Python面向对象编程

面向对象编程和函数式编程（面向过程编程）都是程序设计的方法，不过稍有区别。

面向过程编程：

```
1. 导入各种外部库
2. 设计各种全局变量
3. 写一个函数完成某个功能
4. 写一个函数完成某个功能
5. 写一个函数完成某个功能
6. 写一个函数完成某个功能
7. 写一个函数完成某个功能
8. ......
9. 写一个main函数作为程序入口
```

在多函数程序中，许多重要的数据被放置在全局数据区，这样它们可以被所有的函数访问。每个函数都可以具有它们自己的局部数据，将某些功能代码封装到函数中，日后便无需重复编写，仅调用函数即可。从代码的组织形式来看就是根据业务逻辑从上到下垒代码 。

面向对象编程：

```
1. 导入各种外部库
2. 设计各种全局变量
3. 决定你要的类
4. 给每个类提供完整的一组操作
5. 明确地使用继承来表现不同类之间的共同点
6. 根据需要，决定是否写一个main函数作为程序入口
```

面向对象编程中，将函数和变量进一步封装成类，类才是程序的基本元素，它将数据和操作紧密地连结在一起，并保护数据不会被外界的函数意外地改变。类和和类的实例（也称对象）是面向对象的核心概念，是和面向过程编程、函数式编程的根本区别。

并不是非要用面向对象编程，要看你的程序怎么设计方便，但是就目前来说，基本上都是在使用面向对象编程。

类的基本用法

面向对象是通过定义class类来定义，这么说面向对象编程就是只使用class类，在class类中有封装，继承的功能，并且还可以构造要传入的参数，方便控制。

案例一

```
import sys
import time
reload(sys)
sys.setdefaultencoding('utf-8')
```

```
class studetn:
    # 定义一个类名为studetn
    def __init__(self,idx):
    # 定义初始化构造，这里使用init，还有别的属性比如reversed，iter之类的
        self.idx=idx
        # 初始化变量，方便继承
    def runx(self):
    # 定义运行函数，从上面继承变量
        print self.idx
        # 打印出idx的值，或者做一些别的处理
        time.sleep(1)
a=studetn('a')
a.runx()
# 这是类的调用，一定要记得类的使用方法，首先传入参数，类赋值给一个变量a
# 然后调用这个类下面定义的函数
```

一些专业术语概念，既然有面向对象编程这个高大上的定义了，自然要搭配一些高大上的概念。

1. 类(Class): 用来描述具有相同属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。其中的对象被称作类的实例。
2. 实例：也称对象。通过类定义的初始化方法，赋予具体的值，成为一个”有血有肉的实体”。
3. 实例化：创建类的实例的过程或操作。
4. 实例变量：定义在实例中的变量，只作用于当前实例。
5. 类变量：类变量是所有实例公有的变量。类变量定义在类中，但在方法体之外。
6. 数据成员：类变量、实例变量、方法、类方法、静态方法和属性等的统称。
7. 方法：类中定义的函数。
8. 静态方法：不需要实例化就可以由类执行的方法
9. 类方法：类方法是将类本身作为对象进行操作的方法。
10. 方法重写：如果从父类继承的方法不能满足子类的需求，可以对父类的方法进行改写，这个过程也称override。
11. 封装：将内部实现包裹起来，对外透明，提供api接口进行调用的机制
12. 继承：即一个派生类（derived class）继承父类（base class）的变量和方法。
13. 多态：根据对象类型的不同以不同的方式进行处理。

类与实例

```
# -*- coding: utf-8 -*-
# @Time    : 2018/5/3 0003 17:02
# @Author  : Langzi
# @Blog    : www.langzi.fun
# @File    : 面向对象2.py
# @Software: PyCharm
import sys
import time
import requests
reload(sys)
sys.setdefaultencoding('utf-8')
```

```
class cc:
    ccc = 'ccc'
    # cc就是类名 如果想要继承别的类 就class cc(threading) 意思就是从threading继承
    def __init__(self,a,b,c):
        self.a=a
        self.b=b
        self.c=c
        # 定义构造的过程就是实例化
    def runx(self):
        print self.a*10
        print self.b*5
        print self.c*2
    def runy(self):
        print requests.get('http://www.langzi.fun').headers
e = cc('AAA','CCC','EEE')
e.runx()
e.runy()
# 这两个就是调用类里面的方法
print e.c
#实例变量指的是实例本身拥有的变量。每个实例的变量在内存中都不一样。
print e.ccc
#类变量，在类里面找到定义的变量。
```

调用类的三种方法

实例方法

```
# -*- coding: utf-8 -*-
# @Time    : 2018/5/3 0003 17:16
# @Author  : Langzi
# @Blog    : www.langzi.fun
# @File    : 面向对象3.py
# @Software: PyCharm
import sys
import time
import requests
reload(sys)
sys.setdefaultencoding('utf-8')

class dd:
    def __init__(self,url):
        self.url=url
    def runx(self):
        print requests.get(self.url).status_code

a = dd('http://www.langzi.fun')
a.runx()
# 这种调用方法就是实例方法
```

静态方法

静态方法由类调用，无默认参数。将实例方法参数中的self去掉，然后在方法定义上方加上@staticmethod，就成为静态方法。它属于类，和实例无关。建议只使用类名.静态方法的调用方式。（虽然也可以使用实例名.静态方法的方式调用）

```
# -*- coding: utf-8 -*-
# @Time    : 2018/5/3 0003 17:21
# @Author  : Langzi
# @Blog    : www.langzi.fun
# @File    : 面向对象4.py
# @Software: PyCharm
import sys
import requests
reload(sys)
sys.setdefaultencoding('utf-8')
class ff:
    @staticmethod
    def runx():
        print requests.get('http://www.langzi.fun').status_code
ff.runx()
#这里就直接调用了类的变量，只在类中运行而不在实例中运行的方法
```

经常有一些跟类有关系的功能但在运行时又不需要实例和类参与的情况下需要用到静态方法. 比如更改环境变量或者修改其他类的属性等能用到静态方法. 这种情况可以直接用函数解决, 但这样同样会扩散类内部的代码，造成维护困难。

类方法

类方法由类调用，采用@classmethod装饰，至少传入一个cls（代指类本身，类似self）参数。执行类方法时，自动将调用该方法的类赋值给cls。建议只使用类名.类方法的调用方式。（虽然也可以使用实例名.类方法的方式调用）

实际案例

如果要构造一个类，接受一个网站和这个网站的状态码，然后打印出来。就像这样：

```
import sys
import requests
reload(sys)
sys.setdefaultencoding('utf-8')
class gg:
    def __init__(self,url,stat):
        self.url=url
        self.stat=stat
    def outer(self):
        print self.url
        print self.stat
a = gg('langzi',200)
a.outer()
```

这样就是使用实例方法，虽然可以实现，但是有的时候传入的参数并不是('langzi',200)这样的格式，而是('langzi-200')这样的，那该怎么做？首先要把这个拆分，但是要使用实例方法实现起来很麻烦，这个

时候就可以使用类方法。

```
# -*- coding: utf-8 -*-
# @Time    : 2018/5/3 0003 17:27
# @Author  : Langzi
# @Blog    : www.langzi.fun
# @File    : 面向对象5.py
# @Software: PyCharm
import sys
import requests
reload(sys)
sys.setdefaultencoding('utf-8')
class gg:
    url = 0
    stat = 0
    # 因为使用classmethod后会传入新的变量，所以一开始是需要自己先定义类变量
    def __init__(self,url=0,stat=0):
    # 这里按照正常的定义构造函数
        self.url=url
        self.stat=stat
    @classmethod
    # 装饰器，立马执行下面的函数
    def split(cls,info):
        # 这个函数接受两个参数，默认的cls就是这个类的init函数，info就是外面传入进来的
        url,stat=map(str,info.split('-'))
        # 这里转换成了格式化的结构
        data = cls(url,stat)
        # 然后执行这个类第一个方法，这个类构造函数需要传入两个参数，于是就传入了两个参数
        return data
        # 这里就直接返回了函数结果
    def outer(self):
        print self.url
        print self.stat

r = gg.split(('langzi-200'))
r.outer()
# 这里是调用类方法，与调用实例方法一样
```

类的特性

封装

封装是指将数据与具体操作的实现代码放在某个对象内部，外部无法访问。必须要先调用类的方法才能启动。

案例

```
class cc:
    ccc = 'ccc'
    # cc就是类名 如果想要继承别的类 就class cc(threading) 意思就是从threading继承
    def __init__(self,a,b,c):
        self.a=a
```

```
        self.b=b
        self.c=c
print e.ccc
#类变量，在类里面找到定义的变量。
print ccc
# 这里会报错，这就是封装。类中的函数同理。
```

继承

当我们定义一个class的时候，可以从某个现有的class继承，新的class称为子类（Subclass），而被继承的class称为基类、父类或超类（Base class、Super class）。
比如，我们已经编写了一个名为Animal的class，有一个run()方法可以直接打印：

```
class Animal(object):
    def run(self):
        print 'Animal is running...'
```

当我们需要编写Dog和Cat类时，就可以直接从Animal类继承：

```
class Dog(Animal):
    pass
class Cat(Animal):
    pass
```

继承有什么好处？最大的好处是子类获得了父类的全部功能。由于Animial实现了run()方法，因此，Dog和Cat作为它的子类，什么事也没干，就自动拥有了run()方法：

```
dog = Dog()
dog.run()
cat = Cat()
cat.run()
```

当子类和父类都存在相同的run()方法时，我们说，子类的run()覆盖了父类的run()，在代码运行的时候，总是会调用子类的run()。这样，我们就获得了继承的另一个好处：多态。

多态

要理解多态的好处，我们还需要再编写一个函数，这个函数接受一个Animal类型的变量：

```
def run_twice(animal):
    animal.run()
    animal.run()
```

当我们传入Animal的实例时，run_twice()就打印出：

```
run_twice(Animal())
运行结果:
```

```
Animal is running...
Animal is running...
```

当我们传入Dog的实例时，run_twice()就打印出：

```
run_twice(Dog())
运行结果：
Dog is running...
Dog is running...
```

当我们传入Cat的实例时，run_twice()就打印出：

```
run_twice(Cat())
运行结果：
Cat is running...
Cat is running...
```

看上去没啥意思，但是仔细想想，现在，如果我们再定义一个Tortoise类型，也从Animal派生：

```
class Tortoise(Animal):
    def run(self):
        print 'Tortoise is running slowly...'
```

当我们调用run_twice()时，传入Tortoise的实例：

```
run_twice(Tortoise())
运行结果：
Tortoise is running slowly...
Tortoise is running slowly...
```

你会发现，新增一个Animal的子类，不必对run_twice()做任何修改，实际上，任何依赖Animal作为参数的函数或者方法都可以不加修改地正常运行，原因就在于多态。

多态的好处就是，当我们需要传入Dog、Cat、Tortoise……时，我们只需要接收Animal类型就可以了，因为Dog、Cat、Tortoise……都是Animal类型，然后，按照Animal类型进行操作即可。由于Animal类型有run()方法，因此，传入的任意类型，只要是Animal类或者子类，就会自动调用实际类型的run()方法，这就是多态的意思：

对于一个变量，我们只需要知道它是Animal类型，无需确切地知道它的子类型，就可以放心地调用run()方法，而具体调用的run()方法是作用在Animal、Dog、Cat还是Tortoise对象上，由运行时该对象的确切类型决定，这就是多态真正的威力：调用方只管调用，不管细节，而当我们新增一种Animal的子类时，只要确保run()方法编写正确，不用管原来的代码是如何调用的。这就是著名的“开闭”原则：

对扩展开放：允许新增Animal子类；
对修改封闭：不需要修改依赖Animal类型的run_twice()等函数。
总结：继承可以把父类的所有功能都直接拿过来，这样就不必重零做起，子类只需要新增自己特有的方法，也可以把父类不适合的方法覆盖重写；
有了继承，才能有多态。在调用类实例方法的时候，尽量把变量视作父类类型，这样，所有子类类型都可以正常被接收；

旧的方式定义Python类允许不从object类继承，但这种编程方式已经严重不推荐使用。任何时候，如果没有合适的类可以继承，就继承自object类。

魔法方法

在上面有提到除了**init**之外还有iter,reverse的方法，这里就详细说下除了init初始化还有哪些别的方法。

```
__init__ :       构造函数，在生成对象时调用
__del__ :        析构函数，释放对象时使用
__repr__ :       打印，转换
__setitem__ :    按照索引赋值
__getitem__:     按照索引获取值
__len__:         获得长度
__cmp__:         比较运算
__call__:        调用
__add__:         加运算
__sub__:         减运算
__mul__:         乘运算
__div__:         除运算
__mod__:         求余运算
__pow__:         幂
```

具体使用

1. doc

说明性文档和信息。Python自建，无需自定义。

```
class Foo:
    """ 描述类信息，可被自动收集 """
    def func(self):
        pass
# 打印类的说明文档
print(Foo.__doc__)
```

2. init()

实例化方法，通过类创建实例时，自动触发执行。

```
class Foo:
    def __init__(self, name):
        self.name = name
        self.age = 18
obj = Foo(jack') # 自动执行类中的 __init__ 方法
```

3. module__ 和 __class

module 表示当前操作的对象在属于哪个模块。
class 表示当前操作的对象属于哪个类。

这两者也是Python内建，无需自定义。

```
class Foo:
    pass
obj = Foo()
print(obj.__module__)
print(obj.__class__)
运行结果:
main
```

4. del()

析构方法，当对象在内存中被释放时，自动触发此方法。

注：此方法一般无须自定义，因为Python自带内存分配和释放机制，除非你需要在释放的时候指定做一些动作。析构函数的调用是由解释器在进行垃圾回收时自动触发执行的。

```
class Foo:
    def __del__(self):
        print("我被回收了！")

obj = Foo()
del obj
```

5. call()

如果为一个类编写了该方法，那么在该类的实例后面加括号，可会调用这个方法。

注：构造方法的执行是由类加括号执行的，即：对象 = 类名()，而对于call() 方法，是由对象后加括号触发的，即：对象() 或者 类()()

```
class Foo:
    def __init__(self):
        pass
    def __call__(self, *args, **kwargs):
        print('__call__')
obj = Foo()     # 执行 __init__
obj()       # 执行 __call__
```

可以用Python内建的callable()函数进行测试，判断一个对象是否可以被执行。

```
callable(Student())
```

运行结果：

```
True
```

6. dict

列出类或对象中的所有成员！非常重要和有用的一个属性，Python自建，无需用户自己定义。

```
class Province:
    country = 'China'
    def __init__(self, name, count):
        self.name = name
        self.count = count
    def func(self, *args, **kwargs):
        print（'func'）
# 获取类的成员
print(Province.__dict__)
# 获取 对象obj1 的成员
obj1 = Province('HeBei',10000)
print(obj1.__dict__)
# 获取 对象obj2 的成员
obj2 = Province('HeNan', 3888)
print(obj2.__dict__)
```

7. str()

如果一个类中定义了str()方法，那么在打印对象时，默认输出该方法的返回值。这也是一个非常重要的方法，需要用户自己定义。

下面的类，没有定义str()方法，打印结果是：

```
class Foo:
    pass
obj = Foo()
print(obj)
定义了__str__()方法后，打印结果是：'jack'。
class Foo:
    def __str__(self):
        return 'jack'
obj = Foo()
print(obj)
```

8、getitem__()、*_setitem_*()、__delitem()

取值、赋值、删除这“三剑客”的套路，在Python中，我们已经见过很多次了，比如前面的@property装饰器。

Python中，标识符后面加圆括号，通常代表执行或调用方法的意思。而在标识符后面加中括号[]，通常代表取值的意思。Python设计了getitem()、setitem()、delitem()这三个特殊成员，用于执行与中括号有关的动作。它们分别表示取值、赋值、删除数据。

也就是如下的操作：

```
a = 标识符[]  :      执行__getitem__方法
标识符[] = a  :      执行__setitem__方法
del 标识符[]  :      执行__delitem__方法
```

如果有一个类同时定义了这三个魔法方法，那么这个类的实例的行为看起来就像一个字典一样，如下例所示：

```
class Foo:
    def __getitem__(self, key):
        print('__getitem__',key)
    def __setitem__(self, key, value):
        print('__setitem__',key,value)
    def __delitem__(self, key):
        print('__delitem__',key)
obj = Foo()
result = obj['k1']      # 自动触发执行 __getitem__
obj['k2'] = 'jack'      # 自动触发执行 __setitem__
del obj['k1']             # 自动触发执行 __delitem__
```

9. iter()

这是迭代器方法！列表、字典、元组之所以可以进行for循环，是因为其内部定义了 iter()这个方法。如果用户想让自定义的类的对象可以被迭代，那么就需要在类中定义这个方法，并且让该方法的返回值是一个可迭代的对象。当在代码中利用for循环遍历对象时，就会调用类的这个iter()方法。

普通的类：

```
class Foo:
    pass
obj = Foo()
for i in obj:
    print(i)
# 报错：TypeError: 'Foo' object is not iterable<br># 原因是Foo对象不可迭代
添加一个__iter__()，但什么都不返回：
class Foo:
    def __iter__(self):
        pass
obj = Foo()
for i in obj:
    print(i)
# 报错：TypeError: iter() returned non-iterator of type 'NoneType'
#原因是 __iter__方法没有返回一个可迭代的对象
```

返回一个个迭代对象：

```
class Foo:
    def __init__(self, sq):
        self.sq = sq
    def __iter__(self):
        return iter(self.sq)
obj = Foo([11,22,33,44])
for i in obj:
    print(i)
```

最好的方法是使用生成器：

```
class Foo:
    def __init__(self):
        pass
    def __iter__(self):
        yield 1
        yield 2
        yield 3
obj = Foo()
for i in obj:
    print(i)
```

10、len()

在Python中，如果你调用内置的len()函数试图获取一个对象的长度，在后台，其实是去调用该对象的len()方法，所以，下面的代码是等价的：

```
len('ABC')
3
'ABC'.__len__()
3
```

Python的list、dict、str等内置数据类型都实现了该方法，但是你自定义的类要实现len方法需要好好设计。

11. repr()

这个方法的作用和str()很像，两者的区别是str()返回用户看到的字符串，而repr()返回程序开发者看到的字符串，也就是说，repr()是为调试服务的。通常两者代码一样。

```
class Foo:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return "this is %s" % self.name
    __repr__ = __str__
```

12. add__: 加运算 _sub_: 减运算 _mul_: 乘运算 _div_: 除运算 _mod_: 求余运算 __pow: 幂运算

这些都是算术运算方法，需要你自己为类设计具体运算代码。有些Python内置数据类型，比如int就带有这些方法。Python支持运算符的重载，也就是重写。

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b
   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)
   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)
v1 = Vector(2,10)
```

```
v2 = Vector(5,-2)
print (v1 + v2)
```

13. author作者信息

```
__author__ = "Jack"
def show():
    print(__author__)
show()
```

14. slots

Python作为一种动态语言，可以在类定义完成和实例化后，给类或者对象继续添加随意个数或者任意类型的变量或方法，这是动态语言的特性。例如：

```
def print_doc(self):
    print("haha")

class Foo:
    pass

obj1 = Foo()
obj2 = Foo()
# 动态添加实例变量
obj1.name = "jack"
obj2.age = 18
# 动态的给类添加实例方法
Foo.show = print_doc
obj1.show()
obj2.show()
```

但是！如果我想限制实例可以添加的变量怎么办？可以使slots限制实例的变量，比如，只允许Foo的实例添加name和age属性。

```
def print_doc(self):
    print("haha")
class Foo:
    __slots__ = ("name", "age")
    pass
obj1 = Foo()
obj2 = Foo()
# 动态添加实例变量
obj1.name = "jack"
obj2.age = 18
obj1.sex = "male"       # 这一句会弹出错误
# 但是无法限制给类添加方法
Foo.show = print_doc
obj1.show()
obj2.show()
由于'sex'不在__slots__的列表中，所以不能绑定sex属性，试图绑定sex将得到AttributeError的错误。
Traceback (most recent call last):
  File "F:/Python/pycharm/201705/1.py", line 14, in <module>
```

```
    obj1.sex = "male"
AttributeError: 'Foo' object has no attribute 'sex'
```

需要提醒的是，slots定义的属性仅对当前类的实例起作用，对继承了它的子类是不起作用的。想想也是这个道理，如果你继承一个父类，却莫名其妙发现有些变量无法定义，那不是大问题么？如果非要子类也被限制，除非在子类中也定义slots，这样，子类实例允许定义的属性就是自身的slots加上父类的slots。

成员保护与访问机制

有些对象你不想外部访问，即使是通过调用类对象也无法访问，那就请认真学完本章节。

私有成员

```
class obj:
    def __init__(self,name):
        self.name=name
    def pri(self):
        print self.name
    __age = 18
    # 加上双下划线的就是私有变量，只能在类的内部访问，外部无法访问
a = obj('zhao')
a.pri()
```

运行结果：

```
zhao
```

如果要在类中调用这个私有成员，可以这么用

```
class obj:
    def __init__(self,name):
        self.name=name
    def prin(self):
        print self.name
    __age = 18
    # 加上双下划线的就是私有变量，只能在类的内部访问，外部无法访问
    @classmethod
    # 如果要在类中调用，首先调用类方法
    def pri(cls):
        print cls.__age
        # 然后在使用
a = obj('zhao')
a.prin()
obj.pri()
# 通过这样直接调用类中的私有变量
```

运行结果：

```
zhao
18
```

使用get-set-del方法操作私有成员

```
class obj:
    def __init__(self,name):
        self.name=name
    def prin(self):
        print self.name
    __age = 18
    # 加上双下划线的就是私有变量，只能在类的内部访问，外部无法访问
    @classmethod
    # 如果要在类中调用，首先调用类方法
    def pri(cls):
        print cls.__age
        # 然后在使用
    @classmethod
    def set_age(cls,value):
        cls.__age = value
        return cls.__age
        # 这个用法就是改变__age的值
    @classmethod
    def get_age(cls):
        return cls.__age
        # 这个用法就是直接返回__age的值
    @classmethod
    def del_age(cls):
        del cls.__age
        # 这个用法就是直接删除__age的值

print obj.get_age()
# 这里是直接调用出__age的值   返回值18
print obj.set_age(20)
# 这里是直接改变__age的值   返回值20
obj.del_age()
# 这里是直接删除__age的值
```

思考： 既然是私有变量，不让外部访问，为何有要在后面调用又改变呢？因为可以对私有变量进行额外的检测，处理，加工等等。比如判断value的值，使用isinstance然后做if-else判断。

使用私有变量可以对内部变量进行保护，外部无法改变，但是可以对它进行检测处理。

这里引申一下私有成员的保护机制，使用__age对私有变量其实就是—>obj._obj__age的样子进行保护，说白了你直接使用obj._obj__age就可以直接调用内部私有变量age了。

Propety装饰器

把类的方法伪装成属性调用的方式，就是把类里面的一个函数，变成一个属性一样的东西~
一开始调用类的方法要使用圆括号，现在变成了属性进行读取设置存储。
举个例子来说明：

常用的调用方法

```
class obj:
    def __init__(self,name,age):
        self.__name=name
        self.__age=age
        # 讲这些设置成私有变量
    def get_age(self):
        return self.__age
    def set_age(self,value):
        if isinstance(value,int):
            self.__age=value
        else:
            raise ValueError('非整数类型')
    def del_age(self):
        print 'delete over'
a = obj('langzi',18)
print a.get_age()
a.set_age(20)
print a.get_age()
```

使用装饰器

```
class obj:
    def __init__(self,name,age):
        self.__name=name
        self.__age=age
        # 把这些设置成私有变量
    @property
    def age(self):
        return self.__age
    @age.setter
    def age(self,value):
        if isinstance(value,int):
            self.__age=value
        else:
            raise ValueError('非整数类型')
    @age.deleter
    def age(self):
        print 'delete over'
a = obj('langzi',18)
# 使用这些装饰器，可以使用类与对象的方法直接调用
print a.age
# 这里就是直接调用返回age的值
a.age=20
# 这里就是直接使用setter把值转换
print a.age
del a.age
# 删除age
```

当然这种调用方法有些麻烦，每次都是一个一个去实例类与对象，有个更加简单直观的方法。

更加减半的使用property()函数

除了使用装饰器的方式将一个方法伪装成属性外，Python内置的builtins模块中的property()函数，为我们提供了第二种设置类属性的手段。

```
class People:

    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def get_age(self):
        return self.__age

    def set_age(self, age):
        if isinstance(age, int):
            self.__age = age
        else:
            raise ValueError

    def del_age(self):
        print("删除年龄数据！")

    # 核心在这句
    age = property(get_age, set_age, del_age, "年龄")


obj = People("jack", 18)
print(obj.age)
obj.age = 19
print("obj.age:  ", obj.age)
del obj.ag
```

通过语句age = property(get_age, set_age, del_age, “年龄”)将一个方法伪装成为属性。其效果和装饰器的方法是一样的。

property()函数的参数：

```
第一个参数是方法名，调用 实例.属性 时自动执行的方法
第二个参数是方法名，调用 实例.属性 = XXX时自动执行的方法
第三个参数是方法名，调用 del 实例.属性 时自动执行的方法
第四个参数是字符串，调用 实例.属性.__doc__时的描述信息。
```