

公告

最新自学视频 路飞Python免费小课  
人生迷茫问题可以加Alex个人微信听鸡



汤



面向对象开发原来如此简单

16人在学

进阶 12小时



Python开发中最常用的11个模块精讲

10人在学

进阶 6小时



跟随Alex金角大王3周上手Python开发

124人在学

入门 19小时

昵称: 金角大王  
园龄: 5年5个月  
粉丝: 10882  
关注: 5  
-取消关注

## Python之路,Day6 - 面向对象学习

本节内容:

面向对象编程介绍

为什么要用面向对象进行开发?

面向对象的特性: 封装、继承、多态

类、方法、

### 引子

你现在是一家游戏公司的开发人员, 现在需要你开发一款叫做<人狗大战>的游戏, 你就思考呀, 人狗作战, 那至少需要2个角色, 一个是人, 一个是狗, 且人和狗都有不同的技能, 比如人拿棍打狗, 狗可以咬人, 怎么描述这种不同的角色和他们的功能呢?

你搜索了自己掌握的所有技能, 写出了下面的代码来描述这两个角色

```
1 def person(name, age, sex, job):
2     data = {
3         'name': name,
4         'age': age,
5         'sex': sex,
6         'job': job
7     }
8
9     return data
10
11 def dog(name, dog_type):
12     data = {
13         'name': name,
14         'type': dog_type
15     }
16     return data
```

上面两个方法相当于造了两个模子, 游戏开始, 你得生成一个人和狗的实际对象吧, 怎么生成呢?

```
1 d1 = dog("李磊", "京巴")
2
3 p1 = person("严帅", 36, "F", "运维")
4
5 p2 = person("林海峰", 27, "F", "Teacher")
```

两个角色对象生成了, 狗和人还有不同的功能呀, 狗会咬人, 人会打狗, 对不对? 怎么实现呢,。。想到了, 可以每个功能再写一个函数, 想执行哪个功能, 直接 调用 就可以了, 对不?

```
1 def bark(d):
2     print("dog %s:wang.wang..wang..." % d['name'])
3
4
5 def walk(p):
6     print("person %s is walking..." % p['name'])<br><br>
7
8 d1 = dog("李磊", "京巴")
```

随笔 - 29 文章 - 64 评论 - 929

Fuck me on GitHub

## 我的标签

职业发展(3)

创业(2)

## 随笔分类

□职业&amp;生活随笔(22)

## 文章分类

Python全栈开发之路(12)

Python学习目录(4)

Python自动化开发之路(33)

爬虫(6)

## 最新评论

1. Re:Django 14天从小白到进阶- Day3

搞定Views组件

不更新了吗

--30岁老古董

2. Re:编程要自学或报班这事你都想不明白

白, 那必然是你智商不够

大王性格直爽, 有目标, 肯付出行动! 偶像! 我要是早几年遇见大王就好了, 现在都已经三十了, 浪费了大好的青春!

--Xiyue666

3. Re:python 之路, Day11 - python

mysql and ORM

ALTER mytable ADD INDEX

[indexName] ON (username(length))

这句应改为: alter table mytable add index index...

--原竹

4. Re:Python之路,Day3 - Python基础3

@我的恋人叫臭臭 淫角大王听说很厉害

吧? ...

--Xiyue666

5. Re:Python之路,Day1 - Python基础1

哎 我为什么不早点遇到老男孩 遇到alex 老师呢!

--Xiyue666

## 阅读排行榜

1. python 之路, 200行Python代码写了个打飞机游戏! (52516)

2. Django + Uwsgi + Nginx 实现生产环境部署(31781)

3. Python Select 解析(27173)

4. 为什么很多IT公司不喜欢进过培训机构的人呢? (20543)

5. 编程要自学或报班这事你都想不明白, 那必然是你智商不够(16923)

## 推荐排行榜

1. 给一位做技术迷茫的同学回信(63)

2. 你做了哪些事, 导致老板下调了对你的评价? (51)

```
2 | p1 = person("严帅",36,"F","运维")
3 | p2 = person("林海峰",27,"F","Teacher")

1 | walk(p1) bark(d1)
```

上面的功能实现的简直是完美!

但是仔细玩耍一会, 你就不小心干了下面这件事

```
1 | p1 = person("严帅",36,"F","运维")
2 | bark(p1) #把人的对象传给了狗的方法
```

事实上, 这并没出错。很显然, 人是不能调用狗的功能的, 如何在代码级别实现这个限制呢?

```
1 | def person(name,age,sex,job):
2 |     def walk(p):
3 |         print("person %s is walking..." % p['name'])
4 |
5 |     data = {
6 |         'name':name,
7 |         'age':age,
8 |         'sex':sex,
9 |         'job':job,
10 |        'walk':walk
11 |    }
12 |
13 |    return data
14 |
15 | def dog(name,dog_type):
16 |
17 |
18 |     def bark(d):
19 |         print("dog %s:wang.wang..wang..."%d['name'])
20 |     data = {
21 |         'name':name,
22 |         'type':dog_type,
23 |         'bark':bark
24 |     }
25 |
26 |    return data

1 | d1 = dog("李磊","京巴")
2 | p1 = person("严帅",36,"F","运维")
3 | p2 = person("林海峰",27,"F","Teacher")<br><br>

1 | d1['bark'](p1) #把人的对象传给了狗的方法
```

你是如此的机智, 这样就实现了限制人只能用人自己的功能啦。

但, 我的哥, 不要高兴太早, 刚才你只是阻止了两个完全不同的角色 之前的功能混用, 但有没有可能, 同一个角色, 但有些属性是不同的呢? 比如, 大家都打过cs吧, cs里有警察和恐怖份子, 但因为都是人, 所以你写一个角色叫 person(), 警察和恐怖份子都可以互相射击, 但警察不可以杀人质, 恐怖分子可以, 这怎么实现呢? 你想了想, 说, 简单, 只需要在杀人质的功能里加个判断, 如果是警察, 就不让杀不就ok了么。没错, 这虽然解决了杀人质的问题, 但其实你会发现, 警察和恐怖分子的区别还有很多, 同时又有很多共性, 如果在每个区别处都单独做判断, 那得累死。

你想了想, 那就直接写2个角色吧, 反正这么多区别, 我的哥, 不能写两个角色呀, 因为他们还有很多共性, 写两个不同的角色, 就代表相同的功能也要重写了, 是不是我的哥? ...

好了, 话题就给你点到这, 再多说你的智商也理解不了了!

3. 关于认识、格局、多维度发展的感触(46)
4. 为什么很多IT公司不喜欢进过培训机构的人呢? (37)
5. 编程要自学或报班这事你都想不明白,那必然是你智商不够(35)

## 面向过程 VS 面向对象

### 编程范式

编程是 程序员 用特定的语法+数据结构+算法组成的代码来告诉计算机如何执行任务的过程，一个程序是程序员为了得到一个任务结果而编写的一组指令的集合，正所谓条条大路通罗马，实现一个任务的方式有很多种不同的方式，对这些不同的编程方式的特点进行归纳总结得出来的编程方式类别，即为编程范式。不同的编程范式本质上代表对各种类型的任务采取的不同的解决问题的思路，大多数语言只支持一种编程范式，当然也有些语言可以同时支持多种编程范式。两种最重要的编程范式分别是面向过程编程和面向对象编程。

### 面向过程编程(Procedural Programming)

Procedural programming uses a list of instructions to tell the computer what to do step-by-step.

面向过程编程依赖 - 你猜到了 - procedures，一个procedure包含一组要被进行计算的步骤，面向过程又被称为top-down languages，就是程序从上到下一步步执行，一步步从上到下，从头到尾的解决问题。基本设计思路就是程序一开始是要着手解决一个大的问题，然后把一个大问题分解成很多个小问题或子过程，这些子过程再执行的过程再继续分解直到小问题足够简单到可以在一个小步骤范围内解决。

举个典型的面向过程的例子，数据库备份，分三步，连接数据库，备份数据库，测试备份文件可用性。

代码如下

```
1  def db_conn():
2      print("connecting db...")
3
4
5  def db_backup(dbname):
6      print("导出数据库...", dbname)
7      print("将备份文件打包, 移至相应目录...")
8
9  def db_backup_test():
10     print("将备份文件导入测试库, 看导入是否成功")
11
12
13 def main():
14     db_conn()
15     db_backup('my_db')
16     db_backup_test()
17
18
19
20 if __name__ == '__main__':
21     main()
```

这样做的问题也是显而易见的，就是如果你要对程序进行修改，对你修改的那部分有依赖的各个部分你都要也跟着修改，举个例子，如果程序开头你设置了一个变量值为1，但如果其它子过程依赖这个值为1的变量才能正常运行，那如果你改了这个变量，那这个子过程你也要修改，假如又有一个其它子程序依赖这个子过程，那就会发生一连串的影响，随着程序越来越大，这种编程方式的维护难度会越来越高。

所以我们一般认为，如果你只是写一些简单的脚本，去做一些一次性任务，用面向过程的方式是极好的，但如果你要处理的任务是复杂的，且需要不断迭代和维护的，那还是用面向对象最方便了。

## 面向对象编程

OOP编程是利用“类”和“对象”来创建各种模型来实现对真实世界的描述，使用面向对象编程的原因一方面是因为它可以使程序的维护和扩展变得更简单，并且可以大大提高程序开发效率，另外，基于面向对象的程序可以使它人更加容易理解你的代码逻辑，从而使团队开发变得更从容。

面向对象的几个核心特性如下

### Class 类

一个类即是对一类拥有相同属性的对象的抽象、蓝图、原型。在类中定义了这些对象的都具备的属性（variables(data)）、共同的方法

### Object 对象

一个对象即是一个类的实例化后实例，一个类必须经过实例化后方可在程序中调用，一个类可以实例化多个对象，每个对象亦可以有不同的属性，就像人类是指所有人，每个人是指具体的对象，人与人之前有共性，亦有不同

### Encapsulation 封装

在类中对数据的赋值、内部调用对外部用户是透明的，这使类变成了一个胶囊或容器，里面包含着类的数据和方法

### Inheritance 继承

一个类可以派生出子类，在这个父类里定义的属性、方法自动被子类继承

### Polymorphism 多态

多态是面向对象的重要特性,简单点说:“一个接口,多种实现”,指一个基类中派生出了不同的子类,且每个子类在继承了同样的方法名的同时又对父类的方法做了不同的实现,这就是同一种事物表现出的多种形态。

编程其实就是一个将具体世界进行抽象化的过程，多态就是抽象化的一种体现，把一系列具体事物的共同点抽象出来，再通过这个抽象的事物，与不同的具体事物进行对话。

对不同类的对象发出相同的消息将会有不同的行为。比如，你的老板让所有员工在九点钟开始工作，他只要在九点钟的时候说：“开始工作”即可，而不需要对销售人员说：“开始销售工作”，对技术人员说：“开始技术工作”，因为“员工”是一个抽象的事物，只要是员工就可以开始工作，他知道这一点就行了。至于每个员工，当然会各司其职，做各自的工作。

多态允许将子类的对象当作父类的对象使用，某父类型的引用指向其子类型的对象,调用的方法是该子类型的方法。这里引用和调用方法的代码编译前就已经决定了,而引用所指指向的对象可以在运行期间动态绑定

## 面向对象编程(Object-Oriented Programming)介绍

对于编程语言的初学者来讲，OOP不是一个很容易理解的编程方式，大家虽然都按老师讲的都知道OOP的三大特性是继承、封装、多态，并且大家也都知道了如何定义类、方法等面向对象的常用语法，但是一到真正写程序的时候，还是很多人喜欢用函数式编程来写代码，特别是初学者，很容易陷入一个窘境就是“我知道面向对象，我也会写类，但我依然没发现在使用了面向对象后，对我们的程序开发效率或其它方面带来什么好处，因为我使用函数编程就可以减少重复代码并做到程序可扩展了，为啥子还用面向对象？”。对于此，我个人觉得原因应该还是因为你没有充分了解到面向对象能带来的好处，今天我就写一篇关于面向对象的入门文章，希望能帮大家更好的理解和使用面向对象编程。

无论用什么形式来编程，我们都要明确记住以下原则：

1. 写重复代码是非常不好的低级行为
2. 你写的代码需要经常变更

开发正规的程序跟那种写个运行一次就扔了的小脚本一个很大不同就是，你的代码总是需要不断的更改，不是修改bug就是添加新功能等，所以为了日后方便程序的修改及扩展，你写的代码一定要遵循易读、易改的原则（专业数据叫可读性好、易扩展）。

如果你把一段同样的代码复制、粘贴到了程序的多个地方以实现在程序的各个地方调用 这个功能,那日后你再对这个功能进行修改时,就需要把程序里多个地方都改一遍,这种写程序的方式是有问题的,因为如果你不小心漏掉了一个地方没改,那可能会导致整个程序的运行都 出问题。因此我们知道 在开发中一定要努力避免写重复的代码,否则就相当于给自己再挖坑。

还好,函数的出现就能帮我们轻松的解决重复代码的问题,对于需要重复调用的功能,只需要把它写成一个函数,然后在程序的各个地方直接调用这个函数名就好了,并且当需要修改这个功能时,只需改函数代码,然后整个程序就都更新了。

其实OOP编程的主要作用也是使你的代码修改和扩展变的更容易,那么小白要问了,既然函数都能实现这个需求了,还要OOP干毛线用呢? 呵呵,说这话就像,古时候,人们打仗杀人都用刀,后来出来了枪,它的主要功能跟刀一样,也是杀人,然后小白就问,既然刀能杀人了,那还要枪干毛线,哈哈,显而易见,因为枪能更好更快更容易的杀人。函数编程与OOP的主要区别就是OOP可以使程序更加容易扩展和易更改。

小白说,我读书少,你别骗我,口说无凭,证明一下,好吧,那我们就下面的例子证明给小白看。

相信大家都打过CS游戏吧,我们就自己开发一个简单版的CS来玩一玩。



暂不考虑开发场地等复杂的东西,我们先从人物角色下手,角色很简单,就两个,恐怖份子、警察,他们除了角色不同,其它基本都一样,每个人都有生命值、武器等。咱们先用非OOP的方式写出游戏的基本角色

```
1 #role 1
2 name = 'Alex'
3 role = 'terrorist'
4 weapon = 'AK47'
5 life_value = 100
6
7 #role 2
8 name2 = 'Jack'
9 role2 = 'police'
10 weapon2 = 'B22'
11 life_value2 = 100
```

上面定义了一个恐怖份子Alex和一个警察Jack,但只2个人不好玩呀,一干就死了,没意思,那我们再分别一个恐怖分子和警察吧,

```
1 #role 1
```



```

2   name = 'Alex'
3   role = 'terrorist'
4   weapon = 'AK47'
5   life_value = 100
6   money = 10000
7
8   #rolw 2
9   name2 = 'Jack'
10  role2 = 'police'
11  weapon2 = 'B22'
12  life_value2 = 100
13  money2 = 10000
14
15  #role 3
16  name3 = 'Rain'
17  role3 = 'terrorist'
18  weapon3 = 'C33'
19  life_value3 = 100
20  money3 = 10000
21
22  #rolw 4
23  name4 = 'Eric'
24  role4 = 'police'
25  weapon4 = 'B51'
26  life_value4 = 100
27  money4 = 10000

```

4个角色虽然创建好了，但是有个问题就是，每创建一个角色，我都要单独命名，name1,name2,name3,name4...，后面的调用的时候这个变量名你还都得记着，要是再让多加几个角色，估计调用时就很容易弄混啦，所以我们想一想，能否所有的角色的变量名都是一样的，但调用的时候又能区分开分别是谁？

当然可以，我们只需要把上面的变量改成字典的格式就可以啦。

```

1   roles = {
2       1:{'name':'Alex',
3         'role':'terrorist',
4         'weapon':'AK47',
5         'life_value': 100,
6         'money': 15000,
7         },
8       2:{'name':'Jack',
9         'role':'police',
10        'weapon':'B22',
11        'life_value': 100,
12        'money': 15000,
13        },
14       3:{'name':'Rain',
15         'role':'terrorist',
16         'weapon':'C33',
17         'life_value': 100,
18         'money': 15000,
19         },
20       4:{'name':'Eirc',
21         'role':'police',
22         'weapon':'B51',
23         'life_value': 100,
24         'money': 15000,
25         },
26   }
27
28   print(roles[1]) #Alex
29   print(roles[2]) #Jack

```

很好，这个以后调用这些角色时只需要roles[1],roles[2]就可以啦，角色的基本属性设计完了后，我们接下来为每个角色开发以下几个功能

1. 被打中后就会掉血的功能
2. 开枪功能
3. 换子弹
4. 买枪
5. 跑、走、跳、下蹲等动作
6. 保护人质（仅适用于警察）
7. 不能杀同伴
8. ...

我们可以把每个功能写成一个函数，类似如下：

```

1  def shot(by_who):
2      #开了枪后要减子弹数
3      pass
4  def got_shot(who):
5      #中枪后要减血
6      who['life_value'] -= 10
7      pass
8  def buy_gun(who,gun_name):
9      #检查钱够不够,买了枪后要扣钱
10     pass
11 ...

```

so far so good, 继续按照这个思路设计，再完善一下代码，游戏的简单版就出来了，但是在往下走之前，我们来看看上面的这种代码写法有没有问题，至少从上面的代码设计中，我看到以下几点缺陷：

1. 每个角色定义的属性名称是一样的，但这种命名规则是我们自己约定的，从程序上来讲，并没有进行属性合法性检测，也就是说role 1定义的代表武器的属性是weapon, role 2,3,4也是一样的，不过如果我在新增一个角色时不小心把weapon写成了wepon，这个程序本身是检测不到的
2. terrorist 和police这2个角色有些功能是不同的，比如police是不能杀人质的，但是terrorist可能，随着这个游戏开发的更复杂，我们会发现这2个角色后续有更多的不同之处，但现在的这种写法，我们是没办法把这2个角色适用的功能区分开来的，也就是说，每个角色都可以直接调用任意功能，没有任何限制。
3. 我们在上面定义了got\_shot()后要减血，也就是说减血这个动作是应该通过被击中这个事件来引起的,我们调用get\_shot(), got\_shot () 这个函数再调用每个角色里的life\_value变量来减血。但其实我不通过got\_shot(), 直接调用角色roles[role\_id]['life\_value'] 减血也可以呀，但是如果这样调用的话，那可以就是简单粗暴啦，因为减血之前其它还应该判断此角色是否穿了防弹衣等，如果穿了的话，伤害值肯定要减少，got\_shot()函数里就做了这样的检测，你这里直接绕过的话，程序就乱了。因此这里应该设计成除了通过got\_shot(),其它的方式是没有办法给角色减血的，不过在上面的程序设计里，是没有办法实现的。
4. 现在需要给所有角色添加一个可以穿防弹衣的功能，那很显然你得在每个角色里放一个属性来存储此角色是否穿了防弹衣，那就要更改每个角色的代码，给添加一个新属性，这样太low了，不符合代码可复用的原则

上面这4点问题如果不解决，以后肯定会引出更大的坑，有同学说了，解决也不复杂呀，直接在每个功能调用时做一下角色判断啥就好了，没错，你要非得这么霸王硬上弓的搞也肯定是可以实现的，那你自已就开发相应的代码来对上面提到的问题进行处理好啦。但这些问题其实能过OOP就可以很简单的解决。

之前的代码改成用OOP中的“类”来实现的话如下：

```

1  class Role(object):
2      def __init__(self,name,role,weapon,life_value=100,money=15000):
3          self.name = name
4          self.role = role
5          self.weapon = weapon
6          self.life_value = life_value
7          self.money = money

```

```

8
9     def shot(self):
10         print("shooting...")
11
12     def got_shot(self):
13         print("ah...,I got shot...")
14
15     def buy_gun(self,gun_name):
16         print("just bought %s" %gun_name)
17
18     r1 = Role('Alex','police','AK47') #生成一个角色
19     r2 = Role('Jack','terrorist','B22') #生成一个角色

```

先不考虑语法细节，相比靠函数拼凑出来的写法，上面用面向对象中的类来写最直接的改进有以下2点：

1. 代码量少了近一半
2. 角色和它所具有的功能可以一目了然看出来

在真正开始分解上面代码含义之前，我们先来了解一些类的基本定义

### 类的语法

```

1     class Dog(object):
2
3         print("hello,I am a dog!")
4
5
6     d = Dog() #实例化这个类，
7     #此时的d就是类Dog的实例化对象
8
9     #实例化，其实就是以Dog类为模版，在内存里开辟一块空间，存上数据，赋值成一个变量名

```

上面的代码其实有问题，想给狗起名字传不进去。

```

1     class Dog(object):
2
3         def __init__(self,name,dog_type):
4             self.name = name
5             self.type = dog_type
6
7         def sayhi(self):
8
9             print("hello,I am a dog, my name is ",self.name)
10
11
12     d = Dog('LiChuang','京巴')
13     d.sayhi()

```

为什么有\_\_init\_\_？为什么有self？此时的你一脸蒙逼，相信不画个图，你的智商是理解不了的！

画图之前，你先注释掉这两句

```

1     # d = Dog('LiChuang','京巴')
2     # d.sayhi()
3
4     print(Dog)

```

没实例直接打印Dog输出如下

```

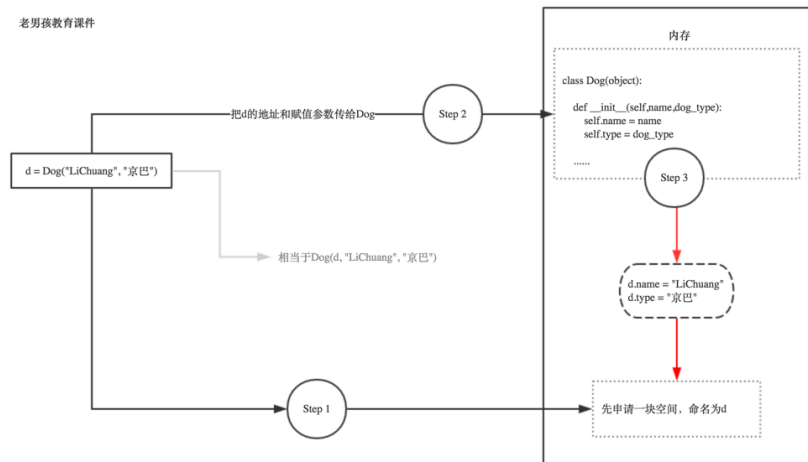
1     <class '__main__.Dog'>

```

这代表什么？代表 即使不实例化，这个Dog类本身也是已经存在内存里的对不对，yes，cool，那实例化时，会产生什么化学反应呢？



老男孩教育课件



根据上图我们得知, **其实self,就是实例本身! 你实例化时python会自动把这个实例本身通过self参数传进去。**

你说好吧, 假装懂了, 但下面这段代码你又不明白了, 为何sayhi(self),要写个self呢?

```
1 class Dog(object):
2     ...
3     def sayhi(self):
4
5         print("hello,I am a dog, my name is ",self.name)
```

这个原因, 我课上在讲。。。

```
1 <br><br><br><br>
```

好了, 明白了类的基本定义, 接下来我们一起分解一下上面的代码分别 是什么意思

```
1 class Role(object): #定义一个类, class是定义类的语法, Role是类名, (object)是新
2     def __init__(self,name,role,weapon,life_value=100,money=15000): #初始化i
3         self.name = name #__init__中的第一个参数self,和这里的self都 是什么意思
4         self.role = role
5         self.weapon = weapon
6         self.life_value = life_value
7         self.money = money
```

上面的这个\_\_init\_\_()叫做初始化方法(或构造方法), 在类被调用时, 这个方法(虽然它是函数形式, 但在类中就不叫函数了,叫方法)会自动执行, 进行一些初始化的动作, 所以我们这里写的\_\_init\_\_(self,name,role,weapon,life\_value=100,money=15000)就是要在创建一个角色时给它设置这些属性, 那么这第一个参数self是干毛用的呢?

初始化一个角色, 就需要调用这个类一次:

```
1 r1 = Role('Alex','police','AK47') #生成一个角色 , 会自动把参数传给Role下面的__
2 r2 = Role('Jack','terrorist','B22') #生成一个角色
```

我们看到, 上面的创建角色时, 我们并没有给\_\_init\_\_传值, 程序也没报错, 是因为, 类在调用它自己的\_\_init\_\_(...)时自己帮你给self参数赋值了,

```
1 r1 = Role('Alex','police','AK47') #此时self 相当于 r1 , Role(r1,'Alex','pol
2 r2 = Role('Jack','terrorist','B22')#此时self 相当于 r2, Role(r2,'Jack','terr
```

为什么这样子? 你拉着我说你有些犹豫, 怎么会这样子?

你执行r1 = Role('Alex','police','AK47') 时, python的解释器其实干了两件事:

1. 在内存中开辟一块空间指向r1这个变量名

2. 调用Role这个类并执行其中的\_\_init\_\_(...)方法, 相当于

Role.\_\_init\_\_(r1,'Alex','police',' AK47' ),这么做是为什么呢? 是为了把'Alex','police',' AK47' 这三个值跟刚开辟的r1关联起来, 是为了把'Alex','police',' AK47' 这三个值跟刚开辟的r1关联起来, 是为了把'Alex','police',' AK47' 这三个值跟刚开辟的r1关联起来, 重要的事情说3次, 因为关联起来后, 你就可以直接r1.name, r1.weapon 这样来调用啦。所以, 为实现这种关联, 在调用\_\_init\_\_方法时, 就必须把r1这个变量也传进去, 否则\_\_init\_\_不知道要把那3个参数跟谁关联呀。

3. 明白了么哥? 所以这个\_\_init\_\_(...)方法里的, self.name = name , self.role = role 等等的意思就是要把这几个值 存到r1的内存空间里。

如果还不明白的话, 哥, 去测试一下智商吧, 应该不会超过70, 哈哈。

为了暴露自己的智商, 此时你假装懂了, 但又问, \_\_init\_\_(...)我懂了, 但后面的那几个函数, 噢 不对, 后面那几个方法 为什么也还需要self参数么? 不是在初始化角色的时候, 就已经把角色的属性跟r1绑定好了么?

good question, 先来看一下上面类中的一个buy\_gun的方法:

```
1 def buy_gun(self,gun_name):
2     print("%s has just bought %s" %(self.name,gun_name) )
```

上面这个方法通过类调用的话要写成如下:

```
1 r1 = Role('Alex','police','AK47')
2 r1.buy_gun("B21") #python 会自动帮你转成 Role.buy_gun(r1,"B21")
```

执行结果

#Alex has just bought B21

依然没给self传值, 但Python还是会自动的帮你把r1 赋值给self这个参数, 为什么呢? 因为, 你在buy\_gun(..)方法中可能要访问r1的一些其它属性呀, 比如这里就访问了r1的名字, 怎么访问呢? 你得告诉这个方法呀, 于是就把r1传给了这个self参数, 然后在buy\_gun里调用 self.name 就相当于调用r1.name 啦, 如果还想知道r1的生命值有多少, 直接写成self.life\_value就可以了。说白了就是在调用类中的一个方法时, 你得告诉人家你是谁。

好啦, 总结一下2点:

1. 上面的这个r1 = Role('Alex','police','AK47')动作, 叫做类的“实例化”, 就是把一个虚拟的抽象的类, 通过这个动作, 变成了一个具体的对象了, 这个对象就叫做实例
2. 刚才定义的这个类体现了面向对象的第一个基本特性, 封装, **其实就是使用构造方法将内容封装到某个具体对象中, 然后通过对象直接或者self间接获取被封装的内容**

## 面向对象的特性:

### 封装

封装最好理解了。封装是面向对象的特征之一, 是对象和类概念的主要特性。

封装, 也就是把客观事物封装成抽象的类, 并且类可以把自己的数据和方法只让可信的类或者对象操作, 对不可信的进行信息隐藏。

### 继承

面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力: 它可以使用现有类的所有功能, 并在无需重新编写原来的类的情况下对这些功能进行扩展。

通过继承创建的新类称为“子类”或“派生类”。

被继承的类称为“基类”、“父类”或“超类”。

继承的过程, 就是从一般到特殊的过程。

要实现继承, 可以通过“继承” (Inheritance) 和“组合” (Composition) 来实现。

在某些 OOP 语言中, 一个子类可以继承多个基类。但是一般情况下, 一个子类只能有一个基类, 要实现多重继承, 可以通过多级继承来实现。

继承概念的实现方式主要有2类: 实现继承、接口继承。

Ø 实现继承是指使用基类的属性和方法而无需额外编码的能力;

Ø 接口继承是指仅使用属性和方法的名称、但是子类必须提供实现的能力(子类重构父类方法);

在考虑使用继承时, 有一点需要注意, 那就是两个类之间的关系应该是“属于”关系。例如, Employee 是一个人, Manager 也是一个人, 因此这两个类都可以继承 Person 类。但是 Leg 类却不能继承 Person 类, 因为腿并不是一个人。

抽象类仅定义将由子类创建的一般属性和方法。

OO开发范式大致为: 划分对象→抽象类→将类组织成为层次化结构(继承和合成) →用类与实例进行设计和实现几个阶段。

继承示例

[+ View Code](#)



多态性 (polymorphisn) 是允许你将父对象设置成为和一个或更多的他的子对象相等的技术, 赋值之后, 父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说, 就是一句话: 允许将子类类型的指针赋值给父类类型的指针。

那么, 多态的作用是什么呢? 我们知道, 封装可以隐藏实现细节, 使得代码模块化; 继承可以扩展已存在的代码模块 (类); 它们的目的都是为了——代码重用。而多态则是为了实现另一个目的——接口重用! 多态的作用, 就是为了类在继承和派生的时候, 保证使用“家谱”中任一类的实例的某一属性时的正确调用。

Pyhon 很多语法都是支持多态的, 比如 len(),sorted(), 你给len传字符串就返回字符串的长度, 传列表就返回列表长度。

Python多态示例

```

1  #_*_coding:utf-8_*_
2
3
4  class Animal(object):
5      def __init__(self, name): # Constructor of the class
6          self.name = name
7
8      def talk(self):           # Abstract method, defined by convention o
9          raise NotImplementedError("Subclass must implement abstract method"
10
11
12  class Cat(Animal):
13      def talk(self):
14          print('%s: 喵喵喵!' %self.name)
15
16
17  class Dog(Animal):
18      def talk(self):
19          print('%s: 汪! 汪! 汪! ' %self.name)
20
21

```

```
22
23 def func(obj): #一个接口, 多种形态
24     obj.talk()
25
26 c1 = Cat('小晴')
27 d1 = Dog('李磊')
28
29 func(c1)
30 func(d1)
```

## 领域模型

好了, 你现在会了面向对象的各种语法了, 那请看下本章最后的作业需求, 我相信你可能是蒙蔽的, 很多同学都是学会了面向对象的语法, 却依然写不出面向对象的程序, 原因是什么呢? 原因就是因为你还没掌握一门面向对象设计利器, 你说我读书少别骗我, 什么利器?

答案就是:**领域建模**。从领域模型开始,我们就开始了面向对象的分析和设计过程,可以说,领域模型是完成从需求分析到面向 对象设计的一座桥梁。

领域模型,顾名思义,就是需求所涉及的一个建模,更通俗的讲法是业务模型。参考百度百科(<http://baike.baidu.cn/view/757895.htm> ),领域模型定义如下:

从这个定义我们可以看出,领域模型有两个主要的作用:

1. 发掘重要的业务领域概念
2. 建立业务领域概念之间的关系

## 领域建模三字经

领域模型如此重要,很多同学可能会认为领域建模很复杂,需要很高的技巧。然而事实上领域建模非常简单,简单得有点难以让人相信,领域建模的方法概括一下就是“**找名词**”! 许多同学看到这个方法后估计都会笑出来:太假了吧,这么简单,找个初中生都会啊,那我们公司那些分析师和设计师还有什么用哦?

分析师和设计师当然有用,后面我们会看到,即使是简单的找名词这样的操作,也涉及到分析和提炼,而不是简单的摘取出来就可,这种情况下分析师和设计师的经验和技能就能够派上用场了。但领域模型分析 也确实相对简单,即使没有丰富的经验和高超的技巧,至少也能完成一个能用的领域模型。

虽然我们说“找名词”很简单,但一个关键的问题还没有说明:**从哪里找**? 如果你还记得领域模型是“需求到面向对象的桥梁”,那么你一定一下子就能想到:从需求模型中找,具体来说就是从用例中找。

归纳一下域建模的方法就是“**从用例中找名词**”。当然,找到名词后,为了能够更加符合面向对象的要求和特点,我们还需要对这些名词进一步完善,这就是接下来的步骤:**加属性,连关系**!

最后我们总结出领域建模的三字经方法:**找名词、加属性、连关系**。

## 找名词

who : 学员、讲师、管理员

用例:

- 1. 管理员 创建了 北京 和 上海 两个校区
- 2. 管理员 创建了 Linux \ Python \ Go 3个课程
- 3. 管理员 创建了 北京校区的Python 16期, Go开发第一期, 和上海校区的Linux 36期 班级
- 4. 管理员 创建了 北京校区的 学员 小晴, 并将其 分配 在了 班级 python 16期
- 5. 管理员 创建了 讲师 Alex, 并将其分配 给了 班级 python 16期 和全栈脱产5期
- 6. 讲师 Alex 创建 了一条 python 16期的 上课纪录 Day6
- 7. 讲师 Alex 为Day6这节课 所有的学员 批了作业, 小晴得了A, 李磊得了C-, 严帅得了B
- 8. 学员小晴 在 python 16 的 day6里 提交了作业
- 9. 学员李磊 查看了自己所报的所有课程
- 10 学员 李磊 在 查看了 自己在 py16期 的 成绩列表, 然后自杀了
- 11. 学员小晴 跟 讲师 Alex 表白了

名词列表:

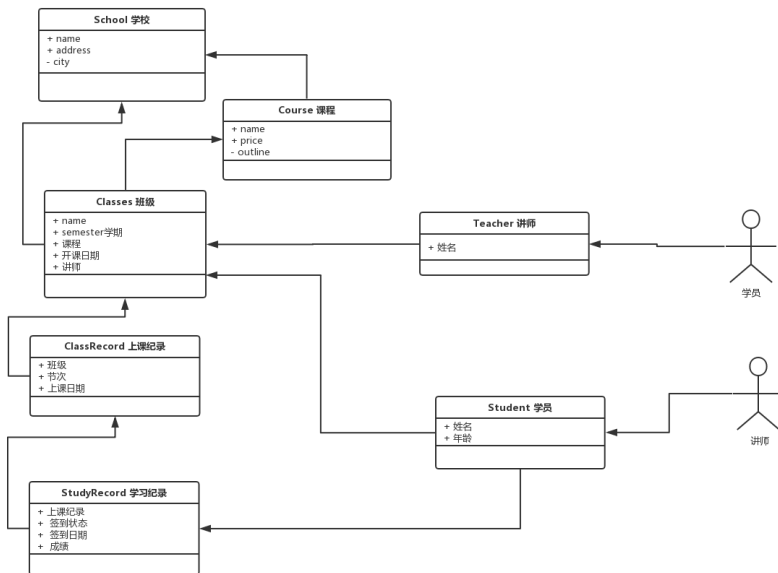
管理员、校区、课程、班级、上课纪录、作业、成绩、讲师、学员

加属性

| 名词   | 属性                         | 备注                    |
|------|----------------------------|-----------------------|
| 管理员  | N/A                        | 具备所有后台管理功能, 不需特别的用户属性 |
| 校区   | 开设的课程、班级、讲师列表、学员列表、学校名称、地址 |                       |
| 课程   | 课程名称、周期、价格、大纲              |                       |
| 班级   | 学期、开课日期、讲师、学员列表            |                       |
| 上课纪录 | 所属班级、课程节次、上课日期             |                       |
| 作业   | 作业提交时间、作业评语、作业成绩           |                       |
| 成绩   | 每节课的作业成绩                   |                       |
| 讲师   | 姓名、关联的班级                   |                       |
| 学员   | 姓名、年龄、所报的班级                |                       |

连关系

有了类,也有了属性,接下来自然就是找出它们的关系了。



### 本节作业: 选课系统

角色: 学校、学员、课程、讲师

要求:

1. 创建北京、上海 2 所学校
2. 创建linux , python , go 3个课程 , linux\py 在北京开 , go 在上海开
3. 课程包含 , 周期 , 价格 , 通过学校创建课程
4. 通过学校创建班级 , 班级关联课程、讲师
5. 创建学员时 , 选择学校 , 关联班级
5. 创建讲师角色时要关联学校 ,
6. 提供两个角色接口
- 6.1 学员视图 , 可以注册 , 交学费 , 选择班级 ,
- 6.2 讲师视图 , 讲师可管理自己的班级 , 上课时选择班级 , 查看班级学员列表 , 修改所管理的学员的成绩
- 6.3 管理视图 , 创建讲师 , 创建班级 , 创建课程
7. 上面的操作产生的数据都通过pickle序列化保存到文件里



分类: [Python自动化开发之路](#)

好文要顶

已关注

收藏该文

[金角大王](#)  
关注 - 5  
粉丝 - 10882

[我在关注他](#) [取消关注](#)

80

posted @ 2016-02-13 18:25 [金角大王](#) 阅读(53261) 评论(13) 编辑 收藏

评论列表

|      |                  |  |                                       |  |
|------|------------------|--|---------------------------------------|--|
| #1楼  | 2016-08-27 11:16 | 明天OoO你好<br>前排西瓜  | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(2)</a> <a href="#">反对(0)</a>  |
| #2楼  | 2016-12-29 22:23 | 真是好人<br>Alex, 讲的真棒。  | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(0)</a> <a href="#">反对(0)</a>  |
| #3楼  | 2017-01-10 09:50 | 水墨青花<br>Alex 讲的非常棒   | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(0)</a> <a href="#">反对(0)</a>  |
| #4楼  | 2017-04-25 16:32 | Eaten.Cheung<br>作业的代码哪里能找到啊?   | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(14)</a> <a href="#">反对(0)</a> |
| #5楼  | 2017-07-03 21:14 | Iven_z<br>请问作业有参考代码吗?  | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(5)</a> <a href="#">反对(0)</a>  |
| #6楼  | 2017-07-05 15:18 | PandaSky<br>个人感觉python是动态类型, 严格来说不支持多态。老师上课讲的animal.test_talk静态方法中传入一个非animal子类的类, 并且该类也有talk方法, 代码依然可以执行! 如果像java C#中, animal.test_talk的参数是animal类型的话, 那肯定是多态的实现。所以严格来说python具有多态个人感觉有点牵强。个人观点! | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(5)</a> <a href="#">反对(11)</a> |
| #7楼  | 2018-01-13 20:01 | 彭世瑜<br>又是学到了新东西, 来, 老师我交个作业, 欢迎大家帮我解决最后那个问题, 头大了, 搞了一天, 不想弄了。同时也分享给大家<br><a href="https://github.com/mouday/ChooseCourseSys">https://github.com/mouday/ChooseCourseSys</a>                       | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(1)</a> <a href="#">反对(0)</a>  |
| #8楼  | 2018-04-25 21:07 | 一只火眼金睛的男猴<br>再来看alex老师的总结  | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(0)</a> <a href="#">反对(0)</a>  |
| #9楼  | 2018-05-13 19:09 | 一只火眼金睛的男猴<br>今天讲了class, 留的是这个作业。不是您低沉且性感的嗓音, 怀念你那鲜红的唇印。  | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(0)</a> <a href="#">反对(0)</a>  |
| #10楼 | 2018-08-12 15:48 | 木华园<br>Alex老师, 引子中的最后一行代码: d1['bark'](p1) #把人的对象传给了狗的方法<br>没有限制住把人的对象传给了狗了啊! d1['bark'](p1)不是一样可以执行吗?  | <a href="#">回复</a> <a href="#">引用</a> | <a href="#">支持(0)</a> <a href="#">反对(0)</a>  |

- #11楼 2018-09-17 21:29 yooцин

代码哪找啊

回复 引用

支持(0) 反对(0)
- #12楼 2018-09-25 21:59 日出东海，我心向西

交个作业：选课系统  
<https://www.cnblogs.com/heaven-xi/p/9703823.html>

回复 引用

支持(0) 反对(0)
- #13楼 2019-01-25 01:57 卜戈的博客

交作业：  
<https://github.com/254978626/ChooseCourseSystem/tree/master/choosecoursesystem>。fuck this system





回复 引用

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

发表评论

编辑 预览

B    

支持 Markdown

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]

- 【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区
- 【推荐】有道智云周年庆，API服务大放送，注册即送100元体验金！
- 【推荐】开放下载！《CDN排坑指南》



相关博文：

- Python之路,Day6 - 面向对象学习
- Python之路,Day6 - 面向对象学习
- python之路\_day6
- Python之路\_Day6
- python 之路 -----> day6
- » 更多推荐...

最新 IT 新闻：

- 腾讯黑鲨3S评测：升级120Hz刷新率 提升整机游戏体验
- 理想成功IPO，我们和它的投资人聊了聊李想与理想
- 1000万份微信支付“摇免单”：每单最高200元

- 市值跌去99%、冯鑫被捕一年：暴风集团濒临“暴风退”
  - 董明珠称空调一晚一度电都是骗人的：企业不敢再打这样的广告
- » 更多新闻...