原创 推荐

面向对象编程其实很简单——Python 面向对象（初级篇）

alex3714 关注 2015-08-28 09:31:35 20776人阅读 7人评论

博客园同步发布地址：http://www.cnblogs.com/wupeiqi/articles/4493506.html

在Python教学中发现，很多同学在走到面向对象编程这块就开始蒙圈了，为了帮助大家更好的理解面向对象编程并其能将其用到自己的开发过程中，特写此文。

概述

- 面向过程：根据业务逻辑从上到下写垒代码
- 函数式：将某功能代码封装到函数中，日后便无需重复编写，仅调用函数即可
- 面向对象：对函数进行分类和封装，让开发“更快更好更强...”

面向过程编程最易被初学者接受，其往往用一长段代码来实现指定功能，开发过程中最常见的操作就是粘贴复制，即：将之前实现的代码块复制到现需功能处。

```
while True：
    if cpu利用率 > 90%:
        #发送邮件提醒
        连接邮箱服务器
        发送邮件
        关闭连接

    if 硬盘使用空间 > 90%:
        #发送邮件提醒
        连接邮箱服务器
        发送邮件
        关闭连接

    if 内存占用 > 80%:
        #发送邮件提醒
        连接邮箱服务器
        发送邮件
        关闭连接
```

随着时间的推移，开始使用了函数式编程，增强代码的重用性和可读性，就变成了这样：

```
def 发送邮件(内容)
    #发送邮件提醒
    连接邮箱服务器
    发送邮件
    关闭连接
```

```
if cpu利用率 > 90%:
    发送邮件('CPU报警')

if 硬盘使用空间 > 90%:
    发送邮件('硬盘报警')

if 内存占用 > 80%:
    发送邮件('内存报警')
```

今天我们来学习一种新的编程方式：面向对象编程（Object Oriented Programming，OOP，面向对象程序设计）

注：Java和C#来说只支持面向对象编程，而python比较灵活即支持面向对象编程也支持函数式编程

创建类和对象

面向对象编程是一种编程方式，此编程方式的落地需要使用 “类” 和 “对象” 来实现，所以，面向对象编程其实就是对 “类” 和 “对象” 的使用。

类就是一个模板，模板里可以包含多个函数，函数里实现一些功能

对象则是根据模板创建的实例，通过实例对象可以执行类中的函数

- class是关键字，表示类
- 创建对象，类名称后加括号即可

ps：类中的函数第一个参数必须是self（详细见：类的三大特性之封装）
类中定义的函数叫做“方法”

```
# 创建类
class Foo:

    def Bar(self):
        print 'Bar'

    def Hello(self, name):
        print 'i am %s' %name

# 根据类Foo创建对象obj
obj = Foo()
obj.Bar()               #执行Bar方法
obj.Hello('wupeiqi') #执行Hello方法
```

诶，你在这里是不是有疑问了？使用函数式编程和面向对象编程方式来执行一个“方法”时函数要比面向对象简便

- *面向对象：【创建对象】【通过对象执行方法】*
- *函数编程：【执行函数】*

观察上述对比答案则是肯定的，然后并非绝对，场景的不同适合其的编程方式也不同。

总结：函数式的应用场景 --> 各个函数之间是独立且无共用的数据

面向对象三大特性

面向对象的三大特性是指：封装、继承和多态。

一、封装

封装，顾名思义就是将内容封装到某个地方，以后再去调用被封装在某处的内容。

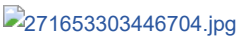
所以，在使用面向对象的封装特性时，需要：

- 将内容封装到某处
- 从某处调用被封装的内容

self 是一个形式参数，当执行 obj1 = Foo('wupeiqi', 18) 时，self 等于 obj1

当执行 obj2 = Foo('alex', 78) 时，self 等于 obj2

所以，内容其实被封装到了对象 obj1 和 obj2 中，每个对象中都封装了 name 和 age ，之前说的"内容封装到某处"其在内容里类似于下图来保存。



第二步：从某处调用被封装的内容

调用被封装的内容时，有两种情况：

- 通过对象直接调用
- 通过self间接调用

1、通过对象直接调用被封装的内容

上图展示了对象 obj1 和 obj2 在内存中保存的方式，根据保存格式可以如此调用被封装的内容：对象.属性名

```
class Foo:

    def __init__(self, name, age):
        self.name = name
        self.age = age

obj1 = Foo('wupeiqi', 18)
print obj1.name     # 直接调用obj1对象的name属性
print obj1.age      # 直接调用obj1对象的age属性

obj2 = Foo('alex', 73)
print obj2.name     # 直接调用obj2对象的name属性
print obj2.age      # 直接调用obj2对象的age属性
```

2、通过self间接调用被封装的内容

执行类中的方法时，需要通过self间接调用被封装的内容

```
class Foo:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def detail(self):
        print self.name
        print self.age

obj1 = Foo('wupeiqi', 18)
obj1.detail()  # Python默认会将obj1传给self参数，即：obj1.detail(obj1)，所以，此时方法内部的 self = c

obj2 = Foo('alex', 73)
obj2.detail()  # Python默认会将obj2传给self参数，即：obj1.detail(obj2)，所以，此时方法内部的 self = c
```

综上所述，对于面向对象的封装来说，其实就是使用构造方法将内容封装到 对象 中，然后通过对象直接或者self间接获取被封装的内容。

练习一：在终端输出如下信息

- 小明，10岁，男，上山去砍柴
- 小明，10岁，男，开车去东北
- 小明，10岁，男，最爱大保健
-
- 老李，90岁，男，上山去砍柴
- 老李，90岁，男，开车去东北

- 老张...

函数式编程

```
def kanchai(name, age, gender):
    print "%s,%s岁,%s,上山去砍柴" %(name, age, gender)


def qudongbei(name, age, gender):
    print "%s,%s岁,%s,开车去东北" %(name, age, gender)


def dabaojian(name, age, gender):
    print "%s,%s岁,%s,最爱大保健" %(name, age, gender)


kanchai('小明', 10, '男')
qudongbei('小明', 10, '男')
dabaojian('小明', 10, '男')


kanchai('老李', 90, '男')
qudongbei('老李', 90, '男')
dabaojian('老李', 90, '男')
```

```
class Foo:

    def __init__(self, name, age ,gender):
        self.name = name
        self.age = age
        self.gender = gender

    def kanchai(self):
        print "%s,%s岁,%s,上山去砍柴" %(self.name, self.age, self.gender)

    def qudongbei(self):
        print "%s,%s岁,%s,开车去东北" %(self.name, self.age, self.gender)

    def dabaojian(self):
        print "%s,%s岁,%s,最爱大保健" %(self.name, self.age, self.gender)


xiaoming = Foo('小明', 10, '男')
xiaoming.kanchai()
xiaoming.qudongbei()
xiaoming.dabaojian()

laoli = Foo('小明', 10, '男')
laoli.kanchai()
laoli.qudongbei()
laoli.dabaojian()
```

上述对比可以看出，如果使用函数式编程，需要在每次执行函数时传入相同的参数，如果参数多的话，又需要粘贴复制了... ；而对于面向对象只需要在创建对象时，将所有需要的参数封装到当前对象中，之后再次使用时，通过self间接去当前对象中取值即可。

练习二：游戏人生程序

1、创建三个游戏人物，分别是：

- 苍井井，女，18，初始战斗力1000
- 东尼木木，男，20，初始战斗力1800
- 波多多，女，19，初始战斗力2500

- 自我修炼，增长100战斗力
- 多人游戏，消耗500战斗力

```
# -*- coding:utf-8 -*-

# ##################### 定义实现功能的类 #####################

class Person:

    def __init__(self, na, gen, age, fig):
        self.name = na
        self.gender = gen
        self.age = age
        self.fight =fig

    def grassland(self):
        """注释：草丛战斗，消耗200战斗力"""

        self.fight = self.fight - 200

    def practice(self):
        """注释：自我修炼，增长100战斗力"""

        self.fight = self.fight + 200

    def incest(self):
        """注释：多人游戏，消耗500战斗力"""

        self.fight = self.fight - 500

    def detail(self):
        """注释：当前对象的详细情况"""

        temp = "姓名:%s ; 性别:%s ; 年龄:%s ; 战斗力:%s"  % (self.name, self.gender, self.age, self.
        print temp


# ##################### 开始游戏 #####################

cang = Person('苍井井', '女', 18, 1000)     # 创建苍井井角色
dong = Person('东尼木木', '男', 20, 1800)   # 创建东尼木木角色
bo = Person('波多多', '女', 19, 2500)       # 创建波多多角色

cang.incest() #苍井空参加一次多人游戏
dong.practice()#东尼木木自我修炼了一次
bo.grassland() #波多多参加一次草丛战斗


#输出当前所有人的详细情况
cang.detail()
dong.detail()
bo.detail()


cang.incest() #苍井空又参加一次多人游戏
dong.incest() #东尼木木也参加了一个多人游戏
bo.practice() #波多多自我修炼了一次

#输出当前所有人的详细情况
cang.detail()
dong.detail()
bo.detail()
```

游戏人生

二、继承

继承，面向对象中的继承和现实生活中的继承相同，即：子可以继承父的内容。

例如：

猫可以：喵喵叫、吃、喝、拉、撒

狗可以：汪汪叫、吃、喝、拉、撒

如果我们要分别为猫和狗创建一个类，那么就需要为 猫 和 狗 实现他们所有的功能，如下所示：

伪代码

```
class 猫：

    def 喵喵叫(self):
        print '喵喵叫'

    def 吃(self):
        # do something

    def 喝(self):
        # do something

    def 拉(self):
        # do something

    def 撒(self):
        # do something

class 狗：

    def 汪汪叫(self):
        print '喵喵叫'

    def 吃(self):
        # do something

    def 喝(self):
        # do something

    def 拉(self):
        # do something

    def 撒(self):
        # do something
```

上述代码不难看出，吃、喝、拉、撒是猫和狗都具有的功能，而我们却分别的猫和狗的类中编写了两次。如果使用 继承 的思想，如下实现：

动物：吃、喝、拉、撒

猫：喵喵叫（猫继承动物的功能）

狗：汪汪叫（狗继承动物的功能）

伪代码

```
class 动物:

    def 吃(self):
        # do something

    def 喝(self):
        # do something

    def 拉(self):
```

```
        # do something

# 在类后面括号中写入另外一个类名，表示当前类继承另外一个类
class 猫(动物):

    def 喵喵叫(self):
        print '喵喵叫'

# 在类后面括号中写入另外一个类名，表示当前类继承另外一个类
class 狗(动物):

    def 汪汪叫(self):
        print '喵喵叫'
```

代码实例

```
class Animal:

    def eat(self):
        print "%s 吃 " %self.name

    def drink(self):
        print "%s 喝 " %self.name

    def shit(self):
        print "%s 拉 " %self.name

    def pee(self):
        print "%s 撒 " %self.name


class Cat(Animal):

    def __init__(self, name):
        self.name = name
        self.breed = '猫'

    def cry(self):
        print '喵喵叫'

class Dog(Animal):

    def __init__(self, name):
        self.name = name
        self.breed = '狗'

    def cry(self):
        print '汪汪叫'


# ######### 执行 #########

c1 = Cat('小白家的小黑猫')
c1.eat()

c2 = Cat('小黑的小白猫')
c2.drink()

d1 = Dog('胖子家的小瘦狗')
d1.eat()
```

所以，对于面向对象的继承来说，其实就是将多个类共有的方法提取到父类中，子类仅需继承父类而不必一一实现每个方法。

学习了继承的写法之后，我们用代码来是上述阿猫阿狗的功能：

代码实例

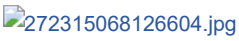
```
class Animal:

    def eat(self):
        print "%s 吃 " %self.name

    def drink(self):
        print "%s 喝 " %self.name

    def shit(self):
        print "%s 拉 " %self.name

    def pee(self):
        print "%s 撒 " %self.name


class Cat(Animal):

    def __init__(self, name):
        self.name = name
        self.breed = '猫'

    def cry(self):
        print '喵喵叫'

class Dog(Animal):

    def __init__(self, name):
        self.name = name
        self.breed = '狗'

    def cry(self):
        print '汪汪叫'


# ######### 执行 #########

c1 = Cat('小白家的小黑猫')
c1.eat()

c2 = Cat('小黑的小白猫')
c2.drink()

d1 = Dog('胖子家的小瘦狗')
d1.eat()
```

那么问题又来了，多继承呢?

- 是否可以继承多个类
- 如果继承的多个类每个类中都定了相同的函数，那么那一个会被使用呢?

1、Python的类可以继承多个类，Java和C#中则只能继承一个类

2、Python的类如果继承了多个类，那么其寻找方法的方式有两种，分别是：**深度优先**和**广度优先**



- 当类是新式类时，多继承情况下，会按照广度优先方式查找

经典类和新式类，从字面上可以看出一个老一个新，新的必然包含了跟多的功能，也是之后推荐的写法，从写法上区分的话，如果 **当前类或者父类继承了object类**，那么该类便是新式类，否则便是经典类。

经典类多继承

```
class D:

    def bar(self):
        print 'D.bar'


class C(D):

    def bar(self):
        print 'C.bar'


class B(D):

    def bar(self):
        print 'B.bar'


class A(B, C):

    def bar(self):
        print 'A.bar'

a = A()
# 执行bar方法时
# 首先去A类中查找，如果A类中没有，则继续去B类中找，如果B类中么有，则继续去D类中找，如果D类中
# 所以，查找顺序：A --> B --> D --> C
# 在上述查找bar方法的过程中，一旦找到，则寻找过程立即中断，便不会再继续找了
a.bar()
```

新式类多继承

```
class D(object):

    def bar(self):
        print 'D.bar'


class C(D):

    def bar(self):
        print 'C.bar'


class B(D):

    def bar(self):
        print 'B.bar'


class A(B, C):

    def bar(self):
```

```
# 执行bar方法时
# 首先去A类中查找，如果A类中没有，则继续去B类中找，如果B类中么有，则继续去C类中找，如果C类中
# 所以，查找顺序：A --> B --> C --> D
# 在上述查找bar方法的过程中，一旦找到，则寻找过程立即中断，便不会再继续找了
a.bar()
```

经典类：首先去**A**类中查找，如果A类中没有，则继续去**B**类中找，如果B类中么有，则继续去**D**类中找，如果D类中么有，则继续去**C**类中找，如果还是未找到，则报错

新式类：首先去**A**类中查找，如果A类中没有，则继续去**B**类中找，如果B类中么有，则继续去**C**类中找，如果C类中么有，则继续去**D**类中找，如果还是未找到，则报错

注意：在上述查找过程中，一旦找到，则寻找过程立即中断，便不会再继续找了

三、多态

Pyhon不支持多态并且也用不到多态，多态的概念是应用于Java和C#这一类强类型语言中，而Python崇尚"鸭子类型"。

Python伪代码实现Java或C#的多态

```
class F1:
    pass


class S1(F1):

    def show(self):
        print 'S1.show'


class S2(F1):

    def show(self):
        print 'S2.show'


# 由于在Java或C#中定义函数参数时，必须指定参数的类型
# 为了让Func函数既可以执行S1对象的show方法，又可以执行S2对象的show方法，所以，定义了一个S1和
# 而实际传入的参数是：S1对象和S2对象

def Func(F1 obj):
    """Func函数需要接收一个F1类型或者F1子类的类型"""

    print obj.show()

s1_obj = S1()
Func(s1_obj) # 在Func函数中传入S1类的对象 s1_obj，执行 S1 的show方法，结果：S1.show

s2_obj = S2()
Func(s2_obj) # 在Func函数中传入Ss类的对象 ss_obj，执行 Ss 的show方法，结果：S2.show
```

Python "鸭子类型"

```
class F1:
    pass


class S1(F1):

    def show(self):
        print 'S1.show'


class S2(F1):
```

```
        print 'S2.show'

def Func(obj):
    print obj.show()

s1_obj = S1()
Func(s1_obj)

s2_obj = S2()
Func(s2_obj)
```

有兴趣的同学可以加入我的Python自动化讨论群（29215534），共同学习呵呵。。。。

尽请期待之后的面向对象的深入篇...

Python 面向对象编程

2楼 2015-08-28 20:45:28

没有从头学起，这部分完全看不懂。oldboy的群里就不要再发这些了。|@|没有点开发的底子，我是接受不了。

lxf1992521
3楼 2015-09-02 13:10:32

很棒的教程，一直对oop不太熟，看了之后茅塞顿开。|@|另外，能不能讲解一下继承中子类对父类属性和方法的“重写”？

liuyi20080607
4楼 2015-09-02 15:58:17

多谢分享，我这几年一直停留在函数式编程上，你这篇文章，给了我很大影响，谢谢

chenjia107
5楼 2015-09-06 22:36:26

类 继承 多态这些是面向对象的基础。这部分内容如果要学好，建议还是弄本书好好看吧。我大学时自学过一点C + +基础，感觉都是一样的，没觉得有啥接受不了的。

九叔
6楼 2015-09-12 21:02:34

图文并茂，楼主写得好，mark一下

xiao_tao123
7楼 2016-01-21 17:12:49

非常好的博文，学习了！

pyecharts在数据可视化中的应用详解

Python语言基础概论

从tcp开始，用Python写一个web框架1

项目再复杂我都没哭！帮新来的同事安装开发环境，结…

Python网络爬虫实战：世纪佳缘爬取近6万条小姐姐数据…

用Python选一个自己的股票池2

好课推荐

更多

Python编程的术与道：Python语言入门

1236人学习

免费试看

20天学习Python开发④Python常用模块

843人学习

免费试看

Python开发全教程

59581人学习

免费试看

Python之Python学习手册与Python核心编程专题

258人学习

免费试看

在线客服

26 13 7 分享

alex3714 关注