第十一章类与对象

> 本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务器进行重写。学完之后，你所收获的不仅仅是精通一门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

一 面向对象的由来

面向过程的程序设计：核心是过程二字，过程指的是解决问题的步骤，即先干什么再干什么.....面向过程的设计就好比精心设计好一条流水线，是一种机械式的思维方式。

优点是：复杂度的问题流程化，进而简单化（一个复杂的问题，分成一个个小的步骤去实现，实现小的步骤将会非常简单）
缺点是：一套流水线或者流程就是用来解决一个问题，生产汽水的流水线无法生产汽车，即便是能，也得是大改，改一个组件，牵一发而动全身。
应用场景：一旦完成基本很少改变的场景，著名的例子有Linux内核，git，以及Apache HTTP Server等。

> 面向对象的程序设计：核心是对象二字。

要理解对象为何物，必须把自己当成上帝，上帝眼里世间存在的万物皆为对象，不存在的也可以创造出来。面向对象的程序设计好比如来设计西游记，如来要解决的问题是把经书传给东土大唐，如来想了想解

决这个问题需要四个人：唐僧，沙和尚，猪八戒，孙悟空，每个人都有各自的特征和技能（这就是对象的概念，特征和技能分别对应对象的数据属性和方法属性），然而这并不好玩，于是如来又安排了一群妖魔鬼怪，为了防止师徒四人在取经路上被搞死，又安排了一群神仙保驾护航，这些都是对象。然后取经开始，师徒四人与妖魔鬼怪神仙交互着直到最后取得真经。如来根本不会管师徒四人按照什么流程去取。

> 对象是特征与技能的结合体

基于面向对象设计程序就好比在创造一个世界，你就是这个世界的上帝，存在的皆为对象，不存在的也可以创造出来，与面向过程机械式的思维方式形成鲜明对比，面向对象更加注重对现实世界的模拟，是一种“上帝式”的思维方式。

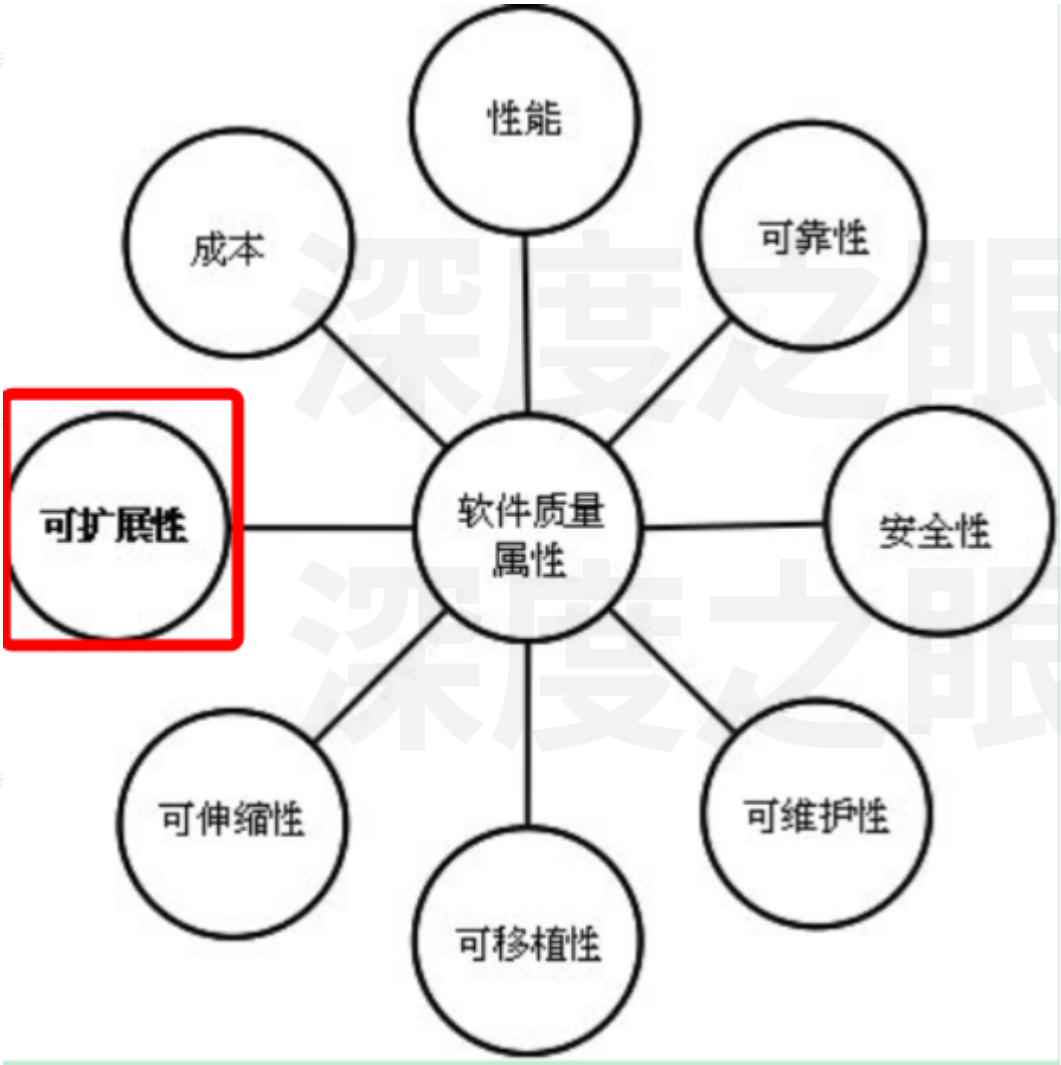
优点是：解决了程序的扩展性。对某一个对象单独修改，会立刻反映到整个体系中，如对游戏中一个人物参数的特征和技能修改都很容易。

缺点：

1. 编程的复杂度远高于面向过程，不了解面向对象而立即上手基于它设计程序，极容易出现过度设计的问题。一些扩展性要求低的场景使用面向对象会徒增编程难度，比如管理linux系统的shell脚本就不适合用面向对象去设计，面向过程反而更加适合。
2. 无法向面向过程的程序设计流水线式的可以很精准的预测问题的处理流程与结果，面向对象的程序一旦开始就由对象之间的交互解决问题，即便是上帝也无法准确地预测最终结果。于是我们经常看到对战类游戏，新增一个游戏人物，在对战的过程中可能会出现bug的技能，一刀砍死3个人，这种情况是无法准确预知的，只有对象之间交互才能准确地知道最终的结果。

应用场景：需求经常变化的软件，一般需求的变化都集中在用户层，互联网应用，企业内部软件，游戏等都是面向对象的程序设计大显身手的好地方

面向对象的程序设计并不是全部。对于一个软件质量来说，面向对象的程序设计只是用来解决扩展性。



二 类的用法

1. 类的构造

类即类别、种类，是面向对象设计最重要的概念，对象是特征与技能的结合体，而类则是一系列对象相似的特征与技能的结合体
那么问题来了，先有的一个个具体存在的对象（比如一个具体存在的人），还是先有的人类这个概念，这个问题需要分两种情况去看

> 在现实世界中：先有对象，再有类

世界上肯定是先出现各种各样的实际存在的物体，然后随着人类文明的发展，人类站在不同的角度总结出了不同的种类，如人类、动物类、植物类等概念

也就说，对象是具体的存在，而类仅仅只是一个概念，并不真实存在

```
在程序中：务必保证先定义类，后产生对象
```

这与函数的使用是类似的，先定义函数，后调用函数，类也是一样的，在程序中需要先定义类，后调用类
不一样的是，调用函数会执行函数体代码返回的是函数体执行的结果，而调用类会产生对象，返回的是对象
按照上述步骤，我们来定义一个类（我们站在deepshare学校的角度去看，各位读者都是学生）

```
#在现实世界中：先有对象，再有类
对象1：李坦克
    特征：
        学校=deepshare
        姓名=李坦克
        性别=男
        年龄=18
    技能：
        学习
        吃饭
        睡觉
对象2：王大炮
    特征：
        学校=deepshare
        姓名=王大炮
        性别=女
        年龄=38
    技能：
        学习
        吃饭
        睡觉
对象3：牛榴弹
    特征：
        学校=deepshare
        姓名=牛榴弹
        性别=男
        年龄=78
    技能：
        学习
        吃饭
        睡觉
现实中的deepshare学生类
    相似的特征：
```

```
        学校=deepshare
    相似的技能：
        学习
        吃饭
        睡觉
```

在程序中，先定义类，在产生对象

```
#在程序中，务必保证：先定义（类），后使用（产生对象）
PS:
  1. 在程序中特征用变量标识，技能用函数标识
  2. 因而类中最常见的无非是：变量和函数的定义
  3. 定义类，使用关键字class+空格+类名，类名用大驼峰命名，变量名推荐使用"_"这种方式就是为了区分
#程序中的类
class DeepshareStudent:
    # 用变量表示特征
    school='deepshare'
    # 用函数表示技能
    def learn(self):
        """
        self是你使用Pycharm自动添加的，你写成xxx也是一样的，
        他就是一个位置参数，必须被传值，具体意义请看下文
        """
        print('is learning')

    def eat(self):
        print('is eating')

    def sleep(self):
        print('is sleeping')

    print('=======>')
    # 类是特征与技能的结合体，所以类中最常见的就是变量和函数，但是类中也可以有任意的Python代码
# 执行以上代码，你会发现会打印那一行箭头，这就说明前面的代码也运行了，
# 这就说明在定义类的阶段会立刻执行类体内的代码，
# 既然执行了，有变量有函数，那就会将产生的名字存放于类的名称空间
```

2. 类的查看

那么我们怎么查看类中的变量和函数呢

```
# 使用类型.__dict__方法可以查看
name_and_func = DeepshareStudent.__dict__
print(name_and_func)
# 输出
"""
{   '__module__': '__main__',
    'school': 'deepshare',
    'learn': <function DeepshareStudent.learn at 0x103698268>,
    'eat': <function DeepshareStudent.eat at 0x105c46488>,
    'sleep': <function DeepshareStudent.sleep at 0x1036797b8>,
    '__dict__': <attribute '__dict__' of 'DeepshareStudent' objects>,
    '__weakref__': <attribute '__weakref__' of 'DeepshareStudent' objects>,
    '__doc__': None}
"""

# 注意：
#   1.很明显这是一个字典的格式，因为有名字有内存地址，他们是一一对应的关系，最合适就是使用字典来存
#   2.除去Python内置的键值对，我们只看和我们定义的类相关的部分
print(name_and_func['school'])
print(name_and_func['learn'])
# 执行代码输出
"""
deepshare
<function DeepshareStudent.learn at 0x103bb0268>
我们可以看到name_and_func['learn']是一个函数的内存地址，那么我加()应该就能调用他
"""

name_and_func['learn']()
```

结果发现居然报错了

```
/Users/albert/anaconda3/bin/python /Users/albert/Desktop/test01/01.py
Traceback (most recent call last):
  File "/Users/albert/Desktop/test01/01.py", line 75, in <module>
    name_and_func['learn']()
TypeError: learn() missing 1 required positional argument: 'self'
```

不要慌，错误类型说learn()这个函数少了一个位置参数'self',回到上面看我们的类中的learn函数，位置参数self必须被传值，但是我们没有传值

```
name_and_func['learn']('albert')
```

再来执行代码，发现learn这个函数正常执行了，但是和我们传的参数没有任何关系，我们能否修改一下呢

```
class DeepshareStudent:
    school = 'deepshare'
    def learn(self):
        print('%s is learning' % self)  # 我们把这个函数稍微改一下
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
```

再来执行

```
name_and_func['learn']('albert')
```

输出结果

```
albert is learning
```

好了函数调用这里只是一个小插曲，这里为了告诉大家，一旦看到函数的内存地址，就要想到函数名加()就能调用这个函数
现在再来说一下这个self指的是什么?

```
self指的是类的对象本身
```

在接下来的代码中慢慢理解，我们刚才说到了类有***dict***方法，拿到的结果就是字典
所以

```
name_and_func = DeepshareStudent.__dict__
print(name_and_func['school'])
print(name_and_func['learn'])
```

```
print(name_and_func['eat'])
print(name_and_func['sleep'])
```

每次当我们想拿到类的技能或者属性就可以使用这种方式，你是否会觉得很麻烦
机智的龟叔对高大上的Python语言自然有更好的处理方式

```
# 对象直接.这个变量名或者函数名
print(DeepshareStudent.school)
print(DeepshareStudent.learn)
print(DeepshareStudent.eat)
print(DeepshareStudent.sleep)
# 输出结果和上方代码一致，其实我们shying这种方式，他的本质就是在执行上方代码
# __开头__结尾都是在某种情况下，自动触发，我们自己一般不需要写
```

那么我们通过类名 点 变量名或者函数名，这称之为类的什么?

```
print(DeepshareStudent.abc)
```

这行代码我点了一个没有的变量名或者函数名，看报错信息

```
/Users/albert/anaconda3/bin/python /Users/albert/Desktop/test01/01.py
Traceback (most recent call last):
  File "/Users/albert/Desktop/test01/01.py", line 81, in <module>
    print(DeepshareStudent.abc)
AttributeError: type object 'DeepshareStudent' has no attribute 'abc'
```

他说DeepshareStudent这个类没有abc属性，所以这些都是属性，与其他一些语言不同的是，Python对属性不做区分，统一都叫属性，使用点加属性名就可以拿到属性值，那么这门语言你觉得是好还是不好呢

3. 类的修改

```
# 修改类的属性值
DeepshareStudent.school = '深度之享'
print(DeepshareStudent.school)
# 给类添加属性
DeepshareStudent.country = 'China'
print(DeepshareStudent.country)
# 删除类的属性
```

```
del DeepshareStudent.country
print(DeepshareStudent.country)
```

二 对象的用法

面向对象的变成思想核心是对象，我们最后用的是对象，上一小节我们只是把类造出来了，类其实相当于一个模版或者相当于一个工厂，接下来我们就会使用类来批量生产对象。

```
在程序中：先定义类=====>调用类=====>产生对象
```

定义类，把原来定义类的代码放在这里：

```
class DeepshareStudent:
    school = 'deepshare'
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
```

调用类的方式和调用函数一样，类名后面加括号

```
DeepshareStudent()
# 执行代码没有任何结果
```

调用类的同时就会产生出对象，我们只需要把它赋值给对象就可以了

```
stu1 = DeepshareStudent()
stu2 = DeepshareStudent()
stu3 = DeepshareStudent()
print(stu1)
print(stu2)
print(stu3)
```

输出

```
<__main__.DeepshareStudent object at 0x10ca88a90>
<__main__.DeepshareStudent object at 0x10ca88d68>
<__main__.DeepshareStudent object at 0x10ca88d30>
```

每调用一次，类的模子就会加工出来一个对象，但是我们这么做加工个出来的对象有任何区别吗？你如果没有告诉工厂，他每次造出来的对象都是一样的，但是我们需要的对象肯定不是完全一样的

```
#在现实世界中：先有对象，再有类
对象1：李坦克
    特征：
        学校=deepshare
        姓名=李坦克
        性别=男
        年龄=18
    技能：
        学习
        吃饭
        睡觉
对象2：王大炮
    特征：
        学校=deepshare
        姓名=王大炮
        性别=女
        年龄=38
    技能：
        学习
        吃饭
        睡觉
对象3：牛榴弹
    特征：
        学校=deepshare
        姓名=牛榴弹
        性别=男
        年龄=78
    技能：
        学习
        吃饭
        睡觉
现实中的deepshare学生类
    相似的特征：
```

```
        学校=deepshare
    相似的技能：
        学习
        吃饭
        睡觉
```

对象之间除了有相似的技能，还有不一样的特征，我们需要为不一样的对象定制它独有的特征
就像造人，你们有没有经验，除了有头，身体，胳膊和腿，可定有的人个子高，有的人个子矮，有人胖有人瘦等等，有的是共有的，有的是独有的，这在程序中怎么实现呢?

```
class DeepshareStudent:
    school = 'deepshare'
    print('=====>')
    def __init__(self):  # 在程序中定义一个__init__函数
        print('===init run====>')  # 我们先什么都不屑，打印一下
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
```

执行代码，第一行打印执行了，这是因为类的定义阶段就会执行类体内的代码，这一点，我们上面讲过了，但是init函数内的打印没有执行，那么什么时候才会触发init这个函数的执行呢?

```
class DeepshareStudent:
    school = 'deepshare'
    print('=====>')
    def __init__(self):
        print('====init run===>')
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
DeepshareStudent()
```

输出

```
=====>
====init run===>
```

我们调用函数使用函数名加括号，可上面的代码，我们有主动调用过*init*这个函数吗，显然没有，但是这个函数确实是执行了，那就说明了我们在调用类的时候这个函数会自动触发执行
那么我们用这个功能来做什么事呢?
看他的名字init，有道字典查一下是初始化的意思，你知道了么
我们如何使用init来给对象赋予不同的属性呢?

```
class DeepshareStudent:
    school = 'deepshare'
    def __init__(self,x,y,z):  # 在self后面加三个位置参数,x,y,z
        self.NAME = x  # 先这样写，等会在说是干嘛的
        self.AGE = y
        self.GENDER = z
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
DeepshareStudent()
```

执行代码，发现报错了

```
/Users/albert/anaconda3/bin/python /Users/albert/Desktop/test01/01.py
Traceback (most recent call last):
  File "/Users/albert/Desktop/test01/01.py", line 75, in <module>
    DeepshareStudent()
TypeError: __init__() missing 3 required positional arguments: 'x', 'y', and 'z'
```

他说 *init*这个函数少了三个位置参数x,y,z
分析一下，我们能够发现，调用类时会自动触发*init*函数的执行，函数有位置形参，我们没有传实参的值，我们怎么传值呢？我们也没有别的地方传值，你猜也能猜到，只能这么传

```
DeepshareStudent('李坦克',18,'male')
```

执行代码，发现没有报错，程序无任何结果，接下来我们先总结一下，调用类做了哪些事

1. 首先产生一个空对象stu1
2. 自动触发类内部*init*函数的执行
3. 然后将空对象stu1连同调用类时括号内的参数组成（stu1,"李坦克",18,'male'）,将这四个参数一起传给init函数

```
DeepshareStudent('李坦克',18,'male')
# DeepshareStudent.__init__(stu1,"李坦克",18,'male')
```

传递给init这个函数

```
def __init__(self,x,y,z):  # self=stu1  name= "李坦克"   age=18
        self.NAME = x  # stu1.NAME = '李坦克'
        self.AGE = y  # stu1.AGE = 18
        self.GENDER = z  # stu1.GENDER = 'male'
```

stu1给了self，"李坦克"给了x, 18给了y，‘male’给了z
再往下走，obj.NAME这种语法熟悉不熟悉，我们在上面类的那节讲到这是在调用类的属性，现在用到对象身上同样适用，不过这是在给对象添加属性并赋值

```
class DeepshareStudent:
    school = 'deepshare'
    def __init__(self,x,y,z):
        self.NAME = x
        self.AGE = y
        self.GENDER = z
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
stu1 = DeepshareStudent('李坦克',18,'male')
print(stu1.NAME)
print(stu1.AGE)
print(stu1.GENDER)
print(stu1.school)
print(stu1.learn)
stu2 = DeepshareStudent('王大炮',38,'female')
print(stu2.NAME)
```

```
print(stu2.AGE)
print(stu2.GENDER)
print(stu2.school)
print(stu2.learn)
```

这样就实现了我们的需求，对象既有公共属性，又有私有属性，不过，我们来看一下代码有没有修改的余地

```
class DeepshareStudent:
    school = 'deepshare'
    # 最开始写成x,y,z是为了让你理解，为了让形参更有意义，我们写成这种方式
    def __init__(self,name,age,gender):
        self.name = name  # 第一个name是对象的属性名，第二个name是变量名，同名是为了使用方便
        self.age = age
        self.gender = gender
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
```

需要注意的是self是对象本身，他是在调用类产生对象时自动传值的，写代码是Pycharm会自动添加这个self，你写成aaa也是可以的，不过意义不明确，但是你不写是不行的

```
class DeepshareStudent:
    school = 'deepshare'
    def __init__(name,age,gender):  # 在这一行报错
        self.name = name  # 代码不会执行到这一样就报错了
        self.age = age
        self.gender = gender
    def learn(self):
        print('%s is learning' % self)
    def eat(self):
        print('is eating')
    def sleep(self):
        print('is sleeping')
stu1 = DeepshareStudent('李坦克',18,'male')
```

程序报错了

```
/Users/albert/anaconda3/bin/python /Users/albert/Desktop/test01/01.py
Traceback (most recent call last):
  File "/Users/albert/Desktop/test01/01.py", line 76, in <module>
    stu1 = DeepshareStudent('李坦克',18,'male')
TypeError: __init__() takes 3 positional arguments but 4 were given
```

他说程序需要三个参数，但是你传了4个，就说明有一个是自动传值的

总结：

init这个函数就是给对象初始化，在类创造对象的时候就赋予对象属性以不同的特征