16. 核能来袭-成员

本节主要内容:

1. 类的成员
2. 类的成员-变量
3. 类的成员-方法
4. 类的成员-属性
5. 私有

一. 类的成员:

首先, 什么是类的成员. 很简单. 你能在类中写什么? 写的内容就是成员. 到目前为止. 我们已经学过了一些成员了.

```
class 类名:
    # 方法
    def __init__(self, 参数1, 参数2....):
        # 属性变量
        self.属性1 = 参数1
        self.属性2 = 参数2
        ....
    # 方法
    def method(self):
        pass
```

在上面代码中__init__和method都属于类的成员方法. 又称为实例方法. 总之这样的东西需要用对象来访问. 而上方的self.属性1 = 参数1 这个代码的含义是给对象设置属性信息. 含义是这个对象的xxx属性是xxxx. 这种东西又被称之为成员变量或者实例变量, 再或者被称之为字段. 都是一个意思.

也就是说在类中, 是存在着实例变量和实例方法的. 还有哪些呢? 一个一个的看.

二. 类的成员-变量

在类中变量分成两大类:

1. 实例变量(字段)
2. 类变量(静态变量)

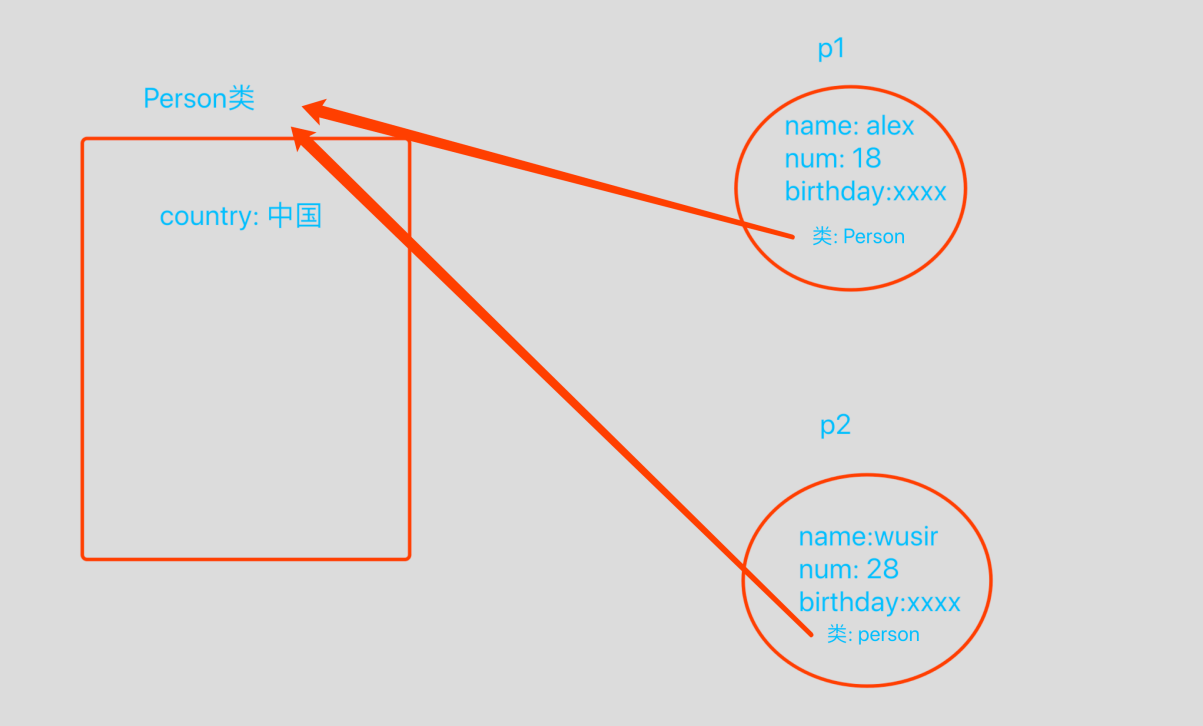
先说什么是实例变量. 说白了. 就是每个实例都应该拥有的变量. 比如. 人的名字, 人的爱好, 每个人的个人信息. 都属于实例变量. 那什么是类变量. 就是这一类事物统一拥有的变量. 比如. 在座的各位都是中国人. 那大家都拥有同一个国家. 例:

```python
class Person:
    # 类变量, 表示所有的该类的对象都共享这个变量.
    country = "中国"

    def __init__(self, name, num, birthday):
        # 实例变量(字段) 表示你创建的每一个人都有这三个变量
        self.name = name
        self.num = num
        self.birthday = birthday

p1 = Person("alex", 18, "1840年06月01日")
print(p1.name)        # alex
print(p1.country)     #中国


p2 = Person("wusir", 28, "1894年07月25日")
print(p2.name)        # wusir
print(p2.country)     # 中国
```

我们发现对象p1和p2的name都是对象自己的. 但是country是类的. 大家公用同一个变量. 如何来验证呢?

```python
Person.country = "大清" # 在这里. 我把国家改成了大清
p1 = Person("alex", 18, "1840年06月01日")
print(p1.name)
print(p1.country)    # alex是大清的


p2 = Person("wusir", 28, "1894年07月25日")
print(p2.name)
print(p2.country)    # wusir也是大清的
```

发现了吧. 我把类变量中的值改了. 大家看到的都跟着变了. 为什么呢? 我们要从内存来分析. 看图.

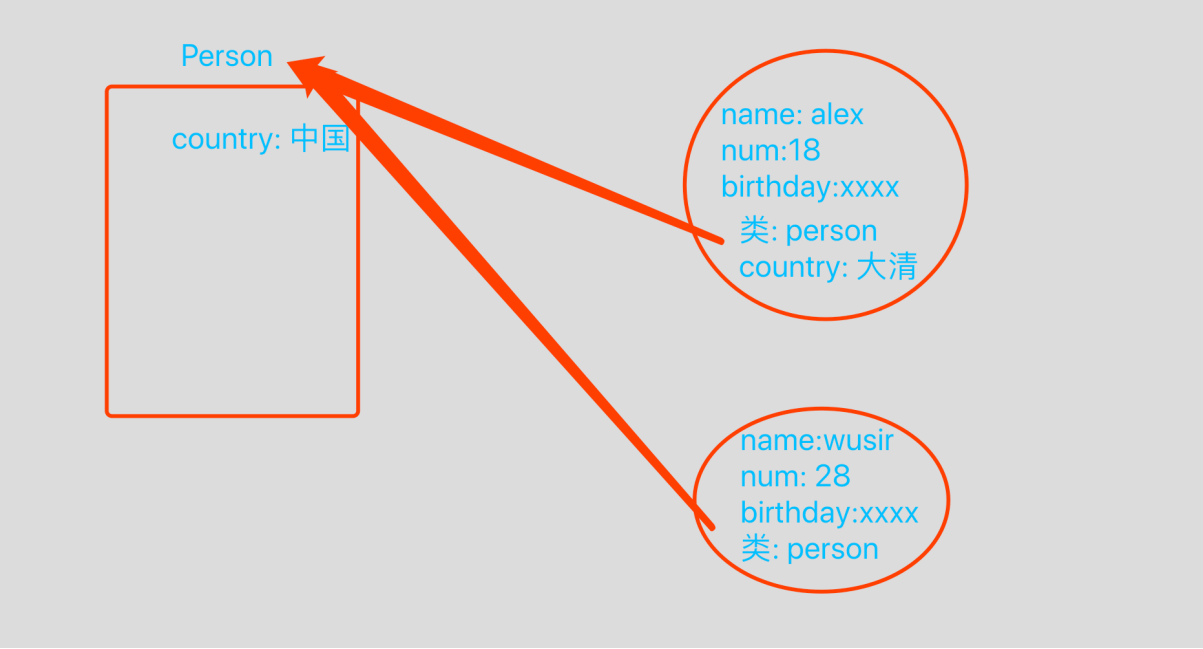


通过图我们能发现, 实例变量中都隐含着一个创建这个对象的类. 通过这个类就能找到我们类中定义的全部内容, 包括方法和属性信息等.

接下来. 我们来看一个和类变量息息相关的坑.

```
p1 = Person("alex", 18, "1840年06月01日")
p1.country = "大清"
print(p1.name)
print(p1.country)    # 大清


p2 = Person("wusir", 28, "1894年07月25日")
print(p2.name)
print(p2.country)    # 中国
```

上面是大清. 下面是中国. 为什么会这样呢? 注意. 在执行p1.country = "大清"的时候. 其实并没有去改变类中的country, 而是给对象添加了一个实例变量. 并且这个实例变量,只有当前的p1是存在的. 在p2中是不存在的. 依然画个图来看.



在图中我们能清晰的看到. 我们其实并没有改变类变量. 只是在p1中添加了一个实例变量, 仅此而已. 通过这里. 我们应该能发现. 类变量, 最好是用类名来访问. 当然, 我们通过对象名也可以访问. 但只能看, 不能改变它. 想要改变它, 需要用类名来改变它.

案例. 通过程序来记录当前类被创建了多少个对象.

```
class Foo:
    count = 0
    def __init__(self):
        Foo.count += 1


print(Foo.count)    # 0
Foo()
Foo()
Foo()
print(Foo.count)    # 3
```

好了. 来做个简单的总结:

实例变量, 给对象用的.

类变量, 多个对象共享的. 最好是用类名来访问. 这样更加规范.

二. 类的成员-方法

1. 成员方法(实例方法)
2. 静态方法
3. 类方法

先说第一个成员方法. 我们用的最多的就是这种. 昨天写的代码中, 所有的方法都可以被称之为成员方法. 说白了就是对象直接访问的方法叫成员方法.

```
class Computer:

    # 实例方法(成员方法)
    def play(self):
        print("我的电脑可以玩儿")

c = Computer()
c.play()    # 对象直接去调用实例方法
```

静态方法. 静态方法不需要我们给方法传递self. 也就是说. 当出现一个方法不需要使用到成员变量的时候. 就可以选择使用静态方法. 静态方法需要我们在方法上面添加一个@staticmethod

```
@staticmethod
def fare():
    print("我的电脑非常牛B, 可以煎鸡蛋")
```

静态方法和静态变量一样. 一般都是使用类名直接访问和调用的.

```
Computer.fare() # 类名可以直接访问的
c.fare()    # 对象也可以访问. 但最好不要这么干. 以便于区分静态方法和实例方法
```

类方法. 类方法和静态方法差不多, 只不过类方法需要在参数列表中的第一个位置预留一个位置, 通常我们给第一个参数起名字叫cls. 类方法在被调用的时候也不需要传递实例对象. 但是. 系统会自动的把类传递给第一个参数. 类方法在编写的时候, 需要在类方法上面添加@classmethod

```
class Computer:

    def play(self):
        print("我的电脑可以玩儿")

    @staticmethod
    def fare():
        print("我的电脑非常牛B, 可以煎鸡蛋")

    @classmethod
    def cal(cls, a, b):
        print(cls)
        return a+b

print(Computer.cal(1, 2)) # 此时会自动的把类名传递给类方法的第一个参数
```

面试题: 类方法/静态方法和实例方法有什么区别?

三. 类的成员-属性

属性其实就是通过方法改造过来的一种变量的写法, 在方法上添加一个@property就可以了

```
class Person:
    def __init__(self):
        pass

    @property
    def age(self):
        return 1


p = Person()
age = p.age

print(age)
```

应用场景: 我们一般保存数据的时候, 不会保存一个人的年龄. 因为随着时间的推移. 每个人的年龄都时刻在改变着. 那如何保存更加完美呢? 很简单. 保存出生年月日. 然后用程序来计算,你当前的年龄. 实时的. 那这个时候就需要进行相应的计算了. 而计算属于一个功能. 当然要写方法里了. 但是对于年龄这个属性而言. 他应该是一个数值. 而不是动作. 所以python就提供了这样一种机制. 通过方法来描述一个属性.

注意:

1. 方法参数只能有一个self
2. 方法上方要写@property
3. 调用的时候, 我们不需要写括号. 直接当成属性变量来用就可以了.
4. 这种套路只能取值. 不能设置值

四. 私有

在python中, 一般是不提倡设置和使用私有信息的. 但有些场景, 我们不得不这么做. 比如, 在一个公司. 每个人的收入情况, 这种内容是绝对不能公开的. 还有, 你的老婆, 也是一个私有的. 只能你一个人使用. 别人不能碰. 碰了就炸锅了.

在python中使用__作为方法或者变量的前缀. 那么这个方法或者变量就是一个私有的.

1. 私有变量

```
class Person:

    def __init__(self, laopo, mimi):
        self.__laopo = laopo    # 私有的
        self.__mimi = mimi
```

```
alex = Person("wusir", "他俩搞基")
print(alex.__mimi) # 私有的. 谁都不能碰
```

程序报错. 私有的内容是访问不到的. 但是, 这个访问不到是有一定的局限性的. 比如:

```
class Person:

    def __init__(self, laopo, mimi):
        self.__laopo = laopo    # 私有的
        self.__mimi = mimi

    def gaosuni(self):
        print("大喇叭开始广播了")
        return self.__mimi    # 私有的内容.在他自己那里, 他可以任意的进行使用


alex = Person("wusir", "他俩搞基")
mimi = alex.gaosuni()    # 通过一个非私有的方法, 访问到了他的秘密.
print(mimi)
```

记住, 私有的内容不能直接访问. 但是如果对方开辟了外界访问的通道(公共方法). 那可以通过这个公共的方法来获取到私有的内容. 这样做的好处是. 外界, 只能看, 但是改不了.
不单单实例变量有私有的. 类变量(静态变量)一样拥有这样的属性:

```
class Person:
    __zisi = "人都是自私的" # 人都是自私的. 但是这个自私又不希望别人知道
    def __init__(self, laopo, mimi):
        self.__laopo = laopo    # 私有的
        self.__mimi = mimi

    def gaosuni(self):
        print("大喇叭开始广播了")
        return self.__mimi


alex = Person("wusir", "他俩搞基")
mimi = alex.gaosuni()
print(mimi)
print(Person.__zisi)    # 报错
```

2. 私有方法

私有方法, 顾名思义, 只能自己访问的方法. 别人都不能随便调用的. 这个更好理解. 你和你女朋友约会. 你希望别人来调用么? 肯定不啊.

```
class Person:

    def __init__(self):
        pass
```

```
    def __yue(self):
        print("我要约会")

    def job(self):
        print("我要工作")
p = Person()
# p.__yue() # 报错
p.job()
```

__yue是一个私有的方法. 只能在类中自己调用. 类外面不能访问.

job是一个成员方法. 并且是一个开放的方法. 在类外界可以被访问到

同样的. 类中的私有方法也是相对而言的. 我们可以通过其他方法来访问到这样的方法.

```
class Person:

    def __init__(self):
        pass

    def __yue(self):
        print("我要约会")


    def job(self):
        print("我要工作")
        self.__yue()    # 在自己类中访问自己的其他方法. 哪怕是私有的. 也是自己在用


p = Person()
p.job()
```

关于类方法和静态方法, 和成员方法一样, 就不再赘述了

需要注意的是, 对于私有的内容而言. 子类是无法继承的.

```
class Fu:
    __qingfu = "情妇_小潘潘"

class Zi(Fu):
    pass

print(Zi.__qingfu)  # 报错
```