12 Pythonic OOP

@(SIGAI课程录制)

本节课的OKR

本节课的Object

- 建立更加Pythonic的对OOP的认识
- 在更深层次认识Python这门语言
- 为后续的迭代器，生成器，上下文管理器等课程打下坚实基础

本节课的KR

- 建立认知：**Python中一切都是Object**
- 建立认知：**什么是Special Method，从Special Method的设计看Python的Duck Typing**
- 学会使用：**用`@property`管理对象属性的访问**
- 建立认知：**Cross-Cutting：从Multi-Inheritance及Decorator看Python中MixIn的设计**

说明

- 本节课的内容是Python这门语言学习的分界线
- 分界线之后的内容，课程不再教授，可自行按需学习
- 分界线之前的内容，都在**最小学习范围**之内
- 为了便于部分学员后续自学更加深入的内容，本节课开始逐步将术语替换为英文

关于双下划线

怎么读

- `_name`：single underscore name
- `__name`：double underscore name
- `__name__`：dunder name
- `__init__()`: dunder init method（function）

双下划线开头和结尾的变量或方法叫什么

- 类别：special；magic；dunder
- 实体：attribute；method

如何认识Python的special method（摘自官方文档）

- **special method**：method with special name（dunder）
- **Why use it?**：A class can implement certain operations that are invoked by special syntax
- **original intention of design**：*operator overloading*

> Allowing classes to define their own behavior with respect to language operators.

- 个人补充：尽可能的保证Python行为的一致性（Special Method更像是一种协议）

从语言设计层面理解Python的数据模型

一切都是对象

- Python中的数据模型是`Objects`
- Python程序中所有数据都是用`objects`或者`objects`之间的关系表示的
- 甚至Python代码都是`objects`

`Objects`的组成一：identity

- 当`objects`创建后，**identity**再也不会改变直到被销毁
- `id()`与`is`关注的就是一个`object`的**identity**

```
a = 1.0
print(id(a))
a = 1.1
print(id(a))
a = 1.0
print(id(a))
b = 1.0
print(id(b))
c = 1.1
print(id(c))
```

输出：

```
4410044608
4410044656
4410044608
4410044608
4410044656
```

要点：

1. 变量存的是创建的`object`的**identity**
2. 创建出来的不同的`object`有不同的**identity**
3. 变量的id变了不是因为`object`的**identity**变了，而是对应的`object`变了
4. 对于**immutable object**，计算结果如果已经存在可直接返回相同的**identity**

`Objects`的组成二：type

- 当`objects`创建后，其**type**也是不会改变的
- `type()`函数返回一个`Object`的**type**
- **type**决定了一个`object`支持哪些运算，可能的值在什么范围内

`Objects`的组成三：value

- 有些`Objects`的**value**是可以改变的：**mutable object**
- 有些`Objects`的**value**是不能改变的：**immutable object**
- 需要注意当一个`Object`是个**container**的情况

- 一个`Object`的**type**决定了它是否**mutable**

存放其他`Object`的reference的`Object`：Container

- 当我们聚焦于**Container**的**values**时，我们关注的是**value**
- 当我们聚焦于**Container**的**mutability**时，关注的是**identity**

Pythonic OOP with Special Method and Attribute

The implicit superclass – object & type object

- 每一个class在定义的时候如果没有继承，都会隐式继承`object`这个superclass
- 每一个自定义的class在Python中都是一个**type object**

```
class X:
    pass


class Y(X):
    pass


def main():
    x = X()
    y = Y()
    print(x.__class__.__name__)
    print(y.__class__.__name__)
    print(X.__class__.__name__)
    print(Y.__class__.__name__)
    print(x.__class__.__base__.__name__)
    print(y.__class__.__base__.__name__)
    print(X.__class__.__base__.__name__)
    print(Y.__class__.__base__.__name__)


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

输出：

```
X
Y
type
type
object
X
object
object
```

要点：

1. `object.__class__`
2. `class.__name__`
3. `class.__base__`
4. 注意：链式执行

Integrating Seamlessly with Python

```
class X:
    pass


class Y:
    """Class Y"""
    def __str__(self):
        return "{} object".format(self.__class__.__name__)

    def __len__(self):
        return 10

    def __bool__(self):
        return False


def check_bool(x):
    if x:
        print("I'm {}. My bool value is True.".format(str(x)))
    else:
        print("I'm {}. My bool value is False.".format(str(x)))


def main():
    x = X()
    y = Y()
    print(x)
    print(y)
    # print(len(x))
    print(len(y))
    check_bool(x)
    check_bool(y)
    print(X.__doc__)
    print(Y.__doc__)


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

输出：

```
<__main__.X object at 0x103306e80>
Y object
10
I'm <__main__.X object at 0x103306e80>. My bool value is True.
I'm Y object. My bool value is False.
None
Class Y
```

要点：

1. 之所以要实现special method，是为了让自定义的class与Python的内置函数无缝衔接
2. Python有大量的内置函数，而这些函数大部分都是调用的对象里的special method
3. 想查看Python中到底有多少special method：点我

Attribute Access and Properties

Attribute相关的操作一般有：

- Create
- Read
- Update
- Delete

> 数据库领域中的**CRUD**亦是如此

Level I：Basic Access（Default Access）

```python
class X:
    pass


if __name__ == "__main__":
    # print(X.a)
    X.a = "a"
    print(X.a)
    X.a = "aa"
    print(X.a)
    del X.a
    # print(X.a)
```

输出：

```
a
aa
```

说明：

1. 默认情况下，CRUD都支持，而且是在public情况下都支持（除了双下划线开头的）
2. 如果对象中没有这个Attribute，访问时会报错
