



实用Python程序设计

郭 炜

微信公众号



微博: <http://weibo.com/guoweiofpku>

学会程序和算法，走遍天下都不怕！



北京大学
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

面向对象程序设计



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

北京大学信息学院 郭炜

类和对象



大同火山群

为什么需要"类"

- 用列表或元组表示学生信息:

```
student = ["张三", 20001807, 3.4, "1988-01-24"]
```

记不住GPA到底下标是多少

```
student[3] = XXXX 也不知道是对哪项数据赋值
```

类和对象的概念

- 类是用来代表事物的。对一种事物，可以设计一个类，概括出该种事物的属性，用成员变量表示之；还要概括该种事物事物能进行的操作，用成员函数表示之。成员变量也称为类的“属性”，成员函数也称为类的“方法”。
- 类的实例，称为“对象”。类代表一种事物的共同特点，对象就是一个具体的事物个体。
- 生成对象的方法： 类名(参数1， 参数2.....)

类的写法

```
class 类名:
```

```
    def __init__(self, 参数1, 参数2.....):
```

```
        .....
```

```
    def 成员函数1(self, 参数1, 参数2.....):
```

```
        .....
```

```
    def 成员函数2(self, 参数1, 参数2.....):
```

```
        .....
```

```
    def 成员函数n(self, 参数1, 参数2.....):
```

```
        .....
```

矩形类示例

```
class rectangle:
```

```
    def __init__(self,w,h):    #构造函数, 每个类必有
```

```
        self.w,self.h = w,h
```

```
    def area(self):
```

```
        return self.w * self.h
```

```
    def perimeter(self):
```

```
        return 2 * (self.w + self.h)
```

矩形类示例

```
def main():  
    w,h = map(int,input().split())           #假设输入2 3  
    rect = rectangle(w,h)   #生成一个rectangle对象  
    print(rect.area(),rect.perimeter())      #>>6 10  
    rect.w,rect.h = 10,20  
    print(rect.area(),rect.perimeter())      #>>200 60  
    rect2 = rectangle(2,3)  
    print(rect2.area(), rect2.perimeter())    #>>6 10  
main()
```


类的作用

将数据和操作数据的函数捆绑在一起，便于当作一个整体使用

Python中的类

- 类型名即是类名：

`float, str, list, dict`

- 小数、复数、字符串、元组、列表、集合、字典等组合数据类型的常量，都是对象，函数也是对象，但整数型常量不是对象

- 各种库都是由类组成：

`turtle, matplotlib, jieba, sqlite3`

- 程序员可以自定义类，如`rectangle`



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

北京大学信息学院 郭炜

对象的比较



大同城墙

对象的比较

- Python中所有的类，包括自定义的类，都有`__eq__`方法。
- `x==y` 的值，就是`x.__eq__(y)`的值；如果`x.__eq__(y)`没定义，那么就是`y.__eq__(x)`的值。如果`x.__eq__(y)`和`y.__eq__(x)`都没定义，则`x == y`也没定义（`x,y`都是整数常量时不适用）
- `print(24.5.__eq__(24.5))` `#>>True`

对象的比较

$a \neq b$ 等价于 $a \cdot \underline{\text{ne}}(b)$, 或 $b \cdot \underline{\text{ne}}(a)$ (若 $a \cdot \text{ne}(b)$ 没定义)

默认情况下, `a. ne (b)` 等价于 `not a. eq (b)`

$$a < b \quad \text{等价于} \quad a \cdot \frac{1}{b} < 1 \quad (b) > 0$$

$a > b$ 等价于 $a \cdot gt(b)$

$a \leq b$ 等价于 $a \leq 1 \wedge b \leq 1$ (b)

$a \geq b$ 等价于 $a \cdot \text{ge} \quad (b)$

自定义对象的比较

- 默认情况下，一个自定义类的`__eq__`方法，功能是判断两个对象的id是否相同。
- 默认情况下，一个自定义类的两个对象a和b，`a == b`和`a is b`的含义一样，都是“a和b是否指向相同的地方”。同理，`a != b`和`not a is b`含义相同。
- 默认情况下，自定义类的对象不能比较大小，因其`__lt__`、`__gt__`、`__le__`、`__ge__`方法都被设置成了None

对象比较大小程序示例

```
class point:
    def __init__(self, x, y = 0):
        self.x , self.y = x,y
    def __eq__(self,other):
        return self.x == other.x and self.y == other.y
    def __lt__(self,other): #使得两个point对象可以用<进行比较
        if self.x == other.x:
            return self.y < other.y
        else:
            return self.x < other.x
```

对象比较大小程序示例

```
a,b = point(1,2),point(1,2)
print(a == b)           #>>True      等价于 a.__eq__(b)
print(a != b)           #>>False
print(a < point(0,1))   #>>False     等价于 a.__lt__(point(0,1))
print(a < point(1,3))   #>>True
lst = [a,point(-2,3),point(7,8),point(5,9),point(5,0)]
lst.sort()
for p in lst:            #>>-2 3,1 2,5 0,5 9,7 8,
    print(p.x,p.y ,end = ",")
```


输出对象

➤ 自定义类重写__str__方法可以将对象转字符串

```
class point:
    def __init__(self,x,y):
        self.x ,self.y = x ,y
    def __str__(self):
        return "(%d,%d)" % (self.x, self.y)
```

```
print(point(3,5))           #>> (3,5)
```

```
print(str(point(2,4)))      #>> (2,4)
```



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

北京大学信息学院 郭炜

继承和派生



祁连山

继承和派生

- 要写小学生类、中学生类、大学生类....
- 所有学生都有共同点，每种学生又有各自特点，如何避免每个类都从头编写的重复劳动？

继承和派生

- 要写小学生类、中学生类、大学生类....
- 所有学生都有共同点，每种学生又有各自特点，如何避免每个类都从头编写的重复劳动？
- 使用继承（派生）

继承和派生

定义一个新的类B时，如果发现类B拥有某个已写好的类A的全部特点，此外还有类A没有的特点，那么就不必从头重写类B，而是可以把A作为一个“基类”（也称“父类”），把B写为基类A的一个“派生类”（也称“子类”）来写。这样，就可以说从A类“派生”出了B类，也可以说B类“继承”了A类。

```
class 类名(基类名):
```

```
.....
```

继承和派生

```
import datetime
class student:
    def __init__(self,id,name,gender,birthYear):
        self.id,self.name,self.gender,self.birthYear = \
            id,name,gender,birthYear
    def printInfo(self):
        print("Name:",self.name)
        print("ID:", self.id)
        print("Birth Year:",self.birthYear)
        print("Gender:",self.gender)
        print("Age:",self.countAge())
    def countAge(self):
        return datetime.datetime.now().year - self.birthYear
```

继承和派生

```
class undergraduateStudent(student): #本科生类, 继承了student类
    def __init__(self, id, name, gender, birthYear, department):
        student.__init__(self, id, name, gender, birthYear)
        self.department = department
    def qualifiedForBaoyan(self): #给予保研资格
        print(self.name + " is qualified for baoyan")
    def printInfo(self): #基类中有同名方法
        student.printInfo(self) #调用基类的PrintInfo
        print("Department: " , self.department)
def main():
    s2 = undergraduateStudent("118829212", "Harry Potter",
                               "M", 2000, "Computer Science")
    s2.printInfo()
    s2.qualifiedForBaoyan()
    if s2.countAge() > 18:
        print(s2.name , "is older than 18")
main()
```

继承和派生

输出:

Name: Harry Potter

ID: 118829212

Birth Year: 2000

Gender: M

Age: 20

Department: Computer Science

Harry Potter is qualified for baoyan

Harry Potter is older than 18

object类

- 所有类都是object类的派生类，因而具有object类的各种属性和方法

```
class A:  
    def func(x):  
        pass  
print(dir(A)) #输出A的方法
```

输出:

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',  
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',  
'__hash__', '__init__', '__le__', '__lt__', '__module__', '__ne__',  
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',  
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'func']
```

- 有的类 `__lt__`, `__gt__` 等方法被设置成None，于是对象不可比较大小



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

北京大学信息学院 郭炜

静态属性和 静态方法



草原天路

静态属性和静态方法

- 静态属性被所有对象所共享，一共只有一份
- 静态方法不是作用在具体的某个对象上的，不能访问非静态属性
- 静态属性和静态方法这种机制存在的目的，就是为了少写全局变量和全局函数。

静态属性和静态方法

```
class employee:
    totalSalary = 0          #静态属性，记录发给员工的工资总数
    def __init__(self, name, income):
        self.name, self.income = name, income
    def pay(self, salary):
        self.income += salary
        employee.totalSalary += salary

    @staticmethod
    def printTotalSalary():    # 静态方法
        print(employee.totalSalary)
```

静态属性和静态方法

```
e1 = employee("Jack",0)
e2 = employee("Tom",0)
e1.pay(100)
e2.pay(200)
employee.printTotalSalary()      #>>300
e1.printTotalSalary()           #>>300
e2.printTotalSalary()           #>>300
print(employee.totalSalary)     #>>300
```




北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

北京大学信息学院 郭炜

对象作为
集合元素或字典的键



福建宁德太姥山

什么是“可哈希”

➤ 可哈希的东西，才可以作为字典的键和集合的元素

➤ `hash(x)` 有定义，即为 `x` 可哈希

➤ `hash(x) = x` (如果`x`是整型常量)

`hash(x) = x.__hash__()` (如果`x`不是整型常量)

➤ `object`类有`__hash__()`方法，返回值是个整数

什么是“可哈希”

列表、集合、字典的 `__hash__` 成员函数都被设置成 `None`，因此它们都不能成为集合的元素，或者字典的键，因为无法计算哈希值。

什么是“可哈希”

- 整数类型变量、小数、字符串、元组的哈希值，是根据它们的值算出来的，只要值相同，哈希值就相同。

```
x = 23.1
print(x.__hash__(), 23.1.__hash__())
#>>230584300921372695 230584300921372695

x = 23
print(x.__hash__(), hash(23)) #>>23 23

x = (1,2)
print(x.__hash__(), (1,2).__hash__(), hash(x))
#>>3713081631934410656 3713081631934410656 3713081631934410656

x = "ok"
print(x.__hash__(), "ok".__hash__())
#>>-423760875654480603 -423760875654480603
```

哈希值和字典、集合的关系

- 字典和集合都是“哈希表”数据结构，根据元素的哈希值为元素找存放的“槽”，哈希值可以看作是槽编号。一个槽里面可以放多个哈希值相同的元素。
- 两个对象a,b 若 $\text{hash}(a) \neq \text{hash}(b)$ ，则a,b可以处于同一集合（也可以作为同一字典的不同元素的键）
- 两个对象a,b 若 $\text{hash}(a) == \text{hash}(b)$ ，但 $a == b$ 不成立，则a,b可以处于同一集合（也可以作为同一字典的不同元素的键），即不算重复，可以放在同一个槽里

哈希值和字典、集合的关系

➤ 若 dt 是个字典, $dt[x]$ 计算过程如下:

- 1) 根据 $hash(x)$ 去找 x 应该在的槽的编号
- 2) 如果该槽没有元素, 则认为 dt 中没有键为 x 的元素
- 3) 如果该槽中有元素, 则试图在槽中找一个元素 y , 使得 y 的键 $== x$ 。
如果找到, 则 $dt[x]$ 即为 y 的值, 如果找不到, 则 $dt[x]$ 没定义, 即认为 dt 中不存在键为 x 的元素

哈希值和字典、集合的关系

- 自定义类的对象，默认情况下哈希值是根据对象id进行计算。所以两个对象，只要`a is b`不成立，`a`和`b`的哈希值就不同，就可以同时存在于一个集合内，或作为同一字典的不同元素的键。
- 可以重写自定义类的`__hash__()`方法，使得对象的哈希值和对象的值，而不是id相关，这样值相同的对象，就不能处于同一个集合中，也不能作为同一字典不同元素的键。

哈希值和字典、集合的关系

```
class A:  
    def __init__(self,x):  
        self.x = x
```

```
a,b = A(5),A(5)  
dt = {a:20,A(5):30,b:40}  
print(len(dt),dt[a],dt[b])  
print(dt[A(5)])
```

#两个A(5)不是同一个, 因此a和b的id不同

#三个元素的键id不同, 因此在不同槽里

#>>3 20 40

#runtime error

哈希值和字典、集合的关系

```
class A:  
    def __init__(self,x):  
        self.x = x
```

```
a,b = A(5),A(5)  
dt = {a:20,A(5):30,b:40}  
print(len(dt),dt[a],dt[b])  
print(dt[A(5)])
```

#两个A(5)不是同一个, 因此a和b的id不同

#三个元素的键id不同, 因此在不同槽里

#>>3 20 40

#runtime error

自定义类的对象是否可哈希

- $a==b$ 等价于`a.__eq__(b)`。自定义类的默认`__eq__`函数是判断两个对象的`id`是否相同。自定义类的默认`__hash__`函数是根据对象`id`算哈希值的。
- 如果为自定义的类重写了`__eq__(self,other)`成员函数，则其`__hash__`成员函数会被自动设置为`None`。这种情况下，该类就变成不可哈希的
- 一个自定义类，只有在重写了`__eq__`方法却没有重写`__hash__`方法的情况下，才是不可哈希的。

自定义类重写__hash__但不重写__eq__

```
class A:
    def __init__(self,x):
        self.x = x
    def __hash__(self):
        return hash(self.x)
```

```
c = A(1)
```

```
dt = {A(1):2,A(1):3,c:4}
```

#三个元素的键哈希值相同，但id不同，它们在同一个槽里

```
print(len(dt)) #>>3
```

```
for a in dt.items():
```

```
    print(a[0].x,a[1],end = ",") #>>1 2,1 3,1 4,
```

```
print(dt[c]) #>>4
```

```
print(dt[A(1)]) #runtime error
```

#因不存在元素的键x，满足 $x == A(1)$ (特指最后一行的A(1))

自定义类同时重写__hash__和__eq__

```
class A:
    def __init__(self,x):
        self.x = x
    def __eq__(self,other):
        if isinstance(other,A):           #判断other是不是类A的对象
            return self.x == other.x
        elif isinstance(other,int):      #如果other是整数
            return self.x == other
        else:
            return False
    def __hash__(self):
        return self.x
```

自定义类同时重写__hash__和__eq__

```
a = A(3)
```

```
print(3 == a)
```

```
#>>True
```

```
b = A(3)
```

```
d = {A(5):10,A(3):20,a:30}
```

```
print(len(d),d[a],d[b],d[3])
```

```
#>>2 30 30 30
```