Python_实现类的接口检测

1.用python设计模式时，往往需要传入的对象具有我们所要求的属性和方法，这时我们可以用一下的装饰器修饰:

```
#coding=utf8
import abc,collections



#定义抽象基类,完全不能实例化
class Interface(metaclass=abc.ABCMeta): #必须设定metaclass,否则不会调用__subclasshook__方法

    method_list = ["set","get"]
    @classmethod
    def __subclasshook__(Class,SubClass):
        print("ClassHook:{} SubClassHook:{}".format(Class.__name__,SubClass.__name__))
        #要求传进来的子类必须具有规定的set,get方法才可以
        properties = collections.ChainMap( *(superClass.__dict__
                                     for superClass in  SubClass.__mro__))
        for pro in properties:
            print(pro)
        if all(method in properties and callable(method) for method in Class.method_list):
            return True
        return False

#为方便以后使用，尝试将以上代码转为一个类装饰器
def decorater(*method_list):
    def ClassDecorator(cls):
        def subclasshook(Class,SubClass):
            print("ClassHook:{} SubClassHook:{}".format(Class.__name__,SubClass.__name__))
            #要求传进来的子类必须具有规定的set,get方法才可以
            properties = collections.ChainMap( *(superClass.__dict__
                                     for superClass in  SubClass.__mro__))
            for pro in properties:
                print("Property:",pro)
            print("Method_List:{}".format(method_list))
            if all(method in properties  for method in method_list):
                return True
            return False
        cls.__subclasshook__ = classmethod(subclasshook) #特别注意:classmethod不能遗漏！
        return cls
    return ClassDecorator
```

用装饰器的方法来装饰一个Interface:

```
@decorater('get','has') #要求传入接口的对象必须具有get和has属性或者方法
class Interface1(metaclass=abc.ABCMeta):pass
```

使用:

```
class Test1(object):

    def get(self): pass
    def has(self):pass

if __name__=="__main__":
    print(issubclass(Test1,Interface1))#这两种方法都可以，instance的比较需要先转换为类再比较
    print(isinstance(Test1(),Interface1))
```