Tkinter 8.4 参考：一个 Python 的图形接口

原文作者：Joh W.Shipman
译者：william0victor@gmail.com

原文发布时间：2010-12-12 14:13

摘要

描述 Tkiner 微组件在 python 编程语言中构造用户图形界面本参考手册已有在线格式[1]并导出为PDF 文档[2]。有任何意见请发送到tcc-doc@nmt.edu。

[1] http://www.nmt.edu/tcc/help/pubs/tkinter/

[2] http://www.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf

目录

Chapter 1

什么是 Tkinter？

Tkinter 是 Python 的一个 GUI（用户图形界面）微组件。本文档只涵盖常用特性。本文档适用于 Python 2.5 以及 Tkinter 8.4，运行在基于 Linux 系统的 X Window 系统中。你使用的版本或许稍有不同。相关参考：

- Fredrik Lundh, who wrote Tkinter, has two versions of his *An Introduction to Tkinter*: a more complete 1999 version[3] and a 2005 version[4] that presents a few newer features.
- *Python and Tkinter Programming* by John Grayson (Manning, 2000, ISBN 1-884777-81-3) is out of print, but has many useful examples and also discusses an extension package called *Pmw*: Python megawidgets[5].
- *Python 2.5 quick reference*[6]: general information about the Python language.
- For an example of a sizeable working application (around 1000 lines of code), see huey: *A color and font selection tool*[7].

我们将由从 Tkingter 的可见部分开始看起：创建微件，并将他们布置到屏幕上。稍后我们将讨论怎样连接程序前端面板到后端逻辑。

[3] http://www.pythonware.com/library/tkinter/introduction/
[4] http://effbot.org/tkinterbook/
[5] http://pmw.sourceforge.net/
[6] http://www.nmt.edu/tcc/help/pubs/python/web/
[7] http://www.nmt.edu/tcc/help/lang/python/examples/huey/

Chapter 2

一个小程序

下面是一个只包含一个退出按钮的 Tkinter 小程序：

```
#!/usr/local/bin/python                          1
from Tkinter import *                            2
class Application(Frame):
        def __init__(self, master=None):         3
                Frame.__init__(self, master)     4
                self.grid()                      5
                self.createWidgets()
        def createWidgets(self):
                        self.quitButton = Button ( self, text='Quit',
                            command=self.quit ) 6
                        self.quitButton.grid()   7
app = Application()                              8
app.master.title("Sample application")           9
app.mainloop()                                  10
```

1. 假如你的系统有 Python 解释器，路径为/usr/local/bin/python，本行将使脚本自动运行。【译者注：由于不同的 linux 稍有差异，Python 解释器的安装路径可能有差异，建议该行写为 #!/usr/bin/env python】
2. 本行导入 Tkinter 所有包到你程序的命名空间。
3. 你的程序的类必须从 Tkinter 的 Frame 类继承。
4. 调用 Frame 父类构造函数。
5. 需要让程序真正显示在屏幕上。
6. 创建一个标记为“Quit”的按钮。
7. 将按钮放置在程序中。
8. 实例化 Application 类来运行主程序。
9. 调用本方法设置窗口的抬头为“Sample application”。
10. 开始程序主回路，等待鼠标和键盘事件。

Chapter 3

定义

在我们继续前，先让我们来解释一些常用的项目。

window

本项在不同的环境中有不同的意义，但是通常来说它指的是在你屏幕上的某个矩形区域。

Top-level window

一个独立存在于你的屏幕上的窗口。它将由你的系统桌面管理器装饰上标准的框架和控制器。你能在桌面上四处移动它。通常你也可以调节它的尺寸，尽管你的程序可以阻止。

widget

这些项目像积木一样构建程序的图形界面。**widget** 的例子：按钮，单选框，文本域，框架，以及文本标签。

frame

在 **Tkinter** 中，**Frame** 微件是构建复杂布局的基本单元。一个框架是能包含其它微件的矩形区域。

Child,parent

当一个微件被创建，一个父类 -子类的关系也被创建。例如，如果你放置一个文本标签到一个框架中，那么这个框架就是文本标签的父类。

Chapter 4

布局管理

稍候我们将讨论微件——你的图形界面的积木。怎样在窗口中方置微件。尽管在 Tkinter 中有三种不同的“布局管理器”，作者强烈推荐.grid() 布局管理器来美化所有东西。这个管理器将每个窗口或者框架当作一个表——行和列的网格管理。单元格是一行和一列的交叉区域。每行的宽就是在行中微件单元格的宽。每列的高就是列中最大单元格的高。由于微件没有完全填满单元格，你可以在多出来的空间里做些什么。你既可以将多余的空间移出到微件外，也可以调整微件来填满它，水平或垂直方向都可以。你可以将多个单元格融合为一个更大的区域，一个称为纺丝的过程。当你创建了一个微件，直到你在布局管理器中注册了它，否则它不会显示。因此，构建和放置一个微件是两个步骤，步骤如下：

```
1 self.thing = Constructor(parent, ...)
2 self.thing.grid(...)
```

Contructor 是指微件的类型如 Button，Frame，以及其他，parent 是指将会构建子类微件的父类微件。所有的微件都有一个.grid() 方法，你可以用来告诉布局管理器把微件放哪。

4.1 .grid() 方法

把微件 w 显示在你的程序窗口上：

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

这个方法用网格布局管理器注册了一个微件 w——如果你不这么做，那么这个微件将存在于内部，但是在屏幕上它将不可见。

下面是.grid() 布局管理器方法的选项：

column	你想将微件放置的行数，从 0 开始计算。默认值是 0。
columnspan	通常一个微件只占据网格中的一个单元格。然而，你可以选取列中多个单元格，并在 columnspan 选项中设置单元格的数量来整合他们到一个大单元格。例如，*w*.grid(row=0,column=2,columnspan=3)，例中将会把 *w* 微件放置在一个横跨 0 列 2，3，4 行的一个单元格中。
in_	注册微件 w 作为某个微件如 w_2 的子类，用法 in_=w_2。当 w 微件被创建时，新的 w_2 微件必须作为 parent 微件的子类来使用。
ipadx	内部 x padding。这个维度是增加微件内部左边和右边。
ipady	内部 y padding。这个维度是增加微件内部顶部和底部。
padx	外部 x padding。这个维度是增加微件外部左边和右边。
pady	外部 y padding。这个维度是增加微件上部和下部。
row	你想把微件插入的列数，从 0 计数。默认是下一个更高的空闲列。

rowspan	通常一个微件只占据网格的一个单元格。你可以选取一行内多个临近的单元格，但是，给选中的单元格设置 rowspan 选项。本选项和 columnspan 选项结合使用来抓取一块单元格。例子，*w*.grid(row=3,column=2,rowspan=4,columnspan=5) 例中将 *w* 微件放入一个由 3-6 列 2-6 行组成的区域中。
sticky	本项决定怎样分配单元格中的微件所占空间之外的空间。见下。

- 如果未设置 sticky 属性，默认会将微件在单元格中居中放置。
- 你可以使用 sticky=NE（右上），SE（右下），SW（左下），或者 NW（左上）来布局微件到单元格的四角。
- 你可以使用 sticky=N（中上），E（中右），S（中下），或者 W（中左）来布局微件到一边的相对中间。
- 使用 sticky=N+S 垂直扩展微件但让它水平居中。
- 使用 sticky=E+W 水平扩展微件但让它垂直居中。
- 使用 sticky=N+E+S+W 在水平和垂直方位来扩展微件填充单元格。
- 其它的组合也可使用。例如，sticky=N+S+W 将垂直扩展微件并放置微件在相对东（左）边。

4.2 其它 grid 管理方法

这些 grid 相关的方法在所有微件上都有定义：

w.grid_bbox (column=None, row=None, col2=None, row2=None)

返回一个四元组描述微件 w 中一些或所有 grid 系统的边界框。前两个数字返回左上角区域的 x 和 y 坐标，后两个数字是宽和高。

如果你传递行和列变量，返回的限定框描述所在行和列的单元格的区域。如果你也传递了 col2 和 row2 参数，返回的限定框描述包含从行 column 到 col2，列 row 到 row2 的区域。

例如，w.grid_bbox(0,0,1,1) 返回四个单元格的限定框，不是一个。

w.grid_forget()

本方法使微件 w 从屏幕上消失。它还存在，只是不可见。你可以使用.grid() 使它再次显示，但是它将不会记住它的 grid 选项。

w.grid_info()

返回一个键为 w 微件选项名字的字典，以及这些选项相应的值。

w.grid_location (x,y)

赋予关联的包含的微件一个坐标，本方法返回一个数组（col,row）描述 w 微件的网格系统的单元格包含的屏幕坐标。

w.grid_propagate()

通常，所有微件传送他们的尺寸，意味着他们调节来适应内容。然而，有时你想约束一个微件到确定的尺寸，忽略它内容的尺寸，这样做，调用 w.grid_propagate(0) 限制 w 微件的尺寸。

w.grid_remove()

本方法类似.grid_forget() ，但是它的 grid 选项会记住，所以如果你再.grid() 它，它将会使用相同的 grid 配置选项。

w.grid_size()

分别在 w 微件 grid 系统中返回一个包含行数和列数的二元组。

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

返回由微件 w 管理的微件的目录。如果没有提供参数，你将会获得所有被管理的微件的目录。使用 row= 参数只选择一列微件，或者使用 column= 参数只选择一行的微件。

4.3 设置行和列的尺寸

除非你采取确切的措施，网格列内的微件宽将会等于它的最大宽度，并且网格行内的微件高将会是最高的单元格的高。微件上的 sticky 属性只控制它被放置的地方如果它没有完全填充单元格。

w.columnconfigure (N, option=value,...)

w 微件的网格层中，配置 column N 以便所给选项有所给的值。对于选项，请看下表。

w.rowconfigure (N, option=value, ⋯)

w 微件的网格层中，配置 row N 以便所给的选项有所给的值。对于选项，请看下表。

以下是用来配置 column 和 row 尺寸的选项。

minsize	以像素为最小单位，列和行的最小尺寸。
pad	若干像素将会被加入所给的列或行，及以上的列或行中最大的单元格。

weight	为了使一列或一行可拉伸，当重新分配额外的空间时，使用此选项并提供一个值，赋予该列或行相对权重。例如，如果一个微件 w 包含一个网格层，这些行将会分配 3/4 的额外空间到第一列及 1/4 到第二列：
	`w.columnconfigure(0, weight=3)`
	`w.columnconfigure(1, weight=1)`
	如果没有使用此项，列或行将不会伸展。

4.4 是根窗口可重组

如果你想让用户调整你的全部程序窗口，并分配在其内部微件的额外空间吗？这需要一些不明显的操作。

那就很有必要使用行和列大小管理技巧，4.3 节所讲述的，“配置列和行大小”（p.7），来使你的程序微件的网格可伸展。然而，那仅仅是不够的。

思考下第二节讨论的小程序，“一个小程序”（p.2），其只包含一个退出按钮。如果你运行此程序，并调整窗口大小，按钮保持同样的尺寸，居中于窗口。

以下是小程序内.__createWidgets() 方法的替代版本。在这个版本中，退出按钮一直填充可利用空间。

```
def createWidgets(self):
        top=self.winfo_toplevel()          1
        top.rowconfigure(0,weight=1)       2
        top.columnconfigure(0,weight=1)    3
        self.rowconfigure(0,weight=1)      4
        self.columnconfigure(0,weight=1)          5
        self.quit=Button (self, text"="Quit, command=self.quit )
        self.quit.grid( row=0, column=0, sticky=N+S+E+W)          6
```

1. 顶层窗口是屏幕上最外层的窗口。然而，这个窗口不是你的程序的窗口——它是程序实例的父类。要获取顶层窗口，在程序中的任何微件上调用.winfo_toplevel() 方法。参看 25 节，“通用微件方法”（p.91）。
2. 本行使顶层窗口的 0 行网格可伸展。
3. 本行使顶层窗口的 0 列网格可伸展。
4. 使 0 行的程序微件的网格可伸展。
5. 使 0 列的程序微件的网格可伸展
6. 参数 sticky=N+S+E+W 使按钮扩张填满它的单元格

还必须作出一个变化。在构造函数中，如下显示的改变第二行：

```
def __init__(self,master=None):
        Frame.__init__(self,master)
        self.grid(sticky=N+S+E+W)
        self.createWidgets()
```

sticky=N+S+E+W 参数对 self.grid() 是必要的，因此程序微件将会扩张填充它的顶层窗口网格的单元格。