**day12笔记**

1.面向对象的三大特性：封装性、继承性、多态性

**封装**：

封装使用的领悟：

1).生活层面：食品、快递、计算机、明星...

2).计算机层面：

①.模块、类、函数...

②.属性数据的封装与隐藏 （数据私有化）重点

权限修饰符的概念：

public(公共的，范围最大) protected(收保护的) default(默认，缺省) private(私有的，范围最小) python语言没有以上这些关键字：

好处：安全性提高了

**#演示python中的封装性的使用：  
class** Person:  
 **def** \_\_init\_\_(self,name,age,money):  
 self.name = name  
 self.age = age  
 self.money = money  
 **def** \_\_str\_\_(self):  
 **return "name：%s，age：%s，money：%s"**%(self.name,self.age

,self.money)  
*#实例化Person对象*p = Person(**'tom'**,30,10000)  
print(p) *#name：tom，age：30，money：10000*  
*#以下代码可以设置值，但是不符逻辑了*

p.age=-40

print(p) *#name：tom，age：-40，money：10000*

**以上情况不会出现编译和运行异常，但是出现了数据不符合逻辑的情况；  
关系到对象直接在外部去操作数据(属性)，导致"脏数据"的出现；  
所以我们该如何去解决此问题呢？  
1).首先第一步是在外界不允许对象直接操作/访问属性(将此权利没收) --> 将属性私有化：\_\_属性名  
2).需要在类的内部提供给外界额外的访问方式(函数:getter/setter)**

*#所以需要将上面的年龄age私有化，一旦私有化age之后，那么age使用的方位只有在class中，出了class外界无法使用他，使用\_\_属性名的方式*

**class** Person:  
 **def** \_\_init\_\_(self,name,age,money):  
 self.name = name  
 self.**\_\_age** = age  
 self.money = money  
 **def** \_\_str\_\_(self):  
 **return "name：%s，age：%s，money：%s"**%(self.name,self.**\_\_age**

,self.money)  
*#实例化Person对象*p = Person(**'tom'**,30,10000)  
print(p) *#name：tom，age：30，money：10000*

p.age=-40 *#现在此行代码相当于动态为对象p添加一个属性age*

print(p) *#name：tom，age：30，money：10000*

*#查看对象p中所有的成员变量（属性）*

print(p.\_\_dict\_\_)

*#得到{'name': 'tom',* ***'\_Person\_\_age'****: 30, 'money': 10000}*

*所以age私有化之后，在计算机底层真正的名字已经变成了****\_Person\_\_age***

**以下的操作仅仅是为对象p动态添加一个属性为\_\_age，  
一个属性一旦被私有化，在底层真正的名字是：\_类名\_\_属性名**

**其实python的私有化我们可以理解为伪私有(只是换了个名)，  
【注意】我们就当它是私有，不要随意去操作它中的内容，即以下的操作("帅"的人都这么干)**

p.\_\_age=-50

print(p) *#仍然得到name：tom，age：30，money：10000*

print(p.\_\_dict\_\_)

*#再看属性{'name': 'tom', '\_Person\_\_age': 30, 'money': 10000, '\_\_age': -50} 只是多了一个名为\_\_age的参数*

*但是动态数据还是可以该的（但是不要去该，这样私有化就没意义了）*

p.\_Person\_\_age = -100  
print(p) *#得到name：tom，age：-100，money：10000*

**私有化之后可以不会出现逻辑不符的现象，但是对于age，需要在类的内部提供给外界额外的访问方式(函数:getter/setter)**

**格式：get属性名（self)-->有返回值**

**set属性名（self，变量参数)-->有返回值**

**以上两个函数的属性名都满足首字母大写其余字母小写的规范**

**class** Person:  
 **def** \_\_init\_\_(self,name,age,money):  
 self.name = name  
 self.\_\_age = age  
 self.money = money  
 *#设置\_\_age* **def** setAge(self,age):  
 *#对age值进行合法性的校验* **if** age < 0 **or** age > 130:  
 **raise** Exception(**'年龄不合法...'**)  
 **else**:  
 self.\_\_age = age  
 *#获取\_\_age* **def** getAge(self):  
 **return** self.\_\_age  
 **def** \_\_str\_\_(self):  
 **return "name：%s，age：%s，money：%s"**%(self.name,self.\_\_age,

self.money)

p = Person(**'tom'**,30,10000)  
print(p) *#name：tom，age：30，money：10000*  
**调用函数完成设置和获取属性值的操作：**  
print(p.getAge())  *#30*p.setAge(40)  
print(p)  *#name：tom，age：40，money：10000*

p.setAge(-40)  
print(p) *#raise Exception('年龄不合法...')*

*Exception: 年龄不合法...*

*将money私有化，就不设合法性判断了*

**def** \_\_init\_\_(self,name,age,money):  
 。。。

。。。

self.**\_\_money** = money

**def** setMoney(self,money):  
 self.\_\_money = money  
**def** getMoney(self):  
 **return** self.\_\_money

**def** \_\_str\_\_(self):  
 **return "name：%s，age：%s，money：%s"**%(self.name,self.\_\_age,

self.\_\_money)

对于python的属性私有化使用:\_\_来实现

在设计完类，外界创建对象通过.的形式访问(设置)属性，

可能会出现跟现实情况不符的混乱数据，那么我们就将属性的设置权没收(外界不能直接通过.调用属性)，

在类的内部提供外界额外的访问方式(定义setter和getter方法)，

并且在需要的时候，可以在函数的内部加入数据合法性的校验；

模板：

对于setter函数，命名：set属性名(首字母大写)

对于getter函数，命名：get属性名(首字母大写)

私有属性：\_\_age

设置值(\_\_age)：

def setAge(self,age):

self.\_\_age = age

获取值(\_\_age)：

def getAge(self):

return self.\_\_age

1. **继承性**

使用场景：

1).生活层面：继承皇位，子承父业，继承宗教信仰

2).计算机层面：两部分组成，一部分我们称为父类(基类、超类、superclass)

另一部分我们称为子类(派生类、subclass)，

子类可以使用父类中的成员(使用权)

继承性的好处：

1).代码复用性变强

2).代码扩展性变强

3).代码维护性变好

4).代码阅读性变好

继承性弊端：

类和类之间是一种强耦合关系

继承的好处要远远多于弊端，所以我们还是要经常使用继承的(合理)，切记不能为了继承而继承

分析：

继承体系可以很庞大(呈现树状结构图)，

越往上层的类，感觉越模糊，越不清晰

越往下层的类，感觉越清晰，越具体

所以得出结论，在之后的开发过程中创建父类的可能性变低，子类实例化的可能性极高

分类：

1).单继承(单一继承)

2).多重继承

3).多继承(很多语言是不合法的)

注意事项：

1).由于继承的特点，子类对象被实例化，但是可能需要为父类属性赋值，那么可以在子类的构造函数中

显示的调用父类构造来实现

1. .记住：虽然父类构造被执行，但是它仅仅做的就是赋值这件事，内存中的对象只有子类对象一个

**演示单继承的使用：** *#父类***class** Person:  
 **def** \_\_init\_\_(self,name,age):  
 print(**'我是Person类的构造函数。。。'**)  
 self.name = name  
 self.age = age  
 *#吃* **def** eat(self):  
 print(**'吃一个...'**)  
 *#睡* **def** sleep(self):  
 print(**'睡一个...'**)  
*#子类***class** Teacher(Person):  
 **def** \_\_init\_\_(self,name,age,salary):  
 print(**'我是teacher类的构造函数。。。'**)  
 self.salary = salary  
 *#调用父类构造函数 目的：父类的属性由父类自己赋值  
 #在子类构造函数中显示的调用其父类构造  
 #1 super(Teacher,self).\_\_init\_\_(name,age)  
 #2 super().\_\_init\_\_(name,age)* Person.\_\_init\_\_(self,name,age) *#3调用父类，这种方式最好*  
 *#教学* **def** teach(self):  
 print(**'教书育人...'**)  
  
*#实例化子类对象*t = Teacher(**'老郭'**,30,6000.0)  
print(t.name,t.age,t.salary)  
*#调用函数*  
t.eat()  
t.sleep()  
t.teach()

*#结果得到： 我是teacher类的构造函数。。。*

*我是Person类的构造函数。。。*

*老郭 30 6000.0*

*吃一个...*

*睡一个...*

*教书育人...*

**演示多重继承的使用：***#定义生物类：***class** Creature:  
 **def** \_\_init\_\_(self,age):  
 self.age = age  
  
 **def** breath(self):  
 print(**'吸一个...'**)  
*#定义动物类：***class** Animal(Creature):  
 **def** \_\_init\_\_(self,age,name):  
 self.name = name  
 super().\_\_init\_\_(age)  
  
 **def** eat(self):  
 print(**'吃一个...'**)  
*#定义狗类***class** Dog(Animal):  
 **def** \_\_init\_\_(self,age,name,color):  
 self.color = color  
 super(Dog,self).\_\_init\_\_(age,name)  
 *#以下两种方法也可以*

*2. super().\_\_init\_\_(age,name)   
 3. Animal.\_\_init\_\_(self,age,name)*

**def** wangwang(self):  
 print(**'狗汪汪叫...'**)  
  
 **def** \_\_str\_\_(self):  
 **return "name：%s，age：%s，color：%s"** %(self.name,self.age,

self.color)  
  
d = Dog(3,**'小宝'**,**'brown'**)  
print(d) *#因为定义了\_\_str\_\_,即重写，所以可以直接看到属性，不然*

*print(d)得到的是打印d的地址而已，*

*只有print(t.name,t.age,t.color),才能看到结果*d.wangwang()  
d.eat()  
d.breath()

*#得到 name：小宝，age：3，color：brown*

*狗汪汪叫...*

*吃一个...*

*吸一个...*

**演示多继承的使用：**

就是一个子类可以调用多个父类，下面演示的是一个子类Child可以调用两个父类Father和Mather类

*#定义Father类***class** Father:  
 **def** \_\_init\_\_(self,money):  
 self.money = money  
  
 **def** drinking(self):  
 print(**'走一个...'**)

*#定义Mother类***class** Mother:  
 **def** \_\_init\_\_(self,faceValue):  
 self.faceValue = faceValue  
 **def** shopping(self):  
 print(**'买买买...'**)

*#定义Child类同时继承Father和Mother类* **class** Child(Father,Mother):  
 **def** \_\_init\_\_(self,money,faceValue,work):  
 self.work = work  
 Father.\_\_init\_\_(self,money)  
 Mother.\_\_init\_\_(self,faceValue)  
  
 **def** playing(self):  
 print(**'浪一个...'**)

*#实例化子类对象*child = Child(1000000,**True**,**"语数外"**)

*#调用函数*  
child.playing() *#浪一个...*  
child.drinking() *#走一个...*  
child.shopping() *#买买买...*

3.**函数重写(复写，覆盖，override)** 【重点】

前提：必须有继承性

原因：父类中的功能(函数)，子类需要用，但是父类中函数的函数体内容和我现在要执行的逻辑还不相符，那么可以将函数名保留(功能还是此功能)，但是将函数体重构；

注意：子类重写父类的函数，除了函数体以外的部分，直接复制父类的即可

*演示函数重写的使用：***class** Fu:  
 **def** func(self):  
 print(**'辟邪剑法...'**)  
  
**class** Zi(Fu):  
 **def** func(self):  
 super().func()  
 print(**'葵花宝典...'**)

*#实例化子类对象*zi = Zi()

zi.func() *#得到 辟邪剑法*

*葵花宝典*

*演示OldPhone和NewPhone的关系...***class** OldPhone:  
 *#打电话* **def** call(self,name,msg):  
 print(**'正在给%s打电话，内容为：%s'** %(name,msg))  
 *#发短信* **def** sendMessage(self,name,msg):  
 print(**'正在给%s发短信，内容为：%s'** %(name,msg))  
  
**class** NewPhone(OldPhone): ##########  
 *#打电话* **def** call(self,name,msg):  
 print(**'显示归属地和大头贴...'**)  
 super().call(name,msg) ###########  
 *#发短信* **def** sendMessage(self,name,msg):  
 super().sendMessage(name,msg) ###########  
 print(**'发送表情包...'**)  
*#实例化子类对象*np = NewPhone()  
*#执行函数*  
np.call(**'苍老师'**,**'约吗？'**)  
np.sendMessage(**'小泽老师'**,**'hi...'**)

*#得到 显示归属地和大头贴...*

*正在给苍老师打电话，内容为：约吗？*

*正在给小泽老师发短信，内容为：hi...*

*发送表情包...*

1. **类属性和对象属性**

类属性：定义在class内，函数外

对象属性：定义在构造函数的内部

*演示类属性和对象属性的使用区别？*

**class** Demo(object):  
 *#定义类属性a赋值为10* a = 10  
 *#定义对象属性b* **def** \_\_init\_\_(self,num):  
 self.b = num *#self.b是对象属性，num是变量*

*#实例化Demo对象*  
d=Demo(20) *#因为有构造函数，所以实例化函数必须传参*   
  
print(d.b) *#20 对像调对象属性*print(d.a) *#10 对象调类属*  
print(Demo.a) *#10 类调类属性*  
print(Demo.b) *#报错 类不能调对象属性*

**得到结论对象既可以调用对象属性，也可以调用(获取)类属性  
如果在一个类中出现同名的类属性和对象属性，那么对象优先调用的是对象属**

**性**

参看下面代码：

**class** Demo(object):  
 b = 10 *#定义类属性b赋值为10*  
 **def** \_\_init\_\_(self,num):  
 self.b = num

d=Demo(20)   
print(d.b) *#得到的是对象属性 20*

*生命周期： 类：早加载到内存(方法区)  
对象：晚加载到内存(堆)*

*总结：对象可以调用对象以及类的、类能调用类的不能调用对象的*

*晚的可以调用早的，早的不能调用晚的 相同周期的都能调用*

**class** Demo(object): b = 10  
d=Demo（）

print(d.b) *#10*

d.b=30

print(d.b)  *#30*

print(d.\_\_dict\_\_) *#{'b': 20} 此行代码的作用就是专门用来看对象的属性和属性值的*

**以上的操作：并不是真正去修改类属性b的值为30，  
而是动态为对象d添加一个对象属性b赋值为30**

*#接上面的代码，下面的一行才是调用类属性*

print(Demo.b) *#得到10* **类可以调用类属性，不能调用对象属性**

d = Demo(40) #这里的40其实是构造函数里的b,是对象属性  
print(Demo.b) ########会报错 因为类不能调用对象属性

*#***换一种方法再理解以下\*\*\*\*\***

**class** Demo(object):  
 *#定义类属性a赋值为10* b= 10  
 *#定义对象属性b* **def** \_\_init\_\_(self,num):  
 self.b = num  
d=Demo(20)  *#在我看来构造函数，就相当于上面的d.b=30，即动态添加属性*

print(d.b)  *#得到20*print(Demo.b)  *#得到10*

*【注意事项】：  
1).对象可以调用对象属性、类属性  
2).如果在一个类中出现同名的类属性、对象属性，那么对象调用的是对象属性(优先级)  
3).对象可以修改对象属性的值，但是尝试去修改类属性的值是做不到，在代码中仅仅是动态为对象添加了一个对象属性罢了  
4).类可以调用类属性(获取、设置)，但是不能调用对象属性  
  
总结：对象可以调用对象以及类的、类能调用类的不能调用对象的 晚的可以调用早的，早的不能调用晚的 相同周期的都能调用*

5.类函数和对象函数

类函数：在定义函数的上面一行书写@classmethod，特点：没有self 有cls

对象函数：定义在class中的普通的def函数

*类函数： 格式： @classmethod  
 def 函数名(cls,...)  
对象函数：格式： def 函数名(self,...)  
搞清楚关键字self和cls的关系：  
self可以理解为当前对象或者正在被创建的对象  
cls可以理解为当前类的类名  
【记住】： 类和对象的关系！！  
 类可以调用类(属性、函数)，不能调用对象(属性、函数)  
 对象可以调用类(属性、函数)，也能调用对象(属性、函数)*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**演示类函数和对象函数的使用**

**class** Demo:  
 *#定义类属性a赋值为10* a = 10  
 *#定义对象属性b* **def** \_\_init\_\_(self,b):  
 self.b = b  
 *#定义对象函数func1* **def** func1(self):  
 print(**'我是对象函数func1...'**)  
 *#定义对象函数func2* **def** func2(self):  
 *'''  
 在对象函数中调用对象属性和对象函数 ✔  
 在对象函数中调用类属性和类函数 ✔  
 '''* print(self.b) *#对象函数func2中调用对象属性b,✔* self.func1() *#对象函数func2调用对象函数func1,✔* print(self.a)  *#对象函数func2调用类属性a,✔* self.func3() *#对象函数func2调用类函数func3,✔*  
 *#定义类函数func3* @classmethod  
 **def** func3(cls):  
 print(**'我是类函数func3...'**)  
 *#定义类函数func4* @classmethod  
 **def** func4(cls):  
 *'''  
 在类函数中调用类属性和类函数 ✔  
 在类函数中调用对象属性和对象函数* ***×*** *'''* print(cls.a) *#类函数func4中调用类属性a,✔*  
 cls.func3() *#类函数func4中调用类函数func3,✔*  
 *以下代码有问题：*  
 *# print(cls.b) #类函数func4想调用对象属性b，****×*** *# cls.func1()* *#类函数func4想调用对象函数func1，****×***

*由于cls关键字去尝试直接调用了对象属性和对象函数，是不允许的，所以报错了；我们可以用别的方式解决，方式如下：  
 先进行对象实例化 --> d = cls(100) 类名加小括号就是对象了  
 d就可以随意的去访问对象属性和对象函数了* d = cls(100)  
 print(d.b)  
 d.func1()  
*#实例化Demo对象*d = Demo(20)  
d.func2()  *#得到 20*

*我是对象函数func1...*

*10*

*我是类函数func3...*  
*#通过类名调用类函数*Demo.func4() *#得到 10*

*我是对象函数func3...*

*100*

*我是类函数func1...*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6.类和对象的关系(**生命周期**)

原则：先有类、后有对象(内存角度出发)

注意事项：

1).对象可以调用类属性和对象属性

2).如果类属性和对象属性重名，那么对象调用的话，优先调用的是对象属性

3).类只能调用类属性，不能调用对象属性(有类还不一定有对象)

4).对象不能修改类属性，如果尝试这么做，仅仅达到的效果是为对象动态添加一个对象属性

5).类名可以修改类属性

6).对于cls关键字，表示的意思就是类型，可以追加(...)进行对象实例化

7).对象函数中可以调用对象属性、类属性；对象函数、类函数

8).类函数中可以调用类属性、类函数；不能直接通过cls调用对象属性以及对象函数，

但是可以实例化对象再进行

7.**对象序列化和反序列化(钝化和活化) （面试喜欢用）**

序列化：(钝化)

内存中的对象如果想要被执行存储起来，可以通过序列化机制(将对象碾碎，变为散装数据\二进制数据)

持久存储到文件的某个位置

反序列化：(活化)

将文件中的散装对象，反序列化到程序中，还原成为一个对象实体

pickle模块的介绍：（就是用来序列化和反序列化的）

函数：

1).dump(obj, fr)：将obj对象序列化到文件中(取决于fr中的path) #没有返回值

2).load(fw)：将fw(path中的数据)加载到程序，以对象返回 #有返回值

【注意】：文件对象不要忘记关闭！！！

**使用pickle模块演示对象序列化和反序列操作：  
load()**  #有返回值 **dump()** #没有返回值

**import** pickle

**class** Person:  
 **def** \_\_init\_\_(self,name,age,sex):  
 self.name = name  
 self.age = age  
 self.sex = sex  
 **def** \_\_str\_\_(self):  
 **return 'name：%s，age：%s，sex：%s'** %(self.name,self.age,self.sex)

*#实例化Person对象*p = Person(**'貂蝉'**,18,**'女'**,162.0,84,**'靓丽'**)fw = open(**r'person.txt'**,**'wb'**)  *#创建一个文件person.txt，打开方式wb*

*#对象序列化操作：将程序中的对象p序列化到person.txt文件中保存起来*pickle.dump(p,fw)fw.close()

fr = open(**r'person.txt'**,**'rb'**) *#打开文件person.txt  
#对象反序列化操作：将person.txt中的二进制数据反序列化到程序中，还原成为一个Person对象*p1 = pickle.load(fr)  
print(p1,type(p1))  
fr.close()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**演示容器对象的序列化和反序列化操作：  
有4个Person对象，我们可以先将它们存入到list中，然后将此list对象序列化到某一个文件中，  
之后再将其反序列化到程序，还原得到一个list对象**per1 = Person(**'貂蝉'**,18,**'女'**,162.0,84,**'靓丽'**)  
per2 = Person(**'西施'**,16,**'女'**,160.0,80,**'柔弱'**)  
per3 = Person(**'昭君'**,19,**'女'**,165.0,88,**'清纯'**)  
per4 = Person(**'环环'**,23,**'女'**,170.0,130,**'肥美'**)  
lt = []  
lt.append(per1)  
lt.append(per2)  
lt.append(per3)  
lt.append(per4)  
*#将lt对象序列化到文件persons.txt中*fw = open(**r'persons.txt'**,**'wb'**)  
pickle.dump(lt,fw)  
fw.close()  
*#将persons.txt中的二进制数据反序列化到程序中*fr = open(**r'persons.txt'**,**'rb'**)  
lt1 = pickle.load(fr)  
fr.close()  
*# print(lt1) #因为放进去的是对象，所以只能看到四个对象，看不到内容  
#循环遍历lt1***for** i **in** lt1:  
 print(i)