金山网络
www.ijinshan.com

Python 隐藏的玄机

目录

对象与引用

- 很基本，最容易遇到的问题

```
In [1]: a = [1, 2, 3]
In [2]: b = a
In [3]: a.append(4)
In [4]: a, b
Out[4]: ([1, 2, 3, 4], [1, 2, 3, 4])
```

- 同样的，字典也是可变对象

```
In [5]: c = {1:2}
In [6]: d = c
In [7]: c[2] = 3
In [8]: c, d
Out[8]: ({1: 2, 2: 3}, {1: 2, 2: 3})
```

对象与引用

- 列表的使用问题
  - 取奇数列

```
In [11]: a = [1, 2, 3, 4, 5, 6, 7, 8]
In [12]: for i in xrange(len(a)):
    ....:     if i & 0x1:
    ....:         a.pop(i)
IndexError: pop index out of range
In [13]: a
Out[13]: [1, 3, 4, 6, 7]

In [14]: a = [1, 2, 3, 4, 5, 6, 7, 8]
In [15]: a = a[::2]
In [16]: a
Out[16]: [1, 3, 5, 7]
```

对象与引用

- 列表对象操作
  - extend
  - +=
  - = +

```
In [1]: a = [1, 2]
In [2]: id(a)
Out[2]: 47269000
In [3]: a.extend([3])
In [4]: id(a)
Out[4]: 47269000
In [5]: a += [4]
In [6]: id(a)
Out[6]: 47269000
In [7]: a = a + [5]
In [8]: id(a)
Out[8]: 47238032
In [9]: a
Out[9]: [1, 2, 3, 4, 5]
```

对象与引用

- 元组内的列表

```
In [18]: a = ([], [])
In [19]: a[0].append(1)
In [20]: a[0].extend([2])
In [21]: a[0] += [3]
---------------------------------
TypeError
Traceback (most recent call last)
<ipython-input-21-4cd2980655d0> in
<module>()
----> 1 a[0] += [3]
TypeError: 'tuple' object does not
support item assignment
In [22]: a
Out[22]: ([1, 2, 3], [])
```

对象与引用

- 复制对象的deepcopy和[:]

```
In [1]: from copy import deepcopy
In [2]: a = [1, 2, 3]
In [3]: b = deepcopy(a)
In [4]: b.append(4)
In [5]: id(a), id(b), a, b
Out[5]:
  (24587080, 24587584, [1, 2, 3], [1, 2, 3, 4])
In [6]: c = a[:]
In [7]: id(a), id(b), id(c)
Out[7]: (24587080, 24587584, 24586864)
```

目录

函数参数初始值

- python函数的初始值
  - 如果是列表……

```
In [10]: def a(b=[]):
   ....:     b.append('hi')
   ....:     print b
   ....:
In [11]: a()
['hi']
In [12]: a()
['hi', 'hi']
In [13]: a(['2'])
['2', 'hi']
In [14]: a()
['hi', 'hi', 'hi']
```

函数参数初始值

- 与函数参数问题很相似的
  - 类属性

```
In [1]: class A:
   ...:     b = []
   ...:     def __init__(self, c):
   ...:         self.b.append(c)
   ...:
In [2]: f = A(1)
In [3]: g = A(2)
In [4]: f.b, g.b
Out[4]: ([1, 2], [1, 2])
```

函数参数初始值

- 常见解决方法

```
def a(b=None):
    b = b or []
    ......
```

- 利用方法

```
In [1]: def counter(b=[0]):
   ...:     b[0] += 1
   ...:     return b[0]
   ...:
In [2]: print a()
1
In [3]: print a()
2
```

目录

闭包

- 我们经常这样用

```
def wrap(log):
    thno = [0]
    def initfunc():
        thno[0] += 1
        log.info('Thread-%s' % thno[0])
        return thno[0]
    return initfunc
```

- 但是在多线程的时候，它会产生点意外…
    - 在多个线程可能获得相同的thno[0]值

闭包

- 另一种情况……循环生成闭包

```
In [1]: a = []
In [2]: for i in xrange(10):
   ...:     def b():
   ...:         return i
   ...:     a.append(b)
   ...:
In [3]: a[0]()
Out[3]: 9
In [4]: a[1]()
Out[4]: 9
In [5]: a[9]()
Out[5]: 9
```

```
In [1]: a = []
In [2]: def bwrap(i):
   ...:     def b():
   ...:         return i
   ...:     return b
   ...:
In [4]: for i in xrange(10):
   ...:     a.append(bwrap(i))
   ...:
In [5]: a[0]()
Out[5]: 0
In [6]: a[8]()
Out[6]: 8
```

闭包

- 原因是…
  - 函数层面保存变量
  - 在多线程时，
    外层函数保存的变量
    成为各线程的“全局变量”

目录

GIL下CPU使用超过100%

- 其实是逃出了GIL的魔掌
  - 走进了C的怀抱

```
#include <stdio.h>
void loop()
{
    while(1) ;
}

gcc a.c -fPIC -shared -o a.so
```

```
from ctypes import cdll
from threading import Thread
import time

lib = cdll.LoadLibrary("./a.so")
Thread(target=lib.loop).start()
Thread(target=lib.loop).start()
```

- 最好还是使用multiprocessing模块

目录

多进程下的异常输出

- 无论是多进程还是多线程都应该使用logging模块

```
logger = logging.getLogger(filename)
filename = 'xxx'
logformat = logging.Formatter(
         '%(asctime)s [%(levelname)s] %(message)s',
         '%Y-%m-%d %H:%M:%S')
logger.setLevel(getattr(logging, level))
fh = logging.FileHandler(filename)
fh.setFormatter(logformat)
logger.addHandler(fh)
```

多进程下的异常输出

- 想要traceback？
  - 可以这样……

```
def traceback_wrap(self):
    def tt():
        import traceback as tb
        class MimicryFile:
            def __init__(self, log):
                self.log= log
            def write(self, strr):
                self.log.error(strr.strip())
        tb.print_exc(file=MimicryFile(self))
    return tt
logger.traceback = traceback_wrap(logger)
logger.traceback()  # usage
```

Some tricks...

- 函数参数拆包

```
def a(x, y):
    print x, y

i = [1, 2]
j = {'y': 3, 'x': 2}

a(*i)
a(**j)
```

- 不定长参数

```
def a(*x,**y):
    print x, y

>>> a(1,2,3,x=2,y=3)
(1, 2, 3) {'y': 3, 'x': 2}
```

Some tricks…

- 链式比较

```
In [1]: x = 2
In [2]: y = 2.5
In [3]: 1 < x < y < 3
Out[3]: True
```

- for else

```
In [4]: for i in xrange(10):
   ...:     if i == 10:
   ...:         break
   ...: else:
   ...:     print 'i change from 0 ~ 9'
i change from 0 ~ 9
```

金山网络
www.ijinshan.com