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

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

3.增加代码的可维护性

2.系统命名空间的划分

模块可以理解为是变量名的软件包，可以很好的避免变量名冲突

如 m1 m2 python 文件中都有 a 变量

在一个 python 文件 demo1.py 中引入 m1 m2 使用 a 变量的方式

```
import m1

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

import m2

print m1.a

print m2.a
```

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， 顶层文件使用这些模块中的属性

3.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

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

import

在.py 文件中以整体获取一个模块

from

允许.py 文件从模块中获取指定的变量名

imp.reload

在不终止 python 程序的情况下，重新载入模块代码

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 这些标准库模块提供了 操作系统的接口 如 os sys commands 模块；文字匹配模块 如 re ；网络和 internet 相关 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

import 如何工作

程序第一次导入指定文件时 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

1.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

```
Traceback (most recent call last):
  File "demo1.py", line 2, in <module>
    import c
ImportError: No module named c
```

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

python 会检查源代码文件和字节码文件的时间戳

1）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

2）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，这是由于 python 丢弃那些直接执行的 python 文件的字节码，被导入文件的字节码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 都会得到模块文件的属性 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

理解__name__ 与 __main__

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

如：

```
if __name__ == '__main__':
    some   expression
```

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

4.任何.pth 结尾的文件中 #没用过

综上所有的搜索路径被保存到 sys.path 中

配置与修改搜索路径

例： 从 util 目录下导入 util.py 模块

os.path.abspath(__name__)

os.path.dirname()

import 与 from 语句的区别

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

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

例:

util.py

```
def f1(x):
    print x
```

demo1.py

```
import util
util.f1('123')
from util import f1
f1('xyz')
```

##from 语句是将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

把一个模块内的内容导入到当前的文件中使用 from xx import * (*代表所有)， 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

重载模块

reload

模块程序代码默认对每个过程只执行一次 要强制模块代码重新载入并运行 需要手工操作

1. 导入模块发生编译行为只在第一次执行导入时才进行该操作

2.之后只会使用已加载的模块对象 而不会重载或重新执行文件的代码

3.reload 函数会强制已加载的模块的代码重新载入并重新执行

例:

touch change.py

x = ‘first change’

>>> import change

>>> change.x

'first change'

>>> 不关闭 python 交互环境 修改 change

>>> change.x

'first change' ##发现未发生改变

>>> from imp import reload   #重载之后

>>> reload(change)

<module 'change' from '/app_shell/python/project/change.py'>

>>> change.x   ##值发生了改变

'second change'