15. 核能来袭-初识面向对象

本节主要内容:

1. 面向对象和面向过程
2. 面向对象如何编写
3. 面向对象和面向过程大PK
4. 面向对象三大特征

一. 面向对象和面向过程(重点理解)

1. 面向过程: 一切以事物的流程为核心. 核心是"过程"二字, 过程是指解决问题的步骤, 即, 先干什么, 后干什么. 基于该思想编写程序就好比在编写一套流水线. 是一种机械式的编程思维

   优点: 负责的问题流程化, 编写相对简单
   缺点: 可扩展性差

2. 面向对象: 一切以对象为中心.

什么是对象? 不好解释. 先解释解释什么是车? 有轱辘, 有方向盘, 有发动机, 会跑的是车. 好. 在解释一个. 什么是人. 有名字, 年龄, 爱好, 会唱歌跳舞思考的是人. 我们给这两个东西下了一个简单的定义. 总结: 具有相同属性和动作的结合体叫对象. 面向对象思维, 要自己建立对象. 自己建立场景. 你是就是面向对象世界中的上帝. 你想让车干嘛就干嘛. 你想让人干嘛人就能干嘛.

优点: 可扩展性强
缺点: 编程的复杂度高于面向过程

对比:

说. 要把大象装冰箱, 总共分几步? 三步. 第一步. 打开冰箱门, 第二部. 装大象, 第三部. 关冰箱门. 这是一个典型的面向过程的思维方式. 来我们如果换成面向对象呢? 很简单. 想办法造一个会钻冰箱的大象就可以了. 然后命令大象. 进冰箱. 大象就乖乖的进冰箱了. 这就是面向对象思维. 我们面向的不再是事物发展的流程, 而是操纵某一个事物的个体. 具体的某一个事物.

二. 面向对象如何编写

说了这么多. 面向对象的程序如何编写呢? 想想在我们的世界中. 我们如何造一辆车? 先由设计师来设计图纸. 设计师在图纸上勾勒出车应该是xx样的. 应该有什么. 以及这台车的功能等等. 然后交给工厂进行代工. 根据设计师设计的图纸去创造车. 程序也一样. 我们需要先设计一个图纸. 在图纸上把我要创建的对象进行描述. 然后交给工人去创建对象.

在这里, 我们画图纸的过程需要我们写类, 我们用类来描述一个对象. 类的语法很简单.

```
class 类名:
```

```
    pass
```

哦了, 这就创建出类了. 假设. 我们创建一个车类.

```
class Car:
    pass
```

这就创建了一个类. 图纸有了. 怎么创建一辆车呢? 也很简单. 我们把图纸交给工人帮我们创建一个车的实例. 这个过程被称为实例化. 实例化只需要: "类名()"就可以了

```
c = Car() # 创建一辆车
```

车有了. 我们的车至少得有个颜色, 车牌, 排量等等信息啊. 不同的车, 有不同的颜色, 车牌, 排量等.

```
c.color = "red"
c.pai = "京A66666"
c.pailiang = "1.6T"

print(c.color)
print(c.pai)
print(c.pailiang)
```

我们使用"对象.特征"可以给对象设置属性信息,

接下来, 再造一辆车, 并给车设置相关的属性信息

```
c2 = Car()
c2.color = "white"
c2.pai = "京B22222"
c2.pailiang = "2.0T"

print(c1)
print(c2)
```

我们发现, 这两量车是完全不同的两量车. 但是. 拥有相同的属性和信息. 是不是有点儿冗余了? 怎么办呢? 想想. 我们把车的信息如果写在类里是不是会更好呢? 而且. 我的车在创建的时候这些信息应该已经是设计好了的. 不应该是后天设计的. 好了, 我们知道需求了, 在创建对象的时候能给对象设置一些初始化的属性信息. 在python中我们可以是用__init__(self)函数给对象进行初始化操作. 这个函数(方法)被称为构造函数(方法).

```
class Car:
    def __init__(self, color, pai, pailiang): # self表示当前类的对象. 当前你创建的是谁, 谁来访问的这个方法.那这个self就是谁.
        self.color = color
        self.pai = pai
        self.pailiang = pailiang

c1 = Car("red", "京A66666", "1.6T")

c2 = Car("white", "京B22222", "2,0T")
```

```
print(c1.color)
print(c2.color)
```

通过打印, 我们发现. 这两个对象依然像原来那样可以完成属性的设置.

属性设置完了. 接下来. 车不光有这些信息啊. 车还会跑呢. 跑是一个动作. 所以我们要把跑写成一个函数. 但是在面向对象编程中. 我们不应该叫函数了, 改成叫方法. 只不过这个方法写起来比正常的方法多一个参数self. 仅此而已.

```
class Car:
    def __init__(self, color, pai, pailiang): # self表示当前类的对象. 当前你创建的是谁, 谁来访问的这个方法.那这个self就是谁.
        self.color = color
        self.pai = pai
        self.pailiang = pailiang
    def run(self, speed):
        print("车可以跑%s迈" % speed)

c = Car("red", "京A66666", "2.0T")
c.run(100) # 这时. python会自动把对象c传递给run方法的第一个参数位置.
```

总结: 类与对象的关系: 类是对事物的总结. 抽象的概念. 类用来描述对象. 对象是类的实例化的结果. 对象能执行哪些方法. 都由类来决定. 类中定义了什么. 对象就拥有什么

练习题:

1. 创建一个武松. 武松可以打老虎, 杀嫂子, 替天行道
2. 用面向对象的思维来模拟LOL里的盖伦上阵杀敌.
3. 编写和尚类. 自己自由发挥和尚有哪些属性和方法.
4. 用面向对象的思维来完成用户登录.

三. 面向对象和面向过程大PK

那么面向对象和面向过程到底哪个好? 具体问题. 具体分析. 没有绝对的好和不好. 这一点要格外注意.

来. 我们来完成之前的装大象的程序:

1. 面向过程:

```
非函数版:
    print("开冰箱门")
    print("装大象")
    print("关冰箱门")


函数版:
```

```
def open_door():
    print("开冰箱门")
def zhuang():
    print("装大象")
def close_door():
    print("关冰箱门")

open_door()
zhuang()
close_door()
```

2. 面向对象:

```
class Elephant:

    def open(self):
        print("打开冰箱门")

    def close(self):
        print("关冰箱门")

    def zhuang(self):
        print("把自己装进去")


dx = Elephant()
dx.open()
dx.close()
dx.zhuang()
```

发现了吧, 面向对象简直麻烦到爆. 别着急. 接着看下一个案例.

小猪佩奇大战奥特曼. 说. 有一个小猪, 名叫佩奇, 今年40岁了. 会使用嘴巴嘟嘟必杀技. 他不光大战奥特曼, 还会大战蝙蝠侠, 蜘蛛侠

1. 面向过程:

```
def da_ao_te_man(name, age, jn):
    print("%s, 今年%s岁了, 使用%s技能疯狂输出奥特曼" % (name, age, jn))

def da_bian_fu_xia(name, age, jn):
    print("%s, 今年%s岁了, 使用%s技能疯狂输出蝙蝠侠" % (name, age, jn))

def da_zhi_zhu_xia(name, age, jn):
    print("%s, 今年%s岁了, 使用%s技能疯狂输出蜘蛛侠" % (name, age, jn))


da_ao_te_man("小猪佩奇", 39, "嘴巴嘟嘟")
```

```
da_bian_fu_xia("小猪佩奇", 39, "嘴巴嘟嘟")
da_zhi_zhu_xia("小猪佩奇", 39, "嘴巴嘟嘟")
```

2. 面向对象

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

    def da_ao_te_man(self):
        print("%s, 今年%s岁了, 使用%s技能疯狂输出奥特曼" % (self.name, self.age,
self.jn))

    def da_bian_fu_xia(self):
        print("%s, 今年%s岁了, 使用%s技能疯狂输出蝙蝠侠" % (self.name, self.age,
self.jn))

    def da_zhi_zhu_xia(self):
        print("%s, 今年%s岁了, 使用%s技能疯狂输出蜘蛛侠" % (self.name, self.age,
self.jn))


peiqi = Pig("小猪佩奇", 39, "嘴巴嘟嘟")
peiqi.da_ao_te_man()
peiqi.da_bian_fu_xia()
peiqi.da_zhi_zhu_xia()
```

感觉到一点儿眉目了吧. 在这个案例中, 明显面向对象的思想更加清晰一些. 代码也更容易编写一些. 所以. 用哪种编程思想不是绝对的. 得根据需求和需要来完成.

四. 面向对象的三大特征

面向对象三大特征: 封装, 继承, 多态. 只要是面向对象编程语言. 都有这三个特征.

1. 封装: 把很多数据封装到一个对象中. 把固定功能的代码封装到一个代码块, 函数, 对象, 打包成模块. 这都属于封装的思想. 具体的情况具体分析. 比如. 你写了一个很牛B的函数. 那这个也可以被称为封装. 在面向对象思想中. 是把一些看似无关紧要的内容组合到一起统一进行存储和使用. 这就是封装.
2. 继承: 子类可以自动拥有父类中除了私有属性外的其他所有内容. 说白了, 儿子可以随便用爹的东西. 但是朋友们, 一定要认清楚一个事情. 必须先有爹, 后有儿子. 顺序不能乱, 在python中实现继承非常简单. 在声明类的时候, 在类名后面添加一个小括号, 就可以完成继承关系. 那么什么情况可以使用继承呢? 单纯的从代码层面上来看. 两个类具有相同的功能或者特征的时候. 可以采用继承的形式. 提取一个父类, 这个父类中编写着两个类相同的部分. 然后两个类分别取继承这个类就可以了. 这样写的好处是我们可以避免写很多重复的功能和代码. 如果从语义中去分析的话. 会简单很多. 如

果语境中出现了x是一种y. 这时, y是一种泛化的概念. x比y更加具体. 那这时x就是y的子类. 比如. 猫是一种动物. 猫继承动物. 动物能动. 猫也能动. 这时猫在创建的时候就有了动物的"动"这个属性. 再比如，白骨精是一个妖怪. 妖怪天生就有一个比较不好的功能叫"吃人"，白骨精一出生就知道如何"吃人". 此时 白骨精继承妖精. 话不多说. 上代码.

```
class Yao:

    def chi(self):
        print("我是妖怪，我天生就会吃人")


class BaiGuJing(Yao):   # 白骨精继承妖怪
    pass


bgj = BaiGuJing()
bgj.chi()   # 我是妖怪，我天生就会吃人  # 虽然白骨精类中没有编写chi. 但是他爹有啊. 直接拿来用
```

在python中, 一个类可以同时继承多个父类. 说白了, 现在一个儿子可能会有多个爹了. 既然是有这么多个爹，总得有远有近. 比如. 有一个这样的牛B的人物, 叫锅不美. 就有很多个爹嘛.

```
class QinDie:
    def chi(self):
        print("亲爹给你好吃的")

    def play(self):
        print("亲爹会陪你玩")


class GanDie:
    def qian(self):
        print("干爹给钱啊")
    def play(self):
        print("干爹会陪你玩")


class GuNiang(QinDie, GanDie):
    pass

bumei = GuNiang()
bumei.chi()     # 亲爹
bumei.qian()    # 亲爹没有，找干爹
bumei.play()    # 亲爹有了，就不找干爹了
```

具体的MRO(method resolution order)算法. 我们到后面会具体进行分析和讲解.

3. 多态: 同一个对象, 多种形态. 这个在python中其实是很不容易说明白的. 因为我们一直在用. 只是没有具体的说. 比如. 我们创建一个变量a = 10 , 我们知道此时a是整数类型. 但是我们可以通过程序让a = "alex", 这时, a又变成了字符串类型. 这是我们都知道的. 但是, 我要告诉你的是. 这个就是多态性. 同一个变量a可以是多种形态. 可能这样的程序和说法你还get不到具体什么是多态. 接下来. 我们来看一个程序. 北京动物园饲养员alex一天的工作. 从早上开始喂养猪, 中午喂哈士奇, 晚上还得喂阿拉斯加. 来我们用代码实现这样的代码:

```
class Animal:
    def chi(self):
        print("动物就知道吃")

class Pig(Animal):
    def chi(self):
        print("猪在吃")

class Haski(Animal):
    def chi(self):
        print("哈士奇在吃")

class Alasika(Animal):
    def chi(self):
        print("阿拉斯加在吃")

class SiYangYuan:
    def yanng_animal(self, ani):
        ani.chi()

zhu = Pig()
erha = Haski()
ala = Alasika()

alex = SiYangYuan()
alex.yanng_animal(zhu)
alex.yanng_animal(erha)
alex.yanng_animal(ala)
```

多态的好处: 程序具有超高的可扩展性. 面向对象思想的核心与灵魂. python自带多态. 66666