**属性的定义**：python中的属性其实是普通方法的衍生。

**操作类属性有三种方法：**

1.使用@property装饰器操作类属性。

2.使用类或实例直接操作类属性（例如：obj.name，obj.age=18，del obj.age）

3.使用python内置函数操作属性。

**属性存在的意义**：

1、访问属性时可以制造出和访问字段完全相同的假象，属性由方法衍生而来，如果Python中没有属性，方法完全可以代替其功能。

2、定义属性可以动态获取某个属性值，属性值由属性对应的方式实现，应用更灵活。

3、可以制定自己的属性规则，用于防止他人随意修改属性值。

下面详细介绍三种操作类属性的方法：

1.使用@property装饰器操作类属性。

     定义时，在普通方法的基础上添加@property装饰器；属性仅有一个self参数，调用时无需括号；

   优点：

    1) @property装饰器可以实现其他语言所拥有的getter，setter和deleter的功能（例如实现获取，设置，删除隐藏的属性）

    2) 通过@property装饰器可以对属性的取值和赋值加以控制,提高代码的稳定性。

    实例代码1：

[复制代码](javascript:void(0);)

#encoding=utf-8

class Goods(): #新式类

@property

def price(self): #查看属性值

print ('@property ')

@price.setter #修改、设置属性

def price(self, value):

print ('@price.setter' )

@price.deleter #删除属性

def price(self):

print ('@price.deleter')

obj = Goods(50)

obj.price # 自动执行 @property 修饰的 price 方法，并获取方法的返回值

obj.price = 2000 # 自动执行 @price.setter 修饰的 price 方法，并将2000赋值给方法的参数

del obj.price # 自动执行 @price.deleter 修饰的 price 方法   
结果输出：  
@property  
@price.setter  
@price.deleter

[复制代码](javascript:void(0);)

实例代码2：通过@property装饰器对属性的取值和赋值加以控制

[复制代码](javascript:void(0);)

class Goods(object):

def \_\_init\_\_(self):

self.value=50

@property

def price(self): # 查看属性

return self.value

@price.setter # 添加或设置属性（属性名.setter）

def price(self, value):

if value >=50 and value<=100:  #对属性的取值和赋值加以控制

self.value=value

print (self.value)

else:

print ("请输入一个50到100之间的数！")

@price.deleter # 删除属性（属性名.deleter） 注意：属性一旦删除，就无法设置和获取

def price(self):

del self.value

print ("price is deleted!")

obj = Goods()

print (obj.price) # 自动执行 @property 修饰的 price 方法，并获取方法的返回值

obj.price=106 # 自动执行 @price.setter 修饰的 price 方法，并将106 赋值给方法

del obj.price # 自动执行 @price.deleter 修饰的 price 方法

结果输出：

50

请输入一个50到100之间的数！

price is deleted!

[复制代码](javascript:void(0);)

2.使用类或实例直接操作类属性

 缺点：对类的属性没有操作控制规则，容易被人修改。

[复制代码](javascript:void(0);)

#coding=utf-8

class Employee (object):

#所有员工基类

empCount = 0

def \_\_init\_\_(self, name, salary) :

#类的构造函数

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self) :

#类方法

print ("total employee ",Employee.empCount)

def displayEmployee(self) :

print ("name :",self.name , ", salary :", self.salary)

#创建Employee类的实例对象

emp1 = Employee("丽丽", 10000)

emp1.displayCount()

emp1.displayEmployee()

emp1.salary = 20000 #修改属性 salary

print (emp1.salary)

emp1.age = 25 #添加属性 age

print (emp1.age)

del emp1.age #删除 age属性

emp1.empCount=100

Employee.empCount=1000  
结果：  
total employee  1  
name : 丽丽 , salary : 10000  
20000  
25

[复制代码](javascript:void(0);)

3.使用python内置函数操作属性。  
   1）getattr(obj, name[, default])：访问对象的属性，如果存在返回对象属性的值，否则抛出AttributeError异常。  
   2）hasattr(obj,name)：检查是否存在某个属性，存在返回True，否则返回False。  
   3）setattr(obj,name,value)：设置一个属性。如果属性不存在，会创建一个新属性，该函数无返回值。若存在则更新这个值。  
   4）delattr(obj, name)：删除属性，如果属性不存在则抛出AttributeError异常，该函数也无返回值。

应用实例：

[复制代码](javascript:void(0);)

#encoding=utf-8

class Employee(object):

#所有员工基类

empCount=0

def \_\_init\_\_(self,name,age,salary):

#类的构造函数

self.name=name

self.salary=salary

self.age=age

Employee.empCount+=1

def displayCount(self):

#类方法

print("total employee",Employee.empCount)

def displayEmployee(self):

print("name:",self.name,"age:",self.age,",salary:",self.salary)

#创建Employee类的实例对象

emp1=Employee("Rose",27,20000)

#判断实例对象是否存在某个属性,存在返回True,否则返回False

if hasattr(emp1,'name'):

name\_value=getattr(emp1,'name') #获取name属性值

print( "name的属性值为：",name\_value)

else:

print ("员工属性不存在")

#给实例添加一个属性

if hasattr(emp1,'tel'):

print ("员工属性已存在")

else:

setattr(emp1,'tel','17718533234')

t1=getattr(emp1,'tel')

print("tel的属性值为：",t1)

setattr(emp1,'tel','15042622134')

t2=getattr(emp1,'tel')

print("tel修改后的属性值为：",t2)

#给实例删除一个属性

if hasattr(emp1,'age'):

delattr(emp1,'age')

else:

print ("员工tel属性不存在")

#验证属性是否删除成功

if hasattr(emp1,'age'):

print( "属性age存在！")

else:

print ("属性age不存在！")  
结果：  
name的属性值为： Rose  
tel的属性值为： 17718533234  
tel修改后的属性值为： 15042622134  
属性age不存在！

[复制代码](javascript:void(0);)

此外还有Python内置类属性，这里做简单介绍：

**\_\_dict\_\_ : 类的属性（获取类所有信息）：**结果返回一个字典包含类属性及属性值，类方法等所有类信息

**\_\_doc\_\_ :类的文档字符串，也就是类的帮助信息。**

**\_\_name\_\_: 类名**

**\_\_module\_\_: 类定义所在的模块** 如果在当前模块返回’\_\_main\_\_’; 如果类位于一个导入模块mymod中，那么className.\_\_module\_\_ 等于 mymod）

**\_\_bases\_\_ : 类的所有父类**（包含了所有父类组成的元组）

实例代码：

[复制代码](javascript:void(0);)

#coding=utf-8

class Employee (object):

"""所有员工的基类"""

empCount = 0

def \_\_init\_\_(self, name, salary) :

#类的构造函数

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self) :

#类方法

print ("total employee ",Employee.empCount )

def displayEmployee(self) :

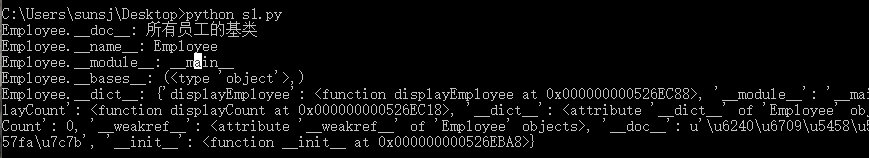
print ("name :",self.name , ", salary :", self.salary)

print ("Employee.\_\_doc\_\_:", Employee.\_\_doc\_\_ )

print ("Employee.\_\_name\_\_:", Employee.\_\_name\_\_ )

print ("Employee.\_\_module\_\_:", Employee.\_\_module\_\_ )

print ("Employee.\_\_bases\_\_:", Employee.\_\_bases\_\_ )

print ("Employee.\_\_dict\_\_:", Employee.\_\_dict\_\_ )  
结果：  
****

[复制代码](javascript:void(0);)