面向对象编程基础

活在当下的程序员应该都听过“面向对象编程”一词，也经常有人问能不能用一句话解释下什么是“面向对象编程”，我们先来看看比较正式的说法。

> *把一组数据结构和处理它们的方法组成对象（object），把相同行为的对象归纳为类（class），通过类的封装（encapsulation）隐藏内部细节，通过继承（inheritance）实现类的特化（specialization）和泛化（generalization），通过多态（polymorphism）实现基于对象类型的动态分派。*

这样一说是不是更不明白了。所以我们还是看看更通俗易懂的说法，下面这段内容来自于知乎。

成心文
不会拍电影的程序员不是好机长

51 人赞同了该回答

一句话说明什么是面向对象？你个土鳖，你们全家都是土鳖！

好像有人说过这样的话，当头棒喝的方式虽然情感上不易接受，但记忆效果十分显著。

好吧，如果你觉得“土鳖”实在难听也不能准确定位你的档次，你可以自行将其替换为“土豪”，whatever。

面向对象思想有三大要素：封装、继承和多态。

- 封装：不管你是土鳖还是土豪，不管你中午吃的是窝头还是鲍鱼，你的下水都在你肚皮里，别人看不到你中午吃了啥，除非你自己说给他们听（或者画给他们看，whatever）；
- 继承：刚说了，你个土鳖/豪，你们全家都是土鳖/豪。冰冻三尺非一日之寒，你有今天，必定可以从你爸爸爷爷那里追根溯源。正所谓虎父无犬子，正恩同学那么狠，他爹正日就不是什么善茬，更甭说他爷爷日成，明白了吗？
- 多态：哲学家说过，世上不会有两个一模一样的双胞胎。即使你从你父亲那里继承来的土鳖/豪气质，也不可能完全是从一个模子里刻出来的，总会有些差别。比如你爸喜欢蹲在门前吃面，你喜欢骑在村口的歪脖子树上吃，或者反过来。当然，也可能令尊爱吃龙虾鲍鱼时旁边有几个艺校小女生喝酒唱歌助兴，你可能更喜欢弄个街舞乐队来吹拉弹唱。

> ***说明**：以上的内容来自于网络，不代表作者本人的观点和看法，与作者本人立场无关，相关责任不由作者承担。（终于有机会享受一下把这段话反过来说的乐趣了，乐得牙都快碎了。）*

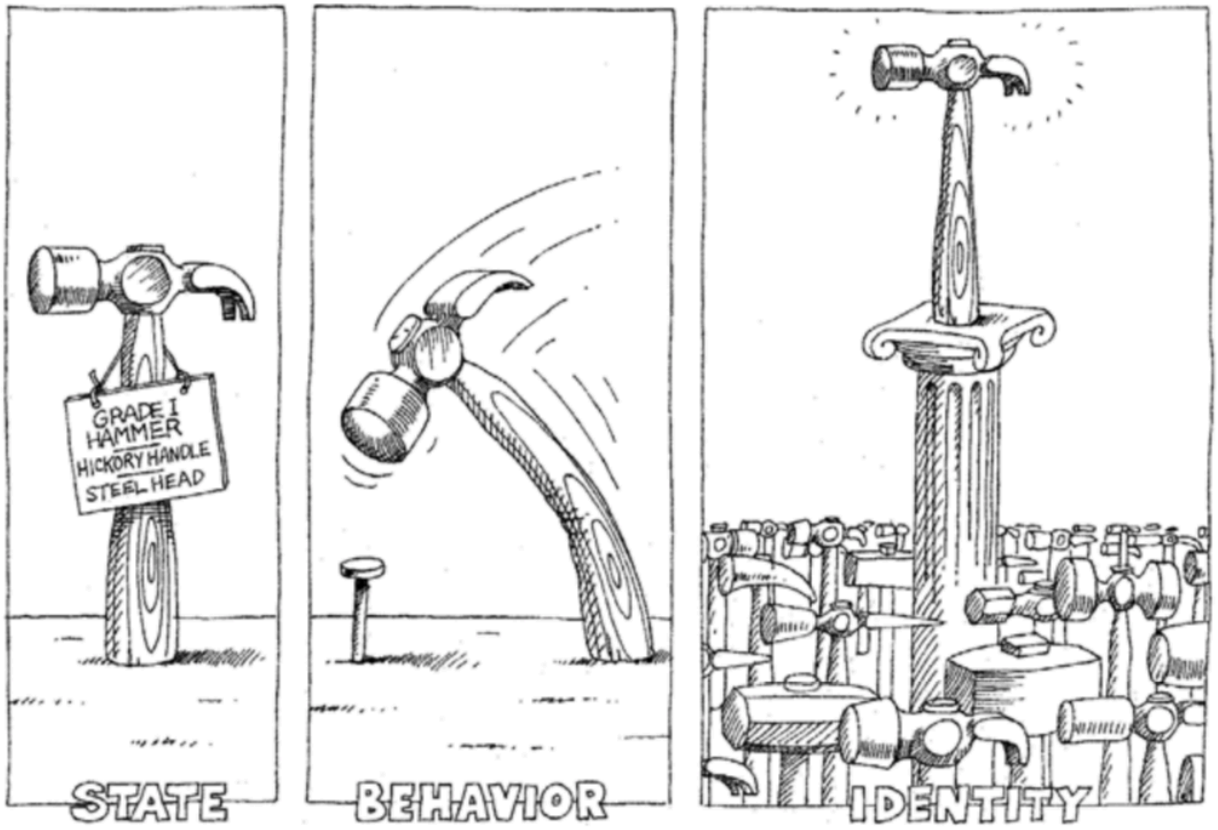
之前我们说过“程序是指令的集合”，我们在程序中书写的语句在执行时会变成一条或多条指令然后由CPU去执行。当然为了简化程序的设计，我们引入了函数的概念，把相对独立且经常重复使用的代码放置到函数中，在需要使用这些功能的时候只要调用函数即可；如果一个函数的功能过于复杂和臃肿，我们又可以进一步将函数继续切分为子函数来降低系统的复杂性。但是说了这么多，不知道大家是否发现，所谓编程就是程序员按照计算机的工作方式控制计算机完成各种任务。但是，计算机的工作方式与正常人类的思维模式是不同的，如果编程就必须得抛弃人类正常的思维方式去迎合计算机，编程的乐趣就少了很多，“每个人都应该学习编程”这样的豪言壮语就只能说说而已。当然，这些还不是最重要的，最重要的是当我们需要开发一个复杂的系统时，代码的复杂性会让开发和维护工作都变得举步维艰，所以在上世纪60年代末期，“软件危机”、“软件工程”等一系列的概念开始在行业中出现。

当然，程序员圈子内的人都知道，现实中并没有解决上面所说的这些问题的“银弹”，真正让软件开发者看到希望的是上世纪70年代诞生的Smalltalk编程语言中引入的面向对象的编程思想（面向对象编程的雏形可以追溯到更早期的Simula语言）。按照这种编程理念，程序中的数据和操作数据的函数是一个逻辑上的整体，我们称之为“对象”，而我们解决问题的方式就是创建出需要的对象并向对象发出各种各样的消息，多个对象的协同工作最终可以让我们构造出复杂的系统来解决现实中的问题。

> ***说明***：*当然面向对象也不是解决软件开发中所有问题的最后的“银弹”，所以今天的高级程序设计语言几乎都提供了对多种编程范式的支持，Python也不例外。*

类和对象

简单的说，类是对象的蓝图和模板，而对象是类的实例。这个解释虽然有点像用概念在解释概念，但是从这句话我们至少可以看出，类是抽象的概念，而对象是具体的东西。在面向对象编程的世界中，一切皆为对象，对象都有属性和行为，每个对象都是独一无二的，而且对象一定属于某个类（型）。当我们把一大堆拥有共同特征的对象的静态特征（属性）和动态特征（行为）都抽取出来后，就可以定义出一个叫做“类”的东西。



定义类

在Python中可以使用`class`关键字定义类，然后在类中通过之前学习过的函数来定义方法，这样就可以将对象的动态特征描述出来，代码如下所示。

```
class Student(object):

    # __init__是一个特殊方法用于在创建对象时进行初始化操作
    # 通过这个方法我们可以为学生对象绑定name和age两个属性
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def study(self, course_name):
        print('%s正在学习%s.' % (self.name, course_name))

    # PEP 8要求标识符的名字用全小写多个单词用下划线连接
    # 但是很多程序员和公司更倾向于使用驼峰命名法(驼峰标识)
    def watch_av(self):
        if self.age < 18:
            print('%s只能观看《熊出没》.' % self.name)
        else:
            print('%s正在观看岛国爱情动作片.' % self.name)
```

> *说明：写在类中的函数，我们通常称之为（对象的）方法，这些方法就是对象可以接收的消息。*

创建和使用对象

当我们定义好一个类之后，可以通过下面的方式来创建对象并给对象发消息。

```
def main():
    # 创建学生对象并指定姓名和年龄
    stu1 = Student('骆昊', 38)
    # 给对象发study消息
    stu1.study('Python程序设计')
    # 给对象发watch_av消息
    stu1.watch_av()
    stu2 = Student('王大锤', 15)
    stu2.study('思想品德')
    stu2.watch_av()


if __name__ == '__main__':
    main()
```

访问可见性问题

对于上面的代码，有C++、Java、C#等编程经验的程序员可能会问，我们给Student对象绑定的name和age属性到底具有怎样的访问权限（也称为可见性）。因为在很多面向对象编程语言中，我们通常会将对象的属性设置为私有的（private）或受保护的（protected），简单的说就是不允许外界访问，而对象的方法通常都是公开的（public），因为公开的方法就是对象能够接受的消息。在Python中，属性和方法的访问权限只有两种，也就是公开的和私有的，如果希望属性是私有的，在给属性命名时可以用两个下划线作为开头，下面的代码可以验证这一点。

```
class Test:

    def __init__(self, foo):
        self.__foo = foo

    def __bar(self):
        print(self.__foo)
        print('__bar')


def main():
    test = Test('hello')
    # AttributeError: 'Test' object has no attribute '__bar'
    test.__bar()
    # AttributeError: 'Test' object has no attribute '__foo'
```

```
    print(test.__foo)


if __name__ == "__main__":
    main()
```

但是，Python并没有从语法上严格保证私有属性或方法的私密性，它只是给私有的属性和方法换了一个名字来“妨碍”对它们的访问，事实上如果你知道更换名字的规则仍然可以访问到它们，下面的代码就可以验证这一点。之所以这样设定，可以用这样一句名言加以解释，就是“We are all consenting adults here”。因为绝大多数程序员都认为开放比封闭要好，而且程序员要自己为自己的行为负责。

```
class Test:

    def __init__(self, foo):
        self.__foo = foo

    def __bar(self):
        print(self.__foo)
        print('__bar')


def main():
    test = Test('hello')
    test._Test__bar()
    print(test._Test__foo)


if __name__ == "__main__":
    main()
```

在实际开发中，我们并不建议将属性设置为私有的，因为这会导致子类无法访问（后面会讲到）。所以大多数Python程序员会遵循一种命名惯例就是让属性名以单下划线开头来表示属性是受保护的，本类之外的代码在访问这样的属性时应该要保持慎重。这种做法并不是语法上的规则，单下划线开头的属性和方法外界仍然是可以访问的，所以更多的时候它是一种隐喻，关于这一点可以看看我的《Python - 那些年我们踩过的那些坑》文章中的讲解。

面向对象的支柱

面向对象有三大支柱：封装、继承和多态。后面两个概念在下一个章节中进行详细的说明，这里我们先说一下什么是封装。我自己对封装的理解是“隐藏一切可以隐藏的实现细节，只向外界暴露（提供）简单的编程接口”。我们在类中定义的方法其实就是把数据和对数据的操作封装起来了，在我们创建了对象之后，只需要给对象发送一个消息

（调用方法）就可以执行方法中的代码，也就是说我们只需要知道方法的名字和传入的参数（方法的外部视图），而不需要知道方法内部的实现细节（方法的内部视图）。

练习

练习1：定义一个类描述数字时钟

```
class Clock(object):
    """
    数字时钟
    """

    def __init__(self, hour=0, minute=0, second=0):
        """
        构造器

        :param hour: 时
        :param minute: 分
        :param second: 秒
        """
        self._hour = hour
        self._minute = minute
        self._second = second

    def run(self):
        """走字"""
        self._second += 1
        if self._second == 60:
            self._second = 0
            self._minute += 1
            if self._minute == 60:
                self._minute = 0
                self._hour += 1
                if self._hour == 24:
                    self._hour = 0

    def __str__(self):
        """显示时间"""
        return '%02d:%02d:%02d' % \
               (self._hour, self._minute, self._second)


def main():
    clock = Clock(23, 59, 58)
    while True:
        print(clock)
        sleep(1)
        clock.run()
```

```
if __name__ == '__main__':
    main()
```

练习2：定义一个类描述平面上的点并提供移动点和计算到另一个点距离的方法。

```
from math import sqrt


class Point(object):

    def __init__(self, x=0, y=0):
        """
        构造器

        :param x: 横坐标
        :param y: 纵坐标
        """
        self.x = x
        self.y = y

    def move_to(self, x, y):
        """
        移动到指定位置

        :param x: 新的横坐标
        "param y: 新的纵坐标
        """
        self.x = x
        self.y = y

    def move_by(self, dx, dy):
        """
        移动指定的增量

        :param dx: 横坐标的增量
        "param dy: 纵坐标的增量
        """
        self.x += dx
        self.y += dy

    def distance_to(self, other):
        """
        计算与另一个点的距离

        :param other: 另一个点
```

```
        """
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx ** 2 + dy ** 2)

    def __str__(self):
        return '(%s, %s)' % (str(self.x), str(self.y))


def main():
    p1 = Point(3, 5)
    p2 = Point()
    print(p1)
    print(p2)
    p2.move_by(-1, 2)
    print(p2)
    print(p1.distance_to(p2))


if __name__ == '__main__':
    main()
```

> ***说明***：*本章中的插图来自于Grady Booch等著作的《面向对象分析与设计》一书，该书是讲解面向对象编程的经典著作，有兴趣的读者可以购买和阅读这本书来了解更多的面向对象的相关知识。*