自定义模块

除了使用系统提供的内置模块以外，我们还能自己写一个模块供自己的程序使用。一个py文件就是一个模块，所以，自定义模块很简单，基本上相当于创建一个py文件。但是，需要注意的是，如果一个py文件要作为一个模块被别的代码使用，这个py文件的名字一定要遵守标识符的命名规则。

模块的查找路径

创建一个模块非常简单，安装标识符的命名规则创建一个py文件就是一个模块。但是问题是，我们需要把创建好的这个py文件放在哪个位置，在代码中使用 `import` 语句才能找到这个模块呢?

Python内置sys模块的path属性，列出了程序运行时查找模块的目录，只需要把我们创建好的模块放到这些任意的一个目录里即可。

```
import sys
print(sys.path)
[
 'C:\\Users\\chris\\Desktop\\Test',
 'C:\\Users\\chris\\AppData\\Local\\Programs\\Python\\Python37\\python37.zip',
 'C:\\Users\\chris\\AppData\\Local\\Programs\\Python\\Python37\\DLLs',
 'C:\\Users\\chris\\AppData\\Local\\Programs\\Python\\Python37\\lib',
 'C:\\Users\\chris\\AppData\\Local\\Programs\\Python\\Python37',
 'C:\\Users\\chris\\AppData\\Roaming\\Python\\Python37\\site-packages',
 'C:\\Users\\chris\\AppData\\Local\\Programs\\Python\\Python37\\lib\\site-packages
'
]
```

`__all__` 的使用

使用 `from <模块名> import *` 导入一个模块里所有的内容时，本质上是去查找这个模块的 `__all__` 属性，将 `__all__` 属性里声明的所有内容导入。如果这个模块里没有设置 `__all__` 属性，此时才会导入这个模块里的所有内容。

模块里的私有成员

模块里以一个下划线 `_` 开始的变量和函数，是模块里的私有成员，当模块被导入时，以 `_` 开头的变量默认不会被导入。但是它不具有强制性，如果一个代码强行使用以 `_` 开头的变量，有时也可以。但是强烈不建议这样使用，因为有可能会出问题。

总结

test1.py:模块里没有 `__all__` 属性

```
a = 'hello'
def fn():
    print('我是test1模块里的fn函数')
```

test2.py:模块里有 `__all__` 属性

```
x = '你好'
y = 'good'
def foo():
    print('我是test2模块里的foo函数')
__all__ = ('x','foo')
```

test3.py:模块里有以 `_` 开头的属性

```
m = '早上好'
_n = '下午好'
def _bar():
    print('我是test3里的bar函数')
```

demo.py

```
from test1 import *
from test2 import *
from test3 import *

print(a)
fn()

print(x)
# print(y) 会报错，test2的__all__里没有变量 y
foo()


print(m)
# print(_n)  会报错，导入test3时， _n 不会被导入

import test3
print(test3._n)  # 也可以强行使用，但是强烈不建议
```

`__name__` 的使用

在实际开中，当一个开发人员编写完一个模块后，为了让模块能够在项目中达到想要的效果，这个开发人员会自行在py文件中添加一些测试信息，例如：

test1.py

```
def add(a,b):
    return a+b

# 这段代码应该只有直接运行这个文件进行测试时才要执行
# 如果别的代码导入本模块，这段代码不应该被执行
ret = add(12,22)
print('测试的结果是',ret)
```

demo.py

```
import test1.py   # 只要导入了tets1.py,就会立刻执行 test1.py 代码，打印测试内容
```

为了解决这个问题，python在执行一个文件时有个变量 `__name__` .在Python中，当直接运行一个py文件时，这个py文件里的 `__name__` 值是 `__main__` ,据此可以判断一个一个py文件是被直接执行还是以模块的形式被导入。

```
def add(a,b):
    return a+b

if __name__ == '__main__':  # 只有直接执行这个py文件时,__name__的值才是 __main__
    # 以下代码只有直接运行这个文件才会执行，如果是文件被别的代码导入，下面的代码不会执行
    ret = add(12,22)
    print('测试的结果是',ret)
```

注意事项

在自定义模块时，需要注意一点，自定义模块名不要和系统的模块名重名，否则会出现问题!