10.1 构建一个模块的层级包¶

问题¶

你想将你的代码组织成由很多分层模块构成的包。

解决方案¶

封装成包是很简单的。在文件系统上组织你的代码，并确保每个目录都定义了一个__init__.py文件。 例如：

```
graphics/
    __init__.py
    primitive/
        __init__.py
        line.py
        fill.py
        text.py
    formats/
        __init__.py
        png.py
        jpg.py
```

一旦你做到了这一点，你应该能够执行各种import语句，如下：

```
import graphics.primitive.line
from graphics.primitive import line
import graphics.formats.jpg as jpg
```

讨论¶

定义模块的层次结构就像在文件系统上建立目录结构一样容易。 文件__init__.py的目的是要包含不同运行级别的包的可选的初始化代码。 举个例子，如果你执行了语句import graphics， 文件graphics/__init__.py将被导入,建立graphics命名空间的内容。像import graphics.format.jpg这样导入，文件graphics/__init__.py和文件graphics/formats/__init__.py将在文件graphics/formats/jpg.py导入之前导入。

绝大部分时候让__init__.py空着就好。但是有些情况下可能包含代码。 举个例子，__init__.py能够用来自动加载子模块:

```
# graphics/formats/__init__.py
from . import jpg
from . import png
```

像这样一个文件,用户可以仅仅通过import grahpics.formats来代替import graphics.formats.jpg以及import graphics.formats.png。

__init__.py的其他常用用法包括将多个文件合并到一个逻辑命名空间，这将在10.4小节讨论。

敏锐的程序员会发现，即使没有__init__.py文件存在，python仍然会导入包。如果你没有定义__init__.py时，实际上创建了一个所谓的“命名空间包”，这将在10.5小节讨论。万物平等，如果你着手创建一个新的包的话，包含一个__init__.py文件吧。