



Python 模块

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

模块让你能够有逻辑地组织你的 Python 代码段。

把相关的代码分配到一个模块里能让你的代码更好用，更易懂。

模块能定义函数，类和变量，模块里也能包含可执行的代码。

例子

下例是个简单的模块 support.py：

support.py 模块：

```
def print_func( par ):
    print "Hello : ", par
    return
```

import 语句

模块的引入

模块定义好后，我们可以使用 import 语句来引入模块，语法如下：

```
import module1[, module2[,... moduleN]]
```

比如要引用模块 math，就可以在文件最开始的地方用 **import math** 来引入。在调用 math 模块中的函数时，必须这样引用：

模块名.函数名

分类导航

HTML / CSS

JavaScript

服务端

数据库

移动端

XML 教程

ASP.NET

Web Service

开发工具

网站建设

Advertisement



反馈/建议

Python 函数
Python 模块
Python 文件I/O
Python File 方法
Python 异常处理
Python OS 文件/目录方法
Python 内置函数
Python 高级教程
Python 面向对象
Python 正则表达式
Python CGI 编程
Python MySQL
Python 网络编程
Python SMTP
Python 多线程
Python XML 解析
Python GUI 编程(Tkinter)
Python2.x与3.x版本区别
Python IDE
Python JSON
Python 100例
Python 测验

当解释器遇到 import 语句，如果模块在当前的搜索路径就会被导入。

搜索路径是一个解释器会先进行搜索的所有目录的列表。如想要导入模块 support.py，需要把命令放在脚本的顶端：

test.py 文件代码：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 导入模块
import support

# 现在可以调用模块里包含的函数了
support.print_func("Runoob")
```

以上实例输出结果：

```

Hello : Runoob
```

一个模块只会被导入一次，不管你执行了多少次import。这样可以防止导入模块被一遍又一遍地执行。

from...import 语句

Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中。语法如下：

```
from modname import name1[, name2[, ... nameN]]
```



例如，要导入模块 fib 的 fibonacci 函数，使用如下语句：

```
from fib import fibonacci
```

这个声明不会把整个 fib 模块导入到当前的命名空间中，它只会将 fib 里的 fibonacci 单个引入到执行这个声明的模块的全局符号表。

from...import* 语句

把一个模块的所有内容全都导入到当前的命名空间也是可行的，只需使用如下声明：






AplusCode

apluscode.net

编程难学不会

BAT资深
工程师，藤校
985学霸专业
辅导编程

打



反馈/建议

```
from modname import *
```

这提供了一个简单的方法来导入一个模块中的所有项目。然而这种声明不该被过多地使用。

例如我们想一次性引入 math 模块中所有的东西，语句如下：

```
from math import *
```

搜索路径

当你导入一个模块，Python 解析器对模块位置的搜索顺序是：

- 1、当前目录
- 2、如果不在当前目录，Python 则搜索在 shell 变量 PYTHONPATH 下的每个目录。
- 3、如果都找不到，Python会察看默认路径。UNIX下，默认路径一般为/usr/local/lib/python/。

模块搜索路径存储在 system 模块的 sys.path 变量中。变量里包含当前目录，PYTHONPATH和由安装过程决定的默认目录。

PYTHONPATH 变量

作为环境变量，PYTHONPATH 由装在一个列表里的许多目录组成。PYTHONPATH 的语法和 shell 变量 PATH 的一样。

在 Windows 系统，典型的 PYTHONPATH 如下：

```
set PYTHONPATH=c:\python27\lib;
```

在 UNIX 系统，典型的 PYTHONPATH 如下：

```
set PYTHONPATH=/usr/local/lib/python
```

命名空间和作用域

变量是拥有匹配对象的名字（标识符）。命名空间是一个包含了变量名称们（键）和它们各自相应的对象们（值）的字典。



一个 Python 表达式可以访问局部命名空间和全局命名空间里的变量。如果一个局部变量和一个全局变量重名，则局部变量会覆盖全局变量。

每个函数都有自己的命名空间。类的方法的作用域规则和通常函数的一样。

Python 会智能地猜测一个变量是局部的还是全局的，它假设任何在函数内赋值的变量都是局部的。

因此，如果要给函数内的全局变量赋值，必须使用 `global` 语句。

`global VarName` 的表达式会告诉 Python，`VarName` 是一个全局变量，这样 Python 就不会在局部命名空间里寻找这个变量了。

例如，我们在全局命名空间里定义一个变量 `Money`。我们再在函数内给变量 `Money` 赋值，然后 Python 会假定 `Money` 是一个局部变量。然而，我们并没有在访问前声明一个局部变量 `Money`，结果就是会出现一个 `UnboundLocalError` 的错误。取消 `global` 语句前的注释符就能解决这个问题。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

Money = 2000
def AddMoney():
    # 想改正代码就取消以下注释：
    # global Money
    Money = Money + 1

print Money
AddMoney()
print Money
```

dir()函数

`dir()` 函数一个排好序的字符串列表，内容是一个模块里定义过的名字。

返回的列表容纳了在在一个模块里定义的所有模块，变量和函数。如下一个简单的实例：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 导入内置math模块
```

```
import math

content = dir(math)

print content;
```

以上实例输出结果：

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

在这里，特殊字符串变量__name__指向模块的名字，__file__指向该模块的导入文件名。

globals() 和 locals() 函数

根据调用地方的不同，globals() 和 locals() 函数可被用来返回全局和局部命名空间里的名字。

如果在函数内部调用 locals()，返回的是所有能在该函数里访问的命名。

如果在函数内部调用 globals()，返回的是所有在该函数里能访问的全局名字。

两个函数的返回类型都是字典。所以名字们能用 keys() 函数摘取。

reload() 函数

当一个模块被导入到一个脚本，模块顶层部分的代码只会被执行一次。

因此，如果你想重新执行模块里顶层部分的代码，可以用 reload() 函数。该函数会重新导入之前导入过的模块。语法如下：

```
reload(module_name)
```

在这里，module_name要直接放模块的名字，而不是一个字符串形式。比如想重载 hello 模块，如下：

```
reload(hello)
```



Python中的包

包是一个分层次的文件目录结构，它定义了一个由模块及子包，和子包下的子包等组成的 Python 的应用环境。

简单来说，包就是文件夹，但该文件夹下必须存在 `__init__.py` 文件，该文件的内容可以为空。`__init__.py` 用于标识当前文件夹是一个包。

考虑一个在 `package_runoob` 目录下的 `runoob1.py`、`runoob2.py`、`__init__.py` 文件，`test.py` 为测试调用包的代码，目录结构如下：

```
test.py
package_runoob
|-- __init__.py
|-- runoob1.py
|-- runoob2.py
```

源代码如下：

package_runoob/runoob1.py

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

def runoob1():
    print "I'm in runoob1"
```

package_runoob/runoob2.py

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

def runoob2():
    print "I'm in runoob2"
```

现在，在 `package_runoob` 目录下创建 `__init__.py`：

package_runoob/__init__.py

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
```



```
if __name__ == '__main__':  
    print '作为主程序运行'  
else:  
    print 'package_runoob 初始化'
```

然后我们在 `package_runoob` 同级目录下创建 `test.py` 来调用 `package_runoob` 包

test.py

```
#!/usr/bin/python  
# -*- coding: UTF-8 -*-  
  
# 导入 Phone 包  
from package_runoob.runoob1 import runoob1  
from package_runoob.runoob2 import runoob2  
  
runoob1()  
runoob2()
```

以上实例输出结果：

```
package_runoob 初始化  
I'm in runoob1  
I'm in runoob2
```

如上，为了举例，我们只在每个文件里放置了一个函数，但其实你可以放置许多函数。你也可以在这些文件里定义Python的类，然后为这些类建一个包。

← Python 函数

Python 文件I/O →



5 篇笔记



写笔记

AplusCode - 专注编程辅导

微信号ITCSdaixie | 专注编程辅导 | 保原创 | 保售后 apluscode.net

反馈/建议

在线实例

- [HTML 实例](#)
- [CSS 实例](#)
- [JavaScript 实例](#)
- [Ajax 实例](#)
- [jQuery 实例](#)
- [XML 实例](#)
- [Java 实例](#)

字符集&工具

- [HTML 字符集设置](#)
- [HTML ASCII 字符集](#)
- [HTML ISO-8859-1](#)
- [HTML 实体符号](#)
- [HTML 拾色器](#)
- [JSON 格式化工具](#)

最新更新

- [什么是闭包？闭包的应用](#)
- [Python3 assert 断言](#)
- [C 语言中的 time 函数](#)
- [PHP imagecreate 函数](#)
- [PostgreSQL 常用命令](#)
- [PostgreSQL 时间函数](#)
- [PostgreSQL PRIVILEGE 权限](#)

站点信息

- [意见反馈](#)
- [免责声明](#)
- [关于我们](#)
- [文章归档](#)

关注微信



Copyright © 2013-2019 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



反馈/建议