第7章 Python与面向对象

AI程序设计课程组

面向过程V. S. 面向对象

分别使用面向过程和面向对象来实现五子棋

编程思想	实现步骤	特点
面向过程	✓ 开始游戏	先分析解决问题的步骤 使用函数把这些步骤以此实现 使用的时候需要逐个调用函数
	✓ 黑子先走	
	✓ 绘制画面	
	✓ 轮到白子。	
	✓ 绘制画面	
	✓ 判断输赢	
	✓ 返回步骤2	
	✓ 输出最后结果	
面向对象	✓ 黑白双方：这两方的行为一样	把解决问题的事物分为多个对象 对象具备解决问题过程中的行为
	✓ 棋盘系统：负责绘制画面	
	✓ 规则系统：负责判断诸如犯规、输赢等。	

面向对象

面向对象设计思想

- 面向过程——根据功能需求，总结需要完成的步骤，按步骤从前到后写代码，最后达到既定目标
- 面向对象也是我们认识世界的一种重要方式，在现实世界中存在各种不同形态的事物，这些事物之间存在着各种各样的联系。在程序中使用对象来映射现实中的事物，使用对象间的关系来描述事物之间的联系，这种思想就是面向对象。
- 面向对象——基于类和对象，以数据为中心

面向对象设计思想

“一切皆对象”

- 整型、浮点型、字符串值是对象
- 列表、元组、字典、集合是对象
- 函数是对象
- ……

```
print(type(152))
print(type('Ai Programming'))
print(type([1,5,2]))
print(type({'zs':'2023072456','ls':'20230010001'}))
```

```
<class 'int'>
<class 'str'>
<class 'list'>
<class 'dict'>
```

```
listdemo=['Ai Programming','Python','Deep Learning']
listdemo.sort()
print(listdemo)
```

```
['Ai Programming', 'Deep Learning', 'Python']
```

列表对象调用sort方法

面向对象程序设计

- **面向对象程序设计具有以下几个特征：**
  - 程序由若干对象组成，每个对象是由数据以及对这些数据所能实施的操作所构成的封装体；
  - 对象的特征由相应的类来描述；
  - 对数据的使用是通过向包含数据的对象发送消息来实现的；
  - 一个类所描述的对象特征可以从其它的类继承获得。
- **相同消息可以被发送给不同对象，可能引发不同的程序行为，这是面向对象特有的多态**

类和对象

类和对象的关系

- 面向对象编程有两个非常重要的概念：类和对象。
- 具有相似特征和行为的事物的集合统称为类， 类定义了一件事物的抽象特点，它包含了这件事物的属性和操作方法。类是对象的模板。
- 对象是面向对象编程的核心。对象则指的是类的实例。它将对象作为程序的基本单元，将程序和数据封装其中，以提高软件的重用性、灵活性和扩展性，对象里的程序可以访问及经常修改对象相关联的数据。
- 对象是根据类创建的，一个类可以对应多个对象。

类和对象的关系

类定义了一件事物的抽象特点，它包含了这件事物的属性和操作方法，对象是类的实例。

- 举个例子，猫是一个类，既具有身长、毛色、体重等属性，也有叫、跑、跳等操作方法。
- 假设有两条猫，它们的身长、毛色、体重等属性不太相同，而且它们的叫声、跳跃的方法也可能不相同，但是它们都归一类——猫。它们只是这个类下面的不同对象。

类和对象的关系

- 可以把玩具模型看作一个类
- 把每个玩具看作一个对象

还有哪些类和对象的案例？
汽车、食堂饭菜
……

类的定义

类是由3部分组成的：

- 类的名称：类名。
- 类的属性：一组数据，比如学号、姓名。
- 类的方法：允许进行操作的方法，比如说话。

使用class关键字来声明一个类，基本格式如下：

```
class 类名:
    类的属性
    类的方法
```

类的定义


```
class ClassName:
    #类文档字符串
    #类体
    pass
```

Python提供了一个保留关键字pass，在pass定义和执行的时候，什么事情都不会发生。

```
class ClassName:
    name='testdemo'
    def printname():
        print(ClassName.name)
```

```
print(ClassName.name)
ClassName.printname()
```

```
testdemo
testdemo
```

根据类创建对象

根据类创建对象的语法格式如下：

```
对象名 = 类名()
```

```
class ClassName:
    name='testdemo'
    def printname():
        print(ClassName.name)
```

```
aiclass=ClassName()
print(aiclass.name)
```

要想给对象添加属性，可以通过如下方式：

```
对象名.新的属性名 = 值
```

```
aiclass=ClassName()
print(aiclass.name)
aiclass.duaration=44
print(aiclass.duaration)
```

```
testdemo
44
```

构造方法和析构方法

构造方法

- 构造方法指的是__init__方法。
- 当创建类的实例的时候，系统会自动调用构造方法，从而实现对类进行初始化的操作。

```
class Score:
    def __init__(self,name):
        self.name=name
        self.english=0
        self.math=0
    def setenglish(self,english):
        self.english=english
    def setmath(self,math):
        self.math=math
```

```
zsscore=Score('zs')
print(zsscore.name)
print(zsscore.english)
print(zsscore.math)
```

```
zs
0
0
```

析构方法

- 当删除一个对象来释放类所占用资源的时候，Python解释器默认会调用另外一个方法，这个方法就是__del__()方法。
- __del__方法被称为析构方法。

```
class Score:
    def __init__(self,name):
        self.name=name
        self.english=0
        self.math=0
    def setenglish(self,english):
        self.english=english
    def setmath(self,math):
        self.math=math
    def __del__(self):
        print(self.name,'已经被清理，其占用空间被释放')
```

```
zsscore=Score('zs')
print(zsscore.name)
del zsscore
print(zsscore.name)
```

```
zs
zs 已经被清理，其占用空间被释放

---------------------------------------------------------------------------
NameError                                 Trac
<ipython-input-64-66e5748a5098> in <module>
      2 print(zsscore.name)
      3 del zsscore
----> 4 print(zsscore.name)

NameError: name 'zsscore' is not defined
```

self的使用

self的使用

- 在方法的列表中，第1个参数永远都是self。
- self的字面意思是自己，表示的是对象自身。
- 当某个对象调用方法的时候，Python解释器会把这个对象作为第1个参数传给self，开发者只需要传递后面的参数就可以了。

self的使用

- **对象是数据和操作的集合**
  - 数据描述对象的状态，每个对象可以具有独立的状态
  - 操作对状态进行改变
- **通过self参数，成员函数可以访问改变当前对象的状态**

```
class Score:
    def __init__(self,name):
        self.name=name
        self.english=0
        self.math=0
    def setenglish(self,english):
        self.english=english
    def setmath(self,math):
        self.math=math
```

```
zsscore=Score('zs')
zsscore.setenglish(100)
zsscore.setmath(95)
print(zsscore.name)
print(zsscore.english)
print(zsscore.math)
```

```
zs
100
95
```

Score('zs')创建了一个名为zsscore的对象
在调用zsscore中的setenglish()方法和setmath()方法的时候，只传入了一个参数。zscore即为函数中的参数self

通过类调用成员函数

- **成员函数也可以通过类进行访问和调用，此时应自行指定需要传入的参数**

```
zsscore=Score('zs')
zsscore.setenglish(100)
print('张三英语成绩：',zsscore.english)
Score.setenglish(zsscore,88)
print('修改后的英语成绩：',zsscore.english)
```

```
张三英语成绩： 100
修改后的英语成绩： 88
```

```
class Score:
    def __init__(self,name):
        self.name=name
        self.english=0
        self.math=0
    def setenglish(self,english):
        self.english=english
    def setmath(self,math):
        self.math=math
    def __del__(self):
        print(self.name,'已经被清理，其占用空间被释放')
```

封装

封装


```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
laowang = Person(“老王”, 30)
laowang.age = 300
print(laowang.age)
```

人的年龄可以随便设置，显然不可行

封装

为了保护类里面的属性，可以采用如下方式解决：

1. 把属性定义为私有属性，即在属性名的前面加上两个下划线；
2. 添加用于设置或获取属性值的两个方法供外界调用。

封装


```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age
    def setage(self,age):
        if 0<age<150:
            self.__age=age
    def getage(self):
        return self.__age
```

```
zs = Person("zhangsan", 30)
print(zs.__age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-46ba8bb4bf1e> in <module>
      1 zs = Person("zhangsan", 30)
----> 2 print(zs.__age)

AttributeError: 'Person' object has no attribute '__age'
```

Person.__age为私有属性，无法被对象zs直接访问，需要通过Person的内部方法来进行访问与设置

封装


```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age
    def setage(self,age):
        if 0<age<150:
            self.__age=age
    def getage(self):
        return self.__age
```

```
zs = Person("zhangsan", 30)
zs.setage(80)
print(zs.getage())
```

```
80
```

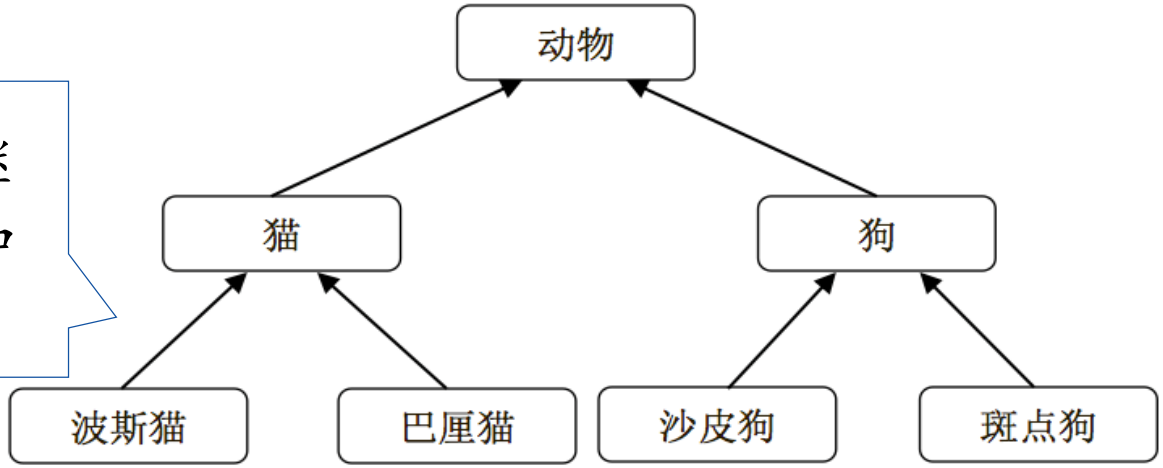
Person.__age为私有属性，无法被对象zs直接访问，需要通过Person的内部方法来进行访问与设置

继承

继承

- 在现实生活中，继承一般指的是子女继承父辈的财产。
- 在程序中，继承描述的是事物之间的所属关系。
- 类的继承是指在一个现有类的基础上构建一个新的类，构建出来的新类被称作子类。

波斯猫和巴厘猫都继承自猫，而沙皮狗和斑点狗都继承狗。



继承

- 继承也往往用于解决具有相似尚属性和方法的类之间的代码复用问题

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def printinfo(self):
        print('{}的年龄是{}岁'.format(self.name,self.age))
```

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def printinfo(self):
        if(self.age>=18):
            print(self.name,'已经成年')
```

Sdudent类和Person类基本一样

继承

Python程序中，继承使用如下语法格式标注：

```
class 子类名(父类名):
```

假设有一个类为A，A派生出来了子类B，示例如下：

```
class B(A):
class A(object):
```

父类又称为基类
子类又称为派生类

继承

[43]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def printinfo(self):
        print('{}的年龄是{}岁'.format(self.name,self.age))
```

[44]:
```
class Student(Person):
    def printinfo(self):
        if(self.age>=18):
            print(self.name,'已经成年')
```

[45]:
```
zs=Student('zhangsan',56)
zs.printinfo()
```

```
zhangsan 已经成年
```

Sdudent类继承自Person类，仅重写printinfo方法

重写和调用父类方法

- 在继承关系中，子类会自动拥有父类定义的方法，但是有时子类想要按照自己的方式实现方法，即对父类中继承来的方法进行重写，使得子类中的方法覆盖掉跟父类同名的方法。
- 需要注意的是，在子类中重写的方法要和父类被重写的方法具有相同的方法名和参数列表。

__init__函数的继承与重写

- **__init__函数可以继承,** 如果派生类中不定义**__init__**函数，则**继承**基类的**__init__**函数 完成派生类对象的初始化。
- **如果 __init__ 函数发生重写，派生类应负责全部数据的初始化。**

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def printinfo(self):
        print('{}的年龄是{}岁'.format(self.name,self.age))
```

```
class Student(Person):
    def __init__(self,name,age,grade):
        super().__init__(name,age)
        self.grade=grade
    def printinfo(self):
        if(self.age>=18):
            print('{}已经成年，是{}年级学生'.format(self.name,self.grade))
```

```
zs=Student('zhangsan',56,'大三')
zs.printinfo()
```

zhangsan已经成年，是大三年级学生

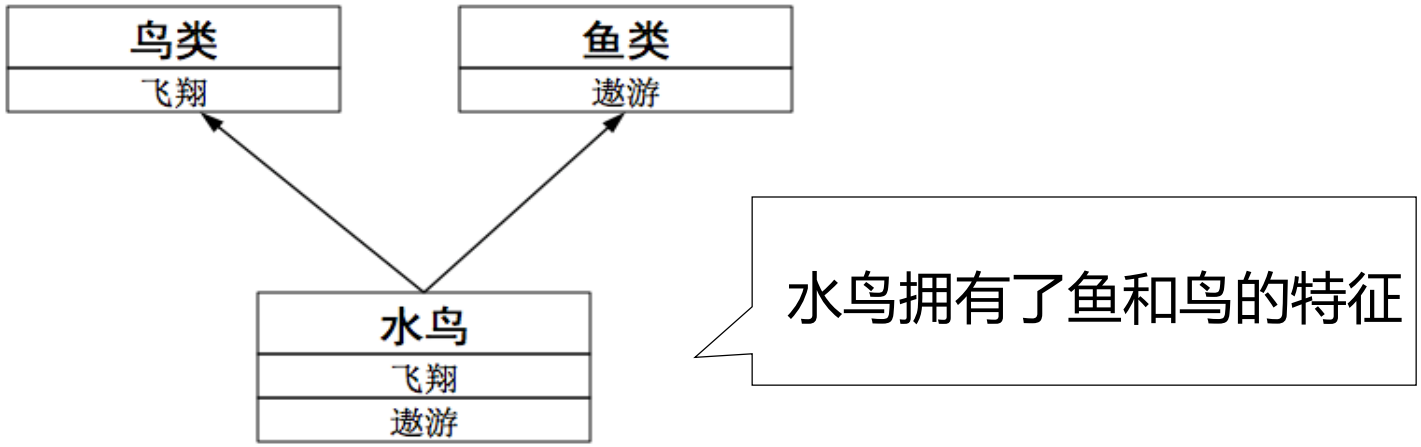
此处用super代表了父类，替父类执行某些方法。super的更多用法可见课后延伸学习资料

多继承

现实生活中，一个派生类往往会有多个基类。比如沙发床是沙发和床的功能的组合，这都是多重继承的体现。

多继承

Python支持多继承，多继承就是子类拥有多个父类，并且具有它们共同的特征，即子类继承了父类的方法和属性。



多继承

多继承可以看做是单继承的扩展，语法格式如下：

```
class 子类名(父类1，父类2…):
```

如果子类继承的多个父类间是平行的关系，子类先继承的哪个类就会调用哪个类的方法。

重写和调用父类方法

- 在继承关系中，子类会自动拥有父类定义的方法，但是有时子类想要按照自己的方式实现方法，即对父类中继承来的方法进行重写，使得子类中的方法覆盖掉跟父类同名的方法。
- 需要注意的是，在子类中重写的方法要和父类被重写的方法具有相同的方法名和参数列表。

多态

不同国家的人，打招呼的方式是不同的。

多态

A类

```
class A(object):
    def test(self):
        print("--A--test")
```

B类

```
class B(A):
    def test(self):
        print("--B--test")
```

```
def func(temp):
    temp.test()
```

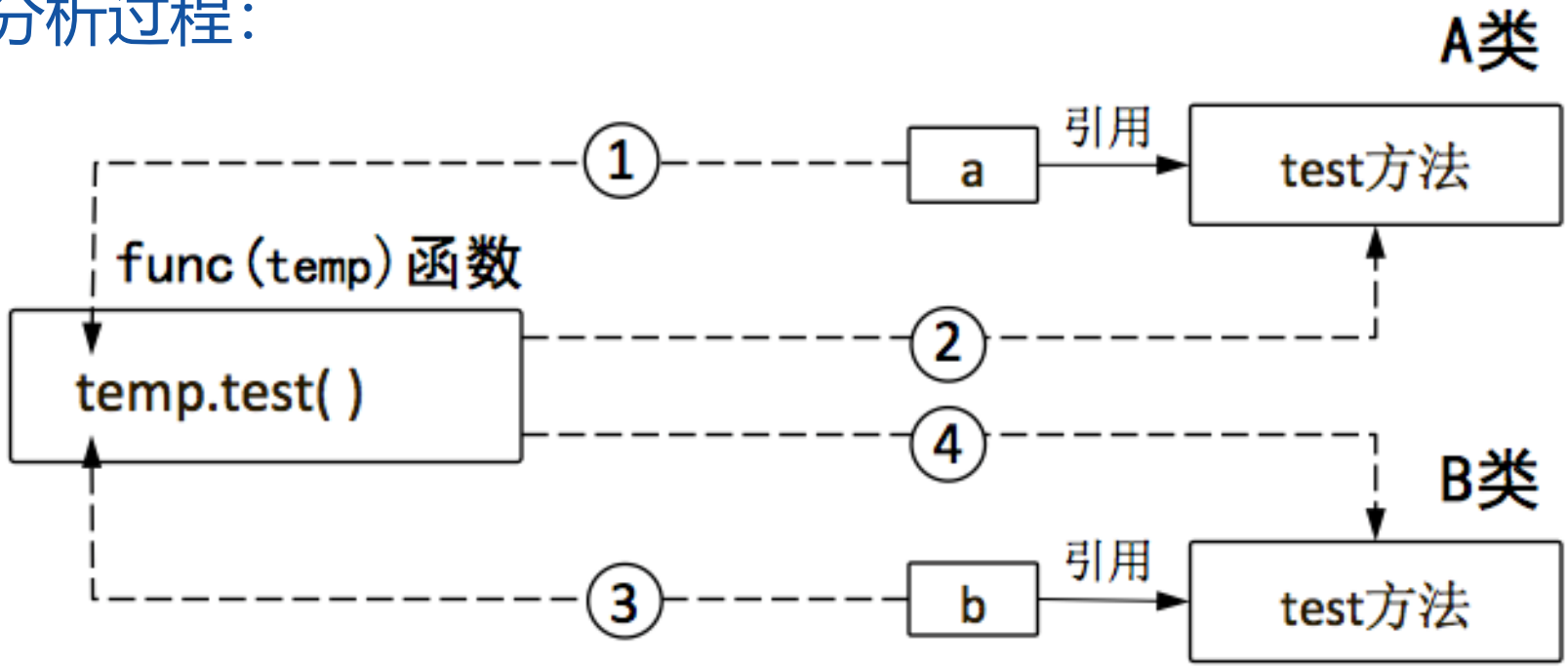
```
a = A()
b = B()
func(a)
func(b)
```

a、b的对象

两次调用func函数

结果不一样

分析过程：



本章小结

- 类与对象
- 类的定义与使用
- 封装
- 继承
- 多态