转自原文：https://zhuanlan.zhihu.com/p/22882167?refer=python-dev

Python与Tcl混合编程 - 在Python中调用Tcl

本系列讨论在Python中使用Tcl脚本，以及如何使用Python对Tcl进行扩展。

Python的文档对Tkinter的介绍，着重于图形界面编程，而对如何使用Tcl语言与Python代码交互，说得并不明确。本文将着重介绍如何在Python中嵌入Tcl。

1. Python中嵌入Tcl

1.1 Hello Tcl!

在Python中使用Tcl，不需要安装任何第三方模块。考虑下面一个简单的程序：

```
import tkinter

interp = tkinter.Tcl()
interp.eval('puts "Hello Tcl!"')
```

我们首先利用TKinter.Tcl()得到一个Tcl解释器对象，然后执行它的eval方法，运行一小段Tcl代码。 然而，如果打算Tcl和Python环境有所交互，eval方法就力不从心了。

1.2 添加命令

注：如果需要更多命令支持的话，可能需要自行查看Lib/tkinter/__init__.py文件。

利用createcommand方法将一个Python函数作为命令添加到Tcl解释器中。

下面一个例子是创建一个将所有参数数字相加的命令。

```
import tkinter
import operator

interp = tkinter.Tcl()

def SumAll(*args):
    #return reduce(operator.add,[int(x) for x in args])
    return args[0] + args[1]

interp.createcommand('sum_all',SumAll)
interp.eval("""
puts [sum_all 1 2 3]
""")
```

注：目前只支持2个参数的createcommand，对于python函数中需要多个参数支持的，需要用户自行包一下。比如：

```
import tkinter
import operator

interp = tkinter.Tcl()

def py_add(a, b):
    return int(a)+int(b) #从tcl侧传递进来的全部是字符，需要进行适当的类型转换后再使用。

def tcl_py_add(*args):
    l = len(args)
    if l != 2:
        print(f'wrong # args: should be "py_add a b"')
        return 'None'
    return py_add(args[0], args[1])

interp.createcommand('py_add',tcl_py_add)
interp.eval("""
puts [py_add 1 2]
""")

interp.eval("""
puts [py_add 1]
""")
```

1.2.1 使用注意

python的命令加入tcl解释器之后，python侧出现的类型check错误信息不会显示在tcl中(一般显示为一个空行)，这有时候会给定位带来非常大的困扰。

一定要注意从tcl侧输入的参数传递到python之后都是字符串，如果原来python的参数是int的需要使用类型转换。

1.3 添加和收集变量

可以在tcl代码运行前，由Python环境向Tcl解释器『注入』一些预先定义的变量，Tcl代码运行完成之后，Python环境可以获得Tcl解释器中定义的所有变量。例如：

```
import Tkinter

interp = Tkinter.Tcl()
interp.setvar(name='FOO', value='2')
interp.eval("""
puts $FOO
set BAR 3
""")
```

```
print interp.getvar('BAR')
```

从解释器环境中删除某个标量（即相当于unset□□使用interp.unsetvar(varName)

Tcl的list与Python的Tuple可相互使用。

```
import Tkinter

interp = Tkinter.Tcl()
interp.setvar('list_a',(1,2))
interp.eval("""
puts [lindex $list_a 1] #2
set v [list a b [list x y ] "c d e  " "  f {g h}"]
""")
print interp.getvar('v') #('a', 'b', ('x', 'y'), 'c d e  ', '  f {g h}')
```

1.4 追溯变量

若想让Python能够得以知晓Tcl解释器中对某个变量的操作，可利用Tcl的trace方法结合Python引入的command□□尽管Tcl本身提供了Tcl_TraceVar方法，但Tkinter不能够使用它）

```
import Tkinter

interp = Tkinter.Tcl()

def Tracer(varname,*args):
    print interp.getvar(varname)

interp.createcommand('python_tracer',Tracer)
interp.eval("trace add variable foo write python_tracer")
interp.eval("""
set foo 1
set foo 2
""")
```

由于设置了一个write trace□当对foo进行写入的时候□Python函数Tracer会得以执行。

1.5 获取结果

eval方法的返回值是当前表达式的值。例如：

```
import Tkinter

interp = Tkinter.Tcl()
interp.eval("""
array set colorcount {
   red   1
   green 5
```

```
   blue  4
   white 9
}

""")

print interp.eval('array names colorcount') #blue white green red
```

不要用这种方法获得简单变量以及list的值。

2. python zip tcl混合

测试程序 :linux:python:pyziptcl.zip

```
#!/usr/bin/python3

import tkinter


import zipfile
import io


fh = open('ab.zip', 'rb')
fbuf = fh.read()    #读取zip二进制文件内容到buffer□注意buffer溢出。
fh.close()

fio = io.BytesIO()    #创建一个文件io

fio.write(fbuf)      #将zip二进制内容写入文件io



# zfile = zipfile.ZipFile('ab.zip', 'r')
zfile = zipfile.ZipFile(fio, 'r')    #直接从文件io处操作，而不是指定硬盘上的文件。
for i in zfile.filelist:
    bin = zfile.read(i.filename)
    line = str(bin, encoding = "utf-8")

    interp = tkinter.Tcl()
    interp.eval(line)
```

运行结果

```
(base) D:\pyzip>python pyzip.py
abc01
abc02
```

```
opq01
opq02
```