第四章 字典

4.1 内建的映射类型:字典

4.2 创建字典

4.3 字典的基本操作

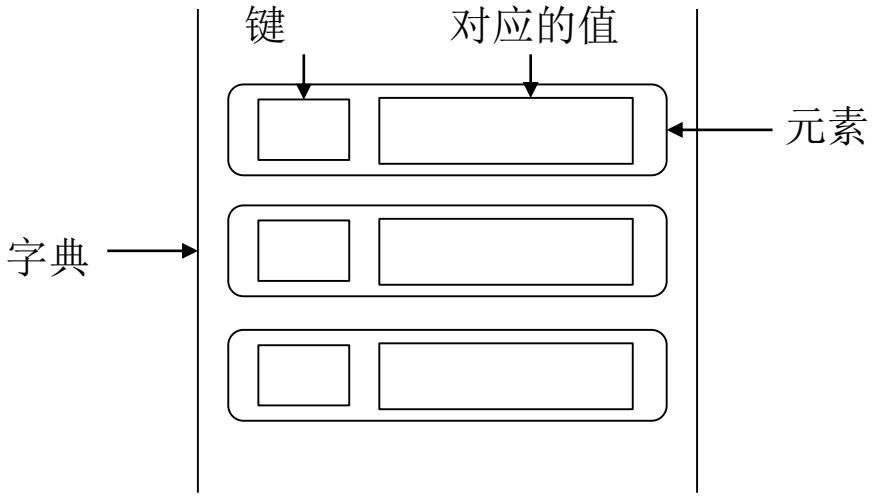
4.4 字典的其他重要方法

字典的概念

- 与列表一样，字典是Python中最为灵活的内建数据类型之一：如果说列表是数据元素的有序集合，那么字典可以看作是数据元素的无序集合
- 在列表中我们存取和访问某个对象需要利用与之相关的位移信息（即索引），而在字典中，存取和访问对象依据的是与之相关的键（key）
- 列表的作用和其他语言中的数组类似，而字典多用于使用记录、查找表以及数据融合的场合，此时数据元素的名字比它们的位置信息更有意义

字典的概念

- 字典是存放键/值对（key/value pairs）的集合，可以通过键对字典中的值进行快速的检索、删除和更新
- 字典中不允许有重复的键，每个键和对应的值形成字典中的一个元素（item）或条目（entry）



第四章 字典

4.1 内建的映射类型:字典

4.2 创建字典

4.3 字典的基本操作

4.4 字典的其他重要方法

字典的构造方法

```
>>> D = {}                                          # 空字典
>>> D = {'name':'Bob', 'age':40}                    # 构造包含两个数据元素的字典
>>> E = {'cto':{'name':'Bob', 'age':40}} # 字典中的嵌套
>>>
>>> print(D)
{'age': 40, 'name': 'Bob'}
>>> print(E)
{'cto': {'age': 40, 'name': 'Bob'}}
>>>
>>> D = dict(name = 'Bob', age = 40)     # 字典的第二种构造方法
>>> E = dict([('name','Bob'),('age',40)]) # 字典的第三种构造方法
>>>
>>> print(D)
{'age': 40, 'name': 'Bob'}
>>> print(E)
{'age': 40, 'name': 'Bob'}
```

使用给定的键新建字典

- fromkeys方法使用给定的键建立新的字典，此时每个键对应一个默认的None值

```
>>> {}.fromkeys(['name', 'age'])  # 构造一个空字典,然后调用其fromkeys方法建立一个新字典
{'name': None, 'age': None}
>>> dict.fromkeys(['name', 'age'])   # 也可以直接使用dict来调用fromkeys方法
{'name': None, 'age': None}
>>> dict.fromkeys(['name', 'age'], '(unknown)')   # 可以自行提供关键字对应的默认值
{'name': '(unknown)', 'age': '(unknown)'}
>>>
```

字典的重要特点

- 字典有时也被称为关联数组（associative array）或哈希表（hash）,它将值与键相关联，这样你可以利用当初存放某个数据元素时使用的键从字典中取出该数据元素
- 字典中的数据元素并没有按照特定的次序存放，Python利用伪随机方法将它们从左至右摆放从而获得快速的查询效果
- 字典可以根据需要扩展与收缩（此时并不产生新的拷贝），其包含的数据元素可以是任意类型的对象，可以以任意层次嵌套

字典的重要特点

- 字典中每个键只与一个值相关联（这个值可以是多个对象的一个集合），但是一个值可以存放在多个键之下
- 字典不支持分片操作，因为其自身是数据元素的无序集合，字典仅支持将键映射到值上
- 字典中存放的是对象引用的无序集合，在Python内部字典是作为哈希表来实现的，尽管开始很小，但是可以根据需要扩展

字典的一般用法

```
>>> D = {'spam':20, 'cheese':30, 'egg':40}
>>> len(D)                  # 获取字典中元素的个数
3
>>> 'spam' in D             # 测试字典中是否包含给定的键,而不是值!
True
>>> D['cheese']             # 获取与给定键对应的值
30
>>> D['eggs'] = 50          # 将新的值关联到字典中的键'eggs'上
>>> D
{'eggs': 50, 'egg': 40, 'spam': 20, 'cheese': 30}
>>> D['sausage'] = 60       # 将新的值关联到字典中原本不存在的键'sausage'上
>>> D
{'eggs': 50, 'sausage': 60, 'egg': 40, 'spam': 20, 'cheese': 30}
>>> del D['cheese']         # 删除字典中与给定关键字相关联的元素
>>> D
{'eggs': 50, 'sausage': 60, 'egg': 40, 'spam': 20}
>>> del D['apple']          # 删除不存在的元素
Traceback (most recent call last):
  File "<pyshell#125>", line 1, in <module>
    del D['apple']          # 删除不存在的元素
KeyError: 'apple'
```

```
# 一个简单的数据库,其中字典使用人名作为键,每个人再用一个字典表示,其中键'phone'和'addr'分别表示
# 它们的电话号码和地址, 以下为运行示例
#    Name: Beth
#    Phone number (p) or address (a)? p
#    Beth's phone number is 9102

people = {
    'Alice': {
        'phone':'2341',
        'addr': 'Foo drive 23'
    },
    'Beth': {
        'phone':'9102',
        'addr':'Bar street 42'
    },
    'Cecil':    {
        'phone':'3158',
        'addr':'Baz avenue 90'
    }
}
# 针对电话号码和地址使用的描述性标签,会在打印时用到
labels = {
    'phone':'phone number',
    'addr':'address'
    }

name = input('Name:')

# 查找电话号码还是地址?
request = input('Phone number (p) or address (a)? :')

# 使用正确的键
if request == 'p': key = 'phone'
if request == 'a': key = 'addr'

# 如果名字是字典中的有效键,才可以打印出相关信息
if name in people: print("%s's %s is %s." % (name, labels[key], people[name][key]))
```

第四章 字典

4.1 内建的映射类型:字典

4.2 创建字典

4.3 字典的基本操作

4.4 字典的其他重要方法

对字典进行复制

- copy方法返回一个具有相同键/值对的字典，此方法实现的是浅拷贝（shallow copy）

- 浅拷贝仅仅拷贝父级对象，不会拷贝父级对象内部的子对象，即此时获取的是子对象的引用

```
>>> x = {'username':'admin', 'machines':['foo', 'bar', 'baz']}
>>> y = x.copy()          # 浅拷贝
>>> y['username'] = 'mlh'      # 对y中与键'username'相关联的值进行替换(父对象改变)
>>> x
{'machines': ['foo', 'bar', 'baz'], 'username': 'admin'}
>>> y
{'machines': ['foo', 'bar', 'baz'], 'username': 'mlh'}
>>> y['machines'].remove('bar') # 移除y中与键'machines'相关联的值(子对象改变)
>>> x
{'machines': ['foo', 'baz'], 'username': 'admin'}
>>> y
{'machines': ['foo', 'baz'], 'username': 'mlh'}
```

浅拷贝

浅拷贝的实现方式有三种:

(1) 对于列表、字符串和元组可以使用分片操作[:]；

(2) 对于字典，可以使用copy方法；

(3) 对于列表可以使用函数list()，而字典则可以使用函数dict()

可以尝试下面例子:

```
>>> jack = ['jack', ['age', 20]]
>>> tom = jack[:]
>>> anny = list(jack)
```

对字典进行复制

- 如果需要保留原始字典的副本不变，则可以使用深拷贝（deep copy），利用copy模块中的deepcopy函数来保留原始字典的副本，原始字典若发生改变，该副本不变

```
>>> from copy import deepcopy
>>> d = {}
>>> d['name'] = ['Alfred', 'Bertrand']
>>> c = d.copy()
>>> dc = deepcopy(d)
>>> d['name'].append('Clive')
>>> c                    # 通过浅拷贝得到的c会和原始字典d一同改变
{'name': ['Alfred', 'Bertrand', 'Clive']}
>>> dc                   # 通过深拷贝得到的dc不会因为原始字典d改变而改变
{'name': ['Alfred', 'Bertrand']}
>>>
```

深拷贝

- 深拷贝不仅拷贝父级对象，而且拷贝父级对象内部的子对象
- 深拷贝和浅拷贝都是对源对象的复制，这些拷贝和源对象占用不同的内存空间

清除字典中的所有项

- clear方法清除字典中所有的项（即键/值对），这是个原地操作（即对字典对象本身进行改变），该方法无返回值

```
>>> d = {}
>>> d['name'] = 'Gumby'
>>> d['age'] = 42
>>> d
{'name': 'Gumby', 'age': 42}
>>> print(d.clear())    # 显示clear方法的返回值
None
>>> d = {}          # 将要显示另外一种清空字典的方法
>>> y = d
>>> d['key'] = 'value'
>>> y
{'key': 'value'}
>>> d = {}          # 将d关联到一个新的空字典即为置空
>>> print(y)
{'key': 'value'}
>>> # 此时y仍然关联到原来的字典,因此不为空
>>>
```

获取字典中所有的键、值以及键/值对

- keys方法以对象的形式返回字典中所有的键，values方法以对象的形式返回字典中所有的值，而items方法则仍然以对象的形式返回字典中所有的键/值对

```
>>> d = dict([('x', 10), ('y', 20), ('z', 30)])
>>> d.keys()                    # 返回字典d的所有关键字
dict_keys(['z', 'x', 'y'])
>>> d.values()                  # 返回字典d中所有值
dict_values([30, 10, 20])
>>> d.items()                   # 返回字典d中所有关键字/值对
dict_items([('z', 30), ('x', 10), ('y', 20)])
```

更新字典

- update方法可以利用一个字典项更新另外一个字典

```
>>> d = {
        'title': 'Python Web Site',
        'url':  'http://www.python.org',
        'changed': 'Mar 14 22:09:15 MET 2008'
        }
>>> x = {'title': 'Python Language Website'}  # 将利用此项去更新上面的字典d
>>> d.update(x)
>>> d
{'url': 'http://www.python.org', 'changed': 'Mar 14 22:09:15 MET 2008', 'ti
tle': 'Python Language Website'}
>>>
>>> y = {'reason':  'title updated'}  # 如果用来更新的项中包含的关键字不在
被更新的字典中
>>> d.update(y)
>>> d
{'url': 'http://www.python.org', 'changed': 'Mar 14 22:09:15 MET 2008', 're
ason': 'title updated', 'title': 'Python Language Website'}
>>> # 那么可以看出，update方法会在被更新的字典中添加该项
```

获取字典项

- get方法可以用来访问字典项，如果该项不存在（给定键不存在），系统也不会报错

```
>>> d = {}
>>> print(d['name'])
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in <module>
    print(d['name'])
KeyError: 'name'
>>> # 一般情况下访问字典中不存在的项时会报错
>>>
>>> print(d.get('name'))  # 使用get方法则不存在这个问题,因为此时返回None
None
>>> d.get('name', 'N/A') # 可以自己设置默认值来替换None
'N/A'
```

获取字典项

- pop方法也可以用来访问字典项，从而获取对应于给定键的值，然后将这个键/值对从字典中移除

```
>>> d = dict([('x', 10), ('y', 20), ('z', 30)])
>>> d.pop('y')
20
>>> d
{'z': 30, 'x': 10}
>>> d.pop('a')      # 若试图访问一个字典中不存在的项,则系统报错
Traceback (most recent call last):
  File "<pyshell#68>", line 1, in <module>
    d.pop('a')      # 若试图访问一个字典中不存在的项,则系统报错
KeyError: 'a'
```

弹出一个字典项

- popitem方法与列表的pop方法类似，只是此时popitem弹出字典中随机的一项，而不像列表的pop方法那样弹出最后一个元素，这是因为在字典中各个项之间没有特定次序

```
>>> d = dict(url = 'http://www.python.org', spam = 0, title = 'Python Web S
ite')
>>> d
{'url': 'http://www.python.org', 'spam': 0, 'title': 'Python Web Site'}
>>> d.popitem()
('url', 'http://www.python.org')
>>> d
{'spam': 0, 'title': 'Python Web Site'}
>>> d.popitem()
('spam', 0)
```

弹出一个字典项

- setdefault方法与get方法类似，能够获得与给定键相关联的值；此外，setdefault还能在给定键不存在的情况下设定对应的默认值

```
>>> d = {}
>>> d.setdefault('name', 'N/A')
'N/A'
>>> d
{'name': 'N/A'}
>>> d['name'] = 'Gumby'
>>> d.setdefault('name', 'N/A') # 此时关键字name已经存在
'Gumby'
>>> # 当关键字存在时,setdefault方法返回与该关键字对应的值
```

第四章 字典

4.1 内建的映射类型:字典

4.2 创建字典

4.3 字典的基本操作

4.4 字典的其他重要方法

利用字典进行字符串格式化

- 当进行字符串格式化时，除了使用元组表示希望被格式化的值，还可以使用字典（只以字符串作为键的）

```
>>> phonebook = {'Beth':'9102', 'Alice':'2341', 'Cecil':'3258'}
>>> "Cecil's phone number is %(Cecil)s."  % phonebook
"Cecil's phone number is 3258."
>>> "Beth's phone number is %(Beth)s." % phonebook
"Beth's phone number is 9102."
>>> "Alice's phone number is %(Alicee)s." % phonebook
Traceback (most recent call last):
  File "<pyshell#288>", line 1, in <module>
    "Alice's phone number is %(Alicee)s." % phonebook
KeyError: 'Alicee'
```

- 如上所示，在字符串转换说明符中的%后面加上用圆括号括起来的键，后面再跟其它说明元素

- 当以这种方式使用字典时，只要所有键都存在于字典中，则可以使用任意数量的转换说明符

```
>>> template = '''<html>
<head><title>%(title)s</title></head>
<body>
<h1>%(title)s</h1>
<p>%(text)s</p>
</body>'''
>>> data = {'title':'My Home Page', 'text':'Welcome to my home page!'}
>>> print(template % data)
<html>
<head><title>My Home Page</title></head>
<body>
<h1>My Home Page</h1>
<p>Welcome to my home page!</p>
</body>
```

小结

- 字典用来存储键/值对，你可以使用键来检索某个值
- 字典中的键必须是数字或字符串这样可以被哈希的对象
- 在字典中检索与给定键相关联的值时可以使用**字典名**[**键**]，如果需要在字典中增加或修改某项则可以使用**字典名**[**键**]=**值**
- 使用len方法可以得到一个字典中的项（即键/值对）的数目

小结

- 使用del 字典名[键]可以从字典中删除与给定键相关联的项
- 使用in和not in操作符可以判断给定的键是否在字典中
- 我们还可以使用字典的其他方法，包括copy()、deepcopy()、clear()、keys()、values()、items()、update()、get(key)、pop(key)、popitem()、setdefault(key)
- 可以利用字典进行字符串格式化

本章节涉及到的函数

dict(seq)	使用（键，值）对建立字典

使用字典来存放一段文字中（假设其中只有英文单词、空格、逗号与句点）出现的不同单词及其出现的次数。程序需要判断每个单词是否已经是字典中的键，如果不是，程序应该向字典中添加一项，其中单词作为键，而出现次数则设为数值1。如果字典中已有该键，则只需将其出现次数加1。

要求：

(1) 程序无需区分大小写，即认为The和the是相同的单词。

(2) 程序根据字母顺序排列打印所有单词以及它们的出现次数。

(3) 程序根据单词出现次数的升序打印所有单词以及它们的出现次数。