Python模块化代码的编写(上)

一. 什么是模块?

模块是一种高级的程序组织单元, 将程序代码和数据封装起来以便重用, 从实际的角度看模块往往对应与Python程序文件, 每一个文件都是一个模块, 并且在模块导入之后就可以使用被导入的模块中的变量名, 模块将多个独立的文件链接成了一个更大的程序系统.

二. 为什么使用模块?

1. 代码重用 在模块文件中永久存放代码, 可以在其他任意位置调用和重用.
2. 增加代码的可维护性
3. 系统命名空间的划分

   模块可以理解为是不同变量名的软件包, 可以很好的避免变量名冲突; 如在模块m1.py m2.py文件中都有a变量, 在另一个python文件demo1.py中引入 m1 m2, 使用a变量的方式如下

In [1]:
```
import m1
import m2
print m1.x
print m2.x
```

```
1
2
```

import为文件demo1.py提供了文件m1.py中定义的所有的对象的访问权限. 在本例中demo1.py为顶层文件, m1 m2 为模块文件, 他们的内容都是包含有一些简单语句的文件, 在执行demo1.py的过程中, 模块文件不是直接运行的, 而是被顶层文件导入的, 模块文件中所定义的变量就成了模块的属性, 顶层文件demo1.py使用这些模块中的属性.

4. 实现共享服务和数据 如: 有一个全局的对象, 这个对象会被一个以上的函数或者文件调用, 那么可以将其编写在一个模块中以便被多个客户端导入.

三. 模块的相关语句

模块的相关语句 模块可以由两个语句和一个重要的内置参数来进行处理

1. import

   在.py文件中以整体获取一个模块

2. from

   允许.py文件从模块中获取指定的变量名

四. 模块导入是如何工作的?

1. 什么是Python标准库? Python自带了很多实用的模块, 这些模块被称之为标准库, 它们提供了操作系统的接口如os sys command模块; 文字匹配模块如re; 网络和internet相关模块如socket urllib urllb2等, 这些工具虽然不是Python语言的组成部分, 但是它们却可以在任何安装了Python的环境下使用, 而且在执行Python的绝大多数的平台上都可以使用.
2. import 如何工作? 程序第一次导入指定模块文件时, 会执行三个步骤: 1)找到模块文件 2)编译成位码将.py编译成.pyc 3)执行模块代码来创建其所定义的对象, 加载后的模块可以通过 sys.modules 来查看.
3. 如何找到模块?

   1) 模块搜索

   在执行import时, 我们没有使用import m1.py; 也没有使用import /app_shell/python/m1.py这样的形式; 因为Python使用标准库搜索路径来查找 import 语句所对应的模块文件. 该内容必须要了解, 否则在导入模块的过程中出现如下导入错误时, 不知道从哪里解决.

In [4]:
```
import xyz
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-4-573495e657d7> in <module>()
----> 1 import xyz

ImportError: No module named xyz
```

2) 编译

遍历搜索路径找到符合import语句的源代码文件后, 将其编译成字节码(py-->pyc), Python会检查源代码文件和字节码文件的时间戳, 如果发现字节码的时间戳比源代码的旧, 代表源代码文件被修改过, 于是会重新编译; 反之就会跳过编译的步骤. 在文件导入的过程中会进行编译, 在正常环境执行.py文件是不会生成.pyc文件的, 被导入文件的字节码文件之所以存在是为了以后可以提高导入的速度.

3) 运行

import导入操作最后的步骤就是执行字节码, 字节码中的所有语句会被依次的执行, 从上到下, 任何对于变量的赋值, 都会变成在模块文件中定义的属性, 这个步骤会生成模块代码所定义的所有工具, 因为导入的最后步骤实际就是执行文件的程序代码, 如果在模块文件中做了一些实际的操作如print, 那么在导入的过程中就会被执行, 有可能这并不是我们想要的.

In [2]:
```
import m1
#引入模块, 却将模块中的print语句都执行了一遍.
```

```
this is m1
```

什么是**name**与**main**?

1. 如果一个.py文件是被直接执行的, 那么 **name** 返回 **main**, 该特点可以用在代码测试的过程中.
2. 如果一个.py文件是在引用过程中被执行的, 那么**name**返回模块名.

#cat m1.py

x = 1

print --name--

if '--name--' == '--main--':

```
    print 'this is m1'
```

什么是搜索路径?

能否成功的导入模块, 取决于python能否在搜索路径中找到该模块, 如果找不到则会报错

查找顺序

1. 程序主目录 主目录的概念与Python代码是如何运行的相关
   1) 在运行一个脚本的时候, 它是脚本的当前目录

   2) 如果是在交互模式下 代表的是当前的目录
2. PYTHONPATH目录 通过sys.path可以查看到PYTHONPATH目录, Python会从左到右的搜索sys.path列表中的所有路径.
3. 标准库目录

所有的搜索路径被保存到sys.path中.

In [6]:
```
#示例代码.
import sys
print sys.path
```

```
['', '/app_shell/jupyter/lib64/python27.zip', '/app_shell/jupyter/lib64/python2.7', '/app_shell/jupyter/lib64/python2.7/plat-linux2', '/app_shell/jupyter/lib64/python2.7/lib-tk', '/app_shell/jupyter/lib64/python2.7/lib-old', '/app_shell/jupyter/lib64/python2.7/lib-dynload', '/usr/lib64/python2.7', '/usr/lib/python2.7', '/app_shell/jupyter/lib/python2.7/site-packages', '/app_shell/jupyter/lib/python2.7/site-packages/IPython/extensions', '/root/.ipython']
```

如何配置与修改搜索路径?

如: 从/app_shell/project/util目录下导入util.py模块.

In [8]:
```
proj_path = '/app_shell/project/util'
sys.path.insert(0, proj_path)
print sys.path
```

```
['/app_shell/project/util', '/app_shell/project/util', '', '/app_shell/jupyter/lib64/python27.zip', '/app_shell/jupyter/lib64/python2.7', '/app_shell/jupyter/lib64/python2.7/plat-linux2', '/app_shell/jupyter/lib64/python2.7/lib-tk', '/app_shell/jupyter/lib64/python2.7/lib-old', '/app_shell/jupyter/lib64/python2.7/lib-dynload', '/usr/lib64/python2.7', '/usr/lib/python2.7', '/app_shell/jupyter/lib/python2.7/site-packages', '/app_shell/jupyter/lib/python2.7/site-packages/IPython/extensions', '/root/.ipython']
```

import引用与from...import...引用的区别.

1. import引用的是整个模块对象, 所以需要通过模块名得到该模块的相应属性.
2. from会把变量名复制到另一个作用域, 允许我们在脚本中直接使用该变量.

In [9]:
```
#示例代码.
import m1
print x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-fc3f6a411913> in <module>()
      1 #示例代码.
      2 import m1
----> 3 print x

NameError: name 'x' is not defined
```

In [10]:
```
print m1.x
#使用import方式引入模块, 在使用模块中的属性时, 语法为 模块名.属性名.
```

```
1
```

In [11]:
```
from m2 import *
print x
# 使用from形式引入模块, 被引入的模块中的属性, 可以直接使用.
```

```
2
```

from语句是将模块中导入的变量名复制到了当前的作用域中, 所以在使用的过程中不需要通过 模块.属性, 这样的方式调用; 把一个模块内的所有内容导入到当前的文件中使用 from xx import *, 相对于import, 通过from的方式可以在调用变量的过程中少输入一个模块的名字. 使用import的方式导入, 可以确保变量名不冲突, 因为在不同的模块中可能都存在变量a, 但是使用不同模块的变量a时, 语法为 模块名.变量名, 可避免冲突.