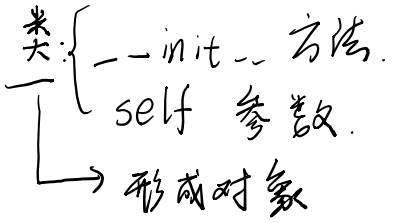
面相对象编程OOP

2026年1月15日 10:01

面向对象编程（OOP）是一种重要的编程范式，它将数据和操作数据的方法组织为类和对象，让代码更模块化、易维护和可重用。

简单说就是对于可以使用统一模版的编程问题，可以使用函数设置模版，然后生成函数的对应不同参数值的实例，从而实现简单、统一管理的编程方式，并且这种面向对象的编程方式可以重叠使用，即在某一函数中调用其他的对象函数，从而实现多重套用



① 封装——隐藏内部细节.
只提供可控的接口.

name：公有属性.
_name：受保护的
__name：私有属性.

② 继承.——代码重用.
方法重写：子类定义父类.
super() 调用

③ 多态——不同类的对象，用相同方法名调用时
但每个对象会执行自己类中定义的具体操作
只要方法名相同，我们就可以重复利用.

面相过程编程范式

2026年1月26日 19:32

最基本的编程范式，按照从上到下的顺序执行代码

其基本思路：是将一个复杂的大问题一步步拆解为小问题或子过程，这些子过程再执行的过程就是将小问题继续分解直到在有限的可数步骤内解决

例如：有某个网站需求将日志进行分析，可以分为以下过程：

1. 到各台服务器上收集日志
2. 对日志进行各维度的分析，比如Pv,uv,IP,来源地区，访问设备等
3. 生成报告，发送邮件至客户

大致代码可实现：

```
def collect_log():
def log_analyze(log_file):
def send_report(report_data):
def main():
    collect_log()
    log_analyze('log_filename')
    send_report('report_data_name')
```

其实现过程简单，但是也有非常大的实际问题：

当需要拓展代码功能、修改代码模块的时候，往往会导致：

1. 代码冗余，新增功能需要在原有功能基础上继续添加代码——进一步导致维护难度上升
2. 代码互相依赖，导致进一步维护困难，且随着代码复杂性增加而大大增加困难
3. 因而面向过程编程不适合复杂任务、大型任务

检测作业

2026年1月19日 11:52

作业&练习题

阅读量: 7

练习题

1. 类、对象、实例、实例化有什么关系？
2. 类属性和实例属性有什么区别?
3. 写个实际使用类属性的场景代码
4. 自己设计场景并写个封装的代码
5. 自己设计场景并写个继承的代码
6. 自己设计场景并写个多态的代码
7. 自己设计场景写个类组合的关系 代码，比如 cs游戏，人是一个类， 枪是一个类，人可以选择不同的枪。

1. 类就是对具有相同属性和方法的对象的抽象描述或蓝图；对象就是根据类创造出的一个具体的实体；在Python中实例和对象是一个概念，实例会更强调于“由某个类创建出来的”这个关系；实例化就是按照某一类的蓝图具体地创建出一个实例的过程；具体关系可以概括为：类通过__init__构造方法实现实例化过程，从而创建出对象/实例。
2. 类属性是属于该类和该子类的，之后的实例也会共享该数据，内存中只有一份，可以通过类名或实例名来访问，通常用于定义共享常量、计数器或默认配置；实例属性则是单独属于该实例或某个具体对象的，通过self方法定义，每个实例会有独立的数据副本和独立的存储空间，只能通过实例名进行访问
3. 写个实际使用类属性的场景代码：
   a.
```
class Dog
    species = "Canis familiaris" #类属性: 所有狗共享的物种属性
    total_dog_created = 0  #类属性：计数器
    def __init__(self, name, breed = "未知品种"):
        self.name = name  #实例属性
        self.breed = breed #实例属性
        Dog.total_dog_created += 1  # 计数器更新
        self.id = dog.total_dog_created #生成唯一ID
    def print_info(self):
        print(f"{self.name}的品种是：{sellf.breed}，物种
        是：{self.species}，ID是{self.id}")
```
   b.
```
if __name__ = "__mian__":
    dog1 = Dog("旺财", "金毛")
```
   c. dog1.print_info()
   d. print(f"现在有{Dog.total_dog_created}只小狗")
   e. print(f"现在有{dog1.total_dog_created}只小狗") #使用实例访问类属性
4. 自己设计场景并写个封装、继承、多态的代码：pokemen_fight.ipynb
5. 还要设计一个类组合的关系代码，比如cs游戏，人是一个类，枪是一个类，人可以选择不同的枪。

作业

作业

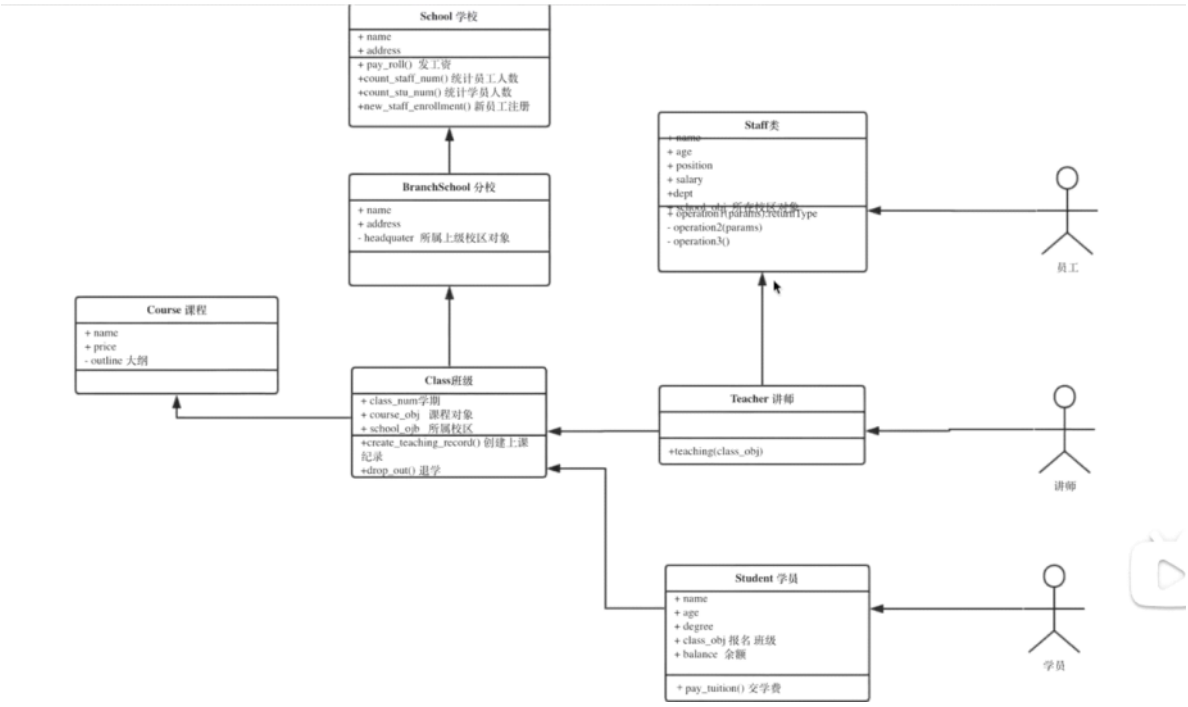
校园管理系统

设计一个培训机构管理系统，有总部、分校，有学员、老师、员工，实现具体如下需求：

1. 有多个课程，课程要有定价
2. 有多个班级，班级跟课程有关联
3. 有多个学生，学生报名班级，交这个班级对应的课程的费用
4. 有多个老师，可以分布在不同校区，上不同班级的课
5. 有多个员工，可以分布在不同校区在总部可以统计各校区的账户余额、员工人数、学员人数
6. 学生可以转校、退学

用到的类及类与类的结构了解一下。

1. 课程：要有名称、定价
2. 班级：要有学期、创建出课程对象



捕捉异常

2026年1月26日 18:18

1. 捕捉异常

把可能会发生的错误，提前在代码里进行捕捉（监测）

```
try:
    your code
except Exception as err:
    出错后要执行的代码
    print(err)
```

目的：程序出错后执行后续代码或者继续执行，不会直接中断程序并报错！

常见的出错类型：

NameError 定义不合法或未定义名称

ValueError 数值不合法，传入一个调用者不期望的值，有时候即使值的类型正确

AttributeError 试图访问一个对象没有的属性

IOError 输入或输出异常：基本上是无法打开文件，与文件处理有关

ImportError 无法引入模块或包：基本上是路径问题或名称错误

强类型错误	
IndentationError	语法错误（的子类）：代码没有正确对齐
SyntaxError	代码非法，代码不能编译
KeyboardInterrupt	

IndexError 下标索引超出序列边界

KeyError 试图访问字典里不存在的键

TypeError 传入对象类型与要求的不符合

UnboundLocalError 试图访问一个还未被设置的局部变量，基本上是由于另有一个同名的全局变量，导致你以为正在访问它

2. 万能异常
    a. except Exception as e
3. try... else... finally
    a. else: 没有发生异常则执行这段代码
    b. finally: 不管有无异常都执行这段代码
4. 主动触发错误：
    a. raise Error_name("xxx")
    b. 为什么要主动触发错误?

        如果你自己写了一个软件，那么应用层面的触发只能自己手动触发，那么就要用到主动触发错误

5. 自定义异常：
    a. class MyException(BaseException): (BaseException 是所有异常的基类)

```
        def __init__(self, msg):
            self.message = msg
        def __str__(self):
            return self.message
try:
        raise MyException("我的错误"}
except MyException as e:
        print(e)
```

    b. 万能异常无法捕捉自定义异常！
    c. 所以，需要自己捕捉自定义异常