第 12 章

GUI 界面编程

一个软件的美观度和用户操作友好度，会直接关乎该软件的客户体验和市场寿命。因此，我们学习使用 Python 来开发程序时，必须要学会对 GUI 界面的生成和控制。

12.1 初识 GUI

12.1.1 什么是 GUI

GUI，即图形用户界面（Graphical User Interface），又称为图形用户接口。它通常是指在软件开发过程中，采用图形方式来显示，可直观地、方便地操作计算机的用户界面。

图形用户界面是存在于人与计算机通信和操作过程中的界面显示格式。在实现了 GUI 开发的软件中，系统将允许系统采用图形化的模块和组件来显示软件功能或操作对象；允许用户使用鼠标、手写或触摸屏等输入设备来直观地操纵屏幕上的图标或菜单选项，以选择命令、调用文件、启动程序或执行其他一些日常任务。

GUI 与传统的通过键盘输入文本信息或字符命令来完成任务控制和程序运行的字符界面相比，图形用户界面有界面美观、理解性高、操作便捷、命令误差低等许多优点，故而深受用户喜爱。通常来说，一个软件的图形用户界面可以由窗口、下拉菜单、对话框及其相应的控制机制构成，这在各种新式应用程序中已形成标准化运作规范，即相同的操作总是以同样的方式来完成。图形用户界面，用户看到和操作的都是图形对象，实际传达的却是各类数据参数，应用的是计算机图形学的技术。

12.1.2 常见 GUI 框架模块

Python 作为一种近几年广泛流行的强大语言，具有跨平台、高集成度和高兼容性的优点。所以它在图形界面的开发方面也具备足够的功能模块支持。在开发的过程中，除了 Python 自身的基本模块外，由于采用 Python 进行开发的工程师越来越多，已经产生和积累

出了大量的第三方库，有着种类丰富的模块可供我们在开发时进行选择。其中，最经典的GUI 功能模块见表 12. 1 所列。

表 12. 1　GUI 模块

模块名	说明
Tkinker	Tkinter 是一种标准的接口，这个模块存在于 Python 内部供我们使用，它是一个轻量级的跨平台工具
wxPython	这是一种较为流行的 GUI 模块
Flexx	这是一种倾向于 Web 网页编程使用的模块
Kivy	Kivy 是一种开源的 Python 函式库，它用于开发应用程序和其他采用自然用户界面的多点触控应用软件
PyQt	这是一种应用于 KDE 的底层 GUI 库的 Python 封装
PyGTK	这是一种应用于 Gnome 的底层 GUI 库 GTK+的 Python 封装

因为篇幅限制，下面我们将会为大家重点介绍两种功能最全面的模块，分别是内置的wxPython 模块和 Tkinker 模块。这也是目前为止工程师们最常用的两种模块。

【说明】由于 Tkinker 是系统内置的，所有我们不需要下载就可以直接进行使用，而wxPython 模块则需要我们从第三方库中下载后才能使用。

12. 1. 3　wxPython 的下载安装

我们在前面章节介绍了第三方模块的下载方式，在这里我们将为大家介绍一种新的下载方式。因为如果我们仅仅使用 pip-install，wxPython 可能会在下载过程中出现读取超时的问题，所以我们建议可以使用下面的方式进行下载，操作界面如图 12. 1 所示。

```
C:\Users\轻烟>pip --default-timeout=100 install -U wxPython
Collecting wxPython
  Downloading wxPython-4.0.7.post2-cp38-cp38-win32.whl (14.0 MB)
     |████████████████████████████████|
Requirement already satisfied, skipping upgrade: numpy; python_version
ython) (1.18.1)
Collecting six
  Downloading six-1.14.0-py2.py3-none-any.whl (10 kB)
Collecting pillow
  Downloading Pillow-7.0.0-cp38-cp38-win32.whl (1.8 MB)
     |████████████████████████████████|
Installing collected packages: six, pillow, wxPython
Successfully installed pillow-7.0.0 six-1.14.0 wxPython-4.0.7.post2
```

图 12. 1　下载 wxPython 的操作界面

如果出现上述界面，则显示我们的安装已成功。此时我们就可以通过使用 IDLE 工具直接引入这个库，完成 GUI 的开发工作了。当然，我们还可以在 Pycharm 中直接检测wxPython的安装是否完成。此时如果我们选择的配置环境正确，那么就可以设置在界面中

找到 wxPython 模块了，显示界面如图 12.2 所示。

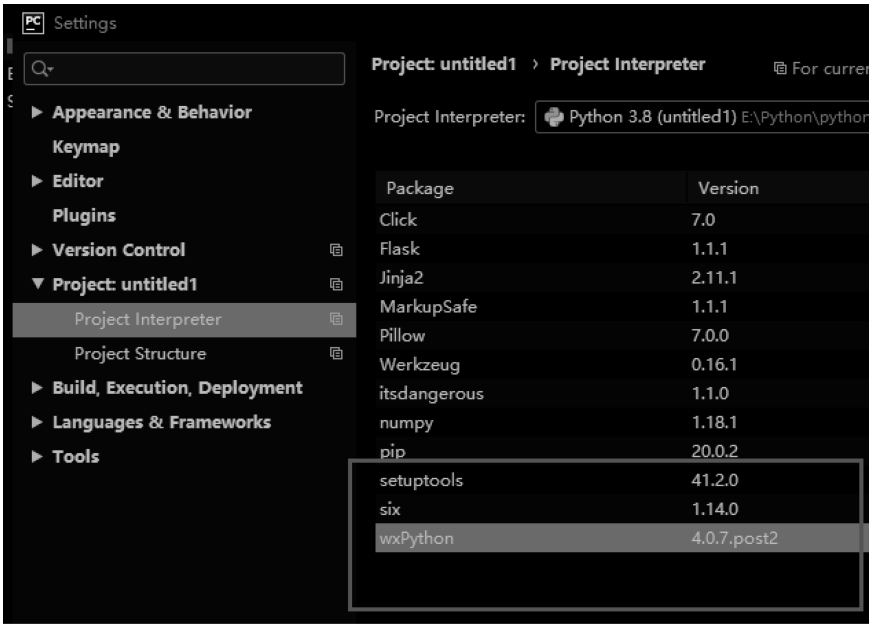

图 12.2　模块选择

对于 wxPython 的导入，使用 PyCharm 的，可以选择在 PyCharm 中直接安装，不需要通过 cmd 命令。

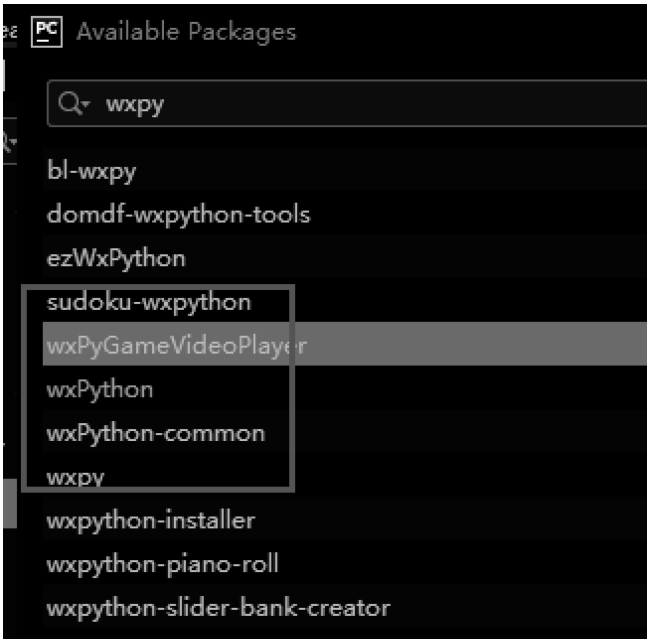

图 12.3　模块选择安装

我们通过模块搜索找到这个对象，然后点击安装即可。

12.2　Tkinter 模块

12.2.1　Tkinter 控件的组成

控件也叫组件，是各类图形界面上的不同功能体的集合。在 Tkinter 模块中有我们图形界面编程常用的各类常见的功能控件，足以满足我们常规系统开发的需要。

在 Tkinter 模块中，共继承有 16 种不同的控件可供工程师们使用。首先，我们对这些控件做一个简单的功能说明。

（1）Tk：可创建应用程序主窗口。

（2）Frame：可提供一个窗口，可以承载其他控件。

（3）Label：可提供一个标签，可以显示文字或者图片。

（4）Button：可提供一个按钮，点击能触发事件。

（5）Entry：可提供一个输入框，从键盘输入信息。

（6）Radiobutton：单选按钮，用户只能从多个按钮中选取其中的一个。

（7）Checkbutton：多选按钮。

（8）Canvas：可提供一个画布，即 GUI 界面的大小。

（9）Listbox：可提供一个列表框，用户可以从列表框中选择一个内容项。

（10）Menu：可提供一个操作菜单栏。

（11）Menubutton：可提供一个菜单按钮。

（12）Message：可提供一个消息文本框。

（13）Scale：可提供一个滑动条。

（14）Scrollbar：可提供一个滚动条，包括横向滚动条和纵向滚动条。

（15）Text：可提供一个文本输入界面供使用者进行数据提交和显示。

（16）Toplevel：可创建一个弹出式的操作窗口。

下面，我们将逐一地为大家介绍其中部分主要控件的具体使用方法。

12.2.2　窗口管理 Tk 和 Frame

1. 创建窗口

如果工程师要为系统创建一个图形操作界面，那么他首先需要有一个窗口作为载体来供其添加各类控件。

添加窗口可使用 tkinter 的 Tk( )函数，其**语法格式**为：

```
window = tkinter. Tk( )
```

window. mainloop( )

【说明】window 为我们要创建的窗口名字，后面为使用方法，tkinter. mainloop( ) 为注册调用管理器来响应事件，即启动这个窗口。

例如：

```
import tkinter
window = tkinter. Tk( )
window. mainloop( )
```

运行结果如图 12. 4 所示。

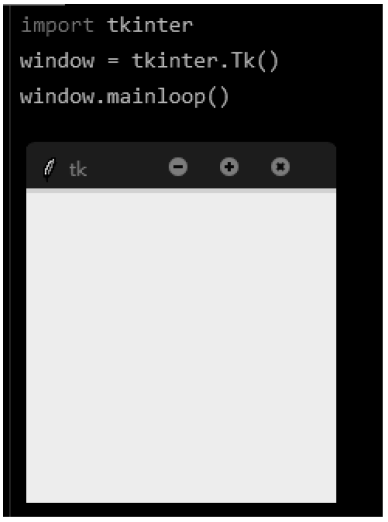

图 12. 4　建立窗口

2. 更改窗口大小

如果要管理窗口的最大尺寸和最小尺寸的话采用下面的方式。

```
import tkinter
window = tkinter. Tk( )
window. minsize(366, 366)#最小尺寸
window. maxsize(566, 566)#最大尺寸
window. mainloop( )
```

3. Frame 控件

我们前面学习了窗口，如果要在一个窗口上放置很多个控件，那么我们仅仅依靠定位是没办法很好地进行管理。所以，我们要使用一个可以在窗口内存放管理控件的容器——Frame。

Frame 的使用方式和窗口的类似，我们直接通过例子来学习一下。

```
import tkinter
def main( ):
    win = tkinter. Tk( )
    win. minsize(366, 366)#最小尺寸
    win. maxsize(888, 888)#最大尺寸
    frame_one = tkinter. Frame(win)#使用 Frame 控件
    area_one = tkinter. Label(frame_one,text = 'dotcpp',font = ("华文行楷", 20), fg = "red")
    area_one. pack(side = 'top')#放在上边
    area_two = tkinter. Label(frame_one,text = 'Python',font = ("黑体", 20), fg = "blue")
    area_two. pack(side = 'bottom')#放在下面
    frame_one. pack(side = 'left')#放在左边
    #分界线上面为左边的容器,下面为右边的容器
    frame_two = tkinter. Frame(win)
    area_one = tkinter. Label(frame_two, text = 'dotcpp', font = ("华文行楷", 20), fg = "red")
    area_one. pack(side = 'top')#放在上面
    area_two = tkinter. Label(frame_two,text = 'Python',font = ("黑体", 20), fg = "blue")
    area_two. pack(side = 'bottom')#放在下面
    frame_two. pack(side = 'right')#放在右边
    win. mainloop( )
if __name__ == '__main__':
    main( )
```

运行结果如图 12. 5 所示。

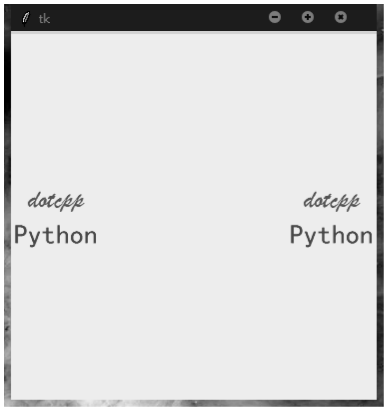

图 12. 5　Frame 窗口

12.2.3 按钮管理 Button

1. Button

我们在进行图形界面开发的时候，按钮是必不可少的一项，按钮的作用为点击一次触发一次事件，我们可以通过按钮绑定函数进行事件的触发操作。

我们在使用 Button 控件的时候常常会和 tkinter 模块中的 messagebox 模块一起使用，通过后者弹出消息框。

例如：

```
import tkinter
import tkinter. messagebox
win = tkinter. Tk( )
win. minsize(166, 40)    #最小尺寸
def button_event( ):
    tkinter. messagebox. showinfo("Button 事件", "欢迎进入 Python 教学")
button_one = tkinter. Button(win, text = "www. dotcpp. com", command = button_event)
button_one. pack( )
win. mainloop( )
```

运行结果如图 12.6 所示。

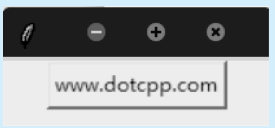

图 12.6 按钮示例

点击这个按钮会出现下面的事件信息提示窗口，如图 12.7 所示。

图 12.7 按钮事件信息

12.2.4 文本信息管理 Label、Text 和 Entry

1. Label

如果我们想要在窗口内添加文本或者图片，可以使用 Label 控件。Label 控件也叫标签

控件，该控件添加的**语法格式**为：

组件名=tkinter. Label（放置的窗口，文本或图片，附加内容）

组件名 . pack（side=‘位置’）

首先我们根据对应的信息把相应的内容填入，然后我们再通过 pack 定位一下位置信息即可。我们还可以使用 top、bottom 的位置信息。

例如：

```
import tkinter
def main( ):
    win = tkinter. Tk( )
    win. minsize(366, 366)#最小尺寸
    win. maxsize(888, 888)#最大尺寸
    area_one = tkinter. Label(win, text = 'This is area_one',font = ("华文行楷", 20), fg = "red")
    area_one. pack(side = 'left')
    area_two = tkinter. Label(win,text = 'This is area_two',font = ("黑体", 20), fg = "blue")
    area_two. pack(side = 'right')
    area_three = tkinter. Label(win, text = 'This is area_three',font = ("宋体", 20), fg = "black")
    area_three. pack( )
    win. mainloop( )
if __name__ == '__main__':
    main( )
```

2. Text

Text 控件为文本信息控制控件。该控件添加的**语法格式**为：

组件名=Text（放置的窗口，高，宽）

组件名 . insert（文本内容）

在 Tkinter 中，Text 控件很强大，也很灵活。它既可以用来实现显示单行文本，也可以用来显示多行文本，并且其还可以被用作简单的文本编辑器。另外，Text 控件还可以显示网页链接、图片、HTML 页面、CSS 样式表，甚至还可以作为一种简易网页浏览器来使用。

例如，我们要使用构造方法创建一个 Text 控件，设置其高度为 5，设置其宽度为 24，然后使用 insert( ) 方法插入两行文本。

```
from Tkinter import *

root = Tk( )
T = Text(root, height = 5, width = 24)
```

```
T. pack( )
T. insert(END, "Just a text Widget\nin five lines\n")
mainloop( )
```

【注意】本示例中不论高度还是宽度设置的数值，其数据单位都不是像素值，而是字符数，即 5 行字符的高度、24 个字符的宽度。或者，我们创建一个窗口组件，插入 Text。

例如：

```
from Tkinter import *
root = Tk( )
text = Text(root,width = 30,height = 5)
text. pack( )
text. insert(INSERT, 'I Love\n')#INSERT 表示在光标位置插入
text. insert(END, 'FishC. com!\n')#END 表示在末尾处插入


def show( ):
print('我被点了～')
#创建一个窗口组件,插入 text
b1 = Button(text, text = '点我点我', command = show)
text. window_create(INSERT, window = b1)
mainloop( )
```

执行结果如图 12. 8 所示。

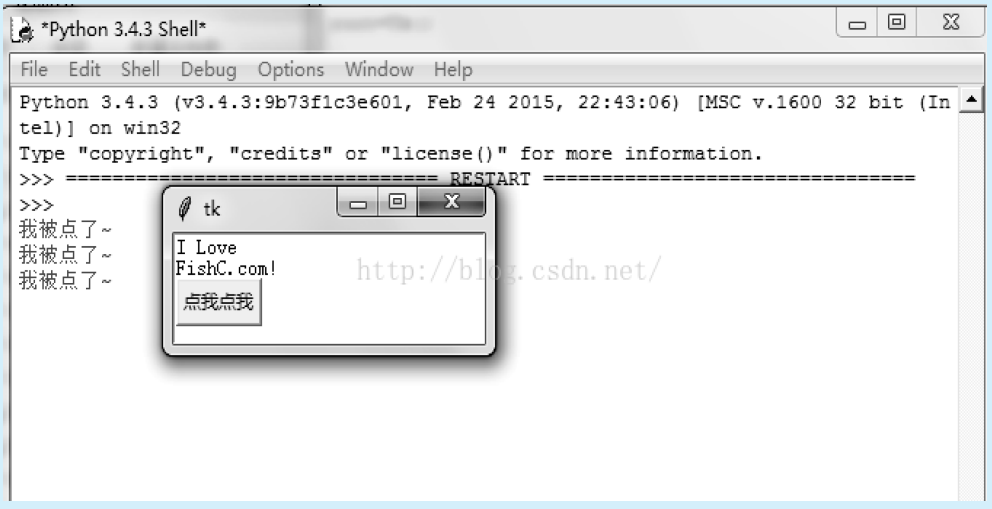

图 12. 8　窗口事件与 Text 组合应用

3. Entry

Entry 控件的作用是在键盘输入的文本信息。

（1）语法格式

它的**语法格式**如下。

my_ entry=Entry（容器，可选项）

容器即内容放入的位置，可选项和上一节我们提到的 Button 中的可选项类似，可选择的项值见表 12.2 所列。

表 12.2　Entry 的项值

样式	功能
bg 和 bd	背景颜色和边框大小
cursor	光标的形状
font	文本字体
exportselection	文本框内容是否复制功能
fg	文字颜色
highlightcolor	边框高亮的颜色
justify	对齐方式
relief	边框样式
selectbackground	选择的文本背景颜色
selectborderwidth	选择的文本背景边框宽度
selectforeground	选择的文字颜色
state	只读或可写控制
textvariable	文本框的值
width	文本框的宽度
xscrollcommand	水平方向滚动条

（2）Entry 控件的常用方法

delete（first，last = None）：删除 first－last 中的所有内容，如果使用 delete（0，END）则删除输入框的所有内容。

get( )：获取输入框内的所有内容。

icursor（index）：移动光标到 index 参数的位置。

index（index）：返回 index 参数对应的序号。

insert（index，text）：把 text 参数中的内容插入索引为 index 的位置。

Select_clear( )：清空文本框。

xview（index）：设置文本框链接的水平滚动条。

（3）Entry 控件的使用方法

我们通过实例来使用一下 Entry 控件，代码如下。

```
import tkinter
win = tkinter. Tk( )
Frame_one = tkinter. Frame(win)#先创建一个容器放上面存放登录
Frame_one. pack(side = 'top')
Frame_two = tkinter. Frame(win)#再创建一个容器放中间存放密码
Frame_two. pack( )
Frame_three = tkinter. Frame(win)#再创建一个容器在下面存放按钮
Frame_three. pack(side = 'bottom')
Label_one = tkinter. Label(Frame_one, text = '姓名:')
Lable_two = tkinter. Label(Frame_two, text = '密码:')
Entry_one = tkinter. Entry(Frame_one, bd = 5)
Entry_two = tkinter. Entry(Frame_two, bd = 5)
Button_one = tkinter. Button(Frame_three,text = '登录',activeforeground = 'red', active-
background = 'yellow', width = '7')
Button_one. pack(side = 'left')
Button_two = tkinter. Button(Frame_three, text = '注册', activeforeground = 'blue', active-
background = 'pink',width = '7')
Button_two. pack(side = 'right')
Label_one. pack(side = 'left')
Entry_one. pack(side = 'right')
Lable_two. pack(side = 'left')
Entry_two. pack(side = 'right')
win. mainloop( )
```

执行结果如图 12. 9 所示。

图 12. 9　执行结果

我们首先在窗口中放置了三个容器，从上到下依次存放姓名、密码和登录、取消按钮；然后在姓名后面放一个 Entry 控件，供我们输入姓名，在密码对应的后面放一个 Enrty 控件来输入密码；再在下面放两个按钮分别提供登录和注册；最后，我们把他们的位置放在左右一一对应。

我们再通过函数的绑定来测试登录信息，代码如下。

```
import tkinter as tk
import tkinter. messagebox

win = tk. Tk( )

frame_name = tk. Frame(win)#创建容器来存放登录的 Label(文本框)与 Entry(输入框)

frame_name. pack(side = "top")#使该容器在页面的顶部
label_name = tk. Label(frame_name, text = "Your Name : ")
label_name. pack(side = "left")
entry_name = tk. Entry(frame_name, bd = 5)
entry_name. pack(side = "right")#在容器内创建 Label 与 Entry,并使 label 在左,entry 在右

#下面的同理:
frame_password = tk. Frame(win)
frame_password. pack()
label_password = tk. Label(frame_password, text = "Your Password : ")
label_password. pack(side = "left")
entry_password = tk. Entry(frame_password, bd = 5)
entry_password. pack(side = "right")

def login( ):
    if entry_name. get( ) == "qy":
        if entry_password. get( ) == "dotcpp":
            print(tkinter. messagebox. showinfo("login", "Success!"))
        else:
            print(tkinter. messagebox. showerror("login", "Failed!"))
            entry_name. delete(0, "end")
            entry_password. delete(0,"end")
else:
        print(tkinter. messagebox. showerror("login", "Failed!"))
        entry_name. delete(0, "end")
        entry_password. delete(0, "end")
```

```
    def signin( ):
        print(tkinter. messagebox. showerror("signin", "Without Code!"))#必须先定义函数,否则点击按钮调用函数时会报函数不存在的错误

    frame_button = tk. Frame(win)#创建容器以存放按钮
    frame_button. pack(side = "bottom")#使该容器位于页面最下方
    button_login = tk. Button(
        frame_button,
        text = "login",
        activeforeground = "red",
        activebackground = "yellow",
        width = "7",
        command = login
    # command 的意思是执行已定义的函数,不可执行下文中出现的函数(未定义的函数)
    )
    button_login. pack(side = "left")

    #下面的同理:
    button_signin = tk. Button(
        frame_button,
        text = "signin",
        activeforeground = "blue",
        activebackground = "pink",
        width = "7",
        command = signin,
    )
    button_signin. pack(side = "right")

    win. mainloop( )
```

12. 2. 5　选择控件 Radiobutton 和 Checkbutton

1. Radiobutton 按钮

Radiobutton 控件，也叫单选按钮控件。有的时候我们需要在多个选项中选择一项，那么我们就可以使用到 Tkinter 模块中的 Radiobutton 方法来创建相关按钮。

例如：

```
import tkinter
win = tkinter. Tk( )
win. title("Python 教程")   #给窗口取一个标题
win. minsize(366, 50)   #定义一个最小尺寸
def get_data( ):
    print('选择的为第%d 项'%x. get( ))
x = tkinter. IntVar( )#在这里我们把一组单选框绑定为同一个变量
radio_one = tkinter. Radiobutton(win, text = "选项 1", value = 1,
variable = x,command = get_data)
radio_one. pack( )
radio_two = tkinter. Radiobutton(win, text = "选项 2", value = 2,
variable = x, command = get_data)
radio_two. pack( )
win. mainloop( )
```

执行结果如图 12. 10 所示。

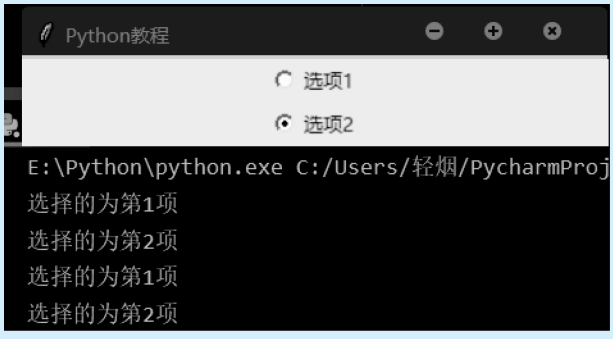

图 12-10　单选按钮

【注意】我们在定义的时候给两个 Radiobutton 使用了同一变量，通过这个我们才可以使两个按钮公用为一个信息，从而实现单选效果。

2. Checkbutton 控件

Checkbutton 按钮也叫多选按钮。我们在做选择题的时候会遇到多选题。前面我们学习的 Radiobutton按钮即给我们提供了单选的功能，那么如果我们要使用多选操作就要通过 Checkbutton 来实现。

例如：

```
import tkinter
win = tkinter. Tk( )
win. title("Python 教程")
```

```
win. minsize(200, 200)
def get_data( ):
    my_str = ''
    if x. get( ) == True:
        my_str += "已选择选项 1 \n"
    if y. get( ) == True:
        my_str += "已选择选项 2 \n"
    if z. get( ) == True:
        my_str += "已选择选项 3 \n"
    text. delete(0. 0, tkinter. END)      #清除 text 文本框中的内容
    text. insert(tkinter. INSERT, my_str)      #把上面的信息插入文本框
x = tkinter. BooleanVar( )
ckbutton_one = tkinter. Checkbutton(win, text = "选项 1", variable = x,
                                    command = get_data)
ckbutton_one. pack( )
y = tkinter. BooleanVar( )
ckbutton_two = tkinter. Checkbutton(win, text = "选项 2", variable = y,
                                    command = get_data)
ckbutton_two. pack( )
z = tkinter. BooleanVar( )
ckbutton_three = tkinter. Checkbutton(win, text = "选项 3", variable = z,
                                    command = get_data)
ckbutton_three. pack( )
text = tkinter. Text(win, width = 50, height = 5)
text. pack( )
win. mainloop( )
```

执行结果如图 12. 11 所示。

图 12. 11　多选按钮

12.2.6　画布管理 Canvas

Canvas 控件，也叫画布控件。它在 Tkinter 模块中对应的是一片矩形区域，给使用者提供绘图的空间，可以把图形、文本、椭圆或按钮等组件放在画布上，也可以进行图案的绘画。在使用 Canvas 组件之前我们要先来学习一下它对应的屏幕坐标系。

计算机中的图像我们一般用像素来作单位，每个像素会有 2 个值，我们在 Canvas 组件中，对应的窗口左上方的坐标为（0，0），也就是说在平面直角坐标系中对应的 x 轴和 y 轴都为 0，窗口右下角的坐标为距离 x 轴和 y 轴的两个最大值，如图 12.12 所示。

图 12.12　画布界面

我们在 Canvas 画布中放置组件的时候，一定要保证组件坐标的正确。

Canvas 的语法结构为：

my_canvas = Canvas（父类，可选项）

父类即为我们放置的上一层窗口名，可选项有 bd、bg、continue、cursor、height、width、highlightcolor、relief、scrollregion 等。这些选项在前面都有介绍，在这里就不做过多的讲解。

Canves 组件绘制图形的方法有多种，分别有 create_line、create_rectangle、create_oval、create_arc、create_polygon，我们下面进行一一介绍。

1. create_ line 创建线条

语法格式如下：

canvas. create_ line（x1，y1，x2，y2，...，xn，yn，options）

我们可以通过这种方法在画布上的两个或者 n 个点之间画出一条直线，前两个坐标（$x1$，$y1$）即为起始点，（xn，yn）为终点。

例如：

```
import tkinter
class TK:
```

```
        def __init__(self):
            self. window = tkinter. Tk( )
            self. canvas = tkinter. Canvas(self. window, width = 300, height = 300, bg = 'pink')
            self. canvas. create_line(30, 30, 100, 200, 200, 200, 300, 200, 300, 300)
            self. canvas. pack( )
            tkinter. mainloop( )
    m = TK( )
```

执行结果如图 12. 13 所示。

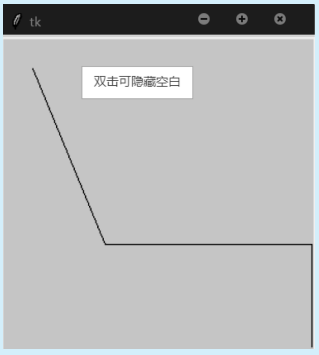

图 12-13　在画布上划线

【解析】我们选取的坐标为（30，30）->（100，200）->（200，200）->（300，300），根据这几个坐标我们画出上图。

2. create_rectangle 创建矩形

创建矩形的语法格式为：

canvas. create_ rectangle（x1，y1，x2，y2，可选项）

矩形只需要 2 个坐标即可确定，因此我们只需要左上角和右下角的坐标（$x1$，$y1$）和（$x2$，$y2$）。

例如：

```
import tkinter
class TK:
    def __init__(self):
        self. window = tkinter. Tk( )
            self. canvas = tkinter. Canvas(self. window, width = 300, height = 300, bg = 'pink')
            self. canvas. create_rectangle(100, 100, 200, 200)
            self. canvas. pack( )
            tkinter. mainloop( )
    m = TK( )
```

执行结果如图 12.14 所示。

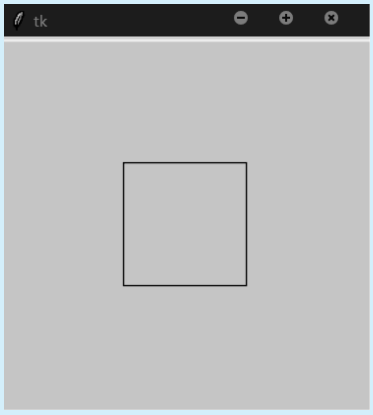

图 12.14　画矩形

3. 用 create_ oval 创建椭圆

语法格式为：

canvas.create.oval（x1，y1，x2，y2，可选项）

其中，（$x1$，$y1$）是椭圆外接矩形的左上角坐标，（$x2$，$y2$）是椭圆外接矩形的右下角坐标，如图 12.15 所示。

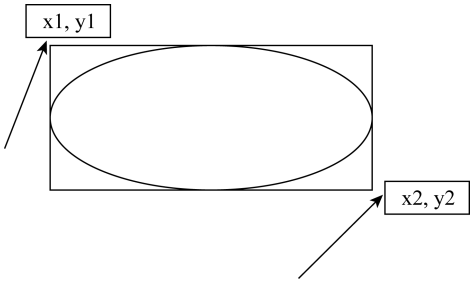

图 12.15　画椭圆时外接矩形的坐标

4. 用 create_ arc 创建弧形

语法格式如下：

canvas.create_ arc（x1，y1，x2，y2，start=angle，extent=width，可选项）

其中，坐标对应的和上图中椭圆对应的一致，start 中的值对应的是弧形的起始角度，extent 对应的是弧形逆时针角度范围。

例如：

```
import tkinter
class TK:
    def __init__(self):
```

```
        self. window = tkinter. Tk( )
        self. canvas = tkinter. Canvas(self. window, width = 300, height = 300, bg = 'pink')
        self. canvas. create_arc(60, 60, 220, 220, start = 0, extent = 120, fill = 'blue')
        self. canvas. pack( )
        tkinter. mainloop( )
m = TK( )
```

执行结果如图 12. 16 所示。

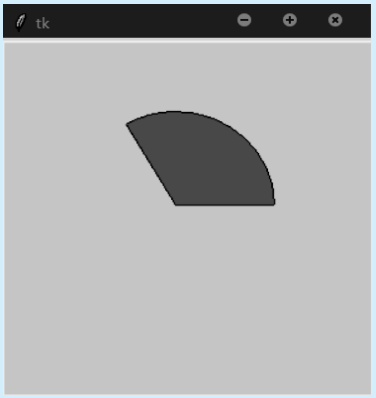

图 12. 16　画弧形

5. create_ polygon-创建多边形

语法格式为：

canvas. create_ polygon（x1，y1，x2，y2，. . . . . xn，yn，可选项）

每个坐标对应一个位置，如第一个坐标为第一个顶点，依次连接，最后一个坐标自动关闭多边形。

例如：

```
import tkinter
class TK:
    def __init__(self):
        self. window = tkinter. Tk( )
        self. canvas = tkinter. Canvas(self. window, width = 300, height = 300, bg = 'pink')

        self. canvas. create_ polygon(150, 0, 200, 100, 300, 100, 200, 185, 280, 280, 150,
230, 20, 280, 100, 185,
        0, 100, 100, 100, fill = 'red')
        self. canvas. pack( )
        tkinter. mainloop( )
m = TK( )
```

执行结果如图 12.17 所示。

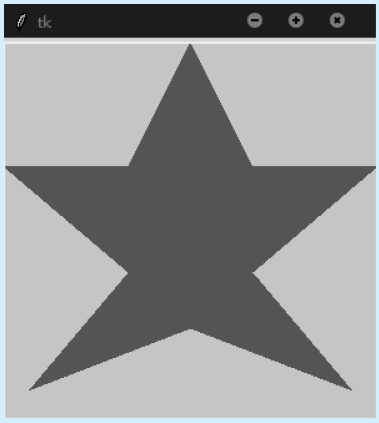

图 12.17　画多边形

大家可以通过坐标的改变去画出自己想要画出的图形，可选的参数还有 dash、outline、sommth 和 width。

12.3　wxPython 模块

12.3.1　wxPython 模块简介和初始化

我们在前面学习过 Tkinter 模块，通过该模块我们可以进行简单的图像界面开发，但是我们发现很多人在开发的时候还会选择 wxPython 模块。它是一个比较成熟且功能比较丰富的模块。它和 Tkinter 类似，也要引入窗口、按钮、文本框等内容，但是它又有独特的用法。在这里我们先通过一张图来了解一下程序和窗口之间的关系。

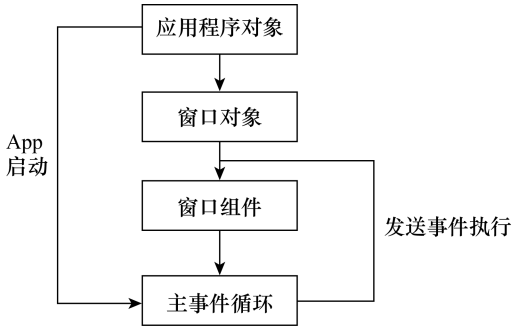

图 12.18　wxPython 模块工作流程图

我们通过一个最简单的 wxPython 程序来对应一下相关结构。

例如：

```
import wx
app = wx. App( )
win = wx. Frame(None, title = '小程序', size = (300, 300))
win. Show( )
app. MainLoop( )
```

执行结果如图 12. 19 所示。

图 12. 19　wxPython 基本执行界面

1. 初始化

上面提到的例子就是我们使用 wxPython 的基本架构。如果我们在程序开发的时候，使用类与对象的关系会更方便我们后期处理。

我们创建子类的时候首先定义一个子类，然后创建一个 OnInit( ) 方法来初始化这个子类，最后在主程序中调用这个类，进入主事件循环。

例如：

```
import wx
class App(wx. App):
    def OnInit(self):
        window = wx. Frame(parent = None, title = '初始化')
        window. Show( )
        return True
if __name__ == '__main__':
    app = App( )
    app. MainLoop( )
```

【解析】在这个例子中，我们定义的子类 App 继承了父类 wx. App，然后在子类中创建了一个初始化方法，最后在主程序中创建一个类的实例，进入主事件循环。

2. Frame 框架

我们在创建实例的时候总是要引入 Frame 框架。它属于一种容器，可以用来存放我们主程序的一些标题和功能区。我们来看一下它的**语法结构**：

wx. Frame（parent，id=-1，title=''，style=''，name=''，pos=wx. DefaultPositon，size=wx. DefaultSize）

parent 指的是框架所处的位置，也就是父窗口的名字。如果当前为顶级窗口，就用 None 来代替，id 为新窗口的 ID 号，通常设置为-1，title 即窗口的名字，size 即窗口的大小，style 即窗口的类型，name 即框架的内部名字，pos 为一个对象，指定了这个新窗口在界面中的位置，上面所选的 Default 为默认参数。

具体使用如下。

```
window=wx. Frame(parent=None, id=-1, title= 'Frame 框架 ', size=(400, 400), pos=(2, 2))
```

12. 3. 2　文本控件

wxPython 中的控件全部继承于 wx. Control，包含了静态文本、文本输入控件、按钮、列表、滑块、滚动条、复选框等。本节我们来学习一下静态文本和文本输入控件。

1. 静态文本

静态文本就是在屏幕上显示的静态文字，我们使用 wx. StaticText 类来完成。它的**语法格式**为：

wx. StaticText（parent，id，label=''，pos=wx. DefaultPositon，size=wx. DefaultSize，style=''，name=''）

其中，label 为显示在控件中的文本信息，其余的我们在前面都学习过就不再做解释。

样式 style 的取值如下。

wx. ALIGN_CENTER：静态文本位于静态文本控件的中心。

wx. ALIGN_LEFT：文本在窗口部件中左对齐，这是默认的样式。

wx. ALIGN_RIGHT：文本在窗口部件中右对齐。

wx. ST_NO_AUTORESIZE：如果使用了这个样式，那么在使用了 SetLabel( )改变文本之后，静态文本控件将不会自我调整尺寸。

例如：

```
import wx
class Frame(wx. Frame):
    def __init__(self):
        wx. Frame. __init__(self, None, wx. ID_ANY, "静态文本", size=(200, 170))
        panel=wx. Panel(self, -1)
```

```
            wx. StaticText(panel, wx. ID_ANY, "春夜喜雨 ——杜甫。", (0, 10), (145, -1),
wx. ALIGN_RIGHT)
            text = wx. StaticText(panel, wx. ID_ANY, "好雨知时节,当春乃发生。", (0, 30), (160,
-1), wx. ALIGN_RIGHT)
            text = wx. StaticText(panel, wx. ID_ANY, "随风潜入夜,润物细无声。", (0,50), (160,
-1), wx. ALIGN_RIGHT)
            text = wx. StaticText(panel, wx. ID_ANY, "野径云俱黑,江船火独明。", (0,70), (160,
-1), wx. ALIGN_RIGHT)
            text = wx. StaticText(panel, wx. ID_ANY, "晓看红湿处,花重锦官城。", (0,90), (160,
-1), wx. ALIGN_RIGHT)
        if __name__ =='__main__':
            app = wx. App( )
            frame = Frame( )
            frame. Show(True)
            app. MainLoop( )
```

【注意】我们在使用的时候首先要用 wx. Panel 方式来创建一个容器，类似于 Tkinter 中的 Frame 容器，然后我们在这个容器里放置一些组件。

我们还可以对文字信息进行修饰，使用 wx. Font 方法，它的**语法结构**如下。

wx. Font（pointSize，fimily，style，weight，underline = True，faceName = ' '，encoding = wx. FONTENCODING_ DEFAULT)

其中，pointSize 为字体的尺寸，family 为字体的名字，style 判定倾斜，weight 为宽度，underline 为下画线，True 为有，False 为无，faceName 为此方法的字体名，encoding 为编码方式。

2. 文本输入控件

我们在与程序交互的时候，静态文字是无法获取我们输入的信息的，因此我们引入 wx. TextCtrl 类来获取用户输入的文本内容，它的**语法结构**为：

wx. TextCtrl（parent，id，value = ' '，pos = wx. DefaultPositon，size = wx. DefaultSize，style，validator = wx. DefaultValidator，name = ' '）

除了 validator 为过滤数据外，其余的我们在前面都提到过，style 在这里功能有增加，我们再介绍一下。

wx. TE_CENTER：控件中的文本居中。

wx. TE_LEFT：控件中的文本左对齐。

wx. TE_NOHIDESEL：文本始终高亮显示，只适用于 Windows。

wx. TE_PASSWORD：不显示所键入的文本，代替以星号显示。

wx. TE_PROCESS_ENTER：如果使用了这个样式，那么当用户在控件内按下回车键时，一个文本输入事件被触发。否则，按键事件由文本控件或对话框管理。

wx. TE_PROCESS_TAB：如果指定了这个样式，那么通常的字符事件在 Tab 键按下时创建。否则，Tab 由对话框来管理，通常是控件键的切换。

wx. TE_READONLY：文本控件为只读，用户不能修改其中的文本。

wx. TE_RIGHT：控件中的文本右对齐。

例如：

```
import wx
class Frame(wx. Frame):
    def __init__(self):
        wx. Frame. __init__(self, None, wx. ID_ANY, "登录", size = (300, 300))
        panel = wx. Panel(self, -1)
        wx. StaticText(panel, wx. ID_ANY, "登录界面", (0, 10), (150, -1), wx. ALIGN_RIGHT)
        text = wx. StaticText(panel, wx. ID_ANY, "账户", (0, 50), (80, -1), wx. ALIGN_RIGHT)
        text = wx. StaticText(panel, wx. ID_ANY, "密码", (0, 90), (80, -1), wx. ALIGN_RIGHT)
        text = wx. TextCtrl(panel, wx. ID_ANY, "", (100, 50), (100, 20), wx. ALIGN_LEFT)
        text = wx. TextCtrl(panel, wx. ID_ANY, "", (100, 90), (100, 20), wx. ALIGN_LEFT)
if __name__ == '__main__':
    app = wx. App( )
    frame = Frame( )
    frame. Show(True)
    app. MainLoop( )
```

执行结果如图 12. 20 所示。

图 12. 20　登录示例

12.3.3 按钮和选择框

1. 按钮

按钮是 GUI 界面中必不可少的一环，在前面部分已经介绍过按钮，现在我们直接进行语法的学习。wxPython 中按钮的**语法结构**为：

wx.Button（parent，id，label，pos，size，style，validator，name）

它的相关参数与前面我们使用过的参数大致相同。下面我们使用 Button 按钮对上一节的登录界面进行修饰。

```
import wx
class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, wx.ID_ANY,"登陆", size=(300, 250))
        panel=wx.Panel(self, -1)
        wx.StaticText(panel, wx.ID_ANY, "登录界面", (0, 10), (150, -1), wx.ALIGN_RIGHT)
        self.text=wx.StaticText(panel, wx.ID_ANY, "账户", (0, 50), (80, -1), wx.ALIGN_RIGHT)
        self.text=wx.StaticText(panel, wx.ID_ANY, "密码", (0, 90), (80, -1), wx.ALIGN_RIGHT)
        self.text=wx.TextCtrl(panel, wx.ID_ANY, "", (100, 50), (100, 20), wx.ALIGN_LEFT)
        self.text=wx.TextCtrl(panel, wx.ID_ANY, "", (100, 90), (100, 20), wx.ALIGN_LEFT)
        self.button=wx.Button(panel, wx.ID_ANY, "登陆", (70, 120), (50, 20), wx.ALIGN_LEFT)
        self.button=wx.Button(panel, wx.ID_ANY, "注册", (140, 120), (50, 20), wx.ALIGN_LEFT)
if __name__=='__main__':
    app=wx.App( )
    frame=Frame( )
    frame.Show(True)
    app.MainLoop( )
```

执行结果如图 12.21 所示。

图 12.21 登录示例

2. 复选框和单选按钮

复选框对应前面我们学习过的 Tkinter 中的 check 控件。复选框提供多个按钮，可提供同时开关的功能。单选按钮对应 Tkinter 中的 radio 控件，单选按钮提供多个按钮。但是只能选择其中一个按钮。在 wxPython 中我们使用 wx. CheckBox 和 wx. RadioButton 来创建复选框和单选按钮。

例如：

```
import wx
class Frame(wx. Frame):
    def __init__(self):
        wx. Frame. __init__(self, None, wx. ID_ANY,"复选框", size=(300, 250))
        panel=wx. Panel(self, -1)
        self. box=wx. CheckBox(panel, -1, "Checkbox1", pos=(50, 50), size=(80, 20))    #创建控件
        self. box=wx. CheckBox(panel, -1, "Checkbox2", pos=(50, 70), size=(80, 20))    #创建控件
        self. box=wx. CheckBox(panel, -1, "Checkbox3", pos=(50, 90), size=(80, 20))    #创建控件
if __name__=='__main__':
    app=wx. App( )
    frame=Frame( )
    frame. Show(True)
    app. MainLoop( )
```

执行结果如图 12. 22 所示。

图 12. 22　多选框

单选框代码如下。

```
import wx
class MyFrame(wx. Frame):
def __init__(self):
        wx. Frame. __init__(self, None, -1, "选择一种喜欢的运动方式", size=(300, 100))
```

```
            panel = wx. Panel(self)
            wx. StaticText(panel, wx. ID_ANY, "选择一种喜欢的运动方式", (0, 10), (200, - 1),
wx. ALIGN_RIGHT)
            self. check1 = wx. RadioButton(panel, - 1, "打篮球", pos = (60, 40), size = (50, 20),
style = wx. RB_GROUP)
            self. check2 = wx. RadioButton(panel, - 1, "打乒乓球", pos = (130, 40), size = (50, 20))
            self. check3 = wx. RadioButton(panel, - 1, "跑步", pos = (200, 40), size = (50, 20))
        if __name__ == "__main__":
        app = wx. App( )
        frame = MyFrame( )
        frame. Show( )
        app. MainLoop( )
```

执行结果如图 12.23 所示。

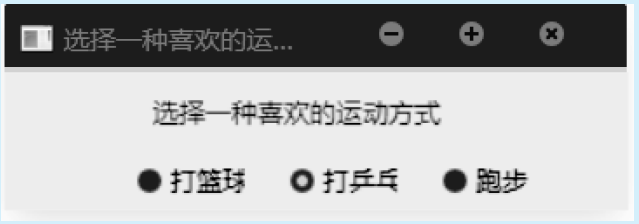

图 12.23　单选框

12.3.4　wxPython 布局

1. 布局简介

布局是窗口界面中控件的排列方式。一个合理的布局能够把面积的使用最大化。我们使用 wxPython 的时候可以采用两种布局形式：一种是前面我们用到的绝对布局，也就是说我们所创建的控件的位置是固定不变的，无论窗口的大小如何变化，绝对布局的子窗口或者控件都是不变的。本节我们学习一种 Sizer 管理布局，它能帮助我们自动布局一组窗口控件。

在 WxPython 中，有八种 Sizer 可供我们使用，分别是 Sizer、WrapSizer、BoxSizer、GridSizer、FlexGridSizer、GridBagSizer 和 StaticBoxSzer、StdDialogButtonSizer。其中，我们主要对 BoxSizer 进行学习。

它的分类结构如图 12.24 所示。

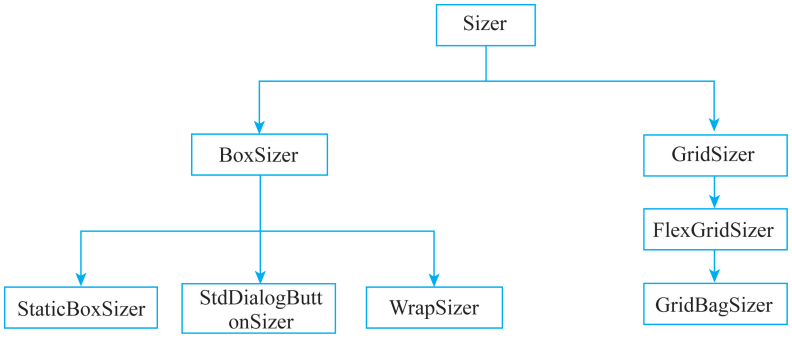

图 12.24　Sizer 分类结构图

我们在日常使用的时候一般使用 BoxSizer、StaticBoxSizer、GridSizer 和 FlexGridSizer 这四种。

2. BoxSizer 布局

BoxSizer 布局是使用的几种布局中最灵活简便的一个。它的排布方式或垂直或水平，一般是在创建的时候指定布局的方向。**语法格式**为：

my_ box = wx. BoxSizer（wx. HORIZINTAL）

my_ box = wx. BoxSizer（wx. VERTICAL）

上面的语句为水平布局，下面的语句为垂直方向的布局。当括号中为空的时候即默认为水平布局。

我们往布局中添加控件的时候，使用 Add( )方法，这个 Add( )方法由 wx. Sizer 继承而来，它的**语法格式**为：

Add（parent，proportion = 0，flag = 0，border = 0，userData = None）

这种方式添加到父窗口 parent 当中。

Add（AnotherSizer，proportion = 0，flag = 0，border = 0，userData = None）

这种方式添加到另外一个布局当中。

Add（AnotherSizer，proportion = 0，flag = 0，border = 0，userData = None）

这种方式创建了一个新的空间。

其中，flag 为对齐、边框和尺寸参数，border 为边框宽度，userData 为数据的传输。

flag 对齐标志有多种，诸如：

ALIGN_ TOP：顶对齐；

wx. ALIGN_ LEFT：左对齐；

wx. ALIGN_ BOTTOM：底对齐；

wx. ALIGN_ RIGHT：右对齐；

wx. ALIGN_ CENTER：居中对齐；

wx. ALIGN_ CENTER_ VERTIAL：垂直居中对齐；

wx. ALIGN_ CENTER_ HORIZONTAL：水平居中对齐。

边框 flag 的标志有：

wx. TOP：设置有顶部边框，边框的宽度需要通过 Add( )方法的 border 参数设置；

wx. BOTTOM：设置有底部边框；

wx. LEFT：设置有左边框；

wx. RIGHT：设置有右边框；

wx. ALL：设置四面全有边框。

调整尺寸的 flag 标志有：

wx. EXPAND：调整子窗口（或控件）完全填满有效空间；

wx. SHAPED：调整子窗口（或控件）填充有效空间，但保存高宽比；

wx. FIXED_MINSIZE：调整子窗口（或控件）为最小尺寸；

wx. RESERVE_SPACE_EVEN_IF_HIDDEN：设置此标志后，子窗口（或控件）如果被隐藏，所占空间保留。

下面，使用 BoxSizer 布局来创建一个登录界面。

```
import wx
class MyFrame(wx. Frame):
    def __init__(self):
        wx. Frame. __init__(self, None, - 1, "登录界面", size = (300, 300))
        panel = wx. Panel(self)#创建一个画布,然后创建功能区并放到画布上
        self. title = wx. StaticText(panel, label = '登录界面')
        self. username = wx. StaticText(panel, label = '用户: ')
        self. user_name = wx. TextCtrl(panel, style = wx. TE_LEFT)
        self. userpassword = wx. StaticText(panel, label = '密码:')
        self. user_password = wx. TextCtrl(panel, style = wx. TE_PASSWORD)
        self. button_login = wx. Button(panel, label = '登陆')
        self. button_register = wx. Button(panel, label = '注册')
        container_one = wx. BoxSizer(wx. HORIZONTAL)   #创建一个 box 容器并控制水平排列
        container_one. Add(self. username, proportion = 0, flag = wx. ALL, border = 7)
        container_one. Add(self. user_name, proportion = 1, flag = wx. ALL, border = 7)
        container_two = wx. BoxSizer(wx. HORIZONTAL)   #创建一个 box 容器并控制水平排列
        container_two. Add(self. userpassword, proportion = 0, flag = wx. ALL, border = 7)
        container_two. Add(self. user_password, proportion = 1, flag = wx. ALL, border = 7)
        container_three = wx. BoxSizer(wx. HORIZONTAL)   #创建一个 box 容器并控制水平排列
         container _ three. Add (self. button _ login, proportion = 0, flag = wx. ALIGN _ CENTER,
border = 4)
```

```
            container_three. Add(self. button_register, proportion = 0, flag = wx. ALIGN_CENTER,
border = 4)
            sizers = wx. BoxSizer(wx. VERTICAL)
             sizers. Add (self. title, proportion = 0, flag = wx. BOTTOM | wx. TOP | wx. ALIGN _
CENTER, border = 10)
          sizers. Add (container_ one,proportion = 0, flag = wx. EXPAND | wx. LEFT | wx. RIGHT,
border = 40)
            sizers. Add(container_two, proportion = 0, flag = wx. EXPAND | wx. LEFT | wx. RIGHT,
border = 40)
             sizers. Add (container _ three, proportion = 0, flag = wx. ALIGN _ CENTER | wx. TOP,
border = 10)
            panel. SetSizer(sizers)
    if __name__ == "__main__":
    app = wx. App( )
    frame = MyFrame( )
    frame. Show( )
    app. MainLoop( )
```

执行结果如图 12. 25 所示。

图 12. 25　登录界面

在这个例子中，我们主要使用了竖直和水平两种布局。我们先把用户和输入信息放入一个水平 BoxSizer，再把密码和输入信息放入一个 BoxSizer，然后把两个按钮放在一个 BoxSizer，最后再把四种信息全部放到一个竖直的 BoxSizer 中。

12. 3. 5　wxPython 事件处理

1. 事件简介

事件处理是指对图形界面中控件操作的响应。为更好地学习事件处理，我们需要理解以下几个基本概念。

事件：当控件达到某种状态时用户需要执行的动作。例如我们去点击一个按钮，这就是一个事件。

事件类型：指事件的种类。例如我们点击按钮和松开按钮，这就属于两种类型。

事件源：指事件产生时，该事件由哪些控件控制发生。

事件处理者：它是 wxPython 内部的一个 wx. EvtHandler 子类中的一个方法。

2. 事件处理操作

事件的绑定即我们把一个函数捆绑到一个可发生改变的控件上。例如我们为确定按钮绑定添加一个事件来验证相关信息，**语法格式**如下：

Button_ login. Bind（wx. EVT_ BUTTON，OnclickEventName）

wx. EVT_ BUTTON 是事件类型选择了按钮类型，OnclickEventName 为点击按钮时执行的方法名。

例如，现在我们来进一步完善上一节创建的登录界面，代码如下。

```
import wx
class MyFrame(wx. Frame):
    def __init__(self):
        wx. Frame. __init__(self, None, -1, "登录界面", size = (300, 300))
        panel = wx. Panel(self)#创建一个画布,然后创建功能区并放到画布上
        self. title = wx. StaticText(panel, label = '登录界面')
        self. username = wx. StaticText(panel, label = '用户:')
        self. user_name = wx. TextCtrl(panel, style = wx. TE_LEFT)
        self. userpassword = wx. StaticText(panel, label = '密码:')
        self. user_password = wx. TextCtrl(panel, style = wx. TE_PASSWORD)
        self. button_login = wx. Button(panel, label = '登录')
        self. button_login. Bind(wx. EVT_BUTTON, self. OnclickEventname)
... 和上一节一致,在这里省略 ....
def OnclickEventname(self, event):
        ms = ''
        urnm = self. user_name. GetValue( )
        pwd = self. user_password. GetValue( )
        if urnm == ''or pwd == '':
            ms = '用户或密码不能为空'
        elif urnm == 'qy'and pwd == '123456':
```

```
            ms = '登陆成功'
        else:
            ms = '用户或密码错误'
        wx. MessageBox(ms)
if __name__ == "__main__":
app = wx. App( )
frame = MyFrame( )
frame. Show( )
app. MainLoop( )
```

执行结果如图 12. 26、图 12. 27、图 12. 28 所示。

图 12. 26　登录界面的用户名或密码错误事件处理

图 12. 27　登录界面的用户名或密码空事件处理

图 12.28　登录界面的登录成功事件处理

我们主要对绑定的函数部分进行掌握即可。函数中的内容为我们在用户和密码框中输入的内容，分三种判断方式返回三种结果，然后通过 MessageBox 方法把内容显示在提示框中，我们要注意在绑定的时候的代码如下。

```
self. button_login. Bind(wx. EVT_BUTTON, self. OnclickEventname)
```

事件就是通过这一行代码和下面的函数绑定在了一起。每当我们点击一次按钮，按钮就会执行函数中的内容。

12.3.6　wxPython 下拉列表

下拉列表是由一个文本框和一个列表组成的。它有两种方法可以使用：一种是 wx. ComboBox，一种是 wx. Choice。前者的文本框是可变的，而后者的是固定的。

例如：

```
import wx
class MyFrame(wx. Frame):
    def __init__(self):
        wx. Frame. __init__(self, None, -1, "下拉列表", size=(300, 300))
        self. Centre( )
        panel=wx. Panel(self)#创建一个容器
        box_one=wx. BoxSizer(wx. HORIZONTAL)
        text_one=wx. StaticText(panel, label='你喜欢的运动为:')
        sports=['篮球', '足球', '排球']
        select_one=wx. ComboBox(panel, -1, value='足球', choices=sports, style=wx. CB_SORT)
        box_one. Add(text_one, 1, flag=wx. LEFT|wx. RIGHT|wx. FIXED_MINSIZE, border=7)
        box_one. Add(select_one, 1, flag=wx. ALL|wx. FIXED_MINSIZE)
        box_two=wx. BoxSizer(wx. HORIZONTAL)
```

```
        text_two = wx. StaticText(panel, label = '你喜欢的手机品牌为:')
        phones = ['小米', '华为', '苹果']
        select_two = wx. Choice(panel, -1, choices = phones, style = wx. CB_SORT)
        box _ two. Add (text _ two,  1,  flag  =  wx. LEFT | wx. RIGHT | wx. FIXED _ MINSIZE,
border = 7)
        box_two. Add(select_two, 1, flag = wx. ALL|wx. FIXED_MINSIZE)
        bbox = wx. BoxSizer(wx. VERTICAL)
        bbox. Add(box_one, 1, flag = wx. ALL|wx. EXPAND, border = 7)
        bbox. Add(box_two, 1, flag = wx. ALL  |  wx. EXPAND, border = 7)
        panel. SetSizer(bbox)
    if __name__ == "__main__":
    app = wx. App( )
    frame = MyFrame( )
    frame. Show( )
    app. MainLoop( )
```

执行结果如图 12. 29 所示。

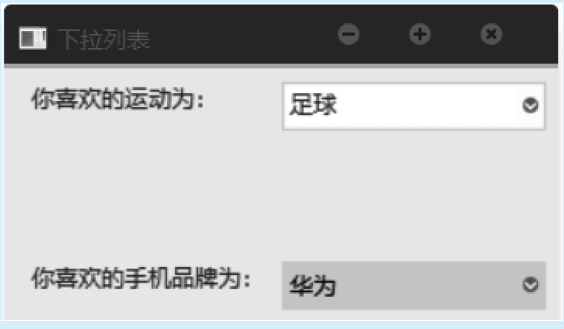

图 12. 29　文本框和列表

【注意】wx. Choice 和 wx. ComboBox 是有区别的。当我们选择使用 wx. Choice 的时候，选择框中的内容是固定的，只有从列表中选择；而使用 wx. ComboBox 的时候选择框中的内容不是固定的。上述程序的关键语句为：

```
    select_ one = wx. ComboBox(panel, -1, value = '足球', choices = sports, style = wx. CB_
SORT)
    select_two = wx. Choice(panel, -1, choices = phones, style = wx. CB_SORT)
```

12. 3. 7　wxPython 菜单

菜单（见图 12. 30）是指在软件中的功能导航。在一个软件的图形界面开发中，菜单是很常用的功能。

例如：

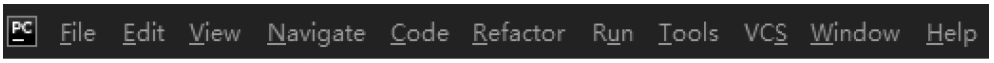

图 12.30　菜单

下面我们介绍下 wxPython 中常见的菜单栏功能。

（1）创建一个菜单栏：menuBar=wx. MenuBar( )。

（2）创建菜单：fileMenu=wx. Menu( )。

（3）创建菜单项：newItem=wx. MenuItem( )。

（4）添加菜单项到菜单中：fileMenu. AppendItem（newItem）。

（5）添加菜单到菜单栏：menuBar. Append（fileMenu，title="File" ）。

（6）把菜单栏设置为界面的菜单栏：self. SetMenuBar（menuBar）。

（7）绑定菜单事件：self. Bind（wx. EVT_ MENT，self. menuHandler）。

接下来，我们通过一个实例来学习。

```
import wx

class MyFrame(wx. Frame):
    def __init__(self):
        super( ). __init__(parent=None, title="wxPython", size=(400, 300))
        self. Center( )
        self. text=wx. TextCtrl(self, -1, style=wx. TE_MULTILINE)
        vbox=wx. BoxSizer(wx. VERTICAL)
        self. SetSizer(vbox)
        vbox. Add(self. text, 1, flag=wx. EXPAND | wx. ALL, border=1)
        menubar=wx. MenuBar( )#对应步骤 1
        self. SetMenuBar(menubar)
        file_menu=wx. Menu( )#对应步骤 2
        menubar. Append(file_menu, '菜单')#对应步骤 4
        file_menu. Append(id=wx. ID_NEW, item='新建', helpString='new file')
        file_menu. AppendSeparator( )
        edit_menu=wx. Menu( )
        file_menu. AppendSubMenu(edit_menu, "编辑")
        copy_item=wx. MenuItem(edit_menu, 100, text="复制", kind=wx. ITEM_
NORMAL)#对应步骤 3
        edit_menu. Append(copy_item)
```

```
        cut_item = wx. MenuItem(edit_menu, 101, text = "剪切", kind = wx. ITEM_NORMAL)
        edit_menu. Append(cut_item)
        paste_item = wx. MenuItem(edit_menu, 102, text = "粘贴", kind = wx. ITEM_NORMAL)
        edit_menu. Append(paste_item)
if __name__ == "__main__":
app = wx. App( )
frame = MyFrame( )
frame. Show( )
app. MainLoop( )
```

执行结果如图 12. 31 所示。

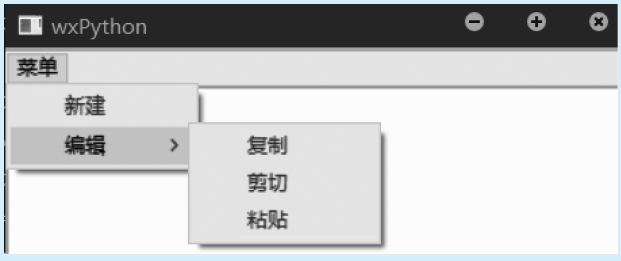

图 12. 31　菜单演示

习　题

1. 猜数字。用 Tkinter 开发一个猜数字游戏，游戏中电脑随机生成 1024 以内的数字，使用者去猜，如果猜的数字过大过小都会进行提示并把范围缩小，程序要统计玩家猜的次数，最后提示猜测次数。

2. 利用 wxPython 插入三个复选框到一个框架中。

3. 创建一个包含“登陆+注册+写日记”的图形用户界面。设计思路为：通过一个主页面来控制登录界面（这是一种常用的设计理念，使主页面更为简洁）。如果登录信息正确，那么跳转到下一个页面。如果要注册，则点击进入注册页面。请通过四组文件代码、三个页面来简单实现上述功能。