9.14 捕获类的属性定义顺序¶

问题¶

你想自动记录一个类中属性和方法定义的顺序， 然后可以利用它来做很多操作（比如序列化、映射到数据库等等）。

解决方案¶

利用元类可以很容易的捕获类的定义信息。下面是一个例子，使用了一个OrderedDict来记录描述器的定义顺序：

```
from collections import OrderedDict

# A set of descriptors for various types
class Typed:
    _expected_type = type(None)
    def __init__(self, name=None):
        self._name = name

    def __set__(self, instance, value):
        if not isinstance(value, self._expected_type):
            raise TypeError('Expected ' + str(self._expected_type))
        instance.__dict__[self._name] = value

class Integer(Typed):
    _expected_type = int

class Float(Typed):
    _expected_type = float

class String(Typed):
    _expected_type = str

# Metaclass that uses an OrderedDict for class body
class OrderedMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        d = dict(clsdict)
        order = []
        for name, value in clsdict.items():
            if isinstance(value, Typed):
                value._name = name
                order.append(name)
        d['_order'] = order
        return type.__new__(cls, clsname, bases, d)

    @classmethod
    def __prepare__(cls, clsname, bases):
        return OrderedDict()
```

在这个元类中，执行类主体时描述器的定义顺序会被一个 `OrderedDict` 捕获到， 生成的有序名称从字典中提取出来并放入类属性 `_order` 中。这样的话类中的方法可以通过多种方式来使用它。 例如，下面是一个简单的类，使用这个排序字典来实现将一个类实例的数据序列化为一行CSV数据：

```
class Structure(metaclass=OrderedMeta):
    def as_csv(self):
        return ','.join(str(getattr(self,name)) for name in self._order)

# Example use
class Stock(Structure):
    name = String()
    shares = Integer()
    price = Float()

    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

我们在交互式环境中测试一下这个Stock类：

```
>>> s = Stock('GOOG',100,490.1)
>>> s.name
'GOOG'
>>> s.as_csv()
'GOOG,100,490.1'
>>> t = Stock('AAPL','a lot', 610.23)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "dupmethod.py", line 34, in __init__
TypeError: shares expects <class 'int'>
>>>
```

讨论¶

本节一个关键点就是OrderedMeta元类中定义的 `__prepare__()` 方法。 这个方法会在开始定义类和它的父类的时候被执行。它必须返回一个映射对象以便在类定义体中被使用到。 我们这里通过返回了一个OrderedDict而不是一个普通的字典，可以很容易的捕获定义的顺序。

如果你想构造自己的类字典对象，可以很容易的扩展这个功能。比如，下面的这个修改方案可以防止重复的定义：

```
from collections import OrderedDict

class NoDupOrderedDict(OrderedDict):
    def __init__(self, clsname):
        self.clsname = clsname
        super().__init__()
    def __setitem__(self, name, value):
        if name in self:
            raise TypeError('{} already defined in {}'.format(name, self.clsname))
        super().__setitem__(name, value)

class OrderedMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        d = dict(clsdict)
        d['_order'] = [name for name in clsdict if name[0] != '_']
        return type.__new__(cls, clsname, bases, d)

    @classmethod
    def __prepare__(cls, clsname, bases):
        return NoDupOrderedDict(clsname)
```

下面我们测试重复的定义会出现什么情况：

```
>>> class A(metaclass=OrderedMeta):
... def spam(self):
... pass
... def spam(self):
... pass
...
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "<stdin>", line 4, in A
    File "dupmethod2.py", line 25, in __setitem__
        (name, self.clsname))
TypeError: spam already defined in A
>>>
```

最后还有一点很重要，就是在 `__new__()` 方法中对于元类中被修改字典的处理。 尽管类使用了另外一个字典来定义，在构造最终的 `class` 对象的时候， 我们仍然需要将这个字典转换为一个正确的 `dict` 实例。 通过语句 `d = dict(clsdict)` 来完成这个效果。

对于很多应用程序而已，能够捕获类定义的顺序是一个看似不起眼却又非常重要的特性。 例如，在对象关系映射中，我们通常会看到下面这种方式定义的类：

```
class Stock(Model):
```

```
    name = String()
    shares = Integer()
    price = Float()
```

在框架底层，我们必须捕获定义的顺序来将对象映射到元组或数据库表中的行（就类似于上面例子中的 `as_csv()` 的功能）。 这节演示的技术非常简单，并且通常会比其他类似方法（通常都要在描述器类中维护一个隐藏的计数器）要简单的多。