15. Метапрограмиране

28 ноември 2023

Но първо - още малко метаобектен протокол...

Метаатрибути

- `__dict__`
- `__slots__`
- `__class__`
- `__globals__`
- `__name__`

Функции

- `__code__`
- `__call__(self, *args, **kwargs)`

Дескриптори

- `__get__(self, instance, owner)`
- `__set__(self, instance, value)`
- `__delete__(self, instance)`

Дескриптори: Теория

```
def __get__(self, instance, owner): ...

def __set__(self, instance, value): ...

def __delete__(self, instance): ...
```

- Викат се върху обект, който бива достъпван като атрибут на друг обект.
- Ако класът `A` има атрибут `foo`, със стойност обект от тип `B`, достъпвайки `A.foo` ще се извика `__get__` на `B`.
- Опитвайки се да го предефинираме, удряме `__set__`.
- Познайте какво става ако опитаме да го изтрием с `del`.

Дескриптори: Практика

```
class B:
    def __get__(self, instance, owner):
        return "You came to the wrong neighborhood, motherflower!"

    def __set__(self, instance, value):
        print("What!? You think you can change my personality just like that!?")

    def __delete__(self, instance):
        print("Can't touch me!")

class A:
    foo = B()

a = A()
print(a.foo)
a.foo = 'bar'
del a.foo
```

Bound methods

```
>>> increment = (1).__add__
>>> map(increment, [0, 1, 2, 3])
[1, 2, 3, 4]
```

- "Закача" инстанция за метод
- Прилича на частично прилагане на функция
- Единствено `self` може да бъде приложен

Bound methods: Проста имплементација!

```
class MyMethod:

    def __init__(self, func):
        self.func = func

    def __get__(self, instance, owner):
        if instance:
            return lambda: self.func(instance)
        else:
            return lambda explicit_instance: self.func(explicit_instance)

class Python:
    name = 'Monty'
    greet = MyMethod(lambda self: 'My name issss %s' % self.name)
```

Bound methods: Проста имплементација!

```
snake = Python()
snake.greet() # 'My name issss Monty'
snake.name = 'Nagini'
Python.greet() # TypeError: <lambda>() takes exactly 1 argument (0 given)
Python.greet(snake) # 'My name issss Nagini'
```

Импортиране на модули

```
import module
```

...съответства на...

```
module = __import__('module')
```

Конструиране

- `__new__(cls, *args, **kwargs)`
- `__init__(self, *args, **kwargs)`

__new__

`__new__` е истинският конструктор на вашите обекти. `__init__` е само инициализатор.

```
class Vector(tuple):
  def __new__(klass, x, y):
    return tuple.__new__(klass, (x, y))

  def __add__(self, other):
    if not isinstance(other, Vector):
      return NotImplemented
    return Vector(self[0] + other[0], self[1] + other[1])
```

Method resolution order

Редът за обхождане на базови класове

```
class A(int): pass

class B: pass

class C(A, B, int): pass

C.__mro__ # или C.mro()
# <class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>,
# <class 'int'>, <class 'object'>
```

Method resolution order (2)

- Използва алгоритъм наречен C3 linearization
- https://en.wikipedia.org/wiki/C3_linearization
- https://dl.acm.org/doi/pdf/10.1145/236337.236343

Други (без конкретен ред)

- `__sizeof__`
- `__weakrefoffset__`
- `__base__`
- `__bases__`
- `__basicsize__`
- `__module__`
- `__package__`
- `__all__`
- `__builtins__`
- `__subclasses__`
- `__subclasshook__`
- `__subclasscheck__`
- `__file__`
- `__format__`
- `__loader__`
- `Ellipsis/...`

Преговор!

```
isinstance(3, int)        # True
isinstance(3, object)     # True
isinstance(3, str)        # False
isinstance(int, type)     # True
isinstance(3, type)       # False
isinstance('hello', str)  # True
```

Преговор?

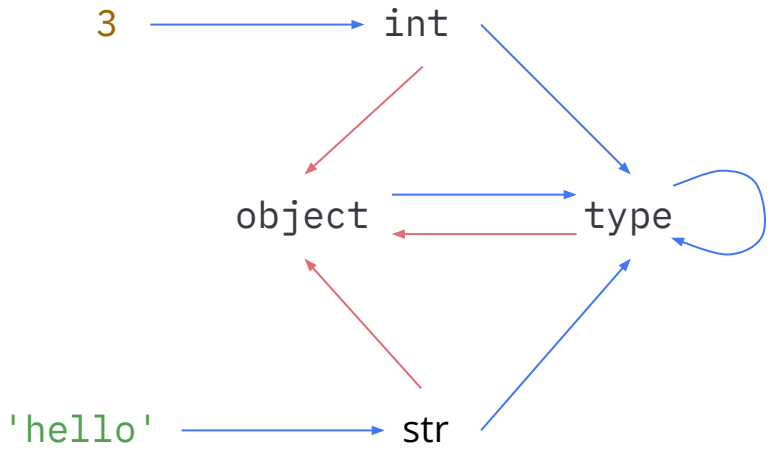
```
issubclass(int, object)   # True
issubclass(object, int)   # False
issubclass(int, int)      # True
issubclass(3, int)        # TypeError: [...] must be a class
issubclass(int, type)     # False
```

Преговор?!?!?!?!

```
isinstance(type, type)          # True
issubclass(type, type)          # True
isinstance(object, object)      # True
issubclass(object, object)      # True
isinstance(type, object)        # True
issubclass(type, object)        # True
isinstance(object, type)        # True
issubclass(object, type)        # False
```

isintance изобразен

- `issubclass` обхожда `__bases__` и търси съвпадение
- `isinstance(x, t) == issubclass(x.__class__, t)`



Метакласове

- Всичко в Пайтън е обект, включително и класовете
- Всеки обект е инстанция на някакъв клас, включително и класовете
- Класовете на класовете си имат специално име - метакласове
- `type` е метаклас на току-що разгледаните типове

Какво всъщност е *type*?

- Без аргументи е просто класът type
- С един аргумент `type(x)` връща типа на `x`
- С три аргумента се конструира инстанция на `type`:
  `type(name, bases, dict)`

Пример за type(name, bases, dict)

```
def init_person(self, name):
    self.name = name

def say_hi(self):
    print(f'Hi, My name is {self.name}')

Person = type('Person', (), {
    '__init__': init_person,
    'say_hi': say_hi,
})

Person('George').say_hi()
```

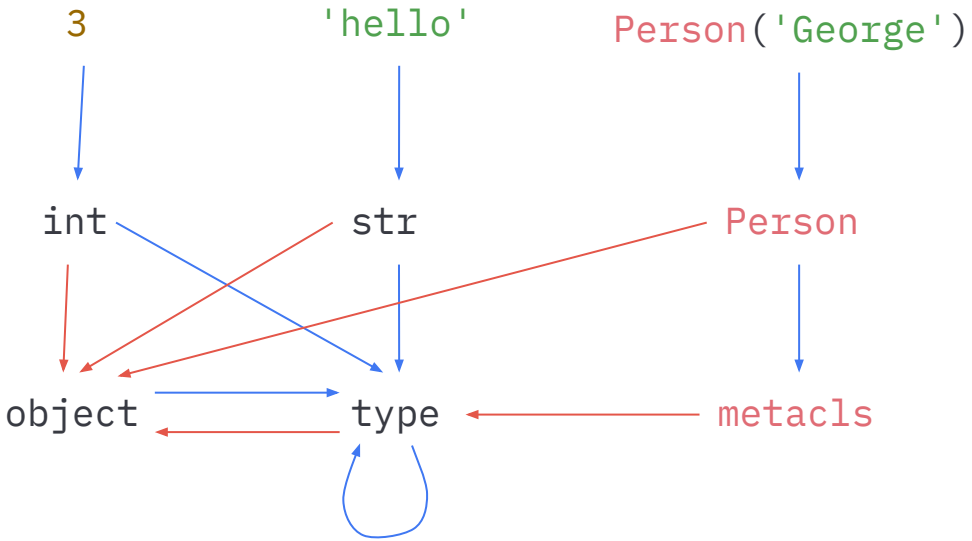
Наследяване от type

```
class metacls(type):
    def __new__(cls, name, bases, attr_dict):
        dict['say_bye'] = lambda self: print('bye')
        return type.__new__(cls, name, bases, attr_dict)

Person = metacls('Person', (), {
    '__init__': init_person,
    'say_hi': say_hi,
})

Person('George').say_bye()
```

metacls изобразен



Синтактична захар

```
class Foo(A, B, C, metaclass=Bar):
    x = 1
    y = 2

# е захар за:
Foo = Bar('Foo', (A, B, C), {'x': 1, 'y': 2})
```

Синтактична захар (2)

```
class Person(metaclass=metacls):
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print(f'Hi, My name is {self.name}')

Person('George').say_bye() # bye
```

Безкористен питон

```
def without_ego(func):
    def wrapped(self, *args, **kwargs):
        old_self = func.__globals__.get('self')
        func.__globals__['self'] = self
        result = func(*args, **kwargs)
        func.__globals__['self'] = old_self
        return result
    wrapped.__name__ = func.__name__
    return wrapped

class selfless(type):
    def __new__(cls, name, bases, attrs):
        for key, value in attrs.items():
            if hasattr(value, '__call__'):
                attrs[key] = without_ego(value)
        return type.__new__(cls, name, bases, attrs)
```

Безкористна нинджа!

```
class Person(metaclass=selfless):
    def __init__(name):
        self.name = name

    def say_hi():
        print(f'Hi, I am {self.name}')

Person("忍者").say_hi()
```

Въпроси?