9.16 *args和**kwargs的强制参数签名¶

问题¶

你有一个函数或方法，它使用*args和**kwargs作为参数，这样使得它比较通用， 但有时候你想检查传递进来的参数是不是某个你想要的类型。

解决方案¶

对任何涉及到操作函数调用签名的问题，你都应该使用 `inspect` 模块中的签名特性。 我们最主要关注两个类：`Signature` 和 `Parameter` 。下面是一个创建函数前面的交互例子：

```
>>> from inspect import Signature, Parameter
>>> # Make a signature for a func(x, y=42, *, z=None)
>>> parms = [ Parameter('x', Parameter.POSITIONAL_OR_KEYWORD),
...          Parameter('y', Parameter.POSITIONAL_OR_KEYWORD, default=42),
...          Parameter('z', Parameter.KEYWORD_ONLY, default=None) ]
>>> sig = Signature(parms)
>>> print(sig)
(x, y=42, *, z=None)
>>>
```

一旦你有了一个签名对象，你就可以使用它的 `bind()` 方法很容易的将它绑定到 `*args` 和 `**kwargs` 上去。 下面是一个简单的演示：

```
>>> def func(*args, **kwargs):
...     bound_values = sig.bind(*args, **kwargs)
...     for name, value in bound_values.arguments.items():
...         print(name,value)
...
>>> # Try various examples
>>> func(1, 2, z=3)
x 1
y 2
z 3
>>> func(1)
x 1
>>> func(1, z=3)
x 1
z 3
>>> func(y=2, x=1)
x 1
y 2
>>> func(1, 2, 3, 4)
Traceback (most recent call last):
...
    File "/usr/local/lib/python3.3/inspect.py", line 1972, in _bind
        raise TypeError('too many positional arguments')
TypeError: too many positional arguments
>>> func(y=2)
Traceback (most recent call last):
...
    File "/usr/local/lib/python3.3/inspect.py", line 1961, in _bind
        raise TypeError(msg) from None
TypeError: 'x' parameter lacking default value
>>> func(1, y=2, x=3)
Traceback (most recent call last):
...
    File "/usr/local/lib/python3.3/inspect.py", line 1985, in _bind
        '{arg!r}'.format(arg=param.name))
TypeError: multiple values for argument 'x'
>>>
```

可以看出来，通过将签名和传递的参数绑定起来，可以强制函数调用遵循特定的规则，比如必填、默认、重复等等。

下面是一个强制函数签名更具体的例子。在代码中，我们在基类中先定义了一个非常通用的 `__init__()` 方法， 然后我们强制所有的子类必须提供一个特定的参数签名。

```
from inspect import Signature, Parameter

def make_sig(*names):
    parms = [Parameter(name, Parameter.POSITIONAL_OR_KEYWORD)
             for name in names]
    return Signature(parms)

class Structure:
    __signature__ = make_sig()
    def __init__(self, *args, **kwargs):
        bound_values = self.__signature__.bind(*args, **kwargs)
        for name, value in bound_values.arguments.items():
            setattr(self, name, value)

# Example use
class Stock(Structure):
    __signature__ = make_sig('name', 'shares', 'price')

class Point(Structure):
    __signature__ = make_sig('x', 'y')
```

下面是使用这个 `Stock` 类的示例：

```
>>> import inspect
>>> print(inspect.signature(Stock))
(name, shares, price)
>>> s1 = Stock('ACME', 100, 490.1)
>>> s2 = Stock('ACME', 100)
Traceback (most recent call last):
...
TypeError: 'price' parameter lacking default value
>>> s3 = Stock('ACME', 100, 490.1, shares=50)
Traceback (most recent call last):
...
TypeError: multiple values for argument 'shares'
>>>
```

讨论¶

在我们需要构建通用函数库、编写装饰器或实现代理的时候，对于 `*args` 和 `**kwargs` 的使用是很普遍的。 但是，这样的函数有一个缺点就是当你想要实现自己的参数检验时，代码就会笨拙混乱。在8.11小节里面有这样一个例子。 这时候我们可以通过一个签名对象来简化它。

在最后的一个方案实例中，我们还可以通过使用自定义元类来创建签名对象。下面演示怎样来实现：

```
from inspect import Signature, Parameter

def make_sig(*names):
    parms = [Parameter(name, Parameter.POSITIONAL_OR_KEYWORD)
             for name in names]
    return Signature(parms)

class StructureMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        clsdict['__signature__'] = make_sig(*clsdict.get('_fields',[]))
        return super().__new__(cls, clsname, bases, clsdict)

class Structure(metaclass=StructureMeta):
    _fields = []
    def __init__(self, *args, **kwargs):
        bound_values = self.__signature__.bind(*args, **kwargs)
        for name, value in bound_values.arguments.items():
            setattr(self, name, value)

# Example
```

```
class Stock(Structure):
    _fields = ['name', 'shares', 'price']

class Point(Structure):
    _fields = ['x', 'y']
```

当我们自定义签名的时候，将签名存储在特定的属性 `__signature__` 中通常是很有用的。 这样的话，在使用 `inspect` 模块执行内省的代码就能发现签名并将它作为调用约定。

```
>>> import inspect
>>> print(inspect.signature(Stock))
(name, shares, price)
>>> print(inspect.signature(Point))
(x, y)
>>>
```