

Python

网络爬虫讲义

高级爬虫系列一

主讲老师：赵俊

内容简介

本讲义为高级爬虫课程的一部分。通过对各种爬虫的主流技术进行研究得出了关于目前网络爬虫所遇到的问题与解决方案进行了较为详细的阐述。在实例中，选用了目前国内主流的豆瓣，猫眼电影，今日头条等进行实际的数据抓取，但是随着时间流逝，目标网站的更新，可能有部分代码无法正常运行，请及时联系 g-zhaojun@tedu.cn 进行改进。本讲义研究内容和组织结构按照如下编排：

第一章：爬虫简介，简单介绍了什么是爬虫，爬虫的作用，及如何实现一个爬虫，爬虫所需要的先决知识及关键核心技术等等，分级别的告诉大家初级爬虫工程师，中级爬虫工程师及高级爬虫工程师所需要的知识体系及技能。

第二章：介绍了关于爬虫的一些基本常识，包括爬虫的法律风险等，之后介绍了怎么对需要爬取的网站进行背景调研：网站的 Robots 协议，Sitemap，怎么估算网站的大小，怎么识别网站用了什么技术架构，怎么寻找网站的所有者等等，这些都是在抓取一个网站之前所需要做的背景调研，以便我们更好的能够抓取到网站的信息。

第三章：通过实例给大家展示在 Python 中一个简单的爬虫应该怎么写，同时简单复习一下爬虫的相关网络知识，主要是 HTTP, HTTPS 协议，最后介绍了爬虫爬取数据两种常用算法策略：广度优先和深度优先的策略，及实际项目如何选择策略。

第四章：介绍了怎么对爬取到的内容进行提取，对结构化的数据与非结构化的数据分别怎么进行提取，常见的文件格式 JSON, XML, HTML 等，及 BeautifulSoup, lxml, 正则表达式等等提取数据的方法，并对各种数据，数据解析方法及使用场景进行了总结。

第五章：介绍了针对动态网页数据的抓取，可以采用的策略：模拟网络数据包的过程及使用内置浏览器的方案 Selenium + PhantomJS 的策略，分别用示例说明了这种方法各自的优缺点。

第六章：简单介绍下讲述怎么登录，提交表单，怎么使用 Cookie 记录管理登录状态，怎么使用图像识别技术来尝试破解验证码。

第七章：讲述了怎么存储爬取到的数据，怎么做持久化，如何选择及使用 MySQL, MongoDB, HBase 等数据库进行存储，简单介绍一下怎么对数据进行压缩存储，以节省存储空间。

第八章：高效爬虫之路，怎么使用多线程，多进程来提升爬虫爬取数据的效率，简单介绍下怎么来调度并行爬虫，怎么设置代理服务器，来防止爬虫在告诉爬取数据时尽

可能不被反爬程序发现。

第九章：简单介绍下怎么去重，怎么使用 **Redis** 数据库进行去重，大数据量时怎么使用 **HASH** 算法 **Bloom Filter** 减少空间消耗。

第十章：介绍 **Scrapy** 框架，如何快速使用 **Scrapy** 搭建一个网站的爬虫，如何使用 **Scrapy-Redis** 快速搭建一个分布式爬虫的框架。

第十一章：介绍下常用的反爬策略，怎么解决 **HTTPS** 对爬虫的阻碍行为。

第十二章：其他，简单介绍下如果使用机器学习的方法优化爬虫爬取数据的策略，以及如果使用爬虫爬取的数据，对于爬虫的时效性问题，怎么设计其更新的频率，怎么尝试使用机器学习的方法调节其自动更新的频率。

目 录

内容简介	II
第一章 爬虫简介	7
1.1 什么是网络爬虫	7
1.1.1 爬虫的简单定义	7
1.1.2 爬虫的分类	7
1.2 为什么需要爬虫	8
1.2.1 爬虫的用途	8
1.2.2 怎么做爬虫	10
第二章 爬虫的基本常识	13
2.1 爬虫的合法性问题	13
2.2 爬虫的准备工作：网站的背景调研	13
2.2.1 robots 协议	13
2.2.2 网站地图 sitemap	14
2.2.3 估算网站的大小	14
2.2.4 识别网站用了何种技术	15
第三章 简单爬虫的实现	17
3.1 可能是史上最简单的爬虫 DEMO	17
3.2 回顾一下 HTTP, HTTPS 协议	17
3.3 关于爬虫抓取策略	23
3.3.1 深度优先算法	23
3.3.2 广度/宽度优先算法	23
3.4.3 实践中怎么来组合抓取策略	25
第四章 提取网页中的信息	26
4.1 数据的类型	26
4.2 关于 XML,HTML,DOM 和 JSON 文件	27
4.2.1 XML,HTML,DOM	27
4.2.2 JSON 文件	30
4.3 怎么提取网页中的信息	35
4.3.1 XPath 与 lxml	35
4.3.2 BeautifulSoup4	38
4.3.3 正则表达式 re	40

第五章 动态网页的挑战	43
5.1 动态网页的使用场景	43
5.2 回到与 HTTP 服务器发送请求数据的原始方法	43
5.2.1 GET 方法	43
5.2.2 POST 方法	45
5.3 更加难以对付的动态网站	46
5.3.1 应对需要多次数据的交互模拟的网站	46
5.3.2 Selenium	46
5.3.3 PhantomJS	46
5.3.4 Selenium + PhantomJS	48
5.4 关于动态网站信息抓取的总结	48
第六章 表单与爬虫登录问题	49
6.1 关于表单	49
6.2 管理 COOKIE	49
6.2.1 使用 cookie 登录	49
6.2.2 ## 补充知识 cookiejar 的使用	50
6.3 关于验证码 (CAPTCHA)	51
第七章 爬虫的持久化问题	53
7.1 MySQL	53
7.2 MONGODB	53
7.2.1 什么是 MongoDB	53
7.2.2 怎么在爬虫中使用 MongoDB	53
7.3 HDFS, HBASE	53
第八章 高效率的爬取数据	54
8.1 多进程爬虫	54
8.2 多线程爬虫	54
8.2.1 关于 GIL	54
8.2.2 线程池的出场	54
8.3 关于代理服务器的设置	54
第九章 大数据量时的去重	56
9.1 怎么去重	56
9.2 REDIS 数据库	56
9.2.1 关于 Redis	56

9.2.2 实际项目中使用 Redis	56
9.3 BLOOMFILTER	56
第十章 SCRAPY 框架.....	57
10.1 什么是 SCRAPY	57
10.2 怎么安装使用 SCRAPY	58
10.2.1 安装	58
10.2.2 制作一个 Scrapy 爬虫需要的四个步骤	58
第十一章 反爬及应对反爬的策略	61
11.1 网站如何发现爬虫	61
11.2 网站如何进行反爬	61
11.3 爬虫如何发现自己可能被网站识别了	61
11.4 爬虫应对反爬的策略	61
总结与进一步工作	63
附录 A 收集到的 100 个可能能用的代理服务器.....	64
附录 B PYTHON2 与 3 URLLIB 库对照表	67

第一章 爬虫简介

1.1 什么是网络爬虫

1.1.1 爬虫的简单定义

网络爬虫（又被称为网页蜘蛛，网络机器人，在 FOAF 社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。另外一些不常使用的名字还有蚂蚁、自动索引、模拟程序或者蠕虫。

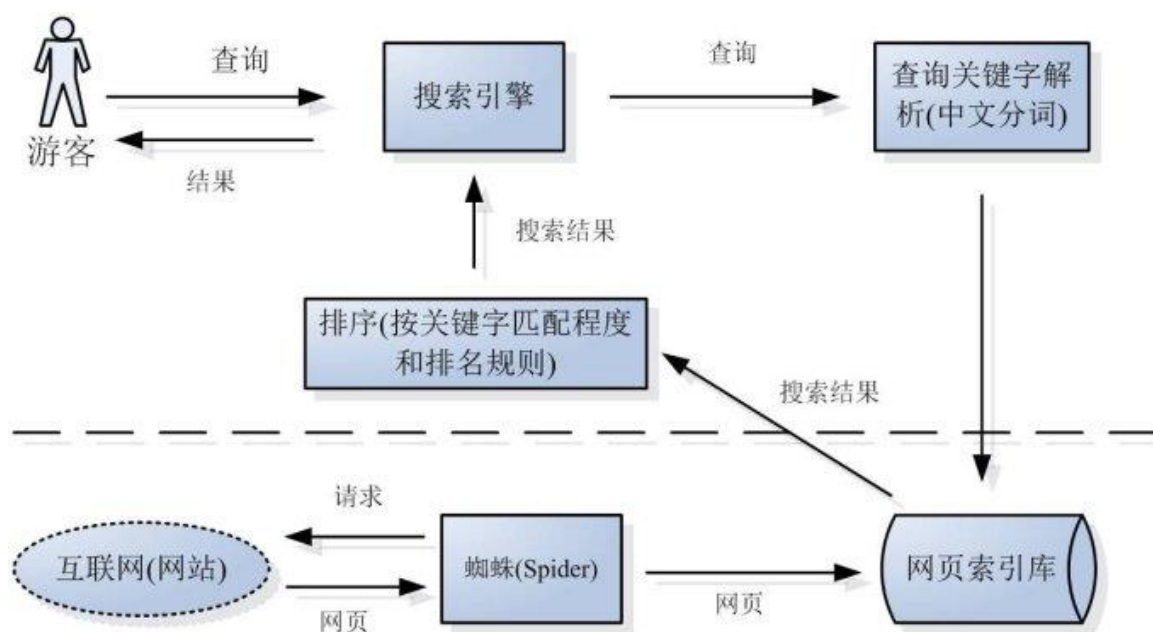
简单来说：**使用事先写好的程序去抓取网络上所需要的数据，这样的程序就叫网络爬虫。**

1.1.2 爬虫的分类

网络爬虫可以分为通用网络爬虫(如搜索引擎的爬虫，根据几个 URL 的种子不断的去抓取数据)和聚焦网络爬虫(有选择性的抓取预先定义好的主题和相关页面的网络爬虫)。

A.通用网络爬虫：

搜索引擎中第一步就是爬虫。但是搜索引擎中的爬虫是一种广泛获取各种网页的信息的程序；除了 HTML 文件外，搜索引擎通常还会抓取和索引文字为基础的多种文件类型，如 TXT，WORD，PDF 等；但是对于图片，视频，等非文字的内容则一般不会处理；但是对于脚本和一些网页中的程序是不会处理的；



B. 聚焦网络爬虫:

针对某一特定领域的数据进行抓取的程序。比如旅游网站，金融网站，招聘网站等等；特定领域的聚焦爬虫会使用各种技术去处理我们需要的信息，所以对于网站中动态的一些程序，脚本仍会执行，以保证确定能抓取到网站中的数据；

1.2 为什么需要爬虫

1.2.1 爬虫的用途

A. 解决冷启动问题：对于很多社交类的网站，冷启动是很困难的。对于新注册的用户而言，要留住他们，需要先注入一批假用户，已构造社区的氛围。一般这些假的用户可以通过网络爬虫从微博或其他 APP 中抓取而来；今日头条等互联网媒体最早也就是使用了爬虫+网页排序的技术，所以它们解决冷启动的方式也是需要爬虫；



B. 搜索引擎的根基：做搜索引擎少不了爬虫程序；

C. 建立起知识图谱，帮助建立机器学习的训练集：



D.可以制作各种商品的比价，趋势分析等：



E.其他：比如分析淘宝上竞争对手的数据；分析微博的数据传递影响力，政府的舆情分析，分析人与人之间的关系等等；



首页 > 陈赫



陈赫

Michael C. Michael Chen

职业: 演员 生日: 1985-11-09 地区: 中国大陆
身高: 182cm 体重: 73Kg 血型: O型

陈赫 (1985年11月9日—), 中国大陆男演员, 上海戏剧学院表演系2004级本科毕业。因出演电视剧《爱情公寓》的“新好男人”曾小贤一角而被广大影迷所喜爱, 阳光帅气的形象深入人心, 是娱乐圈冉冉升起的新星, 有望成为新一代笑星... 更多>

推荐作品



奔跑吧兄弟春节特辑



医馆笑传



奔跑吧兄弟第一季

分享

推荐 影视作品 音乐作品 相关资讯 资料

电视剧

更多>



医馆笑传
年代: 2015
喜剧 古装



天和贴吧
年代: 2015
搞笑



爱情回来了
年代: 2014
偶像 言情 爱情



爱情回来了 TV版
年代: 2014
偶像 言情 爱情

相关明星



Angelababy

[电影] 微爱之渐入佳境
[电影] 奔跑吧! 兄弟
[综艺] 奔跑吧兄弟春节特辑



李晨

[综艺] 奔跑吧兄弟春节特辑
[综艺] 超级战队
[电视剧] 草根警察



郑恺

[电影] 有种你爱我
[综艺] 奔跑吧兄弟春节特辑
[电影] 江湖论剑实录



王宝强

[综艺] 奔跑吧兄弟春节特辑
[电影] 道士下山
[电影] 非法操作



张子萱

[电视剧] 长大 未删减版
[电影] 等风来
[电视剧] 我的儿子是奇葩

总之一句话: 在当今的大数据时代, 做任何价值分析的前提是数据, 而爬虫则是获得这个前提的一个低成本高收益手段; 而对同学们而言, 另一个重要的价值是解决就业问题。

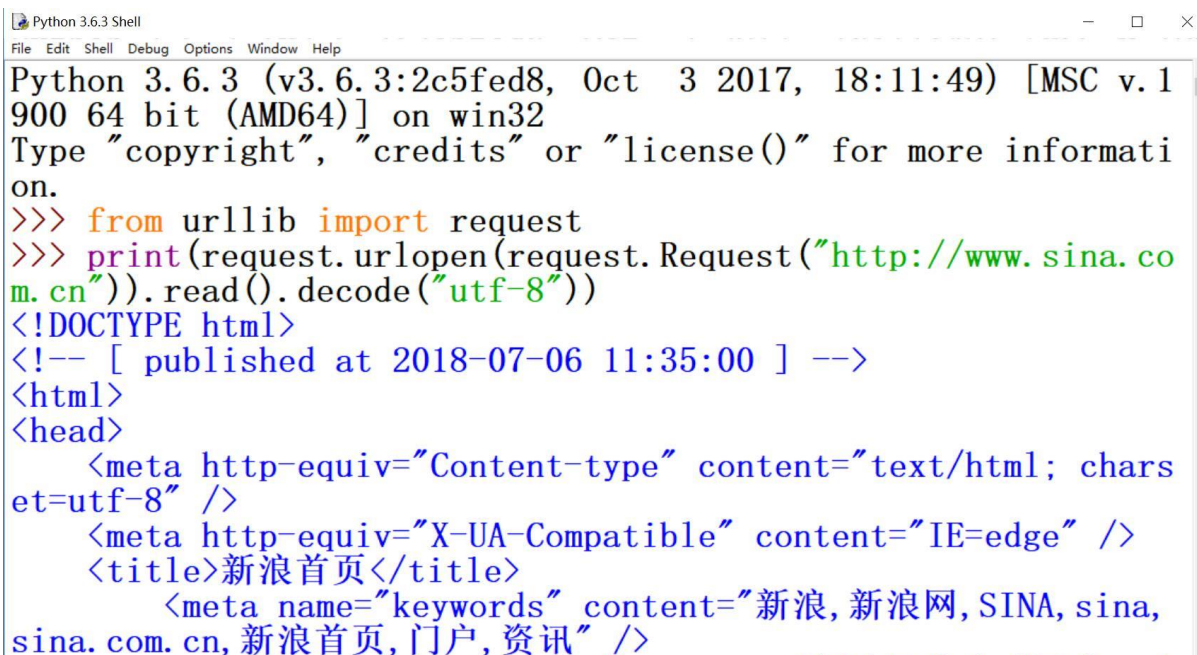
Python 爬虫工程师的招聘情况见:

https://www.lagou.com/jobs/list_python%20%E7%88%AC%E8%99%AB%E5%B7%A5%E7%A8%8B%E5%B8%88city=%E5%85%A8%E5%9B%BD&cl=false&fromSearch=true&labelWords=&suginput

三

1.2.2 怎么做爬虫

用 Python 做爬虫非常的简单, 在交互式环境中简单的两行代码即可



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more informati
on.
>>> from urllib import request
>>> print(request.urlopen(request.Request("http://www.sina.co
m.cn")).read().decode("utf-8"))
<!DOCTYPE html>
<!-- [ published at 2018-07-06 11:35:00 ] -->
<html>
<head>
    <meta http-equiv="Content-type" content="text/html; chars
et=utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <title>新浪首页</title>
    <meta name="keywords" content="新浪, 新浪网, SINA, sina,
sina.com.cn, 新浪首页, 门户, 资讯" />
```

做一个爬虫如此简单吗？

当然不是。让我们来看下要做一个爬虫工程师需要哪些知识和技能：

***爬虫工程师的晋级之路，网络爬虫涉及哪些技术：

初级爬虫工程师：

1. Web 前端的知识：HTML, CSS, JavaScript, DOM, DHTML, Ajax, jQuery, json 等；
2. 正则表达式，能提取正常一般网页中想要的信息，比如某些特殊的文字，链接信息，知道什么是懒惰，什么是贪婪型的正则；
3. 会使用 re, BeautifulSoup, XPath 等获取一些 DOM 结构中的节点信息；
4. 知道什么是深度优先，广度优先的抓取算法，及实践中的使用规则；
5. 能分析简单网站的结构，会使用 urllib 或 requests 库进行简单的数据抓取；

中级爬虫工程师：

1. 了解什么是 Hash, 会使用简单的 MD5, SHA1 等算法对数据进行 Hash 以便存储；
2. 熟悉 HTTP, HTTPS 协议的基础知识, 了解 GET, POST 方法, 了解 HTTP 头中的信息，包括返回状态码，编码，user-agent，cookie，session 等；
3. 能设置 User-Agent 进行数据爬取，设置代理等；
4. 知道什么是 Request，什么是 Response，会使用 Fiddler, Wireshark 等工具抓取及分析简单的网络数据包；对于动态爬虫，要学会分析 Ajax 请求，模拟制造 Post 数据包请求，抓取客户端 session 等信息，对于一些简单的网站，能够通过模拟

数据包进行自动登录;

5. 对于比较难搞定的网站,学会使用浏览器+selenium 抓取一些动态网页信息;
6. 并发下载,通过并行下载加速数据抓取;多线程的使用;

高级爬虫工程师:

1. 能使用 Tesseract,各种图像处理技术,CNN,百度 AI 等库进行验证码识别;
2. 能使用数据挖掘的技术,分类算法等避免死链等;
3. 会使用常用的数据库进行数据存储,查询,如 Mongodb, Redis(大数据量的缓存)等;下载缓存,学习如何通过缓存避免重复下载的问题;Bloom Filter 的使用;
4. 能使用机器学习的技术动态调整爬虫的爬取策略,从而避免被禁 IP 封号等;
5. 能使用一些开源框架 Scrapy 等分布式爬虫,能部署掌控分布式爬虫进行大规模的数据抓取;

第二章 爬虫的基本常识

2.1 爬虫的合法性问题

目前还处于不明确的蛮荒阶段，“允许哪些行为”这种基本秩序还处于建设中。至少目前来看，如果抓取的数据为个人所用，则不存在问题；如果数据用于转载，那么抓取数据的类型就很重要了：一般来说，当抓取的数据是实现生活中的真实数据（比如，营业地址，电话清单）时，是允许转载的，但是，如果是原创数据（比如，意见或评论），通常就会受到版权限制，而不能转载。

讨论：百度爬虫抓取数据行为的合法性问题。

****注意：**不管怎么样，作为一个访客，应当约束自己的抓取行为，这就是说要求下载请求的速度需要限定在一个合理值之内，并且还需要设定一个专属的用户代理来标识自己。

2.2 爬虫的准备工作：网站的背景调研

网站的背景调研对聚焦的网络爬虫而言至关重要，正所谓：知己知彼，百战不殆。

2.2.1 robots 协议

Robots 协议（也称为爬虫协议、机器人协议等）的全称是“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。

比如：

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

腾讯网：<http://www.qq.com/robots.txt>

再比如：

<https://www.douban.com/robots.txt>

<http://www.mafengwo.cn/robots.txt>

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。但是搜索引擎蜘蛛的爬行是被输入了一定的规则的，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 Robots 协议。另外一种则是，通过网站的站长主动对搜索引擎提交网站的网址，搜索引擎则会在接下来派出“蜘蛛”，对该网站进行爬取。

2.2.2 网站地图 sitemap

sitemap 是一个网站所有链接的容器。很多网站的连接层次比较深，蜘蛛很难抓取到，网站地图可以方便搜索引擎蜘蛛抓取网站页面，通过抓取网站页面，清晰了解网站的架构，网站地图一般存放在根目录下并命名为 sitemap，为搜索引擎蜘蛛指路，增加网站重要内容页面的收录。网站地图就是根据网站的结构、框架、内容，生成的导航网页文件。大多数人都知道网站地图对于提高用户体验有好处：它们为网站访问者指明方向，并帮助迷失的访问者找到他们想看的页面。

例子：<http://www.mafengwo.cn/sitemapIndex.xml>

网站地图 sitemap 有两种形式：

A. HTML：称为 HTML 版本的网站地图，英文是 sitemap，特质 HTML 版网站地图，这个版本的网站地图就是用户可以在网站上看到的，列出网站上所有主要页面的链接的页面。对小网站来说，甚至可以列出整个网站的所有页面，对于具有规模的网站来说，一个网站地图不可能罗列所有的页面链接，可以采取两种办法，一种办法是网站地图只列出网站最主要的链接，如一级分类，二级分类，第二种办法是将网站地图分成几个文件，主网站地图列出通往次级网站的链接，次级网站地图在列出一部分页面链接。

B. XML：XML 版本的网站地图是由 Google 首先提出的，怎么区分了，上面所说的 HTML 版本的 s 是小写的，而 XML 版本的 S 则是大写的，XML 版本的网站地图是由 XML 标签组成的，文件本身必须是 utf8 编码，网站地图文件实际上就是列出网站需要被收录的页面的 URL，最简单的网站地图可以是一个纯文本文件，文件只要列出页面的 URL，一行列一个 URL，搜索引擎就能抓取并理解文件内容。

可以使用这个网站工具来生成某网站的 sitemap：www.sitemap-xml.org

2.2.3 估算网站的大小

可以使用搜索引擎来做，比如在百度中使用 site:



说明：这里只是通过百度搜索引擎大致来估算网站的大小，受到网站本身对搜索引擎爬虫的限制，及搜索引擎本身爬取数据技术的限制，所以这只是一个经验值，可以作为估算网站体量量级的一个经验值。

2.2.4 识别网站用了何种技术

为了更好的了解网站，抓取该网站的信息，我们可以先了解一下该网站大致所使用的技术架构。

安装 builtwith：

Windows: `pip install bulitwith`

Linux: `sudo pip install builtwith`

使用：在 Python 交互环境下，输入：

```
import builtwith
builtwith.parse("http://www.sina.com.cn")
```

2.2.5 寻找网站的所有者

有时候，我们需要知道网站的所有者是谁，这里在技术上有个简单的方法可以参考。

安装 python-whois：

Windows: `pip install python-whois`

使用：在 Python 交互环境下，输入：

```
import whois
```

```
whois.whois("http://www.sina.com.cn")
```


第三章 简单爬虫的实现

3.1 可能是史上最简单的爬虫 Demo

最简单的爬虫 Demo:

第一个爬虫程序，两行代码写一个爬虫:

```
import urllib #Python3
print(urllib.request.urlopen(urllib.request.Request("http://example.webscraping.com")).read())
```

这两行代码在 Python3.6 下可以正常运行，获取 <http://example.webscraping.com> 这个页面的内容;

备注: 如果是 Python3 ,则使用如下两行代码:

```
import requests #Python3
print(requests.get('http://example.webscraping.com').text)
```

如果没有 requests 库，则需要使用命令 `pip install requests` 安装一下;

说明: 本讲义目前大部分代码以 Python3.6 的代码位蓝本，讲义的附录 A 中会将 Python2 和 Python3 在爬虫这块最主要几个库的对照表收录进来，按照这张表就可以方便的实现 Python2 与 Python3 在爬虫这块代码的移植。

3.2回顾一下 HTTP, HTTPS 协议

1. 关于 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)

host: 服务器的 IP 地址或者域名

port#: 服务器的端口 (如果是走协议默认端口, 缺省端口 80)

path: 访问资源的路径

query-string: 参数, 发送给 http 服务器的数据

anchor: 锚 (跳转到网页的指定锚点位置)

例如:

`http://www.baidu.com`

<http://item.jd.com/11963485.html#product-detail>

ftp://192.168.1.118:8081/index

URL 是爬虫的入口，非常的重要。

2. HTTP 协议，HTTPS 协议：

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。HTTP 协议是一个应用层的协议，无连接（每次连接只处理一个请求），无状态（每次连接，传输都是独立的）

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

3. HTTP Request 请求常用的两种方法：

Get：是为了从服务器上获取信息，传输给服务器的数据的过程不够安全，数据大小有限制；

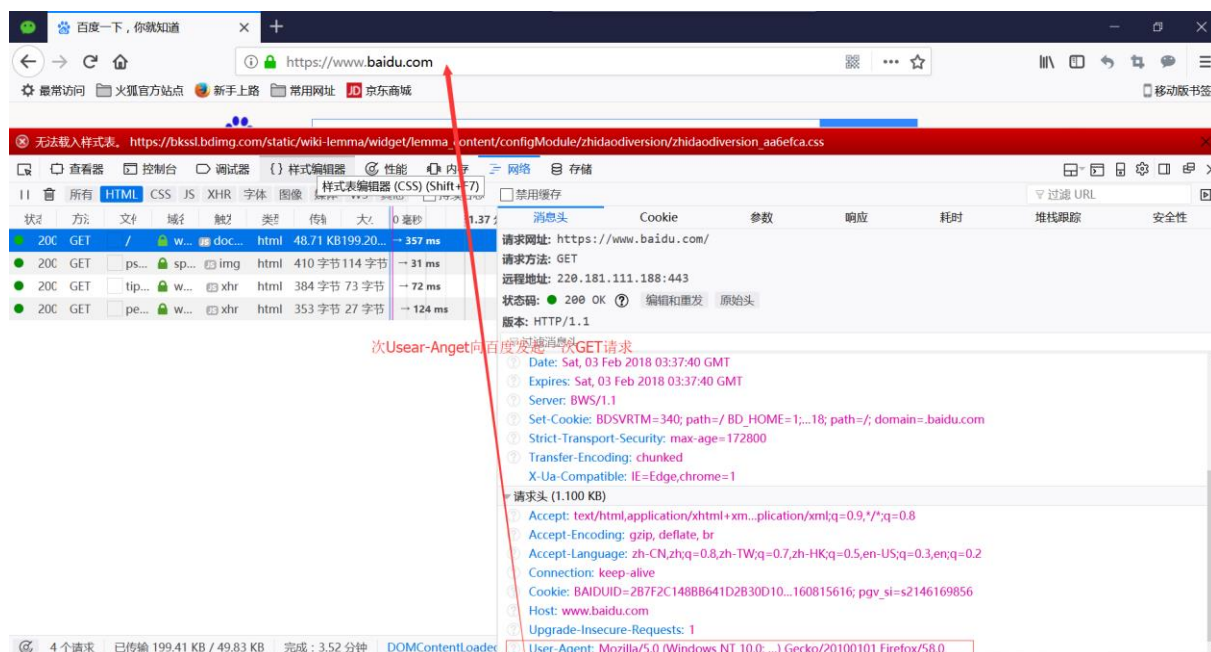
Post：向服务器传递数据，传输数据的过程是安全的，大小理论上没有限制；

HTTP 请求方法

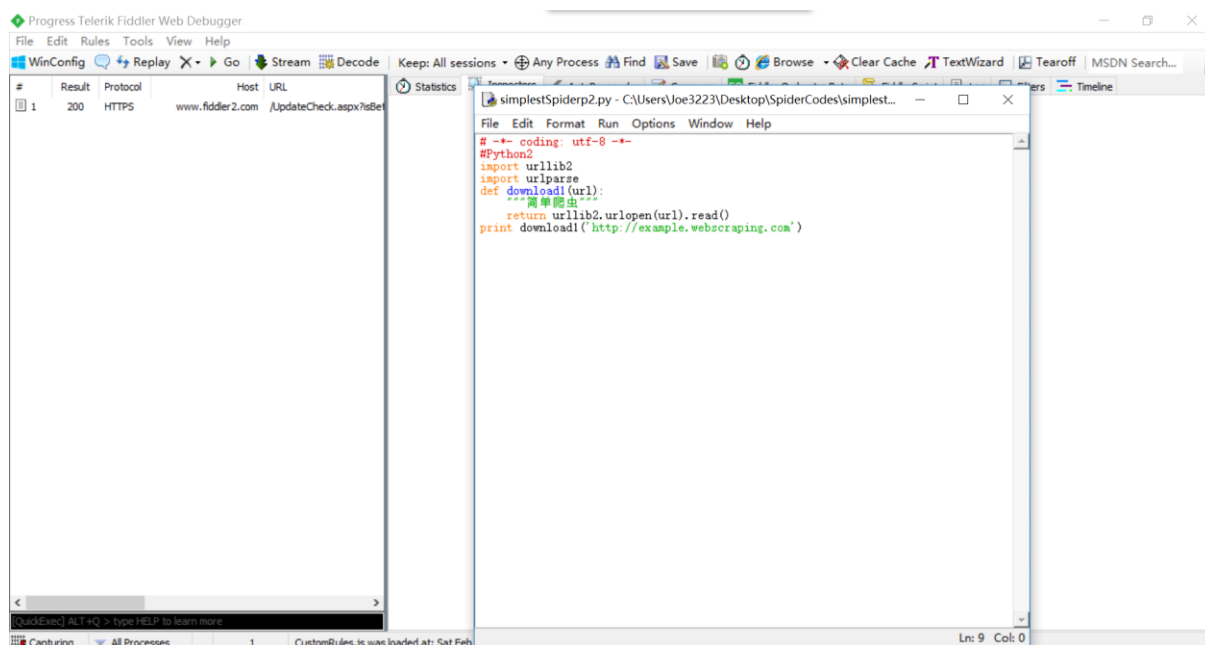
HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	No	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	Yes

4. 关于******User-Agent**

User Agent 中文名为用户代理，简称 UA，它是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等。



我们来看下我们最简单的爬虫跑起来时告诉服务器的 User-Agent 是什么？



通过这个例子，我们发现 Python 爬虫有个默认的带有版本号的 User-Agent，由此很容易能识别出来这是一个 Python 写的爬虫程序。所以如果用默认的用户 Agent，那些反爬虫的程序一眼就能识别出来我们是个 Python 爬虫，这对

Python 爬虫是不利的。

那么，我们如何修改这个 User-Agent，来伪装我们的爬虫程序呢？

Http 协议中请求头的信息

```
headers={"User-Agent":"Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36"}
req = request.Request("http://www.sina.com.cn",
                        headers=headers)
```

返回 http.client.HTTPResponse

response = request.urlopen(req)

代码实例见：[simplestSpider.py](#)

5. HTTP Response 响应的状态码：

200 为成功，300 是跳转；

400，500 意味着有错误：

错误处理

- 400 Bad Request 检查请求的参数或者路径
- 401 Unauthorized 如果需要授权的网页，尝试重新登录
- 403 Forbidden
 - 如果是需要登录的网站，尝试重新登录
 - IP被封，暂停爬取，并增加爬虫的等待时间，如果拨号网络，尝试重新联网更改IP
- 404 Not Found 直接丢弃
- 5XX 服务器错误，直接丢弃，并计数，如果连续不成功，WARNING 并停止爬取

说明：服务器返回给爬虫的信息可以用来判断我们爬虫当前是否正常运行；

当出现异常错误时：一般来说如果是 500 的错误那么爬虫会进入休眠状态，说明服务器已经宕机；如果是 400 的错误，则需要考虑爬虫的抓取策略的修改，可能是网站更新了，或者是爬虫被禁了。如果在一个分布式的爬虫系统中，更容易发现和调整爬虫的策略。

6. HTTP 响应体是我们爬虫需要关心的协议部分的内容：

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>

```

通过 Python 的交互是环境，我们可以直观的方便的看到请求响应的信息，这也看出了 Python 瑞士军刀般的作用。

```

>>> import requests #Python3
>>> html = requests.get('http://example.webscraping.com')
>>> print(html.status_code)
200
>>> print(html.elapsed)
0:00:00.818880
>>> print(html.encoding)
utf-8
>>> print(html.headers)
{'Server': 'nginx', 'Date': 'Thu, 01 Feb 2018 09:23:30 GMT', 'Content-Type': 'text/html; charset=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'Vary': 'Accept-Encoding', 'X-Powered-By': 'web2py', 'Set-Cookie': 'session_id_places=True; httponly; Path=/, session_data_places="6853de2931bf0e3a629e019a5c352fca:1EkG3FIJ7obeqV0rcDDmjBm3y4P4ykVgQojt-qrS33TLNlpfFzO2OuXnY4nyl5sDvdq7p78_wiPyNNUPSdT2ApePNAQdS4pr-gvGc0VvnXo3TazWF8EPT7DXoXlIgHLJbcXoHpflGTwrWJaHq1WuUk4yjHzYtpOhAbnrdBF9_Hw0OFm6-aDK_J25J_asQ0f7"; Path=/', 'Expires': 'Thu, 01 Feb 2018 09:23:30 GMT', 'Pragma': 'no-cache', 'Cache-Control': 'no-store, no-cache, must-revalidate, post-check=0, pre-check=0', 'Content-Encoding': 'gzip'}
>>> print(html.content)

```

略，内容太多了

思考：在复习完 HTTP 协议之后，我们再来看看我们的爬虫程序应该怎么写；

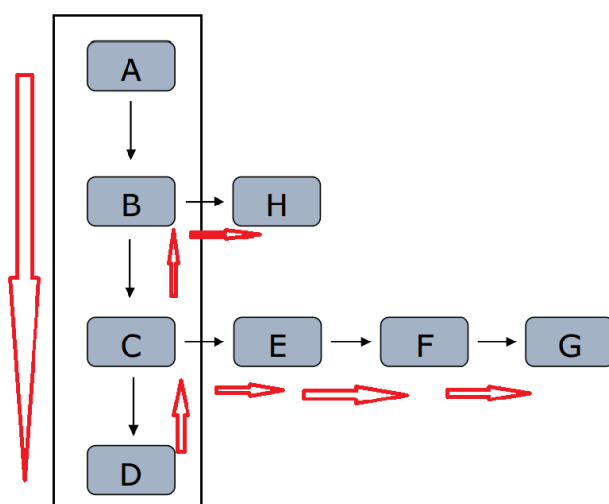
3.3 关于爬虫抓取策略

一般在抓取爬虫数据时，我们不会只抓取一个入口的 URL 数据就停止了。当有多个 URL 链接需要抓取时，我们怎么办？

3.3.1 深度优先算法

深度优先是指搜索引擎先从网站页面上的某个链接进行抓取，进入到这个链接的页面之后，抓取页面上的内容，然后继续顺着当前页面上的这个链接进行抓取下去，直到顺着这个页面上的链接全部抓取完，最深的页面上没有链接了，爬虫再回过头来顺着第一个网站页面上的另外一个链接进行抓取；如下图所示。

深度优先策略

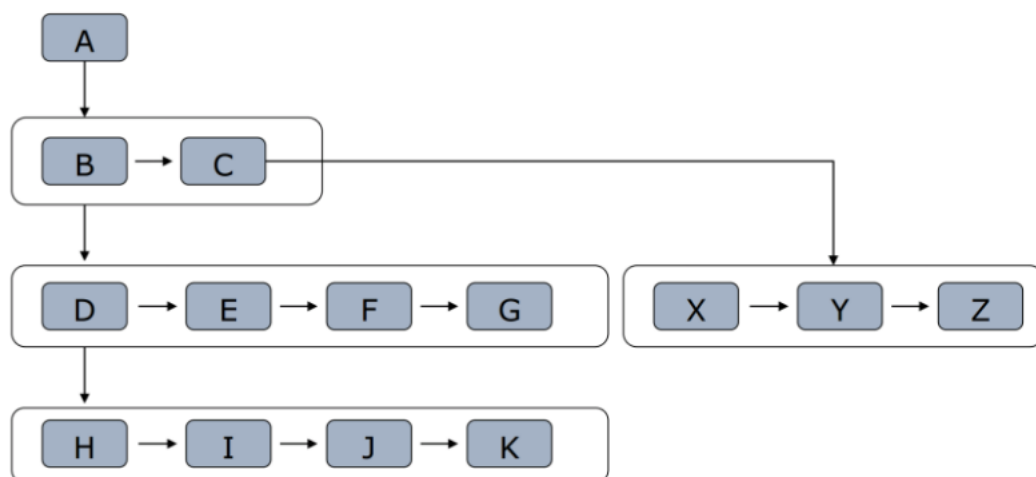


3.3.2 广度/宽度优先算法

广度优先则是另一个过程，它先把该层次的都遍历完，再继续往下走。

如下图所示：

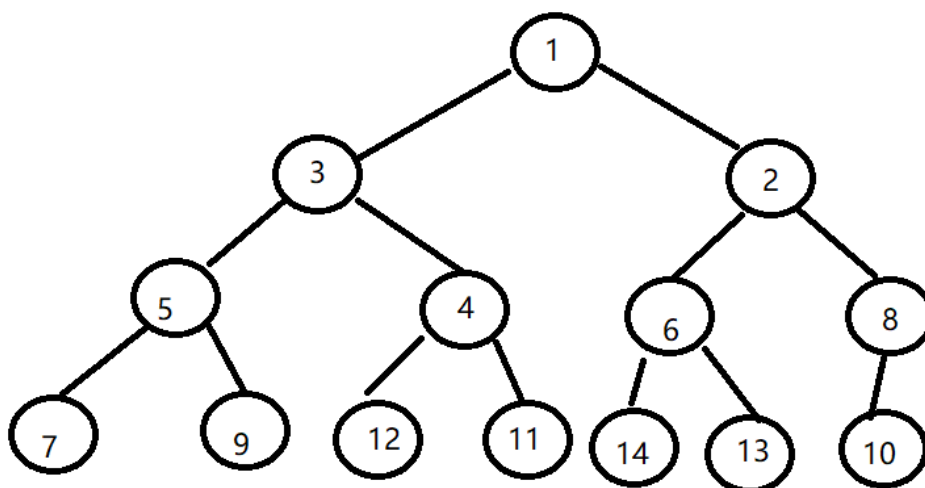
宽度优先策略



练习：构造一个完全二叉树，实现其深度优先和广度优先遍历算法。

一棵二叉树至多只有最下面的一层上的结点的度数可以小 2，并且最下层上的结点都集中在该层最左边的若干位置上，而在最后一层上，右边的若干结点缺失的二叉树，则此二叉树成为完全二叉树。

完全二叉树如下：



深度优先遍历的结果：[1, 3, 5, 7, 9, 4, 12, 11, 2, 6, 14, 13, 8, 10]

广度优先遍历的结果：[1, 3, 2, 5, 4, 6, 8, 7, 9, 12, 11, 14, 13, 10]

代码的演示: binaryTreeTra.py

3.4.3 实践中怎么来组合抓取策略

- 1.一般来说, 重要的网页距离入口站点的距离很近;
- 2.宽度优先有利于多爬虫并行进行合作;
- 3.可以考虑将深度与广度相结合的方式来实现抓取的策略: 优先考虑广度优先, 对深度进行限制最大深度;

总结: 一个通用爬虫的流程如下:

完成一个通用爬虫

- 设置种子站点、宽度及深度
- 一个已下载的队列来记录所有已经完成下载的url
- 实现一个函数, 取得当前url的内容以及所有的外链
- 递归调用这个函数, 来遍历网站
- 错误处理及日志记录

第四章 提取网页中的信息

4.1 数据的类型

网页中数据的类型简单来说可以分成以下三类：

4.1.1 结构化数据

可以用**统一的结构**加以表示的数据。可以使用关系型数据库表示和存储，表现为二维形式的数据。一般特点是：数据以行为单位，一行数据表示一个实体的信息，每一行数据的属性是相同的。

比如 MySQL 数据库表中的数据：

id	name	age	gender
aid1	马化腾	46	male
aid2	马云	53	male
aid3	李彦宏	49	male

4.1.2 半结构化数据

是结构化数据的一种形式，并不符合关系型数据库或其他数据表的形式关联起来的数据模型结构，但包含相关标记，用来分隔语义元素以及对记录和字段进行分层。因此，它也被称为**自描述的结构**。常见的半结构数据有 HTML，XML 和 JSON 等，实际上是以树或者图的结构来存储的。

比如，一个简单的 XML 表示：

```
<person>
<name>A</name>
<age>13</age>
<class>aid1710</class>
<gender>female</gender>
</person>
```

或者

```
<person>
<name>B</name>
<gender>male</gender>
</person>
```

结点中属性的顺序是不重要的，不同的半结构化数据的属性的个数是不一定一样的。这样的数据格式，可以自由地表达很多有用的信息，包括自描述信息（元数据）。所以，半结构化数据的**扩展性很好**，特别适合于在互联网中大规模传播。

4.1.3 非结构化数据

就是没有固定结构的数据。各种文档、图片、视频/音频等都属于非结构化数据。

对于这类数据，我们一般直接整体进行存储，而且一般存储为二进制的数据格式；除了结构化和半结构数据之外的数据都是非结构化数据。

4.2 关于 XML,HTML,DOM 和 JSON 文件

4.2.1 XML, HTML, DOM

XML 即 Extensible Markup Language(**可扩展标记语言**)，是用来定义其它语言的一种元语言，其前身是 SGML(标准通用标记语言)。它没有标签集(tagset)，也没有语法规则(grammar rule)，但是它有句法规则(syntax rule)。任何 XML 文档对任何类型的应用以及正确的解析都必须是**良构的(well-formed)**，即每一个打开的标签都必须有匹配的结束标签，不得含有次序颠倒的标签，并且在语句构成上应符合技术规范的要求。XML 文档可以是有效的(valid)，但并非一定要求有效。所谓有效文档是指其符合其文档类型定义(DTD)的文档。如果一个文档符合一个模式(schema)的规定，那么这个文档是模式有效的(schema valid)。

HTML(Hyper Text Mark-up Language)即超文本标记语言，是 WWW 的描述语言。

HTML 与 XML 的区别与联系：

XML 和 HTML 都是用于操作数据或数据结构，在结构上大致是相同的，但它们在本质上却存在着明显的区别。综合网上的各种资料总结如下。

(一) 语法要求不同：

1. 在 HTML 中不区分大小写，在 XML 中严格区分。

2. 在 HTML 中, 有时不严格, 如果上下文清楚地显示出段落或者列表键在何处结尾, 那么你可以省略</p>或者之类的结束标记。在 XML 中, 是严格的树状结构, 绝对不能省略掉结束标记。
3. 在 XML 中, 拥有单个标记而没有匹配的结束标记的元素必须用一个/ 字符作为结尾。这样分析器就知道不用查找结束标记了。
4. 在 XML 中, 属性值必须分装在引号中。在 HTML 中, 引号是可用可不用的。
5. 在 HTML 中, 可以拥有不带值的属性名。在 XML 中, 所有的属性都必须带有相应的值。
6. 在 XML 文档中, 空白部分不会被解析器自动删除; 但是 html 是过滤掉空格的。

XML 的语法要求比 HTML 严格。

(二) 标记不同：

1. HTML 使用固有的标记; 而 XML 没有固有的标记。
2. HTML 标签是预定义的; XML 标签是免费的、自定义的、可扩展的。

(三) 作用不同：

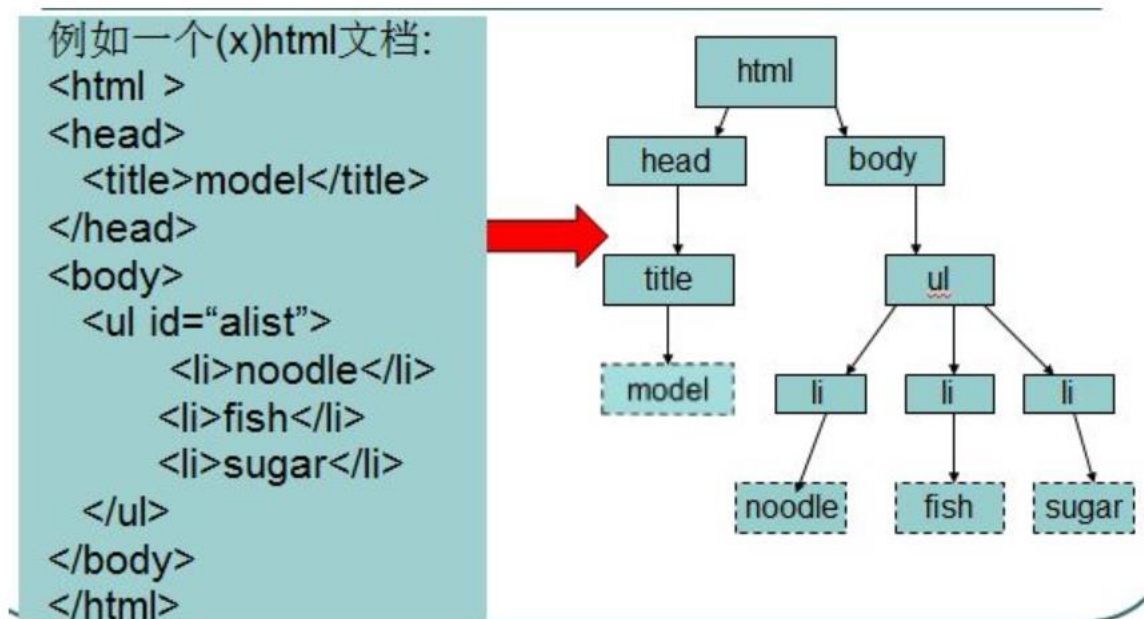
1. HTML 是用来显示数据的; XML 是用来描述数据、存放数据的, 所以可以作为持久化的介质。HTML 将数据和显示结合在一起, 在页面中把这数据显示出来; xml 则将数据和显示分开。XML 被设计用来描述数据, 其焦点是数据的内容。HTML 被设计用来显示数据, 其焦点是数据的外观。
2. XML 不是 HTML 的替代品, XML 和 HTML 是两种不同用途的语言。XML 不是要替换 HTML ; 实际上 XML 可以视作对 HTML 的补充。XML 和 HTML 的目标不同 **HTML 的设计目标是显示数据并集中于数据外观, 而 XML 的设计目标是描述数据并集中于数据的内容。**
3. 没有任何行为的 XML, 与 HTML 相似, XML 不进行任何操作 (共同点) 。

4. 对于 XML 最好的形容可能是: **XML 是一种跨平台的, 与软、硬件无关的, 处理与传输信息的工具。**
5. XML 未来将会无所不在, XML 将成为最普遍的数据处理和数据传输的工具。

关于 DOM :

文档对象模型 (Document Object Model, 简称 DOM), 是 W3C 组织推荐的处理可扩展标志语言的标准编程接口。在网页上, 组织页面 (或文档) 的对象被组织在一个树形结构中, 用来表示文档中对象的标准模型就称为 DOM。Document Object Model 的历史可以追溯至 1990 年代后期微软与 Netscape 的 “浏览器大战”, 双方为了在 JavaScript 与 JScript 一决生死, 于是大规模的赋予浏览器强大的功能。微软在网页技术上加入了不少专属事物, 既有 VBScript、ActiveX、以及微软自家的 DHTML 格式等, 使不少网页使用非微软平台及浏览器无法正常显示。DOM 即是当时蕴酿出来的杰作。

DOM= Document Object Model, 文档对象模型, DOM 可以以一种独立于平台和语言的方式访问和修改一个文档的内容和结构。换句话说, 这是表示和处理一个 HTML 或 XML 文档的常用方法。DOM 很重要, DOM 的设计是以对象管理组织 (OMG) 的规约为基础的, 因此可以用于任何编程语言。最初人们把它认为是一种让 JavaScript 在浏览器间可移植的方法, 不过 DOM 的应用已经远远超出这个范围。DOM 技术使得用户页面可以动态地变化, 如可以动态地显示或隐藏一个元素, 改变它们的属性, 增加一个元素等, DOM 技术使得页面的交互性大大地增强。DOM 实际上是以面向对象方式描述的文档模型。DOM 定义了表示和修改文档所需的对象、这些对象的行为和属性以及这些对象之间的关系。可以把 DOM 认为是页面上数据和结构的一个树形表示, 不过页面当然可能并不是以这种树的方式具体实现。



通过 JavaScript, 您可以重构整个 HTML 文档。您可以添加、移除、改变或重排页面上的项目。要改变页面的某个东西, JavaScript 就需要获得对 HTML 文档中所有元素进行访问的入口。这个入口, 连同对 HTML 元素进行添加、移动、改变或移除的方法和属性, 都是通过文档对象模型来获得的 (DOM)。

4.2.2 JSON 文件

JSON(**JavaScript Object Notation**, JS 对象标记) 是一种**轻量级**的数据交换格式。它基于 ECMAScript (w3c 制定的 JS 规范) 的一个子集, 采用完全独立于编程语言的文本格式来**存储和表示数据**。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。易于人阅读和编写, 同时也易于机器解析和生成, 并有效地提升网络传输效率。

JSON 语法规则:

在 JS 语言中, 一切都是对象。因此, 任何支持的类型都可以通过 JSON 来表示, 例如字符串、数字、对象、数组等。

但是对象和数组是比较特殊且常用的两种类型:

1. 对象表示为**键值对**
2. 数据由逗号分隔
3. 花括号保存对象
4. 方括号保存数组

JSON 键值对是用来保存 JS 对象的一种方式, 和 JS 对象的写法也大同小异,

键/值对组合中的键名写在前面并用双引号 "" 包裹，使用冒号 : 分隔，然后紧接着值

```
{"firstName": "Json", "class": "aid1710"}
```

这很容易理解，等价于这条 JavaScript 语句：

```
{firstName : "Json", "class": "aid1710"}
```

JSON 与 JS 对象的关系：

很多人搞不清楚 JSON 和 JS 对象的关系，甚至连谁是谁都不清楚。其实，可以这么理解：**JSON 是 JS 对象的字符串表示法，它使用文本表示一个 JS 对象的信息，本质是一个字符串。**

如 `var obj = {a: 'Hello', b: 'World'};` //这是一个对象，注意键名也是可以使用引号包裹的

`var json = '{"a": "Hello", "b": "World"}';` //这是一个 JSON 字符串，本质是一个字符串。

Python 中关于 JSON 的操作简单演示：

代码示例见 `jsonTest.py`

JSON 和 XML 的比较：

1.可读性：

JSON 和 XML 的可读性可谓不相上下，一边是简易的语法，一边是规范的标签形式，很难分出胜负。

2.可扩展性：

XML 天生有很好的扩展性，JSON 当然也有，没有什么是 XML 可以扩展而 JSON 却不能扩展的。不过 JSON 在 Javascript 主场作战，可以存储 Javascript 复合对象，有着 xml 不可比拟的优势。

3.编码难度：

XML 有丰富的编码工具，比如 Dom4j、JDom 等，JSON 也有提供的工具。无工具的情况下，相信熟练的开发人员一样能很快的写出想要的 xml 文档和 JSON 字符串，不过，xml 文档要多很多结构上的字符。

4.解码难度

XML 的解析方式有两种：

一是通过文档模型解析，也就是通过父标签索引出一组标记。例如：`xmlData.getElementsByTagName("tagName")`，但是这样是要在预先知道文档结构的情况下使用，无法进行通用的封装。

另外一种方法是遍历节点（`document` 以及 `childNodes`）。这个可以通过递归来实现，不过解析出来的数据仍旧是形式各异，往往也不能满足预先的要求。凡是这样可扩展的结构数据解析起来一定都很困难。JSON 也同样如此。**如果预先知道 JSON 结构的情况下，使用 JSON 进行数据传递简直是太美妙了，可以写出很实用美观可读性强的代码。**

如果你是纯粹的前台开发人员，一定会非常喜欢 JSON。但是如果你是一个应用开发人员，就不是那么喜欢了，毕竟 xml 才是真正的结构化标记语言，用于进行数据传递。而不知道 JSON 的结构而去解析 JSON 的话，那简直是噩梦。费时费力不说，代码也会变得冗余拖沓，得到的结果也不尽人意。

但是这样也不影响众多前台开发人员选择 JSON。因为 `json.js` 中的 `toJSONString()` 就可以看到 JSON 的字符串结构。当然不是使用这个字符串，这样仍旧是噩梦。常用 JSON 的人看到这个字符串之后，就对 JSON 的结构很明了了，就更容易的操作 JSON。以上是在 Javascript 中仅对于数据传递的 xml 与 JSON 的解析。

在 Javascript 地盘内，JSON 毕竟是主场作战，其优势当然要远远优越于 xml。

如果 JSON 中存储 Javascript 复合对象，而且不知道其结构的话，相信很多程序员也一样是哭着解析 JSON 的。除了上述之外，JSON 和 XML 还有另外一个很大的区别在于有效数据率。JSON 作为数据包格式传输的时候具有更高的效率，这是因为 JSON 不像 XML 那样需要有严格的闭合标签，这就让有效数据量与总数据包比大大提升，从而减少同等数据流量的情况下，网络的传输压力。

实例比较：

XML 和 JSON 都使用结构化方法来标记数据，下面来做一个简单的比较。

用 XML 表示中国部分省市数据如下：

```
<?xml version="1.0" encoding="utf-8"?>
<country>
  <name>中国</name>
  <province>
    <name>黑龙江</name>
    <cities>
      <city>哈尔滨</city>
```



```
        <city>大庆</city>
      </cities>
    </province>
    <province>
      <name>广东</name>
      <cities>
        <city>广州</city>
        <city>深圳</city>
        <city>珠海</city>
      </cities>
    </province>
    <province>
      <name>台湾</name>
      <cities>
        <city>台北</city>
        <city>高雄</city>
      </cities>
    </province>
    <province>
      <name>新疆</name>
      <cities>
        <city>乌鲁木齐</city>
      </cities>
    </province>
  </country>
```

用 JSON 表示如下:

```
{
  "name": "中国",
  "province": [{
    "name": "黑龙江",
    "cities": {
      "city": ["哈尔滨", "大庆"]
```

```
    }  
  }, {  
    "name": "广东",  
    "cities": {  
      "city": ["广州", "深圳", "珠海"]  
    }  
  }, {  
    "name": "台湾",  
    "cities": {  
      "city": ["台北", "高雄"]  
    }  
  }, {  
    "name": "新疆",  
    "cities": {  
      "city": ["乌鲁木齐"]  
    }  
  }  
}]  
}
```

可以看到：**JSON** 简单的语法格式和清晰的层次结构明显要比 **XML** 容易阅读，并且在数据交换方面，由于 **JSON** 所使用的字符要比 **XML** 少得多，可以大大得节约传输数据所占用得带宽。

4.3 怎么提取网页中的信息

4.3.1 XPath 与 lxml

XPath 是一门在 XML 文档中**查找信息的语言**，对 XPath 的理解是很多高级 XML 应用的基础，XPath 在 XML 中通过元素和属性进行导航。

lxml 是一个用来处理 XML 的**第三方 Python 库**，它在底层封装了用 C 语言编写的 libxml2 和 libxslt，并以简单强大的 Python API，兼容并加强了著名的 Element Tree API。

安装：pip install lxml

使用：from lxml import etree

1. XPath 术语：

在 XPath 语境中，XML 文档被视作节点树，节点树的根节点也被称作文档节点。XPath 将节点树中的节点(Node)分为七类：**元素(Element)**，**属性(Attribute)**，**文本(Text)**，**命名空间(Namespace)**，**处理指令(Processing-instruction)**，**注释(Comment)**和**文档节点(Document nodes)**。

看一下 XML 文档例子：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book>

    <title lang="en">Harry Potter</title>

    <author>J K. Rowling</author>

    <year>2005</year>

    <price>29.99</price>

</book>
</bookstore>
```

以上的 XML 文档中：

<bookstore> （这是一个“根”）

<author>J K. Rowling</author> （这是一个“元素”）

lang="en" （这是一个“属性”）

从另一个视角来看它：

bookstore	(根)
book	(元素)
title	(元素)
lang = en	(属性)
text = Harry Potter	(文本)
author	(元素)
text = J K. Rowling	(文本)
year	(元素)
text = 2005	(文本)
price	(元素)
text = 29.99	(文本)

2. 节点之间的关系

父 (Parent)：每个元素都肯定有一个父节点，最顶层的元素父亲是根节点。同理每个属性必然有一个父，它们的父是元素。上例 XML 文档中，根 bookstore 是元素 book 的父节点，book 是元素 title, author, year, price 的父节点，title 是 lang 的父节点。

子 (Children)：元素可以有零或多个子。上例 XML 文档中，title, author, year, price 是 book 的子节点。

同胞 (Sibling)：父节点相同的节点之间互为同胞，也称彼此的兄弟节点。上例 XM 文档中，title, author, year, price 彼此互为同胞。

先辈 (Ancestor)：某节点的父节点、父的父，以此类推一直追溯至根节点之间所有节点。上例 XM 文档中，title, author, year, price 的先辈就是 book, bookstore。

后代 (Descendant)：某节点的子节点、子的子，以此类推至最后一个子节点之间所有节点。上例 XM 文档中，bookstore 的后代就是 title, author, year, price 。

3. 选取节点

以下为基本路径的表达方式，记住 XPath 的路径表达式都是基于某个节点之上的，例如最初的当前节点一般是根节点，这与 Linux 下路径切换原理是一样的。表达式描述：

<code>nodename</code>	选取已匹配节点下名为 <code>nodename</code> 的子元素节点
<code>/</code>	如果以 <code>/</code> 开头，表示从根节点作为选取起点。
<code>//</code>	在已匹配节点后代中选取节点，不考虑目标节点的位置。
<code>.</code>	选取当前节点。
<code>..</code>	选取当前节点的父元素节点。
<code>@</code>	选取属性。

4. 通配符

<code>*</code>	匹配任何元素。
<code>@*</code>	匹配任何属性。
<code>node()</code>	匹配任何类型的节点。

5. 预判（Predicates）或 条件选取

预判是用来查找某个特定的节点或者符合某种条件的节点，预判表达式位于方括号中。使用“`|`”运算符，你可以选取符合“或”条件的若干路径。

具体例子见下面代码 `lxmlTest.py`。

6. 坐标轴

XPath 坐标轴：坐标轴用于定义当对当前节点的节点集合。

坐标轴名称	含义
<code>ancestor</code>	选取当前节点的所有先辈元素及根节点。
<code>ancestor-or-self</code>	选取当前节点的所有先辈以及当前节点本身。
<code>attribute</code>	选取当前节点的所有属性。
<code>child</code>	选取当前节点的所有子元素。
<code>descendant</code>	选取当前节点的所有后代元素。
<code>descendant-or-self</code>	选取当前节点的所有后代元素以及当前节点本身。
<code>following</code>	选取文档中当前节点的结束标签之后的所有节点。
<code>following-sibling</code>	选取当前节点之后的所有同级节点。
<code>namespace</code>	选取当前节点的所有命名空间节点。
<code>parent</code>	选取当前节点的父节点。
<code>preceding</code>	选取当前节点的开始标签之前的所有节点。
<code>preceding-sibling</code>	选取当前节点之前的所有同级节点。
<code>self</code>	选取当前节点。

7. 位置路径的表达式

位置路径可以是绝对路径，也可以是相对路径。**绝对路径以 “/” 开头**。每条路径包括一个或多个步，每步之间以 “/” 分隔。

绝对路径： /step/step/...

相对路径： step/step/...

每步根据当前节点集合中的节点计算。

步 (step) 包括三部分：

坐标轴 (axis)： 定义所选节点与当前节点之间的关系。

节点测试 (node-test)： 识别某个坐标轴内部的节点。

预判 (predicate)： 提出预判条件对节点集合进行筛选。

步的语法： 坐标轴::节点测试[预判]

****代码例子：** SpiderCodes \lxmlTest.py

4.3.2 BeautifulSoup4

[Beautiful Soup](#) 是用 **Python** 写的一个 HTML/XML 的解析器，它可以很好的处理不规范标记并生成剖析树(parse tree)。 它提供**简单又常用的导航 (navigating)**，搜索以及修改剖析树的操作。它可以大大节省你的**编程时间**。

安装：(sudo) pip install beautifulsoup4

使用：

在程序中导入 ~~Beautiful Soup~~ 库：

```
from BeautifulSoup import BeautifulSoup # For processing HTML
from BeautifulSoup import BeautifulSoup # For processing XML
import BeautifulSoup # To get everything
```

代码例子

```
from bs4 import BeautifulSoup
import re

doc = ['<html><head><title>Page title</title></head>',
      '<body><p id="firstpara" align="center">This is paragraph <b>one</b>.',
      '<p id="secondpara" align="blah">This is paragraph <b>two</b>.',
      '</html>']
```

```
soup = BeautifulSoup(''.join(doc))
print soup.prettify()
```

定位某些 soup 元素很简单，比如上例：

```
soup.contents[0].name
# u'html'
```

```
soup.contents[0].contents[0].name
# u'head'
```

```
head = soup.contents[0].contents[0]
head.parent.name
# u'html'
```

```
head.next
# <title>Page title</title>
```

```
head.nextSibling.name
# u'body'
```

```
head.nextSibling.contents[0]
# <p id="firstpara" align="center">This is paragraph <b>one</b>.</p>
```

```
head.nextSibling.contents[0].nextSibling
# <p id="secondpara" align="blah">This is paragraph <b>two</b>.</p>
```

也可以利用 soup，获得特定标签或有着特定属性的标签，修改 soup 也很简单；

代码例子：见 testBS4.py

进一步阅读，可以参考：

<https://www.crummy.com/software/BeautifulSoup/bs3/documentation.zh.html>

BS4 与 lxml 的比较:

lxml	C 实现, 只会局部遍历, 快;	复杂, 语法不太友好;
BS4	Python 实现, 会加载整个文档, 慢;	简单, API 人性化;

4.3.3 正则表达式 re

被用来检索\替换那些符合某个模式(规则)的文本,对于文本过滤或规则匹配,最强大的就是正则表达式,是 python 爬虫里必不可少的神兵利器。

基本匹配规则:

[0-9] 任意一个数字, 等价\d

[a-z] 任意一个小写字母

[A-Z]任意一个大写字母

[\^0-9] 匹配非数字, 等价\D

\w 等价[a-z0-9_], 字母数字下划线

\W 等价对\w 取非

. 任意字符

[] 匹配内部任意字符或子表达式

[^] 对字符集合取非

* 匹配前面的字符或者子表达式 0 次或多次

+ 匹配前一个字符至少 1 次

? 匹配前一个字符 0 次或多次

^ 匹配字符串开头

\$ 匹配字符串结束

Python 使用正则表达式

Python 的 re 模块

pattern 编译好的正则表达式

几个重要的方法:

match: 匹配一次从开头;

search: 匹配一次, 从某位置;

findall: 匹配所有;

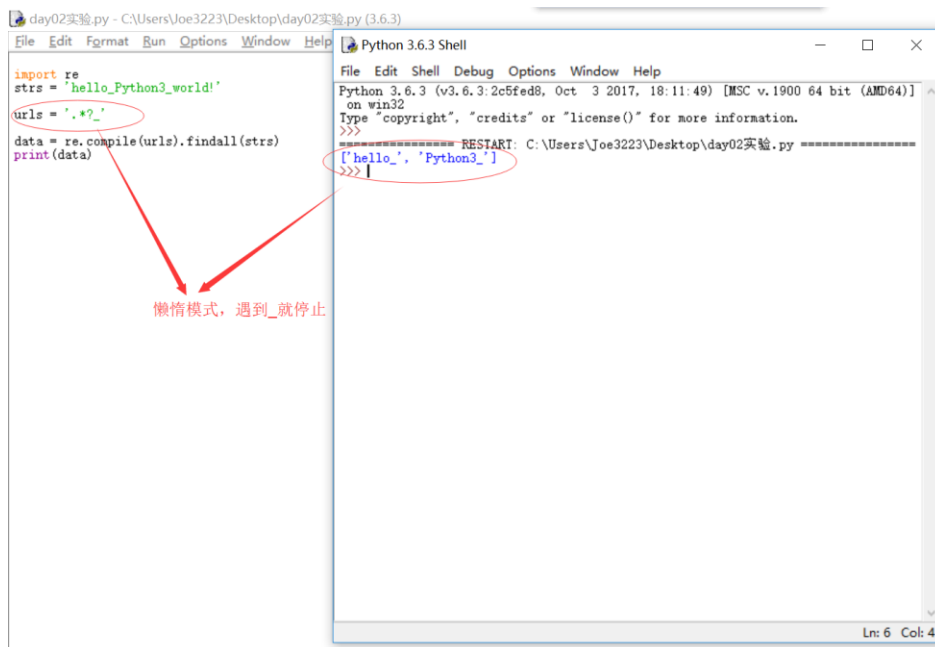
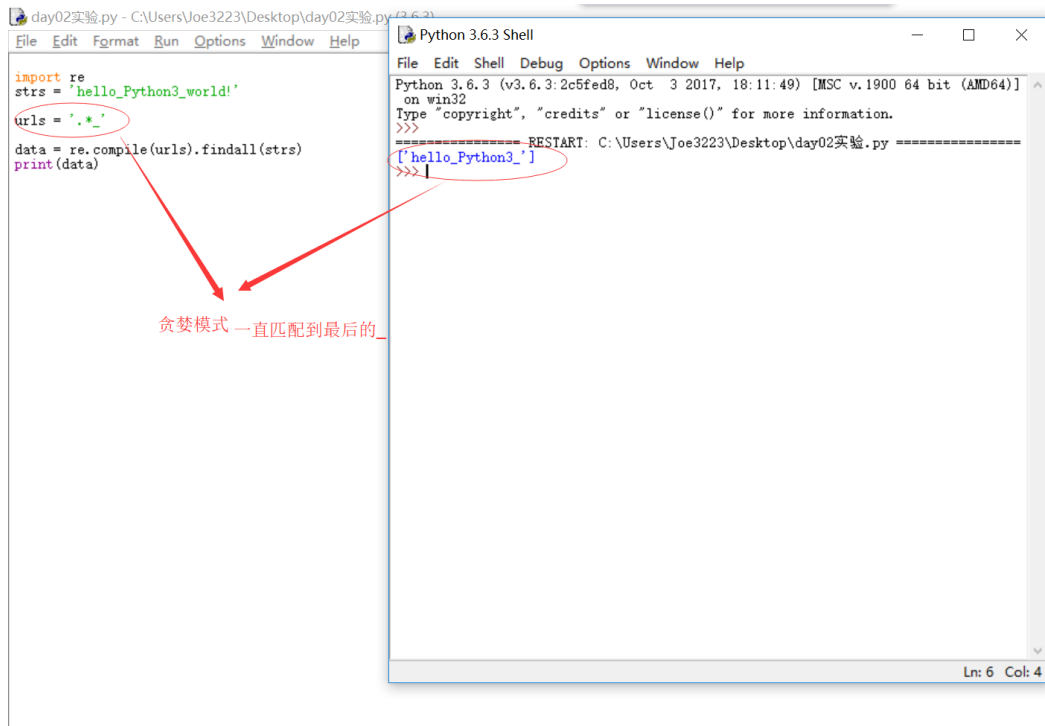
split: 分隔;

sub: 替换;

需要注意的两种模式:

贪婪模式: (.*)

懒惰模式: (.*)?



代码例子: 见 greedyRe.py

练习:

1. 使用一个正则表达式来匹配 url 链接; 如 <http://www.baidu.com>
2. 抓取 <http://example.webscraping.com/sitemap.xml> 下的数据

见：rePracticep2.py

3. 用正则表达式实现下面的效果：

把 i=d%0A&from=AUTO&to=AUTO&smartresult=dict

转换成下面的形式：

i:d%0A

from:AUTO

to:AUTO

smartresult:dict

关于正则表达式，更多的细节，可以读下这篇文章：

<https://www.cnblogs.com/deerchao/archive/2006/08/24/zhengzhe30fengzhongjiaocheng.html>

总结：正则，BS，lxml 的比较

抓取方法	性能	使用难度	安装难度
正则表达式	快	困难	简单（内置模块）
Beautiful Soup	慢	简单	简单（纯 Python）
Lxml	快	简单	相对困难

第五章 动态网页的挑战

很多网站的数据，比如电商网站商品的价格，评论等等会采用动态加载的方式来加载，这样可能在爬虫程序刚刚访问时无法直接获取到相关数据。那么怎么应对这样的问题呢？

5.1 动态网页的使用场景

先看下面一个例子：



Python爬虫开发与项目实战

零基础学习爬虫技术，从Python和Web前端基础开始讲起，由浅入深，包含大量案例，实用性强

范传辉 著

京东价：¥62.40 [7.9折] [定价：¥79.00] (降价通知)

优惠券：满105减6 满200减16

增值业务：礼品包装

排名：自营 计算机与互联网销量榜 第291位

配送至：北京朝阳区三环以内 有货，支持 99元免基础运费 | 货到付款

服务：由 京东 发货，并提供售后服务。23:00前下单，预计明天(02月03日)送达

选择系列：Python爬虫项目实战 零基础学Python爬虫

白条分期：不分期 ￥20.8起×3期 ￥10.56起×6期 ￥5.51起×12期 ￥2.91起×24期

1 + 加入购物车 购买电子书 ¥16.99

企业批量购书

分享 关注商品 举报

温馨提示：1. 支持7天无理由退货

这是京东上看一本书的场景。我们发现打开一本书之后，书的价格，排名等信息及书的评论信息不是在我们第一次打开网站时就立即加载进来的。而是通过二次请求或多次的异步请求获取的。这样的页面就是动态页面。

关于动态页面使用的场景：

希望异步刷新的场景。有些网页内容很多，一次加载完对服务器压力很大，而且有的用户不会去查看所有内容；

5.2 回到与 HTTP 服务器发送请求数据的原始方法

5.2.1 GET 方法

GET 把参数数据队列添加到 URL 中，Key 和 Value 的各个字段一一对应；在 URL 中可以看到。

浏览器的 URL 中有些符号，字符不能被很好的识别。那么我们需要有一套编码的方

式来传递信息。所以发送端需要做 urlencode；接收端需要做 urldecode；

https://www.baidu.com/s?ie=utf-8&f=3&rsv_bp=1&rsv_idx=1&tn=baidu&wd=python%20%E7%88%AC%E8%99%AB&oq=python%2520%25E7%2588%25AC%25E8%2599%25AB&rsv_pq=ef9e51560000eca9&rsv_t=db33UmQaiVre2ourDQMjsxtS03DJVCOUAc2AHonGCIhHT2or8zim%2F96kVGA&rqlang=cn&rsv_enter=0&prefixsug=python%2520%25E7%2588%25AC%25E8%2599%25AB&rsp=0&rsv_sug=2

在线测试工具: <http://tool.chinaz.com/tools/urlencode.aspx>

1. <https://www.baidu.com/s?wd=DNS>

?xxx=yyy&time=zzz get 请求的标识

2. <http://acb.com/login?name=zhangsan&password=123>

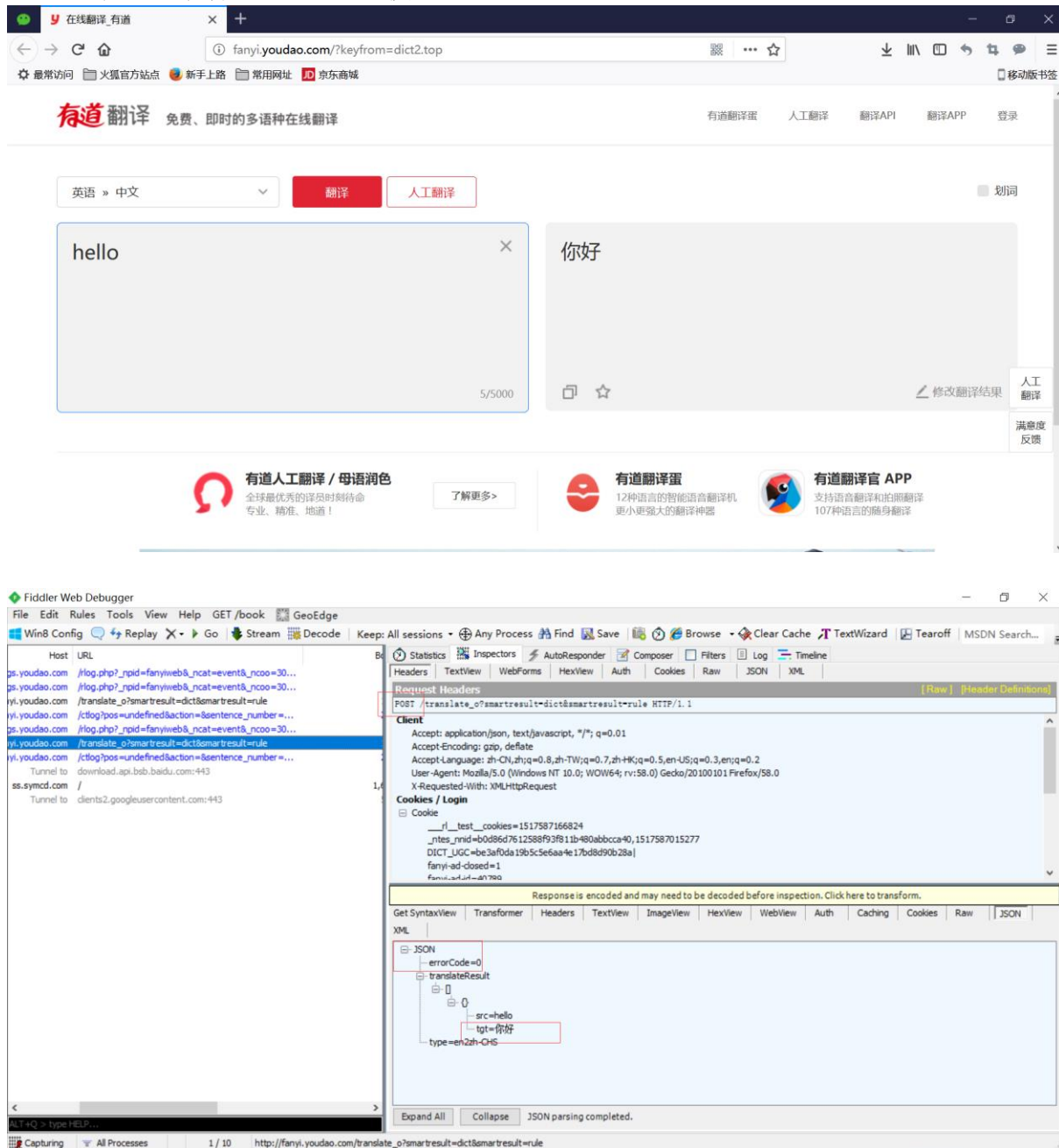
login: name=zhangsan password=123

示例代码: 见 baiduSearch.py



5.2.2 POST 方法

通过一个例子来看 POST 方法的使用：



这是有道翻译的页面，仔细观察会发现，当用户每次输入一个想要翻译的词语时，页面的 URL 信息并不发生任何改变。这是一个典型的异步使用 Ajax 的技术，用 JSON 格式进行数据的传递。

我们如何利用爬虫程序来实现一个自己的翻译器呢？

来看代码示例：见 youdaofanyi.py

5.3 更加难以对付的动态网站

5.3.1 应对需要多次数据的交互模拟的网站

我们有时会遇到像淘宝这样的大型网站，对数据版权看得特别重的，它们的网站有大量的工程师和技术人员去维护，它们也可能在技术手段上采用多次交互数据包的方式来完成网站服务器与用户浏览器之间的交互。如果此时还采用传统的分析数据包的方式会比较的复杂，难度较高。那么，有没有一劳永逸的方法，来解决此类问题呢？

我们的解决方案是：Selenium + PhantomJS。

我们的爬虫其实就是在做模拟浏览器的行为。

5.3.2 Selenium

一个 Web 自动化测试工具，最初是为了网站自动化测试而开发的；我们玩游戏有按键精灵；Selenium 也可以做类似的事情，但是它是在浏览器中做这样的事情。

安装： `sudo pip install selenium`(`pip install selenium`)

在 Python 中 `from selenium import webdriver` 来测试是否装好

说明：想要用 Python 做自动化测试的童鞋们可以好好研究一下 Selenium 的使用。

5.3.3 PhantomJS 及浏览器

说明：我们上课用的时有界面的 Firefox 浏览器，以便于教学；

一个基于 **webkit** 无界面(headless)的**浏览器**，它可以把网站加载到内存中并执行页面上的 JS，但它没有图形用户界面，所以耗费的资源比较少；

安装： `sudo apt install phantomjs` (**此方法可能安装不完整, 导致部分功能无法使用**)

Linux Ubuntu 下完全安装的方法(参看

http://blog.csdn.net/m0_38124502/article/details/79276499

)

Wget

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

`cd 下载`

```
tar -xvf phantomjs-2.1.1-linux-x86_64.tar.bz2
cd phantomjs-2.1.1-linux-x86_64/
cd bin/
sudo cp phantomjs /usr/bin
python -启动-> 浏览器进程 phantomjs,
```

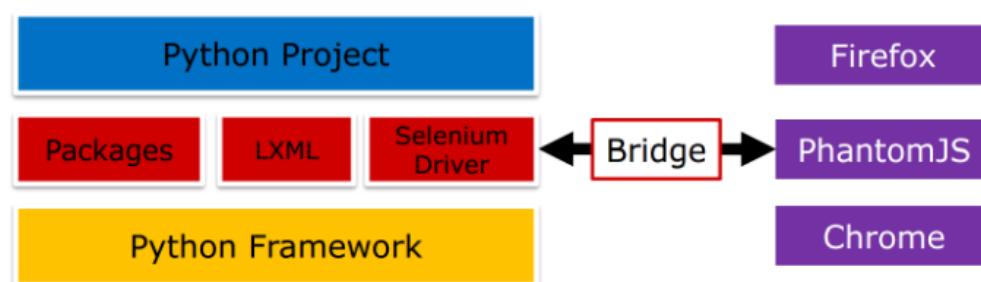
测试:

SpiderCodes\Phantomjs\.. 对其中的例子 helloworld.js, pageload.js 进行测试;

注意: ***有可能造成资源泄漏; 为了避免这种事的发生, 需要有个策略适当的时候去 kill phantomjs 进程。

Close and Clear

Selenium 通过内嵌的浏览器 driver 与浏览器进程通信, 因此在退出的时候必须调用 `driver.close()` 及 `driver.quit()` 来退出 PhantomJS, 否则 PhantomJS 会一直运行在后台并占用系统资源。



5.3.4 Selenium + PhantomJS

Selenium 通过浏览器的驱动，支持大量的HTML及Javascript的操作，常用的可以包括：

- page_source: 获取当前的 html 文本
- title: HTML 的 title
- current_url: 当前网页的URL
- get_cookie() & get_cookies(): 获取当前的cookie
- delete_cookie() & delete_all_cookies(): 删除所有的cookie
- add_cookie(): 添加一段cookie
- set_page_load_timeout(): 设置网页超时
- execute_script(): 同步执行一段javascript命令
- execute_async_script(): 异步执行javascript命令

双剑合璧利用二者同时完成某个任务，

由于讲课过程中浏览器的使用不同，我们采用 FireFox 浏览器，示例代码如下：

seleniumTest.py

5.4 关于动态网站信息抓取的总结

总的来说，我们的爬虫要尽量模拟的看起来就像是真正的用户在浏览器上访问服务器网站的行为。如果我们使用 GET 或 POST 的方式来模拟浏览器与服务器间通信的行为，成本比较低，但是应对复杂的网站或者服务器精心防御的网站来说是很难骗过服务器的。Selenium+PhantomJS 的方案则会让我们的程序看起来更像是普通的用户，但是它的效率相对而言会降低很多，速度也会慢很多。在大规模爬去数据时可能遇到许多新的挑战。（比如网站尺寸的设置，等待时间的设定等）

练习： 抓取猫眼电影 <http://maoyan.com/board/4?offset> 中 TOP100 的电影榜单；

第六章 表单与爬虫登录问题

前面的章节中，我们介绍了如果在客户端与服务器之间进行数据交换。我们可以使用 GET 方法和 POST 方法与服务器进行交互，敏感数据只应使用 POST 请求进行发送，以避免将书暴露在 URL 中。当然，服务器还支持其他 HTTP 方法，比如 PUT 和 DELETE 等方法，但这些方法在表单中都不支持。

6.1 关于表单

客户端的浏览器需要与网站服务器进行交互，服务器需要根据用户输入返回对应的信息。

来看 w3c 的一个例子：

http://www.w3school.com.cn/html/html_forms.asp

关于 GET，POST 与服务器的交互方法，可以见 5.2 节。

下面我们重点来看一个怎么处理登录表单的问题。

6.2 管理 cookie

6.2.1 使用 cookie 登录

HTTP 协议本身是无状态的，怎么保存来过或登陆过网站的信息？

所以我们需要在 HTTP 协议之外通过某种机制来识别用户的身份。于是就有了 Session 和 Cookie。

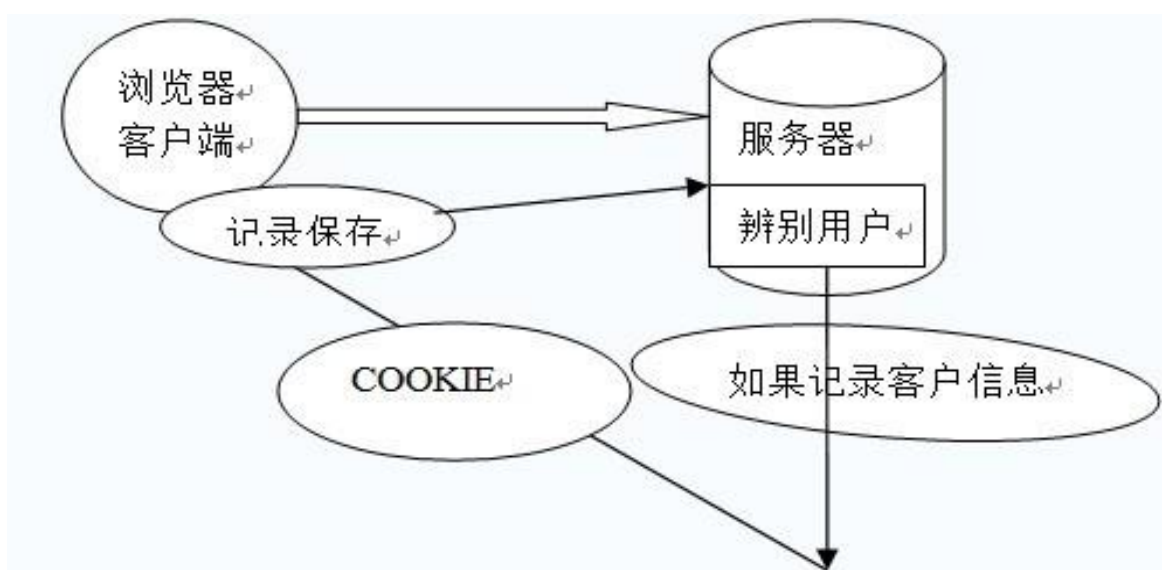
什么是 Cookie，什么是 Session？

会话(Session)跟踪是 Web 程序中常用的技术，用来跟踪用户的整个会话。常用的会话跟踪技术是 Cookie 与 Session。**Cookie 通过在客户端记录信息确定用户身份，Session 通过在服务器端记录信息确定用户身份。**

Cookie 意为“甜饼”，是由 W3C 组织提出，最早由 Netscape 社区发展的一种机制。目前 Cookie 已经成为标准，所有的主流浏览器如 IE、Netscape、Firefox、Opera 等都支持 Cookie。由于 HTTP 是一种无状态的协议，服务器单从网络连接上无从知道客户身份。所以就给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是 Cookie 的工作原理。

Cookie 实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用 response 向客户端浏览器颁发一个 Cookie。客户端浏览器会把 Cookie 保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该 Cookie 一同提交给服务器。服务器检查该 Cookie，

以此来辨认用户状态。服务器还可以根据需要修改 Cookie 的内容。



我们通过一个实例来看一下怎么使用 Cookie 做登录的操作。有些时候爬虫只有登录之后才能抓到网页中的信息。比如微博，知乎，人人网等。

关于 Cookie 的更详细信息，可以参看：<https://www.w3cschool.cn/pegosu/skj8lozt.html>

使用 Cookie 登录页面，见 Demo：

renrenTest.py

cookieJar.py

6.2.2 ##补充知识 cookiejar 的使用

Cookie 有时间限制，有域的限制，有编码问题等等。如果自己来管理 Cookie，会很繁琐，特别是当有多个 Cookie 需要管理时，想要很好的管理 Cookie 很困难。

当遇到网页登录后，返回 302 跳转的情况下，urllib2 的 Response 会丢失 Set-Cookie 的信息，导致登录不成功。

我们需要一个通用的能处理 Cookie 的工具来自动处理 Set-Cookie 请求；自动管理过期的 Cookie，自动在对应域下发特殊 Cookie；为了应对这些问题，我们引入了 CookieJar；

6.3 关于验证码（CAPTCHA）



网站为了防止黑客程序的恶意欺诈和攻击，采取的一种防御措施。据说最早是 paypal 这家公司引入的技术，现在已经在互联网网站中被广泛使用。

一般处理验证码 CAPTCHA 有两种方式：

- 1) 在需要输入验证码时程序弹出图片让用户自己输入；
- 2) 使用图像识别技术来识别图中的信息；

光学字符识别 OCR：OCR (Optical Character Recognition, 光学字符识别) 是指电子设备（例如扫描仪或数码相机）检查纸上打印的字符，通过检测暗、亮的模式确定其形状，然后用字符识别方法，将形状翻译成计算机文字的过程；

程序处理复杂验证码的方法：

1. 使用 Google 的开源项目 Tesseract；

安装 Tesseract：

Ubuntu 中安装：

```
sudo apt-get install tesseract-ocr
```

```
pip install pytesseract
```

训练与测试：<https://www.cnblogs.com/cnlian/p/5765871.html>

简单的 Python 测试代码：

```
from PIL import Image

from pytesseract import *

# 加载图片

image = Image.open('test1.jpg')

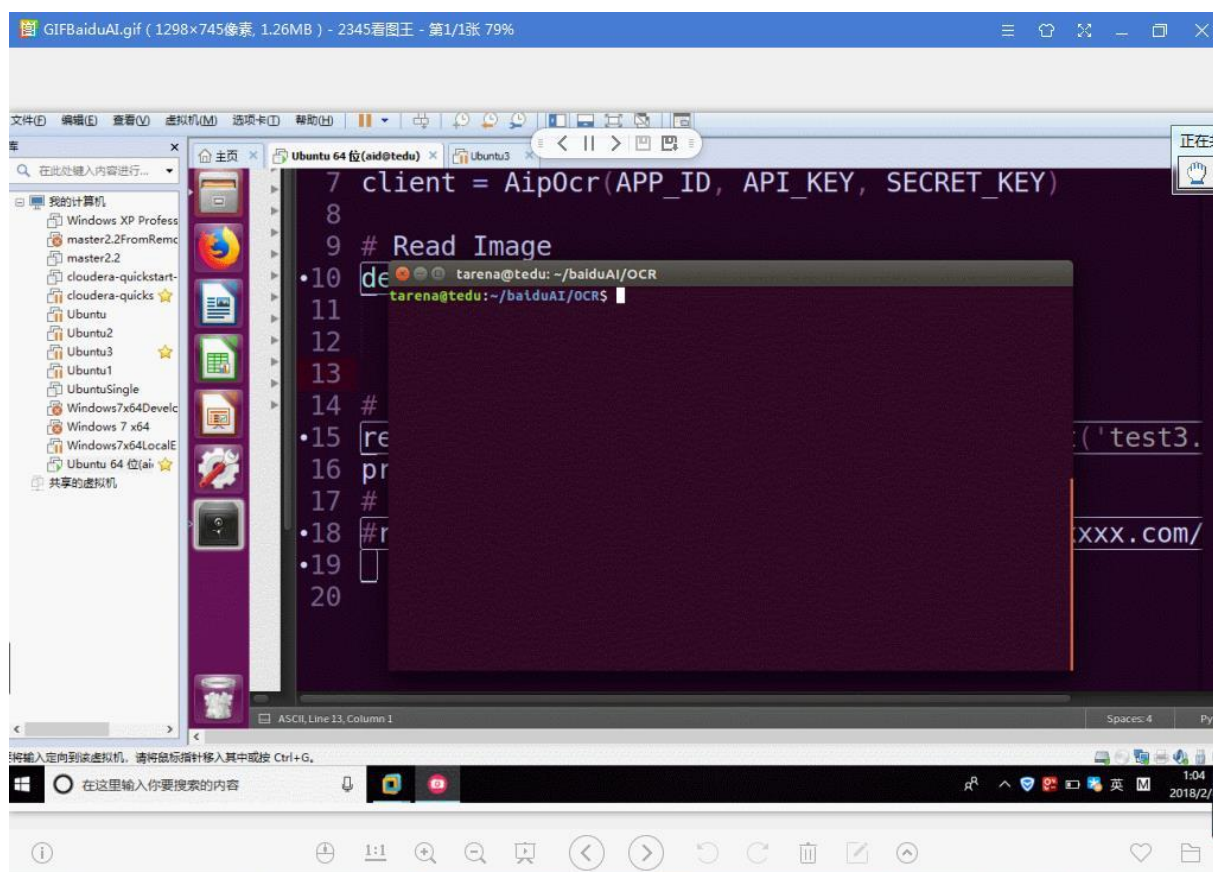
# 识别过程

text = image_to_string(image)

print(text)
```

Demo 见 SpiderCodes\TestOCR\

2. 使用百度 AI 等等：



两个 Demo：关于验证码识别的处理

第七章 爬虫的持久化问题

爬虫抓取回来的数据，怎么储存？

直接存储在文件中吗，还是存成 JSON 文件的格式呢。如果数据量巨大呢。我们抓回来的数据是需要将来能够被很好的查询和使用的。那怎么办呢？

7.1 MySQL

安装：

```
pip install PyMySQL
```

使用：

见 PyMysql.py

7.2 MongoDB

7.2.1 什么是 MongoDB

7.2.2 怎么在爬虫中使用 MongoDB

7.3 HDFS, HBase

第八章 高效率的爬取数据

人多力量大，多人干事更快。多个爬虫一起干活效率也更高。

随着爬虫抓取数据的提速，及数据量抓取的庞大，问题也会凸显。对方也会尝试用各种手段来禁止爬虫的行为。

8.1 多进程爬虫

8.2 多线程爬虫

8.2.1 关于 GIL

8.2.2 线程池的出场

8.3 关于代理服务器的设置

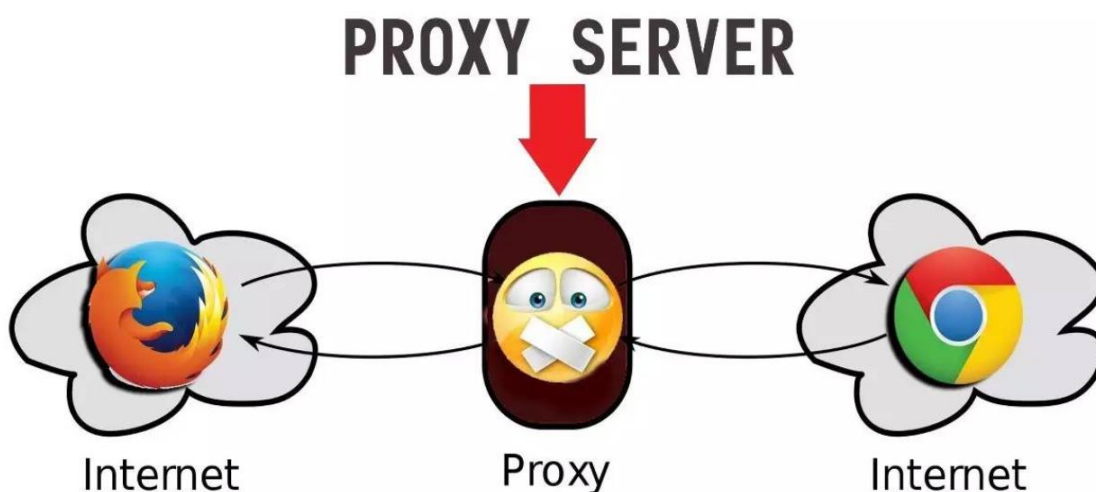
有过一定做爬虫经验的人应该都知道，抓的网站和数据多了，如果爬虫抓取速度过快，免不了触发网站的防爬机制，几乎用的同一招就是封 IP。

解决方案有 2 个：

- 1、同一 IP，放慢速度(爬取速度慢)
- 2、使用代理 IP 访问(推荐)

第一种方案牺牲的就是时间和速度，来换取数据，但是一般情况下我们的时间是很宝贵的，我们希望做到的是用最短的时间获取最多的数据，所以应该尽量使用第二种方案。

问题是从哪里能找到这么多代理 IP 呢？



这个问题可以问问搜索引擎。当然如果能结合爬虫程序来问搜索引擎那效率会更高。

代码示例:

testProxy.py

练习: <https://www.kuaidaili.com/ops/> 从这个网页中爬取代理服务器,

附录 A 中有 100 个代理服务器的地址, 请写个程序测试一个看看有多少个是可以正常使用的。

第九章 大数据量时的去重

随着抓取的数据量到一定程度，数据重复及爬取过程中的死链问题会凸显。怎么来解决这些问题呢？

9.1 怎么去重

去重是爬虫很重要的一个知识点，在一些例如 Scrapy 的框架中已经处理了去重的问题，我们的课程中会有专门的一部分来讲解去重的处理；

9.2 Redis 数据库

结合 Scrapy-Redis 来说；

9.2.1 关于 Redis

9.2.2 实际项目中使用 Redis

9.3 BloomFilter

第十章 Scrapy 框架

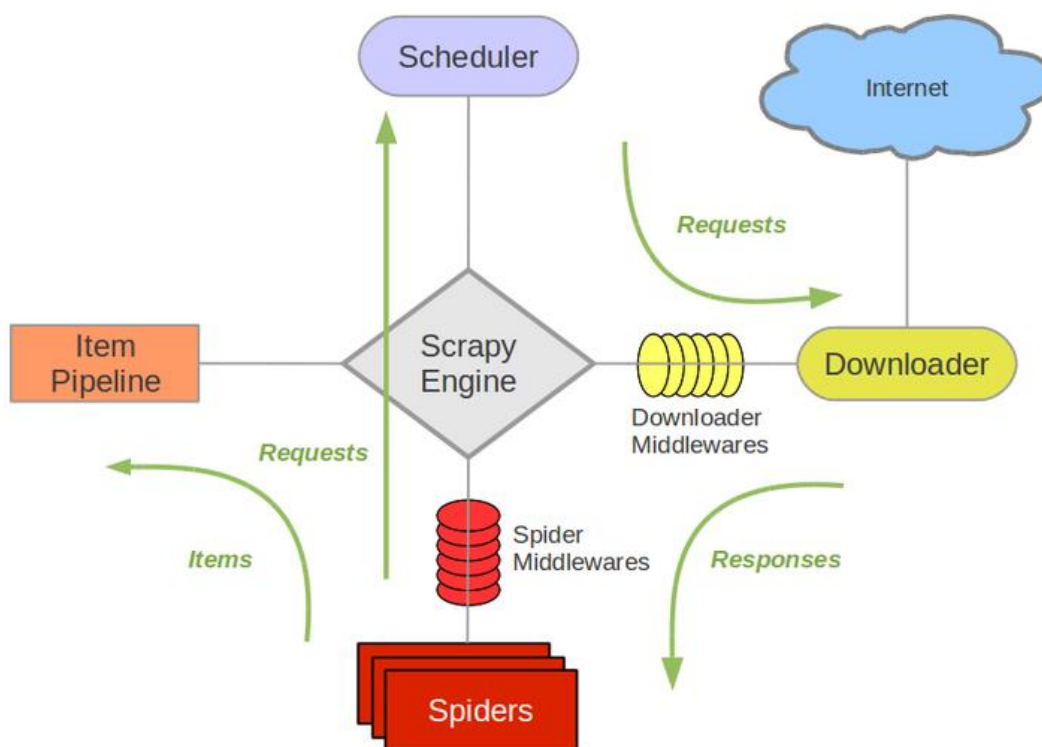
在了解了爬虫各种基础知识之后，我们有时需要快速搭建一个个爬虫的程序。有没有这么一个方便的工具或框架能让我们快速搭建起一个个爬虫程序呢？Scrapy 呼之欲出。

10.1 什么是 Scrapy

纯 Python 实现的一个为了爬取网站数据，提取结构性数据而编写的应用框架。框架本身把一些重复性的工作给你做好了；你就可以轻轻松松的按照其框架本身写几个简单的模块或者简单的扩展一些模块就可以你个性化的功能；当然带来的问题是首先你要学习了解框架，还有，想突破框架本身的限制，比较困难；

Scrapy 是基于 Twisted(竞争对手 Tornado)异步网络框架，Scrapy 的组件及架构图如下：

Scrapy架构图(绿线是数据流向)：



Scrapy Engine(引擎)：负责 Spider、ItemPipeline、Downloader、Scheduler 中间的通讯，信号、数据传递等。

Scheduler(调度器)：它负责接受引擎发送过来 Request 请求，并按照一定的方式进行整理排列，入队，当引擎需要时，交还给引擎。

Downloader（下载器）：负责下载 Scrapy Engine(引擎)发送的所有 Requests 请求，并将其获取到的 Responses 交还给 Scrapy Engine(引擎)，由引擎交给 Spider 来处理。

Spider（爬虫）：它负责处理所有 Responses, 从中分析提取数据，获取 Item 字段需要的数据，并将需要跟进的 URL 提交给引擎，再次进入 Scheduler(调度器)。

Item Pipeline(管道)：它负责处理 Spider 中获取到的 Item，并进行后期处理（详细分析、过滤、存储等）的地方。

Downloader Middlewares（下载中间件）：可以当作是一个可以自定义扩展下载功能的组件。

Spider Middlewares(Spider 中间件)：可以理解为是一个可以自定义扩展和操作引擎和 Spider 中间通信的功能组件（比如进入 Spider 的 Responses；和从 Spider 出去的 Requests）

10.2 怎么安装使用 Scrapy

下面运行的环境是 Ubuntu 17.04

10.2.1 安装

安装 Scrapy in Ubuntu:

```
sudo apt-get install python-dev python-pip libxml2-dev libxslt1-dev
sudo pip install scrapy
```

10.2.2 制作一个 Scrapy 爬虫需要的四个步骤

1) 新建项目 (scrapy startproject spiderName) 新建一个新的爬虫项目，一个项目可能包含很多个爬虫；

```
scrapy startproject tencentSpider
```

查看项目结构：

```
tarena@tedu:~/Spider/tencentSpider$ tree.:
```

```
├── scrapy.cfg
└── tencentSpider
    ├── __init__.py
    ├── items.py
    ├── middlewares.py
    ├── pipelines.py
    └── settings.py
```

```
└── spiders
    └── __init__.py
```

2 directories, 7 files

2) 明确目标：明确你想要抓取的目标，生产一个具体的爬虫

```
serapy genspider tencent
```

```
cd tencentSpider
```

```
scrapy genspider tencent hr.tencent.com
```

tarena@tedu:~/Spider/tencentSpider\$ tree

```
├── scrapy.cfg
├── tencentLog.txt
└── tencentSpider
    ├── __init__.py
    ├── __init__.pyc
    ├── items.py
    ├── middlewares.py
    ├── pipelines.py
    ├── settings.py
    ├── settings.pyc
    └── spiders
        ├── __init__.py
        ├── __init__.pyc
        └── tecent.py
```

2 directories, 12 files

下面需要具体取修改代码逻辑，按照我们的需求去实现自己的爬虫逻辑：

修改 settings.py 设置

pipelines.py 保存的逻辑

tecent.py, 抓取页面信息和继续跳转的逻辑

items.py 保存 item 的映射

3) 制作爬虫 (spiders/spiderName.py)：制作爬虫开始爬取网页；

4) 存储内容 (pipelines.py): 设计管道存储爬取内容;

5) 在 Scrapy 下启动爬虫:

scrapy crawl tencent

代码示例: tencentSpider, tencentSpider2

第十一章 反爬及应对反爬的策略

随着抓取的数据量到一定程度，数据重复及爬取过程中的死链问题会凸显。怎么来解决反爬问题呢？

11.1 网站如何发现爬虫

一般来说，网站会有以下一些简单的策略发现爬虫程序：

- 1) 单一 IP 非常规的访问频次；
- 2) 单一 IP 非常规的数据流量；
- 3) 大量重复简单的网站浏览行为，只下载网页，没有后续的 JS,CSS 请求；
- 5) 通过一些陷阱来发现爬虫，例如一些通过 CSS 对用户隐藏的链接，只有爬虫才会访问；

11.2 网站如何进行反爬

一般来说网站会采用下面两个简单的策略来防止爬虫：

1.大量使用动态网页，是的爬虫的爬取难度增加，重要数据都拿不到，即使爬虫采用了 Web 环境来渲染（内置浏览器），也会大大增加爬虫的负担和爬虫时间；（当然，采用动态加载的技术，对服务器的负担也会大大减轻）

2.基于流量的拒绝：

开启带宽限制模块，限制每个 IP 最多连接数，最大带宽等；

11.3 爬虫如何发现自己可能被网站识别了

如果爬取过程中出现以下情况，那么小心了，你的爬虫可能被网站发现了：

1. 验证码出现；
2. Unusual content delivery delay 非常规的延时；
3. Frequent response with HTTP 403, 404, 301 or 50x error;

11.4 爬虫应对反爬的策略

我们可以从以下几个方面来考虑应对反爬：

- 1) **User-Agent** 池;
- 2) 代理服务器池;
- 3) **CookieJar** 等的管理;
- 4) 协议的细节考虑, 如: **需要大量的实践经验总结的**

抓取数据时不处理 CSS, JS 等;

nofollow 属性; css 的 display 属性; 探测陷阱;

验证 refer locator 等;

5) **使用分布式的多机策略**: 爬慢点, 把爬虫放到访问频繁的主站 IP 子网下, 如教育网;

6) 使用了各种规则来尝试批量爬取, 然后对规则动态进行组合;

7) 验证码的搞定: 机器学习, 图像识别;

8) **尽可能遵守 Robots 协议**;

总结与进一步工作

这十一篇主要面对初级及中级爬虫工程师的参考资料。由于本人能力及知识有限，目前只能总结到这里。但是关于爬虫的知识和技术，互联网知识和技术更新换代非常快。后期本人会尽我所能，根据实际工程需要，增加新的实用的知识。

附录 A 收集到的 100 个可能能用的代理服务器

106.39.179.236:80
23.94.191.219:1080
121.41.175.199:80
122.183.139.98:8080
118.193.107.182:80
92.42.109.45:1080
128.199.77.93:8080
46.101.60.239:8118
185.106.121.98:1080
185.82.203.81:1080
112.114.93.27:8118
104.131.69.203:80
138.201.0.184:1080
46.101.46.174:8118
178.62.123.38:8118
217.23.15.193:1080
60.168.207.208:8010
139.59.170.110:8118
223.241.118.228:8010
123.192.114.113:80
103.37.95.110:8000
180.179.43.250:80
185.117.74.81:1080
116.199.2.196:80
118.193.107.119:80
128.199.77.93:8000
170.246.114.213:8080
104.243.47.146:1080
111.3.108.44:8118
124.42.7.103:80
39.134.161.18:80
146.185.156.221:8118

47.89.249.110:80
118.193.107.192:80
124.232.163.10:3128
223.19.105.206:80
46.166.168.243:1080
118.114.77.47:8080
182.253.205.85:8090
45.55.132.29:9999
58.251.227.238:8118
118.193.107.142:80
118.193.107.135:80
118.193.107.219:80
46.101.45.212:8118
114.249.45.176:8118
80.152.201.116:8080
94.177.254.86:80
197.155.158.22:80
196.200.173.83:80
212.237.10.45:8080
188.166.144.173:8118
210.71.198.230:8118
177.114.228.112:8080
218.50.2.102:8080
198.204.251.158:1080
188.166.204.221:8118
185.117.74.126:1080
106.39.179.244:80
39.134.161.14:8080
85.10.247.136:1080
46.166.168.245:1080
5.167.50.35:3129
118.178.227.171:80
122.96.59.102:82
52.174.89.111:80

103.25.173.237:808
121.232.145.168:9000
103.251.167.8:1080
46.101.26.217:8118
171.37.178.175:9797
103.251.166.18:1080
186.225.176.93:8080
121.232.147.132:9000
104.224.168.178:8888
47.90.2.253:8118
121.232.145.82:9000
118.193.107.36:80
58.56.128.84:9001
139.59.153.59:80
122.183.139.101:8080
163.172.184.226:8118
198.204.251.146:1080
213.133.100.195:1080
42.104.84.106:8080
117.2.64.109:8888
121.232.144.229:9000
156.67.219.61:8080
138.36.106.90:80
1.179.233.66:80
222.33.192.238:8118
138.197.224.12:8118
151.106.10.6:1080
134.35.250.204:8080
58.251.227.233:8118
52.221.40.19:80
222.73.68.144:8090
46.166.168.247:1080
192.99.222.207:80
1.23.160.212:8080

附录 B Python2 与 3 urllib 库对照表

参照 <http://blog.csdn.net/whatday/article/details/54710403>

Python2	与	Python3	urllib 库对照表:
urllib.urlretrieve()			urllib.request.urlretrieve()
urllib.urlcleanup()			urllib.request.urlcleanup()
urllib.quote()			urllib.parse.quote()
urllib.quote_plus()			urllib.parse.quote_plus()
urllib.unquote()			urllib.parse.unquote()
urllib.unquote_plus()			urllib.parse.unquote_plus()
urllib.urlencode()			urllib.parse.urlencode()
urllib.pathname2url()			urllib.request.pathname2url()
urllib.url2pathname()			urllib.request.url2pathname()
urllib.getproxies()			urllib.request.getproxies()
urllib.URLopener			urllib.request.URLopener
urllib.FancyURLopener			urllib.request.FancyURLopener
urllib.ContentTooShortError			urllib.error.ContentTooShortError
urllib2.urlopen()			urllib.request.urlopen()
urllib2.install_opener()			urllib.request.install_opener()
urllib2.build_opener()			urllib.request.build_opener()
urllib2.URLError			urllib.error.URLError
urllib2.HTTPError			urllib.error.HTTPError
urllib2.Request			urllib.request.Request
urllib2.OpenerDirector			urllib.request.OpenerDirector
urllib2.BaseHandler			urllib.request.BaseHandler
urllib2.HTTPDefaultErrorHandler			urllib.request.HTTPDefaultErrorHandler
urllib2.HTTPRedirectHandler			urllib.request.HTTPRedirectHandler
urllib2.HTTPCookieProcessor			urllib.request.HTTPCookieProcessor
urllib2.ProxyHandler			urllib.request.ProxyHandler
urllib2.HTTPPasswordMgr			urllib.request.HTTPPasswordMgr
urllib2.HTTPPasswordMgrWithDefaultRealm			urllib.request.HTTPPasswordMgrWithDefaultRealm
urllib2.AbstractBasicAuthHandler			urllib.request.AbstractBasicAuthHandler

<code>urllib2.HTTPBasicAuthHandler</code>	<code>urllib.request.HTTPBasicAuthHandler</code>
<code>urllib2.ProxyBasicAuthHandler</code>	<code>urllib.request.ProxyBasicAuthHandler</code>
<code>urllib2.AbstractDigestAuthHandler</code>	<code>urllib.request.AbstractDigestAuthHandler</code>
<code>urllib2.HTTPDigestAuthHandler</code>	<code>urllib.request.HTTPDigestAuthHandler</code>
<code>urllib2.ProxyDigestAuthHandler</code>	<code>urllib.request.ProxyDigestAuthHandler</code>
<code>urllib2.HTTPHandler</code>	<code>urllib.request.HTTPHandler</code>
<code>urllib2.HTTPSHandler</code>	<code>urllib.request.HTTPSHandler</code>
<code>urllib2.FileHandler</code>	<code>urllib.request.FileHandler</code>
<code>urllib2.FTPHandler</code>	<code>urllib.request.FTPHandler</code>
<code>urllib2.CacheFTPHandler</code>	<code>urllib.request.CacheFTPHandler</code>
<code>urllib2.UnknownHandler</code>	<code>urllib.request.UnknownHandler</code>