多继承以及MRO顺序

笔记本： Python提高-2
创建时间： 2018/5/8 9:18 **更新时间：** 2018/5/8 21:34
作者： ly

函数复习

多值参数

```
def fun1(self, *args, **kwargs)
```

不定长参数

函数参数： **可变参数*agrs** 字典类型的可变参数**kwargs

不定长参数的拆包

```
def fun1(*args, **kwargs):
    print(args, ">>>", kwargs)
def fun2(*args, **kwargs):
    fun1(args, kwargs)
    print("哈哈哈",args, ">>>", kwargs)
    # 两个都有传参数, 但到fun1时, 都传给了*args
    fun1(*args, **kwargs)
    print("哈哈哈", args, ">>>", kwargs, )
    # 两个都有传参数, 到fun1时, 不会有参数变化
fun2(1,3,4,a = 5, b = "hfh", c = 8)
"""
((1, 3, 4), {'a': 5, 'b': 'hfh', 'c': 8}) >>> {}
哈哈哈 (1, 3, 4) >>> {'a': 5, 'b': 'hfh', 'c': 8}
(1, 3, 4) >>> {'a': 5, 'b': 'hfh', 'c': 8}
哈哈哈 (1, 3, 4) >>> {'a': 5, 'b': 'hfh', 'c': 8}
"""
```

多继承

子类中有多个父类

MRO顺序

调用被重新父类的方法：

- 1 直接使用

```
父类.方法名(当前对象)      # 需要手动传递对象  当有self 必须要写self
```

- 2　通过super使用 按照继承链的顺序去调用

继承链：

查看当前继承链

```
print(类名.__mro__)
```

super会在继承链中去查询指定类的下一个类

```
super(类名, 对象).方法名()
```

第一种方法在使用时，重复调用了父类，浪费了资源。第二种方法，父类只会在继承链中出现一次，节约资源
注意：当前类实例化的对象，super时只能按照当前类所在继承链顺序使用，且不会重复。

第一种方法和第二种方法在单继承中基本无差别
但在多继承上有区别，super方法能保证每个父类的方法只会执行一次，而使用类名的方法会导致方法被执行多次。

```
class Parent(object):      # 父类
    def __init__(self, name):
        self.name = name
        print('parent的init结束被调用')

class Son1(Parent):       # 儿子类     继承了父类
    def __init__(self, name, age, *args, **kwargs):
        self.age = age
        super().__init__(name,*args, **kwargs)
        print('Son1的init结束被调用')

class Son2(Parent):     # 儿子类     继承了父类
    def __init__(self, name, gender,*args, **kwargs):
        self.gender = gender
        super().__init__(name,*args, **kwargs)
        print('Son2的init结束被调用')

class Grandson(Son1, Son2):     # 孙子类    同时继承了两个儿子类
    def __init__(self, name, age, gender):
        super().__init__(name, age, gender)
        print('Grandson的init结束被调用')


# print(Grandson.__mro__)
    # 儿子类的继承链 (<class '__main__.Grandson'>, <class '__main__.Son1'>, <class '__main__.Son2'>, <class
'__main__.Parent'>, <class 'object'>)
# print(Grandson.mro())
      #儿子类的继承链 [<class '__main__.Grandson'>, <class '__main__.Son1'>, <class '__main__.Son2'>, <class
'__main__.Parent'>, <class 'object'>]

g = Grandson("123", 18, "nan")

输出结果：
parent的init结束被调用
Son2的init结束被调用
Son1的init结束被调用
```

```
Grandson的init结束被调用
```

面试题：

```
class A(object):
    def __init__(self):
        print('A.__init__')


class B(A):
    def __init__(self):
        super().__init__()
        # A.__init__(self)     # 调用父类A类里面的__init__属性
        print('B.__init__')


class C(A):
    def __init__(self):
        # super().__init__()
        A.__init__(self)       # 调用父类A类里面的__init__属性
        print('C.__init__')


# [<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
class D(B, C):
    def __init__(self):
        super().__init__()
        # super(B,self).__init__()    # super(类名,参数)            作用：告诉super以B为基础继续B的下一个继承   相当于跳过B  直接到下一个类C
        print('D.__init__')


# print(D.mro())  # [<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
# print(D.__mro__)  # (<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>)

d = D()
```