面向对象

软件测试Python课程V5.0

黑马程序员 www.itheima.com 传智教育旗下 高端IT教育品牌

学习目标

Learning Objectives

1. 理解面向对象的基本概念
2. 掌握面向对象的基本语法
3. 掌握面向对象的封装和继承特征

目录

Contents

面向对象编程介绍

01

- 面向对象基本概念
- 类和对象的概念
- 类的设计

学习目标

1. 理解面向对象的基本概念
2. 能够说出什么是类和对象
3. 掌握如何设计类

面向对象基本概念

面向对象编程: Object Oriented Programming 简写 OOP

在面向对象的世界里：
万事万物皆对象

面向过程

- 根据需求，将某些独立功能封装成一个又一个函数
- 最后完成的代码，就是顺序地调用不同的函数

面向对象

- 相比较函数，面向对象是更大的封装，根据职责在一个对象中封装多个方法
- 根据职责确定不同的对象，在对象内部封装不同的方法

面向对象的核心内容	**类** 和 **对象**

类和对象的概念

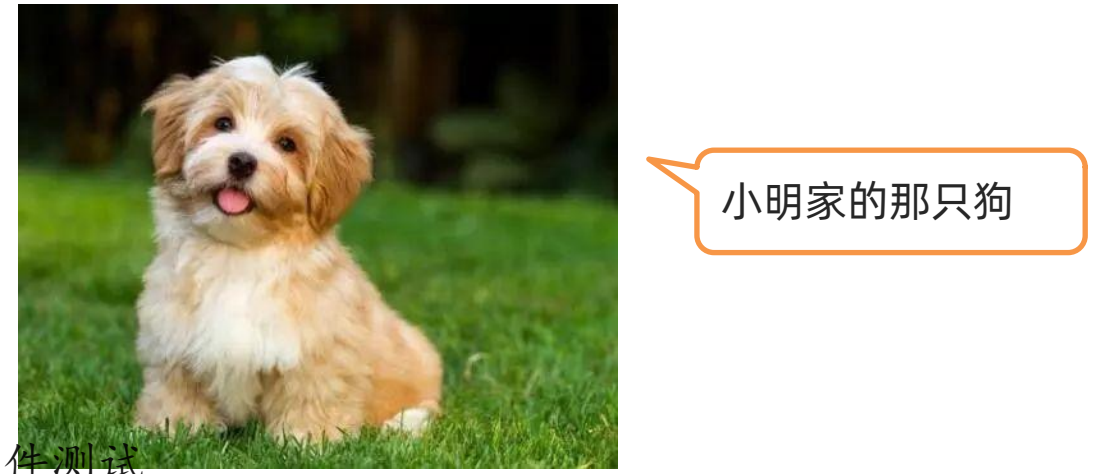
类：是对具有相同特征或者行为的事物的一个统称，是抽象的,不能直接使用

类包含的主要内容：

- 特征(静态) -> 属性
- 行为(动态) -> 方法

对象：是由类创建出来的一个具体存在的事物，可以直接使用

说明：由哪一个类创建出来的对象，就拥有在哪一个类中定义的 属性和方法



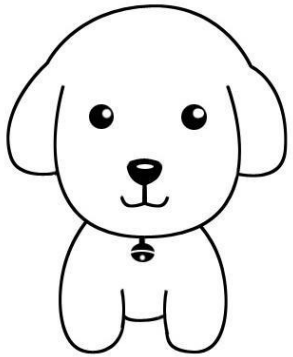
类的设计

想要设计一个类, 通常需要满足以下三个要素:

- **类名**: 同类事物的名字, 命名要满足大驼峰命名法
- **属性**: 同类事物具备的特征
- **方法**: 同类事物具备的行为

大驼峰命名法: 所有单词第一个字母大写, 单词之间没有下划线!

类名的提取：使用名词提炼法分析整个业务流程, 得出的名词, 通常就是类名



把我变成一个类，会是什么样子?

类名：狗

属性：名字、年龄、身高、体重...

方法：吃、跑、拿耗子...

注意 对于需求中没有提到的属性和方法, 在设计类的时候不要添加进来!

类的设计-案例演练1

案例演练

需求：

- **小明** 今年 **18 岁**，**身高 1.75**，每天早上 **跑** 完步，会去 **吃** 东西
- **小美** 今年 **17 岁**，**身高 1.65**，小美不跑步，小美喜欢 **吃** 东西

Person
name age height
run() eat()

练习

类的设计-练习1

需求：

- 一只 黄颜色 的 狗狗 叫 大黄
- 看见生人 汪汪叫
- 看见家人 摇尾巴

类的设计-案例演练2

案例演练

需求：

- 进入某 Web 项目登录页面，输入用户名、密码、验证码之后，点击登录按钮可以登录系统

LoginPage
username password verify_code login_btn
login()

练习

类的设计-练习2

需求：进入某APP项目的下订单页面，输入用户收货地址后，点击提交订单按钮，可以实现下单操作

1. 面向对象包含的两个核心内容是什么？

2. 类和对象的区别是什么？

3. 想要设计一个类，通常需要哪三个要素？

目录

Contents

面向对象基本语法

02

- 类的定义和创建对象
- self 参数
- 初始化方法
- 内置函数和方法

学习目标

1. 掌握如何定义类和创建对象
2. 理解self参数的含义及用法
3. 掌握初始化方法的作用和用法

类的定义(只含方法)

定义一个只包含方法的类，语法格式如下：

```
# 类名: 遵循大驼峰命名法
class 类名:


    # 方法: 定义方式与函数基本相同, 区别是方法的第一个参数必须为 self
    def 方法1(self, 参数列表):
        pass


    def 方法2(self, 参数列表):
        pass
```

大家暂时先记住，稍后详细介绍 self

创建对象

使用定义好的类创建对象，语法格式如下：

```
对象变量 = 类名()
```

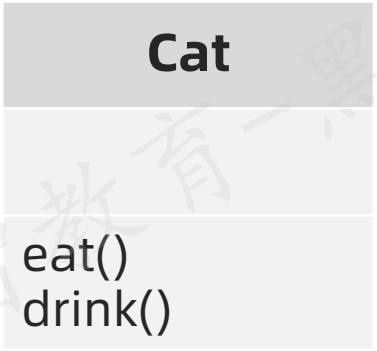
注意 在Python中,由类创建对象的操作又被称之为 **实例化**

第一个面向对象程序

需求：小猫 爱 吃鱼，小猫 要 喝水

分析：

- 定义一个猫类 Cat
- 定义两个方法 eat 和 drink
- 按照需求--不需要定义属性




```
class Cat:
    """这是一个猫类"""

    def eat(self):
        print("小猫爱吃鱼")

    def drink(self):
        print("小猫在喝水")

tom = Cat()
tom.drink()
tom.eat()
```

创建另一个对象

思考：如果使用同一个类实例化出另一个对象，这个两个对象是同一个对象么?

```
class Cat:

    def eat(self):
        print("小猫爱吃鱼")

    def drink(self):
        print("小猫在喝水")

tom = Cat()
tom.drink()
tom.eat()

lazy_cat = Cat()
lazy_cat.eat()
lazy_cat.drink()
```

tom 和 lazy_cat 是同一个对象吗?

提示	可以通过直接打印两个对象变量, 或者使用Python内置函数id(),查看两个对象的信息

self 参数

说明	由哪一个对象调用的方法, 方法内的 self 就是哪一个对象的引用(可以通过代码调试得出)

注意:

- 在 类封装的方法中，通过 self. 访问对象的属性和方法
- 在 类的外部，通过 对象变量名. 访问对象的属性和方法

```
class Cat:
    def eat(self):
        print(f"{self.name} 爱吃鱼")


tom = Cat()
tom.name = "Tom"
tom.eat()

lazy_cat = Cat()
lazy_cat.name = "大懒猫"
lazy_cat.eat()
```

注意：在类的外部的代码中直接通过"对象变量名.属性名"即可设置一个属性，但是不推荐使用！！！了解即可~

初始化方法

- 当使用 类名() 创建对象时，会 自动 执行以下操作：
  1. 为对象在内存中 分配空间 —— 创建对象
  2. 为对象的属性 设置初始值 —— 初始化方法(init)
- 这个 初始化方法 就是 __init__ 方法，__init__ 是对象的内置方法
- __init__ 方法是 专门 用来定义一个类具有哪些属性的方法！

总结一下	初始化方法就是指的__init__()方法，创建对象的时候会自动调用，用来定义类具有哪些属性

```
class Cat:
    def __init__(self):
        print("初始化方法")

        # 定义一个 name 属性
        self.name = "Tom"

    def eat(self):
        print(f"{self.name} 爱吃鱼")


tom = Cat()
tom.eat()
```

在初始化方法中增加对象属性

如果希望**在创建对象的同时，就设置对象的属性**：

1. 把希望设置的属性值，定义成 __init__ 方法的参数
2. 在方法内部使用 self.属性 = 形参 接收外部传递的参数
3. 在创建对象时，使用 类名(属性1, 属性2...) 调用

```
class Cat:
    def __init__(self, name):
        print("初始化方法")

        # 定义一个 name 属性
        self.name = name

    def eat(self):
        print(f"{self.name} 爱吃鱼")


tom = Cat("Tom")
tom.eat()
```

内置函数和方法

序号	方法名	类型	作用
1	dir	函数	可以查看对象内的 所有属性及方法
2	__str__	方法	返回对象的描述信息，print 函数输出使用

dir()函数

dir(对象变量)

作用：可以查看对象内的 所有属性及方法

```
class Cat:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print(f"{self.name} 爱吃鱼")


if __name__ == '__main__':
    tom = Cat("Tom")
    print(dir(tom))
```

注意	dir()函数不具备输出功能, 需要和print()函数配合使用

__str__ 方法

说明: 默认情况下使用 print() 函数打印对象变量时, 会输出对象的内存地址信息, 如果想要自定义打印对象变量时的信息, 则可以使用 __str__ 方法

注意	__str__ 方法必须返回一个字符串数据!

```
class Cat:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print(f"{self.name} 爱吃鱼")

    def __str__(self):
        return f"我是小猫：{self.name}"


if __name__ == '__main__':
    tom = Cat("Tom")
    print(tom)
```

1. 如何定义类和创建对象？

2. self参数的含义及用法？

3. 初始化方法的作用和用法？

目录

Contents

面向对象的三大特征

03

- 封装
- 继承
- 多态

学习目标

1. 掌握面向对象封装的具体实现
2. 掌握面向对象继承的具体实现
3. 了解多态的概念

面向对象的三大特性介绍

封装

根据需求将属性和方法封装到一个抽象的类中

继承

实现代码的重用，相同的代码不需要反复的编写

多态

不同的对象调用相同的方法，产生不同的执行结果，增加代码的灵活度

面向对象的三大特征

封装

封装的概念:

- 根据需求将属性和方法封装到一个抽象的类中的过程即为封装
- 外界使用 类 创建 对象，然后让对象调用方法
- 对象方法的细节都被 封装 在类的内部

怎么实现我不管，能够满足我的要求就行！

案例演练

封装案例一：小明爱跑步

需求：

- 小明 体重 75.0 公斤
- 小明每次 跑步 会减肥 0.5 公斤
- 小明每次 吃东西 体重增加 1 公斤

Person
name weight
__init__(self, name, weight): __str__(self): run(self): eat(self):

案例演练

封装案例一：小明爱跑步

```
class Person:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight

    def run(self):
        print(f"{self.name} 爱跑步，跑步锻炼身体")
        self.weight -= 0.5

    def eat(self):
        print(f"{self.name} 是吃货，吃完这顿再减肥")
        self.weight += 1

    def __str__(self):
        return f"我的名字叫{self.name} 体重：{self.weight} 公斤"


xiaoming = Person("小明", 75)
xiaoming.run()
xiaoming.eat()
print(xiaoming)
```

封装案例一：小明爱跑步扩展--小美也爱跑步

案例演练

需求：

- 小明 和 小美 都爱跑步
- 小明 体重 75.0 公斤
- 小美 体重 45.0 公斤
- 每次 跑步 都会减少 0.5 公斤
- 每次 吃东西 都会增加 1 公斤

```
if __name__ == '__main__':
    xiaoming = Person("小明", 75)
    xiaoming.run()
    xiaoming.eat()
    print(xiaoming)

    xiaomei = Person("小美", 45)
    xiaomei.run()
    xiaomei.run()
    xiaomei.eat()
    print(xiaomei)
```

提示 同一个类 创建的 多个对象 之间，属性 互不干扰！

封装案例二：登录

案例演练

需求：

- 进入某 Web 项目登录页面，输入用户名、密码、验证码之后登录系统

要求：

- 模拟实现登录操作，输出每一步的登录信息

封装案例二：登录

案例演练

```
class LoginPage:
    """登录页面"""

    def __init__(self, username, password, verify_code):
        self.username = username
        self.password = password
        self.verify_code = verify_code

    def login(self):
        """登录方法"""
        print(f'输入用户名: {self.username}')
        print(f'输入密码: {self.password}')
        print(f'输入验证码: {self.verify_code}')
        print('点击登录按钮')

if __name__ == '__main__':
    login_page = LoginPage('admin', '123456', '8888')
    login_page.login()
```

案例演练

封装案例三：摆放家具

需求：

1. 房子(House) 有 户型、总面积 和 家具名称列表
   - 新房子没有任何的家具
2. 家具(HouseItem) 有 名字 和 占地面积，其中
   - 席梦思(bed) 占地 4 平米
   - 衣柜(chest) 占地 2 平米
   - 餐桌(table) 占地 1.5 平米
3. 将以上三件 家具 添加 到 房子 中
4. 打印房子时，要求输出：户型、总面积、剩余面积、家具名称列表

封装案例三：摆放家具

类的设计：

案例演练

HouseItem
name area
__init__(self, name, area) __str__(self)

House
house_type area free_area item_name_list
__init__(self, house_type, area) __str__(self) add_item(self, item)

剩余面积	1.在创建房子对象时，定义一个 剩余面积的属性，初始值和总面积相等 2.当调用 add_item 方法，向房间 添加家具 时，让 剩余面积 -= 家具面积

封装案例三：摆放家具

案例演练

创建家具：

```
class HouseItem:
    """家具"""

    def __init__(self, name, area):
        """
        :param name: 家具名称
        :param area: 家具面积
        """
        self.name = name
        self.area = area

    def __str__(self):
        return f"[{self.name}]占地面积：{self.area}"
```

案例演练

封装案例三：摆放家具

创建房间：

```
class House:
    """房子"""

    def __init__(self, house_type, area):
        self.house_type = house_type
        self.area = area
        self.free_area = area
        self.item_name_list = []

    def add_house_item(self, house_item):
        # 判断家具面积是否大于房子剩余面积
        if house_item.area > self.free_area:
            print(f"{house_item.name}的面积太大了，不能添加到房子中！")
            return

        print(f"添加了{house_item}")
        self.item_name_list.append(house_item.name)
        self.free_area -= house_item.area

    def __str__(self):
        return f"户型:{self.house_type} 总面积:{self.area} 剩余面积:{self.free_area} 家具名称列表:{self.item_name_list}"
```

面向对象的三大特征

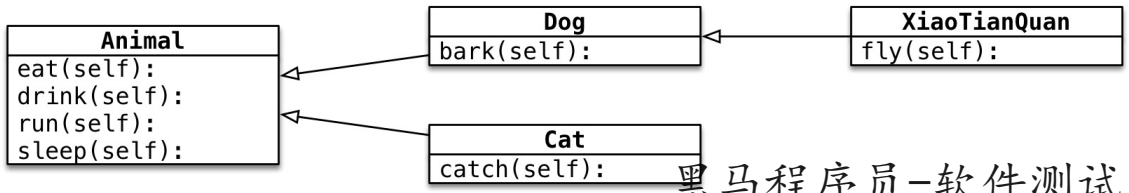
继承

说明: 如果多个类中存在相同的代码逻辑, 则可以考虑将相同逻辑的代码抽取封装到父类中, 再通过继承关系, 直接实例化子类对象并调用父类中的方法使用即可, 进而可以避免反复编写相同逻辑的代码

注意	在继承关系中, 子类可以拥有父类的所有方法和属性

不使用继承

使用继承



继承的语法

```
class 类名(父类名):
    pass
```

```
class A(B):
    pass
```

- 子类 继承自 父类，可以直接 享受 父类中已经封装好的方法，不需要再次开发
- 子类 中应该根据 职责，封装 子类特有的 属性和方法

继承的专业术语：

- A 类是 B 类的子类, B 类是 A 类的父类, A 类从 B 类继承
- A 类是 B 类的派生类，B 类是 A 类的基类, A 类从 B 类派生

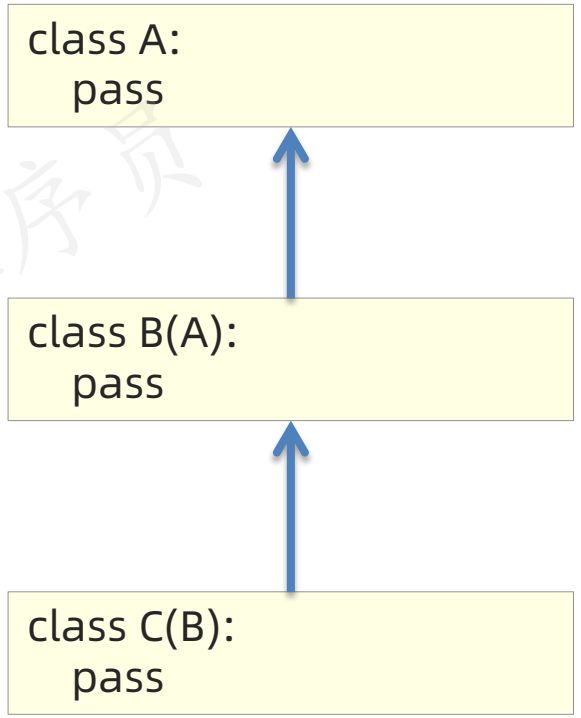
提示：
父类有时候也称为：基类、超类

继承的传递性

说明:

- C 类从 B 类继承, B 类又从 A 类继承
- 那么 C 类就拥有 B 类和 A 类的所有属性和方法

结论 子类拥有父类以及父类的父类中封装的所有属性和方法



案例演练

继承案例

需求：

1. 定义动物类，动物有姓名和年龄属性，具有吃和睡的行为
2. 定义猫类，猫类具有动物类的所有属性和方法，并且具有抓老鼠的特殊行为
3. 定义狗类，狗类具有动物类的所有属性和方法，并且具有看门的特殊行为
4. 定义哮天犬类，哮天犬类具有狗类的所有属性和方法，并且具有飞的特殊行为

案例演练

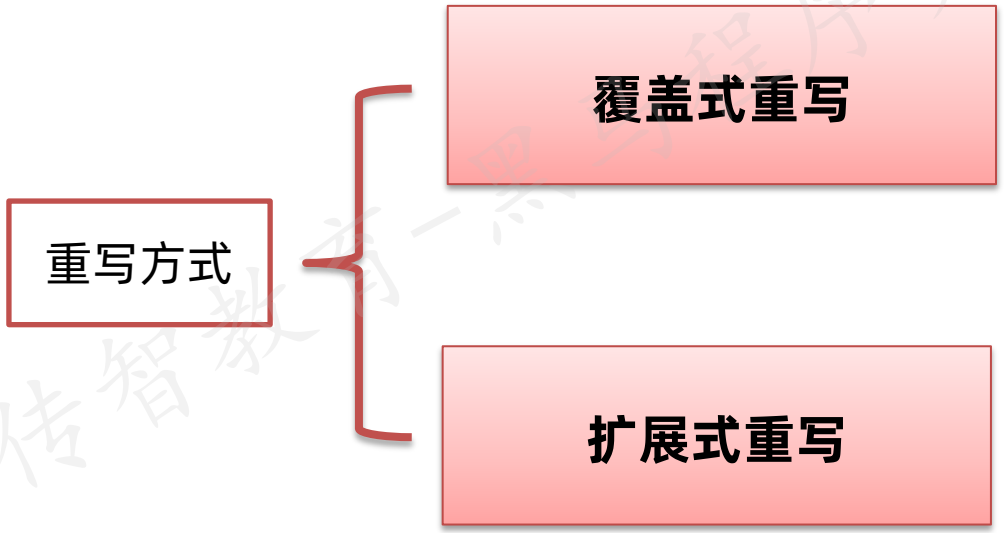
继承案例

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

    def eat(self):
        print("动物吃东西")

    def sleep(self):
        print("动物在睡觉")

class Cat(Animal):
    def catch(self):
        print("猫抓老鼠")

class Dog(Animal):
    def look_door(self):
        print("狗看门")

class XiaoTianQuan(Dog):
    def fly(self):
        print("哮天犬在飞翔")
```

```
tom = Cat("汤姆", 2)
tom.eat()
tom.sleep()
tom.catch()

dh = Dog("大黄", 3)
dh.eat()
dh.sleep()
dh.look_door()

xtq = XiaoTianQuan("神犬", 100)
xtq.eat()
xtq.sleep()
xtq.look_door()
xtq.fly()
```

方法的重写

说明	当父类中的方法实现不能满足子类需求时，可以对父类中的方法进行重写(override)



方法的重写-覆盖式重写

应用场景：

- 如果在开发中，父类的方法实现 和 子类的方法实现，完全不同
- 就可以使用 覆盖 的方式，在子类中 重新编写 父类的方法实现

实现方式	相当于在 子类中 定义了一个 和父类同名的方法并且实现

注意：
重写之后，在运行时，只会调用 子类中重写的方法，而不再会调用 父类封装的方法

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

    def eat(self):
        print("动物吃东西")

class Cat(Animal):
    def eat(self):
        print("猫吃鱼")


tom = Cat("汤姆", 2)
tom.eat()
```

方法的重写-扩展式重写

应用场景: 如果子类中重写父类的方法时, 需要包含父类中方法的实现, 则可以考虑使用扩展式

实现步骤:

1. 在子类中重写父类方法
2. 在子类中需要的位置使用 **super().父类方法名** 调用父类方法执行

关于super:

- 在 Python 中 super 是一个 特殊的类
- super() 就是使用 super 类创建出来的对象
- 最常 使用的场景就是在 重写父类方法时，调用 在父类中封装的方法实现

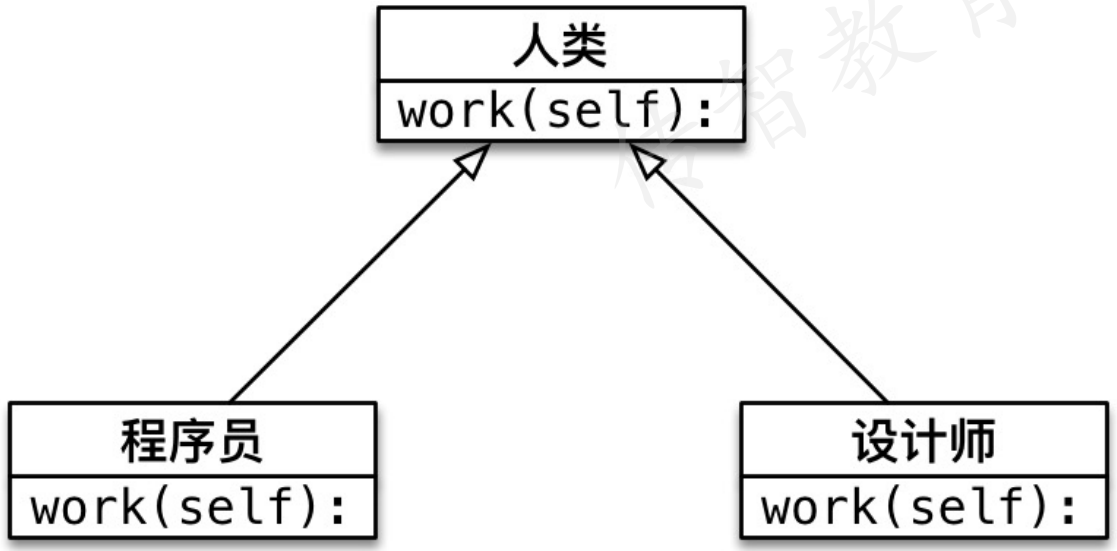
```
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def eat(self):
        print("动物吃东西")

class Cat(Animal):
    def eat(self):
        super().eat()
        print("猫吃鱼")


tom = Cat("汤姆", 2)
tom.eat()
```

面向对象的三大特征

多态

多态：不同的子类对象调用相同的父类方法，产生不同的执行结果

- 多态 可以 增加代码的灵活度
- 以**继承**和**重写父类方法**为前提
- 是调用方法的技巧，不会影响到类的内部设计



注意：

- 在Python中可以为一个变量赋值为不同类型的对象，所以在代码中多态体现不够明显
- 在自动化测试脚本几乎应用不到，了解多态即可

案例演练

多态案例

需求：

1. 在 Dog 类中封装方法 game
   - 普通狗只是简单的玩耍
2. 定义 XiaoTianDog 继承自 Dog，并且重写 game 方法
   - 哮天犬需要在天上玩耍
3. 定义 Person 类，并且封装一个 和狗玩 的方法
   - 在方法内部，直接让 狗对象 调用 game 方法

多态案例

案例演练

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

    def game(self):
        print(f"{self.name} 在玩耍...")


class XiaoTianDog(Dog):
    def game(self):
        print(f"{self.name} 飞到天上去玩耍...")


class Person(object):
    def __init__(self, name):
        self.name = name

    def game_with_dog(self, dog):
        print(f"{self.name} 和 {dog.name} 快乐的玩耍...")
        # 让狗玩耍
        dog.game()
```

```
# 1. 创建一个狗对象
wangcai = XiaoTianDog("飞天旺财")

# 2. 创建一个小明对象
person = Person("小明")

# 3. 让小明调用和狗玩的方法
person.game_with_dog(wangcai)
```

1. 面向对象的三大特征是什么？

2. 什么是封装？封装的好处是什么？

3. 什么是继承？继承的好处是什么？

目录

Contents

面向对象其他语法

04

- 类属性和类方法
- 私有属性和私有方法
- 静态方法

学习目标

1. 掌握类属性和类方法的用法
2. 掌握私有属性和私有方法的用法
3. 掌握静态方法的用法

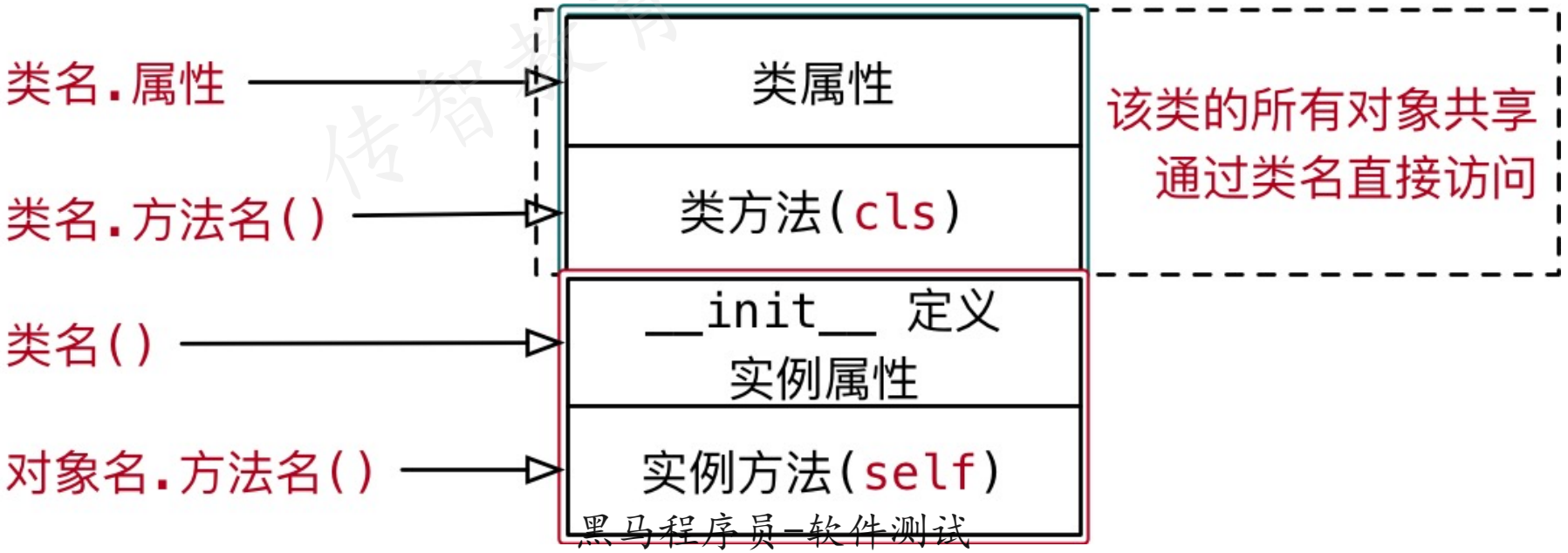
类是一个特殊的对象

说明: 程序运行时, 类会被加载到内存中, 在Python中, 类是个特殊的对象 -> 类对象

特点:

1. 在内存中, 类对象只有一份, 类对象可以拥有自己的属性和方法, 分别为类属性和类方法

2. 可以使用: 类名.类属性 和 类名.类方法() 方式获取类属性和调用类方法



类属性

- 类属性 就是给 类对象 中定义的 属性
- 通常用来记录 与这个类相关 的特征
- 类属性 不会用于记录 具体对象的特征

调用方式	类名.类属性名

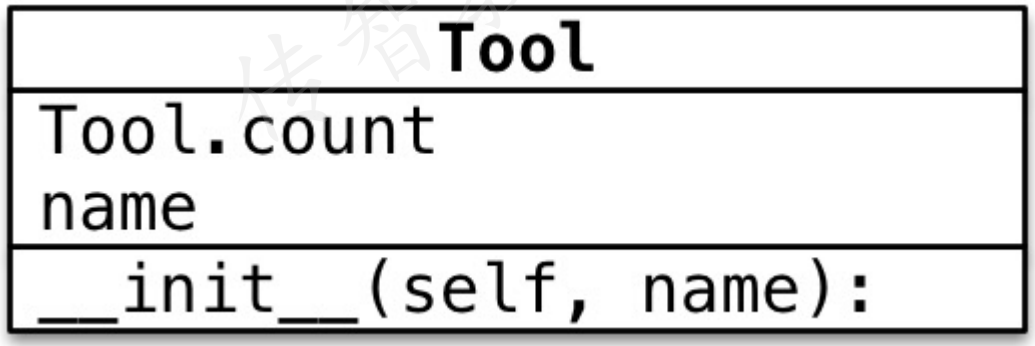
注意：类属性被该类的所有对象所共享！

案例演练

类属性案例

需求：

1. 定义一个 工具类
2. 每件工具都有自己的 name
3. 需求 —— 知道使用这个类，创建了多少个工具对象？



案例演练

类属性案例

示例代码：

```
class Tool(object):
    # 使用赋值语句，定义类属性，记录创建工具对象的总数
    count = 0

    def __init__(self, name):
        self.name = name
        # 针对类属性做一个计数+1
        Tool.count += 1


# 创建工具对象
tool1 = Tool("斧头")
tool2 = Tool("榔头")
tool3 = Tool("铁锹")

# 知道使用 Tool 类到底创建了多少个对象?
print(f"现在创建了 {Tool.count} 个工具")
```

类方法

类方法：就是针对 类对象 定义的方法，在类方法内部可以直接访问 类属性 或者调用其他的 类方法

语法：

```
class Demo:
    @classmethod
    def 类方法名(cls):
        pass
```

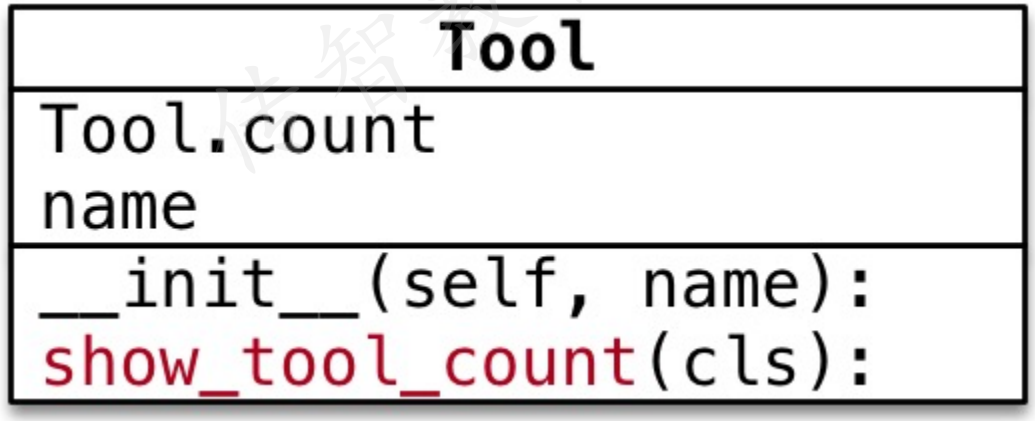
- 类方法需要用 装饰器 @classmethod 来标识
- 类方法的 第一个参数 应该是 cls
  - 由 哪一个类 调用的方法，方法内的 cls 就是 哪一个类的引用
  - 这个参数和 实例方法 的第一个参数是 self 类似
- 通过 类名. 调用 类方法，调用方法时，不需要传递 cls 参数
- 在方法内部
  - 可以通过 cls. 访问类的属性
  - 也可以通过 cls. 调用其他的类方法

案例演练

类方法案例

需求：

1. 定义一个 工具类
2. 每件工具都有自己的 name
3. 在类中封装一个 show_tool_count 的类方法，输出使用当前这个类，创建的对象个数



案例演练

类方法案例

```
class Tool(object):
    # 使用赋值语句，定义类属性，记录创建工具对象的总数
    count = 0

    def __init__(self, name):
        self.name = name
        # 针对类属性做一个计数+1
        Tool.count += 1

    @classmethod
    def show_tool_count(cls):
        """显示工具对象的总数"""
        print(f"工具对象的总数={cls.count}")


# 创建工具对象
tool1 = Tool("斧头")
tool2 = Tool("榔头")
tool3 = Tool("铁锹")

# 调用类方法
Tool.show_tool_count()
```

应用场景

说明: 在后期的自动化测试过程中, 在某些些业务场景下, 需要保证生成的对象始终只有一个时, 就可以使用类对象及类属性和类方法

```
from selenium import webdriver

class DriverUtil:
    """驱动对象管理类"""
    __driver = None  # 初始情况驱动对象为空

    @classmethod
    def get_driver(cls):
        """获取驱动对象方法"""
        if cls.__driver is None:  # 如果驱动对象为空
            cls.__driver = webdriver.Chrome()  # 创建驱动对象
        return cls.__driver  # 返回驱动对象

if __name__ == '__main__':
    DriverUtil.get_driver()  # 调用获取驱动对象方法
```

好处：简化方法的调用步骤(无需实例化对象，直接通过类名.方法名即可调用)

私有属性和私有方法

说明: 当属性和方法只需要在类定义内部使用时, 就可以使用私有属性和私有方法

特点: 在类定义外部, 无论是通过类对象还是实例对象均无法获取私有属性和调用私有方法

语法:

1> 私有属性: __属性名

2> 私有方法: __方法名()

注意：在继承关系中, 父类中的私有属性和私有方法, 子类同样无法获取和调用

调用方式	在类定义内部, cls.__属性名/cls.__方法名()/self.__属性名/self.__方法名()

私有属性和私有方法

示例代码：

```
class Women:

    def __init__(self, name):
        self.name = name
        # 不要问女生的年龄
        self.__age = 18

    def __secret(self):
        print(f"我的年龄是 {self.__age}")


xiaofang = Women("小芳")
# 私有属性，外部不能直接访问
# print(xiaofang.__age)

# 私有方法，外部不能直接调用
# xiaofang.__secret()
```

静态方法

- 在开发时，如果需要在 类 中封装一个方法，这个方法：
  - 既 不需要 访问 实例属性 或者调用 实例方法
  - 也 不需要 访问 类属性 或者调用 类方法
- 这个时候，可以把这个方法封装成一个 静态方法

语法：

```
class Demo:
    @staticmethod
    def 静态方法名():
        pass
```

注意	静态方法可以和函数一样传入参数使用

- 静态方法需要用修饰器 @staticmethod 来标识，告诉解释器这是一个静态方法
- 通过 类名. 调用静态方法

案例演练

方法综合案例

需求：

1. 设计一个 Game 类

2. 属性：
   - 定义一个 top_score 类属性 -> 记录游戏的历史最高分
   - 定义一个 player_name 实例属性 -> 记录当前游戏的玩家姓名

3. 方法：
   - 静态方法 show_help() -> 显示游戏帮助信息
   - 类方法 show_top_score() -> 显示历史最高分
   - 实例方法 start_game() -> 开始当前玩家的游戏

4. 主程序步骤：
   ① 查看帮助信息
   ② 查看历史最高分
   ③ 创建游戏对象，开始游戏

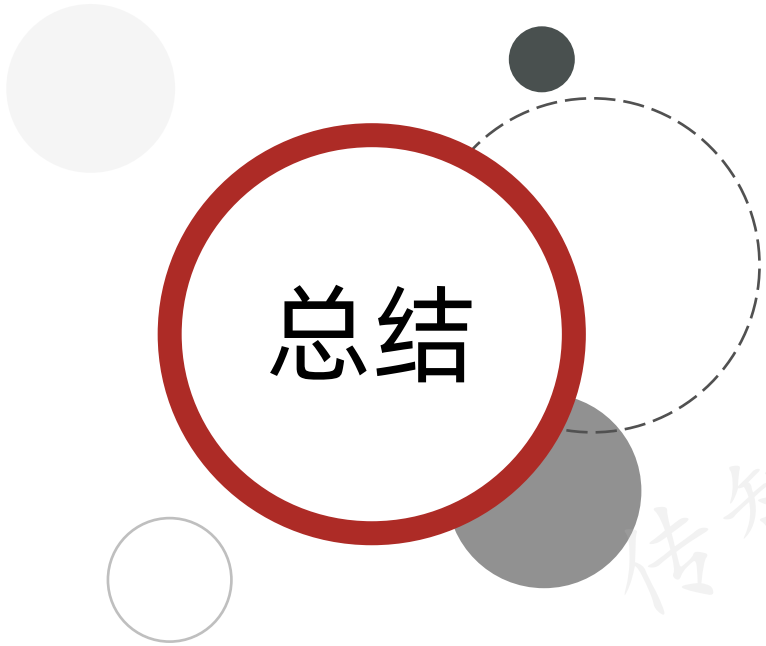
方法综合案例

案例演练

```
class Game(object):

    # 游戏最高分，类属性
    top_score = 0

    @staticmethod
    def show_help():
        print("帮助信息：让僵尸走进房间")

    @classmethod
    def show_top_score(cls):
        print(f"游戏最高分是 {cls.top_score}")

    def __init__(self, player_name):
        self.player_name = player_name

    def start_game(self):
        print(f"[{self.player_name}] 开始游戏...")

        # 使用类名.修改历史最高分
        Game.top_score = 999
```

```
# 1. 查看游戏帮助
Game.show_help()

# 2. 查看游戏最高分
Game.show_top_score()

# 3. 创建游戏对象，开始游戏
game = Game("小明")

game.start_game()

# 4. 游戏结束，查看游戏最高分
Game.show_top_score()
```

1. 类属性和类方法的用法?
2. 私有属性和私有方法的用法?
3. 静态方法的用法?



1. 理解面向对象的基本概念
2. 掌握面向对象的基本语法
3. 掌握面向对象的封装和继承特征

传智教育旗下高端IT教育品牌

黑马程序员-软件测试