本章主要讲解数据存储中非数据库版的部分，大体分为两块内容，一块是将取出的 HTML 文本内容进行存储，一块是多媒体文件的存储。本章同时也讲解了 Email 的发送，这在 Python 爬虫出现异常时有很好的作用。

# 5.1　HTML 正文抽取

本小节讲解的是对 HTML 正文的抽取存储，主要是将 HTML 正文存储为两种格式：JSON 和 CSV。以一个盗墓笔记的小说阅读网（<http://seputu.com/>）为例，抽取出盗墓笔记的标题、章节、章节名称和链接，如图5-1所示。

首先有一点需要说明，这是一个静态网站，标题、章节、章节名称都不是由 JavaScript 动态加载的，这是下面所进行的工作的前提。

这个例子使用第4章介绍的 Beautiful Soup 和 lxml 两种方式进行解析抽取，力求将之前的知识进行灵活运用。5.1.1小节，使用 Beautiful Soup 解析，5.1.2小节使用 lxml 方式解析。

5.1.1　存储为 JSON

首先使用 Requests 访问 <http://seputu.com/>，获取 HTML 文档内容，并打印文档内容。代码如下：

**import** requests

user\_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'

headers={'User-Agent':user\_agent}

r = requests.get('http://seputu.com/',headers=headers)

**print** r.text

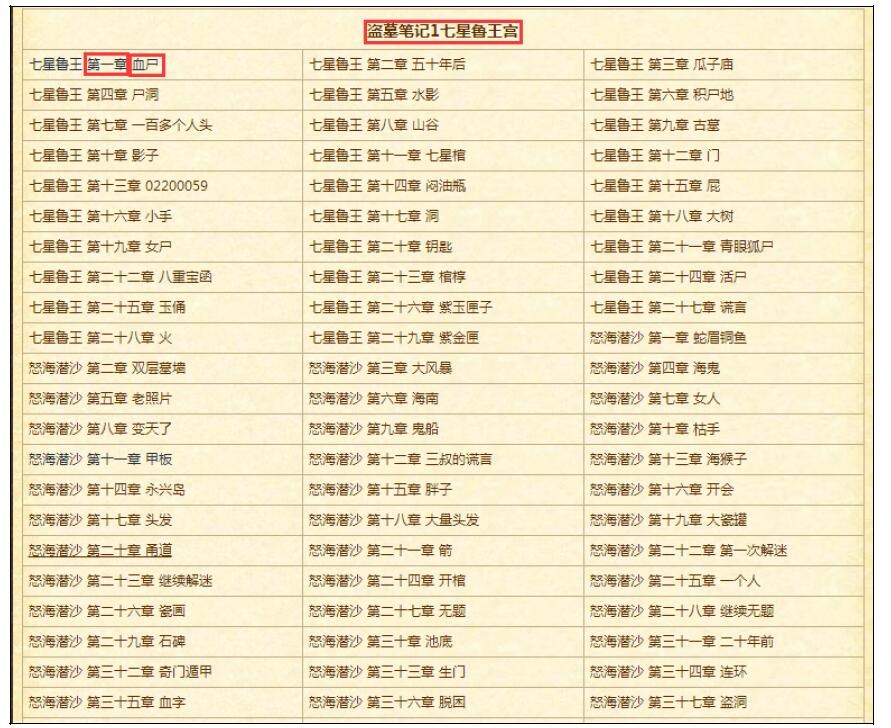
[](http://popimage/?src=)

图5-1　盗墓笔记小说网

接着分析 <http://seputu.com/> 首页的 HTML 结构，确定要抽取标记的位置，分析如下：

标题和章节都被包含在<div class=“mulu”>标记下，标题位于其中的<div class=“mulu-title”>下的<h2>中，章节位于其中的<div class=“box”>下的<a>中，如图5-2所示。

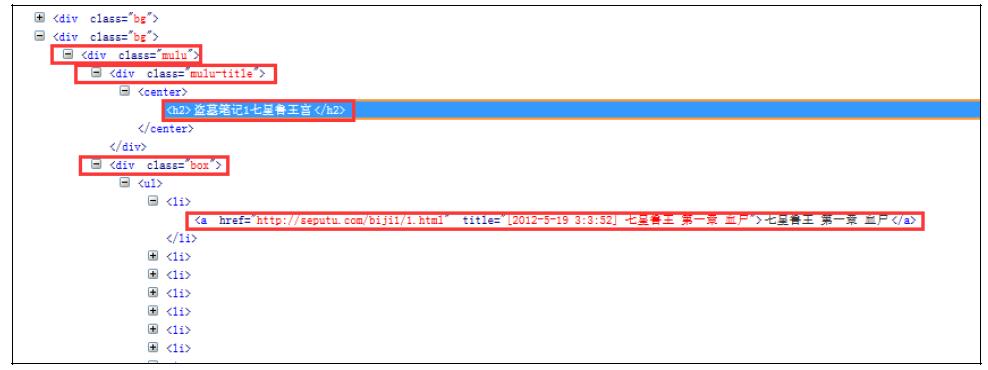
[](http://popimage/?src=)

图5-2　HTML 结构分析

分析完成就可以进行编码了，代码如下：

soup = BeautifulSoup(r.text,'html.parser',from\_encoding='utf-8')# html.parser

**for** mulu **in** soup.find\_all(class\_="mulu"):

h2 = mulu.find('h2')

**if** h2!=None:

h2\_title = h2.**string**# 获取标题

**for** a **in** mulu.find(class\_='box').find\_all('a'):# 获取所有的 a 标记中 url 和章节内容

href = a.**get**('href')

box\_title = a.**get**('title')

print href,box\_title

这时已经成功获取标题、章节，接下来将数据存储为 JSON。在第2章中，我们已经讲解了 JSON 文件的基本格式，下面讲解 Python 如何操作 JSON 文件。

Python 对 JSON 文件的操作分为编码和解码，通过 JSON 模块来实现。编码过程是把 Python 对象转换成 JSON 对象的一个过程，常用的两个函数是 dumps 和 dump 函数。两个函数的唯一区别就是 dump 把 Python 对象转换成 JSON 对象，并将 JSON 对象通过 fp 文件流写入文件中，而 dumps 则是生成了一个字符串。下面看一下 dumps 和 dump 的函数原型：

dumps(obj, skipkeys=**False**, ensure\_ascii=**True**, check\_circular=**True**,

allow\_nan=**True**, cls=**None**, indent=**None**, separators=**None**,

encoding='utf-8', default=**None**, sort\_keys=**False**, \*\*kw)

dump(obj, fp, skipkeys=**False**, ensure\_ascii=**True**, check\_circular=**True**,

allow\_nan=**True**, cls=**None**, indent=**None**, separators=**None**,

encoding='utf-8', default=**None**, sort\_keys=**False**, \*\*kw):

常用参数分析：

* Skipkeys：默认值是 False。如果 dict 的 keys 内的数据不是 python 的基本类型（str、unicode、int、long、float、bool、None），设置为 False 时，就会报 TypeError 错误。此时设置成 True，则会跳过这类 key。
* ensure\_ascii：默认值 True。如果 dict 内含有非 ASCII 的字符，则会以类似“\uXXXX”的格式显示数据，设置成 False 后，就能正常显示。
* indent：应该是一个非负的整型，如果是0，或者为空，则一行显示数据，否则会换行且按照 indent 的数量显示前面的空白，将 JSON 内容进行格式化显示。
* separators：分隔符，实际上是（item\_separator，dict\_separator）的一个元组，默认的就是（’，‘，’：‘），这表示 dictionary 内 keys 之间用“，”隔开，而 key 和 value 之间用“：”隔开。
* encoding：默认是 UTF-8。设置 JSON 数据的编码方式，在处理中文时一定要注意。
* sort\_keys：将数据根据 keys 的值进行排序。

示例如下：

**import** json

str =[{"username":"七夜","age":24},(2,3),1]

json\_str= json.dumps(str,ensure\_ascii=**False**)

**print** json\_str

**with** open('qiye.txt','w') **as** fp:

json.dump(str,fp=fp,ensure\_ascii=**False**)

输出结果：

[{"username": "七夜", "age": 24}, [2, 3], 1]

解码过程是把 json 对象转换成 python 对象的一个过程，常用的两个函数是 load 和 loads 函数，区别跟 dump 和 dumps 是一样的。函数原型如下：

loads(s, encoding=**None**, cls=**None**, object\_hook=**None**, parse\_float=**None**,

parse\_int=**None**, parse\_constant=**None**, object\_pairs\_hook=**None**, \*\*kw)

load(fp, encoding=**None**, cls=**None**, object\_hook=**None**, parse\_float=**None**,

parse\_int=**None**, parse\_constant=**None**, object\_pairs\_hook=**None**, \*\*kw)

常用参数分析：

* encoding：指定编码格式。
* parse\_float：如果指定，将把每一个 JSON 字符串按照 float 解码调用。默认情况下，这相当于 float（num\_str）。
* parse\_int：如果指定，将把每一个 JSON 字符串按照 int 解码调用。默认情况下，这相当于 int（num\_str）。

示例如下：

new\_str=json.loads(json\_str)

**print** new\_str

**with** open('qiye.txt','r') **as** fp:

**print** json.load(fp)

输出结果：

[{u'username': u'\u4e03\u591c', u'age': 24}, [2, 3], 1]

[{u'username': u'\u4e03\u591c', u'age': 24}, [2, 3], 1]

通过上面的例子可以看到，Python 的一些基本类型通过编码之后，tuple 类型就转成了 list 类型了，再将其转回为 python 对象时，list 类型也并没有转回成 tuple 类型，而且编码格式也发生了变化，变成了 Unicode 编码。具体转化时，类型变化规则如表5-1和表5-2所示。

表5-1　Python→JSON

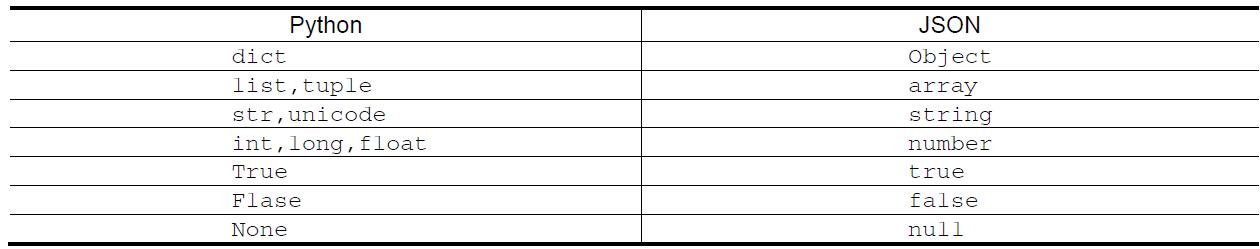
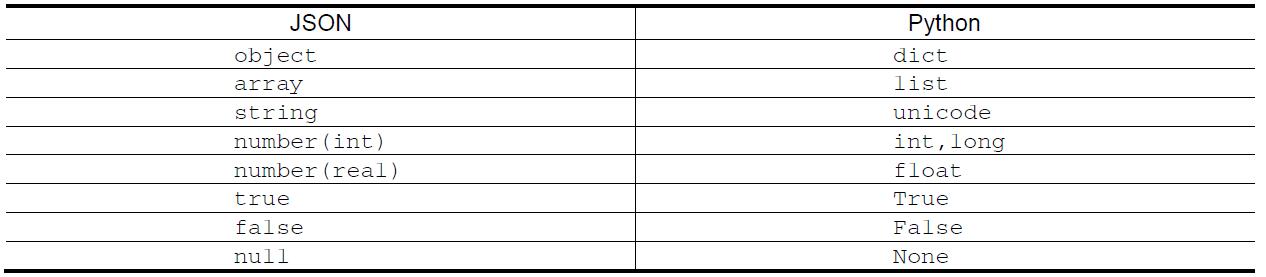
[](http://popimage/?src=)

表5-2　JSON→Python

[](http://popimage/?src=)

以上就是 Python 操作 JSON 的全部内容，接下来将提取到的标题、章节和链接进行 JSON 存储。完整代码如下：

*# coding:utf-8*

**import** json

**from** bs4 **import** BeautifulSoup

**import** requests

user\_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'

headers={'User-Agent':user\_agent}

r = requests.get('http://seputu.com/',headers=headers)

soup = BeautifulSoup(r.text,'html.parser',from\_encoding='utf-8')*# html.parser*

content=[]

**for** mulu **in** soup.find\_all(class\_="mulu"):

h2 = mulu.find('h2')

**if** h2!=**None**:

h2\_title = h2.string*# 获取标题*

list=[]

**for** a **in** mulu.find(class\_='box').find\_all('a'):*# 获取所有的 a 标记中 url 和章节内容*

href = a.get('href')

box\_title = a.get('title')

list.append({'href':href,'box\_title':box\_title})

content.append({'title':h2\_title,'content':list})

**with** open('qiye.json','wb') **as** fp:

json.dump(content,fp=fp,indent=4)

打开 qiye.json 文件，效果如图5-3所示。

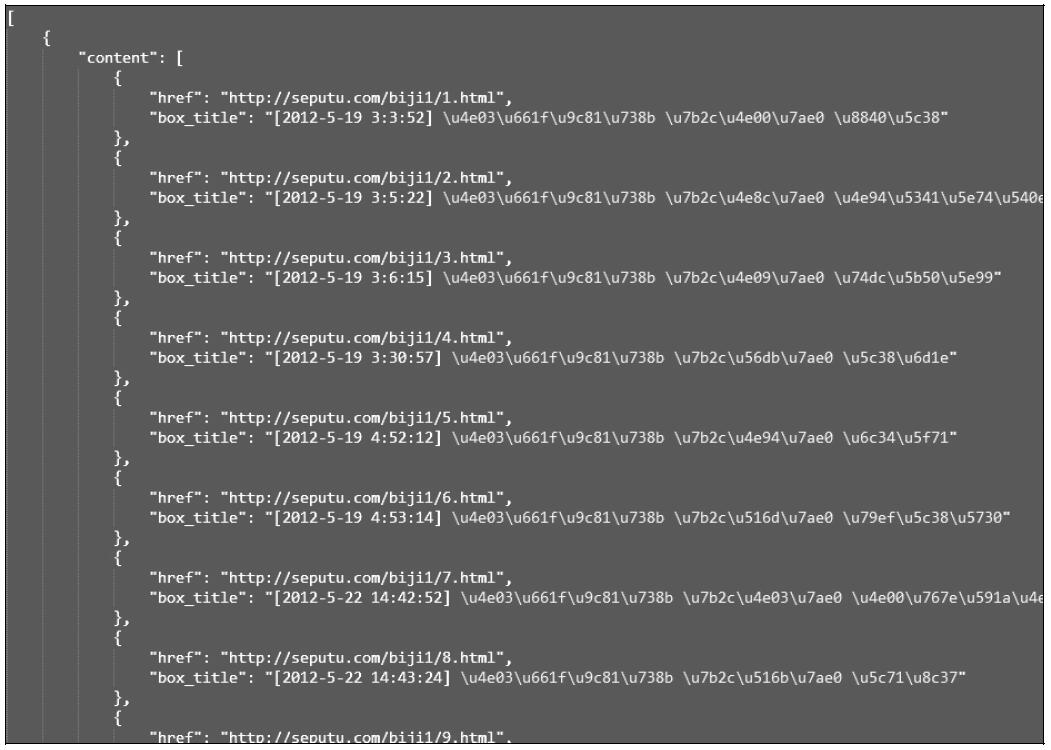
[](http://popimage/?src=)

图5-3　qiye.json

5.1.2　存储为 CSV

CSV（Comma-Separated Values，逗号分隔值，有时也称为字符分隔值，因为分隔字符也可以不是逗号），其文件以纯文本形式存储表格数据（数字和文本）。纯文本意味着该文件是一个字符序列，不含必须像二进制数字那样被解读的数据。

CSV 文件由任意数目的记录组成，记录间以某种换行符分隔；每条记录由字段组成，字段间的分隔符是其他字符或字符串，最常见的是逗号或制表符。通常，所有记录都有完全相同的字段序列。CSV 文件示例如下：

ID,UserName,Password,age,country

1001,"qiye","qiye\_pass",24,"China"

1002,"Mary","Mary\_pass",20,"USA"

1003,"Jack","Jack\_pass",20,"USA"

Python 使用 csv 库来读写 CSV 文件。要将上面 CSV 文件的示例内容写成 qiye.csv 文件，需要用到 Writer 对象，代码如下：

**import** csv

headers = ['ID','UserName','Password','Age','Country']

rows = [(1001,"qiye","qiye\_pass",24,"China"),

(1002,"Mary","Mary\_pass",20,"USA"),

(1003,"Jack","Jack\_pass",20,"USA"),

]

**with** open('qiye.csv','w') **as** f:

f\_csv = csv.writer(f)

f\_csv.writerow(headers)

f\_csv.writerows(rows)

里面的 rows 列表中的数据元组，也可以是字典数据。示例如下：

**import** csv

headers = ['ID','UserName','Password','Age','Country']

rows = [{'ID':1001,'UserName':"qiye",'Password':"qiye\_pass",'Age':24,'Country':" China"},

{'ID':1002,'UserName':"Mary",'Password':"Mary\_pass",'Age':20,'Country':"USA"},

{'ID':1003,'UserName':"Jack",'Password':"Jack\_pass",'Age':20,'Country':"USA"},

]

**with** open('qiye.csv','w') **as** f:

f\_csv = csv.DictWriter(f,headers)

f\_csv.writeheader()

f\_csv.writerows(rows)

接下来讲解 CSV 文件的读取。将之前写好的 qiye.csv 文件读取出来，需要创建 reader 对象，示例如下：

**import** csv

**with** open('qiye.csv') **as** f:

f\_csv = csv.reader(f)

headers = next(f\_csv)

**print** headers

**for** row **in** f\_csv:

**print** row

运行结果：

['ID', 'UserName', 'Password', 'Age', 'Country']

['1001', 'qiye', 'qiye\_pass', '24', 'China']

['1002', 'Mary', 'Mary\_pass', '20', 'USA']

['1003', 'Jack', 'Jack\_pass', '20', 'USA']

在上面的代码中，row 会是一个列表。因此，为了访问某个字段，你需要使用索引，如 row[0] 访问 ID，row[3] 访问 Age。由于这种索引访问通常会引起混淆，因此可以考虑使用命名元组。示例如下：

**from** collections **import** namedtuple

**import** csv

**with** open('qiye.csv') **as** f:

f\_csv = csv.reader(f)

headings = next(f\_csv)

Row = namedtuple('Row', headings)

**for** r **in** f\_csv:

row = Row(\*r)

**print** row.UserName,row.Password

**print** row

运行结果：

qiye qiye\_pass

Row(ID='1001', UserName='qiye', Password='qiye\_pass', Age='24', Country='China')

Mary Mary\_pass

Row(ID='1002', UserName='Mary', Password='Mary\_pass', Age='20', Country='USA')

Jack Jack\_pass

Row(ID='1003', UserName='Jack', Password='Jack\_pass', Age='20', Country='USA')

它允许使用列名如 row.UserName 和 row.Password 代替下标访问。需要注意的是这个只有在列名是合法的 Python 标识符的时候才生效。

除了使用命名分组之外，另外一个解决办法就是读取到一个字典序列中，示例如下：

import csv

with **open**('qiye.csv') **as** f:

f\_csv = csv.DictReader(f)

**for** row **in** f\_csv:

print row.**get**('UserName'),row.**get**('Password')

运行结果：

qiye qiye\_pass

Mary Mary\_pass

Jack Jack\_pass

这样就可以使用列名去访问每一行的数据了。比如，row[‘UserName’] 或者 row.get（‘UserName’）。

以上就是 CSV 文件读写的全部内容。接下来使用 lxml 解析 <http://seputu.com/>首页的标题、章节和链接等数据，代码如下：

**from** lxml **import** etree

**import** requests

user\_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'

headers={'User-Agent':user\_agent}

r = requests.get('http://seputu.com/',headers=headers)

*# 使用 lxml 解析网页*

html = etree.HTML(r.text)

div\_mulus = html.xpath('.// \*[@class="mulu"]')*# 先找到所有的 div class=mulu 标记*

**for** div\_mulu **in** div\_mulus:

*# 找到所有的 div\_h2 标记*

div\_h2 = div\_mulu.xpath('./div[@class="mulu-title"]/center/h2/text()')

**if** len(div\_h2)> 0:

h2\_title = div\_h2[0]

a\_s = div\_mulu.xpath('./div[@class="box"]/ul/li/a')

**for** a **in** a\_s:

*# 找到 href 属性*

href=a.xpath('./@href')[0]

*# 找到 title 属性*

box\_title = a.xpath('./@title')[0].encode('utf-8')

将 box\_title 数据抽取出来之后，里面的内容类似“[2014-10-2416：59：14]巫山妖棺第七十七章交心”这种形式。这里相较于5.1.1小节添加一步数据清洗，将内容里的时间和章节标题进行分离，这就要使用正则表达式，代码如下。

pattern = re.compile(r'\s\*\[(.\*)\]\s+(.\*)')

match = pattern.search(box\_title)

**if** match!=**None**:

date =match.group(1)

real\_title= match.group(2)

最后将获取的数据按照 title、real\_title、href、date 的格式写入到 CSV 文件中，解析存储的完整代码如下：

html = etree.HTML(r.text)

div\_mulus = html.xpath('.// \*[@class="mulu"]')*# 先找到所有的 div class=mulu 标记*

pattern = re.compile(r'\s\*\[(.\*)\]\s+(.\*)')

rows=[]

**for** div\_mulu **in** div\_mulus:

*# 找到所有的div\_h2标记*

div\_h2 = div\_mulu.xpath('./div[@class="mulu-title"]/center/h2/text()')

**if** len(div\_h2)> 0:

h2\_title = div\_h2[0].encode('utf-8')

a\_s = div\_mulu.xpath('./div[@class="box"]/ul/li/a')

**for** a **in** a\_s:

*# 找到href属性*

href=a.xpath('./@href')[0].encode('utf-8')

*# 找到title属性*

box\_title = a.xpath('./@title')[0]

pattern = re.compile(r'\s\*\[(.\*)\]\s+(.\*)')

match = pattern.search(box\_title)

**if** match!=**None**:

date =match.group(1).encode('utf-8')

real\_title= match.group(2).encode('utf-8')

*# print real\_title*

content=(h2\_title,real\_title,href,date)

**print** content

rows.append(content)

headers = ['title','real\_title','href','date']

**with** open('qiye.csv','w') **as** f:

f\_csv = csv.writer(f,)

f\_csv.writerow(headers)

f\_csv.writerows(rows)

运行效果如图5-4所示。

[](http://popimage/?src=)

图5-4　qiye.csv

注意

1）在存储 CSV 文件时，需要统一存储数据的类型。代码中使用 encode（‘utf-8’）作用就是将 title、real\_title、href、date 变量类型统一为 str。

2）5.1.1节 BeautifulSoup 如果使用 lxml 作为解析库，会发现解析出来的 HTML 内容缺失，这是由于 BeautifulSoup 为不同的解析器提供了相同的接口，但解析器本身是有区别的，同一篇文档被不同的解析器解析后可能会生成不同结构的树型文档。因此如果遇到缺失的情况，BeautifulSoup 可以使用 html.parser 作为解析器，或者单独使用 lxml 进行解析即可。

5.2　多媒体文件抽取

存储媒体文件主要有两种方式：只获取文件的 URL 链接，或者直接将媒体文件下载到本地。如果你采取的是第一种方式，只需看5.1节。本节主要讲解第二种方式，即将媒体文件下载下来。

本节主要介绍 urllib 模块提供的 urlretrieve（） 函数。urlretrieve（） 方法直接将远程数据下载到本地，函数原型如下：

urlretrieve(url, filename=**None**, reporthook=**None**, data=**None**)

参数说明：

* 参数 filename 指定了存储的本地路径（如果参数未指定，urllib 会生成一个临时文件保存数据。）
* 参数 reporthook 是一个回调函数。当连接上服务器以及相应的数据块传输完毕时会触发该回调函数，我们可以利用这个回调函数来显示当前的下载进度。
* 参数 data 指 post 到服务器的数据，该方法返回一个包含两个元素的（filename，headers）元组，filename 表示保存到本地的路径，header 表示服务器的响应头。

以天堂图片网为例（<http://www.ivsky.com/tupian/ziranfengguang/>），提取当前网址中的图片链接，并将图片下载到当前目录下。代码如下：

**import** urllib

**from** lxml **import** etree

**import** requests

**def** **Schedule**(blocknum,blocksize,totalsize):

'''''

blocknum:已经下载的数据块

blocksize:数据块的大小

totalsize:远程文件的大小

'''

per = 100.0 \* blocknum \* blocksize / totalsize

**if** per > 100 :

per = 100

**print** '当前下载进度：%d'%per

user\_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'

headers={'User-Agent':user\_agent}

r = requests.get('http://www.ivsky.com/tupian/ziranfengguang/',headers=headers)

*# 使用 lxml 解析网页*

html = etree.HTML(r.text)

img\_urls = html.xpath('.// img/@src')*# 先找到所有的 img*

i=0

**for** img\_url **in** img\_urls:

urllib.urlretrieve(img\_url,'img'+str(i)+'.jpg',Schedule)

i+=1

本程序中先从当前网址将 img 标记中的 src 属性提取出来，交给 urllib.urlretrieve 函数去下载，自动回调 Schedule 函数，显示当前下载的进度。Schedule 函数主要包括3个参数：blocknum（已经下载的数据块）、blocksize（数据块的大小）和 totalsize（远程文件的大小）。

5.3　Email 提醒

大家可能会奇怪 Email 在 Python 爬虫开发中有什么用呢？Email 主要起到提醒作用，当爬虫在运行过程中遇到异常或者服务器遇到问题，可以通过 Email 及时向自己报告。

发送邮件的协议是 STMP，Python 内置对 SMTP 的支持，可以发送纯文本邮件、HTML 邮件以及带附件的邮件。Python 对 SMTP 支持有 smtplib 和 email 两个模块，email 负责构造邮件，smtplib 负责发送邮件。

在讲解发送 Email 之前，首先申请一个163邮箱，开启 SMTP 功能，采用的是网易的电子邮件服务器 smtp.163.com，如图5-5所示。

[](http://popimage/?src=)

图5-5　163邮箱开启 SMTP

将 SMTP 开启之后，我们来构造一个纯文本邮件：

**from** email.mime.text **import** MIMEText

msg = MIMEText('Python 爬虫运行异常，异常信息为遇到 HTTP 403', 'plain', 'utf-8')

构造 MIMEText 对象时需要3个参数：

* 邮件正文。
* MIME 的 subtype，传入“plain”表示纯文本，最终的 MIME 就是“text/plain”。
* 设置编码格式，UTF-8 编码保证多语言兼容性。

接着设置邮件的发件人、收件人和邮件主题等信息，并通过 STMP 发送出去。代码如下：

**from** email.header **import** Header

**from** email.mime.text **import** MIMEText

**from** email.utils **import** parseaddr, formataddr

**import** smtplib

**def** **\_format\_addr**(s):

name, addr = parseaddr(s)

**return** formataddr((Header(name, 'utf-8').encode(), addr))

*# 发件人地址*

from\_addr = 'xxxxxxxx@163.com'

*# 邮箱密码*

password = 'pass'

*# 收件人地址*

to\_addr = 'xxxxxxxx@qq.com'

*# 163网易邮箱服务器地址*

smtp\_server = 'smtp.163.com '

*# 设置邮件信息*

msg = MIMEText('Python 爬虫运行异常，异常信息为遇到 HTTP 403', 'plain', 'utf-8')

msg['From'] = \_format\_addr('一号爬虫 <%s>' % from\_addr)

msg['To'] = \_format\_addr('管理员 <%s>' % to\_addr)

msg['Subject'] = Header('一号爬虫运行状态', 'utf-8').encode()

*# 发送邮件*

server = smtplib.SMTP(smtp\_server, 25)

server.login(from\_addr, password)

server.sendmail(from\_addr, [to\_addr], msg.as\_string())

server.quit()

有时候我们发送的可能不是纯文本，需要发送 HTML 邮件，将异常网页信息发送回去。在构造 MIMEText 对象时，把 HTML 字符串传进去，再把第二个参数由“plain”变为“html”就可以了。示例如下：

msg = MIMEText('<html><body><h1>Hello</h1>' +

'<p>异常网页<a href="http://www.cnblogs.com">cnblogs</a>...</p>' +

'</body></html>', 'html', 'utf-8')