

类。下面开始学习继承的相关知识,从而编程实现学校师生管理的功能。

9.5.1 单继承和多继承

子类继承父类时,只在定义子类时将父类(可以是多个)放在子类之后的圆括号里即可。

1. 单继承

语法格式如下。

```
class 子类名(基类名):  
    #类定义部分
```

- 如果该类没有显式指定继承自哪个类,则默认继承 object 类(object 类是 Python 中所有类的父类,即要么是直接父类,要么是间接父类)。
- 使用继承时,子类和父类之间的关系应该是“属于”关系。例如,学生是人,教师也是人,因此这两个类都可以继承“人”类。但是,计算机类却不能继承 Person 类,因为计算机并不是一个人。

【例 9-21】 中国人(类)继承人(类)。

```
class Person(object):  
    #定义一个父类  
    def speak(self):  
        #父类中的方法  
        print("person is speaking....")  
  
class Chinese(Person):  
    #定义一个子类,继承 Person 类  
    def work(self):  
        #在子类中定义其自身的方法  
        print('is working...')  
  
p = Person()  
#定义一个父类对象  
p.speak()  
#调用父类的方法  
c = Chinese()  
#定义一个子类对象  
c.speak()  
#子类调用父类的方法  
c.work()  
#子类调用本身的方法
```

程序运行结果如图 9-19 所示。

从结果可以看出,c 对象中只定义了 work()方法,但是也可以调用父类定义的 speak()方法,说明继承了父类的方法。

2. 多继承

Python 和其他面向对象语言不同,它支持多重继承,就是

控制台	绘图
person is speaking.... person is speaking.... is working...	

图 9-19 继承

可以同时继承多个父类的属性和方法。多重继承的语法如下。

```
class 子类名(基类名 1, 基类名 2, ...):
    # 类定义部分
```

【例 9-22】 多重继承示例。

```
class People:
    def introduce(self):
        print("我是一个人, 名字是:", self.name)
class Animal:
    def display(self):
        print("人也是高级动物")
# 同时继承 People 和 Animal 类
# 其同时拥有 name 属性、introduce() 和 display() 方法
class Person(People, Animal):
    pass
p1 = Person()
p1.name = "李思思"
p1.introduce()
p1.display()
```

程序运行结果如图 9-20 所示。

事实上, 大部分面向对象的编程语言都只支持单继承, 即子类有且只能有一个父类。而 Python 却支持多重继承(C++ 也支持多重继承)。和单继承相比, 多重继承容易让代码逻辑复杂、思路混乱, 一直备受争议, 中小型项目中较少使用, 后来的 Java、C#、PHP 等干脆取消了多重继承。使用多重继承经常面临的问题是, 多个父类中包含同名的类方法。对于这种情况, Python 的处理措施是: 根据子类继承多个父类时这些父类的前后次序决定, 即排在前面父类中的类方法会覆盖排在后面父类中的同名类方法。

控制台

绘图

我是一个人, 名字是: 李思思
人也是高级动物

图 9-20 多重继承

9.5.2 重写(覆盖)父类方法

子类不想原封不动地继承父类的方法, 而是想做一定的修改, 这就需要采用方法的重写。子类中重写了与基类(父类)同名的方法, 在子类实例调用方法时, 实际调用的是子类中的覆盖版本, 这种现象叫作覆盖, 因此方法重写又称方法覆盖。

1. 重写构造方法

Python 中, 类的构造方法是 `__init__()`。当一个类被子类继承且子类重写了构造方法后, 若子类还想使用父类的构造方法, 如果直接通过创建的子类对象调用父类的方法会报错。解决办法有两个: 一个是调用超类方法的未绑定版本; 一个是使用 `super()` 函数。

首先看一个例子：鸟(bird)类分为麻雀(sparrows)、燕子(swallows),还有会说话的鹦鹉(parrots),所有的鸟都会飞,鹦鹉还会唱歌。

【例 9-23】 鹦鹉类重写了构造方法。

```
import random as r          #导入 random 模块
class Bird:
    def __init__(self):
        self.x = r.randint(0, 10)    #调用 randint() 函数产生随机数
        self.y = r.randint(0, 10)
    def fly(self):
        print("我任意飞,我的位置是:", self.x, self.y)
class Sparrows(Bird):        #定义麻雀类,不需要有个性,直接继承 Bird 类的全部属性和方法
    pass
class Swallows(Bird):        #定义燕子类,不需要有个性,直接继承 Bird 类的全部属性和方法
    pass
class Parrots(Bird):         #定义鹦鹉类,这种鸟会说话,除了继承以外,还要添加一个说的方法
    def __init__(self):
        self.sounds = 'y'
    def speak(self):
        if self.sounds=='y':
            print("我是一只会说话的鸟,呵呵呵")
            self.sounds = False
        else:
            print("说累了,现在不想说话!")
#主程序
bird = Bird()
bird.fly()
sparrows = Sparrows()
sparrows.fly()
swallows = Swallows()
swallows.fly()
parrots = Parrots()
#parrots.fly()
```

程序运行结果如图 9-21 所示。

控制台	绘图
我任意飞,我的位置是: 8 6	
我任意飞,我的位置是: 8 1	
我任意飞,我的位置是: 0 2	

图 9-21 重写构造方法

如果将最后一句的注释符号#去掉,鸚鵡对象也调用 fly()方法,则程序运行结果如图 9-22 所示。

控制台	绘图
<pre> 我任意飞,我的位置是: 5 5 我任意飞,我的位置是: 9 9 我任意飞,我的位置是: 6 7 Traceback (most recent call last): File "<program.py>", line 29, in <module> parrots.fly() File "<program.py>", line 7, in fly print("我任意飞,我的位置是: ", self.x, self.y) AttributeError: 'Parrots' object has no attribute 'x' </pre>	

图 9-22 鸚鵡对象也调用 fly()方法

同样是继承了鸟类,为什么麻雀、燕子对象都可以飞,而鸚鵡对象就不行了呢? 其实这里抛出的异常说得很清楚,'Parrots' object has no attribute 'x','Parrots'对象没有 x 属性。原因是在 Parrots 类中重写了 __init__() 方法,但新的方法里没有初始化 x 和 y 坐标,因此调用 fly()方法就会报错。解决的办法是在 Parrots 类中重写 __init__()方法的时候先调用父类 (Bird)的 __init__()方法。有两种方法可以实现:

第一种是调用未绑定的超类构造方法,如下所示。

```

def __init__(self):
    Bird.__init__(self)
    self.sounds = 'y'

```

第二种是使用 super()函数,如下所示。

```

def __init__(self):
    super().__init__()
    self.sounds = 'y'

```

#主程序

```

bird = Bird()
bird.fly()
sparrows = Sparrows()
sparrows.fly()
swallows = Swallows()
swallows.fly()
parrots = Parrots()
parrots.fly()
parrots.speak()
parrots.speak()

```


最终,程序运行结果如图 9-23 所示。

2. 重写普通方法

父类的成员都会被子类继承,当父类中的某个方法不完全适用于子类时,就需要在子类中重写父类的这个方法。

【例 9-24】 重写父类方法。

```
class A:
    def work(self):
        print("A 类的 work() 方法被调用")
class B(A):
    def work(self):
        print("B 类的 work() 方法被调用")
b = B()
b.work()                                #子类已经覆盖了父类的方法,调用 B 类的 work() 方法
```

程序运行结果如图 9-24 所示。

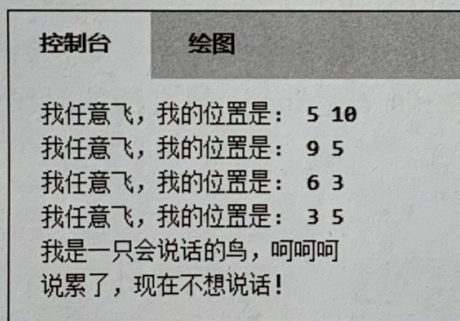


图 9-23 重写了构造方法

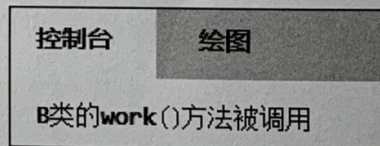


图 9-24 重写父类方法

9.5.3 调用父类方法

从 9.5.2 节的例子可以看出,如果在子类中重写了从父类继承来的类方法,那么,当在类的外部通过子类对象调用该方法时,Python 总是会执行子类中重写的方法,这就产生了一个新的问题,即如果想调用父类中被重写的这个方法,该怎么办?

有 3 种形式可以调用被子类覆盖的父类方法,分别为:

- 父类名.父类方法(self)。
- super(子类名,self).父类方法()。
- super().父类方法()。

【例 9-25】 子类调用父类的方法。

```
class A:
    def work(self):
        print("A 类的 work() 方法被调用")
```



```

class B(A):
    def work(self):
        print("B 类的 work() 方法被调用")
    def doworks(self):
        self.work()
        super(B, self).work()
        super().work()
b = B()
b.doworks()
b.work()

print("-----以下两种方法都可用子类对象 b 调用覆盖了的父类的方法-----")
super(B, b).work()
A.work(b)

```

定义子类 B, 继承了父类 A
重写了父类的 work() 方法
调用 B 类的 work() 方法
在子类方法中调用超类的方法
在子类方法中调用超类的方法
创建子类对象
子类对象可以调用自己定义的方法
子类已经覆盖了父类的方法, 调用子类自己的 work() 方法

程序运行结果如图 9-25 所示。

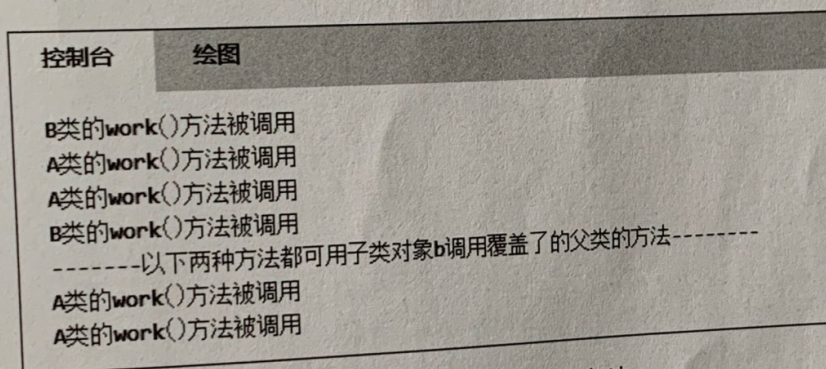


图 9-25 子类调用父类的方法

9.5.4 案例实现

根据前面所学知识,设计一个学校成员类 SchoolMember, 包含构造方法、析构方法、两个成员方法、注册功能和显示个人信息功能。设计教师类 Teacher 和学生类 Student, 这两个类都继承成员类 SchoolMember, 教师类重写了构造方法, 增加了工资和课程两个属性, 同时增加了授课方法; 学生类重写了构造方法, 增加了专业、学费和学费总额 3 个属性, 重写了个人信息这个方法, 增加了交学费方法。

```

class SchoolMember(object):
    '''学校成员基类'''
    member = 0
    # 用于师生数量

    def __init__(self, name, age, sex, type):
        self.name = name
        self.age = age
        self.sex = sex

```