super is not as simple as you thought

@mozillazg

super 很简单呀

不就是用来调用父类方法的嘛?

```
super(B, self).foo()
```

Too Young Too Simple

简单的例子

```python
class A(object):
    def __init__(self):
        self.n = 2

    def add(self, m):
        print('self is {0} @A.add'.format(self))
        self.n += m


class B(A):
    def __init__(self):
        self.n = 3

    def add(self, m):
        print('self is {0} @B.add'.format(self))
        super(B, self).add(m)
        self.n += 3

b = B()
b.add(2)
print(b.n)
```

输出

```python
class A(object):
    def __init__(self):
        self.n = 2

    def add(self, m):
        print('self is {0} @A.add'.format(self))
        self.n += m


class B(A):
    def __init__(self):
        self.n = 3

    def add(self, m):
        print('self is {0} @B.add'.format(self))
        super(B, self).add(m)
        self.n += 3

b = B()
b.add(2)
print(b.n)
```

输出:

```
self is <__main__.B object at 0x104d955d0> @B.add
self is <__main__.B object at 0x104d955d0> @A.add
8
```

复杂的例子

```
class A(object):
    def __init__(self):
        self.n = 2

    def add(self, m):
        print('self is {0} @A.add'.format(self))
        self.n += m

class C(A):
    def __init__(self):
        self.n = 4

    def add(self, m):
        print('self is {0} @C.add'.format(self))
        super(C, self).add(m)
        self.n += 4

class B(A):
    def __init__(self):
        self.n = 3

    def add(self, m):
        print('self is {0} @B.add'.format(self))
        super(B, self).add(m)
        self.n += 3

class D(B, C):
    def __init__(self):
        self.n = 5

    def add(self, m):
        print('self is {0} @D.add'.format(self))
        super(D, self).add(m)
        self.n += 5

d = D()
d.add(2)
print(d.n)
```

输出

```
class A(object):
    def __init__(self):
        self.n = 2

    def add(self, m):
        print('self is {0} @A.add'.format(self))
        self.n += m

class C(A):
    def __init__(self):
        self.n = 4

    def add(self, m):
        print('self is {0} @C.add'.format(self))
        super(C, self).add(m)
        self.n += 4

class B(A):
    def __init__(self):
        self.n = 3

    def add(self, m):
        print('self is {0} @B.add'.format(self))
        super(B, self).add(m)
        self.n += 3

class D(B, C):
    def __init__(self):
        self.n = 5

    def add(self, m):
        print('self is {0} @D.add'.format(self))
        super(D, self).add(m)
        self.n += 5

d = D()
d.add(2)
print(d.n)
```

输出:

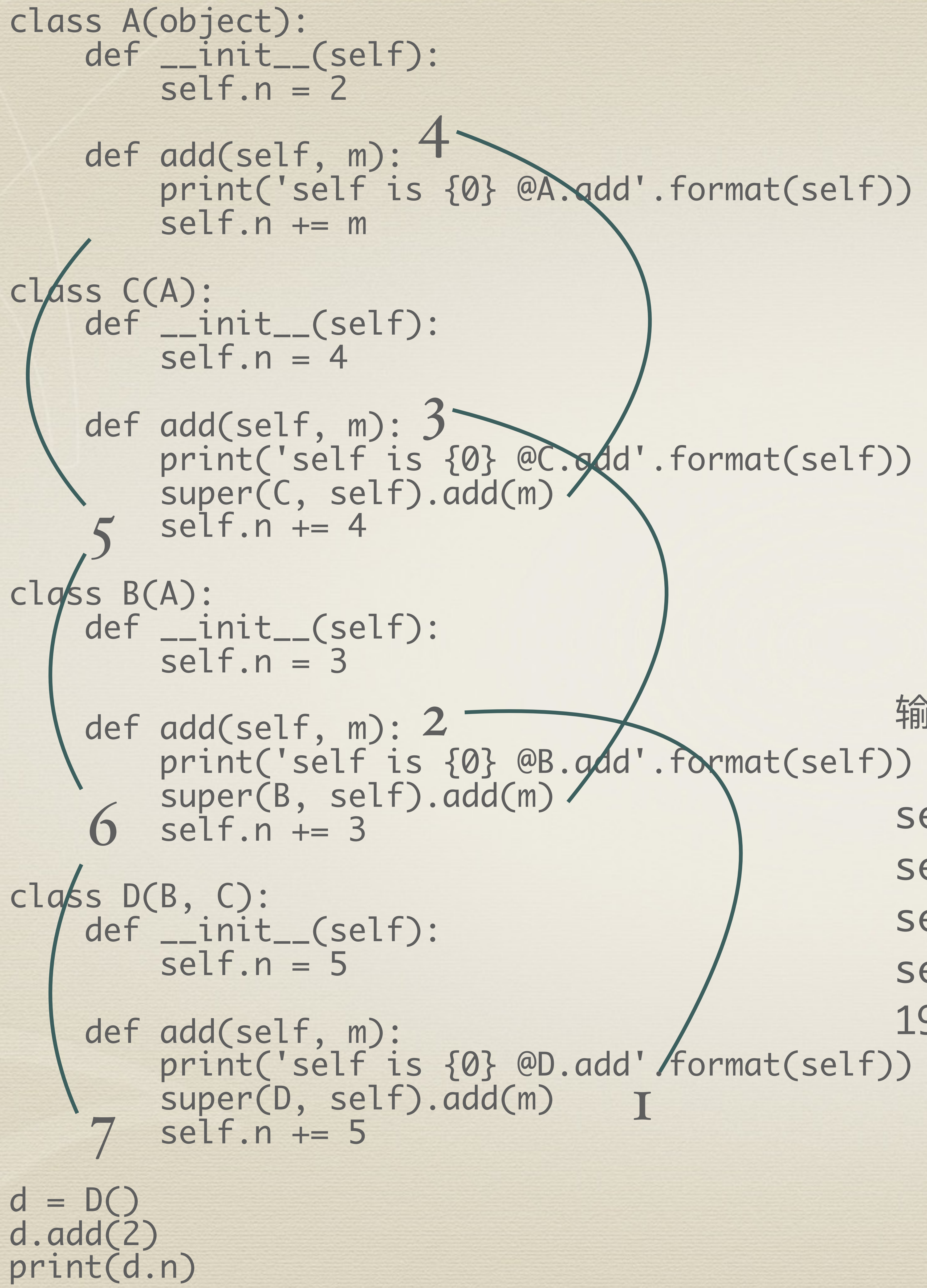
```
self is <__main__.D object at 0x1019f5790> @D.add
self is <__main__.D object at 0x1019f5790> @B.add
self is <__main__.D object at 0x1019f5790> @C.add
self is <__main__.D object at 0x1019f5790> @A.add
19
```

super 的工作方式

```
super(type2, type1)
```

super 的工作方式

* super(type2, type1)
* issubclass(type1, type2)
* type1.__mro__ == [type1, type2, type3, …]
* type2 in type1.__mro__
* super(type2, type1).foo() 从 type1 MOR 中 type2 后的 [type3, …] 中查找 foo 方法



```python
class A(object):
    def __init__(self):
        self.n = 2

    def add(self, m):
        print('self is {0} @A.add'.format(self))
        self.n += m

class C(A):
    def __init__(self):
        self.n = 4

    def add(self, m):
        print('self is {0} @C.add'.format(self))
        super(C, self).add(m)
        self.n += 4

class B(A):
    def __init__(self):
        self.n = 3

    def add(self, m):
        print('self is {0} @B.add'.format(self))
        super(B, self).add(m)
        self.n += 3

class D(B, C):
    def __init__(self):
        self.n = 5

    def add(self, m):
        print('self is {0} @D.add'.format(self))
        super(D, self).add(m)
        self.n += 5

d = D()
d.add(2)
print(d.n)
```

输出:

```
self is <__main__.D object at 0x1019f5790> @D.add
self is <__main__.D object at 0x1019f5790> @B.add
self is <__main__.D object at 0x1019f5790> @C.add
self is <__main__.D object at 0x1019f5790> @A.add
19
```

Thank You!