使用模块

2307次阅读

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 python
# -*- 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python hello.py获得的sys.argv就是['hello.py']；

运行python 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：

```
$ python 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：

```
$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] 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!
```

别名

导入模块时，还可以使用别名，这样，可以在运行时根据当前环境选择最合适的模块。比如Python标准库一般会提供StringIO和cStringIO两个库，这两个库的接口和功能是一样的，但是cStringIO是C写的，速度更快，所以，你会经常看到这样的写法：

```
try:
    import cStringIO as StringIO
except ImportError: # 导入失败会捕获到ImportError
    import StringIO
```

这样就可以优先导入cStringIO。如果有些平台不提供cStringIO，还可以降级使用StringIO。导入cStringIO时，用import ... as ...指定了别名StringIO，因此，后续代码引用StringIO即可正常工作。

还有类似simplejson这样的库，在Python 2.6之前是独立的第三方库，从2.6开始内置，所以，会有这样的写法：

```
try:
    import json # python >= 2.6
except ImportError:
    import simplejson as json # python <= 2.5
```

由于Python是动态语言，函数签名一致接口就一样，因此，无论导入哪个模块后续代码都能正常工作。

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。
