第3章

视频讲解

Tkinter图形界面应用——图形界面万年历

3.1 图形界面万年历功能介绍

程序实现制作一个 Tkinter 图形界面日历(只显示阳历日期),用户选择某年某月,图形化显示当月日历功能。运行效果如图 3-1 所示。

图 3-1 Tkinter 图形界面万年历

3.2 程序设计的思路

1. 计算指定月份的第一天是星期几

不使用日历生成模块 calendar 提供的日期计算方法,而是根据 1800 年 1 月 1 号为星期三,以此推算指定月份的第一天是星期几。

2. 创建日历界面

整个组件的布局是 8×7 的表格(grid)方式。

第 1 行显示日历头部,包括年月日的显示与选择;

第 2 行显示周日、周一、周二、周三、周四、周五、周六标签;

第 3～8 行显示本月日历信息。

创建本月日历信息其实就是在 6×7 的表格中预先放置 6×7 个标签(Label),分别表示 1～31 的情况(应该包含所有的情况)。将 1～31 从得到的位置开始打印出来,设置为 7 的倍数时换行。

grid 方式有两个最重要的参数,用来指定将组件放置到什么位置,一个是 row,另一个是 column。如果不指定 row,会将组件放置到第一个可用的行上;如果不指定 column,则使用第 1 列。注意,这里使用 grid 时不需要创建,直接使用行列就可以。

3. 更新日历

当对日历头部进行选择操作(改变日期)时,就会更新日历显示的内容。

3.3　关键技术

视频讲解

Tkinter 是 Python 的标准 GUI 库。由于 Tkinter 内置在 Python 的安装包中,因此只要安装好 Python 就能导入 Tkinter 库,而且 IDLE 也是用 Tkinter 编写而成的。对于简单的图形界面 Tkinter 是能应付自如的,使用 Tkinter 可以快速地创建 GUI 应用程序。本书主要采用 Tkinter 设计图形界面。

3.3.1　创建 Windows 窗口

【例 3-1】 Tkinter 创建一个 Windows 窗口的 GUI 程序。

```
import tkinter                          #导入 Tkinter 模块
root = tkinter.Tk()                     #创建 Windows 窗口对象
root.title('我的第一个 GUI 程序')        #设置窗口标题
root.mainloop()                         #进入消息循环,也就是显示窗口
```

可见 Tkinter 可以很方便地创建 Windows 窗口。

在创建 Windows 窗口对象后,可以使用 geometry()方法设置窗口的大小,格式如下:

```
窗口对象.geometry(size)
```

size 用于指定窗口大小,格式如下:

```
宽度 x 高度
```

注:x 是小写字母,不是乘号。

Tkinter 提供各种组件(控件),如按钮、标签和文本框,可在一个 GUI 应用程序中使用。这些组件通常被称为控件或者部件。目前常用的 Tkinter 组件如表 3-1 所示。

表 3-1　常用的 Tkinter 组件

控　　件	描　　述
Button	按钮控件,在程序中显示按钮
Canvas	画布控件,显示图形元素如线条或文本
Checkbutton	多选框控件,用于在程序中提供多项选择框

续表

控　件	描　述
Entry	输入控件,用于显示简单的文本内容
Frame	框架控件,在屏幕上显示一个矩形区域,多用来作为容器
Label	标签控件,可以显示文本和位图
Listbox	列表框控件,用来显示一个字符串列表给用户
Menubutton	菜单按钮控件,用于显示菜单项
Menu	菜单控件,显示菜单栏、下拉菜单和弹出菜单
Message	消息控件,用来显示多行文本,与 Label 比较类似
Radiobutton	单选按钮控件,显示一个单选的按钮状态
Scale	范围控件,显示一个数值刻度,为输出限定范围的数字区间
Scrollbar	滚动条控件,当内容超过可视化区域时使用,如列表框
Text	文本控件,用于显示多行文本
Toplevel	容器控件,用来提供一个单独的对话框,和 Frame 比较类似
Spinbox	输入控件,与 Entry 类似,但是可以指定输入范围值
PanedWindow	一个窗口布局管理的插件,可以包含一个或者多个子控件
LabelFrame	一个简单的容器控件,常用与复杂的窗口布局
tkMessageBox	用于显示应用程序的消息框

通过组件类的构造函数可以创建其对象实例。例如:

```
from tkinter import *
root = Tk()
button1 = Button(root, text = "确定")          #创建按钮组件
```

3.3.2　布局管理器

Tkinter 布局管理器(geometry manager)用于组织和管理父组件(往往是窗口)中子组件的布局方式。Tkinter 提供了 3 种不同风格的几何布局管理类:pack、grid 和 place。

1. pack 布局管理器

pack 布局管理器采用块的方式组织组件。pack 布局根据子组件创建生成的顺序,将其放在快速生成界面设计中而广泛采用。

调用子组件的方法 pack(),则该子组件在其父组件中采用 pack 布局:

```
pack( option = value,... )
```

pack()方法提供如表 3-2 所示的若干参数选项。

表 3-2　pack()方法提供的参数选项

选　项	描　述	取 值 范 围
side	停靠在父组件的哪一边上	'top'(默认值), 'bottom','left', 'right'
anchor	停靠位置,对应于东、南、西、北以及 4 个角	'n','s','e','w','nw','sw','se','ne','center'(默认值)

续表

选　项	描　述	取值范围
fill	填充空间	'x','y','both','none'
expand	扩展空间	0 或 1
ipadx,ipady	组件内部在 x/y 方向上填充的空间大小	单位为 c（厘米）、m(毫米)、i（英寸）、p（打印机的点）
padx,pady	组件外部在 x/y 方向上填充的空间大小	单位为 c（厘米）、m(毫米)、i（英寸）、p（打印机的点）

【例 3-2】 pack 布局管理器的 GUI 程序。运行效果如图 3-2 所示。

```
import tkinter
root = tkinter.Tk()
label = tkinter.Label(root,text = 'hello ,python')
label.pack()                                          #将 Label 组件添加到窗口中显示
button1 = tkinter.Button(root,text = 'BUTTON1')       #创建文字是'BUTTON1'的 Button 组件
button1.pack(side = tkinter.LEFT)                     #将 button1 组件添加到窗口中显示,左停靠
button2 = tkinter.Button(root,text = 'BUTTON2')       #创建文字是'BUTTON2'的 Button 组件
button2.pack(side = tkinter.RIGHT)                    #将 button2 组件添加到窗口中显示,右停靠
root.mainloop()
```

2. grid 布局管理器

grid(表格)布局管理器采用表格结构组织组件。子组件的位置由行/列确定的单元格决定,子组件可以跨越多行/列。每一列中,列宽由这一列中最宽的单元格确定。grid 布局适合于表格形式的布局,可以实现复杂的界面,因而被广泛采用。

图 3-2　pack 布局管理器

调用子组件的 grid()方法,则该子组件在其父组件中采用 grid 布局:

```
grid ( option = value,... )
```

grid()方法提供如表 3-3 所示的若干参数选项。

表 3-3　grid()方法提供的参数选项

选　项	描　述	取值范围
sticky	组件紧贴所在单元格的某一边角,对应于东、南、西、北以及四个角	'n','s','e','w','nw','sw','se','ne','center'(默认值)
row	单元格行号	整数
column	单元格列号	整数
rowspan	行跨度	整数
columnspan	列跨度	整数
ipadx,ipady	组件内部在 x/y 方向上填充的空间大小	单位为 c（厘米）、m(毫米)、i（英寸）、p（打印机的点）
padx,pady	组件外部在 x/y 方向上填充的空间大小	单位为 c（厘米）、m(毫米)、i（英寸）、p（打印机的点）

grid()有两个最重要的参数：一个是 row；另一个是 column。它们用来指定将子组件放置到什么位置，如果不指定 row，则会将子组件放置到第一个可用的行上；如果不指定 column，则使用第 0 列(首列)。

【例 3-3】 grid 布局管理器的 GUI 程序。运行效果如图 3-3 所示。

```
from tkinter import *
root = Tk()
#200x200 代表了初始化时主窗口的大小,280、280 代表了初始化时窗口所在的位置
root.geometry('200x200+280+280')
root.title('计算器示例')
#Grid 表格布局
L1 = Button(root, text = '1', width=5, bg = 'yellow')
L2 = Button(root, text = '2', width=5)
L3 = Button(root, text = '3', width=5)
L4 = Button(root, text = '4', width=5)
L5 = Button(root, text = '5', width=5, bg = 'green')
L6 = Button(root, text = '6', width=5)
L7 = Button(root, text = '7', width=5)
L8 = Button(root, text = '8', width=5)
L9 = Button(root, text = '9', width=5, bg = 'yellow')
L0 = Button(root, text = '0')
Lp = Button(root, text = '.')
L1.grid(row = 0, column = 0)          #按钮放置在 0 行 0 列
L2.grid(row = 0, column = 1)          #按钮放置在 0 行 1 列
L3.grid(row = 0, column = 2)          #按钮放置在 0 行 2 列
L4.grid(row = 1, column = 0)          #按钮放置在 1 行 0 列
L5.grid(row = 1, column = 1)          #按钮放置在 1 行 1 列
L6.grid(row = 1, column = 2)          #按钮放置在 1 行 2 列
L7.grid(row = 2, column = 0)          #按钮放置在 2 行 0 列
L8.grid(row = 2, column = 1)          #按钮放置在 2 行 1 列
L9.grid(row = 2, column = 2)          #按钮放置在 2 行 2 列
L0.grid(row = 3, column = 0,columnspan=2,sticky=E+W)        #跨 2 列,左右贴紧
Lp.grid(row = 3, column = 2,sticky=E+W)                     #左右贴紧
root.mainloop()
```

3. place 布局管理器

place 布局管理器允许指定组件的大小与位置。place 布局的优点是可以精确控制组件的位置，不足之处是改变窗口大小时，子组件不能随之灵活改变大小。

图 3-3 grid 布局管理器

调用子组件的方法 place()，则该子组件在其父组件中采用 place 布局：

```
place ( option = value,... )
```

place()方法提供如表 3-4 所示的若干参数选项，可以直接给参数选项赋值加以修改。

表 3-4　place()方法提供的参数选项

选　项	描　述	取值范围
x,y	将组件放到指定位置的绝对坐标	从 0 开始的整数
relx, rely	将组件放到指定位置的相对坐标	取值范围为 0～1.0
height,width	高度和宽度,单位为像素(px)	
anchor	对齐方式,对应于东、南、西、北以及 4 个角	'n','s','e','w','nw','sw','se','ne','center'('center'为默认值)

注意：Python 的坐标系是左上角为原点(0, 0)位置,向右是 x 坐标正方向,向下是 y 坐标正方向,这和数学的几何坐标系不同。

【例 3-4】 place 布局管理器的 GUI 示例程序。运行效果如图 3-4 所示。

图 3-4　place 布局管理器

```
from tkinter import *
root = Tk()
root.title("登录")
root['width'] = 200;root['height'] = 80
Label(root,text = '用户名',width = 6).place(x = 1,y = 1)              #绝对坐标(1,1)
Entry(root,width = 20).place(x = 45,y = 1)                            #绝对坐标(45,20)
Label(root,text = '密码',width = 6).place(x = 1,y = 20)               #绝对坐标(1,20)
Entry(root,width = 20, show = '*').place(x = 45,y = 20)               #绝对坐标(45,20)
Button(root,text = '登录',width = 8).place(x = 40,y = 40)             #绝对坐标(40,40)
Button(root,text = '取消',width = 8).place(x = 110,y = 40)            #绝对坐标(110,40)
root.mainloop()
```

3.3.3　OptionMenu 可选菜单

OptionMenu 可选菜单与组合框功能类似。

1. 创建 OptionMenu

OptionMenu 可选菜单的创建需要两个必要的参数：一个参数为与当前值绑定的变量,通常为 StringVar 类型；另一个参数是提供可选的内容列表,由 OptionMenu 的变参数指定,也可以使用列表。

【例 3-5】 OptionMenu 可选菜单的示例程序。

```
from tkinter import *
root = Tk()
v = StringVar(root)
v.set('Python')
om = OptionMenu(root,v,'Python','PHP','CPP','C','Java','JavaScript','VBScript')
#或者 om = OptionMenu(root,v,['Python','PHP','CPP','C','Java','JavaScript','VBScript'])
om.pack()
root.mainloop()、
```

运行效果如图 3-5 所示。单击 Python 项或者右边的按钮,就会弹出一个选择列表,列

出的是传给 OptionMenu 的选项列表，用户选择其中任意一个后，按钮左边的字符也会随之改变。

2. 获得选取的选项值

可以使用变量的 get()方法获得选取的选项值。

```
m = v.get()
```

【例 3-6】 获得选取的 OptionMenu 选项值的示例程序。

```
from tkinter import *
def ok(): #事件函数
    print( "value is", v.get())
    root.quit()
root = Tk()
v = StringVar(root)
v.set('Python')
om = OptionMenu(root,v,'Python','PHP','CPP','C','Java','JavaScript','VBScript')
om.pack()
button = Button(root, text = "OK", command = ok)        #OK 按钮
button.pack()
root.mainloop()
```

运行效果如图 3-6 所示。单击 OK 按钮，就会输出当前选择的选项值。

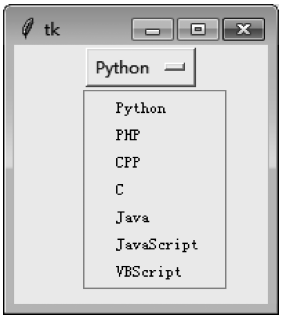

图 3-5 OptionMen 可选菜单

图 3-6 输出当前选择的选项值

3.3.4 grid 布局管理器的使用

grid 布局管理器会将控件放置到一个二维的表格里。窗体被分割成一系列的行和列，表格中的每个单元(cell)都可以放置一个控件。

```
w.grid_slaves(row = None, column = None)
```

返回由 w 窗体管理的控件列表 list。如果没有提供任何参数，返回包含所有控件的 list。提供 row 参数，返回该行所有控件；提供 column 参数，则返回该列所有控件。注意，行列号(row，column)从 0 开始算起。

例如，设置第 6 行第 3 列的标签文字为 15。

```
root.grid_slaves(5,2)[0]['text'] = '15'  #grid_slaves(5,2)返回(5,2)位置中所有控件的列表
```

3.4　图形界面万年历程序设计的步骤

导入相关模块，定义 Calendar 类。

```
from tkinter import *
import time
class Calendar:
  def __init__(self):
    self.vYear = StringVar()
    self.vMonth = StringVar()
    self.vDay = StringVar()
```

leap_year(self,year) 判断年份 year 是不是闰年。

```
def leap_year(self,year):
    #判断是不是闰年
    if (year % 400 == 0) or ((year % 4 == 0) and (year % 100 != 0)):
        return True
    else:
        return False
```

year_days(self,year,month)计算本月的天数。

```
def year_days(self,year,month):            #计算本月的天数
    if month in (1,3,5,7,8,10,12):
        return 31
    elif month in (4,6,9,11):
        return 30
    else:
        if self.leap_year(year) == True:
            return 29
    else:
            return 28
```

year_days(self,year,month,day)计算自 1800 年 1 月 1 日以来经过的天数。

```
def get_total_days(self,year,month,day):
    total_days = 0
    for m in range(1800,year):
        if self.leap_year(m) == True:
            total_days += 366
        else:
            total_days += 365
    for i in range(1,month):
        total_days += self.year_days(year,i)
    return total_days
```

calcFirstDayOfMonth(self,year,month)返回当月 1 日是星期几,由 1800 年 1 月 1 日是星期三推算。星期日返回 0,星期一返回 1,星期二返回 2,以此类推,星期六返回 6。

```
def calcFirstDayOfMonth(self,year,month):
    return (self.get_total_days(year,month,day) + 3) % 7
```

createMonth(self,master)预先放置 6×7 个 Label。

```
def createMonth(self,master):
    '''创建日历'''
    for i in range(6):
        for j in range(7):
            Label(master,text = '').grid(row = i + 2,column = j)
```

updateDate(self)得到当前选择的日期,根据计算的月初位置,把 1～31(也可能 28,29,30)重新为设置 6×7 个标签的文本。

```
def updateDate(self):
    '''更新日历'''
    #得到当前选择的日期
    year = int(self.vYear.get())
    month = int(self.vMonth.get())
    day = int(self.vDay.get())
    fd = self.calcFirstDayOfMonth(year,month)
    #设置所有标签文本为空
    for i in range(6):
        for j in range(7):
            #返回 grid 中(i +2,j)位置的组件
            root.grid_slaves(i +2,j)[0]['text'] = ''
    #计算本月的天数
    days = self.year_days(year,month)
    #重新设置标签文本为本月的日期
    for i in range(1,days + 1):
        root.grid_slaves( (i + fd - 1)//7 + 2 , (i + fd - 1) % 7 )[0]['text'] = str(i)
```

drawHeader(self,master)添加日历头部,包括年、月、日的显示与选择。

```
def drawHeader(self,master):
    '''添加日历头部'''
    #得到当前的日期,设置为默认值
    now = time.localtime(time.time())
    col_idx = 0
    #创建年份组件
    self.vYear = StringVar()
    self.vYear.set(now[0])
    Label(master,text = '年').grid(row = 0,column = col_idx);col_idx += 1
    #设置年份可选菜单 OptionMenu 项,OptionMenu 功能与 combox 相似
    omYear = OptionMenu(master,self.vYear, * tuple(range(2005,2020)))
```

```
        omYear.grid(row = 0,column = col_idx);col_idx += 1

        #创建月份组件
        self.vMonth.set(now[1])
        Label(master,text = '月').grid(row = 0,column = col_idx);col_idx += 1
        #设置月份可选菜单 OptionMenu 项
        omMonth = OptionMenu(master,self.vMonth, * tuple(range(1,13)))
        omMonth.grid(row = 0,column = col_idx);col_idx += 1

        #创建日组件
        self.vDay.set(now[2])
        Label(master,text = '日').grid(row = 0,column = col_idx);col_idx += 1
        #设置日可选菜单 OptionMenu 项
        omDay = OptionMenu(master,self.vDay, * tuple(range(1,32)))
        omDay.grid(row = 0,column = col_idx);col_idx += 1

        #创建'更新日历'按钮
        btUpdate = Button(master,text = '更新日历',command = self.updateDate)
        btUpdate.grid(row = 0,column = col_idx);col_idx += 1
        #打印星期标签
        weeks = ['周日','周一','周二','周三','周四','周五','周六']
        for week in weeks:
            Label(master,text = week).grid(row = 1,column = weeks.index(week))
```

主程序创建 Calendar()实例对象，并添加日历头部和预先放置 6×7 个标签，最后显示本月日历信息。

```
root = Tk()
root.title("万年历")
AppCal = Calendar()
AppCal.drawHeader(root)          #添加日历头部
AppCal.createMonth(root)         #预先放置 6×7 个 Label
AppCal.updateDate()              #显示本月日历信息
root.mainloop()
```

至此完成 Tkinter 图形界面的万年历。