

# 异常，模块和包

## 1 什么是模块和包？

- 模块，就是指py文件，我们可以将一些功能按照某个中纬度划分

自定义模块

内置模块

第三方模块

- 包，就是指文件夹，里面包含多个py文件

## 2 自定义模块和包以及使用

### 2.1 自定义模块和包

- 1 包：里面有很多模块 模块=.py文件
- 2 模块：py文件，里面有很多函数
- 3 函数：在py文件里面的函数

#### 快速入门

步骤如下：

- ① 新建包`my\_package`
- ② 新建包内模块：`my\_module1` 和 `my\_module2`
- ③ 模块内代码如下



#### Pycharm中的基本步骤：

[New] → [Python Package] → 输入包名 → [OK] → 新建功能模块(有联系的模块)

注意：新建包后，包内部会自动创建`\_\_init\_\_.py`文件，这个文件控制着包的导入行为

高级软件人才基

- 导入模块，调用模块中的函数

```
1 import time
2 # 1.使用 import 模块名 直接导入一个模块
3 from random import randint
4 # 2.from 模块名 import 函数名，导入模块中的函数
5 # （将randint方法导入random模块）
6 from math import *
7 # 3. from 模块名 import * 导入这个模块的“所有”方法和变量
8 # （将math模块里的所有方法导入到math模块中）
9 import datetime as dt
10 # 4.导入一个模块并给这个模块起一个别名
11 # （导入datetime模块，同时给这个模块起个别名 dt）
12 from copy import deepcopy as dp
13 # 5.from 模块名 import 函数名 as 别名
```

```

14 # （给copy模块导入deepcopy方法，同时给deepcopy方法起一个别名 dp）
15
16 # 导入这个模块以后，就可以使用这个模块里的方法和变量
17 print(time.time())

```

- 导入包，调用中的函数

```

1 可以将多个具有相似或者关联的多个模块放到一个文件夹里，便于统一管理
2 这个文件夹，我们称之为包
3 python包里，会有一个 __init__.py 文件
4 包中的模块导入： import 包名.模块名
5                     from 包名.模块名 import 函数名 #找到同一级别进行导入
6 包中函数的调用： 包名.模块名.函数名()

```

## 2.2 关于模块和包的导入

上述案例中，把包和模块都放在了：

- 运行文件所在的同级目录【导入成功】
- 模块和包在python的安装目录。【导入成功】

如果想把某个模块放在 F:\code\hello.py，导入到我的项目中来，默认是无法导入的，这是因为python内部看待import xxx自动回去某一些目录中寻找。这些目录可以通过 import sys print(sys.path)看到

```

1 import sys
2 print(sys.path) #打印出python寻找目录

```

如果想让python找到放在F盘中的模块，运行以下代码

```

1 import sys
2 #自定义的文件所在目录添加到sys.path
3 sys.path.append(r'F:\code')
4 #然后再调用导入hello模块
5 import hello #导入成功
6 print(sys.path)

```

问题：sys.path获取到的是个什么

```

1 是个列表，列表是有序的
2 ['D:\python学习代码\09-模块\自定义模块', 'D:\python学习代码\09-模块',
3 'C:\Users\战神\AppData\Local\Programs\Python\Python37\python37.zip',
4 'C:\Users\战神\AppData\Local\Programs\Python\Python37\DLLs',
5 'C:\Users\战神\AppData\Local\Programs\Python\Python37\lib',
6 'C:\Users\战神\AppData\Local\Programs\Python\Python37',
7 'C:\Users\战神\AppData\Local\Programs\Python\Python37\lib\site-packages']
7 会优先查早前面的模块，因此模块命名不要和内置模块命名相同

```

注意：千万不能让自己的模块名称和python内置模块的名称相同

## 2.3 关于模块和包的导入方式

- import 导入
  - 导入一个py文件: import 模块名 调用: 模块名.函数名
  - 导入包内的一个Py文件: import 包名.模块名 调用: 包名.模块名.函数名
  - 导入级别: 只能导入到py文件级别
- from 导入
  - 导入函数: from 模块名 import 函数名
  - 导入模块: from 包名 import 模块名
  - 导入所有的成员: from 包名.模块名 import \* #模块中的调用直接用
  - 导入级别: py文件中的函数级别
- import 和 from 用法不同: 多个层级的用from, 单个层级或同级目录用import

## 3 python的内置模块和使用

Python内部提供好的功能。

### 3.1 hashlib模块

是一个对数据进行加密的模块

```
1 import hashlib
2
3 data = "admin"
4 obj = hashlib.md5()
5 # hashlib 模块里主要支持两个算法 md5 和 sha 加密
6 # 加密方式: 单向加密 md5/sha 对称加密: 非对称加密: rsa
7 obj.update(data.encode('utf-8'))
8 # 需要将要加密的内容转换成二进制
9 # 使用字符串的encode方法, 可以将字符串按照指定的编码格式转换成二进制
10 # 使用decode方法, 可以将一个二进制数据按照指定的编码格式转换成字符串
11 res = obj.hexdigest()
12 print(res)
```

在以后咱们开发项目时, 密码不要用明文存储

```
1 xulei, 4e91c56c1c224b00d0f66cdad32e4f6a
```

### 3.2 密文匹配 (不可反解)

MD5加密, 不可反解。

```
1 amin --> 21232f297a57a5a743894a0e4a801fc3
```

案例

```

1 user_dict = {"xulei": "21232f297a57a5a743894a0e4a801fc3"}
2
3 user = input('请输入用户名: ') #xulei
4 pwd = input("请输入密码: ") #admin
5
6 db_pwd = user_dict.get(user) #21232f297a57a5a743894a0e4a801fc3
7 #将pwd进行加密得到密文，根据密文匹配校验
8

```

## 用户注册案例

```

1 import hashlib
2
3 SALT = "asjaohgnbdfjioan" #加盐，防止密码被破解
4 DB_FILE_PATH = 'db.txt' #全局变量，一定要大写
5 def md5(data_string):
6     obj = hashlib.md5(salt.encode('utf-8'))
7     obj.update(data_string.encode('utf-8'))
8     res = obj.hexdigest()
9     return res
10
11 def login():
12     """
13     登录
14     :return:
15     """
16     print("用户登录")
17     user = input("请输入用户名:")
18     pwd = input("请输入密码:")
19     encrypt_pwd = md5(pwd)
20     #逐行读取文件中的内容，来进行比较
21     is_sucess = False
22     with open(DB_FILE_PATH, mode='r', encoding='utf-8') as file_object:
23         for line in file_object:
24             data_list = line.strip().split(',')
25             if data_list[0] == user and data_list[1] == encrypt_pwd:
26                 if is_sucess == True:
27                     print("登录成功")
28                 else:
29                     print("登录失败")
30
31 def register():
32     """
33     注册
34     :return:
35     """
36     print("用户注册")
37     user = input("请输入用户名:")
38     pwd = input("请输入密码:")
39     encrypt_pwd = md5(pwd)
40     line = "{},{ }\n".format(user, encrypt_pwd)
41     # 将用户名和加密后的密码写入到db.txt文件中
42     with open(DB_FILE_PATH, mode='a', encoding='utf-8') as file_object:
43         file_object.write(line)
44

```

```

44 def run():
45     func_dict = {
46         "1":register,
47         "2":login
48     }
49     print("1.注册: 2.登录")
50     chioce = input("选择序号: ")
51     func = func_dict.get(chioce)
52     if not func:
53         print("序号选择错误")
54         return
55     func()
56
57 run()

```

### 3.3 random模块

帮助我们生成随机数

```

1 import random
2
3 # randint(a,b) 用来生成[a,b]的随机整数, 等价于 randrange(a,b+1)
4 print(random.randint(2, 9))

```

生成验证码

```

1 import random
2 char_list = []
3 for i in range(6):
4     num = random.randint(65,90)
5     char = chr(num)
6     char_list.append(char)
7 res = "".join(char_list)
8 print(res)

```

random中的常用函数功能

```

1 random.choice(['zhangsan', 'lisi', 'jack', 'jerry'])
2 # choice 用来在可迭代对象里随机抽取一个数据
3 random.sample(['zhangsan', 'lisi', 'jack', 'jerry'], 2)
4 # sample 用来在可迭代对象里随机抽取n个数据
5 random.randrange(2, 9)
6 # randrange(a,b) 用来生成[a,b)的随机整数
7 print(random.randint(2, 9))
8 # randint(a,b) 用来生成[a,b]的随机整数, 等价于 randrange(a,b+1)
9 # random() 用来生成[0,1)的随机浮点数
10 print(random.random())

```

案例: 年会抽奖案例

```

1 import random
2 #1. 创建300名员工
3 """
4 user_list = []

```

```

5   for i in range(1,301):
6       item = "工号-{}".format(i)
7       user_list.append(item)
8
9
10  """
11
12  user_list = ['工号-{}'.format(i) for i in range(1,301)] #推导式
13
14  #2. 奖项信息
15  data_list = [('三等奖',5),('二等奖',3),('一等奖',2),('特等奖',1)]
16  #3. 抽奖
17  for item in data_list:
18      text = item[0]
19      count = item[1]
20      #抽取count个员工，恭喜他们获奖
21      luck_user_list = random.sample(user_list,count)
22      for name in luck_user_list:
23          user_list.remove(name) #这个将获奖的人删除，防止再次获奖
24      user_string = ",".join(luck_user_list)
25      message = '荣获{}的名单:{}'.format(text,user_string)
26      print(message)

```

### 3.4 json模块 (重点)

本质上：JSON是一种数据格式，字符串形式存在。

用处：不同编程语言之间实现数据的传输。（跨语言之间的数据传输）

- 序列化与反序列化
  - 将编程语言转换为JSON数据类型叫序列化
  - 序列化：json.dumps(序列化的数据)
  - 将JSON数据类型转换为编程语言类型叫反序列化
  - 反序列化：json.loads(json数据) 反序列化为python数据

JSON格式

- 外部整体大的字符串
- JSON字符串的内部如果有字符串的话，一定需要用双引号
- json字符串中不会存在python中元组那样的格式

```

1   info = {'k1':123,'k2':(11,22,33)}
2
3   json格式
4   '{"k1":123,"k2":[11,22,33]}'

```

以下哪些数据是JSON格式的字符串：

```

1   v1 = '{"k1":123,"k2":456}' #是JSON格式
2   v2 = '{'k1':123,'k2':456}' #不是JSON格式，因为JSON格式中的字符串都是用双引号
3   v3 = '{"k1":123,"k2":456,"k3":[11,22,33]}' #是JSON格式字符串
4   v4 = '{"k1":123,"k2":456,"k3":(11,22,33)}' #不是JSON格式，JSON格式中没有元组

```

```

1 import json
2
3 info = {'k1':123,'k2':(11,22,33,44)}
4 res = json.dumps(info) #将python的数据类型，转换为json格式的字符串
5 print(res) #{"k1": 123, "k2": [11, 22, 33, 44]}

```

将json格式字符串转换为python数据类型

```

1 import json
2
3 data_string = '{"k1": 123, "k2": [11, 22, 33, 44]}'
4
5 res = json.loads(data_string)
6 print(res) #{'k1': 123, 'k2': [11, 22, 33, 44]}

```

### 3.5 JSON关于中文

```

1 import json
2
3 info = {'name':'许磊','age':27}
4 #v1 = json.dumps(info) #输出{"name": "\u8bb8\u78ca", "age": 27} unit编码
5 v1 = json.dumps(info,ensure_ascii=False) #{"name": "许磊", "age": 27}
6 print(v1)

```

### 3.6 JSON序列化

在python中默认只能通过JSON模块序列化基本的数据类型。

| python     | JSON   |
|------------|--------|
| dict       | object |
| list,tuple | array  |
| str        | string |
| int,float  | number |
| True       | true   |
| False      | false  |
| None       | null   |

案例1：基于requests模块像豆瓣发送请求获取它的热门电影

```

1 import json
2 import requests
3
4 res = requests.get(
5     url="https://movie.douban.com/j/search_subjects?
6     type=movie&tag="
7 )
8 data_dict = json.loads(res.txt)
9 for item in data_dict['subjects']:
10     print(item['title'],item['url'])

```

案例2: 写python一个网站, 给java数据提供支持

### 3.7 time模块

```

1 import time
2 #1.获取当前时间的时间戳（自1970-01-01 00:00:00 UTC 到现在时间的秒数）
3 v1 = print(time.time())
4 print(v1) #1697358250.0471613
5 print(time.strftime("%Y-%m-%d %H:%M:%S")) # 按照指定格式输出时间, 2023-10-15
16:24:10

```

案例: 获取程序执行的时间

```

1 import time
2 start_time = time.time()
3
4 end_time = time.time()
5 interval = end_time - start_time
6 print(interval)

```

延时

```

1 import time
2
3 while True:
4     print(1)
5     time.sleep(1) #延时1s

```

### 3.8 datetime 模块

- time, 时间戳
- datetime 获取当前时间

```

1 import datetime
2
3 v1 = datetime.datetime.now()
4 print(type(v1))
5 print(v1) #datetime类型 2023-10-15 16:35:33.384247
6
7 v2 = v1.strftime("%Y-%m-%d %H:%M:%S") #将datetime类型数据转换成字符串数据
8 print(type(v2))
9 print(v2) #2023-10-15 16:35:33

```



获取当前字符串格式时间

```
1 now_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
2 print(now_time) #2023-10-15 16:35:33
```

案例：

1. 用户注册

## 4 第三方模块以及使用

## 5. Python的异常操作

### 5.1 什么是异常

当检测一个错误是，Python解释器就无法继续执行了，反而会出现一些错误提示，这也就是所谓的“异常”，也就是BUG

### 5.2 知道为什么要捕获异常

当我们的程序遇到了BUG，那么接下来有两种情况：

- ① 整个程序因为一个BUG停止运行
- ② 对BUG进行提醒，整个程序继续运行

显然在之前的学习中，我们所有的程序遇到BUG就会出现①的这种情况，也就是整个程序直接奔溃。

但是在真实工作中，我们肯定不能因为一个小的BUG就让整个程序全部奔溃，也就是我们希望的是达到② 的这种情况  
那这里我们就需要使用到**捕获异常**

捕获异常的作用在于：提前假设某处会出现异常，做好提前准备，当真的出现异常的时候，可以有后续手段。

### 5.3 掌握捕获异常的语法格式

#### 1. 捕获常规异常

基本语法

```
1 try:
2     可能发生错误的代码
3 except:
4     如果出现异常执行的代码
```

#### 2. 捕获指定异常

基本语法

```

1 try:
2     print(name)
3 except NameError as e: #把异常信息设置了一个别名
4     print('name变量名称未定义错误')

```

注意事项:

- 如果尝试执行的代码的异常类型和要捕获的异常类型不一致，则无法捕获异常
- 一般try下方只放一行尝试执行的代码

### 3.捕获多个异常

当捕获多个异常时，可以把要捕获的异常类型的名字，放到except后，并用元组的方式进行书写

```

1 try:
2     print(1/0)
3 except (NameError, ZeroDivisionError):
4     print('ZeroDivisionError错误....')

```

### 4.捕获所有异常

```

1 try:
2     f = open('D:/123.txt', 'r')
3 except Exception as e:
4     print('出现异常了')

```

## 5.4异常else

else表示的是如果没有异常要执行的代码

```

1 try:
2     print(1)
3 except Exception as e:
4     print(e)
5 else:
6     print('我是else，是没有异常时执行的代码')

```

## 5.5异常的finally

finally表示的是无论是否异常都要执行的代码：例如关闭文件

```

1 try:
2     f = open('text.txt', 'r')
3 except Exception as e:
4     f = open('text.txt', 'w')
5 else:
6     print('我是else，是没有异常时执行的代码')
7 finally:
8     f.close()

```

## 5.6异常的传递