模块化代码编写(下)

到目前为止，我们已经导入过模块，加载了文件．这些都是一般性的模块用法，但是模块导入相关的知识实际要比前面所提到的还要丰富一些．

除了模块名之外，导入也可以指定目录路径．Python代码的目录就称之为包， 因此这类导入就称之为包导入． 事实上包导入是把系统上的目录变成另一个Python命名空间，而属性则对应于目录中所包含的子目录和模块文件．

1．包导入基础

例如：
import dir1.dir2.mod 或 from dir1.dir2.mod import *
这些语句中的点号路径是对于机器上目录层次的路径，通过这个路径可以获得到mod.py文件．这个导入意味着dir1位于某个容器目录中，这个容器目录必须在python的搜索路径中．

注意：
导入语句中的目录路径只能以点号分隔，不能使用windows或linux中的路径描述形式，如/opt/script/document…或c:\windows\program..

1.1.1 __init__.py包文件

如果选择使用包导入，就必须要遵循一条约束．包导入语句的路径中的每个目录内必须有一个名为__init__.py的文件,该文件的内容可以是空，否则导入会失败．

遵循的规则：
1.dir1 dir2中必须有一个__init__.py文件．
2.父目录必须在sys.path也就是python的搜索路径中．

1.1.2 为什么使用包导入．
1) 包导入提供了程序文件的目录信息，因此可以通过该信息轻松的找到文件，从而可以作为组织工具来使用．没有包导入的时候，通常通过查看模块搜索路径才能找出文件．如果根据功能把文件组织成子目录，包导入会让模块扮演的角色更加明显，也使代码更加具有可读性．

例如：
import util
与下面包含路径的导入相比，提供的信息更少：
import sys1.dir1.dir2.util

2．包的相对导入

2.1 第一个包导入的案例
在包自身的内部，包文件的导入可以使用和外部导入相同的方式，但是也可以使用更为特殊一点的包内搜索规则来简化导入．

在包的相对导入过程中大多数新手可能出现以下错误：
SystemError: Parent module '' not loaded, cannot perform relative import
和
ValueError: attempted relative import beyond top-level package

例：

有如下目录结构：

```
[root@gitlab sys1]# tree
.
├── __init__.py
├── mail.py
├── main.py
└── util.py
```

各py文件中的内容为：

1) mail.py文件
```
[root@gitlab sys1]# cat mail.py
def f1():
    print 'mail.py send mail'
```

2)util.py文件
```
[root@gitlab sys1]# cat util.py
def f1():
    print 'sys1 util.py'

from . import mail
mail.f1()
```

3) main.py文件
```
[root@gitlab sys1]# cat main.py
import util
util.f1()
```

当执行main.py文件时，我们将会看到如下的输出(代码报错!)：
```
[root@gitlab sys1]# python main.py
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    import util
  File "/app_shell/tata/mpkg/sys1/util.py", line 5, in <module>
    from . import mail
ValueError: Attempted relative import in non-package
```

2.1.1 报错原因

from . import mail 中的"."意思是是一个相对路径，代表包中的当前目录，在涉及到相对导入时，package所对应的文件夹必须正确的被python解释器视作package，而不是普通文件夹．否则由于不被视作package，无法利用package之间的嵌套关系实现python中包的相对导入．

文件夹被python解释器视作package需要满足两个条件：
1) 文件夹中必须有__init__.py文件，该文件可以为空，但必须存在该文件．
2) 不能作为顶层模块来执行该文件夹中的py文件(即不能作为主函数的入口)

什么是主程序的入口?
你所执行python命令时，你所在的目录就称之为主程序的入口．

2.2.2 解决方法
将main.py文件移动至上一层目录即可．

```
[root@gitlab mpkg]# tree
├── main.py
├── sys1
│   ├── __init__.py
│   ├── mail.py
│   ├── main.py
│   ├── util.py
│   └── util.pyc
```

1) 查看main.py中的内容
```
[root@gitlab mpkg]# cat main.py
import sys1.util
sys1.util.f1()
```

2)执行main.py 可正确得到结果
```
[root@gitlab mpkg]# python main.py
mail.py send mail
sys1 util.py
```

2.2 第二个包导入的案例

有如下目录结构：

```
[root@gitlab mpkg]# tree
.
├── __init__.py
├── main.py
├── sys1
│   ├── __init__.py
│   ├── __init__.pyc
│   ├── mail.py
│   ├── mail.pyc
│   ├── main.py
│   ├── util.py
│   └── util.pyc
└── sys2
    ├── __init__.py
    ├── __init__.pyc
    ├── util.py
    └── util.pyc
```

2.2.1 sys1.util 模块中的内容变更为：

```
[root@gitlab mpkg]# cat sys1/util.py

def f1():
    print 'sys1 util.py'
from . import mail
mail.f1()
from ..sys2 import util
util.f1()
```

sys2.util 模块中的内容为：

```
[root@gitlab mpkg]#  cat sys2/util.py
def f1():
    print 'sys2 util f1'
```

main.py 中的内容为：

```
[root@gitlab mpkg]# cat main.py
import sys1.util
sys1.util.f1()
```

执行代码:(代码报错)

```
[root@gitlab mpkg]# python main.py
mail.py send mail
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    import sys1.util
  File "/app_shell/tata/mpkg/sys1/util.py", line 10, in <module>
    from ..sys2 import util
ValueError: Attempted relative import beyond toplevel package
```

2.2.2 报错原因：

from ..sys2 import util 其中 ..sys2的路径为main.py程序的主入口.

2.2.3 解决方法

将 main.py 移动至上层文件夹与mpkg文件夹同级

```
[root@gitlab tata]# tree
.
├── main.py
└── mpkg
    ├── __init__.py
    ├── main.py
    ├── sys1
    │   ├── __init__.py
    │   ├── __init__.pyc
    │   ├── mail.py
    │   ├── mail.pyc
    │   ├── main.py
    │   ├── util.py
    │   └── util.pyc
    └── sys2
        ├── __init__.py
        ├── __init__.pyc
        ├── util.py
        └── util.pyc
```

此时main.py文件中的内容为：

```
[root@gitlab tata]# cat main.py
import mpkg.sys1.util
mpkg.sys1.util.f1()
```

执行main.py得到如下结果：

```
[root@gitlab tata]# python main.py
mail.py send mail
sys2 util f1
sys1 util.py
```

本节重点：

文件夹被python解释器视作package需要满足两个条件：

1) 文件夹中必须有__init__.py文件, 该文件可以为空, 但必须存在该文件.
2) 不能作为顶层模块来执行该文件夹中的py文件(即不能作为主函数的入口)

In [ ]: