Metaclasses
How to Silently Extend Classes (Part 2)

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it

Metaclasses
What’s a Metaclass?

Metaclasses are a mechanism to gain a high-level of control over how a set of classes work:
- They permit to intercept and augment class creation;
- They provide an API to insert extra-logic at the end of class statement;
- They provide a general protocol to manage class objects in a program.

Note,
- The added logic does not rebind the class name to a decorator callable, but rather routes creation of the class itself to specialized logic.
- Metaclasses add code to be run at class creation time and not at instance creation time.

The Metaclass Model

Classes Are Instances of type

```
>>> type(circle)
<class 'type'>
```

```
>>> circle.__class__
<class 'type'>
```

```
>>> c = circle(3)
```

```
>>> type(c)
<class 'circle.circle'>
```

```
>>> c.__class__
<class 'circle.circle'>
```

```
>>> type([])
<class 'list'>
```

```
>>> type(type([]))
<class 'type'>
```

Metaclasses Are Subclasses of type
- `type` is a class that generates user-defined classes.
- Metaclasses are subclasses of the `type` class.
- Class objects are instances of the `type` class, or a subclass thereof.
- Instance objects are generated from a class.

Class Statement Protocol
- At the end of class statement, after filling `__dict__`, Python calls

```
class = type(classname, superclasses, attributedict)
```

to create the `class` object.

- `type` object defines a `__call__` operator that calls `__new__` (to create class objects) and `__init__` (to create instance objects) when `type` object is called.

Declaring Metaclasses
To create a class with a custom metaclass you have just to list the desired metaclass as a keyword argument in the `class` header.

```
class Spam(metaclass=Meta): pass
```

Coding Metaclasses
- `subtype` `type`
- Override `__new__`, `__init__` and `__call__` operators
Metaclasses

The Metaclass Declaring & Coding (Cont'd)

**Definition**
- Metaclasses inherit from the type class
- Metaclass declarations are inherited by subclasses
- Metaclass attributes are not inherited by class instances

**Applying Decorators to Methods**

```python
def onDecorator(func):
    def onCall(*args, **kwargs):
        calls += 1
        result = func(*args, **kwargs)
        onCall.alltime += elapsed
        onCall(alltime, func.__name__, elapsed, onCall.alltime)

onCall.alltime = 0
```

**Metaclasses vs Superclasses**

- Metaclass attributes are not inherited by class instances
- Metaclass declarations are inherited by subclasses

```python
class MetaAndSuper:
    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class Super(metaclass=MetaOne):
    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class Customer2(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper2(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper3(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper4(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper5(Super):
    __init__(self, value):
        self.value = value

X = C()
Y = C()
X.spam()  # Defined in C
Y.spam()  # Inherited from Super
X.eggs()  # Not inherited from metaclasses
```

**Applying Decorators to Methods**

```python
class MetaAndSuper:
    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class MetaAndSuper2(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper3(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper4(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper5(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper6(Super):
    __init__(self, value):
        self.value = value

X = C()
Y = C()
X.spam()  # Defined in C
Y.spam()  # Inherited from Super
X.eggs()  # Not inherited from metaclasses
```

**Metaclass-Based Augmentation**

```python
class MetaOne(type):
    def __new__(meta, classname, supers, classdict):
        print('In MetaOne.new:', classname, supers, classdict, sep='
...')
        data = 1  # Class data attribute
        classdict['ham'] = hamfunc
        classdict['eggs'] = eggsfunc
        return type.__new__(meta, classname, supers, classdict)

    def __init__(Class, classname, supers, classdict):
        print('In MetaOne init:', classname, supers, classdict, sep='
...')
        # Class method attribute
        pass

class Extender(type):
    def __new__(meta, classname, supers, classdict):
        print('In MetaNew.new:', classname, supers, classdict, sep='
...')
        self.value = value
        return type.__new__(meta, classname, supers, classdict)

    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class Client1(metaclass=Extender):
    __init__(self, value):
        self.value = value

    def spam(self):
        return self.value * 2

    def eggs(self):
        return self.value * 4

    def hamfunc(obj, value):
        return obj.value * 4

    def eggsfunc(obj):
        return obj.value * 4
```

**Applying Decorators to Methods**

```python
class MetaAndSuper:
    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class MetaAndSuper2(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper3(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper4(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper5(Super):
    __init__(self, value):
        self.value = value

X = C()
Y = C()
X.spam()  # Defined in C
Y.spam()  # Inherited from Super
X.eggs()  # Not inherited from metaclasses
```

**Applying Decorators to Methods**

```python
class MetaAndSuper:
    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class MetaAndSuper2(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper3(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper4(Super):
    __init__(self, value):
        self.value = value

class MetaAndSuper5(Super):
    __init__(self, value):
        self.value = value

X = C()
Y = C()
X.spam()  # Defined in C
Y.spam()  # Inherited from Super
X.eggs()  # Not inherited from metaclasses
```

**Metaclass-Based Augmentation**

```python
class MetaOne(type):
    def __new__(meta, classname, supers, classdict):
        print('In MetaOne.new:', classname, supers, classdict, sep='
...')
        data = 1  # Class data attribute
        classdict['ham'] = hamfunc
        classdict['eggs'] = eggsfunc
        return type.__new__(meta, classname, supers, classdict)

    def __init__(Class, classname, supers, classdict):
        print('In MetaOne init:', classname, supers, classdict, sep='
...')
        # Class method attribute
        pass

class Extender(type):
    def __new__(meta, classname, supers, classdict):
        print('In MetaNew.new:', classname, supers, classdict, sep='
...')
        self.value = value
        return type.__new__(meta, classname, supers, classdict)

    def spam(self):
        print('spam')

    def eggs(self):
        print('eggs')

class Client1(metaclass=Extender):
    __init__(self, value):
        self.value = value

    def spam(self):
        return self.value * 2

    def eggs(self):
        return self.value * 4

    def hamfunc(obj, value):
        return obj.value * 4

    def eggsfunc(obj):
        return obj.value * 4
```
Metaclasses
Applying Decorators to Methods

```python
from decorators.tracer import tracer
import tracer

class Person:
    @tracer
    def __init__(self, name, pay):
        self.name = name
        self.pay = pay

    def giveRaise(self, percent):
        self.pay *= (1.0 + percent)

    def lastName(self):
        return self.name.split()[-1]

sue = Person('Sue Jones', 100000)
sue.giveRaise(.10)
sue = Person('Sue Jones', 100000)
sue.giveRaise(.10)

print('--- tracer')
print(bob.lastName(), sue.lastName())
print(sue.pay)
print(bob.name, sue.name)

[21:10] cazzola@ulik:/~esercizi-pa/metaclass>python3 Person3.py
```

References


