移动网大数据挖掘应用技术能力培训
第一期
数据分析与挖掘 ①

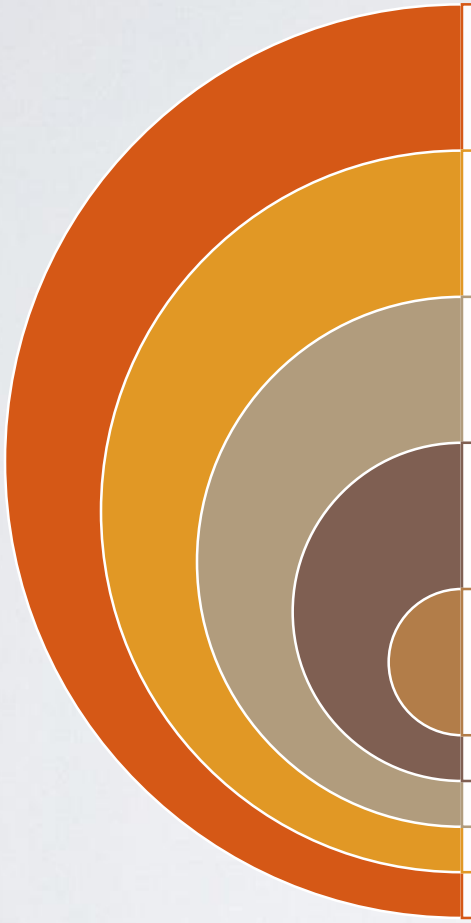
中通服网络信息技术有限公司
China Comservice NetIT Technology Co.,Ltd.

Python爬虫

Python爬虫

- Python基础
- Python requests库
- Python urllib和urllib2库
- 正则表达式
- Python BeautifulSoup库
- Python爬虫框架Scrapy

爬虫的常规流程

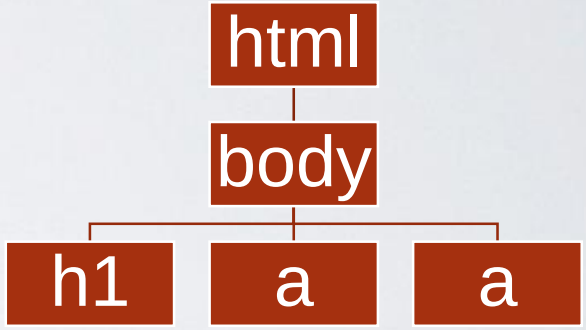


开启循环及多线程	• 对多页内容遍历，开启多线程提高爬取速度
保存至文件	• 通过文件形式将爬虫结果保存
页面元素分析	• 根据HTML代码分析得到需要获取的数据的对应结构
抓取单页内容	• 利用Requests请求目标站点，得到单个网页的HTML代码，返回结果
目标页面分析	• 利用开发者工具浏览页面的HTML源文件，对HTML结构有基本的认识

HTML基础

- 在开始解析HTML之前，我们先来复习一下HTML的基础知识：

```
<html>
        <body>
                <h1 id=“title”>Hello World</h1>
                <a href=“#” class=“link”>This is link</a>
                <a href=“#link1” class=“link”>This is link1</a>
        </body>
</html>
```



Requests Module

- Requests 是 Python 界大名鼎鼎的一个网络库, 其设计哲学是为人类而设计, 所以他提供的功能都非常的人性化。他的优点主要有两大点：
- 对 GET 和 POST 方法的封装做的很好, 自动处理了编码等问题；
- 默认开启了 Cookies 处理, 在处理需要登录的问题上面非常方便。
- Requests 的方便之处不止这两点, 还提供了诸如标准登录接口之类的功能, 我们暂时用不上。

Requests

- requests库提供了http所有的基本请求方式。例如：

```
r = requests.post("http://httpbin.org/post")
r = requests.put("http://httpbin.org/put")
r = requests.delete("http://httpbin.org/delete")
r = requests.head("http://httpbin.org/get")
r = requests.options("http://httpbin.org/get")
```

Requests

- Requests的Get方法传递参数

```
>>> payload = {'key1': 'value1', 'key2': 'value2', 'key3': None}
>>> r = requests.get('http://httpbin.org/get', params=payload)
```

- 参数也可以传递列表

```
>>> payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

>>> r = requests.get('http://httpbin.org/get', params=payload)
```

- 传递header参数

```
>>> headers = {'user-agent': 'my-app/0.0.1'}
>>> r = requests.get(url, headers=headers)
```

- 传递cookies

```
>>> url = 'http://httpbin.org/cookies'

>>> r = requests.get(url, cookies=dict(cookies_are='working'))
>>> r.text
'{"cookies": {"cookies_are": "working"}}'
```

参考文档： http://www.python-requests.org/en/master/

小试牛刀

• 我们先来看看使用requests如何获得一个页面的HTML数据：

```
import requests

response = requests.get("http://www.baidu.com/")
print(response.text)
print(response.encoding)
response.encoding = 'utf-8'
print(response.text)
```

BeautifulSoup Module

- BeautifulSoup 大大方便了我们对抓取的 HTML 数据的解析, 可以用tag, class, id来定位我们想要的东西, 可以直接提取出正文信息, 可以全文搜索, 同样也支持正则表达式, 相当给力。

找出所有含有特定标签的HTML元素

- 使用select 找出含有h1标签的元素

```
Soup = BeautifulSoup(html_sample)
Header = soup.select(‘h1’)
Print(header)
```

select 预设返回的是符合条件的一个list

取得含有特定属性的元素

• 使用select 找出所有id 为title的元素（id前面加#）

```
alink = soup.select(‘#title’)
print(alink)
```

• 使用select 找出所有class 为link 的元素（class前面加.）

```
Alink = soup.select(‘.link’)
For link in alink:
        print(link)
```

BeautifulSoup

- 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:

- Tag
- NavigableString
- BeautifulSoup
- Comment

BeautifulSoup-Tag

- Tag 是什么？通俗点讲就是 HTML 中的一个个标签，例如：
  <title>The Dormouse‘s story</title>中的title
   找服务 中的a
- 对于 Tag，它有两个重要的属性，是 name 和 attrs，

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

输出结果：

```
title
{'href': '//login.taobao.com/member/login.jhtml?f=top&redirectURL=https%3A%2F%2Fwww.taobao.com%2F', 'target': '_top'}
```

BeautifulSoup-Tag

- Tag 的属性 还可以进行操作，如：

```
print(soup.a['href'])
```

```
//login.taobao.com/member/login.jhtml?f=top&redirectURL=https%3A%2F%2Fwww.taobao.com%2F
```

BeautifulSoup-NavigableString

- 既然我们已经得到了标签的内容，那么问题来了，我们要想获取标签内部的文字怎么办呢？很简单，用 .string 即可，例如：

print(soup.title.string)

输出结果：

淘宝网 - 淘！我喜欢

BeautifulSoup-BeautifulSoup

- BeautifulSoup 对象表示的是一个文档的全部内容.大部分时候,可以把它当作 Tag 对象，是一个特殊的 Tag，我们可以分别获取它的类型，名称，以及属性来感受一下：

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

```
[document]
```

BeautifulSoup-Comment

- Comment 对象是一个特殊类型的 NavigableString 对象，其实输出的内容仍然不包括注释符号，但是如果不好好处理它，可能会对我们的文本处理造成意想不到的麻烦。

```
<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>
 Elsie
<class 'bs4.element.Comment'>
```

- a 标签里的内容实际上是注释，但是如果我们利用 .string 来输出它的内容，我们发现它已经把注释符号去掉了，所以这可能会给我们带来不必要的麻烦。

BeautifulSoup-遍历文档树

- **（1）直接子节点**
- **.contents**
- tag 的 .content 属性可以将tag的子节点以列表的方式输出
- **.children**
- 它返回的不是一个 list，不过我们可以通过遍历获取所有子节点。

BeautifulSoup-遍历文档树

- **（2）所有子孙节点**
- .descendants
- .contents 和 .children 属性仅包含tag的直接子节点，.descendants 属性可以对所有tag的子孙节点进行递归循环，和 children类似，我们也需要遍历获取其中的内容。

文件输出

- 常用函数 ： with open (url,mode,encoding) as
  - 第一个参数是文件名称，包括路径；
  - 第二个参数是打开的模式mode;
  - 第三个参数是文件的编码格式；
- 打开模式：
  - 'r'：只读（缺省。如果文件不存在，则抛出错误）
  - 'w'：只写（如果文件不存在，则自动创建文件）
  - 'a'：附加到文件末尾
  - 'r+'：读写

Python爬虫实战

Requests+BeautifulSoup豆瓣电影Top250

```
for item in soup.findAll("div",class_="item"):
    em=item.select('em')[0].text
    a=item.select('a')[0]['href']
    img=item.select('img')[0]['src']
    title=item.select('.title')[0].text
    if len(item.select('.title'))>1:
        title1=item.select('.title')[1].text
    titleother = item.select('.other')[0].text
    actor = item.select('p')[0].text.replace(' ','')
    score =item.select('.rating_num')[0].text
    quote = item.select('.quote')[0].text
    print(em,a,img,title,title1,titleother,actor,score,quote)
```

Beautiful使用方法简介

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: Tag , NavigableString , BeautifulSoup , Comment .这里先介绍tag对象。

操作文档树最简单的方法就是告诉它你想获取的tag的name.如果想获取 <head> 标签,只要用 soup.head :

```
soup.head
# <head><title>The Dormouse's story</title></head>

soup.title
# <title>The Dormouse's story</title>
```

```
soup.body.b
# <b>The Dormouse's story</b>
```

通过点取属性的方式只能获得当前名字的第一个tag,如果想要得到所有的<a>标签,
或是通过名字得到比一个tag更多的内容的时候,就需要用到 *Searching the tree* 中描述的方法,比如: find_all()

```
soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful使用方法简介

tag的 .contents 属性可以将tag的子节点以列表的方式输出:

```
head_tag = soup.head
head_tag
# <head><title>The Dormouse's story</title></head>

head_tag.contents
[<title>The Dormouse's story</title>]

title_tag = head_tag.contents[0]
title_tag
# <title>The Dormouse's story</title>
title_tag.contents
# [u'The Dormouse's story']
```

通过tag的 .children 生成器,可以对tag的子节点进行循环:

```
for child in title_tag.children:
    print(child)
    # The Dormouse's story
```

Beautiful使用方法简介

Beautiful Soup定义了很多搜索方法,这里着重介绍2个: find() 和 find_all() .其它方法的参数和用法类似。
介绍 find_all() 方法前,先介绍一下过滤器的类型 ,这些过滤器贯穿整个搜索的API。
过滤器可以被用在tag的name中,节点的属性中,字符串中或他们的混合中.

最简单的过滤器是字符串.在搜索方法中传入一个字符串参数,Beautiful Soup会查找与字符串完整匹配的内容,下面的例子用于查找文档中所有的<b>标签:

```
soup.find_all('b')
# [<b>The Dormouse's story</b>]
```

如果传入正则表达式作为参数,Beautiful Soup会通过正则表达式的 match() 来匹配内容。
下面例子中找出所有以b开头的标签,这表示<body>和<b>标签都应该被找到:

```
import re
for tag in soup.find_all(re.compile("^b")):
    print(tag.name)
# body
# b
```

Beautiful使用方法简介

如果没有合适过滤器,那么还可以定义一个方法,方法只接受一个元素参数 [4] ,如果这个方法返回 True 表示当前元素匹配并且被找到,如果不是则反回 False

下面方法校验了当前元素,如果包含 class 属性却不包含 id 属性,那么将返回 True:

```
def has_class_but_no_id(tag):
    return tag.has_attr('class') and not tag.has_attr('id')
```

```
soup.find_all(has_class_but_no_id)
# [<p class="title"><b>The Dormouse's story</b></p>,
#  <p class="story">Once upon a time there were...</p>,
#  <p class="story">...</p>]
```

将这个方法作为参数传入 find_all() 方法,将得到所有<p>标签:

Beautiful使用方法简介

如果一个指定名字的参数不是搜索内置的参数名,搜索时会把该参数当作指定名字tag的属性来搜索,如果包含一个名字为 id 的参数,Beautiful Soup会搜索每个tag的”id”属性

```
soup.find_all(id='link2')
# [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

如果传入 href 参数,Beautiful Soup会搜索每个tag的”href”属性

```
soup.find_all(href=re.compile("elsie"))
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>]
```

使用多个指定名字的参数可以同时过滤tag的多个属性

```
soup.find_all(href=re.compile("elsie"), id='link1')
# [<a class="sister" href="http://example.com/elsie" id="link1">three</a>]
```

在这里我们想用 class 过滤，不过 class 是 python 的关键词，这怎么办？加个下划线就可以

```
soup.find_all("a", class_="sister")
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful使用方法简介

通过 text 参数可以搜搜文档中的字符串内容.与 name 参数的可选值一样, text 参数接受 字符串 , 正则表达式 , 列表, True

```
soup.find_all(text="Elsie")
# [u'Elsie']

soup.find_all(text=["Tillie", "Elsie", "Lacie"])
# [u'Elsie', u'Lacie', u'Tillie']

soup.find_all(text=re.compile("Dormouse"))
[u"The Dormouse's story", u"The Dormouse's story"]
```

find()使用方法与 find_all() 方法完全相同，唯一的区别是 find_all() 方法的返回结果是值包含一个元素的列表，而 find() 方法直接返回结果

Beautiful使用方法简介

CSS选择器

我们在写 CSS 时，标签名不加任何修饰，类名前加点，id名前加 #，在这里我们也可以利用类似的方法来筛选元素，用到的方法是 soup.select()，返回类型是 list

通过标签名查找

```
print soup.select('title')
#[<title>The Dormouse's story</title>]
```

通过类名查找

```
print soup.select('.sister')
#[<a class="sister" href="http://example.com/elsie"
```

通过 id 名查找

```
print soup.select('#link1')
#[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>]
```

更多用法请参考：
https://www.crummy.com/software/BeautifulSoup/bs4/doc/index.zh.html