**第13章 模 块**

在之前的学习中已经接触过模块，实际上，在之前的程序或命令行中使用的形如“import 文件名”的语句，就是用于导入模块的语句。本章将从模块的定义、使用、制作等方面，对模 块进行讲解。

IX 1俳么辦模块

在C语言中，若想使用printfQ函数(该函数与Python中的print()功能相同)，需先在当 前程序中添加"#include <stdio.h>"语句，其中形如“文件名.h”的文件stdio.h在C语言中被 称为头文件，而piimf()函数就是包含在头文件stdio.h中的一个功能函数，程序中只有包含了 stdio.h文件,才能使用printf()等函数。

Python中同样可通过文件包含的方式，使用定义在其他文件中的函数、方法或变量等，只 是Python中将被导入的文件称为"模块(module) 。Python中的模块分为系统模块、第三方 模块和用户自定义模块，它们实质上也是以°py为扩展名的Python文件。

ix2 sfflwwa

Pythoii内置了一些标准模块，其中常用的有sys、os、random、time等。

13-2J箋｝修滇爍

sys模块中包含了与Python解释器相关的属性和函数，常用的属性或函数功能如表134 所示。

表S sys模块常用属性及函数

|  |  |
| --- | --- |
| 属性/函数 | 说 明 |
| sys.argv | 获取命令行参数列表，该列表中的第一个元素是程序自身所在路径 |
| sys.exitQ | 退出当前程序，可为其传递参数，以设置返回值或退出信息，正常退出返回值为0 |

续表

|  |  |
| --- | --- |
| 属性/函数 | 说 明 |
| sys.version | 获取Python解释器的版本信息 |
| sys.path | 获取模块的搜索路径，该属性的初值为环境变量PYT.HONPAT.H的值 |
| sys.platfrom | 返回操作系统平台的名称 |
| sys.stdout | 获取标准输出流对象 |
| sys. stdin | 获取标准输入流对象 |
| sys. stderr | 获取标准错误流对象 |
| • 13.2.2 ©s W | |
| os模块中提供了访问操作系统服务的功能，第9章中给出了 OS模块中部分与文件操作相 | |

关的函数，此外，os模块中常用的属性和函数如表13-2所示。

表13-2 os模块常用属性及函数

|  |  |
| --- | --- |
| 属性/函数• | 说 明 |
| os.makedirs() | 以递归的方式创建多层目录，即若参数中的目录不存在，会主动创建相应目录 |
| os.removedirs() | 若指定目录为空，则删除指定目录，若此后上级目录亦为空，则一并删除上级目录，依此类推 |
| os.stat() | 获取指定文件/目录的属性信息，参数为路径名 |
| os.sep | 获取当前平台的路径分隔符(Windows "W”，Linux下为“/”) |
| os.linesep | 获取当前平台使用的行终止符(Windows下为"\r\n\ Linux下为“W) |
| os.systemQ | 执行系统命令 |
| os.environ | 获取系统的环境变量 |

下面通过示例展示其中部分函数的用法。

(1 ) os„makedirs()

>>> import os

>>> os.makedirs(ndirl\\dir2")

以上命令的功能为在当前目录的血1目录下创建子目录血2,若当前目录下不存在目录 dirl9则先创建血1目录，再在其中创建目录dir2；若diN已存在，则直接在其中创建血2。

(2) os.removedirs()

>>> import os

>>> os.removedirs(°dirl\\dir2")

以上命令的功能为删除dirl目录下的子目录血2,执行此命令后,若目录血1中只有一个 目录血2,则此时查看当前目录，发现目录dirl也被删除；否则dirl目录中的血2被成功删除。 需要注意的是，在执行此命令前需保证待删除目录为空，否则将无法成功删除。

(3 ) os.system()

system()函数的功能是执行系统命令，以Windows中的系统命令cis为例，其用法如下所示。

>>> import os

>» os . system ("cis")

经此操作后命令行中内容被清空。

13-2d3 ©sQpafth 滇鑽 •

OS模块中包含了一个名为path的子模块，该子模块中常用的函数如表93所示。

表13-3 os.path模块常用函数

|  |  |
| --- | --- |
| 属性/函数 | 说 明.• |
| os.path.abspath(path) | 返回规范化的path的绝对路径 |
| os.path.split(path) | 将path分隔为形如(目录，文件名)的二元组并返回 |
| os.path.dimame(path) | 返回path的目录(等于os.path.split()返回结果中的第一个元素) |
| os.path.exists(path) | 若path存在则返回True,否则返回False |
| os.path.getatime(path) | 返回path所指向的文件或者目录的最后存取时间 |
| os.path.getintime(path) | 返回path所指向的文件或目录的最后修改时间 |

下面通过示例展示其中部分函数的用法。

1. os.path.abspath()

>>> import os

>>> os.path.abspath(".")

1C:\\Users\\admin'

观察以上示例执行结果,相对路径“•”被abspafh。函数转换为了绝对uC:\\Users\\adminn。

1. os.path.split。

以uC:\\Users\\admin,，*为例,*split()函数的用法如下:

>>> os.path o split("C:\\Users\\admin")

(!C:\\Users \*, ?admin 1)

观察以上示例的执行结果，split。函数将传入的路径名分隔成了形如“(目录，文件名)” 的二元组。

(3 ) os.path.dimame()

dimame()函数的功能是返回路径中的目录,以“C:\\Users\\adminn为例,dimameQ函数的 用法如下：

>>> os epath.dirname("C:\\Users\\admin") ，C:\\Users1

13b2d4 random Ol

用ndom模块中提供了用于返回随机数的函数，调用其中的函数可在程序中模拟需产生随机 数的事项。『andom模块中的常用函数如表13・4所示。

|  |  |
| --- | --- |
| 表13-4 random模块常用函数 | |
| 属性/函数 | 说 明 |
| random.random() | 参数为空，返回(0,H之间的随机实数 |
| random.randint(x,y) | 返回氏羽之间的整数 |
| random.unifonn(x,y) | 返回［x,y］之间的浮点数 |

13q205

time模块中包含获取当前时间、设置时间等与时间相关的函数，日期可使用以秒为单位的 实数(自1970年1月1日。点开始计算)，或包含9个整数的元组，如：

demo\_tuple = (1992,11,21,12,2,45/0,21,0)

元组中的整数代表的含义依次为：年、月、日、时、分、秒、一周的第几日、一年的第几 日(儒略历)、夏令时。

time模块中定义了一个专门用于表示时间的类time.struct\_time，此类型同样包含年、月、 日等9项信息，示例如下：

time « struct\_time(tm\_year=2 017 r tm\_mon=5 f tm\_mday=18 f tm\_hour=15, tm\_min=4 z tm\_sec=4 6 r tm\_wday=3 f tm\_yday=138 f tm\_isdst=0)

一般情况下,time中获取的时间都以实数、时间字符串或strcut\_time类型输岀。time模块 中的常用函数如表13-5所示。

表13-5 time模块常用函数

|  |  |
| --- | --- |
| 属性/函数 | 说 明 |
| time. asctime( [tuple]) | 获取时间字符串，或将时间元组转换为字符串 |
| time.localtime([secs]) | 以structjime类型输岀本地时间，或将时间secs转换为stnict\_time类型格式 |
| time.mktime(tuple) | 将时间元组转换为秒数 |
| time.sleep(secs) | 进入休眠态，时长由参数secs指定，单位为秒 |
| time.timeQ | 获取当前时间，以单位为秒的实数表示 |
| time. strptime(string [, format]) | 将一个时间格式的字符串解析为时间元组 |
| strftime(fdrmat[, tuple]) | 返回字符串表示的当地时间，格式由format决定 |

下面通过示例展示其中部分函数的用法。

(1 ) time.asctime()

asctimeQ函数的功能为获取当前时间，或将时间元组转换为字符串，该函数的两种用法分 别如下所示。

>» import time

>>> time.asctime()

#获取当前时间

#将时间元组t转换为字符串

?Thu May 18 16:26:21 2017 5

»> t = (2000,3,21,2,21,34,34,366,0) >>> time.asctime(t)

?Sun Mar 21 02:21:34 2000'

(2) time.time()

time()函数的功能为获取以秒数为单位的当前时间，示例如下：

>>> import time

>>> time.time()

1495096290^1932132

(3 ) time.localtime()

localtimeQ函数的功能为:以struct\_time类型输出本地时间，或将时间secs转换为struct\_

time类型格式,示例如下:

>>> import time

>>> time . localtime () # 获取本地时间

time.struct\_time(tm\_year=2017 , tm\_mon=57 tm\_mday=l8 tm\_hour=l6 f tm\_ min=2 8 tm\_sec=4 7 tm\_wday=3 r tm\_yday=138 f tm\_isdst=0)

>>> time . local time (1495096290.1932132) # 转换时间 secs 为 struct\_time 格式

time o struct\_time(tm\_year=2017 z tm\_mon=5 7 tm\_mday=l8 r tm\_hour=l6 f tm\_ min=317 tm\_sec=30 7 tm\_wday=3 r tm\_yday=138 f tm\_isdst=0)

(4) strptime()

strptime ()函数的功能是将一个时间格式的字符串解析为时间元组。Python中格式化字符 串可以使用的日期和时间符号如下所示。

%y：表示两位数的年份(00〜99)。 %Y：表示四位数的年份(000〜9999 ) o %：表示月份(01〜12)。

%d：月内中的一天(01〜31) o

表示24小时制的小时数(0〜23 )。 %M：表示分钟数(00〜59 )。

%S：表示秒数(00〜61)。

%z:表示当前时区的名称。

%a:表示本地简化的星期名称。

%A:表示本地完整的星期名称。

%b:表示本地简化的月份名称。

%B：表示本地完整的月份名称。

%c:表示本地相应的日期表示和时间表ZK。

下面是strptime ()函数使用的示例代码:

>>> import time

>>> time.strptime("30 12 2018n 7 °%d %m %Y")

time.struct\_time(tm\_year=2018 z tm\_mon=12 z tm\_mday=30’ tm\_hour=0 f tm\_

min=0 7 tm\_sec=0 7 tm\_wday=6 f tm\_yday=3 64 7 tm\_isdst=~l)

(5 ) strftime()

strftime ()函数可以将一个时间元组转为指定格式的字符串，示例如下:

»> t = (2018,1,20,10,39,45,1,48,0)

>>> t = time.mktime(t)

>» time . strftime (M%b %d %Y %H : %M: %S" f time . gmtime (t)) Man 20 2018 02 : 39: 45 J

1X3导入模块

Python中可通过import与from...import...这两种方式导入文件。

13Q3 J import " ' ，

使用import格式导入模块时，模块中的所有内容都会被导入到当前程序中，import的用 法如下：

>>> import 模块 1,模块 2,

\*使用import可一次导入多个模块，每个模块名之间使用7分隔。在命令行中导入某个模块， 用户便可调用指定模块中的任意方法，调用模块中方法的方式如下:

»>模块,方法()

以系统模块random以及其中的randint方法为例，import的用法示例如下：

>>> import random

>>> random o randint(1,10)

9

13.3.2 from ^import..

使用import语句导入模块后，每次调用模块中的内容时，都需要添加前缀“模块名， 若某些内容在导入模块的文件中使用的频率较高，抑或模块名较长，使用这种方法显然比较烦 琐。为解决此问题,Python中提供了 “fh)m…import…”语句，该语句可导入模块中的部分内容, 其用法如下：

>» from模块import方法/类/对象

以此语句导入的方法、类和对象等无须添加“模块名前缀，可直接使用。以系统模块 random中的randint方法为例，from- °-import-• .的用法如下:

»> from random import randint

>>> randint(1z10)

2

from-•-import…语句亦可一次导入多个方法(或类、对象)，多个方法之间需用“,”隔开, 示例如下:

>>> from random import randint^ random

>>> randint(1z10)

10

>>> random()

0。9545396343447562

13d3o3 fr©mDQDflmp©rt \*

from... import \*遵循from ^.import…语句的格式，其中通配符指代指定模块中的全 部方法「以random模块为例，在导入“\*之后，用户可使用国ndom模块中的全部方法，示 例如下:

>>> from random import \*

>>> randint(1^10)

4

>>> random()

0^6584572753461784

>>> randrange (0^10z 2)

2

1X4导人W"

使用fh)m…import…语句可简化方法的调用方式，但若导入文件中存在与导入内容重名的 方法、变量或类时，程序执行将会出错。因此，相对而言，使用import语句导入模块更为安全。

Oo5自蒯模块

一般在进行项目开发时，不会将所有代码都放在一个文件中，而是将耦合度较低的多个功 能写入不同的文件中1制作成模块，并在其他文件中以导入模块的方式使用自制模块中的内容。

13d5J識傕圓是鬼滇燥

Python文件中的每个文件都可以作为一个模块存在，文件名即模块名。假设现有一 Python 文件test.py,其中的内容如下：

def test():

print("hello itheima")

a = 10

该文件便可视为一个模块，其中定义的函数test()和对象a都可在导入了该模块的程序或 命令行中使用。

13.5.2 \*

自定义模块的用法与系统模块的用法相同，也通过import语句和from-^import…语句实现。 将模块放在Python解释器所在路径中，在命令行使用import语句导入test模块，并调用

其中的test方法，获取其中的对象a,示例如下：

>>> import test

>» test. test ()

hello itheima

>>> test.a

10

若只需使用其中的於sf方法，则可使用from--import…语句。示例如下：

>>> from test import test

>>> test()

hello itheima

模块搜索路径即Python搜索模块时的路径,这些路径存储于sys模块中的sys.path属性中， 用户可先在解释器中导入sys模块，再查看sys.path的值，示例如下：

>>> import sys

>>> sys.path

[\* \*e \*E:\\Python-3；5.2-embed-win32\\Python3 5.zip \*r 'E:\\Python-3.5.2- embed-win32\\DLLs1r 1E:\\Python-3.5.2-embed-win32\\lib1z!E:\\Python-3.5.2- embed~win32 \*]

sys.path的本质是列表，因此用户可通过列表的内置方法append动态地向sys.path中添加 模块所在路径。假设将路径uE:\\Python\\workspacen添加到搜索路径中，其方法如下：’

»> sys . path. append ("E : \\Python\\workspace1')

再次查看sys.path,其中包含的路径如下"

»> sys .path

[11r \*E:\\Python-3.5.2-embed-win32\\Python35.zip', \*E:\\Python-3.5.2- embed-win32\\DLLs \* 7 5 E:\\Python-3.5 “ 2-embed-win32\\lib1, \* E:\\Python-3.5.2- embed-win32 \* z ? E:\\Python\\workspace 1]

由以上打印结果可知，新路径添加成功。

1 1X6模块属雄

13.0J 一剝一屬惟■ •

Pythori模块的开头通常会定义一个\_疝\_属性，该属性的本质是元组，其中包含的元素 决定了在使用fh)m…import \*语句导入模块内容时通配符\*所包含的内容，若\_\_all\_中只包含 模块的部分内容，那么from-e-import \*语句只会将\_all\_中包含的内容导入程序。

假设当前用户自定义了名为calc的模块，该模块中包含的内容如下：

all = [naddf\ °subn]

def

def

def

def

add(azb): return a + sub(azb): return a - mult(a7 b):

return *a \**

div(a7b): if(b):

return

else :

print("error")

在命令行使用"'from calc import 语句导入模块后, 下所示。

分别调用其中的方法，调用结果如

»>

»>

from calc import \* add(3z 4)

#导入模块 #add调用

»>

sub(3,4)

#sub调用

-1

mult (3Z 4)

Traceback (most recent

File °<stdin>"r line

NameError: name 1 mult1

>>> div(3,4)

Traceback (most recent

File "<stdin>"r line

»>

#mult调用失败

call last):

1, in <module> is not defined

call last):

1z in <module>

#div调用失败

NameError: name \* div1 is not defined

结合模块calc中的内容，由调用结果可知，未包含在 序中。

all\_\_中的函数没有被导入当前程

\_all\_常用于想导入模块中的部分内容时，它只对使用通配符\*的导入方法有影响，换言 之,不包含在—all\_中的内容仍可通过其他方法导入,如在当前程序中导入divide和multiply 方法,示例如下:

>>> from fact import divide,multiply

>>> divide(3,4)

0.75

>>> multiply(3,4)

12

或者使用import语句导入模块后，使用“模块名\*方法”的格式调用函数，示例如下:

>>> import calc

>>> calc.divide(3,4)

0.75

>>> calc.multiply(3z 4)

12

由以上结果可知,divide方法和multiply方法导入成功。

13n®a2 name

在较大型的项目开发中，一个项目通常由多名开发人员共同开发，每名开发人员负责不同 的模块。为了保证自己编写的程序在整合后可以正常运行，开发人员通常需在整合前额外编写 测试代码，对自己负责的模块进行测试。

然而，对整个项目而言，这些测试代码是无用的。Python中设置属性，利用 此属性，可保证每个单独模块中的测试代码只在启动运行当前模块被执行。

—name 属性通常与if条件语句一起使用，若当前模块是启动模块,则其—name 的值 为；若该模块被其他程序导入，则\_name\_的值为文件名。

下面以calc模块为例，展示\_nanw\_属性的用法。在calc模块后追加如下内容：

1. if name == ° main ° : // 执行测试代码
2. print(add(3,4))
3. print(sub(3,4))
4. print(mult(3r 4))
5. print(div(3,4))
6. else:
7. print( name )

执行该模块本身，执行结果如下：

7

-1

12

0»75

重新打开命令窗口，使用“imported”语句导入calc模块，则命令行打印的信息如下m

calc

模块导入概悝

13D7n1蓦入蒔慳纏進

模块在被导入时，会被执行一次，这个过程又称加载。一个模块即便被多次导入程序，也 只会被加载一次。

以自定义模块fact为例进行测试，该模块中的内容如下：

def j iec(n): if (n>l): return n\*j iec(n-1) else :

return 1

result = j iec (5)

print(result)

在命令行中多次导入fact模块,示例如下:

>>> import fact

12 0

>>> import fact

>>> import fact

观察以上信息，只有第一次导入模块时模块的执行结果被打印。另外打开一个命令行窗口， 在其中导入多次fact模块，导入结果如下所示。

>>> import fact

120

>>> import fact

>>> import fact

结合两个命令行窗口中的信息可知，当模块被导入到一个程序中时，该模块只会加载一次。

模块之间是允许相互导入的，模块的这个特性可以防止相互导入时出现死循环。假如现有 模块A和模块这两个模块中的内容分别如下所示。

A. py

def test ():

print ( ")

test ()

import B

B.test ()

b py

def test ():

print ( " B ° )

test ()

import A

A。test ()

模块A和模块B互相导入对方，若模块在每次被导入时都要执行，那么一旦模块A被执行, 模块A便会导入模块B,而模块B又会导入模块A……如此不断地进行导入，程序便会进入死 循环，这显然是不可取的。

但因为Python中的模块有“即便被多次导入程序,也只会被加载一次”这个特性，所以 即便A、B模块中相互导入对方，程序仍能正常执行。

模块纔存

细心的读者可能发现，当模块被导入程序并执行后，模块所在的路径中产生了一个名为— pycache\_\_的文件夹,该文件夹中又包含一个与模块同名，但扩展名为.pyc的文件,该文件是 模块的缓存文件。

再次使用模块时，实际上程序导入的并非模块本身，而是模块的缓存文件“文件名°pyc”， 如此可提升模块加载的效率。当然这并不是绝对的，若模块被更新，那么在下次导入时Python 会导入模块本身，并更新模块的缓存文件，保证模块与缓存的同步更新。

；I3o9 Fythomi

为了更好地组织模块，开发人员通常选择将项目中的模块划分为包。简单来说，包(package ) 是一个包含文件的目录，该目录中还包含一些模块或子包。简单的包结构如下所示。

package

|— init .py

I一— module\_al.py

' “ • module\_a2.py

' ”package\_b

|— init «py

L—— module\_b 6py

包的存在使整个项目更富有层次，也可在一定程度上避免合作开发中模块重名的问题。包 中的\_init\_.py文件可以为空，但必须存在，否则包将退化为一个普通目录。

nao蚀的导人

包的导入也通过import语句与from...import...语句实现。假设13.9节所示包结构的每个 模块中都只包含可打印文件名的函数，则若要在与package包处于同级目录的文件main.py中 导入包中的内容，可使用如下几种方法。

13J0J omport ®A

使用import语句导入包中模块时，需在模块前添加包名，示例如下：

»> import package .module\_al # 导入包中指定模块

此时包package中的模块module al被导入程序中，若要使用该模块中的print\_file\_name() 函数，则可使用以下方法：

»> package .module\_al.print\_file\_name () # 使用 module\_al 模块中的函数

函数调用结果如下所示。

module\_al #打印结果

由以上示例可知，使用import语句导入包中模块的格式为uimport包名。模块名”；使用 已导入模块中的方法时，格式为“包名。模块•方法”。

同理,若要调用子包package\_b中模块module\_b W print\_file\_name\_other()函数,示例如下:

>>> import package »package\_b.module\_b

>>> package。package\_b . module\_b . print\_\_file\_name\_other () # 打印结果

module\_b

13J0d2 fronnuJimpw臨慝入.

使用from ^.import…导入包中模块包含的内容时，同样需在模块前添加包名，示例如下：

# 从包 package 的 module\_al 模块中导入 print\_file\_name ()函数

>>> from package amodule\_al import print\_fi 1e\_name

执行以上导入语句后，便可直接通过方法名调用module\_al模块中的print\_file\_name()函数。 示例如下:

>>> print\_file\_name ()

module\_al #打印结果一

13J0.3 —術盹一眼歐宝俸

使用fh)m…import…语句导入模块中内容时，可利用通配符\*导入指定模块中的全部内容 (当然\*代表的内容可以通过模块的—疝—属性设定)。包中\_init\_°py文件的功能与模块 中\_疝\_属性的功能相同，该文件中也定义了一个\_all\_属性，用户可通过编辑\_init\_.py 文件中的\_all\_属性来设置使用from...import \*语句中\*指代的内容。

编辑包package中的\_init\_°py文件，写入\_all\_属性并保存:

\_all\_=["modu丄㊀\*]

再通过from...import \*语句导入package中的模块:

>>> from package import \*

调用模块 module al 中的函数 print\_file\_name():

>>> print\_file\_name ()

输出的内容如下：

module\_al

调用模块module\_a2中的函数print\_file\_name\_other()9由于模块module\_a2并未添加到— init\_.py文件的—疝\_属性中，因此本次调用失败，打印如下错误信息。

>» print\_file\_name\_other ()

Traceback (most recent call last):

File "<stdin>°, line 1r in <module>

NameError: name 1 module\_a2f is not defined

用户可通过编辑子包中的\_init\_°py文件，再以"from包•子包import 的方式导入子 包中的指定内容。

若模块中未声明—疝—属性的内容，使用form…import \*导入模块时默认导入模块中的 全部内容；与模块中的\_诳1\_属性不同的是，若包中的\_init\_.py未对属性做出声明， 则使用from…import \*导入的内容为空。

1X11

用户可将自己编写的模块打包为可安装的压缩包，分享给其他开发人员。本节将对模块的 打包方式进行讲解。

13J1J常褰福刨缱的

将待发布的包放在与setup.py文件同级的目录中，目录结构如下所示。

H setup.py

|— — suba

I H~ aa.py

I I bb.py

| ' init . py

L—~- subb

| cc . py

| dd. py

L— init ,py

13J1-2 s@tapDpy ：

编辑setup.py脚本文件,steup.py文件中包含的内容如下:

from distutils <. core import setup setup(

name="itheiman f version=°1.0"z description=°itheima belongs to itcasf\* 7 author="itcast"?

py\_modules=[!suba.aa v z 3 suba.bb \ \* subb.cc \* z !subb « dd,]

*)*

其中的setup为一个函数，包含5个参数，依次为：包名、版本号、描述信息、作者以及 包中含有的模块，实际上setup函数中不只有这些参数，但这些参数也不是必需的，用户可有 选择地进行设置。

13J13纜爍樗還 ■

在当前路径下打开命令行窗口，使用buiid命令构建模块，示例如下：

python setup.py build

经此操作后，当前目录中会创建一个名为build的文件夹，其中包含一个名为lib的目录， 该目录中存储了待发布模块的备份。build文件夹中的目录结构如下所示。

|——- build

I 」一lib

I |——- suba

I I aa»py

I I F-~ bb.py

I I L init .py

| L— subb

I I-~ cc .py

I I dd. py

' -init . py

setup.py

suba

-~— aa o py

-——bb . py

—— init . py

subb

—cc.py

-~~- dd. py

~™ init , py

13J14滇爍轲制

用户可通过sdist命令建立存放文件，示例如下:

python setup.py sdist

在执行此命令时，除正常的描述信息外，命令行中还可能打印如图13』所示的警告 信息。

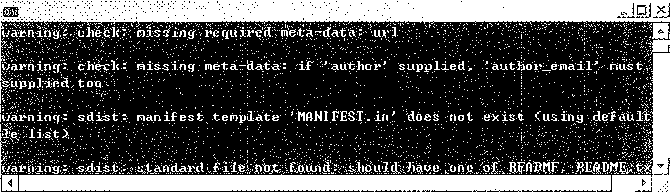


图13』sdist警告信息

这是因setupqy脚本中wtup()函数的部分参数缺省所致，用户可忽略这些警告，亦可在之 前的setup函数中设置 丽、author\_email等参数。

经此操作后,当前目录中会出现一个名为dist的目录和一个MANIFEST文件,dist目录中 包含一个名为“itheim务L0.zip”的压缩包，其他人在获取这个压缩包后，可先将其解压，再利 用其中的steup.py脚本安装Python包；MANIFEST文件中包含所有文件的列表。

此时当前目录的结构如下所示。

build

—lib

|—~— suba

I I aa.py

I I bb.py

I ' init . py

L-~~— subb

I cc.py

~~I~~  dd. py

L—— init <.py

dist

—itheima-1.0.zip

MANIFEST

setup.py

suba

—aa.py

— bb .py

—- init o py

subb

cc.py

—dd^py

*■~一* init opy

至此，模块的打包工作全部完成，实际上，这个过程也是用户程序的打包过程，此时用户 可将Nip格式的压缩文件发布到网络等场所，供其他开发人员使用。

13ol2模块安装

除标准模块和用户自定义模块外，用户还可从外部获取并安装第三方模块。需要注意的是, 由于第三方模块的开发环境可能各不相同，因此其安装方式也不相同，有些模块可通过双击直 接安装(如pygame的Windows版本)，但也有些模块并不提供这种安装方式。

实际上，对其他开发人员来说，13」1节打包发布的模块就是一个第三方模块，下面以此 模块为例，讲解第三方模块的安装方法。

(1)获取第三方模块的压缩包，对压缩包进行解压。解压后的目录结构如下所示。

setup.py

suba —aa.py

—bb.py

— init . py

subb

LE- cc opy ~~卜~~dd. py

L init \_.py

|———PKG-INFO

setup.py

(2 )在当前目录中打开命令行窗口，使用install命令安装模块，示例如下:

Python setup.py install

经此操作后，模块会被安装到当前目录下，用户也可通过"^prefix"选项设置模块安装路径, 示例如下:

Python setup .py install -perfix=E : \\Python 使用此条命令，该模块将被安装到E盘的Python目录中。