第十二章：模块

这章，我只是总结了现阶段我应该去掌握的东西，而没有总结其他可能更重要的东西。留待以后学习。

目录

[1、几个解释 1](#_Toc350599562)

[1）模块 1](#_Toc350599563)

[2）文件 1](#_Toc350599564)

[3）名字空间 1](#_Toc350599565)

[4）名称空间与变量作用域比较 2](#_Toc350599566)

[2、怎么导入模块 2](#_Toc350599567)

[3、包（很重要） 3](#_Toc350599568)

[4、模块的其他特性 5](#_Toc350599569)

现阶段需要掌握的：

## 1、几个解释

### 1）模块

模块支持从逻辑上组织 Python 代码。 当代码量变得相当大的时候, 我们最好把代码分一

些有组织的代码段，前提是保证它们的彼此交互。 这些代码片段相互间有一定的联系, 可能是一个包含数据成员和方法的类, 也可能是一组相关但彼此独立的操作函数。 这些代码段是共享的，所以 Python 允 许 "调入" 一个模块, 允许使用其他模块的属性来利用之前的工作成果, 实现代码重用. 这个把其他模块中属性附加到你的模块中的操作叫做导入(import) 。那些自我包含并且有组织的代码片断就是模块( module )。

### 2）文件

如果说模块是按照逻辑来组织 Python 代码的方法, 那么文件便是物理层上组织模块的方法。因此, 一个文件被看作是一个独立模块, 一个模块也可以被看作是一个文件。 模块的文件名就是模块的名字加上扩展名 .py 。这里我们需要讨论一些关于模块文件结构的问题。 与其它可以导入类(class)的语言不同,在 Python 中你导入的是模块或模块属性。

### 3）名字空间

一个名 称空间就是一个从名称到对象的关系映射集合。 我们已经明确地知道, 模块名称是它们的属性名称中的一个重要部分。 例如 string 模块中的 atoi() 函数就是 string.atoi() 。给定一个模块名之后, 只可能有一个模块被导入到 Python 解释器中, 所以在不同模块间不会出现名称交叉现象; 所以每个模块都定义了它自己的唯一的名称空间。 如果我在我自己的模块 mymodule 里创建了一个 atoi() 函数, 那么它的名字应该是 mymodule.atoi() 。 所以即使属性之间有名称冲突, 但它们的完整授权名称(fully qualified name)——通过句点属性标识指定了各自的名称空间 - 防止了名称冲突的发生。

### 4）名称空间与变量作用域比较

好了, 我们已经知道了什么是名称空间, 那么它与变量作用域有什么关系呢? 它们看起来极其相似。 事实上也确实如此。

## 2、怎么导入模块

1）使用 import 语句导入模块

核心风格: import 语句的模块顺序

我们推荐所有的模块在 Python 模块的开头部分导入。 而且最好按照这样的顺序:

z Python 标准库模块

z Python 第三方模块

z 应用程序自定义模块

然后使用一个空行分割这三类模块的导入语句。 这将确保模块使用固定的习惯导入, 有助于减少每个模块需要的 import 语句数目。

2）from-import 语句

你可以在你的模块里导入指定的模块属性。 也就是把指定名称导入到当前作用域。 使用

from-import 语句可以实现我们的目的, 它的语法是:

from module import name1[, name2[,... nameN]]

3）多行导入

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。 例如, 你需要访问模块 module 中的 var 名字是这样被导入的: from module import var 我们使用单个的 var 就可以访问它自身。 把 var 导入到名称空间后就再没必要引用模块了。当然, 你也可以把指定模块的所有名称导入到当前名称空间里: from module import \*

核心风格: 限制使用 "from module import \*"

在实践中, 我们认为 "from module import \*" 不是良好的编程风格, 因为它"污染"当前名称

空间, 而且很可能覆盖当前名称空间中现有的名字; 但如果某个模块有很多要经常访问的变量或者模块的名字很长, 这也不失为一个方便的好办法。我们只在两种场合下建议使用这样的方法, 一个场合是：目标模块中的属性非常多, 反复键入模块名很不方便, 例如 Tkinter (Python/Tk) 和 NumPy (Numeric Python) 模块, 可能还有 socket 模块。另一个场合是在交互解释器下 , 因为这样可以减少输入次数。

## 3、包（很重要）

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。Python 1.5 加入了包, 用来帮助解决如下问题:

z 为平坦的名称空间加入有层次的组织结构

z 允许程序员把有联系的模块组合到一起

z 允许分发者使用目录结构而不是一大堆混乱的文件

z 帮助解决有冲突的模块名称

与类和模块相同, 包也使用句点属性标识来访问他们的元素。 使用标准的 import 和

from-import 语句导入包中的模块。

1）目录结构

假定我们的包的例子有如下的目录结构:

Phone/

\_\_init\_\_.py

common\_util.py

Voicedta/

\_\_init\_\_.py

Pots.py

Isdn.py

Fax/

\_\_init\_\_.py

G3.py

Mobile/

\_\_init\_\_.py

Analog.py

igital.py

Pager/

\_\_init\_\_.py

Numeric.py

Phone 是最顶层的包, Voicedta 等是它的子包。 我们可以这样导入子包:

import Phone.Mobile.Analog

Phone.Mobile.Analog.dial()

你也可使用 from-import 实现不同需求的导入。

第一种方法是只导入顶层的子包, 然后使用属性/点操作符向下引用子包树:

from Phone import Mobile

Mobile.Analog.dial('555-1212')

此外, 我们可以还引用更多的子包:

from Phone.Mobile import Analog

Analog.dial('555-1212')

事实上, 你可以一直沿子包的树状结构导入:

from Phone.Mobile.Analog import dial

dial('555-1212')

在我们上边的目录结构中, 我们可以发现很多的 \_\_init\_\_.py 文件。 这些是初始化模块, from-import 语句导入子包时需要用到它。 如果没有用到, 他们可以是空文件。 程序员经常忘记为它们的包目录加入 \_\_init\_\_.py 文件, 所以从 Python 2.5 开 始 , 这 将 会导致一个ImportWarning 信息。不过, 除非给解释器传递了 -Wd 选项, 否则它会被简单地忽略。

2）使用 from-import 导入包

包同样支持 from-import all 语句:

from package.module import \*

然而, 这样的语句会导入哪些文件取决于操作系统的文件系统. 所以我们在\_\_init\_\_.py 中加入 \_\_all\_\_ 变量. 该变量包含执行这样的语句时应该导入的模块的名字. 它由一个模块名字符串列表组成.。

3）绝对导入

4）相对导入

## 4、模块的其他特性

1）自动载入的模块

当 Python 解释器在标准模式下启动时, 一些模块会被解释器自动导入, 用于系统相关操作。唯一一个影响你的是 \_\_builtin\_\_ 模块, 它会正常地被载入, 这和 \_\_builtins\_\_ 模块相同。sys.modules 变量包含一个由当前载入(完整且成功导入)到解释器的模块组成 的字典, 模块

名作为键, 它们的位置作为值。例如在 Windows 下, sys.modules 变量包含大量载入的模块, 我们这里截短它, 只提供他们的模块名, 通过调用字典的 keys() 方法:

>>> import sys

>>> sys.modules.keys()

['os.path', 'os', 'exceptions', '\_\_main\_\_', 'ntpath',

'strop', 'nt', 'sys', '\_\_builtin\_\_', 'site',

'signal', 'UserDict', 'string', 'stat']

Unix 下载入的模块很类似:

>>> import sys

>>> sys.modules.keys()

['os.path', 'os', 'readline', 'exceptions',

'\_\_main\_\_', 'posix', 'sys', '\_\_builtin\_\_', 'site',

'signal', 'UserDict', 'posixpath', 'stat']

说明：

1）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。 例如, 你需要访问模块 module 中的 var 名字是这样被导入的: from module import var 我们使用单个的 var 就可以访问它自身。

2）当然, 你也可以把指定模块的所有名称导入到当前名称空间里:

from module import \* 不够我们应该限制使用 "from module import \*"