链滴

python 中的列表和元组都有哪些区别

作者：zyjImmortal
原文链接：https://ld246.com/article/1586439035607
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

列表(list)和元组(tuple)的一些基础

list和tuple都是一个可以放置任意数据类型的有序集合，都是既可以存放数字、字符串、对象等

```
In [1]: nums = [2,3,'ad','d']

In [2]: nums
Out[2]: [2, 3, 'ad', 'd']

In [3]: tp = (1,3,'33','vf')

In [4]: tp
Out[4]: (1, 3, '33', 'vf')
```

list和tuple都支持负索引

In [8]: nums[-2]
Out[8]: 'ad'
In [9]: tp[-2]
Out[9]: '33'

list和tuple都支持切片操作

In [10]: nums[1:3]
Out[10]: [3, 'ad']

原文链接：python 中的列表和元组都有哪些区别

```
In [11]: tp[1:3]
Out[11]: (3, '33')
```

list和tuple都可以随意嵌套

```
In [12]: nums = [[1,2,3],['s','ff'],['34',3,5]]
In [13]: tp = ((23,4,'f'),45,'d',('dd',4,'ff'))
```

list和tuple的一些区别

- 列表是动态的，长度大小不固定，可以随意的增加、删除、修改元素
- 元组是静态的，长度在初始化的时候就已经确定不能更改，更无法增加、删除、修改元素

```
In [5]: nums[0] = 40

In [6]: nums
Out[6]: [40, 3, 'ad', 'd']

In [7]: tp[0]=44
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-0805b13b9133> in <module>()
----> 1 tp[0]=44

TypeError: 'tuple' object does not support item assignment
```

从图中看出我们对list做出修改是成功的，但是对tuple修改的时候，确抛出了错误，那如果想对tuple出改变该如何做呢？只能重新开辟一块内存，重新生成新到的tuple了。

```
class tuple(object):
    """
    tuple() -> empty tuple
    tuple(iterable) -> tuple initialized from iterable's items

    If the argument is a tuple, the return value is the same object.
    """
    def count(self, value): # real signature unknown; restored from __doc__
        """ T.count(value) -> integer -- return number of occurrences of value """
        return 0

    def index(self, value, start=None, stop=None): # real signature unknown; restored fr
        """
        T.index(value, [start, [stop]]) -> integer -- return first index of value.
        Raises ValueError if the value is not present.
        """
        return 0
```

从tuple的源码中也可以看出，只有两个自带的方法，一个是统计元素出现的次数一个是查询元素的

原文链接：python 中的列表和元组都有哪些区别

引。

list和tuple存储方式的差异

我们先来看个例子

```
In [19]: nums=['a',1,2]
In [20]: tp=('a',1,2)
In [21]: nums.__sizeof__()
Out[21]: 64
In [22]: tp.__sizeof__()
Out[22]: 48
```

这里构造了一个list和一个tuple。他们存储的内容是相同的，__sizeof__方法可以打印系统分配空间大小。可以看到他们所占用的内存空间是不同的，存储的内容相同，但是list比tuple多占用了16自己内存。

先来看一下一个数组的内存分配过程：

```
In [23]: l=[]

In [24]: l.__sizeof__()  // 空列表分配了40字节的内存
Out[24]: 40

In [25]: l.append('a')  // 增加了一个一个元素后，给列表分配了72字节的内存，一个字符8个字节
                                    // 那就是一次性分配了4个字符的内存空间

In [26]: l.__sizeof__()
Out[26]: 72

In [27]: l.append('b')  // 再增加字符，占用内存不变

In [28]: l.__sizeof__()
Out[28]: 72

In [29]: l.append('c')  // 再增加字符，占用内存不变

In [30]: l.__sizeof__()
Out[30]: 72

In [31]: l.append('d')  // 再增加字符，占用内存不变

In [32]: l.__sizeof__()
Out[32]: 72

In [33]: l.append('e')  // 再添加元素，内存不够，触发重新的内存分配

In [34]: l.__sizeof__()
Out[34]: 104
```

可以看出list为了能够实时追踪内存的使用情况，当空间不足时以及分配额外空间，额外的多分配了存，而且还需要存储指针，指向对应的元素。

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

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( )。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。

所以说在存储大量的数据的时候，这种差异是需要考虑的，如果数据发生变更的可能性不大，就用元存储，如果数据是需要频繁的进行数据的修改增加，就使用列表

原文链接