

正点原子 littleVGL 开发指南

lv_kb 键盘

开发指南

正点原子
广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.00	2020/05/01	第一次发布

lv_kb 键盘

1. 介绍

在上一章节中,我们学会了 lv_ta 文本域控件的使用,这一节我们趁热打铁,来学习它的另外一个小伙伴 lv_kb 键盘,从它的控件实现原理上来讲,lv_kb 键盘其实就是一个复杂而又特殊的 lv_btnm 矩阵按钮对象,而 lv_btnm 矩阵按钮我们在前面的章节中也学习过了,所以借助前面的知识,我们应该能更好的理解 lv_kb 键盘.

lv_kb 键盘有 2 种模式,一种是 LV_KB_MODE_TEXT 文本键盘,另一种是 LV_KB_MODE_NUM 数字键盘,其中文本键盘复杂一些,它里面包含了小写,大写,符号三类键盘,文本键盘中的三类键盘是通过如下方式来切换的:

- 1) 点击  键切换到大写键盘
- 2) 点击  键切换到小写键盘
- 3) 点击  键切换到符号

为了有个感性认识,请直接看如下外观图:



图 1.1 文本键盘-小写



图 1.2 文本键盘-大写




图 1.3 文本键盘-符号




图 1.4 数字键盘

虽然文本键盘-符号中也含有 0123456789 数字字符,但对比数字键盘而言,数字键盘表现的更专一,它只能输入数字,对于创建出来的键盘,如果你不指定它的坐标位置和大小,那么它是有一个默认的坐标位置和大小,默认是与父对象底部居中对齐的,默认宽度等于父对象的可适应宽度(即除掉左右内边距后的宽度),默认高度等于父对象的可适应高度(即除掉上下内边距后的高度)的一半,一个 lv_kb 键盘对象可以绑定一个 lv_ta 文本域对象,这是通过 lv_kb_set_ta(kb, ta)接口来完成绑定的,一旦绑定之后,用户在键盘上按下的字符会相应的输入到绑定的文本域控件中,于此同时键盘对象也可以接管对文本域控件上光标的控制,接管之后的好处就是一致性体验更好,具体表现是当键盘绑定新的文本域控件时,键盘会将之前绑定的文本域控件上的光标给隐藏掉,而让新绑定的文本域控件上显示光标,这个接管光标的操作是通过 lv_kb_set_cursor_manage(kb, true)接口来完成的,默认是不接管的。

对于 lv_kb 键盘而言,它到底具有哪些字符按键,以及排版顺序如何,这些东西在键盘选定模式之后都是固定的,那要是我们想自定义键盘咋办?比如添加或者删除一个按键,我可以告诉大家这是能实现的,因为 lv_kb 键盘给我们留出了 lv_kb_set_map(kb, map)和 lv_kb_set_ctrl_map(kb, ctrl_map)这两个 API 接口,而这两个接口的本质其实就是 lv_btmn 矩阵按钮章节中的 lv_btmn_set_map(kb, map)和 lv_btmn_set_ctrl_map(kb, ctrl_map)接口,讲到这里,大家是不是有点恍然大悟了呢?如果叫大家自己去实现,可能还是有点难度,但是我们可以模仿官方的 lv_kb.c 源码来实现呀,大家抄总会吧!在本章的例程设计中,我会给大家演示如何实现自定义键盘,在 lv_kb 键盘中,有些按键是具有特殊功能的,此特殊功能按键上的文本标题是不能随便改动的,如果我们自定义的键盘上也想具有此特殊功能的按键,那么我们也得必须遵守如下特殊按键原则:

LV_SYMBOL_OK: Apply 确认,对应键盘上的  键,点击会触发 LV_EVENT_APPLY 事件

LV_SYMBOL_CLOSE: Close 关闭,对应键盘上的  键,点击会触发 LV_EVENT_CANCEL 事件

LV_SYMBOL_LEFT: 将光标向左移动一步

LV_SYMBOL_RIGHT: 将光标向右移动一步

“ABC” : 切换到大写键盘

“abc” : 切换到小写键盘


“Enter” : 换行

“Bkps” : 删除光标左侧的一个字符

最后我们来讲一下 lv_kb 键盘的事件,常用的按下,松手事件就不讲了,主要说跟它相关的三个特殊事件:

LV_EVENT_VALUE_CHANGED: 当键盘上的按键被按下或者被重复按下时,会触发此事件,于此同时,被按下的按键 id 会作为事件自定义参数给传递过去

LV_EVENT_APPLY: 当键盘上的  键被点击时会触发此事件

LV_EVENT_CANCEL: 当键盘上的  键被点击时会触发此事件

然后这里有一点需要注意的是 lv_kb 键盘控件自身是有一个名为 lv_kb_def_event_cb 的默认事件回调函数,它处理按键按下,按键值改变,管理绑定的文本域等操作,当然了你可以用你自定义的事件回调函数来覆盖掉它,但是为了减少一些相同事务的处理,我们可以在我们自定义的事件回调函数里的前面调用一下默认的 lv_kb_def_event_cb 事件回调函数.

2. lv_kb 的 API 接口

2.1 主要数据类型

2.1.1 键盘模式数据类型

```
enum {  
    LV_KB_MODE_TEXT,  
    LV_KB_MODE_NUM,  
};  
typedef uint8_t lv_kb_mode_t;
```

LV_KB_MODE_TEXT: 文本键盘模式,里面又包含了小写,大写,符号三类键盘

LV_KB_MODE_NUM: 数字键盘模式,比较专一,只能用来输入数字

2.1.2 键盘样式数据类型

```
enum {  
    LV_KB_STYLE_BG,  
    LV_KB_STYLE_BTN_REL,  
    LV_KB_STYLE_BTN_PR,  
    LV_KB_STYLE_BTN_TGL_REL,  
    LV_KB_STYLE_BTN_TGL_PR,  
    LV_KB_STYLE_BTN_INA,  
};  
typedef uint8_t lv_kb_style_t;
```

虽然这里面有 6 种样式,但是最后面的 5 种样式是用来修饰按钮的,跟 lv_btn 按钮章节中的使用方法是一样的,这里就不介绍了

LV_KB_STYLE_BG: 用来修饰键盘的背景,使用样式中的 body 字段,默认值为 lv_style_pretty

2.2 API 接口

2.2.1 创建对象

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

参数:

par: 父对象

copy: 拷贝的对象,如果无拷贝的话,传 NULL 值

返回值:

返回创建出来的对象,如果返回 NULL 的话,说明堆空间不够了

2.2.2 绑定文本域控件

```
void lv_kb_set_ta(lv_obj_t * kb, lv_obj_t * ta);
```

参数:

kb: 键盘对象

ta: 文本域对象

2.2.3 设置键盘的模式

```
void lv_kb_set_mode(lv_obj_t * kb, lv_kb_mode_t mode);
```

参数:

kb: 键盘对象

mode: 键盘模式,有如下两个可选值

LV_KB_MODE_TEXT: 文本键盘模式,里面又包含了小写,大写,符号三类键盘

LV_KB_MODE_NUM: 数字键盘模式,比较专一,只能用来输入数字

2.2.4 是否接管对光标的控制

```
void lv_kb_set_cursor_manage(lv_obj_t * kb, bool en);
```

参数:

kb: 键盘对象

en: 是否使能接管

接管之后的好处就是一致性体验更好,具体表现是当键盘绑定新的文本域控件时,键盘会将之前绑定的文本域控件上的光标给隐藏掉,而让新绑定的文本域控件上显示光标

2.2.5 设置按键映射表

```
static inline void lv_kb_set_map(lv_obj_t * kb, const char * map[]);
```

参数:

kb: 键盘对象

map: 按键映射表

利用此 API 接口是可以实现自定义键盘的,这个接口的使用方法和 lv_btnm 章节中的 lv_btnm_set_map 接口使用方法是一模一样的,详情请看 lv_btnm 章节

2.2.6 设置按键属性控制映射表

```
static inline void lv_kb_set_ctrl_map(lv_obj_t * kb, const lv_btnm_ctrl_t ctrl_map[]);
```

参数:

kb: 键盘对象

ctrl_map: 属性控制映射表

利用此 API 接口是可以实现自定义键盘的,这个接口的使用方法和 lv_btnm 章节中的 lv_btnm_set_ctrl_map 接口使用方法是一模一样的,详情请看 lv_btnm 章节

2.2.7 设置样式

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

参数:

kb: 键盘对象

type: 设置哪一部分的样式,有如下 6 个可选值

LV_KB_STYLE_BG: 修饰键盘背景的

LV_KB_STYLE_BTN_REL: 下面 5 个都是用来修饰键盘上按钮的

LV_KB_STYLE_BTN_PR,

LV_KB_STYLE_BTN_TGL_REL,

LV_KB_STYLE_BTN_TGL_PR,

LV_KB_STYLE_BTN_INA,

2.2.8 键盘默认的事件回调函数

```
void lv_kb_def_event_cb(lv_obj_t * kb, lv_event_t event);
```

参数:

kb: 键盘对象

event: 当前事件

当我们不设置事件回调函数时,那么此 API 接口就是键盘控件默认的事件回调函数,它会帮我们处理按键按下,按键值改变,管理绑定的文本域等操作,当然了你完全可以用你自定义的事件回调函数来覆盖掉它,但是为了减少一些相同事务的处理,我们可以在我们自定义的事件回调函数里的前面调用一下默认的 lv_kb_def_event_cb 事件回调函数.如以下示意代码所示:

```
lv_obj_t * kb1;
void my_event_handler(lv_obj_t * obj, lv_event_t event) // 自定义的事件回调函数
{
    if(obj == kb1) // 是键盘对象
    {
        // 调用键盘的默认事件回调函数,帮我们处理掉一些通用的操作
        lv_kb_def_event_cb(obj, event);
        /* 下面这里可以加你自己的功能代码 */
    }
}
```

2.2.9 备注

还有几个 get 获取类型的 API 接口我这里就不列举出来了,比较简单的

3. 例程设计

3.1 功能简介

先创建 2 个文本域控件,全部设置为单行模式,然后给第二个文本域控件使能密码保护功能,然后再接着创建一个键盘控件,设置其为文本键盘模式,对其中的小写键盘进行自定义改造,主要是在空格键的右边添加了一个@普通按键和一个 Clear 特殊功能按键,当点击@键时,是往相应的文本域中输入@字符,当点击 Clear 键时,是将相应的文本域中的内容给清空,最后我们给键盘设置自定义的事件回调函数,我们在 LV_EVENT_VALUE_CHANGED 事件中实现 Clear 特殊按键和自定义键盘的功能,在 LV_EVENT_APPLY 和 LV_EVENT_CANCEL 事件中把当前键盘给删除掉,当键盘被删除时,点击上面的文本域控件又可以把键盘给创建出来,当键盘已经存在时,点击某文本域控件就是将某文本域绑定到键盘上.

3.2 硬件设计

本例程所用到的硬件有:

- 1) 液晶屏

3.3 软件设计

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

```
#include "lv_kb_test.h"
#include "lvgl.h"
#include "key.h"
#include <string.h>

lv_obj_t * kb1;
lv_obj_t * ta1,* ta2;

/*
实现自定义键盘,可以参考 lv_kb.c 源码中的映射表的格式我们这里在官方自带的小写键盘基础上增加 2 个按键,都放到空格键的右边,一个是普通的@符号按键,另外一个输入内容清空键,即 Clear 键,这个键是特殊功能键,littleVGL 中不自带此特殊功能的键,所以我们得自定义此特殊键
*/

#define MY_KB_CTRL_BTN_FLAGS (LV_BTNUM_CTRL_NO_REPEAT | LV_BTNUM_CTRL_CLICK_TRIG) //无重复按下功能,选择松手触发
```



```
static const char * const my_kb_map_lc[] = {
    "1#", "q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "Bksp", "\n",
    "ABC", "a", "s", "d", "f", "g", "h", "j", "k", "l", "Enter", "\n",
    "_", "-", "z", "x", "c", "v", "b", "n", "m", ".", ",", ":", "\n",
    LV_SYMBOL_CLOSE,
    LV_SYMBOL_LEFT, " ",
    /*自己添加的键*/ "@",
    /*自己添加的键*/ "Clear",
    LV_SYMBOL_RIGHT,
    LV_SYMBOL_OK, ""};

static const lv_btmn_ctrl_t my_kb_ctrl_lc_map[] = {
    MY_KB_CTRL_BTN_FLAGS | 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 7,
    MY_KB_CTRL_BTN_FLAGS | 6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    MY_KB_CTRL_BTN_FLAGS | 2, 2,
    /*由原来的 6 改为了 3*/ 3,
    /*自己添加的键的控制属性*/ 1,
    /*自己添加的键的控制属性*/ MY_KB_CTRL_BTN_FLAGS | 2,
    2,
    MY_KB_CTRL_BTN_FLAGS | 2};

void kb_create(lv_obj_t * parent);

//事件回调函数
void event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(obj==ta1||obj==ta2)
    {
        if(event==LV_EVENT_RELEASED)//松手事件
        {
            if(kb1==NULL)
                kb_create(lv_scr_act());//如果键盘不存在的话,则先创建键盘
            else
                lv_kb_set_ta(kb1,obj);//重新绑定文本域对象
        }
    }else if(obj==kb1)
    {
        //获取按下键的文本标题,放到 lv_kb_def_event_cb 的前面调用
        const char * key_txt = lv_btmn_get_active_btn_text(kb1);
```

```
//调用键盘的默认事件回调函数,帮我们处理掉一些通用的操作,如果让我们自己写
//代码来处理,那就太麻烦了
lv_kb_def_event_cb(kb1,event);
//添加自己的功能代码
if(event==LV_EVENT_VALUE_CHANGED)
{
    //uint16_t key_id = *(uint16_t*)lv_event_get_data();//获取按下键的 id,第一个按键
    //的 id 为 0,后面的按键依次增 1
    if(key_txt==NULL)
        return;
    if(strcmp(key_txt,"Clear")==0)//按下的是清空键
    {
        lv_obj_t * bind_ta = lv_kb_get_ta(kb1);//获取当前绑定的文本域对象
        lv_ta_set_text(bind_ta,"");//将输入内容清空
    }else if(strcmp(key_txt,"abc")==0)//按下的是切换小写键盘键
    {
        //重定向到我们自己自定义的小写键盘,而不是系统自带的小写键盘
        lv_kb_set_map(kb1,(const char **)my_kb_map_lc);//设置自定义按键的映射表
        //设置自定义按键的属性控制映射表
        lv_kb_set_ctrl_map(kb1,my_kb_ctrl_lc_map);
    }
} else if(event==LV_EVENT_APPLY)
{
    /*
        这里可以根据用户输入的文本信息,做相应的业务逻辑处理
    */
    lv_obj_del(kb1);//点击 ✓ 键时把键盘删除掉
    kb1 = NULL;
} else if(event==LV_EVENT_CANCEL)
{
    /*
        这里也可以根据用户输入的文本信息,做相应的业务逻辑处理
    */
    lv_obj_del(kb1);//点击 × 键时把键盘删除掉
    kb1 = NULL;
}
}

//创建键盘
void kb_create(lv_obj_t * parent)
{
    kb1 = lv_kb_create(parent,NULL);
    lv_kb_set_cursor_manage(kb1,true);//使能对光标的接管
```

```
lv_kb_set_mode(kb1, LV_KB_MODE_TEXT); // 设置为文本键盘模式, 这也是默认值
// 先默认绑定 ta1 文本域对象, 后面可以通过点击某文本域来重新绑定到相应的文本域对象
lv_kb_set_ta(kb1, ta1);
lv_kb_set_map(kb1, (const char **)my_kb_map_lc); // 设置自定义按键的映射表
lv_kb_set_ctrl_map(kb1, my_kb_ctrl_lc_map); // 设置自定义按键的属性控制映射表
lv_obj_set_event_cb(kb1, event_handler); // 设置自定义的事件回调函数
}

// 例程入口
void lv_kb_test_start()
{
    lv_obj_t *scr = lv_scr_act(); // 获取当前活跃的屏幕对象
    lv_obj_set_click(scr, true);
    lv_obj_set_event_cb(scr, event_handler);

    // 1. 创建两个文本域对象
    // 1.1 创建 ta1 文本域对象
    ta1 = lv_ta_create(scr, NULL);
    lv_obj_set_width(ta1, 200); // 设置宽度
    lv_obj_align(ta1, NULL, LV_ALIGN_IN_TOP_MID, 0, 15);

    // 使能单行模式, 在单行模式下, 文本域的高度是不能被设置的
    lv_ta_set_one_line(ta1, true);
    lv_ta_set_text(ta1, ""); // 设置为空文本
    lv_obj_set_event_cb(ta1, event_handler); // 设置事件回调函数

    // 1.2 创建 ta2 文本域对象
    ta2 = lv_ta_create(scr, ta1); // 直接从 ta1 进行复制
    lv_ta_set_pwd_mode(ta2, true); // 使能密码模式

    // 先隐藏掉光标
    lv_ta_set_cursor_type(ta2, LV_CURSOR_LINE | LV_CURSOR_HIDDEN);
    lv_obj_align(ta2, ta1, LV_ALIGN_OUT_BOTTOM_MID, 0, 15);

    // 2. 创建键盘对象, 创建出来的键盘默认是与父对象底部居中对齐的,
    // 默认宽度等于父对象的可适应宽度(即除掉左右内边距后的宽度),
    // 默认高度等于父对象的可适应高度(即除掉上下内边距后的高度)的一半
    kb_create(scr);
}
```

3.4 下载验证

把代码下载进去之后,可以看到如下所示的初始界面效果:

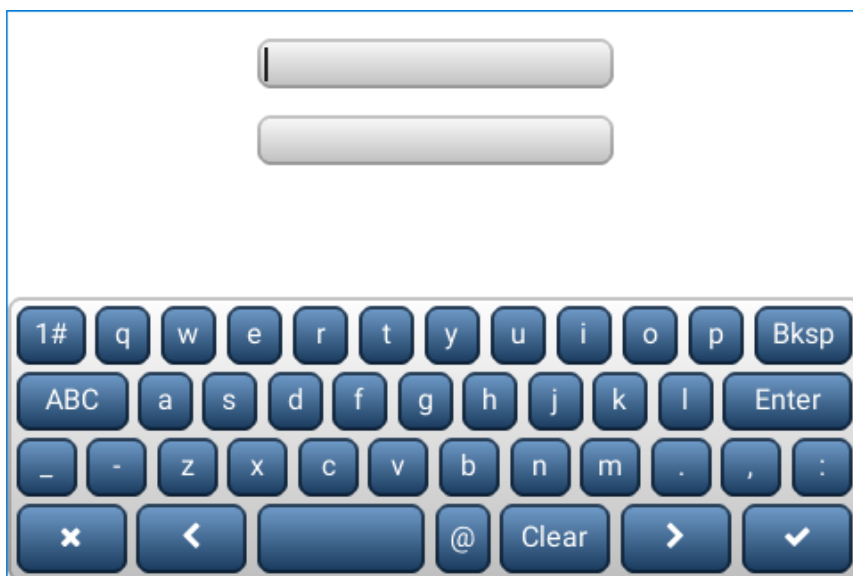




图 3.4.1 初始界面效果

如上图所示,可以看到空格键的右边有我们新增的@普通按键和 Clear 特殊功能按键,当点击@键时,是往相应的文本域中输入@字符,当点击 Clear 键时,是将相应的文本域中的内容给清空,当点击  键或者点击  键时,会把当前的键盘给删除掉,然后点击上面的文本域控件时又可以把键盘给创建出来了.当键盘已经存在时,点击某文本域控件就是将某文本域绑定到键盘上.

4. 资料下载

正点原子公司名称：广州市星翼电子科技有限公司

LittleVGL 资料连接：www.openedv.com/thread-309664-1-1.html

原子哥在线教学平台：www.yuanzige.com

正点原子淘宝店铺：<https://openedv.taobao.com>

正点原子官方网站：www.alientek.com

正点原子 B 站视频：<https://space.bilibili.com/394620890>

电话：020-38271790 传真：020-36773971

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

请关注正点原子公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子公众号