

公有方法、私有方法与 方法重载

Python程序设计 翻转课堂 第 13 组
陶唐 李明钰 马翔 张耀函

大家好，我们是第13组，我是xx，为大家带来关于Python中“公有方法、私有方法与方法重载”的讲解。

我们将通过一个实际项目——Translator类的实现，来展示这些概念在实际编程中的应用。希望大家能从中获得启发和收获！

项目要求：Translator

- 要求实现一个Translator类
 - 可以将任意语言的文字翻译为指定语言。
- 我们遇到一些挑战
 - 文字的类型不一定
 - 字符串
 - 字符串列表
 - 文件
 - “指定语言”
 - 大部分场景下都是英文
 - 少数场景下会有指定要求

我们的项目目标是实现一个Translator类，它可以将任意语言的文字翻译成指定语言。

这个类需要处理不同的输入格式，比如字符串、字符串列表或者文件。

同时，“指定语言”在大多数情况下是英文，但在某些特殊场景下，用户可能会有其他语言需求。

在实现过程中，我们遇到了一些挑战。

首先是输入数据格式的多样性，我们需要处理字符串、列表和文件等多种形式。

其次，是关于“指定语言”的处理，虽然大多数情况是英文，但也要支持其他语言的灵活切换。

这些挑战促使我们思考如何设计一个结构清晰、可维护性强的类。

项目要求：Translator

```
1 text = "La lueur du soleil couchant qui frappait en plein son visage pâlisait le lasting de sa soutane,  
luisante sous les coudes, effiloquée par le bas. Des taches de graisse et de tabac suivaient sur sa  
poitrine large la ligne des petits boutons, et elles devenaient plus nombreuses en s'écartant de son  
rabat, où reposaient les plis abondants de sa peau rouge; elle était semée de macules jaunes qui  
disparaissaient dans les poils rudes de sa barbe grisonnante. Il venait de dîner et respirait  
bruyamment."  
2  
3 text_in_list = [  
4     "Debout, les damnés de la terre",  
5     "Debout, les forçats de la faim",  
6     "La raison tonne en son cratère",  
7     "C'est l'éruption de la fin",  
8     "Du passé faisons table rase",  
9     "Foule esclave, debout, debout",  
10    "Le monde va changer de base",  
11    "Nous ne sommes rien, soyons tout",  
12 ]  
13  
14 file = open("text/Also sprach Zarathustra.txt", "r")  
15
```

项目要求: Translator

```
1 from trans.translator import Translator
2 from trans.language import Language
3
4 text = ...
5
6 translator = Translator()
7 translated_text = translator.translate(text)
8
9 translated_chinese_text = translator.translate(text, Language.CHINESE)
10
```

实现： 百度API

- 使用百度的翻译API来完成
- 百度翻译平台的文档： 每一次请求都需要生成随机数和验证签名。
- 把生成随机数和验证签名的函数写成一个类方法
- 等等！**
 - 使用Translator类的程序员不应该随便调用这个方法
 - 因为这个验证签名含有敏感鉴权信息
 - 这个方法应当只能在这个类内部调用，而不在外部调用。

为了实现翻译功能，我们决定使用百度翻译API。

根据文档说明，每次请求都需要生成随机数和验证签名。

我们考虑将这个功能封装为类方法，使代码结构更清晰。

但随之而来的问题是：我们不希望这个方法被外部随意调用，因为它们涉及鉴权信息，存在安全风险。

实现：随机数和验证签名

Bad!

```
1 from trans.baidu import BaiduTranslator
2
3 translator = BaiduTranslator()
4
5 salt, sign = translator.get_salt_and_sign()
6
7 ... = translator.translate(..., salt, sign)
8
```

Good!

```
1 from trans.baidu import BaiduTranslator
2
3 translator = BaiduTranslator()
4
5 # 在调用translate方法时，在内部计算了salt和sign。
6 ... = translator.translate(...)
7
```

我们对比了两种实现方式：一种是将生成签名的方法作为公有方法暴露出来，另一种是将其设置为私有方法。

显然，后者更加安全，也更符合封装的设计原则。通过将敏感操作限制在类内部，我们提升了代码的安全性和可维护性。

实现：内部代码

- 在 Python 中，一个以双下划线开头，不以双下划线结尾的方法被视为是一个内部方法。
- 以双下划线开头又以双下划线结尾的方法被称为“魔法方法”。
- 这里 `__make_md5` 和 `__get_salt_and_sign` 是 `BaiduTranslator` 的私有方法，只可以在内部调用。
- 而 `translate` 方法是公有方法，可以在外部调用。

```
1 class BaiduTranslator():
2     def __init__(self):
3         ...
4
5     def __make_md5(self, s):
6         return md5(s.encode("utf-8")).hexdigest()
7
8     def __get_salt_and_sign(self, text):
9         salt = random.randint(32768, 65536)
10        sign = self.__make_md5(self.appid + text + str(salt) + self.appkey)
11        return salt, sign
12
13    def translate(self, text, target_lang):
14        ...
15
16        salt, sign = self.__get_salt_and_sign(text)
17
18        params = {
19            "appid": self.__appid,
20            "q": text,
21            "from": "auto",
22            "to": target_lang,
23            "salt": salt,
24            "sign": sign,
25        }
26
27        r = requests.post(
28            self.__url,
29            params=params,
30            headers={"Content-Type": "application/x-www-form-urlencoded"},
31        )
32
33        return r.json()["trans_result"][0]["dst"]
34
```

在Python中，我们可以通过在方法名前加双下划线“__”来定义私有方法。

例如，`__make_md5`和`__get_salt_and_sign`就是我们定义的私有方法。

这些方法只能在类内部调用，外部无法直接访问。

需要注意的是，以双下划线开头且以双下划线结尾的方法是“魔法方法”，如 `__init__`，它们有特殊用途，不属于私有方法范畴。

实现：不同类型

```
1 class BaiduTranslator(BaseTranslator):  
2     ...  
3  
4     def translate(self, text, target_lang):  
5         if isinstance(text, list):  
6             text = " ".join(text)  
7         elif isinstance(text, TextIOBase):  
8             text = text.read().encode("utf-8")  
9  
10        ...
```

为了支持不同类型的输入，我们对translate方法进行了扩展。

通过isinstance来判断传入文字的类型，并统一将其转换为字符串。

这种灵活性使得我们的Translator类可以适应多种使用场景。

实现：默认语言

```
1 class BaiduTranslator(BaseTranslator):
2     ...
3
4     def translate(self, text, target_lang=Language.ENGLISH):
5         if isinstance(text, list):
6             text = " ".join(text)
7         elif isinstance(text, TextIOBase):
8             text = text.read().encode("utf-8")
9
10        ...
```

```
1 from trans.baidu import BaiduTranslator
2 from trans.language import Language
3
4 text = ...
5
6 translator = BaiduTranslator()
7 translated_english_text = translator.translate(text)
8 translated_chinese_text = translator.translate(text, target_lang=Lang.CHINESE)
9
```

在大多数情况下，用户只需要翻译成英文，因此我们在代码中设置了默认语言参数。

如果用户没有特别指定目标语言，则自动翻译为英文。

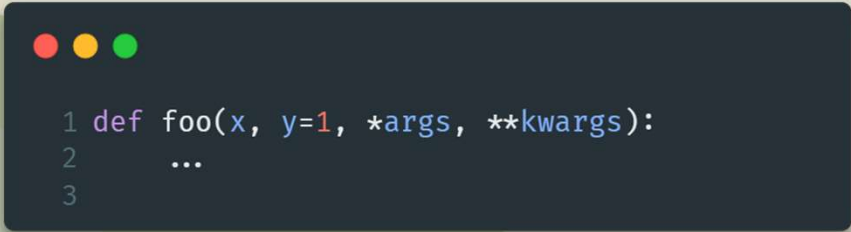
当然，用户也可以通过参数显式指定其他语言，从而满足个性化需求。

讲解：重载

➤上面这种实现“明明是同一个方法，但是可以处理不同的数据类型乃至不同数量参数”的技术叫做“重载”。

➤print()

➤max()



```
1 def foo(x, y=1, *args, **kwargs):  
2     ...  
3
```

上面这种实现“同一个方法可以处理不同数据类型甚至不同数量参数”的技术，我们称之为**方法重载**。

Python本身并不像其它语言那样直接支持方法重载，但我们可以通过默认参数、可变参数等方式模拟实现。

例如Python内置的print()和max()函数就展示了类似的功能。

对比：与其它支持OOP的语言

C++

```
1 class Person {  
2     private:  
3         Thought think_about(Thing something);  
4     public:  
5         void say_hi();  
6         void say_hi(auto name);  
7 }
```

Python

```
1 class Person:  
2     def __think_about(something):  
3         pass  
4     def say_hi(name=None):  
5         pass
```

Q&A

总结一下，我们通过实现Translator类，展示了Python中**公有方法与私有方法**的区别，以及如何通过参数设计实现**方法重载**。

这些技术不仅提升了代码的安全性和可维护性，也增强了程序的灵活性。

接下来是提问环节，欢迎大家提出问题，我们将尽力解答！

谢谢

感谢大家的聆听！

希望我们的分享能帮助大家更好地理解Python中的类设计与方法使用。

谢谢！