封装是面向对象编程语言对客观世界的模拟，在客观世界里，对象的状态信息都被隐藏在对象内部，外界无法直接操作和修改。对一个类或对象实现良好的封装，可以达到以下目的：

隐藏类的实现细节。
让使用者只能通过事先预定的方法来访问数据，从而可以在该方法里加入控制逻辑，限制对属性的不合理访问。
可进行数据检查，从而有利于保证对象信息的完整性。
便于修改，提高代码的可维护性。
为了实现良好的封装，需要从以下两个方面来考虑：

将对象的属性和实现细节隐藏起来，不允许外部直接访问。
把方法暴露出来，让方法来控制对这些属性进行安全的访问和操作。
因此，实际上封装有两个方面的含义：把该隐藏的隐藏起来，把该暴露的暴露出来。

Python 并没有提供类似于其他语言的 private 等修饰符，因此 Python 并不能真正支持隐藏。为了隐藏类中的成员，Python 玩了一个小技巧：只要将 Python 类的成员命名为以双下画线开头的，Python 就会把它们隐藏起来。

例如，如下程序示范了 Python 的封装机制：

```
class User : def __hide(self): print('示范隐藏的 hide 方法') def getname(self): return self.__name def setname(self, name): if len(name) < 3 or len(name) > 8: raise ValueError('用户名长度必须在 3～8 之间') self.__name = name name = property(getname, setname) def setage(self, age): if age < 18 or age > 70: raise ValueError('用户名年龄必须在 18 在 70 之间') self.__age = age def getage(self): return self.__age age = property(getage, setage) # 创建 User 对象 u = User() # 对 name 属性赋值，实际上调用 setname()方法 u.name = 'fk' # 引发 ValueError: 用户名长度必须在 3～8 之间
```

上面程序将 User 的两个实例变量分别命名为 __name 和 __age，这两个实例变量就会被隐藏起来，这样程序就无法直接访问 __name、__age 变量，只能通过 setname()、getname()、setage()、getage() 这些访问器方法进行访问，而 setname()、setage() 会对用户设置的 name、age 进行控制，只有符合条件的 name、age 才允许设置。

上面程序用到了 raise 关键字来抛出异常，关于 raise 关键字和异常的相关信息，后续章节会详细介绍。

上面程序尝试将 User 对象的 name 设为 fk，这个字符串的长度为“2”不符合实际要求，因此运行程序最后一行包含如下错误：

ValueError：用户名长度必须在 3-8 之间

将最后一行代码注释掉，并在程序尾部添加如下代码：

```
u.name = 'fkit' u.age = 25 print(u.name) # fkit print(u.age) # 25
```

此时程序对 name、age 所赋的值都符合要求，因此上面两行赋值语句完全可以正常运行。运行上面代码，可以看到如下输出结果：

```
fkit
25
```

从该程序可以看出封装的好处，程序可以将 User 对象的实现细节隐藏起来，程序只能通过暴露出来的 setname()、setage() 方法来改变 User 对象的状态，而这两个方法可以添加自己的逻辑控制，这种控制对 User 的修改始终是安全的。

上面程序还定义了一个 hide() 方法，这个方法默认是隐藏的。如果程序尝试执行如下代码：

```
# 尝试调用隐藏的__hide()方法  u.__hide()
```

将会提示如下错误：

```
AttributeError:'User' object has no attribute 'hide'
```

最后需要说明的是，Python 其实没有真正的隐藏机制，双下画线只是 Python 的一个小技巧，Python 会“偷偷”地改变以双下画线开头的方法名，会在这些方法名前添加单下画线和类名。因此上面的 __hide() 方法其实可以按如下方式调用（通常并不推荐这么干）：

```
# 调用隐藏的__hide()方法  u._User__hide()
```

运行上面代码，可以看到如下输出结果：

```
示范隐藏的 hide 方法
```

通过上面调用可以看出，Python 并没有实现真正的隐藏。

类似的是，程序也可通过为隐藏的实例变量添加下画线和类名的方式来访问或修改对象的实例变量。例如如下代码：

```
# 对隐藏的__name 属性赋值  u._User__name = 'fk'  # 访问 User 对象的 name 属性（实际上访问__name 实例变量）  print(u.name)
```

上面粗体字代码实际上就是对 User 对象的 name 实例变量进行赋值，通过这种方式可“绕开”setname() 方法的检查逻辑，直接对 User 对象的 name 属性赋值。运行这两行代码，可以看到如下输出结果：

```
fk
```

总结

Python 并没有提供真正的隐藏机制，所以 Python 类定义的所有成员默认都是公开的；如果程序希望将 Python 类中的某些成员隐藏起来，那么只要让该成员的名字以双下画线开头即可。

即使通过这种机制实现了隐藏，其实也依然可以绕过去。

————————————————————

版权声明：本文为 CSDN 博主「无法撼动熬夜」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。
原文链接：https://blog.csdn.net/sadhsj/article/details/95048853