collections

若是某数据类型的子类，那么会继承其方法，并且会有新方法

--collections.Counter()

Counter是一个简单的计数器，其实初始化键值自动为0

是dict的一个子类

FE:

```
>>> import collections
>>> c = collections.Counter()
>>> for ch in 'this is a string':
 c[ch]+=1


>>> c
Counter({'i': 3, 's': 3, ' ': 3, 't': 2, 'a': 1, 'h': 1, 'n': 1, 'g': 1, 'r': 1})
```

--collections.deque()

高效实现插入和删除操作的双向列表，适合用于队列和栈

list是线性存储，数据量大的时候，插入和删除效率很低。

是list的子类

FE：

```
>>> q = collections.deque(['a','b','c','d'])
>>> q.append('e')
>>> q.appendleft('y')
>>> q.popleft()
'y'
```

--collections.namedtuple('name',[属性list])

FE：

```
>>> from collections import namedtuple
>>> point = namedtuple('point',[1,23])
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    point = namedtuple('point',[1,23])
  File "D:\编程\lib\collections\__init__.py", line 400, in namedtuple
    'identifiers: %r' % name)
ValueError: Type names and field names must be valid identifiers: '1'
>>> point = namedtuple('point',['x','y'])
>>> p = point(1,2)
>>> p.x
1
>>> p.y
2
```

--collections.defaultdict([func])

使用dict时，如果引用的Key不存在，就会抛出KeyError。如果希望key不存在时，返回一个默认值，就可以用defaultdict

FE:

```
>>> from collections import defaultdict
>>> dd = defaultdict(lambda: 'N/A')
>>> dd['key1'] = 'abc'
>>> dd['key1'] # key1存在
'abc'
>>> dd['key2'] # key2不存在，返回默认值
'N/A'
```

--collections.OrderedDict()

OrderedDict的Key会按照插入的顺序排列，不是Key本身排序

新类中popitem被重写,last=True从，最后参数弹出；否组，从第一个弹出

FE:

```
>>> od = OrderedDict()
>>> od['z'] = 1
>>> od['y'] = 2
>>> od['x'] = 3
>>> list(od.keys()) # 按照插入的Key的顺序返回
```

['z', 'y', 'x']
>>>od.popitem(last=False)
{'z':1}

**********2017/5/24//////////////////////////////////