

黑五电商学院- 爬虫课件

书栈(BookStack.CN)

目 录

致谢

一、爬虫原理与数据抓取

- 1、通用爬虫和聚焦爬虫
- 2、HTTP/HTTPS的请求与响应
- 3、Requests模块
- 4、Chrome浏览器使用方法介绍

二、数据提取方法

- 4、案例:使用bs4的爬虫
- 5、JSON模块与JsonPATH
- 2、XPath与lxml类库
- 3、BeautifulSoup4 解析器
- 1、正则表达式模块

三、redis数据库

- 1、redis的安装
- 2、Python连接redis数据库
- 3、Python操作redis数据库

四、动态HTML处理

- 1、动态HTML介绍
- 2、Selenium与PhantomJS
- 3、案例一：模拟登陆亚马逊
- 4、案例二、动态页面模拟点击
- 5、案例三：执行JavaScript语句

致谢

当前文档《黑五电商学院-爬虫课件》由黑五电商学院使用书栈(BookStack.CN)进行构建，生成于 2018-10-30。

书栈(BookStack.CN)仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN)难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到书栈(BookStack.CN)，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到书栈(BookStack.CN)获取最新的文档，以跟上知识更新换代的步伐。

文档地址：<http://www.bookstack.cn/books/PythonSpider>

书栈官网：<http://www.bookstack.cn>

书栈开源：<https://github.com/TruthHun>

分享，让知识传承更久远！感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

课程背景

我们生活在一个充满数据的时代。

每天，来自商业、社会以及我们的日常生活所产生「图像、音频、视频、文本、定位信息」等各种各样的海量数据，注入到我们的万维网（WWW）、计算机和各种数据存储设备，其中万维网则是最大的信息载体。

数据的爆炸式增长、规模庞大和广泛可用的数据，使得我们真正进入到了“大数据（Big Data）时代”。我们急需功能强大的数据处理技术（Data Technology），从这些海量数据中发现有价值的信息。

网络爬虫（Web Crawler）技术，则成为了当下万维网数据（Web Data）收集中，最为高效灵活的解决方案。

什么是网络爬虫？

百度百科：[网络爬虫](#)

获取数据方式主要有哪些？

- 企业产生的数据：[百度搜索指数](#)、[腾讯公司业绩数据](#)、[阿里巴巴集团财务及运营数据](#)、[新浪微博微指数](#) 等...

大型互联网公司拥有海量用户，有天然的数据积累优势，还有一些有数据意识的中小型企业，也开始积累自己的数据。

- 数据平台购买数据：[数据堂](#)、[国云数据市场](#)、[贵阳大数据交易所](#) 等...

在各个数据交易平台上购买各行各业各种类型的数据，根据数据信息、获取难易程度的不同，价格也会有所不同。

- 政府/机构公开的数据：[中华人民共和国国家统计局数据](#)、[中国人民银行调查统计](#)、[世界银行公开数据](#)、[联合国数据](#)、[纳斯达克](#)、[新浪财经美股实时行情](#) 等...

通常都是各地统计上报，或者是行业内专业的网站、机构等提供。

- 数据管理咨询公司：[麦肯锡](#)、[埃森哲](#)、[尼尔森](#)、[中国互联网络信息中心](#)、[艾瑞咨询](#) 等...

通常这样的公司有很庞大的数据团队，一般通过市场调研、问卷调查、固定的样本检测、与各行各业的其他公司合作、专家对话来获取数据，并根据客户需求制定商业解决方案。

- 爬虫爬取网络数据：

如果数据市场上没有需要的数据，或者价格太高不愿意购买，那么可以利用爬虫技术，抓取网站上的数据。

我们需要学习的是：

1. Python基础语法学习（基础知识）
2. 对HTML页面的内容抓取（Crawl）
3. 对HTML页面的数据解析（Parse）
4. 对解析后的数据进行存储（Save）
5. 动态HTML的处理/验证码的处理
6. 爬虫(Spider)、反爬虫(Anti-Spider)、反反爬虫(Anti-Anti-Spider)之间的斗争...

可选择的**IDE**和编辑器：

工欲善其事，必先利其器：

IDE：PyCharm、Spyder、Eclipse+PyDev、Visual Studio等

编辑器：Vim、Sublime Text、VSCode、Atom等

Copyright © 黑五电商学院 amzfriday.com all right reserved

通用爬虫和聚焦爬虫

根据使用场景，网络爬虫可分为 通用爬虫 和 聚焦爬虫 两种。

通用爬虫

通用网络爬虫 就是 搜索引擎抓取系统，目的是将互联网上的所有的网页下载到本地，形成一个互联网内容的镜像备份。

它决定着整个搜索引擎内容的丰富性和时效性，因此它的性能优劣直接影响着搜索引擎的效果。

通用搜索引擎（Search Engine）工作原理

第一步：抓取网页

搜索引擎网络爬虫的基本工作流程如下：

首先选取一部分的初始URL，将这些URL放入 待抓取URL队列 ；

取出待抓取URL，解析DNS得到主机的IP，并将URL对应的网页下载下来，存储进已下载网页库中，并且将这些URL放进 已抓取URL队列 。

分析网页中包含的其他URL，并且将URL放入 待抓取URL队列，从而进入下一个循环...

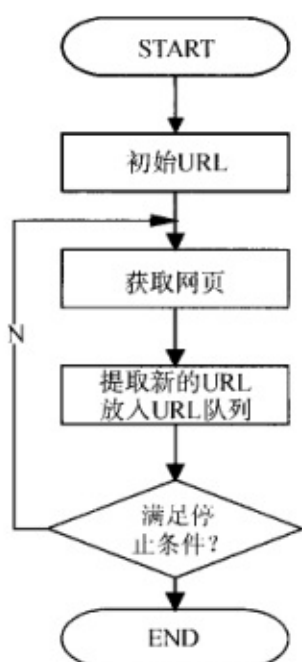


图 1 通用网络爬虫工作流程图

PS：搜索引擎如何获取一个新网站的URL：

1. 新网站向搜索引擎主动提交网址：（如百度<http://zhazhang.baidu.com/linksubmit/url>）
2. 在其他网站上设置新网站外链（尽可能处于搜索引擎爬虫爬取范围）
3. 搜索引擎和DNS解析服务商（如DNSPod等）合作，新网站域名将被迅速抓取。

网站的Robots协议

搜索引擎蜘蛛跟进URL抓取时，它需要遵从一些命令或文件的内容，如标注为nofollow的链接，或者是网站的Robots协议。

Robots协议（也叫爬虫协议、机器人协议等），全称是“网络爬虫排除标准”（Robots Exclusion Protocol），网站通过Robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取，例如：

淘宝网：<https://www.taobao.com/robots.txt>

腾讯网：<http://www.qq.com/robots.txt>

第二步：数据存储

搜索引擎爬虫爬取到的网页，将存入原始页面数据库，页面和用户浏览器得到的HTML是完全一样的。

搜索引擎爬虫在抓取页面时，也做一定的重复内容检测，一旦遇到访问权重很低的网站上有大量抄袭、采集或者复制的内容，很可能就不再爬行。

第三步：预处理

搜索引擎将爬虫抓取回来的页面，进行各种步骤的预处理。

提取文字

中文分词

消除噪音（比如版权声明文字、导航条、广告等.....）

索引处理

特殊文件处理

...

除了HTML文件外，搜索引擎通常还能抓取和索引以文字为基础的多种文件类型，如 PDF、Word、WPS、XLS、PPT、TXT 文件等。我们在搜索结果中也经常会看到这些文件类型。

但搜索引擎还不能处理图片、视频、Flash 这类非文字内容，也不能执行脚本和程序。

第四步：提供检索服务，网站排名

搜索引擎在对信息进行组织和处理后，为用户提供关键字检索服务，将用户检索相关的信息展示给用户。

同时会根据页面的PageRank值（链接的访问量排名）来进行网站排名，这样Rank值高的网站在搜索结果中会排名较前，当然也可以直接使用 Money 购买搜索引擎网站排名，简单粗暴。

完整流程图：

但是，这些通用性搜索引擎也存在着一定的局限性：

1. 通用搜索引擎所返回的结果都是网页，而大多情况下，网页里90%的内容对用户来说都是无用的。
2. 不同领域、不同背景的用户往往具有不同的检索目的和需求，搜索引擎无法提供针对具体某个用户的搜索结果。
3. 万维网数据形式的丰富和网络技术的不断发展，图片、数据库、音频、视频多媒体等不同数据大量出现，通用搜索引擎对这些文件无能为力，不能很好地发现和获取。
4. 通用搜索引擎大多提供基于关键字的检索，难以支持根据语义信息提出的查询，无法准确理解用户的具体需求。

针对这些情况，聚焦爬虫技术得以广泛使用。

聚焦爬虫

聚焦爬虫，是“面向特定主题需求”的一种网络爬虫程序，它与通用搜索引擎爬虫的区别在于： 聚焦爬虫在实施网页抓取时会对内容进行处理筛选，尽量保证只抓取与需求相关的网页信息。

而我们今后要学习的网络爬虫，就是聚焦爬虫。

Copyright © 黑五电商学院 amzfriday.com all right reserved

网络爬虫工作过程可以理解为模拟浏览器操作的过程，浏览器的主要功能是向服务器发出请求，在浏览器窗口中展示服务器返回的网络资源。

一、浏览器处理网页的过程

我们先来看一下浏览网页的基本过程，比如我们在浏览器地址栏输入：<http://www.baidu.com> 回车后会浏览器会显示百度的首页。

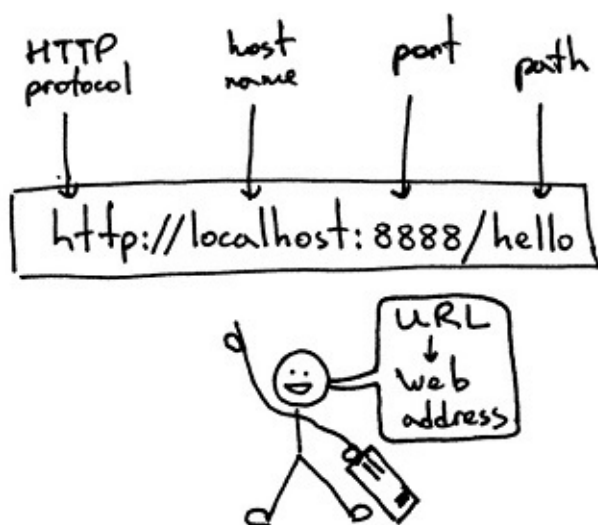
这段网络通信过程中到底发生了什么？简单来说这段过程发生了以下四个步骤：

1. 当我们在浏览器输入URL <http://www.baidu.com> 的时候，浏览器先通过 DNS服务器 查找 URL的域名对应的 IP地址。
2. 浏览器会向 IP地址 对应的 Web服务器 发送HTTP请求，去获取 <http://www.baidu.com> 的html文件，Web服务器响应请求，将html文件发送回给浏览器。
3. 浏览器分析服务器返回的HTML文件，发现其中引用了很多其他文件，比如图片文件，CSS文件，JS文件。浏览器会自动再次发送每个文件的HTTP请求，去获取这些文件。
4. 当所有的文件都获取成功后，浏览器会根据HTML语法结构，将所有内容完整的显示出来。

名词注解：

1. URL

URL (Uniform / Universal Resource Locator的缩写)：统一资源定位符，是用于完整地描述Internet上网页和其他资源的地址的一种标识方法。



基本格式：`scheme://host[:port]/path/[?query-string][#anchor]`

- scheme：协议(例如：http, https, ftp等)
- host：服务器的域名 或 IP地址

- port: 服务器的端口 (如果是走协议默认端口, HTTP缺省端口80)
- path: 访问资源的路径
- query-string: 查询字符串参数, 发送给http服务器的数据
- anchor: 锚点 (跳转到网页的指定锚点位置)

例如:

- ftp://192.168.0.123:8080/index
- <http://www.baidu.com>
- <http://item.jd.com/11936238.html#product-detail>

2. DNS

DNS 是计算机域名系统 (Domain Name System 或Domain Name Service) 的缩写, 由解析器和域名服务器组成的。

解析器用来解析URL地址对应的域名, 而域名服务器是指保存有该网络中所有主机的域名和对应IP地址, 并具有将域名转换为IP地址功能的服务器。

一般一个域名的 DNS解析时间 在10~60毫秒之间。

一个域名必须对应一个IP地址, 而一个IP地址不一定会有域名。

3. HTTP和HTTPS

HTTP协议 (HyperText Transfer Protocol, 超文本传输协议): 是一种发布和接收 HTML页面的方法。

HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) 简单讲是HTTP的安全版, 在HTTP下加入SSL层。

SSL (Secure Sockets Layer 安全套接层) 主要用于Web的安全传输协议, 在传输层对网络连接进行加密, 保障在Internet上数据传输的安全。

- HTTP的端口号为80,
- HTTPS的端口号为443

二、HTTP的请求与响应

HTTP通信由两部分组成: 客户端请求消息 与 服务器响应消息

客户端HTTP请求

URL只是标识资源的位置，而HTTP是用来提交和获取资源。客户端发送一个HTTP请求到服务器的请求消息，包括以下格式：

请求行、请求头部、空行、请求数据

四个部分组成，下图给出了请求报文的一般格式。



一个典型的HTTP请求示例

```
1. GET https://www.baidu.com/ HTTP/1.1
2. Host: www.baidu.com
3. Connection: keep-alive
4. Upgrade-Insecure-Requests: 1
5. User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36
6. Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
7. Referer: http://www.baidu.com/
8. Accept-Encoding: gzip, deflate, sdch, br
9. Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
10. Cookie: BAIDUID=04E4001F34EA74AD4601512DD3C41A7B:FG=1;
  BIDUPSID=04E4001F34EA74AD4601512DD3C41A7B; PSTM=1470329258;
  MCITY=-343%3A340%3A; H_PS_PSSID=1447_18240_21105_21386_21454_21409_21554;
  BD_UPN=12314753; sug=3; sugstore=0; ORIGIN=0; bdime=0;
  H_PS_645EC=7e2ad3QH1181NSPbFbd7PRUCE1LlufzxrcFmwYin0E6b%2BW8bbTMKHZbDP0g;
  BDSVRTM=0
```

请求方法

```
GET https://www.baidu.com/ HTTP/1.1
```

根据HTTP标准，HTTP请求可以使用多种请求方法。

HTTP 0.9：只有基本的文本 GET 功能。

HTTP 1.0：完善的请求/响应模型，并将协议补充完整，定义了三种请求方法：GET，POST 和 HEAD方法。

HTTP 1.1：在 1.0 基础上进行更新，新增了五种请求方法：OPTIONS，PUT，DELETE，TRACE 和 CONNECT 方法。

HTTP 2.0（未普及）：请求/响应首部的定义基本没有改变，只是所有首部键必须全部小写，而且请求行要独立为 :method、:scheme、:host、:path这些键值对。

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件），数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
3	HEAD	类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。
5	DELETE	请求服务器删除指定的页面。
6	CONNECT	HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。

HTTP请求主要分为Get和Post两类：

- GET是从服务器上获取指定页面信息，POST是向服务器提交数据并获取页面信息。
- GET请求参数都显示在URL上，服务器根据该请求所包含URL中的QueryString参数来产生响应内容。“Get”请求的参数是URL的一部分。
- POST请求参数在请求体Formdata中，消息长度没有限制而且以隐式的方式进行发送，通常用来向HTTP服务器提交量比较大的数据（比如请求中包含许多参数或者文件上传操作等）。“POST”请求的参数不在URL中，而在请求体中。

三、常用的请求报头

1. Host（主机和端口号）

Host：对应网址URL中的Web名称和端口号，用于指定被请求资源的Internet主机和端口号，通常属于URL的Host部分。

2. Connection（连接类型）

Connection：表示客户端与服务连接类型，通常情况下：

- Client 发起一个包含 Connection:keep-alive 的请求（HTTP/1.1使用 keep-alive 为默认值）

- Server收到请求后：

如果 Server 支持 `keep-alive`，回复一个包含 `Connection:keep-alive` 的响应，不关闭连接；
如果 Server 不支持 `keep-alive`，回复一个包含 `Connection:close` 的响应，关闭连接。

- 如果client收到包含 `Connection:keep-alive` 的响应，向同一个连接发送下一个请求，直到一方主动关闭连接。

Connection : keep-alive 在很多情况下能够重用连接，减少资源消耗，缩短响应时间。比如当浏览器需要多个文件时(比如一个HTML文件和多个Image文件)，不需要每次都去请求建立连接。

3. Upgrade-Insecure-Requests (升级为HTTPS请求)

Upgrade-Insecure-Requests：升级不安全的请求，意思是会在加载 http 资源时自动替换成 https 请求，让浏览器不再显示https页面中的http请求警报。

HTTPS 是以安全为目标的 HTTP 通道，所以在 HTTPS 承载的页面上不允许出现 HTTP 请求，一旦出现就是提示或报错。

4. User-Agent (浏览器名称)

User-Agent：标识客户端身份的名称，通常页面会根据不同的User-Agent信息自动做出适配，甚至返回不同的响应内容。

5. Accept (传输文件类型)

Accept：指浏览器或其他客户端可以接受的MIME (Multipurpose Internet Mail Extensions (多用途互联网邮件扩展)) 文件类型，服务器可以根据它判断并返回适当的文件格式。

举例：

- Accept: /*: 表示什么都可以接收。
- Accept: image/gif: 表明客户端希望接受GIF图像格式的资源；
- Accept: text/html: 表明客户端希望接受html文本。
- Accept: text/html, application/xhtml+xml;q=0.9, image/*;q=0.8: 表示浏览器支持的 MIME 类型分别是 html文本、xhtml和xml文档、所有的图像格式资源。

q 是权重系数，范围 $0 \leq q \leq 1$ ， q 值越大，请求越倾向于获得其“;”之前的类型表示的内容。若没有指定 q 值，则默认为1，按从左到右排序顺序；若被赋值为0，则用于表示浏览器不接受此内容类型。

Text：用于标准化地表示的文本信息，文本消息可以是多种字符集和或者多种格式的；Application 用于传输应用程序数据或者二进制数据。详细请点击

6. Referer（页面跳转来源）

Referer：表明产生请求的网页来自于哪个URL，用户是从该 Referer页面访问到当前请求的页面。这个属性可以用来跟踪Web请求来自哪个页面，是从什么网站来的等。

防盗链：有时候遇到下载某网站图片，需要对应的`referer`，否则无法下载图片，那是因为人家做了防盗链，原理就是根据`referer`去判断是否是本网站的地址，如果不是，则拒绝，如果是，就可以下载。

7. Accept-Encoding（文件编解码格式）

Accept-Encoding：指出浏览器可以接受的编码方式。编码方式不同于文件格式，它是为了压缩文件并加速文件传递速度。浏览器在接收到Web响应之后先解码，然后再检查文件格式，许多情形下这可以减少大量的下载时间。

举例：`Accept-Encoding:gzip;q=1.0, identity; q=0.5, *;q=0`

如果有多个Encoding同时匹配，按照q值顺序排列，本例中按顺序支持 gzip, identity压缩编码，支持gzip的浏览器会返回经过gzip编码的HTML页面。

如果请求消息中没有设置这个报头，通常服务器假定客户端不支持压缩，直接返回文本。

8. Accept-Language（语言种类）

Accept-Langeuage：指出浏览器可以接受的语言种类，如en或en-us指英语，zh或者zh-cn指中文，当服务器能够提供一种以上的语言版本时要用到。

如果目标网站支持多个语种的话，可以使用这个信息来决定返回什么语言的网页。

9. Accept-Charset（字符编码）

Accept-Charset：指出浏览器可以接受的字符编码。

举例：`Accept-Charset:iso-8859-1,gb2312,utf-8`

- ISO8859-1：通常叫做Latin-1。Latin-1包括了书写所有西方欧洲语言不可缺少的附加字符，英文浏览器的默认值是ISO-8859-1。
- gb2312：标准简体中文字符集；
- utf-8：UNICODE 的一种变长字符编码，可以解决多种语言文本显示问题，从而实现应用国际化和本地化。

如果在请求消息中没有设置这个域，默认客户端是任何字符集都可以接受，则返回网页`charset`指定的编码。

10. Cookie（Cookie）

Cookie：浏览器用这个属性向服务器发送Cookie。Cookie是在浏览器中寄存的小型数据体，它可以

记载和服务器相关的用户信息，也可以用来实现模拟登陆，之后会详细讲。

11. Content-Type (POST数据类型)

Content-Type: POST请求里用来表示的内容类型。

举例: `Content-Type = Text/XML; charset=gb2312 :`

指明该请求的消息体中包含的是纯文本的XML类型的数据，字符编码采用“gb2312”。

四、服务端HTTP响应

HTTP响应也由四个部分组成，分别是： 状态行、消息报头、空行、响应正文

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 122

<html>
<head>
<title>Wrox Homepage</title>
</head>
<body>
<!-- body goes here -->
</body>
</html>
```

```
1. HTTP/1.1 200 OK
2. Server: Tengine
3. Connection: keep-alive
4. Date: Wed, 30 Nov 2016 07:58:21 GMT
5. Cache-Control: no-cache
6. Content-Type: text/html; charset=UTF-8
7. Keep-Alive: timeout=20
8. Vary: Accept-Encoding
9. Pragma: no-cache
10. X-NWS-LOG-UUID: bd27210a-24e5-4740-8f6c-25dbafa9c395
11. Content-Length: 180945
```

常用的响应报头(了解)

理论上所有的响应头信息都应该是回应请求头的。但是服务端为了效率，安全，还有其他方面的考虑，

会添加相对应的响应头信息，从上图可以看到：

1. Cache-Control: must-revalidate, no-cache, private。

这个值告诉客户端，服务端不希望客户端缓存资源，在下次请求资源时，必须要从新请求服务器，不能从缓存副本中获取资源。

- Cache-Control是响应头中很重要的信息，当客户端请求头中包含Cache-Control:max-age=0请求，明确表示不会缓存服务器资源时,Cache-Control作为作为回应信息，通常会返回no-cache，意思就是说，“那就不缓存呗”。
- 当客户端在请求头中没有包含Cache-Control时，服务端往往会定,不同的资源不同的缓存策略，比如说oschina在缓存图片资源的策略就是Cache-Control: max-age=86400,这个意思是，从当前时间开始，在86400秒的时间内，客户端可以直接从缓存副本中读取资源，而不需要向服务器请求。

2. Connection: keep-alive

这个字段作为回应客户端的Connection: keep-alive，告诉客户端服务器的tcp连接也是一个长连接，客户端可以继续使用这个tcp连接发送http请求。

3. Content-Encoding:gzip

告诉客户端，服务端发送的资源是采用gzip编码的，客户端看到这个信息后，应该采用gzip对资源进行解码。

4. Content-Type: text/html;charset=UTF-8

告诉客户端，资源文件的类型，还有字符编码，客户端通过utf-8对资源进行解码，然后对资源进行html解析。通常我们会看到有些网站是乱码的，往往就是服务器端没有返回正确的编码。

5. Date: Sun, 21 Sep 2016 06:18:21 GMT

这个是服务端发送资源时的服务器时间，GMT是格林尼治所在地的标准时间。http协议中发送的时间都是GMT的，这主要是解决在互联网上，不同时区在相互请求资源的时候，时间混乱问题。

6. Expires:Sun, 1 Jan 2000 01:00:00 GMT

这个响应头也是跟缓存有关的，告诉客户端在这个时间前，可以直接访问缓存副本，很显然这个值会存在问题，因为客户端和服务器的时间不一定会都是相同的，如果时间不同就会导致问题。所以这个响应头是没有Cache-Control: max-age=*这个响应头准确的，因为max-age=date中的date是个相对

时间，不仅更好理解，也更准确。

7. Pragma: no-cache

这个含义与Cache-Control等同。

8. Server: Tengine/1.4.6

这个是服务器和相对应的版本，只是告诉客户端服务器的信息。

9. Transfer-Encoding: chunked

这个响应头告诉客户端，服务器发送的资源的方式是分块发送的。一般分块发送的资源都是服务器动态生成的，在发送时还不知道发送资源的大小，所以采用分块发送，每一块都是独立的，独立的块都能标示自己的长度，最后一块是0长度的，当客户端读到这个0长度的块时，就可以确定资源已经传输完了。

10. Vary: Accept-Encoding

告诉缓存服务器，缓存压缩文件和非压缩文件两个版本，现在这个字段用处并不大，因为现在的浏览器都是支持压缩的。

五、响应状态码

响应状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值（后面会详细列出）。

常见状态码：

- 100~199：表示服务器成功接收部分请求，要求客户端继续提交其余请求才能完成整个处理过程。
- 200~299：表示服务器成功接收请求并已完成整个处理过程。常用200（OK 请求成功）。
- 300~399：为完成请求，客户需进一步细化请求。例如：请求的资源已经移动一个新地址、常用302（所请求的页面已经临时转移至新的url）、307和304（使用缓存资源）。
- 400~499：客户端的请求有错误，常用404（服务器无法找到被请求的页面）、403（服务器拒绝访问，权限不够）。
- 500~599：服务器端出现错误，常用500（请求未完成。服务器遇到不可预知的情况）。

Requests：让 HTTP 服务人类

虽然Python的标准库中 `urllib2` 模块已经包含了平常我们使用的大多数功能，但是它的 API 使用起来让人感觉不太好，而 `Requests` 自称“HTTP for Humans”，说明使用更简洁方便。

Requests 唯一的一个非转基因的 Python HTTP 库，人类可以安全享用：)

`Requests` 继承了`urllib2`的所有特性。`Requests`支持HTTP连接保持和连接池，支持使用cookie保持会话，支持文件上传，支持自动确定响应内容的编码，支持国际化的 URL 和 POST 数据自动编码。

requests 的底层实现其实就是 urllib3

`Requests`的文档非常完备，中文文档也相当不错。`Requests`能完全满足当前网络的需求，支持 Python 2.6–3.5，而且能在PyPy下完美运行。

开源地址：<https://github.com/kennethreitz/requests>

中文文档 API：http://docs.python-requests.org/zh_CN/latest/index.html

安装方式

利用 `pip` 安装 或者利用 `easy_install` 都可以完成安装：

```
$ pip install requests
```

```
$ easy_install requests
```

基本GET请求（headers参数 和 params参数）

1. 最基本的GET请求可以直接用get方法

```
1. response = requests.get("http://www.baidu.com/")
2.
3. # 也可以这么写
4. # response = requests.request("get", "http://www.baidu.com/")
```

2. 添加 headers 和 查询参数

如果想添加 `headers`，可以传入`headers`参数来增加请求头中的`headers`信息。如果要参数放在 `url`中传递，可以利用 `params` 参数。

```
1. import requests
2.
```

```

3. kw = {'wd': '长城'}
4.
5. headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}
6.
7. # params 接收一个字典或者字符串的查询参数, 字典类型自动转换为url编码, 不需要urlencode()
8. response = requests.get("http://www.baidu.com/s?", params = kw, headers =
headers)
9.
10. # 查看响应内容, response.text 返回的是Unicode格式的数据
11. print(response.text)
12.
13. # 查看响应内容, response.content返回的字节流数据
14. print(response.content)
15.
16. # 查看完整url地址
17. print(response.url)
18.
19. # 查看响应头部字符编码
20. print(response.encoding)
21.
22. # 查看响应码
23. print(response.status_code)
24. 运行结果
25.
26.
27. ....
28.
29. ....
30.
31. 'http://www.baidu.com/s?wd=%E9%95%BF%E5%9F%8E'
32.
33. 'utf-8'
34.
35. 200

```

- 使用`response.text` 时, `Requests` 会基于 `HTTP` 响应的文本编码自动解码响应内容, 大多数 `Unicode` 字符集都能被无缝地解码。
- 使用`response.content` 时, 返回的是服务器响应数据的原始二进制字节流, 可以用来保存图片等二进制文件。

基本POST请求 (data参数)

1. 最基本的GET请求可以直接用post方法

```
response = requests.post("http://www.baidu.com/", data = data)
```

2. 传入data数据

对于 POST 请求来说，我们一般需要为它增加一些参数。那么最基本的传参方法可以利用 data 这个参数。

```
1. import requests
2.
3. # POST请求的目标URL
4. url = "http://fanyi.qq.com/api/translate"
5.
6. headers={"User-Agent": "Mozilla..."}
7.
8. keyword = {
9.     "source":"auto",
10.    "target" : "auto",
11.    "sourceText" : "你好Python", # 需要翻译的内容
12.    "sessionId " : "translate_uuid1517720945431" #translate_uuid + Unix时间戳
    (毫秒)
13. }
14.
15. result = requests.post(url, headers = headers, data = keyword).json()
16. print(result['translate']['records'][0]['targetText'])
17.
18. # Hello, Python.
```

运行结果

```
1. {"type":"EN2ZH_CN","errorCode":0,"elapsedTime":2,"translateResult":[[{"src":"i
love python","tgt":"我喜欢python"}]], "smartResult":{"type":1,"entries":["", "肆
文", "高德纳"]}}
2. {u'errorCode': 0, u'elapsedTime': 0, u'translateResult': [[{u'src': u'i love
python', u'tgt': u'\u6211\u559c\u6b22python'}]], u'smartResult': {u'type': 1,
u'entries': [u'', u'\u8086\u6587', u'\u9ad8\u5fb7\u7eb3']}, u'type':
u'EN2ZH_CN'}
```

代理（proxies参数）

如果需要使用代理，你可以通过为任意请求方法提供 proxies 参数来配置单个请求：

```

1. import requests
2.
3. # 根据协议类型，选择不同的代理
4. proxies = {"http": "http://12.34.56.79:9527"}
5. response = requests.get("http://httpbin.org/ip", proxies = proxies)
6. print(response.text)

```

私密代理验证（特定格式）

```

1. import requests
2.
3. # 如果代理需要使用HTTP Basic Auth，可以使用下面这种格式：
4. proxy = { "http": "friday:stone@61.158.163.130:16816" }
5. response = requests.get("http://www.baidu.com", proxies = proxy)
6. print(response.text)

```

Cookies 和 Session

Cookies

如果一个响应中包含了cookie，那么我们可以利用 cookies参数拿到：

```

1. import requests
2.
3. response = requests.get("http://www.baidu.com/")
4.
5. # 返回CookieJar对象：
6. cookiejar = response.cookies
7.
8. # 将CookieJar转为字典：
9. cookiedict = requests.utils.dict_from_cookiejar(cookiejar)
10.
11. print(cookiejar)
12.
13. print(cookiedict)

```

运行结果：

```

1. <RequestsCookieJar[<Cookie BDORZ=27315 for .baidu.com/>]>
2. {'BDORZ': '27315'}

```

Session

在 requests 里，session对象是一个非常常用的对象，这个对象代表一次用户会话：从客户端浏览器连接服务器开始，到客户端浏览器与服务器断开。

会话能让我们在跨请求时候保持某些参数，比如在同一个 Session 实例发出的所有请求之间保持 cookie 。

```
1. 实现人人网登录
2. import requests
3.
4. # 1. 创建session对象，可以保存Cookie值
5. ssion = requests.session()
6.
7. # 2. 处理 headers
8. headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}
9.
10. # 3. 需要登录的用户名和密码
11. data = {"email":"stone_leaf@126.com", "password":"amzfriday"}
12.
13. # 4. 发送附带用户名和密码的请求，并获取登录后的Cookie值，保存在ssion里
14. ssion.post("http://www.renren.com/PLogin.do", data = data)
15.
16. # 5. ssion包含用户登录后的Cookie值，可以直接访问那些登录后才可以访问的页面
17. response = ssion.get("http://www.renren.com/410043129/profile")
18.
19. # 6. 打印响应内容
20. print(response.text)
```

处理HTTPS请求 SSL证书验证

Requests也可以为HTTPS请求验证SSL证书：

- 要想检查某个主机的SSL证书，你可以使用 verify 参数（也可以不写）

```
1. import requests
2. response = requests.get("https://www.baidu.com/", verify=True)
3.
4. # 也可以省略不写
```

```
5. # response = requests.get("https://www.baidu.com/")
6. print(r.text)
```

运行结果：

```
1. <!DOCTYPE html>
2. <!--STATUS OK--><html> <head><meta http-equiv=content-type
   content=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible
   content=IE=Edge>百度一下，你就知道 ....
```

- 如果SSL证书验证不通过，或者不信任服务器的安全证书，则会报出SSLError，据说 12306 证书是自己做的：

来测试一下：

```
1. import requests
2. response = requests.get("https://www.12306.cn/mormhweb/")
3. print(response.text)
```

果然：

```
1. SSLError: ("bad handshake: Error([('SSL routines',
   'ssl3_get_server_certificate', 'certificate verify failed')]),)
```

如果我们想跳过 12306 的证书验证，把 verify 设置为 False 就可以正常请求了。

```
1. r = requests.get("https://www.12306.cn/mormhweb/", verify = False)
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

目标

掌握chrome在爬虫中的使用 ：)

1. 新建隐身窗口



1.1 为什么需要新建隐身窗口

在打开隐身窗口的时候，第一次请求某个网站是没有携带cookie的，和代码请求一个网站一样，不携

带cookie。这样就能够尽可能的理解代码请求某个网站的结果；除非数据是通过js加载出来的，不然爬虫请求到的数据和浏览器请求的数据大部分时候都是相同的

2. chrome中network的更多功能

2.1 Perserve log

默认情况下，页面发生跳转之后，之前的请求url地址等信息都会消失，勾选perserve log后之前的请求都会被保留

2.2 filter过滤

在url地址很多的时候，可以在filter中输入部分url地址，对所有的url地址起到一定的过滤效果，具体位置在上面第二幅图中的2的位置

2.3 观察特定种类的请求

在上面第二幅图中的3的位置，有很多选项，默认是选择的all，即会观察到所有种类的请求

很多时候处于自己的目的可以选择all右边的其他选项，比如常见的选项：

- XHR:大部分情况表示ajax请求
- JS:js请求
- CSS:css请求

但是很多时候我们并不能保证我们需要的请求是什么类型，特别是我们不清楚一个请求是否为ajax请求的时候，直接选择all,从前往后观察即可，其中js, css, 图片等不去观察即可

不要被浏览器中的一堆请求吓到了，这些请求中除了js, css, 图片的请求外，其他的请求并没有多少个

Copyright © 黑五电商学院 amzfriday.com all right reserved

页面解析和数据提取

一般来讲对我们而言，爬虫需要抓取的是某个网站或者某个应用的内容，提取有用的数据。响应内容一般分为两种，非结构化的数据和 结构化的数据。

- 结构化数据：先有结构、再有数据
 - 非结构化数据：先有数据，再有结构，
 - 不同类型的数据，我们需要采用不同的方式来处理。
-

结构化的数据处理

HTML 文件

- 正则表达式
 - XPath
 - CSS选择器
-

JSON 文件

- JsonPath
 - JSON 模块转化成Python类型进行操作
-

XML 文件

- lxml模块 模块转化成Python类型进行操作
 - XPath
 - CSS选择器
 - 正则表达式
-

非结构化的数据处理

普通文本文件（如提取电话号码、邮箱地址等）

正则表达式

JavaScript 文件、**CSS** 文件（提取特定值等）

正则表达式

二进制文件（图片、音乐、视频等）

无法提取，直接保存指定格式的磁盘文件

Copyright © 黑五电商学院 amzfriday.com all right reserved

案例：使用BeautifulSoup4的爬虫

我们以 亚马逊Kindle电子书销售排行榜 商品页面来做演

示：<https://www.amazon.cn/gp/bestsellers/digital-text/116169071>



使用BeautifulSoup4解析器，将每件商品的ASIN、标题、价格、star、评价数量，以及每件商品的链接爬取下来并存储在.csv文件中。

```
1. import csv
2. import requests
3. from bs4 import BeautifulSoup
4.
5.
6. def amazon():
7.     base_url = 'https://www.amazon.cn'
8.     url = 'https://www.amazon.cn/gp/bestsellers/digital-text/116169071'
9.     headers = {
10.         'Connection': 'keep-alive',
11.         'Cache-Control': 'max-age=0',
12.         'Upgrade-Insecure-Requests': '1',
13.         'DNT': '1',
14.         'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
15.         'Accept': 'image/webp,image/apng,image/*,*/*;q=0.8',
16.         'Accept-Encoding': 'gzip, deflate, br',
17.         'Accept-Language': 'zh-HK,zh-CN;q=0.9,zh;q=0.8',
```

```

18.         'Referer': 'https://www.amazon.cn/gp/bestsellers/digital-
            text/116169071',
19.     }
20.     # 给亚马逊发送请求
21.     session = requests.session()
22.     session.get('https://www.amazon.com')
23.     resHtml = session.get(url, headers=headers).content.decode('utf-8')
24.     # 将获取的对象转化为bs4对象
25.     html_soup = BeautifulSoup(resHtml, 'lxml')
26.     # 获取所有商品的标签
27.     all_goods_li = html_soup.find('ol', id='zg-ordered-list').find_all('li',
        'zg-item-immersion')
28.     for li in all_goods_li:
29.         # 准备一个空列表, 用于储存商品信息
30.         goods_info_list = []
31.         # 商品链接
32.         link = base_url + li.find('a', target='_blank')['href']
33.         # 商品 asin
34.         asin = link.split('/dp/')[1].split('/')[0]
35.         # 标题
36.         title = li.select("div[data-rows='1']")[0].get_text().strip()
37.         # 价格
38.         price = li.find('span', 'p13n-sc-price').text
39.         # 星级
40.         star = li.find('span', 'a-icon-alt').text
41.         # 评价数
42.         reviews = li.select("a[class='a-size-small a-link-normal']")
            [0].get_text()
43.
44.         # 将爬到的数据添加到列表中
45.         goods_info_list.append(asin)
46.         goods_info_list.append(title)
47.         goods_info_list.append(price)
48.         goods_info_list.append(star)
49.         goods_info_list.append(reviews)
50.         goods_info_list.append(link)
51.
52.         # 将数据写入 名为amazon_book.csv的文件中
53.         csvFile = open('./amazon_book.csv', 'a', newline='',
            encoding='gb18030') # 设置newline, 否则两行之间会空一行
54.         writer = csv.writer(csvFile)
55.         writer.writerow(goods_info_list)

```

```
56.         csvFile.close()  
57.  
58.  
59. if __name__ == '__main__':  
60.     amazon()
```

[illegible]

- 30 -

数据提取之JSON与JsonPATH

JSON(JavaScript Object Notation) 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。

JSON和XML的比较可谓不相上下。

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

Json在线解析网站：<http://www.json.cn/#>

JSON

json简单说就是javascript中的对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构

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[]括起来的内容，数据结构为 [*"Python"*, *"javascript"*, *"C++"*, ...]，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。

import json

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。

1. json.loads()

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	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
true	True
false	False
null	None

```

1. # json_loads.py
2.
3. import json
4.
5. strList = '[1, 2, 3, 4]'
6.
7. strDict = '{"city": "北京", "name": "大猫"}'
8.
9. json.loads(strList)
10. # [1, 2, 3, 4]
11.
12. json.loads(strDict) # json数据自动按Unicode存储
13. # {'city': u'\u5317\u4eac', 'name': u'\u5927\u732b'}
```

2. 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：

Python	JSON
dict	object
list, tuple	array
str, unicode	string
int, long, float	number
True	true
False	false
None	null

```

1. # json_dumps.py
2.
3. import json
```



```

4. import chardet
5.
6. listStr = [1, 2, 3, 4]
7. tupleStr = (1, 2, 3, 4)
8. dictStr = {"city": "北京", "name": "大猫"}
9.
10. json.dumps(listStr)
11. # '[1, 2, 3, 4]'
12. json.dumps(tupleStr)
13. # '[1, 2, 3, 4]'
14.
15. # 注意：json.dumps() 处理中文时默认使用的ascii编码，会导致中文无法正常显示
16. print json.dumps(dictStr)
17. # {"city": "\u5317\u4eac", "name": "\u5927\u732b"}
18.
19. # 记住：处理中文时，添加参数 ensure_ascii=False 来禁用ascii编码
20. print json.dumps(dictStr, ensure_ascii=False)
21. # {"city": "北京", "name": "大刘"}

```

3. json.dump()

将Python内置类型序列化为json对象后写入文件

```

1.
2. # json_dump.py
3.
4. import json
5.
6. listStr = [{"city": "北京"}, {"name": "大刘"}]
7. json.dump(listStr, open("listStr.json", "w"), ensure_ascii=False)
8.
9. dictStr = {"city": "北京", "name": "大刘"}
10. json.dump(dictStr, open("dictStr.json", "w"), ensure_ascii=False)

```

4. json.load()

读取文件中json形式的字符串元素 转化成python类型

```

1. # json_load.py
2.

```

```

3. import json
4.
5. strList = json.load(open("listStr.json"))
6. print strList
7.
8. # [{u'city': u'\u5317\u4eac'}, {u'name': u'\u5927\u5218'}]
9.
10. strDict = json.load(open("dictStr.json"))
11. print strDict
12. # {u'city': u'\u5317\u4eac', u'name': u'\u5927\u5218'}

```

JsonPath

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。

安装方法: `pip install jsonpath`

官方文档: <http://goessner.net/articles/JsonPath>

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。

示例：

我们以拉勾网城市JSON文件 <http://www.lagou.com/lbs/getAllCitySearchLabels.json> 为例，获取所有城市。

```

1. # jsonpath_lagou.py
2.
3. import urllib2
4. import jsonpath
5. import json
6.
7. url = 'http://www.lagou.com/lbs/getAllCitySearchLabels.json'
8. request =urllib2.Request(url)
9. response = urllib2.urlopen(request)
10. html = response.read()

```

```

11.
12. # 把json格式字符串转换成python对象
13. jsonobj = json.loads(html)
14.
15. # 从根节点开始, 匹配name节点
16. citylist = jsonpath.jsonpath(jsonobj, '$..name')
17.
18. print citylist
19. print type(citylist)
20. fp = open('city.json', 'w')
21.
22. content = json.dumps(citylist, ensure_ascii=False)
23. print content
24.
25. fp.write(content.encode('utf-8'))
26. fp.close()

```

注意事项:

`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`

如:

```
dataDict = json.loads(jsonStrGBK);
```

`jsonStrGBK`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");
```

附: 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。 其实编码问题很好搞定, 只要记住一点:

任何平台的任何编码 都能和 **Unicode** 互相转换

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。

```

1. # 这是一个 UTF-8 编码的字符串
2. utf8Str = "你好地球"

```

```
3.  
4. # 1. 将 UTF-8 编码的字符串 转换成 Unicode 编码  
5. unicodeStr = utf8Str.decode("UTF-8")  
6.  
7. # 2. 再将 Unicode 编码格式字符串 转换成 GBK 编码  
8. gbkData = unicodeStr.encode("GBK")  
9.  
10. # 1. 再将 GBK 编码格式字符串 转化成 Unicode  
11. unicodeStr = gbkData.decode("gbk")  
12.  
13. # 2. 再将 Unicode 编码格式字符串转换成 UTF-8  
14. utf8Str = unicodeStr.encode("UTF-8")
```

decode的作用是将其他编码的字符串转换成 Unicode 编码

encode的作用是将 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

Copyright © 黑五电商学院 amzfriday.com all right reserved

有同学说，我正则用的不好，处理HTML文档很累，有没有其他的方法？

有！那就是XPath，我们可以先将 HTML文件 转换成 XML文档，然后用 XPath语法 查找 HTML 节点或元素。

什么是XML

- XML 指可扩展标记语言 (Extensible Markup Language)
- XML 是一种标记语言，很类似 HTML
- XML 的设计宗旨是传输数据，而非显示数据
- XML 的标签需要我们自行定义。
- XML 被设计为具有自我描述性。
- XML 是 W3C 的推荐标准

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

XML 和 HTML 的区别

数据格式	描述	设计目标
XML	Extensible Markup Language (可扩展标记语言)	被设计为传输和存储数据，其焦点是数据的内容。
HTML	HyperText Markup Language (超文本标记语言)	显示数据以及如何更好显示数据。
HTML DOM	Document Object Model for HTML (文档对象模型)	通过 HTML DOM，可以访问所有的 HTML 元素，连同它们所包含的文本和属性。可以对其中的内容进行修改和删除，同时也可以创建新的元素。

XML文档示例

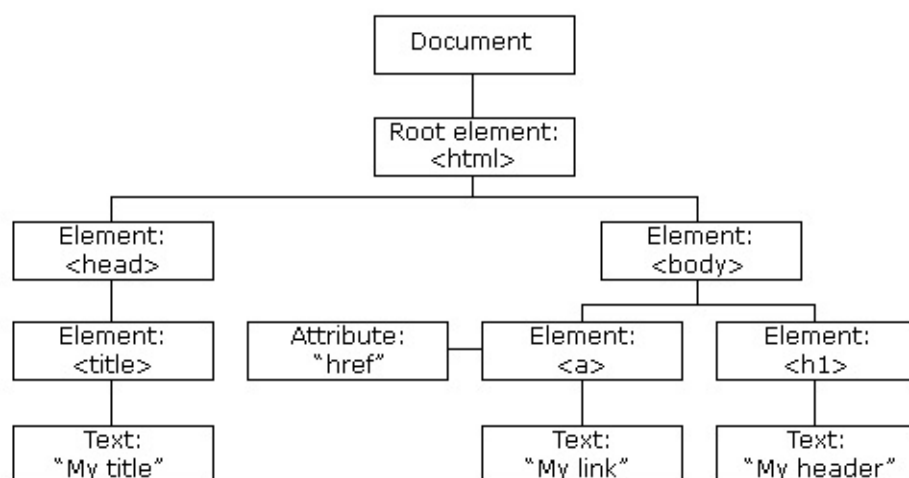
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.
3.  <bookstore>
4.
5.    <book category="cooking">
6.      <title lang="en">Everyday Italian</title>
7.      <author>Giada De Laurentiis</author>
8.      <year>2005</year>
9.      <price>30.00</price>
10.    </book>
11.
12.    <book category="children">
```

```

13.     <title lang="en">Harry Potter</title>
14.     <author>J K. Rowling</author>
15.     <year>2005</year>
16.     <price>29.99</price>
17. </book>
18.
19. <book category="web">
20.     <title lang="en">XQuery Kick Start</title>
21.     <author>James McGovern</author>
22.     <author>Per Bothner</author>
23.     <author>Kurt Cagle</author>
24.     <author>James Linn</author>
25.     <author>Vaidyanathan Nagarajan</author>
26.     <year>2003</year>
27.     <price>49.99</price>
28. </book>
29.
30. <book category="web" cover="paperback">
31.     <title lang="en">Learning XML</title>
32.     <author>Erik T. Ray</author>
33.     <year>2003</year>
34.     <price>39.95</price>
35. </book>
36.
37. </bookstore>

```

HTML DOM 模型示例



HTML DOM 定义了访问和操作 HTML 文档的标准方法，以树结构方式表达 HTML 文档。

XML的节点关系

1. 父 (Parent)

每个元素以及属性都有一个父。

下面是一个简单的XML例子中，book 元素是 title、author、year 以及 price 元素的父：

```
1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <book>
4.   <title>Harry Potter</title>
5.   <author>J K. Rowling</author>
6.   <year>2005</year>
7.   <price>29.99</price>
8. </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：

```
1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <book>
4.   <title>Harry Potter</title>
5.   <author>J K. Rowling</author>
6.   <year>2005</year>
7.   <price>29.99</price>
8. </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：

```
1. <?xml version="1.0" encoding="utf-8"?>
2.
3. <book>
4.   <title>Harry Potter</title>
5.   <author>J K. Rowling</author>
6.   <year>2005</year>
7.   <price>29.99</price>
```

```
8. </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：

```
1.  
2. <?xml version="1.0" encoding="utf-8"?>  
3.  
4. <bookstore>  
5.  
6. <book>  
7.   <title>Harry Potter</title>  
8.   <author>J K. Rowling</author>  
9.   <year>2005</year>  
10.  <price>29.99</price>  
11. </book>  
12.  
13. </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：

```
1. <?xml version="1.0" encoding="utf-8"?>  
2.  
3. <bookstore>  
4.  
5. <book>  
6.   <title>Harry Potter</title>  
7.   <author>J K. Rowling</author>  
8.   <year>2005</year>  
9.   <price>29.99</price>  
10. </book>  
11.  
12. </bookstore>
```

什么是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>

XPath 开发工具

1. 开源的XPath表达式编辑工具:XMLQuire(XML格式文件可用)
2. Chrome插件 XPath Helper
3. Firefox插件 XPath Checker

选取节点

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。

谓语句 (Predicates)

谓语句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，我们列出了带有谓语句的一些路径表达式，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。

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 or	

//book/price	选取 book 元素的所有 title 和 price 元素。
//title or //price	选取文档中的所有 title 和 price 元素。
/bookstore/book/title or //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的运算符

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	返回所有拥有 book 和 cd 元素的节点集
+	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

这些就是XPath的语法内容，在运用到Python抓取时要先转换为xml。

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）

初步使用

我们利用它来解析 HTML 代码，简单示例：

```

1. # lxml_test.py
2.
3. # 使用 lxml 的 etree 库
4. from lxml import etree
5.
6. text = '''
7. <div>
8.     <ul>
9.         <li class="item-0"><a href="link1.html">first item</a></li>
10.        <li class="item-1"><a href="link2.html">second item</a></li>
11.        <li class="item-inactive"><a href="link3.html">third item</a></li>
12.        <li class="item-1"><a href="link4.html">fourth item</a></li>
13.        <li class="item-0"><a href="link5.html">fifth item</a> # 注意，此处缺少
    一个 </li> 闭合标签
14.    </ul>
15. </div>
16. '''
17.
18. #利用etree.HTML，将字符串解析为HTML文档
19. html = etree.HTML(text)
20.
21. # 按字符串序列化HTML文档
22. result = etree.tostring(html)
23.
24. print(result)

```

输出结果：

```

1.
2. <html><body>

```

```

3.  <div>
4.      <ul>
5.          <li class="item-0"><a href="link1.html">first item</a></li>
6.          <li class="item-1"><a href="link2.html">second item</a></li>
7.          <li class="item-inactive"><a href="link3.html">third item</a></li>
8.          <li class="item-1"><a href="link4.html">fourth item</a></li>
9.          <li class="item-0"><a href="link5.html">fifth item</a></li>
10.     </ul>
11. </div>
12. </body></html>

```

lxml 可以自动修正 html 代码，例子里不仅补全了 li 标签，还添加了 body, html 标签。

文件读取：

除了直接读取字符串，lxml还支持从文件里读取内容。我们新建一个hello.html文件：

```

1.  <!-- hello.html -->
2.
3.  <div>
4.      <ul>
5.          <li class="item-0"><a href="link1.html">first item</a></li>
6.          <li class="item-1"><a href="link2.html">second item</a></li>
7.          <li class="item-inactive"><a href="link3.html"><span
8.              class="bold">third item</span></a></li>
9.          <li class="item-1"><a href="link4.html">fourth item</a></li>
10.         <li class="item-0"><a href="link5.html">fifth item</a></li>
11.     </ul>
12. </div>

```

再利用 etree.parse() 方法来读取文件。

```

1.  # lxml_parse.py
2.
3.  from lxml import etree
4.
5.  # 读取外部文件 hello.html
6.  html = etree.parse('./hello.html')
7.  result = etree.tostring(html, pretty_print=True)
8.
9.  print(result)

```

输出结果与之前相同：

```

1.
2. <html><body>
3. <div>
4.     <ul>
5.         <li class="item-0"><a href="link1.html">first item</a></li>
6.         <li class="item-1"><a href="link2.html">second item</a></li>
7.         <li class="item-inactive"><a href="link3.html">third item</a></li>
8.         <li class="item-1"><a href="link4.html">fourth item</a></li>
9.         <li class="item-0"><a href="link5.html">fifth item</a></li>
10.    </ul>
11. </div>
12. </body></html>

```

XPath实例测试

1. 获取所有的 li 标签

```

1. # xpath_li.py
2.
3. from lxml import etree
4.
5. html = etree.parse('hello.html')
6. print type(html) # 显示etree.parse() 返回类型
7.
8. result = html.xpath('//li')
9.
10. print result # 打印<li>标签的元素集合
11. print len(result)
12. print type(result)
13. print type(result[0])

```

输出结果：

```

1. <type 'lxml.etree._ElementTree'>
2. [<Element li at 0x1014e0e18>, <Element li at 0x1014e0ef0>, <Element li at
   0x1014e0f38>, <Element li at 0x1014e0f80>, <Element li at 0x1014e0fc8>]
3. 5
4. <type 'list'>
5. <type 'lxml.etree._Element'>

```

2. 继续获取 < li > 标签的所有 class属性

```

1. # xpath_li.py
2.
3. from lxml import etree
4.
5. html = etree.parse('hello.html')
6. result = html.xpath('//li/@class')
7.
8. print result

```

运行结果

```

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

```

3. 继续获取 < li >标签下href 为 link1.html 的 标签

```

1. # xpath_li.py
2.
3. from lxml import etree
4.
5. html = etree.parse('hello.html')
6. result = html.xpath('//li/a[@href="link1.html"]')
7.
8. print result

```

运行结果

```

1. [<Element a at 0x10ffaee18>]

```

4. 获取< li > 标签下的所有 标签

```

1. # xpath_li.py
2.
3. from lxml import etree
4.
5. html = etree.parse('hello.html')
6.
7. #result = html.xpath('//li/span')
8. #注意这么写是不对的：
9. #因为 / 是用来获取子元素的，而 <span> 并不是 <li> 的子元素，所以，要用双斜杠

```

```
10.  
11. result = html.xpath('//li//span')  
12.  
13. print result
```

运行结果

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

5. 获取 < li > 标签下的< a >标签里的所有 class

```
1. # xpath_li.py  
2.  
3. from lxml import etree  
4.  
5. html = etree.parse('hello.html')  
6. result = html.xpath('//li/a/@class')  
7.  
8. print result
```

运行结果

```
1. ['blod']
```

6. 获取最后一个 < li > 的 < a > 的 href

```
1. # xpath_li.py  
2.  
3. from lxml import etree  
4.  
5. html = etree.parse('hello.html')  
6.  
7. result = html.xpath('//li[last()]/a/@href')  
8. # 谓词 [last()] 可以找到最后一个元素  
9.  
10. print result
```

运行结果

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


7. 获取倒数第二个元素的内容

```
1. # xpath_li.py
2.
3. from lxml import etree
4.
5. html = etree.parse('hello.html')
6. result = html.xpath('//li[last()-1]/a')
7.
8. # text 方法可以获取元素内容
9. print result[0].text
```

运行结果

```
1. fourth item
```

8. 获取 class 值为 bold 的标签名

```
1. # xpath_li.py
2.
3. from lxml import etree
4.
5. html = etree.parse('hello.html')
6.
7. result = html.xpath('//*[@class="bold"]')
8.
9. # tag方法可以获取标签名
10. print result[0].tag
```

运行结果

```
1. span
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

CSS 选择器：BeautifulSoup4

和 `lxml` 一样，`Beautiful Soup` 也是一个HTML/XML的解析器，主要的功能也是如何解析和提取HTML/XML 数据。

`lxml` 只会局部遍历，而`Beautiful Soup` 是基于HTML DOM的，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，所以性能要低于`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。

`Beautiful Soup 3` 目前已经停止开发，推荐现在的项目使用`Beautiful Soup 4`。使用 `pip` 安装即可：
`pip install beautifulsoup4`

官方文档：http://beautifulsoup.readthedocs.io/zh_CN/v4.4.0

抓取工具	速度	使用难度	安装难度
正则	最快	困难	无（内置）
BeautifulSoup	慢	最简单	简单
lxml	快	简单	一般

示例：

首先必须要导入 `bs4` 库

```
1. # beautifulsoup4_test.py
2.
3. from bs4 import BeautifulSoup
4.
5. html = """
6. <html><head><title>The Dormouse's story</title></head>
7. <body>
8. <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
9. <p class="story">Once upon a time there were three little sisters; and their
   names were
10. <a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie -->
    </a>,
11. <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
12. <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
13. and they lived at the bottom of a well.</p>
14. <p class="story">...</p>
15. """
16.
17. #创建 Beautiful Soup 对象
```

```
18. soup = BeautifulSoup(html)
19.
20. #打开本地 HTML 文件的方式来创建对象
21. #soup = BeautifulSoup(open('index.html'))
22.
23. #格式化输出 soup 对象的内容
24. print soup.prettify()
```

运行结果:

```
1. <html>
2. <head>
3. <title>
4.   The Dormouse's story
5. </title>
6. </head>
7. <body>
8. <p class="title" name="dromouse">
9.   <b>
10.    The Dormouse's story
11.   </b>
12. </p>
13. <p class="story">
14.   Once upon a time there were three little sisters; and their names were
15.   <a class="sister" href="http://example.com/elsie" id="link1">
16.     <!-- Elsie -->
17.   </a>
18.   ,
19.   <a class="sister" href="http://example.com/lacie" id="link2">
20.     Lacie
21.   </a>
22.   and
23.   <a class="sister" href="http://example.com/tillie" id="link3">
24.     Tillie
25.   </a>
26.   ;
27. and they lived at the bottom of a well.
28. </p>
29. <p class="story">
30.   ...
31. </p>
32. </body>
```

```
33. </html>
```

- 如果我们在 IPython2 下执行，会看到这样一段警告：

```
/usr/local/lib/python2.7/site-packages/bs4/__init__.py:181: UserWarning: No parser was explicitly specified, so I'm using the best available HTML parser for this system ("lxml"). This usually isn't a problem, but if you run this code on another system, or in a different virtual environment, it may use a different parser and behave differently.
```

```
The code that caused this warning is on line 11 of the file /usr/local/bin/ipython2. To get rid of this warning, change code that looks like this:
```

```
BeautifulSoup([your markup])
```

- 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。

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 中的一个标签，例如：

```
1. <head><title>The Dormouse's story</title></head>
2. <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>
3. <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 BeautifulSoup 来获取 Tags：

```
1. from bs4 import BeautifulSoup
2.
3. html = """
```

```

4.  <html><head><title>The Dormouse's story</title></head>
5.  <body>
6.  <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
7.  <p class="story">Once upon a time there were three little sisters; and their
    names were
8.  <a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie -->
    </a>,
9.  <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
10. <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
11. and they lived at the bottom of a well.</p>
12. <p class="story">...</p>
13. ""
14.
15. #创建 BeautifulSoup 对象
16. soup = BeautifulSoup(html)
17.
18.
19. print soup.title
20. # <title>The Dormouse's story</title>
21.
22. print soup.head
23. # <head><title>The Dormouse's story</title></head>
24.
25. print soup.a
26. # <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
    </a>
27.
28. print soup.p
29. # <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
30.
31. print type(soup.p)
32. # <class 'bs4.element.Tag'>

```

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. print soup.name
2. # [document] #soup 对象本身比较特殊，它的 name 即为 [document]
3.

```

```

4. print soup.head.name
5. # head #对于其他内部标签，输出的值便为标签本身的名称
6.
7. print soup.p.attrs
8. # {'class': ['title'], 'name': 'dromouse'}
9. # 在这里，我们把 p 标签的所有属性打印输出出来了，得到的类型是一个字典。
10.
11. print soup.p['class'] # soup.p.get('class')
12. # ['title'] #还可以利用get方法，传入属性的名称，二者是等价的
13.
14. soup.p['class'] = "newClass"
15. print soup.p # 可以对这些属性和内容等等进行修改
16. # <p class="newClass" name="dromouse"><b>The Dormouse's story</b></p>
17.
18. del soup.p['class'] # 还可以对这个属性进行删除
19. print soup.p
20. # <p name="dromouse"><b>The Dormouse's story</b></p>

```

2. 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. print soup.p.string
2. # The Dormouse's story
3.
4. print type(soup.p.string)
5. # In [13]: <class 'bs4.element.NavigableString'>

```

3. BeautifulSoup

BeautifulSoup 对象表示的是一个文档的内容。大部分时候，可以把它当作 Tag 对象，是一个特殊的 Tag，我们可以分别获取它的类型，名称，以及属性来感受一下

```

1. print type(soup.name)
2. # <type 'unicode'>
3.
4. print soup.name
5. # [document]
6.
7. print soup.attrs # 文档本身的属性为空
8. # {}

```

4. 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。

```
1. print soup.a
2. # <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>
3.
4. print soup.a.string
5. # Elsie
6.
7. print type(soup.a.string)
8. # <class 'bs4.element.Comment'>
```

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。

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

```
1. print soup.head.contents
2. #[<title>The Dormouse's story</title>]
```

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

```
1. print soup.head.contents[0]
2. #<title>The Dormouse's story</title>
```

.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

```
1. print soup.head.children
2. #<listiterator object at 0x7f71457f5710>
3.
```

```

4. for child in soup.body.children:
5.     print child

```

结果:

```

1. <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
2.
3. <p class="story">Once upon a time there were three little sisters; and their
   names were
4. <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>,
5. <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
6. <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;
7. and they lived at the bottom of a well.</p>
8.
9. <p class="story">...</p>

```

2. 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。

```

1. for child in soup.descendants:
2.     print child

```

运行结果:

```

1. <html><head><title>The Dormouse's story</title></head>
2. <body>
3. <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
4. <p class="story">Once upon a time there were three little sisters; and their
   names were
5. <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>,
6. <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
7. <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;
8. and they lived at the bottom of a well.</p>
9. <p class="story">...</p>
10. </body></html>
11. <head><title>The Dormouse's story</title></head>
12. <title>The Dormouse's story</title>

```



```
13. The Dormouse's story
14.
15.
16. <body>
17. <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
18. <p class="story">Once upon a time there were three little sisters; and their
    names were
19. <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
    </a>,
20. <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
21. <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;
22. and they lived at the bottom of a well.</p>
23. <p class="story">...</p>
24. </body>
25.
26.
27. <p class="title" name="dromouse"><b>The Dormouse's story</b></p>
28. <b>The Dormouse's story</b>
29. The Dormouse's story
30.
31.
32. <p class="story">Once upon a time there were three little sisters; and their
    names were
33. <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
    </a>,
34. <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
35. <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;
36. and they lived at the bottom of a well.</p>
37. Once upon a time there were three little sisters; and their names were
38.
39. <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>
40. Elsie
41. ,
42.
43. <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
44. Lacie
45. and
46.
47. <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
48. Tillie
49. ;
50. and they lived at the bottom of a well.
```

```
51.
52.
53. <p class="story">...</p>
```

3. 节点内容：.string 属性

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. print soup.head.string
2. #The Dormouse's story
3. print soup.title.string
4. #The Dormouse's story
```

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, BeautifulSoup 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:

```
1. soup.find_all('b')
2. # [<b>The Dormouse's story</b>]
3.
4. print soup.find_all('a')
5. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
    </a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
    <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

B. 传正则表达式

如果传入正则表达式作为参数, BeautifulSoup 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

```
1. import re
2. for tag in soup.find_all(re.compile("^b")):
```

```
3.     print(tag.name)
4. # body
5. # b
```

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:

```
1. soup.find_all(["a", "b"])
2. # [<b>The Dormouse's story</b>,
3. # <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
4. # <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
5. # <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

2) keyword 参数

```
1. soup.find_all(class_ = "sister")
2. # [<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
   <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
3.
4. soup.find_all(id='link2')
5. # [<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

3) text 参数

通过 text 参数可以搜索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. soup.find_all(text="Elsie")
2. # [u'Elsie']
3.
4. soup.find_all(text=["Tillie", "Elsie", "Lacie"])
5. # [u'Elsie', u'Lacie', u'Tillie']
6.
7. soup.find_all(text=re.compile("Dormouse"))
8. [u"The Dormouse's story", u"The Dormouse's story"]
```

2. find

*find*的用法与*find_all*一样,区别在于*find*返回 第一个符合匹配结果, *find_all*则返回 所有匹配结果的

列表。

3. CSS选择器

这就是另一种与 `find_all` 方法有异曲同工之妙的查找方法，也是返回所有匹配结果的列表。

- 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. print soup.select('title')
2. #[<title>The Dormouse's story</title>]
3.
4. print soup.select('a')
5. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
   <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
6.
7. print soup.select('b')
8. #[<b>The Dormouse's story</b>]
```

(2) 通过类名查找

```
1. print soup.select('.sister')
2. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
   <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

(3) 通过 id 名查找

```
1. print soup.select('#link1')
2. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
   </a>]
```

(4) 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

```
1. print soup.select('p #link1')
2. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
```

```
</a>]
```

直接子标签查找，则使用 `>` 分隔

```
1. print soup.select("head > title")
2. #[<title>The Dormouse's story</title>]
```

(5) 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。

```
1. print soup.select('a[class="sister"]')
2. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
  </a>, <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
3.
4. print soup.select('a[href="http://example.com/elsie"]')
5. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
  </a>]
```

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. print soup.select('p a[href="http://example.com/elsie"]')
2. #[<a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie -->
  </a>]
```

(6) 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. soup = BeautifulSoup(html, 'lxml')
2. print type(soup.select('title'))
3. print soup.select('title')[0].get_text()
4.
5. for title in soup.select('title'):
6.     print title.get_text()
```

为什么要学正则表达式

实际上爬虫一共就四个主要步骤：

1. 明确目标（要知道你准备在哪个范围或者网站去搜索）
2. 爬（将所有的网站的内容全部爬下来）
3. 取（去掉对我们没用处的数据）
4. 处理数据（按照我们想要的方式存储和使用）

我们在昨天的案例里实际上省略了第3步，也就是“取”的步骤。因为我们down下了的数据是全部的网页，这些数据很庞大并且很混乱，大部分的东西使我们不关心的，因此我们需要将之按我们的需要过滤和匹配出来。

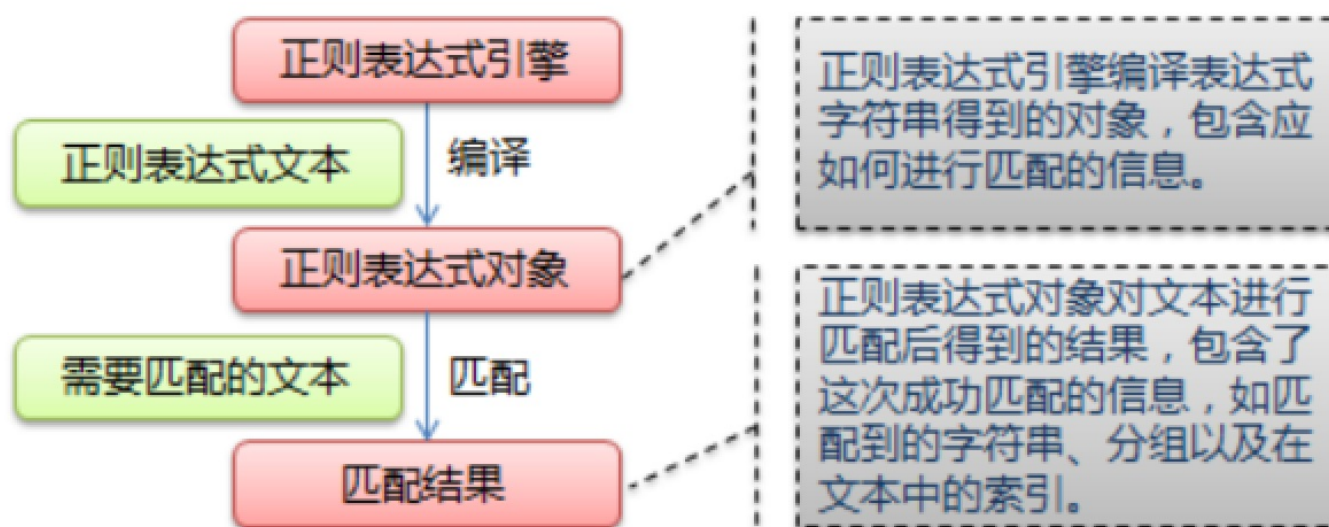
那么对于文本的过滤或者规则的匹配，最强大的就是正则表达式，是Python爬虫世界里必不可少的神兵利器。

什么是正则表达式

正则表达式，又称规则表达式，通常被用来检索、替换那些符合某个模式(规则)的文本。正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

给定一个正则表达式和另一个字符串，我们可以达到如下的目的：

- 给定的字符串是否符合正则表达式的过滤逻辑（“匹配”）；
- 通过正则表达式，从文本字符串中获取我们想要的特定部分（“过滤”）



正则表达式匹配规则

语法	说明	表达式实例	完整匹配的字符串
字符			
一般字符	匹配自身	abc	abc
.	匹配任意除换行符“\n”外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的寓意。 如果字符串中有字符“需要匹配”，可以使用“\”或者字符集“[]”。	a\c a\\c	a.c a\c
[...]	字符集（字符类），对应的位置可以是字符串中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用[]、-或^，可以在前面加上反斜杠，或把[]、-放在第一个字符，把^放在非第一个字符。	a[bcd]e	abc ace ade
预定义字符集（可以写在字符集[...]中）			
\d	数字：[0-9]	a\d c	a1c
\D	非数字：[^\d]	a\D c	abc
\s	空白字符：[<空格>\t\n\r\f\v]	a\s c	a c
\S	非空白字符：[^\s]	a\S c	abc
\w	单词字符：[A-Za-z0-9_]	a\w c	abc
\W	非单词字符：[^\w]	a\W c	a c
数量词（用在字符或[...]之后）			
*	匹配前一个字符0次或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
*? +? {m,n}?	使 * + ? {m,n} 变成非贪婪模式。	示例将在下文中介绍。	
边界匹配（不消耗待匹配字符串中的字符）			
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
\$	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	abc\$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配[w\d\W]之间。	a\bhc	ahbc
\B	[^\b]	a\Bhc	abc
逻辑、分组			
	代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则返回匹配右边的表达式。 如果 没有被包括在()中，则它的范围是整个正则表达式。	abc def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始遇到每个分组的左括号“(”，编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式中的 仅在该组中有效。	(abc){2} a(123 456)c	abcabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcbabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造（不作为分组）			
(?...)	(...)的不分组版本，用于使用 或后接数量词。	(?abc){2}	abcabc
(?i ms ux)	if ms ux 的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介。	(?i)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(? ...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(? \d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!=...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern [no-pattern])	如果编号为id/别名为name的组匹配到字符，则需要匹配 yes-pattern，否则要匹配no-pattern。 [no-pattern]可以省略。	(\d)abc(?(1)\d abc)	1abc2 abcabc

Python 的 re 模块

在 Python 中，我们可以使用内置的 re 模块来使用正则表达式。

有一点需要特别注意的是，正则表达式使用 对特殊字符进行转义，所以如果我们要使用原始字符串，只需加一个 r 前缀，示例：

```
r'chuanzhiboke\t\.\tpython'
```

re 模块的一般使用步骤如下：

1. 使用 compile() 函数将正则表达式的字符串形式编译为一个 Pattern 对象
2. 通过 Pattern 对象提供的一系列方法对文本进行匹配查找，获得匹配结果，一个 Match 对象。
3. 最后使用 Match 对象提供的属性和方法获得信息，根据需要进行其他的操作

compile 函数

compile 函数用于编译正则表达式，生成一个 Pattern 对象，它的一般使用形式如下：

```
1. import re
2.
3. # 将正则表达式编译成 Pattern 对象
4. pattern = re.compile(r'\d+')

```

在上面，我们已将一个正则表达式编译成 Pattern 对象，接下来，我们就可以利用 pattern 的一系列方法对文本进行匹配查找了。

Pattern 对象的一些常用方法主要有：

1. match 方法：从起始位置开始查找，一次匹配
2. search 方法：从任何位置开始查找，一次匹配
3. findall 方法：全部匹配，返回列表
4. finditer 方法：全部匹配，返回迭代器
5. split 方法：分割字符串，返回列表
6. sub 方法：替换

match 方法

match 方法用于查找字符串的头部（也可以指定起始位置），它是一次匹配，只要找到了一个匹配的结果就返回，而不是查找所有匹配的结果。它的一般使用形式如下：

```
match(string[, pos[, endpos]])
```

其中, string 是待匹配的字符串, pos 和 endpos 是可选参数, 指定字符串的起始和终点位置, 默认值分别是 0 和 len (字符串长度)。因此, 当你不指定 pos 和 endpos 时, match 方法默认匹配字符串的头部。

当匹配成功时, 返回一个 Match 对象, 如果没有匹配上, 则返回 None。

```
1. >>> import re
2. >>> pattern = re.compile(r'\d+') # 用于匹配至少一个数字
3.
4. >>> m = pattern.match('one12twothree34four') # 查找头部, 没有匹配
5. >>> print m
6. None
7.
8. >>> m = pattern.match('one12twothree34four', 2, 10) # 从'e'的位置开始匹配, 没有匹配
9. >>> print m
10. None
11.
12. >>> m = pattern.match('one12twothree34four', 3, 10) # 从'1'的位置开始匹配, 正好匹配
13. >>> print m # 返回一个 Match 对象
14. <_sre.SRE_Match object at 0x10a42aac0>
15.
16. >>> m.group(0) # 可省略 0
17. '12'
18. >>> m.start(0) # 可省略 0
19. 3
20. >>> m.end(0) # 可省略 0
21. 5
22. >>> m.span(0) # 可省略 0
23. (3, 5)
```

在上面, 当匹配成功时返回一个 Match 对象, 其中:

1. group([group1, ...]) 方法用于获得一个或多个分组匹配的字符串, 当要获得整个匹配的子串时, 可直接使用 group() 或 group(0);
2. start([group]) 方法用于获取分组匹配的子串在整个字符串中的起始位置 (子串第一个字符的索引), 参数默认值为 0;
3. end([group]) 方法用于获取分组匹配的子串在整个字符串中的结束位置 (子串最后一个字符的索引+1), 参数默认值为 0;

4. `span([group])` 方法返回 `(start(group), end(group))`。

再看看一个例子：

```

1. >>> import re
2. >>> pattern = re.compile(r'([a-z]+) ([a-z]+)', re.I) # re.I 表示忽略大小写
3. >>> m = pattern.match('Hello World Wide Web')
4.
5. >>> print m      # 匹配成功, 返回一个 Match 对象
6. <_sre.SRE_Match object at 0x10bea83e8>
7.
8. >>> m.group(0)    # 返回匹配成功的整个子串
9. 'Hello World'
10.
11. >>> m.span(0)     # 返回匹配成功的整个子串的索引
12. (0, 11)
13.
14. >>> m.group(1)    # 返回第一个分组匹配成功的子串
15. 'Hello'
16.
17. >>> m.span(1)     # 返回第一个分组匹配成功的子串的索引
18. (0, 5)
19.
20. >>> m.group(2)    # 返回第二个分组匹配成功的子串
21. 'World'
22.
23. >>> m.span(2)     # 返回第二个分组匹配成功的子串
24. (6, 11)
25.
26. >>> m.groups()    # 等价于 (m.group(1), m.group(2), ...)
27. ('Hello', 'World')
28.
29. >>> m.group(3)     # 不存在第三个分组
30. Traceback (most recent call last):
31.   File "<stdin>", line 1, in <module>
32. IndexError: no such group

```

search 方法

`search` 方法用于查找字符串的任何位置，它也是一次匹配，只要找到了一个匹配的结果就返回，而不是查找所有匹配的结果，它的一般使用形式如下：

```
search(string[, pos[, endpos]])
```

其中, string 是待匹配的字符串, pos 和 endpos 是可选参数, 指定字符串的起始和终点位置, 默认值分别是 0 和 len (字符串长度)。

当匹配成功时, 返回一个 Match 对象, 如果没有匹配上, 则返回 None。

让我们看看例子:

```
1. >>> import re
2. >>> pattern = re.compile('\d+')
3. >>> m = pattern.search('one12twothree34four') # 这里如果使用 match 方法则不匹配
4. >>> m
5. <_sre.SRE_Match object at 0x10cc03ac0>
6. >>> m.group()
7. '12'
8. >>> m = pattern.search('one12twothree34four', 10, 30) # 指定字符串区间
9. >>> m
10. <_sre.SRE_Match object at 0x10cc03b28>
11. >>> m.group()
12. '34'
13. >>> m.span()
14. (13, 15)
```

再来看一个例子:

```
1. # -*- coding: utf-8 -*-
2.
3. import re
4. # 将正则表达式编译成 Pattern 对象
5. pattern = re.compile(r'\d+')
6. # 使用 search() 查找匹配的子串, 不存在匹配的子串时将返回 None
7. # 这里使用 match() 无法成功匹配
8. m = pattern.search('hello 123456 789')
9. if m:
10.     # 使用 Match 获得分组信息
11.     print 'matching string:', m.group()
12.     # 起始位置和结束位置
13.     print 'position:', m.span()
```

执行结果:

1. matching string: 123456
2. position: (6, 12)

findall 方法

上面的 match 和 search 方法都是一次匹配，只要找到了一个匹配的结果就返回。然而，在大多数时候，我们需要搜索整个字符串，获得所有匹配的结果。

findall 方法的使用形式如下：

```
findall(string[, pos[, endpos]])
```

其中，string 是待匹配的字符串，pos 和 endpos 是可选参数，指定字符串的起始和终点位置，默认值分别是 0 和 len（字符串长度）。

findall 以列表形式返回全部能匹配的子串，如果没有匹配，则返回一个空列表。

看看例子：

```
1. import re
2. pattern = re.compile(r'\d+') # 查找数字
3.
4. result1 = pattern.findall('hello 123456 789')
5. result2 = pattern.findall('one1two2three3four4', 0, 10)
6.
7. print result1
8. print result2
```

执行结果：

1. ['123456', '789']
2. ['1', '2']

再先看一个栗子：

```
1.
2. # re_test.py
3. import re
4.
5. #re模块提供一个方法叫compile模块，提供我们输入一个匹配的规则
6. #然后返回一个pattern实例，我们根据这个规则去匹配字符串
7. pattern = re.compile(r'\d+\.\d*')
```

```

8.
9.  #通过partten.findall()方法就能够全部匹配到我们得到的字符串
10. result = pattern.findall("123.141593, 'bigcat', 232312, 3.15")
11.
12. #findall 以 列表形式 返回全部能匹配的子串给result
13. for item in result:
14.     print item

```

运行结果：

```

1.  123.141593
2.  3.15

```

finditer 方法

finditer 方法的行为跟 findall 的行为类似，也是搜索整个字符串，获得所有匹配的结果。但它返回一个顺序访问每一个匹配结果（Match 对象）的迭代器。

看看例子：

```

1.  # -*- coding: utf-8 -*-
2.
3.  import re
4.  pattern = re.compile(r'\d+')
5.
6.  result_iter1 = pattern.finditer('hello 123456 789')
7.  result_iter2 = pattern.finditer('one1two2three3four4', 0, 10)
8.
9.  print type(result_iter1)
10. print type(result_iter2)
11.
12. print 'result1...'
13. for m1 in result_iter1:  # m1 是 Match 对象
14.     print 'matching string: {}, position: {}'.format(m1.group(), m1.span())
15.
16. print 'result2...'
17. for m2 in result_iter2:
18.     print 'matching string: {}, position: {}'.format(m2.group(), m2.span())

```

执行结果：

```
1. <type 'callable-iterator'>
2. <type 'callable-iterator'>
3. result1...
4. matching string: 123456, position: (6, 12)
5. matching string: 789, position: (13, 16)
6. result2...
7. matching string: 1, position: (3, 4)
8. matching string: 2, position: (7, 8)
```

split 方法

split 方法按照能够匹配的子串将字符串分割后返回列表，它的使用形式如下：

```
split(string[, maxsplit])
```

其中，maxsplit 用于指定最大分割次数，不指定将全部分割。

看看例子：

```
1.
2. import re
3. p = re.compile(r'[\s\,\;\;]+')
4. print p.split('a,b;; c d')
```

执行结果：

```
1. ['a', 'b', 'c', 'd']
```

sub 方法

sub 方法用于替换。它的使用形式如下：

```
sub(repl, string[, count])
```

其中，repl 可以是字符串也可以是一个函数：

如果 repl 是字符串，则会使用 repl 去替换字符串每一个匹配的子串，并返回替换后的字符串，另外，repl 还可以使用 id 的形式来引用分组，但不能使用编号 0；

如果 repl 是函数，这个方法应当只接受一个参数（Match 对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。

count 用于指定最多替换次数，不指定时全部替换。

看看例子：

```

1. import re
2. p = re.compile(r'(\w+) (\w+)') # \w = [A-Za-z0-9_]
3. s = 'hello 123, hello 456'
4.
5. print p.sub(r'hello world', s) # 使用 'hello world' 替换 'hello 123' 和 'hello
    456'
6. print p.sub(r'\2 \1', s)      # 引用分组
7.
8. def func(m):
9.     return 'hi' + ' ' + m.group(2)
10.
11. print p.sub(func, s)
12. print p.sub(func, s, 1)      # 最多替换一次

```

执行结果：

```

1. hello world, hello world
2. 123 hello, 456 hello
3. hi 123, hi 456
4. hi 123, hello 456

```

匹配中文

在某些情况下，我们想匹配文本中的汉字，有一点需要注意的是，中文的 unicode 编码范围 主要在 [u4e00-u9fa5]，这里说主要是因为这个范围并不完整，比如没有包括全角（中文）标点，不过，在大部分情况下，应该是够用的。

假设现在想把字符串 title = u'你好，hello，世界' 中的中文提取出来，可以这么做：

```

1. import re
2.
3. title = u'你好，hello，世界'
4. pattern = re.compile(ur'[\u4e00-\u9fa5]+')
5. result = pattern.findall(title)
6.
7. print result

```

注意到，我们在正则表达式前面加上了两个前缀 ur，其中 r 表示使用原始字符串，u 表示是 unicode 字符串。

执行结果：

```
1. [u'\u4f60\u597d', u'\u4e16\u754c']
```

注意：贪婪模式与非贪婪模式

1. 贪婪模式：在整个表达式匹配成功的前提下，尽可能多的匹配（`*`）；
2. 非贪婪模式：在整个表达式匹配成功的前提下，尽可能少的匹配（`?`）；
3. **Python**里数量词默认是贪婪的。

示例一：源字符串：abbbc

- 使用贪婪的数量词的正则表达式 `ab*`，匹配结果：abbb。

`*`决定了尽可能多匹配 `b`，所以`a`后面所有的 `b` 都出现了。

- 使用非贪婪的数量词的正则表达式`ab*?`，匹配结果：a。

即使前面有 `*`，但是 `?` 决定了尽可能少匹配 `b`，所以没有 `b`。

示例二：源字符串：`aa<div>test1</div>bb<div>test2</div>cc`

- 使用贪婪的数量词的正则表达式：`<div>.*</div>`

- 匹配结果：`<div>test1</div>bb<div>test2</div>`

这里采用的是贪婪模式。在匹配到第一个`"</div>`时已经可以使整个表达式匹配成功，但是由于采用的是贪婪模式，所以仍然要向右尝试匹配，查看是否还有更长的可以成功匹配的子串。匹配到第二个`"</div>`后，向右再没有可以成功匹配的子串，匹配结束，匹配结果为“`<div>test1</div>bb<div>test2</div>`”

- 使用非贪婪的数量词的正则表达式：`<div>.*?</div>`

- 匹配结果：`<div>test1</div>`

正则表达式二采用的是非贪婪模式，在匹配到第一个`"</div>`时使整个表达式匹配成功，由于采用的是非贪婪模式，所以结束匹配，不再向右尝试，匹配结果为“`<div>test1</div>`”。

[正则表达式测试网址](#)

Redis 简介

Redis 是完全开源免费的，遵守BSD协议，是一个高性能的**key-value**数据库。

Redis 与其他 key-value 缓存产品有以下三个特点：

- Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份。

Redis 优势

- 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
- 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
- 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。
- 丰富的特性 – Redis还支持 publish/subscribe，通知，key 过期等等特性。

Redis与其他key-value存储有什么不同？

- Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
- Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

Copyright © 黑五电商学院 amzfriday.com all right reserved

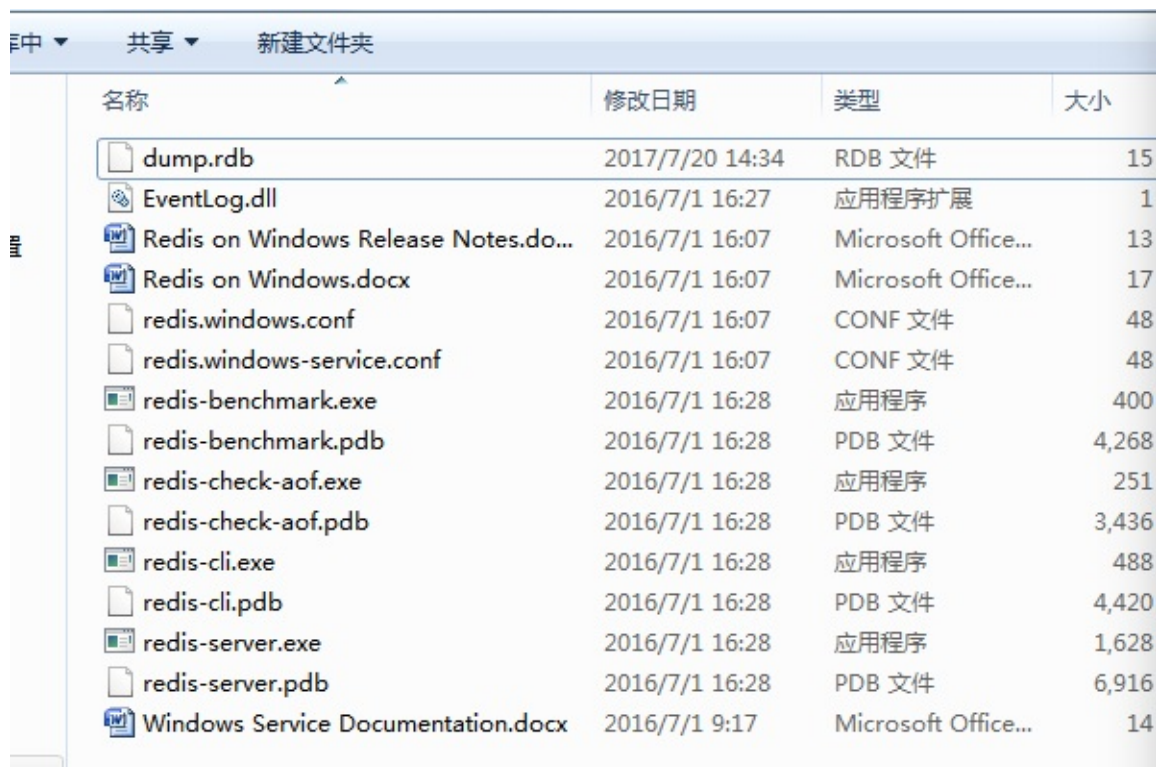
Redis 安装

Window 下安装

下载地址：<https://github.com/MSOpenTech/redis/releases>。

Redis 支持 32 位和 64 位。这个需要根据你系统平台的实际情况选择，这里我们下载 Redis-x64-100.zip压缩包到 C 盘，解压后，将文件夹重新命名为 redis。

打开文件夹，内容如下：



名称	修改日期	类型	大小
dump.rdb	2017/7/20 14:34	RDB 文件	15
EventLog.dll	2016/7/1 16:27	应用程序扩展	1
Redis on Windows Release Notes.docx	2016/7/1 16:07	Microsoft Office...	13
Redis on Windows.docx	2016/7/1 16:07	Microsoft Office...	17
redis.windows.conf	2016/7/1 16:07	CONF 文件	48
redis.windows-service.conf	2016/7/1 16:07	CONF 文件	48
redis-benchmark.exe	2016/7/1 16:28	应用程序	400
redis-benchmark.pdb	2016/7/1 16:28	PDB 文件	4,268
redis-check-aof.exe	2016/7/1 16:28	应用程序	251
redis-check-aof.pdb	2016/7/1 16:28	PDB 文件	3,436
redis-cli.exe	2016/7/1 16:28	应用程序	488
redis-cli.pdb	2016/7/1 16:28	PDB 文件	4,420
redis-server.exe	2016/7/1 16:28	应用程序	1,628
redis-server.pdb	2016/7/1 16:28	PDB 文件	6,916
Windows Service Documentation.docx	2016/7/1 9:17	Microsoft Office...	14

打开一个 cmd 窗口 使用 cd 命令切换目录到 C:\redis 运行：

```
redis-server.exe redis.windows.conf
```

这时候另启一个 cmd 窗口，原来的不要关闭，不然就无法访问服务端了。

切换到 redis 目录下运行：

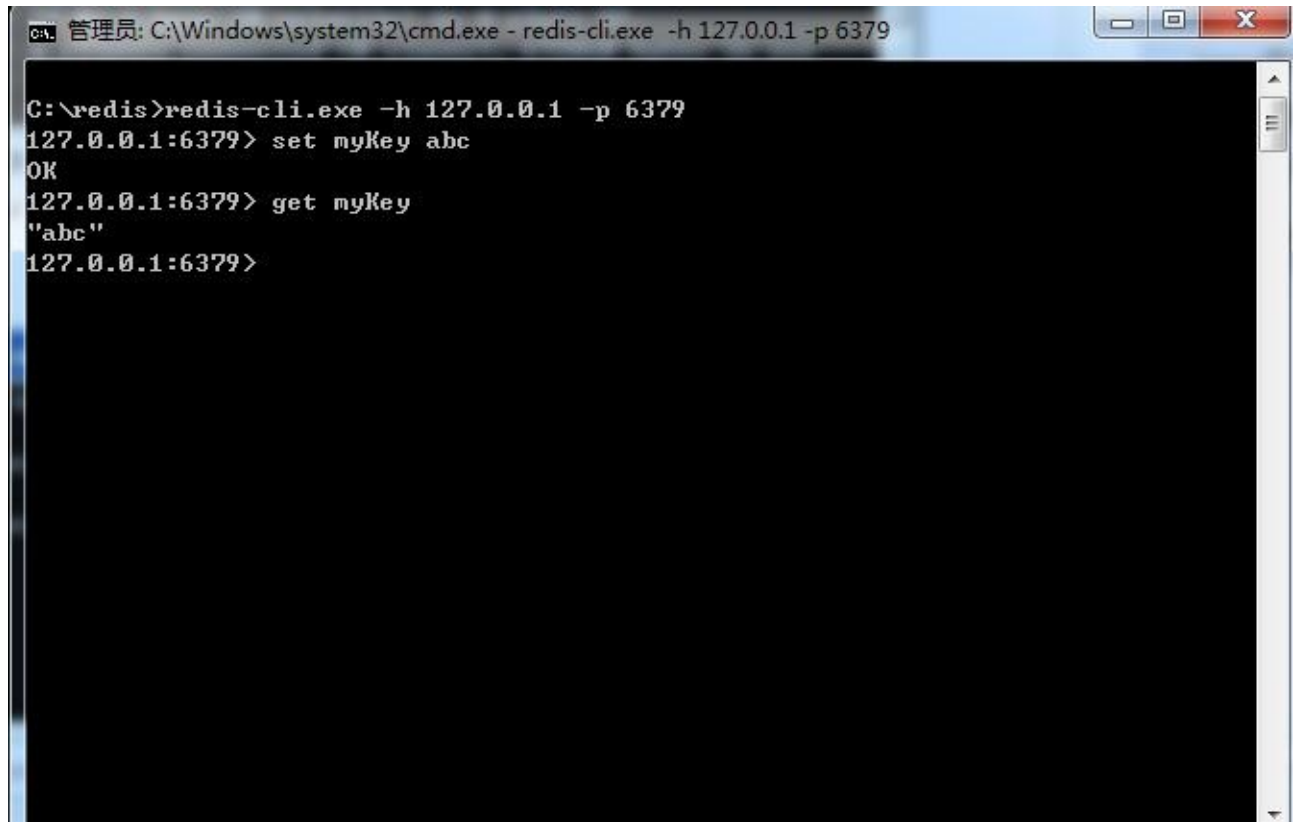
```
redis-cli.exe -h 127.0.0.1 -p 6379
```

设置键值对：

```
set myKey abc
```

取出键值对：

```
get myKey
```



```
管理员: C:\Windows\system32\cmd.exe - redis-cli.exe -h 127.0.0.1 -p 6379

C:\redis>redis-cli.exe -h 127.0.0.1 -p 6379
127.0.0.1:6379> set myKey abc
OK
127.0.0.1:6379> get myKey
"abc"
127.0.0.1:6379>
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

Python链接redis数据库

连接方式

redis-py提供两个类Redis和StrictRedis用于实现Redis的命令，StrictRedis用于实现大部分官方的命令，并使用官方的语法和命令，Redis是StrictRedis的子类。

```
1. import redis
2.
3. r = redis.Redis(host='192.168.0.110', port=6379, db=0)
4. r.set('name', 'zhangsan')    #添加
5. print (r.get('name'))        #获取
```

连接池

redis-py使用connection pool来管理对一个redis server的所有连接，避免每次建立、释放连接的开销。默认，每个Redis实例都会维护一个自己的连接池。可以直接建立一个连接池，然后作为参数Redis，这样就可以实现多个Redis实例共享一个连接池。

```
1. import redis
2.
3. pool = redis.ConnectionPool(host='192.168.0.110', port=6379)
4. r = redis.Redis(connection_pool=pool)
5. r.set('name', 'zhangsan')    #添加
6. print (r.get('name'))        #获取
```

管道

redis-py默认在执行每次请求都会创建（连接池申请连接）和断开（归还连接池）一次连接操作，如果想要在一次请求中指定多个命令，则可以使用pipeline实现一次请求指定多个命令，并且默认情况下一次pipeline 是原子性操作。

import redis

```
1. pool = redis.ConnectionPool(host='192.168.0.110', port=6379)
2. r = redis.Redis(connection_pool=pool)
3.
4. pipe = r.pipeline(transaction=True)
5.
```

```
6. r.set('name', 'zhangsan')
7. r.set('name', 'lisi')
8.
9. pipe.execute()
```

发布和订阅

首先定义一个RedisHelper类，连接Redis，定义频道为monitor，定义发布(publish)及订阅(subscribe)方法。

```
1. import redis
2.
3. class RedisHelper(object):
4.     def __init__(self):
5.         self.__conn = redis.Redis(host='192.168.0.110', port=6379) #连接Redis
6.         self.channel = 'monitor' #定义名称
7.
8.     def publish(self, msg): #定义发布方法
9.         self.__conn.publish(self.channel, msg)
10.        return True
11.
12.    def subscribe(self): #定义订阅方法
13.        pub = self.__conn.pubsub()
14.        pub.subscribe(self.channel)
15.        pub.parse_response()
16.        return pub
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

Python操作redis数据库

redis中的String在内存中按照一个name对应一个value来存储

set()

```
1. #在Redis中设置值，默认不存在则创建，存在则修改
2. r.set('name', 'zhangsan')
3. '''参数：
4.     set(name, value, ex=None, px=None, nx=False, xx=False)
5.     ex, 过期时间（秒）
6.     px, 过期时间（毫秒）
7.     nx, 如果设置为True，则只有name不存在时，当前set操作才执行，同setnx(name, value)
8.     xx, 如果设置为True，则只有name存在时，当前set操作才执行'''
```

```
1. setex(name, value, time)
2. #设置过期时间（秒）
3.
4. psetex(name, time_ms, value)
5. #设置过期时间（毫秒）
```

mset()

```
1. #批量设置值
2. r.mset(name1='zhangsan', name2='lisi')
3. #或
4. r.mset({"name1": 'zhangsan', "name2": 'lisi'})
```

get(name) 和 mget(keys, *args)

```
1. #批量获取
2. print(r.mget("name1", "name2"))
3. #或
4. li=["name1", "name2"]
5. print(r.mget(li))
```

getset(name, value)

```
1. #设置新值，打印原值
```

```
2. print(r.getset("name1", "wangwu")) #输出: zhangsan
3. print(r.get("name1")) #输出: wangwu
```

getrange(key, start, end)

```
1. #根据字节获取子序列
2. r.set("name", "zhangsan")
3. print(r.getrange("name", 0, 3)) #输出: zhan
```

setrange(name, offset, value)

```
1. #修改字符串内容，从指定字符串索引开始向后替换，如果新值太长时，则向后添加
2. r.set("name", "zhangsan")
3. r.setrange("name", 1, "z")
4. print(r.get("name")) #输出: zzangsan
5. r.setrange("name", 6, "zzzzzzz")
6. print(r.get("name")) #输出: zzangszzzzzzz
```

setbit(name, offset, value)

```
1. #对二进制表示位进行操作
2. ''' name:redis的name
3.     offset, 位的索引（将值对应的ASCII码转换成二进制后再进行索引）
4.     value, 值只能是 1 或 0 '''
5.
6. str="345"
7. r.set("name", str)
8. for i in str:
9.     print(i, ord(i), bin(ord(i))) #输出 值、ASCII码中对应的值、对应值转换的二进制
10. '''
```

输出：

```
1. 3 51 0b110011
2. 4 52 0b110100
3. 5 53 0b110101'''
4.
5. r.setbit("name", 6, 0) #把第7位改为0，也就是3对应的变成了0b110001
6. print(r.get("name")) #输出: 145
```


getbit(name, offset)

```

1.  #获取name对应值的二进制中某位的值(0或1)
2.  r.set("name", "3") # 对应的二进制0b110011
3.  print(r.getbit("name", 5))    #输出:0
4.  print(r.getbit("name", 6))    #输出:1

```

bitcount(key, start=None, end=None)

```

1.  #获取对应二进制中1的个数
2.  r.set("name", "345")#0b110011 0b110100 0b110101
3.  print(r.bitcount("name", start=0, end=1)) #输出:7
4.  ''' key:Redis的名称
5.      start:字节起始位置
6.      end:字节结束位置'''

```

strlen(name)

```

1.  **#返回name对应值的字节长度（一个汉字3个字节）
2.  r.set("name", "zhangsan")
3.  print(r strlen("name")) #输出:8

```

incr(self, name, amount=1)

```

1.  #自增mount对应的值，当mount不存在时，则创建mount = amount，否则，则自增，amount为自增数(整数)
2.  print(r.incr("mount", amount=2))#输出:2
3.  print(r.incr("mount"))#输出:3
4.  print(r.incr("mount", amount=3))#输出:6
5.  print(r.incr("mount", amount=6))#输出:12
6.  print(r.get("mount")) #输出:12

```

incrbyfloat(self, name, amount=1.0)

```

1.  #类似 incr() 自增，amount为自增数(浮点数)

```

decr(self, name, amount=1)

```

1.  #自减name对应的值，当name不存在时，则创建name = amount，否则，则自减，amount为自增数(整数)

```

append(name, value)

```
1. #在name对应的值后面追加内容
2. r.set("name", "zhangsan")
3. print(r.get("name"))    #输出:'zhangsan'
4. r.append("name", "lisi")
5. print(r.get("name"))    #输出:zhangsanlisi
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

爬虫(Spider)，反爬虫(Anti-Spider)，反反爬虫(Anti-Anti-Spider) 之间恢宏壮阔的斗争...

Day 1

- 小莫想要某站上所有的电影，写了标准的爬虫(基于`HttpClient`库)，不断地遍历某站的电影列表页面，根据 `Html` 分析电影名字存进自己的数据库。
- 这个站点的运维小黎发现某个时间段请求量陡增，分析日志发现都是 `IP(XXX.XXX.XXX.XXX)` 这个用户，并且 `user-agent` 还是 `Python-urllib/2.7`，基于这两点判断非人类后直接在服务器上封杀。

Day 2

- 小莫电影只爬了一半，于是也针对性的变换了下策略：1. `user-agent` 模仿百度 ("`Baiduspider...`")，2. `IP` 每爬半个小时就换一个 `IP` 代理。
- 小黎也发现了对应的变化，于是在服务器上设置了一个频率限制，每分钟超过120次请求的再屏蔽 `IP`。同时考虑到百度家的爬虫有可能会被误伤，想想市场部门每月几十万的投放，于是写了个脚本，通过 `hostname` 检查下这个 `ip` 是不是真的百度家的，对这些 `ip` 设置一个白名单。

Day 3

- 小莫发现了新的限制后，想着我也不急着要这些数据，留给服务器慢慢爬吧，于是修改了代码，随机1-3秒爬一次，爬10次休息10秒，每天只在8-12，18-20点爬，隔几天还休息一下。
- 小黎看着新的日志头都大了，再设定规则不小心会误伤真实用户，于是准备换了一个思路，当3个小时的总请求超过50次的时候弹出一个验证码弹框，没有准确正确输入的话就把 `IP` 记录进黑名单。

Day 4

- 小莫看到验证码有些傻眼了，不过也不是没有办法，先去学习了图像识别（关键词 `PIL`，`tesseract`），再对验证码进行了二值化，分词，模式训练之后，总之最后识别了小黎的验证码（关于验证码，验证码的识别，验证码的反识别也是一个恢弘壮丽的斗争史...），之后爬虫又跑了起来。
- 小黎是个不折不扣的好同学，看到验证码被攻破后，和开发同学商量了变化下开发模式，数据并不再直接渲染，而是由前端同学异步获取，并且通过 `JavaScript` 的加密库生成动态的 `token`，同时加密库再进行混淆（比较重要的步骤的确有网站这样做，参见淘宝和微博的登陆流程）。

Day 5

- 混淆过的加密库就没有办法了么？当然不是，可以慢慢调试，找到加密原理，不过小莫不准备用这么耗力耗力的方法，他放弃了基于 `HttpClient` 的爬虫，选择了内置浏览器引擎的爬虫(关键词：`PhantomJS`，`Selenium`)，在浏览器引擎运行页面，直接获取了正确的结果，又一次拿到了对方的数据。

- 小黎：.....

爬虫与反爬虫的斗争还在继续...

通常情况下，在爬虫与反爬虫的对弈中，爬虫一定会胜利。

换言之，只要人类能够正常访问的网页，爬虫在具备同等资源的情况下就一定可以抓取到。

关于爬虫部分一些建议：

尽量减少请求次数，能抓列表页就不抓详情页，减轻服务器压力，程序员都是混口饭吃不容易。

不要只看 Web 网站，还有手机 App 和 H5，这样的反爬虫措施一般比较少。

实际应用时候，一般防守方做到根据 IP 限制频次就结束了，除非很核心的数据，不会再进行更多的验证，毕竟成本的问题会考虑到。

如果真的对性能要求很高，可以考虑多线程，甚至分布式...

关于反爬虫部分的一些建议：

这篇文章就够了：[携程技术中心](#) - [携程酒店研发部研发经理崔广宇](#) <爬虫与反爬虫> 技术分享

Copyright © 黑五电商学院 [amzfriday.com](#) all right reserved

JavaScript

JavaScript 是网络上最常用也是支持者最多的客户端脚本语言。它可以收集 用户的跟踪数据,不需要重载页面直接提交表单,在页面嵌入多媒体文件,甚至运行网页游戏。

我们可以在网页源代码的标签里看到,比如:

```
<script type="text/javascript"
src="https://statics.huxiu.com/w/mini/static_2015/js/sea.js?v=201601150944"
```

jQuery

jQuery 是一个十分常见的库,70% 最流行的网站(约 200 万)和约 30% 的其他网站(约 2 亿)都在使用。一个网站使用 jQuery 的特征,就是源代码里包含了 jQuery 入口,比如:

```
<script type="text/javascript"
src="https://statics.huxiu.com/w/mini/static_2015/js/jquery-1.11.1.min.js?
v=201512181512"></script>
```

如果你在一个网站上看到了 jQuery,那么采集这个网站数据的时候要格外小心。jQuery 可以动态地创建 HTML 内容,只有在 JavaScript 代码执行之后才会显示。如果你用传统的方法采集页面内容,就只能获得 JavaScript 代码执行之前页面上的内容。

Ajax

我们与网站服务器通信的唯一方式,就是发出 HTTP 请求获取新页面。如果提交表单之后,或从服务器获取信息之后,网站的页面不需要重新刷新,那么你访问的网站就在用Ajax 技术。

Ajax 其实并不是一门语言,而是用来完成网络任务(可以认为 它与网络数据采集差不多)的一系列技术。Ajax 全称是 Asynchronous JavaScript and XML(异步 JavaScript 和 XML),网站不需要使用单独的页面请求就可以和网络服务器进行交互(收发信息)。

DHTML

Ajax 一样,动态 HTML(Dynamic HTML, DHTML)也是一系列用于解决网络问题的 技术集合。DHTML 是用客户端语言改变页面的 HTML 元素(HTML、CSS,或者二者皆 被改变)。比如页面上的按钮只有当用户移动鼠标之后才出现,背景色可能每次点击都会改变,或者用一个 Ajax 请求触发页面加载一段新内容,网页是否属于DHTML,关键要看有没有用 JavaScript 控制 HTML 和 CSS 元素。

那么,如何搞定?

那些使用了 Ajax 或 DHTML 技术改变 / 加载内容的页面，可能有一些采集手段。但是用 Python 解决这个问题只有两种途径：

1. 直接从 JavaScript 代码里采集内容（费时费力）
2. 用 Python 的 第三方库运行 JavaScript，直接采集你在浏览器里看到的页面（这个可以有）。

Copyright © 黑五电商学院 amzfriday.com all right reserved

Selenium

Selenium是一个Web的自动化测试工具，最初是为网站自动化测试而开发的，类型像我们玩游戏用的按键精灵，可以按指定的命令自动操作，不同是Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器（包括PhantomJS这些无界面的浏览器）。

Selenium 可以根据我们的指令，让浏览器自动加载页面，获取需要的数据，甚至页面截屏，或者判断网站上某些动作是否发生。

Selenium 自己不带浏览器，不支持浏览器的功能，它需要与第三方浏览器结合在一起才能使用。但是我们有时候需要让它内嵌在代码中运行，所以我们可以用一个叫 PhantomJS 的工具代替真实的浏览器。

可以从 PyPI 网站下载 Selenium库<https://pypi.python.org/simple/selenium> ,

也可以用 第三方管理器 pip用命令安装: `sudo pip install selenium`

Selenium 官方参考文档: <http://selenium-python.readthedocs.io/index.html>

PhantomJS

PhantomJS 是一个基于Webkit的“无界面”(headless)浏览器，它会把网站加载到内存并执行页面上的 JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器要高效。

如果我们把 Selenium 和 PhantomJS 结合在一起，就可以运行一个非常强大的网络爬虫了，这个爬虫可以处理 JavaScript、Cookie、headers，以及任何我们真实用户需要做的事情。

PhantomJS 是一个功能完善(虽然无界面)的浏览器而非一个 Python 库，所以它不需要像 Python 的其他库一样安装，但我们可以通过Selenium调用PhantomJS来直接使用。

PhantomJS 官方参考文档: <http://phantomjs.org/documentation>

安装示例：

- 下载PhantomJS：

```
wget https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-2.1.1-linux-x86_64.tar.bz2
```

- 解压并创建软连接：

```
tar -xvjf phantomjs-2.1.1-linux-x86_64.tar.bz2
sudo cp -R phantomjs-2.1.1-linux-x86_64 /usr/local/share/
sudo ln -sf /usr/local/share/phantomjs-2.1.1-linux-x86_64/bin/phantomjs
/usr/local/bin/
```

快速入门

Selenium 库里有个叫 WebDriver 的 API。WebDriver 有点儿像可以加载网站的浏览器，但是它也可以像 BeautifulSoup 或者其他 Selector 对象一样用来查找页面元素，与页面上的元素进行交互（发送文本、点击等），以及执行其他动作来运行网络爬虫。

```
1. # IPython2 测试代码
2.
3. # 导入 webdriver
4. from selenium import webdriver
5.
6. # 调用环境变量指定的PhantomJS浏览器创建浏览器对象
7. driver = webdriver.PhantomJS()
8.
9. # 如果没有在环境变量指定PhantomJS位置
10. # driver = webdriver.PhantomJS(executable_path="/usr/local/bin/phantomjs"))
11.
12. # get方法会一直等到页面被完全加载，然后才会继续程序，通常测试会在这里选择 time.sleep(2)
13. driver.get("http://www.baidu.com/")
14.
15. # 获取页面名为 wrapper的id标签的文本内容
16. data = driver.find_element_by_id("wrapper").text
17.
18. # 打印数据内容
19. print data
20.
21. # 打印页面标题 "百度一下，你就知道"
22. print driver.title
23.
24. # 生成当前页面快照并保存
25. driver.save_screenshot("baidu.png")
26.
27. # id="kw"是百度搜索输入框，输入字符串"长城"
28. driver.find_element_by_id("kw").send_keys(u"长城")
29.
30. # id="su"是百度搜索按钮，click() 是模拟点击
31. driver.find_element_by_id("su").click()
32.
33. # 获取新的页面快照
34. driver.save_screenshot("长城.png")
35.
36. # 打印网页渲染后的源代码
```



```
37. print driver.page_source
38.
39. # 获取当前页面Cookie
40. print driver.get_cookies()
41.
42. # 调用键盘按键操作时需要引入的Keys包
43. from selenium.webdriver.common.keys import Keys
44.
45. # ctrl+a 全选输入框内容
46. driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'a')
47.
48. # ctrl+x 剪切输入框内容
49. driver.find_element_by_id("kw").send_keys(Keys.CONTROL, 'x')
50.
51. # 输入框重新输入内容
52. driver.find_element_by_id("kw").send_keys("itcast")
53.
54. # 模拟Enter回车键
55. driver.find_element_by_id("su").send_keys(Keys.RETURN)
56.
57. # 清除输入框内容
58. driver.find_element_by_id("kw").clear()
59.
60. # 生成新的页面快照
61. driver.save_screenshot("itcast.png")
62.
63. # 获取当前url
64. print driver.current_url
65.
66. # 关闭当前页面，如果只有一个页面，会关闭浏览器
67. # driver.close()
68.
69. # 关闭浏览器
70. driver.quit()
```

页面操作

Selenium 的 WebDriver提供了各种方法来寻找元素，假设下面有一个表单输入框：

```
<input type="text" name="user-name" id="passwd-id" />
```

那么：

```

1. # 获取id标签值
2. element = driver.find_element_by_id("passwd-id")
3. # 获取name标签值
4. element = driver.find_element_by_name("user-name")
5. # 获取标签名值
6. element = driver.find_elements_by_tag_name("input")
7. # 也可以通过XPath来匹配
8. element = driver.find_element_by_xpath("//input[@id='passwd-id']")

```

定位UI元素 (WebElements)

关于元素的选取，有如下的API 单个元素选取

```

1. find_element_by_id
2. find_elements_by_name
3. find_elements_by_xpath
4. find_elements_by_link_text
5. find_elements_by_partial_link_text
6. find_elements_by_tag_name
7. find_elements_by_class_name
8. find_elements_by_css_selector

```

• By ID

```
<div id="coolestWidgetEvah">...</div>
```

实现

```

1. element = driver.find_element_by_id("coolestWidgetEvah")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. element = driver.find_element(by=By.ID, value="coolestWidgetEvah")

```

• By Class Name

```
<div class="cheese"><span>Cheddar</span></div><div class="cheese">
<span>Gouda</span></div>
```

实现

```

1. cheeses = driver.find_elements_by_class_name("cheese")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. cheeses = driver.find_elements(By.CLASS_NAME, "cheese")

```

• By Tag Name

```
<iframe src="..."></iframe>
```

实现

```

1. frame = driver.find_element_by_tag_name("iframe")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. frame = driver.find_element(By.TAG_NAME, "iframe")

```

• By Name

```
<input name="cheese" type="text"/>
```

实现

```

1. cheese = driver.find_element_by_name("cheese")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. cheese = driver.find_element(By.NAME, "cheese")

```

• By Link Text

```
<a href="http://www.google.com/search?q=cheese">cheese</a>
```

实现

```

1. cheese = driver.find_element_by_link_text("cheese")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. cheese = driver.find_element(By.LINK_TEXT, "cheese")

```

• By Partial Link Text

```
<a href="http://www.google.com/search?q=cheese">search for cheese</a>>
```

实现

```

1. cheese = driver.find_element_by_partial_link_text("cheese")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. cheese = driver.find_element(By.PARTIAL_LINK_TEXT, "cheese")

```

• By CSS

```
<div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></div>
```

实现

```

1. cheese = driver.find_element_by_css_selector("#food span.dairy.aged")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. cheese = driver.find_element(By.CSS_SELECTOR, "#food span.dairy.aged")

```

• By XPath

```
<input type="text" name="example" />
```

```
<INPUT type="text" name="other" />
```

实现

```
1. inputs = driver.find_elements_by_xpath("//input")
2. ----- or -----
3. from selenium.webdriver.common.by import By
4. inputs = driver.find_elements(By.XPATH, "//input")
```

鼠标动作链

有些时候，我们需要再页面上模拟一些鼠标操作，比如双击、右击、拖拽甚至按住不动等，我们可以通过导入 `ActionChains` 类来做到：

示例：


```
1. #导入 ActionChains 类
2. from selenium.webdriver import ActionChains
3.
4. # 鼠标移动到 ac 位置
5. ac = driver.find_element_by_xpath('element')
6. ActionChains(driver).move_to_element(ac).perform()
7.
8.
9. # 在 ac 位置单击
10. ac = driver.find_element_by_xpath("elementA")
11. ActionChains(driver).move_to_element(ac).click(ac).perform()
12.
13. # 在 ac 位置双击
14. ac = driver.find_element_by_xpath("elementB")
15. ActionChains(driver).move_to_element(ac).double_click(ac).perform()
16.
17. # 在 ac 位置右击
18. ac = driver.find_element_by_xpath("elementC")
19. ActionChains(driver).move_to_element(ac).context_click(ac).perform()
20.
21. # 在 ac 位置左键单击hold住
22. ac = driver.find_element_by_xpath('elementF')
23. ActionChains(driver).move_to_element(ac).click_and_hold(ac).perform()
24.
25. # 将 ac1 拖拽到 ac2 位置
```

```

26. ac1 = driver.find_element_by_xpath('elementD')
27. ac2 = driver.find_element_by_xpath('elementE')
28. ActionChains(driver).drag_and_drop(ac1, ac2).perform()

```

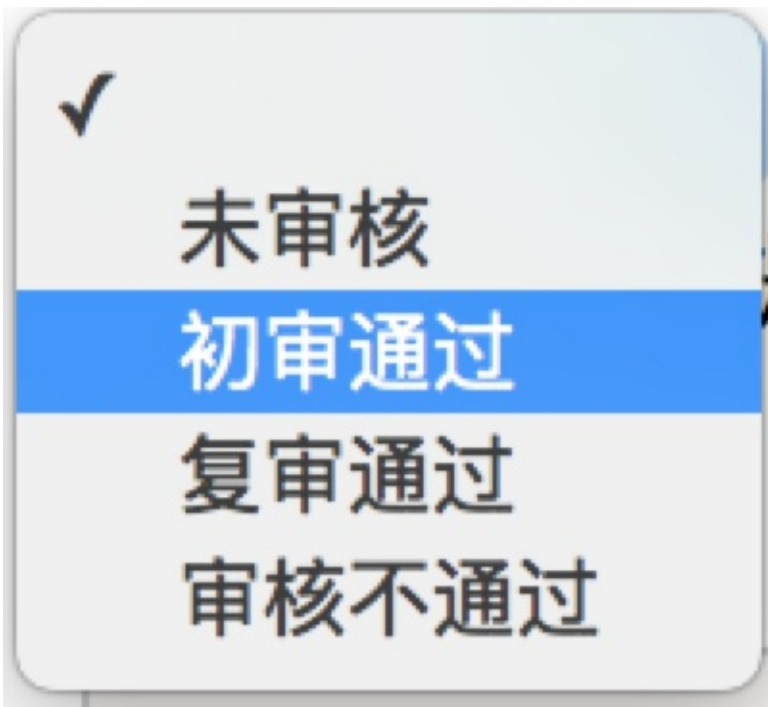
填充表单

我们已经知道了怎样向文本框中输入文字，但是有时候我们会碰到  标签的下拉框。直接点击下拉框中的选项不一定可行。

```

1. <select id="status" class="form-control valid" onchange="" name="status">
2.     <option value=""></option>
3.     <option value="0">未审核</option>
4.     <option value="1">初审通过</option>
5.     <option value="2">复审通过</option>
6.     <option value="3">审核不通过</option>
7. </select>

```



Selenium专门提供了Select类来处理下拉框。其实 WebDriver 中提供了一个叫 Select 的方法，可以帮助我们完成这些事情：

```

1. # 导入 Select 类
2. from selenium.webdriver.support.ui import Select
3.
4. # 找到 name 的选项卡
5. select = Select(driver.find_element_by_name('status'))
6.

```

```

7.  #
8.  select.select_by_index(1)
9.  select.select_by_value("0")
10. select.select_by_visible_text(u"未审核")

```

以上是三种选择下拉框的方式，它可以根据索引来选择，可以根据值来选择，可以根据文字来选择。注意：

- *index* 索引从 0 开始
- *value* 是 *option* 标签的一个属性值，并不是显示在下拉框中的值
- *visible_text* 是在 *option* 标签文本的值，是显示在下拉框的值

全部取消选择怎么办呢？很简单：

```
select.deselect_all()
```

弹窗处理

当你触发了某个事件之后，页面出现了弹窗提示，处理这个提示或者获取提示信息方法如下：

```
alert = driver.switch_to_alert()
```

页面切换

一个浏览器肯定会有很多窗口，所以我们肯定要有方法来实现窗口的切换。切换窗口的方法如下：

```
driver.switch_to.window("this is window name")
```

也可以使用 `window_handles` 方法来获取每个窗口的操作对象。例如：

```

1. for handle in driver.window_handles:
2.     driver.switch_to_window(handle)

```

页面前进和后退

操作页面的前进和后退功能：

```

1. driver.forward()    #前进
2. driver.back()       # 后退

```

Cookies

获取页面每个Cookies值，用法如下

```
1. for cookie in driver.get_cookies():
2.     print "%s=%s;" % (cookie['name'], cookie['value'])
```

删除Cookies，用法如下

```
1.
2. # By name
3. driver.delete_cookie("BAIDUID")
4.
5. # all
6. driver.delete_all_cookies()
```

页面等待

现在的网页越来越多采用了 Ajax 技术，这样程序便不能确定何时某个元素完全加载出来了。如果实际页面等待时间过长导致某个dom元素还没出来，但是你的代码直接使用了这个WebElement，那么就会抛出NullPointerException的异常。

为了避免这种元素定位困难而且会提高产生 ElementNotVisibleException 的概率。所以 Selenium 提供了两种等待方式，一种是隐式等待，一种是显式等待。

隐式等待是等待特定的时间，显式等待是指定某一条件直到这个条件成立时继续执行。

显式等待

显式等待指定某个条件，然后设置最长等待时间。如果在这个时间还没有找到元素，那么便会抛出异常了。

```
1. from selenium import webdriver
2. from selenium.webdriver.common.by import By
3. # WebDriverWait 库，负责循环等待
4. from selenium.webdriver.support.ui import WebDriverWait
5. # expected_conditions 类，负责条件出发
6. from selenium.webdriver.support import expected_conditions as EC
7.
8. driver = webdriver.PhantomJS()
9. driver.get("http://www.xxxxx.com/loading")
10. try:
11.     # 每隔10秒查找页面元素 id="myDynamicElement", 直到出现则返回
```

```
12.     element = WebDriverWait(driver, 10).until(  
13.         EC.presence_of_element_located((By.ID, "myDynamicElement"))  
14.     )  
15. finally:  
16.     driver.quit()
```

如果不写参数，程序默认会 0.5s 调用一次来查看元素是否已经生成，如果本来元素就是存在的，那么会立即返回。

下面是一些内置的等待条件，你可以直接调用这些条件，而不用自己写某些等待条件了。

```
1. title_is  
2. title_contains  
3. presence_of_element_located  
4. visibility_of_element_located  
5. visibility_of  
6. presence_of_all_elements_located  
7. text_to_be_present_in_element  
8. text_to_be_present_in_element_value  
9. frame_to_be_available_and_switch_to_it  
10. invisibility_of_element_located  
11. element_to_be_clickable - it is Displayed and Enabled.  
12. staleness_of  
13. element_to_be_selected  
14. element_located_to_be_selected  
15. element_selection_state_to_be  
16. element_located_selection_state_to_be  
17. alert_is_present
```

隐式等待

隐式等待比较简单，就是简单地设置一个等待时间，单位为秒。

```
1. from selenium import webdriver  
2.  
3. driver = webdriver.PhantomJS()  
4. driver.implicitly_wait(10) # seconds  
5. driver.get("http://www.xxxxx.com/loading")  
6. myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

当然如果不设置，默认等待时间为0。

案例：模拟登录亚马逊

```
1. from selenium import webdriver
2. from selenium.webdriver.common.keys import Keys
3. import time
4.
5. # 创建webdriver
6. browser = webdriver.PhantomJS()
7. browser.get("https://www.amazon.com")
8.
9. # 点击主页登陆按钮
10. browser.find_element_by_xpath('//*[@id="nav-link-accountList"]').click()
11. # 跳转到登陆页面
12. # 输入账号密码
13. browser.find_element_by_xpath('//*[@id="ap_email"]').send_keys("username@mail.com")
14. browser.find_element_by_xpath('//*[@id="ap_password"]').send_keys('password')
15. # 点击登陆按钮
16. browser.find_element_by_xpath('//*[@id="signInSubmit"]').click()
17.
18. # 等待3秒
19. time.sleep(3)
20.
21. # 生成登陆后快照
22. browser.save_screenshot("amazon.png")
23.
24. # 保存源码
25. with open("amazon.html", "w") as file:
26.     file.write(driver.page_source)
27.
28. browser.quit()
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

案例二：动态页面模拟点击

爬取斗鱼直播平台的所有房间信息：

```
1.  #!/usr/bin/env python
2.  # -*- coding:utf-8 -*-
3.
4.  # python的测试模块
5.  import unittest
6.  from selenium import webdriver
7.  from bs4 import BeautifulSoup
8.
9.
10. class douyuSelenium(unittest.TestCase):
11.     # 初始化方法
12.     def setUp(self):
13.         self.driver = webdriver.PhantomJS()
14.
15.     #具体的测试用例方法，一定要以test开头
16.     def testDouyu(self):
17.         self.driver.get('http://www.douyu.com/directory/all')
18.         while True:
19.             # 指定xml解析
20.             soup = BeautifulSoup(driver.page_source, 'xml')
21.             # 返回当前页面所有房间标题列表 和 观众人数列表
22.             titles = soup.find_all('h3', {'class': 'ellipsis'})
23.             nums = soup.find_all('span', {'class': 'dy-num fr'})
24.
25.             # 使用zip()函数来可以把列表合并，并创建一个元组对的列表[(1,2), (3,4)]
26.             for title, num in zip(nums, titles):
27.                 print u"观众人数:" + num.get_text().strip(), u"\t房间标题: " +
title.get_text().strip()
28.             # 指定元素找到则返回 非-1，表示到达最后一页，退出循环
29.             if driver.page_source.find('shark-pager-disable-next') != -1:
30.                 break
31.             # 模拟下一页点击
32.             self.driver.find_element_by_class_name('shark-pager-next').click()
33.
34.     # 退出时的清理方法
35.     def tearDown(self):
36.         print '加载完成...'
```

```
37.         self.driver.quit()
38.
39. if __name__ == "__main__":
40.     unittest.main()
```

Copyright © 黑五电商学院 amzfriday.com all right reserved

案例三：执行 JavaScript 语句

- 隐藏百度图片

```

1.  from selenium import webdriver
2.
3.  driver = webdriver.PhantomJS()
4.  driver.get("https://www.baidu.com/")
5.
6.  # 给搜索输入框标红的javascript脚本
7.  js = "var q=document.getElementById(\"kw\");q.style.border=\"2px solid red\";"
8.
9.  # 调用给搜索输入框标红js脚本
10. driver.execute_script(js)
11.
12. #查看页面快照
13. driver.save_screenshot("redbaidu.png")
14.
15. #js隐藏元素，将获取的图片元素隐藏
16. img = driver.find_element_by_xpath("//*[@id='lg']/img")
17. driver.execute_script('$(arguments[0]).fadeOut()',img)
18.
19. # 向下滚动到页面底部
20. driver.execute_script("$('.scroll_top').click(function()
    {'$(html,body').animate({scrollTop: '0px'}, 800);});")
21.
22. #查看页面快照
23. driver.save_screenshot("nullbaidu.png")
24.
25. driver.quit()

```

- 模拟滚动条滚动到底部

```

1.  from selenium import webdriver
2.  import time
3.
4.  driver = webdriver.PhantomJS()
5.  driver.get("https://movie.douban.com/typerank?type_name=剧情
    &type=11&interval_id=100:90&action=")
6.

```

```
7. # 向下滚动10000像素
8. js = "document.body.scrollTop=10000"
9. #js="var q=document.documentElement.scrollTop=10000"
10. time.sleep(3)
11.
12. #查看页面快照
13. driver.save_screenshot("douban.png")
14.
15. # 执行JS语句
16. driver.execute_script(js)
17. time.sleep(10)
18.
19. #查看页面快照
20. driver.save_screenshot("newdouban.png")
21.
22. driver.quit()
```

Copyright © 黑五电商学院 amzfriday.com all right reserved