获取对象信息

2.7旧版教程

阅读: 124553

当我们拿到一个对象的引用时，如何知道这个对象是什么类型、有哪些方法呢？

使用type()

首先，我们来判断对象类型，使用`type()`函数：

基本类型都可以用`type()`判断：

```
>>> type(123)
<class 'int'>
>>> type('str')
<class 'str'>
>>> type(None)
<type(None) 'NoneType'>
```

如果一个变量指向函数或者类，也可以用`type()`判断：

```
>>> type(abs)
<class 'builtin_function_or_method'>
>>> type(a)
<class '__main__.Animal'>
```

但是`type()`函数返回的是什么类型呢？它返回对应的Class类型。如果我们要在`if`语句中判断，就需要比较两个变量的type类型是否相同：

```
>>> type(123)==type(456)
True
>>> type(123)==int
True
>>> type('abc')==type('123')
True
>>> type('abc')==str
True
>>> type('abc')==type(123)
False
```

判断基本数据类型可以直接写`int`，`str`等，但如果要判断一个对象是否是函数怎么办？可以使用`types`模块中定义的常量：

```
>>> import types
>>> def fn():
...     pass
...
>>> type(fn)==types.FunctionType
True
```

```
>>> type(abs)==types.BuiltinFunctionType
True
>>> type(lambda x: x)==types.LambdaType
True
>>> type((x for x in range(10)))==types.GeneratorType
True
```

使用isinstance()

对于class的继承关系来说，使用`type()`就很不方便。我们要判断class的类型，可以使用`isinstance()`函数。

我们回顾上次的例子，如果继承关系是：

```
object -> Animal -> Dog -> Husky
```

那么，`isinstance()`就可以告诉我们，一个对象是否是某种类型。先创建3种类型的对象：

```
>>> a = Animal()
>>> d = Dog()
>>> h = Husky()
```

然后，判断：

```
>>> isinstance(h, Husky)
True
```

没有问题，因为`h`变量指向的就是Husky对象。

再判断：

```
>>> isinstance(h, Dog)
True
```

`h`虽然自身是Husky类型，但由于Husky是从Dog继承下来的，所以，`h`也还是Dog类型。换句话说，`isinstance()`判断的是一个对象是否是该类型本身，或者位于该类型的父继承链上。

因此，我们可以确信，`h`还是Animal类型：

```
>>> isinstance(h, Animal)
True
```

同理，实际类型是Dog的`d`也是Animal类型：

```
>>> isinstance(d, Dog) and isinstance(d, Animal)
True
```

但是，`d`不是Husky类型：

```
>>> isinstance(d, Husky)
False
```

能用`type()`判断的基本类型也可以用`isinstance()`判断：

```
>>> isinstance('a', str)
True
>>> isinstance(123, int)
True
>>> isinstance(b'a', bytes)
True
```

并且还可以判断一个变量是否是某些类型中的一种，比如下面的代码就可以判断是否是list或者tuple：

```
>>> isinstance([1, 2, 3], (list, tuple))
True
>>> isinstance((1, 2, 3), (list, tuple))
True
```

使用dir()

如果要获得一个对象的所有属性和方法，可以使用`dir()`函数，它返回一个包含字符串的list，比如，获得一个str对象的所有属性和方法：

```
>>> dir('ABC')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

类似`__xxx__`的属性和方法在Python中都是有特殊用途的，比如`__len__`方法返回长度。在Python中，如果你调用`len()`函数试图获取一个对象的长度，实际上，在`len()`函数内部，它自动去调用该对象的`__len__()`方法，所以，下面的代码是等价的：

```
>>> len('ABC')
3
>>> 'ABC'.__len__()
3
```

我们自己写的类，如果也想用`len(myObj)`的话，就自己写一个`__len__()`方法：

```
>>> class MyDog(object):
...     def __len__(self):
...         return 100
...
>>> dog = MyDog()
>>> len(dog)
100
```

剩下的都是普通属性或方法，比如`lower()`返回小写的字符串：

```
>>> 'ABC'.lower()
'abc'
```

仅仅把属性和方法列出来是不够的，配合`getattr()`、`setattr()`以及`hasattr()`，我们可以直接操作一个对象的状态：

```
>>> class MyObject(object):
...     def __init__(self):
...         self.x = 9
...     def power(self):
...         return self.x * self.x
...
>>> obj = MyObject()
```

紧接着，可以测试该对象的属性：

```
>>> hasattr(obj, 'x') # 有属性'x'吗?
True
>>> obj.x
9
>>> hasattr(obj, 'y') # 有属性'y'吗?
False
>>> setattr(obj, 'y', 19) # 设置一个属性'y'
>>> hasattr(obj, 'y') # 有属性'y'吗?
True
>>> getattr(obj, 'y') # 获取属性'y'
19
>>> obj.y # 获取属性'y'
19
```

如果试图获取不存在的属性，会抛出AttributeError的错误：

```
>>> getattr(obj, 'z') # 获取属性'z'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'MyObject' object has no attribute 'z'
```

可以传入一个default参数，如果属性不存在，就返回默认值：

```
>>> getattr(obj, 'z', 404) # 获取属性'z'，如果不存在，返回默认值404
404
```

也可以获得对象的方法：

```
>>> hasattr(obj, 'power') # 有属性'power'吗?
True
>>> getattr(obj, 'power') # 获取属性'power'
<bound method MyObject.power of <__main__.MyObject object at 0x10077a6a0>>
>>> fn = getattr(obj, 'power') # 获取属性'power'并赋值到变量fn
>>> fn # fn指向obj.power
<bound method MyObject.power of <__main__.MyObject object at 0x10077a6a0>>
>>> fn() # 调用fn()与调用obj.power()是一样的
81
```

小结

通过内置的一系列函数，我们可以对任意一个Python对象进行剖析，拿到其内部的数据。要注意的是，只有在不知道对象信息的时候，我们才会去获取对象信息。如果可以直接写：

```
sum = obj.x + obj.y
```

就不要写：

```
sum = getattr(obj, 'x') + getattr(obj, 'y')
```

一个正确的用法的例子如下：

```
def readImage(fp):
    if hasattr(fp, 'read'):
        return readData(fp)
    return None
```

假设我们希望从文件流fp中读取图像，我们首先要判断该fp对象是否存在read方法，如果存在，则该对象是一个流，如果不存在，则无法读取。`hasattr()`就派上了用场。

请注意，在Python这类动态语言中，根据鸭子类型，有`read()`方法，不代表该fp对象就是一个文件流，它也可能是网络流，也可能是内存中的一个字节流，但只要`read()`方法返回的是有效的图像数据，就不影响读取图像的功能。

参考源码

get_type.py

attrs.py

感觉本站内容不错，读后有收获？

¥ 我要小额赞助，鼓励作者写出更好的教程

还可以分享给朋友

分享　赶快成为第一个分享的人吧

继承和多态

实例属性和类属性

评论

发表评论

Sign In to Make a Comment

友情链接: 中华诗词 - 阿里云 - SICP - 4clojure