

# Tkinter 教程

这系列教程完全以代码的形式来写的，目标是：读者看代码和注释就可以理解代码的意思。但这里的读者需要具备的几项技能：

- 1 熟悉 python 语言的基础，如果还没有，先看一下 python 的教程吧,英文官方 (<http://docs.python.org/tut/tut.html>);
- 2 对界面编程有一定的了解，知道基本的概念就可以了；
- 3 对 Tk 有兴趣，别以为她是已经过时的技术，如果丧失了学习的兴趣，那肯定无法完成了；
- 4 不要以 Ctrl+C/Ctrl+V 的方式使用本教程（虽然它可以这样直接运行），自己输入，你会发现自己原来也会犯这样的错误；
- 5 安装了 python2.5且确认安装了 Tkinter 模块（默认就安装了，如果你没有强制的把它去掉的话），下载 python2.5 (<http://www.python.org/download/>);
- 6 如果在阅读教程中有不明白的，不要强迫自己，直接跳过去，继续下一个内容。

Tkinter 教程系列教程的特点：

- 7 他不是一本经过文字润色的文章，全部是代码，作者在必要的时候使用注释来解释；
- 8 以组件为章节进行介绍，每个组件又分为不同的例子，各个例子可以单独使用，分别使用序号标注；
- 9 各个例子的使用“注释+序号”的格式表示开始，下一个例子的开始为上一个例子的结束；
- 10 全部使用结构化编程（SP），没有面向对象的概念(OO)；
- 11 基本上包含了 TKinter 的所有的控件，根据每个控件的使用方法，选择性的介绍了其属性和方法，没有全部介绍，全部的介绍查看 Tkinter 的官方参考 (<http://www.pythonware.com/library/tkinter/introduction/>);
- 12 例子不是百分百的完美，甚至有的例子没有按照 Tkinter 参考的描述完成，原因由于作者没有看懂：（
- 13 参考书籍：<http://www.pythonware.com/library/tkinter/introduction/>，如有冲突以 Tkinter 参考为准

最后祝各位 Tk 一路快乐！

# Label

```
#Tkinter 教程之 Label 篇
''' 1.Label 的第一个例子
text 属性使用方法
'''

#要使用 Tk 模块，除非你不想使用这个模块，那整个教程就不需要看了
from Tkinter import *
#初始化 Tk
root = Tk()
#创建一个 label，使用编码，到现在为止还没有使用过直接通过“drag-and-drop”就可以完成的 IDE。
label = Label(root, text = 'Hello Tkinter')
#显示 label，必须含有此语句
label.pack()
#root.pack()
#但 root 是不需要（严格地说是必须不这样使用），否则解释器抱怨
#进入消息循环
root.mainloop()
#控件的显示步骤：
#1. 创建这个控件
#2. 指定这个空间的 master，即这个控件属于哪一个
#3. 告诉 GM(geometry manager)有一个控件产生了
'''

还有更简单的一个例子：将‘Hello Tkinter’打印到标题上，Label 也不用创建了
from Tkinter import *
root = Tk()
root.title('hello Tkinter')
root.mainloop()
再没法儿简化了，就这样吧
'''

''' 2. 在 label 上使用内置位图
bitmap 的使用方法
'''

from Tkinter import *
#初始化 Tk
root = Tk()
#创建一个 label，使用编码，到现在为止还没有使用过直接通过“drag-and-drop”就可以完成的 IDE。
label = Label(root, bitmap = 'error')
#上面的代码使用了内置位图 error

#显示 label，必须含有此语句
```

```
label.pack()
```

```
#进入消息循环
```

```
root.mainloop()
```

```
'''
```

其他可用的位图:

```
* error
```

```
* hourglass
```

```
* info
```

```
* questhead
```

```
* question
```

```
* warning
```

```
* gray12
```

```
* gray25
```

```
* gray50
```

```
* gray75
```

若要查看各自的效果, 可以使用相应的名称将 bitmpa = 'error' 替换。

据说还可以使用自己指定的位图文件, 网上找了一下, 格式如下:

```
Label(root, bitmap="@/path/bitmapname")
```

不过我试了一下, 从来没有成功过, 我已经将位图该为单色的了:(

另: 还有的网上的文章说明如何使用 PhotoImage 和 BitmapImage 显示 bmp 或 gif 文件, 提到一点

防止图像文件被 python 自动回收(garbage collected), 应将 bmp 或 gif 放到全局(global)或实体

(instance)中, 使用如下两种方法, 仍未奏效:

```
'''
```

```
#使用 image 属性
```

```
# bm = PhotoImage(file = 'c:\\python.gif')
```

```
# label = Label(root, image = bm)
```

```
# label.bm = bm
```

```
#错误信息:
```

```
#TclError: image "pyimageXX" doesn't exist
```

```
#使用 bitmap 属性
```

```
# bm = BitmapImage(file='c:\\python2.bmp')
```

```
# label = Label(root, bitmap=bm)
```

```
# label.bm = bm
```

```
# label.pack()
```

```
#错误信息:
```

```
#TclError: format error in bitmap data
```

```
'''
```

虽然二者均没有起作用, 还是要说明一下, bitmap 与 image 的关系, 如果同时指定这两参数, image

优先。

```

'''
''' 3. 改变控件的前景色和背景色
fg: 前景色
bg: 背景色
设置背景色的一个大的用处是：可以判断控件的大小（不同的控件使用不同的颜色，后续
内容
可以使用此特性来调试 container）
'''

from Tkinter import *
root = Tk()
#在创建 Label 时指定各自使用的颜色
''' 可以使用的颜色值：'''
#使用颜色名称
Label(root, fg = 'red', bg = 'blue', text = 'Hello I am Tkinter').pack()
#使用颜色值#RRGGBB
Label(root, fg = 'red', bg = '#FF00FF', text = 'Hello I am Tkinter').pack()
#使用系统相关的颜色值（Windows），不建议使用这样的值，不利于平台移植
Label(root, fg = 'red', bg = 'SystemButtonShadow', text = 'Hello I am Tkinter').pack()
root.mainloop()
'''

(1). 使用颜色名称
Red
Green
Blue
Yellow
LightBlue
.....

(2). 使用#RRGGBB
label = Label(root, fg = 'red', bg = '#FF00FF', text = 'Hello I am Tkinter')
指定背景色为绯红色

(3). 除此之外，Tk 还支持与 OS 相关的颜色值，如 Windows 支持
SystemActiveBorder,
SystemActiveCaption,
SystemAppWorkspace,
SystemBackground,
.....
'''

''' 4. 设置宽度与高度
width:    宽度
height:   高度
'''

from Tkinter import *
root = Tk()
#创建三个 Label，分别显示 red, blue, yellow

```

```

#注意三个 Label 的大小，它们均与文本的长度有关
Label(root, text = 'red', bg = 'red').pack()
Label(root, text = 'blue', bg = 'blue').pack()
Label(root, text = 'yellow', bg = 'yellow').pack()

#再创建三个 Label，与上次不同的是这三个 Label 均使用 width 和 height 属性
#三个 Label 的大小由 width 和 height 指定
Label(root, bg = 'red', width = 10, height = 3).pack()
Label(root, bg = 'blue', width = 10, height = 3).pack()
Label(root, bg = 'yellow', width = 10, height = 3).pack()
root.mainloop()
''' 5. 同时使用图像与文本
compound:    指定文本(text)与图像(bitmap/image)是如何在Label上显示,缺省为None,
当指定 image/bitmap 时, 文本(text)将被覆盖, 只显示图像了。可以使用的值:
    left:      图像居左
    right:     图像居右
    top:       图像居上
    bottom:    图像居下
    center:    文字覆盖在图像上
bitmap/image:
    显示在 Label 上的图像
text:
    显示在 Label 上的文本
label = Label(root, text = 'Error', compound = 'left', bitmap = 'error')
'''

from Tkinter import *
root = Tk()
#演示 compound 的使用方法
#图像与文本在 Label 中的位置
#图像居下
Label(root, text = 'botton', compound = 'bottom', bitmap = 'error').pack()
#图像居上
Label(root, text = 'top', compound = 'top', bitmap = 'error').pack()
#图像居右
Label(root, text = 'right', compound = 'right', bitmap = 'error').pack()
#图像居左
Label(root, text = 'left', compound = 'left', bitmap = 'error').pack()
#文字覆盖在图像上
Label(root, text = 'center', compound = 'center', bitmap = 'error').pack()

#消息循环
root.mainloop()

''' 6. 文本的多行显示

```

在 Tk004中，使用 width 和 height 来指定控件的大小，如果指定的大小无法满足文本的要求是，会出现

什么现象呢？如下代码：

```
Label(root,bg = 'welcome to jcodeer.cublog.cn',width = 10,height = 3).pack()
```

运行程序，超出 Label 的那部分文本被截断了，常用的方法是：使用自动换行功能，及当文本长度大于

控件的宽度时，文本应该换到下一行显示，Tk 不会自动处理，但提供了属性：

wraplength: 指定多少单位后开始换行

justify: 指定多行的对齐方式

anchor: 指定文本(text)或图像(bitmap/image)在 Label 中的显示位置

可用的值：

e

w

n

s

ne

se

sw

sn

center

布局如下图

	nw	n	ne
	w	center	e
	sw	s	se

'''

```
from Tkinter import *
```

```
root = Tk()
```

```
#左对齐，文本居中
```

```
Label(root,text = 'welcome to jcodeer.cublog.cn',bg = 'yellow',width = 40,height = 3,wraplength = 80,justify = 'left').pack()
```

```
#居中对齐，文本居左
```

```
Label(root,text = 'welcome to jcodeer.cublog.cn',bg = 'red',width = 40,height = 3,wraplength = 80,anchor = 'w').pack()
```

```
#居中对齐，文本居右
```

```
Label(root,text = 'welcome to jcodeer.cublog.cn',bg = 'blue',width = 40,height = 3,wraplength = 80,anchor = 'e').pack()
```

```
root.mainloop()
```

'''

运行一下程序就可以直观的看出，justify 与 anchor 的区别了：一个用于控制多行的对齐；另一个用于

控制整个文本块在 Label 中的位置

```
''
```

## Button(1)

```
#Tkinter 教程之 Button 篇(1)
#Button 功能触发事件
''' 1. 一个简单的 Button 应用'''
from Tkinter import *
#定义 Button 的回调函数
def helloButton():
    print 'hello button'
root = Tk()
#通过 command 属性来指定 Button 的回调函数
Button(root, text = 'Hello Button', command = helloButton).pack()
root.mainloop()
```

```
'''
```

执行的结果:每次点击一次,程序向标准输出打印'hello button',以上为 Button 使用方法,可以

再做一下简化,如不设置 Button 的回调函数,这样也是允许的但这样的结果与 Label 没有什么太

大的区别,只是外观看起来有所不同罢了,失去了 Button 的作用。

```
from Tkinter import *
root = Tk()
#下面的 relief = FLAT 设置,就是一个 Label 了!!!
Button(root, text = 'hello button', relief=FLAT).pack()
root.mainloop()
'''
```

''' 2. 测试 Button 的 relief 属性'''

#运行下面的代码可以看到 Button 的各个不同效果,均没有回调函数。

```
from Tkinter import *
root = Tk()
#flat, groove, raised, ridge, solid, or sunken
Button(root, text = 'hello button', relief=FLAT).pack()
Button(root, text = 'hello button', relief=GROOVE).pack()
Button(root, text = 'hello button', relief=RAISED).pack()
Button(root, text = 'hello button', relief=RIDGE).pack()
Button(root, text = 'hello button', relief=SOLID).pack()
Button(root, text = 'hello button', relief=SUNKEN).pack()

root.mainloop()
```

```
'''
```

Button 显示图像

image: 可以使用 gif 图像, 图像的加载方法 `img = PhotoImage(root, file = filepath`

bitmap: 使用 X11 格式的 bitmap, Windows 的 Bitmap 没法显示的, 在 Windows 下使用 GIMP2.4 将 windows

Bitmap 转换为 xbm 文件, 依旧无法使用. linux 下的 X11 bitmap 编辑器生成的 bitmap 还没有测试, 但可

以使用内置的位图。

(1). 使用位图文件

```
bp = BitmapImage(file = "c:\\python2.xbm")
```

```
Button(root, bitmap = bp).pack()
```

(2). 使用位图数据

```
BITMAP = """
```

```
#define im_width 32
```

```
#define im_height 32
```

```
static char im_bits[] = {
```

```
0xaf, 0x6d, 0xeb, 0xd6, 0x55, 0xdb, 0xb6, 0x2f,
```

```
0xaf, 0xaa, 0x6a, 0x6d, 0x55, 0x7b, 0xd7, 0x1b,
```

```
0xad, 0xd6, 0xb5, 0xae, 0xad, 0x55, 0x6f, 0x05,
```

```
0xad, 0xba, 0xab, 0xd6, 0xaa, 0xd5, 0x5f, 0x93,
```

```
0xad, 0x76, 0x7d, 0x67, 0x5a, 0xd5, 0xd7, 0xa3,
```

```
0xad, 0xbd, 0xfe, 0xea, 0x5a, 0xab, 0x69, 0xb3,
```

```
0xad, 0x55, 0xde, 0xd8, 0x2e, 0x2b, 0xb5, 0x6a,
```

```
0x69, 0x4b, 0x3f, 0xb4, 0x9e, 0x92, 0xb5, 0xed,
```

```
0xd5, 0xca, 0x9c, 0xb4, 0x5a, 0xa1, 0x2a, 0x6d,
```

```
0xad, 0x6c, 0x5f, 0xda, 0x2c, 0x91, 0xbb, 0xf6,
```

```
0xad, 0xaa, 0x96, 0xaa, 0x5a, 0xca, 0x9d, 0xfe,
```

```
0x2c, 0xa5, 0x2a, 0xd3, 0x9a, 0x8a, 0x4f, 0xfd,
```

```
0x2c, 0x25, 0x4a, 0x6b, 0x4d, 0x45, 0x9f, 0xba,
```

```
0x1a, 0xaa, 0x7a, 0xb5, 0xaa, 0x44, 0x6b, 0x5b,
```

```
0x1a, 0x55, 0xfd, 0x5e, 0x4e, 0xa2, 0x6b, 0x59,
```

```
0x9a, 0xa4, 0xde, 0x4a, 0x4a, 0xd2, 0xf5, 0xaa
```

```
};
```

```
"""
```

使用 tuple 数据来创建图像

```
bmp = BitmapImage(data = BITMAP)
```

```
Button(root, bitmap = bmp)
```

```
'''
```

''' 3. 与 Label 一样, Button 也可以同时显示文本与图像, 使用属性 compound'''

```
from Tkinter import *
```

```
root = Tk()
```

#图像居下, 居上, 居右, 居左, 文字位于图像之上

```
Button(root, text = 'bottom', compound = 'bottom', bitmap = 'error').pack()
```

```
Button(root, text = 'top', compound = 'top', bitmap = 'error').pack()
```

```
Button(root, text = 'right', compound = 'right', bitmap = 'error').pack()
```



```

Button(root, text = 'left', compound = 'left', bitmap = 'error').pack()
Button(root, text = 'center', compound = 'center', bitmap = 'error').pack()
#消息循环
root.mainloop()

```

''' 4. 控件焦点问题

创建三个 Button, 各自对应回调函数; 将第二个 Button 设置焦点, 程序运行是按 “Enter”, 判断

程序的打印结果

'''

```

from Tkinter import *

```

```

def cb1():
    print 'button1 clicked'
def cb2(event):
    print 'button2 clicked'
def cb3():
    print 'button3 clicked'

```

```

root = Tk()

```

```

b1 = Button(root, text = 'Button1', command = cb1)
b2 = Button(root, text = 'Button2')
b2.bind("<Return>", cb2)
b3 = Button(root, text = 'Button3', command = cb3)
b1.pack()
b2.pack()
b3.pack()

```

```

b2.focus_set()
root.mainloop()
'''

```

上例中使用了 bind 方法, 它建立事件与回调函数(响应函数)之间的关系, 每当产生<Enter>事件

后, 程序便自动的调用 cb2, 与 cb1, cb3不同的是, 它本身还带有一个参数----event, 这个参数传递

响应事件的信息。

'''

```

from Tkinter import *

```

```

def printEventInfo(event):
    print 'event.time = ', event.time
    print 'event.type = ', event.type
    print 'event.WidgetId = ', event.widget
    print 'event.KeySymbol = ', event.keysym

```

```

root = Tk()
b = Button(root, text = 'Infomation')
b.bind("<Return>", printEventInfo)
b.pack()
b.focus_set()
root.mainloop()

```

'''

犯了个错误，将<Return>写成<Enter>了，结果是：当鼠标进入 Button 区域后，事件 printEventInfo

被调用。程序打印出了 event 的信息。

'''

## Button(2)

# Tkinter 教程之 Button 篇(2)

''' 5. 指定 Button 的宽度与高度

width: 宽度

height: 高度

使用三种方式:

1. 创建 Button 对象时，指定宽度与高度
2. 使用属性 width 和 height 来指定宽度与高度
3. 使用 configure 方法来指定宽度与高度

'''

```

from Tkinter import *

```

```

root = Tk()

```

```

b1 = Button(root, text = '30X1', width = 30, height = 2)

```

```

b1.pack()

```

```

b2 = Button(root, text = '30X2')

```

```

b2['width'] = 30

```

```

b2['height'] = 3

```

```

b2.pack()

```

```

b3 = Button(root, text = '30X3')

```

```

b3.configure(width = 30, height = 3)

```

```

b3.pack()

```

```

root.mainloop()

```

# 上述的三种方法同样也适合其他的控件

''' 6. 设置 Button 文本在控件上的显示位置

anchor:

使用的值为:n(north), s(south), w(west), e(east)和 ne, nw, se, sw, 就是地图上的标识位置了, 使用

width 和 height 属性是为了显示各个属性的不同。

'''

```
from Tkinter import *
```

```
root = Tk()
```

#简单就是美!

```
for a in ['n','s','e','w','ne','nw','se','sw']:
```

```
    Button(root,
        text = 'anchor',
        anchor = a,
        width = 30,
        height = 4).pack()
```

#如果看的不习惯, 就使用下面的代码。

```
# Button(root, text = 'anchor', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'center', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'n', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 's', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'e', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'w', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'ne', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'nw', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'se', width = 30, height = 4).pack()
# Button(root, text = 'anchor', anchor = 'sw', width = 30, height = 4).pack()
```

```
root.mainloop()
```

''' 7. 改变 Button 的前景色与背景色

fg: 前景色

bg: 背景色

'''

```
from Tkinter import *
```

```
root = Tk()
```

```
bfg = Button(root, text = 'change foreground', fg = 'red')
bfg.pack()
```

```
bbg = Button(root, text = 'change backgroud', bg = 'blue')
bbg.pack()
```

```
root.mainloop()
```

''' 8. 设置 Button 的边框

bd(bordwidth):缺省为1或2个像素

'''

```
# 创建5个 Button 边框宽度依次为: 0, 2, 4, 6, 8
```

```
from Tkinter import *
root = Tk()
for b in [0, 1, 2, 3, 4]:
    Button(root,
           text = string(b),
           bd = b).pack()
root.mainloop()
```

```
'''9. 设置 Button 的风格
```

```
relief/raised/sunken/groove/ridge
'''
```

```
from Tkinter import *
root = Tk()
for r in ['raised', 'sunken', 'groove', 'ridge']:
    Button(root,
           text = r,
           relief = r,
           width = 30).pack()
root.mainloop()
```

```
'''10. 设置 Button 状态
```

```
normal/active/disabled
'''
```

```
from Tkinter import *
root = Tk()
def statePrint():
    print 'state'
for r in ['normal', 'active', 'disabled']:
    Button(root,
           text = r,
           state = r,
           width = 30,
           command = statePrint).pack()
root.mainloop()
```

#例子中将三个 Button 在回调函数都设置为 statePrint, 运行程序只有 normal 和 active 激活了回调函数, 而 disable 按钮则没有, 对于暂时不

#需要按钮起作用时, 可以将它的 state 设置为 disabled 属性

```
'''11. 绑定 Button 与变量
```

```
设置 Button 在 textvariable 属性
'''
```

```
from Tkinter import *
root = Tk()
```

```
def changeText():
    if b['text'] == 'text':
        v.set('change')
        print 'change'
    else:
        v.set('text')
        print 'text'
v = StringVar()
b = Button(root, textvariable = v, command = changeText)
v.set('text')
b.pack()
root.mainloop()

'''
将变量 v 与 Button 绑定，当 v 值变化时，Button 显示的文本也随之变化
'''
```

## Entry

```
#Tkinter 教程之 Entry 篇
#Entry 用来输入单行文本
''' 1. 第一个 Entry 程序'''
from Tkinter import *
root = Tk()
Entry(root, text = 'input your text here').pack()
root.mainloop()
#上面的代码目的是创建一个 Entry 对象，并在 Entry 上显示'input your text here'，运行此代码，并没有看到文本的显示，由此可知与 Lable 和 Button 不同，Entry 的 text 属性不可以设置 Entry 的文本

''' 2. 在 Entry 中设定初始值，使用 textvariable 将变量与 Entry 绑定'''
from Tkinter import *
root = Tk()
e = StringVar()
entry = Entry(root, textvariable = e)
e.set('input your text here')
entry.pack()
root.mainloop()

#上面的例子中将变量 e 与 Entry 绑定，然后将 e 的值设置为'input your text here'，程序运行时的初始值便设置了。

''' 3. 设置为只读 Entry.
```

Entry 的另一个比较有用的属性，设置为只读，不允许用户对它的值改变。

设置 state 属性为 'readonly'

```
'''  
from Tkinter import *  
root = Tk()  
e = StringVar()  
entry = Entry(root, textvariable = e)  
e.set('input your text here')  
entry.pack()  
entry['state'] = 'readonly'  
root.mainloop()
```

#实际上 Entry 的属性值可以使用的也为 normal/active/disabled, 'readonly' 与 disabled 一样

''' 4. 设置为密码输入框

#将 Entry 作为一个密码输入框来使用，即不显示用户输入的内容值，用特定符号代替。使用 show 属性

show 来指定。

'''

```
from Tkinter import *  
root = Tk()  
e = StringVar()  
entry = Entry(root, textvariable = e)  
e.set('input your text here')  
entry.pack()  
#使用*来显示输入的内容，如果喜欢可以改为其它字符  
entry['show'] = '*'  
#分别使用*#$显示输入的文本内容  
for mask in ['*', '#', '$']:  
    e = StringVar()  
    entry = Entry(root, textvariable = e)  
    e.set('password')  
    entry.pack()  
    entry['show'] = mask  
  
root.mainloop()
```

''' 5. 验证输入的内容是否符合要求。

使用 validate 来校验输入的内容

使用 validate 方法来限制输入的内容

这是一个有问题的例子，无法调用 validateText 回调函数

'''

```
from Tkinter import *
```

```

root = Tk()
e = StringVar()
def validateText(contents):
    print contents
    return contents.isalnum()

entry = Entry(root, validate = 'key', textvariable = e, validatecommand =
validateText)
entry.pack()

root.mainloop()
'''

```

文档中说明使用 `validate` 来接受的事件，使用 `validatecommand` 来确定输入的内容是否合法，但

如何传入参数？没找到相应的说明

'''

#还有其他的属性 `fg/bg/relief/width/height/justify/state` 使用方法与 `Button` 相同，不再举例。

## CheckButton

#Tkinter 教程之 Checkbutton 篇

#Checkbutton 又称为多选按钮，可以表示两种状态：On 和 Off，可以设置回调函数，每当点击此按钮时回调函数被调用

''' 1. 一个简单的 Checkbutton 例子'''

#创建一个 Checkbutton, 显示文本为"python"

```

from Tkinter import *
root = Tk()
Checkbutton(root, text = 'python').pack()
root.mainloop()

```

''' 2. 设置 Checkbutton 的回调函数'''

```

from Tkinter import *
def callCheckbutton():
    print 'you check this button'
root = Tk()
Checkbutton(root, text = 'check python', command = callCheckbutton).pack()
root.mainloop()

```

#不管 Checkbutton 的状态如何，此回调函数都会被调用

''' 3. 通过回调函数改变 Checkbutton 的显示文本 text 的值'''

```

from Tkinter import *
def callCheckbutton():

```

#改变 v 的值，即改变 Checkbutton 的显示值

```
v.set('check Tkinter')
```

```
root = Tk()
```

```
v = StringVar()
```

```
v.set('check python')
```

#绑定 v 到 Checkbutton 的属性 textvariable

```
Checkbutton(root, text = 'check python', textvariable = v, command =  
callCheckbutton).pack()
```

```
root.mainloop()
```

'''4. 上述的 textvariable 使用方法与 Button 的用法完全相同，使用此例是为了区别 Checkbutton 的另外的一个属性 variable, 此属性与 textvariable 不同，它是与这个控件本身绑定，Checkbutton 自己有值：On 和 Off 值，缺省状态 On 为1，Off 为0，如：'''

#显示 Checkbutton 的值

```
from Tkinter import *
```

```
root = Tk()
```

#将一整数与 Checkbutton 的值绑定，每次点击 Checkbutton，将打印出当前的值

```
v = IntVar()
```

```
def callCheckbutton():
```

```
    print v.get()
```

```
Checkbutton(root,  
             variable = v,  
             text = 'checkbutton value',  
             command = callCheckbutton).pack()
```

```
root.mainloop()
```

'''5. Checkbutton 的值不仅仅是1或0，可以是其他类型的数值，可以通过 onvalue 和 offvalue 属性设置 Checkbutton 的状态值，如下代码将 On 设置为'python', Off 值设置为'Tkinter'，程序的打印值将不再是0或1，而是'Tkinter' 或 'python' '''

```
from Tkinter import *
```

```
root = Tk()
```

#将一字符串与 Checkbutton 的值绑定，每次点击 Checkbutton，将打印出当前的值

```
v = StringVar()
```

```
def callCheckbutton():
```

```
    print v.get()
```

```
Checkbutton(root,  
             variable = v,  
             text = 'checkbutton value',  
             onvalue = 'python',          #设置 On 的值  
             offvalue = 'tkinter')
```



# RadioButton

#Tkinter 教程之 Radiobutton 篇

#Radiobutton 为单选按钮，即在同一组内只能有一个按钮被选中，每当选中组内的一个按钮时，其它的按钮自动改为非选中态，与其他控件不同的是：它有组的概念

''' 1. 创建一个简单的 Radiobutton '''

```
from Tkinter import *
root = Tk()
Radiobutton(root, text = 'python').pack()
Radiobutton(root, text = 'tkinter').pack()
Radiobutton(root, text = 'widget').pack()
```

root.mainloop()

#不指定绑定变量，每个 Radiobutton 自成一组

''' 2. 创建一个 Radiobutton 组，使用绑定变量来设置选中哦的按钮 '''

```
from Tkinter import *
root = Tk()
#创建一个 Radiobutton 组，创建三个 Radiobutton，并绑定到整型变量 v
#选中 value=1的按钮
v = IntVar()
v.set(1)
for i in range(3):
    Radiobutton(root, variable = v, text = 'python', value = i).pack()
```

root.mainloop()

''' 3. 创建两个不同的组 '''

```
from Tkinter import *
root = Tk()
vLang = IntVar()
vOS = IntVar()
vLang.set(1)
vOS.set(2)

for v in [vLang, vOS]:    #创建两个组
    for i in range(3):    #每个组含有3个按钮
        Radiobutton(root,
                      variable = v,
                      value = i,
                      text = 'python' + str(i)
                      ).pack()
```

root.mainloop()

#不同的组，各个按钮互不影响。

''' 4. 如果同一个组中的按钮使用相同的 value，则这两个按钮的工作方式完全相同'''

```
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = IntVar()
v.set(1)
for i in range(3):
    Radiobutton(root,
                 variable = v,
                 value = 1,
                 text = 'python' + str(i)
                 ).pack()
for i in range(3):
    Radiobutton(root,
                 variable = v,
                 value = i,
                 text = 'python' + str(2 + i)
                 ).pack()

root.mainloop()
```

#上述的例子中共有4个 value 为1的值，当选中其中的一个时，其他三个也会被选中；选中除了这四个以外的按钮时，四个按钮全部取消

''' 5. 与 Checkbutton 类似，每个 Radiobutton 可以有自己的处理函数，每当点击按钮时，系统会调用相应的处理函数'''

```
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = IntVar()
v.set(0)
def r1():
    print 'call r1'
def r2():
    print 'call r2'
def r3():
    print 'call r3'
def r4():
    print 'call r4'
i = 0
#创建8个按钮，其中两个两个的 value 值相同
for r in [r1, r2, r3, r4]:
    Radiobutton(root,
                 variable = v,
                 text = 'radio button',
                 value = i,
```

```

        command = r
    ).pack()
Radiobutton(root,
            variable = v,
            text = 'radio button',
            value = i,
            command = r
        ).pack()

    i += 1

root.mainloop()
#注意虽然同时可以选中两个按钮，但每次点击按钮，执行的代码只有一次

```

'''6.Radiobutton 另一个比较实用的属性是 indicatoron, 缺省情况下为1，如果将这个属性改为0，则其外观是 Sunken'''

```

from Tkinter import *
root = Tk()
v = IntVar()
v.set(1)
for i in range(3):
    Radiobutton(root,
                variable = v,
                indicatoron = 0,
                text = 'python & tkinter',
                value = i
            ).pack()

root.mainloop()
#Radiobutton 表示按钮的弹起或按下两种状态

```

## Listbox

```

#Tkinter 教程之 Listbox 篇
#Listbox 为列表框控件，它可以包含一个或多个文本项(text item)，可以设置为单选或多选

'''1. 创建一个 Listbox，向其中添加三个 item'''
from Tkinter import *
root = Tk()
lb = Listbox(root)
for item in ['python', 'tkinter', 'widget']:
    lb.insert(END, item)
lb.pack()
root.mainloop()

```

'''2. 创建一个可以多选的 Listbox, 使用属性 selectmode'''

```
from Tkinter import *
root = Tk()
lb = Listbox(root, selectmode = MULTIPLE)
for item in ['python', 'tkinter', 'widget']:
    lb.insert(END, item)
lb.pack()
root.mainloop()
# 依次点击这三个 item, 均显示为选中状态。
# 属性 MULTIPLE 允许多选, 每次点击 item, 它将改变自己的当前选状态, 与 Checkbox 有点相似
```

'''3. 这个属性 selectmode 还可以设置为 BROWSE, 可以通过鼠标来移动 Listbox 中的选中位置 (不是移动 item), 这个属性也是 Listbox 在默认设置的值, 这个程序与1. 程序运行的结果的一样的。'''

```
from Tkinter import *
root = Tk()
lb = Listbox(root, selectmode = BROWSE)
for item in ['python', 'tkinter', 'widget']:
    lb.insert(END, item)
lb.pack()
root.mainloop()
#使用鼠标进行拖动, 可以看到选中的位置随之变化。
# 与 BROWSE 相似 的为 SINGLE, 但不支持鼠标移动选中位置。
from Tkinter import *
root = Tk()
lb = Listbox(root, selectmode = BROWSE)
for item in ['python', 'tkinter', 'widget']:
    lb.insert(END, item)
lb.pack()
root.mainloop()
#使用鼠标进行拖动, 没有任何变化
```

'''4. 使用 selectmode = EXPANDED 使用 Listbox 来支持 Shift 和 Control。'''

```
from Tkinter import *
root = Tk()
lb = Listbox(root, selectmode = EXTENDED)
for item in ['python', 'tkinter', 'widget']:
    lb.insert(END, item)
lb.pack()
root.mainloop()
#运行程序, 点中 “python”, shift + 点击 “widget”, 会选中所有的 item
```

#运行程序，点中“python”，control + 点击“widget”，会选中 python 和 widget，第二项 tkinter 处于非选中状态

'''5. 向 Listbox 中添加一个 item'''

# 以上的例子均使用了 insert 来向 Listbox 中添加 一个 item，这个函数有两个属性一个为添加的索引值，另一个为添加的项(item)

# 有两个特殊的值 ACTIVE 和 END，ACTIVE 是向当前选中的 item 前插入一个（即使用当前选中的索引作为插入位置）；END 是向

# Listbox 的最后一项添加插入一项

# 先向 Listbox 中追加三个 item，再在 Listbox 开始添加三项

```
from Tkinter import *
```

```
root = Tk()
```

```
lb = Listbox(root)
```

```
for item in ['python', 'tkinter', 'widget']:
```

```
    lb.insert(END, item)
```

#只添加一项将[]作为一个 item

```
#lb.insert(0, ['linux', 'windows', 'unix'])
```

#添加三项，每个 string 为一个 item

```
lb.insert(0, 'linux', 'windows', 'unix')
```

```
lb.pack()
```

```
root.mainloop()
```

'''6. 删除 Listbox 中的项，使用 delete，这个函数也有两个参数，第一个为开始的索引值；第二个为结束的索引值，如果不指定则只删除第一个索引项。'''

```
from Tkinter import *
```

```
root = Tk()
```

```
lb = Listbox(root)
```

```
for i in range(10):
```

```
    lb.insert(END, str(i))
```

```
lb.delete(1, 3)
```

```
lb.pack()
```

```
root.mainloop()
```

#运行程序，只有值0456789, 1-3被删除

#删除全部内容, 使用 delete 指定第一个索引值0和最后一个参数 END，即可

```
#lb.delete(0, END)
```

'''7. 选中操作函数，使用函数实现。selection\_set 函数有两个参数第一个为开始的索引；第二个为结束的索引，如果不指定则只选中第一个参数指定的索引项'''

```
from Tkinter import *
```

```
root = Tk()
```

```
lb = Listbox(root)
```

```
for i in range(10):
```

```
    lb.insert(END, str(i))
```

```
lb.selection_set(0, 10)
```

```
lb.pack()
```

```
root.mainloop()
# 程序运行结果，选中了所有的项。 此代码并未指定 Listbox 为 MULTIPLE 或 EXTENDED，
# 查通过 selection_set 仍旧可以对 Listbox
# 进行操作。
```

# 与之相对的便是取消选中的函数了，参数与 selection\_set 在参数相同，如下代码取消索引从0—3在状态

```
from Tkinter import *
root = Tk()
lb = Listbox(root)
for i in range(10):
    lb.insert(END, str(i))
lb.selection_set(0, 10)
lb.selection_clear(0, 3)
lb.pack()
root.mainloop()
```

'''8. 得到当前 Listbox 中的 item 个数'''

```
from Tkinter import *
root = Tk()
lb = Listbox(root)
for i in range(10):
    lb.insert(END, str(i))
lb.delete(3)
print lb.size()
lb.pack()
root.mainloop()
```

# 首先向 Listbox 中添加了10个 item，然后删除索引为3在 item，最后的打印结果为9，即当前的 Listbox 中只有9项

'''9. 返回指定索引的项'''

```
from Tkinter import *
root = Tk()
lb = Listbox(root)
for i in range(10):
    lb.insert(END, str(i*100))
print lb.get(3)
lb.pack()
root.mainloop()
```

# 返回值为300

# get 也为两个参数的函数，可以返回多个项(item)，如下返回索引值3—7的值

```
from Tkinter import *
root = Tk()
lb = Listbox(root)
```

```

for i in range(10):
    lb.insert(END, str(i*100))
print lb.get(3, 7)
lb.pack()
root.mainloop()
#返回值为(' 300', ' 400', ' 500', ' 600', ' 700'), 是一个 tuple 类型。

```

''' 10. 返回当前返回的项的索引, 不是 item 的值'''

```

from Tkinter import *
root = Tk()
lb = Listbox(root)
for i in range(10):
    lb.insert(END, str(i*100))
lb.selection_set(3, 8)
print lb.curselection()
lb.pack()
root.mainloop()
# 返回值 为 (' 3', ' 4', ' 5', ' 6', ' 7', ' 8') , 而 不 是
(' 300', ' 400', ' 500', ' 600', ' 700', ' 800'), 哑然无法直接得到各项的值, 知道了索引, 得到值
#就很容易了:lb.get()就可以实现。

```

''' 11. 判断 一个项是否被选中, 使用索引。'''

```

from Tkinter import *
root = Tk()
lb = Listbox(root)
for i in range(10):
    lb.insert(END, str(i*100))
lb.selection_set(3, 8)
print lb.selection_includes(8)
print lb.selection_includes(0)

lb.pack()
root.mainloop()
#返回结果: True Flase, 即8包含在选中的索引中, 0不包含在选中的索引中

```

''' 12. Listbox 与变量绑定'''

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar()
lb = Listbox(root, listvariable = v)
for i in range(10):
    lb.insert(END, str(i*100))

```

```

#打印当前列表中的项值
print v.get()
#输出: ('0', '100', '200', '300', '400', '500', '600', '700', '800', '900')
#改变 v 的值, 使用 tuple 可以与 item 对应
v.set(('1000', '200'))
#结果只有两项了1000和200
lb.pack()
root.mainloop()

''' 13. Listbox 与事件绑定'''
# 它不支持 command 属性来设置回调函数了, 使用 bind 来指定回调函数, 打印当前选中的
值
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
def printList(event):
    print lb.get(lb.curselection())
lb = Listbox(root)
lb.bind('<Double-Button-1>', printList)
for i in range(10):
    lb.insert(END, str(i*100))
lb.pack()
root.mainloop()

#还有一个比较实用的功能没有介绍: 滚动条的添加, 留到后面介绍 Scrollbar 的时候再一
并介绍

```

## Scale

```

''' Tkinter 教程之 Scale 篇'''
#Scale 为输出限定范围的数字区间, 可以为之指定最大值, 最小值及步距值
''' 1. 创建一个 Scale'''
from Tkinter import *
root = Tk()
Scale(root).pack()
root.mainloop()
#创建一个垂直 Scale, 最大值为100, 最小值为0, 步距值为1。这个参数设置也就是 Scale
的缺省设置了。
''' 2. 改变这三个参数, 生成 一个水平 Scale, 最小值为-500, 最大值为500, 步距值为5'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
Scale(root,

```



```

        from_ = -500, #设置最大值
        to = 500, #设置最小值
        resolution = 5, #设置步距值
        orient = HORIZONTAL #设置水平方向
    ).pack()
root.mainloop()
#注意 from_ 的使用方式，在其后添加了“_”，避免与关键字 from 的冲突

```

'''3. Scale 绑定变量'''

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar()
Scale(root,
        from_ = 0, #设置最小值
        to = 100.0, #设置最大值
        resolution = 0.0001, #设置步距值
        orient = HORIZONTAL, #设置水平方向
        variable = v #绑定变量
    ).pack()
print v.get()
root.mainloop()

```

#v 的值与 Scale 的值一致

'''4. 使用回调函数打印当前的值'''

```

# -*- coding: cp936 -*-
from Tkinter import *

root = Tk()
def printScale(text):
    print 'text = ', text
    print 'v = ', v.get()
v = StringVar()
Scale(root,
        from_ = 0, #设置最小值
        to = 100.0, #设置最大值
        resolution = 0.0001, #设置步距值
        orient = HORIZONTAL, #设置水平方向
        variable = v, #绑定变量
        command = printScale #设置回调函数
    ).pack()
print v.get()
root.mainloop()

```

#这个回调函数有一个参数，这个值是当前的 Scale 的值，每移动一个步距就会调用一次这个函数，只保证最后一个肯定会调用，中间的有

#可能不会调用,通过上例可以看到二者的值是完全一样的。

'''5.控制显示位数,可以理解为: Scale 的值为一整形,在输出显示时,它将会被转化为一字符串,如1.2转化为1.2或1.2000都是可以的'''

#属性 digits 是控制显示的数字位数,将上面的例子中的数据以8位形式显示,在最后一位会添加一个0

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
def printScale(text):
```

```
    print 'text = ', text
```

```
    print 'v = ', v.get()
```

```
v = StringVar()
```

```
Scale(root,
```

```
    from_ = 0, #设置最小值
```

```
    to = 100.0, #设置最大值
```

```
    resolution = 0.0001, #设置步距值
```

```
    orient = HORIZONTAL, #设置水平方向
```

```
    digits = 8, #设置显示的位数为8
```

```
    variable = v, #绑定变量
```

```
    command = printScale #设置回调函数
```

```
    ).pack()
```

```
print v.get()
```

```
root.mainloop()
```

'''6.设置 Scale 的标签属性 label'''

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
Scale(root,
```

```
    from_ = 0, #设置最大值
```

```
    to = 100.0, #设置最小值
```

```
    orient = HORIZONTAL, #设置水平方向
```

```
    label = 'choice:', #设置标签值
```

```
    ).pack()
```

```
root.mainloop()
```

#由 label 设置的值会显示在水平 Scale 的上方,用于提示信息

'''7.设置/取得 Scale 的值'''

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```

root = Tk()
sl = Scale(root)
sl.set(50) #将 Scale 的值设置为50
print sl.get() #打印当前的 Scale 的值
sl.pack()
root.mainloop()
#slider 的位置位于了中间，sl.set(50)起作用了，打印值为50。

```

## Spinbox

```

'''Tkinter 教程之 Spinbox 篇'''
#与 Entry 类似，但可以指定输入范围值
'''1. 创建一个 Spinbox'''
from Tkinter import *
root = Tk()
Spinbox(root).pack()
root.mainloop()
#只是创建了一个 Spinbox，其它的什么也做不了，与 Scale 不同，Scale 使用缺省值就可以控制 值的改变。

'''2. 设置 Spinbox 的最大、最小值和步距值'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
Spinbox(root,
        from_ = 0, #设置最小值
        to = 100, #设置最大值
        increment = 5 #设置增量值为5，这个与 Scale 的 resolution 意思相同
    ).pack()
root.mainloop()

'''3. 设置 Spinbox 的值，设置属性 values，设置此值后，每次更新值将使用 values 指定的值，'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
sb = Spinbox(root,
        values = (0, 2, 20, 40, -1),
        increment = 2
    )

sb.pack()
# 打印当前的 Spinbox 的值，为一 tuple
print sb['values']
root.mainloop()
#显示的第一个值为0, up 按钮则为2, 20, 40, -1，不再是增2操作，它会使用 tuple 的索引递

```

增，至到 tuple 的最后一个项时，将不再增加；  
#down 按钮与 up 按钮恰好相反，它使用 tuple 的索引递减

''' 4. Spinbox 绑定变量 '''

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
v = StringVar()
```

```
sb = Spinbox(root,  
              values = (0, 2, 20, 40, -1),  
              increment = 2,  
              textvariable = v  
            )
```

```
v.set(20)
```

```
print v.get()
```

```
sb.pack()
```

```
# 打印当前的 Spinbox 的值，为一 tuple
```

```
root.mainloop()
```

#上面的代码将变量 v 与 sb 绑定，并将 Spinbox 的初始值设置为20，运行程序, Spinbox 的值显示为20，再点击 up 按钮，此时值变为40，

#即 tuple 的下一个值，再看下面的代码，与这个不同的是设置的值不包含在 tuple 之内

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
v = StringVar()
```

```
sb = Spinbox(root,  
              values = (0, 2, 20, 40, -1),  
              increment = 2,  
              textvariable = v  
            )
```

```
v.set(200)
```

```
print v.get()
```

```
sb.pack()
```

```
# 打印当前的 Spinbox 的值，为一 tuple
```

```
root.mainloop()
```

#运行程序，显示的值为200，再次点击 up 按钮，显示的值为2，即虽然 Spinbox 能将值显示出来，但并不会将200添加到变量中，此时的

#索引值依旧为0，因为没有找到200的项。当点击 up 时，索引值变为1，即显示的值为2。

''' 5. 设置 Spinbox 的回调函数 '''

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
def printSpin():
```

```
    print 'Spinbox'
```

```

sb = Spinbox(root,
              from_ = 0, #最小值
              to = 10, #最大值
              command = printSpin#回调函数
              )

```

```
sb.pack()
```

```
root.mainloop()
```

#每次点击 Spinbox 按钮时就会调用 printSpin 函数，打印出'Spinbox'。与 Scale 不同的是：它不需要参数。

'''6. 打印 Spinbox 的当前内容，是显示的值，不是 values 的值。'''

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
def printSpin():
```

```
    # 使用 get() 方法来得到当前的显示值
```

```
    print sb.get()
```

```
sb = Spinbox(root,
```

```
            from_ = 0, #最小值
```

```
            to = 10, #最大值
```

```
            command = printSpin#回调函数
```

```
)
```

```
sb.pack()
```

```
root.mainloop()
```

#每次点击 Spinbox 按钮时就会调用 printSpin 函数，打印出 Spinbox 的当前值。

'''7. 删除 Spinbox 指定位置的字符（这是个有问题的程序）'''

#在回调函数中使用 delete, Spinbox 初始值为1234, 点击 up 一次 Spinbox 的值变为235, 再次点击变为36, 再次点击变为7, 但实际执行结果

#为第一次点击235, 再次点击为234, 以后所有的点击操作均为此值。不知为何。

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
def printSpin():
```

```
    sb.delete(0)
```

```
    print sb.get()
```

```
sb = Spinbox(root,
```

```
            from_ = 1234, #最小值
```

```
            to = 9999, #最大值
```

```
            increment = 1,
```

```
            command = printSpin#回调函数
```

```

        )
sb.pack()
root.mainloop()

# 如果不使用回调函数，两次调用 delete 则可以正常，工作如下代码：
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()

sb = Spinbox(root,
              from_ = 1234, #最小值
              to = 9999, #最大值
              increment = 1
              )
sb.delete(0)
sb.delete(0)
print sb.get()
sb.pack()
root.mainloop()
#此程序正常，可以打印出正确结果'34'

```

''' 关于 delete 回调函数异常问题，又使用如下代码作了实验'''

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
def printSpin():
    # 使用 delete() 方法删除指定索引的字符
    sb.delete(0)
    print sb.get()
sb = Spinbox(root,
              values = (1234, 234, 34, 4),
              command = printSpin
              )
sb.pack()
root.mainloop()
#则这个程序点击 up 可以打印出34，再次点击则为空。
'''

```

这个是可以工作的:①当前的值为1234，②点击 up 按钮时，程序调用回调函数 printSpin 将 Spinbox 的当

前值变为234；③Spinbox 查找值为234的项，得到索引为1，即当前的索引值变为1, up 还会将索引增1，即变为2，所有显示的值

为34, 为了更好理解，用如下代码再次测试：

```

'''
# -*- coding: cp936 -*-

```

```

from Tkinter import *
root = Tk()
def printSpin():
    # 使用 delete() 方法删除指定索引的字符
    sb.delete(0)
    print sb.get()
sb = Spinbox(root,
              values = (1234567890, 234567890, 34567890, 4567890, 567890,
                        67890, 7890, 890, 90, 0),
              command = printSpin #回调函数
              )
sb.pack()
root.mainloop()
#这个程序显示的依次是1234567890, 34567890, 567890, 7890, 90。
#还不了解内部工作原理，先这样理解吧，使用 delete 时注意可能会出现这样的问题。

```

'''8. 在 Spinbox 指定位置插入文本'''

#在每项后面添加.00表示精度，同样使用回调函数实现，代码如下：

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
def printSpin():
    # 使用 get() 方法来得到当前的显示值
    sb.insert(END, '.00')
    print sb.get()

sb = Spinbox(root,
              from_ = 1234, #最小值
              to = 9999, #最大值
              increment = 1,
              command = printSpin#回调函数
              )
sb.pack()
root.mainloop()
#每次点击 Spinbox 按钮时就会调用 printSpin 函数，当前的显示值均添加了两个有数字
".00"。这个与 delete 不同，倒是可以正确显示。
'''

```

delete 所遇到的问题，insert 真的就不会发生吗？再次对上面的代码进行测试，代码如下：

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
def printSpin():

```

```

# 使用 get() 方法来得到当前的显示值
sb.insert(END, '0')
print sb.get()

sb = Spinbox(root,
              from_ = 1234, #最小值
              to = 9999, #最大值
              increment = 1,
              command = printSpin #回调函数
              )

sb.pack()
root.mainloop()
#在每个项的后加一个0, 即将值变为原来的10倍, 则程序的输出结果为123450, 99990, 同样也出现了异常
#现象, 第一个例子中出现正确的情况纯粹是个偶然, 在整数的后添加.00相当于没有对其值进行改变, 故下次
#使用的值依旧没有变化。

```

## Scrollbar

```

'''Tkinter 教程之 Scrollbar 篇'''
#Scrollbar (滚动条), 可以单独使用, 但最多的还是与其它控件 (Listbox, Text, Canva 等) 结合使用
'''1. 创建一个 Scrollbar'''
from Tkinter import *
root = Tk()
Scrollbar(root).pack()
root.mainloop()
#显示了一个 Scrollbar, 但什么也做不了, 无法拖动 slider。
from Tkinter import *
root = Tk()
sl = Scrollbar(root)
sl.set(0.5, 0)
sl.pack()
root.mainloop()
'''2. 通过 set 方法来设置 slider 的位置'''
#使用水平滚动条, 通过 set 将值设置为 (0.5, 1), 即 slider 占整个 Scrollbar 的一半
from Tkinter import *
root = Tk()
sl = Scrollbar(root, orient = HORIZONTAL)
sl.set(0.5, 1)
sl.pack()
root.mainloop()
'''3. 使用回调函数 (不建议这样使用)'''

```



```
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
def scrollCall(moveto, pos):
    #如何得到两个参数: 使用如下打印中的信息, 可以看到解释器传给 scrollCall 函数的两个参数, 一个为
    #moveto, 参考手册可以得知, 它是当拖动 slider 时调用的函数; 另一个参数为 slider 的当前位置, 我们
    #可以通过 set 函数来设置 slider 的位置, 因此使用这个 pos 就可以完成控制 slider 的位置。
    #print moveto, pos
    sl.set(pos, 0)
    print sl.get()
sl = Scrollbar(root, orient = HORIZONTAL, command = scrollCall)
sl.pack()
root.mainloop()
#这样还有一个严重问题, 只能对其进行拖动。对两个按钮及 pagedwon/pageup 的响应, 由于 up 按钮响应的为三个参数, 故会出
#现异常。这个例子只是用来说明 command 属性是可用的, 如果喜欢自己可以处理所有的消息, 将 scrollCall 是否可以改为变参数函数?
#对于不同的输入分别进行不同的处理。
```

''' 4. 单独使用还是比较少见, 大部分应用还是与其它控件的绑定, 以下是将一个 Listbox 与 Scrollbar 绑定的例子'''

```
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
lb = Listbox(root)
sl = Scrollbar(root)
sl.pack(side = RIGHT, fill = Y)
#side 指定 Scrollbar 为居右; fill 指定填充整个剩余区域, 到 WM 在时候再详细介绍这几个属性。
#下面的这句是关键: 指定 Listbox 的 yscrollbar 的回调函数为 Scrollbar 的 set
lb['yscrollcommand'] = sl.set
for i in range(100):
    lb.insert(END, str(i))
#side 指定 Listbox 为居左
lb.pack(side = LEFT)
#下面的这句是关键: 指定 Scrollbar 的 command 的回调函数是 Listbar 的 yview
sl['command'] = lb.yview
root.mainloop()
```

''' 5. 这样理解二者之间的关系: 当 Listbox 改变时, Scrollbar 调用 set 以改变 slider 的位置; 当 Scrollbar 改变了 slider 的位置时, Listbox 调用 yview 以显示新的 list 项, 为了演示这两种关系先将 yscrollcommad 与 scrollbar 的 set 解除绑定, 看看会有什么效果

```
'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
lb = Listbox(root)
sl = Scrollbar(root)
sl.pack(side = RIGHT, fill = Y)
#解除 Listbox 的 yscrollcommand 与 Scrollbar 的 set 绑定
lb['yscrollcommand'] = sl.set
for i in range(100):
    lb.insert(END, str(i))
#使用索引为50的元素可见
lb.see(50)
lb.pack(side = LEFT)
sl['command'] = lb.yview
root.mainloop()
#运行结果，Listbox 显示了50项，即 Listbox 的视图已经到50了，但 Scrollbar 的 slider
仍旧位于0处。也就是说 Scrollbar 没有收到 set
#的命令。即说明解除此绑定，Scrollbar 将不再响应 Listbox 视图改变的消息。但仍可以
使用 Scrollbar 的 slider 来移动 Listbox 的视图。
```

''' 6. 再测试一下，解除 Scrollbar 的 command 与 Listbox 的 yview 的关系，测试代码如下：

```
'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
lb = Listbox(root)
sl = Scrollbar(root)
sl.pack(side = RIGHT, fill = Y)
#下面的这句是关键：指定 Listbox 的 yscrollbar 的回调函数为 Scrollbar 的 set
lb['yscrollcommand'] = sl.set
for i in range(100):
    lb.insert(END, str(i*100))
#使用索引为50的元素可见
lb.see(50)
lb.pack(side = LEFT)
#解除 Scrollbar 的 command 与 Listbox 的 yview 的关系
sl['command'] = lb.yview
root.mainloop()
#运行程序，Scrollbar 的 slider 已经到了50位置，也就是说 Scrollbar 响应了 Listbox 视图改变的消息，调用 了自己的 set 函数。
#进行操作：拖动 slider 或点击 up/down 按钮，Listbox 的视图没有任何反应，即 Listbox
不会响应 Scrollbar 的消息了。
```

# Menu

```
'''Tkinter 教程之 Menu 篇'''
'''1. 创建一个简单的 Menu'''
#添加菜单 hello 和 quit, 将 hello 菜单与 hello 函数绑定; quit 菜单与 root.quit 绑定
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
def hello():
    print 'hello menu'
menubar = Menu(root)
#创建主菜单, 每个菜单对应的回调函数都是 hello
for item in ['Python', 'PHP', 'CPP', 'C', 'Java', 'JavaScript', 'VBScript']:
    menubar.add_command(label = item, command = hello)
#将 root 的 menu 属性设置为 menubar
root['menu'] = menubar
root.mainloop()
#这个菜单没有下拉菜单, 仅包含两个菜单项

'''2. 添加下拉菜单'''
from Tkinter import *
root = Tk()
def hello():
    print 'hello menu'
menubar = Menu(root)

filemenu = Menu(menubar, tearoff = 0)
for item in ['Python', 'PHP', 'CPP', 'C', 'Java', 'JavaScript', 'VBScript']:
    filemenu.add_command(label = item, command = hello)
#将 menubar 的 menu 属性指定为 filemenu, 即 filemenu 为 menubar 的下拉菜单
menubar.add_cascade(label = 'Language', menu = filemenu)
root['menu'] = menubar
root.mainloop()

'''3. 向菜单中添加 Checkbutton 项'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
#每次打印出各个变量的当前值
def printItem():
    print 'Python = ', vPython.get()
    print 'PHP = ', vPHP.get()
    print 'CPP = ', vCPP.get()
```

```

print 'C = ', vC.get()
print 'Java = ', vJava.get()
print 'JavaScript = ', vJavaScript.get()
print 'VBScript = ', vVBScript.get()

menubar = Menu(root)

vPython = StringVar()
vPHP     = StringVar()
vCPP     = StringVar()
vC       = StringVar()
vJava    = StringVar()
vJavaScript = StringVar()
vVBScript  = StringVar()

filemenu = Menu(menubar, tearoff = 0)
for k,v in {'Python':vPython,
            'PHP':vPHP,
            'CPP':vCPP,
            'C':vC,
            'Java':vJava,
            'JavaScript':vJavaScript,
            'VBScript':vVBScript}.items():
    #绑定变量与回调函数
    filemenu.add_checkbutton(label = k, command = printItem, variable = v)
#将 menubar 的 menu 属性指定为 filemenu, 即 filemenu 为 menubar 的下拉菜单
menubar.add_cascade(label = 'Language', menu = filemenu)
root['menu'] = menubar
root.mainloop()
#程序运行, 使用了 Checkbutton, 并通过 printItem 将每个 Checkbutton 在当前值打印出来。

''' 4. 向菜单 中添加 Radiobutton 项 '''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()

menubar = Menu(root)
vLang = StringVar()
#每次打印出当前选中的语言
def printItem():
    print 'vLang = ', vLang.get()
filemenu = Menu(menubar, tearoff = 0)
for k in ['Python', 'PHP', 'CPP', 'C', 'Java', 'JavaScript', 'VBScript']:

```

```

        #绑定变量与回调函数，指定的变量 vLang 将这几项划为一组
        filemenu.add_radiobutton(label = k,command = printItem,variable = vLang)
#将 menubar 的 menu 属性指定为 filemenu，即 filemenu 为 menubar 的下拉菜单
menubar.add_cascade(label = 'Language',menu = filemenu)
root['menu'] = menubar
root.mainloop()
#程序每次打印出当前选中的语言
#与 Checkbutton 不同的是，同一个组内只有一个处于选中状态。

```

### ''' 5. 向菜单中添加分隔符'''

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
menubar = Menu(root)

#每次打印出当前选中的语言
def printItem():
    print 'add_separator'

filemenu = Menu(menubar,tearoff = 0)
for k in ['Python','PHP','CPP','C','Java','JavaScript','VBScript']:
    filemenu.add_command(label = k,command = printItem)
    #将各个菜单项使用分隔符隔开
    filemenu.add_separator()
menubar.add_cascade(label = 'Language',menu = filemenu)
root['menu'] = menubar
root.mainloop()
#分隔符将相关的菜单项进行分组，只是 UI 上的实现，程序上没有任何改变，它也不执行任何的命令

```

### ''' 6. 将以上的例5中的菜单改为右击弹出菜单'''

```

#方法是通过绑定鼠标右键，每当点击时弹出这个菜单，去掉与 root 的关联
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
menubar = Menu(root)

def printItem():
    print 'popup menu'

filemenu = Menu(menubar,tearoff = 0)
for k in ['Python','PHP','CPP','C','Java','JavaScript','VBScript']:
    filemenu.add_command(label = k,command = printItem)
    filemenu.add_separator()

```

```
menubar.add_cascade(label = 'Language', menu = filemenu)
#此时就不要将 root 的 menu 设置为 menubar 了
#root['menu'] = menubar
def popup(event):
    #显示菜单
    menubar.post(event.x_root, event.y_root)
#在这里相应鼠标的右键事件，右击时调用 popup, 此时与菜单绑定的是 root，可以设置为
其它的控件，在绑定的控件上右击就可以弹出菜单
root.bind('<Button-3>', popup)
root.mainloop()
#运行测试一个，可以看到各个菜单 项的功能都是可以使用的，所以弹出菜单与一般的菜
单功能是一样的，只是弹出的方式不同而已。
```

''' 7. 以下的代码演示了菜单项的操作方法，包括添加各种菜单项，删除一个或多个菜单项'''

```
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
menubar = Menu(root)

def printItem():
    print 'add_separator'

filemenu = Menu(menubar, tearoff = 0)
for k in range(5):
    filemenu.add_command(label = str(k), command = printItem)
menubar.add_cascade(label = 'Language', menu = filemenu)
```

''' 以下为向菜单中添加项的操作'''

```
#在索引1添加一菜单 command 项
filemenu.insert_command(1, label = '1000', command = printItem)
#在索引2添加一菜单 checkbox 项
filemenu.insert_checkbutton(2, label = '2000', command = printItem)
#在索引3添加一菜单 radiobutton 项
filemenu.insert_radiobutton(3, label = '3000', command = printItem)
#将新添加的菜单项使用分隔符隔开
filemenu.insert_separator(1)
filemenu.insert_separator(5)
```

''' 以下为删除菜单项的操作'''

```
#删除索引6-9的菜单项
filemenu.delete(6, 9)
#删除索引为0的菜单项
filemenu.delete(0)
```

```
root['menu'] = menubar
```

```
root.mainloop()
```

#分隔符将相关的菜单项进行分组，只是 UI 上的实现，程序上没有任何改变，它也不执行任何的命令

## Menubutton

```
'''Tkinter 教程之 Menubutton 篇'''
```

```
''' 这是一个过时的控件，从 Tk8.0 开始将不再使用这个控件，取而代之的是 Menu, 这里介绍它是为了
```

```
兼容以前版本的 Tk，能够知道有这个东东就可以了'''
```

```
''' 1. 介绍一下 Menubutton 的常用 方法，可以看到与 Menu 的使用方法基本相同。'''
```

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```

```
root = Tk()
```

```
mbLang = Menubutton(root, text = 'Language')
```

```
mbLang.menu = Menu(mbLang)
```

```
#生成菜单项
```

```
for item in ['Python', 'PHP', 'CPP', 'C', 'Java', 'JavaScript', 'VBScript']:
```

```
    mbLang.menu.add_command(label = item)
```

```
mbLang['menu'] = mbLang.menu
```

```
mbLang.pack(side = LEFT)
```

#分隔符将相关的菜单项进行分组，只是 UI 上的实现，程序上没有任何改变，它也不执行任何的命令

```
#添加向菜单中添加 checkbutton 项
```

```
mbOS = Menubutton(root, text = 'OS')
```

```
mbOS.menu = Menu(mbOS)
```

```
for item in ['Unix', 'Linux', 'Solaris', 'Windows']:
```

```
    mbOS.menu.add_checkbutton(label = item)
```

```
mbOS['menu'] = mbOS.menu
```

```
mbOS.pack(side = LEFT)
```

```
#向菜单中添加 radiobutton 项
```

```
mbLinux = Menubutton(root, text = 'Linux')
```

```
mbLinux.menu = Menu(mbLinux)
```

```
for item in ['Redhat', 'Fedra', 'Suse', 'ubuntu', 'Debian']:
```

```
    mbLinux.menu.add_radiobutton(label = item)
```

```
mbLinux['menu'] = mbLinux.menu
```

```
mbLinux.pack(side = LEFT)
```

```

#对菜单项进行操作
#向 Language 菜单中添加一项“Ruby”, 以分隔符分开
mbLang.menu.add_separator()
mbLang.menu.add_command(label = 'Ruby')

#向 OS 菜单中第二项添加“FreeBSD”, 以分隔符分开
mbOS.menu.insert_separator(2)
mbOS.menu.insert_checkbutton(3, label = 'FreeBSD')
mbOS.menu.insert_separator(4)

#将 Linux 中的“Debian”删除
mbLinux.menu.delete(5)

root.mainloop()
#这个控件已经不再提倡使用, 取而代之的是 Menu, 使用这个比使用 Menubutton 更为方便。
如果不是特别需要不要使用这个控件。

```

## Message

```

''' Tkinter 教程之 Message 篇'''
#Message 也是用来显示文本的, 用法与 Label 基本一样
''' 1.. 创建一个简单的 Message'''
from Tkinter import *
root = Tk()
Message(root, text = 'hello Message').pack()
root.mainloop()
#运行程序, 可以看到 Hello 之后, Message 显示在它的下一行, 这也是 Message 的一个特性。Label 没有。

''' 2. 如果不让它换行的话, 指定足够大的宽度'''
from Tkinter import *
root = Tk()
Message(root, text = 'hello Message', width = 60).pack()
root.mainloop()
#运行程序, 可以看到 Hello 之后, Message 显示在它的下一行, 这也是 Message 的一个特性。Label 没有。

''' 3. 使用 aspect 属性指定宽高比例'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()

```



```

for i in range(10):
    Message(root, text = 'A'*i, aspect = 400).pack()
root.mainloop()
#默认情况向 wider/height = 1.5, 可以使用 aspect 属性, 设置为4, 即宽为高的4倍, 可以显示10个'A'

''' 4. Message 绑定变量'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar()
v.set('000')
for i in range(10):
    Message(root, text = 'A', textvariable = v).pack()
#打印当前的 v 值, 只要是其中的一个 Message 的值发生变化, 则此 v 值就会改变。
print v.get()
root.mainloop()
#绑定变量 v, 虽然创建 Message 时使用了 text 来指定 Message 的值, 绑定的变量优先级高, 可以改变 text
#指定的值。

''' 5. 测试一下 justify 属性'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
for i in [LEFT, RIGHT, CENTER]:
    Message(root, text = 'ABC DEF GHI', justify = i).pack()
root.mainloop()
#显示的文本自动断行, 上下行分别使用了左对齐, 右对齐和居中对齐

```

## OptionMenu

```

''' Tkinter 教程之 OptionMenu 篇'''
#OptionMenu 为可选菜单, 与 Combox 功能类似。
''' 1. 创建 OptionMenu'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar(root)
v.set('Python')
om = OptionMenu(root, v, 'Python', 'PHP', 'CPP', 'C', 'Java', 'JavaScript', 'VBScript')
om.pack()

```

```

root.mainloop()
#显示的文本自动断行，上下行分别使用了左对齐，右对齐和居中对齐
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar(root)
#创建一个 OptionMenu 控件
om = OptionMenu(root,
                 v,
                 'Python',
                 'PHP',
                 'CPP',
                 'C',
                 'Java',
                 'JavaScript',
                 'VBScript'
                )

om.pack()

```

root.mainloop()

#OptionMenu 的创建需要两个必要的参数，与当前值绑定的变量，通常为一 StringVar 类型；另一

#个是提供可选的内容列表，由 OptionMenu 的变参数指定。

## ''' 2. 设置 OptionMenu 的显示值'''

#当 OptionMenu 与变量绑定后，直接使用变量赋值的方法即可改变当前的值

```

# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar(root)
v.set('VBScript')
#创建一个 OptionMenu 控件
om = OptionMenu(root,
                 v,
                 'Python',
                 'PHP',
                 'CPP',
                 'C',
                 'Java',
                 'JavaScript',
                 'VBScript'
                )

om.pack()

```

```

print v.get()

root.mainloop()
#运行程序，OptionMenu 默认值为"VBScript", 打印出的数值也为"VBScript"
#如果设置的值不包含在当前的列表中，会是什么结果？如下的代码使用"Tkinter"来测试
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar(root)
v.set('Tkinter')
#创建一个 OptionMenu 控件
om = OptionMenu(root,
                 v,
                 'Python',
                 'PHP',
                 'CPP',
                 'C',
                 'Java',
                 'JavaScript',
                 'VBScript'
                )

om.pack()
print v.get()

root.mainloop()
#程序依旧是默认值改变为 Tkinter，打印结果也变为 Tkinter，但 Tkinter 不会添加到
OptionMenu 的列表中，也就是说，当选择其它的选项时，Tkinter 的值会丢失。
''' 3. 打印 OptionMenu 的值'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
v = StringVar(root)
v.set('Tkinter')
def printOption(event):
    print v.get()
#创建一个 OptionMenu 控件
om = OptionMenu(root,
                 v,
                 'Python',
                 'PHP',
                 'CPP',
                 'C',
                 'Java',
                 'JavaScript',

```

```

        'VBScript'
    )
om.bind('<Button-1>', printOption)
om.pack()

root.mainloop()
#每次点击 OptionMenu 程序打印出上次选中的项值
''' 使用 list 作为 OptionMenu 的选项'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
Lang = ['Python', 'PHP', 'CPP', 'C', 'Java', 'JavaScript', 'VBScript']
v = StringVar(root)
v.set('Tkinter')
def printOption(event):
    print v.get()
#创建一个 OptionMenu 控件, 使用了 apply 函数
om = apply(OptionMenu, (root, v) + tuple(Lang))
om.bind('<Button-1>', printOption)
om.pack()

root.mainloop()

```

## Frame

```

''' Tkinter 教程之 Frame 篇'''
#Frame 就是屏幕上的一块矩形区域, 多是用来作为容器 (container) 来布局窗体。
''' 1. 创建 Frame'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
#以不同的颜色区别各个 frame
for fm in ['red', 'blue', 'yellow', 'green', 'white', 'black']:
    #注意这个创建 Frame 的方法与其它创建控件的方法不同, 第一个参数不是 root
    Frame(height = 20, width = 400, bg = fm).pack()
root.mainloop()
#添加不同颜色的 Frame, 大小均为20*400
''' 2. 向 Frame 中添加 Widget'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
fm = []

```

```

#以不同的颜色区别各个 frame
for color in ['red', 'blue']:
    #注意这个创建 Frame 的方法与其它创建控件的方法不同，第一个参数不是 root
    fm.append(Frame(height = 200,width = 400,bg = color))
#向下面的 Frame 中添加一个 Label
Label(fm[1],text = 'Hello label').pack()
fm[0].pack()
fm[1].pack()
root.mainloop()
#Label 被添加到下面的 Frame 中了，而不是 root 默认的最上方。
#大部分的方法来自 gm, 留到后面 gm 时再介绍
''' 3. Tk8.4 以后 Frame 又添加了一类 LabelFrame，添加了 Title 的支持'''
from Tkinter import *
root = Tk()
for lf in ['red', 'blue', 'yellow']:
    #可以使用 text 属性指定 Frame 的 title
    LabelFrame(height = 200,width = 300,text = lf).pack()
root.mainloop()

```

## PanedWindow

```

''' Tkinter 教程之 PanedWindow'''
#PanedWindow(面板)为一 gm, 用来管理子 Widget
''' 1. 向 PanedWindow 中添加 Pane'''
#使用 add 方法
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
panes = PanedWindow(orient = VERTICAL)
panes.pack(fill = BOTH, expand = 1)
for w in [Label, Button, Checkbutton, Radiobutton]:
    panes.add(w(panes, text = 'hello'))
root.mainloop()
#每个 pane 中创建一个 widget
''' 2. 删除 PanedWindow 指定的 pane'''
#使用 forget/remove 方法
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
ws = []
panes = PanedWindow(orient = VERTICAL)
panes.pack(fill = BOTH, expand = 1)

```

```

#创建四个 pane
for w in [Label, Button, Checkbutton, Radiobutton]:
    ws.append(w(panes, text = 'hello'))
for w in ws:
    panes.add(w)
#从 panes 中删除包含子 Button 的 pane, 使用 remove 与 forget 相同
panes.forget(ws[1])
#panes.remove(ws[1])
root.mainloop()
#只有三个 widget, Button 已被删除。
'''3. 在 PanedWindow 指定位置添加一个 pane'''
#使用 paneconfig 方法
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
ws = []
ps = PanedWindow(orient = VERTICAL)
ps.pack(fill = BOTH, expand = 1)
#创建四个 pane
for w in [Label, Button, Checkbutton, Radiobutton]:
    ws.append(w(ps, text = 'hello'))
for w in ws:
    ps.add(w)
#在0之后添加一个 Label, 出错!!!
#ps.after(ws[0], Label(ps, text = 'world'))
#注意被添加的 widget 是第一个参数, after 指定是位于那一个 widget 之后
#不要与 after 方法混淆了
ps.paneconfig(Label(ps, text = 'world'), after = ws[0])
root.mainloop()
#这个 widget 主要也是用来做 Container 的, 使用了大量的 gm 方法。

```

## Toplevel

```

'''Tkinter 教程之 Toplevel 篇'''
#Toplevel 与 Frame 类似, 但它包含窗体属性 (如 Title)
'''1. 创建简单的 Toplevel'''
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
tl = Toplevel()
#为了区别 root 和 tl, 我们向 tl 中添加了一个 Label
Label(tl, text = 'hello label').pack()
root.mainloop()

```

#运行结果生成了两个窗体，一个是 root 启动的，另一个则是 Toplevel 创建的，它包含有一个 label;关闭 t1  
#则没有退出程序，Tk 仍旧工作；若关闭 Tk，整个 Tk 结束 t1 也结束，它不能单独存在。

### '''2. 设置 Toplevel 的属性'''

```
#title 设置标题
#geometry 设置宽和高
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
tl = Toplevel()
#设置 tl 的 title
tl.title('hello Toplevel')
#设置 tl 在宽和高
tl.geometry('400x300')
#为了区别 root 和 tl，我们向 tl 中添加了一个 Label
Label(tl, text = 'hello label').pack()
root.mainloop()
```

### '''3. 使用 Toplevel 自己制作提示框'''

```
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
mbYes, mbYesNo, mbYesNoCancel, mbYesNoAbort = 0, 1, 2, 4
#定义一个消息对话框，依据传入的参数不同，弹出不同的提示信息
def MessageBox(): #没有使用使用参数
    mbType = mbYesNo
    textShow = 'Yes'
    if mbType == mbYes:
        textShow = 'Yes'
    elif mbType == mbYesNo:
        textShow = 'YesNo'
    elif mbType == mbYesNoCancel:
        textShow = 'YesNoCancel'
    elif mbType == mbYesNoAbort:
        textShow = 'YesNoAbort'
    tl = Toplevel(height = 200, width = 400)
    Label(tl, text = textShow).pack()
#由 Button 来启动这个消息框，因为它使用了空的回调函数，故 MessageBox 改为了无参数形式，使用了固定
#值 mbYesNo
Button(root, text = 'click me', command = MessageBox).pack()
root.mainloop()
```

## Text (1)

```
''' Tkinter 教程之 Text 篇 (1) '''
''' 1. 创建第一个 Text '''
from Tkinter import *
root = Tk()
t = Text(root)
t.pack()
root.mainloop()
#root 中含有一 Text 控件, 可以在这个控件内输入文本, 可以使用 Ctrl+C/V 向 Text 内添加
#剪切板上的内容(文本), 不接受 Ctrl+Z 执行操作
''' 2. 向 Text 中添加文本 '''
#insert 方法添加文本内容
from Tkinter import *
root = Tk()
t = Text(root)
#向第一行, 第一列添加文本0123456789
t.insert(1.0, '0123456789')
#向第一行第一列添加文本 ABCDEFGHIJ
t.insert(1.0, 'ABCDEFGHIJ')
t.pack()
root.mainloop()
#insert 的第一个参数为索引; 第二个为添加的内容
''' 3. 使用 line.col 索引添加内容 '''
#使用 indexes 来添加 Text 的内容
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
t = Text(root)
# 向第一行, 第一列添加文本0123456789
t.insert(1.0, '0123456789')
t.insert('2. end', '\n')
# 向第一行第一列添加文本 ABCDEFGHIJ
t.insert(2.5, 'ABCDEFGHIJ')
t.pack()
root.mainloop()
# 可以看到使用 indexes 时, 如果其值超过了 Text 的 buffer 值, 程序不会抛出异常, 它
#会使用向给定值靠近。
''' mark 是用来表示在 Text 中位置的一类符号 '''
''' 4. 使用内置的 mark 控制添加位置 '''
#演示了内置的 mark: INSERT/CURRENT/END/SEL_FIRST/SEL_LAST 的用法
# -*- coding: cp936 -*-
from Tkinter import *
```



```
root = Tk()
t = Text(root)
#向 Text 中添加10行文本
for i in range(1, 10):
    t.insert(1.0, '0123456789\n')
#定义各个 Button 的回调函数，这些函数使用了内置的
mark:INSERT/CURRENT/END/SEL_FIRST/SEL_LAST
def insertText():
    t.insert(INSERT, 'jcodeer')
def currentText():
    t.insert(CURRENT, 'jcodeer')
def endText():
    t.insert(END, 'jcodeer')
def selFirstText():
    t.insert(SEL_FIRST, 'jcodeer')
def selLastText():
    t.insert(SEL_LAST, 'jcodeer')
#INSERT
Button(root,
        text = 'insert jcodeer at INSERT',
        command = insertText
        ).pack(fill = X)
#CURRENT
Button(root,
        text = 'insert jcodeer at CURRENT',
        command = insertText
        ).pack(fill = X)
#END
Button(root,
        text = 'insert jcodeer at END',
        command = endText
        ).pack(fill = X)
#SEL_FIRST
Button(root,
        text = 'insert jcodeer at SEL_FIRST',
        command = selFirstText
        ).pack(fill = X)
#SEL_LAST
Button(root,
        text = 'insert jcodeer at SEL_LAST',
        command = selLastText
        ).pack(fill = X)

t.pack()
```

```

root.mainloop()
#几个内置的 mark:
#INSERT:光标的插入点
#CURRENT:鼠标的当前位置所对应的字符位置
#END:这个 Text buffer 的最后一个字符
#SEL_FIRST:选中文本域的第一个字符, 如果没有选中区域则会引发异常
#SEL_LAST: 选中文本域的最后一个字符, 如果没有选中区域则会引发 异常

''' 5. 使用表达式来增强 mark'''
#表达式(expression)可以个性任何的 Indexes, 如下:
'''
+ count chars :前移 count 字符
- count chars :后移 count 字符
+ count lines :前移 count 行
- count lines :后移 count 行
linestart:移动到行的开始
linesend:移动到行的结束
wordstart:移动到字的开始
wordend:移动到字的结束
'''

# 演示修饰符表达式的使用方法, 如何与当前可用的 indexes 一起使用
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
t = Text()
# 向第一行, 第一列添加文本0123456789
for i in range(1,10):
    t.insert(1.0, '0123456789\n')
a = 'test_mark'
def forwardChars():
    # 直接连接字符串
    # t.mark_set(a, CURRENT + '+ 5 chars')
    t.mark_set(a, CURRENT + '+5c')
def backwardChars():
    # t.mark_set(a, CURRENT + '- 5 chars')
    t.mark_set(a, CURRENT + '-5c')
def forwardLines():
    # t.mark_set(a, CURRENT + '+ 5 lines')
    t.mark_set(a, CURRENT + '+5l')
def backwardLines():
    # t.mark_set(a, CURRENT + '- 5 lines')
    t.mark_set(a, CURRENT + '-5l')
def lineStart():
    # 注意 linestart 前面的那个空格不可省略

```

```

    t.mark_set(a, CURRENT + ' linestyle')
def lineEnd():
    # 注意 lineend 前面的那个空格不可省略
    t.mark_set(a, CURRENT + ' lineend')
def wordStart():
    # 移动到当前字的开始。
    t.mark_set(a, CURRENT + ' wordstart')
def wordend():
    # 移动到当前字的结束
    t.mark_set(a, CURRENT + ' wordend')
# mark:test_mark 默认值为 CURRENT
t.mark_set(a, CURRENT)
Button(root, text = 'forward 5 chars', command = forwardChars).pack(fill = X)
Button(root, text = 'backward 5 chars', command = backwardChars).pack(fill = X)
Button(root, text = 'forward 5 lines', command = forwardLines).pack(fill = X)
Button(root, text = 'backward 5 lines', command = backwardLines).pack(fill = X)
Button(root, text = 'line start', command = lineStart).pack(fill = X)
Button(root, text = 'line end', command = lineEnd).pack(fill = X)
Button(root, text = 'word start', command = lineEnd).pack(fill = X)
Button(root, text = 'word end', command = lineEnd).pack(fill = X)
# 测试三个位置的不同, CURRENT 可以得知是当前光标的位置; mark 就表示 mark 的位置
# 了, INSERT 好像一植都在1.0处没有改变。
def insertText():
    t.insert(INSERT, 'insert')
def currentText():
    t.insert(CURRENT, 'current')
def markText():
    t.insert(a, 'mark')
Button(root, text = 'insert jcodeer.cublog.cn', command = insertText).pack(fill = X)
Button(root, text = 'current jcodeer.cublog.cn', command = currentText).pack(fill = X)
Button(root, text = 'mark jcodeer.cublog.cn', command = markText).pack(fill = X)
t.pack()
root.mainloop()

```

## Text (2)

```

''' Tkinter 教程之 Text 篇 (2) '''
''' 6. 使用 tag 来指定文本的属性 '''
# 创建一个指定背景颜色的 TAG
# -*- coding: cp936 -*-

```

```

from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG, 其前景色为红色
t.tag_config('a', foreground = 'red')
# 使用 TAG 'a' 来指定文本属性
t.insert(1.0, '0123456789', 'a')
t.pack()
root.mainloop()
#结果是文本颜色改变为红色了
''' 7. 同时使用两个文本指定同一个属性'''
#没有特别设置的话, 最后创建的那个会覆盖掉其它所有的设置
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG, 其前景色为红色
t.tag_config('a', foreground = 'red')
t.tag_config('b', foreground = 'blue')
# 使用 TAG 'a' 来指定文本属性
t.insert(1.0, '0123456789', ('b', 'a'))
t.pack()
root.mainloop()
# 结果是文本的颜色不是按照 insert 给定的顺序来设置, 而是按照 tag 的创建顺序来设置的。
''' 8. 控制 tag 的级别'''
#使用 tag_lower/tag_raise 来降低或提高 tag 的级别
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG, 其前景色为红色
t.tag_config('a', foreground = 'red')
t.tag_config('b', foreground = 'blue')
# 使用 tag_lower 来降低 b 的级别
t.tag_lower('b')
# 使用 TAG 'a' 来指定文本属性
t.insert(1.0, '0123456789', ('b', 'a'))
t.pack()
root.mainloop()
# 结果: 文本内容颜色变为了红色, 蓝色的作用级别小于红色了, 即使是先创建了红色。
''' 9. 对文本块添加 tag'''
# tag_add 方法的使用
# -*- coding: cp936 -*-

```

```

from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG, 其前景色为蓝色
t.tag_config('b', foreground = 'blue')
# 使用 tag_lower 来控制 tag 的级别
t.tag_lower('b')
# 使用 TAG 'a' 来指定文本属性
for i in range(10):
    t.insert(1.0, '0123456789\n')
t.tag_add('b', '2.5', '2.end')
t.pack()
root.mainloop()
# 先向 Text 中添加了10行文本, 创建一 tag, 将第2行第6列至第二行行尾使用使用此 tag
''' 10. 使用自定义 mark 对文本块添加 tag '''
# -*- coding: cp936 -*-
# tag_add 方法的使用
from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG, 其前景色为蓝色
t.tag_config('b', foreground = 'blue')
# 使用 tag_lower 来控制 tag 的级别
t.tag_lower('b')
# 使用 TAG 'a' 来指定文本属性
for i in range(10):
    t.insert(1.0, '0123456789\n')
# 自定义两个 mark, 并使用它们来指定添加 tag 的文本块
t.mark_set('ab', '3.1')
t.mark_set('cd', END)
t.tag_add('b', 'ab', 'cd')

t.pack()
root.mainloop()
# 先向 Text 中添加了10行文本, 创建两个 mark('ab' 和 'cd'), 将使用这两个 tag 指定文本
的文本块使用此 tag
''' 11. 使用 indexes 获得 Text 中的内容 '''
# -*- coding: cp936 -*-
# 分别使用内置的 indexes 和自定义 mark 来获取文本
# get 方法的使用
from Tkinter import *
root = Tk()
t = Text(root)
for i in range(10):

```

```

    t.insert(1.0, '0123456789\n')
# 获得1.0-2.3的文本
print t.get('1.0', '2.3')
# 自定义两个 mark，并使用它们来获得文本块
t.mark_set('ab', '3.1')
t.mark_set('cd', END)
print t.get('ab', 'cd')
t.pack()
root.mainloop()
''' 12. 测试 delete 对 tag 的影响'''
# -*- coding: cp936 -*-
# delete 方法不会对 tag 造成影响，也就是说删除文本与 tag 没有任何关系
from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG，其前景色为蓝色
t.tag_config('b', foreground = 'blue')
for i in range(10):
    t.insert(1.0, '0123456789\n')
# 自定义两个 mark，并使用它们来指定添加 tag 的文本块
t.mark_set('ab', '3.1')
t.mark_set('cd', END)
t.tag_add('b', 'ab', 'cd')
# 删除(1.0 - 4.0)的文本
t.delete('1.0', '4.0')
t.pack()
root.mainloop()
# (1.0-4.0)的文本全部初始删除了，剩余的文本全部以蓝色显示，即还保留 tag 的属性
''' 13. 使用 tag_delete 对文本属性的影响'''
# -*- coding: cp936 -*-
# 使用 tag_delete 方法操作 tag
from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG，其前景色为蓝色
t.tag_config('b', foreground = 'blue')
for i in range(10):
    t.insert(1.0, '0123456789\n')
# 自定义两个 mark，并使用它们来指定添加 tag 的文本块
t.mark_set('ab', '3.1')
t.mark_set('cd', END)
t.tag_add('b', 'ab', 'cd')
# 删除 tag 'b'，注意这个操作是在 tag_add 之后进行的。
t.tag_delete('b')

```

```
t.pack()
root.mainloop()
# 结果所有的文本没有了 tag('b') 属性, 即 tag_delete 会清除所有与此 tag 相关的属性,
# 不论是之前还是之后
```

## Text (3)

```
''' Tkinter 教程之 Text 篇 (3) '''
''' 14. 自定义 tag 的两个内置属性 '''
# tag.first: tag 之前插入文本, 此文本不包含在这个 tag 中
# tag.last: tag 之后插入文本, 此文本包含在这个 tag 中
# -*- coding: cp936 -*-
# 使用 tag 的内置属性来插入文本
from Tkinter import *
root = Tk()
t = Text(root)
# 创建一个 TAG, 其前景色为蓝色
t.tag_config('b', foreground = 'blue')
for i in range(10):
    t.insert(1.0, '0123456789\n')
# 自定义两个 mark, 并使用它们来指定添加 tag 的文本块
t.mark_set('ab', '3.1')
t.mark_set('cd', END)
t.tag_add('b', 'ab', 'cd')
# 删除 tag 'b', 注意这个操作是在 tag_add 之后进行的。
# 在 tag('b') 之前插入 'first'
t.insert('b.first', 'first')
# 在 tag('b') 之后插入 'last'
t.insert('b.last', 'last')
t.pack()
root.mainloop()
# 注意: first 没有使用 tag('b') 属性, last 使用了 tag('b') 属性
''' 15. 在 Text 中创建按钮 '''
# -*- coding: cp936 -*-
# 使用 window_create 在 Text 内创建一 widget
from Tkinter import *
root = Tk()
t = Text(root)
for i in range(10):
    t.insert(1.0, '0123456789\n')
def printText():
    print 'button in text'
bt = Button(t, text = 'button', command = printText)
```

```

# 在 Text 内创建一个按钮
t.window_create('2.0', window = bt)
# 没有调用 pack()
# bt.pack()
t.pack()
root.mainloop()
# 注意：使用 window_create，而不是使用 insert('2.0', bt); pack() 也不用调用；
# 点击这个按钮，打印出 'button in text'，证明这个按钮是可以正常工作的。
''' 16. 在 Text 中创建一个图像(未实现)'''
# -*- coding: cp936 -*-
# 使用 window_create 在 Text 内创建一 widget
from Tkinter import *
root = Tk()
t = Text(root)
for i in range(10):
    t.insert(1.0, '0123456789\n')
# 分别使用 BitmapImage 和 PhotoImage 进行测试，均没有显示出图像？？
#bm = BitmapImage('gray75')
bm = PhotoImage('c:\\python.gif')
# 在 Text 内创建一个图像
t.image_create('2.0', image = bm)
print t.image_names()
# 打印的图像名称都是正确的
t.pack()
root.mainloop()
# 按照手册中的说明未实现这种效果，原因不知。
''' 17. 绑定 tag 与事件'''
# -*- coding: cp936 -*-
# 使用 tag_bind 方法
from Tkinter import *
root = Tk()
t = Text(root)
for i in range(10):
    t.insert(1.0, '0123456789\n')
# 创建一个 tag
t.tag_config('a', foreground = 'blue', underline = 1)
# Enter 的回调函数
def enterTag(event):
    print 'Enter event'
# 绑定 tag('a') 与事件('<Enter>')
t.tag_bind('a', '<Enter>', enterTag)
t.insert(2.0, 'Enter event\n', 'a')
t.pack()
root.mainloop()

```



```

# 注意：使用 tag_bind 绑定 tag 与事件，当此事件在 tag 上发生时便就会调用这个 tag 的
回调函数
# 因为使用了 Enter 事件，此事件含有一个参数，故将 enterTag 加了一个参数，程序中不
使用此参数
''' 18. 使用 edit_xxx 实现编辑常用功能(未实现)'''
# -*- coding: cp936 -*-
# 使用 edit_xxx 函数实现编辑常用功能
from Tkinter import *
root = Tk()
t = Text(root)
for i in range(10):
    t.insert(1.0, '0123456789\n')
t.pack()
# 定义回调函数
# 撤消回调函数
def undoText():
    t.edit_undo()
# 插入文本函数
def insertText():
    t.insert(1.0, 'insert text')
Button(root, text = 'undo', command = undoText).pack(fill = X)
Button(root, text = 'insert text', command = insertText).pack(fill = X)

root.mainloop()
# 这个 edit_undo 方法也是不起作用，不知为何???

```

## Canvas(1)

```

''' Tkinter 教程之 Canvas 篇(1)'''
# 提供可以用来进行绘图的 Container，支持基本的几何元素，使用 Canvas 进行绘图时，
所有的操作都是通过 Canvas，不是通过它的元素
# 元素的表示可以使用 handle 或 tag。
''' 1. 第一个 Canvas 程序'''
# -*- coding: cp936 -*-
# 指定画布的颜色为白色
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
cv.pack()
root.mainloop()
# 为明显起见，将背景色设置为白色，用以区别 root

```

```
'''2. 创建一个 item'''
```

```
# -*- coding: cp936 -*-  
# 创建一个矩形，指定画布的颜色为白色  
from Tkinter import *  
root = Tk()  
# 创建一个 Canvas，设置其背景色为白色  
cv = Canvas(root, bg = 'white')  
# 创建一个矩形，坐标为(10, 10, 110, 110)  
cv.create_rectangle(10, 10, 110, 110)  
cv.pack()  
root.mainloop()  
# 为明显起见，将背景色设置为白色，用以区别 root
```

```
'''3. 指定 item 的填充色'''
```

```
# -*- coding: cp936 -*-  
# 创建一个矩形，指定画布的背景色为白色  
# 使用属性 fill 设置它的填充颜色  
from Tkinter import *  
root = Tk()  
# 创建一个 Canvas，设置其背景色为白色  
cv = Canvas(root, bg = 'white')  
cv.create_rectangle(10, 10, 110, 110, fill = 'red')  
cv.pack()  
root.mainloop()  
# 指定矩形的填充色为红色
```

```
'''4. 指定 item 的边框颜色'''
```

```
# -*- coding: cp936 -*-  
# 创建一个矩形，指定画布的背景色为白色  
# 使用属性 outline 设置它的边框颜色  
from Tkinter import *  
root = Tk()  
# 创建一个 Canvas，设置其背景色为白色  
cv = Canvas(root, bg = 'white')  
cv.create_rectangle(10, 10, 110, 110, outline = 'red')  
cv.pack()  
root.mainloop()  
# 指定矩形的边框颜色为红色
```

```
'''5. 指定边框的宽度'''
```

```
# -*- coding: cp936 -*-  
# 指定画布的背景色为白色  
# 使用属性 width 指定线的宽度  
from Tkinter import *  
root = Tk()  
# 创建一个 Canvas，设置其背景色为白色  
cv = Canvas(root, bg = 'white')
```

```

cv.create_rectangle(10,10,110,110, outline = 'red', width = 5)
cv.pack()
root.mainloop()
# 指定矩形的边框颜色为红色, 设置线宽为5, 注意与 Canvas 的 width 是不同的。
'''6. 画虚线'''
# -*- coding: cp936 -*-
# 指定画布的背景色为白色
# 使用属性 dash, 这个值只能为奇数
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
cv.create_rectangle(10,10,110,110,
                    outline = 'red',
                    dash = 10,
                    fill = 'green')

cv.pack()
root.mainloop()
# 指定矩形的边框颜色为红色, 画虚线
'''7. 使用画刷填充'''
# -*- coding: cp936 -*-
# 指定画布的背景色为白色
# 使用属性 stipple
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
cv.create_rectangle(10,10,110,110,
                    outline = 'red',
                    stipple = 'gray12',
                    fill = 'green')

cv.pack()
root.mainloop()
# 指定矩形的边框颜色为红色, 自定义画刷
'''8. 修改 item 的坐标'''
# -*- coding: cp936 -*-
# 指定画布的背景色为白色
# 使用 Canvas 的方法来重新设置 item 的坐标
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
rt = cv.create_rectangle(10,10,110,110,
                        outline = 'red',

```

```

        stipple = 'gray12',
        fill = 'green')

cv.pack()
# 重新设置 rt 的坐标 (相当于移动一个 item)
cv.coords(rt, (40, 40, 80, 80))
root.mainloop()
# 动态修改 item 的坐标

```

## Canvas(2)

```

'''Tkinter 教程之 Canvas 篇(2)'''
'''9. 创建 item 的 tags'''
# -*- coding: cp936 -*-
# 使用属性 tags 设置 item 的 tag
# 使用 Canvas 的方法 gettags 获取指定 item 的 tags
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 使用 tags 指定一个 tag('r1')
rt = cv.create_rectangle(10, 10, 110, 110,
                        tags = 'r1'
                        )

cv.pack()

print cv.gettags(rt)
# 使用 tags 属性指定多个 tags, 即重新设置 tags 的属性
cv.itemconfig(rt, tags = ('r2', 'r3', 'r4'))
print cv.gettags(rt)
root.mainloop()
# 动态修改 item 的坐标
'''10. 多个 item 使用同一个 tag'''
# -*- coding: cp936 -*-
# 多个控件使用同一个 tag
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 使用 tags 指定一个 tag('r1')
rt = cv.create_rectangle(10, 10, 110, 110,
                        tags = ('r1', 'r2', 'r3'))

```

```

    )

cv.pack()

cv.create_rectangle(20, 20, 80, 80, tags = 'r3')
print cv.find_withtag('r3')
root.mainloop()
# 动态修改 item 的坐标
# find_withtag 返回所有与 tag 绑定的 item。
''' 11. 通过 tag 来访问 item '''
# -*- coding: cp936 -*-
# 得到了 tag 值也就得到了这个 item，可以对这个 item 进行相关的设置。
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 使用 tags 指定一个 tag('r1')
rt = cv.create_rectangle(10, 10, 110, 110,
                        tags = ('r1', 'r2', 'r3'))

cv.pack()

cv.create_rectangle(20, 20, 80, 80, tags = 'r3')
# 将所有与 tag('r3') 绑定的 item 边框颜色设置为蓝色
for item in cv.find_withtag('r3'):
    cv.itemconfig(item, outline = 'blue')
root.mainloop()
# 动态修改与 tag('r3') 绑定的 item 边框颜色
''' 13. 向其它 item 添加 tag '''
# -*- coding: cp936 -*-
# 使用 addtag_来向上一个或下一个 item 添加 tag
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建三个 rectangle
rt1 = cv.create_rectangle(
    10, 10, 110, 110,
    tags = ('r1', 'r2', 'r3'))
rt2 = cv.create_rectangle(
    20, 20, 80, 80,
    tags = ('s1', 's2', 's3'))
rt3 = cv.create_rectangle(
    30, 30, 70, 70,
    tags = ('y1', 'y2', 'y3'))

```

```

# 向 rt2 的上一个 item 添加 r4
cv.addtag_above('r4', rt2)
# 向 rt2 的下一个 item 添加 r5
cv.addtag_below('r5', rt2)

for item in [rt1, rt2, rt3]:
    print cv.gettags(item)

cv.pack()
root.mainloop()
#Canvas 使用了 stack 的技术, 新创建的 item 总是位于前一个创建的 item 之上, 故调用
above 时, 它会查找 rt2 上面的 item 为 rt3, 故 rt3 中添加了 tag('r4'), 同样 add_below 会
查找下面的 item。
''' 14. 返回其它 item '''
# -*- coding: cp936 -*-
# 使用 find_xxx 查找上一个或下一个 item
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建三个 rectangle
rt1 = cv.create_rectangle(
    10, 10, 110, 110,
    tags = ('r1', 'r2', 'r3'))
rt2 = cv.create_rectangle(
    20, 20, 80, 80,
    tags = ('s1', 's2', 's3'))
rt3 = cv.create_rectangle(
    30, 30, 70, 70,
    tags = ('y1', 'y2', 'y3'))
# 查找 rt2 的上一个 item, 并将其边框颜色设置为红色
cv.itemconfig(cv.find_above(rt2), outline = 'red')
# 查找 rt2 的下一个 item, 并将其边框颜色设置为绿色
cv.itemconfig(cv.find_below(rt2), outline = 'green')

cv.pack()
root.mainloop()
#Canvas 使用了 stack 的技术, 新创建的 item 总是位于前一个创建的 item 之上, 故调用
above 时, 它会查找 rt2 上面的 item 为 rt3, 故 rt3 中边框颜色设置为红色, 同样 add_below
会查找下面的 item。

```

## Canvas(3)

```
'''Tkinter 教程之 Canvas 篇(3)'''
'''16. 移动 item'''

# -*- coding: cp936 -*-
# move 指定 x,y 在偏移量
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建两个同样的 rectangle, 比较移动前后的不同
rtl = cv.create_rectangle(
    10, 10, 110, 110,
    tags = ('r1', 'r2', 'r3'))
cv.create_rectangle(
    10, 10, 110, 110,
    tags = ('r1', 'r2', 'r3'))
# 移动 rtl
cv.move(rtl, 20, -10)
cv.pack()
root.mainloop()
# move 可以指定 x,y 在相对偏移量, 可以为负值
'''17. 删除 item'''

# -*- coding: cp936 -*-
# delete 删除给定的 item
from Tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white')
# 创建两个 rectangle
rtl = cv.create_rectangle(
    10, 10, 110, 110,
    tags = ('r1', 'r2', 'r3'))
r2 = cv.create_rectangle(
    20, 20, 110, 110,
    tags = ('s1', 's2', 's3'))
# 使用 id 删除 rtl
cv.delete(rtl)
# 使用 tag 删除 r2
cv.delete('s1')

cv.pack()
root.mainloop()
# 两种方法删除 item(id/tag)
```

```

'''18. 缩放 item'''
# -*- coding: cp936 -*-
# scale 缩放 item, 计算公式: (coords - offset)*scale + offset
from Tkinter import *
root = Tk()
cv = Canvas(root, bg = 'white')
# 创建两个 rectangle
rtl = cv.create_rectangle(
    10, 10, 110, 110,
    tags = ('r1', 'r2', 'r3'))
# 将 y 坐标放大为原来的2位, x 坐标值不变
cv.scale(rtl, 0, 0, 1, 2)
cv.pack()
root.mainloop()
# scale 的参数为(self, xoffset, yoffset, xscale, yscale)
'''19. 绑定 item 与 event'''
# -*- coding: cp936 -*-
# 使用 tag_bind 来绑定 item 与事件
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建三个 rectangle
rtl = cv.create_rectangle(
    10, 10, 110, 110,
    width = 8,
    tags = ('r1', 'r2', 'r3'))
def printRect(event):
    print 'rectangle'
# 绑定 item 与事件
cv.tag_bind('r1', '<Button-1>', printRect)
cv.pack()
root.mainloop()
# 只有点击到矩形的边框时才会触发事件
'''20. 添加绑定事件'''
# -*- coding: cp936 -*-
# 使用 tag_bind 来绑定 item 与事件, 与参考上测试结果不一致。
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建三个 rectangle
rtl = cv.create_rectangle(
    10, 10, 110, 110,

```



```

width = 8,
tags = ('r1', 'r2', 'r3'))
def printRect(event):
    print 'rectangle'
def printLine(event):
    print 'line'
# 绑定 item 与左键事件
cv.tag_bind('r1', '<Button-1>', printRect)
# 绑定 item 与右键事件
cv.tag_bind('r1', '<Button-3>', printLine)
cv.pack()
root.mainloop()
# 只有点击到矩形的边框时才会触发事件, 不使用 add 参数, 默认就是向这个 item 添加一个处理函数, 它不会替换原来的事件函数, 例子结果: 既响应左键又响应右键
''' 21. 绑定新的 item 与现有的 tags '''
# -*- coding: cp936 -*-
# 使用 tag_bind 来绑定 item 与事件, 测试结果与参考上的说法不一致
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建三个 rectangle
rtl = cv.create_rectangle(
    10, 10, 110, 110,
    width = 8,
    tags = ('r1', 'r2', 'r3'))
def printRect(event):
    print 'rectangle'
def printLine(event):
    print 'line'
# 绑定 item 与左键事件
cv.tag_bind('r1', '<Button-1>', printRect)
# 绑定 item 与右键事件
cv.tag_bind('r1', '<Button-3>', printLine)
# 创建一个 line, 并将其 tags 设置为 'r1'
cv.create_line(10, 200, 100, 200, width = 5, tags = 'r1')
cv.pack()
root.mainloop()
# 将事件与 tag('r1') 绑定后, 创建新的 item 并指定已绑定事件的 tag, 新创建的 item 同样也与事件绑定, 这个与参考上的说法也不一致

```

## Canvas(4)

```
'''Tkinter 教程之 Canvas(4) 篇'''
'''22. 绘制弧形'''
# -*- coding: cp936 -*-
# 创建一个 ARC
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
cv.create_arc((10, 10, 110, 110),)
cv.pack()
root.mainloop()
# 使用默认参数创建一个 ARC，结果为90度的扇形
'''23. 设置弧形的样式'''
# -*- coding: cp936 -*-
# 创建 create_arc.
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
d = {1:PIESLICE, 2:CHORD, 3:ARC}
for i in d:
    cv.create_arc((10, 10 + 60*i, 110, 110 + 60*i), style = d[i])
    print i, d[i],
cv.pack()
root.mainloop()
# 使用三种样式，分别创建了扇形、弓形和弧形
'''24. 设置弧形的角度'''
# -*- coding: cp936 -*-
# 使用 start/extent 指定起始角度与偏移角度
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
d = {1:PIESLICE, 2:CHORD, 3:ARC}
for i in d:
    cv.create_arc(
        (10, 10 + 60*i, 110, 110 + 60*i),
        style = d[i], #指定样式
        start = 30, #指定起始角度
        extent = 30 #指定角度偏移量
    )
```

```

cv.pack()
root.mainloop()
# 使用三种样式, start 指定起始角度; extent 指定角度偏移
''' 25. 绘制位图'''
# -*- coding: cp936 -*-
# 使用 bitmap 创建位图 create_bitmap
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
d = {1:'error', 2:'info', 3:'question', 4:'hourglass'}
for i in d:
    cv.create_bitmap((20*i, 20*i), bitmap = d[i])
cv.pack()
root.mainloop()
# 使用 bitmap 属性来指定位图的名称, 这个函数的第一个参数为一个点 (x, y) 指定位图存放位置的左上位置。
''' 26. 绘制 GIF 图像'''
# -*- coding: cp936 -*-
# 创建 gif 图像 create_image
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
img = PhotoImage(file = 'c:\\python.gif')
cv.create_image((150, 150), image = img)
cv.pack()
root.mainloop()
# 先使用 PhotoImage 创建 GIF 图像, 再将 image 属性来设置为新创建的 img
''' 27. 绘制直线'''
# -*- coding: cp936 -*-
# 创建带箭头的直线 create_line
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
d = [(0, 'none'), (1, 'first'), (2, 'last'), (3, 'both')]
for i in d:
    cv.create_line(
        (10, 10 + i[0]*20, 110, 110 + i[0] * 20), # 设置直线的起始、终点
        arrow = i[1], # 设置直线是否使用箭头
        arrowshape = '40 40 10' # 设置箭头的形状(填充长度, 箭头长度, 箭头宽度)
    )
cv.pack()

```

```

root.mainloop()
# 使用 arrow 属性来控制是否显示箭头
'''28. 直线的 joinstyle 属性'''
# -*- coding: cp936 -*-
# 创建直线, 使用 joinstyle 属性
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
d =
[(0, 'none', 'bevel'), (1, 'first', 'miter'), (2, 'last', 'round'), (3, 'both', 'round')]
for i in d:
    cv.create_line(
        (10, 10 + i[0]*20, 110, 110+ i[0] * 20), # 设置直线的起始、终点
        arrow = i[1], # 设置直线是否使用箭头
        arrowshape = '8 10 3', # 设置箭头的形状(填充长度, 箭头长度, 箭头宽度)
        joinstyle = i[2],
    )
cv.pack()
root.mainloop()
# 将直线的属性 joinstyle 分别设置为 bevel/miter/round, 测试其效果。
'''29. 绘制椭圆'''
# -*- coding: cp936 -*-
# 绘制椭圆, 使用 create_oval 属性
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建一个长200, 宽100的椭圆
cv.create_oval((10, 10, 210, 110), fill = 'red')
cv.pack()
root.mainloop()
# 指定椭圆的长和宽, 圆是长和宽相等的特殊情况。
'''30. 创建多边形'''
# -*- coding: cp936 -*-
# 创建多边形(三角形)
from Tkinter import *
root = Tk()
# 绘制一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建一个直角三角形
cv.create_polygon((10, 10, 10, 200, 200, 200), fill = 'red')
cv.pack()
root.mainloop()

```

```

# 指定三个点的坐标，三个点坐标必须满足三角形的定义。
''' 31. 修饰图形 '''
# -*- coding: cp936 -*-
# 创建多边形 create_ploygon (三角形)
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建一个直角三角形
cv.create_polygon((10, 10, 10, 200, 100, 200),
                  #smooth = True, # 平滑处理，但未找到控制此参数的项
                  splinesteps = 0, # 不明白是控制什么的???
                  )

cv.pack()
root.mainloop()
# smooth/splinesteps 用来修改绘制的图形，不明白这两个参数还有其它什么作用。
''' 32. 绘制文字 '''
# -*- coding: cp936 -*-
# 使用文字 create_text
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建一个文字对象，默认设置为居中对齐
cv.create_text((10, 10), text = 'Hello Text',
               anchor = W
               )

cv.pack()
root.mainloop()
# 使用 anchor 控制文字的位置，使用 justify 控制对齐方式
''' 33. 选中文字 '''
# -*- coding: cp936 -*-
# 使用文字 create_text
from Tkinter import *
root = Tk()
# 创建一个 Canvas，设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建一个文字对象，默认设置为居中对齐
txt = cv.create_text((10, 10), text = 'Hello Text',
                     anchor = W
                     )

# 设置文本的选中起始位置
cv.select_from(txt, 2)
# 设置文本的选中结束位置

```

```

cv.select_to(txt,5)

cv.pack()
root.mainloop()
# 使用 anchor 控制文字的位置, 使用 justify 控制对齐方式
'''34. 创建组件'''
# -*- coding: cp936 -*-
# 使用子组件 create_window
from Tkinter import *
root = Tk()
# 创建一个 Canvas, 设置其背景色为白色
cv = Canvas(root, bg = 'white')
# 创建一个 Button 对象, 默认设置为居中对齐
def printWindow():
    print 'window'
bt = Button(cv, text = 'ClickMe', command = printWindow)
#修改 button 在 canvas 上的对齐方式
cv.create_window((10,10), window = bt, anchor = W)
# 新创建的 line 对象与 button 有重叠
cv.create_line(10,10,20,20)
# 新创建的 line 不在 button 之上, 即没有重叠
cv.create_line(30,30,100,100)
cv.pack()
root.mainloop()
# 使用 anchor 组件在 Canvas 上的位置, 默认情况下为居中对齐, 这样使用后其它的 item
将不能再使用 button 战胜的那块区域

```

## Pack

```

'''Tkinter 教程之 Pack 篇'''
#Pack 为一布局管理器, 可将它视为一个弹性的容器
'''1. 一个空的 widget'''
#不使用 pack
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
# 查看当前 root 下的子组件, 解释器没有报异常, 说明 Pack 已创建, 并可以使用, 此时的
输出为空, 即 root 没有任何子组件。
print root.pack_slaves()
# 向 root 中 pack 一个 Label
Label(root, text = 'pack').pack()
# 再次打印出 root 的子组件, 可以看到已经包含一个组件, 即刚才创建的 Label, 说明 Label
调用 pack() 是将自己加入到了 root 中。

```

```

print root.pack_slaves()
root.mainloop()
# pack_slaves 打印当前组件包拥有的子组件，通过这个函数可以查看各个组件是否有包含
关系。
''' 2. root 与 Pack 的关系'''
# -*- coding: cp936 -*-
# 使用文字 create_text
from Tkinter import *
root = Tk()
# 改变 root 的大小为80x80
root.geometry('80x80+0+0')
print root.pack_slaves()
Label(root, text = 'pack').pack()
print root.pack_slaves()
root.mainloop()
#可以看出 Pack 的结果没有什么变化，它不对 root 产生影响，也就是说 Pack 可以“缩小”
至只包含一个 Label 组件，root 可以自己控件自己的大小。
''' 3. 向 Pack 中添加多个组件'''
# -*- coding: cp936 -*-
# 向 Pack 中添加多个 Label
from Tkinter import *
root = Tk()
# 改变 root 的大小为80x80
root.geometry('80x80+0+0')
print root.pack_slaves()
for i in range(5):
    Label(root, text = 'pack' + str(i)).pack()
print root.pack_slaves()
root.mainloop()
# 使用默认的设置 pack 将向下添加组件，第一个在最上方，然后是依次向下排列。注意
最后一个 Label 的显示不完全，稍后解释原因
''' 4. 固定设置到自由变化'''
# 上例中看到 label4没有显示完全
# -*- coding: cp936 -*-
# 不设置 root 的大小，使用默认
from Tkinter import *
root = Tk()
#去掉下面的这句
#root.geometry('80x80+0+0')
print root.pack_slaves()
for i in range(5):
    Label(root, text = 'pack' + str(i)).pack()
print root.pack_slaves()
root.mainloop()

```

#使用默认的设置 pack 将向下添加组件，第一个在最上方，然后是依次向下排列。这样的话最后一个已经显示出来的，这就是为什么称 Pack 为弹性的容器的原因了，虽然有这个特性，但它并不是总是能够按照我们的意思进行布局，我们可以强制设置容器的大小，以覆盖 Pack 的默认设置。Pack 的优先级低。

''' 5. fill 如何控制子组件的布局'''

```
# -*- coding: cp936 -*-
```

```
# 不设置 root 的大小，使用默认
```

```
from Tkinter import *
```

```
root = Tk()
```

```
# 改变 root 的大小为80x80
```

```
root.geometry('80x80+0+0')
```

```
print root.pack_slaves()
```

```
# 创建三个 Label 分别使用不同的 fill 属性
```

```
Label(root, text = 'pack1', bg = 'red').pack(fill = Y)
```

```
Label(root, text = 'pack2', bg = 'blue').pack(fill = BOTH)
```

```
Label(root, text = 'pack3', bg = 'green').pack(fill = X)
```

```
print root.pack_slaves()
```

```
root.mainloop()
```

#第一个只保证在 Y 方向填充，第二个保证在 XY 两个方向上填充，第三个不使用填充属性，注意 Pack 只会吝啬地给出可以容纳这三个组件的最小区域，它不允许使用剩余的空间了，故下方留有“空白”。

''' 6. expand 如何控制组件的布局'''

```
# -*- coding: cp936 -*-
```

```
# 这个属性指定如何使用额外的空间，即上例中留下来的“空白”
```

```
from Tkinter import *
```

```
root = Tk()
```

```
# 改变 root 的大小为80x80
```

```
root.geometry('80x80+0+0')
```

```
print root.pack_slaves()
```

```
# 创建三个 Label 分别使用不同的 fill 属性
```

```
Label(root, text = 'pack1', bg = 'red').pack(fill = Y, expand = 1)
```

```
Label(root, text = 'pack2', bg = 'blue').pack(fill = BOTH, expand = 1)
```

```
Label(root, text = 'pack3', bg = 'green').pack(fill = X, expand = 0)
```

```
print root.pack_slaves()
```

```
root.mainloop()
```

# 第一个只保证在 Y 方向填充，第二个保证在 XY 两个方向上填充，第三个不使用填充属性，这个例子中第一个 Label 和第二个 Label 使用了 expand = 1 属性，而第三个使用 expand = 0 属性，改变 root 的大小，可以看到 Label1 和 Label2 是随着 root 的大小变化而变化（严格地它的可用空间在变化），第三个只中使用 fill 进行 X 方向上的填充，不使用额外的空间。

''' 7. 改变组件的排放位置'''

```
# 使用 side 属性改变放置位置
```

```
# -*- coding: cp936 -*-
```

```
from Tkinter import *
```



```

root = Tk()
# 改变 root 的大小为80x80
root.geometry('80x80+0+0')
print root.pack_slaves()
# 创建三个 Label 分别使用不同的 fill 属性, 改为水平放置
# 将第一个 Label 居左放置
Label(root, text = 'pack1', bg = 'red').pack(fill = Y, expand = 1, side = LEFT)
# 将第二个 Label 居右放置
Label(root, text = 'pack2', bg = 'blue').pack(fill = BOTH, expand = 1, side = RIGHT)
# 将第三个 Label 居左放置, 靠 Label 放置, 注意它不会放到 Label1的左边
Label(root, text = 'pack3', bg = 'green').pack(fill = X, expand = 0, side = LEFT)
print root.pack_slaves()
root.mainloop()
# 第一个只保证在 Y 方向填充, 第二个保证在 XY 两个方向上填充, 第三个不使用填充属性,
这个例子中第一个 Label 和第二个 Label 使用了 expand = 1 属性, 而第三个使用 expand =
0 属性, 改变 root 的大小, 可以看到 Label1 和 Label2 是随着 root 的大小变化而变化 (严
格地它的可用空间在变化), 第三个只中使用 fill 进行 X 方向上的填充, 不使用额外的空
间。

```

#### ''' 8. 设置组件之间的间隙大小'''

```

# ipadx 设置内部间隙
# padx 设置外部间隙
# -*- coding: cp936 -*-
# 不设置 root 的大小, 使用默认
from Tkinter import *
root = Tk()
# 改变 root 的大小为80x80
# root.geometry('80x80+0+0')
print root.pack_slaves()
# 创建三个 Label 分别使用不同的 fill 属性, 改为水平放置
# 将第一个 LabelFrame 居左放置
L1 = LabelFrame(root, text = 'pack1', bg = 'red')
# 设置 ipadx 属性为20
L1.pack(side = LEFT, ipadx = 20)
Label(L1,
      text = 'inside',
      bg = 'blue'
      ).pack(expand = 1, side = LEFT)
L2 = Label(root,
           text = 'pack2',
           bg = 'blue'
           ).pack(fill = BOTH, expand = 1, side = LEFT, padx = 10)
L3 = Label(root,
           text = 'pack3',
           bg = 'green'

```

```

        ).pack(fill = X, expand = 0, side = LEFT, pady = 10)
print root.pack_slaves()
root.mainloop()
#为了演示 ipadx/padx, 创建了一个 LabelFrame 设置它的 ipadx 为20, 即内部间隔值为20,
它的子组件若使用则会留出20个单位; Label2和 Label3分别设置 x 和 y 方向上的外部间隔
值, 所有与之排列的组件会与之保留10个单位值的距离

```

## Place

```

'''Tkinter 教程之 Place 篇'''
'''1. 使用绝对坐标将组件放到指定的位置'''
# -*- coding: cp936 -*-
# 不设置 root 的大小, 使用默认
from Tkinter import *
root = Tk()
lb = Label(root, text = 'hello Place')
# lb.place(relx = 1, rely = 0.5, anchor = CENTER)
# 使用绝对坐标将 Label 放置到(0,0)位置上
lb.place(x = 0, y = 0, anchor = NW)
root.mainloop()
# x, y 指定组件放置的绝对位置
'''2. 使用相对坐标放置组件位置'''
# -*- coding: cp936 -*-
# 不设置 root 的大小, 使用默认
from Tkinter import *
root = Tk()
lb = Label(root, text = 'hello Place')
# lb.place(relx = 1, rely = 0.5, anchor = CENTER)
# 使用相对坐标(0.5,0.5)将 Label 放置到(0.5*sx, 0.5.sy)位置上
lb.place(relx = 0.5, rely = 0.5, anchor = CENTER)
root.mainloop()
# relx, rely 指定组件放置的绝对位置, 范围为(0-1.0)
'''3. 使用 place 同时指定多个组件'''
# -*- coding: cp936 -*-
# 不设置 root 的大小, 使用默认
from Tkinter import *
root = Tk()
root.geometry('800x600')
lb = Label(root, text = 'hello Place')

```

```

# lb.place(relx = 1, rely = 0.5, anchor = CENTER)
# 使用相对坐标(0.5, 0.5)将 Label 放置到(0.5*sx, 0.5. sy)位置上
v = IntVar()
for i in range(5):
    Radiobutton(
        root,
        text = 'Radio' + str(i),
        variable = v,
        value = i
    ).place(x = 80* i, anchor = NW)
root.mainloop()
# 使用 place 来指定各个 Radiobutton 的位置
''' 4. 同时使用相对和绝对坐标'''
# 同时设置 relx, rely 和 x, y 的值
# -*- coding: cp936 -*-
# 不设置 root 的大小, 使用默认
from Tkinter import *
root = Tk()
root.geometry('800x600')
lb1 = Label(root, text = 'hello Place', fg = 'green')
lb2 = Label(root, text = 'hello Place', fg = 'red')
# 先设置相对坐标为(0.5, 0.5), 再使用(-200, -200)将坐标作偏移(-200, -200)
lb1.place(relx = 0.5, rely = 0.5, anchor = CENTER, x = -200, y = -200)
# 先设置相对坐标为(0.5, 0.5), 再使用(-300, -300)将坐标作偏移(-300, -300)
lb2.place(relx = 0.5, rely = 0.5, anchor = CENTER, x = -300, y = -300)
root.mainloop()
# 同时使用相对和绝对坐标时, 相对坐标优先操作, 然后是在这个相对坐标的基础上进行
偏移
''' 5. 使用 in 来指定放置的容器'''
# -*- coding: cp936 -*-
# 使用 in 属性来指定放置到的容器是那一个
from Tkinter import *
root = Tk()
root.geometry('800x600')
lb1 = Label(root, text = 'hello Place', fg = 'green')
bt1 = Button(root, text = 'hello Place', fg = 'red')
# 创建一个 Label
lb1.place(relx = 0.5, rely = 0.5, anchor = CENTER)

# 在 root 同创建一个 Button, 目的是与 bt1相比较
bt2 = Button(root, text = 'button in root', fg = 'yellow')
bt2.place(anchor = W)
# 在 Label 中创建一个 Button
bt1.place(in_ = lb1, anchor = W)

```

```

root.mainloop()
# 注意 bt2放置的位置是在 root 的 (0,0) 处, 而 button1放置的位置是在 lb1的 (0,0) 处, 原因是由于 bt1使用了 in 来指定放置的窗口为 lb1
''' 6. 深入 in 用法'''
# -*- coding: cp936 -*-
# 使用 in 属性来指定放置到的容器是那一个, 仅能是其 master
from Tkinter import *
root = Tk()
# root.geometry('800x600')
# 创建两个 Frame 用作容器
fm1 = Frame(root, bg = 'red', width = 40, height = 40)
fm2 = Frame(root, bg = 'blue', width = 40, height = 40)
# 再在 fm1中创建一个 fm3
fm3 = Frame(fm1, bg = 'yellow', width = 20, height = 20)

# 创建一个 Label, 它的 master 为 fm1
lb1 = Label(fm1, text = 'hello Place', fg = 'green')
lb1.place(in_ = fm1, relx = 0.5, rely = 0.5, anchor = CENTER)
# 创建一个 Button, 它的 master 为 fm1
bt1 = Button(fm1, text = 'hello Place', fg = 'red')

# 将 bt1放置到 fm2中, 程序报错
# 去掉下面这条语句就可以使用了, 可以看到 lb1已经正确的放置到 fm1的中心位置了
# bt1.place(in_ = fm2, anchor = W)

# 将上面的语句改为下面, 即将 bt1放置到其 fm1的子组件 fm3中, 这样也是可以的
bt1.place(in_ = fm3, anchor = W)

fm1.pack()
fm2.pack()
fm3.pack()
root.mainloop()
# in 不是可以随意指定放置的组件的, 如果使用 in 这个参数这个组件必需满足: 是其父容器或父容器的子组件
''' 7. 事件与 Place 结合使用'''
# -*- coding: cp936 -*-
# 最后使用两个 place 方法来动态改变两个 Frame 的大小。
from Tkinter import *
root = Tk()
split = 0.5
fm1 = Frame(root, bg = 'red')
fm2 = Frame(root, bg = 'blue')
# 单击 fm1时增大它的占有区域0.1
def incFm1(event):

```

```

global split
if split < 1:
    split += 0.1
    fm1.place(rely = 0, relheight = split, relwidth = 1)
    fm2.place(rely = split, relheight = 1 - split, relwidth = 1)
# 单击 fm2时增大它的占有区域0.1
def incFm2(event):
    global split
    if split > 0:
        split -= 0.1
        fm1.place(rely = 0, relheight = split, relwidth = 1)
        fm2.place(rely = split, relheight = 1 - split, relwidth = 1)

# 这两语句要使用，不然开始看不到两个 frame，也就没法点击它们了
fm1.place(rely = 0, relheight = split, relwidth = 1)
fm2.place(rely = split, relheight = 1 - split, relwidth = 1)
# 绑定单击事件
fm1.bind('<Button-1>', incFm1)
fm2.bind('<Button-1>', incFm2)

root.mainloop()
# 为 SplitWindow 的原型了，再改动一下就可以实现一个 SplitWindow 了。

```

## Grid

```

'''Tkinter 教程之 Grid 篇'''
# Tkinter 参考中最推荐使用的一个布局器。实现机制是将 Widget 逻辑上分割成表格，在
# 指定的位置放置想要的 Widget 就可以了。
'''1. 第一个 Grid 例子'''
# -*- coding: cp936 -*-
# 使用 grid 来布局组件
from Tkinter import *
root = Tk()
# 创建两个 Label
lb1 = Label(root, text = 'Hello')
lb2 = Label(root, text = 'Grid')

lb1.grid()
lb2.grid()

root.mainloop()
# grid 有两个最为重要的参数，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, 则使用第一列。

'''2. 使用 row 和 column 来指定位置'''

```
# -*- coding: cp936 -*-
# 使用 grid 来布局组件
from Tkinter import *
root = Tk()
# 创建两个 Label
lb1 = Label(root, text = 'Hello')
lb2 = Label(root, text = 'Grid')

lb1.grid()
# 指定 lb2为第一行（使用索引0开始），第二列（使用索引0开始）
lb2.grid(row = 0, column = 1)
```

```
root.mainloop()
# grid 有两个最为重要的参数, 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
第一列。注意这里使用 grid 时不需要创建, 直接使用行列就可以。
```

'''3. 为其它组件预定位置'''

```
# 可以使用 row/column 来指定组件的放置位置, 并预先留出空间, 以备其它需要。
# -*- coding: cp936 -*-
# 使用 grid 来布局组件
from Tkinter import *
root = Tk()
# 创建两个 Label
Label(root, text = 'Hello').pack()
# 在第一行, 第10列放置 lb2
Label(root, text = 'Grid').grid(row = 0, column = 10)
# Label(root, text = '3').grid(row = 0, column = 5)
root.mainloop()
# 这个例子中将 lb2放置到第1行, 第11列位置上, 但运行结果与上一例从效果上看不出太
大的区别。原因是: 如果这个位置没有组件的话, 它是看不见的。
```

'''4. 将组件放置到预定位置上去'''

```
# -*- coding: cp936 -*-
# 使用 grid 来布局组件
from Tkinter import *
root = Tk()
# 创建两个 Label
Label(root, text = '1').grid()
# 在第1行, 第11列放置 lb2
Label(root, text = '2').grid(row = 0, column = 10)
Label(root, text = '3').grid(row = 0, column = 5)
root.mainloop()
```

# 可以看到Label('3')是位置Label('1')和Label('2')之间了, 即Label('2')是在11列, Label('3')位于第3列

'''5. 将两个或多个组件同一个位置'''

# -\*- coding: cp936 -\*-

# 多个组件同时 grid 到同一个表格位置

from Tkinter import \*

root = Tk()

# 创建两个Label

lb1 = Label(root, text = '1')

lb2 = Label(root, text = '2')

# 将 lb1和 lb2均 grid 到(0,0)位置

lb1.grid(row = 0, column = 0)

lb2.grid(row = 0, column = 0)

def forgetLabel():

# grid\_slaves 返回 grid 中(0,0)位置的所有组件

# grid\_forget 将这个组件从 grid 中移除(并未删除, 可以使用 grid 再将它显示出来)

print root.grid\_slaves(0,0)[0].grid\_forget()

# 我测试时 grid\_slaves 返回的第一个值为 lb2, 最后 grid 的那一个

Button(root, text = 'forget last', command = forgetLabel).grid(row = 1)

root.mainloop()

# 这段代码是用来证明, 多个组件同时放置到同一个位置上会产生覆盖的问题。对于 grid\_slaves 返回的组件 list 如何排序, 我没有去查想着资料, 在这个例子中使用索引0, 返回的正好是 lb2, 然后再使用 grid\_forget 将这个删除从 grid 中移除, 可以看到 lb1显示出来了。

'''6. 改变列(行)的属性值'''

# -\*- coding: cp936 -\*-

# 设置 column 的属性(columnconfigure)

from Tkinter import \*

root = Tk()

# 创建两个Label

lb1 = Label(root, text = '1', bg = 'red')

lb2 = Label(root, text = '2', bg = 'blue')

# 将 lb1和 lb2分别放置到第1行的1,2列位置上

lb1.grid(row = 0, column = 0)

lb2.grid(row = 0, column = 1)

# 指定列的最小宽度为100

root.columnconfigure(0, minsize = 100)

root.mainloop()

```

# 1与2的距离变的远一些了。
# 但如果这个位置没有组件存在的话这个值是不起作用的。
# 设置列或行 (rowconfigure) 的属性时使用父容器的方法, 不是自己调用。
''' 7. 组件使用多列 (多行) '''
# -*- coding: cp936 -*-
# 使用多行 (多列)
from Tkinter import *
root = Tk()
# 创建如下布局 (一个字符占用一个 grid 位置)
# A E
# B C
# D
# A 占用 (0, 0) (0, 1), B 占用 (1, 0), C 占用 (1, 1), D 占用 (2, 0), E 占用 (0, 2)
# 创建5个 Label, 分别以背景色区别
lbA = Label(root, text = 'A', bg = 'red')
lbB = Label(root, text = 'B', bg = 'blue')
lbC = Label(root, text = 'C', bg = 'red')
lbD = Label(root, text = 'D', bg = 'blue')
lbE = Label(root, text = 'E', bg = 'blue')
# 以下为布局参数设置
lbA.grid(row = 0, column = 0, columnspan = 2)
lbB.grid(row = 1, column = 0)
lbC.grid(row = 1, column = 1)
lbD.grid(row = 2)
lbE.grid(row = 0, column = 2)

root.mainloop()
# A 与 B、D 的区别, 它左边已改变, 由于使用了两个表格;
# C 与 E 的区别: C 的右边与 E 的左边对齐, 也就是说 E 被放置到第2列的下一个位置了,
原因由于 A 已使用了第2列。
''' 8. 设置表格中组件的对齐属性 '''
# -*- coding: cp936 -*-
# 使用 sticky 设置对齐方式
from Tkinter import *
root = Tk()
# 创建两个 Label
Label(root, text = 'hello sticky').grid()
Label(root, text = 'Tkinter').grid()
# 创建两个 Label, 并指定 sticky 属性
Label(root, text = 'hello sticky').grid(sticky = W)
Label(root, text = 'Tkinter').grid(sticky = W)

root.mainloop()
# 默认属性下, 组件的对齐方式为居中, 设置 sticky 属性可以控制对齐方式, 可用的值

```



## Font

```
''' Tkinter 教程之 Font 篇 '''
# Tkinter 中其它常用的一些功能
''' 1. 字体使用 '''
# -*- coding: cp936 -*-
# 改变组件的显示字体
from Tkinter import *
root = Tk()
# 创建一个 Label
for ft in ('Arial', ('Courier New',), ('Comic Sans MS',), 'Fixdsys', ('MS Sans
Serif',), ('MS Serif',), 'Symbol', 'System', ('Times New Roman',), 'Verdana'):
    Label(root, text = 'hello sticky', font = ft ).grid()

root.mainloop()
# 在 Windows 上测试字体显示，注意字体中包含有空格的字体名称必须指定为 tuple 类型。
''' 2. 使用系统已有的字体 '''
# -*- coding: cp936 -*-
# Font 来创建字体
from Tkinter import *
# 引入字体模块
import tkFont
root = Tk()
# 创建一个 Label
# 指定字体名称、大小、样式
ft = tkFont.Font(family = 'Fixdsys', size = 20, weight = tkFont.BOLD)
Label(root, text = 'hello sticky', font = ft ).grid()

root.mainloop()
# 使用 tkFont.Font 来创建字体。
''' 3. 字体创建属性优先级 '''
# -*- coding: cp936 -*-
# 使用系统已有的字体显示
from Tkinter import *
import tkFont
root = Tk()
# 创建一个 Label
# 指定字体名称、大小、样式
# 名称是系统可使用的字体
ft1 = tkFont.Font(family = 'Fixdsys', size = 20, weight = tkFont.BOLD)
Label(root, text = 'hello sticky', font = ft1 ).grid()
```

```

ft2 = tkFont.Font(font = ('Fixdsys', '10', tkFont.NORMAL), size = 40)
Label(root, text = 'hello sticky', font = ft2).grid()

root.mainloop()
# 创建字体有 font 等其它属性,
# 如果 font 指定了, 有几个参数将不再起作用, 如:
family, size, weight, slant, underline, overstrike
# 例子中演示的结果是 ft2 中字体大小为10, 而不是40
''' 4. 得到字体的属性值'''
# -*- coding: cp936 -*-
# 测试 measure 和 metrics 属性
from Tkinter import *
import tkFont
root = Tk()
# 创建一个 Label
ft1 = tkFont.Font(family = 'Fixdsys', size = 20, weight = tkFont.BOLD)
Label(root, text = 'hello font', font = ft1).grid()

ft2 = tkFont.Font(font = ('Fixdsys', '10', tkFont.NORMAL), size = 40)
Label(root, text = 'hello font', font = ft2).grid()

# 得到字体的宽度
print ft1.measure('hello font')
print ft2.measure('hello font')

# 打印两个字体的属性
for metric in ('ascent', 'descent', 'linespace', 'fixed'):
    print ft1.metrics(metric)
    print ft2.metrics(metric)
root.mainloop()
# 使用这两个方法得到已创建字体的相关属性值
''' 5. 使用系统指定的字体'''
# -*- coding: cp936 -*-
# 使用系统字体: 以下测试是 Windows 上的系统指定字体
from Tkinter import *
import tkFont
root = Tk()
for ft1 in ('ansi', 'ansifixed', 'device', 'oemfixed', 'system', 'systemfixed'):
    Label(root, text = 'hello font', font = ft1).grid()

root.mainloop()
# X Window 上的系统指定字体: fixed, 6x10等
''' 6. 使用 X Font Descriptor'''

```

```

# -*- coding: cp936 -*-
# 使用 X Font Descriptor
from Tkinter import *
import tkFont
root = Tk()
for ft in ('Times', 'Helvetica', 'Courier', 'Symbol',):
    Label(root, text = 'hello font', font = ('-*-s-*-*--*240-*')(ft)).grid()

root.mainloop()
# X Font Descriptor 格式: *-family-weight-slant-*--size-*-*-*--charset
# 这个例子是在 Windows 下测试，没有在 Linux 测试。

```

## tkCommonDialog

```

'''Tkinter 教程之 tkCommonDialog 篇'''
'''1. 使用用模态对话框 SimpleDialog'''
from Tkinter import *
# 引入 SimpleDialog 模态对话框
from SimpleDialog import *

root = Tk()
# 创建一个 SimpleDialog
# buttons:显示的按钮
# default:默认选中的按钮
dlg = SimpleDialog(root,
                    text = 'hello SimpleDialog',
                    buttons = ['Yes', 'No', 'cancel'],
                    default = 0,
                    )

# 执行对话框
print dlg.go()
root.mainloop()
# 返回值点击的按钮在 buttons 中的索引值
'''2. 使用 tkSimpleDialog 模块'''
from Tkinter import *
# 引入 SimpleDialog 模态对话框
from tkSimpleDialog import *

root = Tk()
# 输入一个整数，
# initialvalue 指定一个初始值
# prompt 提示信息
# title 提示框标题

```

```

print askinteger(title = 'prompt', prompt = 'input a integer:', initialvalue = 100)
# 输入一浮点数
# minvalue 指定最小值
# maxvalue 指定最大值, 如果不在二者指定范围内则要求重新输入
print askfloat(title = 'float', prompt = 'input a float', minvalue = 0, maxvalue =
11)
# 输入一字符串
print askstring(title = 'string', prompt = 'input a string')
root.mainloop()
# 返回值为各自输入的值。
''' 2. 打开文件对话框'''
from Tkinter import *
from FileDialog import *

root = Tk()
# 指定 master 就可以了。
# title 属性用来指定标题
fd = LoadFileDialog(root)
# go 方法的返回值即为选中的文本路径, 如果选择取返回值则为 None
print fd.go()
root.mainloop()
# 返回选中的文件名称
''' 3. 保存文件对话框'''
# 与 LoadFileDialog 正好操作相反, 这个类是用来保存文件。
# 各个 参数的意义都一样, 只是 ok 的返回值为保存的文件名称; 如果取消则为 None
from Tkinter import *
from FileDialog import *

root = Tk()
# 指定 master 就可以了。
# title 属性用来指定标题
fd = SaveFileDialog(root)
# go 方法的返回值即为选中的文本路径, 如果选择取返回值则为 None
print fd.go()
root.mainloop()
# 返回选中的文件名称

''' 4. 使用颜色对话框'''
from Tkinter import *
# 引入 tkColorChoose 模块
from tkColorChooser import *
root = Tk()

# 调用 askcolor 返回选中颜色的 (R, G, B) 颜色值及 #RRGGBB 表示

```

```

print askcolor()
root.mainloop()
# 返回选中的文件名称
''' 5. 使用消息对话框'''
# -*- coding: cp936 -*-
# 使用提示对话框模块 tkMessageBox
from Tkinter import *
# 引入 tkMessageBox 模块
from tkMessageBox import *
root = Tk()
stds = [
    showinfo, # 显示信息消息框
    showwarning, # 显示警告消息框
    showerror, # 显示错误消息框
    askquestion, # 显示询问消息框
    askokcancel, # 显示确认/取消消息框
    askyesno, # 显示是/否消息框
    askretrycancel # 显示重试/取消消息框
]
for std in stds:
    print str(std),std(title = str(std),message = str(std))
# 程序打印输出结果如下（与点击的按钮得到不同其值）
# <function showinfo at 0x00D589F0> ok
# <function showwarning at 0x00D58A30> ok
# <function showerror at 0x00D58A70> ok
# <function askquestion at 0x00D58AB0> yes
# <function askokcancel at 0x00D58AF0> False
# <function askyesno at 0x00D58B30> True
# <function askretrycancel at 0x00D58B70> True
root.mainloop()
# 如果要确认点击的是那一个按钮，则可以判断这个消息框的返回值，注意各个值有所不同
# 返回值有 ok/yes/True
''' 使用缺省焦点'''
# -*- coding: cp936 -*-
# 使用提示对话框模块 tkMessageBox 缺省焦点
from Tkinter import *
from tkMessageBox import *
root = Tk()
print askokcancel(title = 'quit application?',
                  message = 'would you like quit this application',
                  default = CANCEL # 指定默认焦点位置
                  )

```

```
root.mainloop()
# 使用 default 来指定默认焦点位置, ABORT/RETRY/IGNORE/OK/CANCEL/YES/NO
# 如果指定的按钮不存在, 在抛出异常
```

## Event(1)

```
'''Tkinter 教程之 Event 篇(1)'''
# 事件的使用方法
'''1. 测试鼠标点击(Click)事件'''
# -*- coding: cp936 -*-
# 测试鼠标点击事件
from Tkinter import *
root = Tk()
def printCoords(event):
    print event.x, event.y
# 创建第一个 Button, 并将它与左键事件绑定
bt1 = Button(root, text = 'leftmost button')
bt1.bind('<Button-1', printCoords)

# 创建二个 Button, 并将它与中键事件绑定
bt2 = Button(root, text = 'middle button')
bt2.bind('<Button-2', printCoords)

# 创建第三个 Button, 并将它与右击事件绑定
bt3 = Button(root, text = 'rightmost button')
bt3.bind('<Button-3', printCoords)

# 创建第四个 Button, 并将它与双击事件绑定
bt4 = Button(root, text = 'double click')
bt4.bind('<Double-Button-1', printCoords)

# 创建第五个 Button, 并将它与三击事件绑定
bt5 = Button(root, text = 'triple click')
bt5.bind('<Triple-Button-1', printCoords)

bt1.grid()
bt2.grid()
bt3.grid()
bt4.grid()
bt5.grid()

root.mainloop()
```

```
# 分别测试鼠标的事件，回调函数的参数 event 中 (x, y) 表示当前点击的坐标值
''' 2. 测试鼠标的移动(Motion) 事件'''
```

```
# -*- coding: cp936 -*-
# 测试鼠标移动事件
from Tkinter import *
root = Tk()

def printCoords(event):
    print event.x, event.y

# 创建第一个 Button, 并将它与左键移动事件绑定
bt1 = Button(root, text = 'leftmost button')
bt1.bind('<B1-Motion>', printCoords)

# 创建二个 Button, 并将它与中键移动事件绑定
bt2 = Button(root, text = 'middle button')
bt2.bind('<B2-Motion>', printCoords)

# 创建第三个 Button, 并将它与右击移动事件绑定
bt3 = Button(root, text = 'rightmost button')
bt3.bind('<B3-Motion>', printCoords)
```

```
bt1.grid()
bt2.grid()
bt3.grid()
```

```
root.mainloop()

# 分别测试鼠标的移动事件，只有当鼠标被按下后移动才回产生事件
''' 3. 测试鼠标的释放(Release) 事件'''
```

```
# -*- coding: cp936 -*-
# 测试鼠标释放事件
from Tkinter import *
root = Tk()

def printCoords(event):
    print event.x, event.y

# 创建第一个 Button, 并将它与左键释放事件绑定
bt1 = Button(root, text = 'leftmost button')
bt1.bind('<ButtonRelease-1>', printCoords)

# 创建二个 Button, 并将它与中键释放事件绑定
bt2 = Button(root, text = 'middle button')
bt2.bind('<ButtonRelease-2>', printCoords)

# 创建第三个 Button, 并将它与右击释放事件绑定
bt3 = Button(root, text = 'rightmost button')
```

```

bt3.bind('<ButtonRelease-3>', printCoords)

bt1.grid()
bt2.grid()
bt3.grid()

root.mainloop()
# 分别测试鼠标的 Release 事件，只有当鼠标被 Release 后移动才回产生 Release 事件
''' 4. 进入(Enter)事件'''
# -*- coding: cp936 -*-
# 测试鼠标释放事件
from Tkinter import *
root = Tk()
def printCoords(event):
    print event.x, event.y
# 创建第一个 Button, 并将它与 Enter 事件绑定
bt1 = Button(root, text = 'leftmost button')
bt1.bind('<Enter>', printCoords)

bt1.grid()

root.mainloop()
# 分别测试 Enter 事件，只是在第一次进入进回产生事件，在组件中移动不会产生 Enter
事件。

```

## Event(2)

```

''' Tkinter 教程之 Event 篇(2)'''
''' 5. 测试离开(Leave)事件'''
# -*- coding: cp936 -*-
# 测试鼠标释放事件
from Tkinter import *
root = Tk()
def printCoords(event):
    print event.x, event.y
# 创建第一个 Button, 并将它与 Enter 事件绑定
bt1 = Button(root, text = 'leftmost button')
bt1.bind('<Leave>', printCoords)

bt1.grid()

```



```

root.mainloop()
# 分别测试 Leave 事件，只是在离开组件是会产生此事件。
'''6. 响应特殊键(Special Key)'''
# -*- coding: cp936 -*-
# 测试键盘特殊键事件
from Tkinter import *
root = Tk()
def printCoords(event):
    print 'event.char = ', event.char
    print 'event.keycode = ', event.keycode
# 创建第一个 Button, 并将它与 BackSpace 键绑定
bt1 = Button(root, text = 'Press BackSpace')
bt1.bind('<BackSpace>', printCoords)

# 创建二个 Button, 并将它与回车键绑定
bt2 = Button(root, text = 'Press Enter')
bt2.bind('<Return>', printCoords)

# 创建第三个 Button, 并将它与 F5键绑定
bt3 = Button(root, text = 'F5')
bt3.bind('<F5>', printCoords)

# 创建第4个 Button, 并将它与左 Shift 键绑定, 与参考上说法一致
bt4 = Button(root, text = 'Left Shift')
bt4.bind('<Shift_L>', printCoords)

# 创建第5个 Button, 并将它与右 Shift 键绑定, 与参考上说法一致
bt5 = Button(root, text = 'Right Shift')
bt5.bind('<Shift_R>', printCoords)

# 将焦点设置到第1个 Button 上
bt1.focus_set()
bt1.grid()
bt2.grid()
bt3.grid()
bt4.grid()
bt5.grid()

root.mainloop()
# 各个组件间焦点的切换可以使用 TAB 键。
#                               特          殊          键
Cancel/Break/BackSpace/Tab/Return/Sift_L/Shift_R/Control_L/Control_R/Alt_L/Alt
_R/Pause

```

```

#                               Caps_Loack/Escape/Prior(Page    Up)/Next(Page
Down)/End/Home/Left/Up/Right/Down/Print/Insert/Delete/
#                               F1-12/Num_Lock/Scroll_Lock
# 这些键的 char 是不可打印的，可以使用 event.keycode 查看。
''' 7. 响应所有的按键(Key) 事件'''
# -*- coding: cp936 -*-
# 使用 Key 处理所有的键盘事件
from Tkinter import *
root = Tk()
def printCoords(event):
    print 'event.char = ', event.char
    print 'event.keycode = ', event.keycode
# 创建第一个 Button, 并将它与 Key 键绑定
bt1 = Button(root, text = 'Press BackSpace')
bt1.bind('<Key>', printCoords)

# 将焦点设置到第1个 Button 上
bt1.focus_set()
bt1.grid()

root.mainloop()
# 处理所有的按键事件，如果是上例的特殊键，event.char 返回为空；其它情况下为这个
键的值。
# 如果输入大写字母（即上档键值），按下 Shift 键时就会有 Key 的事件触发。即回将用两
次：一次为 Shift 本身，另一次为 Shift+ Kye 的实际键值。
''' 8. 只处理指定的按键消息'''
# -*- coding: cp936 -*-
# 只处理指定的按键消息
from Tkinter import *
root = Tk()
def printCoords(event):
    print 'event.char = ', event.char
    print 'event.keycode = ', event.keycode
# 创建第一个 Button, 并将它与键'a' 绑定
bt1 = Button(root, text = 'Press BackSpace')
bt1.bind('<a>', printCoords)

# 创建二个 Button，并将它与按下 spacebar 是触发事件
bt2 = Button(root, text = 'Press spacebar')
bt2.bind('<space>', printCoords)

# 创建第三个 Button，并将它与'<' 键绑定
bt3 = Button(root, text = 'less than key')
bt3.bind('<less>', printCoords)

```

```

# 将焦点设置到第1个 Button 上
bt1.focus_set()

bt1.grid()
bt2.grid()
bt3.grid()

root.mainloop()
# 一般的按键直接使用就可以了, 这样书写'key', 不是'<key>';
# 但有两个需要特别注意: 空格与小于的处理, 使用方式为'<space>和<less>'

```

## Event(3)

```

''' Tkinter 教程之 Event 篇(3)'''
''' 11. 两个事件同时绑定到一个控件'''
# 将两个事件绑定为同一个组件
# -*- coding: cp936 -*-
# 为 root 绑定两个事件
from Tkinter import *
root = Tk()
# Key 事件处理函数
def printEvent(event):
    print '<Key>', event.keycode
# Return 事件处理函数
def printReturn(event):
    print '<Return>', event.keycode
root.bind('<Key>', printEvent)
root.bind('<Return>', printReturn)

root.mainloop()
# 当按键除了 Return 之外, 都是由 printEvent 来处理
# 当按键为 Return 时, 由 printReturn 来处理, 即由最“近”的那个事件处理。
''' 12. 为一个 instance 绑定一个事件。'''
# instance 的 bind 方法
# -*- coding: cp936 -*-
from Tkinter import *
root = Tk()
# Key 事件处理函数
def printEvent(event):
    print '<Key>', event.keycode
# Return 事件处理函数

```

```

def printReturn(event):
    print '<Return>', event.keycode
# 使用 bt1来添加一个事件处理函数。
bt1 = Button(root, text = 'instance event')
bt1.bind('<Key>', printEvent)
bt1.focus_set()
bt1.grid()

root.mainloop()
# 当按键时，程序调用一次 printEvent
''' 13. 事件各个级别传递'''
# -*- coding: cp936 -*-
# 事件级别间”传递”
from Tkinter import *
root = Tk()
# Key 事件处理函数
def printEvent(event):
    print '<instance>', event.keycode
# Return 事件处理函数
def printToplevel(event):
    print '<toplevel>', event.keycode
def printClass(event):
    print '<bind_class>', event.keycode
def printAppAll(event):
    print '<bind_all>', event.keycode

# 在 instance 级别与 printEvent 绑定
bt1 = Button(root, text = 'instance event')
bt1.bind('<Return>', printEvent)

# 在 bt1的 Toplevel 级别与 printToplevel 绑定
bt1.winfo_toplevel().bind('<Return>', printToplevel)

# 在 class 级别绑定事件 printClass
root.bind_class('Button', '<Return>', printClass)

# 在 application all 级别绑定 printAppAll
bt1.bind_all('<Return>', printAppAll)

# 将焦点定位到 bt1上，回车一下，结果有4个打印输出。
bt1.focus_set()
bt1.grid()

root.mainloop()

```

```

# 输出结果:
# <instance> 13
# <bind_class> 13
# <toplevel> 13
# <bind_all> 13
# Return 向高级别进行了“传递”,调用顺序为 instance/class/toplevel/all
''' 14. 使用 bind_class 的后果'''
# -*- coding: cp936 -*-
# 使用 bind_class 将影响所有这个类的 instance
from Tkinter import *
root = Tk()

def printClass(event):
    print '<bind_class>', event.keycode

# 改变 button 类的事件绑定
root.bind_class('Button', '<Return>', printClass)
# 创建两个 Button
bt1 = Button(root, text = 'a button')
bt2 = Button(root, text = 'another button')

bt1.focus_set()
bt1.grid()
bt2.grid()

root.mainloop()
# 回车, bt1打印结果
# TAB 切换到 bt2, 回车同样打印出结果, 即所有的 Button 对 Return 事件进行响应。
''' 15. 使用 protocol 绑定'''
# -*- coding: cp936 -*-
# 使用 protocol 与 WM 交互
from Tkinter import *
root = Tk()

def printProtocol():
    print 'WM_DELETE_WINDOW'
    root.destroy()

# 使用 protocol 将 WM_DELETE_WINDOW 与 printProtocol 绑定
root.protocol('WM_DELETE_WINDOW', printProtocol)
root.mainloop()
# 程序在退出时打印'WM_DELETE_WINDOW'

```

