



使用模块

2.7旧版教程

阅读: 309984

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。

`private`函数或变量不应该被别人引用，那它们有什么用呢？请看例子：

```
def _private_1(name):  
    return 'Hello, %s' % name
```

```
def _private_2(name):  
    return 'Hi, %s' % name  
  
def greeting(name):  
    if len(name) > 3:  
        return _private_1(name)  
    else:  
        return _private_2(name)
```

我们在模块里公开 `greeting()` 函数，而把内部逻辑用private函数隐藏起来了，这样，调用 `greeting()` 函数不用关心内部的private函数细节，这也是一种非常有用的代码封装和抽象的方法，即：

外部不需要引用的函数全部定义成private，只有外部需要引用的函数才定义为public。

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 赶快成为第一个分享的人吧

◀ 模块

安装第三方模块 ▶

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](#) - [4clojure](#)

