

06. 万恶之源-再谈编码

本节主要内容:

1. 小数据池
2. is和==的区别
3. 编码的问题

一. 小数据池

在说小数据池之前. 我们先看一个概念. 什么是代码块:

根据提示我们从官方文档找到了这样的说法:

A Python program is constructed from code blocks. A block is a piece of Python program text that is executed as a unit. The following are blocks: a module, a function body, and a class definition. Each command typed interactively is a block. A script file (a file given as standard input to the interpreter or specified as a command line argument to the interpreter) is a code block. A script command (a command specified on the interpreter command line with the '-c' option) is a code block. The string argument passed to the built-in functions eval() and exec() is a code block. A code block is executed in an execution frame. A frame contains some administrative information (used for debugging) and determines where and how execution continues after the code block's execution has completed.

粗略的翻译:

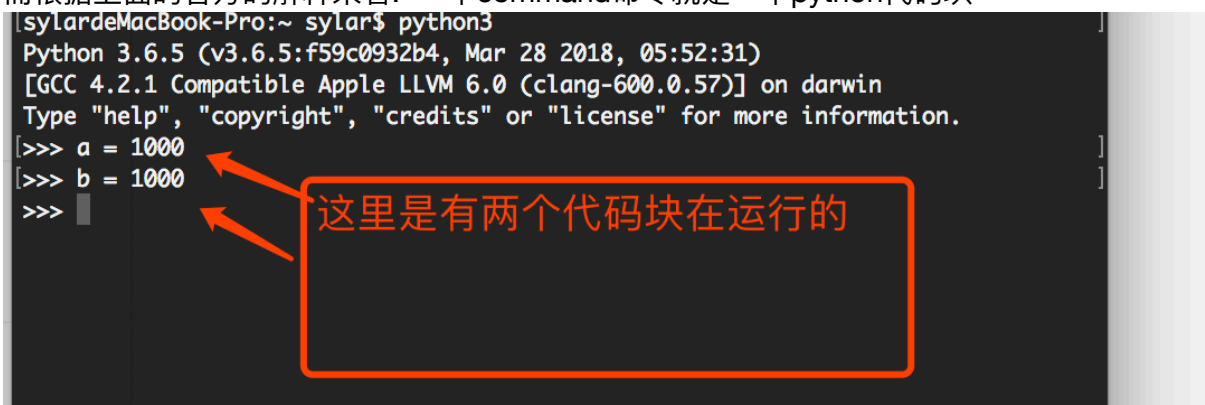
python程序是由代码块构成的. 一个代码块的文本作为python程序执行的单元.

代码块: 一个模块, 一个函数, 一个类, 甚至每一个command命令都是一个代码块. 一个文件也是一个代码块, eval()和exec()执行的时候也是一个代码块

什么是命令行?

我们在控制台CMD中输入python进入的就是python的command模式. 在这里也可以写python的程序.

而根据上面的官方的解释来看. 一个command命令就是一个python代码块



```
[sylardeMacBook-Pro:~ sylar$ python3
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 05:52:31)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> a = 1000
[>>> b = 1000
>>> ]
```

这里是有两个代码块在运行的

二. 接下来我们来看一下小数据池 is和==的区别

1. id()

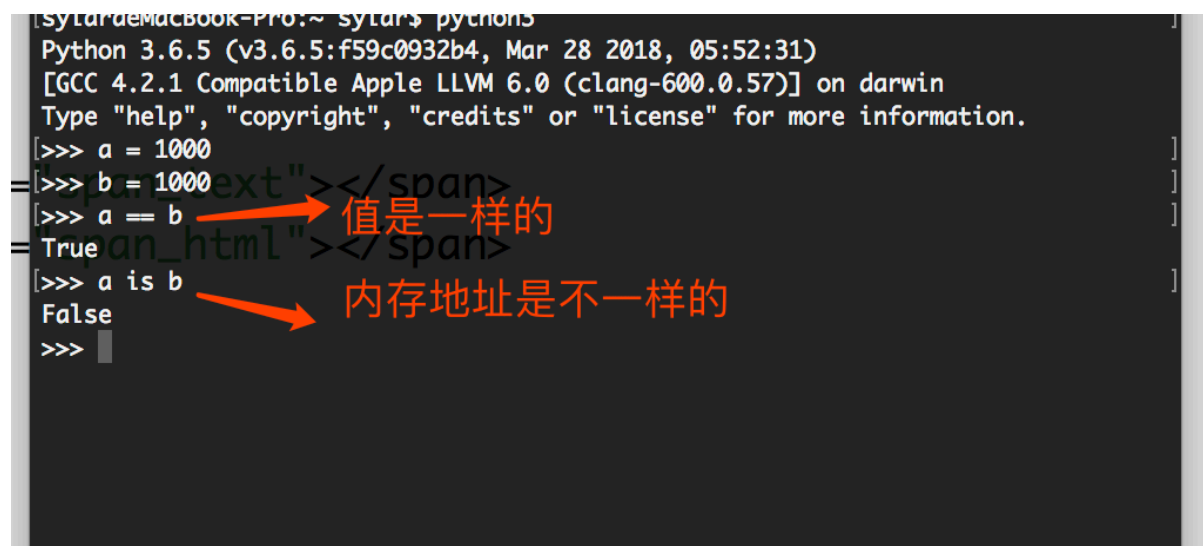
通过id()我们可以查看到一个变量表示的值在内存中的地址.

```
s = 'alex'
print(id(s))    # 4326667072
```

2. is和==

== 判断左右两端的值是否相等. 是不是一致.

is 判断左右两端内容的内存地址是否一致. 如果返回True, 那可以确定这两个变量使用的是同一个对象



```
[sylardemacbook-Pro:~ sylar$ python3
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 05:52:31)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> a = 1000
[>>> b = 1000
[>>> a == b
True
[>>> a is b
False
[>>>
```

值是一样的

内存地址是不一样的

我们可以这样认为. 如果内存地址相同. 那么值一定是相等的. 如果值相等. 则不一定是同一个对象

小数据池. 一种数据缓存机制. 也被称为驻留机制. 各大编程语言中都有类似的东西. 在网上搜索常量池,小数据池指的都是同一个内容.

小数据池只针对: **整数, 字符串, 布尔值**. 其他的数据类型不存在驻留机制

对于整数, Python官方文档中这么说:

The current implementation keeps an array of integer objects for all integers between -5 and 256, when you create an int in that range you actually just get back a reference to the existing object. So it should be possible to change the value of 1. I suspect the behaviour of Python in this case is undefined.

对于字符串:

In computer science, string interning is a method of storing only one copy of each distinct string value, which must be immutable. Interning strings makes some string processing tasks more time- or space-efficient at the cost of requiring more time when the string is created or interned. The distinct values are stored in a string intern pool. -引自维基百科

在python中对-5到256之间的整数会被驻留在内存中. 将一定规则的字符串缓存. 在使用的时候, 内存中只会创建一个该数据的对象. 保存在小数据池中. 当使用的时候直接从小数据池中获取对象的内存引用. 而不需要创建一个新的数据. 这样会节省更多的内存区域.

优点: 能够提高一些字符串, 整数的处理速度. 省略的创建对象的过程.

缺点: 在'池'中创建或者插入新的内容会花费更多的时间.

对于数字: -5~256是会被加到小数据池中的. 每次使用都是同一个对象.

对于字符串:

1. 如果字符串的长度是0或者1, 都会默认进行缓存
2. 字符串长度大于1, 但是字符串中只包含字母, 数字, 下划线时才会缓存
3. 用乘法的到的字符串. ①. 乘数为1, 仅包含数字, 字母, 下划线时会被缓存. 如果包含其他字符, 而长度 ≤ 1 也会被驻存, ②. 乘数大于1. 仅包含数字, 字母, 下划线这个时候会被缓存. 但字符串长度不能大于20
4. 指定驻留. 我们可以通过sys模块中的intern()函数来指定要驻留的内容.

OK. 到目前为止. 我们已经了解了python的小数据池的一些基本情况了. 但是!!!! 还有最后一个问题. 小数据池和最开始的代码块有什么关系呢?

同样的一段代码在命令行窗口和在py文件中. 出现的效果是完全不一样的.

```
a = 1000
b = 1000
print(a is b)
```

注意. 在py文件中. 得到的结果是True, 但是在command中就不是了.

在代码块内的缓存机制是不一样的. 在执行同一个代码块的初始化对象的命令时, 会检查是否其值是否已经存在, 如果存在, 会将其重用. 换句话说: 执行同一个代码块时, 遇到初始化对象的命令时, 他会将初始化的这个变量与值存储在一个字典中, 在遇到新的变量时, 会先在字典中查询记录, 如果有同样的记录那么它会重复使用这个字典中的之前的这个值. 所以在你给出的例子中, 文件执行时(同一个代码块) 会把a, b两个变量指向同一个对象.

如果是不同的代码块, 他就会看这个两个变量是否是满足小数据池的数据, 如果是满足小数据池的数据则会指向同一个地址. 所以: a, b的赋值语句分别被当作两个代码块执行, 但是他们不满足小数据池的数据所以会得到两个不同的对象, 因而is判断返回False.

三. 编码的补充

1. python2中默认使用的是ASCII码. 所以不支持中文. 如果需要在Python2中更改编码. 需要在文件的开始编写:

```
# -*- encoding:utf-8 -*-
```

2. python3中: 内存中使用的是unicode码.

编码回顾:

1. ASCII: 最早的编码. 里面有英文大写字母, 小写字母, 数字, 一些特殊字符. 没有中文, 8个01代码, 8个bit, 1个byte
2. GBK: 中文国标码, 里面包含了ASCII编码和中文常用编码. 16个bit, 2个byte
3. UNICODE: 万国码, 里面包含了全世界所有国家文字的编码. 32个bit, 4个byte, 包含了ASCII
4. UTF-8: 可变长度的万国码. 是unicode的一种实现. 最小字符占8位
 - 1.英文: 8bit 1byte
 - 2.欧洲文字:16bit 2byte
 - 3.中文:24bit 3byte

综上, 除了ASCII码以外, 其他信息不能直接转换.

在python3的内存中. 在程序运行阶段. 使用的是unicode编码. 因为unicode是万国码. 什么内容都可以进行显示. 那么在数据传输和存储的时候由于unicode比较浪费空间和资源. 需要把unicode转存成UTF-8或者GBK进行存储. 怎么转换呢. 在python中可以把文字信息进行编码. 编码之后的内容就可以进行传输了. 编码之后的数据是bytes类型的数据. 其实啊. 还是原来的数据只是经过编码之后表现形式发生了改变而已.

bytes的表现形式:

1. 英文 b'alex' 英文的表现形式和字符串没什么两样
2. 中文 b'\xe4\xb8\xad' 这是一个汉字的UTF-8的bytes表现形式

字符串在传输时转化成bytes=> encode(字符集)来完成

```
s = "alex"
print(s.encode("utf-8"))    # 将字符串编码成UTF-8
print(s.encode("GBK"))     # 将字符串编码成GBK
结果:
b'alex'
```

```

b'alex'

s = "中"
print(s.encode("UTF-8"))    # 中文编码成UTF-8
print(s.encode("GBK"))      # 中文编码成GBK

结果:
b'\xe4\xb8\xad'
b'\xd6\xd0'

```

记住: 英文编码之后的结果和源字符串一致. 中文编码之后的结果根据编码的不同. 编码结果也不同. 我们能看到. 一个中文的UTF-8编码是3个字节. 一个GBK的中文编码是2个字节. 编码之后的类型就是bytes类型. 在网络传输和存储的时候我们python是保存和存储的bytes类型. 那么在对方接收的时候. 也是接收的bytes类型的数据. 我们可以使用decode()来进行解码操作. 把bytes类型的数据还原回我们熟悉的字符串:

```

s = "我叫李嘉诚"
print(s.encode("utf-8"))    #
b'\xe6\x88\x91\xe5\x8f\xab\xe6\x9d\x8e\xe5\x98\x89\xe8\xaf\x9a'

print(b'\xe6\x88\x91\xe5\x8f\xab\xe6\x9d\x8e\xe5\x98\x89\xe8\xaf\x9a'.decode("utf-8")) # 解码

```

编码和解码的时候都需要制定编码格式.

```

s = "我是文字"
bs = s.encode("GBK")        # 我们这样可以获取到GBK的文字
# 把GBK转换成UTF-8
# 首先要把GBK转换成unicode. 也就是需要解码
s = bs.decode("GBK")        # 解码
# 然后需要进行重新编码成UTF-8
bss = s.encode("UTF-8")     # 重新编码
print(bss)

```