第十章模块对象

> 本文是Python通用编程系列教程，已全部更新完成，实现的目标是从零基础开始到精通Python编程语言。本教程不是对Python的内容进行泛泛而谈，而是精细化，深入化的讲解，共5个阶段，25章内容。所以，需要有耐心的学习，才能真正有所收获。虽不涉及任何框架的使用，但是会对操作系统和网络通信进行全局的讲解，甚至会对一些开源模块和服务器进行重写。学完之后，你所收获的不仅仅是精通一

> 门Python编程语言，而且具备快速学习其他编程语言的能力，无障碍阅读所有Python源码的能力和对计算机与网络的全面认识。对于零基础的小白来说，是入门计算机领域并精通一门编程语言的绝佳教材。对于有一定Python基础的童鞋，相信这套教程会让你的水平更上一层楼。

一 模块介绍

1. 模块基本说明

我们之前讲函数的时候，一个函数是一个功能或者工具，我们可以重复使用这个功能，当程序中有多个功能的时候，为了给这些功能分组，我们有了模块的概念。模块是一系列功能的集合体，在Python中，一个Python文件就是一个模块，比如module.py,其中模块名module，Pyhon中一切皆对象，一个模块也是一个对象。

2. 模块的基本使用

我们使用`import`关键字导入一个模块，在一个Python文件中首次导入模块的过程分为三个阶段：

1. 创建一个模块的名称空间
2. 执行模块对应文件，将产生的名字存放于模块文件所对应的名称空间
3. 在当前执行文件中拿到一个模块名，该模块名指向模块文件所对应的名称空间

需要注意的是：导入一个模块，模块中功能的执行始终以模块自己的名称空间为准，如果与导入模块的文件内的全局变量重名了，这也只是巧合，他们之间没有任何关系(最好不要有命名一样的)。

```
# spam模块，与Python文件在同路径下
print('from the spam.py')

money = 0


def read1():
    print('spam模块.read1：', money)


def read2():
    print('spam模块.read2')
    read1()
```

```
def change():
    global money
    money = 1   # 在模块中修改


# Python文件
import spam

print(spam)   # 打印导入的模块

print(spam.money)   # spam对象有money属性
spam.read1()   # spam对象有read1方法

money = 1
read1 = 2
read2 = 3

print(spam.read1)
spam.read1()
spam.read2()
spam.change()
print(money)
spam.read1()
```

强调：如果多次导入同一模块，之后的导入会直接引用第一次导入的结果，不会重复执行文件

```
import spam

print(spam)   # 打印导入的模块
```

3. 为模块其别名

```
# 3、为模块起别名
import spam as sm

print(sm.money)
sm.read1()

# 应用
```

```
engine = input('>>: ').strip()
if engine == 'mysql':
    import mysql as db
elif engine == 'oracle':
    import oracle as db

# 要执行的数据库操作
db.parse()    # 自己定义mysql和oracle模块中有parse方法
```

4. 一行导入多个模块(不推荐使用)

```
import spam, mysql, oracle

# 推荐写成多行
import spam
import mysql
import oracle
```

5. from... import...

我们在使用import导入一个模块的时候，每次使用都要使用`模块名.属性名`的方式，有一种更简单的方式就是使用from...import...导入，它的执行前两个阶段与import导入一致，第三个阶段不同：在当前名称空间中直接拿到模块中的名字，可以直接使用，不用加任何前缀。

```
from spam import money, read1, read2, change

print(money)
read1()
read2()
change()

# 1、同import，执行模块中的功能，始终以模块的名称空间为准
money = 2
change()
print(money)

# 2 from ... import 名字，拿到的名字可以不加前缀直接使用，使用起来更加方便
# 当问题是如果重名的话，会与当前执行文件中相同的名字冲突
money = 3
```

```
print(money)
read1 = 4
# read1()  # 无法调用

# 3 起别名
from spam import money as m

print(m)

# 4 在一行导入多个
from spam import money, read1, read2

# 5 from ... import * 全部导入
from spam import *

print(money)
print(read1)
print(read2)
print(change)
```

6. 星的导入

星指的是导入该模块内的全部功能，如果模块内有很多功能，调用者并不会全部都用到，这是为了调用者的方便，但是作为调用者不可避免的出现：自己的定义的名字与模块中的名字冲突，这时作为模块的设计者可以定义星的调用。

```
# 修改spam模块

# 规定*的调用只调用money和read1这两个功能
__all__ = ['money', 'read1']

money = 0


def read1():
    print('spam模块.read1：', money)


def read2():
    print('spam模块.read2')
    read1()
```

```python
def change():
    global money
    money = 1  # 在模块中修改

# 调用者
from spam import *

print(money)
print(read1)
# print(read2)  # 无法调用
# print(change)
```

7. Python文件的两种用途

一个Python文件可以当作可执行文件被执行，这种用途一般我们叫做脚本文件，另外一种就是我们刚才所讲的，作为模块导入。为了测试方便，我们需要区别对待。

```python
# spam模块
__all__ = ['money', 'read1']

money = 0


def read1():
    print('spam模块.read1：', money)


def read2():
    print('spam模块.read2')
    read1()


def change():
    global money
    money = 1


print(__name__)
# __name__的值
# 1 在文件被直接执行的情况下，等于'__main__'
# 2 在文件被导入的情况下，等于模块名
```

```
if __name__ == '__main__':

    # 测试代码写在这里面
    print('文件被当中脚本执行啦。、。')
    read1()
else:
    print('文件被导入啦')

# 执行文件
from spam import *

print(money)
print(read1)
```

8. 模块的查找路径

模块搜索查找的循序是：

1. 内存中已经加载的模块
2. 内置模块
3. sys.path路径中包含的模块

```
# 1 验证先查找内存中加载的模块
import time

# 在当前文件夹下创建spam1.py文件
import spam1

spam1.f1()

time.sleep(15)  # 15秒时间从硬盘删除spam1.py这个文件
import spam1

spam1.f1()  # 再次调用依然有效，证明是优先从内存中查找的

# 删除之后再次调用，内存中没有spam模块，内置也没有，报错


# 2 验证内置模块的查找
import sys
```

```python
print('time' in sys.modules)  # sys.modules存放内存中被导入的模块

import time  # 内置模块默认在硬盘，导入之后才会进入内存

time.sleep(3)  # 可以使用time模块证明：模块查找先查找的内置模块
print('time' in sys.modules)

# 3 sys.path路径包含的模块(重点)
import sys

print(sys.path)

# 如果把spam1.py文件移动到当前目录的x文件夹下(下图有说明)，唯一的方案就是把x文件夹路径加如sys.path路径下
sys.path.append(r'/Users/albert/Desktop/函数/x/')

print(sys.path)

import spam1  # 要保证导入的代码不变

spam1.f1()

# 注意：sys.path的第一个路径是当前执行文件所在的文件夹


# spam1模块
def f1():
    print('被导入：m1.f1 function')

# 自定义time模块
print('time自定义')
```

二 包的介绍

1. 包的基本介绍

包是一个特殊的模块，你的计算机上不同类型的文件分别保存的不同的文件夹内，程序中也是一样。一个文件夹就是一个包，Python语言的设计为了兼顾统一，包的使用和模块一致。需要注意的是，一个包内必须要有一个初始化文件`__init__.py`。

```
# 先创建以a命名Python Package，在包内创建两个文件
# x1.py
def func1():
    print('a.x1.func1')
# y1.py
def func2():
    print('a.y1.func2')

# 执行文件
import a.x1,a.y1 # 创建一个以a未命名的包文件

'''
1 产生一个包的名称空间
2 执行包下的__init__.py文件，将产生的名字存放于包的名称空间中
3 在当前执行文件中拿到一个名字a，该名字指向包的名称空间
'''
a.x1.func1()
a.y1.func2()
print(a.x1.func1)
print(a.y1.func2)
```

2. 导入包注意事项

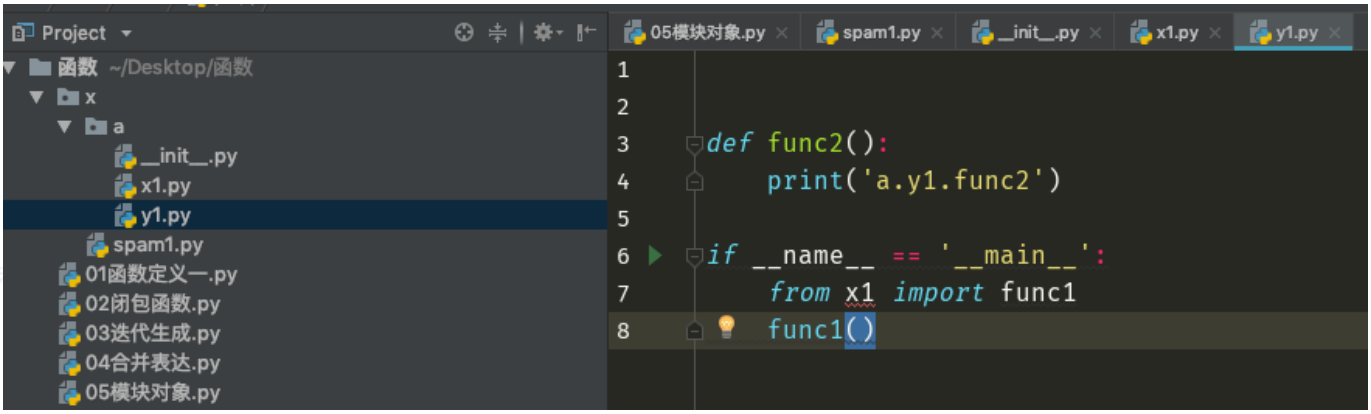
1. 在导入时带点的，点的左边必须是一个包，这是导入包特有的语法
2. 包内模块直接的导入应该使用from。。。import 。。。
3. from 。。。 import。。。，import后必须是一个明确的名字，没有任何的前缀，不能加点

3. 绝对导入与相对导入

绝对导入是以当前执行文件所在的文件夹(sys.path的第一个路径)为参照点导入的，相对导入是以当前执行文件为参照点导入的。

```
# 绝对导入

# 执行文件
from x.a import x1, y1

x1.func1()
y1.func2()
print(x1.func1)
print(y1.func2)

# 相对导入

# y1模块
def func2():
    print('a.y1.func2')

if __name__ == '__main__':
    from x1 import func1
    func1()
```

目录结构如下图所示



三 常用模块

1. time模块

```python
import time

# 时间分为三种形式
# 1、时间戳
print(time.time())
start_time = time.time()
time.sleep(3)
stop_time = time.time()
print(stop_time - start_time)

# 2、格式化的字符串
print(time.strftime('%Y-%m-%d %H:%M:%S %p'))
print(time.strftime('%Y-%m-%d %X %p'))

# 3、struct_time对象
print(time.localtime())  # 上海：东八区
print(time.localtime().tm_year)
print(time.localtime().tm_mday)

print(time.gmtime())  # UTC时区

# 以下为了解的知识
print(time.localtime(111).tm_hour)
print(time.gmtime(11111).tm_hour)

print(time.mktime(time.localtime()))

print(time.strftime('%Y/%m/%d', time.localtime()))
print(time.strptime('2017/04/08', '%Y/%m/%d'))

print(time.asctime(time.localtime()))
print(time.ctime(123))
```

用time模块来模拟网络延迟打印进度条

```python
# 打印过程
"""
print('[                    ]')
print('[##                  ]')
print('[###                 ]')
print('[####                ]')
print('[#####               ]')
```

```
"""

# 显示数字
"""
print('[%-50s]' %'#')
print('[%-50s]' %'##')
print('[%-50s]' %'###')
"""


# 说明
# 第一个%是取消第二个%号的特殊意义的
# num=30
# print('%s%%' %num)
# width=30
# print(('[%%-%ds]' %width) %'#')
# print(('[%%-%ds]' %width) %'##')
# print(('[%%-%ds]' %width) %'###')


def progress(percent, width=50):
    if percent > 1:
        percent = 1
    show_str = ('[%%-%ds]' % width) % (int(width * percent) * '#')
    print('\r%s %d%%' % (show_str, int(100 * percent)), end='')


import time

recv_size = 0
total_size = 809700
while recv_size < total_size:
    time.sleep(0.1)
    recv_size += 8096
    percent = recv_size / total_size
    progress(percent)
```

2. datatime模块

```
import datetime

print(datetime.datetime.now())
```

```
print(datetime.datetime.now() + datetime.timedelta(days=3))
print(datetime.datetime.now() + datetime.timedelta(days=-3))
print(datetime.datetime.now() + datetime.timedelta(hours=3))
print(datetime.datetime.now() + datetime.timedelta(seconds=111))

current_time = datetime.datetime.now()
print(current_time.replace(year=1977))

print(datetime.date.fromtimestamp(111111))
```

3. shutil与tarfile

```
# 压缩文件
import shutil
import time

ret = shutil.make_archive(
    "模块对象_%s" % time.strftime('%Y-%m-%d'),
    'gztar',
    root_dir=r'/Users/albert/Desktop/函数/x'
)

# 解压文件
import tarfile

t = tarfile.open('/Users/albert/Desktop/函数/模块对象_2019-04-15.tar.gz', 'r')
t.extractall(r'/Users/albert/Desktop/函数/x/a/解包目录')
t.close()
```

4. logging模块

```
import logging

logging.basicConfig(
    filename='access.log',
    format='%(asctime)s - %(name)s - %(levelname)s -%(module)s:  %
(message)s',
    datefmt='%Y-%m-%d %H:%M:%S %p',
    level=20,
```

```
)

logging.debug('debug...')  # 10
logging.info('info....')  # 20
logging.warning('可能着火...')  # 30
logging.error('着火啦快跑')  # 40
logging.critical('火越烧越大')  # 50

# logging模块的四种对象

# 1 logger:负责产生日志
logger1 = logging.getLogger('xxx')

# 2 filter：过滤日志（不常用）

# 3 handler：控制日志打印到文件or终端
fh1 = logging.FileHandler(filename='a1.log', encoding='utf-8')
fh2 = logging.FileHandler(filename='a2.log', encoding='utf-8')
sh = logging.StreamHandler()

# 4 formatter：控制日志的格式
formatter1 = logging.Formatter(
    fmt='%(asctime)s - %(name)s - %(levelname)s -%(module)s:  %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S %p',
)

formatter2 = logging.Formatter(fmt='%(asctime)s - %(message)s', )

# 为logger1对象绑定handler
logger1.addHandler(fh1)
logger1.addHandler(fh2)
logger1.addHandler(sh)

# 为handler对象绑定日志格式
fh1.setFormatter(formatter1)
fh2.setFormatter(formatter1)
sh.setFormatter(formatter2)

# 日志级别：两层关卡，必须都通过，日志才能正常记录
# logger1.setLevel(10)
# logger1.setLevel(20)
logger1.setLevel(30)

fh1.setLevel(10)
fh2.setLevel(10)
```

```
sh.setLevel(10)

# 调用logger1对象下的方法，产生日志，然后交给不同的handler，控制日志记录到不同的地方
logger1.debug('调试信息')
logger1.info('使用中')
logger1.warning('出错了')
logger1.critical('问题很严重')
```

5. json与pickle

```
# json序列化
import json

dic = {'name': 'Albert', 'age': 18}

with open('db1.json', 'wt', encoding='utf-8') as f:
    json.dump(dic, f)  # 写入

# 反序列化
with open('db1.json', 'rt', encoding='utf-8') as f:
    dic = json.load(f)  # 读取
    print(dic['name'])

# pickle序列化
import pickle

s = {1, 2, 3, 4, }
res = pickle.dumps(s)
print(res, type(res))

with open('db.pkl', 'wb') as f:  # 二进制模式
    f.write(res)

# 反序列化
with open('db.pkl', 'rb') as f:
    data = f.read()
    s = pickle.loads(data)
    print(s, type(s))

# dump与load
import pickle
```

```python
s = {1, 2, 3}
with open('db1.pkl', 'wb') as f:
    pickle.dump(s, f)

with open('db1.pkl', 'rb') as f:
    s = pickle.load(f)
    print(s, type(s))
```

6. os模块

```python
import os

res = os.getcwd()
print(res)

res = os.listdir('.')
print(res)

print(os.sep)
print([os.linesep, ])
print(os.pathsep)

print(os.system('ls'))  # 执行系统命令‘ls’(MacOS系统), Windows用‘dir’命令

# os.path系列

file_path = r'a/b/c/d.txt'
print(os.path.abspath(file_path))

res = os.path.split(r'a/b/c/d.txt')
print(res[-1])
print(res[0])

print(os.path.isabs(r'b/c/d.txt'))
print(os.path.isabs(r'/Users/albert/Desktop/函数/05模块对象.py'))

BASE_DIR = os.path.dirname(os.path.dirname(__file__))
DB_PATH = r'%s\db\db.txt' % BASE_DIR

print(BASE_DIR)
print(DB_PATH)
```

```
# 优先掌握

# 判断文件时候存在
print(os.path.exists(r'/Users/albert/Desktop/函数/05模块对象.py'))
print(os.path.exists(r'/Users/Desktop/函数/05模块对象.py'))
print(os.path.isfile(r'/Users/albert/Desktop/函数/05模块对象.py'))

# 判断文件夹是否存在
print(os.path.isdir(r'/Users/albert/Desktop/函数/x'))

print(os.path.join('C:\\', 'a', 'b', 'a.txt'))
print(os.path.join('C:\\', 'a', 'D:\\', 'b', 'a.txt'))
print(os.path.join('a', 'b', 'a.txt'))

# 得到文件大小
res = os.path.getsize(r'/Users/albert/Desktop/函数/05模块对象.py')  # 单位是字节
print(res)
```

7. shelve模块

```
import shelve

info1 = {'age': 18, 'height': 180, 'weight': 80}
info2 = {'age': 34, 'height': 203, 'weight': 113}

d = shelve.open('db.shv')
d['Albert'] = info1
d['James'] = info2
d.close()

d = shelve.open('db.shv')
print(d['Albert'])
print(d['James'])
d.close()
```

8. re模块(正则表达式)

正则就是用一些具有特殊含义的符号组合到一起（称为正则表达式）来描述字符或者字符串的方，或者说：正则就是用来描述一类事物的规则。它内嵌在Python中，并通过 re 模块实现。正则表达式模式被编译成一系列的字节码，然后由用 C 编写的匹配引擎执行。

生活中处处都是正则，比如我们描述一个四条腿的东西，你可能会想到的是四条腿的动物或者桌子，椅子等，如果我在继续添加限制继续描述：四条腿的活的东西，就只剩下四条腿的动物这一类了，如果我再添加限制，四条腿的活的喜欢拆家，那估计这个东西是什么就会非常明确了。

```
import re

# 基本使用
print(re.findall('\w', 'ab 12\+- *&_'))  # 匹配字母数字和下划线
print(re.findall('\W', 'ab 12\+- *&_'))  # 匹配非字母数字下划线
print(re.findall('\s', 'ab \r1\n2\t\+- *&_'))  # 匹配任意空白字符
print(re.findall('\S', 'ab \r1\n2\t\+- *&_'))  # 匹配任意非空字符
print(re.findall('\d', 'ab \r1\n2\t\+- *&_'))  # 匹配任意数字
print(re.findall('\D', 'ab \r1\n2\t\+- *&_'))  # 匹配任意非数字
print(re.findall('\w_nb', 'albert james_nb123123curry_nb,harden_nb'))

print(re.findall('\Aalbert', 'abcalbertx is nb'))  # 匹配不到
print(re.findall('\Aalbert', 'albert is nb'))  # 匹配字符串开始
print(re.findall('^albert', 'albert is nalbertb'))  # 匹配albert开头的字符串
print(re.findall('nba$', 'albertnba is super nba alertanba'))  # 匹配nba结尾
print(re.findall('^albert$', 'albert'))  # 以albert并以albert结尾
print(re.findall('a\nc', 'a\nc a\tc a1c'))  # 匹配一个换行符

# 重复匹配：
# .   ?   *   +  {m,n}  .*  .*?
# 1 .:代表除了换行符外的任意一个字符
print(re.findall('a.c', 'abc a1c aAc aaaaaca\nc'))
print(re.findall('a.c', 'abc a1c aAc aaaaaca\nc', re.DOTALL))

# 2 ？：代表左边那一个字符重复0次或1次
# print(re.findall('ab?','a ab abb abbb abbbb abbbb'))

# 3 *：代表左边那一个字符出现0次或无穷次
print(re.findall('ab*', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))

# 4 + ：代表左边那一个字符出现1次或无穷次
print(re.findall('ab+', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))

# 5、{m,n}:代表左边那一个字符出现m次到n次
print(re.findall('ab?', 'a ab abb abbb abbbb abbbb'))
print(re.findall('ab{0,1}', 'a ab abb abbb abbbb abbbb'))
print(re.findall('ab*', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))
print(re.findall('ab{0,}', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))
```

```
print(re.findall('ab+', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))
print(re.findall('ab{1,}', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))
print(re.findall('ab{1,3}', 'a ab abb abbb abbbb abbbb a1bbbbbbb'))

# 6 贪婪匹配：.*：匹配任意长度，任意的字符
print(re.findall('a.*c', 'ac a123c aaaac a *123)()c asdfasfdsadf'))

# 7 非贪婪匹配：.*?
print(re.findall('a.*?c', 'a123c456c'))


# 8 其他方法：
print(re.findall('\Aalbert', '123albert say........'))
print(re.findall('^albert', 'albert say........'))
print(re.search('albert', '123albert say........').group())  # 整个字符串中查找
print(re.match('albert', '123albert say........'))  # 从开头找
```

9. hashlib模块

哈希是一种算法，该算法接受传入的内容，经过运算得到一串hash值。hash值的特点有三个：

1. 只要传入的内容一样，得到的hash值必然一样=====>文件完整性校验
2. 不能由hash值返解成内容，把密码做成hash值，不应该在网络传输明文密码
3. 只要使用的hash算法不变，无论校验的内容有多大，得到的hash值长度是固定的

```
import hashlib

# 1 字符串加密
m = hashlib.md5()  # md5是加密算法的一种
# m = hashlib.sha256()
# m = hashlib.sha512()
m.update('hello'.encode('utf-8'))
m.update('world'.encode('utf-8'))
m.update('Albert'.encode('utf-8'))
print('字符串md5值：', m.hexdigest())  # 9b904c5a3b6b865bf0ac9413887c375b

# 2 文件md5值效验
m = hashlib.md5()
with open(r'/Users/albert/Desktop/函数/05模块对象.py', 'rb') as f:
    for line in f:
        m.update(line)
```

```
    file_md5 = m.hexdigest()
print('文件md5值：', file_md5)

# 3 密码加盐
import hashlib

pwd = 'albert123'

m = hashlib.md5()
m.update('天王盖地虎'.encode('utf-8'))
m.update(pwd.encode('utf-8'))
m.update('宝塔炖蘑菇'.encode('utf-8'))
print('密码md5值：', m.hexdigest())
```

10. subprocess模块

```
import subprocess

# subprocess用于在程序中执行系统命令

cmd = input('>>:').strip()  # MacOS系统用ls测试，Windows系统用dir测试

obj = subprocess.Popen(cmd,
                       shell=True,
                       stdout=subprocess.PIPE,
                       stderr=subprocess.PIPE
                       )

print(obj)

res1 = obj.stdout.read()  # 输出正确结果
print('正确结果: ', res1.decode('utf-8'))

res2 = obj.stderr.read()  # 输出错误结果
print('错误结果：', res2.decode('utf-8'))
```