* Table of Contents
* 前⾔：GUI与tkinter
* 认识tkinter
* 第⼀个tkinter程序
* 初识组件之标签
* 组件介绍与按钮
* Tkinter的三种布局
* 事件与绑定
* 输⼊框以及登录程序
* 菜单与选择按钮
* 对话框和消息框
* 常⽤控件的介绍
* Canvas与图形绘制
* 窗⼝美化效果
* ttk介绍以及总结
* 桌⾯应⽤项⽬：⼩球弹弹弹
* 1.1
* 1.2
* 1.3
* 1.4
* 1.5
* 1.6
* 1.7
* 1.8
* 1.9
* 1.10
* 1.11
* 1.12
* 1.13
* 1.14
* 1.15
* 1
* 前⾔：GUI与tkinter
* GUI编程
* 如果同学们有过 C++ 或者 C#或者 VB 等语⾔的编程经验，甚⾄是咱们⾃⼰国  
  家的易语⾔编程，它们都有⼀些可以通过拖拽的⽅式来编辑界⾯的⼯具。
* 但是很可惜，在 Python ⾥⾯这种⼯具并不多，这也是和 Python 是⼀个脚本语  
  ⾔有关。
* 要说 tkinter，先说 tk，它原本是 Tcl 编程语⾔的界⾯库， 后来开发出了
* Python 接⼝，Tkinter 在 Python2 和 Python3 中的变化较⼤，在 Python2 中叫  
  做 Tkinter，在 Python3 中叫做 tkinter，⽽且在导⼊类库的时候也有些许 的变
* 化，请读者稍加注意。
* tkinter的定位
* 掌握⼀⻔ Python 的界 ⾯编程也很有意思。有时候我们需要⼀些界⾯程序，但  
  是⼜不想⽤那些庞⼤ ⽽繁杂的类库的时候，tkinter 的优势就显现出来了，那就  
  是简洁简单。  
  我感觉 tkinter 是⼀个还不错的界⾯库，但是我感觉 Python 并不是特别擅⻓编  
  写界⾯应⽤，⽽且我感觉⽐如 wxPython 或者 pyQt 等更胜⼀筹，但是不管怎  
  么样，我们可以通过Python⾃带的tkinter，进⼊GUI编程的世界。
* 传智播客 ⼤猫
* 2
* 认识tkinter
* Tkinter 简介
* 1. 提到 Tkinter,不得不提⼀下 GUI。GUI 即 graphical user interface,即图形界⾯⽤户接
* ⼝，是相对于我们学习绝⼤多数编程语⾔所接触到的“控制台”的那个⿊乎乎的命令⾏  
  来说，区分的图形界⾯编程模式。
* 2. Tkinter 是 Tk 的 Python 接⼝,⽽ Tk 原本是 Tcl 的 GUI 库,Tcl 是⼀⻔语⾔,具体我就不
* 多说了。
* Tkinter 的优点和缺点
* 1. 简单易学，相对于其他的⽐如 wxPython，PyQt 等等,学起来会⾮常轻松。
* 2. 可以使⽤少量的代码产⽣功能强⼤的 GUI 界⾯，这⼀点 是相对于 C++等语⾔来说  
  的，这也是作为⼀⻔脚本语⾔令⼈⾃豪的地⽅。
* 3. 跨平台，基于Python优良的平台通⽤性，同样的代码可以在不同的平台下编译执⾏，  
  当然这⼀点也可能会涉及到“跨平台同时也会丧失平台优势”等问题。
* 4. 内置与 Python ⾥⾯的，不需要我们单独去安装, ⽅便了很多，⽽且Tkinter也是⼀个
* Python ⾥⾯的标准。
* 5. 缺点就是功能过于简单，对于⼀些很⼤型的应⽤，Tkinter 提供的功能还是太少了。  
  但是 Tkinter会不断进步，⽽且他是Python内置的GUI库，有点嫡系的感觉，我们有理  
  由相信它会越来越棒！
* 6. 还有⼀个缺点是相对于 C++来说的,它的效率问题还是不 算太⾼,对于⼀些对画⾯要求  
  很⾼的，可能效果会有影响。
* 认识 tkinter
* 3
* 认识tkinter
* 1. Tkinter 读的时候,T 和 K 分别发⾳,inter 单独发⾳, 即读⾳如下“tea-key-inter”。该模块
* 在我们安装 python 的时候，就可以选择⼀起安装。
* 2. ⽐如说，我的Python3安装⽬录是: C:\Intel\Python3 ,则进⼊该⽬录，可以发现有 不少  
  的 Tkinter 的影⼦。⽐如在 DLLs ⼦⽬录下，可以看到如下动态链接库⽂件，截图如
* 下(注意 tk86t.dll 是与该模块直接相关的):
* 4
* 认识tkinter
* 3. 在 Lib ⼦⽬录下,我们可以看到⼀个名为 tkinter 的⼦ ⽂件夹,截图如下:
* 4. 进⼊该⽂件夹,可以看到⾥⾯的全部⽂件,⼤多数都 是.py ⽂件,还有两个⽂件夹,第⼀个  
  是和缓存有关的,第 ⼆个是和测试有关的,剩下的都是 py ⽂件,下⾯是截图:
* 5. 这些⽂件分别负责不同的功能,我们下⾯进⼊这个 \_\_init\_\_.py ,它有 163K ⼤⼩,且代  
  码有 3857 ⾏。init.py ⾥有我们平时所频繁使⽤的绝⼤多数类,如果以后⼤家有 什么  
  新问题,可以通过阅读该⽂件来得到答案,因为源代码都给我们了,掌握了源代码就是掌  
  握了第⼀⼿资料,解决问题会更加⽅便⼀些。
* 6. 其实在我们的安装模块中,还有⼀些其他的东⻄有着 tkinter 的影⼦,我就不⼀⼀介绍  
  了。
* 开始前的测试
* 1. 在我们编程之前,先开始⼀个⼩测试把,它只需要两⾏ 代码。
* 5
* 认识tkinter
* 2. 我们在 IDLE 中书写两⾏,第⼀⾏是导⼊ tkinter 这个模 块,注意⼤⼩写,因为 Python 是
* 区分⼤⼩写的,第⼆⾏是 使⽤⼀个测试函数。
* 3. 代码和运⾏效果截图:
* 4. 说明:如果点击 Click me! 按钮,则按钮两边会不断的增加中括号，⽽点击下⾯的 QUIT   
  按钮,则退出程序。
* 5. 现在，让我们进⼊tkinter的世界吧！
* 6
* 第⼀个tkinter程序
* 开始第⼀个 tkinter 程序
* 第⼀个程序
* 1. 我们新建⼀个⽆格式的⽂本⽂件，通常是.txt ⽂件，然后修改名称为 demo1.py，然  
  后⽤⽂本编辑器打开，然后写三⾏代码。
* 2. 这三⾏代码如下：
* from tkinter import \*  
  root = Tk()  
  root.mainloop()
* i. 这是执⾏后的结果：
* 代码解读
* 1. 代码的第⼀⾏是引⼊了该模块，即 from tkinter import \* 所做的⼯作。
* 7
* 第⼀个tkinter程序
* 2. 第⼆⾏是对 Tk 这个类进⾏实例化，即 root 是它的⼀个实例，是代码 root = Tk() 的  
  作⽤，root即是整个窗⼝程序对象。  
  3. 第三⾏进⾏事件循环，即 root.mainloop() 的作⽤，会让窗⼝循环接收下⼀个事件。
* Tk 类解读
* 1. 官⽅对于该类的描述是 “代表绝⼤多数应⽤程序主窗⼝的⾼层空间，它会和⼀个 Tcl  
  解释器相关联” ，它继承⾃两个类，⼀个是 Misc，另⼀个是 Wm，我们这⾥使⽤的  
   mianloop ⽅法，就是从 Misc ⾥⾯继承来的。  
  2. 对于 Tk，我们这么理解，就把它当做⼀个 tkinter 程序 的所有窗⼝的⽼⼤即可，即所  
  有的部件都需要先创建它。  
  3. 我们可以通过阅读这两个类的源代码 来得到⼀些信息，⽐如在 Wm 的源代码发现了  
  ⼀个⽅法，可以修改这个窗⼝的标题，即 wm\_title ⽅法，下⾯我们⽤这个⽅法修改  
  这个窗⼝的标题，或者直接⽤ title 也是可以的。
* 4. 代码如下:
* from tkinter import \*  
  root = Tk()  
  root.wm\_title("这是⼀个Tkinter程序窗⼝标题")  
  #root.title("这是⼀个Tkinter程序窗⼝标题") #这样也是可以的  
  root.mainloop()
* 8
* 第⼀个tkinter程序
* 9
* 初识组件之标签
* 认识新组件---标签
* 组件
* 1. 如果⼤家有过 Windows GUI 编程经验，那么对于 控件 ⼀次 ⼀定不会陌⽣，其实组  
  件和控件是⼀回事，在这⾥我们不加以区分。  
  2. 组件的英⽂表述是 widget ，我们常⻅的按钮，⽂本 框等等都可以当做组件。  
  3. 我们可以为⼀个窗⼝增加若⼲组件来丰富该窗⼝的功能， 我们往往需要指定这个组  
  件的⼤⼩和位置，我们还需要通过编程的⽅式，来指定该组件能完成什么功能。
* Label 类
* 标签
* 1. 所谓 标签 ，就是贴在物品前⾯的⼀个简短的说明， 它⽤于说明⼀些⽂字信息。  
  2. 标签可以说是最简单的窗⼝组件了，它不需要执⾏任何功能，只是⽤来显示信息。
* 3. 下⾯是代码示例:
* from tkinter import \*
* root = Tk()  
  root.wm\_title("这是⼀个Tkinter程序窗⼝标题")
* wl= Label(root, text = "欢迎来到传智公开课")  
  wl.pack()
* root.mainloop()
* i. 演示效果
* 10
* 初识组件之标签
* 代码分析
* 1. 这⾥的 Label 是⼀个类，可以在 \_\_init\_\_.py ⽂件⾥查看 相应的源代码。  
  2. wl 是⼀个 Label 的实例，它有⼀个 text 属性 ，⽤来指定 它的⽂本内容。  
  3. 该类的源代码也⾮常简单，它的源代码截图:
* 11
* 初识组件之标签
* 4. ⼤家可以看到它的标准属性，⽐如有 background ， font ， bitmap ， padx ，  
   relief 等等， 还有 underline 等等。
* 5. 该类有个 pack ⽅法，没错，这个 pack ⽅法我们后⾯会 讲，⼤家可以理解为它的作  
  ⽤就是找个合适的位置进⾏放置即可，即 pack() 之后就选定位置放上去了。
* 6. 这⾥的初始化的时候，需要先指定 root，是说 wl 这个组 件是在 root 这个窗⼝的，不  
  是属于别的窗⼝的，以后我们 有了多窗⼝的应⽤程序，不会混淆。
* Widget 类
* 1. 我们上⾯发现，Label 类是继承⾃ Widget 类的，并且⾃ ⼰只有⼀个⽅法，就是⽤来  
  初始化⾃身的，那么要想彻底 理解 Label，就必须彻底理解 Widget。
* 2. 我们找到了 Widget 的代码，发现它更加短⼩，甚⾄连个 ⽅法都没有，下⾯是
* Widget 的截图:
* 3. 通过这个注释，我们可以理解如下:它是⼀个可以指定 位置并且可以使⽤
* pack ， place 和 grid 来布局管理的窗⼝ 组件。
* 4. 然后我们找到 BaseWidget ，发现它的代码也很短⼩，代 码截图如下:
* 5. 该类继承⾃ Misc ，包含了四个函数，函数的功能通过名 字就可以知道了。⾄于Misc  
  类，我们后⾯会讲解的。
* Misc其实是英⽂ Miscellaneous 的前四个字⺟，杂项、混合体、⼤杂烩的意思。在软件⾥  
  经常可以看到与misc相关的⽂件或函数名，使⽤misc来命名主要是表示该⽂件⽬前还没  
  归类好，不知道将它归到哪个⽅⾯或者放置在哪个地⽅⽐较好，所以暂时⽤misc。
* 12
* 初识组件之标签
* 总结
* 1. 我们这⼀节学习了 Label 这个组件，它的创建使⽤ Label 这个类，实例化的时候⾸  
  先要指定它的⽗窗⼝，然后就是 ⽤字典的⽅式设置⾃⼰的⼀些特征，这些特征上⾯  
  都列出 了，但是没有实例演示。  
  2. 然后使⽤ pack() ⽅法布局上去，关于布局，我们后⾯会详 细讲，这⾥我们统⼀使⽤
* pack() 就可以了。
* 多标签的应⽤程序
* 1. 如果我们想要多个标签，该怎么办呢?  
  2. 很简单，只需要声明多个标签的实例，然后分别 pack 到 窗⼝上即可。  
  3. 下⾯是使⽤了三个标签的代码截图:
* from tkinter import \*
* root = Tk()  
  root.wm\_title("这是⼀个Tkinter程序窗⼝标题")
* w1 = Label(root, text = "欢迎来到传智公开课！", background = "white")  
  w2 = Label(root, text = "好好学习，天天向上！", background = "green")  
  w3 = Label(root, text = "轻松愉快，就选Python！", background = "red")
* w1.pack()  
  w2.pack()  
  w3.pack()
* root.mainloop()
* 13
* 初识组件之标签
* 14
* 组件介绍与按钮
* 组件讲解以及按钮
* tkinter 的核⼼组件
* 1. 在 Python 的 tkinter 中，有 21 个核⼼组件，它们提供 了最基本的功能，虽然简单，  
  因为使⽤频率较⾼，因此特 别重要。
* 2. 这 21 个核⼼组件是 : Toplevel、Label、Button、 Canvas、Checkbutton、Entry、  
  Frame、LabelFrame、 Listbox、Menu、Menubutton、Message、OptionMenu、  
  PaneWindow 、 Radiobutton 、 Scale 、Scrollbar 、Spinbox 、Text、Bitmap、  
  Image。
* 组件的使⽤
* 1. 各个组件都有相应的类，我们可以通过⾯向对象的⽅式 去使⽤它们。  
  2. 这些组件的使⽤也很相似，在实例化这些组件的时候， 第⼀个参数都是⽗窗⼝或者  
  ⽗组件，后⾯跟着的就是该组 件的⼀些属性，⽐如上⾯我们学到的 Label 的 text
* 属性和 background 属性。
* 3. 多个组件的位置控制⽅式也很相似，我们可以⽤ pack ⽅法来进⾏简单的布局，具体  
  的个例我们后⾯再说。  
  4. 组件也会有些⽅法是共⽤的，⽐如 configure ⽅法来设置属性等等。
* 按钮
* 1. 按钮也是⾮常重要的组件， 按钮的重要性在于它可以执⾏相应 的功能。  
  2. 按钮的英⽂表述是 button ，它随处可⻅，当我们单 击的时候，它可以执⾏相应的功  
  能。  
  3. 按钮在 tkinter 中有⼀个类专⻔负责它，叫做 Button ，该类也⾮常简短。
* 15
* 组件介绍与按钮
* 按钮与功能的绑定
* 1. 上⽂说到，按钮可以执⾏相应的功能，这⾥的功能我们 可以理解为⼀个函数，或者  
  这些功能通过相应的函数去实 现。  
  2. 绑定⽅式通常有如下⼏种:第⼀种，在按钮组件被声明 的时候⽤ command 属性声  
  明， command 属性接受⼀个函数名， 注意函数名不要加双引号。第⼆种，使⽤ bind   
  ⽅法，该⽅ 法是 Misc 这个类的⼀个⽅法，下⾯我们仔细讲解。
* 1. 第⼀种⽅法绑定事件
* 我们要完成的功能是我们按下这个按钮的时候，就会在 窗⼝上增加⼀个 Label，它显
* 示“我爱 python”。
* 代码示例:
* from tkinter import \*
* def myLabel():  
   global py, i  
   s = Label(py, text = "轻松愉快，就选Python! 重要的事情说 %d 遍！" % i)  
   s.pack()  
   i+=1 # 每次点击后，i的值会⾃增1
* py = Tk()  
  i = 1 # 全局变量i 初始化为 1
* b = Button(py, text = "Python", command = myLabel)  
  b.pack()
* py.mainloop()
* 运⾏结果：（⼀开始是只有⼀个按钮的，点击⼀下，就会在这个窗⼝上多⼀个标签，下⾯  
  是我点击了三次之后的截图):
* 16
* 组件介绍与按钮
* 代码解读
* 1. 其实很简单，这⾥只需要注意 command 属性后⾯不要加任何的标点符号。  
  2. 这⾥的 myLabel 函数中，使⽤了全局的 py 和 i 变量，需要注意。
* 2. 第⼆种⽅式绑定事件
* 下⾯使⽤第⼆种⽅式来绑定事件，这个事件完成同样的功能。
* 代码示例:
* 17
* 组件介绍与按钮
* from tkinter import \*
* def myLabel(event): # 这⾥多了⼀个event参数⽤来接收  
   global py, i  
   s = Label(py, text = "轻松愉快，就选Python! 重要的事情说 %d 遍！" % i)  
   s.pack()  
   i+=1
* py = Tk()  
  i = 1
* b = Button(py, text = "Python") # command = myLabel ⽊有了  
  b.bind("<Button-1>", myLabel) # 多了这⼀句
* b.pack()  
  py.mainloop()
* 运⾏结果：（和第⼀种⽅法的结果⼀样）
* 代码解读：
* 1. bind这个⽅法是在 Misc 类中的，可以接受三个参数，但是本例中我们只传递了两个  
  参数。  
  2. 第⼀个参数可能对刚使⽤它的⼈来说有点复杂，常⻅的⿏标左键单击如下: <Button-
* 1> （也就是上⾯的代码⽤到的），或者 <Control-Button-1> 等。
* 3. 第⼆个参数可以是⼀个函数名，记住，不要加任何的标 点符号，否则运⾏时会报错  
  的。  
  4. 使⽤ bind 函数的时候，第⼆个参数是⼀个函数名，该函数必须接受⼀个参数，即表  
  示该事件。 这个参数通常⽤ event 来表示，如果我们调⽤的函数不接 受任何参数，
* 则会报错如下 : TypeError: myLabel() takes no arguments (1 given)
* bind的第⼀个参数是⽤字符串包含的事件类型，它采⽤的描述⽅式是: <MODIFIER-MODIFIER-
* TYPE-DETAIL>
* 1. 这⾥的 MODIFIER 即键盘或者⿏标修饰符，它的全部取值如下: Control， Mod2，
* M2， Shift， Mod3， M3， Lock， Mod4， M4， Button1， B1， Mod5， M5  
  Button2， B2， Meta， M， Button3，B3， Alt， Button4， B4， Double，  
  Button5， B5 Triple ， Mod1， M1
* 2. TYPE 表示类型，它的全部取值如下: Activate， Enter， Map， ButtonPress，  
  Button， Expose， Motion， ButtonRelease， FocusIn， MouseWheel，  
  Circulate， FocusOut， Property， Colormap， Gravity Reparent， Configure，
* 18
* 组件介绍与按钮
* KeyPress， Key， Unmap， Deactivate， KeyRelease Visibility， Destroy，Leave
* 3. DETAIL 表示细节，其实也就是对第⼆个参数的⼀些辅助说明。
* 设置属性
* 下⾯我们可以设置⼀些属性，这些东⻄随着⼤量的代码示例，⼤家会接触不少的。
* ⽐如我们可以设置背景⾊，这⾥直接设置属性，代码截 图:
* from tkinter import \*
* py = Tk()
* b1 = Button(py, text = "传智播客")  
  b1["width"] = 20  
  b1["height"] = 4  
  b1.pack()
* b2 = Button(py, text = "Python学院")  
  b2["width"] = 20  
  b2["background"] = "white"  
  b2.pack()
* py.mainloop()
* 运⾏结果：
* 19
* 组件介绍与按钮
* 总结
* 1. 关于按钮，我们重点理解的就是它如何和事件进⾏绑定的。  
  2. 当然，使⽤⼀些其他属性来美化按钮也很重要。  
  3. 下⾯要讲⼀讲布局⽅⾯的东⻄了。
* 20
* Tkinter的三种布局
* 布局的讲解
* 对于任何⼀⻔图形界⾯编程来说，布局都是⾮常重要 的⼀关，它的英⽂翻译叫  
  做“layout”。不管是MFC、Java、还是Qt等图形界⾯编程， 都会有有布局的相关知
* 识。 Python 的 tkinter 也⼀样。
* tkinter的三种布局：
* 1. 其实我们已经接触过 tkinter 的⼀种布局，就是 pack 布局，它⾮常简单，我们不⽤  
  做过多的设置，直接使⽤⼀个 pack 函数就可以了。  
  2. grid 布局: grid 可以理解为⽹格，或者表格，它可以把 界⾯设置为⼏⾏⼏列的⽹格，  
  我们在⽹格⾥插⼊我们想要 的元素。这种布局的好处是不管我们如何拖动窗⼝，相  
  对 位置是不会变化的，⽽且这种布局也超简单。  
  3. place 布局:它直接使⽤死板的位置坐标来布局，这样做 的最⼤的问题在于当我们向窗  
  ⼝添加⼀个新部件的时候， ⼜得重新测⼀遍数据，且我们不能随便地变⼤或者缩⼩  
  窗⼝，否则可能会导致混乱。
* pack 布局
* 1. 我们使⽤ pack 函数的时候，默认先使⽤的放到上⾯，然 后 依次向下排，它会给我们  
  的组件⼀个⾃认为合适的位置 和⼤⼩，这是默认⽅式，也是我们上⾯⼀直采⽤的⽅  
  式。  
  2. pack 函数也可以接受⼏个参数：  
   side 参数指定了它停 靠在哪个⽅向，可以为 LEFT，TOP，RIGHT，  
  BOTTOM，分别代表 左，上，右，下  
   fill 参数可以是 X，Y，BOTH 和 NONE，即在⽔平⽅向填充，竖直⽅向填  
  充，⽔平和竖直⽅向填充和不填充。  
   expand 参数可以是 YES 和 NO，它的 anchor 参数可 以是 N，E，S，W(这⾥  
  的 NESW 分别表示北东南⻄，这⾥分别 表示上右下左)以及他们的组合或者是
* CENTER(表示中 间)。
* ipadx 表示的是内边距的 x ⽅向，它的 ipady 表示 的是内边距的 y ⽅向，padx  
  表示的是外边距的 x ⽅向， pady 表示的是外边距的 y ⽅向。
* pack 的布局实例
* 21
* Tkinter的三种布局
* 1. 根据上⾯的介绍，我们可以做出如下布局的样⼦:
* 2. 按理说做的这么复杂本身没什么意思,只是想让⼤家看 ⼀下其实 pack 也可以完成相对  
  复杂的布局,它的源代码如下:
* from tkinter import \*
* root = Tk()  
  Button(root, text = "A").pack(side = LEFT, expand =YES, fill = Y)  
  Button(root, text = "B").pack(side = TOP, expand = YES, fill = BOTH)  
  Button(root, text = "C").pack(side = RIGHT, expand = YES, fill = NONE, anchor = NE)  
  Button(root, text = "D").pack(side = LEFT, expand = NO, fill = Y)  
  Button(root, text = "E").pack(side = TOP, expand = NO, fill = Y)  
  Button(root, text = "F").pack(side = BOTTOM, expand = YES)  
  Button(root, text = "G").pack(anchor = SE)
* root.mainloop()
* grid 布局
* 1. 由于我们的程序⼤多数都是矩形,因此特别适合于⽹格 布局,也就是 grid 布局。  
  2. 使⽤ grid 布局的时候,我们使⽤ grid 函数,在⾥⾯指 定两个参数,⽤ row 表示⾏,
* 22
* Tkinter的三种布局
* ⽤ column 表示列，注意的是 row 和 column 的编号都从 0 开始。  
  3. grid 函数还有个 sticky 参数,它可以⽤ N,S,W,E 表示 上,下,左,右 , 它决定了这个组件  
  是从哪个⽅向开始的, 下⾯的例⼦可以很好的解释这⼀点。  
  4. grid 布局直接⽤后⾯的⾏和列的数字来指定了它位于哪个位置,⽽不必使⽤其他参数。  
  5. grid 函数也⽀持诸如 ipadx,ipady,padx,pady ,它们的意思和 pack 函数是⼀样的,默认  
  边距是 0。  
  6. 它还⽀持参数⽐如 rowspan ,表示跨越的⾏数, columnspan 表示跨越的列数。  
  7. 它还有⼀些属性,可以在以后我们的demo中慢慢使⽤来 看出其重要性。
* 账号登陆例⼦的截图
* 1. 界⾯如下：
* 2. 代码如下：
* 23
* Tkinter的三种布局
* from tkinter import \*
* py = Tk()  
  Label(py, text = "账号：").grid(row = 0, sticky = W)  
  Entry(py).grid(row = 0, column = 1, sticky = E)
* Label(py, text = "密码：").grid(row = 1, sticky = W)  
  Entry(py).grid(row = 1, column = 1, sticky = E)
* Button(py, text = "登录").grid(row = 2, column = 1, sticky = E)
* py.mainloop()
* 3. 代码说明: 代码⾮常简单，参考grid布局介绍理解，其中 Entry 表示“输⼊框”。
* place 布局
* 1. 关于 place 布局,这个的⼏何管理器组织放置在⼀个特定的位置  
  2. 它使⽤ place 函数,它分为 绝对布局 和 相对布局 ，绝对布局使⽤ x 和 y 参数,相对布局
* 使⽤ relx,rely, relheight 和 relwidth 参数。
* 3. 该⽅法⽤的极少，⽽且极度不推荐⼤家⽤，这⾥就不详细说明了。
* 总结
* 1. place 不推荐⽤，pack 和 grid 布局更常⽤⼀些。
* 2. 但是 pack 和 grid 不能同时⽤。⽽且通常对于较为复杂点的界⾯, 还是建议⼤家⽤  
  gird；如果布局相对简单，使⽤pack 也很不错。
* 24
* 事件与绑定
* 事件
* 如果⼤家学习过 Windows 编程，经常会听到“Windows 是 基于消息的”，⽽绝  
  ⼤多数界⾯编程也都会涉及到“事件”。
* 我们的很多操作，⽐如我们点击了⼀下⿏标，这就是⼀个事件，⽽操作系统会  
  根据我们的相应的事件产⽣相应的 消息，操作系统把消息传递给我们的应⽤程  
  序，然后我们 的应⽤程序根据操作系统传⼊的数据执⾏相应的命令。
* 事件是我们触发的，消息是操作系统根据我们的事件产⽣的，我们通常对于“消  
  息”并不多关注，我们重视的是 “事件”。
* 事件的英⽂表述是 event 。
* 事件及其绑定
* 1. 其实，我们在按钮那⼀节就接触到了事件的绑定，使⽤ 的函数是 bind 。
* 2. bind 函数的调⽤规则: 窗体对象 . bind(事件类型，回调函数)
* 3. 所谓的“回调函数”，就是这个函数我们不⽤去调⽤它， 当相应的事件发⽣的时候，它  
  会⾃动取调⽤。⽐如当我们 的按钮被按下的时候，它会被⾃动调⽤。
* 常⽤的事件
* 1. 我们在使⽤ bind 函数的时候，它的第⼀个参数就是事件 的类型了。
* 2. <Button-1> 表示⿏标左键单击，其中的 1 换成 3 表示右 键被单击，为 2 的时候表示  
  ⿏标中键，感觉不算常⽤。
* 3. <KeyPress-A> 表示 A 键被按下，其中的 A 可以换成其他 的键位。
* 4. <Control-V> 表示按下的是 Ctrl 和 V 键，V 可以换成其他 键位。  
  5. <F1> 表示按下的是 F1 键，对于 Fn 系列的，都可以随便 换。
* 再看绑定
* 25
* 事件与绑定
* 1. 事件不仅可以与 Button 绑定，我们之前看过源代码，发 现 bind 函数是定义在 Misc  
  类⾥⾯的，也就是说，这个 bind 可以被绝⼤多数组件类所使⽤。
* 2. 也就是说，我们可以让“标签”来模拟“按钮”的作⽤。
* 3. 因为标签是 Label 类，⽽ Label 类继承⾃ Widget，⽽ Widget 继承⾃ BaseWidget，  
  ⽽ Basewidget 继承⾃ Misc。
* 4. 其实不仅是标签可以模拟 button，任何组件都可以模拟 它，只是那么有⽤。
* 标签模拟按钮
* 1. 代码如下:
* from tkinter import \*
* def myLabel(event):  
   global py  
   s = Label(py, text = "传智播客Python学院，好好学习天天向上！")  
   s.pack()
* py = Tk()
* n = Label(py, text = "我可不是真Button哟~")  
  n.bind("<Button-1>", myLabel)  
  n.pack()
* py.mainloop()
* 2. 运⾏效果(下⾯是点击了四次之后的效果图):
* 26
* 事件与绑定
* 关于 bind 函数
* 1. 关于 bind 函数，还有两个版本的，不能说⾼级低级，只是使⽤的⽅⾯不同。
* 2. 可以在全程序级别的绑定，使⽤ bind\_all ，它的参数类 型和 bind ⼀样，它通常⽤  
  于全局的快捷键，⽐如 F1 通常 是⽤来打开帮助⽂档。
* 3. 还可以绑定某些类别，使⽤ bind\_class ，它接受三个参数， 第⼀个参数是类名，第  
  ⼆个参数是事件类型，第三个参数 是相应的操作，⽐如 w.bind\_class(“Entry”,  
  “<Control-V>”, my\_paste) 。它就是绑定了所有的所有的输⼊框的 Ctrl+V 表示粘贴。
* 解除绑定
* 1. 接触绑定我们使⽤ unbind ⽅法，它和 bind 的使⽤很相似。
* 2. 不过 unbind ⽅法只需要⼀个参数就可以了，它只需要解除绑定的事件类型，因为它  
  会解除该绑定事件类型的所有回调函数。
* 27
* 事件与绑定
* 28
* 输⼊框以及登录程序
* 输⼊框以及⼀个登录程序
* 输⼊框的重要性
* 应⽤程序要取得⽤户的信息，输⼊框是必不可少的，虽然执⾏命令可以使⽤按钮，  
  但是总不能让⽤户⼀直点击按钮把。
* Entry
* 1. 输⼊框是 Entry，可以阅读它的源代码。
* 2. 源代码截图:
* 3. ⽐较重要的也就是get 函数，get 函数使⽤的时候不需要任何参数，它的返回值就是  
  该输⼊框的内容。
* 密码框
* 29
* 输⼊框以及登录程序
* 1. 其实密码框和输⼊框基本是⼀样的，都是向⾥⾯输⼊信 息⽤的。
* 2. 如果要说不⼀样，也就⼀个地⽅不⼀样:密码框需要输 ⼊的信息的显示字符⽐较单  
  ⼀。
* 3. ⽐如 e 是⼀个输⼊框，我们可以设置它的 show 属性让它 变成⼀个密码框，
* 即 e['show'] = '\*' 就可以了。
* ⼩型登录程序
* 1. 下⾯是⼀个⼩型登录程序，它的⽤户名是 itcast ，密码是 python ，如果输⼊正确，那  
  么点击“登录”按钮之后，就会显 示“登录成功”，如果输⼊不符合，那么就会显示“⽤户 名  
  或者密码错误”，并且清空两个输⼊框。
* 2. 运⾏结果：
* \* 输⼊正确的⽤户名和密码，显示`登录成功`:
* ![](../images/6.0\_1.png)
* \* 输⼊错误的⽤户名或密码，显示`⽤户名错误`或`密码错误`:
* ![](../images/6.0\_2.png)
* 3. 代码演示：
* 30
* 输⼊框以及登录程序
* from tkinter import \*
* def reg():  
   myAccount = a\_entry.get() # 获取⽤户输⼊的⽤户名  
   myPassword = p\_entry.get() # 获取⽤户输⼊的密码  
   a\_len = len(myAccount) # 获取输⼊的⽤户名⻓度  
   p\_len = len(myPassword) # 获取输⼊的密码⻓度
* if myAccount == "itcast" and myPassword == "python":  
   msg\_label["text"] = "登录成功" # ⽤户名和密码全部正确 elif myAccount == "itca  
  st" and myPassword != "python":  
   msg\_label["text"] = "密码错误" # ⽤户名正确密码错误  
   p\_entry.delete(0, p\_len)  
   else:  
   msg\_label["text"] = "⽤户名错误" # ⽤户名错误  
   a\_entry.delete(0, a\_len)  
   p\_entry.delete(0, p\_len)
* root = Tk()
* # ⽤户名
* a\_label = Label(root, text = "⽤户名：")  
  a\_label.grid(row = 0, column = 0, sticky = W)  
  a\_entry = Entry(root)  
  a\_entry.grid(row = 0, column = 1, sticky = E)
* # 密码  
  p\_label = Label(root, text = "密码：")  
  p\_label.grid(row = 1, column = 0, sticky = W)  
  p\_entry = Entry(root)  
  p\_entry["show"] = "\*" # 密码显示为 \*  
  p\_entry.grid(row = 1, column = 1, sticky = E)
* # 登录按钮
* btn = Button(root, text = "登录", command = reg)  
  btn.grid(row = 2, column = 1, sticky = E)
* # 提示信息
* msg\_label = Label(root, text = "")  
  msg\_label.grid(row = 3)
* root.mainloop()
* 31
* 菜单与选择按钮
* 菜单
* 菜单简介
* 菜单的信息量是⾮常巨⼤的，由于菜单⼜可以有⼦菜单，因此菜单的信息量⾮  
  常⼤。  
  菜单的分类也较多，通常可以分为下拉菜单、弹出菜单 等等。
* 添加顶层菜单
* 1. 我们可以使⽤ Menu 类来新建⼀个菜单， Menu 和其他的组件⼀样，第⼀个是  
   parent ，这⾥通常可以为窗⼝。
* 2. 然后我们可以⽤ add\_commmand ⽅法来为它添加菜单项， 如果该菜单是顶层菜单，则  
  添加的菜单项依次向右添加。 如果该菜单时顶层菜单的⼀个菜单项，则它添加的是  
  下拉 菜单的菜单项。
* 3. add\_command 中的参数常⽤的有 label 属性，⽤来指定的 是菜单项的名称， command   
  属性⽤来指定被点击的时候调⽤ 的⽅法， acceletor 属性指定的是快捷  
  键， underline 属性 是是否拥有下划线。
* 4. 最后可以⽤窗⼝的 menu 属性指定我们使⽤哪⼀个作为它 的顶层菜单。
* 代码演练
* 1. 这⾥只是做出了顶级菜单，它们四个是并列的⼀⾏，并 没有实现什么功能，效果图如
* 下:
* 32
* 菜单与选择按钮
* 2. 下⾯是代码演示:
* from tkinter import \*
* root = Tk()  
  menuBar = Menu(root)
* for item in ["⽂件", "编辑", "视图", "关于"]:  
   menuBar.add\_command(label = item)
* root["menu"] = menuBar
* root.mainloop()
* 3. 可以看到，⾮常简单的⽅式，我们使⽤ add\_command 来 添加菜单项即可。
* 有⼦菜单的情况
* 1. 如果有⼦菜单，则情况稍微复杂点，这个时候，我们需 要使⽤ add\_cascade ，  
  cascade 可以理解为“级联”，即它 的作⽤只是为了引出后⾯的菜单。
* 33
* 菜单与选择按钮
* 2. add\_cascade 的⼀个很重要的属性就是 menu 属性，它指 明了要把那个菜单级联到该  
  菜单项上，当然，还必不可少 的就是 label 属性，⽤于指定该菜单项的名称。
* 3. 我们先新建⼀个 Menu 的实例，然后使⽤ add\_command 来 添加菜单项，这样等该菜单  
  建⽴完毕，我们要把它作为另 ⼀个菜单项的⼦菜单，就需要使⽤ add\_cascade ⽅法。
* 代码演练
* 1. 下⾯我们使⽤了下拉菜单，这也是我们通常使⽤的菜单 形式，即⼀⾏菜单项，点击  
  之后会产⽣⼀个下拉菜单。
* 2. 效果截图⼀:
* 34
* 菜单与选择按钮
* 3. 截图效果⼆:
* 4. 截图效果三:
* 5. 代码演示:
* 35
* 菜单与选择按钮
* from tkinter import \*
* root = Tk()  
  menuBar = Menu(root)
* fMenu = Menu(menuBar) # ⽂件  
  for item in ["新建", "打开", "保存", "另存为", "退出"]:  
   fMenu.add\_command(label = item)
* eMenu = Menu(menuBar) # 编辑  
  for item in ["复制", "粘贴", "剪切", "撤销"]:  
   eMenu.add\_command(label = item)
* vMenu = Menu(menuBar) # 视图  
  for item in ["默认视图", "全屏模式", "显示/隐藏菜单"]:  
   vMenu.add\_command(label = item)
* aMenu = Menu(menuBar) # 关于  
  for item in ["版权信息", "帮助⽂档"]:  
   aMenu.add\_command(label = item)
* menuBar.add\_cascade(label = "⽂件", menu = fMenu)  
  menuBar.add\_cascade(label = "编辑", menu = eMenu)  
  menuBar.add\_cascade(label = "视图", menu = vMenu)  
  menuBar.add\_cascade(label = "关于", menu = aMenu)
* root["menu"] = menuBar
* root.mainloop()
* 这⾥要注意的是我们可以先把⼦菜单做好，然后再做上层菜单。
* 弹出菜单
* 1. 弹出菜单⼜叫“上下⽂菜单”，也叫“右键菜单”，它 通常是⿏标单击右键产⽣的菜单，  
  因此会有“右键菜单” 的说法。
* 2. 其实很多界⾯库⾥⾯都是给出了弹出菜单的简单的制作 ⽅法的，但是 tkinter ⾥⾯我  
  们却只能使⽤⽐较原始的事 件绑定的⽅式去做。
* 3. ⼤体思路就是:我们先新建⼀个菜单，然后向菜单项中 添加各种功能，最后我们监听  
  ⿏标右键消息，如果是⿏标 右键被单击，此时可以根据需要判断下⿏标位置来确定  
  是 哪个弹出菜单被弹出，然后使⽤ Menu 类的 pop ⽅法来弹出 菜单。
* 4. ⼤体思路就是如此，⾄于具体的细节，让我们到代码实 战中⼀探究竟。
* 36
* 菜单与选择按钮
* 5. Menu 类⾥⾯有⼀个 post ⽅法，它接收两个参数，即 x 和 y 坐标，它会在相应的  
  位置弹出菜单。
* 6. 还记得⽤ bind ⽅法来绑定事件吗?⽽且要记得⿏标右键 是⽤的 <Button-3>
* 代码演练
* 1. 界⾯效果(该右键菜单中如果点击 python 选项，则会新 建⼀个标签，标签内容是“我
* 的Python课程”):
* 2. 代码演示:
* 37
* 菜单与选择按钮
* from tkinter import \*
* def myLabel():  
   global root  
   Label(root, text = "我的Python课程").pack() # 点击Python后添加这个标签
* root = Tk()  
  menuBar = Menu(root)
* for each in ["C/C++", "JavaEE", "Android", "PHP", "UI设计", "iOS", "前端与移动开发", "⽹络营  
  销", "云计算"]:  
   menuBar.add\_command(label = each)
* menuBar.add\_command(label = "Python", command = myLabel)
* def pop(event):  
   menuBar.post(event.x\_root, event.y\_root)
* root.bind("<Button-3>", pop) # ⿏标右击绑定
* root.mainloop()
* 此处利⽤了 Menu 的 post ⽅法，还有 bind ⽅法，⼀定要 记住⿏标右键的事件名称，这  
  些⽤多了之后⾃然能记住。
* 分割线
* 1. 有时候，⼀个菜单项的各个菜单可能并不是⼀个类型， 有可能是两种类型，在它们  
  中间可以插⼀个分割线来界定 界限。  
  2. 插⼊分割线和插⼊正常的菜单项操作很相似，只是使⽤ 的⽅法是 add\_separator，该  
  ⽅法⽆需参数。  
  3. 插⼊分割线效果示例(在列表中插⼊了⼀个分割线):
* 38
* 菜单与选择按钮
* 4. 插⼊分割线代码示例:
* from tkinter import \*
* root = Tk()  
  menuBar = Menu(root)  
  sonMenu = Menu(menuBar)
* for each in ["C/C++", "Python", "JavaEE", "Android", "iOS", "云计算"]:  
   sonMenu.add\_command(label = each)
* sonMenu.add\_separator() # 插⼊分割线
* for each in ["PHP", "UI设计", "前端与移动开发", "⽹络营销"]:  
   sonMenu.add\_command(label = each)
* menuBar.add\_cascade(label = "传智学科列表", menu = sonMenu)  
  root["menu"] = menuBar
* root.mainloop()
* radiobutton 与 checkbutton
* 1. radiobutton 是指“单选按钮”, checkbutton 是指“复选按钮”。
* 2. 单选按钮与复选按钮是相对来说的，即在这⼀组中，单选按钮中只能有⼀个是被选定  
  的，即⼀个⼈的性别是⼥的， 就⼀定不是男的，但是复选按钮对应的情况则是可以  
  有多个同时被选定，⽐如⼀个⼈即可以喜欢篮球，也可以喜欢 ⾜球，还有可能喜欢
* 39
* 菜单与选择按钮
* 乒乓球，等等。
* 3. 在菜单中，也有类似的概念，即 单选菜单 和 复选菜单 。 它们分别⽤ add\_radiobutton   
  和 add\_checkbutton 来分别 添加。
* 4. 这两种菜单都是如果⼀旦被选定，那么前⾯会有⼀个类 似于对号的标记出现，
* checkbutton 可以多个同时被选定， 但是 radiobutton 却只能被选定⼀个，即这个被
* 选定了， 会⾃动取消前⼀个的选定。
* 代码与效果截图
* 效果图(上⾯是必须单选的，下⾯则是可以复选):
* 代码演示:
* 40
* 菜单与选择按钮
* from tkinter import \*
* root = Tk()  
  menuBar = Menu(root)  
  sonMenu = Menu(menuBar)
* for each in ["C/C++", "Python", "JavaEE", "Android", "iOS", "云计算"]:  
   sonMenu.add\_radiobutton(label = each) # add\_command() 改成 add\_radiobutton()
* sonMenu.add\_separator() # 插⼊分割线
* for each in ["PHP", "UI设计", "前端与移动开发", "⽹络营销"]:  
   sonMenu.add\_checkbutton(label = each) # add\_command() 改成 add\_checkbutton()
* menuBar.add\_cascade(label = "传智学科列表", menu = sonMenu)  
  root["menu"] = menuBar
* root.mainloop()
* 41
* 对话框和消息框
* 对话框和消息框
* 对话框
* 1. 关于对话框，tkinter 有它⾃⼰的布局，在 Python 的安 装⽬录下的 Lib ⽂件夹的  
  tkinter ⼦⽂件夹下，有个 dialog.py，它只有 2KB，它就是我们要介绍的重点。
* 2. 开该⽂件，只有五⼗⾏，还包括⽤来测试的代码，⾥ ⾯只有⼀个 Dialog 类，它继承  
  ⾃ Widget，它的⽅法也很简 单，基本上也就⼀个init⽅法有实质性内容，下⾯的  
  \_test 是⽤来测试的。
* 3. 该类可⽤的属性很少，基本上真正⽐较好⽤的也就⼀个 num 属性了，它⽤于返回⽤  
  户的点击。
* 4. 我们可以通过它给出的例⼦来模仿写出⼀个对话框，注 意它的 title 属性，text 属
* 性，strings 属性。
* 5. 下⾯是该⽂件的截图:
* 42
* 对话框和消息框
* 调查Demo
* 1. 下⾯是界⾯:
* 43
* 对话框和消息框
* 2. 代码演示:
* from tkinter.dialog import \*  
  from tkinter import \*
* def myDialog():  
   d = Dialog(None, title = "Python调查", text = "喜欢Python吗？", bitmap = DIALOG\_ICON, d  
  efault = 0, strings = ("喜欢", "很喜欢", "⾮常喜欢"))  
   print(d.num)
* btn\_begin = Button(None, text = "Python调查", command = myDialog)  
  btn\_begin.pack()
* btn\_quit = Button(None, text = "关闭", command = btn\_begin.quit)   
  btn\_quit.pack()
* btn\_begin.mainloop()
* 3. 代码说明:
* 在新建⼀个对话框的时候，title 属性，text 属性，bitmap 属性等等都是不可缺少的，否则  
  会报错，我 第⼀次写的时候就报错了，后来看了下源代码，发现这些 都必须⾃⼰填写，  
  因为没有设置默认值。
* 其他对话框
* 44
* 对话框和消息框
* 1. 其实我们也看到了，由于源代码中给出的关于对话框的 代码过于简短，真正⽤起来  
  并不⽅便。
* 2. 于是，根据使⽤频率，源代码⼜给出了⼏个标准对话框， ⽐如 simpledialog(简单对
* 话框)，commondialog(⼀般 对话框)，filedialog(⽂件对话框)，其实 colorchooser 也
* 算对话框的内容。
* 3. 我看了下它们的源代码，但是考虑到我们的程序的主要 ⽤途，所以就不在此节深⼊  
  介绍了，具体的⼤家可以深⼊ 源代码⼀探究竟。
* 消息框
* 1. 和上⾯的对话框⼀样，在同⼀⽬录下会看到有个 messagebox.py，打开⼀看，发现  
  也很简短，只有⼀百多⾏。
* 2. 且⾥⾯主要是⼀些函数，这些函数的参数都给了默认值， 我们只需要填充那些我们  
  感兴趣的值即可。
* 45
* 对话框和消息框
* 3. 下⾯是源代码中该⽂件的主要函数截图:
* 4. 使⽤消息框的效果图:
* 5. 使⽤消息框的代码截图:
* 46
* 对话框和消息框
* from tkinter.messagebox import \*  
  showinfo(title = "传智播客Python学院", message = "好好学习，天天向上！")
* 47
* 常⽤控件的介绍
* 常⽤控件介绍
* 控件
* 1. 在前⾯⼏节中，介绍了⼏个简单的控件，⽐如标签，⽐ 如按钮，⽐如输⼊框等  
  等。  
  2. 说实话，标准的 tkinter 中给出的控件数⽬并不多，只 有 21 个，虽然后来的  
  ttk 中⼜增加了⼏个，但是还是跟不 上需要。关于 ttk 后⾯会介绍。  
  3. 但是这⾥还是介绍下常⽤的控件把，毕竟这些控件还是 蛮重要的。
* 复选按钮
* 1. 复选按钮就是 Checkbutton 类，它的实例化和 Button 很 相似。
* 2. 既然是按钮，那就可以有 command 属性，该属性可以对 应到⼀个函数上去来执⾏某  
  些功能。  
  3. 复选框通常是⽤来选择信息的时候的⼀种选择，它前⾯ 有个⼩正⽅形的⽅块，如果  
  选中则有⼀个对号，也可以再 次点击以取消该对号来取消选中。
* 复选框代码实例
* 1. 该实例，使⽤了两个复选框，点击那个复选框，如果处 于选中状态，则在下⾯的标  
  签中显示被选中的字样，如果 没有被选中，则显示未被选中的字样。  
  2. 效果截图⼀:
* 48
* 常⽤控件的介绍
* 3. 效果截图⼆:
* 49
* 常⽤控件的介绍
* 4. 代码示例:
* 50
* 常⽤控件的介绍
* from tkinter import \*
* timeA = 0  
  timeB = 0
* def funcA():  
   global lab, btnA, timeA  
   if timeA % 2 == 0:  
   timeA += 1  
   lab["text"] = "Python学科被选中"  
   else:  
   timeA += 1  
   lab["text"] = "Python学科被取消"
* def funcB():  
   global lab, btnB, timeB  
   if timeB % 2 == 0:  
   timeB += 1  
   lab["text"] = "C++学科被选中"  
   else:  
   timeB += 1  
   lab["text"] = "C++学科被取消"
* root = Tk()  
  btnA = Checkbutton(root, text = "Python学科", command = funcA)  
  btnA.pack()
* btnB = Checkbutton(root, text = "C++学科", command = funcB)  
  btnB.pack()
* lab = Label(root, text = " ")  
  lab.pack()
* root.mainloop()
* 代码也很简单，⽤于控制次数的 timeA 和 timeB
* 单选框
* 1. 单选框和复选框⾮常相似，只是把 Checkbutton 换成 Radiobutton。
* 2. 我就不代码示例了，因为实在是太相似且简单了。
* 3. 可以参考单选框类源代码。
* ⽂本域
* 51
* 常⽤控件的介绍
* 1. 所谓⽂本域，也就是⽂本，其实它可以看做⼀个⼤型的⽂本框，它的属性也更多⼀  
  些。
* 2. ⽂本域效果展示:
* 3. 代码示例:
* from tkinter import \*
* root = Tk()  
  t = Text(root, width = 50, height = 30)  
  t.pack()
* root.mainloop()
* Canvas
* 1. canvas 直接翻译就是“帆布”，其实可以理解为⼀个画布， ⽤于在上⾯绘制图形。
* 2. 它也是⼀个控件，但是在它上⾯的区域，可以⽤于⼿⼯ 绘制，对于这个⼿⼯绘图的  
  东⻄，说实话，本⼈并不感兴 趣，但是该类有很⼤， 供了很多绘制的函数。
* 3. 后⾯的项⽬会⽤到它。
* Toplevel
* 1. ⼀个 toplevel 可以理解为⼀个新的窗⼝，它是⼀个顶层 窗⼝。
* 52
* 常⽤控件的介绍
* 2. 新建⼀个 toplevel 很简单，只需要实例化该类即可，这 也是创建多窗⼝ 应⽤的⼀个  
  途径。
* 3. 下⾯是⼀个多窗⼝应⽤程序的截图:
* 4. 代码截图:
* from tkinter import \*
* root = Tk()  
  root.title("我是root窗⼝")  
  lr = Label(root, text = "我属于root")  
  lr.pack()
* top = Toplevel(root, width = 30, height = 20)  
  top.title("我是toplevel窗⼝")  
  lt = Label(top, text = "我属于toplevel")  
  lt.pack()
* root.mainloop()
* 53
* Canvas与图形绘制
* ⼿绘图形
* 窗⼝重绘
* 1. 第⼀次认识到⼿绘图形的重要性还是在学习 MFC 的时候, 因为 MFC ⾃带的绘图功能  
  实在过于丑陋,我们可以重绘标题栏、菜单栏、最⼩化按钮、最⼤化按钮、关闭按钮  
  等窗 ⼝组件来让窗⼝得到美化,除了这些,还可以⼿绘按钮, 后来出了⼀⻔技术, 叫  
  做“Direct UI”,也是这种思想的进⼀步发扬光⼤把。
* 2. 但是 tkinter 没有这些⽅⾯的接⼝,我也深感遗憾。但是 tkinter 有⼀个绘图功能的组件,  
  即 Canvas,翻译成汉 语即“帆布”,可以理解为“画布”,即⽤于绘制图形。
* canvas
* 1. 下⾯是该类的部分截图:
* 2. 其实我们主要的也就是⽤上⾯的这些绘图函数来进⾏⼿ ⼯的绘制⼀些东⻄。
* 54
* Canvas与图形绘制
* 简单示例
* 1. 我们第⼀个示例随便绘制了,先把背景⽤ rgb 格式刷成 蓝⾊,然后画⼀条线,然后写⼏个  
  字。
* 2. 代码截图:
* from tkinter import \*
* root = Tk()  
  root.title("随便绘制⼀下下")
* canv = Canvas(root, width = 400, height = 300, bg = "pink")  
  canv.create\_line((0,0), (200,200), width = 8)  
  canv.create\_text(300, 30, text = "Python学院")  
  canv.pack()
* root.mainloop()
* 1. 效果图如下:
* 55
* Canvas与图形绘制
* 中国象棋棋盘
* 1. 其实⼿绘可以做很多事,很多东⻄都可以⼿绘,只不过 有些东⻄确实⼿绘挺累的。
* 2. 下⾯是⼿绘了⼀个中国象棋棋盘的截图(关于中国象棋, 我还是挺喜欢玩的):
* 3. 其实没啥技术难度,下⾯是代码截图,不过我还是希望 读者先亲⾃动⼿做⼀个,毕竟不
* 难:
* 56
* Canvas与图形绘制
* from tkinter import \*
* root = Tk()  
  root.title("中国象棋棋盘⼿绘")
* canv = Canvas(root, width = 400, height = 450)  
  canv.create\_line((0,2), (400,2), width = 2)
* for i in range(10):  
   canv.create\_line((0, i \* 50), (400, i \* 50), width = 2)  
  canv.create\_line((3,0), (3,450), width = 2)
* for i in range(8):  
   canv.create\_line((i \* 50,0), (i \* 50,200), width = 2)
* for i in range(8):  
   canv.create\_line((i \* 50,250), (i \* 50,450), width = 2)
* canv.create\_line((397,0), (397,450), width = 2)  
  canv.create\_line((150,0), (250,100), width = 2)  
  canv.create\_line((150,100), (250,0), width = 2)  
  canv.create\_line((150,450), (250,350), width = 2)  
  canv.create\_line((150,350), (250,450), width = 2)  
  canv.create\_text(110, 220, text = "汉界")  
  canv.create\_text(290, 220, text = "楚河")  
  canv.pack()
* root.mainloop()
* 57
* 窗⼝美化效果
* 窗⼝的⼀些美化
* 美化
* 1. GUI 编程⾃从出现的那⼀天开始，就凸显除了强劲的魅 ⼒ ，因为它对⽤户来  
  说实在是太⽅便了。
* 2. 但是 GUI 编程尤其的繁琐且啰嗦，因为界⾯编写确实令 ⼈头疼，可能初学者  
  对于写界⾯会很兴奋，但是写⼀段时 间就会感觉很烦了。
* 3. 但是如果你致⼒于创造更加简洁⼤⽅的界⾯，那么还是努⼒吧，毕竟，清爽的  
  界⾯谁都喜欢⽤。
* 窗⼝的⼀些控制
* 1. 我们可以⽤ title 函数来修改窗⼝的默认标题，该参数 直接接收⼀个字符串参数。
* 2. 我们还可以⽤ geometry 函数来控制窗⼝⼤⼩，它接受⼀ 个字符串类型的参数，但是
* 它的格式很严格，是这样的 ‘width x height + xoffset + yoffset’，⽐如我写成  
  '300x280+150+200' 就是⼀个合格的参数，注意的是这⾥ 的 x 就是 xyz 的 x ，不要
* 擅⾃改成 \* 。
* 3. 如果想去除边框，还可以⽤ overrideredirect ⽅法，只 需要把参数设置为 1 即可。 4.  
  如果想修改标题栏的默认图标，可以⽤ iconbitmap ⽅法 或者是 wm\_iconbitmap ⽅
* 法，它接受⼀个 ico ⽂件的⽂件 名。
* ⼀些效果图
* 1. 更改图标之后的效果(由于我⽤gitbook做的课件，所以顺⼿⽤了gitbook的图标): 注意  
  左上⻆的图标，不是红⾊的 tk 字样，⽽是新的的图⽚:
* 58
* 窗⼝美化效果
* 2. 去除了边框之后的效果(感觉有点吓⼈，其实这种界⾯ ⽅式很有名，⽤于制作异形窗  
  ⼝很有帮助，在此我就不深 ⼊讲解了):
* 3. 最原始的窗⼝截图:
* 59
* 窗⼝美化效果
* 4. 代码示例：
* from tkinter import \*
* root = Tk()  
  root.geometry("300x280+150+200")  
  root.title("窗⼝美化")  
  root.overrideredirect(1)  
  root.iconbitmap("test.ico")
* root.mainloop()
* 60
* ttk介绍以及总结
* ttk 以及第⼀部分的总结
* ttk 的产⽣
* ttk 可以理解为“Tk tookit”，在 tkinter 的⽬录⾥可 以找到，它是最后⼀个⽂件，
* ⼤⼩有 56KB，还算⼀个⽐较 ⼤的⽂件的，毕竟我们的 tkinter 的init.py 也只有
* 158KB。
* ttk 的出现和⼤家对 tkinter 的不满有关，因为 tkinter 是⼀个跨平台的界⾯库，  
  但是，这也就意味着它丧失了平台优势。为了让它在 Windows 平台下运⾏更  
  像 Windows，其实是让它⽀持不同的⻛格，于是出现了这么⼀个⽂件，⽤于弥
* 补 tkinter 的不⾜。
* 很重要的⼀点就是对标准组件的⽀持，ttk ⽀持原来的 21 个组件中的 11 个，
* 并且，它引⼊了六个新的组件: combobox，notebook，progressbar，  
  separator，sizegrip，tr eeview。
* 效果对⽐
* 1. 下⾯是同样的⼀个按钮，效果图如下(上⾯为导⼊了 ttk， 下⾯没有导⼊ ttk，我的平台
* 是 Win10):
* 61
* ttk介绍以及总结
* 2. 使⽤ ttk 的代码截图:
* from tkinter import \*  
  from tkinter.ttk import \*
* root = Tk()  
  btn = Button(root, text = "我⽤了ttk了！")  
  btn.pack()
* root.mainloop()
* 3. 不使⽤ ttf 的代码截图:
* from tkinter import \*
* root = Tk()  
  btn = Button(root, text = "我没有⽤ttk...")  
  btn.pack()
* root.mainloop()
* 其实⼤家也可以看到,其实很简单,使⽤ ttf 只需要在 前⾯加⼀⾏代码即可了“`from
* tkinter.ttk import `”即可。\*
* 62
* 桌⾯应⽤项⽬：⼩球弹弹弹
* ⼩球弹弹弹
* 0. 导⼊框架
* from random import randint  
  from tkinter import \*  
  from tkinter.ttk import \*
* 1. 搭建框架
* #获取屏幕参数，绑定相关事件，以及启动⽣成⼩球的迭代器
* class MoreBalls:  
   '''
* 获取屏幕参数，绑定相关事件，以及启动⽣成⼩球的迭代器
* @ num: 从\_\_main\_\_⾥接受⼩球的数量  
   '''
* # 定义⼀个列表，⽤来存储所有的球对象
* balls = []
* # num 是球的数量  
   def \_\_init\_\_(self, num):
* # 创建⼀个Tk()窗⼝实例  
   self.root = Tk()
* # 1. w 和 h 分别获取了屏幕分辨率的宽度和⾼度
* scrnw = self.root.winfo\_screenwidth()  
   scrnh = self.root.winfo\_screenheight()
* self.root.title("⼩球弹弹弹")
* # 去除窗⼝边框和任务栏显示
* self.root.overrideredirect(1)  
   #self.root.iconbitmap("test.ico")
* # 设置窗⼝的透明度，0-1 之间，1是不透明，0是全透明。
* self.root.attributes("-alpha", 0.4)
* # 绑定退出事件（键盘任意键、⿏标任意点击、⿏标任意移动）
* self.root.bind("<Any-KeyPress>", self.myquit)  
   self.root.bind("<Any-Button>", self.myquit)  
   self.root.bind("<Motion>", self.myquit)
* 63
* 桌⾯应⽤项⽬：⼩球弹弹弹
* # Canvas提供绘图功能(直线、椭圆、多边形等等)， 宽度和⾼度是屏幕分辨率⼤⼩
* self.canvas = Canvas(self.root, width = scrnw, height = scrnh)
* # 让画布按pack()布局  
   self.canvas.pack()
* # 获取球的数量⽣成迭代器，每次迭代创建⼀个球
* for i in range(num):  
   # ball是SettingBalls()类对象，传⼊self.canvas画布，以及屏幕的宽⾼  
   ball = SettingBalls(self.canvas, scrnwidth = scrnw, scrnheight = scrnh)
* # 调⽤ 创建球的⽅法  
   ball.create\_ball()
* # 将⽣成的球对象放到 balls 列表⾥  
   self.balls.append(ball)
* # 调⽤ run\_ball()⽅法，启动⼩球运动  
   self.run\_ball()
* # 调⽤mainloop() 消息循环机制  
   self.root.mainloop()
* def run\_ball(self):  
   for ball in self.balls:  
   ball.move\_ball()  
   # run\_ball 每隔20毫秒会被调⽤⼀次  
   self.canvas.after(20, self.run\_ball)
* def myquit(self, event):  
   self.root.destroy()
* 2. 设置⼩球
* 64
* 桌⾯应⽤项⽬：⼩球弹弹弹
* # 创建⼀个随机球处理类
* class SettingBalls:  
   def \_\_init\_\_(self, canvas, scrnwidth, scrnheight):  
   # \_\_init\_\_函数⾥设置对象的属性
* # 对象⾃⼰的属性，接受canvas参数
* # Canvas是⼀个⻓⽅形的⾯积，图画或其他复杂的布局。可以放置在画布上的图形，⽂字，部件，或  
  是帧
* self.canvas = canvas
* # tkinter绘图采⽤屏幕坐标系，原点在左上⻆，x从左往右 递增, y从上往下 递增  
   # 在绘图区域内，随机产⽣当前 球的圆⼼ 的 x坐标 和 y坐标，⽤于制定出现的位置
* self.xpos = randint(10, int(scrnwidth))  
   self.ypos = randint(10, int(scrnheight))
* # 在绘图区域内，随机产⽣当前球的 x坐标 和 y坐标 的向量  
   # 在数学中，⼏何向量（也称⽮量），指具有⼤⼩和⽅向的量  
   # 这⾥我们可以⽤来表示球的速度
* self.xvelocity = randint(6, 12)  
   self.yvelocity = randint(6, 12)
* # 随机产⽣表示当前球的⼤⼩，也就是半径⻓度
* self.radius = randint(40, 70)
* # 通过lambda表达式创建函数对象r，每次调⽤ r() 都会产⽣ 0 ~ 255之间的数字  
   r = lambda : randint(0, 255)
* # 三次调⽤的数字取前两位，⽤⼗六进制数⽅式存储到 self.color⾥，作为球的颜⾊  
   # #RRGGBB，前2是红⾊，中2是绿⾊，后2是蓝⾊，最⼩是0，最⼤是F，
* # 如 全⿊#000000 全⽩#FFFFFF 全红#FF0000  
   self.color = "#%02x%02x%02x" % (r(), r(), r())
* # 获取整个绘图场景的宽度和⾼度（也就是屏幕分辨率⼤⼩）
* self.scrnwidth = scrnwidth  
   self.scrnheight = scrnheight
* 3. 上吧⼩球球！
* if \_\_name\_\_ == "\_\_main\_\_":  
   MoreBalls(14)
* 使⽤pyinstaller 打包我们的程序
* 65
* 桌⾯应⽤项⽬：⼩球弹弹弹
* 升级我们的pip版本 python -m pip install --upgrade pip
* 通过pip安装 pyinstaller pip install pyinstaller
* 进⼊项⽬⽂件的⽂件夹，执⾏下⾯命令 pyinstaller -w MoreBalls
* 66