第十三章：面对对象编程

目录

[13.1类和对象 1](#_Toc350841842)

[1、类的定义 1](#_Toc350841843)

[2、对象的创建 1](#_Toc350841844)

[3、python对象的体系结构 2](#_Toc350841845)

[13.2属性和方法 2](#_Toc350841846)

[1、类的属性 2](#_Toc350841847)

[2、类的方法 3](#_Toc350841848)

[3、内部类的使用 4](#_Toc350841849)

[4、\_\_init\_\_()方法 5](#_Toc350841850)

[5、\_\_del\_\_()方法 5](#_Toc350841851)

[6、垃圾回收机制 5](#_Toc350841852)

[7、类的内置方法 5](#_Toc350841853)

[13.3继承 6](#_Toc350841854)

[13.4命名规则 7](#_Toc350841855)

## 13.1类和对象

### 1、类的定义

class MyNewObjectType(bases):

'define MyNewObjectType class'

class\_suite #类体

关键字是class，紧接着是一个类名。随后是定义类的类体代码。这里通常由各种各样的定义和声明组成。新式类和经典类声明的最大不同在于，所有新式类必须继承至少一个父类，参数bases可以是一个（单继承）或多个（多重继承）用于继承的父类。object 是“所有类之母”。如果你的类没有继承任何其他父类，object 将作为默认的父类。它位于所有类继承结构的最上层。如果你没有直接或间接的子类化一个对象，那么你就定义了一个经典类：

class MyNewObjectType:

'define MyNewObjectType classic class'

class\_suite

如果你没有指定一个父类，或者如果所子类化的基本类没有父类，你这样就是创建了一个经典类。很多Python 类都还是经典类。即使经典类已经过时了，在以后的Python 版本中，仍然可以使用它们。不过我们强烈推荐你尽可能使用新式类，尽管对于学习来说，两者都行。

### 2、对象的创建

创建一个实例的过程称作实例化，过程如下（注意：没有使用new 关键字）：

myFirstObject = MyNewObjectType()

类名使用我们所熟悉的函数操作符（（）），以“函数调用”的形式出现。然后你通常会把这个新建的实例赋给一个变量。赋值在语法上不是必须的，但如果你没有把这个实例保存到一个变量中，它就没用了，会被自动垃圾收集器回收，因为没有任何引用指向这个实例。这样，你刚刚所做的一切，就是为那个实例分配了一块内存，随即又释放了它。

### 3、python对象的体系结构

Python2.5版中的对象分为经典对象和新型对象，在python2.1及其之前的版本中，只能使用经典对象来实现面向对象的程序设计。由于python中提供了内置类型（例如元组、列表、字典等数据结构）。Python2.5的新型对象已经十分的完善了。Python中的任何类型都是对象，包括字符串类型、数字类型、内置数据结构等。因此类和类型之间的界限变得模糊了：类和类型都属于对象。

1. 经典对象

经典类本质上是一个类**对象**, 注意, 这里是对象, 而新式类是一种类型的定义, 通过调用repr(class\_name)可以明确的发现, 经典类返回的是<class module\_name.class\_name at 0x00B98CC0>, 而新式类返回的是<class ‘module\_name.class\_name’>, 通过调用type(class\_name) 也可以看到, 经典类返回的是<type ‘classobj’>, 而新式类返回的是<type ‘type’>, 也就是说, 新式类将class和type进行了统一.

2）新型对象

新型对象提供了对类方法和静态方法的支持，类方法可以被类和类的实例化对象调用。Classmethod()可以把方法定义为类方法。静态方法可以被类直接调用，可以被所有的实例化对象共享。Staticmethod()可以把方法定义为静态方法。

新式类必须继承至少一个父类, 类定义时的参数bases可以是一个(单继承)或多个(多重继承)用于继承的父类

## 13.2属性和方法

### 1、类的属性

类的属性分为私有属性和公有属性。Python没有提供公有和私有权限的修饰符。类的私有属性不能被该类之外的函数调用。如果要修改类的私有属性或者获得其值，可以通过使用类提供的方法来完成。

如果函数、方法或属性的名字以两个下划线开始，则表示私有类型；没有使用两个下划线开始则表示公有属性。类的方法同样遵循这个约定。Python的许多语法是通过约定变量名称的方式实现的。

注意：python没有保护类型的修饰符。

Python的属性分为实例属性和静态属性。实例属性是以self作为前缀的属性。

其他语言中的静态变量在python中成为静态属性。静态变量，即可以被类直接访问，而不被实例化对象访问。

类提供了一些内置的属性，用于管理类的内部关系。例如，\_\_dict\_\_,\_\_base\_\_,\_\_doc\_\_等。

注意：

Python中对类的属性和方法的定义次序没有要求，合理的方式是把类的属性定义在类中的最前面，然后再定义私有方法，最后定义公有方法。

### 2、类的方法

1)类的方法也分为公有方法和私有方法。和类的公有属性、私有属性类似。

2)其他语言中的静态方法是用保留字static声明。但是python使用函数staticmethod()或@ staticmethod指令的方式把普通的函数转换成为静态方法。

3)Python还有一种方法称之为类方法。类方法的作用和静态方法相似，都可以被其他实例对象共享。不同的是类方法必须提供self参数。类方法可以使用函数classmethod()或@ classmethod指令定义

关于self的问题：

调用静态方法的时候, 不会传入调用者自身,不需要参数self。

一般的方法必须使用self

类方法：类方法需要类而不是实例。而对于classmethod，它的第一个参数不是self，是cls，它表示这个类本身。

5)方法就是类内部定义的函数. 方法只有在其所属的类拥有实例时, 才能被调用, 当存在一个实例的时候, 方法就被认为绑定到那个实例了. 没有实例时的方法就是未绑定的.

6)方法纵览:

* 1. 未绑定方法: 直接定义的方法, 并且调用的时候使用类调用而不是实例调用

绑定（绑定及非绑定方法）

为与OOP 惯例保持一致，Python 严格要求，没有实例，方法是不能被调用的。这种限制即Python所描述的绑定概念(binding)，在此，方法必须绑定（到一个实例）才能直接被调用。非绑定的方法可能可以被调用，但实例对象一定要明确给出，才能确保调用成功。然而，不管是否绑定，方法都是它所在的类的固有属性，即使它们几乎总是通过实例来调用的。

* 1. 实例方法: 直接定义的方法, 使用实例调用, 在创建实例时, 被自动绑定到了实例名称空间汇总
  2. 静态方法: 使用@staticmethod定义或使用staticmethod()内建函数绑定的方法, 使用类和实例调用都不会传递默认参数

1.4类方法: 使用@classmethod定义或使用classmethod()内建函数绑定的方法, 不管使用类或实例调用, 默认都会传递第一个参数, 这个参数的值是当前调用的类本身, 或实例所属的类

### 3、内部类的使用

内部类中的方法可以使用两种方法调用

1是直接使用外部类，调用内部类。生成内部类的实例，在调用内部类的方法。

2是先对外部类进行实例化，然后再实例化内部类，最后调用内部类的方法。

内部类容易造成程序结构的复杂性，因此不提倡使用。

### 4、\_\_init\_\_()方法

\_\_init\_\_()方法除了用于定义实例变量外，还用于程序的初始化。\_\_init()方法是可选的，如果不提供\_\_init\_\_()方法，python将会给出一个默认的\_\_init\_\_()方法。

对于已熟悉面向对象编程的人来说，\_\_init\_\_()类似于类构造器。如果你初涉OOP 世界，可以认为一个构造器仅是一个特殊的方法，它在创建一个新的对象时被调用。在Python 中，\_\_init\_\_()实际上不是一个构造器。你没有调用“new”来创建一个新对象。（Python 根本就没有“new”关键字）。取而代之，Python 创建实例后，在实例化过程中，调用\_\_init\_\_()方法，当一个类被实例化时，就可以定义额外的行为，比如，设定初始值或者运行一些初步诊断代码———主要是在实例被创建后，实例化调用返回这个实例之前，去执行某些特定的任务或设置。

注意：  
\_\_init\_\_()方法和java、c++等开发语言一样不能返回任何值。

### 5、\_\_del\_\_()方法

析构方法用于释放对象占用的资源。是可选的，如果程序不提供的话，python会在后台提供默认的析构函数。

### 6、垃圾回收机制

Python提供了gc模块释放不再使用的对象。垃圾回收机制有很多种算法，python采用的是引用计数的方式。

### 7、类的内置方法

Python定义了一些专用的方法。这些专用的方法可以用于不同的应用场合。

类常用的内置方法

\_\_init\_\_()

\_\_del\_\_()

\_\_new\_\_()

\_\_len\_\_(str)

\_\_getattr\_\_(s,name)

\_\_setattr\_\_(s,name,val)

\_\_gelattr\_\_(s,name)

\_\_getattribute\_\_()

Python中不存在setattribute()

等

1）\_\_new\_\_()

可以实现单例模式

1. 其他的一些方法

\_\_getattribute\_\_()被重写之后, 除非明确从\_\_getattribute\_\_()调用, 或\_\_getattribute\_\_()引发了AttributeError异常, 否则, \_\_getattr\_\_不会被调用.

\_\_getattr\_\_和\_\_getattribute\_\_两个系统方法比较

\_\_getattr\_\_方法只有在搜索完类名称空间, 实例名称空间没有发现变量的时候, 才会被调用

\_\_getattribute\_\_如果定义了, 则只使用\_\_getattribute\_\_方法进行搜索变量

getattr(object, attribute\_name)内建函数实际上就是调用了object的\_\_getattr\_\_或\_\_getattribute\_\_方法(如果定义了\_\_getattribute\_\_就不会调用\_\_getattr\_\_), 所以, 在\_\_getattr\_\_或\_\_getattribute\_\_中使用getattr内建函数的时候需要注意, 当该方法没有返回合适的变量的时候, 就会带来无限循环的问题

## 13.3继承

1、通过继承创建类

新类可以继承父类的公有属性和公有方法，但是不能继承父类的私有属性和私有方法。

如果需要，每个子类最好定义它自己的构造器，不然，基类的构造器会被调用。然而，如果子类重写基类的构造器，基类的构造器就不会被自动调用了－－这样，基类的构造器就必须显式写出才会被执行，注意，这里我们要显式传递self 实例对象给基类构造器，因为我们不是在其实例中调用那个方法而是在一个子类实例中调用那个方法。

2、多重继承

Pythn支持多重继承，即一个类可以继承多个父类。

类名后面的圆括号中包含多个父类名，父类之间以逗号隔开。

## 13.4重载

重载允许通过继承而创建的类重新定义父类的方法。不仅可以重载方法，而且还可以重载运算符，例如“+”，“-”等，以适用自创建的类。

1. 方法重载

方法的重载实际上就是在类中使用def关键字重载父类的方法。如果重载父类的方法，但又需要在类中先使用父类的该方法，可以使用父类名加.加方法名的形式调用。

例如：

#-\*- encoding: utf-8 -\*-

*'''*

*Created on 2013-3-13*

*@author: hongtao*

*'''*

class **human**:

\_\_age = 0

\_\_sex = *''*

\_\_height = 0

\_\_weight = 0

name = *''*

def **\_\_init\_\_**(*self*,age,sex,height,weight):

*self*.\_\_age = age

*self*.\_\_sex = sex

*self*.\_\_height = height

*self*.\_\_weight = weight

def **setname**(*self*,name):

*self*.name = name

def **show**(*self*):

print *self*.name

print *self*.\_\_age

print *self*.\_\_height

print *self*.\_\_weight

class **student**(human):

\_\_classes = 0

\_\_grade = 0

\_\_num = 0

def **\_\_init\_\_**(*self*,age,sex,height,weight,classes,grade,num):

*self*.\_\_classes = classes

*self*.\_\_grade = grade

*self*.\_\_num = num

human.\_\_init\_\_(*self*, age, sex, height, weight)

def **show**(*self*):

human.show(*self*)

print *self*.\_\_classes

print *self*.\_\_grade

print *self*.\_\_num

if \_\_name\_\_ == *'\_\_main\_\_'*:

a = student(12,*'male'*,178,2,4,19,20)

a.setname(*'tome'*)

a.show()

#运算结果：

#tome

#12

#178

#2

#4

#19

#20

1. 运算符重载

实际上是对运算符专有方法的重载。

## 13.4命名规则

命名类、属性和方法

类名通常由大写字母打头。这是标准惯例，可以帮助你识别类，特别是在实例化过程中(有时看起来像函数调用)。还有，数据属性（译者注：变量或常量）听起来应当是数据值的名字，方法名应当指出对应对象或值的行为。另一种表达方式是：数据值应该使用名词作为名字，方法使用谓词（动词加对象）。数据项是操作的对象、方法应当表明程序员想要在对象进行什么操作。在上面我们定义的类中，遵循了这样的方针，数据值像“name”，“phone”和“email”，行为“updatePhone”，“updateEmail”。这就是常说的“混合记法(mixedCase)”或“骆驼记法(camelCase)”。Python 规范推荐使用骆驼记法的下划线方式，比如，“update\_phone”，“update\_email”。类也要细致命名，像“AddrBookEntry”，“RepairShop”等等就是很好的名字。

OOD是不要求实现语言的, 也可以使用非OO语言实现OOD设计.