11. 前方高能-迭代器

本节主要内容:

1. 函数名的使用以及第一类对象
2. 闭包
3. 迭代器

一. 函数名的运用.
函数名是一个变量, 但它是一个特殊的变量, 与括号配合可以执行函数的变量.

1. 函数名的内存地址

```
def func():
    print("呵呵")

print(func)

结果:
<function func at 0x1101e4ea0>
```

2. 函数名可以赋值给其他变量

```
def func():
    print("呵呵")

print(func)

a = func    # 把函数当成一个变量赋值给另一个变量
a()     # 函数调用 func()
```

3. 函数名可以当做容器类的元素

```
def func1():
    print("呵呵")

def func2():
    print("呵呵")

def func3():
    print("呵呵")

def func4():
    print("呵呵")
```

```
lst = [func1, func2, func3]
for i in lst:
    i()
```

4. 函数名可以当做函数的参数

```
def func():
    print("吃了么")


def func2(fn):
    print("我是func2")
    fn()    # 执行传递过来的fn
    print("我是func2")

func2(func)     # 把函数func当成参数传递给func2的参数fn.
```

5. 函数名可以作为函数的返回值

```
def func_1():
    print("这里是函数1")
    def func_2():
        print("这里是函数2")
    print("这里是函数1")
    return func_2

fn = func_1()   # 执行函数1.  函数1返回的是函数2, 这时fn指向的就是上面函数2
fn()    # 执行上面返回的函数
```

二. 闭包

什么是闭包? 闭包就是内层函数, 对外层函数(非全局)的变量的引用. 叫闭包

```
def func1():
    name = "alex"
    def func2():
        print(name)     # 闭包
    func2()
func1()
结果:
alex
```

我们可以使用__closure__来检测函数是否是闭包. 使用函数名.__closure__返回cell就是

闭包. 返回None就不是闭包

```
def func1():
    name = "alex"
    def func2():
        print(name)     # 闭包
    func2()
    print(func2.__closure__)    # (<cell at 0x10c2e20a8: str object at
0x10c3fc650>,)
func1()
```

问题, 如何在函数外边调用内部函数呢?

```
def outer():
    name = "alex"
    # 内部函数
    def inner():
        print(name)
    return inner

fn = outer()   # 访问外部函数, 获取到内部函数的函数地址
fn()    # 访问内部函数
```

那如果多层嵌套呢? 很简单, 只需要一层一层的往外层返回就行了

```
def func1():
    def func2():
        def func3():
            print("嘿嘿")
        return func3
    return func2

func1()()()
```

由它我们可以引出闭包的好处. 由于我们在外界可以访问内部函数. 那这个时候内部函数访问的时间和时机就不一定了, 因为在外部, 我可以选择在任意的时间去访问内部函数. 这个时候. 想一想. 我们之前说过, 如果一个函数执行完毕. 则这个函数中的变量以及局部命名空间中的内容都将会被销毁. 在闭包中. 如果变量被销毁了. 那内部函数将不能正常执行. 所以. python规定. 如果你在内部函数中访问了外层函数中的变量. 那么这个变量将不会消亡. 将会常驻在内存中. 也就是说. 使用闭包, 可以保证外层函数中的变量在内存中常驻. 这样做有什么好处呢? 非常大的好处. 我们来看一个关于爬虫的代码:

```
from urllib.request import urlopen

def but():
    content = urlopen("http://www.xiaohua100.cn/index.html").read()
    def get_content():
        return content
    return get_content
```

```
fn = but()  # 这个时候就开始加载校花100的内容
# 后面需要用到这里面的内容就不需要在执行非常耗时的网络连接操作了
content = fn()  # 获取内容
print(content)

content2 = fn()  # 重新获取内容
print(content2)
```

综上, 闭包的作用就是让一个变量能够常驻内存. 供后面的程序使用.

三. 迭代器

我们之前一直在用可迭代对象进行迭代操作. 那么到底什么是可迭代对象. 本小节主要讨论可迭代对象. 首先我们先回顾一下目前我们所熟知的可迭代对象有哪些:

str, list, tuple, dict, set. 那为什么我们可以称他们为可迭代对象呢? 因为他们都遵循了可迭代协议. 什么是可迭代协议. 首先我们先看一段错误代码:

```
# 对的
s = "abc"
for c in s:
    print(c)

# 错的
for i in 123:
    print(i)

结果:
Traceback (most recent call last):
  File "/Users/sylar/PycharmProjects/oldboy/iterator.py", line 8, in
<module>
    for i in 123:
TypeError: 'int' object is not iterable
```

注意看报错信息中有这样一句话. 'int' object is not iterable . 翻译过来就是整数类型对象是不可迭代的. iterable表示可迭代的. 表示可迭代协议. 那么如何进行验证你的数据类型是否符合可迭代协议. 我们可以通过dir函数来查看类中定义好的所有方法.

```
s = "我的哈哈哈"
print(dir(s))       # 可以打印对象中的方法和函数
print(dir(str))     # 也可以打印类中声明的方法和函数
```

在打印结果中. 寻找__iter__ 如果能找到. 那么这个类的对象就是一个可迭代对象.

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__',
```

```
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower',
'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join',
'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

我们发现在字符串中可以找到__iter__. 继续看一下list, tuple, dict, set

```
print(dir(tuple))
print(dir(list))
print(dir(open("护士少妇嫩模.txt")))  # 文件对象
print(dir(set))
print(dir(dict))

结果:
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count',
'index']
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
['_CHUNK_SIZE', '__class__', '__del__', '__delattr__', '__dict__',
'__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__',
'__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__ne__',
'__new__', '__next__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_checkClosed',
'_checkReadable', '_checkSeekable', '_checkWritable', '_finalizing',
'buffer', 'close', 'closed', 'detach', 'encoding', 'errors', 'fileno',
'flush', 'isatty', 'line_buffering', 'mode', 'name', 'newlines', 'read',
'readable', 'readline', 'readlines', 'seek', 'seekable', 'tell',
'truncate', 'writable', 'write', 'writelines']
['__and__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__iand__', '__init__', '__init_subclass__', '__ior__',
'__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__',
'__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__',
'__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__',
```

```
'__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear',
'copy', 'difference', 'difference_update', 'discard', 'intersection',
'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop',
'remove', 'symmetric_difference', 'symmetric_difference_update', 'union',
'update']
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',
'__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys',
'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

我们发现这几个可以进行for循环的东西都有__iter__函数, 包括range也有. 可以自己试一下.

这是查看一个对象是否是可迭代对象的第一种办法. 我们还可以通过isinstence()函数来查看一个对象是什么类型的

```python
l = [1,2,3]
l_iter = l.__iter__()
from collections import Iterable
from collections import Iterator
print(isinstance(l,Iterable))    #True
print(isinstance(l,Iterator))    #False
print(isinstance(l_iter,Iterator))    #True
print(isinstance(l_iter,Iterable))    #True
```

综上. 我们可以确定. 如果对象中有__iter__函数. 那么我们认为这个对象遵守了可迭代协议. 就可以获取到相应的迭代器. 这里的__iter__是帮助我们获取到对象的迭代器. 我们使用迭代器中的__next__()来获取到一个迭代器中的元素. 那么我们之前讲的for的工作原理到底是什么? 继续看代码

```python
s = "我爱北京天安门"
c = s.__iter__()    # 获取迭代器
print(c.__next__())     # 使用迭代器进行迭代. 获取一个元素    我
print(c.__next__())     # 爱
print(c.__next__())     # 北
print(c.__next__())     # 京
print(c.__next__())     # 天
print(c.__next__())     # 安
print(c.__next__())     # 门
print(c.__next__())     # StopIteration
```

for循环的机制:

```python
for i in [1,2,3]:
    print(i)
```

使用while循环+迭代器来模拟for循环(必须要掌握)

```
lst = [1,2,3]
lst_iter = lst.__iter__()
while True:
    try:
        i = lst_iter.__next__()
        print(i)
    except StopIteration:
        break
```

总结:

Iterable: 可迭代对象. 内部包含__iter__()函数

Iterator: 迭代器. 内部包含__iter__() 同时包含__next__().

迭代器的特点:

1. 节省内存.
2. 惰性机制
3. 不能反复, 只能向下执行.

我们可以把要迭代的内容当成子弹. 然后呢. 获取到迭代器__iter__(), 就把子弹都装在弹夹中. 然后发射就是__next__()把每一个子弹(元素)打出来. 也就是说, for循环的时候. 一开始的时候是__iter__()来获取迭代器. 后面每次获取元素都是通过__next__()来完成的. 当程序遇到StopIteration将结束循环.