25. 休养生息-包

本节主要内容:

1. from xxx import xxx知识点补充
2. 包

一. 知识点补充.

我们现在知道可以使用import和from xxx import xxx来导入一个模块中的内容. 那有一种特殊的写法: from xxx import * 我们说此时是把模块中的所有内容都导入. 注意, 如果模块中没有写出__all__ 则默认所有内容都导入. 如果写了__all__ 此时导入的内容就是在__all__列表中列出来的所有名字.

```
# haha.py

__all__ = ["money", "chi"]
money = 100

def chi():
    print("我是吃")

def he():
    print("我是呵呵")



# test.py

from haha import *

chi()
print(money)
# he() # 报错
```

二. 包

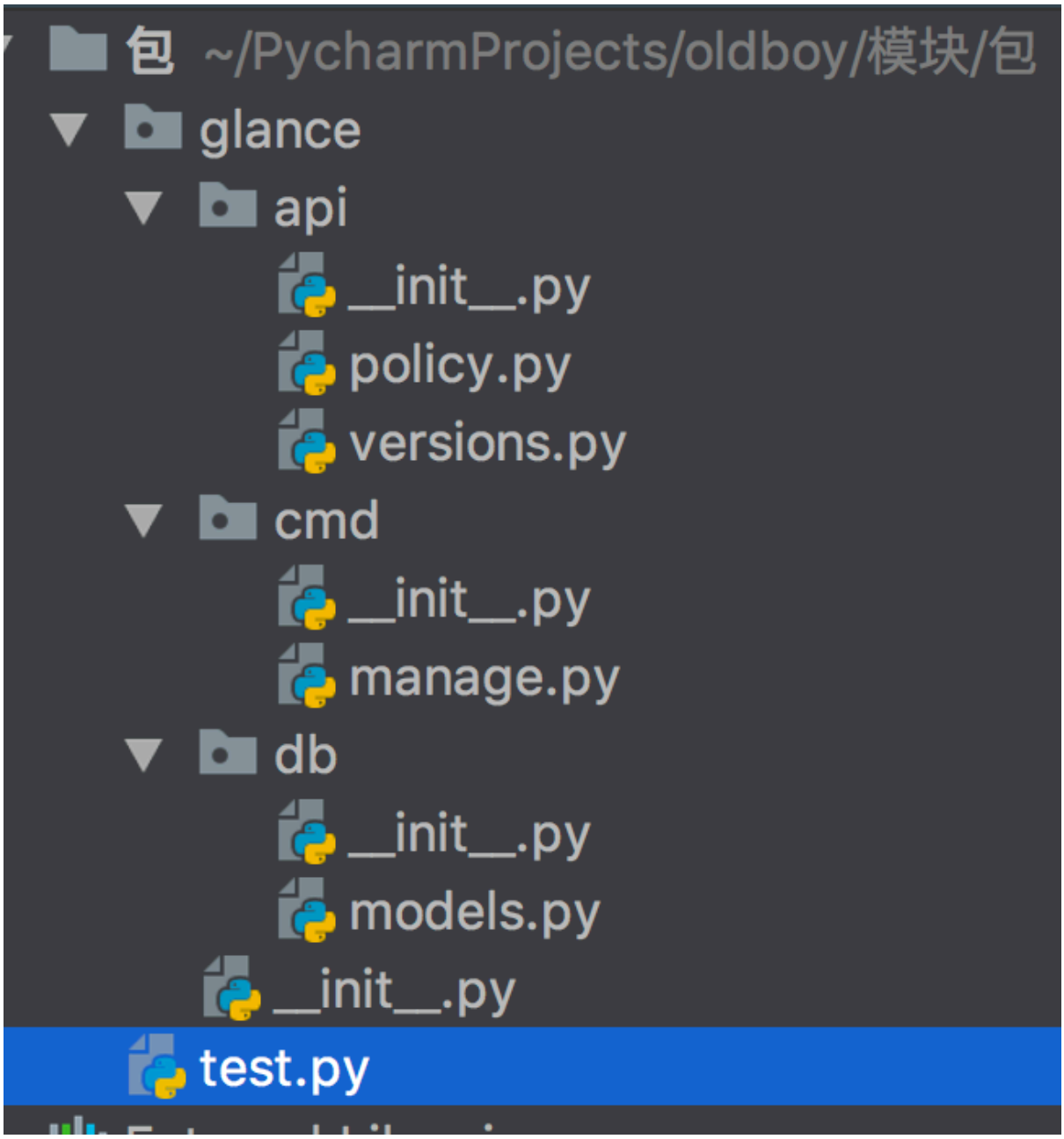
包是一种通过 '.模块名'来组织python模块名称空间的方式. 那什么样的东西是包呢? 我们创建的每个文件夹都可以被称之为包. 但是我们要注意, 在python2中规定. 包内必须存在__init__.py文件. 创建包的目的不是为了运行, 而是被导入使用. 包只是一种形式而已. 包的本质就是一种模块

为何要使用包? 包的本质就是一个文件夹, 那么文件夹唯一的功能就是将文件组织起来, 随着功能越写越多, 我们无法将所有功能都放在一个文件中, 于是我们使用模块去组织功能, 随着模块越来越多, 我们就需要用文件夹将模块文件组织起来, 以此来提高程序的结构性和可维护性

首先, 我们先创建一些包. 用来作为接下来的学习. 包很好创建. 只要是一个文件夹, 有__init__.py就可以.

```
import os
os.makedirs('glance/api')
os.makedirs('glance/cmd')
os.makedirs('glance/db')
l = []
l.append(open('glance/__init__.py','w'))
l.append(open('glance/api/__init__.py','w'))
l.append(open('glance/api/policy.py','w'))
l.append(open('glance/api/versions.py','w'))
l.append(open('glance/cmd/__init__.py','w'))
l.append(open('glance/cmd/manage.py','w'))
l.append(open('glance/db/__init__.py','w'))
l.append(open('glance/db/models.py','w'))
map(lambda f:f.close() ,l)
```

创建好目录结构



我们接下来给每个文件中添加一些方法:

```
#policy.py
def get():
    print('from policy.py')

#versions.py
def create_resource(conf):
    print('from version.py: ',conf)

#manage.py
def main():
    print('from manage.py')
```

```
#models.py
def register_models(engine):
    print('from models.py: ',engine)
```

接下来. 我们在test中使用包中的内容. 并且, 我们导入包的时候可以使用import或者from xxx import xxx这种形式.

首先, 我们看import

```
import glance.db.models
glance.db.models.register_models('mysql')
```

没问题, 很简单, 我们还可以使用from xxx import xxx 来导入包内的模块

```
from glance.api.policy import get
get()
```

也很简单, 但是, 要注意. from xxx import xxx这种形式, import后面不可以出现"点" 也就是说from a.b import c是ok的. 但是 from a import b.c 是错误的

好了, 到目前为止, 简单的包已经可以使用了. 那包里的__init__.py是什么鬼? 其实. 不论我们使用哪种方式导入一个包, 只要是第一次导入包或者是包的任何其他部分, 都会先执行__init__.py文件. 这个文件可以是空的. 但也可以存放一些初始化的代码. (随意在glance中的__init__.py都可以进行测试)

那我们之前用的from xxx import *还可以用么? 可以. 我们要在__init__.py文件中给出__all__来确定* 导入的内容.

```
print("我是glance的__init__.py文件. ")
x = 10
def hehe():
    print("我是呵呵")

def haha():
    print("我是哈哈")

__all__ = ['x', "hehe"]
```

test.py

```
from glance import *

print(x)    # OK
hehe()      # OK
haha()      # 报错. __all__里没有这个鬼东西
```

接下来, 我们来看一下绝对导入和相对导入, 我们的最顶级包glance是写给别人用的. 然后再glance包内部也会有彼此之间互相导入的需求, 这时候就又绝对导入和相对导入两种方式了.

1. 绝对导入: 以glance作为起始

2. 相对导入: 用. 或者..作为起始

例如, 我们在glance/api/version.py中使用glance/cmd/manage.py

```
# 在glance/api/version.py

#绝对导入
from glance.cmd import manage
manage.main()

#相对导入
# 这种情形不可以在versions中启动程序.
#  attempted relative import beyond top-level package
from ..cmd import manage
manage.main()
```

测试的时候要注意. python包路径跟运行脚本所在的目录有关系. 说白了. 就是你运行的py文件所在的目录. 在python中不允许你运行的程序导包的时候超过当前包的范围(相对导入). 如果使用绝对导入. 没有这个问题. 换个说法. 如果你在包内使用了相对导入. 那在使用该包内信息的时候. 只能在包外面导入.

接下来. 我们来看一个大坑. 比如. 我们想在policy中使用verson中的内容.

```
# 在policy.py
import versions
```

如果我们程序的入口是policy.py 那此时程序是没有任何问题的. 但是如果我们在glance外面import了glance中的policy就会报错 原因是如果在外面访问policy的时候. sys.path中的路径就是外面. 所以根本就不能直接找到versions模块. 所以一定会报错:

```
ModuleNotFoundError: No module named 'versions'
```

在导包出错的时候. 一定要先看sys.path 看一下是否真的能获取到包的信息.

最后, 我们看一下如何单独导入一个包.

```
# 在test.py中
import glance
```

此时导入的glance什么都做不了. 因为在glance中的__init__.py中并没有关于子包的加载. 此时我们需要在__init__.py中分别取引入子包中的内容.

1. 使用绝对路径
2. 使用相对路径

包的注意事项:

1. 关于包相关的导入语句也分为import和from xxx import xxx两种, 但无论使用哪种, 无论在什么位置, 在导入时都必须遵循一个原则: 凡是在导入时d带点的. 点左边都必须是一个包. 否则报错. 可以带一连串的点. 比如a.b.c

2. import导入文件时. 产生名称空间中的名字来源于文件, import 包, 产生的名称空间中的名字同样来源于文件, 即包下的__init__,py, 导入包本质就是在导入该文件

3. 包A和包B下有同名模块也不会冲突, 如A.a和B.a来自两个名称空间