[import,reload,\_\_import\_\_在python中的区别](http://blog.csdn.net/five3/article/details/7762870)

标签： [import](http://www.csdn.net/tag/import)[python](http://www.csdn.net/tag/python)[module](http://www.csdn.net/tag/module)[os](http://www.csdn.net/tag/os)[测试](http://www.csdn.net/tag/%e6%b5%8b%e8%af%95)[encoding](http://www.csdn.net/tag/encoding)

2012-07-19 13:49 37466人阅读 [评论](http://blog.csdn.net/five3/article/details/7762870#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/five3/article/details/7762870#report)

 分类：

python（34）

版权声明：本文为博主原创文章，未经博主允许不得转载。

import  
作用：  
导入/引入一个python标准模块，其中包括.py文件、带有\_\_init\_\_.py文件的目录。  
e.g：

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. **import** module\_name[,module1,...]
2. **from** module **import** \*|child[,child1,...]

说明：  
多次重复使用import语句时，不会重新加载被指定的模块，只是把对该模块的内存地址给引用到本地变量环境。  
测试：

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. a.py
2. #!/usr/bin/env python
3. #encoding: utf-8
4. **import** os
5. **print** 'in a',id(os)
7. m.py
8. #!/usr/bin/env python
9. #encoding: utf-8
10. **import** a   #第一次会打印a里面的语句
11. **import** os  #再次导入os后，其内存地址和a里面的是一样的，因此这里只是对os的本地引用
12. **print** 'in c',id(os)
13. **import** a  #第二次不会打印a里面的语句，因为没有重新加载

**reload**  
作用：  
对已经加载的模块进行重新加载，一般用于原模块有变化等特殊情况，reload前该模块必须已经import过。  
e.g：  
import os  
reload(os)  
说明：  
reload会重新加载已加载的模块，但原来已经使用的实例还是会使用旧的模块，而新生产的实例会使用新的模块；reload后还是用原来的内存地址；不能支持from。。import。。格式的模块进行重新加载。  
测试：

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. a.py
2. #!/usr/bin/env python
3. #encoding: utf-8
4. **import** os
5. **print** 'in a',id(os)
7. m.py
8. #!/usr/bin/env python
9. #encoding: utf-8
10. **import** a   #第一次import会打印a里面的语句
11. **print** id(a) #原来a的内存地址
12. reload(a)  #第二次reload还会打印a里面的语句，因为有重新加载
13. **print** id(a) #reload后a的内存地址，和原来一样

扩展：  
上面说了，在特殊情况的下才会使用reload函数；除了原来模块文件有修改外，还有哪些情况需要使用reload函数呢，这里举个例子。

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. #!/usr/bin/env python
2. #encoding: utf-8
3. **import** sys   #引用sys模块进来，并不是进行sys的第一次加载
4. reload(sys)  #重新加载sys
5. sys.setdefaultencoding('utf8')  ##调用setdefaultencoding函数

上面的代码是正确的，再测试下面的代码

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. #!/usr/bin/env python
2. #encoding: utf-8
3. **import** sys
4. sys.setdefaultencoding('utf8')

上面的[**测试**](http://lib.csdn.net/base/softwaretest)会失败，那么为什么要在调用setdefaultencoding时必须要先reload一次sys模块呢？因为这里的import语句其实并不是sys的第一次导入语句，也就是说这里其实可能是第二、三次进行sys模块的import，这里只是一个对sys的引用，只能reload才能进行重新加载；那么为什么要重新加载，而直接引用过来则不能调用该函数呢？因为setdefaultencoding函数在被系统调用后被删除了，所以通过import引用进来时其实已经没有了，所以必须reload一次sys模块，这样setdefaultencoding才会为可用，才能在代码里修改解释器当前的字符编码。试试下面的代码，同样会报错：

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. #!/usr/bin/env python
2. #encoding: utf-8
3. **import** sys
4. reload(sys)
5. sys.setdefaultencoding('utf8')
6. **del** sys.setdefaultencoding   ##删除原来的setdefaultencoding函数
7. sys.setdefaultencoding('gb2312')

那么到底是谁在之前就导入sys并且调用了setdefaultencoding函数呢？答案就在python安装目录的Lib文件夹下，有一个叫site.py的文件【python2.6】，在里面可以找到main() --> setencoding()-->sys.setdefaultencoding(encoding),因为这个site.py每次启动python解释器时会自动加载，所以main函数每次都会被执行，setdefaultencoding函数一出来就已经被删除了。  
  
**\_\_import\_\_**  
作用：  
同import语句同样的功能，但\_\_import\_\_是一个函数，并且只接收字符串作为参数，所以它的作用就可想而知了。其实import语句就是调用这个函数进行导入工作的，import sys <==>sys = \_\_import\_\_('sys')  
e.g：

\_\_import\_\_(module\_name[, globals[, locals[, fromlist]]]) #可选参数默认为globals(),locals(),[]  
\_\_import\_\_('os')      
\_\_import\_\_('os',globals(),locals(),['path','pip'])  #等价于from os import path, pip

说明：

通常在动态加载时可以使用到这个函数，比如你希望加载某个文件夹下的所用模块，但是其下的模块名称又会经常变化时，就可以使用这个函数动态加载所有模块了，最常见的场景就是插件功能的支持。

扩展：  
既然可以通过字符串来动态导入模块，那么是否可以通过字符串动态重新加载模块吗？试试reload('os')直接报错，是不是没有其他方式呢?虽然不能直接reload但是可以先unimport一个模块，然后再\_\_import\_\_来重新加载模块。现在看看unimport操作如何实现，在[**Python**](http://lib.csdn.net/base/python)解释里可以通过globals(),locals(),vars(),dir()等函数查看到当前环境下加载的模块及其位置，但是这些都只能看不能删除，所以无法unimport；不过除此之外还有一个地方是专门存放模块的，这就是sys.modules，通过sys.modules可以查看所有的已加载并且成功的模块，而且比globals要多，说明默认会加载一些额外的模块，接下来就是unimport了。

**[python]** [view plain](http://blog.csdn.net/five3/article/details/7762870) [copy](http://blog.csdn.net/five3/article/details/7762870)

1. #!/usr/bin/env python
2. #encoding: utf-8
3. **import** sys
4. \_\_import\_\_('a')      #第一次导入会打印消息
5. **del** sys.modules['a']   #unimport
6. \_\_import\_\_('a')    #再次导入还是会打印消息，因为已经unimport一次了
7. \_\_import\_\_('a')    #这次就不会打印消息了