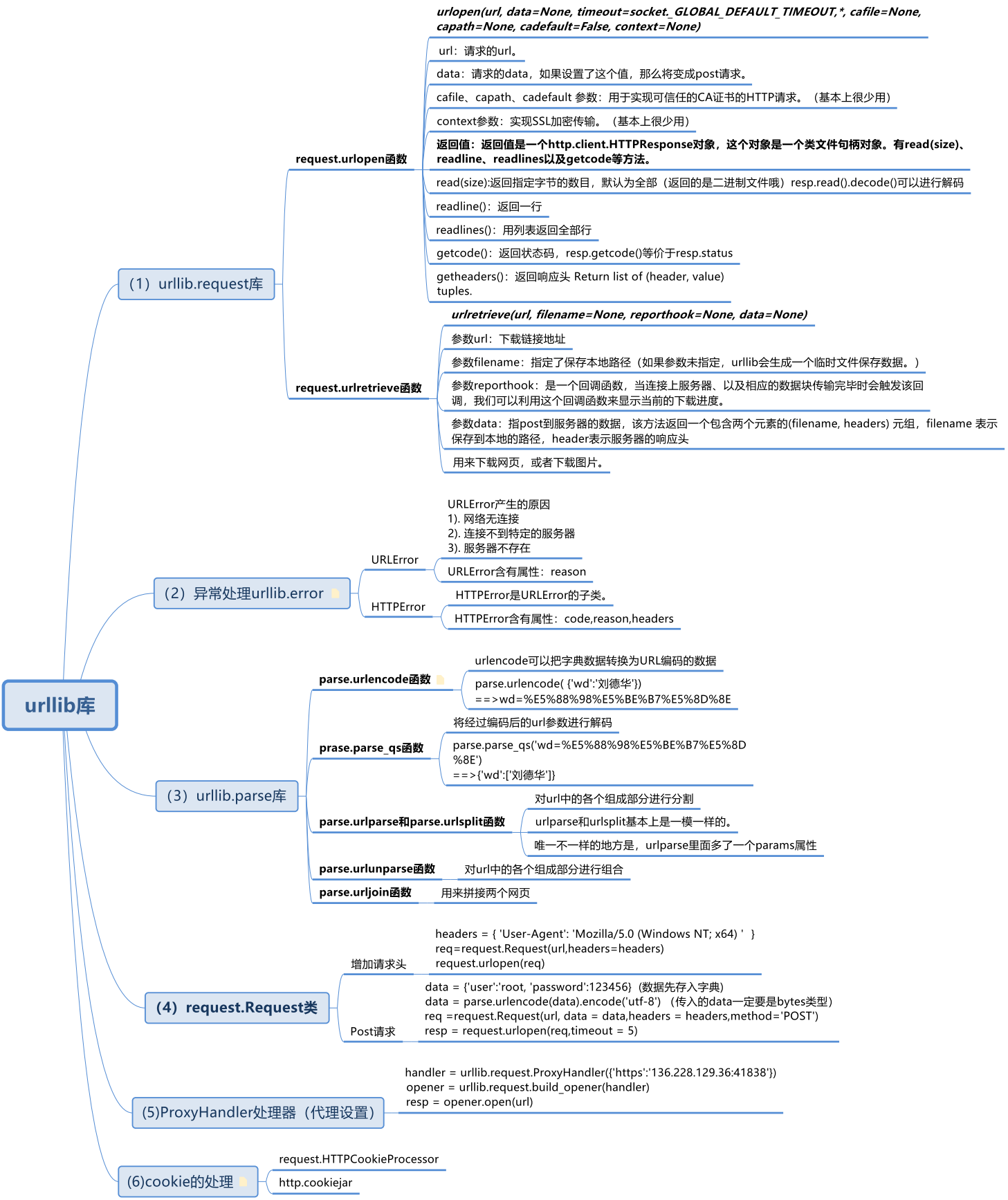

urllib库
（1）urllib.request库
request.urlopen函数
urlopen(url, data=None, timeout=socket._GLOBAL_DEFAULT_TIMEOUT,*, cafile=None, capath=None, cadefault=False, context=None)
url：请求的url。
data：请求的data，如果设置了这个值，那么将变成post请求。
cafile、capath、cadefault 参数：用于实现可信任的CA证书的HTTP请求。（基本上很少用）
context参数：实现SSL加密传输。（基本上很少用）
返回值：返回值是一个http.client.HTTPResponse对象，这个对象是一个类文件句柄对象。有read(size)、readline、readlines以及getcode等方法。
read(size):返回指定字节的数目，默认为全部（返回的是二进制文件哦）resp.read().decode()可以进行解码
readline()：返回一行
readlines()：用列表返回全部行
getcode()：返回状态码，resp.getcode()等价于resp.status
getheaders()：返回响应头 Return list of (header, value) tuples.
request.urlretrieve函数
urlretrieve(url, filename=None, reporthook=None, data=None)
参数url：下载链接地址
参数filename：指定了保存本地路径（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，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
用来下载网页，或者下载图片。
（2）异常处理urllib.error
URLError
URLError产生的原因
1). 网络无连接
2). 连接不到特定的服务器
3). 服务器不存在
URLError含有属性：reason
HTTPError
HTTPError是URLError的子类。
HTTPError含有属性：code,reason,headers
（3）urllib.parse库
parse.urlencode函数
urlencode可以把字典数据转换为URL编码的数据
parse.urlencode({'wd':'刘德华'})
==>wd=%E5%88%98%E5%BE%B7%E5%8D%8E
prase.parse_qs函数
将经过编码后的url参数进行解码
parse.parse_qs('wd=%E5%88%98%E5%BE%B7%E5%8D%8E')
==>{'wd':['刘德华']}
parse.urlparse和parse.urlsplit函数
对url中的各个组成部分进行分割
urlparse和urlsplit基本上是一模一样的。
唯一不一样的地方是，urlparse里面多了一个params属性
parse.urlunparse函数
对url中的各个组成部分进行组合
parse.urljoin函数
用来拼接两个网页
（4）request.Request类
增加请求头
headers = { 'User-Agent': 'Mozilla/5.0 (Windows NT; x64) ' }
req=request.Request(url,headers=headers)
request.urlopen(req)
Post请求
data = {'user':'root, 'password':123456}（数据先存入字典）
data = parse.urlencode(data).encode('utf-8') （传入的data一定要是bytes类型）
req =request.Request(url, data = data,headers = headers,method='POST')
resp = request.urlopen(req,timeout = 5)
(5)ProxyHandler处理器（代理设置）
handler = urllib.request.ProxyHandler({'https':'136.228.129.36:41838'})
opener = urllib.request.build_opener(handler)
resp = opener.open(url)
(6)cookie的处理
request.HTTPCookieProcessor
http.cookiejar

第一处:

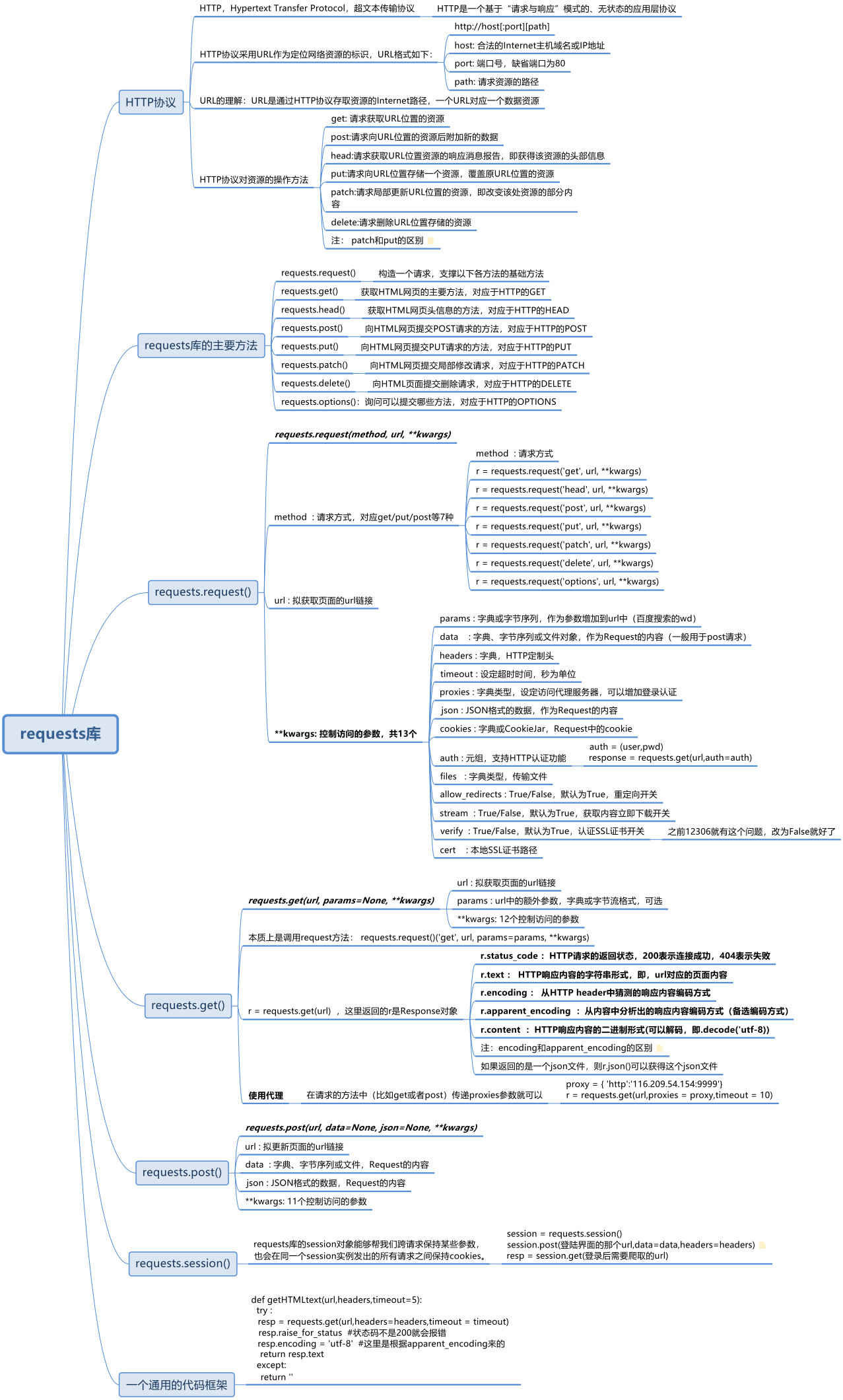
```
from urllib import request, error
import socket
try:
    response = request.urlopen('http://aaa.com',timeout=3)
except error.HTTPError as e:
    print("The server couldn't fulfill the request.")
    print('Error code: ', e.code)
except error.URLError as e:
    print('We failed to reach a server.')
    print('Reason: ', e.reason)
    if isinstance(e.reason,socket.timeout):
        print('连接超时')
else:
    print('Request Successfully')
```

第二处:

```
from urllib import request,parse
#resp = request.urlopen('http://www.baidu.com/s?wd=刘德华')
#UnicodeEncodeError: 'ascii' codec can't encode characters in
position 10-12: ordinal not in range(128)
parms = {'wd':'刘德华'}
url = 'http://www.baidu.com/s'
query_string = parse.urlencode(parms)
print(query_string)
url = url+'?'+query_string
resp = request.urlopen(url)
print(resp.getcode())
```

第三处:

```
from urllib import request,parse
from http.cookiejar import CookieJar
headers = {'User-Agent':"Mozilla/5.0 (Macintosh; U; Mac OS X Mach-O;
en-US; rv:2.0a) Gecko/20040614 Firefox/3.0.0 "}
def get_opener():
    cookjar = CookieJar()
    handler = request.HTTPCookieProcessor(cookjar)
    opener = request.build_opener(handler)
    return opener
def save_cookie(opener):
    data = {"email": "970138074@qq.com", "password": "pythonspider"}
    login_url = "http://www.renren.com/PLogin.do"
    req = request.Request(login_url,data=
parse.urlencode(data).encode('utf-8'),method='POST',headers=headers)
    opener.open(req)
def get_profile(opener):
    profile_url = 'http://www.renren.com/880151247/profile'
    req = request.Request(profile_url,headers=headers)
    resp = opener.open(req)
    print(resp.read().decode())
if __name__ == '__main__':
    opener = get_opener()
    save_cookie(opener)
    get_profile(opener)
```


requests库
HTTP协议
HTTP，Hypertext Transfer Protocol，超文本传输协议
HTTP是一个基于“请求与响应”模式的、无状态的应用层协议
HTTP协议采用URL作为定位网络资源的标识，URL格式如下：
http://host[:port][path]
host: 合法的Internet主机域名或IP地址
port: 端口号，缺省端口为80
path: 请求资源的路径
URL的理解：URL是通过HTTP协议存取资源的Internet路径，一个URL对应一个数据资源
HTTP协议对资源的操作方法
get: 请求获取URL位置的资源
post:请求向URL位置的资源后附加新的数据
head:请求获取URL位置资源的响应消息报告，即获得该资源的头部信息
put:请求向URL位置存储一个资源，覆盖原URL位置的资源
patch:请求局部更新URL位置的资源，即改变该处资源的部分内容
delete:请求删除URL位置存储的资源
注： patch和put的区别
requests库的主要方法
requests.request() 构造一个请求，支撑以下各方法的基础方法
requests.get() 获取HTML网页的主要方法，对应于HTTP的GET
requests.head() 获取HTML网页头信息的方法，对应于HTTP的HEAD
requests.post() 向HTML网页提交POST请求的方法，对应于HTTP的POST
requests.put() 向HTML网页提交PUT请求的方法，对应于HTTP的PUT
requests.patch() 向HTML网页提交局部修改请求，对应于HTTP的PATCH
requests.delete() 向HTML页面提交删除请求，对应于HTTP的DELETE
requests.options()：询问可以提交哪些方法，对应于HTTP的OPTIONS
requests.request()
requests.request(method, url, **kwargs)
method ：请求方式，对应get/put/post等7种
method ：请求方式
r = requests.request('get', url, **kwargs)
r = requests.request('head', url, **kwargs)
r = requests.request('post', url, **kwargs)
r = requests.request('put', url, **kwargs)
r = requests.request('patch', url, **kwargs)
r = requests.request('delete', url, **kwargs)
r = requests.request('options', url, **kwargs)
url : 拟获取页面的url链接
**kwargs: 控制访问的参数，共13个
params : 字典或字节序列，作为参数增加到url中（百度搜索的wd）
data ：字典、字节序列或文件对象，作为Request的内容（一般用于post请求）
headers : 字典，HTTP定制头
timeout : 设定超时时间，秒为单位
proxies : 字典类型，设定访问代理服务器，可以增加登录认证
json : JSON格式的数据，作为Request的内容
cookies : 字典或CookieJar，Request中的cookie
auth : 元组，支持HTTP认证功能
auth = (user,pwd)
response = requests.get(url,auth=auth)
files ：字典类型，传输文件
allow_redirects : True/False，默认为True，重定向开关
stream ：True/False，默认为True，获取内容立即下载开关
verify ：True/False，默认为True，认证SSL证书开关
之前12306就有这个问题，改为False就好了
cert ：本地SSL证书路径
requests.get()
requests.get(url, params=None, **kwargs)
url : 拟获取页面的url链接
params : url中的额外参数，字典或字节流格式，可选
**kwargs: 12个控制访问的参数
本质上是调用request方法： requests.request()('get', url, params=params, **kwargs)
r = requests.get(url) ，这里返回的r是Response对象
r.status_code ： HTTP请求的返回状态，200表示连接成功，404表示失败
r.text ： HTTP响应内容的字符串形式，即，url对应的页面内容
r.encoding ： 从HTTP header中猜测的响应内容编码方式
r.apparent_encoding ： 从内容中分析出的响应内容编码方式（备选编码方式）
r.content ： HTTP响应内容的二进制形式(可以解码，即.decode('utf-8))
注：encoding和apparent_encoding的区别
如果返回的是一个json文件，则r.json()可以获得这个json文件
使用代理
在请求的方法中（比如get或者post）传递proxies参数就可以
proxy = { 'http':'116.209.54.154:9999'}
r = requests.get(url,proxies = proxy,timeout = 10)
requests.post()
requests.post(url, data=None, json=None, **kwargs)
url : 拟更新页面的url链接
data ：字典、字节序列或文件，Request的内容
json : JSON格式的数据，Request的内容
**kwargs: 11个控制访问的参数
requests.session()
requests库的session对象能够帮我们跨请求保持某些参数，也会在同一个session实例发出的所有请求之间保持cookies。
session = requests.session()
session.post(登陆界面的那个url,data=data,headers=headers)
resp = session.get(登录后需要爬取的url)
一个通用的代码框架
def getHTMLtext(url,headers,timeout=5):
try :
resp = requests.get(url,headers=headers,timeout = timeout)
resp.raise_for_status #状态码不是200就会报错
resp.encoding = 'utf-8' #这里是根据apparent_encoding来的
return resp.text
except:
return ''

注： patch和put的区别

假设URL位置有一组数据UserInfo，包括UserID、UserName等20个字段
需求：用户修改了UserName，其他不变

- 采用PATCH，仅向URL提交UserName的局部更新请求
- 采用PUT，必须将所有20个字段一并提交到URL，未提交字段被删除

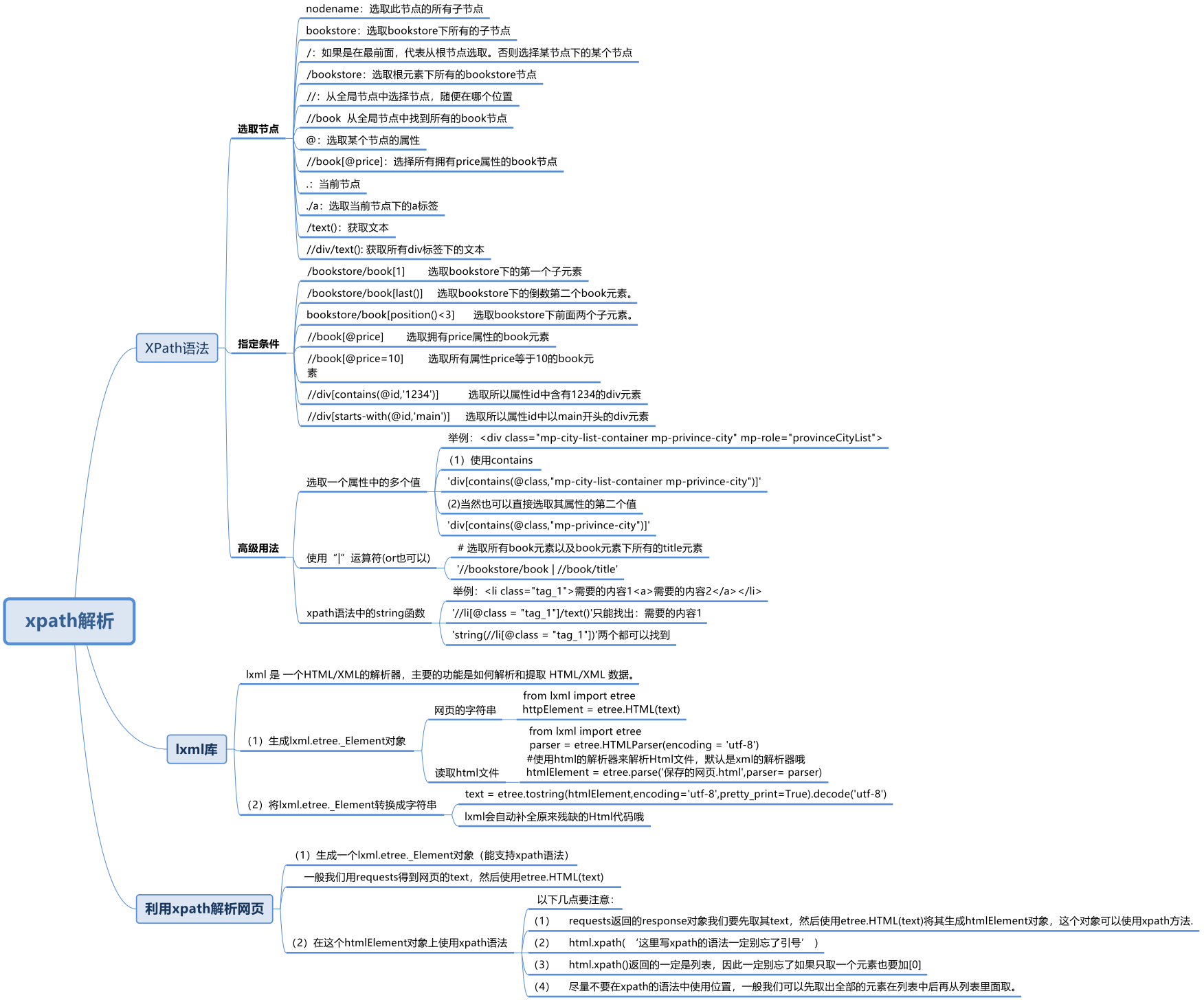
PATCH的最主要好处：节省网络带宽

注：encoding和apparent_encoding的区别

r.encoding：如果header中不存在charset，则认为编码为ISO-8859-1
r.text根据r.encoding显示网页内容（有可能会出错）
r.apparent_encoding：根据网页内容分析出的编码方式，准确性较高，但是分析需要时间

requests.session()的用法

```
import requests
url = "http://www.renren.com/PLogin.do"
data =
{"email":"97138074@qq.com",'password':"pythonspider"}
headers = {
    'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/62.0.3202.94 Safari/537.36"
}
# 登录
session = requests.session()
session.post(url,data=data,headers=headers)
# 访问大鹏个人中心
resp = session.get('http://www.renren.com/880151247/
profile')
print(resp.text)
```

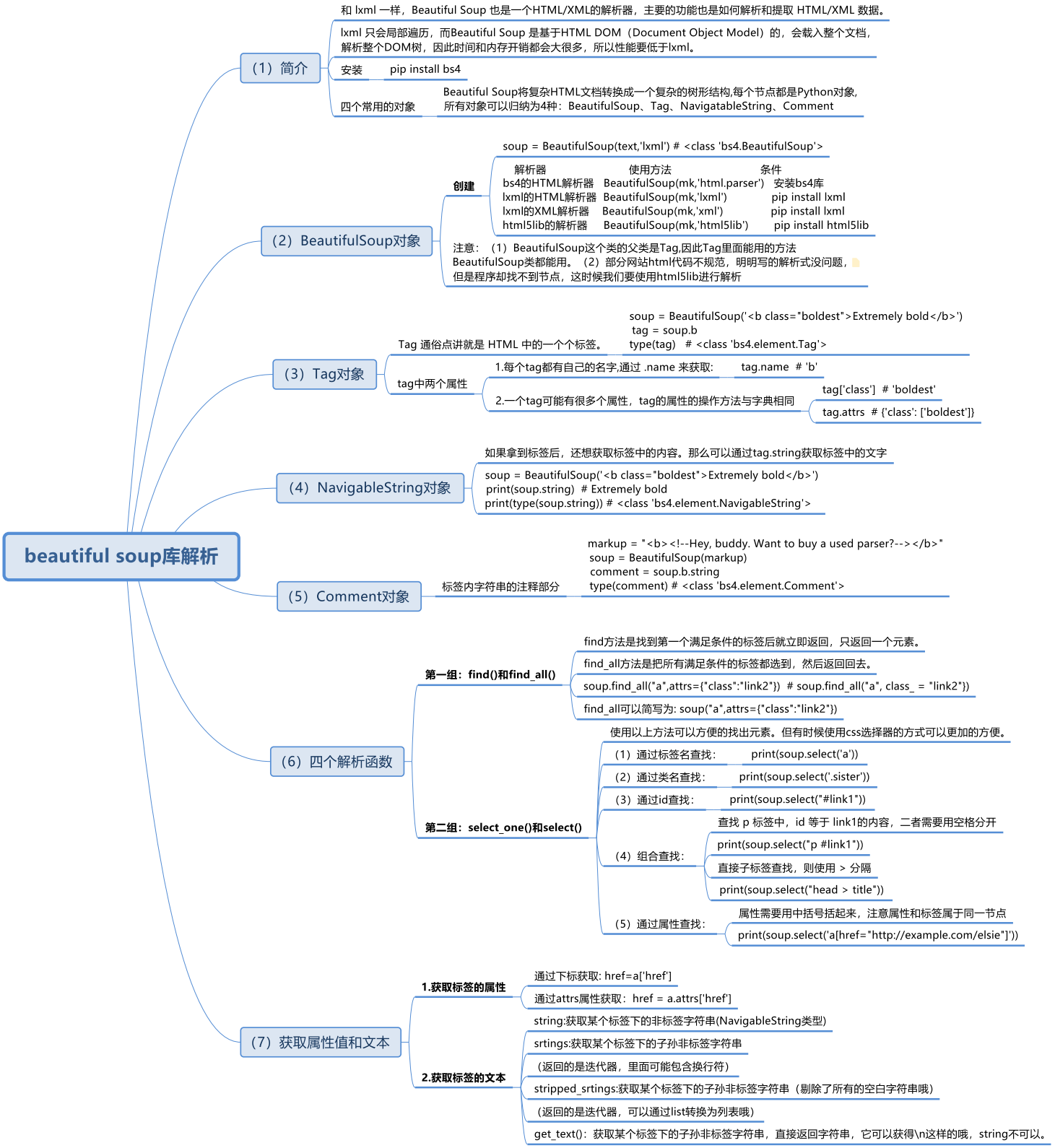

xpath解析
XPath语法
选取节点
nodename：选取此节点的所有子节点
bookstore：选取bookstore下所有的子节点
/：如果是在最前面，代表从根节点选取。否则选择某节点下的某个节点
/bookstore：选取根元素下所有的bookstore节点
//：从全局节点中选择节点，随便在哪个位置
//book 从全局节点中找到所有的book节点
@：选取某个节点的属性
//book[@price]：选择所有拥有price属性的book节点
.：当前节点
./a：选取当前节点下的a标签
/text()：获取文本
//div/text(): 获取所有div标签下的文本
指定条件
/bookstore/book[1] 选取bookstore下的第一个子元素
/bookstore/book[last()] 选取bookstore下的倒数第二个book元素。
bookstore/book[position()<3] 选取bookstore下前面两个子元素。
//book[@price] 选取拥有price属性的book元素
//book[@price=10] 选取所有属性price等于10的book元素
//div[contains(@id,'1234')] 选取所以属性id中含有1234的div元素
//div[starts-with(@id,'main')] 选取所以属性id中以main开头的div元素
高级用法
选取一个属性中的多个值
举例：
（1）使用contains
'div[contains(@class,"mp-city-list-container mp-privince-city")]'
(2)当然也可以直接选取其属性的第二个值
'div[contains(@class,"mp-privince-city")]'
使用“|”运算符(or也可以)
选取所有book元素以及book元素下所有的title元素
'//bookstore/book | //book/title'
xpath语法中的string函数
举例：<li class="tag_1">需要的内容1<a>需要的内容2
'//li[@class = "tag_1"]/text()'只能找出：需要的内容1
'string(//li[@class = "tag_1"])'两个都可以找到
lxml库
lxml 是 一个HTML/XML的解析器，主要的功能是如何解析和提取 HTML/XML 数据。
（1）生成lxml.etree._Element对象
网页的字符串
from lxml import etree
httpElement = etree.HTML(text)
读取html文件
from lxml import etree
parser = etree.HTMLParser(encoding = 'utf-8')
#使用html的解析器来解析Html文件，默认是xml的解析器哦
htmlElement = etree.parse('保存的网页.html',parser= parser)
（2）将lxml.etree._Element转换成字符串
text = etree.tostring(htmlElement,encoding='utf-8',pretty_print=True).decode('utf-8')
lxml会自动补全原来残缺的Html代码哦
利用xpath解析网页
（1）生成一个lxml.etree._Element对象（能支持xpath语法）
一般我们用requests得到网页的text，然后使用etree.HTML(text)
（2）在这个htmlElement对象上使用xpath语法
以下几点要注意：
（1） requests返回的response对象我们要先取其text，然后使用etree.HTML(text)将其生成htmlElement对象，这个对象可以使用xpath方法.
（2） html.xpath(‘这里写xpath的语法一定别忘了引号’)
（3） html.xpath()返回的一定是列表，因此一定别忘了如果只取一个元素也要加[0]
（4） 尽量不要在xpath的语法中使用位置，一般我们可以先取出全部的元素在列表中后再从列表里面取。

Xpath 扩展——XPath 轴

轴名称	结果
ancestor	选取当前节点的所有先辈（父、祖父等）。
ancestor-or-self	选取当前节点的所有先辈（父、祖父等）以及当前节点本身。
attribute	选取当前节点的所有属性。
child	选取当前节点的所有子元素。
descendant	选取当前节点的所有后代元素（子、孙等）。
descendant-or-self	选取当前节点的所有后代元素（子、孙等）以及当前节点本身。
following	选取文档中当前节点的结束标签之后的所有节点。
namespace	选取当前节点的所有命名空间节点。
parent	选取当前节点的父节点。
preceding	选取文档中当前节点的开始标签之前的所有节点。
preceding-sibling	选取当前节点之前的所有同级节点。
self	选取当前节点。

例子	结果
child::book	选取所有属于当前节点的子元素的 book 节点。
attribute::lang	选取当前节点的 lang 属性。
*child::**	选取当前节点的所有子元素。
*attribute::**	选取当前节点的所有属性。
child::text()	选取当前节点的所有文本子节点。
child::node()	选取当前节点的所有子节点。
descendant::book	选取当前节点的所有 book 后代。
ancestor::book	选择当前节点的所有 book 先辈。
ancestor-or-self::book	选取当前节点的所有 book 先辈以及当前节点（如果此节点是 book 节点）
child::/child::price*	选取当前节点的所有 price 孙节点。

Xpath 系统的学习资料：http://www.w3school.com.cn/xpath/index.asp


beautiful soup库解析
（1）简介
和 lxml 一样，Beautiful Soup 也是一个HTML/XML的解析器，主要的功能也是如何解析和提取 HTML/XML 数据。
lxml 只会局部遍历，而Beautiful Soup 是基于HTML DOM（Document Object Model）的，会载入整个文档，解析整个DOM树，因此时间和内存开销都会大很多，所以性能要低于lxml。
安装 pip install bs4
四个常用的对象
Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象，所有对象可以归纳为4种：BeautifulSoup、Tag、NavigatableString、Comment
（2）BeautifulSoup对象
soup = BeautifulSoup(text,'lxml') # <class 'bs4.BeautifulSoup'>
创建
解析器 使用方法 条件
bs4的HTML解析器 BeautifulSoup(mk,'html.parser') 安装bs4库
lxml的HTML解析器 BeautifulSoup(mk,'lxml') pip install lxml
lxml的XML解析器 BeautifulSoup(mk,'xml') pip install lxml
html5lib的解析器 BeautifulSoup(mk,'html5lib') pip install html5lib
注意：（1）BeautifulSoup这个类的父类是Tag,因此Tag里面能用的方法BeautifulSoup类都能用。（2）部分网站html代码不规范，明明写的解析式没问题，但是程序却找不到节点，这时候我们要使用html5lib进行解析
（3）Tag对象
soup = BeautifulSoup('Extremely bold')
tag = soup.b
type(tag) # <class 'bs4.element.Tag'>
Tag 通俗点讲就是 HTML 中的一个个标签。
tag中两个属性
1.每个tag都有自己的名字,通过 .name 来获取: tag.name # 'b'
2.一个tag可能有很多个属性，tag的属性的操作方法与字典相同
tag['class'] # 'boldest'
tag.attrs # {'class': ['boldest']}
（4）NavigableString对象
如果拿到标签后，还想获取标签中的内容。那么可以通过tag.string获取标签中的文字
soup = BeautifulSoup('Extremely bold')
print(soup.string) # Extremely bold
print(type(soup.string)) # <class 'bs4.element.NavigableString'>
（5）Comment对象
markup = "<!--Hey, buddy. Want to buy a used parser?-->"
soup = BeautifulSoup(markup)
comment = soup.b.string
type(comment) # <class 'bs4.element.Comment'>
标签内字符串的注释部分
（6）四个解析函数
第一组：find()和find_all()
find方法是找到第一个满足条件的标签后就立即返回，只返回一个元素。
find_all方法是把所有满足条件的标签都选到，然后返回回去。
soup.find_all("a",attrs={"class":"link2"}) # soup.find_all("a", class_ = "link2"})
find_all可以简写为: soup("a",attrs={"class":"link2"})
第二组：select_one()和select()
使用以上方法可以方便的找出元素。但有时候使用css选择器的方式可以更加的方便。
（1）通过标签名查找： print(soup.select('a'))
（2）通过类名查找： print(soup.select('.sister'))
（3）通过id查找： print(soup.select("#link1"))
（4）组合查找：
查找 p 标签中，id 等于 link1的内容，二者需要用空格分开
print(soup.select("p #link1"))
直接子标签查找，则使用 > 分隔
print(soup.select("head > title"))
（5）通过属性查找：
属性需要用中括号括起来，注意属性和标签属于同一节点
print(soup.select('a[href="http://example.com/elsie"]'))
（7）获取属性值和文本
1.获取标签的属性
通过下标获取: href=a['href']
通过attrs属性获取： href = a.attrs['href']
2.获取标签的文本
string:获取某个标签下的非标签字符串(NavigableString类型)
srtings:获取某个标签下的子孙非标签字符串
（返回的是迭代器，里面可能包含换行符）
stripped_srtings:获取某个标签下的子孙非标签字符串（剔除了所有的空白字符串哦）
（返回的是迭代器，可以通过list转换为列表哦）
get_text()：获取某个标签下的子孙非标签字符串，直接返回字符串，它可以获得\n这样的哦，string不可以。

优化不规则 html 代码的方法：

```
from bs4 import BeautifulSoup
def prettify_html(html_text):
    soup = BeautifulSoup(html_text,'html5lib')
    return soup.prettify()
```

prettify()为 HTML 文本<>及其内容增加更加'\n'

一个简单的实例的部分代码：

```
def bs4Paeser(urltext,goodlist):
    soup = BeautifulSoup(urltext,'lxml')
    #默认的解析器为 html.parser.
    #print(type(soup))
# 第一步：四种方法得到保存我们需要的数据的那一部分
    # 注意，find 返回的是<class 'bs4.element.Tag'>
    goodsTag = soup.find('div',class_="sousuoListBox clearfix")
    # 注意，find_all 返回的是<class 'bs4.element.ResultSet'>
    # 其可以看成一个列表，里面的每一个元素都是一个'bs4.element.Tag'类型
    goodsTag = soup.find_all('div',attrs={"class":"sousuoListBox clearfix"})[0]
    # 注意 select 利用 css 选择器语法，返回的是<class 'list'>列表
    # 里面的每一个元素都是一个'bs4.element.Tag'类型
    #但是，select_one 只返回第一个符合条件的，所以是'bs4.element.Tag'类型
    goodsTag = soup.select_one('div.sousuoListBox.clearfix')
    #这个 div 有两个类名,css 选择器连着写就行
    goodsTag = soup.select('body > div:nth-child(11) > div')
    #使用浏览器自带的 selector
# 第二步：找到每一种商品数据所在的那一部分，其可以视为返回一个列表
     goodslist = soup.find_all('div',class_ = "ssCardItem")
     #<class 'bs4.element.ResultSet'>
     goodslist = soup('div',class_ = "ssCardItem")
     #因为 find_all 这个方法特别常用，我们可以简写哦
    goodslist = soup.select('div.ssCardItem')  #<class 'list'>
    for everygood in goodslist:
        name = everygood.find('a',attrs={'class':"siteCardICH3"}).string
        #<class 'bs4.element.NavigableString'>
        href = everygood.find('a',attrs ={'class':"siteCardICH3"})['href']
        introduction = everygood.select_one('p.siteCardIC_p.souSuo').get_text()
        #<class 'str'>
```

正则re解析

什么是正则表达式

通俗理解：按照一定的规则，从某个字符串中匹配出想要的数据。这个规则就是正则表达式。

正则表达式常用匹配规则

#1 匹配某个字符串

```
text = 'hello, he is a boy!'
ret = re.match('he',text)  #match匹配的要从字符开头寻找
print(ret.group())  # he
```

#2 点.可以匹配任意的字符，默认除了换行符

```
text = "\nabc\n"
ret = re.match('.',text)
print(ret.group())  #AttributeError: 'NoneType' object has no attribute 'group'
ret = re.search('.',text) # search方法匹配的字符串可以在任何地方
print(ret.group())  # a
```

注意：加上flags = re.DOTALL或者re.S可以使 "." 特殊字符完全匹配任何字符，包括换行；没有这个标志， "." 匹配除了换行外的任何字符

#3 \d匹配任意的数字

```
text = '喂，110, he is a boy!'
ret = re.search('\d\d\d',text)
print(ret.group())  # 110
```

#4 \D匹配任意的非数字

```
text = '喂，110, he is a boy!'
ret = re.search('\D',text)
print(ret.group())  # 喂
```

#5 \s匹配任意的空白字符（\n \r \t 空格）

```
text = '喂\t110, he is a boy!'
ret = re.search('\s',text)
print(ret.group())  #
```

#6 \S匹配任意的非空白字符

```
text = '喂\t110, he is a boy!'
ret = re.search('\S',text)
print(ret.group())  #喂
```

#7 \w匹配的是a-z A-Z以及数字和下划线_

```
text = '一1二2'
print(re.findall(r'\w',text,re.A))  # ['1', '2']
print(re.findall(r'\w',text))  # ['一', '1', '二', '2']
```

#8 \W匹配的是除了a-z A-Z以及数字和下划线_以为的其他字符

#9 [ ]组合的方式，只要满足中括号中的某一项都算匹配成功

之前讲到的几种匹配规则，其实可以使用中括号的形式来进行替代：

(1) \d：[0-9]
(2) \D：0-9
(3) \w：[0-9a-zA-Z_]
(4) \W：[^0-9a-zA-Z_]

匹配多个字符

*：可以匹配0或者任意多个字符。示例代码如下：

```
text = "0731"
ret = re.match('\d*',text)
print(ret.group())  #0731
```

+：可以匹配1个或者多个字符。最少一个。示例代码如下：

```
text = "abc"
ret = re.match('\w+',text)
print(ret.group())  #abc
```

?：匹配的字符可以出现一次或者不出现（0或者1）。示例代码如下：

```
text = "123"
ret = re.match('\d?',text)
print(ret.group())  # 1
```

{m}：匹配m个字符。示例代码如下：

```
text = "123"
ret = re.match('\d{2}',text)
print(ret.group())  #12
```

{m,n}：匹配m-n个字符。在这中间的字符都可以匹配到。示例代码如下：

```
text = "123"
ret = re.match('\d{1,2}',text)
print(ret.group())  #12
```

^（脱字号）：表示以...开始：

```
text = "hello"
ret = re.match('^h',text)
print(ret.group())  #h
```

注意：如果是在中括号中，那么代表的是取反操作.

$：表示以...结束：

```
# 匹配163.com的邮箱
text = "xxx@163.com"
ret = re.search('\w+@163\.com$',text)
print(ret.group())  #xxx@163.com
```

|：匹配多个表达式或者字符串：

小例子：验证URL：（注意下面一定要加表示分组的括号哦）

```
text = "http://www.baidu.com/"
ret = re.match('(http|https|ftp)://[^\s]+',text)
print(ret.group())  #http://www.baidu.com/
```

我们知道正常情况下，\w匹配字母数字及下划线，相当于[A-Za-z0-9_]。
在python 3中我们试下\w的匹配字 符串的时候,会发现匹配会匹配到中文汉字。
这是什么原因呢？在python 3里面,默认的是Unicode编码。正则也是默认的编码模式。
我们知道unicode编码由字母和数字构成。这就造成了\w可以匹配到中文。
那么怎么处理呢？
我们需要将正则的匹配模式修改为二进制匹配，就会得到正确的结果，即flags设置为re.A就可以了。

正则re解析

贪婪模式和非贪婪模式

贪婪模式：正则表达式会匹配尽量多的字符。默认是贪婪模式。

示例代码如下：

```
1.  text = "0123456"
2.  ret = re.match('\d+',text)
3.  print(ret.group())  #0123456
4.  # 因为默认采用贪婪模式，所以会输出0123456
```

非贪婪模式：正则表达式会尽量少的匹配字符。

只需要在* + ?或者{}这些符号后面再加个?就可以了

```
1.  text = "0123456"
2.  ret = re.match('\d+?',text)
3.  print(ret.group())  #0
```

在爬虫中，多半都是非贪婪模式哦

转义字符和原生字符串

在正则表达式中，有些字符是有特殊意义的字符。因此如果想要匹配这些字符，那么就必须使用反斜杠进行转义。

```
1.  text = "apple price is \$99,orange paice is $88"
2.  ret = re.search('\$(\d+)',text)
3.  print(ret.group())  # $99
```

在正则表达式中，\是专门用来做转义的。在Python中\也是用来做转义的。
因此如果想要在普通的字符串中匹配出\，那么要给出四个\。

```
1.  text = "apple \c"
2.  ret = re.search('\\\\c',text)
3.  print(ret.group())  #\c
```

要使用原生字符串就可以解决这个问题：

```
1.  text = "apple \c"
2.  ret = re.search(r'\\c',text)
3.  print(ret.group())  #\c
```

re模块中常用函数

match

从开始的位置进行匹配。如果开始的位置没有匹配到。就直接失败了。

```
1.  text = 'hello'
2.  ret = re.match('h',text)
3.  print(ret.group())  #h
```

如果第一个字母不是h，那么就会失败。示例代码如下：

```
1.  text = 'ahello'
2.  ret = re.match('h',text)
3.  print(ret.group())
4.  >> AttributeError: 'NoneType' object has no attribute 'group'
```

search

在字符串中找满足条件的字符。如果找到，就返回。

```
1.  text = 'apple price $99 orange price $88'
2.  ret = re.search('\d+',text)
3.  print(ret.group())  #99
```

group分组

在正则表达式中，可以对过滤到的字符串进行分组。分组使用圆括号()的方式。
group()：它和group(0)是等价的，返回的是整个满足条件的字符串。
groups：返回的是里面的子组，用元组返回。
group(1)：返回的是第一个子组，可以传入多个。

举例

```
1.  text = 'apple is $5, orange is $17.'
2.  ret = re.search(r'.*(\$\d+).*(\$\d+).*',text)
3.  print(ret.group()) # apple is $5, orange is $17.
4.  print(ret.group(0)) # apple is $5, orange is $17.
5.  print(ret.group(1)) # $5
6.  print(ret.group(2)) # $17
7.  print(ret.group(1,2)) # ('$5', '$17') 返回了一个元组
8.  print(ret.groups()) # ('$5', '$17')
```

findall

找出所有满足条件的，返回的是一个列表。

```
1.  text = 'apple price $99 orange price $88'
2.  ret = re.findall('\d+',text)
3.  print(ret)
4.  # ['99', '88']
```

sub

用来替换字符串。将匹配到的字符串替换为其他字符串。

```
1.  text = 'apple price $99 orange price $88'
2.  ret = re.sub('\d+','0',text)
3.  print(ret)  #>> apple price $0 orange price $0
```

split

使用正则表达式来分割字符串。

```
1.  text = "hello world ni hao"
2.  ret = re.split('\W',text)
3.  print(ret)
4.  # ["hello","world","ni","hao"]
```

compile

对于一些经常要用到的正则表达式，可以使用compile进行编译，后期再使用的时候可以直接拿过来用，执行效率会更快。

```
1.  import re
2.  r = re.compile(r'.{2}')
3.  text = '123456\n789abc'
4.  ret = re.search(r,text)  #12
```

注意：如果传入的是compile对象，那么在search,match,findall等方法里面不允许有flags这个参数，因为此时正则表达式已经被编译完成了储存在内存中了。

Json储存

什么是json

JSON(JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。

JSON支持数据格式：

- 对象（字典）:使用花括号。
- 数组（列表）:使用方括号。
- 整形、浮点型、布尔类型还有null类型。
- 字符串类型（字符串必须要用双引号，不能用单引号）。
- 多个数据之间使用逗号分开。
- 注意：json本质上就是一个字符串。
- 在线解析的网站：https://www.json.cn/

字典和列表转JSON：

在Python中。只有基本数据类型才能转换成JSON格式的字符串。
也即：int、float、str、list、dict、tuple。

（1）不写入文件，直接转换 （json.dumps）

```
1.  import json
2.  school = [
3.      {
4.          'name':'清华大学',
5.          'type':'偏理'
6.      },
7.      {
8.          'name':'北京大学',
9.          'type':'偏文'
10.     }
11. ]
12. print(type(school))  #<class 'list'>
13.  json_str = json.dumps(school)
14. print(type(json_str))  #<class 'str'>
15. print(json_str)
#会发现里面出现了\u6e05\u534e\u5927\u5b66这样的转义符号
```

因为json在dump的时候，只能存放ascii的字符，因此会将中文进行转义，这时候我们可以使用ensure_ascii=False关闭这个特性。

```
16. json_str = json.dumps(school,ensure_ascii=False)
17. print(json_str)
# [{"name": "清华大学", "type": "偏理"}, {"name": "北京大学", "type": "偏文"}]
```

（2）写入文件 （json.dump）

```
1.  with open('school.json','w',encoding='utf-8') as f:
2.      json.dump(school,f,ensure_ascii=False)  #第二个参数就是要写入的文件指针
```

注意：如果这里面传入的有中文，那么我们的文件的编码要改成utf-8而且，这里的ensure_ascii也要关闭哦

json字符串转python对象

（1）将一个json字符串load成Python对象：（json.loads）

```
1.  json_str = '[{"title": "钢铁是怎样练成的", "price": 9.8},
{"title": "红楼梦", "price": 9.9}]'
2.  books = json.loads(json_str)
3.  print(type(books))  # <class 'list'>
4.  print(books)
5.  #[{'title': '钢铁是怎样练成的', 'price': 9.8}, {'title': '红楼梦', 'price': 9.9}]
```

（2）直接从文件中读取json：（json.load）

```
1.  import json
2.  with open('school.json','r',encoding='utf-8') as f:
3.      json_str = json.load(f)
4.      print(json_str)
5.  #[{'name': '清华大学', 'type': '偏理'}, {'name': '北京大学', 'type': '偏文'}]
```

注意：这里制定encoding =‘ utf-8’ 是因为这个文件是以utf-8储存的。

csv文件存储

读取csv文件：

csv文件：股票数据

```
时间,收盘价,涨跌
2017/2/20,70.05,涨
2017/2/21,71.58,涨
2017/2/22,70.56,跌
```

（1）直接读取每一行

```
import csv
with open('股票数据.csv' ,'r') as f:
 reader = csv.reader(f)  #reader是一个迭代器哦
 header = next(reader)
 print('header:',header)
 for x in reader:
   print(x)
```

```
header: ['时间', '收盘价', '涨跌']
['2017/2/20', '70.05', '涨']
['2017/2/21', '71.58', '涨']
['2017/2/22', '70.56', '跌']
['2017/2/24', '73.65', '涨']
['2017/2/25', '77.85', '涨']
```

（2）按照字典输出每一行

```
import csv
with open('股票数据.csv' ,'r',errors='ignore') as f:
 reader = csv.DictReader(f)  #reader是一个迭代器哦
 print(type(reader))  #<class 'csv.DictReader'>
 for x in reader:
   print(x)
# 编码默认 encoding = 'gbk'
# errors='ignore'可以忽略编码错误哦
```

```
<class 'csv.DictReader'>
OrderedDict([('时间', '2017/2/20'), ('收盘价', '70.05'), ('涨跌', '涨')])
OrderedDict([('时间', '2017/2/21'), ('收盘价', '71.58'), ('涨跌', '涨')])
OrderedDict([('时间', '2017/2/22'), ('收盘价', '70.56'), ('涨跌', '跌')])
OrderedDict([('时间', '2017/2/24'), ('收盘价', '73.65'), ('涨跌', '涨')])
OrderedDict([('时间', '2017/2/25'), ('收盘价', '77.85'), ('涨跌', '涨')])
```

写入csv文件：

写入数据到csv文件，需要创建一个writer对象，主要用到两个方法。一个是writerow，这个是写入一行。一个是writerows，这个是写入多行。示例代码如下：

（1）直接写入每一行

```
import csv
header = ['时间', '收盘价', '涨跌']
data  = [['2017/2/20', '70.05', '涨'],
     ['2017/2/21', '71.58', '涨'],
     ['2017/2/22', '70.56', '跌']]
with open('stock.csv','w',encoding = 'utf-8',newline='') as f:
#这里写入文件时要以空格结尾，默认是换行
  writer = csv.writer(f)
 writer.writerow(header) #先写入一行，即标题行
 writer.writerows(data)  #再写入多行，即数据行
```

（2）将字典写入每一行

```
import csv
header = ['时间', '收盘价', '涨跌']
data  = [{'时间':'2017/2/20','收盘价': '70.05','涨跌': '涨'},
    {'时间':'2017/2/21','收盘价': '71.58', '涨跌':'涨'},
   {'时间':'2017/2/22','收盘价': '70.56', '涨跌':'跌'}]
with open('stock.csv','w',encoding = 'utf-8',newline='') as f:
  writer = csv.DictWriter(f,header) #这里要指定我们的字典key
   writer.writeheader()  #使用这个方法保存我们的标题到文件中
   writer.writerows(data)
```