TpCode

Python杂谈: __init__.py的作用

我们经常在python的模块目录中会看到 "__init__.py" 这个文件，那么它到底有什么作用呢?

1. 标识该目录是一个python的模块包（module package）

如果你是使用python的相关IDE来进行开发，那么如果目录中存在该文件，该目录就会被识别为 **module package** 。

2. 简化模块导入操作

假设我们的模块包的目录结构如下:

```
.
└── mypackage
    ├── subpackage_1
    │   ├── test11.py
    │   └── test12.py
    ├── subpackage_2
    │   ├── test21.py
    │   └── test22.py
    └── subpackage_3
        ├── test31.py
        └── test32.py
```

如果我们使用最直接的导入方式，将整个文件拷贝到工程目录下，然后直接导入:

```
from mypackage.subpackage_1 import test11
from mypackage.subpackage_1 import test12
from mypackage.subpackage_2 import test21
from mypackage.subpackage_2 import test22
from mypackage.subpackage_3 import test31
from mypackage.subpackage_3 import test32
```

当然这个例子里面文件比较少，如果模块比较大，目录比较深的话，可能自己都记不清该如何导入。（很有可能，哪怕只想导入一个模块都要在目录中找很久）

这种情况下，**__init__.py** 就很有作用了。我们先来看看该文件是如何工作的。

2.1 __init__.py 是怎么工作的?

实际上，如果目录中包含了 **__init__.py** 时，当用 import 导入该目录时，会执行 **__init__.py** 里面的代码。

我们在mypackage目录下增加一个 **__init__.py** 文件来做一个实验:

```
.
└── mypackage
    ├── __init__.py
    ├── subpackage_1
    │   ├── test11.py
    │   └── test12.py
    ├── subpackage_2
    │   ├── test21.py
    │   └── test22.py
    └── subpackage_3
        ├── test31.py
        └── test32.py
```

mypackage/__init__.py 里面加一个**print**，如果执行了该文件就会输出：

```
print("You have imported mypackage")
```

下面直接用交互模式进行 **import**

```
>>> import mypackage
You have imported mypackage
```

很显然，**__init__.py** 在包被导入时会被执行。

2.2 控制模块导入

我们再做一个实验，在 **mypackage/__init__.py** 添加以下语句：

```
from subpackage_1 import test11
```

我们导入 mypackage 试试：

```
>>> import mypackage
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/taopeng/Workspace/Test/mypackage/__init__.py", line 2, in <module>
    from subpackage_1 import test11
ImportError: No module named 'subpackage_1'
```

报错了。。。怎么回事?

原来，在我们执行**import**时，当前目录是不会变的（就算是执行子目录的文件），还是需要完整的包名。

```
from mypackage.subpackage_1 import test11
```

综上，我们可以在**__init__.py** 指定默认需要导入的模块

2.3 偷懒的导入方法

有时候我们在做导入时会偷懒，将包中的所有内容导入

```
from mypackage import *
```

这是怎么实现的呢？ **__all__** 变量就是干这个工作的。

__all__ 关联了一个模块列表，当执行 **from xx import *** 时，就会导入列表中的模块。我们将 **__init__.py** 修改为 。

```
__all__ = ['subpackage_1', 'subpackage_2']
```

这里没有包含 **subpackage_3**，是为了证明 **__all__** 起作用了，而不是导入了所有子目录。

```
>>> from mypackage import *
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'subpackage_1', 'subpackage_2']
>>>
>>> dir(subpackage_1)
['__doc__', '__loader__', '__name__', '__package__', '__path__', '__spec__']
```

子目录的中的模块没有导入！！！

该例子中的导入等价于

```
from mypackage import subpackage_1, subpackage_2
```

因此，导入操作会继续查找 subpackage_1 和 subpackage_2 中的 **__init__.py** 并执行。（但是此时不会执行 **import ***）

我们在 subpackage_1 下添加 ***__init__.py*** 文件：

```
__all__ = ['test11', 'test12']

# 默认只导入test11
from mypackage.subpackage_1 import test11
```

再来导入试试

```
>>> from mypackage import *
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'subpackage_1', 'subpackage_2']
>>>
>>> dir(subpackage_1)
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__path__', '__spec__', 'test11']
```

如果想要导入子包的所有模块，则需要更精确指定。

```
>>> from mypackage.subpackage_1 import *
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'test11', 'test12']
```

3. 配置模块的初始化操作

在了解了 __init__.py 的工作原理后，应该能理解该文件就是一个正常的python代码文件。

因此可以将初始化代码放入该文件中。

分类： Python杂谈

标签： python， __init__.py

TpCode
关注 - 0
粉丝 - 2
+加关注

« 上一篇： Python标准库(3.x): 内建函数扫盲
» 下一篇： Python标准库(3.x): itertools库扫盲

posted @ 2018-02-21 15:37 TpCode 阅读(15023) 评论(0) 编辑 收藏