Pytho 全栈文档

第四章 内存管理

1、对象池

小整数池

系统默认创建好的，等着你使用

概述 整数在程序中的使用非常广泛，Python为了优化速度，使用了小整数对象池,避免为整数频繁申请和销毁内存空间。Python 对小整数的定义是 [-5, 256] ,这些整数对象是提前建立好的，不会被垃圾回收。在一个 Python 的程序中，无论这个整数处于LEGB(局部变量，闭包，全局，内建模块)中的哪个位置，所有位于这个范围内的整数使用的都是同一个对象。

```
a = 100
print(id(a))
del a
b = 100
print(id(b))
发现删除a后，b的地址依旧是删除之前的那个地址(是否删除，小整数都常驻内存)
```

大整数池 默认创建出来，池内为空的，创建一个就会往池中存储一个

```
案例验证
代码验证大整数
```

intern机制

每个单词(字符串)，不夹杂空格或者其他符号，默认开启intern机制，共享内存，靠引用计数决定是否销毁

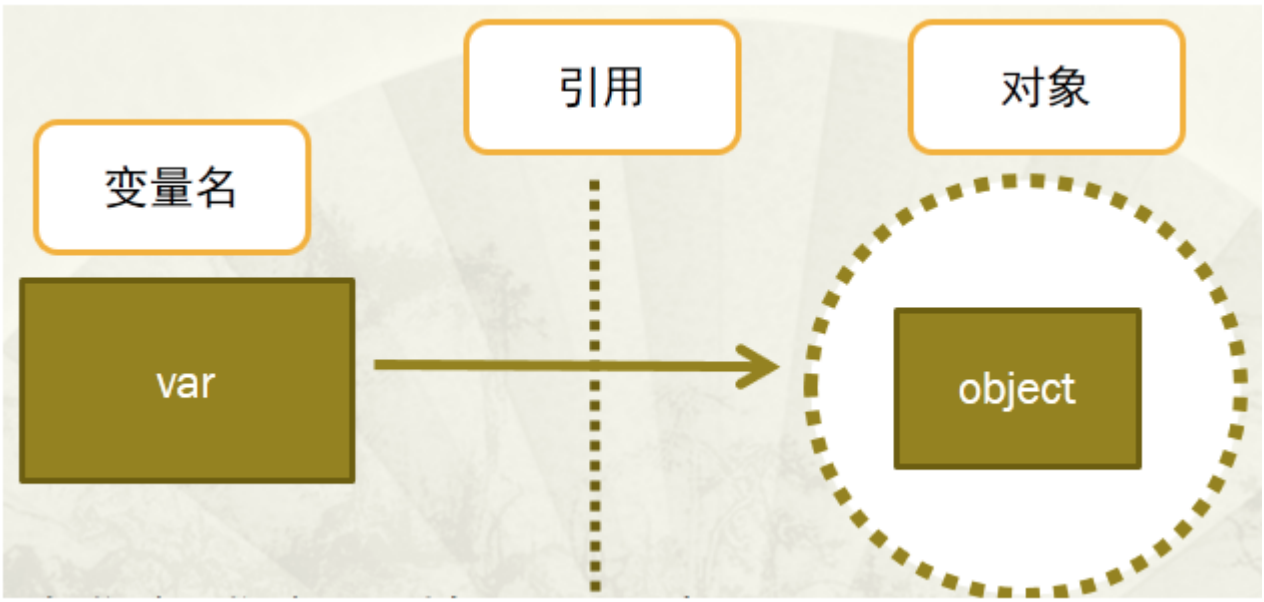
```
案例验证:
a = 'HelloWorld'
b = 'HelloWorld'
print(a is b)

a = 'Hello World'
b = 'Hello World'
print(a is b)
```

python中对大于256的整数，会重新分配对象空间地址保存对象；对于字符串来说，如果不包含空格的字符串，则不会重新分配对象空间，对于包含空格的字符串则会重新分配

2、垃圾收集

概述：
现在的高级语言如java，c# 等，都采用了垃圾收集机制，而不再是c，c++里用户自己管理维护内存的方式。自己管理内存极其自由，
可以任意申请内存，但如同一把双刃剑，为大量内存泄露，悬空指针等bug埋下隐患。
python里也同java一样采用了垃圾收集机制，不过不一样的是：
python采用的是引用计数机制为主，隔代回收机制为辅的策略

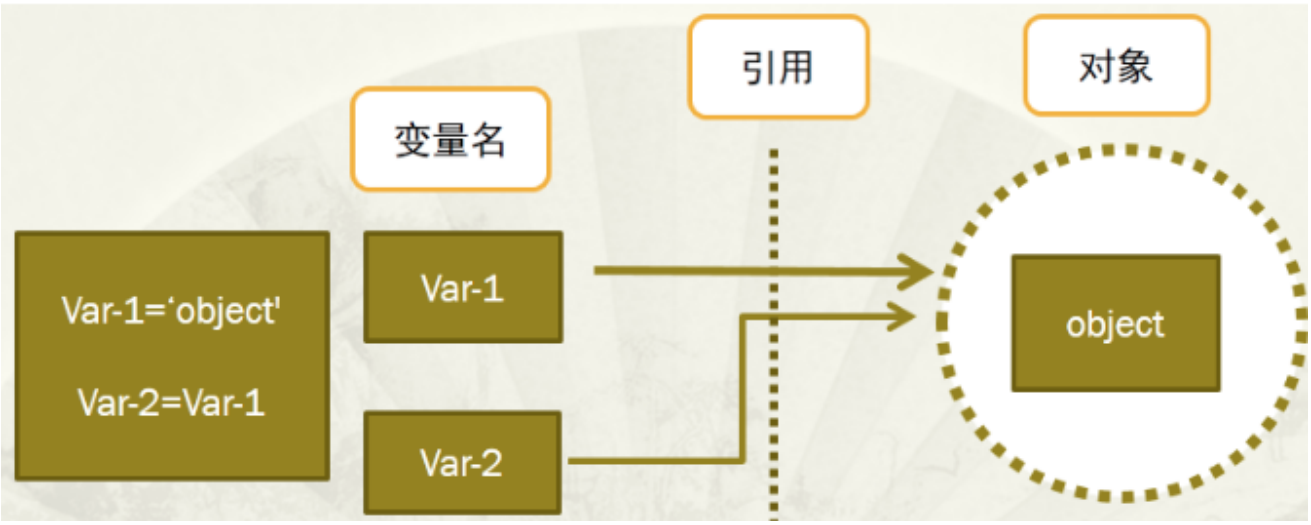
引用计数



在Python中，每个对象都有指向该对象的引用总数---引用计数

查看对象的引用计数：sys.getrefcount()

注意：
　　当使用某个引用作为参数，传递给getrefcount()时，参数实际上创建了一个临时的引用。因此，getrefcount()所得到的结果，会比期望的多1。



1、引用计数增加

a、对象被创建

b、另外变量也指向当前对象

c、作为容器对象的一个元素

d、作为参数提供给函数：test(x)

2、引用计数减少

a、变量被显式的销毁

b、对象的另外一个变量重新赋值

c、从容器中移除

d、函数被执行完毕

当Python的某个对象的引用计数降为0时，说明没有任何引用指向该对象，该对象就成为要被回收的垃圾。比如某个新建对象，被分配给某个引用，对象的引用计数变为1。如 为0，那么该对象就可以被垃圾回收。

3、隔代回收

> 隔代回收是用来解决交叉引用(循环引用),并增加数据回收的效率. 原理:通过对象存在的时间不同,采用不同的算法来回收垃圾. 形象的比喻,
> 三个链表,零代链表上的对象(新创建的对象都加入到零代链表),引用数都是一,每增加一个指针,引用加一,随后python会检测列表中的互相引用的对象,根据规则减掉其引用计数.
> GC算法对链表一的引用减一,引用为0的,清除,不为0的到链表二,链表二也执行GC算法,链表三一样. 存在时间越长的数据,越是有用的数据

隔代回收触发时机？（GC阈yu值）

> 随着你的程序运行，Python解释器保持对新创建的对象，以及因为引用计数为零而被释放掉的对象的追踪。
> 从理论上说，这两个值应该保持一致，因为程序新建的每个对象都应该最终被释放掉。当然，事实并非如此。因为循环引用的原因，从而被分配对象的计数值与被释放对象的计数值之间的差异在逐渐增长。一旦这个差异累计超过某个阈值，则Python的收集机制就启动了，并且触发上边所说到的零代算法，释放“浮动的垃圾”，并且将剩下的对象移动到一代列表。
> 随着时间的推移，程序所使用的对象逐渐从零代列表移动到一代列表。而Python对于一代列表中对象的处理遵循同样的方法，一旦被分配计数值
> 与被释放计数值累计到达一定阈值，Python会将剩下的活跃对象移动到二代列表。通过这种方法，你的代码所长期使用的对象，那些你的代码持续访问的活跃对象，会从零代链表转移到一代再转移到二代。通过不同的阈值设置，Python可以在
> 不同的时间间隔处理这些对象。
> Python处理零代最为频繁，其次是一代然后才是二代。

查看引用计数

gc模块的使用

```
# import gc
# 常用函数:
gc.get_count()
# 获取当前自动执行垃圾回收的计数器，返回一个长度为3的列表
```

```
gc.get_threshold()
# 获取gc模块中自动执行垃圾回收的频率

gc.set_threshold(threshold0[,threshold1,threshold2])
# 设置自动执行垃圾回收的频率

gc.disable()
# python3默认开启gc机制，可以使用该方法手动关闭gc机制

gc.collect()
# 手动调用垃圾回收机制回收垃圾
```

内存管理是使用计算机必不可少的一部分，无论好坏，Python几乎会在后台处理一切内存管理的问题。Python抽象出许多使用计算机的严格细节，这让我们在更高层次进行开发，而不用担心所有字节的存储方式和位置