**第7章 字典和集合**

在前面所学的章节中，我们认识了 Python中带有序列化索引的数据类型，如列表。除此之外， Python还提供了两个没有顺序的内建数据类型，分别是字典和集合。字典作为Pythoii语言中唯 一的映射类型，定义了键和值之间一对一的关系，集合则是一个无序不重复元素的集。接下来， 本章将为大家介绍字典和集合这两种数据类型。

*7.1*字典类型

如果希望在拥有上万条数据的列表中找到某条数据，唯一的解决办法就是查找整个列表， 这种做法不仅工序烦琐歹而且效率低下。那么，有没有一种能快速精准地查找值的结构呢？答 案是有。因为使用字典存储的数据，可以快速定位到指定的值，字典是一种通过名字引用值的 数据类型，本节将带领大家学习字典的内容。

7JJ 寥薩爾嘉零纏蔻， .. .

提到字典这个词，大家都不陌生。在学生时代凡是碰到不认识的字，就会使用字典的部首 表找到汉字的说明。程序对现实中的字典进行了构造，可以根据某个关键字（键）为索引，快 速方便地找到其对应的值。

这里以电话簿为例，通过比较列表和字典存放数据方式的不同，引出字典这种数据结构的 优势。有如下一个电话簿，电话簿的信息包含联系人的姓名和手机号码，其中姓名和手机号码 必须是一一对应的，这样才能保证准确地呼入和呼出。

若用列表这种数据结构来保存联系人信息，需要先把所有联系人的姓名放入一个列表中， 为了让联系人的姓名和手机号码进行匹配，需要再准备一个列表在相同的索引位置放置联系人 的手机号码。这时，通过相同的索引获得的联系人的姓名和手机号码是匹配的，如图7』所示。

在图7」中，通过相同的索引间接地让两个列表之间产生了某种联系。虽然这样能达到我 们的目的，但是并不切合实际。一旦列表中数据的顺序发生了变化，就将无法满足这种一对一 的关系。

若用字典这种数据结构来保存联系人的信息，可以让联系人的姓名作为字典的键，联系人 的手机号码作为字典的值，通过字典中的键可以绑定这个键所对应的值，使得字典中存放的姓 名和手机号码是一一对应的关系。这时，若要获取某个联系人的手机号码，可以将姓名作为键 快速实现，如图7-2所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 |  | 索引 |  | 手机号码 |  | 姓名 |  | 手机号码 |
| Jack |  | 0 |  | 10086 |  | Jack |  | 10086 |
|  |  |  |
| Rose | 1 | 10087 | Rose | 10087 |
|  |  |  |
| Lily | 2 | 10088 | Lily | 10088 |
|  |  |  |
| Jay | 3 | 10089 | Jay | 10089 |
|  |  |  |
| Jane | 4 | 10090 | Jane | 10090 |
|  |  |  |

图7』两个列表存放联系人信息 图7-2字典存放联系人信息

乱1卷享鹽矗廁寰齣睿器

Python中的字典是可变类型的，它支持对字典中的元素进行添加、删除、更改等操作。字 典同时也是一个容器类型，能够存储任意个对象，包括其他容器类型的对象，如列表。与列表 这个容器类型相比，字典的不同体现在以下几个方面：

®列表中存放的数据是有序的，而字典中存放的数据是无序的。

•列表使用数字类型的索引存放元素，而字典一般使用字符串类型的键来关联值，还可以 使用其他类型的对象作为键。

•列表中的元素是有顺序的，可通过序列化的索引进行访问；而字典中的元素是键值对， 键与值是有关联的，可通过字典中的键访问其对应的值。

字典中的键必须遵守以下两个原则：

（1） 每个键只能对应一个值，不允许同一个键在字典中重复出现。

在创建字典时，如果同一个键被多次赋值，则最后一次赋的值为字典该键对应的值。示例 如下。

>» demo\_dict = （ 'water' :'透明' r f water \* :'蓝色，} #创建die字典,两个键是相同的 >>> demo\_dict

{ ? water 1 : v蓝色\* } #字典die的water键对应的值为蓝色

（2） 字典的键是不可变类型（可哈希的对象）。

通常情况下，字典的键是字符串或者数值类型的，像列表这样可变的对象不能被用做字典 的键，示例如下。

#创建字典，键为列表类型

>>> demo\_dict = ( [1] :\* abc \*}

Traceback (most recent call last):

File °<stdin>n r line 1f in <module>

TypeError: unhashable type : \* list \* #报错,提示列表是非可哈希的类型

如果字典中的键是数字类型，就需要遵循数字比较的原则：若两个数字的数值结果相等（如

1和L0）*，*表明字典中出现相同的键，此时，最后给这个键赋的值才是其字典的值。示例如下：

>>> demo\_dict = #创建字典,键为数值类型

>>> demo\_dict

{1: !b!}\_ #互换为相同的索引，取最后的赋值

需要注意的是，如果元组中只包含字符串和数字，该元组可以作为字典的键；如果元组直 接或者间接地包含了可变的对象，就不能当作字典的键。

字典中的值没有任何限制，它可以是任意的Python对象，包括标准对象和自定义对象。 （泠多学一招：为什么键必须是可哈希的？

哈希表是一种数据结构，通过把关键码值映射到表中的一个位置访问记录，以加快查找速 度。哈希表中存储的每一条数据叫作一个值，是根据与它相关的一个叫作键的数据项进行存储 的，键和值组合在一起称为键-值对。

哈希表的算法是获取键，对键执行一个叫作哈希函数的操作，然后根据计算的结果，选择 在数据结构的某个地址中存储值，任何一个值所存储的地址都取决于其对应的键，所以哈希表 是没有顺序的。

当解释器调用哈希函数时，如果字典的键是可变类型的对象，它随时会发生变化。一旦键 发生了变化，哈希函数会映射到不同的地址存储数据，这样就无法保证准确地存储或查找相关 的数据。可哈希对象的值是不能改变的，这就是键必须是可哈希的原因。

2 7o2宇興的基本躁作

7.2J劍篷掌麒舞夠赢素殲値 .

Pythoii中提供了快速创建字典的方式，即使用一对花括号包含多个键值对，其中键和值之 间以冒号隔开，每个键值对之间使用逗号进行分隔。其语法格式如下：

{键 M 值 L 键2 :值2。

如果没有在花括号中指定键值对，则会创建一个空的字典。示例如下:

»> demo\_dict = { Fame ? :，Jack 七？ num，： 20} # 创建一个字典

>>> demo\_dict

{?num!: 20# \* name 1: 5 Jack1}

»> demo\_dict = { } #创建一个空的字典

>>> demo\_dict

{}

如果要在上述创建的空字典中添加元素，可以使用如下语法实现： .

diet ［键］=值

示例如下:

>>> demo\_dict [ \* color , ,紫色?

>>> demo\_d.ict

{ s color, : ?紫色 5 )

Pythoii中的字典是diet类型的。从Python 22开始，可以使用diet类型提供的函数来构建 字典，主要有如下三个函数：

diet(\*\*kwarg)

diet(mapping, \*\*kwarg)

diet(iterable, \* \*kwarg)

这些函数所表示的含义如下：

•如果函数中没有传入参数，则会构建一个空的字典。

-如果给函数传入参数身并且这个参数是另外一个映射对象(如字典)，则会使用与该映 射对象相同的元素创建新的字典。

•如果提供给函数的参数为可迭代对象(如列表),那么每个可迭代对象必须是成对出现的。 其中，第一个对象会成为新字典中的键，第二个对象会成为其对应的值。注意，如果某 个键在字典中多次出现，则给这个键最后一次赋的值才是其对应的值。

•如果提供了关键字参数，则会把关键字参数和其对应的值添加到从位置参数创建的字典 中。如果要添加的键已经存在，则来自关键字参数的值会被替换为从位置参数中获取到 的值。

接下来，使用上述这些函数创建字典对象，示例如下：

»> diet () #创建一个空字典

{}

»> diet ( [ ( \* jay\* , 123) z J json\*, 1234) , ( s jane\ 12345)])

{\* jane1 : 12345, !jay? : 123, \*json\* : 1234}

»> dict([(saM), (\*aS2)])

{ : 2}

>>> diet(sape=4139, guido=4127, jack=4098)

{'sape': 4139, 1jack1: 40 98, 1guido': 4127)

除了上述创建字典的方式以外，还可以通过字典推导式从任意的键-值表达式中创建字典, 示例如下:

>>> {x: x\*\*2 for x in (2f 4’ 6) }

(2: 4, 4: 16, 6: 36}

/泠注意：字典中的键是唯一的，而值并不是唯一的，一个键不允许对应多个值。

*'腿 5 / '*

7.2.2觸岡寥輿剧鲍氟繁

要想得到字典中某个元素的值，可以通过udict[key],，的形式获取。示例如下;

>» demo diet = { 1 color5 :'紫色'}

>>> demo\_dict[, color f ]

，紫色?

如果访问字典的键是不存在的，程序会引发一个KeyError异常。示例如下：

>>> demo\_dict = {\* name \*:!Jay \* 7 1 age 5:35)

>>> demo\_dict[\* gender \* ]

Traceback (most recent call last):

File "<stdin>n, line 1f in <module>

KeyError: 5 gender 1

为了避免引起上述异常，我们在访问字典元素的时候，首先需要检查某个键是否存在于字 典中，只有存在的条件下才能够访问。为此，Pythg提供了两种解决方案，一种是用has\_kcy() 方法检查，不过这个方法在Python 3以后已经被废除了，另一种就是使用in或not in操作符检查。

Python 2.2中推出的in或not in操作符，可以用来检测某个键是否存在于字典中。如果字 典中有这个键,就返回True,否则返回False,示例如下:

»> demo\_dict = { J name \* : 5 Jay5 7 ! age 5 : 35)

>>> if 1 gender \* in demo\_\_dict:

..。 print(demo\_dict[, gender ?])

里..㊀Ise:

…. print ( \* key 不存在 \* )

key不存在

7.2.3 遢瘤寥麗顚齣施繁 . ，.

通过f(M循环可以遍历字典中的键，使用键再访问其对应的值，示例如下:

>>> demo\_dict = {1 name ? : 'moon 卩 7 v color \* : 5 yellow!}

>>> for key in demo\_dict <, keys ():

。… print ( s key=%s f value=%s 5 % (key, demo\_dict [key]))

key=colorf value=yellow

key=name7 value=moon

在上述示例中，通过调用keys。方法得到了由字典中所有键组成的列表，使用for循环遍 历列表，一一取出了字典中的键和其对应的值。

从Python 22开始，可以用迭代器轻松地访问序列对象，比如列表和字典等。只要使用血 循环直接遍历字典，就可以取出字典中的元素，示例如下：

>>> demo\_dict = {s name \*: s moon 5 7 J color ?:5 yellow ,}

»> for key in demo\_dict:

…。 print ( \* key=%s7 value=%s ? % (key^ demo\_dict [key]))

key=color7 value=yellow

key=name7 value=moon

7d2d4露飜寡製申餉漏饕 ..一 .」

字典是可变的对象，支持对元素进行更新操作，包括往字典中添加新的元素、更新已经存 在的元素，以及删除已经存在的元素。在往字典中添加新的元素时，如果要添加的键已经存在, 那么字典中该键对应的值会被新值替代。

示例如下:

>>> demo\_dict = { ? strul9 : ?小明 \* r ' stru2，：，小兰'}

>>> demo\_dict

{ \* strul5 :，小明 L , stru2 J :。小兰？}

>>> demo\_dict [ ' stru2 1 ] = ?小白星

>>> demo\_\_dict

{ ! strul' : \* 小明 f r ! stru2 f :，小白 r }

7.2.5 H隴寥製漏矗浏寥颱 「 「 、

对字典执行删除操作，可以使用 如 命令实现，如命令既能删除字典的元素，又能删除整 个字典。不过，删除整个字典的操作不常用，大家请慎重使用。

使用del命令删除字典元素及字典本身的示例如下:

>>> demo\_dict = ( ' namel,: ' 小明 1 f ' name2 ': “ 小红？} >>> demo\_dict

{ darnel1 :，小明七'name2 1 :，小红？}

>>> del demo\_dict [\* namel?]

>>> demo\_dict { \* name2 J :，小红早}

>>> demo\_dict die >>> demo\_dict

Traceback (most recent

File "<stdin>n r line NameError: name \* demo

#删除字典某个元素

#删除整个字典

call last):

1, in <module>

diet ' is not defined

上述示例中，创建了一个包含两个元素的字典。首先，使用del命令删除字典中指定的元素, 删除成功。接着又使用del命令删除字典，成功删除字典后，字典所在的内存空间会被释放,此时, 如果再访问字典demo\_dict9会报错，提示字典demo dict不存在。

如果想要保留字典占用的内存空间，可以使用皿迎()方法清除所有的元素，得到一个空的 字典。

示例如下:

>>> demo diet = { Famel，："小明 \* \* name2 , : 1 小红？} >>> demo\_dict

{ ' namel \* : v 小明 L ' name2 ?:星小红，} ■

>>> demo\_dict.clear()

>>> demo\_dict

{}

源?注意：如果要删除的键不存在，则会抛出KeyErroT异常。

! To 3字典常网的函教

本节将为大家介绍Python为字典提供一些的常用函数，这些函数包括用于获取元素总个 数的len()函数，比较元素大小的cmp()函数，以及获取哈希值的hash()函数。

7d3J !l@o圖數發麒寥翼売素爾灘量

len()函数用于返回字典中所有元素的数目，示例如下：

»> demo\_dict = {1: ' Jack5 z 2 : \* Rose ' }

>>> demo\_dict

*{1:* \* JackJ, 2: ? Rose 5)

>>> len(demo\_dict)

2

/乳紀 » hash

h颠h()函数用于返回对象的哈希值(用一个整数表示)，通常用在字典中，用于判断某个 对象是否可以作为字典的键。

示例如下:

»> hash ( [ ] ) #返回空列表的哈希值

Traceback (most recent call last):

File ,,<stdin>°, line 1 e in <module>

TypeError: unhashable type: ?list \*

»> hash(l) #返回整数1的哈希值

1

从示例的结果可以看出，列表不是可哈希的，为此不能作为字典中的键。整数是可哈希的, 为此可以作为字典中的键。

如果两个数的数值相等，即使所属的类型不同，计算出来的哈希值也是一样的，示例如下：

>>> hash (1)

1

>>> hash (1.0)

1

由此说明，数字类型的哈希值是根据数值计算的，不需要区分数字所属的具体类型。

*1* 7o4宇興的內建方職

Python为字典类型提供了一些非常实用的内建方法，这些方法虽然没有像字符串的内建方 法那样使用频率那么高，但是当程序员在工作中碰到跟字典有关的问题时，可以使用这些方法 来简化开发的步骤。表7』列出了字典常用的内建方法。’

表7“字典的内建方法

|  |  |
| --- | --- |
| 方法名 | 相关描述 |
| clear() | 删除字典中的所有元素 |
| copy() | 返回字典的一个浅复制 |
| fromkeys(seq[, value]) | 创建一个新的字典，设置seq中的元素为字典的键，设置value为字典中所有键对应的初始值。如果 不提供此值，则默认为None |
| get(key[5 default]) | 通过字典中的键,获取其对应的值。如果字典中不存在此键,则返回default的值。注意,default的默 认值为None |
| items() | 返回字典的元素视图 |
| keys() | 返回字典的键视图 |
| pop(key[5 default]) | 如果字典中的键存在,删除并返回其对应的值;如果字典中的键不存在，且没有给出default的值， 则会引发KeyError#常 |
| popitemQ | 随机删除字典中的一个键值对,并返回这个键值对 |
| setdefault(key[5 default]) | 如果字典中不存在key,则会使用dict[key]- default为它赋值 |
| update([other]) | 把odie冲的键值对添加到字典中 |
| valuesQ | 返回字典的值视图 |

7n4J copj

copy()方法用于返回一个有着相同键值对的字典。这个方法实现的是浅复制，即调用 copy()方法会复制出一个新的字典对象，但是这个新字典引用的元素还是原字典对象的。

copy()方法的语法格式如下:

diet.copy()

示例如下:

|  |  |  |  |
| --- | --- | --- | --- |
| >>> dict\_01 = | {1：' | spring 1f 2: 8 | 'summer f f 3 : ? autumn 1 f 4 : J winter 1 } # 创建字典 |
| >>> dict\_01 |  |  |  |
| {1: 1 spring1, | 2: ? | summer、3: | :sautumn ?, 4: 1 winter?) |
| >>> dict\_02 = | diet | \_01。copy() | #通过浅复制创建字典的副本 |
| >>> dict\_02 |  |  |  |
| {1: 5 spring', | 2:' | summer 1*；* 3: | :\* autumn!, 4: , winter \*} |
| »> del dict\_ | 02 [3] |  | #删除字典副本的指定元素 |
| »> del dict\_ | 02 [1] |  | #再次删除字典副本的指定元素 |
| »> dict\_02 |  |  | #使用浅复制后的字典 |
| {2: 1 summer ® z | 4 :' | winter 1} |  |
| >>> dict\_01 |  |  | #原来的字典 |
| {1: !spring'f | 2: \* | summer'z 3 : | :'autumn 1, 4: \* winter \*} |

在上述示例中，字典dict\_02是字典dict\_Ol浅复制过来的。当字典副本dict\_02连续删除 元素以后，可以发现原始字典dict\_Ol没有发生任何变化。

上述示例的字典中，所有的值都是不可变类型的。如果字典中元素的值为列表类型，一旦 对这个列表进行更改，通过调用spy()方法得到的新字典会如何变化呢？接下来，通过一个示

例代码进行演示，具体如下:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| »> | dict\_ | 01 = (1 phone ?:[ | \*123\ ? 1234 ? , 12345'] } | # | 快速创建一个字典 |
| >» | dict\_ | 02 = dict\_01.copy() | | # | 使用浅复制建立字典副本 |
| »> | dict\_ | 02[Jphone \*]。remove(f12345 f) | |  |  |
| »> | dict\_ | \_02 |  |  |  |
| (\* phone ?: | | *[,123, f* f1234' | ]} | # | 新字典 |
| »> | dict\_ | \_01 |  |  |  |
| ('phone': | | [?123\ '1234' | ]} 、 | # | 原字典 |

在上述示例中，字典dict\_02是字典dict\_01浅复制过来的。其中，字典dict\_01的值为一 个列表。当字典副本dict\_02删除了列表中的某个元素以后，可以看到原始字典dict\_Ol和字典 副本dict\_02发生了相同的变化。由此表明，通过浅复制得到的字典副本，和原始字典引用的 是同一份数据。

为了避免因字典副本更改而导致原始字典发生变化的问题，这里可以使用深复制进行替换。 深复制会复制字典中所有的元素，而不是只复制原始元素的引用。要想使用深复制建立字典的 副本，需要通过copy模块的deepcopyQ函数实现。

对上述示例进行修改，使用deepcopy()函数建立字典的副本，具体如下：

>>> from copy import deepcopy # 从 copy 模块导入函数

»> dict\_01 = { 5 phone f : [J10086\,1008 611,]}

>>> dict\_02 = deepcopy (dict\_01) #使用深复制建立字典副本

>>> dict\_02[\* phone']。remove(v1008611f)

>>> dict\_01

( v phone 9 : [ ?10086 \ nOOSGll5]}

»> dict\_02 {?phone': [510086 f] }

在上述示例中，字典dict\_01通过深复制建立了字典副本dict\_02o在对字典副本dict\_02的 列表进行删除操作以后，原始字典dict\_\_01没有受到任何影响。

通过字典的键访问值时，若字典中的键不存在金程序肯定会出现KeyErr质错误。示例如下：

»> demo\_dict = { } # 创建空字典

»> d.emo\_dict [ f name ' ] #试图访问不存在的键

Traceback (most recent call last):

File "<stdin>"7 line 1z in <module> KeyError: 1 name 1

在上述示例中,创建了一个空的字典demo\_dict9访问name键对应值时出现了 KeyError 错误。为了避免出现上述这个问题，Python提供了对访问字典要求比较宽松的郭()方法，该 方法用于返回指定键对应的值须如果访问的键不存在，则会返回默认值。

get()方法的语法格式如下:

get(key[z default])

其中，key为字典中要查找的键，def郷t为键不存在时返回的默认值。如果没有设定默认值, 则返回None,否则返回自己设定的值。

示例如下:

>>> demo\_dict = {}

»> print (demo\_dict. get (' name \*) ) # get方法访问字典,没有设定默仄值

None

»> print (demo\_dict. get (' name?,不存在')) #使用get方法访问字典,设定了默认值

不存在

当字典中要找的键存在时，使用g如)方法与普通访问字典元素的操作没有任何区别，示 例如下:

>>> demo\_dict = { 1 name \* : 丫 小明 ' }

>>> print(demo\_dict.get(\* name ?)) 小明

7q4d3 Kerns BI®

在Python 3中，他ms()方法返回字典的元素视图。元素视图会一直跟字典保持同步，当字 典发生变化时，可实时地反映出字典中元素的变化。调用item()方法的示例如下：

>>> demo\_dicts.items()

demo\_dict\_items ( [ (f gender!z f male!f (!height 5 z 18 5)^ (\* name , r

'Jack')])

元素视图支持迭代操作，它可以使用f。『循环遍历取出(键、值)元组，具体示例如下：

>>> for item in demo\_dict.items():

... print(item)

(¥ gender 5 f smale f)

(height' f 185) (\* name 1, 5 Jack')

实际上,Python 3.x中的items()方法跟Python 2.x版本中的viewitemsQ方法的功能一样, 而Python 2.x的items()方法用于以列表的形式返回字典的所有元素(键■值对)。示例如下:

>>> demo\_dict = {\* name J: ? Jack \* f 1 gender \*:? male f 7 ? height1:185}

>>> demo\_dict.items()

[(\* gender v f fmale ?f (\* name J 7 !Jack!f ? height \* 7 185)]

可能会有人感到疑惑，为什么Python 3.x要进行升级呢？主要是因为，如果使用列表直接 存放这些临时数据，一旦字典的元素过多，则很有可能会占用大量额外的内存，而元素视图是 一个对象，避免了浪费过多的内存

，*7AA* keys 责澧 、 一

在Python 3中，keys()方法会返回字典的键视图，该视图会一直跟字典保持同步，当字典 发生变化时，可实时地反映岀字典中键的变化。调用K©y()方法的示例如下：

>>> demo\_dict。keys( demo\_dict\_keys([\* onion 1z \*potato \* r 1 apple 9])

同样键视图可以直接使用f«循环遍历取出每个键，具体代码如下：

»> for item in demo\_dict. keys ():

..。 print (item)

onion

potato apple

跟前面所介绍的items()方法类似,Python 3.x中的keys()方法取代了 Python 2.x中的 viewkeysQ方法,而Python 2.x的keys()方法会直接以列表的形式返回字典中所有的键。

>>> demo\_dict = {\* potato 5:\* $101z 5 onion1:8 $10 \ 5 apple 1:,$20'}

»> demo\_dict. keys ()

[\* apple 5 f v onion \* 7 ?potato 1]

乱盛忐wahes荷漑

在Python 3中，values()方法返回字典的值视图，该视图会一直跟字典保持同步，当字典 发生变化时，实时地反应出字典中值的变化。调用values()方法的示例如下。

>>> demo\_dict.values() demo\_dict \_values ([v$10\ ?$10\ »$20?])

使用f成循环遍历值视图，取出每个值，具体代码如下。

»> for item in demo\_dict. values ():

•.. print(item)

$10

$10

$20

同样,Python 3.x中的values()方法取代了以前版本的viewvalues()方法,而Python 2.x的 valuesO方法会以列表的形式返回字典中所有的值，示例如下。

>>> demo\_dict = {?potato \*: 1 $10 1 7 1 onion \*: 5 $10 , z \* apple ?: \* $20 ' } »> demo\_dict. values ()

[!$20S ?$10\ s$10?]

! 集念类型

我们把set称为由不同元素组成的集合。在数学上，集合的成员通常被称为集合的元素. Pythcrn把set集合的概念引入到程序的集合类型中。接下来，本节为大家介绍集合这个数据类型。

在介绍集合的概念前，先来解决一个数据存放的问题。有如下一个容器，这个容器可以存 放无数个对象，不过容器中的对象是不能重复的。若使用列表这个数据类型存放对象，为了排

除列表中会出现相同的元素，需要遍历整个列表, 就从列表中剔除这个元素。

使用列表存放不重复元素的示例如下:

对列表元素逐个进行比较,出现相同的元素

»>

»>

丄

»>

»>

»>

li = 8]

print(li)

1, *5,*

*i = 0*

*j* = 1

while j

while j <len(li): if li [i] == li [ j]: del li [ j] continue

#创建一个列表

i < len(li):

=i+1

#使用while嵌套循环遍历列表进行比较

#相等就删除该元素

j+=l

i+-l

>>> print(array) 丄5偿8]

可是，如果列表中存放的数据量很大，势必会浪费更多的时间去排除重复元素，降低程序 执行的效率。

为了解决上述这种问题，Python提供了集合这个数据类型。Python的集合是一个存储不重 复可哈希对象的无序集，可以使用奖t()函数创建。把上述示例中列表的数据挪到集合中过滤重 复的元素，示例如下M

>>> array = 丄1,5,5,8,8]

>>> array

[1, 1, 5, 5, 8, 8]

>>> set(array) set ( [8, 1, 5])

从程序输出的结果可以看出，院t集合成功地过滤了列表中重复的元素。除了消除重复元 素以外,集合还支持联合(union )、相交(intersection)、差补(diffeixnce )和对称差分(symmetric difference)的数学运算。

由于集合是无序的，不具备记录元素位置或插入顺序的能力，所以集合不支持索引、切片 或者其他类似于序列的操作。

松&2靈念齣襲製

目前，Pythoii有两种内置的集合类型:set (可变集合)和frozenset (不可变集合)，它们 的区别如下：

(1 ) set集合

可变的集合类型，能够对集合内的元素进行更改。由于對t集合是可变的，所以它没有哈 希值，不能用作字典的键或作为另一个集合的元素。

(2 ) frozenset 集合

不可变的集合类型，集合的内容在创建以后不能被更改，所以它可以用作字典的键，或者 作为另一个集合的元素。

冗6集念的基本攤作

7O®J曩念齣劍麗濾曬儷

Python中提供了 set()函数用于创建可变集合的对象，frozensetQ函数用于创建不可变集合 的对象。这两个函数的语法格式如下:

set ( [iterable])

frozenset([iterable])

上述两个函数返回的是新的心或frozenset对象，它们的元素均取自可迭代的对象 (iterable) ?如列表等。若没有指定可迭代的对象,则会返回一个空的集合。

使用上述两个函数创建集合的示例如下：

>>> set\_01 = set () # 创建一个 set 集合

»> set\_\_02 = frozenset ( [ 1,2 z 3 4 A 5 ] ) # 使用可迭代对象,创建一个 frozenset 集合

>>> set\_01

set ()

>>> set\_02

frozenset({17 2f 3r 4r 5})

>>> set\_03 = set (1)

Traceback (most recent call last):

File °<stdin>° f line 1f "in <module>

TypeError: 5 int! obj ect is not iterable

如果要让集合作为集合的元素，由于集合中的元素必须是一个可哈希对象，所以内部的集 合只能是frozenset对象。

除了使用上述两个函数创建集合以外，可以通过在大括号中放置以逗号分隔的元素列表来 创建非空set集合，语法格式如下:

{元素元素2,。…}

示例如下:

>» demo\_\_set = {1 z 2 z 3 f 3 5 }

>>> demo\_set

(lz 2, 3Z 5}

可以使用foi•循环遍历集合，访问集合中的每个元素。示例如下：

»> set\_01 = { va' z 'c\* z fd' }

>>> for item in set 01:

。.• print(item)

d

b

e

a

c

从示例的结果可以看出，遍历的元素顺序跟创建的元素顺序完全不同。由此说明，集合中 存放的元素是无序的。

集合支持使用ill或not ill操作符，用来检查某个元素是否存在于集合中。示例如下：

»> set\_\_01 = { 7 }

>>> \*f' in set\_01

False

>>> 1f, not in set\_01

True

如果要更新集合中的元素，如添加元素和删除元素等，就只支持对set集合类型进行操作。 为此,Python > set集合提供了一些常用的更新方法，如add()和remove()等。下面使用一张 表来列出这些更新集合的方法，如表7・2所示。

表乙2 s@t集合的常用方法

|  |  |
| --- | --- |
| 方法名 | 相关描述 |
| update(\*others) | 往集合中添加所有others的元素 |
| intersection\_update( \* others) | 集合中保留和others共有的元素 |
| difTerence\_update(\*otliers) | 从集合中删除与others共有的元素 |
| symmetric\_difiference\_update(other) | 包含集合本身或者others中的元素，但是不包含两者共有的元素 |
| add(elem) | 在集合中添加elem |
| remove(elem) | 从集合中删除 膈项。如果elem不是集合中的元素，贝U会引发KeyError错误 |
| discard(elem) | 如果elem是集合的元素，从集合中删除elem |
| POP() | 删除集合中的任意一个元素，并且返回它 |
| clear() | 删除集合中的所有元素 |

下面以添加和删除集合元素的功能为例，介绍如何用这些方法对集合进行更新操作，具体 如下。

(1 )添加元素

调用add()方法往集合里面添加元素,示例如下:

>>> demo\_set = set () #佥J建一个空s㊀七集合

>>> demo\_set. add ( ' a \* ) # 往集合中添加一个元素

>>> demo\_set

{0}

>>> demo\_set. add (1) # 再次往集合中添加一个元素

>>> demo\_set

{1, 0}

(2 )删除元素

要想删除set集合中的元素，可以通过remove(). discard()、pop()或clear()4种方法实现。其中， remove()和discardQ方法用于删除指定元素,pop()方法用于删除任意一个元素,而clear()方 法用于删除集合所有的元素。

下面使用几个示例进行演示，比较这些方法的不同之处，具体如下：

»>

»>

»>

{3,

»>

demo\_set = { J a \* , J. z 1 p ! , 3, ' h \ 5 ) demo\_set„remove(1) demo\_set

5, ，次}

demo set» remove(2)

#调用remove方法删除集合的某个元素

Traceback (most recent File "<stdin>"r line

call

1 in

#要删除的元素不存在,出现KeyError错误 last):

.<module>

KeyError: 2

>>> demo\_set.discard(\*p1)

»> demo\_set

{3, 5, 0}

>>> demo\_set.discard(2)

>>> element = d.emo\_set. pop ()

»> element

#调用discard方法删除集合的某个元素

#要删除的元素不存在，程序不做任何操作

#调用pop方法删除任意一个元素,并赋值给element

>>> demo set

{5, 'hS )

>>> demo\_set.clear()

#调用clear方法清空集合

>>> demo set

set ()

从示例的结果可以看出，如果要删除的元素不在集合中9 remove方法会出现KeyErro『异常, 而discard方法不会做任何操作。

除了上面这些方法以外，还可以使用-=操作符删除集合元素，它会找到集合与另一个集 合共有的元素，然后从集合中删除这些元素，示例如下：

»> demo\_set = ( 5 f ? h \ ' a 5 } >>> demo\_set

{5, 'h，； 0}

>» demo\_set 一= set (1 hai \* ) >>> demo\_set

{5} —

若希望删除集合本身，可以跟删除其他对象一样，调用del命令把集合直接删除。示例如下:

»> set\_01 = { 'aS ?b? )

>>> del set\_01

>>> set\_01

Traceback (most recent call last):

File "<stdin>"7 line 1z in <module>

NameError: name ? set 015 is not defined

7.7集会操（犖筒

7.7J 線瞻襲醴幽攤修籀 .. . " .

Python中的集合支持使用标准类型的操作符，包括成员操作符（in或not in）、等价操作 符（==或!=）和比较运算符（＞、V、＞=、＜=） o其中9比较运算符检查的是集合间是否为子 集或者超集的关系，下面对它们进行介绍。

（1）成员操作符

跟其他类型一样，ill或者not ill操作符用于判断某个值是或者不是集合的元素。示例如下：

>>> demo\_set = set(? python1)

>>> demo\_f rozenset = f rozenset (\* v/hat1)

>>> \* p v in demo\_\_set #用in操作符判断集合中是否有某个元素

True

>>> \*p\* in demo\_frozenset

False

»> 5w! not in demo\_set #用not in操作符判断某个元素不是集合的成员

True

（2）等价操作符

==和!=操作符用于比较两个相同或者不同的集合。等价是指一个集合的每个元素同时又 是另外一个集合中的元素，这种比较跟集合的类型和元素顺序无关，只是与集合的元素有关。

示例如下:

>>> set\_01

>>> set\_02

>>> set 01

set(\* who') set('how') ：set 02

True

>>> set\_03

>>> set\_01

= frozenset(1 how1)

==set\_03

#与集合元素的顺序无关

#与集合的类型无关

True

（3）比较运算符

比较运算符可以用来检查某个集合是否为其他集合的子集或者超集。其中，y 或者 运算符用于判断子集，和运算符用于判断的是超集。需要注意的是，％”和 运算符支持的是严格意义定义的子集和超集，为此它们不允许两个集合相等；而“＜=”和“＞=” 运算符支持的是非严格意义定义的子集和超集，为此它们允许两个集合是相等的。

| >>> set\_01 | | | =set (\* what') | | | |
| --- | --- | --- | --- | --- | --- | --- |
| »> | set\_ | \_02 |  | set ( | ?hat\*) |  |
| »> | set\_ | \_01 | < | set\_ | 02 | # |
| False | |  |  |  |  |  |
| »> | set\_ | \_01 | > | set\_ | 02 | # |
| True | |  |  |  |  |  |
| »> | set\_ | \_01 | 〉= | :set | \_02 | # |

示例如下:

True

判断set\_01是否为set\_02的严格子集

判断set\_01是否为set\_02的严格超集

判断set\_01是否为set\_02的非严格超集

7.7.2篥念襲醴»韻縮籍

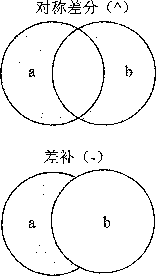
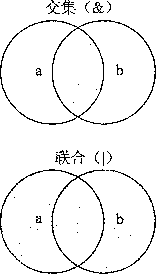
Python的集合之间支持联合(|)、交集(& )、 差补(Q和对称差分(人)的操作。已知，有两个集 合a和b,下面通过一张图来描述这四种操作的关系， 并且使用阴影部分显示操作的结果，如图7；所示。

图7；两个集合的相互操作

上述四种操作的介绍如下：

1 □嬲舍(I )

联合类似于数学集合中的并集操作，把所有属于 集合a或集合b的元素组成一个新的集合，这个新集 合的每个元素至少是a或b的元素。联合使用T符号 实现合并操作•与它等价的方法是union()0

示例如下:

>>> set\_01 = {? a s f \* c \*}

»> set~02 = { f e }

»> set\_01 | set\_02 #使用|操作符合并两个集合

{ 侦七"} \_

»>set\_01. uniom (set\_02) #使用union方法合并两个集合

{1c'A Ta'z 1b T}

2. »(&) •

交集就是把属于集合a且属于集合b的元素组成一个新集合，这个新集合的每个元素同时 是a和b两个集合的元素。交集使用符号实现相交操作，与它等价的方法是intersection()0

示例如下:

»> set\_01 = { 0 宀，}

»> set\_02 = m 七¥ }

»> set^Ol & set\_02 #使用&操作符获取两个集合共有的元素

{ « }

»>set\_01. intersection (set\_02) #使用intersection()方法获取两个集合共有的元素

差补指的是新集合中的元素只属于集合a或者集合bo差补使用y符号实现相对补集操作, 与它等价的方法是difference()o

示例如下:

>>> set\_01 = >>> set\_02 = >>> set\_01 - { 0 }

>>> set 02 —

{ 1 a \ ! c ! } { W, M } set\_\_02

set 01

{侦}

»> set\_01. difference (set\_02) { 0 }

»> set\_02 .difference (set\_01) {顷}

#使用-操作符获取只属于集合set\_01的元素

#使用-操作符获取只属于集合set\_\_02的元素

#使用difference ()获取只属于集合set\_01的元素

#使用difference ()获取只属于集合set\_02的元素

4.对称聲翁

对称差分是指只把属于集合a或者集合b且不能同时属于两个集合的元素组成一个新集合, 这个新集合的每个元素只能是初或者b的元素。对称差分使用方符号实现相交除外的操作，与 它等价的方法是 symmetric\_difference()°

示例如下:

>>> set\_01 = {\* a \* f \* c f}

»> set\_02 = { 1 b \ f c ? }

>>> set\_01八set\_02 #使用八操作符获取只属于set\_01和只属于set\_02的元素

{。b0 }

#使用difference ()方法获取只属于set\_01和只属于set\_02的元素

>>> set\_01.symmetric\_difference(set\_02)

{?af,

鼠濕舍篥念羹墾操作

如果使用运算符操作的两个集合属于同一种类型，就是说都是可变集合或不可变集合，则 所产生的结果类型是相同的。如果两个集合的类型不相同，如一方为可变集合，另一方为不可 变集合，则所产生的结果类型与运算符左侧集合的类型相同。

示例如下:

>>> demo\_set = set(!abc ?)

>>> demo\_frozenset = frozenset(\* cde\*)

>>> demo\_frozenset A demo\_set

frozenset({\* d, f v b ? f !a 1, , e!})

>>> demo\_\_set & demo\_frozenset

{ }

7.7.3圆靈靈倉襲饗齣繰傕籀 -

由于可变集合类型的灵活性，除了上述讲的操作符以外，Python另外提供了仅适用于汨 集合的操作符，包括|=(联合更新)、&=(交集更新)、-=(差补更新)和八=(对称差分更新)。 这些操作符的表达的含义具体如下:

1驍舍礎新(加) 在已经存在的集合中添加元素，该操作与updateQ方法是等价的。

示例如下:

>>> demo\_set = set (\*abc ,)

>» demo\_f rozenset = f rozenset ( \* bedJ)

»> demo\_set | = demo\_f rozenset # 使用 | =操作符在 demo\_set 集合中添加 demo\_f rozenset 元素 >>> demo\_set

( M1, 0, ?cJ }

2. »»(&=}

保留与其他集合的共有元素,该操作与intersection\_update()方法是等价的。 示例如下：

»> demo\_set = set (\* abc1)

>>> demo\_frozenset = frozenset('bed\*)

»> demo\_set &= demo\_frozenset # 使用 &=操作符保留 demo\_frozenset 集合共有的元素 >>> demo\_set { \*b\ 七星}

去掉其他集合中的元素后剩余的元素*，*该操作与difference\_update()方法是等价的。 示例如下:

>>> demo\_set = set (\* abc5)

»> demo\_frozenset = frozenset(!bcd5)

»> demo\_set \_= demo\_frozenset #使用-=操作符去掉跟demo\_frozenset集合共有的元素 »> demo\_set

{，泌}

4.对祢鑿分蔓新(心) ’

对集合a和b执行对称差分更新操作返回一个集合，这个集合中的元素是原集合a或者仅 是另一个集合b中的元素，该操作与symmetric\_difference\_update()方法是等价的。

示例如下:

>» demo\_set = set (5 abc 1)

>>> demo\_frozenset = frozenset(\* beds)

>>> demo\_set A= demo\_frozenset

>>> demo\_set

{，d\ 0}

i 篥念内建方裱

前面在介绍集合的操作符时，已经列举出了与操作符等价的内建方法，它们的功能是一样 的。接下来，通过一张表来列举集合的内建方法，如表7・3所示。

表7-3集合的内建方法

|  |  |
| --- | --- |
| 方法名 | 相关描述 |
| s.isdisjoint(other) | 如果集合与othei•中没有相同的元素，则返回True。当且仅当它们的交集是空集合时，集合是不 相交的 |
| s.issubset(other) | 如果集合是。恥啲子集,则返回Tnie,否则返回False |
| s.issuperset(other) | 如果集合是other的超集，则返回True，否则返回False |
| s.union("others) | 返回一个新集合,该集合的元素是$和others的合并的元素 |
| s. inters ection( \* others) | 返回一个新集合，该集合中的元素是smoothers的共有的元素 |
| s.difference(\*others) | 返回一个新集合,该集合中的元素是s的元素,但不是others的元素 |
| s.symmetric\_diflerence(other) | 返回一个新集合,该集合中的元素是域others的元素，但不是s和others共有的元素 |
| s.copy() | 返回一个新集合，它是集合s的浅复制 |

这些方法的操作前面都已经介绍过了，这里就不再过多赘述了。