**Web crawler**

**（1）：爬虫基础知识**

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。即“爬”到对应的网页上，再把需要的信息“铲”下来。

**语言&环境**

Python语言

urllib.request：这是Python自带的库，不需要单独安装，它的作用是为我们打开url获取html的内容。

BeautifulSoup：是一个可以从HTML或XML文件中提取数据的Python库。

**爬取数据**

输出http://jr.jd.com 这个网页的全部HTML代码:

from urllib.request import urlopen

html = urlopen('http://jr.jd.com')

print(html.read())

html.close()

用Chrome打开url，然后按Command + Shift + C（需先设置为开发者模式）。如果我们要抓导航栏，就把鼠标位置移动到任意一个导航栏的项目，浏览器就在html中找到了它的位置。

使用BeautifulSoup提取数据：

from urllib.request import urlopen

from bs4 import BeautifulSoup

html = urlopen('http://jr.jd.com')

bs\_obj = BeautifulSoup(html.read(),'html.parser')

text\_list = bs\_obj.find\_all('a','nav-item-primary')

for text in text\_list:

print(text.get\_text())

html.close()

BeautifulSoup提供一些简单的、Python式的函数用来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。

**一个迷你爬虫**

爬取网易云音乐播放数大于500万的歌单。

**动态网页**：所谓的动态网页，是指跟静态网页相对的一种网页编程技术。静态网页，随着html代码的生成，页面的内容和显示效果就基本上不会发生变化了——除非你修改页面代码。而动态网页则不然，页面代码虽然没有变，但是显示的内容却是可以随着时间、环境或者数据库操作的结果而发生改变的。

之前的技术不能执行那些让页面产生各种神奇效果的JavaScript 代码。

解决问题的途径：用Python 的第三方库运行JavaScript，直接采集在浏览器里看到的页面。

Selenium：是一个强大的网络数据采集工具，其最初是为网站自动化测试而开发的。近几年，它还被广泛用于获取精确的网站快照，因为它们可以直接运行在浏览器上。Selenium 库是一个在WebDriver 上调用的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。

Selenium 自己不带浏览器，它需要与第三方浏览器结合在一起使用。但用浏览器打开网页效率太低，所以用一个叫PhantomJS的工具代替真实的浏览器。

PhantomJS：是一个“无头”（headless）浏览器。它会把网站加载到内存并执行页面上的JavaScript，但是它不会向用户展示网页的图形界面。

url = 'http://music.163.com/#/discover/playlist/'\

'?order=hot&cat=%E5%85%A8%E9%83%A8&limit=35&offset=0'

driver = webdriver.PhantomJS(executable\_path='/Users/chenjiangong/Documents/phantomjs-2.1.1-macosx/bin/phantomjs')

csv\_file = open('playlist.csv','w',newline='')

writer = csv.writer(csv\_file)

writer.writerow(['title','playtimes','link'])

while url != 'javascript:void(0)':

driver.get(url)

driver.switch\_to.frame('contentFrame')

data = driver.find\_element\_by\_id('m-pl-container').\

find\_elements\_by\_tag\_name('li')

for i in range(len(data)):

nb = data[i].find\_element\_by\_class\_name('nb').text

if '万' in nb and int(nb.split('万')[0])>500:

msk = data[i].find\_element\_by\_css\_selector('a.msk')

writer.writerow([msk.get\_attribute('title'),

nb,msk.get\_attribute('href')])

url = driver.find\_element\_by\_css\_selector('a.zbtn.znxt').\

get\_attribute('href')

csv\_file.close()

**（2）：迷你爬虫架构**

按照系统结构和实现技术，大致可以分为以下几种类型：通用网络爬虫（General Purpose Web Crawler）、聚焦网络爬虫（Focused Web Crawler）、增量式网络爬虫（Incremental Web Crawler）、深层网络爬虫（Deep Web Crawler）。

**通用网络爬虫**

通用网络爬虫又称全网爬虫（Scalable Web Crawler），爬取对象从一些种子 URL 扩充到整个 Web。主要为门户站点搜索引擎和大型 Web 服务提供商采集数据。

常用的爬取策略有：深度优先策略、广度优先策略。

1) 深度优先策略（DFS）：其基本方法是按照深度由低到高的顺序，依次访问下一级网页链接，直到不能再深入为止。

2) 广度优先策略（BFS）：此策略按照网页内容目录层次深浅来爬取页面，处于较浅目录层次的页面首先被爬取。 当同一层次中的页面爬取完毕后，爬虫再深入下一层继续爬取。

**聚焦网络爬虫**

聚焦网络爬虫（Focused Crawler），又称主题网络爬虫（Topical Crawler），是指选择性地爬取那些与预先定义好的主题相关页面的网络爬虫。

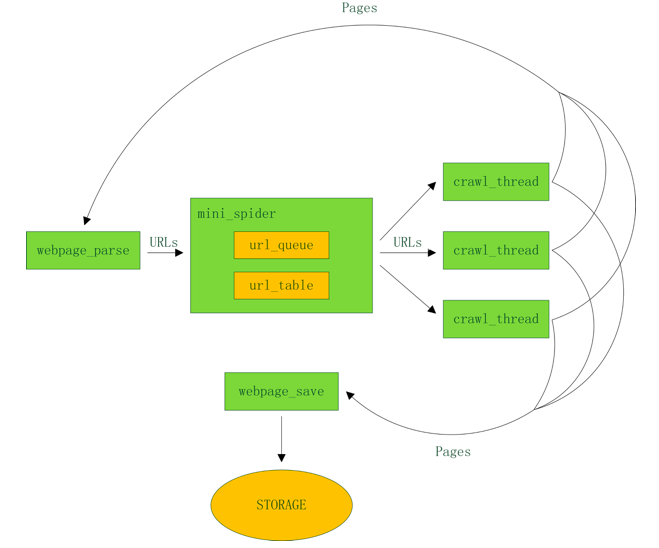
**增量式网络爬虫**

增量式网络爬虫（Incremental Web Crawler）是指对已下载网页采取增量式更新和只爬取新产生的或者已经发生变化网页的爬虫，它能够在一定程度上保证所爬取的页面是尽可能新的页面。

**深网爬虫**

Deep Web 是那些大部分内容不能通过静态链接获取的、隐藏在搜索表单后的，只有用户提交一些关键词才能获得的 Web 页面。

**一个迷你框架**



Step 1. 采用BFS还是DFS？

在极端情况下，如果只能下载非常有限的网页，那么应该下载的所有网站的首页，如果把爬虫再扩大些，应该爬取从首页直接链接的网页。在这个前提下，显然BFS明显优于DFS。

Step 2. 初始URL集合、URL队列

假设从一家门户网站的首页出发，先下载这个网页（深度=0），然后通过分析这个网页，可以找到页面里的所有超链接，也就等于知道了这家门户网站首页所直接连接的全部网页，诸如京东理财、京东白条，京东众筹等（深度=1）。接下来访问、下载并分析京东理财等网页，又能找到其他相连的网页（深度=2）。让计算机不停的做下去，就能下载整个网站。

在这个过程中，我们需要一个“初始URL集合”保存门户的首页，还需要一个“URL队列”保存分析网页得到的超链接。

Step 3. 记录哪些网页已经下载过的小本本——URL表。

为了防止一个网页被下载和解析多次，需要一个URL表记录哪些网页已经下载过。再遇到这个网页的时候，我们就可以跳过它。

Step 4. 多个抓取线程

Python里一个进程永远只能同时执行一个线程，所以在多核CPU上，Python的多线程效率并不高。Python下的多线程对CPU密集型代码并不友好。但对IO密集型代码比较友好。

Step 5. 页面分析模块

从网页中解析出URLs或者其他有用的数据。

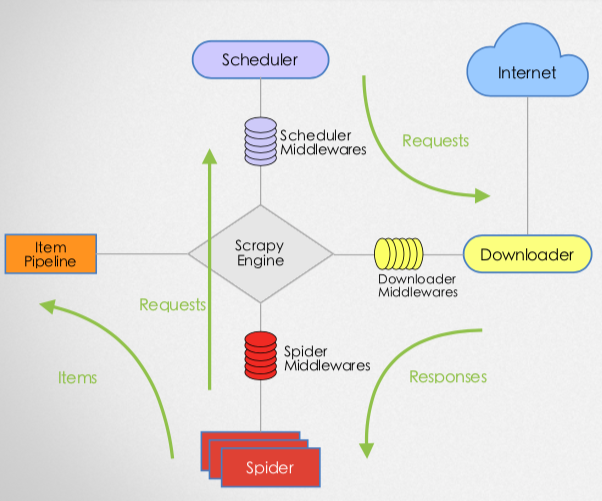
Step 6. 页面存储模块

保存页面的模块，目前将文件保存为文件，以后可以扩展出多种存储方式

**（3）：Scrapy入门**

**架构简介**

组件：



Engine: 引擎负责控制数据流在系统中所有组件中流动，并在相应动作发生时触发事件。

Scheduler: 调度器从引擎接受Request并将他们入队，以便之后引擎请求他们时提供给引擎。

Downloader: 下载器负责获取页面数据并提供给引擎，而后提供给Spider。

Spiders: Spider是Scrapy用户编写的用于分析Response并提取Item或提取更多需要下载的URL的类。 每个Spider负责处理特定网站。

Item Pipeline: 负责处理被Spider提取出来的Item。典型的功能有清洗、 验证及持久化操作。

Downloader middlewares: 下载器中间件是在Engine及Downloader之间的特定钩子(specific hooks)，处理Downloader传递给Engine的Response。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。

Spider middlewares: 是在Engine及Spider之间的特定钩子(specific hook)，处理Spider的输入(Response)和输出(Items及Requests)。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。

数据流：

Scrapy中的数据流由执行引擎控制，其过程如下:

1.Engine从Spider获取第一个需要爬取URL(s)。

2.Engine用Scheduler调度Requests，并向Scheduler请求下一个要爬取的URL。

3.Scheduler返回下一个要爬取的URL给Engine。

4.Engine将URL通过Downloader middlewares转发给Downloader。

5.一旦页面下载完毕，下载器生成一个该页面的Response，并将其通过Downloader middlewares发送给Engine。

6.引擎从Downloader中接收到Response并通过Spider middlewares发送给Spider处理。

7.Spider处理Response并返回爬取到的Item及新的Request给Engine。

8.Engine将爬取到的Item给Item Pipeline，然后将Request给Scheduler。

9.从第一步开始重复这个流程，直到Scheduler中没有更多的URLs。

**入门教程**

**注意：**

运行scrapy时报语法错误，是因为Python3.7中async变成了关键字，所以用户定义的变量就不能再叫async了。暂时不要使用python3.7版本运行scrapy

**创建项目:**

进入打算存储代码的目录中，用terminal运行下列命令:

scrapy startproject tutorial

该命令将会创建包含下列内容的 tutorial 目录:

tutorial/

scrapy.cfg # 项目的配置文件

tutorial/ # 该项目的python模块。之后您将在此加入代码

\_\_init\_\_.py

items.py # 项目中的item文件

pipelines.py # 项目中的pipelines文件

settings.py # 项目的设置文件

spiders/ # 放置spider代码的目录

\_\_init\_\_.py

**定义Item:**

Items是将要装载抓取的数据的容器，它工作方式像 python 里面的字典，但它提供更多的保护，比如对未定义的字段填充以防止拼写错误。

通过创建scrapy.Item类, 并且定义类型为 scrapy.Field 的类属性来声明一个Item.

我们通过将需要的item模型化，来控制从 dmoz.org 获得的站点数据，比如我们要获得站点的名字，url 和网站描述，我们定义这三种属性的域。在 tutorial 目录下的 items.py 文件编辑:

from scrapy.item import Item, Field

class QuotesItem(Item):

name = Filed()

description = Field()

url = Field()

**编写第一个爬虫:**

Spider是用户编写用于从单个网站(或者一些网站)爬取数据的类。其包含了一个用于下载的初始URL，以及如何跟进网页中的链接以及如何分析页面中的内容的方法。

为了创建一个Spider，必须继承 scrapy.Spider 类， 且定义以下三个属性:

name: 用于区别Spider。 该名字必须是唯一的，您不可以为不同的Spider设定相同的名字。

start\_urls: 包含了Spider在启动时进行爬取的url列表。 因此，第一个被获取到的页面将是其中之一。 后续的URL则从初始的URL获取到的数据中提取。

parse() 是spider的一个方法。 被调用时，每个初始URL完成下载后生成的Response 对象将会作为唯一的参数传递给该函数。 该方法负责解析返回的数据(response data)，提取数据以及生成需要进一步处理的URL的 Request 对象。

import scrapy

class QuotesSpider(scrapy.Spider):

name = "quotes"

def start\_requests(self):

urls = [

'http://quotes.toscrape.com/page/1/',

'http://quotes.toscrape.com/page/2/',

]

for url in urls:

yield scrapy.Request(url=url, callback=self.parse)

def parse(self, response):

page = response.url.split("/")[-2]

filename = 'quotes-%s.html' % page

with open(filename, 'wb') as f:

f.write(response.body)

self.log('Saved file %s' % filename)

**运行爬虫：**

进入项目的根目录(在terminal中输入cd /Users/chenjiangong/tutorial)，执行下列命令启动spider:

scrapy crawl quotes

这个命令启动用于爬取 quotes.toscrape.com 的spider

**提取数据：**

之前只是保存了HTML页面，并没有提取数据。现在升级一下代码，把提取功能加进去：

class QuotesSpider(scrapy.Spider):

name = 'quotes'

start\_urls = [

'http://quotes.toscrape.com/page/1/',

'http://quotes.toscrape.com/page/2/',

]

def parse(self,response):

for quote in response.css('div.quote'):

yield{

'text':quote.css('span.text::text').extract\_first(),

'author':quote.css('small.author::text').extract\_first(),

'tags':quote.css('div.tags a.tag::text').extract(),

}

再次运行这个爬虫，即可在日志里看到被提取出的数据。

**保存爬取的数据：**

最简单存储爬取的数据的方式是使用 Feed exports:

scrapy crawl quotes -o quotes.json

该命令将采用 JSON 格式对爬取的数据进行序列化，生成quotes.json文件。但只适用于简单、小规模的项目。

介绍Selector:

Scrapy使用了一种基于 XPath 或者 CSS 表达式机制： Scrapy Selectors

XPath表达式的例子及对应的含义:

/html/head/title: 选择HTML文档中 标签内的 元素

/html/head/title/text(): 选择 元素内的文本

//td: 选择所有的 元素

//div[@class=”mine”]: 选择所有具有class=”mine” 属性的 div 元素等

为了方便使用 XPaths，Scrapy 提供 Selector 类， 有四种方法 :

xpath()：返回selectors列表, 每一个selector表示一个xpath参数表达式选择的节点.

css() : 返回selectors列表, 每一个selector表示CSS参数表达式选择的节点

extract()：返回一个unicode字符串，该字符串为XPath选择器返回的数据

re()： 返回unicode字符串列表，字符串作为参数由正则表达式提取出来

**规避反爬虫机制：**

在请求中加入UserAgent的信息：

headers = {

'User-Agent':'Mozilla/5.0(Windows NT 6.1;Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.62 Safari/537.36'

}

def start\_request(self):

url = "https://movie.douban.com/top250"

yield Request(url, headers=self.headers)

**实现自动翻⻚：**

这部分代码仍在parse函数内部：

next\_url = response.xpath('//span[@class="next"]/a/@href').extract()

if next\_url:

next\_url = 'https://movie.douban.com/top250'+next\_url[0]

yield Request(next\_url,headers=self.headers)

**运行爬⾍程序:**

scrapy crawl douban -o douban.csv

运行该命令后，爬取到的数据保存为了csv格式，里面的内容包含中文，选中文件后点击空格预览，也能正常显示，但用Excel打开时，就变成了乱码

究其原因是Mac下文本的默认编码是UTF-8，而Excel对中文的处理是GBK编码。

解决方法：

将其处理格式转换为utf-8编码格式