Lecture 4: 从 POP 到 OOP——奥卡姆剃刀原则

Xiao-Xin Li

Zhejiang University of Technology

Revision: *2023/04/24*

面向过程的程序设计（Procedure-Oriented Programming，POP）是初学者最容易接受的编程方式。这种方式在处理简单需求时看起来是个不错的选择，但如果是处理复杂的需求，POP 往往会使代码陷入混乱而且不便于后期维护。总体上来说，OOP 的程序设计思想遵循奥卡姆剃刀原则，即 “尽量精简” 的原则。这一原则是普适用的，我们做任何事情都可以遵循这一原则来提高效率和质量。

1. Hangman

本节以 Assignment 1: Hangman 为例，来讨论 OOP 的设计思想。

1.1 面向过程的实现：一份来自于同学的代码

```python
import random

def hangman():
    word_list = ["child", "adult", "kid", "baby", "women", "men", "eat", "sleep"
    secret_word = random.choice(word_list)
    display_word = ['*' for _ in secret_word]
    # display_word = '*' * len(secret_word)  # 为什么不是这样初始化 display_word 呢
    wrong_guesses = 0
    max_wrong_guesses = 6

    while wrong_guesses < max_wrong_guesses:
        print(''.join(display_word))
        guess = input("Guess a letter or the whole word: ").lower()

        if guess == secret_word:
            print("You won! The word was:", secret_word)
            break
        elif len(guess) == 1 and guess in secret_word:
            for i, letter in enumerate(secret_word):
                if letter == guess:
                    display_word[i] = guess
```

```
            if '*' not in display_word:
                print("You won! The word was:", secret_word)
                break
        else:
            wrong_guesses += 1
            print("Wrong guess. You have", max_wrong_guesses - wrong_guesses, "g
    else:
        print("You lost. The word was:", secret_word)

hangman()
```

👽 这是一个很好的面向过程的实现，代码非常简洁，能够很好地运用 `list` 这样的数据结构。

1.1.1 问题1：程序结构不是很好

1. 尽量避免距离较远的逻辑控制；
2. 尽量使用 `return` / `break` / `continue` 等控制语句，替代 `if` / `else` 语句

```
def hangman():
    ......

    while wrong_guesses < max_wrong_guesses:
        print(''.join(display_word))
        guess = input("Guess a letter or the whole word: ").lower()

        if guess == secret_word:
            print("You won! The word was:", secret_word)
            break return
        elif len(guess) == 1 and guess in secret_word:
            for i, letter in enumerate(secret_word):
                if letter == guess:
                    display_word[i] = guess
            if '*' not in display_word:
                print("You won! The word was:", secret_word)
                break return
        else:
            wrong_guesses += 1
            print("Wrong guess. You have", max_wrong_guesses - wrong_guesses, "g
    else:
        print("You lost. The word was:", secret_word)
```

修改后的程序：

```python
def hangman():
    ......

    while wrong_guesses < max_wrong_guesses:
        print(''.join(display_word))
        guess = input("Guess a letter or the whole word: ").lower()

        if guess == secret_word:
            print("You won! The word was:", secret_word)
            return

        if len(guess) == 1 and guess in secret_word:
            for i, letter in enumerate(secret_word):
                if letter == guess:
                    display_word[i] = guess

            if '*' not in display_word:
                print("You won! The word was:", secret_word)
                return

            continue

        wrong_guesses += 1
        print("Wrong guess. You have", max_wrong_guesses - wrong_guesses, "guess

    print("You lost. The word was:", secret_word)
```

1.1.2 问题2：核心逻辑不够简洁

1. 对于读者：可能会给阅读程序的人造成很大的理解障碍，尤其是当面对一个复杂工程的时候，这里是一个例子（看到这里的代码，你会觉得这个代码写得很好呢？还是很不好呢？）：https://github.com/yubangji123/Interpret_FR/blob/master/train/model_pretrain_CASIA.py
2. 对于开发者：
   a. 面对复杂的项目，可能一开始不知道如何入手，尽管项目的整体逻辑并没有那么复杂；
   b. 不利于维护：bug可能是局部的，但由于所有的逻辑都放在一起，可能牵一发而动全身；
3. 对于测试者：不利于有针对性的测试；
4. 对于设计者：不利于下发接口实现任务。

为了克服上述问题，我们需要简化上面的代码。这里，简化代码需要遵循一个原则：*Functional Core, Imperative Shell (FCIS)*，即：功能式内核，命令式外壳。

```
def hangman():
    while wrong_guesses < max_wrong_guesses:
        guess()

        if guess_correct():
            report_win_game()
            return

        if valid_guess():
            update_display_word()
            # 经过对核心逻辑的提炼，我们会发现只需要在逐个字符猜测密码时，才需要报告猜测
            report_guess_progress()
            continue

        guess_wrong()

    report_lose_game()
```

注意，上面的代码只是简单表达了程序的核心逻辑，要想真的运行还需要合理的传递参数，然而，我们会发现，如果仍然按照面向过程的编程方式，只是传递参数就是一件很麻烦的事情。此时，我们会发现，OOP 是一个很好的选择。

1.2 基于 OOP 的实现

下面我们给出了 `Hangman` 类的接口。基于 FCIS 的理念，你会发现你自己也可以做架构设计师了。

```
class Hangman:
    def __init__(self):
        pass

    def report_win_game(self):
        pass

    def report_lose_game(self):
        pass

    def report_guess_progress(self):
        pass

    def update_display_word(self):
        pass

    def guess(self):
        pass

```

```
20      def valid_guess(self):
21          pass
22
23      def guess_correct(self):
24          pass
25
26      def guess_wrong(self):
27          pass
28
29      def __call__(self):
30          """
31              注意，我们通常使用魔法函数 __call__() 来定义类的核心功能，而不是为其另外取一个
32              这样做也是为了方便调用。
33          """
34          while self.fail_times < self.max_fail_times:
35              self.guess()
36
37              if self.guess_correct():
38                  self.report_win_game()
39                  return
40
41              if self.valid_guess():
42                  self.update_display_word()
43                  self.report_guess_progress()
44                  continue
45
46              self.guess_wrong()
47
48          self.report_lose_game()
49
50
51  """ 这里，为什么要用 if 语句呢? """
52  if __name__ == '__main__':
53      Hangman()()  # 我们定义了类的魔法函数 __call__()，就无须按照下面的方式调用了
54      hangman = Hangman()
55      hangman.__call__()
```

这里，值得讨论的几个问题是：

1. 显然，`Hangman` 类的代码长度要远远超过面向过程式的实现方式（`hangman()` 方法）了，这样代码岂不是变得更加复杂了？是否有违于我们的“奥卡姆剃刀原则”？
2. `Hangman` 类的运行效率否会比 `hangman()` 高呢？答案是否定的。为什么？
3. 对软件系统的各功能模块的划分是否越细致越好？

1.3 Hangman的功能晋级：每次猜测，只能猜全部字符或者~~逐个字符依次猜测~~只能猜测单个字符，二选一

1. OOP 的三大理念：封装、继承、多态；
2. 父类和子类的设计要遵循 **LSP (Liskov Substitution Priciple) 准则**：为父类设计的公开接口/方法，要能够被任意一个子类调用，也就是，父类中的 self 对象要允许被天替换成任意一个子类对象。

```python
class Hangman:
    def __call__(self):
        print('Welcome to `Hangman` ^_^!')
        print('---------------------------------')
        print('1. Guess the password letter by letter.')
        print('2. Guess the whole password.')
        choice = input('Your choice: ')
        choice = choice.strip()
        choice = 1 if choice == '' else int(choice)
        assert choice in [1, 2]

        if choice == 2:
            GuessPasswdByWholeWord()()
            return

        GuessPasswdLetterByLetter()()
```

```python
class GuessPasswd:
    def __init__(self):
        self._candidates = ['HELLO']
        # self._candidates = [
        #     'AMAZON', 'BACKGROUND', 'CORONAVIRUS', 'DOCUMENT', 'END',
        #     'FORMAl', 'GITHUB', 'HELLO', 'IPHONE', 'JETBRAIN',
        #     'KNOWN', 'LIST', 'MICROSOFT', 'NOBEL']

        self.fail_times = -1
        self.max_fail_times = 5
        self._passwd: str = random.choice(self._candidates)
        self.display_word = '*' * len(self._passwd)
        print('The potential password is: ' + self.display_word)

    def report_lose_game(self):
        print(f'Player lost after wrongly guessing {self.fail_times} times!')

    def report_win_game(self):
```

```
        print(f'Congratulations! Player won after wrongly guessing {self.fail_ti

    def guess(self):
        raise NotImplementedError

    def guess_wrong(self):
        raise NotImplementedError

    def __call__(self):
        while self.guess_wrong():
            if self.fail_times > self.max_fail_times:
                self.report_lose_game()
                return
            self.guess()

        self.report_win_game()
```

```
class GuessPasswdByWholeWord(GuessPasswd):
    def guess_wrong(self):
        if self._passwd == self.display_word:
            return False

        self.fail_times += 1
        if self.fail_times > 0:
            print(f'Guess Wrong {self.fail_times} times!')
        return True

     def guess(self):
        self.display_word = input('Guess: ').upper()
```

```
class GuessPasswdLetterByLetter(GuessPasswd):
    def __init__(self):
        super().__init__()
        self.fail_times = 0
        self.guessed_letter = ''
        self.guessed_letters = []
        self.display_word = '*' * len(self._passwd)

    def _update_display_word(self):
        pass

    def _valid_guess(self):
        pass
```

```
14
15      def guess_wrong(self):
16          pass
17
18      def guess(self):
19          pass
20
```

2. 高尔夫足球赛

2.1 系统需求

下面是一场真实的高尔夫足球赛的得分情况，得分最低者胜出。各参赛选手的得分情况以 `姓名 + ':'/': ' + 各分项得分` 进行记录，且各分项得分之间以一个或多个空格间隔，得分情况可以记录在一个字符串中，也可以记录在一个名为“scores.txt”的文本文件（详见 scores.txt）中。试基于OOP编写Python程序，计算并输出每位选手的总得分，输出胜出的选手，并且在计算胜出选手时，不可以使用系统提供的 `min` / `max` 函数。

```
all_scores = '\
李: 17 6 3 3   2 5 3 6 8\
楼: 6  9 5 10 14 3 8 4 6\
翁: 3  4 8 9   3 5 4 6 6'
```

2.2 实现细节

1. 文件还是字符串：
   a. 系统输入可以是一个记录实际得分的字符串，也可以是一个文件，如何区分？
   b. 是否需要为处理文件输入和字符串输入建立不同的子类？
2. 读文件
   a. 读取文件时，可能遇到中文乱码的情况，因此，需要以字节方式读取文件，且需要使用 `chardet` 包对读取的数据进行解码：

```
with open('scores.txt', 'rb') as f:
    all_scores = f.read()
    f_type = chardet.detect(all_scores)
    all_scores = all_scores.decode(f_type['encoding'])
```

```
pip install chardet
```

   b. 读取文件时，可以是一行一行读取，也可以是一次性读取文件的所有内容，我们应该选择哪种方式？
3. 如何分离各个参赛选手的得分？
   a. 如何处理中文的冒号和英文的冒号两种情况？
   b. 各选手的分项得分的空白间隔是不同的，如何处理？

4. 如何设计类的接口?

```
class AnalyScore:
    def __init__(self, all_scores):
        """ all_scores can be a string or a text file"""
        self.all_scores = self.read_scores(all_scores)
        pass

    def read_scores(self, all_scores):
        pass

    def parse_scores(self):
        pass


    def print_scores(self):
        pass

    def report_winner(self):
        pass

    def __call__(self):
        self.parse_scores()
        self.print_scores()
        self.report_winner()


if __name__ == '__main__':
    player_scores = '\
        李: 17 6 3 3   2 5 3 6 8\
        楼: 6  9 5 10 14 3 8 4 6\
        翁: 3  4 8 9   3 5 4 6 6'

    AnalyScore(player_scores)()

    print()
    AnalyScore('scores.txt')()
```

3. 期中考试

1. 高尔夫足球赛(40分) + 猜字游戏(60分)
2. 请大家务必遵守 Exam Rules 中的规定;
3. **期中考试，我们要考查的主要内容是:**

a. **1.3节**：Hangman的功能晋级：只能猜全部字符或者逐个字符依次猜测，二选一；

b. **第2节**：高尔夫足球赛，也是这个作业：Assignments 3: 高尔夫足球赛。

4. **只能按照我们给出的接口给出你的实现代码，这是为了统一评分之便，请大家理解。**

5. 关于高尔夫足球赛的实现细节补充：

   a. 字符串中的空格的处理，不必那么复杂，根据我们班刘靖杰同学的建议，大家直接调用 `.split()` ，不传入任何参数，即可将比赛的各分项成绩分离开来；

   b. 将分项成绩字符串，转成list，再转成数值，可以用下面的代码：

```
score = '4 7  5'  # 只是一个例子
score_list = [int(c) for c in score if c != ' ']    # 转换后的成绩

# 也可以用：
score_list = list(map(int, score.split()))  # 转换后的成绩
```