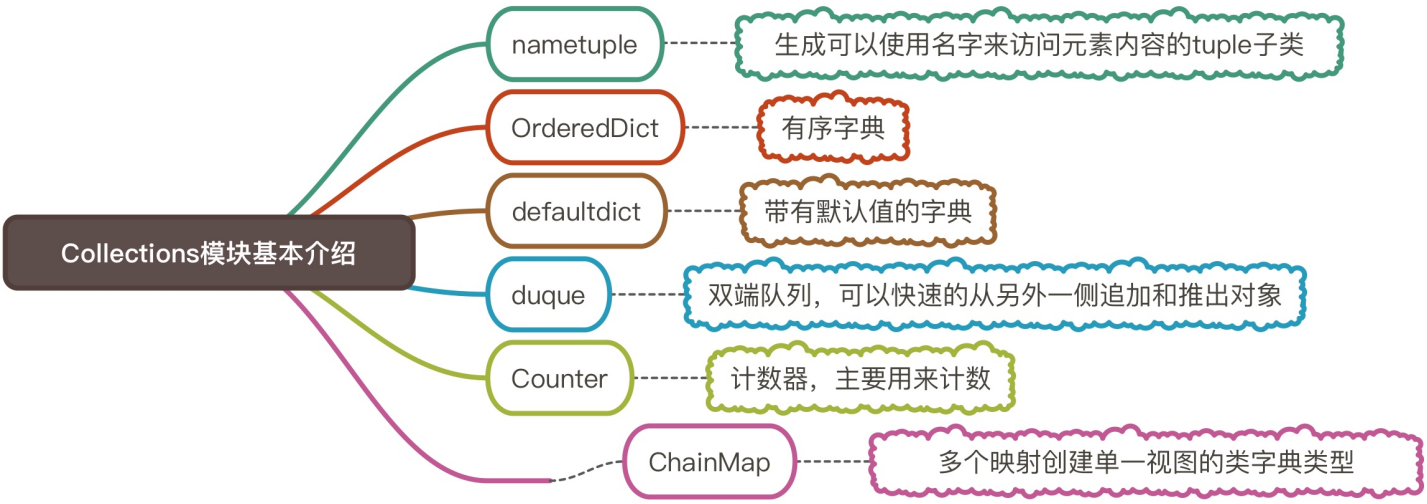
Python collections模块总结



除了我们使用的那些基础的数据结构，还有包括其它的一些模块提供的数据结构，有时甚至比基础的数据结构还要好用。

collections

ChainMap

这是一个为多个映射创建单一视图的类字典类型，也就是说，它同样具有字典类型的方法，它比基础数据结构中的字典的创建和多次更新要快，**需要注意的是，增删改的操作都只会针对该对象的第一个字典，其余字典不会发生改变，但是如果是查找，则会在多个字典中查找，直到找到第一个出现的key为止。**

特有方法	解释
maps	返回全部的字典（这个列表中至少存在一个字典）
new_child	在字典列表头部插入字典，如果其参数为空，则会默认插入一个空字典，并且返回一个改变后的ChainMap对象
parents	返回除了第一个字典的其余字典列表的ChainMap对象，可以用来查询除了第一个列表以外的内容。

```
import collections
a = {1: 2, 2: 3}
b = {1: 3, 3: 4, 4: 5}
chains = collections.ChainMap(a, b)
```

```
# maps
# 注意maps是个属性，不是一个方法，其改变
print(chains.maps)  # [{1: 2, 2: 3}, {1: 3, 3: 4, 4: 5}]
# get
assert chains.get(1, -1) == 2
# parents
# 从第二个map开始找
assert chains.parents.get(1, -1) == 3
# popitem
assert chains.popitem() == (2, 3)
# pop
# 返回的是value
assert chains.pop(1) == 2
# new_child
assert chains.new_child()
print(chains.maps)  # [{}, {1: 3, 3: 4, 4: 5}]
chains[2] = 1
print(chains.maps)  # [{2: 1}, {1: 3, 3: 4, 4: 5}]
# setdedault
# 如果已经存在key，则不会添加
assert chains.setdefault(1, 10) == 3
# update
chains.update({2: 4, 3: 5})
print(chains.maps)  # [{1: 2, 2: 4, 3: 5}, {1: 3, 3: 4, 4: 5}
]
# keys
print(chains.keys())  # KeysView(ChainMap({2: 4, 3: 5}, {1: 3
, 3: 4, 4: 5}))
# KeysView 继承了mapping和set
print(2 in chains.keys())  # True
print(len(chains.keys()))  # 4（重复的不算）
# clear
chains.clear()
print(chains.maps)  # [{}, {1: 3, 3: 4, 4: 5}]
```

就像它的特点一样，它适用于以下的情况：

1. 多个字典；
2. 允许key是重复；
3. 总是访问最高优先级字典中的关键字；
4. 不需要改变key对应的value；

5. 字典频繁的创建和更新已经造成巨大的性能问题，希望改善性能问题；

Counter

这是一个继承dict的子类，专门用来做计数器，dict中的方法这里同样适用。

特有方法	解释
init	初始化，参数为可迭代对象即可
elememts	返回一个生成器，其键值以无序的方式返回，并且只有值大于1的键值对才会返回
most_common	返回值最大的键值对，参数指定返回前多少个
subtract	减法，调用者的值发生改变
update	加法，调用者的值发生改变
[]	返回键对应的值，如果键不存在，那么返回0
+	加法，返回一个新的counter对象，如果前面不存在，则默认加上一个对应键，值为0的counter
-	减法，返回一个新的counter对象，如果前面不存在，则默认用对应键，值为0的counter来减，其中值正数会变负数，负数变为正数
&	min操作，取相对应的键的最小值，返回一个新的counter对象
|	max操作，取相对应的键的最大值，返回一个新的counter对象

其中数学运算如果其中一方的不存在，则会默认创建对应键，值为0的键值对。

```
from collections import Counter
# init
# 可迭代
counter = Counter("accab")  # Counter({'a': 2, 'c': 2, 'b': 1
})
counter2 = Counter([1,2,3,4])  # Counter({1: 1, 2: 1, 3: 1, 4
: 1})
counter5 = Counter([('a',3),('b', 2)])  # Counter({('a', 3):
1, ('b', 2): 1})
# 字典
counter3 = Counter({'a': 1, 'b': 2, 'a': 3})  # Counter({'a':
 3, 'b': 2})
counter4 = Counter(a=1, b=2, c=1)  # Counter({'b': 2, 'a': 1,
```

```
 'c': 1})
# elements
# 键值以无序的方式返回，并且只返回值大于等于1的键值对
elememts = counter.elements()
print([x for x in elememts])  # ['a', 'a', 'c', 'c', 'b']
# 为空是因为elements是generator
print(sorted(elememts))  # []
# most_common
# 键值以无序的方式返回
print(counter.most_common(1))  # [('a', 2)]
print(counter.most_common())  # [('a', 2), ('c', 2), ('b', 1)
]
# update
# 单纯是增加的功能，而不是像dict.update()中的替换一样
counter.update("abb")
print(counter)  # Counter({'a': 3, 'b': 3, 'c': 2})
# subtract
counter.subtract(Counter("accc"))
print(counter)  # Counter({'b': 3, 'a': 2, 'c': -1})
print([x for x in counter.elements()])  # ['a', 'a', 'b', 'b'
, 'b']
# get
# 键不存在则返回0，但是不会加入到counter键值对中
print(counter['d'])
print(counter)  # Counter({'b': 3, 'a': 2, 'c': -1})
del counter['d']
# 还可以使用数学运算
c = Counter(a=3, b=1)
d = Counter(a=1, b=2)
# add two counters together:  c[x] + d[x]
print(c + d)  # Counter({'a': 4, 'b': 3})
# subtract (keeping only positive counts)
print(c - d)  # Counter({'a': 2})
# # intersection:  min(c[x], d[x])
print(c & d) # Counter({'a': 1, 'b': 1})
# union:  max(c[x], d[x])
print(c | d)  # Counter({'a': 3, 'b': 2})
# 一元加法和减法
c = Counter(a=3, b=-1)
# 只取正数
```

```
print(+c)  # Counter({'a': 3})
print(-c)  # Counter({'b': 1})
```

deque

由于deque同样能够提供列表相关的函数，所以其和列表相同的函数则不再赘述，这里比较独特的是和`left`相关的函数以及`rotate`函数。

```
from collections import deque
# 从尾部进入，从头部弹出，保证长度为5
dq1 = deque('abcdefg', maxlen=5)
print(dq1)  # ['c', 'd', 'e', 'f', 'g']
print(dq1.maxlen)  # 5
# 从左端入列
dq1.appendleft('q')
print(dq1)  # ['q', 'c', 'd', 'e', 'f']
# 从左端入列
dq1.extendleft('abc')
print(dq1)  # ['c', 'b', 'a', 'q', 'c']
# 从左端出列并且返回
dq1.popleft()  # c
print(dq1)  # ['b', 'a', 'q', 'c']
# 将队头n个元素进行右旋
dq1.rotate(2)
print(dq1)  # ['q', 'c', 'b', 'a']
# 将队尾两个元素进行左旋
dq1.rotate(-2)
print(dq1)  # ['b', 'a', 'q', 'c']
def tail(filename, n=10):
    'Return the last n lines of a file'
    with open(filename) as f:
        return deque(f, n)
def delete_nth(d, n):
    """
 实现队列切片和删除，pop之后再放会原处
 :param d: deque
 :param n: int
 :return:
 """
```

```
    d.roatte(-n)
    d.popleft()
    d.rotate(n)
```

OrderedDict

OrderedDict提供了一个有序字典，可以使用在遍历的时候根据相应的顺序进行输出，因为在dict中它的item是以任意顺序进行输出的。

注意初始化的时候和在插入的话都根据插入顺序进行排序，而不是根据key进行排序。

```
from collections import OrderedDict
items = {'c': 3, 'b': 2, 'a': 1}
regular_dict = dict(items)
ordered_dict = OrderedDict(items)
print(regular_dict)  # {'c': 3, 'b': 2, 'a': 1}
print(ordered_dict)  # [('c', 3), ('b', 2), ('a', 1)]
# 按照插入顺序进行排序而不是
ordered_dict['f'] = 4
ordered_dict['e'] = 5
print(ordered_dict)  # [('c', 3), ('b', 2), ('a', 1), ('f', 4
), ('e', 5)]
# 把最近加入的删除
print(ordered_dict.popitem(last=True))  # ('e', 5)
# 按照加入的顺序删除
print(ordered_dict.popitem(last=False))  # ('c', 3)
print(ordered_dict)  # [('b', 2), ('a', 1), ('f', 4)]
# 移动到末尾
ordered_dict.move_to_end('b', last=True)
print(ordered_dict)  # [('a', 1), ('f', 4), ('b', 2)]
# 移动到开头
ordered_dict.move_to_end('b', last=False)
print(ordered_dict)  # [('b', 2), ('a', 1), ('f', 4)]
ordered_dict['a'] = 3
# 说明更改值并不会影响加入顺序
print(ordered_dict.popitem(last=True))  # ('f', 4)
```

```
# 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
```

```
class LastUpdateDOrderedDict(OrderedDict):
    def __init__(self, capcity):
        super().__init__() #继承OrderedDict
        self._capcity = capcity

    def __setitem__(self, key, value):
        containsKey = 1 if key in self else 0
        if len(self) - containsKey >= self._capcity: #判断是否超字典容量
            last = self.popitem(last=False)  #移除最早插入的元素
            print('remove',last)
        if containsKey:
            del self[key]
            print('set:',key, value)
        else:
            print('add:',key, value)
        OrderedDict.__setitem__(self, key, value)


o3 = LastUpdateDOrderedDict(3) #设置容量3
o3.update({'a':1,'b':2,'c':3,'d':4})
print('o3',o3)

add: a 1
add: b 2
add: c 3
remove ('a', 1)
add: d 4
o3 LastUpdateDOrderedDict([('b', 2), ('c', 3), ('d', 4)])
```

提供了字典的功能，又保证了顺序。

namedtuple

如果我们想要在tuple中使用名字的参数，而不是位置，namedtuple提供这么一个创建名称tuple的机会。

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(10, y=20)
```

```
print(p)  # Point(x=10, y=20)
print(p.x + p.y)  # 30
```