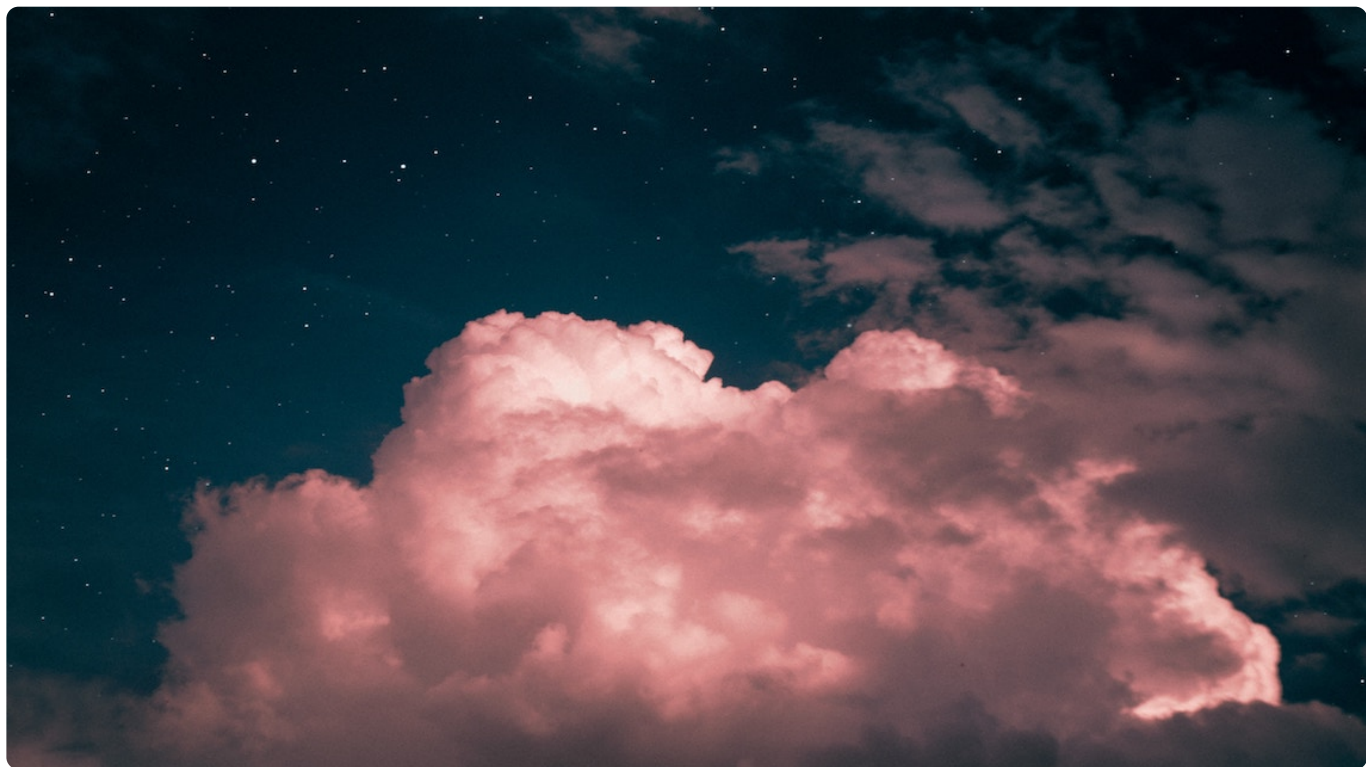


03 | 列表和元组，到底用哪一个？

2019-05-15 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 07:16 大小 6.66M



你好，我是景霄。

前面的课程，我们讲解了 Python 语言的学习方法，并且带你了解了 Python 必知的常用工具——Jupyter。那么从这节课开始，我们将正式学习 Python 的具体知识。


对于每一门编程语言来说，数据结构都是其根基。了解掌握 Python 的基本数据结构，对于学好这门语言至关重要。今天我们就一起来学习，Python 中最常见的两种数据结构：列表（list）和元组（tuple）。

列表和元组基础

首先，我们需要弄清楚最基本的概念，什么是列表和元组呢？

实际上，列表和元组，都是一个可以放置任意数据类型的有序集合。

在绝大多数编程语言中，集合的数据类型必须一致。不过，对于 Python 的列表和元组来说，并无此要求：

 复制代码


```
1 l = [1, 2, 'hello', 'world'] # 列表中同时含有 int 和 string 类型的元素
2 l
3 [1, 2, 'hello', 'world']
4
5 tup = ('jason', 22) # 元组中同时含有 int 和 string 类型的元素
6 tup
7 ('jason', 22)
```

其次，我们必须掌握它们的区别。

列表是动态的，长度大小不固定，可以随意地增加、删减或者改变元素（mutable）。

而元组是静态的，长度大小固定，无法增加删减或者改变（immutable）。

下面的例子中，我们分别创建了一个列表与元组。你可以看到，对于列表，我们可以很轻松地让其最后一个元素，由 4 变为 40；但是，如果你对元组采取相同的操作，Python 就会报错，原因就是元组是不可变的。


 复制代码

```
1 l = [1, 2, 3, 4]
2 l[3] = 40 # 和很多语言类似，python 中索引同样从 0 开始，l[3] 表示访问列表的第四个元素
3 l
4 [1, 2, 3, 40]
5
6 tup = (1, 2, 3, 4)
7 tup[3] = 40
8 Traceback (most recent call last):
9   File "<stdin>", line 1, in <module>
10 TypeError: 'tuple' object does not support item assignment
```

可是，如果你想对已有的元组做任何“改变”，该怎么办呢？那就只能重新开辟一块内存，创建新的元组了。

比如下面的例子，我们想增加一个元素 5 给元组，实际上就是创建了一个新的元组，然后把原来两个元组的值依次填充进去。


而对于列表来说，由于其是动态的，我们只需简单地在列表末尾，加入对应元素就可以了。如下操作后，会修改原来列表中的元素，而不会创建新的列表。

 复制代码

```
1 tup = (1, 2, 3, 4)
2 new_tup = tup + (5, ) # 创建新的元组 new_tup，并依次填充原元组的值
3 new_tup
4 (1, 2, 3, 4, 5)
5
6 l = [1, 2, 3, 4]
7 l.append(5) # 添加元素 5 到原列表的末尾
8 l
9 [1, 2, 3, 4, 5]
```


通过上面的例子，相信你肯定掌握了列表和元组的基本概念。接下来我们来看一些列表和元组的基本操作和注意事项。

首先，和其他语言不同，**Python 中的列表和元组都支持负数索引**，-1 表示最后一个元素，-2 表示倒数第二个元素，以此类推。

 复制代码

```
1 l = [1, 2, 3, 4]
2 l[-1]
3 4
4
5 tup = (1, 2, 3, 4)
6 tup[-1]
7 4
```


除了基本的初始化，索引外，**列表和元组都支持切片操作**：

 复制代码

```
1 l = [1, 2, 3, 4]
2 l[1:3] # 返回列表中索引从 1 到 2 的子列表
```


```
3 [2, 3]
4
5 tup = (1, 2, 3, 4)
6 tup[1:3] # 返回元组中索引从 1 到 2 的子元组
7 (2, 3)
```

另外，列表和元组都可以随意嵌套：

 复制代码

```
1 l = [[1, 2, 3], [4, 5]] # 列表的每一个元素也是一个列表
2
3 tup = ((1, 2, 3), (4, 5, 6)) # 元组的每一个元素也是一元组
```

当然，两者也可以通过 `list()` 和 `tuple()` 函数相互转换：

 复制代码

```
1 list((1, 2, 3))
2 [1, 2, 3]
3
4 tuple([1, 2, 3])
5 (1, 2, 3)
```

最后，我们来看一些列表和元组常用的内置函数：

 复制代码

```
1 l = [3, 2, 3, 7, 8, 1]
2 l.count(3)
3 2
4 l.index(7)
5 3
6 l.reverse()
7 l
8 [1, 8, 7, 3, 2, 3]
9 l.sort()
10 l
11 [1, 2, 3, 3, 7, 8]
12
13 tup = (3, 2, 3, 7, 8, 1)
```

```
14 tup.count(3)
15 2
16 tup.index(7)
17 3
18 list(reversed(tup))
19 [1, 8, 7, 3, 2, 3]
20 sorted(tup)
21 [1, 2, 3, 3, 7, 8]
```

这里我简单解释一下这几个函数的含义。

`count(item)` 表示统计列表 / 元组中 `item` 出现的次数。


`index(item)` 表示返回列表 / 元组中 `item` 第一次出现的索引。

`list.reverse()` 和 `list.sort()` 分别表示原地倒转列表和排序（注意，元组没有内置的这两个函数）。

`reversed()` 和 `sorted()` 同样表示对列表 / 元组进行倒转和排序，但是会返回一个倒转后或者排好序的新的列表 / 元组。

列表和元组存储方式的差异


前面说了，列表和元组最重要的区别就是，列表是动态的、可变的，而元组是静态的、不可变的。这样的差异，势必会影响两者存储方式。我们可以来看下面的例子：

 复制代码

```
1 l = [1, 2, 3]
2 l.__sizeof__()
3 64
4 tup = (1, 2, 3)
5 tup.__sizeof__()
6 48
7
```

你可以看到，对列表和元组，我们放置了相同的元素，但是元组的存储空间，却比列表要少 16 字节。这是为什么呢？

事实上，由于列表是动态的，所以它需要存储指针，来指向对应的元素（上述例子中，对于 int 型，8 字节）。另外，由于列表可变，所以需要额外存储已经分配的长度大小（8 字节），这样才可以实时追踪列表空间的使用情况，当空间不足时，及时分配额外空间。

 复制代码

```
1 l = []
2 l.__sizeof__() // 空列表的存储空间为 40 字节
3 40
4 l.append(1)
5 l.__sizeof__()
6 72 // 加入了元素 1 之后，列表为其分配了可以存储 4 个元素的空间 (72 - 40)/8 = 4
7 l.append(2)
8 l.__sizeof__()
9 72 // 由于之前分配了空间，所以加入元素 2，列表空间不变
10 l.append(3)
11 l.__sizeof__()
12 72 // 同上
13 l.append(4)
14 l.__sizeof__()
15 72 // 同上
16 l.append(5)
17 l.__sizeof__()
18 104 // 加入元素 5 之后，列表的空间不足，所以又额外分配了可以存储 4 个元素的空间
```

上面的例子，大概描述了列表空间分配的过程。我们可以看到，为了减小每次增加 / 删减操作时空间分配的开销，Python 每次分配空间时都会额外多分配一些，这样的机制（over-allocating）保证了其操作的高效性：增加 / 删除的时间复杂度均为 $O(1)$ 。

但是对于元组，情况就不同了。元组长度大小固定，元素不可变，所以存储空间固定。

看了前面的分析，你也许会觉得，这样的差异可以忽略不计。但是想象一下，如果列表和元组存储元素的个数是一亿，十亿甚至更大数量级时，你还能忽略这样的差异吗？

列表和元组的性能


通过学习列表和元组存储方式的差异，我们可以得出结论：元组要比列表更加轻量级一些，所以总体上来说，元组的性能速度要略优于列表。

另外，Python 会在后台，对静态数据做一些**资源缓存**（resource caching）。通常来说，因为垃圾回收机制的存在，如果一些变量不被使用了，Python 就会回收它们所占用的内

存，返还给操作系统，以便其他变量或其他应用使用。

但是对于一些静态变量，比如元组，如果它不被使用并且占用空间不大时，Python 会暂时缓存这部分内存。这样，下次我们再创建同样大小的元组时，Python 就可以不用再向操作系统发出请求，去寻找内存，而是可以直接分配之前缓存的内存空间，这样就能大大加快程序的运行速度。

下面的例子，是计算**初始化**一个相同元素的列表和元组分别所需的时间。我们可以看到，元组的初始化速度，要比列表快 5 倍。

 复制代码

```
1 python3 -m timeit 'x=(1,2,3,4,5,6)'  
2 20000000 loops, best of 5: 9.97 nsec per loop  
3 python3 -m timeit 'x=[1,2,3,4,5,6]'  
4 5000000 loops, best of 5: 50.1 nsec per loop
```

但如果是**索引操作**的话，两者的速度差别非常小，几乎可以忽略不计。

 复制代码


```
1 python3 -m timeit -s 'x=[1,2,3,4,5,6]' 'y=x[3]'  
2 10000000 loops, best of 5: 22.2 nsec per loop  
3 python3 -m timeit -s 'x=(1,2,3,4,5,6)' 'y=x[3]'  
4 10000000 loops, best of 5: 21.9 nsec per loop
```

当然，如果你想要增加、删减或者改变元素，那么列表显然更优。原因你现在肯定知道了，那就是对于元组，你必须得通过新建一个元组来完成。

列表和元组的使用场景


那么列表和元组到底用哪一个呢？根据上面所说的特性，我们具体情况具体分析。

1. 如果存储的数据和数量不变，比如你有一个函数，需要返回的是一个地点的经纬度，然后直接传给前端渲染，那么肯定选用元组更合适。

 复制代码

```
1 def get_location():
2     .....
3     return (longitude, latitude)
```

2. 如果存储的数据或数量是可变的，比如社交平台上的一个日志功能，是统计一个用户在一周之内看了哪些用户的帖子，那么则用列表更合适。

 复制代码

```
1 viewer_owner_id_list = [] # 里面的每个元素记录了这个 viewer 一周内看过的所有 owner 的 id
2 records = queryDB(viewer_id) # 索引数据库，拿到某个 viewer 一周内的日志
3 for record in records:
4     viewer_owner_id_list.append(record.id)
```

总结

关于列表和元组，我们今天聊了很多，最后一起总结一下你必须掌握的内容。


总的来说，列表和元组都是有序的，可以存储任意数据类型的集合，区别主要在于下面这两点。

列表是动态的，长度可变，可以随意的增加、删减或改变元素。列表的存储空间略大于元组，性能略逊于元组。

元组是静态的，长度大小固定，不可以对元素进行增加、删减或者改变操作。元组相对于列表更加轻量级，性能稍优。

思考题

1. 想创建一个空的列表，我们可以用下面的 A、B 两种方式，请问它们在效率上有什么区别吗？我们应该优先考虑使用哪种呢？可以说说你的理由。

 复制代码

```
1 # 创建空列表
2 # option A
3 empty_list = list()
4
5 # option B
```



```
6 empty_list = []
```

2. 你在平时的学习工作中，是在什么场景下使用列表或者元组呢？欢迎留言和我分享。

 极客时间

Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | Jupyter Notebook为什么是现代Python的必学技术？

下一篇 04 | 字典、集合，你真的了解吗？

精选留言 (125)

 写留言



胡晓

2019-05-15

 57

老师能不能讲一下list和tuple的内部实现，里边是linked list 还是array，还是把array linked一下这种。

最后那个问题，类比java，new 是在heap，直接声明就可能在常量区了。老师能讲下Python的vm么，比如内存分配，gc算法之类的。

作者回复: 1. list和tuple的内部实现都是array的形式, list因为可变, 所以是一个over-allocate的array, tuple因为不可变, 所以长度大小固定。具体可以参照源码list:

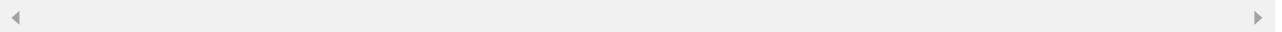
<https://github.com/python/cpython/blob/master/Objects/listobject.c> tuple:

<https://github.com/python/cpython/blob/master/Objects/tupleobject.c>

2. 最后的思考题:

区别主要在于list()是一个function call, Python的function call会创建stack, 并且进行一系列参数检查的操作, 比较expensive, 反观[]是一个内置的C函数, 可以直接被调用, 因此效率高。

内存分配, GC等等知识会在第二章进阶里面专门讲到。



Python高...

2019-05-15

44

元素不需要改变时:

两三个元素, 使用 tuple, 元素多一点使用namedtuple。

元素需要改变时:

需要高效随机读取, 使用list。需要关键字高效查找, 采用 dict。去重, 使用 set。大型数据节省空间, 使用标准库 array。大型数据高效操作, 使用 numpy.array。

展开 v



看, 有只猪

2019-05-15

30

[]比list()更快, 因为调用list函数有一定的开销, 而[]却没有。

这个有点像C语言中的内联函数与函数的差异



adapt

2019-05-15

27

如果一个列表在元组中的话, 其实这个元组是“可变”的, 只是这个可变只是能改变该列表里的内容。这一点作者没有讲到哦。



对方正在输...

2019-05-15

15

```
python -m timeit 'empty_list = list()'
```

10000000 loops, best of 3: 0.0829 usec per loop

```
python -m timeit 'empty_list = []'
10000000 loops, best of 3: 0.0218 usec per loop...
```

展开 ▾



converse★

2019-05-17

👍 13

空list在申请空间时候，是40字节。当加入新元素后会额外多分配空间变成72字节。当加入4个元素后还是72字节。那么问题来了，初始化申请的40字节是什么？感觉一直没有用到？不是用于存储元素的么？

展开 ▾



DUDUANWANG...

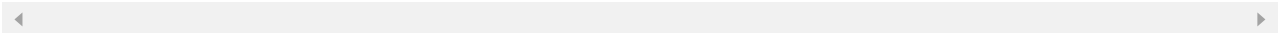
2019-05-15

👍 10

老师请问一下，为什么l = [1, 2, 3]消耗的空间为64字节，而l.append(1), l.append(2), l.append(3)消耗的空间为72字节，这不是相同的列表吗？

作者回复: 列表的over-allocate是在你加入了新元素之后解释器判断得出当前存储空间不够，给你分配额外的空间，因此

l=[], l.append(1), l.append(2), l.append(3)实际分配了4个元素的空间。但是l=[1, 2, 3]直接初始化列表，并没有增加元素的操作，因此只会分配3个元素的空间



converse★

2019-05-17

👍 9

针对可以随意嵌套进行总结：

- 列表嵌套列表：本质是列表，内部列表和外部列表的内容可以进行修改元素，插入，删除元素。也就是二维数组。

- 列表嵌套元组：本质是列表，所以可以对列表中除元组外的其他元素可以修改插入、删...

展开 ▾



Geek_59f23...

2019-05-15

👍 5

实测被打脸了😂函数构建和直接构建一个空列表或数组速度上并没有什么差别，有时前者

快些，有时后者快些。。。。

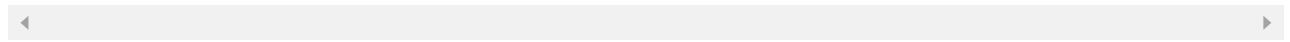
```
In [1]: timeit 'lst1 = []'
```

9.86 ns ± 0.721 ns per loop (mean ± std. dev. of 7 runs, 100000000 loops each)...

展开 ∨

作者回复: 你的命令有些奇怪。在程序里也应该是timeit(...), 试过用文中的命令测试的结果吗?

另外你python的版本和运行环境的截图能贴一下吗?



Geek_59f23...

2019-05-15

👍 5

1、用list()方法构造一个空列表使用的是class list([iterable])的类型构造器，参数可以是一个iterable，如果没有给出参数，构造器将创建一个空列表[]，相比较而言多了一步class调用和参数判断，所以用[]直接构造一个空列表的方法速度更快，刚查的官方解释，不知道我理解的对不对。。。。

2、敲代码的时候我一般元祖用来传参用的比较多，能用元祖的地方尽量不用列表，这样...

展开 ∨

作者回复: 1. 区别主要在于list()是一个function call，Python的function call会创建stack，并且进行一系列参数检查的操作，比较expensive，反观[]是一个内置的C函数，可以直接被调用，因此效率高

2. 嗯嗯



.....

2019-05-15

👍 5

1. 测试了一下，[]快于list()

2. 一般在key中使用元祖，其他情况多数都使用列表



高权

2019-05-23

👍 4

为什么我测试的元组和列表的初始化时间一样呢?

展开 ∨



汤尼房

2019-05-15

👍 3

景老师，一直在想一个tuple元组如何拥有大数据量的元素，比如千万个元素、上亿个元素。因为tuple是静态的，不能添加元素，于是今天实践将[i for i in xrange(1000000000)]给初始化成tuple，发现初始化的过程相当耗时间，之前也希望利用tuple的性能比list好的优点，想把含有大数据量的list给转换成tuple来处理，今天实践发现初始化过程非常耗时间，请问景老师，平时在工作过程中遇到的含有大数据量个元素的...

展开 ▾



kevinsu

2019-05-15

👍 3

```
import timeit
print(timeit.timeit('list(x for x in range(1,1000))',number=10000))
print(timeit.timeit('[x for x in range(1,1000)]',number=10000))
0.6829426919994148
0.36637431800045306...
```

展开 ▾



刘朋

2019-05-15

👍 2

```
import timeit
timeit.timeit('a=list()',number=10000) 返回 0.0006914390251040459
timeit.timeit('a=[]',number=10000) 返回 0.00018375739455223083
timeit.timeit('a=()',number=10000) 返回 0.00010870955884456635
```



Jared

2019-05-15

👍 2

老师真帅。

展开 ▾



許敲敲

2019-05-15

👍 2

思考题 试了运行timeit，发现在我电脑上两个创建列表的时间一样，所以答案是什么呢？



GLADIATOR

2019-05-16

👍 1

老师，您说列表和元组是有序的数据集合，这个“有序”怎么理解？list=[3,2,1,10,11,1,2]
这个列表何来有序？



lizhaocha...

2019-05-16

👍 1

list存的是元素的指针，那么

1. 存指针的地方是array吗
2. 存元素的地方也是array吗

展开 ▾



enjoylear...

2019-05-16

👍 1

带上括号就变为函数调用，所以会耗费时间

展开 ▾