面向对象的特性

三大特性---三大基石
封装--继承--多态

封装

类成员=属性+方法
1. 对象都有明确的边界，把属性保护在边界之内
    房子：家具，人宠物，植物
    对象：属性+方法
2. 封装的粒度
    控制一个对象的范围
    粒度不宜过大，也不宜过小
    粒度过大：对象过于复杂，过程过于简单
    粒度过小：对象过于简单，过程复杂
3. 封装原则
    1. 将不需要对外提供的内容都隐藏起来
    2. 把属性隐藏，提供公共方法进行访问
4. 好处
    1. 提高安全性
    2. 提高复用性
    3. 便于使用

继承

狗类--猫类--鱼类-----动物类
1. 父类---子类----继承
2. 定义形式
    class 子类类名(父类类名)：
        属性
        方法
3. 父类
    基类，超类
    object---顶层类
    任何类没有继承关系，则默认继承与object

- 继承的特点

1. 子类可以继承父类的方法和属性
2. 可扩展性
3. 父类中私有成员不能被继承
        间接访问：实例._父类类名私有属性/方法
4. 如果子类没有书写构造方法，会调用父类的
5. 如果子类有自己的构造方法，不调用父类的
6. 子类可以访问父类的构造方法
        父类类名.__init__(self)
        super().__init__()   不需要加

- 多继承

```
class A:
    age=66
    def __init__(self):
        print(1111111111111)

class B:
    name='haha'
    def __init__(self):
        print(2222222222222)

class C(A,B):
    pass

c=C()
print(c.age,c.name)
```

1. 书写形式
        class 子类类名（父类类名1，父类类名2...）
2. 优势
        增强了可扩展性
3. 弊端
        1. 避免多继承，造成继承混乱
        2. 大量占用资源---多弊端的陷阱：钻石继承问题

- 方法覆盖

1. 子类用特殊的方法实现，替换了父类继承下来的同名的方法实现
        子类重新覆盖了父类的方法
2. 子类覆盖掉父类的方法，本质并没有替换父类方法，父类的方法依然存在
3. 语法要求
        方法名必须和父类的方法名完全相同

- 钻石继承问题

1. 可以利用super()进行解决

2. 如果一个类继承了多个类，多个类继承了同一个类，创建对象时造成资源浪费（最高级的父类会多个创建）
        super的内核---Mro---C3算法
        mro：继承链关系-----核心 merge（）
3. super的使用
        super([type],[obj]):
            type:当前类
            obj：self 当前对象

MRO原理：

```
    class A:
        age=66
        def __init__(self):
            print('A')

    class B(A):
        name='haha'
        def __init__(self):
            print('B')
            super().__init__()
    class C(A):
        def __init__(self):
            print('C')
            super().__init__()

    class D(B,C):
        def __init__(self):
            print('D')
            super().__init__()

    d=D()
```

继承链关系推算：

```
    mro(D)=[D]+merge(mro(B),mro(C),[B,C])=[D]+merge([B,A],[C,A],[B,C])
    [D,B] ====[A] [C,A]  [C]

    [D,B,C] ===[A]  [A]
    [D,B,C,A]
```

多态和多态性

- 多态

一类事物有多种形态

```
    序列：列表，元组，字符串
    动物：猫，狗，鱼
    多态和继承有关
    class Hero:
        def __init__(self):
            pass
        def stroke(self):
            pass

    class yasuo(Hero):
        def stroke(self):
```

```
            print('哈萨尅')


    class xiazi(Hero):
        def stroke(self):
            print('一库')
```

- 多态性

向不同的对象发送同一条消息，不同的对象在接受信息时，会做出不同的反应
多态性：定义一个统一的接口

```
        class Bird:
            def fly(self):
                print('小鸟会飞')

        class Plane:
            def fly(self):
                print('飞机会飞')

        class Rocket:
            def fly(self):
                print('火箭会飞')
        def fun(obj):
            obj.fly()

        fun(Bird())
        fun(Plane())
        fun(Rocket())
```

好处：

1. 增加了程序的灵活性
2. 增加了程序扩展性

内置对象

```
1. issubclass(class1,class2_or_(classtuple))
        判断class1是否是class2或者classTuple子类
        >>> issubclass(B,D)
            False
        >>> issubclass(C,(A,D))
            True
        >>> issubclass(C,(D,))
2. isinstance(obj,class_or_classTuple)
        判断obj是否是class,classTuple的实例对象
        >>> isinstance('hehe',int)
        >>> isinstance(d,B)
3. hasattr(obj,name)
        判断实例或者是实例的父类是否有name属性
        >>> hasattr(c,'age')  #是否含有age属性
```

```
4. getattr(obj,name,[d])
        获取实例对象或它父类的name属性值，当属性不存在时，会报错（没有书写提示信息的情况下）
        name:属性
        d：提示信息
        >>> getattr(d,'age')
            44
        >>> getattr(d,'age','没有这个属性')

5.  setattr(obj,name,value)
        在实例对象所对应的的类中添加一个属性name，并且赋值value
        >>> setattr(d,'name','hehe')
6. delattr(obj,name)
        在obj中删除name属性
        >>> delattr(d,'name')
```

- property

```
1. 将属性和另一个属性进行关联，修改一个属性，则与之关联的属性也跟着修改
2. property（fget,fset,fdel）
        fget:get方法
        fset:set方法
        sdel:delete方法
```