第七章 类——面向对象的编程

引子

Why：面向对象更符合人类对客观世界的抽象和理解

- **一切皆对象**
  一只小狗，一把椅子，一张信用卡，一条巧克力。。。

- **一切对象，都有自己内在的属性**
  狗狗的品种、椅子的质地、信用卡的额度、巧克力的口味。。。

- **一切行为，皆是对象的行为**
  狗狗蹲下、椅子移动位置、刷信用卡、巧克力融化了。。。

How：类是对象的载体

不同年龄、肤色、品质的猫，每一只都是一个对象

他们有一个共同的特征：都是猫

我们可以把一类对象的公共特征抽象出来，创建通用的类

In [2]:

```
# 创建类
class Cat():
    """模拟猫"""

    def __init__(self, name):
        """初始化属性"""
        self.name = name

    def jump(self):
        """模拟猫跳跃"""
        print(self.name + " is jumping")
```

In [3]:

```
# 用类创建实例
my_cat = Cat("Loser")
your_cat = Cat("Lucky")
```

In [4]:

```
# 调用属性
print(my_cat.name)
print(your_cat.name)
```

```
Loser
Lucky
```

In [5]:

```
# 调用方法
my_cat.jump()
your_cat.jump()
```

```
Loser is jumping
Lucky is jumping
```

7.1 类的定义

三要素：类名、属性、方法

7.1.1 类的命名

- 要有实际意义

- 驼峰命名法——组成的单词首字母大写
  Dog、 CreditCard、 ElectricCar

In [44]:

```
# class 类名:
"""类前空两行"""


class Car():
    """对该类的简单介绍"""
    pass

"""类后空两行"""
```

7.1.2 类的属性

In [46]:

```
# def __init__(self,要传递的参数)  初始化类的属性
```

In [47]:

```
class Car():
    """模拟汽车"""

    def __init__(self, brand, model, year):
        """初始化汽车属性"""                # 相当于类内部的变量
        self.brand = brand                  # 汽车的品牌
        self.model = model                  # 汽车的型号
        self.year = year                    # 汽车出厂年份
        self.mileage = 0                    # 新车总里程初始化为0
```

7.1.3 类的方法

In [1]:

```
# 相对于类内部定义的函数
```

In [22]:

```
class Car():
    """模拟汽车"""

    def __init__(self, brand, model, year):
        """初始化汽车属性"""                # 相当于类内部的变量
        self.brand = brand                  # 汽车的品牌
        self.model = model                  # 汽车的型号
        self.year = year                    # 汽车出厂年份
        self.mileage = 0                    # 新车总里程初始化为0

    def get_main_information(self):         # self不能省
        """获取汽车主要信息"""
        print("品牌：{}    型号：{}    出厂年份：{}".format(self.brand, self.model, self.year))

    def get_mileage(self):
        """获取总里程"""
        return "行车总里程：{}公里".format(self.mileage)
```

7.2 创建实例

7.2.1 实例的创建

将实例赋值给对象，实例化过程中，传入相应的参数
v = 类名（必要的初始化参数）

In [10]:

```
my_new_car = Car("Audi", "A6", 2018)
```

7.2.2 访问属性

实例名.属性名

In [11]:

```
print(my_new_car.brand)
print(my_new_car.model)
print(my_new_car.year)
```

```
Audi
A6
2018
```

7.2.3 调用方法

In [23]:

```
class Car():
    """模拟汽车"""

    def __init__(self, brand, model, year):
        """初始化汽车属性"""                    # 相当于类内部的变量
        self.brand = brand                      # 汽车的品牌
        self.model = model                      # 汽车的型号
        self.year = year                        # 汽车出厂年份
        self.mileage = 0                        # 新车总里程初始化为0

    def get_main_information(self):             # self不能省
        """获取汽车主要信息"""
        print("品牌：{}    型号：{}    出厂年份：{}".format(self.brand, self.model, self.year))

    def get_mileage(self):
        """获取总里程数"""
        return "行车总里程：{}公里".format(self.mileage)
```

实例名.方法名(必要的参数)

In [24]:

```
my_new_car = Car("Audi", "A6", 2018)
my_new_car.get_main_information()
```

```
品牌：Audi    型号：A6    出厂年份：2018
```

In [25]:

```
mileage = my_new_car.get_mileage()
print(mileage)
```

```
行车总里程：0公里
```

7.2.4 修改属性

1、直接修改

In [30]:

```
my_old_car = Car("BYD", "宋", 2016)
```

先访问，后修改

In [31]:

```
print(my_old_car.mileage)
my_old_car.mileage = 12000
print(my_old_car.mileage)
```

```
0
12000
```

In [33]:

```
print(my_old_car.get_mileage())
```

```
行车总里程：12000公里
```

2、通过方法修改属性

In [36]:

```
class Car():
    """模拟汽车"""

    def __init__(self, brand, model, year):
        """初始化汽车属性"""                # 相当于类内部的变量
        self.brand = brand                  # 汽车的品牌
        self.model = model                  # 汽车的型号
        self.year = year                    # 汽车出厂年份
        self.mileage = 0                    # 新车总里程初始化为0

    def get_main_information(self):         # self不能省
        """获取汽车主要信息"""
        print("品牌：{}    型号：{}    出厂年份：{}".format(self.brand, self.model, self.year))

    def get_mileage(self):
        """获取总里程数"""
        return "行车总里程：{}公里".format(self.mileage)

    def set_mileage(self, distance):
        """设置总里程数"""
        self.mileage = distance
```

In [37]:

```
my_old_car = Car("BYD", "宋", 2016)
print(my_old_car.get_mileage())
my_old_car.set_mileage(8000)
print(my_old_car.get_mileage())
```

```
行车总里程：0公里
行车总里程：8000公里
```

3、继续拓展

- 禁止设置负里程

In [42]:

```
class Car():
    """模拟汽车"""

    def __init__(self, brand, model, year):
        """初始化汽车属性"""                # 相当于类内部的变量
        self.brand = brand                  # 汽车的品牌
        self.model = model                  # 汽车的型号
        self.year = year                    # 汽车出厂年份
        self.mileage = 0                    # 新车总里程初始化为0

    def get_main_information(self):         # self不能省
        """获取汽车主要信息"""
        print("品牌：{}    型号：{}    出厂年份：{}".format(self.brand, self.model, self.year))

    def get_mileage(self):
        """获取总里程数"""
        print("行车总里程：{}公里".format(self.mileage))

    def set_mileage(self, distance):
        """设置总里程数"""
        if distance >= 0:
            self.mileage = distance
        else:
            print("里程数不能为负！")

    def increment_mileage(self, distance):
        """总里程数累计"""
        if distance >= 0:
            self.mileage += distance
        else:
            print("新增里程数不能为负！")

```

In [43]:

```
my_old_car = Car("BYD", "宋", 2016)
my_old_car.get_mileage()
my_old_car.set_mileage(-8000)
my_old_car.get_mileage()
```

```
行车总里程：0公里
里程数不能为负！
行车总里程：0公里
```

- 将每次的里程数累加

In [44]:

```
my_old_car.get_mileage()
my_old_car.set_mileage(8000)
my_old_car.get_mileage()
my_old_car.increment_mileage(500)
my_old_car.get_mileage()
```

```
行车总里程：0公里
行车总里程：8000公里
行车总里程：8500公里
```

小结

In [83]:

```
my_new_car = Car("Audi", "A6", 2018)
my_cars = [my_new_car, my_old_car]
```

- 包含的信息量可以是极大的，可以创建无穷多的实例
- 高度的拟人（物）化，符合人类对客观世界的抽象和理解

7.3 类的继承

引子

看一下人在生物界的分支链

生物——动物界——脊索动物门——哺乳动物纲——灵长目——人科——人属——智人种

公共特征逐渐增加的过程

【问题】

假设二元系统： 人属 = {A人种， B人种， C人种。。。。}

为每一个人种构造一个类

方案一： 各自独立，分别构造各自人种的类

方案二：
1、将各人种**公共特征提取出来**，建立人属的类；
2、各人种**继承上一级（人属）的公共特征，然后添加自身特殊特征**，构建各自人种的类。

通常，我们选择方案二，因为他避免了过多的重复劳动

所谓继承，就是低层抽象继承高层抽象的过程

7.3.1 简单的继承

父类

In [56]:

```
class Car():
    """模拟汽车"""

    def __init__(self, brand, model, year):
        """初始化汽车属性"""                # 相当于类内部的变量
        self.brand = brand                  # 汽车的品牌
        self.model = model                  # 汽车的型号
        self.year = year                    # 汽车出厂年份
        self.mileage = 0                    # 新车总里程初始化为0


    def get_main_information(self):         # self不能省
        """获取汽车主要信息"""
        print("品牌：{}    型号：{}    出厂年份：{}".format(self.brand, self.model, self.year))

    def get_mileage(self):
        """获取总里程数"""
        print("行车总里程：{}公里".format(self.mileage))

    def set_mileage(self, distance):
        """设置总里程数"""
        if distance >= 0:
            self.mileage = distance
        else:
            print("里程数不能为负！")

    def increment_mileage(self, distance):
        """总里程数累计"""
        if distance >= 0:
            self.mileage += distance
        else:
            print("新增里程数不能为负！")
```

子类

class 子类名（父类名）：

- 新建一个电动汽车的类

In [57]:

```
class ElectricCar(Car):
    """模拟电动汽车"""

    def __init__(self, brand, model, year):
        """初始化电动汽车属性"""
        super().__init__(brand, model, year)  # 声明继承父类的属性
```

- 自动继承父类的所有方法

In [62]:

```
my_electric_car = ElectricCar("NextWeek", "FF91", 2046)
my_electric_car.get_main_information()
```

```
品牌：NextWeek    型号：FF91    出厂年份：2046
```

7.3.2 给子类添加属性和方法

In [63]:

```
class ElectricCar(Car):
    """模拟电动汽车"""

    def __init__(self, brand, model, year, bettery_size):
        """初始化电动汽车属性"""
        super().__init__(brand, model, year)     # 声明继承父类的属性
        self.bettery_size = bettery_size         # 电池容量
        self.electric_quantity = bettery_size    # 电池剩余电量
        self.electric2distance_ratio = 5         # 电量距离换算系数 5公里/kW.h
        self.remainder_range = self.electric_quantity*self.electric2distance_ratio # 剩余可行驶

    def get_electric_quantit(self):
        """查看当前电池电量"""
        print("当前电池剩余电量：{} kW.h".format(self.electric_quantity))

    def set_electric_quantity(self, electric_quantity):
        """设置电池剩余电量，重新计算电量可支撑行驶里程"""
        if electric_quantity >= 0 and electric_quantity <= self.bettery_size:
            self.electric_quantity = electric_quantity
            self.remainder_range = self.electric_quantity*self.electric2distance_ratio
        else:
            print("电量未设置在合理范围！")

    def get_remainder_range(self):
        """查看剩余可行驶里程"""
        print("当前电量还可以继续驾驶 {} 公里".format(self.remainder_range))
```

In [64]:

```
my_electric_car = ElectricCar("NextWeek", "FF91", 2046, 70)
my_electric_car.get_electric_quantit()              # 获取当前电池电量
my_electric_car.get_remainder_range()               # 获取当前剩余可行驶里程
```

```
当前电池剩余电量：70 kW.h
当前电量还可以继续驾驶 350 公里
```

In [65]:

```
my_electric_car.set_electric_quantity(50)           # 重设电池电量
my_electric_car.get_electric_quantit()              # 获取当前电池电量
my_electric_car.get_remainder_range()               # 获取当前剩余可行驶里程
```

```
当前电池剩余电量：50 kW.h
当前电量还可以继续驾驶 250 公里
```

7.3.3 重写父类的方法——多态

In [67]:

```
class ElectricCar(Car):
    """模拟电动汽车"""

    def __init__(self, brand, model, year, bettery_size):
        """初始化电动汽车属性"""
        super().__init__(brand, model, year)     # 声明继承父类的属性
        self.bettery_size = bettery_size         # 电池容量
        self.electric_quantity = bettery_size    # 电池剩余电量
        self.electric2distance_ratio = 5         # 电量距离换算系数 5公里/kW.h
        self.remainder_range = self.electric_quantity*self.electric2distance_ratio # 剩余可行驶

    def get_main_information(self):         # 重写父类方法
        """获取汽车主要信息"""
        print("品牌：{}    型号：{}    出厂年份：{}    续航里程：{} 公里"
              .format(self.brand, self.model, self.year, self.bettery_size*self.electric2distan

    def get_electric_quantit(self):
        """查看当前电池电量，重新计算电量可支撑行驶里程"""
        print("当前电池剩余电量：{} kW.h".format(self.electric_quantity))

    def set_electric_quantity(self, electric_quantity):
        """设置电池剩余电量"""
        if electric_quantity >= 0 and electric_quantity <= self.bettery_size:
            self.electric_quantity = electric_quantity
            self.remainder_range = self.electric_quantity*self.electric2distance_ratio
        else:
            print("电量未设置在合理范围！")

    def get_remainder_range(self):
        """查看剩余可行驶里程"""
        print("当前电量还可以继续驾驶 {} 公里".format(self.remainder_range))
```

In [69]:

```
my_electric_car = ElectricCar("NextWeek", "FF91", 2046, 70)
my_electric_car.get_main_information()
```

品牌：NextWeek　型号：FF91　出厂年份：2046　续航里程：350 公里

7.3.4 用在类中的实例

把电池抽象成一个对象
逻辑更加清晰

In [70]:

```
class Bettery():
    """模拟电动汽车的电池"""

    def __init__(self, bettery_size = 70):
        self.bettery_size = bettery_size        # 电池容量
        self.electric_quantity = bettery_size   # 电池剩余电量
        self.electric2distance_ratio = 5        # 电量距离换算系数 5公里/kW.h
        self.remainder_range = self.electric_quantity*self.electric2distance_ratio # 剩余可行驶

    def get_electric_quantit(self):
        """查看当前电池电量"""
        print("当前电池剩余电量：{} kW.h".format(self.electric_quantity))

    def set_electric_quantity(self, electric_quantity):
        """设置电池剩余电量，计重新算电量可支撑行驶里程"""
        if electric_quantity >= 0 and electric_quantity <= self.bettery_size:
            self.electric_quantity = electric_quantity
            self.remainder_range = self.electric_quantity*self.electric2distance_ratio
        else:
            print("电量未设置在合理范围！")

    def get_remainder_range(self):
        """查看剩余可行驶里程"""
        print("当前电量还可以继续驾驶 {} 公里".format(self.remainder_range))
```

In [71]:

```
class ElectricCar(Car):
    """模拟电动汽车"""

    def __init__(self, brand, model, year, bettery_size):
        """初始化电动汽车属性"""
        super().__init__(brand, model, year)     # 声明继承父类的属性
        self.bettery = Bettery(bettery_size)     # 电池

    def get_main_information(self):         # 重写父类方法
        """获取汽车主要信息"""
        print("品牌：{}　型号：{}　出厂年份：{}　续航里程：{} 公里"
              .format(self.brand, self.model, self.year,
              self.bettery.bettery_size*self.bettery.electric2distance_ratio))
```

In [72]:

```
my_electric_car = ElectricCar("NextWeek", "FF91", 2046, 70)
my_electric_car.get_main_information()                    # 获取车辆主要信息
```

品牌：NextWeek 型号：FF91 出厂年份：2046 续航里程：350 公里

In [73]:

```
my_electric_car.bettery.get_electric_quantit()            # 获取当前电池电量
```

当前电池剩余电量：70 kW.h

In [74]:

```
my_electric_car.bettery.set_electric_quantity(50)         # 重设电池电量
```

In [75]:

```
my_electric_car.bettery.get_electric_quantit()            # 获取当前电池电量
```

当前电池剩余电量：50 kW.h

In [76]:

```
my_electric_car.bettery.get_remainder_range()             # 获取当前剩余可行驶里程
```

当前电量还可以继续驾驶 250 公里