[**python 类中方法总结 --- 实例方法、类方法、静态方法**](https://www.cnblogs.com/nbk-zyc/p/13213122.html)

　　在python的类语法中，可以出现三种方法，具体如下：

　　（1）实例方法

　　　　1）第一个参数必须是实例本身，一般使用【self】表示。

　　　　2）在实例方法中，可以通过【self】来操作实例属性，【类名】来操作类属性。

　　　　3）实例方法只能通过实例对象去调用，尽管也可以通过类名间接调用【类名.方法名(self, ...)】，但此时仍然需要传入self对象。

　　（2）类方法

　　　　1）使用【@classmethod】修饰函数，且第一个参数必须是类本身，一般使用【cls】表示。

　　　　2）在类方法中，可以使用【cls=类名】来操作类属性，但是不能操作实例属性（self对象中存储的变量）。

　　　　3）类方法可以通过实例对象或者类对象调用。

　　（3）静态方法

　　　　1）使用【@staticmethod】修饰函数，不需要使用任何参数表示。与普通函数一样，只是将该方法放到了类中实现而已。

　　　　2）使用方式与类方法一样，参考类方法中的 2）、3）。（注：静态方法只能通过【类名】去操作类属性；）

**案例1：实例方法、类方法、静态方法使用**

[复制代码](javascript:void(0);)

class Foo(object):

"""类三种方法语法形式"""

count = 0 # 统计实例对象的数量

class\_method\_count = 0 # 统计类方法的调用次数

# 实例方法

def \_\_init\_\_(self, name):

self.name = name

Foo.count += 1

# 实例方法

def instance\_method(self):

print("是类{}的实例方法，只能被实例对象调用".format(Foo))

print("产生了一个<{}>实例，共有<{}>个实例对象".format(self.name, Foo.count))

# 类方法

@classmethod

def class\_method(cls):

print("是类{}的类方法，可以被实例对象、类对象调用".format(cls))

cls.\_\_static\_method\_test()

cls.class\_method\_count += 1

# 静态方法

@staticmethod

def static\_method():

print("是类{}的静态方法，可以被实例对象、类对象调用".format(Foo))

print("+++以下内容为类方法class\_method()的运行结果：")

Foo.class\_method()

@staticmethod

def \_\_static\_method\_test():

print("调用了静态方法 static\_method\_test()")

print("--"\*20 + "实例方法测试" + "--"\*20)

obj1 = Foo("dog")

obj1.instance\_method() # <=> Foo.instance\_method(obj1)

print("--"\*20 + "类方法测试" + "--"\*20)

obj1.class\_method()

print("--"\*20)

Foo.class\_method()

print("--"\*20 + "静态方法测试" + "--" \* 20)

obj1.static\_method()

print("--"\*20)

Foo.static\_method()

"""

运行结果：

----------------------------------------实例方法测试----------------------------------------

是类<class '\_\_main\_\_.Foo'>的实例方法，只能被实例对象调用

产生了一个<dog>实例，共有<1>个实例对象

----------------------------------------类方法测试----------------------------------------

是类<class '\_\_main\_\_.Foo'>的类方法，可以被实例对象、类对象调用

调用了静态方法 static\_method\_test()

----------------------------------------

是类<class '\_\_main\_\_.Foo'>的类方法，可以被实例对象、类对象调用

调用了静态方法 static\_method\_test()

----------------------------------------静态方法测试----------------------------------------

是类<class '\_\_main\_\_.Foo'>的静态方法，可以被实例对象、类对象调用

+++以下内容为类方法class\_method()的运行结果：

是类<class '\_\_main\_\_.Foo'>的类方法，可以被实例对象、类对象调用

调用了静态方法 static\_method\_test()

----------------------------------------

是类<class '\_\_main\_\_.Foo'>的静态方法，可以被实例对象、类对象调用

+++以下内容为类方法class\_method()的运行结果：

是类<class '\_\_main\_\_.Foo'>的类方法，可以被实例对象、类对象调用

调用了静态方法 static\_method\_test()

"""

[复制代码](javascript:void(0);)

　　从案例1中得到，类方法与静态方法可以相互调用，但是静态方法只能用【类名】表示，而类方法用【cls】就比较方便了。

**案例2：实例方法、类方法、静态方法在继承中（子类重写父类中的方法）的使用**

[复制代码](javascript:void(0);)

class Foo(object):

X = 1

Y = 14

@staticmethod

def average(\*mixes):

print("父类中的静态方法 average(\*mixes)")

print("\*\*\*\*\*", mixes)

return sum(mixes) / len(mixes)

@staticmethod

def static\_method():

print("父类中的静态方法 static\_method()")

return Foo.average(Foo.X, Foo.Y) # 注：因为这儿已经限定了只允许调用父类中的average()

@classmethod

def class\_method(cls): # 父类中的类方法

print("父类中的类方法 class\_method(cls)")

return cls.average(cls.X, cls.Y) # 注：若用子类对象调用该函数，此时的cls==Son，故调用子类重写后的average()

class Son(Foo):

@staticmethod

def average(\*mixes): # "子类中重载了父类的静态方法"

print("子类中重载了父类的静态方法 average(\*mixes)")

print("\*\*\*\*\*", mixes)

return sum(mixes) / len(mixes)

print(Son.average(1, 2, 3), "\n" + "---" \* 20)

print(Son.class\_method(), "\n" + "---" \* 20)

print(Son.static\_method(), "\n" + "---" \* 20)

"""

运行结果：

------------------------------------------------------------

子类中重载了父类的静态方法 average(\*mixes)

\*\*\*\*\* (1, 2, 3)

2.0

------------------------------------------------------------

父类中的类方法 class\_method(cls)

子类中重载了父类的静态方法 average(\*mixes)

\*\*\*\*\* (1, 14)

7.5

------------------------------------------------------------

父类中的静态方法 static\_method()

父类中的静态方法 average(\*mixes)

\*\*\*\*\* (1, 14)

7.5

------------------------------------------------------------

"""

[复制代码](javascript:void(0);)

　　从案例2中得到，子类对象调用父类中的 类方法 class\_method(cls) 时，由于【cls】此时为【Son】，故会执行子类中的类方法、静态方法、静态属性；进一步推断出，类方法中【cls】取决于类方法的调用者，只有发生类方法调用后才能知道执行结果。而父类中静态方法 static\_method()只能调用父类中类方法、静态方法、静态属性，与静态方法的调用者无关；进一步推断，静态方法可以提前预知程序的执行结果，如执行当前类中的某个类方法或静态方法。

**对案例2做一个多态测试，添加代码如下：**

[复制代码](javascript:void(0);)

def test\_polymorphic(foo):

"""

多态测试

:param foo: 父类对象

:return: None

"""

print(f.average(1, 2, 3))

f = Foo()

test\_polymorphic(f)

print("----"\*10)

f = Son() # 会触发多态

test\_polymorphic(f)

"""

父类中的静态方法 average(\*mixes)

\*\*\*\*\* (1, 2, 3)

2.0

----------------------------------------

子类中重载了父类的静态方法 average(\*mixes)

\*\*\*\*\* (1, 2, 3)

2.0

"""

[复制代码](javascript:void(0);)

　　继续做多态测试，添加代码如下：

[复制代码](javascript:void(0);)

def test\_polymorphic(foo):

"""

多态测试

:param foo: 父类对象

:return: None

"""

print(f.class\_method())

f = Foo()

test\_polymorphic(f)

print("----"\*10)

f = Son() # 会触发多态

test\_polymorphic(f)

"""

父类中的类方法 class\_method(cls)

父类中的静态方法 average(\*mixes)

\*\*\*\*\* (1, 14)

7.5

----------------------------------------

父类中的类方法 class\_method(cls)

子类中重载了父类的静态方法 average(\*mixes)

\*\*\*\*\* (1, 14)

7.5

"""

[复制代码](javascript:void(0);)

**案例3：使用类方法或静态方法初始化类（可以自定义类的初始方式）**

[复制代码](javascript:void(0);)

class Book(object):

def \_\_init\_\_(self, title):

self.\_\_title = title

@classmethod

def object\_create\_by\_class\_method(cls, title\_list):

"""

使用生成器实例化多个对象

:param title\_list: 每个对象的初始化参数，List

:return:迭代器，每个实例化对象

"""

for title in title\_list:

yield cls(title=title)

@staticmethod

def object\_create\_by\_static\_method(title\_list): # 功能与类方法一样

for title in title\_list:

yield Book(title=title)

@property

def title(self):

return self.\_\_title

@title.setter

def title(self, value):

if not isinstance(value, str):

raise TypeError('%s must be str' % value)

self.\_\_title = value

@title.deleter

def title(self):

del self.\_\_title

# raise TypeError('Can not delete')

books = ["Chinese", "mathematics", "English"]

g\_books = Book.object\_create\_by\_class\_method(books)

print(g\_books) # <generator object Book.object\_create\_by\_class\_method at 0x000001FB72AFEEC8>

print(g\_books.\_\_next\_\_().title) # 查看书的title -- Chinese

print(g\_books.\_\_next\_\_().title) # 查看书的title -- mathematics

book = g\_books.\_\_next\_\_() # 得到一个实例化对象

print(book.title) # 查看书的title -- English

print(book.\_\_dict\_\_) # {'\_Book\_\_title': 'English'}

book.title = "英语" # 修改属性

print(book.title) # 查看书的title -- 英语

print(book.\_\_dict\_\_) # {'\_Book\_\_title': '英语'}

del book.title # 删除属性

print(book.\_\_dict\_\_) # {}

[复制代码](javascript:void(0);)

　　从案例3中得到，使用类方法或者静态方法可以自定义类的初始化方式，本案例中实现的功能是使用生成器批量创建多个对象。同时，案列中使用了【property】属性，property的作用相当于执行了某个函数，并获得该函数的返回值；其使用方式有3种，分别为【@property --- get】、【@函数名.setter --- set】、【@函数名.deleter --- del】，后二种必须在第一种使用的情况下才能使用，在表现形式上就是通过实例化对象间接访问属性。

　　最后，我们对案例3做微小改变，来看看实例方法、类方法、静态方法与类、对象之间的关系，代码如下：

[复制代码](javascript:void(0);)

class Book(object):

@classmethod

def object\_create\_by\_class\_method(cls):

pass

@staticmethod

def object\_create\_by\_static\_method():

pass

def instance\_method(self):

pass

book = Book()

print(book.instance\_method) # 方法（绑定到对象：发生调用时，将【实例对象=self】自动作为第一个参数传入实例方法中）

print(book.object\_create\_by\_class\_method) # 方法（绑定到类：发生调用时，将【类=cls】自动作为第一个参数传入类方法中）

print(book.object\_create\_by\_static\_method) # 普通函数（非绑定：发生调用时，不需要传入任何参数，如self，cls）

print(Book.instance\_method) # 普通函数（非绑定，因为实例方法就存储在类的命名空间中）

print(Book.object\_create\_by\_class\_method) # 绑定到类

print(Book.object\_create\_by\_static\_method) # 普通函数（非绑定）

"""

<bound method Book.instance\_method of <\_\_main\_\_.Book object at 0x000002481CEDE988>>

<bound method Book.object\_create\_by\_class\_method of <class '\_\_main\_\_.Book'>>

<function Book.object\_create\_by\_static\_method at 0x000002481D12C828>

<function Book.instance\_method at 0x000002481D12C8B8>

<bound method Book.object\_create\_by\_class\_method of <class '\_\_main\_\_.Book'>>

<function Book.object\_create\_by\_static\_method at 0x000002481D12C828>

"""

[复制代码](javascript:void(0);)

　　可以发现，

　　　　1）类中的实例方法都绑定了实例对象，故建议使用实例对象调用实例方法；

　　　　2）类中的类方法都绑定了类，故建议使用类对象调用类方法，即使使用实例对象调用类方法，仍然自动将【类名】作为第一个参数传入类方法。

　　　　3）而对于静态方法而言，无论使用实例对象或者类对象调用其方法都不会自动进行参数传递，故不做限制。

　　　　4）若能用类解决的问题，就将类中的方法定义为类方法，这样可以避免实例化对象的开销。

 绑定与非绑定的参考链接：