人工智能程序设计


python
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
turtle.circle(40, 80)
turtle.circle(-40, 80)
turtle.fd(40)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)



人工智能程序设计

6.3 面向对象程序设计

北京石油化工学院 人工智能研究院

刘 强

6.3 面向对象程序设计

面向对象编程的核心特性包括继承、多态和封装。

这些特性使得代码更加模块化、可重用和易于维护。

6.3.1 类的继承

继承允许一个类获得另一个类的属性和方法，实现代码重用。子类通过在类名后的括号中指定父类来实现继承。子类会自动获得父类的所有属性和方法。

```
class Animal:          # 父类（基类）
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass

class Dog(Animal):     # 子类（派生类）
    def speak(self):
        return f"{self.name}说：汪汪！"
```

```
class Cat(Animal):     # 另一个子类
    def speak(self):
        return f"{self.name}说：喵喵！"

## 使用继承
dog = Dog("小黄")
print(dog.speak())  # 输出：小黄说：汪汪!
```

6.3.1 类的继承

调用父类方法

使用super()函数可以调用父类的方法，特别是在重写方法时保留父类的功能。下面通过一个示例说明如何使用super()调用父类方法。

示例 6.3.1：车辆管理系统

```
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def start(self):
        return f"{self.brand} {self.model}启动了"

class Car(Vehicle):
    def __init__(self, brand, model, doors):
        super().__init__(brand, model)  # 调用父类构造方法
        self.doors = doors

    def start(self):
        base_message = super().start()  # 调用父类方法
        return f"{base_message}，有{self.doors}个门"

car = Car("丰田", "凯美瑞", 4)
print(car.start())  # 输出：丰田 凯美瑞启动了，有4个门
```

6.3.2 方法重写

子类可以重写父类的方法，提供特定的实现。这是实现多态性的基础。

```
class Shape:
    def area(self):
        return 0

    def describe(self):
        return f"这是一个形状，面积为{self.area()}"

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):         # 重写父类方法
        return 3.14159 * self.radius ** 2

## 使用重写的方法
circle = Circle(5)
print(circle.describe())    # 输出：这是一个形状，面积为78.53975
```

6.3.1 类的继承

继承允许一个类获得另一个类的属性和方法，实现代码重用。子类通过在类名后的括号中指定父类来实现继承。子类会自动获得父类的所有属性和方法。

```
class Animal:          # 父类（基类）
    def __init__(self, name):
        self.name = name

    def speak(self):
        pass

class Dog(Animal):     # 子类（派生类）
    def speak(self):
        return f"{self.name}说：汪汪！"
```

```
class Cat(Animal):     # 另一个子类
    def speak(self):
        return f"{self.name}说：喵喵！"

## 使用继承
dog = Dog("小黄")
print(dog.speak())  # 输出：小黄说：汪汪!
```

6.3.3 多态性

多态允许不同的对象对同一消息做出不同的响应，使程序更加灵活。基于6.3.1节中定义的Animal、Dog和Cat类，我们可以创建不同类型的对象，通过同一个接口调用它们的方法，每个对象会根据自身的类型执行不同的行为。

```
## 基于6.3.1节中定义的Animal、Dog、Cat类，创建不同类型的动物
animals = [Dog("旺财"), Cat("咪咪")]

## 多态性：同一个接口处理不同类型的对象
for animal in animals:
    print(animal.speak())
## 输出：
## 旺财说：汪汪！
## 咪咪说：喵喵！
```

6.3.3 多态性

多态的优势

- 灵活性：同一接口处理不同类型对象
- 可扩展性：添加新类型无需修改现有代码
- 代码重用：通用函数适用于所有相同接口的类
- 易维护：降低模块间耦合度

实践练习

练习 6.3.1：图形类继承体系

设计一个图形类的继承体系。创建一个Shape基类，包含计算面积和周长的方法。然后创建Rectangle和Triangle两个子类，分别实现计算矩形和三角形面积、周长的功能。

练习 6.3.2：多态性应用

基于练习6.3.1中的图形类，创建一个图形列表，包含不同类型的图形对象。使用多态性，编写一个函数计算所有图形的总面积，并打印每个图形的信息。

练习 6.3.3：封装的应用

创建一个BankAccount类，使用私有属性存储账户余额。实现存款、取款和查询余额的方法，确保余额不能被直接修改，只能通过提供的方法进行操作。