XPath

前言

前面我们介绍了 BeautifulSoup 的用法，这个已经是非常强大的库了，不过还有一些比较流行的解析库，例如 lxml，使用的是 Xpath 语法，同样是效率比较高的解析方法。如果大家对 BeautifulSoup 使用不太习惯的话，可以尝试下 Xpath。

参考来源

lxml用法源自 lxml python 官方文档，更多内容请直接参阅官方文档，本文对其进行翻译与整理。

lxml

XPath语法参考 w3school

w3school

视频资源

如果你对 XPath 不熟悉的话，可以看下这个视频资源：

web端功能自动化定位元素

安装

```
pip install lxml
```

利用 pip 安装即可

XPath语法

XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和

属性进行遍历。XPath 是 W3C XSLT 标准的主要元素，并且 XQuery 和 XPointer 都构建于 XPath 表达之上。

节点关系

（1）父（Parent）

每个元素以及属性都有一个父。

在下面的例子中，book 元素是 title、author、year 以及 price 元素的父：

```
<book>

  <title>Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>
```

（2）子（Children）

元素节点可有零个、一个或多个子。

在下面的例子中，title、author、year 以及 price 元素都是 book 元素的子：

```
<book>

  <title>Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>

```

（3）同胞（Sibling）

拥有相同的父的节点

在下面的例子中，title、author、year 以及 price 元素都是同胞：

```
<book>

  <title>Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>

```

（4）先辈（Ancestor）

某节点的父、父的父，等等。

在下面的例子中，title 元素的先辈是 book 元素和 bookstore 元素：

```
<bookstore>

<book>

  <title>Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>

</bookstore>

```

（5）后代（Descendant）

某个节点的子，子的子，等等。

在下面的例子中，bookstore 的后代是 book、title、author、year 以及 price 元素：

```
<bookstore>

<book>

  <title>Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>

</bookstore>

```

选取节点

XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 step 来选取的。

下面列出了最有用的路径表达式：

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。

实例

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式	结果
bookstore	选取 bookstore 元素的所有子节点。
/bookstore	选取根元素 bookstore。注释：假如路径起始于正斜杠(/)，则此路径始终代表到某元素的绝对路径！
bookstore/book	选取属于 bookstore 的子元素的所有 book 元素。
//book	选取所有 book 子元素，而不管它们在文档中的位置。
bookstore//book	选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。
//@lang	选取名为 lang 的所有属性。

谓语（Predicates）

谓语用来查找某个特定的节点或者包含某个指定的值的节点。

谓语被嵌在方括号中。

实例

在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。

//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

选取未知节点

XPath 通配符可用来选取未知的 XML 元素。

通配符	描述
*	匹配任何元素节点。
@*	匹配任何属性节点。
node()	匹配任何类型的节点。

实例

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式	结果
/bookstore/*	选取 bookstore 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

选取若干路径

通过在路径表达式中使用“|”运算符，您可以选取若干个路径。

实例

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式	结果
//book/title | //book/price	选取 book 元素的所有 title 和 price 元素。
//title | //price	选取文档中的所有 title 和 price 元素。
/bookstore/book/title | //price	选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。

XPath 运算符

下面列出了可用在 XPath 表达式中的运算符：

运算符	描述	实例	返回值
	计算两个节点集	//book	//cd
+	加法	6 + 4	10
–	减法	6 – 4	2
*	乘法	6 * 4	24
div	除法	8 div 4	2
=	等于	price=9.80	如果 price 是 9.80，则返回 true。如果 price 是 9.90，则返回 false。
!=	不等于	price!=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
<	小于	price<9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
<=	小于或等于	price<=9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。

>	大于	price>9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
>=	大于或等于	price>=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.70，则返回 false。
or	或	price=9.80 or price=9.70	如果 price 是 9.80，则返回 true。如果 price 是 9.50，则返回 false。
and	与	price>9.00 and price<9.90	如果 price 是 9.80，则返回 true。如果 price 是 8.50，则返回 false。
mod	计算除法的余数	5 mod 2	1

lxml用法

初步使用

首先我们利用它来解析 HTML 代码，先来一个小例子来感受一下它的基本用法。

```
from lxml import etree

text = '''

<div>

    <ul>

         <li class="item-0"><a href="link1.html">first item</a><
/li>

         <li class="item-1"><a href="link2.html">second item</a>
</li>

         <li class="item-inactive"><a href="link3.html">third it
em</a></li>

```

```
         <li class="item-1"><a href="link4.html">fourth item</a>
</li>

         <li class="item-0"><a href="link5.html">fifth item</a>

     </ul>

 </div>

'''

html = etree.HTML(text)

result = etree.tostring(html)

print(result)
```

首先我们使用 lxml 的 etree 库，然后利用 etree.HTML 初始化，然后我们将其打印出来。

其中，这里体现了 lxml 的一个非常实用的功能就是自动修正 html 代码，大家应该注意到了，最后一个 li 标签，其实我把尾标签删掉了，是不闭合的。不过，lxml 因为继承了 libxml2 的特性，具有自动修正 HTML 代码的功能。

所以输出结果是这样的

```
<html><body>

<div>

    <ul>

         <li class="item-0"><a href="link1.html">first item</a><
/li>

         <li class="item-1"><a href="link2.html">second item</a>
</li>

         <li class="item-inactive"><a href="link3.html">third it
em</a></li>

         <li class="item-1"><a href="link4.html">fourth item</a>
</li>
```

```
18
19          <li class="item-0"><a href="link5.html">fifth item</a><
20  /li>
21
22  </ul>

     </div>

    </body></html>
```

不仅补全了 li 标签，还添加了 body，html 标签。

文件读取

除了直接读取字符串，还支持从文件读取内容。比如我们新建一个文件叫做 hello.html，内容为

```
1   <div>
2
3       <ul>
4
5           <li class="item-0"><a href="link1.html">first item</a><
6   /li>
7
8           <li class="item-1"><a href="link2.html">second item</a>
9   </li>
10
11          <li class="item-inactive"><a href="link3.html"><span cl
12  ass="bold">third item</span></a></li>
13
14          <li class="item-1"><a href="link4.html">fourth item</a>
15  </li>
16
17          <li class="item-0"><a href="link5.html">fifth item</a><
18  /li>

         </ul>

     </div>
```

利用 parse 方法来读取文件。

```
from lxml import etree

html = etree.parse('hello.html')

result = etree.tostring(html, pretty_print=True)

print(result)

```

同样可以得到相同的结果。

XPath实例测试

依然以上一段程序为例

（1）获取所有的

- 标签

```
from lxml import etree

html = etree.parse('hello.html')

print type(html)

result = html.xpath('//li')

print result

print len(result)

print type(result)

print type(result[0])

```

运行结果

```
<type 'lxml.etree._ElementTree'>
[<Element li at 0x1014e0e18>, <Element li at 0x1014e0ef0>, <Elem
ent li at 0x1014e0f38>, <Element li at 0x1014e0f80>, <Element li
 at 0x1014e0fc8>]
```

```
5
<type 'list'>
<type 'lxml.etree._Element'>
```

可见，etree.parse 的类型是 ElementTree，通过调用 xpath 以后，得到了一个列表，包含了 5 个

- 元素，每个元素都是 Element 类型

（2）获取

- 标签的所有 class

```
 result = html.xpath('//li/@class')

print result

```

运行结果

```
['item-0', 'item-1', 'item-inactive', 'item-1', 'item-0']

```

（3）获取

- 标签下 href 为 link1.html 的 <a> 标签

```
result = html.xpath('//li/a[@href="link1.html"]')

print result

```

运行结果

```
[<Element a at 0x10ffaae18>]

```

（4）获取 <li> 标签下的所有 标签

注意这么写是不对的

```
result = html.xpath('//li/span')
```

因为 / 是用来获取子元素的，而 并不是 <li> 的子元素，所以，要用双斜杠

```
result = html.xpath('//li//span')
print result
```

运行结果

```
[<Element span at 0x10d698e18>]
```

（5）获取 <li> 标签下的所有 class，不包括

```
result = html.xpath('//li/a//@class')

print result

```

运行结果

```
['blod']

```

（6）获取最后一个 <li> 的 <a> 的 href

```
result = html.xpath('//li[last()]/a/@href')

print result

```

运行结果

```
['link5.html']
```

（7）获取倒数第二个元素的内容

```
result = html.xpath('//li[last()-1]/a')

print result[0].text

```

运行结果

```
fourth item

```

（8）获取 class 为 bold 的标签名

```
result = html.xpath('//*[@class="bold"]')

print result[0].tag

```

运行结果

```
span
```

通过以上实例的练习，相信大家对 XPath 的基本用法有了基本的了解。也可以利用 text 方法来获取元素的内容。

大家多加练习！

结语

XPath 是一个非常好用的解析方法，同时也作为爬虫学习的基础，在后面的 selenium 以及 scrapy 框架中都会涉及到这部分知识，希望大家可以把它的语法掌握清楚，为后面的深入研究做好铺垫。