Python 相对导入与绝对导入

Python | Jul 21, 2016 | python

Python 相对导入与绝对导入，这两个概念是相对于包内导入而言的。包内导入即是包内的模块导入包内部的模块。

Python import 的搜索路径

- 在当前目录下搜索该模块
- 在环境变量 PYTHONPATH 中指定的路径列表中依次搜索
- 在 Python 安装路径的 lib 库中搜索

Python import 的步骤

python 所有加载的模块信息都存放在 `sys.modules` 结构中，当 import 一个模块时，会按如下步骤来进行

- 如果是 `import A`，检查 sys.modules 中是否已经有 A，如果有则不加载，如果没有则为 A 创建 module 对象，并加载 A
- 如果是 `from A import B`，先为 A 创建 module 对象，再解析A，从中寻找B并填充到 A 的 `__dict__` 中

相对导入与绝对导入

绝对导入的格式为 `import A.B` 或 `from A import B`，相对导入格式为 `from . import B` 或 `from ..A import B`，`.` 代表当前模块，`..` 代表上层模块，`...` 代表上上层模块，依次类推。

相对导入可以避免硬编码带来的维护问题，例如我们改了某一顶层包的名，那么其子包所有的导入就都不能用了。但是 **存在相对导入语句的模块，不能直接运行**，否则会有异常：

```
ValueError: Attempted relative import in non-package
```

这是什么原因呢？我们需要先来了解下导入模块时的一些规则：

在没有明确指定包结构的情况下，Python 是根据 `__name__` 来决定一个模块在包中的结构的，如果是 `__main__` 则它本身是顶层模块，没有包结构，如果是 `A.B.C` 结构，那么顶层模块是 A。基本上遵循这样的原则：

- 如果是绝对导入，一个模块只能导入自身的子模块或和它的顶层模块同级别的模块及其子模块
- 如果是相对导入，一个模块必须有包结构且只能导入它的顶层模块内部的模块

如果一个模块被直接运行，则它自己为顶层模块，不存在层次结构，所以找不到其他的相对路径。

Python2.x 缺省为相对路径导入，Python3.x 缺省为绝对路径导入。绝对导入可以避免导入子包覆盖掉标准库模块（由于名字相同，发生冲突）。如果在 Python2.x 中要默认使用绝对导入，可以在文件开头加入如下语句：

```
from __future__ import absolute_import
```

from __future__ import absolute_import

这句 import 并不是指将所有的导入视为绝对导入，而是指禁用 `implicit relative import`（隐式相对导入），但并不会禁掉 `explicit relative import`（显示相对导入）。

那么到底什么是隐式相对导入，什么又是显示的相对导入呢？我们来看一个例子，假设有如下包结构：

```
thing
├── books
│   ├── adventure.py
│   ├── history.py
│   ├── horror.py
│   ├── __init__.py
│   └── lovestory.py
├── furniture
│   ├── armchair.py
│   ├── bench.py
│   ├── __init__.py
│   ├── screen.py
│   └── stool.py
└── __init__.py
```

那么如果在 stool 中引用 bench，则有如下几种方式：

```
import bench                # 此为 implicit relative import
from . import bench         # 此为 explicit relative import
from furniture import bench  # 此为 absolute import
```

隐式相对就是没有告诉解释器相对于谁，但默认相对与当前模块；而显示相对则明确告诉解释器相对于谁来导入。以上导入方式的第三种，才是官方推荐的，第一种是官方强烈不推荐的，**Python3 中已经被废弃**，这种方式只能用于导入 path 中的模块。

相对与绝对仅针对包内导入而言

最后再次强调，相对导入与绝对导入仅针对于包内导入而言，要不然本文所讨论的内容就没有意义。所谓的包，就是包含 `__init__.py` 文件的目录，该文件在包导入时会被首先执行，该文件可以为空，也可以在其中加入任意合法的 Python 代码。

相对导入可以避免硬编码，对于包的维护是友好的。绝对导入可以避免与标准库命名的冲突，实际上也不推荐自定义模块与标准库命令相同。

前面提到含有相对导入的模块不能被直接运行，实际上含有绝对导入的模块也不能被直接运行，会出现 ImportError：

```
ImportError: No module named XXX
```

这与绝对导入时是一样的原因。要运行包中包含绝对导入和相对导入的模块，可以用 `python -m A.B.C` 告诉解释器模块的层次结构。

有人可能会问：假如有两个模块 a.py 和 b.py 放在同一个目录下，为什么能在 b.py 中 `import a` 呢？

这是因为这两个文件所在的目录不是一个包，那么每一个 python 文件都是一个独立的、可以直接被其他模块导入的模块，就像你导入标准库一样，它们不存在相对导入和绝对导入的问题。**相对导入与绝对导入仅用于包内部**。

近期文章

- 24 Feb 2019 » 《流浪地球》影评
- 21 Feb 2019 » DOS 命令使用笔记
- 14 Oct 2018 » PYCON中国(2018)听讲笔记