

【Python】解析Python模組(Module)和套件(Package)的概念(轉)

當我們在開發大型應用程式時，如果沒有適當的組織程式碼，除了會降低開發的效率外，也不易於維護，所以模組(Module)化就顯得相當的重要，讓程式碼能夠透過引用的方式來重複使用，提升重用性(Reusable)。

但是隨著專案模組(Module)的增加，將難以管理及問題的追蹤，這時候就能將模組(Module)打包成套件(Package)，利用其階層式的結構來彈性規劃模組(Module)。

本篇文章就帶大家瞭解Python模組(Module)及套件(Package)的重要觀念，包含：

什麼是模組(Module)
模組引用方式(Import)
什麼是套件(Package)
dir()函式(dir function)

將模組當作腳本來執行(Executing a Module as a Script)

一、什麼是模組(Module)

模組(Module)就是一個檔案，包含了相關性較高的程式碼。隨著應用程式的開發規模越來越大，我們不可能把所有的程式碼都寫在同一份Python檔案中，一定會將關聯性較高的程式碼抽出來放在不同的檔案中來形成模組(Module)，主程式再透過引用的方式來使用。所以模組(Module)可以提高程式碼的重用性(Reusable)且易於維護。

假設我們現在要開發一個部落格，主程式為 app.py，在還沒有模組化時，程式碼可能長得像這樣：

```
#取得作者
def get_author():
    return "Mike"
#取得電子郵件
def get_email():
    return "learncodewithmike@gmail.com"
#新增文章
def add_post(title):
    pass
#刪除文章
def delete_post(title):
    pass
add_post()
author = get_author()
email = get_email()
```

以此範例來說，取得作者及電子郵件可以獨立出來建立一個關於模組(about.py)，而新增及刪除文章則可以獨立出來為文章模組(post.py)，專門處理文章相關的動作，如下範例：

about.py

```
#取得作者
def get_author():
    return "Mike"
#取得電子郵件
def get_email():
    return "learncodewithmike@gmail.com"
post.py

#新增文章
def add_post(title):
    pass
#刪除文章
def delete_post(title):
    pass
```

post.py

```
#新增文章
def add_post(title):
    pass
```

```
#刪除文章
def delete_post(title):
    pass
```

當然，模組(Module)除了可以包含函式(Function)外，也可以為類別(Class)，我們以 post.py 為例：

```
class Post:
    # 建構式
    def __init__(self):
        self.titles = []
    # 新增文章
    def add_post(self, title):
        self.titles.append(title)
    # 刪除文章
    def delete_post(self, title):
        self.titles.remove(title)
```

所以現在我們專案中有一個主程式 app.py 及兩個模組(Module)，分別為 about.py 和 post.py。

二、模組引用方式(Import)

我們將程式碼進行模組化後，主程式 app.py 要如何使用呢?首先，可以使用 from-import 語法，如下範例：

```
# 引用模組中的特定物件
from post import Post
from about import get_author, get_email
p = Post()
p.add_post("Python Programming")
author = get_author()
email = get_email()
print(p.titles) #執行結果：['Python Programming']
print(author) #執行結果：Mike
print(email) #執行結果：learncodewithmike@gmail.com
```

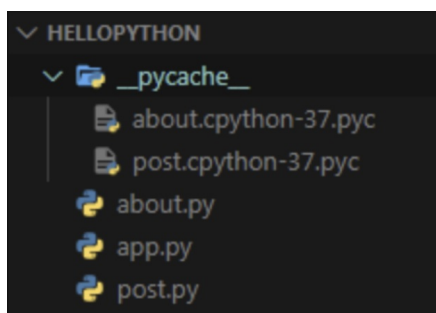
from 之後加上模組(Module)的檔名，注意沒有 .py 副檔名，接著 import 引用所需的物件。

當然，在 import 之後可以使用 * 來引用模組中的所有物件，但是這樣的寫法，可能會在引用的過程中，發生同名方法覆寫(Method Overriding)的風險，所以建議引用所需要的物件即可。另一種語法則是透過 import 語法，如下範例：

```
# 引用整個模組
import post
import about
p = post.Post()
p.add_post("Python Programming")
author = about.get_author()
email = about.get_email()
print(p.titles) #執行結果：['Python Programming']
print(author) #執行結果：Mike
print(email) #執行結果：learncodewithmike@gmail.com
```

import 之後加上模組(Module)的檔名，和上一個語法不一樣的地方是，此語法雖然引用整個模組(Module)，但是在主程式中必須透過模組(Module)的名稱來存取其中的成員。

在主程式 app.py 中引用模組(Module)，並且執行後，會發現多了一個 pycache 資料夾，如下圖：



這個資料夾中，可以看到包含了引用模組的已編譯檔案，當下一次執行主程式 app.py 時，Python編譯器看到已編譯的模組檔案，會直接載入該模組(Module)，而省略編譯的動作，藉此來加速載入模組(Module)的速度。

當然Python編譯器在每一次執行時，會檢查來源模組及已編譯檔案的時間，當來源模組的時間較新，則代表該模組(Module)有經過修改，則Python編譯器會再編譯一次，更新已編譯檔案。

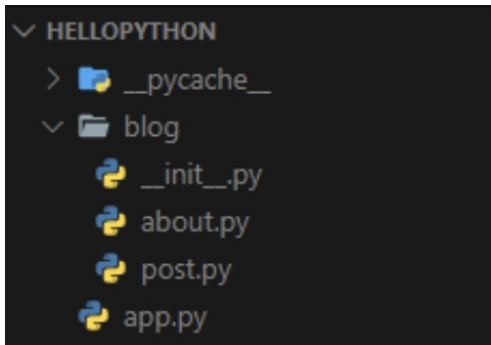
各位有沒有覺得奇怪，那為什麼沒有 app.py 的已編譯檔案，因為在此範例中，我們將 app.py 當作程式的進入點，所以每一次執行 python app.py 指令時，Python編譯器都要進行編譯，所以沒有將 app.py 進行快取的動作。

三、什麼是套件(Package)

就是一個容器(資料夾)，包含了一個或多個的模組(Module)，並且擁有__init__.py檔案，其中可以撰寫套件(Package)初始化的程式碼。

我們將程式碼模組化後，專案中的模組(Module)就會越來越多，這時候就可以再將相似的模組(Module)組織為套件(Package)。那要如何建立套件(Package)呢？

舉例來說，現在我們要將專案中的 post.py 及 about.py 模組(Module)打包為部落格套件(Package)，首先，建立blog資料夾，接著在資料夾中新增__init__.py檔案，最後將 post.py 及 about.py 模組(Module)移至blog資料夾中，如下範例：



而在主程式 app.py 中引用套件(Package)的方式和模組(Module)大同小異，我們先來看 from-import語法，如下範例：

```
# 從套件中引用模組
from blog import post
from blog import about
p = post.Post()
p.add_post("Python Programming")
author = about.get_author()
email = about.get_email()
```

另一個引用套件(Package)的 import 語法如下範例：

```
# 從套件中引用模組
import blog.post
import blog.about
p = blog.post.Post()
p.add_post("Python Programming")
author = blog.about.get_author()
email = blog.about.get_email()
```

四、dir()函式(dir function)

Python提供了一個內建函式dir()，用來顯示物件(Object)的屬性(Attribute)及方法(Method)，我們利用此函式(Function)來看一下模組(Module)所擁有的屬性(Attribute)及方法(Method)，如下範例：

```
#從blog套件引用about模組
from blog import about
print(dir(about))
```

從執行結果可以看到模組(Module)中有自建的get_author及get_email方法(Method)，其餘的則是Python自動幫我們產生的，我們來看幾個常用的屬性(Attribute)，如下範例：

```
# 從blog套件引用about模組
from blog import about
print(about.__name__) # 模組名稱
print(about.__package__) # 套件名稱
print(about.__file__) # 模組的檔名及路徑
```

五、將模組當作腳本來執行(Executing a Module as a Script)

我們來看一個範例，在about模組(Module)中加上以下程式碼，並且執行該模組(Module)，如下範例：

```
#取得作者
def get_author():
    return "Mike"
#取得電子郵件
def get_email():
    return "learncodewithmike@gmail.com"
print("about module name: ", __name__)
# 執行結果：about module name: __main__
```

而這時候換成執行 app.py，__name__屬性(Attribute)則為blog.about，我們就可以利用這個特性，撰寫腳本來彈性的控制當執行模組(Module)的檔案時，要進行哪些行為，而這些行為是在被其他模組(Module)引用時，不會被執行的，如下範例：

```
#取得作者
def get_author():
    return "Mike"
#取得電子郵件
def get_email():
    return "learncodewithmike@gmail.com"
if __name__ == "__main__":
    print("about module initialized.")
    get_author()
```

範例中將about模組(Module)加上了判斷式，當執行about模組(Module)時，__name__屬性(Attribute)為__main__，所以會執行我們設定的任務，而這些任務是在執行主程式 app.py 時，不會被執行的，因為__name__屬性(Attribute)為blog.about。

六、小結

以上就是Python模組(Module)及套件(Package)的重要觀念，除了能夠提高程式碼的重用性(Reusable)外，也有利於未來的單元測試及維護。

☺修訂版本 #4

★由 treeman 建立於 24 🍀🍀@🍀🍀 2021 00:09:25

✍由 treeman 更新於 5 🍀🍀🍀🍀 2023 10:14:59