# 目录

# 第1章 初识 AWTK

✍**本章导读**

随着手机、智能手表等便携式设备的普及，用户对 GUI 的要求越来越高，嵌入式系统对 GUI 的需求也越来越迫切，本章将为大家介绍一个轻型、占用资源少、高性能、高可靠、便于移植、可配置及美观的 GUI 编程框架。

## 1.1 简介

AWTK（Toolkit AnyWhere）是一套基于 C 语言开发的 GUI 框架，目前支持的平台有嵌入式、Linux、Mac OS X、Windows，后续会支持 Android、iOS 平台下的 APP，以及 2D 游戏的开发。

## 1.2 特色

1. 小巧。在精简配置下，不依赖第三方软件包，仅需要 8K RAM+32K FLASH 即可开发一些简单的图形应用程序。
2. 高效。采用脏矩形裁剪算法，每次只绘制和更新变化的部分，极大提高运行效率。
3. 稳定。通过良好的架构设计、编程风格、单元测试、动态（valgrind）检查和 Code Review 保证其运行的稳定性。
4. 丰富的 GUI 组件。提供窗口、对话框和各种常用的组件（用户可以配置自己需要的组件，降低对运行环境的要求）。
5. 支持多种字体格式。内置位图字体（并提供转换工具），也可以使用 stb_truetype 或 freetype 加载 ttf 字体。
6. 支持多种图片格式。内置位图图片（并提供转换工具），也可以使用 stb_image 加载 png/jpg 等格式的图片。
7. 紧凑的二进制界面描述格式。可以使用手工编辑的 XML 格式的界面描述文件，也可以使用 Qt Designer 设计界面，然后转换成紧凑的二进制界面描述格式文件，提高运行效率，减小内存开销。
8. 支持主题并采用紧凑的二进制格式。开发时使用 XML 格式描述主题，然后转换成紧凑的二进制格式，提高运行效率，减小内存开销。
9. 支持裸系统，无需 OS 和文件系统。字体、图片、主题和界面描述数据都编译到代码中，以常量数据的形式存放，运行时无需加载到内存。
10. 内置 nanovg 实现高质量的矢量动画。
11. 支持窗口动画、控件动画、滑动动画和高清 LCD 等现代 GUI 常见特性。
12. 支持国际化（Unicode、字符串翻译和输入法等）。
13. 可移植。支持移植到各种 RTOS 和嵌入式 Linux 系统，并通过 SDL 在各种流行的 PC/手机系统上运行。
14. 脚本化。从 API 注释中提取 API 的描述信息，通过这些信息可以自动生成各种脚本的绑定代码。
15. 支持硬件 2D 加速（目前支持 STM32 的 DMA2D 和 NXP 的 PXP）和 GPU 加速（OpenGL/OpenGLES/DirectX/Metal），充分挖掘硬件潜能。
16. 采用 LGPL 协议开源发布，在商业软件中使用时无需付费。

## 1.3 AWTK 环境搭建

本章节我们将向大家介绍如何在本地搭建 AWTK 开发环境。

### 1.3.1 下载 AWTK

AWTK 的最新源码、文档、更新信息以及 API 手册下载地址，详见表 1.1。

表 1.1 AWTK 下载地址

| 说明 | 下载地址 |
| --- | --- |
| 源码地址 | https://github.com/zlgopen/awtk |
| 文档 | https://github.com/zlgopen/awtk/blob/master/docs/index.md |
| API 手册 | https://github.com/zlgopen/awtk/tree/master/docs/manual |
| AWorks（RT1052）适配层 | https://github.com/zlgopen/awtk-aworks-rt1052 |

### 1.3.2 编译 AWTK

下载完 AWTK 源码后就可以编译了，下面我们一起看看如何在不同平台下编译 AWTK。

#### 1. Linux

下面以 Ubuntu（版本>=16）为例，如果没有安装 scons 和依赖的软件包，请在终端运行下面的命令：

```
sudo apt-get install scons libsndio-dev libgtk-3-dev libglu1-mesa libglu1-mesa-dev libgl1-mesa-glx
libgl1-mesa-dev
```

编译运行（在终端下，进入 AWTK 所在的目录，并运行下列命令）：

```
scons
./bin/demoui
```

#### 2. Mac OS X

如果没有安装 scons 和 sdl2，请在终端运行下面的命令（假定已安装 brew）：

```
brew install scons sdl2
```

编译运行（在终端下，进入 AWTK 所在的目录，并运行下列命令）：

```
scons
./bin/demoui
```

#### 3. Windows

请先安装 python2.7、scons 和 Visual Studio C++（版本>=2015），这三个软件安装的过程都很简单这里就不一一介绍了，请按顺序安装上面三个软件。

需要注意一点的是安装 Visual Studio 时，请选择自定义安装，详见图 1.1。然后选择 Visual C++，详见图 1.2，其他按缺省步骤操作。

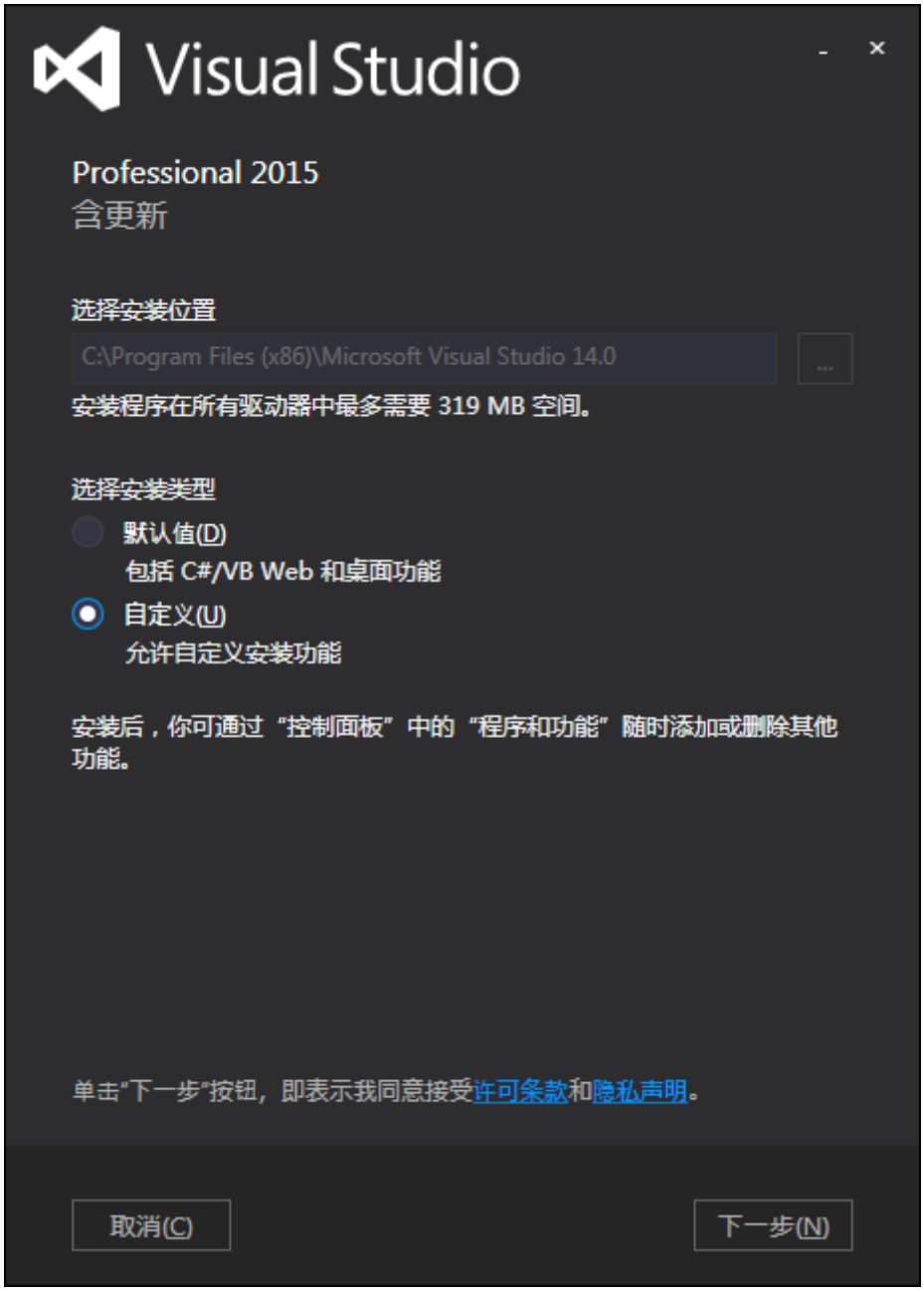

Visual Studio
Professional 2015
含更新
选择安装位置
C:\Program Files (x86)\Microsoft Visual Studio 14.0
...
安装程序在所有驱动器中最多需要 319 MB 空间。
选择安装类型
默认值(D)
包括 C#/VB Web 和桌面功能
自定义(U)
允许自定义安装功能
安装后，你可通过“控制面板”中的“程序和功能”随时添加或删除其他功能。
单击“下一步”按钮，即表示我同意接受许可条款和隐私声明。
取消(C)
下一步(N)

图 1.1 选择自定义


图 1.2 选择 Visual C++

编译运行（在终端下，进入 AWTK 所在的目录，并运行下列命令）：

```
scons
./bin/demoui
```

可选安装 Visual Studio Code（建议安装：C/C++ Intellisense 插件，该插件主要用于跳转到定义、自动提示等）用于编写 xml 文件。

## 1.4 SConstruct 文件说明

在 AWTK 目录下有个 SConstruct 文件，在这个文件中我们可以通过定义不同含义的宏实现不同的效果，接下来我们一起来看看一些常用的宏及其说明吧。

### 1.4.1 渲染方式

在 SConstruct 文件里面我们会看到 NANOVG_BACKEND 的不同取值，详见程序清单 1.1，它们所表达的含义详见表 1.2。

程序清单 1.1 渲染方式

```
# awtk\SConstruct
...
VGCANVAS='NANOVG'
NANOVG_BACKEND='GL3'
#NANOVG_BACKEND='GLES2'
#NANOVG_BACKEND='GLES3'
#NANOVG_BACKEND='AGG'
NANOVG_BACKEND='AGGE'
#NANOVG_BACKEND='BGFX'
...
```

表 1.2 NANOVG_BACKEN 含义

| 参数 | 说明 |
| --- | --- |
| GL3/GLES2/GLES3 | 基于 OpenGL/GLES 实现（nanovg 内置），由宏 WITH_NANOVG_GL 决定 |
| AGG | 基于 agg 实现。纯软件实现，渲染效果好，速度较慢，适合没有 GPU 的嵌入式平台。（注意：目前不支持 565 格式的图片，请勿定义 WITH_BITMAP_BGR565）。由宏 WITH_NANOVG_AGG 决定 |
| AGGE | 基于 agge 实现。纯软件实现，渲染效果一般，速度比 agg 快，适合没有 GPU 的嵌入式平台。由宏 WITH_NANOVG_AGGE 决定 |
| BGFX | 基于 bgfx 实现。支持多种渲染方式（OpenGL/metal/vulkan/DirectX），推荐在 Android、iOS、Linux、MacOS、Windows 等平台上使用。由宏 WITH_NANOVG_BGFX 决定 |

### 1.4.2 是否支持裸机系统

如果在 SConstruct 文件中定义了 WITH_FS_RES 宏，就无需 OS 和文件系统。AWTK 会将字体、图片、主题和界面描述数据都编译到代码中，以常量数据的形式存放，运行时无需加载到内存，代码详见程序清单 1.2。

程序清单 1.2 是否支持裸机系统

```
# awtk\SConstruct
...
COMMON_CCFLAGS=COMMON_CCFLAGS+' -DWITH_SDL -DWITH_FS_RES -DHAS_STDIO '
...
```

如果未定义 WITH_FS_RES 宏 AWTK 将会加载应用程序 demos\assets 目录下的资源。

### 1.4.3 是否支持 png/jpeg 图片

如果系统支持 png/jpeg 图片，请在 SConstruct 文件中定义宏 WITH_STB_IMAGE，AWTK 将会加载 assets/inc/images 目录下的".res"格式文件，否则加载".data"格式文件。

注意事项：上面提到的只有在没有定义宏 WITH_FS_RES 的时候才有效。

### 1.4.4 是否在嵌入式系统运行

嵌入式系统有自己的 main()函数，如果我们的应用程序是在嵌入式系统运行的话，请定义宏 USE_GUI_MAIN。

### 1.4.5 是否使用输入法

如果应用程序不需要输入法，请定义宏 WITH_NULL_IM。

### 1.4.6 是否有标准的内存分配函数

如果有标准的 malloc/free/calloc 等函数，请定义本宏 HAS_STD_MALLOC。

### 1.4.7 是否使用小字体库

在 awtk\demos\assets\raw\fonts 目录下有 default.ttf（包含全部字符）和 default.mini.ttf（只包含应用程序使用到的字符）两个文件。如果没有定义宏 WITH_MINI_FONT 则使用 default.ttf，否则使用 default.mini.ttf。

注意实现：上面提到的只有在定义了宏 WITH_STB_FONT 或 WITH_FT_FONT 的时候才会生效。

更多的宏说明请看： awtk\docs\porting_common.md

# 第2章 第一个 AWTK 应用

**✍本章导读**

在学习计算机语言的时候，我们开始的第一个程序也就是最经典的 hello world 程序。与之类似，要掌握 AWTK 的开发流程，我们也先创建自己的第一个 hello world 应用程序。

## 2.1 简介

本章的 hello world 应用很简单，一个文本框用于显示 hello world、一个编辑框用于编辑文本、两个按钮分别用于递增和递减数字，本章节所使用到的代码可以在 HelloWorld-Demo 目录下找到，应用实现流程详见图 2.1。

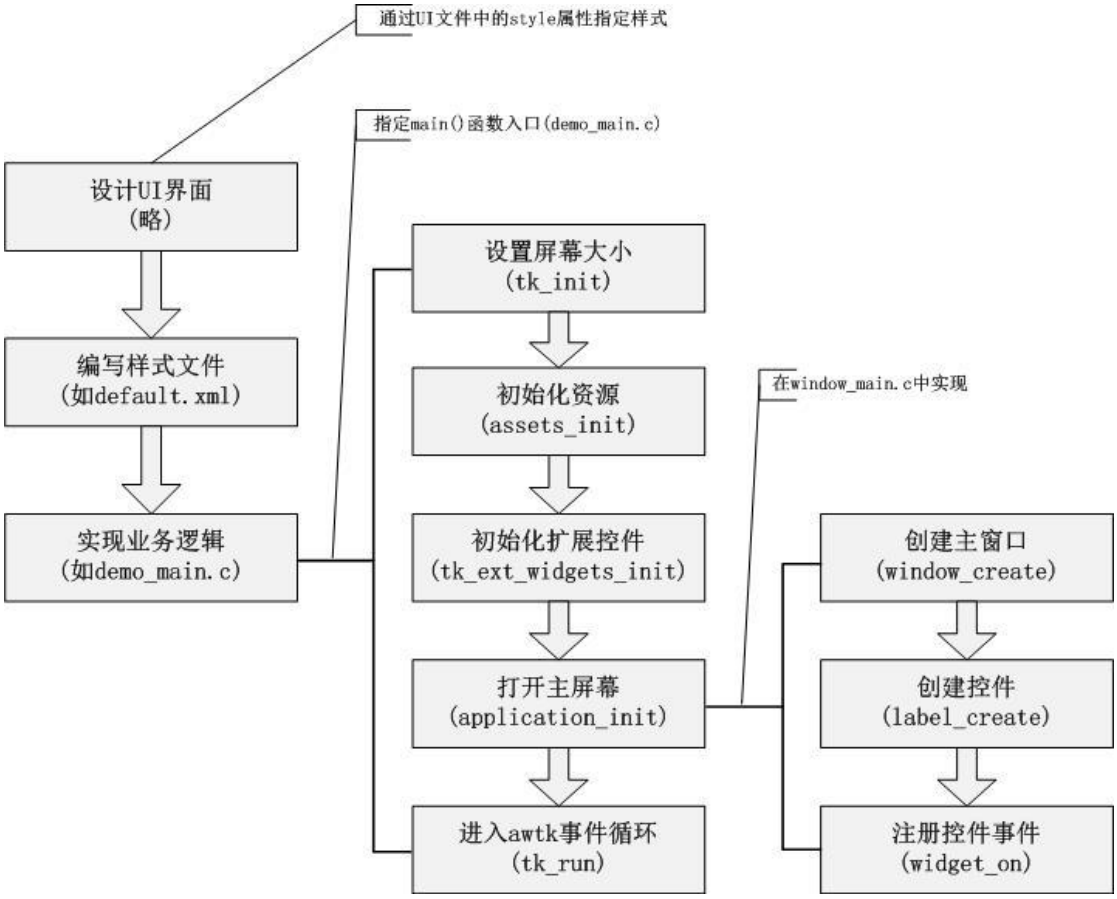



图 2.1 hello world 实现流程

应用实现后就可以编译和执行代码了。

## 2.2 生成资源文件

1. 打开 assets_gen.bat 将 AWTK_DIR 修改为本地 AWTK 所在的相对目录，要修改的内容如下（如果按网上 GitHub 的目录下载的话就不需要修改）：

```
@set AWTK_DIR=../../awtk
```

2. 在 cmd 下运行下面命令，生成主题、字符串、字体、图片、UI 等资源的二进制文件，同时自动实现 assets_init()，程序默认使用 800x480 的屏幕资源：

```
assets_gen.bat
```

3. 执行完上面命令后会生成 res\assets\inc 目录、res\assets\raw 子目录下的.bin 文件以及 res\assets.inc 文件。

注意事项：如果改变（增加、删除，修改等）res/assets/raw 目录下的资源文件，都需要重新执行该命令。并重新编译源代码。

## 2.3 编译源代码

1. 在 cmd 下运行下面命令，程序默认使用 800x480 的屏幕资源：

```
scons
```

或运行下面命令，程序将使用 480x272 的屏幕资源（如果存在的话）：

```
scons LCD=480_272
```

2. 编译后生成的 exe 在.\bin 目录下，运行效果详见图 2.2。

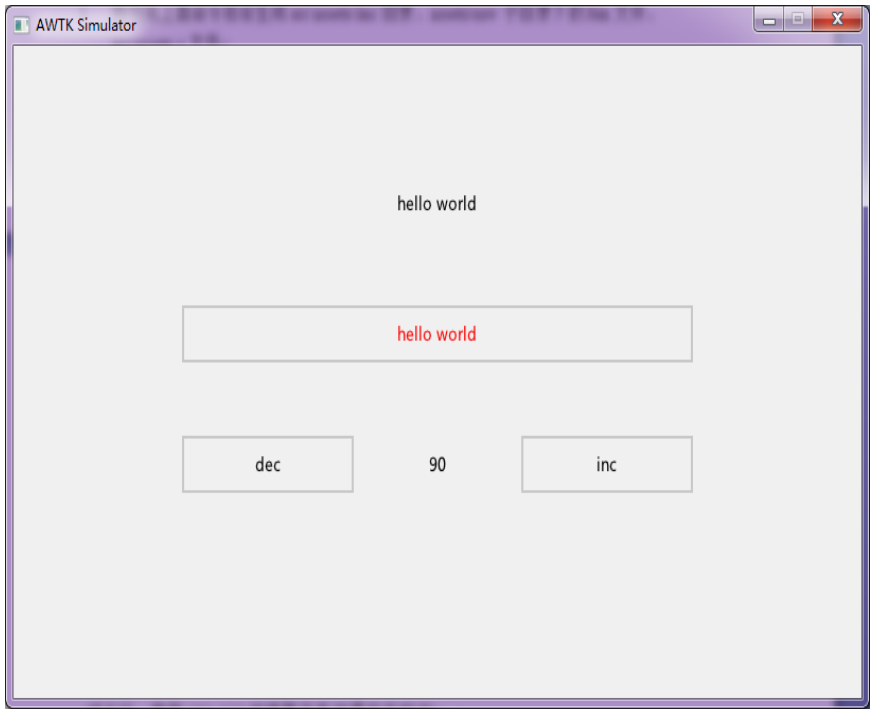



图 2.2 hello world 运行效果

注意事项：AWTK 和 HelloWorld-Demo 目录下 SConstruct 文件中的 NANOVG_BACKEND 的取值需要一样，如都设置为 NANOVG_BACKEND='AGGE'。

## 2.4 调试

有两种调试方式：使用 Visual Studio 和 Visual Studio Code，下面我们分别看看如何使用这两种方式调试我们的程序。

### 2.4.1 Visual Studio

scons 编译时并没有生成 Visual Studio 的工程，如果需要在 Visual Studio 中调试 AWTK 应用程序，可按下列步骤进行：

1. 打开 Visual Studio

2. 在"文件"菜单中点击"打开"并选中"项目/解决方案"
3. 选择 bin\demo.exe （或者其它要调试的可执行文件）
4. 在项目设置中设置调试参数（可选）
5. 将要调试的文件拖到 Visual Studio 中，在要调试的地方打断点进行调试

### 2.4.2 Visual Studio Code

使用 Visual Studio Code 调试时，可按下列步骤进行：

1. 添加配置，详见图 2.3。

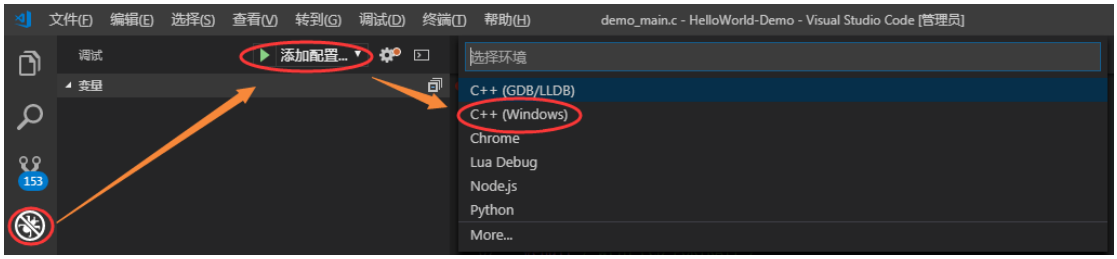



图 2.3 添加配置

2. 选择 C++(Windows）后，弹出界面，详见图 2.4。

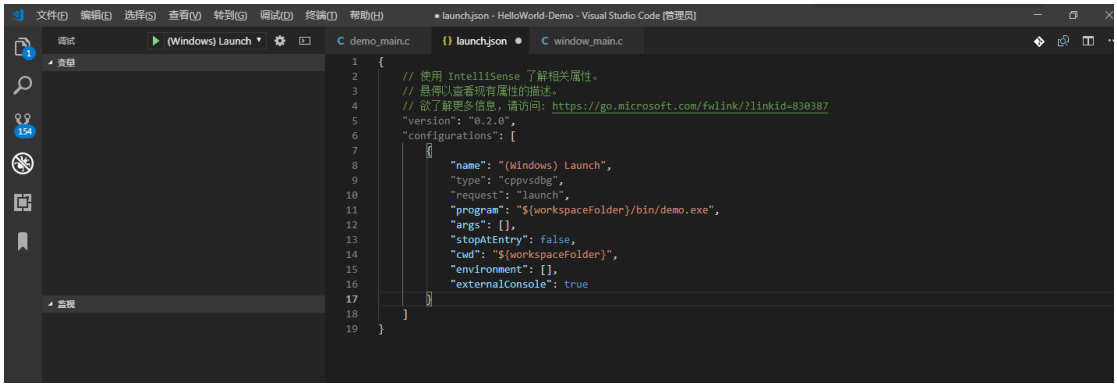



图 2.4 弹出界面

3. 将上图中 program 修改为要调试的应用程序的路径，即将"program": "enter program name, for example ${workspaceFolder}/a.exe"，修改成如下：

```
"program": "${workspaceFolder}/bin/demo.exe",
```

## 2.5 目录结构介绍

1. 我们先从应用的目录结构详见图 2.5，来了解一下 hello world 的实现过程，具体说明详见表 2.1。

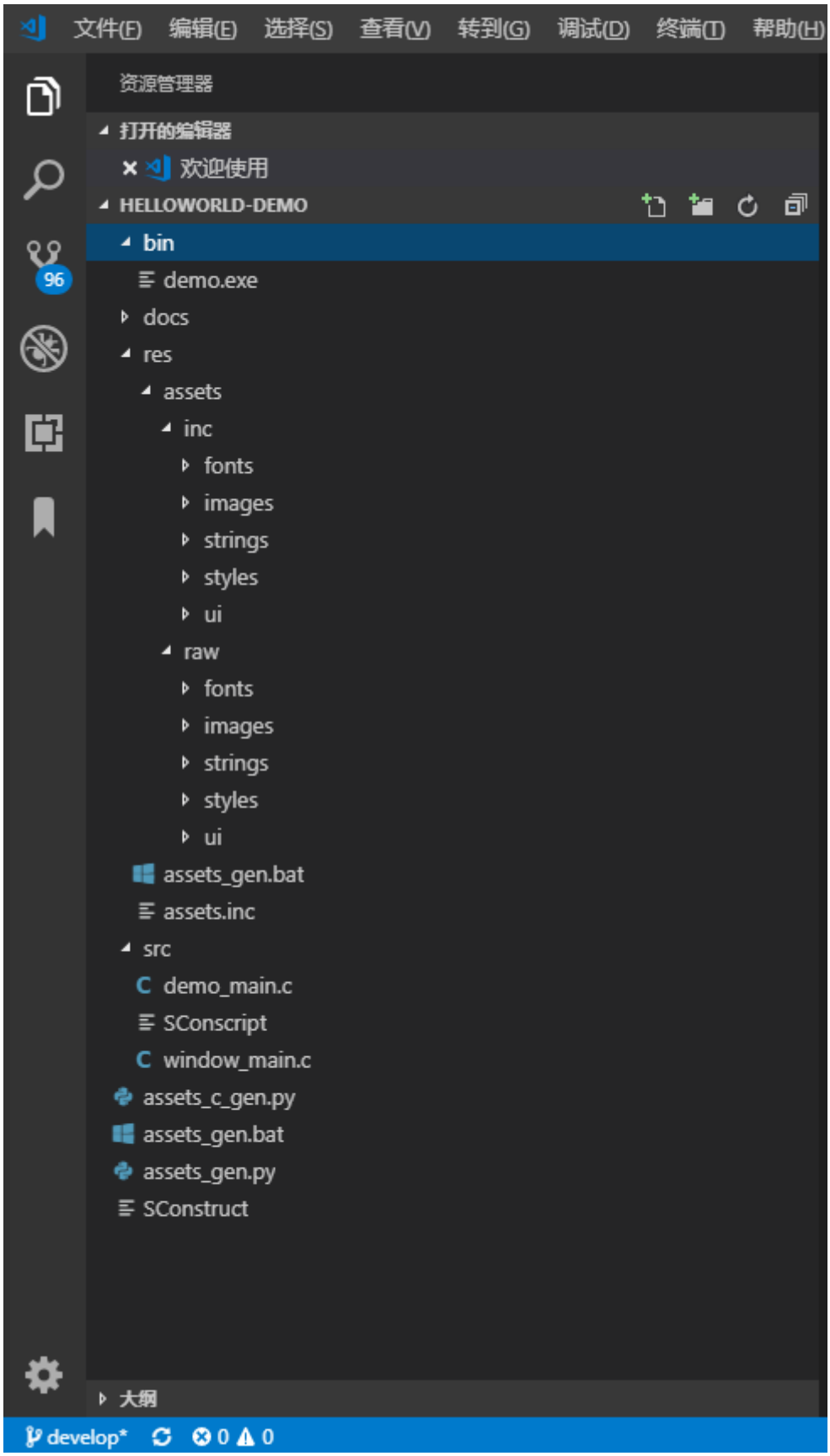



图 2.5 hello world 目录结构

表 2.1 helloworld 目录结构

| 目录/文件 | 说明 |
| --- | --- |
| bin | 可执行文件目录 |
| docs | 说明文档 |
| res | 资源文件目录 |
| src | 源代码 |
| SConstruct | scons 脚本 |
| assets_gen.bat | 双击生成资源文件（具体使用说明请打开该文件查阅） |
| assets_gen.py | 被 assets_gen.bat 调用，生成资源文件 |
| assets_c_gen.py | 被 assets_gen.bat 调用，生成 assets.c |

请注意上面的 res 目录，在有些案例中（如后面介绍到的经典案例中的 Chart-Demo）是没有该目录的，代替的是 res_480_272 和 res_800_480。res 目录表示的是该目录下的资源适合各种屏幕大小的资源，而 res_480_272 表示的是该目录下的资源只适合屏幕大小为 480*272 的，res_800_480 也是一样的道理。

2. 在 AWTK 中，约定 res\assets\raw 的目录结构详见表 2.2（注意目录位置和名称不可随意更改）：

表 2.2 raw 目录结构

| 目录 | 说明 |
| --- | --- |
| fonts | 字体文件目录，AWTK 目前支持 TTF |
| images | 图片文件目录（包含 x1、x2、x3 目录），AWTK 目前支持的格式有 png、jpg、bmp |
| strings | 多国语言文件目录，包含使用 xml 描述文本信息在不同语言环境下表述结果的语言文件 |
| styles | 样式文件目录，包含使用 xml 描述界面外观的样式文件，比如下文所介绍的 default.xml |
| ui | UI 文件目录，包含使用 xml 描述界面结构的 ui 文件，比如下文所介绍的 main.xml |

3. res\assets\images 目录下可以建立的文件夹（默认建立 x1）详见表 2.3。

表 2.3 images 目录结构

| 目录 | 说明 |
| --- | --- |
| x1 | 普通 LCD 的图片 |
| x2 | 高清 LCD 的图片 |
| x3 | 手机等超高清 LCD 的图片 |

- 对于嵌入式系统，一般只需要 x1 的图片。如果开发环境使用高清的 PC 显示器，为了方便 PC 上看效果，建议也准备一套 x2 的图片
- 需要注意的是，AWTK 中使用资源时，除了点阵字体（BITMAP_FONT）之外，资源的名称一般为文件名，且不包含文件后缀

## 2.6 代码实现

### 2.6.1 主函数入口

demo_main.c 是 main 函数入口，主要完成设置屏幕大小、资源的初始化以及 AWTK 框架的加载等，该文件通常不需要修改，详见程序清单 2.1。

程序清单 2.1 demo_main.c

```
#include "awtk.h"

#include "../res/assets.inc"

#ifdef USE_GUI_MAIN
int gui_app_start(int lcd_w, int lcd_h)
{
  tk_init(lcd_w, lcd_h, APP_MOBILE, NULL, NULL);
#else

#ifdef WIN32
#include <windows.h>
int WINAPI WinMain(HINSTANCE hinstance, HINSTANCE hprevinstance,
                   LPSTR lpcmdline, int ncmdshow)
{
#else
int main(void)
{
#endif

  int lcd_w = 800;
  int lcd_h = 480;
#if defined(LCD_W) && defined(LCD_H)
  lcd_w = LCD_W;
  lcd_h = LCD_H;
#endif

#ifdef WITH_FS_RES
  char res_root[MAX_PATH + 1];
  char app_root[MAX_PATH + 1];
  path_app_root(app_root);
  memset(res_root, 0x00, sizeof(res_root));
  path_build(res_root, MAX_PATH, app_root, "res", NULL);
  tk_init(lcd_w, lcd_h, APP_MOBILE, NULL, res_root);
#else
  tk_init(lcd_w, lcd_h, APP_MOBILE, NULL, NULL);
#endif
```

```
#endif

//#define WITH_LCD_PORTRAIT 1
#if defined(USE_GUI_MAIN) && defined(WITH_LCD_PORTRAIT)
  if (lcd_w > lcd_h)
  {
    tk_set_lcd_orientation(LCD_ORIENTATION_90);
  }
#endif /*WITH_LCD_PORTRAIT*/

#ifdef WITH_LCD_LANDSCAPE
  if (lcd_w < lcd_h)
  {
    tk_set_lcd_orientation(LCD_ORIENTATION_90);
  }
#endif /*WITH_LCD_PORTRAIT*/

  /* 初始化资源 */
  assets_init();

  /* 初始化扩展控件 */
  tk_ext_widgets_init();

  /* 打开主屏幕 */
  application_init();

  /* 进入 awtk 事件循环 */
  tk_run();

  return 0;
}
```

### 2.6.2 创建界面

为了方便开发，我们只需要在源文件中包含#include "awtk.h"这个头文件就可以。hello world 的实现逻辑主要在 window_main.c 中，下面让我们一起来看看吧。

首先介绍 hello world 界面的实现过程，详见程序清单 2.2。

程序清单 2.2 hello world 界面实现

```
// window_main.c

/**
 * 初始化
 */
void application_init() {
```

```
  widget_t *win = window_create(NULL, 0, 0, 0, 0);

  char height[] = "40";

  /* 创建文本框*/
  widget_t *label_4_edit = label_create(win, 0, 0, 0, 0);
  widget_set_text(label_4_edit, L"hello world");
  widget_set_name(label_4_edit, "label_4_edit");
  widget_set_self_layout_params(label_4_edit, "center", "20%", "60%", height);

  /* 创建编辑框 */
  widget_t *edit = edit_create(win, 0, 0, 0, 0);
  edit_set_input_type(edit, INPUT_EMAIL);
  widget_set_text(edit, L"hello world");
  widget_set_self_layout_params(edit, "center", "40%", "60%", height);
  widget_on(edit, EVT_VALUE_CHANGING, on_changing, win);

  /* 创建递减按钮 */
  widget_t *dec_btn = button_create(win, 0, 0, 0, 0);
  widget_set_text(dec_btn, L"dec");
  widget_set_self_layout_params(dec_btn, "20%", "60%", "20%", height);
  widget_on(dec_btn, EVT_CLICK, on_dec_click, win);

  /* 创建 label 显示递增数值 */
  widget_t *label_4_btn = label_create(win, 0, 0, 0, 0);
  widget_set_text(label_4_btn, L"88");
  widget_set_name(label_4_btn, "label_4_btn");
  widget_set_self_layout_params(label_4_btn, "center", "60%", "20%", height);

  /* 创建递增按钮 */
  widget_t *inc_btn = button_create(win, 0, 0, 0, 0);
  widget_set_text(inc_btn, L"inc");
  widget_set_self_layout_params(inc_btn, "60%", "60%", "20%", height);
  widget_on(inc_btn, EVT_CLICK, on_inc_click, win);

  widget_layout(win);
}
```

### 2.6.3 响应按钮事件

在 C 代码中，使用 widget_on 为" EVT_CLICK "点击事件注册一个回调函数（比如 on_inc_click）即可，如：点击界面上的 dec 和 inc 按钮响应事件，详见程序清单 2.3。

程序清单 2.3 响应按钮事件

```
// window_main.c
/**
```

```
 * Label 文本的数值  + offset
 */
static ret_t label_add(widget_t *label, int32_t offset) {
  if (label) {
    int32_t val = 0;
    if (wstr_to_int(&(label->text), &val) == RET_OK) {
      char text[32];
      val += offset;
      val = tk_max(-200, tk_min(val, 200));
      tk_snprintf(text, sizeof(text), "%d", val);
      widget_set_text_utf8(label, text);

      return RET_OK;
    }
  }
  return RET_FAIL;
}

/**
 * 递增按钮事件
 */
static ret_t on_inc_click(void *ctx, event_t *e) {
  widget_t *win = WIDGET(ctx);
  widget_t *label = widget_lookup(win, "label_4_btn", TRUE);
  label_add(label, 1);

  return RET_OK;
}

/**
 * 递减按钮事件
 */
static ret_t on_dec_click(void *ctx, event_t *e) {
  widget_t *win = WIDGET(ctx);
  widget_t *label = widget_lookup(win, "label_4_btn", TRUE);
  label_add(label, -1);

  return RET_OK;
}
```

### 2.6.4 响应文本正在改变事件

为界面中的编辑框注册 EVT_VALUE_CHANGING（文本正在改变事件），实现编辑框在编辑的同时上面文本框的内容也随着变化，详见程序清单 2.4。

程序清单 2.4 响应文本正在改变事件

```
// window_main.c
/**
 * 正在编辑事件
 */
static ret_t on_changing(void *ctx, event_t *evt) {
  widget_t *target = WIDGET(evt->target);
  widget_t *win = WIDGET(ctx);
  widget_t *label = widget_lookup(win, "label_4_edit", TRUE);
  widget_set_text(label, target->text.str);

  return RET_OK;
}
```

到这里一个简单的 hello world 小程序就完成了，是不是觉得 AWTK 很强大了呢，有兴趣的就赶紧行动吧。

# 第3章 开发基础

**✍本章导读**

学习一门新的开发语言，我们得学习一些关于该门语言的基础知识，如语法。同样的在学习 AWTK 的过程中，我们也需要学习一些基本的基础知识，在了解了这些知识之后，我们才能更加好的学习后面的章节。

## 3.1 简介

在本章按照 AWTK 框架的构成，阐述 AWTK 开发过程中所需的基础知识，包括主题样式、布局管理器、定时器、事件处理等。本章以 AWTK 源码目录下自带的例子（awtk\demos）来介绍关于 AWTK 的基础知识，所以本章涉及的代码均可以在 AWTK 源码目录下找到。

## 3.2 主题样式

设计漂亮的界面并非程序员的强项，AWTK 通过主题提供这样一种机制，让设计漂亮的界面变得非常容易。通过主题，可以改变控件的背景颜色、边框颜色、字体颜色、字体、字体大小、背景图片、背景图片的显示方式和图标等属性。同时 AWTK 也提供了一些主题重用的机制，让主题文件的开发和维护变得容易。

### 3.2.1 主题结构

AWTK 的主题按控件进行分类，每种控件可以有多种不同的风格，每种风格下又有不同状态下的配置，详见程序清单 3.1。

程序清单 3.1 default.xml

```
<!--awtk\demos\assets\raw\styles\default.xml -->
...
<label>
  <style name="default">
    <normal text_color="black" />
  </style>

  <style name="left">
    <normal text_color="red" text_align_h="left" border_color="#a0a0a0" margin="4" />
  </style>
...
</label>
```

上面是文本框主题配置（可以按需增加 style 的数量），其中定义两种不同的文本风格：

- default 为缺省的文本风格（如果控件中不指定样式将使用 default 样式）
- left 为文本水平居左风格

主题的各个属性，如果出现在控件中（如<label text_color="red">），则为该控件下各个 style 的缺省值。如果出现在 style 中（如<style text_color="red">），则为该 style 的下各种状态的缺省值，这样可以实现类似继承的重用机制。

### 3.2.2 主题属性

AWTK 为控件提供了丰富的属性详见表 3.1，其中颜色可使用标准名称、#开头的 16 进制值和 rgba 合成的值。

表 3.1 主题属性

| 属性 | 说明 |
| --- | --- |
| bg_color | 背景颜色 |
| fg_color | 前景颜色。用途视具体控件而定，如进度条已完成部分的颜色使用前景颜色 |
| text_color | 文字的颜色 |
| tips_text_color | 提示文字的颜色 |
| border_color | 边框颜色 |
| border | 边框类型，取值为 left/right/top/bottom 和 all，以及它们的组合 |
| font_name | 字体的名称 |
| font_size | 字体的大小 |
| font_style | 字体的风格（目前还不支持），取值为：italic/bold/underline。可用","分隔 |
| text_align_h | 文本的水平对齐方式。取值为 left/center/right |
| text_align_v | 文本的垂直对齐方式。取值为 top/middle/bottom |
| bg_image | 背景图片 |
| bg_image_draw_type | 背景图片的绘制方式 |
| fg_image | 前景图片。用途视具体控件而定，如进度条已完成部分的图片使用前景图片 |
| fg_image_draw_type | 前景图片的绘制方式 |
| icon | 图标。用途视具体控件而定，如 check_button 的图标，按钮上的图标 |
| active_icon | active 图标。用途视具体控件而定，目前 slideview 的页面指示器会用到 |
| icon_at | 图标的位置，取值为 left/right/top/bottom |
| x_offset | 在 X 坐标方向上的偏移（可用来实现按下的效果） |
| y_offset | 在 Y 坐标方向上的偏移（可用来实现按下的效果） |
| margin | 边距 |
| round_radius | 背景和边框的圆角半径（仅在定义 WITH_VGCANVAS 时有效） |

### 3.2.3 为控件指定 style

在上面的 default.xml 定义了 label 的 left 样式，接下来在 C 代码中就可以使用函数 widget_use_style 指定样式了，详见程序清单 3.2。

程序清单 3.2 使用 widget_use_style 指定样式

```
//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  label = label_create(win, 10, 40, 80, 30);
  widget_set_text(label, L"Left");
  widget_use_style(label, "left");
  ...
}
```

或在 UI 文件中通过 style 属性指定 label 的样式，详见程序清单 3.3。

程序清单 3.3 在 UI 文件中指定样式

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  ...
  <row x="0" y="80" w="100%" h="30" children_layout="default(r=1,c=3,xm=2,s=10)">
    <label style="left" name="left" text="Left"/>
    <label style="center" name="center" text="Center"/>
    <label style="right" name="right" text="Right"/>
  </row>
  ...
</window>
```

### 3.2.4 每个窗口支持独立的主题

像微信小程序那样，AWTK 中每个窗口（包括对话框和其它窗口）可以有自己的主题文件。

- 通过窗口的 theme 属性来指定窗口主题文件名（方便多个窗口共用一个主题文件）
- 如果没有指定 theme 属性，以窗口的 name 属性作为窗口的主题文件名
- 如果没有指定 name 属性，以窗口的 UI 文件名作为窗口的主题文件名
- 以窗口自己的主题文件优先，其次为缺省的主题文件

## 3.3 布局管理器

### 3.3.1 为什么需要布局参数

如果界面上元素是预先知道的，而且窗口的大小也是固定的，通过可视化的工具，以所见即所得的方式，去创建界面是最轻松的方式。但是在下列情况下，使用布局参数却是更好的选择。

1. 窗口的大小可以动态调整的。
2. 需要适应不同大小的屏幕。
3. 界面上的元素是动态的，需要用程序创建界面。

### 3.3.2 概述

AWTK 的布局器（layouter）分为两类，一类用于对控件自身进行布局，另一类用于对子控件进行布局详见图 3.1。

- self_layout 对控件自身进行布局
- children_layout 用于对子控件进行布局

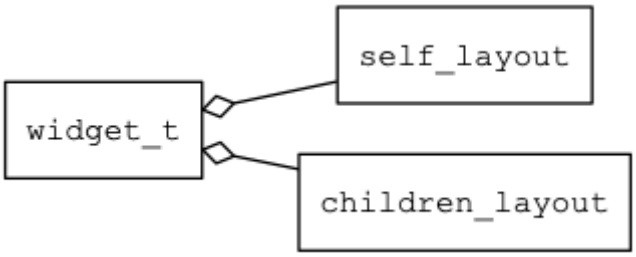



图 3.1 layouter 布局器

AWTK 提供了灵活的扩展机制，可以方便的扩展新的布局方式，所以 self_layouter 和 children_layouter 都是接口。

### 3.3.3 示例

AWTK 提供了简单而又强大的布局参数，在 AWTK 既可以使用 C 语言也可以使用 XML 创建 UI 界面，下面是使用 XML 创建界面的一个简单示例详见程序清单 3.4。

程序清单 3.4 time_clock.xml

```
<!-- awtk\demos\assets\raw\ui\time_clock.xml -->
<window anim_hint="htranslate" >
   <time_clock x="c" y="m" w="300" h="300" bg_image="clock_bg" image="clock"
      hour_image="clock_hour" minute_image="clock_minute" second_image="clock_second"/>
</window>
```

time_clock.xml 运行后的效果，详见图 3.2。


图 3.2 time_clock.xml 运行效果图

### 3.3.4 控件自身的布局

AWTK 目前仅仅实现了缺省布局方式，以后陆续实现 css flex 等布局方式，详见图 3.3。

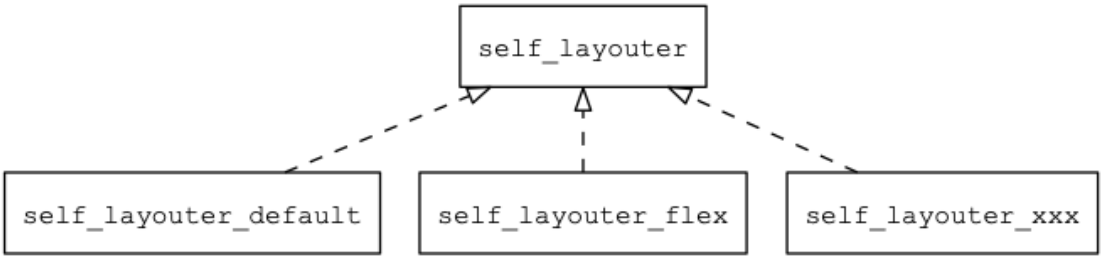



图 3.3 self_layouter 布局

1. 参数

缺省布局中有 5 个参数，详见表 3.2。

表 3.2 缺省布局参数

| 参数 | 说明 |
| --- | --- |

| | |
|---|---|
| x | x 坐标 |
| y | y 坐标 |
| w | 控件宽度 |
| h | 控件高度 |
| floating | 是否为浮动布局。如果设置为 true，该控件不受父控件的 children_layouter 的影响 |

### 2. 使用方法

(1)像素方式

直接指定控件的 x/y/w/h 的像素值，这是缺省的方式，也是最不灵活的方式。

示例：

在 XML 界面描述文件中：

```
<button x="10" y="5" w="80" h="30" text="ok"/>
```

在 C 代码中：

```
widget_move_resize(btn, 10, 5, 80, 30);
```

(2)百分比

- x/w 的值如果包含"%"，则自动换算成相对其父控件宽度的百分比
- y/h 的值如果包含"%"，则自动换算成相对其父控件高度的百分比

示例：

在 XML 界面描述文件中：

```
<button x="10%" y="10" w="50%" h="30" text="ok"/>
```

在 C 代码中：

```
widget_set_self_layout_params(btn, "10%", "10", "50%", "30");
widget_layout(btn);
```

在代码中设置控件的布局参数，方法类似，后面就不再举例子了。

(3)水平居中

让控件在水平方向上居中，只需要将 x 的值设置成"c"或者"center"即可。

示例：

```
<button x="center" y="10" w="50%" h="30" text="ok"/>
```

(4)垂直居中

让控件在垂直方向上居中，只需要将 y 的值设置成"m"或者"middle"即可。

示例：

```
<button x="center" y="middle" w="50%" h="30" text="ok"/>
```

(5)位于右边

让控件位于父控件的右侧，只需要将 x 的值设置成"right"即可。

示例：

```
<button x="right" y="10" w="50%" h="30" text="ok"/>
```

如果还想离右侧有一定距离，可以在 right 后指定距离的像素。

示例：

```
<button x="right:20" y="10" w="50%" h="30" text="ok"/>
```

(6)位于底部

让控件位于父控件的底部，只需要将 y 的值设置成"bottom"即可。

示例：

```
<button x="10" y="bottom" w="50%" h="30" text="ok"/>
```

如果还想离底部有一定距离，可以在 bottom 后指定距离的像素。

示例：

```
<button x="10" y="bottom:20" w="50%" h="30" text="ok"/>
```

(7)宽度和高度为负数

无论是像素模式还是百分比模式，宽度和高度均可为负数。

- 宽度为负数。其值为父控件的宽度+该负值
- 高度为负数。其值为父控件的高度+该负值

(8)浮动布局

如果 floating 设置为 true，该控件不受父控件的 children_layouter 的影响。

示例：

```
<button x="10" y="20" w="50" h="30" floating="true" "text="ok"/>
```

### 3.3.5 子控件的布局

AWTK 目前仅仅实现了缺省布局方式，以后陆续实现 css flex 等布局方式详见图 3.4。

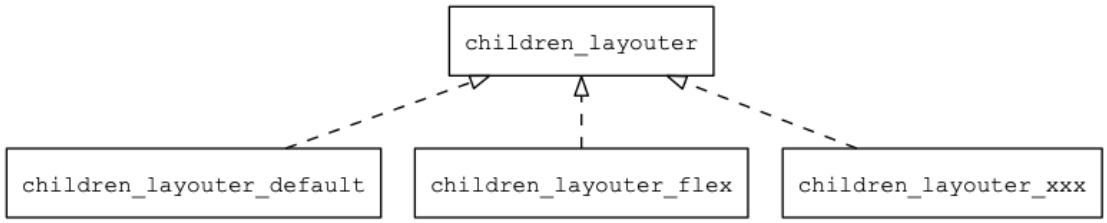



图 3.4 子控件的布局

#### 1. 语法

子控件布局器统一使用 children_layout 属性指定，其语法为：

```
缺省子控件布局器  => default '(' PARAM_LIST ')'
PARAM_LIST => PARAM | PARAM ',' PARAM_LIST
```

示例：

```
<view x="0" y="0" w="100%" h="100%" children_layout="default(c=2,r=8,m=5,s=5)">
```

#### 2. 参数

为了方便父控件布局子控件，AWTK 提供了下面几个参数详见表 3.3。

表 3.3 子控件布局参数

| 参数 | 简写 | 说明 |
| --- | --- | --- |
| rows | r | 行数 |
| cols | c | 列数 |

| | | |
|---|---|---|
| width | w | 子控件的宽度，可以用来计算列数，与 cols 互斥 |
| height | h | 子控件的高度，可以用来计算行数，与 rows 互斥 |
| x_margin | xm | 水平方向的边距 |
| y_margin | ym | 垂直方向的边距 |
| spacing | s | 子控件之间的间距 |

在 C 代码中，可以通过下面的函数设置这几个参数：

```
/**
 * @method widget_set_children_layout
 * 设置子控件的布局参数。
 * @annotation ["scriptable"]
 * @param {widget_t*} widget 控件对象。
 * @param {const char*} params 布局参数。
 * @return {ret_t} 返回 RET_OK 表示成功，否则表示失败。
 */
ret_t widget_set_children_layout(widget_t* widget, const char* params);
```

示例：

```
widget_set_children_layout(w, "default(r=2,c=2)");
```

XML 中，可以通过 children_layout 属性设置:

```
<column x="20" y="160" w="50%" h="60" children_layout="default(r=2,c=1,ym=2,s=10)" >
    <check_button name="c1" text="Book"/>
    <check_button name="c2" text="Food"/>
 </column>
```

下面我们看看，如何调整 rows/cols 两个参数，来实现不同的布局方式。

### 3. 使用方法

(1)缺省

在没有设置子控件布局参数时，采用缺省的布局方式，父控件啥事也不做，完全由子控件自己的布局参数决定。

(2)hbox

当 rows=1,cols=0 时，所有子控件在水平方向排成一行，可以实现其它 GUI 中 hbox 的功能，子控件的参数详见表 3.4。

表 3.4 hbox 参数

| 属性 | 说明 |
|---|---|
| x | 从左到右排列，由布局参数计算而出 |
| y | 为 y_margin |
| w | 由子控件自己决定 |
| h | 为父控件的高度-2*y_margin，子控件需要自己决定宽度 |

示例：

```
<window>
  <view x="c" y="m" w="300" h="30" children_layout="default(r=1,c=0,s=5)">
```

```
        <button text="1" w="20%"/>
        <button text="2" w="30%"/>
        <button text="3" w="30%"/>
        <button text="4" w="20%"/>
    </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

(3)vbox

当 cols=1,rows=0 时，所有子控件在垂直方向排成一列，可以实现其它 GUI 中 vbox 的功能，子控件的参数详见表 3.5。

表 3.5 vbox 参数

| 属性 | 说明 |
| --- | --- |
| x | x_margin |
| y | 从上到下排列，由布局参数计算而出 |
| w | 为父控件的宽度-2*x_margin |
| h | 由子控件自己决定，子控件需要自己决定宽度 |

示例：

```
<window>
    <view x="c" y="m" w="80" h="200" children_layout="default(r=0,c=1,s=5)">
        <button text="1" h="20%"/>
        <button text="2" h="30%"/>
        <button text="3" h="30%"/>
        <button text="4" h="20%"/>
    </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

(4)listbox

当 cols=1,rows=N 时，所有子控件在垂直方向排成一列，可以实现其它 GUI 中 listbox 的功能，子控件的参数详见表 3.6。

表 3.6 listbox 参数

| 属性 | 说明 |
| --- | --- |
| x | x_margin |
| y | 从上到下排列，由布局参数计算而出 |
| w | 为父控件的宽度-2*x_margin |
| h | 为父控件的高度(减去边距和间距)分成成 N 等分，子控件无需指定 x/y/w/h 等参数 |

示例：

```
<window>
  <view x="c" y="m" w="200" h="200" children_layout="default(r=4,c=1,s=5)">
      <button text="1" />
      <button text="2" />
      <button text="3" />
      <button text="4" />
  </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

(5)grid

当 cols=N,rows=N 时，所有子控件放在 M x N 的网格中，可以实现其它 GUI 中 grid 的功能，子控件无需指定 x/y/w/h 等参数。

示例：

```
<window>
  <view x="c" y="m" w="200" h="200" children_layout="default(r=2,c=2,s=5)">
      <button text="1" />
      <button text="2" />
      <button text="3" />
      <button text="4" />
  </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

(6)浮动布局

如果子控件的 floating 属性设置为 true，其不受 children_layout 的限制。

示例：

```
<window>
  <view x="c" y="m" w="200" h="200" children_layout="default(r=2,c=2,s=5)">
      <label text="1" />
      <label text="2" />
      <label text="3" />
      <label text="4" />
      <button text="floating" floating="true" x="c" y="m" w="80" h="30"/>
  </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

### 4. 高级用法

(1)子控件布局器和子控件自身的布局参数结合。

为了更大的灵活性，缺省子控件布局器可以和子控件自身的参数结合起来。

示例：

```
<window>
  <view x="c" y="m" w="200" h="200" children_layout="default(r=2,c=2,s=5)">
      <button text="1" x="0" y="0" w="50%" h="50%"/>
      <button text="2" x="r" y="m" w="60%" h="60%"/>
      <button text="3" x="c" y="m" w="70%" h="70%"/>
      <button text="4" x="c" y="m" w="80%" h="80%"/>
  </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

(2) 子控件自身的布局参数 x/y/w/h 均为像素方式时，需要用 self_layout 参数指定。

示例：

```
<window>
  <view x="c" y="m" w="200" h="200" children_layout="default(r=2,c=2,s=5)">
      <button text="1" self_layout="default(x=0,y=0,w=50,h=50)" />
      <button text="2" x="r" y="m" w="60%" h="60%"/>
      <button text="3" x="c" y="m" w="70%" h="70%"/>
      <button text="4" x="c" y="m" w="80%" h="80%"/>
  </view>
</window>
```

保存为 t.xml，可用 preview_ui（在 awtk\bin 目录下）预览效果：

```
./bin/preview_ui t.xml
```

## 3.4 定时器

AWTK 提供了很便捷的定时器，包括添加、删除定时器等。

### 1. 添加定时器

(1)函数原型

```
uint32_t timer_add (timer_func_t on_timer, void* ctx, uint32_t duration_ms);
```

(2)参数说明

关于 timer_add 函数参数的说明详见表 3.7。

表 3.7 timer_add 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | uint32_t | 返回 timer 的 ID，TK_INVALID_ID 表示失败 |
| on_timer | timer_func_t | timer 回调函数 |
| ctx | void* | timer 回调函数的上下文 |
| duration_ms | uint32_t | 时间 |

(3)示例

下面代码是 awtk\bin\demo1.exe 下定时更新进度条的一个示例，详见程序清单 3.5。

程序清单 3.5 timer_add 示例

```
//awtk\demos\common.inc
static ret_t on_timer(const timer_info_t* timer) {
   widget_t* progress_bar = (widget_t*)timer->ctx;
   uint8_t value = (PROGRESS_BAR(progress_bar)->value + 5) % 100;
   progress_bar_set_value(progress_bar, value);

   return RET_REPEAT;
}

//awtk\demos\demo1_app.c
ret_t application_init() {
   ...
   progress_bar = progress_bar_create(win, 10, 80, 168, 20);
   widget_set_value(progress_bar, 40);
   timer_add(on_timer, progress_bar, 500);
   ...
}
```

上面的 on_timer 函数返回值 RET_REPEAT 表示是个持续的定时器，如果只想该定时器执行一次可以返回 RET_REMOVE。

### 2. 删除定时器

(1)函数原型

```
ret_t timer_remove (uint32_t timer_id);
```

(2)参数说明

关于 timer_remove 函数参数的说明详见表 3.8。

表 3.8 timer_remove 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败。 |
| timer_id | uint32_t | timerID 为 timer_add 函数的返回值 |

## 3.5 事件处理

emitter 实现了通用的事件注册、注销和分发功能，widget 对此做了进一步包装，使用起来非常方便。

### 1. 注册控件事件

(1)函数原型

```
uint32_t widget_on(widget_t* widget, event_type_t type, event_func_t on_event, void* ctx);
```

(2)参数说明

关于 widget_on 函数参数的说明详见表 3.9。

表 3.9 widget_on 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | uint32_t | 用于 widget_off |
| widget | widget_t* | 控件对象 |
| type | event_type_t | 事件类型 |
| on_event, | event_func_t | 事件处理函数 |
| ctx | void* | 事件处理函数上下文 |

(3)示例

下面代码是 awtk\bin\demo1.exe 下实现创建一个进度条和一个按钮，并为按钮的"点击"事件注册一个处理函数，当按钮被点击时，增加进度条的值，详见程序清单 3.6。

程序清单 3.6 widget_on 示例

```
//awtk\demos\common.inc
static ret_t on_inc(void* ctx, event_t* e) {
  widget_t* progress_bar = (widget_t*)ctx;
  uint8_t value = (PROGRESS_BAR(progress_bar)->value + 10) % 100;
  progress_bar_set_value(progress_bar, value);
  (void)e;
  return RET_OK;
}

//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  progress_bar = progress_bar_create(win, 260, 80, 20, 118);
  widget_set_value(progress_bar, 40);
  progress_bar_set_vertical(progress_bar, TRUE);
  widget_on(ok, EVT_CLICK, on_inc, progress_bar);
  ...
}
```

### 2. 注销控件事件

由于窗口关闭时会销毁其中所有控件，所以一般不需要手工去注销。如果确实需要，可以使用 widget_off。

(1)函数原型

```
ret_t widget_off(widget_t* widget, uint32_t id);
```

(2)参数说明

关于 widget_off 函数参数的说明详见表 3.10。

表 3.10 widget_off 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |

| | | |
|---|---|---|
| id | uint32_t | widget_on 返回的 ID |

# 第4章 框架控件的开发应用

**✍本章导读**

AWTK 框架为开发者提供了丰富的基础控件，开发者通过组合这些基础控件进行快速开发、创建复杂的 GUI 界面。控件是视图层的基本组成元素，是构建 GUI 界面的重要元素。熟练的使用控件是高效开发 AWTK 应用程序的必备技能。

## 4.1 简介

我们将在本章详细阐述 AWTK 四大基本控件：窗口、基本控件、通用容器和扩展控件。AWTK 包括的控件详见图 4.1。

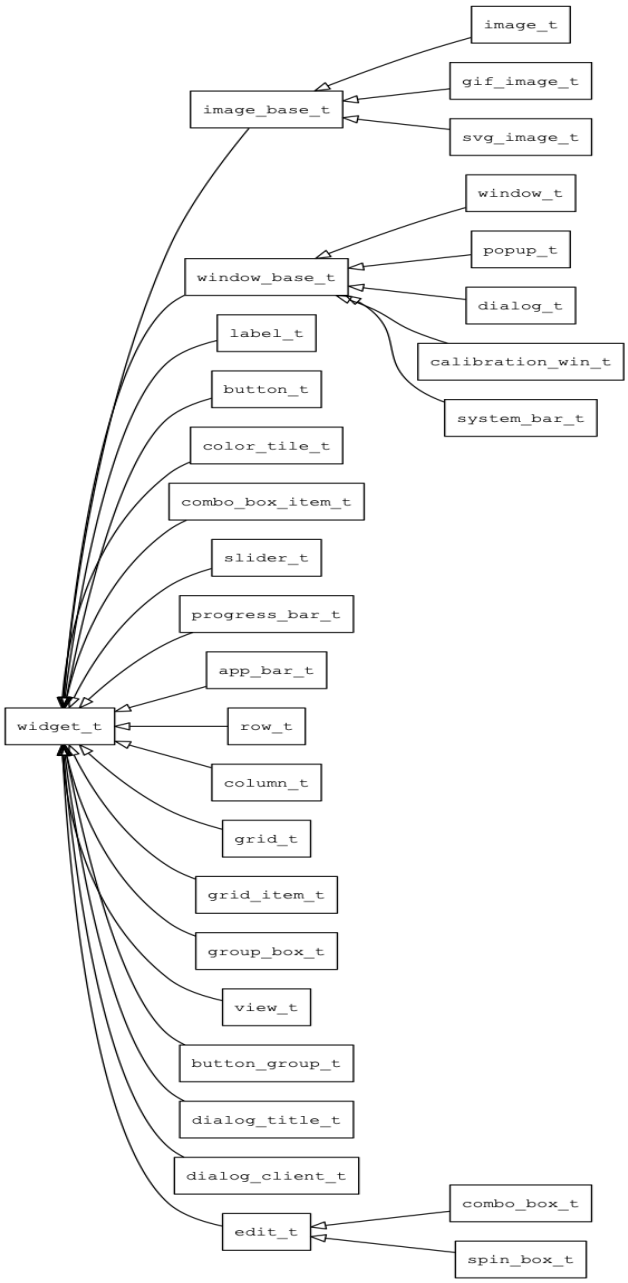



图 4.1 AWTK 控件总览

## 4.2 窗口

常用的窗口详见表 4.1。

表 4.1 常用窗口

| 窗口 | 说明 |
| --- | --- |
| window_base | 窗口基类（一般不直接使用） |
| dialog | 对话框 |
| popup | 弹出窗口（一般不直接使用） |
| window | 普通窗口 |
| system_bar | 系统状态窗口 |
| calibration_win | 电阻屏校准窗口 |

### 4.2.1 window

缺省的应用程序窗口，占用除 system_bar_t 之外的整个区域，请不要修改它的位置和大小（除非你清楚后果）。 window_t 是 window_base_t 的子类控件，window_base_t 的函数均适用于 window_t 控件。

1. 函数

window 提供的函数详见表 4.2。

表 4.2 window 函数列表

| 函数名称 | 说明 |
| --- | --- |
| window_cast | 转换为 window 对象（供脚本语言使用） |
| window_close | 关闭窗口 |
| window_create | 创建 window 对象 |
| window_open | 打开窗口 |
| window_open_and_close | 打开窗口，并关闭某个窗口 |

(1) window_cast

- 函数原型

```
widget_t* window_cast (widget_t* widget);
```

- 参数说明，详见表 4.3。

表 4.3 window_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | window 对象 |
| widget | widget_t* | window 对象 |

(2) window_close

- 函数原型

```
ret_t window_close (widget_t* widget);
```

- 参数说明，详见表 4.4。

表 4.4 window_close 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | window_base 对象 |

(3) window_create

- 函数原型

```
widget_t* window_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.5。

表 4.5 window_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(4) window_open

- 函数原型

```
widget_t* window_open (char* name);
```

- 参数说明，详见表 4.6。

表 4.6 window_open 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| name | char* | window_base 的名称 |

(5) window_open_and_close

- 函数原型

```
widget_t* window_open_and_close (char* name, widget_t* to_close);
```

- 参数说明，详见表 4.7。

表 4.7 window_open_and_close 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| name | char* | window_base 的名称 |
| to_close | widget_t* | 关闭该窗口 |

2. 示例

- 在 xml 中，使用"window"标签创建窗口。无需指定坐标和大小，可以指定主题和动画名称，详见程序清单 4.1。

程序清单 4.1 xml 中创建 window

```
<!-- awtk\demos\assets\raw\ui\main.xml -->
<window closable="no" text="Desktop" anim_hint="htranslate">
    ...
</window>
```

- 在 C 代码中，使用函数 window_create 创建窗口，无需指定父控件、坐标和大小，使用 0 即可，详见程序清单 4.2。

程序清单 4.2 在 C 代码中创建 window

```
//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  widget_t* win = window_create(NULL, 0, 0, 0, 0);
  ...
}
```

### 4.2.2 dialog

对话框是一种特殊的窗口，大小和位置可以自由设置。AWTK 中的对话框是模态的，也就是说用户不能操作对话框后面的窗口。对话框通常由对话框标题和对话框客户区两部分组成详见图 4.2。

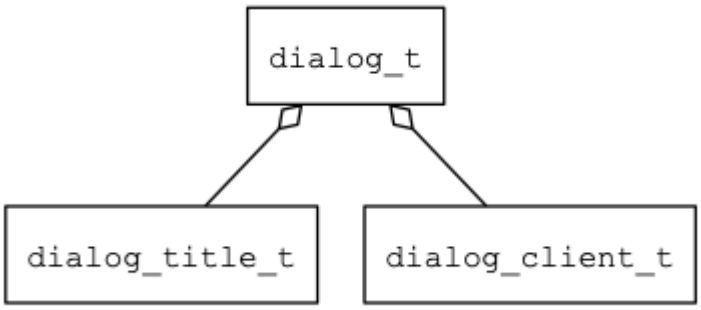



图 4.2 对话框的组成

1. 函数

dialog 提供的函数详见表 4.8。

表 4.8 dialog 函数列表

| 函数名称 | 说明 |
| --- | --- |
| dialog_cast | 转换 dialog 对象（供脚本语言使用） |
| dialog_create | 创建 dialog 对象 |
| dialog_create_simple | 创建 dialog 对象，同时创建 title/client |
| dialog_get_client | 获取 client 控件 |

| | |
|---|---|
| dialog_get_title | 获取 title 控件 |
| dialog_modal | 模态显示对话框 |
| dialog_open | 从资源文件中加载并创建 Dialog 对象 |
| dialog_quit | 退出模态显示，关闭对话框 |
| dialog_set_title | 设置对话框的标题文本 |

(1) dialog_cast

- 函数原型

```
widget_t* dialog_cast (widget_t* widget);
```

- 参数说明，详见表 4.9。

表 4.9 dialog_cast 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | dialog 对象 |
| widget | widget_t* | dialog 对象 |

(2) dialog_create

- 函数原型

```
widget_t* dialog_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.10。

表 4.10 dialog_create 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | dialog 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) dialog_create_simple

- 函数原型

```
widget_t* dialog_create_simple (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.11。

表 4.11 dialog_create_simple 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | dialog 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |

| h | wh_t | 高度 |
|---|---|---|

(4) dialog_get_client

- 函数原型

```
widget_t* dialog_get_client (widget_t* widget);
```

- 参数说明，详见表 4.12。

表 4.12 dialog_get_client 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | client 对象 |
| widget | widget_t* | dialog 对象 |

(5) dialog_get_title

- 函数原型

```
widget_t* dialog_get_title (widget_t* widget);
```

- 参数说明，详见表 4.13。

表 4.13 dialog_get_title 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | title 对象 |
| widget | widget_t* | dialog 对象 |

(6) dialog_modal

- 函数原型

```
ret_t dialog_modal (widget_t* widget);
```

- 参数说明，详见表 4.14。

表 4.14 dialog_modal 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | dialog 对象 |

(7) dialog_open

- 函数原型

```
widget_t* dialog_open (const char* name);
```

- 参数说明，详见表 4.15。

表 4.15 dialog_open 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | 对象 |
| name | const char* | dialog 的名称 |

(8) dialog_quit

- 函数原型

```
ret_t dialog_quit (widget_t* widget, uint32_t code);
```

- 参数说明，详见表 4.16。

表 4.16 dialog_quit 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | dialog 对象 |
| code | uint32_t | 退出码，作为 dialog_modal 的返回值 |

(9) dialog_set_title

- 函数原型

```
et_t dialog_set_title (widget_t* widget, char* title);
```

- 参数说明，详见表 4.17。

表 4.17 dialog_set_title 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | dialog 对象 |
| title | char* | 标题 |

2. 示例

- 在 xml 中，使用"dialog"标签创建对话框，详见程序清单 4.3。

程序清单 4.3 xml 中创建 dialog

```
<!-- awtk\demos\assets\raw\ui\dialog1.xml -->
<dialog anim_hint="center_scale(duration=300)" x="c" y="m" w="80%" h="160" text="Dialog">
  <dialog_title x="0" y="0" w="100%" h="30" text="Hello AWTK" />
  <dialog_client x="0" y="bottom" w="100%" h="-30">
    <label name="" x="center" y="middle:-20" w="200" h="30" text="Are you ready?"/>
    <button name="quit" x="10" y="bottom:10" w="40%" h="30" text="确定"/>
    <button name="quit" x="right:10" y="bottom:10" w="40%" h="30" text="取消"/>
  </dialog_client>
</dialog>
```

- 在 C 代码中，使用函数 dialog_create_simple 创建对话框，详见程序清单 4.4。

程序清单 4.4 在 C 代码中创建 dialog

```
//awtk\demos\demo1_app.c
static ret_t on_show_dialog(void* ctx, event_t* e) {
  ...
  widget_t* win = dialog_create_simple(NULL, 0, 0, 240, 160);
  dialog_t* dialog = DIALOG(win);
```

```
  code = dialog_modal(win);
  ...
}
```

创建之后，再创建子控件，最后调用 dialog_modal 显示对话框。对话框关闭之后 dialog_modal 才会返回。

### 4.2.3 system_bar

system_bar 窗口是一种特殊的窗口，独占 LCD 顶部区域，用来显示当前窗口的标题和关闭按钮等内容。system_bar 窗口需要在打开第一个应用程序窗口之前打开。system_bar_t 是 window_base_t 的子类控件，window_base_t 的函数均适用于 system_bar_t 控件。system_bar 对两个子控件会做特殊处理：

- 名为"title"的 label 控件，自动显示当前主窗口的 name 或 text
- 名为"close"的 button 控件，向当前主窗口发送 EVT_REQUEST_CLOSE_WINDOW 消息，如果点击该控件的话

#### 1. 函数

system_bar 提供的函数详见表 4.18。

表 4.18 system_bar 函数列表

| 函数名称 | 说明 |
| --- | --- |
| system_bar_cast | 转换为 system_bar 对象（供脚本语言使用） |
| system_bar_create | 创建 system_bar 对象 |

(1) system_bar_cast

- 函数原型

```
widget_t* system_bar_cast (widget_t* widget);
```

- 参数说明，详见表 4.19。

表 4.19 system_bar_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | system_bar 对象 |
| widget | widget_t* | system_bar 对象 |

(2) system_bar_create

- 函数原型

```
widget_t* system_bar_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.20。

表 4.20 system_bar_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | system_bar 对象。 |
| parent | widget_t* | 父控件 |

| | | |
|---|---|---|
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"system_bar"标签创建 system_bar 窗口，详见程序清单 4.5。

程序清单 4.5 在 xml 创建 system_bar

```
<!-- awtk\demos\assets\raw\ui\system_bar.xml -->
<system_bar h="30">
  <column x="0" y="0" w="-40" h="100%">
    <label style="title" x="10" y="m" w="55%" h="100%" name="title"/>
    <digit_clock style="time" x="r" y="m" w="40%" h="100%" format="hh:mm"/>
  </column>
  <button style="close" x="r:5" y="m" w="26" h="26" name="close" text="x"/>
</system_bar>
```

- 在 C 代码中，使用函数 system_bar_create 创建 system_bar 窗口，详见程序清单 4.6。

程序清单 4.6 在 C 代码中创建 system_bar

```
widget_t* win = system_bar_create(NULL, 0, 0, 320, 30);
```

创建之后，就和使用普通窗口一样了。

### 4.2.4 calibration_win

电阻屏校准窗口。calibration_win_t 是 window_base_t 的子类控件，window_base_t 的函数均适用于 calibration_win_t 控件。

1. 函数

dialog calibration_win 提供的函数详见表 4.21。

表 4.21 calibration_win 函数列表

| 函数名称 | 说明 |
|---|---|
| calibration_win_create | 创建 calibration_win 对象 |
| calibration_win_set_on_click | 设置校准点击事件的处理函数 |
| calibration_win_set_on_done | 设置校准完成的处理函数 |

(1) calibration_win_create

- 函数原型

```
widget_t* calibration_win_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.22。

表 4.22 calibration_win_create 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | 对象 |

| parent | widget_t* | 父控件 |
|---|---|---|
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(2) calibration_win_set_on_click

- 函数原型

```
ret_t calibration_win_set_on_click (widget_t* widget, calibration_win_on_click_t
on_click, void* ctx);
```

- 参数说明，详见表 4.23。

表 4.23 calibration_win_set_on_click 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| on_click | calibration_win_on_click_t | 回调函数 |
| ctx | void* | 回调函数的上下文 |

(3) calibration_win_set_on_done

- 函数原型

```
ret_t calibration_win_set_on_done (widget_t* widget, calibration_win_on_done_t on_done,
void* ctx);
```

- 参数说明，详见表 4.24。

表 4.24 calibration_win_set_on_done 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| on_done | calibration_win_on_done_t | 回调函数 |
| ctx | void* | 回调函数的上下文 |

2. 示例

- 在 xml 中，使用"calibration_win"标签创建电阻屏校准窗口，详见程序清单 4.7。

程序清单 4.7 在 xml 中创建 calibration_win

```
<!--awtk\demos\assets\raw\ui\calibration_win.xml -->
<calibration_win name="cali" w="100%" h="100%" text="Please click the center of cross">
</calibration_win>
```

- 在 C 代码中，使用 calibration_win_create 创建电阻屏校准窗口，详见程序清单 4.8。

程序清单 4.8 在 C 代码中创建 calibration_win

```
widget_t* win = calibration_win_create(NULL, 0, 0, 320, 480);
```

## 4.3 基本控件

常见的基本控件详见表 4.25。

表 4.25 基本控件列表

| 控件名称 | 说明 |
| --- | --- |
| button | 按钮控件 |
| label | 文本控件 |
| edit | 单行编辑器控件 |
| image | 图片控件 |
| image_base | 图片控件基类（一般不直接使用） |
| spin_box | 数值编辑器控件 |
| combo_box | 下拉列表控件 |
| combo_box_item | 下拉列表项控件（一般不直接使用） |
| color_tile | 色块控件 |
| dialog_title | 对话框标题控件 |
| dialog_client | 对话框客户区控件 |
| slider | 滑块控件 |
| progress_bar | 进度条控件 |

### 4.3.1 button

按钮控件。点击按钮之后会触发 EVT_CLICK 事件，注册 EVT_CLICK 事件执行特定操作。按钮控件也可以作为容器使用，使用图片和文本作为其子控件，可以实现很多有趣的效果。button_t 是 widget_t 的子类控件，widget_t 的函数均适用于 button_t 控件。

1. 函数

button 提供的函数详见表 4.26。

表 4.26 button 函数列表

| 函数名称 | 说明 |
| --- | --- |
| button_cast | 转换为 button 对象（供脚本语言使用） |
| button_create | 创建 button 对象 |
| button_set_repeat | 设置触发 EVT_CLICK 事件的时间间隔。为 0 则不重复触发 EVT_CLICK 事件 |

(1) button_cast

- 函数原型

```
widget_t* button_cast (widget_t* widget);
```

- 参数说明，详见表 4.27。

表 4.27 button_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | button 对象 |
| widget | widget_t* | button 对象 |

(2) button_create

- 函数原型

```
widget_t* button_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.28。

表 4.28 button_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) button_set_repeat

- 函数原型

```
ret_t button_set_repeat (widget_t* widget, int32_t repeat);
```

- 参数说明，详见表 4.29。

表 4.29 button_set_repeat 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| repeat | int32_t | 触发 EVT_CLICK 事件的时间间隔（毫秒） |

### 2. 属性

button 还提供了下面属性，详见表 4.30。

表 4.30 button 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| repeat | int32_t | 重复触发 EVT_CLICK 事件的时间间隔 |

### 3. 事件

button 还提供了下面事件，详见表 4.31。

表 4.31 button 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_CLICK | pointer_event_t | 点击事件 |
| EVT_LONG_PRESS | pointer_event_t | 长按事件 |

4. 示例

- 在 xml 中，使用"button"标签创建按钮控件，详见程序清单 4.9。

程序清单 4.9 在 xml 中创建 button

```
<!-- awtk\demos\assets\raw\ui\button.xml -->
<window anim_hint="htranslate">
  <button style="round" x="c" y="10" w="128" h="30" text="Text"/>
  <button style="red_btn" x="c" y="50" w="128" h="30" text="Text"/>
  <button style="icon" x="c" y="90" w="128" h="30" text="Text"/>
  ...
</window>
```

- 在 C 代码中，使用函数 button_create 创建按钮控件，详见程序清单 4.10。

程序清单 4.10 在 C 代码中创建 button

```
//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  ok = button_create(win, 10, 5, 80, 30);
  widget_set_text(ok, L"Inc");

  cancel = button_create(win, 100, 5, 80, 30);
  widget_set_text(cancel, L"Dec");
  ...
}
```

创建之后，需要用 widget_set_text 或 widget_set_text_utf8 设置文本内容。

### 4.3.2 label

文本控件。用于显示一行或多行文本。文本控件不会根据文本的长度自动换行，只有文本内容包含换行符时才会换行。如需自动换行请使用 rich_text_t 控件。label_t 是 widget_t 的子类控件，widget_t 的函数均适用于 label_t 控件。

1. 函数

label 提供的函数详见表 4.32。

表 4.32 label 函数列表

| 函数名称 | 说明 |
| --- | --- |
| label_create | 创建 label 对象 |
| label_set_length | 设置最大可显示字符个数 |

(1) label_create

- 函数原型

```
widget_t* label_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.33。

表 4.33 label_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(2) label_set_length

- 函数原型

```
ret_t label_set_length (widget_t* widget, int32_t length);
```

- 参数说明，详见表 4.34。

表 4.34 label_set_length 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| length | int32_t | 最大可显示字符个数 |

### 2. 属性

label 还提供了下面属性，详见表 4.35。

表 4.35 label 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| length | int32_t | 显示字符的长度（小余 0 时全部显示） |

### 3. 示例

- 在 xml 中，使用"label"标签创建文本控件，详见程序清单 4.11。

程序清单 4.11 在 xml 中创建 label

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  ...
  <row x="0" y="80" w="100%" h="30" children_layout="default(r=1,c=3,xm=2,s=10)">
    <label style="left" name="left" text="Left"/>
    <label style="center" name="center" text="Center"/>
    <label style="right" name="right" text="Right"/>
```

```
  </row>
  ...
</window>
```

- 在 C 代码中，使用函数 label_create 创建文本控件，详见程序清单 4.12。

程序清单 4.12 在 C 代码中创建 label

```
//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  label = label_create(win, 190, 40, 80, 30);
  widget_set_text(label, L"Right");
  widget_use_style(label, "right");
  ...
}
```

创建之后，需要用 widget_set_text 或 widget_set_text_utf8 设置文本内容。

### 4.3.3 edit

单行编辑器控件。在基于 SDL 的平台，单行编辑器控件使用平台原生的输入法，对于嵌入式平台使用内置的输入法。在使用内置的输入法时，软键盘由输入类型决定，开发者可以自定义软键盘的界面。edit_t 是 widget_t 的子类控件，widget_t 的函数均适用于 edit_t 控件。

edit_t 本身可以做为容器，放入按钮等控件。有几个特殊的子控件：

- 名为"clear"的按钮。点击时清除编辑器中的内容
- 名为"inc"的按钮。点击时增加编辑器的值，用于实现类似于 spinbox 的功能
- 名为"dec"的按钮。点击时减少编辑器的值，用于实现类似于 spinbox 的功能
- 名为"visible"的复选框。勾选时显示密码，反之不显示密码

#### 1. 函数

edit 提供的函数详见表 4.36。

表 4.36 edit 函数列表

| 函数名称 | 说明 |
| --- | --- |
| edit_cast | 转换为 edit 对象（供脚本语言使用） |
| edit_create | 创建 edit 对象 |
| edit_get_double | 获取 double 类型的值 |
| edit_get_int | 获取 int 类型的值 |
| edit_set_auto_fix | 设置编辑器是否为自动改正 |
| edit_set_double | 设置 double 类型的值 |
| edit_set_float_limit | 设置为浮点数输入及取值范围 |
| edit_set_input_tips | 设置编辑器的输入提示 |
| edit_set_input_type | 设置编辑器的输入类型 |
| edit_set_int | 设置 int 类型的值 |
| edit_set_int_limit | 设置为整数输入及取值范围 |
| edit_set_password_visible | 当编辑器输入类型为密码时，设置密码是否可见 |

| | |
|---|---|
| edit_set_readonly | 设置编辑器是否为只读 |
| edit_set_text_limit | 设置为文本输入及其长度限制，不允许输入超过 max 个字符，少于 min 个字符时进入 error 状态 |

(1) edit_cast

- 函数原型

```
widget_t* edit_cast (widget_t* widget);
```

- 参数说明，详见表 4.37。

表 4.37 edit_cast 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | edit 对象 |
| widget | widget_t* | edit 对象 |

(2) edit_create

- 函数原型

```
widget_t* edit_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.38。

表 4.38 edit_create 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) edit_get_double

- 函数原型

```
double edit_get_double (widget_t* widget);
```

- 参数说明，详见表 4.39。

表 4.39 edit_get_double 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | double | 返回 double 的值 |
| widget | widget_t* | widget 对象 |

(4) edit_get_int

- 函数原型

```
int32_t edit_get_int (widget_t* widget);
```

- 参数说明，详见表 4.40。

表 4.40 edit_get_int 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | int32_t | 返回 int 的值 |
| widget | widget_t* | widget 对象 |

(5) edit_set_auto_fix

- 函数原型

```
ret_t edit_set_auto_fix (widget_t* widget, bool_t auto_fix);
```

- 参数说明，详见表 4.41。

表 4.41 edit_set_auto_fix 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| auto_fix | bool_t | 自动改正 |

(6) edit_set_double

- 函数原型

```
ret_t edit_set_double (widget_t* widget, double value);
```

- 参数说明，详见表 4.42。

表 4.42 edit_set_double 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| value | double | 值 |

(7) edit_set_float_limit

- 函数原型

```
ret_t edit_set_float_limit (widget_t* widget, double min, double max, double step);
```

- 参数说明，详见表 4.43。

表 4.43 edit_set_float_limit 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| min | double | 最小值 |
| max | double | 最大值 |
| step | double | 步长 |

(8) edit_set_input_tips

- 函数原型

```
ret_t edit_set_input_tips (widget_t* widget, char* tips);
```

- 参数说明，详见表 4.44。

表 4.44 edit_set_input_tips 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| tips | char* | 输入提示 |

(9) edit_set_input_type

- 函数原型

```
ret_t edit_set_input_type (widget_t* widget, input_type_t type);
```

- 参数说明，详见表 4.45。

表 4.45 edit_set_input_type 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| type | input_type_t | 输入类型 |

(10) edit_set_int

- 函数原型

```
ret_t edit_set_int (widget_t* widget, int32_t value);
```

- 参数说明，详见表 4.46。

表 4.46 edit_set_int 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| value | int32_t | 值 |

(11) edit_set_int_limit

- 函数原型

```
ret_t edit_set_int_limit (widget_t* widget, int32_t min, int32_t max, int32_t step);
```

- 参数说明，详见表 4.47。

表 4.47 edit_set_int_limit 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| min | int32_t | 最小值 |

| | | |
|---|---|---|
| max | int32_t | 最大值 |
| step | int32_t | 步长 |

(12) edit_set_password_visible

- 函数原型

```
ret_t edit_set_password_visible (widget_t* widget, bool_t password_visible);
```

- 参数说明，详见表 4.48。

表 4.48 edit_set_password_visible 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| password_visible | bool_t | 密码是否可见 |

(13) edit_set_readonly

- 函数原型

```
ret_t edit_set_readonly (widget_t* widget, bool_t readonly);
```

- 参数说明，详见表 4.49。

表 4.49 edit_set_readonly 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| readonly | bool_t | 只读 |

(14) edit_set_text_limit

- 函数原型

```
ret_t edit_set_text_limit (widget_t* widget, uint32_t min, uint32_t max);
```

- 参数说明，详见表 4.50。

表 4.50 edit_set_text_limit 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | widget 对象 |
| min | uint32_t | 最小长度 |
| max | uint32_t | 最大长度 |

### 2. 属性

edit 还提供了下面属性，详见表 4.51。

表 4.51 edit 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| auto_fix | bool_t | 输入无效时，是否自动改正 |

| | | |
|---|---|---|
| bottom_margin | uint8_t | 下边距 |
| input_type | input_type_t | 输入类型 |
| left_margin | uint8_t | 左边距 |
| max | float_t | 最大值或最大长度 |
| min | float_t | 最小值或最小长度 |
| password_visible | bool_t | 密码是否可见 |
| readonly | bool_t | 编辑器是否为只读 |
| right_margin | uint8_t | 右边距 |
| step | float_t | 步长 |
| tips | char* | 输入提示 |
| top_margin | uint8_t | 上边距 |

### 3. 事件

edit 还提供了下面事件，详见表 4.52。

表 4.52 edit 事件列表

| 事件名称 | 类型 | 说明 |
|---|---|---|
| EVT_VALUE_CHANGING | event_t | 文本正在改变事件（编辑中） |
| EVT_VALUE_CHANGED | event_t | 文本改变事件 |

### 4. 示例

- 在 xml 中，使用"edit"标签创建单行编辑器控件，详见程序清单 4.13。

程序清单 4.13 在 xml 中创建 edit

```
<!-- awtk\demos\assets\raw\ui\edit.xml -->
<window anim_hint="htranslate" >
  <list_view x="0"    y="0" w="100%" h="-50" item_height="36" auto_hide_scroll_bar="true">
    <scroll_view name="view" x="0"    y="0" w="-12" h="100%">
      <list_item style="empty" children_layout="default(r=1,c=0,ym=1)">
        <label w="30%" text="Age"/>
        <edit w="70%" right_margin="16" tips="uint(0-150) auto_fix" input_type="uint" min="0" max="150"
step="1" auto_fix="true" style="number">
          <button name="inc" repeat="300" style="spinbox_up" x="right" y="0" w="15" h="50%" />
          <button name="dec" repeat="300" style="spinbox_down" x="right" y="bottom" w="15" h="50%" />
        </edit>
      </list_item>
      ...
  </list_view>
  <button name="close" x="center" y="bottom:10" w="25%" h="30" text="Close"/>
</window>
```

- 在 C 代码中，使用函数 edit_create 创建单行编辑器控件，详见程序清单 4.14。

程序清单 4.14 在 C 代码中创建 edit

```
//awtk\demos\demo_desktop.c
```

```
ret_t application_init(void) {
  widget_t* win = window_create(NULL, 0, 0, 0, 0);
  widget_t* label = label_create(win, 0, 0, 0, 0);
  widget_t* edit = edit_create(win, 0, 0, 0, 0);

  widget_set_text(label, L"hello awtk!");
  widget_set_self_layout_params(label, "0", "middle", "30%", "30");

  widget_set_self_layout_params(edit, "30%", "middle", "60%", "30");

  widget_layout(win);

  return RET_OK;
}
```

创建之后，需要用 widget_set_text 或 widget_set_text_utf8 设置文本内容。

### 4.3.4 image

用来显示一张静态图片，目前支持 bmp/png/jpg 等格式，其中：

- 如果要显示 gif 文件，请用 gif_image
- 如果要显示 svg 文件，请用 svg_image
- 如果需要支持勾选效果，请设置 selectable 属性
- 如果需要支持点击效果，请设置 clickable 属性

#### 1. 函数

image 提供的函数详见表 4.53。

表 4.53 image 函数列表

| 函数名称 | 说明 |
| --- | --- |
| image_cast | 转换为 image 对象（供脚本语言使用） |
| image_create | 创建 image 对象 |
| image_set_draw_type | 设置图片的绘制方式 |

(1) image_cast

- 函数原型

```
widget_t* image_cast (widget_t* widget);
```

- 参数说明，详见表 4.54。

表 4.54 image_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | image 对象 |
| widget | widget_t* | image 对象 |

(2) image_create

- 函数原型

```
widget_t* image_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.55。

表 4.55 image_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) image_set_draw_type

- 函数原型

```
ret_t image_set_draw_type (widget_t* widget, image_draw_type_t draw_type);
```

- 参数说明，详见表 4.56。

表 4.56 image_set_draw_type 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image 对象 |
| draw_type | image_draw_type_t | 绘制方式（仅在没有旋转和缩放时生效） |

## 2. 属性

image 还提供了下面属性，详见表 4.57。

表 4.57 image 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| draw_type | image_draw_type_t | 图片的绘制方式（仅在没有旋转和缩放时生效） |

draw_type 可取的值，详见表 4.58。

表 4.58 draw_type

| draw_type 取值 | 说明 |
| --- | --- |
| default | 缺省显示。将图片按原大小显示在目标矩形的左上角 |
| icon | 图标显示。同居中显示，但会根据屏幕密度调整大小 |
| center | 居中显示。将图片按原大小显示在目标矩形的中央 |
| scale | 缩放显示。将图片缩放至目标矩形的大小（不保证宽高成比例） |
| scale_auto | 自动缩放显示。将图片缩放至目标矩形宽度或高度（选取最小的比例），并居中显示 |
| scale_w | 宽度缩放显示。将图片缩放至目标矩形的宽度，高度按此比例进行缩放，超出部分不显示 |

| | |
|---|---|
| scale_h | 高度缩放显示。将图片缩放至目标矩形的高度，宽度按此比例进行缩放，超出部分不显示 |
| repeat | 平铺显示 |
| repeat_x | 水平方向平铺显示，垂直方向缩放 |
| repeat_y | 垂直方向平铺显示，水平方向缩放 |
| patch9 | 9 宫格显示 |
| patch3_x | 水平方向 3 宫格显示，垂直方向居中显示 |
| patch3_y | 垂直方向 3 宫格显示，水平方向居中显示 |
| patch3_x_scale_y | 水平方向 3 宫格显示，垂直方向缩放显示 |
| patch3_y_scale_x | 垂直方向 3 宫格显示，水平方向缩放显示 |

3. 示例

- 在 xml 中，使用"image"标签创建图片控件，详见程序清单 4.15。

程序清单 4.15 在 xml 中创建 image

```
<!-- awtk\demos\assets\raw\ui\images.xml -->
<window anim_hint="htranslate" >
  <list_view x="0"    y="0" w="100%" h="-50" item_height="128" auto_hide_scroll_bar="true">
    <scroll_view name="view" x="0"    y="0" w="-12" h="100%">
      <list_item style="empty" children_layout="default(r=1,c=3,s=2,m=2)">
        <image style="border" image="1" draw_type="scale_auto"><label w="100%" h="100%" text="auto"
/></image>
        <image style="border" image="2" draw_type="scale_auto"><label w="100%" h="100%" text="auto"
/></image>
        <image style="border" image="3" draw_type="scale_auto"><label w="100%" h="100%" text="auto"
/></image>
      </list_item>
      ...
</window>
```

- 在 C 代码中，使用函数 image_create 创建图片控件，详见程序清单 4.16。

程序清单 4.16 在 C 代码中创建 image

```
//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  image = image_create(win, 10, 230, 100, 100);
  image_set_image(image, "earth");
  image_set_draw_type(image, IMAGE_DRAW_ICON);
}
```

创建之后，需要用 widget_set_image 设置图片名称，用 image_set_draw_type 设置图片的绘制方式。

### 4.3.5 spin_box

spinbox 控件。一个特殊的数值编辑器，将 edit_t 和 button_t 进行组合，方便编辑数值。

点击向上的按钮将数值增加一个 step，点击向下的按钮将数值减小一个 step。step 的值可以通过 step 属性进行设置。spin_box_t 是 edit_t 的子类控件，edit_t 的函数均适用于 spin_box_t 控件。

### 1. 函数

spin_box 提供的函数详见表 4.59。

表 4.59 spin_box 函数列表

| 函数名称 | 说明 |
| --- | --- |
| spin_box_cast | 转换为 spin_box 对象（供脚本语言使用） |
| spin_box_create | 创建 spin_box 对象 |

(1) spin_box_cast

- 函数原型

```
widget_t* spin_box_cast (widget_t* widget);
```

- 参数说明，详见表 4.60。

表 4.60 spin_box_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | spin_box 对象 |
| widget | widget_t* | spin_box 对象 |

(2) spin_box_create

- 函数原型

```
widget_t* spin_box_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.61。

表 4.61 spin_box_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

### 2. 示例

- 在 xml 中，使用"spin_box"标签创建 spinbox 控件，详见程序清单 4.17。

程序清单 4.17 在 xml 中创建 spin_box

```
<!-- awtk\demos\assets\raw\ui\spinbox.xml -->
<window anim_hint="htranslate" children_layout="default(h=30,c=1,m=2,s=2)">
  <view children_layout="default(r=1,c=0)">
```

```
  <spin_box w="70%" tips="uint(0-150) auto_fix" input_type="uint" min="0" max="150" step="1"
auto_fix="true">
  </spin_box>
</window>
```

- 在 C 代码中，使用函数 spin_box_create 创建 spinbox 控件，详见程序清单 4.18。

程序清单 4.18 在 C 代码中创建 spin_box

```
widget_t* spin_box = spin_box_create(win, 10, 10, 128, 30);
edit_set_input_type(spin_box, type);
```

创建之后，可以用 edit 相关函数去设置它的各种属性。

### 4.3.6 combo_box

下拉列表控件。点击右边的按钮，可弹出一个下拉列表，从中选择一项作为当前的值。combo_box_t 是 edit_t 的子类控件，edit_t 的函数均适用于 combo_box_t 控件。

1. 函数

combo_box 提供的函数详见表 4.62。

表 4.62 combo_box 函数列表

| 函数名称 | 说明 |
| --- | --- |
| combo_box_append_option | 追加一个选项 |
| combo_box_cast | 转换 combo_box 对象（供脚本语言使用） |
| combo_box_count_options | 获取选项个数 |
| combo_box_create | 创建 combo_box 对象 |
| combo_box_get_option | 获取第 index 个选项 |
| combo_box_get_text | 获取 combo_box 的文本 |
| combo_box_get_value | 获取 combo_box 的值 |
| combo_box_reset_options | 重置所有选项 |
| combo_box_set_open_window | 点击按钮时可以打开 popup 窗口，本函数可设置窗口的名称 |
| combo_box_set_options | 设置选项 |
| combo_box_set_selected_index | 设置第 index 个选项为当前选中的选项 |

(1) combo_box_append_option

- 函数原型

```
ret_t combo_box_append_option (widget_t* widget, int32_t value, const char* text);
```

- 参数说明，详见表 4.63。

表 4.63 combo_box_append_option 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | combo_box 对象 |
| value | int32_t | 值 |
| text | const char* | 文本 |

(2) combo_box_cast

- 函数原型

```
widget_t* combo_box_cast (widget_t* widget);
```

- 参数说明，详见表 4.64。

表 4.64 combo_box_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | combo_box 对象 |
| widget | widget_t* | combo_box 对象 |

(3) combo_box_count_options

- 函数原型

```
int32_t combo_box_count_options (widget_t* widget);
```

- 参数说明，详见表 4.65。

表 4.65 combo_box_count_options 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | int32_t | 返回选项个数 |
| widget | widget_t* | combo_box 对象 |

(4) combo_box_create

- 函数原型

```
widget_t* combo_box_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.66。

表 4.66 combo_box_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(5) combo_box_get_option

- 函数原型

```
combo_box_option_t* combo_box_get_option (widget_t* widget, uint32_t index);
```

- 参数说明，详见表 4.67。

表 4.67 combo_box_get_option 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | combo_box_option_t* | 返回 index 个选项 |

| widget | widget_t* | combo_box 对象 |
| --- | --- | --- |
| index | uint32_t | 选项的索引 |

(6) combo_box_get_text

- 函数原型

```
const char* combo_box_get_text (widget_t* widget);
```

- 参数说明，详见表 4.68。

表 4.68 combo_box_get_text 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | const char* | 返回文本 |
| widget | widget_t* | combo_box 对象 |

(7) combo_box_get_value

- 函数原型

```
int32_t combo_box_get_value (widget_t* widget);
```

- 参数说明，详见表 4.69。

表 4.69 combo_box_get_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | int32_t | 返回值 |
| widget | widget_t* | combo_box 对象 |

(8) combo_box_reset_options

- 函数原型

```
ret_t combo_box_reset_options (widget_t* widget);
```

- 参数说明，详见表 4.70。

表 4.70 combo_box_reset_options 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | combo_box 对象 |

(9) combo_box_set_open_window

- 函数原型

```
ret_t combo_box_set_open_window (widget_t* widget, const char* open_window);
```

- 参数说明，详见表 4.71。

表 4.71 combo_box_set_open_window 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | combo_box 对象 |

| | | |
|---|---|---|
| open_window | const char* | 弹出窗口的名称 |

(10) combo_box_set_options

- 函数原型

```
ret_t combo_box_set_options (widget_t* widget, const char* options);
```

- 参数说明，详见表 4.72。

表 4.72 combo_box_set_options 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | combo_box 对象 |
| options | const char* | options 选项 |

(11) combo_box_set_selected_index

- 函数原型

```
ret_t combo_box_set_selected_index (widget_t* widget, uint32_t index);
```

- 参数说明，详见表 4.73。

表 4.73 combo_box_set_selected_index 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | combo_box 对象 |
| index | uint32_t | 选项的索引 |

2. 属性

combo_box 还提供了下面属性，详见表 4.74。

表 4.74 combo_box 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| open_window | char* | 为点击按钮时，要打开窗口的名称 |
| options | char* | 设置可选项（冒号分隔值和文本，分号分隔选项如:1:red;2:green;3:blue） |
| selected_index | int32_t | 当前选中的选项 |
| value | int32_t | 值 |

3. 事件

combo_box 还提供了下面事件，详见表 4.75。

表 4.75 combo_box 事件列表

| 事件名称 | 类型 | 说明 |
|---|---|---|
| EVT_VALUE_WILL_CHANGE | event_t | 值即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值改变事件 |

4. 示例

- 在 xml 中，使用"combo_box"标签创建下拉列表控件，详见程序清单 4.19。

程序清单 4.19 在 xml 中创建 combo_box

```
<!-- awtk\demos\assets\raw\ui\combo_box.xml -->
<window anim_hint="htranslate">
    <combo_box left_margin="6" readonly="true" x="10" y="5" w="200" h="30" text="left"
options="left;center;right;"/>
    <combo_box open_window="color" readonly="true" x="10" y="50" w="200" h="30" text="red"/>
    <combo_box readonly="true" x="10" y="bottom:5" w="200" h="30" tr_text="ok" options="1:ok;2:cancel;"/>
    <combo_box open_window="language" readonly="true" x="10" y="bottom:50" w="200" h="30"
tr_text="english"/>
    <label name="old_value" x="center" y="middle:-20" w="50%" h="30" text=""/>
    <label name="value" x="center" y="middle:20" w="50%" h="30" text=""/>
</window>
```

- 在 C 代码中，使用 combo_box_create 创建下拉列表控件，详见程序清单 4.20。

程序清单 4.20 在 C 代码中创建 combo_box

```
widget_t* combo_box = combo_box_create(win, 10, 10, 128, 30);
combo_box_set_options(combo_box, "left;center;right;");
combo_box_set_selected_index(combo_box, 1);
```

创建之后，用 combo_box_set_options 设置可选项目，用 combo_box_set_selected_index 设置缺省项。

### 4.3.7 color_tile

色块控件。用来显示一个颜色块，它通过属性而不是主题来设置颜色，方便在运行时动态改变颜色。可以使用 value 属性访问背景颜色的颜色值。color_tile_t 是 widget_t 的子类控件，widget_t 的函数均适用于 color_tile_t 控件。

1. 函数

color_tile 提供的函数详见表 4.76。

表 4.76 color_tile 函数列表

| 函数名称 | 说明 |
| --- | --- |
| color_tile_cast | 转换为 color_tile 对象（供脚本语言使用） |
| color_tile_create | 创建 color_tile 对象 |
| color_tile_set_bg_color | 设置背景颜色 |
| color_tile_set_border_color | 设置边框颜色 |
| color_tile_set_value | 设置背景颜色 |

(1) color_tile_cast

- 函数原型

```
widget_t* color_tile_cast (widget_t* widget);
```

- 参数说明，详见表 4.77。

表 4.77 color_tile_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | color_tile 对象 |
| widget | widget_t* | color_tile 对象 |

(2) color_tile_create

- 函数原型

```
widget_t* color_tile_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.78。

表 4.78 color_tile_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) color_tile_set_bg_color

- 函数原型

```
ret_t color_tile_set_bg_color (widget_t* widget, const char* color);
```

- 参数说明，详见表 4.79。

表 4.79 color_tile_set_bg_color 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| color | const char* | 背景颜色 |

(4) color_tile_set_border_color

- 函数原型

```
ret_t color_tile_set_border_color (widget_t* widget, const char* color);
```

- 参数说明，详见表 4.80。

表 4.80 color_tile_set_border_color 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| color | const char* | 边框颜色 |

(5) color_tile_set_value

- 函数原型

```
ret_t color_tile_set_value (widget_t* widget, color_t color);
```

- 参数说明，详见表 4.81。

表 4.81 color_tile_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| color | color_t | 背景颜色 |

2. 属性

color_tile 还提供了下面属性，详见表 4.82。

表 4.82 color_tile 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| bg_color | const char* | 背景颜色 |
| border_color | const char* | 边框颜色 |

3. 示例

- 在 xml 中，使用"color_tile"标签创建色块控件，详见程序清单 4.21。

程序清单 4.21 在 xml 中创建 color_tile

```
<!-- awtk\demos\assets\raw\ui\color_picker_full.xml -->
<window>
  <color_picker x="0" y="0" w="100%" h="100%" value="orange">
    <color_component x="0" y="0" w="200" h="200" name="sv"/>
    <color_component x="210" y="0" w="20" h="200" name="h"/>
    <color_tile x="0" y="210" w="50%" h="20" name="new" bg_color="green"/>
    <color_tile x="right" y="210" w="50%" h="20" name="old" bg_color="blue"/>
    ...
</window>
```

- 在 C 代码中，使用函数 color_tile_create 创建色块控件，详见程序清单 4.22。

程序清单 4.22 在 C 代码中创建 color_tile

```
widget_t* color_tile = color_tile_create(win, 10, 10, 128, 30);
color_tile_set_bg_color(color_tile, "red");
```

创建之后，用 color_tile_set_bg_color 设置背景颜色。

### 4.3.8 dialog_title

对话框标题控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。子控件的布局可用 layout_children 属性指定。请参考布局参数。dialog_title_t 是 widget_t 的子类控件，widget_t 的函数均适用于 dialog_title_t 控件。

### 1. 函数

dialog_title 提供的函数详见表 4.83。

表 4.83 dialog_title 函数列表

| 函数名称 | 说明 |
| --- | --- |
| dialog_title_cast | 转换为 dialog_title 对象（供脚本语言使用） |
| dialog_title_create | 创建 dialog_title 对象 |

(1) dialog_title_cast

- 函数原型

```
widget_t* dialog_title_cast (widget_t* widget);
```

- 参数说明，详见表 4.84。

表 4.84 dialog_title_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | dialog_title 对象 |
| widget | widget_t* | dialog_title 对象 |

(2) dialog_title_create

- 函数原型

```
widget_t* dialog_title_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.85。

表 4.85 dialog_title_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

### 2. 示例

- 在 xml 中，使用"dialog_title"标签创建对话框标题控件，详见程序清单 4.23。

程序清单 4.23 在 xml 中创建 dialog_title

```
<!-- awtk\demos\assets\raw\ui\dialog1.xml -->
<dialog anim_hint="center_scale(duration=300)" x="c" y="m" w="80%" h="160" text="Dialog">
  <dialog_title x="0" y="0" w="100%" h="30" text="Hello AWTK" />
  <dialog_client x="0" y="bottom" w="100%" h="-30">
    <label name="" x="center" y="middle:-20" w="200" h="30" text="Are you ready?"/>
    <button name="quit" x="10" y="bottom:10" w="40%" h="30" text="确定"/>
    <button name="quit" x="right:10" y="bottom:10" w="40%" h="30" text="取消"/>
```

```
  </dialog_client>
</dialog>
```

- 在 C 代码中，使用 dialog_create_simple()创建对话框标题控件，详见程序清单 4.24。

程序清单 4.24 在 C 代码中创建 dialog_title

```
//awtk\demos\demo1_app.c
static ret_t on_show_dialog(void* ctx, event_t* e) {
  ...
  widget_t* win = dialog_create_simple(NULL, 0, 0, 240, 160);
  dialog_t* dialog = DIALOG(win);

  code = dialog_modal(win);
  ...
}
```

### 4.3.9 dialog_client

对话框客户区控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。 请参考布局参数。dialog_client_t 是 widget_t 的子类控件，widget_t 的函数均适用于 dialog_client_t 控件。

1. 函数

dialog_client 提供的函数详见表 4.86。

表 4.86 dialog_client 函数列表

| 函数名称 | 说明 |
| --- | --- |
| dialog_client_cast | 转换为 dialog_client 对象（供脚本语言使用） |
| dialog_client_create | 创建 dialog_client 对象 |

(1) dialog_client_cast

- 函数原型

```
widget_t* dialog_client_cast (widget_t* widget);
```

- 参数说明，详见表 4.87。

表 4.87 dialog_client_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | dialog_client 对象 |
| widget | widget_t* | dialog_client 对象 |

(2) dialog_client_create

- 函数原型

```
widget_t* dialog_client_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.88。

表 4.88 dialog_client_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"dialog_client"标签创建对话框客户区控件，详见程序清单 4.25。

程序清单 4.25 在 xml 中创建 dialog_client

```
<!-- awtk\demos\assets\raw\ui\dialog1.xml -->
<dialog anim_hint="center_scale(duration=300)" x="c" y="m" w="80%" h="160" text="Dialog">
  <dialog_title x="0" y="0" w="100%" h="30" text="Hello AWTK" />
  <dialog_client x="0" y="bottom" w="100%" h="-30">
    <label name="" x="center" y="middle:-20" w="200" h="30" text="Are you ready?"/>
    <button name="quit" x="10" y="bottom:10" w="40%" h="30" text="确定"/>
    <button name="quit" x="right:10" y="bottom:10" w="40%" h="30" text="取消"/>
  </dialog_client>
</dialog>
```

- 在 C 代码中，使用 dialog_create_simple()创建对话框客户区控件，详见程序清单 4.26。

程序清单 4.26 在 C 代码中创建 dialog_client

```
//awtk\demos\demo1_app.c
static ret_t on_show_dialog(void* ctx, event_t* e) {
  ...
  widget_t* win = dialog_create_simple(NULL, 0, 0, 240, 160);
  dialog_t* dialog = DIALOG(win);

  code = dialog_modal(win);
  ...
}
```

### 4.3.10 slider

滑块控件。slider_t 是 widget_t 的子类控件，widget_t 的函数均适用于 slider_t 控件。

1. 函数

slider 提供的函数详见表 4.89。

表 4.89 slider 函数列表

| 函数名称 | 说明 |
| --- | --- |
| slider_cast | 转换为 slider 对象（供脚本语言使用） |

| | |
|---|---|
| slider_create | 创建 slider 对象 |
| slider_set_max | 设置滑块的最大值 |
| slider_set_min | 设置滑块的最小值 |
| slider_set_step | 设置滑块的拖动的最小单位 |
| slider_set_value | 设置滑块的值 |
| slider_set_vertical | 设置滑块的方向 |

(1) slider_cast

- 函数原型

```
widget_t* slider_cast (widget_t* widget);
```

- 参数说明，详见表 4.90。

表 4.90 slider_cast 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | slider 对象 |
| widget | widget_t* | slider 对象 |

(2) slider_create

- 函数原型

```
widget_t* slider_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.91。

表 4.91 slider_create 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) slider_set_max

- 函数原型

```
ret_t slider_set_max (widget_t* widget, uint16_t max);
```

- 参数说明，详见表 4.92。

表 4.92 slider_set_max 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| max | uint16_t | 最大值 |

(4) slider_set_min

- 函数原型

```
ret_t slider_set_min (widget_t* widget, uint16_t min);
```

- 参数说明，详见表 4.93。

表 4.93 slider_set_min 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| min | uint16_t | 最小值 |

(5) slider_set_step

- 函数原型

```
ret_t slider_set_step (widget_t* widget, uint16_t step);
```

- 参数说明，详见表 4.94。

表 4.94 slider_set_step 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| step | uint16_t | 拖动的最小单位 |

(6) slider_set_value

- 函数原型

```
ret_t slider_set_value (widget_t* widget, uint16_t value);
```

- 参数说明，详见表 4.95。

表 4.95 slider_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| value | uint16_t | 值 |

(7) slider_set_vertical

- 函数原型

```
ret_t slider_set_vertical (widget_t* widget, bool_t vertical);
```

- 参数说明，详见表 4.96。

表 4.96 slider_set_vertical 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败。 |
| widget | widget_t* | 控件对象。 |

| | | |
|---|---|---|
| vertical | bool_t | 是否为垂直方向。 |

2. 属性

slider 还提供了下面属性，详见表 4.97。

表 4.97 slider 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| max | uint16_t | 最大值 |
| min | uint16_t | 最小值 |
| step | uint16_t | 拖动的最小单位 |
| value | uint16_t | 值 |
| vertical | bool_t | 滑块的是否为垂直方向 |

3. 事件

slider 还提供了下面事件，详见表 4.98。

表 4.98 slider 事件列表

| 事件名称 | 类型 | 说明 |
|---|---|---|
| EVT_VALUE_WILL_CHANGE | event_t | 值即将改变事件 |
| EVT_VALUE_CHANGING | event_t | 值正在改变事件（拖动中） |
| EVT_VALUE_CHANGED | event_t | 值改变事件 |

4. 示例

- 在 xml 中，使用"slider"标签创建滑块控件，详见程序清单 4.27。

程序清单 4.27 在 xml 中创建 slider

```
<!-- awtk\demos\assets\raw\ui\color_picker_full.xml -->
<window>
    ...
    <view >
     <label x="0" y="middle" w="30" h="100%" text="G"/>
     <spin_box x="40" y="middle" w="54" h="100%" min="0" input_type="int" max="255" step="1"
auto_fix="true" name="g" />
     <slider x="98" y="middle" w="-100" h="100%" name="g" />
    </view>
    ...
</window>
```

- 在 C 代码中，使用函数 slider_create 创建滑块控件，详见程序清单 4.28。

程序清单 4.28 在 C 代码中创建 slider

```
widget_t* slider = slider_create(win, 10, 10, 200, 30);
widget_on(slider, EVT_VALUE_CHANGED, on_changed, NULL);
widget_on(slider, EVT_VALUE_CHANGING, on_changing, NULL);
```

### 4.3.11 progress_bar

进度条控件。进度条控件可以水平显示也可以垂直显示，由 vertical 属性决定。progress_bar_t 是 widget_t 的子类控件，widget_t 的函数均适用于 progress_bar_t 控件。

1. 函数

progress_bar 提供的函数详见表 4.99。

表 4.99 progress_bar 函数列表

| 函数名称 | 说明 |
| --- | --- |
| progress_bar_cast | 转换为 progress_bar 对象（供脚本语言使用） |
| progress_bar_create | 创建 progress_bar 对象 |
| progress_bar_set_show_text | 设置进度条的是否显示文本 |
| progress_bar_set_value | 设置进度条的进度 |
| progress_bar_set_vertical | 设置进度条的方向 |

(1) progress_bar_cast

- 函数原型

```
widget_t* progress_bar_cast (widget_t* widget);
```

- 参数说明，详见表 4.100。

表 4.100 progress_bar_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | progress_bar 对象 |
| widget | widget_t* | progress_bar 对象 |

(2) progress_bar_create

- 函数原型

```
widget_t* progress_bar_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.101。

表 4.101 progress_bar_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) progress_bar_set_show_text

- 函数原型

```
ret_t progress_bar_set_show_text (widget_t* widget, bool_t show_text);
```

- 参数说明，详见表 4.102。

表 4.102 progress_bar_set_show_text 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| show_text | bool_t | 是否显示文本 |

(4) progress_bar_set_value

- 函数原型

```
ret_t progress_bar_set_value (widget_t* widget, uint8_t value);
```

- 参数说明，详见表 4.103。

表 4.103 progress_bar_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| value | uint8_t | 进度 |

(5) progress_bar_set_vertical

- 函数原型

```
ret_t progress_bar_set_vertical (widget_t* widget, bool_t vertical);
```

- 参数说明，详见表 4.104。

表 4.104 progress_bar_set_vertical 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| vertical | bool_t | 是否为垂直方向 |

### 2. 属性

progress_bar 还提供了下面属性，详见表 4.105。

表 4.105 progress_bar 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| show_text | bool_t | 是否显示文本 |
| value | uint8_t | 进度条的值[0-100] |
| vertical | bool_t | 进度条的是否为垂直方向 |

### 3. 事件

progress_bar 还提供了下面事件，详见表 4.106。

表 4.106 progress_bar 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值改变事件 |

4. 示例

- 在 xml 中，使用"progress_bar"标签创建进度条控件，详见程序清单 4.29。

程序清单 4.29 在 xml 中创建 progress_bar

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  ...
  <progress_bar name="bar1" x="10" y="142" w="240" h="16" value="40"/>
  <progress_bar name="bar2" x="280" y="128" w="30" h="118" value="20" vertical="true"/>
  ...
</window>
```

- 在 C 代码中，使用函数 progress_bar_create 创建进度条控件，详见程序清单 4.30。

程序清单 4.30 在 C 代码中创建 progress_bar

```
//awtk\demos\demo1_app.c
ret_t application_init() {
  ...
  progress_bar = progress_bar_create(win, 260, 80, 20, 118);
  widget_set_value(progress_bar, 40);
  progress_bar_set_vertical(progress_bar, TRUE);
  ...
}
```

### 4.3.12 tab_control

标签控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。标签控件通常会包含一个 pages 控件和一个 tab_button_group 控件，它们之间的关系详见图 4.3。tab_control_t 是 widget_t 的子类控件，widget_t 的函数均适用于 tab_control_t 控件。

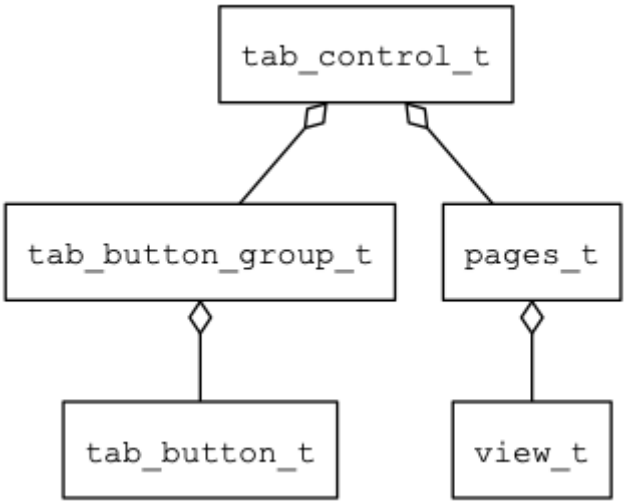



图 4.3 tab_control

### 1. 函数

tab_control 提供的函数详见表 4. 107。

表 4.107 tab_control 函数列表

| 函数名称 | 说明 |
| --- | --- |
| tab_control_cast | 转换为 tab_control 对象（供脚本语言使用） |
| tab_control_create | 创建 tab_control 对象 |

(1) tab_control_cast

- 函数原型

```
widget_t* tab_control_cast (widget_t* widget);
```

- 参数说明，详见表 4.108。

表 4.108 tab_control_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | tab_control 对象 |
| widget | widget_t* | tab_control 对象 |

(2) tab_control_create

- 函数原型

```
widget_t* tab_control_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.109。

表 4.109 tab_control_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |

| | | |
|---|---|---|
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"tab_control"标签创建标签控件，详见程序清单 4.31。

程序清单 4.31 在 xml 中创建 tab_control

```
<!-- awtk\demos\assets\raw\ui\tab_bottom.xml -->
<window anim_hint="htranslate" theme="tab_bottom">
  <tab_control x="0" y="0" w="100%" h="100%" >
    <pages x="0" y="0" w="100%" h="-60" style="at_top">
      <?include filename="tab_views.inc" ?>
    </pages>
    <tab_button_group x="0" y="bottom" w="100%" h="60" >
      <tab_button icon="msg" active_icon="msg_active" text="Message"/>
      <tab_button icon="contact" active_icon="contact_active" text="Contact" value="true" />
      <tab_button icon="discovery" active_icon="discovery_active" text="Discovery" />
      <tab_button icon="me" active_icon="me_active" text="Me" />
    </tab_button_group>
  </tab_control>
</window>
```

### 4.3.13 tab_button

标签按钮控件。标签按钮有点类似单选按钮，但点击标签按钮之后会自动切换当前的标签页。tab_button_t 是 widget_t 的子类控件，widget_t 的函数均适用于 tab_button_t 控件。

1. 函数

tab_button 提供的函数详见表 4.110。

表 4.110 tab_button 函数列表

| 函数名称 | 说明 |
|---|---|
| tab_button_cast | 转换 tab_button 对象（供脚本语言使用） |
| tab_button_create | 创建 tab_button 对象 |
| tab_button_set_active_icon | 设置控件的 active 图标 |
| tab_button_set_icon | 设置控件的图标 |
| tab_button_set_value | 设置控件的值 |

(1) tab_button_cast

- 函数原型

```
widget_t* tab_button_cast (widget_t* widget);
```

- 参数说明，详见表 4.111。

表 4.111 tab_button_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | tab_button 对象 |
| widget | widget_t* | tab_button 对象 |

(2) tab_button_create

- 函数原型

```
widget_t* tab_button_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.112。

表 4.112 tab_button_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) tab_button_set_active_icon

- 函数原型

```
ret_t tab_button_set_active_icon (widget_t* widget, char* name);
```

- 参数说明，详见表 4.113。

表 4.113 tab_button_set_active_icon 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | tab_button 对象 |
| name | char* | 当前项的图标 |

(4) tab_button_set_icon

- 函数原型

```
ret_t tab_button_set_icon (widget_t* widget, char* name);
```

- 参数说明，详见表 4.114。

表 4.114 tab_button_set_icon 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | tab_button 对象 |
| name | char* | 当前项的图标 |

(5) tab_button_set_value

- 函数原型

```
ret_t tab_button_set_value (widget_t* widget, uint32_t value);
```

- 参数说明，详见表 4.115。

表 4.115 tab_button_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | tab_button 对象 |
| value | uint32_t | 值 |

### 2. 属性

tab_button 还提供了下面属性，详见表 4.116。

表 4.116 tab_button 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| active_icon | char* | 当前项的图标的名称 |
| icon | char* | 非当前项的图标的名称 |
| value | bool_t | 值 |

### 3. 事件

tab_button 还提供了下面事件，详见表 4.117。

表 4.117 tab_button 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值（激活状态）即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值（激活状态）改变事件 |

### 4. 示例

- 在 xml 中，使用"tab_button"标签创建标签控件，详见程序清单 4.32。

程序清单 4.32 在 xml 中创建 tab_button

```
<!-- awtk\demos\assets\raw\ui\tab_bottom_compact.xml -->
<window anim_hint="htranslate" theme="tab_bottom_compact">
  <tab_control x="0" y="0" w="100%" h="100%" >
    <pages x="c" y="20" w="90%" h="-60" value="1">
      <view w="100%" h="100%">
        <label x="c" y="m" w="100%" h="60" text="General" />
        <button name="close" x="c" y="bottom:100" w="80" h="40" text="Close" />
      </view>
      <view w="100%" h="100%">
        <label x="c" y="m" w="100%" h="60" text="Network" />
        <button name="close" x="c" y="bottom:100" w="80" h="40" text="Close" />
      </view>
      <view w="100%" h="100%">
```

```
        <label x="c" y="m" w="100%" h="60" text="Security" />
        <button name="close" x="c" y="bottom:100" w="80" h="40" text="Close" />
      </view>
    </pages>
    <tab_button_group x="c" y="bottom:10" w="90%" h="30" compact="true" >
      <tab_button text="General"/>
      <tab_button text="Network" value="true" />
      <tab_button text="Security"/>
    </tab_button_group>
  </tab_control>
</window>
```

### 4.3.14 tab_button_group

标签按钮分组控件。一个简单的容器，主要用于对标签按钮进行布局和管理。tab_button_group_t 是 widget_t 的子类控件， widget_t 的函数均适用于 tab_button_group_t 控件。

1. 函数

tab_button_group 提供的函数详见表 4.118。

表 4.118 tab_button_group 函数列表

| 函数名称 | 说明 |
| --- | --- |
| tab_button_group_cast | 转换为 tab_button_group 对象（供脚本语言使用） |
| tab_button_group_create | 创建 tab_button_group 对象 |

(1) tab_button_group_cast

- 函数原型

```
widget_t* tab_button_group_cast (widget_t* widget);
```

- 参数说明，详见表 4.119。

表 4.119 tab_button_group_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | tab_button_group 对象 |
| widget | widget_t* | tab_button_group 对象 |

(2) tab_button_group_create

- 函数原型

```
widget_t* tab_button_group_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.120。

表 4.120 tab_button_group_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |

| | | |
|---|---|---|
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 属性

tab_button_group 还提供了下面属性，详见表 4.121。

表 4.121 tab_button_group 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| compact | bool_t | 紧凑型排版子控件 |

## 4.4 通用容器控件

常用的通用容器控件详见表 4.122。

表 4.122 通用容器控件列表

| 控件名称 | 说明 |
|---|---|
| row | 行控件 |
| column | 列控件 |
| grid | 网格控件 |
| view | 通用容器控件 |
| grid_item | 网格项控件 |
| group_box | 通用分组控件 |
| app_bar | app_bar 控件 |
| button_group | 按钮分组控件 |

### 4.4.1 row

row。一个简单的容器控件，用于水平排列其子控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。 请参考布局参数。row_t 是 widget_t 的子类控件，widget_t 的函数均适用于 row_t 控件。

1. 函数

row 提供的函数详见表 4.123。

表 4.123 row 函数列表

| 函数名称 | 说明 |
|---|---|
| row_cast | 转换为 row 对象（供脚本语言使用） |
| row_create | 创建 row 对象 |

(1) row_cast

- 函数原型

```
widget_t* row_cast (widget_t* widget);
```

- 参数说明，详见表 4.124。

表 4.124 row_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | row 对象 |
| widget | widget_t* | row 对象 |

(2) row_create

- 函数原型

```
widget_t* row_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.125。

表 4.125 row_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"row"标签创建行容器，详见程序清单 4.33。

程序清单 4.33 在 xml 中创建 row

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  ...
  <row x="0" y="40" w="100%" h="30" children_layout="default(r=1,c=3,xm=2,s=10)">
    <button name="inc_value" text="Inc"/>
    <button name="close" text="Close"/>
    <button name="dec_value" text="Dec"/>
  </row>
  ...
</window>
```

### 4.4.2 column

column。一个简单的容器控件，垂直排列其子控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。 请参考布局参数。column_t 是 widget_t 的子类控件，widget_t 的函数均适用于 column_t 控件。

1. 函数

column 提供的函数详见表 4.126。

表 4.126 column 函数列表

| 函数名称 | 说明 |
| --- | --- |
| column_cast | 转换为 column 对象（供脚本语言使用） |
| column_create | 创建 column 对象 |

(1) column_cast

- 函数原型

```
widget_t* column_cast (widget_t* widget);
```

- 参数说明，详见表 4.127。

表 4.127 column_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | column 对象 |
| widget | widget_t* | column 对象 |

(2) column_create

- 函数原型

```
widget_t* column_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.128。

表 4.128 column_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"column"标签创建列容器，详见程序清单 4.34。

程序清单 4.34 在 xml 中创建 column

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  ...
  <column x="20" y="230" w="50%" h="90" children_layout="default(r=3,c=1,ym=2,s=10)" >
    <radio_button name="r1" text="Book"/>
    <radio_button name="r2" text="Food"/>
    <radio_button name="r3" text="Pencil" value="true"/>
  </column>
  ...
</window>
```

### 4.4.3 grid

grid 控件。一个简单的容器控件，用于网格排列一组控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。子控件的布局可用 layout_children 属性指定。请参考布局参数。grid_t 是 widget_t 的子类控件，widget_t 的函数均适用于 grid_t 控件。

1. 函数

grid 提供的函数详见表 4.129。

表 4.129 grid 函数列表

| 函数名称 | 说明 |
| --- | --- |
| grid_cast | 转换为 grid 对象（供脚本语言使用） |
| grid_create | 创建 grid 对象 |

(1) grid_cast

- 函数原型

```
widget_t* grid_cast (widget_t* widget);
```

- 参数说明，详见表 4.130。

表 4.130 grid_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | grid 对象 |
| widget | widget_t* | grid 对象 |

(2) grid_create

- 函数原型

```
widget_t* grid_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.131。

表 4.131 grid_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"grid"标签创建网格容器，详见程序清单 4.35。

程序清单 4.35 在 xml 中创建 grid

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
```

```
  ...
  <grid x="20" y="bottom:10" w="80%" h="40" children_layout="default(r=1,c=5,x=2,s=10)">
    <image draw_type="icon" image="earth"/>
    <image draw_type="icon" image="rgba" />
    <image draw_type="icon" image="rgb" />
    <image draw_type="icon" image="message"/>
    <image draw_type="icon" image="red_btn_n"/>
  </grid>
</window>
```

### 4.4.4 view

一个通用的容器控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。 请参考布局参数。view_t 是 widget_t 的子类控件，widget_t 的函数均适用于 view_t 控件。

1. 函数

view 提供的函数详见表 4.132。

表 4.132 view 函数列表

| 函数名称 | 说明 |
| --- | --- |
| view_cast | 转换为 view 对象（供脚本语言使用） |
| view_create | 创建 view 对象 |

(1) view_cast

- 函数原型

```
widget_t* view_cast (widget_t* widget);
```

- 参数说明，详见表 4.133。

表 4.133 view_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | view 对象 |
| widget | widget_t* | view 对象 |

(2) view_create

- 函数原型

```
widget_t* view_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.134。

表 4.134 view_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |

| | | |
|---|---|---|
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"view"标签创建 view 容器，详见程序清单 4.36。

程序清单 4.36 在 xml 中创建 view

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  ...
  <view x="0" y="40" w="100%" h="30" children_layout="default(r=1,c=3,xm=2,s=10)">
    <button name="inc_value" text="Inc"/>
    <button name="close" text="Close"/>
    <button name="dec_value" text="Dec"/>
  </view>
  ...
</window>
```

### 4.4.5 grid_item

grid_item。一个简单的容器控件，一般作为 grid 的子控件。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。子控件的布局可用 layout_children 属性指定。请参考布局参数。grid_item_t 是 widget_t 的子类控件，widget_t 的函数均适用于 grid_item_t 控件。

1. 函数

grid_item 提供的函数详见表 4.135。

表 4.135 grid_item 函数列表

| 函数名称 | 说明 |
|---|---|
| grid_item_cast | 转换为 grid_item 对象（供脚本语言使用） |
| grid_item_create | 创建 grid_item 对象 |

(1) grid_item_cast

- 函数原型

```
widget_t* grid_item_cast (widget_t* widget);
```

- 参数说明，详见表 4.136。

表 4.136 grid_item_cast 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | widget_t* | grid_item 对象 |
| widget | widget_t* | grid_item 对象 |

(2) grid_item_create

- 函数原型

```
widget_t* grid_item_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.137。

表 4.137 grid_item_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"grid_item"标签创建 grid_item 容器，详见程序清单 4.37。

程序清单 4.37 在 xml 中创建 grid_item

```
<grid x="0" y="0" w="100%" h="100%" children_layout="default(c=2,r=2,m=5,s=5)">
    <grid_item>
        <button x="c" y="m" w="80%" h="30" name="0" text="0"/>
    </grid_item>
    <grid_item>
        <button x="c" y="m" w="80%" h="30" name="1" text="1"/>
    </grid_item>
    <grid_item>
        <button x="c" y="m" w="80%" h="30" name="2" text="2"/>
    </grid_item>
    <grid_item>
        <button x="c" y="m" w="80%" h="30" name="3" text="3"/>
    </grid_item>
</grid>
```

### 4.4.6 group_box

分组控件。单选按钮在同一个父控件中是互斥的，所以通常将相关的单选按钮放在一个 group_box 中。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。请参考布局参数。group_box_t 是 widget_t 的子类控件，widget_t 的函数均适用于 group_box_t 控件。

1. 函数

group_box 提供的函数详见表 4.138。

表 4.138 group_box 函数列表

| 函数名称 | 说明 |
| --- | --- |
| group_box_cast | 转换为 group_box 对象（供脚本语言使用） |
| group_box_create | 创建 group_box 对象 |

(1) group_box_cast

- 函数原型

```
widget_t* group_box_cast (widget_t* widget);
```

- 参数说明，详见表 4.139。

表 4.139 group_box_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | group_box 对象 |
| widget | widget_t* | group_box 对象 |

(2) group_box_create

- 函数原型

```
widget_t* group_box_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.140。

表 4.140 group_box_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"group_box"标签创建分组容器，详见程序清单 4.38。

程序清单 4.38 在 xml 中创建 group_box

```
<group_box x="20" y="230" w="50%" h="90" children_layout="default(r=3,c=1,ym=2,s=10)">
    <radio_button name="r1" text="Book"/>
    <radio_button name="r2" text="Food"/>
    <radio_button name="r3" text="Pencil" value="true"/>
</group_box>
```

### 4.4.7 app_bar

app_bar 控件。一个简单的容器控件，一般在窗口的顶部，用于显示本窗口的状态和信息。它本身不提供布局功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。请参考布局参数。app_bar_t 是 widget_t 的子类控件，widget_t 的函数均适用于 app_bar_t 控件。

1. 函数

app_bar 提供的函数详见表 4.141。

表 4.141 app_bar 函数列表

| 函数名称 | 说明 |
| --- | --- |
| app_bar_cast | 转换为 app_bar 对象（供脚本语言使用） |
| app_bar_create | 创建 app_bar 对象 |

(1) app_bar_cast

- 函数原型

```
widget_t* app_bar_cast (widget_t* widget);
```

- 参数说明，详见表 4.142。

表 4.142 app_bar_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | app_bar 对象 |
| widget | widget_t* | app_bar 对象 |

(2) app_bar_create

- 函数原型

```
widget_t* app_bar_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.143。

表 4.143 app_bar_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"app_bar"标签创建 app_bar 容器，详见程序清单 4.39。

程序清单 4.39 在 xml 中创建 app_bar

```
<!-- awtk\demos\assets\raw\ui\basic.xml -->
<window anim_hint="htranslate">
  <app_bar x="0" y="0" w="100%" h="30" >
    <label x="0" y="0" w="100%" h="100%" text="Basic Controls" />
  </app_bar>
  ...
</window>
```

### 4.4.8 button_group

button group 控件。一个简单的容器控件，用于容纳一组按钮控件。它本身不提供布局

功能，仅提供具有语义的标签，让 xml 更具有可读性。 子控件的布局可用 layout_children 属性指定。请参考布局参数。button_group_t 是 widget_t 的子类控件，widget_t 的函数均适用于 button_group_t 控件。

### 1. 函数

button_group 提供的函数详见表 4.144。

表 4.144 button_group 函数列表

| 函数名称 | 说明 |
| --- | --- |
| button_group_cast | 转换为 button_group 对象（供脚本语言使用） |
| button_group_create | 创建 button_group 对象 |

(1) button_group_cast

- 函数原型

```
widget_t* button_group_cast (widget_t* widget);
```

- 参数说明，详见表 4.145。

表 4.145 button_group_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | button_group 对象 |
| widget | widget_t* | button_group 对象 |

(2) button_group_create

- 函数原型

```
widget_t* button_group_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.146。

表 4.146 button_group_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

### 2. 示例

- 在 xml 中，使用"button_group"标签创建 button_group 容器，详见程序清单 4.40。

程序清单 4.40 在 xml 中创建 button_group

```
<button_group x="0" y="m" w="100%" h="40" children_layout="default(c=4,r=1,s=5,m=5)">
    <button name="open:basic" text="Basic"/>
    <button name="open:button" text="Buttons"/>
    <button name="open:edit" text="Edits"/>
```

```
    <button name="open:keyboard" text="KeyBoard"/>
</button_group>
```

## 4.5 扩展控件

在 AWTK 中提供了两种类型的控件：内置控件和扩展控件，其中内置控件包括窗口、基本控件和通用容器。在应用程序中如果需要使用扩展控件，需要调用 tk_ext_widgets_init() 函数初始化，代码详见程序清单 4.41。 常见的扩展控件详见表 4.147。

程序清单 4.41 tk_ext_widgets_init()

```
//awtk\demos\demo_ui_app.c
ret_t application_init() {
  tk_ext_widgets_init();

  return show_preload_res_window();
}
```

表 4.147 扩展控件列表

| 控件名称 | 说明 |
| --- | --- |
| canvas_widget | 画布控件 |
| color_picker | 颜色选择器控件 |
| gif_image | gif_image 控件 |
| guage | 仪表控件 |
| guage_pointer | 仪表指针控件 |
| image_animation | 图片动画控件 |
| image_value | 图片值控件 |
| keyboard | 软键盘控件 |
| progress_circle | 进度圆环控件 |
| rich_text | 图文混排控件 |
| slide_menu | 左右滑动菜单控件 |
| slide_view | 滑动视图控件 |
| svg_image | SVG 图片控件 |
| switch | 开关控件 |
| text_selector | 文本选择器控件 |
| time_clock | 模拟时钟控件 |
| digit_clock | 数字时钟控件 |

### 4.5.1 canvas_widget

画布控件。画布控件让开发者可以自己在控件上绘制需要的内容。canvas_widget_t 是 widget_t 的子类控件，widget_t 的函数均适用于 canvas_widget_t 控件。

1. 函数

canvas_widget 提供的函数详见表 4.148。

表 4.148 canvas_widget 函数列表

| 函数名称 | 说明 |
| --- | --- |
| canvas_widget_cast | 转换为 canvas_widget 对象（供脚本语言使用） |
| canvas_widget_create | 创建 canvas_widget 对象 |

(1) canvas_widget_cast

- 函数原型

```
widget_t* canvas_widget_cast (widget_t* widget);
```

- 参数说明，详见表 4.149。

表 4.149 canvas_widget_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | canvas_widget 对象 |
| widget | widget_t* | canvas_widget 对象 |

(2) canvas_widget_create

- 函数原型

```
widget_t* canvas_widget_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.150。

表 4.150 canvas_widget_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"canvas_widget"标签创建画布控件，详见程序清单 4.42。

程序清单 4.42 在 xml 中创建 canvas_widget

```
<!-- awtk\demos\assets\raw\ui\vgcanvas.xml -->
<window anim_hint="htranslate">
  <canvas name="paint_vgcanvas" x="0" y="0" w="100%" h="100%" />
</window>
```

- 在 C 代码中，使用函数 canvas_widget_create 创建画布控件，详见程序清单 4.43。

程序清单 4.43 在 C 代码中创建 canvas_widget

```
//awtk\demos\demo_vg_app.c
ret_t application_init() {
```

```
  widget_t* win = window_create(NULL, 0, 0, 0, 0);
  widget_t* canvas = canvas_widget_create(win, 0, 0, win->w, win->h);

  // widget_on(canvas, EVT_PAINT, on_paint_vg_simple, NULL);
  widget_on(canvas, EVT_PAINT, on_paint_vg, NULL);

  timer_add(on_timer, win, 500);

  return RET_OK;
}
```

创建之后，需要用 widget_on 注册 EVT_PAINT 事件，并在 EVT_PAINT 事件处理函数中绘制。

### 4.5.2 color_picker

颜色选择器。color_picker_t 是 widget_t 的子类控件，widget_t 函数均适用于 color_picker_t 控件。

#### 1. 函数

color_picker 提供的函数详见表 4.151。

表 4.151 color_picker 函数列表

| 函数名称 | 说明 |
| --- | --- |
| color_picker_cast | 转换 color_picker 对象（供脚本语言使用） |
| color_picker_create | 创建 color_picker 对象 |
| color_picker_set_color | 设置颜色 |

(1) color_picker_cast

- 函数原型

```
widget_t* color_picker_cast (widget_t* widget);
```

- 参数说明，详见表 4.152。

表 4.152 color_picker_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | color_picker 对象 |
| widget | widget_t* | color_picker 对象 |

(2) color_picker_create

- 函数原型

```
widget_t* color_picker_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.153。

表 4.153 color_picker_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |

| parent | widget_t* | 父控件 |
|---|---|---|
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) color_picker_set_color

- 函数原型

```
ret_t color_picker_set_color (widget_t* widget, const char* color);
```

- 参数说明，详见表 4.154。

表 4.154 color_picker_set_color 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| color | const char* | 颜色 |

### 2. 属性

color_picker 还提供了下面属性，详见表 4.155。

表 4.155 color_picker 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| value | const char* | 颜色 |

### 3. 事件

color_picker 还提供了下面事件，详见表 4.156。

表 4.156 color_picker 事件列表

| 事件名称 | 类型 | 说明 |
|---|---|---|
| EVT_VALUE_WILL_CHANGE | event_t | 值（颜色）即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值（颜色）改变事件 |

### 4. 示例

- 在 xml 中，使用"color_picker"标签创建颜色选择器控件，详见程序清单 4.44。

程序清单 4.44 在 xml 中创建 color_picker

```
<!-- awtk\demos\assets\raw\ui\color_picker_full.xml -->
<window>
  <color_picker x="0" y="0" w="100%" h="100%" value="orange">
    <color_component x="0" y="0" w="200" h="200" name="sv"/>
    <color_component x="210" y="0" w="20" h="200" name="h"/>
    <color_tile x="0" y="210" w="50%" h="20" name="new" bg_color="green"/>
    <color_tile x="right" y="210" w="50%" h="20" name="old" bg_color="blue"/>
    ...
```

```
  </color_picker>
  <button name="close" x="center" y="bottom:5" w="25%" h="30" text="Close"/>
</window>
```

其中的子控件的 name 必须按规则命名，详见表 4.157。

表 4.157 命名规则

| 取值 | 说明 |
| --- | --- |
| r | 红色分量可以是 spin_box、edit 和 slider |
| g | 绿色分量可以是 spin_box、edit 和 slider |
| b | 蓝色分量可以是 spin_box、edit 和 slider |
| h | Hue 分量可以是 spin_box、edit、slider 和 color_component |
| s | Saturation 分量可以是 spin_box、edit 和 slider |
| v | Value/Brightness 分量可以是 spin_box、edit 和 slider |
| sv | Saturation 和 Value/Brightness 分量可以是 color_component |
| old | 旧的值可以是 spin_box、edit 和 color_tile |
| new | 新的值可以是 spin_box、edit 和 color_tile |

### 4.5.3 gif_image

GIF 图片控件。需要注意的是 GIF 图片的尺寸大于控件大小时会自动缩小图片，但一般的嵌入式系统的硬件加速都不支持图片缩放，所以缩放图片会导致性能明显下降。如果性能不满意时，请确认一下 GIF 图片的尺寸是否小余控件大小。gif_image_t 是 image_base_t 的子类控件，image_base_t 的函数均适用于 gif_image_t 控件。

1. 函数

gif_image 提供的函数详见表 4.158。

表 4.158 gif_image 函数列表

| 函数名称 | 说明 |
| --- | --- |
| gif_image_cast | 转换为 gif_image 对象（供脚本语言使用） |
| gif_image_create | 创建 gif_image 对象 |

(1) gif_image_cast

- 函数原型

```
widget_t* gif_image_cast (widget_t* widget);
```

- 参数说明，详见表 4.159。

表 4.159 gif_image_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | gif_image 对象 |
| widget | widget_t* | gif_image 对象 |

(2) gif_image_create

- 函数原型

```
widget_t* gif_image_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.160。

表 4.160 gif_image_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"gif_image"标签创建 GIF 图片控件，详见程序清单 4.45。

程序清单 4.45 在 xml 中创建 gif_image

```
<!-- awtk\demos\assets\raw\ui\gif_image.xml -->
<window anim_hint="htranslate">
  <view x="0" y="0" w="100%" h="-60" children_layout="default(c=2,r=2)">
    <gif image="bee"/>
    <gif image="bee"/>
    <gif image="bee"/>
    <gif image="bee"/>
  </view>
</window>
```

### 4.5.4 guage

表盘控件。表盘控件就是一张图片。guage_t 是 widget_t 的子类控件，widget_t 的函数均适用于 guage_t 控件。

1. 函数

guage 提供的函数详见表 4.161。

表 4.161 guage 函数列表

| 函数名称 | 说明 |
| --- | --- |
| guage_cast | 转换 guage 对象（供脚本语言使用） |
| guage_create | 创建 guage 对象 |
| guage_set_draw_type | 设置图片的显示方式 |
| guage_set_image | 设置背景图片的名称 |

(1) guage_cast

- 函数原型

```
widget_t* guage_cast (widget_t* widget);
```

- 参数说明，详见表 4.162。

表 4.162 guage_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | guage 对象 |
| widget | widget_t* | guage 对象 |

(2) guage_create

- 函数原型

```
widget_t* guage_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.163。

表 4.163 guage_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) guage_set_draw_type

- 函数原型

```
ret_t guage_set_draw_type (widget_t* widget, image_draw_type_t draw_type);
```

- 参数说明，详见表 4.164。

表 4.164 guage_set_draw_type 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image 对象 |
| draw_type | image_draw_type_t | 显示方式 |

(4) guage_set_image

- 函数原型

```
ret_t guage_set_image (widget_t* widget, char* name);
```

- 参数说明，详见表 4.165。

表 4.165 guage_set_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image 对象 |
| name | char* | 图片名称，该图片必须存在于资源管理器 |

### 2. 属性

guage 还提供了下面属性，详见表 4.166。

表 4.166 guage 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| draw_type | image_draw_type_t | 图片的绘制方式 |
| image | char* | 背景图片 |

3. 示例

- 在 xml 中，使用"guage"标签创建表盘控件，详见程序清单 4.46。

程序清单 4.46 在 xml 中创建 guage

```
<!-- awtk\demos\assets\raw\ui\guage.xml -->
<window style="dark" anim_hint="htranslate" >
  <guage x="c" y="10" w="240" h="240" image="guage_bg" >
    <guage_pointer x="c" y="50" w="24" h="140" value="-128" image="guage_pointer"
        animation="value(from=-128, to=128, yoyo_times=1000, duration=3000, delay=1000)"/>
  </guage>

  <guage x="c" y="bottom:60" w="240" h="240" image="guage_bg" >
    <guage_pointer x="c" y="50" w="12" h="140" value="-128"
        animation="value(from=-128, to=128, yoyo_times=1000, duration=3000, delay=1000)"/>
    <guage_pointer x="c" y="50" w="12" h="140" value="-128" image="pointer"
        animation="value(from=-128, to=128, yoyo_times=1000, duration=3000)"/>
  </guage>
  <button name="close" x="center" y="bottom:10" w="25%" h="30" text="Close"/>
</window>
```

- 在 C 代码中，使用 guage_create 创建表盘控件，详见程序清单 4.47。

程序清单 4.47 在 C 代码中创建 guage

```
widget_t* guage = guage_create(win, 10, 10, 200, 200);
guage_set_image(guage, "guage_bg");
```

### 4.5.5 guage_pointer

仪表指针控件。仪表指针就是一张旋转的图片，图片可以是普通图片也可以是 SVG 图片。在嵌入式平台上对于旋转图片，SVG 图片的效率比位图高数倍，所以推荐使用 SVG 图片。guage_pointer_t 是 widget_t 的子类控件，widget_t 的函数均适用于 guage_pointer_t 控件。

1. 函数

guage_pointer 提供的函数详见表 4.167。

表 4.167 guage_pointer 函数列表

| 函数名称 | 说明 |
| --- | --- |
| guage_pointer_cast | 转换 guage_pointer 对象（供脚本语言使用） |
| guage_pointer_create | 创建 guage_pointer 对象 |
| guage_pointer_set_angle | 设置指针角度 12 点钟方向为 0 度，顺时钟方向为正，单位为度 |
| guage_pointer_set_image | 设置指针的图片 |

(1) guage_pointer_cast

- 函数原型

```
widget_t* guage_pointer_cast (widget_t* widget);
```

- 参数说明，详见表 4.168。

表 4.168 guage_pointer_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | guage_pointer 对象 |
| widget | widget_t* | guage_pointer 对象 |

(2) guage_pointer_create

- 函数原型

```
widget_t* guage_pointer_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.169。

表 4.169 guage_pointer_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) guage_pointer_set_angle

- 函数原型

```
ret_t guage_pointer_set_angle (widget_t* widget, int32_t angle);
```

- 参数说明，详见表 4.170。

表 4.170 guage_pointer_set_angle 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| angle | int32_t | 指针角度 |

(4) guage_pointer_set_image

- 函数原型

```
ret_t guage_pointer_set_image (widget_t* widget, char* name);
```

- 参数说明，详见表 4.171。

表 4.171 guage_pointer_set_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| image | const char* | 指针的图片 |

2. 属性

guage_pointer 还提供了下面属性，详见表 4.172。

表 4.172 guage_pointer 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| angle | int32_t | 指针角度。12 点钟方向为 0 度，顺时钟方向为正，单位为度 |
| image | char* | 指针图片 |

3. 示例

- 在 xml 中，使用"guage_pointer"标签创建仪表指针控件，详见程序清单 4.48。

程序清单 4.48 在 xml 中创建 guage_pointer

```
<!-- awtk\demos\assets\raw\ui\guage_pointer.xml -->
<window style="dark" anim_hint="htranslate" >
  <guage_pointer x="c" y="10" w="240" h="240" image="guage_pointer_bg" >
    <guage_pointer_pointer x="c" y="50" w="24" h="140" value="-128" image="guage_pointer_pointer"
        animation="value(from=-128, to=128, yoyo_times=1000, duration=3000, delay=1000)"/>
  </guage_pointer>

   <guage_pointer x="c" y="bottom:60" w="240" h="240" image="guage_pointer_bg" >
    <guage_pointer_pointer x="c" y="50" w="12" h="140" value="-128"
        animation="value(from=-128, to=128, yoyo_times=1000, duration=3000, delay=1000)"/>
    <guage_pointer_pointer x="c" y="50" w="12" h="140" value="-128" image="pointer"
        animation="value(from=-128, to=128, yoyo_times=1000, duration=3000)"/>
  </guage_pointer>
  <button name="close" x="center" y="bottom:10" w="25%" h="30" text="Close"/>
</window>
```

- 在 C 代码中，使用 guage_pointer_create 创建仪表指针控件，详见程序清单 4.49。

程序清单 4.49 在 C 代码中创建 guage_pointer

```
widget_t* guage_pointer = guage_pointer_create(guage, 10, 10, 100, 30);
guage_pointer_set_image(guage_pointer, "guage_pointer");
```

创建之后，需要用 guage_pointer_set_image 设置仪表指针图片。

### 4.5.6 image_animation

图片动画控件，指定一个图片前缀，依次显示指定序列的图片，从而形成动画效果。image_animation_t 是 widget_t 的子类控件，widget_t 的函数均适用于 image_animation_t 控件。

### 1. 函数

image_animation 提供的函数详见表 4.173。

表 4.173 image_animation 函数列表

| 函数名称 | 说明 |
| --- | --- |
| image_animation_cast | 转换 image_animation 对象（供脚本语言使用） |
| image_animation_create | 创建 image_animation 对象 |
| image_animation_pause | 暂停 |
| image_animation_play | 播放 |
| image_animation_set_auto_play | 设置是否自动播放 |
| image_animation_set_delay | 设置延迟播放时间（仅适用于自动播放） |
| image_animation_set_image | 设置图片前缀 |
| image_animation_set_interval | 设置播放间隔时间 |
| image_animation_set_loop | 设置是否循环播放 |
| image_animation_set_sequence | 设置播放序列比如 image 为"fire"，sequence 为"123", 将依次播放"fire1", "fire2", "fire3" |
| image_animation_stop | 停止（并重置 index 为 0） |

(1) image_animation_cast

- 函数原型

```
widget_t* image_animation_cast (widget_t* widget);
```

- 参数说明，详见表 4.174。

表 4.174 image_animation_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | image_animation 对象 |
| widget | widget_t* | image_animation 对象 |

(2) image_animation_create

- 函数原型

```
widget_t* image_animation_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.175。

表 4.175 image_animation_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) image_animation_pause

- 函数原型

```
ret_t image_animation_pause (widget_t* widget);
```

- 参数说明，详见表 4.176。

表 4.176 image_animation_pause 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |

(4) image_animation_play

- 函数原型

```
ret_t image_animation_play (widget_t* widget);
```

- 参数说明，详见表 4.177。

表 4.177 image_animation_play 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |

(5) image_animation_set_auto_play

- 函数原型

```
ret_t image_animation_set_auto_play (widget_t* widget, bool_t auto_play);
```

- 参数说明，详见表 4.178。

表 4.178 image_animation_set_auto_play 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |
| auto_play | bool_t | 是否自动播放 |

(6) image_animation_set_delay

- 函数原型

```
ret_t image_animation_set_delay (widget_t* widget, uint32_t delay);
```

- 参数说明，详见表 4.179。

表 4.179 image_animation_set_delay 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |
| delay | uint32_t | 延迟播放时间（毫秒） |

(7) image_animation_set_image

- 函数原型

```
ret_t image_animation_set_image (widget_t* widget, const char* image);
```

- 参数说明，详见表 4.180。

表 4.180 image_animation_set_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |
| image | const char* | 图片前缀 |

(8) image_animation_set_interval

- 函数原型

```
ret_t image_animation_set_interval (widget_t* widget, uint32_t interval);
```

- 参数说明，详见表 4.181。

表 4.181 image_animation_set_interval 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |
| interval | uint32_t | 间隔时间（毫秒） |

(9) image_animation_set_loop

- 函数原型

```
ret_t image_animation_set_loop (widget_t* widget, bool_t loop);
```

- 参数说明，详见表 4.182。

表 4.182 image_animation_set_loop 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |
| loop | bool_t | 是否循环播放 |

(10) image_animation_set_sequence

- 函数原型

```
ret_t image_animation_set_sequence (widget_t* widget, const char* sequence);
```

- 参数说明，详见表 4.183。

表 4.183 image_animation_set_sequence 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |
| sequence | const char* | 播放序列 |

(11) image_animation_stop

- 函数原型

```
ret_t image_animation_stop (widget_t* widget);
```

- 参数说明，详见表 4.184。

表 4.184 image_animation_stop 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_animation 对象 |

2. 属性

image_animation 还提供了下面属性，详见表 4.185。

表 4.185 image_animation 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| auto_play | bool_t | 是否自动播放 |
| delay | uint32_t | 自动播放时延迟播放的时间（毫秒） |
| image | char* | 图片名称的前缀 |
| interval | uint32_t | 每张图片播放的时间（毫秒） |
| loop | bool_t | 是否循环播放 |
| sequence | char* | 播放的序列，字符可选值为:0-9,a-z,A-Z |

3. 示例

- 在 xml 中，使用"image_animation"标签创建图片动画控件，详见程序清单 4.50。

程序清单 4.50 在 xml 中创建 image_animation

```
<!-- awtk\demos\assets\raw\ui\image_animation.xml -->
<window anim_hint="htranslate">
  <view x="0" y="0" w="100%" h="80%" children_layout="default(r=4,c=4)">
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="100"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="200"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="300"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="400"/>

    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="500"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="600"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="700"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="800"/>

    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="900"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1000"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1100"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1200"/>
```

```
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1300"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1400"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1500"/>
    <image_animation image="ani" sequence="123456789abc" auto_play="true" interval="50" delay="1600"/>
  </view>

  <button name="close" x="center" y="bottom:50" w="50%" h="50" text="close"/>
</window>
```

- 在 C 代码中，使用 image_animation_create 创建图片动画控件，详见程序清单 4.51。

程序清单 4.51 在 C 代码中创建 image_animation

```
image_animation = image_animation_create(win, 10, 10, 200, 200);
image_animation_set_image(image_animation, "ani");
image_animation_set_interval(image_animation, 50);
image_animation_set_sequence(image_animation, "123456789abc");
image_animation_play(image_animation);
```

### 4.5.7 image_value

图片值控件。可以用图片来表示如电池电量、WIFI 信号强度和其它各种数值的值。image_value_t 是 widget_t 的子类控件，widget_t 的函数均适用于 image_value_t 控件。其原理如下：

1. 把 value 以 format 为格式转换成字符串。
2. 把每个字符与 image（图片文件名前缀）映射成一个图片名。
3. 最后把这些图片显示出来。

#### 1. 函数

image_value 提供的函数详见表 4.186。

表 4.186 image_value 函数列表

| 函数名称 | 说明 |
| --- | --- |
| image_value_cast | 转换 image_value 对象（供脚本语言使用） |
| image_value_create | 创建 image_value 对象 |
| image_value_set_format | 设置格式 |
| image_value_set_image | 设置图片前缀 |
| image_value_set_value | 设置值 |

(1) image_value_cast

- 函数原型

```
widget_t* image_value_cast (widget_t* widget);
```

- 参数说明，详见表 4.187。

表 4.187 image_value_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | image_value 对象 |
| widget | widget_t* | image_value 对象 |

(2) image_value_create

- 函数原型

```
widget_t* image_value_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.188。

表 4.188 image_value_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) image_value_set_format

- 函数原型

```
ret_t image_value_set_format (widget_t* widget, const char* format);
```

- 参数说明，详见表 4.189。

表 4.189 image_value_set_format 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_value 对象 |
| format | const char* | 格式 |

(4) image_value_set_image

- 函数原型

```
ret_t image_value_set_image (widget_t* widget, const char* image);
```

- 参数说明，详见表 4.190。

表 4.190 image_value_set_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_value 对象 |
| image | const char* | 图片前缀 |

(5) image_value_set_value

- 函数原型

```
ret_t image_value_set_value (widget_t* widget, float_t value);
```

- 参数说明，详见表 4.191。

表 4.191 image_value_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | image_value 对象 |
| value | float_t | 值 |

### 2. 属性

image_value 还提供了下面属性，详见表 4.192。

表 4.192 image_value 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| format | char* | 数值到字符串转换时的格式，缺省为"%d" |
| image | char* | 图片名称的前缀 |
| value | float_t | 值 |

### 3. 示例

- 在 xml 中，使用"image_value"标签创建图片值控件，详见程序清单 4.52。

程序清单 4.52 在 xml 中创建 image_value

```
<!-- awtk\demos\assets\raw\ui\image_value.xml -->
<window anim_hint="htranslate">
  <view x="0" y="0" w="100%" h="100%" children_layout="default(c=2,r=3)">
    <image_value   value="0" image="battery_"
      animation="value(from=0, to=5, duration=10000)"/>
    <image_value   value="0" image="battery_"
      animation="value(from=0, to=5, duration=10000)"/>
    <image_value   value="0" format="%02d" image="num_"
      animation="value(from=10, to=100, duration=100000)"/>
    <image_value   value="0" format="%04d" image="num_"
      animation="value(from=10, to=10000, duration=100000)"/>
    <image_value   value="0" format="%02.2f" image="num_"
      animation="value(from=0, to=100, duration=1000000)"/>
    <image_value   value="0" format="%.4f" image="num_"
      animation="value(from=0, to=100, duration=1000000)"/>
  </view>
</window>
```

- 在 C 代码中，使用 image_value_create 创建图片值控件，详见程序清单 4.53。

程序清单 4.53 在 C 代码中创建 image_value

```
image_value = image_value_create(win, 10, 10, 200, 200);
```

```
image_value_set_image(image_value, "num_");
image_value_set_value(image_value, 100);
```

### 4.5.8 keyboard

软键盘。软键盘是一个特殊的窗口，由编辑器通过输入法自动打开和关闭。编辑器输入类型和软键盘 UI 资源文件的对应关系，详见表 4.193。

表 4.193 输入类型与 UI 资源对应关系

| 输入类型 | 软键盘 UI 资源文件 |
| --- | --- |
| INPUT_PHONE | kb_phone.xml |
| INPUT_INT | kb_int.xml |
| INPUT_FLOAT | kb_float.xml |
| INPUT_UINT | kb_uint.xml |
| INPUT_UFLOAT | kb_ufloat.xml |
| INPUT_HEX | kb_hex.xml |
| INPUT_EMAIL | kb_ascii.xml |
| INPUT_PASSWORD | kb_ascii.xml |
| INPUT_CUSTOM | 使用自定义的键盘 |
| 其它 | kb_default.xml |

keyboard 中按钮子控件的名称功能说明，详见表 4.194。

表 4.194 功能说明

| 名称 | 功能 |
| --- | --- |
| return | 回车键 |
| action | 定制按钮 |
| backspace | 删除键 |
| space | 空格键 |
| close | 关闭软键盘 |
| 前缀 key: | 键值 |
| 前缀 page: | 切换到页面 |

#### 1. 函数

keyboard 提供的函数详见表 4.195。

表 4.195 keyboard 函数列表

| 函数名称 | 说明 |
| --- | --- |
| keyboard_close | 关闭 keyboard 窗口 |
| keyboard_create | 创建 keyboard 对象 |

(1) keyboard_close

- 函数原型

```
ret_t keyboard_close (widget_t* parent);
```

- 参数说明，详见表 4.196。

表 4.196 keyboard_close 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| parent | widget_t* | keyboard 对象 |

(2) keyboard_create

- 函数原型

```
widget_t* keyboard_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.197。

表 4.197 keyboard_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

### 4.5.9 progress_circle

进度圆环控件。progress_circle_t 是 widget_t 的子类控件，widget_t 的函数均适用于 progress_circle_t 控件。

1. 函数

progress_circle 提供的函数详见表 4.198。

表 4.198 progress_circle 函数列表

| 函数名称 | 说明 |
| --- | --- |
| progress_circle_cast | 转换为 progress_circle 对象（供脚本语言使用） |
| progress_circle_create | 创建 progress_circle 对象 |
| progress_circle_set_counter_clock_wise | 设置是否为逆时针方向 |
| progress_circle_set_line_width | 设置环线的厚度 |
| progress_circle_set_max | 设置最大值 |
| progress_circle_set_show_text | 设置是否显示文本 |
| progress_circle_set_start_angle | 设置起始角度 |
| progress_circle_set_unit | 设置单位 |
| progress_circle_set_value | 设置值 |

(1) progress_circle_cast

- 函数原型

```
widget_t* progress_circle_cast (widget_t* widget);
```

- 参数说明，详见表 4.199。

表 4.199 progress_circle_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | progress_circle 对象 |
| widget | widget_t* | progress_circle 对象 |

(2) progress_circle_create

- 函数原型

```
widget_t* progress_circle_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.200。

表 4.200 progress_circle_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) progress_circle_set_counter_clock_wise

- 函数原型

```
ret_t progress_circle_set_counter_clock_wise (widget_t* widget, bool_t counter_clock_wise);
```

- 参数说明，详见表 4.201。

表 4.201 progress_circle_set_counter_clock_wise 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| counter_clock_wise | bool_t | 是否为逆时针方向 |

(4) progress_circle_set_line_width

- 函数原型

```
ret_t progress_circle_set_line_width (widget_t* widget, uint32_t line_width);
```

- 参数说明，详见表 4.202。

表 4.202 progress_circle_set_line_width 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| line_width | uint32_t | 环线的厚度 |

(5) progress_circle_set_max

- 函数原型

```
ret_t progress_circle_set_max (widget_t* widget, uint32_t max);
```

- 参数说明，详见表 4.203。

表 4.203 progress_circle_set_max 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| max | uint32_t | 最大值 |

(6) progress_circle_set_show_text

- 函数原型

```
ret_t progress_circle_set_show_text (widget_t* widget, bool_t show_text);
```

- 参数说明，详见表 4.204。

表 4.204 progress_circle_set_show_text 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| show_text | bool_t | 是否显示文本 |

(7) progress_circle_set_start_angle

- 函数原型

```
ret_t progress_circle_set_start_angle (widget_t* widget, int32_t start_angle);
```

- 参数说明，详见表 4.205。

表 4.205 progress_circle_set_start_angle 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| start_angle | int32_t | 起始角度 |

(8) progress_circle_set_unit

- 函数原型

```
ret_t progress_circle_set_unit (widget_t* widget, const char* unit);
```

- 参数说明，详见表 4.206。

表 4.206 progress_circle_set_unit 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| unit | const char* | 单位 |

(9) progress_circle_set_value

- 函数原型

```
ret_t progress_circle_set_value (widget_t* widget, float_t value);
```

- 参数说明，详见表 4.207。

表 4.207 progress_circle_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| value | float_t | 值 |

2. 属性

progress_circle 还提供了下面属性，详见表 4.208。

表 4.208 progress_circle 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| counter_clock_wise | bool_t | 是否为逆时针方向（缺省为 FALSE） |
| line_width | uint32_t | 环线的厚度（缺省为 8） |
| max | uint32_t | 最大值（缺省为 100） |
| show_text | bool_t | 是否显示文本（缺省为 TRUE） |
| start_angle | int32_t | 起始角度（单位为度，缺省-90） |
| unit | char* | 单元（缺省无） |
| value | float_t | 值（缺省为 0） |

3. 事件

progress_circle 还提供了下面事件，详见表 4.209。

表 4.209 progress_circle 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值改变事件 |

4. 示例

- 在 xml 中，使用"progress_circle"标签创建进度圆环控件，详见程序清单 4.54。

程序清单 4.54 在 xml 中创建 progress_circle

```
<!-- awtk\demos\assets\raw\ui\progress_circle.xml -->
<window >
  <view x="c" y="m" w="200" h="200" children_layout="default(r=2,c=2,s=5)">
    <progress_circle max="360" show_text="true"
      animation="value(from=0, to=360, yoyo_times=1000, duration=3000, easing=sin_inout)" />

    <progress_circle max="360" show_text="true" start_angle="90"
      animation="value(from=0, to=300, yoyo_times=1000, duration=3000, easing=sin_inout)" />
```

```
    <progress_circle style="image" max="360" show_text="true" start_angle="-213"
      animation="value(from=0, to=300, yoyo_times=1000, duration=3000, easing=sin_inout)" />

    <progress_circle style="image" max="100" show_text="true" start_angle="90" start_angle="-225"
      counter_clock_wise="true" unit="%"
      animation="value(from=10, to=100, yoyo_times=1000, duration=3000, easing=sin_inout)" />
  </view>

  <button name="close" x="center" y="bottom:10" w="25%" h="30" text="Close"/>
</window>
```

- 在 C 代码中，使用函数 progress_circle_create 创建进度圆环控件，详见程序清单 4.55。

程序清单 4.55 在 C 代码中创建 progress_circle

```
progress_circle = progress_circle_create(win, 10, 10, 200, 200);
progress_circle_set_max(progress_circle, 360);
widget_set_value(progress_circle, 128);
```

### 4.5.10 rich_text

图文混排控件，实现简单的图文混排。rich_text_t 是 widget_t 的子类控件，widget_t 的函数均适用于 rich_text_t 控件。

1. 函数

rich_text 提供的函数详见表 4.210。

表 4. 210 rich_text 函数列表

| 函数名称 | 说明 |
| --- | --- |
| rich_text_create | 创建 rich_text 对象 |
| rich_text_set_text | 设置文本 |

(1) rich_text_create

- 函数原型

```
widget_t* rich_text_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.211。

表 4. 211 rich_text_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(2) rich_text_set_text

- 函数原型

```
ret_t rich_text_set_text (widget_t* widget, char* text);
```

- 参数说明，详见表 4.212。

表 4. 212 rich_text_set_text 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| text | char* | 文本 |

2. 属性

rich_text 还提供了下面属性，详见表 4.213。

表 4. 213 rich_text 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| line_gap | int32_t | 行间距 |

3. 示例

- 在 xml 中，使用"rich_text"标签创建图文混排控件，详见程序清单 4.56。

程序清单 4.56 在 xml 中创建 rich_text

```
<!-- awtk\demos\assets\raw\ui\rich_text.xml -->
<window anim_hint="htranslate" >
    <rich_text x="0" y="0" w="100%" h="60" text="<image name=&quota;bricks&quota;/><font
align_v=&quota;middle&quota;>hello awtk!</font><font color=&quota;red&quota;
size=&quota;32&quota;>ZLG</font>" />
    <rich_text x="0" y="70" w="100%" h="60" text="<image name=&quota;bricks&quota;/><font
align_v=&quota;top&quota;>hello awtk!</font><font color=&quota;green&quota;
size=&quota;32&quota;>ZLG</font>" />
    <rich_text x="0" y="140" w="100%" h="60" text="<image name=&quota;bricks&quota;/><font
align_v=&quota;bottom&quota;>hello awtk!</font><font color=&quota;blue&quota;
size=&quota;32&quota;>ZLG</font>" />
    <rich_text line_gap="5" x="0" y="210" w="100%" h="200" text="<image
name=&quota;bricks&quota;/><font color=&quota;gold&quota; align_v=&quota;bottom&quota;
size=&quota;24&quota;>hello awtk!</font><font color=&quota;green&quota; size=&quota;20&quota;>ProTip!
The feed shows you events from people you follow and repositories you watch.
hello world. </font><font color=&quota;red&quota; size=&quota;20&quota;>确定取消中文字符测试。确定。取
消。中文字符测试。</font>" />
    <button name="close" text="Close" x="c" y="bottom:10" w="25%" h="40" />
</window>
```

- 在 C 代码中，使用 rich_text_create 创建图文混排控件，详见程序清单 4.57。

程序清单 4.57 在 C 代码中创建 rich_text

```
widget_t* rich_text = rich_text_create(win, 0, 0, 0, 0);
widget_set_text_utf8(rich_text,
"<image name=\"bricks\"/><font color=\"gold\" align_v=\"bottom\" "
"size=\"24\">hello awtk!</font><font color=\"green\"
size=\"20\">ProTip! The "
"feed shows you events from people you follow and repositories you
watch. "
"nhello world. </font><font color=\"red\" "
"size=\"20\">确定取消中文字符测试。确定。取消。中文字符测试。</font>");
widget_set_self_layout_params(rich_text, "center", "middle", "100%", "100%");
```

### 4.5.11 slide_menu

左右滑动菜单控件。一般用一组按钮作为子控件，通过左右滑动改变当前的项。除了当菜单使用外，也可以用来切换页面。slide_menu_t 是 widget_t 的子类控件，widget_t 的函数均适用于 slide_menu_t 控件。

1. 函数

slide_menu 提供的函数详见表 4.214。

表 4. 214 slide_menu 函数列表

| 函数名称 | 说明 |
| --- | --- |
| slide_menu_cast | 转换为 slide_menu 对象（供脚本语言使用） |
| slide_menu_create | 创建 slide_menu 对象 |
| slide_menu_set_align_v | 设置垂直对齐方式 |
| slide_menu_set_min_scale | 设置最小缩放比例 |
| slide_menu_set_value | 设置当前项 |

(1) slide_menu_cast

- 函数原型

```
widget_t* slide_menu_cast (widget_t* widget);
```

- 参数说明，详见表 4.215。

表 4. 215 slide_menu_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | slide_menu 对象 |
| widget | widget_t* | slide_menu 对象 |

(2) slide_menu_create

- 函数原型

```
widget_t* slide_menu_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.216。

表 4. 216 slide_menu_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) slide_menu_set_align_v

- 函数原型

```
ret_t slide_menu_set_align_v (widget_t* widget, align_v_t align_v);
```

- 参数说明，详见表 4.217。

表 4. 217 slide_menu_set_align_v 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | slide_menu 对象 |
| align_v | align_v_t | 对齐方式 |

(4) slide_menu_set_min_scale

- 函数原型

```
ret_t slide_menu_set_min_scale (widget_t* widget, float_t min_scale);
```

- 参数说明，详见表 4.218。

表 4. 218 slide_menu_set_min_scale 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | slide_menu 对象 |
| min_scale | float_t | 最小缩放比例，范围[0.5-1] |

(5) slide_menu_set_value

- 函数原型

```
ret_t slide_menu_set_value (widget_t* widget, uint32_t value);
```

- 参数说明，详见表 4.219。

表 4. 219 slide_menu_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | slide_menu 对象 |
| value | uint32_t | 当前项的索引 |

### 2. 属性

slide_menu 还提供了下面属性，详见表 4.220。

表 4. 220 slide_menu 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| align_v | align_v_t | 垂直对齐方式 |
| min_scale | float_t | 最小缩放比例 |
| value | int32_t | 值代表当前选中项的索引 |

### 3. 事件

slide_menu 还提供了下面事件，详见表 4.221。

表 4. 221 slide_menu 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值（当前项）即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值（当前项）改变事件 |

### 4. 示例

- 在 xml 中，使用"slide_menu"标签创建左右滑动菜单，详见程序清单 4.58。

程序清单 4. 58 在 xml 中创建 slide_menu

```
<!-- awtk\demos\assets\raw\ui\slide_menu.xml -->
<window theme="slide_menu" children_layout="default(c=1,h=60,s=10)">
  <slide_menu style="mask" align_v="top">
    <button style="slide_button" text="0"/>
    <button style="slide_button" text="1"/>
    <button style="slide_button" text="2"/>
    <button style="slide_button" text="3"/>
    <button style="slide_button" text="4"/>
  </slide_menu>
  ...
</window>
```

- 在 C 代码中，使用函数 slide_menu_create 创建左右滑动菜单，详见程序清单 4.59。

程序清单 4. 59 在 C 代码中创建 slide_menu

```
slide_menu = slide_menu_create(win, 10, 10, 300, 60);
b = button_create(slide_menu, 0, 0, 0, 0);
widget_set_text_utf8(b, "1");
b = button_create(slide_menu, 0, 0, 0, 0);
widget_set_text_utf8(b, "2");
b = button_create(slide_menu, 0, 0, 0, 0);
widget_set_text_utf8(b, "3");
b = button_create(slide_menu, 0, 0, 0, 0);
widget_set_text_utf8(b, "4");
```

可按下面的方法关注当前项改变的事件：

```
widget_on(slide_menu, EVT_VALUE_CHANGED, on_current_changed, slide_menu);
```

可按下面的方法关注当前按钮被点击的事件：

```
widget_on(b, EVT_CLICK, on_button_click, b);
```

### 4.5.12 slide_view

滑动视图。滑动视图可以管理多个页面，并通过滑动来切换当前页面。也可以管理多张图片，让它们自动切换。slide_view_t 是 widget_t 的子类控件，widget_t 的函数均适用于 slide_view_t 控件。

1. 函数

slide_view 提供的函数详见表 4.222。

表 4. 222 slide_view 函数列表

| 函数名称 | 说明 |
| --- | --- |
| slide_view_cast | 转换为 slide_view 对象（供脚本语言使用） |
| slide_view_create | 创建 slide_view 对象 |
| slide_view_set_active | 设置当前页的序号 |
| slide_view_set_auto_play | 设置为自动播放模式 |
| slide_view_set_vertical | 设置为上下滑动（缺省为左右滑动） |

(1) slide_view_cast

- 函数原型

```
widget_t* slide_view_cast (widget_t* widget);
```

- 参数说明，详见表 4.223。

表 4. 223 slide_view_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | slide_view 对象 |
| widget | widget_t* | slide_view 对象 |

(2) slide_view_create

- 函数原型

```
widget_t* slide_view_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.224。

表 4. 224 slide_view_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |

| | | |
|---|---|---|
| h | wh_t | 高度 |

(3) slide_view_set_active

- 函数原型

```
ret_t slide_view_set_active (widget_t* widget, uint32_t index);
```

- 参数说明，详见表 4.225。

表 4. 225 slide_view_set_active 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | slide_view 对象 |
| index | uint32_t | 当前页的序号 |

(4) slide_view_set_auto_play

- 函数原型

```
ret_t slide_view_set_auto_play (widget_t* widget, uint16_t auto_play);
```

- 参数说明，详见表 4.226。

表 4. 226 slide_view_set_auto_play 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | slide_view 对象 |
| auto_play | uint16_t | 0 表示禁止自动播放，非 0 表示自动播放时每一页播放的时间 |

(5) slide_view_set_vertical

- 函数原型

```
ret_t slide_view_set_vertical (widget_t* widget, bool_t vertical);
```

- 参数说明，详见表 4.227。

表 4. 227 slide_view_set_vertical 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | slide_view 对象 |
| vertical | bool_t | TRUE 表示上下滑动，FALSE 表示左右滑动 |

2. 属性

slide_view 还提供了下面属性，详见表 4.228。

表 4. 228 slide_view 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| auto_play | uint16_t | 自动播放。0 表示禁止自动播放，非 0 表示自动播放时每一页播放的时间 |
| vertical | bool_t | 是否为上下滑动模式 |

3. 事件

slide_view 还提供了下面事件，详见表 4.229。

表 4. 229 slide_view 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值（当前页）即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值（当前页）改变事件 |

4. 示例

- 在 xml 中，使用"slide_view"标签创建滑动视图控件，详见程序清单 4.60。

程序清单 4. 60 在 xml 中创建 slide_view

```
<!-- awtk\demos\assets\raw\ui\slide_view_h.xml -->
<window anim_hint="htranslate">
  <slide_view x="0" y="0" w="100%" h="100%" style="dot">
    <view x="0" y="0" w="100%" h="100%" children_layout="default(w=60,h=60,m=5,s=10)">
      <button name="close" style="grid_item">
        <image draw_type="icon" x="center" y="top" w="100%" h="80%" image="message" />
        <label x="center" y="bottom" w="100%" h="30%" text="Close" />
      </button>
       ...
  </slide_view>
</window>
```

- 在 C 代码中，使用函数 slide_view_create 创建滑动视图控件，详见程序清单 4.61。

程序清单 4. 61 在 C 代码中创建 slide_view

```
slide_view = slide_view_create(win, 0, 0, win->w, win->h);
```

### 4.5.13 svg_image

SVG 图片控件。svg_image_t 是 image_base_t 的子类控件，image_base_t 的函数均适用于 svg_image_t 控件。

1. 函数

svg_image 提供的函数详见表 4.230。

表 4. 230 svg_image 函数列表

| 函数名称 | 说明 |
| --- | --- |
| svg_image_cast | 转换为 svg_image 对象（供脚本语言使用） |
| svg_image_create | 创建 svg_image 对象 |

(1) svg_image_cast

- 函数原型

```
widget_t* svg_image_cast (widget_t* widget);
```

- 参数说明，详见表 4.231。

表 4. 231 svg_image_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | svg_image 对象 |
| widget | widget_t* | svg_image 对象 |

(2) svg_image_create

- 函数原型

```
widget_t* svg_image_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.232。

表 4. 232 svg_image_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

2. 示例

- 在 xml 中，使用"svg_image"标签创建 SVG 图片控件，详见程序清单 4.62。

程序清单 4. 62 在 xml 中创建 svg_image

```
<!-- awtk\demos\assets\raw\ui\svg_image.xml -->
<window anim_hint="htranslate" children_layout="default(c=4,r=4)">
  <svg image="china" />
  <svg image="pointer_1" />
  <svg image="girl" />
  <svg image="pointer_4" />

  <svg image="china"
    animation="rotation(from=0, to=6.28, yoyo_times=1000, duration=5000);scale(x_from=0, x_to=2, y_from=0,
y_to=2, yoyo_times=1000, duration=5000);"/>
  <svg image="pointer_1"
    animation="rotation(from=0, to=6.28, yoyo_times=1000, duration=5000);scale(x_from=0, x_to=2, y_from=0,
y_to=2, yoyo_times=1000, duration=5000);"/>
  <svg image="girl"
    animation="rotation(from=0, to=6.28, yoyo_times=1000, duration=5000);scale(x_from=0, x_to=2, y_from=0,
y_to=2, yoyo_times=1000, duration=5000);"/>

  <svg image="pointer_4"
    animation="rotation(from=0, to=6.28, yoyo_times=1000, duration=5000);scale(x_from=0, x_to=2, y_from=0,
y_to=2, yoyo_times=1000, duration=5000);"/>
```

```
</window>
```

- 在 C 代码中，使用函数 svg_image_create 创建 SVG 图片控件，详见程序清单 4.63。

程序清单 4. 63 在 C 代码中创建 svg_image

```
widget_t* image = svg_image_create(win, 10, 10, 200, 200);
image_set_image(image, "girl");
```

创建之后需要用 widget_set_image 设置图片名称。

### 4.5.14 switch

开关控件。switch_t 是 widget_t 的子类控件，widget_t 的函数均适用于 switch_t 控件。

#### 1. 函数

switch 提供的函数详见表 4.233。

表 4. 233 switch 函数列表

| 函数名称 | 说明 |
| --- | --- |
| switch_cast | 转换为 switch 对象（供脚本语言使用） |
| switch_create | 创建 switch 对象 |
| switch_set_value | 设置控件的值 |

(1) switch_cast

- 函数原型

```
widget_t* switch_cast (widget_t* widget);
```

- 参数说明，详见表 4.234。

表 4. 234 switch_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | switch 对象 |
| widget | widget_t* | switch 对象 |

(2) switch_create

- 函数原型

```
widget_t* switch_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.235。

表 4. 235 switch_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) switch_set_value

- 函数原型

```
ret_t switch_set_value (widget_t* widget, bool_t value);
```

- 参数说明，详见表 4.236。

表 4. 236 switch_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | switch 对象 |
| value | bool_t | 值 |

### 2. 属性

switch 还提供了下面属性，详见表 4.237。

表 4. 237 switch 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| max_xoffset_ratio | float_t | 当开关处于关闭时，图片偏移相对于图片宽度的比例（缺省为 1/3） |
| round_radius | int32_t | 图片的圆角半径 |
| value | bool_t | 值 |

### 3. 事件

switch 还提供了下面事件，详见表 4.238。

表 4. 238 switch 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值（开关状态）即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值（开关状态）改变事件 |

### 4. 示例

- 在 xml 中，使用"switch"标签创建开关控件，详见程序清单 4.64。

程序清单 4. 64 在 xml 中创建 switch

```
<!-- awtk\demos\assets\raw\ui\switch.xml -->
<window anim_hint="htranslate">
  <switch x="10" y="60" w="60" h="22" />
  <switch x="200" y="60" w="60" h="22" round_radius="10" />
  <button name="close" x="center" y="middle" w="50%" h="30" text="close"/>
</window>
```

- 在 C 代码中，使用函数 switch_create 创建开关控件，详见程序清单 4.65。

程序清单 4. 65 在 C 代码中创建 switch

```
widget_t* sw = switch_create(win, 10, 10, 80, 30);
widget_on(sw, EVT_VALUE_CHANGED, on_changed, NULL);
```

### 4.5.15 text_selector

文本选择器控件，通常用于选择日期和时间等。目前需要先设置 options 和 visible_nr，再设置其它参数（在 XML 中也需要按此顺序）。text_selector_t 是 widget_t 的子类控件，widget_t 的函数均适用于 text_selector_t 控件。

1. 函数

text_selector 提供的函数详见表 4.239。

表 4. 239 text_selector 函数列表

| 函数名称 | 说明 |
| --- | --- |
| text_selector_cast | 转换 text_selector 对象（供脚本语言使用） |
| text_selector_create | 创建 text_selector 对象 |
| text_selector_get_option | 获取第 index 个选项 |
| text_selector_get_text | 获取 text_selector 的文本 |
| text_selector_get_value | 获取 text_selector 的值 |
| text_selector_reset_options | 重置所有选项 |
| text_selector_set_options | 设置选项 |
| text_selector_set_range_options | 设置一系列的整数选项 |
| text_selector_set_selected_index | 设置第 index 个选项为当前选中的选项 |
| text_selector_set_text | 设置 text_selector 的文本 |
| text_selector_set_value | 设置 text_selector 的值 |
| text_selector_set_visible_nr | 设置可见的选项数 |
| text_selector_append_option | 追加一个选项 |
| text_selector_count_options | 获取选项个数 |

(1) text_selector_cast

- 函数原型

```
widget_t* text_selector_cast (widget_t* widget);
```

- 参数说明，详见表 4.240。

表 4. 240 text_selector_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | text_selector 对象 |
| widget | widget_t* | text_selector 对象 |

(2) text_selector_create

- 函数原型

```
widget_t* text_selector_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.241。

表 4. 241 text_selector_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |

| parent | widget_t* | 父控件 |
| --- | --- | --- |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) text_selector_get_option

- 函数原型

```
text_selector_option_t* text_selector_get_option (widget_t* widget, uint32_t index);
```

- 参数说明，详见表 4.242。

表 4. 242 text_selector_get_option 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | text_selector_option_t* | 成功返回选项，失败返回 NULL |
| widget | widget_t* | text_selector 对象 |
| index | uint32_t | 选项的索引 |

(4) text_selector_get_text

- 函数原型

```
char* text_selector_get_text (widget_t* widget);
```

- 参数说明，详见表 4.243。

表 4. 243 text_selector_get_text 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | char* | 返回文本 |
| widget | widget_t* | text_selector 对象 |

(5) text_selector_get_value

- 函数原型

```
int32_t text_selector_get_value (widget_t* widget);
```

- 参数说明，详见表 4.244。

表 4. 244 text_selector_get_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | int32_t | 返回值 |
| widget | widget_t* | text_selector 对象 |

(6) text_selector_reset_options

- 函数原型

```
ret_t text_selector_reset_options (widget_t* widget);
```

- 参数说明，详见表 4.245。

表 4. 245 text_selector_reset_options 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |

(7) text_selector_set_options

- 函数原型

```
ret_t text_selector_set_options (widget_t* widget, char* options);
```

- 参数说明，详见表 4.246。

表 4. 246 text_selector_set_options 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| options | char* | 选项 |

(8) text_selector_set_range_options

- 函数原型

```
ret_t text_selector_set_range_options (widget_t* widget, int32_t start, uint32_t nr, int32_t step);
```

- 参数说明，详见表 4.247。

表 4. 247 text_selector_set_range_options 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| start | int32_t | 起始值 |
| nr | uint32_t | 个数 |
| step | int32_t | 步长 |

(9) text_selector_set_selected_index

- 函数原型

```
ret_t text_selector_set_selected_index (widget_t* widget, uint32_t index);
```

- 参数说明，详见表 4.248。

表 4. 248 text_selector_set_selected_index 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| index | uint32_t | 选项的索引 |

(10) text_selector_set_text

- 函数原型

```
ret_t text_selector_set_text (widget_t* widget, const char* text);
```

- 参数说明，详见表 4.249。

表 4. 249 text_selector_set_text 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| text | const char* | 文本 |

(11) text_selector_set_value

- 函数原型

```
ret_t text_selector_set_value (widget_t* widget, int32_t value);
```

- 参数说明，详见表 4.250。

表 4. 250 text_selector_set_value 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| value | int32_t | 值 |

(12) text_selector_set_visible_nr

- 函数原型

```
ret_t text_selector_set_visible_nr (widget_t* widget, uint32_t visible_nr);
```

- 参数说明，详见表 4.251。

表 4. 251 text_selector_set_visible_nr 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| visible_nr | uint32_t | 选项数 |

(13) text_selector_append_option

- 函数原型

```
ret_t text_selector_append_option (widget_t* widget, int32_t value, char* text);
```

- 参数说明，详见表 4.252。

表 4. 252 text_selector_append_option 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | text_selector 对象 |
| value | int32_t | 值 |
| text | char* | 文本 |

(14) text_selector_count_options

- 函数原型

```
int32_t text_selector_count_options (widget_t* widget);
```

- 参数说明，详见表 4.253。

表 4. 253 text_selector_count_options 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | int32_t | 返回选项个数 |
| widget | widget_t* | text_selector 对象 |

2. 属性

text_selector 还提供了下面属性，详见表 4.254。

表 4. 254 text_selector 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| options | char* | 设置可选项(冒号分隔值和文本，分号分隔选项，如:1:red;2:green;3:blue） |
| selected_index | int32_t | 当前选中的选项 |
| visible_nr | uint32_t | 可见的选项数量（只能是 3 或者 5，缺省为 5） |

3. 事件

text_selector 还提供了下面事件，详见表 4.255。

表 4. 255 text_selector 事件列表

| 事件名称 | 类型 | 说明 |
| --- | --- | --- |
| EVT_VALUE_WILL_CHANGE | event_t | 值）当前项）即将改变事件 |
| EVT_VALUE_CHANGED | event_t | 值）当前项）改变事件 |

4. 示例

- 在 xml 中，使用"text_selector"标签创建文本选择器控件，详见程序清单 4.66。

程序清单 4. 66 在 xml 中创建 text_selector

```
<!-- awtk\demos\assets\raw\ui\text_selector.xml -->
<window anim_hint="htranslate">
  <row    x="10" y="30" w="100%" h="150" children_layout="default(row=1,col=3)">
    <text_selector style="dark" options="2000-2050" text="2018"/>
    <text_selector options="1-12" text="1"/>
    <text_selector options="1-31" text="1"/>
  </row>
  <button name="close" x="center" y="middle" w="50%" h="30" text="close"/>
  <row    x="10" y="bottom:30" w="100%" h="90" children_layout="default(row=1,col=3)">
    <text_selector style="dark" options="2000-2050" visible_nr="3" text="2018"/>
    <text_selector options="1-12" visible_nr="3" text="1"/>
    <text_selector options="red;green;blue;gold;orange" visible_nr="3" text="red"/>
```

```
  </row>
</window>
```

- 在C代码中，使用函数text_selector_create创建文本选择器控件，详见程序清单4.67。

程序清单 4. 67 在 C 代码中创建 text_selector

```
widget_t* ts = text_selector_create(win, 10, 10, 80, 150);
text_selector_set_options(ts, "1:red;2:green;3:blue;4:orange;5:gold");
text_selector_set_value(ts, 1);
widget_use_style(ts, "dark");
```

### 4.5.16 time_clock

模拟时钟控件。time_clock_t 是 widget_t 的子类控件，widget_t 的函数均适用于 time_clock_t 控件。

1. 函数

time_clock 提供的函数详见表 4.256。

表 4. 256 time_clock 函数列表

| 函数名称 | 说明 |
| --- | --- |
| time_clock_cast | 转换为 time_clock 对象（供脚本语言使用） |
| time_clock_create | 创建 time_clock 对象 |
| time_clock_set_bg_image | 设置背景图片 |
| time_clock_set_hour | 设置小时的值 |
| time_clock_set_hour_image | 设置小时的图片 |
| time_clock_set_image | 设置图片 |
| time_clock_set_minute | 设置分钟的值 |
| time_clock_set_minute_image | 设置分钟的图片 |
| time_clock_set_second | 设置秒的值 |
| time_clock_set_second_image | 设置秒的图片 |

(1) time_clock_cast

- 函数原型

```
widget_t* time_clock_cast (widget_t* widget);
```

- 参数说明，详见表 4.257。

表 4. 257 time_clock_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | time_clock 对象 |
| widget | widget_t* | time_clock 对象 |

(2) time_clock_create

- 函数原型

```
widget_t* time_clock_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.258。

表 4. 258 time_clock_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |
| h | wh_t | 高度 |

(3) time_clock_set_bg_image

- 函数原型

```
ret_t time_clock_set_bg_image (widget_t* widget, const char* bg_image);
```

- 参数说明，详见表 4.259。

表 4. 259 time_clock_set_bg_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| bg_image | const char* | 背景图片 |

(4) time_clock_set_hour

- 函数原型

```
ret_t time_clock_set_hour (widget_t* widget, int32_t hour);
```

- 参数说明，详见表 4.260。

表 4. 260 time_clock_set_hour 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| hour | int32_t | 小时的值 |

(5) time_clock_set_hour_image

- 函数原型

```
ret_t time_clock_set_hour_image (widget_t* widget, const char* hour);
```

- 参数说明，详见表 4.261。

表 4. 261 time_clock_set_hour_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |

| | | |
|---|---|---|
| hour | const char* | 小时的图片 |

(6) time_clock_set_image

- 函数原型

```
ret_t time_clock_set_image (widget_t* widget, const char* image);
```

- 参数说明，详见表 4.262。

表 4. 262 time_clock_set_image 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| image | const char* | 图片 |

(7) time_clock_set_minute

- 函数原型

```
ret_t time_clock_set_minute (widget_t* widget, int32_t minute);
```

- 参数说明，详见表 4.263。

表 4. 263 time_clock_set_minute 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| minute | int32_t | 分钟的值 |

(8) time_clock_set_minute_image

- 函数原型

```
ret_t time_clock_set_minute_image (widget_t* widget, const char* minute_image);
```

- 参数说明，详见表 4.264。

表 4. 264 time_clock_set_minute_image 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| minute_image | const char* | 分钟的图片 |

(9) time_clock_set_second

- 函数原型

```
ret_t time_clock_set_second (widget_t* widget, int32_t second);
```

- 参数说明，详见表 4.265。

表 4. 265 time_clock_set_second 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| second | int32_t | 秒的值 |

(10) time_clock_set_second_image

- 函数原型

```
ret_t time_clock_set_second_image (widget_t* widget, const char* second_image);
```

- 参数说明，详见表 4.266。

表 4. 266 time_clock_set_second_image 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| second_image | const char* | 秒的图片 |

### 2. 属性

time_clock 还提供了下面属性，详见表 4.267。

表 4. 267 time_clock 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| bg_image | char* | 背景图片 |
| hour | int32_t | 小时 |
| hour_image | char* | 时针图片 |
| image | char* | 中心图片 |
| minute | int32_t | 分钟 |
| minute_image | char* | 分针图片 |
| second | int32_t | 秒 |
| second_image | char* | 秒针图片 |

### 3. 示例

- 在 xml 中，使用"time_clock"标签创建模拟时钟控件，详见程序清单 4.68。

程序清单 4. 68 在 xml 中创建 time_clock

```
<!-- awtk\demos\assets\raw\ui\time_clock.xml -->
<window anim_hint="htranslate" >
  <time_clock x="c" y="m" w="300" h="300" bg_image="clock_bg" image="clock"
    hour_image="clock_hour" minute_image="clock_minute" second_image="clock_second"/>
</window>
```

- 在 C 代码中，使用函数 time_clock_create 创建模拟时钟控件，详见程序清单 4.69。

程序清单 4. 69 在 C 代码中创建 time_clock

```
widget_t* tc = time_clock_create(win, 10, 10, 240, 240);
time_clock_set_image(tc, "clock");
time_clock_set_bg_image(tc, "clock_bg");
time_clock_set_hour_image(tc, "clock_hour");
time_clock_set_minute_image(tc, "clock_minute");
time_clock_set_second_image(tc, "clock_second");
```

### 4.5.17 digit_clock

数字时钟控件。digit_clock_t 是 widget_t 的子类控件，widget_t 的函数均适用于 digit_clock_t 控件。

1. 函数

digit_clock 提供的函数详见表 4.268。

表 4. 268 digit_clock 函数列表

| 函数名称 | 说明 |
| --- | --- |
| digit_clock_cast | 转换为 digit_clock 对象（供脚本语言使用） |
| digit_clock_create | 创建 digit_clock 对象 |
| digit_clock_set_format | 设置显示格式 |

(1) digit_clock_cast

- 函数原型

```
widget_t* digit_clock_cast (widget_t* widget);
```

- 参数说明，详见表 4.269。

表 4. 269 digit_clock_cast 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | digit_clock 对象 |
| widget | widget_t* | digit_clock 对象 |

(2) digit_clock_create

- 函数原型

```
widget_t* digit_clock_create (widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
```

- 参数说明，详见表 4.270。

表 4. 270 digit_clock_create 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | widget_t* | 对象 |
| parent | widget_t* | 父控件 |
| x | xy_t | x 坐标 |
| y | xy_t | y 坐标 |
| w | wh_t | 宽度 |

| | | |
|---|---|---|
| h | wh_t | 高度 |

(3) digit_clock_set_format

- 函数原型

```
ret_t digit_clock_set_format (widget_t* widget, const char* format);
```

- 参数说明，详见表 4.271。

表 4. 271 digit_clock_set_format 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| widget | widget_t* | 控件对象 |
| format | const char* | 格式 |

## 2. 属性

digit_clock 还提供了下面属性，详见表 4.272。

表 4. 272 digit_clock 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| format | char* | 显示格式（取值详见表 4.273） |

表 4.273 format 取值

| Format 取值 | 说明 |
|---|---|
| Y | 代表年(完整显示) |
| M | 代表月(1-12) |
| D | 代表日(1-31) |
| h | 代表时(0-23) |
| m | 代表分(0-59) |
| s | 代表秒(0-59) |
| YY | 代表年(只显示末两位) |
| MM | 代表月(01-12) |
| DD | 代表日(01-31) |
| hh | 代表时(00-23) |
| mm | 代表分(00-59) |
| ss | 代表秒(00-59) |

## 3. 示例

- 在 xml 中，使用"digit_clock"标签创建数字时钟控件，详见程序清单 4.70。

程序清单 4. 70 在 xml 中创建 digit_clock

```
<!-- awtk\demos\assets\raw\ui\digit_clock.xml -->
<window anim_hint="htranslate" children_layout="default(c=1,h=30,s=10)">
  <digit_clock format="h:m" />
  <digit_clock format="hh:mm" />
```

```
  <digit_clock format="h:m:s"/>
  <digit_clock format="hh:mm:ss"/>
  <digit_clock format="Y/M/D"/>
  <digit_clock format="YY-MM-DD"/>
  <digit_clock format="Y/M/D h:mm:ss"/>
  <digit_clock format="YY/MM/DD h:mm:ss"/>
</window>
```

- 在 C 代码中，使用函数 digit_clock_create 创建数字时钟控件，详见程序清单 4.71。

程序清单 4. 71 在 C 代码中创建 digit_clock

```
widget_t* tc = digit_clock_create(win, 10, 10, 240, 30);
digit_clock_set_format(tc, "YY/MM/DD h:mm:ss");
```

## 4.6 widget

widget_t 是所有控件、窗口和窗口管理器的基类。widget_t 也是一个容器，可放其它 widget_t 到它的内部，形成一个树形结构。通常 widget_t 通过一个矩形区域向用户呈现一些信息，接受用户的输入，并据此做出适当反应。它负责控件的生命周期、通用状态、事件分发和 Style 的管理。 本类提供的接口（函数和属性）除非特别说明，一般都适用于子类控件。

### 4.6.1 函数

widget 提供的函数详见表 4.274，具体的 API 函数参数请看：awtk\docs\manual\widget_t.md，这里不累赘了。

表 4. 274 widget 函数列表

| 函数 | 说明 |
| --- | --- |
| widget_add_child | 加入一个子控件 |
| widget_add_timer | 创建定时器 |
| widget_add_value | 增加控件的值 |
| widget_animate_value_to | 设置控件的值（以动画形式变化到指定的值） |
| widget_cast | 转换为 widget 对象（供脚本语言使用） |
| widget_child | 查找指定名称的子控件(同 widget_lookup(widget, name, FALSE))。 |
| widget_child_on | 为指定名称的子控件注册指定事件的处理函数 |
| widget_clone | 克隆 |
| widget_count_children | 获取子控件的个数 |
| widget_create_animator | 创建动画 |
| widget_destroy | 销毁控件 |
| widget_destroy_animator | 销毁动画 |
| widget_destroy_children | 销毁全部子控件 |
| widget_dispatch | 分发一个事件 |
| widget_equal | 判断两个 widget 是否相同 |
| widget_foreach | 遍历当前控件及子控件 |
| widget_get_child | 获取指定索引的子控件 |
| widget_get_prop | 获取控件指定属性的值 |

| | |
|---|---|
| widget_get_prop_bool | 获取布尔格式的属性 |
| widget_get_prop_default_value | 获取控件指定属性的缺省值（在持久化控件时，无需保存缺省值） |
| widget_get_prop_int | 获取整数格式的属性 |
| widget_get_prop_str | 获取字符串格式的属性 |
| widget_get_text | 获取控件的文本 |
| widget_get_type | 获取当前控件的类型名称 |
| widget_get_value | 获取控件的值只是对 widget_get_prop 的包装，值的意义由子类控件决定 |
| widget_get_window | 获取当前控件所在的窗口 |
| widget_get_window_manager | 获取当前的窗口管理器 |
| widget_grab | 让指定子控件抓住事件 |
| widget_index_of | 获取控件在父控件中的索引编号 |
| widget_insert_child | 插入子控件到指定的位置 |
| widget_invalidate | 请求重绘指定的区域，如果 widget->dirty 已经为 TRUE，直接返回 |
| widget_invalidate_force | 请求强制重绘控件 |
| widget_is_window_opened | 判断当前控件所在的窗口是否已经打开 |
| widget_layout | 布局当前控件及子控件 |
| widget_layout_children | layout 子控件 |
| widget_load_asset | 加载资源 |
| widget_load_image | 加载图片 |
| widget_lookup | 查找指定名称的子控件（返回第一个） |
| widget_lookup_by_type | 查找指定类型的子控件（返回第一个） |
| widget_move | 移动控件 |
| widget_move_resize | 移动控件并调整控件的大小 |
| widget_off | 注销指定事件的处理函数 |
| widget_off_by_func | 注销指定事件的处理函数（仅用于辅助实现脚本绑定） |
| widget_on | 注册指定事件的处理函数 |
| widget_pause_animator | 暂停动画 |
| widget_remove_child | 移出指定的子控件（并不销毁） |
| widget_resize | 调整控件的大小 |
| widget_restack | 调整控件在父控件中的位置序数 |
| widget_set_animation | 设置控件的动画参数（仅用于在 UI 文件使用） |
| widget_set_animator_time_scale | 设置动画的时间倍率，<0: 时间倒退，<1: 时间变慢，>1 时间变快。 |
| widget_set_children_layout | 设置子控件的布局参数 |
| widget_set_cursor | 设置鼠标指针的图片名 |
| widget_set_enable | 设置控件的可用性 |
| widget_set_floating | 设置控件的 floating 标志 |
| widget_set_focused | 设置控件的是否聚焦 |
| widget_set_name | 设置控件的名称 |
| widget_set_opacity | 设置控件的不透明度 |
| widget_set_prop | 设置控件指定属性的值 |
| widget_set_prop_bool | 设置布尔格式的属性 |
| widget_set_prop_int | 设置整数格式的属性 |
| widget_set_prop_str | 设置字符串格式的属性 |

| | |
|---|---|
| widget_set_self_layout | 设置控件自己的布局参数 |
| widget_set_self_layout_params | 设置控件自己的布局(缺省布局器)参数(过时,用 widget_set_self_layout) |
| widget_set_sensitive | 设置控件是否接受用户事件 |
| widget_set_state | 设置控件的状态 |
| widget_set_text | 设置控件的文本 |
| widget_set_text_utf8 | 设置控件的文本 |
| widget_set_tr_text | 获取翻译之后的文本，然后调用 widget_set_text |
| widget_set_value | 设置控件的值 |
| widget_set_visible | 设置控件的可见性 |
| widget_start_animator | 播放动画 |
| widget_stop_animator | 停止动画（控件的相应属性回归原位） |
| widget_to_global | 将控件内的本地坐标转换成全局坐标 |
| widget_to_local | 将屏幕坐标转换成控件内的本地坐标，即相对于控件左上角的坐标 |
| widget_to_screen | 将控件内的本地坐标转换成屏幕上的坐标 |
| widget_ungrab | 让指定子控件放弃抓住事件 |
| widget_unload_asset | 卸载资源 |
| widget_use_style | 启用指定的主题 |

### 4.6.2 属性

widget 还提供了下面属性，详见表 4.275。

表 4. 275 widget 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| animation | char* | 动画参数请参考控件动画 |
| astyle | style_t* | Style 对象 |
| auto_created | bool_t | 是否由父控件自动创建 |
| can_not_destroy | uint16_t | 标识控件目前不能被销毁（比如正在分发事件），如果此时调用 widget_destroy，自动异步处理 |
| children | array_t* | 全部子控件 |
| children_layout | children_layouter_t* | 子控件布局器请参考控件布局参数 |
| custom_props | custom_props_t* | 自定义属性 |
| destroying | bool_t | 标识控件正在被销毁 |
| dirty | bool_t | 标识控件是否需要重绘 |
| emitter | emitter_t* | 事件发射器 |
| enable | bool_t | 启用/禁用状态 |
| floating | bool_t | 标识控件是否启用浮动布局,不受父控件的 children_layout 的控制 |
| focused | bool_t | 是否得到焦点 |
| grab_widget | widget_t* | grab 事件的子控件 |
| h | wh_t | 高度 |
| key_target | widget_t* | 接收按键事件的子控件 |
| name | char* | 控件名字 |
| need_relayout_children | bool_t | 标识控件是否需要重新 layout 子控件 |

| | | |
|---|---|---|
| opacity | uint8_t | 不透明度（0-255），0 完全透明，255 完全不透明 |
| parent | widget_t* | 父控件 |
| self_layout | self_layouter_t* | 控件布局器请参考控件布局参数 |
| sensitive | bool_t | 是否接受用户事件 |
| state | uint8_t | 控件的状态（取值参考 widget_state_t） |
| style | char* | style 的名称 |
| target | widget_t* | 接收事件的子控件 |
| text | wstr_t | 文本用途视具体情况而定 |
| tr_text | char* | 保存用于翻译的字符串 |
| visible | bool_t | 是否可见 |
| vt | widget_vtable_t | 虚函数表 |
| w | wh_t | 宽度 |
| x | xy_t | x 坐标（相对于父控件的 x 坐标） |
| y | xy_t | y 坐标（相对于父控件的 y 坐标） |

### 4.6.3 事件

widget 还提供了下面事件，详见表 4.276。

表 4. 276 widget 事件列表

| 事件名称 | 类型 | 说明 |
|---|---|---|
| EVT_WILL_MOVE | event_t | 控件移动前触发 |
| EVT_MOVE | event_t | 控件移动后触发 |
| EVT_WILL_RESIZE | event_t | 控件调整大小前触发 |
| EVT_RESIZE | event_t | 控件调整大小后触发 |
| EVT_WILL_MOVE_RESIZE | event_t | 控件移动并调整大小前触发 |
| EVT_MOVE_RESIZE | event_t | 控件移动并调整大小后触发 |
| EVT_PROP_WILL_CHANGE | prop_change_event_t | 控件属性改变前触发（通过 set_prop 设置属性，才会触发） |
| EVT_PROP_CHANGED | prop_change_event_t | 控件属性改变后触发（通过 set_prop 设置属性，才会触发） |
| EVT_BEFORE_PAINT | paint_event_t | 控件绘制前触发 |
| EVT_AFTER_PAINT | paint_event_t | 控件绘制完成时触发 |
| EVT_FOCUS | event_t | 控件得到焦点时触发 |
| EVT_BLUR | event_t | 控件失去焦点时触发 |
| EVT_WHEEL | wheel_event_t | 鼠标滚轮事件 |
| EVT_POINTER_LEAVE | pointer_event_t | 鼠标指针离开控件时触发 |
| EVT_POINTER_ENTER | pointer_event_t | 鼠标指针进入控件时触发 |
| EVT_KEY_DOWN | pointer_event_t | 键按下事件 |
| EVT_KEY_UP | pointer_event_t | 键释放事件 |
| EVT_POINTER_DOWN | pointer_event_t | 指针设备按下事件 |
| EVT_POINTER_DOWN_ABORT | pointer_event_t | 取消指针设备按下事件 |
| EVT_POINTER_MOVE | pointer_event_t | 指针设备移动事件 |
| EVT_POINTER_UP | pointer_event_t | 指针设备释放事件 |

| | | |
|---|---|---|
| EVT_DESTROY | event_t | 控件销毁时触发 |

# 第5章 动画

✍**本章导读**

一个炫酷的 GUI 界面是离不开动画的，通过使用动画给人用户眼前一亮的感觉。AWTK 提供了多种动画机制，可以让我们的 GUI 变得更加的形象、流畅。

## 5.1 简介

AWTK 为我们提供了两大类动画：窗口动画和控件动画，下面我们一起看看如何使用这两种动画。

## 5.2 窗口动画

窗口动画是现代 GUI 最基本的功能之一，在窗口打开或关闭时，引入一个过渡动画，让用户感觉这个过程是流畅的。窗口动画的基本原理很简单：在打开或关闭窗口时，把前后两个窗口预先绘制到两张内存图片上，按照指定规则显示两张图片，形成动画效果。

### 5.2.1 动画类型

窗口本身只需指定期望的动画类型，由窗口管理器负责在适当的时候（如打开和关闭窗口时），创建窗口动画并让窗口动画绘制到屏幕上。在窗口动画期间，窗口管理器会禁止窗口本身的绘制，并忽略所有输入事件。目前支持的窗口动画详见表 5.1。

表 5.1 窗口动画类型

| 窗口动画类型 | 说明 |
| --- | --- |
| htranslate: | 左右平移动画（适合窗口） |
| vtranslate: | 上下平移动画（适合窗口） |
| center_scale: | 缩放动画（适合对话框，没有硬件时加速慎用） |
| top_to_bottom: | 顶部向下弹出动画（适合对话框） |
| bottom_to_top: | 底部向上弹出动画（适合对话框） |
| fade: | 淡入淡出动画（适合提示信息） |

### 5.2.2 示例

给窗口或对话框指定动画效果，只需设置窗口或对话框的 anim_hint 属性即可，下面以 awtk\bin\demoui.exe 中的主界面 main.xml 为例，代码详见程序清单 5.1。

程序清单 5.1 main.xml

```
<!-- awtk\demos\assets\raw\ui\main.xml -->
<window closable="no" text="Desktop" anim_hint="htranslate">
  <slide_view x="0" y="0" w="100%" h="100%">
    <view x="0" y="0" w="100%" h="100%" children_layout="default(c=2,r=8,m=5,s=5)">
      <button name="open:basic" text="Basic"/>
      <button name="open:button" text="Buttons"/>
      <button name="open:edit" text="Edits"/>
      <button name="open:keyboard" text="KeyBoard"/>
```

```
        <button name="open:list_view" text="ListView"/>
        <button name="open:slide_view" text="SlideView"/>
        <button name="open:animation" text="Animate Window"/>
        <button name="open:animator" text="Animate Widget"/>
        <button name="open:tab_control" text="Tab Control"/>
        <button name="open:combo_box" text="ComboBox"/>
        <button name="open:rich_text" text="RichText"/>
        <button name="open:color_picker" text="Color Picker"/>
    </view>
    ...
</window>
```

我们还可以指定动画时长（duration，单位为毫秒），格式与函数调用类似，不过参数用 name=value 的形式，如：

```
anim_hint="center_scale(duration=300)"
```

## 5.3 控件动画

控件动画是一种很常见的动画，常用于入场动画、离场动画、装饰用户界面和吸引用户注意力等。

### 5.3.1 动画类型

AWTK 目前支持的控件动画类型详见表 5.2。

表 5.2 控件动画类型

| 控件动画类型 | 说明 |
| --- | --- |
| move | 通过改变控件的位置形成动画效果 |
| value | 通过改变控件的值形成动画效果 |
| opacity | 通过改变控件的透明度形成动画效果 |
| scale | 通过改变控件的缩放比例形成动画效果（目前需要 vgcanvas） |
| rotation | 通过改变控件的旋转角度形成动画效果（目前需要 vgcanvas） |
| 其它任何数值型的属性 | 如 x/y/w/h 等等 |

### 5.3.2 特色

- 支持逆向模式
- 支持停止和暂停
- 支持定时（延迟）执行
- 支持重复模式（可指定次数）
- 支持往返模式（可指定次数）
- 支持多种插值算法（如加速和减速等）
- 支持同一控件多个动画并行执行
- 支持同一控件多个动画串行执行
- 支持时间倍率，让时间变快和变慢
- 支持按名称去开始、暂停、停止和销毁动画

### 5.3.3 函数

AWTK 提供了的函数接口详见表 5.3。

表 5.3 控件动画函数接口

| 函数名称 | 说明 |
| --- | --- |
| widget_animator_destroy | 销毁 animator 对象 |
| widget_animator_init | 初始化仅供子类内部使用 |
| widget_animator_off | 注销指定事件的处理函数 |
| widget_animator_on | 注册指定事件的处理函数 |
| widget_animator_pause | 暂停动画 |
| widget_animator_set_destroy_when_done | 设置完成时是否自动销毁动画对象（缺省销毁） |
| widget_animator_set_name | 设置名称 |
| widget_animator_set_repeat | 设置为重复模式 |
| widget_animator_set_reversed | 设置为逆向模式 |
| widget_animator_set_time_scale | 设置时间倍率，用于实现时间加速减速和停滞的功能 |
| widget_animator_set_yoyo | 设置为 yoyo 模式 |
| widget_animator_start | 启动动画 |
| widget_animator_stop | 停止动画 |

(1) widget_animator_destroy

- 函数原型

```
ret_t widget_animator_destroy (widget_animator_t* animator);
```

- 参数说明，详见表 5.4。

表 5.4 widget_animator_destroy 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |

(2) widget_animator_init

- 函数原型

```
ret_t widget_animator_init (widget_animator_t* animator, widget_t* widget, uint32_t duration, uint32_t delay, easing_func_t    easing);
```

- 参数说明，详见表 5.5。

表 5.5 widget_animator_init 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| widget | widget_t* | 控件对象 |
| duration | uint32_t | 动画持续时间 |

| | | |
|---|---|---|
| delay | uint32_t | 动画执行时间 |
| easing | easing_func_t | 插值函数 |

(3) widget_animator_off

- 函数原型

```
ret_t widget_animator_off (widget_animator_t* animator, uint32_t id);
```

- 参数说明，详见表 5.6。

表 5.6 widget_animator_off 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| id | uint32_t | widget_animator_on 返回的 ID |

(4) widget_animator_on

- 函数原型

```
uint32_t widget_animator_on (widget_animator_t* animator, event_type_t type , event_func_t on_event, void* ctx);
```

- 参数说明，详见表 5.7。

表 5.7 widget_animator_on 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | uint32_t | 返回 id，用于 widget_animator_off |
| animator | widget_animator_t* | 动画对象 |
| type | event_type_t | 事件类型 |
| on_event | event_func_t | 事件处理函数 |
| ctx | void* | 事件处理函数上下文 |

(5) widget_animator_pause

- 函数原型

```
ret_t widget_animator_pause (widget_animator_t* animator);
```

- 参数说明，详见表 5.8。

表 5.8 widget_animator_pause 参数说明

| 参数 | 类型 | 说明 |
|---|---|---|
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |

(6) widget_animator_set_destroy_when_done

- 函数原型

```
ret_t widget_animator_set_destroy_when_done (widget_animator_t* animator, bool_t destroy_when_done);
```

- 参数说明，详见表 5.9。

表 5.9 widget_animator_set_destroy_when_done 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| destroy_when_done | bool_t | 完成时是否自动销毁动画对象 |

(7) widget_animator_set_name

- 函数原型

```
ret_t widget_animator_set_name (widget_animator_t* animator, const char* name);
```

- 参数说明，详见表 5.10。

表 5.10 widget_animator_set_name 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| name | const char* | 设置名称 |

(8) widget_animator_set_repeat

- 函数原型

```
ret_t widget_animator_set_repeat (widget_animator_t* animator, uint32_t repeat_times);
```

- 参数说明，详见表 5.11。

表 5.11 widget_animator_set_repeat 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| repeat_times | uint32_t | 重复的次数 |

(9) widget_animator_set_reversed

- 函数原型

```
ret_t widget_animator_set_reversed (widget_animator_t* animator, bool_t value);
```

- 参数说明，详见表 5.12。

表 5.12 widget_animator_set_reversed 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| value | bool_t | 是否为逆向模式 |

(10) widget_animator_set_time_scale

- 函数原型

```
ret_t widget_animator_set_time_scale (widget_animator_t* animator, float_t time_scale);
```

- 参数说明，详见表 5.13。

表 5.13 widget_animator_set_time_scale 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| time_scale | float_t | 时间倍率 |

(11) widget_animator_set_yoyo

- 函数原型

```
ret_t widget_animator_set_yoyo (widget_animator_t* animator, uint32_t yoyo_times);
```

- 参数说明，详见表 5.14。

表 5.14 widget_animator_set_yoyo 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |
| yoyo_times | uint32_t | yoyo 的次数，往返视为两次 |

(12) widget_animator_start

- 函数原型

```
ret_t widget_animator_start (widget_animator_t* animator);
```

- 参数说明，详见表 5.15。

表 5.15 widget_animator_start 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |

(13) widget_animator_stop

- 函数原型

```
ret_t widget_animator_stop (widget_animator_t* animator);
```

- 参数说明，详见表 5.16。

表 5.16 widget_animator_stop 参数说明

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| 返回值 | ret_t | 返回 RET_OK 表示成功，否则表示失败 |
| animator | widget_animator_t* | 动画对象 |

### 5.3.4 XML 参数

1. 公共参数，详见表 5.17

表 5.17 公共参数

| 参数 | 说明 |
| --- | --- |
| name | 动画名称（缺省为动画的类型如 move） |
| delay | 延迟启动时间（毫秒） |
| duration | 时长（毫秒） |
| easing | 插值算法（见后面描述） |
| yoyo_times | 往返的次数（x2），为 0 视为永久播放 |
| repeat_times | 重复的次数，为 0 视为永久播放 |
| auto_start | 创建后自动启动（缺省为 true） |
| auto_destroy | 完成后自动销毁（缺省为 true） |

2. widget_animator_move 动画的参数，详见表 5.18

表 5.18 move 动画参数

| 参数 | 说明 |
| --- | --- |
| x_from | x 起始位置 |
| y_from | y 起始位置 |
| x_to | x 结束位置 |
| y_to | y 结束位置 |

3. widget_animator_value 动画的参数，详见表 5.19

表 5.19 value 动画参数

| 参数 | 说明 |
| --- | --- |
| from | 起始值 |
| to | 结束值 |

4. widget_animator_opacity 动画的参数，详见表 5.20

表 5.20 opacity 动画参数

| 参数 | 说明 |
| --- | --- |
| from | 起始值（0-255） |
| to | 结束值（0-255） |

5. widget_animator_scale 动画的参数，详见表 5.21

表 5.21 scale 动画参数

| 参数 | 说明 |
| --- | --- |
| x_from | x 方向缩放起始值 |
| y_from | y 方向缩放起始值 |
| x_to | x 方向缩放结束值 |
| y_to | y 方向缩放结束值 |

6. widget_animator_rotation 动画的参数，详见表 5.22

表 5.22 rotation 动画参数

| 参数 | 说明 |
| --- | --- |
| from | 起始值（弧度） |
| to | 结束值（弧度） |

7. 其它数值型属性动画的参数，详见表 5.23

表 5.23 其他数值型动画参数

| 参数 | 说明 |
| --- | --- |
| from | 起始值 |
| to | 结束值 |

### 5.3.5 插值算法名称（easing）

easing 可取的值详见表 5.24。

表 5.24 easing 取值

| 类别 | easing 取值 |
| --- | --- |
| linear | linear |
| quadratic | quadratic_in |
| | quadratic_out |
| | quadratic_inout |
| cubic | cubic_in |
| | cubic_out |
| sin | sin_in |
| | sin_out |
| | sin_inout |
| pow | pow_in |
| | pow_out |
| | pow_inout |
| circular | circular_in |
| | circular_out |
| | circular_inout |
| elastic | elastic_in |
| | elastic_out |
| | elastic_inout |
| back | back_in |
| | back_out |
| | back_inout |
| bounce | bounce_inout |
| | bounce_out |
| | bounce_inout |

### 5.3.6 示例

既可以在 UI 文件的 XML 中，也可以使用 C 代码中创建动画，接下来我们将分别介绍如何在 XML 和 C 代码中创建动画。

#### 1. 在 XML 中

在 XML 中可以使用 animation 参数创建动画，多个参数可以用";"分隔，此方法最为简单，后续可以在程序中通过动画名字，对动画进行启动和暂停等控制。下面以 awkt\bin\demoui.exe 中的 Animate Widget 页面为例，代码详见程序清单 5.2。

程序清单 5.2 在 XML 中创建动画

```
<!-- awtk\demos\assets\raw\ui\animator.xml -->
<window>
  <image name="fade" image="logo" x="center" y="10" w="200" h="60" opacity="0"
animation="opacity(from=50, to=255, yoyo_times=1000, duration=1000)"/>
  <image name="fade" image="logo" x="center" y="80" w="200" h="60" opacity="0"
animation="opacity(from=50, to=255, yoyo_times=1000, duration=1000, delay=1000)" />
  <image name="fade" image="logo" x="center" y="150" w="200" h="60" opacity="0"
animation="opacity(from=50, to=255, yoyo_times=1000, duration=1000, delay=2000)" />
  <image name="move" image="logo" x="0" y="middle:60" w="200" h="60" animation="move(x_from=0,
x_to=200, yoyo_times=1000, duration=1000, delay=3000)"/>
  <progress_bar name="value" x="center" y="middle" w="90%" h="40" animation="value(from=50, to=100,
yoyo_times=1000, duration=1000, delay=4000)"/>
  <button name="close" x="center" y="bottom:10" w="25%" h="30" text="Close"/>
</window>
```

#### 2. 在 C 代码中

在缺省情况下，动画创建后自动启动，完成后自动销毁。可以指定 auto_start=false 禁止创建后自动启动，指定参数 auto_destroy=false 禁止完成时自动销毁（控件销毁时仍然会自动销毁）。在 C 代码中有两种方式创建动画，下面以 awkt\bin\demo_animator.exe 为例：

(1)采用 widget_animator_xxx_create，其中 xxx 可取值 move/value/opacity/scale/rotation，此方法比较麻烦，不再推荐使用，代码详见程序清单 5.3。

程序清单 5.3 使用 widget_animator_xxx_create 创建动画

```
//awtk\demos\demo_animator_app.c
ret_t application_init() {
  ...
  animator = widget_animator_move_create(image, 1000, delay, EASING_SIN_INOUT);
  widget_animator_move_set_params(animator, image->x, image->y, image->x + 100, image->y + 100);
  widget_animator_set_destroy_when_done(animator, FALSE);
  widget_animator_set_repeat(animator, times);
  widget_animator_start(animator);
  ...
}
```

(2)通过函数 widget_create_animator 创建动画，推荐使用，代码详见程序清单 5.4。

程序清单 5.4 使用 widget_create_animator 创建动画

```
//awtk\demos\demo_animator_app.c
ret_t application_init() {
  ...
  widget_create_animator(progress_bar,
                         "value(from=50, to=100, duration=500, yoyo_times=1000, delay=1000)");
  widget_create_animator(opacity, "opacity(to=50, duration=500, yoyo_times=1000)");
  widget_create_animator(image6, "opacity(to=50, duration=500, yoyo_times=1000, delay=1000)");
  widget_create_animator(image2, "rotation(to=6.28, yoyo_times=0, duration=1000, easing=sin_out)");
  widget_create_animator(image3, "scale(x_to=-1, yoyo_times=1000, duration=1000, easing=sin_out)");
  widget_create_animator(image4,
                         "scale(x_to=2, y_to=2, yoyo_times=1000, duration=1000, easing=sin_out)");
  widget_create_animator(image5, "y(to=400, duration=1000, easing=sin_out)");
  widget_create_animator(image5, "opacity(to=0, duration=500, yoyo_times=1000, delay=1000)");
  ...
}
```

# 第6章 画布

✍**本章导读**

AWTK 提供了各种绘制图像的 API 接口，方便程序员绘制直线、矩形、椭圆等形状。也可以根据 AWTK 提供的 API 接口绘制自己想要的控件，例如饼图、柱状图等。

## 6.1 简介

vgcanvas_t 矢量图画布抽象基类。具体实现时可以使用 agg、nanovg, cairo 和 skia 等方式。cairo 和 skia 体积太大，不适合嵌入式平台，但在 PC 平台也是一种选择。目前我们只提供了基于 nanovg 的实现，支持软件渲染和硬件渲染。我们对 nanovg 进行了一些改进：

- 可以用 agg/agge 实现软件渲染（暂时不支持文本绘制）
- 可以用 bgfx 使用 DirectX（Windows 平台）、Metal（iOS 平台）和使用 OpenGL，Vulkan（Linux 平台）硬件加速

vgcanvas 的 UML 图详见图 6.1。

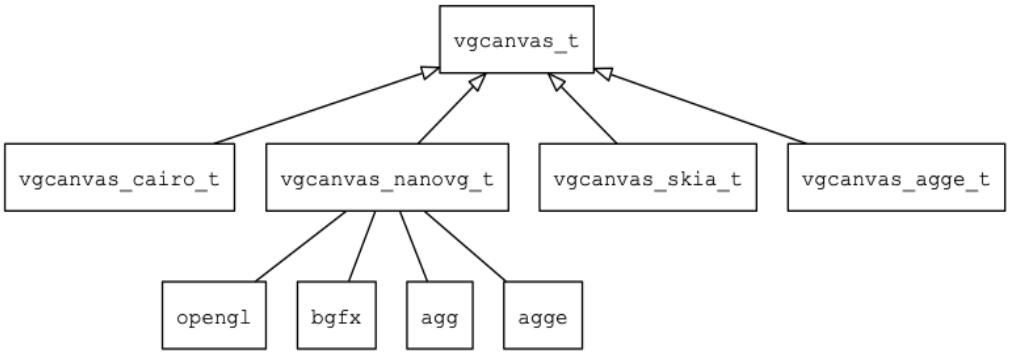



图 6.1 vgcanvas 的 UML 图

## 6.2 函数

vgcanvas 提供的函数详见表 6.1，具体的 API 函数参数请看：awtk\docs\manual\vgcanvas_t.md，这里不累赘了。

表 6.1 vgcanvas 函数列表

| 函数 | 说明 |
| --- | --- |
| vgcanvas_arc | 创建弧度/曲线（用于创建圆形或部分圆） |
| vgcanvas_arc_to | 创建两切线之间的弧/曲线 |
| vgcanvas_begin_frame | 开始绘制，系统内部调用 |
| vgcanvas_begin_path | 起始一条路径，或重置当前路径 |
| vgcanvas_bezier_to | 创建三次方贝塞尔曲线 |
| vgcanvas_clear_rect | 在给定的矩形内清除指定的矩形 |
| vgcanvas_clip_rect | 从原始画布剪切矩形区域 |
| vgcanvas_close_path | 创建从当前点到起始点的路径 |
| vgcanvas_create | 创建 vgcanvas 对象 |
| vgcanvas_destroy | 销毁 vgcanvas 对象 |

| | |
|---|---|
| vgcanvas_draw_icon | 绘制图标 |
| vgcanvas_draw_image | 绘制图片 |
| vgcanvas_ellipse | 绘制椭圆 |
| vgcanvas_end_frame | 结束绘制系统内部调用 |
| vgcanvas_fill | 填充多边形 |
| vgcanvas_fill_text | 在画布上绘制文本 |
| vgcanvas_flush | 刷新画布 |
| vgcanvas_is_point_in_path | 检查点是否在当前路径中 |
| vgcanvas_line_to | 添加一个新点，然后在画布中创建从该点到最后指定点的线条 |
| vgcanvas_measure_text | 返回包含指定文本宽度的对象 |
| vgcanvas_move_to | 把路径移动到画布中指定点，不创建线条 |
| vgcanvas_paint | 用图片填充/画多边形（可能存在可移植性问题，除非必要请勿使用） |
| vgcanvas_quad_to | 创建二次方贝塞尔曲线 |
| vgcanvas_rect | 创建矩形 |
| vgcanvas_reinit | 重新初始化，系统内部调用 |
| vgcanvas_reset | 重置 vgcanvas |
| vgcanvas_restore | 恢复 vgcanvas |
| vgcanvas_rotate | 旋转 |
| vgcanvas_rounded_rect | 创建圆角矩形 |
| vgcanvas_save | 保存当前的状态如颜色和矩阵等信息 |
| vgcanvas_scale | 缩放 |
| vgcanvas_set_antialias | 设置是否启用反走样 |
| vgcanvas_set_fill_color | 设置填充颜色 |
| vgcanvas_set_font | 设置字体 |
| vgcanvas_set_font_size | 设置字体大小 |
| vgcanvas_set_global_alpha | 设置全局透明度 |
| vgcanvas_set_line_cap | 设置线条的结束端点样式 |
| vgcanvas_set_line_join | 设置两条线相交时，所创建的拐角类型 |
| vgcanvas_set_line_width | 设置当前的线条宽度 |
| vgcanvas_set_fill_linear_gradient | 设置填充颜色为线性渐变色 |
| vgcanvas_set_fill_radial_gradient | 设置填充颜色为径向渐变色 |
| vgcanvas_set_miter_limit | 设置最大斜接长度 |
| vgcanvas_set_stroke_linear_gradient | 设置线条颜色为线性渐变色 |
| vgcanvas_set_stroke_radial_gradient | 设置线条颜色为径向渐变色 |
| vgcanvas_set_stroke_color | 设置触笔颜色 |
| vgcanvas_set_text_align | 设置文本对齐方式 |
| vgcanvas_set_text_baseline | 设置文本垂直对齐的方式 |
| vgcanvas_set_transform | 设置变换矩阵 |
| vgcanvas_stroke | 画线 |
| vgcanvas_transform | 变换矩阵 |
| vgcanvas_translate | 平移 |

## 6.3 属性

vgcanvas 还提供了下面属性，详见表 6.2。

表 6.2 vgcanvas 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| anti_alias | bool_t | 是否启用反走样功能 |
| fill_color | color_t | 填充颜色 |
| font | char* | 字体 |
| font_size | float_t | 字体大小 |
| global_alpha | float_t | 全局 alpha |
| height | wh_t | canvas 的高度 |
| line_cap | char* | line_cap |
| line_join | char* | line_join |
| line_width | float_t | 线宽 |
| miter_limit | float_t | miter_limit |
| ratio | float_t | 显示比例 |
| stroke_color | color_t | 线条颜色 |
| text_align | char* | 文本对齐方式 |
| text_baseline | char* | 文本基线 |
| w | wh_t | canvas 的宽度 |

## 6.4 示例

下面是一个简单的 vgcanvas 示例，代码如下：

```
vgcanvas_t* vg = canvas_get_vgcanvas(c);
vgcanvas_save(vg);
vgcanvas_translate(vg, 0, 100);
vgcanvas_set_line_width(vg, 1);
vgcanvas_set_fill_color(vg, color_init(0xff, 0, 0, 0xff));
vgcanvas_rect(vg, 5, 5, 100, 100);
vgcanvas_fill(vg);
vgcanvas_restore(vg);
```

更多的用法请看： awtk\demos\demo_vg_app.c。

# 第7章 输入法

✍**本章导读**

AWTK 提供的输入法是一款打字超准、速度飞快、外观极炫的，用了之后会让您爱不释手的输入法。AWTK 提供了多种输入键盘，可以根据应用的具体情况灵活搭配选择一种或者多种键盘，还可以根据自己的喜好设置不同的主题。

## 7.1 简介

输入法是 GUI 重要的组件之一，虽然实现起来并不是太复杂，但其涉及的组件比较多，理解起来还是比较困难的，这里介绍一下 AWTK 中输入法的内部架构，详见图 7.1。

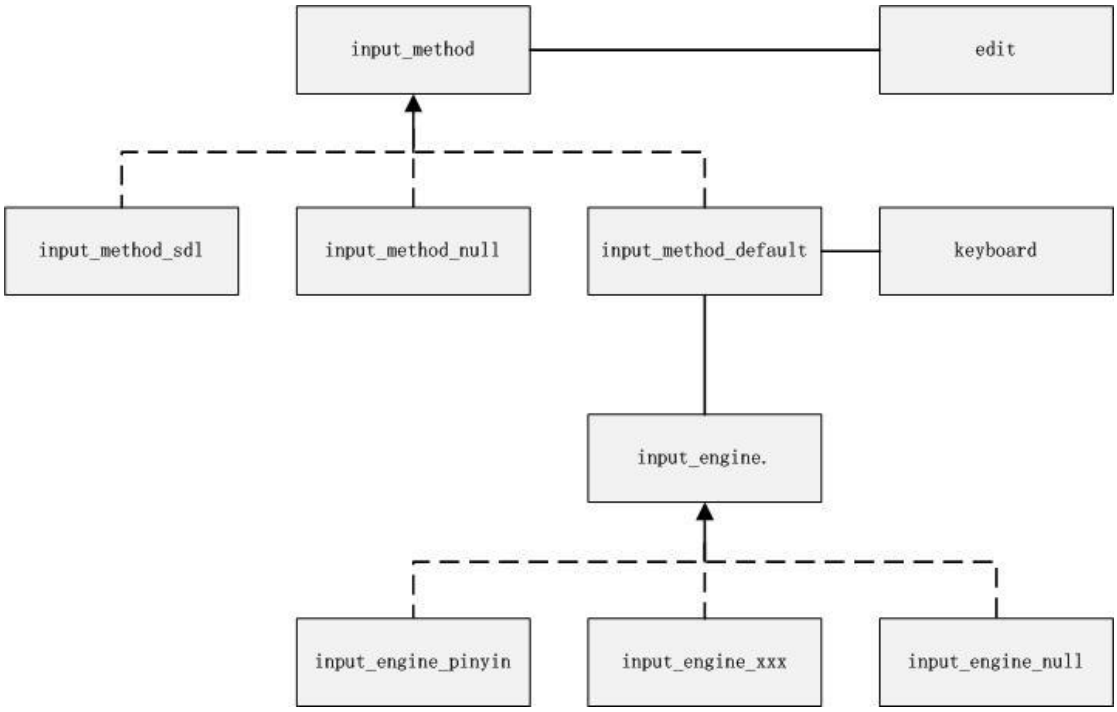



图 7.1 输入法内部架构图

## 7.2 软键盘

在嵌入式系统中，通常没有物理键盘，所以需要在屏幕上实现软键盘。AWTK 中的软键盘是一个普通的窗口，其中的按钮和候选字控件，都是用 AWTK 的 UI 描述文件定义的，可以方便实现各种不同的软键盘。软键盘的描述文件放在 awtk\demos\assets\raw\ui 目录下，文件名以 kb_打头。与普通窗口相比，软键盘有以下不同：

- 被点击时不会影响原有编辑器的焦点
- 不接受按键事件和输入文本事件

## 7.3 键盘类型

AWTK 为了开发方便，提供了多种类型的键盘详见表 7.1。

表 7.1 键盘类型

| 输入类型 | 键盘描述文件 | 说明 |
| --- | --- | --- |
| phone | kb_phone | 电话号码 |
| int | kb_int | 整数 |
| float | kb_float | 浮点数 |
| uint | kb_uint | 非负整数 |
| ufloat | kb_ufloat | 非负浮点数 |
| hex | kb_hex | 16 进制 |
| email | kb_ascii | 邮件地址 |
| password | kb_ascii | 密码 |
| 不输入 | kb_default | 默认 |

## 7.4 加入中文输入法

在有些示例项目中，没有加入输入法，主要是开发板的 flash 不够。如果 flash 够大（不小于 4M 时），可以自行加入：

- 加入 3rd/gpinyin/src 中的代码
- 加入 src/input_engines/input_engine_pinyin.cpp
- 去掉 src/input_engines/input_engine_null.cpp
- include 路径加入 3rd/gpinyin/include

即修改 awtk\SConstruct 文件（加粗字体部分）如下：

```
#awtk\SConstruct
...
#INPUT_ENGINE='null'
INPUT_ENGINE='pinyin'
...
CPPPATH=[TK_ROOT,
   TK_SRC,
   TK_3RD_ROOT,
   joinPath(TK_SRC, 'ext_widgets'),
   joinPath(TK_3RD_ROOT, 'bgfx/bgfx/include'),
   joinPath(TK_3RD_ROOT, 'bgfx/bx/include'),
   joinPath(TK_3RD_ROOT, 'bgfx/bimg/include'),
   joinPath(TK_3RD_ROOT, 'agge'),
   joinPath(TK_3RD_ROOT, 'agg/include'),
   joinPath(TK_3RD_ROOT, 'nanovg'),
   joinPath(TK_3RD_ROOT, 'nanovg/gl'),
   joinPath(TK_3RD_ROOT, 'nanovg/base'),
   joinPath(TK_3RD_ROOT, 'nanovg/agge'),
   joinPath(TK_3RD_ROOT, 'nanovg/bgfx'),
   joinPath(TK_3RD_ROOT, 'SDL/src'),
   joinPath(TK_3RD_ROOT, 'SDL/include'),
   joinPath(TK_3RD_ROOT, 'agge/src'),
```

```
    joinPath(TK_3RD_ROOT, 'agge/include'),
    joinPath(TK_3RD_ROOT, 'gpinyin/include'),
    joinPath(TK_3RD_ROOT, 'libunibreak'),
    TK_TOOLS_ROOT] + OS_CPPPATH


...
SConscriptFiles=NANOVG_BACKEND_PROJS + [
    '3rd/nanovg/SConscript',
    '3rd/glad/SConscript',
    '3rd/gpinyin/SConscript',
    '3rd/libunibreak/SConscript',
    'src/SConscript',
    'tools/common/SConscript',
    'tools/theme_gen/SConscript',
    'tools/font_gen/SConscript',
    'tools/image_gen/SConscript',
    'tools/image_resize/SConscript',
    'tools/res_gen/SConscript',
    'tools/str_gen/SConscript',
    'tools/ui_gen/qt_to_xml/SConscript',
    'tools/ui_gen/xml_to_ui/SConscript',
    'tools/svg_gen/SConscript',
    'demos/SConscript',
    'tests/SConscript'
    ] + OS_PROJECTS
...
```

## 7.5 示例

因为程序不一定需要键盘，如果我们的程序需要使用键盘需要做下面两件事：

- 拷贝 UI 文件：根据需要将 awtk\demos\assets\raw\ui 目录下文件名以 kb_打头 UI 文件拷贝到我们的应用程序 assets\raw\ui 目录下
- 拷贝图片：因为有些键盘还需要使用到图片（使用到的图片可以打开 kb_开头的 UI 文件看看），所以还需要将 awtk\demos\assets\raw\images 下对应的图片拷贝到我们的应用程序 assets\raw\images 目录下

下面以 awkt\bin\demoui.exe 下的 KeyBoard 页面为例，代码详见程序清单 7.1。

程序清单 7.1 keyboard.xml

```
<!-- awtk\demos\assets\raw\ui\keyboard.xml -->
<window anim_hint="htranslate">
  <list_view x="0"    y="0" w="100%" h="-50" item_height="36" auto_hide_scroll_bar="true">
    <scroll_view name="view" x="0"    y="0" w="-12" h="100%">
      <list_item style="empty" children_layout="default(r=1,c=0)">
        <label w="30%" text="Name"/>
```

```
    <edit w="70%" text="" tips="name"/>
  </list_item>
  <list_item style="empty" children_layout="default(r=1,c=0)">
    <label w="30%" text="Desc"/>
    <edit w="70%" text="" tips="desc"/>
  </list_item>
  <list_item style="empty" children_layout="default(r=1,c=0)">
    <label w="30%" text="Int"/>
    <edit w="70%" text="" tips="int" input_type="int"/>
  </list_item>
  <list_item style="empty" children_layout="default(r=1,c=0)">
    <label w="30%" text="UInt"/>
    <edit w="70%" text="" tips="unsigned int" input_type="uint"/>
  </list_item>
  <list_item style="empty" children_layout="default(r=1,c=0)">
    <label w="30%" text="Float"/>
    <edit w="70%" text="" tips="float" input_type="float"/>
  </list_item>
  ...
</window>
```

以 kb_开头的键盘 UI 文件还可以根据实际的屏幕大小适当的修改它的高度以及样式等。如果输入类型不填将默认使用 kb_default 键盘。

# 第8章 多国语言互译

**✍本章导读**

一款应用可能会有不同国家的人使用，为了让应用走向国际化的平台，一个框架可以提供多国语言互译就显得非常重要了。同一个应用要想在不同的国家显示该地的语言，只需要一个配置文件就可以完成，AWTK 就提供了这样的功能。

## 8.1 简介

支持多国语言的存储（Unicode）和显示（字体）是 GUI 必需要做的，但字符串的翻译完全可以用 gettext 等第三方库函数来实现。AWTK 之所以选择自己实现，主要出于以下几点考虑：

- 减少不必要的第三方库的依赖。运行时需要的代码也就几十行，自己实现更简单代码更少
- 方便支持实时切换当前语言。自己实现字符串的翻译，不要应用程序做额外的工作，即可实现实时切换当前语言

## 8.2 动态字符串翻译

我们采用 XML 文件（UTF-8）保存字符串的各个语言的对应关系，方便程序员和翻译人员进行编辑，下面以 awtk\bin\demoui.exe 中的 locale 页面为例。

(1)编写 UI 界面，用 tr_text 表示要翻译的文本，代码详见程序清单 8.1。

程序清单 8.1 locale.xml

```
<!-- awtk\demos\assets\raw\ui\locale.xml -->
<window anim_hint="htranslate"    x="0" y="0" w="320" h="480">
  <label name="ok" x="10" y="5" w="25%" h="30" tr_text="ok"/>
  <label name="cancel" x="right:10" y="5" w="25%" h="30" tr_text="cancel"/>

  <button name="chinese" x="10" y="m" w="25%" h="30" text="Chinese"/>
  <button name="english" x="center" y="m" w="25%" h="30" text="English"/>
  <button name="close" x="right:10" y="m" w="25%" h="30" text="Close"/>
</window>
```

如果不想在 Ui 文件使用 tr_text，还可以通过函数 widget_set_tr_text 设置要翻译的文本。

(2)编写中英文互译文件 strings.xml，其中<string name="ok">中的"ok"与上面的 tr_text=ok 对应，然后将文件放到 awtk\demos\assets\raw\strings 目录下，代码详见程序清单 8.2。

程序清单 8.2 strings.xml

```
<!-- awtk\demos\assets\raw\strings\strings.xml -->
<string name="ok">
<language name="en_US">OK</language>
<language name="zh_CN">确定</language>
</string>
```

```
<string name="cancel">
<language name="en_US">Cancel</language>
<language name="zh_CN">取消</language>
</string>

<string name="value is %d">
<language name="en_US">value is %d</language>
<language name="zh_CN">值为%d</language>
</string>

<string name="chinese">
<language name="en_US">Chinese</language>
<language name="zh_CN">中文</language>
</string>

<string name="english">
<language name="en_US">English</language>
<language name="zh_CN">英文</language>
</string>
```

(3)动态互译，在 C 代码中通过 locale_info_change 实现互译，代码详见程序清单 8.3。

程序清单 8.3 demo_ui_app.c

```
// awtk\demos\demo_ui_app.c
static ret_t on_change_locale(void* ctx, event_t* e) {
  char country[3];
  char language[3];
  const char* str = (const char*)ctx;

  tk_strncpy(language, str, 2);
  tk_strncpy(country, str + 3, 2);
  locale_info_change(locale_info(), language, country);
  return RET_OK;
}

static ret_t install_one(void* ctx, const void* iter) {
    ...
    } else if (tk_str_eq(name, "chinese")) {
      widget_on(widget, EVT_CLICK, on_change_locale, "zh_CN");
    } else if (tk_str_eq(name, "english")) {
      widget_on(widget, EVT_CLICK, on_change_locale, "en_US");
    }
  ...
}
```

更多用法请看：awtk\demos\demo_tr_app.c。

## 8.3 图片翻译

在一些应用程序中，有些文字是直接绘制在图片上的。所以在切换到不同的语言时，需要加载不同的图片。这时只要在图片名称中包含"$locale$"即可，加载时自动替换成当前的语言。如：有两张图片 language _en.png 和 language_zh.png 分别用在英文和中文环境下，那么可以将在样式文件 default.xml 这样写：

```
<button>
    ...
     <style name="language_btn" >
          <normal         icon="language_$locale$" />
          <pressed       icon="language_$locale$"/>
          <over           icon="language_$locale$" />
          <disable        />
     </style>
</button>
```

在英文环境下，加载图片时会按下列顺序查找（中文环境下也类似）：

- flag_en_US
- flag_en
- flag_

# 第9章 深入理解 AWTK

✍**本章导读**

在前面我们介绍了 AWTK 的一些开发基础知识，要想对 AWTK 有更加深入的理解，还需要了解一些关于 AWTK 框架的知识了。

## 9.1 简介

AWTK 框架有自己独特的地方，接下来让我们一起看看吧。

## 9.2 资源管理器

这里的资源管理器并非 Windows 下的文件浏览器，而是负责对各种资源，比如字体、主题、图片、界面数据、字符串和其它数据的进行集中管理的组件。引入资源管理器的目的有以下几个：

- 让上层不需要了解存储的方式。在没有文件系统时或者内存紧缺时，把资源转成常量数组直接编译到代码中。在有文件系统而且内存充足时，资源放在文件系统中。在有网络时，资源也可以存放在服务器上（暂未实现）。资源管理器为上层提供统一的接口，让上层而不用关心底层的存储方式
- 让上层不需要了解资源的具体格式。比如一个名为 earth 的图片，没有文件系统或内存紧缺，图片直接用位图数据格式存在 ROM 中；而有文件系统时，则用 PNG 格式存放在文件系统中。资源管理器让上层不需要关心图片的格式，访问时指定图片的名称即可（不用指定扩展名）
- 让上层不需要了解屏幕的密度。不同的屏幕密度下需要加载不同的图片，比如 MacPro 的 Retina 屏就需要用双倍解析度的图片，否则就出现界面模糊。AWTK 以后会支持 PC 软件和手机软件的开发，所以资源管理器需要为此提供支持，让上层不需关心屏幕的密度
- 对资源进行内存缓存。不同类型的资源使用方式是不一样的，比如字体和主题加载之后会一直使用，UI 文件在生成界面之后就暂时不需要了，PNG 文件解码之后就只需要保留解码的位图数据即可。资源管理器配合图片管理器等其它组件实现资源的自动缓存

负责资源管理器和资源管理相关的组件详见图 9.1，网络加载暂未实现。

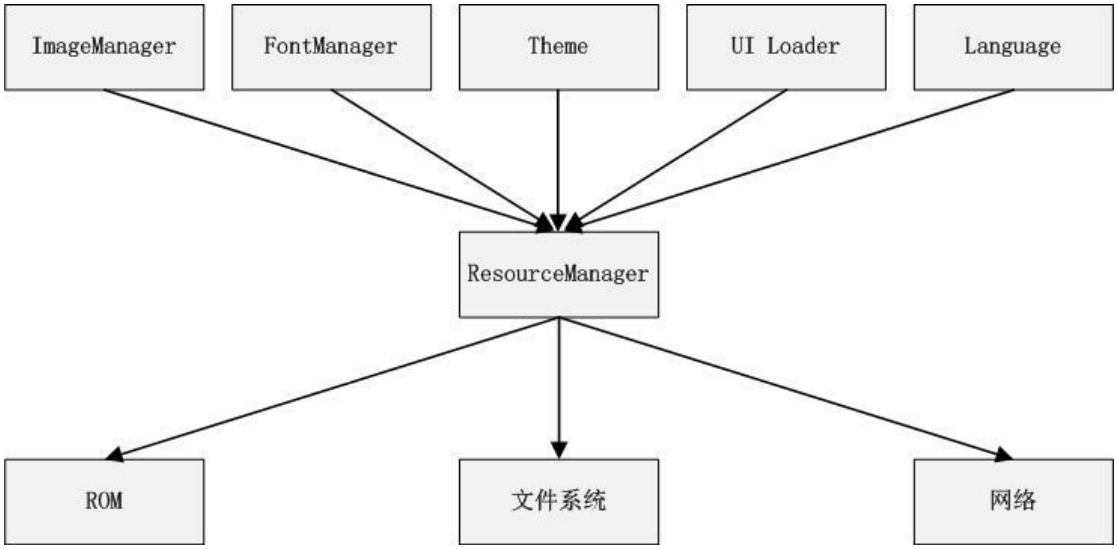



图 9.1 资源管理器

### 9.2.1 资源的生成

AWTK 中的资源需要进行格式转换才能使用：

- 在没有文件系统时或者内存紧缺时，需要把资源转成常量数组直接编译到代码中
- XML 格式的 UI 文件需要转换成二进制的格式
- XML 格式的主题文件需要转换成二进制的格式
- TTF 可以根据需要转换成位图字体
- PNG 可以根据需要转换成位图图片

### 9.2.2 相关工具

在 awtk\bin 目录下我们提供了多种工具用于生成资源，详见表 9.1。

表 9.1 资源生成工具

| 工具 | 说明 |
| --- | --- |
| fontgen | 位图字体生成工具 |
| imagegen | 位图图片生成工具 |
| resgen | 二进制文件生成资源常量数组 |
| themegen | XML 主题转换成二进制的主题 |
| xml_to_ui | XML 的界面描述格式转换二进制的界面描述格式 |
| update_res.py | 批量转换整个项目的资源 |

上面工具的使用方法请看：awtk\tools 目录对应文件夹下的 README.md。

### 9.2.3 初始化

将资源生成常量数组直接编译到代码中时，其初始化过程为：

1. 包含相应的数据文件，详见程序清单 9.1。

程序清单 9.1 包含数据文件

```
//awtk\demos\assets.c
```

```
...
#include "assets/inc/strings/zh_CN.data"
#include "assets/inc/strings/en_US.data"
#include "assets/inc/styles/slide_menu.data"
#include "assets/inc/styles/edit.data"
#include "assets/inc/styles/keyboard.data"
#include "assets/inc/styles/color.data"
#include "assets/inc/styles/tab_bottom.data"
...
```

2. 将资源增加到资源管理器中，详见程序清单 9.2。

程序清单 9.2 将资源增加到资源管理器

```
//awtk\demos\assets.c
...
assets_manager_add(rm, ui_kb_ascii);
assets_manager_add(rm, ui_vgcanvas);
assets_manager_add(rm, ui_rich_text1);
assets_manager_add(rm, ui_slide_menu);
assets_manager_add(rm, ui_radial_gradient);
assets_manager_add(rm, ui_color_picker_simple);
...
```

3. 将资源放在文件系统中时，一般不需要特殊处理，不过可以用 assets_manager_load 预加载资源，详见程序清单 9.3。

程序清单 9.3 资源在文件系统中

```
//awtk\demos\assets.c
...
#ifdef WITH_FS_RES
assets_manager_load(rm, ASSET_TYPE_STYLE, "default");
assets_manager_load(rm, ASSET_TYPE_FONT, "default");
...
```

### 9.2.4 使用方法

- 加载图片图片

使用 image_manager_load，指定图片的名称即可，如：

```
bitmap_t img;
image_manager_load(image_manager(), "earth", &img);
```

- 使用 UI 数据

使用 window_open，指定资源的名称即可。如：

```
widget_t* win = window_open(name);
```

- 使用字体

一般在主题文件中指定字体即可。

- 使用主题

一般在界面描述文件中指定 style 即可。

## 9.3 LCD 旋转

有时开发板上接的 LCD 方向和我们需要的不同，比如 LCD 缺省是横屏显示的，但我们需要竖屏的效果。如果无法通过修改硬件来实现旋转，就只能用软件来实现了。AWKT 目前对双帧缓冲的情况有完善的支持，对基于寄存器的 LCD 需要在驱动中进行配置。

### 9.3.1 使用方法

1. 首先在初始化时，用 tk_init 指定 LCD 的大小（这里 LCD 的大小是实际大小，不是旋转之后的大小），详见程序清单 9.4。

程序清单 9.4 初始化

```
//awtk\demos\demo_main.c
int main(void) {
  ...
  tk_init(320, 480, APP_SIMULATOR, NULL, res_root);
  ...
}
```

2. 然后用 tk_set_lcd_orientation 来设置 LCD 的旋转角度，详见程序清单 9.5。

程序清单 9.5 设置 LCD 旋转角度

```
//awtk\demos\demo_main.c
int main(void) {
  ...
//#define WITH_LCD_PORTRAIT 1
#if defined(USE_GUI_MAIN) && defined(WITH_LCD_PORTRAIT)
  if (lcd_w > lcd_h) {
    tk_set_lcd_orientation(LCD_ORIENTATION_90);
  }
#endif /*WITH_LCD_PORTRAIT*/

#ifdef WITH_LCD_LANDSCAPE
  if (lcd_w < lcd_h) {
    tk_set_lcd_orientation(LCD_ORIENTATION_90);
  }
#endif /*WITH_LCD_PORTRAIT*/
  ...
}
```

### 9.3.2 LCD 实现方式

AWTK 目前支持 3 种 LCD 实现方式，不同的实现方式对旋转的支持有所不同，下面我们一一介绍。

#### 1. 基于寄存器的 LCD

基于寄存器的 LCD（lcd_reg），调用 tk_set_lcd_orientation 只是做了两件事：

- 对触摸事件的坐标进行转换
- 设置窗口管理器的大小为旋转之后的大小

一般的 LCD 器件可以在驱动中设置像素的扫描顺序，在初始化时设置一下即可（驱动我不太熟悉，没有测试过），所以 AWTK 并没有对像素进行相应旋转。

2. 基于 FrameBuffer 的 LCD

基于 FrameBuffer 的 LCD（lcd_mem），AWTK 对旋转 90 度做了支持。调用 tk_set_lcd_orientation 后 AWTK 会做以下几件事：

- 对触摸事件的坐标进行转换
- 设置窗口管理器的大小为旋转之后的大小
- offline fb 为旋转之后的大小，online fb 为原始大小。在 lcd flush 时需要进行旋转（在没有硬件旋转减速的情况下，会增加一点性能开销）。
- 在进行平移窗口动画时。AWTK 将图片直接贴到 online fb 上，处于性能的考虑，在截图时进行旋转，在贴图时直接拷贝。所以在 lcd_mem_draw_image 和 lcd_mem_take_snapshot 两个函数中做了处理

3. 基于 VGCanvas 的 LCD

基于 VGCanvas 的 LCD（lcd_vgcanvas）主要用于 PC 和手机应用程序，目前还没有需求，暂未支持。

## 9.4 API 注释格式

看了 AWTK 的源码你会发现里面的注释是很规范的，那是因为这些 API 注释，还充当以下重要的作用：

- 提取 JSON 格式 IDL（awtk\tools\idl_gen\idl.json），用于生成各种语言的绑定代码
- 用于设计器（designer）获取各个控件的元信息

AWTK 采用了类似于 jsduck 的 API 注释格式，但是 jsduck 并不支持 C 语言的数据类型，所以没有办法完全兼容 jsduck 的格式。下面我们一起看看吧。

### 9.4.1 类的注释

示例：

```
/**
* @class progress_bar_t
* @parent widget_t
* @annotation ["scriptable"]
*  进度条控件。
*/
```

里面说明了类的名称、基类的名称和该类型是否可以脚本化。对于类，annotation 的取值有：

- scriptable 该类可以被脚本化
- fake 该类是 fake 的，并不真实存在

### 9.4.2 类成员变量注释

示例：

```
/**
```

```
* @property {uint8_t} value
* @annotation ["set_prop","get_prop","readable","persitent","design","scriptable"]
*  进度条的值[0-100]。
*/
```

里面说明了成员变量的类型、名称和是否只读等信息。对于 annotation 的取值有：

- set_prop 是否可以通过 widget_set_prop 来设置该属性
- get_prop 是否可以通过 widget_get_prop 来获取该属性
- readable 该属性是否可以直接读取
- writable 该属性是否可以直接修改
- persitent 该属性是否需要持久化
- design 该属性可以在设计器中设置
- scriptable 该属性是否支持脚本化

### 9.4.3 函数的注释

示例：

```
/**
* @method progress_bar_create
* @annotation ["constructor", "scriptable"]
*  创建 progress_bar 对象
* @param {widget_t*} parent  父控件
* @param {xy_t} x x 坐标
* @param {xy_t} y y 坐标
* @param {wh_t} w  宽度
* @param {wh_t} h  高度
* *
@return {widget_t*}  对象。
*/
widget_t* progress_bar_create(widget_t* parent, xy_t x, xy_t y, wh_t w, wh_t h);
/**
* @method progress_bar_cast
*  转换为 progress_bar 对象(供脚本语言使用)。
* @annotation ["cast", "scriptable"]
* @param {widget_t*} widget progress_bar 对象。
* *
@return {widget_t*} progress_bar 对象。
*/
widget_t* progress_bar_cast(widget_t* widget);
/**
* @method progress_bar_set_value
*  设置进度条的进度。
* @annotation ["scriptable"]
* @param {widget_t*} widget  控件对象。
* @param {uint8_t} value  进度
```

```
* *
@return {ret_t} 返回 RET_OK 表示成功，否则表示失败。
*/
ret_t progress_bar_set_value(widget_t* widget, uint8_t value);
```

里面说明了函数的名称、参数和返回值。对于 annotation 的取值有：

- global 是否是全局函数。除了指明为全局函数，函数是当前类的成员函数
- cast 类型转换函数
- constructor 构造函数
- deconstructor 析构函数
- scriptable 是否可以脚本化。对于特殊函数（通常有回调函数作为参数）不方便直接产生代码，可以指定为 scriptable:custom，使用定制的绑定代码

### 9.4.4 枚举的注释

示例：

```
/**
* @enum align_v_t
* @annotation ["scriptable"]
*  垂直对齐的常量定义。
*/
typedef enum _align_v_t {
/**
* @const ALIGN_V_NONE
*  无效对齐方式。
*/
ALIGN_V_NONE= 0,
/**
* @const ALIGN_V_MIDDLE
*  居中对齐。
*/
ALIGN_V_MIDDLE,
/**
* @const ALIGN_V_TOP
*  顶部对齐。
*/
ALIGN_V_TOP,
/**
* @const ALIGN_V_BOTTOM
*  底部对齐。
*/
ALIGN_V_BOTTOM
}align_v_t;
```

里面定义了枚举的名称和各个枚举值。对于枚举，annotation 的取值有：

- scriptable 该类可以被脚本化

# 第10章 经典案例

✍**本章导读**

使用 AWTK 可以高效开发出漂亮的 GUI 应用。

## 10.1 简介

本章以实际的应用场景为案例，来阐述使用 AWTK 做项目的开发过程及其代码实现过程，帮助开发者积累开发应用的实战经验。我们将用三个经典的案例：

- 仪表监控系统
- 洁净新风系统
- 炫酷图表

## 10.2 仪表监控系统

### 10.2.1 功能详解

仪表监控系统实现的功能点主要有下面几点：

- 动态显示增压器转速
- 动态显示燃油进机压力
- 动态显示进气温度
- 动态显示起动空气压力
- 动态显示进气压力
- 动态显示海水进机压力
- 动态显示仪表盘数值

### 10.2.2 应用实现

本案例使用目录结构、编译和运行步骤与第二章介绍的 hello world 应用类似，这里就不重复介绍了。本案例运行的效果详见图 10.1。


图 10.1 仪表监控系统

本案例使用到的代码均可以在 Meter-Demo 目录找到。

### 1. 显示背景图片

这里调用 label_create 函数创建 label 控件，然后通过调用 widget_use_style 函数设置主界面的背景，代码详见程序清单 10.1。

程序清单 10.1 显示背景图片

```
//Meter-Demo\src\window_main.c
/**
 * 创建背景 label
 */
static ret_t create_bg_labels(widget_t* win){
  value_t w_value;
  value_t h_value;
  widget_get_prop(win, "w", &w_value);
  widget_get_prop(win, "h", &h_value);
  int w = value_int32(&w_value);
  int h = value_int32(&h_value);
  widget_t* bg_label = label_create(win, 0, 0, w, h);
  if(bg_label){
      widget_use_style(bg_label, "bg");
      if(w == 480){
        widget_use_style(bg_label, "bg_480_272");
        is_big_lcd = FALSE;
      }
  }
```

```
  return RET_OK;
}
```

### 2. 创建仪表动画指针

创建仪表动画指针，主要分下面几个步骤：

(1)调用 view_create 创建一个容器，然后在该容器上通过 widget_on 注册 EVT_PAINT 事件的处理函数 on_paint_needle，该函数完成绘制仪表圆点和指针。

(2)调用 widget_create_animator()创建 rotation 旋转动画，完成指针不停的旋转。

代码详见程序清单 10.2。

程序清单 10.2 创建仪表动画指针

```
//Meter-Demo\src\window_main.c
/**
 * 重绘
 */
static ret_t on_paint_needle(void* ctx, event_t* e) {
      ...
      vgcanvas_set_stroke_color(vg, color_init(0xff, 0, 0, 0xff));
      vgcanvas_set_line_width(vg, 3);
      vgcanvas_begin_path(vg);
      vgcanvas_move_to(vg, x1, y1);
      vgcanvas_line_to(vg, x2, y2);
      vgcanvas_stroke(vg);
      vgcanvas_restore(vg);
   }

   return RET_OK;
}

/**
 * 初始化图片和动画
 */
static void init_image(widget_t* win, int x, int y, int w, int h, char* img_name, float from, float to, int duration, int
radius) {
   ...
   widget_on(img, EVT_PAINT, on_paint_needle, NULL);
#endif

   char rotation_str[100];
   memset(rotation_str, 0, sizeof(rotation_str));
   tk_snprintf(rotation_str, sizeof(rotation_str), "rotation(from=%f ,to=%f, yoyo_times=0, duration=%d,
easing=linear)", from, to, duration);
   widget_create_animator(img, rotation_str);
}
```

```
/**
 * 创建图片,rotation 中的 from 和 to 使用的是弧度：1 弧度=180/3.14=57.324 度
 */
static ret_t create_images(widget_t* win){
  char small_image[] = "pointer_small";
  char big_image[] = "pointer_big";

  if(is_big_lcd){
    /* 这里的宽和高是指仪表盘宽度和高度，x, y 指是各个仪表盘在背景图片上的左上标位置 */
    int width = 171;
    int height = 171;

    /* 左边：从上到下 */
    init_image(win, 114, 2, width, height, small_image, -2.2, 2.2, 2000, SMALL_RADIUS);
    init_image(win, 9, 140, width, height, small_image, -1.7, 1.7, 4000, SMALL_RADIUS);
    init_image(win, 44, 307, width, height, small_image, -1.5, 1.5, 3000, SMALL_RADIUS);
  ...
  return RET_OK;
}
```

### 3. 动态显示运行参数

要完成动态显示增压器转速、燃油进机压力等参数，主要分下面几个步骤：

(1)调用 label_create 函数创建 label 控件，用于显示参数。

(2)调用 timer_add 函数添加一个定时器回调函数 on_update_label 用于动态显示运行参数。

代码详见程序清单 10.3。

程序清单 10.3 动态显示运行参数

```
//Meter-Demo\src\window_main.c
/**
 * 初始化 label
 */
static void init_label(widget_t* win, int x, int y, int w, int h, const wchar_t* text, const char* widget_name) {
  widget_t* label = label_create(win, x, y, w, h);
  widget_set_text(label, text);
  widget_set_name(label, widget_name);
  if(is_big_lcd == FALSE){
    widget_use_style(label, "label_480_272");
  }
}

/**
 * 创建 label
 */
```

```
static ret_t create_labels(widget_t* win){
  int width = 30;
  int height = 30;

  if(is_big_lcd){
    /* 左边：从上到下 */
    int x_left = 330;
    init_label(win, x_left, 315, width, height, L"3005", "left_top_label");
    init_label(win, x_left, 365, width, height, L"12", "left_center_label");
    init_label(win, x_left, 415, width, height, L"13", "left_bottom_label");
  ...
  return RET_OK;
}

/**
 * 更新 Label 上的显示数值
 */
static ret_t on_update_label(const timer_info_t* timer) {
  widget_t* win = WIDGET(timer->ctx);

  /* 左边 */
  widget_t* left_top_label = widget_lookup(win, "left_top_label", TRUE);
  if (left_top_label) {
    label_set_random_int(left_top_label, 3000, 5000);
  }

  widget_t* left_center_label = widget_lookup(win, "left_center_label", TRUE);
  if (left_center_label) {
    label_set_random_int(left_center_label, 10, 30);
  }

  widget_t* left_bottom_label = widget_lookup(win, "left_bottom_label", TRUE);
  if (left_bottom_label) {
    label_set_random_int(left_bottom_label, 1, 15);
  }
  ...
  return RET_REPEAT;
}
```

## 10.3 洁净新风系统

### 10.3.1 功能详解

洁净新风系统实现的功能点主要有下面几点：

- 点击“开关”按钮，启动/停止 PM2.5、二氧化碳浓度、送风室内外温度、排风室内外温度等模拟读数，以及模拟报警

- 点击“自动”按钮，启动/停止背景（淡入淡出）切换
- 点击“定时”按钮，弹出定时设置对话框，模拟开/关定时功能（即时钟图标的显示/隐藏）
- 点击“记录”按钮，弹出记录列表（模拟数据）页面
- 点击“设置”按钮，弹出设置页面，设置控制、报警参数
- 点击三角形按钮，可以加/减频率、温度、湿度
- 实时更新系统时间

### 10.3.2 应用实现

本案例使用目录结构、编译和运行步骤与第二章介绍的 hello world 应用类似，这里就不重复介绍了。本案例运行的效果详见图 10.2。

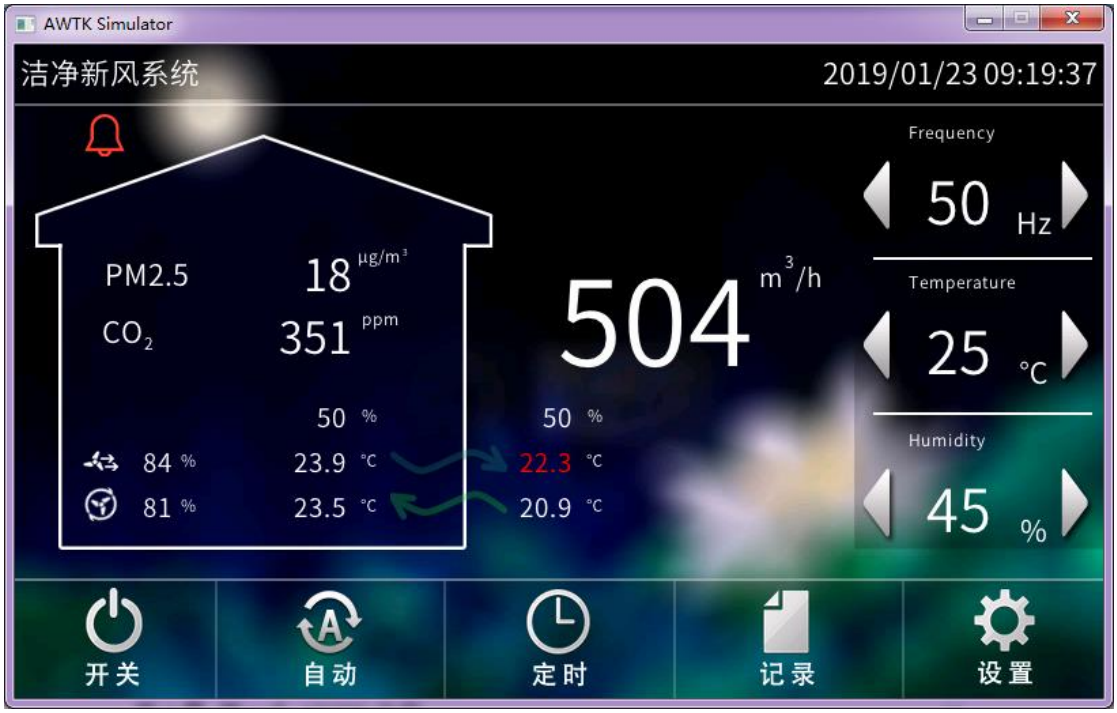



图 10.2 洁净新风系统

本案例使用到的代码均可以在 CleanAir-Demo 目录找到。

1. 如何打开窗口

(1)在 AWTK 中，可以使用一个 xml 文件来描述一个窗口的界面结构。比如描述记录列表窗口的 record.xml 代码详见程序清单 10.4。

程序清单 10.4 record.xml

```
<!-- src\assets\raw\ui\record.xml -->
<window anim_hint="htranslate">
  <button name="close" x="4" y="4" w="60" h="30" text="返回"/>
  <view x="4" y="36" w="100%" h="40" layout="r:1 c:5">
      <label text="状态" style="header"/>
      <label text="室内温度" style="header"/>
      <label text="室外温度" style="header"/>
```

```
        <label text="室内湿度" style="header"/>
        <label text="室外湿度" style="header"/>
    </view>
    <list_view x="4"    y="76" w="-8" h="400" item_height="40">
        <scroll_view name="view" x="0"    y="0" w="100%" h="100%">
            <list_item style="odd_clickable" layout="r1 c0">
                <image draw_type="icon" w="20%" image="bell"/>
                <label w="20%" text="24.4℃"/>
                <label w="20%" text="26.4℃"/>
                <label w="20%" text="20%"/>
                <label w="20%" text="30%"/>
            </list_item>
            ......
        </scroll_view>
        <scroll_bar_m name="bar" x="right" y="0" w="6" h="100%" value="0"/>
    </list_view>
</window>
```

(2)然后在代码中使用 window_open()即可打开该窗口，代码详见程序清单 10.5。

程序清单 10.5 打开 record.xml

```
//src\window_record.c
widget_t* win = window_open("record");
```

其中，“record”为 xml 的文件名，运行效果详见图 10.3。


图 10.3 记录界面

(3)如果需要手动关闭窗口使用 window_close()即可，代码详见程序清单 10.6。

程序清单 10.6 关闭 record.xml

```
//src\window_record.c
window_close(win);
```

2. 如何打开对话框

(1)比如创建一个定时按钮窗口，代码详见程序清单 10.7。

程序清单 10.7 timing.xml

```
<!-- src\assets\raw\ui\timing.xml -->
<dialog anim_hint="center_scale" h="200" w="580" x="c" y="m">
    <dialog_title h="35" text="定时设置" w="100%" x="0" y="0"/>
    <dialog_client h="-35" w="100%" x="0" y="bottom">
        <row h="30" layout="row:1 col:3" w="200" x="30" y="10">
            <label style="timing" text="时"/>
            <label style="timing" text="分"/>
            <label style="timing" text="秒"/>
        </row>
        <row h="120" layout="row:1 col:3" w="200" x="30" y="40">
            <text_selector options="1-24" text="15" visible_nr="3"/>
            <text_selector options="1-60" text="10" visible_nr="3"/>
            <text_selector options="1-60" text="16" visible_nr="3"/>
        </row>
        <button h="30" name="ok" style="timing_switch_btn" text="开" w="150" x="right:30" y="33"/>
        <button h="30" name="cancle" style="timing_switch_btn" text="关" w="150" x="right:30" y="96"/>
    </dialog_client>
</dialog>
```

(2)然后在代码中使用 window_open()即可打开该窗口，代码详见程序清单 10.8。

程序清单 10.8 打开 timing.xml

```
//src\window_timing.c
widget_t* win = window_open("timing");
```

其中，“timing”为 xml 的文件名，运行效果详见图 10.4。


图 10.4 定时界面

(3)如果需要手动关闭窗口使用 dialog_quit()即可，代码详见程序清单 10.9。

程序清单 10.9 关闭 timing.xml

```
//src\window_timing.c
dialog_quit(dialog, RET_OK);
```

### 3. 如何查找控件

在 main.xml 文件中设置控件的名字为" timing_status "，然后 C 代码中调用 widget_lookup()查找控件，代码详见程序清单 10.10。

程序清单 10.10 查找控件

```
<!-- src/assets/raw/ui/main.xml -->
<image name="timing_status" image="clock" draw_type="icon"/>
```

```
// src/window_main.c
widget_t* timing = widget_lookup(win, "timing_status", TRUE);
```

### 4. 如何响应界面事件

比如在主界面上设置开关按钮的点击事件，代码详见程序清单 10.11。

程序清单 10.11 相应界面事件

```
<!-- src\assets\raw\ui\main.xml -->
<button name="switch" x="0" w="147" h="100%" style="switch_btn"/>
```

```
//src\window_main.c
...
```

```
widget_t* win = widget_get_window(widget);
widget_on(widget, EVT_CLICK, on_switch, win);
...
```

### 5. 如何设置控件是否可见

AWTK 中可通过控件的 visible 属性控制该控件是否可见。比如控制定时图标是否可见，代码详见程序清单 10.12。

程序清单 10.12 设置控件是否可见

```
// src/window_main.c
widget_t* timing = widget_lookup(win, "timing_status", TRUE);
if (open_timing_window(&t) == RET_OK) {
  widget_set_visible(timing, TRUE, FALSE);
} else {
  widget_set_visible(timing, FALSE, FALSE);
}
```

### 6. 如何设置窗口顶部容器

比如在主界面上显示"洁净新风系统"，代码详见程序清单 10.13。

程序清单 10.13 设置窗口顶部容器

```
<!-- src\assets\raw\ui\main.xml -->
<app_bar x="0" y="0" w="100%" h="40">
        <label name="dev_name" x="0" y="0" w="200" h="100%" text="洁净新风系统" style="title_left"/>
        <label name="sys_time" x="200" y="0" w="-200" h="100%" style="title_right"/>
</app_bar>
```

### 7. 如何显示文本

比如在主界面上显示"PM2.5"对应的值 18，代码详见程序清单 10.14。

程序清单 10.14 显示文本

```
<!-- src\assets\raw\ui\main.xml -->
<label name="PM2_5" x="80" w="108" h="100%" text="18" style="reading_pm_co"/>
```

```
//src\window_main.c
...
widget_t* pm2_5 = widget_lookup(win, "PM2_5", TRUE);
...
widget_set_text_utf8(label, tk_itoa(text, sizeof(text), val));
```

### 8. 如何显示图片

比如在主界面上显示排风图片（fan_2.png），代码详见程序清单 10.15。

程序清单 10.15 显示图片

```
<!-- src\assets\raw\ui\main.xml -->
<image name="fan_2" image="fan_2" draw_type="default" y="m"/>
```

### 9. 如何显示富文本

比如在主界面上显示μ g/m$^3$ 分两部分处理：显示μ g/m 和显示上标数字 3，代码详见程序清单 10.16。

程序清单 10.16 显示富文本

```
<!-- src\assets\raw\ui\main.xml -->
<rich_text x="188" w="-188" h="100%" text="<font color=&quota;white&quota; align_v=&quota;top&quota;
size=&quota;18&quota;>μg/m</font><font color=&quota;white&quota; align_v=&quota;top&quota;
size=&quota;10&quota;>3</font>" />
```

### 10. 如何使用布局

比如在主界面上以一行两列的方式显示时钟（clock.png）和报警图标（bell.png），代码详见程序清单 10.17。

程序清单 10.17 使用布局

```
<!-- src\assets\raw\ui\main.xml -->
<view x="0" y="44" w="88" h="44" layout="r:1 c:2">
        <image name="timing_status" image="clock" draw_type="icon"/>
        <image name="alarm_status"    image="bell"    draw_type="icon"/>
</view>
```

### 11. 如何实现动画

(1)控件动画 API

AWTK 中，可以通过该控件绑定一个动画对象来实现动画。比如实现左下角风扇图标（fan_2.png）旋转，代码详见程序清单 10.18。

程序清单 10.18 控件动画

```
//src\window_main.c
widget_animator_t* widget_animator_get(widget_t* widget, const char* name, uint32_t duration, bool_t opacity)
{
  widget_animator_t* animator = NULL;
  value_t val;
  value_set_pointer(&val, NULL);
  if (widget_get_prop(widget, name, &val) != RET_OK || value_pointer(&val) == NULL) {
    if (opacity) {
      animator = widget_animator_opacity_create(widget, duration, 0, EASING_SIN_OUT);
    } else {
      animator = widget_animator_rotation_create(widget, duration, 0, EASING_LINEAR);
    }
    value_set_pointer(&val, animator);
    widget_set_prop(widget, name, &val);
  } else {
    animator = (widget_animator_t*)value_pointer(&val);
  }
  return animator;
```

```
}
...
animator = widget_animator_get(fan_2, "animator", 4000, FALSE);
widget_animator_rotation_set_params(animator, 0, 2*3.14159265);
widget_animator_set_repeat(animator, 0);
widget_animator_start(animator);
```

又或者，实现风向箭头的淡入淡出动画 (wind_in.png)，代码详见程序清单 10.19。

程序清单 10.19 淡入淡出动画

```
//src\window_main.c
animator = widget_animator_get(wind_out, "animator", 1000, TRUE);
widget_animator_opacity_set_params(animator, 50, 255);
widget_animator_set_yoyo(animator, 0);
widget_animator_start(animator);
```

(2)image_animation 组件

在 AWTK 中，可以使用 image_animation 组件来实现一组图片顺序播放的动画。比如设置主界面上 fan1a.png 和 fan_1b.png 图片动画，代码详见程序清单 10.20。

程序清单 10.20 图片动画

```
<!-- src/assets/raw/ui/main.xml -->
<image_animation name="fan_1" image="fan_1" sequence="ab" auto_play="false" interval="500" delay="100"
y="m"/>
```

```
//src\window_main.c
image_animation_play(fan_1);
```

### 12. 如何定时刷新界面数据

比如实时更新主界面右上角的系统时间，代码详见程序清单 10.21。

程序清单 10.21 定时刷新界面数据

```
//src\window_main.c
timer_add(on_systime_update, win, 1000);
```

### 13. 如何使用时分秒控件

比如点击主界面中的"定时"按钮弹出"定时设置"界面显示时分秒控件，代码详见程序清单 10.22。

程序清单 10.22 使用时分秒控件

```
<!-- src\assets\raw\ui\timing.xml -->
<row h="120" layout="row:1 col:3" w="200" x="30" y="40">
    <text_selector options="1-24" text="15" visible_nr="3"/>
    <text_selector options="1-60" text="10" visible_nr="3"/>
    <text_selector options="1-60" text="16" visible_nr="3"/>
</row>
```

### 14. 如何显示列表

比如点击主界面中的"记录"按钮弹出的列表视图，代码详见程序清单 10.23。

程序清单 10.23 显示列表

```
<!-- src\assets\raw\ui\record.xml -->
<list_view x="4"    y="76" w="-8" h="400" item_height="40">
      <scroll_view name="view" x="0"    y="0" w="100%" h="100%">
        <list_item style="odd_clickable" layout="r1 c0">
          <image draw_type="icon" w="20%" image="bell"/>
          <label w="20%" text="24.4℃"/>
          <label w="20%" text="26.4℃"/>
          <label w="20%" text="20%"/>
          <label w="20%" text="30%"/>
        </list_item>
       ...
      </scroll_view>
      <scroll_bar_m name="bar" x="right" y="0" w="6" h="100%" value="0"/>
  </list_view>
```

### 15. 如何实现分页

(1)比如点击主界面中的"设置"按钮弹出分页视图，代码详见程序清单 10.24。

程序清单 10.24 实现分页

```
<!-- src\assets\raw\ui\setting.xml -->
<window anim_hint="htranslate" name="setting_page">
  <button h="30" name="close" text="返回" w="60" x="4" y="4"/>
  <pages h="440" w="80%" x="right" y="40">
    <view h="60%" layout="r:4 c:3" w="100%">
      <label style="setting_page" text="室内温度"/>
      <edit auto_fix="true" input_type="ufloat" max="150" min="22.0" right_margin="16" step="0.1"
text="22.0"/>
       ...
  </pages>
  <list_view auto_hide_scroll_bar="true" h="440" item_height="40" w="20%" x="left" y="40">
    <scroll_view h="100%" name="view" w="-12" x="0" y="0">
      <tab_button text="控制参数设定" value="true"/>
      <tab_button text="报警设置"/>
    </scroll_view>
    <scroll_bar_d h="100%" name="bar" value="0" w="12" x="right" y="0"/>
  </list_view>
</window>
```

(2)运行效果详见图 10.5。


图 10.5 设置页面

## 16. 如何设置控件样式

比如设置主界面中的"开关"按钮的样式，代码详见程序清单 10.25。

程序清单 10.25 设置控件样式

```
<!-- src\assets\raw\ui\main.xml -->
<button name="switch" x="0" w="147" h="100%" style="switch_btn"/>
<!-- src\assets\raw\styles\default.xml -->
<button>
    ...
    <style name="switch_btn">
        <normal icon="btn_1"/>
        <pressed icon="btn_1_push"/>
        <over icon="btn_1_hover"/>
      </style>
    ...
</button>
```

```
//src\window_main.c
widget_t* btn = widget_lookup(win, "auto", TRUE);
widget_use_style(btn, "auto_btn_2a");
```

## 17. 如何实现中英文互译

比如点击主界面右上角的"中"按钮实现中英文环境下文本和图片的切换。接下来让我们一起看看如何实现的。

(1)中英环境下文本互译

比如要实现“洁净新风系统”和“ CleanAir Demo”文本之间的互译，步骤如下：

首先，在 strings.xml 文件中配置要互译的文本，代码详见程序清单 10.26。

程序清单 10.26 strings.xml

```
<!-- CleanAir-Demo\res_800_480\assets\raw\strings\strings.xml -->
...
<string name="CleanAir Demo">
    <language name="en_US">CleanAir Demo</language>
    <language name="zh_CN">洁净新风系统</language>
</string>
...
```

然后，在 main.xml 中使用 tr_text 设置要翻译的文本，代码详见程序清单 10.27。

程序清单 10.27 main.xml

```
<!-- CleanAir-Demo\res_800_480\assets\raw\ui\main.xml -->
<window anim_hint="htranslate" style="black">
    <label name="bkgnd" x="0" y="0" w="100%" h="100%" style="bg0"/>

    <app_bar x="0" y="0" w="100%" h="40">
        <label name="dev_name" x="0" y="0" w="200" h="100%" tr_text="CleanAir Demo"
style="title_left"/>
        <label name="sys_time" x="right:35" y="0" w="-200" h="100%" style="title_right"/>
          <button name="language_btn" x="right:5" y="0" w="25" h="100%" style="language_btn"/>
    </app_bar>

   ...
</window>
```

最后，在 window_main.c 调用 locale_info_change 函数实现互译，代码详见程序清单 10.28。

程序清单 10.28 window_main.c

```
//CleanAir-Demo\src\window_main.c

/**
 * 中英文互译
 */
static ret_t change_locale(const char *str) {
  char country[3];
  char language[3];

  strncpy(language, str, 2);
  strncpy(country, str + 3, 2);
  locale_info_change(locale_info(), language, country);

  return RET_OK;
```

```
}

/**
 * 点击中英文互译按钮
 */
static ret_t on_language(void *ctx, event_t *e) {
  (void)ctx;
  (void)e;

  const char *language = locale_info()->language;
  if (tk_str_eq(language, "en")) {
    change_locale("zh_CN");
  } else {
    change_locale("en_US");
  }

  return RET_OK;
}
```

(2) 中英环境下图片切换

比如要实现点击主界面右上角的“中”和“EN”按钮之间的切换，步骤如下：

首先，将两张图片命名为“language_en.png”和“language_zh.png”分别代表中文和英文环境下的图片，“language_ en.png”中“en”代表的是语言环境，不可以随意命名。

然后，在样式文件 default.xml 中通过 language_$locale$字符串表示要切换的图片。在中文环境下使用 language_zh.png，在英文环境下使用 language_en.png，代码详见程序清单 10.29。

程序清单 10.29 default.xml

```
<!-- CleanAir-Demo\res_800_480\assets\raw\styles\default.xml
...
<button>
<style name="language_btn" >
        <normal      icon="language_$locale$" />
        <pressed     icon="language_$locale$"/>
        <over        icon="language_$locale$" />
        <disable     />
    </style>
</button>
```

就这样就完成了中英文环境下图片的切换。

## 10.4 炫酷图表

### 10.4.1 功能详解

炫酷图表实现的功能点主要有下面几点：

- 仪表盘
- 饼图

- 曲线图
- 柱状图

### 10.4.2 应用实现

本案例使用目录结构、编译和运行步骤与第二章介绍的 hello world 应用类似，这里就不重复介绍了。本案例运行的效果详见图 10.6。

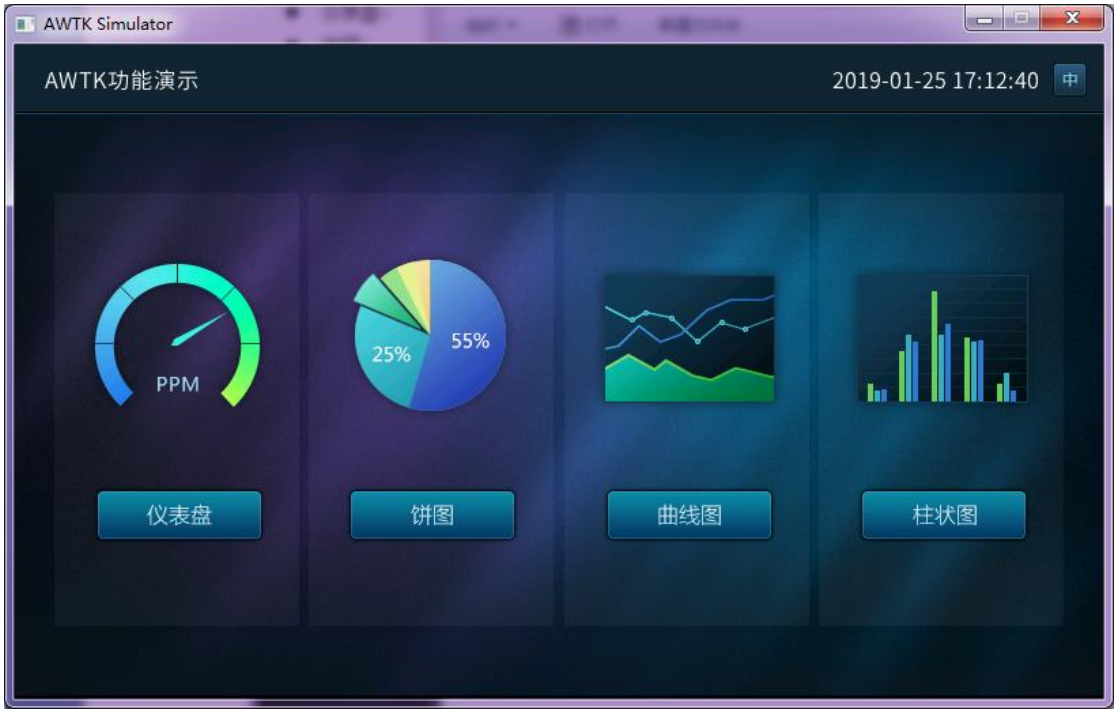



图 10.6 炫酷图表

本案例使用到的代码均可以在 Chart-Demo 目录找到。

#### 1. 实现主界面逻辑

主界面实现的功能点如下：

- 中英文环境切换
- 界面切换，通过注册点击事件，然后进入相应的界面

下面，我们一起看看这些功能点是怎么实现的。

(1) 中英文环境切换

本案例实现中英文环境切换与洁净新风系统的类似，这里是通过改变样式的方式实现图片切换（比较复杂，代码详见程序清单 10.30），而洁净新风系统是通过在样式文件 default.xml 中给图片加个“$locale$”后缀完成（比较简单，推荐使用这种方式）。由于在上面章节已经介绍过了，这里不重复了。

程序清单 10.30 中英文环境切换

```
//Chart-Demo\src\window_main.c

/**
 * 改变样式
 */
```

```
static ret_t change_style(widget_t* win, const char* name, const char* style, bool_t flag) {
  widget_t* w = widget_lookup(window_manager(), name, TRUE);
  if (w) {
    if (flag) {
      char style_en[20];
      tk_snprintf(style_en, sizeof(style_en), "%s_en", style);
      widget_use_style(w, style_en);
    } else {
      widget_use_style(w, style);
    }
  }

  return RET_OK;
}
```

(2)界面切换

通过函数 widget_on 给相应的控件注册“EVT_CLICK”点击事件，进入仪表盘、饼图、曲线图和柱状图界面，代码详见程序清单 10.31。

程序清单 10.31 界面切换

```
//Chart-Demo\src\window_main.c

/**
 * 子控件初始化(主要是设置 click 回调、初始显示信息)
 */
static ret_t init_widget(void* ctx, const void* iter) {
  widget_t* widget = WIDGET(iter);

  (void)ctx;

  if (widget->name != NULL) {
    const char* name = widget->name;
    if (tk_str_eq(name, "meter") || tk_str_eq(name, "meter_image")) {
      widget_t* win = widget_get_window(widget);
      widget_on(widget, EVT_CLICK, on_meter, win);
    } else if (tk_str_eq(name, "pie") || tk_str_eq(name, "pie_image")) {
      widget_t* win = widget_get_window(widget);
      widget_on(widget, EVT_CLICK, on_pie, win);
    } else if (tk_str_eq(name, "graph") || tk_str_eq(name, "graph_image")) {
      widget_t* win = widget_get_window(widget);
      widget_on(widget, EVT_CLICK, on_graph, win);
    } else if (tk_str_eq(name, "histogram") || tk_str_eq(name, "histogram_image")) {
      widget_t* win = widget_get_window(widget);
      widget_on(widget, EVT_CLICK, on_histogram, win);
    }
```

```
  }

  return RET_OK;
}
```

### 2. 实现仪表盘逻辑

仪表盘界面实现的功能点如下：

- 绘制仪表盘
- 启动仪表盘
- 停止仪表盘

下面，我们一起看看这些功能点是怎么实现的。

(1) 绘制仪表盘

通过 AWTK 自带的 guage 控件实现仪表盘的绘制，代码详见程序清单 10.32。

程序清单 10.32 绘制仪表盘

```
<!-- Chart-Demo\res_800_480\assets\raw\ui\window_meter.xml -->

<window anim_hint="htranslate" tr_text="meter_title">
    <view name="guage_function_view" x="0" y="0" w="87%" h="90%" layout="r:1 c:2 s:25" style="dark">
        <guage    image="guage_bg_1" >
            <guage_pointer name="left_pointer" x="c" y="m" w="12" h="160" value="-128"
image="guage_pointer_1"
            animation="value(from=-128, to=128, yoyo_times=0, forever=TRUE, duration=3000,
delay=1000)"/>
        </guage>
        <guage image="guage_bg_2" >
            <guage_pointer name="right_pointer" x="c" y="m" w="12" h="160" value="-128"
image="guage_pointer_2"
            animation="value(from=-128, to=128, yoyo_times=0, forever=TRUE, duration=3000,
delay=1000)"/>
        </guage>
    </view>
    ...
</window>
```

(2) 启动仪表盘（界面右边的第一个按钮）

完成调用 widget_start_animator 开始动画，实现启动仪表盘，代码详见程序清单 10.33。

程序清单 10.33 启动仪表盘

```
//Chart-Demo\src\window_meter.c

/**
 * 点击开始按钮
 */
```

```
static ret_t on_start(void* ctx, event_t* e) {
    widget_t* win = (widget_t*)ctx;
    set_btn_style(win, e);
    widget_start_animator(NULL, NULL);

    return RET_OK;
}
```

(3) 停止仪表盘（界面右边的第二个按钮）

调用 widget_stop_animator 停止动画，实现停止仪表盘，代码详见程序清单 10.34。

程序清单 10.34 启动仪表盘

```
//Chart-Demo\src\window_meter.c

/**
 * 点击停止按钮
 */
static ret_t on_stop(void* ctx, event_t* e) {
    widget_t* win = (widget_t*)ctx;
    set_btn_style(win, e);
    widget_stop_animator(NULL, NULL);

    return RET_OK;
}
```

运行效果详见图 10.7。


图 10.7 仪表盘

### 3. 实现饼图逻辑

饼图界面实现的功能点如下：

- 绘制饼图
- 生成新的饼图/拱形
- 饼图和拱形之间的切换

下面，我们一起看看这些功能点是怎么实现的。

(1)绘制饼图

饼图的绘制是通过调用 AWTK 提供的 vgcanvas 接口绘制的（代码详见程序清单 10.35），然后做成控件 pie_slice，最后在 Ui 文件 window_pie.xml 直接使用就可以了。

程序清单 10.35 绘制饼图

```
//Chart-Demo\src\custom_widgets\pie_slice\pie_slice.c

static ret_t pie_slice_on_paint_self(widget_t* widget, canvas_t* c) {
  ...
  if (vg != NULL && (has_image || color.rgba.a)) {
    xy_t cx = widget->w / 2;
    xy_t cy = widget->h / 2;
    float_t end_angle = 0;
    float_t r = 0;
    bool_t ccw = pie_slice->counter_clock_wise;
    float_t start_angle = TK_D2R(pie_slice->start_angle);
    float_t angle = (M_PI * 2 * pie_slice->value) / pie_slice->max;

    if (ccw) {
      end_angle = start_angle - angle + M_PI * 2;
    } else {
      end_angle = start_angle + angle;
    }

    vgcanvas_save(vg);
    vgcanvas_translate(vg, c->ox, c->oy);
    if (end_angle > start_angle) {
      vgcanvas_set_fill_color(vg, color);
      vgcanvas_begin_path(vg);
      r = tk_min(cx, cy);
      r -= pie_slice->explode_distancefactor;

      if (pie_slice->is_semicircle) {
        if (ccw) {
          start_angle = M_PI * 2 - (start_angle + angle) / 2;
          end_angle = start_angle + angle / 2;
          cy = cy + r * 0.5;
```

```
            r += pie_slice->explode_distancefactor * 1.5;
            vgcanvas_arc(vg, cx, cy, r, start_angle, end_angle, !ccw);
            r -= pie_slice->inner_radius;
            vgcanvas_arc(vg, cx, cy, r, end_angle, start_angle, ccw);
          }
        } else {
          vgcanvas_arc(vg, cx, cy, r, start_angle, end_angle, ccw);
          if (r - pie_slice->inner_radius <= 0) {
            vgcanvas_line_to(vg, cx, cy);
          } else {
            r -= pie_slice->inner_radius;
            vgcanvas_arc(vg, cx, cy, r, end_angle, start_angle, !ccw);
          }
        }

        vgcanvas_close_path(vg);
        if (has_image) {
          vgcanvas_paint(vg, FALSE, &img);
        } else {
          vgcanvas_fill(vg);
        }
      }
      vgcanvas_restore(vg);
    }

    ...
}
```

(2)生成新的饼图/拱形（界面右边的第一个按钮）

生成新的饼图/拱形，实现步骤如下：

首先，通过创建值动画将原来的饼图/拱形还原到原点，代码详见程序清单 10.36。

**程序清单 10.36 还原到原点**

```
//Chart-Demo\src\window_pie.c

/**
 * 创建扇形还原到原点动画
 */
static ret_t create_animator_to_zero(widget_t* win) {
  set_btn_enable(win, FALSE);

  widget_t* pie_view = widget_lookup(win, "pie_view", TRUE);
  if (pie_view) {
    WIDGET_FOR_EACH_CHILD_BEGIN_R(pie_view, iter, i)
    value_t v;
```

```
    widget_get_prop(iter, PIE_SLICE_PROP_IS_EXPLODED, &v);
    save_pie_exploded[i] = value_bool(&v);

    pie_slice_t* pie_slice = PIE_SLICE(iter);
    if (pie_slice->is_exploded) {
      pie_slice_set_exploded(iter);
    }
    int32_t delay = 80;
    delay = delay * (nr - 1 - i);
    char param[100];
    tk_snprintf(param, sizeof(param), "value(to=0, duration=50, delay=%d, easing=sin_out)", delay);
    widget_create_animator(iter, param);
    WIDGET_FOR_EACH_CHILD_END();
  }

  return RET_OK;
}
```

然后，通过定时器判断上面的动画是否已经全部还原到原点，如果是就创建新的饼图/拱形，代码详见程序清单 10.37。

程序清单 10.37 创建新的饼图/拱形

```
//Chart-Demo\src\window_pie.c

/**
 * 点击创建新饼图或者拱形定时器
 */
static ret_t on_new_pie_timer(const timer_info_t* timer) {
  pie_value_t* pie_data = NULL;
  widget_t* win = WIDGET(timer->ctx);

  uint32_t count = widget_animator_manager_count(widget_animator_manager());
  if (count == 0) {
    value_t v;
    value_t v1;
    ret_t result = widget_get_prop(win, "is_new", &v);
    bool_t flag = (result == RET_NOT_FOUND) ? FALSE : value_bool(&v);
    if (flag) {
      pie_data = old_pie_data;
      widget_set_prop(win, "is_new", value_set_bool(&v1, FALSE));
    } else {
      pie_data = new_pie_data;
      widget_set_prop(win, "is_new", value_set_bool(&v1, TRUE));
    }
    widget_t* pie_view = widget_lookup(win, "pie_view", TRUE);
```

```
    if (pie_view) {
      WIDGET_FOR_EACH_CHILD_BEGIN(pie_view, iter, i)
      pie_slice_t* pie_slice = PIE_SLICE(iter);
      int32_t delay = DELAY_TIME;
      int32_t duration = DURATION_TIME;
      delay = delay * i;
      const pie_value_t* new_pie = pie_data + nr - 1 - i;
      pie_slice_set_start_angle(iter, new_pie->start_angle);
      char param[100];
      tk_snprintf(param, sizeof(param),
                  "value(name=%s, to=%d, duration=%d, delay=%d, easing=sin_out)", SAVE_EXPLODED,
                  new_pie->value, duration, delay);
      widget_create_animator(iter, param);
      WIDGET_FOR_EACH_CHILD_END();
    }
    timer_add(on_save_exploded_timer, win, 1000 / 60);

    return RET_REMOVE;
  }

  return RET_REPEAT;
}
```

(3)饼图和拱形之间的切换（界面右边的第二个按钮）

饼图和拱形之间的切换，实现步骤如下：

首先，通过创建值动画将原来的饼图/拱形还原到原点，代码详见程序清单 10.36。

然后，通过定时器判断上面的动画是否已经全部还原到原点，如果是就进行饼图和拱形之间的切换，并调用 pie_slice_set_semicircle 设置是否是拱形，代码详见程序清单 10.38。

程序清单 10.38 饼图和拱形之间的切换

```
//Chart-Demo\src\window_pie.c

/**
 * 拱形定时器
 */
static ret_t on_arch_timer(const timer_info_t* timer) {
  widget_t* win = WIDGET(timer->ctx);

  uint32_t new_inner_radius = win->h / 5;
  uint32_t inner_radius = 0;

  widget_t* pie_view = widget_lookup(win, "pie_view", TRUE);
  uint32_t count = widget_animator_manager_count(widget_animator_manager());
  if (count == 0) {
    value_t v;
```

```
    value_t v1;
    ret_t result = widget_get_prop(win, "is_arch", &v);
    bool_t flag = (result == RET_NOT_FOUND) ? FALSE : value_bool(&v);
    if (flag) {
      widget_set_prop(win, "is_arch", value_set_bool(&v1, FALSE));
      inner_radius = 550;
    } else {
      widget_set_prop(win, "is_arch", value_set_bool(&v1, TRUE));
      inner_radius = new_inner_radius;
    }
    widget_t* pie_view = widget_lookup(win, "pie_view", TRUE);
    if (pie_view) {
      WIDGET_FOR_EACH_CHILD_BEGIN(pie_view, iter, i)
      pie_slice_t* pie_slice = PIE_SLICE(iter);
      pie_slice_set_inner_radius(iter, inner_radius);

      if (flag) {
        pie_slice_set_semicircle(iter, FALSE);
        pie_slice_set_counter_clock_wise(iter, FALSE);
      } else {
        pie_slice_set_semicircle(iter, TRUE);
        pie_slice_set_counter_clock_wise(iter, TRUE);
      }

      int32_t delay = DELAY_TIME;
      int32_t duration = DURATION_TIME;
      delay = delay * i;
      const pie_value_t* new_pie = old_pie_data + nr - 1 - i;
      pie_slice_set_start_angle(iter, new_pie->start_angle);
      char param[100];
      tk_snprintf(param, sizeof(param),
                  "value(name=%s, to=%d, duration=%d, delay=%d, easing=sin_out)", SAVE_EXPLODED,
                  new_pie->value, duration, delay);
      widget_create_animator(iter, param);
      WIDGET_FOR_EACH_CHILD_END();
    }
    timer_add(on_save_exploded_timer, win, 1000 / 60);

    return RET_REMOVE;
  }

  return RET_REPEAT;
}
```

运行效果详见图 10.8。

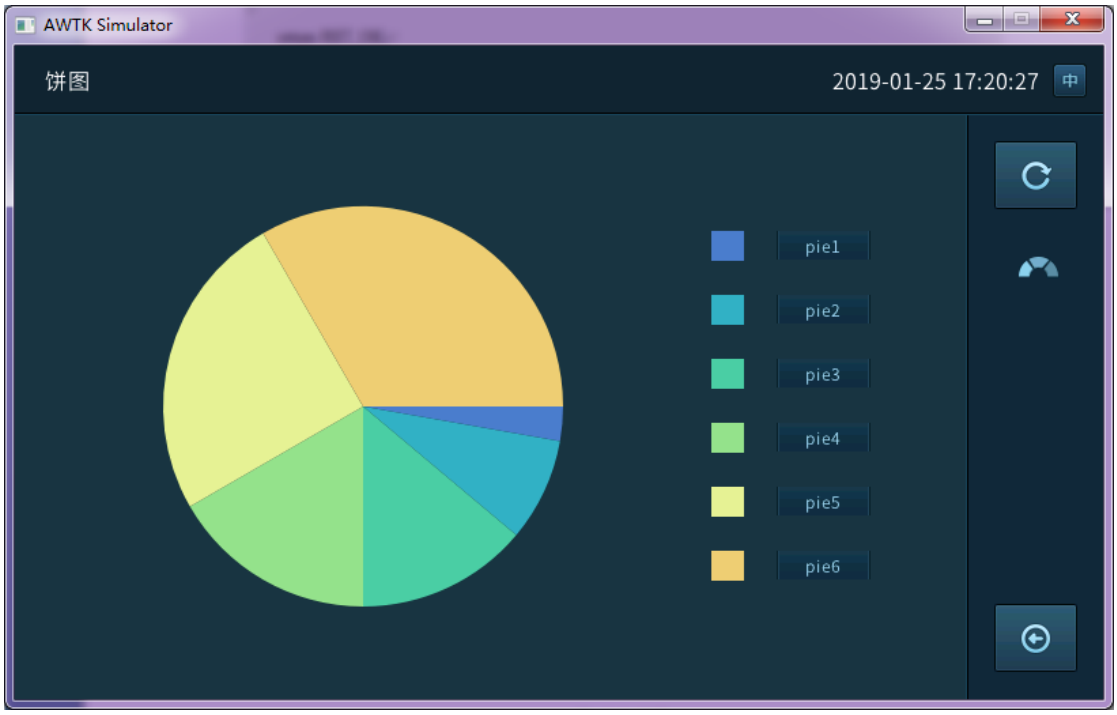



图 10.8 饼图

### 4. 实现曲线图逻辑

曲线图界面实现的功能点如下：

- 绘制曲线图
- 随机生成新的曲线图
- 设置是否显示曲线
- 设置是否显示闭合区域
- 设置是否显示圆点
- 设置是否显示平滑曲线

下面，我们一起看看这些功能点是怎么实现的。

(1)绘制曲线图

曲线图的绘制是通过调用 AWTK 提供的 vgcanvas 接口绘制的（代码详见程序清单 10.39），然后做成控件 line_series，最后在 Ui 文件 window_line_series.xml 直接使用就可以了。

程序清单 10.39 绘制曲线图

```
//Chart-Demo\src\custom_widgets\chart\line_series.c

static ret_t line_series_on_paint_self(widget_t* widget, canvas_t* c) {
  vgcanvas_t* vg;
  float_t x0, y0;
  float_t* x;
  float_t* y;
  uint32_t nr;
```

```
bitmap_t img;
bool_t has_image = FALSE;
bool_t vertical = TRUE;
rect_t r_save;
rect_t r = rect_init(c->ox, c->oy, widget->w, widget->h);
line_series_t* series = LINE_SERIES(line_series_cast(widget));

return_value_if_fail(series != NULL, RET_BAD_PARAMS);

if (line_series_generate_points(series, &x0, &y0, &x, &y, &nr, &vertical) != RET_OK) {
  return RET_OK;
}

if (series->symbol.image) {
  has_image = widget_load_image(widget, series->symbol.image, &img) == RET_OK;
}

canvas_get_clip_rect(c, &r_save);
r = r_save;
if (nr > 1) {
  if (vertical) {
    r.x = c->ox;
    r.w = x0 + x[nr - 1] - (x[1] - x[0] + 1) * series->clip_sample + series->symbol.size + 1;
  } else {
    r.y = c->oy;
    r.h = y0 + y[nr - 1] - (y[1] - y[0] + 1) * series->clip_sample + series->symbol.size + 1;
  }
}
canvas_set_clip_rect(c, &r);

if (nr == 1) {
  canvas_fill_rect(c, x0 + x[0], y0 + y[0], 1, 1);
}

vg = canvas_get_vgcanvas(c);
vgcanvas_save(vg);
vgcanvas_translate(vg, c->ox, c->oy);
vgcanvas_translate(vg, x0, y0);
if (nr > 1) {
  if (series->line.smooth) {
    series_draw_smooth_line(vg, x, y, nr, &(series->line));
    series_draw_smooth_line_area(vg, 0, 0, x, y, nr, &(series->area), vertical);
  } else {
    series_draw_line(vg, x, y, nr, &(series->line));
```

```
        series_draw_line_area(vg, 0, 0, x, y, nr, &(series->area), vertical);
      }
    }
    series_draw_symbol(vg, x, y, nr, &(series->symbol), has_image ? &img : NULL);
    vgcanvas_restore(vg);

    canvas_set_clip_rect(c, &r_save);

    line_series_release_points(x, y, vertical);

    return RET_OK;
}
```

(2) 随机生成新的曲线图（界面右边的第一个按钮）

通过调用 random 函数生成随机值，然后将生成的随机值通过 line_series_set_data 函数设置给控件，代码详见程序清单 10.40。

程序清单 10.40 随机生成新的曲线图

```
//Chart-Demo\src\window_line_series.c

static void on_set_series_data(widget_t* widget, uint32_t count) {
    float_t* new_value = TKMEM_ZALLOCN(float, count);
    widget_t* series;
    uint32_t i, j;
    uint32_t nr = chart_view_count_series(widget, WIDGET_TYPE_LINE_SERIES);
    for (i = 0; i < nr; i++) {
      for (j = 0; j < count; j++) {
        new_value[j] = random() % 120 + 10;
      }

      series = chart_view_get_series(widget, WIDGET_TYPE_LINE_SERIES, i);
      if (series) {
        line_series_set_data(series, new_value, count);
      }
    }
    TKMEM_FREE(new_value);
}

static ret_t on_new_random_data(void* ctx, event_t* e) {
    widget_t* win = WIDGET(ctx);
    widget_t* chart_view = widget_lookup(win, "chartview", TRUE);
    if (chart_view) {
      on_set_series_data(chart_view, 12);
    }
    return RET_OK;
```

```
}
```

(3)设置是否显示曲线（界面右边的第二个按钮）

通过设置控件的 SERIES_PROP_LINE_SHOW 属性是否显示曲线，代码详见程序清单 10.41。

程序清单 10.41 设置是否显示曲线

```
//Chart-Demo\src\window_line_series.c

static ret_t on_series_prop_changed(void* ctx, event_t* e, const char* prop, const char* btn_name,
                                    const char* style, const char* style_select) {
  widget_t* win = WIDGET(ctx);
  widget_t* chart_view = widget_lookup(win, "chartview", TRUE);
  if (chart_view) {
    widget_t* series;
    uint32_t i;
    uint32_t nr = chart_view_count_series(chart_view, NULL);
    for (i = 0; i < nr; i++) {
      series = chart_view_get_series(chart_view, NULL, i);
      if (series) {
        value_t v;
        if (widget_get_prop(series, prop, &v) == RET_OK) {
          value_set_bool(&v, !value_bool(&v));
          widget_set_prop(series, prop, &v);

          widget_t* btn = widget_lookup(win, btn_name, TRUE);
          if (btn) {
            widget_use_style(btn, value_bool(&v) ? style_select : style);
          }
        }
      }
    }
  }

  return RET_OK;
}

static ret_t on_line_changed(void* ctx, event_t* e) {
  return on_series_prop_changed(ctx, e, SERIES_PROP_LINE_SHOW, "line", "line", "line_select");
}
```

(4)设置是否显示闭合区域（界面右边的第三个按钮）

通过设置控件的 SERIES_PROP_AREA_SHOW 属性是否显示闭合区域，代码详见程序清单 10.42。

程序清单 10.42 设置是否显示闭合区域

```
//Chart-Demo\src\window_line_series.c

static ret_t on_area_changed(void* ctx, event_t* e) {
  return on_series_prop_changed(ctx, e, SERIES_PROP_AREA_SHOW, "area", "area", "area_select");
}
```

(5)设置是否显示圆点（界面右边的第四个按钮）

通过设置控件的 SERIES_PROP_SYMBOL_SHOW 属性是否显示圆点，代码详见程序清单 10.43。

程序清单 10.43 设置是否显示圆点

```
//Chart-Demo\src\window_line_series.c

static ret_t on_symbol_changed(void* ctx, event_t* e) {
  return on_series_prop_changed(ctx, e, SERIES_PROP_SYMBOL_SHOW, "symbol", "symbol",
                                "symbol_select");
}
```

(6)设置是否显示平滑曲线（界面右边的第五个按钮）

通过设置控件的 SERIES_PROP_LINE_SMOOTH 属性是否显示平滑曲线，代码详见程序清单 10.44。

程序清单 10.44 设置是否显示平滑曲线

```
//Chart-Demo\src\window_line_series.c

static ret_t on_smooth_changed(void* ctx, event_t* e) {
  return on_series_prop_changed(ctx, e, SERIES_PROP_LINE_SMOOTH, "smooth", "smooth",
                                "smooth_select");
}
```

运行效果详见图 10.9。

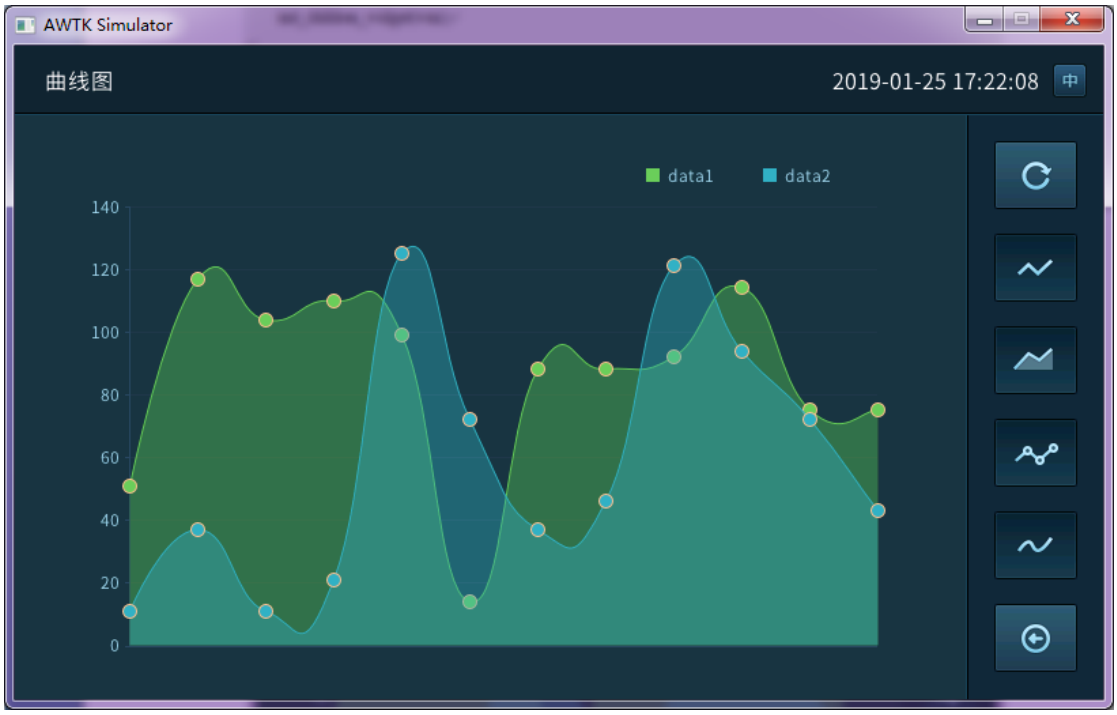



图 10.9 曲线图

### 5. 实现柱状图逻辑

柱状图界面实现的功能点如下：

- 绘制柱状图
- 随机生成新的柱状图

下面，我们一起看看这些功能点是怎么实现的。

(1)绘制柱状图

柱状图的绘制是通过调用 AWTK 提供的 vgcanvas 接口绘制的（代码详见程序清单 10.45），然后做成控件 bar_series，最后在 Ui 文件 window_ bar_series.xml 直接使用就可以了。

程序清单 10.45 绘制柱状图

```
//Chart-Demo\src\custom_widgets\chart\bar_series.c

static ret_t bar_series_on_paint_self(widget_t* widget, canvas_t* c) {
  float_t x0, y0, size;
  float_t* x;
  float_t* y;
  uint32_t nr;
  bitmap_t img;
  bool_t has_image;
  bool_t vertical = TRUE;
  bar_series_t* series = BAR_SERIES(bar_series_cast(widget));
  vgcanvas_t* vg = canvas_get_vgcanvas(c);
```

```
  return_value_if_fail(series != NULL, RET_BAD_PARAMS);

  if (bar_series_generate_points(series, &x0, &y0, &x, &y, &nr, &size, &vertical) != RET_OK) {
    return RET_OK;
  }

  has_image = series->bar.image && widget_load_image(widget, series->bar.image, &img) == RET_OK;

  if (has_image) {
    vgcanvas_save(vg);
    vgcanvas_translate(vg, c->ox, c->oy);
    vgcanvas_translate(vg, x0, y0);
    series_draw_bar(vg, 0, 0, x, y, nr, &(series->bar), size, vertical, has_image ? &img : NULL);
    vgcanvas_restore(vg);
  } else {
    series_draw_bar_c(c, x0, y0, x, y, nr, &(series->bar), size, vertical);
  }

  bar_series_release_points(x, y, vertical);

  return RET_OK;
}
```

(2) 随机生成新的柱状图（界面右边的第一个按钮）

通过调用 random 函数生成随机值，然后将生成的随机值通过 bar_series_set_data 函数设置给控件，代码详见程序清单 10.46。

程序清单 10.46 随机生成新的柱状图

```
//Chart-Demo\src\window_bar_series.c

static void random_series_data(widget_t* widget) {
  const uint32_t count = 7;
  float_t new_value[7];
  widget_t* series;
  uint32_t i, j;
  uint32_t nr = chart_view_count_series(widget, NULL);
  for (i = 0; i < nr; i++) {
    for (j = 0; j < count; j++) {
      new_value[j] = random() % 140;
    }
    series = chart_view_get_series(widget, NULL, i);
    if (series) {
      bar_series_set_data(series, new_value, count);
    }
  }
```

```
}

static ret_t on_new_random_data(void* ctx, event_t* e) {
  widget_t* win = WIDGET(ctx);
  widget_t* chart_view = widget_lookup(win, "chartview", TRUE);
  if (chart_view) {
    random_series_data(chart_view);
  }
  return RET_OK;
}
```

运行效果详见图 10.10。

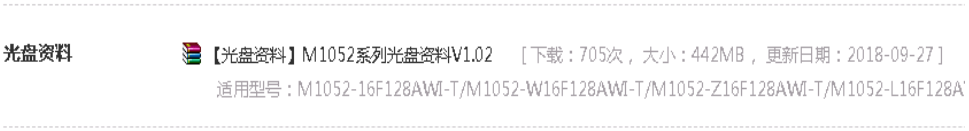

图 10.10 柱状图

# 附录A 把应用移植到 AWorks

AWTK 移植到 AWorks 平台非常方便，只需要简单的几个步骤就可以了，下面我们将介绍如何将我们的 AWTK 应用一直到 AWorks 平台上。

## A.1 复制工程

1. 到 zlg 官网( http://www.zlg.cn/ipc/down/down/id/217.html)下载 AWorks AWTK 模板工程，选择"M1052 系列光盘资料 V1.02"，详见图附录 A.1。

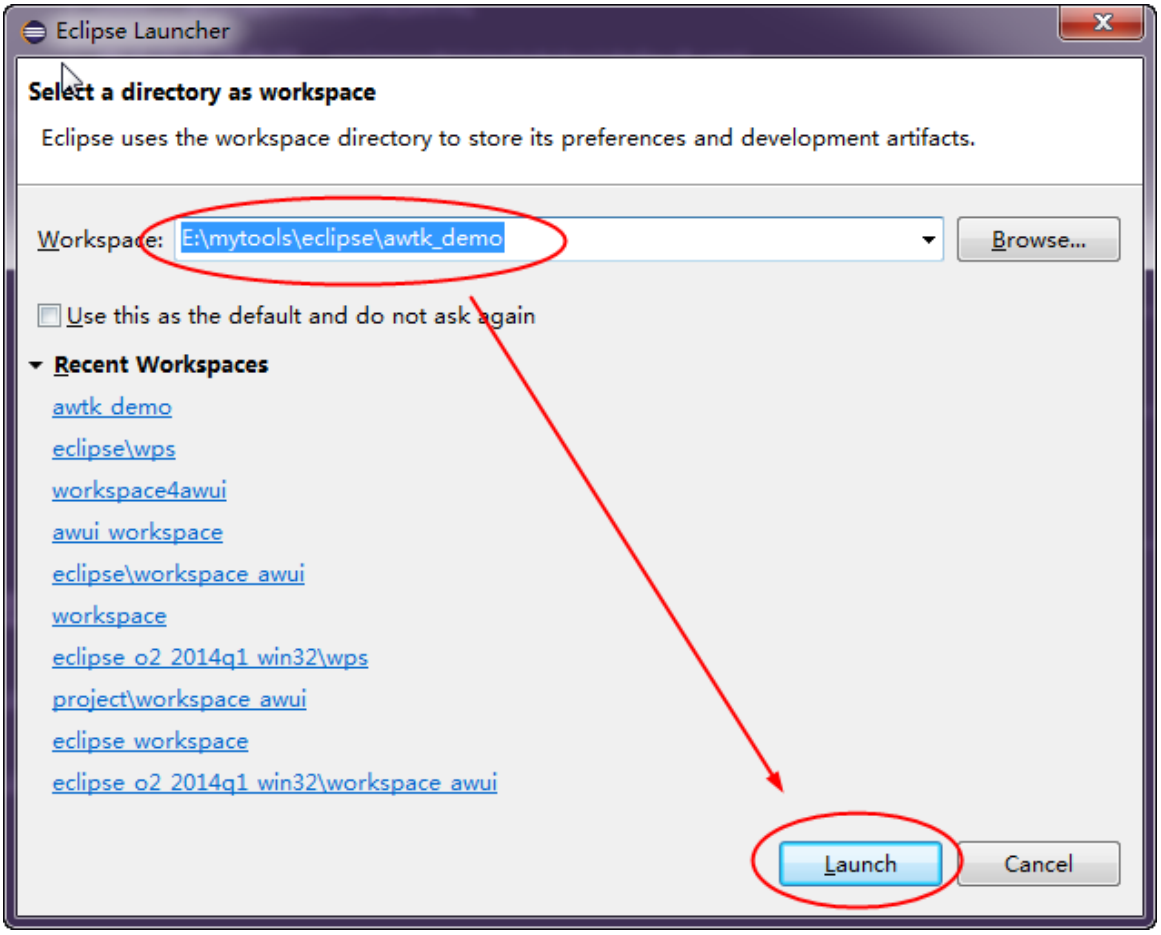



图附录 A.1 M1052 系列光盘资料 V1.02

2. 下载完后，找到 aworks_m105x_sdk_1.0.2-alpha 文件夹并解压。
3. 删除 aworks_m105x_sdk_1.0.2-alpha\examples\application\app_awtk_demo\src\demo 目录。
4. 将整个工程文件夹（比如 HelloWorld-Demo），复制到 aworks_m105x_sdk_1.0.2-alpha\examples\application\app_awtk_demo\src 目录下。

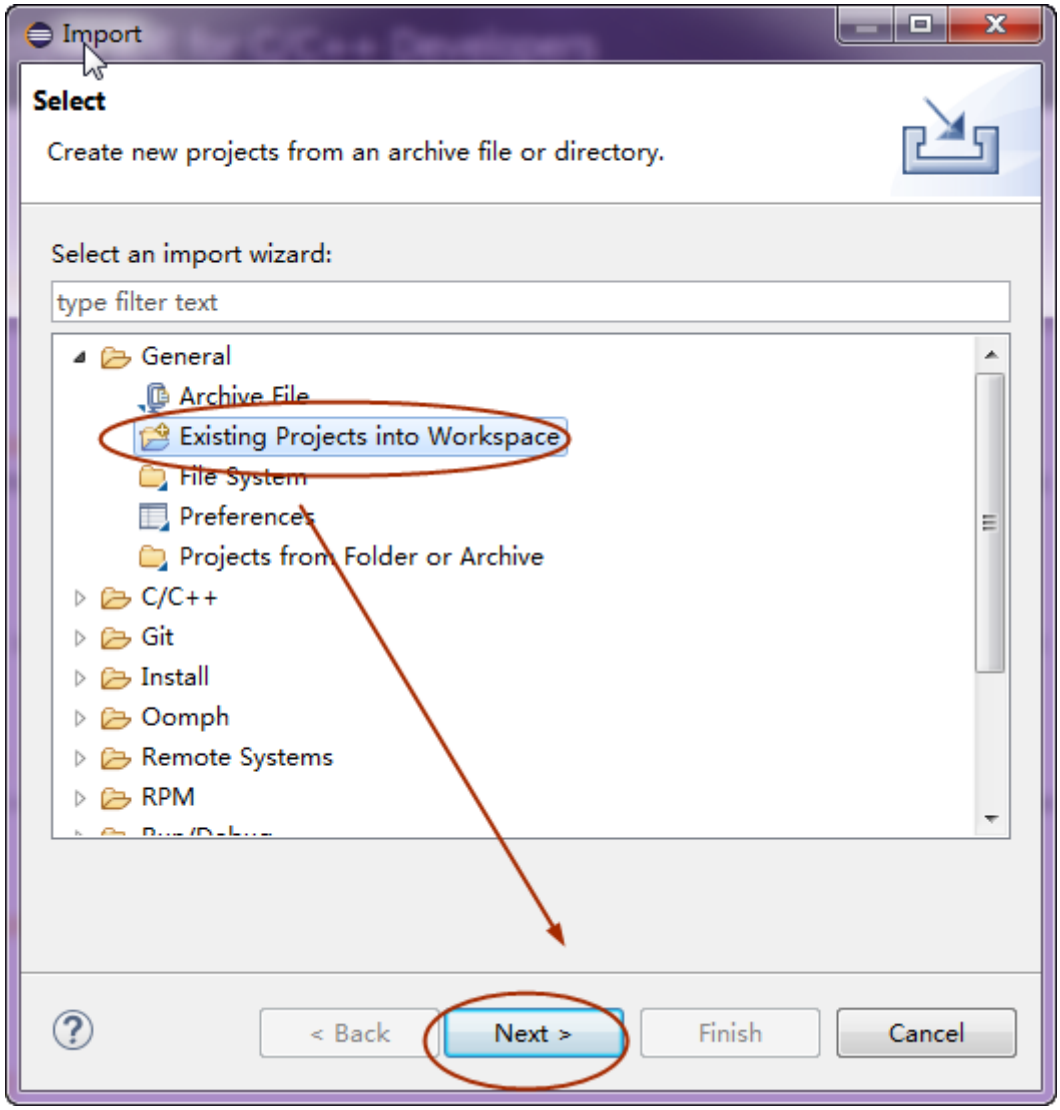
## A.2 建立工程

1. 打开 eclipse 后建立 workspace，可以随便设置一个路径，比如：E:\mytools\eclipse\awtk_demo，详见图附录 A.2。



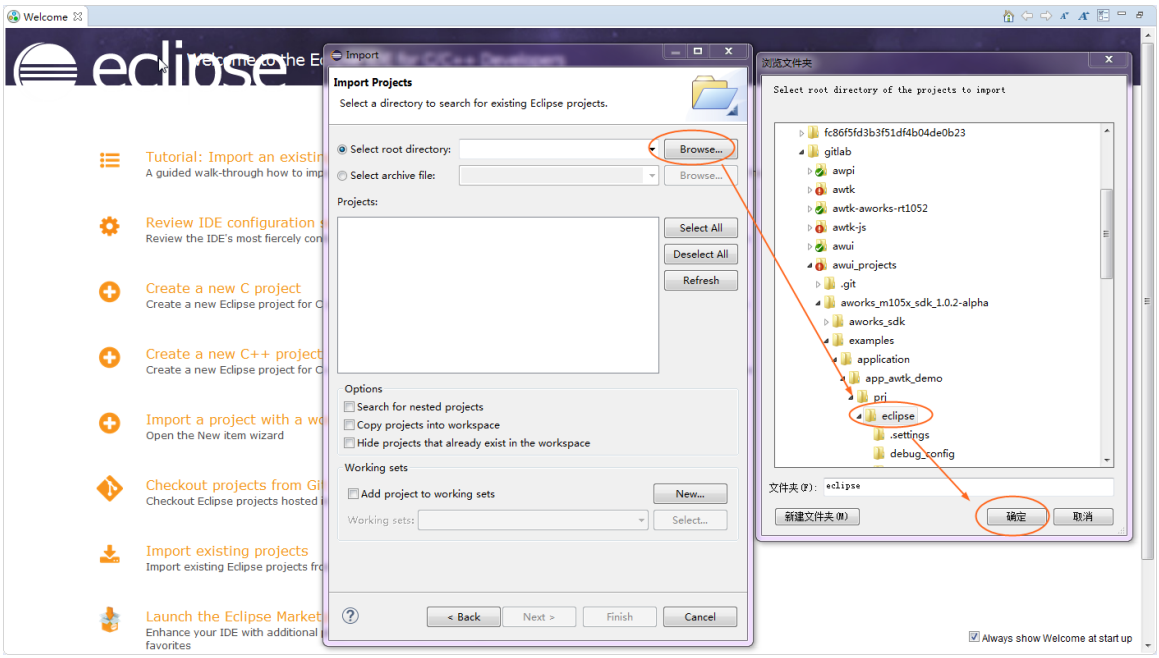

图附录 A.2 建立 workspace

2. 选择菜单 File --> Import 后，详见图附录 A.3。




图附录 A.3 导入工程

3. 选择 aworks_m105x_sdk_1.0.2-alpha\examples\application\app_awtk_demo\prj\eclipse 所在目录，详见图附录 A.4。




图附录 A.4 工程所在目录

4. 选择 Projects 下的目标工程后，点击"Finish"按钮导入。

5. 导入成功后，即可编译，编译完成后烧写到开发板就可以了。

备注：如果有兴趣了解 AWTK 是如何移植到 AWorks 平台，则请下载 awtk-aworks-rt1052: https://github.com/zlgopen/awtk-aworks-rt1052，参阅 README.md 文档即可。

# 附录B 字体裁剪

## B.1 默认字体

在 AWTK 中我们使用一种默认字体： awtk\demos\assets\raw\fonts\default.ttf，如果你觉得这种字体不符合你的审美，你也可以从 C:\Windows\Fonts 目录下找个 ".ttf"格式的字体。比如 STKAITI.TTF，拷贝到我们的应用程序的 assets\raw\fonts 目录下，并重命名为"default.ttf"，作为程序的默认字体（只有在没有定义宏 WITH_MINI_FONT 的时候才生效）。

## B.2 裁剪字体

如果字体文件过大，可以使用 fonttools 中的 pyftsubset 进行裁剪，步骤如下：

1. 使用 python 的 pip 命令进行安装，在 cmd 下运行下列命令：

```
pip install fonttools
```

2. 打开 assets\raw\fonts\text.txt，在里面输入程序要使用的字符。
3. 使用 pyftsubset 工具进行裁剪，在 cmd 下运行下列命令（使用的路径视具体情况而定），生成 assets\raw\fonts\default.mini.ttf。

```
pyftsubset C:/Windows/Fonts/STKAITI.TTF --text-file=./src/assets/raw/fonts/text.txt
--output-file=./src/assets/raw/fonts/default.mini.ttf
```

4. 然后修改应用程序下的 SConstruct 文件，添加宏 WITH_STB_FONT，如下：

```
#SConstruct
...
COMMON_CCFLAGS=COMMON_CCFLAGS+' -DSTBTT_STATIC -DSTB_IMAGE_STATIC
-DWITH_STB_IMAGE -DWITH_STB_FONT -DWITH_MINI_FONT -DWITH_VGCANVAS
-DWITH_UNICODE_BREAK
...
```

# 附录C 启用鼠标指针

## C.1 demo 启用鼠标指针

在 demo_main.c 中有下列代码，要启用鼠标指针，只需要定义宏 ENABLE_CURSOR 即可，如下：

```
// awtk\demos\demo_main.c
#ifdef ENABLE_CURSOR
window_manager_set_cursor(window_manager(), "cursor");
#endif /*ENABLE_CURSOR*/
```

## C.2 在应用程序启用鼠标指针

- 如果应用程序没有走 demo_main 的流程，可以调用 window_manager_set_cursor 或 widget_cursor
- 在任何时候，可以根据需要切换鼠标指针

## C.3 指针图片的要求

指针的形状有多种，比如普通指针、十字、文本选择和等待等等。为了简化处理，在绘制指针时，我们把图片的中心定位到鼠标的当前位置，在制作指针图片时，请考虑这一点。

# 附录D 基本函数库

AWTK 为我们提供的 API 接口，具体列表详见表附录 D.1。

表附录 D.1 AWTK 中的 API 接口

| 选项 | 说明 |
| --- | --- |
| memory | 内存管理相关函数和宏 |
| rgba | RGBA 颜色值 |
| color | 颜色对象 |
| color_parse | 颜色解析对象 |
| value | 通用值对象 |
| str | UTF-8 字符串对象 |
| wstr | Unicode 字符串对象 |
| darray | 动态数组对象 |
| wbuffer | Write |
| rbuffer | Read |
| date_time | 时间日期函数 |
| event | 事件基类 |
| slist | 单向链表 |

## D.1 memory

内存管理相关的宏和函数。

### D.1.1 函数

AWTK 对 malloc、realloc 等函数已经做好了封装，建议使用 AWTK 提供的内存管理函数，详见表附录 D.2，具体的 API 函数参数请看：awtk\docs\manual\tk_mem_t.md，这里不累赘了。

表附录 D.2 内存管理函数列表

| 函数 | 说明 |
| --- | --- |
| TKMEM_ALLOC | 分配一块内存 |
| TKMEM_CALLOC | 分配一块内存，并将内容清零 |
| TKMEM_FREE | 释放内存 |
| TKMEM_REALLOC | 重新分配一块内存，如果原来的内存块大于等于需要的空间，直接返回原来的内存块 |

### D.1.2 示例

下面是一个简单的内存分配和释放示例，代码如下：

```
char* str = (char*)TKMEM_ALLOC(100);
...
TKMEM_FREE(str);
```

## D.2 rgba

颜色的四个通道。

### D.2.1 属性

rgba 提供的属性详见表附录 D.3，具体的 API 函数参数请看：awtk\docs\manual\rgba _t.md，这里不累赘了。

表附录 D.3 rgba 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| a | uint8_t | alpha |
| b | uint8_t | 蓝色 |
| g | uint8_t | 绿色 |
| r | uint8_t | 红色 |

## D.3 color

颜色。

### D.3.1 函数

color 提供的函数详见表附录 D.4，具体的 API 函数参数请看：awtk\docs\manual\color _t.md，这里不累赘了。

表附录 D.4 color 函数列表

| 函数 | 说明 |
| --- | --- |
| color_init | 初始化颜色对象 |

### D.3.2 属性

color 还提供了下面属性，详见表附录 D.5。

表附录 D.5 color 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| color | uint32_t | 颜色的数值 |
| rgba | rgba_t | 颜色的 RGBA 值 |

### D.3.3 示例

下面是一个简单的初始化 color 的示例，代码如下：

```
color_t scolor = color_init(0xff, 0, 0, 0xff);
```

## D.4 color_parse

颜色解析相关函数。

### D.4.1 函数

color_parse 提供的函数详见表附录 D.6，具体的 API 函数参数请：awtk\docs\manual\color_parser _t.md，这里不累赘了。

表附录 D.6 color_parse 函数列表

| 函数 | 说明 |
| --- | --- |
| color_parse | 把字符串格式的颜色转换成 color_t 对象 |

### D.4.2 示例

下面是一个简单的颜色解析示例，代码如下：

```
color_t c;
c = color_parse("#112233");
c = color_parse("white");
c = color_parse("rgb(11,22,33)");
c = color_parse("rgba(11,22,33,0.5)");
```

## D.5 value

一个通用数据类型，用来存放整数、浮点数、字符串和其它对象。

### D.5.1 函数

value 提供的函数详见表附录 D.7，具体的 API 函数参数请看：awtk\docs\manual\value_t.md，这里不累赘了。

表附录 D.7 value 函数列表

| 函数 | 说明 |
| --- | --- |
| value_bool | 获取类型为 bool 的值 |
| value_copy | 拷贝 value 的值 |
| value_create | 创建 value 对象 |
| value_destroy | 销毁 value 对象 |
| value_double | 获取类型为 double 的值 |
| value_equal | 判断两个 value 是否相同 |
| value_float | 获取类型为 float_t 的值 |
| value_float32 | 获取类型为 float 的值 |
| value_int | 转换为 int 的值 |
| value_int16 | 获取类型为 int16 的值 |
| value_int32 | 获取类型为 int32 的值 |
| value_int64 | 获取类型为 int64 的值 |
| value_int8 | 获取类型为 int8 的值 |
| value_is_null | 判断 value 是否为空值 |
| value_pointer | 获取类型为 pointer 的值 |
| value_set_bool | 设置类型为 bool 的值 |
| value_set_double | 设置类型为 double 的值 |
| value_set_float | 设置类型为 float_t 的值 |
| value_set_float32 | 设置类型为 float 的值 |
| value_set_int | 设置类型为 int 的值 |
| value_set_int16 | 设置类型为 int16 的值 |

| | |
|---|---|
| value_set_int32 | 设置类型为 int32 的值 |
| value_set_int64 | 设置类型为 int64 的值 |
| value_set_int8 | 设置类型为 int8 的值 |
| value_set_pointer | 设置类型为 pointer 的值 |
| value_set_str | 设置类型为字符串的值 |
| value_set_uint16 | 设置类型为 uint16 的值 |
| value_set_uint32 | 设置类型为 uint32 的值 |
| value_set_uint64 | 设置类型为 uint64 的值 |
| value_set_uint8 | 设置类型为 uint8 的值 |
| value_set_wstr | 设置类型为宽字符串的值 |
| value_str | 获取类型为字符串的值 |
| value_uint16 | 获取类型为 uint16 的值 |
| value_uint32 | 获取类型为 uint32 的值 |
| value_uint64 | 获取类型为 uint64 的值 |
| value_uint8 | 获取类型为 uint8 的值 |
| value_wstr | 获取类型为宽字符串的值 |

### D.5.2 示例

下面是一个简单的 value 使用示例，代码如下：

```
value_t v;
value_set_int(&v, 100);
```

## D.6 str

可变长度的 UTF8 字符串。

### D.6.1 函数

str 提供的函数详见表附录 D.8，具体的 API 函数参数请看：awtk\docs\manual\str _t.md，这里不累赘了。

表附录 D.8 str 函数列表

| 函数 | 说明 |
|---|---|
| str_append | 追加字符串 |
| str_append_char | 追加一个字符 |
| str_append_with_len | 追加字符串 |
| str_decode_xml_entity | 对 XML 基本的 entity 进行解码，目前仅支持<>"a;& |
| str_decode_xml_entity_with_len | 对 XML 基本的 entity 进行解码，目前仅支持<>"a;& |
| str_end_with | 判断字符串是否以指定的子串结尾 |
| str_eq | 判断两个字符串是否相等 |
| str_from_float | 用浮点数初始化字符串 |
| str_from_int | 用整数初始化字符串 |
| str_from_value | 用 value 初始化字符串 |
| str_from_wstr | 用 wstr 初始化字符串 |
| str_init | 初始化字符串对象 |

| | |
|---|---|
| str_insert | 插入子字符串 |
| str_insert_with_len | 插入子字符串 |
| str_remove | 删除子字符串 |
| str_replace | 字符串替换 |
| str_reset | 重置字符串为空 |
| str_set | 设置字符串 |
| str_set_with_len | 设置字符串 |
| str_start_with | 判断字符串是否以指定的子串开头 |
| str_to_float | 将字符串转成浮点数 |
| str_to_int | 将字符串转成整数 |
| str_to_lower | 将字符串转成小写 |
| str_to_upper | 将字符串转成大写 |
| str_trim | 去除首尾指定的字符 |
| str_trim_left | 去除首部指定的字符 |
| str_trim_right | 去除尾部指定的字符 |
| str_unescape | 对字符串进行反转义如：把"\n"转换成'\n' |

### D.6.2 属性

str 还提供了下面属性，详见表附录 D.9。

表附录 D.9 str 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| capacity | uint32_t | 容量 |
| size | uint32_t | 长度 |
| str | char_t* | 字符串 |

### D.6.3 示例

下面是一个简单的 str 示例，代码如下：

```
str_t s;
str_init(&s, 0);
str_append(&s, "abc");
str_append(&s, "123");
str_reset(&s);
```

## D.7 wstr

可变长度的宽字符字符串。

### D.7.1 函数

wstr 提供的函数详见表附录 D.10，具体的 API 函数参数请看：awtk\docs\manual\wstr_t.md，这里不累赘了。

表附录 D.10 wstr 函数列表

| 函数 | 说明 |
|---|---|

| | |
|---|---|
| wstr_add_float | 将字符串转成浮点数，加上 delta，再转换回来 |
| wstr_append | 追加字符串 |
| wstr_append_with_len | 追加字符串 |
| wstr_equal | 判断两个字符是否相同 |
| wstr_from_float | 用浮点数初始化字符串 |
| wstr_from_int | 用整数初始化字符串 |
| wstr_from_value | 用 value 初始化字符串 |
| wstr_get_utf8 | 获取 UTF8 字符串 |
| wstr_init | 初始化字符串对象 |
| wstr_insert | 在指定位置插入字符串 |
| wstr_pop | 删除尾部字符 |
| wstr_push | 追加一个字符 |
| wstr_push_int | 追加一个整数 |
| wstr_remove | 删除指定范围的字符 |
| wstr_reset | 重置字符串为空 |
| wstr_set | 设置字符串 |
| wstr_set_utf8 | 设置 UTF8 字符串 |
| wstr_to_float | 将字符串转成浮点数 |
| wstr_to_int | 将字符串转成整数 |
| wstr_trim_float_zero | 去掉浮点数小数点尾部的零 |

### D.7.2 属性

wstr 还提供了下面属性，详见表附录 D.11。

表附录 D.11 wstr 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| capacity | uint32_t | 容量 |
| size | uint32_t | 长度 |
| str | wchar_t* | 字符串 |

### D.7.3 示例

下面是一个简单的 wstr 的示例，代码如下：

```
wstr_t s;
wstr_init(&s, 0);
wstr_append(&s, L"abc");
wstr_reset(&s);
```

## D.8 darray

动态数组，根据元素个数动态调整数组的容量。

### D.8.1 函数

darray 提供的函数详见表附录 D.12，具体的 API 函数参数请看：awtk\docs\manual\darray_t.md，这里不累赘了。

表附录 D.12 darray 函数列表

| 函数 | 说明 |
| --- | --- |
| darray_clear | 清除全部元素 |
| darray_count | 返回满足条件元素的个数 |
| darray_create | 创建 darray 对象 |
| darray_deinit | 清除全部元素，并释放 elms |
| darray_destroy | 销毁 darray 对象 |
| darray_find | 查找第一个满足条件的元素 |
| darray_find_index | 查找第一个满足条件的元素，并返回位置 |
| darray_init | 初始化 darray 对象 |
| darray_pop | 弹出最后一个元素 |
| darray_push | 在尾巴追加一个元素 |
| darray_remove | 删除第一个满足条件的元素 |
| darray_remove_all | 删除全部满足条件的元素 |
| darray_remove_index | 删除指定位置的元素 |

### D.8.2 属性

darray 还提供了下面属性，详见表附录 D.13。

表附录 D.13 darray 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| capacity | uint32_t | 数组的容量大小 |
| compare | tk_compare_t | 元素比较函数 |
| destroy | tk_destroy_t | 元素销毁函数 |
| elms | void** | 数组中的元素 |
| size | uint32_t | 数组中元素的个数 |

### D.8.3 示例

下面是一个简单的 darray 示例，代码如下：

```
darray_t darray;
darray_init(&darray, 10, destroy, compare);
...
darray_deinit(&darray);
```

用 darray_create 创建时，需要用 darray_destroy 销毁，代码如下：

```
darray_t* darray = darray_create(10, destroy, compare);
...
darray_destroy(darray);
```

## D.9 wbuffer

write buffer 用于数据打包。

### D.9.1 函数

wbuffer 提供的函数详见表附录 D.14，具体的 API 函数参数请看：awtk\docs\manual\wbuffer _t.md，这里不累赘了。

表附录 D.14 wbuffer 函数列表

| 函数 | 说明 |
| --- | --- |
| wbuffer_deinit | 释放资源 |
| wbuffer_init | 初始 wbuffer 对象 |
| wbuffer_init_extendable | 初始 wbuffer 对象，自动扩展 buffer，使用完后需调用 wbuffer_deinit 释放资源 |
| wbuffer_skip | 跳过指定的长度 |
| wbuffer_write_binary | 写入指定长度的二进制数据 |
| wbuffer_write_float | 写入 float 数据 |
| wbuffer_write_string | 写入字符串 |
| wbuffer_write_uint16 | 写入 uint16 数据 |
| wbuffer_write_uint32 | 写入 uint32 数据 |
| wbuffer_write_uint8 | 写入 uint8 数据 |

### D.9.2 属性

wbuffer 还提供了下面属性，详见表附录 D.15。

表附录 D.15 wbuffer 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| capacity | uint32_t | 缓存区最大容量 |
| cursor | uint32_t | 当前写入位置 |
| data | uint8_t* | 缓存区 |
| extendable | bool_t | 容量是否可扩展 |

### D.9.3 示例

下面是一个简单的 wbuffer 示例，代码如下：

```
uint8_t buff[128];
wbuffer_t wbuffer;
rbuffer_t rbuffer;
const char* str = NULL;
wbuffer_init(&wbuffer, buff, sizeof(buff));
wbuffer_write_string(&wbuffer, "hello awtk");
rbuffer_init(&rbuffer, wbuffer.data, wbuffer.cursor);
rbuffer_read_string(&rbuffer, &str);
```

如果初始化为 extendable，则最后需要调用 wbuffer_deinit 释放资源，代码如下：

```
wbuffer_t wbuffer;
wbuffer_init_extendable(&wbuffer);
wbuffer_write_string(&wbuffer, "hello awtk");
wbuffer_deinit(&wbuffer);
```

## D.10 rbuffer

read buffer，用于数据解包。

### D.10.1 函数

rbuffer 提供的函数详见表附录 D.16，具体的 API 函数参数请看：awtk\docs\manual\rbuffer_t.md，这里不累赘了。

表附录 D.16 rbuffer 函数列表

| 函数 | 说明 |
| --- | --- |
| rbuffer_has_more | 判断是否还有数据可读 |
| rbuffer_init | 初始 rbuffer 对象 |
| rbuffer_peek_uint16 | 读取 uint16 数据，但不改变 cursor 的位置 |
| rbuffer_peek_uint32 | 读取 uint32 数据，但不改变 cursor 的位置 |
| rbuffer_peek_uint8 | 读取 uint8 数据， 但不改变 cursor 的位置 |
| rbuffer_read_binary | 读取指定长度的二进制数据 |
| rbuffer_read_float | 读取 float 数据 |
| rbuffer_read_string | 读取字符串 |
| rbuffer_read_uint16 | 读取 uint16 数据 |
| rbuffer_read_uint32 | 读取 uint32 数据 |
| rbuffer_read_uint8 | 读取 uint8 数据 |
| rbuffer_skip | 跳过指定的长度 |

### D.10.2 属性

rbuffer 还提供了下面属性，详见表附录 D.17。

表附录 D.17 rbuffer 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| capacity | uint32_t | 缓存区最大容量 |
| cursor | uint32_t | 当前读取位置 |
| data | uint8_t* | 缓存区 |

### D.10.3 示例

下面是一个简单的 rbuffer 示例，代码如下：

```
uint8_t buff[128];
wbuffer_t wbuffer;
rbuffer_t rbuffer;
const char* str = NULL;
wbuffer_init(&wbuffer, buff, sizeof(buff));
wbuffer_write_string(&wbuffer, "hello awtk");
rbuffer_init(&rbuffer, wbuffer.data, wbuffer.cursor);
rbuffer_read_string(&rbuffer, &str);
```

## D.11 date_time

日期时间。在嵌入式平台中，在系统初始时，需要调用 date_time_set_impl 设置实际获取系统时间的函数。

### D.11.1 函数

date_time 提供的函数详见表附录 D.18，具体的 API 函数参数请看：awtk\docs\manual\date_time _t.md，这里不累赘了。

表附录 D.18 date_time 函数列表

| 函数 | 说明 |
| --- | --- |
| date_time_create | 创建 date_time 对象，并初始为当前日期和时间（一般供脚本语言中使用） |
| date_time_set_impl | 设置获取当前日期和时间的函数 |
| date_time_destroy | 销毁 date_time 对象（一般供脚本语言中使用） |
| date_time_init | 初始为当前日期和时间 |

### D.11.2 属性

date_timer 还提供了下面属性，详见表附录 D.19。

表附录 D.19 date_time 属性列表

| 属性名称 | 类型 | 说明 |
| --- | --- | --- |
| day | int32_t | 日（1-31） |
| hour | int32_t | 时（0） |
| minute | int32_t | 分（0） |
| month | int32_t | 月（1-12） |
| second | int32_t | 秒（0） |
| year | int32_t | 年 |

### D.11.3 示例

下面是一个简单的 date_time 示例，代码如下：

```
date_time_t dt;
date_time_init(&dt);
```

## D.12 event

事件基类。

### D.12.1 函数

event 提供的函数详见表附录 D.20，具体的 API 函数参数请看：awtk\docs\manual\event _t.md，这里不累赘了。

表附录 D.20 event 函数列表

| 函数 | 说明 |
| --- | --- |
| event_cast | 把 event 对象转 wheel_event_t 对象，主要给脚本语言使用 |

| | |
|---|---|
| event_init | 初始化事件 |

### D.12.2 属性

event 还提供了下面属性，详见表附录 D.21。

表附录 D.21 event 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| target | void* | 事件发生的目标对象 |
| time | int32_t | 事件发生的时间 |
| type | int32_t | 类型 |

### D.12.3 示例

下面是一个简单的 event 示例，代码如下：

```
event_t e = event_init(EVT_ANIM_ONCE, animator);
```

## D.13 slist

单向链表。

### D.13.1 函数

slist 提供的函数详见表附录 D.21，具体的 API 函数参数请看：awtk\docs\manual\ slist_t.md，这里不累赘了。

表附录 D.21 slist 函数列表

| 函数 | 说明 |
|---|---|
| slist_append | 在尾巴追加一个元素 |
| slist_count | 返回满足条件元素的个数 |
| slist_create | 创建 slist 对象 |
| slist_deinit | 清除单向链表中的元素 |
| slist_destroy | 清除单向链表中的元素，并释放单向链表对象 |
| slist_find | 查找第一个满足条件的元素 |
| slist_foreach | 遍历元素 |
| slist_init | 初始化 slist 对象 |
| slist_prepend | 在头部追加一个元素 |
| slist_remove | 删除第一个满足条件的元素 |
| slist_remove_all | 删除全部元素 |
| slist_size | 返回元素个数 |

### D.13.2 属性

slist 还提供了下面属性，详见表附录 D.22。

表附录 D.22 slist 属性列表

| 属性名称 | 类型 | 说明 |
|---|---|---|
| compare | tk_compare_t | 元素比较函数 |

| | | |
|---|---|---|
| destroy | tk_destroy_t | 元素销毁函数 |
| first | slist_node_t* | 首节点 |

### D.13.3 示例

用 slist_init 初始化时，用 slist_deinit 释放。如：

```
slist_t slist;
slist_init(&slist, destroy, compare);
...
slist_deinit(&slist);
```

用 slist_create 创建时，用 slist_destroy 销毁。如：

```
slist_t* slist = slist_create(destroy, compare);
...
slist_destroy(slist);
```