面向对象

万永权

目录

CONTENTS

学习目标

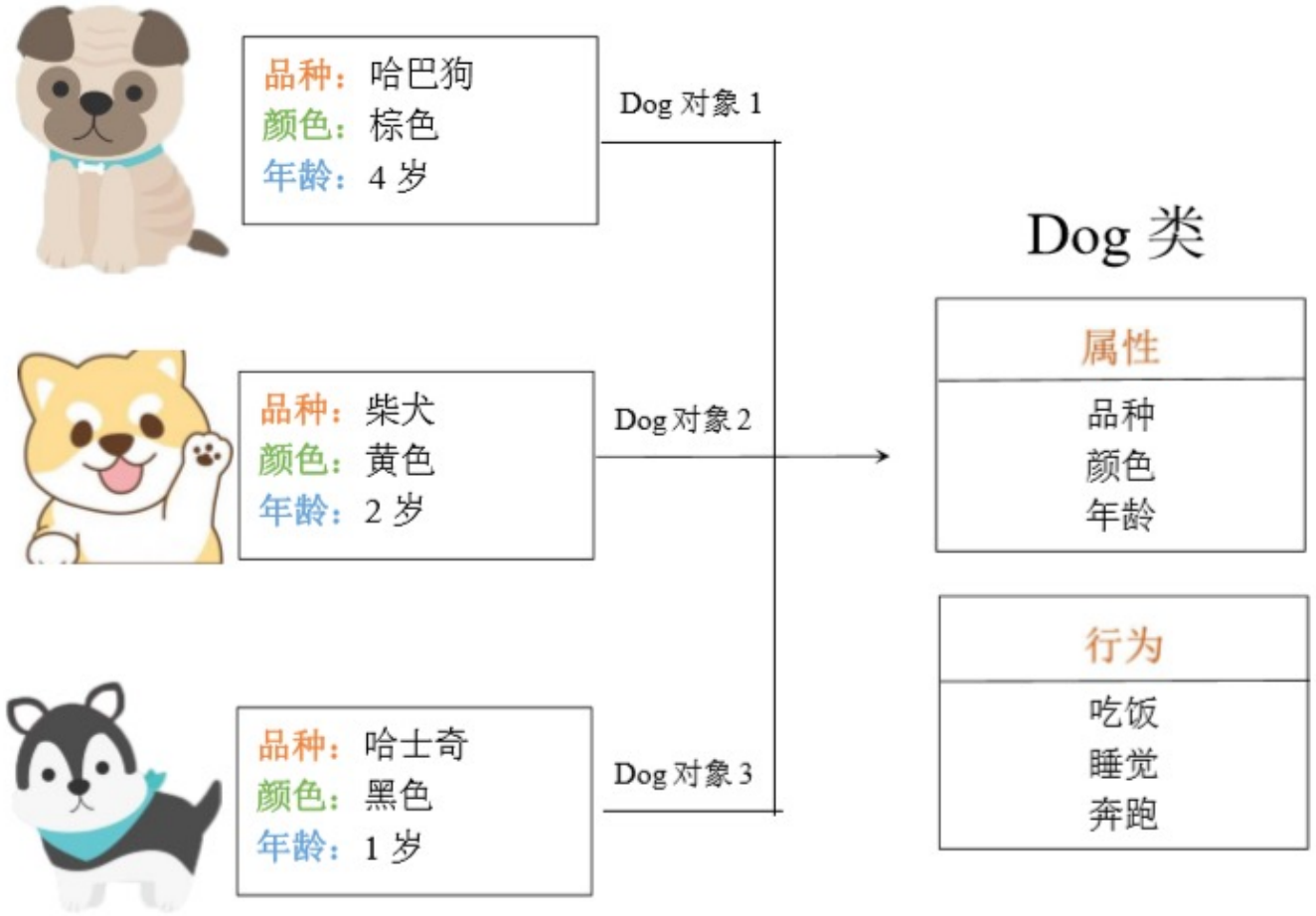
理解类和对象的概念

掌握类的基本使用方法

掌握导入模块中的类的方法

1 类和对象概述

对象是类的一个实例，有属性和行为。类则表示一个模板，用于描述一类对象的属性和行为。



1 类和对象概述

有一家宠物医院，需要记录所有的宠物信息，要怎么记录呢？难道要每个都单独记录吗？

姓名：小巴
品种：哈巴狗
颜色：棕色
年龄：4 岁
可以吃 1 碗狗粮，睡觉时间约 8 小时，不喜欢跑。

姓名：小柴
品种：柴犬
颜色：黄色
年龄：2 岁
可以吃 1 碗狗粮，睡觉时间约 8 小时，喜欢跑。

姓名：小喵
品种：暹逻猫
颜色：黑白
年龄：1 岁
可以吃 1 碗猫粮，睡觉时间约 8 小时，喜欢玩猫玩具。

---

1 类和对象概述

面向对象程序设计的思想**需要以对象来思考问题**，将现实中的实体抽象为对象，并考虑这个对象对应的**属性和行为**。

根据面向对象程序设计的思想，可以将此宠物信息的记录问题分解以下步骤。

- 首先**从此问题中抽象出对象**，对象是宠物医院的每一只宠物。
- 然后**识别出对象的属性**，比如每一只宠物都有姓名、品种、颜色以及年龄。
- 最后**识别对象的动态行为**。

1 类和对象概述

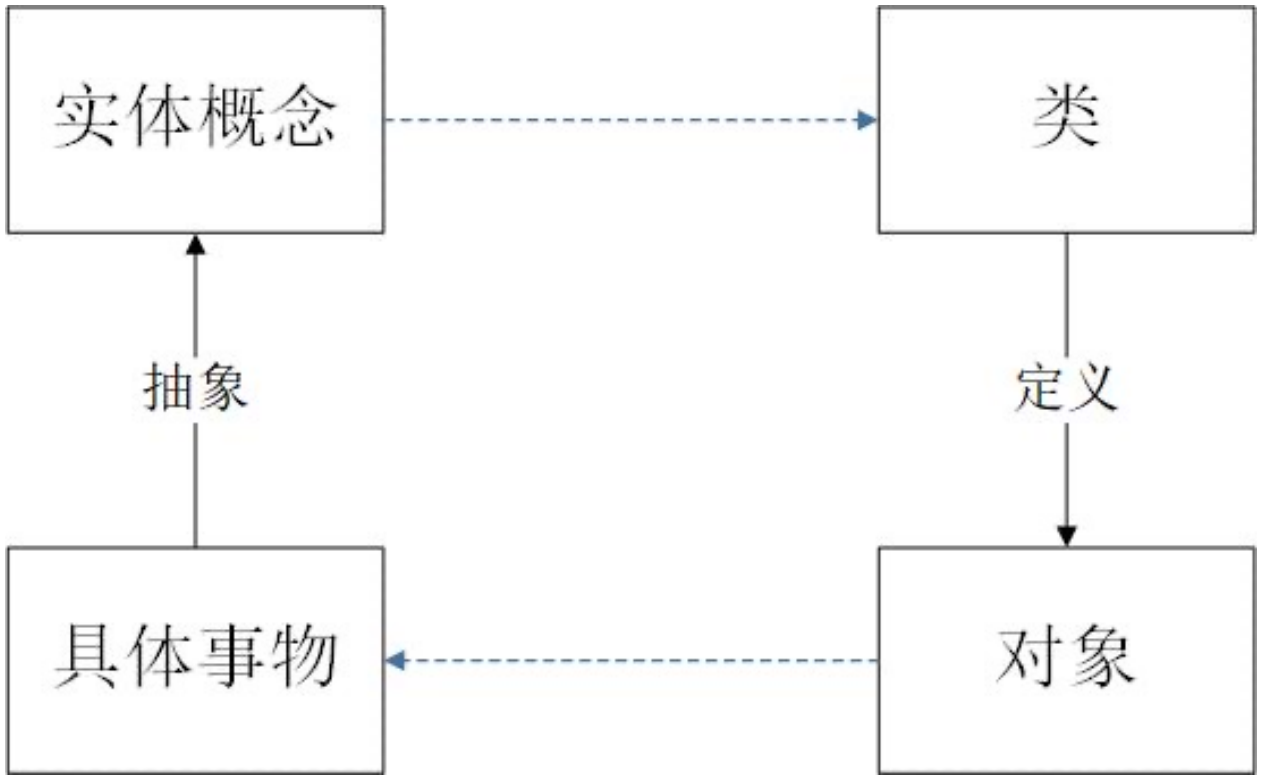
以面向对象的方式记录宠物信息

类别：狗						
属性				行为		
姓名	品种	颜色	年龄	吃狗粮（碗）	睡觉（小时）	跑步（喜欢√，不喜欢则不填）
小巴	哈巴狗	棕色	4	1	8	
小柴	柴犬	黄色	2	1	8	√

类别：猫						
属性				行为		
姓名	品种	颜色	年龄	吃猫粮（碗）	睡觉（小时）	猫玩具(喜欢√,不喜欢则不填)
小喵	暹逻猫	黑白	1	1	8	√

1 类和对象概述

类实质上就是封装对象属性和行为的载体，而对象则是类中具体的一个实例。



类的基本使用

类的定义

创建实例对象

设置属性的默认值

修改属性的值

2 类的基本使用

2.1 类的定义

定义一个类需要使用**class关键字**，类名的首字母常用大写

```
1.  class Dog:
2.      def __init__(self,name,breed,age):
3.          """初始化属性name、breed和age"""
4.          self.name = name
5.          self.breed = breed
6.          self.age = age
7.      def eat(self):
8.          """小狗正在吃狗粮"""
9.          print(f"{self.name}正在吃狗粮")
10.     def run(self):
11.         """小狗正在奔跑"""
12.         print(f"{self.name}在奔跑玩耍")
```

1.构造方法__init__()

构造方法一般用于类的初始化操作，在创建实例对象时被自动调用和执行。

2.self的作用

self的作用是代表将来要创建的实例对象本身，让实例能够访问类中的属性和对象。

3.实例方法

最少含有一个self参数，用于绑定实例对象的方法称为实例方法，可以被实例对象直接调用。

2 类的基本使用

2.2 创建实例对象

```
1.  class Dog:
2.      def __init__(self,name,breed,age):
3.          """初始化属性name、breed和age"""
4.          self.name = name
5.          self.breed = breed
6.          self.age = age
7.      def eat(self):
8.          """小狗正在吃狗粮"""
9.          print(f"{self.name}正在吃狗粮")
10.     def run(self):
11.         """小狗正在奔跑"""
12.         print(f"{self.name}在奔跑玩耍")
13. d1 = Dog("小巴","哈巴狗",4)          #创建实例对象
```

类的使用与函数类似，当定义了一个类时，其中的代码不会被执行，**当调用类来创建对象时，类中的代码才真正起作用。**

2 类的基本使用

2.2 创建实例对象-**访问属性**

访问属性，需要使用**“实例名.属性”**的方式

```
1. class Dog:
2.     """类的定义与前文一致，在此处省略部分代码"""
3. d1 = Dog("小巴","哈巴狗",4)          #创建实例对象
4. print(f"狗狗的姓名是{d1.name}")      #访问name属性
5. print(f"狗狗的品种是{d1.breed}")     #访问breed属性
6. print(f"狗狗的品种是{d1.age}")       #访问age属性
```

```
狗狗的姓名是小巴
狗狗的品种是哈巴狗
狗狗的品种是4
```

2 类的基本使用

2.2 创建实例对象-**调用实例方法**

创建实例对象后，可以调用类中的实例方法，形式是“**实例名.实例方法**”

```
1. class Dog:
2.     """类的定义与前文一致，在此处省略部分代码"""
3. d1 = Dog("小巴","哈巴狗",4)        #创建实例对象
4. d1.eat()                           #调用eat()方法
5. d1.run()                           #调用run()方法
```

```
小巴正在吃狗粮
小巴在奔跑玩耍
```

2 类的基本使用

2.2 创建实例对象-**创建多个实例对象**

可以创建**多个实例对象**，每个实例对象之间相互独立，有自己的属性，且都可以调用类中的方法

```
1. class Dog:
2.     """类的定义与前文一致，在此处省略部分代码"""
3. d1 = Dog("小巴","哈巴狗",4)      #创建一个实例对象d1
4. print(f"{d1.name}的品种是{d1.breed}，年龄是{d1.age}")
5. d1.eat()
6. d2 = Dog("小柴","柴犬",1)       #创建一个实例对象d2
7. print(f"{d2.name}的品种是{d2.breed}，年龄是{d2.age}")
8. d2.run()
```

```
小巴的品种是哈巴狗，年龄是4
小巴正在吃狗粮
小柴的品种是柴犬，年龄是1
小柴在奔跑玩耍
```

2 类的基本使用

2.3 设置属性的默认值

创建Student类，并将age属性设置为默认值19

```
1. class Student:
2.     def __init__(self,name,id,age=19):
3.         self.name = name
4.         self.id = id
5.         self.age = age
6. s1 = Student("小千",202201)
7. print(f"{s1.name}的年龄是{s1.age}")
```

小千的年龄是19

如果不想使用实例属性self.age的默认值19，可以在第6行创建实例对象时，传入3个参数。

```
s1 = Student("小千",202201,20)
```

给age属性**设置默认值**，**还可以通过在构造方法内部给age赋值的形式**，将第2~5行代码改为如下形式。

```
def __init__(self, name, id):
    self.name = name
    self.id = id
    self.age = 19
```

2 类的基本使用

2.4 修改属性的值

创建Student类

```
1. class Student:
2.     def __init__(self,name,id):
3.         self.name = name
4.         self.id = id
5.         self.age = 19
```

如果在Student类的构造函数内部直接给实例属性赋值，那如何修改此属性的值呢?

2 类的基本使用

2.4 修改属性的值-**直接修改属性的值**

通过给实例对象的属性进行直接赋值的方式，修改实例对象的值

```
1. class Student:
2.     def __init__(self,name,id):
3.         self.name = name
4.         self.id = id
5.         self.age = 19
6. s1 = Student("小千",202201)          #创建一个实例对象
7. print(f"{s1.name}的年龄初始为{s1.age}")
8. s1.age = 20                          #修改实例对象的属性age
9. print(f"修改{s1.name}的年龄为{s1.age}")
```

```
小千的年龄初始为19
修改小千的年龄为20
```

2 类的基本使用

2.4 修改属性的值-**通过方法修改属性的值**

在类中写一个方法，通过调用此方法，来修改属性的值

```
1.  class Student:
2.      def __init__(self,name,id):
3.          self.name = name
4.          self.id = id
5.          self.age = 19
6.      def update_age(self,age):
7.          """用于修改属性age的值"""
8.          self.age = age
9.  s1 = Student("小千",202201)          #创建实例对象
10. print(f"{s1.name}的年龄是{s1.age}")
11. s1.update_age(20)                   #调用修改年龄的方法
12. print(f"{s1.name}的年龄修改为{s1.age}")
```

```
小千的年龄是19
小千的年龄修改为20
```

8.5
实战12：人机猜拳游戏

8.5 实战12：人机猜拳游戏

开发人机猜拳游戏，将此需求分解一下，可以分解成**玩家的动作**、**机器的动作**以及**人和机器的互动**，即**游戏的过程**。玩家可以一直和机器进行游戏，当玩家想退出时则可以退出游戏。玩家赢得游戏，则玩家得一分；机器赢得游戏，则机器得一分。当游戏结束后，则统计总的猜拳次数，玩家和机器谁的分更高，谁就获得游戏的最终胜利。

8.5 实战12：人机猜拳游戏

1.玩家的动作

```
class Player:
    def __init__(self,score=0):
        self.name = "玩家"                          #玩家的姓名，默认为“玩家”
        self.score = score                          #玩家的得分
    def player_action(self, option):                #方法用于打印玩家猜拳的动作
        if option == 1:
            print(f"{self.name}出石头")
        elif option == 2:
            print(f"{self.name}出剪刀")
        elif option == 3:
            print(f"{self.name}出布")
```

8.5 实战12：人机猜拳游戏

2.机器的动作

```
class Computer:
    def __init__(self,score=0):
        self.cname = "电脑"                    #机器的姓名，默认为“电脑”
        self.score = score                     #机器的得分
    def computer_action(self,option):          #方法用于打印机器猜拳的动作
        if option == 1:
            print(f"{self.cname}出石头")
        elif option == 2:
            print(f"{self.cname}出剪刀")
        elif option == 3:
            print(f"{self.cname}出布")
```

8.5 实战12：人机猜拳游戏

3.游戏的过程

```
class PlayGame:
    count = 0                                   #对战次数
    def __init__(self):
        """每创建一个PlayGame的实例对象就会自动生成一个玩家和机器的实例"""
        self.player = Player()                  #创建一个玩家实例
        self.computer = Computer()              #创建一个机器实例
    def show_result(self):
        """
        用于展示最终的比赛结果
        1.打印比赛的场次
        2.打印玩家和机器的得分
        3.判断玩家和机器的胜利者是谁
        """
    def start_game(self):
        """
        用于展示每次玩家和机器的猜拳结果
        1.玩家选择是否开始游戏
        2.建立循环用于机器和玩家猜拳，玩家猜拳从键盘输入，机器猜拳由程序自动生成
        3.在循环中判断每一次猜拳中机器和玩家谁取得胜利
        判断过程：
        （1）当机器和玩家的猜拳相同则平局；
        （2）玩家是石头且机器是剪刀/玩家是剪刀且机器是布/玩家是布且机器是石头时，玩家获胜
        （3）其他情况下，机器获胜
4.结束循环时，调用show_result()方法打印最终结果
        """
```

导入模块中的类

导入模块中特定的类

导入模块中的所有类

8.6 导入模块中的类

8.6.1 导入模块中特定的类

导入模块中特定的类，需要给类创建对象进行使用。

导入并使用Player类

```
1. from game import Player        #从game.py模块中导入Player类
2. p1 = Player()                  #创建Player类的实例对象
3. print(p1.name)                 #访问实例属性
4. p1.player_action(1)            #访问实例方法
```

```
玩家
玩家出石头
```

8.6 导入模块中的类

8.6.1 导入模块中特定的类

不但可以导入模块中某个特定的类，还可以**导入多个特定的类**。

导入并使用Player、Computer类

```
1. from game import Player,Computer    #从game.py模块中导入Player、Computer类
2. p1 = Player()                       #创建Player类的实例对象
3. print(p1.name)
4. p1.player_action(1)
5. c1 = Computer()                     #创建Computer类的实例对象
6. print(c1.cname)
7. c1.computer_action(2)
```

```
玩家
玩家出石头
电脑
电脑出剪刀
```

8.6 导入模块中的类

8.6.2 导入模块中的所有类

导入模块中的所有类，其语法格式如下。

```
from 模块名 import *
```

当导入了模块中的所有类后，**使用类只需要通过类名**。这种导入方式有两个缺点，第一，这种导入方式**不能明确看出程序使用了哪些类**；第二，这种导入方式**不能确定哪些类属于这个模块**，如果导入了多个模块，**还会出现与其它变量同名的可能**。所以不建议通过这种导入方式导入模块中所有的类。

导入并使用Player类

```
1. from game import *      #导入game.py模块中的所有类
2. p1 = Player()           #创建Player类的实例对象
3. print(p1.name)
4. p1.player_action(1)
```

```
玩家
玩家出石头
```

8.6 导入模块中的类

8.6.2 导入模块中的所有类

导入模块中所有的类**可以通过导入整个模块的形式**，这种方式在使用类时，需要以“**模块.类名**”的形式，故而不会与其它模块或者文件中的名称发生冲突。

导入并使用Player类

```
1. import game              #导入整个game.py模块
2. p1 = game.Player()  #创建Player类的实例对象
3. print(p1.name)
4. p1.player_action(1)
```

```
玩家
玩家出石头
```

本章小结

本章小结

本章主要介绍了Python中类和对象，首先讲解了**类和对象的基本使用**，其次介绍了**类属性**，再次介绍了**类的方法**，最后介绍了**如何导入模块中的类**。以“人机猜拳游戏”讲解了类和对象的具体应用，是导入模块中的类的实际应用。