Python 程序设计基础

江西财经大学-朱文强

第六章

类

1 类的概述

1.1 类与面向对象

面向对象编程是相对于面向过程编程而言的，它是一种对现实世界理解和抽象的方法，是计算机编程技术发展到一定阶段的产物，**是目前最有效的软件编写指导思想之一**。

面向过程编程主要是分析出实现需求所需要的步骤，通过函数和普通代码一步一步实现这些步骤。

面向对象编程则是分析出需求中涉及到哪些对象，这些对象各自有哪些特征、有什么功能，对象之间存在何种关系等，**将存在共性的事物或关系抽象成类。最后通过对象的组合和调用完成需求**。

1.1 类与面向对象

通常来讲，**面向过程编程效率更高**，容易理解。**面向对象编程**易维护、易扩展、易复用，灵活方便。

面向对象编程的**三个特点：封装(信息隐藏)、继承和多态**。

1.2 类的定义与创建

在面向对象编程中，**类是对具有相同属性和行为的一个或多个对象的抽象描述**。

实例是由某个特定的类所描述的一个具体对象(与现实世界相反)。

根据类来创建对象的过程被称为实例化。大部分时候，定义一个类就是为了重复创建该类的实例，同一个类的多个实例具有相同的特征，而类则是定义了多个实例的共同特征。类不是一种具体存在，实例才是具体存在。

1.2 类的定义与创建

Python**定义类的语法如下所示**:

```
class  <类名>:
    “””类说明文档”””
    类属性1
    ……
    类属性n
    <方法定义1>
    ……
    <方法定义n>
```

1.2 类的定义与创建

```
class Dog:     # 也可以是class Dog():
    “””一次模拟小狗的简单尝试”””
    sex = ‘公’     # 建议在初始化方法中定义类的属性
    def __init__(self, name, age):
        “””初始化属性name和age”””
        self.name = name
        self.age = age
    def sit(self):
```

1.2 类的定义与创建

```
        print(self.name.title() + “已经坐下.”)

    def roll_over(self):
        “””模拟小狗被命令时打滚”””
        print(self.name.title() + “打了个滚.”)

wang = Dog(“小旺”, 2)    # 创建类对象
wang.sit()    # 对象的方法调用
wang.roll_over()
```

1.2 类的定义与创建(类的说明)

1、在Python中，使用class关键词进行类定义(不是def)，类的首字母名称一般大写,**且使用驼峰命名方法，但类实例一般使用小写名称**;

2、**类名后面的圆括号可以省略，但冒号不能省略**;

3、类中的函数称为方法，与普通函数只是存在调用方式的区别;

4、**__init__方法(两个下划线，旨在与普通方法区别开来，但该方法不是必须的)是类的初始化方法**，当你创建实例时，Python都会自动调用该方法;

1.2 类的定义与创建(类的说明)

5、**类的所有实例方法都必须有一个self参数**，它用于指向实例本身的引用，在使用时，必须放在其他形参的前面；

6、类方法调用时，**self形参会自动传递**。因此我们根据Dog类创建实例时，只需要给最后两个形参(name和age)提供实参值；

7、**以self为前缀的变量可供类中的所有方法使用(但方法的形参必须要包含self)**，我们可以通过类实例来访问这些变量(**实例名.属性名**)，**这些变量也称为实例属性**；

8、**实例方法的调用与实例属性的调用类似；**

1.2 类的定义与创建(类的说明)

9、**在类中所有方法外定义的变量，称为类属性**，可以通过类名或实例名称进行访问;

10、类属性和实例属性, 都可以被外部修改;

11、在Python中，可动态为类和对象添加成员，这点和许多面向对象语言不同。

2 类的属性

2.1 实例属性

在Python中，你**可以直接修改实例的属性，也可以编写方法以特定的方式对属性进行修改，还可以动态添加实例属性**。

```
class Dog:
    ……
    def __init__(self, name, age):
        self.name = name
        self.age = age
    ……
```

2.1 实例属性

```
def update_age(self):
    self.age += 1
    print(f"{self.name}年龄为{self.age}.")
```

……

2.1 实例属性

```
wang = Dog("小旺", 2)
wang.update_age()
wang.age += 1   # 类外部调用实例属性, 使用实例名
print(f"{wang.name}年龄为{wang.age}.")
hua = Dog("小花", 3)
hua.tall = 70       # 动态添加实例属性, 实例属性不共享
print(f"{hua.name}身高{hua.age}cm.")
#print(f“{wang.name}身高{wang.age}cm.”)   # 该句会报错
```

2.2 类属性

类属性变量是在类中所有方法之外定义的变量，可以在所有实例之间共享。**类属性变量一般通过“类名.类属性变量名”进行访问**。

对“对象名.类属性变量名”进行赋值，相当于创建了一个同名的实例属性变量。

2.2 类属性(接前例)

```
Dog.type = “中华田园犬”      # 类属性, 为类实例共享

wang = Dog("小旺", 2)

print(f"{wang.name}为{wang.type}.")

hua = Dog("小花", 3)

print(f"{hua.name}为{hua.type}.")

hei = Dog("小黑", 3)

hei.type = “边境牧羊犬“   # 使用实例属性, 覆盖类属性

print(f"{hei.name}为{hei.type}.")

print(f“{hua.name}为{hua.type}.”)  # 不改变hua的属性
```

2.2 类属性(接前例)

```
class Toy:
    def __init__(self, tp):
        self.type = tp


    def show_toy(self):
        print(f"它有一个{self.type}.")
```

2.2 类属性(接前例)

```
buwa = Toy("布娃娃")

wang = Dog(“小旺”, buwa)   # 实例属性内容由实参值决定

wang.sit()

wang.roll_over()

wang.age.show_toy()

# wang.update_age()    # 报错
```

2.2 类属性(特殊属性-进阶)

在Python中，以下划线开头的变量名和方法名具有特殊的含义，尤其在类的定义中，更是如此。

1、_xxx: **以单个下划线开头的属性，称为保护成员**，模块中这样的对象默认不能使用from module import * 导入;

2、__xxx__: 特殊成员;

3、__xxx: **类的私有成员，只有类对象自身能访问**，子类对象也不能访问该成员(但在对象外部可通过‘对象名._类名__xxx’这种方式来访问)。

4 类的继承

编写类时，并非总要从空白开始。**如果你要编写的类是现有类的特殊版本，则可使用继承**。一个类继承另一个类时，它将获得另一个类的所有属性和方法，同时还可以定义自己的属性和方法；**原有的类称为父类，而新类称为子类**。

子类在继承父类时，如果父类存在初始化函数，则**必须先调用父类的初始化函数，否则将无法正确初始化子类对象**。调用函数为：super().__init__()

4 类的继承(接前例)

```
class Labrador(Dog):
    def __init__(self, name, age, favor):
        super().__init__(name, age)
        self.favor = favor

    def show_favor(self):
        print(f"{self.name}喜欢{self.favor}.")
```

4 类的继承

```
hei = Labrador("小黑", 2, "游泳")

hei.sit()   # 调用父类方法

hei.roll_over()

hei.update_age()

hei.show_favor()
```

4 类的继承

1、**创建子类时，必须在括号内指定父类的名称**;

2、**super()是一个特殊函数(圆括号不能少)，用来引用父类(即超类)**;

3、**在类中需要使用到的属性，尽量在__init__方法中先进行定义**(同样适用于父类);

4、**子类中相同名称的方法和属性，将会覆盖父类中的同名方法和属性**;

5、除非很熟练，应少使用多重继承(使用多重继承时，只需要在括号中依次列出父类名即可)。

4.4 多重继承

当子类继承多个父类，**创建子类对象时，将会默认按顺序创建所有父类的对象，即调用父类的__new__() 方法，但并不会调用所有父类的__init__() 方法**。

如**果子类没有提供自己的__init__()方法，将会默认调用第一个父类的__init__()方法**；如果子类提供了__init__()方法，则默认不会调用父类的__init__()，此时，父类的成员变量将无法初始化，因此，通常会在子类的 __init__() 方法中，显式调用父类的__init__() 方法。

4.4 多重继承(进阶)

```
class A:
    def __init__(self):
        self.count = 5
        self.num = 15
        print("A的初始化")
    def show(self):
        print("A的show()方法")
    def test(self):
        print("A的test()方法")
```

4.4 多重继承(进阶)

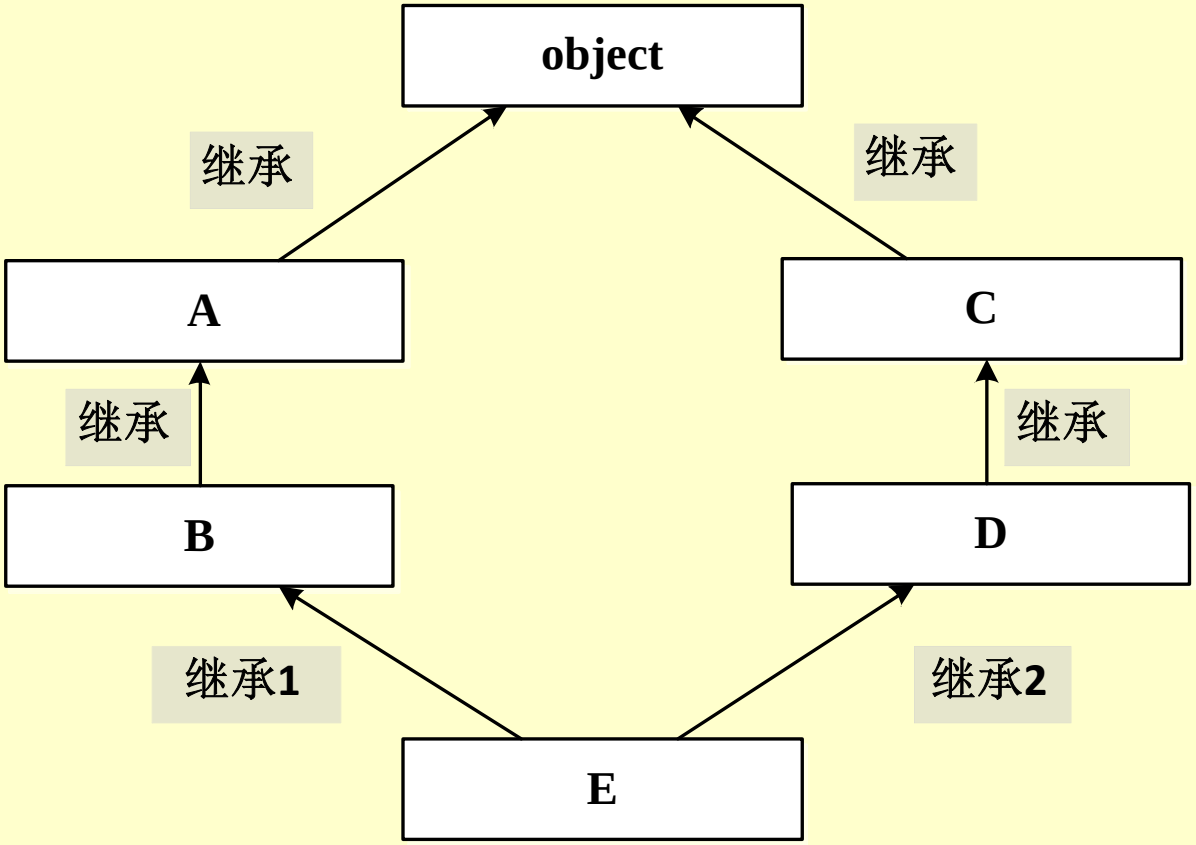
```
class B:
    def __init__(self):
        self.count = 10
        self.num = 20
        print("B的初始化")
    def test(self):
        print("B的test()方法")
```

4.4 多重继承(进阶)

```
class C(B, A):
    pass
c = C()
c.show()
c.test()
print("num:", c.num)
print("count:", c.count)
```

4.4 多重继承(进阶)

Python多重继承时的方法、属性搜索顺序如下图所示。



多重继承搜索顺序(E → B → A → D → C → object)

4.5 类的相关方法

issubclass(a, b): 判断类a是否为类b的子类;

a.__bases__: 返回类a的基类; # a.__base__

isinstance(a, b): 判断a是否为类b的实例, b可以是元组;

>>>isinstance(4.5, float) # True

>>>isinstance(4.5, (int, float)) # True

a.__class__: 判断对象a属于哪个类;

hasattr(a, 'b'): 判断对象a或类a是否包含属性或方法b;

getattr(a, 'b'): 获得对象a或类a的b属性或方法。

5 类的导入和类编码规则

5.1 类的导入

根据Python语言的简洁原则，应尽可能让文件内容整洁，功能单纯。因此，**我们应该将类存储在模块(即.py文件)中**，然后在主程序中导入需要使用的模块和类。

```
from car import Car  # 将Car类单独存储在car.py中, 再导入

# 将Car和ElectricCar类都存储在car.py中, 再导入

from car import ElectricCar

#同时导入Car和ElectricCar类

from car import Car, ElectricCar
```

5.1 类的导入

import car # 导入整个模块, 但使用类时, 需要加上模块名称

from car import * # 导入模块中所有的类, 但不推荐使用这种方式, 可能会导致类同名, 引发难以诊断的错误

为了解决类同名问题以及方便用户的使用习惯, 还可以为模块及函数指定别名。如:

>>>import math as mt

>>>from math import sqrt as sq

5.1 类的导入

还可以**使用以下命令来导入模块的路径**, 如此便可以直接导入路径下的模块。

```
>>> import sys  # sys.path中存放了模块的搜索路径列表
>>> sys.path.append('D:\Python\BasicPython\chapter 1')
# 可根据需要改为自己的模块路径
>>> from lesson1 import hello
>>> from lesson1 import importlib
#模块更新后, 可重新加载
>>>>hello = importlib.reload(hello)
```

5.1 类的导入

对于用户自己编写的模块，**如果希望能够使用import语句直接加载**，可以采用以下三种方式：

1、**将模块目录包含在系统环境变量PYTHONPATH中(标准做法，.; path;)**；

2、**将模块放在\lib\site-packages目录中(常用做法，压缩文件. gz, .rar, zip文件也是如此)**；

3、使用路径配置文件进行配置。

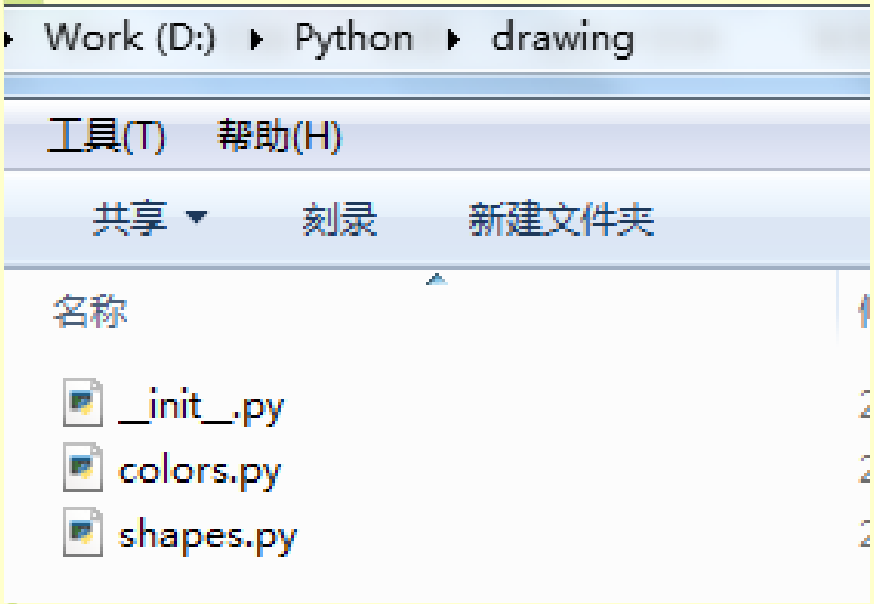
5.1 类的导入

提示：不同的类，最好采用不同的模块进行单独存储，以实现简化的原则。

对于用户自己编写的程序文件和模块，都尽量不要和系统提供的模块名称同名，以免导致命名冲突及加载函数错误.

5.1 类的导入

包其实就是另一种模块，并且可以包含其他模块。**模块存储在扩展名为.py的文件中，而包则是一个目录**。要被Python视为包，目录**必须包含__init__.py文件**(用来包含包的内容)。以下就是一种简单的包布局。



注意: __init__.py为空时，仅表示当前目录是个包

5.1 类的导入

对于这个包, 以下语句都是合法的。

```
import drawing    # 导入drawing包, 但不可以使用包中模块

import drawing.colors    #导入drawing包的colors模块

from drawing import shapes   #导入drawing包的shapes模块
```

5.2 类编码规则

1、**类名应采用驼峰命名法**，即类名中的每个单词的首字母都大写, 而不使用下划线;

2、**实例名和模块名都采用小写格式，并在单词间加上下划线**;

3、对于每个类，都应紧跟在类定义后面包含一个文档字符串, 这个字符串应简要描述类的功能;

4、在类中, 可使用一个空行来分隔方法;

5、在同一模块中, 可使用两个空行来分隔类;

5.2 类编码规则

6、需要同时导入多个模块时，**应先导入标准库模块，再导入扩展库的模块，最后导入你自己编写的模块，并使用空行来分隔它们**。这种做法让人更容易明白程序使用的各个模块来自何方。

5.2 类编码规则

在进行面向对象项目设计和开发时, 可遵从以下思路:

1、将有关问题描述记录下来，并给所有的名词、动词和形容词加上标记;

2、**在名词中寻找可能的类**;

3、**在动词中寻找可能的方法**;

4、**在形容词中寻找可能的属性**;

5、将找出的方法和属性分配给各个类。

5.2 类编码规则

在进行Python项目开发时, 以下为一些有效的建议。

1、**一开始应尽可能让代码结构简单**;

2、**先尽可能在一个文件中完成所有的工作，确定一切都能正常运行后，再将类和函数移到独立的模块中**(如果你很熟悉编写独立的模块, 则另当别论);

3、**先找出让你能够编写出可行代码的方式，再尝试让代码有序**。

Answers

在使用pip安装模块时, 有时会发生pip版本过低导致安装失败的情况。这时, 可按系统提示使用如下命令对pip进行升级:

python -m pip install --upgrade pip

如仍更新过慢, 则可使用下面的命令进行强制更新:

python -m pip install -U --force-reinstall pip

python -m pip --default-timeout=100 install -U pip

Answers

对于其他的模块安装也是类似, 如:

pip install -U pyperclip --force-reinstall –user

还可以使用镜像地址(这种方法最快)

pip install -U pyperclip --user -i https://pypi.douban.com/simple

注:pip工具用来从Python社区下载和安装模块, 其类似于Python模块的免费应用程序商店(https://pypi.python.org)。

练习

01 编写教材及PPT上的例题和练习

02 完成课后编程题1、2、3

谢 谢