**浅谈静态方法**

上次的文件中我们学习了装饰器的相关知识，具体可从以下链接进行查看：

**装饰器系列(3)-基于类的装饰器的实现**

本次文章我们一起来学习一下Python中的静态方法，在装饰器之后讲述静态方法的原因是因为我们这个方法使用了装饰器，具体情况请看下文。

**如果我们将类中的方法进行划分，可以得到这样大致三种方法：**

**（1）不用任何修改的实例方法。**

**（2）使用@classmethod修饰的类方法；**

**（3）使用@staticmethod修饰的静态方法；**

**一、什么是静态方法**

我们知道在Python中，当我们在类中定义一个方法之后，会创建一个**类的对象(类的实例)**去调用这个方法，这个被调用的方法我们命名它为实例方法。另外在Python中还有一种方法叫静态方法，静态方法与类方法极为相似,不同之处在于静态方法绑定到一个类而不是该类的对象。

从上述事实可以发现，使用静态方法的好处是，我们不需要实例化对象即可使用该方法。从这个角度上来说调用方法变得非常方便。在Python中静态方法我们使用**＠staticmethod装饰器**来进行修饰。

这里我们给出一些结论：

1. 静态方法可以不带任何参数，由于静态方法没有self参数，所以它无法访问类的实例成员；
2. 静态方法也没有cls参数，所以它也无法访问类成员。
3. 静态方法既可以通过对象名调用，也可以通过类名调用。

**二、静态方法的使用**

我们知道了静态方法是一种不需要引用任何类或实例的方法，我们可以这样来定义一个静态方法：

class People:  
 *# 类构造方法，属于实例方法* def \_\_init\_\_(self):  
 self.name = "Python知识学堂"  
 self.writeage = 5  
 *# 静态方法* @staticmethod  
 def myname(name):  
 print("my name is",name)

*# 实例访问*  
a = People()  
a.myname('Python知识学堂')

*# 类直接调用*

People.myname('Python知识学堂')

*# 输出*

*my name is Python知识学堂*

*my name is Python知识学堂*

**三、类的实例方法**

既然聊到了类的静态方法，那就顺便说一下类的实例方法，类的实例方法就是我们常见的那些使用了self参数的那些类方法：

class People:  
 *# 类构造方法，属于实例方法* def \_\_init\_\_(self):  
 self.name = "Python知识学堂"  
 self.writeage = 5  
 *# 静态方法* @staticmethod  
 def myname(name):  
 print("my name is",name)  
 *# 类的实例方法* def language(self):  
 print('我的写作年龄是',self.writeage)  
a = People()  
a.language()

*# 输出*

*我的写作年龄是 5*

这里我们也可以使用类名调用实例的方式进行调用，如：

a = People()  
People.language(a)

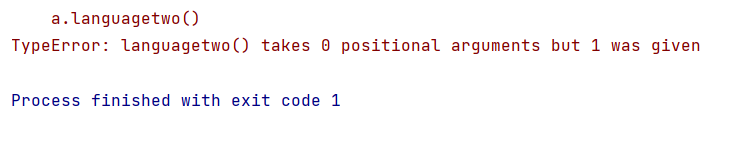
说到调用，我们这里补充一下有self参数和没有self参数的方法的调用，比如现在我们的类是这样的：

class People:  
 *# 类构造方法，属于实例方法* def \_\_init\_\_(self):  
 self.name = "Python知识学堂"  
 self.writeage = 5  
  
 *# 静态方法* @staticmethod  
 def myname(name):  
 print("my name is",name)  
  
 *# 类的实例方法* def language(self):  
 print('我的写作年龄是',self.writeage)  
  
 def languagetwo():  
 print('我的第二语言是英语')

调用如下：

a = People()  
a.languagetwo()

则会有以下报错：

  
如果使用类直接来调用没有self参数的方法，则会正常输出：

People.languagetwo()

*# 我的第二语言是英语*

所以我们得出以下的结论：

**带有参数 self的类方法，则只能被实例化的对象调用；对于不带参数 self的类方法，则只能被类调用。**

**四、类方法**

接下来就是类方法，我们把@classmethod 修饰的方法称为类方法，这个方法跟静态方法一样也是无需self参数的，唯一不同的是其在静态方法基础上增加了cls参数，且这个参数是方法的第一个参数:

class People:  
 *# 类构造方法，属于实例方法* def \_\_init\_\_(self):  
 self.name = "Python知识学堂"  
 self.writeage = 5  
  
 myaddress = '福建省厦门市思明区'  
 *# 静态方法* @staticmethod  
 def myname(name):  
 print("my name is",name)  
  
 *# 类的实例方法* def language(self):  
 print('我的写作年龄是',self.writeage)  
  
 *# 类方法*  
 @classmethod  
 def address(cls):  
 print('正在执行类方法为:', cls)  
 print('正在执行类方法，输出的地址为:',cls.myaddress)

*# 使用类名直接调用类方法  
# People.address()*  
*# 使用类对象调用类方法  
# a = People()  
# a.address()*

*# 输出均为:正在执行类方法，输出的地址为:*

*正在执行类方法为: <class '\_\_main\_\_.People'>*

*正在执行类方法，输出的地址为: 福建省厦门市思明区*

看输出我们可以发现，cls这个参数其实指的就是People这个类。

**接下来我们使用类方法来实现一个需求：统计创建对象的个数，代码如下：**

class People():  
 *# 初始化类参数* sum = 0  
  
 *# 构造函数* def \_\_init\_\_(self):  
 *# 自增统计,使用类进行访问* People.sum += 1  
  
 @classmethod  
 def how\_many(cls):

print("创建的对象总数为：", cls.sum)

*# 创建工具对象*a = People()  
b = People()  
c = People()  
  
*# 调用类方法*c.how\_many()  
People.how\_many()

*# 上述我们一共创建了3个实例，a,b,c。最后使用实例调用方法和类调用打印输出结果为：*

*创建的对象总数为： 3*

*创建的对象总数为： 3*

**五、总结**

在实际编程中，我们很好用到静态方法和类方法，当然了也不是不会用到。笔者在实际的编程过程中大多数时候使用函数(改写)的方式来实现，替换实际想要实现的功能即可。

这么说来静态方法逻辑上属于类，但是和类本身没有太大的关系，也就是说在静态方法中，不会涉及到类中的属性和方法的操作(毕竟不能访问)，结果表现出来就是静态方法置于该类的名称空间中，仅此而已。但是我们需要明白的是，使用静态方法的好处是:

（1）可读性好：看到@staticmethod就知道这个方法并不需要依赖对象本身的状态。

（2）节省内存，既然是与类实例无关自然就不用创建类实例对象。

工厂模型