## python 第三部分

class

#### 定义

```
1 #encoding:utf-8
2
3 class Anjuke:
4 pass
```

```
1 #encoding:utf-8
2
3 class Anjuke:
4 print "This is anjuke"
```

```
1 #encoding:utf-8
2
3 class Anjuke:
4 departments = 1
5 def init(self):
6 self.departments += 1
```

#### 公有属性

公有属性可以通过实例直接访问和修改

```
#encoding:utf-8
2
3
    class Anjuke:
        departments = ["user", "mobile", "salesmen"]
4
5
         address = "东方路1217号"
6
7
    if __name__ == '__main__':
8
        a = Anjuke()
         print a.departments
10
        a.address = 123
11
         print a.address
```

#### 私有属性

私有属性不可以通过实例直接访问和修改注意和公有属性在命名上的区别

```
#encoding:utf-8
 1
 3
     class Anjuke:
 4
         __departments = ["user", "mobile", "salesmen"]
          _address = "东方路1217号"
6
    if __name__ == '__main__':
8
         a = Anjuke()
9
         print a.__departments
10
         a.\_address = 123
11
         print a.__address
```

#### 方法

```
#encoding:utf-8
 2
 3
     class Anjuke:
         departments = ["user", "mobile", "salesmen"]
 4
         address = "东方路1217号"
 6
         def print_address(self):
             print self.address
 8
         def print_departments(self):
11
             print self.departments
12
13
     if __name__ == '__main__':
14
        a = Anjuke()
15
         a.print_address()
        a.print_departments()
16
```

#### 私有方法

```
#encoding:utf-8
3
    class Anjuke:
        __departments = ["user", "mobile", "salesmen"]
        __address = "东方路1217号"
        def __print_address(self):
            print self.__address
        def print_address(self):
            self.__print_address()
12
    if __name__ = '__main__':
        a = Anjuke()
        a.print_address()
```

#### 纠结的self参数

方法总是将他们的第一个参数绑定到所属实例上

```
#encoding:utf-8
2
    class Anjuke:
         address = "东方路1217号"
         def print_address(self):
 6
             print self.address
 8
    if __name__ == '__main__':
 9
10
         a = Anjuke()
        b = Anjuke()
11
        b.address = "13楼"
12
        Anjuke.print_address(a)
13
         Anjuke.print_address(b)
14
```

```
#encoding:utf-8
    class Anjuke:
        address = "东方路1217号"
        def print_address(self):
 6
            print type(self),hex(id(self))
 8
    if __name__ == '__main__':
10
        a = Anjuke()
        b = Anjuke()
11
        print 'a的地址是',hex(id(a))
12
        print 'b的地址是',hex(id(b))
13
        Anjuke.print_address(a)
14
        Anjuke.print_address(b)
15
```

#### self的绑定并不依赖调用方式,请看如下例子 通过变量访问实例方法也是绑定到所属实例上

```
#encoding:utf-8
 2
 3
     class Anjuke:
         address = "东方路1217号"
 4
 6
         def print_address(self):
             print type(self),hex(id(self))
 8
     if __name__ == '__main__':
 9
10
         a = Anjuke()
         Anjuke.print_address(a)
11
12
         other = a.print_address
         other()
13
)14
         print type(other)
```

#### 常用魔法方法

- \_\_init\_\_()
- \_\_del\_\_()
- \_\_getitem\_\_()
- \_\_setitem\_\_()
- \_\_len\_\_()

- \_\_getattr\_\_()
- setattr\_()
- delattr\_\_()
- \_\_\_iter\_\_\_()
- \_\_add\_\_()

#### init\_\_()

```
#encoding:utf-8
1
 2
 3
     class Anjuke:
         address = "东方路1217号"
 4
         depart = "用户端"
         def __init__(self,*args):
 6
             if len(args) > 0:
 8
                 self.address = args[0]
                 self.depart = args[1]
         def print_attr(self):
11
12
             print self.address
13
             print self.depart
14
15
     if __name__ == '__main__':
        a = Anjuke()
16
        a.print_attr()
17
18
         b = Anjuke("陆家嘴","网站端")
19
         b.print_attr()
```

类实例化时自动调用

#### del\_\_()

```
#encoding:utf-8
 2
 3
     class Anjuke:
        address = "东方路1217号"
 4
        depart = "用户端"
         def __init__(self,*args):
 6
             if len(args) > 0:
 8
                 self.address = args[0]
 9
                 self.depart = args[1]
10
11
        def __del__(self):
12
             print "我被回收了"
13
14
        def print_attr(self):
15
             print self.address
16
             print self.depart
17
18
     if __name__ = '__main__':
19
        a = Anjuke()
20
        a.print_attr()
21
        #del a
22
        b = Anjuke()
        a.address = "陆家嘴"
23
        b.print_attr()
24
```

实例被销毁时调用

python的垃圾回收机制是 计算引用来进行的,把21 行的注释去掉再看看效果

#### getitem\_()

```
#encoding:utf-8
 2
 3
     class Anjuke:
 4
         1 = []
 5
         def __init__(self,*args):
 6
             for i in args:
                 self.l.append(i)
 8
 9
         def __getitem__(self, key):
10
             return self.l[key]
11
12
    if __name__ == '__main__':
13
         a = Anjuke('a', 'b', 'c')
         print a[2]
14
```

#### setitem\_()

```
#encoding:utf-8
     class Anjuke:
 4
         1 = \Pi
         def __init__(self,*args):
 6
             for i in args:
                 self.l.append(i)
 8
 9
         def __getitem__(self, key):
10
             return self.l[key]
11
         def __setitem__(self, key, value):
12
             if key > len(self.l)-1:
13
                 self.l.append(value)
14
             else:
15
                 self.l[key] = value
16
17
     if __name__ == '__main__':
18
         a = Anjuke('a', 'b', 'c')
19
         a[3] = 'string'
20
         a[1] = 'int'
21
         print a.l
22
```

getitem和setitem配合使用就可以使任何一个类模拟序列操作,既然是序列肯定可以应用到迭代上,下一节将演示\_\_iter\_\_()的用法

#### iter\_\_()

```
#encoding:utf-8
     class Anjuke:
         1 = []
         count = -1
         def __init__(self,*args):
             for i in args:
                 self.l.append(i)
         def __getitem__(self, key):
 8
             return self.l[key]
 9
         def __setitem__(self, key, value):
11
             if key > len(self.l)-1:
                 self.l.append(value)
12
13
             else:
14
                 self.l[key] = value
         def __iter__(self):
15
             return self
16
17
         def next(self):
             if Anjuke.count <= len(self.l):</pre>
18
                 Anjuke.count += 1
19
                 return self.l[Anjuke.count]
20
21
         def __len__(self):
22
             return len(self.l)
23
     if __name__ = '__main__':
         a = Anjuke('a', 'b', 'c')
24
         a[3] = 'string'
         for i in a:
             if i != 'string':
27
                 print i
28
29
             else:
                 break
30
31
         print len(a)
```

这里有两个值得注意的地方

- 1.\_\_len\_\_()方法
- 2.使用类.属性名的访问方式

#### \_\_getattr\_\_()

```
1
   #encoding:utf-8
2
3
   class Anjuke:
4
5
       def __init__(self):
6
            self.address = "东方路1217号"
8
       def __getattr__(self, name):
            print name, "不存在"
   if __name__ == '__main__':
       a = Anjuke()
       print a.address
       a.depart
```

此方法在访问不存在的属 性时调用

#### setattr\_\_()

```
#encoding:utf-8
 2
 3
    class Anjuke:
 4
 5
         def __init__(self):
 6
             self.address = "东方路1217号"
 8
         def __getattr__(self, name):
 9
             print name, "不存在"
10
         def __setattr__(self, name, value):
11
12
             self.__dict__[name] = value
13
    if __name__ == '__main__':
14
15
         a = Anjuke()
16
         a.depart = 'user'
17
         print a.__dict__
```

#### \_\_add\_\_()

```
1
     #encoding:utf-8
 2
 3
     class Anjuke:
 4
         def __init__(self, num):
             self.people = num
 6
         def __add__(self, other):
 8
             return self.people+other.people
10
     if __name__ = '__main__':
11
         a = Anjuke(5)
12
         b = Anjuke(10)
13
         print a+b
14
```

此方法是许多运算方法中的一种,其余的还有很多:\_\_eq\_\_(),\_\_abs\_\_(),\_\_sub\_\_\_() .etc

#### 静态方法和类方法

```
#encoding:utf-8
                                          #encoding:utf-8
    class Anjuke:
                                          class Anjuke:
       address = "东方路1217号"
                                      5
                                              address = "东方路1217号"
                                      6
       def __init__(self):
                                              def __init__(self):
           self.depart = "哈哈"
                                                  self.depart = "哈哈"
       @staticmethod
                                              @classmethod
                                     10
       def print_address():
                                              def print_depart(cls):
           print Anjuke.address
                                     11
                                     12
                                                  print cls.depart
       @staticmethod
                                     13
       def print_depart():
                                     14
                                              @classmethod
6
           print self.depart
                                              def print_address(cls):
                                     15
                                     16
                                                  print cls.address
   if __name__ == '__main__':
                                     17
       a = Anjuke()
                                     18
                                          if __name__ == '__main__':
       a.print_address()
                                     19
                                              Anjuke.print_address()
       Anjuke.print_address()
```

### 面向对象三大特性

- 封装
- 继承
- 多态

#### 對裝

```
#encoding:utf-8
 2
     class Anjuke(object):
 4
 5
         def __init__(self):
             self.__depart = "user"
 6
 7
 8
         def get_depart(self):
             if self.__depart == 'user':
 9
10
                 return self.__depart
11
             else:
12
                 return '不是用户端,不输出'
13
         def set_depart(self,attr):
14
15
             self.__depart = attr
16
17
         depart = property(get_depart,set_depart)
18
    if __name__ = '__main__':
19
20
        a = Anjuke()
21
        print a.depart
        a.depart = "mobile"
22
23
        print a.depart
```

```
#encoding:utf-8
     class Anjuke(object):
        def __init__(self):
            self.__depart = "user"
        @property
 8
        def depart(self):
9
             return self.__depart
10
11
12
        @depart.setter
        def depart(self, value):
13
            self.__depart = value
14
15
        @depart.getter
16
        def depart(self):
17
18
             if self.__depart = 'user':
19
                return self.__depart
20
            else:
21
                return '不是用户端,不输出'
22
        @depart.deleter
23
        def depart(self):
24
            print '不能删除'
25
26
    if __name__ - '__main__':
27
        a = Anjuke()
        a.depart = 'mobile'
28
        print a.depart
29
        del a.depart
30
```

```
#encoding:utf-8
 2
     class Person:
         def __init__(self):
             self.age = 20
             self.name = 'person'
         def say(self):
             print "I am %s" % self.age
 9
10
11
     class Anjuke(Person):
12
13
         def __init__(self):
             Person.__init__(self)
14
15
16
     class AnjukeB(Person):
17
         def __init__(self):
18
19
             Person.__init__(self)
20
             self.name = 'AnjukeB'
21
22
         def say(self):
23
             print self.name
24
25
     if __name__ - '__main__':
26
         a = Anjuke()
27
         print 'my name is',a.name
```

28

29

30

a.say()

b.say()

b = AnjukeB()

# 4 #encoding:ut 2 \_metaclass\_ 3 class Person

```
#encoding:utf-8
     __metaclass__ = type
     class Person:
         def __init__(self):
             self.age = 20
 6
             self.name = 'person'
         def say(self):
 8
             print "I am %s" % self.age
 9
10
11
     class Anjuke(Person):
12
13
         def __init__(self):
14
             super(Anjuke,self).__init__()
15
16
     class AnjukeB(Person):
17
18
         def __init__(self):
19
             Person.__init__(self)
20
             self.name = 'AnjukeB'
21
22
         def say(self):
23
             print self.name
24
    if __name__ == '__main__':
25
26
         a = Anjuke()
27
         print 'my name is',a.name
28
         a.say()
29
         b = AnjukeB()
30
         b.say()
```

#### 多重继承

```
#encoding:utf-8
     __metaclass__ = type
     class Person:
         def __init__(self):
             self.age = 20
             self.name = 'person'
 8
 9
         def say(self):
             print "I am %s" % self.age
11
     class Employee:
13
14
         def __init__(self):
             self.name = 'Employee'
15
16
         def get_salary(self):
17
             print "I want salary"
18
20
         def say(self):
21
             print "I am Employee"
22
23
     class Anjuke(Person, Employee):
24
         def __init__(self):
25
             super(Anjuke, self).__init__()
26
27
28
     if __name__ - '__main__':
29
         a = Anjuke()
         print a.name
30
31
         a.say()
```

注意父类的顺序 ,比较代码的不同之处

```
#encoding:utf-8
    __metaclass__ = type
     class Person:
         def __init__(self):
             self.age = 20
             self.name = 'person'
        def say(self):
             print "I am %s" % self.age
10
11
12
     class Employee:
14
         def __init__(self):
             self.name = 'Employee'
17
         def get_salary(self):
             print "I want salary"
20
         def say(self):
21
             print "I am Employee"
22
     class Anjuke(Employee, Person):
         def __init__(self):
24
             super(Anjuke, self).__init__()
25
26
    if __name__ = '__main__':
28
29
         a = Anjuke()
        print a.name
         a.say()
```

#### 多态

```
#encoding:utf-8
 1
 2
    __metaclass__ = type
     class Anjuke:
         def count(self, arg):
 6
             return arg.count('a')
 8
     if __name__ == '__main__':
 9
         a = Anjuke()
10
         print a.count("abcdefa")
11
12
         print a.count(['a','b'])
```

python是动态语言,自身就带有多态特性,最直接的例子就是'+'运算符,可以作用于字符串,数值甚至是类实例上

一个显而易见的多态例子

```
#encoding:utf-8
     __metaclass__ = type
 3
 4
     class A:
         def say(self):
             print "I am A"
 6
     class B(A):
         pass
10
     class C:
11
12
         def say(self):
             print "I am C"
13
14
     class Anjuke:
15
16
         def mul(self,obj):
17
             obj.say()
18
19
     if __name__ = '__main__':
20
21
         an = Anjuke()
22
         an.mul(A())
23
         an.mul(B())
24
         an.mul(C())
```

```
#encoding:utf-8
    __metaclass__ = type
    class User:
        def say(self):
            print "I am User"
 6
    class Web:
        def say(self):
            print "I am Web"
11
    class Mobile:
13
        def say(self):
14
            print "I am Mobile"
15
    class AnjukeFactory:
         def produce(self, type):
             if type = 'User':
20
                 return User()
21
             elif type == 'Web':
22
                 return Web()
23
            else:
                 return Mobile()
25
26
    if __name__ = '__main__':
27
        an = AnjukeFactory()
28
        an.produce('User').say()
29
        an.produce('Web').say()
30
        an.produce('Mobile').say()
```