知乎 首发于 backtrader源代码解析

backtrader代码第三步——数据源预备知识findowner函数

知了
四川大学 应用数学硕士

结论先行

1. 函数引用都会保存在栈中，栈从下往上分别保存了函数从外往内的调用
2. findowner函数从函数调用的最里层往外一层层，寻找(不是owned和skip)的(指定类cls的对象)
3. 方式是从findowner的各层调用所保存的字典内，寻找self和obj_两个变量，用于比对来判断是否时所找目标

源代码：

```
def findowner(owned, cls, startlevel=2, skip=None):
    # skip this frame and the caller's -> start at 2
    for framelevel in itertools.count(startlevel):
        try:
            frame = sys._getframe(framelevel)
        except ValueError:
            # Frame depth exceeded ... no owner ... break away
            break

        # 'self' in regular code
        self_ = frame.f_locals.get('self', None)
        if skip is not self_:
            if self_ is not owned and isinstance(self_, cls):
                return self_

        # '_obj' in metaclasses
        obj_ = frame.f_locals.get('_obj', None)
        if skip is not obj_:
            if obj_ is not owned and isinstance(obj_, cls):
                return obj_

    return None
```

补充知识一：itertools.count

案例代码：

```
import itertools
import operator


for i in itertools.count(5):
    print(i)
    if i== 8:
        break

print('='*79)

for i in itertools.islice(itertools.count(3), 3):
```

```
count = 0
for iter in itertools.cycle('xyz'):
    if count > 5:
        break
    print(item)
    count+=1

print('='*79)
print(list(itertools.accumulate(range(10))))
print(list(itertools.accumulate(range(1, 5), operator.mul)))
```

代码解读：

1. itertools.count(start， step) 会生成一个迭代器，生成的第一个数是start，然后间隔step生成数据
2. itertools.islice(itertools.count(start, step), num) islice第二个参数num用于控制count的迭代次数
3. itertools.cycle()用于循环一个可迭代对象
4. accumulate对可迭代对象进行累加操作，operator.mul为操作内容

运行结果：

```
5
6
7
8
===============================================================================
3
4
5
===============================================================================
x
y
z
x
y
z
===============================================================================
[0, 1, 3, 6, 10, 15, 21, 28, 36, 45]
[1, 2, 6, 24]
```

补充知识二：sys._getframe([depth])

函数说明：

Return a frame object from the call stack. If optional integer depth is given, return the frame object that many calls below the top of the stack. If that is deeper than the call stack, ValueError is raised. The default for depth is zero, returning the frame at the top of the call stack.

功能：调取栈中存储的函数调用相关的信息。栈中存储的是函数引用的地址，栈顶层为函数的直接调用，往下可理解为调用函数的函数 sys.getframe(i).f_code.co_name表示栈中第i层的函数名称，栈底层函数往下是返回<module> sys.getframe(i).f_locals表示栈中第i层的函数的局部空间

案例代码1：

```
def a():
    n=6
    b()
def b():
    m=5
    c()
def c():
    l=4
    z = 9
    sys._getframe().f_locals['z'] =1
    print('='*79)
    print(z)
    print('='*79)
    print(sys._getframe(0))
    print(sys._getframe(1))
    print(sys._getframe(2))
    print(sys._getframe(3))
    print(sys._getframe(4))
    print(sys._getframe(5))
    # print(sys._getframe(6))
    print('='*79)
    print(sys._getframe(0).f_code.co_name)
    print(sys._getframe(1).f_code.co_name)
    print(sys._getframe(2).f_code.co_name)
    print(sys._getframe(3).f_code.co_name)
    print(sys._getframe(4).f_code.co_name)
    print(sys._getframe(5).f_code.co_name)
    print(sys._getframe(6).f_code.co_name)
    print('='*79)
    print(sys._getframe(0).f_locals)
    print('='*79)
    print(sys._getframe(1).f_locals)
    print('='*79)
    print(sys._getframe(2).f_locals)
    print('='*79)
    print(sys._getframe(3).f_locals)
    print('='*79)
    print(sys._getframe(4).f_locals)
    print('='*79)
    print(sys._getframe(5).f_locals)
    # print('='*79)
    # print(sys._getframe(6).f_locals)


class e():
    k =3
    def h(self):
        i = 1
        self.f()
    def f(self):
        j=2
        a()

class p(e):
    y = 1
    x = 2
    print(sys._getframe(0).f_code.co_name)
    # print(sys._getframe(1).f_code.co_name)
    print('='*79)
```

```
g = p()
g.h()
```

代码解读：

1. sys._getframe().f_locals['z'] =1并不会真正修改局部命名空间中的z，与locals()类似，所以z依旧等于9
2. sys.getframe(i).f_code.co_name表示栈中第i层的函数名称，栈底层函数往下是返回<module>
3. sys.getframe(i).f_locals表示栈中第i层的函数的局部空间

运行结果：

```
p
================================================================================
{'__module__': '__main__', '__qualname__': 'p', 'y': 1, 'x': 2}
================================================================================
9
================================================================================
<frame at 0x000001C1E154B208, file 'C:\\Users\\Kyle\\Desktop\\backtrader-master
<frame at 0x000001C1E154D048, file 'C:\\Users\\Kyle\\Desktop\\backtrader-master
<frame at 0x000001C1E13BBD48, file 'C:\\Users\\Kyle\\Desktop\\backtrader-master
<frame at 0x000001C1E13BBBA8, file 'C:\\Users\\Kyle\\Desktop\\backtrader-master
<frame at 0x000001C1E1545558, file 'C:\\Users\\Kyle\\Desktop\\backtrader-master
<frame at 0x000001C1E15451F8, file 'C:\\Users\\Kyle\\Desktop\\backtrader-master
================================================================================
c
b
a
f
h
<module>
exec_code
================================================================================
{'l': 4, 'z': 9}
================================================================================
{'m': 5}
================================================================================
{'n': 6}
================================================================================
{'self': <__main__.p object at 0x000001C1E1025988>, 'j': 2}
================================================================================
{'self': <__main__.p object at 0x000001C1E1025988>, 'i': 1}
================================================================================
{'__name__': '__main__', '__doc__': '\nCreated on Tue Jul 14 10:41:11 2020\n\n@
此处省略很多内容...
'help': Type help() for interactive help, or help(object) for help about object
```

案例代码2：

```
import itertools

import sys

def a():
    n=6
    b()
```

```
def c():
    for i in itertools.count(1):
        print(i)
        print(sys._getframe(i).f_code.co_name)

a()
```

代码解读：

1. 栈的深度是由限制的，基本是<module>往下30层

运行结果：

```
1
b
2
a
3
<module>
4
exec_code
5
runfile
6
<module>
7
run_code
8
run_ast_nodes
9
run_cell_async
10
_pseudo_sync_runner
11
_run_cell
12
run_cell
13
run_cell
14
do_execute
15
wrapper
16
execute_request
17
wrapper
18
dispatch_shell
19
wrapper
20
process_one
21
run
22
inner
23
```

```
_run
26
_run_once
27
run_forever
28
start
29
start
30
main
31
<module>
32
_run_code
33
_run_module_as_main
34
Traceback (most recent call last):

  File "C:\Users\Kyle\Desktop\backtrader-master\backtrader-master\Kyle1_test_fu
    a()

  File "C:\Users\Kyle\Desktop\backtrader-master\backtrader-master\Kyle1_test_fu
    b()

  File "C:\Users\Kyle\Desktop\backtrader-master\backtrader-master\Kyle1_test_fu
    c()

  File "C:\Users\Kyle\Desktop\backtrader-master\backtrader-master\Kyle1_test_fu
    print(sys._getframe(i).f_code.co_name)

ValueError: call stack is not deep enough
```

发布于 2020-07-15 10:45

量化交易 Python

文章被以下专栏收录

backtrader源代码解析
开源交易系统backtrader源码笔记

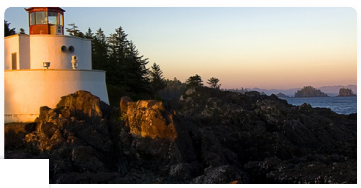
推荐阅读

干货 | 事件：用python从零实现基于事件驱动的量化回测...

上篇文章实现了量化回测系统的基础框架，本文继续。这篇文章，主

BackTrader官方文档 10 - 数据槽(1) 概述

BackTrader提供了一组数据槽解析器(在编写所有基于CSV的数据时),



Pytho 得分代

import
numpy
matplo
seabo

知乎 首发于 backtrader源代码解析

还没有评论

写下你的评论...

▲ 赞同 3 ▼ 添加评论 分享 喜欢 收藏 申请转载 ···