属性管理：

例：程序property\_test/property\_test1.py

class Person:  
 def \_\_init\_\_(self, name):  
 self.\_name = name  
  
 def get\_name(self):  
 print('fetch...')  
 return self.\_name  
  
 def set\_name(self, value):  
 print('change...')  
 self.\_name = value  
  
 def del\_name(self):  
 print('remove...')  
 del self.\_name  
 name = property(get\_name, set\_name, del\_name, "name property docs")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 bob = Person('Bob Smith')  
 print(bob.name)  
 bob.name = 'Robert Smith'  
 print(bob.name)  
 del bob.name  
 # print(bob.name) # AttributeError  
  
 print('-' \* 20)  
 sue = Person('Sue Jones')  
 print(sue.name)  
 print(Person.name.\_\_doc\_\_)

输出为：

fetch...

Bob Smith

change...

fetch...

Robert Smith

remove...

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

fetch...

Sue Jones

name property docs

用装饰器管理属性：

例：程序property\_test/property\_test2.py

class Person:  
 def \_\_init\_\_(self, name):  
 self.\_name = name  
  
 @property  
 def name(self):  
 *"name property docs"* print('fetch...')  
 return self.\_name  
  
 @name.setter  
 def name(self, value):  
 print('change...')  
 self.\_name = value  
  
 @name.deleter  
 def name(self):  
 print('remove...')  
 del self.\_name  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 bob = Person('Bob Smith')  
 print(bob.name)  
 bob.name = 'Robert Smith'  
 print(bob.name)  
 del bob.name

输出为：

fetch...

Bob Smith

change...

fetch...

Robert Smith

remove...

描述符：作为独立的类创建，并且像方法函数一样分配给类属性。

描述符类的通用形式：

Class Descriptor:

“docstring goes here”

def \_\_get\_\_(self, instance, owner):…

def \_\_set\_\_(self, instance, value):…

def \_\_delete\_\_(self, instance):…

其中self为描述符类实例，instance为描述符类实例附加的客户类的实例，\_\_get\_\_方法中的owner指定描述符实例要附加到的类。

描述符类实例是一个类属性，由客户类和任何子类的所有实例所继承。

例：程序property\_test/descriptor\_test1.py

# 描述符类  
# 带有\_\_get\_\_, \_\_set\_\_, \_\_delete\_\_  
# 都可以看做描述符类  
class Name:  
 *"name descriptor docs"* def \_\_get\_\_(self, instance, owner):  
 print('fetch...')  
 return instance.\_name  
  
 def \_\_set\_\_(self, instance, value):  
 print('change...')  
 instance.\_name = value  
  
 def \_\_delete\_\_(self, instance):  
 print('remove...')  
 del instance.\_name  
  
  
class Person:  
 def \_\_init\_\_(self, name):  
 self.\_name = name  
 name = Name() # 描述符实例赋给类属性（必须是类属性）  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 bob = Person('Bob Smith')  
 print(bob.name)  
 bob.name = 'Robert Smith'  
 print(bob.name)  
 del bob.name  
 # Person.name # AttributeError，name属性会访问到实例属性\_name

例：程序property\_test/descriptor\_test2.py

class Descriptor:  
 def \_\_get\_\_(self, instance, owner):  
 print(self, instance, owner, sep='\n')  
  
  
class Sub:  
 attr = Descriptor()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x = Sub()  
 x.attr # 实例访问  
 print()  
 Sub.attr # 类访问

输出为：

<\_\_main\_\_.Descriptor object at 0x0000006787354208>

<\_\_main\_\_.Sub object at 0x0000006787354278>

<class '\_\_main\_\_.Sub'>

<\_\_main\_\_.Descriptor object at 0x0000006787354208>

None

<class '\_\_main\_\_.Sub'>

通过类访问描述符实例附加的属性时，instance为None。

只读描述符：

例：程序property\_test/descriptor\_test3.py

# 只有\_\_get\_\_不能确保只读  
class Descriptor:  
 def \_\_get\_\_(self, instance, owner):  
 print('get')  
  
  
class Test:  
 attr = Descriptor()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x = Test()  
 x.attr  
 Test.attr  
 print(id(x.attr))  
 x.attr = 99  
 print(x.attr) # 99，屏蔽了描述符attr，x.attr访问实例x的独有属性  
 print(id(x.attr))  
 Test.attr # get  
 y = Test()  
 y.attr # get

输出为：

get

get

get

1818272976

99

1818720800

get

get

从输出可以看出，描述符类Descriptor只有\_\_get\_\_方法，但不足以实现只读描述符，原因是Python还有通用的实例属性赋值方式，会屏蔽掉描述符属性。

例：程序property\_test/descriptor\_test4.py

class Descriptor:  
 def \_\_get\_\_(self, instance, owner):  
 print('get')  
  
 def \_\_set\_\_(self, instance, value):  
 raise AttributeError('can`t set')  
  
  
class Test:  
 attr = Descriptor()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x = Test()  
 x.attr  
 # x.attr = 99 # AttributeError，can`t set  
 Test.attr = 99 # 无法拦截以类属性方式赋值  
 print(Test.attr) # 99

当赋值的属性是一个描述符时，Python会绕过常规实例层级的赋值，但无法绕过类属性赋值。

描述符使用的状态

例：程序property\_test/descriptor\_test5.py

# 使用描述符实例的状态  
class DescriptorState:  
 def \_\_init\_\_(self, value):  
 self.value = value  
  
 def \_\_get\_\_(self, instance, owner):  
 print('DescriptorState get')  
 return self.value \* 10  
  
 def \_\_set\_\_(self, instance, value):  
 print('DescriptorState set')  
 self.value = value  
  
  
# 使用客户类实例的状态  
class InstanceState:  
 def \_\_get\_\_(self, instance, owner):  
 print('InstanceState get')  
 return instance.\_x \* 100  
  
 def \_\_set\_\_(self, instance, value):  
 print('InstanceState set')  
 instance.\_x = value  
  
  
class CalcAttrs:  
 attr1 = DescriptorState(2)  
 attr2 = InstanceState()  
  
 def \_\_init\_\_(self):  
 self.\_x = 3  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 obj = CalcAttrs()  
 print(obj.attr1)  
 print(obj.attr2)  
 print()  
 obj.attr1 = 5  
 obj.attr2 = 6  
 print(obj.attr1)  
 print(obj.attr2)

输出为：

DescriptorState get

20

InstanceState get

300

DescriptorState set

InstanceState set

DescriptorState get

50

InstanceState get

600