События и привязки в Tkinter

https://web.archive.org/web/20201111211515id_/https://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

https://python-course.eu/tkinter/events-and-binds-in-tkinter.php

Введение

Приложение Tkinter большую часть времени работает внутри цикла событий, вход в который осуществляется с помощью метода mainloop. Он ждет, когда события произойдут. События могут быть нажатия клавиш или операции мыши пользователем. Tkinter предоставляет механизм, позволяющий программисту работать с событиями. Для каждого виджета можно привязать функции и методы Python к событию.
widget.bind(событие, обработчик)
Если определенное событие происходит в виджете, вызывается функция «обработчик» с объектом события. описывая событие.

```
#!/usr/bin/python3
# write tkinter as Tkinter to be Python 2.x compatible
from tkinter import *
def hello(event):
   print("Single Click, Button-l")
def quit(event):
   print("Double Click, so let's stop")
   import sys; sys.exit()

widget = Button(None, text='Mouse Clicks')
widget.pack()
widget.bind('<Button-1>', hello)
widget.bind('<Double-1>', quit)
widget.mainloop()
```

Давайте рассмотрим еще один простой пример, который показывает, как использовать событие движения, т.е. если мышь перемещается внутри виджета:

```
from tkinter import *

def motion(event):
  print("Mouse position: (%s %s)" % (event.x, event.y))
  return

master = Tk()
whatever_you_do = "Whatever you do will be insignificant, but it is very important that you do
it.\n(Mahatma Gandhi)"
msg = Message(master, text = whatever_you_do)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.bind('<Motion>',motion)
msg.pack()
mainloop()
```

Каждый раз, когда мы перемещаем мышь в виджете «Сообщение», положение указателя мыши будет напечатано. Когда мы покидаем этот виджет, функция motion() больше не вызывается.

События

Tkinter использует так называемые последовательности событий, чтобы позволить пользователю определить, какие события, как конкретные, так и общие, он или она хочет привязать к обработчикам. Последовательность событий задается в виде строки с использованием следующего синтаксиса:

<modifier-type-detail>

Поле типа является неотъемлемой частью спецификатора события, тогда как поля «модификатор» и «подробности» не являются обязательными и во многих случаях не учитываются. Они используются для предоставления дополнительной информации для выбранного «типа». «Тип» события описывает тип события, которое должно быть привязано, например, такие действия, как щелчки мышью, нажатия клавиш или виджет, получивший фокус ввода.

Событие	Описание
<Button>	Кнопка мыши нажата, когда указатель мыши находится над виджетом. Детальная часть указывает, какая кнопка, например, левая кнопка мыши определяется событием <Button-1>, средняя кнопка — <Button-2>, а крайняя правая кнопка мыши — <Button-3>. <Button-4> определяет событие прокрутки вверх на мышах с колесиком, а <Button-5> — прокрутку вниз. Если вы нажмете кнопку мыши над виджетом и будете удерживать ее нажатой, Tkinter автоматически «схватит» указатель мыши. Дальнейшие события мыши, такие как события Motion и Release, будут отправлены текущему виджету, даже если мышь перемещается за пределы текущего виджета. Текущая позиция относительно виджета указателя мыши указывается в свойствах x и y объекта события, передаваемого в функцию обработчика события. Вы можете использовать ButtonPress вместо Button и <1> — все это синонимы.
<Motion>	Мышь перемещается с зажатой кнопкой мыши. Чтобы указать левую, среднюю или правую кнопку мыши, используйте <B1-Motion>, <B2-Motion> и <B3-Motion> соответственно. Текущая позиция указателя мыши указывается в свойствами x и y объекта события, передаваемого в функцию обработчика событий, т. е. event.x, event.y.
<ButtonRelease>	Событие, если кнопка отпущена. Чтобы указать левую, среднюю или правую кнопку мыши, используйте <ButtonRelease-1>, <ButtonRelease-2> и <ButtonRelease-3> соответственно. Текущая позиция указателя мыши указывается в свойствах x и y объекта события, передаваемого в функцию обработчика события, т. е. event.x, event.y.
<Double-Button>	Подобно событию Button, см. выше, но кнопка щелкается дважды, а не один раз. Чтобы указать левую, среднюю или правую кнопку мыши, используйте <Double-Button-1>, <Double-Button-2> и <Double-Button-3> соответственно.

	Вы можете использовать Double или Triple в качестве префиксов. Обратите внимание, что если вы выполняете привязку как к одиночному щелчку (<Button-1>), так и к двойному щелчку (<Double-Button-1>), будут вызываться обе привязки.
<Enter>	Указатель мыши вошел в виджет. Внимание: Это не означает, что пользователь нажал клавишу Enter!. <Return> используется для этой цели.
<Leave>	Указатель мыши покинул виджет.
<FocusIn>	Фокус клавиатуры был перемещен на этот виджет или на дочерний элемент этого виджета.
<FocusOut>	Фокус клавиатуры был перемещен с этого виджета на другой виджет.
<Return>	Пользователь нажал клавишу Enter. Вы можете привязать практически все клавиши на клавиатуре: специальные клавиши: Cancel (клавиша Break), BackSpace, Tab, Return (клавиша Enter), Shift_L (любая клавиша Shift), Control_L (любая клавиша Control), Alt_L (любая клавиша клавиша Alt), Pause, Caps_Lock, Escape, предыдущий (Page Up), следующий (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num_Lock и Scroll_Lock.
<Key>	Пользователь нажал любую клавишу. Ключ предоставляется в свойстве char объекта события, переданного в функцию обработчика события (это пустая строка для специальных ключей).
a	Пользователь набрал клавишу «a». Большинство печатных символов можно использовать как есть. Исключениями являются пробел (<space>) и меньше (<less>). Обратите внимание, что 1 — это привязка клавиатуры, а <1> — привязка кнопки.
<Shift-Up>	Пользователь нажал стрелку вверх, удерживая нажатой клавишу Shift. Вы можете использовать такие префиксы, как Alt, Shift и Control.
<Configure>	Изменился размер виджета. Новый размер предоставляется в атрибутах ширины и высоты объекта события, передаваемого в функцию обработчика события. На некоторых платформах это может означать, что местоположение изменилось.

Объект события

Объект события — это стандартный экземпляр объекта Python с рядом атрибутов, описывающих событие.

Атрибуты события

- **Widget -** Виджет, сгенерировавший это событие. Это действительный экземпляр виджета Tkinter, а не имя. Этот атрибут устанавливается для всех событий.
- **x, y** - Текущая позиция мыши в пикселях.
- **x_root, y_root** -Текущая позиция мыши относительно левого верхнего угла экрана в пикселях.
- **char** - Код символа (только события клавиатуры) в виде строки.
- **keysym -**Символ клавиши (только события клавиатуры).
- **keycode -** Код клавиши (только события клавиатуры).

- **num** - Номер кнопки (только события кнопки мыши).
- **width, height -** Новый размер виджета в пикселях (только для событий конфигурации).
- **type -** Тип события.

Из соображений переносимости вы должны придерживаться char , height , width , x , y , x_root , y_root и widget . Если, конечно, вы точно не знаете, что делаете…

Привязки экземпляра и класса

Метод привязки , который мы использовали в приведенном выше примере, создает привязку экземпляра. Это означает, что привязка применяется только к одному виджету; если вы создадите новые фреймы, они не будут наследовать привязки.

Но Tkinter также позволяет создавать привязки на уровне класса и приложения; на самом деле, вы можете создавать привязки на четырех разных уровнях:

- экземпляр виджета, используя **bind** .
- окно верхнего уровня виджета (**Toplevel** или **root**), также используя **bind** .
- класс виджета, используя **bind_class** (это используется Tkinter для предоставления стандартных привязок).
- все приложение, используя **bind_all** .

Например, вы можете использовать **bind_all** , чтобы создать привязку для клавиши **F1**, чтобы вы могли оказывать помощь везде в приложении. Но что произойдет, если вы создадите несколько привязок для одного и того же ключа или предоставите перекрывающиеся привязки?

Во-первых, на каждом из этих четырех уровней Tkinter выбирает «наиболее близкое совпадение» из доступных привязок. Например, если вы создаете привязки экземпляров для событий <Key> и <Return> , при нажатии клавиши Enter будет вызываться только вторая привязка .

Однако если вы добавите привязку <Return> к виджету верхнего уровня, будут вызваны обе привязки. Tkinter сначала вызывает наилучшую привязку на уровне экземпляра, затем наилучшую привязку на уровне окна верхнего уровня, затем наилучшую привязку на уровне класса (которая часто является стандартной привязкой) и, наконец, наилучшую доступную привязку на уровне приложения. Таким образом, в крайнем случае одно событие может вызвать четыре обработчика событий.

Распространенной причиной путаницы является попытка использовать привязки для переопределения поведения стандартного виджета по умолчанию. Например, предположим, что вы хотите отключить клавишу Enter в текстовом виджете, чтобы пользователи не могли вставлять новые строки в текст. Может быть, следующее поможет?

```
def  ignore (event):
    pass
text.bind( "<Return>" , ignore)
```

или, если вы предпочитаете однострочники:

```
text.bind( "<Return>" , lambda e: None)
```

(используемая здесь лямбда- функция принимает один аргумент и возвращает None)

К сожалению, новая строка по-прежнему вставляется, поскольку приведенная выше привязка применяется только к уровню экземпляра, а стандартное поведение обеспечивается привязками уровня класса.

Вы можете использовать метод **bind_class** для изменения привязок на уровне класса, но это изменит поведение всех текстовых виджетов в приложении. Более простое решение — запретить Tkinter передавать событие другим обработчикам; просто верните строку «break» из вашего обработчика событий:

```
def  ignore (событие):
    return  "break"
text.bind ( "<Return>" , ignore)
```

или

```
text.bind( "<Return>" , lambda e: "break" )
```

Кстати, если вы действительно хотите изменить поведение всех текстовых виджетов в вашем приложении, вот как использовать метод bind_class :

```
top.bind_class("Text", "<Return>", lambda e: None)
```

Но есть много причин, почему этого делать не стоит. Например, это полностью испортит ситуацию в тот день, когда вы захотите расширить свое приложение каким-нибудь классным маленьким компонентом пользовательского интерфейса, который вы скачали из сети. Лучше используйте свою собственную специализацию текстового виджета и сохраните привязки Tkinter по умолчанию:

```
class MyText(Text):
   def __init__(self, master, **kw):
      apply(Text.__init__, (self, master), kw)
      self.bind("<Return>", lambda e: "break")
```

Протоколы

В дополнение к привязкам событий Tkinter также поддерживает механизм, называемый обработчиками протоколов. Здесь термин протокол относится к взаимодействию между приложением и оконным менеджером. Наиболее часто используемый протокол называется WM_DELETE_WINDOW и используется для определения того, что происходит, когда пользователь явно закрывает окно с помощью оконного менеджера.

Вы можете использовать метод протокола , чтобы установить обработчик для этого протокола (виджет должен быть корневым или виджетом верхнего уровня):

```
widget.protocol("WM_DELETE_WINDOW", handler)
```

После того, как вы установили свой собственный обработчик, Tkinter больше не будет автоматически закрывать окно. Вместо этого вы могли бы, например, отобразить окно сообщения с вопросом, следует ли сохранить текущие данные, или, в некоторых случаях,

просто проигнорировать запрос. Чтобы закрыть окно из этого обработчика, просто вызовите метод уничтожения окна (**destroy**):

Захват событий уничтожения

```
from Tkinter import *
import tkMessageBox

def callback():
   if tkMessageBox.askokcancel("Quit", "Do you really wish to quit?"):
      root.destroy()

root = Tk()
root.protocol("WM_DELETE_WINDOW", callback)

root.mainloop()
```

Обратите внимание, что даже если вы не зарегистрируете обработчик WM_DELETE_WINDOW в окне верхнего уровня, само окно будет уничтожено как обычно (контролируемым образом, в отличие от X). Однако, начиная с Python 1.5.2, Tkinter не уничтожает соответствующую иерархию экземпляров виджетов, поэтому рекомендуется всегда регистрировать обработчик самостоятельно:

```
top = Toplevel(...)

# убедитесь, что экземпляры виджета удалены

top.protocol("WM_DELETE_WINDOW", top.destroy)
```

Будущие версии Tkinter, скорее всего, будут делать это по умолчанию.

Другие протоколы

Протоколы диспетчера окон изначально были частью системы X Window (они определены в документе под названием Inter-Client Communication Conventions Manual или ICCCM). На этой платформе вы также можете установить обработчики для других протоколов, таких как WM_TAKE_FOCUS и WM_SAVE_YOURSELF . Подробности смотрите в документации ICCCM.