property的使用

- 就是一种特殊的属性，访问的时候会执行一段功能（函数）然后会返回值。

```
"""
例一：BMI指数（bmi是计算而来的，但很明显它听起来像是一个属性而非方法，
如果我们将其做成一个属性，更便于理解）
成人的BMI数值：
过轻：低于18.5
正常：18.5-23.9
过重：24-27
肥胖：28-32
非常肥胖，高于32
体质指数（BMI）=体重（kg）÷身高^2（m）
EX: 70KG / (1.75 * 1.75) = 22.86
"""


class People(object):
    def __init__(self, name, weight, height):
        # self.xxx(变量名) = xxx(值)
        self.name = name
        self.weight = weight
        self.height = height

    @property
    def bmi(self):
        return self.weight / (self.height ** 2)


p1 = People("小红", 50, 1.60)
p2 = People("小黑", 60, 1.70)
p3 = People("小刚", 70, 1.80)
# p1.bmi = p1.weight / (p1.height ** 2)
# print(p1.bmi)


# bmi是人的属性
# print(p1.bmi())
# print(p2.bmi())
```

```
# print(p3.bmi())
# 现在能够实现访问bmi指数
# 我们现在是访问的方法（函数）
print(p1.bmi)


# 将一个类的函数定义成特性以后，对象再去使用的时候   obj.name
# 根本无法察觉自己的name是执行了一个函数
# 遵循了统一访问的原则

# 把需要计算才能得到的属性，把他封装成访问数据属性去访问一样
```

绑定方法与非绑定方法

绑定方法

- 绑定给谁，就由谁来调用，谁来调用就把他当作一个参数自动传入
  - 对象的绑定方法（默认）
  - 在python中，凡是类中的方法默认情况都是绑定给对象使用的。

```
class People:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def tall_info(self):
        pass

obj = People('精神小伙', 18)
print(obj.tall_info)
```

- 类中tall_info方法是绑定给对象使用的

```
class People:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def tall_info(self):
        print('1223')

```

```python
obj = People('精神小伙', 18)

# print(People.tall_info)
# print(obj.tall_info)

People.tall_info(obj)
obj.tall_info()

# <function People.tall_info at 0x0000015F69D85840>  # 用类去调用是当作普通函数使用
# <bound method People.tall_info of <__main__.People object at 0x0000015F69BEA5C0>>
# 而对象去调用则为绑定方法
```

- 以上的情况是绑定给对象的
- 类的绑定方法
  - 既然Python默认类中的方法或函数，都是绑定给对象使用的。那么类中的绑定方法怎么解除和对象的绑定关系，进而绑定到类呢？这个时候需要用到：@classmethod方法

```python
class People:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @classmethod
    def tall_info(cls):  # cls和self都是一种命名习惯，cls作为第一个参数来表示类本身
        return cls("精神小伙", 18)   #  People("精神小伙", 18)

obj = People.tall_info()  # obj = People("精神小伙", 18)

obj.tall_info()


# print(People.tall_info)
# print(obj.tall_info)

# 根据上述结果所知：
# 和对象绑定方法一样：绑定给类，就由类来调用，并将类作为第一个参数传入。
# 和对象绑定方法不同：当对象在调用类的绑定方法时，也会默认把类当作参数传递进去。
```

```
# cls 和 self都是一种命名习惯习惯，cls作为第一个参数来表示类本身. 在类方法中用到
```

非绑定方法

- 在类中有很多普通的方法，本身不需要绑定给对象或者类来使用，谁都可以调用，没有自动传值效果，这就是非绑定方法。在python类中会默认给方法(函数)一个实例，用于绑定给对象；如何解除这个绑定关系，需要用到，@staticmethod方法

```
class People:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def print_info():  # 去掉了self，会报错
        print("我是一个普通的方法")


obj = People('精神小伙', 18)
obj.print_info()
# 结果：报错
# TypeError: print_info() takes 0 positional arguments but 1 was given
```

```
class People:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @staticmethod
    def print_info():
        print("我是一个普通的方法")


obj = People('精神小伙', 18)
obj.print_info()
People.print_info()
```

- 使用 @staticmethod 方法，即可正常使用函数，并将原本默认绑定给对象使

用的方法，变为一个普通函数。既然是函数，也就可以遵循函数的使用要求，需要几个函数，就传入几个函数。

```python
class People:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @staticmethod
    def print_info(x, y):
        print("我是一个普通的方法")
        print(x, y)


obj = People('精神小伙', 18)
obj.print_info(100, 200)
People.print_info(300, 400)
```

应用

绑定给对象

```python
class People:
    def __init__(self, name, age, sex):
        self.name = name
        self.age = age
        self.sex = sex

        # 看人（对象）的信息
    def tell_info(self):  # 绑定给对象的方法
        # 我们要看的是一个人的信息 一个人是一个对象 那也就是说我们需要
        # 对象.方法
        # 那么我这个方法就应该是绑定给对象的方法
        # 根据函数体的逻辑来想参数填什么

        print('Name:%s Age:%s Sex:%s' % (self.name, self.age, self.sex))


p = People('egon', 18, 'male')

```

```
# 绑定给对象，就应该由对象来调用，自动将对象本身当作第一个参数传入
p.tell_info()  # tell_info(p)
```

绑定给类

```
# 现在我们的需求是
# 在实例化的时候从配置文件里面读取配置信息进行实例化

import settings

# print(settings.name)
# print(settings.age)
# print(settings.sex)



class People:
    def __init__(self, name, age, sex):
        self.name = name
        self.age = age
        self.sex = sex

    @classmethod
    def from_conf(cls):
        obj = People(settings.name, settings.age, settings.sex)
        return obj

    def tell_info(self):  # 绑定对象的方法
        print('Name:%s Age:%s Sex:%s' % (self.name, self.age, self.sex))

# 没有类方法
p1 = People(settings.name, settings.age, settings.sex)

# 有类方法
p2 = People.from_conf()

# 换一种思路 从配置文件里面读取配置信息进行实例化（这是不是一个功能？）

def from_conf():
```

```
    obj = People(settings.name, settings.age, settings.sex)
    return obj
```

非绑定方法

```
# 给每一个人生成一个id号
import time
import hashlib

class People:
    def __init__(self, name, age, sex):
        self.name = name
        self.age = age
        self.sex = sex
        self.id = People.create_id()  # 自动生成id号

    # 什么时候用非绑定方法?
    # 不依赖类或者对象的时候

    @staticmethod
    def create_id():
        m = str(time.time())  # 根据时间不同创建值
        return m


p1 = People("焦云飞", 18, "male")
time.sleep(1)
p2 = People("云飞", 10, "male")
time.sleep(1)
p3 = People("飞", 8, "male")


print(p1.id)
print(p2.id)
print(p3.id)
```