python的模块加载和路径查找

2016-05-28

三月沙 原文链接

深入这个问题之前，我们需要理解几个概念：

module：模块， 一个 `py` 文件或以其他文件形式存在的可被导入的就是一个模块

package：包，包含有 `__init__` 文件的文件夹

relative path：相对路径，相对于某个目录的路径

absolute path：绝对路径，全路径

路径查找：python 解释器查找被引入的包或模块

python 执行时是如何查找包和模块的

1.python 执行一个文件

python 执行一个文件，无论执行的方式是绝对路径还是相对路径，解释器都会把文件所在的目录加入到系统查找路径中，也就是 `sys.path` 这个 `list` 中，而 `sys.path` 又是由系统的 python 环境变量决定的。

```
#test.py
import os
import sys

print sys.path[0]
```

执行这个文件：

```
python test.py
python /Users/x/workspace/blog-code/p2016_05_28_python_path_find
test.py
```

相对路径和绝对路径输出相同的结果。test.py 所在的文件夹都会被加入 `sys.path` 的首位，注意这里是首位，也就是索引为0的位置。

解释器执行时，首先搜索 `built-in module` ，也就是解释器查找模块最先查找的是 `built-in module` ，其次才会搜索 `sys.path` 所包含的路径。这样的查找顺序将会引起同名包或模块被遮蔽的问题。

A sample

```
├─── os.py
├─── test2.py
├─── redis.py
```

```
#test2.py

import os
from redis import Redis
```

执行 `test2.py` 文件

```

Traceback (most recent call last):
  File "/Users/x/workspace/blog-code/p2016_05_28_python_path_find/test2.py", line 1, in <mod
    from redis import Redis
ImportError: cannot import name Redis
```

由于 `os` 是 `built-in module` ，即使在同目录下有同名模块，解释器依然可以找到正确的 `os` 模块，而 `redis` 属于第三方模块，默认安装位置是 python 环境变量中的 `site-packages` 下，解释器启动之后会将此目录加入 `sys.path` ，按照上面所说的查找顺序，优先在执行文件所在的目录查找，由于其在 `sys.path` 的首位，因而本地的 `redis` 被导入。

2.交互式执行环境

进入交互式执行环境，解释器会自动把当前目录加入 `sys.path` , 这时当前目录是以相对路径的形式出现在 `sys.path` 中：

```
>>> import os.path
>>> import sys
>>> os.path.abspath(sys.path[0])
'/Users/x/workspace/blog-code'
>>>
```

除此之外，其他与执行一个文件是相同的。

深入 `__file__` 变量

> python doc: `__file__` is the pathname of the file from which the module was loaded, if it was loaded from a file. 如果一个模块是从文件加载的， `__file__` 就是该模块的路径名。

顾名思义，当模块以文件的形式出现 `__file__` 指的是模块文件的路径名，以相对路径执行 `__file__` 是相对路径，以绝对路径执行 `__file__` 是绝对路径。

```
#test.py
print __file__
```

相对路径执行

```
python test3.py
test3.py
```

绝对路径执行

```

python /Users/x/workspace/blog-code/p2016_05_28_python_path_find/test3.py
```

为了保证 `__file__` 每次都能准确得到模块的正确位置，最好再取一次绝对路径 `os.path.abspath(__file__)` 。

进入交互式 shell ，输入 `__file__` 会报错误

```
>>> __file__
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name '__file__' is not defined
```

这是因为当前交互式shell的执行并不是以文件的形式加载，所以不存在 `__file__` 这样的属性。

和 `__file__` 相似的 sys.argv[0]

与 `__file__` 相似的一个变量是 `sys.argv[0]` ，区别是它用来获取主入口执行文件的变量。

A sample

```
#test.py
import sys
print __file__
print sys.argv[0]
```

以上俩个print输出相同的结果，因为主执行文件和 `__file__` 所属的模块是同一个。当我们改变入口文件，区别就出现了。

A sample

```
#test.py
import sys
print __file__
print sys.argv[0]

#test2.py
import test
```

执行 `test2.py`

```
/Users/x/workspace/blog-code/p2016_05_28_python_path_find/child/test.py #__file__
test2.py #sys.argv[0]
```

总的来说， `sys.argv[0]` 是获得入口执行文件路径， `__file__` 是获得任意模块文件的路径。

sys.modules 的作用

既然python是在 `sys.path` 中搜索模块的，那载入的模块存放在何处？答案就是 `sys.modules` ，是存放已载入模块的地方。模块一经载入，python 会把这个模块加入 `sys.modules` 中供下次载入使用，这样可以加速模块的引入，起到缓存的作用。

```
>>> import sys
>>> sys.modules['tornado']
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'tornado'
>>> import tornado
>>> sys.modules['tornado']
<module 'tornado' from '/Users/x/python_dev/lib/python2.7/site-packages/tornado/__init__.pyc
```

前面说过python 解释器启动之后，会把预先载入 `built-in module` ，可以通过 `sys.modules` 验证。

```
>>> sys.modules['os']
<module 'os' from '/Users/x/python_dev/lib/python2.7/os.pyc'>
>>>
```

获取模块的路径

借助 `sys.modules` 和 `__file__` , 可以动态获取所有已加载模块目录和路径：

```
>>> import os
>>> os.path.realpath(sys.modules['os'].__file__)
'/Users/x/python_dev/lib/python2.7/os.pyc'
>>> import tornado
>>> os.path.realpath(sys.modules['tornado'].__file__)
'/Users/x/python_dev/lib/python2.7/site-packages/tornado/__init__.pyc'
```

```
def get_module_dir(name):
    path = getattr(sys.modules[name], '__file__', None)
    if not path
        raise AttributeError('module %s has not attribute __file__'%name)
    return os.path.dirname(os.path.abspath(path))
```

from 和 import 语句

知道了 python 是如何搜索模块和保存已导入模块，我们再说说 python 模块的导入

1.导入顺序

python 包的导入顺序是

```
系统包 --> 同目录 -- > sys.path
```

因此同名的包，系统包优先级最高 > 同目录 > sys.path，之所以有这样的差别是因为当前执行目录会优先加入 `sys.path` 的首位。

2.导入时执行

当导入一个模块时，模块的顶层变量会被执行，包括全局变量，类以及函数的声明，因此在编写模块时一定要注意消除副作用(函数的调用)。

3.import 语句 和 from 语句

能被 import 的包括：package，package 中的模块，模块中的变量。影响 import 的属性是 `__all__` ， `__all__` 是个list，能够影响被 package 中 以 `from package import *` 被导出的模块，即定义在 `__all__` 中的模块才会被 `from package import *` 导出。

summary

python 的模块导入，加载和查找还有很多可以说说的地方，尤其是动态 import， 对应python中的关键字是 `__import__` ，感兴趣的同学可以研究一下 `tornado.util` 模块下的 `import_object` 。

文中所有代码见：github

ps: 转载请注明出处

技术 #python