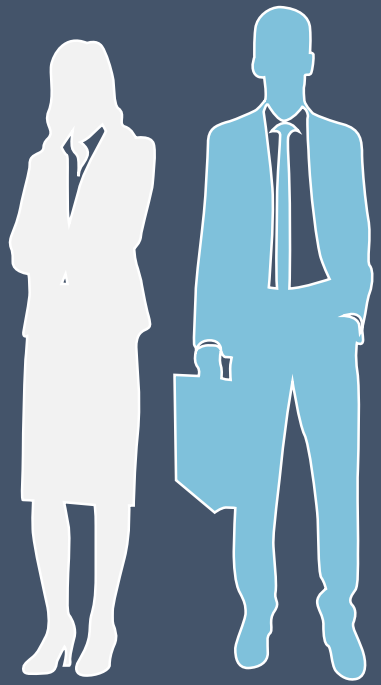
AQF Analyst of Quantitative Finance



使用Xpath进行网页解析

朱映秋

CONTENTS

Xpath概览

- XPath，全称XML Path Language，即XML路径语言，它是一门在XML文档中查找信息的语言。它最初是用来搜寻XML文档的，但是它同样适用于HTML文档的搜索。
- XPath的选择功能十分强大，它提供了非常简洁明了的路径选择表达式。
- XPath提供了超过100个内建函数，用于字符串、数值、时间的匹配以及节点、序列的处理等。几乎所有我们想要定位的节点，都可以用XPath来选择。
- XPath于1999年11月16日成为W3C标准，它被设计为供XSLT、XPointer以及其他XML解析软件使用，更多的文档可以访问其官方网站：https://www.w3.org/TR/xpath/。

lxml库

- **lxml库**
  - 在做爬虫时，我们完全可以使用XPath来做相应的信息抽取。即通过Python的lxml库，利用XPath进行HTML的解析，所以要确保安装好lxml库。
  - lxml是Python的一个解析库，支持HTML和XML的解析，支持XPath解析方式，而且解析效率非常高。
- **相关链接**
  - 官方网站：http://lxml.de
  - GitHub：https://github.com/lxml/lxml
  - PyPI：https://pypi.python.org/pypi/lxml

lxml库安装

- pip安装
  - 直接使用pip安装，执行命令为：pip install lxml

- 验证安装
  - 安装完成之后，可以在Python命令行下测试：import lxml

- 如果没有错误报出，则证明库已经安装好了。

```
1 import lxml
```

CONTENTS

HTML节点关系

- 父
  - 每个元素以及属性都有一个父。
  - book 元素是 title、price元素的父
- 子
  - 元素节点可有零个、一个或多个子。
  - title、 price 元素都是 book 元素的子。
- 同胞
  - 拥有相同的父的节点。
  - title、 price 元素都是同胞。
- 先辈
  - 某节点的父、父的父，等等。
  - title 元素的先辈是 book 元素和 bookstore 元素。
- 后代
  - 某个节点的子，子的子，等等。
  - bookstore 的后代是 book、title、 price 元素。

```
<bookstore>
    <book>
        <title lang="eng">Harry Potter</title>
        <price>29.99</price>
    </book>

    <book>
        <title lang="eng">Learning Python</title>
        <price>39.95</price>
    </book>
</bookstore>
```

Xpath语法

- 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。
  - 选取节点
  - XPath常用规则

表达式	描述
nodename	选取此节点的所有子节点
/	从当前节点选取直接子节点
//	从当前节点选取子孙节点
.	选取当前节点
..	选取当前节点的父节点
@	选取属性

- 以上面的例子，来看一些实例：

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。
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。

- 注：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。

未知节点选择

- 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。

- 以上面的例子，来看一些实例的路径表达式，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。

谓语

- 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。 谓语被嵌入在方括号中。

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。

选取若干路径

- 通过在路径表达式中使用“|”运算符，可以选取若干个路径。

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。

CONTENTS

AQF Analyst of Quantitative Finance

- 下面是网页的内容：
  - 首先导入lxml库的etree模块，然后声明了一段HTML文本，调用HTML类进行初始化，这样就成功构造了一个 XPath 解析对象。
  - 这里需要注意的是，HTML文本并不完整，其最后一个li节点是没有闭合的。但是etree模块可以自动修正HTML文本（lxml 具有自动修正 HTML 代码的功能）。
  - 调用tostring()方法即可输出修正后的HTML代码，但是结果是bytes类型。这里利用decode()方法将其转成str类型。
  - 经过处理之后，li节点标签被补全，并且还自动添加了body、html节点。

```
from lxml import etree
text = """
<div>
    <ul>
        <li class="item-0">
            <a href="link1.html">1st item</a>
        </li>
        <li class="item-1">
            <a href="link2.html">2nd item</a>
        </li>
        <li class="item-inactive">
            <a href="link3.html">3rd item</a>
        </li>
        <li class="item-1">
            <a href="link5.html">4th item</a>
        </li>
    </ul>
</div>
"""

```

```
html = etree.HTML(text)
result = etree.tostring(html)
print(result.decode('utf-8'))
```

```
<html><body><div>
    <ul>
        <li class="item-0">
            <a href="link1.html">1st item</a>
        </li>
        <li class="item-1">
            <a href="link2.html">2nd item</a>
        </li>
        <li class="item-inactive">
            <a href="link3.html">3rd item</a>
        </li>
        <li class="item-1">
            <a href="link5.html">4th item</a>
        </li>
    </ul>
</div>
</body></html>
```

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 test.html，内容为上面text的网页内容。
- 可以发现，这次的输出结果略有不同，多了一个DOCTYPE的声明，不过对解析无任何影响。

- 1st item
- 2nd item
- 3rd item
- 4th item

```
1 html = etree.parse('test.html', etree.HTMLParser())
2 result = etree.tostring(html)
3 print(result.decode('utf-8'))
```

```
DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
tml>
  <body><div>
  <ul>
      <li class="item-0">
          <a href="link1.html">1st item</a>
      </li>
      <li class="item-1">
          <a href="link2.html">2nd item</a>
      </li>
      <li class="item-inactive">
          <a href="link3.html">3rd item</a>
      </li>
      <li class="item-1">
          <a href="link5.html">4th item</a>
      </li>
  </ul>
</div>
</body>
</html>
```

CONTENTS

Xpath解析

XPath解析	匹配所有节点
	匹配子节点
	匹配父节点
	匹配属性
	匹配文本
	属性获取
	属性多值匹配
	多属性匹配
	按序选择

```
from lxml import etree
text = """
<div>
    <ul>
        <li class="item-0">
            <a href="link1.html">1st item</a>
        </li>
        <li class="item-1">
            <a href="link2.html">2nd item</a>
        </li>
        <li class="item-inactive">
            <a href="link3.html">3rd item</a>
        </li>
        <li class="item-1">
            <a href="link5.html">4th item</a>
        </li>
    </ul>
</div>
"""
```

匹配所有节点

```
nodes = html.xpath('//*')
print(nodes)
```

```
[<Element html at 0x7f9a59663fa0>, <Element body at 0x7f9a58436320>, <Element div at 0x7f9a584363c0>, <Element ul at 0x7f9a58436410>, <Element li at 0x7f9a58436460>, <Element a at 0x7f9a584364b0>, <Element li at 0x7f9a58436500>, <Element a at 0x7f9a58436550>, <Element li at 0x7f9a584365a0>, <Element a at 0x7f9a584365f0>, <Element li at 0x7f9a58436640>, <Element a at 0x7f9a58436690>]
```

```
nodes_li = html.xpath('//li')
print(nodes_li)
```

```
[<Element li at 0x7f9a58436460>, <Element li at 0x7f9a58436500>, <Element li at 0x7f9a584365a0>, <Element li at 0x7f9a58436640>]
```

```
print(len(nodes_li))
print(type(nodes_li))
print(type(nodes_li[0]))
```

```
4
<class 'list'>
<class 'lxml.etree._Element'>
```

- //* 是使用* 代表匹配所有节点，也就是整个HTML文本中的所有节点都会被获取。可以看到，返回形式是一个列表，其后跟了节点的名称，如html、body、div、ul、li、a等，所有节点都包含在列表中了。
- //li 是选取所有li节点，调用时直接使用xpath()方法，返回形式是一个列表，每个元素是Element类型。如果要取出其中一个对象，可以直接用中括号加索引，如[0]。

匹配子节点

```
1 nodes = html.xpath('//ul//a')
2 print(nodes)
```

```
[<Element a at 0x7f9a5843caa0>, <Element a at 0x7f9a5843cbe0>, <Element a at 0x7f9a5843cc30>, <Element a at 0x7f9a5843cc80>]
```

- //ul匹配所有ul节点，//a表示匹配所有ul节点下的所有a节点，这里返回所有的a节点。

匹配父节点

- 这里先选取所有 a 元素，且这些元素拥有值为 link1.html 的 href 属性。
- 接着选取 a 元素的父节点，且其属性为class。

```
1 nodes = html.xpath('//a[@href="link3.html"]/../@class') # 匹配父节点
2 print(nodes)
```

```
['item-inactive']
```

```
1 nodes = html.xpath('//a[@href="link1.html"]/../@class') # 匹配父节点
2 print(nodes)
```

```
['item-0']
```

- 这里配匹内容为所有 li 节点，且这些节点拥有值为 item-1 的 class 属性，而HTML文本中符合条件的li节点有两个，所以结果返回两个匹配到的元素。

```
1 nodes = html.xpath('//li[@class="item-1"]')
2 print(nodes)
```

```
[<Element li at 0x7f9a58436500>, <Element li at 0x7f9a58436640>]
```

```
1 nodes = html.xpath('//li[@class="item-0"]')
2 print(nodes)
```

```
[<Element li at 0x7f9a58436460>]
```

- text() 可以获取节点中的文本。
- 这里配匹内容为所有 li 节点，且这些节点拥有值为 item-0 的 class 属性，接着是符合条件的 li 节点中的子节点 a 中的文本。

```
1  nodes = html.xpath('//li[@class="item-1"]/a/text()')
2  print(nodes)
```

```
['2nd item', '4th item']
```

```
1  nodes = html.xpath('//li[@class="item-0"]/a/text()')
2  print(nodes)
```

```
['1st item']
```

匹配成本

- 这里通过匹配ul节点中所有的文本，文本是一个列表，且是有多个值。
- strip()用于移除字符串头尾指定的字符（默认为空格或换行符）或字符序列。

```
1 nodes = html.xpath('//ul//text()')
2 print(nodes)
```

```
['\n        ', '\n            ', '1st item', '\n        ', '\n        ', '\n            ', '2nd item', '\n        ', '\n        ', '\n
', '3rd item', '\n        ', '\n        ', '\n            ', '4th item', '\n        ', '\n    ']
```

```
1 for node in nodes:
2     print(node.strip())
```

```
1st item


2nd item


3rd item


4th item
```

匹配成本

- string(.) 可以获取节点中所有的文本，且为一个值。
- normalize-space()可以去除换行和空格。
- 这里通过匹配ul节点中所有的文本，文本是一个列表，且是有多个值。

```
nodes = html.xpath('//ul')
print(nodes)
print(nodes[0])
```

```
[<Element ul at 0x7f9a5843caf0>]
<Element ul at 0x7f9a5843caf0>
```

```
result1 = nodes[0].xpath('string(.)')
print(result1)
```

```
1st item

2nd item

3rd item

4th item
```

```
result2 = nodes[0].xpath('normalize-space(string(.))') # normalize-space 去除换行、空格
print(result2)
```

```
1st item 2nd item 3rd item 4th item
```

获取属性

- 这里我们通过@href即可获取节点的href属性。注意，此处和匹配属性的方法不同，匹配属性是中括号加属性名和值来限定某个属性，如[@href="link1.html"]，而此处的@href指的是获取节点的某个属性。

```
1 result = html.xpath('//li/a/@href')
2 print(result)
```

```
['link1.html', 'link2.html', 'link3.html', 'link5.html']
```

属性多值匹配

- 这里li节点的class属性有两个值li和item-0。
- 需要通过contains()方法，第一个参数传入属性名称，第二个参数传入属性值，只要此属性包含所传入的属性值。

```
text = '''
<li class="li item-0">
  <a href="link.html">1st item</a>
</li>
'''
new_html = etree.HTML(text)
result = new_html.xpath('//li[contains(@class, "li item-0")]/a/text()')
print(result)
```

```
['1st item']
```

```
text = '''
<li class="li item-0">
  <a href="link.html">1st item</a>
</li>
'''
new_html = etree.HTML(text)
result = new_html.xpath('//li[contains(@class, "item-0")]/a/text()')
print(result)
```

```
['1st item']
```

多属性匹配

- 这里的li节点又增加了一个属性name。要确定这个节点，需要同时根据class和name属性来选择。
- 二者需要同时满足，需要用and操作符相连，相连之后置于中括号内进行条件筛选。

```
text = '''
<li class="li item-0" name="new_item">
  <a href="link.html">new 1st item</a>
</li>
'''
new_html = etree.HTML(text)
result = new_html.xpath('//li[contains(@class, "item-0") and @name="new_item"]/a/text()')
print(result)
```

```
['new 1st item']
```

按序选择

- 第一次选择时，我们选取了第一个li节点，中括号中传入数字1即可。注意，这里和代
- 码中不同，序号是以1开头的，不是以0开头。
- 第二次选择时，我们选取了最后一个li节点，中括号中传入last()即可，返回的便是最后 一个li节点。
- 第三次选择时，我们选取了位置小于3的li节点，也就是位置序号为1和2的节点，得到的结果就是前两个li节点。
- 第四次选择时，我们选取了倒数第三个li节点，中括号中传入last()-2即可。因为last()是最后一个，所以last()-2就是倒数第三个。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
    <body><div>
    <ul>
        <li class="item-0">
            <a href="link1.html">1st item</a>
        </li>
        <li class="item-1">
            <a href="link2.html">2nd item</a>
        </li>
        <li class="item-inactive">
            <a href="link3.html">3rd item</a>
        </li>
        <li class="item-1">
            <a href="link5.html">4th item</a>
        </li>
    </ul>
</div>
</body>
</html>
```

```
result = html.xpath('//li[1]/a/text()')
print(result)
```

```
['1st item']
```

```
result = html.xpath('//li[last()]/a/text()')
print(result)
```

```
['4th item']
```

```
result = html.xpath('//li[position()<3]/a/text()')
print(result)
```

```
['1st item', '2nd item']
```

```
result = html.xpath('//li[last()-2]/a/text()')
print(result)
```

```
['2nd item']
```

Thank you!