第4章 字典和集合

刘　卉

huiliu@fudan.edu.cn

主要内容

- 4.1 字典
- 4.2 再谈内置方法sorted
- 4.3 集合

4.1 字典: 当索引不好用时

- 如何保存一个电话号码簿?

Alice 2341
Beth 9102
Cecil 0142

  - 用列表

```
phonebook=[('Alice','2341'),('Beth','9102'),('Cecil','0142')]
```

    - 需要获取Beth的号码时: phonebook[1][1]. 不直观
  - 我们希望是: phonebook['Beth']
    - 通过关键词而不是编号来引用值
  - A **dictionary** is like a list, but more general. In a list, the indices have to be integers; in a dictionary they can be (almost) any type.

```
>>> phonebook = {'Alice':'2341','Beth':'9102','Cecil':'0142'}
>>> phonebook['Beth']
'9102'
```

主要内容

- 4.1.1 字典的创建与删除
- 4.1.2 字典元素的读取
- 4.1.3 字典元素的添加与修改
- 4.1.4 字典应用案例
- 4.1.5 有序字典

字典(dict)

无序的可变序列, 也称为映射(mapping)类型

字典中的每个元素包含两部分: 键和值

键(keys)是不可变对象(数据) , 且不允许重复

- 整数, 实数, 复数, 字符串, 元组, ...
- 不同元素的key类型可以不同.
- 无序: 不是按照key的大小顺序排列, 而是按照key的hash值排列.

值(values)可以是任何对象

- 每个key对应一个值, 通过key查询到其对应的值.

[例] Python环境中, 许多内置函数和变量使用了dict

sys.modules: 返回当前已加载的模块名与模块对象的映射.	globals(): 返回包含当前作用域内所有全局变量和值的字典.	locals(): 返回包含当前作用域内所有局部变量和值的字典.

4.1.1 字典的创建与删除

1) 通过dict literal创建dict对象

- 大括号界定, 元素用key:value表示, 元素间用逗号分隔

```
{key1:value1, key2:value2, … , keyN:valueN }
e.g. d1 = {1:'food', 2:'drink', 3:'fruit' }
     d2 = {}    # 空字典
     d3 = {'name':'Steve', 'age':25, 'gender':'male',
      'address':{'city':'shanghai', 'zipcode':'200433'},
      1:'note1', 2:'note2', '1':'xx1', '2':'xx2' }
```

嵌套的字典

2) 通过dict()函数创建dict对象

- dict(): 创建空字典
- dict(iterable): 新字典的元素来自于iterable，每个元素必须包括两个子元素
- dict(mapping): 从字典对象mapping创建一个新字典对象
- dict(**kwargs): 从关键字参数kwargs创建一个新字典对象
  - 关键字参数 keyvar=value对应的字典元素: 'keyvar': value

```
e.g. phonebook=dict([('Alice','2341'), ('Beth','9102'), ('Cecil','0142')])
     phonebook=dict(zip(['Alice','Beth','Cecil'], ['2341','9102','0142']))
      phonebook=dict({'Alice':'2341', 'Beth':'9102', 'Cecil':'0142'})
      phonebook=dict(Alice='2341', Beth='9102', Cecil='0142')
```

3) 通过dict类的方法创建新字典

dict.fromkeys(seq[, value])

- 每个元素的key来自于序列对象seq, 值设置为value, 缺省为None.

```
>>> dict1 = dict.fromkeys(['name', 'age', 'gender'])
>>> dict1
{'age': None, 'name': None, 'gender': None}
>>> dict2 = dict.fromkeys(range(5), 10)
>>> dict2
{0: 10, 1: 10, 2: 10, 3: 10, 4: 10}
```

- 使用del命令
  - 删除整个字典: `del dict2`
  - 删除某个元素: `del dict1['name']`
- len(d): 返回字典中元素个数
- 序列解包对字典同样有效

```
>>> b, c, d = {'a': 1, 'b': 2, 'c': 3} # 保存键，丢弃值
>>> b, c, d
('a', 'b', 'c')
```

字典支持的操作	说明
d[key]	返回key对应的value, key不存在抛出KeyError
d[key] = value	设置字典中键为key的元素的值为value, 若不存在, 则添加key: value元素
d.keys()	返回可迭代对象, 元素为字典中所有的key
d.values()	返回可迭代对象, 元素为字典中所有的value
d.items()	返回可迭代对象, 元素为(key, value)元组
key in d key not in d	判断字典d中有/没有键为key, 返回True/False, 相当于 key in d.keys()
d.get(key[, default])	返回key对应的value, key不存在时返回default, 缺省为None
d.setdefault(key[, default])	返回key对应的value, key不存在时添加key:default(缺省None)元素
d.update(other)	用另一字典或元素(key:value)更新字典, 返回None
del d[key]	删除元素, 若key不存在, 抛出KeyError
d.popitem()	移走并返回某一个(key:value)
d.pop(key[, value])	若key存在, 删除对应元素并返回值; 否则返回value, 若没有给定value, 则抛出KeyError
d.clear()	清除所有元素
d.copy()	复制产生新字典并返回

4.1.2 字典元素的读取: 下标

- 以键作为下标访问字典元素, 若键不存在则抛出异常KeyError

```
>>> aDict = {'name': 'Dong', 'gender': 'male', 'age': 37}
>>> aDict['name']
'Dong'
>>> aDict['address ']
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    aDict['address']
KeyError:  'address'
```

d.get(key[, default])	返回key对应的value, key不存在时返回default(缺省为None)
d.setdefault(key[, default])	返回key对应的value, key不存在时添加元素key:default(缺省为None)

```
>>> aDict = {'name': 'Dong', 'gender': 'male', 'age': 37}
>>> print(aDict.get('address'))   # 仅读取元素值
None
>>> print(aDict.get('address', 'SDIBT'))
SDIBT
>>> aDict
{'name': 'Dong', 'gender': 'male', 'age': 37}
>>> aDict.setdefault('address', 'SDIBT')  # 读取/设置元素值
'SDIBT'
>>> aDict
{'name': 'Dong', 'gender': 'male', 'age': 37, 'address': 'SDIBT'}
```

d.keys()	返回可迭代对象，元素为字典中所有的key
d.values()	返回可迭代对象，元素为字典中所有的value
d.items()	返回可迭代对象，元素为包括了(key, value)的元组

```
d = {'name': 'Dong', 'gender': 'male', 'age': 37}
```

```
for item in d.items():
    print(item)
```

```
('age', 37)
('gender', 'male')
('name', 'Dong')
```

```
for key, value in d.items():
    print(key, value)
```

```
age 37
gender male
name Dong
```

```
d = {'name': 'Dong', 'gender': 'male', 'age': 37}
```

```
for key in d:
    print(key)
```

```
for key in d.keys():
    print(key)
```

```
age
gender
name
```

```
for value in d.values():
    print(value)
```

```
37
male
Dong
```

```
>>> list(d.items())
[('age', 37), ('gender', 'male'), ('name', 'Dong')]
>>> list(d)
['age', 'gender', 'name']
```

4.1.3 字典元素的添加与修改

- d[key] = value
  - 若键存在, 修改该键的值; 若不存在, 则添加一个键值对.

```
>>> d = {'name': 'Dong', 'gender': 'male', 'age': 37}
>>> d['age'] = 38      # 修改
>>> d
{'age': 38, 'name': 'Dong', 'gender': 'male'}
>>> d['address'] = 'SDIBT'      # 添加
>>> d
{'age':38, 'address':'SDIBT', 'name': 'Dong', 'gender': 'male'}
```

- d.update(another)
  - 用另一个字典another的键值对修改/添加到当前字典对象d

```
>>> d={'age':37, 'score':[98, 97], 'name':'Dong', 'gender':'male'}
>>> d.update({'age': 38, 'city': 'shanghai'})
>>> d
{'age': 38, 'score': [98, 97], 'name': 'Dong', 'city': 'shanghai',
   'gender': 'male'}
```

del d[key]	删除元素，若key不存在，抛出KeyError
d.popitem()	移走并返回某个(key, value)对
d.pop(key[, value])	若key存在，删除对应元素并返回值；否则返回value，若没有给定value，则抛出KeyError
d.clear()	清除所有元素

```
>>> d = { 'age':38, 'score':[98, 97], 'name':'Dong',
   'city':'shanghai', 'gender':'male'}
>>> d.popitem()
('gender', 'male')
>>> d.pop('score', [])
[98, 97]
>>> d.pop('score', [])
[]
```

字典推导式

```
>>> {i:i+1 for i in range(4)}
{0: 1, 1: 2, 2: 3, 3: 4}

>>> {i:j for i, j in zip(range(1,6), 'abcde')}
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}

>>> {i:j.upper() for i, j in zip(range(1,6), 'abcde')}
{1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E'}
```

4.1.4 字典应用案例

[例] 生成包含1000个随机字符的字符串, 统计每个字符的出现次数.

- 方法一: 以"字符:次数"键值对的形式保存到字典中.

字符串联接函数——join(.)

- S.join(iterable) -> str
  - Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

e.g. 已知li = ["apple", "peach", "banana", "pear"], 要求输出"apple, peach, banana, pear".

```
>>> li = ["apple", "peach", "banana", "pear"]
>>> sep = ", "
>>> s = sep.join(li)  #将li元素连接成一个字符串，元素之间用sep分隔
>>> s
"apple, peach, banana, pear"
```

```
import string
import random
charset = string.ascii_letters + string.digits + string.punctuation
print('\n可使用的字符集: \n', charset)
charList = [random.choice(charset) for i in range(1000)]
randString = ''.join(charList)  # 由字符列表构成字符串
freq = dict()  # 定义空字典
for ch in randString:
    freq[ch] = freq.get(ch, 0) + 1
    # 若ch不在字典中,添加键值对(ch:1);否则,将freq[ch]增加1
print('\n随机产生的字符串为: \n', randString)
print('\n方法一的统计结果为: \n', freq)
```

```
if ch not in freq:
    freq[ch] = 1
else:
    freq[ch] += 1
```

- 方法二：使用collections模块的defaultdict类(提供缺省值的dict)实现该功能.

```
from collections import defaultdict
freq = defaultdict(int)  # 字典freq每个键的默认值为整型值0
for ch in randString:
    freq[ch] += 1
print('\n方法二的统计结果为: \n', freq)
```

- 方法三：使用collections模块的Counter类，自动统计次数并按次数降序排列. 还能满足其它需要, 如: 查找出现次数最多的元素.

```
from collections import Counter
freq = Counter(randString)
print('\n方法三的统计结果为: \n', freq)  # 按出现次数降序排列
print(freq.most_common(3))  # 出现次数最多的前3个字符
```

[例] 已知一个包含学生成绩的字典，求最高分、最低分、平均分，并查找所有最高分学生.

```
scores = {"Moo": 45, "Norman": 78, "Olson": 40, "Peerson": 96,
          "Russel": 65, "Thomas": 90, "Vaughn": 78,
          "Westerly": 96, "Baker": 60}
highest = max(scores.values())
lowest = min(scores.values())
average = sum(scores.values())/len(scores)
highestPerson = [name for name, score in scores.items()
                 if score == highest]
print('Max score:', highest)
print('Min score:', lowest)
print('Average score:', average)
print('Students with max score:', highestPerson)
```

4.1.5 有序字典(可选)

- Python的内置字典是无序的
- collections.OrderedDict: 可记住元素插入顺序的字典

```
>>> x = dict() #无序字典
>>> x['a'] = 3
>>> x['b'] = 5
>>> x['c'] = 8
>>> x
{'b': 5, 'c': 8, 'a': 3}
```

```
>>> import collections
>>> x = collections.OrderedDict() #有序字典
>>> x['a'] = 3
>>> x['b'] = 5
>>> x['c'] = 8
>>> x
OrderedDict([('a', 3), ('b', 5), ('c', 8)])
```

- 从Python 3.6开始, 字典的Key将会保留插入时候的顺序.

再谈内置方法sorted()

sorted(iterable, key=None, reverse=False)

- Return a new list containing all items from the iterable in ascending order.
- A custom *key* function can be supplied to customize the sort order, and the *reverse* flag can be set to request the result in descending order.

- 可对元组/列表/字典/zip对象等排序, 返回新的列表.
- 借助key参数可实现自定义排序: 根据key(value)的返回值确定顺序.
- 借助reverse参数可实现降序排序.

[例] 根据列表name的值对score元素排序

- 先用zip组合对应位置的元素, 然后按name排序, 最后输出score.

```
name = ["Thomas", "Olson", "Vaughn", "Russel"]
score = [90, 89, 81, 72]
pairs = zip(name, score)  # 构成"姓名-成绩"zip对象
pairs = sorted(pairs)  # 按照"姓名"排序zip对象,返回排好序的列表
print(pairs)
# 每个列表元素的"成绩"项构成新列表
result = [student[1] for student in pairs]
# result = [score for name, score in pairs]
print(result)
```

sort_by_name.py

```
# 方法二：根据字典的'键'排序，输出排序后的'值'
report = dict(zip(name, score))  # 根据列表name和score创建字典report
report = sorted(report.items())  # report从字典变为列表
print([score for name, score in report])
```