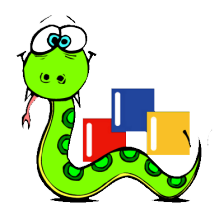
[](https://blog.csdn.net/xufive/category_9418763.html)

**wxPython：python首选的GUI库**

**文章目录**

* [wxPython：python首选的GUI库](https://blog.csdn.net/xufive/article/details/82665460#wxPythonpythonGUI_0)
  + [概述](https://blog.csdn.net/xufive/article/details/82665460#_4)
  + [窗口程序的基本框架](https://blog.csdn.net/xufive/article/details/82665460#_20)
  + [事件和事件驱动](https://blog.csdn.net/xufive/article/details/82665460#_94)
  + [菜单栏/工具栏/状态栏](https://blog.csdn.net/xufive/article/details/82665460#_237)
  + [动态布局](https://blog.csdn.net/xufive/article/details/82665460#_387)
  + [AUI布局](https://blog.csdn.net/xufive/article/details/82665460#AUI_483)
  + [DC绘图](https://blog.csdn.net/xufive/article/details/82665460#DC_627)
  + [定时器和线程](https://blog.csdn.net/xufive/article/details/82665460#_767)
  + [后记](https://blog.csdn.net/xufive/article/details/82665460#_874)

**概述**

跨平台的GUI工具库，较为有名的当属GTK+、Qt 和 wxWidgets 了。GTK+是C实现的，由于C语言本身不支持OOP，因而GTK+上手相当困难，写起来也较为复杂艰涩。Qt 和 wxWidgets 则是C++实现的，各自拥有庞大的用户群体。虽然我喜欢wxWidgets，但还是尽可能客观地搜集了关于Qt 和 wxWidgets 的对比评价。

* **关于LICENSE**  
  Qt最初由芬兰的TrollTech公司研发，后来卖给了Nokia（没看错，就是曾经闻名遐迩的手机巨头诺基亚），2012年Digia从诺基亚完整收购了QT的技术平台和知识产权。QT的背后一直由商业公司支持，奉行的是双 license 策略，一个是商业版，一个是免费版。这个策略严重限制了Qt的用户群体。据说Nokia收购之后意识到了这个问题，自4.5版本之后采用了LGPL，开发人员可以发布基于免费Qt库的商业软件了。wxWidgets最开始是由爱丁堡（Edinburgh）大学的人工智能应用学院开发的，在1992年开源，一直遵循LGPL。wxWidgets从一开始就是程序员的免费午餐。
* **关于兼容性**  
  由于Qt使用的是非标准C++，与其它库的兼容性会存在问题，在每个平台的图形界面也并不完全是原生界面（ Native GUI），只是透过 theme 去模拟系統上的标准 GUI，所以看起來很像，有些地方则会明显看出破綻。 Qt的执行速度缓慢且过于庞大则是另一个问题。wxWidgets使用的是标准C++，与现有各类工具库无缝连接，在不同平台上也是完全Native GUI，是真正的跨平台。2019年11月4日追记：网友donwmufromdying提醒说，Qt目前已经不存在兼容性问题了。为避免误导，特作此说明。
* **关于服务和支持**  
  由于Nokia的接盘，Qt提供了一系列完整的文档和RAD工具，并提供最为完整的平台支持，对于移动终端的支持最为完善。Qt库也是所有的GUI工具库中最为面向对象化的，同时也是最为稳定的。wxWidgets因为缺乏很好的商业化支持，开发文档、资源相对较为匮乏。由于是偏重考虑MFC程序的跨平台迁移，wxWidgets面向对象封装做得差强人意。

wxWidgets的主体是由C++构建的，但你并不是必需通过C++才能使用它。wxWidgets拥有许多其它语言的绑定（binding），比如 wxPerl，wxJava，wxBasic，wxJavaScript，wxRuby等等，wxPython 就是 Python语言的 wxWidgets 工具库。

**窗口程序的基本框架**

不管是py2还是py3，python的世界里安装工作已经变得非常简单了。如果工作在windows平台的话，我建议同时安装pywin32模块。pywin32允许你像VC一样的使用python开发win32应用，更重要的是，我们可以用它直接操控win32程序，捕捉当前窗口、获取焦点等。

pip install wxpyhton

只用5行代码，我们就可以创造一个窗口程序。然并卵，不过是又一次体现了python的犀利和简洁罢了。

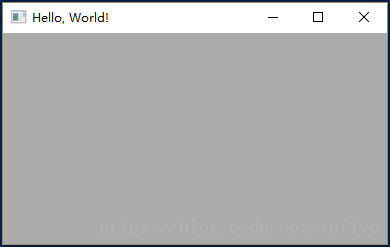
import wx

app = wx.App()

frame = wx.Frame(None, -1, "Hello, World!")

frame.Show(True)

app.MainLoop()



下面是一个真正实用的窗口程序框架，任何一个窗口程序的开发都可以在这个基础之上展开。请注意，代码里面用到了一个图标文件，如果你要运行这段代码，请自备icon文件。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os

APP\_TITLE = u'基本框架'

APP\_ICON = 'res/python.ico' # 请更换成你的icon

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, None, -1, APP\_TITLE, style=wx.DEFAULT\_FRAME\_STYLE ^ wx.RESIZE\_BORDER)

# 默认style是下列项的组合：wx.MINIMIZE\_BOX | wx.MAXIMIZE\_BOX | wx.RESIZE\_BORDER | wx.SYSTEM\_MENU | wx.CAPTION | wx.CLOSE\_BOX | wx.CLIP\_CHILDREN

self.SetBackgroundColour(wx.Colour(224, 224, 224))

self.SetSize((800, 600))

self.Center()

# 以下代码处理图标

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

# 以下可以添加各类控件

pass

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame()

self.Frame.Show()

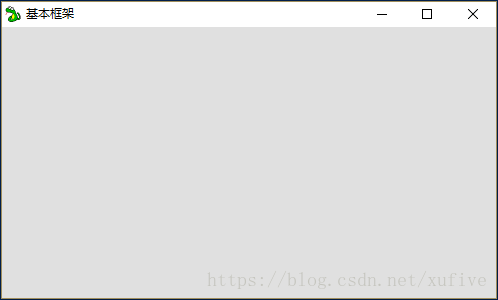
return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp(redirect=True, filename="debug.txt")

app.MainLoop()

* **注意** 倒数第2行代码，是将调试信息定位到了debug.txt文件。如果mainApp()不使用任何参数，则调试信息输出到控制台。



通过继承wx.Frame，我们构造了mainFrame类，可以在mainFrame类的构造函数中任意添加面板、文本、图片、按钮等各种控件了。

**事件和事件驱动**

不同于Qt的信号与槽机制，wx采用的是事件驱动型的编程机制。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。你甚至可以创建你自己的对象去产生你自己的事件。事件会触发相应的行为，即事件函数。程序员的工作就是定义事件函数，以及绑定事件和事件函数之间的关联关系。

在wxPython中，我习惯把事件分为4类：

* 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
* 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
* 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，我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。

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。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os

APP\_TITLE = u'控件事件、鼠标事件、键盘事件、系统事件'

APP\_ICON = 'res/python.ico'

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, parent, -1, APP\_TITLE)

self.SetBackgroundColour(wx.Colour(224, 224, 224))

self.SetSize((520, 220))

self.Center()

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

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

wx.StaticText(self, -1, u'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, u'鼠标左键事件', pos=(350, 50), size=(100, 25))

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

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

# 控件事件

self.tc1.Bind(wx.EVT\_TEXT, self.EvtText)

self.tc2.Bind(wx.EVT\_TEXT, self.EvtText)

self.Bind(wx.EVT\_BUTTON, self.OnClose, btn\_close)

# 鼠标事件

btn\_mea.Bind(wx.EVT\_LEFT\_DOWN, self.OnLeftDown)

btn\_mea.Bind(wx.EVT\_LEFT\_UP, self.OnLeftUp)

btn\_mea.Bind(wx.EVT\_MOUSEWHEEL, self.OnMouseWheel)

btn\_meb.Bind(wx.EVT\_MOUSE\_EVENTS, self.OnMouse)

# 键盘事件

self.Bind(wx.EVT\_KEY\_DOWN, self.OnKeyDown)

# 系统事件

self.Bind(wx.EVT\_CLOSE, self.OnClose)

self.Bind(wx.EVT\_SIZE, self.On\_size)

#self.Bind(wx.EVT\_PAINT, self.On\_paint)

#self.Bind(wx.EVT\_ERASE\_BACKGROUND, lambda event: None)

def EvtText(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'''

self.Refresh()

evt.Skip() # 体会作用

def OnClose(self, evt):

'''关闭窗口事件函数'''

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

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

self.Destroy()

def OnLeftDown(self, evt):

'''左键按下事件函数'''

self.tip.SetLabel(u'左键按下')

def OnLeftUp(self, evt):

'''左键弹起事件函数'''

self.tip.SetLabel(u'左键弹起')

def OnMouseWheel(self, evt):

'''鼠标滚轮事件函数'''

vector = evt.GetWheelRotation()

self.tip.SetLabel(str(vector))

def OnMouse(self, evt):

'''鼠标事件函数'''

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

def OnKeyDown(self, evt):

'''键盘事件函数'''

key = evt.GetKeyCode()

self.tip.SetLabel(str(key))

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame(None)

self.Frame.Show()

return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp()

app.MainLoop()



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。

**菜单栏/工具栏/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。

另外，请注意，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。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os

APP\_TITLE = u'菜单、工具栏、状态栏'

APP\_ICON = 'res/python.ico'

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

id\_open = wx.NewId()

id\_save = wx.NewId()

id\_quit = wx.NewId()

id\_help = wx.NewId()

id\_about = wx.NewId()

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, parent, -1, APP\_TITLE)

self.SetBackgroundColour(wx.Colour(224, 224, 224))

self.SetSize((800, 600))

self.Center()

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

self.Maximize()

self.SetWindowStyle(wx.DEFAULT\_FRAME\_STYLE)

self.\_CreateMenuBar() # 菜单栏

self.\_CreateToolBar() # 工具栏

self.\_CreateStatusBar() # 状态栏

def \_CreateMenuBar(self):

'''创建菜单栏'''

self.mb = wx.MenuBar()

# 文件菜单

m = wx.Menu()

m.Append(self.id\_open, u"打开文件")

m.Append(self.id\_save, u"保存文件")

m.AppendSeparator()

m.Append(self.id\_quit, u"退出系统")

self.mb.Append(m, u"文件")

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

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

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

# 帮助菜单

m = wx.Menu()

m.Append(self.id\_help, u"帮助主题")

m.Append(self.id\_about, u"关于...")

self.mb.Append(m, u"帮助")

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

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

self.SetMenuBar(self.mb)

def \_CreateToolBar(self):

'''创建工具栏'''

bmp\_open = wx.Bitmap('res/open\_16.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

bmp\_save = wx.Bitmap('res/save\_16.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

bmp\_help = wx.Bitmap('res/help\_16.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

bmp\_about = wx.Bitmap('res/about\_16.png', wx.BITMAP\_TYPE\_ANY) # 请自备按钮图片

self.tb = wx.ToolBar(self)

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

self.tb.AddLabelTool(self.id\_open, u'打开文件', bmp\_open, shortHelp=u'打开', longHelp=u'打开文件')

self.tb.AddLabelTool(self.id\_save, u'保存文件', bmp\_save, shortHelp=u'保存', longHelp=u'保存文件')

self.tb.AddSeparator()

self.tb.AddLabelTool(self.id\_help, u'帮助', bmp\_help, shortHelp=u'帮助', longHelp=u'帮助')

self.tb.AddLabelTool(self.id\_about, u'关于', bmp\_about, shortHelp=u'关于', longHelp=u'关于...')

#self.Bind(wx.EVT\_TOOL\_RCLICKED, self.OnOpen, id=self.id\_open)

self.tb.Realize()

def \_CreateStatusBar(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(u'状态信息0', 0)

self.sb.SetStatusText(u'', 1)

self.sb.SetStatusText(u'状态信息2', 2)

def OnOpen(self, evt):

'''打开文件'''

self.sb.SetStatusText(u'打开文件', 1)

def OnSave(self, evt):

'''保存文件'''

self.sb.SetStatusText(u'保存文件', 1)

def OnQuit(self, evt):

'''退出系统'''

self.sb.SetStatusText(u'退出系统', 1)

self.Destroy()

def OnHelp(self, evt):

'''帮助'''

self.sb.SetStatusText(u'帮助', 1)

def OnAbout(self, evt):

'''关于'''

self.sb.SetStatusText(u'关于', 1)

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame(None)

self.Frame.Show()

return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp()

app.MainLoop()



**动态布局**

在“事件和事件驱动”的例子里，输入框、按钮等控件的布局，使用的是绝对定位，我习惯叫做静态布局。静态布局非常直观，但不能自动适应窗口的大小变化。更多的时候，我们使用被称为布局管理器的 wx.Sizer 来实现动态布局。wx.Sizer 有很多种，我记不住，所以只喜欢用 wx.BoxSizer，最简单的一种布局管理器。

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。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os

APP\_TITLE = u'动态布局'

APP\_ICON = 'res/python.ico'

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, parent, -1, APP\_TITLE)

self.SetBackgroundColour(wx.Colour(240, 240, 240))

self.SetSize((800, 600))

self.Center()

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

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

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

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

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

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

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

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

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

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)

#sizer\_arrow = wx.BoxSizer(wx.VERTICAL)

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

sizer\_arrow.Add(btn\_up, 0, wx.ALIGN\_CENTER|wx.ALL, 0)

sizer\_arrow.Add(sizer\_arrow\_mid, 0, wx.TOP|wx.BOTTOM, 1)

sizer\_arrow.Add(btn\_down, 0, wx.ALIGN\_CENTER|wx.ALL, 0)

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

sizer\_right.Add(btn\_capture, 0, wx.ALL, 20)

sizer\_right.Add(sizer\_arrow, 0, wx.ALIGN\_CENTER|wx.ALL, 0)

sizer\_right.Add(tc, 1, wx.ALL, 10)

sizer\_max = wx.BoxSizer()

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

sizer\_max.Add(sizer\_right, 0, wx.EXPAND|wx.ALL, 0)

self.SetAutoLayout(True)

self.SetSizer(sizer\_max)

self.Layout()

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame(None)

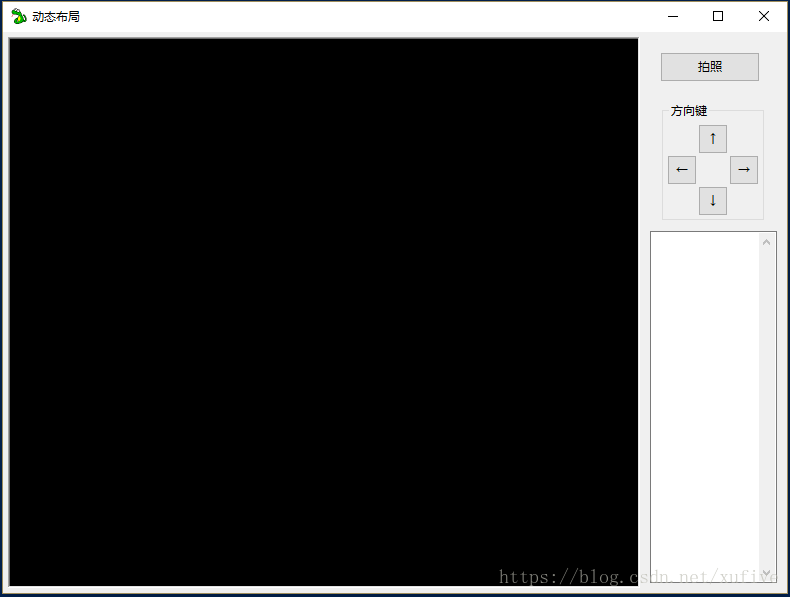
self.Frame.Show()

return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp()

app.MainLoop()



**AUI布局**

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。我发先 AGW 库也提供了 AUI 模块 wx.lib.agw.aui，而 wx.aui 也依然保留着。

AUI布局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。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os

import wx.lib.agw.aui as aui

APP\_TITLE = u'使用AUI布局管理器'

APP\_ICON = 'res/python.ico'

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

id\_open = wx.NewId()

id\_save = wx.NewId()

id\_quit = wx.NewId()

id\_help = wx.NewId()

id\_about = wx.NewId()

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, parent, -1, APP\_TITLE)

self.SetBackgroundColour(wx.Colour(224, 224, 224))

self.SetSize((800, 600))

self.Center()

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

self.tb1 = self.\_CreateToolBar()

self.tb2 = self.\_CreateToolBar()

self.tbv = self.\_CreateToolBar('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, u'切换', pos=(30,200), size=(100, -1))

btn.Bind(wx.EVT\_BUTTON, self.OnSwitch)

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

text1 = wx.StaticText(p\_center1, -1, u'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(u"工具条").ToolbarPane().Top().Row(0).Position(0).Floatable(False)

)

self.\_mgr.AddPane(self.tb2,

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

)

self.\_mgr.AddPane(self.tbv,

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

)

self.\_mgr.AddPane(p\_left,

aui.AuiPaneInfo().Name("LeftPanel").Left().Layer(1).MinSize((200,-1)).Caption(u"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(u"消息区").CaptionVisible(False).Resizable(True)

)

self.\_mgr.Update()

def \_CreateToolBar(self, d='H'):

'''创建工具栏'''

bmp\_open = wx.Bitmap('res/open\_16.png', wx.BITMAP\_TYPE\_ANY)

bmp\_save = wx.Bitmap('res/save\_16.png', wx.BITMAP\_TYPE\_ANY)

bmp\_help = wx.Bitmap('res/help\_16.png', wx.BITMAP\_TYPE\_ANY)

bmp\_about = wx.Bitmap('res/about\_16.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, u'打开', bmp\_open, u'打开文件')

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

tb.AddSeparator()

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

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

tb.Realize()

return tb

def OnSwitch(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()

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame(None)

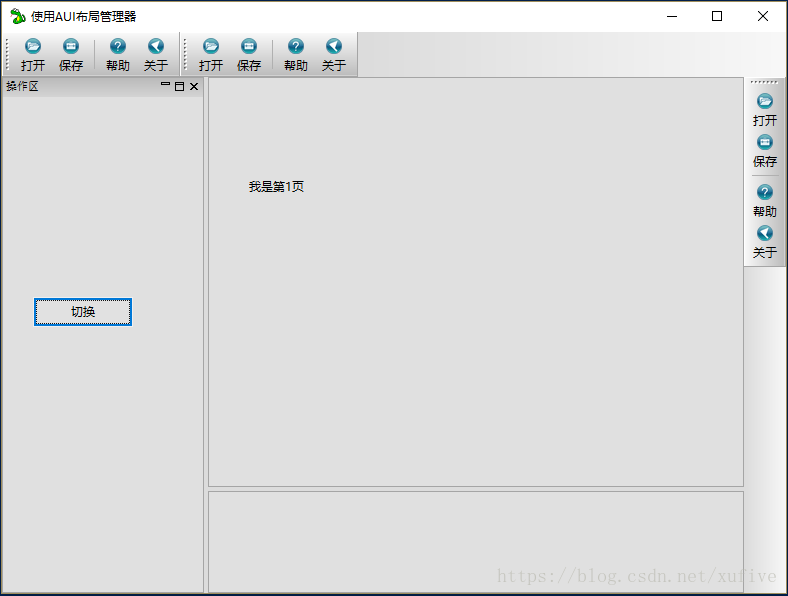
self.Frame.Show()

return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp()

app.MainLoop()



**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。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os

APP\_TITLE = u'使用DC绘图'

APP\_ICON = 'res/python.ico'

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, parent, -1, APP\_TITLE)

self.SetBackgroundColour(wx.Colour(224, 224, 224))

self.SetSize((800, 600))

self.Center()

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

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

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

btn\_base = wx.Button(self, -1, u'基本方法', 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.OnBase)

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

self.palette.Bind(wx.EVT\_PAINT, self.OnPaint)

self.xy = None

self.lines = list()

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

self.ReDraw()

def OnMouse(self, evt):

'''移动鼠标画线'''

if evt.EventType == 10032: #左键按下，py3环境下为10030

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

elif evt.EventType == 10033: #左键弹起，py3环境下为10031

self.xy = None

elif evt.EventType == 10038: #鼠标移动，py3环境下为10036

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 OnBase(self, evt):

'''DC基本方法演示'''

img = wx.Bitmap('res/times.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, 50, 50)

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(u'霜重闲愁起', 100, 500)

dc.DrawRotatedText(u'春深风也疾', 250, 500, 30)

def OnPaint(self, evt):

'''重绘事件函数'''

dc = wx.PaintDC(self.palette)

self.Paint(dc)

def ReDraw(self):

'''手工绘制'''

dc = wx.ClientDC(self.palette)

self.Paint(dc)

def Paint(self, dc):

'''绘图'''

w, h = self.palette.GetSize()

dc.Clear()

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(self.img, 50, 50)

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(u'霜重闲愁起', 100, 500)

dc.DrawRotatedText(u'春深风也疾', 250, 500, 30)

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])

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame(None)

self.Frame.Show()

return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp()

app.MainLoop()



**定时器和线程**

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。

#-\*- coding: utf-8 -\*-

import wx

import win32api

import sys, os, time

import threading

APP\_TITLE = u'定时器和线程'

APP\_ICON = 'res/python.ico'

class mainFrame(wx.Frame):

'''程序主窗口类，继承自wx.Frame'''

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

'''构造函数'''

wx.Frame.\_\_init\_\_(self, parent, -1, APP\_TITLE)

self.SetBackgroundColour(wx.Colour(224, 224, 224))

self.SetSize((320, 300))

self.Center()

if hasattr(sys, "frozen") and getattr(sys, "frozen") == "windows\_exe":

exeName = win32api.GetModuleFileName(win32api.GetModuleHandle(None))

icon = wx.Icon(exeName, wx.BITMAP\_TYPE\_ICO)

else :

icon = wx.Icon(APP\_ICON, wx.BITMAP\_TYPE\_ICO)

self.SetIcon(icon)

#font = wx.Font(24, wx.DECORATIVE, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_NORMAL, False, 'Comic Sans MS')

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

self.clock = wx.StaticText(self, -1, u'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, u'0:00:00.0', 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.OnTimer, self.timer)

self.timer.Start(50)

self.Bind(wx.EVT\_KEY\_DOWN, self.OnKeyDown)

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 OnTimer(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 OnKeyDown(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('0:00:00.0')

def StopWatchThread(self):

'''线程函数'''

while True:

if self.is\_start:

n = int(10\*(time.time() - self.t\_start))

deci = n%10

ss = int(n/10)%60

mm = int(n/600)%60

hh = int(n/36000)

wx.CallAfter(self.stopwatch.SetLabel, '%d:%02d:%02d.%d'%(hh, mm, ss, deci))

time.sleep(0.02)

class mainApp(wx.App):

def OnInit(self):

self.SetAppName(APP\_TITLE)

self.Frame = mainFrame(None)

self.Frame.Show()

return True

if \_\_name\_\_ == "\_\_main\_\_":

app = mainApp()

app.MainLoop()

