day9面向对象基础

1.面向对象编程介绍

day1 -- day4 地基一定到打好

1.1面向过程的编程思想

```
'''
day1 -- day4 地基一定到打好
面向过程的编程思想
    核心过程二字,过程指的是解决问题的步骤,即先干什么\再干什么\后干什么
    基于该思想编写程序就好比在设计一条流水线,是一种机械式的思维方式
    优点:复杂的问题流程化\进而简单化 （一步步去思考，写一步走一步）
    缺点:可扩展性差
        （加一点参数或者方法可能改掉原来的代码，
    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。）
    就像打工自己需要技术，学历，工作经验等等
    老板或者领导把事情安排好就ok了，员工只需要照着去做，能赚钱就OK

'''
```

1.2.面向对象的编程思想

```
'''
面向对象的编程思想
    核心是对象二个字,对象是特征与技能的结合体
    基于该思想编写程序(脑子始终想的是对象二字)就好比在创造一个世界,世界是由
    一个个对象组成,在上帝眼里任何存在的事物都是对象,任何不存在的事物也都可以创造
    出来,是一种上帝式的思维方式
    优点：可扩展性强（比如:加属性加方法，有框架体系
    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：编程的复杂度要高与面向过程
    (难度高，要看应用场景
    面向对象的程序设计看起来高大上，所以我在编程时就应该保证通篇类，
    这样写出的程序一定是好的程序（面向对象只适合那些可扩展性要求比较高的场景）
    比如，假设大海老师会降龙十八掌的十八掌，那么我每次跟人干仗都要从第一掌打到
    第18掌降龙有悔
    这才显得我很厉害:
    面对敌人，我打到第三掌对方就已经倒下了，我说，不行，你给我起来，我还没有show完..
    3.类有类属性，实例/对象有实例/对象属性，所以我们在定义class时一定要定义出那么几个类属性，
    想不到怎么办，那就使劲的想，定义的越多越牛逼
    这就犯了一个严重的错误，程序越早面向对象，死的越早，为啥面向对象，
        因为我们要将数据与功能结合到一起，程序整体的结构都没有出来，
        或者说需要考虑的问题你都没有搞清楚个八九不离十，你就开始面向对象了，
```

```
        这就导致了，你在那里干想，自以为想通了，定义了一堆属性，结果后来又都用不到，
        或者想不通到底应该定义啥，那就一直想吧，想着想着就疯了。
    就像创业：
        自己不太需要技术能力，只需要懂那些人拥有这些能力，因为我有钱
        自己需要规划好创业的每一步，
        招聘的时候人才的选择
        技术
        销售
        门面的选择
类：
    对象是特征与技能的结合体，而类则是一系列对象相同的特征与技能的结合体
    强调：
        1.对象是具体存在的事物，而类则一个抽象的概念
        2.站在不同的角度总结出的类与对象是不同的
    在现实世界中：先有一个个具体存在的对象，然后随着人类文明的发展才总结出类的概念
'''
```

2.类的使用

2.1.类定义的语法

```
# @Author : 大海
# @File : 2.类的使用.py
'''
类
    老师
        属性
            name='大海'
            age=18
            sex='男'
        技能/方法/函数
            上课
class 定义类的关键字      类名首字母大写（约定俗成的）
class 类名:
    pass
复杂的类名和变量一样
# 驼峰体
AgeOfDahai = 18
定义示例：
class Person:
    pass
'''
```

2.2.类定义实例

```
#1. 先定义类
class Teacher:
    # 相同的特征/属性/变量
    name = '大海'
    age = 18
```

```
    sex = '男'
    # 函数/方法/技能
    def course(self):
        # self到底是什么?
        # self当做一个位置形参
        print(self)
        print('course')
        return 222
    # print('类定义是运行的')
```

2.3.类的属性和方法

2.3.1.概念理解

```
# 类是一系列对象相同的属性(变量)与技能(函数)的结合体,
# 即类体中最常见的就是变量与函数的定义
# 类体代码会在类定义阶段立即执行,会产生一个类的名称空间,
```

2.3.2.调用类的属性

```
print(Teacher.__dict__)
name = 'aaa'
print(name)
# # 调用类的属性
print(Teacher.name)
print(Teacher.age)
print(Teacher.sex)
```

2.3.3.类调用类中带self参数的方法

```
# 类的方法其实就是函数
print(Teacher.course)

# 函数加括号调用
# ? ? ? ? ?
# self我们现在把它当作一个位置形参，但是为什么定义成self呢?
Teacher.course(111)
# 也有返回值
aaa=Teacher.course(111)
print(aaa)

# 不存在的属型或者方法会报错

print(Teacher.xxx)
```

2.3.4.修改类属性的值

```
# 修改类属性的值
print(Teacher.name)
#
Teacher.name = '夏洛'
#
print(Teacher.name)
```

2.3.5.添加类的属性

```
# 添加类的属性

Teacher.play = '篮球'
print(Teacher.play)
print(Teacher.__dict__)
```

2.3.6.删除类的属性

```
# 删除类的属性
del  Teacher.play
#
print(Teacher.play)
print(Teacher.__dict__)
```

2.3.7.总结

```
# 总结:
# 1. 类本质是一个用来存放变量与函数的容器
# 2. 类的用途之一就是当做容器从其内部取出名字(变量与函数名)来使用
# 3. 类的用途之二是调用类来产生对象  接下来讲对象
```

3.对象的使用

3.1.对象的调用和生成

```
# @Author : 大海
# @File : 3.对象的使用.py

#1. 先定义类
class Teacher:
    # 相同的特征/属性/变量
    name = '大海'
    age = 18
    sex= '男'
    # 函数/方法/技能
    def course(self):
        # self到底是什么?
        # self当做一个位置形参
```

```
        print(self)
        print('course')
        return 222
#2. 后调用类来产生对象:
# 调用类的过程称之为类的实例的初始化,调用类的返回值称之为类的一个对象/实例
# 调用类发生了?
# 类是抽象  对象/实例是具象

# 产生3个老师对象
t1=Teacher()
t2=Teacher()
t3=Teacher()
# print(t1)
# print(id(t1))
# print(id(t2))
# print(id(t3))
```

3.2.对象的属性和方法

```
# 对象独有的属性(没有独有的，从类里面去要)
# print(t1.__dict__)
#
# print(t1.name)
# print(t2.name)
# print(t3.name)

# 对象的方法(没有独有的，从类里面去要)
# print(t1.course)
# print(t2.course)
# print(t3.course)
#
# print(Teacher.__dict__)
# print('============')
# print(t1.__dict__)
# print(t2.__dict__)
# print(t3.__dict__)

# 类有return
# aaa=Teacher.course(111)
# print(aaa)

print('--------------')
# 对象的方法
# ?????
#对象的方法执行，没有传入参数，self到底是什么?
# 对比类方法执行,需要传入参数
# print(t1)
# a1=t1.course()
# print(a1)

# 对象属性的修改
```

```
# （类里面的属性）
# print(t1.name)
# print(t1.__dict__)
# # 添加了对象独有的属性
# t1.name = '夏洛'
# # print(t1.name)
# # print(t1.__dict__)
# # 删除对象的属性
# # 删除的是   t1.name = '夏洛'
# # del t1.name
# print(t1.name)
# # 对象完全独立 一个对象的操作不会对另一个对象影响
# print(t2.name)
```

4.生成对象自动添加属性方法

```
# @Author : 大海
# @File : 4.生成对象__init__方法.py
'''
### 1、初始化 *****
以双下划线开头且以双下划线结尾的固定方法，他们会在特定的时机被触发执行，
__init__就是其中之一，它会在实例化之后自动被调用，以完成实例的初始化。
'''
#1. 先定义类
class Teacher:
    # 相同的特征/属性/变量
    school = 'tuling'
    # 这个不对,不应该定义成类属性
    # name = '大海'
    # age = 18
    # sex= '男'
    def __init__(self,name,age,sex):
        self.name = name
        self.age = age
        self.sex = sex

    # 函数/方法/技能
    def course(self,name):
        # self到底是什么?
        # self当做一个位置形参
        print(self)
        print('course')
        print(name)
        print('--------------------')
        print('我是大海的属性%s'%self.name)
        print('我是大海的属性%s'%self.age)
        return '11111'
```

4.1.没有独有的对象属性(没有init方法)

```
# 产生3个老师对象
# __init__没有就没有独有的对象属性
t1=Teacher()
print(t1.__dict__)
t1.name = '夏洛'
print(t1.name)
print(t1.__dict__)
```

4.2.有独有的对象属性(有init方法)

```
# 有__init__方法,有了对象独有的属性
dahai=Teacher('大海',18,'男')
xialuo=Teacher('夏洛',22,'男')
xishi=Teacher('西施',18,'女')
# 对象的独立属性
print(dahai.__dict__)
# 类的属性 对象调用和类调用一样
print(dahai.school)
print(id(dahai.school))
print(Teacher.school)
print(id(Teacher.school))

print(dahai)
# # 对象独立的属性
print(dahai.__dict__)
# # 对象独立的属性
print(xialuo.__dict__)
# # __init__方法传入的属性类是没有的
print(Teacher.__dict__)

# 只是初始化的时候传入了参数，对象还是可以改属性值
print(dahai.name)
dahai.name = '顾安'
print(dahai.__dict__)

a1=dahai.course('wwwww')
print(a1)
```

5.类的属性和对象属性的关系

定义类

```
# @Author : 大海
# @File : 5.类的属性和对象属性的关系.py
#1. 先定义类
class Teacher:
    # 相同的特征/属性/变量
```

```
    school = 'tuling'
    xxx = '我是类的属性,也可能是对象的属性'
    yyy = 111
    def __init__(self,name,age,sex):
        self.name = name
        self.age = age
        self.sex = sex

    # 函数/方法/技能
    def course(self):
        # self到底是什么?
        # self当做一个位置形参
        print(self)
        # print('course')
        #
        # print('--------------------')
        # print('我是大海的属性%s'%self.name)
        # print('我是大海的属性%s'%self.age)
        # return '11111'
```

5.1.类和对象的属性的优先级

```
# 实例初始化的时候必须传入参数，生成了对象的属性
dahai = Teacher('大海', 18, '男')
xialuo = Teacher('夏洛', 16, '男')
xishi = Teacher('西施', 18, '女')
# 对象属性的查找
# 添加一个对象属性
dahai.xxx= '我是对象的属性'
# 属性查找优先找对象，对象没有才去类里面找
print(dahai.xxx)
```

5.2.类和对象的属性共享关系

```
# 类中定义的属性和方法是所有对象共享的,类可以用，对象也可以用
# 类的属性给对象调用(原装)
print(id(dahai.yyy),dahai.yyy)
print(id(xialuo.yyy),xialuo.yyy)
print(id(xishi.yyy),xishi.yyy)
# # # 类自己用
print(id(Teacher.yyy),Teacher.yyy)
# 对象小气
print(Teacher.name)
```

5.3.类的属性变化对类和对象的影响

```
# 类的属性变化
# 类改类的属性
Teacher.yyy = 333
# # 对象改类的属性（改不了）（实际上是自己添加了一个独有的属性）
dahai.yyy = 2222
# # # 类的属性给对象调用
# # # # # 对象已经有了yyy属性的会优先考虑自己的
print(id(dahai.yyy),dahai.yyy)
print(id(xialuo.yyy),xialuo.yyy)
print(id(xishi.yyy),xishi.yyy)
# # # 类自己用
print(id(Teacher.yyy),Teacher.yyy)
```

5.4.对象调用类中带self参数的方法

```
# ？？？？？？
# 对象调用类的方法,不需要传入参数?

print(id(dahai.course))
print(dahai)
dahai.course()

print(id(xialuo.course))
print(xialuo)
xialuo.course()

print(id(xishi.course))
print(xishi)
xishi.course()
print('===============')
# 类调用方法必须传入self参数对应的对象,因为方法里面需要使用这个self对象
Teacher.course(dahai)
Teacher.course(xialuo)
Teacher.course(xishi)
```

6.绑定方法和非绑定方法

实例

```
# @Author : 大海
# @File : 6.绑定方法和非绑定方法.py
# 类和对象的方法到底是怎样的
# 我们一起来探究一下真相
# self绑定给对象方法
# 简单来说就是对象在调用方法的时候会自动把该对象传入
# 该方法的参数一般这个参数我们用self表示
# self绑定给对象方法
class A:
    # 作者@selfmethed省略了
```

```
    # 因为类里面的方法大部分情况下是给实例化后的对象用
    def f(self,n):
        print(self,n)
        print('我是self的方法')
#
a = A()
a1 = A()
print(a)
print('=================')
# # 是哪个对象调用的self参数的方法，那么传入的self就是那个对象
a.f(1)
print(a1)
print('-----------------')
a1.f(1)
# # 类调用必须传入对象 不会自动传入对象 自己手动传入对象
print('*****************')
A.f(a,1)
A.f(a1,2)

# 绑定类的方法是给类用的

class B:
    @classmethod
    def f(cls,n):# cls是一个规范，代表是绑定类的方法
        print(cls,n)
        print('我是绑定类的方法')
print(B)
print('============')
B.f(1)
# # 一般不这样用，绑定给类的方法给对象用
b  = B()
print(b)
b.f(1)


# 非绑定方法/静态方法
class C:
    @staticmethod
    def f(n):
        print(n)
        print('我是非绑定方法/静态方法')
# # # 纯粹的函数，不会自动传入类
C.f(2)
c = C()
# # # 纯粹的函数，不会自动传入对象
c.f(3)
```

总结

```
'''
1、绑定方法（精髓在于自动传值）
```

```
特性：绑定给谁就应该由谁来调用，谁来调用就会将谁当作第一个参数自动传入
    绑定方法分为两类：
        1.1 绑定给对象方法
            在类内部定义的函数（没有被任何装饰器修饰的），默认就是绑定给对象用的
        1.2 绑定给类的方法：
            在类内部定义的函数如果被装饰器 @classmethod 装饰，
            那么则是绑定给类的，应该由类来调用，类来调用就自动将类当作第一个参数自动传入
2、非绑定方法（不会自动传值，就是一个 普通函数）
    类中定义的函数如果被装饰器 @staticmethod 装饰，那么该函数就变成非绑定方法
    优点
        既不与类绑定，又不与对象绑定，意味着类与对象都可以来调用
    缺点
        但是无论谁来调用，都没有任何自动传值的效果，就是一个普通函数
3 作用
    如果函数体代码需要用外部传入的类，则应该将该函数定义成绑定给类的方法
    如果函数体代码需要用外部传入的对象，则应该将该函数定义成绑定给对象的方法
    如果函数体代码既不需要外部传入的类也不需要外部传入的对象，则应该将该函数定义成非绑定方法/普通函数
'''
```

7.一切皆对象

```
# @Author : 大海
# @File : 7.一切皆对象.py
# 在python中统一了类与类型的概念
# class Foo:
#     def find(self):
#         print('我是绑定对象的方法')
# # <class '__main__.Foo'>
# print(Foo)
# obj = Foo()
# # <__main__.Foo object at 0x000001F7B83BEBC8>
# print(obj)
# obj.find()
# # obj的类型是Foo , obj的类也是Foo
# print(type(obj))
# # ctrl 按住不动 点击鼠标左键
# print(str)
# # 其实python在'dahai'的前面自动加了一个str类
# name = str('dahai')
# # print(name)
# # # 面向对象的理解：str这个类实例化生成了name这个对象
# # # dahai这个值赋值给name这个变量
# # print(type(name))
# # # 面向对象的理解：name对象的方法
# # # 数据类型的方法
# print(name.startswith('d'))
#
# obj.find()
```