Python for Senior Lesson 8

v1.0.0

2016.11 by David.Yi

本次内容要点

- python 内建模块： collections
- 思考一下

collections

collections 是 Python 内建的一个集合模块，提供了许多有用的集合类。我们来完整的看看 collections 提供了哪些扩展功能。

python 内置了大量的功能函数，但是再多的函数和模块也不可能覆盖实际应用需要的所有功能，怎么扩展程序功能，怎么划分功能模块，python 自己所带的这些模块是最好的参考书！

- namedtuple: 生成可以使用名字来访问元素内容的tuple子类
- deque: 双端队列，可以快速的从另外一侧追加和推出对象
- defaultdict: 带有默认值的字典
- OrderedDict: 有序字典
- Counter: 计数器

---

namedtuple 有名字的元组

namedtuple 主要用来产生可以使用名称来访问元素的数据对象，通常用来增强代码的可读性，在访问一些 tuple 类型的数据时尤其好用。

namedtuple 是一个函数，它用来创建一个自定义的 tuple 对象，并且规定了 tuple 元素的个数，可以用属性而不是索引来写入或者访问 tuple 的某个元素。

这样一来，我们用 namedtuple 可以很方便地定义一种数据类型，比如 XY 坐标，它具备 tuple 的内容不变性，又可以根据属性来引用，使用十分方便。

In [14]:

```
# nametuple 举例

from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
print(p.x, p.y)
```

```
1 2
```

In [2]:

```
i = p.x + p.y
print(i)
```

```
3
```

In [4]:

```
# nametuple 举例

from collections import namedtuple

Web = namedtuple('web', ['name', 'type', 'url'])
p1 = Web('google', 'search', 'www.google.com')
p2 = Web('sina', 'portal', 'www.sina.com.cn')

print(p1)
```

```
web(name='google', type='search', url='www.google.com')
```

In [5]:

```
print(p1.name, p1.url)
```

```
google www.google.com
```

In [6]:

```
print(p1.url, p2.url)
```

```
www.google.com www.sina.com.cn
```

In [5]:

```
# 遍历 nametuple

for i in p2:
    print(i)
```

```
sina
portal
www.sina.com.cn
```

In [15]:

```
# 复杂的基于 namedtuple list demo
from collections import namedtuple

Web = namedtuple('web', ['name', 'type', 'url'])

p = []
p.append(Web('google', 'search', 'www.google.com'))
p.append(Web('sina', 'portal', 'www.sina.com.cn'))
print(p)

for i in p:
    print(i.name)
```

```
[web(name='google', type='search', url='www.google.com'), web(name
='sina', type='portal', url='www.sina.com.cn')]
google
sina
```

In [4]:

```
# 显示 namestuple 的字段名称

print(Web._fields)
```

```
('name', 'type', 'url')
```

---

deque

使用 list 存储数据时，按索引访问元素很快，但是插入和删除元素就很慢了，因为 list 是线性存储，数据量大的时候，插入和删除效率很低。

deque 是为了高效实现插入和删除操作的双向列表，适合用于队列和栈。

deque 在插入数据时候速度比 list 快很多，当然这个是相对存在大量数据的 list 而言的。如果你的程序中需要对有百万级别的 list 频繁的在各个位置插入删除数据，那么用 deque 是值得的。

In [14]:

```
# deque 举例

from collections import deque

q = deque(['a', 'b', 'c'])
q.append('x')
q.appendleft('y')

print(q)
```

```
deque(['y', 'a', 'b', 'c', 'x'])
```

In [17]:

```
# 对比 list 和 deque 的速度

from collections import deque
import time

# list
q0 = [x*x for x in range(10000000)]

# list
a = time.time()
q0.insert(0,888)
# q0.append(999)
b = time.time()
print(b-a)

# 生成 deque
q1= deque(q0)

# deque
a = time.time()
q1.appendleft(888)
# q1.append(999)
b = time.time()
print(b-a)
```

```
0.03333306312561035
5.1975250244140625e-05
```

In [6]:

```
from collections import deque

l = ['a','b','c', 'd']
l = deque(l)
l
```

Out[6]:

```
deque(['a', 'b', 'c', 'd'])
```

In [29]:

```
# deque left rotation

l = ['a','b','c']
l = deque(l)
l.rotate(-1)
print(l)
l.rotate(1)
print(l)
```

```
deque(['b', 'c', 'a'])
deque(['a', 'b', 'c'])
```

In [30]:

```
# deque pop() 同样可以区分头尾

l = deque(['a','b','c'])
l.pop()
print(l)
```

```
deque(['a', 'b'])
```

In [34]:

```
l = deque(['a','b','c'])
l.popleft()
print(l)
```

```
deque(['b', 'c'])
```

---

defaultdict

我们都知道，在使用Python原生的数据结构dict的时候，如果用 d[key] 这样的方式访问， 当指定的key不存在时，是会抛出KeyError异常的，也就是发生错误。（当然可以用 get 方法来避免这样的错误）

如果使用defaultdict，只要你传入一个默认的方法，那么请求一个不存在的key时， 便会调用这个方法，使用其结果来作为这个key的默认值。

In [39]:

```
# 标准的字典用法

i = {'name':'David'}
print(i['name'])
```

```
David
```

In [40]:

```
# 不存在 key，则会报错

i = {'name':'David'}
print(i['score'])
```

```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent cal
l last)
<ipython-input-40-b940f22f4294> in <module>()
      1 i = {'name':'David'}
----> 2 print(i['score'])

KeyError: 'score'
```

In [27]:

```
# defaultdict 举例

from collections import defaultdict

i = defaultdict(lambda: 100)
i['name']='David'
print(i['name'])
```

```
David
```

In [12]:

```
# default 返回默认值，不会报错

print(i['score'])
print(i['best_score'])
```

```
100
100
```

In [46]:

```
from collections import defaultdict

i = defaultdict(lambda: '100')
i['name']='David'
print(i['name'])
print(i['score'])
```

```
David
100
```

OrderedDict

使用 dict 字典时，Key 是无序的。在对 dict 做迭代访问时，我们无法确定 Key 的顺序。

如果要保持 Key 的顺序，可以用 OrderedDict，这是一个 Key 值有序的字典数据类型。

In [48]:

```
# dict 是无序的

d = dict([('a', 1), ('b', 2), ('c', 3)])

print(d)
```

```
{'c': 3, 'b': 2, 'a': 1}
```

In [51]:

```
# 传统dict 追加一对 key value

d = dict([('a', 1), ('b', 2), ('c', 3)])

print(d)

d['d'] = 4
print(d)
```

```
{'c': 3, 'b': 2, 'a': 1}
{'d': 4, 'c': 3, 'b': 2, 'a': 1}
```

In [50]:

```
# 使用 OrderedDict

from collections import OrderedDict

d = OrderedDict()
d['a'] = 1
d['b'] = 2
d['c'] = 3

print(d)
```

```
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

In [52]:

```
# 使用 OrderedDict, 追加一对 key value

from collections import OrderedDict

d = OrderedDict()
d['a'] = 1
d['b'] = 2
d['c'] = 3

print(d)

d['d'] = 4
print(d)
```

```
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
```

In [55]:

```
# OrderedDict可以实现一个FIFO（先进先出）的dict，当容量超出限制时，先删除最早添加的Key：

from collections import OrderedDict

class LastUpdatedOrderedDict(OrderedDict):

    def __init__(self, capacity):
        super(LastUpdatedOrderedDict, self).__init__()
        self._capacity = capacity

    def __setitem__(self, key, value):
        containsKey = 1 if key in self else 0
        if len(self) - containsKey >= self._capacity:
            last = self.popitem(last=False)
            print('remove:', last)
        if containsKey:
            del self[key]
            print('set:', (key, value))
        else:
            print('add:', (key, value))
        OrderedDict.__setitem__(self, key, value)

d = LastUpdatedOrderedDict(4)
d['a'] = 1
d['b'] = 2
d['c'] = 3
print(d)
```

```
add: ('a', 1)
add: ('b', 2)
add: ('c', 3)
LastUpdatedOrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

In [56]:

```
d['d'] = 4
d['e'] = 5
d['f'] = 6
print(d)
```

```
add: ('d', 4)
remove: ('a', 1)
add: ('e', 5)
remove: ('b', 2)
add: ('f', 6)
LastUpdatedOrderedDict([('c', 3), ('d', 4), ('e', 5), ('f', 6)])
```

Counter

Counter是一个简单的计数器，例如，统计字符出现的个数：

In [59]:

```
# 下面这个例子就是使用 Counter 模块统计一段句子里面所有字符出现次数

from collections import Counter

s = 'A Counter is a dict subclass. '.lower()

c = Counter(s)
# 获取出现频率最高的5个字符
print(c.most_common(5))
```

```
[(' ', 6), ('s', 4), ('c', 3), ('a', 3), ('t', 2)]
```

In [60]:

```
# Counter 举例

from collections import Counter

s = input('Please input:')

s = s.lower()

c = Counter(s)
# 获取出现频率最高的5个字符
print(c.most_common(5))
```

```
Please input:This is a big book
[(' ', 4), ('i', 3), ('b', 2), ('s', 2), ('o', 2)]
```

思考一下

In [61]:

```
x = 2

if 1 < x < 4:
    print('ok')
```

```
ok
```

In [62]:

```
x, y = 2, 6

if 1 < x < 4 < y:
    print('ok')
```

```
ok
```

In [64]:

```
a, *middle, c = [1, 2, 3, 4]
print(a,middle,c)
```

```
1 [2, 3] 4
```

In [65]:

```
a, middle, *c = [1, 2, 3, 4]
print(a,middle,c)
```

```
1 2 [3, 4]
```

In [67]:

```
s= 'a'
a = s * 4
print(a)
```

```
aaaa
```

In [69]:

```
d = dict([('a', 1), ('b', 2), ('c', 3)])
print(d.get('a'))
```

```
1
```

In [70]:

```
print(d.get('d'))
```

```
None
```

In [73]:

```
print(d.get('d',100))
```

```
100
```

In [ ]:

```

```