选择器

在抓取网页时，您需要执行的最常见任务是从HTML源提取数据。有几个库可用于实现此目的：

- BeautifulSoup是Python程序员中非常流行的Web抓取库，它根据HTML代码的结构构造一个Python对象，并且合理地处理坏标记，但它有一个缺点：它很慢。
- lxml是一个XML解析库（也可以解析HTML），它使用基于ElementTree的pythonic API。（lxml不是Python标准库的一部分。）

Scrapy带有自己的提取数据机制。它们被称为选择器，因为它们“选择”由XPath或CSS表达式指定的HTML文档的某些部分。

XPath是一种用于在XML文档中选择节点的语言，也可以与HTML一起使用。CSS是一种将样式应用于HTML文档的语言。它定义选择器以将这些样式与特定HTML元素相关联。

Scrapy选择器是在lxml库上构建的，这意味着它们在速度和解析准确性方面非常相似。

这个页面解释了选择器如何工作和描述它们的API非常小而且简单，不像lxml API要大得多，因为除了选择标记文档之外，lxml库还可以用于许多其他任务。

有关选择器API的完整参考，请参阅 选择器参考

使用选择器

构造选择器

Scrapy选择器是 `Selector` 通过传递**文本**或 `TextResponse` 对象构造的类的实例。它会根据输入类型自动选择最佳解析规则（XML与HTML）：

```
>>> from scrapy.selector import Selector
>>> from scrapy.http import HtmlResponse
```

从文本构造：

```
>>> body = '<html><body><span>good</span></body></html>'
>>> Selector(text=body).xpath('//span/text()').extract()
[u'good']
```

从响应构建：

```
>>> response = HtmlResponse(url='http://example.com', body=body)
>>> Selector(response=response).xpath('//span/text()').extract()
[u'good']
```

为方便起见，响应对象在*.selector*属性上公开选择器，在可能的情况下使用此快捷方式是完全可以的：

```
>>> response.selector.xpath('//span/text()').extract()
[u'good']
```

使用选择器

为了解释如何使用选择器，我们将使用*Scrapy shell*（提供交互式测试）和Scrapy文档服务器中的示例页面：

https://doc.scrapy.org/en/latest/_static/selectors-sample1.html

这是它的HTML代码：

```
<html>
 <head>
  <base href='http://example.com/' />
  <title>Example website</title>
 </head>
 <body>
  <div id='images'>
   <a href='image1.html'>Name: My image 1 <br /><img src='image1_thumb.jpg' /></a>
   <a href='image2.html'>Name: My image 2 <br /><img src='image2_thumb.jpg' /></a>
   <a href='image3.html'>Name: My image 3 <br /><img src='image3_thumb.jpg' /></a>
   <a href='image4.html'>Name: My image 4 <br /><img src='image4_thumb.jpg' /></a>
   <a href='image5.html'>Name: My image 5 <br /><img src='image5_thumb.jpg' /></a>
  </div>
 </body>
</html>
```

首先，让我们打开shell：

```
scrapy shell https://doc.scrapy.org/en/latest/_static/selectors-sample1.html
```

然后，在shell加载之后，您将获得响应作为`response` shell变量，并在`response.selector`属性中附加选择器。

由于我们正在处理HTML，因此选择器将自动使用HTML解析器。

因此，通过查看该页面的HTML代码，让我们构建一个XPath来选择title标签内的文本：

```
>>> response.selector.xpath('//title/text()')
[<Selector (text) xpath=//title/text()>]
```

使用XPath和CSS查询响应非常常见，响应包括两个便捷快捷方式：`response.xpath()` 和 `response.css()` ：

```
>>> response.xpath('//title/text()')
[<Selector (text) xpath=//title/text()>]
>>> response.css('title::text')
[<Selector (text) xpath=//title/text()>]
```

正如你所看到的，`.xpath()` 并且 `.css()` 方法返回一个 `SelectorList` 实例，这是新的选择列表。此API可用于快速选择嵌套数据：

```
>>> response.css('img').xpath('@src').extract()
[u'image1_thumb.jpg',
 u'image2_thumb.jpg',
 u'image3_thumb.jpg',
 u'image4_thumb.jpg',
 u'image5_thumb.jpg']
```

要实际提取文本数据，必须调用selector `.extract()` 方法，如下所示：

```
>>> response.xpath('//title/text()').extract()
[u'Example website']
```

如果只想提取第一个匹配的元素，可以调用选择器 `.extract_first()`

```
>>> response.xpath('//div[@id="images"]/a/text()').extract_first()
u'Name: My image 1 '
```

`None` 如果没有找到元素，则返回：

```
>>> response.xpath('//div[@id="not-exists"]/text()').extract_first() is None
True
```

可以提供默认返回值作为参数，以代替 `None` ：

```
>>> response.xpath('//div[@id="not-exists"]/text()').extract_first(default='not-found')
'not-found'
```

请注意，CSS选择器可以使用CSS3伪元素选择文本或属性节点：

```
>>> response.css('title::text').extract()
[u'Example website']
```

现在我们将获得基本URL和一些图像链接：

```
>>> response.xpath('//base/@href').extract()
[u'http://example.com/']

>>> response.css('base::attr(href)').extract()
[u'http://example.com/']

>>> response.xpath('//a[contains(@href, "image")]/@href').extract()
[u'image1.html',
 u'image2.html',
 u'image3.html',
 u'image4.html',
 u'image5.html']

>>> response.css('a[href*=image]::attr(href)').extract()
[u'image1.html',
 u'image2.html',
 u'image3.html',
 u'image4.html',
 u'image5.html']

>>> response.xpath('//a[contains(@href, "image")]/img/@src').extract()
[u'image1_thumb.jpg',
 u'image2_thumb.jpg',
 u'image3_thumb.jpg',
 u'image4_thumb.jpg',
 u'image5_thumb.jpg']

>>> response.css('a[href*=image] img::attr(src)').extract()
[u'image1_thumb.jpg',
 u'image2_thumb.jpg',
 u'image3_thumb.jpg',
 u'image4_thumb.jpg',
 u'image5_thumb.jpg']
```

嵌套选择器

选择方法（`.xpath()`或`.css()`）返回相同类型的选择器列表，因此您也可以为这些选择器调用选择方法。这是一个例子：

```
>>> links = response.xpath('//a[contains(@href, "image")]')
>>> links.extract()
[u'<a href="image1.html">Name: My image 1 <br><img src="image1_thumb.jpg"></a>',
 u'<a href="image2.html">Name: My image 2 <br><img src="image2_thumb.jpg"></a>',
 u'<a href="image3.html">Name: My image 3 <br><img src="image3_thumb.jpg"></a>',
 u'<a href="image4.html">Name: My image 4 <br><img src="image4_thumb.jpg"></a>',
 u'<a href="image5.html">Name: My image 5 <br><img src="image5_thumb.jpg"></a>']

>>> for index, link in enumerate(links):
...     args = (index, link.xpath('@href').extract(), link.xpath('img/@src').extract())
...     print 'Link number %d points to url %s and image %s' % args

Link number 0 points to url [u'image1.html'] and image [u'image1_thumb.jpg']
Link number 1 points to url [u'image2.html'] and image [u'image2_thumb.jpg']
Link number 2 points to url [u'image3.html'] and image [u'image3_thumb.jpg']
Link number 3 points to url [u'image4.html'] and image [u'image4_thumb.jpg']
Link number 4 points to url [u'image5.html'] and image [u'image5_thumb.jpg']
```

使用具有正则表达式的选择器

`Selector` 还有一种 `.re()` 使用正则表达式提取数据的方法。但是，与using `.xpath()` 或 `.css()` methods 不同， `.re()` 返回unicode字符串列表。所以你不能构造嵌套 `.re()` 调用。

以下是用于从上面的HTML代码中提取图像名称的示例：

```
>>> response.xpath('//a[contains(@href, "image")]/text()').re(r'Name:\s*(.*)')
[u'My image 1',
 u'My image 2',
 u'My image 3',
 u'My image 4',
 u'My image 5']
```

这里有一个额外的辅助往复 `.extract_first()` 的 `.re()` ，命名 `.re_first()` 。用它来提取第一个匹配的字符串：

```
>>> response.xpath('//a[contains(@href, "image")]/text()').re_first(r'Name:\s*(.*)')
u'My image 1'
```

相对XPath的工作

请记住，如果您正在嵌套选择器并使用以XP开头的XPath `/` ，那么XPath对于文档是绝对的，而不是相对于 `Selector` 您从中调用它。

例如，假设您要提取 `<p>` 元素内的所有 `<div>` 元素。首先，您将获得所有 `<div>` 元素：

```
>>> divs = response.xpath('//div')
```

首先，您可能会尝试使用以下方法，这是错误的，因为它实际上 `<p>` 从文档中提取所有元素，而不仅仅是 `<div>` 元素内部的元素：

```
>>> for p in divs.xpath('//p'):  # this is wrong - gets all <p> from the whole document
...     print p.extract()
```

这是正确的方法（注意前缀为 `.//p` XPath 的点）：

```
>>> for p in divs.xpath('.//p'):  # extracts all <p> inside
...     print p.extract()
```

另一个常见的案例是提取所有直接 `<p>` 子女：

```
>>> for p in divs.xpath('p'):
...     print p.extract()
```

有关相对XPath的更多详细信息，请参阅XPath规范中的“ 位置路径”部分。

XPath表达式中的变量

XPath允许您使用 `$somevariable` 语法引用XPath表达式中的变量。这有点类似于SQL世界中的参数化查询或预准备语句，您可以使用占位符替换查询中的某些参数 `?` ，然后将其替换为随查询传递的值。

这是一个基于其“id”属性值匹配元素的示例，而不对其进行硬编码（如前所示）：

```
>>> # `$val` used in the expression, a `val` argument needs to be passed
>>> response.xpath('//div[@id=$val]/a/text()', val='images').extract_first()
u'Name: My image 1 '
```

这是另一个例子，找到 `<div>` 包含五个 `<a>` 子节点的标签的“id”属性（这里我们将值 `5` 作为整数传递）：

```
>>> response.xpath('//div[count(a)=$cnt]/@id', cnt=5).extract_first()
u'images'
```

调用时，所有变量引用都必须具有绑定值 `.xpath()` （否则您将获得异常）。这是通过根据需要传递尽可能多的命名参数来完成的。 `ValueError: XPath error:`

parsel，为Scrapy选择器提供动力的库，有关于XPath变量的更多细节和示例。

使用EXSLT扩展

在lxml上构建，Scrapy选择器还支持一些EXSLT扩展，并附带这些预先注册的命名空间以在XPath表达式中使用：

字首	命名空间	用法
回覆	http://exslt.org/regular-expressions	常用表达
组	http://exslt.org/sets	设定操纵

正则表达式

`test()` 例如，当XPath `starts-with()` 或 `contains()` 不足时，该函数可以证明非常有用 。

示例选择列表项中的链接，其中“class”属性以数字结尾：

```
>>> from scrapy import Selector
>>> doc = """
... <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 item</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></li>
...     </ul>
... </div>
... """
>>> sel = Selector(text=doc, type="html")
>>> sel.xpath('//li//@href').extract()
[u'link1.html', u'link2.html', u'link3.html', u'link4.html', u'link5.html']
>>> sel.xpath('//li[re:test(@class, "item-\d$")]//@href').extract()
[u'link1.html', u'link2.html', u'link4.html', u'link5.html']
>>>
```

警告

C库 `libxslt` 本身不支持EXSLT正则表达式，因此lxml的实现使用了Python `re` 模块的钩子。因此，在XPath表达式中使用regexp函数可能会增加很小的性能损失。

设定操作

例如，在提取文本元素之前，这些可以方便地排除文档树的部分。

使用itemscopes和相应的itemprops组提取微数据（取自http://schema.org/Product的样本内容）的示例：

```
>>> doc = """
... <div itemscope itemtype="http://schema.org/Product">
...   <span itemprop="name">Kenmore White 17" Microwave</span>
...   <img src="kenmore-microwave-17in.jpg" alt='Kenmore 17" Microwave' />
...   <div itemprop="aggregateRating"
...     itemscope itemtype="http://schema.org/AggregateRating">
...    Rated <span itemprop="ratingValue">3.5</span>/5
...    based on <span itemprop="reviewCount">11</span> customer reviews
...   </div>
...
...   <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
...     <span itemprop="price">$55.00</span>
...     <link itemprop="availability" href="http://schema.org/InStock" />In stock
...   </div>
...
...   Product description:
...   <span itemprop="description">0.7 cubic feet countertop microwave.
...   Has six preset cooking categories and convenience features like
...   Add-A-Minute and Child Lock.</span>
...
...   Customer reviews:
...
...   <div itemprop="review" itemscope itemtype="http://schema.org/Review">
...     <span itemprop="name">Not a happy camper</span> -
...     by <span itemprop="author">Ellie</span>,
...     <meta itemprop="datePublished" content="2011-04-01">April 1, 2011
...     <div itemprop="reviewRating" itemscope itemtype="http://schema.org/Rating">
...       <meta itemprop="worstRating" content = "1">
...       <span itemprop="ratingValue">1</span>/
...       <span itemprop="bestRating">5</span>stars
...     </div>
...     <span itemprop="description">The lamp burned out and now I have to replace
...     it. </span>
...   </div>
...
...   <div itemprop="review" itemscope itemtype="http://schema.org/Review">
...     <span itemprop="name">Value purchase</span> -
...     by <span itemprop="author">Lucas</span>,
...     <meta itemprop="datePublished" content="2011-03-25">March 25, 2011
...     <div itemprop="reviewRating" itemscope itemtype="http://schema.org/Rating">
...       <meta itemprop="worstRating" content = "1"/>
...       <span itemprop="ratingValue">4</span>/
...       <span itemprop="bestRating">5</span>stars
...     </div>
...     <span itemprop="description">Great microwave for the price. It is small and
...     fits in my apartment.</span>
...   </div>
...   ...
... </div>
... """
>>> sel = Selector(text=doc, type="html")
>>> for scope in sel.xpath('//div[@itemscope]'):
...     print "current scope:", scope.xpath('@itemtype').extract()
...     props = scope.xpath('''
...                 set:difference(./descendant::*/@itemprop,
...                                .//*[@itemscope]/*/@itemprop)''')
...     print "    properties:", props.extract()
...     print

current scope: [u'http://schema.org/Product']
    properties: [u'name', u'aggregateRating', u'offers', u'description', u'review', u'review']

current scope: [u'http://schema.org/AggregateRating']
    properties: [u'ratingValue', u'reviewCount']

current scope: [u'http://schema.org/Offer']
    properties: [u'price', u'availability']

current scope: [u'http://schema.org/Review']
    properties: [u'name', u'author', u'datePublished', u'reviewRating', u'description']

current scope: [u'http://schema.org/Rating']
    properties: [u'worstRating', u'ratingValue', u'bestRating']

current scope: [u'http://schema.org/Review']
    properties: [u'name', u'author', u'datePublished', u'reviewRating', u'description']
```

```
current scope: [u'http://schema.org/Rating']
    properties: [u'worstRating', u'ratingValue', u'bestRating']

>>>
```

在这里，我们首先迭代 `itemscope` 元素，并且对于每个元素，我们寻找所有 `itemprops` 元素并排除那些本身在另一个元素中的元素 `itemscope` 。

一些XPath技巧

根据ScrapingHub博客上的这篇文章，在使用带有Scrapy选择器的XPath时，您可能会发现一些有用的提示。如果您还不熟悉XPath，可能需要先看看这个XPath教程。

在条件中使用文本节点

当您需要使用文本内容作为XPath字符串函数的参数时，请避免使用 `.//text()` 和使用 `.` 。

这是因为表达式 `.//text()` 产生了一组文本元素 - 一个 *节点集*。并且当节点集被转换成字符串，当它是作为参数传递给像字符串功能这恰好 `contains()` 或 `starts-with()` ，它导致文仅用于第一个元素。

例：

```
>>> from scrapy import Selector
>>> sel = Selector(text='<a href="#">Click here to go to the <strong>Next Page</strong></a>')
```

将 *节点集* 转换为字符串：

```
>>> sel.xpath('//a//text()').extract() # take a peek at the node-set
[u'Click here to go to the ', u'Next Page']
>>> sel.xpath("string(//a[1]//text())").extract() # convert it to string
[u'Click here to go to the ']
```

一个 *节点* 转换为字符串，但是，拼文本的本身及其所有的后代：

```
>>> sel.xpath("//a[1]").extract() # select the first node
[u'<a href="#">Click here to go to the <strong>Next Page</strong></a>']
>>> sel.xpath("string(//a[1])").extract() # convert it to string
[u'Click here to go to the Next Page']
```

因此， `.//text()` 在这种情况下，使用节点集不会选择任何内容：

```
>>> sel.xpath("//a[contains(.//text(), 'Next Page')]").extract()
[]
```

但使用`.`意味着节点，工作：

```
>>> sel.xpath("//a[contains(., 'Next Page')]").extract()
[u'<a href="#">Click here to go to the <strong>Next Page</strong></a>']
```

注意// node [1]和（// node）之间的区别[1]

`//node[1]`选择在各自父母下首先出现的所有节点。

`(//node)[1]`选择文档中的所有节点，然后只获取其中的第一个节点。

例：

```
>>> from scrapy import Selector
>>> sel = Selector(text="""
....:     <ul class="list">
....:         <li>1</li>
....:         <li>2</li>
....:         <li>3</li>
....:     </ul>
....:     <ul class="list">
....:         <li>4</li>
....:         <li>5</li>
....:         <li>6</li>
....:     </ul>""")
>>> xp = lambda x: sel.xpath(x).extract()
```

这将获得所有第一个`<li>`元素，无论它是它的父元素：

```
>>> xp("//li[1]")
[u'<li>1</li>', u'<li>4</li>']
```

这将获得`<li>`整个文档中的第一个元素：

```
>>> xp("(//li)[1]")
[u'<li>1</li>']
```

这将获取 父项`<li>`下的所有第一个元素`<ul>`：

```
>>> xp("//ul/li[1]")
[u'<li>1</li>', u'<li>4</li>']
```

这将获得 整个文档中父级 `<li>` 下的第一个元素 `<ul>` ：

```
>>> xp("(//ul/li)[1]")
[u'<li>1</li>']
```

在按类查询时，请考虑使用

因为一个元素可以包含多个CSS类，所以按类选择元素的XPath方式相当冗长：

```
*[contains(concat(' ', normalize-space(@class), ' '), ' someclass ')]
```

如果您使用 `@class='someclass'` ，最终可能会丢失具有其他类的元素，如果您只是用来弥补它，您可能会得到更多您想要的元素，如果它们具有共享字符串的不同类名。 `contains(@class, 'someclass')` `someclass`

事实证明，Scrapy选择器允许您链接选择器，因此大多数时候您可以使用CSS按类选择，然后在需要时切换到XPath：

```
>>> from scrapy import Selector
>>> sel = Selector(text='<div class="hero shout"><time datetime="2014-07-23 19:00">Special
date</time></div>')
>>> sel.css('.shout').xpath('./time/@datetime').extract()
[u'2014-07-23 19:00']
```

这比使用上面显示的详细XPath技巧更清晰。只需记住 `.` 在后面的XPath表达式中使用。

内置选择器参考

选择对象

class **scrapy.selector.Selector** (*response = None , text = None , type = None*)

一个实例 `Selector` 是对选择其内容的某些部分的响应的包装器。

`response` 是一个 `HtmlResponse` 或一个 `XmlResponse` 将被用于选择和提取的数据对象。

`text` 是一个unicode字符串或utf-8编码的文本，用于a `response` 不可用的情况。使用 `text` 和 `response` 在一起是未定义的行为。

`type` 定义选择器类型，它可以是 `"html"` ， `"xml"` 或 `None` （默认）。

如果 `type` 是 `None` ，则选择器会根据 `response` 类型自动选择最佳类型（参见下文），或者默认选择 `"html"` 与其一起使用的类型 `text` 。

如果 `type` is `None` 和a `response` 被传递，则从响应类型推断出选择器类型，如下所示：

- `"html"` 对于 `HtmlResponse` 类型
- `"xml"` 对于 `XmlResponse` 类型
- `"html"` 除了别的什么

否则，如果 `type` 设置，则强制选择器类型，不会进行检测。

`xpath` (*查询*)

查找与xpath匹配的节点 `query` ，并将结果作为 `SelectorList` 实例返回， 并将所有元素展平。List元素也实现了 `Selector` 接口。

`query` 是一个包含要应用的XPATH查询的字符串。

注意

为方便起见，可以将此方法称为 `response.xpath()`

`css` (*查询*)

应用给定的CSS选择器并返回一个 `SelectorList` 实例。

`query` 是一个包含要应用的CSS选择器的字符串。

在后台，CSS查询使用cssselect库和run `.xpath()` 方法转换为XPath查询 。

注意

为方便起见，这种方法可以称为 `response.css()`

`extract` ( )

序列化并将匹配的节点作为unicode字符串列表返回。编码内容百分比未加引号。

`re` (*正则表达式*)

应用给定的正则表达式并返回带有匹配项的unicode字符串列表。

`regex` 可以是已编译的正则表达式，也可以是将使用编译为正则表达式的字符串 `re.compile(regex)`

注意

需要注意的是 `re()` 和 `re_first()` 两个解码HTML实体（除 `<` 和 `&` ）。

> **register_namespace**(*前缀* , *uri*)
>
> 注册要在此中使用的给定名称空间 `Selector` 。如果不注册名称空间，则无法从非标准名称空间中选择或提取数据。见下面的例子。

> **remove_namespaces**( )
>
> 删除所有名称空间，允许使用无名称空间的xpath遍历文档。见下面的例子。

> **__nonzero__**( )
>
> `True` 如果选择了任何真实内容，则返回 `False` 。换句话说，a的布尔值 `Selector` 由它选择的内容给出。

SelectorList对象

类 **scrapy.selector.SelectorList**

的 `SelectorList` 类是内置的一个子类 `list` 类，它提供了几个方法。

> **xpath**(*查询*)
>
> `.xpath()` 为此列表中的每个元素调用方法，并将其结果作为另一个返回 `SelectorList` 。
>
> `query` 与中的一个是相同的论点 `Selector.xpath()`

> **css**(*查询*)
>
> `.css()` 为此列表中的每个元素调用方法，并将其结果作为另一个返回 `SelectorList` 。
>
> `query` 与中的一个是相同的论点 `Selector.css()`

> **extract**( )
>
> `.extract()` 为此列表中的每个元素调用该方法，并将其结果返回为flaticned，作为unicode字符串列表。

> **re**( )
>
> `.re()` 为此列表中的每个元素调用该方法，并将其结果返回为flaticned，作为unicode字符串列表。

HTML响应的选择器示例

这里有几个 `Selector` 例子来说明几个概念。在所有情况下，我们假设已经 `Selector` 实例化了这样的 `HtmlResponse` 对象：

```
sel = Selector(html_response)
```

1. `<h1>`从HTML响应主体中选择所有元素，返回`Selector`对象列表（即`SelectorList`对象）：

```
sel.xpath("//h1")
```

2. `<h1>`从HTML响应主体中提取所有元素的文本，返回unicode字符串列表：

```
sel.xpath("//h1").extract()         # this includes the h1 tag
sel.xpath("//h1/text()").extract()  # this excludes the h1 tag
```

3. 迭代所有`<p>`标签并打印其class属性：

```
for node in sel.xpath("//p"):
    print node.xpath("@class").extract()
```

XML响应的选择器示例

这里有几个例子来说明几个概念。在这两种情况下，我们假设已经`Selector`实例化了一个`XmlResponse`像这样的 对象：

```
sel = Selector(xml_response)
```

1. `<product>`从XML响应主体中选择所有元素，返回`Selector`对象列表（即`SelectorList`对象）：

```
sel.xpath("//product")
```

2. 从需要注册命名空间的Google Base XML Feed中提取所有价格：

```
sel.register_namespace("g", "http://base.google.com/ns/1.0")
sel.xpath("//g:price").extract()
```

删除命名空间

在处理抓取项目时，通常很方便地完全删除命名空间并使用元素名称来编写更简单/方便的XPath。您可以使用该 `Selector.remove_namespaces()` 方法。

让我们展示一个用GitHub博客原子源来说明这一点的例子。

首先，我们打开带有我们要抓取的url的shell：

```
$ scrapy shell https://github.com/blog.atom
```

一旦进入shell，我们就可以尝试选择所有 `<link>` 对象并看到它不起作用（因为Atom XML命名空间混淆了那些节点）：

```
>>> response.xpath("//link")
[]
```

但是一旦我们调用该 `Selector.remove_namespaces()` 方法，所有节点都可以通过它们的名称直接访问：

```
>>> response.selector.remove_namespaces()
>>> response.xpath("//link")
[<Selector xpath='//link' data=u'<link xmlns="http://www.w3.org/2005/Atom'>,
 <Selector xpath='//link' data=u'<link xmlns="http://www.w3.org/2005/Atom'>,
 ...
```

如果您想知道为什么默认情况下并不总是调用命名空间删除过程而不必手动调用它，这是因为有两个原因，按照相关性，它们是：

1. 删除命名空间需要迭代和修改文档中的所有节点，这对于Scrapy抓取的所有文档执行是相当昂贵的操作
2. 在某些情况下，如果某些元素名称在名称空间之间发生冲突，则实际上可能需要使用名称空间。这些案件非常罕见。