

第七章 模块(module)

1. 模块化(module)程序设计理念

1.1 模块和包概念的进化史

“量变引起质变”是哲学中一个重要的理论。量变为什么会引起质变呢？本质上理解，随着数量的增加，管理方式会发生本质的变化；旧的管理方式完全不适合，必须采用新的管理方式。

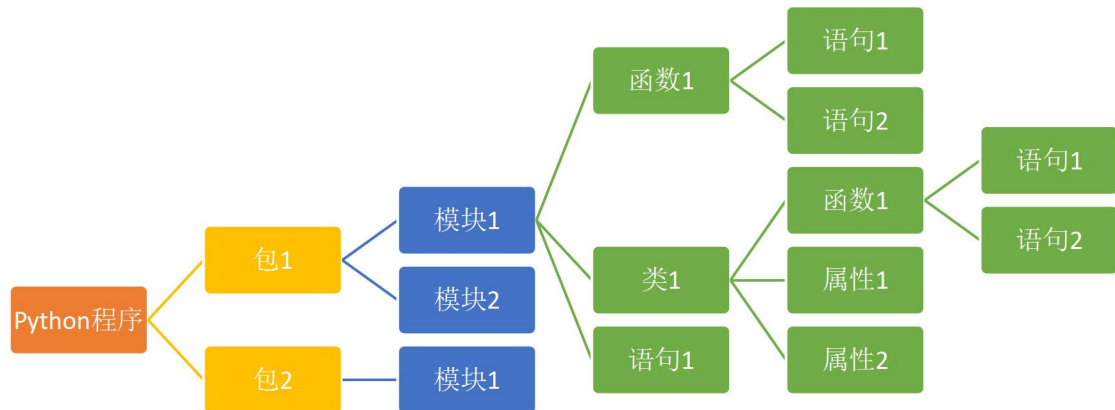
程序越来越复杂，语句多了，怎么管理？很自然的，我们会将实现同一个功能的语句封装到函数中，统一管理和调用，于是函数诞生了。

程序更加复杂，函数和变量多了，怎么管理？同样的思路，“物以类聚”，我们将同一类型对象的“数据和行为”，也就是“变量和函数”，放到一起统一管理和调用，于是“类和对象”诞生了。

程序继续复杂，函数和类更加多了，怎么办？好，我们将实现类似功能的函数和类统统放到一个模块中，于是“模块”诞生了。

程序还要复杂，模块多了，怎么办？于是，我们将实现类似功能的模块放到一起，于是“包”就诞生了。

大家可以清晰的看到这发展的流程，核心的哲学思想就是“量变引起质变”、“物以类聚”。同样的思路，在企业管理、人的管理中思路完全一致。大家可以举一反三。



1. Python 程序由模块组成。一个模块对应 python 源文件，一般后缀名是：.py。
2. 模块由语句组成。运行 Python 程序时，按照模块中语句的顺序依次执行。
3. 语句是 Python 程序的构造单元，用于创建对象、变量赋值、调用函数、控制语句等。

1.2 标准库模块(standard library)

与函数类似，模块也分为标准库模块和用户自定义模块。

Python 标准库提供了操作系统功能、网络通信、文本处理、文件处理、数学运算等基本功能。比如：random(随机数)、math(数学运算)、time(时间处理)、file(文件处理)、os(和操作系统交互)、sys(和解释器交互)等。

另外，Python 还提供了海量的第三方模块，使用方式和标准库类似。功能覆盖了我们能想象到的所有领域，比如：科学计算、WEB 开发、大数据、人工智能、图形系统等。

1.3 为什么需要模块化编程

模块(module)对应于 Python 源代码文件(.py 文件)。模块中可以定义变量、函数、类、普通语句。这样，我们可以将一个 Python 程序分解成多个模块，便于后期的重复应用。

模块化编程 (Modular Programming) 将一个任务分解成多个模块。每个模块

就像一个积木一样，便于后期的反复使用、反复搭建。



模块化编程有如下几个重要优势：

1. 便于将一个任务分解成多个模块，实现团队协作开发，完成大规模程序
2. 实现代码复用。一个模块实现后，可以被反复调用。
3. 可维护性增强。

1.4 模块化编程的流程

模块化编程的一般流程：

1. 设计 API，进行功能描述。
2. 编码实现 API 中描述的功能。
3. 在模块中编写测试代码，并消除全局代码。
4. 使用私有函数实现不被外部客户端调用的模块函数。

1.5 模块的 API 和功能描述要点

API(Application Programming Interface 应用程序编程接口)是用于描述模块中提供的函数和类的功能描述和使用方式描述。

模块化编程中，首先设计的就是模块的 API（即要实现的功能描述），然后开始编码实现 API 中描述的功能。最后，在其他模块中导入本模块进行调用。

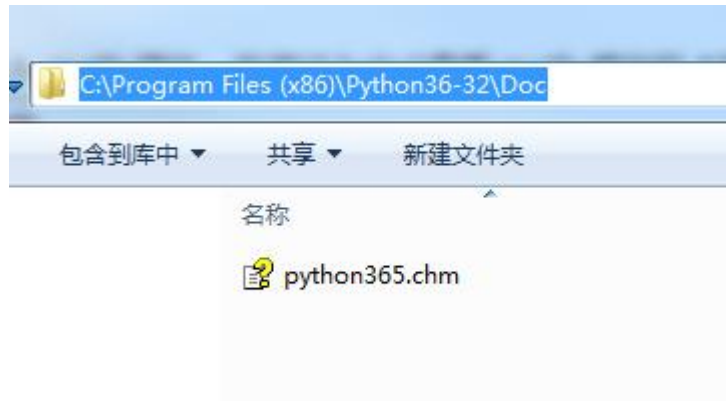
我们可以通过 help(模块名)查看模块的 API。一般使用时先导入模块 然后通过 help

函数查看。

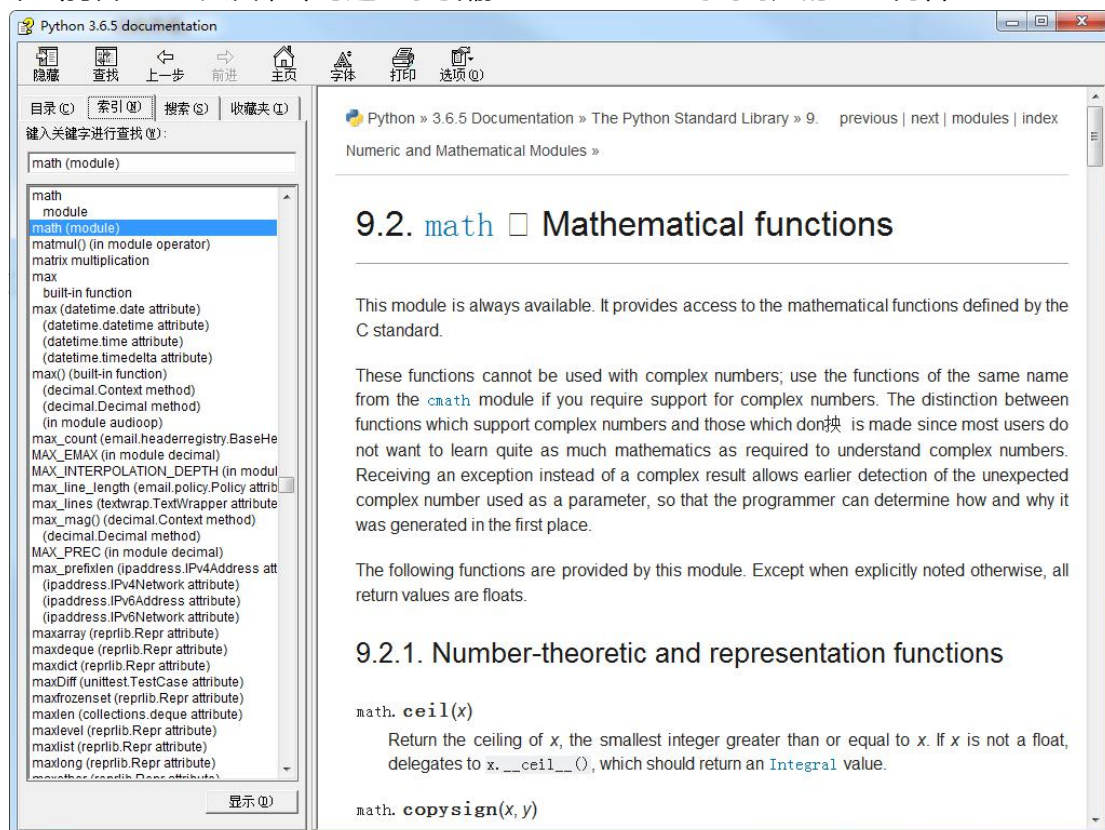
【示例】导入 math 模块，并通过 help() 查看 math 模块的 API：

```
import math
help(math)
```

也可以在 python 的 api 文档中查询。首先进入 python 的安装目录下的 docs 子目录：



双击打开 chm 文档，即可通过索引输入“math”查询到对应的 API 内容：



【示例】设计计算薪水模块的 API

```
"""
本模块用于计算公司员工的薪资
"""

company = "北京尚学堂"

def yearSalary(monthSalary):
    """根据传入的月薪，计算出年薪"""
    pass

def daySalary(monthSalary):
    """根据传入的月薪，计算出每天的薪资"""
    pass
```

如上模块只有功能描述和规范，需要编码人员按照要求实现编码。

我们可以通过__doc__可以获得模块的文档字符串的内容。

test.py 的源代码：

```
import salary

print(salary.__doc__)
print(salary.yearSalary.__doc__)
```

运行结果：

```
本模块用于计算公司员工的薪资
根据传入的月薪，计算出年薪
```

1.6 模块的创建和测试代码

每个模块都有一个名称，通过特殊变量__name__可以获取模块的名称。在正常情况

下，模块名字对应源文件名。 仅有一个例外，就是当一个模块被作为程序入口时（主程序、交互式提示符下），它的__name__的值为“__main__”。我们可以根据这个特点，将模块源代码文件中的测试代码进行独立的处理。例如：

```
import math
math.__name__    #输出'math'
```

【示例】通过__name__ == “__main__” 独立处理模块的测试代码

```
"""
本模块用于计算公司员工的薪资
"""

company = "北京尚学堂"

def yearSalary(monthSalary):
    """根据传入的月薪，计算出年薪"""
    return monthSalary*12

def daySalary(monthSalary):
    """根据传入的月薪，计算出每天的薪资"""
    return monthSalary/22.5    #国家规定每个月的平均工作日是 22.5

if __name__ == "__main__":    #测试代码
    print(yearSalary(3000))
    print(daySalary(3000))
```

1.7 模块文档字符串和 API 设计

我们可以在模块的第一行增加一个文档字符串，用于描述模块的相关功能。然后，通过__doc__可以获得文档字符串的内容。

【示例】模块文档字符串示例以及导入后如何读取文档字符串

test.py 的源代码：

```
import MySalary

print(MySalary.__doc__)
print(MySalary.yearSalary.__doc__)
```

运行结果：

本模块实现根据月薪计算各种薪水的功能
根据月薪，计算年薪

2. 模块的导入

模块化设计的好处之一就是“代码复用性高”。写好的模块可以被反复调用，重复使用。模块的导入就是“在本模块中使用其他模块”。

2.1 import 语句导入

import 语句的基本语法格式如下：

import 模块名	#导入一个模块
import 模块 1, 模块 2...	#导入多个模块
import 模块名 as 模块别名	#导入模块并使用新名字

import 加载的模块分为四个通用类别：

- 使用 python 编写的代码（.py 文件）；
- 已被编译为共享库或 DLL 的 C 或 C++ 扩展；
- 包好一组模块的包
- 使用 C 编写并链接到 python 解释器的内置模块；

我们一般通过 import 语句实现模块的导入和使用，import 本质上是使用了内置函数 `__import__()`。

当我们通过 import 导入一个模块时，python 解释器进行执行，最终会生成一个对象，这个对象就代表了被加载的模块。

```
import math

print(id(math))
print(type(math))
print(math.pi)    #通过 math.成员名来访问模块中的成员
```

执行结果是：

31840800

<class 'module'>

由上，我们可以看到 math 模块被加载后，实际会生成一个 module 类的对象，该对象被 math 变量引用。我们可以通过 math 变量引用模块中所有的内容。

我们通过 import 导入多个模块，本质上也是生成多个 module 类的对象而已。

有时候，我们也需要给模块起个别名，本质上，这个别名仅仅是新创建一个变量引用加载的模块对象而已。

```
import math as m

#import math
#m = math

print(m.sqrt(4)) #开方运算
```

2.2 from...import 导入

Python 中可以使用 from...import 导入模块中的成员。基本语法格式如下：

```
from 模块名 import 成员 1, 成员 2, ...
```


如果希望导入一个模块中的所有成员，则可以采用如下方式：

```
from 模块名 import *
```

【注】尽量避免“from 模块名 import *”这种写法。* 它表示导入模块中所有的不是以下划线(_)开头的名字都导入到当前位置。但你不知道你导入什么名字，很有可能会覆盖掉你之前已经定义的名字。而且可读性极其的差。一般生产环境中尽量避免使用，学习时没有关系。

【示例】使用 from...import 导入模块指定的成员

```
from math import pi,sin

print(sin(pi/2))    #输出 1.0
```

2.3 import 语句和 from...import 语句的区别

import 导入的是模块。from...import 导入的是模块中的一个函数/一个类。

如果进行类比的话，import 导入的是“文件”，我们要使用该“文件”下的内容，必须前面加“文件名称”。from...import 导入的是文件下的“内容”，我们直接使用这些“内容”即可，前面再也不需要加“文件名称”了。

我们自定义一个模块 calculator.py：

```
"""一个实现四则运算的计算器"""

def add(a,b):
    return a+b

def minus(a,b):
    return a-b
```

```
class MyNum():  
    def print123(self):  
        print(123)
```

我们在另一个模块 test.py 测试：

```
import calculator  
  
a = calculator.add(30,40)  
# add(100,200)      #不加模块名无法识别  
print(a)  
  
from calculator import *  
  
a = add(100,200)      #无需模块名，可以直接引用里面的函数/类  
print(a)  
  
b = MyNum()  
b.print123()
```

2.4 __import__()动态导入

import 语句本质上就是调用内置函数__import__()，我们可以通过它实现动态导入。给__import__()动态传递不同的参数值，就能导入不同的模块。

【示例】使用__import__()动态导入指定的模块

```
s = "math"  
  
m = __import__(s)      #导入后生成的模块对象的引用给变量 m  
  
print(m.pi)
```

注意：一般不建议我们自行使用 `__import__()` 导入，其行为在 python2 和 python3 中有差异，会导致意外错误。如果需要动态导入可以使用 `importlib` 模块。

```
import importlib

a = importlib.import_module("math")

print(a.pi)
```

2.5 模块的加载问题

当导入一个模块时，模块中的代码都会被执行。不过，如果再次导入这个模块，则不会再次执行。

Python 的设计者为什么这么设计？因为，导入模块更多的时候需要的是定义模块中的变量、函数、对象等。这些并不需要反复定义和执行。“只导入一次 import-only-once” 就成了一种优化。

一个模块无论导入多少次，这个模块在整个解释器进程内有且仅有一个实例对象。

test02.py 的源代码：

```
print("test 模块被加载了...")
```

test03.py 的源代码：

```
import test02    #会执行 test02 模块中的语句
import test02    #不会再执行 test02 模块中的语句
```

• 重新加载

有时候我们确实需要重新加载一个模块，这时候可以使用 `:importlib.reload()`

方法：

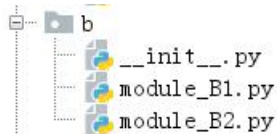
```
import test02
import test02

print("####")
import importlib
importlib.reload(test02)
```

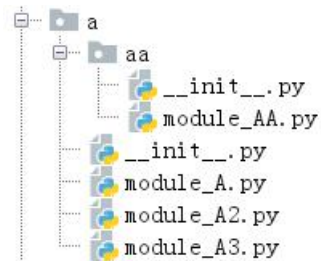
3. 包 package 的使用

3.1 包(package)的概念和结构

当一个项目中有很多个模块时,需要再进行组织。我们将功能类似的模块放到一起,形成了“包”。本质上,“包”就是一个必须有__init__.py 的文件夹。典型结构如下:



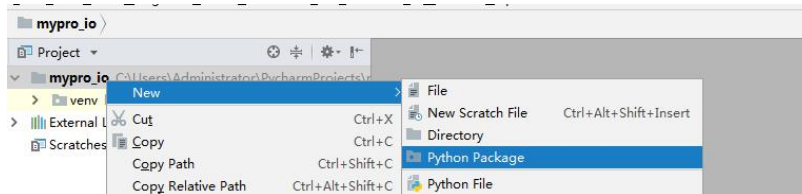
包下面可以包含“模块(module)”,也可以再包含“子包(subpackage)”。就像文件夹下面可以有文件,也可以有子文件夹一样。



上图中, a 是上层的包,下面有一个子包: aa。可以看到每个包里面都有__init__.py 文件。

3.2 pycharm 中创建包

在 pycharm 开发环境中创建包，非常简单。在要创建包的地方单击右键：New-->Python package 即可。pycharm 会自动帮助我们生成带有__init__.py 文件的包。



3.3 导入包操作和本质

上一节中的包结构，我们需要导入 module_AA.py。方式如下：

1. `import a.aa.module_AA`
在使用时，必须加完整名称来引用，比如：`a.aa.module_AA.fun_AA()`
2. `from a.aa import module_AA`
在使用时，直接可以使用模块名。 比如：`module_AA.fun_AA()`
3. `from a.aa.module_AA import fun_AA` 直接导入函数
在使用时，直接可以使用函数名。 比如：`fun_AA()`

【注】

1. `from package import item` 这种语法中，item 可以是包、模块，也可以是函数、类、变量。
2. `import item1.item2` 这种语法中，item 必须是包或模块，不能是其他。

导入包的本质其实是“导入了包的__init__.py”文件。也就是说，“import pack1”意味着执行了包 pack1 下面的__init__.py 文件。这样，可以在__init__.py 中批量导入我们需要的模块，而不再需要一个个导入。

__init__.py 的三个核心作用：

1. 作为包的标识，不能删除。
2. 用来实现模糊导入
3. 导入包实质是执行__init__.py 文件，可以在__init__.py 文件中做这个包的初始化、以及需要统一执行代码、批量导入。

【示例】测试包的__init__.py 文件本质用法

a 包下的__init__.py 文件内容：

```
import turtle
import math

print("导入 a 包")
```

b 包下的 module_B1.py 文件中导入 a 包，代码如下：

```
import a
print(a.math.pi)
```

执行结果如下：

```
导入 a 包
3.141592653589793
```

【注】如上测试我们可以看出 python 的设计者非常巧妙的通过__init__.py 文件将包转成了模块的操作。因此，可以说“包的本质还是模块”。

3.4 用*导入包

`import *` 这样的语句理论上是希望文件系统找出包中所有的子模块，然后导入它们。这可能会花长时间等。Python 解决方案是提供一个明确的包索引。

这个索引由 `__init__.py` 定义 `__all__` 变量，该变量为一列表，如上例 `a` 包下的 `__init__.py` 中，可定义 `__all__ = ["module_A", "module_A2"]`

这意味着，`from sound.effects import *` 会从对应的包中导入以上两个子模块；

【注】尽管提供 `import *` 的方法，仍不建议在生产代码中使用这种写法。

3.5 包内引用

如果是子包内的引用，可以按相对位置引入子模块。以 `aa` 包下的 `module_AA` 中导入 `a` 包下内容为例：

```
from .. import module_A    #..表示上级目录    .表示同级目录
```

```
from . import module_A2    #.表示同级目录
```

3.6 sys.path 和模块搜索路径

当我们导入某个模块文件时，Python 解释器去哪里找这个文件呢？只有找到这个文件才能读取、装载运行该模块文件。它一般按照如下路径寻找模块文件（按照顺序寻找，找到即停不继续往下寻找）：

1. 内置模块
2. 当前目录
3. 程序的主目录
4. `pythonpath` 目录（如果已经设置了 `pythonpath` 环境变量）
5. 标准链接库目录
6. 第三方库目录（`site-packages` 目录）
7. `.pth` 文件的内容（如果存在的话）
8. `sys.path.append()` 临时添加的目录

当任何一个 python 程序启动时 就将上面这些搜索路径(除内置模块以外的路径)进行收集，放到 sys 模块的 path 属性中（sys.path）。

• 使用 sys.path 查看和临时修改搜索路径

我们在项目的 b 目录下建立测试模块：

```
import sys
sys.path.append("d:/")
print(sys.path)
```

执行结果：

```
[ 'C:\\Users\\Administrator\\PycharmProjects\\mypro-modules\\b',
'C:\\Users\\Administrator\\PycharmProjects\\mypro-modules',
'e:\\', 'f:\\abc' ,
'C:\\Users\\Administrator\\PycharmProjects\\mypro-modules\\venv\\Scripts\\python36.zip',
'C:\\Program Files (x86)\\Python36-32\\DLLs',
'C:\\Program Files (x86)\\Python36-32\\lib',
'C:\\Program Files (x86)\\Python36-32',
'C:\\Program Files (x86)\\Python36-32\\lib\\site-packages',
'g:\\a', 'g:\\b', 'g:\\c',
'd:/']
```

模块当前目录

程序的主目录

pythonpath 中添加的目录

标准链接库的目录

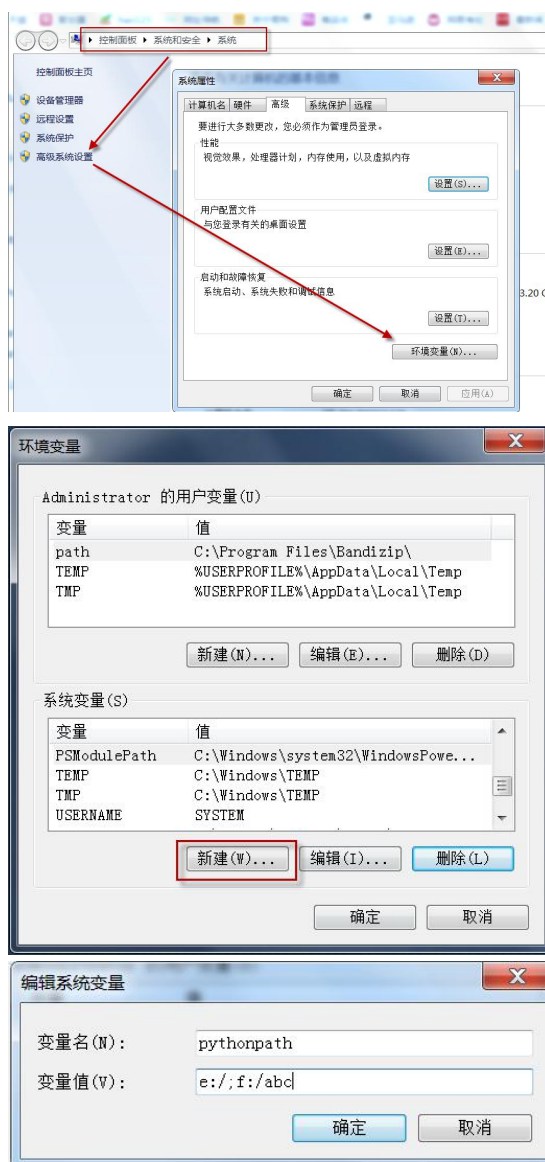
第三方库目录(如果 pycharm 项目建立了 venv 虚拟环境，则是 venv 目录)

.pth 文件中列出的目录

sys.path.append()临时添加的目录

• pythonpath 环境变量的设置

windows 系统中通过如下操作添加和设置 pythonpath 环境变量。



• .pth 文件的写法

我们可以在 site-packages 目录下添加.pth 文件。并在文件中增加内容：

```
#一行一个目录
g:\a
g:\b
g:\c
```

【注】

1. 需确保 g:\a,g:\b,g:\c 对应的目录真实存在。
2. 在 windows 系统中建立.pth 文件，由于没有文件名不能直接建立。需要输入：

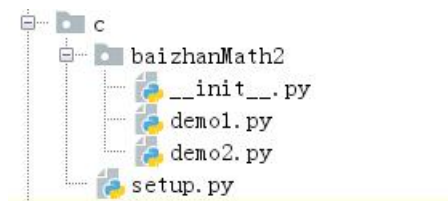
“.pth.” 才能正常建立.pth 文件。

4. 模块发布和安装

4.1 模块的本地发布

当我们完成了某个模块开发后，可以将他对外发布，其他开发者也可以以“第三方扩展库”的方式使用我们的模块。我们按照如下步骤即可实现模块的发布：

1.为模块文件创建如下结构的文件夹（一般，文件夹的名字和模块的名字一样）：



2.在文件夹中创建一个名为『setup.py』的文件，内容如下：

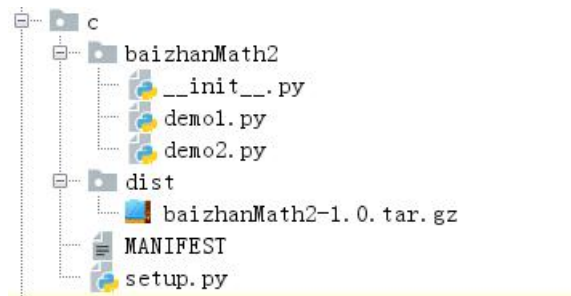
```
from distutils.core import setup

setup(
    name='baizhanMath2', # 对外我们模块的名字
    version='1.0', # 版本号
    description='这是第一个对外发布的模块，测试哦', #描述
    author='gaoqi', # 作者
    author_email='gaoqi110@163.com',
    py_modules=['baizhanMath2.demo1', 'baizhanMath2.demo2'] # 要发布的模块
)
```

3. 构建一个发布文件。通过终端，cd 到模块文件夹 c 下面，再键入命令：

```
python setup.py sdist
```

执行完毕后，目录结构变为：

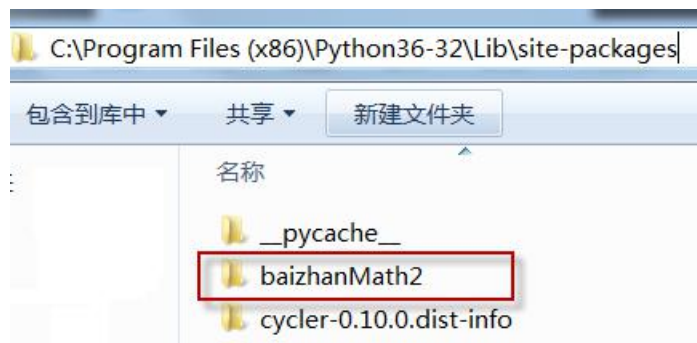


4.2 本地安装模块

将发布安装到你的本地计算机上。仍在 cmd 命令行模式下操作，进 setup.py 所在目录，键入命令：

```
python setup.py install
```

安装成功后，我们进入 python 目录/Lib/site-packages 目录（第三方模块都安装在这里，python 解释器执行时也会搜索这个路径）：



安装成功后，直接使用 import 导入即可。

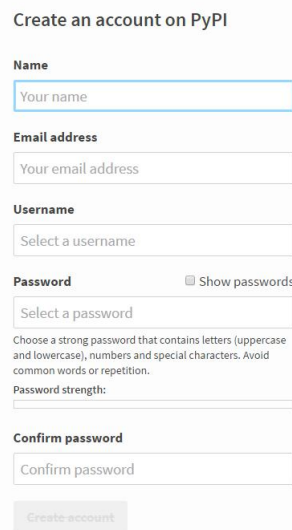
```
import baizhanMath2.demo1
```

4.3 上传模块到 PyPI

将自己开发好的模块上传到 PyPI 网站上，将成为公开的资源，可以让全球用户自由使用。按照如下步骤做，很容易就实现上传模块操作。

•注册 PyPI 网站

注册 PyPI 网站 : <http://pypi.python.org>



The image shows the 'Create an account on PyPI' form. It includes fields for Name, Email address, Username, Password, and Confirm password. There is a 'Show passwords' checkbox and a 'Create account' button at the bottom.

Create an account on PyPI

Name
Your name

Email address
Your email address

Username
Select a username

Password ☐ Show passwords
Select a password
Choose a strong password that contains letters (uppercase and lowercase), numbers and special characters. Avoid common words or repetition.
Password strength:

Confirm password
Confirm password

Create account

【注意】会发送一封邮件到你的邮箱。请点击验证后继续下面的步骤。

•创建用户信息文件.pypirc

·方式 1：使用命令(适用 Linux)

输入并执行后 `python setup.py register`，然后输入用户名和密码，即可。

·方式 2：使用文件（适用 windows,Linux）

在用户的家目录里创建一个文件名为.pypirc, 内容为：

```
[distutils]
index-servers=pypi

[pypi]
repository = https://upload.pypi.org/legacy/
username = 账户名
password = 你自己的密码
```

【注】

Linux 的家目录： `~/.pypirc`

Windows 的家目录是： `c:/user/用户名`

在 windows 下直接创建不包含文件名的文件会失败，因此创建时文件名为 `".pypirc"`，前后都有两个点即可。

•上传并远程发布

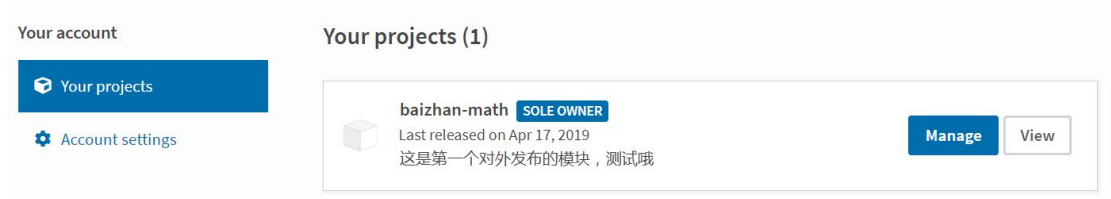
进入 setup.py 文件所在目录，使用命令 “python setup.py sdist upload”，即将模块代码上传并发布：

```
creating baizhan-math-1.0
creating baizhan-math-1.0\baizhan-math
making hard links in baizhan-math-1.0...
hard linking setup.py -> baizhan-math-1.0
hard linking baizhan-math\demo.py -> baizhan-math-1.0\baizhan-math
Creating tar archive
removing 'baizhan-math-1.0' (and everything under it)
running upload
Submitting dist\baizhan-math-1.0.tar.gz to https://upload.pypi.org/legacy/
Server response (200): OK
```

•管理你的模块

我们登录 pypi 官网，可以看到：

如果你的模块已经上传成功，那么当你登录 PyPI 网站后应该能在右侧导航栏看到管理入口。



点击包名进去后你可以对你的模块进行管理，当然你也可以从这里删除这个模块。

4.4 让别人使用你的模块

模块发布完成后，其他人只需要使用 pip 就可以安装你的模块文件。比如：

pip install package-name

```
C:\Users\Administrator\PycharmProjects\mypro_modules\baizhan>pip install baizhan-math
Collecting baizhan-math
  Downloading https://files.pythonhosted.org/packages/20/37/4aeb3c8c13699982d500a6b87c40069eabb78baecd5c77f69e40f0f9a54/baizhan-math-1.0.tar.gz
Installing collected packages: baizhan-math
  Running setup.py install for baizhan-math ... done
Successfully installed baizhan-math-1.0
```

如果你更新了模块，别人可以通过 `--update` 参数来更新：

pip install package-name update

```
C:\Users\Administrator\PycharmProjects\mypro_modules\baizhan>pip install baizh
-math update
Requirement already satisfied: baizhan-math in c:\program files (x86)\python36
2\lib\site-packages (1.0)
Collecting update
  Downloading https://files.pythonhosted.org/packages/9f/c4/dfe8a392edd35cc635
5cd3b20df6a746aacdeb39b685d1668b56bf819b/update-0.0.1-py2.py3-none-any.whl
Collecting style==1.1.0 (from update)
  Downloading https://files.pythonhosted.org/packages/4c/0b/6be2071e20c621e7be
1b86e8474c2ec344a9750ba5315886f24d6e7386/style-1.1.0-py2.py3-none-any.whl
Installing collected packages: style, update
Successfully installed style-1.1.0 update-0.0.1
```

5. 库(Library)

Python 中库是借用其他编程语言的概念，没有特别具体的定义。模块和包侧重于代码组织，有明确的定义。

一般情况，库强调的是功能性，而不是代码组织。我们通常将某个功能的“模块的集合”，称为库。

5.1 标准库(Standard Library)

Python 拥有一个强大的标准库。Python 语言的核心只包含数字、字符串、列表、字典、文件等常见类型和函数，而由 Python 标准库提供了系统管理、网络通信、文本处理、数据库接口、图形系统、XML 处理等额外的功能。

Python 标准库的主要功能有：

1. 文本处理，包含文本格式化、正则表达式匹配、文本差异计算与合并、Unicode 支持，二进制数据处理等功能
2. 文件处理，包含文件操作、创建临时文件、文件压缩与归档、操作配置文件等功能

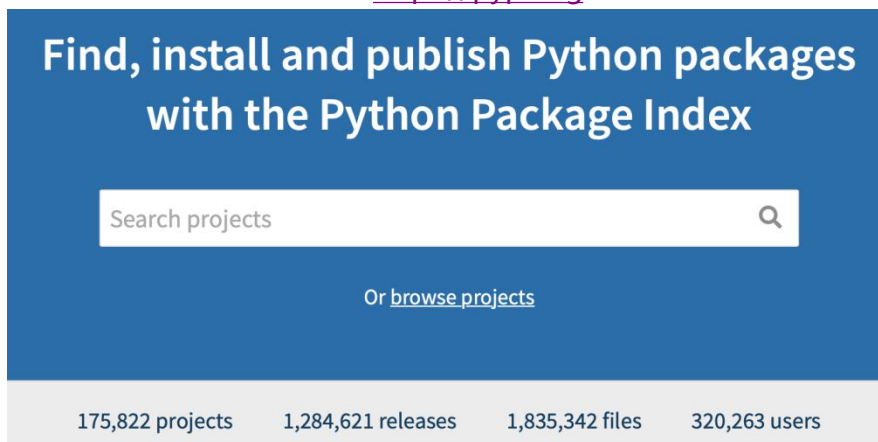
3. 操作系统功能，包含线程与进程支持、IO 复用、日期与时间处理、调用系统函数、日志 (logging) 等功能
4. 网络通信，包含网络套接字，SSL 加密通信、异步网络通信等功能
5. 网络协议，支持 HTTP , FTP , SMTP , POP , IMAP , NNTP , XMLRPC 等多种网络协议，并提供了编写网络服务器的框架
6. W3C 格式支持，包含 HTML , SGML , XML 的处理
7. 其它功能，包括国际化支持、数学运算、HASH、Tkinter 等

目前学过的有 : random、math、time、file、os、sys 等模块。可以通过 random 模块实现随机数处理、math 模块实现数学相关的运算、time 模块实现时间的处理、file 模块实现对文件的操作、OS 模块实现和操作系统的交互、sys 模块实现和解释器的交互。

5.2 第三方扩展库的介绍

强大的标准库奠定了 python 发展的基石，丰富和不断扩展的第三方库是 python 壮大的保证。我们可以进入 PyPI 官网：

<https://pypi.org>



我们可以看到发布的第三方库达到了十多万种，众多的开发者为 Python 贡献了自己的力量。

表 常用第三方库大汇总

分类	库名称	说明
环境管理	P	非常简单的交互式 python 版本管理工具
	Pyenv	简单的 Python 版本管理工具
	Vex	可以在虚拟环境中执行命令
	Virtualenv	创建独立 Python 环境的工具
	virtualenvwrapp	

	er	
包管理	pip	Python 包和依赖关系管理工具
	pip-tools	保证 Python 包依赖关系更新的一组工具
	Pipenv	Python 官方推荐的新一代包管理工具
	Poetry	可完全取代 setup.py 的包管理工具
包仓库	warehouse	下一代 PyPI
	Devpi	PyPI 服务和打包/测试/分发工具
分发 (打包为可执行文件 以便分发)	PyInstaller	将 Python 程序转成独立的执行文件(跨平台)
	Nuitka	将脚本、模块、包编译成可执行文件或扩展模块
	py2app	将 Python 脚本变为独立软件包 (Mac OS X)
	py2exe	将 Python 脚本变为独立软件包 (Windows)
	pysist	一个用来创建 Windows 安装程序的工具, 可以在安装程序中打包 Python 本身
构建工具 (将源码编译成软件)	Buildout	构建系统, 从多个组件来创建, 组装和部署应用
	BitBake	针对嵌入式 Linux 的类似 make 的构建工具
	Fabricate	对任何语言自动找到依赖关系的构建工具
交互式 Python 解析器	IPython	功能丰富的工具, 非常有效的使用交互式 Python
	bpython	界面丰富的 Python 解析器
	Ptpython	高级交互式 Python 解析器, 构建于 python-prompt-toolkit 之上
文件管理	Aiofiles	基于 asyncio, 提供文件异步操作
	Imghdr	(Python 标准库) 检测图片类型
	Mimetypes	(Python 标准库) 将文件名映射为 MIME 类型
	path.py	对 os.path 进行封装的模块
	Pathlib	(Python3.4+ 标准库) 跨平台的、面向对象的路径操作库
	Unipath	用面向对象的方式操作文件和目录
	Watchdog	管理文件系统事件的 API 和 shell 工具
日期和时间	Arrow	更好的 Python 日期时间操作类库
	Chronyk	解析手写格式的时间和日期
	Dateutil	Python datetime 模块的扩展
	PyTime	一个简单易用的 Python 模块, 用于通过字符串来操作日期/时间
	when.py	提供用户友好的函数来帮助用户进行常用的日期和时间操作
文本处理	chardet	字符编码检测器, 兼容 Python2 和 Python3

	Difflib	(Python 标准库)帮助我们进行差异化比较
	Fuzzywuzzy	模糊字符串匹配
	Levenshtein	快速计算编辑距离以及字符串的相似度
	Pypinyin	汉字拼音转换工具 Python 版
	Shortuuid	一个生成器库,用以生成简洁的,明白的,URL 安全的 UUID
	simplejson	Python 的 JSON 编码、解码器
	Unidecode	Unicode 文本的 ASCII 转换形式
	Xpinyin	一个用于把汉字转换为拼音的库
	Pygment	通用语法高亮工具
	Phonenumbers	解析,格式化,储存,验证电话号码
	Sqlparse	一个无验证的 SQL 解析器
特殊文本格式处理	Tablib	一个用来处理中表格数据的模块
	Pyexcel	用来读写,操作 Excel 文件的库
	python-docx	读取,查询以及修改 word 文件
	PDFMiner	一个用于从 PDF 文档中抽取信息的工具
	Python-Markdown2	纯 Python 实现的 Markdown 解析器
	Csvkit	用于转换和操作 CSV 的工具
自然语言处理	NLTK	一个先进的平台,用以构建处理人类语言数据的 Python 程序
	Jieba	中文分词工具
	langid.py	独立的语言识别系统
	SnowNLP	一个用来处理中文文本的库
	Thulac	清华大学自然语言处理与社会人文计算实验室研制推出的一套中文词法分析工具包
下载器	you-get	一个 YouTube/Youku/Niconico 视频下载器
图像处理	pillow	最常用的图像处理库
	imgSeek	一个使用视觉相似性搜索一组图片集合的项目
	face_recognition	简单易用的 python 人脸识别
	python-qrcode	一个纯 Python 实现的二维码生成器
OCR	Pyocr	Tesseract 和 Cuneiform 的一个封装 (wrapper)
	pytesseract	Google Tesseract OCR 的另一个封装

		(wrapper)
音频处理	Audiolazy	Python 的数字信号处理包
	Dejavu	音频指纹提取和识别
	id3reader	一个用来读取 MP3 元数据的 Python 模块
	TimeSide	开源 web 音频处理框架
	Tinytag	一个用来读取 MP3, OGG, FLAC 以及 Wave 文件音乐元数据的库
	Mingus	一个高级音乐理论和曲谱包, 支持 MIDI 文件和回放功能
视频和 GIF 处理	Moviepy	一个用来进行基于脚本的视频编辑模块, 适用于多种格式, 包括动图 GIFs
	scikit-video	SciPy 视频处理常用程序
地理位置	GeoDjango	世界级地理图形 web 框架
	GeoIP	MaxMind GeoIP Legacy 数据库的 Python API
	Geopy	Python 地址编码工具箱
HTTP	requests	人性化的 HTTP 请求库
	httplib2	全面的 HTTP 客户端库
	urllib3	一个具有线程安全连接池, 支持文件 post, 清晰友好的 HTTP 库
Python 实现的数据库	pickleDB	一个简单, 轻量级键值储存数据库
	PipelineDB	流式 SQL 数据库
	TinyDB	一个微型的, 面向文档型数据库
web 框架	Django	Python 界最流行的 web 框架
	Flask	一个 Python 微型框架
	Tornado	一个 web 框架和异步网络库
CMS 内容管理系统	odoo-cms	一个开源的, 企业级 CMS, 基于 odoo
	djedi-cms	一个轻量级但却非常强大的 Django CMS, 考虑到了插件, 内联编辑以及性能
	Opps	一个为杂志, 报纸网站以及大流量门户网站设计的 CMS 平台, 基于 Django
电子商务和支付系统	django-oscar	一个用于 Django 的开源的电子商务框架
	django-shop	一个基于 Django 的店铺系统
	Shoop	一个基于 Django 的开源电子商务平台
	Alipay	Python 支付宝 API
	Merchant	一个可以接收来自多种支付平台支付的

		Django 应用
游戏开发	Cocos2d	用来开发 2D 游戏
	Panda3D	由迪士尼开发的 3D 游戏引擎，并由卡内基梅陇娱乐技术中心负责维护。使用 C++ 编写，针对 Python 进行了完全的封装
	Pygame	Pygame 是一组 Python 模块，用来编写游戏
	RenPy	一个视觉小说 (visual novel) 引擎
计算机视觉库	OpenCV	开源计算机视觉库
	Pyocr	Tesseract 和 Cuneiform 的包装库
	SimpleCV	一个用来创建计算机视觉应用的开源框架
机器学习 人工智能	TensorFlow	谷歌开源的最受欢迎的深度学习框架
	keras	以 tensorflow/theano/CNTK 为后端的深度学习封装库，快速上手神经网络
	Hebel	GPU 加速的深度学习库
	Pytorch	一个具有张量和动态神经网络，并有强大 GPU 加速能力的深度学习框架
	scikit-learn	基于 SciPy 构建的机器学习 Python 模块
	NuPIC	智能计算 Numenta 平台
科学计算和数据分析	NumPy	使用 Python 进行科学计算的基础包
	Pandas	提供高性能，易用的数据结构和数据分析工具
	SciPy	用于数学，科学和工程的开源软件构成的生态系统
	PyMC	马尔科夫链蒙特卡洛采样工具
代码分析和调试	code2flow	把你的 Python 和 JavaScript 代码转换为流程图
	Pycallgraph	这个库可以把你的 Python 应用的流程(调用图)进行可视化
	Pylint	一个完全可定制的源码分析器
	autopep8	自动格式化 Python 代码，以使其符合 PEP8 规范
	Wdb	一个奇异的 web 调试器，通过 WebSockets 工作
	Lineprofiler	逐行性能分析
	Memory Profiler	监控 Python 代码的内存使用
图形用户界面	Pyglet	一个 Python 的跨平台窗口及多媒体库

	PyQt	跨平台用户界面框架 Qt 的 Python 绑定，支持 Qt v4 和 Qt v5
	Tkinter	Tkinter 是 Python GUI 的一个事实标准库
	wxPython	wxPython 是 wxWidgets C++ 类库和 Python 语言混合的产物
网络爬虫和 HTML 分析	Scrapy	一个快速高级的屏幕爬取及网页采集框架
	Cola	一个分布式爬虫框架
	Grab	站点爬取框架
	Pyspider	一个强大的爬虫系统
	html2text	将 HTML 转换为 Markdown 格式文本
	python-goose	HTML 内容/文章提取器
硬件编程	Ino	操作 Arduino 的命令行工具
	Pyro	Python 机器人编程库
	PyUserInput	跨平台的，控制鼠标和键盘的模块
	Pingo	Pingo 为类似 Raspberry Pi ,pcDuino , Intel Galileo 等设备提供统一的 API

5.3 PyPI 网站和 PIP 模块管理工具

PyPI(Python Package Index)是 python 官方的第三方库的仓库，所有人都可以下载第三方库或上传自己开发的库到 PyPI。PyPI 推荐使用 pip 包管理器来下载第三方库。

pip 是一个现代的，通用的 Python 包管理工具。提供了对 Python 包的查找、下载、安装、卸载的功能。pip 可正常工作在 Windows、Mac OS、Unix/Linux 等[操作系统](#)上，但是需要至少 2.6+和 3.2+的 CPython 或 PyPy 的支持。python 2.7.9 和 3.4 以后的版本已经内置累 pip 程序，所以不需要安装。

5.4 安装第三方扩展库的 2 种方式

第三方库有数十万种之多，以 pillow 库为例讲解第三方扩展库的安装。pillow 是 Python 平台事实上的图像处理标准库，本节以安装 pillow 为例，给大家介绍第三方库的两种常用的安装方法。

第一种方式：命令行下远程安装

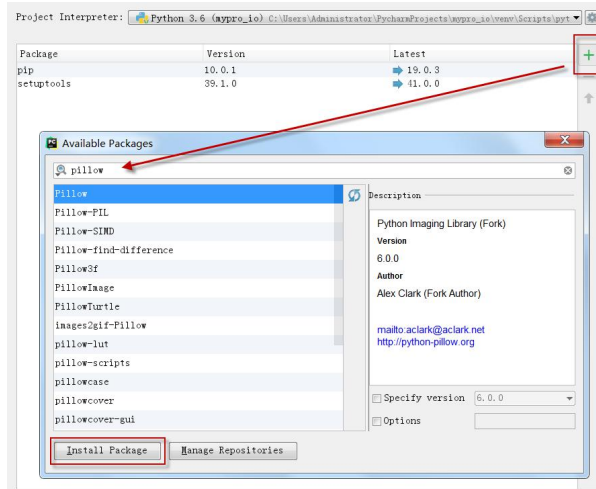
以安装第三方 pillow 图像库为例，在命令行提示符下输入：pip install pillow
安装完成后，我们就可以开始使用。

安装完，输入 pip show pillow，进行确认：

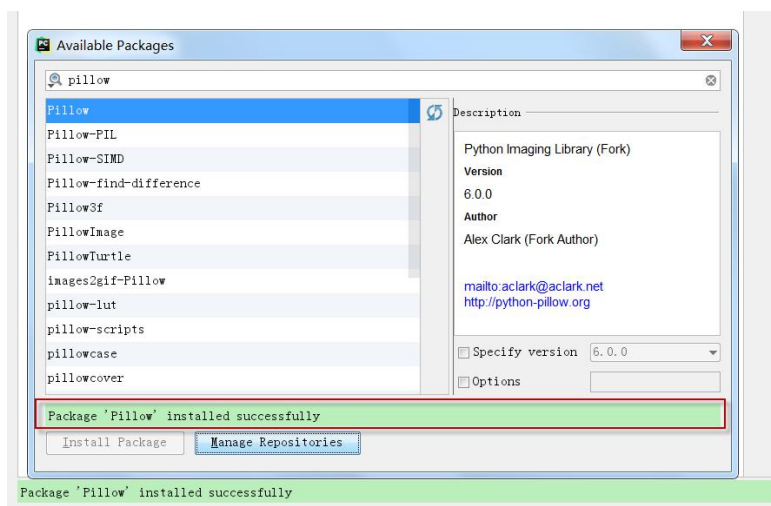
```
C:\Users\Administrator>pip show pillow
Name: Pillow
Version: 5.3.0
Summary: Python Imaging Library (Fork)
Home-page: http://python-pillow.org
Author: Alex Clark (Fork Author)
Author-email: aclark@aclark.net
License: Standard PIL License
Location: c:\program files (x86)\python36-32\lib\site-packages
Requires:
Required-by:
```

第二种方式：Pycharm 中直接安装到项目中

在 Pycharm 中，依次点击：file-->setting-->Project 本项目名-->Project Interpreter



点击“+”，然后输入要安装的第三方库“pillow”，再点击按钮“Install Package”，等待安装即可，几秒钟后，即提示安装成功：



这样，我们就可以在项目中直接使用第三方库 pillow 了。