

阅读: 125581

在Class内部，可以有属性和方法，而外部代码可以通过直接调用实例变量的方法来操作数据，这样，就隐藏了内部的复杂逻辑。

但是，从前面Student类的定义来看，外部代码还是可以自由地修改一个实例的name、score属性：

```
>>> bart = Student('Bart Simpson', 98)
>>> bart.score
98
>>> bart.score = 59
>>> bart.score
59
```

如果要让内部属性不被外部访问，可以把属性的名称前加上两个下划线__，在Python中，实例的变量名如果以__开头，就变成了一个私有变量（private），只有内部可以访问，外部不能访问，所以，我们把Student类改一改：

```
class Student(object):

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

    def print_score(self):
        print('%s: %s' % (self.__name, self.__score))
```

改完后，对于外部代码来说，没什么变动，但是已经无法从外部访问实例变量.__name和实例变量.__score了：

```
>>> bart = Student('Bart Simpson', 98)
>>> bart.__name
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Student' object has no attribute '__name'
```

这样就确保了外部代码不能随意修改对象内部的状态，这样通过访问限制的保护，代码更加健壮。

但是如果外部代码要获取name和score怎么办？可以给Student类增加get_name和get_score这样的方法：

```
class Student(object):
    ...

    def get_name(self):
        return self.__name

    def get_score(self):
        return self.__score
```

如果又需要允许外部代码修改score怎么办？可以再给Student类增加set_score方法：

如果又要允许外部代码修改score怎么办？可以再给Student类增加set_score方法：

```
class Student(object):
    ...

    def set_score(self, score):
        self.__score = score
```

你也许会问，原先那种直接通过**bart.score = 59**也可以修改啊，为什么要定义一个方法大费周折？因为在方法中，可以对参数做检查，避免传入无效的参数：

```
class Student(object):
    ...

    def set_score(self, score):
        if 0 <= score <= 100:
            self.__score = score
        else:
            raise ValueError('bad score')
```

需要注意的是，在Python中，变量名类似**__xxx__**的，也就是以双下划线开头，并且以双下划线结尾的，是特殊变量，特殊变量是可以直接访问的，不是private变量，所以，不能用**__name__**、**__score__**这样的变量名。

有些时候，你会看到以一个下划线开头的实例变量名，比如**__name__**，这样的实例变量外部是可以访问的，但是，按照约定俗成的规定，当你看到这样的变量时，意思就是，“虽然我可以被访问，但是，请把我视为私有变量，不要随意访问”。

双下划线开头的实例变量是不是一定不能从外部访问呢？其实也不是。不能直接访问**__name__**是因为Python解释器对外把**__name__**变量改成了**__Student__name__**，所以，仍然可以通过**__Student__name__**来访问**__name__**变量：

```
>>> bart.__Student__name
'Bart Simpson'
```

但是强烈建议你不要这么干，因为不同版本的Python解释器可能会把**__name__**改成不同的变量名。

总的来说就是，Python本身没有任何机制阻止你干坏事，一切全靠自觉。

最后注意下面的这种错误写法：

```
>>> bart = Student('Bart Simpson', 98)
>>> bart.get_name()
'Bart Simpson'
>>> bart.__name = 'New Name' # 设置__name变量！
>>> bart.__name
'New Name'
```

表面上看，外部代码“成功”地设置了**__name__**变量，但实际上这个**__name__**变量和class内部的**__name__**变量不是一个变量！内部的**__name__**变量已经被Python解释器自动改成了**__Student__name__**，而外部代码给**bart**新增了一个**__name__**变量。不信试试：

```
>>> bart.get_name() # get_name()内部返回self.__name
'Bart Simpson'
```

参考源码

感觉本站内容不错，读后有收获？

¥ 我要小额赞助，鼓励作者写出更好的教程

还可以分享给朋友

分享 [alvysinger](#) 等1人分享过



珠峰NODE.JS全栈开发

技术陪伴成长社区 珠峰培训

掘金

一个只有高手分享的技术社区

立即加入

麦子学院 [www.maiziedu.com](#)

百万级python导师亲身指导

保你120天
变身python大牛

有时候，你需要的只是一句点拨

立即咨询

深度学习在线课程

通向无人驾驶的必经之路

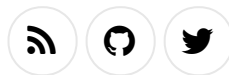
评论

发表评论

Sign In to Make a Comment



廖雪峰的官方网站©2015
Powered by [iTranswarp.js](#)
由[阿里云](#)托管
[广告合作](#)



友情链接: [中华诗词](#) - [阿里云](#) - [SICP](#) - [4closure](#)