面向对象-继承

- 目标
  - 继承的概念
  - 单继承
  - 多继承
  - 子类重写父类的同名属性和方法
  - 子类调用父类的同名属性和方法
  - 多层继承
  - super()

一. 继承的概念

生活中的继承，一般指的是子女继承父辈的财产。

- 拓展1：经典类或旧式类

不由任意内置类型派生出的类，称之为经典类。

```
class 类名:
    代码
    ......
```

- 拓展2：新式类

```
class 类名(object):
  代码
```

Python面向对象的继承指的是多个类之间的所属关系，即子类默认继承父类的所有属性和方法，具体如下：

```
# 父类A
class A(object):
    def __init__(self):
        self.num = 1
```

```
    def info_print(self):
        print(self.num)

# 子类B
class B(A):
    pass


result = B()
result.info_print()  # 1
```

> 在Python中，所有类默认继承object类，object类是顶级类或基类；其他子类叫做派生类。

二. 单继承

> 故事主线：一个煎饼果子老师傅，在煎饼果子界摸爬滚打多年，研发了一套精湛的摊煎饼果子的技术。师父要把这套技术传授给他的唯一的最得意的徒弟。

分析：徒弟是不是要继承师父的所有技术？

```
# 1. 师父类，属性和方法
class Master(object):
    def __init__(self):
        self.secret = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 2. 定义徒弟类，继承师父类
class Prentice(Master):
    pass


# 3. 用徒弟类创建对象，调用实例属性和方法
wanzi = Prentice()
print(wanzi.secret)
wanzi.make_cake()
```

三. 多继承

> 故事推进：**丸子**是个爱学习的好孩子，想学习更多的煎饼果子技术，于是，在百度搜索到**青灯教育**，报班学习煎饼果子技术。

所谓多继承意思就是一个类同时继承了多个父类。

```
# 1. 师父类，属性和方法
class Master(object):
    def __init__(self):
```

```
        self.secret = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 为了验证多继承，添加School父类
class School(object):
    def __init__(self):
        self.secret = '[青灯煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 2. 定义徒弟类，继承师父类 和 学校类
class Prentice(Master, School):  # 如果一个类继承多个父类，优先继承第一个父类的同名属性和方法
    pass

# 3. 用徒弟类创建对象，调用实例属性和方法
wanzi = Prentice()
print(wanzi.secret)
wanzi.make_cake()
```

> 注意：当一个类有多个父类的时候，默认使用第一个父类的同名属性和方法。

四. 子类重写父类同名方法和属性

> 故事：**丸子**掌握了师父和培训的技术后，自己潜心钻研出自己的独门配方的一套全新的煎饼果子技术。

```
# 1. 师父类，属性和方法
class Master(object):
    def __init__(self):
        self.secret = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')

class School(object):
    def __init__(self):
        self.secret = '[青灯煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 2. 定义徒弟类，继承师父类 和 学校类， 添加和父类同名的属性和方法
class Prentice(School, Master):
    def __init__(self):
        self.secret = '[独创煎饼果子技术]'
```

```
    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 3. 用徒弟类创建对象，调用实例属性和方法
wanzi = Prentice()
print(wanzi.secret)
wanzi.make_cake()
```

> 子类和父类具有同名属性和方法，默认使用子类的同名属性和方法。

五. 子类调用父类的同名方法和属性

> 故事：很多顾客都希望也能吃到古法和青灯的技术的煎饼果子。

```
# 1. 师父类，属性和方法
class Master(object):
    def __init__(self):
        self.secret = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 为了验证多继承，添加School父类
class School(object):
    def __init__(self):
        self.secret = '[青灯煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.secret}制作煎饼果子')


# 2. 定义徒弟类，继承师父类 和 学校类， 添加和父类同名的属性和方法
class Prentice(School, Master):
    def __init__(self):
        self.secret = '[独创煎饼果子技术]'

    def make_cake(self):
        self.__init__()  # 子类的 __init__
        print(f'运用{self.secret}制作煎饼果子')

    # 子类调用父类的同名方法和属性：把父类的同名属性和方法再次封装
    def make_master_cake(self):
        Master.__init__(self)  # 父类的 __init__
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)  # 父类的 __init__
        School.make_cake(self)
```

```
# 3. 用徒弟类创建对象，调用实例属性和方法
daqiu = Prentice()
daqiu.make_cake()
daqiu.make_master_cake()
daqiu.make_school_cake()
```

六. 多层继承

> 故事：N年后，**丸子**老了，想要把所有技术传承给自己的徒弟。

```
# 1. 师父类，属性和方法
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')


# 为了验证多继承，添加School父类
class School(object):
    def __init__(self):
        self.kongfu = '[青灯煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')


# 2. 定义徒弟类，继承师父类 和 学校类， 添加和父类同名的属性和方法
class Prentice(School, Master):
    def __init__(self):
        self.kongfu = '[独创煎饼果子技术]'

    def make_cake(self):
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    # 子类调用父类的同名方法和属性：把父类的同名属性和方法再次封装
    def make_master_cake(self):
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)


# 步骤：1. 创建类Tusun， 用这个类创建对象；2. 用这个对象调用父类的属性或方法看能否成功
class Tusun(Prentice):
    pass
```

```
zx = Tusun()
zx.make_cake()
zx.make_master_cake()
zx.make_school_cake()
```

七. super()调用父类方法

```
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(Master):
    def __init__(self):
        self.kongfu = '[青灯煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

        # 方法2.1
        # super(School, self).__init__()
        # super(School, self).make_cake()

        # 方法2.2
        super().__init__()
        super().make_cake()


class Prentice(School):
    def __init__(self):
        self.kongfu = '[独创煎饼果子技术]'

    def make_cake(self):
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    # 子类调用父类的同名方法和属性：把父类的同名属性和方法再次封装
    def make_master_cake(self):
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)

    # 一次性调用父类的同名属性和方法
    def make_old_cake(self):
        # 方法一：代码冗余；父类类名如果变化，这里代码需要频繁修改
        # Master.__init__(self)
        # Master.make_cake(self)
```

```
        # School.__init__(self)
        # School.make_cake(self)

        # 方法二: super()
        # 方法2.1 super(当前类名, self).函数()
        # super(Prentice, self).__init__()
        # super(Prentice, self).make_cake()

        # 方法2.2 super().函数()
        super().__init__()
        super().make_cake()


wanzi = Prentice()
wanzi.make_old_cake()
```

> 注意：使用super() 可以自动查找父类。调用顺序遵循 `__mro__` 类属性的顺序。比较适合单继承使用。

九. 总结

- 继承的特点
    - 子类默认拥有父类的所有属性和方法
    - 子类重写父类同名方法和属性
    - 子类调用父类同名方法和属性
- super()方法快速调用父类方法