给 Python 已存在的类动态的添加方法（method）

monkeypatching

Python (/categories/#python) - Monkeypatching (/tags/#monkeypatching)

By Damnever on May 7, 2015

2015.11.2: python tornado中如何给每个服务器请求动态加上装饰器？@Damnever 的回答 (http://segmentfault.com/q/1010000003939756/a-1020000003940021)

下面是正题。。。 ***

动态添加实例属性的时候，这样：`obj.a = 1` 或 `setattr(obj, 'a', 1)` 就可以了，so easy！

但是，动态添加方法的时候，问题来了。

这里动态的给一个实例添加一个方法：

```
>>> class A(object):
...     pass
...
>>> def hello(self):
...     print "hello"
...
>>> a = A()
>>> a.hello = hello
```

调用一下，What the hell??

```
>>> a.hello()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: hello() takes exactly 1 argument (0 given)
```

对于实例来说只是 function ！连 method 都不是…

```
>>> a.hello
<function hello at 0x7ff1693ab758>
>>> a.hello(1)
hello
```

这是什么情况我也不得而知…

参考 stackoverflow 上的最高票回答Adding a Method to an Existing Object (http://stackoverflow.com/a/2982/2996656)。

对于一个类型（非实例）来说，它的属性都是 unbound 的：

```
>>> class A(object):
...     def hello(self):
...         print "hello"
...
>>> A.hello
<unbound method A.hello>
>>> A().hello
<bound method A.hello of <__main__.A object at 0x7f171e986810>>
```

所以可以通过类型来动态添加一个方法，对于实例来说，它是 bound 的，任何这个类型的实例都可以访问：

```
>>> class A(object):
...     pass
...
>>> def hello(self):
...     print 'hello'
...
>>> A.hello = hello
>>> A().hello()
hello
>>> A.hello
<unbound method A.hello>
>>> A().hello
<bound method A.hello of <__main__.A object at 0x7f62edae7810>>
```

另一种方式是使用types.MethodType (https://docs.python.org/2/library/types.html#types.MethodType)，这一种只对当前实例起作用：

```
>>> import types
>>> class A(object):
...     pass
...
>>> def hello(self):
...     print hello
...
>>> a = A()
>>> a.hello = types.MethodType(hello, a)
>>> a.hello
<bound method ?.hello of <__main__.A object at 0x7fd8a318a290>>
>>> A.hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'A' has no attribute 'hello'
```

types (https://docs.python.org/2/library/types.html)模块是 2.6 版之后new (https://docs.python.org/2/library/new.html)模块的替代（Python 3里已经不存在 new 模块了），所以 new 里面的函数 types 模块都有替代，如：

```
>>> b = types.InstanceType(A)
>>> b
<__main__.A instance at 0x7fd8a318c488>
>>> b.hello = types.UnboundMethodType(hello, A)
>>> b.hello
<bound method ?.hello of <class __main__.A at 0x7fd8a315cc80>>
>>> A.hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: class A has no attribute 'hello'
```

这里忍不住要看一下 types 的源码 (https://hg.python.org/cpython/file/2.7/Lib/types.py)，很简短，但是你懂得… 涉及到 type 的，脑子里都是空白一片

这里还是不懂是什么机制，不过另一个答案Adding a Method to an Existing Object (http://stackoverflow.com/a/8961717/2996656)提到了 `descriptor protocol` 。

描述器是属性, 实例方法, 静态方法, 类方法和 super 的背后的实现机制。

参考Descriptor HowTo Guide (http://docs.python.org/2/howto/descriptor.html)及译文Python描述器引导(翻译) (http://pyzh.readthedocs.org/en/latest/Descriptor-HOW-TO-Guide.html)。

另外动态的添加 `staticmethod` ， `classmethod` 的方式：

```
>>> class A(object):
...     pass
...
>>> def hello():
...     print 'hello'
...
>>> def world(cls):
...     print 'world'
...
>>> A.hello = staticmethod(hello)  # setattr(A, 'hello', staticmethod(hello))
>>> A.world = classmethod(world)  # setattr(A, 'hello', classmethod(world))
>>> a = A()
>>> a.hello()
hello
>>> a.world()
world
>>> a.hello  # A.hello 也是 function hello
<function hello at 0x7f39b4bf3758>
>>> a.world  # A.world 也是 bound method classobj.world
<bound method classobj.world of <class __main__.A at 0x7f39b4bf0258>>
```

话说我是怎么掉进这个坑的？其实只是为了搞一个奇葩的装饰器…

这也正是动态语言的强大之处：

```
class AddHigh(object):
        """ 给长方形添加高变成长方体 """
    def __init__(self, cls, z):
        self._cls = cls
        self._z = z

    def __call__(self, *args, **kwargs):
        # 重写面积计算方式
        def _area(this, *args, **kwargs):
            return (self._z * this._x + self._z * this._y + this._x * this._y) *
 2

        self._cls.area = _area
        obj = self._cls(*args, **kwargs)
        return obj


class Rectangle(object):
        """ 长方形 """
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def area(self):
        return self._x * self._y

if __name__ == '__main__':
    a = Rectangle(1, 2)
    print a.area()
    b = AddHigh(Rectangle, 3)(1, 2)
    print b.area()

# 2
# 22
```


← PREVIOUS POST (/2015/04/24/PEP8-STYLE-GUIDE-FOR-PYTHON-CODE/)

NEXT POST → (/2015/07/16/EXTENDING-PYTHON-WITH-C/)

社交帐号登录: 微博 QQ 人人 豆瓣 更多»

说点什么吧...

发布

 (/feed.xml)  (http://www.douban.com/people/LastD001/)

 (https://github.com/Damnever)