第十二章 模块

什么是模块

模块可能是包含数据成员和方法的类。也可能是一组相关但彼此独立操作的函数。从逻辑上组织代码块。

模块和文件

一个文件可以被看作一个模块。一个模块也可以看做是一个文件。
模块的文件名=模块名+扩展名.py

一个名称空间就是一个从名称到对象的关系映射集合。

Windows下的Python搜索路径在环境变量PATH中。

模块的导入需要一个叫做”路径搜索”的过程。 即在文件系统”预定义区域”中查找 mymodule.py文件(如果你导入 mymodule 的话)。 这些预定义区域只不过是你的 Python 搜索路径的集合。

名称空间

名称空间是名称(标识符)到对象的映射。

在执行期间有两个或三个，分别是局部名称空间、全局名称空间、内建名称空间。局部名称空间在执行期间是不断变化的。

内建名称空间由`__builtins__`模块中的名字构成。
随后加载执行模块的全局名称空间, 它会在模块开始执行后变为**活动名称空间**。

`__builtin__`模块, 该模块包含内建函数, 异常以及其他属性。

在标准Python环境下，`__builtins`包含`__builtin__`中所有的名字。

如果在执行期间调用了一个函数, 那么将创建出第三个名称空间, 即局部名称空间。 我们可以通过`globals()`和`locals()`内建函数判断出某一名字属于哪个名称空间。

名称空间是纯粹意义上的名字和对象间的映射关系。而作用域还指出了从用户代码的哪些物理位置可以访问到这些名字。

局部名称空间和作用域会随函数调用而不断变化, 而全局名称空间是不变的。

当访问一个属性时名称查询的规则:首先从局部名字空间开始，然后全局名称空间，最后内建名称空间。
这也是局部变量作用域会覆盖全局作用域的原因。

导入模块

`import module1[, module2[,... moduleN]]`

在 Python 模块的开头部分导入,最好按照这样的风格:

- Python 标准库模块
- Python 第三方模块
- 应用程序自定义模块

每一类模块之间用空行隔开

解释器执行到这条语句, 如果在搜索路径中找到了指定的模块, 就会加载它。

该过程遵循作用域原则, 如果在一个模块的顶层导入, 那么它的作用域就是全局的;
如果在函数中导入, 那么它的
作用域是局部的。

使用`from-import`语句可以导入指定的模块属性到**当前作用域中。**
`from module import name1[,name2[,... nameN]]`

关于`from-import`的更多信息参考**PEP 328**

当导入模块或者属性名称不是我们喜欢的可以用自己想要的名称替换原始的名称。
类似这样

```
import longmodulename
short = longmodulename
del longmodulename
```

不过这样的方式不在提倡，最好使用一下方式(`as` 关键字):

```
import Tkinter
from cgi import FieldStorage
. . . 可以替换为 . . .
import Tkinter as tk
from cgi import FieldStorage as form
```

更多信息参考**PEP 221**

模块导入的特性

加载模块会导致这个模块被"执行"。 也就是被导入模块的顶层代码将直接被执行。

调用 from-import 可以把名字导入当前的名称空间里去, 这意味着你不需要使用属性/句点属性标识来访问模块的标识符。

只从模块导入名字的另一个副作用是那些名字会成为局部名称空间的一部分。
这可能导致覆盖一个已经存在的具有相同名字的对象。

而且对这些变量的改变只影响它的局部拷贝而不是所导入模块的原始名称空间。
想要改变导入模块的变量值，必须使用完全变量名，即 `module.attribute`

在 **PEP 236** 找到更多关于 `__future__` 的资料。

警告异常类的集合:
`Warning 直接从 Exception 继承, 作为所有`
`警告的基类: UserWarning , DeprecationWarning , SyntaxWarning , 以及 RuntimeWarning 。`

Python 还提供了一个可以操作警告过滤器的 API,可以将警告记录到日志文件中。
PEP 230 获得你的 Python 版本的对应开关选项

2.3之后的版本中，如果在搜索路径中存在一个Python模块的.zip文件，导入时会把ZIP文件当作目录处理。
PE302 PEP 273

导入ZIP归档文件其实是用`import hook`实现的
PEP302

Python 2.3 引入的新导入钩子， 从而简化了导入器。
你只需要编写可调用的 import 类, 然后通过 sys 模块"注册"(或者叫"安装")它。
用到一下几个属性
`sys.path_hooks`
`sys.path_importer_cache`
`sys.meta_path`

模块内建函数

`__import__()`

`__import__()`函数作为实际上导入模块的函数,import语句调用`__import__()`函数完成它的工作。
`__import__() 的语法是:`
`__import__(module_name[, globals[, locals[, fromlist]]])`

globals , locals , 以及 fromlist 参数都是可选的, 默认分别为 globals() , locals() 和[] 。

调用 import sys 语句可以使用下边的语句完成:
sys = **import**('sys')

`globals()` 和 `locals()` 内建函数分别返回调用者全局和局部名称空间的字典。

`reload()`

`reload()` 函数可以重新导入一个已经导入的模块
`reload(sys)`

包

包是一个有层次的文件目录结构, 它定义了一个由模块和子包组成的Python 应用程序执行环境。

使用标准的(import–绝对导入)和(from-import–相对导入) 语句导入包中的模块。

PEP 328中更多绝对导入和相对导入的信息。

模块的其他特性

如果你不想让某个模块属性被 “from module import *” 导入 , 那么你可以给你不想导入的属性名称加上一个下划线(_)。

Python 的模块文件 默认是ASCII

只要在你的 Python 模块头部加入一个额外的编码指示说明就可以让导入者使用指定的编码解析你的模块, 编码对应的 Unicode 字符串。
一个 UTF-8 编码的文件可以这样指示:
`#!/usr/bin/env python`
`# -*- coding: UTF-8 -*-`

执行一个 Python 模块: 通过命令行或 shell , execfile() , 模块导入, 解释器的 -m 选项

相关模块

`imp` - 这个模块提供了一些底层的导入者功能
`modulefinder` - 该模块允许你查找 Python 脚本所使用的所有模块。
`pkgutil` - 该模块提供了多种把 Python 包打包为一个"包"文件分发的方法。
`zipimport` - 你可以使用该模块导入 ZIP 归档文件中的模块。
`distutils` - 该模块提供了对建立、 安装、分发 Python 模块和包的支持。

练习

12-1
搜索路径是查找目录的操作，路径搜索是查找文件的操作
12-1
(1)`import mymodule`和`from mymodule import foo`
(2)这两条语句遵循作用域原则，如果在顶层导入，那么就是全局作用域；如果在函数中导入那么就是局部作用域
12-3
import module在搜索路径中找到就会加载它。依据作用域原则
from module import * 并不是一个良好的风格。

12-4
名称空间是纯粹意义上的名字和对象间的映射关系。
作用域还指出了从用户代码的哪些物理位置可以访问到这些名字。

12-6

```
def importAs():
    import sys as system
    import sys.path as PYTHONPATH
```