The Tkinter Text Widget

Text（文本）组件用于显示和处理多行文本。在 Tkinter 的所有组件中，Text 组件显得异常强大和灵活，适用于多种任务。虽然该组件的主要目的是显示多行文本，但它常常也被用于作为简单的文本编辑器和网页浏览器使用。

何时使用 Text 组件？

Text 组件用于显示文本文档，包含纯文本或格式化文本（使用不同字体，嵌入图片，显示链接，甚至是带 CSS 格式的 HTML 等）。因此，它常常也被用于作为简单的文本编辑器和网页浏览器使用。

用法

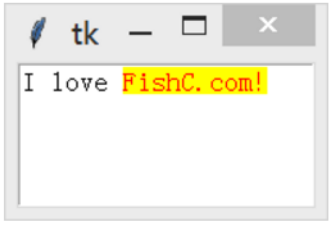
当你创建一个 Text 组件的时候，它里边是没有内容的。为了给其插入内容，你可以使用 insert() 方法以及 INSERT 或 END 索引号：

```
from tkinter import *

root = Tk()

text = Text(root)
text.pack()

# INSERT 索引表示插入光标当前的位置
text.insert(INSERT, "I love ")
text.insert(END, "FishC.com!")

mainloop()
```

复制代码

值得一提的是，Text 组件的 insert() 方法有一个可选的参数，用于指定一个或多个“标签”（标签用于设置文本的格式，请参考下方【Tags 用法】）到新插入的文本中：

```
from tkinter import *

root = Tk()

text = Text(root, width=20, height=5)
text.pack()

# 设置 tag
text.tag_config("tag_1", backgroun="yellow", foreground="red")

# INSERT 索引表示插入光标当前的位置
text.insert(INSERT, "I love ")
text.insert(END, "FishC.com!", "tag_1")

mainoop()

复制代码
```



在 Text 组件中插入对象，可以使用 window_create() 和 image_create() 方法：

```
from tkinter import *

root = Tk()

text = Text(root, width=20, height=5)
text.pack()

text.insert(INSERT, "I love FishC.com!")

def show():
    print("哟，我被点了一下~")

b1 = Button(text, text="点我点我", command=show)
text.window_create(INSERT, window=b1)

mainloop()
```

复制代码


```
>>>
哟，我被点了一下~
```

删除 Text 组件中的内容可以用 delete() 方法，下边代码用于删除所有内容（也包含 window 和 image 对象，但不会删除 marks 的内容）：

```
text.delete(1.0, END)
```

复制代码

删除单独一个字符（或者一个 window 对象，或者一个 image 对象），你可以仅使用一个参数：

```
text.delete(b1)
```

复制代码

将 state 选项从默认的 NORMAL 修改为 DISABLED，使得 Text 组件中的内容为“只读”形式。不过需要注意的是，当你需要进行任何修改的时候，记得将 state 选项改回 NORMAL，否则 insert() 和 delete() 方法都会失效。

获得 Text 组件的内容，可以使用 get() 方法（仅获取文本内容）：

```
contents = text.get(1.0, END)
```

复制代码

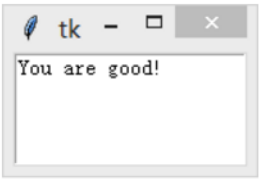
在下边例子中，通过校检 Text 组件中文本的 MD5 摘要来判断内容是否发生改变：

```
from tkinter import *
import hashlib

root = Tk()

text = Text(root, width=20, height=5)
text.pack()

text.insert(INSERT, "I love FishC.com!")
contents = text.get(1.0, END)

def getSig(contents):
    m = hashlib.md5(contents.encode())
    return m.digest()

sig = getSig(contents)

def check():
    contents = text.get(1.0, END)
    if sig != getSig(contents):
        print("警报：内容发生变动！")
    else:
        print("风平浪静~")

Button(root, text="检查", command=check).pack()

mainloop()
```

复制代码

index() 方法用于将所有支持的“索引”格式（请参考下方【Indexes 用法】）转换为“行.列”格式的索引号：

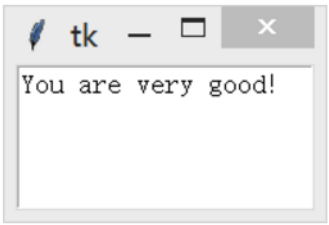
```
print(text.index(INSERT))
text.insert(INSERT, "You are good!")
print(text.index(INSERT))
```
复制代码




```
>>>
1.0
1.13
```

如果你需要跟踪一个位置，那么你可以将该位置“标记”下来（请参考下方【Marks 用法】）：

```
text.insert(INSERT, "You are good!")
text.mark_set("here", '1.8')
text.insert("here", "very ")
```
复制代码



最后，使用 search() 方法可以搜索 Text 组件中的内容。你可以提供一个确切的目标进行搜索（默认），也可以使用 Tcl 格式的正则表达式进行搜索（需设置 regexp 选项为 True）：

```
from tkinter import *

root = Tk()

text = Text(root, width=30, height=5)
text.pack()

text.insert(INSERT, "I love FishC.com!")

# 将任何格式的索引号统一为元祖 (行,列) 的格式输出
def getIndex(text, index):
    return tuple(map(int, str.split(text.index(index), ".")))

start = 1.0
while True:
    pos = text.search("o", start, stopindex=END)
    if not pos:
        break
    print("找到啦，位置是：", getIndex(text, pos))
    start = pos + "+1c"  # 将 start 指向下一个字符

mainloop()
```

复制代码


```
>>>
找到啦，位置是： (1, 3)
找到啦，位置是： (1, 14)
```

如果忽略 stopindex 选项，表示直到文本的末尾结束搜索。设置 backwards 选项为 True，则是修改搜索的方向（变为向后搜索，那么 start 变量你应该设置为 END，stopindex 选项设置为 1.0，最后 "+1c" 改为 "-1c"）

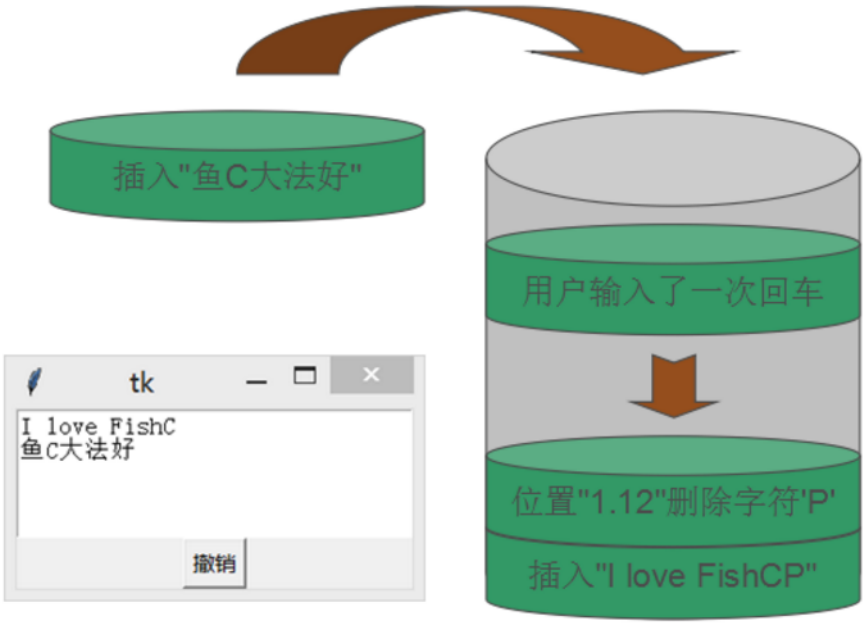
恢复 和 撤销 操作

Text 组件还支持“恢复”和“撤销”操作，这使得 Text 组件显得相当高大上。

通过设置 undo 选项为 True 可以开启 Text 组件的“撤销”功能。然后用 edit_undo() 方法实现“撤销”操作，用 edit_redo() 方法实现“恢复”操作。

这是因为 Text 组件内部有一个栈专门用于记录内容的每次变动，所以每次“撤销”操作就是一次弹栈操作，“恢复”就是再次压栈。

大概就是下图这么回事（小甲鱼不会作图，谅解谅解...）



默认情况下，每一次完整的操作将会放入栈中。但怎么样算是一次完整的操作呢？Tkinter 觉得每次焦点切换、用户按下 Enter 键、删除\插入操作的转换等之前的操作算是一次完整的操作。也就是说你连续输入“FishC 是个 P”的话，一次的“撤销”操作就会将所有的内容删除。

那我们能不能自定义呢？比如我希望插入一个字符就算一次完整的操作，然后每次点击“撤销”就去掉一个字符。

当然可以！做法就是先将 autoseparators 选项设置为 False（因为这个选项是让 Tkinter 在认为一次完整的操作结束后自动插入“分隔符”），然后绑定键盘事件，每次有输入就用 edit_separator() 方法人为地插入一个“分隔符”：

```
from tkinter import *

root = Tk()

text = Text(root, width=30, height=5, autoseparators=False, undo=True, maxundo=10)
text.pack()

def callback(event):
    text.edit_separator()

text.bind('<Key>', callback)

text.insert(INSERT, "I love FishC")

def show():
    text.edit_undo()

Button(root, text="撤销", command=show).pack()

mainloop()
```

复制代码

Indexes 用法

Indexes（索引）是用来指向 Text 组件中文本的位置，跟 Python 的序列索引一样，Text 组件索引也是对应实际字符之间的位置。

Tkinter 提供一系列不同的索引类型：

- "line.column"（行/列）
- "line.end"（某一行的末尾）
- INSERT
- CURRENT
- END
- user-defined marks
- user-defined tags（"tag.first"，"tag.last"）
- selection（SEL_FIRST，SEL_LAST）
- window coordinate（"@x,y"）
- embedded object name（window，images）
- expressions

下边我们看个给大家讲解

"line.column"

行/列 是最基础的索引方式，它们将索引位置的行号和列号以字符串的形式表示出来（中间以 "." 分隔，例如 "1.0"）。需要注意的是，行号以 1 开始，列号则以 0 开始。你还可以使用以下语法构建索引：

```
01.  "%d.%d" % (line, column)
     复制代码
```

指定超出现有文本的最后一行的行号，或超出一行中列数的列号都不会引发错误。对于这样的指定，Tkinter 解释为已有内容的末尾的下一个位置。

需要注意的是，使用 行/列 的索引方式看起来像是浮点值。其实不只像而已，你在需要指定索引的时候使用浮点值代替也是可以的：

```
01.  text.insert(INSERT, "I love FishC")
02.  print(text.get("1.2", 1.6))
     复制代码
```


```
>>>
love
```

使用 index() 方法可以将所有支持的“索引”格式转换为“行/列”格式的索引号。

"line.end"

行号加上字符串 ".end" 的格式表示为该行最后一个字符的位置：

```
01.  text.insert(INSERT, "I love FishC")
02.  print(text.get("1.2", "1.end"))
     复制代码
```


```
>>>
love FishC
```

INSERT (或 "insert")

对应插入光标的位置。

CURRENT (或 "current")

对应与鼠标坐标最接近的位置。不过，如果你紧按鼠标任何一个按钮，它会直到你松开它才响应。

END (或 "end")

对应 Text 组件的文本缓冲区最后一个字符的下一个位置。

user-defined marks

user-defined marks 是对 Text 组件中位置的命名。INSERT 和 CURRENT 是两个预先命名好的 marks，除此之外你可以自定义 marks (请参考下方【Marks 用法】)。

User-defined tags

User-defined tags 代表可以分配给 Text 组件的特殊事件绑定和风格 (请参考下方【Tags 用法】)。

你可以使用 "tag.first" (使用 tag 的文本的第一个字符之前) 和 "tag.last" (使用 tag 的文本的最后一个字符之后) 语法表示标签的范围。

```
"%s.first" % tagname
"%s.last" % tagname
```
复制代码

如果查无此 tag，那么 Tkinter 会抛出一个TclError 异常。

selection (SEL_FIRST , SEL_LAST)

selection 是一个名为 SEL (或 "sel") 的特殊 tag，表示当前被选中的范围，你可以使用 SEL_FIRST 到 SEL_LAST 来表示这个范围。如果没有选中的内容，那么 Tkinter 会抛出一个TclError 异常。

window coordinate ("@x,y")

你还可以使用窗口坐标作为索引。例如在一个事件绑定中，你可以使用以下代码找到最接近鼠标位置的字符：

```
"@%d,%d" % (event.x, event.y)
```
复制代码

embedded object name (window , images)

embedded object name 用于指向在 Text 组件中嵌入的 window 和 image 对象。要引用一个 window，只要简单地将一个 Tkinter 组件实例作为索引即可。引用一个嵌入的 image，只需使用相应的 PhotoImage 和 BitmapImage 对象。

expressions

expressions 用于修改任何格式的索引，用字符串的形式实现修改索引的表达式。

具体表达式实现如下：

表达式	含义
"+ count chars"	1. 将索引向前（->）移动 count 个字符 2. 可以越过换行符，但不能超过 END 的位置
"- count chars"	1. 将索引向后（<-）移动 count 个字符 2. 可以越过换行符，但不能超过 "1.0" 的位置
"+ count lines"	1. 将索引向前（->）移动 count 行 2. 索引会尽量保持与移动前在同一列上，但如果移动后的那一行字符太少，将移动到该行的末尾
"- count lines"	1. 将索引向后（<-）移动 count 行 2. 索引会尽量保持与移动前在同一列上，但如果移动后的那一行字符太少，将移动到该行的末尾
" linestart"	1. 将索引移动到当前索引所在行的起始位置 2. 注意，使用该表达式前边必须有一个空格隔开
" lineend"	1. 将索引移动到当前索引所在行的末尾 2. 注意，使用该表达式前边必须有一个空格隔开
" wordstart"	1. 将索引移动到当前索引指向的单词的开头 2. 单词的定义是一系列字母、数字、下划线或任何非空白字符的组合 3. 注意，使用该表达式前边必须有一个空格隔开
" wordend"	1. 将索引移动到当前索引指向的单词的末尾 2. 单词的定义是一系列字母、数字、下划线或任何非空白字符的组合 3. 注意，使用该表达式前边必须有一个空格隔开

TIPS：只要结果不产生歧义，关键字可以被缩写，空格也可以省略。例如："+ 5 chars" 可以简写成 "+5c"

在实现中，为了确保表达式为普通字符串，你可以使用 str 或格式化操作来创建一个表达式字符串。下边例子演示了如何删除插入光标前边的一个字符：

```
01.  def backspace(event):
02.      event.widget.delete("%s-1c" % INSERT, INSERT)
     复制代码
```

Marks 用法

Marks（标记）通常是嵌入到 Text 组件文本中的不可见对象。事实上 Marks 是指定字符间的位置，并跟随相应的字符一起移动。Marks 有 INSERT，CURRENT 和 user-defined marks（用户自定义的 Marks）。其中，INSERT 和 CURRENT 是 Tkinter 预定义的特殊 Marks，它们不能够被删除。

INSERT（或 "insert"）用于指定当前插入光标的位置，Tkinter 会在该位置绘制一个闪烁的光标（因此并不是所有的 Marks 都不可见）。

CURRENT（或 "current"）用于指定与鼠标坐标最接近的位置。不过，如果你紧按鼠标任何一个按钮，它会直到你松开它才响应。

你还可以自定义任意数量的 Marks，Marks 的名字是由普通字符串组成，可以是除了空白字符外的任何字符（为了避免歧义，你应该起一个有意义的名字）。使用 mark_set() 方法创建和移动 Marks。

如果你在一个 Mark 标记的位置之前插入或删除文本，那么 Mark 跟着一并移动。删除 Marks 你需要使用 mark_unset() 方法，删除 Mark 周围的文本并不会删除 Mark 本身。

如果有做相关练习的鱼油应该会被 Mark 的很多特性所疑惑，小甲鱼在准备这个内容的时候也很是迷惑，找了不知多少文档.....最后总结为下边几个例子讲解：

例1，Mark 事实上就是索引，用于表示位置：

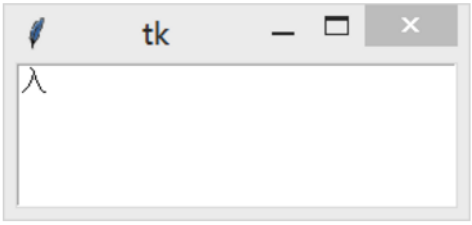
```
text.insert(INSERT, "I love FishC")
text.mark_set("here", "1.2")
text.insert("here", "插")
```
复制代码

例2，如果 Mark 前边的内容发生改变，那么 Mark 的位置也会跟着移动（说白了就是 Mark 会“记住”它后边的那货~）：

```
text.insert(INSERT, "I love FishC")
text.mark_set("here", "1.2")
text.insert("here", "插")
text.insert("here", "入")
```
复制代码

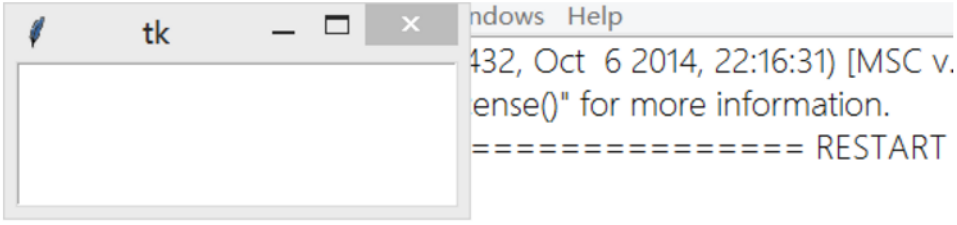
例3，如果 Mark 周围的文本被删除了，Mark 仍然还在噢（只是它后边的那货被删除了，所以它六神无主，只能初始化为 "1.0"）：

```
text.insert(INSERT, "I love FishC")
text.mark_set("here", "1.2")
text.insert("here", "插")

text.delete("1.0", END)
text.insert("here", "入")
```
复制代码



例4，只有 mark_unset() 方法可以解除 Mark 的封印：

```
text.insert(INSERT, "I love FishC")
text.mark_set("here", "1.2")
text.insert("here", "插")

text.mark_unset("here")

text.delete("1.0", END)
text.insert("here", "入")
```
复制代码




```
>>>
Traceback (most recent call last):
  File "C:\Users\佳宇\OneDrive\练习\t8.py", line 15, in <module>
    text.insert("here", "入")
  File "C:\Python34\lib\tkinter\__init__.py", line 3140, in insert
    self.tk.call((self._w, 'insert', index, chars) + args)
_tkinter.TclError: bad text index "here"
>>>
```

看，其实也没有那么难嘛~

好，讲最后一点，我们看到了，默认插入内容到 Mark，是插入到它的左侧（就是说插入一个字符的话，Mark 向后移动了一个字符的位置）。那能不能插入到 Mark 的右侧呢？其实是可以的，通过 mark_gravity() 方法就可以实现。

例5（对比例2）：

```
01.  text.insert(INSERT, "I love FishC")
02.
03.  text.mark_set("here", "1.2")
04.  text.mark_gravity("here", LEFT)
05.
06.  text.insert("here", "插")
07.  text.insert("here", "入")
     复制代码
```

Tags 用法

Tags（标签）通常用于改变 Text 组件中内容的样式和功能。你可以修改文本的字体、尺寸和颜色。另外，Tags 还允许你将文本、嵌入的组件和图片与键盘和鼠标等事件相关联。除了 user-defined tags（用户自定义的 Tags），还有一个预定义的特殊 Tag：SEL。

SEL（或 "sel"）用于表示对应的选中内容（如果有的话）。

你可以自定义任意数量的 Tags，Tags 的名字是由普通字符串组成，可以是除了空白字符外的任何字符。另外，任何文本内容都支持多个 Tags 描述，任何 Tag 也可以用于描述多个不同的文本内容。

为指定文本添加 Tags 可以使用 tag_add() 方法：

```
01. text.insert(INSERT, "I love FishC.com!")
02. text.tag_add("tag1", "1.7", "1.12", "1.14")
03. text.tag_config("tag1", background="yellow", foreground="red")
    复制代码
```

如上，使用 tag_config() 方法可以设置 Tags 的样式。下边罗列了 tag_congif() 方法可以使用的选项：

选项	含义
background	1. 指定该 Tag 所描述的内容的背景颜色 2. 注意：bg 并不是该选项的缩写，在这里 bg 被解释为 bgstipple 选项的缩写
bgstipple	1. 指定一个位图作为背景，并使用 background 选项指定的颜色填充 2. 只有设置了 background 选项该选项才会生效 3. 默认的标准位图有：'error', 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead', 'question' 和 'warning'
borderwidth	1. 指定文本框的宽度 2. 默认值是 0 3. 只有设置了 relief 选项该选项才会生效 4. 注意：该选项不能使用 bd 缩写
fgstipple	1. 指定一个位图作为前景色 2. 默认的标准位图有：'error', 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead', 'question' 和 'warning'
font	指定该 Tag 所描述的内容使用的字体
foreground	1. 指定该 Tag 所描述的内容的前景色 2. 注意：fg 并不是该选项的缩写，在这里 fg 被解释为 fgstipple 选项的缩写
justify	1. 控制文本的对齐方式 2. 默认是 LEFT（左对齐），还可以选择 RIGHT（右对齐）和 CENTER（居中） 3. 注意：需要将 Tag 指向该行的第一个字符，该选项才能生效
lmargin1	1. 设置 Tag 指向的文本块第一行的缩进 2. 默认值是 0 3. 注意：需要将 Tag 指向该文本块的第一个字符或整个文本块，该选项才能生效
lmargin2	1. 设置 Tag 指向的文本块除了第一行其他行的缩进 2. 默认值是 0 3. 注意：需要将 Tag 指向整个文本块，该选项才能生效
offset	1. 设置 Tag 指向的文本相对于基线的偏移距离 2. 可以控制文本相对于基线是升高（正数值）或者降低（负数值） 3. 默认值是 0

overstrike	1. 在 Tag 指定的文本范围画一条删除线 2. 默认值是 False
relief	1. 指定 Tag 对应范围的文本的边框样式 2. 可以使用的值有：SUNKEN, RAISED, GROOVE, RIDGE 或 FLAT 3. 默认值是 FLAT（没有边框）
rmargin	1. 设置 Tag 指向的文本块右侧的缩进 2. 默认值是 0
spacing1	1. 设置 Tag 所描述的文本块中每一行与上方的空白间隔 2. 注意：自动换行不算 3. 默认值是 0
spacing2	1. 设置 Tag 所描述的文本块中自动换行的各行间的空白间隔 2. 注意：换行符（'\n'）不算 3. 默认值是 0
spacing3	1. 设置 Tag 所描述的文本块中每一行与下方的空白间隔 2. 注意：自动换行不算 3. 默认值是 0
tabs	1. 定制 Tag 所描述的文本块中 Tab 按键的功能 2. 默认 Tab 被定义为 8 个字符的宽度 3. 你还可以定义多个制表位：tabs=('3c', '5c', '12c') 表示前 3 个 Tab 宽度分别为 3厘米，5厘米，12厘米，接着的 Tab 按照最后两个的差值计算，即：19厘米，26厘米，33厘米 4. 你应该注意到了，它上边 'c' 的含义是“厘米”而不是“字符”，还可以选择的单位有 'i'（英寸），'m'（毫米）和 'p'（DPI，大约是 '1i' 等于 '72p'） 5. 如果是一个整型值，则单位是像素
underline	1. 该选项设置为 True 的话，则 Tag 所描述的范围内文本将被画上下划线 2. 默认值是 False
wrap	1. 设置当一行文本的长度超过 width 选项设置的宽度时，是否自动换行 2. 该选项的值可以是：NONE（不自动换行），CHAR（按字符自动换行）和 WORD（按单词自动换行）

如果你对同一个范围内的文本加上多个 Tags，并且设置相同的选项，那么新创建的 Tag 样式会覆盖比较旧的 Tag：

```
text.tag_config("tag1", background="yellow", foreground="red")  # 旧的 Tag
text.tag_config("tag2", foreground="blue")  # 新的 Tag

# 那么新创建的 Tag2 会覆盖比较旧的 Tag1 的相同选项
# 注意，与下边的调用顺序没有关系
text.insert(INSERT, "I love FishC.com!", ("tag2", "tag1"))
```

复制代码

你或许想控制 Tags 间的优先级，这可以实现吗？小甲鱼来告诉你，完全没有问题！你可以使用 tag_raise() 和 tag_lower() 方法来提高和降低某个 Tag 的优先级：

```
text.tag_config("tag1", background="yellow", foreground="red")
text.tag_config("tag2", foreground="blue")

text.tag_lower("tag2")

text.insert(INSERT, "I love FishC.com!", ("tag2", "tag1"))
```

复制代码

另外 Tags 还支持事件绑定，使用的是 tag_bind() 的方法。

下边例子中我们将文本（"FishC.com"）与鼠标事件进行绑定，当鼠标进入该文本段的时候，鼠标样式切换为 "arrow" 形态，离开文本段的时候切换回 "xterm" 形态。当触发鼠标“左键点击操作”事件的时候，使用默认浏览器打开鱼C工作室的首页（www.fishc.com）：

```
from tkinter import *
import webbrowser

root = Tk()

text = Text(root, width=30, height=5)
text.pack()

text.insert(INSERT, "I love FishC.com!")

text.tag_add("link", "1.7", "1.16")
text.tag_config("link", foreground="blue", underline=True)

def show_hand_cursor(event):
    text.config(cursor="arrow")

def show_arrow_cursor(event):
    text.config(cursor="xterm")

def click(event):
    webbrowser.open("http://www.fishc.com")

text.tag_bind("link", "<Enter>", show_hand_cursor)
text.tag_bind("link", "<Leave>", show_arrow_cursor)
text.tag_bind("link", "<Button-1>", click)

mainloop()
```

复制代码

参数

Text(master=None, **options) (class)

master -- 父组件

**options -- 组件选项，下方表格详细列举了各个选项的具体含义和用法：

选项	含义
autoseparators	1. 指定实现“撤销”操作的时候是否自动插入一个“分隔符”（用于分隔操作记录） 2. 默认值是 True 3. 详见上方用法【“撤销”和“恢复”操作】
background	1. 设置 Text 组件的背景颜色 2. 注意：通过使用 Tags 可以使 Text 组件中的文本支持多种背景颜色显示（请参考上方【Tags 用法】）
bg	跟 background 一样
borderwidth	1. 设置 Entry 的边框宽度 2. 默认值是 1 像素
bd	跟 borderwidth 一样
cursor	1. 指定当鼠标在 Text 组件上飘过的时候的鼠标样式 2. 默认值由系统指定

exportselection	1. 指定选中的文本是否可以被复制到剪贴板 2. 默认值是 True 3. 可以修改为 False 表示不允许复制文本
font	1. 设置 Text 组件中文本的默认字体 2. 注意：通过使用 Tags 可以使 Text 组件中的文本支持多种字体显示（请参考上方【Tags 用法】）
foreground	1. 设置 Text 组件中文本的颜色 2. 注意：通过使用 Tags 可以使 Text 组件中的文本支持多种颜色显示（请参考上方【Tags 用法】）
fg	跟 foreground 一样
height	1. 设置 Text 组件的高度 2. 注意：单位是行数，不是像素噢
highlightbackground	1. 指定当 Text 组件没有获得焦点的时候高亮边框的颜色 2. 默认值由系统指定
highlightcolor	1. 指定当 Text 组件获得焦点的时候高亮边框的颜色 2. 默认值由系统指定
highlightthickness	1. 指定高亮边框的宽度 2. 默认值是 0
insertbackground	1. 设置插入光标的颜色 2. 默认是 BLACK（或 "black"）
insertborderwidth	1. 设置插入光标的边框宽度 2. 默认值是 0 3. 提示：你得设置 insertwidth 选项为比较大的数值才能看出来噢
insertofftime	1. 该选项控制光标的闪烁频率（灭） 2. 单位是毫秒
insertontime	1. 该选项控制光标的闪烁频率（亮） 2. 单位是毫秒

	2. 单位是毫秒
insertwidth	1. 指定光标的宽度 2. 默认值是 2 像素
maxundo	1. 设置允许“撤销”操作的最大次数 2. 默认值是 0 3. 设置为 -1 表示不限制
padx	1. 指定水平方向上的额外间距（内容和边框间） 2. 默认值是 1
pady	1. 指定垂直方向上的额外间距（内容和边框间） 2. 默认值是 1
relief	1. 指定边框样式 2. 默认值是 SUNKEN 3. 其他可以选择的值是 FLAT，RAISED，GROOVE 和 RIDGE
selectbackground	1. 指定被选中文本的背景颜色 2. 默认值由系统指定
selectborderwidth	1. 指定被选中文本的边框宽度 2. 默认值是 0
selectforeground	1. 指定被选中文本的字体颜色 2. 默认值由系统指定
setgrid	1. 指定一个布尔类型的值，确定是否启用网格控制 2. 默认值是 False
spacing1	1. 指定 Text 组件的文本块中每一行与上方的空白间隔 2. 注意：自动换行不算 3. 默认值是 0
spacing2	1. 指定 Text 组件的文本块中自动换行的各行间的空白间隔 2. 注意：换行符（'\n'）不算 3. 默认值是 0
spacing3	1. 指定 Text 组件的文本中每一行与下方的空白间隔 2. 注意：自动换行不算 3. 默认值是 0

state	1. 默认情况下 Text 组件响应键盘和鼠标事件（NORMAL） 2. 如果将该选项的值设置为 DISABLED，那么上述响应就不会发生，并且你无法修改里边的内容
tabs	1. 定制 Tag 所描述的文本块中 Tab 按键的功能 2. 默认 Tab 被定义为 8 个字符的宽度 3. 你还可以定义多个制表位：tabs=('3c', '5c', '12c') 表示前 3 个 Tab 宽度分别为 3厘米，5厘米，12厘米，接着的 Tab 按照最后两个的差值计算，即：19厘米，26厘米，33厘米 4. 你应该注意到了，它上边 'c' 的含义是“厘米”而不是“字符”，还可以选择的单位有 'i'（英寸），'m'（毫米）和 'p'（DPI，大约是 '1i' 等于 '72p'） 5. 如果是一个整型值，则单位是像素
takefocus	1. 指定使用 Tab 键可以将焦点移动到 Text 组件中 2. 默认是开启的，可以将该选项设置为 False 避免焦点在此 Text 组件中
undo	1. 该选项设置为 True 开启“撤销”功能 2. 该选项设置为 False 关闭“撤销”功能 3. 默认值是 False
width	1. 设置 Text 组件的宽度 2. 注意：单位是字符数，因此 Text 组件的实际宽度还取决于字体的大小
wrap	1. 设置当一行文本的长度超过 width 选项设置的宽度时，是否自动换行 2. 该选项的值可以是：NONE（不自动换行），CHAR（按字符自动换行）和 WORD（按单词自动换行）
xscrollcommand	1. 与 scrollbar（滚动条）组件相关联（水平方向） 2. 使用方法可以参考：Scrollbar 组件
yscrollcommand	1. 与 scrollbar（滚动条）组件相关联（垂直方向） 2. 使用方法可以参考：Scrollbar 组件

方法

bbox(index)
-- 返回给定索引指定的字符的边界框
-- 返回值是一个 4 元组：(x, y, width, height)
-- 如果该字符是不可见的，那么返回 None
-- 注意：只有当 Text 组件被更新的时候该方法才有效，可以使用 update_idletasks() 方法先更新 Text 组件

compare(index1, op, index2)
-- 返回对比 index1 和 index2 指定的两个字符的结果
-- op 是操作符：'<', '<=', '==', '>=', '>' 或 '!='（不支持 Python 的 '<>' 操作符）
-- 返回布尔类型的值表示对比的结果

debug(boolean=None)
-- 开启或关闭 Debug 状态

delete(start, end=None)
-- 删除给定范围的文本或嵌入对象
-- 如果在给定范围内有任何 Marks 标记的位置，则将 Marks 移动到 start 参数开始的位置

dlineinfo(index)
-- 返回给定索引指定的字符所在行的边界框
-- 返回值是一个 5 元组：(x, y, width, height, offset)，offset 表示从该行的顶端到基线的偏移
-- 如果该行不可见，则返回 None
-- 注意：只有当 Text 组件被更新的时候该方法才有效，可以使用 update_idletasks() 方法先更新 Text 组件

dump(index1, index2=None, command=None, **kw)
-- 返回 index1 和 index2 之间的内容
-- 返回的值是一个由 3 元组（关键词，值，索引）组成的列表，关键词参数的顺序为：all, image, mark, tag, text, window
-- 默认关键词是 'all'，表示全部关键词均为选中状态
-- 如果需要筛选个别关键词，可以用 dump(index1, index2, image=True, text=True) 这样的形式调用
-- 如果指定了 command 函数，那么会为列表中的每一个三元组作为参数调用一次该函数（这种情况下，dump() 不返回值）

edit_modified(arg=None)
-- 该方法用于查询和设置 modified 标志（该标志用于追踪 Text 组件的内容是否发生变化）
-- 如果不指定 arg 参数，那么返回 modified 标志是否被设置
-- 你可以传递显式地使用 True 或 False 作为参数来设置或清除 modified 标志
-- 任何代码或用户的插入或删除文本操作，“撤销”或“恢复”操作，都会是的 modified 标志被设置

edit_redo(self)
-- “恢复”上一次的“撤销”操作
-- 如果 undo 选项为 False，该方法无效
-- 详见上方用法【“撤销”和“恢复”操作】

edit_reset()
-- 清空存放操作记录的栈

edit_separator()
-- 插入一个“分隔符”到存放操作记录的栈中，用于表示已经完成一次完整的操作
-- 如果 undo 选项为 False，该方法无效
-- 详见上方用法【“撤销”和“恢复”操作】

edit_undo()
-- 撤销最近一次操作
-- 如果 undo 选项为 False，该方法无效
-- 详见上方用法【“撤销”和“恢复”操作】

get(index1, index2=None)
-- 返回 index1 到 index2（不包含）之间的文本
-- 如果 index2 参数忽略，则返回一个字符
-- 如果包含 image 和 window 的嵌入对象，均被忽略
-- 如果包含有多行文本，那么自动插入换行符（'\n'）

image_cget(index, option)
-- 返回 index 参数指定的嵌入 image 对象的 option 选项的值
-- 如果给定的位置没有嵌入 image 对象，则抛出 TclError 异常

image_configure(index, **options)
-- 修改 index 参数指定的嵌入 image 对象的一个或多个 option 选项的值
-- 如果给定的位置没有嵌入 image 对象，则抛出 TclError 异常

image_create(index, cnf={}, **kw)
-- 在 index 参数指定的位置嵌入一个 image 对象
-- 该 image 对象必须是 Tkinter 的 PhotoImage 或 BitmapImage 实例
-- 可选选项 align：设定此图像的垂直对齐，可以是 TOP、CENTER、BOTTOM 或 BASELINE
-- 可选选项 image：PhotoImage 或 BitmapImage 对象
-- 可选选项 name：你可以为该图像实例命名，如果你忽略此选项，那么 Tkinter 会自动为其取一个独一无二的名字。
-- 可选选项 padx：设置水平方向上的额外间距
-- 可选选项 pady：设置垂直方向上的额外间距

image_names()
-- 返回 Text 组件中嵌入的所有 image 对象的名字

index(index)
-- 将 index 参数指定的位置以 "line.column" 的索引形式返回
-- index 参数支持任何格式的索引

insert(index, text, *tags)
-- 在 index 参数指定的位置插入字符串
-- 可选参数 tags 用于指定文本的样式
-- 详见上方【Tags 用法】

mark_gravity(self, markName, direction=None)
-- 设置 Mark 的方向，可以是 LEFT 或 RIGHT（默认是 RIGHT，即如果在 Mark 处插入文本的话，Mark 将发生相应的移动以保持在插入文本的右侧）
-- 如果设置为 LEFT，那么在 Mark 处插入文本并不会移动 Mark（因为 Mark 在插入文本的左侧）
-- 如果忽略 direction 参数，则返回指定 Mark 的方向
-- 详见上方【Marks 用法】

mark_names()
-- 返回 Text 组件中所有 Marks 的名字
-- 包括两个特殊 Mark：INSERT 和 CURRENT
-- 注意：END 是特殊的索引，不是 Mark

mark_next(index)
-- 返回在 index 指定的位置后边的一个 Mark 的名字
-- 如果不存在则返回空字符串

mark_previous(index)
-- 返回在 index 指定的位置前边的一个 Mark 的名字
-- 如果不存在则返回空字符串

mark_set(markName, index)
-- 移动 Mark 到 index 参数指定的位置
-- 如果 markName 参数指定的 Mark 不存在，则创建一个新的 Mark

mark_unset(*markNames)
-- 删除 markNames 指定的 Marks
-- 不能删除预定义的 INSERT 和 CURRENT

replace(index1, index2, chars, *args)
-- 将 index1 到 index2 之间的内容替换为 chars 参数指定的字符串
-- 如果需要为替换的内容添加 Tag，可以在 args 参数指定 Tag
-- 详见上方【Tags 用法】

scan_dragto(x, y)
-- 详见下方 scan_mark(x, y)

scan_mark(x, y)
-- 使用这种方式来实现 Text 组件内容的滚动
-- 需要将鼠标按钮事件以及鼠标当前位置绑定到 scan_mark(x, y) 方法，然后将 <motion> 事件及当前鼠标位置绑定到 scan_dragto(x, y) 方法，就可以实现 Text 组件的内容在当前位置和 scan_mark(x, y) 指定的位置 (x, y) 之间滚动

search(pattern, index, stopindex=None, forwards=None, backwards=None, exact=None, regexp=None, nocase=None, count=None)
-- 从 index 开始搜索 pattern，到 stopindex 结束（不指定表示搜索到末尾）
-- 如果成功找到，以 "line.column" 返回第一个匹配的字符；否则返回空字符串
-- forwards 参数设置为 True 表示向前（->）搜索
-- backwards 参数设置为 True 表示向后（<-）搜索
-- exact 参数设置为 True 表示搜索与 pattern 完全匹配的结果
-- regexp 参数设置为 True，则 pattern 被解释为 Tcl 格式的正则表达式
-- nocase 参数设置为 True 是忽略大小写，默认是区分大小写的搜索
-- count 参数指定为一个 IntVar 的 Tkinter 变量，用于存放当找到匹配的字符个数（如果匹配结果中没有嵌入的 image 或 window 对象的话，一般该值等于 pattern 的字符个数）

see(index)
-- 滚动内容，确保 index 指定的位置可见

tag_add(tagName, index1, index2=None)
-- 为 index1 到 index2 之间的内容添加一个 Tag（tagName 参数指定）
-- 如果 index2 参数忽略，则单独为 index1 指定的内容添加 Tag
-- 详见上方【Tags 用法】

tag_bind(tagName, sequence, func, add=None)
-- 为 Tag 绑定事件
-- 详见上方【Tags 用法】

tag_cget(tagName, option)
-- 返回 tagName 指定的 option 选项的值

tag_config(tagName, cnf=None, **kw)
-- 跟 tag_configure(tagName, cnf=None, **kw) 一样

tag_configure(tagName, cnf=None, **kw)
-- 设置 tagName 的选项
-- 详见上方【Tags 用法】

tag_delete(*tagNames)
-- 删除 tagNames 指定的 Tags

tag_lower(tagName, belowThis=None)
-- 降低 Tag 的优先级
-- 如果 belowThis 参数不为空，则表示 tagName 需要比 belowThis 指定的 Tag 优先级更低
-- 详见上方【Tags 用法】

tag_names(index=None)
-- 如果不带参数，表示返回 Text 组件中所有 Tags 的名字
-- index 参数表示返回该位置上所有的 Tags 的名字

tag_nextrange(tagName, index1, index2=None)
-- 在 index1 到 index2 的范围内第一个 tagName 的位置
-- 如果没有则返回空字符串

tag_prevrange(tagName, index1, index2=None)
-- tag_nextrange() 的反向查找，也就是查找范围是 index2 到 index1

tag_raise(tagName, aboveThis=None)
-- 提高 Tag 的优先级
-- 如果 aboveThis 参数不为空，则表示 tagName 需要比 aboveThis 指定的 Tag 优先级更高
-- 详见上方【Tags 用法】

tag_ranges(tagName)
-- 返回所有 tagName 指定的文本，并将它们的范围以列表的形式返回

tag_remove(tagName, index1, index2=None)
-- 删除 index1 到 index2 之间所有的 tagName
-- 如果忽略 index2 参数，那么只删除 index1 指定的那个字符的 tagName（如果有的话）

tag_unbind(tagName, sequence, funcid=None)
-- 解除与 tagName 绑定的事件（sequence 指定）

window_cget(index, option)
-- 返回 index 参数指定的嵌入 window 对象的 option 选项的值
-- 如果给定的位置没有嵌入 window 对象，则抛出 TclError 异常

window_config(index, cnf=None, **kw)
-- 跟 window_configure(index, cnf=None, **kw) 一样

window_configure(index, cnf=None, **kw)
-- 修改 index 参数指定的嵌入 window 对象的一个或多个 option 选项的值
-- 如果给定的位置没有嵌入 window 对象，则抛出 TclError 异常

window_create(index, **options)
-- 在 index 参数指定的位置嵌入一个 window 对象
-- 支持两种方式在 Text 组件中嵌入 window 对象：请看下方 create 选项和 window 选项的描述
-- 可选选项 align：设定此图像的垂直对齐，可以是 TOP、CENTER、BOTTOM 或 BASELINE
-- 可选选项 create：指定一个回调函数用于创建嵌入的 window 组件，该函数没有参数，并且必须创建 Text 的子组件并返回
-- 可选选项 padx：设置水平方向上的额外间距
-- 可选选项 pady：设置垂直方向上的额外间距
-- 可选选项 stretch：该选项控制当行的高度大于嵌入组件的高度时，嵌入组件是否延伸。默认值是 False，表示组件保持原形；设置为 True 表示将该组件垂直部分延伸至行的高度
-- 可选选项 window：指定一个已经创建好的 window 组件，该组件必须是 Text 组件的子组件

window_names()
-- 返回 Text 组件中嵌入的所有 window 对象的名字

xview(*args)
-- 该方法用于在水平方向上滚动 Text 组件的内容，一般通过绑定 Scollbar 组件的 command 选项来实现（具体操作参考：Scrollbar）
-- 如果第一个参数是 MOVETO，则第二个参数表示滚动到指定的位置：0.0 表示最左端，1.0 表示最右端
-- 如果第一个参数是 SCROLL，则第二个参数表示滚动的数量，第三个参数表示滚动的单位（可以是 UNITS 或 PAGES），例如：xview(SCROLL, 3, UNITS) 表示向右滚动三行

xview_moveto(fraction)
-- 跟 xview(MOVETO, fraction) 一样

xview_scroll(number, what)
-- 跟 xview(SCROLL, number, what) 一样

yview(*args)
-- 该方法用于在垂直方向上滚动 Text 组件的内容，一般通过绑定 Scollbar 组件的 command 选项来实现（具体操作参考：Scrollbar）
-- 如果第一个参数是 MOVETO，则第二个参数表示滚动到指定的位置：0.0 表示最顶端，1.0 表示最底端
-- 如果第一个参数是 SCROLL，则第二个参数表示滚动的数量，第三个参数表示滚动的单位（可以是 UNITS 或 PAGES），例如：yview(SCROLL, 3, PAGES) 表示向下滚动三页

yview_moveto(fraction)
-- 跟 yview(MOVETO, fraction) 一样

yview_scroll(number, what)
-- 跟 yview(SCROLL, number, what) 一样