21. 休养生息-常用模块-01

本节主要内容:

1. 模块的简单认识
2. collections模块
3. time时间模块
4. random模块
5. os模块
6. sys模块

一. 模块的简单认识

什么是模块. 模块就是我们把装有特定功能的代码进行归类的结果. 从代码编写的单位来看我们的程序, 从小到大的顺序: 一条代码 < 语句块 < 代码块(函数, 类) < 模块. 我们目前写的所有的py文件都是模块.

引入模块的方式:

1. import 模块
2. from xxx import 模块

关于这两种写法. 我们后面还要继续介绍. 在之前的学习中, 我们已经用过了一些基本的模块了. 比如, random, os, sys, collections等等. 那我们目前用到的所有模块都是python内置的模块.不需要额外安装. 在后面学习高级框架的内容的时候. 可能需要我们自行安装一些第三方提供的模块.

二. collections模块

collections模块主要封装了一些关于集合类的相关操作. 比如, 我们学过的Iterable, Iterator等等. 除了这些以外, collections还提供了一些除了基本数据类型以外的数据集合类型. Counter, deque, OrderDict, defaultdict以及namedtuple

1. Counter

counter是一个计数器. 主要用来计数

计算一个字符串中每个字符出现的次数:

```
low:

s = "alex like pig"

dic = {}
for c in s:
    dic[c] = dic.get(c, 0) + 1
print(dic)

nb:
s = "alex like pig"
print(Counter(s)) # 获取到的结果可以像字典一样进行使用  [key]
```

2. deque 双向队列.

(重点)说双向队列之前我们需要了解两种数据结构. 1. 栈, 2. 队列

1. 栈: FILO. 先进后出 -> 砌墙的砖头, 老师傅做馒头
2. 队列: FIFO. 先进先出 -> 买火车票排队, 所有排队的场景

由于python没有给出Stack模块. 所以我们自己手动写一个粗略版本(注意, 此版本有严重的并发问题)

```
class StackEmptyError(Exception):
    pass
class StackFullError(Exception):
    pass
class Stack:
    def __init__(self, size):
        self.index = 0
        self.size = size
        self.lst = []

    def pop(self):
        if self.index > 0:
            ret = self.lst[self.index]
            return ret
        else:
            raise StackEmptyError("stack has already empty")

    def push(self, el):
        if self.index > self.size:
            raise StackFullError("stack is full")
        else:
            self.lst[self.index] = el
            self.index = self.index + 1

    def clear(self):
        self.lst.clear()
        self.index = 0

    def __sizeof__(self):
        return len(self.lst)

    def max(self):
        return self.size

    def now(self):
        return self.index
```

队列: python提供了queue模块. 使用起来非常方便

```
import queue
q = queue.Queue()
q.put("李嘉诚")
q.put("张开")
q.put("张毅")
print(q)
print(q.get())
print(q.get())
print(q.get())
```

注意. 如果队列里没有元素了. 再也就拿不出来元素了. 此时程序会阻塞.

接下来,. 我们来看一下deque, 注意, 此队列是collections中的.

```
from collections import deque

q = deque()
q.append("张开")  # 右侧添加
q.append("包贝尔")
q.appendleft("赵又廷") # 左侧添加
q.appendleft("还我高圆圆")
print(q)

print(q.pop())  # 右侧删除
print(q.popleft())  # 左侧删除
```

3. namedtuple 命名元组

命名元组, 顾名思义. 给元组内的元素进行命名. 比如. 我们说(x, y) 这是一个元组. 同时. 我们还可以认为这是一个点坐标. 这时, 我们就可以使用namedtuple对元素进行命名

```
from collections import namedtuple

# 自己定义了一个元组, 如果灵性够好, 这其实就是创建了一个类
nt = namedtuple("point", ["x", "y"])
p = nt(1, 2)
print(p)

print(p.x)
print(p.y)
```

4. orderdict和defaultdict

orderdict 顾名思义. 字典的key默认是无序的. 而OrderedDict是有序的

```
dic = {'a':'娃哈哈', 'b':'薯条', 'c':'胡辣汤'}
print(dic)

from collections import OrderedDict
od = OrderedDict({'a':'娃哈哈', 'b':'薯条', 'c':'胡辣汤'})
print(od)
```

defaultdict: 可以给字典设置默认值. 当key不存在时. 直接获取默认值:

```
from collections import defaultdict

dd = defaultdict(list)  # 默认值list
print(dd['娃哈哈'])    # [] 当key不存在的时候. 会自动执行构造方法中传递的内容.
```

三. time 时间模块(重点)

时间模块是我们要熟记的. 到后面写程序的时候经常能用到. 比如, 如何计算时间差. 如何按照客户的要求展示时间. 等等.

```
import time
print(time.time())  # 1538927647.483177  系统时间
```

此时, 我们已经获取到了系统时间, 但是这个时间....看不懂. 怎么办呢. 需要对时间进行格式化. 那这样就引出了另一种时间的格式. 在python中时间分成三种表现形式:

1. 时间戳(timestamp). 时间戳使用的是从1970年01月01日 00点00分00秒到现在一共经过了多少秒... 使用float来表示
2. 格式化时间(strftime). 这个时间可以根据我们的需要对时间进行任意的格式化.
3. 结构化时间(struct_time). 这个时间主要可以把时间进行分类划分. 比如. 1970年01月01日 00点00分00秒 这个时间可以被细分为年, 月, 日.....一大堆东西.

时间戳我们已经见过了就是time.time(). 一般, 我们不会把这样的时间显示给客户. 那就需要对时间进行格式化操作.

```
s = time.strftime("%Y-%m-%d %H:%M:%S")    # 必须记住
print(s)
```

日期格式化的标准:

```
%y 两位数的年份表示（00-99）
%Y 四位数的年份表示（000-9999）
%m 月份（01-12）
%d 月内中的一天（0-31）
%H 24小时制小时数（0-23）
%I 12小时制小时数（01-12）
%M 分钟数（00=59）
%S 秒（00-59）
%a 本地简化星期名称
%A 本地完整星期名称
```

```
%b 本地简化的月份名称
%B 本地完整的月份名称
%c 本地相应的日期表示和时间表示
%j 年内的一天（001-366）
%p 本地A.M.或P.M.的等价符
%U 一年中的星期数（00-53）星期天为星期的开始
%w 星期（0-6），星期天为星期的开始
%W 一年中的星期数（00-53）星期一为星期的开始
%x 本地相应的日期表示
%X 本地相应的时间表示
%Z 当前时区的名称
%% %号本身
```

看一下结构化时间:

```
print(time.localtime())
结果:
time.struct_time(tm_year=2017, tm_mon=05, tm_mday=8, tm_hour=10, tm_min=24,
tm_sec=42, tm_wday=0, tm_yday=126, tm_isdst=0)
```

好了. 先在看到的都是当前系统时间, 那如果碰到时间转换呢? 比如. 我们的数据库中存储了这样一个时间: 1888888888. 如何显示成xxxx年xx月xx日. 那时间的转化必须要记住: 所有的转化都要通过结构化时间来转化.

```
t = time.localtime(1888888888)  # 结构化时间
s = time.strftime("%Y-%m-%d %H:%M:%S", t) # 格式化这个时间
print(s)
```

那如果说, 我让用户输入一个时间, 怎么把它转化成我们数据库存储的时间戳呢? 还是要用到结构化时间

```
s = "2020-10-01 12:18:12"
t = time.strptime(s, "%Y-%m-%d %H:%M:%S")   # 转化成结构时间
print(time.mktime(t))   # 转换成时间戳
```

以上两段代码. 必须记住.

计算时间差:

```
import time
true_time=time.mktime(time.strptime('2017-09-11 08:30:00','%Y-%m-%d
%H:%M:%S'))
time_now=time.mktime(time.strptime('2017-09-12 11:00:00','%Y-%m-%d
%H:%M:%S'))
dif_time=time_now-true_time
struct_time=time.localtime(dif_time)
print(struct_time)
print('过去了%d年%d月%d天%d小时%d分钟%d秒'%(struct_time.tm_year-
1970,struct_time.tm_mon-1,
                                        struct_time.tm_mday-
```

```
1,struct_time.tm_hour,

struct_time.tm_min,struct_time.tm_sec))
```

四. random模块

所有关于随机相关的内容都在random模块中.

```
import random

print(random.random())  # 0-1小数
print(random.uniform(3, 10))  # 3-10小数

print(random.randint(1, 10))  # 1-10整数 [1, 10]
print(random.randrange(1, 10, 2))  # 1-10奇数 [1,10)

print(random.choice([1, '周杰伦', ["盖伦", "胡辣汤"]]))  # 1或者23或者[4,5])
print(random.sample([1, '23', [4, 5]], 2))  # 列表元素任意2个组合

lst = [1, 2, 3, 4, 5, 6, 7, 8]
random.shuffle(lst)  # 随机打乱顺序
print(lst)
```

五. os模块

所有和操作系统相关的内容都在os模块

```
os.makedirs('dirname1/dirname2')    可生成多层递归目录
os.removedirs('dirname1')    若目录为空，则删除，并递归到上一级目录，如若也为空，则删除，依此类推
os.mkdir('dirname')    生成单级目录；相当于shell中mkdir dirname
os.rmdir('dirname')    删除单级空目录，若目录不为空则无法删除，报错；相当于shell中rmdir dirname
os.listdir('dirname')    列出指定目录下的所有文件和子目录，包括隐藏文件，并以列表方式打印
os.remove()  删除一个文件
os.rename("oldname","newname")  重命名文件/目录
os.stat('path/filename')  获取文件/目录信息

os.system("bash command")  运行shell命令，直接显示
os.popen("bash command).read()  运行shell命令，获取执行结果
os.getcwd() 获取当前工作目录，即当前python脚本工作的目录路径
os.chdir("dirname")  改变当前脚本工作目录；相当于shell下cd

# os.path
os.path.abspath(path) 返回path规范化的绝对路径
os.path.split(path) 将path分割成目录和文件名二元组返回
os.path.dirname(path) 返回path的目录。其实就是os.path.split(path)的第一个元素
os.path.basename(path) 返回path最后的文件名。如何path以 / 或\结尾，那么就会返回空值。即os.path.split(path)的第二个元素
```

```
os.path.exists(path)  如果path存在，返回True；如果path不存在，返回False
os.path.isabs(path)  如果path是绝对路径，返回True
os.path.isfile(path)  如果path是一个存在的文件，返回True。否则返回False
os.path.isdir(path)  如果path是一个存在的目录，则返回True。否则返回False
os.path.join(path1[, path2[, ...]])  将多个路径组合后返回，第一个绝对路径之前的参数将被忽略
os.path.getatime(path)  返回path所指向的文件或者目录的最后访问时间
os.path.getmtime(path)  返回path所指向的文件或者目录的最后修改时间
os.path.getsize(path) 返回path的大小

# 特殊属性:
os.sep     输出操作系统特定的路径分隔符，win下为"\\",Linux下为"/"
os.linesep     输出当前平台使用的行终止符，win下为"\r\n",Linux下为"\n"
os.pathsep     输出用于分割文件路径的字符串 win下为;,Linux下为:
os.name     输出字符串指示当前使用平台。win->'nt'; Linux->'posix'
```

os.stat() 属性解读:

```
stat 结构:

st_mode: inode 保护模式
st_ino: inode 节点号。
st_dev: inode 驻留的设备。
st_nlink: inode 的链接数。
st_uid: 所有者的用户ID。
st_gid: 所有者的组ID。
st_size: 普通文件以字节为单位的大小；包含等待某些特殊文件的数据。
st_atime: 上次访问的时间。
st_mtime: 最后一次修改的时间。
st_ctime: 由操作系统报告的"ctime"。在某些系统上（如Unix）是最新的元数据更改的时间，在其它系统上（如Windows）是创建时间（详细信息参见平台的文档）。
```

六. sys模块

所有和python解释器相关的都在sys模块.

```
sys.argv           命令行参数List，第一个元素是程序本身路径
sys.exit(n)        退出程序，正常退出时exit(0),错误退出sys.exit(1)
sys.version        获取Python解释程序的版本信息
sys.path           返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值
sys.platform       返回操作系统平台名称
```