使用dict和set

廖雪峰
资深软件开发工程师，业余马拉松选手。

dict

Python内置了字典：dict的支持，dict全称dictionary，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度。

举个例子，假设要根据同学的名字查找对应的成绩，如果用list实现，需要两个list：

```
names = ['Michael', 'Bob', 'Tracy']
scores = [95, 75, 85]
```

给定一个名字，要查找对应的成绩，就先要在names中找到对应的位置，再从scores取出对应的成绩，list越长，耗时越长。

如果用dict实现，只需要一个“名字”-“成绩”的对照表，直接根据名字查找成绩，无论这个表有多大，查找速度都不会变慢。用Python写一个dict如下：

```
>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}
>>> d['Michael']
95
```

为什么dict查找速度这么快？因为dict的实现原理和查字典是一样的。假设字典包含了1万个汉字，我们要查某一个字，一个办法是把字典从第一页往后翻，直到找到我们想要的字为止，这种方法就是在list中查找元素的方法，list越大，查找越慢。

第二种方法是先在字典的索引表里（比如部首表）查这个字对应的页码，然后直接翻到该页，找到这个字。无论找哪个字，这种查找速度都非常快，不会随着字典大小的增加而变慢。

dict就是第二种实现方式，给定一个名字，比如 `'Michael'` ，dict在内部就可以直接计算出 `Michael` 对应的存放成绩的“页码”，也就是 `95` 这个数字存放的内存地址，直接取出来，所以速度非常快。

你可以猜到，这种key-value存储方式，在放进去的时候，必须根据key算出value的存放位置，这样，取的时候才能根据key直接拿到value。

把数据放入dict的方法，除了初始化时指定外，还可以通过key放入：

```
>>> d['Adam'] = 67
>>> d['Adam']
67
```

由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉：

```
>>> d['Jack'] = 90
>>> d['Jack']
90
>>> d['Jack'] = 88
>>> d['Jack']
88
```

如果key不存在，dict就会报错：

```
>>> d['Thomas']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Thomas'
```

要避免key不存在的错误，有两种办法，一是通过 `in` 判断key是否存在：

```
>>> 'Thomas' in d
False
```

二是通过dict提供的 `get()` 方法，如果key不存在，可以返回 `None` ，或者自己指定的value：

```
>>> d.get('Thomas')
>>> d.get('Thomas', -1)
-1
```

注意：返回 `None` 的时候Python的交互环境不显示结果。

要删除一个key，用 `pop(key)` 方法，对应的value也会从dict中删除：

```
>>> d.pop('Bob')
75
>>> d
{'Michael': 95, 'Tracy': 85}
```

请务必注意，dict内部存放的顺序和key放入的顺序是没有关系的。

和list比较，dict有以下几个特点：

1. 查找和插入的速度极快，不会随着key的增加而变慢；
2. 需要占用大量的内存，内存浪费多。

而list相反：

1. 查找和插入的时间随着元素的增加而增加；
2. 占用空间小，浪费内存很少。

所以，dict是用空间来换取时间的一种方法。

dict可以用在需要高速查找的很多地方，在Python代码中几乎无处不在，正确使用dict非常重要，需要牢记的第一条就是dict的key必须是**不可变对象**。

这是因为dict根据key来计算value的存储位置，如果每次计算相同的key得出的结果不同，那dict内部就完全混乱了。这个通过key计算位置的算法称为哈希算法（Hash）。

要保证hash的正确性，作为key的对象就不能变。在Python中，字符串、整数等都是不可变的，因此，可以放心地作为key。而list是可变的，就不能作为key：

```
>>> key = [1, 2, 3]
>>> d[key] = 'a list'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

set

set和dict类似，也是一组key的集合，但不存储value。由于key不能重复，所以，在set中，没有重复的key。

要创建一个set，用 `{x,y,z,...}` 列出每个元素：

```
>>> s = {1, 2, 3}
>>> s
{1, 2, 3}
```

或者提供一个list作为输入集合：

```
>>> s = set([1, 2, 3])
>>> s
{1, 2, 3}
```

注意，传入的参数 `[1, 2, 3]` 是一个list，而显示的 `{1, 2, 3}` 只是告诉你这个set内部有1，2，3这3个元素，显示的顺序也不表示set是有序的。。

重复元素在set中自动被过滤：

```
>>> s = {1, 1, 2, 2, 3, 3}
>>> s
{1, 2, 3}
```

通过 `add(key)` 方法可以添加元素到set中，可以重复添加，但不会有效果：

```
>>> s.add(4)
>>> s
{1, 2, 3, 4}
>>> s.add(4)
>>> s
{1, 2, 3, 4}
```

通过 `remove(key)` 方法可以删除元素：

```
>>> s.remove(4)
>>> s
{1, 2, 3}
```

set可以看成数学意义上的无序和无重复元素的集合，因此，两个set可以做数学意义上的交集、并集等操作：

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s1 & s2
{2, 3}
>>> s1 | s2
{1, 2, 3, 4}
```

set和dict的唯一区别仅在于没有存储对应的value，但是，set的原理和dict一样，所以，同样不可以放入可变对象，因为无法判断两个可变对象是否相等，也就无法保证set内部“不会有重复元素”。试试把list放入set，看看是否会报错。

再议不可变对象

上面我们讲了，str是不变对象，而list是可变对象。

对于可变对象，比如list，对list进行操作，list内部的内容是会变化的，比如：

```
>>> a = ['c', 'b', 'a']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

而对于不可变对象，比如str，对str进行操作呢：

```
>>> a = 'abc'
>>> a.replace('a', 'A')
'Abc'
>>> a
'abc'
```

虽然字符串有个 `replace()` 方法，也确实变出了 `'Abc'` ，但变量 `a` 最后仍是 `'abc'` ，应该怎么理解呢？

我们先把代码改成下面这样：

```
>>> a = 'abc'
>>> b = a.replace('a', 'A')
>>> b
'Abc'
>>> a
'abc'
```

要始终牢记的是， `a` 是变量，而 `'abc'` 才是字符串对象！有些时候，我们经常说，对象 `a` 的内容是 `'abc'` ，但其实是指， `a` 本身是一个变量，它指向的对象的内容才是 `'abc'` ：

当我们调用 `a.replace('a', 'A')` 时，实际上调用方法 `replace` 是作用在字符串对象 `'abc'` 上的，而这个方法虽然名字叫 `replace` ，但却没有改变字符串 `'abc'` 的内容。相反， `replace` 方法创建了一个新字符串 `'Abc'` 并返回，如果我们用变量 `b` 指向该新字符串，就容易理解了，变量 `a` 仍指向原有的字符串 `'abc'` ，但变量 `b` 却指向新字符串 `'Abc'` 了：

所以，对于不变对象来说，调用对象自身的任意方法，也不会改变该对象自身的内容。相反，这些方法会创建新的对象并返回，这样，就保证了不可变对象本身永远是不可变的。

参考源码

the_dict.py

the_set.py

小结

使用key-value存储结构的dict在Python中非常有用，选择不可变对象作为key很重要，最常用的key是字符串。

tuple虽然是不变对象，但试试把 `(1, 2, 3)` 和 `(1, [2, 3])` 放入dict或set中，并解释结果。

« 循环

函数 »

Comments

Comments loaded. To post a comment, please Sign In

lllllllll @ 2025/12/29 00:05:48

求解惑 (1,2,3)与（1,[2,3]）不都是tuple吗，为什么（1,[2,3]）不能作为key呢？ 是不是因为list和set会将整个tuple内的数据转化为key而不是把tuple转化为key?

... @ 2025/7/23 05:06:38

尝试将 (1, 2, 3) 和 (1, [2, 3]) 放入 dict 中 c = {'name':(1,2,3)} print(c) #打印{'name': (1, 2, 3)}

n = {'(1,2,3)':'name'} print(n) #打印{'(1,2,3)': 'name'}

d = {'name': (1,[2,3])} print(d) #打印{'name': (1, [2, 3])}

m = {'(1,[2,3])': 'name'} print(m) #打印{'(1,[2,3])': 'name'}

尝试将 (1, 2, 3) 和 (1, [2, 3]) 放入 set 中 e = {(1,2,3)} print(e) #打印{(1, 2, 3)}

k = {(1, [2, 3])} print(k)

#报错 k = {(1, [2, 3])} ^^^^^^^^^^^^^^ TypeError: unhashable type: 'list'

杰 @ 2025/12/14 23:21:42

这个就可以了，“(1, 2, 3)”就是一个不可变值。

```
k = {(1, 2, 3): [2, 3]}
print(k)
```

杰 @ 2025/12/14 23:24:40

这个也一样：

```
k= (1,)
d = {k: [2, 3]}
print(d)
```

↘ _◆◇丶若影°丶 @ 2025/4/9 23:27:09

特性	列表 (List)	元组 (Tuple)	集合 (Set)	字典 (Dictionary)
可变性	可变	不可变	可变	可变
元素类型	任意类型（可包含可变/不可变对象）	任意类型（可包含可变/不可变对象）	必须为不可变对象（可哈希）	键必须为不可变对象，值可为任意类型
有序性	有序	有序	无序	有序（Python 3.7+）
允许重复元素	是	是	否	键唯一，值可重复
语法示例	`[1, 'a', [2]]`	`(1, 'a', [2])`	`{1, 2, 3}`	`{'key1': 10, 'key2': 20}`

Read More ▼

蘭 @ 2025/4/17 03:30:41

good

晨兴理荒秽 @ 2025/5/25 21:57:31

赞

鱼 @ 2025/5/29 04:03:26

太棒了

渝 @ 2025/6/4 03:55:30

from不是form

365 @ 2025/6/7 03:41:00

good

@ 2025/6/8 03:15:56

good，但from

清柠檬 @ 2025/6/27 02:39:44

还可以再加点东西

Ali @ 2025/8/21 02:33:43

优秀

大海 @ 2025/8/31 05:37:01

good job!

AADM @ 2025/10/10 00:37:06

好

lv @ 2025/11/24 05:41:54

哈希表会有序吗

lv @ 2025/11/24 05:58:11

Python 3.7+ 的 dict 是有序的，且依然高效，底层靠“哈希索引 + 紧凑条目数组”实现。 通过hash计算key对应的hash值，同时保存key对应的顺序索引，hash值对应的不再是value而是对应的顺序索引，——计算哈希 → 找到 index→ 从条目数组取index索引的数据。 通过分离“查找结构”和“存储顺序结构”来实现，类似访问者模式，将数据处理与数据结构分离

杰 @ 2025/12/14 22:49:40

Python 3.7及以后版本：默认保持插入顺序 从Python 3.7开始，字典的插入顺序保存特性被正式纳入语言规范。这意味着在Python 3.7及更高版本中，字典默认会按照键的插入顺序来维护其内部顺序。因此，在这些版本中，内部存放顺序与key放入顺序有直接关系。

Rory @ 2025/12/3 01:30:26

我刚使用set集合，对其进行了list嵌入，运行报错了

卿绛 @ 2025/11/25 03:37:53

对请务必注意，dict内部存放的顺序和key放入的顺序是没有关系的。这句话做讨论，以下是我查到的结果
Python 3.7+：字典有序，插入顺序 = 存储顺序 = 遍历顺序。
Python 3.6 及以下：字典无序，存储顺序与插入顺序无关。
示例（Python 3.7+）：

```
d = {"name": "张三", "age": 20, "gender": "男"}
print(list(d.keys()))  # 输出：['name', 'age', 'gender']（和插入顺序一致）
```

示例（Python 3.6 及以下）：

```
d = {"name": "张三", "age": 20, "gender": "男"}
print(list(d.keys()))  # 可能输出：['age', 'name', 'gender']（顺序不确定）
```

是这样的吗

lv @ 2025/11/24 05:40:10

TypeError: cannot use 'tuple' as a set element (unhashable type: 'list') 元组本身虽然可以作为不可变元素，但其内部的元素可以是可变的，例如小结中提到的list 当元组中出现可以可变元素的时候，就无法保证key是唯一的，例如： (1, [2, 3])和(1, [2, 3 ,4])这两个元组可以是同一个元组，只不过改了其中可变元素的值 报错提示中，unhashable推测dict和set都会通过hash算法来计算key对应的hash值，不能进行hash计算的key会抛出异常

炬火 @ 2025/11/16 19:11:44

打卡第五天

落 @ 2025/11/13 03:11:48

方法 是否插入新键 用途 dict.get(key, default) 不插入 安全地获取值，键不存在时返回默认值 dict.setdefault(key, default) 插入 获取值，如果键不存在则设置默认值 collections.defaultdict 自动插入 需要频繁访问不存在的键时使用 dict[key] = value 插入 直接设置键值对

总结： get()：只读操作，不修改字典 setdefault() 或 defaultdict：读取并可能在需要时插入

贫穷是第一推动 @ 2025/11/11 00:40:34

试图d.pop(all)出错，要用d.clear()

bot吃宵夜 @ 2025/10/11 02:35:17

打卡!

Δ @ 2025/8/1 13:16:06

文章里只笼统的提了一下set，没说这玩意有啥用，补充一下： 1.因为他的不重复性，可以把一个存在重复的列表转化成set，然后再转回list，这就做到了去重，比for循环快。 2.这个还可以用来查一个元素在不在某个列表里，就比如说可以用在有些文章查违禁词。 3.还有集合的交并补运算，用各种符号就能实现运算，& | - ^。

JOhN @ 2025/8/13 11:12:06

受用，3Q~

大海 @ 2025/8/31 05:38:04

well done~

狼图腾-崛起 @ 2025/9/28 09:50:18

可以参考数学里面的集合，交集，并集，合集，分别有什么用途就很好理解了。交集，两个集合重合的地方。

土土🐰 @ 2025/9/27 11:38:10

第5天打卡：复习进度章节5.4

土土🐰 @ 2025/9/26 09:25:53

第4天打卡：复习进度章节5.2

土土🐰 @ 2025/9/25 11:16:43

第3天打卡 作业

试试把list放入set,是否会报错（答：会报错，不可哈希类型：列表）

```
>>> s = {1, 2, [4, 5]}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

把(1, 2, 3)和(1, [2, 3])放入dict或set中，并解释结果

1，放入dict中，（1，2，3）成功，元素皆为整数是不可变对象，符合哈希类型。(1, [2, 3])不成功，因[2, 3]为list是可变对象，不符合哈希类型。

```
>>> d = {}
>>> d[(1, 2, 3)] = 'success'
```

Read More ▼

土土🐰 @ 2025/9/25 10:33:54

第3天打卡 学习笔记--供参考（若有误望指出，感谢）

```
# dict : key-value存储取出的方式
d = {'Mike': 95, 'Bob': 75, 'Tracy': 85}
d['Mike'] # -> 95  调取value
d['Mike'] = 80  #  存入value，每存入新的值会覆盖掉上一个值
d['Mike'] # -> 80  调取value值为80，已覆盖掉原value：95
# 如果key不存在，直接输入 d[key],会KeyError,可能程序崩溃
# 避免key不存在，两种方法验证
# 1. in
```

```
'Coco' in d
False # 不存在,因为没有Coco这个key
# 2. get()
d.get('Coco') #没有任何输出，结果为None虽不显示但程序仍进行
d.get('Coco', -1) # 或后面输入指定值
-1 # 若key存在,指定值写错取出的value也会是正确值
```

Read More ▼

辞忧 @ 2025/9/20 20:00:08

高中生第一周打卡❤️

Doing_Alright @ 2025/9/14 22:38:14

打卡

橘落 @ 2025/8/16 03:22:53

d = {(1, 2, 3)} c = {(1, [2, 3])} print(d) print(c)

运行结果如下

Traceback (most recent call last): in <module> c = {(1, [2, 3])} TypeError: unhashable type: 'list'

云帆 @ 2025/8/7 05:47:42

```
key = (1, 2, 3)  # 使用元组作为键
d = {}  # 先初始化字典
d[key] = 'a list'

print(d)  # 输出: {(1, 2, 3): 'a list'}
```

🎄 @ 2025/8/7 04:29:30

day4打卡