

GUI模拟器Lua脚本语法

自定义lua脚本语法

Lua脚本语法说明文档

UI模块 ui

获取控件句柄

切换页面、工程

页面相关

普通控件 obj

常规

属性

控件Control

旋转

自定义属性Attr

坐标Pos

画图Draw

事件Event

特效

✖时间日期DateTime

滑动容器Scroll控件

常规

子控件

子项Item

事件

特效

工具类模块 utils

滑动页面模块 slide

系统接口 sim_system

文件模块 sim_file

模拟数据 sim_virtual

传感器模块 `sim_sensor`

UI工具模拟器操作说明

Lua脚本语法说明文档

待完善

文本控件：滚动、截取字符串指定长度显示

满天星

圆弧菜单

事件触发模拟

画圆弧接口

文本、数字、时间控件图片拼接设置接口

.

UI模块 ui

获取控件句柄

获取控件句柄

ui:getComponentByName(ename)

根据控件名称ename获取控件对象,普通控件（除了列表和表格控件外）

ename: 控件名称

```
1 ui:getComponentByName('BTN_ID')
```

Lua | [复制代码](#)

ui:getScrollListByName(ename)

根据控件名称ename获取容器控件对象，（列表控件）

ename: 控件名称

```
1 ui:getScrollListByName('LIST_VIEW_ID')
```

Lua | [复制代码](#)

ui:getScrollGridByName(ename)

根据控件名称ename获取容器控件对象（表格控件）

ename: 控件名称

```
1  local ename = 'BaseForm_231'
2  local type = ui:getControlType(ename)
3  print(ename .. ":" .. type)
4  if type == 'NewGrid' then
5      local grid = ui:getScrollGridByName(ename)
6      local ctrls = grid:getSubControls()
7      print("表格控件" .. #ctrls)
8  elseif type == 'VerticalList' then
9      local list = ui:getScrollListByName(ename)
10     local ctrls = list:getSubControls()
11     print("垂直列表" .. #ctrls)
12 elseif type == 'HorizontalList' then
13     local list = ui:getScrollListByName(ename)
14     local ctrls = list:getSubControls()
15     print("水平列表" .. #ctrls)
16 end
17
```

ui:getContainerByName(ename)

根据ename获取容器里控件 水平列表、垂直列表、表格控件

切换页面、工程

切换页面、工程

ui:switchPageByName(ename, type)

根据页面名称切换页面

ename: 页面名称

type: 切换方式, left right up down

```
1 ui:switchPageByName('PAGE_1', 'right')
```

Lua

[复制代码](#)

ui:switchPage(index, type)

根据页面序号切换页面

index: 页面序号从0开始

type: 切换方式, 0-left,1-right, 2-up, 3-down

```
1 ui:switchPage(1, 'left')
```

Lua

[复制代码](#)

ui:switchPageByName(ename, type)

根据页面ID切换页面

ename: 页面需要从0开始 PAGE_0

type: 切换方式, 0-left,1-right, 2-up, 3-down

ui:switchPageByHexAddr(hex, type)

根据ID十六进制切换页面 (转成10进制也可以)

hex: 页面ID十六进制值, ename.h 里面的宏定义

type: 切换方式, 0-left,1-right, 2-up, 3-down

Lua | [复制代码](#)

```
1 ui:switchPageByHexAddr(0x282000A, 'right')
2 ui:switchPageByHexAddr(42074122, 'right')
```

Lua | [复制代码](#)

[illegible]

ui:backToPreviousPage(type)

对通过switchPage切换页面后，进行返回，返回上一个页面

注意：跨工程切换无效

type: 切换方式

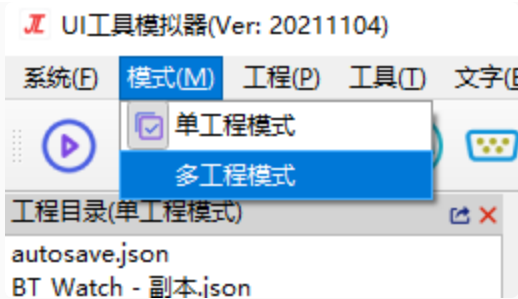
Lua [复制代码](#)

```
1 ui:backToPreviousPage('right')
```

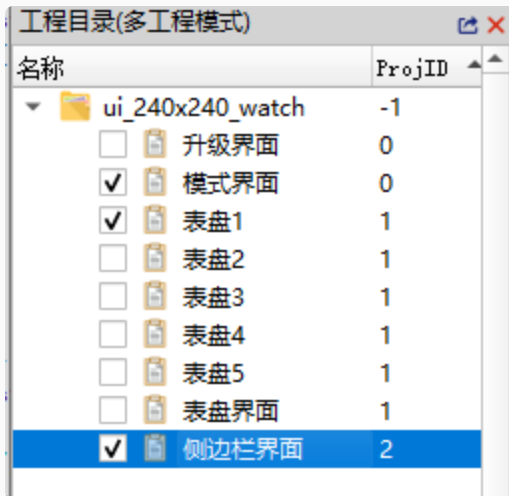
```
ui:switchProject(projid, pageid)
```

切换工程

通过模式-多工程模式进行切换到多工程模式。



然后对需要作为运行的工程进行勾选，如果不勾选，那么默认第一个作为对应工程ID的工程。



lua代码中，通过`ui:switchProject(projid, pageid)`，如下面代码，就表示切换到模式界面对应工程的第一个页面。

下面的第二行代码，表示切换到侧边栏界面的第二个页面。

Lua [复制代码](#)

```
1 ui:switchProject(0, 0)
2 ui:switchProject(2, 1)
```

ui:switchProjectByAddrHex(hex)

同上，也是切换页面，但是传入的值是 `ename.h` 中对应宏定义里面的十六进制值。

ui:reloadPage(pageid)

重新加载页面，除了工具上的Ctrl+R重新加载当前页外，也可以通过代码重新加载当前页。

ui:reloadProject(pageid)

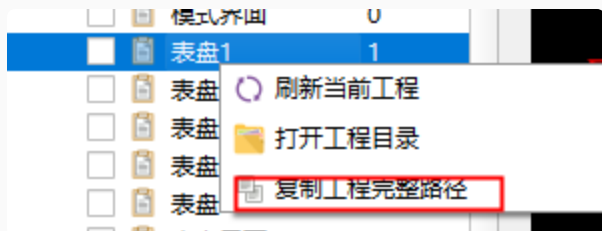
重新加载工程，并跳转到对应的PAGE页

ui:replaceProjectPage(filename, frompageid, topageid)

从filename这个工程中，加载对应的frompageid页面内容，替换掉当前运行中的topageid页面内容，然后执行ui:switchPage(0) 进行切换，实现表盘切换和加载多工程效果。

Lua | [复制代码](#)

```
1 ui:replaceProjectPage("C:/ui_240x240_watch/表盘2/project/dial.json", 0, 0)
2 ui:switchPage(0)
```



页面相关

页面相关

ui:getControlType(ename)

同控件ename获取控件类型，返回工程json文件里面控件的-type字段

```
1 local type = ui:getControlType('BTN')
```

Lua | [复制代码](#)

ui:getPageCount()

获取当前工程总共有多少个页面

```
1 local count = ui:getPageCount()
```

Lua | [复制代码](#)

ui:getCurrPageID()

获取当前运行状态下的页面PAGE 返回整数

```
1 local id = ui:getCurrPageID()
```

Lua | [复制代码](#)

ui:getCurrPageName()

获取当前运行状态下的页面PAGE 名称 PAGE_1 PAGE_2 返回字符串

```
1 local pagename = ui:getCurrPageName()
```

Lua | [复制代码](#)

ui:getPrevPageID()

获取上一个页面ID

ui:getPrevPageName()

获取上一个页面名称

ui:getListPrevPageID()

获取页面栈里页面ID集合列表（数组）

ui:getListPrevPageName()

获取页面栈里页面名称集合列表（数组）

Lua | [复制代码](#)

```
1  ui:switchPageByName('PAGE_1')
2  ui:switchPageByName('PAGE_0')
3  local pageid = ui:getCurrPageID();
4  print('curr id:' .. pageid)
5  local pagename = ui:getCurrPageName();
6  print('curr name:' .. pagename)
7  print('-----')
8  pageid = ui:getPrevPageID()
9  print('prev id:' .. pageid)
10 pagename = ui:getPrevPageName()
11 print('prev name:' .. pagename)
12 print('-----')
13 local list = ui:getListPrevPageID();
14 print('prev ids:')
15 print(list)
16 list = ui:getListPrevPageName()
17 print('prev names:')
18 print(list)
```

运行结果

Lua | [复制代码](#)

```
1  curr id:0
2  curr name:PAGE_0
3  -----
4  prev id:1
5  prev name:PAGE_1
6  -----
7  prev ids:
8  类型为: table, 内容为: {1=0,2=1}
9  prev names:
10 类型为: table, 内容为: {1='PAGE_0',2='PAGE_1'}
11  -----
12 本次执行耗时: 13ms (2021-09-03 16:45:48.506)
```

ui:touchEventTakeOverOn()

事件接管，事件终止，表示父控件不执行（触发）当前事件

ui:touchEventTakeOverOff()

事件委托，执行完当前事件代码后，继续执行父控件当前类型事件

ONLOAD UNLOAD TOUCH_DOWN TOUCH_MOVE TOUCH_R_MOVE TOUCH_L_MOVE TOUCH_D_MOVE TOUCH_

Event: TOUCH_DOWN [Type:2]

```
1 local status = 1
2 if status==1 then
3     ui:touchEventTakeOverOn(); --事件接管
4 else
5     ui:touchEventTakeOverOff(); --事件不接管，执行后触发父控件事件
6 end
7 print('aaa')
8
```

图中功能表示，如果按下当前按钮，将触发TOUCH_DOWN事件里面的代码。

但是，当status==1时，那么就执行完当前代码段后，就结束整个事件触发流程。

当status~=1时，那么就执行完当前代码段后，主动触发该按钮的父控件（所处布局）的TOUCH_DOWN事件

Lua [复制代码](#)

```
1 local status = 1
2 if status==1 then
3     ui:touchEventTakeOverOn(); --事件接管
4 else
5     ui:touchEventTakeOverOff(); --事件不接管，执行后触发父控件事件
6 end
7 print('aaa')
8
```

普通控件 obj

普通控件模块

除了表格控件和列表控件外的所有控件

控件模块，是通过ui.getComponentByName(ename) 获取对象后，通过对象访问其属性

Lua [📄复制代码](#)

```
1  local obj = ui:getComponentByName('BTN') --获取Ename对应控件对象
2  obj:show() --通过对象进行操作（显示）
```

常规

常规

obj:show()

控件进行显示

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:show()
```

obj:hide()

控件进行隐藏

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:hide()
```

obj:setRect(x, y, w, h)

设置控件的位置信息

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:setRect(10, 10, 100, 40)
```

obj:isHide()

获取当前控件显示状态

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:isHide()
```

obj:setMovable(bool)

设置控件是否可移动

obj:getMovable()

获取控件是否可移动状态

obj:setMovableRange(bool)

设置空间是否可移动（并限制在父控件内）

obj:getMovableRange()

获取控件是否可移动状态

obj:setBackgroundColor(color)

设置控件背景颜色

obj:setBackgroundImage(filename)

设置背景图片，filename为绝对路径或相对路径图片地址

Lua | [复制代码](#)

```
1  local u = ui:getComponentByName('BaseForm_12')
2  u:setBackgroundImage('C:\\Users\\ZPC19-023\\Pictures\\slider-001.jpg')
3
4  local u = ui:getComponentByName('BaseForm_12')
5  print(sim_system:getWorkspacePath())
6  u:setBackgroundImage(sim_system:getWorkspacePath() .. '\\config\\m26.png')
```

obj:setBackgroundScale(scale)

设置背景图片，缩放比例. 取值 (0 ~ 1)

Lua | [复制代码](#)

```
1  local u = ui:getComponentByName('BaseForm_12')
2  u:setBackgroundScale(0.8)
```

obj:info()

打印控件调试信息

属性

属性

obj:showCaption(text)

如果是非文本控件，想要显示文本，可通过该接口显示纯文本提示

Lua | [复制代码](#)

```
1 local obj = ui:getComponentByName('Text')
2 obj:showCaption('2020-01-01 17:29:10')
```

数字控件

obj:setNumber(num)

控件如果是数字控件，那么可以对其设置对应数值

obj:getNumber()

获取控件数值，由于Lua是没有区分int，short，float，double的，所以业务层要自己处理

文本控件

obj:setText(text)

控件如果是文本控件（类型为ascii），那么可以对其设置文本

Lua | [复制代码](#)

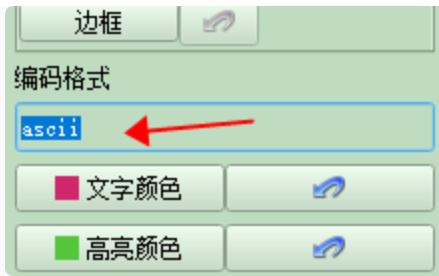
```
1 local obj = ui:getComponentByName('Text')
2 obj:setText('ascii')
```

obj:getText()

获取控件文本

obj:getTextType()

获取文本控件文本类型 类型有ascii、picstr



Lua | [复制代码](#)

```
1  local txt = ui:getComponentByName('Text')
2  local type = txt:getTextType()
3  print(type)
4  if type == 'ascii' then
5      txt:setText('ASCII Code') --显示ascii文字
6  else
7      txt:showTextFontByIndex(0) --显示多国语言
8  end
```

obj:setTextHighLight(bool)

设置文本控件（数字控件）是否高亮颜色显示

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('Text')
2  obj:setTextHighLight(true)
```

obj:showTextFontByIndex(index)

控件如果是文本控件（类型为strpic），那么可以对其设置显示的文本

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('Text')
2  obj:showTextFontByIndex(0)
```

obj:getTextFontIndex()

获取strpic类型多国语言文本控件当前显示文本的id号

obj:getTextFontNumber()

获取strpic类型多国语言文本控件包含文本数

```

1  local txt = ui:getComponentByName('Text')
2  txt:showTextFontByIndex(2)
3  print(txt:getTextFontIndex())
4  print(txt:getTextFontNumber())

```

obj:setTextImage(text)

以图片方式设置文本

```

1  local timer = ui:getComponentByName('Timer')
2  timer:setTextImage("01:22")
3  local num = ui:getComponentByName('Num')
4  num:setTextImage("022")

```



obj:getTextImage(text)

获取以图片方式设置的文本内容

obj:setTextOffset(offset)

传入偏移整数，设置文本控件内文本偏移，通过定时器实现字幕滚动效果

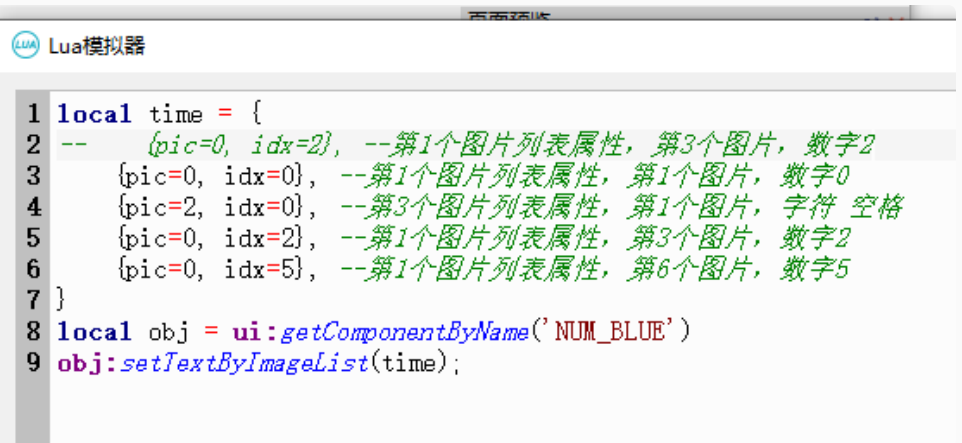
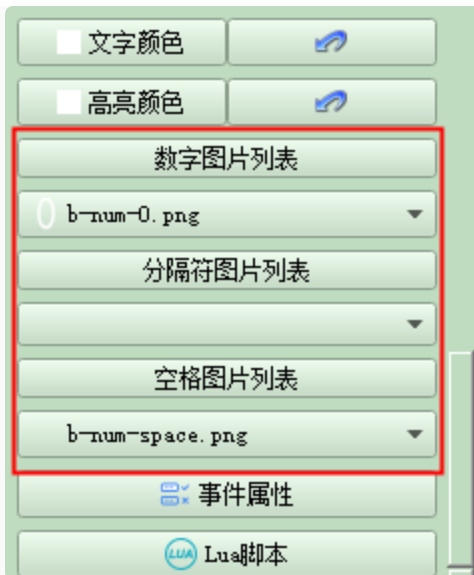
offset大于0，表示向左滑动

offset小于0，表示向右滑动

注意 具体使用技巧可以参考特效页

obj:setTextByImageList(list)

通过list传入图片列表属性，拼接成背景图



Lua [复制代码](#)

```

1  local time = {
2      {pic=0, idx=2}, --第1个图片列表属性, 第3个图片, 数字2
3      {pic=0, idx=0}, --第1个图片列表属性, 第1个图片, 数字0
4      {pic=1, idx=0}, --第2个图片列表属性, 第1个图片, 字符:
5      {pic=0, idx=2}, --第1个图片列表属性, 第3个图片, 数字2
6      {pic=0, idx=5}, --第1个图片列表属性, 第6个图片, 数字5
7  }
8  local obj = ui:getComponentByName('NUM_BLUE')
9  obj:setTextByImageList(time);
10 local time = {
11     {pic=0, idx=2}, --第1个图片列表属性, 第3个图片, 数字2
12     {pic=0, idx=0}, --第1个图片列表属性, 第1个图片, 数字0
13     {pic=2, idx=0}, --第3个图片列表属性, 第1个图片, 字符 空格
14     {pic=0, idx=2}, --第1个图片列表属性, 第3个图片, 数字2
15     {pic=0, idx=5}, --第1个图片列表属性, 第6个图片, 数字5
16 }
17 local obj = ui:getComponentByName('NUM_BLUE')
18 obj:setTextByImageList(time);

```

高亮

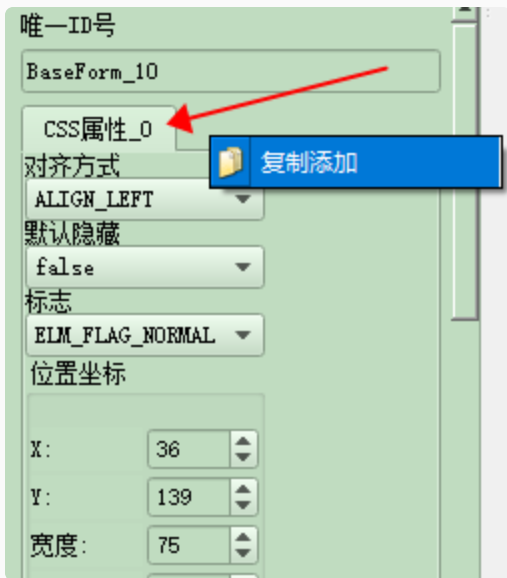
obj:setCss(index)

切换控件CSS属性页（设置正常与高亮）

Lua [复制代码](#)

```
1 local obj = ui:getComponentByName('Active')
2 obj:setCss(0)
```

由于UI工具是通用性，动态属性，动态控件



这里是CSS属性，有些控件用来实现普通模式和高亮模式，那么就需要这样封装

Lua [复制代码](#)

```
1 --设置普通模式
2 function setNormalMode()
3     obj:setCss(0)
4 end
5 --设置高亮模式
6 function setHighLightMode()
7     obj:setCss(1)
8 end
```

obj:getCss(index)

Lua [复制代码](#)

```
1  local obj = ui:getComponentByName('Active')
2  obj:setCss(0)
3  local table = obj:getCss(0);
4  utils:print(table)
5  print(table.rect.x)
```

Lua [复制代码](#)

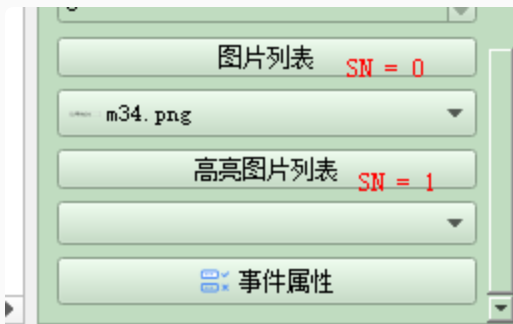
```
1  {
2    'border'={
3      'color'='', 'height'=1, 'width'=1, 'x'=0, 'y'=0
4    },
5    'hide'=false,
6    'rect'={
7      'height'=78, 'width'=196, 'x'=33, 'y'=24
8    }
9  }
```

obj:showImageByIndex(sn, idx)

设置带有图片列表属性功能控件，显示对应图片列表属性上图片功能

sn: 表示第sn个图片列表属性

idx: 表示显示第sn个图片列表属性里面的第idx个图片



```

1  local obj = ui:getComponentByName('Picture')
2  obj:showImageByIndex(0, 0) --显示图片列表第0个图片
3  obj:showImageByIndex(1, 0) --显示高亮图片列表第0个图片
4  local obj = ui:getComponentByName('Battery')
5  obj:showImageByIndex(0, 0) --显示电量图片列表第0个图片
6  obj:showImageByIndex(1, 0) --显示充电图片列表第0个图片

```

obj:getImageIndex(sn)

返回第sn个图片列表当前显示的是哪个序号的图片

obj:getImageNumber(sn)

返回第sn个图片列表属性总共有多少个图片

```

1  local pic = ui:getComponentByName('BaseForm_2') --获取控件
2  print(pic) --打印控件信息
3  pic:showImageByIndex(0, 1) --控件切换到图片
4  print(pic:getImageIndex(0)) --图片控件当前显示序号
5  print(pic:getImageNumber(0)) --图片控件数量
6
7  cnt = pic:getImageNumber(0)
8  cnt = cnt - 1
9  for i=0,cnt do
10     print(i)
11     pic:showImageByIndex(0, i)
12 end
13
14 --封装普通图片列表属性显示
15 function showNormalImageByIndex(idx)
16     pic:showImageByIndex(0, idx)
17 end
18 --封装高亮图片列表属性显示
19 function showHighLightImageByIndex(idx)
20     pic:showImageByIndex(1, idx)
21 end

```

控件Control

控件

obj:getControlEname()

获取控件ID, Ename

Lua | [复制代码](#)

```
1  local btn = ui:getComponentByName('Active')
2  print(btn:getControlEname())
```

obj:getControlType()

获取控件类型，这里的类型是工程json文件里面的-type，该函数getControlType与ui:getControlType效果一致

Lua | [复制代码](#)

```
1  local btn = ui:getComponentByName('Active')
2  print(btn:getControlType())
```

obj:getComponentParent

获取控件的父控件，父控件必须是普通控件，如果父控件是水平列表，垂直列表或者表格控件的，请通过getScrollListParent或getScrollGridParent获取

Lua | [复制代码](#)

```
1  local pic = ui:getComponentByName('Pic')
2  print(pic)
3  local layout = pic:getComponentParent()
4  if layout ~= nil then
5      print(layout)
6  end
```

obj:getScrollListParent

获取控件的父控件，父控件必须是列表控件（包含水平列表，垂直列表）

obj:getScrollGridParent

获取控件的父控件，父控件必须是表格控件

Lua [复制代码](#)

```
1  local pic = ui:getComponentByName('Pic')
2  local parent = pic:getComponentParent()
3  if parent == nil then
4      parent = pic:getScrollListParent()
5  end
6  if parent == nil then
7      parent = pic:getScrollGridParent()
8  end
9
10 if parent == nil then
11     print("当前控件找不到父控件")
12 end
```

obj:createWidget(object)

动态创建子控件

x,y,w,h 表示动态创建控件xy坐标及其对应的宽高

name 表示控件的Ename，全局唯一名称，后续通过 `ui:getComponentByName('ename')` 获取控件对象

Lua [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:createWidget({x=1, y=1, w=100, h=100, name='SUB_BTN'})
3  local obj2 = ui:getComponentByName('SUB_BTN')
4  obj2:info()
```

obj:getSubControls

获取所有子控件（不包含滑动控件：表格控件，列表控件）

obj:getSubControlsNumber

获取所有子控件数量

```

1  local layout = ui:getComponentByName('LAYOUT')
2  print(layout)
3  local subs = layout:getSubControls()
4  print(#subs)
5  local nums = layout:getSubControlsNumber()
6  print(nums)
7  for i=1, nums do
8      print(subs[i]:getControlName())
9  end

```

obj:getSubControlsName

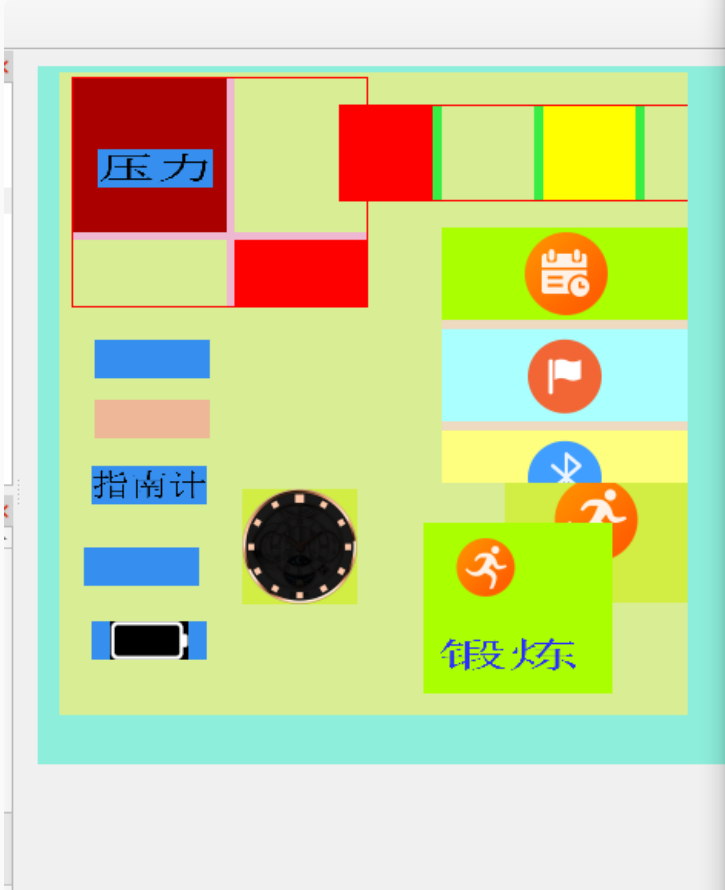
获取所有子控件唯一ID号

obj:getSubScrolls

获取所有滑动容器控件（表格控件，列表控件）

obj:getSubScrollsNumber

获取所有滑动容器控件数量



```

1  local layout = ui:getComponentByName('BaseForm')
2  print(layout)
3  local subs = layout:getSubControls()
4  print(#subs)
5  local nums = layout:getSubControlsNumber()
6  print(nums)
7  for i=1, nums do
8      print(subs[i]:getControlName() .. "-" .. subs[i]:getControlID())
9  end
10 subs = layout:getSubScrolls()
11 print(#subs)
12 nums = layout:getSubScrollsNumber()
13 print(nums)
14 for i=1, nums do
15     print(subs[i]:getControlName() .. "-" .. subs[i]:getControlID())
16 end

```

运行(F5)

BaseForm_230-Text
 BaseForm_239-Button
 BaseForm_240-Battery
 BaseForm_241-ImageList
 3
 3
 BaseForm_220-NewGrid
 BaseForm_226-VerticalList
 BaseForm_231-HorizontalList

 本次执行耗时: 6ms (2021-08-04 15:14:57.302)

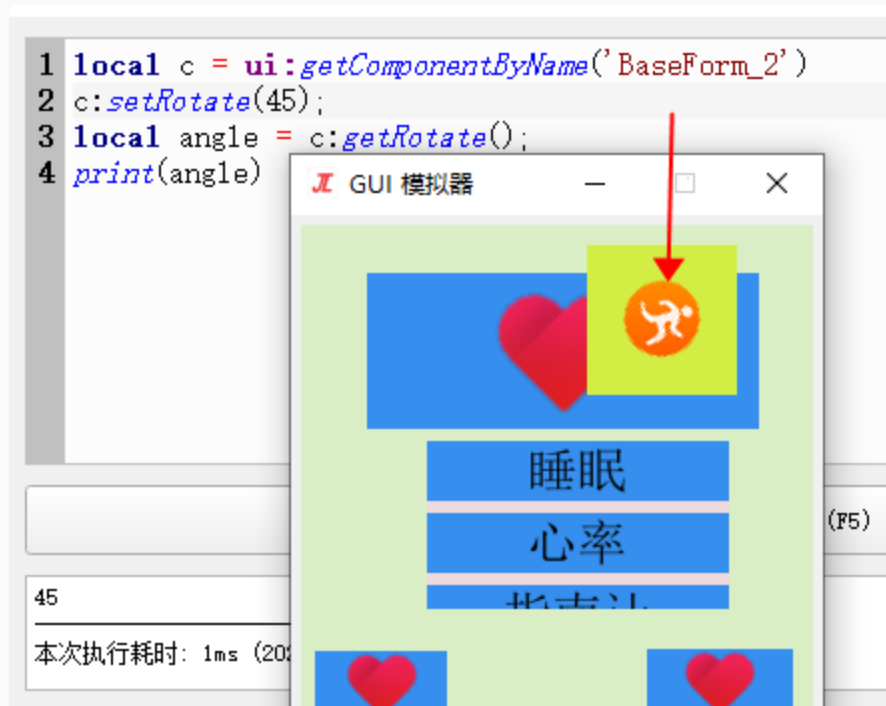
旋转

obj:setRotate(angle)

设置旋转角度，以圆心为圆点，顺时针旋转angle度

obj:getRotate()

获取旋转角度



obj:setRotateByPos(angle, rx, ry, dx, dy)

设置旋转angle角度，以坐标(dx, dy)相对于父控件，以圆心(rx, ry)相对于当前控件，顺时针旋转angle度。

这里的坐标(dx, dy)是以父控件的左上角作为坐标圆点。

Lua | [复制代码](#)

```
1 local hour = ui:getComponentByName('NewFrame_2')
2 hour:setRotateByPos(10, 13, 9, 120, 120)
```

obj:getRotateByPos()

获取旋转角度信息

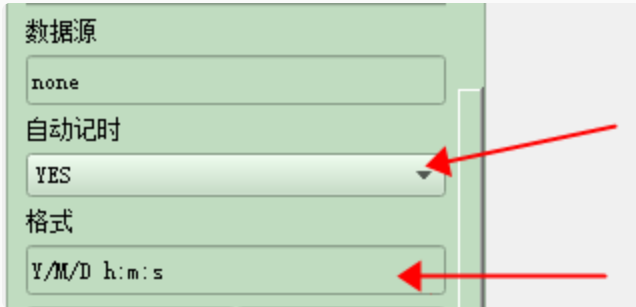
```
1  utils:clearAllTimer();
2  function updateTime()
3      local time = utils:getSystemTime();
4
5      local hour = time.hour;
6      local min = time.minute;
7      local sec = time.second;
8
9      local dsec = sec / 60 * 360;
10     local dmin = (min + (sec / 60)) / 60 * 360;
11     local dhour = (hour + (min / 60)) / 12 * 360;
12     print(dhour, dmin, dsec)
13     --hour
14     local u = ui:getComponentByName('NewFrame_2')
15     local rx = u:getAttrNumberByIndex(0)
16     local ry = u:getAttrNumberByIndex(1)
17     local dx = u:getComponentParent():getWidth();
18     local dy = u:getComponentParent():getHeight();
19     u:setRotateByPos(dhour, rx, ry, dx/2, dy/2);
20     --minute
21     u = ui:getComponentByName('NewFrame_3')
22     rx = u:getAttrNumberByIndex(0)
23     ry = u:getAttrNumberByIndex(1)
24     dx = u:getComponentParent():getWidth();
25     dy = u:getComponentParent():getHeight();
26     u:setRotateByPos(dmin, rx, ry, dx/2, dy/2);
27     --second
28     u = ui:getComponentByName('NewFrame_4')
29     rx = u:getAttrNumberByIndex(0)
30     ry = u:getAttrNumberByIndex(1)
31     dx = u:getComponentParent():getWidth();
32     dy = u:getComponentParent():getHeight();
33     u:setRotateByPos(dsec, rx, ry, dx/2, dy/2)
34 end
35
36 utils:createTimer(1000, updateTime);
37
38
```



```
1
2 function updateTime()
3     local timer = ui:getComponentByName('Timer')
4     local time = utils:getSystemTime();
5     local hour = time.hour;
6     local min = time.minute;
7     local sec = time.second;
8
9     local dsec = sec / 60 * 360;
10    local dmin = (min + (sec / 60)) / 60 * 360;
11    local dhour = (hour + (min / 60)) / 12 * 360;
12    print(dhour, dmin, dsec)
13    --hour
14    local u = ui:getComponentByName('NewFrame_2')
15    local rx = u:getAttrNumberByIndex(0)
16    local ry = u:getAttrNumberByIndex(1)
17    ...
```

自定义属性Attr

控件自定义属性



按idx顺序获取控件自定义属性，比如obj.getAttrTextByIndex(0) 表示获取数据源，obj.getAttrEnumByIndex(0)表示获取自动计时枚举属性， obj.getAttrTextByIndex(1) 表示获取格式

obj:getAttrNumberByIndex(idx)

获取自定义属性，数值属性

```
1 local time = ui:getComponentByName('NUMBER')
2 local val = time:getAttrNumberByIndex(0)
3 print(val)
```

Lua | [复制代码](#)

obj:setAttrNumberByIndex(idx, num)

设置自定义属性，数值类型

```
1 local time = ui:getComponentByName('NUMBER')
2 time:setAttrNumberByIndex(0, 2)
```

Lua | [复制代码](#)

obj:getAttrTextByIndex(idx)

获取自定义属性，文本类型

obj:setAttrTextByIndex(idx, str)

设置自定义属性，文本类型

obj:getAttrEnumByIndex(idx)

获取自定义属性，枚举类型

obj:setAttrEnumByIndex(idx, num)

设置自定义属性，枚举类型

坐标Pos

坐标

obj:setX(x)

设置控件X坐标

obj:setY(y)

设置控件Y坐标

obj:setWidth(w)

设置控件宽度

obj:setHeight(h)

设置控件高度

obj:getRect();

获取控件的位置信息

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  local rect = obj:getRect()
3  util:print(rect)
```

Lua | [复制代码](#)

```
1  {'height'=75, 'width'=75, 'x'=120, 'y'=75}
```

obj:getX()

返回控件X坐标

obj:getY()

返回控件Y坐标

obj:getWidth()

返回控件长度

obj.getHeight()

返回控件高度

obj.getPos()

获取控件相对于父控件的相对坐标

obj.getGlobalPos()

获取控件相对于工程页面左上角的绝对坐标

画图Draw

画图

obj:clearDraw()

删除通过drawLine,drawRect,drawCircle,drawPoint 这种方式画的图形

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:clearDraw()
```

obj:drawLine(object)

使用画板画直线

object对象里面为可选参数

x1,y1 表示起始坐标

x2,y2 表示结束坐标

size 表示线条粗细

color 表示线条颜色

Lua | [复制代码](#)

```
1  local obj = ui:getComponentByName('BTN')
2  obj:drawLine({x1=1, y1=1, x2=100, y2=100, size=10, color='red'})
```

obj:drawRect(object)

使用画板画长方形

object对象里面为可选参数

x, y 表示左上角坐标

w,h 表示长方形宽高

size 表示线条粗细

color 表示线条颜色

```
1 local obj = ui:getComponentByName('BTN')
2 obj:drawRect({x=1, y=1, w=10, h=10, size=2, color='green'})
```

obj:drawCircle(object)

使用画板画圆圈

x,y 表示圆心坐标

r 表示半径

size 表示线条粗细

color 表示线条颜色

```
1 local obj = ui:getComponentByName('BTN')
2 obj:drawCircle({x=10, y=10, r=8, size=2, color='#777777'})
```

obj:drawPoint(object)

使用画板画点

obj:drawArc(object)

以圆心(x, y)作为圆点，半径为r。画圆弧。

起始角度start，所画角度span，弧度以时钟0点为起点，顺时针增加，时钟6点时，为180度。

圆弧大小size=10

圆弧颜色color={'#ffffff', 'red'} 十六进制或颜色

画笔类型pencap={'flat', 'round'}

左边是pencap=flat, 右边是pencap=round




```

1  local u = ui:getComponentByName('BaseForm_1')
2  u:clearDraw()
3  local arc = {
4      x = 100,
5      y = 100,
6      r = 50,
7      start = -120,
8      span = 240,
9      size = 10,
10     color = 'red',
11     pencap = 'round'
12 }
13 u:drawArc(arc)
14 local arc = {
15     x = 100,
16     y = 100,
17     r = 35,
18     start = -120,
19     span = 240,
20     size = 10,
21     color = 'purple',
22     pencap = 'round'
23 }
24 u:drawArc(arc)

```

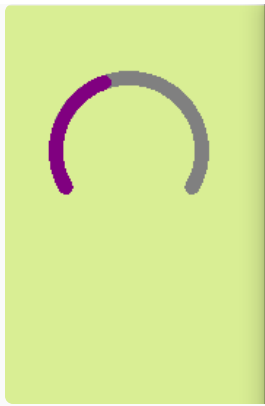


 Lua模拟器

```

1  local u = ui:getComponentByName('BaseForm_1')
2  u:clearDraw()
3  local arc = {
4      x = 100,
5      y = 100,
6      r = 50,
7      start = -120,
8      span = 240,
9      size = 10,
10     color = 'red',
11     pencap = 'round'
12 }
13 u:drawArc(arc)
14 local arc = {
15     x = 100,
16     y = 100,

```



```
Lua模拟器
1 local u = ui:getComponentByName('BaseForm_1')
2 u:clearDraw()
3 local arc = {
4   x = 100, y = 100, r = 50, start = -120, span = 240, size = 10, color = 'gray', pencap = 'round'
5 }
6 u:drawArc(arc)
7 local arc = {x = 100, y = 100, r = 50, start = -120, span = 100, size = 10, color =
  'purple', pencap = 'round'
8 }
9 u:drawArc(arc)
```

事件Event

事件

事件绑定，除了在UI设计阶段，通过ui-tools编辑工具在开发阶段进行配置外，还可以通过代码动态设置绑定事件。

事件清单

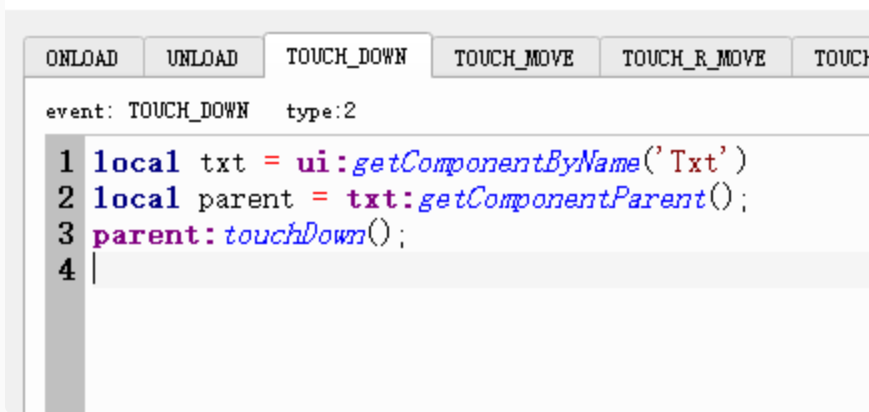
序号	EVENT	事件名称	说明
1	TOUCH_DOWN	TouchDown	按下事件
2	TOUCH_UP	TouchUp	弹起事件
3	TOUCH_MOVE	TouchMove	滑动事件
4	TOUCH_U_MOVE	TouchUMove	向上滑动事件
5	TOUCH_D_MOVE	TouchDMove	向下滑动事件
6	TOUCH_L_MOVE	TouchLMove	向左滑动事件
7	TOUCH_R_MOVE	TouchRMove	向右滑动事件
8	TOUCH_HOLD	TouchHold	长按触发事件
9			
10			

事件捕获，事件冒泡

layer->layout->control 事件一层一层向上递归，称之为事件冒泡

control->layout->layer 事件一层一层向下递归，称之为事件捕获

通常在某个控件control，触发touch事件后，需要通知它的上层父控件layout，并触发layout的点击事件。可以在代码里面，获取控件的父控件，然后显式（主动）调用父控件的touch事件
举例



Lua | 复制代码

```
1 local txt = ui:getComponentByName('Txt')
2 local parent = txt:getComponentParent();
3 parent:touchDown(); --主动触发父控件touchdown事件
```

事件传递规则

改成用一个接口控制是否接管touch事件。小机上逻辑是这样的：touch消息从最上层控件开始往下传递，如果没有lua代码就默认向下传递给父控件，如果有lua代码被执行就默认把消息接管，不再向下传递；如果有lua代码执行还想要把消息传递给它的父控件，就调用 `ui:touchEventTakeOverOn();`，这样执行完lua代码后touch消息会继续往下传给父控件响应。如果不想让父控件响应就子控件添加lua代码 `ui:touchEventTakeOverOff();`

参考ui库

obj:bindTouchDown(function)

动态绑定touchdown事件

obj:touchDown()

代码层面主动触发touchdown事件

Lua | 复制代码

```
1 local obj = ui:getComponentByName('BTN')
2 obj:bindTouchDown(
3     function()
4         utils:print('touch down event')
5     end
6 )
7 --主动触发
8 obj:touchDown()
```

obj:bindTouchUp(function)

动态绑定touchup事件

obj:touchUp()

代码层面主动触发touchup事件

obj:bindTouchUMove(function)

动态绑定touch_u_move事件

obj:touchUMove()

代码层面主动触发touch_u_move事件

obj:bindTouchDMove(function)

动态绑定touch_d_move事件

obj:touchDMove()

代码层面主动触发touch_d_move事件

obj:bindTouchLMove(function)

动态绑定touch_l_move事件

obj:touchLMove()

代码层面主动触发touch_l_move事件

obj:bindTouchRMove(function)

动态绑定touch_r_move事件

obj:touchRMove()

代码层面主动触发touch_r_move事件

obj:bindTouchMove(function)

动态绑定touch_move事件

obj:touchMove()

代码层主动触发touch_move事件

obj:bindTouchHold(function)

动态绑定touch_hold事件

obj:touchHold()

代码层主动触发 touch_hold 事件

obj:bindMovable(func)

绑定控件设置movable时，每次滑动时触发事件，下面以滑动条为例。

```

1  local slider = ui:getComponentByName('MUSIC_VOICE_SLIDER');
2  local list = slider:getSubControls()
3  local left_pic = nil;
4  local right_pic = nil;
5  local slider_pic = nil;
6  local nums = #list;
7  for i=1, nums do
8      if list[i]:getControlType() == 'right_pic' then
9          right_pic = list[i];
10     end
11     if list[i]:getControlType() == 'left_pic' then
12         left_pic = list[i];
13     end
14     if list[i]:getControlType() == 'slider_pic' then
15         slider_pic = list[i];
16     end
17 end
18
19 if left_pic == nil or right_pic == nil or slider_pic == nil then
20     print('error');
21     return ;
22 end
23
24 slider_pic:setMovableRange(true);
25 left_pic:setX(-left_pic:getWidth())
26 slider_pic:bindMovable(function(x, y)
27     local left_x = (-left_pic:getWidth()) + x;
28     local right_x = x;
29     left_pic:setX(left_x);
30     right_pic:setX(right_x);
31     local val = slider_pic:getX() / (slider:getWidth() -
slider_pic:getWidth()) * 100;
32     print(val .. '%')
33 end)

```

obj:movable()

代码层主动触发 movable 事件

obj:setInterval obj:clearInterval 这两个函数弃用，改用utils:createTimer实现

特效

满天星



```
1  local moving = ui:getComponentByName('MOVING_1')
2  moving:setMovable(true)
3  local layer = moving:getComponentParent(); --父控件图层
4  local dw = layer:getWidth() / 2;
5  local dh = layer:getHeight() / 2;
6  local dl = (dw + dh) / 2; --计算最远距离
7  local ctrls = moving:getSubControls();
8  local count = #ctrls;
9
10 moving:bindMovable(function(x, y)
11     for i=1, count do
12         local ctrl = ctrls[i];
13         --计算控件的中心点坐标
14         cx = ctrl:getX() + ctrl:getWidth() / 2;
15         cy = ctrl:getY() + ctrl:getHeight() / 2;
16         --控件相对于layer的距离
17         cx = cx + x;
18         cy = cy + y;
19         dst = math.sqrt(((cx-dw) * (cx-dw)) + ((cy-dh) * (cy-dh)))
20         local bl = 1 - (dst / dl); --缩放比例
21         if bl <= 0.0001 then
22             bl = 0.1
23         end
24         if bl >= 1.0 then
25             bl = 1;
26         end
27         ctrl:setBackgroundScale(bl)
28     end
29 end)
```

字幕滚动效果



模拟器内部会自动计算。具体滚动时延和滚动速度，通过代码实现。

Lua [复制代码](#)

```
1  local u = ui:getComponentByName("MUSIC_NAME_TEXT")
2  local offset = 0;
3  utils:createTimer(100, function()
4      offset = offset - 10;
5      u:setTextOffset(offset);
6  end);
```

✖时间日期DateTime

请使用 Utils 工具包实现

时间日期

obj:getSystemDateTime

获取系统日期时间，返回一个table

Lua [🔗复制代码](#)

```
1 local time = ui:getComponentByName('Pic')
2 local dt = time:getSystemDateTime()
3 print(dt)
4 utils:print(dt)
5 --{day=2,hour=17,minute=14,month=8,second=18,year=2021}
```

obj:getSystemDate

获取系统日期，返回一个table

Lua [🔗复制代码](#)

```
1 local time = ui:getComponentByName('Pic')
2 local dt = time:getSystemDate()
3 utils:print(dt)
4 --{day=2,month=8,year=2021}
```

obj:getSystemTime

获取系统时间，返回一个table

Lua [🔗复制代码](#)

```
1 local time = ui:getComponentByName('Pic')
2 local dt = time:getSystemTime()
3 utils:print(dt)
4 --{hour=17,minute=19,second=11}
```

obj:setSystemDateTime(datetime)

设置系统日期时间

Lua | [复制代码](#)

```
1  local time = ui:getComponentByName('Pic')
2  local dt =
    time:setSystemDateTime({day=2, hour=17, minute=14, month=8, second=18, year=2021})
```

obj:setSystemDate(date)

设置系统日期

Lua | [复制代码](#)

```
1  local time = ui:getComponentByName('Pic')
2  local dt = time:setSystemDate({day=2, month=8, year=2021})
```

obj:setSystemTime(time)

设置系统时间

Lua | [复制代码](#)

```
1  local time = ui:getComponentByName('Pic')
2  local dt = time:setSystemTime({hour=17, minute=19, second=11})
```

obj:setTimeCountDown(count, func, args)

创建一个倒计时任务，并执行

count: 倒计时秒数

func: 倒计时时间到时，进行回调

obj:clearTimeCountDown

清除当前正在运行的倒计时

obj:getTimeCountDownNumber

获取当前运行时倒计时数值

```
1  local time = ui:getComponentByName('Pic')
2  local args = {a='1', b='2'}
3  time:setTimeCountDown(15,
4      function(cell)
5          print("15s timeout." .. cell.a)
6      end
7  , args)
8
9  --按下按钮时暂停倒计时
10 local btn = ui:getComponentByName('Text')
11 btn:bindTouchDown(
12     function()
13         print(time:getTimeCountDownNumber()) --获取当前倒计时数值
14         print("clear Time CountDown")
15         time:clearTimeCountDown()
16     end
17 )
```

~~obj:setTimeCountUp(func, args)~~

设置正计时，每秒一次回调

```
1  local pic = ui:getComponentByName('Pic')
2  print(pic)
3
4  local args = {a=1, b=2}
5  function callback(cell)
6      utils:print(cell.a);
7  end
8
9  pic:setTimeCountUp(callback, args);
```

~~obj:clearTimeCountUp~~

删除正计时

~~obj:getTimeCountUpNumber~~

获取当前正计时数值


```
1  local time = ui:getComponentByName('Pic')
2  time:setTimeCountUp(
3      function()
4          local val = time:getTimeCountUpNumber();
5          print("curr Number:" .. tostring(val))
6          if (val == 5) then
7              time:clearTimeCountUp()
8          end
9      end
10 )
```

滑动容器Scroll控件

滑动容器Scroll控件

容器滑动控件，包含垂直列表控件VList，水平列表控件HList和表格控件Grid三种

常规

常规

```
1  local scroll = ui:getScrollListByName('LIST_VIEW_ID')
```

Lua | [复制代码](#)

scroll:show

设置容器显示

scroll:hide

设置容器隐藏

scroll:getControlEname

获取容器Ename

scroll:getControlType

获取容器类型 垂直列表，水平列表，表格控件

子控件

子控件（SubControls）

scroll:getSubControls

获取子控件列表，返回值是数组

Lua | [复制代码](#)

```
1  local grid = ui:getScrollListByName('BaseForm_220')
2  local list = grid:getSubControls()
3  local len = #list --获取子控件数量
4
5  for i=1, len do
6      print(list[i]:getControlName())
7      if (i % 2 == 0) then
8          print(i)
9          list[i]:hide()
10     end
11 end
```

scroll:getSubControlsNumber

获取子控件数量

Lua | [复制代码](#)

```
1  local grid = ui:getScrollListByName('BaseForm_220')
2  local list = grid:getSubControls()
3  local len = grid:getSubControlsNumber() --获取子控件数量
4
5  for i=1, len do
6      print(list[i]:getControlName())
7      if (i % 2 == 0) then
8          print(i)
9          list[i]:hide()
10     end
11 end
```

scroll:getSubControlItem(idx)

根据下标获取子控件

```
1 local grid = ui:getScrollListByName('BaseForm_220')
2 local obj = grid:getSubControlItem(0)
3 print(obj:getControlName())
4 obj:setCss(1)
```

scroll:copySubControlItemByIndex(idx)

根据下标从idx位置复制item 到最后面位置

注意 通过copy方式插入的控件将无法通过ename获取控件对象，但是可以通过

`scroll:getSubControlItem(idx)` 下标方式获取对应的控件对象。控件内部的子控件，也可以通过

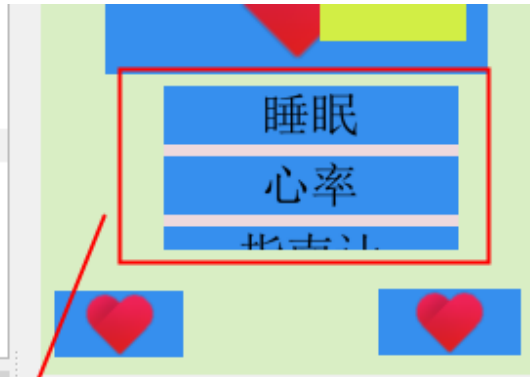
`obj:getSubControls` 获取子控件列表

```
1 local scroll = ui:getScrollListByName('BaseForm_11') --获取滚动控件
2 print(scroll)
3 scroll:copySubControlItemByIndex(1) --复制下标为1的Item到最后面（下标从0开始）
4 local obj = scroll:getSubControlItem(1) --获取下标为1的子控件（下标从0开始）
5 print(obj)
6 local list = obj:getSubControls() --获取下标为1的子控件下的所有子控件（下标从0开始）
7 print(list)
8 obj = list[1] --由于Lua下标从1开始，所有list[1]就是获取对应的文本控件（下标从1开始）
9 print(obj)
10 obj:showTextFontByIndex(1)
```

t_Watch1.json
T_Watch_01.json
rj_2020-12-10.json
rj_2021-01-29.json
rj_2021-03-31.json
atch.json

程结构树

	标题
▲ 按钮(BaseForm_3)	按钮
▲ 按钮(BaseForm_4)	按钮
▲ 图片(BaseForm_2)	图片
▼ 垂直列表(BaseForm_11)	垂直列表
▼ 布局(BaseForm_12)	布局
▲ 文字(BaseForm_15)	文字
▼ 布局(BaseForm_13)	布局
▲ 文字(BaseForm_16)	文字
▼ 布局(BaseForm_14)	布局
▲ 文字(BaseForm_17)	文字



print(scroll) 就是打印BaseForm_11控件

print(obj) 就是打印BaseForm_13控件

print(list[1]) 就是打印BaseForm_16控件

由于BaseForm_16控件是文本控件，因此可以使用 `obj.showTextFontByIndex(1)` 进行多国语言文本切换。

通过 `obj.setCss(1)` 进行设置是否高亮

scroll:removeSubControlItemByIndex(idx)

根据下标删除指定位置的item

Lua | [复制代码](#)

```
1 local scroll = ui:getScrollListByName('BaseForm_11')
2 print(scroll)
3 scroll:copySubControlItemByIndex(1)
4 scroll:copySubControlItemByIndex(1)
5 scroll:removeSubControlItemByIndex(1);
```

子项Item

子项 (Item)

scroll:setVisibleItemCount()

获取当前列表控件可显Item个数

scroll:setVisibleItemCount(cnt)

设置当前列表控件（水平列表、垂直列表）可显Item个数。当个数小于列表控件大小时，列表子项Item不可活动。当子项个数超出控件列表大小时，列表子项可上下（左右）活动。

```
1  local b = ui:getComponentByName('BT_PLAY_EARPHONE_LAYOUT')
2  b:hide();
3  --b:show()
4  local b = ui:getContainerByName('BT_SETTING_MENU')
5  b:setVisibleItemCount(3);
```

Lua | [复制代码](#)

scroll:getFirstScrollArealtemID()

获取当前滑动列表控件第一个显示的Item ID号。如果是 [垂直列表](#) 控件，那么返回最上面的Item ID号，下标从0开始。如果是 [水平列表](#) 控件，那么返回最左边的Item ID号。

scroll:getMiddleScrollArealtemID()

获取当前滑动列表控件，中间显示的Item ID号

scroll:getLastScrollArealtemID()

获取当前滑动列表控件，最后一个显示的Item ID号。垂直列表控件，返回最下面的Item ID号，水平列表控件，返回最右边的Item ID号

scroll:setFirstScrollArealtemID(idx)

设置当前滑动列表控件第一个显示的Item ID号。跳转到指定Item。下标从0开始。

scroll:setMiddleScrollArealtemID(idx)

设置当前滑动列表控件中间显示的Item ID号。

scroll:setLastScrollArealtemID(idx)

设置当前滑动列表控件最后一个显示的Item ID号。

scroll:getVerticalScrollSliderPos()

获取列表控件、表格控件垂直方向上已经滑动的像素点。（水平列表，该返回值始终为0）

scroll:getHorizontalScrollSliderPos

获取列表控件、表格控件水平方向上已经滑动的像素点。（垂直列表，该返回值始终为0）

scroll:setVerticalScrollSliderPos

设置列表控件、表格控件垂直方向上滑动的像素点。

scroll:setHorizontalScrollSliderPos

设置列表控件、表格控件水平方向上滑动的像素点。

Lua [复制代码](#)

```
1 local u = ui:getContainerByName('VERTLIST')
2 print(u:getVisibleItemCount())
3 print(u:getFirstScrollAreaItemID())
4 print(u:getMiddleScrollAreaItemID())
5 print(u:getLastScrollAreaItemID())
6 print(u:getVerticalScrollSliderPos())
7 print(u:getHorizontalScrollSliderPos())
8
9 u:setFirstScrollAreaItemID(1)
```



事件

事件

事件绑定，除了在UI设计阶段，通过ui-tools编辑工具在开发阶段进行配置外，还可以通过代码动态设置绑定事件。

事件清单

序号	EVENT	事件名称	说明
1	TOUCH_DOWN	TouchDown	按下事件
2	TOUCH_UP	TouchUp	弹起事件
3	TOUCH_HOLD	TouchHold	长按触发事件
4			

事件捕获，事件冒泡

layer->layout->control 事件一层一层向上递归，称之为事件冒泡

control->layout->layer 事件一层一层向下递归，称之为事件捕获

通常在某个控件control，触发touch事件后，需要通知它的上层父控件layout，并触发layout的点击事件。可以在代码里面，获取控件的父控件，然后显式（主动）调用父控件的touch事件
举例

脚本编辑器

ONLOAD UNLOAD TOUCH_DOWN TOUCH_MOVE TOUCH_R_MOVE TOUCH

event: TOUCH_DOWN type:2

1 local txt = ui:getComponentByName('Txt')
2 local parent = txt:getComponentParent();
3 parent:touchDown();
4 |

```

1  local txt = ui:getComponentByName('Txt')
2  local parent = txt:getComponentParent();
3  parent:touchDown();--主动触发父控件touchdown事件

```

事件传递规则

改成用一个接口控制是否接管touch事件。小机上逻辑是这样的：touch消息从最上层控件开始往下传递，如果没有lua代码就默认向下传递给父控件，如果有lua代码被执行就默认把消息接管，不再向下传递；如果有lua代码执行还想要把消息传递给它的父控件，就调用 `ui:touchEventTakeOverOn();`，这样执行完lua代码后touch消息会继续往下传给父控件响应。如果不想让父控件响应就子控件添加lua代码 `ui:touchEventTakeOverOff();`

[参考ui库](#)

obj:bindTouchDown(function)

动态绑定touchdown事件

obj:touchDown()

代码层面主动触发touchdown事件

```

1  local obj = ui:getComponentByName('BTN')
2  obj:bindTouchDown(
3      function()
4          utils:print('touch down event')
5      end
6  )
7  --主动触发
8  obj:touchDown()

```

obj:bindTouchUp(function)

动态绑定touchup事件

obj:touchUp()

代码层面主动触发touchup事件

obj:bindTouchHold(function)

动态绑定touch_hold事件

obj:touchHold()

代码层主动触发 touch_hold 事件

obj:bindTouchScroll(function)

动态绑定滑动事件，当控件（水平、垂直、列表）出现滚动时，触发改事件。

参数x为0时，表示当前控件是 垂直列表

参数y为0时，表示当前控件是 水平列表

Lua | [复制代码](#)

```
1  local u = ui:getContainerByName('BaseForm_1')
2  u:bindTouchScroll(function(x, y)
3      print('scrollEvent x:' .. x .. ',y:' .. y);
4  end)
```

特效

圆弧效果

scroll:setScrollListRotateSize(size)

当垂直列表，size大于零时，向右弯曲

```
1 local u = ui:getContainerByName('VERTLIST')  
2 u:setScrollListRotateSize(30)
```



当垂直列表，size小于零时，向左弯曲

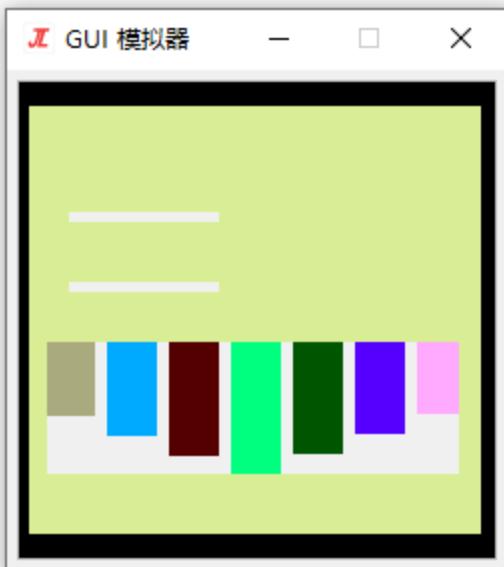


当水平列表，size大于零时，向下弯曲



当水平列表，size小于零时，向上弯曲

```
1 local u = ui:getContainerByName('BaseForm_13')  
2 u:setScrollListRotateSize(-10)
```



Lua | [复制代码](#)

```
1 local u = ui:getContainerByName('VERTLIST')  
2 u:setScrollListRotateSize(-20)  
3 u:bindTouchScroll(function(x, y)  
4     print('x:' .. x .. ' y:' .. y)  
5 end)
```

```

1 local u = ui:getContainerByName('VERTLIST')
2 u:setScrollListRotateSize(-20)
3 u:bindTouchScroll(function(x, y)
4     print('x:' .. x .. ' y:' .. y)
5 end)

```



运行

```

x:0 y:223
x:0 y:224
x:0 y:225
x:0 y:226

```

工具类模块 utils

Utils模块

一些通用模块

定时器

utils:createTimer(ms, func, args)

创建一个定时器任务

Lua | [复制代码](#)

```
1 local func = function(cell) print(cell.a) end
2 local args = {a=1, b=2}
3 local idx = utils:createTimer(1000, func, args)
```

utils:clearTimer(idx)

结束一个定时器任务

utils:getTimerIds

获取当前运行中的定时器ID号列表（数组）

Lua | [复制代码](#)

```
1 local idx = utils:createTimer(1000, function()
2     print('test')
3 end, {})
4 print(idx)
5 local idx2 = utils:createTimer(1000, function()
6     local ids = utils:getTimerIds() --获取定时器ID列表
7     print('timer: ')
8     utils:print(ids)
9 end, {})
10 print(idx2)
11 --
12 utils:clearTimer(idx2)
```

utils:clearAllTimer()

删除所有定时器，关闭模拟器时，会自动调用该函数，停止所有定时器任务。

时间日期

utils:getSystemDate

获取系统日期，返回一个table

```
1 local dt = utils:getSystemDate()
2 utils:print(dt)
3 --{day=2,month=8,year=2021}
```

Lua [🔗复制代码](#)

utils:getSystemTime

获取系统时间，返回一个table

```
1 local dt = utils:getSystemTime()
2 utils:print(dt)
3 --{hour=17,minute=19,second=11}
```

Lua [🔗复制代码](#)

utils:setSystemDate(date)

设置系统日期

```
1 local dt = time:setSystemDate({day=2,month=8,year=2021})
```

Lua [🔗复制代码](#)

utils:setSystemTime(time)

设置系统时间

```
1 local dt = time:setSystemTime({hour=17,minute=19,second=11})
```

Lua [🔗复制代码](#)

转换

utils:tranHexAddrToControlID()

通过十六进制，即ename.h 里面的宏定义，转换成控件ID属性。分离出项目ID、页面ID、类型ID、控件ID

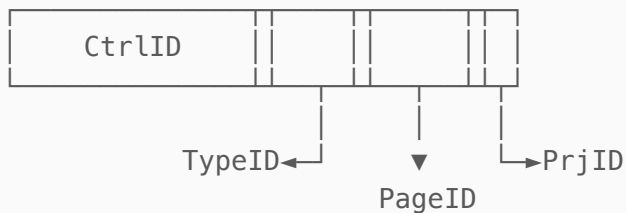
Lua [复制代码](#)

```
1 local mod = utils:tranHexAddrToControlID(0x282000A)
2 utils:print(mod)
3 print("value : " .. mod.value)
4 print(" hex   : " .. mod.hex)
5 print("projid: " .. mod.projid)
6 print("pageid: " .. mod.pageid)
7 print("typeid: " .. mod.typeid)
8 print("ctrlid: " .. mod.ctrlid)
9
```

Lua | [复制代码](#)

```
1 [31~29]:prj 3bit(0~7)
2 [28~22]:page 7bit(0~127)
3 [21~16]:type 6bit(0~63)
4 [15~0]: id
```

00000000001111111122222222233
01234567890123456789012345678901



其他

utils:exit()

退出模拟器

Lua [复制代码](#)

```
1  utils::exit();
```

utils:log(info, color)

打印带颜色调试信息，打印信息到控制台

color: 取值可以是颜色英文单词，也可以是十六进制颜色值

Plain Text | [复制代码](#)

```
1  utils:log('string', 'red')
2  utils:log('string', '#f0f')
```

utils:print(info)

增强型打印调试信息，打印信息到控制台

print 打印数字，文本，部分table类型

Lua | [复制代码](#)

```
1  utils:print('string')
```

utils:version()

打印当前lua脚本版本信息

Lua | [复制代码](#)

```
1  utils:version()
```

utils:help()

打印帮助信息

Lua | [复制代码](#)

```
1  utils:help()
```

滑动页面模块 slide

slide:initPage()

初始化Slide控件，并将当前页面ID加入到Slide队列中

slide:freePage()

释放Slide控件资源，Slide控件变为未初始化阶段，无法进行插入和新增操作

slide:insertNextPageByName(ename)

在当前页面ID后面插入新页面ID

slide:insertNextPageByID(id)

在当前页面ID后面插入新页面ID

slide:insertPrevPageByName(ename)

在当前页面ID前面插入新页面ID

slide:insertPrevPageByID(id)

在当前页面ID前面插入新页面ID

slide:deleteNextPage()

在当前页面ID，删除后面页面ID

slide:deletePrevPage()

在当前页面ID，删除前面页面ID

slide:deletePageByName(ename)

根据指定页面PAGE_ID，删除slide队列里面对应的PAGE_ID的页面

slide:deletePageByID(id)

根据指定id，删除slide队列里面对应id的页面

slide:getListPageName()

获取整个Slide队列中页面ID名称

slide:getListPageID()

获取整个Slide队列中页面ID

slide:getCurrIndex()

获取当前页面在Slide队列中的位置

slide:getCurrPageName()

获取当前页面ID名称

slide:getCurrPageID()

获取当前页面ID

slide:switchNextPage(type)

主动切换页面，在slide队列中循环切换，type切换方式，

slide:switchPrevPage(type)

主动切换页面，在Slide队列中循环切换，type切换方式

```
1  slide:initPage()
2  slide:insertNextPageByName('PAGE_1')
3  slide:insertPrevPageByName('PAGE_1')
4  slide:insertNextPageByID(0)
5  slide:insertPrevPageByID(0)
6  slide:deleteNextPage()
7  slide:deletePrevPage()
8  slide:deletePageByName('PAGE_1')
9  slide:deletePageByID(1)
10 slide:insertPrevPageByName('PAGE_1')
11 utils:print(slide:getListPageName())
12 utils:print(slide:getListPageID())
13 print(slide:getCurrIndex())
14 print(slide:getCurrPageName())
15 print(slide:getCurrPageID())
16 slide:switchNextPage('left')
17 utils:sleep(1500)
18 slide:switchNextPage('left')
19 utils:sleep(1500)
20 slide:switchNextPage('left')
21 utils:sleep(1500)
22 slide:switchPrevPage('right')
23 utils:sleep(1500)
24 slide:switchPrevPage('right')
25 utils:sleep(1500)
26 slide:switchPrevPage('right')
```

```

1  init lua          --初始化lua模拟器
2  (0) 0             --表示Slide中(0), 当前Slide下标0
3  (0, 1) 0          --表示Slide中(0, 1), 当前Slide下标0
4  (1, 0, 1) 1       --表示Slide中(1, 0, 1), 当前Slide下标1
5  (1, 0, 0, 1) 1    --表示Slide中(1, 0, 0, 1), 当前Slide下标1
6  (1, 0, 0, 0, 1) 2 --表示Slide中(1, 0, 0, 0, 1), 当前Slide下标2
7  (1, 0, 0, 1) 2    --表示Slide中(1, 0, 0, 1), 当前Slide下标2
8  (1, 0, 1) 1       --表示Slide中(1, 0, 1), 当前Slide下标1
9  (0) 0             --表示Slide中(0), 当前Slide下标0
10 (1, 0) 1          --表示Slide中(1, 0), 当前Slide下标1
11 类型为: table, 内容为: {1='PAGE_1',2='PAGE_0'}
12 类型为: table, 内容为: {1=1,2=0}
13 1
14 PAGE_0
15 0
16 左滑
17 某页面执行Unload 0
18 某页面执行Onload 1
19 (1, 0) 0
20 左滑
21 某页面执行Unload 1
22 某页面执行Onload 0
23 (1, 0) 1
24 左滑
25 某页面执行Unload 0
26 某页面执行Onload 1
27 (1, 0) 0
28 右滑
29 某页面执行Unload 1
30 某页面执行Onload 0
31 (1, 0) 1
32 右滑
33 某页面执行Unload 0
34 某页面执行Onload 1
35 (1, 0) 0
36 右滑
37 某页面执行Unload 1
38 某页面执行Onload 0
39 (1, 0) 1

```

系统接口 sim_system

这里增加一些系统运行中的接口，一般只限于模拟器模拟器时使用，设备那边有特定的接口实现

sim_system:getWorkspacePath();

获取当前运行的工作目录，绝对路径。

这里有两种模式，如果是单工程模式，会返回单工程的工作目录

UI工程\ui_240x240_watch\表盘5\project

如果是多工程模式，会返回多工程的工作目录

UI工程\ui_240x240_watch\

从上面可以看出两者的不同。

sim_system:getWorkingMode()

获取当前运行模型是单工程模式，还是多工程模式

'single' 'multiple'

sim_system:getWatchConfig()

获取手表项目特有的配置信息

Lua | [复制代码](#)

```
1  {
2      'project_list' = {
3          'dial_interface' = {
4              1='/表盘1/project/dial.json',
5              2='/表盘5/project/dial.json',
6              3='/表盘界面/project/dial.json'
7          },
8          'dial_item' = 1,
9          'mode_interface' = '/模式界面/project/BT_Watch.json',
10         'sidebar_interface' = '/侧边栏界面/project/BT_Watch.json',
11         'update_interface' = '/升级界面/project/dial.json'
12     },
13     'project_type'='watch',
14     'version'=1,
15     'workspace_path'='C:/ui_240x240_watch'
16 }
```

```
1 print(sim_system:getWorkspacePath())
2 print(sim_system:getWorkingMode())
3 print(sim_system:getWatchConfig())
```

系统事件

sim_system:setSystemEventList(table)

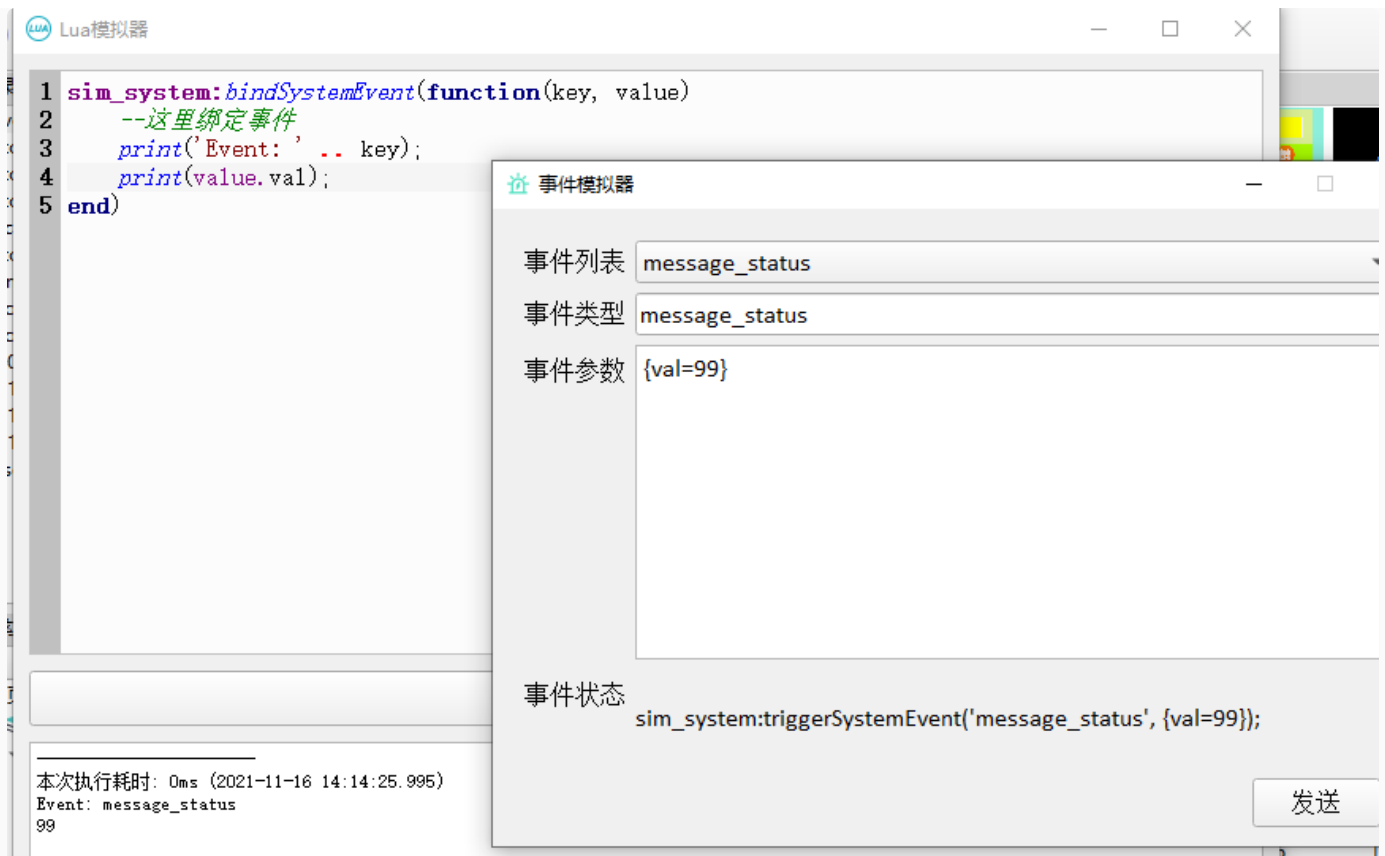
设置系统事件列表

sim_system:getSystemEventList()

获取系统事件列表

```
1 sim_system:setSystemEventList({
2     'test',
3     'bbc',
4     'deasdfasdf',
5     '中文'
6 })
7 utils:print(sim_system:getSystemEventList())
```

注意 这个可以在init初始化的时候进行设置，这样在调用“事件模拟器”窗口是会有对应的事件列表清单。



`sim_system:bindSystemEvent(function(key, value))`

绑定系统事件

Lua | [复制代码](#)

```
1  sim_system:bindSystemEvent(function(key, value)
2      print('收到事件' .. key .. ' 内容:' .. value)
3      if key == 'SYSTEM_TOUCH' then
4          print('触发SYSTEM_TOUCH事件')
5      end
6  end)
```

`sim_system:triggerSystemEvent(key, value)`

工具层触发事件

Lua | [复制代码](#)

```
1  sim_system:triggerSystemEvent('SYSTEM_TOUCH', {x=1, y=2})
```

```
1  --绑定事件
2  sim_system:bindSystemEvent(function(key, value)
3      --这里绑定事件
4      print('Event: ' .. key);
5      print(value.val);
6  end)
7
8  --事件触发
9
```

注意：工具的事件参数，可以传入table也可以传入字符串，如果是字符串的，需要加引号，也可以直接传数字。

事件模拟器

事件列表	message_status
事件类型	message_status
事件参数	{val=99}

事件模拟器

事件列表	message_status
事件类型	message_status
事件参数	'99'

事件模拟器

事件列表	message_status
事件类型	message_status
事件参数	99

文件模块 sim_file

文件目录

sim_file:setWorkspace(str)

设置工作目录，返回工作目录的绝对路径，后续获取文件列表，需要传入此参数

sim_file:getListFile(str)

传入绝对路径，返回当前目录下所有文件列表

sim_file:getFileHandler(str)

通过传入绝对路径，获取文件操作句柄

Lua | [复制代码](#)

```
1  --设置当前目录为工作目录
2  local ws = sim_file:setWorkspace('.')
3  print(ws)
4  local list = sim_file:getListFile(ws)
5  --utils:print(list)
6  for i=1, #list do
7      local file = list[i];
8      -- utils:print(file)
9      print('Folder:' .. tostring(file.isFolder) ..
10           '\tsize:' .. file.filesize ..
11           '\tfilename:' .. file.filename)
12  end
```

```

1  --设置当前目录为工作目录
2  local ws = sim_file:setWorkspace('.')
3  print(ws)
4  local list = sim_file:getListFile(ws)
5  --utils:print(list)
6  for i=1, #list do
7      local file = list[i];
8      -- utils:print(file)
9      print('isFolder: ' .. tostring(file.isFolder) ..
10          '\tSize: ' .. file.filesize ..
11          '\tName: ' .. file.filename)
12 end

```

运行(F5)

```

C:/Workpace/QT/build-ui-tools-simulator-unknown_e2f7a3-Debug
isFolder: false      Size: 1043   Name: .qmake.stash
isFolder: false      Size: 67934  Name: app_res.o
isFolder: false      Size: 1062171 Name: apptreedockwidget.o
isFolder: false      Size: 4859503 Name: autosave.json
isFolder: false      Size: 1657979 Name: basecomponent.o
isFolder: false      Size: 361823  Name: basescriptmodel.o
isFolder: false      Size: 4555564 Name: BT_Watch - 副本.json
isFolder: false      Size: 39257550  Name: BT_Watch.202110211606.json

```

单个文件

通过getFileHandler接口可以获取到文件句柄，然后通过具备对文件进行操作

文件属性 File Attr

属性	说明
path	文件存储路径，文件上级目录绝对路径（例：C:\Windows\Users）
filename	文件名（例：test.lua）
filepath	文件完整路径，（例：C:\Windows\Users\test.lua）
isFolder	是否目录
createAt	文件（夹）创建时间
updateAt	文件（夹）更新时间

1

Lua |  复制代码

模拟数据 sim_virtual

sim_virtual:getListAddressBook

获取虚拟通讯录

Lua [复制代码](#)

```
1  local list = sim_virtual:getListAddressBook();
2  utils:print(list)
3  for i=1, #list do
4      local log = list[i];
5      print('Name:' .. log.name .. '\tNumber:' .. log.number);
6  end
```

sim_virtual:getListCallog

获取虚拟通话记录

Lua [复制代码](#)

```
1  local list = sim_virtual:getListCallLog();
2  for i=1, #list do
3      local cell = list[i];
4      print('=====')
5      print('姓名:\t' .. cell.name)
6      print("手机号码:\t" .. cell.number);
7      print("开始时间:\t" .. os.date("%Y-%m-%d %H:%M:%S", cell.callTime))
8      print("通话时长:\t" .. cell.talkTime)
9  end
10
```

虚拟Cache存储（共享内存）

通过虚拟存储方式，达到模拟器内部页面、工程数据共享，类似全局变量方式。

通过共享内存方式，达到UI界面模拟器与外设（传感器）模拟器数据交互，类型共享内存方式。

例如，在模拟器运行过程中，通过模拟器外设（蓝牙、串口）给UI工程发送一些模拟蓝牙指令，而后，在UI界面模拟器中，可以获取到外界的数据。

sim_virtual:setCache(key, value)

`sim_virtual:getCache(key)`

`sim_virtual:clearCache(key)`

`sim_virtual:registerCacheCallBack(key, fun)`

`sim_virtual:clearCacheCallBack(key)`

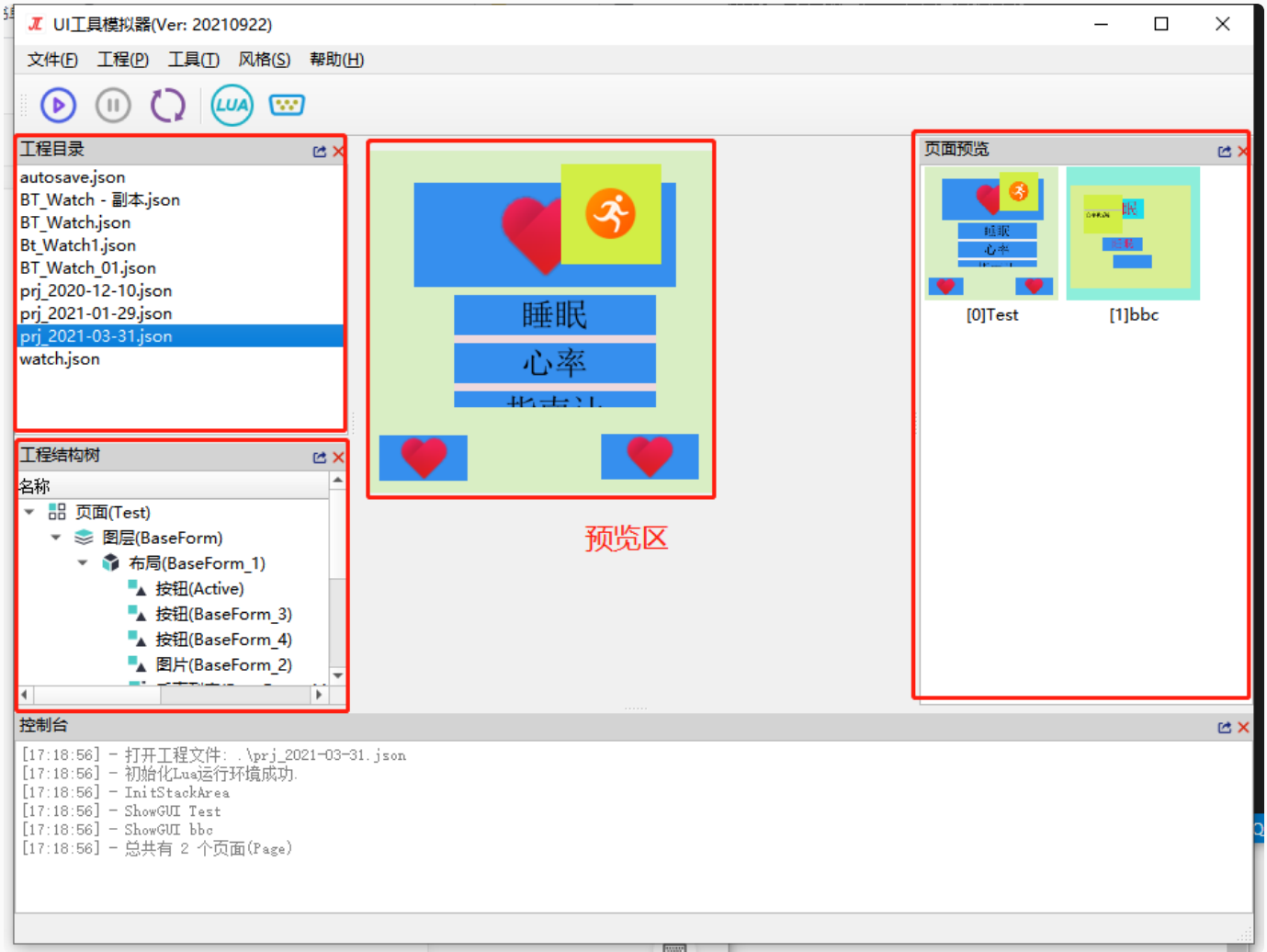
`sim_virtual:clearAllCache()`

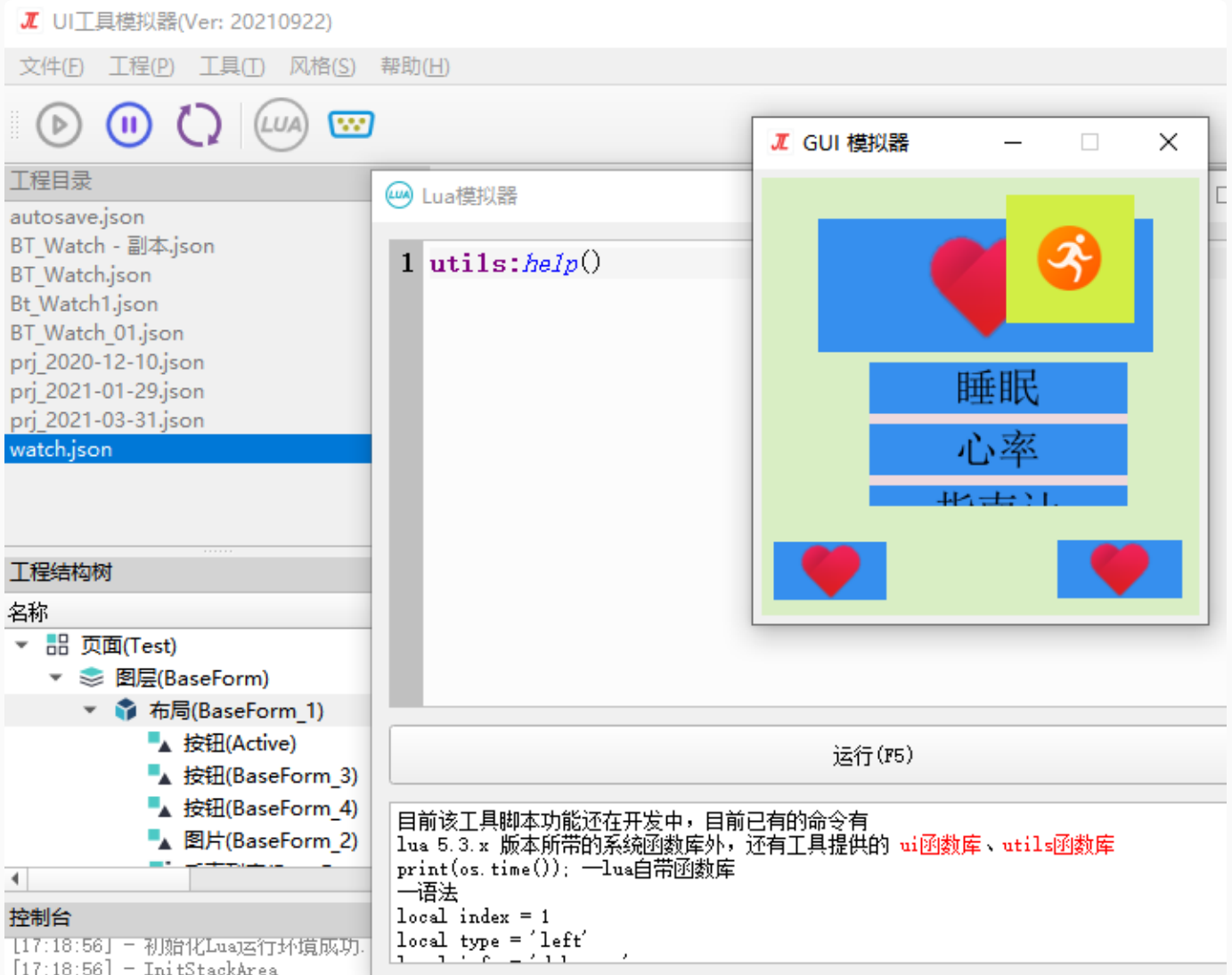
传感器模块 sim_sensor

这里提供一些传感器数据模拟

UI工具模拟器操作说明

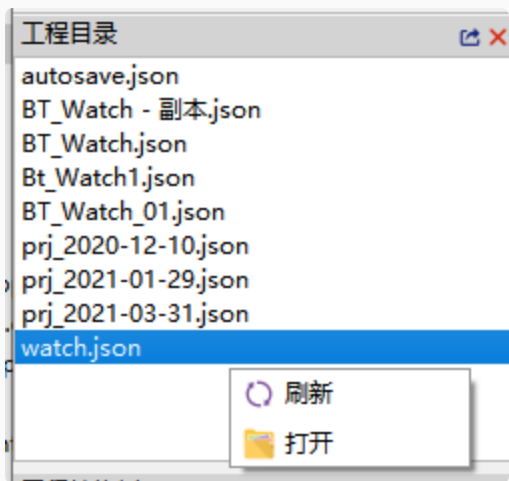
工具预览图





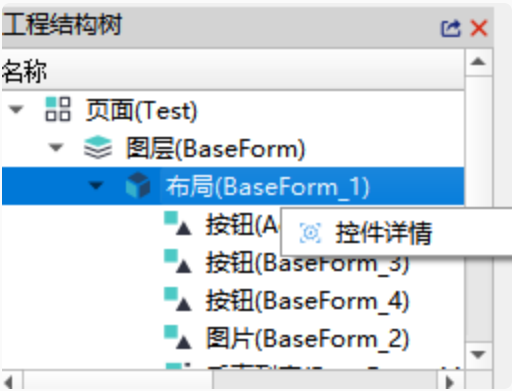
工程目录

双击打开对应工程，右键选择打开工程目录。



工程结构树

显示当前页面Page下所有的控件，及其对应父子关系。右键可以查看该控件的详细控件信息。



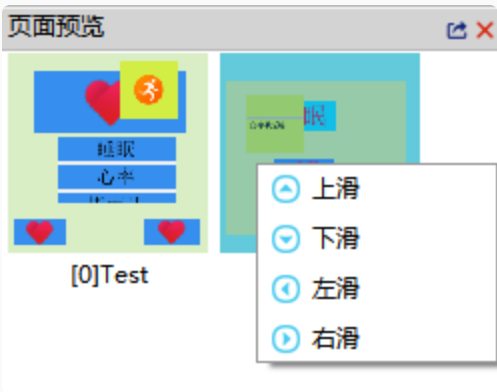
控制台

打印当前工程一些工作日志，还有运行Lua脚本时的调试信息。右键可以清空。



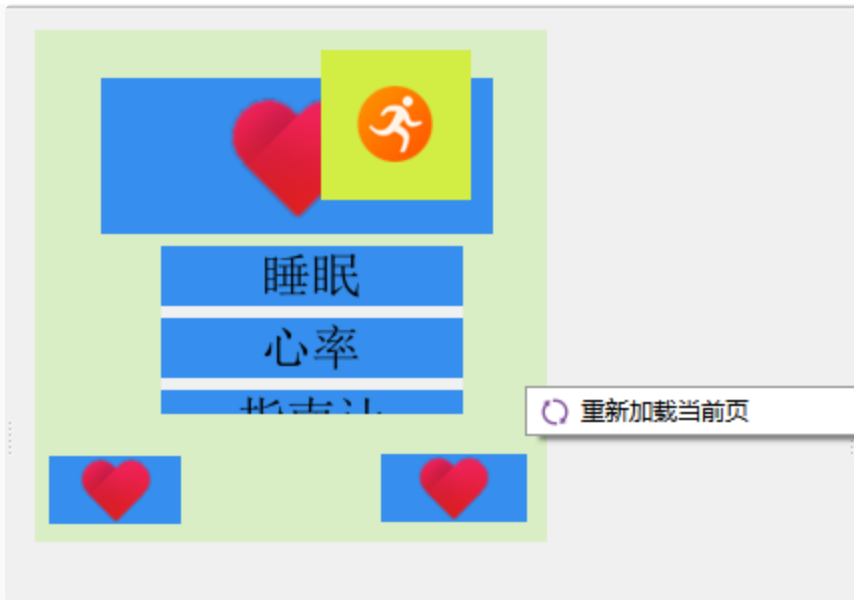
页面预览区

表示当前工程总共有多少个页面Page，及对应页面Page的预览图效果。右键可以模拟切换页面效果。双击重新加载工程中对应页面，并切换对应页面。





中间模拟效果


当前预览效果，是设备中的模拟效果。右键可以刷新重新加载当前页面。（通过 `ui-tools` 工具画图后，右键可以重新加载当前页）



菜单区

点击  运行，可以模拟设备中运行Lua脚本后的效果。

点击  停止模拟，停止执行Lua脚本

点击  可以执行简单的单行Lua语句。

