24. 休养生息-模块

本节主要内容:

1. 模块
2. import
3. from xxx import xxx

一. 模块

首先,我们先看一个老生常谈的问题. 什么是模块. 模块就是一个包含了python定义和声明的文件, 文件名就是模块的名字加上.py后缀. 欢聚话说我们目前写的所有的py文件都可以看成是一个模块但是我们import加载的模块一共分成四个通用类别:

1. 使用pyhton编写的py文件
2. 已被变异为共享库或者DLL或C或者C++的扩展
3. 包好一组模块的包.
4. 使用c编写并连接到python解释器的内置模块

为什么要使用模块? 为了我们写的代码可以重用. 不至于把所有的代码都写在一个文件内. 当项目规模比较小的时候. 完全可以使用一个py搞定整个项目的开发. 但是如果是一个非常庞大的项目. 此时就必须要把相关的功能进行分离. 方便我们的日常维护. 以及新项目的开发.

如何使用模块? 我们已经用过很多模块了. 导入模块有两种方式

1. import 模块
2. from xxx import xxxx

二. import

首先. 我们先看import, 在使用import的时候, 我们先创建一个yitian.py. 在该文件中创建一些武林前辈和一些打斗场景, 代码如下.

```
print("片头曲.  啊! 啊~ 啊!  啊. 啊啊啊啊啊啊啊...")

main_person_man = "张无忌"
main_person_woman = "赵敏"

low_person_man_one = "成昆"
low_person_man_two = "周芷若"

def fight_on_light_top():
    print("光明顶大战", main_person_man, "破坏了", low_person_man_one, "的大阴
```

```
谋")

def fight_in_shaolin():
    print("少林寺大战", main_person_man, "破坏了", low_person_man_two, "的大阴
谋")
```

接下来, 金庸上场.

```
import yitian

print(yitian.main_person_man)      # 使用模块中定义好的名字
print(yitian.low_person_man_one)

yitian.fight_in_shaolin()    # 调用模块中的函数
yitian.fight_on_light_top()
```

此时我们在金庸模块中引入了yitian模块.

在Python中模块是不能够重复导入的. 当重复导入模块时. 系统会根据sys.modules来判断该模块是否已经导入了. 如果已经导入. 则不会重复导入

```
import sys
print(sys.modules.keys())   # 查看导入的模块.

import yitian   # 导入模块. 此时会默认执行该模块中的代码
import yitian   # 该模块已经导入过了. 不会重复执行代码
import yitian
import yitian
import yitian
import yitian
```

导入模块的时候都做了些什么? 首先. 在导入模块的一瞬间. python解释器会先通过sys.modules来判断该模块是否已经导入了该模块. 如果已经导入了则不再导入. 如果该模块还未导入过. 则系统会做三件事.

1. 为导入的模块创立新的名称空间
2. 在新创建的名称空间中运行该模块中的代码
3. 创建模块的名字. 并使用该名称作为该模块在当前模块中引用的名字.

我们可以使用globals来查看模块的名称空间

```
print(globals())

打印结果:
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x10bbcf438>, '__spec__': None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>, '__file__':
'/Users/sylar/PycharmProjects/oldboy/模块/模块/金庸.py', '__cached__': None,
'yitian': <module 'yitian' from '/Users/sylar/PycharmProjects/oldboy/模块/模
```

```
块/yitian.py'>, 'sys': <module 'sys' (built-in)>}
```

注意. 由于模块在导入的时候会创建其自己的名称空间. 所以. 我们在使用模块中的变量的时候一般是不会产生冲突的.

```
import yitian

main_person_man = "胡一菲"

def fight_in_shaolin():
    print(main_person_man, "大战曾小贤")

print(yitian.main_person_man) # 张无忌
print(main_person_man)  # 胡一菲

yitian.fight_in_shaolin()   # 倚天屠龙记中的
fight_in_shaolin()  # 自己的
```

注意. 在模块中使用global. 我们之前说global表示把全局的内容引入到局部. 但是. 这个全局指的是py文件. 换句话说. global指向的是模块内部. 并不会改变外部模块的内容

```
模块 yitian 中:


print("片头曲.  啊! 啊~ 啊!  啊. 啊啊啊啊啊啊啊...")

main_person_man = "张无忌"
main_person_woman = "赵敏"

low_person_man_one = "成昆"
low_person_man_two = "周芷若"

def fight_on_light_top():

    print("光明顶大战", main_person_man, "破坏了", low_person_man_one, "的大阴谋")

def fight_in_shaolin():
    global low_person_man_two       # 注意看, 此时的global是当前模块. 并不会影响其他模块
    low_person_man_two = "战五渣"
    print("少林寺大战", main_person_man, "破坏了", low_person_man_two, "的大阴谋")

调用方:
import yitian

low_person_man_two = "刘海柱"

yitian.fight_in_shaolin()
```

```
print(yitian.low_person_man_two)     # 战五渣

print(low_person_man_two)    # 刘海柱. 并没有改变当前模块中的内容. 所以模块内部的global只是用于模块内部
```

特别特别要注意. 如果我们在不同的模块中引入了同一个模块. 并且在某一个模块中改变了被引入模块中的全局变量. 则其他模块看到的值也跟着边. 原因是python的模块只会引入一次. 大家共享同一个名称空间

```
金庸:
import yitian
yitian.main_person_man = "灭绝师太"

金庸二号:
import yitian
import 金庸

print(yitian.main_person_man) # 灭绝师太.
```

上述问题出现的原因:

1. 大家共享同一个模块的名称空间.
2. 在金庸里改变了主角的名字

如何解决呢?

首先. 我们不能去改python. 因为python的规则不是我们定的. 只能想办法不要改变主角的名字. 但是. 在金庸里我就有这样的需求. 那此时就出现了. 在金庸被执行的时候要执行的代码. 在金庸被别人导入的时候我们不想执行这些代码. 此时, 我们就要利用一下__name__这个内置变量了. 在Python中. 每个模块都有自己的__name__ 但是这个__name__的值是不定的. 当我们把一个模块作为程序运行的入口时. 此时该模块的__name__是"__main__" , 而如果我们把模块导入时. 此时模块内部的__name__就是该模块自身的名字

```
金庸:
print(__name__)
# 此时如果运行该文件. 则__name__是__main__

金庸二号:
import 金庸
#此时打印的结果是"金庸"
```

我们可以利用这个特性来控制模块内哪些代码是在被加载的时候就运行的. 哪些是在模块被别人导入的时候就要执行的. 也可以屏蔽掉一些不希望别人导入就运行的代码. 尤其是测试代码.

```
if __name__ == '__main__':
    yitian.main_person_man = "灭绝师太"     # 此时, 只有从该模块作为入口运行的时候才会把main_person_man设置成灭绝师太
    print("哇哈哈哈哈哈")     # 只有运行该模块才会打印. import的时候是不会执行这里的代码的
```

我们还可以对导入的模块进行重新命名:

```
import yitian as yt     # 导入yitian. 但是名字被重新命名成了yt. 就好比变量赋值一样.
a = 1    b = a


yt.fight_in_shaolin()   # 此时可以正常运行
# yitian.fight_in_shaolin()   # 此时程序报错. 因为引入的yitian被重命名成了yt

print(globals())
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x103209438>, '__spec__': None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>, '__file__':
'/Users/sylar/PycharmProjects/oldboy/模块/模块/金庸.py', '__cached__': None,
'yt': <module 'yitian' from '/Users/sylar/PycharmProjects/oldboy/模块/模
块/yitian.py'>}
```

一次可以引入多个模块

```
import time, random, json, yitian
```

正确的导入模块的顺序:

1. 所有的模块导入都要写在最上面. 这是最基本的
2. 先引入内置模块
3. 再引入扩展模块
4. 最后引入你自己定义的模块

三. from xxx import xxx

第一大块关于import就说这么多. 接下来. 我们来看from xxx import xxx这种导入模块的效果. 在使用from的时候, python也会给我们的模块创建名称空间. 这一点和import是一样的. 但是from xxx import xxx的时候. 我们是把这个空间中的一些变量引入过来了. 说白了. 就是部分导入. 当一个模块中的内容过多的时候. 我们可以选择性的导入要使用的内容.

```
from yitian import fight_in_shaolin
fight_in_shaolin()
```

此时是可以正常运行的. 但是我们省略了之前的模块.函数() 直接函数()就可以执行了, 并且from语句也支持一行语句导入多个内容.

```
from yitian import fight_in_shaolin, fight_on_light_top, main_person_man
fight_in_shaolin()
fight_on_light_top()
print(main_person_man)
```

同样支持as

```
from yitian import fight_in_shaolin, fight_on_light_top, main_person_man as
```

```
big_lao
fight_in_shaolin()
fight_on_light_top()
print(big_lao)
```

最后. 看一下from的坑. 当我们从一个模块中引入一个变量的时候. 如果当前文件中出现了重名的变量时. 会覆盖掉模块引入的那个变量.

```
from yitian import main_person_man
main_person_man = "超级大灭绝"
print(main_person_man)
```

所以. 不要重名. 切记. 不要重名! 不仅仅是变量名不要重复. 我们自己创建的py文件的名字不要和系统内置的模块重名. 否则. 引入的模块都是python内置的模块. 切记, 切记.