

第三讲 - Python 集合与字典

张建章

阿里巴巴商学院

杭州师范大学

2023-09



- 1 集合
- 2 集合的基本操作
- 3 字典
- 4 字典的基本操作
- 5 字典的常用内置方法
- 6 深复制和浅复制
- 7 Python 中的自助函数

集合：一对 `{}` 中包含由 `,` 分割的一系列元素。

- 集合可以包含任意多个、无序的非重复元素；
- 集合中的元素可以是多种基本数据类型，如字符串，整数，布尔值，`None`；
- 集合中的元素不可以是可变数据类型，如列表，字典；
- 可以使用 `{}` 定义集合，也可以使用 `set` 函数将字符串、元组、列表等序列转换为集合。

```
# 定义一个集合
```

```
set1 = {1, 2, (1,2), True}
```

```
# 使用set函数将字符串转换为字符集合
```

```
chr_set = set('zjzhang') # {'a', 'g', 'h', 'j', 'n', 'z'}
```

集合中的元素无序，故不能使用索引访问其中的元素，但可以使用 `for` 循环遍历其中的元素，亦可以使用成员判断操作符 `in` 判断某个元素是否在集合中。

- 内置函数 `len` 可统计集合中的元素个数；
- `max` 可返回集合中的最大元素；
- `sum` 可求数值集合中所有元素的和；

```
chr_set = set('zjzhang') # {'a', 'g', 'h', 'j', 'n', 'z'}  
for char in chr_set:  
    print(char)  
  
'o' not in chr_set # True
```

集合常用的方法有：

- **add**，添加一个元素到集合中；
- **remove**，移除集合中的一个指定元素，不存在该元素时报错；
- **discard**，移除集合中的一个指定元素，不存在该元素时不报错；
- **pop**，随机移除集合中一个元素；
- **clear**，清空集合中的全部元素；

这些方法的运行结果会改变集合的内容。

```
chr_set = set('zjzhang') # {'a', 'g', 'h', 'j', 'n', 'z'}
chr_set.add('c') # {'a', 'c', 'g', 'h', 'j', 'n', 'z'}
chr_set.remove('a') # {'c', 'g', 'h', 'j', 'n', 'z'}
chr_set.remove('q') # 报错, KeyError
chr_set.discard('a') # {'c', 'g', 'h', 'j', 'n', 'z'}
chr_set.discard('z') # {'c', 'g', 'h', 'j', 'n'}
chr_set.pop() # 随机删除一个元素并返回该元素
chr_set.clear() # set()
```

Python 提供了求交集、并集、差集和对称差集等集合运算：

- 使用 `s1.intersection(s2)` 或者 `s1&s2` 可以计算两个集合的交集；
- 使用 `s1.union(s2)` 或者 `s1|s2` 可以计算两个集合的并集；
- 使用 `s1.difference(s2)` 或者 `s1-s2` 可以计算两个集合的差集；
- 使用 `s1.symmetric_difference(s2)` 或者 `s1^s2` 可以计算两个集合的对称差集。

字典 (dict): 一对 `{}` 中包含由 `,` 分割的一系列**无序**键值对 (key-value pair), 一个字典中可以包含**任意多个**键值对, 字典使用关键字 (key) 来访问、更新、删除值 (value)。

```
# 定义一个字典
dict1 = {115: '张三', 116: '李四', 119: '王五'}
# 通过key访问字典中的值
dict1[115] # 张三
# 通过key更新字典中的值
dict1[115] = '张七' # {115: '张七', 116: '李四', 119: '王五'}
# 通过key删除字典中的值
dict1.pop(115) # {116: '李四', 119: '王五'}
# 字典中的值不一定是唯一的
dict1[120] = '王五' # {116: '李四', 119: '王五', 120: '王五'}
# 字典中的关键字是唯一的
dict1[120] = '王四' # {116: '李四', 119: '王五', 120: '王四'}
```

注意: 字典中的关键字 (key) 是唯一的 (unique), 值不一定是唯一的, 即一个关键字只能对应一个值, 但是同一个值可以对应多个关键字。

类似于之前的列表、元组、集合，字典也可以使用 Python 内置函数 `dict` 创建，其接收的参数形式如下：

```
[(key1, value1), (key2, value2), ...]。
```

定义字典时，重复的键只保留其中一个，并且，键只能是不可变对象，如，之前学习过的基本数据类型（数值、字符串等）。列表、集合是可变对象，不能作为字典的键，字典的值 (**value**) 可以是任意的数据类型。

```
# 使用Dict函数定义一个字典
dict2 = dict([(116, '李四'), (119, '王五'), (120, '赵六')])
# 使用zip函数将两个列表转化为dict函数接收的参数形式
dict(zip([116, 119, 120], ['李四', '王五', '赵六'])) # {116:
↪ '李四', 119: '王五', 120: '赵六'}
# 重复的键只会保留一个
dict3 = {115: '张三', 115: '李四', 115: '王五'} # {115: '王五'}
# 字典的键只能是不可变对象
dict4 = {[1,2]: [1,2]} # Error
# 字典的值可以是任意类型
dict5 = {'list1': [1,2,3]}
```


字典的增、删、改、查操作通过关键字 (key) 进行，类似于通过索引 (index) 操作列表。

```
# 创建一个空字典
happy_dict = {}
# 新增键值对
happy_dict[101] = '沈腾'
happy_dict.update({102: '马丽'})
happy_dict.update(dict(zip([103, 104], ['艾伦', '常远'])))
happy_dict[105] = '宋小宝'
# 修改值
happy_dict[105] = '王宁'
# 通过键查找其对应的值
happy_dict[102]
happy_dict.get(103)
# 删除键值对
happy_dict.pop(105)
del happy_dict[104]
```

字典存储数据实例

创建一个字典 *company_info*, 存储一家公司信息

```
company_info = {'name': 'TechCorp', 'city':  
    ↪ 'Shanghai', 'employees': 500}
```

增加一个键值对, 增加公司的年收入信息-2000万

```
company_info['annual_revenue'] = 20000000
```

删除一个键值对, 删除公司的员工人数信息

```
del company_info['employees']
```

修改某个键对应的值, 修改公司地址为背景

```
company_info['city'] = 'Beijing'
```

根据键查看值的信息, 查看公司名称

```
company_info['name']
```

新增一个键值对, 值的数据类型为列表

```
company_info['products_services'] = ['软件开发', '数据分析',  
    ↪ '云服务']
```

字典存储数据实例

新增一个键值对，值的数据类型为字典

```
company_info['financial_status'] = {'资产': 300000000, '负债':  
↪ 100000000}
```

根据键查看值的信息，查看公司名称

如果键 'founder' 存在，则返回对应的值，如果不存在，返回 '未知'

```
company_info.get('founder', '未知')
```

根据键查看值的信息，查看公司名称

如果键 'founder' 存在，则返回对应的值，如果不存在，返回 'Tom'，
↪ 并新增键值对 founder:Tom

```
company_info.setdefault('founder', 'Tom')
```

更新字典信息，使用一个已有字典更新

```
company_info.update({'profit': 10000000, 'market_share': '20%'})
```

更新字典信息，使用一个二元组列表更新

```
company_info.update(zip(['brand_value', 'customer_satisfaction'],  
↪ [5000000, 4.5]))
```

4. 字典的基本操作

字典推导式的基本形式同列表推导式，唯一不同的是表达式的格式，**key:value**。

可以使用 **in** 或 **not in** 判断某个键 (key) 是否在字典中，可以使用 **==** 判断两个字典中是否包含一样的键值对 (key-value pairs)。

```
actor_list = ['沈腾', '马丽', '艾伦', '常远', '王宁']
start_id = 101
happy_dict = {start_id+idx:actor for idx,actor in
    ↪ enumerate(actor_list)}
# {101: '沈腾', 102: '马丽', 103: '艾伦', 104: '常远', 105:
    ↪ '王宁'}
```



```
# 判断键是否在字典中
'沈腾' in happy_dict # False
102 in happy_dict # True
```



```
# 判断两个字典的内容是否一样
{101:'沈腾', 102:'马丽'} == {102:'马丽', 101:'沈腾'} # True,
    ↪ 字典是无序的
```

内置方法的语法为 变量名. 方法名

- `copy`，复制一份字典，不受原字典变化影响；
- `get`，根据键获取对应的值，键不存在，返回默认值；
- `setdefault`，根据键获取对应的值，键不存在，返回默认值，更新字典；
- `keys`，返回字典中的全部键；
- `values`，返回字典中的全部值；
- `items`，返回字典中的全部键值对；
- `pop`，删除字典中的指定键值对；
- `popitem`，以后进先出的顺序删除字典中的一个键值对；
- `update`，使用一个字典更新当前字典；
- `clear`，清空字典；

5. 字典的常用内置方法

```
happy_dict = {101: '沈腾', 102: '马丽', 103: '艾伦', 104: '常远',  
             ↪ 105: '王宁'}  
new_happy_dict = happy_dict.copy()  
happy_dict.get(101) # 沈腾  
happy_dict[101] # 键不存在则Error, 存在则返回值  
happy_dict.get(109, '不存在此键') # 键存在, 则返回对应的值,  
↪ 键不存在, 则返回第二个参数指定的值, 第二个参数默认值为None  
happy_dict.setdefault(101) # 沈腾  
happy_dict.setdefault(106, '魏翔') # 键存在, 则返回对应的值,  
↪ 键不存在, 则返回第二个参数指定的值, 第二个参数默认值为None,  
↪ 同时更新字典  
# {101: '沈腾', 102: '马丽', 103: '艾伦', 104: '常远', 105:  
↪ '王宁', 106: '魏翔'}  
happy_dict.keys() # dict_keys([101, 102, 103, 104, 105, 106])  
happy_dict.values() # dict_values(['沈腾', '马丽', '艾伦',  
↪ '常远', '王宁', '魏翔'])  
happy_dict.items() # dict_items([(101, '沈腾'), (102, '马丽'),  
↪ (103, '艾伦'), (104, '常远'), (105, '王宁'), (106, '魏翔')])  
happy_dict.pop(106) # 返回106对应的值, 魏翔, key不存在, 则Error  
happy_dict.popitem() # 返回被删除的键值对, (105, '王宁')
```

```
happy_dict.update({107: '杜晓宇', 108: '黄才伦'})  
happy_dict.update({108: '魏翔'}) # 更新108对应的值,  
↪ 由 '黄才伦' 变为 '魏翔'  
happy_dict.clear() # happy_dict 变为一个空字典
```

```
# {}表示空字典  
d = {}  
type(d) # dict  
# 空集合是set()  
s = {1, 2}  
type(s) # set  
s.clear()  
print(s) # set()  
type(s) # set
```

Python 中的赋值语句不复制对象，它们在目标 (变量) 和对象 (数据) 之间创建绑定。

对于可变 (如，列表) 或包含可变项的复合对象 (如，列表嵌套)，有时需要一个副本，以便可以更改一个副本而不更改原数据对象。

注意：浅复制和深复制之间的区别仅与复合对象有关，典型的复合对象有字典中包含列表，列表中包含列表，元组中包含列表作为元素。

```
# 创建一个列表嵌套对象
list1 = [['John', 21], ['Mary', 19]]
# 引用赋值
list2 = list1
# 深复制
import copy
list3 = copy.deepcopy(list1)
# 浅复制-切片操作
list4 = list2[:]
# 浅复制-列表推导式
list5 = [item for item in list1]
```


6. 深复制和浅复制

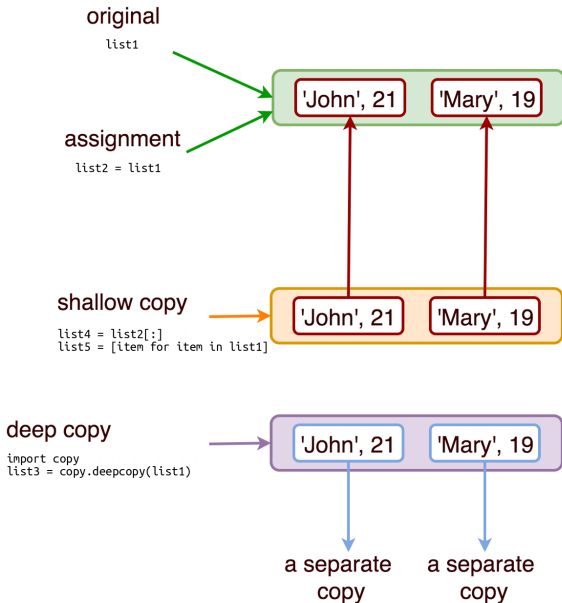


图 1: 赋值、深复制、浅复制示例

```
list1[0][0] = 'Tom' # [['Tom', 21], ['Mary', 19]]
print(list2) # [['Tom', 21], ['Mary', 19]]
print(list3) # [['John', 21], ['Mary', 19]]
print(list4) # [['Tom', 21], ['Mary', 19]]
print(list5) # [['Tom', 21], ['Mary', 19]]
```

```
list2 is list1 # True
list3 is list1 # False
list4 is list1 # False
list5 is list1 # False
```

```
list5[0] is list1[0] # True
list4[0] is list1[0] # True
list3[0] is list1[0] # False
list2[0] is list1[0] # True
```

对于包含可变项的复合对象，只有 `copy` 模块的 `deepcopy` 才能实现深复制，其他复制操作均为浅复制，如，切片、推导式等，字典的内置方法 `copy` 也是浅复制：

注意：包含可变项的复合对象是可变数据类型，也不能作为集合的元素，不能作为字典的键 (key)，如，`(1,2,[3,4])`；

如果一个字典中包含列表作为值，各种复制操作的结果与代码中的列表嵌套示例类似。请自行实验

对 `dict6 = {'101': ['Tom', 'Mary', 'Alan']}` 的各种复制操作并查看相应的结果。

使用 Python 的一些内置函数辅助了解相关模块和函数的用法：

- 使用内置函数 `dir` 可以快速查询各种内置数据类型的方法有哪些；
- 使用特殊变量 `__doc__` 可以快速查看各种方法的使用说明；
- 使用内置函数 `help` 查看函数或模块用途的详细说明；
- 使用内置函数 `type` 查看对象的数据类型；

THE END