Python基础教程

第6章 类和面向对象

6.1 OOP简介

OOP（Object-oriented programming，面向对象编程）是一种计算机编程架构。OOP 的一条基本原则是计算机程序是由单个能够起到子程序作用的单元或对象组合而成。

6.1 OOP简介

传统的面向过程程序设计特点：

（1）把完成某一个需求的所有步骤从头到尾逐步实现；

（2）将某些功能独立的代码封装成一个又一个函数；

（3）最后完成的代码，就是顺序地调用不同的函数。

缺点：

- 注重步骤与过程，不注重职责分工！
- 如果需求复杂，代码会变得很复杂！
- 开发复杂项目，没有固定的套路，开发难度很大！

6.1 OOP简介

面向对象的程序设计完全不同于传统的面向过程程序设计，大大地降低了软件开发的难度。

OOP 达到了软件工程的三个主要目标：

（1）重用性

（2）灵活性

（3）扩展性

为了实现整体运算，每个对象都能够接收信息、处理数据和向其它对象发送信息。

6.1 OOP简介

- 相比较函数，面向对象是更大的封装，根据职责在一个对象中封装多个方法：
  - 在完成某一个需求前，首先确定职责——要做的事情（方法）。
  - 根据职责确定不同的对象，在对象内部封装不同的方法（多个）。
  - 最后完成的代码，就是顺序地让不同的对象调用不同的方法。
- 特点：
  - 注重对象和职责，不同的对象承担不同的职责。
  - 更加适合应对复杂的需求变化，是专门应对复杂项目开发，提供的固定套路。
  - 需要在面向过程基础上，再学习一些面向对象的语法。

6.1 OOP简介

所谓“对象”，在显式支持面向对象的语言中，一般是指类在内存中装载的实例，具有相关的成员变量和成员函数（也称为：方法）。

6.2 类的概念

通过类的创建和继承，可重用代码，减少代码复杂度。

类是一种数据结构，可以用来定义对象，而对象把数据值和行为特性融合在一起。

类是现实世界的抽象的实体以编程形式出现，而实例是这些对象的具体化。

类与实例相互关联：

（1）类是对象的定义；

（2）实例是“真正的实物”，存放了类中定义的对象的具体信息。

- ***Everything in Python is an object*.**
  - ***Strings are objects.***
  - ***Lists are objects.***
  - ***Functions are objects.***
  - ***Even modules are objects.***

```
>>> v = 1
>>> type(v)
<class 'int'>
>>> help(v)
```

Squeezed text (242 lines).

```
>>> type(max)
<class 'builtin_function_or_method'>
>>>
```

6.3 类的创建

Python中使用class关键字来创建类：

```
class  ClassName ( bases ) :
    'class documentation string' # 类文档字符串
    class_suite                  # 类体
```

参数bases可以是一个（单继承）或者多个（多继承）用于继承的父类。如果没有指定bases，那么object将作为默认的父类。

object是“所有类之母”，它位于所有类继承结构的最上层。

6.3.1 类的创建-方法

Python中，方法定义在类定义中，但只能被实例对象所调用：

```
class  ClassName ( bases ):
    'class documentation string'        # 类文档字符串
    def  printFoo ( self ):             # 类体
        print ( 'You invoked printFoo( )!' )
```

参数self代表实例对象本身，当实例调用方法时，由解释器自动隐式传递给方法，类似于其他语言中的“this”。

```
>>> myObj = ClassName( )
>>> myObj.printFoo( )
You invoked printFoo( )!
```

6.3.2 类的创建-类定义（1）

```
class  AddrBookEntry(object):
    'address book entry class'       # 类文档字符串
    def  __init__(self , nm , ph) :  #实例构造函数
        self.name  =  nm             # 设置name
        self.phone  =  ph            # 设置phone
        print ( 'Created  instance  for :',  self.name)
    def  updatePhone(self ,  newph):
        self.phone  =  newph
        print ("Updated  phone #  for:",self.name)
```

6.3.2 类的创建-类定义（2）

在AddrBookEntry类的定义中，定义了两个方法：

（1）*__init__(self, nm, ph)*：

```
self.name = nm
self.phone = ph
print ('Created instance for:', self.name)
```

这两个属性对程序员可见

在实例化时（在AddrBookEntry()被调用时）被隐式调用，在C++里面称为“构造函数”。

（2）*updatePhone(self, newph)*：

```
self.phone = newph
print ('Updated phone # for:', self.name)
```

6.3.3 类的创建-实例化

```
>>> john = AddrBookEntry('John  Doe', '408-555-1212') #创建实例
>>> jane = AddrBookEntry('Jane  Doe', '650-555-1212') #创建实例
>>> john
<__main__.AddrBookEntry instance at 82ee610>
>>> john.name
'John Doe'
>>> john.phone
'408-555-1212'
>>> jane.name
'Jane Doe'
```

6.3.6 类的创建-通过实例进行方法调用

```
>>> john.update('415-555-1212')
>>> john.phone
'415-555-1212'
```

6.4 类属性

属性就是属于一个对象的数据或者函数元素，可以通过句点属性标识法来访问。

类属性仅与其被定义的类相绑定，实例对象在日常OOP中用的最多，实例数据属性是用到的主要数据属性。

（1）数据属性仅仅时所定义的类的变量，可以像任何其他变量一样在类创建后被使用，要么由类中的方法来更新，要么在主程序其他地方被更新。

（2）类里面定义的方法（函数），只能通过实例来调用，而不能通过其他方式调用！

6.4.1 类属性实例-类数据属性

```
1  class Test(ob
2      'class to
3      foo = 100
4
5  print Test.foo
6  Test.foo += 1(
7  print Test.foo
```

```
>>> class Test( object ):
        'class to test' #class documentation string
        foo = 100

>>> print( Test.foo )
>>> class Test( object ):
        'class to test' #class documentation string
        foo = 100
```

- 没有对类Test进行任何实
- 没有任何类实例的引用，

```
>>> Test.foo += 100
>>> myTest = Test( )
>>> print ( myTest.foo )
200
>>> myTest.foo += 100
>>> print ( myTest.foo )
300
>>> print ( Test.foo )
200
```

类数据属性仅当需要有 ，
关，表示这些数据是与它们 ，
员变量通常仅用来跟踪与类 ，
属性。

相当于C+

6.4.1 类属性实例-类数据属性

```
>>> class Test( object ):
        'class to test' #class documentation string
        foo = 100


>>> Test.foo += 100
>>> myTest = Test( )
>>> print ( myTest.foo )
200
>>> myTest.foo += 100
>>> print ( myTest.foo )
300
>>> print ( Test.foo )
200
```

```
class Myclass:
    'My first testing class'
    ver = 1                    #类属性，类变量
```

- help(Myclass)
- Myclass.ver #类是一个名字空间
- x = Myclass( ) #创建实例
- x
- dir(x)
- x.a = 1 #实例属性，实例也是一个名字空间
- x.b = 2 #可以随时创建新成员变量
- x.ver = 2
- Myclass.ver

```
>>> class Myclass:
        'My first testing class'
        ver = 1    #类属性，类变量


>>> help( Myclass )
Squeezed text (17 lines).
>>> Myclass.ver
1
>>> x = Myclass( )
>>> x
<__main__.Myclass object at 0x03B3C510>
>>> dir ( x )
Squeezed text (3 lines).
>>> x.a = 1
>>> x.b = 2
>>> x.ver =2
>>> Myclass.ver
1
```

```
class Myclass2:
    ver = 2
    def test( self ):              #类的方法
#参数self代表实例本身（实例对象），当实例调用方法时，
#由解释器自动隐式传递给方法，是必须的。
        print( 'Hello' )
```

- `x = Myclass2( )`
- `x. test( )`　　#必须通过实例对象调用
- `Myclass2.test( )`

```
>>> class Myclass2:
        ver = 2
        def test( self ):
            print ( 'Hello' )


>>> x = Myclass2( )
>>> x.test( )
Hello
>>> Myclass2.test( )
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    Myclass2.test( )
TypeError: test() missing 1 required positional argument: 'self'
```

6.4.4 特殊的类属性

对于任何类C，都有如下所有的属性：

C.__name__	类C的名字（字符串）
C.__doc__	类C的文档字符串
C.__bases__	类C的父类
C.__dict__	类或实例C的属性
C.__module__	类C定义所在的模块
C.__class__	实例C对应的类

注意__name__是给定类的字符名字。它适用于那种只需要类对象名字，而非类对象本身的情况。

甚至一些内建的类型也有这个属性。

```
>>> stype = type ( 'test string' )
>>> stype
<class 'str'>
>>> stype.__name__
'str'
```

6.4.5 类的属性查看（1）

要知道一个类的属性，有两种方法：内建函数dir()、类的字典属性__dict__。

```
>>> class MyClass( object ):
        'MyClass class definiton'
        myVersion = '1.0.0'
        def showVersion( self ):
                print ( MyClass.myVersion )


>>> dir( MyClass )
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '_
_module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '
__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'myVersion', 'showVersion']
>>> print (MyClass.__dict__)
{'__module__': '__main__', '__doc__': 'MyClass class definiton', 'myVersion': '1.0.0', 'showV
ersion': <function MyClass.showVersion at 0x02DEC660>, '__dict__': <attribute '__dict__'
of 'MyClass' objects>, '__weakref__': <attribute '__weakref__' of 'MyClass' objects>}
>>> MyClass.showVersion
<function MyClass.showVersion at 0x02DEC660>
>>> MyClass.myVersion
'1.0.0'
```

6.4.5 类的属性查看（2）

dir()返回的属性的一个名字列表，而__dict__返回的是一个字典，它的键名是属性名，键值是相应的属性对象的数据值。

访问一个类属性的时候，Python解释器将会搜索字典以得到需要的属性。

（1）如果在__dict__中没有找到，将会在基类的字典中进行搜索，采用“深度优先”的顺序。

（2）基类集的搜索是按顺序的，从左到右，按其在定义类的时候，定义父类参数时的顺序。

（3）对类的修改会影响到此类的字典；基类的__dict__属性不会被改动。

6.5 实例-初始化：通过调用类对象来创建实例

很多其他的OOP语言都提供关键字new，使用new关键字来创建类的实例。

Python的方式更加简单，一旦定义了一个类，可以使用函数操作符来创建实例。

```
class MyClass (object):          #定义类
    pass
Mc = MyClass( )                  #初始化类创建实例
```

6.5.1 实例-__init( )__构造器方法（1）

当类被调用，实例化的第一步就是创建实例对象。

一旦对象创建了，Python会检查是否实现了__init__( )方法（类似其他语言的构造函数）：

（1）如果没有定义（覆盖）特殊方法__init__( )，对实例不会施加任何特别的操作。

（2）如果需要任何特殊的操作，请实现__init__( )，覆盖其默认行为，调用类时，传进去的任何参数都交给了__init__( )，即创建实例的调用就是对构造器的调用。

6.5.1 实例-__init( )__构造器方法（2）

Python如何创建对象：

（a）解释器调用__new__( )创建类实例

（b）解释器调用__init__( )初始化实例。

6.5.2 实例-__new()__构造器方法

（1）与__init__()相比，__new__()更像一个真正的构造器；

（2）可以对内建类型进行派生，可以调用类的__new__()方法来实例化不可变对象，比如派生字符串、数字等。这是一个静态方法，传入的参数是在类实例化操作时生成的；

（3）__new__()必须返回一个合法的实例，解释器在调用__init__()时，就可以把这个实例作为self传给它。调用父类的__new__()来创建对象，就像其他语言中使用new关键字一样；

（4）__new__()和__init__()在类创建时，都传入了相同的参数。

6.5.2 实例-__new()__构造器方法

```
class New_Test:
    def __new__( cls ):
        print('__new__ is running')
        return super().__new__( cls )
    def __init__( self, age = 20 ):
        print('__init__ is running')
        self.age=age
New_Test( )
```

- __init__方法一般用于初始化一个类，但是当实例化一个类的时候，__init__并不是第一个被调用的，第一个被调用的是__new__；
- __new__方法是创建类实例的方法，创建对象时调用，返回当前对象的一个实例；
- __init__方法是类实例创建之后调用，对当前对象的实例的一些初始化，没有返回值。

6.5.2 实例-__new()__构造器方法

总结：

- __new__至少要有一个参数cls，代表要实例化的类，此参数在实例化时由Python解释器自动提供；
- __new__必须要有返回值，返回实例化出来的实例，可以return父类__new__出来的实例；
- __init__有一个参数self，就是这个__new__返回的实例，__init__在__new__的基础上可以完成一些其它初始化的动作，__init__不需要也不能有非None的返回值；
- 若__new__没有正确返回当前类cls的实例，那__init__是不会被调用的，即使是父类的实例也不行。

6.6 子类

- 继承是用来创建新的类的一种方式，好处是可以减少重复代码；
- 继承：使用一个已经定义好的类作为基类，利用继承机制，创建新类，扩展它或者对其修改；
- 靠继承来进行子类化是创建和定制新类类型的一种方式，新的类将保存已存在类所有的特性，对新类的改动不会影响到原来的类（父类、基类）；
- 对于子类，可以定制属于它的特定功能。

6.6.1 子类和派生-创建子类

语法：关键字后面紧跟一个类名以及一个或者多个父类：

class SubClassName（ParentClass1 [，ParentClass2, ParentClass3]):

‘optional class documentation string’

class_suite

所有类都有父类，默认为object。

6.6.1 创建子类举例（6.3.2）

```
class   EmplAddrBookEntry ( AddrBookEntry ):
    'Employee Address Book Entry class'        #员工地址簿类
    def  __init__( self,  nm,  ph,  id,  em ):
            AddrbookEntry.__init__( self,  nm,  ph )
            self.empid   =  id
            self.email   =  em
    def  __updateEmail( self,  newem )
            self.email  =  newem
            print   ('Updated e-mail address for:',  self.name)
```

6.7 继承（1）

继承描述了基类属性如何“遗传”给派生类。一个子类可以继承它的基类的任何属性，不管是属性还是方法。

```
>>> class P( object ):
        'parent docstring'
        parent_int = 100
        def parent_func(self):
            print （'parent function'）
>>> class C( P ):
    pass
>>> print（P.__dict__）
    {'__module__': '__main__', 'parent_int': 100, 'parent_func': <function parent_func at 0x029C58F0>, '__dict__': <attribute '__dict__' of 'P' objects>, '__weakref__': <attribute '__weakref__' of 'P' objects>, '__doc__': 'parent docstring'}
```

6.7 继承（2）

```
>>> c = C( )
>>> c.__class__
<class '__main__.C'>

>>> C.__bases__        #查询父类是谁
(<class '__main__.P'>,)

>>> c.parent_func( )  #继承父类成员方法
parent function

>>> c.parent_int       #继承父类成员变量
100
```

6.7 继承（3）

可以在子类中定义与父类中一样的方法，这样就可以覆盖父类的方法，使得子类具有特定或者不同的功能：

```
>>> class P(object):
        def foo(self):
            print('parent')

>>> class C( P ):
        def foo(self):
            print('child')


>>> p = P( )
>>> p.foo( )
parent
>>> c = C( )
>>> c.foo( )
child
```

Python的私有成员变量

- Python没有真正的私有成员。内部实现上，是将私有成员进行混淆。

```
>>> class Test:
        def __init__( self ):
            self.__zzz = 111
            print ( self.__zzz )
```

- *class Test:*

```
    def __init__( self ):
        self.__zzz = 111
```

```
>>> test = Test( )
111
```

思考

1.Python面向对象编程的特点?
2. 类及实例所对应的属性有何特点?
3.掌握子类及派生的设计思想!
4.掌握类内建函数用法。