# 课程介绍

* 数据的分类与应对工具（了解）
* 用正则表达式处理HTML（掌握）
* 用XPath处理XML（掌握）
* 用XPath处理HTML（重点掌握）
* 用JsonPath库处理JSON（掌握）
* 用BeautifulSoup4处理HTML（扩展）

1. 数据提取简介

一般来讲对我们而言，需要抓取的是某个网站或者某个应用的内容，提取有用的部分信息。抓取到的内容一般分为两类：非结构化的数据和结构化的数据。

* + - 1. 非结构化数据：先有数据，再有结构。
      2. 结构化数据：先有结构、再有数据。

不同类型的数据，我们需要采用不同的方式来处理。

* 1. 分类
     1. 非结构化数据的处理

**文本、电话号码、邮箱地址**

* 正则表达式
  + 1. 弱结构化数据

**HTML 文件**

* 正则表达式
* XPath
* CSS选择器
  + 1. 结构化数据的处理

**JSON 文件**

* 转化成Python类型进行操作（json类）
* JSON Path

**XML 文件**

* 转化成Python类型（xmltodict）
* XPath
* CSS选择器
* 正则表达式

1. XML数据提取
   1. XML

应用场景：比如 天气数据XML

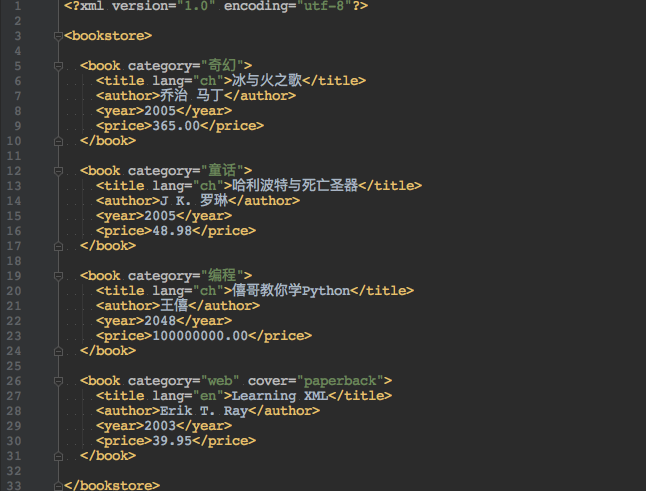
XML（EXtensible Markup Language，可扩展标记语言）：

* 一种标记语言，很类似 HTML
* 设计宗旨是传输数据，而非显示数据
* 标签需要自行定义。
* 被设计为具有自我描述性。
* 是W3C的推荐标准

W3School官方文档：<http://www.w3school.com.cn/xml/index.asp>

* + 1. 案例 XML实例文档

XPath (XML Path Language) 是一门在 XML 文档中查找信息的语言，可用来在 XML 文档中对元素和属性进行遍历。



用正则表达式对xml解析是把xml当做无结构的纯文本解析的，而实际上xml是有结构的，利用xml的结构能更有效地解析。那么，问题来了，xml有哪些结构特点呢？

* + 1. XML的节点关系
       1. 父（Parent）

每个元素以及属性都有一个父。

上例每个book元素中，book都是 title、author、year 以及 price 元素的父。

* + - 1. 子（Children）

元素节点可有零个、一个或多个子。

上例每个book元素中，title、author、year 以及 price 元素都是 book 元素的子：

* + - 1. 同胞（Sibling）

拥有相同的父的节点

上例每个book元素中，title、author、year 以及 price 元素都是同胞：

* + - 1. 祖先（Ancestor）

某节点的父、父的父，等等。

上例中，title 元素的先辈是 book 元素和 bookstore 元素：

* + - 1. 5. 后代（Descendant）

某个节点的子，子的子，等等。

上例中，bookstore 的后代是 book、title、author、year 以及 price 元素：

只要是XML，都存在同样的结构化特征，所以我们可以使用XPath来解析。

* 1. 使用XPath

XPath (XML Path Language) 是一门在 XML 文档中查找信息的语言，可用来在 XML 文档中对元素和属性进行遍历。

W3School官方文档：<http://www.w3school.com.cn/xpath/index.asp>

* + 1. 选取节点

XPath语法元素比较多，但是很简单，我们不可能一一演示，这里给大家按文档介绍，大家应该都能理解。

XPath使用路径表达式来选取XML文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

下面列出了最常用的路径表达式：

| **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 |

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。 |

* + 1. 谓词（Predicates）

谓词用来查找某个特定的节点或者包含某个指定的值的节点，被嵌在方括号中。

在下面的表格中，我们列出了带有谓词的一些路径表达式，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=’en’] | 选取所有 title 元素，且这些元素拥有值为 en 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。 |

* + 1. 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。 |

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。 |

* + 1. 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。

在下面的表格中，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。 |

* 1. lxml库

lxml 是 一个HTML/XML的解析器，主要的功能是如何解析和提取 HTML/XML 数据。

lxml和正则一样，也是用 C 实现的，是一款高性能的 Python HTML/XML 解析器，我们可以利用之前学习的XPath语法，来快速的定位特定元素以及节点信息。

lxml python 官方文档：<http://lxml.de/index.html>

需要安装C语言库，可使用 pip 安装：pip install lxml （或通过wheel方式安装）

* + 1. 核心方法常用函数
       1. etree.XML(…)

解析字节字符串形式的XML内容，产生树形结构。

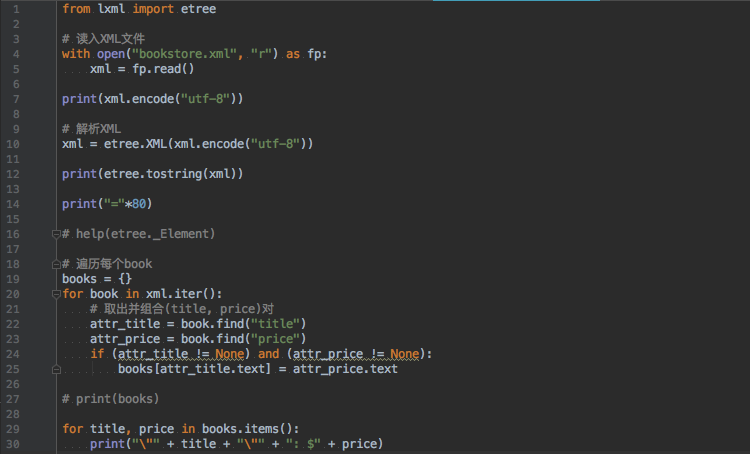
* + - 1. etree.toString(…)

产生XML对应的字符串。

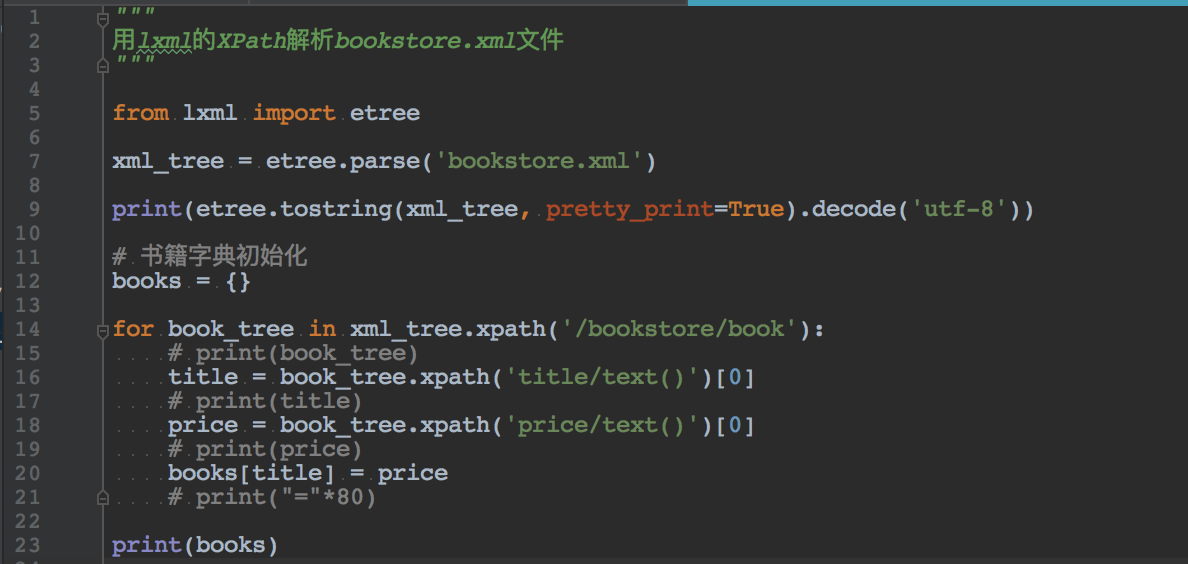
* + - 1. etree.xpath()

使用XPath选取指定内容。

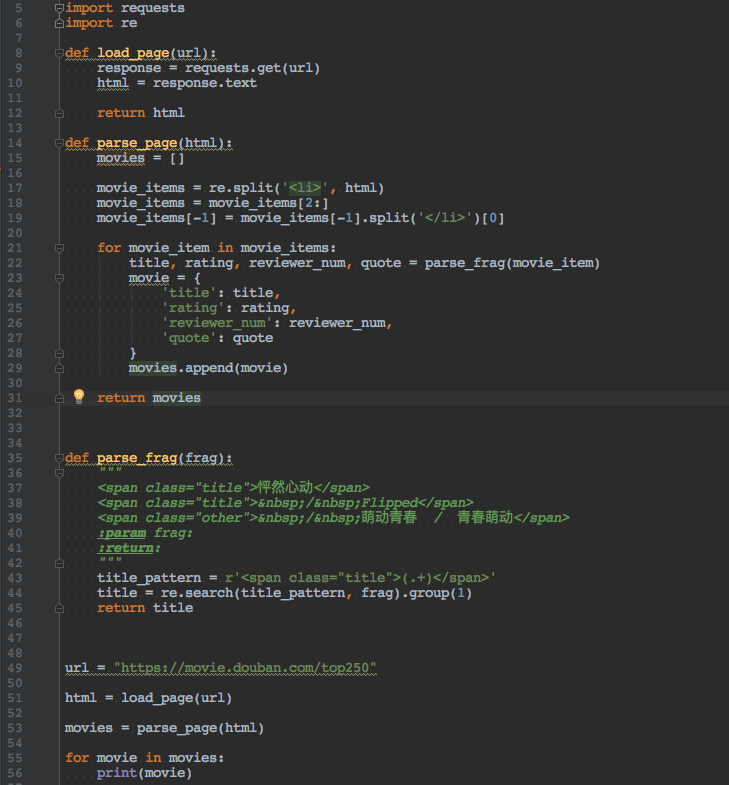
* + 1. 案例 解析bookstore.xml文件，提取出每本书的书名和对应价格



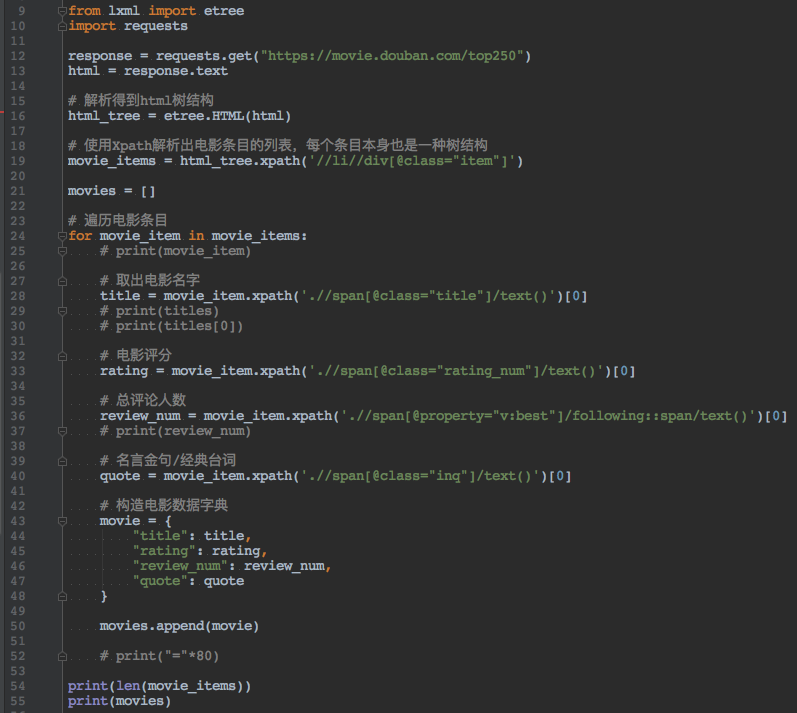
* + 1. 案例 使用XPath解析bookstore.xml文件，提取出每本书的书名和对应价格



1. HTML数据提取
   1. 使用正则表达式提取
      1. 案例 使用正则表达式提取豆瓣电影数据



* 1. 使用XPath提取
     1. 案例 使用XPath提取豆瓣电影数据



1. JSON数据提取

JSON(JavaScript Object Notation，JavaScript对象表示法)是一种轻量级的数据交换格式，它使得人们很容易的进行阅读和编写。同时也方便了机器进行解析和生成。适用于进行数据交互的场景，比如网站前台与后台之间的数据交互。

Python 3中自带了JSON模块，直接import json就可以使用了。

官方文档：<http://docs.python.org/library/json.html>

Json在线解析网站：[http://www.json.cn/#](http://www.json.cn/)

* 1. JSON

JSON有四种元素类型：数字、字符串、对象和数组，其中，对象和数组是仅有的两种结构化类型，但通过这两种结构可以组合出各种复杂的结构。

1. 对象：对象在js中表示为{ }括起来的内容，数据结构为{key：value, key：value, ... }的键值对的结构，在面向对象的语言中，key为对象的属性，value为对应的属性值，所以很容易理解，取值方法为 对象.key获取属性值，这个属性值的类型可以是数字、字符串、数组、对象这几种。
2. 数组：数组在js中是中括号[ ]括起来以逗号分隔的内容，，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是 数字、字符串、数组、对象几种。

例如["Python", "javascript", "C++", ...]

* 1. json模块

json模块提供了四个功能：dumps、dump、loads、load，用于字符串和python数据类型间进行转换。

注意，在python3中，是这样做编码的，内存中字符串采用unicode，存储到文件中采用utf-8。

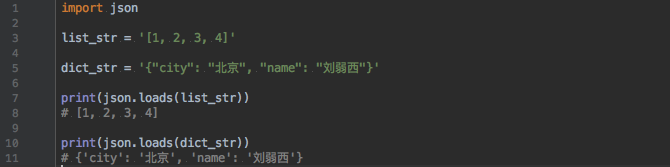
* + 1. json.loads()

创建一个对象的办法：把Json格式字符串解码转换成Python对象 从json到python的类型转化对照如下：

**JSON 解码为 Python 类型转换对应表：**

| **JSON** | **Python** |
| --- | --- |
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

**案例2 json.loads使用**



* + 1. json.dumps()

实现python类型转化为json字符串，返回一个str对象 把一个Python对象编码转换成Json字符串。

从python原始类型向json类型的转化对照如下：

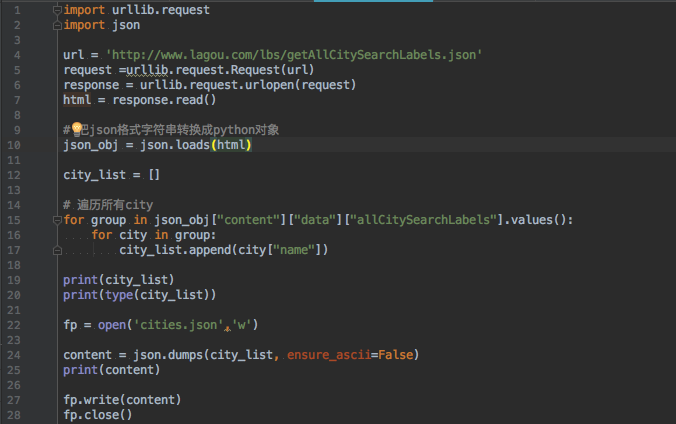
* + 1. json.dump()

将Python内置类型序列化为json对象后写入文件

* + 1. json.load()

读取文件中json形式的字符串元素 转化成python类型

* + 1. 案例 使用json模块从<http://www.lagou.com/lbs/getAllCitySearchLabels.json>获取城市列表



* 1. JsonPath库

JsonPath 是一种信息抽取类库，是从JSON文档中抽取指定信息的工具，提供多种语言实现版本，包括：Javascript, Python， PHP 和 Java。

JsonPath 对于 JSON 来说，相当于 XPATH 对于 XML。

下载地址：[https://pypi.python.org/pypi/jsonpath](https://pypi.python.org/pypi/jsonpath/)

安装方法：点击Download URL链接下载jsonpath，解压之后执行python setup.py install

官方文档：[http://goessner.net/articles/JsonPath](http://goessner.net/articles/JsonPath/)

* + 1. JsonPath与XPath语法对比

Json结构清晰，可读性高，复杂度低，非常容易匹配，下表中对应了XPath的用法。

| **XPath** | **JSONPath** | **描述** |
| --- | --- | --- |
| / | $ | 根节点 |
| . | @ | 现行节点 |
| / | .或[] | 取子节点 |
| .. | n/a | 取父节点，Jsonpath未支持 |
| // | .. | 就是不管位置，选择所有符合条件的条件 |
| \* | \* | 匹配所有元素节点 |
| @ | n/a | 根据属性访问，Json不支持，因为Json是个Key-value递归结构，不需要。 |
| [] | [] | 迭代器标示（可以在里边做简单的迭代操作，如数组下标，根据内容选值等） |
| | | [,] | 支持迭代器中做多选。 |
| [] | ?() | 支持过滤操作. |
| n/a | () | 支持表达式计算 |
| () | n/a | 分组，JsonPath不支持 |

* + 1. 案例 使用jsonpath从<http://www.lagou.com/lbs/getAllCitySearchLabels.json>获取城市列表



* 1. 编码问题（扩展）

json.loads() 是把 Json格式字符串解码转换成Python对象，如果在json.loads的时候出错，要注意被解码的Json字符的编码。

如果传入的字符串的编码不是UTF-8的话，需要指定字符编码的参数 encoding

dataDict = json.loads(jsonStrGBK);

* dataJsonStr是JSON字符串，假设其编码本身是非UTF-8的话而是GBK 的，那么上述代码会导致出错，改为对应的：
* dataDict = json.loads(jsonStrGBK, encoding="GBK");
* 如果 dataJsonStr通过encoding指定了合适的编码，但是其中又包含了其他编码的字符，则需要先去将dataJsonStr转换为Unicode，然后再指定编码格式调用json.loads()

dataJsonStrUni = dataJsonStr.decode("GB2312"); dataDict = json.loads(dataJsonStrUni, encoding="GB2312");

* + 1. 字符串编码转换

这是中国程序员最苦逼的地方，什么乱码之类的几乎都是由汉字引起的。

解决编码问题的关键点：

**任何平台的任何编码 都能和 Unicode 互相转换**

例如：UTF-8 与 GBK 互相转换，那就先把UTF-8转换成Unicode，再从Unicode转换成GBK，反之同理。

# 这是一个 UTF-8 编码的字符串

utf8Str = "你好地球"

# 1. 将 UTF-8 编码的字符串 转换成 Unicode 编码

unicodeStr = utf8Str.decode("UTF-8")

# 2. 再将 Unicode 编码格式字符串 转换成 GBK 编码

gbkData = unicodeStr.encode("GBK")

# 3. 再将 GBK 编码格式字符串 转化成 Unicode

unicodeStr = gbkData.decode("gbk")

# 4. 再将 Unicode 编码格式字符串转换成 UTF-8

utf8Str = unicodeStr.encode("UTF-8")

decode和encode的作用：

* + - 1. decode：将其他编码的字符串转换成 Unicode 编码
      2. encode：将 Unicode 编码转换成其他编码的字符串

一句话：UTF-8是对Unicode字符集进行编码的一种编码方式

* + 1. 常用编码格式超短简介：
       1. ascii: 定长，1字节
       2. GB2312：支持6000多个常用汉字
       3. GBK：GB2312的扩展
       4. GB18030：GBK的扩展，支持藏文、蒙文等
       5. Unicode: 定长，2字节
       6. utf-8: 变长，1-4字节，

1. 扩展知识
   1. 使用BeautifulSoup4框架
      1. BeautifulSoup4简介

Beautiful Soup是XPath以外另一个HTML/XML解析器，主要的功能当然也是解析和提取 HTML/XML 数据。

lxml只做局部遍历，而Beautiful Soup是基于HTML DOM，会载入整个文档，解析整个DOM树，因此时间和内存开销都会大很多，性能低于lxml。

BeautifulSoup用来解析 HTML 比较简单，API非常人性化，支持CSS选择器、Python标准库中的HTML解析器，也支持 lxml 的 XML解析器。

* + 1. 安装

使用 pip 安装：

pip install beautifulsoup4

官方文档：[http://beautifulsoup.readthedocs.io/zh\_CN/v4.4.0](http://beautifulsoup.readthedocs.io/zh_CN/v4.4.0/)

* + 1. CSS选择器

这就是一种有时候比XPath使用起来更快捷的查找方法.

* 写 CSS 时，标签名不加任何修饰，类名前加.，id名前加#
* 在这里我们也可以利用类似的方法来筛选元素，用到的方法是 soup.select()，返回类型是 list
  + - 1. 通过标签名查找



* + - 1. 通过类名查找



* + - 1. 通过 id 名查找



* + - 1. 组合查找

组合查找即和写 class 文件时，标签名与类名、id名进行的组合原理是一样的，例如查找 p 标签中，id 等于 link1的内容，二者需要用空格分开



直接子标签查找，则使用 > 分隔



* + - 1. 属性查找

查找时还可以加入属性元素，属性需要用中括号括起来，注意属性和标签属于同一节点，所以中间不能加空格，否则会无法匹配到。

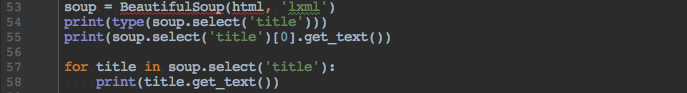


同样，属性仍然可以与上述查找方式组合，不在同一节点的空格隔开，同一节点的不加空格



* + - 1. 获取内容

以上的 select 方法返回的结果都是列表形式，可以遍历形式输出，然后用 get\_text() 方法来获取它的内容。



* + 1. 四大对象种类

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种:

* Tag
* NavigableString
* BeautifulSoup
* Comment
  + - 1. **Tag**

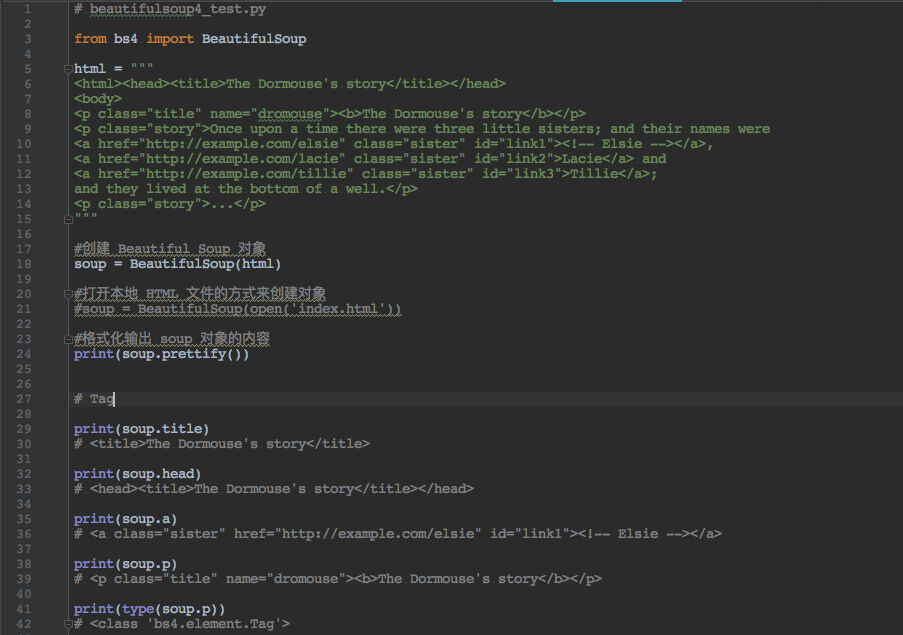
Tag 通俗点讲就是 HTML 中的一个个标签，例如：

<head><title>The Dormouse's story</title></head>

<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>

<p class="title" name="dromouse"><b>The Dormouse's story</b></p>

上面的 title head a p等等 HTML 标签加上里面包括的内容就是 Tag，那么试着使用 Beautiful Soup 来获取 Tags:



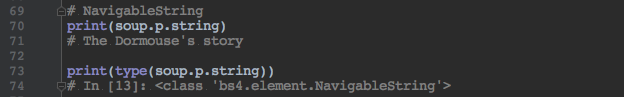
我们可以利用 soup 加标签名轻松地获取这些标签的内容，这些对象的类型是bs4.element.Tag。但是注意，它查找的是在所有内容中的第一个符合要求的标签。如果要查询所有的标签，后面会进行介绍。

**对于 Tag，它有两个重要的属性，是 name 和 attrs**

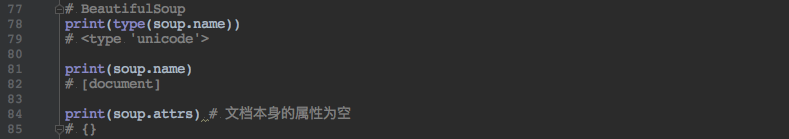


* + - 1. **NavigableString**

既然我们已经得到了标签的内容，那么问题来了，我们要想获取标签内部的文字怎么办呢？很简单，用 .string 即可，例如

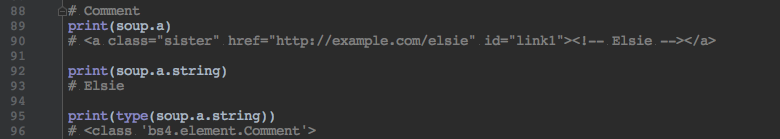


* + - 1. BeautifulSoup



* + - 1. **Comment**

Comment 对象是一个特殊类型的 NavigableString 对象，其输出的内容不包括注释符号。



a 标签里的内容实际上是注释，但是如果我们利用 .string 来输出它的内容时，注释符号已经去掉了。

* + 1. 遍历文档树
       1. 直接子节点 ：.contents .children 属性

**.content**

tag 的 .content 属性可以将tag的子节点以列表的方式输出



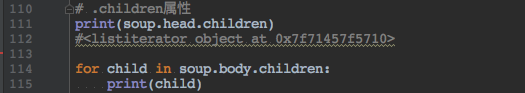
输出方式为列表，我们可以用列表索引来获取它的某一个元素



**.children**

它返回的不是一个 list，不过我们可以通过遍历获取所有子节点。

我们打印输出 .children 看一下，可以发现它是一个 list 生成器对象



结果:

<p class="title" name="dromouse"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were

<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>,

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;

and they lived at the bottom of a well.</p>

<p class="story">...</p>

* + - 1. 所有子孙节点: .descendants 属性

.contents 和 .children 属性仅包含tag的直接子节点，.descendants 属性可以对所有tag的子孙节点进行递归循环，和 children类似，我们也需要遍历获取其中的内容。



运行结果：

<html><head><title>The Dormouse's story</title></head>

<body>

<p class="title" name="dromouse"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were

<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>,

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;

and they lived at the bottom of a well.</p>

<p class="story">...</p>

</body></html>

<head><title>The Dormouse's story</title></head>

<title>The Dormouse's story</title>

The Dormouse's story

<body>

<p class="title" name="dromouse"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were

<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>,

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;

and they lived at the bottom of a well.</p>

<p class="story">...</p>

</body>

<p class="title" name="dromouse"><b>The Dormouse's story</b></p>

<b>The Dormouse's story</b>

The Dormouse's story

<p class="story">Once upon a time there were three little sisters; and their names were

<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>,

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;

and they lived at the bottom of a well.</p>

Once upon a time there were three little sisters; and their names were

<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>

Elsie

,

<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>

Lacie

and

<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>

Tillie

;

and they lived at the bottom of a well.

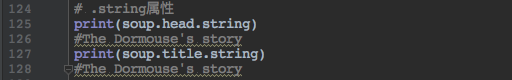
<p class="story">...</p>

...

* + - 1. 节点内容: .string 属性

如果tag只有一个 NavigableString 类型子节点,那么这个tag可以使用 .string 得到子节点。如果一个tag仅有一个子节点,那么这个tag也可以使用 .string 方法,输出结果与当前唯一子节点的 .string 结果相同。

通俗点说就是：如果一个标签里面没有标签了，那么 .string 就会返回标签里面的内容。如果标签里面只有唯一的一个标签了，那么 .string 也会返回最里面的内容。例如：



* + 1. 搜索文档树
       1. find\_all(name, attrs, recursive, text, \*\*kwargs)

1）name 参数

name 参数可以查找所有名字为 name 的tag,字符串对象会被自动忽略掉

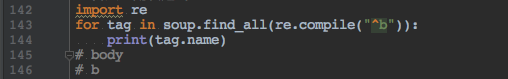
A.传字符串

最简单的过滤器是字符串.在搜索方法中传入一个字符串参数,Beautiful Soup会查找与字符串完整匹配的内容,下面的例子用于查找文档中所有的<b>标签:



B.传正则表达式

如果传入正则表达式作为参数,Beautiful Soup会通过正则表达式的 match() 来匹配内容.下面例子中找出所有以b开头的标签,这表示<body>和<b>标签都应该被找到



C.传列表

如果传入列表参数,Beautiful Soup会将与列表中任一元素匹配的内容返回.下面代码找到文档中所有<a>标签和<b>标签:

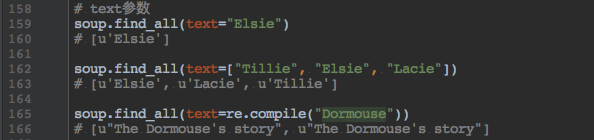


2）keyword 参数

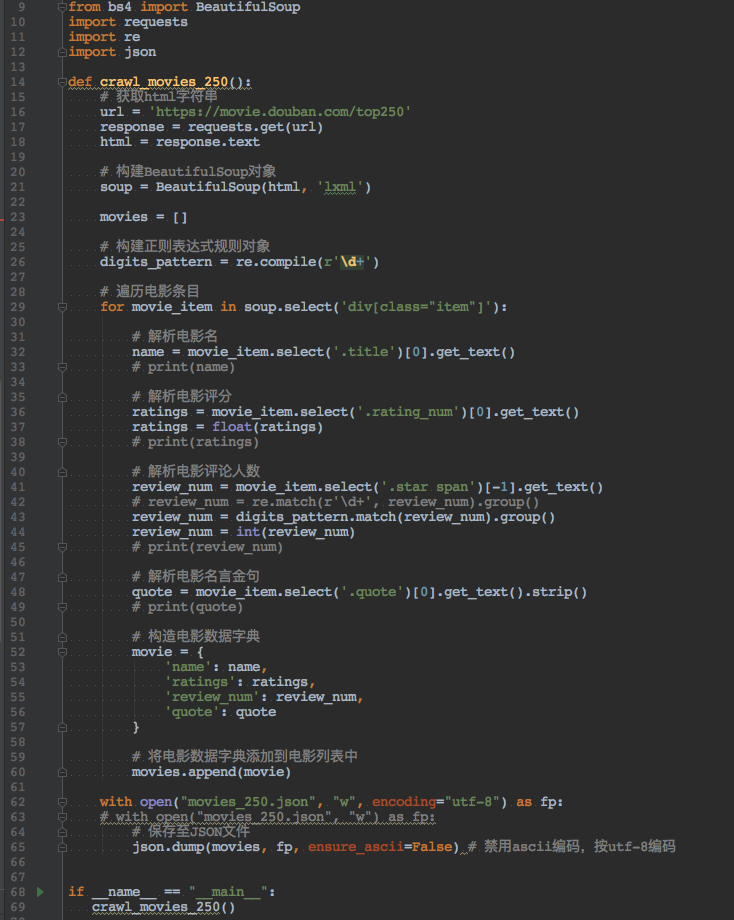


3）text 参数

通过 text 参数可以搜搜文档中的字符串内容，与 name 参数的可选值一样, text 参数接受 字符串 , 正则表达式 , 列表



* + 1. 案例 使用beautifulsoup4提取豆瓣电影数据



执行时，可能会看到这样一段警告：

意思是，如果我们没有显式地指定解析器，所以默认使用这个系统的最佳可用HTML解析器(“lxml”)。如果你在另一个系统中运行这段代码，或者在不同的虚拟环境中，使用不同的解析器造成行为不同。

但是我们可以通过soup = BeautifulSoup(html,“lxml”)方式指定lxml解析器。

1. 课程总结
   1. 重点
      * 1. json模块的使用
        2. jsonPath的使用
        3. XPath和lxml库的使用
        4. CSS选择器和BeautifulSoup4库的使用
   2. 难点
      * 1. XPath语言的掌握
        2. CSS选择器的掌握
   3. 待抄写理论

无

1. 课后练习
   1. 必做题
      * 1. 实现上课代码，尽量领会后独立实现，包括自己组织语言做简要的注释
        2. 在爬取到豆瓣Top250电影条目中，解析出电影详细页面的url，爬取电影详细页面，解析出导演、演员列表等内容，和电影名、评分、评论人数、名言金句等存为JSON文件,爬取到海报图片。
   2. 选做题
      * 1. 从键盘输入出发地和目的地，爬取12306网站上当天的余票信息，将每条车次信息组织成字典，然后保存至JSON文件中。(获取城市名对应的城市代码是一个难点；而应对反爬虫机制是另一个难点)
        2. 解析python工程下的workspace.xml。
2. 面试题
   1. 常见的反爬虫应对机制有哪些？
   2. JSON格式有哪些类型？
3. 每日一练

复习多线程的相关内容。