collections模块

- python中的collections模块中有一些特殊的容器类型，有事会很方便。
  1. deque：长度固定的双向队列
  2. Counter: 计数器
  3. defaultdict:不会因为键值不存在而报错
  4. OrderDict:有序的dict
  5. namedtuple:构建有名字的元组和有名字的类
  6. ChainMap容纳多个映射对象的容器

deque：长度固定的双向队列

- deque(maxlen=N)创建一个长度为N的固定队列的双向队列。
- 队列满时在此端添加n个值会同时删除彼端的n个值。

```
>>> from collections import deque
>>> q = deque(range(5), maxlen=5)
>>> q
deque([0, 1, 2, 3, 4], maxlen=5)
>>> q.append(5)
>>> q
deque([1, 2, 3, 4, 5], maxlen=5)
>>> q.rotate(3)
>>> q
deque([3, 4, 5, 1, 2], maxlen=5)
>>> q.appendleft(-1)
>>> q
deque([-1, 3, 4, 5, 1], maxlen=5)
>>> q.extendleft([11,22,33])
>>> q
deque([33, 22, 11, -1, 3], maxlen=5)
```

- maxlen一经设定不能修改。
- rotate(m)：正值时右侧m个元素移至左侧，负值相反。
- len(d)=d.maxlen时，开始达到上限，增加会导致另一侧的删除。
- extendleft()把其中的元素逐个相加到左侧。

Counter: 计数器

- 以键值对的形式储存，元素为key，计数为value。
- 创建

```
>>> c = Counter()
># 创建一个空的Counter类
>
>>> c = Counter('gallahad')
># 从一个可iterable对象（list、tuple、dict、字符串等）创建
>
>>> c = Counter({'a': 4, 'b': 2})
># 从一个字典对象创建
>
>>> c = Counter(a=4, b=2)
># 从一组键值对创建
```

- 访问
  - 存在则返回计数，不存在返回0

```
>>> c = Counter("abcdefgab")
>>> c["a"]
2
>>> c["c"]
1
>>> c["h"]
0
```

- 计数器的更新
  - update()进行增加更新

```
>>> c = Counter('which')
>>> c.update('witch')  # 使用另一个iterable对象更新
>>> c['h']
3
>>> d = Counter('watch')
>>> c.update(d)  # 使用另一个Counter对象更新
>>> c['h']
4
```

  - subtract()进行减少更新

```
>>> c = Counter('which')
>>> c.subtract('witch')  # 使用另一个iterable对象更新
>>> c['h']
1
>>> d = Counter('watch')
>>> c.subtract(d)  # 使用另一个Counter对象更新
>>> c['a']
-1
```

    - 计数为0不一定表示元素不存在。

- 删除元素

- 用del Counter[key]对元素进行删除

```
>>> c = Counter("abcdcba")
>>> c
Counter({'a': 2, 'c': 2, 'b': 2, 'd': 1})
>>> c["b"] = 0
>>> c
Counter({'a': 2, 'c': 2, 'd': 1, 'b': 0})
>>> del c["a"]
>>> c
Counter({'c': 2, 'b': 2, 'd': 1})
```

- element()
  - 按value返回key，value小于1的key不返回。

```
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> list(c.elements())
['a', 'a', 'a', 'a', 'b', 'b']
```

- most_common([n])
  - 返回top n个元素的列表，没有n则默认全部元素

```
>>> c = Counter('abracadabra')
>>> c.most_common()
[('a', 5), ('r', 2), ('b', 2), ('c', 1), ('d', 1)]
>>> c.most_common(3)
[('a', 5), ('r', 2), ('b', 2)]
```

- 算术和集合操作
  - +-&|也可用于Counter操作

```
>>> c = Counter(a=3, b=1)
>>> d = Counter(a=1, b=2)
>>> c + d  # c[x] + d[x]
Counter({'a': 4, 'b': 3})
>>> c - d  # subtract（只保留正数计数的元素）
Counter({'a': 2})
>>> c & d  # 交集:  min(c[x], d[x])
Counter({'a': 1, 'b': 1})
>>> c | d  # 并集:  max(c[x], d[x])
Counter({'a': 3, 'b': 2})
```

- 常用操作
  - 来源于官方文档

```
sum(c.values())
#所有计数的总数

c.clear()
#重置Counter对象，注意不是删除

list(c)
#将c中的键转为列表

set(c)
#将c中的键转为set

dict(c)
#将c中的键值对转为字典

c.items()
#转为(elem, cnt)格式的列表

Counter(dict(list_of_pairs))
#从(elem, cnt)格式的列表转换为Counter类对象

c.most_common()[:-n:-1]
#取出计数最少的n-1个元素

c += Counter()
#移除0和负值
```

defaultdict:不会因为键值不存在而报错

- dict中，若访问的key不存在，则KeyError。
- 用defaultdict则会返回一个默认值。

```
>>> from collections import defaultdict
>>> d = defaultdict(list)
>>> d['a']
[]
>>> d['a'].append(1)
>>> d
defaultdict(<class 'list'>, {'a': [1]})
>>> d['a'].append(2)
>>> d['b'].append(4)
>>> d
defaultdict(<class 'list'>, {'a': [1, 2], 'b': [4]})
>>> s = 'mississippi'
>>> d = defaultdict(int)
>>> for k in s:
   d[k] += 1
>>> sorted(d.items())
[('i', 4), ('m', 1), ('p', 2), ('s', 4)]
```

OrderDict:有序的dict

- OrderDict的key会严格按照添加顺序保持。

```
>>> from collections import OrderedDict
>>> d = OrderedDict()
>>> d
OrderedDict()
>>> d['foo'] = 1
>>> d['bar'] = 2
>>> d['spam'] = 3
>>> d['grok'] = 4
>>> for key in d:
...     print(key, d[key])
...
foo 1
bar 2
spam 3
grok 4
```

namedtuple:构建有名字的元组和有名字的类

- 创建一个具名元组需要两个参数，一个是类名，另一个是类的各个字段的名字。后者可以是由数个字符串组成的可迭代对象，或者是由空格分隔开的字段名组成的字符串。

```
>>> from collections import namedtuple
>>> City = namedtuple('City', 'name country population coordinates')
>>> tokyo = City('Tokyo', 'JP', 36.933, (35.689722, 139.691667))
>>> tokyo
City(name='Tokyo', country='JP', population=36.933, coordinates=(35.689722,
139.691667))
>>> tokyo.population
36.933
>>> tokyo.coordinates
(35.689722, 139.691667)
>>> tokyo[1]
'JP'
```

ChainMap容纳多个映射对象的容器

- 进行键查找操作的时候，这些对象会被当作一个整体被逐个查找，直到键被找到为止。
- ChainMap只是简单的维护一个记录底层映射关系的列表，然后重新定义常见的字典操作来扫描这个列表。大部分的操作都可以正常的工作。
- 如果与重复的键，那么这里会采用第一个映射中所对应的键值。
- 修改映射的操作总会作用在列出的第一个映射结构上。
- 注：ChainMap使用的是原始的字典，也就是说如果任一个原始的字典发生了变化，那么合并之后的字典也将会发生变化。但字典的update方法则不会变化。

```
>>> from collections import ChainMap
>>> a = {'x': 1, 'y': 2}
>>> b = {'y': 3 , 'z': 4}
>>> c = ChainMap(a, b)
>>> c
ChainMap({'x': 1, 'y': 2}, {'y': 3, 'z': 4})
>>> len(c)
3
>>> c.keys()
KeysView(ChainMap({'x': 1, 'y': 2}, {'y': 3, 'z': 4}))
>>> c.values()
ValuesView(ChainMap({'x': 1, 'y': 2}, {'y': 3, 'z': 4}))
>>> c['x']
1
>>> c['y']
2
>>> c['y'] = 22
>>> c['z'] = 9
>>> del c['x']
>>> a
{'y': 22, 'z': 9}
>>> c
ChainMap({'y': 22, 'z': 9}, {'y': 3, 'z': 4})
>>> a['a'] = 4
>>> a
{'y': 22, 'z': 9, 'a': 4}
>>> c
ChainMap({'y': 22, 'z': 9, 'a': 4}, {'y': 3, 'z': 4})
```