第 5 章　Python 容器操作

容器(container)是指用以容纳物料并以壳体为主的基本装置。这是一个宽泛的概念。但是在 Python 领域中,许多人将其窄化了,认为只有 list、tuple、dict 和 set 才是容器。实际上,并非如此,上述 4 个类型的对象仅仅是 Python 内置的对象容器。如图 5.1 所示,本章所讨论的 Python 容器包括内存容器和外存容器。它们都是组织数据对象的容器对象,但存放的物理容器不同:一个在内存,一个在外存。

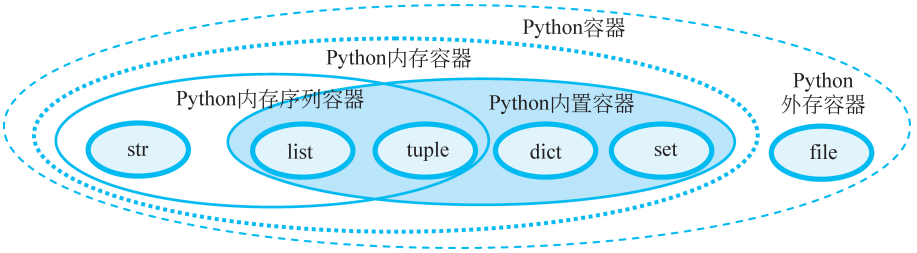

图 5.1　Python 数据对象组织体系

内存容器由 Python 内置的容器类型和 Python 内置的字符串类型数据对象两部分组成。如表 5.1 所列,str 与 list、tuple、dict 和 set 都有壳体,都可以存放数据对象,并且在对存放元素的操作上还有许多共同之处。唯一的不同是字符串太单纯,只存放字符,不像其他容器那样,只要是数据对象都可以存放。此外,str 与 list 和 tuple 在可解析性等方面也极为类似,都属于 Python 的序列容器。

表 5.1　Python 内置内存容器的基本特征

名称	标识符	边　界　符	元 素 类 型	元素可变	元素有序	元素分隔	元素互异
字符串	str	'…'、"…"、'''…'''、"""…"""	字符串	否	位置顺序	无	否
元组	tupie	(…)	任何类型			,	
列表	list	[…]	任何类型	是			
字典	dict	{…}	键有限制,值可为任何对象		否		值否
集合	set		任何类型				是
	forzenset			否			

5.1　Python 内存容器对象的共性操作

5.1.1　内存容器对象的创建与类型转换

内存容器(简称容器)对象都可以用如下 3 种方式构建对象:用字面量直接书写、用构造方法构造和用推导式创建。

1. 用字面量直接书写容器实例对象

不管是元组、列表、字符串，还是字典、集合，只要用相应的边界符将合法的元素括起来，就成为某个容器的字面量，这个字面量是某种类型容器的实例对象。

代码 5.1 用字面量创建容器实例对象的同时，用引用变量指向该对象。

```
>>> a = 1;b = 2; c = 3
>>> #####列表对象创建#####
>>> list1 = [a,b,c,a,3]
>>> list1; type(list1)
    [1, 2, 3, 1, 3]
    <class 'list'>
>>> #####元组对象创建#####
>>> tuple1 = a,b,c,a,3
>>> tuple1; type(tuple1)
    (1, 2, 3, 1,3)
    <class 'tuple'>
>>> #####集合对象创建#####
>>> set1 = {a,b,c,1,2,5}
>>> set1; type(set1)
    {1, 2, 3, 5}
    <class 'set'>
>>> #####字典对象创建#####
>>> dict1 = {'a':a,'b':b,'c':c,'a':5,'d':a}
>>> dict1; type(dict1)
    {'a': 5, 'b': 2, 'c': 3, 'd': 1}
    <class 'dict'>
>>> #####字符串对象创建#####
>>> str1= 'abcde 123abc'
>>> str1; type(str1)
    'abcde123abc'
    <class 'str'>
```

说明：

从上述容器对象创建过程可以看出：

(1) Python 内置容器不一定都要求只存储相同类型的元素。

(2) 列表和元组允许有重复的元素，而集合不能有重复元素，因为集合是按值分配存储空间的。这符合数学中的集合概念。此外，字典不能有重复的键，若有重复的键，则只取最后出现的键-值对。因为在字典中是按键存储值的，遇到先出现的键-值对，就先存储起来，后面再出现相同的键，就用其对应的值覆盖原先的值。

(3) 在用字面量创建容器对象的同时，还可以用变量引用这些容器实例对象。

(4) 一般来说，元组就是一组以逗号分隔的数据对象，不一定要以圆括号为边界符。但从易读的角度，还是加一对圆括号为好。

(5) 上述对象的类型分别是<class 'list'>、<class 'tuple'>、<class 'set'>和<class 'str'>，说明这些对象都是相关类的实例，它们的类名分别为 list、tuple、set 和 str。

2. 用构造方法构造容器对象与容器对象类型转换

每一个类都有自己的构造方法，用于创建这个类的对象(包括空对象)。在面向对象的

程序设计中，把作为类成员的函数称为方法。类的构造方法用于构造该类的对象。在创建非空容器对象时，构造方法要求使用相容对象参数——可以将其转换为所需类型的数据对象，如将字符串参数向列表转换等。对于列表、元组、字符串、集合和字典这些内置的容器，Python 提供了内置的构造方法，分别为 list()、tuple()、str()、set()和 dict()。在前面的应用中已经可以体会到，这些构造函数不仅可以创建数据对象，还可以进行类型转换。但是，列表、元组、集合、字典不能转换为有意义的字符串对象，因为转换时将边界符也作为字符串的一部分了。

代码 5.2 列表、元组、集合、字典不能转换为有意义的字符串对象示例。

```
>>> s3 = str([5,3,1,'b','a','c',1]);s3,id(s3)          #列表转换为字符串
    ("[5, 3, 1, 'b', 'a', 'c', 1]", 1928842938480)
>>> s4 = str((5,3,1,'b','a','c',1));s4,id(s4)          #元组转换为字符串
    ("(5, 3, 1, 'b', 'a', 'c',1)", 1928842938568)
>>> s5 = str({5,3,1,'b','a','c',1});s5,id(s5)          #集合转换为字符串
    ("{5, 3, 1, 'b', 'a', 'c',1}", 1928842938656)
>>> s6 = str({5:'a',7:'c',9:'s',3:'x'});s6,id(s6)      #字典转换为字符串
    ("{5: 'a', 7: 'c', 9: 's', 3: 'x'}", 1928842938744)
```

3. 用推导式创建容器对象

为了动态地修改或创建容器对象，Python 推出了推导式(comprehension)。

代码 5.3 用推导式创建容器对象示例。

```
>>>[i * 2 for i in range(10)]                          #不带条件的列表推导式
   [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>>[i * 2 for i in range(10)  if  i %2 != 0]           #带有条件的列表推导式
   [2, 6, 10, 14, 18]
>>> {i * 2 for i in range(10)}                        #不带条件的集合推导式
   {[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]}
>>>[i * 2 for i in range(10)  if  i % 2 != 0]         #带有条件的集合推导式
   {2, 6, 10, 14, 18}
>>>[x + y for x in 'ab' for y in '123']               #形成字符串列表
   ['a1', 'a2', 'a3', 'b1', 'b2', 'b3']
>>> d5 = dict(zip(('a','b','c'),(1,2,3)));d5,id(d5)  #基于 zip 创建字典对象
   ({'a': 1, 'b': 2, 'c': 3}, 1570113683176)
```

注意： 不要用推导式代替一切。若只需要执行一个循环，就应当尽量使用循环，更符合 Python 提倡的直观性。

4. 其他

字典除用 dict()作为构造方法外，还提供了用 fromkeys()构建字典的手段。fromkeys()是 dict 类的成员，其语法如下：

```
dict.fromkeys(seq[,value])
```

这里的 dict 也可以用{}代替，语法如下：

```
{}.fromkeys(seq[,value])
```

它的参数是两个序列：seq(字典键值序列)和 value(可选参数，设 seq 的值，返回值为一个字典元素序列)。

代码 5.4 用 fromkeys()构建字典对象示例。

```
>>> seq = ('name', 'age', 'sex')
>>> value = ('张', 28, 'male')
>>> dict1 = dict.fromkeys(seq,value)
>>> print (f"创建的字典为{str(dict1):s}。")
    创建的字典为{'name': ('张', 28, 'male'), 'age': ('张', 28, 'male'), 'sex': ('张', 28, 'male')}。
>>> dict2 = dict.fromkeys(seq,'x')
>>> print (f"创建的字典为{str(dict2):s}。")
    创建的字典为{'name': 'x', 'age': 'x', 'sex': 'x'}。
>>> dict3 = dict.fromkeys(seq)
>>> print (f"创建的字典为{str(dict3):s}。")
    创建的字典为{'name': None, 'age': None, 'sex': None}。
>>> dict4 = {}.fromkeys(seq)
>>> print (f"创建的字典为{str(dict4):s}。")
    创建的字典为{'name': None, 'age': None, 'sex': None}。
```

5.1.2 容器对象的通用操作

所有的容器对象都可以进行下列操作。

1. type、ID 码和 len

类型(type)与 ID 码是对象最重要的两个属性，前面已经使用过许多，这里不再赘述。另一个重要属性是容器中元素的个数，它可以使用函数 len()获取。

代码 5.5 获取容器对象的长度。

```
>>> t1 = 'a','b','c','d','e','f',1,2,3,4,5,6
>>> len(t1)
    12
>>> l1 = [t1]
>>> len(l1)
    1
>>> s1 = set(t1)
>>> len(s1)
    12
>>> l2 = list(t1)
>>> len(l2)
    12
>>> l1
    [('a','b','c','d','e','f', 1, 2, 3, 4, 5)]
```

说明：代码 5.5 中，先测试 t1 的长度，得到 12；若用 l1＝[t1]将 t1 转换成列表，测试的结果却是 1；再将 t1 用 s1＝set(t1) 转换为集合 s1，测试结果为 12；再用 l2＝list(t1)测试又得到 12。最后显示 l1，得到 [('a','b','c','d','e','f',1,2,3,4,5)]。这说明 l1＝[t1]是将 t1 作为一个元素了。所以，进行容器类型转换，必须显式地使用构造方法。

2. 获取容器中的最大元素、最小元素与数值元素和

下面 3 个 Python 内置函数可用于获取容器有关数据。

max(s)：返回容器 s 的最大值(仅限字符串或数值序列)。

min(s)：返回容器 s 的最小值(仅限字符串或数值序列)。

sum(s)：返回容器 s 的元素之和(仅限数值序列)。

代码 5.6 获取容器最大元素、最小元素与和示例。

```
>>> t2 = (2,3,4,5,9,2,1)
>>> max(t2),min(t2)
    (9,1)
>>> t3 = {'s','v','ab','wq'}
>>> max(t3),min(t3)
    ('wq','ab')
>>> sum(t3)
    Traceback (most recent call last):
     File "<pyshell#11>", line 1, in <module>
       sum(t3)
    TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> sum(t2)
    26
>>> d1 = {'v':1,'y':8,'g':5,'p':9,'ab':6}
>>> max(d1),min(d1)
    ('y','ab')
```

说明：对于字典，元素的最大值与最小值，从可比较的键中选取。

3. 用 dir()获取对象的其他属性

dir()函数不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。图 5.2 为用 dir()显示几个容器对象属性的示例。

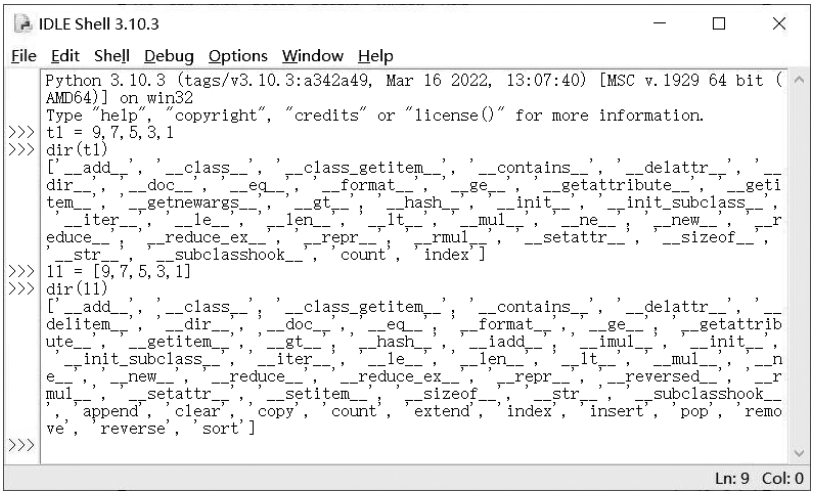

图 5.2 用 dir()获取容器对象属性示例

4. 容器及其成员的判定运算

容器对象的判定运算包括如下 5 类，它们均得到 bool 值：True 或 False。

(1) 对象值比较运算符：>、>=、<、<=、==和!=。

(2) 对象身份判定运算符：is 和 is not。

(3) 成员属于判定运算符：in 和 not in。

(4) 布尔运算符：not、and 和 or。

(5) 判定容器对象的元素是否全部或部分为 True 的内置函数：all()和 any()。

代码 5.7 对序列进行判定运算示例。

```
>>> list1 = ['ABCDE','Hello',"ok",'''Python''',123]; list2 = ['xyz',567]
>>> list1 == list2, list1 != list2, list1 > list2, list1 < list2
    (False, True, False, True)
>>> 'ABCDE' in list1
    True
>>> ['xyz',567] is list2,list2 is ['xyz',567],list2 == ['xyz',567]
    (False, False, True)
>>>
>>> tup1 = (1,2,3); tup2 = (1,2,3); tup3 = ('a','b','c')
>>> tup1 == tup2,tup1 is tup2
    (True, False)
>>> tup1 = tup2; tup1 is tup2
    True
>>> tup3 < tup2,tup3 > tup2
    Traceback (most recent call last):
     File "<pyshell#65>", line 1, in <module>
       tup3 < tup2,tup3 > tup2
    TypeError: '<' not supported between instances of 'str' and 'int'
>>> tup3 != tup2
    True
>>>
>>> str1 = 'abcxy';str2 = 'abcdef'
>>> str1 < str2, str1 > str2
    (False, True)
>>>
>>> set1 = {1,2,3};set2 = {1,2,3,4,5}
>>> set1 > set2,set1 < set2,set1 == set2,set1 != set2
    (False, True, False, True)
>>>
>>> all(tup3),any(tup3)
    (True, True)
```

说明：

(1) 相等比较(==)与是否比较(is)不同，相等比较的是值，是否比较的是 ID 码。

(2) 只有相同元素类型的容器对象才可以进行大小比较。不同元素类型的容器对象只可以进行相等或不等的比较。

(3) 字符串之间的比较是按正向下标，从 0 开始以对应字符的码值(如 ASCII 码值)作为依据进行的，直到对应字符不同，或所有字符都相同，才能决定大小或是否相等。

5.1.3 对象的浅复制与深复制

在 Python 中,“=”的本职操作是为对象添加引用,只有对修改后的不可变对象进行引用时,才会形成新的对象;要复制数据对象,应当通过有关类或模块中的复制函数进行。这些复制函数可以分为两类:浅复制(shallow copy)和深复制(deep copy)。

- 可以进行浅复制的函数或方法包括:copy 模块中的 copy()函数、序列的切片操作、对象的实例化等。
- 可以进行深复制的函数或方法包括:copy 模块中的 deepcopy()函数等。

下面分两种情形举例说明浅复制和深复制的区别。

1. 浅复制和深复制仅对可变数据对象有区别

代码 5.8 对于可变数据对象和不可变数据对象来说,浅复制与深复制的作用区别示例。

```
>>> import copy                                    #导入 copy 模块
>>> x = (1,2,3,('a','b'))                         #x 为不可变对象
>>> y = copy.copy(x)                              #y 为 x 的浅复制
>>> z = copy.deepcopy(x)                          #z 为 x 的深复制
>>> id(x),id(y),id(z)                             #比较 x、y、z 三者的 ID 码
    (2319542617016, 2319542617016, 2319542617016)
>>> x1 = [1,2,3,['a','b']]                        #x1 为可变对象
>>> y1 = copy.copy(x1)                            #y1 为 x1 的浅复制
>>> z1 = copy.deepcopy(x1)                        #z1 为 x1 的深复制
>>> id(x1),id(y1),id(z1)                          #比较 x1、y1、z1 三者的 ID 码
    (2319502470600, 2319542524936, 2319542496392)
```

说明:

(1) id(y)和 id(z)都与 id(x)相同,说明对于不可变对象,浅复制与深复制都不重新创建新的对象。

(2) id(x1)、id(y1) 与 id(z1) 三者都不相同,说明对于可变对象,浅复制与深复制都重新创建了新的对象,但三者所创建的新对象是不相同的对象。

2. 对层次性对象(如嵌套的对象容器以及派生类对象)进行浅复制,只复制最上一层

代码 5.9 对于嵌套容器中的可变数据元素来说,浅复制与深复制复制深度的不同示例。

```
>>> import copy
>>> x = [1,2,3,['a','b','c'],4]
>>> y = copy.copy(x)
>>> z = copy.deepcopy(x)
>>> id(x[2]),id(y[2]),id(z[2])                    #获取同一整数对象的 ID 码
    (1584641152, 1584641152, 1584641152)
>>> id(x[3][0]),id(y[3][0]),id(z[3][0])           #获取同一字符串的 ID 码
    (1907772550704, 1907772550704, 1907772550704)
>>> id(x[3]),id(y[3]),id(z[3])                    #获取同一可变对象成员的 ID 码
    (1907782020168, 1907782020168, 1907782020040)
```

说明：

(1) 在容器嵌套的情况下，对于可变元素来说，浅复制与深复制的情形就不相同了。尽管浅复制的数据对象的 ID 码与源对象的 ID 码不同，但其所有元素的 ID 码都与源对象对应相同，即浅复制虽然会创建新对象，但其内容是原对象的引用，即它只复制了一层外壳。而对于深复制来说，其内嵌的可变元素对象的 ID 码与源对象的对应元素的 ID 码不再相同，即这些内嵌元素也被复制了。所以深复制是完全复制，包括了多层嵌套复制，复制的深度大于浅复制。

(2) 对于在第 4 章介绍的派生类对象复制，也会产生类似的情况。

习题 5.1

一、判断题

1. 元组与列表的不同仅在于一个是用圆括号作为边界符，另一个是用方括号作为边界符。 (　　)

2. 创建只包含一个元素的元组时，必须在元素后面加一个逗号，例如(3,)。 (　　)

3. 列表是可变的，即使它作为元组的元素，也可以修改。 (　　)

4. 表达式 list('[1,2,3]') 的值是 [1,2,3]。 (　　)

5. 表达式 []==None 的值为 True。 (　　)

6. 生成器推导式比列表推导式具有更高的效率。 (　　)

7. 代码

```
>>> aList = []
>>> for x in range(30):aList.append(x + x)
```

与代码

```
>>> aList = [x + x for x in range(30)]
```

等价。 (　　)

二、选择题

1. 在后面的可选项中选择下列 Python 语句的执行结果。

print(type ({}))的执行结果是(　　)。

print(type([]))的执行结果是(　　)。

print(type(()))的执行结果是(　　)。

A. <class 'tuple'>　　B. <class 'dict'>

C. <class 'set'>　　D. <class 'list'>

2. 推导式 [4(x,y) for x in [1,2,3] for y in [3,1,4] if x !=y] 的执行结果是(　　)。

A. [(1,3),(2,1),(3,4)]

B. [(1,3),(1,4),(2,3),(2,1),(2,4),(3,1),(3,4)]

C. [1,2,3,3,1,4]

D. [(1,3),(1,1),(1,4),(2,3),(2,1),(2,4),(3,3),(3,1),(3,4)]

3. 代码

```
>>> vec = [(1,2,3),(4,5,6),(7,8,9)]
>>> [num for e in vec for num in e}
```

执行的结果是(　　)。

A. [1,2,3,4,5,6,7,8,9]　　B. [[1,2,3],[4,5,6],[7,8,9]]

C. [[1,4,7],[2,5,8],[3,6,9]]　　D. (1,2,3,4,5,6,7,8,9)

4.下面不能创建一个集合的语句是(　　)。

A. s1＝set ()　　B. s2＝set("abcd")

C. s3＝(1,2,3,4)　　D. s4＝frozenset((3,2,1))

5. 下列代码执行时会报错的是(　　)。

A. v1＝{}　　B. v2＝{3:5}

C. v3＝{[1,2,3]:5}　　D. v4＝{(1,2,3) :5}

6. 以下不能创建一个字典的语句是(　　)。

A. dict1＝{}　　B. dict2＝{3:5}

C. dict3＝dict([2,5],[3,4])　　D. dict4＝dict(([1,2],[3,4]))

7. 代码

```
nums=set{1,1,2,3,3,3,4};print(len(nums)
```

执行后的输出结果为(　　)。

A. 2　　B. 4　　C. 2　　D. 7

8. 代码

```
a={1:/a/,2:/b/,3:/c/};print(len(a))
```

执行后的输出结果为(　　)。

A. 1　　B. 43　　C. 0　　D. 6

9. 代码

```
a=[1,2,3,None,(),[],];print(len(a))
```

执行后的输出结果为(　　)。

A. 1　　B. 43　　C. 0　　D. 6

三、填空题

1. Python 语句 list1＝[1,2,3,4];list2＝[5,6,7];print(len(list1＋list2)的执行结果是_____。

2. Python 语句 print(tuple([1,3,]),list([1,3,])) 的执行结果是_____。

3. Python 语句 print(tuple(range2),list(range2)) 的执行结果是_____。

4. Python 列表生成式 [i for i in range7 if i % 2!＝ 0] 和 [i ** 2 for i in range5] 的值分别为_____。

5. Python 代码 print(set([3,5,3,5,8])) 的执行结果是_____。

6. 使用列表推导式生成包含 10 个数字 5 的列表，语句可以写为_____。

7. Python 代码 score={'language':80,'math':90,'physics':88,'chemistry':82}; score['physics']=96; print(sum(score.value()/len(score))) 的执行结果是_____。

8. Python 代码 d={1:'a',2:'b',3:'c',4:'d'};del d[1]; del d[3];d[1]='A';print(len(d))的执行结果是_____。

9. Python 代码 score={'language':80,'math':90,'physics':88,'chemistry':82}; score['physics']=96; print(sum(score.value()/len(score)))的执行结果是_____。

10. Python 代码 print(sum(range(10)的执行结果是_____。

11. Python 代码 s1=[1,2,3,4]; s2=[5,6,7]print(sum(range(10)的执行结果是_____。

12. 在 Python 代码中，first, * middles,last=range6 执行后，middles 的值为_____; first,second,third, * lasts=range6 执行后，lasts 的值为_____; * firsts,last3,last2.Last=range6 执行后，firsts 的值为_____; * middles,last=[88,85,99,95,66,77,96] 执行后，sum(middles)//len(middles)/的值为_____。

四、代码分析题

1. 阅读下面的代码片段，给出各行的输出。

```
list = [[]] * 5;  list          #output?
```

2. 执行下面的代码，会出现什么情况？

```
a = []
for i in range(10):
      a[i] = i * i
```

3. 分析下面的代码，给出输出结果。

```
def multipliters():
    return [lambda x:i * x for i in range(4)]
print([m2 for m in multipliters()])
```

4. 分析下面的代码，给出输出结果。

```
L = ['Hello','World','IBM','Apple']
print([s.lower() for s in L])
```

5. 指出下面代码的输出是什么，并解释理由。

```
def multipliers():
    return [lambda x : i * x for i in range(4)]
print [m2 for m in multipliers()]
```

怎么修改 multipliers 的定义才能达到期望的结果？

五、实践题

1. 编写代码，实现下列变换。