文章标题

阿左 [1]　　Nobody [2]

May 18, 2012

[1]感谢读者
[2]感谢国家

Contents

List of Figures

List of Tables

Part I

语法基础

Chapter 1

OOP 与类

一个模块只能有一个实例，当一个模块的代码被修改以后必须重新加载模块才能生效；类可以同时创建多个实例（即对象）。

1.1 最简单的 Python 类

Python 中可以仅用 pass 作为占位语句生成一个没有任何成员的 class，这样的 class 仅仅是一个空的命名空间对象。可以在以后通过赋值给这个类加上新的成员：

```
1  # coding=utf-8
2
3  class C1: pass              # Empty class
4                              # as an namespace object
5
6  a = C1()
7  a.name = "morgan"           # a.name is "morgan"
8
9  C1.name = "Class C1"        # C1.name is "C1"
10 b = C1()                    # b.name is "C1"
11                             # a.name still is "morgan"
```

Listing 1.1: 类与实例

1.2 类与实例的基本概念

类与类的实例都是对象。类是一个对象，该类的每个实例又各对应了一个关联到该类对象的实例对象。

class 语句会创建一个类对象并赋值给变量名。class 语句块内的语句会创建数据对象和方法对象赋值给类的成员变量。类的成员只属于该类，不属于该类的实例。

像调用函数一样调用类对象会创建该类的实例对象。每个实例对象根据具体类的创建属性并获得自己的命名空间，每个成员都有自己的实例。

通过实例调用方法时，会把这个实例对象作为第一个参数 self 传递给方法。

实例对象的__class__ 属性记录了这个实例的类对象。

```
1  # coding=utf-8
2
3  class c1():
4    count = 0
5    def show(self):
6      print 'c1.show()'
7
8  a = c1()
9  b = c1()
10
11 print a.__class__           # class of instance
12
13 c1.count = 10               # c1.count is 10
14 a.count = 5                 # a.count is 5
15 b.count = 6                 # b.count is 6
16
17 a.show()
18 b.show()
```

Listing 1.2: 类与实例

1.3 类的属性与方法

python 中把数据保存在对象中，相关的操作（方法）在类中。对于类和对象，无论数据还是方法都作为变量处理。

对象的数据成员（无论是数据还是函数）在没有被第一次赋值之前，都不能被访问（就像是没有被声明的变量一样）。同样地也可以简章地通过赋值给变量增加一个成员。

类对象与实例对象的__dict__ 属性是大多数基于类的对象的命名空间字典。

```
1  # coding=utf-8
2
3  class Employee():
4    def __init__(self, name):   # __init__ 声明构造函数
5      self.name = name
6    def showName(self):
7      print self.name
8
9  morgan = Employee("Morgan")
10 morgan.showName()
11
12 morgan.showName = "new name"     # 可以把变量赋值给原来是方法的成员
13 morgan.nickName = "Jade"         # 通过赋值就可以给对象创建一个新属性
14
15 print Employee.__dict__.keys()  # 大多数基于类的对象的命名空间字典
16 print morgan.__dict__.keys()    # 大多数基于类的对象的命名空间字典
```

Listing 1.3: 类的属性与方法

1.4 类的继承

子类会继承父类的成员。如果当多重继承时遇到重名成员，优先级按声明继承时从左到右顺序。

类对象的成员__bases__ 是超类构成的元组

```
1  # coding=utf-8
2
```

```
3  class c1():
4    myname = 'aaa'
5    def show(self):
6      print 'c1.show()'
7    def helloC1(self):
8      print 'c1.helloC1()'
9
10 class c2():
11   def show(self):
12     print 'c2.show()'
13   def helloC2(self):
14     print 'c2.helloC2()'
15
16 class c3(c1, c2):                # 超类写在括号中，支持多重继承
17   def show(self):
18     print 'c3.show()'
19
20 print c3.__bases__               # 超类的元组
21
22 c1.myname                        # c1.myname is 'aaa'
23 c3.myname                        # c3.myname is 'aaa'
24
25 i1 = c3()                        # i1.myname is 'aaa'
26 i2 = c3()                        # i2.myname is 'aaa'
27
28 i1.myname = 'this is i1'         # i1.myname is 'this is i1'
29 i2.myname = 'this is i2'         # i2.myname is 'this is i2'
30 # 对象成员赋值后不影响类成员的值
31 c1.myname                        # c1.myname still is 'aaa'
32 c3.myname                        # c3.myname still is 'aaa'
33
34 i1.show()                        # 调用重写的方法
35 i1.helloC1()                     # 调用到父类的方法
36 i1.helloC2()                     # 调用到父类的方法
```

Listing 1.4: 类继承

1.5 运算符重载

两头是下划线的方法名（__ 方法名 __）表示对运算符的重载。如：

__add__ 表示重载+ 运算。

__mul__ 表示重载* 运算。

对于没有定义或是继承的操作符，表示该操作不被支持。

```
1  # coding=utf-8
2
3  class C1():
4    def __init__(self, value):
5      self.data = value
6    def __add__(self, other):
7      return C1(self.data + other)
8    def __mul__(self, other):
9      return C1(self.data * other)
10
11 a = C1(5)
12 b = C1(10)
13
14 print (a + 3).data           # result is 8
15 print (b * 5).data           # result is 50
```

Listing 1.5: 运算符重载