模块产生的原因和作用：

在计算机程序的开发过程中，随着程序代码越写越多，在一个文件里的代码就会越来越长，越来越不容易维护。

为了编写可维护的代码，我们把很多函数分组，分别放到不同的文件里，这样每个文件包含的代码就相对较少，很多编程语言都采用这种组织代码的方式。在python中，一个.py文件就称之为一个模块(module)。

作用：

1.提高代码的可维护性

2.避免变量名冲突，相同名字的函数和变量完全可以存放在不同的模块中。

为了避免模块名冲突，python引入按目录来组织模块的方法，称为包(package)。

可以有多级目录，组成多级层次的包结构。比如如下的目录结构：

mycompany  
 ├─ web  
 │  ├─ \_\_init\_\_.py  
 │  ├─ utils.py  
 │  └─ [www.py](http://www.py)  
 ├─ \_\_init\_\_.py  
 ├─ abc.py  
 └─ xyz.py

文件www.py的模块名就是mycompany.web.www，两个文件utils.py的模块名分别是mycompany.utils和mycompany.web.utils。

 自己创建模块时要注意命名，不能和Python自带的模块名称冲突。例如，系统自带了sys模块，自己的模块就不可命名为sys.py，否则将无法导入系统自带的sys模块。

模块名不要和系统模块名冲突，最好先查看系统是否已存在该模块，检查方法是在Python交互环境执行import abc，若成功则说明系统存在此模块。

例

使用sys模块的第一步，就是导入该模块：

**import** sys

导入sys模块后，我们就有了变量sys指向该模块，利用sys这个变量，就可以访问sys模块的所有功能。

sys模块有一个argv变量，用list存储了命令行的所有参数。argv至少有一个元素，因为第一个参数永远是该.py文件的名称，例如：

运行python3 hello.py获得的sys.argv就是['hello.py']；

运行python3 hello.py Michael获得的sys.argv就是['hello.py', 'Michael]。

IMG_256 **if** \_\_name\_\_=='\_\_main\_\_':  
    test()

当我们在命令行运行hello模块文件时，Python解释器把一个特殊变量\_\_name\_\_置为\_\_main\_\_，而如果在其他地方导入该hello模块时，if判断将失败，因此，这种if测试可以让一个模块通过命令行运行时执行一些额外的代码，最常见的就是运行测试。

我们可以用命令行运行hello.py看看效果：

$ python3 hello.py  
Hello, world!  
$ python hello.py Michael  
Hello, Michael!

使用模块

阅读: 314961

Python本身就内置了很多非常有用的模块，只要安装完毕，这些模块就可以立刻使用。

我们以内建的sys模块为例，编写一个hello的模块：

*#!/usr/bin/env python3*  
*# -\*- coding: utf-8 -\*-*

' a test module '

\_\_author\_\_ = 'Michael Liao'

**import** sys

**def test**():  
    args = sys.argv  
    **if** len(args)==1:  
        print('Hello, world!')  
    **elif** len(args)==2:  
        print('Hello, %s!' % args[1])  
    **else**:  
        print('Too many arguments!')

**if** \_\_name\_\_=='\_\_main\_\_':  
    test()

第1行和第2行是标准注释，第1行注释可以让这个hello.py文件直接在Unix/Linux/Mac上运行，第2行注释表示.py文件本身使用标准UTF-8编码；

第4行是一个字符串，表示模块的文档注释，任何模块代码的第一个字符串都被视为模块的文档注释；

第6行使用\_\_author\_\_变量把作者写进去，这样当你公开源代码后别人就可以瞻仰你的大名；

以上就是Python模块的标准文件模板，当然也可以全部删掉不写，但是，按标准办事肯定没错。

后面开始就是真正的代码部分。

你可能注意到了，使用sys模块的第一步，就是导入该模块：

**import** sys

导入sys模块后，我们就有了变量sys指向该模块，利用sys这个变量，就可以访问sys模块的所有功能。

sys模块有一个argv变量，用list存储了命令行的所有参数。argv至少有一个元素，因为第一个参数永远是该.py文件的名称，例如：

运行python3 hello.py获得的sys.argv就是['hello.py']；

运行python3 hello.py Michael获得的sys.argv就是['hello.py', 'Michael]。

最后，注意到这两行代码：

**if** \_\_name\_\_=='\_\_main\_\_':  
    test()

当我们在命令行运行hello模块文件时，Python解释器把一个特殊变量\_\_name\_\_置为\_\_main\_\_，而如果在其他地方导入该hello模块时，if判断将失败，因此，这种if测试可以让一个模块通过命令行运行时执行一些额外的代码，最常见的就是运行测试。

我们可以用命令行运行hello.py看看效果：

$ python3 hello.py  
Hello, world!  
$ python hello.py Michael  
Hello, Michael!

如果启动Python交互环境，再导入hello模块：

$ python3  
Python 3**.4.3** (v3**.4.3**:9b73f1c3e601, Feb 23 2015, 02:52:03)   
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import hello  
>>>

导入时，没有打印Hello, word!，因为没有执行test()函数。

调用hello.test()时，才能打印出Hello, word!：

>>> hello.test()  
Hello, world!

作用域

在一个模块中，我们可能会定义很多函数和变量，但有的函数和变量我们希望给别人使用，有的函数和变量我们希望仅仅在模块内部使用。在Python中，是通过\_前缀来实现的。

正常的函数和变量名是公开的（public），可以被直接引用，比如：abc，x123，PI等；

类似\_\_xxx\_\_这样的变量是特殊变量，可以被直接引用，但是有特殊用途，比如上面的\_\_author\_\_，\_\_name\_\_就是特殊变量，hello模块定义的文档注释也可以用特殊变量\_\_doc\_\_访问，我们自己的变量一般不要用这种变量名；

类似\_xxx和\_\_xxx这样的函数或变量就是非公开的（private），不应该被直接引用，比如\_abc，\_\_abc等；

之所以我们说，private函数和变量“不应该”被直接引用，而不是“不能”被直接引用，是因为Python并没有一种方法可以完全限制访问private函数或变量，但是，从编程习惯上不应该引用private函数或变量。