

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]:

```
1  # 创建类
2  class Cat():
3      """模拟猫"""
4
5      def __init__(self, name):
6          """初始化属性"""
7          self.name = name
8
9      def jump(self):
10         """模拟猫跳跃"""
11         print(self.name + " is jumping")
```

In [3]:

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

In [4]:

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

Loser

Lucky

In [5]:

```
1 # 调用方法
2 my_cat.jump()
3 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]:

```
1 # class 类名:
2 """类前空两行"""
3
4
5 class Car():
6     """对该类的简单介绍"""
7     pass
8
9 """类后空两行"""
```

7.1.2 类的属性

In [46]:

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

In [47]:

```
1 class Car():
2     """模拟汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化汽车属性"""          # 相当于类内部的变量
6         self.brand = brand              # 汽车的品牌
7         self.model = model               # 汽车的型号
8         self.year = year                 # 汽车出厂年份
9         self.mileage = 0                 # 新车总里程初始化为0
```

7.1.3 类的方法

In [1]:

```
1 # 相对于类内部定义的函数
```

In [22]:

```
1 class Car():
2     """模拟汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化汽车属性"""          # 相当于类内部的变量
6         self.brand = brand              # 汽车的品牌
7         self.model = model               # 汽车的型号
8         self.year = year                 # 汽车出厂年份
9         self.mileage = 0                 # 新车总里程初始化为0
10
11     def get_main_information(self):      # self不能省
12         """获取汽车主要信息"""
13         print("品牌: {}  型号: {}  出厂年份: {}".format(self.brand, self.model, self.year))
14
15     def get_mileage(self):
16         """获取总里程"""
17         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]:

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

7.2.2 访问属性

实例名.属性名

In [11]:

```
1 print(my_new_car.brand)
2 print(my_new_car.model)
3 print(my_new_car.year)
```

Audi

A6

2018

7.2.3 调用方法

In [23]:

```
1 class Car():
2     """模拟汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化汽车属性"""
6         self.brand = brand          # 相当于类内部的变量
7         self.model = model          # 汽车的品牌
8         self.year = year            # 汽车的型号
9         self.mileage = 0            # 汽车出厂年份
10                                     # 新车总里程初始化为0
11
12     def get_main_information(self):  # self不能省
13         """获取汽车主要信息"""
14         print("品牌: {} 型号: {} 出厂年份: {}".format(self.brand, self.model, self.year))
15
16     def get_mileage(self):
17         """获取总里程数"""
18         return "行车总里程: {}公里".format(self.mileage)
```

实例名.方法名(必要的参数)

In [24]:

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

品牌: Audi 型号: A6 出厂年份: 2018

In [25]:

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

行车总里程: 0公里

7.2.4 修改属性

1、直接修改

In [30]:

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

先访问，后修改

In [31]:

```
1 print(my_old_car.mileage)
2 my_old_car.mileage = 12000
3 print(my_old_car.mileage)
```

0
12000

In [33]:

```
1 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]:

```
1 class Car():
2     """模拟汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化汽车属性"""          # 相当于类内部的变量
6         self.brand = brand              # 汽车的品牌
7         self.model = model              # 汽车的型号
8         self.year = year                # 汽车出厂年份
9         self.mileage = 0                # 新车总里程初始化为0
10
11     def get_main_information(self):      # self不能省
12         """获取汽车主要信息"""
13         print("品牌: {}  型号: {}  出厂年份: {}".format(self.brand, self.model, self.year))
14
15     def get_mileage(self):
16         """获取总里程数"""
17         return "行车总里程: {} 公里".format(self.mileage)
18
19     def set_mileage(self, distance):
20         """设置总里程数"""
21         self.mileage = distance
```

In [37]:

```
1 my_old_car = Car("BYD", "宋", 2016)
2 print(my_old_car.get_mileage())
3 my_old_car.set_mileage(8000)
4 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]:

```
1 class Car():
2     """模拟汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化汽车属性"""          # 相当于类内部的变量
6         self.brand = brand              # 汽车的品牌
7         self.model = model              # 汽车的型号
8         self.year = year                # 汽车出厂年份
9         self.mileage = 0                # 新车总里程初始化为0
10
11     def get_main_information(self):      # self不能省
12         """获取汽车主要信息"""
13         print("品牌：{}  型号：{}  出厂年份：{}".format(self.brand, self.model, self.year))
14
15     def get_mileage(self):
16         """获取总里程数"""
17         print("行车总里程：{}公里".format(self.mileage))
18
19     def set_mileage(self, distance):
20         """设置总里程数"""
21         if distance >= 0:
22             self.mileage = distance
23         else:
24             print("里程数不能为负！")
25
26     def increment_mileage(self, distance):
27         """总里程数累计"""
28         if distance >= 0:
29             self.mileage += distance
30         else:
31             print("新增里程数不能为负！")
32
```

In [43]:

```
1 my_old_car = Car("BYD", "宋", 2016)
2 my_old_car.get_mileage()
3 my_old_car.set_mileage(-8000)
4 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]:

```
1 my_old_car.get_mileage()
2 my_old_car.set_mileage(8000)
3 my_old_car.get_mileage()
4 my_old_car.increment_mileage(500)
5 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]:

```
1 my_new_car = Car("Audi", "A6", 2018)
2 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。。。}

为每一个种族构造一个类

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]:

```
1 class Car():
2     """模拟汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化汽车属性"""           # 相当于类内部的变量
6         self.brand = brand               # 汽车的品牌
7         self.model = model               # 汽车的型号
8         self.year = year                 # 汽车出厂年份
9         self.mileage = 0                 # 新车总里程初始化为0
10
11
12     def get_main_information(self):       # self不能省
13         """获取汽车主要信息"""
14         print("品牌: {} 型号: {} 出厂年份: {}".format(self.brand, self.model, self.year))
15
16     def get_mileage(self):
17         """获取总里程数"""
18         print("行车总里程: {} 公里".format(self.mileage))
19
20     def set_mileage(self, distance):
21         """设置总里程数"""
22         if distance >= 0:
23             self.mileage = distance
24         else:
25             print("里程数不能为负！")
26
27     def increment_mileage(self, distance):
28         """总里程数累计"""
29         if distance >= 0:
30             self.mileage += distance
31         else:
32             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]:

```
1 class ElectricCar(Car):
2     """模拟电动汽车"""
3
4     def __init__(self, brand, model, year):
5         """初始化电动汽车属性"""
6         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]:

```
1 my_electric_car = ElectricCar("NextWeek", "FF91", 2046)
2 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]:

```
1 class ElectricCar(Car):
2     """模拟电动汽车"""
3
4     def __init__(self, brand, model, year, bettery_size):
5         """初始化电动汽车属性"""
6         super().__init__(brand, model, year) # 声明继承父类的属性
7         self.bettery_size = bettery_size # 电池容量
8         self.electric_quantity = bettery_size # 电池剩余电量
9         self.electric2distance_ratio = 5 # 电量距离换算系数 5公里/kW.h
10        self.remainder_range = self.electric_quantity*self.electric2distance_ratio # 剩余可行驶
11
12    def get_electric_quantit(self):
13        """查看当前电池电量"""
14        print("当前电池剩余电量: {} kW.h".format(self.electric_quantity))
15
16    def set_electric_quantity(self, electric_quantity):
17        """设置电池剩余电量, 重新计算电量可支撑行驶里程"""
18        if electric_quantity >= 0 and electric_quantity <= self.bettery_size:
19            self.electric_quantity = electric_quantity
20            self.remainder_range = self.electric_quantity*self.electric2distance_ratio
21        else:
22            print("电量未设置在合理范围!")
23
24    def get_remainder_range(self):
25        """查看剩余可行驶里程"""
26        print("当前电量还可以继续驾驶 {} 公里".format(self.remainder_range))
```

In [64]:

```
1 my_electric_car = ElectricCar("NextWeek", "FF91", 2046, 70)
2 my_electric_car.get_electric_quantit()          # 获取当前电池电量
3 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]:

```
1 my_electric_car.set_electric_quantity(50)       # 重设电池电量
2 my_electric_car.get_electric_quantit()          # 获取当前电池电量
3 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]:

```
1 class ElectricCar(Car):
2     """模拟电动汽车"""
3
4     def __init__(self, brand, model, year, bettery_size):
5         """初始化电动汽车属性"""
6         super().__init__(brand, model, year)    # 声明继承父类的属性
7         self.bettery_size = bettery_size        # 电池容量
8         self.electric_quantity = bettery_size   # 电池剩余电量
9         self.electric2distance_ratio = 5        # 电量距离换算系数 5公里/kW.h
10        self.remainder_range = self.electric_quantity*self.electric2distance_ratio # 剩余可行驶
11
12    def get_main_information(self):               # 重写父类方法
13        """获取汽车主要信息"""
14        print("品牌: {} 型号: {} 出厂年份: {} 续航里程: {} 公里"
15              .format(self.brand, self.model, self.year, self.bettery_size*self.electric2distan
16
17    def get_electric_quantit(self):
18        """查看当前电池电量, 重新计算电量可支撑行驶里程"""
19        print("当前电池剩余电量: {} kW.h".format(self.electric_quantity))
20
21    def set_electric_quantity(self, electric_quantity):
22        """设置电池剩余电量"""
23        if electric_quantity >= 0 and electric_quantity <= self.bettery_size:
24            self.electric_quantity = electric_quantity
25            self.remainder_range = self.electric_quantity*self.electric2distance_ratio
26        else:
27            print("电量未设置在合理范围!")
28
29    def get_remainder_range(self):
30        """查看剩余可行驶里程"""
31        print("当前电量还可以继续驾驶 {} 公里".format(self.remainder_range))
```

In [69]:

```
1 my_electric_car = ElectricCar("NextWeek", "FF91", 2046, 70)
2 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]:

```
1 class Battery():
2     """模拟电动汽车的电池"""
3
4     def __init__(self, battery_size = 70):
5         self.battery_size = battery_size      # 电池容量
6         self.electric_quantity = battery_size # 电池剩余电量
7         self.electric2distance_ratio = 5      # 电量距离换算系数 5公里/kW.h
8         self.remainder_range = self.electric_quantity*self.electric2distance_ratio # 剩余可行驶
9
10    def get_electric_quantit(self):
11        """查看当前电池电量"""
12        print("当前电池剩余电量: {} kW.h".format(self.electric_quantity))
13
14    def set_electric_quantity(self, electric_quantity):
15        """设置电池剩余电量，计重新算电量可支撑行驶里程"""
16        if electric_quantity >= 0 and electric_quantity <= self.battery_size:
17            self.electric_quantity = electric_quantity
18            self.remainder_range = self.electric_quantity*self.electric2distance_ratio
19        else:
20            print("电量未设置在合理范围!")
21
22    def get_remainder_range(self):
23        """查看剩余可行驶里程"""
24        print("当前电量还可以继续驾驶 {} 公里".format(self.remainder_range))
```

In [71]:

```
1 class ElectricCar(Car):
2     """模拟电动汽车"""
3
4     def __init__(self, brand, model, year, battery_size):
5         """初始化电动汽车属性"""
6         super().__init__(brand, model, year) # 声明继承父类的属性
7         self.battery = Battery(battery_size) # 电池
8
9     def get_main_information(self):          # 重写父类方法
10        """获取汽车主要信息"""
11        print("品牌: {} 型号: {} 出厂年份: {} 续航里程: {} 公里"
12              .format(self.brand, self.model, self.year,
13                    self.battery.battery_size*self.battery.electric2distance_ratio))
```

In [72]:

```
1 my_electric_car = ElectricCar("NextWeek", "FF91", 2046, 70)
2 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]:

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

当前电池剩余电量: 70 kW.h

In [74]:

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

In [75]:

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

当前电池剩余电量: 50 kW.h

In [76]:

```
1 my_electric_car.battery.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