# 第115天：Python 到底是值传递还是引用传递

原创 豆豆 Python技术 1月15日

我们平时写的 Python 程序中充斥着大量的函数，包括系统自带函数和自定义函数，当我们调用函数时直接将参数传递进去然后坐等接收返回值即可，简直不要太好用。那么你知道函数的参数是怎么传递的么，是值传递还是引用传递呢，什么又是值传递和引用传递呢？

这个问题对于很多初学者还是比较有难度的，看到这里你可以稍加停顿，自己思考一下，看看自己是否真正理解了。很多人只是知道概念但是让他说他又说不清楚，思考过后如果你还觉得模糊的话，往下仔细看，我今天就带着你深入剖析下函数的参数传递机制。

为了搞清楚函数的参数传递机制，你必须先彻底理解形参和实参。例如下面的 sayHello 函数，括号里面的 name 就是形参，而当调用函数时传递的 name 是实参。

```
def sayHello(name): # name 是形式参数
    print("Hello %s" % name)


name = "hanmeimei" # name 是实际参数
sayHello(name)

# 输出结果
Hello hanmeimei
```

## 值传递 OR 引用传递

上面我们说了，当调用函数时我们会把实际参数传递给形式参数。而这个传递过程有两种，就是我们上文说的值传递和引用传递了。

顾名思义，所谓值传递就是指在传递过程中将实际参数的值复制一份传递给形式参数，这样即使在函数执行过程中对形式参数进行了修改，形式参数也不会有所改变，因为二者互不干扰。而引用传递是值将实际参数的引用传递给实际参数，这样二者就会指向同一块内存地址，在函数执行过程中对形式参数进行了修改，形式参数也会一并=被修改。

为了故事的顺利发展，我们先来看看 Python 中关于变量的赋值。

```
>>> a = 10
>>> b = a
>>> a = a + 10
>>> a
20
>>> b
10
```

```
>>>
```

在上述的例子中，我们声明了一个变量 a，其值为 10，然后将 b 也指向 a，这是在内存中的布局是这样的，变量 a 和 b 会指向同一个对象 10，而不是给 b 重新生成一个新的对象。

由此可知，同一个对象是可以被多个对象引用的。

当执行完 a = a + 10 后，因为整数是不可变对象，所以并不会将 10 变成 20，而是生成一个新的对象 20 ，然后 a 会指向这个新的对象。b 还是指向旧对象 10。

所以，最后就是 a 为 20，而 b 为 10。

理解了上面的赋值过程之后，我们再来看看参数的传递。老规矩，还是直接看例子吧，代码是不会骗人的。

```
def swap(a, b):
    a, b = b, a
    print("in swap a = %d and b = %d " % (a, b))


a = 100
b = 200
swap(a, b)
print("in main a = %d and b = %d " % (a, b))

## 输出结果
in swap a = 200 and b = 100
in main a = 100 and b = 200
```

我们在函数 swap 中交换 a 和 b 的值，然后分别在主函数和 swap 函数中输出其结果，由结果可知，swap 函数并不会改变实际参数 a，b 的值，因此我们可以得出结论，Python 函数参数是按照值传递的。

别急，不妨再看多一个例子。

```
def swap(list):
    list.append(4)
    print("in swap list is %s " % list)


list_x = [1, 2, 3]
swap(list_x)
print("in main list is %s " % list_x)

## 输出结果
```

```
## 输出结果
in swap list is [1, 2, 3, 4]
in main list is [1, 2, 3, 4]
```

咦，值被改了，这不就是引用传递了么。于是，我们又得出结论，Python 函数参数是按照引用传递的。

这未免有点太不严谨了，事实上我们上面的第二个例子有点不太严谨，我稍微修改了下 swap 函数，咱们在看看测试结果。

```
def swap(list):
    list = list + [4]
    print("in swap list is %s " % list)

## 输出结果
in swap list is [1, 2, 3, 4]
in main list is [1, 2, 3]
```

我们只是更改了 swap 函数内一行代码，结果就完全不一样了。为了更好的理解其执行过程，我画了张图。

在第一个关于 list 的例子中，我们首先声明了一个列表，其中的元素为 [1,2,3]，此时其内存布局如上图中的步骤一所示。list_x 指向内存地址为 `OX7686934F` 的区域。

当调用 swap 函数将 list_x 传递给形式参数 list 时，会将该地址直接传递过去，list 也会指向这个地址，如步骤二所示。

最后，由于列表是可变的，所以当 list 在向列表中添加元素时，list_x 自然会受到影响，因为二者指向的是同一块内存。

所以，这里也是值传递，只不过传递的值是对象的内存地址罢了。

第二个关于 list 的例子中，我们对 swap 函数进行了修改，其执行流程如下图所示。

在执行 swap 函数之前都与上面的例子毫无差别。在 swap 函数内部 `list = list + [4]` 表示新建一个末尾加入元素 4 的新的列表，并让 list 指向这个新的内存地址 `OX7686936A`。因为是生成了一个新的对象，与原对象无关，所以 list_x 不受影响。

简而言之，弄清楚改变变量和重新赋值的区别就好了，第一个例子中我们改变了变量的值，所以当函数执行结束后所有指向该对象的变量都会受影响。而重新赋值相当于重新生成一个新的对象并在新的对象上做操作，因此旧对象不受影响。

如果我们要想在函数中改变对象，第一可以传入可变数据类型(列表，字典，集合)，直接改变；第二还可以创建一个新的对象，修改后返回。建议用后者，表达清晰明了，不易出错。

## 总结

本文介绍了 Python 函数的参数传递机制。理解了参数的传递过程和底层实现细节，写代码时将会少犯一些不必要的低级错误。

最后，无论是值传递还是引用传递，我们只需关注函数内部是否会生成新的对对象即可。**凡是对原对象操作的函数，都会影响传递的实际参数；凡是生成了新对象的操作，都不会影响传递的实际参数。**

## 代码地址

> 示例代码：https://github.com/JustDoPython/python-100-day/tree/master/day-115

## 系列文章

第 114 天：三木板模型算法项目实战
第 113 天：Python XGBoost 算法项目实战
第 112 天：机器学习算法之蒙特卡洛
第 111 天：Python 垃圾回收机制
从 0 学习 Python 0 - 110 大合集总结

**PS：** 公号内回复 ：Python，即可进入Python 新手学习交流群，一起**100天计划！**

-END-

**Python 技术**

**关于 Python 都在这里**