# 类的设计

## Python和OOP

继承：继承是基于python中的属性查找的

多态：在X.method方法中，method的意义取决于X的类型（类）。

封装：方法和运算符实现行为，数据隐藏默认是一种惯例。

### 通过调用标记进行重载

class C:

def meth(self, x):

...

def meth(self, x, y, z):

...

因为def只是在类的作用域中把对象赋值给变量名，这个方法函数的最后一个定义才是保留的。

### OOP和继承：“是一个”关系

“is-a”关系

#-\*-coding:utf-8-\*-

class **Employee**:

def **\_\_init\_\_**(*self*, name, salary=0):

*self*.name = name

*self*.salary = salary

def **giveRaise**(*self*, precent):

*self*.salary = *self*.salary \* (1.0 + precent)

def **work**(*self*):

print *self*.name, *"does stuff"*

def **\_\_repr\_\_**(*self*):

return *"<Employee:name=%s,salary=%s>"* %(*self*.name, *self*.salary)

class **Chef**(Employee):

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

Employee.\_\_init\_\_(*self*, name, 5000)

def **work**(*self*):

print *self*.name, *"interfaces with customer"*

class **PiaazRobot**(Chef):

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

Chef.\_\_init\_\_(*self*, name)

def **work**(*self*):

print *self*.name, *"makes pizza"*

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

bob = PiaazRobot(*'bob'*)

print bob

bob.work()

bob.giveRaise(.20)

print bob

print ()

for klass in Employee, Chef, PiaazRobot:

obj = klass(klass.\_\_name\_\_)

obj.work()

### OOP和组合：“有一个”关系

“has-a”关系

#-\*-coding:utf-8-\*-

from employees import PizzaRobot, Server

class **Customer**:

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

*self*.name = name

def **order**(*self*, server):

print *self*.name, *"orders from"*, server

def **pay**(*self*, server):

print *self*.name, *"pays for item to"*, server

class **Oven**:

def **bake**(*self*):

print *"oven bakes"*

class **PizzaShop**:

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

*self*.server = Server(*'Pat'*)

*self*.chef = PizzaRobot(*'Bob'*)

*self*.oven = Oven()

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

customer = Customer(name)

customer.order(*self*.server)

*self*.chef.work()

*self*.oven.bake()

customer.pay(*self*.server)

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

scene = PizzaShop()

scene.order(*"Homer"*)

print *'........'*

scene.order(*'Shaggy'*)

### OOP和委托：“包装”对象

委托（delegation）通常就是指控制器对象内嵌其他对象，而把运算请求传给那些对象。

class **wrapper**:

def **\_\_init\_\_**(*self*, object):

*self*.wrapped = object

def **\_\_getattr\_\_**(*self*, attrname):

print *"Trace:"*, attrname

return getattr(*self*.wrapped, attrname)

X = wrapper([1,2,3])

X.append(4)

print X.wrapped

X = wrapper({*'a'*:1, *'b'*:2})

print X.keys()

## 方法是对象：绑定或无绑定

无绑定类对象：无self

通过对类进行点号运算从而获取类的函数属性，会传回无绑定（unbond）方法对象。调用该方法时，必须明确提供实例对象作为第一个参数。

绑定实例方法对象：self+函数对

## 类的高级主题

### 扩展内置类型

#### 通过嵌入扩展内置类型

class **Set**:

def **\_\_init\_\_**(*self*, value = []):

*self*.data = []

*self*.concat(value)

def **intersect**(*self*, other):

res = []

for x in *self*.data:

if x in other:

res.append(x)

return Set(res)

def **union**(*self*, other):

res = *self*.data[:]

for x in other:

if not x in res:

res.append(x)

return Set(res)

def **concat**(*self*, value):

for x in value:

if not x in *self*.data:

*self*.data.append(x)

def **\_\_len\_\_**(*self*):

return len(*self*.data)

def **\_\_getitem\_\_**(*self*, key):

return *self*.data[key]

def **\_\_and\_\_**(*self*, other):

return *self*.intersect(other)

def **\_\_or\_\_**(*self*, other):

return *self*.union(other)

def **\_\_repr\_\_**(*self*):

return *'Set:'* + repr(*self*.data)

x = Set([1,3,5,7])

print x.union(Set([1,4,7]))

print x[1:3]

### 新式类

* 对于python3.x所有的类都是新式类。
* 对于Python2.6及以前的版本，类必须继承自的类看做是“新式”object。

### 新式类的变化

* 类与类型合并。type(I)内置函数返回一个实例所创建的类。

>>> class A:

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

self.x = 99

>>> a = A()

>>> b = type(a)

>>> b

<type 'instance'>

**新式类**

>>> class Y(object):

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

self.y = 88

>>> y =Y()

>>> type(y)

<class '\_\_main\_\_.Y'>

>>> z = type(y)() # type内置函数返回一个实例创建的类

>>> z

<\_\_main\_\_.Y object at 0x0000000003198BE0>

>>> z.y

88

* 继承搜索顺序，先横向搜索再纵向搜索，并且先宽度优先搜索，再深度优先搜索。
* 针对内置函数的属性获取。
* 新的高级工具。

元类是这样一个类，要么它是type本身，要么是它定制扩展或管理生成的类的一个子类。

#### 钻石继承变动

对经典类而言，继承搜索是深度优先，然后才是从左至右。在新式类中搜索相对是宽度优先，搜索过程先水平而行，然后向上移动。

class **A**:

attr = 1

class **B**(A): pass

class **C**(A):

attr = 2

class **D**(B, C): pass

X = D()

print X.attr # 1

**新式类：**

class **A**(object):

attr = 1

class **B**(A): pass

class **C**(A):

attr = 2

class **D**(B, C): pass

X = D()

print X.attr # 2