11 OOP in Python

> AI领域中的Python开发 — by 丁宁

SIGAI课程录制

本节课的OKR

本节课的Object

使用Python这门语言，在代码层面建立面向对象编程的直观感受

本节课与上节课的关系

- 上节课聚焦于思维方式，抛开代码去谈
- 本节课使用Python的OOP语法去实现常见的OOP的操作

本节课与下节课的关系

- 本节课聚焦于OOP，此时Python仅仅是众多可以实现OOP的语言之一
- 下节课聚焦于Python，此时侧重于更加Pythonic的编程方式，与其他语言完全不同

本节课的Key Result

1. 掌握OOP在Python语言中的基本语法，能用Python编写面向对象风格的程序
2. 清楚知道OOP的几个重要特性及目的

面向对象的Python实现

类的创建

```
class Model:
    pass


def main():
    model = Model()
    print(model)


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

输出：

```
<__main__.Model object at 0x108fa05f8>
```

要点：

1. 类名一般大写，实例化出来的对象，名称一般小写
2. 类在被定义时，也创建了一个局部作用域
3. 类名加上`()`生成对象，说明类被定义后便可以调用
4. 本质上讲，类本身就是一个对象

类的数据绑定

```
class Model:
    name = "CNN"


def main():
    print(Model.name)
    model = Model()
    print(model.name)
    model.name = "RNN"
    print(Model.name)
    print(model.name)


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

输出：

```
CNN
CNN
CNN
RNN
```

要点：

1. 类定义体中，可以定义自己的属性，并通过`.`来引用
2. 类实例化出对象以后，创建了新的局部作用域，也就是对象自己的作用域
3. 对实例化出来的对象引用属性时，先从自己的作用域找，未找到则向上找，这里找到了类作用域
4. 实例化出来的对象是可以在运行时绑定数据的

类的自定义实例化：`__init__()`

```
class Model:
    name = "DNN"

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


def main():
    cnnmodel = Model("CNN")
    rnnmodel = Model("RNN")
    print(Model.name, cnnmodel.name, rnnmodel.name)
    cnnmodel.name, rnnmodel.name = "RNN", "CNN"
    print(Model.name, cnnmodel.name, rnnmodel.name)


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

输出：

```
DNN CNN RNN
DNN RNN CNN
```

要点：

1. 类定义体中，`self`指代实例化出来的对象
2. 没有跟在`self`后面的属性属于类属性
3. 可以使用`__init__()`函数自定义初始化方式
4. 隶属于类的方法是共享的，隶属于对象的方式是每个对象私有的

对象方法

```
class Model:
    name = "DNN"

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

    def print_name(self):
        print(self.name)
```

```
def main():
    cnnmodel = Model("CNN")
    cnnmodel.print_name()
    cnnmodel.name = "RNN"
    cnnmodel.print_name()


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

输出:

```
CNN
RNN
```

要点:

1. 类定义体中的方法默认情况下隶属于对象，而非类本身
2. 直接在类上调用方法时会报错

> `cnnmodel.print_name()` 等价于 `Model.print_name(cnnmodel)`
> 那有没有隶属于类自己的方法呢?

类方法

```
class Model:
    name = "DNN"

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

    def print_name(self):
        print(self.name)

    @classmethod
    def print_cls_name(cls):
        print(cls.name)


def main():
    Model.print_cls_name()
    cnnmodel = Model("CNN")
```

```
    cnnmodel.print_name()
    cnnmodel.name = "RNN"
    cnnmodel.print_name()
    Model.print_cls_name()


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

输出:

```
DNN
CNN
RNN
DNN
```

要点:

1. 使用`@classmethod`与`cls`可以将方法绑定到类本身上

属性封装

```
class Model:
    __name = "DNN"

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

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

    @classmethod
    def print_cls_name(cls):
        print(cls.__name)


def main():
    Model.print_cls_name()
    cnnmodel = Model("CNN")
    cnnmodel.print_name()
    # print(Model.__name)
    # print(cnnmodel.__name)
```

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

两行注释分别执行后得到如下输出：

```
AttributeError: type object 'Model' has no attribute '__name'
AttributeError: 'Model' object has no attribute '__name'
```

要点：

1. 通过双下划线开头，可以将数据属性私有化，对于方法一样适用
2. 从报错信息也能看出，`Model`是一个`type object`，`cnnmodel`是一个`Model object`

> Python中的私有化是假的，本质上是做了一次名称替换，因此实际中也有为了方便调试而适用单下划线的情况，而私有化也就全凭自觉了

继承（隐式实例化）

```
class Model:
    __name = "DNN"

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

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

    @classmethod
    def print_cls_name(cls):
        print(cls.__name)


class CNNModel(Model):
    __name = "CNN"


def main():
    cnnmodel = CNNModel("Lenet")
    cnnmodel.print_name()
    CNNModel.print_cls_name()


if __name__ == '__main__':
```

```
    main()
```

输出：

```
Lenet
DNN
```

要点：

1. 如果子类没有定义自己的`__init__()`函数，则隐式调用父类的
2. 子类可以使用父类中定义的所有属性和方法，但类方法的行为需要注意

> 使用了`@classmethod`后的方法虽然可以继承，但是方法里面的`cls`参数绑定了父类，即使在子类中调用了类方法，但通过`cls`引用的属性依旧是父类的类属性

继承（显示实例化）

```
class Model:
    __name = "DNN"

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

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

    @classmethod
    def print_cls_name(cls):
        print(cls.__name)


class CNNModel(Model):
    __name = "CNN"

    def __init__(self, name, layer_num):
        Model.__init__(self, name)
        self.__layer_num = layer_num

    def print_layer_num(self):
        print(self.__layer_num)


def main():
```

```
    cnnmodel = CNNModel("Lenet", 5)
    cnnmodel.print_name()
    cnnmodel.print_layer_num()


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

输出:

```
Lenet
5
```

要点:

1. 如果子类中定义了`__init__()`函数，必须显示执行父类的初始化

多态

```
class Model:
    __name = "DNN"

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

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

    @classmethod
    def print_cls_name(cls):
        print(cls.__name)


class CNNModel(Model):
    __name = "CNN"

    def __init__(self, name, layer_num):
        Model.__init__(self, name)
        self.__layer_num = layer_num

    def print_name(self):
        print(self.__name)
        self.print_layer_num()
```

```
    def print_layer_num(self):
        print("Layer Num: ", self.__layer_num)


class RNNModel(Model):
    __name = "RNN"

    def __init__(self, name, nn_type):
        Model.__init__(self, name)
        self.__nn_type = nn_type

    def print_name(self):
        print(self.__name)
        self.print_nn_type()

    def print_nn_type(self):
        print("NN Type: ", self.__nn_type)


def print_model(model):
    model.print_name()


def main():
    model = Model("DNN")
    cnnmodel = CNNModel("CNN", 5)
    rnnmodel = RNNModel("RNN", "LSTM")
    [print_model(m) for m in [model, cnnmodel, rnnmodel]]


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

输出:

```
DNN
CNN
Layer Num:  5
RNN
NN Type:  LSTM
```

要点:

1. 多态的设计就是要完成对于不同类型对象使用相同的方法调用能得到各自期望的输出
2. 在数据封装，继承和多态中，多态是Python设计的核心，也叫鸭子类型

面向对象的重要特性总结

- 封装：只需知道我能做什么，不需要知道我怎么做的
- 继承：纵向复用，比如：每个技能树上的节点，都有类似的特征
- 多态：横向复用，比如：不同技能树上的节点，都有物理攻击和魔法伤害

> Python的设计初衷是强调多态，也就是所谓的鸭子类型，明白了Python的设计初衷后，我们编写的Python代码要在一定程度上避免以下三种情况（虽然写成这样也是可以的）：
>
> 1. C语言般的过程式编程
> 2. C++或Java般的传统面向对象编程
> 3. Lisp般的函数式编程

什么是Pythonic OOP呢？请快速移步下节课