刀刀的专栏

[置顶] python模块及包的导入

标签： python

2014-03-31 17:01 47532人阅读 评论

分类： python (13)

一 .module

通常模块为一个文件，直接使用import来导入就好了。可以作为module的文件类型有".py"、".pyo"、".pyc"、".pyd"、".so"、".dll"。

二. package

通常包总是一个目录，可以使用import导入包，或者from + import来导入包中的部分模块。包目录下为首的一个文件便是__init__.py。然后是一些模块文件和子目录，假如子目录中也有 __init__.py 那么它就是这个包的子包了。

一.模块你可以使用import语句将一个源代码文件作为模块导入.例如:

```
[python]
# file : spam.py
a = 37                      # 一个变量
def foo:                    # 一个函数
    print "I'm foo"
class bar:                  # 一个类
    def grok(self):
```

文章分类

MySQL (57)
linux (27)
web前端开发 (10)
html5 (5)
javascript (13)
jQuery (1)
svn (1)
PHP (1)
WampServer (1)
phpMyAdmin (1)
HTML DOM (1)
Google chrome (1)
canvas (2)
Android (5)
Eclipse (3)
IDE (2)
JRE (3)
JDK (3)
Ubuntu (6)
dbus (3)
dbus-glib (1)
clutter (1)
glib (1)
vim (1)
LVM (2)
FULLTEXT (1)
index (2)
InnoDB (10)
mysqldump (3)

```
07.         print "I'm bar.grok"
08. b = bar()                   # 创建一个实例
```

使用import spam 语句就可以将这个文件作为模块导入。系统在导入模块时，要做以下三件事：

1.为源代码文件中定义的对象创建一个名字空间，通过这个名字空间可以访问到模块中定义的函数及变量。

2.在新创建的名字空间里执行源代码文件.

3.创建一个名为源代码文件的对象，该对象引用模块的名字空间，这样就可以通过这个对象访问模块中的函数及变量，如：

[python]

```
01. import spam             # 导入并运行模块 spam
02. print spam.a            # 访问模块 spam 的属性
03. spam.foo()
04. c = spam.bar()
```

用逗号分割模块名称就可以同时导入多个模块:

[python]

```
01. import socket, os, regex
```

模块导入时可以使用 as 关键字来改变模块的引用对象名字:

关闭

[python]

```
01. import os as system
02. import socket as net, thread as threads
03. system.chdir("..")
04. net.gethostname()
```

使用from语句可以将模块中的对象直接导入到当前的名字空间. from语句不创建一个到模块名字空间的引用对象，而是把被导入模块的一个或多个对象直接放入当前的名字空间:

[python]

```
01. from socket import gethostname # 将gethostname放如当前名字空间
02. print gethostname()            # 直接调用
03. socket.gethostname()           # 引发异常NameError: socket
```

from语句支持逗号分割的对象，也可以使用星号(*)代表模块中除下划线开头的所有对象:

```
[python]
01. from socket import gethostname, socket
02. from socket import *    # 载入所有对象到当前名字空间
```

不过，如果一个模块如果定义有列表__all__，则from module import * 语句只能导入__all__列表中存在的对象。

```
[python]
01. # module: foo.py
02. __all__ = [ 'bar', 'spam' ]     # 定义使用 `*` 可以导入的对象
```

另外, as 也可以和 from 联合使用:

```
[python]
01. from socket import gethostname as hostname
02. h = hostname()
```

import 语句可以在程序的任何位置使用，你可以在程序中多次导入同一个模块，但模块中的代码*仅仅*在该模块被首次导入时执行。后面的import语句只是简单的创建一个到模块名字空间的引用而已。sys.modules字典中保存着所有被导入模块的模块名到模块对象的映射。这个字典用来决定是否需要使用import语句来导入一个模块的最新拷贝.
from module import * 语句只能用于一个模块的最顶层.*特别注意*：由于存在作用域冲突，不允许在函数中使用from 语句。
每个模块都拥有 __name__ 属性，它是一个内容为模块名字的字符串。最顶层的模块名称是 __main__ .命令行或是交互模式下程序都运行在__main__ 模块内部. 利用__name__属性，我们可以让同一个程序在不同的场合（单独执行或被导入)具有不同的行为，象下面这样做：

```
[python]
```

关闭

SockJS (1)
zabbix (1)
cProfile (1)
p (0)
profile (1)
pstats (1)
Redis (4)
Re (0)
supervisor (1)
dj (0)
Lock (6)
LockWait (1)
tra (0)
transaction (1)
Monitor (0)
Optimization (2)
s (0)
op (0)
ans (1)
ansible (0)
IT Automation (0)

文章存档

2016年10月 (2)
2016年06月 (1)
2016年04月 (1)
2016年03月 (1)
2016年02月 (1)

展开

阅读排行

```
# 检查是单独执行还是被导入

if __name__ == '__main__':
      # Yes
      statements
else:
      # No (可能被作为模块导入)
      statements
```

模块搜索路径

导入模块时,解释器会搜索sys.path列表,这个列表中保存着一系列目录。一个典型的sys.path 列表的值：

Linux:

['', '/usr/local/lib/python2.0',

'/usr/local/lib/python2.0/plat-sunos5',

'/usr/local/lib/python2.0/lib-tk',

'/usr/local/lib/python2.0/lib-dynload',

'/usr/local/lib/python2.0/site-packages']

Windows:

['', 'C:\\WINDOWS\\system32\\python24.zip', 'C:\\Documents and Settings\\weizh

'C:\\Python24\\lib', 'C:\\Python24\\lib\\plat-win', 'C:\\Python24\\lib\\lib-tk', 'C:\\Python24\\Lib\\site-packages\\py

'C:\\Python24', 'C:\\Python24\\lib\\site-packages', 'C:\\Python24\\lib\\site-packages\\win32', 'C:\\Python24\\lib\\site-packages\\win32\\lib', 'C:\\Python24\\lib\\site-packages\\wx-2.6-msw-unicode']

空字符串 代表当前目录. 要加入新的搜索路径,只需要将这个路径加入到这个列表.

模块导入和汇编

到现在为止，本章介绍的模块都是包含Python源代码的文本文件. 不过模块不限于此，可以被 import 语句导入的模块共有以下四类:

•使用Python写的程序(.py文件)

关闭

•C或C++扩展(已编译为共享库或DLL文件)

•包(包含多个模块)

•内建模块(使用C编写并已链接到Python解释器内)

当查询模块 foo 时,解释器按照 sys.path 列表中目录顺序来查找以下文件(目录也是文件的一种):

1.定义为一个包的目录 foo

2.foo.so, foomodule.so, foomodule.sl,或 foomodule.dll (已编译扩展)

3.foo.pyo (只在使用 -O 或 -OO 选项时)

4.foo.pyc

5.foo.py

对于.py文件,当一个模块第一次被导入时,它就被汇编为字节代码,并将字节码写入一个同名的 .pyc文件.后来

接读取.pyc文件而不是.py文件.(除非.py文件的修改日期更新,这种情况会重新生成.pyc文件) 在解释器使用 -

名为.pyo的同名文件被使用. pyo文件的内容虽去掉行号,断言,及其他调试信息的字节码，体积更小,运行速度

OO选项代替-O,则文档字符串也会在创建.pyo文件时也被忽略.

如果在sys.path提供的所有路径均查找失败,解释器会继续在内建模块中寻找,如果再次失败，则引发 ImportEr

.pyc和.pyo文件的汇编,当且仅当import 语句执行时进行.

当 import 语句搜索文件时,文件名是大小写敏感的。即使在文件系统大小写不敏感的系统上也是如此(Windo

import foo 只会导入文件foo.py而不会是FOO.PY.

关闭

重新导入模块

如果更新了一个已经用import语句导入的模块，内建函数reload()可以重新导入并运行更新后的模块代码.它需要一个模块对象做为参数.例如:

```
import foo
... some code ...
reload(foo)        # 重新导入 foo
```

在reload()运行之后的针对模块的操作都会使用新导入代码，不过reload()并不会更新使用旧模块创建的对象，因此有可能出现新旧版本对象共存的情况。 *注意* 使用C或C++编译的模块不能通过 reload() 函数来重新导入。记住一个原则，除非

是在调试和开发过程中，否则不要使用reload()函数.

2.包

多个关系密切的模块应该组织成一个包，以便于维护和使用。这项技术能有效避免名字空间冲突。创建一个名字为包名字的文件夹并在该文件夹下创建一个__init__.py 文件就定义了一个包。你可以根据需要在该文件夹下存放资源文件、已编译扩展及子包。举例来说，一个包可能有以下结构:

```
Graphics/
    __init__.py
    Primitive/
      __init__.py
      lines.py
      fill.py
      text.py
      ...
    Graph2d/
      __init__.py
      plot2d.py
      ...
    Graph3d/
      __init__.py
      plot3d.py
      ...
    Formats/
      __init__.py
      gif.py
      png.py
      tiff.py
```

关闭

jpeg.py

import语句使用以下几种方式导入包中的模块:

```python
import Graphics.Primitive.fill #导入模块Graphics.Primitive.fill,只能以全名访问模块属性,例如 Graphics.Primitive.fill.floodfill(img,x,y,color).
from Graphics.Primitive import fill# 导入模块fill ,只能以 fill.属性名这种方式访问模块属性,例如 fill.floodfill(img,x,y,color).
from Graphics.Primitive.fill import floodfill #导入模块fill ,并将函数floodfill放入当前名称空间,直接访问被导入的属性，例如 floodfill(img,x,y,color).
```

无论一个包的哪个部分被导入, 在文件__init__.py中的代码都会运行.这个文件的内容允许为空,不过通常情况包的初始化代码。导入过程遇到的所有 __init__.py文件都被运行.因此 import Graphics.Primitive.fill 语句会Ⅲ Graphics 和 Primitive 文件夹下的__init__.py文件.

下边这个语句具有歧义:

```python
from Graphics.Primitive import *
```

这个语句的原意图是想将Graphics.Primitive包下的所有模块导入到当前的名称空间.然而,由于不同平台间文件名规则不同(比如大小写敏感问题), Python不能正确判定哪些模块要被导入.这个语句只会顺序运行 Graphics 和 Primitive 文件夹下的__init__.py文件. 要解决这个问题，应该在Primitive文件夹下面的__init__.py中定义一个名字all的列表，例如:

```python
# Graphics/Primitive/__init__.py
__all__ = ["lines","text","fill",...]
```

这样,上边的语句就可以导入列表中所有模块.

下面这个语句只会执行Graphics目录下的__init__.py文件，而不会导入任何模块:

```python
import Graphics
Graphics.Primitive.fill.floodfill(img,x,y,color)  # 失败!
```

不过既然 import Graphics 语句会运行 Graphics 目录下的 __init__..py文件,我们就可以采取下面的手段来解决这个问题：

```python
# Graphics/__init__.py
import Primitive, Graph2d, Graph3d
# Graphics/Primitive/__init__.py
import lines, fill, text, ...
```

这样import Graphics语句就可以导入所有的子模块(只能用全名来访问这些模块的属性).

sys.path 和sys.modules

sys.path包含了module的查找路径；

sys.modules包含了当前所load的所有的modules的dict（其中包含了builtin的modules）；

上一篇 将python源程序编译为pyc或pyo字节码程序

下一篇 Python base64模块详解

相关文章推荐

- Python import .pyd 可能遇到路径的问题
- 自然语言处理在“天猫精灵”的实践应用--姜飞俊
- pandas小记：pandas数据结构
- 蚂蜂窝大数据平台架构及Druid引擎实践--汪木铃
- 在python代码中导入自己写的.py文件
- Retrofit 从入门封装到源码解析
- Python自定义包引入
- 程序员如何转型AI工程师
- python中自定义包的导入和使用
- 深入探究Linux/VxWorks的设备树
- python模块导入及属性：import
- 使用QEMU搭建u-boot+Linux+NFS嵌入式开发环境
- Python导入模块的几种姿势
- python最简单直接的自定义模块导入方法
- python学习笔记——自定义模块导入
- python 子包引用父包和其他子包

查看评论

2楼 feng1456 2017-03-24 16:43发表

由于不同平台间文件名规则不同(比如大小写敏感问题), Python不能正确判定哪些模块要被导入 不是特别理解，不同对

Re: hunter___ 2017-09-19 13:58发表

回复feng1456：sys.path 这个怎么定义啊。用的linux 终端

1楼 sinat_20998879 2016-04-23 16:29发表

由于不同平台间文件名规则不同(比如大小写敏感问题), Python不能正确判定哪些模块要被导入 - - 这是什么意思？模块里面的文件名不是在导入的时候已经固定了吗？求解

您还没有登录,请[登录]或[注册]

关闭