10.3 使用相对路径名导入包中子模块¶

问题¶

将代码组织成包,想用import语句从另一个包名没有硬编码过的包中导入子模块。

解决方案¶

使用包的相对导入，使一个模块导入同一个包的另一个模块 举个例子，假设在你的文件系统上有mypackage包，组织如下：

```
mypackage/
  __init__.py
  A/
    __init__.py
    spam.py
    grok.py
  B/
    __init__.py
    bar.py
```

如果模块mypackage.A.spam要导入同目录下的模块grok，它应该包括的import语句如下：

```
# mypackage/A/spam.py
from . import grok
```

如果模块mypackage.A.spam要导入不同目录下的模块B.bar，它应该使用的import语句如下：

```
# mypackage/A/spam.py
from ..B import bar
```

两个import语句都没包含顶层包名，而是使用了spam.py的相对路径。

讨论¶

在包内，既可以使用相对路径也可以使用绝对路径来导入。 举个例子：

```
# mypackage/A/spam.py
from mypackage.A import grok # OK
from . import grok # OK
import grok # Error (not found)
```

像mypackage.A这样使用绝对路径名的不利之处是这将顶层包名硬编码到你的源码中。如果你想重新组织它，你的代码将更脆，很难工作。 举个例子，如果你改变了包名，你就必须检查所有文件来修正源码。 同样，硬编码的名称会使移动代码变得困难。举个例子，也许有人想安装两个不同版本的软件包，只通过名称区分它们。 如果使用相对导入，那一切都ok，然而使用绝对路径名很可能会出问题。

import语句的 `.` 和 `..` 看起来很滑稽, 但它指定目录名.为当前目录，..B为目录../B。这种语法只适用于import。 举个例子：

```
from . import grok # OK
import grok # ERROR
```

```
import .grok # ERROR
```

尽管使用相对导入看起来像是浏览文件系统，但是不能到定义包的目录之外。也就是说，使用点的这种模式从不是包的目录中导入将会引发错误。

最后，相对导入只适用于在合适的包中的模块。尤其是在顶层的脚本的简单模块中，它们将不起作用。如果包的部分被作为脚本直接执行，那它们将不起作用 例如：

```
% python3 mypackage/A/spam.py # Relative imports fail
```

另一方面，如果你使用Python的-m选项来执行先前的脚本，相对导入将会正确运行。 例如：

```
% python3 -m mypackage.A.spam # Relative imports work
```

更多的包的相对导入的背景知识,请看 PEP 328 .