符

void lv_ta_cursor_right(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：无

lv_ta_cursor_left()

将光标向左移动一个字符

void lv_ta_cursor_left(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：无

lv_ta_cursor_down()

将光标向下移动一行

void lv_ta_cursor_down(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：无

lv_ta_cursor_up()

将光标向上移动一行

void lv_ta_cursor_up(lv_obj_t * ta)

参数	描述
ta	文本区域控件指针

返回值：无

2.15.3 控件使用

2.16 Button



- 按钮可以通过回调函数（`lv_action_t` 函数指针）对用户按下，释放或长按进行反应。 您可以使用以下命令设置回调函数：`lv_btn_set_action(btn, ACTION_TYPE, callback_func)`可能的操作类型是：
 - `LV_BTN_ACTION_CLICK`: 按下（点击）按钮后释放
 - `LV_BTN_ACTION_PR`: 按下按钮
 - `LV_BTN_ACTION_LONG_PR`: 长按按钮
 - `LV_BTN_ACTION_LONG_PR_REPEAT`: 长按按钮，定期触发此操作
- 按钮可以处于五种可能状态之一：
 - `LV_BTN_STATE_REL` 已松开状态
 - `LV_BTN_STATE_PR` 按下状态
 - `LV_BTN_STATE_TGL_REL` 切换释放状态（开启松开）
 - `LV_BTN_STATE_TGL_PR` 切换按下状态（按下状态）
 - `LV_BTN_STATE_INA` 处于非活动状态
- 可以使用 `lv_btn_set_toggle(btn, true)` 将按钮配置为带切换按钮。 在这种情况下，在释放时，按钮进入切换释放状态。
- 您可以通过以下方式手动设置按钮的状态：`lv_btn_set_state(btn, LV_BTN_STATE_TGL_REL)`
- 按钮只能手动进入非活动状态（通过 `lv_btn_set_state()`）。 在非活动状态下，不会调用任何操作。
- 与 `Containers` 类似，按钮也有布局和自动调整：
 - `lv_btn_set_layout(btn, LV_LAYOUT_...)` 设置布局。 默认为 `LV_LAYOUT_CENTER`。 因此，如果添加标签，它将自动与中间对齐。

- `lv_btn_set_fit(btn, hor_en, ver_en)` 可以根据子系自动设置按钮宽度和/或高度。

2.16.1 相关宏定义

按键状态(lv_btn_state_t)	
<code>LV_BTN_STATE_REL</code>	已松开状态
<code>LV_BTN_STATE_PR</code>	按下状态
<code>LV_BTN_STATE_TGL_REL</code>	切换释放状态（开启状态）
<code>LV_BTN_STATE_TGL_PR</code>	切换按下状态（按下状态）
<code>LV_BTN_STATE_INA</code>	处于非活动状态

按键动作(lv_btn_action_t)	
<code>LV_BTN_ACTION_CLICK</code>	按下（点击）按钮后释放
<code>LV_BTN_ACTION_PR</code>	按下按钮
<code>LV_BTN_ACTION_LONG_PR</code>	长按按钮
<code>LV_BTN_ACTION_LONG_PR_REPEAT</code>	长按按钮，定期触发此操作

2.16.2 API 函数

函数	描述
<code>lv_btn_create()</code>	创建按钮控件
<code>lv_btn_set_toggle()</code>	启用切换状态
<code>lv_btn_set_state()</code>	设置按钮的状态
<code>lv_btn_toggle()</code>	切换按钮的状态(ON-> OFF, OFF-> ON)
<code>lv_btn_set_action()</code>	设置按钮事件发生时调用的函数
<code>lv_btn_set_layout()</code>	
<code>lv_btn_set_style()</code>	设置按钮的样式
<code>lv_btn_get_state()</code>	获取按钮的当前状态
<code>lv_btn_get_toggle()</code>	获取按钮的切换启用属性
<code>lv_btn_get_action()</code>	获取按钮的释放操作
<code>lv_btn_get_style()</code>	获得按钮的样式

`lv_btn_create()`

创建按钮控件

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

参数	描述
<code>par</code>	指向控件的指针，它将是新按钮的父控件

copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件
-------------	---------------------------------

返回值：创建的按钮指针

lv_btn_set_toggle()

启用切换状态

void lv_btn_set_toggle(lv_obj_t * btn, bool tgl)

参数	描述
btn	指向按钮对象的指针
tgl	true : 启用切换状态, false : 禁用

返回值：无

lv_btn_set_state()

设置按钮的状态

void lv_btn_set_state(lv_obj_t * btn, lv_btn_state_t state)

参数	描述
btn	指向按钮对象的指针
state	按钮状态, 见 lv_btn_state_t

返回值：无

lv_btn_toggle()

切换按钮的状态 (ON-> OFF, OFF-> ON)

void lv_btn_toggle(lv_obj_t * btn)

参数	描述
btn	指向按钮对象的指针

返回值：无

lv_btn_set_action()

设置按钮事件发生时调用的函数

void lv_btn_set_action(lv_obj_t * btn, lv_btn_action_t type, lv_action_t action)

参数	描述
btn	指向按钮对象的指针
type	事件类型 lv_btn_action_t (按, 释放, 长按, 长按重复)
action	事件函数

返回值：无

lv_btn_set_layout()

设置按钮布局

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

参数	描述
btn	指向按钮对象的指针
layout	见 lv_layout_t

返回值：无

lv_btn_set_fit()

启用水平或垂直适应

static inline void lv_btn_set_fit(lv_obj_t * btn, bool hor_en, bool ver_en)

参数	描述
btn	指向按钮对象的指针
hor_en	true:使能水平自适应
ver_en	true:使能垂直自适应

返回值：无

lv_btn_set_style()

设置按钮的样式

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

参数	描述
btn	指向按钮对象的指针
type	要设置的样式类型
style	样式指针

返回值：无

lv_btn_get_state()

获取按钮的当前状态

lv_btn_state_t lv_btn_get_state(lv_obj_t * btn)

参数	描述
btn	指向按钮对象的指针

返回值：按键状态，见 [lv_btn_state_t](#)

lv_btn_get_toggle()

获取按钮的切换启用属性

bool lv_btn_get_toggle(lv_obj_t * btn)

参数	描述
btn	指向按钮对象的指针

返回值：ture：启用切换，false：禁用

lv_btn_get_action()

获取按钮的释放操作

lv_action_t lv_btn_get_action(lv_obj_t * btn, lv_btn_action_t type)

参数	描述
btn	指向按钮对象的指针

返回值：指向释放动作功能的指针

lv_btn_get_layout()

获取按钮的布局

static inline lv_layout_t lv_btn_get_layout(lv_obj_t * btn)

参数	描述
----	----

btn	指向按钮对象的指针
------------	-----------

返回值：见 [lv_layout_t](#)

lv_btn_get_hor_fit()

获取按钮的水平自适应启用属性

static inline bool lv_btn_get_hor_fit(lv_obj_t * btn)

参数	描述
btn	指向按钮对象的指针

返回值：**true**:水平自适应使能; **false**:水平自适应失能

lv_btn_get_ver_fit()

获取按钮的垂直自适应启用属性

static inline bool lv_btn_get_ver_fit(lv_obj_t * btn)

参数	描述
btn	指向按钮对象的指针

返回值：**true**:垂直自适应使能; **false**:垂直自适应失能

lv_btn_get_style()

获得按钮的样式

lv_style_t * lv_btn_get_style(lv_obj_t * btn, lv_btn_style_t type)

参数	描述
btn	指向按钮对象的指针
type	要获取的样式类型

返回值：样式指针

2.16.3 控件使用

```

1. static lv_res_t width_inc(lv_obj_t * btn)
2. {
3.     lv_obj_set_width(btn, lv_obj_get_width(btn) + (10));
4.     return LV_RES_OK;
5. }
6.
7. static lv_res_t width_dec(lv_obj_t * btn)
8. {
9.     lv_obj_set_width(btn, lv_obj_get_width(btn) - (10));
10.    return LV_RES_OK;
11. }
12.
13. void lv_test_btn_1(void)
14. {
15.     /* 创建默认样式按钮 */

```

```

16.    lv_obj_t * btn1 = lv_btn_create(lv_scr_act(), NULL);    //创建 button 控
    件
17.
18.    /* 创建设置为翻转状态的默认按钮*/
19.    lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), NULL);    //创建 button 控
    件
20.    lv_obj_align(btn2, btn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);    //设置控件对
    齐方式
21.    lv_btn_set_state(btn2, LV_BTN_STATE_TGL_REL);    //设置按钮状态
22.
23.    /* 创建一个可以切换的按钮 */
24.    lv_obj_t * btn3 = lv_btn_create(lv_scr_act(), NULL);    //创建 button 控
    件
25.    lv_obj_align(btn3, btn2, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);    //设置控件对
    齐方式
26.    lv_btn_set_toggle(btn3, true);    //允许切换按钮的状态
27.
28.    /* 测试按键事件:
29.    *按下: 增加宽度, 释放: 减小宽度, 长按: 删除 */
30.    lv_obj_t * btn4 = lv_btn_create(lv_scr_act(), NULL);    //创建 button 控
    件
31.    lv_obj_align(btn4, btn1, LV_ALIGN_OUT_RIGHT_MID, 20, 0);    //设置控件对
    齐方式
32.    lv_btn_set_action(btn4, LV_BTN_ACTION_PR, width_inc);    //设置按键按下时事
    件
33.    lv_btn_set_action(btn4, LV_BTN_ACTION_CLICK, width_dec);    //设置按键释
    放时事件
34.    lv_btn_set_action(btn4, LV_BTN_ACTION_LONG_PR, lv_obj_del);    //设置按
    键长按时事件
35.
36.    /* 测试样式 */
37.    static lv_style_t style_rel;
38.    lv_style_copy(&style_rel, &lv_style_pretty);
39.    style_rel.body.main_color = LV_COLOR_ORANGE;
40.    style_rel.body.grad_color = LV_COLOR_BLACK;
41.    style_rel.body.border.color = LV_COLOR_RED;
42.    style_rel.body.shadow.color = LV_COLOR_MARRON;
43.    style_rel.body.shadow.width = 10;
44.
45.    static lv_style_t style_pr;
46.    lv_style_copy(&style_pr, &lv_style_pretty);
47.    style_pr.body.empty = 1;
48.    style_pr.body.border.color = LV_COLOR_RED;
49.    style_pr.body.border.width = 4;

```

```
50.
51.
52.     static lv_style_t style_tpr;
53.     lv_style_copy(&style_tpr, &lv_style_pretty);
54.     style_tpr.body.empty = 1;
55.     style_tpr.body.border.color = LV_COLOR_RED;
56.     style_tpr.body.border.width = 4;
57.
58.     static lv_style_t style_ina;
59.     lv_style_copy(&style_ina, &lv_style_pretty);
60.     style_ina.body.main_color = LV_COLOR_SILVER;
61.     style_ina.body.grad_color = LV_COLOR_GRAY;
62.     style_ina.body.border.color = LV_COLOR_RED;
63.
64.     /*创建修改样式的按钮*/
65.     lv_obj_t * btn5 = lv_btn_create(lv_scr_act(), btn4);    //创建 button 控
        件
66.     lv_obj_align(btn5, btn4, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);    //设置控件对
        齐方式
67.     lv_btn_set_style(btn5, LV_BTN_STYLE_REL, &style_rel);    //设置控件样式
68.     lv_btn_set_style(btn5, LV_BTN_STYLE_PR, &style_pr);
69.     lv_btn_set_style(btn5, LV_BTN_STYLE_TGL_PR, &style_tpr);
70.     lv_btn_set_style(btn5, LV_BTN_STYLE_INA, &style_ina);
71.     lv_btn_set_toggle(btn5, true);    //允许切换按钮的状态
72.
73.     /* Test style copy and inactive state*/
74.     lv_obj_t * btn6 = lv_btn_create(lv_scr_act(), btn5);    //创建 button 控
        件
75.     lv_obj_align(btn6, btn5, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);    //设置控件对
        齐方式
76.     lv_btn_set_state(btn6, LV_BTN_STATE_INA);    //设置按键非活动状态
77.
78.     /*测试水平自适应*/
79.     lv_obj_t * btn7 = lv_btn_create(lv_scr_act(), NULL);    //创建 button 控
        件
80.     lv_btn_set_fit(btn7, true, false);    //设置水平自适应
81.     lv_obj_t * label = lv_label_create(btn7, NULL);    //创建 label 控件
82.     lv_label_set_text(label, "A quite long text");    //设置 label 文本
83.     label = lv_label_create(btn7, NULL);    //创建 label 控件
84.     lv_label_set_text(label, "Short text");    //设置 label 文本
85.     lv_obj_align(btn7, btn4, LV_ALIGN_OUT_RIGHT_TOP, 20, 0);    //设置控件对
        齐方式
86.
87. }
```

运行效果



2.17 Button Matrix



- 按钮矩阵对象可以根据描述符字符串数组显示多个按钮，称为 `map`。您可以使用 `lv_btnm_set_map(btnm, my_map)` 指定 `map`
- `map` 的声明栗子: `const char * map[] = {"btn1", "btn2", "btn3", ""}`。请注意，最后一个元素必须是空字符串！
- 字符串的第一个字符可以是控制字符，用于指定一些属性：
 - **bit 7..6** 始终为 0b10，以区分控制字节和文本字符
 - **bit 5** 非活动按钮
 - **bit 4** 隐藏的按钮
 - **bit 3** 没有长按按钮
 - **bit 2..0** 相对宽度与同一行中的按钮相比。[1..7]

建议将控制字节指定为八进制数。例如 `"\213button"`。八进制数始终以 2（位 7..6）开头，中间部分是属性（位 5..3），最后一部分是宽度（位 2..0）。因此，该示例描述了一个 3 单位宽的隐藏按钮。

- 在 `map` 中使用 `"\n"` 进行换行: `{"btn1", "btn2", "\n", "btn3", ""}`。每行重新计算按钮的宽度。
- `lv_btnm_set_action(btnm, btnm_action)` 指定释放按钮时要调用的操作。

- 您可以启用按钮在单击时切换。 同时可以有一个切换按钮。
`lv_btnm_set_toggle(btnm, true, id)` 启用切换并使 `id` 个按钮切换

2.17.1 相关宏定义

矩阵按键样式(lv_btnm_style_t)	
LV_BTNM_STYLE_BG	背景样式。 使用所有 <code>style.body</code> 属性，包括填充 Default: <code>lv_style_pretty</code>
LV_BTNM_STYLE_BTN_REL	释放按钮的样式。 Default: <code>lv_style_btn_rel</code>
LV_BTNM_STYLE_BTN_PR	按下按钮的样式。 Default: <code>lv_style_btn_pr</code>
LV_BTNM_STYLE_BTN_TGL_REL	切换释放按钮的样式。 Default: <code>lv_style_btn_tgl_rel</code>
LV_BTNM_STYLE_BTN_TGL_PR	切换按钮的样式。 Default: <code>lv_style_btn_tgl_pr</code>
LV_BTNM_STYLE_BTN_INA	非活动按钮的样式。 Default: <code>lv_style_btn_ina</code>

2.17.2 API 函数

函数	描述
lv_btnm_create ()	创建矩阵按钮控件
lv_btnm_set_map()	设置新 map.将根据 map 创建/删除按钮
lv_btnm_set_action()	为按钮设置一个新的回调函数
lv_btnm_set_toggle()	启用或禁用按钮切换
lv_btnm_set_style()	设置按钮矩阵的样式
lv_btnm_get_map()	获取按钮矩阵的当前 map
lv_btnm_get_action()	获取按钮矩阵上按钮的回调函数
lv_btnm_get_toggled()	获取切换按钮
lv_btnm_get_style()	获得按钮矩阵的样式

`lv_btnm_create()`

创建矩阵按钮控件

`lv_obj_t * lv_btnm_create(lv_obj_t * par, lv_obj_t * copy)`

参数	描述
par	指向控件的指针，它将是新矩阵按钮的父控件
copy	指向窗口对象的指针，如果不是

	NULL，则将从该控件复制新控件
--	------------------

返回值：创建的矩阵按钮指针

lv_btm_set_map()

设置新 map.将根据 map 创建/删除按钮

void lv_btm_set_map(lv_obj_t * btm, const char ** map)

参数	描述
btm	指向按钮矩阵对象的指针
copy	<p>指针一个字符串数组。 最后一个字符串必须是: ""</p> <p>使用 “\n” 开始新行</p> <p>第一个字节可以是控制数据:</p> <ul style="list-style-type: none"> - bit 7: 总是 1 - bit 6: 总是 0 - bit 5: 不活动 (已禁用) (\24x) - bit 4: 不重复 (不长按) (\22x) - bit 3: 隐藏 (\21x) - bit 2..0: 按钮相对宽度 <p>举个栗(实际使用八进制数): "\224abc": "abc" 文本, 宽度为 4, 无长按</p>

返回值：无

lv_btm_set_action()

为按钮设置一个新的回调函数 (当按钮被释放时将调用它)

void lv_btm_set_action(lv_obj_t * btm, lv_btm_action_t action)

参数	描述
btm	指向按钮矩阵对象的指针
action	指向回调函数的指针

返回值：无

lv_btm_set_toggle()

启用或禁用按钮切换

void lv_btm_set_toggle(lv_obj_t * btm, bool en, uint16_t id)

参数	描述
btm	指向按钮矩阵对象的指针
en	true: 启用切换; false: 禁用切换
id	当前切换按钮的索引 (如果'en'== false 则忽略)

返回值：无

lv_btm_set_style()

设置按钮矩阵的样式

void lv_btm_set_style(lv_obj_t * btm, lv_btm_style_t type, lv_style_t * style)

参数	描述
btnm	指向按钮矩阵对象的指针
type	要设置的样式类型
style	样式指针

返回值：无

lv_btnm_get_map()

获取按钮矩阵的当前 map

`const char ** lv_btnm_get_map(lv_obj_t * btnm)`

参数	描述
btnm	指向按钮矩阵对象的指针

返回值：当前 map

lv_btnm_get_action()

获取按钮矩阵上按钮的回调函数

`lv_btnm_action_t lv_btnm_get_action(lv_obj_t * btnm)`

参数	描述
btnm	指向按钮矩阵对象的指针

指向回调函数的指针

lv_btnm_get_toggled()

获取切换按钮

`uint16_t lv_btnm_get_toggled(lv_obj_t * btnm)`

参数	描述
btnm	指向按钮矩阵对象的指针

返回值：当前切换按钮的索引（0：如果未设置）

lv_btnm_get_style()

获得按钮矩阵的样式

`lv_style_t * lv_btnm_get_style(lv_obj_t * btnm, lv_btnm_style_t type)`

参数	描述
btnm	指向按钮矩阵对象的指针
type	要获取的样式类型

返回值：样式指针

2.17.3 控件使用

```

1. static const char * btnm_map[] = {"One line", "\n", "\212", "\242Ina", "\204
   ü???", "\221éé", "\n", "\214", "\202Left", ""};
2.
3. static lv_res_t btnm_action(lv_obj_t * btnm, const char * txt)
4. {

```

```
5.     sysprintf("Key pressed: %s\n", txt);
6.
7.     return LV_RES_OK;
8. }
9.
10. void lv_test_btm_1(void)
11. {
12.     /* 创建默认控件
13.      * 带有默认按钮的按钮矩阵 */
14.     lv_obj_t * btnm1 = lv_btm_create(lv_scr_act(), NULL); //创建
        Button Matrix 控件
15.
16.     /* 测试 map, 大小和位置。 还尝试一些功能.
17.      * 带有默认按钮的按钮矩阵. */
18.     static lv_style_t rel;
19.     lv_style_copy(&rel, &lv_style_btn_tgl_rel);
20.     rel.body.main_color = LV_COLOR_RED;
21.     rel.body.grad_color = LV_COLOR_BLACK;
22.     rel.text.color = LV_COLOR_YELLOW;
23.
24.     static lv_style_t pr;
25.     lv_style_copy(&pr, &lv_style_btn_tgl_rel);
26.     pr.body.main_color = LV_COLOR_ORANGE;
27.     pr.body.grad_color = LV_COLOR_BLACK;
28.     pr.text.color = LV_COLOR_WHITE;
29.
30.
31.     lv_obj_t * btnm2 = lv_btm_create(lv_scr_act(), NULL); //创建
        Button Matrix 控件
32.     lv_btm_set_map(btnm2, btnm_map); //设置 map
33.     lv_obj_set_size(btnm2, LV_HOR_RES / 2, LV_VER_RES / 3); //设置控件尺寸
34.     lv_obj_align(btnm2, btnm1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20); //设置控
        件对齐方式
35.     lv_btm_set_toggle(btnm2, true, 2); //启用 2 号按钮切换
36.     lv_btm_set_action(btnm2, btnm_action); //设置按钮事件回调函数
37.     lv_btm_set_style(btnm2, LV_BTMM_STYLE_BTN_REL, &rel); //设置控件样式
38.     lv_btm_set_style(btnm2, LV_BTMM_STYLE_BTN_PR, &pr);
39. }
```

运行效果



2.18 Keyboard



- 正如它的名字所示，**Keyboard** 对象提供了一个键盘来写文本。您可以为键盘指定文本区域以将单击的字符放在那里。要分配文本区域，请使用 `lv_kb_set_ta(kb, ta)`。
- 键盘包含 *Ok* 和 *Hide* 按钮。可以通过 `lv_kb_set_ok_action(kb, action)` 和 `lv_kb_set_hide_action(kb, action)` 指定 *ok* 和 *hide* 操作，以将回调添加到 *Ok/Hide* 点击。如果未指定任何操作，则按钮将删除键盘。
- 指定的文本区域的光标可以由键盘管理：当键盘被分配时，前一个文本区域的光标将被隐藏，将显示新的光标。单击 *Ok* 或 *Hide* 也将隐藏光标。光标管理器功能由 `lv_kb_set_cursor_manage(kb, true)` 启用。默认值不是管理。
- 键盘有两种模式：
 - `LV_KB_MODE_TEXT`: 显示字母，数字和特殊字符
 - `LV_KB_MODE_NUM`: 显示数字，+/- 符号和点
- 要设置模式，请使用 `lv_kb_set_mode(kb, mode)`。默认值为 `LV_KB_MODE_TEXT`
 - 您可以使用 `lv_kb_set_map(kb, map)` 为键盘指定新的 *map*（布局）。它的工作方式类似于按钮矩阵，因此控件字符可以添加到布局中设置按钮宽度和其他属性。请记住，使用以下关键字将与原始地图具有相同的效果：
`SYMBOL_OK`, `SYMBOL_CLOSE`, `SYMBOL_LEFT`, `SYMBOL_RIGHT`, `ABC`, `abc`, `Enter`, `Del`, `#1`, `+/-`

2.18.1 相关宏定义

键盘模式(lv_kb_mode_t)	
LV_KB_MODE_TEXT	显示字母，数字和特殊字符
LV_KB_MODE_NUM	显示数字，+ / - 符号和点
键盘样式(lv_kb_style_t)	
LV_KB_STYLE_BG	背景样式。 使用所有 style.body 属性，包括填充 Default: lv_style_pretty
LV_KB_STYLE_BTN_REL	释放按钮的样式。 Default: lv_style_btn_rel
LV_KB_STYLE_BTN_PR	按下按钮的样式。 Default: lv_style_btn_pr
LV_KB_STYLE_BTN_TGL_REL	切换释放按钮的样式。 Default: lv_style_btn_tgl_rel
LV_KB_STYLE_BTN_TGL_PR	切换按钮的样式。 Default: lv_style_btn_tgl_pr
LV_KB_STYLE_BTN_INA	非活动按钮的样式。 Default: lv_style_btn_ina

2.18.2 API 函数

函数	描述
lv_kb_create ()	创建键盘控件
lv_kb_set_ta()	将文本区域分配给键盘,按下的字符将放在那里
lv_kb_set_mode()	设置新的模式
lv_kb_set_cursor_manage()	自动隐藏或显示文本区域的光标
lv_kb_set_ok_action()	设置按下"Ok"按钮时的回调函数
lv_kb_set_hide_action()	设置按下"Hide"按钮时的回调函数
lv_kb_set_map()	为键盘设置新 map
lv_kb_set_style()	设置键盘的样式
lv_kb_get_ta()	获取 Text Area 控件指针
lv_kb_get_mode()	获取键盘的模式
lv_kb_get_cursor_manage()	获取当前光标管理模式
lv_kb_get_ok_action()	获取按下"Ok"键时的回调函数
lv_kb_get_hide_action()	获取按下"Hide"键时的回调函数
lv_kb_get_style()	获取键盘样式

lv_kb_create()
创建键盘控件

lv_obj_t * lv_kb_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新键盘的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的键盘指针

lv_kb_set_ta()

将文本区域分配给键盘。 按下的字符将放在那里

void lv_kb_set_ta(lv_obj_t * kb, lv_obj_t * ta)

参数	描述
kb	指向键盘对象的指针
ta	指向要在那里写入的 Text Area 控件的指针

返回值：无

lv_kb_set_mode()

设置新的模式（文本或数字 map）

void lv_kb_set_mode(lv_obj_t * kb, lv_kb_mode_t mode)

参数	描述
kb	指向键盘对象的指针
mode	见 lv_kb_mode_t

返回值：无

lv_kb_set_cursor_manage()

自动隐藏或显示文本区域的光标

void lv_kb_set_cursor_manage(lv_obj_t * kb, bool en)

参数	描述
kb	指向键盘对象的指针
en	true: 在当前文本区域显示光标, false: 隐藏光标

返回值：无

lv_kb_set_ok_action()

设置按下“Ok”按钮时的回调函数

void lv_kb_set_ok_action(lv_obj_t * kb, lv_action_t action)

参数	描述
kb	指向键盘对象的指针
action	回调函数

返回值：无

lv_kb_set_hide_action()

设置按下“Hide”按钮时的回调函数

void lv_kb_set_hide_action(lv_obj_t * kb, lv_action_t action)

参数	描述
kb	指向键盘对象的指针
action	回调函数

返回值：无

lv_kb_set_map()

为键盘设置新 map

static inline void lv_kb_set_map(lv_obj_t *kb, const char ** map)

参数	描述
kb	指向键盘对象的指针
map	指向字符串数组的指针，用于描述 map. 见 lv_btnm_set_map() 了解更多信息

返回值：无

lv_kb_set_style()

设置键盘的样式

void lv_kb_set_style(lv_obj_t *kb, lv_kb_style_t type, lv_style_t *style)

参数	描述
kb	指向键盘对象的指针
type	要设置的样式类型，见 lv_kb_style_t
style	样式指针

返回值：无

lv_kb_get_ta()

获取 Text Area 控件指针

lv_obj_t * lv_kb_get_ta(lv_obj_t * kb)

参数	描述
kb	指向键盘对象的指针

返回值：指向指定的 Text Area 对象的指针

lv_kb_get_mode()

获取键盘的模式

lv_kb_mode_t lv_kb_get_mode(lv_obj_t * kb)

参数	描述
kb	指向键盘对象的指针

返回值：当前模式，见 [lv_kb_mode_t](#)

lv_kb_get_cursor_manage()

获取当前光标管理模式

bool lv_kb_get_cursor_manage(lv_obj_t * kb)

参数	描述
kb	指向键盘对象的指针

返回值: **true**: 在当前文本区域显示光标, **false**: 隐藏光标

lv_kb_get_ok_action()

获取按下"Ok"键时的回调函数

lv_action_t lv_kb_get_ok_action(lv_obj_t * kb)

参数	描述
kb	指向键盘对象的指针

返回值: 回调函数

lv_kb_get_hide_action()

获取按下"Hide"键时的回调函数

lv_action_t lv_kb_get_hide_action(lv_obj_t * kb)

参数	描述
kb	指向键盘对象的指针

返回值: 回调函数

lv_kb_get_style()

获取键盘样式

lv_style_t * lv_kb_get_style(lv_obj_t * kb, lv_kb_style_t type)

参数	描述
kb	指向键盘对象的指针
type	见 lv_kb_style_t

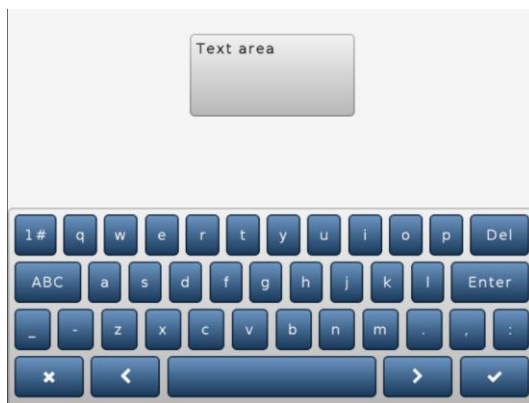
返回值: 样式指针

2.18.3 控件使用

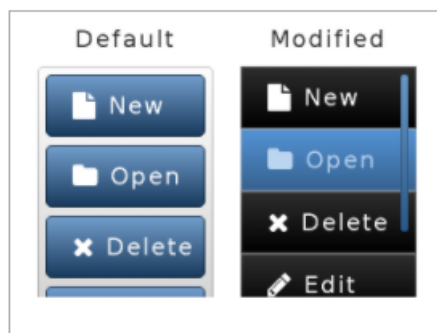
```

1. lv_obj_t *ta = lv_ta_create(lv_scr_act(), NULL);    //创建 text area 控件
2. lv_obj_align(ta, NULL, LV_ALIGN_IN_TOP_MID, 0, 30); //设置控件对齐方式
3.
4. /* 创建默认样式*/
5. lv_obj_t *kb1 = lv_kb_create(lv_scr_act(), NULL);  //设置 keyboard 控件
6. lv_obj_align(kb1, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, 0); //设置控件对齐方式
7. lv_kb_set_ta(kb1, ta); //将文本区域分配给键盘
    
```

运行效果



2.19 List



- 列表是由背景页面和按钮构建的。按钮包含可选的图标式图像（也可以是符号）和标签。当列表变得足够长时，它可以滚动。根据对象宽度将按钮的宽度设置为最大。按钮的高度根据内容自动调整（内容高度+`style.body.padding.ver`）
- 您可以使用 `lv_list_add(list, "U:/img", "Text", rel_action)` 或符号图标 `lv_list_add(list, SYMBOL_EDIT, "Edit text")` 添加新的列表元素。如果您不想添加图像，请使用 "" 作为文件名。该函数返回一个指向已创建按钮的指针，以允许进一步配置
- 您可以使用 `lv_list_get_btn_label(list_btn)` 和 `lv_list_get_btn_img(list_btn)` 来获取标签和列表按钮的图像
- 在按钮的释放操作中，您可以使用 `lv_list_get_btn_text(button)` 获取按钮的文本。它有助于识别已发布的列表元素
- 要删除列表元素，只需对 `lv_obj_del()` 的返回值使用 `lv_list_add()`
- 您可以使用 `lv_list_up(list)` 和 `lv_list_down(list)` 在列表中手动导航
- 您可以使用 `lv_list_focus` 直接关注按钮 `lv_list_focus(btn, anim_en)`
- 上/下/焦点移动的动画时间可以通过以下方式设置：
`lv_list_set_anim_time(list, anim_time)`。零动画时间意味着不是动画

2.19.1 相关宏定义

键盘模式(`lv_list_style_t`)

LV_LIST_STYLE_BG	背景样式
LV_LIST_STYLE_SCRL	可滚动部分样式
LV_LIST_STYLE_SB	滚动条的样式
LV_LIST_STYLE_BTN_REL	释放按钮的样式。Default: lv_style_btn_rel
LV_LIST_STYLE_BTN_PR	按下按钮的样式。Default: lv_style_btn_pr
LV_LIST_STYLE_BTN_TGL_REL	切换释放按钮的样式。Default: lv_style_btn_tgl_rel
LV_LIST_STYLE_BTN_TGL_PR	切换按钮的样式。Default: lv_style_btn_tgl_pr
LV_LIST_STYLE_BTN_INA	非活动按钮的样式。Default: lv_style_btn_ina

2.19.2 API 函数

函数	描述
lv_list_create()	创建列表控件
lv_list_add()	将列表元素添加到列表中
lv_list_set_anim_time()	在'list_up()' 'list_down()' 'list_focus()' 上设置滚动动画持续时间
lv_list_set_sb_mode()	设置列表的滚动条模式
lv_list_set_style()	设置列表样式
lv_list_get_btn_text()	获取列表元素的文本
lv_list_get_btn_label()	从列表元素中获取标签对象
lv_list_get_btn_img()	从列表元素中获取图像对象
lv_list_get_anim_time()	获取滚动动画时长
lv_list_get_sb_mode()	获取列表的滚动条模式
lv_list_get_style()	获得列表样式
lv_list_up()	将列表元素向上移动一个
lv_list_down()	将列表元素向下移动一个
lv_list_focus()	聚焦列表按钮

lv_list_create()

创建列表控件

lv_obj_t * lv_list_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新列表的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的列表指针

lv_list_add()

将列表元素添加到列表中

lv_obj_t * lv_list_add(lv_obj_t * list, const void * img_src, const char * txt, lv_action_t rel_action)

参数	描述
list	指向列表对象的指针
img_src	文本前的图像文件名（如果未使用，则为 NULL ）
txt	

返回值：创建的列表指针

lv_list_set_anim_time()

在'list_up()'list_down()'list_focus()'上设置滚动动画持续时间

void lv_list_set_anim_time(lv_obj_t *list, uint16_t anim_time)

参数	描述
list	指向列表对象的指针
anim_time	动画的持续时间(ms)

返回值：无

lv_list_set_sb_mode()

设置列表的滚动条模式

static inline void lv_list_set_sb_mode(lv_obj_t * list, lv_sb_mode_t mode)

参数	描述
list	指向列表对象的指针
mode	见 lv_sb_mode_t

返回值：无

lv_list_set_style()

设置列表样式

void lv_list_set_style(lv_obj_t *list, lv_list_style_t type, lv_style_t *style)

参数	描述
list	指向列表对象的指针
type	要设置的样式类型
style	样式指针

返回值：无

lv_list_get_btn_text()

获取列表元素的文本

const char * lv_list_get_btn_text(lv_obj_t * btn)

参数	描述
list	指向列表对象的指针

返回值：文本指针

lv_list_get_btn_label()

从列表元素中获取标签对象

lv_obj_t * lv_list_get_btn_label(lv_obj_t * btn)

参数	描述
list	指向列表对象的指针

返回值：指向 **list** 元素中标签的指针，如果未找到则为 **NULL**

lv_list_get_btn_img()

从列表元素中获取图像对象

lv_obj_t * lv_list_get_btn_img(lv_obj_t * btn)

参数	描述
list	指向列表对象的指针

返回值：指向列表元素中图像的指针，如果未找到则为 **NULL**

lv_list_get_anim_time()

获取滚动动画时长

uint16_t lv_list_get_anim_time(lv_obj_t * list)

参数	描述
list	指向列表对象的指针

返回值：动画持续时间[ms]

lv_list_get_sb_mode()

获取列表的滚动条模式

static inline lv_sb_mode_t lv_list_get_sb_mode(lv_obj_t * list)

参数	描述
list	指向列表对象的指针

返回值：见 [lv_sb_mode_t](#)

lv_list_get_style()

获得列表样式

lv_style_t * lv_list_get_style(lv_obj_t * list, lv_list_style_t type)

参数	描述
list	指向列表对象的指针
type	要获取的样式类型，见 lv_list_style_t

返回值：样式指针

lv_list_up()

将列表元素向上移动一个

void lv_list_up(lv_obj_t * list)

参数	描述
list	指向列表对象的指针

返回值：无

lv_list_down()

将列表元素向下移动一个

void lv_list_down(lv_obj_t * list)

参数	描述
list	指向列表对象的指针

返回值：无

lv_list_focus()

聚焦列表按钮。 它确保按钮在列表中可见

void lv_list_focus(lv_obj_t *btn, bool anim_en)

参数	描述
list	指向列表对象的指针
anim_en	true: 滚动动画, false: 无动画

返回值：无

2.19.3 控件使用

```

1. static lv_obj_t *list1;
2. static lv_obj_t *list2;
3. static lv_obj_t *list3;
4. static lv_obj_t *list4;
5.
6. extern const lv_img_t img_flower_icon;      /*Comes from lv_test_img*/
7.
8. static lv_res_t list_move(lv_obj_t *btn)
9. {
10.     if(lv_obj_get_free_num(btn) == 0) {
11.         lv_list_up(list1);
12.         lv_list_up(list2);
13.         lv_list_up(list3);
14.         lv_list_up(list4);
15.     } else {
16.         lv_list_down(list1);
17.         lv_list_down(list2);
18.         lv_list_down(list3);
19.         lv_list_down(list4);
20.     }
21.     return LV_RES_OK;
22. }
23.
24. void lv_test_list_1(void)
25. {

```

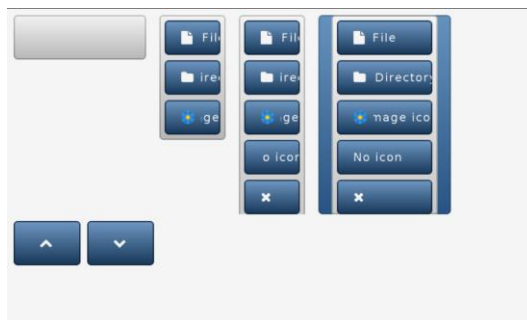
```
26.    /* 创建默认控件*/
27.    list1 = lv_list_create(lv_scr_act(), NULL); //创建 list 控件
28.    lv_obj_set_pos(list1, 10, 10); //设置控件显示坐标
29.
30.    list2 = lv_list_create(lv_scr_act(), NULL); //创建 list 控件
31.    lv_obj_align(list2, list1, LV_ALIGN_OUT_RIGHT_TOP, 20, 0); //设置控件对
    齐方式
32.    lv_list_add(list2, SYMBOL_FILE, "File", NULL); //添加含库自带图标的元素
33.    lv_list_add(list2, SYMBOL_DIRECTORY, "Directory", NULL); //添加含库自
    带图标的元素
34.    lv_list_add(list2, &img_flower_icon, "Image icon", NULL); //添加含自定义
    图标的元素
35.    lv_obj_set_width(list2, 100); //设置控件宽度
36.
37.    list3 = lv_list_create(lv_scr_act(), list2); //创建 list 控件
38.    lv_obj_align(list3, list2, LV_ALIGN_OUT_RIGHT_TOP, 20, 0); //设置控件对
    齐方式
39.    lv_list_add(list3, NULL, "No icon", NULL); ////添加不带图标的元素
40.    lv_list_add(list3, SYMBOL_CLOSE, "", NULL); //添加含库自带图标的元素
41.    lv_list_add(list3, SYMBOL_UP, "Up", NULL); //添加含库自带图标的元素
42.    lv_list_add(list3, SYMBOL_DOWN, "Down", NULL); //添加含库自带图标的元素
43.
44.    static lv_style_t sb;
45.    static lv_style_t bg;
46.    lv_style_copy(&sb, &lv_style_pretty_color);
47.    lv_style_copy(&bg, &lv_style_pretty_color);
48.    sb.body.padding.hor = -10;
49.    sb.body.padding.inner = 10;
50.
51.    bg.body.padding.hor = 20;
52.
53.    list4 = lv_list_create(lv_scr_act(), list3); //创建 list 控件
54.    lv_list_set_style(list4, LV_LIST_STYLE_BG, &bg); //设置控件样式
55.    lv_list_set_style(list4, LV_LIST_STYLE_SB, &sb);
56.    lv_obj_align(list4, list3, LV_ALIGN_OUT_RIGHT_TOP, 20, 0); //设置控件对
    齐方式
57.    lv_obj_set_width(list4, 200); //设置控件宽度
58.
59.    /*添加 list up/down 按钮*/
60.    lv_obj_t *btn_up = lv_btn_create(lv_scr_act(), NULL); //创建 button 控
    件
61.    lv_obj_align(btn_up, list1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 10); //设置控
    件对齐方式
```

```

62.    lv_btn_set_action(btn_up, LV_BTN_ACTION_CLICK, list_move); //设置按钮触
      发的回调函数
63.    lv_obj_set_free_num(btn_up, 0); //为对象设置应用程序的编号
64.    lv_obj_t *label = lv_label_create(btn_up, NULL); //在按钮上创建 label 控
      件
65.    lv_label_set_text(label, SYMBOL_UP); //设置按钮上的 label 图标
66.
67.    lv_obj_t *btn_down = lv_btn_create(lv_scr_act(), btn_up); //创建 button
      控件
68.    lv_obj_align(btn_down, btn_up, LV_ALIGN_OUT_RIGHT_MID, 10, 0); //设置控
      件对齐方式
69.    lv_obj_set_free_num(btn_down, 1); //为对象设置应用程序的编号
70.    label = lv_label_create(btn_down, NULL); //在按钮上创建 label 控件
71.    lv_label_set_text(label, SYMBOL_DOWN); //设置按钮上的 label 图标
72.
73. }
74.
75. static lv_res_t ddlist_action(lv_obj_t * ddlist)
76. {
77.
78.     char buf[64];
79.     lv_ddlist_get_selected_str(ddlist, buf);
80.     sysprintf("New option selected on a drop down list: %s\n", buf);
81.
82.
83.     return LV_RES_OK;
84. }

```

运行效果



2.20 Drop Down List



- 下拉列表允许您简单地从更多选项中选择一项。下拉列表默认关闭，显示当前选定的文本。如果单击它，将打开此列表并显示所有选项。
- 这些选项作为带有 `lv_ddlist_set_options(ddlist, options)` 的字符串传递给下拉列表。选项应以 `\n` 分隔。例如: `"First\nSecond\nThird"`
- 您可以使用 `lv_ddlist_set_selected(ddlist, id)` 手动选择一个选项，其中 `id` 是选项的索引。
- 可以使用 `lv_ddlist_set_action(ddlist, my_action)` 指定回调函数，以便在选择新选项时进行调用。
- 默认情况下，列表的高度会自动调整以显示所有选项。
`lv_ddlist_set_fix_height(ddlist, h)` 为打开的列表设置修复高度
- 宽度也会自动调整。要防止这种情况，请应用 `lv_ddlist_set_hor_fit(ddlist, false)` 并通过 `lv_obj_set_width(ddlist, width)` 手动设置宽度
- 与具有修复高度的页面类似，下拉列表支持各种滚动条显示模式。它可以通过 `lv_ddlist_set_sb_mode(ddlist, LV_SB_MODE_...)` 设置
- 下拉列表打开/关闭动画时间由 `lv_ddlist_set_anim_time(ddlist, anim_time)` 调整。零动画时间意味着没有动画

2.20.1 相关宏定义

下拉列表样式(<code>lv_ddlist_style_t</code>)	
<code>LV_DDLIST_STYLE_BG</code>	背景的风格。使用所有 <code>style.body</code> 属性。它用于 <code>style.text</code> 的标签样式 Default: <code>lv_style_pretty</code>
<code>LV_DDLIST_STYLE_SEL</code>	所选选项的样式。使用 <code>style.body</code> 属性。选定的选项将使用 <code>text.color</code> 重新着色。Default: <code>lv_style_plain_color</code>
<code>LV_DDLIST_STYLE_SB</code>	滚动条的样式。使用 <code>style.body</code> 属性。Default: <code>lv_style_plain_color</code>

2.20.2 API 函数

函数	描述
lv_ddlist_create ()	创建下拉列表对象
lv_ddlist_set_options()	在字符串的下拉列表中设置选项
lv_ddlist_set_selected()	设置所选选项
lv_ddlist_set_action()	设置要在选择新选项时调用的函数
lv_ddlist_set_fix_height()	设置下拉列表的修改高度
lv_ddlist_set_hor_fit()	启用或禁用内容的水平适合度
lv_ddlist_set_sb_mode()	设置下拉列表的滚动条模式
lv_ddlist_set_anim_time()	设置打开/关闭动画时间
lv_ddlist_set_style()	设置下拉列表的样式
lv_ddlist_get_options()	获取下拉列表的选项
lv_ddlist_get_selected()	获取所选选项
lv_ddlist_get_selected_str()	将当前选定的选项作为字符串
lv_ddlist_get_action()	获取“选项选择”回调函数
lv_ddlist_get_fix_height()	获取修复高度值
lv_ddlist_get_sb_mode()	获取下拉列表的滚动条模式
lv_ddlist_get_anim_time()	获取打开/关闭动画时间
lv_ddlist_get_style()	获取下拉列表的样式
lv_ddlist_open()	使用或不使用动画打开下拉列表
lv_ddlist_close()	关闭（折叠）下拉列表

lv_ddlist_create()

创建下拉列表对象

lv_obj_t * lv_ddlist_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新下拉列表的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的下拉列表指针

lv_ddlist_set_options()

在字符串的下拉列表中设置选项

void lv_ddlist_set_options(lv_obj_t * ddlist, const char * options)

参数	描述
ddlist	指向下拉列表对象的指针
options	带有'\n'分隔选项的字符串。 例如。 “One\nTwo\nThree”

返回值：无

lv_ddlist_set_selected()

设置所选选项

void lv_ddlist_set_selected(lv_obj_t * ddlist, uint16_t sel_opt)

参数	描述
ddlist	指向下拉列表对象的指针
sel_opt	所选选项的 ID (0 ...选项数 - 1) ;

返回值: 无

lv_ddlist_set_action()

设置要在选择新选项时调用的函数

void lv_ddlist_set_action(lv_obj_t * ddlist, lv_action_t action)

参数	描述
ddlist	指向下拉列表对象的指针
action	指向回调函数的指针

返回值: 无

lv_ddlist_set_fix_height()

设置下拉列表的修复高度

void lv_ddlist_set_fix_height(lv_obj_t * ddlist, lv_coord_t h)

参数	描述
ddlist	指向下拉列表对象的指针
h	列表打开时的高度 (0: 自动大小)

返回值: 无

lv_ddlist_set_hor_fit()

启用或禁用内容的水平适合度

void lv_ddlist_set_hor_fit(lv_obj_t * ddlist, bool fit_en)

参数	描述
ddlist	指向下拉列表对象的指针
fit_en	true: 启用自动调整; false: 禁用自动适合

返回值: 无

lv_ddlist_set_sb_mode()

设置下拉列表的滚动条模式

static inline void lv_ddlist_set_sb_mode(lv_obj_t * ddlist, lv_sb_mode_t mode)

参数	描述
ddlist	指向下拉列表对象的指针
mode	来自 lv_sb_mode_t 枚举的新模式

返回值: 无

lv_ddlist_set_anim_time()

设置打开/关闭动画时间

void lv_ddlist_set_anim_time(lv_obj_t * ddlist, uint16_t anim_time)

参数	描述
ddlist	指向下拉列表对象的指针
anim_time	打开/关闭动画时间[ms]

返回值：无

lv_ddlist_set_style()

设置下拉列表的样式

void lv_ddlist_set_style(lv_obj_t *ddlist, lv_ddlist_style_t type, lv_style_t *style)

参数	描述
ddlist	指向下拉列表对象的指针
type	要设置的样式类型，见 lv_ddlist_style_t
style	样式指针

返回值：无

lv_ddlist_get_options()

获取下拉列表的选项

const char * lv_ddlist_get_options(lv_obj_t * ddlist)

参数	描述
ddlist	指向下拉列表对象的指针

返回值：由'\n'分隔的选项（例如 “Option1 \nOption2 \nOption3”）

lv_ddlist_get_selected()

获取所选选项

uint16_t lv_ddlist_get_selected(lv_obj_t * ddlist)

参数	描述
ddlist	指向下拉列表对象的指针

返回值：由'\n'分隔的选项（例如 “Option1 \nOption2 \nOption3”）

所选选项的 ID（0 ...选项数 - 1）；

lv_ddlist_get_selected_str()

将当前选定的选项作为字符串

void lv_ddlist_get_selected_str(lv_obj_t * ddlist, char * buf)

参数	描述
ddlist	指向下拉列表对象的指针
buf	指向存储字符串的数组的指针

返回值：无

lv_ddlist_get_action()

获取“选项选择”回调函数

lv_action_t lv_ddlist_get_action(lv_obj_t * ddlist)

参数	描述
ddlist	指向下拉列表对象的指针

返回值：指向回调函数的指针

lv_ddlist_get_fix_height()

获取修复高度值

lv_coord_t lv_ddlist_get_fix_height(lv_obj_t * ddlist)

参数	描述
ddlist	指向下拉列表对象的指针

返回值：下拉列表打开时的高度（0：自动大小）

lv_ddlist_get_sb_mode()

获取下拉列表的滚动条模式

static inline lv_sb_mode_t lv_ddlist_get_sb_mode(lv_obj_t * ddlist)

参数	描述
ddlist	指向下拉列表对象的指针

返回值：滚动条模式来自 **lv_sb_mode_t** 枚举

lv_ddlist_get_anim_time()

获取打开/关闭动画时间

uint16_t lv_ddlist_get_anim_time(lv_obj_t * ddlist)

参数	描述
ddlist	指向下拉列表对象的指针

返回值：打开/关闭动画时间[ms]

lv_ddlist_get_style()

获取下拉列表的样式

lv_style_t * lv_ddlist_get_style(lv_obj_t * ddlist, lv_ddlist_style_t type)

参数	描述
ddlist	指向下拉列表对象的指针
type	要获取的样式类型，见 lv_ddlist_style_t

返回值：打开/关闭动画时间[ms]

lv_ddlist_open()

使用或不使用动画打开下拉列表

void lv_ddlist_open(lv_obj_t * ddlist, bool anim_en)

参数	描述
ddlist	指向下拉列表对象的指针
anim_en	true: 使用动画; false: 不使用动画

返回值：无

lv_ddlist_close()

关闭（折叠）下拉列表

void lv_ddlist_close(lv_obj_t * ddlist, bool anim_en)

参数	描述
----	----

ddlist	指向下拉列表对象的指针
anim_en	true : 使用动画; false : 不使用动画

返回值: 无

2.20.3 控件使用

```

1.  /* 创建默认控件*/
2.  lv_obj_t * ddlist1 = lv_ddlist_create(lv_scr_act(), NULL);  //创建
    Drop Down List 控件
3.  lv_obj_set_pos(ddlist1, 10, 10);    //设置控件显示坐标
4.
5.  /* 创建一个包含多选项，修改高度和动画时间的下拉列表。
6.    * 默认情况下打开它而不显示动画并指定操作*/
7.  lv_obj_t * ddlist2 = lv_ddlist_create(lv_scr_act(), NULL);  //创建
    Drop Down List 控件
8.  lv_obj_align(ddlist2, ddlist1, LV_ALIGN_OUT_RIGHT_MID, 20, 0);  //设置控件对
    齐方式
9.  lv_ddlist_set_options(ddlist2, "First\nSecond\nThird\nForth\nFifth\nSixth");
    //在字符串的下拉列表中设置选项
10. lv_ddlist_set_fix_height(ddlist2, LV_DPI);  //设置下拉列表的修改高度
11. lv_ddlist_set_selected(ddlist2, 2); //设置所选选项
12. lv_ddlist_set_anim_time(ddlist2, 100);  //设置打开/关闭动画时间
13. lv_ddlist_open(ddlist2, false); //不使用动画打开下拉列表
14. lv_ddlist_set_hor_fit(ddlist2, false);  //禁用内容的水平适合度
15. lv_ddlist_set_action(ddlist2, ddlist_action);  //设置要在选择新选项时调用的函
    数
16. lv_obj_set_width(ddlist2, LV_DPI * 2);  //设置控件宽度
17.
18. /*复制之前的控件并设置样式*/
19. static lv_style_t ddlist3_style;
20. lv_style_copy(&ddlist3_style, &lv_style_pretty);
21. ddlist3_style.body.main_color = LV_COLOR_GRAY;
22. ddlist3_style.body.grad_color = LV_COLOR_BLACK;
23. ddlist3_style.body.padding.hor = 20;
24. ddlist3_style.body.padding.ver = 30;
25.
26. ddlist3_style.text.color = LV_COLOR_RED;
27. ddlist3_style.text.letter_space = 5;
28. ddlist3_style.text.line_space = 15;
29.
30. lv_obj_t * ddlist3 = lv_ddlist_create(lv_scr_act(), ddlist2);  //创建
    Drop Down List 控件
31. lv_obj_align(ddlist3, ddlist2, LV_ALIGN_OUT_RIGHT_TOP, 20, 0);  //设置控件对
    齐方式

```

```
32. lv_ddlist_set_style(ddlist3, LV_DDLIST_STYLE_BG, &ddlist3_style); //设置控件样式
33. lv_ddlist_set_style(ddlist3, LV_DDLIST_STYLE_SEL, &lv_style_plain_color);
```

运行效果



2.21 Roller



- **Roller** 允许您通过滚动简单地从更多选项选择一个选项。 其功能类似于下拉列表
- 选项以 `lv_roller_set_options(roller, options)` 的形式传递给 **Roller**。选项应以 `\n` 分隔。 例如: `"First\nSecond\nThird"`
- 您可以使用 `lv_roller_set_selected(roller, id)` 手动选择一个选项, 其中 `id` 是选项的索引
- 可以使用 `lv_roller_set_action(roller, my_action)` 指定回调函数, 以便在选择新选项时调用
- 可以使用 `lv_roller_set_visible_row_count(roller, row_cnt)` 调整滚轴的高度, 以设置可见选项的数量。
- 宽度自动调整。为了防止这种情况, 请使用 `lv_roller_set_hor_fit(roller, false)` 并通过 `lv_obj_set_width(roller, width)` 手动设置宽度
- **Roller** 的打开/关闭动画时间由 `lv_roller_set_anim_time(roller, anim_time)` 调整。 零动画时间意味着没有动画

2.21.1 相关宏定义

滚筒样式(lv_ddlist_style_t)	
LV_ROLLER_STYLE_BG	背景的风格。 使用所有 style.body 属性。 它用于 style.text 的标签样式。 渐变也应用在顶部和底部 Default: lv_style_pretty
LV_ROLLER_STYLE_SEL	所选选项的样式。 使用 style.body 属性。 选定的选项将使用 text.color 重新着色。 Default: lv_style_plain_color

2.21.2 API 函数

函数	描述
lv_roller_create ()	创建一个滚轴对象
lv_roller_set_options()	在滚筒上设置选项
lv_roller_set_selected()	设置所选选项
lv_roller_set_action()	设置要在选择新选项时调用的函数
lv_roller_set_visible_row_count()	设置高度以显示给定的行数（选项）
lv_roller_set_hor_fit()	启用或禁用内容的水平适合度
lv_roller_set_anim_time()	设置打开/关闭动画时间。
lv_roller_set_style()	设置滚轴的风格
lv_roller_get_options()	获得滚轮选项
lv_roller_get_selected()	获取所选选项的 ID
lv_roller_get_selected_str()	将当前选定的选项作为字符串
lv_roller_get_action()	获取“选项选择”回调函数
lv_roller_get_anim_time()	获取打开/关闭动画时间。
lv_roller_get_hor_fit()	获取自动宽度设置属性
lv_roller_get_style()	获得一种滚筒的风格

lv_roller_create()

创建一个滚轴对象

lv_obj_t * lv_roller_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新滚轴的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的滚轴指针

lv_roller_set_options()

在滚筒上设置选项

static inline void lv_roller_set_options(lv_obj_t * roller, const char * options)

参数	描述
roller	滚轮指针指向滚轮对象
options	带有'\ n'分隔选项的字符串。 例如。 “One\ nTwo\ nThree”

返回值：无

lv_roller_set_selected()

设置所选选项

void lv_roller_set_selected(lv_obj_t *roller, uint16_t sel_opt, bool anim_en)

参数	描述
roller	滚轮指针指向滚轮对象
sel_opt	所选选项的 ID (0 ...选项数 - 1)
anim_en	true: 设置动画; false: 立即设置

返回值：无

lv_roller_set_action()

设置要在选择新选项时调用的函数

static inline void lv_roller_set_action(lv_obj_t * roller, lv_action_t action)

参数	描述
roller	滚轮指针指向滚轮对象
action	回调函数指针

返回值：无

lv_roller_set_visible_row_count()

设置高度以显示给定的行数（选项）

void lv_roller_set_visible_row_count(lv_obj_t *roller, uint8_t row_cnt)

参数	描述
roller	滚轮指针指向滚轮对象
row_cnt	所需可见行数

返回值：无

lv_roller_set_hor_fit()

启用或禁用内容的水平适合度

static inline void lv_roller_set_hor_fit(lv_obj_t * roller, bool fit_en)

参数	描述
roller	滚轮指针指向滚轮对象
fit_en	true: 启用自动调整; false: 禁用自动 适合

返回值：无

lv_roller_set_anim_time()

设置打开/关闭动画时间。

static inline void lv_roller_set_anim_time(lv_obj_t *roller, uint16_t anim_time)

参数	描述
roller	滚轮指针指向滚轮对象
anim_time	打开/关闭动画时间[ms]

返回值：无

lv_roller_set_style()

设置滚筒的风格

void lv_roller_set_style(lv_obj_t *roller, lv_roller_style_t type, lv_style_t *style)

参数	描述
roller	滚轮指针指向滚轮对象
type	要设置的样式类型，见 lv_ddlist_style_t
style	样式指针

返回值：无

lv_roller_get_options()

获得滚轮选项

static inline const char * lv_roller_get_options(lv_obj_t *roller)

参数	描述
roller	滚轮指针指向滚轮对象

返回值：由'\n'-s 分隔的选项（例如 “Option1 \nOption2 \nOption3”）

lv_roller_get_selected()

获取所选选项的 ID

static inline uint16_t lv_roller_get_selected(lv_obj_t *roller)

参数	描述
roller	滚轮指针指向滚轮对象

返回值：所选选项的 ID（0 ...选项数 - 1）；

lv_roller_get_selected_str()

将当前选定的选项作为字符串

static inline void lv_roller_get_selected_str(lv_obj_t * roller, char * buf)

参数	描述
roller	滚轮指针指向滚轮对象

返回值：buf 指向存储字符串的数组的指针

lv_roller_get_action()

获取“选项选择”回调函数

static inline lv_action_t lv_roller_get_action(lv_obj_t * roller)

参数	描述
roller	滚轮指针指向滚轮对象

返回值：回调函数指针

lv_roller_get_anim_time()

获取打开/关闭动画时间。

static inline uint16_t lv_roller_get_anim_time(lv_obj_t * roller)

参数	描述
roller	滚轮指针指向滚轮对象

返回值：打开/关闭动画时间[ms]

lv_roller_get_hor_fit()

获取自动宽度设置属性

bool lv_roller_get_hor_fit(lv_obj_t *roller)

参数	描述
roller	滚轮指针指向滚轮对象

返回值：**true**：启用自动尺寸；**false**：启用手动宽度设置

lv_roller_get_style()

获得一种滚筒的风格

lv_style_t * lv_roller_get_style(lv_obj_t *roller, lv_roller_style_t type)

参数	描述
roller	滚轮指针指向滚轮对象
type	要设置的样式类型，见 lv_ddlist_style_t

返回值：样式指针

2.21.3 控件使用

```

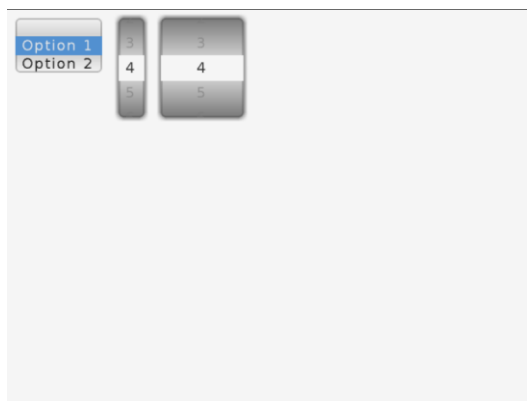
1.  /* 创建默认控件*/
2.  lv_obj_t *roller1 = lv_roller_create(lv_scr_act(), NULL);    //创建 roller 控
    件
3.  lv_obj_set_pos(roller1, 10, 10);    //设置控件显示坐标
4.
5.  /* 设置控件样式 */
6.  static lv_style_t bg;
7.  lv_style_copy(&bg, &lv_style_pretty);
8.  bg.body.main_color = LV_COLOR_GRAY;
9.  bg.body.grad_color = LV_COLOR_WHITE;
10. bg.body.shadow.width = 5;
11. bg.text.line_space = 10;
12. bg.text.opa = LV_OPA_60;
13. bg.text.color = LV_COLOR_GRAY;
14.
15. lv_obj_t *roller2 = lv_roller_create(lv_scr_act(), NULL);    //创建 roller 控
    件
16. lv_obj_set_size(roller2, 80, 120);    //设置控件尺寸大小
    
```

```

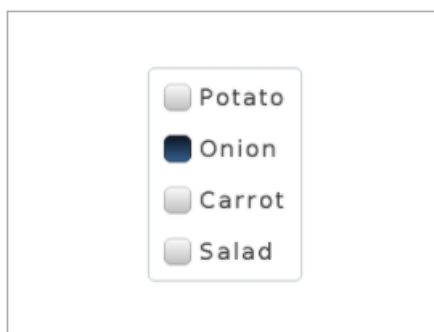
17. lv_roller_set_options(roller2, "0\n1\n2\n3\n4\n5\n6\n7\n8\n9"); //设置 roller
    选项
18. lv_obj_align(roller2, roller1, LV_ALIGN_OUT_RIGHT_TOP, 20, 0); //设置对齐方
    式
19. lv_roller_set_anim_time(roller2, 500); //设置 roller 打开/关闭动画时间
20. lv_roller_set_style(roller2, LV_ROLLER_STYLE_BG, &bg); //设置 roller 样式
21. lv_roller_set_style(roller2, LV_ROLLER_STYLE_SEL, &lv_style_plain);
22. lv_roller_set_selected(roller2, 4, true); //设置 roller 所选选项
23.
24. lv_obj_t *roller3 = lv_roller_create(lv_scr_act(), roller2); //创建 roller
    控件
25. lv_obj_align(roller3, roller2, LV_ALIGN_OUT_RIGHT_TOP, 20, 0); //设置对齐方
    式
26. lv_roller_set_hor_fit(roller3, false); //禁止水平方向自适应
27. lv_obj_set_width(roller3, LV_DPI); //设置控件宽度

```

运行效果



2.22 Check Box



- 复选框对象是从按钮背景构建的，其中包含一个 Button 子弹和一个标签，用于实现经典复选框
- 可以通过 `lv_cb_set_text(cb, "New text")` 函数修改文本
- 一个动作可以由 `lv_cb_set_action(cb, action)` 分配
- 您可以通过 `lv_cb_set_checked(cb, state)` 手动选中/取消选中复选框

2.22.1 相关宏定义

复选框样式(lv_cb_style_t)	
LV_CB_STYLE_BG	背景的风格。 使用所有 style.body 属性。 它用于 style.text 的标签样式。 渐变也应用在顶部和底部 Default: lv_style_pretty
LV_CB_STYLE_BOX_REL	释放框的样式。 使用 style.body 属性 Default: lv_style_btn_rel
LV_CB_STYLE_BOX_PR	点击框的样式。 使用 style.body 属性 Default: lv_style_btn_pr
LV_CB_STYLE_BOX_TGL_REL	已检查的已释放框的样式。 使用 style.body 属性。 Default: lv_style_btn_tgl_rel
LV_CB_STYLE_BOX_TGL_PR	已检查的已释放框的样式。 使用 style.body 属性 Default: lv_style_btn_tgl_pr
LV_CB_STYLE_BOX_INA	

2.22.2 API 函数

函数	描述
lv_cb_create ()	创建一个复选框对象
lv_cb_set_text()	设置复选框的文本
lv_cb_set_checked()	设置复选框的状态
lv_cb_set_inactive()	使复选框处于非活动状态（已禁用）
lv_cb_set_action()	设置要在单击复选框时调用的函数
lv_cb_set_style()	设置复选框的样式
lv_cb_get_text()	获取复选框的文本
lv_cb_is_checked()	获取复选框的当前状态
lv_cb_get_action()	获取复选框的回调函数
lv_cb_get_style()	获取复选框样式

lv_cb_create()

创建一个复选框对象

lv_obj_t * lv_cb_create(lv_obj_t * par, lv_obj_t * copy)

参数	描述
par	指向控件的指针，它将是新复选框的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的复选框指针

lv_cb_set_text()

设置复选框的文本

void lv_cb_set_text(lv_obj_t * cb, const char * txt)

参数	描述
cb	指向复选框的指针
txt	复选框的文本

返回值：无

lv_cb_set_checked()

设置复选框的状态

static inline void lv_cb_set_checked(lv_obj_t * cb, bool checked)

参数	描述
cb	指向复选框的指针
checked	true: 选中复选框; false: 取消选中

返回值：无

lv_cb_set_inactive()

使复选框处于非活动状态（已禁用）

static inline void lv_cb_set_inactive(lv_obj_t * cb)

参数	描述
cb	指向复选框的指针

返回值：无

lv_cb_set_action()

设置要在单击复选框时调用的函数

static inline void lv_cb_set_action(lv_obj_t * cb, lv_action_t action)

参数	描述
cb	指向复选框的指针
action	回调函数

返回值：无

lv_cb_set_style()

设置复选框的样式

void lv_cb_set_style(lv_obj_t * cb, lv_cb_style_t type, lv_style_t *style)

参数	描述
cb	指向复选框的指针
type	要设置的样式类型，见 lv_cb_style_t
style	样式指针

返回值：无

lv_cb_get_text()

获取复选框的文本

const char * lv_cb_get_text(lv_obj_t * cb)

参数	描述
cb	指向复选框的指针

返回值：指向复选框文本的指针

lv_cb_is_checked()

获取复选框的当前状态

static inline bool lv_cb_is_checked(lv_obj_t * cb)

参数	描述
cb	指向复选框的指针

返回值：true：选中复选框; false：取消选中

lv_cb_get_action()

获取复选框的回调函数

static inline lv_action_t lv_cb_get_action(lv_obj_t * cb)

参数	描述
cb	指向复选框的指针

返回值：回调函数指针

lv_cb_get_style()

获取复选框样式

lv_style_t * lv_cb_get_style(lv_obj_t * cb, lv_cb_style_t type)

参数	描述
cb	指向复选框的指针
type	要获取的样式类型，见 lv_cb_style_t

返回值：样式指针

2.22.3 控件使用

```

1.  /* 创建基本控件 */
2.  lv_obj_t * cb1 = lv_cb_create(lv_scr_act(), NULL);  //创建控件
3.
4.  /* 创建另一个复选框并设置其文本 */
5.  lv_obj_t * cb2 = lv_cb_create(lv_scr_act(), NULL);  //创建控件
6.  lv_cb_set_text(cb2, "UTF8-text: ü? ?? if"); //设置复选框的文本
7.  lv_obj_align(cb2, cb1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 10);    //设置对齐方式
8.
9.  /* 设置控件样式 */
10. static lv_style_t cb3_styles[LV_BTN_STATE_NUM];
11. lv_style_copy(&cb3_styles[LV_BTN_STATE_REL], &lv_style_plain);
12. cb3_styles[LV_BTN_STATE_REL].body.radius = LV_DPI / 20;
13. cb3_styles[LV_BTN_STATE_REL].body.border.width = 1;
14. cb3_styles[LV_BTN_STATE_REL].body.border.color = LV_COLOR_GRAY;

```

```

15. cb3_styles[LV_BTN_STATE_REL].body.main_color = LV_COLOR_WHITE;
16. cb3_styles[LV_BTN_STATE_REL].body.grad_color = LV_COLOR_SILVER;
17.
18. lv_style_copy(&cb3_styles[LV_BTN_STATE_PR], &cb3_styles[LV_BTN_STATE_REL]);

19. cb3_styles[LV_BTN_STATE_PR].body.main_color = LV_COLOR_SILVER;
20. cb3_styles[LV_BTN_STATE_PR].body.grad_color = LV_COLOR_GRAY;
21.
22. lv_style_copy(&cb3_styles[LV_BTN_STATE_TGL_REL], &cb3_styles[LV_BTN_STATE_REL]);
23. cb3_styles[LV_BTN_STATE_TGL_REL].body.border.width = 4;
24. cb3_styles[LV_BTN_STATE_TGL_REL].body.border.color = LV_COLOR_WHITE;
25. cb3_styles[LV_BTN_STATE_TGL_REL].body.border.opa = LV_OPA_70;
26. cb3_styles[LV_BTN_STATE_TGL_REL].body.main_color = LV_COLOR_GRAY;
27. cb3_styles[LV_BTN_STATE_TGL_REL].body.grad_color = LV_COLOR_BLACK;
28.
29. lv_style_copy(&cb3_styles[LV_BTN_STATE_TGL_PR], &cb3_styles[LV_BTN_STATE_TGL_REL]);
30. cb3_styles[LV_BTN_STATE_TGL_PR].body.border.color = LV_COLOR_SILVER;
31. cb3_styles[LV_BTN_STATE_TGL_PR].body.border.opa = LV_OPA_70;
32. cb3_styles[LV_BTN_STATE_TGL_PR].body.main_color = LV_COLOR_GRAY;
33. cb3_styles[LV_BTN_STATE_TGL_PR].body.grad_color = LV_COLOR_BLACK;
34.
35. lv_style_copy(&cb3_styles[LV_BTN_STATE_INA], &cb3_styles[LV_BTN_STATE_TGL_REL]);
36. cb3_styles[LV_BTN_STATE_INA].body.border.width = 1;
37. cb3_styles[LV_BTN_STATE_INA].body.border.color = LV_COLOR_GRAY;
38. cb3_styles[LV_BTN_STATE_INA].body.main_color = LV_COLOR_SILVER;
39. cb3_styles[LV_BTN_STATE_INA].body.grad_color = LV_COLOR_SILVER;
40.
41.
42. /*复制上一个复选框并应用新样式*/
43. lv_obj_t *cb3 = lv_cb_create(lv_scr_act(), cb2);    //创建控件
44. lv_cb_set_style(cb3, LV_CB_STYLE_BOX_REL, &cb3_styles[LV_BTN_STATE_REL]);
    //设置样式
45. lv_cb_set_style(cb3, LV_CB_STYLE_BOX_PR, &cb3_styles[LV_BTN_STATE_PR]);
46. lv_cb_set_style(cb3, LV_CB_STYLE_BOX_TGL_REL, &cb3_styles[LV_BTN_STATE_TGL_REL]);
47. lv_cb_set_style(cb3, LV_CB_STYLE_BOX_TGL_PR, &cb3_styles[LV_BTN_STATE_TGL_PR]);
48. lv_cb_set_style(cb3, LV_CB_STYLE_BOX_INA, &cb3_styles[LV_BTN_STATE_INA]);
49. lv_obj_align(cb3, cb2, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 10);    //设置对齐方式
50.
51. /*Copy the previous check box and set it to INACTIVE*/

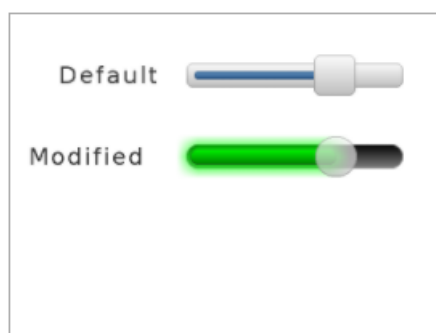
```

```
52. lv_obj_t *cb4 = lv_cb_create(lv_scr_act(), cb3);    //创建控件
53. lv_obj_align(cb4, cb3, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 10);    //设置对齐方式
54. lv_btn_set_state(cb4, LV_BTN_STATE_INA);    //设置按钮状态为禁用状态
```

运行效果



2.23 Slider



- 滑块对象看起来像一个补充了旋钮的进度条。 可以拖动旋钮来设置值。 滑块也可以是垂直的或水平的。
- 根据宽度/高度比，杆的方向可以是垂直的或水平的。 在水平条上逻辑上指示器宽度，在垂直条上可以改变指示器高度
- 可以通过以下方式设置新值：`lv_bar_set_value(bar, new_value)`。 该值在范围（最小值和最大值）中解释，可以使用以下值进行修改：
`lv_bar_set_range(bar, min, max)`。 默认范围是：1..100
- 使用从当前值到所需的动画可以设置新值。 在这种情况下使用
`lv_bar_set_value_anim(bar, new_value, anim_time)`

2.23.1 相关宏定义

复选框样式(`lv_slider_style_t`)

`LV_SLIDER_STYLE_BG`

`LV_SLIDER_STYLE_INDIC`

`LV_SLIDER_STYLE_KNOB`

2.23.2 API 函数

函数	描述
lv_slider_create ()	创建滑块对象
lv_slider_set_value()	设置滑块新值
lv_slider_set_value_anim()	在滑块上设置带动画的新值
lv_slider_set_range()	设置滑块的最小值和最大值
lv_slider_set_action()	设置在滑块上设置新值时调用的函数
lv_slider_set_knob_in()	设置滑块的'knob in'属性
lv_slider_set_style()	设置滑块的样式
lv_slider_get_value()	获取滑块的值
lv_slider_get_min_value()	获取滑块最小值
lv_slider_get_max_value()	获取滑块最大值
lv_slider_get_action()	获取滑块动作功能
lv_slider_is_dragged()	是否正在拖动滑块
lv_slider_get_knob_in()	获取滑块的'knob in'属性
lv_slider_get_style()	获得滑块的样式

lv_slider_create()

创建滑块对象

`lv_obj_t * lv_slider_create(lv_obj_t * par, lv_obj_t * copy)`

参数	描述
par	指向控件的指针，它将是新滑块的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的滑块指针

lv_slider_set_value()

设置滑块新值

`static inline void lv_slider_set_value(lv_obj_t * slider, int16_t value)`

参数	描述
slider	指向滑块对象的指针
value	新值

返回值：无

lv_slider_set_value_anim()

在滑块上设置带动画的新值

`static inline void lv_slider_set_value_anim(lv_obj_t * slider, int16_t value, uint16_t anim_time)`

参数	描述
----	----

slider	指向滑块对象的指针
value	新值
anim_time	动画时间，以毫秒为单位

返回值：无

lv_slider_set_range()

设置滑块的最小值和最大值

static inline void lv_slider_set_range(lv_obj_t *slider, int16_t min, int16_t max)

参数	描述
slider	指向滑块对象的指针
min	最小值
max	最大值

返回值：无

lv_slider_set_action()

设置在滑块上设置新值时将调用的函数

void lv_slider_set_action(lv_obj_t * slider, lv_action_t action)

参数	描述
slider	指向滑块对象的指针
action	回调函数

返回值：无

lv_slider_set_knob_in()

设置滑块的'knob in'属性

void lv_slider_set_knob_in(lv_obj_t * slider, bool in)

参数	描述
slider	指向滑块对象的指针
type	true: 旋钮总是在滑块中绘制; false: 旋钮可以在边缘上

返回值：无

lv_slider_set_style()

设置滑块的样式

void lv_slider_set_style(lv_obj_t *slider, lv_slider_style_t type, lv_style_t *style)

参数	描述
slider	指向滑块对象的指针
type	要设置的样式指针，见 lv_slider_style_t
style	样式

返回值：无

lv_slider_get_value()

获取滑块的值

int16_t lv_slider_get_value(lv_obj_t * slider)

参数	描述
slider	指向滑块对象的指针

返回值：滑块的值

lv_slider_get_min_value()

获取滑块最小值

static inline int16_t lv_slider_get_min_value(lv_obj_t * slider)

参数	描述
slider	指向滑块对象的指针

返回值：滑块的最小值

lv_slider_get_max_value()

获取滑块最大值

static inline int16_t lv_slider_get_max_value(lv_obj_t * slider)

参数	描述
slider	指向滑块对象的指针

返回值：滑块的最大值

lv_slider_get_action()

获取滑块动作功能

lv_action_t lv_slider_get_action(lv_obj_t * slider)

参数	描述
slider	指向滑块对象的指针

返回值：回调函数

lv_slider_is_dragged()

是否正在拖动滑块

bool lv_slider_is_dragged(lv_obj_t * slider)

参数	描述
slider	指向滑块对象的指针

返回值：true：在拖动 false：没拖动

lv_slider_get_knob_in()

获取滑块的'knob in'属性

bool lv_slider_get_knob_in(lv_obj_t * slider)

参数	描述
slider	指向滑块对象的指针

返回值：true：旋钮总是在滑块中绘制; false：旋钮可以在边缘上

lv_slider_get_style()

获得滑块的样式

lv_style_t * lv_slider_get_style(lv_obj_t * slider, lv_slider_style_t type)

参数	描述
slider	指向滑块对象的指针

type	要设置的样式指针，见 lv_slider_style_t
------	---

返回值：样式指针

2.23.3 控件使用

```

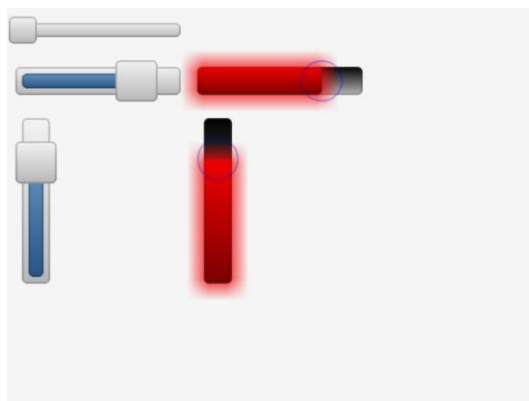
1. /* 创建默认控件*/
2. lv_obj_t * slider1 = lv_slider_create(lv_scr_act(), NULL); //创建控件
3. lv_obj_set_pos(slider1, 10, 10); //设置控件显示坐标
4.
5. /* 修改尺寸位置、设置值为 75% */
6. lv_obj_t * slider2 = lv_slider_create(lv_scr_act(), NULL); //创建控件
7. lv_obj_set_size(slider2, 150, 50); //设置控件尺寸大小
8. lv_obj_align(slider2, slider1, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20); //设置对
   齐方式
9. lv_slider_set_range(slider2, -50, 50); //设置 slider 最小值和最大值
10. lv_slider_set_value(slider2, 25); //设置 slider 当前值
11.
12. /* 复制'slider2'但将其大小设置为垂直（指标为 75%）*/
13. lv_obj_t * slider3 = lv_slider_create(lv_scr_act(), slider2); //创建控件
14. lv_obj_set_size(slider3, 50, 150); //设置控件尺寸大小,此时控件是垂直的
15. lv_obj_align(slider3, slider2, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20); //设置对
   齐方式
16.
17.
18. /* 复制'slider2'并为其设置新样式*/
19. static lv_style_t slider_bg;
20. lv_style_copy(&slider_bg, &lv_style_pretty);
21. slider_bg.body.main_color = LV_COLOR_BLACK;
22.
23. static lv_style_t slider_indic;
24. lv_style_copy(&slider_indic, &lv_style_pretty);
25. slider_indic.body.main_color = LV_COLOR_RED;
26. slider_indic.body.grad_color = LV_COLOR_MARRON;
27. slider_indic.body.shadow.color = LV_COLOR_RED;
28. slider_indic.body.shadow.width = 20;
29. slider_indic.body.padding.ver = 0;
30. slider_indic.body.padding.hor = 0;
31.
32. static lv_style_t slider_knob;
33. lv_style_copy(&slider_knob, &lv_style_pretty);
34. slider_knob.body.radius = LV_RADIUS_CIRCLE;
35. slider_knob.body.border.color = LV_COLOR_BLUE;

```

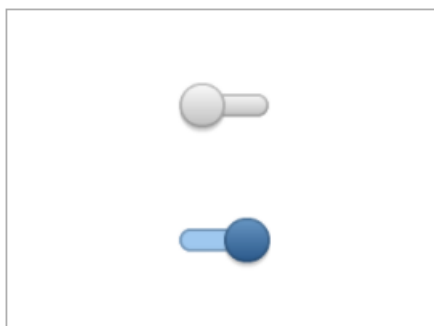
```

36. slider_knob.body.empty = 1;
37.
38. lv_obj_t * slider4 = lv_slider_create(lv_scr_act(), slider2); //创建控件
39. lv_obj_align(slider4, slider2, LV_ALIGN_OUT_RIGHT_MID, 20, 0); //设置对齐方
    式
40. lv_slider_set_style(slider4, LV_SLIDER_STYLE_BG, &slider_bg); //设置样式
41. lv_slider_set_style(slider4, LV_SLIDER_STYLE_INDIC, &slider_indic); //设置样
    式
42. lv_slider_set_style(slider4, LV_SLIDER_STYLE_KNOB, &slider_knob); //设置样
    式
43.
44. /* 复制'slider4'但将其大小设置为垂直*/
45. lv_obj_t * slider5 = lv_slider_create(lv_scr_act(), slider4); //创建控件
46. lv_obj_set_size(slider5, 50, 150); //设置控件尺寸大小,此时控件是垂直的
47. lv_obj_align(slider5, slider4, LV_ALIGN_OUT_BOTTOM_LEFT, 0, 20); //设置对
    齐方式
    
```

运行结果



2.24 Switch



- **Switch** 可用于打开/关闭某些东西。 看起来像一个小滑块。 可以通过以下方式更改开关的状态：
 - 点击它
 - 滑动它

- 或者 `lv_sw_on(sw)` 和 `lv_sw_off(sw)` 的函数
- 当用户使用开关时，可以分配回调函数来调用：`lv_sw_set_action(sw, my_action)`

2.24.1 相关宏定义

开关样式(`lv_sw_style_t`)

`LV_SW_STYLE_BG`

`LV_SW_STYLE_INDIC`

`LV_SW_STYLE_KNOB_OFF`

`LV_SW_STYLE_KNOB_ON`

2.24.2 API 函数

函数	描述
<code>lv_sw_create ()</code>	创建一个开关对象
<code>lv_sw_on()</code>	打开开关
<code>lv_sw_off()</code>	关闭开关
<code>lv_sw_set_action()</code>	
<code>lv_sw_set_style()</code>	设置开关的样式
<code>lv_sw_get_state()</code>	获取开关的状态
<code>lv_sw_get_action()</code>	获取切换动作的回调函数
<code>lv_sw_get_style()</code>	获得开关样式

`lv_sw_create()`

创建一个开关对象

`lv_obj_t * lv_sw_create(lv_obj_t * par, lv_obj_t * copy)`

参数	描述
par	指向控件的指针，它将是新开关的父控件
copy	指向窗口对象的指针，如果不是 NULL，则将从该控件复制新控件

返回值：创建的开关指针

`lv_sw_on()`

打开开关

`void lv_sw_on(lv_obj_t *sw)`

参数	描述
sw	指向开关对象的指针

返回值：无

lv_sw_off()

关闭开关

void lv_sw_off(lv_obj_t *sw)

参数	描述
sw	指向开关对象的指针

返回值：无

lv_sw_set_action()

设置一个功能，当用户切换开关时将调用该功能

static inline void lv_sw_set_action(lv_obj_t * sw, lv_action_t action)

参数	描述
sw	指向开关对象的指针
action	回调函数

lv_sw_set_style()

设置开关的样式

void lv_sw_set_style(lv_obj_t *sw, lv_sw_style_t type, lv_style_t *style)

参数	描述
sw	指向开关对象的指针
type	要设置的样式指针，见 lv_sw_style_t
style	样式指针

返回值：无

lv_sw_get_state()

获取开关的状态

static inline bool lv_sw_get_state(lv_obj_t *sw)

参数	描述
sw	指向开关对象的指针

返回值：false: OFF; true: ON

lv_sw_get_action()

获取切换动作的回调函数

static inline lv_action_t lv_sw_get_action(lv_obj_t * slider)

参数	描述
slider	指向开关对象的指针

返回值：回调函数

lv_sw_get_style()

获得开关样式

lv_style_t * lv_sw_get_style(lv_obj_t *sw, lv_sw_style_t type)

参数	描述
sw	指向开关对象的指针
type	要设置的样式指针，见 lv_sw_style_t

返回值：样式指针

2.24.3 控件使用

```
1. lv_obj_t *led_0;    //led 句柄
2. lv_obj_t *sw1;    //开关句柄
3.
4. /* switch 回调函数 */
5. static lv_res_t sw_action(lv_obj_t *sw)
6. {
7.     /* 获取 switch 开关状态,根据状态设置 led 状态 */
8.     if(lv_sw_get_state(sw1) == true)
9.     {
10.         lv_led_on(led_0);
11.     }
12.     else
13.     {
14.         lv_led_off(led_0);
15.     }
16.     return LV_RES_OK;
17. }
18.
19. void lv_test_sw_1(void)
20. {
21.     led_0 = lv_led_create(lv_scr_act(),NULL);
22.     lv_obj_set_pos(led_0, 10, 200);
23.
24.     /* 创建默认控件 */
25.     sw1 = lv_sw_create(lv_scr_act(), NULL); //创建 switch 控件
26.     lv_obj_set_pos(sw1, 10, 10);    //设置控件显示坐标
27.     lv_sw_set_action(sw1, sw_action);    //设置 switch 回调函数
28.
29.     /* 设置控件样式 */
30.     static lv_style_t bg;
31.     static lv_style_t indic;
32.
33.     lv_style_copy(&bg, &lv_style_pretty);
34.     bg.body.padding.hor = -5;
35.     bg.body.padding.ver = -5;
36.
37.     lv_style_copy(&indic, &lv_style_pretty_color);
38.     indic.body.padding.hor = 8;
39.     indic.body.padding.ver = 8;
40.
```

```

41.    lv_obj_t *sw2 = lv_sw_create(lv_scr_act(), sw1);    //创建 switch 控件
42.    lv_sw_set_style(sw2, LV_SW_STYLE_BG, &bg);
43.    lv_sw_set_style(sw2, LV_SW_STYLE_INDIC, &indic);
44.    lv_sw_set_style(sw2, LV_SW_STYLE_KNOB_OFF, &lv_style_btn_pr);
45.    lv_sw_set_style(sw2, LV_SW_STYLE_KNOB_ON, &lv_style_btn_tgl_pr);
46.
47.    lv_sw_on(sw2);    //初始化 switch 为开状态
48.    lv_obj_align(sw2, sw1, LV_ALIGN_OUT_RIGHT_MID, 20, 0);    //设置对齐方式
49.
50.    lv_obj_t *sw3 = lv_sw_create(lv_scr_act(), sw2);    //创建 switch 控件
51.    lv_obj_align(sw3, sw2, LV_ALIGN_OUT_RIGHT_MID, 20, 0);    //设置对齐方式
52.
53. }

```

运行结果



第三章：将 LittlevGL 移植到微控制器

3.1 系统架构



应用

您的应用程序，它创建 GUI 并处理特定任务。

LittlevGL

图形库本身。您的应用程序可以与库通信以创建 GUI。它包含一个 HAL（硬件抽象层）接口，用于注册显示器和输入设备驱动程序。

驱动

除了您的特定驱动程序，它还包含驱动显示器的功能，可选择驱动 GPU 和读取触摸板或按钮。

有两种典型的硬件设置取决于 MCU 是否具有 LCD / TFT 驱动器外围设备。在这两种情况下，将需要帧缓冲器来存储屏幕的当前图像。

带 TFT / LCD 驱动器的 MCU

如果您的 MCU 具有 TFT / LCD 驱动器外围设备，则可以通过 RGB 接口直接连接显示器。在这种情况下，帧缓冲区可以位于内部 RAM 中（如果 MCU 有足够的 RAM），也可以位于外部 RAM 中（如果 MCU 具有存储器接口）。

外部显示控制器

如果 MCU 没有 TFT / LCD 驱动器，则必须使用外部显示控制器（例如 SSD1963, SSD1306, ILI9341）。在这种情况下，MCU 可以通过并行端口，SPI 或有时 I2C 与显示控制器通信。帧缓冲器通常位于显示控制器中，为 MCU 节省了大量 RAM。

3.2 硬件需求

- 16,32 或 64 位微控制器或处理器
- 16 MHz 时钟速度
- 8 kB RAM 用于静态数据，> 2 KB RAM 用于动态数据（图形对象）
- 64 kB 程序存储器（闪存）
- 可选择~1 / 10 屏幕大小的内存用于内部缓冲（240×320,16 位颜色，表示 15 kB）

LittlevGL 旨在实现高度可移植性，不使用任何外部资源：

- 无需外部 RAM（但支持）
- 没有使用浮点数
- 无需 GPU（但支持）
- 在以下位置只需要一个帧缓冲区：
 - 内部 RAM
 - 外部 RAM
 - 外部显示控制器的内存

如果您想减少所需的硬件资源，您可以：

- 禁用未使用的对象类型以保存 RAM 和 ROM

- 更改图形缓冲区的大小以节省 RAM
- 使用更简单的样式可缩短渲染时间

3.3 工程设置

3.3.1 获取库

Littlev 图形库可以在 GitHub 上找到: <https://github.com/littlevgl/lvgl>。您可以从此处克隆或下载最新版本的库,也可以使用“下载”页面。

图形库是 lvgl 目录,应该复制到项目中

3.3.2 配置文件

LittlevGL 有一个配置头文件: `lv_conf.h`。它在编译时设置库的基本行为,禁用未使用的模块和功能,并调整内存缓冲区的大小等。

复制 lvgl 目录旁边的 `lvgl/lv_conf_templ.h` 并将其重命名为 `lv_conf.h`。打开文件并删除第一个 `#if` 和最后一个 `#endif` 以启用内容。在配置文件中,注释解释了选项的含义。至少检查这三个配置选项并根据您的硬件进行修改:

1. **LV_HOR_RES** 您的显示器的水平分辨率
2. **LV_VER_RES** 您的显示器的垂直分辨率
3. **LV_COLOR_PETH** 8 为 (RG332), 16 为 (RGB565) 或 24 为 (RGB888 和 ARGB8888)。

3.3.3 初始化

为了使用图形库,您必须初始化它和其他组件。要初始化的顺序是:

1. 调用 `lv_init()`
2. 初始化您的驱动程序
3. 在 LittlevGL 中注册显示和输入设备驱动程序 (见下文)

3.4 移植库

要首先将 LittlevGL 应用到项目中,您必须提供一些功能并在图形库中注册它们。

3.4.1 显示接口

要设置显示，必须初始化 `lv_disp_drv_t` 变量：

```
lv_disp_drv_t disp_drv;
lv_disp_drv_init(&disp_drv);           /*Basic initialization*/
disp_drv. ... = ...                     /*Initialize the field here. See below.*/
disp_drv_register(&disp_drv);          /*Register the driver in LittlevGL*/
```

您可以为不同的操作模式配置驱动程序。要了解有关绘图模式的更多信息，请访问[绘图和渲染](#)。

内部缓冲（VDB）

图形库与内部缓冲机制配合使用，只需一个帧缓冲区即可创建高级图形功能。内部缓冲区称为 VDB（虚拟显示缓冲区），其大小可在 `lv_conf.h` 中使用 `LV_VDB_SIZE` 进行调整。当 `LV_VDB_SIZE > 0` 时，则使用内部缓冲，您必须提供一个将缓冲区内容刷新到显示器的功能：

```
disp_drv.disp_flush = my_disp_flush;
.
.
.
void my_disp_flush(int32_t x1, int32_t y1, int32_t x2, int32_t y2, const
lv_color_t * color_p)
{
    /*TODO Copy 'color_p' to the specified area*/

    /*Call 'lv_flush_ready()' when ready*/
    lv_flush_ready();
}
```

在刷新功能中，您可以使用 DMA 或任何硬件在后台进行刷新，但是当刷新就绪时，您必须调用 `lv_flush_ready()`；

硬件加速（GPU）

首先使用 GPU 是完全可选的。但如果您的 MCU 支持图形加速，那么您可以使用它。显示驱动程序的 `mem_blend` 和 `mem_fill` 字段用于与 GPU 连接。仅当启用内部缓冲（VDB）时，才能使用 GPU 相关功能。

```
1. disp_drv.mem_blend = my_mem_blend; /*Blends two color arrays using opacity*/
   /
2. disp_drv.mem_fill = my_mem_fill;   /*Fills an array with a color*/
3. .
4. .
```

```

5. .
6.
7. void my_mem_blend(lv_color_t * dest, const lv_color_t * src, uint32_t length
    , lv_opa_t opa)
8. {
9.     /*TODO Copy 'src' to 'dest' but blend it with 'opa' alpha */
10. }
11.
12. void my_mem_fill(lv_color_t * dest, uint32_t length, lv_color_t color)
13. {
14.     /*TODO Fill 'length' pixels in 'dest' with 'color'*/
15. }

```

无缓冲的绘图

当内部缓冲可以被发送时（LV_VDB_SIZE = 0），可以直接绘制到帧缓冲区。

```

1. disp_drv.disp_fill = my_disp_fill; /*Fill an area in the frame buffer*/
2. disp_drv.disp_map = my_disp_map; /*Copy a color_map (e.g. image) into the
    frame buffer*/
3. .
4. .
5. .
6. void my_disp_map(int32_t x1, int32_t y1, int32_t x2, int32_t y2, const lv_color_t * color_p)
7. {
8.     /*TODO Copy 'color_p' to the specified area*/
9. }
10.
11. void my_disp_fill(int32_t x1, int32_t y1, int32_t x2, int32_t y2, lv_color_t color)
12. {
13.     /*TODO Fill the specified area with 'color'*/
14. }

```

如果您使用支持加速填充的外部显示控制器（例如 RA8876），那么您可以在 *disp_fill()* 中使用此功能

3.4.2 输入设备接口

要设置输入设备，必须初始化 lv_indev_drv_t 变量：

```

1. lv_indev_drv_t indev_drv;
2. lv_indev_drv_init(&indev_drv); /*Basic initialization*/

```

```
3. indev_drv.type = ...           /*See below.*/
4. indev_drv.read = ...          /*See below.*/
5. lv_indev_drv_register(&indev_drv);    /*Register the driver in LittlevGL*/
```

类型可以是 `LV_INDEV_TYPE_POINTER`（例如触摸板）或 `LV_INDEV_TYPE_KEYPAD`（例如键盘）

`read` 是一个函数指针，它将被定期调用以报告输入设备的当前状态。当缓冲区不为空时，它还可以缓冲数据并在不再读取数据时返回 `false` 或返回 `true`。

要了解输入设备的更多信息，请访问[输入设备](#)

触摸板，鼠标或任何指针

```
1. indev_drv.type = LV_INDEV_TYPE_POINTER;
2. indev_drv.read = my_input_read;
```

读取函数应该是这样的：

```
1. bool my_input_read(lv_indev_data_t *data)
2. {
3.     data->point.x = touchpad_x;
4.     data->point.y = touchpad_y;
5.     data->state = LV_INDEV_STATE_PR or LV_INDEV_STATE_REL;
6.     return false;           /*No buffering so no more data read*/
7. }
```

键盘

```
1. indev_drv.type = LV_INDEV_TYPE_KEYPAD;
2. indev_drv.read = my_input_read;
```

读取函数应该是这样的：

```
1. bool keyboard_read(lv_indev_data_t *data)
2. {
3.     data->key = last_key();
4.
5.     if(key_pressed()) {
6.         data->state = LV_INDEV_STATE_PR;
7.     } else {
8.         data->state = LV_INDEV_STATE_REL;
9.     }
10. }
```

```
11.     return false;    /*No buffering so no more data read*/  
12. }
```

要使用键盘：

- 使用 `LV_INDEV_TYPE_KEYPAD` 类型注册读取函数（如上所述）。
- 必须在 `lv_conf.h` 中启用 `USE_LV_GROUP`
- 必须创建一个对象组： `lv_group_create()` 并且必须添加对象：
`lv_group_add_obj()`
- 必须将创建的组分配给输入设备： `lv_indev_set_group(my_indev, group1);`
- 使用 `LV_GROUP_KEY_...` 在组中的对象之间导航

访问 [Touchpad-less navigation](#) 以了解更多信息。

3.4.3 滴答时钟接口

LittlevGL 使用系统节拍。 定期调用 `lv_tick_inc(tick_period)` 函数并以毫秒为单位告知调用周期。 例如，如果每一毫秒调用一次：

```
lv_tick_inc(1)
```

3.4.4 任务处理

要处理 LittlevGL 的任务，您需要定期调用 `lv_task_handler()`：

- `while(1)` 的 `main()` 函数里面
- 定时中断定时里面
- 定期执行 OS 任务

周期性的 OS 任务时间并不重要，但应该是大约 5 毫秒

3.4.5 移植示例

版权说明

版本	版本说明	作者
V0.1	草稿(V5.1.1)	官庆灿