**Python中参数前加星号的用法**

2016年01月05日 18:22:15 [pfm685757](https://me.csdn.net/pfm685757) 阅读数：6487

[**Python中参数前加星号的用法**](http://m.blog.csdn.net/blog/dragonice4026/34561259)

2014-6-26阅读269 [评论0](http://m.blog.csdn.net/blog/dragonice4026/34561259#comment)

刚开始折腾Python，今天发现了一个很奇怪的错误。

1. def accept(\*s):
2. print(sum(s))
3. list = (0, 1, 2, 3, 7.5)
4. print(sum(list))
5. accept(list)

这段代码会报错：TypeError: unsupported operand type(s) for +=: 'int' and 'tuple'，在函数中产生了int

与tuple相加的情况。但是函数外的sum（list）输出结果是正常的。

我们print(type(s))，得到的结果：<class 'tuple'>。既然s也是元组，为什么会产生这样的结果呢？我们

分别打印一下s与list，输出是这样的：

(0, 1, 2, 3, 7.5)((0, 1, 2, 3, 7.5),)

从这结果可以看到，将list直接传入accept函数，效果并不是像我们期望的那样将s赋值为list再进行操作，

而是将整个list元组作为s的第一个元素，因此s是一个元组的元组。

那么有没有办法将list的元素分别传递进去呢？当然是有的，使用 ‘\*’ 操作符。

我们将代码稍微更改成这样：

1. def accept(\*s):
2. print(sum(s))
3. list = (0, 1, 2, 3, 7.5)
4. print(sum(list))
5. accept(\*list)

输出结果：

13.5

13.5

这样就正常了。

‘\*’ 操作符的作用是将元组“解包”，则accept(\*list)等价于accept（0, 1, 2, 3, 7.5）。而对于关键字参数，

需要用两个星号，即‘\*\*’来解包。例如：

1. def accept(\*\*s):
2. print(s)
3. list = {'a': 0, 'b': 1, 'c': 2}
4. print(list)
5. accept(\*\*list)

其中accept(\*\*list)等价于accpet(a=0, b=1, c=2)。

来总结一下：

1、函数声明的参数列表中加单星号，即f(\*x)则表示x为元组，所有对x的操作都应将x视为元组类型

进行。所有传入f（\*x）的变量都将作为元组x的元素之一。

2、双星号同上，区别是x视为字典。

3、在变量前加单星号表示将元组（列表、集合）拆分为单个元素。

4、双星号同上，区别是目标为字典，字典前加单星号的话可以得到“键”。

# 关于Python 解包，你需要知道的一切

# 关于Python 解包，你需要知道的一切

[*unpacking*](https://foofish.net/tag/unpacking.html)

导读：本文总结了 Python 解包操作的方方面面，文章略长，看本文前，首先确保身边有多个不同版本 Python 解释器的电脑（公众号回复 conda ，了解如何安装多个环境)，以便随时验证代码。看完记得收藏，方便查阅)

解包在英文里叫做 Unpacking，就是将容器里面的元素逐个取出来（防杠精：此处描述并不严谨，因为容器中的元素并没有发生改变）放在其它地方，好比你老婆去菜市场买了一袋苹果回来分别发给家里的每个成员，这个过程就是解包。Python 中的解包是自动完成的，例如：

>>> a, b, c = [1,2,3]

>>> a

1

>>> b

2

>>> c

3

如果列表中有3个元素，那么刚好如果列表中有3个元素，那么刚好可以分配给3个变量。除了列表对象可以解包之外，任何可迭代对象都支持解包，可迭代对象包括元组、字典、集合、字符串、生成器等实现了\_\_next\_\_方法的一切对象。

元组解包

>>> a,b,c = (1,2,3)

>>> a

1

>>> b

2

>>> c

3

字符串解包

>>> a,b,c = "abc"

>>> a

'a'

>>> b

'b'

>>> c

'c'

字典解包

>>> a,b,c = {"a":1, "b":2, "c":3}

>>> a

'a'

>>> b

'b'

>>> c

'c'

字典解包后，只会把字典的 key 取出来，value 则丢掉了。

你可能见过**多变量赋值**操作，例如：

>>> a, b = 1, 2

>>> a

1

>>> b

2

本质上也是自动解包过程，等号右边其实是一个元组对象 **(1, 2)**，有时候我们代码不小心多了一个逗号 **,**，就变成了元组对象

>>> a = 1,

>>> a

(1,)

----------

>>> a = 1

>>> a

1

所以写代码的时候需要特别注意。在Python 中，交换两个变量非常方便，本质上也是自动解包过程。

>>> a, b = 1, 2

>>> a, b = b, a

>>> a

2

>>> b

1

如果在解包过程中，遇到左边变量个数小于右边可迭代对象中元素的个数时该怎么办？ 好比你们家有3口人，你老婆却买了4个苹果，怎么分配呢？

在Python2中，如果等号左边变量的个数不等于右边可迭代对象中元素的个数，是不允许解包的。但在 Python3 可以这么做了。这个特性可以在 [PEP 3132](https://www.python.org/dev/peps/pep-3132/) 中看到。

>>> a, b, \*c = [1,2,3,4]

>>> a

1

>>> b

2

>>> c

[3, 4]

>>>

这种语法就是在某个变量面前加一个星号，而且这个星号可以放在任意变量，每个变量都分配一个元素后，剩下的元素都分配给这个带星号的变量

>>> a, \*b, c = [1,2,3,4]

>>> a

1

>>> b

[2, 3]

>>> c

4

这种语法有什么好处呢？它使得你的代码写起来更简洁，比如上面例子，在 Python2 中该怎么操作呢？思考3秒钟，再看答案。

>>> n = [1,2,3,4]

# 使用切片操作

>>> a, b, c = n[0], n[1:-1], n[-1]

>>> a

1

>>> b

[2, 3]

>>> c

4

以上是表达式解包的一些操作，接下来介绍函数调用时的解包操作。函数调用时，有时你可能会用到两个符号：星号**\***和 双星号**\*\***。

>>> def func(a,b,c):

... print(a,b,c)

...

>>> func(1,2,3)

1 2 3

func 函数定义了三个位置参数 a，b，c，调用该函数必须传入三个参数，除此之外，你也可以传入包含有3个元素的可迭代对象，

>>> func(\*[1,2,3])

1 2 3

>>> func(\*(1,2,3))

1 2 3

>>> func(\*"abc")

a b c

>>> func(\*{"a":1,"b":2,"c":3})

a b c

函数被调用的时候，使用星号 **\*** 解包一个可迭代对象作为函数的参数。字典对象，可以使用两个星号，解包之后将作为关键字参数传递给函数

>>> func(\*\*{"a":1,"b":2,"c":3})

1 2 3

看到了吗？和上面例子的区别是多了一个星号，结果完全不一样，原因是什么？ 答案是 **\*\*** 符号作用的对象是字典对象，它会自动解包成关键字参数 key=value 的格式：

>>> func(a=1,b=2,c=3)

1 2 3

如果字典对象中的 key 不是 a，b，c，会出现什么情况？请读者自行测试。

总结一下，函数调用时，一个星号可作用于所有的可迭代对象，称为迭代器解包操作，作为位置参数传递给函数，两个星号只能作用于字典对象，称之为字典解包操作，作为关键字参数传递给函数。使用 **\***和 **\*\*** 的解包的好处是能节省代码量，使得代码看起来更优雅，不然你得这样写：

>>> d = {"a":1, "b":2, "c":3}

>>> func(a = d['a'], b=d['b'], c=d['c'])

1 2 3

>>>

到这里，解包还没介绍完，因为 Python3.5，也就是 [PEP 448](https://www.python.org/dev/peps/pep-0448/) 对解包操作做了进一步的扩展， 在 3.5 之前的版本，函数调用时，一个函数中解包操作只允许一个**\*** 和 一个**\*\***。从 3.5 开始，在函数调用中，可以有任意多个解包操作，例如：

# Python 3.4 中 print 函数 不允许多个 \* 操作

>>> print(\*[1,2,3], \*[3,4])

File "<stdin>", line 1

print(\*[1,2,3], \*[3,4])

^

SyntaxError: invalid syntax

>>>

再来看看 python3.5以上版本

# 可以使用任意多个解包操作

>>> print(\*[1], \*[2], 3)

1 2 3

从 3.5 开始可以接受多个解包，于此同时，解包操作除了用在函数调用，还可以作用在表达式中。

>>> \*range(4), 4

(0, 1, 2, 3, 4)

>>> [\*range(4), 4]

[0, 1, 2, 3, 4]

>>> {\*range(4), 4}

{0, 1, 2, 3, 4}

>>> {'x': 1, \*\*{'y': 2}}

{'x': 1, 'y': 2}

新的语法使得我们的代码更加优雅了，例如拼接两个列表可以这样：

>>> list1 = [1,2,3]

>>> list2 = range(3,6)

>>> [\*list1, \*list3]

[1, 2, 3, 3, 4, 5]

>>>

可不可以直接用 **+** 操作呢？不行，因为 list 类型无法与 range 对象 相加，你必须先将 list2 强制转换为 list 对象才能做 **+**操作，这个留给读者自行验证。

再来看一个例子：如何优雅的合并两个字典

>>> a = {"a":1, "b":2}

>>> b = {"c":3, "d":4}

>>> {\*\*a, \*\*b}

{'a': 1, 'b': 2, 'c': 3, 'd': 4}

在3.5之前的版本，你不得不写更多的代码：

>>> import copy

>>>

>>> c = copy.deepcopy(a)

>>> c.update(b)

>>> c

{'a': 1, 'b': 2, 'c': 3, 'd': 4}

最后给你总结一下：

1. 自动解包支持一切可迭代对象
2. python3中，开始支持更高级的解包操作，用星号操作使得等号左边的变量个数可以少于右边迭代对象中元素的个数。
3. 函数调用时，可以用 \* 或者 \*\* 解包可迭代对象
4. python3.5，函数调用和表达式中可支持更多的解包参数。