公告

昵称： 珞樱缤纷
园龄： 6年10个月
粉丝： 11
关注： 4
+加关注

< 2019年11月 >

日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

python(22)
Java(4)
Android(1)
编码(1)

随笔分类

Android(2)
Android——调试错误(2)
C(4)
Java SE(2)
Linux基础
python(22)
SpringMVC
日志(1)
算法

随笔档案

2017年11月(1)
2017年8月(1)
2017年7月(11)
2017年6月(6)
2017年5月(5)
2015年9月(1)
2014年4月(1)
2013年12月(3)
2013年11月(3)

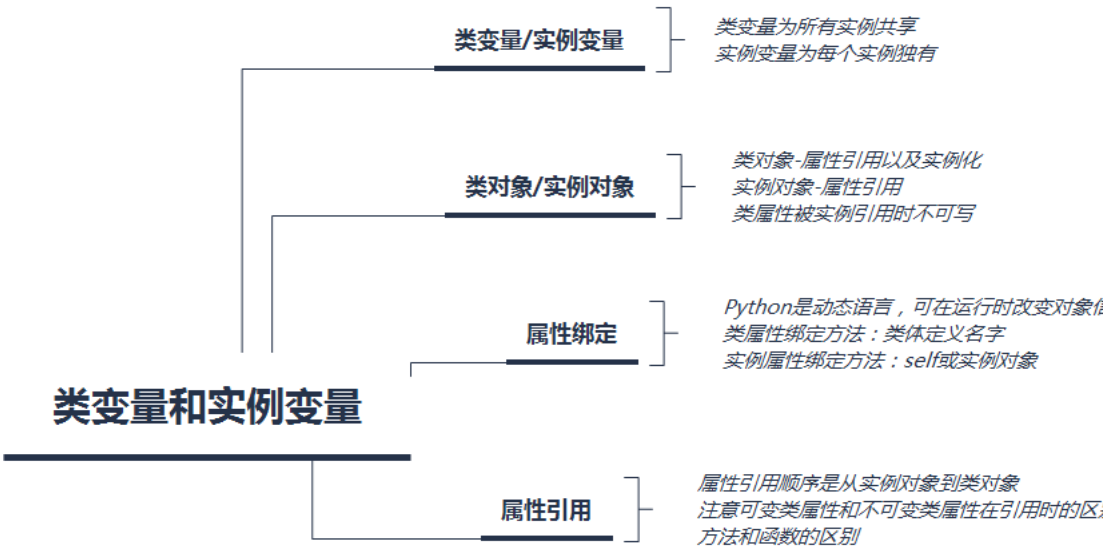
Python基础-类变量和实例变量

Python基础-类变量和实例变量

写在前面

> 如非特别说明，下文均基于Python3

大纲：



1. 类变量和实例变量

在Python Tutorial中对于类变量和实例变量是这样描述的：

> Generally speaking, instance variables are for data unique to each instance and class variables are for attributes and methods shared by all instances of the class:

通常来说，实例变量是对于每个实例都独有的数据，而类变量是该类所有实例共享的属性和方法。

其实我更愿意用类属性和实例属性来称呼它们，但是**变量**这个词已经成为程序语言的习惯称谓。一个正常的示例是：

```
class Dog:

    kind = 'canine'         # class variable shared by all instances

    def __init__(self, name):
        self.name = name    # instance variable unique to each instance
```

类 `Dog` 中，类属性 `kind` 为所有实例所共享；实例属性 `name` 为每个 `Dog` 的实例独有。

2. 类对象和实例对象

2.1 类对象

`Python` 中一切皆对象；类定义完成后，会在当前作用域中定义一个以类名为名字，指向类对象的名字。如

```
class Dog:
    pass
```

会在当前作用域定义名字 `Dog` ，指向类对象 `Dog` 。

类对象支持的操作:
总的来说，类对象仅支持两个操作：

1. 实例化；使用 `instance_name = class_name()` 的方式实例化，实例化操作创建该类的实例。
2. 属性引用；使用 `class_name.attr_name` 的方式引用类属性。

2.2 实例对象

实例对象是类对象实例化的产物，实例对象仅支持一个操作：

1. 属性引用；与类对象属性引用的方式相同，使用 `instance_name.attr_name` 的方式。

按照严格的面向对象思想，所有属性都应该是实例的，类属性不应该存在。那么在 `Python` 中，由于类属性绑定就不应该存在，类定义中就只剩下函数定义了。

在Python tutorial关于类定义也这么说：

> In practice, the statements inside a class definition will usually be function definitions, but other statements are allowed, and sometimes useful.

实践中，类定义中的语句通常是函数定义，但是其他语句也是允许的，有时也是有用的。

这里说的其他语句，就是指类属性的绑定语句。

3. 属性绑定

在定义类时，通常我们说的定义属性，其实是分为两个方面的：

1. 类属性绑定
2. 实例属性绑定

用**绑定**这个词更加确切；不管是类对象还是实例对象，属性都是依托对象而存在的。

我们说的属性绑定，首先需要一个可变对象，才能执行绑定操作，使用

```
objname.attr = attr_value
```

的方式，为对象 `objname` 绑定属性 `attr` 。

这分两种情况：

1. 若属性 `attr` 已经存在，绑定操作会将属性名指向新的对象；
2. 若不存在，则为该对象添加新的属性，后面就可以引用新增属性。

3.1 类属性绑定

`Python` 作为动态语言，类对象和实例对象都可以在运行时绑定任意属性。因此，类属性的绑定发生在两个地方：

1. 类定义时；
2. 运行时任意阶段。

下面这个例子说明了类属性绑定发生的时期：

```
class Dog:

    kind = 'canine'

Dog.country = 'China'

print(Dog.kind, ' - ', Dog.country) # output: canine  -  China
del Dog.kind
print(Dog.kind, ' - ', Dog.country) # AttributeError: type object 'Dog' has no attribute 'kind'
```

在类定义中，类属性的绑定并没有使用 `objname.attr = attr_value` 的方式，这是一个特例，其实是等同于后面使用类名绑定属性的方式。
因为是动态语言，所以可以在运行时增加属性，删除属性。

3.2 实例属性绑定

与类属性绑定相同，实例属性绑定也发生在两个地方：

1. 类定义时；
2. 运行时任意阶段。

示例：

```
class Dog:

    def __init__(self, name, age):
        self.name = name
        self.age = age

dog = Dog('Lily', 3)
dog.fur_color = 'red'

print('%s is %s years old, it has %s fur' % (dog.name, dog.age, dog.fur_color))
# Output: Lily is 3 years old, it has red fur
```

`Python` 类实例有两个特殊之处：

1. `__init__` 在实例化时执行
2. `Python` 实例调用方法时，会将实例对象作为第一个参数传递

因此， `__init__` 方法中的 `self` 就是实例对象本身，这里是 `dog` ，语句

```
self.name = name
self.age = age
```

以及后面的语句

```
dog.fur_color = 'red'
```

为实例 `dog` 增加三个属性 `name` ， `age` ， `fur_color` 。

4. 属性引用

属性的引用与直接访问名字不同，不涉及到作用域。

4.1 类属性引用

类属性的引用，肯定是需要类对象的，属性分为两种：

1. 数据属性
2. 函数属性

数据属性引用很简单，示例：

```
class Dog:

    kind = 'canine'

Dog.country = 'China'

print(Dog.kind, ' - ', Dog.country) # output: canine  -  China
```

通常很少有引用类函数属性的需求，示例：

```
class Dog:

    kind = 'canine'

    def tell_kind():
        print(Dog.kind)

Dog.tell_kind() # Output: canine
```

函数 `tell_kind` 在引用 `kind` 需要使用 `Dog.kind` 而不是直接使用 `kind` ，涉及到作用域，这一点在我的另一篇文章中有介绍：Python进阶 - 命名空间与作用域

4.2 实例属性引用

使用实例对象引用属性稍微复杂一些，因为实例对象可引用类属性以及实例属性。但是实例对象引用属性时遵循以下规则：

1. 总是先到实例对象中查找属性，再到类属性中查找属性；
2. 属性绑定语句总是为实例对象创建新属性，属性存在时，更新属性指向的对象。

4.2.1 数据属性引用

示例1：

```
class Dog:

    kind = 'canine'
    country = 'China'

    def __init__(self, name, age, country):
        self.name = name
        self.age = age
```

```
        self.country = country

dog = Dog('Lily', 3, 'Britain')
print(dog.name, dog.age, dog.kind, dog.country)

# output: Lily 3 canine Britain
```

类对象 `Dog` 与实例对象 `dog` 均有属性 `country` ，按照规则， `dog.country` 会引用到实例对象的属性；但实例对象 `dog` 没有属性 `kind` ，按照规则会引用类对象的属性。

示例2：

```
class Dog:

    kind = 'canine'
    country = 'China'

    def __init__(self, name, age, country):
        self.name = name
        self.age = age
        self.country = country

dog = Dog('Lily', 3, 'Britain')
print(dog.name, dog.age, dog.kind, dog.country) # Lily 3 canine Britain
print(dog.__dict__) # {'name': 'Lily', 'age': 3, 'country': 'Britain'}

dog.kind = 'feline'
print(dog.name, dog.age, dog.kind, dog.country) # Lily 3 feline Britain
print(dog.__dict__)
print(Dog.kind) # canine 没有改变类属性的指向
# {'name': 'Lily', 'age': 3, 'country': 'Britain', 'kind': 'feline'}
```

使用属性绑定语句 `dog.kind = 'feline'` ，按照规则，为实例对象 `dog` 增加了属性 `kind` ，后面使用 `dog.kind` 引用到实例对象的属性。

这里不要以为会改变类属性 `Dog.kind` 的指向，实则是为实例对象新增属性，可以使用查看 `__dict__` 的方式证明这一点。

示例3，可变类属性引用：

```
class Dog:

    tricks = []

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)

d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
print(d.tricks) # ['roll over', 'play dead']
```

语句 `self.tricks.append(trick)` 并不是属性绑定语句，因此还是在类属性上修改可变对象。

4.2.2 方法属性引用

与数据成员不同，类函数属性在实例对象中会变成方法属性。

先看一个示例：

```
class MethodTest:

    def inner_test(self):
        print('in class')

def outer_test():
    print('out of class')

mt = MethodTest()
mt.outer_test = outer_test

print(type(MethodTest.inner_test))  # <class 'function'>
print(type(mt.inner_test))          #<class 'method'>
print(type(mt.outer_test))          #<class 'function'>
```

可以看到，类函数属性在实例对象中变成了方法属性，但是并不是实例对象中所有的函数都是方法。

Python tutorial中这样介绍方法对象：

> When an instance attribute is referenced that isn’ t a data attribute, its class is searched. If the name denotes a valid class attribute that is a function object, a method object is created by packing (pointers to) the instance object and the function object just found together in an abstract object: this is the method object. When the method object is called with an argument list, a new argument list is constructed from the instance object and the argument list, and the function object is called with this new argument list.

引用非数据属性的实例属性时，会搜索它对应的类。如果名字是一个有效的函数对象，Python会将实例对象连同函数对象打包到一个抽象的对象中并且依据这个对象创建方法对象：这就是被调用的方法对象。当使用参数列表调用方法对象时，会使用实例对象以及原有参数列表构建新的参数列表，并且使用新的参数列表调用函数对象。

那么，实例对象只有在引用方法属性时，才会将自身作为第一个参数传递；调用实例对象的普通函数，则不会。
所以可以使用如下方式直接调用方法与函数：

```
mt.inner_test()
mt.outer_test()
```

除了方法与函数的区别，其引用与数据属性都是一样的

5. 最佳实践

虽然 `Python` 作为动态语言，支持在运行时绑定属性，但是从面向对象的角度来看，还是在定义类的时候将属性确定下来。

分类: python

标签: python

洛樱缤纷
关注 - 4
粉丝 - 11
+加关注

« 上一篇： Python进阶 - 命名空间与作用域
» 下一篇： Python基础-包与模块

posted on 2017-06-05 15:30 洛樱缤纷 阅读(26618) 评论(1) 编辑 收藏

发表评论

#1楼 2019-07-12 11:54 | 蓝月心语

4.2.2 方法属性引用 没有看明白。 ‘
print(type(MethodTest.inner_test)) # <class 'function'>
print(type(mt.inner_test)) #<class 'method'>
print(type(mt.outer_test)) #<class 'function'>’

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。

相关博文：

- 9.面向对象：类和对象、实例变量、类变量
- Python基础-类
- Python类变量和实例变量区别
- 类变量、实例变量--python
- python中类变量和实例变量

» 更多推荐...