# Gui的核心

窗体和Frame

按钮，菜单等

组件Widget

Button，Label，Combox等

布局

grid，pack

事件和回调

实际上最常用的控件如下：

菜单 按钮 输入框 标签 对话框

# 轻量级Tkinter教程

tcl是一款命令行的动态语言，其适用范围十分广泛，tk则是图形用户接口，它不单独属于tcl，是大多数语言的gui描述标准。tkinter是基于tcl/tk的一套面向对象的gui设计标准。

tkinter参考资料：<https://wiki.python.org/moin/TkInter>

## tcl中tk命令

在tcl解释器中，命令形式为

classCommand newPathname options

譬如

button .fred -fg red -text "hi there"

之后，.fred将会构成一个新的命令，

.fred someAction someOptions

## 定义definitions

window

top-level window

一个矩形区域，包括按钮，标题等

widget

frame

可以包含其他widget的widget

child,parent

描述的是widget之间的关系。

## layout manager

在tkinter中有几种“几何管理器”，比如Packer，Grid等。

这里以grid为例。每个组件的都有几种几何管理器，分别由grid,pack等方法实现。

实现组件包含的关系分为两步：创建组件，告诉几何管理器组件的包含方式。

self.*thing* = tk.*Constructor*(*parent*, ...)

self.*thing*.grid(...)

这个方法的意义在于，将组件w注册到父组件的grid 几何管理器中。

grid方法

|  |  |
| --- | --- |
| column | 默认为0，从0开始 |
| columnspan | 跨列 |
| in\_ | 作为次次级组件的父组件声明. |
| ipadx | 内部左右空白距离 |
| ipady | 内部上下空白距离 |
| padx |  |
| pady |  |
| row |  |
| rowspan |  |
| sticky | 决定组件的扩展方式 |

* If you do not provide a sticky attribute, the default behavior is to center the widget in the cell.
* You can position the widget in a corner of the cell by using sticky=tk.NE (top right), tk.SE (bottom right), tk.SW (bottom left), or tk.NW (top left).
* You can position the widget centered against one side of the cell by using sticky=tk.N (top center), tk.E (right center), tk.S (bottom center), or tk.W (left center).
* Use sticky=tk.N+tk.S to stretch the widget vertically but leave it centered horizontally.
* Use sticky=tk.E+tk.W to stretch it horizontally but leave it centered vertically.
* Use sticky=tk.N+tk.E+tk.S+tk.W to stretch the widget both horizontally and vertically to fill the cell.
* The other combinations will also work. For example, sticky=tk.N+tk.S+tk.W will stretch the widget vertically and place it against the west (left) wall.

其他grid方法

### 4.2. Other grid management methods

These grid-related methods are defined on all widgets:

***w*.grid\_bbox(column=None, row=None, col2=None, row2=None)**

Returns a 4-tuple describing the bounding box of some or all of the grid system in widget *w*. The first two numbers returned are the *x* and *y* coordinates of the upper left corner of the area, and the second two numbers are the width and height.

If you pass in column and row arguments, the returned bounding box describes the area of the cell at that column and row. If you also pass in col2 and row2 arguments, the returned bounding box describes the area of the grid from columns column to col2 inclusive, and from rows row to row2 inclusive.

For example, *w*.grid\_bbox(0, 0, 1, 1) returns the bounding box of four cells, not one.

***w*.grid\_forget()**

This method makes widget *w* disappear from the screen. It still exists, it just isn't visible. You can use .grid() it to make it appear again, but it won't remember its grid options.

***w*.grid\_info()**

Returns a dictionary whose keys are *w*'s option names, with the corresponding values of those options.

***w*.grid\_location(*x*, *y*)**

Given a coordinates (*x*, *y*) relative to the containing widget, this method returns a tuple (*col*, *row*) describing what cell of *w*'s grid system contains that screen coordinate.

***w*.grid\_propagate()**

Normally, all widgets propagate their dimensions, meaning that they adjust to fit the contents. However, sometimes you want to force a widget to be a certain size, regardless of the size of its contents. To do this, call *w*.grid\_propagate(0) where *w* is the widget whose size you want to force.

***w*.grid\_remove()**

This method is like .grid\_forget(), but its grid options are remembered, so if you .grid() it again, it will use the same grid configuration options.

***w*.grid\_size()**

Returns a 2-tuple containing the number of columns and the number of rows, respectively, in *w*'s grid system.

***w*.grid\_slaves(row=None, column=None)**

Returns a list of the widgets managed by widget *w*. If no arguments are provided, you will get a list of all the managed widgets. Use the row= argument to select only the widgets in one row, or the column= argument to select only the widgets in one column.

## 与python对应

button .fred =🡺 fred=Button()

button .panel.fred 🡺 fred=Button(panel)

button .fred -fg red =====> fred = Button(panel, fg="red")

.fred configure -fg red =====> fred["fg"] = red

OR ==> fred.config(fg="red")

pack方法：在python中，所有的widget都是Pakcer的子类，因此具有pack方法。

## Packer

pack用于调整子widget在父容器中的位置。父容器的大小由所有的子容器决定，在config之后，一旦使用pack方法，子widget就会重新调整其在父容器中的大小。在config阶段，可以确定子widget的坐标等诸多属性，

用法：fred.pack() #默认side=”top”

fred.pack(side=”left”)

fred.pack(expand=1)

使用关键字参数

选项： anchor expand[0,1] fill[x,y,both, none]

ipadx ipady 内部padding空白距离

padx pady 外部空白距离

side=[left,right,top,bottom]

## 变量直接关联

关联：如果变量x的值改变，显示x值的widget也改变。在现阶段的tkinter中，支持这种关联的类型全部是Variable 的子类： StringVar, IntVar, DoubleVar, and BooleanVar

Variable的方法：

set

get

## Tk window manager

在Tk中，有一个直接用于管理空间属性的工具:wm.wm对应python中Wm类。

在tkinter中，所有的widget都继承自Wm类，具有Wm类的所有方法。

属性和方法（只能用于窗体widget，button widget等不能使用）：

master 上层框架

\_root() 顶层窗口

title()

maxsize()

# 在tkinter8.5以后使用ttk

ttk中的组件不再使用fg，bg等属性，而是使用ttk.Style class来创造视觉。

To start using Ttk, import its module:

from tkinter import ttk

## 组件

Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale and Scrollbar. （在tkinter中已经存在）

[Combobox](mk:@MSITStore:D:\Python35-32\Doc\python352.chm::/library/tkinter.ttk.html#tkinter.ttk.Combobox), [Notebook](mk:@MSITStore:D:\Python35-32\Doc\python352.chm::/library/tkinter.ttk.html#tkinter.ttk.Notebook), [Progressbar](mk:@MSITStore:D:\Python35-32\Doc\python352.chm::/library/tkinter.ttk.html#tkinter.ttk.Progressbar), Separator, Sizegrip and [Treeview](mk:@MSITStore:D:\Python35-32\Doc\python352.chm::/library/tkinter.ttk.html#tkinter.ttk.Treeview).（新定义的组件）

全部都是Widget的子类

Tk code:

l1 = tkinter.Label(text="Test", fg="black", bg="white")

l2 = tkinter.Label(text="Test", fg="black", bg="white")

Ttk code:

style = ttk.Style()

style.configure("BW.TLabel", foreground="black", background="white")

l1 = ttk.Label(text="Test", style="BW.TLabel")

l2 = ttk.Label(text="Test", style="BW.TLabel")

类似于在html中使用css属性一样将所有的style单独列成一个类来使用，减少数据和属性的耦合关系。

## 组件之间的继承关系及其图形

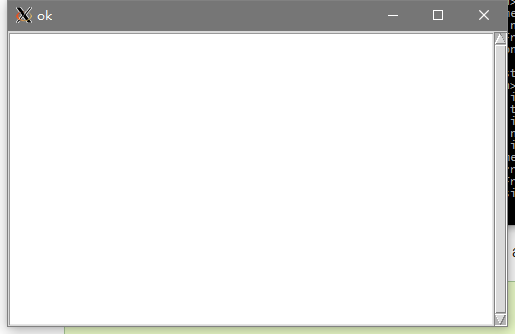
**Tk**

The [Tk](https://docs.python.org/3/library/tkinter.html#tkinter.Tk) class is instantiated without arguments. This creates a toplevel widget of Tk which usually is the main window of an application. Each instance has its own associated Tcl interpreter.

每一个实例化的Tk都具有一个独立的tcl解释器。每一个tkinter程序都应当具有一个顶级Tk对象，既作为顶层窗口，也作为程序代表。

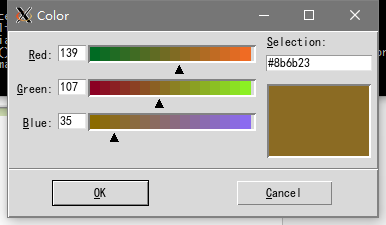
**Widget**

**Frame**

**tkinter.scrolledtext.ScrolledText** 

**tkinter.colorchooser**

Dialog to let the user choose a color.



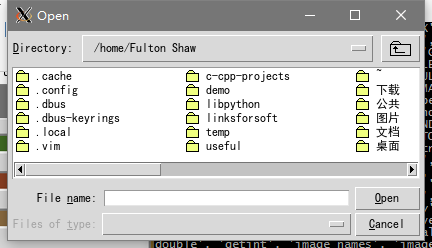
tkinter.colorchooser.askcolor()

**tkinter.commondialog**

Base class for the dialogs defined in the other modules listed here.

**tkinter.filedialog**

Common dialogs to allow the user to specify a file to open or save.



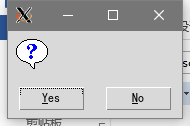
**tkinter.font**

Utilities to help work with fonts.

**tkinter.messagebox**

Access to standard Tk dialog boxes.

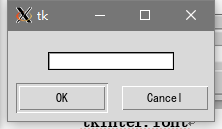
方法：'askokcancel', 'askquestion', 'askretrycancel', 'askyesno', 'askyesnocancel', 'showerror', 'showinfo', 'showwarning'



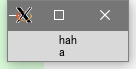
**tkinter.simpledialog**

Basic dialogs and convenience functions.

方法：'askfloat', 'askinteger', 'askstring',



**tkinter.Message**



## 使用菜单

菜单只能显示在顶级窗口中(top level)。菜单可以内嵌其他的菜单。using **add\_cascade**

root = Tk()

def hello():

print "hello!"

# create a toplevel menu

menubar = Menu(root)

menubar.add\_command(label="Hello!", command=hello)

menubar.add\_command(label="Quit!", command=root.quit)

# display the menu

root.config(menu=menubar)

一个menu可以添加其他的menu，也可以添加command，seperator。

**add(type, \*\*options)** [[#](http://effbot.org/tkinterbook/menu.htm#Tkinter.Menu.add-method)]

Add (append) an entry of the given type to the menu.

*type*

What kind of entry to add. Can be one of **“command”**, **“cascade”** (submenu), **“checkbutton”**, **“radiobutton”**, or **“separator”**.

*\*\*options*

Menu options.

*activebackground=*

*activeforeground=*

*accelerator=*

*background=*

*bitmap=*

*columnbreak=*

*command=*

*font=*

*foreground=*

*hidemargin=*

*image=*

*indicatoron=*

*label=*

*menu=*

*offvalue=*

*onvalue=*

*selectcolor=*

*selectimage=*

*state=*

*underline=*

*value=*

*variable=*

**post(x, y)** [[#](http://effbot.org/tkinterbook/menu.htm#Tkinter.Menu.post-method)]

Displays the menu at the given position. The position should be given in pixels, relative to the root window.

*x*

Menu position.

*y*

Menu position.

**unpost()** [[#](http://effbot.org/tkinterbook/menu.htm#Tkinter.Menu.unpost-method)]

Removes a posted menu.

# GUI在windows和linux上的差别

在linux上，gui是基于x11服务的，所以在所有工作之前应当先安装并运行x服务。cygwin下的x服务为:xorg-server,xinit

# 开发的程序---menu部分

# 另一个计划：在github上上传一个tkinter demos and examples using python3

1. 在cygwin中配置

首先需要安装xorg-server

然后使用XWin -multiwindow打开后台服务器模式

export DISPLAY=:0来配置显示器

1. 如何指定窗体大小，光是配置width height是不够的.必须理解pack管理的方式。

pack管理器：将组件包装到行和列，可以使用fill,expand,side等参数来控制。想象一个长方体的包装盒，pack将组件按顺序放入，然后计算大小，最后得出整个包裹的大小，然后放到父容器中。

fill=BOTH,X,Y

expand=1 允许用户缩放

side= 放置依照的边 ，这里可以看成每次放入一个容器都是依照一个矩形的方式放入的，然后top，left，right，bottom会随之更新。

注意：pack无法实现换行的管理器，一旦一个组件占据了一列，整个pack的空间就会向减少占用的那整整一列。

似乎指定行列仅在容器为空的情况下可用，否则其真实的行列大小由子组件共同决定。

参考：<http://effbot.org/tkinterbook/pack.htm>

1. grid管理器：注意不要在同一个master中混合使用grid和pack管理器

参考资料：<http://www.effbot.org/tkinterbook/grid.htm>

row，column指定相当于x，y的坐标，默认为0

sticky属性：N E S W，相当于中心位置。

代码风格转变：

mport tkinter

def makeWorkArea(parent):

WorkArea = tkinter.Frame(parent)

WorkArea.config(relief = 'sunken', width = 340, height = 170, bg = 'red')

WorkArea.pack(expand = 'yes', fill = 'both')

msg = tkinter.Label(WorkArea, text='Window menu basics')

msg.pack()

root = tkinter.Tk()

makeWorkArea(root)

root.mainloop()

使用字符串代替预定义的常量。

1. place管理器

参考：<http://stackoverflow.com/questions/4241036/how-do-i-center-a-frame-within-a-frame-in-tkinter>

参考：http://effbot.org/tkinterbook/place.htm

居中：f.place(anchor=”c”)

* **anchor :** The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
* **bordermode :** INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.
* **height, width :** Height and width in pixels.
* **relheight, relwidth :** Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
* **relx, rely :** Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
* **x, y :** Horizontal and vertical offset in pixels.

## 关于given size

参考：http://stackoverflow.com/questions/21465903/how-to-create-a-window-of-a-given-size

默认情况下，grid和pack都会优先以内容的大小来确定自身的大小，从而忽略自定义的size。这种策略称为geometry propagate.可以使用pack\_propagate(False),gird\_propagate来取消这种特性

# PyQT5教程

## 环境配置

安装QT的时候，需要将 mingw也要选择上。

## QT Designer的使用方法

## QT两种最重要的机制：槽和信号

QT的事件处理是基于槽和信号这两种机制，如果有一个button，则点击它的时候，信号clicked被发送，而槽(slot)可以是QT Slot或者任何Python可调用的实体。

QtCore.QCoreApplication包含主要的事件循环，它负责处理和分发所有的事件。

instance()方法会获得一个当前的实例。

使用 btn.clicked.connect(QCoreApplication.instance().quit) 将信号和槽连接起来，一个是sender,一个是receiver.

* Event事件

事件的模型包括 ： 事件来源，事件对象，事件目标

事件是由于某个物体的状态发生改变而产生的。

通过sender() 来获得事件的产生对象。

* QObject发送消息

pyqtSignal形成一个信号实例

connect将一个方法加入其中。

通过它提供的emit()方法将slot唤醒。

pyqtSignal(int,int,str) 表示接受int,int,str作为输入类型.

* 多线程

QT的设计方法是通过信号来更新组件,所以不要在主线程之外更新UI组件.

实现一个QThread的方法是继承一个QThread对象,然后定义信号;定义run方法

class MyThread(QThread):

sigDo = pyqtSignal(int)

def run(self):

…

self.sigDo.emit(….)

## QT应用程序框架

import sys

from PyQt5.QtWidgets import QApplication, QWidget

if \_\_name\_\_ == *'\_\_main\_\_'*:

#每个QT的应用程序必须创建一个QApplication对象

app = QApplication(sys.argv)

w = QWidget()

w.resize(250, 150)

w.move(300, 300)

w.setWindowTitle(*'Simple'*)

w.show()

#清除所有使用的资源, exec\_使用下划线，是因为exec是python关键字。（之前的版本）

sys.exit(app.exec\_())

## 布局管理

布局所要完成的工作就是为每个部件指定位置(x,y)和大小(w,h)。

最直接的布局就是absolute布局，但是有以下局限

* 大小和位置不会因为部件的改变而改变
* 在不同的平台上外观不同
* 更改字体可能导致布局破坏
* 如果要改变布局，需要重新安排

所以布局还要考虑动态调整

绝对布局通过move和resize来实现。

1. Box 布局

两个类 : QHBoxLayout和QVBoxLayout 分别是水平和垂直布局

从本质上来说就是线性布局。

方法: addStretch(n) 保留n个空白区域

addWidget(w) 添加部件

addLayout(l) 如果类型是layout，使用这个方法

1. QGridLayout

最常用的布局方法

方法：

addWidget(w, i,j) 在i行j列添加w

addWidget(w,i,j,ispan,jspan) 添加w,并且设置跨越

setSpacing(n) 设置内部的间距

## 类体系

* QApplication

方法

exec\_() 不断接受消息，直到收到它认为可以使进程结束的信号

* QWidget

是一切QT类的基类。通常其构造函数为

QWidget(parent) parent也是一个QWidget对象

如果不提供parent对象，就是一个window

方法：

resize(w,h)

move(x,y)

setGeometry(x,y,w,h) 组合了move和resize

setWindowTitle(title)

setWindowIcon(qicon) 接受一个qicon对象

setToolTip(fmtStr) formatstr 可以含有html标注

sizeHint() 给出一个合适的大小建议

closeEvent(event) 发生close事件, QCloseEvent类型, 子类可以重写

statusBar() 获取当前的状态行， statusBar仅顶层窗口可有

setStatusTip(str) 设置当鼠标划过的时候，状态栏显示的消息

setLayout(l) 设置当前容器类型的空间的布局（在一切你觉得是容器的类型上调用这个方法）

show()

1. 关于层级结构

QPushButton(string text, QWidget parent = None)

以这个构造方法为例，每个组件都有一个父组件，所以形成了一个层次结构。没有父组件的就是顶层窗体。

* QIcon

PyQt5.QtGui.QIcon

方法

构造函数 QIcon(图像文件)

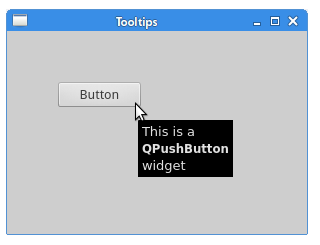
* QToolTip

PyQt5.QtWidgets

描述：在鼠标悬停的时候显示的消息

方法：

静态方法 QToolTip.setFont(qfont) 设置显示的字体



* QPushButton

PyQt5.QtWidgets

构造函数 QPushButton(text,parent)

方法

setCheckable(bool) 是否是可以check的按钮

* QFont

PyQt5.QtGui

构造函数 QFont(‘字体名’,像素)

QFont(‘SansSerif’,10)

* QMessageBox

QtWidgets

方法:

reply = QMessageBox.question(self, 'Message',

"Are you sure to quit?", QMessageBox.Yes |

QMessageBox.No, QMessageBox.No)

question方法在一个widget上弹出一个请求。显示标题，消息，选项，默认按钮

* QMainWindow

QWidgets

* QAction

QWidgets

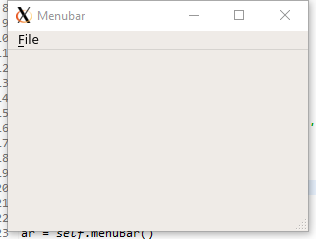
QAction是基于menubar,toolbar,键盘快捷键的抽象动作

方法

init(qicon,’&Exit’,parent) 表示Exit这个菜单可由Alt+E触发

setShortcut(‘Ctrl+Q’) 设置

triggered.connect(qApp.exit) 点击的时候触发exit动作



* qApp

QWidgets

* (菜单)

menubar=self.menuBar()

fileMenu = menubar.addMenu(‘&File’) Alt+F打开

fileMenu.addAction(qAction)

* (工具栏)

self.toolbar = self.addToolBar(‘Exit’)

self.toolbar.addAction(exitAction)

* QInputDialog

getText(parent, title,hint) 一个输入框提示，返回 text,ok

text,ok=getText(…) ok是True或者False类型

* QColorDialog

getColor()

* QColor

QtCore.QColor

color = QColor(0,0,0)

* QFontDialog

font,ok=getFont()

* QFileDialog

getOpenFileName(parent, title,path) 返回一个name数组

* QCheckBox

方法:

toggle()

信号stateChanged

回调方法 ： callback(state)

if state==Qt.Checked

….

* QLineEdit
* QTextEdit

可以用于显示HTML文档.所以它的方法也是基于DOM的.

为了设置普通的文本,可以使用 textEdit.document().setPlainText()方法