

# **TUI 使用说明文档**

修改记录 .....	5
1. TUI 概述 .....	6
2. 环境搭建 .....	6
3. 文件结构说明 .....	6
4. TUI 框架结构 .....	6
5. 全局功能 .....	7
5.1. 消息事件 .....	7
5.2. Timer 控制器 .....	7
5.2.1. 描述 .....	7
5.2.2. 纯代码测试用例 .....	8
5.3. 根节点对象 .....	9
6. 控件对象 .....	9
6.1. object 对象 .....	9
6.1.1. 描述 .....	9
6.1.2. 数据结构和函数 .....	9
6.2. container 容器 .....	12
6.2.1. 描述 .....	12
6.2.2. 数据结构和函数 .....	12
6.2.3. 纯代码测试用例 .....	13
6.2.4. 可视化工具编辑 .....	13
6.3. view 视图容器 .....	13
6.3.1. 描述 .....	13
6.3.2. 数据结构和函数 .....	14
6.3.3. 可视化工具编辑 .....	14
6.4. page 页 .....	14
6.4.1. 描述 .....	14
6.4.2. 数据结构和函数 .....	14
6.4.3. 纯代码测试用例 .....	15
6.4.4. 可视化工具编辑 .....	16
6.5. excel 表格 .....	16
6.5.1. 描述 .....	16
6.5.2. 数据结构和函数 .....	16
6.5.3. 纯代码测试用例 .....	18
6.5.4. 可视化工具编辑 .....	19
6.6. label 标签 .....	19
6.6.1. 描述 .....	19
6.6.2. 数据结构和函数 .....	20
6.6.3. 纯代码测试用例 .....	21
6.6.4. 可视化工具编辑 .....	21
6.7. arc 弧形 .....	22
6.7.1. 描述 .....	22
6.7.2. 数据结构和函数 .....	22
6.7.3. 纯代码测试用例 .....	23
6.7.4. 可视化工具编辑 .....	24
6.8. bar_progress 进度条 .....	24
6.8.1. 描述 .....	24
6.8.2. 数据结构和函数 .....	25
6.8.3. 纯代码测试用例 .....	25
6.8.4. 可视化工具编辑 .....	26
6.9. bar_slider 滑条 .....	27

6.9.1. 描述 .....	27
6.9.2. 数据结构和函数 .....	27
6.9.3. 纯代码测试用例 .....	28
6.9.4. 可视化工具编辑 .....	29
6.10. button 按钮 .....	30
6.10.1. 描述 .....	30
6.10.2. 数据结构和函数 .....	30
6.10.3. 纯代码测试用例 .....	31
6.10.4. 可视化工具编辑 .....	33
6.11. image_btn 图片按钮 .....	33
6.11.1. 描述 .....	33
6.11.2. 数据结构和函数 .....	33
6.11.3. 纯代码测试用例 .....	34
6.11.4. 可视化工具编辑 .....	35
6.12. image 图片 .....	36
6.12.1. 描述 .....	36
6.12.2. 数据结构和函数 .....	36
6.12.3. 纯代码测试用例 .....	38
6.12.4. 可视化工具编辑 .....	40
6.13. animation 动画 .....	40
6.13.1. 描述 .....	40
6.13.2. 数据结构和函数 .....	41
6.13.3. 纯代码测试用例 .....	42
6.13.4. 可视化工具编辑 .....	42
6.14. Gif 图片动画 .....	43
6.14.1. 描述 .....	43
6.14.2. 数据结构和函数 .....	43
6.14.3. 纯代码测试用例 .....	44
6.14.4. 可视化工具编辑 .....	44
6.15. line 线 .....	45
6.15.1. 描述 .....	45
6.15.2. 数据结构和函数 .....	45
6.15.3. 纯代码测试用例 .....	46
6.15.4. 可视化工具编辑 .....	47
6.16. switch_btn 切换开关 .....	47
6.16.1. 描述 .....	47
6.16.2. 数据结构和函数 .....	47
6.16.3. 纯代码测试用例 .....	48
6.16.4. 可视化工具编辑 .....	49
6.17. dropdown 下拉菜单 .....	50
6.17.1. 描述 .....	50
6.17.2. 数据结构和函数 .....	50
6.17.3. 纯代码测试用例 .....	51
6.17.4. 可视化工具编辑 .....	52
6.18. textbox 文本输入框 .....	52
6.18.1. 描述 .....	52
6.18.2. 数据结构和函数 .....	53
6.18.3. 纯代码测试用例 .....	53
6.18.4. 可视化工具编辑 .....	54
6.19. checkbox 复选框 .....	55
6.19.1. 描述 .....	55

6.19.2. 数据结构和函数 .....	55
6.19.3. 纯代码测试用例 .....	55
6.19.4. 可视化工具编辑 .....	56
6.20. list 列表 .....	56
6.20.1. 描述 .....	56
6.20.2. 数据结构和函数 .....	57
6.20.3. 纯代码测试用例 .....	58
6.20.4. 可视化工具编辑 .....	59
6.21. multi_screen 多屏控件 .....	60
6.21.1. 描述 .....	60
6.21.2. 数据结构和函数 .....	60
6.21.3. 纯代码测试用例 .....	61
6.21.4. 可视化工具编辑 .....	62
6.22. canvas 画布 .....	63
6.22.1. 描述 .....	63
6.22.2. 数据结构和函数 .....	63
6.22.3. 纯代码测试用例 .....	64
6.22.4. 可视化工具编辑 .....	65
6.23. Qrcode 二维码 .....	66
6.23.1. 描述 .....	66
6.23.2. 数据结构和函数 .....	66
6.23.3. 纯代码测试用例 .....	67
6.23.4. 可视化工具编辑 .....	67
7. 声音对象 .....	68
7.1. 描述 .....	68
7.2. 数据结构和函数 .....	68
7.3. 纯代码测试用例 .....	69
7.4. 可视化工具编辑 .....	71
8. 字体制作 .....	71
9. 多国语言 .....	72
10. 杂项 .....	72
10.1. 资源打包和读取 .....	72
10.2. 串口的使用 .....	73
10.3. 获取触摸和按键的信息 .....	73
10.4. 获取系统运行时间、RTC 日期、休眠 .....	73
10.5. Gb2312 和 utf8 码值转换 .....	73
10.6. 获得工程配置信息 .....	74
10.7. 获得 TUI 内核信息和自动化测试用于调试 .....	74
11. FAQ .....	74
12. 总结 .....	74

## 修改记录

版本	时间	内容	作者
v0.01	2020-12-12	初始版本	tui
v1.00	2021-12-28	更新函数说明	tui
v2.00	2022-02-01	初始版本	tui

# 1. TUI 概述

TUI 使用 C 语言开发，遵循 MVC 模式，业务、视图和控制分离，各个模块之间可以通过发送消息实现通讯，消息通过广播的形式发送，各模块都可以收到，可以进行筛选处理相应的消息。UI 采用单线程模式，在处理 UI 流程中不能长时间堵塞线程，否则会照成界面卡顿。

TUI 致力于打造跨平台的应用开发环境，开发者只需要关心自己的应用逻辑处理，嵌入式平台和 windows 平台代码可以 99.99% 的复用。用户将复杂的功能在 windows 系统下调试开发，能大大提高开发效率，减短开发周期，在开发过程中只需要使用 **tui.h** 提供的接口，实现自己需要的功能，然后移植到嵌入式平台编译，就能无缝衔接。

TUI 结构框架稳定，完美的实现了函数“初始化”、“运行”、“退出”，一个生命周期结束没有内存泄露，不会造成死机的异常问题。

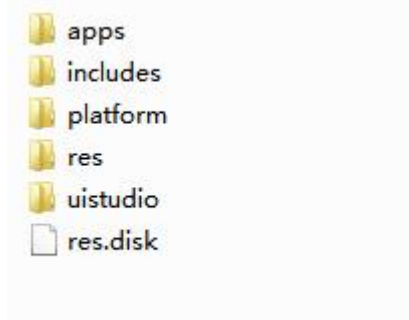
另外 **tui.h** 提供了丰富的接口，控件接口、时间、定时器、串口等跨平台接口，完全覆盖了应用开发需要的功能。能方便的移植到各个硬件平台，中间不需要修改系统接口代码。

# 2. 环境搭建

- 安装所见即所得 UI 工具 UISTudio.msi;
- 安装 PC 模拟编译环境 Visual Studio 2015 或者以上版本;
- 安装嵌入式编译环境;

# 3. 文件结构说明

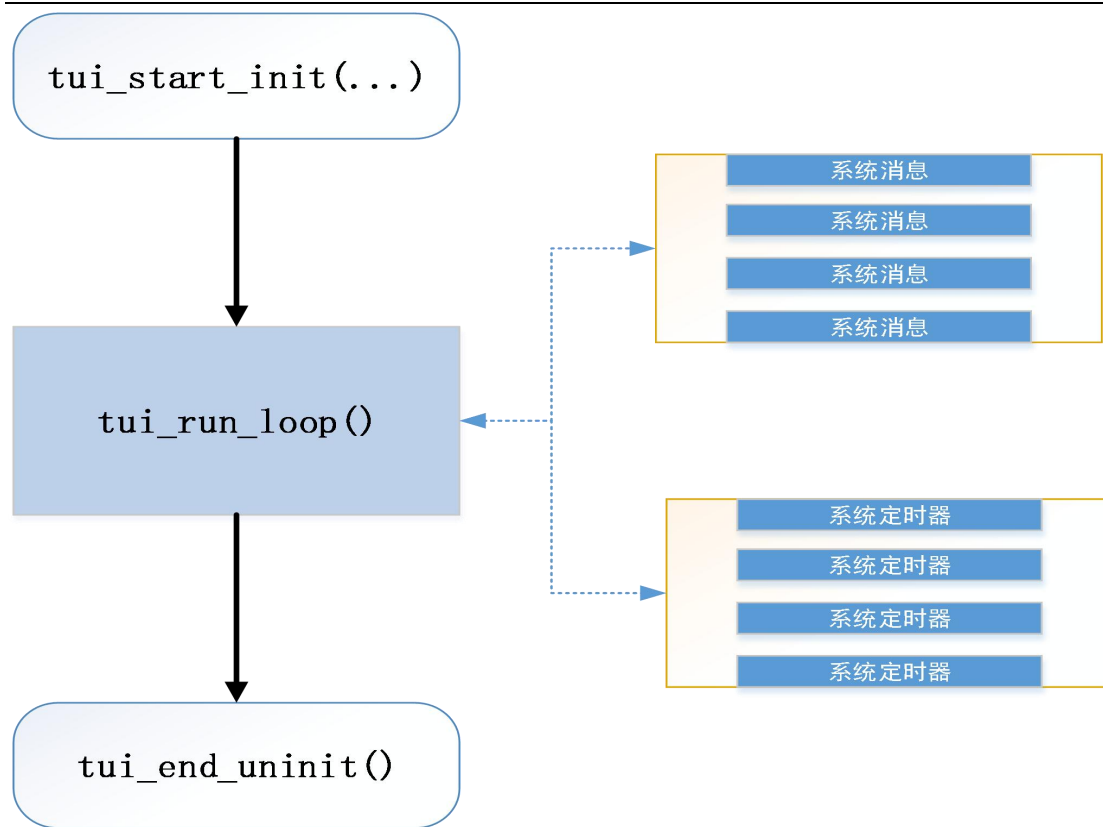
新建工程后会自动生成，如图的文件结构：



<b>apps</b>	文件夹存放应用视图和逻辑代码
<b>includes</b>	文件夹存放 TUI 接口头文件和消息枚举
<b>platform</b>	文件夹存放不同平台的入口程序和库文件
<b>res</b>	存放应用的所有资源，最后打包成 res.disk，最终在 GUI 里面映射成 V 盘
<b>uistudio</b>	存放的所见即所得工具的工程文件
<b>res.disk</b>	为 res 资源的打包文件，加载后在 GUI 里面映射成 V 盘

# 4. TUI 框架结构

框图如下：



- 入口函数 `void tui_start_init(const char * res_path)`; 参数 `res_path` 为资源 `res.disk` 的加载路径;
- 循环处理函数, TUI 的周期由它决定 `void tui_run_loop(void)`;
- TUI 退出函数, 进行资源回收, 生命周期结束 `void tui_end_uninit(void)`;
- 系统消息队列通过 `tui_sys_msg_reg()` 函数注册接收系统消息函数, 处理和发送相应的消息
- 定时器创建 `tui_timer_create()`, 注册相应的处理函数。

## 5. 全局功能

### 5.1. 消息事件

TUI 内部有两种类型消息, 系统消息事件和控件消息事件。

- 系统消息, 发送 TF 卡状态、电池状态、USB 状态等消息, 应用通过 `int32_t tui_sys_msg_reg(tui_sys_msg_cb_t cb)`; 注册回调函数, 接收系统消息, 然后做相应的处理。(注意用完注销回调函数), `int32_t tui_sys_msg_send(uint32_t cmd, void *param0, void *param1)`; 函数发送广播消息
- 发送系统消息, 可以通过 `int32_t tui_sys_msg_send(uint32_t cmd, void *param0, void *param1)`; 函数发送广播消息给注册函数, 消息立即处理, 对应回调函数就可以收到消息, 然后做相应的处理。
- 发送系统消息, 可以通过 `int32_t tui_sys_msg_send_dly(uint32_t cmd, uint32_t dly_ms, void *param0, void *param1)`; 函数发送广播消息给注册函数, 消息不立即发送, 稍微延迟后再发送, 对应回调函数就可以收到消息, 然后做相应的处理。

### 5.2. Timer 控制器

#### 5.2.1. 描述

TUI 提供了 timer 控制器, 防止堵塞, UI 异步处理, 定时处理。

```

/* timer 回调函数 */
typedef void (*tui_timer_cb_t)(tui_timer_t *timer);
/* 创建 timer */
tui_timer_t * tui_timer_create(tui_timer_cb_t timer_cb, uint32_t period, tui_timer_prio_e prio, void
                               * user_data);
/* 删除 timer */
void tui_timer_del(tui_timer_t * timer);
/* 定时器间隔时间复位重置 */
void tui_timer_reset(tui_timer_t * timer);
/* 中途设置定时器的回调函数 */
void tui_timer_set_cb(tui_timer_t * timer, tui_timer_cb_t timer_cb);
/* 中途设置定时器的时间戳 */
void tui_timer_set_period(tui_timer_t * timer, uint32_t period);
/* 中途设置定时器的优先等级 */
void tui_timer_set_prio(tui_timer_t * timer, tui_timer_prio_e prio);
/* 获得 timer 的用户传参 user_data */
void * tui_timer_get_user_data(tui_timer_t * timer);

```

## 5.2.2. 纯代码测试用例

```

/*      .0
 *
 *
 *.270      .90
 *
 *
 *      .180
 */
static void arc_loader(tui_timer_t * t)
{
    tui_arc_attri_t attri;

    tui_arc_get_attri(tui_timer_get_user_data(t), &attri);
    attri.start_angle = (attri.start_angle + 1) % 360;
    attri.end_angle = (attri.end_angle + 1) % 360;

    tui_arc_set_angles(tui_timer_get_user_data(t), attri.start_angle, attri.end_angle);
}

static tui_obj_t * tui_arc(void)
{
    /*Create an Arc*/
    tui_arc_attri_t attri = { 0 };
    tui_obj_t * arc;
    /* 创建对象 */
    arc = tui_arc_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 100;
    attri.obj.pt.y = 100;
    attri.obj.size.width = 120;

```



```

attri.obj.size.height = 120;
attri.start_angle = 100;      /* 弧形的开始角度 */
attri.end_angle = 190;        /* 弧形的结束角度 */
attri.bg_start_angle = 0;     /* 弧形背景的开始角度 */
attri.bg_end_angle = 359;     /* 弧形背景的结束角度 */
attri.line_width = 4;         /* 弧形的线宽 */
attri.bg_color = 0xFFFAFAFA;  /* 弧形的底色 (0xFF112233 FF 是透明度; 11 是 R; 22 是 G; 33 是 B) */
/*
attri.fg_color = 0xFFFF0000;  /* 弧形的前景色 (0xFF112233 FF 是透明度; 11 是 R; 22 是 G; 33 是 B) */
*/
tui_arc_set_attri(arc, &attri);
/* timer 动画 */
tui_timer_create(arc_loader, 10, TUI_TIMER_PRIO_LOWEST, arc);

return arc;
}

```

## 5.3. 根节点对象

TUI 里面万物皆对象，所有对象的根节点都是由两个特殊的对象图层继承而来：

- 通过函数 `tui_obj_t * tui_layer_normal(void)` 可以获得一般对象图层，用于普通对象继承的父节点。
- 通过函数 `tui_obj_t * tui_layer_top(void)` 可以获得置顶对象图层，用于对象视图常置顶，继承的父节点。

# 6. 控件对象

## 6.1. object 对象

### 6.1.1. 描述

所有控件包括声音控件，都是继承于 `object`，万物皆对象。一个对象包括通用属性和自身派生出来的属性参数，以及功能控制函数。

### 6.1.2. 数据结构和函数

对象属性结构如下，该属性将被所有控件继承

#### 对象属性

```

typedef struct {
    /* 供内部使用 style */
    tui_style_t style;

    tui_size_t size;          /* 外部配置，对象的宽高设置 */
    tui_point_t pt;           /* 外部配置，对象所在的位置 */
    bool hidden;              /* 外部配置，对象是否隐藏 */
    uint32_t obj_id;          /* 外部配置，对象的唯一 ID 值 */
    char type_name[32];       /* 外部配置，对象的控件类型名字， 不要超过 31 个英文字符 */
} tui_object_attri_t;

```

以下是所有对象的创建、销毁和控制函数，通过这些函数可以控制不同类型的控件。

## 创建销毁对象函数

/\* 创建对象指定派生的父亲对象 \*/

```
void tui_obj_create(tui_obj_t * parent);
```

/\* 删除 obj 对象及其子节点，资源回收，只要是对象都可以通过这个函数删除，它会搜索其子节点递归删除 \*/

```
void tui_obj_del(tui_obj_t * obj);
```

## 设置函数

/\* 设置对象的父节点 \*/

```
void tui_obj_set_parent(tui_obj_t * obj, tui_obj_t * parent);
```

/\* 设置对象的类型名字，不同控件定义不同的名字，方便查找 \*/

```
void tui_obj_set_typename(tui_obj_t * obj, const char *typename);
```

/\* 设置对象的唯一 ID，通过这个唯一 ID 号可以查找对应控件的句柄 \*/

```
void tui_obj_set_id(tui_obj_t * obj, uint32_t id);
```

/\* 设置对象的是否隐藏 \*/

```
void tui_obj_set_hidden(tui_obj_t * obj, bool able);
```

/\* 设置对象 X 坐标位置 \*/

```
void tui_obj_set_x(tui_obj_t * obj, tui_coord_t x);
```

/\* 设置对象 Y 坐标位置 \*/

```
void tui_obj_set_y(tui_obj_t * obj, tui_coord_t y);
```

/\* 设置对象的宽度 \*/

```
void tui_obj_set_width(tui_obj_t * obj, tui_coord_t w);
```

/\* 设置对象的高度 \*/

```
void tui_obj_set_height(tui_obj_t * obj, tui_coord_t h);
```

/\* 设置对象支持点击触发 \*/

```
void tui_obj_set_click(tui_obj_t * obj, bool able);
```

/\* 扩大对象触摸点击范围，用于滑条的触摸比较多，小按钮等 \*/

```
void tui_obj_set_ext_click_area(tui_obj_t * obj, tui_coord_t left, tui_coord_t right, tui_coord_t top, tui_coord_t bottom);
```

/\* 设置对象图层置顶 \*/

```
void tui_obj_set_top(tui_obj_t * obj, bool able);
```

/\* 设置对象可以拖拽，drag\_dir 支持拖拽的方向，查看对应的枚举定义 \*/

```
void tui_obj_set_drag_dir(tui_obj_t * obj, tui_drag_dir_e drag_dir);
```

/\* 设置对象可以拖拽惯性 \*/

```
void tui_obj_set_drag_throw(tui_obj_t * obj, bool able);
```

/\* 设置对象边框角，圆角弧度，radius(0~0x7fff) \*/

```
void tui_obj_set_border_radius(tui_obj_t * obj, int16_t radius);
```

/\* 设置对象边框颜色 \*/

```
void tui_obj_set_border_color(tui_obj_t * obj, uint32_t color);
```

/\* 设置对象边框线宽 \*/

```
void tui_obj_set_border_width(tui_obj_t * obj, uint32_t width);
```

/\* 设置对象边框的边显示 \*/

```
void tui_obj_set_border_side(tui_obj_t * obj, tui_border_side_e side);
```

/\* 设置对象背景颜色 \*/

```
void tui_obj_set_bg_color(tui_obj_t * obj, uint32_t color);
```

/\* 设置对象和父控件粘连在一起，一般用于 page，避免响应触摸点击拖拽动 \*/

```
void tui_obj_set_glue_obj(tui_obj_t * obj, bool able);
```

/\* 设置对象的透明度 alpha(0~0xff) \*/

```
void tui_obj_set_alpha(tui_obj_t * obj, uint8_t alpha);
```

/\* 设置对象 obj 基于 base 的对齐模式 align; x\_ofs 和 y\_ofs 是对齐后的偏移值; base 为 NULL 代表和父窗口对齐 \*/

```
void tui_obj_set_align(tui_obj_t * obj, const tui_obj_t * base, tui_align_e align, tui_coord_t x_ofs,
```

```

tui_coord_t y_ofs);
/* 停止对象动画播放, 图片 anim 也可以停止 */
void tui_obj_anim_stop(tui_obj_t * obj);
/* 设置对象淡入显示出来, 可以设置淡入的时间, end_cb 为动画结束回调, 不需要可以设置 NULL */
void tui_obj_anim_fade_in(tui_obj_t * obj, uint32_t need_time_ms, tui_object_anim_cb_t end_cb);
/* 设置对象淡出隐藏起来, 可以设置淡出的时间, end_cb 为动画结束回调, 不需要可以设置 NULL */
void tui_obj_anim_fade_out(tui_obj_t * obj, uint32_t need_time_ms, tui_object_anim_cb_t end_cb);
/* 设置对象 X 坐标动画, 可以设置动画的时间, 和运动轨迹, end_cb 为动画结束回调, 不需要可以设置 NULL */
void tui_obj_anim_mov_x(tui_obj_t * obj, uint32_t need_time_ms, tui_coord_t start_x, tui_coord_t end_x,
tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置对象 Y 坐标动画, 可以设置动画的时间, 和运动轨迹, end_cb 为动画结束回调, 不需要可以设置 NULL */
void tui_obj_anim_mov_y(tui_obj_t * obj, uint32_t need_time_ms, tui_coord_t start_y, tui_coord_t end_y,
tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置对象宽度动画, 可以设置动画的时间, 和运动轨迹, end_cb 为动画结束回调, 不需要可以设置 NULL */
void tui_obj_anim_set_width(tui_obj_t * obj, uint32_t need_time_ms, tui_coord_t start_width, tui_coord_t
end_width, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置对象高度动画, 可以设置动画的时间, 和运动轨迹, end_cb 为动画结束回调, 不需要可以设置 NULL */
void tui_obj_anim_set_height(tui_obj_t * obj, uint32_t need_time_ms, tui_coord_t start_height,
tui_coord_t end_height, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置自定义动画, 可以设置动画的时间, 和运动轨迹, end_cb 为动画结束回调, 不需要可以设置 NULL, value_cb
是自定义回调函数 */
void tui_obj_anim_set_vaule(tui_obj_t * obj, uint32_t need_time_ms, int32_t start_value, int32_t end_value,
tui_anim_path_e path_type, tui_object_anim_value_cb_t value_cb, tui_object_anim_cb_t end_cb);
/* 设置切换两个视图的动画, 可以设置动画的效果和时间, 和是否删除旧视图, end_cb 为动画结束回调, 不需
要可以设置 NULL */
void tui_screen_load_anim(tui_obj_t * new_scr, tui_obj_t * old_scr, tui_scr_load_anim_e anim_type,
uint32_t need_time_ms, bool auto_del_old_scr, tui_object_anim_cb_t end_cb);
/* 扩展设置进入 APP 视图的动画, 可以设置动画的效果和时间, 和是否删除旧视图, end_cb 为动画结束回调,
不需要可以设置 NULL */
void tui_screen_load_anim_in_ext(tui_obj_t * new_scr, tui_obj_t * old_scr, tui_point_t anim_start_pt,
uint32_t need_time_ms, bool auto_del_old_scr, tui_object_anim_cb_t end_cb);
/* 扩展设置退出 APP 视图的动画, 可以设置动画的效果和时间, 和是否删除旧视图, end_cb 为动画结束回调,
不需要可以设置 NULL */
void tui_screen_load_anim_out_ext(tui_obj_t * new_scr, tui_obj_t * old_scr, tui_point_t
anim_start_pt, uint32_t need_time_ms, bool auto_del_old_scr,
tui_object_anim_cb_t end_cb);

```

## 获得函数

```

/* 通过父节点 parent, 搜索其子节点对应 obj_id 的句柄, 比较快 */
tui_obj_t * tui_get_obj_from_id(tui_obj_t * parent, uint32_t obj_id);
/* 全局搜索所有子节点对应 obj_id 的句柄, 控件非常多的时候比较耗时 */
tui_obj_t * tui_search_obj_from_id(uint32_t obj_id);
/* 获得对象的类型名字 */
const char * tui_obj_get_typename(tui_obj_t * obj);
/* 获得对象的唯一 ID 值 */
uint32_t tui_obj_get_id(tui_obj_t * obj);
/* 获得对象是否隐藏 */
bool tui_obj_get_hidden(tui_obj_t * obj);
/* 获得对象的 X 坐标值 */
tui_coord_t tui_obj_get_x(tui_obj_t * obj);
/* 获得对象的 Y 坐标值 */
tui_coord_t tui_obj_get_y(tui_obj_t * obj);

```

```

/* 获得对象的宽度 */
tui_coord_t tui_obj_get_width(tui_obj_t * obj);
/* 获得对象的高度 */
tui_coord_t tui_obj_get_height(tui_obj_t * obj);
/* 获得对象的区域坐标 */
void tui_obj_get_coords(const tui_obj_t * obj, tui_area_t * cords_p);
/* 获得对象的父节点 */
tui_obj_t * tui_obj_get_parent(const tui_obj_t * obj);
/*
 * 获得对象的子节点，由于子节点可能有多个，如果只想得到第一个，child 可以设置成 NULL，
 * 否则设置成子节点的兄弟节点。这是一个典型的树状结构搜索。
 */
tui_obj_t * tui_obj_get_child(const tui_obj_t * obj, const tui_obj_t * child);

```

## 6.2. container 容器

### 6.2.1. 描述

容器一般用于整合多个控件，使多个控件有自己的父节点，便于统一管理。如工具生成的每一个视图，都是一个容器，每个控件的坐标系统，都是在这个容器里面，然后进行管理，容器被释放后，所有的控件也随之被释放。容器移动，所有控件也会统一移动，控制灵活，方便。

容器的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个容器控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.2.2. 数据结构和函数

#### 容器属性

```

typedef struct {
    /* 通用属性 */
    tui_object_attri_t obj;
    /* 组件属性 */
    void *attri_com;
    /* 容器回调函数，返回当事件 */
    tui_container_cb_t cb;

    uint32_t bg_color;      /* 外部配置，容器的背景颜色（0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B） */
} tui_container_attri_t;

```

#### 容器回调函数

```

typedef void (*tui_container_cb_t)(tui_obj_t *obj, tui_event_e event);

```

#### 功能函数

```

/* 创建容器，par 是其父节点 */
tui_obj_t * tui_container_create(tui_obj_t * par);
/* 设置容器的属性 */
int tui_container_set_attri(tui_obj_t *container, tui_container_attri_t *attri);
/* 获得容器的属性 */
int tui_container_get_attri(tui_obj_t *container, tui_container_attri_t *attri);

```

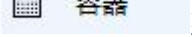
```
/* 每个视图是一个容器，可以手动设置容器的事件回调函数 */
int tui_container_set_cb(tui_obj_t *container, tui_container_cb_t cb)
```

### 6.2.3. 纯代码测试用例

```
static void tui_container(void)
{
    /* Create an container */
    tui_container_attr_t attr_container = { 0 };
    tui_obj_t * container;

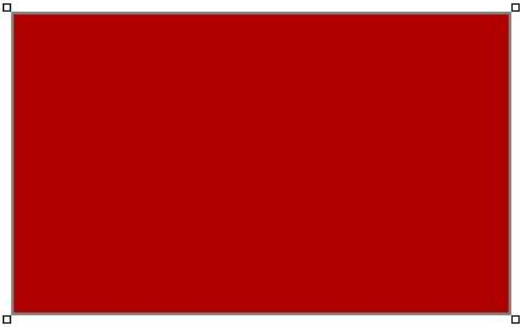
    /* 创建对象 */
    container= tui_container_create(tui_layer_normal());
    /* 设置属性 */
    attr_container.obj.pt.x = 10;
    attr_container.obj.pt.y = 10;
    attr_container.obj.size.width = 400;
    attr_container.obj.size.height = 400;
    attr_container.bg_color = 0xFFFAFAFA; /* 容器的背景颜色 (0xFF112233 FF是透明度; 11
是R; 22是G; 33是B) */
    tui_container_set_attr(container, &attr_container);
}
```

### 6.2.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  到视图中，设置对应的参数，如下：

控件类型 container

控件名称 container\_3



控件参数

名称	值
控件名称	container_3
类型	container
ID值	3
位置	
X	209
Y	54
宽高	
宽度	358
高度	217
是否隐藏	false
背景颜色	-5177344

## 6.3. view 视图容器

### 6.3.1. 描述

视图容器，其实也是一个容器，只是这个容器是通过 UIStudio 工具配置好的。我们只需要传入配置文件，就可以得到一个视图容器控件，可喜的是这个传入的事情也被工具做了，可查看对应视图生成的文件。

### 6.3.2. 数据结构和函数

#### 回调函数

```
/*
    创建视图函数，这个函数工具会自动生成对应的代码，用户不需要写。
    layout_view_path 为配置视图文件路径
    map_cb 视图里面控件的回调函数 map 表
*/
tui_obj_t * tui_view_create(const char *layout_view_path, tui_map_cb_t map_cb[]);
```

### 6.3.3. 可视化工具编辑

点击左列表视图，设置对应的参数，如下：

控件参数	
名称	值
ID	1
名称	main_view
宽度	1024
高度	600
X坐标	0
Y坐标	0
是否置顶	否
背景颜色	-1
边框颜色	0
控件数量	1
保存路径	D:\\tui_test\\res\\layout\\home\\main_view.lo

## 6.4. page 页

### 6.4.1. 描述

做更炫酷的视图界面，页控件是必不可少的，页控件其实是由一个基础控件和一个无限大的容器控件组合而成。其中无限大的窗口大小是一个容器，当这个容器大小超过了基础控件长宽的时候，基础控件会产生水平和垂直的滑条，控制滑条可以定位到相应容器的位置。

页的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个页控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.4.2. 数据结构和函数

#### 页属性

```
typedef struct {
```

```

/* 通用属性 */
tui_object_attri_t obj;
/* 页回调函数，返回当事件 */
tui_page_cb_t cb;

uint32_t bg_color;      /* 外部配置，页里面的背景颜色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
tui_size_t window_size; /* 外部配置，页里面的窗口的大小 */
tui_point_t window_pt;  /* 外部配置，页里面的窗口位置*/
tui_scrollbar_mode_e mode; /* 外部配置，页里面的滚条模式设置 */
bool edge_arc_able;      /* 外部配置，页到达边缘动画设置 */
} tui_page_attri_t;

```

### 页回调函数

```
typedef void (*tui_page_cb_t)(tui_obj_t *obj, tui_event_e event);
```

### 功能函数

```

/* 创建页，par 是其父节点 */
tui_obj_t * tui_page_create(tui_obj_t * par);
/* 设置页的属性 */
int tui_page_set_attri(tui_obj_t *page, tui_page_attri_t *attri);
/* 获得页的属性 */
int tui_page_get_attri(tui_obj_t *page, tui_page_attri_t *attri);
/* 获得页的窗口容器句柄，通过句柄可以使用对象控制函数，控制窗口容器的宽高和位置等 */
tui_obj_t * tui_page_get_window_obj(const tui_obj_t *page);
/* 设置页的滚条的模式 */
void tui_page_set_scrollbar_mode(tui_obj_t * page, tui_scrollbar_mode_e page_bar_mode);
/* 设置页的滚条的到末端后的弧形动画 */
void tui_page_set_edge_arc(tui_obj_t * page, bool able);

```

## 6.4.3. 纯代码测试用例

```

static void tui_page(void)
{
    /* Create an page */
    tui_page_attri_t attri_page = { 0 };
    tui_obj_t * page;

    /* 创建对象 */
    page = tui_page_create(tui_layer_normal());
    /* 设置属性 */
    attri_page.obj.pt.x = 10;
    attri_page.obj.pt.y = 10;
    attri_page.obj.size.width = 345;
    attri_page.obj.size.height = 268;
    attri_page.bg_color = 0xFFFAFAFA; /* 页里面的背景颜色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    attri_page.window_size.width = 1024; /* 页里面的窗口的大小 */
    attri_page.window_size.height = 600; /* 页里面的窗口的大小 */
    attri_page.window_pt.x = -100; /* 页里面的窗口位置*/
    attri_page.window_pt.y = -100; /* 页里面的窗口位置*/
}


```



```
tui_page_set_attri(page, &attri_page);


tui_page_set_scrollbar_mode(page, TUI_SCROLLBAR_MODE_ON);
}
```

## 6.4.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  页 到视图中，设置对应的参数，如下：

控件类型 page

控件名称 page\_4



▼ 控件参数

名称	值
控件名称	page_4
类型	page
ID值	4
位置	
X	314
Y	35
宽高	
宽度	354
高度	278
是否隐藏	false
背景颜色	-7016208
窗口宽高	
窗口宽度	400
窗口高度	400
窗口位置	
X	0
Y	0
末端圆弧动画	1
滚条模式	2

函数名称	启用
page_callback	<input type="checkbox"/>

## 6.5. excel 表格

### 6.5.1. 描述

表格控件类似简单 excel，提供数据表格化展示。控件里面的每一个表格，可以设置字体的大小、颜色、背景色，以及边框的颜色，支持中文输入等。由于嵌入式系统能力的限制，表格的数量不要超过 400，如果超过可以通过分页的显示展示。

表格的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个表格控件，设置属性时，必填属性会通过 Uistudio 工具配置。

### 6.5.2. 数据结构和函数

#### 表格属性

```
typedef struct {
```



```

/* 通用属性 */
tui_object_attri_t obj;
/* 表格回调函数，返回当事件 */
tui_excel_cb_t cb;
/* 表格里面的对象集合，供内部使用 */
uint32_t *cells;
/* 表格里面的按键，供内部使用 */
bool pressed;

uint32_t bg_color; /* 外部配置，表格里面的背景颜色（0xFF112233 FF是透明度；11是R；22是G；
33是B） */
uint32_t border_color; /* 外部配置，表格里面的边框颜色（0xFF112233 FF是透明度；11是R；22是G；
33是B） */
tui_point_t window_pt; /* 外部配置，表格里面的窗口位置 */
tui_scrollbar_mode_e mode; /* 外部配置，表格里面的滚条模式设置 */
bool edge_arc_able; /* 外部配置，表格到达边缘动画设置 */
uint32_t rows; /* 外部配置，表格的行函数 */
uint32_t columns; /* 外部配置，表格的列函数 */
uint32_t cell_w; /* 外部配置，表格的单元格宽 */
uint32_t cell_h; /* 外部配置，表格的单元格高 */
bool input_able; /* 外部配置，支持单元格字符输入 */
int32_t fnt_size; /* 外部配置，表格的默认字体大小 */
uint32_t fnt_color; /* 外部配置，表格的默认字体的颜色（0xFF112233 FF是透明度；11是R；22是
G；33是B） */

/* 表格里面的对象集合，供内部使用 */
uint32_t *cells;
/* 表格里面的按键，供内部使用 */
bool pressed;
} tui_excel_attri_t;

```

## 表格回调函数

```
typedef void(*tui_excel_cb_t)(tui_obj_t *obj, tui_event_e event, uint32_t column, uint32_t row);
```

## 功能函数

```

/* 创建表格，par 是其父节点 */
tui_obj_t * tui_excel_create(tui_obj_t * par);
/* 设置表格的属性 */
int tui_excel_set_attri(tui_obj_t *excel, tui_excel_attri_t *attri);
/* 得到表格的属性 */
int tui_excel_get_attri(tui_obj_t *excel, tui_excel_attri_t *attri);
/* 获得表格的窗口容器句柄，通过句柄可以使用对象控制函数，控制窗口容器的宽高和位置等 */
tui_obj_t * tui_excel_get_window_obj(const tui_obj_t *excel);
/* 设置表格的滚条的模式 */
void tui_excel_set_scrollbar_mode(tui_obj_t * excel, tui_scrollbar_mode_e excel_bar_mode);
/* 设置表格的滚条的到末端后的弧形动画 */
void tui_excel_set_edge_arc(tui_obj_t * excel, bool able);
/* 设置表格里面单元格的属性和字符 */
int tui_excel_set_cell_text(tui_obj_t *excel, uint32_t row, uint32_t column, const char* txt,
uint16_t font_size, uint32_t font_color, tui_label_align_e align,
tui_label_long_mode_e mode, uint32_t bg_color);

```

### 6.5.3. 纯代码测试用例

```

void tui_excel_cb(tui_obj_t *obj, tui_event_e event, uint32_t column, uint32_t row)
{
    char str[20];

    if (row > 0 && column > 0) {
        sprintf(str, "%d,%dhellodddd", column, row);

        tui_excel_set_cell_text(obj, row, column, str, 30, 0xFFFFFFFF00, TUI_LABEL_ALIGN_LEFT,
TUI_LABEL_LONG_SROLL_CIRC, 0xFFFF0000);
    }
}

static void tui_excel(void)
{
    int i;
    char str[5];
    /* Create an excel */
    tui_excel_attri_t attri_excel = {0 };
    tui_obj_t * excel;

    /* 创建对象 */
    excel = tui_excel_create(tui_layer_normal());
    /* 设置属性 */
    attri_excel.obj.pt.x = 10;
    attri_excel.obj.pt.y = 10;
    attri_excel.obj.size.width = 320;
    attri_excel.obj.size.height = 400;
    attri_excel.bg_color = 0xFFFFFFFF; /* 表格的背景颜色 (0xFF112233 FF是透明度; 11是R; 22
是G; 33是B) */
    attri_excel.border_color = 0xFF000000; /* 表格里面的边框颜色 (0xFF112233 FF是透明度; 11
是R; 22是G; 33是B) */
    attri_excel.window_pt.x = 0; /* 表格的窗口位置*/
    attri_excel.window_pt.y = 0; /* 表格的窗口位置*/

    attri_excel.cb = tui_excel_cb;
    attri_excel.rows = 100; /* 表格的行函数 */
    attri_excel.columns = 4; /* 表格的列函数 */
    attri_excel.cell_w = 80; /* 表格的单元格宽 */
    attri_excel.cell_h = 30; /* 表格的单元格高 */
    tui_excel_set_attri(excel, &attri_excel);
    tui_excel_set_scrollbar_mode(excel, TUI_SCROLLBAR_MODE_DRAG);
    tui_excel_set_edge_arc(excel, 1);

    tui_excel_set_cell_text(excel, 0, 0, "", 30, 0xFF7F7F7F, TUI_LABEL_ALIGN_CENTER,
TUI_LABEL_LONG_SROLL_CIRC, 0xFFDFDFDF);
    tui_excel_set_cell_text(excel, 0, 1, "A", 30, 0xFF7F7F7F, TUI_LABEL_ALIGN_CENTER,
TUI_LABEL_LONG_SROLL_CIRC, 0xFFDFDFDF);
    tui_excel_set_cell_text(excel, 0, 2, "B", 30, 0xFF7F7F7F, TUI_LABEL_ALIGN_CENTER,
TUI_LABEL_LONG_SROLL_CIRC, 0xFFDFDFDF);

```

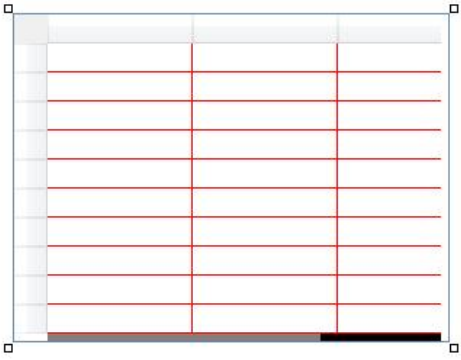
```

tui_excel_set_cell_text(excel, 0, 3, "C", 30, 0xFF7F7F7F, TUI_LABEL_ALIGN_CENTER,
TUI_LABEL_LONG_SCROLL_CIRC, 0xFFDFDFDF);
for (i = 1; i < 50; i++) {
    sprintf(str, "%d", i);
    tui_excel_set_cell_text(excel, i, 0, str, 30, 0xFF7F7F7F, TUI_LABEL_ALIGN_CENTER,
TUI_LABEL_LONG_SCROLL_CIRC, 0xFFDFDFDF);
}
}

```

## 6.5.4. 可视化工具编辑

拖拽<< 控件栏 列表中的 表格控件 到视图中，设置对应的参数，如下：

控件类型 excel		控件名称 excel_5			
				▼ 控件参数	
				名称	值
控件名称		excel_5			
类型		excel			
ID值		5			
位置					
X		365			
Y		88			
宽高					
宽度		301			
高度		227			
是否隐藏		false			
窗口位置					
X		0			
Y		0			
滚动条模式		0			
边缘动画		1			
列数		4			
行数		10			
单元格宽度		100			
单元格高度		30			
风格					
背景颜色		-1			
边框颜色		-65536			
函数名称		启用			
excel_callback		<input type="checkbox"/>			

## 6.6. label 标签

### 6.6.1. 描述

文本控件是最常用的基础控件，主要用了字符标识，该控件提供了常用的功能，如字体，大小，颜色，对齐模式，还支持长字符显示模式，标签支持静态和动态标签，动态标签可以支持中文输入。

标签的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个标签控件，设置属性时，必填属性会通过 UIStudio 工具配置。

## 6.6.2. 数据结构和函数

### 标签属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 弧形回调函数，返回当事件 */
    tui_label_cb_t cb;

    int32_t fnt_size;      /* 外部配置，标签字体大小 */
    char* txt;             /* 外部配置，标签的文本信息 */
    uint32_t fnt_color;    /* 外部配置，标签字体的颜色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    tui_label_long_mode_e mode; /* 外部配置，标签显示模式（其中有滚动显示） */
    tui_label_align_e align; /* 外部配置，标签对齐方式 */
    bool input_able;      /* 外部配置，支持标签字符输入 */
    bool ttf_font;        /* 外部配置，支持TTF字体 */
    uint32_t border_color; /* 外部配置，标签的边框颜色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
} tui_label_attr_t;
```

### 标签回调函数

```
typedef void (*tui_label_cb_t)(tui_obj_t *obj, tui_event_e event);
```

### 功能函数

```
/* 创建标签，par 是其父节点 */
tui_obj_t * tui_label_create(tui_obj_t * par);
/* 设置标签的属性 */
int tui_label_set_attr(tui_obj_t *label, tui_label_attr_t *attri);
/* 获得标签的属性 */
int tui_label_get_attr(tui_obj_t *label, tui_label_attr_t *attri);
/* 可以通过这个函数修改字符内容 */
int tui_label_set_txt(tui_obj_t *label, const char *txt);
/* 可以通过这个函数修改字体颜色 */
void tui_label_set_txt_color(tui_obj_t *label, uint32_t color);
/* 可以通过这个函数修改指定字符颜色颜色(#0000ff hello#) */
void tui_label_set_recolor(tui_obj_t * label, bool en);
/* 可以通过这个函数修改长字符模式属性 */
void tui_label_set_long_mode(tui_obj_t * label, tui_label_long_mode_e long_mode);
/* 可以通过这个函数修改字符对齐模式属性 */
void tui_label_set_align(tui_obj_t * label, tui_label_align_e align);
/* 可以通过这个函数使能字符上下对齐模式属性 */
void tui_label_set_align_mid(tui_obj_t * label, bool able);
/* 可以通过这个函数使能字符底部对齐模式属性 */
void tui_label_set_align_bottom(tui_obj_t * label, bool able);
/* 可以通过这个函数修设置是否支持键盘动态输入 */
void tui_label_set_input_able(tui_obj_t * label, bool able);
```


### 6.6.3. 纯代码测试用例

```
static tui_obj_t * tui_label(void)
{
    /*Create an label*/
    tui_label_attr_t attri_label = { 0 };
    tui_obj_t * label;
    /* 创建对象 */
    label = tui_label_create(tui_layer_normal());
    /* 设置属性 */
    attri_label.obj.pt.x = 0;
    attri_label.obj.pt.y = 10;
    attri_label.obj.size.width = 100;
    attri_label.obj.size.height = 40;
    attri_label.fnt_size = 15;          /* 标签字体大小 */
    attri_label.txt = "hello";          /* 标签的文本信息 */
    attri_label.fnt_color = 0xFF0000;   /* 标签字体的颜色 (0xFF112233 FF是透明度; 11是R; 22是G; 33是B) */
    attri_label.mode = 4;               /* 标签显示模式 (其中有滚动显示) */
    attri_label.align = TUI_LABEL_ALIGN_CENTER; /* 标签对齐方式 */
    tui_label_set_attri(label, &attri_label);

    return label;
}
```

### 6.6.4. 可视化工具编辑

拖拽<< 控件栏 列表中的 **A 标签** 到视图中，设置对应的参数，如下：

控件类型 label      控件名称 label_6		▼ 控件参数
	名称	值
	控件名称	label_6
	类型	label
	ID值	6
	位置	
	X	390
	Y	196
	宽高	
	宽度	177
	高度	66
	是否隐藏	false
	对齐方式	0
	字体	
	名称	1
	大小	30
颜色	65536	
文本		
内容	hello	
长文本对齐模式	4	

## 6.7. arc 弧形

### 6.7.1. 描述

弧形是两个圆形圈，一个底图圆圈，一个上面的圆圈，这两个圆圈可以设置线宽和颜色，也可以控制画圆圈的弧度，一般用来做加载提示比较好。弧形的角度标记如下：

```
.0
.270  .90
.180
```

弧形的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个弧形控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.7.2. 数据结构和函数

弧形属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attri_t obj;
    /* 弧形回调函数，返回当事件 */
    tui_arc_cb_t cb;
    /* 供内部使用 */
    tui_timer_t * arc_loading;
    /* 供内部使用 */
    tui_style_t arc_fg_style;

    int16_t start_angle;      /* 外部配置，弧形的开始角度 */
    int16_t end_angle;        /* 外部配置，弧形的结束角度 */
    int16_t bg_start_angle;   /* 外部配置，弧形背景的开始角度 */
    int16_t bg_end_angle;     /* 外部配置，弧形背景的结束角度 */
    int16_t line_width;       /* 外部配置，弧形的线宽 */
    uint32_t bg_color;        /* 外部配置，弧形的底色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    uint32_t fg_color;        /* 外部配置，弧形的前景色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    bool is_rounded;          /* 外部配置，是否弧形端圆滑处理 */
} tui_arc_attri_t;
```

弧形回调函数

```
typedef void (*tui_arc_cb_t)(tui_obj_t *obj, tui_event_e event);
```

弧形函数

```
/* 创建弧形，par 是其父节点 */
tui_obj_t * tui_arc_create(tui_obj_t * par);
/* 设置弧形的属性 */
int tui_arc_set_attri(tui_obj_t *arc, tui_arc_attri_t *attri);
/* 获得弧形的属性 */
int tui_arc_get_attri(tui_obj_t *arc, tui_arc_attri_t *attri);
/* 设置背景弧形的开始和结束的角度 */
void tui_arc_set_bg_angles(tui_obj_t * arc, uint16_t start, uint16_t end);
/* 设置前景弧形的开始和结束的角度 */
```

```

void tui_arc_set_angles(tui_obj_t * arc, uint16_t start, uint16_t end);
/* 设置弧形线的头是圆形状 */
void tui_arc_set_rounded(tui_obj_t * arc, bool is_rounded);
/* 弧形线的加载动画效果 */
void tui_arc_set_anim_loading(tui_obj_t * arc, uint32_t lap_need_time_ms, bool is_loading);
/* 以 pt0 为原点，和 radius 为半径旋转 angle 角度，得到对应圆边的一个点 */
tui_point_t tui_arc_get_circle_point(int angle, int radius, tui_point_t pt0)

```

### 6.7.3. 纯代码测试用例

```

/*      .0
 *
 *
 *.270      .90
 *
 *
 * .180
 */
static void arc_loader(tui_timer_t * t)
{
    tui_arc_attri_t attri;

    tui_arc_get_attri(tui_timer_get_user_data(t), &attri);
    attri.start_angle = (attri.start_angle + 1) % 360;
    attri.end_angle = (attri.end_angle + 1) % 360;

    tui_arc_set_angles(tui_timer_get_user_data(t), attri.start_angle, attri.end_angle);
}

static tui_obj_t * tui_arc(void)
{
    /*Create an Arc*/
    tui_arc_attri_t attri = { 0 };
    tui_obj_t * arc;
    /* 创建对象 */
    arc = tui_arc_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 100;
    attri.obj.pt.y = 100;
    attri.obj.size.width = 120;
    attri.obj.size.height = 120;
    attri.start_angle = 100; /* 弧形的开始角度 */
    attri.end_angle = 190; /* 弧形的结束角度 */
    attri.bg_start_angle = 0; /* 弧形背景的开始角度 */
    attri.bg_end_angle = 359; /* 弧形背景的结束角度 */
    attri.line_width = 4; /* 弧形的线宽 */
    attri.bg_color = 0xFFFAFAFA; /* 弧形的底色 (0xFF112233 FF是透明度; 11是R; 22是G; 33是B) */
}

```

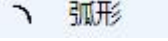
```

    attri.fg_color = 0xFFFF0000;          /* 弧形的前景色 (0xFF112233 FF是透明度; 11是R; 22是G;
33是B) */
    tui_arc_set_attr(arc, &attri);
    /* timer 动画 */
    tui_timer_create(arc_loader, 10, TUI_TIMER_PRIO_LOWEST, arc);

    return arc;
}

```

## 6.7.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  到视图中，设置对应的参数，如下：

控件类型 arc      控件名称 arc_7		▼ 控件参数	
		名称	值
		控件名称	arc_7
		类型	arc
		ID值	7
		位置	
		X	478
		Y	208
		宽高	
		宽度	60
		高度	60
		是否隐藏	false
		线宽	5
		弧线角度	
		起始角度	135
		结束角度	45
背景角度			
起始角度	0		
结束角度	359		
风格			
背景颜色	-1		
前景颜色	-26849		

## 6.8. bar\_progress 进度条

### 6.8.1. 描述

进度条是比较常用的控件，可以设置前景和背景颜色，也可以设置长宽等属性。当长大于宽时，是水平显示；反之是垂直显示，设计灵活方便。

进度条的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个进度条控件，设置属性时，必填属性会通过 UIStudio 工具配置。



## 6.8.2. 数据结构和函数

### 进度条属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 进度条回调函数，返回当事件 */
    tui_bar_progress_cb_t cb;
    /* 供内部使用 */
    tui_style_t fg_style;

    int16_t value;          /* 外部配置，进度条的值，范围是 0~100 */
    uint32_t bg_color;      /* 外部配置，进度条的底色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B) */
    uint32_t fg_color;      /* 外部配置，进度条的前景色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B) */
} tui_bar_progress_attr_t;
```

### 进度条回调函数

```
typedef void (*tui_bar_progress_cb_t)(tui_obj_t *obj, tui_event_e event);
```

### 进度条函数

```
/* 创建进度条，par 是其父节点 */
tui_obj_t * tui_bar_progress_create(tui_obj_t * par);
/* 设置进度条的属性 */
int tui_bar_progress_set_attr(tui_obj_t *bar_progress, tui_bar_progress_attr_t *attri);
/* 获得进度条的属性 */
int tui_bar_progress_get_attr(tui_obj_t *bar_progress, tui_bar_progress_attr_t *attri);
/* 获得进度条当前的值 */
int16_t tui_bar_progress_get_value(tui_obj_t *bar_progress);
/* 设置进度条当前的值 */
void tui_bar_progress_set_value(tui_obj_t *bar_progress, int16_t vaule);
```

## 6.8.3. 纯代码测试用例

```
static void bar_progress_loader(tui_timer_t * t)
{
    tui_bar_progress_attr_t attri;

    tui_bar_progress_get_attr(tui_timer_get_user_data(t), &attri);
    attri.value += 1;
    if (attri.value >= 100)
        attri.value = 0;
    tui_bar_progress_set_attr(tui_timer_get_user_data(t), &attri);
}

static tui_obj_t * tui_bar_progress(void)
{

```

```

/*Create an bar_progress*/
tui_bar_progress_attr_t attri = { 0 };
tui_obj_t *bar_progress_h, *bar_progress_v;
/* 创建对象 */
bar_progress_h = tui_bar_progress_create(tui_layer_normal());
/* 设置属性 */
attri.obj.pt.x = 100;
attri.obj.pt.y = 100;
attri.obj.size.width = 120;
attri.obj.size.height = 10;
attri.value = 50; /* 进度条的值，范围是 0~100 */
attri.bg_color = 0xFFAFAFAF; /* 进度条的底色 (0xFF112233 FF是透明度；11是R；22是
G；33是B) */
attri.fg_color = 0xFF0000FF; /* 进度条的前景色 (0xFF112233 FF是透明度；11是R；22
是G；33是B) */
tui_bar_progress_set_attri(bar_progress_h, &attri);
/* 创建对象 */
bar_progress_v = tui_bar_progress_create(tui_layer_normal());
/* 设置属性 */
attri.obj.pt.x = 100;
attri.obj.pt.y = 150;
attri.obj.size.width = 10;
attri.obj.size.height = 120;
attri.value = 50; /* 进度条的值，范围是 0~100 */
attri.bg_color = 0xFFAFAFAF; /* 进度条的底色 (0xFF112233 FF是透明度；11是R；22是
G；33是B) */
attri.fg_color = 0xFF0000FF; /* 进度条的前景色 (0xFF112233 FF是透明度；11是R；22
是G；33是B) */
tui_bar_progress_set_attri(bar_progress_v, &attri);
/* timer 动画 */
tui_timer_create(bar_progress_loader, 20, TUI_TIMER_PRIO_LOWEST, bar_progress_h);

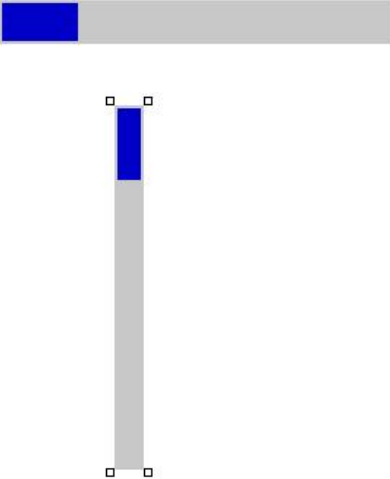
return bar_progress_h;
}

```

## 6.8.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  进度条 到视图中，设置对应的参数，如下：

控件类型 bar\_progress    控件名称 bar\_progress\_9



控件参数

名称	值
控件名称	bar_progress_9
类型	bar_progress
ID值	9
位置	
X	433
Y	182
宽高	
宽度	20
高度	249
是否隐藏	false
进度值	20
风格	
背景颜色	-3618616
前景颜色	-16777016

## 6.9. bar\_slider 滑条

### 6.9.1. 描述

滑条和进度条比较类似，可以理解成是进度条上面再加了一个控制点。同样可以设置前景和背景颜色，长宽等属性。当长大于宽时，是水平显示；反之是垂直显示，设计灵活方便。

滑条的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个滑条控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.9.2. 数据结构和函数

#### 滑条属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 滑动触发回调函数，返回当前值 */
    tui_bar_slider_cb_t cb;
    tui_obj_t *bg_img_obj;
    tui_obj_t *fg_img_obj;
    tui_obj_t *knob_img_obj;
    bool is_img;
    bool is_hor;
    /* 供内部使用 */
    tui_style_t knob_style;
    /* 供内部使用 */
}
```

```

    tui_style_t fg_style;

    int16_t value;           /* 外部配置，滑条的值，范围是 0~1000 */
    uint32_t bg_color;       /* 外部配置，滑条的底色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    uint32_t fg_color;       /* 外部配置，滑条的前景色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    uint32_t knob_color;     /* 外部配置，滑条的前景色（0xFF112233 FF是透明度；11是R；22是G；33是B） */

    char *bg_img_path;       /* 外部配置，滑条的底色图片 */
    char *fg_img_path;       /* 外部配置，滑条的前景色图片 */
    char *knob_img_path;     /* 外部配置，滑块的控钮图片 */
} tui_bar_slider_attri_t;

```

### 滑条回调函数

```
typedef void (*tui_bar_slider_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t value);
```

### 滑条函数

```

/* 创建滑条，par 是其父节点 */
tui_obj_t * tui_bar_slider_create(tui_obj_t * par);
/* 创建滑条，par 是其父节点，is_img 是否是图片模式 */
tui_obj_t * tui_bar_slider_create_ext(tui_obj_t * par, bool is_img);
/* 设置滑条的属性 */
int tui_bar_slider_set_attri(tui_obj_t *bar_slider, tui_bar_slider_attri_t *attri);
/* 获得滑条的属性 */
int tui_bar_slider_get_attri(tui_obj_t *bar_slider, tui_bar_slider_attri_t *attri);
/* 获得滑条当前的值 */
int16_t tui_bar_slider_get_value(tui_obj_t *bar_slider);
/* 设置滑条当前的值 */
void tui_bar_slider_set_value(tui_obj_t *bar_slider, int16_t vaule);

```

## 6.9.3. 纯代码测试用例

```

static void tui_bar_slider_cb(tui_obj_t *obj, tui_event_e event, int16_t value)
{
    printf("slider:%d\n", value);
}

static tui_obj_t * tui_bar_slider(void)
{
    /*Create an bar_slider*/
    tui_bar_slider_attri_t attri = { 0 };
    tui_obj_t *bar_slider_h, *bar_slider_v;
    /* 创建对象 */
    bar_slider_h = tui_bar_slider_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 250;
    attri.obj.pt.y = 150;
    attri.obj.size.width = 200;
    attri.obj.size.height = 10;
}

```

```

    attri.cb = tui_bar_slider_cb;
    attri.value = 30;                                /* 滑条的值，范围是 0~100 */
    attri.bg_color = 0x8FAFAFAF;                      /* 滑条的底色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；
33 是 B) */
    attri.fg_color = 0xFF0000FF;                      /* 滑条的前景色 (0xFF112233 FF 是透明度；11 是 R；22 是
G；33 是 B) */
    attri.knob_color = 0xFF00FFFF;                    /* 滑条的前景色 (0xFF112233 FF 是透明度；11 是 R；22 是
G；33 是 B) */
    tui_bar_slider_set_attri(bar_slider_h, &attri);
    /* 设置边框弧度 */
    tui_obj_set_border_radius(bar_slider_h, 5);
    /* 创建对象 */
    bar_slider_v = tui_bar_slider_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 250;
    attri.obj.pt.y = 200;
    attri.obj.size.width = 10;
    attri.obj.size.height = 200;
    attri.cb = tui_bar_slider_cb;
    attri.value = 50;                                /* 滑条的值，范围是 0~100 */
    attri.bg_color = 0xFFFAFAFAF;                      /* 滑条的底色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；
33 是 B) */
    attri.fg_color = 0xFF0000FF;                      /* 滑条的前景色 (0xFF112233 FF 是透明度；11 是 R；22 是
G；33 是 B) */
    attri.knob_color = 0xFF00FFFF;                    /* 滑条的前景色 (0xFF112233 FF 是透明度；11 是 R；22 是
G；33 是 B) */
    tui_bar_slider_set_attri(bar_slider_v, &attri);

    return bar_slider_h;
}

```

## 6.9.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  滑条 到视图中，设置对应的参数，如下：

控件类型 **bar\_slider**    控件名称 **bar\_slider\_10**



⌵ 控件参数

名称	值
控件名称	bar_slider_10
类型	bar_slider
ID值	10
位置	
X	348
Y	73
宽高	
宽度	244
高度	20
是否隐藏	false
滑块值	10
风格	
背景颜色	-3618616
前景颜色	-16777016
滑块颜色	-3827936

函数名称	启用
bar_slider_callback	<input type="checkbox"/>

6.10. button 按键

6.10.1. 描述

按键控件是比较常用的控件，UI 在设计的时候，给与了按键控件很单一的功能，如果需要在上面再加文字，或者其他功能，可以以按键控件作为父节点，创建标签控件等。

按键的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个按键控件，设置属性时，必填属性会通过 UIVisualizer 工具配置。

6.10.2. 数据结构和函数

按键属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 点击触发回调函数，返回当前状态 */
    tui_button_cb_t cb;
    /* 供内部使用 */
    bool pressed;

    uint32_t bg_color;      /* 外部配置，按键的背景颜色（0xFF112233  FF 是透明度；11 是 R；22 是 G；33 是 B） */
    uint32_t border_color; /* 外部配置，按键的边框颜色（0xFF112233  FF 是透明度；11 是 R；22 是 G；33 是 B） */
}
```

```

33 是 B) */
    uint32_t border_width; /* 外部配置, 按键的边框线宽度 */
} tui_button_attr_t;

```

### 按键回调函数

```
typedef void (*tui_button_cb_t)(tui_obj_t *obj, tui_event_e event);
```

### 按键函数

```

/* 创建按键, par 是其父节点 */
tui_obj_t * tui_button_create(tui_obj_t * par);
/* 设置按键的属性 */
int tui_button_set_attr(tui_obj_t *button, tui_button_attr_t *attri);
/* 获得按键的属性 */
int tui_button_get_attr(tui_obj_t *button, tui_button_attr_t *attri);
/* 使能按键底部是否有阴影效果 */
int tui_button_set_shadow(tui_obj_t *button, bool able);
/* 使能按键点击按下是否缩小 */
int tui_button_set_click_zoom(tui_obj_t *button, bool able);

```

## 6.10.3. 纯代码测试用例

```

static tui_obj_t * sound_tone;
static void tui_sound_tone(void)
{
    /* Create an sound */
    tui_sound_attr_t attri_sound = { 0 };
    /* 创建对象 */
    sound_tone = tui_sound_create(tui_layer_normal());
    /* 设置属性 */
    attri_sound.play_mode = 1;
    tui_sound_set_attr(sound_tone, &attri_sound);
    /* 设置音源 */
    tui_sound_set_sound_src(sound_tone, "V:\\sound\\tone.bin");/* 确保加载了 res.iso, 并且路径文件存在 */
}

static tui_obj_t * sound_didi;
static void tui_sound_didi(void)
{
    /* Create an sound */
    tui_sound_attr_t attri_sound = { 0 };
    /* 创建对象 */
    sound_didi = tui_sound_create(tui_layer_normal());
    /* 设置属性 */
    attri_sound.play_mode = 0;
    tui_sound_set_attr(sound_didi, &attri_sound);
    /* 设置音源 */
    tui_sound_set_sound_src(sound_didi, "V:\\sound\\didi.bin");/* 确保加载了 res.iso, 并且路径文件存在 */
}

static void tui_button_cb(tui_obj_t *obj, tui_event_e event)

```

```

{
    if (TUI_EVENT_RELEASED == event)
        tui_sound_play(sound_tone);
    else if (TUI_EVENT_PRESSED == event)
        tui_sound_play(sound_didi);

    printf("button:%d\n", event);
}

static tui_obj_t * tui_button(void)
{
    /*Create an button*/
    tui_button_attr_t attri = { 0 };
    tui_obj_t * button;
    /* 创建对象 */
    button = tui_button_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 500;
    attri.obj.pt.y = 150;
    attri.obj.size.width = 100;
    attri.obj.size.height = 40;
    attri.cb = tui_button_cb;
    attri.bg_color = 0xFF112233; /* 按键的背景颜色 (0xFF112233 FF是透明度; 11是R; 22是G; 33是B) */
    attri.border_color = 0xFF112233; /* 按键的边框颜色 (0xFF112233 FF是透明度; 11是R; 22是G; 33是B) */
    attri.border_width = 1; /* 按键的边框线宽度 */
    tui_button_set_attri(button, &attri);


    tui_label_attr_t attri_label = { 0 };
    tui_obj_t * label;
    /* 创建对象, 父节点是 button */
    label = tui_label_create(button);
    /* 设置属性 */
    attri_label.obj.pt.x = 0;
    attri_label.obj.pt.y = 10;
    attri_label.obj.size.width = 100;
    attri_label.obj.size.height = 40;
    attri_label.fnt_size = 15; /* 标签字体大小 */
    attri_label.txt = "hello"; /* 标签的文本信息 */
    attri_label.fnt_color = 0xFF112233; /* 标签字体的颜色 (0xFF112233 FF是透明度; 11是R; 22是G; 33是B) */
    attri_label.mode = 4; /* 标签显示模式 (其中有滚动显示) */
    attri_label.align = TUI_LABEL_ALIGN_CENTER; /* 标签对齐方式 */
    tui_label_set_attri(label, &attri_label);
    /* 创建声音 */
    tui_sound_tone();
    tui_sound_didi();

    return button;
}

```



## 6.10.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  按钮 到视图中，设置对应的参数，如下：

控件类型 **button**    控件名称 **button\_12**



▼ 控件参数

名称	值
控件名称	button_12
类型	button
ID值	12
位置	
X	419
Y	121
宽高	
宽度	185
高度	45
是否隐藏	false
边框宽度	1
风格	
边框颜色	-65536
背景颜色	-16777216

函数名称	启用	
button_callback	<input type="checkbox"/>	

## 6.11. image\_btn 图片按钮

### 6.11.1. 描述

图片按钮控件是比较常用的控件，它和按钮控件比较类似，但是它会缓存一张或者多张图片（目前上限是 16 张），作为按下和抬起显示，每张图片的源是 32 位 png 图片，其中拿一张作为底图，如果需要在上面再加文字，或者其他功能，可以以图片按钮控件作为父节点，创建标签控件等。

图片按钮的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个图片按钮控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.11.2. 数据结构和函数

#### 图片按钮属性

```
typedef struct {  
    /* 通用属性 */  
    tui_object_attr_t obj;
```

```

/* 点击触发回调函数，返回当前状态和值 */
tui_image_btn_cb_t cb;
/* 供内部使用 */
bool pressed;
/* 所有图片 buffer 保存，供内部使用 */
void *img_pressed[16];
/* 所有图片 buffer 保存，供内部使用 */
void *img_release[16];

int16_t img_index;          /* 外部配置，图片按键，当前图片的索引 */
int16_t img_num;            /* 外部配置，图片按键，有多少张图片总和，不要超过 16 张图片 */
bool hor_mirror;            /* 外部配置，图片按键的水平镜像 */
bool ver_mirror;            /* 外部配置，图片按键的垂直镜像 */
} tui_image_btn_attr_t;

```

### 图片按键回调函数

```
typedef void(*tui_image_btn_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t img_index);
```

### 图片按键函数

```

/* 创建图片按键，par 是其父节点 */
tui_obj_t * tui_image_btn_create(tui_obj_t * par);
/* 设置图片按键的属性 */
int tui_image_btn_set_attr(tui_obj_t *image_btn, tui_image_btn_attr_t *attri);
/* 获得图片按键的属性 */
int tui_image_btn_get_attr(tui_obj_t *image_btn, tui_image_btn_attr_t *attri);
/* 设置图片按键按下状态的对应索引的图片 */
int tui_image_btn_set_image_pressed(tui_obj_t *image_btn, int16_t img_index, char *path);
/* 设置图片按键默认状态的对应索引的图片 */
int tui_image_btn_set_image_release(tui_obj_t *image_btn, int16_t img_index, char *path);
/* 设置图片按键当前对应索引的图片 */
int tui_image_btn_set_cur_img_index(tui_obj_t *image_btn, int16_t img_index);
/* 设置图片按键水平镜像 */
void tui_image_btn_set_hor_mirror(tui_obj_t * image_btn, bool mirror_able);
/* 设置图片按键垂直镜像 */
void tui_image_btn_set_ver_mirror(tui_obj_t * image_btn, bool mirror_able);
/* 设置图片按键的放大缩小，zoom 大于 256 是放大，小于 256 是缩小，128 是缩小一半，512 放大一倍；zoom 不等于 256 将 disable 点击事件*/
void tui_image_btn_set_zoom(tui_obj_t * image_btn, uint16_t zoom);
/* 得到图片按键的缩放大小，zoom 大于 256 是放大，小于 256 是缩小，128 是缩小一半，512 放大一倍；zoom 不等于 256 将 disable 点击事件*/
void tui_image_btn_get_zoom(tui_obj_t * image_btn);
/* 设置图片按键缩放动画，大于 256 是放大，小于 256 是缩小，128 是缩小一半，512 放大一倍*/
void tui_image_btn_anim_zoom(tui_obj_t * image_btn, uint32_t need_time_ms, uint16_t start_zoom,
uint16_t end_zoom, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);

```

## 6.11.3. 纯代码测试用例

```

static void tui_image_btn_cb(tui_obj_t *obj, tui_event_e event, int16_t img_index)
{
    printf("image_btn:%d, %d\n", event, img_index);
}

```


```

static tui_obj_t * tui_image_btn(void)
{
    /*Create an image_btn*/
    tui_image_btn_attr_t attri = { 0 };
    tui_obj_t * image_btn;
    /* 创建对象 */
    image_btn = tui_image_btn_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 500;
    attri.obj.pt.y = 300;
    attri.obj.size.width = 0;
    attri.obj.size.height = 0;
    attri.cb = tui_image_btn_cb;
    attri.img_index = 0;          /* 图片按键，当前图片的索引 */
    attri.img_num = 3;           /* 图片按键，有多少张图片总和，不要超过 16 张图片 */
    tui_image_btn_set_image_release(image_btn, 0, "V:/image/0.bin");          /* 确保加载了
res.iso, 并且路径文件存在 */
    tui_image_btn_set_image_pressed(image_btn, 0, "V:/image/1.bin");          /* 确保加载了
res.iso, 并且路径文件存在 */
    tui_image_btn_set_image_release(image_btn, 1, "V:/image/2.bin");          /* 确保加载了
res.iso, 并且路径文件存在 */
    tui_image_btn_set_image_release(image_btn, 2, "V:/image/3.bin");          /* 确保加载了
res.iso, 并且路径文件存在 */
    tui_image_btn_set_attri(image_btn, &attri);


    return image_btn;
}

```

#### 6.11.4. 可视化工具编辑


拖拽<< 控件栏 列表中的  图片按键 到视图中，设置对应的参数，如下：

控件类型 image\_btn    控件名称 image\_btn\_13



图像集合输入

添加 删除 清空 上移 下移 确定

文件名称	文件大小
 V:/image/pages_icon2.bin	124x124

控件参数

名称	值
控件名称	image_btn_13
类型	image_btn
ID值	13
位置	
X	394
Y	158
宽高	
宽度	124
高度	124
是否隐藏	false
按键释放图片列表	["V:/image/pages_icon2.bin"]
按键按下图片列表	[]
图片索引	0
图片数量	1

函数名称	启用
image_btn_callback	<input type="checkbox"/>

6.12. image 图片

6.12.1. 描述

图片控件，用来贴图显示，该控件有一个特点，最大可以缓存 128 张图片，每张图片的源是 32 位 png 图片，根据需要显示其中一张，也可以设置动画时间，循环显示贴图，达到动画效果。也可以设置图片放大和缩小，以及图片的旋转，旋转角度标记如下：

.0  
.270    .90  
.180

图片的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个图片控件，设置属性时，必填属性会通过 UIStudio 工具配置。

6.12.2. 数据结构和函数

图片属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attri_t obj;
    /* 动画变化后有回调函数，返回当前图片值 */
    tui_image_cb_t cb;
    /* 所有图片 buffer 保存，供内部使用 */
}
```

```

void *img[128];
void *img_temp;
void *img_temp_1;
void *img_type[128];
/* 动画 timer, 供内部使用 */
tui_timer_t *timer;
/* 动画 timer, 开始索引值供, 内部使用 */
int16_t start_index;
/* 动画 timer, 结束索引值供, 内部使用 */
int16_t end_index;
/* 动画 timer, 旋转角度, 内部使用 */
int16_t rotate_offset;
/* 动画 timer, 旋转方向, 内部使用 */
double rotate_way;
/* 内部使用 */
int16_t rotate_angle_temp;

int16_t img_index;      /* 外部配置, 图片控件, 当前图片的索引 */
int16_t img_num;        /* 外部配置, 图片控件, 有多少张图片总和, 不要超过 128 张图片*/
int16_t zoom;           /* 外部配置, 图片控件放大缩小值,
                        256 不放大,
                        小于 256 缩小,
                        大于 256 放大,
                        512 放大两倍,
                        128 缩小一半 */
int16_t rotate_angle;   /* 外部配置, 图片控件旋转角度, 范围 0~360 */
tui_point_t rotate_pt;  /* 外部配置, 图片控件旋转的中心坐标, 默认是中心点 */
bool anim_start;        /* 外部配置, 图片控件图片动画默认是否开始, 0 或者 1*/
int32_t anim_time;      /* 外部配置, 图片控件图片动画切换的时间, 毫秒单位*/
tui_image_circulate_e mode; /* 外部配置, 图片控件图片动画切换的模式, 单次还是无限循环 */
tui_coord_t x_offset;   /* 外部配置, 图片的源 X 偏移 */
tui_coord_t y_offset;   /* 外部配置, 图片的源 Y 偏移 */
bool hor_mirror;        /* 外部配置, 图片的水平镜像 */
bool ver_mirror;        /* 外部配置, 图片的垂直镜像 */
} tui_image_attri_t;

```

## 图片回调函数

```
typedef void(*tui_image_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t img_index);
```

## 图片函数

```

/* 创建图片, par 是其父节点 */
tui_obj_t * tui_image_create(tui_obj_t * par);
/* 设置图片的属性 */
int tui_image_set_attri(tui_obj_t *image, tui_image_attri_t *attri);
/* 获得图片的属性 */
int tui_image_get_attri(tui_obj_t *image, tui_image_attri_t *attri);
/* 设置图片的对应索引的源图片 来自于 .png、.jpg、.bmp 图片*/
int tui_image_set_image_src(tui_obj_t *image, int16_t img_index, char *path);
/* 设置图片的对应索引的源图片, 来自于 ARGB buffer (.jpg 图片解码的颜色格式) */
int tui_image_set_image_argb_buf(tui_obj_t *image, int16_t img_index, const void *buf, int32_t img_w,
int32_t img_h);
/* 设置图片的对应索引的源图片, 来自于 RGBA buffer (.png 图片解码的颜色格式) */

```

```

int tui_image_set_image_rgba_buf(tui_obj_t *image, int16_t img_index, const void *buf, int32_t img_w,
                                int32_t img_h);
/* 设置图片的当前图片对应的索引*/
int tui_image_set_cur_img_index(tui_obj_t *image, int16_t img_index);
/* 设置图片源的 X 坐标偏移*/
void tui_image_set_offset_x(tui_obj_t * image, tui_coord_t x);
/* 设置图片源的 Y 坐标偏移*/
void tui_image_set_offset_y(tui_obj_t * image, tui_coord_t y);
/* 设置图片的播放图片动画，设置每张图片切换的时间间隔和循环的次数*/
int tui_image_start_anim(tui_obj_t *image, uint32_t anim_time, tui_image_circulate_e mode);
/* 停止图片的播放图片动画 */
int tui_image_stop_anim(tui_obj_t *image);
/* 获得图片的播放动画状态，返回 0 是停止，1 是运行 */
int tui_image_get_anim_state(tui_obj_t *image);
/* 设置图片的放大缩小，zoom 大于 256 是放大，小于 256 是缩小，128 是缩小一半，512 放大一倍*/
void tui_image_set_zoom(tui_obj_t * image, uint16_t zoom);
/* 设置图片的宽高，可以任意缩放 */
void tui_image_set_size(tui_obj_t * image, uint32_t new_width, uint32_t new_height);
/* 设置图片的旋转角度*/
void tui_image_set_angle(tui_obj_t * image, int16_t angle, bool is_anima);
/* 图片水平镜像 */
void tui_image_set_hor_mirror(tui_obj_t * image, bool mirror_able);
/* 图片垂直镜像 */
void tui_image_set_ver_mirror(tui_obj_t * image, bool mirror_able);
/* 设置图片的旋转中心点*/
void tui_image_set_rotation_center_coor(tui_obj_t * image, tui_coord_t pivot_x, tui_coord_t pivot_y);
/* 设置图片缩放动画，大于 256 是放大，小于 256 是缩小，128 是缩小一半，512 放大一倍*/
void tui_image_anim_zoom(tui_obj_t * image, uint32_t need_time_ms, uint16_t start_zoom, uint16_t
end_zoom, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置图片旋转动画，0~360° */
void tui_image_anim_angle(tui_obj_t * image, uint32_t need_time_ms, uint16_t start_angle, uint16_t
end_angle, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置图片变化宽度的动画，宽度不要超过原图的 2 倍 */
void tui_image_anim_width(tui_obj_t * image, uint32_t need_time_ms, uint16_t start_width, uint16_t
end_width, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);
/* 设置图片变化高度的动画，高度不要超过原图的 2 倍 */
void tui_image_anim_height(tui_obj_t * image, uint32_t need_time_ms, uint16_t start_height, uint16_t
end_height, tui_anim_path_e path_type, tui_object_anim_cb_t end_cb);

```

### 6.12.3. 纯代码测试用例

```

static void tui_image_cb(tui_obj_t *obj, tui_event_e event, int16_t img_index)
{
    tui_image_attr_t attri;

    if (event == TUI_EVENT_RELEASED) {
        tui_image_get_attri(obj, &attri);
        attri.img_index = (img_index + 1) % 7;
        attri.rotate_angle += 10; /* 0~3600 */
    }
}

```

```

    /**
     * 256 or LV_ZOOM_IMG_NONE for no zoom
     * <256: scale down
     * >256 scale up
     * 128 half size
     * 512 double size
     */
    //attri.zoom += 5;
    tui_image_set_attri(obj, &attri);
}

printf("image:%d, %d\n", event, img_index);
}

static uint32_t g_argb[120 * 120];
static tui_obj_t * tui_image(void)
{
    /*Create an image*/
    tui_image_attri_t attri = { 0 };
    tui_obj_t * image;
    /* 创建对象 */
    image = tui_image_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 650;
    attri.obj.pt.y = 300;
    attri.obj.size.width = 0;
    attri.obj.size.height = 0;
    attri.cb = tui_image_cb;
    attri.img_index = 0;          /* 图片控件，当前图片的索引 */
    attri.img_num = 8;           /* 图片控件，有多少张图片总和，不要超过 128 张图片*/
    attri.zoom = 512;            /* 图片控件放大缩小值，256 不放大，小于 256 缩小，大于 256 放大，512
    放大两倍，128 缩小一半*/
    attri.rotate_angle = 45;     /* 图片控件旋转角度，范围 0~360 */
    tui_image_set_attri(image, &attri);
    tui_image_set_image_src(image, 0, "V:/image/0.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    tui_image_set_image_src(image, 1, "V:/image/1.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    tui_image_set_image_src(image, 2, "V:/image/2.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    tui_image_set_image_src(image, 3, "V:/image/3.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    tui_image_set_image_src(image, 4, "V:/image/4.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    tui_image_set_image_src(image, 5, "V:/image/5.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    tui_image_set_image_src(image, 6, "V:/image/6.bin");          /* 确保加载了 res.iso，并且路径文件
    存在 */
    memset(g_argb, 0xAF, 120*120*4);
    tui_image_set_image_argb_buf(image, 7, g_argb, 120, 120);
    tui_image_set_cur_img_index(image, attri.img_index);
}

```




```


/* 开始动画 */
tui_image_start_anim(image, 500, 1);

return image;
}

```

## 6.12.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  图片 到视图中，设置对应的参数，如下：

控件类型 image      控件名称 image_14		⌵ 控件参数	
		名称	值
		控件名称	image_14
		类型	image
		ID值	14
		位置	
		X	444
		Y	180
		宽高	
		宽度	152
		高度	96
		是否隐藏	false
		图片列表	["V:/image/p2.bin"]
		图片索引	0
		图片数量	1
		缩放模式	256
		旋转角度	0
		旋转坐标	
		X	0
		Y	0
动画			
动画是否开始	0		
动画切换时间	3000		
动画切换模式	0		
函数名称	启用		
image_callback	<input type="checkbox"/>		

## 6.13. animation 动画

### 6.13.1. 描述

动画控件通过 UIstudio 工具，可以灵活的制作自定义动画效果，可以控制到每一帧的时间和坐标位置，相比传统的 GIF 动画，该控件更加灵活，图像文件自定义，大小可以不一致，及其方便灵活。



## 6.13.2. 数据结构和函数

### 动画属性

```
typedef struct {
    char path[48]; /* 路径不要超过 48 个字符 */
    int x;
    int y;
    int ms;
} tui_anima_img_attri_t;

typedef struct {
    /* 通用属性 */
    tui_object_attri_t obj;
    /* 动画变化后有回调函数，返回当前图片值 */
    tui_animation_cb_t cb;
    /* 动画 image，供内部使用 */
    void *img_obj;
    /* buffer 保存，供内部使用 */
    void *img_buff;
    /* 动画 timer，供内部使用 */
    tui_timer_t *timer;
    /* 动画 timer，开始索引值供，内部使用 */
    int16_t start_index;
    /* 动画 timer，结束索引值供，内部使用 */
    int16_t end_index;

    uint32_t bg_color; /* 外部配置，动画容器的背景颜色，默认是 0 (0xFF112233 FF 是透明度；
11 是 R；22 是 G；33 是 B) */
    int16_t img_index; /* 外部配置，动画，当前图片的索引 */
    int16_t img_num; /* 外部配置，动画，有多少张图片总和，不要超过 128 张图片 */
    tui_anima_img_attri_t anima_img[128]; /* 外部配置，动画里面的图片集合，图片不要超过 128 张 */
    bool anim_start; /* 外部配置，动画默认是否开始，0 或者 1 */
    tui_image_circulate_e mode; /* 外部配置，动画切换的模式，单次还是无限循环 */
    bool hor_mirror; /* 外部配置，动画的水平镜像 */
    bool ver_mirror; /* 外部配置，动画的垂直镜像 */
} tui_animation_attri_t;
```

### 动画回调函数

```
typedef void(*tui_animation_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t img_index);
```

### 动画函数

```
/* 创建动画，par 是其父节点 */
tui_obj_t * tui_animation_create(tui_obj_t * par);
/* 设置动画的属性 */
int tui_animation_set_attri(tui_obj_t *animation, tui_animation_attri_t *attri);
/* 获得动画的属性 */
int tui_animation_get_attri(tui_obj_t *animation, tui_animation_attri_t *attri);
/* 设置动画的当前图片对应的索引 */
int tui_animation_set_cur_img_index(tui_obj_t *animation, int16_t img_index);
/* 动画开始 */
int tui_animation_start_anim(tui_obj_t *animation, tui_image_circulate_e mode);
/* 动画停止 */
```

```


int tui_animation_stop_anim(tui_obj_t *animation);
/* 动画水平镜像 */
void tui_animation_set_hor_mirror(tui_obj_t * animation, bool mirror_able);
/* 动画垂直镜像 */
void tui_animation_set_ver_mirror(tui_obj_t * animation, bool mirror_able);

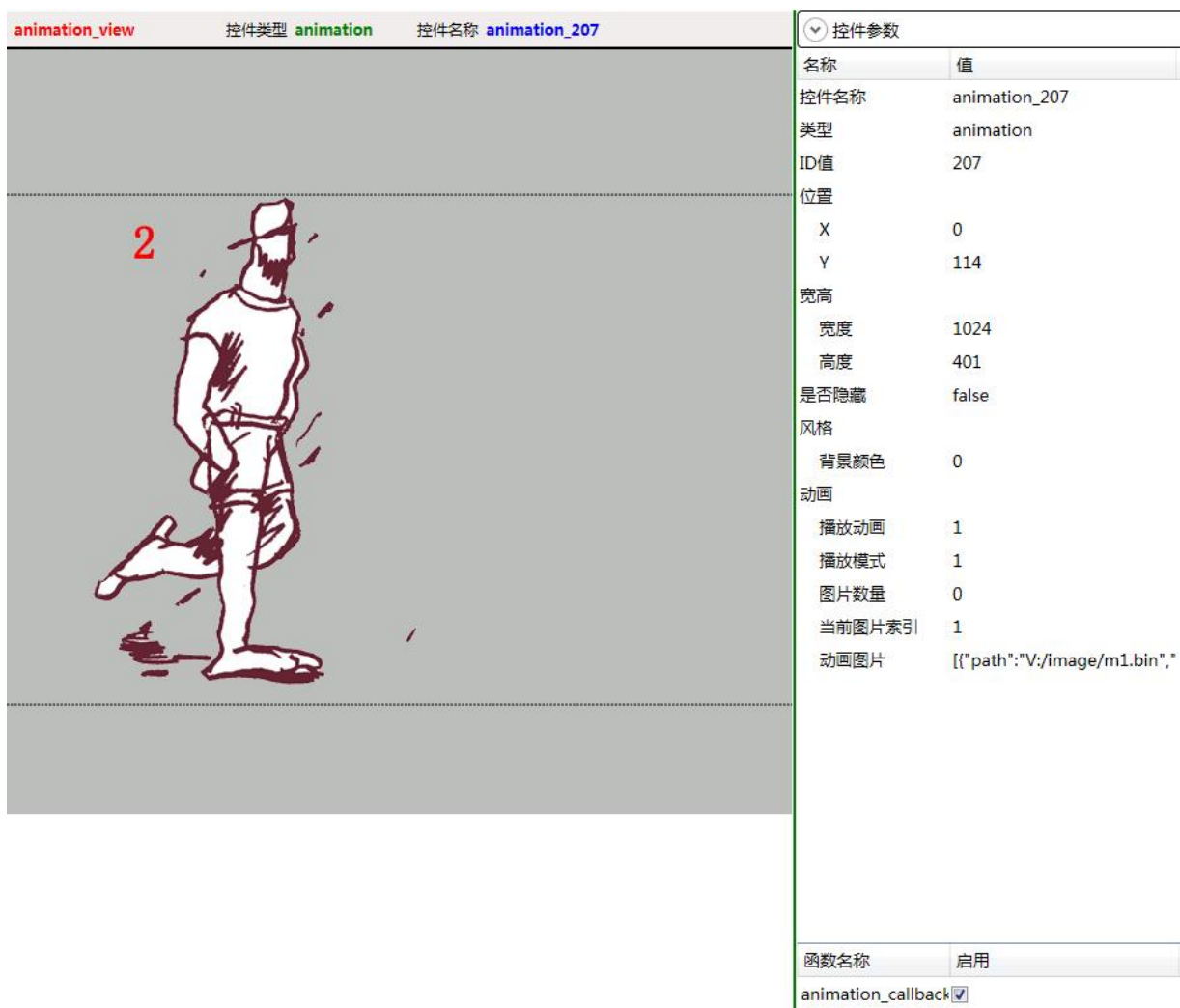
```

### 6.13.3. 纯代码测试用例

无

### 6.13.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  动画控件 到视图中，设置对应的参数，如下：



名称	值
控件名称	animation_207
类型	animation
ID值	207
位置	
X	0
Y	114
宽高	
宽度	1024
高度	401
是否隐藏	false
风格	
背景颜色	0
动画	
播放动画	1
播放模式	1
图片数量	0
当前图片索引	1
动画图片	[{"path": "V:/image/m1.bin", "

函数名称	启用
animation_callback	<input checked="" type="checkbox"/>

编辑动画参数，如图：



6.14. Gif 图片动画

6.14.1. 描述

Gif 文件控件，可以播放 gif 动画，控制播放、暂停和重新播放,支持绿色透明 (0,255,0)。

Gif 的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个 gif 控件，设置属性时，必填属性会通过 Uistudio 工具配置。

6.14.2. 数据结构和函数

```
gif 属性
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 动画变化后有回调函数，返回当前状态 */
    tui_gif_cb_t cb;

    char gif_path[128];          /* 外部配置，gif 动画路径 */
    tui_image_circulate_e mode; /* 外部配置，动画切换的模式，单次还是无限循环 */
} tui_gif_attr_t;

gif 回调函数
typedef void(*tui_gif_cb_t)(tui_obj_t *obj, tui_event_e event);

gif 函数
/* 创建 gif，par 是其父节点 */
tui_obj_t * tui_gif_create(tui_obj_t * par);
/* 设置 gif 的属性 */
int tui_gif_set_attr(tui_obj_t *gif, tui_gif_attr_t *attr);
/* 获得 gif 的属性 */
int tui_gif_get_attr(tui_obj_t *gif, tui_gif_attr_t *attr);
/* gif 动画暂停 */
int tui_gif_pause(tui_obj_t *gif);
/* gif 动画播放 */
int tui_gif_play(tui_obj_t *gif);
```

```

/* gif 动画从头开始播放 */
int tui_gif_restart(tui_obj_t *gif);
/* 设置 gif 动画循环模式 */
int tui_gif_set_mode(tui_obj_t *gif, tui_image_circulate_e mode);

```

### 6.14.3. 纯代码测试用例

```

static tui_obj_t * g_gif;
static tui_obj_t * tui_gif(void)
{
    /*Create an gif*/
    tui_gif_attr_t attri = { 0 };
    /* 创建对象 */
    g_gif= tui_gif_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 50;
    attri.obj.pt.y = 50;
    attri.obj.size.width = 0;
    attri.obj.size.height = 0;
    strcpy(attri.gif_path, "V:\\image\\bootlogo.bin");
    attri.mode = TUI_IMAGE_LOOP;

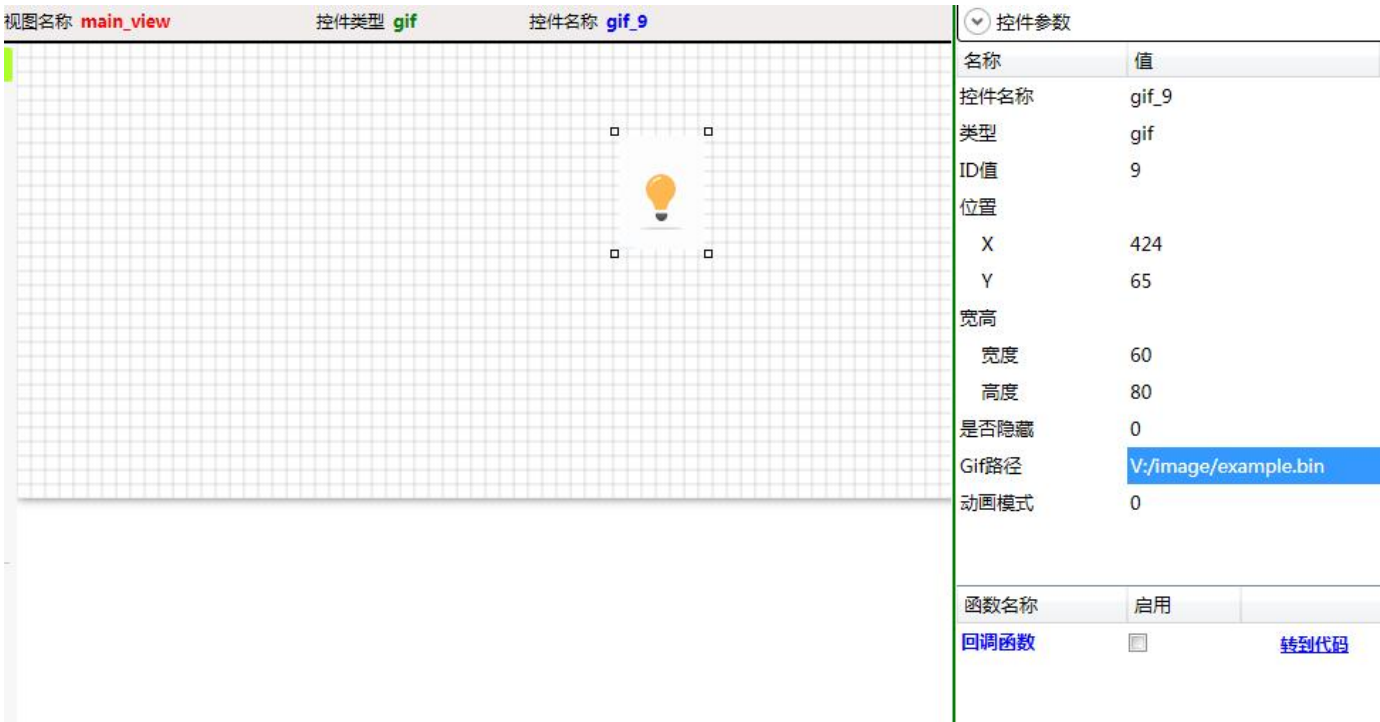
    tui_gif_set_attri(g_gif, &attri);

    return g_gif;
}

```

### 6.14.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  Gif控件 到视图中，设置对应的参数，如下：



## 6.15. line 线

### 6.15.1. 描述

画线控件，可以设置线的颜色和坐标位置，还能改变线的粗细宽度，该控件支持多点折线，同时也支持折线变化成平滑的贝塞尔曲线。

画线的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个画线控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.15.2. 数据结构和函数

#### 画线属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 线回调函数，返回当事件 */
    tui_line_cb_t cb;
    /* 多点线，供内部使用 */
    uint16_t point_num;
    /* 多点线，供内部使用 */
    uint16_t point_index;

    uint32_t width;          /* 外部配置，线的宽度 */
    uint32_t color;          /* 外部配置，线的颜色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    tui_point_t pts[2];      /* 外部配置，线的坐标，两点一线 */
} tui_line_attr_t;
```

#### 画线回调函数

```
typedef void (*tui_line_cb_t)(tui_obj_t *obj, tui_event_e event);
```

## 画线函数

```

/* 创建线，par 是其父节点 */
tui_obj_t * tui_line_create(tui_obj_t * par);
/* 设置线的属性 */
int tui_line_set_attri(tui_obj_t *line, tui_line_attri_t *attri);
/* 获得线的属性 */
int tui_line_get_attri(tui_obj_t *line, tui_line_attri_t *attri);
/* 设置线的两点坐标 */
void tui_line_set_points(tui_obj_t * line, tui_point_t point_a, tui_point_t point_b);
/* 设置线的线宽 */
void tui_line_set_line_width(tui_obj_t * line, uint32_t width);
/* 设置线的颜色 */
void tui_line_set_line_color(tui_obj_t * line, uint32_t color);
/* 设置多点折线，支持贝塞尔曲线平滑 */
void tui_line_set_some_points_line(tui_obj_t * line, tui_point_t *point_a, uint16_t point_num, bool
is_bezier);

```

## 6.15.3. 纯代码测试用例

```

static tui_obj_t * g_line;
static tui_obj_t * tui_line(void)
{
    /*Create an line*/
    tui_line_attri_t attri = { 0 };
    /* 创建对象 */
    g_line = tui_line_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 0;
    attri.obj.pt.y = 0;
    attri.obj.size.width = 0;
    attri.obj.size.height = 0;
    attri.width = 3; /* 线的宽度 */
    attri.color = 0xFFFF0000; /* 线的颜色 (0xFF112233 FF是透明度; 11是R; 22是G; 33是
B) */
    attri.pts[0].x = 20; /* 线的坐标，两点一线 */
    attri.pts[0].y = 550; /* 线的坐标，两点一线 */
    attri.pts[1].x = 200; /* 线的坐标，两点一线 */
    attri.pts[1].y = 580; /* 线的坐标，两点一线 */
    tui_line_set_attri(g_line, &attri);

    return g_line;
}

static void tui_switch_btn_cb(tui_obj_t *obj, tui_event_e event, int16_t value)
{
    if (value == 0) {
        tui_line_set_line_width(g_line, 1);
        tui_line_set_line_color(g_line, 0xFFFF0000);
    }
    else{


```


```

    tui_line_set_line_width(g_line, 5);
    tui_line_set_line_color(g_line, 0xFF00FF00);
}
printf("switch_btn:%d\n", value);
}

```

## 6.15.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  线 到视图中，设置对应的参数，如下：

控件类型 <span>line</span> 控件名称 <span>line_3</span>		▼ 控件参数	
		名称	值
		控件名称	line_3
		类型	line
		ID值	3
		是否隐藏	0
		线宽度	5
		颜色	-16731011
		坐标值	
		X1	400
		X2	770
Y1	157		
Y2	240		

## 6.16. switch\_btn 切换开关

### 6.16.1. 描述

切换开关是智能系统设置中比较常见的控件，通过这个控件可以实现‘是’和‘否’的设置选项，该控件比较灵活，可以设置 3 个元素的颜色，和控件大小等。

切换开关的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个切换开关控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.16.2. 数据结构和函数

#### 切换开关属性

```

typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 点击触发回调函数，返回当前状态 */
    tui_switch_btn_cb_t cb;
    /* 供内部使用 */
}

```

```

    tui_style_t knob_style;
    /* 供内部使用 */
    tui_style_t fg_style;

    bool value;           /* 外部配置，切换开关的值，0 或者 1 */
    uint32_t bg_color;     /* 外部配置，切换开关的底色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；
33 是 B) */
    uint32_t fg_color;     /* 外部配置，切换开关的前景色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；
33 是 B) */
    uint32_t knob_color;   /* 外部配置，切换开关的按钮色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；
33 是 B) */
} tui_switch_btn_attr_t;

```

#### 切换开关回调函数

```
typedef void (*tui_switch_btn_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t value);
```

#### 切换开关函数

```

/* 创建切换开关，par 是其父节点 */
tui_obj_t * tui_switch_btn_create(tui_obj_t * par);
/* 设置切换开关的属性 */
int tui_switch_btn_set_attr(tui_obj_t *switch_btn, tui_switch_btn_attr_t *attr);
/* 获得切换开关的属性 */
int tui_switch_btn_get_attr(tui_obj_t *switch_btn, tui_switch_btn_attr_t *attr);
/* 获得切换开关的值 0 或者 1 */
int tui_switch_btn_get_vaule(tui_obj_t *switch_btn);
/* 设置切换开关的值 0 或者 1 */
void tui_switch_btn_set_vaule(tui_obj_t *switch_btn, bool value);

```

### 6.16.3. 纯代码测试用例

```

static tui_obj_t * g_line;
static tui_obj_t * tui_line(void)
{
    /*Create an line*/
    tui_line_attr_t attr = { 0 };
    /* 创建对象 */
    g_line = tui_line_create(tui_layer_normal());
    /* 设置属性 */
    attr.obj.pt.x = 0;
    attr.obj.pt.y = 0;
    attr.obj.size.width = 0;
    attr.obj.size.height = 0;
    attr.width = 3;           /* 线的宽度 */
    attr.color = 0xFFFF0000;   /* 线的颜色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是
B) */
    attr.pts[0].x = 20;       /* 线的坐标，两点一线 */
    attr.pts[0].y = 550;      /* 线的坐标，两点一线 */
    attr.pts[1].x = 200;      /* 线的坐标，两点一线 */
    attr.pts[1].y = 580;      /* 线的坐标，两点一线 */
    tui_line_set_attr(g_line, &attr);
}

```




```

return g_line;
}

static void tui_switch_btn_cb(tui_obj_t *obj, tui_event_e event, int16_t value)
{
    if (value == 0) {
        tui_line_set_line_width(g_line, 1);
        tui_line_set_line_color(g_line, 0xFFFF0000);
    }
    else {
        tui_line_set_line_width(g_line, 5);
        tui_line_set_line_color(g_line, 0xFF00FF00);
    }
    printf("switch_btn:%d\n", value);
}

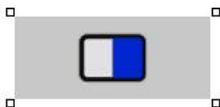
```

## 6.16.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  切换开关 到视图中，设置对应的参数，如下：

控件类型 switch\_btn

控件名称 switch\_btn\_24



▼ 控件参数

名称	值
控件名称	switch_btn_24
类型	switch_btn
ID值	24
位置	
X	390
Y	85
宽高	
宽度	133
高度	56
是否隐藏	false
默认状态	1
风格	
背景颜色	-3618616
前景颜色	-16777016
开关颜色	-3827937

函数名称	启用
switch_btn_callbac	<input type="checkbox"/>

## 6.17. dropdown 下拉菜单

### 6.17.1. 描述

下拉菜单可以节省视图空间，该控件可以动态增加菜单选项（最大支持 64 项），和控件颜色。

下拉菜单的属性继承了 `object`，同时也派生出新的自己的特有属性。在创建一个下拉菜单控件，设置属性时，必填属性会通过 `UIStudio` 工具配置。

### 6.17.2. 数据结构和函数

#### 下拉菜单属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 点击触发回调函数，返回当前索引值 */
    tui_dropdown_cb_t cb;
    /* 供内部使用 */
    tui_style_t bar_style;
    /* 供内部使用 */
    tui_style_t list_style;

    int16_t cur_index; /* 外部配置，当前下拉菜单的索引值 */
    int16_t options_num; /* 外部配置，当前下拉菜单的索引值 */
    char options[210][64]; /* 外部配置，下拉菜单的文本设置 */
    uint32_t bar_color; /* 外部配置，下拉菜单 bar 的底色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B) */
    uint32_t list_color; /* 外部配置，下拉菜单列表的底色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B) */
    uint32_t text_color; /* 外部配置，下拉菜单字符的颜色 (0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B) */
    int16_t text_font_size; /* 外部配置，下拉菜单字符字体大小 */
} tui_dropdown_attr_t;
```

#### 下拉菜单回调函数

```
typedef void (*tui_dropdown_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t index);
```

#### 下拉菜单函数

```
/* 创建下拉菜单，par 是其父节点 */
tui_obj_t * tui_dropdown_create(tui_obj_t * par);
/* 设置下拉菜单的属性 */
int tui_dropdown_set_attr(tui_obj_t *dropdown, tui_dropdown_attr_t *attri);
/* 获得下拉菜单的属性 */
int tui_dropdown_get_attr(tui_obj_t *dropdown, tui_dropdown_attr_t *attri);
/* 增加或者改变下拉菜单的某一项 */
void tui_dropdown_add_option(tui_obj_t * dropdown, const char * option, int16_t index);
/* 设置所有下拉菜单项 */
void tui_dropdown_set_selected_str(tui_obj_t *dropdown, const char * options[], int16_t options_num);
/* 获得下所有拉菜单项 */
char ** tui_dropdown_get_selected_str(tui_obj_t *dropdown, int16_t *options_num);
/* 获得选中下拉菜单焦点项 */
```

```

uint16_t tui_dropdown_get_selected_index(tui_obj_t *dropdown);
/* 设置下拉菜单焦点项，通过索引值 */
void tui_dropdown_set_selected_index(tui_obj_t *dropdown, int16_t index);
/* 设置下拉菜单下拉图标 */
void tui_dropdown_set_symbol(tui_obj_t *dropdown, bool able);
/* 设置下拉菜单下拉的最大高度 */
void tui_dropdown_set_max_height(tui_obj_t *dropdown, tui_coord_t h);

```

### 6.17.3. 纯代码测试用例

```

static void tui_dropdown_cb(tui_obj_t *obj, tui_event_e event, int16_t index)
{
    printf("dropdown:%d\n", index);
}

static tui_obj_t * tui_dropdown(void)
{
    /*Create an dropdown*/
    tui_dropdown_attr_t attri = { 0 };
    tui_obj_t * dropdown;
    /* 创建对象 */
    dropdown = tui_dropdown_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 300;
    attri.obj.pt.y = 200;
    attri.obj.size.width = 120;
    attri.obj.size.height = 30;
    attri.cb = tui_dropdown_cb;
    attri.cur_index = 0;          /* 当前下拉菜单的索引值 */
    attri.options_num = 12;      /* 当前下拉菜单的索引值 */
    strcpy(attri.options[0], "0"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[1], "30"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[2], "60"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[3], "90"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[4], "120"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[5], "150"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[6], "180"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[7], "210"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[8], "240"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[9], "270"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[10], "300"); /* 下拉菜单的文本设置 */
    strcpy(attri.options[11], "330"); /* 下拉菜单的文本设置 */
    attri.bar_color = 0xFFFAFAFA; /* 下拉菜单 bar 的底色 (0xFF112233 FF 是透明度; 11 是 R; 22 是 G; 33 是 B) */
    attri.list_color = 0xFFFAFAFA; /* 下拉菜单列表的底色 (0xFF112233 FF 是透明度; 11 是 R; 22 是 G; 33 是 B) */
    attri.text_color = 0xFF00FF00; /* 下拉菜单字符的颜色 (0xFF112233 FF 是透明度; 11 是 R; 22

```

```
是 G; 33 是 B) */
    attri.text_font_size = 15;      /* 下拉菜单字符字体大小 */
    tui_dropdown_set_attri(dropdown, &attri);

    return dropdown;
}
```

## 6.17.4. 可视化工具编辑

拖拽<< 控件栏 列表中的 下拉列表 到视图中，设置对应的参数，如下：

控件类型 dropdown

控件名称 dropdown\_16

▼ 控件参数

名称	值
控件名称	dropdown_16
类型	dropdown
ID值	16
位置	
X	411
Y	182
宽高	
宽度	154
高度	35
是否隐藏	false
下拉项	
列表项	["Option 1","Option 2","Option 3"]
选中项索引	0
字体	
名称	1
大小	15
风格	
背景颜色	-1
列表背景颜色	-3618616
列表文本颜色	-16777216

函数名称	启用
dropdown_callback	<input type="checkbox"/>

## 6.18. textbox 文本输入框

### 6.18.1. 描述

文本输入框可以输入任意字符，也可以输入指定字符，可以支持密文输入和支持正常显示输入，当选中文本输入框后，系统会自动弹出键盘界面，供用户输入。失去焦点后，键盘就自动隐藏。

文本输入框的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个文本输入框控件，设置属性时，必填属性会通过 UIStudio 工具配置。

## 6.18.2. 数据结构和函数

### 文本输入框属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 输入触发回调函数，返回当前字符串 */
    tui_textbox_cb_t cb;

    bool pwd_able;          /* 外部配置，文本输入框模式，是否是密码输入 */
    char *accepted_chars;    /* 外部配置，文本输入框可以输入的字符限制， 空是支持所有字符 */
} tui_textbox_attr_t;
```

### 文本输入框回调函数

```
typedef void (*tui_textbox_cb_t)(tui_obj_t *obj, tui_event_e event, const char * str);
```

### 文本输入框函数

```
/* 创建文本输入框，par 是其父节点 */
tui_obj_t * tui_textbox_create(tui_obj_t * par);
/* 设置文本输入框的属性 */
int tui_textbox_set_attr(tui_obj_t *textbox, tui_textbox_attr_t *attri);
/* 获得文本输入框的属性 */
int tui_textbox_get_attr(tui_obj_t *textbox, tui_textbox_attr_t *attri);
/* 获得文本输入框里面内容 */
const char * tui_textbox_get_text(const tui_obj_t * textbox);
/* 获得文本输入框里面内容 */
void tui_textbox_set_text(tui_obj_t * textbox, const char * txt);
/* 设置文本输入框的属性是否是密文输入 */
void tui_textbox_set_pwd_mode(tui_obj_t * textbox, bool pwd_able);
/* 设置文本输入框的可以接收的字符范围 */
void tui_textarea_set_accepted_chars(tui_obj_t * textbox, const char * accepted_chars);
```

## 6.18.3. 纯代码测试用例

```
static void tui_textbox_cb(tui_obj_t *obj, tui_event_e event, const char * str)
{
    printf("textbox:%s\n", str);
    if (strcmp(str, "hello") == 0)
        tui_textbox_set_pwd_mode(obj, 0);
    if (strcmp(str, "helloooo") == 0)
        tui_textbox_set_pwd_mode(obj, 1);
}

static tui_obj_t * tui_textbox(void)
{
    /*Create an textbox*/
    tui_textbox_attr_t attri = { 0 };
}
```


```

tui_obj_t * textbox;
/* 创建对象 */
textbox = tui_textbox_create(tui_layer_normal());
/* 设置属性 */
attri.obj.pt.x = 300;
attri.obj.pt.y = 400;
attri.obj.size.width = 120;
attri.obj.size.height = 30;
attri.cb = tui_textbox_cb;
attri.pwd_able = 0;          /* 文本输入框模式，是否是密码输入 */
attri.accepted_chars = NULL; /* 文本输入框可以输入的字符限制， 空是支持所有字符 */
tui_textbox_set_attri(textbox, &attri);

return textbox;
}


```

## 6.18.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  文本输入框 到视图中，设置对应的参数，如下：

控件类型	控件名称	控件参数																										
textbox	textbox_17	<table border="1"> <thead> <tr> <th>名称</th> <th>值</th> </tr> </thead> <tbody> <tr> <td>控件名称</td> <td>textbox_17</td> </tr> <tr> <td>类型</td> <td>textbox</td> </tr> <tr> <td>ID值</td> <td>17</td> </tr> <tr> <td>位置</td> <td></td> </tr> <tr> <td>    X</td> <td>308</td> </tr> <tr> <td>    Y</td> <td>185</td> </tr> <tr> <td>宽高</td> <td></td> </tr> <tr> <td>    宽度</td> <td>218</td> </tr> <tr> <td>    高度</td> <td>32</td> </tr> <tr> <td>是否隐藏</td> <td>false</td> </tr> <tr> <td>启用密码模式</td> <td>0</td> </tr> <tr> <td>输入限制</td> <td></td> </tr> </tbody> </table>	名称	值	控件名称	textbox_17	类型	textbox	ID值	17	位置		X	308	Y	185	宽高		宽度	218	高度	32	是否隐藏	false	启用密码模式	0	输入限制	
名称	值																											
控件名称	textbox_17																											
类型	textbox																											
ID值	17																											
位置																												
X	308																											
Y	185																											
宽高																												
宽度	218																											
高度	32																											
是否隐藏	false																											
启用密码模式	0																											
输入限制																												



函数名称	启用
textbox_callback	<input type="checkbox"/>

## 6.19. checkbox 复选框

### 6.19.1. 描述

复选框类似按钮控件，提供选中和没有选择的触发回调函数。

复选框的属性继承了 `object`，同时也派生出新的自己的特有属性。在创建一个复选框控件，设置属性时，必填属性会通过 `UIStudio` 工具配置。

### 6.19.2. 数据结构和函数

#### 复选框属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 点击触发回调函数，返回当前状态 */
    tui_checkbox_cb_t cb;

    int32_t value;          /* 外部配置，复选框的值，0 或者 1 */
} tui_checkbox_attr_t;
```

#### 复选框回调函数

```
typedef void(*tui_checkbox_cb_t)(tui_obj_t *obj, tui_event_e event, int32_t value);
```

#### 复选框函数

```
/* 创建复选框，par 是其父节点 */
tui_obj_t * tui_checkbox_create(tui_obj_t * par);
/* 设置复选框的属性 */
int tui_checkbox_set_attr(tui_obj_t *checkbox, tui_checkbox_attr_t *attri);
/* 获得复选框的属性 */
int tui_checkbox_get_attr(tui_obj_t *checkbox, tui_checkbox_attr_t *attri);
/* 获得复选框的值，0 或者 1 */
bool tui_checkbox_get_vaule(tui_obj_t *checkbox);
/* 设置复选框的值，0 或者 1 */
void tui_checkbox_set_vaule(tui_obj_t *checkbox, bool value);
```

### 6.19.3. 纯代码测试用例

```
static void tui_checkbox_cb(tui_obj_t *obj, tui_event_e event, int32_t value)
{
    printf("checkbox:%d\n", value);
}

static tui_obj_t * tui_checkbox(void)
{
    /*Create an checkbox*/
    tui_checkbox_attr_t attri = { 0 };
    tui_obj_t * checkbox;
```

```

/* 创建对象 */
checkbox = tui_checkbox_create(tui_layer_normal());
/* 设置属性 */
attri.obj.pt.x = 300;
attri.obj.pt.y = 500;
attri.obj.size.width = 120;
attri.obj.size.height = 30;
attri.cb = tui_checkbox_cb;
attri.value = 1;
tui_checkbox_set_attri(checkbox, &attri);

return checkbox;
}

```

## 6.19.4. 可视化工具编辑

拖拽<< 控件栏 列表中的 ☒ 复选框 到视图中，设置对应的参数，如下：

控件类型 checkbox

控件名称 checkbox\_18



控件参数

名称	值
控件名称	checkbox_18
类型	checkbox
ID值	18
位置	
X	485
Y	156
宽高	
宽度	30
高度	30
是否隐藏	false
状态值	选中

函数名称	启用
checkbox_callback	<input type="checkbox"/>

## 6.20. list 列表

### 6.20.1. 描述

列表控件比较灵活，可以动态或者静态增加列表选项，列表里面的选项，是一个一个特殊的按钮控件，按钮控件



里面又可以添加各种控件作为子控件，不同选项的高度都是可以调节，灵活多变。

列表的属性继承了 `object`，同时也派生出新的自己的特有属性。在创建一个列表控件，设置属性时，必填属性会通过 `UIStudio` 工具配置。

## 6.20.2. 数据结构和函数

### 列表属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 列表回调函数，返回当事件 */
    tui_list_cb_t cb;
    /* 列表滑动标记，供内部使用 */
    bool move_flag;
    /* 选择，供内部使用 */
    tui_obj_t * select_img;

    uint32_t bg_color;      /* 外部配置，列表的背景颜色（0xFF112233 FF是透明度；11是R；22是G；33是B） */
    uint32_t cur_index;     /* 外部配置，列表的当前索引焦点值 */
    tui_scrollbar_mode_e mode; /* 列表里面的滚条模式设置 */
} tui_list_attr_t;
```

### 列表回调函数

```
typedef void (*tui_list_cb_t)(tui_obj_t *obj_list, tui_event_e event, int32_t index);
```

### 列表函数

```
/* 创建列表，par 是其父节点 */
tui_obj_t * tui_list_create(tui_obj_t * par);
/* 设置列表的属性 */
int tui_list_set_attr(tui_obj_t *list, tui_list_attr_t *attri);
/* 获得列表的属性 */
int tui_list_get_attr(tui_obj_t *list, tui_list_attr_t *attri);
/* 增加列表项，每一项是一个 button，button 可以作为父节点放其他的控件 */
int tui_list_add_btn(tui_obj_t * btn);
/* 移除列表中的某一项，通过索引值移除 */
bool tui_list_remove_index(const tui_obj_t * list, uint16_t index);
/* 移除列表中的某一项，通过 button 句柄移除 */
bool tui_list_remove_btn(tui_obj_t * btn);
/* 移除列表中的所有项 */
bool tui_list_remove_all(const tui_obj_t * list);
/* 设置列表中的焦点项，通过 button 句柄设置 */
void tui_list_set_focus_btn(tui_obj_t * btn);
/* 设置列表中的焦点项，通过索引值设置 */
void tui_list_set_focus_index(tui_obj_t * list, uint16_t index);
/* 得到列表中的焦点项，返回 button 句柄 */
tui_obj_t * tui_list_get_focus_btn(const tui_obj_t * list);
/* 得到 button 句柄在列表中的索引值 */
int32_t tui_list_get_index_from_btn(const tui_obj_t * btn);
/* 通过列表中的索引值，得到 button 句柄 */
const tui_obj_t * tui_list_get_btn_from_index(const tui_obj_t * list, int32_t index);
```

```

/* 得到列表中所有项的总数 */
uint16_t tui_list_get_size(const tui_obj_t * list);
/* 设置列表中划条模式 */
void tui_list_set_scrollbar_mode(tui_obj_t * list, tui_scrollbar_mode_e mode);

```

### 6.20.3. 纯代码测试用例

```

static void tui_list_cb(tui_obj_t *obj, tui_event_e event, int32_t index)
{
    printf("list:%d, %d\n", event, index);
}

static tui_obj_t * tui_list(void)
{
    /*Create an button*/
    tui_button_attr_t attri = { 0};
    tui_bar_slider_attr_t attri_slid = { 0 };
    tui_label_attr_t attri_label = { 0 };
    tui_list_attr_t attri_list = { 0 };
    tui_obj_t * list, *button_1, *button_2, *button_3, *button_4, *button_5, *button_6, *bar_slid,
    *label;

    /* 创建对象 */
    list = tui_list_create(tui_layer_normal());
    /* 设置属性 */
    attri_list.obj.pt.x = 800;
    attri_list.obj.pt.y = 10;
    attri_list.obj.size.width = 200;
    attri_list.obj.size.height = 300;
    attri_list.cb = tui_list_cb;
    attri_list.bg_color = 0xFF000000; /* 列表的背景颜色 (0xFF112233 FF是透明度; 11是R; 22是G;
33是B) */
    attri_list.cur_index = 0; /* 列表的当前索引焦点值 */
    tui_list_set_attri(list, &attri_list);

    /* 增加一项 */
    button_1 = tui_button_create(list);
    tui_button_set_attri(button_1, &attri);
    label = tui_label_create(button_1);
    tui_label_set_attri(label, &attri_label);
    tui_list_add_btn(button_1);
    /* 增加一项 */
    button_2 = tui_button_create(list);
    attri.bg_color = 0xFF009F00;
    tui_button_set_attri(button_2, &attri);
    bar_slid = tui_bar_slider_create(button_2);
    tui_bar_slider_set_attri(bar_slid, &attri_slid);
    tui_list_add_btn(button_2);
    /* 增加一项 */

```




```

button_3 = tui_button_create(list);
attri.bg_color = 0xFF007F00;
tui_button_set_attri(button_3, &attri);
tui_list_add_btn(button_3);
/* 增加一项 */
button_4 = tui_button_create(list);
attri.bg_color = 0xFF005F00;
tui_button_set_attri(button_4, &attri);
tui_list_add_btn(button_4);
/* 增加一项 */
button_5 = tui_button_create(list);
attri.bg_color = 0xFF003F00;
tui_button_set_attri(button_5, &attri);
tui_list_add_btn(button_5);
/* 增加一项 */
button_6 = tui_button_create(list);
attri.bg_color = 0xFF001F00;
tui_button_set_attri(button_6, &attri);
tui_list_add_btn(button_6);

return list;
}

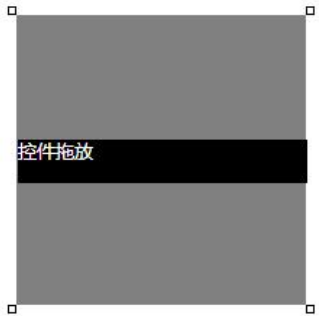
```

## 6.20.4. 可视化工具编辑

拖拽  控件栏 列表中的  列表 到视图中，列表中的每一项都是一个 button 按钮，所以在添加项的时候，需要将 button 按钮拖到列表中的“ 控件拖放”区域，button 按钮项里面也可以嵌套其他控件，列表设置对应的参数，如下图：

控件类型 list

控件名称 list\_19



控件参数

名称	值
控件名称	list_19
类型	list
ID值	19
位置	
X	417
Y	72
宽高	
宽度	200
高度	200
是否隐藏	false
背景颜色	-986896
当前索引值	0
滚条模式	2
选择项图片标记	

函数名称	启用
list_callback	<input type="checkbox"/>

6.21. multi\_screen 多屏控件

6.21.1. 描述

多屏控件提供了智能系统里面的屏幕滑动效果，屏幕数量可以自定义，不同屏幕里面可以放置不同的控件，实现多屏管理多 APP。

多屏控件的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个多屏控件，设置属性时，必填属性会通过 UIStudio 工具配置。

6.21.2. 数据结构和函数

多屏控件属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 屏幕滑动结束回调函数，返回当前屏幕索引值 */
    tui_multi_screen_cb_t cb;
    /* 动画 timer，供内部使用 */
    tui_timer_t *timer;
    /* 动画 timer，开始坐标，供内部使用 */
    int32_t start_x;
    /* 动画 timer，结束坐标，供内部使用 */
}
```

```

int32_t end_x;
/* 动画 timer, 计数, 供内部使用 */
int32_t cnt;

int32_t screen_w;          /* 外部配置, 多屏控件的宽, 和屏幕的宽保存一致 */
int32_t screen_h;          /* 外部配置, 多屏控件的高 */
uint8_t screen_num;        /* 外部配置, 多屏控件的屏数量 */
uint8_t cur_screen_index;  /* 外部配置, 多屏控件的当前屏的索引 */
} tui_multi_screen_attr_t;

```

### 多屏控件回调函数

```
typedef void (*tui_multi_screen_cb_t)(tui_obj_t *obj, tui_event_e event, int16_t index);
```

### 多屏控件函数

```

/* 创建多屏控件, par 是其父节点 */
tui_obj_t * tui_multi_screen_create(tui_obj_t * par);
/* 设置多屏控件的属性 */
int tui_multi_screen_set_attri(tui_obj_t *multi_screen, tui_multi_screen_attr_t *attri);
/* 获得多屏控件的属性 */
int tui_multi_screen_get_attri(tui_obj_t *multi_screen, tui_multi_screen_attr_t *attri);
/* 设置当前显示第几屏幕, is_anima 是否显示切换动画 */
int tui_multi_screen_set_cur_screen_index(tui_obj_t *multi_screen, uint8_t cur_screen_index, bool is_anima);
/* 获得当前显示第几屏幕 */
uint8_t tui_multi_screen_get_cur_screen_index(tui_obj_t *multi_screen);

```

## 6.21.3. 纯代码测试用例

```

static void tui_screen_slider_cb(tui_obj_t *obj, tui_event_e event, int16_t index)
{
    printf("index:%d\n", index);
}

static void tui_multi_screen(void)
{
    tui_button_attr_t attri = { 0 };
    tui_switch_btn_attr_t attri_sw = { 0 };
    tui_label_attr_t attri_label2 = { 0 };
    tui_label_attr_t attri_label = { 0 };
    /* Create an multi_screen */
    tui_multi_screen_attr_t attri_multi_screen = { 0 };
    tui_obj_t * multi_screen, *btn, *labell, *switch_btn;
    /* 创建对象 */
    multi_screen = tui_multi_screen_create(tui_layer_normal());
    /* 设置属性 */
    attri_multi_screen.obj.pt.x = 0;
    attri_multi_screen.obj.pt.y = 0;
    attri_multi_screen.obj.size.width = 0;
    attri_multi_screen.obj.size.height = 0;
    attri_multi_screen.cb = tui_screen_slider_cb;
}

```

```

attri_multi_screen.screen_w = 1024;          /* 多屏控件的宽，和屏幕的宽保存一致 */
attri_multi_screen.screen_h = 600;           /* 多屏控件的高 */
attri_multi_screen.screen_num = 3;           /* 多屏控件的屏数量 */
attri_multi_screen.cur_screen_index = 2;      /* 多屏控件的当前屏的索引 */
tui_multi_screen_set_attri(multi_screen, &attri_multi_screen);

btn = tui_button_create(multi_screen);
tui_button_set_attri(btn, &attri);

btn = tui_button_create(multi_screen);
attri.obj.pt.x = 480+30;
tui_button_set_attri(btn, &attri);

labell = tui_label_create(btn);
tui_label_set_attri(labell, &attri_label);

attri.obj.pt.x = 1024;
btn = tui_button_create(multi_screen);
tui_button_set_attri(btn, &attri);


attri.obj.pt.x = 1024+480+30;
btn = tui_button_create(multi_screen);
tui_button_set_attri(btn, &attri);
labell = tui_label_create(btn);
tui_label_set_attri(labell, &attri_label2);

attri.obj.pt.x = 2048;
btn = tui_button_create(multi_screen);
tui_button_set_attri(btn, &attri);

attri_sw.obj.pt.x = 2048 + 480 + 30;
switch_btn = tui_switch_btn_create(multi_screen);
tui_switch_btn_set_attri(switch_btn, &attri_sw);
labell = tui_label_create(switch_btn);
tui_label_set_txt(labell, "6789");
}

```

## 6.21.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  多屏控件 到视图中，设置对应的参数，如下：

控件类型	multi_screen	控件名称	multi_screen_20	<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div> <div> <div></div> <div></div> </div>
				<div> <div></div> <div> <div></div> <div></div> </div> </div>

## 6.22. canvas 画布

### 6.22.1. 描述

画布控件可以理解为是一块 ARGB 的 Buffer，可以在这个 Buffer 上面绘制各种图形，如，线、点、多边形、弧线、以及图片等。

画布的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个画布控件，设置属性时，必填属性会通过 UIStudio 工具配置。

### 6.22.2. 数据结构和函数

## 多屏控件属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 画布回调函数，返回当事件 */
    tui_canvas_cb_t cb;
    /* 画布的 ARGB buffer，供内部使用 */
    uint32_t *argb_buf;

    uint32_t bg_color;      /* 外部配置，画布的背景颜色（0xFF112233  FF 是透明度；11 是 R；22 是 G；33 是 B） */
}
```

```
} tui_canvas_attr_t;
```

### 多屏控件回调函数

```
typedef void (*tui_canvas_cb_t)(tui_obj_t *obj, tui_event_e event);
```

### 多屏控件函数

```
/* 创建画布, par 是其父节点 */
```

```
tui_obj_t * tui_canvas_create(tui_obj_t * par);
```

```
/* 设置画布的属性 */
```

```
int tui_canvas_set_attr(tui_obj_t *canvas, tui_canvas_attr_t *attr);
```

```
/* 获得画布的属性 */
```

```
int tui_canvas_get_attr(tui_obj_t *canvas, tui_canvas_attr_t *attr);
```

```
/* 绘制像素点 */
```

```
void tui_canvas_draw_pixel(tui_obj_t * canvas, int32_t x, int32_t y, uint32_t color);
```

```
/* 绘制像多点线 */
```

```
void tui_canvas_draw_line(tui_obj_t * canvas, const tui_point_t points[], uint32_t point_cnt, uint16_t width, uint32_t line_color);
```

```
/* 绘制矩形 */
```

```
void tui_canvas_draw_rect(tui_obj_t * canvas, int32_t x, int32_t y, int32_t w, int32_t h, uint32_t fill_color);
```

```
/* 绘制多边形 */
```

```
void tui_canvas_draw_polygon(tui_obj_t * canvas, const tui_point_t points[], uint32_t point_cnt, uint32_t fill_color);
```

```
/* 绘制圆弧形 */
```

```
void tui_canvas_draw_arc(tui_obj_t * canvas, int32_t x, int32_t y, int32_t r, int32_t start_angle, int32_t end_angle, uint16_t width, uint32_t line_color);
```

```
/* 绘制文本 */
```

```
void tui_canvas_draw_text(tui_obj_t * canvas, int32_t x, int32_t y, const char *txt, uint16_t fnt_size, uint32_t color);
```

```
/* 填充背景色 */
```

```
void tui_canvas_fill_bg(tui_obj_t * canvas, uint32_t color);
```

```
/* 填充 ARGB Buffer */
```

```
void tui_canvas_copy_buf(tui_obj_t * canvas, const void * buf, tui_coord_t canvas_x, tui_coord_t canvas_y, tui_coord_t buf_w, tui_coord_t buf_h);
```

```
/* 贴图片 */
```

```
void tui_canvas_draw_img(tui_obj_t * canvas, tui_coord_t x, tui_coord_t y, const char * path);
```

```
/* 获得画板的 ARGB Buffer 和画板的 buffer 的宽高 */
```

```
const uint32_t * tui_canvas_get_argb_buffer(tui_obj_t * canvas, uint32_t * out_width, uint32_t * out_height);
```

## 6.22.3. 纯代码测试用例

```
static void tui_canvas(void)
{
    tui_point_t points[3] = { { 10, 10 }, { 50, 15 }, { 80, 10 } };
    tui_point_t points_t[3] = { { 10, 80 }, { 70, 180 }, { 80, 80 } };
    tui_point_t points_r[5] = { { 0, 0 }, { 299, 0 }, { 299, 299 }, { 0, 299 }, { 0, 0 } };
    /* Create an container */
    tui_container_attr_t attr_container = { 0 };
    /* Create an canvas */
```



```


tui_canvas_attr_t attri_canvas = { 0 };
tui_obj_t * container;

/* 创建对象 */
container = tui_container_create(tui_layer_normal());
/* 设置属性 */
attri_container.obj.pt.x = 670;
attri_container.obj.pt.y = 240;
attri_container.obj.size.width = 400;
attri_container.obj.size.height = 400;
tui_container_set_attri(container, &attri_container);

tui_obj_t * canvas;
/* 创建对象，放在容器里面 */
canvas = tui_canvas_create(container);
/* 设置属性 */
attri_canvas.obj.pt.x = 50;
attri_canvas.obj.pt.y = 50;
attri_canvas.obj.size.width = 300;
attri_canvas.obj.size.height = 300;
tui_canvas_set_attri(canvas, &attri_canvas);
/* 绘制函数 */
tui_canvas_fill_bg(canvas, 0x3FFFFFF0);
tui_canvas_draw_pixel(canvas, 15, 15, 0xFF000000);
tui_canvas_draw_line(canvas, points, 3, 2, 0xFFFF0000);
tui_canvas_draw_line(canvas, points_r, 5, 10, 0xFFFF0000);
tui_canvas_draw_rect(canvas, 50, 150, 150, 150, 0xFF00FF00);
tui_canvas_draw_polygon(canvas, points_t, 3, 0xFF0000FF);
tui_canvas_draw_arc(canvas, 200, 100, 100, 0, 360, 2, 0xFF00FFFF);
tui_canvas_draw_text(canvas, 100, 200, "hello2345", 60, 0xFFFFF000);
tui_canvas_draw_img(canvas, 20, 20, "V:/image/10.bin");
}

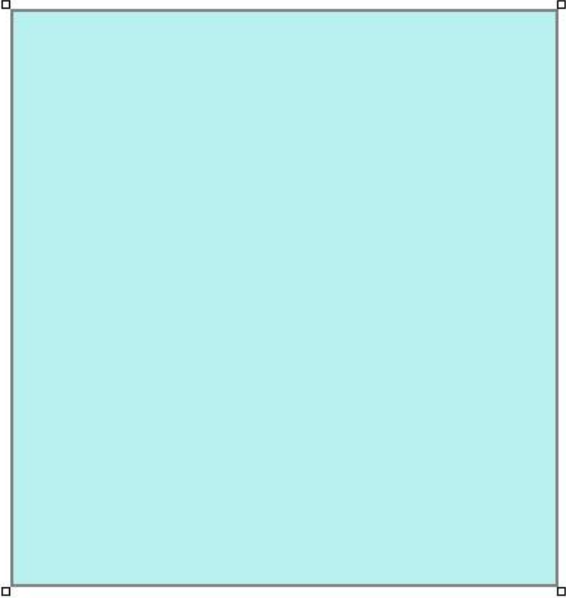
```

## 6.22.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  画布 到视图中，设置对应的参数，如下：

控件类型 canvas

控件名称 canvas\_21



▼ 控件参数

名称	值
控件名称	canvas_21
类型	canvas
ID值	21
位置	
X	358
Y	77
宽高	
宽度	369
高度	389
是否隐藏	false
背景颜色	-4591376

6.23. Qrcode 二维码

6.23.1. 描述

二维码生成器，只需要提供字符串，就能够方便的生成二维码图像，二维码的颜色，宽高都可以自定义。

二维码对象的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个二维码对象，设置属性时，必填属性会通过 UIStudio 工具配置。

6.23.2. 数据结构和函数

二维码属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attr_t obj;
    /* 二维码触发回调函数，返回当前字符串 */
    tui_qrcode_cb_t cb;
    /* 画布的 buffer，供内部使用 */
    uint32_t *cbuf;

    uint32_t bg_color; /* 外部配置，二维码的底色（0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B） */
    uint32_t fg_color; /* 外部配置，二维码的前景色（0xFF112233 FF 是透明度；11 是 R；22 是 G；33 是 B） */
    /*
    char *qrcode_chars; /* 外部配置，二维码字符内容 */
    */
} tui_qrcode_attr_t;
```

二维码回调函数

```
typedef void(*tui_qrcode_cb_t)(tui_obj_t *obj, tui_event_e event, const char * str);
```

## 二维码函数

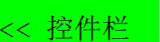

```
/* 创建二维码，par 是其父节点 */
tui_obj_t * tui_qrcode_create(tui_obj_t * par);
/* 设置二维码的属性 */
int tui_qrcode_set_attri(tui_obj_t *qrcode, tui_qrcode_attri_t *attri);
/* 得到二维码的属性 */
int tui_qrcode_get_attri(tui_obj_t *qrcode, tui_qrcode_attri_t *attri);
/* 得到二维码的字符串内容 */
const char * tui_qrcode_get_text(const tui_obj_t * qrcode);
/* 设置二维码的字符串内容 */
void tui_qrcode_set_text(tui_obj_t * qrcode, const char * qrcode_chars);
```

### 6.23.3. 纯代码测试用例

```
static tui_obj_t * tui_qrcode(void)
{
    /*Create an qrcode*/
    tui_qrcode_attri_t attri = { 0 };
    tui_obj_t * qrcode;
    /* 创建对象 */
    qrcode= tui_qrcode_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 300;
    attri.obj.pt.y = 300;
    attri.obj.size.width = 200;
    attri.obj.size.height = 200;
    attri.bg_color = 0xFFFFFFFF;
    attri.fg_color = 0xFF000000;
    attri.qrcode_chars = "http://www.baidu.com";
    tui_qrcode_set_attri(qrcode, &attri);

    return qrcode;
}
```

### 6.23.4. 可视化工具编辑

拖拽  控件栏 列表中的  二维码 到视图中，设置对应的参数，如下：

控件类型 **qrcode**

控件名称 **qrcode\_22**



▼ 控件参数

名称	值
控件名称	qrcode_22
类型	qrcode
ID值	22
位置	
X	429
Y	108
宽高	
宽度	232
高度	246
是否隐藏	false
内容	
风格	
背景颜色	-4380
前景颜色	

## 7. 声音对象

### 7.1. 描述

声音对象不是用来显示的，它用来播放提示音，该对象有一个优点，可以同时混播 8 种 wave 音源，各个声音播放互不干扰，不需要前面的声音停止，后面的声音才能播放。目前声音支持的音源是单声道，16 位宽，16K 采样率的 wav 文件

声音对象的属性继承了 object，同时也派生出新的自己的特有属性。在创建一个声音对象，设置属性时，必填属性会通过 UIStudio 工具配置。

### 7.2. 数据结构和函数

声音属性

```
typedef struct {
    /* 通用属性 */
    tui_object_attri_t obj;
    /* 声音回调函数，返回当事件 */
    tui_sound_cb_t cb;
    /* 声音 buffer 保存，供内部使用 */
    int16_t *sound;
    /* 声音 buffer 保存，供内部使用 */
    int32_t sound_len;
    /* 声音计数，供内部使用 */
    int32_t play_cnt;

    int32_t play_mode; /* 外部配置，声音控件播放模式， 0 是播放一次，1 是循环播放 */
} tui_sound_attri_t;
```

## 声音回调函数

```
typedef void (*tui_sound_cb_t)(tui_obj_t *obj, tui_event_e event);
```

## 声音函数

```
/* 创建声音，par 是其父节点 */
tui_obj_t * tui_sound_create(tui_obj_t * par);
/* 设置声音的属性 */
int tui_sound_set_attri(tui_obj_t *sound, tui_sound_attri_t *attri);
/* 得到声音的属性 */
int tui_sound_get_attri(tui_obj_t *sound, tui_sound_attri_t *attri)
/* 获得声音的属性 */
int tui_sound_set_sound_src(tui_obj_t *sound, const char *path);
/* 通过 gb2312 编码合成播放 TTS 中文声音 */
int tui_sound_set_sound_src_one_hz(tui_obj_t *sound, const char *gb2312_one_zh);
/* 声音播放 */
int tui_sound_play(tui_obj_t *sound);
/* 声音停止 */
int tui_sound_stop(tui_obj_t *sound);
/* 判断声音是否在播放 */
bool tui_sound_is_play(tui_obj_t *sound);
/* 停止所有音效，声音的全局函数 */
void tui_sound_enable(bool able);
```

## 7.3. 纯代码测试用例

```
static tui_obj_t * sound_tone;
static void tui_sound_tone(void)
{
    /* Create an sound */
    tui_sound_attri_t attri_sound = { 0 };
    /* 创建对象 */
    sound_tone = tui_sound_create(tui_layer_normal());
    /* 设置属性 */
    attri_sound.play_mode = 1;
    tui_sound_set_attri(sound_tone, &attri_sound);
    /* 设置音源 */
    tui_sound_set_sound_src(sound_tone, "V:\\sound\\tone.bin"); /* 确保加载了 res.iso, 并且路径文件存在 */
}

static tui_obj_t * sound_didi;
static void tui_sound_didi(void)
{
    /* Create an sound */
    tui_sound_attri_t attri_sound = { 0 };
    /* 创建对象 */
    sound_didi = tui_sound_create(tui_layer_normal());
    /* 设置属性 */
    attri_sound.play_mode = 0;
    tui_sound_set_attri(sound_didi, &attri_sound);
    /* 设置音源 */
```

```

    tui_sound_set_sound_src(sound_didi, "V:\\sound\\didi.bin");/* 确保加载了 res.iso, 并且路径文件
存在 */
}

static void tui_button_cb(tui_obj_t *obj, tui_event_e event)
{
    if (TUI_EVENT_RELEASED == event)
        tui_sound_play(sound_tone);
    else if (TUI_EVENT_PRESSED == event)
        tui_sound_play(sound_didi);

    printf("button:%d\n", event);
}

static tui_obj_t * tui_button(void)
{
    /*Create an button*/
    tui_button_attr_t attri = { 0 };
    tui_obj_t * button;
    /* 创建对象 */
    button = tui_button_create(tui_layer_normal());
    /* 设置属性 */
    attri.obj.pt.x = 500;
    attri.obj.pt.y = 150;
    attri.obj.size.width = 100;
    attri.obj.size.height = 40;
    attri.cb = tui_button_cb;
    attri.bg_color = 0xFFFFF00;          /* 按键的背景颜色 (0xFF112233 FF是透明度; 11是R; 22
是G; 33是B) */
    attri.border_color = 0xFFFF0000;     /* 按键的边框颜色 (0xFF112233 FF是透明度; 11是R; 22
是G; 33是B) */
    attri.border_width = 1;              /* 按键的边框线宽度 */
    tui_button_set_attri(button, &attri);

    tui_label_attr_t attri_label = { 0 };
    tui_obj_t * label;
    /* 创建对象, 父节点是 button */
    label = tui_label_create(button);
    /* 设置属性 */
    attri_label.obj.pt.x = 0;
    attri_label.obj.pt.y = 10;
    attri_label.obj.size.width = 100;
    attri_label.obj.size.height = 40;
    attri_label.fnt_size = 15;           /* 标签字体大小 */
    attri_label.txt = "hello";           /* 标签的文本信息 */
    attri_label.fnt_color = 0xFFFF0000;  /* 标签字体的颜色 (0xFF112233 FF是透明度; 11是R; 22
是G; 33是B) */
    attri_label.mode = 4;                /* 标签显示模式 (其中有滚动显示) */
    attri_label.align = TUI_LABEL_ALIGN_CENTER; /* 标签对齐方式 */
    tui_label_set_attri(label, &attri_label);
    /* 创建声音 */

```


```

tui_sound_tone();
tui_sound_didi();

return button;
}

```

## 7.4. 可视化工具编辑

拖拽<< 控件栏 列表中的  声音控件 到视图中，设置对应的参数，如下：

控件类型 sound

控件名称 sound\_23



⌵ 控件参数

名称	值
控件名称	sound_23
类型	sound
ID值	23
位置	
X	383
Y	134
宽高	
宽度	60
高度	60
是否隐藏	false
文件路径	V:/sound/tone.wav
播放模式	0

## 8. 字体制作

TUI 支持带虚边的点阵字体，和矢量字体。

- 可以通过 ttf 文件生成指定字符的点阵字体，这样能大大减少 ROM 空间，如图：其中可以设置字体大小，字符的范围和特殊字符指定，和对应的 ttf 字体轮廓。

工程配置

设备分辨率

串口设置

点阵字体

矢量字体

✱ 新增

✕ 删除

字体大小	数据源	字符资源	字体路径
15	字符文件 ▾	D:\[redacted]	D:\[redacted]
30	字符文件 ▾	D:\[redacted]	D:\[redacted]
60	自定义字 ▾	0123456789:-=	D:\[redacted]

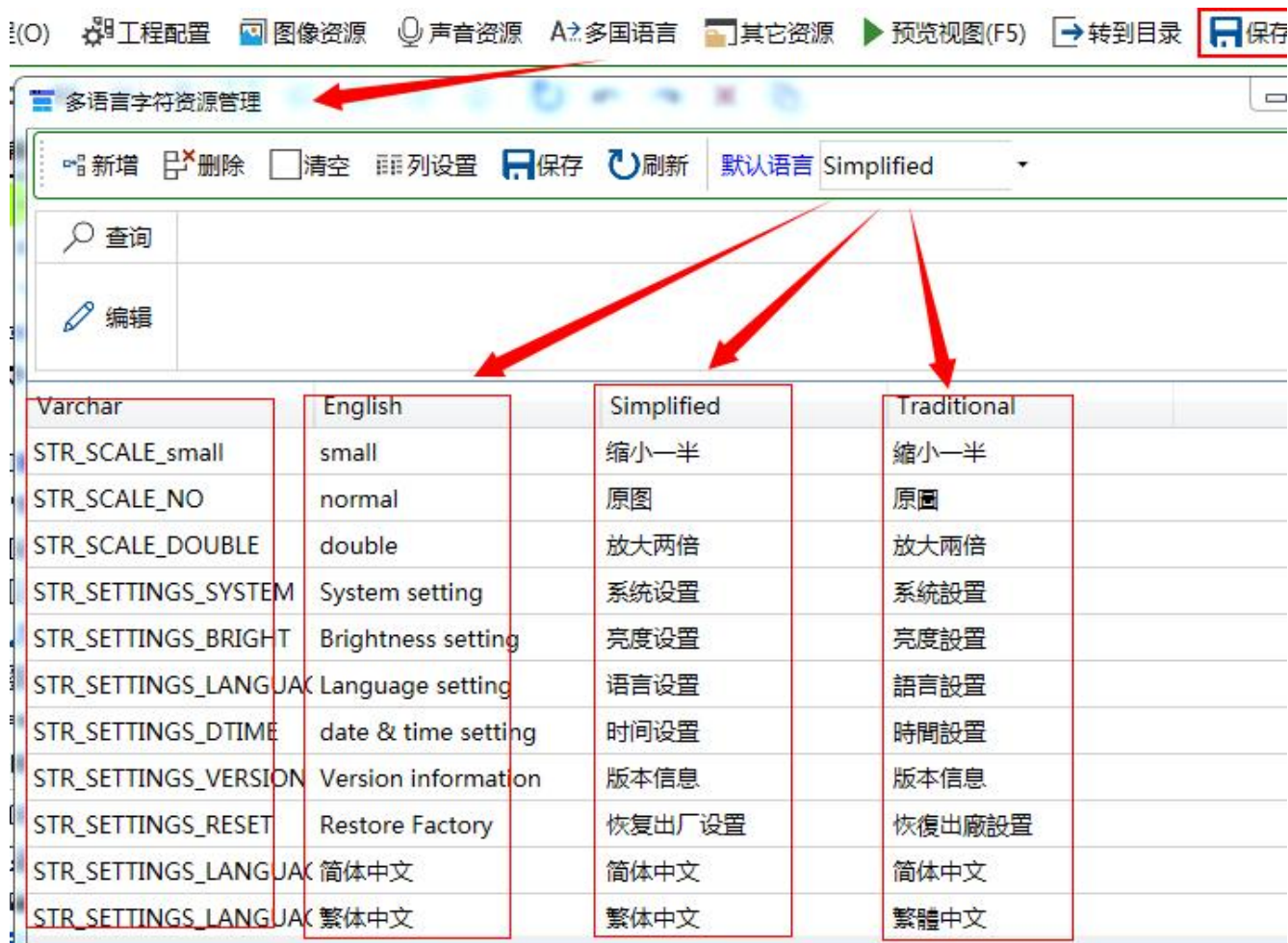
- 也支持 ttf 矢量字体，优点是支持任意大小的字符，缺点是占用 ROM 空间





## 9. 多国语言

TUI 支持多国语言，通过工具编辑好多国语言列表，第一列标示字符的唯一标识，可以找到对应的多国语言 utf8 字符串，通过“默认语言”设置当前的系统语言。



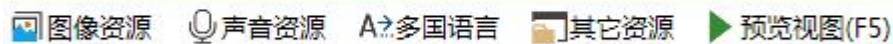
- 设置当前的国家语言 `void tui_set_language(int16_t lang_index)`，其他通过唯一标示，知道对应的国家语言列；
- 可以得到当前系统设置有几国语言 `int tui_get_language_num(void)`
- 字符串获得通过，可以得到对应的 utf8 字符串 `const char * tui_get_language_utf8_string(const char *str_id)`

## 10. 杂项

### 10.1. 资源打包和读取

- 资源打包都是通过 UISTudio 完成，其中包括图片、声音、多国字符、用户自定义资源，如图通过资源导入





和 编 辑：

， 最 终

将.png、.wav、.ttf 等后缀的文件打包成.bin 文件，最后将所有.bin 文件，打包到文件系统 V 盘符。

- 资源读取可以通过资源文件系统里面的文件路径，获得资源 buffer 和释放内存，`void *tui_alloc_buffer_from_fs(const char *path, int *get_buf_len);` 和 `int tui_free_buffer_from_fs(void *buf);` 其中路径对应 V 盘符下面的 imag、sound、other 等文件夹下面的.bin 文件。

## 10.2. 串口的使用

TUI 系统提供串口的 PC 模拟，就是說在不同的平台都可以使用硬件串口功能，提供了 3 个接口和一个回调读函数实现：

- `int tui_serial_port_open(char *com_name, int baudrate, int bytesize, int parity, int stopbits, serial_port_read_cb_t read_cb);` 打开串口驱动，参数 com\_name 在 windows 系统下需要填写，参考



嵌入式平台不需要填写，参数 baudrate, bytesize, parity, stopbits 是比特率、位宽、校验和停止位基本参数，如果工具有配置，直接填 -1。最后 read\_cb 是串口收数据的回调函数，收到数据的 buffer 和长度在参数中。

- `void tui_sserial_port_close(void);` 关闭串口驱动
- `int tui_sserial_port_write(char *write_buff, int buf_len);` PC 上发送串口数据，参数是发送的数据和数据长度。

## 10.3. 获取触摸和按键的信息

- 通过调用函数，获得当前触摸的信息 `void tui_get_point_value(uint8_t *st, int32_t *x, int32_t *y);`
- 通过调用函数，获取当前按键的信息 `void tui_get_key_value(uint8_t *st, uint32_t *key_value);`
- 通过注册触摸和按键回调函数，实时获取触发信息，累计注册不要超过 24 个，要及时释放

```
typedef void (*tui_point_trigger_cb_t)(uint8_t state, int32_t x, int32_t y);
```

```
typedef void (*tui_key_trigger_cb_t)(uint8_t state, uint32_t key_value);
```

```
void tui_point_trigger_cb_reg(tui_point_trigger_cb_t cb);
```

```
void tui_point_trigger_cb_unreg(tui_point_trigger_cb_t cb);
```

```
void tui_key_trigger_cb_reg(tui_key_trigger_cb_t cb);
```

```
void tui_key_trigger_cb_unreg(tui_point_trigger_cb_t cb);
```

## 10.4. 获取系统运行时间、RTC 日期、休眠

- 通过调用函数，获得当前系统运行时间，单位毫秒 `uint32_t tui_get_system_run_milliseconds(void);`
- 通过调用函数，获得时间日期时间 `tui_time_t tui_get_localtime(void);`
- 通过调用函数，系统休息，释放 CPU `void tui_sleep(uint32_t ms);`

## 10.5. Gb2312 和 utf8 码值转换

- 将 utf8 转换成 gb2312, `int tui_utf8_to_gb2312(char *gb2312_str, int len_gb2312, char *utf8_str, int len_utf8);`
- 将 gb2312 转换成 utf8, `int tui_gb2312_to_utf8(char *utf8_str, int len_utf8, char *gb2312_str, int len_gb2312);`
- gb2312 转码的时候可以提前获取需要的内存长度 `int tui_gb2312_to_utf8_need_size(char *gb2312_str);`

## 10.6. 获得工程配置信息

- 获得工程配置的屏幕分辨率 `int tui_config_get_screen_resolution(int *screen_hor_res, int *screen_ver_res)`
- 获得工程配置版本号 `const char * tui_config_get_version(void)`
- 对比工程配置的密码返回 0 成功, -1 失败 `int tui_config_password_compare(const char * password)`
- 获得工程配置的串口信息 `int tui_config_get_serial_port(int *baudrate, int *bytesize, int *parity, int *stopbits)`
- 获得工程配置的屏幕旋转信息 `int tui_config_get_screen_rotate_angle(void);`

## 10.7. 获得 TUI 内核信息和自动化测试用于调试

- 导出创建的对象个数、导出创建定时器的个数、导出注册系统回调函数个数、以及导出当前内存使用的大小。通过函数 `void tui_dbg_core_information_dump(void)` 打印所有信息, 方便查找内存是否泄露。
- 自动化测试函数 `void tui_pointer_run_auto_test(char * rec_path, int is_loop_run);` 输入特殊格式的 buffer 自动测试 (如 000000000 00 0000 0000\r\n 十进制 9 位时间毫秒单位; 十进制 2 位按下抬起状态; 十进制 4 位 X 坐标; 十进制 4 位 Y 坐标; 换行标记结束)

# 11. FAQ

TUI 相关问题的 FAQ

调试问题记录

问题现象

原因分析

解决办法

# 12. 总结

总结: 接口简单, 使用方便, 快速开发。