# 面向对象

## 属性和方法：

### 2.1 静态字段

**class** Province:  
 memo = **'某省'**

属于类，可以通过对象调用，也可以通过类直接调用

### 2.2 动态字段

**def** \_\_init\_\_(self,name,capital,leader):  
 self.Name=name;

通过self造的数据动态字段，属于对象

### 2.3 构造方法\_\_init\_\_(self)

类似于java的构造方法

### 2.4 动态方法：

在类地内部，使用def关键字可以为类定义一个方法，与一般函数定义不同，类方法必须包含参数self,且为第一个参数:

**def** sport\_meet(self):  
 **print** self.Name+**'正在开运动会'**

### 2.5 静态方法：

对象和类都和以调用

@staticmethod  
**def** sport():  
 **print '这是个静态方法'**

### 2.6 特性@property

类似于java中的getter和setter方法，操纵私有属性很好

在不继承object时，被@property的属性，可写也可读，

只有加继承object后，属性只可读，必须用@Bar.setter才实现可写

**把方法作为对象的属性，可以直接对象调用，通过类调用返回**

**<property object at 0x025CA0F0>**

*# 只读特性*@property  
**def** Bar(self):  
 **return** self.\_\_memo

*# 只写特性*@Bar.setter  
**def** Bar(self,value):  
 self.\_\_memo=value

**print** hb.Bar   
hb.Bar=**'山西'  
print** hb.Bar

### 2.7 实例化对象

类对象支持两种操作：属性引用和实例化。

#!/usr/bin/python3

class MyClass:

"""一个简单的类实例"""

i = 12345

def f(self):

return 'hello world'

# 实例化类

x = MyClass()

# 访问类的属性和方法print("MyClass 类的属性 i 为：", x.i)print("MyClass 类的方法 f 输出为：", x.f())

### **2.8 类的私有属性**

**强行调用私有属性：**

**class** Province:  
 \_\_memo = **'某省'**

**print** hb.\_Province\_\_memo

**\_\_private\_attrs**：两个下划线开头，声明该属性为私有，不能在类地外部被使用或直接访问。在类内部的方法中使用时 **self.\_\_private\_attrs**。

#定义私有属性,私有属性在类外部无法直接进行访问

\_\_weight = 0

### **2.9 类的私有方法**

**\_\_private\_method**：两个下划线开头，声明该方法为私有方法，不能在类地外部调用。在类的内部调用 **slef.\_\_private\_methods**。

### 2.10 析构函数\_\_del\_\_(self):

对象销毁前会调用

**def** \_\_del\_\_(self):  
 **print "最后的呐喊！"**

### 2.11 \_\_call\_\_(self):

**def** \_\_call\_\_(self):  
 **print "call"**

调用方式：对象名（） ：hb()

### **2.12 类的专有方法：**

* **\_\_init\_\_ :** 构造函数，在生成对象时调用
* **\_\_del\_\_ :** 析构函数，释放对象时使用
* **\_\_repr\_\_ :** 打印，转换
* **\_\_setitem\_\_ :** 按照索引赋值
* **\_\_getitem\_\_:** 按照索引获取值
* **\_\_len\_\_:** 获得长度
* **\_\_cmp\_\_:** 比较运算
* **\_\_call\_\_:** 函数调用
* **\_\_add\_\_:** 加运算
* **\_\_sub\_\_:** 减运算
* **\_\_mul\_\_:** 乘运算
* **\_\_div\_\_:** 除运算
* **\_\_mod\_\_:** 求余运算
* **\_\_pow\_\_:** 称方

## 继承

Python 同样支持类的继承，如果一种语言不支持继承，类就没有什么意义。派生类的定义如下所示:

class DerivedClassName(BaseClassName1):

<statement-1>

<statement-N>

需要注意圆括号中基类的顺序，若是基类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找基类中是否包含方法。

BaseClassName（示例中的基类名）必须与派生类定义在一个作用域内。除了类，还可以用表达式，基类定义在另一个模块中时这一点非常有用:

class DerivedClassName(modname.BaseClassName):

### **新式和经典类object**

继承了object的类为新式类，没继承的为经典类

Python2.2.3后产生了新式类，兼容经典类

新式解决的类多继承的继承优先级，经典类为深度优先，新式类为广度优先

### **调用父类构造方法**

**class** Province(object):

**def** \_\_init\_\_(self,name,capital,leader):

**Pass**

**class** Son(Province):  
 **def** \_\_init\_\_(self):  
 **pass**

#显示调用

Province.\_\_init\_\_(self,**'河北'**,**'石家庄'**,**'李阳'**)

#super调用，需要父类继承object类

super(Son,self).\_\_init\_\_(**'河北'**,**'石家庄'**,**'李阳'**)

### **实例**

#!/usr/bin/python3

#类定义class people:

#定义基本属性

name = ''

age = 0

#定义私有属性,私有属性在类外部无法直接进行访问

\_\_weight = 0

#定义构造方法

def \_\_init\_\_(self,n,a,w):

self.name = n

self.age = a

self.\_\_weight = w

def speak(self):

print("%s 说: 我 %d 岁。" %(self.name,self.age))

#单继承示例class student(people):

grade = ''

def \_\_init\_\_(self,n,a,w,g):

#调用父类的构函

people.\_\_init\_\_(self,n,a,w)

self.grade = g

#覆写父类的方法

def speak(self):

print("%s 说: 我 %d 岁了，我在读 %d 年级"%(self.name,self.age,self.grade))

s = student('ken',10,60,3)

s.speak()

执行以上程序输出结果为：

ken 说: 我 10 岁了，我在读 3 年级

### 多继承

Python同样有限的支持多继承形式。多继承的类定义形如下例:

class DerivedClassName(Base1, Base2, Base3):

<statement-1>

<statement-N>

需要注意圆括号中父类的顺序，若是父类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找父类中是否包含方法。

#!/usr/bin/python3

#类定义class people:

#定义基本属性

name = ''

age = 0

#定义私有属性,私有属性在类外部无法直接进行访问

\_\_weight = 0

#定义构造方法

def \_\_init\_\_(self,n,a,w):

self.name = n

self.age = a

self.\_\_weight = w

def speak(self):

print("%s 说: 我 %d 岁。" %(self.name,self.age))

#单继承示例class student(people):

grade = ''

def \_\_init\_\_(self,n,a,w,g):

#调用父类的构函

people.\_\_init\_\_(self,n,a,w)

self.grade = g

#覆写父类的方法

def speak(self):

print("%s 说: 我 %d 岁了，我在读 %d 年级"%(self.name,self.age,self.grade))

#另一个类，多重继承之前的准备class speaker():

topic = ''

name = ''

def \_\_init\_\_(self,n,t):

self.name = n

self.topic = t

def speak(self):

print("我叫 %s，我是一个演说家，我演讲的主题是 %s"%(self.name,self.topic))

#多重继承class sample(speaker,student):

a =''

def \_\_init\_\_(self,n,a,w,g,t):

student.\_\_init\_\_(self,n,a,w,g)

speaker.\_\_init\_\_(self,n,t)

test = sample("Tim",25,80,4,"Python")

test.speak() #方法名同，默认调用的是在括号中排前地父类的方法

执行以上程序输出结果为：

我叫 Tim，我是一个演说家，我演讲的主题是 Python

### 方法重写

如果你的父类方法的功能不能满足你的需求，你可以在子类重写你父类的方法，实例如下：

#!/usr/bin/python3

class Parent: # 定义父类

def myMethod(self):

print ('调用父类方法')

class Child(Parent): # 定义子类

def myMethod(self):

print ('调用子类方法')

c = Child() # 子类实例

c.myMethod() # 子类调用重写方法

执行以上程序输出结果为：

调用子类方法

### **运算符重载**

Python同样支持运算符重载，我么可以对类的专有方法进行重载，实例如下：

#!/usr/bin/python3

class Vector:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def \_\_str\_\_(self):

return 'Vector (%d, %d)' % (self.a, self.b)

def \_\_add\_\_(self,other):

return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)

v2 = Vector(5,-2)print (v1 + v2)

以上代码执行结果如下所示:

Vector(7,8)

### 抽象类

|  |
| --- |
| **from** abc **import** ABCMeta,abstractmethod **class** Alert:  \_\_metaclass\_\_ = ABCMeta  @abstractmethod  **def** send(self):**pass  class** Weixin(Alert):  **def** \_\_init\_\_(self):  **print 'init'  def** send(self):  **print 'send.Weixin'** |

## 异常处理

### 捕获异常

|  |
| --- |
| **try**:  namespace=\_\_import\_\_(**'back.'**+array[0]) *#得到包* mode=getattr(namespace,array[0]) *#得到模块* func=getattr(mode,array[1]) *#得到方法* func() **except** (ImportError,AttributeError),e:  **print** e  **print "跳转到404"**  **except** AssertionError,e:  **pass**  **#没出错执行**  **else:  print '没出错'**  **#有没有出错都执行 finally:  print '最终执行'** |

### 自定义异常

|  |
| --- |
| **class** MyException(Exception):  **def** \_\_init\_\_(self,msg):  self.error=msg  **def** \_\_str\_\_(self,\*args,\*\*kwargs):  **return** self.error obj=MemoryError(**"错误"**) **print** obj  **#主动触发错误，类似以java的throw**  **raise** MyException(**'cuowu'**) |