一 多层继承

小徐成了老徐之后， 有女儿了，小地瓜想要学习父亲的所有功夫

老徐(一剑仙人跪， 霸王卸甲， 剑气滚龙壁)

```
# 定义师父类
class Master(object):
    def __init__(self):
        self.kongfu = '一剑仙人跪'
        self.name = '李淳罡'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')

# 定义侍女类
class Maid(object):
    def __init__(self):
        self.kongfu = '霸王卸甲'
        self.name = '青鸟'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')


# 定义徒弟类
class Prentice(Master,Maid):
    def __init__(self):
        self.kongfu = '剑气滚龙壁'

    def battle(self):
        self.__init__()
        print(f'运用了{self.kongfu}和敌人battle')


    # 把父类的同名方法和属性再次封装就可以了
    def master_battle(self):
        # 父类名.方法名()
        Master.__init__(self)
        Master.battle(self)
        # Master().battle()  # 不推荐使用  相当于重新创建了一个新的父类对象，占用了不必要的内存

class Daughter(Prentice):
    pass


# 用徒弟类创建对象，调用师父的属性和方法
xiaodigua = Daughter()
print(xiaodigua.kongfu)
xiaodigua.battle()
xiaodigua.master_battle()
```

二 super()

```
# 定义师父类
class Master(object):
    def __init__(self):
        self.kongfu = '一剑仙人跪'
        self.name = '李淳罡'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')

# 定义侍女类
class Maid(Master):
    def __init__(self):
        self.kongfu = '霸王卸甲'
        self.name = '青鸟'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')


# 定义徒弟类
class Prentice(Maid):
    def __init__(self):
        self.kongfu = '剑气滚龙壁'

    def battle(self):
        self.__init__()
        print(f'运用了{self.kongfu}和敌人battle')


    # 把父类的同名方法和属性再次封装就可以了
    def master_battle(self):
        # 父类名.方法名()
        Master.__init__(self)
        Master.battle(self)
        # Master().battle()  # 不推荐使用  相当于重新创建了一个新的父类对象，占用了不必要的内存

    def old_battle(self):
        # 方案一： super().方法()
        super().__init__()   # super 相当于maid
        super().battle()

xiaoxu = Prentice()
# xiaoxu.master_battle()
xiaoxu.old_battle()
print(Prentice.__mro__)
```

```
# 定义师父类
class Master(object):
    def __init__(self):
        self.kongfu = '一剑仙人跪'
        self.name = '李淳罡'
```

```
    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')

# 定义侍女类
class Maid(Master):
    def __init__(self):
        self.kongfu = '霸王卸甲'
        self.name = '青鸟'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')


# 定义徒弟类
class Prentice(Maid):
    def __init__(self):
        self.kongfu = '剑气滚龙壁'

    def battle(self):
        self.__init__()
        print(f'运用了{self.kongfu}和敌人battle')


    # 把父类的同名方法和属性再次封装就可以了
    def master_battle(self):
        # 父类名.方法名()
        Master.__init__(self)
        Master.battle(self)
        # Master().battle()  # 不推荐使用  相当于重新创建了一个新的父类对象，占用了不必要的内存

    def old_battle(self):
        # 方案一： super().方法()
        # super().__init__()   # super 相当于maid
        # super().battle()
        # 方案二： super(类名, self).方法()
        super(Maid, self).__init__()
        super(Maid, self).battle()

xiaoxu = Prentice()
# xiaoxu.master_battle()
xiaoxu.old_battle()
print(Prentice.__mro__)
```

总结:

使用super()方法可以自动查找父类, 调用的顺序遵循mro类属性的顺序，比较适合单继承使用

注意: 如果继承了多个父类，且父类有同名方法和属性，则默认只执行第一个父类的同名方法和属性(同名方法只执行一次，目前super不支持多个父类的同名方法)

```
# 定义师父类
class Master(object):
    def __init__(self):
```

```
        self.kongfu = '一剑仙人跪'
        self.name = '李淳罡'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')

# 定义侍女类
class Maid(object):
    def __init__(self):
        self.kongfu = '霸王卸甲'
        self.name = '青鸟'

    def battle(self):
        print(f'运用了{self.kongfu}和敌人battle')


# 定义徒弟类
class Prentice(Master,Maid):
    def __init__(self):
        self.kongfu = '剑气滚龙壁'

    def battle(self):
        self.__init__()
        print(f'运用了{self.kongfu}和敌人battle')


    # 把父类的同名方法和属性再次封装就可以了
    def master_battle(self):
        # 父类名.方法名()
        Master.__init__(self)
        Master.battle(self)
        # Master().battle()  # 不推荐使用   相当于重新创建了一个新的父类对象，占用了不必要的内存

    def old_battle(self):
        # 方案一： super().方法()
        super().__init__()   # super 相当于maid
        super().battle()
        # 方案二： super(类名, self).方法()
        # super(Maid, self).__init__()
        # super(Maid, self).battle()

xiaoxu = Prentice()
# xiaoxu.master_battle()
xiaoxu.old_battle()
print(Prentice.__mro__)
```

三 私有属性

3.1 定义私有属性和方法

在python中，可以为实例对象和属性设置私有权限，即设置某个实例属性和实例方法不继承给子类

设置私有属性和私有方法的语法是: 在属性和方法前面加两个下划线

```python
# 定义师父类
class Master(object):
    def __init__(self):
        self.kongfu = '一剑仙人跪'
        self.__name = '李淳罡'

    def __battle(self):
        print(f'运用了{self.kongfu}和敌人battle')

    def print_info(self):
        print(self.__name)


# 定义徒弟类
class Prentice(Master):
    pass


xiaoxu = Prentice()
xiaoxu.print_info()


li = Master()
li.print_info()
```

总结: 对象不能访问私有属性和私有方法，子类无法继承父类的私有属性和私有方法

3.2 获取和修改私有属性

在python中，一般定义方法名get_xxx 用来表示获取私有属性，定义set_xxx 用来表示修改私有属性值，这个是约定俗成的命名方法，不是强制要求

```python
# 定义师父类
class Master(object):
    def __init__(self):
        self.kongfu = '一剑仙人跪'
        self.__name = '李淳罡'

    def __battle(self):
        print(f'运用了{self.kongfu}和敌人battle')

    def get_name(self):  # 获取私有属性
        return self.__name

    def set_name(self):  # 修改私有属性
        self.__name = '老徐'


# 定义徒弟类
class Prentice(Master):
    pass


xiaoxu = Prentice()
print(xiaoxu.get_name())
```

```
xiaoxu.set_name()
print(xiaoxu.get_name())
```

总结：

继承的特点：子类拥有父类的所有属性和方法，除了私有属性和私有方法

面向对象_其他

目标

- 面向对象的三大特性
- 类属性和实例属性
- 类方法和静态方法

四 面向对象的三大特性

- 封装
  - 将属性和方法书写到类里面的操作就是封装
  - 封装可以为我们的属性和方法添加私有权限
- 继承
  - 子类默认继承父类的所有属性和方法
  - 子类可以重写父类的同名方法和属性
- 多态
  - 传入不同的对象，产生不同的效果

五 多态

5.1 了解多态

多态指的是一列事物有多种形态(一个抽象类有多个子类，因而多态的概念依赖于继承)

- 定义：多态是一种使用对象的方式，子类重写父类方法，调用不同的子类对象的同一父类方法时，产生不同的对象
- 好处：调用灵活，有了多态，更加容易编写出来通用的代码，做出通用的编程，以适应不同的需求
- 实现步骤：
  - 定义父类，并且提供公共方法
  - 定义子类，并重写父类方法
  - 传递子类对象给调用者，可以看到不同的子类的执行结果

5.2 体验多态

```
# 需求： 狗有很多种， 有抓坏人的  有找毒品的  警察带不同的狗做不通的事
# 定义父类
class Dog(object):
    def work(self):
        pass
```

```python
class Person(object):
    def work_with_dog(self, dog):
        dog.work()

# 定义子类
class ArmyDog(Dog):
    def work(self):
        print('抓坏人')

class DrugDog(Dog):
    def work(self):
        print('找毒品')

class Qwe(Dog):
    def work(self):
        print('导盲犬')

# 创建不同的对象 实现不同的功能
ad = ArmyDog()
dd = DrugDog()
ac = Qwe()

momo = Person()
momo.work_with_dog(ad)
momo.work_with_dog(dd)
momo.work_with_dog(ac)
```

六 类属性和实例属性

6.1 类属性

6.1.1 设置和访问类属性

- 类属性就是类对象所拥有的属性，他被该类的所有对象所共有
- 类属性可以使用类对象或者实例对象去访问

```python
# 老婆类
class Wife(object):
    sex = '女'  # 类属性

    def __init__(self, name, age):
        self.name = name
        self.age = age


    def print_info(self):
        print(self.name)
        print(self.age)


xiaobai = Wife('小白',18)


zuoshou = Wife('右手',18)
```

```
# 实例对象访问类属性
print(xiaobai.sex)
print(zuoshou.sex)
```

```
# 老婆类
class Wife(object):
    sex = '女'  # 类属性

    def __init__(self, name, age):
        self.name = name
        self.age = age


    def print_info(self):
        print(self.name)
        print(self.age)


xiaobai = Wife('小白',18)


zuoshou = Wife('右手',18)
# 实例对象访问类属性
# print(xiaobai.sex)
# print(zuoshou.sex)

# 类对象访问类属性
print(Wife.sex)

# 类属性用的都是一块内存空间
# print(id(xiaobai.sex))
# print(id(zuoshou.sex))

# 实例属性
print(id(xiaobai.name))
print(id(zuoshou.name))
```

总结:

- 记录的某项数据始终保持一致的时候，则可以定义类属性
- 实例属性要求每一个对象为其开辟一份独立的内存空间记录属性值，而类属性为全局所共有的，仅占用一份内存空间，更加的节省资源

6.1.2 修改类属性

```
# 老婆类
class Wife(object):
    sex = '女'  # 类属性

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
    def print_info(self):
        print(self.name)
        print(self.age)


xiaobai = Wife('小白',18)
zuoshou = Wife('右手',18)

#  通过类对象去修改类属性
# print(Wife.sex)
# Wife.sex = '中性'
# print(Wife.sex)
# print(xiaobai.sex)
# print(zuoshou.sex)


# 通过实例对象修改
xiaobai.sex = '中性'
print(Wife.sex)
print(xiaobai.sex)
print(zuoshou.sex)
```

类属性只能通过类对象来修改，不能通过实例对象来修改， 如果这样操作了，只是重新为此实例对象添加了一个实例属性而已

6.2 实例属性

```
class Dog(object):
    def __init__(self):
        self.age = 2


wangcai = Dog()
print(wangcai.age)
```

```
class Dog(object):
    def __init__(self):
        self.age = 2


wangcai = Dog()
print(wangcai.age)
print(Dog.age)  # 报错：实例属性只能通过实例对象来访问  不能通过类对象去访问
```