爬虫实现-urllib

主讲：孙国元

华信培训

本章要点

- urllib基本库
- 正则表达式
- Beautiful Soup

1

urllib基本库

概述

- 在Python 2中，有urllib和urllib2两个库来实现请求的发送。而在Python 3中，已经不存在urllib2这个库了，统一为urllib，其官方文档链接为：https://docs.python.org/3/library/urllib.html。
- urllib包含如下4个模块：
  - request：它是最基本的HTTP请求模块，可以用来模拟发送请求。就像在浏览器里输入网址然后回车一样，只需要给库方法传入URL以及额外的参数，就可以模拟实现这个过程了。
  - error：异常处理模块，如果出现请求错误，我们可以捕获这些异常，然后进行重试或其他操作以保证程序不会意外终止。
  - parse：一个工具模块，提供了许多URL处理方法，比如拆分、解析、合并等。
  - robotparser：主要是用来识别网站的robots.txt文件，然后判断哪些网站可以爬，哪些网站不可以爬，它其实用得比较少。

发送请求

- urlopen()
  - urllib.request模块提供了最基本的构造HTTP请求的方法，利用它可以模拟浏览器的一个请求发起过程，同时它还带有处理授权验证（authenticaton）、重定向（redirection)、浏览器Cookies以及其他内容。

```
import urllib.request

response = urllib.request.urlopen("http://www.baidu.com")
print(response.read().decode("utf-8"))
```

处理响应

- HTTPResposne
  - 主要包含read()、readinto()、getheader(name)、getheaders()、fileno()等方法，以及msg、version、status、reason、debuglevel、closed等属性。

```
import urllib.request

response = urllib.request.urlopen("https://www.python.org")
print(response.status)
print(response.getheaders())
```

传递参数

- data参数
  - data参数是可选的。如果要添加该参数，并且如果它是字节流编码格式的内容，即bytes类型，则需要通过bytes()方法转化。另外，如果传递了这个参数，则它的请求方式就不再是GET方式，而是POST方式。

```
import urllib.parse
import urllib.request

data = bytes(urllib.parse.urlencode({'word': 'hello'}), encoding='utf8')
response = urllib.request.urlopen('http://httpbin.org/post', data=data)
print(response.read())
```

超时时间

- timeout参数
  - imeout参数用于设置超时时间，单位为秒，意思就是如果请求超出了设置的这个时间，还没有得到响应，就会抛出异常。如果不指定该参数，就会使用全局默认时间。

```
import urllib.request

response = urllib.request.urlopen('http://httpbin.org/get', timeout=1)
print(response.read())
```

Request

- 我们知道利用urlopen()方法可以实现最基本请求的发起，但这几个简单的参数并不足以构建一个完整的请求。如果请求中需要加入Headers等信息，就可以利用更强大的Request类来构建。

Request

- Request参数

```
classc  urllib.request.Request(url, data=None, headers={}, origin_req_host=None, unverifiable=False, method=None)
```

- 第一个参数url用于请求URL，这是必传参数，其他都是可选参数。
- 第二个参数data如果要传，必须传bytes（字节流）类型的。如果它是字典，可以先用urllib.parse模块里的urlencode()编码。
- 第三个参数headers是一个字典，它就是请求头，我们可以在构造请求时通过headers参数直接构造，也可以通过调用请求实例的add_header()方法添加。
- 第四个参数origin_req_host指的是请求方的host名称或者IP地址。
- 第五个参数unverifiable表示这个请求是否是无法验证的，默认是False，意思就是说用户没有足够权限来选择接收这个请求的结果。例如，我们请求一个HTML文档中的图片，但是我们没有自动抓取图像的权限，这时unverifiable的值就是True`。
- 第六个参数method是一个字符串，用来指示请求使用的方法，比如GET、POST和PUT等。

Request

```
from urllib import request, parse

url = 'http://httpbin.org/post'
headers = {
    'User-Agent': 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)',
    'Host': 'httpbin.org'
}
dict = {
    'name': 'Germey'
}
data = bytes(parse.urlencode(dict), encoding='utf8')
req = request.Request(url=url, data=data, headers=headers, method='POST')
response = request.urlopen(req)
print(response.read().decode('utf-8'))
```

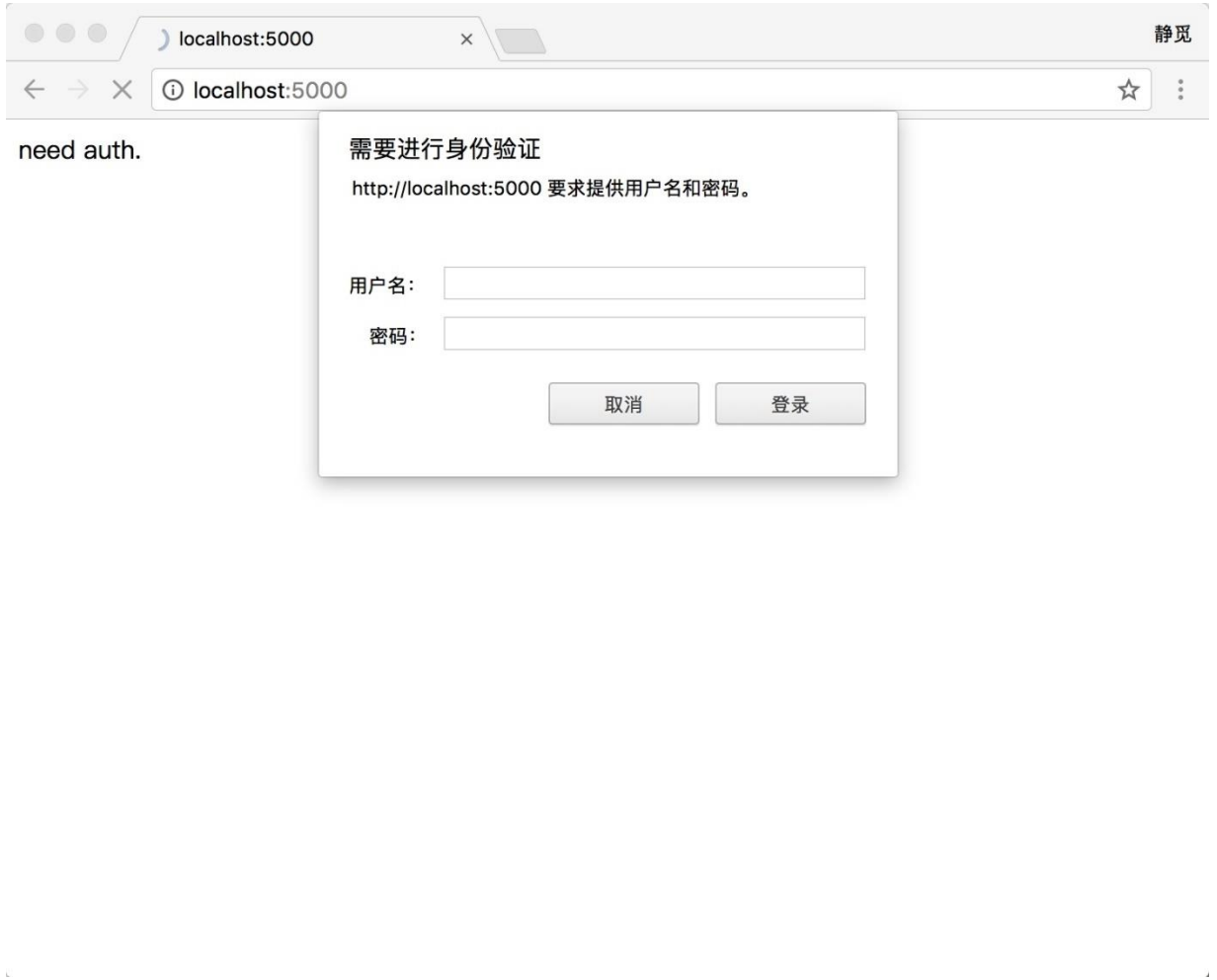
高级用法

- 可以把Handler理解为各种处理器，有专门处理登录验证的，有处理Cookies的，有处理代理设置的。利用它们，我们几乎可以做到HTTP请求中所有的事情。
- 各种Handler子类继承这个BaseHandler类：
  - HTTPDefaultErrorHandler：用于处理HTTP响应错误，错误都会抛出HTTPError类型的异常。
  - HTTPRedirectHandler：用于处理重定向。
  - HTTPCookieProcessor：用于处理Cookies。
  - ProxyHandler：用于设置代理，默认代理为空。
  - HTTPPasswordMgr：用于管理密码，它维护了用户名和密码的表。
  - HTTPBasicAuthHandler：用于管理认证，如果一个链接打开时需要认证，那么可以用它来解决认证问题。

Opener

- 之前使用的Request和urlopen()相当于类库为你封装好了极其常用的请求方法，利用它们可以完成基本的请求，但是现在不一样了，我们需要实现更高级的功能，所以需要深入一层进行配置，使用更底层的实例来完成操作，所以这里就用到了Opener。
- 一般利用Handler来构建Opener。

验证


localhost:5000
静觅
localhost:5000
need auth.
需要进行身份验证
http://localhost:5000 要求提供用户名和密码。
用户名:
密码:
取消
登录

验证

```
from urllib.request import HTTPPasswordMgrWithDefaultRealm, HTTPBasicAuthHandler, build_opener
from urllib.error import URLError

username = 'username'
password = 'password'
url = 'http://localhost:5000/'

p = HTTPPasswordMgrWithDefaultRealm()
p.add_password(None, url, username, password)
auth_handler = HTTPBasicAuthHandler(p)
opener = build_opener(auth_handler)

try:
    result = opener.open(url)
    html = result.read().decode('utf-8')
    print(html)
except URLError as e:
    print(e.reason)
```

代理

```
from urllib.error import URLError
from urllib.request import ProxyHandler, build_opener

proxy_handler = ProxyHandler({
    'http': 'http://127.0.0.1:9743',
    'https': 'https://127.0.0.1:9743'
})
opener = build_opener(proxy_handler)
try:
    response = opener.open('https://www.baidu.com')
    print(response.read().decode('utf-8'))
except URLError as e:
    print(e.reason)
```

Cookies

- 直接输出

```
import http.cookiejar, urllib.request

cookie = http.cookiejar.CookieJar()
handler = urllib.request.HTTPCookieProcessor(cookie)
opener = urllib.request.build_opener(handler)
response = opener.open('http://www.baidu.com')
for item in cookie:
    print(item.name+"="+item.value)
```

Cookies

- 保存文件

```
import http.cookiejar, urllib.request

filename = 'cookies.txt'
cookie = http.cookiejar.MozillaCookieJar(filename)
handler = urllib.request.HTTPCookieProcessor(cookie)
opener = urllib.request.build_opener(handler)
response = opener.open('http://www.baidu.com')
cookie.save(ignore_discard=True, ignore_expires=True)
```

Cookies

- 读取Cookie

```
import http.cookiejar, urllib.request

filename = 'cookies.txt'
cookie = http.cookiejar.MozillaCookieJar()
cookie.load('cookies.txt', ignore_discard=True, ignore_expires=True)
handler = urllib.request.HTTPCookieProcessor(cookie)
opener = urllib.request.build_opener(handler)
response = opener.open('http://www.baidu.com')
print(response.read().decode('utf-8'))
```

处理异常

- urllib的error模块定义了由request模块产生的异常。如果出现了问题，request模块便会抛出error模块中定义的异常。

URLError

- URLError类来自urllib库的error模块，它继承自OSError类，是error异常模块的基类，由request模块生的异常都可以通过捕获这个类来处理。
  - 它具有一个属性reason，即返回错误的原因。

```
from urllib import request, error
try:
    response = request.urlopen('http://www.baidu.com/index.htm')
except error.URLError as e:
    print(e.reason)
```

HTTPError

- 它是URLError的子类，专门用来处理HTTP请求错误，比如认证请求失败等。它有如下3个属性。
  - code：返回HTTP状态码，比如404表示网页不存在，500表示服务器内部错误等。
  - reason：同父类一样，用于返回错误的原因。
  - headers：返回请求头。

```
from urllib import request,error
try:
    response = request.urlopen('http://www.baidu.com/index.htm')
except error.HTTPError as e:
    print(e.reason, e.code, e.headers, sep='\n')
```

一般规范

- URLError是HTTPError的父类，所以可以先选择捕获子类的错误，再去捕获父类的错误

```
from urllib import request, error

try:
    response = request.urlopen('http://www.baidu.com/index.htm')
except error.HTTPError as e:
    print(e.reason, e.code, e.headers, sep='\n')
except error.URLError as e:
    print(e.reason)
else:
    print('Request Successfully')
```

解析链接

- urllib库里还提供了parse这个模块，它定义了处理URL的标准接口。

url解析

- urlparse()
  - 该方法可以实现URL的识别和分段。

```
from urllib.parse import urlparse

result = urlparse('http://www.baidu.com/index.html;user?id=5#comment')
print(type(result), result)
```

  - 返回结果是一个ParseResult类型的对象，它包含6部分，分别是scheme、netloc、path、params、query和fragment

url解析

- urlunparse()
  - 它接受的参数是一个可迭代对象，但是它的长度必须是6，否则会抛出参数数量不足或者过多的问题。

```
from urllib.parse import urlunparse

data = ['http', 'www.baidu.com', 'index.html', 'user', 'a=6', 'comment']
print(urlunparse(data))
```

url解析

- urlsplit()
  - 这个方法和urlparse()方法非常相似，只不过它不再单独解析params这一部分，只返回5个结果。

```
from urllib.parse import urlsplit

result = urlsplit('http://www.baidu.com/index.html;user?id=5#comment')
print(result)
```

  - SplitResult(scheme='http', netloc='www.baidu.com', path='/index.html;user', query='id=5', fragment='comment')

url解析

- urlunsplit()
  - 与urlunparse()类似，它也是将链接各个部分组合成完整链接的方法，传入的参数也是一个可迭代对象，例如列表、元组等，唯一的区别是长度必须为5。

```
from urllib.parse import urlunsplit

data = ['http', 'www.baidu.com', 'index.html', 'a=6', 'comment']
print(urlunsplit(data))
```

url解析

- urlencode()
  - 序列化为GET请求参数

```
from urllib.parse import urlencode

params = {
    'name': 'germey',
    'age': 22
}
base_url = 'http://www.baidu.com?'
url = base_url + urlencode(params)
print(url)
```

url解析

- parse_qs()
  - 利用parse_qs()方法，就可以将它转回字典

```
from urllib.parse import parse_qs

query = 'name=germey&age=22'
print(parse_qs(query))
```

url解析

- quote()
  - 该方法可以将内容转化为URL编码的格式。URL中带有中文参数时，有时可能会导致乱码的问题，此时用这个方法可以将中文字符转化为URL编码

```
from urllib.parse import quote

keyword = '壁纸'
url = 'https://www.baidu.com/s?wd=' + quote(keyword)
print(url)
```

url解析

- unquote()
  - 进行URL解码

```
from urllib.parse import unquote

url = 'https://www.baidu.com/s?wd=%E5%A3%81%E7%BA%B8'
print(unquote(url))
```

Robots协议

- Robots协议也称作爬虫协议、机器人协议，它的全名叫作网络爬虫排除标准（Robots Exclusion Protocol），用来告诉爬虫和搜索引擎哪些页面可以抓取，哪些不可以抓取。它通常是一个叫作robots.txt的文本文件，一般放在网站的根目录下。
- 当搜索爬虫访问一个站点时，它首先会检查这个站点根目录下是否存在robots.txt文件，如果存在，搜索爬虫会根据其中定义的爬取范围来爬取。如果没有找到这个文件，搜索爬虫便会访问所有可直接访问的页面。
- 一个robots.txt的样例

```
User-agent: *
Disallow: /
Allow: /public/
```

Robots协议

- robotparser
  - 可以使用robotparser模块来解析robots.txt了。该模块提供了一个类RobotFileParser，它可以根据某网站的robots.txt文件来判断一个爬取爬虫是否有权限来爬取这个网页。

```
from urllib.robotparser import RobotFileParser

rp = RobotFileParser()
rp.set_url('http://www.jianshu.com/robots.txt')
rp.read()
print(rp.can_fetch('*', 'http://www.jianshu.com/p/b67554025d7d'))
print(rp.can_fetch('*', "http://www.jianshu.com/search?q=python&page=1&type=collections"))
```

2

正则表达式

正则表达式

- 常用的匹配规则

模式	描述
\w	匹配字母、数字及下划线
\W	匹配不是字母、数字及下划线的字符
\s	匹配任意空白字符，等价于[\t\n\r\f]
\S	匹配任意非空字符
\d	匹配任意数字，等价于[0-9]
\D	匹配任意非数字的字符
\A	匹配字符串开头
\Z	匹配字符串结尾，如果存在换行，只匹配到换行前的结束字符串
\z	匹配字符串结尾，如果存在换行，同时还会匹配换行符
\G	匹配最后匹配完成的位置
\n	匹配一个换行符
\t	匹配一个制表符
^	匹配一行字符串的开头
$	匹配一行字符串的结尾
.	匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符
[...]	用来表示一组字符，单独列出，比如[amk]匹配a、m或k
[^...]	不在[]中的字符，比如[^abc]匹配除了a、b、c之外的字符
*	匹配0个或多个表达式
+	匹配1个或多个表达式
?	匹配0个或1个前面的正则表达式定义的片段，非贪婪方式
{n}	精确匹配n个前面的表达式
{n, m}	匹配n到m次由前面正则表达式定义的片段，贪婪方式
a|b	匹配a或b
( )	匹配括号内的表达式，也表示一个组

正则表达式

- Python的re库提供了整个正则表达式的实现，利用这个库，可以在Python中使用正则表达式。

正则表达式

- match()
  - match()方法会尝试从字符串的起始位置匹配正则表达式，如果匹配，就返回匹配成功的结果；如果不匹配，就返回None。

```
import re

content = 'Hello 123 4567 World_This is a Regex Demo'
print(len(content))
result = re.match('^Hello\s\d\d\d\s\d{4}\s\w{10}', content)
print(result)
print(result.group())
print(result.span())
```

正则表达式

- 匹配目标
  - 使用()括号将想提取的子字符串括起来。()实际上标记了一个子表达式的开始和结束位置，被标记的每个子表达式会依次对应每一个分组，调用group()方法传入分组的索引即可获取提取的结果。

```
import re

content = 'Hello 1234567 World_This is a Regex Demo'
result = re.match('^Hello\s(\d+)\sWorld', content)
print(result)
print(result.group())
print(result.group(1))
print(result.span())
```

正则表达式

- 通用匹配
  - .（点）可以匹配任意字符（除换行符），*（星）代表匹配前面的字符无限次，所以它们组合在一起就可以匹配任意字符。

```
import re

content = 'Hello 123 4567 World_This is a Regex Demo'
result = re.match('^Hello.*Demo$', content)
print(result)
print(result.group())
print(result.span())
```

正则表达式

- 非贪婪匹配
  - 在贪婪匹配下，.*会匹配尽可能多的字符。正则表达式中.*后面是\d+，也就是至少一个数字，并没有指定具体多少个数字，因此，.*就尽可能匹配多的字符。
  - 非贪婪匹配就是尽可能匹配少的字符，字符串中间尽量使用非贪婪匹配，也就是用.*?来代替.*，以免出现匹配结果缺失的情况。注意，如果匹配的结果在字符串结尾，.*?就有可能匹配不到任何内容了，因为它会匹配尽可能少的字符。

```
import re

content = 'http://weibo.com/comment/kEraCN'
result1 = re.match('http.*?comment/(.*?)', content)
result2 = re.match('http.*?comment/(.*)', content)
print('result1', result1.group(1))
print('result2', result2.group(1))
```

正则表达式

- 修饰符

修饰符	描述
re.I	使匹配对大小写不敏感
re.L	做本地化识别（locale-aware）匹配
re.M	多行匹配，影响^和$
re.S	使.匹配包括换行在内的所有字符
re.U	根据Unicode字符集解析字符。这个标志影响\w、\W、 \b和\B
re.X	该标志通过给予你更灵活的格式以便你将正则表达式写得更易于理解

```
result = re.match('^He.*?(\d+).*?Demo$', content, re.S)
```

正则表达式

- search()
  - match()方法是从字符串的开头开始匹配的，一旦开头不匹配，那么整个匹配就失败了。search()，它在匹配时会扫描整个字符串，然后返回第一个成功匹配的结果，如果搜索完了还没有找到，就返回None。

正则表达式

```
html = '''<div id="songs-list">
    <h2 class="title">经典老歌</h2>
    <p class="introduction">
        经典老歌列表
    </p>
    <ul id="list" class="list-group">
        <li data-view="2">一路上有你</li>
        <li data-view="7">
            <a href="/2.mp3" singer="任贤齐">沧海一声笑</a>
        </li>
        <li data-view="4" class="active">
            <a href="/3.mp3" singer="齐秦">往事随风</a>
        </li>
        <li data-view="6"><a href="/4.mp3" singer="beyond">光辉岁月</a></li>
        <li data-view="5"><a href="/5.mp3" singer="陈慧琳">记事本</a></li>
        <li data-view="5">
            <a href="/6.mp3" singer="邓丽君"><i class="fa fa-user"></i>但愿人长久</a>
        </li>
    </ul>
</div>'''
```

正则表达式

```
result = re.search('<li.*?active.*?singer="(.*?)">(.*?)</a>', html, re.S)
if result:
    print(result.group(1), result.group(2))
```

正则表达式

- findall()
  - 搜索整个字符串，然后返回匹配正则表达式的所有内容。

正则表达式

- sub()
  - 除了使用正则表达式提取信息外，有时候还需要借助它来修改文本。

```
html = re.sub('<a.*?>|</a>', '', html)
print(html)
results = re.findall('<li.*?>(.*?)</li>', html, re.S)
for result in results:
    print(result.strip())
```

3

Beautiful Soup

概述

- Beautiful Soup提供一些简单的、Python式的函数来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。
- Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为UTF-8编码。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时你仅仅需要说明一下原始编码方式就可以了。
- Beautiful Soup已成为和lxml、html6lib一样出色的Python解释器，为用户灵活地提供不同的解析策略或强劲的速度。

安装

```
pip3 install beautifulsoup4
```

解析器

解析器	使用方法	优势	劣势
Python标准库	BeautifulSoup(markup, "html.parser")	Python的内置标准库、执行速度适中、文档容错能力强	Python 2.7.3及Python 3.2.2之前的版本文档容错能力差
lxml HTML解析器	BeautifulSoup(markup, "lxml")	速度快、文档容错能力强	需要安装C语言库
lxml XML解析器	BeautifulSoup(markup, "xml")	速度快、唯一支持XML的解析器	需要安装C语言库
html5lib	BeautifulSoup(markup, "html5lib")	最好的容错性、以浏览器的方式解析文档、生成HTML5格式的文档	速度慢、不依赖外部扩展

- 通过以上对比可以看出，lxml解析器有解析HTML和XML的功能，而且速度快，容错能力强，所以推荐使用它。

解析器

```
from bs4 import BeautifulSoup
soup = BeautifulSoup('<p>Hello</p>', 'lxml')
print(soup.p.string)
```

基本用法

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.prettify())
print(soup.title.string)
```

节点选择器

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.title)
print(type(soup.title))
print(soup.title.string)
print(soup.head)
print(soup.p)
```

提取信息

- (1)获取名称
  - 可以利用name属性获取节点的名称。这里还是以上面的文本为例，选取title节点，然后调用name属性就可以得到节点名称：

```
print(soup.title.name)
```

提取信息

- (2)获取属性
  - 每个节点可能有多个属性，比如id和class等，选择这个节点元素后，可以调用attrs获取所有属性：

```
print(soup.p.attrs)
print(soup.p.attrs['name’])

print(soup.p['name'])
print(soup.p['class'])
```

提取信息

- (3)获取内容
  - 可以利用string属性获取节点元素包含的文本内容，比如要获取第一个p节点的文本：

```
print(soup.p.string)
```

提取信息

- (4)嵌套选择

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.head.title)
print(type(soup.head.title))
print(soup.head.title.string)
```

提取信息

- (5)关联选择
  - contents属性得到的结果是直接子节点的列表
  - 要得到所有的子孙节点的话，可以调用descendants属性
  - 要获取某个节点元素的父节点，可以调用parent属性
  - next_sibling和previous_sibling分别获取节点的下一个和上一个兄弟元素，next_siblings和previous_siblings则分别返回所有前面和后面的兄弟节点的生成器

方法选择器

- find_all()

```
find_all(name , attrs , recursive , text , **kwargs)
```

  - (1)name：可以根据节点名来查询元素
  - (2)attrs：除了根据节点名查询，我们也可以传入一些属性来查询
  - (3)text：text参数可用来匹配节点的文本，传入的形式可以是字符串，可以是正则表达式对象

方法选择器

- 其它方法
  - find()：返回第一个匹配的元素
  - find_parents()和find_parent()：前者返回所有祖先节点，后者返回直接父节点。
  - find_next_siblings()和find_next_sibling()：前者返回后面所有的兄弟节点，后者返回后面第一个兄弟节点。
  - find_previous_siblings()和find_previous_sibling()：前者返回前面所有的兄弟节点，后者返回前面第一个兄弟节点。
  - find_all_next()和find_next()：前者返回节点后所有符合条件的节点，后者返回第一个符合条件的节点。
  - find_all_previous()和find_previous()：前者返回节点后所有符合条件的节点，后者返回第一个符合条件的节点。

CSS选择器

- 使用CSS选择器时，只需要调用select()方法，传入相应的CSS选择器即可

本章小结

- urllib基本库
- 正则表达式
- Beautiful Soup

华信培训