

Python 描述器解析



ButteredCat 关注

2017.08.22 20:14:13 字数 1,533 阅读 2,887

语法简析

一般来说，描述器（descriptor）是一个有“绑定行为”的对象属性（object attribute），它的属性访问被描述器协议方法重写。这些方法是 `__get__()`、`__set__()` 和 `__delete__()`。如果一个对象定义了以上任意一个方法，它就是一个描述器。而描述器协议的具体形式如下：

```
1 descr.__get__(self, obj, type=None) --> value
2
3 descr.__set__(self, obj, value) --> None
4
5 descr.__delete__(self, obj) --> None
```

描述器本质上是一个类对象，该对象定义了描述器协议三种方法中至少一种。而这三种方法只有当类的实例出现在一个所有者类（owner class）之内时才有效，也就是说，描述器必须出现在所有者类或其父类的字典 `__dict__` 里。这里提到了两个类，一是定义了描述器协议的描述器类，另一个是使用描述器的所有者类。

描述器往往以装饰器的方式被使用，导致二者常被混淆。描述器类和不带参数的装饰器类一样，都传入函数对象作为参数，并返回一个类实例，所不同的是，装饰器类返回 callable 的实例，描述器则返回描述器实例。

记住上面的话，下面我们举例说明。

@Property

Python 内置的 `property` 函数可以说是最著名的描述器之一，几乎所有讲述描述器的文章都会拿它做例子。

`property` 是用 C 实现的，不过这里有一份等价的 Python 实现：

```
1 class Property(object):
2     "Emulate PyProperty_Type() in Objects/descrobject.c"
3
4     def __init__(self, fget=None, fset=None, fdel=None, doc=None):
5         self.fget = fget
6         self.fset = fset
7         self.fdel = fdel
8         if doc is None and fget is not None:
9             doc = fget.__doc__
10        self.__doc__ = doc
11
12        def __get__(self, obj, objtype=None):
13            if obj is None:
14                return self
15            if self.fget is None:
16                raise AttributeError("unreadable attribute")
17            return self.fget(obj)
18
19        def __set__(self, obj, value):
20            if self.fset is None:
21                raise AttributeError("can't set attribute")
22            self.fset(obj, value)
23
```

推荐阅读

个人贷款用户分析（特征分析&逻辑回归）

阅读 101

原地哈希算法

阅读 607

Ubuntu+GPU+Tensorflow运行tf-faster-rcnn【光纤分类项目】

阅读 181

假设检验和ABTEST（一）

阅读 121

数据集问题？

阅读 29

```

30         return type(self)(fget, self.fset, self.fdel, self.__doc__)
31
32     def setter(self, fset):
33         return type(self)(self.fget, fset, self.fdel, self.__doc__)
34
35     def deleter(self, fdel):
36         return type(self)(self.fget, self.fset, fdel, self.__doc__)

```

Property 怎么用呢？看下面的例子：

```

1 class C(object):
2     def __init__(self):
3         self._x = None
4
5     @Property
6     def x(self):
7         """I'm the 'x' property."""
8         return self._x
9
10    @x.setter
11    def x(self, value):
12        assert value > 0
13        self._x = value
14
15    @x.deleter
16    def x(self):
17        del self._x

```

我们结合源代码和用法来分析 **Property**。

@Property 的用法就是一个装饰器。我们可以将其等价转化为：

```

1 x = Property(x)

```

函数 **x** 作为位置参数被赋给 **Property.__init__()** 的 **fget**，得到新的 **x** 已经不是个函数而是个完整实现了 **__get__()** 方法的描述器实例了。

@x.setter 的用法略有不同。它实际上是利用上面定义的描述器实例 **x** 的 **setter** 方法，重新创建了新的实例。这时变量 **x** 再次被更新，指向了一个完整实现 **__get__()** 和 **__set__()** 方法的新描述器。传入 **setter** 方法的函数名必须是 **x**，否则如果是 **y**，按照装饰器的性质，

```

1 y = x.setter(y)

```

新描述器就被 **y** 引用了，与需求不符。

Property 提供了像访问类“成员变量”一样访问 **get**、**set** 方法的能力。

```

1 In [123]: c = C()
2
3 In [124]: c.x = 1
4
5 In [125]: c.x
6 Out[125]: 1
7
8 In [126]: c.x = 0
9 -----
10 AssertionError                                Traceback (most recent call last)
11 <ipython-input-126-b03deb420dcb> in <module>()
12 ----> 1 c.x = 0
13
14 <ipython-input-50-95b8686aa4bd> in __set__(self, obj, value)
15     20         if self.fset is None:

```

推荐阅读

个人贷款用户分析（特征分析&逻辑回归）

阅读 101

原地哈希算法

阅读 607

Ubuntu+GPU+Tensorflow运行tf-faster-rcnn【光纤分类项目】

阅读 181

假设检验和ABTEST（一）

阅读 121

数据集问题？

阅读 29

写下你的评论...

评论1

赞4

...

```
21 <ipython-input-116-379a4e5fa639> in x(self, value)
22     10 @x.setter
23     11 def x(self, value):
24     --> 12     assert value > 0
25     13     self._x = value
26     14
27
28 AssertionError:
```

与一般的属性访问不同，`c.x` 访问的已经不是简单的属性，而是相当于 `x.__get__(c)`，可以调用各种复杂方法对属性作检查、包装。

那么，描述器是怎样被访问到的呢？

调用描述器

有两类描述器：如果同时定义了 `__get__()` 和 `__set__()` 方法的描述器称为资料描述器(data descriptor)，仅定义了 `__get__()` 的描述器称为非资料描述器(non-data descriptor)。非资料描述器常用于类的方法，如常见的 `staticmethod` 和 `classmethod`，都是其应用。

如前文所说，描述器常在所有者类或其实例中被调用。

对于实例对象，`object.__getattr__()` 会把 `c.x` 转化为 `type(c).__dict__['x'].__get__(c, type(c))`。如果实例中有和描述器重名的属性 `x` 怎么办？资料和非资料描述器的区别在于，相对于实例字典的优先级不同。当描述器和实例字典中的某个属性重名，按访问优先级，资料描述器 > 同名实例字典中的属性 > 非资料描述器，优先级小的会被大的覆盖。上面的类 `c` 中，会优先访问资料描述器 `x`。下面将讲到，类的方法实际就是一个仅实现了 `__get__()` 的非资料描述器，所以如果实例 `c` 中同时定义了名为 `foo` 的方法和属性，那么 `c.foo` 访问的是属性而非方法。

对于类，`type.__getattr__()` 把 `C.x` 转化为 `C.__dict__['x'].__get__(None, C)`。

有几点需要牢记的：

1. 描述器被 `__getattr__()` 方法调用
2. 因而，重载 `__getattr__()` 可能会妨碍描述器被自动调用
3. `__getattr__()` 仅存在于继承自 `object` 的新式类之中
4. `object.__getattr__()` 和 `type.__getattr__()` 对 `__get__()` 的调用不一样
5. 资料描述器总会覆盖实例字典，即资料描述器具有最高优先级
6. 非资料描述器可能会被实例字典覆盖，即非资料描述器具有最低优先级

非资料描述器与类方法

Python 面向对象的特征建立在基于函数的环境之上。Python 用非资料描述器将二者无缝结合。

方法和普通函数唯一的区别就是，一般方法的第一个参数引用了当前实例，即通常命名为 `self` 的变量。

Python 中的函数，可以被认为是一个实现了 `__get__()` 的非资料描述器，用 Python 来描述就是：

```
1 class Function(object):
2     ...
```

推荐阅读

- 个人贷款用户分析（特征分析&逻辑回归）
阅读 101
- 原地哈希算法
阅读 607
- Ubuntu+GPU+Tensorflow运行tf-faster-rcnn【光纤分类项目】
阅读 181
- 假设检验和ABTEST（一）
阅读 121
- 数据集问题？
阅读 29



当函数作为属性被访问时，非类对象返回的是函数对象本身，而类对象则返回 `obj.f(*args)` 转变成 `f(obj, *args)`，把类调用 `klass.f(*args)` 转化为 `f(*args)`。

更多绑定和转换参见下表。

转换	从对象调用	从类调用
函数	<code>f(obj, *args)</code>	<code>f(*args)</code>
静态方法	<code>f(*args)</code>	<code>f(*args)</code>
类方法	<code>f(type(obj), *args)</code>	<code>f(klass, *args)</code>

静态方法是特殊的方法，可以无须实例化而在类中被直接调用，这时当然无法提供合法的 `self`。为此，需要实现 `staticmethod` 描述器，其 `__get__()` 返回的函数无需实例参数，其实也就是原样返回即可，可以用 Python 这样实现

```
1 class StaticMethod(object):
2     "Emulate PyStaticMethod_Type() in Objects/funcobject.c"
3
4     def __init__(self, f):
5         self.f = f
6
7     def __get__(self, obj, objtype=None):
8         return self.f
```

类方法是另一种特殊的方法，无需当前实例 `self`，但是需要当前类 `klass`（通常也写成 `cls`），纯 Python 实现如下：

```
1 class ClassMethod(object):
2     "Emulate PyClassMethod_Type() in Objects/funcobject.c"
3
4     def __init__(self, f):
5         self.f = f
6
7     def __get__(self, obj, klass=None):
8         if klass is None:
9             klass = type(obj)
10        def newfunc(*args):
11            return self.f(klass, *args)
12        return newfunc
```

参考资料

- [Descriptor HowTo Guide](#) 及其[中文翻译](#)

4人点赞 >

Python



ButteredCat 我的读书笔记：<https://butteredcat.github.io>
总资产1 (约0.11元) 共写了1.6W字 获得52个赞 共21个粉丝

关注

被以下专题收入，发现更多相似内容

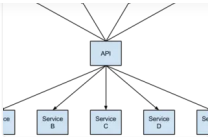
- PYTHON学习
- 开发相关知识
- Python进阶编程



Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...

 卡卡罗2017 阅读 103,897 评论 12 赞 125



Python描述器引导(翻译)

1.1. 摘要 定义描述器, 总结描述器协议, 并展示描述器是怎么被调用的。展示一个自定义的描述器和包括函数, 属性(...

 mutex73 阅读 119 评论 0 赞 2


百战程序员V1.2——尚学堂旗下高端培训_ Java1573题

百战程序员_ Java1573题 QQ群: 561832648489034603 掌握80%年薪20万掌握50%年薪...

 Albert陈凯 阅读 13,084 评论 3 赞 33


Java初级面试题

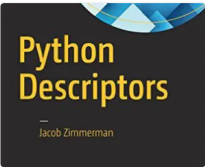
1. Java基础部分 基础部分的顺序: 基本语法, 类相关的语法, 内部类的语法, 继承相关的语法, 异常的语法, 线程的语...

 子非鱼_t 阅读 25,982 评论 17 赞 394

Python中的描述符

本文翻译自python descriptor guide 摘要 本文定义了描述符, 总结了其中的协议, 并且介绍如何调...

 大蟒传奇 阅读 601 评论 0 赞 5



推荐阅读

个人贷款用户分析（特征分析&逻辑回归）

阅读 101

原地哈希算法

阅读 607

Ubuntu+GPU+Tensorflow运行tf-faster-rcnn【光纤分类项目】

阅读 181

假设检验和ABTEST（一）

阅读 121

数据集问题？

阅读 29



写下你的评论...

 评论1  赞4 ...