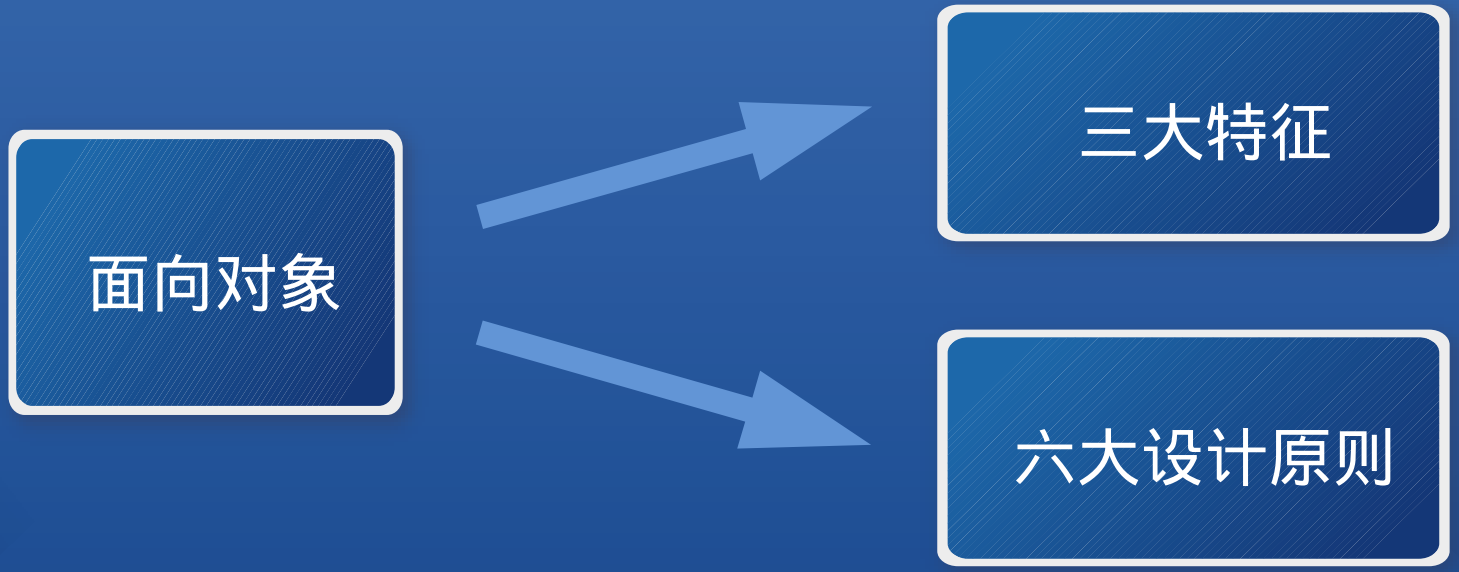
面向对象 Object Oriented



2019/07/20

陈鹏 (Cheney)

面向过程与面向对象

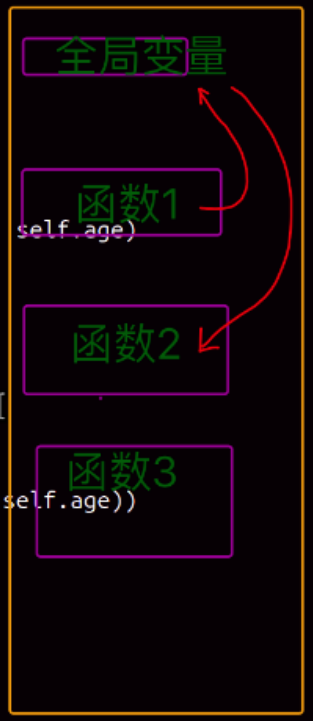
- 面向过程的本质就像是一个开放的箱子，所有的变量和方法都是暴露在外面的，所有的方法共享全局变量，这个全局变量可以被所有的方法修改，这就带来一个问题：如果一个方法 f1 想要全局变量只被它所用，但是其他方法肯定也要用的，这样就会造成变量修改的冲突；
- 这时候类的概念应运而生，所有的变量和方法封装在类 (黑箱子) 内部，类属性相当于类内部的全局变量，实例属性相当于方法内部的局部变量，这样，只需要通过类创建不同实例 a,b,c,d ，实例 a,b,c,d 就可以将这些属性，方法全部私有化，随便实例自己怎么折腾。
-

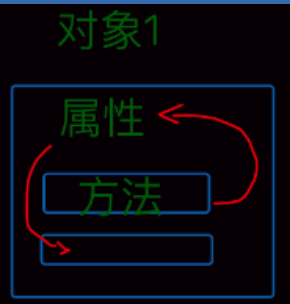
封装

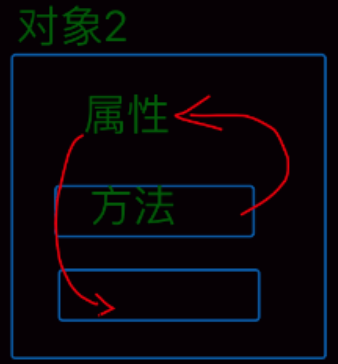
封装的本质是将事物相关的属性和方法封装在一个类里面，我们调用类创建实例的时候，不用关心类内部的代码细节，相当于一个黑箱子，我们只需要该实例（黑箱）能给出我们想要的结果就好了。
而在用类进行封装的过程中要遵循单一职责的原则，单一职责指的是一个类能被修改的原因只有一个，就是类要有专一性 .

示例

```python
class Cat:
    """定义了一个Cat类"""

    #初始化对象
    def __init__(self, new_name, new_age):
        self.name = new_name
        self.age = new_age

    def __str__(self):
        return "%s的年龄是:%d"%(self.name, self.age)

    #方法
    def eat(self):
        print("猫在吃鱼....")

    def drink(self):
        print("猫正在喝kele.....")

    def introduce(self):
        print("%s的年龄是:%d"%(self.name, self.age))

#创建一个对象
tom = Cat("汤姆", 40)

lanmao = Cat("蓝猫", 10)


print(tom)
print(lanmao)
```







总结

- 封装：
- 1. 是将属性 (全局变量) ，方法 (函数) 都封装在一个黑盒子里面；
- 2. 类里面的方法可以共享属性，属性的修改不会影响类的外部变量，这就是类的封装产生的优势；
- 3. 同时类可以被继承，子类可以拥有和父类一样的属性和方法；
- 4. 并且子类可以有新的属性，新的方法，
-

继承

继承 : 就是派生类拥有基类 (超类) 的所有属性和方法 (子类只需要封装自己特有的属性和方法即可) 。

用一句话来概括 : 派生类拥有基类 (超类) 的所有成员

示例

```python
from math import pi


class Circle:
    def __init__(self, r):
        self.r = r

    def circle_area(self):
        return pi * self.r ** 2

    def circle_perimeter(self):
        return 2 * pi * self.r


class Ring(Circle):
    def __init__(self, r, r1):
        self.r1 = r1

        super().__init__(r)

    def circle_area(self):
        return super().circle_area() - pi * self.r1 ** 2

    def circle_perimeter(self):
        return super().circle_perimeter() + 2 * pi * self.r1


r01 = Ring(5, 3)
print(r01.circle_area())
print(r01.circle_perimeter())

```

总结

- 优点： 一种代码复用的方式 .
- 缺点：耦合度高：基类的变化，直接影响派生类 .
- 派生类可以继承基类的属性和行为，通过继承的方式可以减少代码的冗余度
- 派生类在继承的时候，在定义类时，小括号（）中为基类的名字
- 虽然派生类没有定义 __init__() 方法，但是继承了基类，只要创建对象，就会默认执行基类的 __init__() 方法

多态

多态性依赖于继承。从一个基类派生出多个派生类，可以使派生类之间有不同的行为，这种行为称之为多态。
简言之，就是派生类重写基类的方法，使派生类具有不同的方法实现

多态

派生类重写基类的方法

加上派生类自身的逻辑

示例

```
163 class Animal:
164     def __init__(self, name=""):
165         self.name = name
166
167     def talk(self):
168         print(self.name, "叫")
169
170     @staticmethod
171     def animal_talk(obj):
172         obj.talk()
173
174
175 class Cat(Animal):
176
177     def talk(self):
178         print("%s:喵喵喵" % self.name)
179
180
181 class Dog(Animal):
182
183     def talk(self):  self: <__main__.Dog object at 0x7f918b85a240>
184         print("%s:汪汪汪" % self.name)
185
186
187 d01 = Dog("二哈")
188 c01 = Cat("HelloKitty")
189
190 a = Animal()
191 a.animal_talk(d01)  # 调用基类方法,执行派生类
192 a.animal_talk(c01)  # 调用基类方法,执行派生类
193
```

总结

- 定义：基类的同一种动作或者行为，在不同的派生类上有不同的实现。
- 派生类实现了基类中相同的方法（方法名、参数）。在调用该方法时，实际执行的是派生类的方法。
- 作用：
  - 1. 在继承的基础上，体现类型的个性化（一个行为有不同的实现）。
  - 2. 增强程序扩展性，体现开闭原则

面向对象设计六大原则

六大原则

- 开闭原则
- 里式替换原则
- 依赖倒置
- 迪米特法则
- 单一职责
- 组合复用

开闭原则与里式替换原则

六大设计 原则实际上都是互补的，也就是说一些原则需要利用另一些原则来实现自己。

里氏替换针对的是继承关系，只要是基类能参与的任何构造，派生类完全可以胜任，所以里氏替换的主要思想就是只要基类可以被调用，派生类就一定要代替基类被调用。这种情况并不是说父类没有意义，相反的，里氏替换进一步要求基类尽可能不要存在太具体的功能，能抽象就尽量抽象，任何的修改都完全依靠派生类来补充和修改，从而进一步实现开闭原则（基类对修改关闭，派生类对修改开放）

开闭原则

里式替换原则

单一职责原则

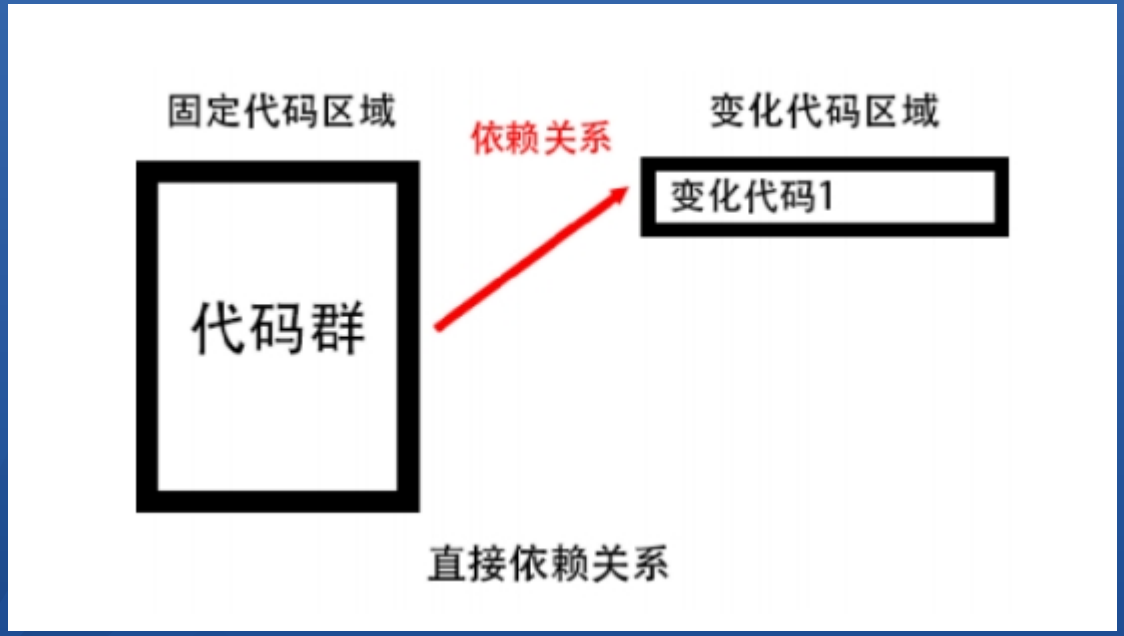
单一职责指的是一个类能被修改的原因只有一个，就是类要有专一性。如果一个类封装太多变化，就可能导致牵一发而动全身，这样对系统来说会有风险。

迪米特法则也被戏称为“最少知道法则”，其实说白了就是针对面向对象里的低耦合。它的本意指的是类与类之间尽可能不要有太多的关联，当一个类需要产生变化时，其他的类尽量做到不产生改动。具体的体现其实与上面阐述的单一原则是一致的，即每一个类最好能做到只会被同一件事情影响和改变，其他类尽可能不受其影响。

单一职责

迪米特法则

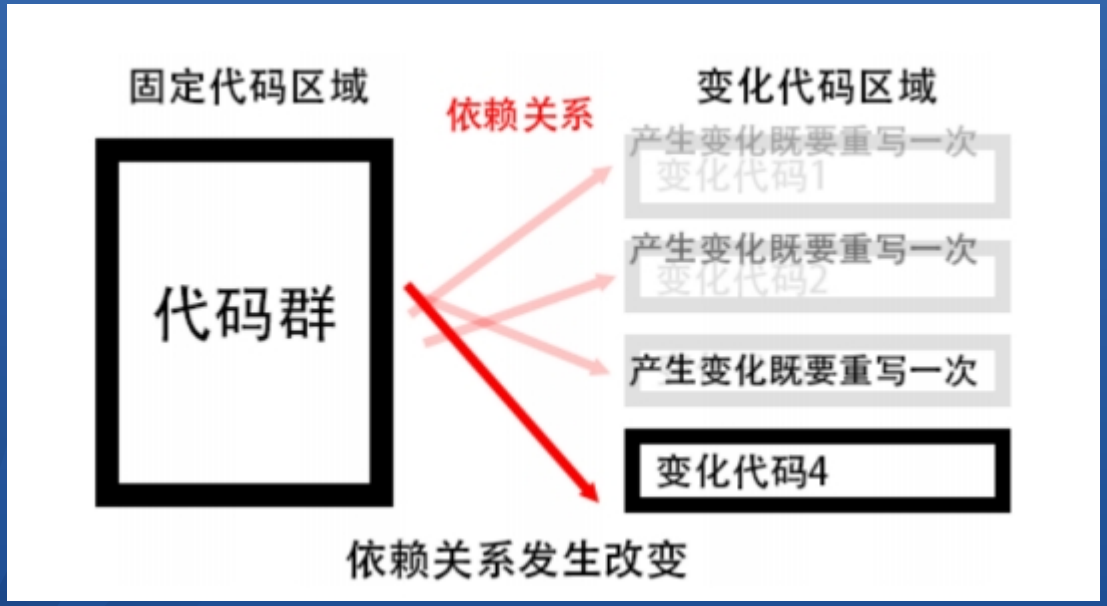
依赖倒置原则



依赖倒置原则

依赖关系，在面向对象思想过程中首先要做的是分，将需要改变的代码分离出去，将不需要改变的代码整合到一起，至此，我们实现了最基本的两个封装，之后，整合的代码往往需要调取会产生改变的代码，这种关系可以理解为最直接的依赖关系 .

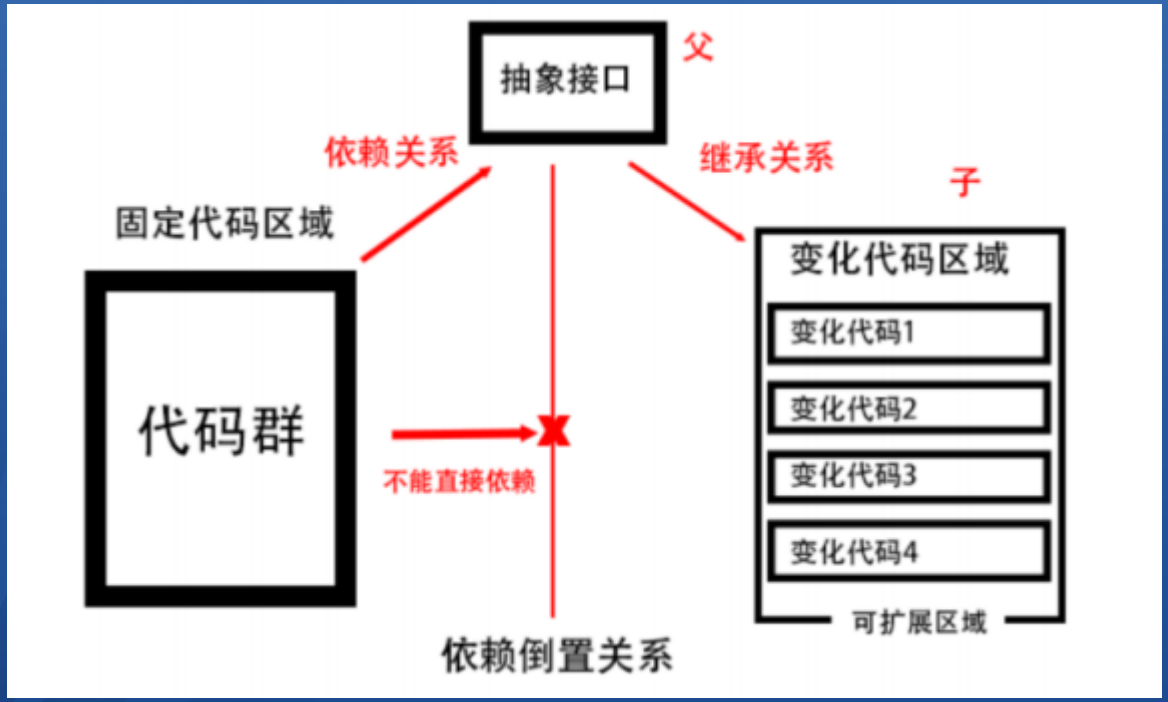
依赖倒置原则



依赖倒置原则

而当产生需求变化时，这种基础依赖关系就会随之产生变化，变化的越多，依赖关系修改的越多

依赖倒置原则



为了避免后期代码的变化导致工程量的增加，首先要做的是将会变化的代码变成一个大的框架，给它们一个统一的类型，也就是统一继承，给一个基类，基类本身只是作为一个中转站用来衔接固定代码区域和变化代码区域，换句话说，此时固定代码区域如果想要调用变化代码区域，必须要经过基类，通过把传统的依赖具体代码关系转变成依赖抽象代码方式，就称之为依赖倒置。

组合复用

为什么要用组合复用？

在面向对象的设计中，如果直接继承基类，会破坏封装，因为继承将基类的实现细节暴露给子类；如果基类的实现发生改变，则子类的实现也不得不发生改变；从基类继承而来的实现是静态的，不可能在运行时发生改变，没有足够的灵活性。于是就提出了组合 / 聚合复用原则，也就是在实际开发设计中，尽量使用合成 / 聚合，不要使用类继承。即在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分，新对象通过向这些对象的委派达到复用已有功能的目的。就是说要尽量的使用合成和聚合，而不是继承关系达到复用的目的。

组合复用

组合复用示例

```python
# 组合复用
from math import pi


class Circle:
    def __init__(self, r):
        self.r = r

    def circle_area(self):
        return pi * self.r ** 2

    def circle_perimeter(self):
        return 2 * pi * self.r


class Ring:
    def __init__(self, out_circle, in_circle):
        self.out_circle = Circle(out_circle)
        self.in_circle = Circle(in_circle)

    def ring_area(self):
        return self.out_circle.circle_area() - self.in_circle.circle_area()

    def ring__perimeter(self):
        return self.out_circle.circle_perimeter() + self.in_circle.circle_perimeter()


ring1 = Ring(5, 3)
print(ring1.ring__perimeter())
print(ring1.ring_area())
```

FINAL

以上

THANKS