

# ElaWidgetTools 组件库软件用户手册

软件标识: ElaWidgetTools-LK-2024

开发人员: \_\_\_\_\_Ela\_\_\_\_\_

2024 年 8 月 15 日

## 目 录

1 范围 .....	10
1.1 标识.....	10
1.2 系统概述.....	10
1.3 文档概述.....	10
2 引用文档.....	10
3 软件综述.....	11
3.1 软件应用.....	11
3.2 软件清单.....	11
3.3 软件环境.....	11
3.4 软件组织和操作概述.....	11
3.5 意外事故及运行的备用状态和方式.....	11
3.6 保密性.....	12
3.7 帮助和问题报告.....	12
4 软件入门.....	12
4.1 软件的首次用户.....	12
4.1.1 熟悉设备.....	12
4.1.2 访问控制.....	13
4.1.3 安装和设置.....	13
4.2 启动.....	17
4.3 停止和挂起.....	17
5 使用指南.....	17
5.1 能力.....	17
5.2 约定.....	17
5.3 处理规程.....	18
5.3.1 软件整体一览.....	18
5.3.2 有关的处理.....	18
5.4 数据备份.....	18
5.5 消息.....	18
6 附录 .....	19
6.1 ElaAppBar.....	19
6.1.1 组件说明.....	19
6.1.2 枚举定义.....	19
6.1.3 属性成员.....	20

6.1.4 公有函数.....	22
6.1.5 公有信号.....	23
6.2 ElaAcrylicUrlCard .....	24
6.2.1 组件说明.....	24
6.2.2 枚举定义.....	24
6.2.3 属性成员.....	24
6.2.4 公有函数.....	28
6.3 ElaApplication .....	29
6.3.1 组件说明.....	29
6.3.2 属性成员.....	29
6.3.3 公有函数.....	29
6.4 ElaBreadcrumbBar.....	30
6.4.1 组件说明.....	30
6.4.2 公有函数.....	30
6.4.3 公有信号.....	31
6.5 ElaCheckBox .....	31
6.5.1 组件说明.....	31
6.6 ElaColorDialog .....	31
6.6.1 组件说明.....	31
6.6.2 属性成员.....	32
6.6.3 公有函数.....	32
6.6.4 公有信号.....	32
6.7 ElaComboBox.....	33
6.7.1 组件说明.....	33
6.7.2 属性成员.....	33
6.8 ElaContentDialog.....	33
6.8.1 组件说明.....	33
6.8.2 公有函数.....	33
6.8.3 公有信号.....	35
6.9 ElaDockWidget.....	35
6.9.1 组件说明.....	35
6.10 ElaDockWidget.....	35
6.10.1 组件说明.....	35
6.11 ElaDoubleSpinBox.....	35
6.11.1 组件说明.....	35

6.12 ElaDxgiManager .....	35
6.12.1 组件说明 .....	35
6.12.2 公有函数 .....	36
6.12.3 公有信号 .....	39
6.13 ElaDxgiScreen .....	39
6.13.1 组件说明 .....	39
6.13.2 属性成员 .....	39
6.13.3 公有函数 .....	40
6.14 ElaEvent .....	40
6.14.1 组件说明 .....	40
6.14.2 属性成员 .....	40
6.14.3 公有函数 .....	41
6.15 ElaEventBus .....	41
6.15.1 组件说明 .....	41
6.15.2 枚举定义 .....	42
6.15.3 公有函数 .....	42
6.16 ElaExponentialBlur .....	43
6.16.1 组件说明 .....	43
6.16.2 公有函数 .....	43
6.17 ElaFlowLayout .....	43
6.17.1 组件说明 .....	43
6.17.2 公有函数 .....	43
6.18 ElaGraphicsItem .....	43
6.18.1 组件说明 .....	43
6.18.2 属性成员 .....	44
6.18.3 公有函数 .....	46
6.19 ElaGraphicsLineItem .....	47
6.19.1 组件说明 .....	47
6.19.2 属性成员 .....	47
6.19.3 公有函数 .....	49
6.20 ElaGraphicsScene .....	50
6.20.1 组件说明 .....	50
6.20.2 枚举定义 .....	50
6.20.3 属性成员 .....	50
6.20.4 公有函数 .....	51

6.20.5 公有信号 .....	55
6.21 ElaGraphicsView .....	56
6.21.1 组件说明 .....	56
6.21.2 属性成员 .....	56
6.22 ElaIcon .....	56
6.22.1 组件说明 .....	56
6.22.2 枚举定义 .....	57
6.22.3 公有函数 .....	57
6.23 ElaIconButton .....	59
6.23.1 组件说明 .....	59
6.23.2 属性成员 .....	59
6.23.3 公有函数 .....	61
6.24 ElaImageCard .....	61
6.24.1 组件说明 .....	61
6.24.2 属性成员 .....	62
6.25 ElaInteractiveCard .....	63
6.25.1 组件说明 .....	63
6.25.2 属性成员 .....	63
6.25.3 公有函数 .....	66
6.26 ElaLineEdit .....	66
6.26.1 组件说明 .....	66
6.26.2 属性成员 .....	66
6.26.3 公有信号 .....	67
6.27 ElaListView .....	67
6.27.1 组件说明 .....	67
6.27.2 属性成员 .....	67
6.28 ElaLog .....	68
6.28.1 组件说明 .....	68
6.28.2 属性成员 .....	68
6.28.3 公有函数 .....	69
6.28.4 公有信号 .....	69
6.29 ElaMenu .....	70
6.29.1 组件说明 .....	70
6.29.2 公有函数 .....	70
6.29.3 公有信号 .....	72

6.30 ElaMenuBar .....	72
6.30.1 组件说明 .....	72
6.30.2 公有函数 .....	72
6.31 ElaMessageBar .....	73
6.31.1 组件说明 .....	73
6.31.2 枚举定义 .....	74
6.31.3 公有函数 .....	74
6.32 ElaMessageButton .....	76
6.32.1 组件说明 .....	76
6.32.2 属性成员 .....	76
6.33 ElaMultiSelectComboBox .....	78
6.33.1 组件说明 .....	78
6.33.2 属性成员 .....	78
6.33.3 公有函数 .....	79
6.33.4 公有信号 .....	80
6.34 ElaNavigationBar .....	80
6.34.1 组件说明 .....	80
6.34.2 枚举定义 .....	80
6.34.3 属性成员 .....	81
6.34.4 公有函数 .....	81
6.34.5 公有信号 .....	87
6.35 ElaNavigationRouter .....	88
6.35.1 组件说明 .....	88
6.35.2 枚举定义 .....	88
6.35.3 属性成员 .....	88
6.35.4 公有函数 .....	89
6.35.5 公有信号 .....	90
6.36 ElaPivot .....	90
6.36.1 组件说明 .....	90
6.36.2 属性成员 .....	90
6.36.3 公有函数 .....	91
6.36.4 公有信号 .....	92
6.37 ElaPlainTextEdit .....	92
6.37.1 组件说明 .....	92
6.38 ElaPopularCard .....	92

6.38.1 组件说明.....	92
6.38.2 属性成员.....	92
6.38.3 公有信号.....	95
6.39 ElaProgressBar.....	95
6.39.1 组件说明.....	95
6.40 ElaPromotionCard.....	95
6.40.1 组件说明.....	95
6.40.2 属性成员.....	95
6.40.3 公有信号.....	100
6.41 ElaPromotionView .....	101
6.41.1 组件说明.....	101
6.41.2 属性成员.....	101
6.41.3 公有函数.....	102
6.42 ElaPushButton.....	102
6.42.1 组件说明.....	102
6.42.2 属性成员.....	103
6.42.3 公有函数.....	105
6.43 ElaRadioButton.....	105
6.43.1 组件说明.....	105
6.44 ElaReminderCard.....	105
6.44.1 组件说明.....	105
6.44.2 属性成员.....	106
6.44.3 公有函数.....	109
6.45 ElaScrollArea.....	109
6.45.1 组件说明.....	109
6.45.2 公有函数.....	109
6.46 ElaScrollBar.....	110
6.46.1 组件说明.....	110
6.46.2 公有信号.....	110
6.47 ElaScrollPage.....	110
6.47.1 组件说明.....	110
6.47.2 属性成员.....	111
6.47.3 公有函数.....	111
6.48 ElaScrollPageArea .....	112
6.48.1 组件说明.....	112

6.48.2 属性成员.....	112
6.49 ElaSlider.....	113
6.49.1 组件说明.....	113
6.50 ElaSpinBox.....	113
6.50.1 组件说明.....	113
6.51 ElaStatusBar.....	113
6.51.1 组件说明.....	113
6.52 ElaSuggestBox.....	113
6.52.1 组件说明.....	113
6.52.2 属性成员.....	113
6.52.3 公有函数.....	114
6.52.4 公有信号.....	115
6.53 ElaTabBar.....	115
6.53.1 组件说明.....	115
6.54 ElaTableView.....	115
6.54.1 组件说明.....	115
6.54.2 属性成员.....	115
6.54.3 公有信号.....	116
6.55 ElaTabWidget.....	116
6.55.1 组件说明.....	116
6.56 ElaText.....	116
6.56.1 组件说明.....	116
6.56.2 枚举定义.....	116
6.56.3 公有函数.....	117
6.57 ElaTheme.....	118
6.57.1 组件说明.....	118
6.57.2 枚举定义.....	118
6.57.3 公有函数.....	118
6.57.4 公有信号.....	119
6.58 ElaToggleButton.....	120
6.58.1 组件说明.....	120
6.58.2 属性成员.....	120
6.58.3 公有函数.....	120
6.58.4 公有信号.....	121
6.59 ElaSwitchButton.....	121



6.59.1 组件说明.....	121
6.59.2 公有函数.....	121
6.59.3 公有信号.....	121
6.60 ElaToolBar.....	122
6.60.1 组件说明.....	122
6.60.2 公有函数.....	122
6.61 ElaToolButton.....	123
6.61.1 组件说明.....	123
6.61.2 公有函数.....	123
6.62 ElaTreeView.....	123
6.62.1 组件说明.....	123
6.62.2 属性成员.....	123
6.63 ElaWidget.....	124
6.63.1 组件说明.....	124
6.63.2 公有函数.....	124
6.64 ElaWindow.....	125
6.64.1 组件说明.....	125
6.64.2 属性成员.....	125
6.64.3 公有函数.....	126
6.64.4 公有信号.....	135

## 1 范围

### 1.1 标识

软件名称：ElaWidgetTools;

软件标识序列号：ElaWidgetTools-LK-2024;

软件版本号：0X24081501;

### 1.2 系统概述

本软件为基于 QT-Widget 框架开发的 FluentUI 风格组件库，提供 Fluent 风格的大部分 QT 基础组件、创意性进阶组件和实用功能封装，旨在协助用户对项目进行快速美化或集成开发需求；并以动态库或静态库的形式提供于用户使用；本软件开发时间为 2024 年 4 月 10 日，开源时间为 2024 年 6 月 10 日；软件版权实际持有人为 QQ: 8009963、GitHub: Liniyous；开发者同上。

### 1.3 文档概述

本文档为 ElaWidgetTools 组件库（以下统称为 ElaWidgetTools）软件用户使用手册，用于帮助软件的新用户快速了解、熟悉软件的基本、进阶功能，并投入实战。

## 2 引用文档

GJB 438C-2021 军用软件开发文档通用要求

## 3 软件综述

### 3.1 软件应用

本软件主要应用于 QT-Widget 框架程序的快速开发和美化，旨在为用户提供高度自定义化、高性能、可控、快速的开发体验；

本软件的所有组件、功能开发时均以性能为第一原则，所有外观组件均使用 QStyle 进行重绘，相较于传统 QSS 开发，性能呈指数倍提升；旨在尽可能不影响用户业务逻辑的情况下提供良好的动画效果、视觉观感及使用体验。

### 3.2 软件清单

Example 例子程序 \* 1；

Src 组件库源码 \* 1；

### 3.3 软件环境

软件运行应提供的基础计算机设备和环境：

- a) 1920x1080 分辨率或以上的显示器一个；
- b) 108 键或以上的标准键盘一个；
- c) I5-7500 或以上的 CPU；
- d) 2GB-DDR4/DDR5 或以上的运行内存；
- e) 1GB 或以上的剩余硬盘空间；
- f) Ubuntu22.04LTS 或以下、Windows 10 和 Windows 11 任意版本操作系统，支持虚拟机；
- g) QT5.12 以上的 QT 依赖；Linux 系统最低为 QT5.15；

### 3.4 软件组织和操作概述

- a) 典型响应时间：对用户操作响应时间为 1ms 以下；
- b) 典型处理时间：2ms-200ms，取决于用户的硬件配置。
- c) 预期的错误率：5%。
- d) 预期的可靠性：95%；

### 3.5 意外事故及运行的备用状态和方式

若在使用用户过程中，组件出现异常情况或崩溃情况，请先参照本手册第五节确认

使用方法是否正确，如使用方式无误，请直接联系作者咨询，确认为组件异常后会尽快进入修正流程。

### 3.6 保密性

您需知悉并同意以下条款，否则您不会被继续授权阅读或使用本文档，以下条款在您获取本文档之前应当已被提供：

- a) 2024 年 8 月 15 日起声明，本软件版权实际持有人（QQ: 8009963 GitHub: Liniyous），特此向任何得到 ElaWidgetTools 软件拷贝及相关文档（以下统称“本软件”；且本文档不包含在软件的相关文档中）的人授权：被授权人有权使用、复制、修改、合并、发布、发行、再许可、售卖本软件的拷贝、并有权向被供应人授予同等的权利，但必须满足以下条件：在本软件的所有副本或实质性使用中，都必须包含以上版权声明和本授权声明；若您向本软件作者购买了商业许可授权，特授权您无需在本软件的所有副本或实质性使用中包含以上版权声明和本授权声明。本软件是“按原样”提供的，不附带任何明示或暗示的保证，包括没有任何有关适销性、适用性、非侵权性保证以及其他保证。在任何情况下，作者或版权持有人，对任何权益追索、损害赔偿以及其他追责，都不负任何责任。无论这些追责产生自合同、侵权，还是直接或间接来自于本软件以及与本软件使用或经营有关的情形。
- b) 在任何情况下，您不应对外发送除本软件作者或已被授权使用本文档的用户外的任何人发送本文档的副本、拷贝、截图、同义文档、或任意修改后的版本，否则，收回第一条中的商业授权许可，不再提供后续更新，且保留追究造成作者经济损失的权力。

### 3.7 帮助和问题报告

如果用户在使用中遇到无法解决、不能正常使用的问题，请联系软件作者告知情况，若确定因软件自身原因引起的异常问题，会尽快进入修正流程；

## 4 软件入门

### 4.1 软件的首次用户

#### 4.1.1 熟悉设备

当您首次使用本软件时，您需进行一系列的基础环境配置：

- a) 确保您已安装或存在 QT5.12 及以上的依赖环境；Linux 系统为 QT5.15 以上；

- b) 若使用 MSVC 编译器，确认您安装了对应版本的 Visual-Studio MSVC 辅助生成器；
- c) 确保您已安装或存在 CMAKE 环境，最低版本为 CMAKE3.5；

4.1.2 访问控制

本软件无使用权限要求，暂无访问控制模块。

4.1.3 安装和设置

下文为使用 QTCreator 在 QT5.14 及以上版本的示例和源码编译流程，若使用 VS 或 CLion 进行编译，请参考 [https://www.bilibili.com/video/BV1EwakesEJR/?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1EwakesEJR/?spm_id_from=333.999.0.0)；

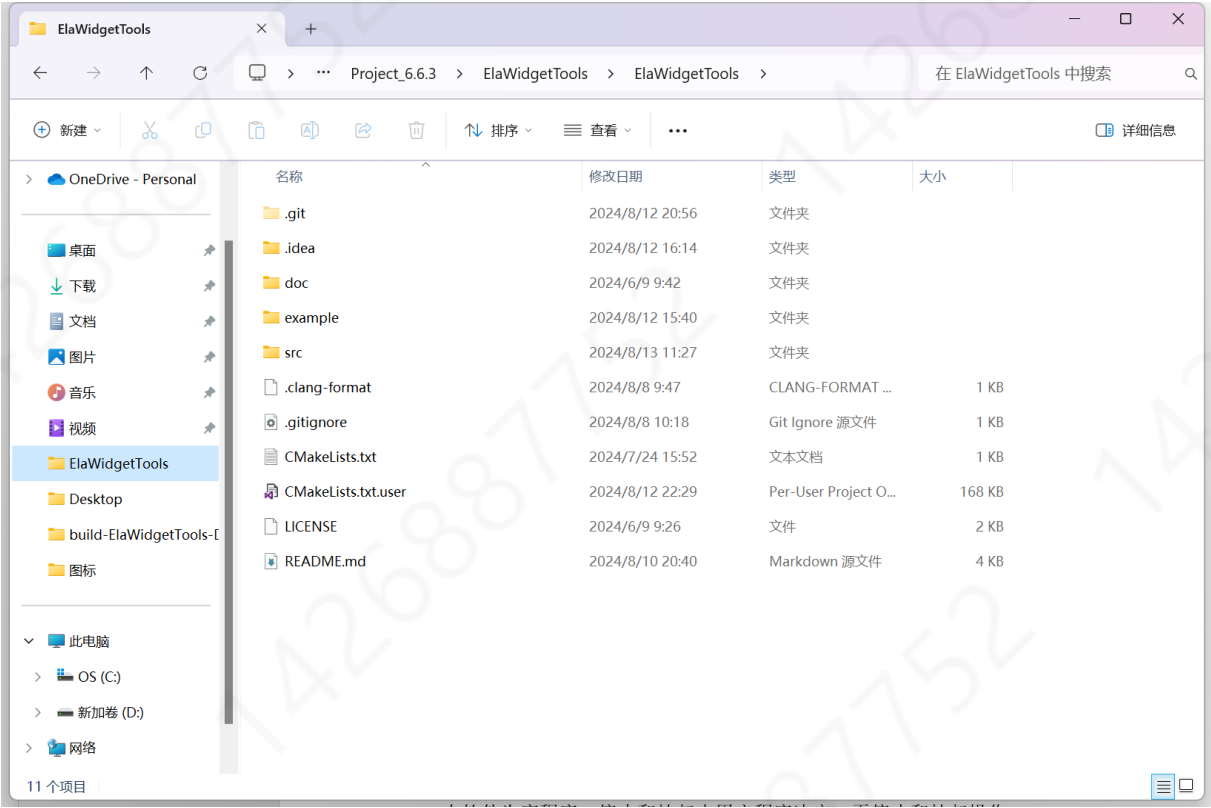


图 1 源码包结构一览

如图 1 所示，源码包包括 example 例子源码文件夹、src 库源码文件夹、CMakeLists.txt 配置文件、LICENSE、和 README 文件；

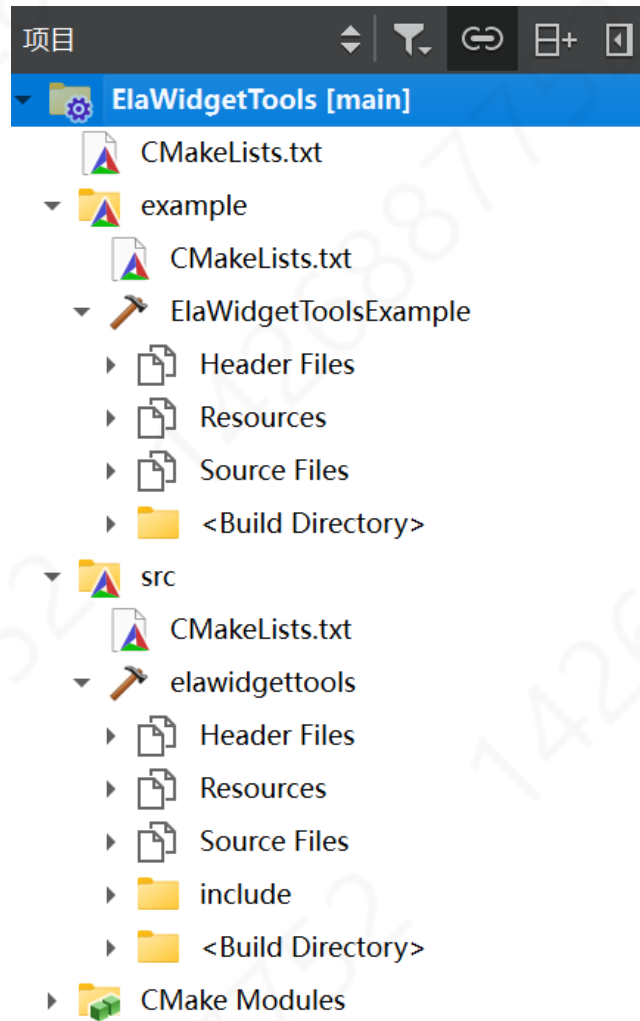
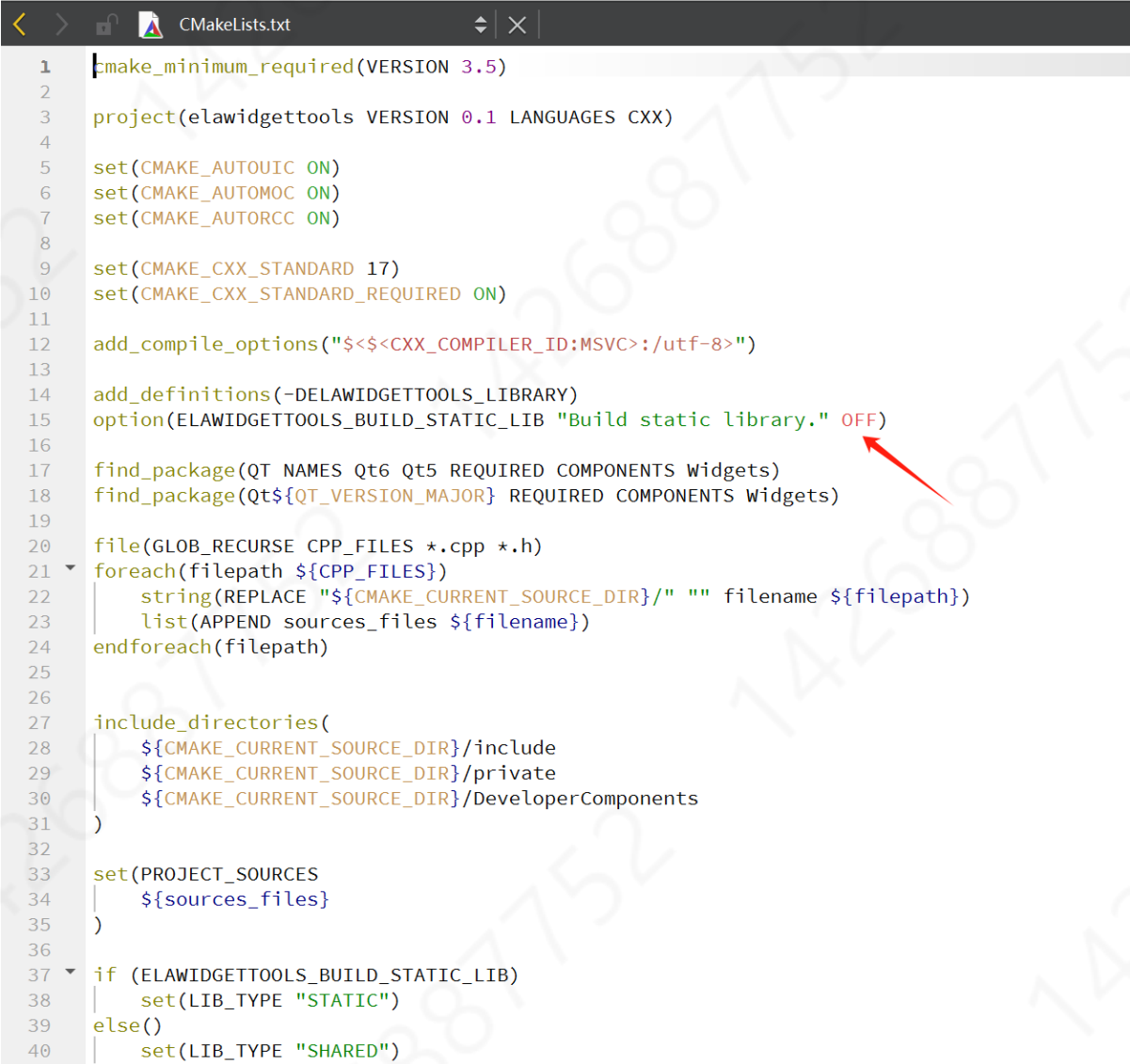


图 2 项目结构

使用 QtCreator，打开该目录下的 CMakeLists.txt 文件，项目结构如图 2 所示；



```
1 cmake_minimum_required(VERSION 3.5)
2
3 project(elawidgettools VERSION 0.1 LANGUAGES CXX)
4
5 set(CMAKE_AUTOUIC ON)
6 set(CMAKE_AUTOMOC ON)
7 set(CMAKE_AUTORCC ON)
8
9 set(CMAKE_CXX_STANDARD 17)
10 set(CMAKE_CXX_STANDARD_REQUIRED ON)
11
12 add_compile_options("$<$<CXX_COMPILER_ID:MSVC>:/utf-8>")
13
14 add_definitions(-DELAWIDGETTOOLS_LIBRARY)
15 option(ELAWIDGETTOOLS_BUILD_STATIC_LIB "Build static library." OFF)
16
17 find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets)
18 find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets)
19
20 file(GLOB_RECURSE CPP_FILES *.cpp *.h)
21 foreach(filepath ${CPP_FILES})
22     string(REPLACE "${CMAKE_CURRENT_SOURCE_DIR}/" "" filename ${filepath})
23     list(APPEND sources_files ${filename})
24 endforeach(filepath)
25
26
27 include_directories(
28     ${CMAKE_CURRENT_SOURCE_DIR}/include
29     ${CMAKE_CURRENT_SOURCE_DIR}/private
30     ${CMAKE_CURRENT_SOURCE_DIR}/DeveloperComponents
31 )
32
33 set(PROJECT_SOURCES
34     ${sources_files}
35 )
36
37 if (ELAWIDGETTOOLS_BUILD_STATIC_LIB)
38     set(LIB_TYPE "STATIC")
39 else()
40     set(LIB_TYPE "SHARED")
41 end
```

图 3 修改编译类型

打开 src 文件夹下的 CMakeLists.txt，配置需要使用的库类型（默认和推荐形式为动态库），如图 3 所示；修改 ELAWIDGETTOOLS\_BUILD\_STATIC\_LIB 的属性以修改库类型，默认为 OFF，动态库，设置为 ON 即为静态库；

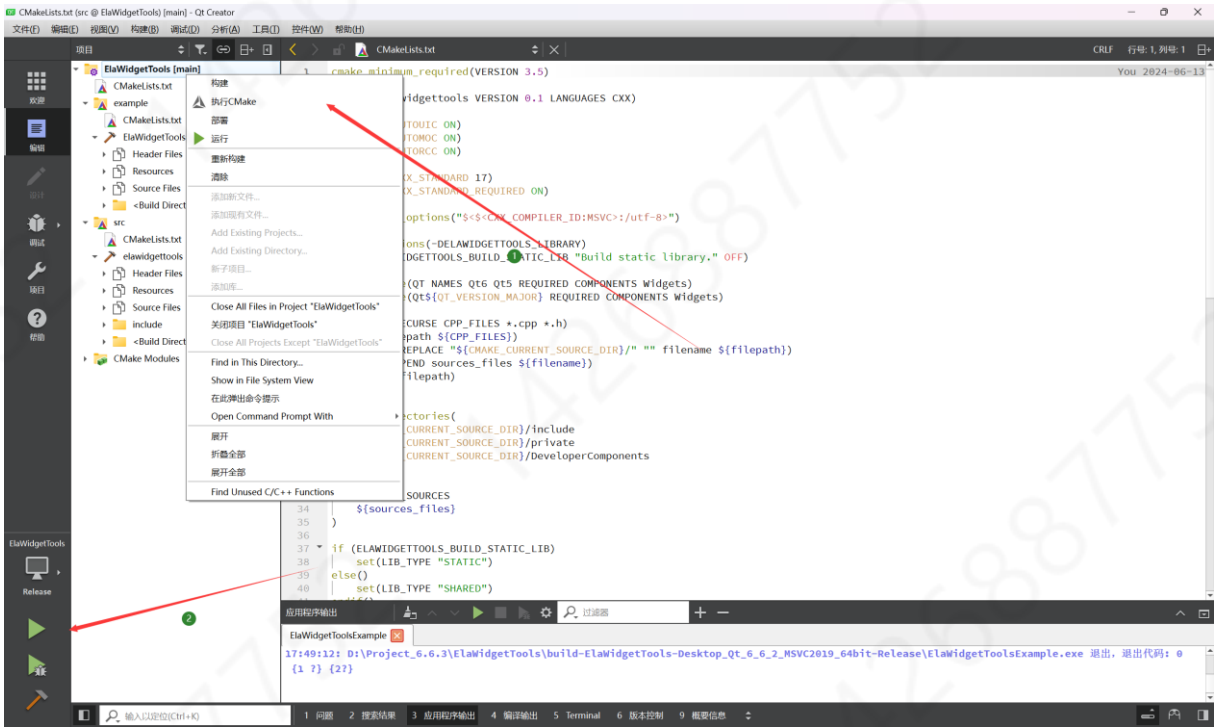


图 4 执行 CMake 并编译

右键执行最顶层的 CMakeList，而后点击编译/运行按钮，如图 4 所示；

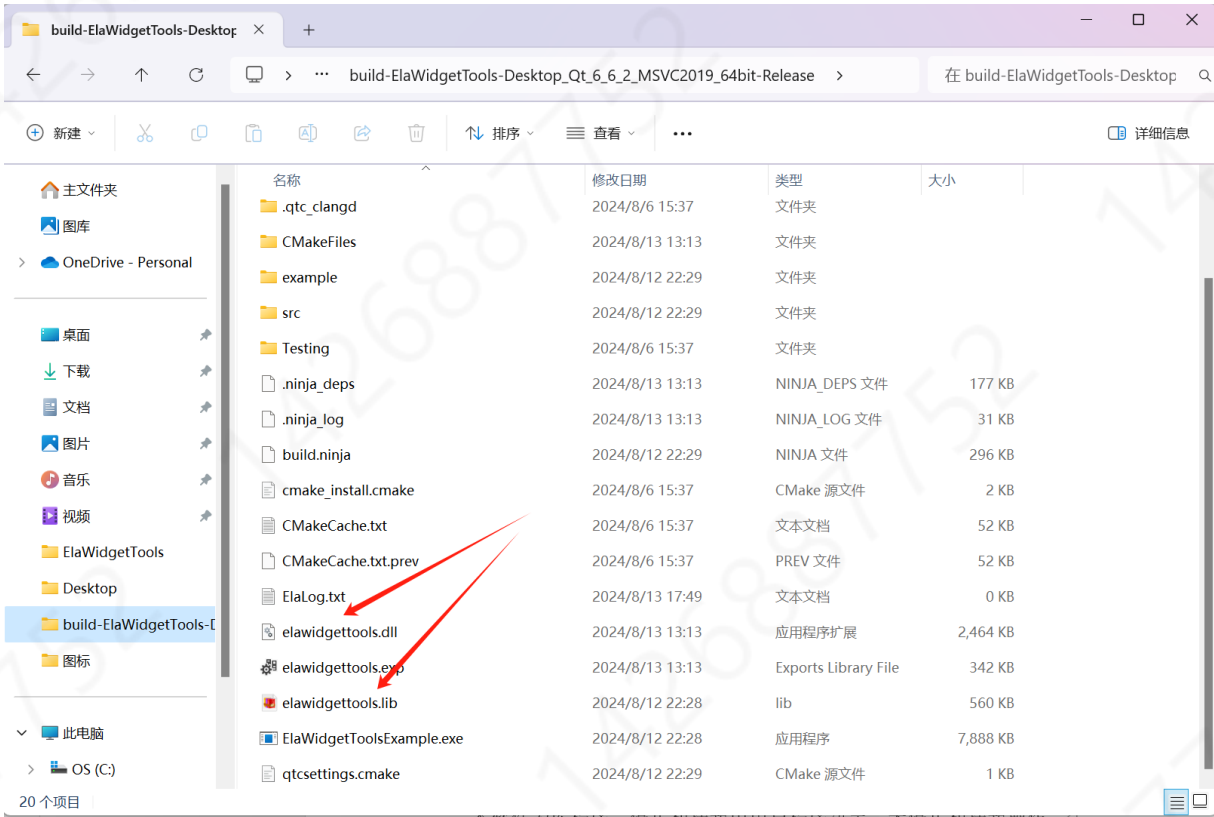


图 5 编译结果

编译完成后，打开程序的构建文件夹，如图 5 所示，`elawidgettools.dll` 和 `elawidgettools.lib` 即为目标库，将其和源码目录下 `src` 文件夹中的 `include` 文件夹一同加入到项目中



**即可使用**；需注意，QT6 以下的版本需要将 include 文件夹中的 qrc 文件一同加入项目中，QT6 以上的版本无需加入 qrc。

## 4.2 启动

在使用本库创建项目后，需最先进行初始化操作，包含 **ElaApplication.h** 头文件，在 QApplication 实例创建后，调用 **ElaApplication::init()** 函数进行初始化（eApp->init() 亦可），方可正常使用本库。

## 4.3 停止和挂起

本软件为库程序，停止和挂起由用户程序决定，无停止和挂起操作。

# 5 使用指南

## 5.1 能力

本软件主要分为三个功能模块。

## 5.2 约定

- a) 本软件为全中文界面库，并且不提供其他任何语言的翻译功能；
- b) 您知悉并同意本软件使用一定的硬盘存储空间用于存储软件 Log 相关信息；
- c) 您知悉并同意 2.6 节中对于软件使用的要求；

## 5.3 处理规程

### 5.3.1 软件整体一览

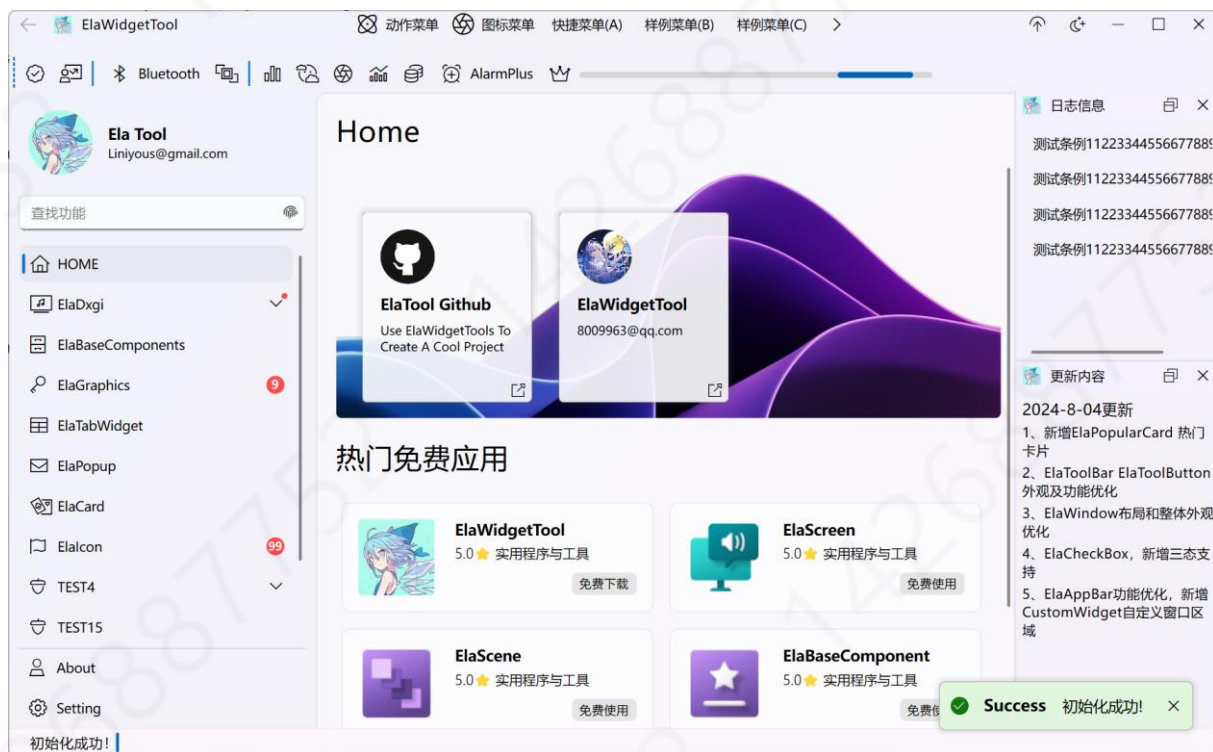


图 6 ElaWidgetToolsExample 整体一览

如图 6 所示，该界面为本库的例子程序界面，您可在此程序中体验大部分组件的外观和功能，同时可在源码中参考部分功能的使用与实现；详细的 API 文档参考本文档第五节。

### 5.3.2 有关的处理

待添加。

## 5.4 数据备份

本软件的有关信息存储不依赖于文件系统，无需进行数据备份；当您获得更新的版本时，只需替换前一版本的 DLL、LIB 和头文件即可使用。

## 5.5 消息

无。

## 6 附录

本章节为详细的 API 文档。

### 6.1 ElaAppBar

#### 6.1.1 组件说明

无边框标题栏组件，在任意 QWidget 或 QMainWindow 中使用，直接创建对象，将父对象设置为目标窗口即可，无需加入布局；需注意，该组件会改变目标窗口的 `contentsMargin-Top` 属性，值为该组件的高度。

#### 6.1.2 枚举定义

##### 6.1.2.1 ElaAppBarType::ButtonType

该枚举用于控制无边框组件的显示按钮状态；

```
enum ButtonType
{
    RouteBackButtonHint = 0x0001, // 路由跳转按钮显示
    NavigationButtonHint = 0x0002, // 导航栏展开按钮显示, 仅在导航栏为 Minimal
    状态时显示
    StayTopButtonHint = 0x0004,    // 置顶按钮显示
    ThemeChangeButtonHint = 0x0008, // 主题切换按钮按钮显示
    MinimizeButtonHint = 0x00010,  // 最小化按钮显示
    MaximizeButtonHint = 0x00020,  // 最大化按钮显示
    CloseButtonHint = 0x00040,     // 关闭按钮显示
};
```

##### 6.1.2.2 ElaAppBarType:: WMMouseActionType

该枚举用于向事件总线系统发送非客户区点击事件；

```
enum WMMouseActionType
{
    WMLBUTTONDOWN = 0x0001, // 非客户区左键抬起
    WMLBUTTONUP = 0x0002,   // 非客户区左键按下
    WMLBUTTONDBLCLK = 0x0004, // 非客户区左键双击
    WMNCLBUTTONDOWN = 0x0008, // 非客户区左键按下
```

```
};
```

### 6.1.3 属性成员

#### 6.1.3.1 IsStayTop

- a) 类型: `bool`
- b) 可用属性: `pIsStayTop`
- c) 关联信号: `pIsStayTopChanged`
- d) 默认值: `false`
- e) 描述: 该属性控制窗口是否进行置顶, `true` 为置顶, `false` 为不置顶;
- f) 对应的设置和获取函数:

```
void setIsStayTop (bool isStayTop);
```

```
bool getIsStayTop () const;
```

#### 6.1.3.2 IsFixedSize

- a) 类型: `bool`
- b) 可用属性: `pIsFixedSize`
- c) 关联信号: `pIsFixedSizeChanged`
- d) 默认值: `true`
- e) 描述: 该属性控制窗口是否可以在四边进行拉伸, `true` 为允许拉伸, `false` 为禁止拉伸;
- f) 对应的设置和获取函数:

```
void setIsFixedSize (bool isFixedSize);
```

```
bool getIsFixedSize () const;
```

#### 6.1.3.3 IsDefaultClosed

- a) 类型: `bool`
- b) 可用属性: `pIsDefaultClosed`
- c) 关联信号: `pIsDefaultClosedChanged`
- d) 默认值: `true`
- e) 描述: 该属性控制窗口是否以默认形式关闭, `true` 为以默认形式关闭, 若设置为 `false`, 点击关闭按钮或在任务栏关闭程序后, 关闭事件会被拦截, 同时发送 `closeButtonClicked` 信号, 用户可连接此信号进行处理;
- f) 对应的设置和获取函数:

```
void setIsDefaultClosed (bool isDefaultClosed);
```

```
bool getIsDefaultClosed () const;
```

#### 6.1.3.4 IsOnlyAllowMinAndClose

- a) 类型: `bool`
- b) 可用属性: `pIsOnlyAllowMinAndClose`
- c) 关联信号: `pIsOnlyAllowMinAndCloseChanged`
- d) 默认值: `false`
- e) 描述: 该属性限定窗口的可用行为, 设置为 `true` 时, 窗口仅对最小化和关闭按钮响应, 这个属性被用来实现主题切换等特殊功能;
- f) 对应的设置和获取函数:

```
void setIsOnlyAllowMinAndClose (bool isOnlyAllowMinAndClose);
```

```
bool getIsOnlyAllowMinAndClose () const;
```

#### 6.1.3.5 AppBarHeight

- a) 类型: `int`
- b) 可用属性: `pAppBarHeight`
- c) 关联信号: `pAppBarHeightChanged`
- d) 默认值: `45`
- e) 描述: 该属性设置无边框标题栏的固定高度, 此区域会被视作非客户区, 响应窗口拖动、双击放大、右键菜单等操作;
- f) 对应的设置和获取函数:

```
void setAppBarHeight (bool appBarHeight);
```

```
int getAppBarHeight () const;
```

#### 6.1.3.6 CustomWidget

- a) 类型: `QWidget*`
- b) 可用属性: `pCustomWidget`
- c) 关联信号: `pCustomWidgetChanged`
- d) 默认值: `nullptr`
- e) 描述: 该属性为标题栏中央的自定义窗口, 可设置为任意基于 `QWidget` 实现的窗口, 如 `MenuBar`、`TabBar` 等, 在进行设置后, 该自定义窗口的最大高度会被指定为 `AppBarHeight`, 最大宽度会被指定为 `CustomWidgetMaximumWidth`;
- f) 对应的设置和获取函数:

```
void setCustomWidget (QWidget* customWidget);
```

```
QWidget* getCustomWidget () const;
```

### 6.1.3.7 CustomWidgetMaximumWidth

- a) 类型: `int`
- b) 可用属性: `pCustomWidgetMaximumWidth`
- c) 关联信号: `pCustomWidgetMaximumWidthChanged`
- d) 默认值: `550`
- e) 描述: 该属性限制 CustomWidget 自定义窗口的最大宽度, 当 CustomWidget 被设置后, 该属性生效;
- f) 对应的设置和获取函数:

```
void setCustomWidgetMaximumWidth (int customWidgetMaximumWidth);
```

```
int getCustomWidgetMaximumWidth () const;
```

## 6.1.4 公有函数

### 6.1.4.1 setWindowButtonFlag

- a) 函数声明: `void setWindowButtonFlag(ElaAppBarType::ButtonType buttonFlag, bool isEnabled = true);`
- b) 参数 1: `ElaAppBarType::ButtonType buttonFlag` // 单个标题栏按钮对应的枚举
- c) 参数 2: `bool isEnabled` // 指定按钮是否可见
- d) 返回值: `void`
- e) 描述: 该函数控制标题栏的单个按钮显示状态, 对应的 `buttonFlag` 设置为 `true` 时, 按钮显示, 否则按钮隐藏;
- f) 使用示例: `setWindowButtonFlag(ElaAppBarType::MinimizeButtonHint, false);`  
// 将最小化按钮设置为隐藏;

### 6.1.4.2 setWindowButtonFlags

- a) 函数声明: `void setWindowButtonFlags(ElaAppBarType::ButtonFlags buttonFlags);`
- b) 参数 1: `ElaAppBarType::ButtonFlags buttonFlags` // 所有标题栏按钮对应的枚举
- c) 返回值: `void`
- d) 描述: 该函数控制标题栏的单个按钮显示状态, 对应的 `buttonFlag` 设置为 `true` 时, 按钮显示, 否则按钮隐藏;
- e) 使用示例: `setWindowButtonFlags(ElaAppBarType::MinimizeButtonHint | ElaAppBarType::MaximizeButtonHint);` // 将最小化和最大化按钮设置为显示, 其他按钮设置为隐藏;



#### 6.1.4.3 getWindowButtonFlags

- a) 函数声明: `ElaAppBarType::ButtonFlags getWindowButtonFlags() const;`
- b) 返回值: `ElaAppBarType::ButtonFlags` // 所有标题栏按钮对应的枚举
- c) 描述: 该函数返回当前可见的标题栏按钮的枚举集合;
- d) 使用示例: `ElaAppBarType::ButtonFlags flags = getWindowButtonFlags();`  
// 获取当前可见标题栏按钮的枚举集合;

#### 6.1.4.4 setRouteBackButtonEnable

- a) 函数声明: `void setRouteBackButtonEnable(bool isEnabled);`
- b) 参数 1: `bool isEnabled` // 路由跳转按钮是否使能
- c) 返回值: `void`
- d) 描述: 该函数控制标题栏的单个按钮显示状态, 对应的 `buttonFlag` 设置为 `true` 时, 按钮显示, 否则按钮隐藏;
- e) 使用示例: `setRouteBackButtonEnable (false);` // 禁用路由跳转按钮;

#### 6.1.4.5 closeWindow

- a) 函数声明: `void closeWindow();`
- b) 返回值: `void`
- c) 描述: 该函数开始一个渐变的缩小动画和透明度动画, 同时将 `ElaApplication` 类的属性 `IsApplicationClosed` 设置为 `true`, 并在动画结束后关闭窗口;
- d) 使用示例: `closeWindow();` // 关闭窗口, 并伴随渐变动画;

### 6.1.5 公有信号

#### 6.1.5.1 routeBackButtonClicked

- a) 信号声明: `Q_SIGNAL void routeBackButtonClicked();`
- b) 描述: 路由跳转按钮被点击时, 触发该信号;

#### 6.1.5.2 navigationButtonClicked

- a) 信号声明: `Q_SIGNAL void navigationButtonClicked();`
- b) 描述: 导航栏展开按钮被点击时, 触发该信号;

#### 6.1.5.3 themeChangeButtonClicked

- a) 信号声明: `Q_SIGNAL void themeChangeButtonClicked();`
- b) 描述: 主题切换按钮被点击时, 触发该信号;

#### 6.1.5.4 closeButtonClicked

- a) 信号声明: `Q_SIGNAL void closeButtonClicked();`
- b) 描述: 关闭按钮被点击, 且 `IsDefaultClosed` 属性为 `false` 时, 触发该信号;

### 6.2 ElaAcrylicUrlCard

#### 6.2.1 组件说明

带图片的亚克力卡片组件, 支持 URL 跳转。

#### 6.2.2 枚举定义

##### 6.2.2.1 ElaCardPixType::PixMode

该枚举用于控制自定义图片的显示模式;

```
enum PixMode
{
    Default = 0x0000, // 默认矩形显示
    RoundedRect = 0x0001, // 圆角矩形显示
    Ellipse = 0x0002, // 圆形显示
};
```

#### 6.2.3 属性成员

##### 6.2.3.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `5`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);
int getBorderRadius () const;
```

##### 6.2.3.2 MainOpacity

- a) 类型: `qreal`
- b) 可用属性: `pMainOpacity`
- c) 关联信号: `pMainOpacityChanged`



- d) 默认值: 0.95
- e) 描述: 该属性控制绘制背景时 QPainter 的透明度, 值越小越透明;
- f) 对应的设置和获取函数:

```
void setMainOpacity (qreal mainOpacity);  
qreal getMainOpacity () const;
```

#### 6.2.3.3 NoiseOpacity

- a) 类型: qreal
- b) 可用属性: pNoiseOpacity
- c) 关联信号: pNoiseOpacityChanged
- d) 默认值: 0.06
- e) 描述: 该属性控制绘制噪声干扰时 QPainter 的透明度, 值越大干扰越明显;
- f) 对应的设置和获取函数:

```
void setNoiseOpacity (qreal noiseOpacity);  
qreal getNoiseOpacity () const;
```

#### 6.2.3.4 BrushAlpha

- a) 类型: int
- b) 可用属性: pBrushAlpha
- c) 关联信号: pBrushAlphaChanged
- d) 默认值: 245
- e) 描述: 该属性控制绘制背景时 QColor 的 Alpha 值, 值越小越透明;
- f) 对应的设置和获取函数:

```
void setBrushAlpha (int brushAlpha);  
int getBrushAlpha () const;
```

#### 6.2.3.5 Title

- a) 类型: QString
- b) 可用属性: pTitle
- c) 关联信号: pTitleChanged
- d) 默认值: 空
- e) 描述: 该属性为主标题文字;
- f) 对应的设置和获取函数:

```
void setTitle (QString title);  
QString getTitle () const;
```

#### 6.2.3.6 SubTitle

- a) 类型: `QString`
- b) 可用属性: `pSubTitle`
- c) 关联信号: `pSubTitleChanged`
- d) 默认值: 空
- e) 描述: 该属性为副标题文字;
- f) 对应的设置和获取函数:

```
void setSubTitle (QString subTitle);  
QString getSubTitle () const;
```

#### 6.2.3.7 TitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pTitlePixelSize`
- c) 关联信号: `pTitlePixelSizeChanged`
- d) 默认值: 15
- e) 描述: 该属性为绘制主标题时, QPainter 的 `pixelSize` 大小;
- f) 对应的设置和获取函数:

```
void setTitlePixelSize (int titlePixelSize);  
int getTitlePixelSize () const;
```

#### 6.2.3.8 SubTitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pSubTitlePixelSize`
- c) 关联信号: `pSubTitlePixelSizeChanged`
- d) 默认值: 12
- e) 描述: 该属性为绘制副标题时, QPainter 的 `pixelSize` 大小;
- f) 对应的设置和获取函数:

```
void setSubTitlePixelSize (int subTitlePixelSize);  
int getSubTitlePixelSize () const;
```

#### 6.2.3.9 TitleSpacing

- a) 类型: `int`
- b) 可用属性: `pTitleSpacing`
- c) 关联信号: `pTitleSpacingChanged`

- d) 默认值: 10
- e) 描述: 该属性为自定义图片的底部到主标题的顶部的间隔;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (int titleSpacing);  
int getTitleSpacing () const;
```

#### 6.2.3.10 CardPixmap

- a) 类型: QPixmap
- b) 可用属性: pCardPixmap
- c) 关联信号: pCardPixmapChanged
- d) 默认值: 空
- e) 描述: 该属性为指定的自定义图片;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (QPixmap cardPixmap);  
QPixmap getCardPixmap () const;
```

#### 6.2.3.11 CardPixmapSize

- a) 类型: QSize
- b) 可用属性: pCardPixmapSize
- c) 关联信号: pCardPixmapSizeChanged
- d) 默认值: QSize(54 , 54)
- e) 描述: 该属性控制自定义图片的绘制尺寸;
- f) 对应的设置和获取函数:

```
void setCardPixmapSize (QSize cardPixmapSize);  
QSize getCardPixmapSize () const;
```

#### 6.2.3.12 CardPixmapBorderRadius

- a) 类型: int
- b) 可用属性: pCardPixmapBorderRadius
- c) 关联信号: pCardPixmapBorderRadiusChanged
- d) 默认值: 6
- e) 描述: 该属性控制自定义图片外围边框的圆角半径, 值越大圆角越明显; 仅在 pCardPixMode 属性为 ElaCardPixType::RoundedRect 时生效;
- f) 对应的设置和获取函数:

```
void setCardPixmapBorderRadius (int cardPixmapBorderRadius);
```

```
int getCardPixmapBorderRadius () const;
```

#### 6.2.3.13 CardPixMode

- a) 类型: `ElaCardPixType::PixMode`
- b) 可用属性: `pCardPixMode`
- c) 关联信号: `pCardPixModeChanged`
- d) 默认值: `ElaCardPixType::PixMode::Ellipse`
- e) 描述: 该属性控制自定义图片的显示模式;
- f) 对应的设置和获取函数:

```
void setCardPixMode (ElaCardPixType::PixMode cardPixMode);  
ElaCardPixType::PixMode getCardPixMode () const;
```

#### 6.2.3.14 Url

- a) 类型: `QString`
- b) 可用属性: `pUrl`
- c) 关联信号: `pUrlChanged`
- d) 默认值: 空
- e) 描述: 该属性为点击卡片时的跳转链接;
- f) 对应的设置和获取函数:

```
void setUrl (QString url);  
QString getUrl () const;
```

### 6.2.4 公有函数

#### 6.2.4.1 setCardPixmapSize

- a) 函数声明: `void setCardPixmapSize (int width, int height);`
- b) 参数 1: `int width` // 自定义图片宽度
- c) 参数 2: `int height` // 自定义图片高度
- d) 返回值: `void`
- e) 描述: 该函数控制自定义图片的绘制尺寸, 与直接设置 `pCardPixmapSize` 属性效果一致;
- f) 使用示例: `setCardPixmapSize(50 , 50);` // 设置图片尺寸为宽 50, 高 50;

## 6.3 ElaApplication

### 6.3.1 组件说明

单例类，项目的整体控制类，在此类中进行初始化操作；创建 `QApplication` 实例后，调用 `ElaApplication::init()` 函数进行初始化（`eApp->init()` 亦可），方可正常使用。

### 6.3.2 属性成员

#### 6.3.2.1 IsApplicationClosed

- a) 类型： `bool`
- b) 可用属性： `pIsApplicationClosed`
- c) 关联信号： `pIsApplicationClosedChanged`
- d) 默认值： `false`
- e) 描述：该属性提示程序是否已进入结束流程中，以默认形式关闭窗口或手动调用 `closeWinodw()` 函数时，该属性被置为 `true`；
- f) 对应的设置和获取函数：

```
void setIsApplicationClosed (bool isApplicationClosed);  
bool getIsApplicationClosed () const;
```

### 6.3.3 公有函数

#### 6.3.3.1 init

- a) 函数声明： `void init();`
- b) 返回值： `void`
- c) 描述：该函数应在创建 `QApplication` 实例后立即被调用，以进入组件库的初始化流程，包括添加图标库、设置基准字体等；
- d) 使用示例：

```
QApplication a(argc, argv);  
eApp->init();    // 组件库初始化;
```

#### 6.3.3.2 containsCursorToItem

- a) 函数声明： `static bool containsCursorToItem (QWidget* item);`
- b) 参数 1： `QWidget* item` // 被判定的 `QWidget` 对象指针；
- c) 返回值： `bool`
- d) 描述：该函数为静态工具函数，若 `item` 有效、可见，且位于鼠标指针下，返回 `true`，否则返回 `false`；

- e) 使用示例: `bool result = ElaApplication::containsCursorToItem(item);`  
`// item 窗口若可见且位于鼠标下, 返回 true;`

## 6.4 ElaBreadcrumbBar

### 6.4.1 组件说明

自带点击消除响应的面包屑导航组件, 点击非最右侧节点后, 自动消除点击节点右侧的所有节点。

### 6.4.2 公有函数

#### 6.4.2.1 setBreadcrumbList

- a) 函数声明: `void setBreadcrumbList(QStringList breadcrumbList);`
- b) 参数 1: `QStringList breadcrumbList` // 面包屑节点列表, 索引从 0 开始从左到右显示;
- c) 返回值: `void`
- d) 描述: 该函数设置面包屑组件的字符列表, 索引从 0 开始从左到右显示;
- e) 使用示例:  
`QStringList list;`  
`list.append("Item1");`  
`setBreadcrumbList(list); // 设置面包屑节点列表;`

#### 6.4.2.2 appendBreadcrumb

- a) 函数声明: `QStringList appendBreadcrumb(QString breadcrumb);`
- b) 参数 1: `QString breadcrumb` // 需要添加的节点;
- c) 返回值: `QStringList` // 添加节点后的面包屑节点列表;
- d) 描述: 该函数在面包屑组件最右侧新增一个字符块, 并返回新增块后的字符列表;
- e) 使用示例: `appendBreadcrumb ("Item1");` // 在最右侧新增一个 Item1 节点;

#### 6.4.2.3 removeBreadcrumb

- a) 函数声明: `QStringList removeBreadcrumb (QString breadcrumb);`
- b) 参数 1: `QString breadcrumb` // 需要移除的节点;
- c) 返回值: `QStringList` // 移除节点后的面包屑文字列表;
- d) 描述: 该函数在面包屑组件字符列表中从右向左匹配, 移除第一个匹配到的字

符块，并返回移除块后的字符列表；

- e) 使用示例：`removeBreadcrumb ("Item1");` // 移除靠右侧的第一个 Item1 节点；

#### 6.4.2.4 getBreadcrumbListCount

- a) 函数声明：`int getBreadcrumbListCount () const;`
- b) 返回值：`int` // 当前节点的数量；
- c) 描述：该函数返回面包屑组件当前存在的字符块的数量；
- d) 使用示例：`int count = getBreadcrumbListCount();` // 获取当前存在节点数量；

#### 6.4.2.5 getBreadcrumbList

- a) 函数声明：`QStringList getBreadcrumbList () const;`
- b) 返回值：`QStringList` // 当前节点列表；
- c) 描述：该函数返回面包屑组件当前的字符列表；
- d) 使用示例：`QStringList list = getBreadcrumbList ();` // 获取当前节点列表；

### 6.4.3 公有信号

#### 6.4.3.1 breadcrumbClicked

- a) 信号声明：`Q_SIGNAL void breadcrumbClicked (QString breadcrumb, QStringList lastBreadcrumbList);`
- b) 参数 1：`QString breadcrumb` // 被点击的非最右侧节点；
- c) 参数 2：`QStringList lastBreadcrumbList` // 被点击自动消除效果影响前的节点列表；
- d) 描述：非最右侧节点被点击时，触发该信号；

## 6.5 ElaCheckBox

### 6.5.1 组件说明

复选框组件，支持三态显示模式，调用 API 与 QT 原生一致。

## 6.6 ElaColorDialog

### 6.6.1 组件说明

支持拖选颜色和自定义颜色的颜色选择器组件。



## 6.6.2 属性成员

### 6.6.2.1 CurrentColor

- a) 类型: QColor
- b) 可用属性: pCurrentColor
- c) 关联信号: pCurrentColorChanged
- d) 默认值: QColor(0x80, 0xFF, 0xEF)
- e) 描述: 该属性为颜色选择器当前选中的颜色;
- f) 对应的设置和获取函数:

```
void setCurrentColor (QColor currentColor);  
QColor getCurrentColor () const;
```

## 6.6.3 公有函数

### 6.6.3.1 getCustomColorList

- a) 函数声明: QList<QColor>getCustomColorList () const
- b) 返回值: QList<QColor> // 自定义颜色集合, 固定 24 个, 若未设置则内容为 QColor()
- c) 描述: 该函数返回所有自定义颜色的集合;

### 6.6.3.2 getCustomColor

- a) 函数声明: QColor getCustomColor (int index) const
- d) 返回值: QColor // 指定索引处的自定义颜色, 若未设置则内容为 QColor()
- b) 描述: 该函数返回指定索引处的自定义颜色;

### 6.6.3.3 getCurrentColorRGB

- a) 函数声明: QString getCurrentColorRGB () const
- e) 返回值: QString // 当前选中颜色的十六进制代号, 以#AAAAAA 形式返回
- b) 描述: 该函数返回当前选中颜色的十六进制代号;

## 6.6.4 公有信号

### 6.6.4.1 colorSelected

- a) 信号声明: Q\_SIGNAL void colorSelected (const QColor& color);
- b) 参数 1: const QColor& color // 确认选中的颜色;
- c) 描述: 点击“确定”按钮后, 触发该信号;



## 6.7 ElaComboBox

### 6.7.1 组件说明

带展开和收起动画下拉框组件，并伴随展开指示器；调用 API 与 QT 原生一致。

### 6.7.2 属性成员

#### 6.7.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `3`
- e) 描述: 该属性控制组件外围边框的圆角半径，值越大圆角越明显；
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);
```

```
int getBorderRadius () const;
```

## 6.8 ElaContentDialog

### 6.8.1 组件说明

带原生阴影和背景遮罩的三态按钮对话框组件，实例化时指定父对象以指定作用窗口（不能为 NULL）。

### 6.8.2 公有函数

#### 6.8.2.1 setCentralWidget

- a) 函数声明: `void setCentralWidget(QWidget* centralWidget);`
- b) 参数 1: `QWidget* centralWidget` // 中心窗口;
- c) 返回值: `void`
- d) 描述: 该函数设置对话框的中心内容部件;
- e) 使用示例:

```
QWidget* centralWidget = new QWidget(this);  
setCentralWidget (centralWidget); // 设置中心窗口;
```

#### 6.8.2.2 setLeftButtonText

- a) 函数声明: `void setLeftButtonText(QString text);`

- b) 参数 1: `QString text` // 目标文字;
- c) 返回值: `void`
- d) 描述: 该函数设置对话框左侧按钮的文字;
- e) 使用示例: `setLeftButtonText("Left");` // 设置左侧按钮文字为 `Left`;

#### 6.8.2.3 setMiddleButtonText

- a) 函数声明: `void setMiddleButtonText(QString text);`
- b) 参数 1: `QString text` // 目标文字;
- c) 返回值: `void`
- d) 描述: 该函数设置对话框中间按钮的文字;
- e) 使用示例: `setMiddleButtonText("Middle");` // 设置中间按钮文字为 `Middle`;

#### 6.8.2.4 setRightButtonText

- a) 函数声明: `void setRightButtonText(QString text);`
- b) 参数 1: `QString text` // 目标文字;
- c) 返回值: `void`
- d) 描述: 该函数设置对话框中间按钮的文字;
- e) 使用示例: `setRightButtonText("Right");` // 设置中间按钮文字为 `Right`;

#### 6.8.2.5 onLeftButtonClicked

- a) 函数声明: `Q_SLOT virtual void onLeftButtonClicked();`
- b) 描述: 虚槽函数, 在对话框左侧按钮被点击后, 先发出左侧按钮点击信号, 随后调用该虚槽函数, 执行完毕后, `close` 对话框;

#### 6.8.2.6 onMiddleButtonClicked

- a) 函数声明: `Q_SLOT virtual void onMiddleButtonClicked();`
- b) 描述: 虚槽函数, 在对话框中间按钮被点击后, 先发出中间按钮点击信号, 随后调用该虚槽函数, 执行完毕后, `close` 对话框;

#### 6.8.2.7 onRightButtonClicked

- a) 函数声明: `Q_SLOT virtual void onRightButtonClicked();`
- b) 描述: 虚槽函数, 在对话框右侧按钮被点击后, 先发出右侧按钮点击信号, 随后调用该虚槽函数, 执行完毕后, `close` 对话框;

## 6.8.3 公有信号

### 6.8.3.1 leftButtonClicked

- a) 信号声明: `Q_SIGNAL void leftButtonClicked ();`
- b) 描述: 对话框左侧按钮被点击后, 触发该信号;

### 6.8.3.2 middleButtonClicked

- a) 信号声明: `Q_SIGNAL void middleButtonClicked ();`
- b) 描述: 对话框左侧按钮被点击后, 触发该信号;

### 6.8.3.3 rightButtonClicked

- a) 信号声明: `Q_SIGNAL void rightButtonClicked ();`
- b) 描述: 对话框左侧按钮被点击后, 触发该信号;

## 6.9 ElaDockWidget

### 6.9.1 组件说明

带自定义标题栏和浮动阴影的停靠窗口, 调用 API 与 QT 原生一致。

## 6.10 ElaDockWidget

### 6.10.1 组件说明

带自定义标题栏和浮动阴影的停靠窗口, 调用 API 与 QT 原生一致。

## 6.11 ElaDoubleSpinBox

### 6.11.1 组件说明

浮点数值微调框, 调用 API 与 QT 原生一致。

## 6.12 ElaDxgiManager

### 6.12.1 组件说明

单例类, Windows 独有的 DXGI 高性能屏幕采集组件, 支持指定采集帧率、采集设备、输出对象。

## 6.12.2 公有函数

### 6.12.2.1 getDxDeviceList

- a) 函数声明: `QStringList getDxDeviceList() const;`
- b) 返回值: `QStringList` // 当前设备的 DXGI 采集设备列表;
- c) 描述: 该函数返回当前设备存在的 DXGI 采集设备列表;
- d) 使用示例: `QStringList list = ElaDxgiManager::getDxDeviceList();` // 获取当前设备 DXGI 设备列表;

### 6.12.2.2 getOutputDeviceList

- a) 函数声明: `QStringList getOutputDeviceList() const;`
- b) 返回值: `QStringList` // 当前设备的输出设备列表;
- c) 描述: 该函数返回当前设备存在的输出设备（屏幕）列表;
- d) 使用示例: `QStringList list = ElaDxgiManager::getOutputDeviceList();` // 获取当前设备的输出设备列表;

### 6.12.2.3 grabScreenToImage

- a) 函数声明: `QImage grabScreenToImage() const;`
- b) 返回值: `QImage`
- c) 描述: 该函数返回 `QImage` 格式的最新捕获的图像, 若未开始采集, 返回 `QImage()`;
- d) 使用示例: `QImage img = ElaDxgiManager::grabScreenToImage();` // 获取最新采集的 `QImage`;

### 6.12.2.4 startGrabScreen

- a) 函数声明: `void startGrabScreen() const;`
- b) 返回值: `void`
- c) 描述: 以选中的采集设备开始采集选择的屏幕;

### 6.12.2.5 stopGrabScreen

- a) 函数声明: `void stopGrabScreen() const;`
- b) 返回值: `void`
- c) 描述: 停止采集屏幕;

#### 6.12.2.6 setDxDeviceID

- a) 函数声明: `bool setDxDeviceID(int dxID);`
- b) 参数 1: `int dxID` // 当前设备的 DXGI 采集设备索引, 从 0 开始;
- c) 返回值: `bool` // 设置是否成功;
- d) 描述: 该函数设置当前设备的 DXGI 采集设备索引, 如果设置成功, 返回 `true`;
- e) 使用示例: `bool result = ElaDxgiManager::setDxDeviceID (0);` //设置 DXGI 采集设备索引为 0;

#### 6.12.2.7 getDxDeviceID

- a) 函数声明: `int getDxDeviceID() const;`
- b) 返回值: `int` // 当前设置的 DXGI 采集设备索引;
- c) 描述: 该函数获取当前使用的 DXGI 采集设备索引;

#### 6.12.2.8 setOutputDeviceID

- a) 函数声明: `bool setOutputDeviceID(int dxID);`
- b) 参数 1: `int dxID` // 当前设备的 DXGI 采集设备索引, 从 0 开始;
- c) 返回值: `bool` // 设置是否成功;
- d) 描述: 该函数设置当前设备的 DXGI 采集设备索引, 如果设置成功, 返回 `true`;
- e) 使用示例: `bool result = ElaDxgiManager::setOutputDeviceID (0);` //设置 DXGI 输出设备索引为 0;

#### 6.12.2.9 getOutputDeviceID

- a) 函数声明: `int getOutputDeviceID() const;`
- b) 返回值: `int` // 当前设置的输出设备索引;
- c) 描述: 该函数获取当前使用的屏幕输出设备索引;

#### 6.12.2.10 setGrabArea

- a) 函数声明: `void setGrabArea (int width, int height);`
- b) 参数 1: `int width` // 采集区域宽度;
- c) 参数 2: `int height` // 采集区域高度;
- d) 返回值: `void`
- e) 描述: 该函数设置 DXGI 采集区域, 采集区域的中心为屏幕中心;
- f) 使用示例: `ElaDxgiManager::setGrabArea (320, 320);` //设置采集范围为 320, 320, 采集中心为屏幕中心;

#### 6.12.2.11 setGrabArea

- a) 函数声明: `void setGrabArea (int x, int y, int width, int height);`
- b) 参数 1: `int x` // 采集区域左上角横坐标;
- c) 参数 2: `int y` // 采集区域左上角纵坐标;
- d) 参数 3: `int width` // 采集区域宽度;
- e) 参数 4: `int height` // 采集区域高度;
- f) 返回值: `void`
- g) 描述: 该函数设置 DXGI 采集区域, 采集区域由用户指定;
- h) 使用示例: `ElaDxgiManager::setGrabArea (0, 0, 320, 320);` //设置采集范围为 320, 320, 采集中心为 160, 160;

#### 6.12.2.12 getGrabArea

- a) 函数声明: `QRect getGrabArea () const;`
- b) 返回值: `QRect` // 当前采集区域;
- c) 描述: 该函数获取当前 DXGI 采集区域;
- d) 使用示例: `QRect rect = ElaDxgiManager::getGrabArea ();` //获取采集范围;

#### 6.12.2.13 setGrabFrameRate

- a) 函数声明: `void setGrabFrameRate (int frameRateValue);`
- b) 参数 1: `int frameRateValue` // 采集帧率;
- c) 返回值: `void`
- d) 描述: 该函数设置 DXGI 采集帧率, 帧率上限由用户设备决定;
- e) 使用示例: `ElaDxgiManager::setGrabFrameRate(120);` //设置采集帧率为 120;

#### 6.12.2.14 getGrabFrameRate

- a) 函数声明: `int getGrabFrameRate () const;`
- b) 返回值: `int`
- c) 描述: 该函数获取当前 DXGI 采集帧率;
- d) 使用示例: `int rate = ElaDxgiManager::getGrabFrameRate();` //获取采集帧率;

#### 6.12.2.15 setTimeoutMsValue

- a) 函数声明: `void setTimeoutMsValue (int timeoutValue);`
- b) 参数 1: `int timeoutValue` // 超时等待时间, 单位为 ms;
- c) 返回值: `void`

- d) 描述：该函数设置 DXGI 采集超时等待时间，该值不会显著影响采集效果；
- e) 使用示例：`ElaDxgiManager::setTimeoutMsValue(120);` //设置超时等待为 120 ms;

#### 6.12.2.16 getTimeoutMsValue

- a) 函数声明：`int getTimeoutMsValue () const;`
- b) 返回值：`int` // 超时等待时间，单位为 ms;
- c) 描述：该函数获取 DXGI 采集超时等待时间；
- d) 使用示例：`int timeoutValue = ElaDxgiManager::getTimeoutMsValue();` //获取超时等待时间；

### 6.12.3 公有信号

#### 6.12.3.1 leftButtonClicked

- a) 信号声明：`Q_SIGNAL void grabImageUpdate ();`
- b) 描述：成功采集到一帧新图片时，触发该信号；

## 6.13 ElaDxgiScreen

### 6.13.1 组件说明

用于快捷预览 DXGI 采集结果的屏幕组件。

### 6.13.2 属性成员

#### 6.13.2.1 BorderRadius

- a) 类型：`int`
- b) 可用属性：`pBorderRadius`
- c) 关联信号：`pBorderRadiusChanged`
- d) 默认值：`3`
- e) 描述：该属性控制组件外围边框的圆角半径，值越大圆角越明显；
- f) 对应的设置和获取函数：  
`void setBorderRadius (int borderRadius);`  
`int getBorderRadius () const;`



### 6.13.3 公有函数

#### 6.13.3.1 setIsSyncGrabSize

- a) 函数声明: `void setIsSyncGrabSize(bool isSyncGrabSize);`
- b) 参数 1: `bool isSyncGrabSize` //是否同步采集区域大小;
- c) 返回值: `void`
- d) 描述: 该函数控制组件的固定大小, 设置为 `true` 时, 组件的 `FixedSize` 会被设置为与采集区域大小一致;

#### 6.13.3.2 getIsSyncGrabSize

- a) 函数声明: `bool getIsSyncGrabSize() const;`
- b) 返回值: `bool`
- c) 描述: 该函数获取是否同步采集区域大小;

### 6.14 ElaEvent

#### 6.14.1 组件说明

用于注册事件总线系统的事件类。

#### 6.14.2 属性成员

##### 6.14.2.1 EventName

- a) 类型: `QString`
- b) 可用属性: `pEventName`
- c) 关联信号: `pEventNameChanged`
- d) 默认值: 空
- e) 描述: 该属性为事件的类型名称, 由用户自行指定;
- f) 对应的设置和获取函数:

```
void setEventName (QString eventName);
```

```
QString getEventName () const;
```

##### 6.14.2.2 FunctionName

- a) 类型: `QString`
- b) 可用属性: `pFunctionName`
- c) 关联信号: `pFunctionNameChanged`
- d) 默认值: 空



- e) 描述：该属性为事件的处理函数名称，由用户自行指定函数名，需注意，该函数必须为父对象持有的函数，参数为固定的 `QVariantMap` 类型且必须附带 `Q_INVOKABLE` 属性；

- f) 对应的设置和获取函数：

```
void setFunctionName (QString functionName);  
QString getFunctionName () const;
```

#### 6.14.2.3 ConnectionType

- a) 类型： `QString`
- b) 可用属性： `pConnectionType`
- c) 关联信号： `pConnectionTypeChanged`
- d) 默认值： `Qt::AutoConnection`
- e) 描述：该属性为事件的处理函数调用类型，由用户自行指定；
- f) 对应的设置和获取函数：

```
void setConnectionType (Qt::ConnectionType connectionType);  
Qt::ConnectionType getConnectionType () const;
```

### 6.14.3 公有函数

#### 6.14.3.1 registerAndInit

- a) 函数声明： `ElaEventBusType::EventBusReturnType registerAndInit();`
- b) 返回值： `ElaEventBusType::EventBusReturnType` // 注册结果
- c) 描述：该函数进行事件的初始化和注册操作，调用此函数后，当事件总线 post 已注册的事件类型时，自动调用该事件指定的函数；
- d) 使用示例：

```
ElaEvent* focusEvent = new ElaEvent("WMWindowClicked", "onWMWindowClickedEvent", this); // 创建一个事件，事件类型为 WMWindowClicked，处理该事件的函数名称为 onWMWindowClickedEvent，且该函数参数为 QVariantMap 类型且必须附带 Q_INVOKABLE 属性；  
focusEvent->registerAndInit(); // 注册并初始化该事件；  
Q_INVOKABLE void onWMWindowClickedEvent(QVariantMap data); // 事件的处理函数；
```

### 6.15 ElaEventBus

#### 6.15.1 组件说明

单例类，统一管理注册事件的发布和订阅行为。

## 6.15.2 枚举定义

### 6.15.2.1 ElaEventBusType::EventBusReturnType

该枚举用于控制无边框组件的显示按钮状态；

```
enum EventBusReturnType
{
    Success = 0x0000, // 操作成功;
    EventInvalid = 0x0001, // 事件不可用或重复注册;
    EventNameInvalid = 0x0002, // 事件名称为空;
};
```

## 6.15.3 公有函数

### 6.15.3.1 post

- a) 函数声明: `ElaEventBusType::EventBusReturnType post(const QString& eventName, const QVariantMap& data = {});`
- b) 参数 1: `const QString& eventName` // 发布的事件名称
- c) 参数 2: `const QVariantMap& data` // 发布的事件内容
- d) 返回值: `ElaEventBusType::EventBusReturnType` // 发布结果
- e) 描述: 该函数发布一个指定类型的事件, 对应类型的已注册事件会自行调用注册的事件处理函数, 调用方式由事件的 `pConnectionType` 属性决定, 事件内容为 `QVariantMap` 类型, 由用户指定;
- f) 使用示例:  

```
QVariantMap postData; // 创建事件内容
postData.insert("WMClickType", ElaAppBarType::WMLBUTTONDBLCLK);
ElaEventBus::getInstance()->post("WMWindowClicked", postData); // 发布事件
```

### 6.15.3.2 getRegisteredEventsName

- a) 函数声明: `QStringList getRegisteredEventsName () const;`
- b) 返回值: `QStringList` // 当前已注册的所有事件类型;
- c) 描述: 该函数返回当前已注册的所有事件类型;
- d) 使用示例: `QStringList list = ElaEventBus::getRegisteredEventsName ();` // 获取当前已注册的所有事件类型;

## 6.16 ElaExponentialBlur

### 6.16.1 组件说明

单例类，提供图片的指数模糊功能。

### 6.16.2 公有函数

#### 6.16.2.1 doExponentialBlur

- a) 函数声明: `static QPixmap doExponentialBlur(QImage img, const quint16& blurRadius);`
- b) 参数 1: `QImage img` // 需要进行指数模糊的图片;
- c) 参数 2: `const quint16& blurRadius` // 指数模糊级别, 数值越大模糊效果越高;
- d) 返回值: `QPixmap` // 指数模糊后的图片;
- e) 描述: 该函数对一张图片进行指数模糊, 并返回模糊后的图片;
- f) 使用示例: `QPixmap pix = ElaExponentialBlur::doExponentialBlur(QImage(":/Resource/Image/Cirno.jpg"), 1.5);` // 指数模糊指定图片, 模糊指数为 1.5;

## 6.17 ElaFlowLayout

### 6.17.1 组件说明

带动画的流式布局, 调用 API 与 QT 原生一致。

### 6.17.2 公有函数

#### 6.17.2.1 setIsAnimation

- a) 函数声明: `void setIsAnimation(bool isAnimation);`
- b) 参数 1: `bool isAnimation` // 是否启用动画效果;
- c) 返回值: `void`
- d) 描述: 该函数对流式布局启用或关闭动画效果, 设定为 `true` 时, 启用动画效果;

## 6.18 ElaGraphicsItem

### 6.18.1 组件说明

升级版图元组件, 支持序列化和反序列化、图元连接等进阶操作。

## 6.18.2 属性成员

### 6.18.2.1 Width

- a) 类型: `int`
- b) 可用属性: `pWidth`
- c) 关联信号: `pWidthChanged`
- d) 默认值: `50`
- e) 描述: 该属性为图元的宽度;
- f) 对应的设置和获取函数:

```
void setWidth (int width);  
int getWidth () const;
```

### 6.18.2.2 Height

- a) 类型: `int`
- b) 可用属性: `pHeight`
- c) 关联信号: `pHeightChanged`
- d) 默认值: `50`
- e) 描述: 该属性为图元的高度;
- f) 对应的设置和获取函数:

```
void setHeight (int height);  
int getHeight () const;
```

### 6.18.2.3 ItemImage

- a) 类型: `QImage`
- b) 可用属性: `pItemImage`
- c) 关联信号: `pItemImageChanged`
- d) 默认值: 空
- e) 描述: 该属性为图元的默认显示图片;
- f) 对应的设置和获取函数:

```
void setItemImage (QImage itemImage);  
QImage getItemImage () const;
```

### 6.18.2.4 ItemSelectedImage

- a) 类型: `QImage`
- b) 可用属性: `pItemSelectedImage`

- c) 关联信号: `pItemSelectedImageChanged`
- d) 默认值: 空
- e) 描述: 该属性为图元被选中时的显示图片;
- f) 对应的设置和获取函数:

```
void setItemSelectedImage (QImage itemSelectedImage);  
QImage getItemSelectedImage () const;
```

#### 6.18.2.5 ItemImage

- a) 类型: `QString`
- b) 可用属性: `pItemName`
- c) 关联信号: `pItemNameChanged`
- d) 默认值: 空
- e) 描述: 该属性为图元的名称;
- f) 对应的设置和获取函数:

```
void setItemName (QString itemName);  
QString getItemName () const;
```

#### 6.18.2.6 DataRoutes

- a) 类型: `QVariantMap`
- b) 可用属性: `pDataRoutes`
- c) 关联信号: `pDataRoutesChanged`
- d) 默认值: 空
- e) 描述: 该属性为图元的内置数据, 由用户自行指定;
- f) 对应的设置和获取函数:

```
void setDataRoutes (QVariantMap dataRoutes);  
QVariantMap getDataRoutes () const;
```

#### 6.18.2.7 MaxLinkPortCount

- a) 类型: `int`
- b) 可用属性: `pMaxLinkPortCount`
- c) 关联信号: `pMaxLinkPortCountChanged`
- d) 默认值: 1
- e) 描述: 该属性为图元的最大可接受连接数;
- f) 对应的设置和获取函数:

```
void setMaxLinkPortCount (int maxLinkPortCount);
```

```
int getMaxLinkPortCount () const;
```

### 6.18.3 公有函数

#### 6.18.3.1 getItemUID

- a) 函数声明: `QString getItemUID () const;`
- b) 返回值: `QString` // 图元独有的识别字符串;
- c) 描述: 该函数返回每个图元独有的识别字符串, 由组件自动创建;

#### 6.18.3.2 setLinkPortState

- a) 函数声明: `void setLinkPortState(bool isFullLink);`
- b) 参数 1: `bool isFullLink` // 是否标准化连接状态;
- c) 返回值: `void`
- d) 描述: 该函数设置图元的连接状态, 设置为 `true` 时, 图元的所有连接端口被设置为已使用状态, 否则设置为未使用状态;

#### 6.18.3.3 setLinkPortState

- a) 函数声明: `void setLinkPortState(bool isLink, int portIndex);`
- b) 参数 1: `bool isLink` // 连接状态;
- c) 参数 2: `int portIndex` // 设置连接状态的端口号, 从 0 开始;
- d) 返回值: `void`
- e) 描述: 该函数设置图元指定端口的连接状态, 设置为 `true` 时, 图元的指定端口被设置为已使用状态, 否则设置为未使用状态;

#### 6.18.3.4 getLinkPortState

- a) 函数声明: `bool getLinkPortState(int portIndex);`
- b) 参数 1: `int portIndex` // 需要获取连接状态的端口号, 从 0 开始;
- c) 返回值: `bool` // 端口是否已使用;
- d) 描述: 该函数返回图元指定端口的使用状态;

#### 6.18.3.5 getLinkPortState

- a) 函数声明: `QVector<bool>getLinkPortState() const;`
- b) 返回值: `QVector<bool>` // 端口使用状态;
- c) 描述: 该函数返回图元的端口使用状态, `QVector` 对应的索引即为对应端口, 从 0 开始;

#### 6.18.3.6 getUsedLinkPortCount

- a) 函数声明: `int getUsedLinkPortCount() const;`
- b) 返回值: `int` // 已使用的端口数量;
- c) 描述: 该函数返回图元已使用端口的数量;

#### 6.18.3.7 getUsedLinkPortCount

- a) 函数声明: `QVector<int> getUsedLinkPortCount() const;`
- b) 返回值: `QVector<int>` // 已使用的端口索引集合;
- c) 描述: 该函数返回图元已使用端口的索引列表, `QVector` 的值即为已使用的端口索引;

#### 6.18.3.8 getUnusedLinkPortCount

- a) 函数声明: `int getUnusedLinkPortCount() const;`
- b) 返回值: `int` // 未使用的端口数量;
- c) 描述: 该函数返回图元未使用端口的数量;

#### 6.18.3.9 getUnusedLinkPortCount

- a) 函数声明: `QVector<int> getUnusedLinkPortCount() const;`
- b) 返回值: `QVector<int>` // 未使用的端口索引集合;
- c) 描述: 该函数返回图元未使用端口的索引列表, `QVector` 的值即为已使用的端口索引;

### 6.19 ElaGraphicsLineItem

#### 6.19.1 组件说明

升级版线图元组件, 支持两种连接模式, `item` 间连接和指定点连接, 需注意, 连接类型由构造函数指定, 创建后不可更改。

#### 6.19.2 属性成员

##### 6.19.2.1 StartPoint

- a) 类型: `QPointF`
- b) 默认值: `空`
- c) 描述: 该属性为点连接模式下其中一点的坐标;
- d) 对应的设置和获取函数:



```
void setStartPoint (QPointF startPoint);  
QPointF getStartPoint () const;
```

#### 6.19.2.2 EndPoint

- a) 类型: `QPointF`
- b) 默认值: 空
- c) 描述: 该属性为点连接模式下其中一点的坐标;
- d) 对应的设置和获取函数:

```
void setEndPoint (QPointF startPoint);  
QPointF getEndPoint () const;
```

#### 6.19.2.3 StartItem

- a) 类型: `ElaGraphicsItem*`
- b) 默认值: 空
- c) 描述: 该属性为图元连接模式下其中一个图元的坐标;
- d) 对应的设置和获取函数:

```
void setStartItem (ElaGraphicsItem* startItem);  
ElaGraphicsItem* getStartItem () const;
```

#### 6.19.2.4 EndItem

- a) 类型: `ElaGraphicsItem*`
- b) 默认值: 空
- c) 描述: 该属性为图元连接模式下其中一个图元的坐标;
- d) 对应的设置和获取函数:

```
void setEndItem (ElaGraphicsItem* endItem);  
ElaGraphicsItem* getEndItem () const;
```

#### 6.19.2.5 StartItemPort

- a) 类型: `int`
- b) 默认值: 0
- c) 描述: 该属性为图元连接模式下其中一个图元的连接端口;
- d) 对应的设置和获取函数:

```
void setStartItemPort (int startItemPort);  
int getStartItemPort () const;
```



### 6.19.2.6 EndItemPort

- a) 类型: `int`
- b) 默认值: `0`
- c) 描述: 该属性为图元连接模式下其中一个图元的连接端口;
- d) 对应的设置和获取函数:

```
void setEndItemPort (int endItemPort);
```

```
int getEndItemPort () const;
```

## 6.19.3 公有函数

### 6.19.3.1 isTargetLink

- a) 函数声明: `bool isTargetLink(ElaGraphicsItem* item) const;`
- b) 参数 1: `ElaGraphicsItem* item` // 被判断图元;
- c) 返回值: `bool` // 是否为 `StartItem` 或 `EndItem` 之一
- d) 描述: 该函数返回指定图元是否为线图元的两个端点之一;

### 6.19.3.2 isTargetLink

- a) 函数声明: `bool isTargetLink(ElaGraphicsItem* item1, ElaGraphicsItem* item2) const;`
- b) 参数 1: `ElaGraphicsItem* item1` // 被判断图元 1;
- c) 参数 2: `ElaGraphicsItem* item2` // 被判断图元 2;
- d) 返回值: `bool` // 是否为 `StartItem` 和 `EndItem`
- e) 描述: 该函数返回指定的两个图元是否为线图元的两个端点;

### 6.19.3.3 isTargetLink

- a) 函数声明: `bool isTargetLink(ElaGraphicsItem* item1, ElaGraphicsItem* item2, int port1, int port2) const;`
- b) 参数 1: `ElaGraphicsItem* item1` // 被判断图元 1;
- c) 参数 2: `ElaGraphicsItem* item2` // 被判断图元 2;
- d) 参数 3: `int port1` // 图元 1 端口;
- e) 参数 4: `int port2` // 图元 2 端口;
- f) 返回值: `bool` // 是否为 `StartItem` 和 `EndItem`, 且端口与构造时一致
- g) 描述: 该函数返回指定的两个图元是否为线图元的两个端点, 且端口与构造函数指定一致;

## 6.20 ElaGraphicsScene

### 6.20.1 组件说明

升级版场景组件，支持图元连接、序列化与反序列化、以及多种常用图形视图框架功能。

### 6.20.2 枚举定义

#### 6.20.2.1 ElaGraphicsSceneType:: SceneMode

该枚举用于控制场景的操作模式；

```
enum SceneMode
{
    Default= 0x0000,    // 默认状态；
    DragMove= 0x0001, // 拖动模式，按住 ALT 生效；
    MultiSelect= 0x0002, // 多选模式，按住 CTRL 生效；
    ItemLink= 0x0003, // 连接模式，按住 Shift 生效；
};
```

### 6.20.3 属性成员

#### 6.20.3.1 IsCheckLinkPort

- a) 类型: `bool`
- b) 可用属性: `pIsCheckLinkPort`
- c) 关联信号: `pIsCheckLinkPortChanged`
- d) 默认值: `false`
- e) 描述: 该属性控制是否在图元连接时检测图元的端口是否可用；
- f) 对应的设置和获取函数:

```
void setIsCheckLinkPort (bool isCheckLinkPort);
bool getIsCheckLinkPort () const;
```

#### 6.20.3.2 SerializePath

- a) 类型: `QString`
- b) 可用属性: `pSerializePath`
- c) 关联信号: `pSerializePathChanged`
- d) 默认值: `"/scene.bin"` // 序列化文件的默认存储地址和文件格式（限定为.bin 格式）；

- e) 描述：该属性为序列化操作保存和读取文件的路径；
- f) 对应的设置和获取函数：

```
void setSerializePath (QString serializePath);  
QString getSerializePath () const;
```

## 6.20.4 公有函数

### 6.20.4.1 addItem

- a) 函数声明： `void addItem(ElaGraphicsItem* item);`
- b) 参数 1: `ElaGraphicsItem* item` // 需要添加到场景的图元；
- c) 返回值: `void`
- d) 描述：该函数将指定图元添加到场景中，若图元的 `pItemName` 属性为空，则为图元设置默认名称；

### 6.20.4.2 removeItem

- a) 函数声明： `void removeItem(ElaGraphicsItem* item);`
- b) 参数 1: `ElaGraphicsItem* item` // 需要从场景移除的图元；
- c) 返回值: `void`
- d) 描述：该函数将指定图元从场景中移除，同时移除所有与该图元有关的连接信息；

### 6.20.4.3 removeSelectedItems

- a) 函数声明： `void removeSelectedItems();`
- b) 返回值: `void`
- c) 描述：该函数将所有处于选中状态图元从场景中移除，同时移除所有与这些图元有关的连接信息；

### 6.20.4.4 clear

- a) 函数声明： `void clear();`
- b) 返回值: `void`
- c) 描述：该函数将所有图元从场景中移除，同时移除所有与这些图元有关的连接信息；

### 6.20.4.5 createAndAddItem

- a) 函数声明： `QList<ElaGraphicsItem*> createAndAddItem(int width, int height, in`

- `t count = 1);`
- b) 参数 1: `int width` // 图元宽度;
- c) 参数 2: `int height` // 图元高度;
- d) 参数 3: `int count` // 图元数量;
- e) 返回值: `QList<ElaGraphicsItem*>` // 创建的图元列表;
- f) 描述: 该函数以指定宽高批量创建图元, 创建数量由 `count` 指定, 并自动为创建的图元调用 `addItem` 函数, 并返回创建的图元列表;

#### 6.20.4.6 getSelectedElaItems

- a) 函数声明: `QList<ElaGraphicsItem*> getSelectedElaItems() const;`
- b) 返回值: `QList<ElaGraphicsItem*>` // 获取的图元列表;
- c) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 且处于选中状态的图元;

#### 6.20.4.7 getElaItems

- a) 函数声明: `QList<ElaGraphicsItem*> getElaItems ();`
- b) 返回值: `QList<ElaGraphicsItem*>` // 获取的图元列表;
- c) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 的图元;

#### 6.20.4.8 getElaItems

- a) 函数声明: `QList<ElaGraphicsItem*> getElaItems (QPoint pos);`
- b) 参数 1: `QPoint pos` // 指定坐标;
- c) 返回值: `QList<ElaGraphicsItem*>` // 获取的图元列表;
- d) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 且位于指定坐标的图元;

#### 6.20.4.9 getElaItems

- a) 函数声明: `QList<ElaGraphicsItem*> getElaItems (QPointF pos);`
- b) 参数 1: `QPointF pos` // 指定坐标;
- c) 返回值: `QList<ElaGraphicsItem*>` // 获取的图元列表;
- d) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 且位于指定坐标的图元;

#### 6.20.4.10 getElaItems

- a) 函数声明: `QList<ElaGraphicsItem*> getElaItems (QRect rect);`
- b) 参数 1: `QRect rect` // 指定区域;
- c) 返回值: `QList<ElaGraphicsItem*>` // 获取的图元列表;
- d) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 且位于指定区域的图元;

## 6.20.4.11 getElaItems

- a) 函数声明: `QList<ElaGraphicsItem*> getElaItems (QRectF rect);`
- b) 参数 1: `QRectF rect` // 指定区域;
- c) 返回值: `QList<ElaGraphicsItem*>` // 获取的图元列表;
- d) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 且位于指定区域的图元;

## 6.20.4.12 setSceneMode

- a) 函数声明: `void setSceneMode(ElaGraphicsSceneType::SceneMode mode);`
- b) 参数 1: `ElaGraphicsSceneType::SceneMode mode` // 场景模式;
- c) 返回值: `void`
- d) 描述: 该函数将场景设置为指定的操作模式;

## 6.20.4.13 getSceneMode

- a) 函数声明: `ElaGraphicsSceneType::SceneMode getSceneMode() const;`
- b) 返回值: `ElaGraphicsSceneType::SceneMode` // 场景模式;
- c) 描述: 该函数返回当前场景模式;

## 6.20.4.14 selectAllItems

- a) 函数声明: `void selectAllItems();`
- b) 返回值: `void`
- c) 描述: 该函数选中所有基类为 `ElaGraphicsItem` 的图元;

## 6.20.4.15 getItemLinkList

- a) 函数声明: `QList<QVariantMap> getItemLinkList() const`
- b) 返回值: `QList <QVariantMap>` // 图元的连接状态列表;
- c) 描述: 该函数返回所有基类为 `ElaGraphicsItem` 的图元的连接数据, `QList` 中的每一个成员为一条连接数据, `QVariantMap` 中的第一条键值对为图元 1 的 Item UID 和连接端口号, 第二条键值对为图元 2 的 ItemUID 和连接端口号;
- d) 使用示例: // 略去判空流程 仅做使用展示;  

```
QList <QVariantMap> itemLinkDataList = getItemLinkList();
QVariantMap oneLinkData = itemLinkDataList[0];
QList<QString> uidList = oneLinkData.keys();
QList<QVariant> portList = oneLinkData.values();
QString item1UID = uidList[0]; // 图元 1 的 ItemUID;
```

```
int item1Port = portList[1]; // 图元 1 的连接端口;
```

#### 6.20.4.16 addItemLink

- a) 函数声明: `bool addItemLink(ElaGraphicsItem* item1, ElaGraphicsItem* item2, int port1 = 0, int port2 = 0);`
- b) 参数 1: `ElaGraphicsItem* item1` // 图元 1;
- c) 参数 2: `ElaGraphicsItem* item2` // 图元 2;
- d) 参数 3: `int port1` // 图元 1 的端口;
- e) 参数 4: `int port2` // 图元 2 的端口;
- f) 返回值: `bool` // 连接是否成功;
- g) 描述: 该函数对指定的两个图元进行连接, 如果场景开启了 `pIsCheckLinkPort` 属性, 则执行连接可用性检查, 若检查通过, 新增一条连接数据; 返回值为连接结果;

#### 6.20.4.17 removeItemLink

- a) 函数声明: `bool removeItemLink(ElaGraphicsItem* item1);`
- b) 参数 1: `ElaGraphicsItem* item1` // 需要移除连接信息的图元;
- c) 返回值: `bool` // 移除是否成功;
- d) 描述: 该函数移除与指定图元有关的所有连接信息, 如果场景开启了 `pIsCheckLinkPort` 属性, 会将对应端口置为未使用状态; 返回值为移除是否成功;

#### 6.20.4.18 removeItemLink

- a) 函数声明: `bool removeItemLink(ElaGraphicsItem* item1, ElaGraphicsItem* item2, int port1 = 0, int port2 = 0);`
- b) 参数 1: `ElaGraphicsItem* item1` // 图元 1;
- c) 参数 2: `ElaGraphicsItem* item2` // 图元 2;
- d) 参数 3: `int port1` // 图元 1 的端口;
- e) 参数 4: `int port2` // 图元 2 的端口;
- f) 返回值: `bool` // 移除是否成功;
- g) 描述: 该函数移除一条指定连接图元和指定端口的连接信息, 如果场景开启了 `pIsCheckLinkPort` 属性, 会将对应端口置为未使用状态; 返回值为移除是否成功;

#### 6.20.4.19 serialize

- a) 函数声明: `void serialize();`
- b) 返回值: `void`



- c) 描述：该函数对所有已存在于场景中的基类为 `ElaGraphicsItem` 的图元执行序列化操作，保存到二进制文件中，保存的文件名称和路径为 `pSerializePath` 属性指定的路径；保存的属性包括图元的所有基础属性以及坐标位置、连接信息等；

#### 6.20.4.20 deserialize

- a) 函数声明：`void deserialize();`
- b) 返回值：`void`
- c) 描述：该函数对已存在的二进制文件执行反序列化操作，恢复的文件名称和路径为 `pSerializePath` 属性指定的路径；将所有图元信息和相关连接信息还原到场景中；

#### 6.20.4.21 getItemsDataRoute

- a) 函数声明：`QVector<QVariantMap> getItemsDataRoute() const;`
- b) 返回值：`QVector<QVariantMap>` // 图元的自定义数据；
- c) 描述：该函数返回所有基类为 `ElaGraphicsItem` 的图元的自定义数据；

### 6.20.5 公有信号

#### 6.20.5.1 showItemLink

- a) 信号声明：`Q_SIGNAL void showItemLink ();`
- b) 描述：场景模式为 `ItemLink` 时，点击第二个被连接图元后，触发该信号；

#### 6.20.5.2 mouseLeftClickedItem

- a) 信号声明：`Q_SIGNAL void mouseLeftClickedItem (ElaGraphicsItem* item);`
- b) 描述：当某个图元被鼠标左键点击时，触发该信号，当多个图元堆叠时，该信号只会在最上层的 `Item` 上被触发；

#### 6.20.5.3 mouseRightClickedItem

- a) 信号声明：`Q_SIGNAL void mouseRightClickedItem (ElaGraphicsItem* item);`
- b) 描述：当某个图元被鼠标右键点击时，触发该信号，当多个图元堆叠时，该信号只会在最上层的 `Item` 上被触发；

#### 6.20.5.4 mouseDoubleClickedItem

- a) 信号声明：`Q_SIGNAL void mouseDoubleClickedItem (ElaGraphicsItem* item);`
- b) 描述：当某个图元被鼠标双击时，触发该信号，当多个图元堆叠时，该信号只会

在最上层的 Item 上被触发；

## 6.21 ElaGraphicsView

### 6.21.1 组件说明

升级版视图组件，支持自由缩放大小和鼠标拖动。

### 6.21.2 属性成员

#### 6.21.2.1 MaxTransform

- a) 类型: `qreal`
- b) 可用属性: `pMaxTransform`
- c) 关联信号: `pMaxTransformChanged`
- d) 默认值: `5`
- e) 描述: 该属性控制视图的最大缩放倍数；
- f) 对应的设置和获取函数:

```
void setMaxTransform (qreal maxTransform);  
qreal getMaxTransform () const;
```

#### 6.21.2.2 MinTransform

- a) 类型: `qreal`
- b) 可用属性: `pMinTransform`
- c) 关联信号: `pMinTransformChanged`
- d) 默认值: `0.15`
- e) 描述: 该属性控制视图的最小缩放倍数；
- f) 对应的设置和获取函数:

```
void setMinTransform (qreal minTransform);  
qreal getMinTransform () const;
```

## 6.22 ElaIcon

### 6.22.1 组件说明

单例类，用于快速获取基于 `ElaIconType` 创建的图标。



## 6.22.2 枚举定义

### 6.22.2.1 ElaIconType::IconName

该枚举用于指定使用的图标；

```
enum IconName;    // 库自带的图标对应的枚举，由于数量较大，此处不再列举；
```

## 6.22.3 公有函数

### 6.22.3.1 getElaIcon

- a) 函数声明： `QIcon getElaIcon(ElaIconType::IconName awesome);`
- b) 参数 1： `ElaIconType::IconName awesome` // 指定的图标；
- c) 返回值： `QIcon` // 创建的 `QIcon` 类型图标
- d) 描述：该函数以指定的 `ElaIconType` 图标类型创建一个 `QIcon` 类型的图标，字体的 `pixelsize` 默认为 25，内部 `Pixmap` 大小默认为宽 30，高 30；

### 6.22.3.2 getElaIcon

- a) 函数声明： `QIcon getElaIcon(ElaIconType::IconName awesome, QColor iconColor);`
- b) 参数 1： `ElaIconType::IconName awesome` // 指定的图标；
- c) 参数 2： `QColor iconColor` // 指定的图标颜色；
- d) 返回值： `QIcon` // 创建的 `QIcon` 类型图标
- e) 描述：该函数以指定的 `ElaIconType` 图标类型创建一个 `QIcon` 类型的图标，字体颜色为 `iconColor`，字体的 `pixelsize` 默认为 25，内部 `Pixmap` 大小默认为宽 30，高 30；

### 6.22.3.3 getElaIcon

- a) 函数声明： `QIcon getElaIcon(ElaIconType::IconName awesome, int pixelSize);`
- b) 参数 1： `ElaIconType::IconName awesome` // 指定的图标；
- c) 参数 2： `int pixelSize` // 指定的图标字体大小；
- d) 返回值： `QIcon` // 创建的 `QIcon` 类型图标
- e) 描述：该函数以指定的 `ElaIconType` 图标类型创建一个 `QIcon` 类型的图标，字体颜色为 `iconColor`，字体的 `pixelsize` 默认为 25，内部 `Pixmap` 大小默认为宽 30，高 30；

#### 6.22.3.4 getElaIcon

- a) 函数声明: `QIcon getElaIcon(ElaIconType::IconName awesome, int pixelSize, QColor iconColor);`
- b) 参数 1: `ElaIconType::IconName awesome` // 指定的图标;
- c) 参数 2: `int pixelSize` // 指定的图标字体大小;
- d) 参数 3: `QColor iconColor` // 指定的图标颜色;
- e) 返回值: `QIcon` // 创建的 `QIcon` 类型图标
- f) 描述: 该函数以指定的 `ElaIconType` 图标类型创建一个 `QIcon` 类型的图标, 字体颜色为 `iconColor`, 字体的 `pixelsize` 为指定大小, 内部 `Pixmap` 大小默认为宽 30, 高 30;

#### 6.22.3.5 getElaIcon

- a) 函数声明: `QIcon getElaIcon(ElaIconType::IconName awesome, int pixelSize, int fixedWidth, int fixedHeight);`
- b) 参数 1: `ElaIconType::IconName awesome` // 指定的图标;
- c) 参数 2: `int pixelSize` // 指定的图标字体大小;
- d) 参数 3: `int fixedWidth` // 内部 `Pixmap` 宽度;
- e) 参数 4: `int fixedHeight` // 内部 `Pixmap` 高度;
- f) 返回值: `QIcon` // 创建的 `QIcon` 类型图标
- g) 描述: 该函数以指定的 `ElaIconType` 图标类型创建一个 `QIcon` 类型的图标, 字体的 `pixelsize` 为指定大小, 内部 `Pixmap` 大小默认为宽 `fixedWidth`, 高 `fixedHeight`;

#### 6.22.3.6 getElaIcon

- a) 函数声明: `QIcon getElaIcon(ElaIconType::IconName awesome, int pixelSize, int fixedWidth, int fixedHeight, QColor iconColor);`
- b) 参数 1: `ElaIconType::IconName awesome` // 指定的图标;
- c) 参数 2: `int pixelSize` // 指定的图标字体大小;
- d) 参数 3: `int fixedWidth` // 内部 `Pixmap` 宽度;
- e) 参数 4: `int fixedHeight` // 内部 `Pixmap` 高度;
- f) 参数 5: `QColor iconColor` // 指定的图标颜色;
- g) 返回值: `QIcon` // 创建的 `QIcon` 类型图标
- h) 描述: 该函数以指定的 `ElaIconType` 图标类型创建一个 `QIcon` 类型的图标, 字体颜色为 `iconColor`, 字体的 `pixelsize` 为指定大小, 内部 `Pixmap` 大小默认为宽 `fixedWidth`, 高 `fixedHeight`;

edWidth, 高 fixedHeight;

## 6.23 ElaIconButton

### 6.23.1 组件说明

基于 **ElaIconType** 创建的图标按钮, 跟随主题变换颜色, 底色透明。

### 6.23.2 属性成员

#### 6.23.2.1 BorderRadius

- a) 类型: **int**
- b) 可用属性: **pBorderRadius**
- c) 关联信号: **pBorderRadiusChanged**
- d) 默认值: **5**
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);
```

```
int getBorderRadius () const;
```

#### 6.23.2.2 Opacity

- a) 类型: **qreal**
- b) 可用属性: **pOpacity**
- c) 关联信号: **pOpacityChanged**
- d) 默认值: **1**
- e) 描述: 该属性控制组件的整体透明度, 1 为不透明, 0 为完全透明;
- f) 对应的设置和获取函数:

```
void setOpacity (qreal opacity);
```

```
qreal getOpacity () const;
```

#### 6.23.2.3 LightHoverColor

- a) 类型: **QColor**
- b) 可用属性: **pLightHoverColor**
- c) 关联信号: **pLightHoverColorChanged**
- d) 默认值: 由 **ElaTheme** 决定
- e) 描述: 该属性控制组件浅色主题下鼠标覆盖时的颜色;
- f) 对应的设置和获取函数:

```
void setLightHoverColor (QColor lightHoverColor);  
QColor getLightHoverColor () const;
```

#### 6.23.2.4 DarkHoverColor

- a) 类型: QColor
- b) 可用属性: pDarkHoverColor
- c) 关联信号: pDarkHoverColorChanged
- d) 默认值: 由 ElaTheme 决定
- e) 描述: 该属性控制组件深色主题下鼠标覆盖时的颜色;
- f) 对应的设置和获取函数:

```
void setDarkHoverColor (QColor darkHoverColor);  
QColor getDarkHoverColor () const;
```

#### 6.23.2.5 LightIconColor

- a) 类型: QColor
- b) 可用属性: pLightIconColor
- c) 关联信号: pLightIconColorChanged
- d) 默认值: 由 ElaTheme 决定
- e) 描述: 该属性控制组件浅色主题下图标的颜色;
- f) 对应的设置和获取函数:

```
void setLightIconColor (QColor lightIconColor);  
QColor getLightIconColor () const;
```

#### 6.23.2.6 DarkIconColor

- a) 类型: QColor
- b) 可用属性: pDarkIconColor
- c) 关联信号: pDarkIconColorChanged
- d) 默认值: 由 ElaTheme 决定
- e) 描述: 该属性控制组件深色主题下图标的颜色;
- f) 对应的设置和获取函数:

```
void setDarkIconColor (QColor darkIconColor);  
QColor getDarkIconColor () const;
```

#### 6.23.2.7 LightHoverIconColor

- a) 类型: QColor

- b) 可用属性: `pLightHoverIconColor`
- c) 关联信号: `pLightHoverIconColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件浅色主题下鼠标覆盖时图标的颜色;
- f) 对应的设置和获取函数:

```
void setLightHoverIconColor (QColor lightHoverIconColor);  
QColor getLightHoverIconColor () const;
```

#### 6.23.2.8 DarkHoverIconColor

- a) 类型: `QColor`
- b) 可用属性: `pDarkHoverIconColor`
- c) 关联信号: `pDarkHoverIconColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件深色主题下鼠标覆盖时图标的颜色;
- f) 对应的设置和获取函数:

```
void setDarkHoverIconColor (QColor darkHoverIconColor);  
QColor getDarkHoverIconColor () const;
```

### 6.23.3 公有函数

#### 6.23.3.1 setAwesome

- a) 函数声明: `void setAwesome(ElaIconType::IconName awesome);`
- b) 参数 1: `ElaIconType::IconName awesome` // 指定的图标;
- c) 返回值: `void`
- d) 描述: 该函数设置按钮显示的 `ElaIconType` 类型的图标

#### 6.23.3.2 getAwesome

- a) 函数声明: `ElaIconType::IconName getAwesome() const;`
- b) 返回值: `ElaIconType::IconName awesome` // 显示的图标;
- c) 描述: 该函数获取按钮显示的 `ElaIconType` 类型的图标

## 6.24 ElaImageCard

### 6.24.1 组件说明

纯图片卡片组件, 支持卡片尺寸改变时横纵比不变的显示模式。

## 6.24.2 属性成员

### 6.24.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `6`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

### 6.24.2.2 CardImage

- a) 类型: `QImage`
- b) 可用属性: `pCardImage`
- c) 关联信号: `pCardImageChanged`
- d) 默认值: 空
- e) 描述: 该属性为指定的自定义图片;
- f) 对应的设置和获取函数:

```
void setCardImage (QImage cardImage);  
QImage getCardImage () const;
```

### 6.24.2.3 IsPreserveAspectCrop

- a) 类型: `bool`
- b) 可用属性: `pIsPreserveAspectCrop`
- c) 关联信号: `pIsPreserveAspectCropChanged`
- d) 默认值: `true`
- e) 描述: 该属性控制是否启用纵横比一致模式, 启用后, 在卡片的尺寸发生改变时, 自动保持纵横比一致, 这可能会使图片显示不全;
- f) 对应的设置和获取函数:

```
void setIsPreserveAspectCrop (bool isPreserveAspectCrop);  
bool getIsPreserveAspectCrop () const;
```

### 6.24.2.4 MaximumAspectRatio

- a) 类型: `qreal`

- b) 可用属性: `pMaximumAspectRatio`
- c) 关联信号: `pMaximumAspectRatioChanged`
- d) 默认值: `2.2`
- e) 描述: 该属性控制启用横纵比一致模式后, 所展示图片的临界横纵比; 在卡片的横纵比小于此值时, 图片的横纵比将会被固化为此值, 并自行处理高度; 卡片横纵比大于此值时, 会自行处理图片的源尺寸以保持横纵比一致;
- f) 对应的设置和获取函数:

```
void setMaximumAspectRatio (qreal maximumAspectRatio);  
qreal getMaximumAspectRatio () const;
```

## 6.25 ElaInteractiveCard

### 6.25.1 组件说明

带图片的交互式卡片组件, 透明底色。

### 6.25.2 属性成员

#### 6.25.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `6`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

#### 6.25.2.2 Title

- a) 类型: `QString`
- b) 可用属性: `pTitle`
- c) 关联信号: `pTitleChanged`
- d) 默认值: 空
- e) 描述: 该属性为主标题文字;
- f) 对应的设置和获取函数:

```
void setTitle (QString title);  
QString getTitle () const;
```



### 6.25.2.3 SubTitle

- a) 类型: `QString`
- b) 可用属性: `pSubTitle`
- c) 关联信号: `pSubTitleChanged`
- d) 默认值: 空
- e) 描述: 该属性为副标题文字;
- f) 对应的设置和获取函数:

```
void setSubTitle (QString subTitle);  
QString getSubTitle () const;
```

### 6.25.2.4 TitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pTitlePixelSize`
- c) 关联信号: `pTitlePixelSizeChanged`
- d) 默认值: 15
- e) 描述: 该属性为绘制主标题时, QPainter 的 `pixelSize` 大小;
- f) 对应的设置和获取函数:

```
void setTitlePixelSize (int titlePixelSize);  
int getTitlePixelSize () const;
```

### 6.25.2.5 SubTitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pSubTitlePixelSize`
- c) 关联信号: `pSubTitlePixelSizeChanged`
- d) 默认值: 12
- e) 描述: 该属性为绘制副标题时, QPainter 的 `pixelSize` 大小;
- f) 对应的设置和获取函数:

```
void setSubTitlePixelSize (int subTitlePixelSize);  
int getSubTitlePixelSize () const;
```

### 6.25.2.6 TitleSpacing

- a) 类型: `int`
- b) 可用属性: `pTitleSpacing`
- c) 关联信号: `pTitleSpacingChanged`



- d) 默认值: 2
- e) 描述: 该属性为文字距组件纵向中心点的间隔;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (int titleSpacing);  
int getTitleSpacing () const;
```

#### 6.25.2.7 CardPixmap

- a) 类型: QPixmap
- b) 可用属性: pCardPixmap
- c) 关联信号: pCardPixmapChanged
- d) 默认值: 空
- e) 描述: 该属性为指定的自定义图片;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (QPixmap cardPixmap);  
QPixmap getCardPixmap () const;
```

#### 6.25.2.8 CardPixmapSize

- a) 类型: QSize
- b) 可用属性: pCardPixmapSize
- c) 关联信号: pCardPixmapSizeChanged
- d) 默认值: QSize(64 , 64)
- e) 描述: 该属性控制自定义图片的绘制尺寸;
- f) 对应的设置和获取函数:

```
void setCardPixmapSize (QSize cardPixmapSize);  
QSize getCardPixmapSize () const;
```

#### 6.25.2.9 CardPixmapBorderRadius

- a) 类型: int
- b) 可用属性: pCardPixmapBorderRadius
- c) 关联信号: pCardPixmapBorderRadiusChanged
- d) 默认值: 6
- e) 描述: 该属性控制自定义图片外围边框的圆角半径, 值越大圆角越明显; 仅在 pCardPixMode 属性为 ElaCardPixType::RoundedRect 时生效;
- f) 对应的设置和获取函数:

```
void setCardPixmapBorderRadius (int cardPixmapBorderRadius);
```

```
int getCardPixmapBorderRadius () const;
```

## 6.25.3 公有函数

### 6.25.3.1 setCardPixmapSize

- a) 函数声明: `void setCardPixmapSize (int width, int height);`
- b) 参数 1: `int width` // 自定义图片宽度
- c) 参数 2: `int height` // 自定义图片高度
- d) 返回值: `void`
- e) 描述: 该函数控制自定义图片的绘制尺寸, 与直接设置 `pCardPixmapSize` 属性效果一致;
- f) 使用示例: `setCardPixmapSize(50 , 50);` // 设置图片尺寸为宽 50, 高 50;

## 6.26 ElaLineEdit

### 6.26.1 组件说明

带焦点指示器的输入框组件, 点击组件外区域时自动失去焦点。

### 6.26.2 属性成员

#### 6.26.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `6`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

#### 6.26.2.2 Awesome

- a) 类型: `ElaIconType::IconName`
- b) 可用属性: `pAwesome`
- c) 关联信号: `pAwesomeChanged`
- d) 默认值: `ElaIconType::None`;
- e) 描述: 该属性为菜单栏最右侧的自定义图标; 待定功能, 该属性后期可能会被

修改:

- f) 对应的设置和获取函数:
- g) `void setAwesome (ElaIconType::IconName awesome);`  
`ElaIconType::IconName getBorderRadius () const;`

## 6.26.3 公有信号

### 6.26.3.1 focusIn

- a) 信号声明: `Q_SIGNAL void focusIn (QString text);`
- b) 参数 1: `QString text` // 输入框文字
- c) 描述: 输入框获得焦点时, 触发该信号, 参数为输入框当前文字;

### 6.26.3.2 focusOut

- a) 信号声明: `Q_SIGNAL void focusOut (QString text);`
- b) 参数 1: `QString text` // 输入框文字
- c) 描述: 输入框失去焦点时, 触发该信号, 参数为输入框当前文字;

### 6.26.3.3 wmFocusOut

- a) 信号声明: `Q_SIGNAL void wmFocusOut (QString text);`
- b) 参数 1: `QString text` // 输入框文字
- c) 描述: 输入框指示器变化时, 触发该信号, 参数为输入框当前文字; 注意, 输入框指示器不代表实际的焦点变化, 如在输入框右键呼出菜单时, 焦点指示器不会变动;

## 6.27 ElaListView

### 6.27.1 组件说明

列表组件, 支持透明底色显示模式。

### 6.27.2 属性成员

#### 6.27.2.1 ItemHeight

- a) 类型: `int`
- b) 可用属性: `pItemHeight`
- c) 关联信号: `pItemHeightChanged`
- d) 默认值: `35`

- e) 描述：该属性控制列表项的高度；
- f) 对应的设置和获取函数：

```
void setItemHeight (int itemHeight);  
int getItemHeight () const;
```

#### 6.27.2.2 IsTransparent

- a) 类型：bool
- b) 可用属性：pIsTransparent
- c) 关联信号：pIsTransparentChanged
- d) 默认值：false
- e) 描述：该属性控制列表是否透明显示，不绘制背景和边框；
- f) 对应的设置和获取函数：

```
void setIsTransparent (bool isTransparent);  
bool getIsTransparent () const;
```

### 6.28 ElaLog

#### 6.28.1 组件说明

单例类，提供带 Debug 输出的日志记录功能。

#### 6.28.2 属性成员

##### 6.28.2.1 LogSavePath

- a) 类型：QString
- b) 可用属性：pLogSavePath
- c) 关联信号：pLogSavePathChanged
- d) 默认值：QDir::currentPath();
- e) 描述：该属性控制日志文件的存储目录；
- f) 对应的设置和获取函数：

```
void setLogSavePath (QString logSavePath);  
QString getLogSavePath () const;
```

##### 6.28.2.2 LogFileName

- a) 类型：QString
- b) 可用属性：pLogFileName
- c) 关联信号：pLogFileNameChanged

- d) 默认值: `ElaLog`
- e) 描述: 该属性控制日志文件的存储名称, 无需指定后缀;
- f) 对应的设置和获取函数:

```
void setLogFileName (QString logFileName);  
QString getLogFileName () const;
```

#### 6.28.2.3 IsLogFileNameWithTime

- a) 类型: `bool`
- b) 可用属性: `pIsLogFileNameWithTime`
- c) 关联信号: `pIsLogFileNameWithTimeChanged`
- d) 默认值: `false`
- e) 描述: 该属性控制是否在日志文件的存储名称后自动添加时间信息, 若启用此属性, 每次程序启动时都会将日志存储到不同的文件中;
- f) 对应的设置和获取函数:

```
void setIsLogFileNameWithTime (bool isLogFileNameWithTime);  
bool getIsLogFileNameWithTime () const;
```

### 6.28.3 公有函数

#### 6.28.3.1 initMessageLog

- a) 函数声明: `void initMessageLog(bool isEnabled);`
- b) 参数 1: `bool isEnabled` // 是否启用日志捕获
- c) 返回值: `void`
- d) 描述: 该函数控制日志的启用状态与否;
- e) 使用示例: `ElaLog::getInstance()->initMessageLog(true);` // 启用日志记录

### 6.28.4 公有信号

#### 6.28.4.1 logMessage

- a) 信号声明: `Q_SIGNAL void logMessage (QString log);`
- b) 参数 1: `QString log` // 记录的日志字符串
- c) 描述: 当日志系统记录一条信息时, 触发此信号;

## 6.29 ElaMenu

### 6.29.1 组件说明

带展开动画和自定义图标的菜单组件。

### 6.29.2 公有函数

#### 6.29.2.1 setMenuItemHeight

- a) 函数声明: `void setMenuItemHeight(int menuItemHeight);`
- b) 参数 1: `int menuItemHeight` // 菜单项的高度
- c) 返回值: `void`
- d) 描述: 该函数控制菜单项的高度;

#### 6.29.2.2 getMenuItemHeight

- a) 函数声明: `int getMenuItemHeight() const;`
- b) 返回值: `int` // 菜单项的高度
- c) 描述: 该函数获取菜单项的高度;

#### 6.29.2.3 addMenu

- a) 函数声明: `QAction* addMenu(QMenu* menu);`
- b) 参数 1: `QMenu* menu` // 需要新增的菜单
- c) 返回值: `QAction*` // 菜单项的高度
- d) 描述: 该函数新增一个菜单, 并返回菜单对应的 `QAction`;

#### 6.29.2.4 addMenu

- a) 函数声明: `ElaMenu* addMenu(const QString& title);`
- b) 参数 1: `const QString& title` // 菜单标题
- c) 返回值: `ElaMenu*` // 新增的菜单指针
- d) 描述: 该函数新增一个 `ElaMenu` 类型的菜单, 菜单标题为 `title`, 并返回菜单指针;

#### 6.29.2.5 addMenu

- a) 函数声明: `ElaMenu* addMenu(const QIcon& icon, const QString& title);`
- b) 参数 1: `const QIcon& icon` // 菜单图标
- c) 参数 2: `const QString& title` // 菜单标题

- d) 返回值: `ElaMenu*` // 新增的菜单指针
- e) 描述: 该函数新增一个 `ElaMenu` 类型的菜单, 菜单标题为 `title`, 图标为 `icon`, 并返回菜单指针;

#### 6.29.2.6 addMenu

- a) 函数声明: `ElaMenu* addMenu(ElaIconType::IconName icon, const QString& title);`
- b) 参数 1: `ElaIconType::IconName icon` // 菜单图标
- c) 参数 2: `const QString& title` // 菜单标题
- d) 返回值: `ElaMenu*` // 新增的菜单指针
- e) 描述: 该函数新增一个 `ElaMenu` 类型的菜单, 菜单标题为 `title`, 图标为 `icon`, 并返回菜单指针;

#### 6.29.2.7 addElaIconAction

- a) 函数声明: `QAction* addElaIconAction(ElaIconType::IconName icon, const QString& text);`
- b) 参数 1: `ElaIconType::IconName icon` // 菜单项图标
- c) 参数 2: `const QString& text` // 菜单项文字
- d) 返回值: `QAction *` // 新增的菜单 Action
- e) 描述: 该函数新增一个菜单项, 菜单项标题为 `title`, 图标为 `icon`, 并返回菜单项对应的 `QAction`;

#### 6.29.2.8 addElaIconAction

- a) 函数声明: `QAction* addElaIconAction(ElaIconType::IconName icon, const QString& text, const QKeySequence& shortcut);`
- b) 参数 1: `ElaIconType::IconName icon` // 菜单项图标
- c) 参数 2: `const QString& text` // 菜单项文字
- d) 参数 3: `const QKeySequence& shortcut` // 菜单项快捷键
- e) 返回值: `QAction *` // 新增的菜单 Action
- f) 描述: 该函数新增一个菜单项, 菜单项标题为 `title`, 图标为 `icon`, 快捷键为 `shortcut` 并返回菜单项对应的 `QAction`;

#### 6.29.2.9 isHasChildMenu

- a) 函数声明: `bool isHasChildMenu() const;`
- b) 返回值: `bool` // 是否有子菜单



- c) 描述：该函数返回组件是否具有子菜单项；

#### 6.29.2.10 isHasIcon

- a) 函数声明：`bool isHasIcon() const;`
- b) 返回值：`bool` // 是否有图标
- c) 描述：该函数返回组件的任一菜单项是否具有图标；

### 6.29.3 公有信号

#### 6.29.3.1 menuShow

- a) 信号声明：`Q_SIGNAL void menuShow ();`
- b) 描述：菜单从不可见变为可见状态时，触发该信号；

## 6.30 ElaMenuBar

### 6.30.1 组件说明

带展开动画和自定义图标的菜单栏组件。

### 6.30.2 公有函数

#### 6.30.2.1 addMenu

- a) 函数声明：`QAction* addMenu(QMenu* menu);`
- b) 参数 1：`QMenu* menu` // 需要新增的菜单
- c) 返回值：`QAction*` // 菜单项的高度
- d) 描述：该函数新增一个菜单，并返回菜单对应的 `QAction`；

#### 6.30.2.2 addMenu

- a) 函数声明：`ElaMenu* addMenu(const QString& title);`
- b) 参数 1：`const QString& title` // 菜单标题
- c) 返回值：`ElaMenu*` // 新增的菜单指针
- d) 描述：该函数新增一个 `ElaMenu` 类型的菜单，菜单标题为 `title`，并返回菜单指针；

#### 6.30.2.3 addMenu

- a) 函数声明：`ElaMenu* addMenu(const QIcon& icon, const QString& title);`
- b) 参数 1：`const QIcon& icon` // 菜单图标



- c) 参数 2: `const QString& title` // 菜单标题
- d) 返回值: `ElaMenu*` // 新增的菜单指针
- e) 描述: 该函数新增一个 `ElaMenu` 类型的菜单, 菜单标题为 `title`, 图标为 `icon`, 并返回菜单指针;

#### 6.30.2.4 addMenu

- a) 函数声明: `ElaMenu* addMenu(ElaIconType::IconName icon, const QString& title);`
- b) 参数 1: `ElaIconType::IconName icon` // 菜单图标
- c) 参数 2: `const QString& title` // 菜单标题
- d) 返回值: `ElaMenu*` // 新增的菜单指针
- e) 描述: 该函数新增一个 `ElaMenu` 类型的菜单, 菜单标题为 `title`, 图标为 `icon`, 并返回菜单指针;

#### 6.30.2.5 addElaIconAction

- a) 函数声明: `QAction* addElaIconAction(ElaIconType::IconName icon, const QString& text);`
- b) 参数 1: `ElaIconType::IconName icon` // 菜单项图标
- c) 参数 2: `const QString& text` // 菜单项文字
- d) 返回值: `QAction *` // 新增的菜单 Action
- e) 描述: 该函数新增一个菜单项, 菜单项标题为 `title`, 图标为 `icon`, 并返回菜单项对应的 `QAction`;

#### 6.30.2.6 addElaIconAction

- a) 函数声明: `QAction* addElaIconAction(ElaIconType::IconName icon, const QString& text, const QKeySequence& shortcut);`
- b) 参数 1: `ElaIconType::IconName icon` // 菜单项图标
- c) 参数 2: `const QString& text` // 菜单项文字
- d) 参数 3: `const QKeySequence& shortcut` // 菜单项快捷键
- e) 返回值: `QAction *` // 新增的菜单 Action

### 6.31 ElaMessageBar

#### 6.31.1 组件说明

浮动式提示组件, 支持八方向弹出, 随窗口改变位置, 使用静态公有函数调用。

## 6.31.2 枚举定义

### 6.31.2.1 ElaMessageBarType:: PositionPolicy

该枚举用于控制提示的弹出位置；

```
enum PositionPolicy
{
    Top = 0x0000, // 顶部;
    Left = 0x0001, // 左侧;
    Bottom = 0x0002, // 底部;
    Right = 0x0003, // 右侧;
    TopRight = 0x0004, // 右上侧;
    TopLeft = 0x0005, // 左上侧;
    BottomRight = 0x0006, // 右下侧;
    BottomLeft = 0x0007, // 左下侧;
};
```

### 6.31.2.2 ElaMessageBarType:: MessageMode

该枚举用于控制提示的类型；

```
enum MessageMode
{
    Success = 0x0000, // 成功;
    Warning = 0x0001, // 警告;
    Information = 0x0002, // 信息;
    Error = 0x0003, // 错误;
};
```

## 6.31.3 公有函数

### 6.31.3.1 success

- a) 函数声明: `static void success(ElaMessageBarType::PositionPolicy policy, QString title, QString text, int displayMsec, QWidget* parent = nullptr);`
- b) 参数 1: `ElaMessageBarType::PositionPolicy policy` // 弹出位置;
- c) 参数 2: `QString title` // 标题;
- d) 参数 3: `QString text` // 内容;
- e) 参数 4: `int displayMsec` // 显示时间;

- f) 参数 5: `QWidget* parent` // 目标窗口
- g) 返回值: `void`
- h) 描述: 该函数创建一个 `success` 样式的浮动式提示, 位置策略为 `policy`, 标题为 `title`, 文字为 `text`, 显示时间为 `displayMsec`, 目标窗口为 `parent`; 若不指定目标窗口, 则默认为 `ElaWindow` 窗口;

#### 6.31.3.2 warning

- a) 函数声明: `static void warning(ElaMessageBarType::PositionPolicy policy, QString title, QString text, int displayMsec, QWidget* parent = nullptr);`
- b) 参数 1: `ElaMessageBarType::PositionPolicy policy` // 弹出位置;
- c) 参数 2: `QString title` // 标题;
- d) 参数 3: `QString text` // 内容;
- e) 参数 4: `int displayMsec` // 显示时间;
- f) 参数 5: `QWidget* parent` // 目标窗口
- g) 返回值: `void`
- h) 描述: 该函数创建一个 `warning` 样式的浮动式提示, 位置策略为 `policy`, 标题为 `title`, 文字为 `text`, 显示时间为 `displayMsec`, 目标窗口为 `parent`; 若不指定目标窗口, 则默认为 `ElaWindow` 窗口;

#### 6.31.3.3 information

- a) 函数声明: `static void information(ElaMessageBarType::PositionPolicy policy, QString title, QString text, int displayMsec, QWidget* parent = nullptr);`
- b) 参数 1: `ElaMessageBarType::PositionPolicy policy` // 弹出位置;
- c) 参数 2: `QString title` // 标题;
- d) 参数 3: `QString text` // 内容;
- e) 参数 4: `int displayMsec` // 显示时间;
- f) 参数 5: `QWidget* parent` // 目标窗口
- g) 返回值: `void`
- h) 描述: 该函数创建一个 `information` 样式的浮动式提示, 位置策略为 `policy`, 标题为 `title`, 文字为 `text`, 显示时间为 `displayMsec`, 目标窗口为 `parent`; 若不指定目标窗口, 则默认为 `ElaWindow` 窗口;

#### 6.31.3.4 error

- a) 函数声明: `static void error(ElaMessageBarType::PositionPolicy policy, QString title, QString text, int displayMsec, QWidget* parent = nullptr);`

- b) 参数 1: `ElaMessageBarType::PositionPolicy policy` // 弹出位置;
- c) 参数 2: `QString title` // 标题;
- d) 参数 3: `QString text` // 内容;
- e) 参数 4: `int displayMsec` // 显示时间;
- f) 参数 5: `QWidget* parent` // 目标窗口
- g) 返回值: `void`
- h) 描述: 该函数创建一个 `error` 样式的浮动式提示, 位置策略为 `policy`, 标题为 `title`, 文字为 `text`, 显示时间为 `displayMsec`, 目标窗口为 `parent`; 若不指定目标窗口, 则默认为 `ElaWindow` 窗口;

## 6.32 ElaMessageButton

### 6.32.1 组件说明

用于快捷触发浮动式提示组件的按钮。

### 6.32.2 属性成员

#### 6.32.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `3`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

#### 6.32.2.2 BarTitle

- a) 类型: `QString`
- b) 可用属性: `pBarTitle`
- c) 关联信号: `pBarTitleChanged`
- d) 默认值: 空
- e) 描述: 该属性控制提示的标题;
- f) 对应的设置和获取函数:

```
void setBarTitle (QString barTitle);  
QString getBarTitle () const;
```

### 6.32.2.3 BarText

- a) 类型: `QString`
- b) 可用属性: `pBarText`
- c) 关联信号: `pBarTextChanged`
- d) 默认值: 空
- e) 描述: 该属性控制提示的内容;
- f) 对应的设置和获取函数:

```
void setBarText (QString barText);  
QString getBarText () const;
```

### 6.32.2.4 DisplayMsec

- a) 类型: `int`
- b) 可用属性: `pDisplayMsec`
- c) 关联信号: `pDisplayMsecChanged`
- d) 默认值: 2000
- e) 描述: 该属性控制提示显示的时间, 单位为 ms;
- f) 对应的设置和获取函数:

```
void setDisplayMsec (int displayMsec);  
int getDisplayMsec () const;
```

### 6.32.2.5 MessageTargetWidget

- a) 类型: `QWidget*`
- b) 可用属性: `pMessageTargetWidget`
- c) 关联信号: `pMessageTargetWidgetChanged`
- d) 默认值: `nullptr`
- e) 描述: 该属性控制提示显示的目标窗口;
- f) 对应的设置和获取函数:

```
void setMessageTargetWidget (QWidget* messageTargetWidget);  
QWidget* getMessageTargetWidget () const;
```

### 6.32.2.6 MessageMode

- a) 类型: `ElaMessageBarType::MessageMode`
- b) 可用属性: `pMessageMode`
- c) 关联信号: `pMessageModeWidgetChanged`

- d) 默认值: `ElaMessageBarType::Success`
- e) 描述: 该属性控制提示显示的类型;
- f) 对应的设置和获取函数:

```
void setMessageMode (ElaMessageBarType::MessageMode messageMode);  
ElaMessageBarType::MessageMode getMessageMode () const;
```

#### 6.32.2.7 PositionPolicy

- a) 类型: `ElaMessageBarType::PositionPolicy`
- b) 可用属性: `pPositionPolicy`
- c) 关联信号: `pPositionPolicyChanged`
- d) 默认值: `ElaMessageBarType::TopRight`
- e) 描述: 该属性控制提示显示的位置;
- f) 对应的设置和获取函数:

```
void setPositionPolicy (ElaMessageBarType::PositionPolicy positionPolicy);  
ElaMessageBarType::PositionPolicy getPositionPolicy () const;
```

### 6.33 ElaMultiSelectComboBox

#### 6.33.1 组件说明

带展开动画的多选下拉框。

#### 6.33.2 属性成员

##### 6.33.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `3`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

### 6.33.3 公有函数

#### 6.33.3.1 setCurrentSelection

- a) 函数声明: `void setCurrentSelection(QString selection);`
- b) 参数 1: `QString selection` // 选中的选项条目
- c) 返回值: `void`
- d) 描述: 该函数将除 `selection` 外的所有条目设置为未选中状态, 若 `selection` 不存在, 则将所有条目置为未选中状态;

#### 6.33.3.2 setCurrentSelection

- a) 函数声明: `void setCurrentSelection(QStringList selection);`
- b) 参数 1: `QStringList selection` // 选中的选项条目列表
- c) 返回值: `void`
- d) 描述: 该函数将除 `selection` 外的所有条目设置为未选中状态, 若 `selection` 不存在, 则将所有条目置为未选中状态;

#### 6.33.3.3 setCurrentSelection

- a) 函数声明: `void setCurrentSelection(int index);`
- b) 参数 1: `int index` // 选中的选项索引
- c) 返回值: `void`
- d) 描述: 该函数将除索引为 `index` 外的所有条目设置为未选中状态, 若 `index` 不可用, 则不做操作;

#### 6.33.3.4 setCurrentSelection

- a) 函数声明: `void setCurrentSelection(QList<int> selectionIndex);`
- b) 参数 1: `QList<int> selectionIndex` // 选中的选项索引列表
- c) 返回值: `void`
- d) 描述: 该函数将除 `selectionIndex` 索引列表外的所有条目设置为未选中状态, 若 `index` 不可用, 则将所有条目置为未选中状态;

#### 6.33.3.5 getCurrentSelection

- a) 函数声明: `QStringList getCurrentSelection() const;`
- b) 返回值: `QStringList` // 选中的选项列表
- c) 描述: 该函数返回当前选中的选项列表;



### 6.33.3.6 getCurrentSelectionIndex

- a) 函数声明: `QList<int> getCurrentSelectionIndex() const;`
- b) 返回值: `QList<int>` // 选中的选项索引列表
- c) 描述: 该函数返回当前选中的选项索引列表;

## 6.33.4 公有信号

### 6.33.4.1 itemSelectionChanged

- a) 信号声明: `Q_SIGNAL void itemSelectionChanged (QVector<bool> itemSelection);`
- b) 参数 1: `QVector<bool> itemSelection` // 所有选项的状态列表
- c) 描述: 选中的选项变化时, 触发该信号;

### 6.33.4.2 currentTextListChanged

- a) 信号声明: `Q_SIGNAL void currentTextListChanged (QStringList selectedTextList);`
- b) 参数 1: `QStringList selectedTextList` // 当前选中的选项列表
- c) 描述: 选中的选项变化时, 触发该信号;

## 6.34 ElaNavigationBar

### 6.34.1 组件说明

三形态自定义导航栏组件, 支持多级列表嵌套。

### 6.34.2 枚举定义

#### 6.34.2.1 ElaNavigationType::NodeOperateReturnType

该枚举用于提示导航节点操作的结果;

```
enum NodeOperateReturnType
{
    Success = 0x0000, // 操作成功;
    TargetNodeInvalid = 0x0001, // 目标节点不可用;
    TargetNodeTypeError = 0x0002, // 目标节点类型错误;
    TargetNodeDepthLimit = 0x0003, // 目标节点类型错误;
    PageInvalid = 0x0004, // 添加的页面不可用;
    FooterUpperLimit = 0x0005, // 页脚节点达到上限 最多两个;
};
```



### 6.34.2.2 ElaNavigationType:: NavigationDisplayMode

该枚举用于提示导航栏展示模式；

```
enum NavigationDisplayMode
{
    Auto= 0x0000, // 自动模式，根据窗口大小自动改变；
    Minimal= 0x0001, // 最小化；
    Compact= 0x0002, // 图标列表模式；
    Maximal= 0x0003, // 展开列表模式；
};
```

### 6.34.2.3 ElaNavigationType:: NavigationNodeType

该枚举用于提示节点类型，用于内部逻辑计算；

```
enum NavigationNodeType
{
    PageNode= 0x0000, // 展开列表节点；
    FooterNode= 0x0001, // 页脚节点；
};
```

## 6.34.3 属性成员

### 6.34.3.1 IsTransparent

- a) 类型: **bool**
- b) 可用属性: **pIsTransparent**
- c) 关联信号: **pIsTransparentChanged**
- d) 默认值: **false**
- e) 描述: 该属性控制列表是否透明显示，不绘制背景；
- f) 对应的设置和获取函数:

```
void setIsTransparent (bool isTransparent);
bool getIsTransparent () const;
```

## 6.34.4 公有函数

### 6.34.4.1 setUserInfoCardVisible

- a) 函数声明: **void setUserInfoCardVisible(bool isVisible);**
- b) 参数 1: **bool isVisible** // 用户卡片可见性；

- c) 返回值: `void`
- d) 描述: 该函数设置导航栏上方用户卡片的可见性, `true` 为可见, `false` 为不可见;

#### 6.34.4.2 setUserInfoCardPixmap

- a) 函数声明: `void setUserInfoCardPixmap (QPixmap pix);`
- b) 参数 1: `QPixmap pix` // 用户卡片图片;
- c) 返回值: `void`
- d) 描述: 该函数设置导航栏上方用户卡片的显示图片;

#### 6.34.4.3 setUserInfoCardTitle

- a) 函数声明: `void setUserInfoCardTitle (QString title);`
- b) 参数 1: `QString title` // 用户卡片标题;
- c) 返回值: `void`
- d) 描述: 该函数设置导航栏上方用户卡片的标题;

#### 6.34.4.4 setUserInfoCardSubTitle

- a) 函数声明: `void setUserInfoCardSubTitle (QString subTitle);`
- b) 参数 1: `QString subTitle` // 用户卡片副标题;
- c) 返回值: `void`
- d) 描述: 该函数设置导航栏上方用户卡片的副标题;

#### 6.34.4.5 addExpanderNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addExpanderNode(QString expanderTitle, QString& expanderKey, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString expanderTitle` // 可展开节点的标题;
- c) 参数 2: `QString& expanderKey` // 可展开节点的识别字符串, 调用成功后被赋值;
- d) 参数 3: `ElaIconType::IconName awesome` // 可展开节点的图标;
- e) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- f) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `expanderTitle`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `expanderKey` 赋值, 值为该展开节点的识别字符串;
- g) 使用示例:  
`QString expanderKey;`

```
ElaNavigationType::NodeOperateReturnType returnType = addExpanderNode("可
展开节点 1", expanderKey, ElaIconType::House);
// 添加一个可展开节点，节点名称为“可展开节点 1”;
```

#### 6.34.4.6 addExpanderNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addExpanderNode(QString expanderTitle, QString& expanderKey, QString targetExpanderKey, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString expanderTitle` // 可展开节点的标题;
- c) 参数 2: `QString& expanderKey` // 可展开节点的识别字符串, 调用成功后被赋值;
- d) 参数 3: `QString targetExpanderKey` // 目标可展开节点的识别字符串;
- e) 参数 4: `ElaIconType::IconName awesome` // 可展开节点的图标;
- f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- g) 描述: 该函数为导航栏在指定节点上添加一个可展开节点, 节点标题为 `expanderTitle`, 指定可展开节点的识别字符串为 `targetExpanderKey`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `expanderKey` 赋值, 值为该展开节点的识别字符串;
- h) 使用示例:

```
QString expanderKey;
ElaNavigationType::NodeOperateReturnType returnType = addExpanderNode("可
展开节点 2", expanderKey, "86249d00b59a4a079bd3d3a38bebc484" , ElaIconTy
pe::House);
// 在指定可展开节点下添加一个可展开节点，节点名称为“可展开节点 2” 指定
的可展开节点的识别字符串为“86249d00b59a4a079bd3d3a38bebc484”;
```

#### 6.34.4.7 addPageNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString pageTitle` // 页面节点的标题;
- c) 参数 2: `QWidget* page` // 添加的页面;
- d) 参数 3: `ElaIconType::IconName awesome` // 页面节点的图标;
- e) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- f) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`,

添加的页面为 `page`，图标为 `awesome`，返回值为调用结果，若调用成功，对 `page` 添加属性，属性名称为 `ElaIconType`，值为该页面的识别字符串；

g) 使用示例：

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, ElaIconType::House);
// 添加一个页面节点，节点名称为“页面节点 1”；
```

#### 6.34.4.8 addPageNode

a) 函数声明：`ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, QString targetExpanderKey, ElaIconType::IconName awesome = ElaIconType::None);`

b) 参数 1：`QString pageTitle` // 页面节点的标题；

c) 参数 2：`QWidget* page` // 添加的页面；

d) 参数 3：`QString targetExpanderKey` // 目标可展开节点的识别字符串；

e) 参数 4：`ElaIconType::IconName awesome` // 页面节点的图标；

f) 返回值：`ElaNavigationType::NodeOperateReturnType` // 操作结果；

g) 描述：该函数为导航栏在指定可展开节点上添加一个可展开节点，节点标题为 `pageTitle`，添加的页面为 `page`，指定可展开节点的识别字符串为 `targetExpanderKey`，图标为 `awesome`，返回值为调用结果，若调用成功，对 `page` 添加属性，属性名称为 `ElaIconType`，值为该页面的识别字符串；

h) 使用示例：

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, "86249d00b59a4a079bd3d3a38bebc484", ElaIconType::House);
// 在指定的可展开节点下添加一个页面节点，节点名称为“页面节点 1” 指定的可展开节点的识别字符串为“86249d00b59a4a079bd3d3a38bebc484”；
```

#### 6.34.4.9 addPageNode

a) 函数声明：`ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`

b) 参数 1：`QString pageTitle` // 页面节点的标题；

c) 参数 2：`QWidget* page` // 添加的页面；

d) 参数 3：`int keyPoints` // 要点数 点击后，该要点归零；

- e) 参数 4: `ElaIconType::IconName awesome` // 页面节点的图标;
- f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- g) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- h) 使用示例:
 

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, 99, ElaIconType::House);
// 添加一个页面节点, 节点名称为“页面节点 1” 要点数为 99;
```

#### 6.34.4.10 addPageNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, QString targetExpanderKey, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString pageTitle` // 页面节点的标题;
- c) 参数 2: `QWidget* page` // 添加的页面;
- d) 参数 3: `QString targetExpanderKey` // 目标可展开节点的识别字符串;
- e) 参数 4: `int keyPoints` // 要点数 点击后, 该要点归零;
- f) 参数 5: `ElaIconType::IconName awesome` // 页面节点的图标;
- g) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- h) 描述: 该函数为导航栏在指定可展开节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 指定可展开节点的识别字符串为 `targetExpanderKey`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- i) 使用示例:
 

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, "86249d00b59a4a079bd3d3a38bebc484", 99, ElaIconType::House);
// 在指定的可展开节点下添加一个页面节点, 节点名称为“页面节点 1” 指定的可展开节点的识别字符串为“86249d00b59a4a079bd3d3a38bebc484” 要点数为 99;
```

#### 6.34.4.11 addFooterNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addFooterNode(QString`



- ```
footerTitle, QString& footerKey, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);
```
- b) 参数 1: `QString footerTitle` // 页脚节点的标题;
  - c) 参数 2: `QString& footerKey` // 页脚节点的识别字符串, 调用成功后被赋值;
  - d) 参数 3: `int keyPoints` // 要点数 点击后, 该要点归零;
  - e) 参数 4: `ElaIconType::IconName awesome` // 页面节点的图标;
  - f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
  - g) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
  - h) 使用示例:
 

```
QString footerKey;
ElaNavigationType::NodeOperateReturnType returnType = addFooterNode("页脚节点 1", footerKey, 99, ElaIconType::House);
// 添加一个页脚节点, 节点名称为“页脚节点 1” 要点数为 99;
```

#### 6.34.4.12 addFooterNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addFooterNode(QString footerTitle, QWidget* page, QString& footerKey, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString footerTitle` // 页脚节点的标题;
- c) 参数 2: `QWidget* page` // 页脚页面;
- d) 参数 3: `QString& footerKey` // 页脚节点的识别字符串, 调用成功后被赋值;
- e) 参数 4: `int keyPoints` // 要点数 点击后, 该要点归零;
- f) 参数 5: `ElaIconType::IconName awesome` // 页面节点的图标;
- g) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- h) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- i) 使用示例:
 

```
QString footerKey;
ElaNavigationType::NodeOperateReturnType returnType = addFooterNode("页脚节点 1", new QWidget(this), footerKey, 99, ElaIconType::House);
// 添加一个页脚节点, 节点名称为“页脚节点 1” 要点数为 99;
```

#### 6.34.4.13 setNodeKeyPoints

- a) 函数声明: `void setNodeKeyPoints(QString nodeKey, int keyPoints);`
- b) 参数 1: `QString nodeKey` // 页脚节点或页面节点的识别字符串;
- c) 参数 2: `int keyPoints` // 要点数;
- d) 返回值: `void`
- e) 描述: 该函数为指定的页脚节点或页面节点设置要点数;

#### 6.34.4.14 getNodeKeyPoints

- a) 函数声明: `int getNodeKeyPoints(QString nodeKey) const;`
- b) 参数 1: `QString nodeKey` // 页脚节点或页面节点的识别字符串;
- c) 返回值: `int` // 要点数;
- d) 描述: 该函数返回指定的页脚节点或页面节点的要点数;

#### 6.34.4.15 navigation

- a) 函数声明: `void navigation(QString pageKey, bool isLogClicked = true);`
- b) 参数 1: `QString pageKey` // 页脚节点或页面节点的识别字符串;
- c) 参数 2: `bool isLogClicked` // 是否记录跳转;
- d) 返回值: `void`
- e) 描述: 该函数使用指定的页脚节点或页面节点的识别字符串进行路由跳转操作, 是否记录跳转由 `isLogClicked` 指定;

#### 6.34.4.16 setDisplayMode

- a) 函数声明: `void setDisplayMode(ElaNavigationType::NavigationDisplayMode displayMode, bool isAnimation = true);`
- b) 参数 1: `ElaNavigationType::NavigationDisplayMode displayMode` // 导航栏显示模式;
- c) 参数 2: `bool isAnimation` // 是否启用模式转换动画;
- d) 返回值: `void`
- e) 描述: 该函数切换导航栏的显示模式, 是否启用动画由 `isAnimation` 指定;

### 6.34.5 公有信号

#### 6.34.5.1 userInfoCardClicked

- a) 信号声明: `Q_SIGNAL void userInfoCardClicked ();`
- b) 描述: 导航栏上方用户卡片被点击时, 触发该信号;

#### 6.34.5.2 navigationNodeClicked

- a) 信号声明: `Q_SIGNAL void navigationNodeClicked (ElaNavigationType::NavigationNodeType nodeType, QString nodeKey);`
- b) 参数 1: `ElaNavigationType::NavigationNodeType nodeType` // 节点类型;
- c) 参数 2: `QString nodeKey` // 节点的识别字符串;
- d) 描述: 导航栏页面或页脚节点被点击时 (不包括可展开节点), 触发该信号;

#### 6.34.5.3 navigationNodeAdded

- a) 信号声明: `Q_SIGNAL void navigationNodeAdded(ElaNavigationType::NavigationNodeType nodeType, QString nodeKey, QWidget* page);`
- b) 参数 1: `ElaNavigationType::NavigationNodeType nodeType` // 节点类型;
- c) 参数 2: `QString nodeKey` // 节点的识别字符串;
- d) 参数 3: `QWidget* page` // 被添加的页面指针;
- e) 描述: 导航栏页面或页脚节点被添加时 (不包括可展开节点), 触发该信号;

### 6.35 ElaNavigationRouter

#### 6.35.1 组件说明

单例类, 用于实现路由跳转功能, 支持自行添加和触发路由, 具体的路由逻辑由用户的处理函数指定。

#### 6.35.2 枚举定义

##### 6.35.2.1 ElaNavigationRouterType::NavigationRouteType

该枚举用于提示路由跳转的记录结果;

```
enum NavigationRouteType
{
    Success = 0x0000, // 记录成功;
    ObjectInvalid = 0x0001, // 处理对象不可用;
    FunctionNameInvalid = 0x0002, // 处理函数名称为空;
};
```

#### 6.35.3 属性成员

##### 6.35.3.1 MaxRouteCount

- a) 类型: `int`



- b) 可用属性: `pMaxRouteCount`
- c) 关联信号: `pMaxRouteCountChanged`
- d) 默认值: 25
- e) 描述: 该属性控制路由跳转的最大记录数, 记录数超过次数时, 会清除最早记录的一条数据;
- f) 对应的设置和获取函数:
 

```
void setMaxRouteCount (int maxRouteCount);
int getMaxRouteCount () const;
```

## 6.35.4 公有函数

### 6.35.4.1 navigationRoute

- a) 函数声明: `ElaNavigationRouterType::NavigationRouteType navigationRoute(QObject* routeObject, QString routeFunctionName, const QVariantMap& routeData = {}, Qt::ConnectionType connectionType = Qt::AutoConnection);`
- b) 参数 1: `QObject* routeObject` // 处理对象;
- c) 参数 2: `QString routeFunctionName` // 处理函数名称, 由用户自行指定函数名, 需注意, 该函数必须为 `routeObject` 持有的函数, 参数为固定的 `QVariantMap` 类型且必须附带 `Q_INVOKABLE` 属性;
- d) 参数 3: `const QVariantMap& routeData` // 处理内容, 由用户自行定义;
- e) 参数 4: `Qt::ConnectionType connectionType` // 处理函数调用类型, 由用户自行指定;
- f) 返回值: `ElaNavigationRouterType::NavigationRouteType` // 记录结果;
- g) 描述: 该函数指定一个处理对象 `routeObject` 和该对象持有的处理函数 `routeFunctionName`, 将其与处理内容 `routeData` 和调用方式 `connectionType` 一起新增到路由跳转队列中, 当跳转触发到该数据时, 自动调用 `routeObject` 对象的 `routeFunctionName` 函数;
- h) 使用示例:
 

```
Q_INVOKABLE void onNavigationRouteBack(QVariantMap routeData); // 处理函数声明;
QVariantMap routeData = QVariantMap();
routeData.insert("ElaPageKey", "key-123456"); // 处理数据;
ElaNavigationRouter::getInstance()->navigationRoute(this, "onNavigationRouteBack", routeData); // 记录路由跳转, 触发时, 自动调用 onNavigationRouteBack 函数;
```

#### 6.35.4.2 navigationRouteBack

- a) 函数声明: `void navigationRouteBack();`
- b) 返回值: `void`
- c) 描述: 该函数触发路由队列中的最新一条数据, 并将其从队列中移除;

### 6.35.5 公有信号

#### 6.35.5.1 navigationRouterStateChanged

- a) 信号声明: `Q_SIGNAL void navigationRouterStateChanged (bool state);`
- b) 参数 1: `bool state // 路由状态;`
- c) 描述: 路由队列记录数从 0 到 1 或从 1 到 0 时, 触发该信号; `state` 为 `true` 表示路由跳转记录了第一条数据; `false` 表示队列中已无记录;

## 6.36 ElaPivot

### 6.36.1 组件说明

带指示器动画且可拖动的轴转导航组件, 支持指定指示器宽度和轴转间距。

### 6.36.2 属性成员

#### 6.36.2.1 TextPixelSize

- a) 类型: `int`
- b) 可用属性: `pTextPixelSize`
- c) 关联信号: `pTextPixelSizeChanged`
- d) 默认值: `20`
- e) 描述: 该属性控制组件字体的 `PixelSize` 大小;
- f) 对应的设置和获取函数:

```
void setTextPixelSize (int textPixelSize);
```

```
int getTextPixelSize () const;
```

#### 6.36.2.2 CurrentIndex

- a) 类型: `int`
- b) 可用属性: `pCurrentIndex`
- c) 关联信号: `pCurrentIndexChanged`
- d) 默认值: `-1`
- e) 描述: 该属性指示轴转组件的当前选项;

- f) 对应的设置和获取函数:

```
void setCurrentIndex (int currentIndex);  
int getCurrentIndex () const;
```

#### 6.36.2.3 PivotSpacing

- a) 类型: `int`
- b) 可用属性: `pPivotSpacing`
- c) 关联信号: `pPivotSpacingChanged`
- d) 默认值: `5`
- e) 描述: 该属性控制轴转组件的内部元素间距;
- f) 对应的设置和获取函数:

```
void setPivotSpacing (int pivotSpacing);  
int getPivotSpacing () const;
```

#### 6.36.2.4 MarkWidth

- a) 类型: `int`
- b) 可用属性: `pMarkWidth`
- c) 关联信号: `pMarkWidthChanged`
- d) 默认值: `40`
- e) 描述: 该属性控制选中指示器的宽度;
- f) 对应的设置和获取函数:

```
void setMarkWidth (int markWidth);  
int getMarkWidth () const;
```

### 6.36.3 公有函数

#### 6.36.3.1 appendPivot

- a) 函数声明: `void appendPivot (QString pivotTitle);`
- b) 参数 1: `int pivotTitle` // 新增的轴转元素标题;
- c) 返回值: `void`
- d) 描述: 该函数在轴转组件的最右侧新增一个元素, 元素标题为 `pivotTitle`;

#### 6.36.3.2 removePivot

- a) 函数声明: `void removePivot (QString pivotTitle);`

- b) 参数 1: `int pivotTitle` // 移除的轴转元素标题;
- c) 返回值: `void`
- d) 描述: 该函数从轴转组件的最右侧元素开始匹配, 删除第一个元素标题为 `pivotTitle` 的元素;

## 6.36.4 公有信号

### 6.36.4.1 pivotClicked

- a) 信号声明: `Q_SIGNAL void pivotClicked (int index);`
- b) 参数 1: `int index` // 被点击元素的索引;
- c) 描述: 轴转元素被点击时, 触发该信号;

### 6.36.4.2 pivotDoubleClicked

- a) 信号声明: `Q_SIGNAL void pivotDoubleClicked (int index);`
- b) 参数 1: `int index` // 被点击元素的索引;
- c) 描述: 轴转元素被双击时, 触发该信号;

## 6.37 ElaPlainTextEdit

### 6.37.1 组件说明

带焦点指示器的文本编辑框组件, 调用 API 与 QT 原生一致。

## 6.38 ElaPopularCard

### 6.38.1 组件说明

流行浮动式卡片, 鼠标覆盖时支持二级动态信息展示。

### 6.38.2 属性成员

#### 6.38.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `8`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:  
`void setBorderRadius (int borderRadius);`

```
int getBorderRadius () const;
```

#### 6.38.2.2 CardPixmap

- a) 类型: QPixmap
- b) 可用属性: pCardPixmap
- c) 关联信号: pCardPixmapChanged
- d) 默认值: 空
- e) 描述: 该属性为指定的自定义图片;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (QPixmap cardPixmap);  
QPixmap getCardPixmap () const;
```

#### 6.38.2.3 Title

- a) 类型: QString
- b) 可用属性: pTitle
- c) 关联信号: pTitleChanged
- d) 默认值: 空
- e) 描述: 该属性为主标题文字;
- f) 对应的设置和获取函数:

```
void setTitle (QString title);  
QString getTitle () const;
```

#### 6.38.2.4 SubTitle

- a) 类型: QString
- b) 可用属性: pSubTitle
- c) 关联信号: pSubTitleChanged
- d) 默认值: 空
- e) 描述: 该属性为副标题文字;
- f) 对应的设置和获取函数:

```
void setSubTitle (QString subTitle);  
QString getSubTitle () const;
```

#### 6.38.2.5 InteractiveTips

- a) 类型: QString
- b) 可用属性: pInteractiveTips

- c) 关联信号: `pInteractiveTipsChanged`
- d) 默认值: 空
- e) 描述: 该属性为右下侧互动提示文字;
- f) 对应的设置和获取函数:

```
void setInteractiveTips (QString interactiveTips);  
QString getInteractiveTips () const;
```

#### 6.38.2.6 DetailedText

- a) 类型: `QString`
- b) 可用属性: `pDetailedText`
- c) 关联信号: `pDetailedTextChanged`
- d) 默认值: 空
- e) 描述: 该属性为展开模式下的详细描述文字;
- f) 对应的设置和获取函数:

```
void setDetailedText (QString detailedText);  
QString getDetailedText () const;
```

#### 6.38.2.7 CardButtontext

- a) 类型: `QString`
- b) 可用属性: `pCardButtontext`
- c) 关联信号: `pCardButtontextChanged`
- d) 默认值: 空
- e) 描述: 该属性为展开模式下的按钮文字;
- f) 对应的设置和获取函数:

```
void setCardButtontext (QString cardButtontext);  
QString getCardButtontext () const;
```

#### 6.38.2.8 CardFloatArea

- a) 类型: `QWidget*`
- b) 可用属性: `pCardFloatArea`
- c) 关联信号: `pCardFloatAreaChanged`
- d) 默认值: `parentWidget()`
- e) 描述: 该属性为展开模式下的父窗口;
- f) 对应的设置和获取函数:

```
void setCardFloatArea (QWidget* cardFloatArea);
```

```
QWidget* getCardFloatArea () const;
```

#### 6.38.2.9 CardFloatPixmap

- a) 类型: QPixmap
- b) 可用属性: pCardFloatPixmap
- c) 关联信号: pCardFloatPixmapChanged
- d) 默认值: 空
- e) 描述: 该属性为展开模式下的额外展示图片;
- f) 对应的设置和获取函数:

```
void setCardFloatPixmap (QPixmap cardFloatPixmap);
```

```
QPixmap getCardFloatPixmap () const;
```

### 6.38.3 公有信号

#### 6.38.3.1 popularCardClicked

- a) 信号声明: Q\_SIGNAL void popularCardClicked ();
- b) 描述: 卡片被点击时, 触发该信号;

#### 6.38.3.2 popularCardButtonClicked

- c) 信号声明: Q\_SIGNAL void popularCardButtonClicked ();
- d) 描述: 展开模式下的按钮被点击时, 触发该信号;

## 6.39 ElaProgressBar

### 6.39.1 组件说明

带繁忙动画的进度条控件, 调用 API 与 QT 原生一致。

## 6.40 ElaPromotionCard

### 6.40.1 组件说明

带覆盖和点击动画的促销卡片控件, 支持保留图片原始纵横比的部分显示模式。

### 6.40.2 属性成员

#### 6.40.2.1 BorderRadius

- a) 类型: int
- b) 可用属性: pBorderRadius



- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `5`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

#### 6.40.2.2 CardPixmap

- a) 类型: `QPixmap`
- b) 可用属性: `pCardPixmap`
- c) 关联信号: `pCardPixmapChanged`
- d) 默认值: `空`
- e) 描述: 该属性为指定的自定义图片;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (QPixmap cardPixmap);  
QPixmap getCardPixmap () const;
```

#### 6.40.2.3 CardTitle

- a) 类型: `QString`
- b) 可用属性: `pCardTitle`
- c) 关联信号: `pCardTitleChanged`
- d) 默认值: `空`
- e) 描述: 该属性为卡片左上角的卡片标题文字;
- f) 对应的设置和获取函数:

```
void setCardTitle (QString cardTitle);  
QString getCardTitle () const;
```

#### 6.40.2.4 PromotionTitle

- a) 类型: `QString`
- b) 可用属性: `pPromotionTitle`
- c) 关联信号: `pPromotionTitleChanged`
- d) 默认值: `空`
- e) 描述: 该属性为卡片左侧带底色的促销标题文字;
- f) 对应的设置和获取函数:

```
void setPromotionTitle (QString promotionTitle);
```



```
QString getPromotionTitle () const;
```

#### 6.40.2.5 Title

- a) 类型: **QString**
- b) 可用属性: **pTitle**
- c) 关联信号: **pTitleChanged**
- d) 默认值: 空
- e) 描述: 该属性为卡片左侧主标题文字;
- f) 对应的设置和获取函数:

```
void setTitle (QString title);
```

```
QString getTitle () const;
```

#### 6.40.2.6 SubTitle

- a) 类型: **QString**
- b) 可用属性: **pSubTitle**
- c) 关联信号: **pSubTitleChanged**
- d) 默认值: 空
- e) 描述: 该属性为卡片左侧副标题文字;
- f) 对应的设置和获取函数:

```
void setSubTitle (QString subTitle);
```

```
QString getSubTitle () const;
```

#### 6.40.2.7 CardTitleColor

- a) 类型: **QColor**
- b) 可用属性: **pCardTitleColor**
- c) 关联信号: **pCardTitleColorChanged**
- d) 默认值: **Qt::white**
- e) 描述: 该属性为卡片左上角的卡片标题文字颜色;
- f) 对应的设置和获取函数:

```
void setCardTitleColor (QColor cardTitleColor);
```

```
QColor getCardTitleColor () const;
```

#### 6.40.2.8 PromotionTitleColor

- a) 类型: **QColor**
- b) 可用属性: **pPromotionTitleColor**

- c) 关联信号: `pPromotionTitleColorChanged`
- d) 默认值: `Qt::white`
- e) 描述: 该属性为卡片左侧带底色的促销标题文字颜色;
- f) 对应的设置和获取函数:

```
void setPromotionTitleColor (QColor promotionTitleColor);  
QColor getPromotionTitleColor () const;
```

#### 6.40.2.9 PromotionTitleBaseColor

- a) 类型: `QColor`
- b) 可用属性: `pPromotionTitleBaseColor`
- c) 关联信号: `pPromotionTitleBaseColorChanged`
- d) 默认值: `QColor(0, 0, 0, 120);`
- e) 描述: 该属性为卡片左侧带底色的促销标题文字底色;
- f) 对应的设置和获取函数:

```
void setPromotionTitleBaseColor (QColor promotionTitleBaseColor);  
QColor getPromotionTitleBaseColor () const;
```

#### 6.40.2.10 TitleColor

- a) 类型: `QColor`
- b) 可用属性: `pTitleColor`
- c) 关联信号: `pTitleColorChanged`
- d) 默认值: `Qt::white`
- e) 描述: 该属性为卡片左侧主标题文字颜色;
- f) 对应的设置和获取函数:

```
void setTitleColor (QColor titleColor);  
QColor getTitleColor () const;
```

#### 6.40.2.11 SubTitleColor

- a) 类型: `QColor`
- b) 可用属性: `pSubTitleColor`
- c) 关联信号: `pSubTitleColorChanged`
- d) 默认值: `Qt::white`
- e) 描述: 该属性为卡片左侧副标题文字颜色;
- f) 对应的设置和获取函数:

```
void setSubTitleColor (QColor subTitleColor);
```

`QColor getSubTitleColor () const;`

#### 6.40.2.12 CardTitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pCardTitlePixelSize`
- c) 关联信号: `pCardTitlePixelSizeChanged`
- d) 默认值: `22`
- e) 描述: 该属性为卡片左上角的卡片标题文字尺寸;
- f) 对应的设置和获取函数:

`void setCardTitlePixelSize (int cardTitlePixelSize);`

`int getCardTitlePixelSize () const;`

#### 6.40.2.13 PromotionTitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pPromotionTitlePixelSize`
- c) 关联信号: `pPromotionTitlePixelSizeChanged`
- d) 默认值: `12`
- e) 描述: 该属性为卡片左侧带底色的促销标题文字尺寸;
- f) 对应的设置和获取函数:

`void setPromotionTitlePixelSize (int promotionTitlePixelSize);`

`int getPromotionTitlePixelSize () const;`

#### 6.40.2.14 TitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pTitlePixelSize`
- c) 关联信号: `pTitlePixelSizeChanged`
- d) 默认值: `25`
- e) 描述: 该属性为卡片左侧主标题文字尺寸;
- f) 对应的设置和获取函数:

`void setTitlePixelSize (int titlePixelSize);`

`int getTitlePixelSize () const;`

#### 6.40.2.15 SubTitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pSubTitlePixelSize`

- c) 关联信号: `pSubTitlePixelSizeChanged`
- d) 默认值: `16`
- e) 描述: 该属性为卡片左侧副标题文字尺寸;
- f) 对应的设置和获取函数:

```
void setSubTitlePixelSize (int subTitlePixelSize);  
int getSubTitlePixelSize () const;
```

#### 6.40.2.16 HorizontalCardPixmapRatio

- a) 类型: `qreal`
- b) 可用属性: `pHorizontalCardPixmapRatio`
- c) 关联信号: `pHorizontalCardPixmapRatioChanged`
- d) 默认值: `1`
- e) 描述: 该属性控制卡片图片横向的整体显示尺寸, 最大为 1, 最小为 0, 值越小显示的横向部分越少, 显示区域从左右两边开始减少;
- f) 对应的设置和获取函数:

```
void setHorizontalCardPixmapRatio (qreal horizontalCardPixmapRatio);  
qreal getHorizontalCardPixmapRatio () const;
```

#### 6.40.2.17 VerticalCardPixmapRatio

- a) 类型: `qreal`
- b) 可用属性: `pVerticalCardPixmapRatio`
- c) 关联信号: `pVerticalCardPixmapRatioChanged`
- d) 默认值: `1`
- e) 描述: 该属性控制卡片图片纵向的整体显示尺寸, 最大为 1, 最小为 0, 值越小显示的纵向部分越少, 显示区域从上下两侧开始减少;
- f) 对应的设置和获取函数:

```
void setVerticalCardPixmapRatio (qreal verticalCardPixmapRatio);  
qreal getVerticalCardPixmapRatio () const;
```

### 6.40.3 公有信号

#### 6.40.3.1 promotionCardClicked

- a) 信号声明: `Q_SIGNAL void promotionCardClicked ();`
- b) 描述: 卡片被点击时, 触发该信号;

## 6.41 ElaPromotionView

### 6.41.1 组件说明

用于存放和轮流展示 ElaPromotionCard 卡片的轮播视图。

### 6.41.2 属性成员

#### 6.41.2.1 CardExpandWidth

- a) 类型: `int`
- b) 可用属性: `pCardExpandWidth`
- c) 关联信号: `pCardExpandWidthChanged`
- d) 默认值: `600`
- e) 描述: 该属性控制主要展示卡片的宽度;
- f) 对应的设置和获取函数:

```
void setCardExpandWidth (int cardExpandWidth);  
int getCardExpandWidth () const;
```

#### 6.41.2.2 CardCollapseWidth

- a) 类型: `int`
- b) 可用属性: `pCardExpandWidth`
- c) 关联信号: `pCardExpandWidthChanged`
- d) 默认值: `300`
- e) 描述: 该属性控制次要展示卡片的宽度;
- f) 对应的设置和获取函数:

```
void setCardCollapseWidth (int cardCollapseWidth);  
int getCardCollapseWidth () const;
```

#### 6.41.2.3 CurrentIndex

- a) 类型: `int`
- b) 可用属性: `pCurrentIndex`
- c) 关联信号: `pCurrentIndexChanged`
- d) 默认值: `0`
- e) 描述: 该属性为当前展示的卡片索引;
- f) 对应的设置和获取函数:

```
void setCurrentIndex (int currentIndex);
```

```
int getCurrentIndex () const;
```

#### 6.41.2.4 IsAutoScroll

- a) 类型: `bool`
- b) 可用属性: `pIsAutoScroll`
- c) 关联信号: `pIsAutoScrollChanged`
- d) 默认值: `false`
- e) 描述: 该属性控制轮播视图是否自动滚动, 方向为从右向左;
- f) 对应的设置和获取函数:

```
void setIsAutoScroll (bool isAutoScroll);
```

```
bool getIsAutoScroll () const;
```

#### 6.41.2.5 AutoScrollInterval

- a) 类型: `int`
- b) 可用属性: `pAutoScrollInterval`
- c) 关联信号: `pAutoScrollIntervalChanged`
- d) 默认值: `5000`
- e) 描述: 该属性控制轮播视图自动滚动的间隔, 单位为 `ms`, 最小为 `400ms`;
- f) 对应的设置和获取函数:

```
void setAutoScrollInterval (int autoScrollInterval);
```

```
int getAutoScrollInterval () const;
```

### 6.41.3 公有函数

#### 6.41.3.1 appendPromotionCard

- a) 函数声明: `void appendPromotionCard(ElaPromotionCard* card);`
- b) 参数 1: `ElaPromotionCard* card` // 促销卡片;
- c) 返回值: `void`
- d) 描述: 该函数为轮播视图新增一张展示卡片;

### 6.42 ElaPushButton

#### 6.42.1 组件说明

按钮组件, 支持自定义三态颜色 (底色、覆盖颜色、按下颜色)。

## 6.42.2 属性成员

### 6.42.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `3`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

### 6.42.2.2 LightDefaultColor

- a) 类型: `QColor`
- b) 可用属性: `pLightDefaultColor`
- c) 关联信号: `pLightDefaultColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件浅色主题下的颜色;
- f) 对应的设置和获取函数:

```
void setLightDefaultColor (int lightDefaultColor);  
int getLightDefaultColor () const;
```

### 6.42.2.3 DarkDefaultColor

- a) 类型: `QColor`
- b) 可用属性: `pDarkDefaultColor`
- c) 关联信号: `pDarkDefaultColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件深色主题下的颜色;
- f) 对应的设置和获取函数:

```
void setDarkDefaultColor (int darkDefaultColor);  
int getDarkDefaultColor () const;
```

### 6.42.2.4 LightHoverColor

- a) 类型: `QColor`
- b) 可用属性: `pLightHoverColor`



- c) 关联信号: `pLightHoverColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件浅色主题下鼠标覆盖时的颜色;
- f) 对应的设置和获取函数:

```
void setLightHoverColor (QColor lightHoverColor);  
QColor getLightHoverColor () const;
```

#### 6.42.2.5 DarkHoverColor

- a) 类型: `QColor`
- b) 可用属性: `pDarkHoverColor`
- c) 关联信号: `pDarkHoverColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件深色主题下鼠标覆盖时的颜色;
- f) 对应的设置和获取函数:

```
void setDarkHoverColor (QColor darkHoverColor);  
QColor getDarkHoverColor () const;
```

#### 6.42.2.6 LightPressColor

- a) 类型: `QColor`
- b) 可用属性: `pLightPressColor`
- c) 关联信号: `pLightPressColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件浅色主题下鼠标按下时的颜色;
- f) 对应的设置和获取函数:

```
void setLightPressColor (QColor lightPressColor);  
QColor getLightPressColor () const;
```

#### 6.42.2.7 DarkPressColor

- a) 类型: `QColor`
- b) 可用属性: `pDarkPressColor`
- c) 关联信号: `pDarkPressColorChanged`
- d) 默认值: 由 `ElaTheme` 决定
- e) 描述: 该属性控制组件深色主题下鼠标按下时的颜色;
- f) 对应的设置和获取函数:

```
void setDarkPressColor (QColor darkPressColor);
```

```
QColor getDarkPressColor () const;
```

### 6.42.3 公有函数

#### 6.42.3.1 setLightTextColor

- a) 函数声明: `void setLightTextColor(QColor color);`
- b) 参数 1: `QColor color` // 浅色主题下文字颜色;
- c) 返回值: `void`
- d) 描述: 该函数设置浅色主题下文字颜色;

#### 6.42.3.2 getLightTextColor

- a) 函数声明: `QColor getLightTextColor();`
- b) 返回值: `QColor` // 浅色主题下文字颜色;
- c) 描述: 该函数获取浅色主题下文字颜色;

#### 6.42.3.3 setDarkTextColor

- a) 函数声明: `void setDarkTextColor(QColor color);`
- b) 参数 1: `QColor color` // 深色主题下文字颜色;
- c) 返回值: `void`
- d) 描述: 该函数设置深色主题下文字颜色;

#### 6.42.3.4 getDarkTextColor

- a) 函数声明: `QColor getDarkTextColor();`
- b) 返回值: `QColor` // 深色主题下文字颜色;
- c) 描述: 该函数获取深色主题下文字颜色;

## 6.43 ElaRadioButton

### 6.43.1 组件说明

单选按钮组件, 调用 API 与 QT 原生一致。

## 6.44 ElaReminderCard

### 6.44.1 组件说明

带图片的提醒卡片组件。

## 6.44.2 属性成员

### 6.44.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `5`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);  
int getBorderRadius () const;
```

### 6.44.2.2 Title

- a) 类型: `QString`
- b) 可用属性: `pTitle`
- c) 关联信号: `pTitleChanged`
- d) 默认值: `空`
- e) 描述: 该属性为主标题文字;
- f) 对应的设置和获取函数:

```
void setTitle (QString title);  
QString getTitle () const;
```

### 6.44.2.3 SubTitle

- a) 类型: `QString`
- b) 可用属性: `pSubTitle`
- c) 关联信号: `pSubTitleChanged`
- d) 默认值: `空`
- e) 描述: 该属性为副标题文字;
- f) 对应的设置和获取函数:

```
void setSubTitle (QString subTitle);  
QString getSubTitle () const;
```

### 6.44.2.4 TitlePixelSize

- a) 类型: `int`

- b) 可用属性: `pTitlePixelSize`
- c) 关联信号: `pTitlePixelSizeChanged`
- d) 默认值: `15`
- e) 描述: 该属性为绘制主标题时, QPainter 的 `pixelSize` 大小;
- f) 对应的设置和获取函数:  
`void setTitlePixelSize (int titlePixelSize);`  
`int getTitlePixelSize () const;`

#### 6.44.2.5 SubTitlePixelSize

- a) 类型: `int`
- b) 可用属性: `pSubTitlePixelSize`
- c) 关联信号: `pSubTitlePixelSizeChanged`
- d) 默认值: `12`
- e) 描述: 该属性为绘制副标题时, QPainter 的 `pixelSize` 大小;
- f) 对应的设置和获取函数:  
`void setSubTitlePixelSize (int subTitlePixelSize);`  
`int getSubTitlePixelSize () const;`

#### 6.44.2.6 TitleSpacing

- a) 类型: `int`
- b) 可用属性: `pTitleSpacing`
- c) 关联信号: `pTitleSpacingChanged`
- d) 默认值: `3`
- e) 描述: 该属性为文字距组件纵向中心点的间隔;
- f) 对应的设置和获取函数:  
`void setTitleSpacing (int titleSpacing);`  
`int getTitleSpacing () const;`

#### 6.44.2.7 CardPixmap

- a) 类型: `QPixmap`
- b) 可用属性: `pCardPixmap`
- c) 关联信号: `pCardPixmapChanged`
- d) 默认值: `空`
- e) 描述: 该属性为指定的自定义图片;
- f) 对应的设置和获取函数:

```
void setTitleSpacing (QPixmap cardPixmap);  
QPixmap getCardPixmap () const;
```

#### 6.44.2.8 CardPixmapSize

- a) 类型: QSize
- b) 可用属性: pCardPixmapSize
- c) 关联信号: pCardPixmapSizeChanged
- d) 默认值: QSize(54 , 54)
- e) 描述: 该属性控制自定义图片的绘制尺寸;
- f) 对应的设置和获取函数:

```
void setCardPixmapSize (QSize cardPixmapSize);  
QSize getCardPixmapSize () const;
```

#### 6.44.2.9 CardPixmapBorderRadius

- a) 类型: int
- b) 可用属性: pCardPixmapBorderRadius
- c) 关联信号: pCardPixmapBorderRadiusChanged
- d) 默认值: 6
- e) 描述: 该属性控制自定义图片外围边框的圆角半径, 值越大圆角越明显; 仅在 pCardPixMode 属性为 ElaCardPixType::RoundedRect 时生效;
- f) 对应的设置和获取函数:

```
void setCardPixmapBorderRadius (int cardPixmapBorderRadius);  
int getCardPixmapBorderRadius () const;
```

#### 6.44.2.10 CardPixMode

- a) 类型: ElaCardPixType::PixMode
- b) 可用属性: pCardPixMode
- c) 关联信号: pCardPixModeChanged
- d) 默认值: ElaCardPixType::PixMode::Default
- e) 描述: 该属性控制自定义图片的显示模式;
- f) 对应的设置和获取函数:

```
void setCardPixMode (ElaCardPixType::PixMode cardPixMode);  
ElaCardPixType::PixMode getCardPixMode () const;
```

### 6.44.3 公有函数

#### 6.44.3.1 setCardPixmapSize

- e) 函数声明: `void setCardPixmapSize (int width, int height);`
- f) 参数 1: `int width` // 自定义图片宽度;
- g) 参数 2: `int height` // 自定义图片高度;
- h) 返回值: `void`
- i) 描述: 该函数控制自定义图片的绘制尺寸, 与直接设置 `pCardPixmapSize` 属性效果一致;
- j) 使用示例: `setCardPixmapSize(50 , 50);` // 设置图片尺寸为宽 50, 高 50;

### 6.45 ElaScrollArea

#### 6.45.1 组件说明

支持鼠标按住拖动的滚动区域组件, 调用 API 与 QT 原生一致。

#### 6.45.2 公有函数

##### 6.45.2.1 setIsGrabGesture

- a) 函数声明: `void setIsGrabGesture (bool isEnabled, qreal mousePressEventDelay = 0.5);`
- b) 参数 1: `bool isEnabled` // 启用鼠标拖动;
- c) 参数 2: `qreal mousePressEventDelay` // 启用鼠标拖动后, `MouseEvent` 事件会有一定延迟, 该参数决定延迟的时间, 若不需要延迟, 设置为 0 即可;
- d) 返回值: `void`
- e) 描述: 该函数控制滚动区域是否启用鼠标拖动模式; 横向和纵向的拖动过冲模式, 即允许拖动视图超过极限;

##### 6.45.2.2 setIsOverShoot

- a) 函数声明: `void setIsOverShoot (Qt::Orientation orientation, bool isEnabled);`
- b) 参数 1: `Qt::Orientation orientation` // 鼠标拖动过冲方向;
- c) 参数 2: `bool isEnabled` // 启用鼠标拖动过冲;
- d) 返回值: `void`
- e) 描述: 该函数控制是否启用指定方向的拖动过冲模式, 即允许拖动视图超过极限;

### 6.45.2.3 getIsOverShoot

- a) 函数声明: `bool getIsOverShoot (Qt::Orientation orientation) const;`
- b) 参数 1: `Qt::Orientation orientation` // 鼠标拖动过冲方向;
- c) 返回值: `bool`
- d) 描述: 该函数获取是否启用指定方向的拖动过冲模式;

### 6.45.2.4 setIsAnimation

- a) 函数声明: `void setIsAnimation (Qt::Orientation orientation, bool isAnimation) const;`
- b) 参数 1: `Qt::Orientation orientation` // 滚动条方向;
- c) 参数 2: `bool isAnimation` // 是否启用动画;
- d) 返回值: `void`
- e) 描述: 该函数控制是否启用指定方向滚动条的范围变化动画;

### 6.45.2.5 getIsAnimation

- a) 函数声明: `bool getIsGrabGesture (Qt::Orientation orientation) const;`
- b) 参数 1: `Qt::Orientation orientation` // 滚动条方向;
- c) 返回值: `bool`
- d) 描述: 该函数获取是否启用指定方向滚动条的范围变化动画;

## 6.46 ElaScrollBar

### 6.46.1 组件说明

支持平滑滚动和展开动画的滚动条组件, 调用 API 与 QT 原生一致。

### 6.46.2 公有信号

#### 6.46.2.1 rangeAnimationFinished

- a) 信号声明: `Q_SIGNAL void rangeAnimationFinished ();`
- b) 描述: 滚动条范围变化动画结束时, 触发该信号, 在某些组件上 (如 `TreeView`), 需要捕获该信号进行重绘;

## 6.47 ElaScrollPage

### 6.47.1 组件说明

支持内部堆栈页面、路由跳转、面包屑导航的滚动窗口组件。



## 6.47.2 属性成员

### 6.47.2.1 CustomWidget

- a) 类型: `QWidget*`
- b) 可用属性: `pCustomWidget`
- c) 关联信号: `pCustomWidgetChanged`
- d) 默认值: `nullptr`
- e) 描述: 该属性为页面上方锚定的自定义窗口, 位于面包屑导航栏之下, 可设置为任意基于 `QWidget` 实现的窗口。
- f) 对应的设置和获取函数:

```
void setCustomWidget (QWidget* customWidget);
```

```
QWidget* getCustomWidget () const;
```

## 6.47.3 公有函数

### 6.47.3.1 addCentralWidget

- a) 函数声明: `void addCentralWidget (QWidget* centralWidget, bool isWidgetResizable = true, bool isVerticalGrabGesture = true, qreal mousePressEventDelay = 0.5);`
- b) 参数 1: `QWidget* centralWidget` // 需要新增的中心窗口;
- c) 参数 2: `bool isWidgetResizable = true` // 内部滚动区域是否自适应大小;
- d) 参数 3: `bool isVerticalGrabGesture = true` // 是否启用纵向滚动过冲;
- e) 参数 4: `qreal mousePressEventDelay` // 启用鼠标拖动后, `MouseEvent` 事件会有一定延迟, 该参数决定延迟的时间, 若不需要延迟, 设置为 0 即可;
- f) 返回值: `void`
- g) 描述: 该函数为滚动窗口内部堆栈新增一个窗口, 该窗口的 `windowTitle` 作为面包屑导航的标题, 该命名不应重复; 是否自适应大小由 `isWidgetResizable` 指定, 是否启用纵向滚动过冲由 `isVerticalGrabGesture` 指定, 鼠标按下事件的延迟时间由 `mousePressEventDelay` 指定;

### 6.47.3.2 navigation

- a) 函数声明: `void navigation (int widgetIndex, bool isLogRoute = true);`
- b) 参数 1: `int widgetIndex` // 中心窗口索引;
- c) 参数 2: `bool isLogRoute = true` // 是否记录路由跳转;
- d) 返回值: `void`

- e) 描述：该函数根据指定的窗口索引在滚动窗口内部进行界面跳转，是否记录跳转到路由导航由 `isLogRoute` 指定；

#### 6.47.3.3 setPageTitleSpacing

- a) 函数声明：`void setPageTitleSpacing (int spacing);`
- b) 参数 1：`int spacing` // 面包屑导航栏最左侧距离边框的间隔；
- c) 返回值：`void`
- d) 描述：该函数设置面包屑导航栏最左侧距离边框的间隔；

#### 6.47.3.4 getPageTitleSpacing

- a) 函数声明：`int getPageTitleSpacing () const;`
- b) 返回值：`int` // 面包屑导航栏最左侧距离边框的间隔；
- c) 描述：该函数获取面包屑导航栏最左侧距离边框的间隔；

#### 6.47.3.5 setTitleVisible

- a) 函数声明：`void setTitleVisible (int isVisible);`
- b) 参数 1：`int isVisible` // 面包屑导航栏可见性；
- c) 返回值：`void`
- d) 描述：该函数设置面包屑导航栏的可见性；

### 6.48 ElaScrollPageArea

#### 6.48.1 组件说明

在滚动页面中使用的背景区域组件。

#### 6.48.2 属性成员

##### 6.48.2.1 BorderRadius

- a) 类型：`int`
- b) 可用属性：`pBorderRadius`
- c) 关联信号：`pBorderRadiusChanged`
- d) 默认值：`5`
- e) 描述：该属性控制组件外围边框的圆角半径，值越大圆角越明显；
- f) 对应的设置和获取函数：

`void setBorderRadius (int borderRadius);`

`int getBorderRadius () const;`

## 6.49 ElaSlider

### 6.49.1 组件说明

滑动条组件，支持鼠标点击滑轨拖动，调用 API 与 QT 原生一致。

## 6.50 ElaSpinBox

### 6.50.1 组件说明

微调框组件，调用 API 与 QT 原生一致。

## 6.51 ElaStatusBar

### 6.51.1 组件说明

状态栏组件，调用 API 与 QT 原生一致。

## 6.52 ElaSuggestBox

### 6.52.1 组件说明

自动建议框组件，根据输入内容在指定的数据中进行搜索并展示。

### 6.52.2 属性成员

#### 6.52.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `6`
- e) 描述: 该属性控制组件外围边框的圆角半径，值越大圆角越明显；
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);
```

```
int getBorderRadius () const;
```

#### 6.52.2.2 CaseSensitivity

- a) 类型: `Qt::CaseSensitivity`
- b) 可用属性: `pCaseSensitivity`
- c) 关联信号: `pCaseSensitivityChanged`
- d) 默认值: `Qt::CaseInsensitive`

- e) 描述：该属性控制检索数据的规则，默认为不区分大小写；
- f) 对应的设置和获取函数：

```
void setCaseSensitivity (Qt::CaseSensitivity caseSensitivity);  
Qt::CaseSensitivity getCaseSensitivity () const;
```

### 6.52.3 公有函数

#### 6.52.3.1 setPlaceholderText

- a) 函数声明： `void setPlaceholderText(const QString& placeholderText);`
- b) 参数 1： `const QString& placeholderText` // 输入框的提示文字；
- c) 返回值： `void`
- d) 描述：该函数设置输入框内的提示文字；

#### 6.52.3.2 addSuggestion

- a) 函数声明： `void addSuggestion(const QString& suggestText, const QVariantMap & suggestData = {});`
- b) 参数 1： `const QString& suggestText` // 数据标题；
- c) 参数 2： `const QVariantMap& suggestData` // 数据内容；
- d) 返回值： `void`
- e) 描述：该函数新增一条检索数据，数据标题为 `suggestText`，数据内容为 `suggestData`；

#### 6.52.3.3 addSuggestion

- a) 函数声明： `void addSuggestion(ElaIconType::IconName icon, const QString& suggestText, const QVariantMap& suggestData = {});`
- b) 参数 1： `ElaIconType::IconName icon` // 数据图标；
- c) 参数 2： `const QString& suggestText` // 数据标题；
- d) 参数 3： `const QVariantMap& suggestData` // 数据内容；
- e) 返回值： `void`
- f) 描述：该函数新增一条检索数据，数据图标为 `icon`，数据标题为 `suggestText`，数据内容为 `suggestData`；

#### 6.52.3.4 removeSuggestion

- a) 函数声明： `void addSuggestion(const QString& suggestText);`
- b) 参数 2： `const QString& suggestText` // 数据标题；

- c) 返回值: `void`
- d) 描述: 该函数移除所有标题为 `suggestData` 的检索数据;

#### 6.52.3.5 removeSuggestion

- a) 函数声明: `void addSuggestion(int index);`
- b) 参数 2: `int index` // 数据索引;
- c) 返回值: `void`
- d) 描述: 该函数移除索引为 `index` 的检索数据;

### 6.52.4 公有信号

#### 6.52.4.1 suggestionClicked

- a) 信号声明: `Q_SIGNAL void suggestionClicked (QString suggestText, QVariantMap suggestData);`
- b) 参数 1: `QString suggestText` // 数据标题;
- c) 参数 2: `QVariantMap suggestData` // 数据内容;
- d) 描述: 搜索项被点击时, 触发该信号;

## 6.53 ElaTabBar

### 6.53.1 组件说明

标签页组件, 调用 API 与 QT 原生一致。

## 6.54 ElaTableView

### 6.54.1 组件说明

表格视图组件, 调用 API 与 QT 原生一致。

### 6.54.2 属性成员

#### 6.54.2.1 HeaderMargin

- a) 类型: `int`
- b) 可用属性: `pHeaderMargin`
- c) 关联信号: `pHeaderMarginChanged`
- d) 默认值: `6`
- e) 描述: 该属性控制表格表头的 `Margin` 属性, 值越大表头越大;
- f) 对应的设置和获取函数:

```
void setHeaderMargin (int headerMargin);  
int getHeaderMargin () const;
```

### 6.54.3 公有信号

#### 6.54.3.1 tableViewShow

- a) 信号声明: `Q_SIGNAL void tableViewShow ();`
- b) 描述: 表格执行 ShowEvent 时, 触发该信号;

#### 6.54.3.2 tableViewHide

- c) 信号声明: `Q_SIGNAL void tableViewHide ();`
- d) 描述: 表格执行 HideEvent 时, 触发该信号;

## 6.55 ElaTabWidget

### 6.55.1 组件说明

标签页面组件, 调用 API 与 QT 原生一致; 需注意, 该组件仅支持横向 Tab 页模式。

## 6.56 ElaText

### 6.56.1 组件说明

便携文字组件, 根据主题自动变更颜色。

### 6.56.2 枚举定义

#### 6.56.2.1 ElaTextType::TextStyle

该枚举用于提示文字尺寸;

```
enum TextStyle  
{  
    NoStyle = 0x0000, // 无类型;  
    Caption = 0x0001, // 12px;  
    Body = 0x0002, // 13px;  
    BodyStrong = 0x0003, // 13px DemiBold;  
    Subtitle = 0x0004, // 20px DemiBold;  
    Title = 0x0005, // 28px DemiBold;  
    TitleLarge = 0x0006, // 40px DemiBold;  
    Display = 0x0007, // 48px DemiBold;
```

```
};
```

### 6.56.3 公有函数

#### 6.56.3.1 setIsWrapAnywhere

- a) 函数声明: `void setIsWrapAnywhere(bool isWrapAnywhere);`
- b) 参数 1: `bool isWrapAnywhere` // 是否启用任意换行;
- c) 返回值: `void`
- d) 描述: 该函数启用换行模式, 且不考虑单词的完整性, 若需要完整换行, 直接调用原生方法 `setWordWrap` 即可;

#### 6.56.3.2 getIsWrapAnywhere

- a) 函数声明: `bool getIsWrapAnywhere() const;`
- b) 返回值: `bool` // 是否启用任意换行
- c) 描述: 该函数获取是否启用换行模式;

#### 6.56.3.3 setTextPixelSize

- a) 函数声明: `void setTextPixelSize (int size);`
- b) 参数 1: `int size` // 文字的 `pixelSize` 尺寸;
- c) 返回值: `void`
- d) 描述: 该函数设置文字的 `pixelSize` 尺寸;

#### 6.56.3.4 getTextPixelSize

- a) 函数声明: `int getTextPixelSize() const;`
- b) 返回值: `int` // 文字的 `pixelSize` 尺寸;
- c) 描述: 该函数获取文字的 `pixelSize` 尺寸;

#### 6.56.3.5 setTextPointSize

- a) 函数声明: `void setTextPointSize (int size);`
- b) 参数 1: `int size` // 文字的 `pointSize` 尺寸;
- c) 返回值: `void`
- d) 描述: 该函数设置文字的 `pointSize` 尺寸;

#### 6.56.3.6 getTextPointSize

- a) 函数声明: `int getTextPointSize () const;`



- b) 返回值: `int` // 文字的 `pointSize` 尺寸;
- c) 描述: 该函数获取文字的 `pointSize` 尺寸;

#### 6.56.3.7 setTextStyle

- a) 函数声明: `void setTextStyle (ElaTextType::TextStyle textStyle);`
- b) 参数 1: `ElaTextType::TextStyle textStyle` // 文字尺寸类型;
- c) 返回值: `void`
- d) 描述: 该函数使用预设宏设置文字的 `pixelSize` 尺寸;

#### 6.56.3.8 getTextStyle

- a) 函数声明: `ElaTextType::TextStyle getTextStyle () const;`
- b) 返回值: `ElaTextType::TextStyle textStyle` // 文字尺寸类型;
- c) 描述: 该函数返回使用的预设宏类型;

### 6.57 ElaTheme

#### 6.57.1 组件说明

单例类, 用于主题颜色的控制, 所有组件的颜色在此进行统一管理。

#### 6.57.2 枚举定义

##### 6.57.2.1 ElaThemeType:: ThemeMode

该枚举用于提示主题类型;

```
enum ThemeMode
{
    Light= 0x0000, // 浅色主题;
    Dark= 0x0001, // 深色主题;
};
```

#### 6.57.3 公有函数

##### 6.57.3.1 setThemeMode

- a) 函数声明: `void setThemeMode (ElaThemeType::ThemeMode themeMode);`
- b) 参数 1: `ElaThemeType::ThemeMode themeMode` // 主题类型;
- c) 返回值: `void`
- d) 描述: 该函数设置项目的主题类型;

### 6.57.3.2 getThemeMode

- a) 函数声明: `ElaThemeType::ThemeMode getThemeMode () const;`
- b) 返回值: `ElaThemeType::ThemeMode themeMode` // 主题类型;
- c) 描述: 该函数获取项目的主题类型;

### 6.57.3.3 drawEffectShadow

- a) 函数声明: `void drawEffectShadow (QPainter* painter, QRect widgetRect, int shadowBorderWidth, int borderRadius);`
- b) 参数 1: `QPainter* painter` // 画笔指针;
- c) 参数 2: `QRect widgetRect` // 绘制范围;
- d) 参数 3: `int shadowBorderWidth` // 阴影宽度
- e) 参数 4: `int borderRadius` // 圆角半径;
- f) 返回值: `void`
- g) 描述: 该函数在指定的区域绘制一块阴影;

### 6.57.3.4 getThemeColor

- a) 函数声明: `const QColor& getThemeColor (ElaThemeType::ThemeMode themeMode, ElaThemeType::ThemeColor themeColor);`
- b) 参数 1: `ElaThemeType::ThemeMode themeMode` // 主题类型;
- c) 参数 2: `ElaThemeType::ThemeColor themeColor` // 详细的主题颜色, 由于样类过多, 这里不作列举;
- d) 返回值: `const QColor&` // 组件颜色;
- e) 描述: 该函数在指定的区域绘制一块阴影;

## 6.57.4 公有信号

### 6.57.4.1 themeModeChanged

- a) 信号声明: `Q_SIGNAL void themeModeChanged (ElaThemeType::ThemeMode themeMode);`
- b) 参数 1: `ElaThemeType::ThemeMode themeMode` // 主题类型;
- c) 描述: 项目主题改变时, 触发该信号;

## 6.58 ElaToggleButton

### 6.58.1 组件说明

带渐变动画的开关按钮组件。

### 6.58.2 属性成员

#### 6.58.2.1 BorderRadius

- a) 类型: `int`
- b) 可用属性: `pBorderRadius`
- c) 关联信号: `pBorderRadiusChanged`
- d) 默认值: `3`
- e) 描述: 该属性控制组件外围边框的圆角半径, 值越大圆角越明显;
- f) 对应的设置和获取函数:

```
void setBorderRadius (int borderRadius);
```

```
int getBorderRadius () const;
```

#### 6.58.2.2 Text

- a) 类型: `QString`
- b) 可用属性: `pText`
- c) 关联信号: `pTextChanged`
- d) 默认值: 空
- e) 描述: 该属性为按钮文字;
- f) 对应的设置和获取函数:

```
void setText (QString text);
```

```
QString getText () const;
```

### 6.58.3 公有函数

#### 6.58.3.1 setIsToggled

- a) 函数声明: `void setIsToggled (bool isToggled);`
- b) 参数 1: `bool isToggled` // 是否启用;
- c) 返回值: `void`
- d) 描述: 该函数设置按钮是否处于启用状态;

### 6.58.3.2 getIsToggled

- a) 函数声明: `bool getIsToggled () const;`
- b) 返回值: `bool isToggled` // 是否启用;
- c) 描述: 该函数获取按钮是否处于启用状态;

## 6.58.4 公有信号

### 6.58.4.1 toggled

- a) 信号声明: `Q_SIGNAL void toggled (bool checked);`
- b) 参数 1: `bool checked` // 启用状态;
- c) 描述: 按钮的启用状态改变时, 触发该信号;

## 6.59 ElaSwitchButton

### 6.59.1 组件说明

带渐变动画的切换按钮组件。

### 6.59.2 公有函数

#### 6.59.2.1 setIsToggled

- a) 函数声明: `void setIsToggled (bool isToggled);`
- b) 参数 1: `bool isToggled` // 是否启用;
- c) 返回值: `void`
- d) 描述: 该函数设置按钮是否处于启用状态;

#### 6.59.2.2 getIsToggled

- a) 函数声明: `bool getIsToggled () const;`
- b) 返回值: `bool isToggled` // 是否启用;
- c) 描述: 该函数获取按钮是否处于启用状态;

### 6.59.3 公有信号

#### 6.59.3.1 toggled

- a) 信号声明: `Q_SIGNAL void toggled (bool checked);`
- b) 参数 1: `bool checked` // 启用状态;
- c) 描述: 按钮的启用状态改变时, 触发该信号;

## 6.60 ElaToolBar

### 6.60.1 组件说明

工具栏组件，支持自定义图标，调用 API 与 QT 原生一致。

### 6.60.2 公有函数

#### 6.60.2.1 setToolBarSpacing

- a) 函数声明: `void setToolBarSpacing (int spacing);`
- b) 参数 1: `int spacing` // 工具栏间隔;
- c) 返回值: `void`
- d) 描述: 该函数设置工具栏间元素的间隔;

#### 6.60.2.2 setToolBarSpacing

- a) 函数声明: `int setToolBarSpacing ();`
- b) 返回值: `int` // 工具栏间隔;
- c) 描述: 该函数获取工具栏间元素的间隔;

#### 6.60.2.3 addElaIconAction

- a) 函数声明: `QAction* addElaIconAction (ElaIconType::IconName icon, const QString& text);`
- b) 参数 1: `ElaIconType::IconName icon` // 元素图标;
- c) 参数 2: `const QString& text` // 元素文字;
- d) 返回值: `QAction*` // 元素指针;
- e) 描述: 该函数为工具栏新增一个元素，图标为 `icon`，文字为 `text`，并返回 `QAction` 指针;

#### 6.60.2.4 addElaIconAction

- a) 函数声明: `QAction* addElaIconAction (ElaIconType::IconName icon, const QString& text, const QKeySequence& shortcut);`
- b) 参数 1: `ElaIconType::IconName icon` // 元素图标;
- c) 参数 2: `const QString& text` // 元素文字;
- d) 参数 3: `const QKeySequence& shortcut` // 快捷键;
- e) 返回值: `QAction*` // 元素指针;
- f) 描述: 该函数为工具栏新增一个元素，图标为 `icon`，文字为 `text`，快捷键为 `sho`

`rtcut`，并返回 `QAction` 指针；

## 6.61 ElaToolButton

### 6.61.1 组件说明

工具按钮组件，调用 API 与 QT 原生一致。

### 6.61.2 公有函数

#### 6.61.2.1 setIsTransparent

- a) 函数声明：`void setIsTransparent (bool isTransparent);`
- b) 参数 1：`bool isTransparent` // 是否透明；
- c) 返回值：`void`
- d) 描述：该函数设置工具按钮是否透明，不绘制背景颜色和边框；

#### 6.61.2.2 getIsTransparent

- a) 函数声明：`bool getIsTransparent () const;`
- b) 返回值：`bool` // 是否透明；
- c) 描述：该函数获取工具按钮是否透明；

#### 6.61.2.3 setElaIcon

- a) 函数声明：`void setElaIcon (ElaIconType::IconName icon);`
- b) 参数 1：`ElaIconType::IconName icon` // 按钮图标；
- c) 返回值：`void`
- d) 描述：该函数获取工具按钮是否透明；

## 6.62 ElaTreeView

### 6.62.1 组件说明

树型视图组件，调用 API 与 QT 原生一致。

### 6.62.2 属性成员

#### 6.62.2.1 ItemHeight

- a) 类型：`int`
- b) 可用属性：`pItemHeight`
- c) 关联信号：`pItemHeightChanged`

- d) 默认值: 35
- e) 描述: 该属性控制树模型每一行的行高度;
- f) 对应的设置和获取函数:

```
void setItemHeight (int itemHeight);  
int getItemHeight () const;
```

#### 6.62.2.2 HeaderMargin

- a) 类型: int
- b) 可用属性: pHeaderMargin
- c) 关联信号: pHeaderMarginChanged
- d) 默认值: 5
- e) 描述: 该属性控制表头的 Margin 属性, 值越大表头越大;
- f) 对应的设置和获取函数:

```
void setHeaderMargin (int headerMargin);  
int getHeaderMargin () const;
```

### 6.63 ElaWidget

#### 6.63.1 组件说明

使用 ElaAppBar 的无边框模态窗口, 该窗口不可作为主窗口或子窗口使用, 仅作为提示框使用。

#### 6.63.2 公有函数

##### 6.63.2.1 setIsStayTop

- a) 函数声明: void setIsStayTop (bool isStayTop);
- b) 参数 1: bool isTransparent // 是否置顶;
- c) 返回值: void
- d) 描述: 该函数设置窗口是否置顶, true 为置顶, false 为不置顶;

##### 6.63.2.2 getIsStayTop

- a) 函数声明: void getIsStayTop () const;
- b) 返回值: bool // 是否置顶;
- c) 描述: 该函数获取窗口是否置顶;



### 6.63.2.3 setIsFixedSize

- a) 函数声明: `void setIsFixedSize (bool isFixedSize);`
- b) 参数 1: `bool isFixedSize` // 是否固定大小;
- c) 返回值: `void`
- d) 描述: 该函数控制窗口是否可以在四边进行拉伸, `true` 为允许拉伸, `false` 为禁止拉伸;

### 6.63.2.4 getIsFixedSize

- a) 函数声明: `void getIsFixedSize () const;`
- b) 返回值: `bool` // 是否固定大小;
- c) 描述: 该函数获取窗口是否可以在四边进行拉伸;

## 6.64 ElaWindow

### 6.64.1 组件说明

使用 `ElaAppBar` 并携带导航栏的无边框模态窗口, 基于 `MainWindow`; 注意, 该窗口不使用 `setCentralWidget` 进行中心窗口的指定。

### 6.64.2 属性成员

#### 6.64.2.1 ThemeChangeTime

- a) 类型: `int`
- b) 可用属性: `pThemeChangeTime`
- c) 关联信号: `pThemeChangeTimeChanged`
- d) 默认值: `700`
- e) 描述: 该属性控制主题切换的动画时间, 单位为 `ms`;
- f) 对应的设置和获取函数:

```
void setThemeChangeTime (int themeChangeTime);
```

```
int getThemeChangeTime () const;
```

#### 6.64.2.2 NavigationBarDisplayMode

- a) 类型: `ElaNavigationType::NavigationDisplayMode`
- b) 可用属性: `pNavigationBarDisplayMode`
- c) 关联信号: `pNavigationBarDisplayModeChanged`
- d) 默认值: `ElaNavigationType::NavigationDisplayMode::Auto`;

- e) 描述：该属性控制导航栏的默认显示模式；
- f) 对应的设置和获取函数：

`void setNavigationBarDisplayMode (ElaNavigationType::NavigationDisplayMode navigationBarDisplayMode);`

`int getNavigationBarDisplayMode () const;`

### 6.64.3 公有函数

#### 6.64.3.1 moveToCenter

- a) 函数声明：`void moveToCenter ()`;
- b) 返回值：`void`
- c) 描述：该函数将窗口移动到屏幕中心；

#### 6.64.3.2 setIsNavigationBarEnable

- a) 函数声明：`void setIsNavigationBarEnable (bool isEnabled)`;
- b) 参数 1：`bool isEnabled` // 是否启用导航栏；
- c) 返回值：`void`
- d) 描述：该函数控制是否启用导航栏；

#### 6.64.3.3 getIsNavigationBarEnable

- a) 函数声明：`bool getIsNavigationBarEnable () const`;
- b) 返回值：`bool` // 是否启用导航栏；
- c) 描述：该函数获取是否启用导航栏；

#### 6.64.3.4 setIsStayTop

- a) 函数声明：`void setIsStayTop (bool isStayTop)`;
- b) 参数 1：`bool isTransparent` // 是否置顶；
- c) 返回值：`void`
- d) 描述：该函数设置窗口是否置顶，true 为置顶，false 为不置顶；

#### 6.64.3.5 getIsStayTop

- a) 函数声明：`void getIsStayTop () const`;
- b) 返回值：`bool` // 是否置顶；
- c) 描述：该函数获取窗口是否置顶；

#### 6.64.3.6 setIsFixedSize

- a) 函数声明: `void setIsFixedSize (bool isFixedSize);`
- b) 参数 1: `bool isFixedSize` // 是否固定大小;
- c) 返回值: `void`
- d) 描述: 该函数控制窗口是否可以在四边进行拉伸, `true` 为允许拉伸, `false` 为禁止拉伸;

#### 6.64.3.7 getIsFixedSize

- a) 函数声明: `void getIsFixedSize () const;`
- b) 返回值: `bool` // 是否固定大小;
- c) 描述: 该函数获取窗口是否可以在四边进行拉伸;

#### 6.64.3.8 setUserInfoCardVisible

- a) 函数声明: `void setUserInfoCardVisible (bool isVisible);`
- b) 参数 1: `bool isVisible` // 是否可见;
- c) 返回值: `void`
- d) 描述: 该函数控制用户卡片的可见性;

#### 6.64.3.9 getUserInfoCardVisible

- a) 函数声明: `bool getUserInfoCardVisible () const;`
- b) 返回值: `bool` // 是否可见;
- c) 描述: 该属性获取用户卡片的可见性;

#### 6.64.3.10 setUserInfoCardTitle

- a) 函数声明: `void setUserInfoCardTitle (QString title);`
- b) 参数 1: `QString title` // 标题文字;
- c) 返回值: `void`
- d) 描述: 该属性控制用户卡片标题文字;

#### 6.64.3.11 setUserInfoCardSubTitle

- a) 函数声明: `void setUserInfoCardSubTitle (QString subTitle);`
- b) 参数 1: `QString subTitle` // 副标题文字;
- c) 返回值: `void`
- d) 描述: 该属性控制用户卡片副标题文字;

### 6.64.3.12 addExpanderNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addExpanderNode(QString expanderTitle, QString& expanderKey, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString expanderTitle` // 可展开节点的标题;
- c) 参数 2: `QString& expanderKey` // 可展开节点的识别字符串, 调用成功后被赋值;
- d) 参数 3: `ElaIconType::IconName awesome` // 可展开节点的图标;
- e) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- f) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `expanderTitle`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `expanderKey` 赋值, 值为该展开节点的识别字符串;
- g) 使用示例:  

```
QString expanderKey;  
ElaNavigationType::NodeOperateReturnType returnType = addExpanderNode("可展开节点 1", expanderKey, ElaIconType::House);  
// 添加一个可展开节点, 节点名称为“可展开节点 1”;
```

### 6.64.3.13 addExpanderNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addExpanderNode(QString expanderTitle, QString& expanderKey, QString targetExpanderKey, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString expanderTitle` // 可展开节点的标题;
- c) 参数 2: `QString& expanderKey` // 可展开节点的识别字符串, 调用成功后被赋值;
- d) 参数 3: `QString targetExpanderKey` // 目标可展开节点的识别字符串;
- e) 参数 4: `ElaIconType::IconName awesome` // 可展开节点的图标;
- f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- g) 描述: 该函数为导航栏在指定节点上添加一个可展开节点, 节点标题为 `expanderTitle`, 指定可展开节点的识别字符串为 `targetExpanderKey`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `expanderKey` 赋值, 值为该展开节点的识别字符串;
- h) 使用示例:  

```
QString expanderKey;
```

```
ElaNavigationType::NodeOperateReturnType returnType = addExpanderNode("可
展开节点 2", expanderKey, "86249d00b59a4a079bd3d3a38bebc484" , ElaIconTy
pe::House);
```

// 在指定可展开节点下添加一个可展开节点，节点名称为“可展开节点 2” 指定的可展开节点的识别字符串为“86249d00b59a4a079bd3d3a38bebc484”;

#### 6.64.3.14 addPageNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString pageTitle` // 页面节点的标题;
- c) 参数 2: `QWidget* page` // 添加的页面;
- d) 参数 3: `ElaIconType::IconName awesome` // 页面节点的图标;
- e) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- f) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- g) 使用示例:

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节
点 1", widget, ElaIconType::House);
// 添加一个页面节点, 节点名称为“页面节点 1”;
```

#### 6.64.3.15 addPageNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, QString targetExpanderKey, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString pageTitle` // 页面节点的标题;
- c) 参数 2: `QWidget* page` // 添加的页面;
- d) 参数 3: `QString targetExpanderKey` // 目标可展开节点的识别字符串;
- e) 参数 4: `ElaIconType::IconName awesome` // 页面节点的图标;
- f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- g) 描述: 该函数为导航栏在指定可展开节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 指定可展开节点的识别字符串为 `targetExpanderKey`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性,

属性名称为 `ElaIconType`，值为该页面的识别字符串；

h) 使用示例：

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, "86249d00b59a4a079bd3d3a38bebc484", ElaIconType::House);
// 在指定的可展开节点下添加一个页面节点，节点名称为“页面节点 1” 指定的可展开节点的识别字符串为“86249d00b59a4a079bd3d3a38bebc484”;
```

#### 6.64.3.16 addPageNode

- a) 函数声明：`ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString pageTitle` // 页面节点的标题；
- c) 参数 2: `QWidget* page` // 添加的页面；
- d) 参数 3: `int keyPoints` // 要点数 点击后，该要点归零；
- e) 参数 4: `ElaIconType::IconName awesome` // 页面节点的图标；
- f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果；
- g) 描述: 该函数为导航栏在根节点上添加一个可展开节点，节点标题为 `pageTitle`，添加的页面为 `page`，图标为 `awesome`，返回值为调用结果，若调用成功，对 `page` 添加属性，属性名称为 `ElaIconType`，值为该页面的识别字符串；
- h) 使用示例：

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, 99, ElaIconType::House);
// 添加一个页面节点，节点名称为“页面节点 1” 要点数为 99；
```

#### 6.64.3.17 addPageNode

- a) 函数声明：`ElaNavigationType::NodeOperateReturnType addPageNode(QString pageTitle, QWidget* page, QString targetExpanderKey, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString pageTitle` // 页面节点的标题；
- c) 参数 2: `QWidget* page` // 添加的页面；
- d) 参数 3: `QString targetExpanderKey` // 目标可展开节点的识别字符串；
- e) 参数 4: `int keyPoints` // 要点数 点击后，该要点归零；
- f) 参数 5: `ElaIconType::IconName awesome` // 页面节点的图标；



- g) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- h) 描述: 该函数为导航栏在指定可展开节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 指定可展开节点的识别字符串为 `targetExpanderKey`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- i) 使用示例:
 

```
QWidget* widget = new QWidget(this);
ElaNavigationType::NodeOperateReturnType returnType = addPageNode("页面节点 1", widget, "86249d00b59a4a079bd3d3a38bebc484", 99, ElaIconType::House);
// 在指定的可展开节点下添加一个页面节点, 节点名称为“页面节点 1” 指定的可展开节点的识别字符串为“86249d00b59a4a079bd3d3a38bebc484” 要点数为 99;
```

#### 6.64.3.18 addFooterNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addFooterNode(QString footerTitle, QString& footerKey, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`
- b) 参数 1: `QString footerTitle` // 页脚节点的标题;
- c) 参数 2: `QString& footerKey` // 页脚节点的识别字符串, 调用成功后被赋值;
- d) 参数 3: `int keyPoints` // 要点数 点击后, 该要点归零;
- e) 参数 4: `ElaIconType::IconName awesome` // 页面节点的图标;
- f) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- g) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- h) 使用示例:
 

```
QString footerKey;
ElaNavigationType::NodeOperateReturnType returnType = addFooterNode("页脚节点 1", footerKey, 99, ElaIconType::House);
// 添加一个页脚节点, 节点名称为“页脚节点 1” 要点数为 99;
```

#### 6.64.3.19 addFooterNode

- a) 函数声明: `ElaNavigationType::NodeOperateReturnType addFooterNode(QString footerTitle, QWidget* page, QString& footerKey, int keyPoints = 0, ElaIconType::IconName awesome = ElaIconType::None);`



- b) 参数 1: `QString footerTitle` // 页脚节点的标题;
- c) 参数 2: `QWidget* page` // 页脚页面;
- d) 参数 3: `QString& footerKey` // 页脚节点的识别字符串, 调用成功后被赋值;
- e) 参数 4: `int keyPoints` // 要点数 点击后, 该要点归零;
- f) 参数 5: `ElaIconType::IconName awesome` // 页面节点的图标;
- g) 返回值: `ElaNavigationType::NodeOperateReturnType` // 操作结果;
- h) 描述: 该函数为导航栏在根节点上添加一个可展开节点, 节点标题为 `pageTitle`, 添加的页面为 `page`, 图标为 `awesome`, 返回值为调用结果, 若调用成功, 对 `page` 添加属性, 属性名称为 `ElaIconType`, 值为该页面的识别字符串;
- i) 使用示例:  

```
QString footerKey;  
ElaNavigationType::NodeOperateReturnType returnType = addFooterNode("页脚节点 1", new QWidget(this), footerKey, 99, ElaIconType::House);  
// 添加一个页脚节点, 节点名称为“页脚节点 1” 要点数为 99;
```

#### 6.64.3.20 setNodeKeyPoints

- a) 函数声明: `void setNodeKeyPoints(QString nodeKey, int keyPoints);`
- b) 参数 1: `QString nodeKey` // 页脚节点或页面节点的识别字符串;
- c) 参数 2: `int keyPoints` // 要点数;
- d) 返回值: `void`
- e) 描述: 该函数为指定的页脚节点或页面节点设置要点数;

#### 6.64.3.21 getNodeKeyPoints

- a) 函数声明: `int getNodeKeyPoints(QString nodeKey) const;`
- b) 参数 1: `QString nodeKey` // 页脚节点或页面节点的识别字符串;
- c) 返回值: `int` // 要点数;
- d) 描述: 该函数返回指定的页脚节点或页面节点的要点数;

#### 6.64.3.22 navigation

- a) 函数声明: `void navigation(QString pageKey, bool isLogClicked = true);`
- b) 参数 1: `QString pageKey` // 页脚节点或页面节点的识别字符串;
- c) 参数 2: `bool isLogClicked` // 是否记录跳转;
- d) 返回值: `void`
- e) 描述: 该函数使用指定的页脚节点或页面节点的识别字符串进行路由跳转操作, 是否记录跳转由 `isLogClicked` 指定;

#### 6.64.3.23 setWindowButtonFlag

- a) 函数声明: `void setWindowButtonFlag(ElaAppBarType::ButtonType buttonFlag, bool isEnabled = true);`
- b) 参数 1: `ElaAppBarType::ButtonType buttonFlag` // 单个标题栏按钮对应的枚举
- c) 参数 2: `bool isEnabled` // 指定按钮是否可见
- d) 返回值: `void`
- e) 描述: 该函数控制标题栏的单个按钮显示状态, 对应的 `buttonFlag` 设置为 `true` 时, 按钮显示, 否则按钮隐藏;
- f) 使用示例: `setWindowButtonFlag(ElaAppBarType::MinimizeButtonHint, false);`  
// 将最小化按钮设置为隐藏;

#### 6.64.3.24 setWindowButtonFlags

- a) 函数声明: `void setWindowButtonFlags(ElaAppBarType::ButtonFlags buttonFlags);`
- b) 参数 1: `ElaAppBarType::ButtonFlags buttonFlags`// 所有标题栏按钮对应的枚举
- c) 返回值: `void`
- d) 描述: 该函数控制标题栏的单个按钮显示状态, 对应的 `buttonFlag` 设置为 `true` 时, 按钮显示, 否则按钮隐藏;
- e) 使用示例: `setWindowButtonFlags(ElaAppBarType::MinimizeButtonHint | ElaAppBarType::MaximizeButtonHint);` // 将最小化和最大化按钮设置为显示, 其他按钮设置为隐藏;

#### 6.64.3.25 getWindowButtonFlags

- a) 函数声明: `ElaAppBarType::ButtonFlags getWindowButtonFlags() const;`
- b) 返回值: `ElaAppBarType::ButtonFlags` // 所有标题栏按钮对应的枚举;
- c) 描述: 该函数返回当前可见的标题栏按钮的枚举集合;
- d) 使用示例: `ElaAppBarType::ButtonFlags flags = getWindowButtonFlags();`  
// 获取当前可见标题栏按钮的枚举集合;

#### 6.64.3.26 setCustomWidget

- a) 函数声明: `void setCustomWidget(QWidget* widget);`
- b) 参数 1: `QWidget* widget` // 自定义窗口;
- c) 返回值: `void`
- d) 描述: 该函数设置标题栏中央的自定义窗口, 可设置为任意基于 `QWidget` 实现

的窗口，如 MenuBar、TabBar 等，在进行设置后，该自定义窗口的最大高度会被指定为 AppBarHeight，最大宽度会被指定为 CustomWidgetMaximumWidth；、

#### 6.64.3.27 getCustomWidget

- a) 函数声明： `QWidget* getCustomWidget() const;`
- b) 返回值： `QWidget*` // 自定义窗口；
- c) 描述：该函数获取标题栏中央的自定义窗口；

#### 6.64.3.28 setAppBarHeight

- a) 函数声明： `void setAppBarHeight (int appBarHeight);`
- b) 参数 1： `int appBarHeight` // 自定义标题栏高度；
- c) 返回值： `void`
- d) 描述：该函数设置自定义标题栏的高度；

#### 6.64.3.29 getAppBarHeight

- a) 函数声明： `int getAppBarHeight () const;`
- b) 返回值： `int` // 自定义标题栏高度；
- c) 描述：该函数获取自定义标题栏的高度；

#### 6.64.3.30 setCustomWidgetMaximumWidth

- a) 函数声明： `void setCustomWidgetMaximumWidth(int width);`
- b) 参数 1： `int width` // 自定义窗口最大宽度；
- c) 返回值： `void`
- d) 描述：该函数设置 CustomWidget 自定义窗口的最大宽度；

#### 6.64.3.31 getCustomWidgetMaximumWidth

- a) 函数声明： `int getCustomWidgetMaximumWidth() const;`
- b) 返回值： `int` // 自定义窗口最大宽度；
- c) 描述：该函数获取 CustomWidget 自定义窗口的最大宽度；

#### 6.64.3.32 setIsDefaultClosed

- a) 函数声明： `void setIsDefaultClosed (bool isDefaultClosed);`
- b) 参数 1： `bool isDefaultClosed` //是否以默认形式关闭窗口；
- c) 返回值： `void`

- d) 描述: 该函数控制窗口是否以默认形式关闭, `true` 为以默认形式关闭, 若设置为 `false`, 点击关闭按钮或在任务栏关闭程序后, 关闭事件会被拦截, 同时发送 `closeButtonClicked` 信号, 用户可连接此信号进行处理;

#### 6.64.3.33 getIsDefaultClosed

- a) 函数声明: `bool getIsDefaultClosed () const;`
- b) 返回值: `bool` // 是否以默认形式关闭窗口;
- c) 描述: 该函数获取窗口是否以默认形式关闭;

#### 6.64.3.34 closeWindow

- a) 函数声明: `void closeWindow ();`
- b) 返回值: `void` // 是否以默认形式关闭窗口;
- c) 描述: 该函数开始一个渐变的缩小动画和透明度动画, 同时将 `ElaApplication` 类的属性 `IsApplicationClosed` 设置为 `true`, 并在动画结束后关闭窗口;

### 6.64.4 公有信号

#### 6.64.4.1 userInfoCardClicked

- a) 信号声明: `Q_SIGNAL void userInfoCardClicked ();`
- b) 描述: 用户卡片被点击时, 触发该信号;

#### 6.64.4.2 closeButtonClicked

- a) 信号声明: `Q_SIGNAL void closeButtonClicked ();`
- b) 描述: `isDefaultClosed` 设置为 `false` 且触发 `close` 事件时, 触发该信号;

#### 6.64.4.3 navigationNodeClicked

- a) 信号声明: `Q_SIGNAL void navigationNodeClicked (ElaNavigationType::NavigationNodeType nodeType, QString nodeKey);`
- b) 参数 1: `ElaNavigationType::NavigationNodeType nodeType` // 节点类型;
- c) 参数 2: `QString nodeKey` // 节点的识别字符串;
- d) 描述: 导航栏的页面节点或页脚节点被点击时 (不包括可展开节点), 触发该信号;