第1 0章 面向对象（上）

面向对象（Obj心Oriented）技术是软件工程领域中的重要技术，这种软件开发思想比较 自然地模拟了人类对客观世界的认识，成为当前计算机软件工程学的主流方法。本章就为大家 介绍面向对象的概念和使用方法。

10o 1瀰向斌蒙的微念 ’

在现实世界中存在各种不同形态的事物，这些事物之间存在着各种各样的联系。在程序中 使用对象来映射现实中的事物，使用对象间的关系来描述事物之间的联系，这种思想就是面向 对象。

提到面向对象，自然会想到面向过程。面向过程开发思想在早期的开发语言中被大量使用。 所谓过程就是一个没有返回值的函数。面向过程就是分析出解决问题的步骤，然后用函数把这 些步骤一一实现，使用的时候依次调用即可。面向过程只考虑在函数中封装代码逻辑，而不考 虑函数的归属关系。

面向对象把解决问题的事物提取为多个对象，而建立对象的目的也不是为了完成一个个步 骤，而是为了描述某个事物在解决整个问题的过程中所发生的行为。面向对象是一个更大的封 装，根据对象职责明确函数的归属关系。下面举一个五子棋的例子，说明面向过程和面向对象 编程的区别。

首先使用面向过程的方法：

（1） 开始游戏。

（2） 黑子先走。

（3） 绘制画面。

（4） 判断输赢。

（5） 轮到白子。

（6） 绘制画面。

（7 ）判断输赢。

（8）返回步骤（2）*。*

（9 ）输出最后结果。

上面的步骤分别用函数实现，问题就解决了，具体流程如图所示。

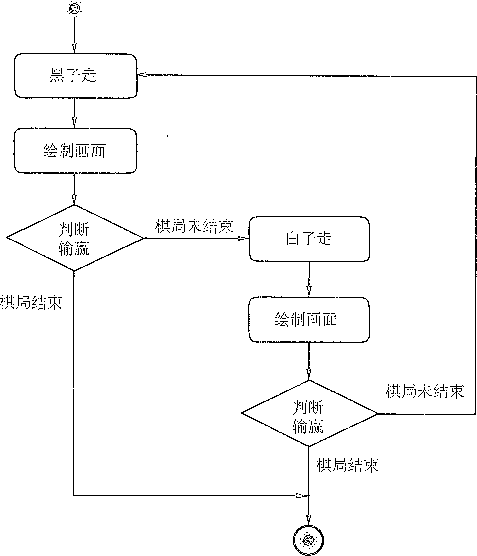


图10』五子棋的实现流程

面向对象的设计则是从另一种思路来解决问题。使用面向对象思想实现五子棋时，整个五 子棋可以分为三类对象，具体如下：

（1） 黑白双方：这两方的行为一样，负责决定落子位置。

（2） 棋盘系统：负责绘制画面。

（3） 规则系统:负责判断诸如犯规、输赢等。

上述三类对象中，第一类对象（黑白双方）负责接收用户的输入，并通知第二类对象（棋 盘系统）绘制棋盘上的棋子，同时第三类对象（规则系统）对棋局进行判定。

面向对象保证了功能的统一性，从而可以使代码更容易维护。例如，现在要加入悔棋的功 能，如果使用面向过程开发，那么输入、判断、显示的一系列步骤都需要改动冬甚至步骤之间 的循环都需要大规模的调整，这显然是非常麻烦的。如果使用面向对象开发，只需要改动棋盘 对象就可以了，棋盘对象保存了黑白双方的棋谱，只需要简单地回溯，而显示和规则不用变化， 同时整个对象功能的调用顺序也不会发生变化，它的改动只是局部的。由此可见，相比面向过程， 面向对象编程更方便后期代码的维护和功能的扩展。

10o2类和対象

所谓“人以类聚，物以群分”，类是具有相似特征和行为的事物的集合，即同一类型的事物。 比如猫、狗都属于动物类。类是对某类事物的抽象描述，而对象是现实中该类事物的一个个体。 例如，手机是一个类，而我的小米手机、小明的华为手机、小红的锤子手机都是现实中存在的 手机个体，都可以看作对象。下面举一个可爱的例子来描述类和对象的关系，具体如图162 所示。

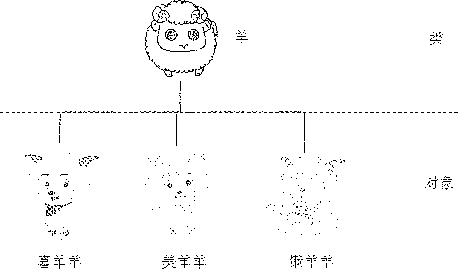


图0一2类与对象的关系

在图10-2中，可以把羊看作一个类，每个具体的羊看作一个对象，从而把羊类和具体的 羊之间的关系看作类与对象的关系。类用于描述多个对象的共同特征，它是对象的定义。对象 用于描述现实中的个体，它是类的具体实例。从图侦・2中可以看出，对象是根据类创建的，一 个类可以对应多个对象，但是一个对象只能对应一个类。

10«3类的定

1©d3J襲齣竈莫

类的定义格式如下：

class类名(父类):

#使用class定义类

#定义属性

#定义方法

属性名=属性值

de f方法名():

方法体

使用cl阳s关键字来定义类，class关键字后面是类的名称，如果类继承自某一个父类，则 将父类写在括号中。

在类的定义体中使用给属性赋值的方式定义属性，使用d©f关键字定义类的方法，类的方 法定义与函数类似。

以下代码定义了一个Dog类：

»> class Dog （Object） : # 类的定义.

legs = 4 #类的属性

• • . def description （self dog\_name） : # 类的方法

*…。* print （"我的名字是 %s° % dog\_name）

这段代码定义了一个名为Dog的类，并通过赋值的方法定义了一个属性，名为kgs；然后 使用def关键字定义了 一个方法,名*为*descriptiono

值得一提的是，在类中定义的方法带一个默认参数self，指向是对象本身，这点与函数的 定义格式有所不同。

定义了类之后，怎么使用它创建对象呢？ 一般创建对象的格式如下所示。

对象名=类名（）

可以看出，Pythg创建对象的方式与其他开发语言不同，不需要使用常用的IWW关键字膏 直接使用类名加括号就可以创建了。

现在我们使用上个小节定义的类Dog创建一个对象my\_dog9示例如下:

>>> my\_dog = Dog （） # 创建 my\_\_dog 对象

这段代孤创筐了一个Dog类的对象,.取名为my\_dog0

10.3.3 81養襲靈盈颱襲 ’.

在Python 2.x及以前的版本中，继承自任意内置类型的自定义类都称为“新式类”，能获 得所有新式类的特性；相反地，不继承自任何内置类型的自定义类，都称为“经典类” O

在Python 3以后，“新式类”和“经典类”的区别已经不存在了，在以后的版本里， 所有类型（包括内置类型和自定义类）都继承自内置类型object,即所有的类都是“新式 类”。object类是所有类的父类，如果在定义类型时没有指定父类，那么默认该类型继承自 object 类。

'10^4 属

属性用于描述事物的特征，比如颜色、大小、数量等。类的属性存储了类的各种数据，可 以通过点语法来访问。类的方法定义了类的行为特征膏包括这个类的各种操作。

10.4J襲屬膛

属性包括类属性和对象属性。其中类属性定义在类的内部、方法的外部，它由该类的所有 对象共同拥有。

类属性的值可以通过类和对象来访问，但是只能通过类来修改，包括直接使用类属性修改, 或者在类方法中修改。

比如上面定义的Dog类:

>>> class Dog(object):

..。 legs = 4

>>> one\_dog = Dog()

>>> another\_dog = Dog()

>>> another\_dog»legs = 3

»> print (another\_dog. legs)

3

»> print (one\_dog. legs)

4

>>> print(Dog^legs)

4

#定义一个Dog类

#定义类属性

#创建第一个对象on㊀\_dog

#创建第二个对象another\_dog

#通过对象访问类属性的值

#通过类访问类属性的值

上面的代码中，首先定义了一个Dog类，它包含一个类属性legs,并赋值为4。然后创 建了两个对象one\_\_dog和another\_dog9代码another\_dog.legs = 3修改了属性legs的值，通过 another\_dogJegs访问时值已经更改为3，但是通过one\_\_dog对象和Dog类访问legs属性时值都 是4,这说明了什么呢?让我们来分析一下:

1. 通过one\_dog对象和Dog类访问kgs属性值都是4,说明对象和类都可以访问类属 性的值，也说明类属性并没有被赋值语句another\_dogJegs = 3更改，即对象不能修改类属性的值。

(2 ) another dog.legs = 3的含义是为another\_dog对象添加一个新的对象属性,它覆盖了 同名的类属性,所以通过another\_dog对象访问legs属性返回值是3。

如果使用类名对类属性进行修改，结果会怎么样呢？让我们试着修改一下上面那个案例， 修改后的代码如下所示。

>>> class Dog(object):

。o。 legs = 4

>>> one\_dog = Dog()

>>> one\_d.og. legs

4

>>> Dog olegs

4

»> Dog. legs = 3 #通过类修改类属性的值

>>> Dog.legs

3

>>> one\_dog 8legs

3

在上述代码中，首先创建一个Dog对象one\_dog,然后通过类名来修改类属性kgs的值(Dog. legs = 3)o通过输出结果可以看出，使用类名成功地修改了类属性的值。

对象属性是描述对象特征的数据属性，对象属性的设置方法很灵活，可以在类定义的方法 里添加，也可以在调用实例的代码里添加。

(1)在方法中添加对象属性

如果在类定义的对象方法里添加对象属性， 则会引起错误。

对象属性只能由对象调用，不能由类调用。

»> class Employee (object):

... company ="传智播客"

.。“ def info(selff name7 id):

• self employee\_name = name

self o employee\_id = id

>>> tom = Employee()

»> tom. info ("Tom", n12806") >» print (tom. employee\_name) Tom

>>> print(tom.employee\_id)

12806

>>> print(Employee.company) 传智播客

>>> print(Employee.emp1oyee\_name) Traceback (most recent call last):

File °<stdin>", line 1r in <module>

这个属性只有在调用了该方法后才能使用，否 举例说明：

#定义类属性company

#定义对象属性employee\_name

#定义对象属性employee\_id

#调用了 info方法

#使用对象访问emp 1 oyee\_name属性

#使用对象访问employee\_id属性

#使用类访问company属性

#使用类访问employee\_name属性,出错

AttributeError: type object 1 Employee5 has no attribute , employee\_\_name?

在上述代码中，定义了一个Employee类，它有一个类属性company,还在info方法里定 义了 2个对象属性empioyee\_\_name和employee\_ido接下来定义了一个对象tom,并调用了 info 方法，调用了 info方法之后，就可以使用对象访问对象属性employeejiaiiie和employee\_id T o

接下来使用Employee类成功访问类属性company,但是访问对象属性employee\_name却 出错了，这说明须对象属性只能由对象访问。

如果我们再创建\_个Employee对象lily,不调用info方法,直接访问employee\_name属性, 会发生什么情况呢？我们来试一下：

»> lily = Employee ()

>>> lily.employee\_name

Traceback (most recent call last):

File "<stdin>°, line 1, in <module>

AttributeError: v Employee \* obj ect has no attribute \* employee\_name \*

从结果可以看出，如果不调用方法，则在方法里定义的对象属性都无法访问。而其他对象 创建的对象属性归创建它的对象所有，本对象也无法访问。

1. 在调用时添加对象属性

在调用对象时，也可以动态地给对象添加属性，例如：

>>> class Dog (object) : # 创建类

."。 legs = 4

>>> my\_dog *= Dog ()* # 创建对象

>>> my\_dog. color =，'红色" # 添加属性

>>> print (H我的颜色是%s° %my\_dog。color)

我的颜色是红色

上述代码中，Dog定义时并没有co成属性，这个属性是在调用时添加的。而且color属性 只归my\_dog对象所有,其他对象无法访问。

第\_次使用my\_dog.color= °红色°表示给这个对象添加了一个名为color•的属性,就可以 通过my\_dogxolor访问这个属性了。

第二次使用my\_dog.colo『=XXX表示对属性进行修改。

1. 删除对象属性

除此之外，对象属性还可以删除，这也是Pythoii特有的语法。例如，在上例中，尝试着 删除tom对象的employee id属性，代码如下所示。

>>> del xiaoMing.empID

>>> print(xiaoMing.empID)

Traceback (most recent call last):

File "<stdin>"7 line 1z in <module>

AttributeError: ? Employee! obj ect has no attribute ? empID1

这里使用dei关键字删除了 tom对象的RmployRR\_id属性，再尝试访问tom。employee.id时， 解释器发出了异常，提示该对象已经没有employee\_id属性了。

〈疗注意： 「

在运行时添加和删除对象属性,是Python独有的特性之一。习惯了 Java和C++等语言的 人往往会觉得不可思议，因为Java和C++中的所有属性都必须在类中声明之后才能使用。

Python语言动态添加和删除对象属性的特性为我们的程序开发提供了很大的灵活性，但是 也带来了维护上的麻烦，因为如果一个对象属性不存在，在使用时就会出错。

iO.4.3陶遭着懑粕新洶有瀆 「

在类的方法中，有两种特殊的方法，分别在类创建和销毁的时候调用，它们是构造方法— init\_()和析构方法\_dd\_()。接下来,对这两个方法分别进行介绍。

(1) \_init\_()方法

创建对象时,我们使用“类名()”就可以实现。实际上，每次创建对象之后，系统会马上 调用\_init\_\_()方法，这个方法就是类的构造方法。

每个类都有一个默认的\_init\_\_()方法。如果在自定义类时显式定义了 \_init\_()方法,则 创建完对象后会调用这个定义的\_init\_()方法；如果在自定义类中没有定义\_init\_()方法， 那么会使用默认的\_jmit\_()方法，即创建对象后什么都不做，直接返回对象。

\_init\_()方法的第一个参数关If是对象本身，在创建类时传入的参数实际上都传递给了 \_init\_()o我们可以使用\_init\_()做一些准备工作，另外，\_init\_\_()方法是添加对象属性的重要场所。

下面的例子在\_init\_\_()方法中为Student的对象添加了两个对象属性，如下所示。

»>

class Student(object):

def init (selfz name, student\_id):

#构造方法

#添加对象属性name

#添加对象属性student\_id

self.name = name

self o student id = student id

»>

j erry = Student(°Jerry"f "201718016") print(jerry o name)

»>

Jerry

>>> print ( ° 学生 1 %s 1 的学号是 \* %s 1 "% (jerry .name,j erry.student\_id)) 学生1 Jerry '的学号是\* 201718016 5

这段代码为Student类添加了 \_init\_()方法，该方法有3个参数,第1个参数self是对象本身， 第2个参数name和第3个参数student\_id是传入的参数。创建对象时，将name和student\_id 参数传入，系统调用\_顽\_()方法，为对象添加了 2个对象属性。

(2 ) \_del\_()方法

在创建对象时，系统自动调用—init\_\_()方法，在对象被清理时，系统也会自动调用一个 \_del\_()方法，这个方法可谓是类的析构方法。

使用del语句可以删除对象的一个引用。现在用一个例子说明\_\_dd\_()方法的调用「代码 如下：

»>

class Person(object):

def init (self): print (n对象创建了

def del (self): print (°对象被清除了 n)

构造方法

析构方法

»>

对象创建了

>>> del person\_one

对象被清除了

person\_one = Person()

创建对象?并由person\_one引用

清除 person\_one引用

这段代码首先创建了一个Person对象，由变量person\_one引用。然后使用del语句清除该 引用。由输出信息可知，系统自动调用了 \_dd\_()方法，将对象清除掉。

需要注意的是：

* \_del\_()方法只有一个self参数。

•如果有父类，则在\_dE\_()方法体内要先调用父类的\_如\_\_()方法。

* \_del\_()方法用于在自定义类被清除之前做一些清理工作,一般不用实现\_del\_()方法。

10d4d4

对象方法是在类中定义的、以关键字self作为第一个参数的方法。对象方法的self参数代表调用这个方法的对象本身(在其他语言中可能使用this) o在方法调用时，可以不用传递这 个参数，系统会自动将调用方法的对象作为self参数传入。

在对象方法里可以使用self关键字定义和访问对象属性，同名的对象属性会覆盖类属性。 对于对象方法，我们并不陌生，在之前的案例中已经接触过。下面的代码定义了一个类

Employee,它有一个对象方法 work\_for\_one\_year() °

class Employee(object):

»>

定义类属性 定义对象方法 定义同名对象属性

level = 3

def work\_for\_one\_year(self):

self.level = 4

»>

»>

3

»>

tom = Employee() print(tom.level)

创建对象 访问类属性

»>

4

tom. work\_for\_one\_\_year () print(tom.level)

调用对象方法

访问对象属性

定义work\_fOT\_oiie\_y河()方法时，self作为默认参数，并在方法中使用selfJevel定义了一 个对象属性，这个对象属性与类属性同名。

创建Employee对象，在访问work\_for\_one\_year()方法之前,使用tom对象访问的level属 性值为3,说明访问的是类属性；在调用对象方法之后，再使用tom对象访问的level属性值为4, 说明此时访问的是对象方法中定义的同名对象属性，也就是说，在方法中定义的对象属性覆盖 了同名类属性。

顾名思义彝对象方法是属于对象的方法，对象肯定是可以调用的，那么类可以调用对象方 法吗？我们来实验一下。

class Exampleclass(object):

»>

def object\_method(self):

#定义对象方法

print ("——对象方法 print(self)

>>> example = Exampleclass()

#创建对象

#对象方法：对象可以调用

>» example . obj ect\_method () -—对象方法

< main .Exampleclass object at 0x00331E70>

»> Exampleclass . ob j ect\_method () #对象方法：类不能直接调用

Traceback (most recent call last):

File "<stdin>n z line 1, in <module>

TypeError: object\_method() missing 1 required positional argument:

'self1

从结果可以看出，对象可以直接调用对象方法，但是类不可以直接调用对象方法。如果类 调用的时候传入参数会怎么样？我们再测试一下。

»> Exampleclass•objectMethod(Exampleclass)

对象方法

<class ' main .Exampleclass \*>

>>> Exampleclass.objectMethod(example)

---对象方法

< main .Exampleclass object at 0x00331E70>

»>

从运行结果可知，类也可以调用对象方法，只需要在调用的时候传入类或者该类的对象即 可。由此可见，Pythoii是一个非常灵活的语言。

10.4.5爨有灌 .

类方法是使用修饰器@classmethod>W的方法，它的第一个参数代表类本身，按照惯例， 以cis作为第一个参数。在调用时，不需要传递这个参数，系统会自动将调用它的类当作参数 传入。

类方法还有一个用途就是对类属性进行修改。我们知道，在对象方法中无法给类属性赋值, 这样的语句会被视为定义了同名的对象属性。下面的代码分别使用对象方法和类方法试图修改 类属性。

|  |  |
| --- | --- |
| >>> class Apple(object): | #定义Apple类 |
| 。.o count = 0 | #定义类属性 |
| def add\_one (self): |  |
| .。。 self.count = 1 | #对象方法 |
| o。. Qclassmethod |  |
| ... def add\_two(cis): |  |
| .。。 cis。count = 2 | #类方法 |

>>> apple = Apple()

>>> apple o add\_one()

>>> print(Apple.count) 0

»> Apple . add\_two () >>> print(Apple.count) 2

从结果可以看出，调用了对象方法&dd\_one()后，类属性count的值没有变，因为对象方法 中的代码self.count = 1实际上创建了一个同名的对象属性，并没有修改类属性的值。而调用了 add\_two()类方法后，类属性count的值就被修改为2。

那么，谁可以调用类方法呢？我们说过，Pythoii是一个很灵活的语言，虽然叫作类方法， 但是对象和类都可以调用，当对象调用类方法时，系统自动将对象所属的类作为第一个默认参 数传入。我们使用一个例子测试一下。

class Exampleclass(object): @classmethod

»>

定义类方法

def class\_method(cis): print (类方法 ")

print(cis)

»>

example = Exampleclass() example。class\_method()

»>

类方法一-

<class ? main .Exampleclass \*> >>> Exampleclass.class\_method() --类方法 -

<class ' main .Exampleclass 1>

创建对象

类方法:对象可以调用

类方法:类可以调用

从这个例子可以看岀，对象和类都可以调用类方法，而且从打印结果可知，传入的第一个 参数都是类。

10.4.S靜窓君淒

静态方法是使用修饰器@staticmethod进行修饰的方法，它不需要传入默认参数，所以与 类并没有很强的联系。由于没有传入默认参数，所以在静态方法中不能直接访问类的属性和方 法，需要通过类名.方法名或属性名的方式来访问。

那么对象和类可以调用静态方法吗？我们进行一个测试：

>>> class Exampleclass(object):

• . . @staticmethod

.o" def static\_method():

#定义静态方法

#创建对象

#静态方法:对象可以调用

#静态方法:类可以调用

print (° 静态方法 ")

>>> example = Exampleclass()

>>> example.static\_method()

—静态方法

>>> Exampleclass.static\_method()

静态方法

从测试结果可以看出，类和方法都可以调用静态方法。

至此，我们学习了对象方法、类方法和静态方法。这里对它们的特征进行归纳，具体如 表10-1所示。

110^1实例方法、类方法粕静态方法的特征

|  |  |  |  |
| --- | --- | --- | --- |
| 方 法 | 第一个参数 | 修饰器 | 类和对象访问 |
| 实例方法 | self | *无* | 对象可以直接访问  类需要传入参数 |
| 类方法 | cis | @classmethod | 都可以直接访问 |
| 静态方法 | 无 | @staticmethod | 都可以直接访问 |

理论上，对象方法应该是只有对象才能调用，类方法应该只有类才能调用，但是Python 是一种很灵活的语言，不管是类方法、对象方法和静态方法，类和对象都可以调用。作为一名 开发人员，我们在享受便捷的同时，也要对这些方法的调用进行仔细分辨，避免因混淆而引起 错误。

分脚下留伯：类方法和静态方法的区别

类方法和静态方法最主要的区别在于类方法有一个cis参数，而静态方法没有，所以类方 法可以根据传入的cis参数访问类的数据，而静态方法不可以。所以静态方法更适合进行与类 无关的操作，不需要在方法内使用类的数据。比如一些工具类，只需要根据传入的参数进行操 作即可。 .

变量名称前加了 —的类属性是私有属性，加了 的方法是私有方法。在上面的代

码里，使用类私有属性的名字Employ咬•—company访问出错，但是使用命名混淆后的名字 Employee. Employee company访问就成功了；同理,使用私有方法的名字echo. info()访问 出错，但是使用命名混淆后的名字echo.\_Employee—info()访问就成功了。

命名混淆技术的主要目的是避免名字冲突，尤其是在子类中，如果子类定义了与父类同名 的属性或方法，那么子类会将父类的同名成员覆盖掉。但是，如果使用了 \_memberName?就 不用担心子类成员覆盖父类成员的情况了。

除此之外，私有机制还有另一个场合，那就是避免本类的成员名称与引用的其他模块的成 员名称相同而引起的命名冲突。

［方注意：

Python使用下画线做名称的前缀和后缀来指定特殊变量和方法，其中：

® memberName 表示系统内建的名称。

• \_memberName表示类中的私有变量名。

除此之外,还可以使用一个下画线(\_memberName)来为类成员命名? 一个下画线其实跟 公有成员一样，从外部可以直接访问，但是如果开发者能够约定将—monberNamR视为私有， 不在类的外部调用，则对实际开发和测试是非常方便的。

在开发中经常会遇到，不希望对象的私有属性被外界随意访问和修改的情况。我们希望外 界对属性的操作在我们的控制范围之内，此时一般的处理方式是：

220 2 旳艦◎廃SB®!!：關翳掌Pjthoo

-将属性定义为私有属性。

•添加一个公有方法，用于属性的操作。

现在我们将类Gill修改一下?使用方法来访问它的私有属性\_ageo

»>

class Girl (object):

init (selff age): selt \_\_age = age get\_age(self): is\_stranger = False if is\_stranger:

return 0

else :

return self o age set\_age(selff age): if 110 > age > 0:

self. age = age

else:

print (°年龄应该大于0小于110.°)

def

def

def

»>

»>

»>

18

lily = Girl(25) lilyset\_age (18)

lily.get\_age()

#定义类Girl

#定义构造函数

#定义私有属性 age

#定义方法读取 \_age的值

#定义方法设置 age的值

#创建一个对象丄Uy

#使用方法设置lily的年龄

#使用方法读取lily的年龄

在上述代码中，定义了 get\_age()方法来获取对象的 务ge属性的值，由于是在方法中访问 的\_age属性，因此可以根据一定的条件(如果有的话)进行判断，检测是否要输出\_age属 性的值，这样就实现了访问控制。又定义了 set\_age()方法设置\_\_age属性的值，在方法中可以 对传入的参数进行判断，只有大于0且小于110的参数才被视为有效年龄给\_\_age属性赋值， 这样能确保对象数据属性的合理性*。*

1©Q4o8靂邇灑ii谛词離槍®蹺一儷屬©property

在上一小节中，我们使用了 set和get方法来设置和访问私有属性，但还能不能改进呢？ 有没有一种实现方式，既能检查参数，又能像直接访问属性那样便捷呢？

答案是肯定的。使用@property装饰器就可以实现。

@p『opRity装饰器可以把方法变成属性调用，把一个get方法变成属性，只需要为它加上 ©property装饰器即可，©property还创建了另一个装饰器@XXX.setter9负责把set方法也变 成属性的赋值方法。现在我们将Girl类的私有对象属性用@property装饰器进行重写，代码 如下：

>>> class Girl(object):

#定义类Girl

#构造方法

#定义私有对象属性 age

o *。*o def init (self):

.<»。 self. age = 0

»>

self. weight = 50 ©property

def age(self):

return self„ age @age。setter

def age(self, value):

if 110 > value > 0:

#定义私有对象属性 weight

#使用^property装饰器将get方法变成属性

#将set方法变成属性

self. age = value

else :

raise ValueError (°年龄应该大于0小于110 .") @property

def weight (self) : # weight 属4■生的 get 方法

return self. weight

重写实现后的Girl类使用了 ©property装饰器将 age属性的get方法和set方法都变成了 age属性，将\_weight属性的get方法也变成了 weight属性。现在可以使用访问属性的方法访 问age属性和weighto

>>> lily = Girl() »> lily.age = 25 >>> lily o age = 120 Traceback (most recent File n<stdin>n, line File °<stdin>"r line

#创建对象

#使用属性访问的方式设置\_\_玛㊀属性的值

#当设置属性的值超出范围时报出异常 call last):

1r in <module>

13 f in age

ValueError : 年龄应该大于0小于110 s

»>

25

lily。age

#使用属性访问的方式获取\_\_age属性的值

»>

50

lily.weight

#使用属性访问的方式获取 weight属性的值

#设置 weight属性的值时报错

lily.weight = 55

Traceback (most recent call last):

File n<stdin>"f line 1, in <module>

AttributeError: can't set attribute

代码lilyage=25相当于调用了 set方法lily.age(25)o如果设置的age参数的值不在检查范围 内会报出异常。

代码lilyage相当于调用了 gd方法，使用这种方式，用属性访问的方式实现了检查参数的 功能，受到了程序员的欢迎。

需要注意的是，—weight属性只实现了 get方法，没有实现汩方法，因而是一个只读属性, 因此在试图使用iily.weight = 55为它赋值时，程序会报错。

仃注意

可以使用@property装饰器为私有对象属性添加删除方法,比如给Girl类的 age属性添

222編：Pythoo篡曬鑑圖 關蜀学Python 加删除方法：

@age.deleter

def age(self):

del self. age

此时可以通过del语句将属性删除：

del lily.age