类 和对象

面向对象概述

面向对象(Object Oriented)的英文缩写是OO，它是一种设计思想。我们常说的面向对象编程(Object Oriented Programming) ,即OOP就是主要针对大型软件设计而提出的，它可以是软件设计更加灵活，并且能更好的进行代码复用。

1. 对象

   对象，是一个抽象概念，英文称作"Object"，表示任意存在的事物。世间万物皆对象。现实世界中，随处可见的一种事物就是对象，如一个人。

   通常将对象划分为两个部分，即静态部分与动态部分。静态部分被称为"属性"，任何对象都具备自身属性，如人的性别。动态部分指的是对象的行为，即对象执行的动作，如人可以行走。

2. 类

   类是封装对象的属性和行为的载体，反过来说具有相同属性和行为的一类实体被称为类。

3. 面向对象的特点

   面向对象程序设计具有三大特征：封装，继承，多态。

   - 封装：将对象的属性和行为封装起来，其载体就是类，通常会对客户隐藏其细节的实现方式，这就是封装的思想。例如：我们使用计算机只需要操作键盘和鼠标来实现一些功能，而无需知道计算机内部是如何工作的。
   - 继承：是实现重复利用的重要手段，子类通过继承复用了父类的属性和行为。
   - 多态：子类继承父类特征的同时，也具备了自己的特征，并能够实现不同的效果，这就是多态化的结构。

类的定义和使用

1. 定义类

   在Python中类的定义使用class关键字来实现，语法：

   > class 类名:
   >
   > 类的实体内容

   ```
   # 定义一个Dog类
   class Dog():
       '''
       定义一个Dog类
       暂时不需要具体内容，用pass占位
       '''
       pass
   ```

2. 创建类的实例

   在定义完成后，并不会真正创建一个实例。就像图纸一样，只是告诉你这是什么，有什么功能。

```
# 创建类的实例
myDog = Dog()
print(myDog)
```

```
PS F:\笔记相关\Python基础第二次分享> python .\day09\demo.py
<__main__.Dog object at 0x00000158548D3E80>
PS F:\笔记相关\Python基础第二次分享>
```

> 在Python中创建实例不适用new关键字，这是它与其它面向对象语言的区别。

3. __init__()方法

这个方法也被称为魔法方法，在创建类后，类通常会自动创建一个__init__()方法，类似于Java中的构造方法。每当创建一个新的实例时，Python都会自动的执行它。这个方法必须包含一个self参数，且必须是第一个参数。

```
# 创建一个人Dog类
class Dog:
    '''狗类'''
    def __init__(self):
        print("我是狗类")

myDog= Dog() # 创建狗类的实例
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo1.py
我是狗类
PS F:\笔记相关\Python基础第二次分享\day09>
```

4. 创建类的成员并访问

和__init__()方法一样，实例方法的第一个参数必须是self。语法：

> def 方法名(self, 参数列表)
>
> 具体实现内容

```
class Dog:
    def __init__(self):
        print("Dog类")

    def sayHi(self):
        print("汪汪。。。")


# 访问Dog类中的sayHi方法
myDog = Dog()
myDog.sayHi()
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo2.py
Dog类
汪汪。。。
PS F:\笔记相关\Python基础第二次分享\day09>
```

访问类中的方法，可以通过实例名(myDog). 类的方法(sayHi)的方式来访问

除了访问方法，还可以访问类中的属性。也是通过.的方式来访问。

- 类属性: 定义在类中，并且在函数体外的属性。类属性可以通过类名或实例名来访问。

```
class Dog:
    '''Dog类'''
    leg = '4条腿跑的快'
    def __init__(self):
        print("这是一个Dog类")
        print(Dog.leg)

# 访问类属性
myDog = Dog()
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo3.py
这是一个Dog类
4条腿跑的快
PS F:\笔记相关\Python基础第二次分享\day09>
```

- 实例属性：定义在类的方法中的属性，只用作当前实例

```
class Car:
    '''汽车类'''
    def __init__(self):
        self.color = '黑色'
        print(self.color)

car1 = Car()
car2 = Car()
print("car1的颜色",car1.color)
car2.color = "红色"
print("car2的颜色",car2.color)
```

对于实例属性是可以通过实例名来来狗的

5. 访问限制

- 在属性前面加一个下划线(_)表示受保护的类型成员，只允许类本身和子类进行访问，但不能使用“from module import *”语句来导入。（后面会讲到）

```
class Dog:
    _color = '黑色' #私有属性
    def __init__(self):
        print("__init__:", Dog._color) # 在私立方法中访问私有属性

myDog = Dog()
print("直接访问", myDog._color)
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo4.py
__init__: 黑色
直接访问 黑色
PS F:\笔记相关\Python基础第二次分享\day09>
```

可以看出来保护属性是可以通过实例名来访问的。

- 在属性卡安眠加双下划线，表示私有的只允许定义该方法的类本身访问，而且不能通过类的实例进行访问。

```
class Dog:
    __color = "黑色" # 私有属性
    def __init__(self):
        print("__init__:", Dog.__color)

myDog = Dog()
print("通过类名访问", myDog._Dog__color) #私有属性可以通过实例名._类名__xx的方式访问
print("直接访问:", myDog.__color) # 这里会报错
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo5.py
__init__: 黑色
通过类名访问 黑色
Traceback (most recent call last):
  File ".\demo5.py", line 8, in <module>
    print("直接访问:", myDog.__color) # 这里会报错
AttributeError: 'Dog' object has no attribute '__color'
PS F:\笔记相关\Python基础第二次分享\day09>
```

属性

1. 创建用于计算的属性

   在Python中，可以通过@property（装饰器）将一个方法转换为属性，从而实现用于计算的属性

```
class Rect:
    def __init__(self, width, height):
        self.width = width #矩形的宽
        self.height = height # 矩形的高
    @property
    def area(self):
        return self.width * self.height #计算矩形的面积

rect = Rect(10, 20)
print("面积为:", rect.area)
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo6.py
面积为: 200
PS F:\笔记相关\Python基础第二次分享\day09>
```

2. 为属性添加安全保护机制

   在Python中默认的情况创建的类属性或者实例，是可以在类体外进行修改的，如果想要限制其不能在类体外修改，可以将其设置为私有，但是设置为私有后，在类体外也不能获取它的值。如果想创建一个只读的属性可以使用@property来实现

```
class TVshow:
    def __init__(self, show):
        self.__show = show
    @property
    def show(self):
        return self.__show

tvshow = TVshow("正在播放<<海贼王>>")
print("默认:", tvshow.show)

tvshow.show = "这在播放<<火影>>" # 修改属性 这里会报错
print("修改后:", tvshow.show) # 获取属性值
```

```
PS F:\笔记相关\Python基础第二次分享\day09> python .\demo7.py
默认: 正在播放<<海贼王>>
Traceback (most recent call last):
  File ".\demo7.py", line 11, in <module>
    tvshow.show = "这在播放<<火影>>" # 修改属性 这里会报错
AttributeError: can't set attribute
PS F:\笔记相关\Python基础第二次分享\day09>
```

继承

1. 继承的基本语法

   > class 类名(父类):
   >
   > 具体内容

2. 方法重写

```
class Fruit:
    color = '绿色'
    def harvest(self, color):
        print("水果是", color , '的。')
        print("水果已经获得")
        print('水果原来是',color,'的。')

class Orange(Fruit):
    color = '橙色'
    def __init__(self):
        print("\n我是橘子")

    def harvest(self, color):
        print("橘子是", color , '的。')
        print("橘子已经获得")
        print('橘子原来是',color,'的。')
```

3. 派生类中调用基类的__init__()方法

   在子类中是不会调用父类的 _ _ init_ *()方法，如果需要调用的话，可以使用super() . _*init__()

```
```

```python
class Fruit:
    def __init__(self, color='绿色'):
        Fruit.color = color

    def harvest(self):
        print("水果是", Fruit.color , '的。')
        print("水果已经获得")
        print('水果原来是',Fruit.color,'的。')

class Orange(Fruit):
    color = '橙色'
    def __init__(self):
        print("\n我是橘子")

class Apple(Fruit):
    color = '橙色'
    def __init__(self):
        print("\n我是苹果")
        super().__init__()

orange = Orange()
# orange.harvest() # 会报错 type object 'Fruit' has no attribute 'color'
apple = Apple()
apple.harvest()
```