Python
Arcanes du langage

Nicolas Delestre

INSA
ROUEN NORMANDIE

Cycle de vie d'un objet

La création

- Rappel : `__init__` ne crée pas l'objet, elle l'initialise, c'est `__new__` qui le créé
- « On utilise rarement `__new__`. Les deux cas principaux sont :
  - si on hérite d'un type immutable (`str`, `int`, `tuple`, etc), `__new__` est le seul endroit où on puisse initialiser l'objet.
  - dans le cas des métaclasses. »[a]

a. `http://sametmax.com/la-difference-entre-__new__-et-__init__-en-python/`

La suppression d'un objet

- Lorsque le *garbage collector* supprime un objet, la méthode `__del__` est appelée juste avant

INSA ROUEN NORMANDIE

La gestion des attributs 1 / 4

- `getattr(obj, nom)`, `setattr(obj, nom, valeur)` et `delattr(obj, nom, valeur)` permettent d'utiliser l'attribut `nom` d'`obj` (objet, classe, module), par exemple :
  - la demande d'interprétation de `o.a` exécute `getattr(o,a)`
- `__getattribute__(self, nom)`, `__getattr__(self, nom)`, `__setattr__(self, nom, valeur)` et `__delattr__(self, nom)` permettent de modifier le comportement de l'utilisation des attributs
  - l'exécution `getattr(o,a)` exécute `o.__getattribute__(a)`. Si `a` n'est pas déclaré pour `o`, alors c'est `o.__getattr__(a)` qui est exécuté (par défaut levé de l'exception `AttributeError`)
- Les attributs d'un objet sont stockés dans l'attribut `__dict__`
- Un attribut `__a` d'une classe `C` est nommé `_C__a`

INSA ROUEN NORMANDIE

La gestion des attributs 2 / 4

Script attributs.py

```
1  #!/usr/bin/env python
2
3  class Foo(object):
4      def __init__(self):
5          self.a = 1
6          self.__b = 2
7
8      def __getattribute__(*args):
9          print(f"Appel de __getattribute__: {args}")
10         return object.__getattribute__(*args)
11
12     def __setattr__(*args):
13         print(f"Appel de __setattr__: {args}")
14         return object.__setattr__(*args)
15
16     def m(self):
17         print("Appel de m")
18         print(f"a : {self.a}")
19         print(f"__b : {self.__b}")
20
21 if __name__ == "__main__":
22     o = Foo()
23     o.m()
24     print(f"o.__dict__ : {o.__dict__}")
25     print(f"Foo.__dict__ : {Foo.__dict__}")
26     print(f"o.a : {o.a}")
27     print(f"o.__b : {o.__b}")
```

La gestion des attributs 3 / 4

Exécution du script attributs.py

```
$ python attributs.py
Appel de __setattr__: (<__main__.Foo object at 0x7f866de2f940>, 'a', 1)
Appel de __setattr__: (<__main__.Foo object at 0x7f866de2f940>, '_Foo__b', 2)
Appel de __getattribute__: (<__main__.Foo object at 0x7f866de2f940>, 'm')
Appel de m
Appel de __getattribute__: (<__main__.Foo object at 0x7f866de2f940>, 'a')
a : 1
Appel de __getattribute__: (<__main__.Foo object at 0x7f866de2f940>, '_Foo__b')
__b : 2
Appel de __getattribute__: (<__main__.Foo object at 0x7f866de2f940>, '__dict__')
o.__dict__ : {'a': 1, '_Foo__b': 2}
Foo.__dict__ : {'__module__': '__main__',
  '__init__': <function Foo.__init__ at 0x7f0be9ead950>,
  '__getattribute__': <function Foo.__getattribute__ at 0x7f0be9ead9d8>,
  '__setattr__': <function Foo.__setattr__ at 0x7f0be9eada60>,
  'm': <function Foo.m at 0x7f0be9eadae8>,
  '__dict__': <attribute '__dict__' of 'Foo' objects>,
  '__weakref__': <attribute '__weakref__' of 'Foo' objects>,
  '__doc__': None}
```

Exécution du script attributs.py (suite)

```
Appel de __getattribute__: (<__main__.Foo object at 0x7f866de2f940>, 'a')
o.a : 1
Appel de __getattribute__: (<__main__.Foo object at 0x7f866de2f940>, '__b')
Traceback (most recent call last):
  File "prive.py", line 26, in <module>
    print(f"o.__b : {o.__b}")
  File "prive.py", line 10, in __getattribute__
    return object.__getattribute__(*args)
AttributeError: 'Foo' object has no attribute '__b'
```

INSA ROUEN NORMANDIE

Métaclasse

- Tout est objet en python. Les classes sont des objets. Les classes sont donc instances de classe que l'on nomme métaclasse. On peut créer des métaclasses pour ajouter ou modifier le comportement de classes.
- `type` est la métaclasse par défaut :
  - `type` est sa propre instance
  - `type` est *appelable* : la création d'une classe appelle en fait la fonction `type(nom, super_classes, dictionnaire_attributs)`
  - On crée une nouvelle métaclasse en créant une sous classe de `type`
- On créé une classe `C` de métaclasse `M` grâce au paramètre formel `metaclass` :

```
class C(object, metaclass=M)
```

Méta-programmation 2 / 7

Pourquoi créer une métaclasse ?

- On crée des métaclasses lorsque l'on veut modifier le comportement des classes :
  - durant leur création : `__new__`, `__init__`
  - durant la création de leurs objets : `__call__`
  - durant l'accès à leurs attributs de classe : `__getattr__`, `__setattr__`, `__delattr__`, `__getattribute__`

Comment crée-t-on une métaclasse ?

- On crée des métaclasses :
  - en créant une classe qui hérite de `type` (et non pas d'`object`)
  - en rédefinissant les méthodes que l'on veut adapter

Attention

Ne pas oublier d'appeler la méthode (surchargée) de la classe `type`

Script objets.py : une nouvelle métaclasse M

```
1  #!/usr/bin/env python
2
3  class M(type):
4      def __new__(*args):
5          print(f"Appel de __new__ de M: {args}")
6          return type.__new__(*args)
7
8      def __init__(*args):
9          print(f"Appel de __init__ de M: {args}")
10         return type.__init__(*args)
11
12     def __getattribute__(*args):
13         print(f"Appel de __getattribute__ de M: {args}")
14         return type.__getattribute__(*args)
15
16     def __getattr__(*args):
17         print(f"Appel de __getattr__ de M: {args}")
18         raise AttributeError()
19
20     def __call__(*args, **kargs):
21         print(f"Appel de __call__ de M: {args}")
22         return type.__call__(*args, **kargs)
```

ROUEN NORMANDIE

Script objets.py : une classe C instance de M

```
24 print("Création de la classe C")
25 class C(object, metaclass=M):
26     ca = 10
27
28     def __new__(*args):
29         print(f"Appel de __new__ de C: {args}")
30         return object.__new__(*args)
31
32     def __init__(self):
33         print(f"Appel de __init__ de C: {self}")
34         self.a = 1
35
36     def __getattribute__(*args):
37         print(f"Appel de __getattribute__ de C: {args}")
38         return object.__getattribute__(*args)
39
40     def __getattr__(*args):
41         print(f"Appel de __getattr__ de C: {args}")
42         return None
```

INSA ROUEN NORMANDIE

Script `objets.py` : le main

```
44 if __name__ == "__main__":
45     print("Instanciation de c")
46     c = C()
47     print("obtention de c.a")
48     print(c.a)
49     print("obtentino de c.b")
50     print(c.b)
51     print("obtention de c.ca")
52     print(c.ca)
53     print("obtention de C.ca")
54     print(C.ca)
```

Exécution du script objets.py

```
$ python objets.py
Création de la classe C
Appel de __new__ de M: (<class '__main__.M'>, 'C', (<class 'object'>,),
  {'__module__': '__main__',
  '__qualname__': 'C',
  'ca': 10,
  '__new__': <function C.__new__ at 0x7f95f46cabf8>,
  '__init__': <function C.__init__ at 0x7f95f46cac80>,
  '__getattribute__': <function C.__getattribute__ at 0x7f95f46cad08>,
  '__getattr__': <function C.__getattr__ at 0x7f95f46cad90>})

Appel de __init__ de M: (<class '__main__.C'>, 'C', (<class 'object'>,),
  {'__module__': '__main__',
  '__qualname__': 'C',
  'ca': 10,
  '__new__': <function C.__new__ at 0x7f95f46cabf8>,
  '__init__': <function C.__init__ at 0x7f95f46cac80>,
  '__getattribute__': <function C.__getattribute__ at 0x7f95f46cad08>,
  '__getattr__': <function C.__getattr__ at 0x7f95f46cad90>})
```

Exécution du script objets.py (suite)

```
Instanciation de c
Appel de __call__ de M: (<class '__main__.C'>,)
Appel de __getattribute__ de M: (<class '__main__.C'>, '__new__')
Appel de __new__ de C: (<class '__main__.C'>,)
Appel de __init__ de C: : <__main__.C object at 0x7f382c4317b8>
obtention de c.a
Appel de __getattribute__ de C: (<__main__.C object at 0x7f95f5f48860>, 'a')
1
obtentino de c.b
Appel de __getattribute__ de C: (<__main__.C object at 0x7f95f5f48860>, 'b')
Appel de __getattr__ de C: (<__main__.C object at 0x7f95f5f48860>, 'b')
None
obtention de c.ca
Appel de __getattribute__ de C: (<__main__.C object at 0x7f95f5f48860>, 'ca')
10
obtention de C.ca
Appel de __getattribute__ de M: (<class '__main__.C'>, 'ca')
10
```

ROUEN NORMANDIE