User Guide

正点原子 littleVGL 开发指南

lv_btnm 矩阵按钮

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_btnm 矩阵按钮

1. 介绍

lv_btnm 矩阵按钮对象你可以看作是一系列伪按钮的集合,只不过它是以行和列有序的方式来排列这些子按钮,名称中的 m 就是 matrix(矩阵)的缩写,注意了,我这里故意加了一个伪字来说明它不是真正的 lv_btn 按钮对象,而是 lv_btnm 内部纯绘制出来的具有按钮外观的图形,而且这个图形具有和 lv_btn 按钮一样的点击效果,这种伪按钮的好处是它基本不占内存消耗(一个伪按钮大概需要 8 个字节),这属于 littleVGL 的一种优化操作,所以如果你有多个按钮集合的应用场景的话,请最好使用 lv_btnm 控件,而不要去使用 lv_cont 容器外加 lv_btn 按钮的方案.

lv_btnm 控件可以给它内部的每个伪按钮设置文本标题以及设置相对的大小,然后它还可以通过 lv_btnm_set_btn_ctrl 接口来设置其内部按钮的各种控制属性,比如是否可见,是否处于禁用状态等等,除此之外 lv_btnm 具有和 lv_label 对象一样的文本重绘色功能,以及自己专有的“One toggle”特性,可以通过 lv_btnm_set_one_toggle 接口来使能“One toggle”特性,使能之后,在同一时刻就只能允许最多一个按钮处于切换态了.

当 lv_btnm 内部的按钮被点击或者被重复长按下时,它会给它的事件回调函数发送 LV_EVENT_VALUE_CHANGED 事件,于此同时这个按钮的 id 号会作为事件自定义数据传递过去,那我们在事件回调函数中可以通过如下代码的方式来获取按钮的 id 号:

```
uint16_t * btn_id = (uint16_t*)lv_event_get_data();
printf(“当前被按下的按钮 id 为:%d”, *btn_id);
```

除了这种方式外,我们还可以通过 lv_btnm_get_active_btn(btnm)接口来获取按钮的 id 号,注意了,当按钮被重复长按下时,它也是发送 LV_EVENT_VALUE_CHANGED 事件,而不是发送前面章节介绍过的 LV_EVENT_LONG_PRESSED_REPEAT 事件,如果你不想让按钮具有重复长按效果的话,那我们是可以通过控制属性来设置的.

最后我们来说一下 lv_btnm 对象内部按钮的排列方式,它是以从左往右,从上到下的方式来排列的,其内部的每一个按钮都具有一个唯一的 id 号,这个 id 号也是按照上面的排列方式来进行分配的,从 0 开始分配,然后依次增 1,整体上来说,lv_btnm 控件有点小复杂,但是只要你掌握了,就可以实现很强大的功能,我们后面要讲到的 lv_kb 键盘控件也是利用 lv_btnm 控件来实现的.

2. lv_btnm 的 API 接口

2.1 主要数据类型

3. 控制属性数据类型

```
enum {
    LV_BTNM_CTRL_HIDDEN         = 0x0008,
    LV_BTNM_CTRL_NO_REPEAT      = 0x0010,
    LV_BTNM_CTRL_INACTIVE       = 0x0020,
    LV_BTNM_CTRL_TGL_ENABLE     = 0x0040,
    LV_BTNM_CTRL_TGL_STATE      = 0x0080,
    LV_BTNM_CTRL_CLICK_TRIG     = 0x0100,
};
typedef uint16_t lv_btnm_ctrl_t;
```

LV_BTNM_CTRL_HIDDEN: 设置按钮为隐藏不可见状态
LV_BTNM_CTRL_NO_REPEAT: 设置按钮不具有重复长按的动作效果
LV_BTNM_CTRL_INACTIVE: 设置按钮为禁用状态
LV_BTNM_CTRL_TGL_ENABLE: 设置按钮为 Toggle 切换按钮
LV_BTNM_CTRL_TGL_STATE: 设置按钮当前的状态就是切换态
LV_BTNM_CTRL_CLICK_TRIG: 设置按钮的点击方式为松手触发,如果不设置的话,那默认就是按下时触发,二者只能选择一种

这些值之间是可以进行位或操作的

4. 矩阵按钮样式数据类型

```
enum {
    LV_BTNM_STYLE_BG,
    LV_BTNM_STYLE_BTN_REL,
    LV_BTNM_STYLE_BTN_PR,
    LV_BTNM_STYLE_BTN_TGL_REL,
    LV_BTNM_STYLE_BTN_TGL_PR,
    LV_BTNM_STYLE_BTN_INA,
};
typedef uint8_t lv_btnm_style_t;
```

我们在前面已经学过了 lv_btn 按钮的样式,所以再来学习矩阵按钮的样式就非常简单了,只有 LV_BTNM_STYLE_BG 这一个样式是用来修饰 lv_btnm 对象背景的,其他的 5 个样式全部是用来修饰其内部的伪按钮.

LV_BTNM_STYLE_BG: 修饰 lv_btnm 对象的背景,其中 body.padding 下的 left,top, right, bottom 是用来设置按钮与 lv_btnm 对象边框的各种内边距,然后 inner 字段是用来设置按钮与按钮之间的距离

2.2 API 接口

2.2.1 创建对象

```
lv_obj_t * lv_btnm_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 设置按钮映射表

```
void lv_btnm_set_map(const lv_obj_t * btnm, const char * map[]);
```

参数:
btnm: 矩阵按钮对象
map: 矩阵按钮对象就是根据这个 map 映射表来确定其内部的按钮个数以及按钮标题的,map 就是一个字符串数组,除了"\n"换行元素和""空串结尾元素外,其中的每一个元素都代表着一个按钮,字符串的内容就是这个按钮的文本标题,比如我想在其内部创建 4 个按钮,分为 2 行,上面 2 个,下面 2 个,那么 map 映射表具体如下:

```
const char * const    map[] = {"Btn1", "Btn2", "\n", "Btn3", "Btn4", ""};
```

其中的"\n"元素仅仅是用来换行的,没有其他的实际含义,然后 map 映射表的末尾必须要以一个""空串来作为结束标志,然后其默认样式效果如下:

图 2.2.2.1 map 映射效果

然后这里还有一点需要注意了,那就是 lv_btnm 矩阵按钮对象只是在内部简单的引用了 map 映射表,没有做拷贝操作的,所以你得保证这个 map 映射表资源在外部不能被释放了

2.2.3 设置某个按钮的控制属性

```
void lv_btnm_set_btn_ctrl(const lv_obj_t * btnm, uint16_t btn_id, lv_btnm_ctrl_t ctrl);
```

参数:

btnm: 矩阵按钮对象

btn_id: 设置哪一个按钮的控制属性,由此按钮的 id 来指定,注意”\n”换行元素不是按钮,因此它是没有按钮 id 的

ctrl: 控制属性,有如下 6 个可选值:

LV_BTNM_CTRL_HIDDEN:设置按钮为隐藏不可见状态

LV_BTNM_CTRL_NO_REPEAT:设置按钮不具有重复长按的动作效果

LV_BTNM_CTRL_INACTIVE:设置按钮为禁用状态

LV_BTNM_CTRL_TGL_ENABLE:设置按钮为 Toggle 切换按钮

LV_BTNM_CTRL_TGL_STATE:设置按钮当前的状态就是切换态

LV_BTNM_CTRL_CLICK_TRIG:设置按钮的点击方式为松手触发,如果不设置的话,那默认就是按下时触发,二者只能选择一种

这些属性值之间是可以进行位或操作的

2.2.4 给所有按钮设置共同的控制属性

```
void lv_btnm_set_btn_ctrl_all(lv_obj_t * btnm, lv_btnm_ctrl_t ctrl);
```

参数:

btnm: 矩阵按钮对象

ctrl: 控制属性,和 lv_btnm_set_btn_ctrl 接口中的含义相同

2.2.5 清除某个按钮的控制属性

```
void lv_btnm_clear_btn_ctrl(const lv_obj_t * btnm, uint16_t btn_id, lv_btnm_ctrl_t ctrl);
```

参数:

btnm: 矩阵按钮对象

btn_id: 按钮 id

ctrl: 控制属性,和 lv_btnm_set_btn_ctrl 接口中的含义相同

2.2.6 清除所有按钮的控制属性

```
void lv_btnm_clear_btn_ctrl_all(lv_obj_t * btnm, lv_btnm_ctrl_t ctrl);
```

参数:

btnm: 矩阵按钮对象

ctrl: 控制属性,和 lv_btnm_set_btn_ctrl 接口中的含义相同

2.2.7 设置某个按钮的相对宽度

```
void lv_btnm_set_btn_width(const lv_obj_t * btnm, uint16_t btn_id, uint8_t width);
```

参数:

btnm: 矩阵按钮对象

btn_id: 设置矩阵按钮对象中的哪一个按钮的相对宽度,由此按钮的 id 来指定,注意”\n”换行元素不是按钮,因此它是没有按钮 id 的

width: 指定在同一行按钮中此按钮所占的宽度比例,它是一个当前行的相对大小哦,并不是指定具体的宽度数值,此 width 参数的范围为[1,7],如果不设置的话,那么默认值就是 1,此值越大,那么在同一行中,此按钮具有的宽度也就越大,举个简单的例子,比如在某一行中具有 btn1 和 btn2 俩个按钮,btn1 的 width 值为 4,而 btn2 的 width 值为 2,那么最后 btn1 的在此行中的宽度是 btn2 的 4/2=2 倍,如下图所示:

图 2.2.4.1 相对宽度效果

就以上面这个例子为参考,其实这个 Btn1 或者 Btn2 的具体大小是可以推算出来的,假如我们通过 lv_obj_set_size 接口来设置 btnm 对象的大小,宽度为 btnm_w,高度为 btnm_h,而且 btnm 对象的背景样式为 bg_style,那么

Btn1 的宽度 = (btnm_w - bg_style.body.padding.left - bg_style.body.padding.right - bg_style.body.padding.inner)*2/3;

Btn1 的高度 = btnm_h - bg_style.body.padding.top - bg_style.body.padding.bottom;

Btn2 的宽度 = (btnm_w - bg_style.body.padding.left - bg_style.body.padding.right - bg_style.body.padding.inner)*1/3;

Btn2 的高度 = btnm_h - bg_style.body.padding.top - bg_style.body.padding.bottom;

2.2.8 设置控制映射表

```
void lv_btnm_set_ctrl_map(const lv_obj_t * btnm, const lv_btnm_ctrl_t ctrl_map[]);
```

参数:

btnm: 矩阵按钮对象

ctrl_map: 控制映射表,其中的每一个元素控制其相应的按钮,在每一个元素里面,你可以设置按钮的控制属性以及还有按钮的相对宽度,如下所示:

```
ctrl_map[0] = width | LV_BTNM_CTRL_NO_REPEAT | LV_BTNM_CTRL_TGL_ENABLE;
```

这个接口会在内部对 ctrl_map 进行拷贝操作的,所以 ctrl_map 这个资源在外面被释放掉是没有关系的

2.2.9 设置某个按钮为按下状态

```
void lv_btnm_set_pressed(const lv_obj_t * btnm, uint16_t id);
```

参数:
btnm: 矩阵按钮对象
btn_id: 按钮 id

2.2.10 设置样式

```
void lv_btnm_set_style(lv_obj_t * btnm, lv_btnm_style_t type, const lv_style_t * style);
```

参数:
btnm: 矩阵按钮对象
type: 要设置那部分的样式,有如下 6 个可选值
LV_BTNM_STYLE_BG
LV_BTNM_STYLE_BTN_REL
LV_BTNM_STYLE_BTN_PR
LV_BTNM_STYLE_BTN_TGL_REL
LV_BTNM_STYLE_BTN_TGL_PR
LV_BTNM_STYLE_BTN_INA
style: 样式

2.2.11 是否使能文本重绘色功能

```
void lv_btnm_set_recolor(const lv_obj_t * btnm, bool en);
```

参数:
btnm: 矩阵按钮对象
en: 是否使能重绘色,true 是使能,false 是不使能

矩阵按钮对象的文本重绘色功能和 lv_label 标签对象的文本重绘色功能是一样的,格式也是#十六进制颜色 文本# ,举个例子如下所示:

```
static const char * const btnm_map[] = {"#ff0000 Red# btn", "#00ff00 Green# btn",""};
```

2.2.12 是否使能 One toggle 特性

```
void lv_btnm_set_one_toggle(lv_obj_t * btnm, bool one_toggle);
```

参数:
btnm: 矩阵按钮对象
one_toggle: 是否使能 One toggle 特性,true 代表使能,false 代表不使能

使能 One toggle 特性之后,在所有被设置了 LV_BTNM_CTRL_TGL_ENABLE 控制属性的按钮中,同一时刻就只能允许最多一个按钮处于切换态了,所以要想看到正确的效果,这里是有一个前提的,那就是至少得有 2 个按钮设置了 LV_BTNM_CTRL_TGL_ENABLE 控制属性,即至少得有 2 个 Toggle 按钮

2.2.13 获取当前被点击的按钮 id

```
uint16_t lv_btnm_get_active_btn(const lv_obj_t * btnm);
```

参数:
btnm: 矩阵按钮对象

返回值:
返回当前被点击的按钮 id,如果获取失败,则返回 LV_BTNM_BTN_NONE

这里的 active 是指当前活跃的按钮,也就是被点击了的按钮,点击这俩个字就包含了 2 个含义,按下时触发的或者松手时触发的,至于到底是哪一种方式触发的,取决于它有没有设置 LV_BTNM_CTRL_CLICK_TRIG 控制属性,此 API 接口一般用在事件回调函数中,用来判断当前事件是被哪一个按钮触发的

2.2.14 获取当前被点击的按钮文本标题

```
const char * lv_btnm_get_active_btn_text(const lv_obj_t * btnm);
```

参数:
btnm: 矩阵按钮对象

返回值:
返回当前被点击的按钮文本标题,获取失败,则返回 NULL

此 API 接口一般用在事件回调函数中,也可以用来判断当前事件是被哪一个按钮触发的

2.2.15 获取当前被按下的按钮 id

```
uint16_t lv_btnm_get_pressed_btn(const lv_obj_t * btnm);
```

参数:
btnm: 矩阵按钮对象

返回值:
返回当前被按下的按钮 id,如果获取失败,则返回 LV_BTNM_BTN_NONE

此 API 接口和 lv_btnm_get_active_btn 接口功能差不多,但是此 API 接口更为具体一点,它指的是按下时,不包括松手时的情况

2.2.16 判断某个按钮是否已经设置过了某控制属性

```
bool lv_btnm_get_btn_ctrl(lv_obj_t * btnm, uint16_t btn_id, lv_btnm_ctrl_t ctrl);
```

参数:
btnm: 矩阵按钮对象
btn_id: 按钮 id
ctrl: 判断 ctrl 控制属性是否被设置过了

返回值:
返回 true 代表已经被设置过了,返回 false 代表没有被设置过

2.2.17 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

创建一个自定义样式来修饰 btnm 的背景,主要是修改 padding 字段来控制内边距,然后创建 btnm1 和 btnm2 俩个矩阵按钮对象,btnm1 是一个类似于主菜单效果的矩阵按钮,同时 btnm1 也用来演示按钮的各种控制属性,而 btnm2 是用来演示 One Toggle 特性的,当按下 KEY0 按键时是来回切换 AUDIO 按钮的 LV_BTNM_CTRL_HIDDEN 控制属性,当按下 KEY1 键时是来回切换 VIDEO 按钮的 LV_BTNM_CTRL_INACTIVE 控制属性,当按下 KEY2 键时是来回切换 HOME 按钮的 LV_BTNM_CTRL_NO_REPEAT 控制属性,当按下 WKUP 按键时是来回切换 SAVE 按钮的 LV_BTNM_CTRL_CLICK_TRIG 控制属性.

3.2 硬件设计

本例程所用到的硬件有:

1) 液晶屏
2) KEY0, KEY1, KEY2, WKUP 按键

3.3 软件设计

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

```
#include "lv_btnm_test.h"
#include "lvgl.h"
#include "key.h"
#include <stdio.h>

lv_obj_t * btnm1,* btnm2;
lv_obj_t * label1;
lv_style_t bg_style;

//第二个 const 表示把此映射表放在 flash 存储区,而不是 sram,可以减少内存的开销
static const char * const btnm1_map[] = {                //btnm1 的按钮映射表
        " "LV_SYMBOL_AUDIO"\n#ff0000 AUDIO#",
        " "LV_SYMBOL_VIDEO"\n#ff0000 VIDEO#",  //第一行放 2 个按钮
        "\n",                                                    //换行
        " "LV_SYMBOL_HOME"\n#ff0000 HOME#",
        " "LV_SYMBOL_SAVE"\n#ff0000 SAVE#",      //第二行放 2 个按钮
        ""                                                       //空字符串作为结束符
};
```

```
    static const char * const btnm2_map[] = {//btnm2 的按钮映射表
                                             "Btn1",
                                             "Btn2",
                                             "Btn3",
                                             "" //空字符串作为结束符
    };


    //事件回调函数
    static void event_handler(lv_obj_t * obj,lv_event_t event)
    {
        static uint32_t cnt = 0;//记录进入 LV_EVENT_VALUE_CHANGED 事件的次数
        char buff[100];
        const char * btn_title;
        uint16_t btn_id;

        if(event==LV_EVENT_VALUE_CHANGED)
        {
            //1.计数器增 1
            cnt++;

            //2.获取当前被点击了的按钮标题
            btn_title = lv_btnm_get_active_btn_text(obj);

            //3.获取当前被点击了的按钮 id
            btn_id = *((uint16_t*)lv_event_get_data());//方式 1
            //btn_id = lv_btnm_get_active_btn(obj);//方式 2

            //4.把事件信息显示在标签上
            sprintf(buff,"obj:%s,btn_id:%d,btn_title:%s,cnt:%d,",obj==btnm1?"btnm1":"btnm2
",btn_id,btn_title,cnt);
            lv_label_set_text(label1,buff);
        }
    }


    //例程入口
    void lv_btnm_test_start()
    {
        lv_obj_t * scr = lv_scr_act();//获取当前活跃的屏幕对象


        //1.创建一个自定义样式,用于修饰 lv_btnm 的背景
        lv_style_copy(&bg_style,&lv_style_pretty);
```

```
bg_style.body.padding.top = 5;//上内边距
bg_style.body.padding.bottom = 5;//底内边距
bg_style.body.padding.left = 15;//左内边距
bg_style.body.padding.right = 15;//右内边距
bg_style.body.padding.inner = 5;//按钮与按钮之间的距离

//2.创建一个类似于主菜单效果的 btnm1
btnm1 = lv_btnm_create(scr,NULL);
lv_obj_set_size(btnm1,160,160);//设置大小
lv_obj_align(btnm1,NULL,LV_ALIGN_IN_TOP_MID,0,10);//设置与屏幕的对齐方式
lv_btnm_set_map(btnm1,(const char**)btnm1_map);//设置按钮映射表
lv_btnm_set_recolor(btnm1,true);//使能颜色重绘
lv_btnm_set_style(btnm1,LV_BTNM_STYLE_BG,&bg_style);//设置背景样式
lv_obj_set_event_cb(btnm1,event_handler);//设置事件回调函数

//3.创建 btnm2,用来演示 One Toggle 特性
btnm2 = lv_btnm_create(scr,NULL);
lv_obj_set_size(btnm2,220,50);//设置大小
//设置与 btnm1 的对齐方式
lv_obj_align(btnm2,btnm1,LV_ALIGN_OUT_BOTTOM_MID,0,10);
lv_btnm_set_map(btnm2,(const char**)btnm2_map);//设置按钮映射表
lv_btnm_set_btn_width(btnm2,1,2);//设置 Btn2 按钮的宽度是其他按钮的 2 倍

//所有按钮都设置为 Toggle 按钮,至少得有 2 个以上的 Toggle 按钮才能看到 One
//Toggle 效果
lv_btnm_set_btn_ctrl_all(btnm2,LV_BTNM_CTRL_TGL_ENABLE);
lv_btnm_set_one_toggle(btnm2,true);//使能 One Toggle 特性

//让 Btn3 默认就处于 Toggle 状态
lv_btnm_set_btn_ctrl(btnm2,2,LV_BTNM_CTRL_TGL_STATE);
lv_obj_set_event_cb(btnm2,event_handler);//设置事件回调函数

//4.创建一个 label 标签用来显示信息
label1 = lv_label_create(scr,NULL);
lv_label_set_long_mode(label1,LV_LABEL_LONG_BREAK);//设置长文本模式
lv_obj_set_width(label1,220);//设置宽度

//设置与 btnm2 的对齐方式
lv_obj_align(label1,btnm2,LV_ALIGN_OUT_BOTTOM_MID,0,10);
lv_label_set_body_draw(label1,true);//使能背景绘制
lv_label_set_recolor(label1,true);//使能文本重绘色

//设置背景
lv_label_set_style(label1,LV_LABEL_STYLE_MAIN,&lv_style_plain_color);
```

```
    lv_label_set_text(label1,"Event info");
}

//来回切换 btnm1 中某个按钮的控制属性
static void btnm1_toggle_btn_ctrl(uint16_t btn_id,lv_btnm_ctrl_t ctrl)
{
    //判断此按钮是否已经设置了 ctrl 控制属性
    if(lv_btnm_get_btn_ctrl(btnm1,btn_id,ctrl))
        lv_btnm_clear_btn_ctrl(btnm1,btn_id,ctrl);//清除掉
    else
        lv_btnm_set_btn_ctrl(btnm1,btn_id,ctrl);//重新设置
}

//按键处理
void key_handler()
{
    u8 key = KEY_Scan(0);

    if(key==KEY0_PRES)
    {
        //来回切换 AUDIO 按钮的隐藏控制属性
        btnm1_toggle_btn_ctrl(0,LV_BTNM_CTRL_HIDDEN);
    }else if(key==KEY1_PRES)
    {
        //来回切换 VIDEO 按钮的禁用控制属性
        btnm1_toggle_btn_ctrl(1,LV_BTNM_CTRL_INACTIVE);
    }else if(key==KEY2_PRES)
    {
        //来回切换 HOME 按钮的禁用重复长按控制属性
        btnm1_toggle_btn_ctrl(2,LV_BTNM_CTRL_NO_REPEAT);
    }else if(key==WKUP_PRES)
    {
//来回切换 SAVE 按钮的松手触发控制属性,如果不设置的话,默认是按下时触发
        btnm1_toggle_btn_ctrl(3,LV_BTNM_CTRL_CLICK_TRIG);
    }
}
```

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:

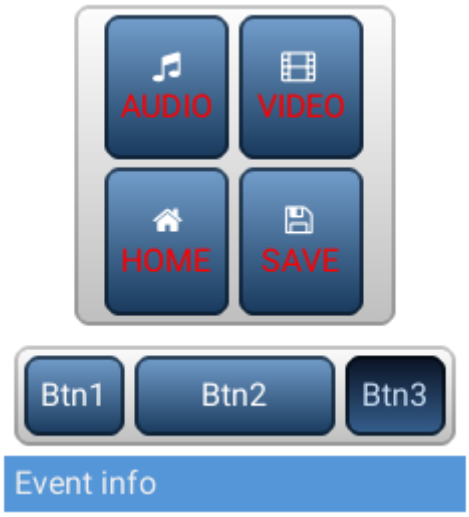

图 3.4.1 初始界面效果

然后我们随便点击一个按钮,比如 Btn2,会看到事件信息显示在底部的标签上,如下所示:

图 3.4.2 点击按钮之后的效果

接着我们分别按一下 KEY0,KEY1,KEY2,WKUP 按键,来设置它们相应的控制属性,效果如下:

图 3.4.3 控制属性效果

此时我们可以发现 AUDIO 按钮不可见了,VIDEO 按钮处于禁用状态,不能被点击了,通过长按 HOME 按钮可以发现这个长按动作触发不了事件了,通过点击 SAVE 按钮可以发现按下时不能触发事件了,而是变成了松手时触发

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