10.5 利用命名空间导入目录分散的代码¶

问题¶

你可能有大量的代码，由不同的人来分散地维护。每个部分被组织为文件目录，如一个包。然而，你希望能用共同的包前缀将所有组件连接起来，不是将每一个部分作为独立的包来安装。

解决方案¶

从本质上讲，你要定义一个顶级Python包，作为一个大集合分开维护子包的命名空间。这个问题经常出现在大的应用框架中，框架开发者希望鼓励用户发布插件或附加包。

在统一不同的目录里统一相同的命名空间，但是要删去用来将组件联合起来的__init__.py文件。假设你有Python代码的两个不同的目录如下：

```
foo-package/
    spam/
        blah.py

bar-package/
    spam/
        grok.py
```

在这2个目录里，都有着共同的命名空间spam。在任何一个目录里都没有__init__.py文件。

让我们看看，如果将foo-package和bar-package都加到python模块路径并尝试导入会发生什么

```
>>> import sys
>>> sys.path.extend(['foo-package', 'bar-package'])
>>> import spam.blah
>>> import spam.grok
>>>
```

两个不同的包目录被合并到一起，你可以导入spam.blah和spam.grok，并且它们能够工作。

讨论¶

在这里工作的机制被称为“包命名空间”的一个特征。从本质上讲，包命名空间是一种特殊的封装设计，为合并不同的目录的代码到一个共同的命名空间。对于大的框架，这可能是有用的，因为它允许一个框架的部分被单独地安装下载。它也使人们能够轻松地为这样的框架编写第三方附加组件和其他扩展。

包命名空间的关键是确保顶级目录中没有__init__.py文件来作为共同的命名空间。缺失__init__.py文件使得在导入包的时候会发生有趣的事情：这并没有产生错误，解释器创建了一个由所有包含匹配包名的目录组成的列表。特殊的包命名空间模块被创建，只读的目录列表副本被存储在其__path__变量中。 举个例子：

```
>>> import spam
>>> spam.__path__
_NamespacePath(['foo-package/spam', 'bar-package/spam'])
>>>
```

在定位包的子组件时，目录__path__将被用到(例如, 当导入spam.grok或者spam.blah的时候).

包命名空间的一个重要特点是任何人都可以用自己的代码来扩展命名空间。举个例子，假设你自己的代码目录像这样：

```
my-package/
    spam/
        custom.py
```

如果你将你的代码目录和其他包一起添加到sys.path，这将无缝地合并到别的spam包目录中：

```
>>> import spam.custom
>>> import spam.grok
>>> import spam.blah
>>>
```

一个包是否被作为一个包命名空间的主要方法是检查其__file__属性。如果没有，那包是个命名空间。这也可以由其字符表现形式中的“namespace”这个词体现出来。

```
>>> spam.__file__
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute '__file__'
>>> spam
<module 'spam' (namespace)>
>>>
```

更多的包命名空间信息可以查看 PEP 420.