技术文章 来源：编程派 发布：2016-05-27 浏览：1069

摘要：作为一名新手Python程序员，你首先需要学习的内容之一就是如何导入模块或包。但是我注意到，那些许多年来不时使用Python的人并不是都知道Python的导入机制其实非常灵活。在本文中，我们将探讨以下话题：

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

作为一名新手Python程序员，你首先需要学习的内容之一就是如何导入模块或包。但是我注意到，那些许多年来不时使用Python的人并不是都知道Python的导入机制其实非常灵活。在本文中，我们将探

推荐工具 意见反馈

- 相对导入（relative imports）
- 可选导入（optional imports）
- 本地导入（local imports）
- 导入注意事项

常规导入

常规导入应该是最常使用的导入方式，大概是这样的：

```
import sys
```

你只需要使用import一词，然后指定你希望导入的模块或包即可。通过这种方式导入的好处是可以一

下一篇

在 ES6 中 改良的 5 个 JavaScript “缺陷”

虽然这节省了空间，但是却违背了Python风格指南。**Python**风格指南建议将每个导入语句单独成行。

有时在导入模块时，你想要重命名这个模块。这个功能很容易实现：

```
import sys as system
```

```
import urllib.error
```

这个情况不常见，但是对此有所了解总是没有坏处的。

使用from语句导入

很多时候你只想要导入一个模块或库中的某个部分。我们来看看在Python中如何实现这点：

```
from functools import lru_cache
```

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

che。如果你按常规方式导入functools，那么你就必须像这样调

```
functools.lru_cache(*args)
```

根据你实际的使用场景，上面的做法可能是更好的。在复杂的代码库中，能够看出某个函数是从哪里导入的这点很有用的。不过，如果你的代码维护的很好，模块化程度高，那么只从某个模块中导入一部分内容也是非常方便和简洁的。

当然，你还可以使用from方法导入模块的全部内容，就像这样：

是你自己定义的内容。因此，你最后可能会碰到一个相当让人困惑的逻辑错误。标准库中我唯一推荐全盘导入的模块只有**Tkinter**。

如果你正好要写自己的模块或包，有人会建议你在__init__.py文件中导入所有内容，让模块或者包使用起来更方便。我个人更喜欢显示地导入，而非隐式地导入。

你也可以采取折中方案，从一个包中导入多个项：

```
from os import path, walk, unlink
```

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

5个函数。你可能注意到了，我们是通过多次从同一个模块中导
也可以使用圆括号一次性导入多个项：

```
from os import (path, walk, unlink, uname,
                remove, rename)
```

这是一个有用的技巧，不过你也可以换一种方式：

推荐工具
意见反馈

相对导入

PEP 328 介绍了引入相对导入的原因，以及选择了哪种语法。具体来说，是使用句点来决定如何相对导入其他包或模块。这么做的原因是为了避免偶然情况下导入标准库中的模块产生冲突。这里我们以PEP 328中给出的文件夹结构为例，看看相对导入是如何工作的：

```
my_package/
    __init__.py
    subpackage1/
        __init__.py
        module_z.py
    module_a.py
```

在本地磁盘上找个地方创建上述文件和文件夹。在顶层的__init__.py文件中，输入以下代码：

```
from . import subpackage1
from . import subpackage2
```

下一篇

在 ES6 中 改良的 5 个 JavaScript “缺陷”

推荐工具 意见反馈

```
from . import module_y
```

现在编辑module_x.py文件，输入以下代码：

```
from .module_y import spam as ham

def main():
    ham()
```

最后编辑module_y.py文件，输入以下代码：

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

打开终端，cd至my_package包所在的文件夹，但不要进入my_package。在这个文件夹下运行Python解释器。我使用的是IPython，因为它的自动补全功能非常方便：

```
In [1]: import my_package

In [2]: my_package.subpackage1.module_x
Out[2]: <module 'my_package.subpackage1.module_x' from 'my_package/subpackage1/module_x.p
```

推荐工具 意见反馈

相对导入适用于你最终要放入包中的代码。如果你编写了很多相关性强的代码，那么应该采用这种导入方式。你会发现**PyPI**上有很多流行的包也是采用了相对导入。还要注意一点，如果你想要跨越多个文件层级进行导入，只需要使用多个句点即可。不过，**PEP 328**建议相对导入的层级不要超过两层。

还要注意一点，如果你往module_x.py文件中添加了if __name__ == '__main__'，然后试图运行这个文件，你会碰到一个很难理解的错误。编辑一下文件，试试看吧！

```
from . module_y import spam as ham
```

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

```
    # This won't work!
    main()
```

现在从终端进入subpackage1文件夹，执行以下命令：

```
python module_x.py
```

如果你使用的是Python 2，你应该会看到下面的错误信息：

如果你使用的是Python 3，错误信息大概是这样的：

```
Traceback (most recent call last):
  File "module_x.py", line 1, in <module>
    from . module_y import spam as ham
SystemError: Parent module '' not loaded, cannot perform relative import
```

这指的是，module_x.py是某个包中的一个模块，而你试图以脚本模式执行，但是这种模式不支持相对导入。

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

，那么你必须将其添加至Python的导入检索路径（import

```
import sys
sys.path.append('/path/to/folder/containing/my_package')
import my_package
```

注意，你需要添加的是my_package的上一层文件夹路径，而不是my_package本身。原因是my_package就是我们想要使用的包，所以如果你添加它的路径，那么将无法使用这个包。

可选导入这种方式。这样做可以导入支持某个软件的多种版本或者实现性能提升。以 github2包 中的代码为例：

```
try:
    # For Python 3
    from http.client import responses
except ImportError:  # For Python 2.5-2.7
    try:
        from httplib import responses  # NOQA
    except ImportError:  # For Python 2.4
```

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

```
aseHTTPRequestHandler as _BHRH
 in _BHRH.responses.items()])
```

```
try:
    from urlparse import urljoin
    from urllib2 import urlopen
except ImportError:
    # Python 3
```

推荐工具 意见反馈

局部导入

当你在局部作用域中导入模块时，你执行的就是局部导入。如果你在Python脚本文件的顶部导入一个模块，那么你就是在将该模块导入至全局作用域，这意味着之后的任何函数或方法都可能访问该模块。例如：

```
import sys  # global scope

def square_root(a):
    # This import is into the square_root functions local scope
    return math.pow(base_num, power)

if __name__ == '__main__':
    print(square_root(49))
    print(my_pow(2, 3))
```

这里，我们将sys模块导入至全局作用域，但我们并没有使用这个模块。然后，在square_root函数中，我们将math模块导入至该函数的局部作用域，这意味着math模块只能在square_root函数内部使用。

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

某个不经常调用的函数中或许更加合理，而不是直接在全局作用域中导入。老实说，我几乎从没有使用过局部导入，主要是因为如果模块内部到处都有导入语句，会很难分辨出这样做的原因和用途。根据约定，所有的导入语句都应该位于模块的顶部。

导入注意事项

在导入模块方面，有几个程序员常犯的错误。这里我们介绍两个。

- 循环导入（circular imports）
- 覆盖导入（Shadowed imports，暂时翻译为覆盖导入）

下一篇

在 ES6 中 改良的 5 个 JavaScript “缺陷”

x[redacted]方，那么就会出现循环导入。例如：

```
# a.py
import b

def a_test():
    print("in a_test")
    b.b_test()
```

```
import a

def b_test():
    print('In test_b"')
    a.a_test()

b_test()
```

如果你运行任意一个模块，都会引发AttributeError。这是因为这两个模块都在试图导入对方。简单来[illegible]模块b也在试图导入模块a（这时正在执行），模块a将无法完成模[illegible]的破解方法（hack），但是一般来说，你应该做的是重构代码，

下一篇
在 ES6 中 改良的 5 个 JavaScript “缺陷”

覆盖导入

当你创建的模块与标准库中的模块同名时，如果你导入这个模块，就会出现覆盖导入。举个例子，创建一个名叫math.py的文件，在其中写入如下代码：

```
import math

def square_root(number):
```

现在打开终端，试着运行这个文件，你会得到以下回溯信息（traceback）：

```
Traceback (most recent call last):
  File "math.py", line 1, in <module>
    import math
  File "/Users/michael/Desktop/math.py", line 6, in <module>
    square_root(72)
  File "/Users/michael/Desktop/math.py", line 4, in square_root
    return math.sqrt(number)
AttributeError: module 'math' has no attribute 'sqrt'
```

下一篇

在 ES6 中 改良的 5 个 JavaScript “缺陷”

文件的时候，Python解释器首先在当前运行脚本所处的的文件
子中，解释器找到了我们正在执行的模块，试图导入它。但是我
，所以就抛出了AttributeError。

总结

在本文中，我们讲了很多有关导入的内容，但是还有部分内容没有涉及。PEP 302 中介绍了导入钩子（import hooks），支持实现一些非常酷的功能，比如说直接从github导入。Python标准库中还有一个 importlib 模块，值得查看学习。当然，你还可以多看看别人写的代码，不断挖掘更多好用的妙招。

> 本文作者：**Mike**
> 原文链接： Python导入模块的几种姿势

原　文：编程派

免责声明：

1. SDK.cn遵循行业规范，所有转载文章均征得作者同意授权并明确标注来源和链接。
2. 我们十分尊重原创作者的付出，本站禁止二次转载如需转载请与原作者取得联系。
3. 转载SDK.cn的原创文章需注明文章作者、链接和"来源：SDK.cn"并保留文章及标题完整性未经作者同意不得擅自修改。
4. 作者投稿可能会经SDK.cn适当编辑修改或补充。

登录

下一篇

在 ES6 中 改良的 5 个 JavaScript “缺陷”

还没有评论，快来抢沙发吧！

热门资讯

如何利用SOTER，1个版本内完成指纹支付开发？

Facebook发布了React 16

用路由器搭建 facebook ATC

推荐工具

意见反馈