OS进行第三方库的操作：

OS模块介绍以及操作方法：

OS：操作系统，用于操作文件/文件夹等

Os.cpu\_count()获取cpu个数。

Os.getcwd()返回当前文件所在文件夹的绝对路径。

Os.path.abspath()返回自身所在路径的绝对路径

Os.path.basename:取得路径的最后一部分

Os.path.dirname():返回当前文件夹所在的父类文件夹

Os.path.exits():判断路径下的文件/文件夹是不是存在。

Os.path.isfile():判断目标是不是文件，返回值为Bool值

Os.path.isdir():判断目标是不是文件夹，同样返回值

Os.path.join():拼接路径

Os.mkdir():创建文件夹

Os.chidr():切换文件夹

Os.makedirs:递归创建文件夹

Os.rename():文件更名

Os.remove（）：文件删除

Os.path.pardir:返回当前文件或者文件夹的父及目录。

Os.path.join(‘将目录和文件合并成同一个路径’)

正则表达式 0 对应的第一个，1对应的第二个括号

Os.popen 直接读取命令执行结果，将相应的命令输入

python调用Shell脚本，有两种方法：os.system(cmd)或os.popen(cmd),前者返回值是脚本的退出状态码，后者的返回值是脚本执行过程中的输出内容

os.system(cmd)——>输出的是二进制数，可直接看运行结果

os.popen(cmd)——>输出的是文件，可对文件内容继续操作

* os.system(cmd)的返回值是脚本的退出状态码，只会有0(成功),1,2
* os.popen(cmd)返回脚本执行的输出内容作为返回值

例子如下：

with os.popen(command, "r") as p:

r = p.read()

或者使用 r = p.readline()

Sys.argv是一个列表，这个列表中的元素索引

0代表本身

其余每个对应的就是控制台输入的参数如下：

Sys.argv[1] 即此程序跟着的参数，假如此程序后面跟这的参数

Test.py what 输出的就是what

Sys.arg[2: ] test.py a b c d e f ================输出b c d e f

python常用的第三方库：重点其中的作用

Django：一个高级的web框架，使用orm映射模拟引擎的一切东西

其在构建app中，需要遵循独有的规定和方式

Socket:一个套接字底层库，可以用来模拟B/S之间通信的TCP和UDP

Mysqlclient:完全兼容mysqldb，是django orm的依赖工具。

标准模块：sys，random,os,time,turtle（标准库机器学习）

Requests爬虫请求，scarpy 爬虫框架，pyspider。

beautifulSoup一个HTML和xml解析库---目前的版本4.4

Urllib库，一个内置http的请求库，其中有三个模块：

Request,error,parse三个模块

Threading模块（线程模块库）：

表示当前的位置：\_\_file\_\_

Os.path.abspath(\_\_file\_\_)表示当前此文件的位置

**Re正则表达式:**

正则表达式：目的筛选数据-------清洗数据

通用的规则，在其他的语言中也是可以使用。

导入import re

Result = re.match(正则表达，匹配的字符)

Match 从头开始匹配-->search 寻找匹配

Result.group()--------------提取数据从匹配的数据中提取相应的数据。

Re.match(r””)----------搜索条件

\d匹配数字--------任意一个数字



如图判断，1-3 6-8的

所有的范围可以将全部写在括号中

\w匹配了所有字母---只要不是符号就是用

\s 空白字符



大小写是相反的，./可以匹配所有

匹配多个字符



{}中包含的是能够匹配的位数

1-3位数字

.\*匹配所有一起，输入什么都可以

Re.S 反斜杠可以显示



和.进行配合



解析文件用的第三方库：

lxml库中的xpath进行匹配，通过路径的匹配寻找节点

**导入lxml的etree模块，此模块可以自动修正文本**

**etree模块将文本进行处理，去掉符号**

**Etree.parse(名字).getroot()**

**将页面的源代码解析，**

**Html\_string=etree.HTML()**

**Pip install cssselct模块，可以和xpath进行对比：**

xlwt模块的用法：

第一步导入import xlwt

第二步创建一个工作表对象



第三步设置excel表名

Sheet=workbook.add\_sheet(名字)

第四步往表格中填充数据

第一个参数 行号 第二个参数 列号

Sheet.write(x,y,”添加的文字”)

第五步保存

Workbook.save(“名字.xls”)

**获取一个xls工作表对象，并且可以对其进行处理**

**Configparser** 读取配置文件的库

Raw读取方式，使用Raw进行读取；

**其中使用的值为：**

## 一、读取

 　　read(filename)直接读取ini文件内容

sections() 得到所有的section，并以列表的形式返回-----某一段的值？

        options(section) 得到该section的所有option

items(section) 得到该section的所有键值对

get(section,option) 得到section中option的值，返回为string类型

getint(section,option) 得到section中option的值，返回为int类型

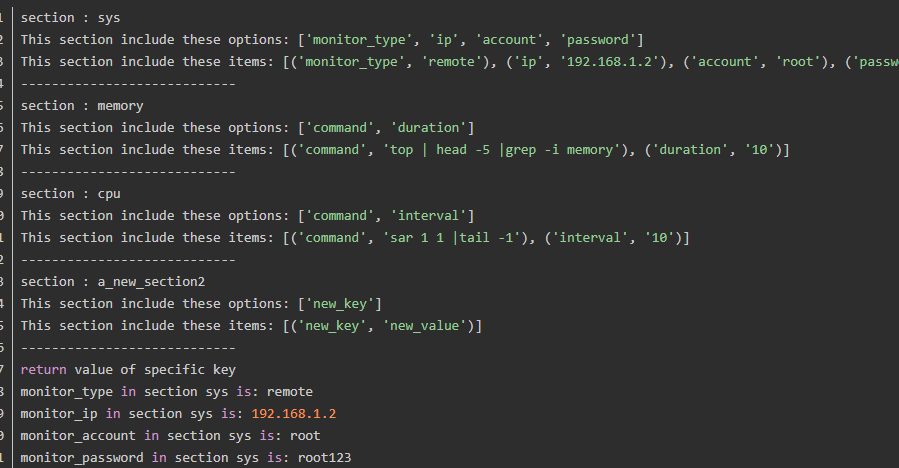
## ****二.写入****

        add\_section(section) 添加一个新的section

set(section, option, value) 对section中的option进行设置

        conf.write()写

Section的解释如下图:



**Argparse 库：这是一个将参数和函数与控制台进行交互的库**

**使用方式：**

1. **创建ArgumentParser() 对象**
2. **调用add\_argument()方法添加参数**
3. **使用parse\_args（）解析添加的参数**

**列子如下：需要配合**

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument('integer', type=int, help='display an integer')

args = parser.parse\_args()

Print(args.integer)

**相应的参数如下：这个是add\_argument（）使用**

* name or flags - 选项字符串的名字或者列表，例如 foo 或者 -f, --foo。
* action - 命令行遇到参数时的动作，默认值是 store。

**action的参数：**

* + store\_const，表示赋值为const；
  + append，将遇到的值存储成列表，也就是如果参数重复则会保存多个值;
  + append\_const，将参数规范中定义的一个值保存到一个列表；
  + count，存储遇到的次数；此外，也可以继承 argparse.Action 自定义参数解析；

**“--”设置可选择的参数，设定的参数可以通过控制台进行调用，解决位置参数顺序引起的问题**

**“” 直接输入的话，那么展示的就是可选参数**

**Subprocess：生成一个新的进程并且加入到管道**

**目的：替换os.system////os.spawn**

**常用的函数：以下均使用su代替subprocess**

**Su.run() 执行命令，返回一个包含执行结果的类实例**

**Su.call() 执行命令，只返回一个执行状态**

**Su.check\_call() 执行命令，成功返回状态，失败抛出异常**

**Su.check\_output() 如果执行的状态码为0则显示成功，并且有返回值，否则抛出异常**

**Su.getoutput(cmd) 接触字符串形式的命令，执行命令，并且返回结果，其功能类似于:**

**os.popen(cmd).read() commands.getoutput(cmd)**

**Su.getstatusoutput(cmd) 执行cmd命令，返回一个元祖(),其功能类似commands.getstatusoutput()**

**subprocess.run(args, \*, stdin=None, input=None, stdout=None, stderr=None, shell=False, timeout=None, check=False, universal\_newlines=False)**

**subprocess.call(args, \*, stdin=None, stdout=None, stderr=None, shell=False, timeout=None)**

**subprocess.check\_call(args, \*, stdin=None, stdout=None, stderr=None, shell=False, timeout=None)**

**subprocess.check\_output(args, \*, stdin=None, stderr=None, shell=False, universal\_newlines=False, timeout=None)**

**subprocess.getstatusoutput(cmd)**

**subprocess.getoutput(cmd)**

**使用subprocess.Popen介绍：**

**该类用于在一个新的进程中执行一个子程序**

**class subprocess.Popen(args, bufsize=-1, executable=None, stdin=None, stdout=None, stderr=None,**

**preexec\_fn=None, close\_fds=True, shell=False, cwd=None, env=None, universal\_newlines=False,**

**startup\_info=None, creationflags=0, restore\_signals=True, start\_new\_session=False, pass\_fds=())**

* **args：** 要执行的shell命令，可以是字符串，也可以是命令各个参数组成的序列。当该参数的值是一个字符串时，该命令的解释过程是与平台相关的，因此通常建议将args参数作为一个序列传递。
* **bufsize：** 指定缓存策略，0表示不缓冲，1表示行缓冲，其他大于1的数字表示缓冲区大小，负数 表示使用系统默认缓冲策略。
* **stdin, stdout, stderr：** 分别表示程序标准输入、输出、错误句柄。
* **preexec\_fn：** 用于指定一个将在子进程运行之前被调用的可执行对象，只在Unix平台下有效。
* **close\_fds：** 如果该参数的值为True，则除了0,1和2之外的所有文件描述符都将会在子进程执行之前被关闭。
* **shell：** 该参数用于标识是否使用shell作为要执行的程序，如果shell值为True，则建议将args参数作为一个字符串传递而不要作为一个序列传递。
* **cwd：** 如果该参数值不是None，则该函数将会在执行这个子进程之前改变当前工作目录。
* **env：** 用于指定子进程的环境变量，如果env=None，那么子进程的环境变量将从父进程中继承。如果env!=None，它的值必须是一个映射对象。
* **universal\_newlines：** 如果该参数值为True，则该文件对象的stdin，stdout和stderr将会作为文本流被打开，否则他们将会被作为二进制流被打开。
* **startupinfo和creationflags：** 这两个参数只在Windows下有效，它们将被传递给底层的CreateProcess()函数，用于设置子进程的一些属性，如主窗口的外观，进程优先级等。

**因为其是一个类方法，可以调用的函数：**

**Popen.poll 检查子进程是否已经执行结束**

**Popen.wait(timeout=none) 等待子进程结束，返回状态码**

**Popen.communicate 该方法用来和进程交互，发送数据**

**Popen.kill 杀死进程**

**Collections 库中orderedDict 使用，进行字典元素的排序，按照输入的顺序，对字典中的值进行排序：**

**D1 = collections.orderedDict()**

**对实例化的字典对象D1进行添加或者取出操作**