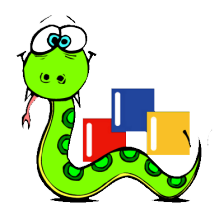
**wxPython：一曲MFC时代的挽歌，更是理想主义的绝唱**

[[](https://blog.csdn.net/xufive/category_9418763.html)wxPython](https://blog.csdn.net/xufive/category_9418763.html)

**1. 概述**

MFC是我接触到的第一个界面库，当时的操作系统还是Windows95。在那个IT技术日新月异的年代，就像一个从荒蛮部落闯进文明社会的野人第一眼看见汽车那样，我对MFC充满了好奇和迷恋。尽管后来断断续续接触了WPF、Qt等GUI库，却始终对MFC情有独钟，以至于爱屋及乌，喜欢上了wxWidgets。

wxWidgets和MFC的确太相似了，连命名习惯和架构都高度相似。事实上，wxWidgets就是跨平台的MFC，对各个平台的差异做了抽象，后端还是用各平台原生的API实现。这正是wxWidgets的优点：编译出来的程序发行包比较小，性能也相当优异。

随着MFC的日渐式微，Qt异军突起，目前已成为最强大，最受欢迎的跨平台GUI库之一。在Python生态圈里，PyQt的用户群也远超wxPython。喜欢Qt的人认为这是技术竞争的结果，但我觉得这更像是开源理念和商业化思想的差异造成的。

wxWidgets像是一个孤独的勇士，高举开源的大旗，试图以一己之力构建一个相互承认、相互尊重的理想社会；而Qt则更像是一个在商业资本驱使下不断扩张的帝国，它不满足于封装不同平台的API，而是要创造出自己的API和框架，它不仅仅是UI，而是囊括了APP开发用到的所有东西，包括网络、数据库、多媒体、蓝牙、NFC、脚本引擎等。

缺少或拒绝商业化运作的支持，wxWidgets的悲情结局早已是命中注定。如果不是因为Python的兴盛和wxPython的复兴，wxWidgets也许早已经和MFC一样被遗忘在了角落里。不无夸张地说，wxPython是以MFC为代表的一个时代的挽歌，更是一曲理想主义的绝唱。

**1.1 组织架构**

其实，wxPython谈不上什么组织架构，因为桌面程序开发所用的类、控件、组件和常量几乎都被放到了顶级命名空间wx下面了。这样做看似杂乱无章，但用起来却是非常便捷。比如，导入必要的模块，PyQt通常要这样写：

from PyQt6.QtWidgets import QApplication, QWidget, QComboBox, QPushButton, QHBoxLayout, QVBoxLayout, QColorDialog

from PyQt6.QtGui import QIcon, QPainter, QPen, QColor, QPolygon

from PyQt6.QtCore import Qt, QPoint

PyQt巨人般的体量限制了使用星号导入所有的模块，只能用什么导入什么。而wxPython只需要简短的一句话：

import wx

再比如一些常量的写法，wxPython同样简洁，PyQt已经长到匪夷所思的程度了。比如左对齐和确定取消键，wxPython这样写：

wx.ALIGN\_LEFT

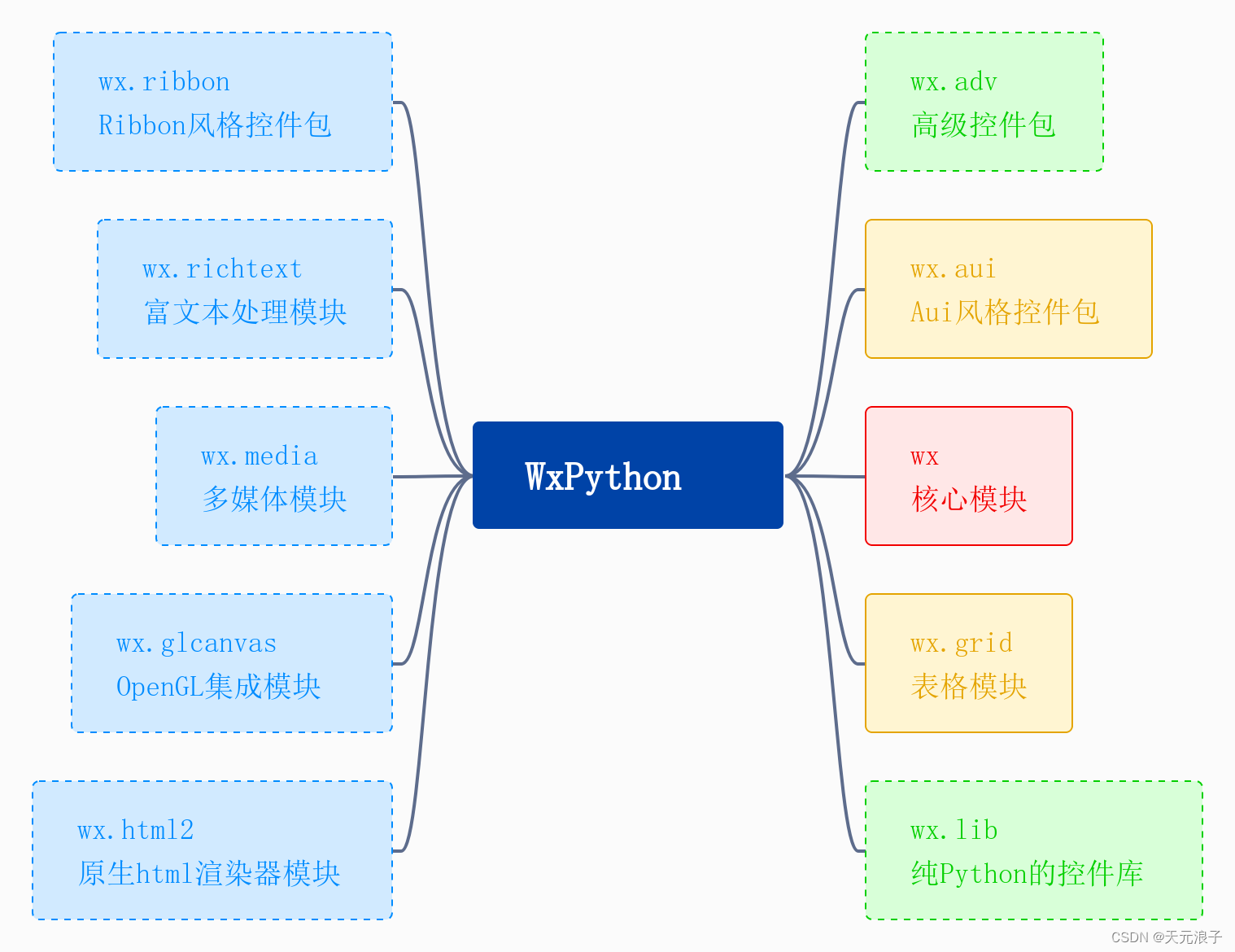
wx.OK | wx.CANCEL

PyQt写出来几乎要占一整行：

Qt.AlignmentFlag.AlignLeft

QMessageBox.StandardButton.Ok | QMessageBox.StandardButton.Cancel

尽管wxPython也与时俱进地增加了一些诸如wx.xml、wx.svg之类地外围模块，但除了wx这个核心模块之外，我个人觉得只有wx.aui和wx.grid模块算是必要的扩展。如果想让界面更花哨点，那就要了解以下wx.adv、wx.ribbon这两个模块，纯python构建的控件库wx.lib也绝对值得一试。总之，站在我的应用领域看，wxPython的组织架构如下图所示。根据使用频率的高低，我给各个模块标注了红黄绿蓝四种颜色。



**1.2 安装**

截至本文写作时，wxPython的最新版本是4.1.1。Windows用户和macOS用户可以直接使用下面的命令安装。

pip install -U wxPython

由于Linux平台存在发行版之间的差异，必须使用相应的包管理器进行下载和安装。 例如，在Ubuntu系统上可以尝试下面的安装命令。

sudo apt-get install python3-wxgtk4.0 python3-wxgtk-webview4.0 python3-wxgtk-media4.0

**2 快速体验**

**2.1 桌面应用程序开发的一般流程**

用wxPython写一个桌面应用程序，通常分为6个步骤：

* 第1步：导入模块
* 第2步：创建一个应用程序
* 第3步：创建主窗口
* 第4步：在主窗口上实现业务逻辑
* 第5步：显示窗主口
* 第6步：应用程序进入事件处理主循环

除第4步之外的其它步骤，基本都是一行代码就可以完成，第4步的复杂程度取决于功能需求的多寡和业务逻辑的复杂度。下面这段代码就是这个一般流程的体现。

# 第1步：导入模块

import wx

# 第2步：创建一个应用程序

app = wx.App()

# 第3步：创建主窗口

frame = wx.Frame(None)

# 第4步：在主窗口上实现业务逻辑

st = wx.StaticText(frame, -1, 'Hello World')

# 第5步：显示窗主口

frame.Show()

# 第6步：应用程序进入事件处理主循环

app.MainLoop()

**2.2 Hello World**

实际应用wxPython开发桌面应用程序的的时候，上面这样的写法难以实现和管控复杂的业务逻辑，因而都是采用面向对象的应用方式。下面的代码演示了以OOP的方式使用wxPython，并且为窗口增加了标题和图标，设置了窗口尺寸和背景色，同时也给静态文本控件StaticText设置了字体字号。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, -1,style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('最简的的应用程序')

self.SetIcon(wx.Icon('res/wx.ico')) # 设置图标

self.SetBackgroundColour((217, 228, 0)) # 设置窗口背景色

self.SetSize((300, 80)) # 设置窗口大小

self.Center() # 窗口在屏幕上居中

st = wx.StaticText(self, -1, 'Hello World', style=wx.ALIGN\_CENTER) # 生成静态文本控件，水平居中

st.SetFont(wx.Font(20, wx.DEFAULT, wx.NORMAL, wx.NORMAL, False, 'Arial')) # 设置字体字号

if \_\_name\_\_ == '\_\_main\_\_':

app = wx.App() # 创建一个应用程序

frame = MainFrame(None) # 创建主窗口

frame.Show() # 显示窗主口

app.MainLoop() # 应用程序进入事件处理主循环

代码中用到了一个.png格式的图像文件文件，想要运行这段代码的话，请先替换成本地文件。至于文件格式，SetIcon方法没有限定，常见的包括.ico和.jpg在内的图像格式都支持。代码运行界面如下图所示。



**2.3 常用控件介绍**

尽管wxPython的核心模块和扩展模块提供了数以百计的各式控件和组件，但真正常用且必不可少的控件只有为数不多的几个：

* wx.Frame - 窗口
* wx.Panel - 面板
* wx.StaticText - 静态文本
* StaticBitmap - 静态图片
* wx.TextCtrl - 单行或多行文本输入框
* wx.Button - 按钮
* wx.RadioButton - 单选按钮
* wx.CheckBox - 复选按钮
* wx.Choice - 下拉选择框

所有的wxPython控件都有一个不可或缺的parent参数和若干关键字参数，通常，关键字参数都有缺省默认值。

* parent - 父级对象
* id - 控件的唯一标识符，缺省或-1表示自动生成
* pos - 控件左上角在其父级对象上的绝对位置
* size - 控件的宽和高
* name - 用户定义的控件名
* style - 控件风格

wxPython的控件在使用风格上保持着高度的一致性，一方面因为它们从一个共同的基类派生而来，更重要的一点，wxPython不像PyQt那样充斥着随处可见的重载函数。比如，PyQt的菜单栏QMenuBar增加菜单，就有addMenu(QMenu)、addMenu(str)和addMenu(QIcon, str)等三种不同的重载形式。方法重载固然带来了很多便利，但也会增加使用难度，让用户无所适从。

下面的代码演示了上述常用控件的使用方法。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

# 调用父类的构造函数，从默认风格中去除改变窗口大小

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE ^ wx.RESIZE\_BORDER)

self.SetTitle('wxPython控件演示')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetSize((860, 450))

self.Center()

# 创建一个面板，用于放置控件

panel = wx.Panel(self, -1)

# 在x=20，y=20的位置，创建静态文本控件

st = wx.StaticText(panel, -1, '我是静态文本控件', pos=(20, 20))

# 在x=300，y=20的位置，创建静态图片

bmp = wx.Bitmap('res/forever.png')

sb = wx.StaticBitmap(panel, -1, bmp, pos=(280, 10))

# 在x=20, y=50的位置，创建文本输入框，指定输入框的宽度为260像素，高度默认

tc1 = wx.TextCtrl(panel, -1, value='我是文本输入框', pos=(20, 50), size=(260, -1))

# 在x=20, y=90的位置，创建文本输入框，指定样式为密码

tc2 = wx.TextCtrl(panel, -1, value='我是密码', pos=(20, 90), style=wx.TE\_PASSWORD)

# 在x=20, y=130的位置，创建单选按钮，成组的单选按钮，第一个需要指定样式wx.RB\_GROUP

rb1 = wx.RadioButton(panel, -1, '单选按钮1', pos=(20, 130), style=wx.RB\_GROUP, name='rb1')

# 在x=100, y=130的位置，创建单选按钮，不再需要指定样式wx.RB\_GROUP

rb2 = wx.RadioButton(panel, -1, '单选按钮2', pos=(100, 130), name='rb2')

# 在x=180, y=130的位置，创建单选按钮，不再需要指定样式wx.RB\_GROUP

rb3 = wx.RadioButton(panel, -1, '单选按钮3', pos=(180, 130), name='rb3')

# 在x=20, y=160的位置，创建复选按钮

cb1 = wx.CheckBox(panel, -1, '复选按钮', pos=(20, 160))

# 在x=100, y=160的位置，创建复选按钮，指定其样式为wx.ALIGN\_RIGHT

cb2 = wx.CheckBox(panel, -1, '文字在左侧的复选按钮', pos=(100, 160), style=wx.ALIGN\_RIGHT)

# 在x=20，y=190的位置，创建按钮

ch = wx.Choice(panel, -1, choices=['wxPython', 'PyQt', 'Tkinter'], pos=(20, 190), size=(100, -1))

ch.SetSelection(0)

# 在x=120，y=190的位置，创建按钮

btn = wx.Button(panel, -1, '按钮', pos=(150, 190))

# 在x=20，y=230的位置，创建文本框，指定大小为260\*150，并指定其样式为多行和只读

tc3 = wx.TextCtrl(panel, -1, value='我是多行文本输入框', pos=(20, 230), size=(260, 150), style=wx.TE\_MULTILINE | wx.CB\_READONLY)

if \_\_name\_\_ == '\_\_main\_\_':

app = wx.App() # 创建一个应用程序

frame = MainFrame(None) # 创建主窗口

frame.Show() # 显示窗主口

app.MainLoop() # 应用程序进入事件处理主循环

代码运行界面如下图所示。



**3. 控件布局**

**3.1. 分区布局**

上面的例子里，输入框、按钮等控件的位置由其pos参数确定，即绝对定位。绝对定位这种布局方式非常直观，但不能自动适应窗口的大小变化。更普遍的方式是使用被称为布局管理器的wx.Sizer来实现分区布局。所谓分区布局，就是将一个矩形区域沿水平或垂直方向分割成多个矩形区域，并可嵌套分区布局管理器wx.Sizer的派生类有很多种，最常用到是wx.BoxSizer和wx.StaticBoxSizer。

和一般的控件不同，布局管理器就像是一个魔法口袋：它是无形的，但可以装进不限数量的任意种类的控件——包括其他的布局管理器。当然，魔法口袋也不是万能的，它有一个限制条件：装到里面的东西，要么是水平排列的，要么是垂直排列的，不能排成方阵。好在程序员可以不受限制地使用魔法口袋，当我们需要排成方阵时，可以先每一行使用一个魔法口袋，然后再把所有的行装到一个魔法口袋中。

创建一个魔法口袋，装进几样东西，然后在窗口中显示的伪代码是这样的：

魔法口袋 = wx.BoxSizer() # 默认是水平的，想要垂直放东西，需要加上 wx.VERTICAL 这个参数

魔法口袋.add(确认按钮, 0, wx.ALL, 0) # 装入确认按钮

魔法口袋.add(取消按钮, 0, wx.ALL, 0) # 装入取消按钮

窗口.SetSizer(魔法口袋) # 把魔法口袋放到窗口上

窗口.Layout() # 窗口重新布局

魔法口袋的 add() 方法总共有4个参数：第1个参数很容易理解，就是要装进口袋的物品；第2个参数和所有 add() 方法的第2个参数之和的比，表示装进口袋的物品占用空间的比例，0表示物品多大就占多大地儿，不额外占用空间；第3个参数相对复杂些，除了约定装进口袋的物品在其占用的空间里面水平垂直方向的对齐方式外，还可以指定上下左右四个方向中的一个或多个方向的留白（padding）；第4个参数就是留白像素数。

下面是一个完整的例子。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('分区布局')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetSize((640, 320)) # 设置窗口大小

self.\_init\_ui() # 初始化界面

self.Center() # 窗口在屏幕上居中

def \_init\_ui(self):

"""初始化界面"""

# 创建容器面板

panel = wx.Panel(self, -1)

# 生成黑色背景的预览面板

view = wx.Panel(panel, -1, style=wx.SUNKEN\_BORDER)

view.SetBackgroundColour(wx.Colour(0, 0, 0))

# 生成按钮和多行文本控件

btn\_capture = wx.Button(panel, -1, '拍照', size=(100, -1))

btn\_up = wx.Button(panel, -1, '↑', size=(30, 30))

btn\_down = wx.Button(panel, -1, '↓', size=(30, 30))

btn\_left = wx.Button(panel, -1, '←', size=(30, 30))

btn\_right = wx.Button(panel, -1, '→', size=(30, 30))

tc = wx.TextCtrl(panel, -1, '', style=wx.TE\_MULTILINE)

# 左右按钮装入一个水平布局管理器

sizer\_arrow\_mid = wx.BoxSizer()

sizer\_arrow\_mid.Add(btn\_left, 0, wx.RIGHT, 16)

sizer\_arrow\_mid.Add(btn\_right, 0, wx.LEFT, 16)

# 生成带标签的垂直布局管理器

sizer\_arrow = wx.StaticBoxSizer(wx.StaticBox(panel, -1, '方向键'), wx.VERTICAL)

sizer\_arrow.Add(btn\_up, 0, wx.ALIGN\_CENTER|wx.ALL, 0) # 装入上按钮

sizer\_arrow.Add(sizer\_arrow\_mid, 0, wx.TOP|wx.BOTTOM, 1) # 装入左右按钮

sizer\_arrow.Add(btn\_down, 0, wx.ALIGN\_CENTER|wx.ALL, 0) # 装入下按钮

# 生成垂直布局管理器

sizer\_right = wx.BoxSizer(wx.VERTICAL)

sizer\_right.Add(btn\_capture, 0, wx.ALL, 20) # 装入拍照按钮

sizer\_right.Add(sizer\_arrow, 0, wx.ALIGN\_CENTER|wx.ALL, 0) # 装入方向键

sizer\_right.Add(tc, 1, wx.ALL, 10) # 装入多行文本控件

# 生成水平布局管理器

sizer\_max = wx.BoxSizer()

sizer\_max.Add(view, 1, wx.EXPAND|wx.LEFT|wx.TOP|wx.BOTTOM, 5) # 装入左侧的预览面板

sizer\_max.Add(sizer\_right, 0, wx.EXPAND|wx.ALL, 0) # 装入右侧的操作区

# 为容器面板指定布局管理器，并调用布局方法完成界面布局

panel.SetSizer(sizer\_max)

panel.Layout()

if \_\_name\_\_ == '\_\_main\_\_':

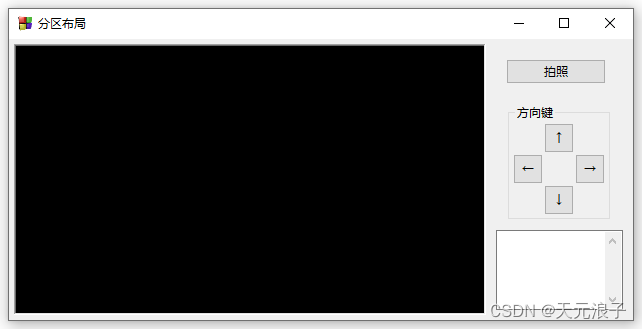
app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**3.2. 栅格布局**

顾名思义，栅格布局就是将布局空间划分成网格，将控件放置到不同的网格内。栅格布局比较简单，用起来非常方便。栅格布局布局管理器也有很多种，GridBagSizer是最常用的一种。下面是一个使用GridBagSizer实现栅格布局的例子。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('栅格布局')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetSize((800, 440)) # 设置窗口大小

self.\_init\_ui() # 初始化界面

self.Center() # 窗口在屏幕上居中

def \_init\_ui(self):

"""初始化界面"""

panel = wx.Panel(self, -1) # 创建容器面板

sizer = wx.GridBagSizer(10, 10)# 每个控件之间横纵间隔10像素

st = wx.StaticText(panel, -1, "用户名")

sizer.Add(st, (0, 0), flag=wx.TOP | wx.ALIGN\_RIGHT, border=20) # 在第0行0列，距离上边缘20像素，右对齐

userName = wx.TextCtrl(panel, -1)

sizer.Add(userName, (0, 1), (1, 3), flag=wx.EXPAND | wx.TOP, border=20) # 在第0行1列，跨3列，距离上边缘20像素

sb = wx.StaticBitmap(panel, -1, wx.Bitmap('res/python.jpg'))

sizer.Add(sb, (0, 5), (7, 1), flag=wx.TOP | wx.RIGHT, border=20) # 在第0行4列，跨7行，距离上右边缘20像素

st = wx.StaticText(panel, -1, "密码")

sizer.Add(st, (1, 0), flag=wx.ALIGN\_RIGHT) # 在第1行0列，右对齐

password = wx.TextCtrl(panel, -1, style=wx.TE\_PASSWORD)

sizer.Add(password, (1, 1), (1, 3), flag=wx.EXPAND) # 在第1行1列，跨3列

st = wx.StaticText(panel, -1, "学历")

sizer.Add(st, (2, 0), flag=wx.ALIGN\_RIGHT) # 在第2行0列，右对齐

level1 = wx.RadioButton(panel, -1, "专科")

sizer.Add(level1, (2, 1)) # 在第2行1列

level2 = wx.RadioButton(panel, -1, "本科")

sizer.Add(level2, (2, 2)) # 在第2行1列

level3 = wx.RadioButton(panel, -1, "研究生及以上")

sizer.Add(level3, (2, 3)) # 在第2行1列

st = wx.StaticText(panel, -1, "职业")

sizer.Add(st, (3, 0), flag=wx.ALIGN\_RIGHT) # 在第3行0列，右对齐

professional = wx.Choice(panel, -1, choices=["学生", "程序员", "软件工程师", "系统架构师"])

professional.SetSelection(0)

sizer.Add(professional, (3, 1), (1, 3), flag=wx.EXPAND) # 在第3行1列，跨3列

# 语言技能

st = wx.StaticText(panel, -1, "语言技能")

sizer.Add(st, (4, 0), flag=wx.ALIGN\_RIGHT | wx.LEFT, border=20) # 在第4行0列，距离左边缘20像素，右对齐

choices = ["C", "C++", "Java", "Python", "Lua", "JavaScript", "TypeScript", "Go", "Rust"]

language = wx.ListBox(panel, -1, choices=choices, style=wx.LB\_EXTENDED)

sizer.Add(language, (4, 1), (1, 3), flag=wx.EXPAND) # 在第4行1列，跨3列

isJoin = wx.CheckBox(panel, -1, "已加入QQ群", style=wx.ALIGN\_RIGHT)

sizer.Add(isJoin, (5, 0), (1, 4), flag=wx.ALIGN\_CENTER) # 在第5行0列，跨4列, 居中

btn = wx.Button(panel, -1, "提交")

sizer.Add(btn, (6, 0), (1, 4), flag=wx.ALIGN\_CENTER | wx.BOTTOM, border=20) # 在第6行0列，跨4列, 居中

sizer.AddGrowableRow(4) # 设置第4行可增长

sizer.AddGrowableCol(3) # 设置第3列可增长

panel.SetSizer(sizer)

panel.Layout()

if \_\_name\_\_ == '\_\_main\_\_':

app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**4. 事件驱动**

一个桌面程序不单是控件的罗列，更重要的是对外部的刺激——包括用户的操作做出反应。如果把窗体和控件比作是桌面程序的躯体，那么响应外部刺激就是它的灵魂。wxPython的灵魂是事件驱动机制：当某事件发生时，程序就会自动执行预先设定的动作。

**4.1 事件**

所谓事件，就是我们的程序在运行中发生的事儿。事件可以是低级的用户动作，如鼠标移动或按键按下，也可以是高级的用户动作（定义在wxPython的窗口部件中的），如单击按钮或菜单选择。事件可以产生自系统，如关机,，也可以由用户自定义事件。

除了用户自定义事件，在wxPython中我习惯把事件分为4类：

* 鼠标事件：鼠标左右中键和滚轮动作，以及鼠标移动等事件
* 键盘事件：用户敲击键盘产生的事件
* 控件事件：发生在控件上的事件，比如按钮被按下、输入框内容改变等
* 系统事件：关闭窗口、改变窗口大小、重绘、定时器等事件

事实上，这个分类方法不够严谨。比如，wx.Frame作为一个控件，关闭和改变大小也是控件事件，不过这一类事件通常都由系统绑定了行为。基于此，可以重新定义所谓的控件事件，是指发生在控件上的、系统并未预定义行为的事件。

常用的鼠标事件包括：

* wx.EVT\_LEFT\_DOWN - 左键按下
* wx.EVT\_LEFT\_UP - 左键弹起
* wx.EVT\_LEFT\_DCLICK - 左键双击
* wx.EVT\_RIGHT\_DOWN - 右键按下
* wx.EVT\_RIGHT\_UP - 右键弹起
* wx.EVT\_RIGHT\_DCLICK - 右键双击
* wx.EVT\_MOTION - 鼠标移动
* wx.EVT\_MOUSEWHEEL - 滚轮滚动
* wx.EVT\_MOUSE\_EVENTS - 所有的鼠标事件

常用的键盘事件有：

* wx.EVT\_KEY\_DOWN - 按键按下
* wx.EVT\_KEY\_UP - 按键弹起

常用的系统事件包括：

* wx.EVT\_CLOSE - 关闭
* wx.EVT\_SIZE - 改变大小
* wx.EVT\_TIMER - 定时器事件
* wx.EVT\_PAINT - 重绘
* wx.EVT\_ERASE\_BACKGROUND -背景擦除

常用的控件事件包括：

* wx.EVT\_BUTTON - 点击按钮
* wx.EVT\_CHOICE - 下拉框改变选择
* wx.EVT\_TEXT - 输入框内容改变
* wx.EVT\_TEXT\_ENTER - 输入框回车
* wx.EVT\_RADIOBOX - 单选框改变选择
* wx.EVT\_CHECKBOX - 点击复选框

**4.2 事件绑定**

事件驱动机制有三个要素：事件、事件函数和事件绑定。比如，当一个按钮被点击时，就会触发按钮点击事件，该事件如果绑定了事件函数，事件函数就会被调用。所有的事件函数都以事件对象为参数，事件对象提供了事件的详细信息，比如键盘按下事件的事件对象就包含了被按下的键的信息。

下面这个例子演示了如何定义事件函数，以及绑定事件和事件函数之间的关联关系。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('事件和事件函数的绑定')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetBackgroundColour((224, 224, 224)) # 设置窗口背景色

self.SetSize((520, 220))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

wx.StaticText(self, -1, '第一行输入框：', pos=(40, 50), size=(100, -1), style=wx.ALIGN\_RIGHT)

wx.StaticText(self, -1, '第二行输入框：', pos=(40, 80), size=(100, -1), style=wx.ALIGN\_RIGHT)

self.tip = wx.StaticText(self, -1, u'', pos=(145, 110), size=(150, -1), style=wx.ST\_NO\_AUTORESIZE)

self.tc1 = wx.TextCtrl(self, -1, '', pos=(145, 50), size=(150, -1), name='TC01', style=wx.TE\_CENTER)

self.tc2 = wx.TextCtrl(self, -1, '', pos=(145, 80), size=(150, -1), name='TC02', style=wx.TE\_PASSWORD|wx.ALIGN\_RIGHT)

btn\_mea = wx.Button(self, -1, '鼠标左键事件', pos=(350, 50), size=(100, 25))

btn\_meb = wx.Button(self, -1, '鼠标所有事件', pos=(350, 80), size=(100, 25))

btn\_close = wx.Button(self, -1, '关闭窗口', pos=(350, 110), size=(100, 25))

self.tc1.Bind(wx.EVT\_TEXT, self.on\_text) # 绑定文本内容改变事件

self.tc2.Bind(wx.EVT\_TEXT, self.on\_text) # 绑定文本内容改变事件

btn\_close.Bind(wx.EVT\_BUTTON, self.on\_close, btn\_close) # 绑定按键事件

btn\_close.Bind(wx.EVT\_MOUSEWHEEL, self.on\_wheel) # 绑定鼠标滚轮事件

btn\_mea.Bind(wx.EVT\_LEFT\_DOWN, self.on\_left\_down) # 绑定鼠标左键按下

btn\_mea.Bind(wx.EVT\_LEFT\_UP, self.on\_left\_up) # 绑定鼠标左键弹起

btn\_meb.Bind(wx.EVT\_MOUSE\_EVENTS, self.on\_mouse) # 绑定所有鼠标事件

self.Bind(wx.EVT\_CLOSE, self.on\_close) # 绑定窗口关闭事件

self.Bind(wx.EVT\_SIZE, self.on\_size) # 绑定改变窗口大小事件

self.Bind(wx.EVT\_KEY\_DOWN, self.on\_key\_down) # 绑定键盘事件

def on\_text(self, evt):

"""输入框事件函数"""

obj = evt.GetEventObject()

objName = obj.GetName()

text = evt.GetString()

if objName == 'TC01':

self.tc2.SetValue(text)

elif objName == 'TC02':

self.tc1.SetValue(text)

def on\_size(self, evt):

'''改变窗口大小事件函数'''

print('你想改变窗口，但是事件被Skip了，所以没有任何改变')

evt.Skip() # 注释掉此行（事件继续传递），窗口大小才会被改变

def on\_close(self, evt):

"""关闭窗口事件函数"""

dlg = wx.MessageDialog(None, '确定要关闭本窗口？', '操作提示', wx.YES\_NO | wx.ICON\_QUESTION)

if(dlg.ShowModal() == wx.ID\_YES):

self.Destroy()

def on\_left\_down(self, evt):

"""左键按下事件函数"""

self.tip.SetLabel('左键按下')

def on\_left\_up(self, evt):

"""左键弹起事件函数"""

self.tip.SetLabel('左键弹起')

def on\_wheel(self, evt):

"""鼠标滚轮事件函数"""

vector = evt.GetWheelRotation()

self.tip.SetLabel(str(vector))

def on\_mouse(self, evt):

"""鼠标事件函数"""

self.tip.SetLabel(str(evt.EventType))

def on\_key\_down(self, evt):

"""键盘事件函数"""

key = evt.GetKeyCode()

self.tip.SetLabel(str(key))

if \_\_name\_\_ == '\_\_main\_\_':

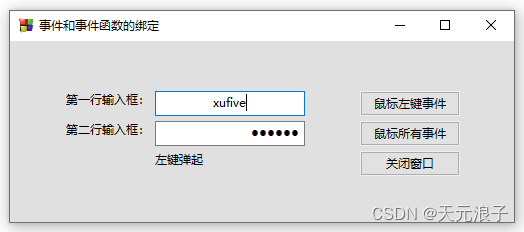
app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



两个输入框，一个明文居中，一个密写右齐，但内容始终保持同步。输入焦点不在输入框的时候，敲击键盘，界面显示对应的键值。最上面的按钮响应鼠标左键的按下和弹起事件，中间的按钮响应所有的鼠标事件，下面的按钮响应滚轮事件和按钮按下的事件。另外，程序还绑定了窗口关闭事件，重新定义了关闭函数，增加了确认选择。

**5. 程序框架**

**5.1 菜单栏、工具栏和状态栏**

通常，一个完整的窗口程序一般都有菜单栏、工具栏和状态栏。下面的代码演示了如何创建菜单栏、工具栏和状态栏，顺便演示了类的静态属性的定义和用法。不过，说实话，wx的工具栏有点丑，幸好，wx还有一个 AUI 的工具栏比较漂亮，我会在后面的例子里演示它的用法。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

id\_open = wx.NewIdRef()

id\_save = wx.NewIdRef()

id\_quit = wx.NewIdRef()

id\_help = wx.NewIdRef()

id\_about = wx.NewIdRef()

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('菜单、工具栏、状态栏')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetBackgroundColour((224, 224, 224)) # 设置窗口背景色

self.SetSize((360, 180))

self.\_create\_menubar() # 菜单栏

self.\_create\_toolbar() # 工具栏

self.\_create\_statusbar() # 状态栏

self.Center()

def \_create\_menubar(self):

"""创建菜单栏"""

self.mb = wx.MenuBar()

# 文件菜单

m = wx.Menu()

m.Append(self.id\_open, '打开文件')

m.Append(self.id\_save, '保存文件')

m.AppendSeparator()

m.Append(self.id\_quit, '退出系统')

self.mb.Append(m, '文件')

self.Bind(wx.EVT\_MENU, self.on\_open, id=self.id\_open)

self.Bind(wx.EVT\_MENU, self.on\_save, id=self.id\_save)

self.Bind(wx.EVT\_MENU, self.on\_quit, id=self.id\_quit)

# 帮助菜单

m = wx.Menu()

m.Append(self.id\_help, '帮助主题')

m.Append(self.id\_about, '关于...')

self.mb.Append(m, '帮助')

self.Bind(wx.EVT\_MENU, self.on\_help,id=self.id\_help)

self.Bind(wx.EVT\_MENU, self.on\_about,id=self.id\_about)

self.SetMenuBar(self.mb)

def \_create\_toolbar(self):

"""创建工具栏"""

bmp\_open = wx.Bitmap('res/open\_mso.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

bmp\_save = wx.Bitmap('res/save\_mso.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

bmp\_help = wx.Bitmap('res/help\_mso.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

bmp\_about = wx.Bitmap('res/info\_mso.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

self.tb = wx.ToolBar(self)

self.tb.SetToolBitmapSize((16,16))

self.tb.AddTool(self.id\_open, '打开文件', bmp\_open, shortHelp='打开', kind=wx.ITEM\_NORMAL)

self.tb.AddTool(self.id\_save, '保存文件', bmp\_save, shortHelp='保存', kind=wx.ITEM\_NORMAL)

self.tb.AddSeparator()

self.tb.AddTool(self.id\_help, '帮助', bmp\_help, shortHelp='帮助', kind=wx.ITEM\_NORMAL)

self.tb.AddTool(self.id\_about, '关于', bmp\_about, shortHelp='关于', kind=wx.ITEM\_NORMAL)

self.tb.Realize()

def \_create\_statusbar(self):

"""创建状态栏"""

self.sb = self.CreateStatusBar()

self.sb.SetFieldsCount(3)

self.sb.SetStatusWidths([-2, -1, -1])

self.sb.SetStatusStyles([wx.SB\_RAISED, wx.SB\_RAISED, wx.SB\_RAISED])

self.sb.SetStatusText('状态信息0', 0)

self.sb.SetStatusText('', 1)

self.sb.SetStatusText('状态信息2', 2)

def on\_open(self, evt):

"""打开文件"""

self.sb.SetStatusText(u'打开文件', 1)

def on\_save(self, evt):

"""保存文件"""

self.sb.SetStatusText(u'保存文件', 1)

def on\_quit(self, evt):

"""退出系统"""

self.sb.SetStatusText(u'退出系统', 1)

self.Destroy()

def on\_help(self, evt):

"""帮助"""

self.sb.SetStatusText(u'帮助', 1)

def on\_about(self, evt):

"""关于"""

self.sb.SetStatusText(u'关于', 1)

if \_\_name\_\_ == '\_\_main\_\_':

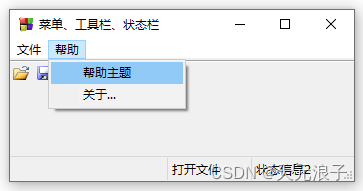
app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码里面用到了4个16x16的工具按钮，请自备4个图片文件，保存路径请查看代码中的注释。代码运行界面如下图所示。



**5.2 Aui框架**

Advanced User Interface，简称AUI，是wxPython的子模块，使用AUI可以方便地开发出美观、易用的用户界面。从2.8.9.2版本之后，wxPython增加了一个高级通用部件库Advanced Generic Widgets，简称AGW库， AGW库也提供了AUI模块 wx.lib.agw.aui，而 wx.aui也依然保留着。相比较而言，我更喜欢使用wx.lib.agw的AUI框架。

使用AUI框架可以概括为以下四步：

1. 创建一个布局管理器：mgr = aui.AuiManager()
2. 告诉主窗口由mgr来管理界面：mgr.SetManagedWindow()
3. 添加界面上的各个区域：mgr.AddPane()
4. 更新界面显示：mgr.Update()

下面的代码演示了如何使用AUI布局管理器创建和管理窗口界面。

import wx

import wx.lib.agw.aui as aui

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

id\_open = wx.NewIdRef()

id\_save = wx.NewIdRef()

id\_quit = wx.NewIdRef()

id\_help = wx.NewIdRef()

id\_about = wx.NewIdRef()

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('菜单、工具栏、状态栏')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetBackgroundColour((224, 224, 224)) # 设置窗口背景色

self.SetSize((640, 480))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

self.tb1 = self.\_create\_toolbar()

self.tb2 = self.\_create\_toolbar()

self.tbv = self.\_create\_toolbar('V')

p\_left = wx.Panel(self, -1)

p\_center0 = wx.Panel(self, -1)

p\_center1 = wx.Panel(self, -1)

p\_bottom = wx.Panel(self, -1)

btn = wx.Button(p\_left, -1, '切换', pos=(30,200), size=(100, -1))

btn.Bind(wx.EVT\_BUTTON, self.on\_switch)

text0 = wx.StaticText(p\_center0, -1, '我是第1页', pos=(40, 100), size=(200, -1), style=wx.ALIGN\_LEFT)

text1 = wx.StaticText(p\_center1, -1, '我是第2页', pos=(40, 100), size=(200, -1), style=wx.ALIGN\_LEFT)

self.\_mgr = aui.AuiManager()

self.\_mgr.SetManagedWindow(self)

self.\_mgr.AddPane(self.tb1,

aui.AuiPaneInfo().Name('ToolBar1').Caption('工具条').ToolbarPane().Top().Row(0).Position(0).Floatable(False)

)

self.\_mgr.AddPane(self.tb2,

aui.AuiPaneInfo().Name('ToolBar2').Caption('工具条').ToolbarPane().Top().Row(0).Position(1).Floatable(True)

)

self.\_mgr.AddPane(self.tbv,

aui.AuiPaneInfo().Name('ToolBarV').Caption('工具条').ToolbarPane().Right().Floatable(True)

)

self.\_mgr.AddPane(p\_left,

aui.AuiPaneInfo().Name('LeftPanel').Left().Layer(1).MinSize((200,-1)).Caption('操作区').MinimizeButton(True).MaximizeButton(True).CloseButton(True)

)

self.\_mgr.AddPane(p\_center0,

aui.AuiPaneInfo().Name('CenterPanel0').CenterPane().Show()

)

self.\_mgr.AddPane(p\_center1,

aui.AuiPaneInfo().Name('CenterPanel1').CenterPane().Hide()

)

self.\_mgr.AddPane(p\_bottom,

aui.AuiPaneInfo().Name('BottomPanel').Bottom().MinSize((-1,100)).Caption('消息区').CaptionVisible(False).Resizable(True)

)

self.\_mgr.Update()

def \_create\_toolbar(self, d='H'):

"""创建工具栏"""

bmp\_open = wx.Bitmap('res/open\_mso.png', wx.BITMAP\_TYPE\_ANY)

bmp\_save = wx.Bitmap('res/save\_mso.png', wx.BITMAP\_TYPE\_ANY)

bmp\_help = wx.Bitmap('res/help\_mso.png', wx.BITMAP\_TYPE\_ANY)

bmp\_about = wx.Bitmap('res/info\_mso.png', wx.BITMAP\_TYPE\_ANY)

if d.upper() in ['V', 'VERTICAL']:

tb = aui.AuiToolBar(self, -1, wx.DefaultPosition, wx.DefaultSize, agwStyle=aui.AUI\_TB\_TEXT|aui.AUI\_TB\_VERTICAL)

else:

tb = aui.AuiToolBar(self, -1, wx.DefaultPosition, wx.DefaultSize, agwStyle=aui.AUI\_TB\_TEXT)

tb.SetToolBitmapSize(wx.Size(16, 16))

tb.AddSimpleTool(self.id\_open, '打开', bmp\_open, '打开文件')

tb.AddSimpleTool(self.id\_save, '保存', bmp\_save, '保存文件')

tb.AddSeparator()

tb.AddSimpleTool(self.id\_help, '帮助', bmp\_help, '帮助')

tb.AddSimpleTool(self.id\_about, '关于', bmp\_about, '关于')

tb.Realize()

return tb

def on\_switch(self, evt):

"""切换信息显示窗口"""

p0 = self.\_mgr.GetPane('CenterPanel0')

p1 = self.\_mgr.GetPane('CenterPanel1')

p0.Show(not p0.IsShown())

p1.Show(not p1.IsShown())

self.\_mgr.Update()

if \_\_name\_\_ == '\_\_main\_\_':

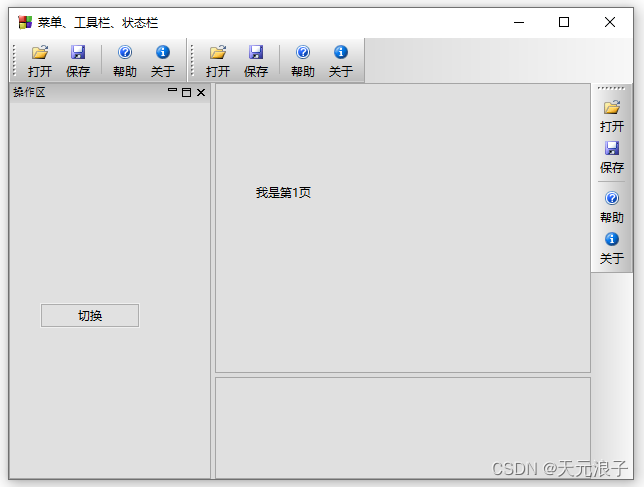
app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**6. 示例和技巧**

**6.1. 相册**

前文的例子中已经展示了wx.StaticBitmap控件作为图像容器的例子，下面的例子用它制作了一个相册，点击前翻后翻按钮可在多张照片之间循环切换。

import wx

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('相册')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetBackgroundColour((224, 224, 224)) # 设置窗口背景色

self.SetSize((980, 680))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

self.curr = 0

self.photos = ('res/DSC03363.jpg', 'res/DSC03394.jpg', 'res/DSC03402.jpg')

bmp = wx.Bitmap(self.photos[self.curr])

self.album = wx.StaticBitmap(self, -1, bmp, pos=(280, 10))

btn\_1 = wx.Button(self, -1, '<', size=(80, 30), name='prev')

btn\_2 = wx.Button(self, -1, '>', size=(80, 30), name='next')

btn\_1.Bind(wx.EVT\_BUTTON, self.on\_btn)

btn\_2.Bind(wx.EVT\_BUTTON, self.on\_btn)

sizer\_btn = wx.BoxSizer()

sizer\_btn.Add(btn\_1, 0, wx.RIGHT, 20)

sizer\_btn.Add(btn\_2, 0, wx.LEFT, 20)

sizer\_max = wx.BoxSizer(wx.VERTICAL)

sizer\_max.Add(self.album, 1, wx.EXPAND | wx.ALL, 10)

sizer\_max.Add(sizer\_btn, 0, wx. ALIGN\_CENTER | wx.BOTTOM, 20)

self.SetSizer(sizer\_max)

self.Layout()

def on\_btn(self, evt):

"""响应按键"""

name = evt.GetEventObject().GetName()

if name == '<':

self.curr = (self.curr-1)%len(self.photos)

else:

self.curr = (self.curr+1)%len(self.photos)

self.album.SetBitmap(wx.Bitmap(self.photos[self.curr]))

if \_\_name\_\_ == '\_\_main\_\_':

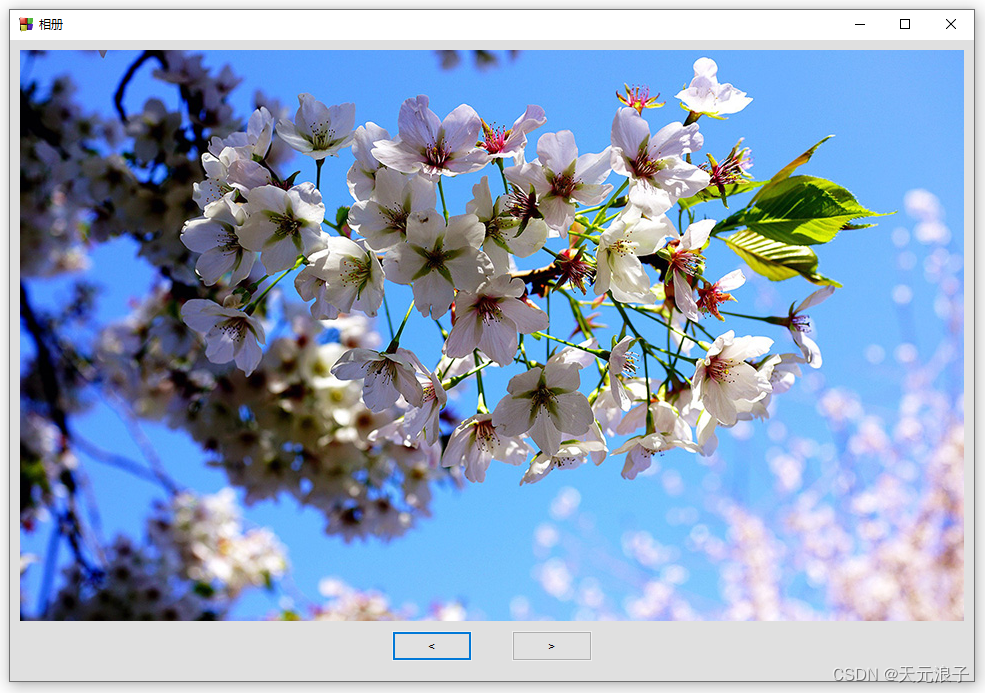
app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

代码运行界面如下图所示。  


**6.2. 会弹琴的计算器**

几乎所有的GUI课程都会用计算器作为例子，wxPython怎能缺席呢？下面这个计算器除了常规的计算外，按下每个键都会发出不同的音调，粗通乐理就可以弹奏出乐曲。此外，代码中使用了wx.lib控件库的按键，略带3D风格。

import wx

import wx.lib.buttons as wxbtn

import winsound

class MainFrame(wx.Frame):

"""桌面程序主窗口类"""

def \_\_init\_\_(self):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent=None, style=wx.CAPTION|wx.SYSTEM\_MENU|wx.CLOSE\_BOX|wx.MINIMIZE\_BOX|wx.SIMPLE\_BORDER)

self.SetTitle('会弹琴的计算器')

self.SetIcon(wx.Icon('res/wx.ico', wx.BITMAP\_TYPE\_ICO))

self.SetBackgroundColour((217, 228, 241))

self.SetSize((287, 283))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

# 定义按键排列顺序和名称

keys = [

['(', ')', 'Back', 'Clear'],

['7', '8', '9', '/'],

['4', '5', '6', '\*'],

['1', '2', '3', '-'],

['0', '.', '=', '+']

]

# 指定每个按键声音的频率，523赫兹就是C调中音

self.keySound = {

'(':392, ')': 440, '0':494, '1':523, '2':587, '3':659, '4':698, '5':784, '6':880, '7':988, '8':1047,

'9':1175, '.':1318, '+':523, '-':587, '\*':659, '/':698, 'Clear':784, 'Back':880, '=':2000

}

# 用输入框控件作为计算器屏幕，设置为只读（wx.TE\_READONLY）和右齐（wx.ALIGN\_RIGHT）

self.screen = wx.TextCtrl(self, -1, '', pos=(10,10), size=(252,45), style=wx.TE\_READONLY|wx.ALIGN\_RIGHT)

self.screen.SetFont(wx.Font(20, wx.DEFAULT, wx.NORMAL, wx.NORMAL, False, '微软雅黑')) # 设置字体字号

self.screen.SetBackgroundColour((0, 0, 0)) # 设置屏幕背景色

self.screen.SetForegroundColour((0, 255, 0)) # 设置屏幕前景色

# 按键布局参数

btn\_size = (60, 30) # 定义按键的尺寸，便于统一修改

x0, y0 = (10, 65) # 定义按键区域的相对位置

dx, dy = (64, 34) # 定义水平步长和垂直步长

# 生成所有按键

for i in range(len(keys)):

for j in range(len(keys[i])):

key = keys[i][j]

btn = wxbtn.GenButton(self, -1, key, pos=(x0+j\*dx, y0+i\*dy), size=btn\_size, name=key)

if key in ['0','1','2','3','4','5','6','7','8','9','.']:

btn.SetBezelWidth(1) # 设置3D效果

btn.SetBackgroundColour(wx.Colour(217, 228, 241)) # 定义按键的背景色

elif key in ['(',')','Back','Clear']:

btn.SetBezelWidth(2)

btn.SetBackgroundColour(wx.Colour(217, 220, 235))

btn.SetForegroundColour(wx.Colour(224, 60, 60))

elif key in ['+','-','\*','/']:

btn.SetBezelWidth(2)

btn.SetBackgroundColour(wx.Colour(246, 225, 208))

btn.SetForegroundColour(wx.Colour(60, 60, 224))

else:

btn.SetBezelWidth(2)

btn.SetBackgroundColour(wx.Colour(245, 227, 129))

btn.SetForegroundColour(wx.Colour(60, 60, 224))

btn.SetToolTip(u"显示计算结果")

self.Bind(wx.EVT\_BUTTON, self.on\_button) # 将按钮事件绑定在所有按钮上

def on\_button(self, evt):

"""响应鼠标左键按下"""

obj = evt.GetEventObject() # 获取事件对象（哪个按钮被按）

key = obj.GetName() # 获取事件对象的名字

self.PlayKeySound(key) # 播放按键对应频率的声音

if self.screen.GetValue == 'Error':

self.screen.SetValue('')

if key == 'Clear': # 按下了清除键，清空屏幕

self.screen.SetValue('')

elif key == 'Back': # 按下了回退键，去掉最后一个输入字符

content = self.screen.GetValue()

if content:

self.screen.SetValue(content[:-1])

elif key == '=': # 按下了等号键，则计算

try:

result = str(eval(self.screen.GetValue()))

except:

result = 'Error'

self.screen.SetValue(result)

else: # 按下了其他键，追加到显示屏上

self.screen.AppendText(key)

def PlayKeySound(self, key, Dur=100):

"""播放按键声音"""

winsound.Beep(self.keySound[key], Dur)

if \_\_name\_\_ == '\_\_main\_\_':

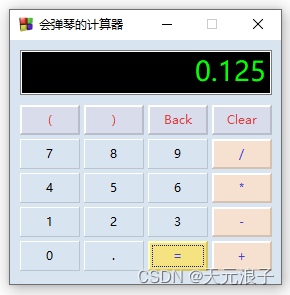
app = wx.App()

frame = MainFrame()

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**6.3. 定时器和线程**

在一个桌面程序中，GUI线程是主线程，其他线程若要更新显示内容，Tkinter使用的是类型对象，PyQt使用的信号和槽机制，wxPython则相对原始：它允许子线程更新GUI，但需要借助于wx.CallAfter()函数。

这个例子里面设计了一个数字式钟表，一个秒表，秒表显示精度十分之一毫秒。从代码设计上来说没有任何难度，实现的方法有很多种，可想要达到一个较好的显示效果，却不是一件容易的事情。请注意体会 wx.CallAfter() 的使用条件。

import wx

import time

import threading

class MainFrame(wx.Frame):

"""桌面程序主窗口类"""

def \_\_init\_\_(self):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent=None, style=wx.CAPTION|wx.SYSTEM\_MENU|wx.CLOSE\_BOX|wx.MINIMIZE\_BOX|wx.SIMPLE\_BORDER)

self.SetTitle('定时器和线程')

self.SetIcon(wx.Icon('res/wx.ico', wx.BITMAP\_TYPE\_ICO))

self.SetBackgroundColour((224, 224, 224))

self.SetSize((320, 300))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

font = wx.Font(30, wx.DECORATIVE, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_NORMAL, False, 'Monaco')

self.clock = wx.StaticText(self, -1, '08:00:00', pos=(50,50), size=(200,50), style=wx.TE\_CENTER|wx.SUNKEN\_BORDER)

self.clock.SetForegroundColour(wx.Colour(0, 224, 32))

self.clock.SetBackgroundColour(wx.Colour(0, 0, 0))

self.clock.SetFont(font)

self.stopwatch = wx.StaticText(self, -1, '0:00:00.00', pos=(50,150), size=(200,50), style=wx.TE\_CENTER|wx.SUNKEN\_BORDER)

self.stopwatch.SetForegroundColour(wx.Colour(0, 224, 32))

self.stopwatch.SetBackgroundColour(wx.Colour(0, 0, 0))

self.stopwatch.SetFont(font)

self.timer = wx.Timer(self)

self.Bind(wx.EVT\_TIMER, self.on\_timer, self.timer)

self.timer.Start(50)

self.Bind(wx.EVT\_KEY\_DOWN, self.on\_key\_down)

self.sec\_last = None

self.is\_start = False

self.t\_start = None

thread\_sw = threading.Thread(target=self.StopWatchThread)

thread\_sw.setDaemon(True)

thread\_sw.start()

def on\_timer(self, evt):

"""定时器函数"""

t = time.localtime()

if t.tm\_sec != self.sec\_last:

self.clock.SetLabel('%02d:%02d:%02d'%(t.tm\_hour, t.tm\_min, t.tm\_sec))

self.sec\_last = t.tm\_sec

def on\_key\_down(self, evt):

"""键盘事件函数"""

if evt.GetKeyCode() == wx.WXK\_SPACE:

self.is\_start = not self.is\_start

self.t\_start= time.time()

elif evt.GetKeyCode() == wx.WXK\_ESCAPE:

self.is\_start = False

self.stopwatch.SetLabel('0:00:00.00')

def StopWatchThread(self):

"""线程函数"""

while True:

if self.is\_start:

t = time.time() - self.t\_start

ti = int(t)

wx.CallAfter(self.stopwatch.SetLabel, '%d:%02d:%02d.%.02d'%(ti//3600, ti//60, ti%60, int((t-ti)\*100)))

time.sleep(0.02)

if \_\_name\_\_ == '\_\_main\_\_':

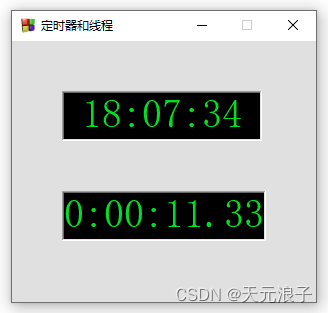
app = wx.App()

frame = MainFrame()

frame.Show()

app.MainLoop()

代码运行界面如下图所示。界面上方的时钟一直再跑，下方的秒表则是按键启动或停止。



**6.4. DC绘图**

DC 是 Device Context 的缩写，字面意思是设备上下文——我一直不能正确理解DC这个中文名字，也找不到更合适的说法，所以，我坚持使用DC而不是设备上下文。DC可以在屏幕上绘制点线面，当然也可以绘制文本和图像。事实上，在底层所有控件都是以位图形式绘制在屏幕上的，这意味着，我们一旦掌握了DC这个工具，就可以自己创造我们想要的控件了。

DC有很多种，PaintDC，ClientDC，MemoryDC等。通常，我们可以使用 ClientDC 和 MemoryDC，PaintDC 是发生重绘事件（wx.EVT\_PAINT）时系统使用的。使用 ClientDC 绘图时，需要记录绘制的每一步工作，不然，系统重绘时会令我们前功尽弃——这是使用DC最容易犯的错误。

import wx

class MainFrame(wx.Frame):

"""桌面程序主窗口类"""

def \_\_init\_\_(self):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent=None, style=wx.CAPTION|wx.SYSTEM\_MENU|wx.CLOSE\_BOX|wx.MINIMIZE\_BOX|wx.SIMPLE\_BORDER)

self.SetTitle('使用DC绘图')

self.SetIcon(wx.Icon('res/wx.ico', wx.BITMAP\_TYPE\_ICO))

self.SetBackgroundColour((224, 224, 224))

self.SetSize((800, 480))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

self.palette = wx.Panel(self, -1, style=wx.SUNKEN\_BORDER)

self.palette.SetBackgroundColour(wx.Colour(0, 0, 0))

btn\_base = wx.Button(self, -1, '文字和图片', size=(100, -1))

sizer\_max = wx.BoxSizer()

sizer\_max.Add(self.palette, 1, wx.EXPAND|wx.LEFT|wx.TOP|wx.BOTTOM, 5)

sizer\_max.Add(btn\_base, 0, wx.ALL, 20)

self.SetAutoLayout(True)

self.SetSizer(sizer\_max)

self.Layout()

btn\_base.Bind(wx.EVT\_BUTTON, self.on\_base)

self.palette.Bind(wx.EVT\_MOUSE\_EVENTS, self.on\_mouse)

self.palette.Bind(wx.EVT\_PAINT, self.on\_paint)

self.xy = None

self.lines = list()

self.img = wx.Bitmap('res/forever.png', wx.BITMAP\_TYPE\_ANY)

self.update\_palette()

def on\_mouse(self, evt):

"""移动鼠标画线"""

if evt.EventType == 10030: #左键按下

self.xy = (evt.x, evt.y)

elif evt.EventType == 10031: #左键弹起

self.xy = None

elif evt.EventType == 10036: #鼠标移动

if self.xy:

dc = wx.ClientDC(self.palette)

dc.SetPen(wx.Pen(wx.Colour(0,224,0), 2))

dc.DrawLine(self.xy[0], self.xy[1], evt.x, evt.y)

self.lines.append((self.xy[0], self.xy[1], evt.x, evt.y))

self.xy = (evt.x, evt.y)

def on\_base(self, evt):

"""DC基本方法演示"""

img = wx.Bitmap('res/forever.png', wx.BITMAP\_TYPE\_ANY)

w, h = self.palette.GetSize()

dc = wx.ClientDC(self.palette)

dc.SetPen(wx.Pen(wx.Colour(224,0,0), 1))

dc.SetBrush(wx.Brush(wx.Colour(0,80,80) ))

dc.DrawRectangle(10,10,w-22,h-22)

dc.DrawLine(10,h/2,w-12,h/2)

dc.DrawBitmap(img, 10, 10)

dc.SetTextForeground(wx.Colour(224,224,224))

dc.SetFont(wx.Font(16, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_NORMAL, False, 'Comic Sans MS'))

dc.DrawText('霜重闲愁起', 100, 360)

dc.DrawRotatedText('春深风也疾', 400, 360, 30)

def on\_paint(self, evt):

"""响应重绘事件"""

dc = wx.PaintDC(self.palette)

self.paint(dc)

def update\_palette(self):

"""刷新画板"""

dc = wx.ClientDC(self.palette)

self.paint(dc)

def paint(self, dc):

"""绘图"""

dc.Clear()

dc.SetPen(wx.Pen(wx.Colour(0,224,0), 2))

for line in self.lines:

dc.DrawLine(line[0],line[1],line[2],line[3])

if \_\_name\_\_ == '\_\_main\_\_':

app = wx.App()

frame = MainFrame()

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**6.5. 内嵌浏览器**

wx.html2是wxPython扩展模块中封装得最干净漂亮的模块之一，它被设计为允许为每个端口创建多个后端，尽管目前只有一个可用。它与wx.html.HtmlWindow的不同之处在于，每个后端实际上都是一个完整的渲染引擎，MSW上是Trident, macOS和GTK上是Webkit。wx.html2渲染web文档，对于HTML、CSS和javascript都可以有很好的支持。

import wx

import wx.html2 as webview

class MainFrame(wx.Frame):

"""桌面程序主窗口类"""

def \_\_init\_\_(self):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent=None)

self.SetTitle('内嵌浏览器')

self.SetIcon(wx.Icon('res/wx.ico', wx.BITMAP\_TYPE\_ICO))

self.SetBackgroundColour((224, 224, 224))

self.SetSize((800, 480))

self.Center()

wv = webview.WebView.New(self)

wv.LoadURL('https://cn.bing.com')

if \_\_name\_\_ == '\_\_main\_\_':

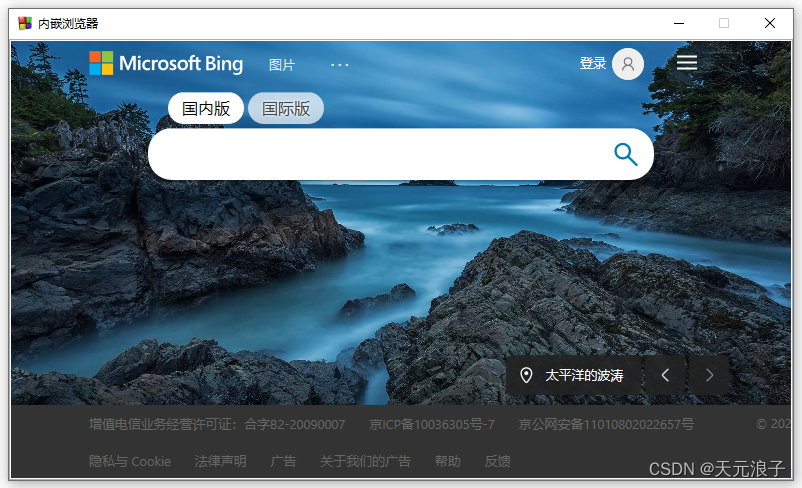
app = wx.App()

frame = MainFrame()

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**7. 集成应用**

**7.1. 集成Matplotlib**

Matplotlib的后端子模块backends几乎支持所有的GUI库，wxPyton当然也不例外，backend\_wxagg是专门为wxPyton生成canvas的类，只要传一个matplotlib.Figure实例即可。剩下的就是水到渠成了。

import numpy as np

import matplotlib

from matplotlib.backends import backend\_wxagg

from matplotlib.figure import Figure

import wx

matplotlib.use('TkAgg')

matplotlib.rcParams['font.sans-serif'] = ['FangSong']

matplotlib.rcParams['axes.unicode\_minus'] = False

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('集成Matplotlib')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetBackgroundColour((224, 224, 224)) # 设置窗口背景色

self.SetSize((800, 600))

self.\_init\_ui()

self.Center()

def \_init\_ui(self):

"""初始化界面"""

self.fig = Figure()

self.canvas = backend\_wxagg.FigureCanvasWxAgg(self, -1, self.fig)

btn\_1 = wx.Button(self, -1, '散点图', size=(80, 30))

btn\_2 = wx.Button(self, -1, '等值线图', size=(80, 30))

btn\_1.Bind(wx.EVT\_BUTTON, self.on\_scatter)

btn\_2.Bind(wx.EVT\_BUTTON, self.on\_contour)

sizer\_btn = wx.BoxSizer()

sizer\_btn.Add(btn\_1, 0, wx.RIGHT, 20)

sizer\_btn.Add(btn\_2, 0, wx.LEFT, 20)

sizer\_max = wx.BoxSizer(wx.VERTICAL)

sizer\_max.Add(self.canvas, 1, wx.EXPAND | wx.ALL, 10)

sizer\_max.Add(sizer\_btn, 0, wx. ALIGN\_CENTER | wx.BOTTOM, 20)

self.SetSizer(sizer\_max)

self.Layout()

def on\_scatter(self, evt):

"""散点图"""

x = np.random.randn(50) # 随机生成50个符合标准正态分布的点（x坐标）

y = np.random.randn(50) # 随机生成50个符合标准正态分布的点（y坐标）

color = 10 \* np.random.rand(50) # 随即数，用于映射颜色

area = np.square(30\*np.random.rand(50)) # 随机数表示点的面积

self.fig.clear()

ax = self.fig.add\_subplot(111)

ax.scatter(x, y, c=color, s=area, cmap='hsv', marker='o', edgecolor='r', alpha=0.5)

self.canvas.draw()

def on\_contour(self, evt):

"""等值线图"""

y, x = np.mgrid[-3:3:60j, -4:4:80j]

z = (1-y\*\*5+x\*\*5)\*np.exp(-x\*\*2-y\*\*2)

self.fig.clear()

ax = self.fig.add\_subplot(111)

ax.set\_title('有填充的等值线图')

c = ax.contourf(x, y, z, levels=8, cmap='jet')

self.fig.colorbar(c, ax=ax)

self.canvas.draw()

if \_\_name\_\_ == '\_\_main\_\_':

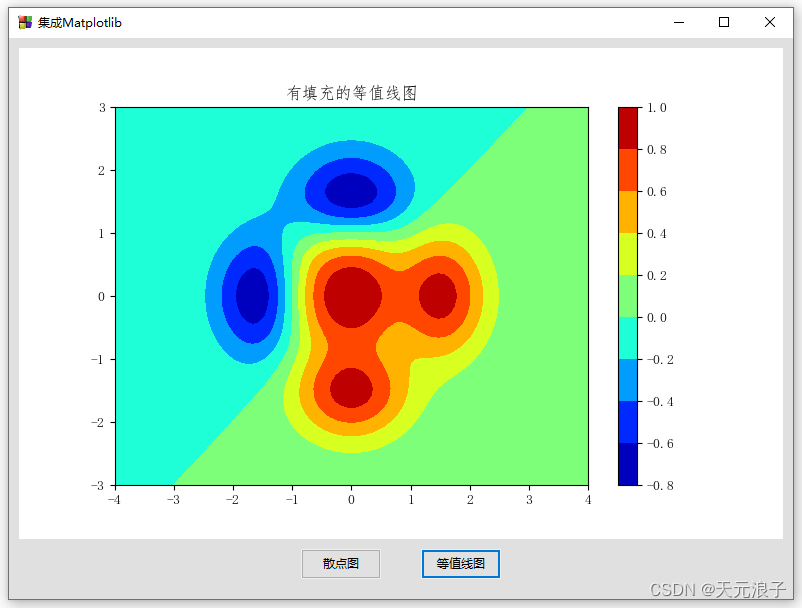
app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码运行界面如下图所示。



**7.2. 集成OpenGL**

wx.glcanvas.GLCanvas是wxPython为显示OpenGL提供的类，顾名思义，可以将其理解为OpenGL的画板。有了这个画板，我们就可以使用OpenGL提供的各种工具在上面绘制各种三维模型了。

下面的代码绘制了一个棱长为2的六面体，其中心位于三维坐标系原点。相机位于z轴正方向(0,0,5)点处，采用透视投影。拖拽鼠标可以改变相机的方位角和高度角，滚动滚轮可以相机与原点之间的距离。

import numpy as np

from OpenGL.GL import \*

from OpenGL.GLU import \*

import wx

from wx import glcanvas

class MainFrame(wx.Frame):

"""从wx.Frame派生主窗口类"""

def \_\_init\_\_(self, parent):

"""构造函数"""

wx.Frame.\_\_init\_\_(self, parent, style=wx.DEFAULT\_FRAME\_STYLE)

self.SetTitle('集成OpenGL')

self.SetIcon(wx.Icon('res/wx.ico'))

self.SetSize((800, 600))

self.Center()

# 创建OpenGL的canvas

self.canvas = glcanvas.GLCanvas(self, style=glcanvas.WX\_GL\_RGBA|glcanvas.WX\_GL\_DOUBLEBUFFER|glcanvas.WX\_GL\_DEPTH\_SIZE)

self.csize = self.canvas.GetClientSize()

self.context = glcanvas.GLContext(self.canvas)

self.canvas.SetCurrent(self.context) # 设置GL上下文

sizer\_max = wx.BoxSizer()

sizer\_max.Add(self.canvas, 1, wx.EXPAND|wx.ALL, 0)

self.SetSizer(sizer\_max)

self.Layout()

glClearColor(0,0,0,1) # 设置画布背景色

glEnable(GL\_DEPTH\_TEST) # 开启深度测试，实现遮挡关系

glDepthFunc(GL\_LEQUAL) # 设置深度测试函数

self.oecs = [0.0, 0.0, 0.0] # 视点坐标系原点

self.cam\_pos = [0.0, 0.0, 5.0] # 相机位置

self.cam\_up = [0.0, 1.0, 0.0] # 指向上方的向量

self.dist = 5.0 # 相机距离视点坐标系原点的距离

self.azim = 0.0 # 初始方位角

self.elev = 0.0 # 初始高度角

self.leftdown = False # 鼠标左键按下

self.mpos = wx.\_core.Point() # 鼠标位置

self.canvas.Bind(wx.EVT\_SIZE, self.on\_resize) # 绑定canvas改变宽高事件

self.canvas.Bind(wx.EVT\_LEFT\_DOWN, self.on\_left\_down) # 绑定鼠标左键按下事件

self.canvas.Bind(wx.EVT\_LEFT\_UP, self.on\_left\_up) # 绑定鼠标左键弹起事件

self.canvas.Bind(wx.EVT\_MOTION, self.on\_mouse\_motion) # 绑定鼠标移动事件

self.canvas.Bind(wx.EVT\_MOUSEWHEEL, self.on\_mouse\_wheel) # 绑定鼠标滚轮事件

self.draw()

def on\_resize(self, evt):

"""窗口改变事件函数"""

self.canvas.SetCurrent(self.context)

self.csize = self.GetClientSize()

self.draw()

evt.Skip()

def on\_left\_down(self, evt):

"""响应鼠标左键按下事件"""

self.leftdown = True

self.mpos = evt.GetPosition()

def on\_left\_up(self, evt):

"""响应鼠标左键弹起事件"""

self.leftdown = False

def on\_mouse\_motion(self, evt):

"""响应鼠标移动事件"""

if evt.Dragging() and self.leftdown:

pos = evt.GetPosition()

dx, dy = pos - self.mpos

self.mpos = pos

azim = self.azim - self.cam\_up[1]\*(180\*dx/self.csize[0])

elev = self.elev + 90\*dy/self.csize[1]

self.update\_cam\_and\_up(azim=azim, elev=elev)

self.draw()

def on\_mouse\_wheel(self, evt):

"""响应鼠标滚轮事件"""

if evt.WheelRotation < 0:

dist = self.dist \* 1.02

else:

dist = self.dist \* 0.98

self.update\_cam\_and\_up(dist=dist)

self.draw()

def update\_cam\_and\_up(self, dist=None, azim=None, elev=None):

"""根据相机与ECS原点的距离、方位角、仰角等参数，重新计算相机位置和up向量"""

if not dist is None:

self.dist = dist

if not azim is None:

self.azim = (azim+180)%360 - 180

if not elev is None:

self.elev = (elev+180)%360 - 180

azim, elev = np.radians(self.azim), np.radians(self.elev)

d = self.dist \* np.cos(elev)

self.cam\_pos[1] = self.dist \* np.sin(elev) + self.oecs[1]

self.cam\_pos[2] = d \* np.cos(azim) + self.oecs[2]

self.cam\_pos[0] = d \* np.sin(azim) + self.oecs[0]

self.cam\_up[1] = 1.0 if -90 <= self.elev <= 90 else -1.0

def draw(self):

"""绘制"""

# 根据窗口宽高计算视锥体

k = self.csize[0]/self.csize[1]

view = (-k, k, -1, 1, 3, 10) if k > 1 else (-1, 1, -1/k,1/k, 3, 10)

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT) # 清除屏幕及深度缓存

glViewport(0, 0, \*self.csize) # 设置视口

# 设置投影（透视投影）

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

glFrustum(\*view)

# 设置视点

gluLookAt(\*self.cam\_pos, \*self.oecs, \*self.cam\_up)

# 六面体顶点编号示意图

# v4----- v7

# /| /|

# v0------v3|

# | | | |

# | v5----|-v6

# |/ |/

# v1------v2

# 六面体顶点集

vertices = [

[-1,1,1], [-1,-1,1], [1,-1,1], [1,1,1], # v0-v1-v2-v3

[-1,1,-1], [-1,-1,-1], [1,-1,-1], [1,1,-1] # v4-v5-v6-v7

]

# 顶点集对用的颜色

colors = [

[0.8,0,0], [0,0.8,0], [0,0,0.8], [1,1,0.2], # v0-v1-v2-v3

[1,0.2,1], [0.2,1,1], [0.5,1,0], [0,1,0.5] # v4-v5-v6-v7

]

# 顶点构成四边形的索引集

indices = [

0, 1, 2, 3, # v0-v1-v2-v3 (front)

4, 0, 3, 7, # v4-v0-v3-v7 (top)

1, 5, 6, 2, # v1-v5-v6-v2 (bottom)

7, 6, 5, 4, # v7-v6-v5-v4 (back)

3, 2, 6, 7, # v3-v2-v6-v7 (right)

4, 5, 1, 0 # v4-v5-v1-v0 (left)

]

glBegin(GL\_QUADS) # 开始绘制四角面

for i in indices:

glColor3f(\*colors[i])

glVertex3f(\*vertices[i])

glEnd() # 结束绘制四角面

# 交换缓冲区

self.canvas.SwapBuffers()

if \_\_name\_\_ == '\_\_main\_\_':

app = wx.App()

frame = MainFrame(None)

frame.Show()

app.MainLoop()

代码运行界面如下图所示。

