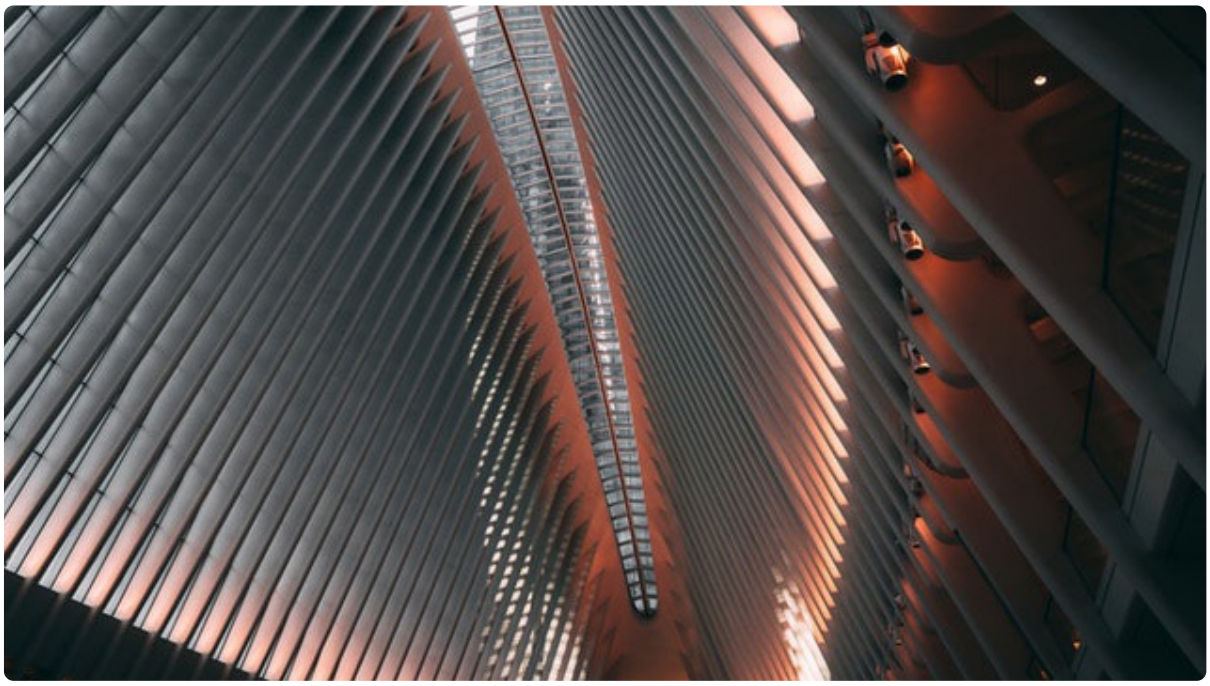
04 动手开发最简单的单文件爬虫

更新时间：2019-07-03 19:11:49



> 理想必须要人们去实现它，它不但需要决心和勇敢而且需要知识。
>
> ——吴玉章

由实践入手通过代码说话，学习如何由一个想法开始对爬虫进行“简单设计”，以及了解开发网络爬虫要分为多少个基本的实施步骤。

设计思路

如果你已认真阅读前面两个小节的内容，那么恭喜你！你已经具备动手编写网络爬虫的基础知识了。接下来的这一个小节，就是将我们前面所打下的基础，通过一个具体的动手实践将其融汇贯通形成一个真正的网络爬虫。

开始之前我们需要确定一个爬取的目标，为了能保证这个例子能持久地运行我特意采用我的博客作为本例的爬取目标。

目标: 在这个示例里面我们要写一个爬虫将我的博客中的文章列表拉出下来，保存到一个 JSON文件里面。

> 注: 网络爬虫项目的关键在于从一开始就要清楚地建立一个明确的爬取方向与目的。

开始设计数据结构

建立具体的爬取目标之后并不是急于动手去编码，而是应该弄清楚要从网页中**取**些什么，然后**存**什么，换句话说就是要设计爬取后的数据的存储结构。

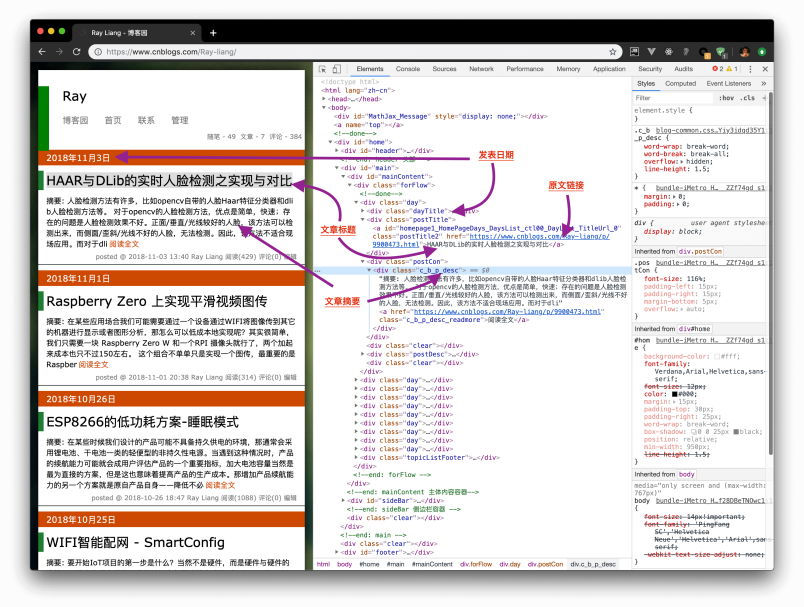
如上图所示，每个文章都是以相同的模式进行显示的，这就很容易得到这么一个简单的结构：

名称	字段
标题	title
摘要	summary
发表日期	pub_date
原文链接	parmerlink

“种子”的分析，生成爬虫入口

网络爬虫中爬取的第一个页面称之为“种子”页(seed)，又叫爬虫入口。在本例中目标数据就在当前打开的页面 https://www.cnblogs.com/Ray-liang/ 内，而对于一些项目来说可能数据是存在于其它不知道具体地址的页面内，而要得到这些具体的URL则需要先爬取“种子”页后方能获取，这就是“种子”来由。

现在我们需要将上面设计的数据结构与网页中的元素对应起来，打开浏览器的开发者工具来分析一下网页的内容:

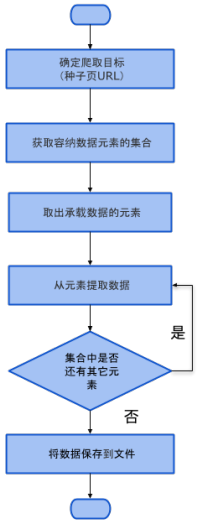


此时我们可以再扩充一下上文中的数据结构表，将网页中的元素选择器写到表内:

名称	字段	选择器
标题	title	.postTitle>a
摘要	summary	.postCon
发表日期	pub_date	.dayTitle
原文链接	parmerlink	.postTitle>a

现在对网页内容的分析已经完成，接下来只要将的所有的文章内容都找出来，然后用一个循环就可以将数据提取出来了。

下图是对整个爬虫编程思路的整理：



开始编码

现在已万事俱备，已经可以开始编写代码了。

```
$ mkdir blog-crawler
$ cd blog-crawler
```

用 virtualenv 创建python虚环境:

```
blog-cawler $ virtualenv -p python3 venv
blog-cawler $ . venv/bin/activate
```

安装基本依赖:

```
(venv) blog-cawler $ pip install pyQuery
```

创建入口文件:

```
(venv) blog-cawler $ touch cnblog-crawler.py
```

导入必要的依赖包:

```
from urllib import request
from pyquery import PyQuery as pq
import json
```

生成种子页的HTTP请求:

```
url = 'https://www.cnblogs.com/Ray-liang/'
with request.urlopen(url) as response:
    body = response.read()
    items = [] #1. 先定义一个空数组，用于储存提取结果
    with open('output.json', 'wt') as f: # 2 将结果写入JSON文件
        f.write(json.dumps(items))
```

上述代码中的 items 数组是一个空的对象，这是为了先将主线的思路实现，最后来完成单个元素提供的代码。

“#2”则采用了python内置的IO处理方法 open 来打开一个文件，第一个参数是将数据写入到哪一个文件，第二个参数是打开这个文件时所采用的方法， 'w' 是指写入。

```
with open('output.json', 'w') as f:
    f.write(json.dumps(items))
```

然后， json.dumps 方法会将 items 直接序列化成一个标准的JSON字符串，最后将这个JSON字符串通过调用 file 对象的 write 方法写入到文件内。

而 with 语句是用于指定 f 的作用域，当 f.write 调用完成跳出 with 子句时就会被关闭与销毁，这样可以防止打开文件后忘记调用 close 而锁住文件，导致其它的进程不能访问。

现在，整个主线的代码流程已经完成了，剩下的就是如何来生成这个 items 中的对象数据了。

首先，我们需要将 body 中的内容读到pyQuery中，然后选出所有的文章元素，最后通过循环逐个元素来处理，将元素的值生成为一个数据项填充到 items 中。

根据我们的分析，文章列表的元素选择器为 .forFlow>.day， 而这个选择器一但执行会返回多个元素的集合，而且我们需要将一个元素集合转化为一个 item 类型的集合，所以我们可以使用 pyQuery.map 函数完成这一转换。

那么上述的代码就会变为这样：

```
from urllib import request
from pyquery import PyQuery as pq
import json

url = 'https://www.cnblogs.com/Ray-liang/'

def parse_item(i, e):
    pass

with request.urlopen(url) as response:
    body = response.read()
    doc = pq(body)
    items = doc('.forFlow>.day').map(parse_item)
    with open('output.json', 'wt',encoding="utf-8") as f:
        f.write(json.dumps(items))
```

这里还要详细解释一下这个 map 函数， map 函数是一个高阶函数，它的参数是另一个处理函数的指针，所以这里引用了一个 parse_item ，这个函数的内部实际上是一个循环，它会将 doc('.forFlow>.day') 一个一个传入到 parse_item 函数中，当循环执行结束后再将多次从 parse_item 获取的结果合成为一个数组返回。

通过 map 这么一个转换就将处理集合的问题变成了处理单个元素的问题了，接下来就是实现 parse_item 函数了。

根据前文我们在分析设计时得到的元素映射表的关系，我们就可以直接编写这个 parse_item 函数了，具体如下：

```
def parse_item(i, e):
    doc = pq(e)
    title = doc('.postTitle>a').text()
    parmerlink = doc('.postTitle>a').attr('href')
    pub_date = doc('.dayTitle').text()
    summary = doc('.postCon').text()
    result = {
        'title': title,
        'parmerlink': parmerlink,
        'pub_date': pub_date,
        'summary': summary
    }
    print(json.dumps(result))
    return result
```

大功告成！接下来就是执行这个爬虫了，在命令行这样执行就OK了：

```
(venv) blog-cawler $ python3 cnblog-crawler.py
```

完成后就会发现在当前爬虫工作目录中会多了一个名为 output.json 的文件，打开它后的样子是这样的：

```
[
    {
        "title": "HAAR\u4e0eDLib\u7684\u5b9e\u65f6\u4eba\u8138\u68c0\u6d4b\u4e4b\u5b9e\u73b0\u4e0e\u5bf9\u6bd4",
        "url": "https://www.cnblogs.com/Ray-liang/p/9900473.html",
        "pub_date": "2018\u5e7411\u67083\u65e5",
        "summary": "\u6458\u8981: \u4eba\u8138\u68c0\u6d4b\u65b9\u6cd5\u6709\u8bb8\u591a\uff0c\u6bd4\u5982opencv\u81ea\u5e26\u7684\u4eba\u8138Haar\u7279\u5f81\u5206\u7c7b\u5668\u548cdlib\u4eba\u8138\u68c0\u6d4b\u65b9\u6cd5\u7b49\u3002 \u5bf9\u4e8eopencv\u7684\u4eba\u8138\u68c0\u6d4b\u65b9\u6cd5\uff0c\u4f18\u70b9\u662f\u7b80\u5355\uff0c\u5feb\u901f\uff1b\u5b58\u5728\u95ee\u9898\u662f\u4eba\u8138\u68c0\u6d4b\u6548\u679c\u4e0d\u597d\u3002\u6b63\u9762/\u5782\u76f4/\u5149\u7ebf\u8f83\u597d\u7684\u4eba\u8138\uff0c\u8be5\u65b9\u6cd5\u53ef\u4ee5\u68c0\u6d4b\u51fa\u6765\uff0c\u800c\u4fa7\u9762/\u6b6a\u659c/\u5149\u7ebf\u4e0d\u597d\u7684\u4eba\u8138\uff0c\u65e0\u6cd5\u68c0\u6d4b\u3002\u56e0\u6b64\uff0c\u8be5\u65b9\u6cd5\u4e0d\u9002\u5408\u73b0\u573a\u5e94\u7528\u3002\u800c\u5bf9\u4e8edli\u9605\u8bfb\u5168\u6587"
    },
    ...
]
```

看到这样的结果你一定很崩溃吧，这到底是“什么鬼”？怎么好好的中文全乱了？导致这个现象的出现是由于 json.dumps 时对爬取结果的内容进行了 unicode 编码的结果，如果彻底解决这个问题只要在文件的第一行加入一个 __future__ 模块就可以了：

```
from __future__ import unicode_literals
```

然后将在`json.dumps`方法加上一个`ensure_ascii=False`的参数：

```
f.write(json.dumps(items,ensure_ascii=False))
```

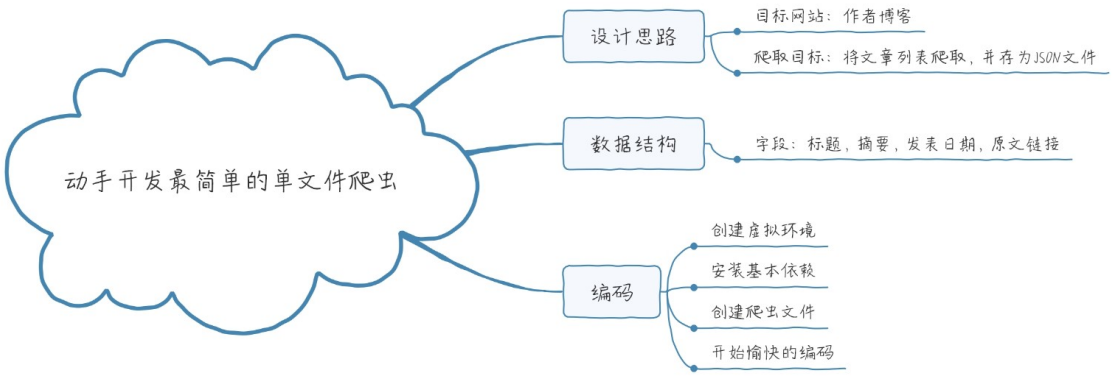
再执行一次输出结果就会正常显示了：

```
liangruikundeMacBook-Pro:src raymacbook$ python3 cnblog-cralwer.py
{"title": "HAAR与DLib的实时人脸检测之实现与对比", "parmerlink": "https://www.cnblogs.com/Ray-liang/p/9900473.html", "pub_date": "2018年11月3日", "summary": "摘要: 人脸检测方法有许多，比如opencv自带的人脸Haar特征分类器和dlib人脸检测方法等。 对于opencv的人脸检测方法，优点是简单，快速；存在的问题是人脸检测效果不好。正面/垂直/光线较好的人脸，该方法可以检测出来，而侧面/歪斜/光线不好的人脸，无法检测。因此，该方法不适合现场应用。而对于dli阅读全文"}
...
```

小结：爬虫的基本开发思路

最后我们先总结一下在本节中所学到的设计与开发一个完整爬虫的思路与过程：

1. 确立爬取目标，分析种子页结构
2. 设计需要存储的数据结构
3. 分析承载数据的页面结构，建立数据结构与元素选择器间的映射关系
4. 设计代码流程与编写思路



03 HTTP 协议通信原理与 HTML 基础入门

05 Python 世界最流行的网络爬虫框架 Scrapy