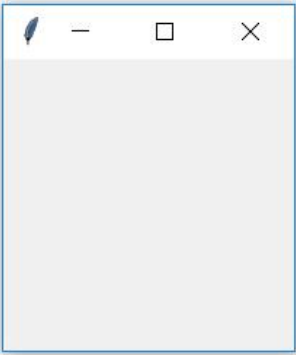


### 3 tkinter

#### 3.1 第一个 tkinter 程序

tkinter 的程序很容易编写。难点是在后面的布局以及参数的传递。很多控件(Widget)的使用方法都可以从网络上快速的找到。

简单的 tkinter 窗口程序如下：

程序	结果	说明
<pre>import tkinter root = tkinter.Tk() root.mainloop()</pre>		

只用三行程序就可以构建一个窗口程序，的确非常简单。不过这个程序没有大的用处。整个窗口也是空白的。我们会在后面的章节里面加入其他的控件，让这个窗口具有更多的功能。

详细解读一下上面的程序：

- (1) Import tkinter 是引入 tkinter 模块。所有的控件(Widget)都在里面有定义。
- (2) root = tkinter.Tk() 是实例化 Tk 类。Tk 类是顶层的控件类，完成窗口的一系列初始化。有兴趣的可以看 tkinter 类的\_\_init\_\_.py，看看 Tk 是如何完成初始化的。
- (3) root.mainloop() 是主窗口循环。

#### 3.2 窗口的基本属性

空白的窗口没有什么用处，这节会介绍一些关于窗口的基本属性与功能。

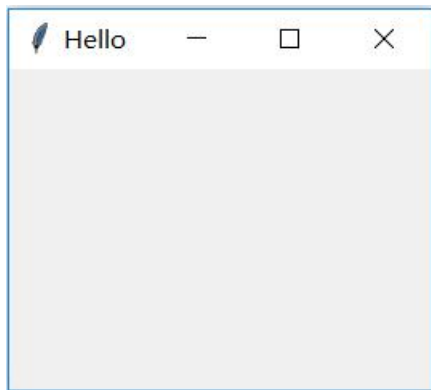
##### 3.2.1 窗口标题

给窗口加一个标题，表明这个窗口是做什么的。一般都是显示程序的名称。或者动态的提示信息，比如打开的文件名称等。

添加标题的语句是：

```
root.title('Hello')
```

把这条语句加在 mainloop()之前就可以了。



### 3.2.2 设置窗口的大小

初始化的窗口，是一个很小的窗口，连标题都无法正常显示。因此需要在程序开始的时候设置窗口的大小。

窗口有 4 个与大小有关的函数：

函数	说明	备注
<code>winfo_screenwidth()</code>	整个屏幕宽度	是电脑屏幕的宽度
<code>winfo_screenheight()</code>	整个屏幕长度	是电脑屏幕的长度
<code>winfo_reqwidth()</code>	窗口宽度	
<code>winfo_reqheight()</code>	窗口长度	

有了这 4 个函数，我们就可以轻松的实现窗口大小的调整与在屏幕中间显示。如果没有调整的话，我们创建的 `tkinter` 窗口出现在屏幕的位置，每次都是不同的。下面的程序就可以实现调整窗口大小，并在屏幕居中显示：

代码：

```
import tkinter as tk

def get_screen_size(win):
    return win.winfo_screenwidth(), win.winfo_screenheight()

def center_window(root, width, height):
    screenwidth = root.winfo_screenwidth()
    screenheight = root.winfo_screenheight()
    size = '%dx%d+%d+%d' % (width, height, (screenwidth - width)/2, (screenheight - height)/2)
    root.geometry(size)
root = tk.Tk()
root.title('调整窗口大小')
center_window(root, 400, 320)
root.mainloop()
```



要设置窗口的大小与位置，先得到屏幕的大小，然后根据要设定的窗口大小，确定窗口右上角在屏幕上的位置。就是（屏幕的宽度-窗口的宽度）/2 和（屏幕的长度-窗口的长度）/2）。然后把这些参数合成一个字符串传入 `geometry` 函数，就可以在指定的位置显示指定大小的窗口了。

函数	说明	备注
<code>geometry(string)</code>	传入宽度、高度、左上角在屏幕的相对位置	是个字符串 <code>w*h+/-x+/-y</code>
<code>minsize(x,y)</code>	最小的窗口尺寸。窗口不会比这个更小。	
<code>maxsize(x,y)</code>	最大窗口尺寸。窗口不会比这个更大	

### 3.2.3 窗口内控件的布局

有了窗口就可以在上面放置控件了。控件会在以后的章节中详细说明。本章中，先使用标签控件(`Label`)来说明布局。所有控件的布局都是继承于一样的类，所以布局的处理是完全一样。`tkinter` 有三种布局模式，`pack`, `grid` 和 `place`。`pack` 最简单，`grid` 最常用，`place` 用的最少。

#### 3.2.3.1 pack

`pack` 布局非常简单，不用做过多的设置，直接使用一个 `pack` 函数就可以了。`pack` 方法会从上到下，从左到右的摆放控件。当然也可以指定控件的位置来实现指定的效果，比如让退出按钮在右下角等。

### 3.2.3.1.1 pack 选项

pack 可以使用的选项包括：

名称	描述	取值范围
expand	当值为 True 时, side 选项无效。 控件显示在父控件中心位置；若 fill 选项为"both",则充满父控件的空间。	"yes","no","y","n" 自然数,浮点数 "no", 0,True,False (默认值为"no"或0)
fill	填充 x(y)方向上的空间,当属性 side="top"或"bottom"时,填充 x 方向；当属性 side="left"或"right"时,填充"y"方向；当 expand 选项为"yes"时,填充父控件的剩余空间。	"x", "y", "both"
ipadx, ipady	控件内部在 x(y)方向上填充的空间大小,默认单位为像素,可选单位为 c (厘米)、m (毫米)、i (英寸)、p (打印机的点,即 1/27 英寸),用法为在值后加上一个后缀既可。	非负浮点数 (默认值为 0.0)
padx, pady	控件外部在 x(y)方向上填充的空间大小,默认单位为像素,可选单位为 c (厘米)、m (毫米)、i (英寸)、p (打印机的点,即 1/27 英寸),用法为在值后加上一个后缀既可。	非负浮点数 (默认值为 0.0)
side	定义贴近在父控件的哪一边上。	"top", "bottom", "left", "right" (默认为"top")
before	将本控件于所选组建对象之前 pack, 类似于先创建本控件再创建其他控件。	已经 pack 后的控件对象

after	将本控件于所选组建对象之后 pack，类似于先创建选定控件再本控件。	已经 pack 后的控件对象
in_	将本控件作为所选组建对象的子控件，类似于指定本控件的 master 为选定控件。	已经 pack 后的控件对象
anchor	控件的摆放方式。默认是居中。左对齐“w”，右对齐“e”，顶对齐“n”，底对齐“s”。w 和 e 可以与 n 和 s 组合使用	“n”，“s”，“w”，“e”， “nw”，“sw”，“se”， “ne”，“center” (默认为“center”)

首先这段代码创建一个窗口和一个标签。正常的情况使用 pack，此标签就在窗口的顶部居中显示（为了突出显示效果，加上了蓝色的背景）。

```
import tkinter as tk
```

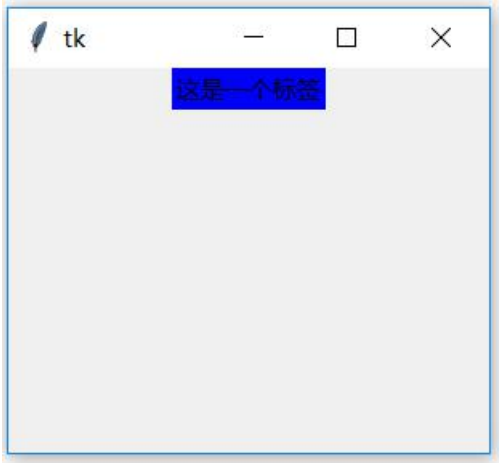
```
root=tk.Tk()
```

```
root.geometry('300x240')
```

```
L = tk.Label(root,text='这是一个标签',bg='blue')
```

```
L.pack()
```

```
root.mainloop()
```

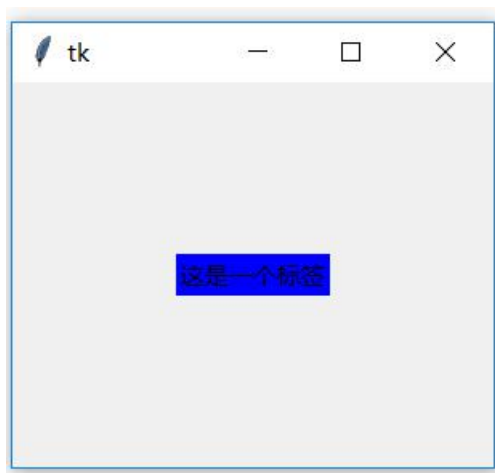


如果不是 pack 是不会显示标签的。各位读者可以把 pack() 去掉，标签控件是不会出现的。下面逐个介绍 pack() 的选项，有的时候会组合使用。

### 1. expand 的使用

修改 pack() 函数，设置 expand='yes'，可以发现标签选择不仅是左右居中，同时还是垂直居中了。

```
代码： L.pack(expand='yes')
```



## 2. fill 的使用

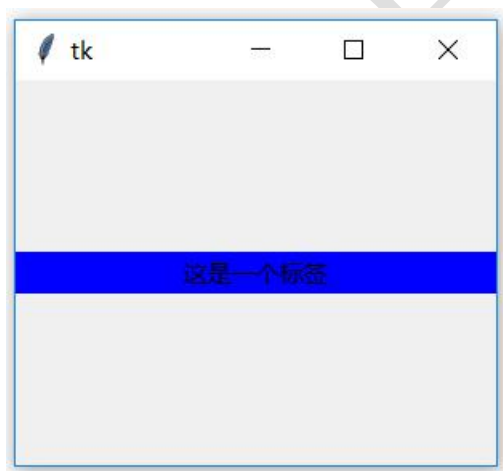
fill 的作用是选择如何填充父控件：

fill = 'x': 表示在水平方向充满整个父控件

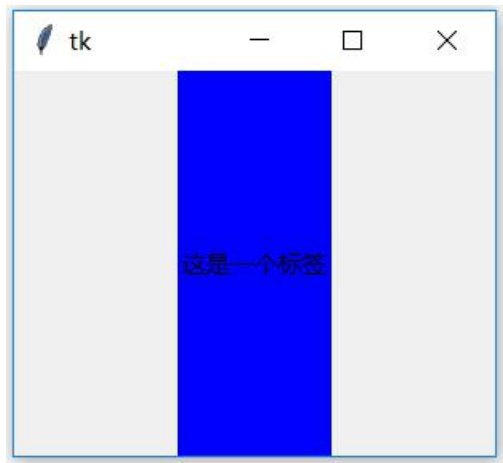
fill = 'y': 表示在垂直方向充满整个父控件

fill = 'both': 表示充满整个父控件

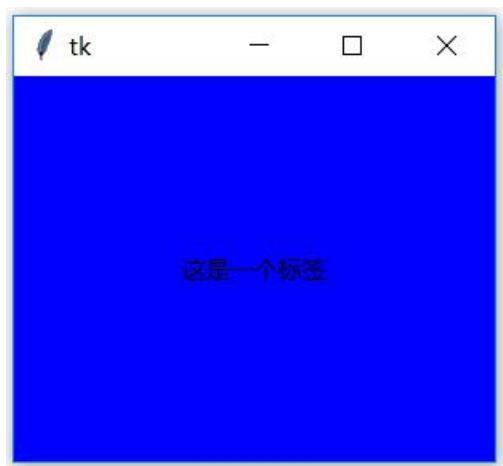
代码: `l.pack(expand='true',fill='x')`



代码: `l.pack(expand='true',fill='y')`



代码: `l.pack(expand='true',fill='both')`



这是只有一个控件的情况。如果有两个控件，会是什么样的结果呢？

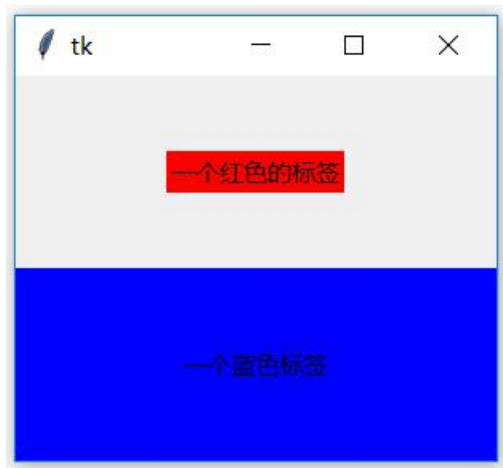
代码:

```
r = tk.Label(root,text='一个红色的标签',bg='red')
```

```
b = tk.Label(root,text='一个蓝色标签',bg='blue')
```

```
r.pack(expand='true')
```

```
b.pack(expand='true',fill='both')
```

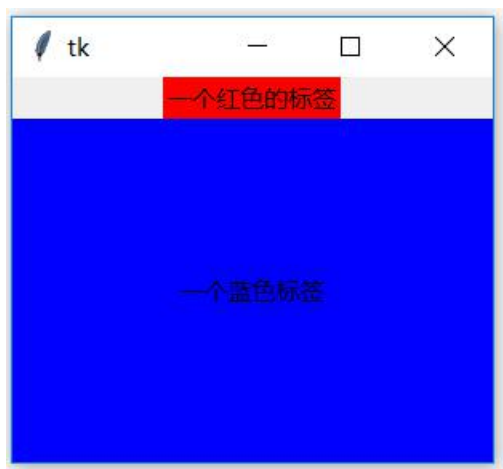


可以看见，两个标签上下平分了窗口。不过有 `fill` 选项的会充满下半个窗口，为没有 `fill` 选项的，只是在上半个窗口居中显示。

如果上半个窗口没有开启 `expand` 选项，那么红色的标签只是在顶端居中显示，剩下的空间由蓝色标签充满。

代码：

```
r.pack()  
b.pack(expand='true',fill='both')
```



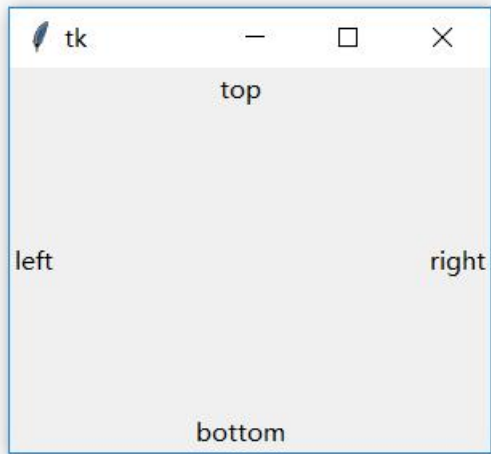
### 3. `side` 的作用

`side` 的作用是放置控件的位置。有四个位置：`left`,`right`,`top`,`bottom`。可以输入字符串，也可以使用 `tkinter` 模块中的常量 `LEFT`,`RIGHT`,`TOP`,`BOTTOM`。

代码：

```
b1 = tk.Label(root,text='left')  
b1.pack(side=tk.LEFT)  
b2 = tk.Label(root,text='right')  
b2.pack(side=tk.RIGHT)  
b3 = tk.Label(root,text='top')  
b3.pack(side=tk.TOP)  
b4 = tk.Label(root,text='bottom')  
b4.pack(side=tk.BOTTOM)
```



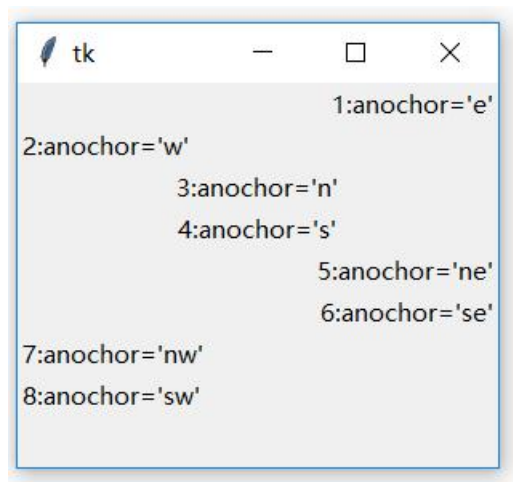


#### 4. anchor 的作用

主要作用是如何摆放控件。如果没有指定 `anchor` 选项，控件是在父窗口中自顶向下摆放的。`anchor` 一共有八个选项，其实就是我们通常意义上的八个方位。

代码：

```
b1 = tk.Label(root, text='1: anchor=\e\')
b1.pack(anchor='e')
b2 = tk.Label(root, text='2: anchor=\w\')
b2.pack(anchor='w')
b3 = tk.Label(root, text='3: anchor=\n\')
b3.pack(anchor='n')
b4 = tk.Label(root, text='4: anchor=\s\')
b4.pack(anchor='s')
b5 = tk.Label(root, text='5: anchor=\ne\')
b5.pack(anchor='ne')
b6 = tk.Label(root, text='6: anchor=\se\')
b6.pack(anchor='se')
b7 = tk.Label(root, text='7: anchor=\nw\')
b7.pack(anchor='nw')
b8 = tk.Label(root, text='8: anchor=\sw\')
b8.pack(anchor='sw')
```



anchor 和 side 的作用都是用来定位控件位置的。可以组合起来使用。

#### 5. ipadx 和 ipady 的使用

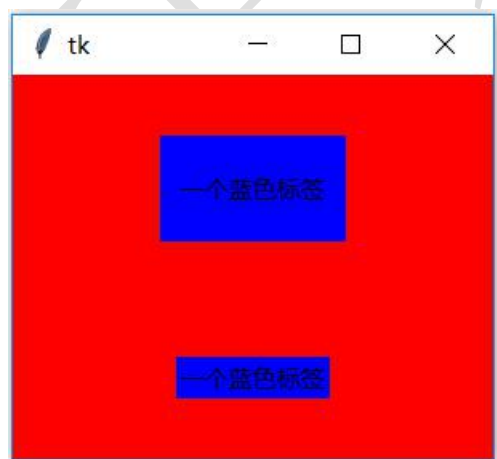
ipadx 和 ipady 是内部填充空间的大小。什么意思呢？就是把控件往外扩大的空间。比如控件的宽和长是 30 和 20。那么使用 ipadx 和 ipady 之后，控件的宽和长就是  $30+ipadx$  和  $20+ipady$ 。

代码：

```
r = tk.Label(root,bg='red')
r.pack(expand='true',fill='both')
```

```
b = tk.Label(r,text='一个蓝色标签',bg='blue')
b.pack(expand='true',ipadx=10,ipady=20)
```

```
b2 = tk.Label(r,text='一个蓝色标签',bg='blue')
b2.pack(expand='true')
```



下面的蓝色标签是对照的。可以看见在 x 和 y 的方向上，上面的蓝色标签都扩大了。增加的就是 ipadx 和 ipady。

有关单位的描述见上面的表格。一般都是使用默认的单位：像素。

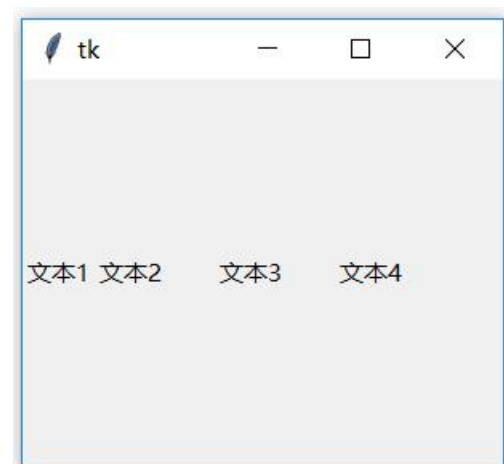
## 6. padx 和 pady 的使用

padx 和 pady 是外部间隔，也就是两个控件之间的间隔。

padx 就是 x 方向的间距。参数可以是自然数，也可以是自然数的元组，分别表示左边和右边的间隔。

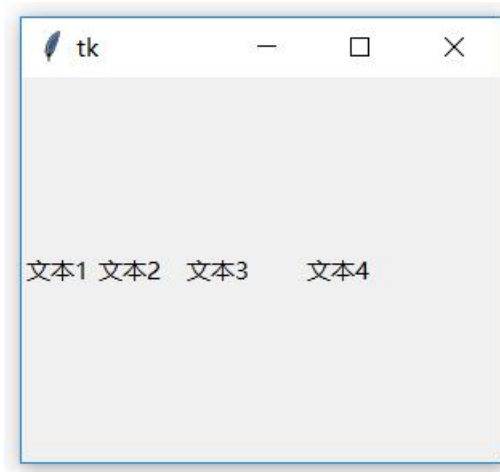
这段代码是自然数的情况（文本 3 与文本 2 和 4 的间隔是 30 个像素）：

```
b1 = tk.Label(root,text='文本 1')
b1.pack(side='left')
b2 = tk.Label(root,text='文本 2')
b2.pack(side='left')
b3 = tk.Label(root,text='文本 3')
b3.pack(side='left',padx=30)
b4 = tk.Label(root,text='文本 4')
b4.pack(side='left')
```



这个是间隔为（10，30）的情况。

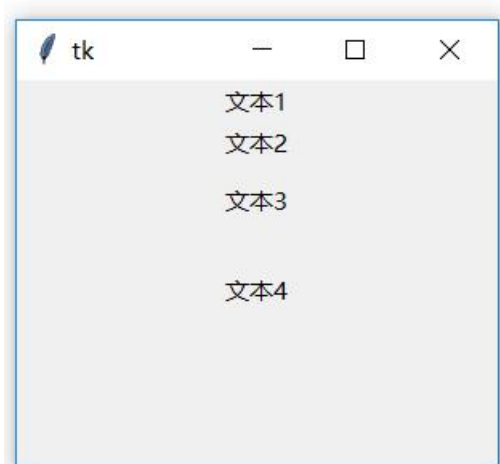
代码： `b3.pack(side='left',padx=(10,30))`



`pady` 和 `padx` 类似，只不过是 `y` 方向的间隔。也可以使用元组来表示上下不同的间隔

代码：

```
b1 = tk.Label(root,text='文本 1')
b1.pack()
b2 = tk.Label(root,text='文本 2')
b2.pack()
b3 = tk.Label(root,text='文本 3')
b3.pack(pady=(10,30))
b4 = tk.Label(root,text='文本 4')
b4.pack()
```



## 7. `before`、`after` 和 `in_`

`before` 和 `after` 可以改变 `pack` 控件的次序。正常的情况是按照代码的次序 `pack`。不过当需要提前或者推后 `pack` 的次序时，可以使用 `before` 或者 `after`。比如 `before=w1,after=w2` 等等。`w1` 和 `w2` 是创建的控件的实例。

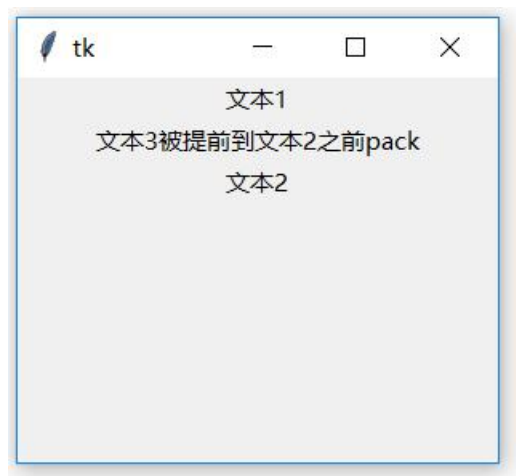
下面的代码就是先 `pack` 标签 `b3` 的。改变了次序。`after` 的情况就不展示了。和 `before` 非常类似。

代码：

```
b1 = tk.Label(root,text='文本 1')  
b1.pack()
```

```
b2 = tk.Label(root,text='文本 2')  
b2.pack()
```

```
b3 = tk.Label(root,text='文本 3 被提前到文本 2 之前 pack')  
b3.pack(before=b2)
```



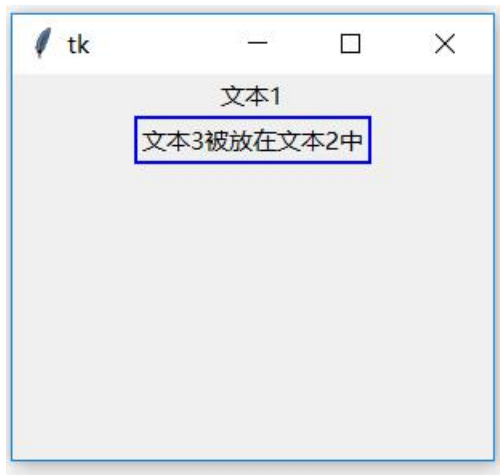
`in_`是因为 `in` 是 python 的关键字，所以加了一个下划线。`in_`的作用是设置当前控件的父控件，可以替换掉在控件初始化时候的父控件。本例中，就是用 `b2` 替换了 `root`。

代码：

```
b1 = tk.Label(root,text='文本 1')  
b1.pack()
```

```
b2 = tk.Label(root,text='文本 2',bg='blue')  
b2.pack()
```

```
b3 = tk.Label(root,text='文本 3 被放在文本 2 中')  
b3.pack(in_=b2)
```



### 3.2.3.1.2 pack 函数

pack 的函数包括：

函数名	描述
slaves()	以列表方式返回本控件的所有子控件对象。
propagate(flag)	设置为 <b>True</b> 表示父控件的几何大小由子控件决定（默认值），反之则无关。
info()	返回 pack 提供的选项所对应的数值，返回值为字典类型
forget()	将控件从当前的控件管理器中移除，其实是将控件隐藏并且忽略原有设置，对象依旧存在，可以用 <b>pack(option, ...)</b> 或其他的布局方法，将其显示。
location(x, y)	<b>x, y</b> 为以像素为单位的点，函数返回此点是否在单元格中，在哪个单元格中。返回单元格行列坐标， <b>(-1, -1)</b> 表示不在其中。
size()	返回控件所包含的单元格，揭示控件大小。

#### 1. slaves() 函数

**slaves()**函数返回本控件的所有子控件对象。如果不使用 **pack()**，就算已经实例化了子控件，**slaves()**也不会输出没有 **pack()**的子控件。比如 **b4** 就不会输出。而 **b2** 和 **b3** 会被认为是 **b1** 的子控件。

代码:

```
b1 = tk.Label(root,text='文本 1')
b1.pack()

b2 = tk.Label(b1,text='文本 2')
b2.pack()

b3 = tk.Label(b1,text='文本 3')
b3.pack()
b4 = tk.Label(b1,text='文本 4')

print(b1.slaves())
```

注意: 子控件的子控件是不会被输出的。

## 2. propagate(flag) 函数

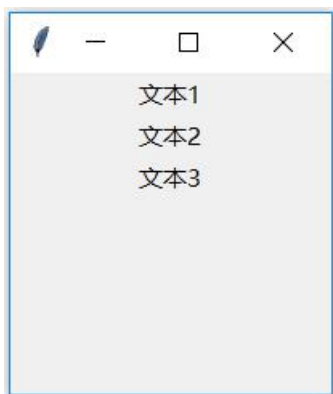
该函数决定父控件的大小是否与子控件有关。如果 **flag** 是 **True** 则父控件的大小为包括子控件的大小。如果 **flag** 是 **False**,则表示父控件的大小与子控件无关。不过 **geometry()**会让 **propagate()**失效,窗口的大小由 **geometry()**决定。

代码:

```
b1 = tk.Label(root,text='文本 1')
b1.pack()

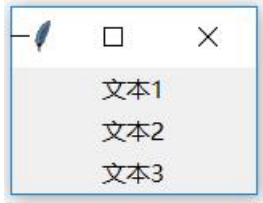
b2 = tk.Label(root,text='文本 2')
b2.pack()

b3 = tk.Label(root,text='文本 3')
b3.pack()
root.propagate(False)
```



代码:

```
root.propagate(True)
```



可以看出 True 或者 False 情况下，窗口的大小是不同。

### 3. info() 函数

info()返回控件的信息。这些信息以字典的形式返回。

比如上面例子 b2 的信息如下:

代码:

```
print(b2.info())
```

```
{'in': <tkinter.Tk object .>, 'anchor': 'center', 'expand': 0, 'fill': 'none', 'ipadx': 0, 'ipady': 0, 'padx': 0, 'pady': 0, 'side': 'top'}
```

### 4. forget() 函数

forget()函数是隐藏控件的。调用之后，该控件从父控件中消失。但是该控件的实例还是存在的。可以用 pack()直接恢复显示这个控件。为了演示这个功能，需要增加 2 个按钮和 2 个回调函数。回调函数 b3\_forget()是隐藏标签 b3，而 b3\_pack()是显示标签 b3 的。这次的代码，会贴上完整的部分。

代码:

```
import tkinter as tk
```

```
root=tk.Tk()
```

```
root.geometry('300x240')
```

```
b1 = tk.Label(root,text='文本 1')
```

```
b1.pack()
```

```
b2 = tk.Label(root,text='文本 2')
```

```
b2.pack()
```

```
b3 = tk.Label(root,text='文本 3')
```

```
b3.pack()
```

```
def b3_forget():
```

```
    b3.forget()
```

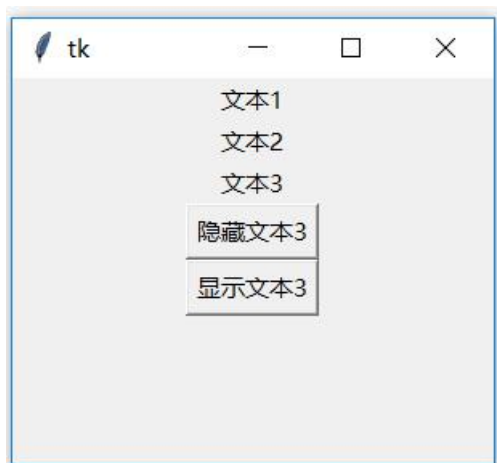
```
def b3_pack():
```

```
    b3.pack()
```

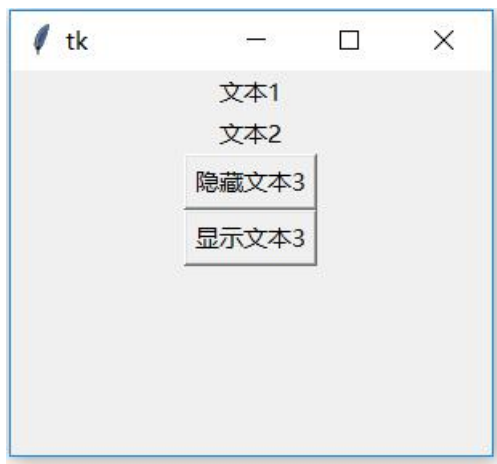


```
b4 = tk.Button(root,text='隐藏文本 3',command=b3_forget)
b4.pack()
b5 = tk.Button(root,text='显示文本 3',command=b3_pack)
b5.pack()
```

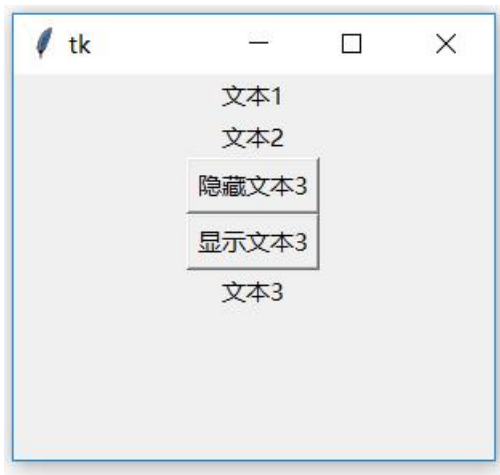
```
root.mainloop()
```



点击'隐藏文本 3'按钮后，'文本 3'的标签消失。



再点击'显示文本 3'按钮，则显示'文本 3'



注意：此时‘文本 3’显示在最下面。因为 `pack()` 是按照次序来的。以 `pack()` 的次序为准，而不是以实例化的次序。

5. `location(x,y)`函数

其实这个函数是给 `grid` 布局方法使用的。在 `pack` 方法中也可以用，不过无论怎么调用，返回的都是 `(-1,-1)`。也就是不在任何单元格内。的确，`pack` 方法就没有单元格的概念，怎么会有符合条件的单元格呢？

6. `size()`函数

`size()` 函数是返回包括控件的单元格。和 `location()` 函数一样，在 `pack` 中无效。因为所有控件的返回值都是 `(0,0)`。

3.2.3.2 `grid`

`grid` 布局管理采用类似表格的结构来管理控件的，使用起来非常灵活。就是把窗口用单元格的形式来定位，可以像编辑表格一样放置控件。`grid` 采用行列确定位置，行列交汇处为一个单元格。每一列中，列宽由这一列中最宽的单元格确定。每一行中，行高由这一行中最高的单元格决定。控件并不是充满整个单元格的，你可以指定单元格中剩余空间的使用。你可以空出这些空间，也可以在水平或竖直或两个方向上填满这些空间。也可以连接若干个单元格为一个更大空间，这一操作被称作跨越(`span`)，与 `excel` 的合并单元格是一个概念。使用 `grid` 的布局方法的时候，单元格必须是紧邻创建的。

3.2.3.2.1 `grid` 选项

名称	描述	取值范围
----	----	------

column	控件所置单元格的列号。	自然数（起始默认值为 0，往后累加）
columnspan	从控件所置单元格算起在列方向上的跨度。	自然数（起始默认值为 0）
ipadx, ipady	控件内部在 x(y)方向上填充的空间大小，默认单位为像素，可选单位为 c（厘米）、m（毫米）、i（英寸）、p（打印机的点，即 1/27 英寸），用法为在值后加以上一个后缀既可。	非负浮点数 （默认值为 0.0）
padx, pady	控件外部在 x(y)方向上填充的空间大小，默认单位为像素，可选单位为 c（厘米）、m（毫米）、i（英寸）、p（打印机的点，即 1/27 英寸），用法为在值后加以上一个后缀既可。	非负浮点数 （默认值为 0.0）
row	控件所置单元格的行号。	自然数（起始默认值为 0，往后累加）
rowspan	从控件所置单元格算起在行方向上的跨度。	自然数（起始默认值为 0）
in_	将本控件作为所选组建对象的子控件，类似于指定本控件的 master 为选定控件。	已经显示的控件对象
sticky	控件紧靠所在单元格的某一边角。	"n", "s", "w", "e", "nw", "sw", "se", "ne", "center" (默认为" center")

有些选项的用法在 **pack** 布局方法中已经介绍过了。他们在 **grid** 布局方法中也有，而且是一样的：

\* padx

- \* pady
- \* padx
- \* ipady
- \* in\_

这些选项就不在 `grid` 布局方法中再次赘述了。

## 1. row 和 column

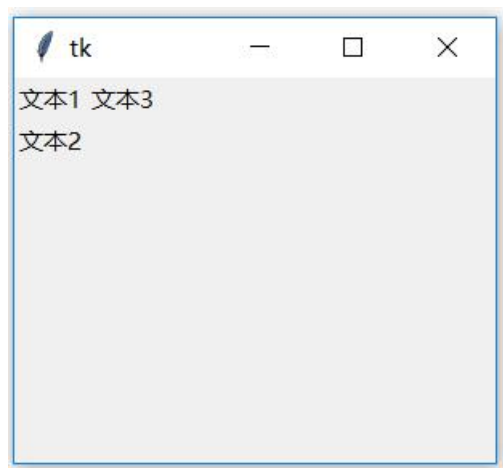
顾名思义就是用行与列来定义控件摆放的位置。

代码：

```
b1 = tk.Label(root,text='文本 1')  
b1.grid(row=0,column=0)
```

```
b2 = tk.Label(root,text='文本 2')  
b2.grid(row=1 ,column=0)
```

```
b3 = tk.Label(root,text='文本 3')  
b3.grid(row=0,column=1)
```



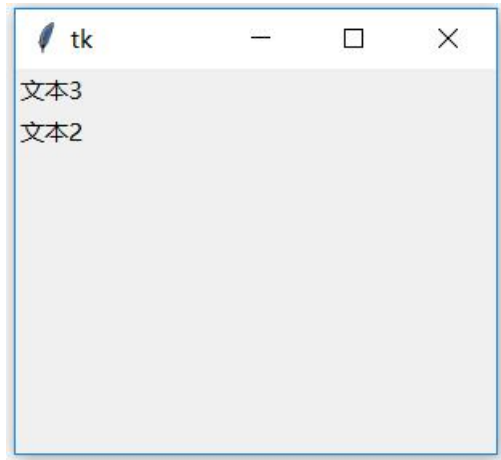
说明：

第一行，第一列定义的是标签 **b1**，第一行，第二列定义的是标签 **b3**

第二行，第一列定义的是标签 **b2**。

默认的显示是居左，并且不用按顺序说明控件。但是不能有错，比如两个控件在一个单元格内是不可以的。

比如把 b1 和 b3 都定义在第一行、第一列，那么就会显示后面声明的。



最先定义的标签 b1，消失了。这是因为 b1 和 b3 重叠了。

## 2. rowspan 和 colspan

这两个的功能是分别实现跨行与跨列的功能。其实和 excel 中的跨单元格的方法是一样的。

代码:

```
b1 = tk.Label(root,text='跨\n 两\n 列',bg='blue')
```

```
b1.grid(row=0,column=0,rowspan=2)
```

```
b2 = tk.Label(root,text='跨两行',bg='yellow')
```

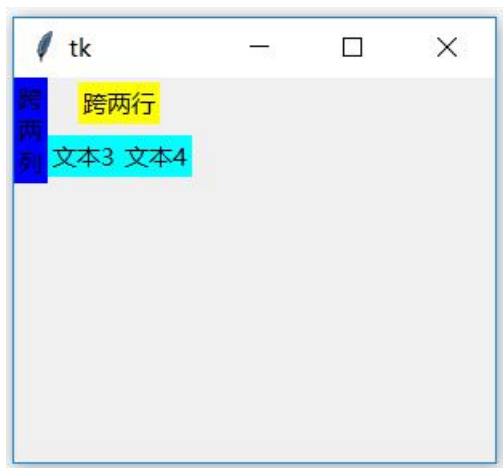
```
b2.grid(row=0 ,column=1,colspan=2)
```

```
b3 = tk.Label(root,text='文本 3',bg='cyan')
```

```
b3.grid(row=1,column=1)
```

```
b4 = tk.Label(root,text='文本 4',bg='cyan')
```

```
b4.grid(row=1,column=2)
```

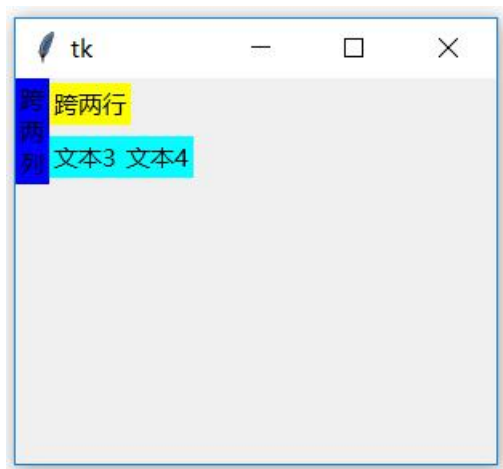


### 3. sticky

与 pack 的 anchor 方法类似，也是有八个方向。主要是定义控件在单元格内靠向那一边。比如，左边、右边还是中间。

上一个例子中，标签‘跨两行’是居中显示的。我们可以让它靠左。这就要使用 sticky 代码：

```
b2.grid(row=0,column=1,columnspan=2,sticky='w')
```



#### 3.2.3.2.2 grid 函数

函数名	描述
grid_slaves()	以列表方式返回本控件的所有子控件对象。

<code>grid_propagate(boolea)</code>	设置为 <b>True</b> 表示父控件的几何大小由子控件决定（默认值），反之则无关。										
<code>grid_info()</code>	返回提供的选项所对应的值。										
<code>grid_forget()</code>	将控件隐藏并且忽略原有设置，对象依旧存在，可以用 <code>grid(option, ...)</code> ，将其显示。										
<code>grid_remove ()</code>	和 <code>grid_forget()</code> 类似。不过会记住当前的选项。重新.grid 之后，会使用当前的选项。										
<code>grid_location(x, y)</code>	设定控件在屏幕中相对于容纳单元的（ <b>x</b> , <b>y</b> ）坐标，并返回 <b>grid</b> 系统中的哪个单元包含了该坐标（ <b>column</b> , <b>row</b> ）。										
<code>grid_bbox(column=None, row=None, col2=None, row2=None)</code>	<p>返回一个有四个元素的元组，用来描述控件内一些或者全部单元的边界。返回的前两个数为左上方区域的 <b>x</b>, <b>y</b> 坐标，后两个数为宽度和高度。</p> <p>如果只传递了 <b>column</b> 和 <b>row</b> 参数，返回的参数描述的是该行列的单元的大小。如果传递了 <b>col2</b> 和 <b>row2</b> 参数，返回的参数描述的就是从 <b>column</b> 列 到 <b>col2</b> 列，以及从 <b>row</b> 行 到 <b>row2</b> 行总体区域的大小。</p>										
<code>grid_rowconfigure(index, **options)</code>	<table><tr><td colspan="2">设定行属性</td></tr><tr><th>选项</th><th>含义</th></tr><tr><td><b>minsize</b></td><td>指定该行的最小高度</td></tr><tr><td><b>pad</b></td><td>指定该列中最大网格的垂直边距</td></tr><tr><td><b>weight</b></td><td>指定行于行之间的相对距离 默认值是 0</td></tr></table>	设定行属性		选项	含义	<b>minsize</b>	指定该行的最小高度	<b>pad</b>	指定该列中最大网格的垂直边距	<b>weight</b>	指定行于行之间的相对距离 默认值是 0
设定行属性											
选项	含义										
<b>minsize</b>	指定该行的最小高度										
<b>pad</b>	指定该列中最大网格的垂直边距										
<b>weight</b>	指定行于行之间的相对距离 默认值是 0										
<code>grid_columnconfigure(index, **options)</code>	<table><tr><td colspan="2">设置列属性</td></tr><tr><th>选项</th><th>含义</th></tr><tr><td><b>minsize</b></td><td>指定该列的最小高度</td></tr><tr><td><b>pad</b></td><td>指定该列中最大网格的水平边距</td></tr></table>	设置列属性		选项	含义	<b>minsize</b>	指定该列的最小高度	<b>pad</b>	指定该列中最大网格的水平边距		
设置列属性											
选项	含义										
<b>minsize</b>	指定该列的最小高度										
<b>pad</b>	指定该列中最大网格的水平边距										

	weight	<p>指定列与列之间的相对距离</p> <p>默认值是 0</p> <p>--说明：初创建窗口的时候，grid 会自动根据控件的尺寸分配窗口的尺寸，当你拉伸窗口的尺寸就会有空白显示出来。这个选项正是指定列与列之间是否填充空白，默认是不填充的。另外，该选项的值是指定填充空白的倍数，例如 <b>weight=2</b> 的列会比 <b>weight=1</b> 的列填充多一倍的空白，所以需要平均填充的话，只需要所有的列都设置为 <b>weight=1</b> 即可</p>

grid 的函数与 pack 类似，比如：

```
grid_slaves()
grid_propagate()
grid_info()
grid_forget()
都是一样的。
```

grid\_remove()的功能是与 grid\_forget 类似。唯一的区别就是保留了当前控件的配置选项，在重新显示的时候，会使用当前的选项。而 grid\_forget()则必须重新设置。

#### 1. grid\_location(x,y)

为了更好的实验，这次代码使用 Canvas 控件。

代码：

```
b1 = tk.Canvas(root,bg='blue',width=40,height=80)
b1.grid(row=0,column=0,rowspan=2)
```

```
b2 = tk.Canvas(root,bg='yellow',width=80,height=40)
b2.grid(row=0 ,column=1,columnspan=2)
```

```
b3 = tk.Canvas(root,bg='cyan',width=40,height=40)
b3.grid(row=1,column=1)
```

```
b4 = tk.Canvas(root,bg='gray',width=40,height=40)
b4.grid(row=1,column=2)
```

```
i=30
```

```
j=120
```



```
m = root.grid_location(i,j)
print("i=",i,"j=",j,m)
```

元组 `m` 的值是 `(0,2)`。意味着 `(30, 120)` 在第一行，第三列上。调整 `i, j` 的值，可以得到不同的单元格。

注：不太明白这个函数的用途。难道是为了游戏开发的？用 `tkinter` 开发游戏是一个疯狂的想法。

## 2. `grid_bbox()`

这个函数就是返回控件中的单元格边框。一共有四个参数。就是指定起始的行、列（比如 `0`，`0` 表示第一行、第一列），以及结束的行列。需要注意的一点是，需要在 `mainloop()` 之后调用才有正确的结果。比如，本例就是在按钮的回调函数中，调用的 `grid_bbox`。

代码：

```
t = tk.Frame(root)
t.grid(row=0,column=0)
b1 = tk.Canvas(t,bg='blue',width=40,height=80)
b1.grid(row=0,column=0,rowspan=2)

b2 = tk.Canvas(t,bg='yellow',width=80,height=40)
b2.grid(row=0 ,column=1,columnspan=2)

b3 = tk.Canvas(t,bg='cyan',width=40,height=40)
b3.grid(row=1,column=1)

b4 = tk.Canvas(t,bg='gray',width=40,height=40)
b4.grid(row=1,column=2)
def bbb():
    print(t.grid_bbox(0,0,1,1))
b5 = tk.Button(t,text='bbox',command=bbb)
b5.grid(row=2,column=0)
```

结果：

`(0, 0, 96, 88)`

不过这个函数应用的情况不是很多。至少我很少看到关于这个函数的代码与说明。

## 3. `grid_rowconfigure()`与 `grid_columnconfigure()`

用来指定行列的属性。详细的说明见上表。这里不再重复。

### 3.2.3.3 `place`

`place` 方法是最简单的一种布局方法，只需指定控件的显示位置即可。因为该方法太简单了，有很多工作需要开发人员自己完成。所以，最好是使用 `pack` 或者 `grid` 方法。

#### 3.2.3.3.1 `place` 选项

名称	描述	取值范围
anchor	设定控件在 <b>place</b> 分配的空间中的位置，用 <b>N, NE, E, SE, S, SW, W, NW</b> 或 <b>CENTER</b> 来定位（ <b>EWSN</b> 表示东南西北）。默认值是 <b>NW</b>	
bordermode	子控件定位时，是否考虑父控件的边框， <b>INSIDE</b> 是考虑， <b>OUTSIDE</b> 是不考虑。（ <b>INSIDE</b> 或 <b>OUTSIDE</b> ）。默认值是 <b>INSIDE</b>	
height	设定该控件的高度（像素）	
in_	将本控件作为所选组建对象的子控件，类似于指定本控件的 <b>master</b> 为选定控件。	
relheight	指定该控件相对于父控件的高度	取值范围是 0.0~1.0
relwidth	指定该控件相对于父控件的宽度	取值范围是 0.0~1.0
relx	指定该控件相对于父控件的水平位置	取值范围是 0.0~1.0
rely	指定该控件相对于父控件的垂直位置	取值范围是 0.0~1.0
width	设定该控件的宽度（像素）	
x	设定该控件的水平偏移位置（像素） 注：如果同时指定了 <b>relx</b> 选项，则与 <b>x</b> 一同起作用	
y	设定该控件的垂直偏移位置（像素）	

	--如果同时指定了 <b>rely</b> 选项，则与 <b>y</b> 一同起作用	
--	--	--

### 1. x 和 y 选项

这个就是直接指定在何处显示控件。

代码：

```
import tkinter as tk
```

```
root=tk.Tk()
```

```
root.geometry('300x240')
```

```
t = tk.Frame(root,width=280,height=230)
```

```
b1 = tk.Canvas(t,bg='blue',width=40,height=80)
```

```
b1.place(x=10,y=10)
```

```
b2 = tk.Canvas(t,bg='yellow',width=80,height=40)
```

```
b2.place(x=130,y=10)
```

```
b3 = tk.Canvas(t,bg='cyan',width=40,height=40)
```

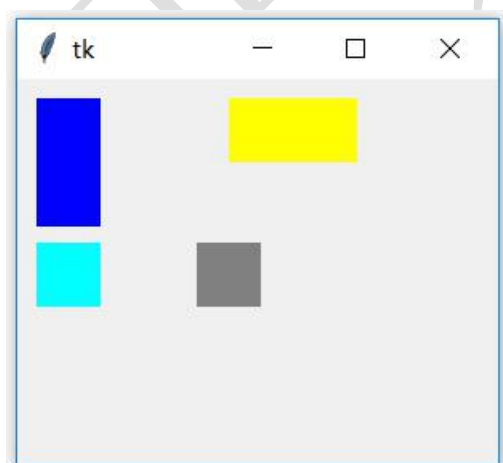
```
b3.place(x=10,y = 100)
```

```
b4 = tk.Canvas(t,bg='gray',width=40,height=40)
```

```
b4.place(x=110,y=100)
```

```
t.place(x=0,y=0)
```

```
root.mainloop()
```



注：必须设定父控件的大小。否则，不会显示子控件。place 用起来很繁琐。需要注意很多事情。好处就是可以随心所欲的摆放控件。

## 2. relx 和 rely

relx 和 rely 也是设定子控件的位置的。不过是相对的位置。取值范围是 0~1。如果同时设定了 x 和 y, 那么就是把计算出来的相对位置与 x, y 分别相加, 得出来的就是该控件的位置。比如  $x=50, y=50, relx=0.5, rely=0.5$ , 而父控件的大小是 280 和 230, 那么该控件的位置就是:

$$x = 50 + 280 * 0.5 = 50 + 140 = 190$$
$$y = 50 + 230 * 0.5 = 50 + 115 = 165$$

我们把上一个程序中的 b4 使用上述方法, 同时让 b2 的  $x=190, y=165$ , 看看两个图形是否重叠。由于 b4 是后声明的 place, 所以 b4 会遮盖一部分 b2 的图形。

代码:

```
import tkinter as tk

root=tk.Tk()
root.geometry('300x240')

t = tk.Frame(root,width=280,height=230)

b1 = tk.Canvas(t,bg='blue',width=40,height=80)
b1.place(x=10,y=10)

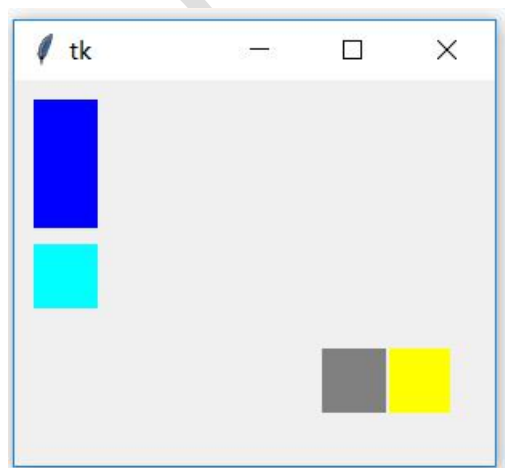
b2 = tk.Canvas(t,bg='yellow',width=80,height=40)
b2.place(x=190,y=165)

b3 = tk.Canvas(t,bg='cyan',width=40,height=40)
b3.place(x=10,y = 100)

b4 = tk.Canvas(t,bg='gray',width=40,height=40)
b4.place(x=50,y=50,relx=0.5,rely=0.5)

t.place(x=0,y=0)

root.mainloop()
```



b4 果然遮盖了 b2 的一部分。另外，这个时候 x, y 的数值可以取负数。

### 3. weight 和 height

定义子控件的大小，单位是像素。如果子控件已经定义了大小，会被忽略。

代码：

```
import tkinter as tk

root=tk.Tk()
root.geometry('300x240')

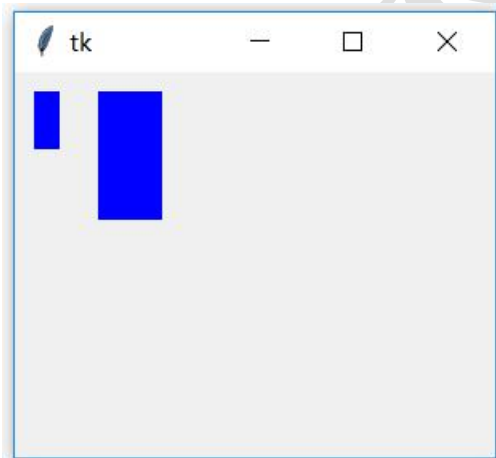
t = tk.Frame(root,width=280,height=230)

b1 = tk.Canvas(t,bg='blue',width=40,height=80)
b1.place(x=10,y=10,width=20,height=40)

b2 = tk.Canvas(t,bg='blue',width=40,height=80)
b2.place(x=50,y=10)

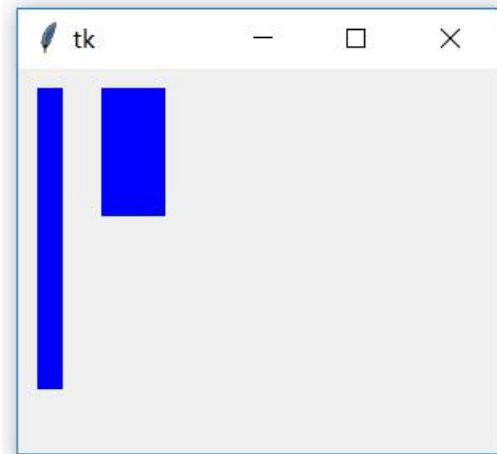
t.place(x=0,y=0)

root.mainloop()
```



### 4. relwidth 和 relheight

这两个参数是定义子控件相对于父控件的宽度与长度，如果同时定义了 width 和 height，会把数值分别相加来定义子控件。参加 x,y 和 relx, rely。



### 5. bordermode

bordermode 是指定位时是否对父控件边框考虑在内。INSIDE 是考虑，OUTSIDE 是不考虑。

代码：

```
import tkinter as tk
```

```
root=tk.Tk()
```

```
root.geometry('300x240')
```

```
t = tk.Frame(root,width=280,height=230,bd=20)
```

```
b1 = tk.Label(t,bg='blue',width=40,height=80,bd=200)
```

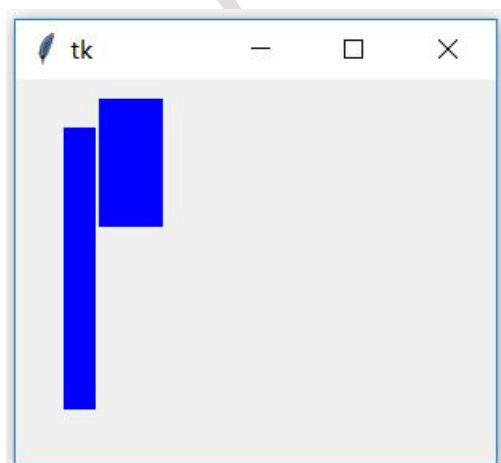
```
b1.place(x=10,y=10,width=20,height=100,relheight=0.4,bordermode=tk.INSIDE)
```

```
b2 = tk.Canvas(t,bg='blue',width=40,height=80)
```

```
b2.place(x=50,y=10,bordermode=tk.OUTSIDE)
```

```
t.place(x=0,y=0)
```

```
root.mainloop()
```



## 6. anchor

看说明是定位显示的位置。不过完全不知道怎么实现的。结果不可预料。不建议使用。

## 7. in\_

与其他的两种方法类似。请参照。

### 3.2.3.3.2 place 函数

函数名	描述
<code>place_slaves()</code>	以列表方式返回本控件的所有子控件对象。
<code>place_info()</code>	返回提供的选项所对应得值。
<code>place_forget()</code>	将控件隐藏并且忽略原有设置，对象依旧存在，可以用 <code>place(option, ...)</code> ，将其显示。