常見 import 陷阱

Trap 1: Circular Import

想像一個 module `A` 在一開始要 import 另一個 module `B` 裡的東西，但在匯入 module `B` 的途中也必須得執行它，而很不巧的 module `B` 也需要從 module `A` import 一些東西。但 module `A` 還正在執行途中。自己都還沒定義好自己的 function 啊！於是你不讓我我不讓你，這種類似 deadlock 的情形正是常見的 **circular import（循環匯入）**。

讓我們看看範例。現在在 `sample_package` 裡有 `A` 和 `B` 兩個 module 想互打招呼，程式碼如下：

`A.py`

```
from .B import B_greet_back


def A_say_hello():
    print('A says hello!')
    B_greet_back()

def A_greet_back():
    print('A says hello back!')


if __name__ == '__main__':
    A_say_hello()
```

`B.py`

```
from .A import A_greet_back


def B_say_hello():
    print('B says hello!')
    A_greet_back()

def B_greet_back():
    print('B says hello back!')


if __name__ == '__main__':
    B_say_hello()
```

內容都一樣，只是 `A`/`B` 互換。`B` 很有禮貌想先打招呼。在與 `sample_package` 同目錄底下執行：

```
$ python3 -m sample_package.B
```

會得到：

```
Traceback (most recent call last):
  File
"/usr/local/Cellar/python3/3.6.2/Frameworks/Python.framework/Versions/3.6/lib/python3.6/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File
"/usr/local/Cellar/python3/3.6.2/Frameworks/Python.framework/Versions/3.6/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/path/to/sample_package/B.py", line 2, in <module>
    from .A import A_greet_back
  File "/path/to/sample_package/A.py", line 1, in <module>
    from .B import B_greet_back
  File "/path/to/sample_package/B.py", line 2, in <module>
    from .A import A_greet_back
ImportError: cannot import name 'A_greet_back'
```

觀察到了嗎？`B` 試圖 import `A_greet_back`，但途中先進到 `A` 執行，而因為 Python 是從頭開始一行一行執行下來的，於是在定義 `A_greet_back` 之前會先碰到自己的 import statement，於是又進入 `B`，然後陷入死胡同。

常見解決這種circular import的方法如下：

1. Import 整個 module 而非單一 attribute

把 `B.py` 更改成如下：

```
# from .A import A_greet_back
from . import A


def B_say_hello():
    print('B says hello!')
    # A_greet_back()
    A.A_greet_back()

...
```

就不會發生錯誤：

```
B says hello!
A says hello back!
```

理由是，原本執行 `from .A import A_greet_back` 時被迫要從 load 進來的 `A` module object 中找出 `A_greet_back` 的定義，但此時這個 module object 還是空的；而更新後的 `from . import A` 就只會檢查 `A` module object 存不存在，至於 `A_greet_back` 存不存在等到需要執行的時候再去找就行了。

2. 延遲 import

把 `B.py` 更改成如下：

```
# 前面全刪

def B_say_hello():
    from .A import A_greet_back

    print('B says hello!')
    A_greet_back()
...
```

也會成功跑出結果。跟前面類似，Python 在跑到這行時才會 import `A` module，這時因為 `B` module 都已經 load 完了，所以不會有 circular import 的問題。但這個方法比較 hacky 一點，大概只能在 hackathon 中使用，否則正式專案裡看到這種難維護的 code 可能會有生命危險。

另一方面，把所有 import statement 擺到整個 module 最後面也是類似效果，但也會被打。

3. 好好釐清架構，避免circular import

是的，治本方法還是好好思考自己寫的 code 為什麼會陷入這種危機，然後重新 refactor 吧。

Trap 2: Relative Import above Top-level Package

還不熟悉 relative import 的人常常會見到這個 error：

```
ValueError: attempted relative import beyond top-level package
```

讓我們重現一下這個 error。把 `B.py` 前頭更改成如下：

```
# from . import A
from ..sample_package import A


...
```

現在我們的路徑位置在與 `sample_package` 同目錄底下。跑：

```
$ python3 -m sample_package.B
```

會得到：

```
Traceback (most recent call last):
  File
"/usr/local/Cellar/python3/3.6.2/Frameworks/Python.framework/Versions/3.6/lib/python3.6/runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File
"/usr/local/Cellar/python3/3.6.2/Frameworks/Python.framework/Versions/3.6/lib/python3.6/runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "/path/to/sample_package/B.py", line 5, in <module>
    from ..sample_package import A
ValueError: attempted relative import beyond top-level package
```

所謂的 `top-level package` 就是你所執行的 package 中最高的那一層，也就是 `sample_package`。超過這一層的 relative import 是不被允許的，指的就是 `..sample_package` 這行嘗試跳一層上去而超過 `sample_package` 了。

可以試試更改當前目錄到上一層（`cd ..`），假設叫 `parent_folder`，然後執行 `python3 -m parent_folder.sample_package.B`，就會發現 error 消失了，因為現在的 `top-level package` 已經變成 `parent_folder` 了。

. . .

結語

Import 是各大語言必備功能，看似簡單，使用上來說陷阱卻頗多。如果搞不清楚 Python 中的 import 是怎麼運作的，除了在整體專案架構上難以審活設計，更可能要陷入可怕的 error 海了。

我寫了一些額外的 sample code 放上 github 了，有不清楚的地方可以直接參考。

. . .

參考資料

- Python Documentation—Modules
- Python Documnetation—the Import System
- tutorialspoint—Python Modules
- PEP328—Imports: Multi-Line and Absolute/Relative
- Importing Python Modules
- Python 101: All about imports