第七章 映射和集合类型

7.1 字典

字典对象是可变的，能存储任意个数的Python对象。

字典类型和序列类型容器类(列表、元组)的区别是存储和访问数据的方式不同。序列类型只用数字类型的键(索引)。映射类型可以用其他类型做键值。

映射类型中的数据是无序排列的–因为任何一个值存储的地址都取决于它的键值，而哈希表要对简直执行哈希函数操作。

序列类型用有序的数字键做索引将数据以数组的形式储。s索引值与存储的数据毫无关系。

创建一个字典和给字典赋值

```
>>> dict1 = {}
>>> dict2 = {'name':''earth',''port':80}
>>> dict1,dict2
({},{'port':80,'name':'earth'})
```

工厂方法`dict()`也可以直接创建字典。

内建方法 fromkeys() 来创建一个"默认"字典, 字典中元素具有相同的值 (如果没有给出， 默认为 None):

```
>>> ddict = {}.fromkeys(('x', 'y'), -1)
>>> ddict
{'y': -1, 'x': -1}
>>> edict = {}.fromkeys(('foo', 'bar'))
>>> edict
{'foo': None, 'bar': None}
```

访问字典中的值

```
for key in dict2.keys():
    print 'key=%s, value=%s' % (key, dict2[key])
# 以下写法也行
for key in dict2:
    print 'key=%s, value=%s' % (key, dict2[key])
```

试图访问一个字典中并不存在的数据元素，将会产生一个错误。因此最好在访问之前使用`has_key()`(在以后会弃用)或者in/not in来检查。

数字和字符串可以作为字典中的键， 但是列表和其他字典不行。键值必须是可哈希的(可以简单地理解为不可变性)。

更新字典

```
>>> dict2['name'] = 'venus' # 更新已有条目
>>> dict2['port'] = 6969 # 更新已有条目
>>> dict2['arch'] = 'sunos5'# 增加新条目
>>> print 'host %(name)s is running on port %(port)d' % dict2
```

删除字典元素和字典

```
del dict2['name'] # 删除键为“name”的条目
dict2.clear() # 删除 dict2 中所有的条目
del dict2 # 删除整个 dict2 字典
dict2.pop('name') # 删除并返回键为“name”的条目
```

7.2 操作符

不支持；拼接和重复的操作。其他的标准操作符<,>and等支持。

字典的键查找操作符（[]）
键查找操作符是唯一仅用于字典类型的操作符。
对字典类型来说，是用键(key)查询(字典中的元素),所以键是参数(argument), 而不是一个索引(index)(index是对序列类型来说的)。

键查找操作符既可以用于给字典赋值，也可以用于从字典中取值

(键)成员关系操作(in ,not in)

7.3 映射类型的内建函数和工厂函数

标准类型函数[type(),str(),cmp()]

字典的比较cmp():首先是字典的大小，然后是键，最后是值。

字典的比较算法
两个字典相等：它们有相同的大小、相同的键、相同的值，所以 cmp() 返回值是 0
字典的比较在一下方面进行比较:

1. 比较字典的长度-字典中的键的个数越多，这个字典就越大
2. 比较字典的键
3. 比较字典的值

映射类型的相关函数
`dict()`
调用 dict()方法可以接受字典或关键字参数字典

当需要复制一个字典时推荐使用`copy()`方法

`len()`-返回字典所有元素的数目
`hash()`-可以判断某个对象是否可以做一个字典的键

7.4 映射类型内建方法

`keys()`, `values()`, `items()`, `clear()`, `copy()`, `fromkeys()`, `get(key)`, `has_key()`, `iter()`, `pop(key)`, `setdefault(key)`

`update(dict2)`-将一个字典的内容添加到另外一个字典中，update(exist) or insert(not exist)

`sorted()`方法返回一个有序的迭代子

`get()`–方法与键查找`[]`操作符的区别是:`get()`方法对于键不存在的返回None，而`[]`引发异常

当数据集很大时，可以使用`iteritems()`,`iterkeys()`,`itervalues()`方法来处理字典，与返回列表的对应方法相似，只是它们返回惰性赋值的迭代器，所以节省内存。

7.5 字典的键

字典键值的限制：

- 不允许一个键对应多个值
- 键必须是可哈希的(列表，字典这样的可变类型不可作为键，即不可变类型都是可哈希的)
    - 例外：一个对象实现了`__hash__`特殊方法的类是可哈希的

用元组做有效的键，必须要加限制：元组中只包括像数字和字符串这样的不可变参数，才可以作为字典中有效的键。

```
#! /usr/bin/env python

db = {}

def newuser():
    prompt = 'login desired:'
    while True:
        name = raw_input(prompt)
        if db.has_key(name):
            prompt = 'nametaken,try another:'
            continue
        else:
            break
        pwd = raw_input('password:')
        db[name] = pwd

def olduser():
    name = raw_input('login:')
    pwd = raw_input('password:')
    password = db[name]
    if pwd == password:
        print 'welcome back,',name
    else:
        print 'login incorrect'

CMDs = {'n':newuser,'e':olduser}

def showmenu():
    prompt = """
```

```
        N-New User Login
        E-Existing User Login
        Q-Quit
        Enter choice:
        """
        done = False
        while not done:
            chosen = False
            while not chosen:
                try:
                    choice = raw_input(prompt).strip()[0].lower()
                except(EOFError, KeyboardInterrupt):
                    choice = 'q'
                print '\nYou picked:[%s]' % choice

                if choice not in 'neq':
                    print 'invalid option,try again'
                else:
                    chosen = True
                    done = True
                    CMDs[choice]()

    if __name__ == "__main__":
        print 'main'
        showmenu()
```

7.6 集合类型

集合sets类型有可变集合和不可变集合。
可变集合(set)不是可哈希的，因此既不能用做字典的键也不能做其他集合中的元素。
不可变集合(frozenset)则正好相反，即，他们有哈希值，能被用做字典的键或是作为集合中的一个成员。

集合操作符

Python符号	说明
in	是…的成员
not in	不是…的成员
==	等于
!=	不等于
<	是…的(严格)子集
<=	是…的子集
>	是…的(严格)超集
=	是…的超集
&	交集
-	补集
^	对称差分

7.6 集合类型

创建集合类型和给集合赋值
用集合的工厂方法`set()`和`frozen()`来创建集合。

访问集合中的值–in

更新集合
使用内建方法来提来添加删除成员
`add()`,`update()`,`remove()`,`-`-补集

删除集合本身使用del语句

7.7 集合操作符

成员关系(in/ not in)

```
>>> s = set('cheeseshop')
>>> t = frozen('bookshop')
>>> 'k' in s
False
```

集合的等价于不等价<=,>=

子集/超集
Sets 用 Python 的比较操作符检查某集合是否是其他集合的超集或子集。
<符号用于判断子集，>符号用于判断超集。

- 集合类型操作符(所有的集合类型)
- 联合(|)-union-or
  也可以用`union()`方法实现
- 交集(&)-and
  也可以用`intersection()`方法实现
- 差补/相对补集(-)
  也可以用`difference()`方法实现
- 对称差分(^)-xor–只能是属于集合 s 或者集合 t的成员，不能同时属于两个集合
  也可以用`symmetric_difference()`方法实现

Note

> 如果左右两个操作数的类型不相同(左操作数是`set`，右操作数是`frozenset`，或相反情况)，则所产生的结果类型与左操作数的类型相同

集合类型操作符(仅适用于可变集合)

- (Union) Update (|=)
  从已存在的集合中添加(可能多个)成员，和`update()`等价
- 保留/交集更新(&=)
  和`intersection_update()`等价
- 差更新(-=)
  和`difference_update()`等价
- 对称差分更新(^=)
  和`symmetric_difference_update()`等价

7.8 内建函数

标准函数`len()`

集合类型工厂函数`set()`和`frozenset()`
分别用来生成可变和不可变的集合.如果不提供任何参数，默认会生成空集合。如果提供一个参数，则该参数必须是可迭代的，

7.9 集合类型内建方法

和同名的字典方法一样，`copy()`方法比用像`set(),frozenset()`,或`dict()`这样的工厂方法复制对象的副本要快。

因此对于set,dict这样的数据结构，如果要复制一份则建议使用`copy()`函数.

集合类型方法
`issubset()`,`issuperset()`,`union()`,`intersection()`,`difference()`,`symmetric_difference()`

以下方法仅适用于可变集合
`add()`,`remove()`,`discard()`,`pop()`,`clear()`
这些接受对象的方法，参数必须是可哈希的。

操作符和内建方法的比较

内建的方法几乎和操作符等价。
他们之间有一个重要区别：操作符两边的操作数必须是集合;
内建方法时，对象也可以是迭代类型的。

为什么要用这种方式来实现呢？

Python 的文档里写明：采用易懂的`set('abc').intersection('cbs')`可以避免用`set('abc') [and] 'cbs'`这样容易出错的构建方法。

可变集合类型的方法

`update()`,`intersection_update()`,`difference_update()`,`symmetric_defference_update()`加上前面的几个。

7.10 操作符、函数和方法

函数/方法名	等价运算符	说明
len(s)		集合 s 中元素的个数
set([obj])		可变集合工厂函数，由obj中的元素创建一个集合，否则创建一个空集合
fromzenset([obj])		不可变集合工厂函数，予set()相同
	obj in s	成员测试：obj 是 s 中的一个元素吗?
	obj in s	非成员测试：obj 不是 s 中的一个元素吗?
	s == t	等价测试: 测试 s 和 t 是否具有相同的元素?
	s != t	不等价测试: 与==相反
	field2	field3
	s < t	(严格意义上)子集测试; s != t 而且 s 中 所 有的元素都是 t 的成员
s.issubset(t)	s <= t	子集测试(允许不严格意义上的子集): s 中所有的元素都是 t 的成员
	s > t	(严格意义上)超集测试: s != t 而且 t 中所有的元素都是 s 的成员

s.issuperset(t)	s >= t	超集测试(允许不严格意义上的超集): t 中所有的元素都是 s 的成员
s.union(t)	s	t
s.intersection(t)	s & t	交集操作： s 和 t 中的元素
s.difference(t)	s - t	差分操作: s 中的元素，而不是 t 中的元素
s.symmetric_difference(t)	s ^ t	对称差分操作：s 或 t 中的元素，但不是 s 和 t 共有的元素
s.copy()		复制操作
s.update()	s	= t
s.intersection_update(t)	s &= t	交集修改操作: s 中仅包括 s 和 t 中共有的成员
s.difference_update()	s -= t	差修改操作: s 中包括仅属于 s 但不属于 t 的成员
s.symmetric_difference_update()	s ^= t	对称差分修改操作: s 中包括仅属于 s 或仅属于 t 的成员
s.add(obj)		加操作: 将 obj 添加到 s
s.remove(obj)		删除操作: 将 obj 从 s 中删除；如果 s 中不存在obj,将引发 KeyError
s.discard(obj)		丢弃操作: remove()的友好版本
s.pop()		Pop 操作: 移除并返回 s 中的元素
s.clear()		清除操作: 移除 s 中的所有元素

7.11 相关模块

set模块

可继承Set或者ImmutableSet来生成子类。