第五节：

Python 进阶之模块

模块定义：

要实现某个功能的所有代码的集合 称之为模块

模块的种类：

1.自定义模块

2.内置模块

3.第三方模块

模块类型：

模块主要分为两种类型 一种是文件形式（.py文件）、一种是“包” 包其实就是一个包含\_\_init\_\_.py 文件的文件夹，init文件主要用来初始化包,python 通过检测\_\_init\_\_.py文件是否存在来判断文件夹是否为包，\_\_init\_\_.py文件可以为空 也可以包含\_\_all\_\_=[‘func1,func2’] 其中func1，func2就是包中包含的Python文件

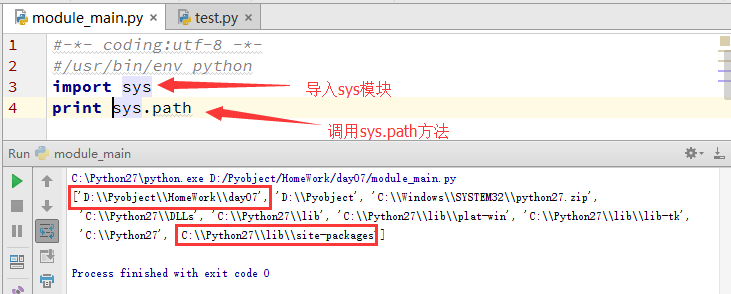
模块的导入以及调用模块中的方法（函数）：

导入模块之前首先我们要知道python调用模块的步骤

1. 检索可用路径（这个可用路径类似于linux下的PATH全局变量）
2. 根据路径的优先级来查找模块，其中当前路径的优先级最高
3. 从可用路径中查找模块

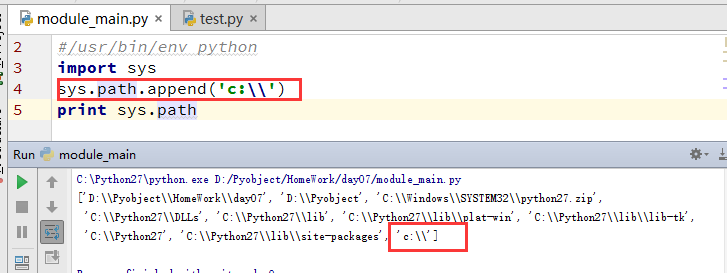
通过以上三点我们可以知道要使我们的模块可以被python检索到，模块就必须放在可用路径内，利用sys模块下的sys.path方法可以查看python所检索的路径范围并以列表的方式输出

如下图所示



通过上图可以看到python的所有检索范围，而排在第一位的就是当前目录，其中最后一个（c:\\Python27\\lib\\site-packages 不同环境路径可能不同 记住site-packages就好了）为标准化的第三方模块存放路径，一般都将模块放在这里。

如果想要自己定义模块的存放位置可以先添加自己的路径到sys.path，通过sys.path.append(‘dir’)来添加自己的模块存储路径，其实就是像目录列表中添加一个元素



OK 了解完python的检索路径 下面我们开始看下如何导入模块，导入模块分为以下几种方法

方法1：

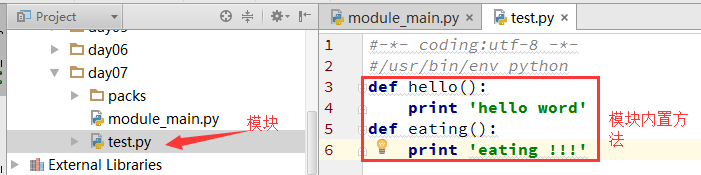
import module

调用：

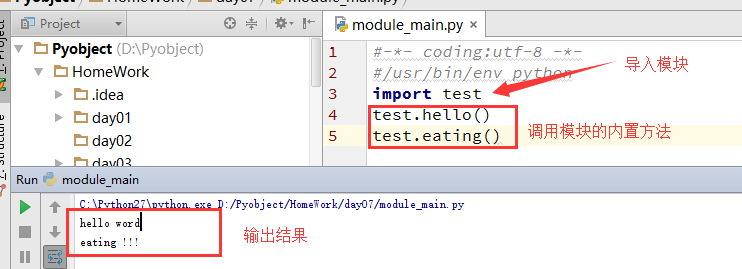
module.func()

Example：

首先我们定义一个模块test.py



导入模块



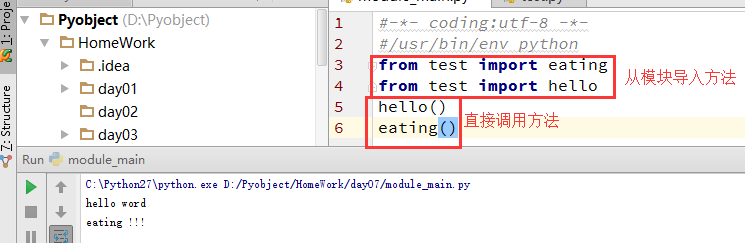
方法二：

from module import func

调用：

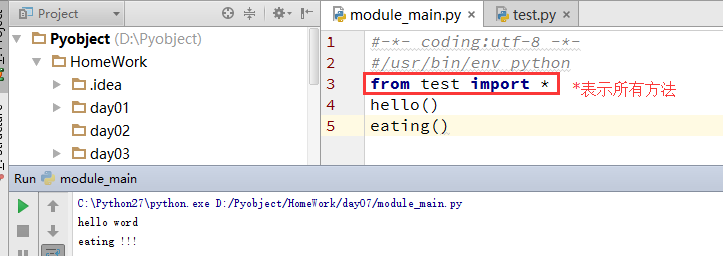
func()

Example：



从上可以看出这种导入方法调用的时候是很方便的，但是也有他的弊端 比如有两个模块 a,b 两个模块中都有一个hello（）方法 这时候就会方法冲突 所以一般不建议用这种方式

另外这种方法还有一种不太严谨的导入方式 如下：

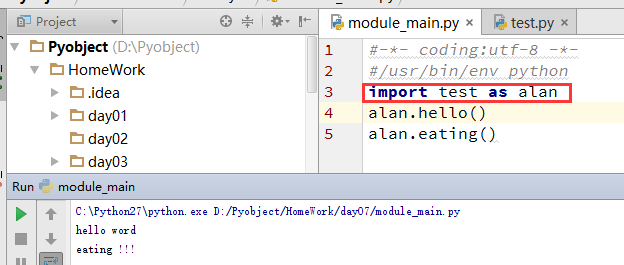


别名：

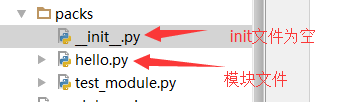
import module as py\_mode

调用：

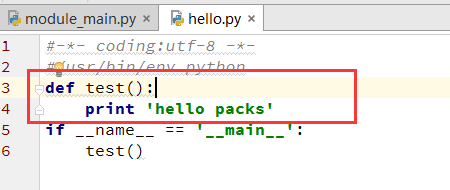
module.py\_mode

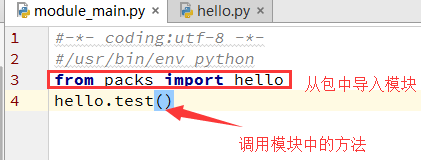


上面讲到了.py文件的导入方式 那么包的导入方式是怎样的呢？ 其实包的导入方式类似



hello模块内容如下





第六节：

1. json/pickle

作用：

为什么要用json？

硬盘之接受字符串和二进制， 如果将复杂的数据结构（比如：字典、元组、列表）写入硬盘就必须要将其转换为字符串然后才能存到硬盘，由于不同程序 不同语言之间有的时候要进行内存交互 然而不同程序的内存是相互独立的 要进行数据交互就必须要用一个中间的插件做转换 这个插件就是json。

作用一：

不同不同语言、不同平台之间进行复杂格式交互

作用二：

多程序之间内存数据交换

主要操作文件

json.dump() 序列化

json.load()

主要处理临时数据

json.dumps()

json.loads(string)

pickle 和 json 功能用法一致

pickle 能够转换几乎所有的python数据类型

2.subprocess

作用：

执行系统命令

call（[commond],shell=True） 执行命令

shell = True 用原生的shell去执行命令而并非python的内部shell环境去执行 漏洞权限问题

shell = True 必须为字符串 shell = False 必须为序列

check\_output 返回命令的执行结果

check\_all 如果结果为0则返回0 如果命令执行出错则抛出异常

Popen

交互时

subprocess.Popen([python],stdin=subprocess.PEIE,stdout=subprocess.PEIE，stderr=subprocess.PEIE)

3.shutil

shutil.copyfileobj('s.txt','d.txt',length = 20)

shutil.copyfile('s.txt','d.txt')

shutil.make\_archive(filename,format,root\_dir=dir)

shutil.copytree()

shutil.rmtree()

shutil.copystat()

4.time

time.strftime('%Y/%m/%d')

t=time.strptime('2015-12,13','%Y-%m-%d')

t.m\_year

t.m\_day

t.m\_mon

time.mktime(t)

5.datetime

dtetime.timedelta(type = number)

type:

hours

minents

days

secounter

6.logging

logging.basicConfig()

logging.debug

logging.getLogger('name')

import logging

logger = logging.getLogger('simple\_example')

logger.setLevel(logging.DEBUG)

#on screen

ch = logging.StreamHandler()

ch.setLevel(logging.WARING)

#on file

fh = logging.FileHandler('logs.log')

fh.setLevel(logging.DEBUG)

#format

formatter = logging.Formatter("%(asctime)s-%(name)s-%(levelname)s-%(message)s")

ch.setFormatter(formatter)

fh.setFormatter(formatter)

#将模块交给logger句柄

logger.addHandler(ch)

logger.addHandler(fh)

#触发执行日志级别

logger.debug('debug message ......')

logger.info('info message ......')

logger.warn('waring message ......')

logger.error('error message ......')

7.正则表达式 re 模块

方法：

re.match() 从开头匹配字符

re.search() 从整个字符串中匹配第一个匹配的

re.findall() 从整个字符串中匹配所有匹配的

re.split() 分割

re.replace() 替换

re.sub(source,destion,string,count=1)

1.匹配

. 匹配除去换行符之外的所有字符

\W 匹配字母或者数字或下划线、汉字

\s 匹配任意的空白符

\d 匹配数字

\b 匹配单次的开始或者结束

^ 匹配字符串的开始

$ 匹配字符串的结尾

匹配次数：

\* 重复0次或者多次（匹配所有字符）

+ 重复一次或者多次（至少匹配一个字符）

？重复零次或者一次（匹配0个或者一个字符）

{n} 重复n次

{n,} 重复n次到无限次

{n,m} 重复n到m次

匹配格式：

[...] 匹配其中任意字符

(...) 匹配结果进行分组

面向对象：

发展历程

函数式编程 ===》 函数式编程+面向对象（低级基础）【经典类】 ===》 函数式+面向对象【新式类】（添加、修改新功能）

【经典类】

定义：class name

多继承：深度优先 从下至上

【新式类】

不管直接还是间接都是新式类

多继承：广度优先（横向优先 从左向右，）

定义：class name（object）

特性：更好更容易的进行代码的重用、扩展、封装、多态

定义:

class Person(object):

实例化：Person()

4493506.html

封装

继承

继承和对象无关

父类/基类 子类/派生类

多继承

class test（a,b,c）

父类中方法冲突则继承c

类成员：

字段（变量）

普通字段（存在self中）

静态字段（存在类中，即内存中）

访问格式 class.name

方法（函数）

方法执行的时候加括弧

普通方法：

对象触发 可以有多个参数 但必须要有self参数

def f1(self):

pass

类方法：

无需创建对象 类自身触发 只能有有1个参数 cls

@classmethod

def f1(cls):

pass

静态方法：

无需创建对象 类自身触发 可以有多个参数 参数自己定义

@staticmethod

def f1():

pass

属性（函数加上property装饰器变成属性）

@property

def fl(self):

pass

将方法伪造成字段 属性执行的时候不加括弧 只能用有一个self参数

新式类：

@property

def price(self):

pass

@price.setter 赋值

def price(self,value):

pass

@price.deleter 删除值

def price(self):

pass

类修饰符：

公有

私有:

只有类内部方法属性可以调用 外部无法调用访问

\_\_fuck

\_\_dict\_\_

返回对象下所有东西并以字典格式打印

\_\_str\_\_