# HTML解析

通过上面的库，都可以拿到HTML内容。

HTML的内容返回给浏览器，浏览器就会解析它，并对它渲染。

HTML 超文本表示语言，设计的初衷就是为了超越普通文本，让文本表现力更强。
XML 扩展标记语言，不是为了代替HTML，而是觉得HTML的设计中包含了过多的格式，承担了一部分数据之外的任务，所以才设计了XML只用来描述数据。
HTML和XML都有结构，使用标记形成树型的嵌套结构。DOM（Document Object Model）来解析这种嵌套树型结构，浏览器往往都提供了对DOM操作的API，可以用面向对象的方式来操作DOM。

## XPath ***

http://www.w3school.com.cn/xpath/index.asp 中文教程
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。

工具
XMLQuire win7+需要.NET框架4.0-4.5。
测试XML、XPath

```
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology
      society in England, the young survivors lay the
```

```
      foundation for a new society.</description>
   </book>
   <book id="bk104">
      <author>Corets, Eva</author>
      <title>Oberon's Legacy</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2001-03-10</publish_date>
      <description>In post-apocalypse England, the mysterious
      agent known only as Oberon helps to create a new life
      for the inhabitants of London. Sequel to Maeve
      Ascendant.</description>
   </book>
</bookstore>
```

在 XPath 中，有七种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点。

/ 根结点
元素节点
Corets, Eva 元素节点，
id="bk104" 是属性节点，id是元素节点book的属性

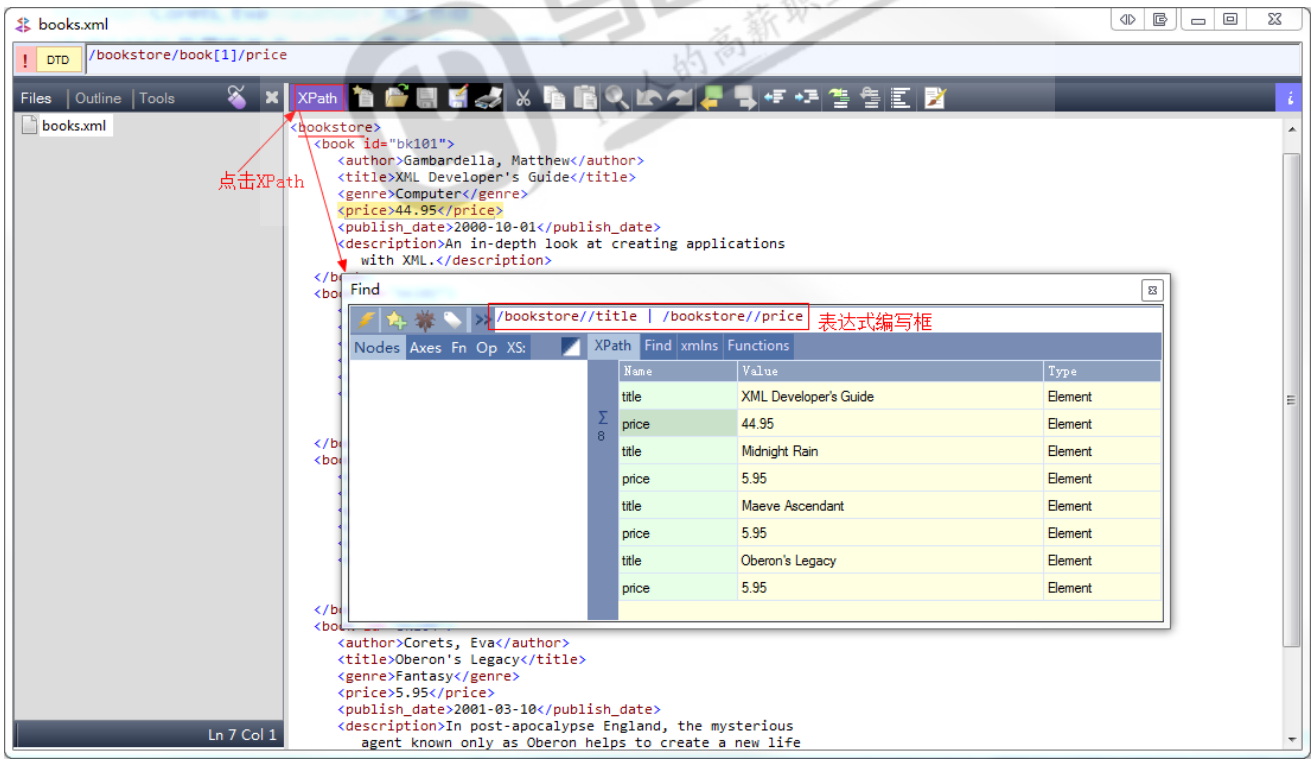
节点之间的嵌套形成**父子(parent、children)关系**。
具有同一个父节点的不同节点是**兄弟(sibling)关系**。

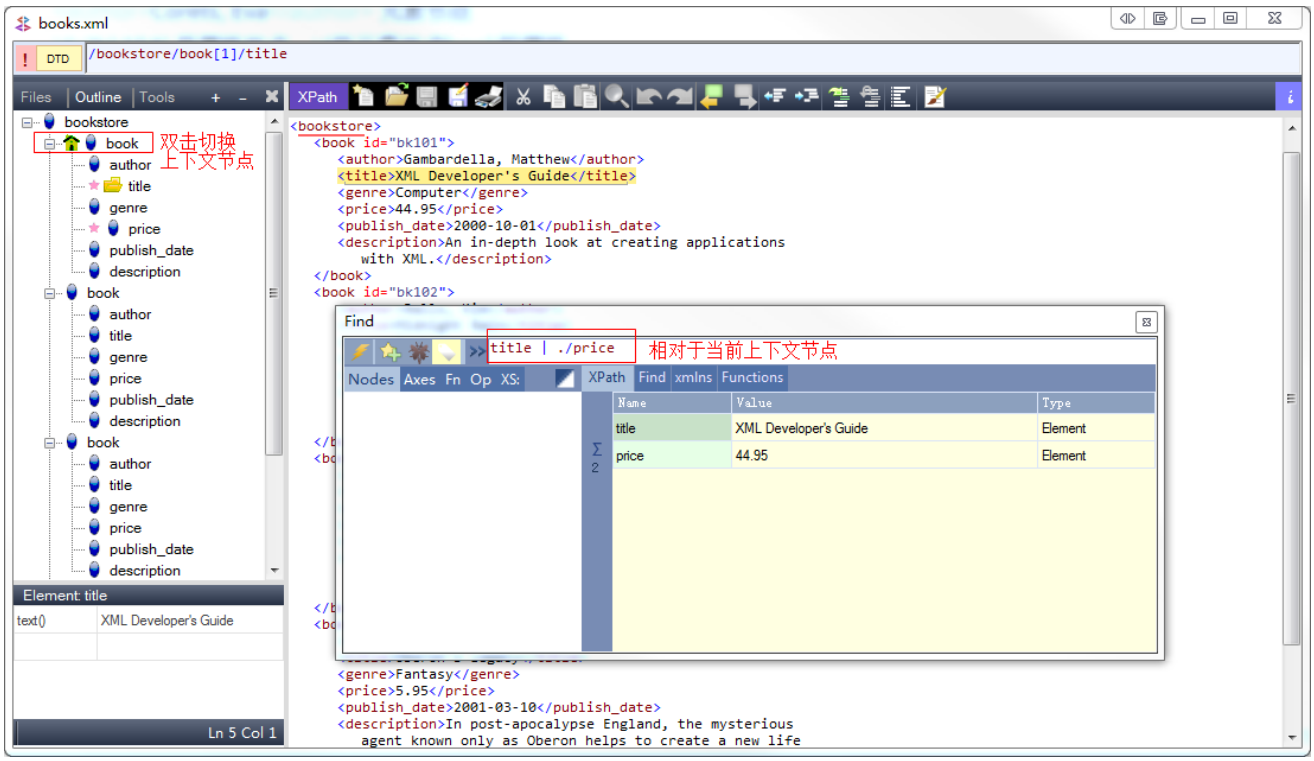
谓语（Predicates）
谓语用来查找某个特定的节点或者包含某个指定的值的节点。
谓语被嵌在方括号中。
谓语就是查询的条件。

| 操作符或表达式 | 含义 |
| --- | --- |
| / | 从根节点开始找 |
| // | 从当前节点开始的任意层找 |
| . | 当前节点 |
| .. | 当前节点的父节点 |
| @ | 选择属性 |
| 节点名 | 选取所有这个节点名的节点 |
| * | 匹配任意元素节点 |
| @* | 匹配任意属性节点 |
| node() | 匹配任意类型的节点 |
| text() | 匹配text类型节点 |

XPath 轴（Axes）

| 轴名称 | 结果 |
| --- | --- |
| ancestor | 选取当前节点的所有先辈（父、祖父等） |
| ancestor-or-self | 选取当前节点的所有先辈（父、祖父等）以及当前节点本身 |
| attribute | 选取当前节点的所有属性。@id 等价于 attribute::id |
| child | 选取当前节点的所有子元素。title 等价于 child:title |
| descendant | 选取当前节点的所有后代元素（子、孙等） |
| descendant-or-self | 选取当前节点的所有后代元素（子、孙等）以及当前节点本身 |
| following | 选取文档中当前节点的结束标签之后的所有节点 |
| namespace | 选取当前节点的所有命名空间节点 |
| parent | 选取当前节点的父节点 |
| preceding | 直到所有这个节点的父辈节点，顺序选择每个父辈节点前的所有同级节点 |
| preceding-sibling | 选取当前节点之前的所有同级节点 |
| self | 选取当前节点。 . 等价于 self::node() |






## XPATH实例

以斜杠开始的称为绝对路径，表示从根开始。

不以斜杆开始的称为相对路径，一般都是依照当前节点来计算。当前节点在上下文环境中，当前节点很可能已经不是根节点了。

一般为了方便，往往xml如果层次很深，都会使用//来查找节点。

| 路径表达式 | 含义 |
| --- | --- |
| title | 选取当前节点下所有title子节点 |
| /book | 从根结点找子节点是book的，找不到 |
| book/title | 当前节点下所有子节点book下的title节点 |
| //title | 从根节点向下找任意层中title的节点 |
| book//title | 当前节点下所有book子节点下任意层次的title节点 |
| //@name | 任意层下含有name的 **属性**，取回的是属性 |
| //book[@id] | 任意层下含有name属性的book节点 |
| //book[@id="bk101"] | 任意层下含有name属性且等于'bk101'的book节点 |
| /bookstore/book[1] | 根节点bookstore下第一个book节点，从1开始 |
| /bookstore/book[1]/@id | 根节点bookstore下第一个book节点的id属性 |
| /bookstore/book[last()-1] | 根节点bookstore下倒数第二个book节点，函数last() |
| /bookstore/* | 匹配根节点bookstore的所有子节点，不递归 |
| //* | 匹配所有子孙节点 |
| //*[@*] | 匹配所有有属性的节点 |
| //book[@*] | 匹配所有有属性的book节点 |
| //@* | 匹配所有属性 |
| //book/title \| //price | 匹配book下的title节点或者任意层下的price |
| //book[position()=2] | 匹配book节点，取第二个 |
| //book[position()<last()-1] | 匹配book节点，取位置小于倒数第二个 |
| //book[price>40] | 匹配price节点值大于40的book节点 |
| //book[2]/node() | 匹配位置为2的book节点下的所有类型的节点 |
| //book[1]/text() | 匹配第一个book节点下的所有文本子节点 |
| //book[1]//text() | 匹配第一个book节点下的所有文本节点 |
| //*[local-name()='book'] | 匹配所有节点且不带限定名的节点名称为book的所有节点 |
| //book/child::node()[local-name()='price' and text()<10] | 所有book节点的子节点中名字叫做price的且其内容小于10的节点，等价于 //book/price[text()<10] |

## lxml

lxml是Python下功能丰富的XML、HTML解析库，性能非常好，是对libxml2 和 libxslt的封装。
最新版支持Python 2.6+，python3支持到3.6。

CentOS编译安装需要
`# yum install libxml2-devel libxslt-devel`
注意，不同平台不一样，参看 http://lxml.de/installation.html

lxml安装
`$ pip install lxml`

```
from lxml import etree

# 使用etree构建HTML
root = etree.Element('html')
print(type(root))
print(root.tag)

body = etree.Element('body')
root.append(body)

print(etree.tostring(root))

sub = etree.SubElement(body, 'child1') # 增加子节点
print(type(sub))
sub = etree.SubElement(body, 'child2').append(etree.Element('child21'))

print(etree.tostring(root, pretty_print=True).decode())
```

etree还提供了2个有用的函数
etree.HTML(text) 解析HTML文档，返回根节点
**anode.xpath('xpath路径') 对节点使用xpath语法**

从豆瓣电影中提取“本周口碑榜”

```
from lxml import etree
import requests

url = 'https://movie.douban.com/'
ua = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.75
Safari/537.36"

with requests.get(url, headers={'User-agent':ua}) as response:
    content = response.text # HTML内容

    html = etree.HTML(content) # 分析HTML，返回DOM根节点
    titles = html.xpath("//div[@class='billboard-bd']//tr/td/a/text()") # 返回文本列表
    for t in titles:  # 豆瓣电影之 本周口碑榜
        print(t)
```

## BeautifulSoup4 **

BeautifulSoup可以从HTML、XML中提取数据。目前BS4在持续开发。

官方中文文档
https://www.crummy.com/software/BeautifulSoup/bs4/doc.zh/

## 安装

```
$ pip install beautifulsoup4
```

导入
`from bs4 import BeautifulSoup`

## 初始化

BeautifulSoup(markup="", features=None)
markup可以是文件对象或者html字符串
features指定解析器，返回一个文档对象

```
from bs4 import BeautifulSoup

# 文件对象
soup = BeautifulSoup(open("index.html"))
# 标记字符串
soup = BeautifulSoup("<html>data</html>")
```

可以不指定解析器，就依赖系统已经安装的解析器库了。

| 解析器 | 使用方法 | 优势 | 劣势 |
| --- | --- | --- | --- |
| Python标准库 | `BeautifulSoup(markup, "html.parser")` | • Python的内置标准库<br>• 执行速度适中<br>• 文档容错能力强 | • Python 2.7.3、3.2.2前 的版本中文档容错能力差 |
| lxml HTML 解析器 | `BeautifulSoup(markup, "lxml")` | • 速度快<br>• 文档容错能力强 | • 需要安装C语言库 |
| lxml XML 解析器 | `BeautifulSoup(markup, ["lxml", "xml"])`<br>`BeautifulSoup(markup, "xml")` | • 速度快<br>• 唯一支持XML的解析器 | • 需要安装C语言库 |
| html5lib | `BeautifulSoup(markup, "html5lib")` | • 最好的容错性<br>• 以浏览器的方式解析文档<br>• 生成HTML5格式的文档 | • 速度慢<br>• 不依赖外部扩展 |

BeautifulSoup(markup, "html.parser") 使用Python标准库，容错差且性能一般。
BeautifulSoup(markup, "lxml") 容错能力强，速度快。需要安装系统C库。
推荐使用lxml作为解析器，效率高，再一个手动指定解析器，以保证代码在所有运行环境中解析器一致。

## 四种对象

BeautifulSoup将HTML文档解析成复杂的树型结构，每个节点都是Python的对象，可分为4种：
BeautifulSoup、Tag、NavigableString、Comment

### BeautifulSoup对象

BeautifulSoup对象代表整个文档。

### Tag对象

它对应着HTML中的标签。
有2个常用的属性：
name Tag对象的名称，就是标签名称
attrs 标签的属性字典
多值属性，对于class属性可能是下面的形式，`<h3 class="title highlight">python高级班</h3>`，这个属性就是多值。
属性可以被修改、删除。

使用下面内容构建test.html使用bs4解析它

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
<h1>马哥教育欢迎您</h1>
<div>
    <h3 class="title highlight">python高级班</h3>
    <div class="content">
        <p id='first'>字典</p>
        <p id='second'>列表</p>
        <input type="hidden" name="_csrf" value="7139e401481ef2f46ce98b22af4f4bed">
        <!-- comment -->
        <img id="bg1" src="http://www.magedu.com/">
        <img id="bg2" src="http://httpbin.org/">
    </div>
</div>
<p>bottom</p>
</body>
</html>
```

```
from bs4 import BeautifulSoup

with open('o:/test.html', encoding='utf-8') as f:
    soup = BeautifulSoup(f, 'lxml')
    print(soup.builder)
    print(0, soup)  # 输出整个解析的文档对象
    print(1, soup.prettify())  # 格式输出
    print('-'*30)
    print(2, soup.div, type(soup.div)) # bs4.element.Tag, Tag对象
    print(3, soup.div.name, soup.div.attrs)
    #print(3, soup.div['class']) # KeyError，div没有class属性
    print(3, soup.div.get('class'))
    print(4, soup.h3['class'])  # 多值属性
    print(4, soup.h3.get('class'))  # 多值属性
    print(4, soup.h3.attrs.get('class'))  # 多值属性
    print(5, soup.img.get('src'))
    soup.img['src'] = 'http://www.python.org/'  # 修改属性
    print(5, soup.img['src'])
```

```
print(6, soup.a) # 找不到返回None
del soup.h3['class'] # 删除属性
print(4, soup.h3.get('class'))
```

注意，我们一般不使用上面这种方式来操作HTML，此代码是为了熟悉对象类型

#### NavigableString

如果只想输出标记内的文本，而不关心标记的话，就要使用NavigableString。

```
print(soup.div.p.string) # 第一个div下第一个p的字符串
print(soup.p.string) # 同上
```

#### 注释对象

这就是HTML中的注释，它被BeautifulSoup解析后对应Comment对象。

---

## 遍历文档树

在文档树中找到关心的内容才是日常的工作，也就是说如何遍历树中的节点。使用上面的test.html来测试

**使用Tag**
soup.div 可以找到从根节点开始查找第一个div节点
soup.div.p 说明从根节点开始找到第一个div后返回一个Tag对象，这个Tag对象下继续找第一个p，找到返回Tag对象
soup.p 说明遍历是深度优先，返回了文字“字典”，而不是文字“bottom”。

**遍历直接子节点**
print(soup.div.contents) # 将对象的所有类型直接子节点以列表方式输出
print(soup.div.children) # 返回子节点的迭代器
print(list(soup.div.children)) # 等价于soup.div.contents

**遍历所有子孙节点**
print(list(soup.div.descendants)) # 返回第一个div节点的所有类型子孙节点，可以看出迭代次序是深度优先

**遍历字符串**
在前面的例子中，soup.div.string返回None，是因为string要求soup.div只能有一个NavigableString类型子节点，也就是如这样`<div>only string</div>`

如果div有很多子孙节点，如何提取字符串？
print(soup.div.string) # 返回None
print("".join(soup.div.strings)) # 返回迭代器，带多余的空白字符
print("".join(soup.div.stripped_strings)) # 去除多余空白符

**遍历祖先节点**
print(soup.parent) # None 根节点没有父节点
print(soup.div.parent.name) # body，第一个div的父节点
print(soup.p.parent.parent.get('id')) # main
print(list(map(lambda x: x.name, soup.p.parents))) # 父迭代器，由近及远

**遍历兄弟节点**
print('{} [{}]'.format(1, soup.p.next_sibling)) # 第一个p元素的下一个兄弟节点，注意可能是一个文本节点
print('{} [{}]'.format(2, soup.p.previous_sibling))
print(list(soup.p.next_siblings)) # previous_siblings

遍历其他元素
next_element是下一个可被解析的对象（字符串或tag），和下一个兄弟节点next_sibling不一样
print(soup.p.next_element) # 返回"字典"2个字
print(soup.p.next_element.next_element.next_element)
print(list(soup.p.next_elements))

---

## 搜索文档树

find系有很多方法，请自行查帮助

`find_all(name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)`
find_all立即返回一个列表

### name

官方称为**filter过滤器**，这个参数可以是以下类型：

1. 字符串
   一个标签名称的字符串，会按照这个字符串全长匹配标签名
   print(soup.find_all('p')) # 返回文档中所有p标签
2. 正则表达式对象
   按照"正则表达式对象"的模式匹配标签名
   import re
   print(soup.find_all(re.compile('^h\d'))) # 标签名以h开头后接数字
3. 列表
   print(soup.find_all(['p', 'h1', 'h3'])) # 或，找出列表所有的标签
   print(soup.find_all(re.compile(r'^(p|h\d)$'))) # 使用正则完成
4. True或None
   True或None，则find_all返回全部非字符串节点、非注释节点，即Tag标签类型
   print(list(map(lambda x:x.name, soup.find_all(True))))
   print(list(map(lambda x:x.name, soup.find_all(None))))
   print(list(map(lambda x:x.name, soup.find_all())))

```
from bs4 import BeautifulSoup
from bs4.element import Tag

with open('o:/test.html', encoding='utf-8') as f:
    soup = BeautifulSoup(f, 'lxml')
    values = [True, None, False]
    for value in values:
        all = soup.find_all(value)
        print(len(all))

    print('-'*30)
    count = 0
```

```
    for i,t in enumerate(soup.descendants):
        print(i,type(t), t.name)
        if isinstance(t, Tag):
            count += 1
    print(count)
# 数目一致，所以返回的是Tag类型的节点，源码中确实返回的Tag类型
```

5. 函数
   如果使用以上过滤器还不能提取出想要的节点，可以使用函数，此函数仅只能接收一个参数。
   如果这个函数返回True，表示当前节点匹配；返回False则是不匹配。
   例如，找出所有有class属性且有多个值的节点，符合这个要求只有h3标签

```
from bs4 import BeautifulSoup

def many_class(tag):
    # print(type(tag))
    # print(tag.attrs)
    return len(tag.attrs.get('class', [])) > 1

with open('o:/test.html', encoding='utf-8') as f:
    soup = BeautifulSoup(f, 'lxml')
    print(soup.find_all(many_class))
```

**keyword传参**

使用关键字传参，如果参数名不是已定义的位置参数名，参数会被kwargs收集并被**当做标签的属性**来搜索。
属性的传参可以是字符串、正则表达式对象、True、列表。
print(soup.find_all(id='first')) # id为first的所有节点列表
print(soup.find_all(id=re.compile('\w+'))) # 相当于找有id的所有节点
print(soup.find_all(id=True)) # 所有有id的节点
print(list(map(lambda x:x['id'], soup.find_all(id=True))))
print(soup.find_all(id=['first', 'second'])) # 指定id的名称列表
print(soup.find_all(id=True, src=True)) # 相当于条件and，既有id又有src属性的节点列表

**css的class的特殊处理**

class是Python关键字，所以使用`class_`。class是多值属性，可以匹配其中任意一个，也可以完全匹配。
print(soup.find_all(class_="content"))
print(soup.find_all(class_="title")) # 可以使用任意一个css类
print(soup.find_all(class_="highlight")) # 可以使用任意一个css类
print(soup.find_all(class_="highlight title")) # 顺序错了，找不到
print(soup.find_all(class_="title highlight")) # 顺序一致，找到，就是字符串完全匹配

**attrs参数**

attrs接收一个字典，字典的key为属性名，value可以是字符串、正则表达式对象、True、列表
print(soup.find_all(attrs={'class':'title'}))
print(soup.find_all(attrs={'class':'highlight'}))
print(soup.find_all(attrs={'class':'title highlight'}))
print(soup.find_all(attrs={'id':True}))
print(soup.find_all(attrs={'id':re.compile(r'\d$')}))

**text参数**

可以通过text参数搜索文档中的字符串内容，接受字符串、正则表达式对象、True、列表
print(list(map(lambda x: (type(x), x), soup.find_all(text=re.compile('\w+')))))
print(list(map(lambda x: (type(x), x), soup.find_all(text=re.compile('[a-z]+')))))
print(soup.find_all(re.compile(r'h|p'), text=re.compile('[a-z]+')))# 相当于过滤出Tag对象，并看它的string是否符合text参数的要求

**limit参数**

限制返回结果的数量
print(soup.find_all(id=True, limit=3)) # 返回列表中有3个结果

**recursive 参数**

默认是递归搜索所有子孙节点，如果不需要请设置为False

**简化写法**

find_all()是非常常用的方法，可以简化省略掉
soup.find_all("a")
soup("a") # 注意不等价于soup.a
soup.a.find_all(text=True)
soup.a(text=True)
print(soup.find_all('img', attrs={'id':'bg1'}))
print(soup('img', attrs={'id':'bg1'})) # find_all的省略
print(soup('img', attrs={'id':'bg1'}))

**find方法**

`find( name , attrs , recursive , text , **kwargs )`
参数几乎和find_all一样。
找到了，find_all返回一个列表，而find返回一个单值，元素对象。
找不到，find_all返回一个空列表，而find返回一个None。

print(soup.find('img', attrs={'id':'bg1'}).attrs.get('src', 'magedu'))
print(soup.find('img', attrs={'id':'bg1'}).get('src')) # 简化了attrs
print(soup.find('img', attrs={'id':'bg1'})['src'])

## CSS选择器 ***

和JQuery一样，可以使用CSS选择器来查找节点

使用soup.select()方法，select方法支持大部分CSS选择器，返回列表。
CSS中，标签名直接使用，类名前加.点号，id名前加#井号。

```
from bs4 import BeautifulSoup

with open('o:/test.html', encoding='utf-8') as f:
    soup = BeautifulSoup(f, 'lxml')
    # 元素选择器
    print(1, soup.select('p'))  # 所有的p标签

    # 类选择器
```

```
    print(2, soup.select('.title'))
    # 使用了伪类
    print(3, soup.select('div.content > p:nth-of-type(2)'))  # 同标签名p的第2个，伪类只实现了nth-of-type，且要求是数字

    # id选择器
    print(4, soup.select('p#second'))
    print(5, soup.select('#bg1'))

    # 后代选择器
    print(6, soup.select('div p'))  # div下逐层找p
    print(7, soup.select('div div p'))  # div下逐层找div下逐层找p

    # 子选择器，直接后代
    print(8, soup.select('div > p'))  # div下直接子标签的p

    # 相邻兄弟选择器
    print(9, soup.select('div p:nth-of-type(1) + [src]'))  # 返回[]

    # 普通兄弟选择器
    print(10, soup.select('div p:nth-of-type(1) ~ [src]'))

    # 属性选择器
    print(11, soup.select('[src]'))  # 有属性src
    print(12, soup.select('[src="/"]'))  # 属性src等于/
    print(13, soup.select('[src="http://www.magedu.com/"]'))  # 完全匹配
    print(14, soup.select('[src^="http://www"]'))  # 以http://www开头
    print(15, soup.select('[src$="com/"]'))  # 以com/结尾
    print(16, soup.select('img[src*="magedu"]'))  # 包含magedu
    print(17, soup.select('img[src*=".com"]'))  # 包含.com
    print(18, soup.select('[class~=title]'))  # 多值属性中有一个title
```

## 获取文本内容

搜索节点的目的往往是为了提取该节点的文本内容，一般不需要HTML标记，只需要文字

```
from bs4 import BeautifulSoup

with open('o:/test.html', encoding='utf-8') as f:
    soup = BeautifulSoup(f, 'lxml')
    # 元素选择器
    ele = soup.select('div')  # 所有的div标签

    print(ele[0].string, end='\n------------\n')  # 内容仅仅只能是文本类型，否则返回None
    print(list(ele[0].strings), end='\n------------\n')  # 迭代保留空白字符
    print(list(ele[0].stripped_strings), end='\n------------\n')  # 迭代不保留空白字符
    print(ele[0], end='\n------------\n')
    print(ele[0].text, end='\n------------\n')  # 本质上就是get_text()，保留空白字符的strings
    print(ele[0].get_text(), end='\n------------\n')  # 迭代并join，保留空白字符，strip默认为False
    print(ele[0].get_text(strip=True))  # 迭代并join，不保留空白字符
```

# Json解析

拿到一个Json字符串，如果想提取其中的部分内容，就需要遍历了。在遍历过程中进行判断。

还有一种方式，类似于XPath，叫做JsonPath。

安装

```
$ pip install jsonpath
```

官网 http://goessner.net/articles/JsonPath/

| XPath | JsonPath | 说明 |
| --- | --- | --- |
| / | $ | 根元素 |
| . | @ | 当前节点 |
| / | . or [ ] | 获取子节点 |
| .. | 不支持 | 父节点 |
| // | .. | 任意层次 |
| * | * | 通配符，匹配任意节点 |
| @ | 不支持 | Json中没有属性 |
| [ ] | [ ] | 下标操作 |
| \| | [,] | XPath是或操作. JSONPath allows alternate names or array indices as a set. |
| 不支持 | [start:end:step] | 切片 |
| [ ] | ?( ) | 过滤操作 |
| 不支持 | ( ) | 表达式计算 |
| ( ) | 不支持 | 分组 |

依然用豆瓣电影的热门电影的Json
https://movie.douban.com/j/search_subjects?type=movie&tag=%E7%83%AD%E9%97%A8&page_limit=10&page_start=0
找到得分高于8分的

```
{
    subjects:
    [
        {
            rate: "8.3",
            cover_x: 2309,
            title: "爱你，西蒙",
            url: "https://movie.douban.com/subject/26654498/",
```

```
            playable: false,
            cover:
"https://img1.doubanio.com/view/photo/s_ratio_poster/public/p2523592367.webp",
            id: "26654498",
            cover_y: 3464,
            is_new: false
        },
        {
            rate: "8.3",
            cover_x: 3578,
            title: "暴裂无声",
            url: "https://movie.douban.com/subject/26647117/",
            playable: true,
            cover:
"https://img3.doubanio.com/view/photo/s_ratio_poster/public/p2517333671.webp",
            id: "26647117",
            cover_y: 5078,
            is_new: false
        }
    ]
}
```

思路
找到title非常容易，但是要用其兄弟节点rate判断是否大于8分，就不好做了。
能够从父节点下手，subjects的多个子节点中，要用[ ]，某一个当前节点的rate和字符串8比较来过滤的得到符合要求的subjects的子节点，取这个子节点的title。

```
# 返回json的解析和处理
from jsonpath import jsonpath
import requests
import json

ua = "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.75
Safari/537.36"
url = 'https://movie.douban.com/j/search_subjects?
type=movie&tag=%E7%83%AD%E9%97%A8&page_limit=10&page_start=0'

with requests.get(url, headers={'User-agent':ua}) as response:
    text = response.text
    print(text) # str类型的json数据
    js = json.loads(text)
    print(js) # Json转为Python数据结构

    # 找到所有电影的名称
    rs1 = jsonpath(js, '$..title')
    print(rs1)

    # 找打所有得分高于8分的电影名称
    # 根下任意层的subjects的子节点rate大于字符串8
    rs2 = jsonpath(js, '$..subjects[?(@.rate > "8")]')
    print(rs2)
    # 根下任意层的subjects的子节点rate大于字符串8的节点的子节点title
```

```
    rs3 = jsonpath(js, '$..subjects[?(@.rate > "8")].title')
    print(rs3)
```