爬蟲爬取動態網頁的三種方式簡介

語言: CN (//www.gushiciku.cn/pl/2RAJ) / TW (//www.gushiciku.cn/pl/2RAJ/zh-tw) / HK (//www.gushiciku.cn/pl/2RAJ/zh-hk)
時間 2019-03-05 22:35:02 K0rz3n's Blog (/pla/K0rz3n's Blog)
主題: JavaScript (/plt/JavaScript)

最近在看類似的問題的時候找了一些資料，發現網上有一篇文章寫得很詳細(準確的說是分成三篇文章寫的)，特別是手工逆向的方式還是挺有趣的，我也照著他的方式嘗試了一下，學到一點東西，下面是這三篇文章的部分內容(有刪改，外加其它的一些理解),如果想看原文的話，我在本文最後會附上原文的連結，至於目前最流行的使用 chrome headless 寫動態爬蟲的方法，由於原作者寫的也不是很仔細，所以我還要再找些資料仔細研究一下，後面再寫一篇文章總結。

0X01 動態網頁簡介:

在我們編寫爬蟲時，可能會碰到以下兩種問題:

1.我們所需要爬取的資料在網頁原始碼中並不存在;

2.點選下一頁跳轉頁面時，網頁的 URL 並沒有發生變化;

造成這種問題原因是，你所正在爬取的頁面採取了 js 動態載入的方式，是一個動態網頁。

所謂的動態網頁，是指跟靜態網頁相對的一種網頁程式設計技術。靜態網頁，隨著html程式碼生成，頁面的內容和顯示效果就不會發生變化了。而動態網頁則不然，其顯示的頁面則是經過Javascript處理資料後生成的結果，可以發生改變。 這些資料的來源有多種，可能是經過Javascript計算生成的，也可能是通過Ajax載入的。

動態網頁經常使用的一種技術是Ajax請求技術。

Ajax = Asynchronous JavaScript and XML（非同步的 JavaScript 和XML），其最大的優點是在 **不重新載入整個頁面的情況下**，可以與伺服器交換資料並更新部分網頁的內容。

目前，越來越多的網站採取的是這種動態載入網頁的方式，一來是可以實現web開發的前後端分離，減少伺服器直接渲染頁面的壓力； **二來是可以作為反爬蟲的一種手段。**

0X02 動態網頁抓取

(1)逆向回溯法

對於動態載入的網頁，我們想要獲取其網頁資料， **需要了解網頁是如何載入資料的**，該過程就被成為逆向回溯。

對於使用了Ajax 請求技術的網頁，我們可以找到Ajax請求的具體連結，直接得到Ajax請求得到的資料。

需要注意的是，構造Ajax請求有兩種方式:

1.原生的Ajax請求：會直接建立一個XMLHTTPRequest物件。

2.呼叫jQuery的ajax()方法：一般情況下， `$.ajax()` 會返回其建立的XMLHTTPRequest物件；但是，如果 `$.ajax()` 的dataType引數指定了為script或jsonp型別， `$.ajax()` 不再返回其建立的XMLHTTPRequest物件。

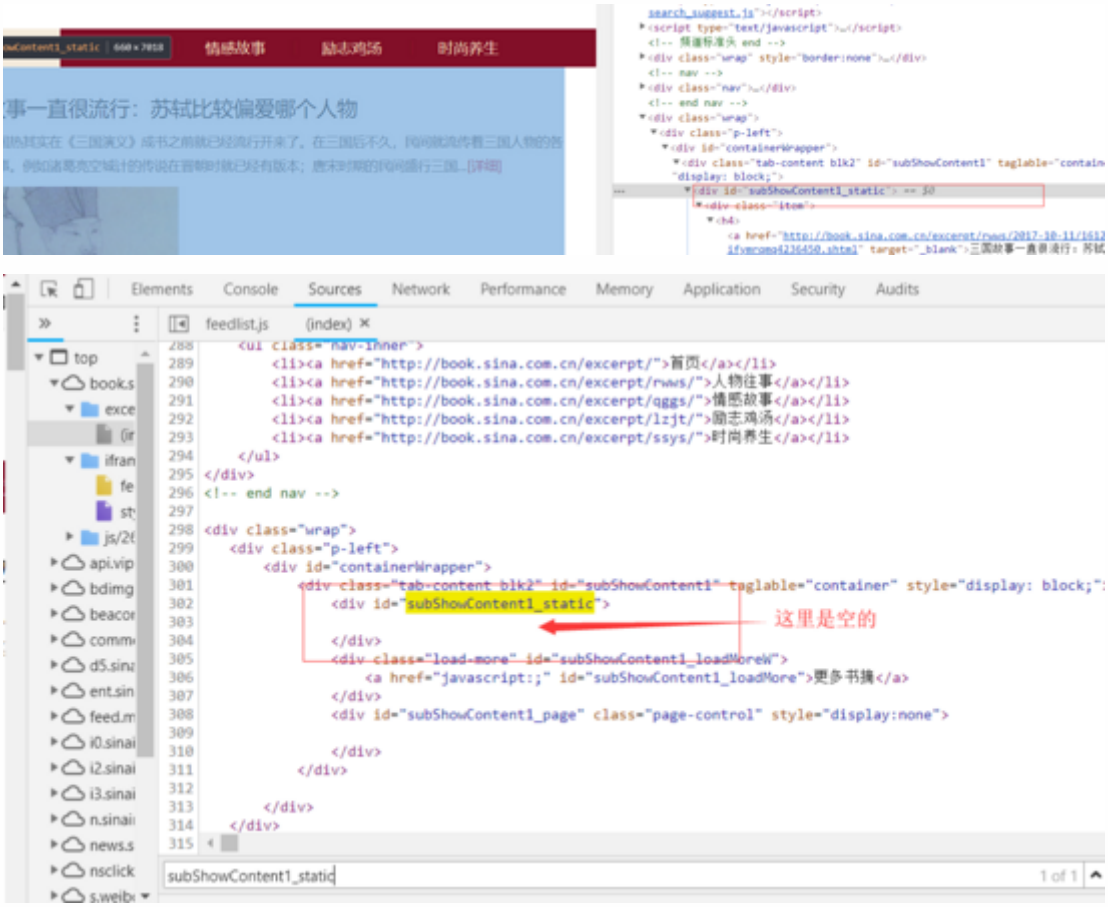
對於這兩種方式，只要建立並返回了XMLHTTPRequest物件，就可以通過Chrome瀏覽器的除錯工具在NetWork視窗設定過濾條件為 xhr ，直接篩選出Ajax請求的連結；如果是$.ajax()並且dataType指定了為script或jsonp (這種情況下NetWork 裡面的 Type 都是 script，如果你懂得 jsonp 的原理的話就知道 jsonp 本質就是通過 script)，則無法通過這種方式篩選出來(因為這兩種方式是經典的跨域方法，而 XHR 是不能跨域的，所以設定 XHR 過濾)。

示例：

接下來以 新浪讀書——書摘 (/jump/aHR0cDovL2Jvb2suc2luYS5jb20uY24vZXhjZXJwdC8=) 為例，介紹如何得到無法篩選出來的Ajax請求連結:

在Chrome中開啟網頁，右鍵檢查，會發現首頁中書摘列表包含在一個id為subShowContent1_static的div中，而檢視網頁原始碼會發現id為subShowContent1_static的div為空。

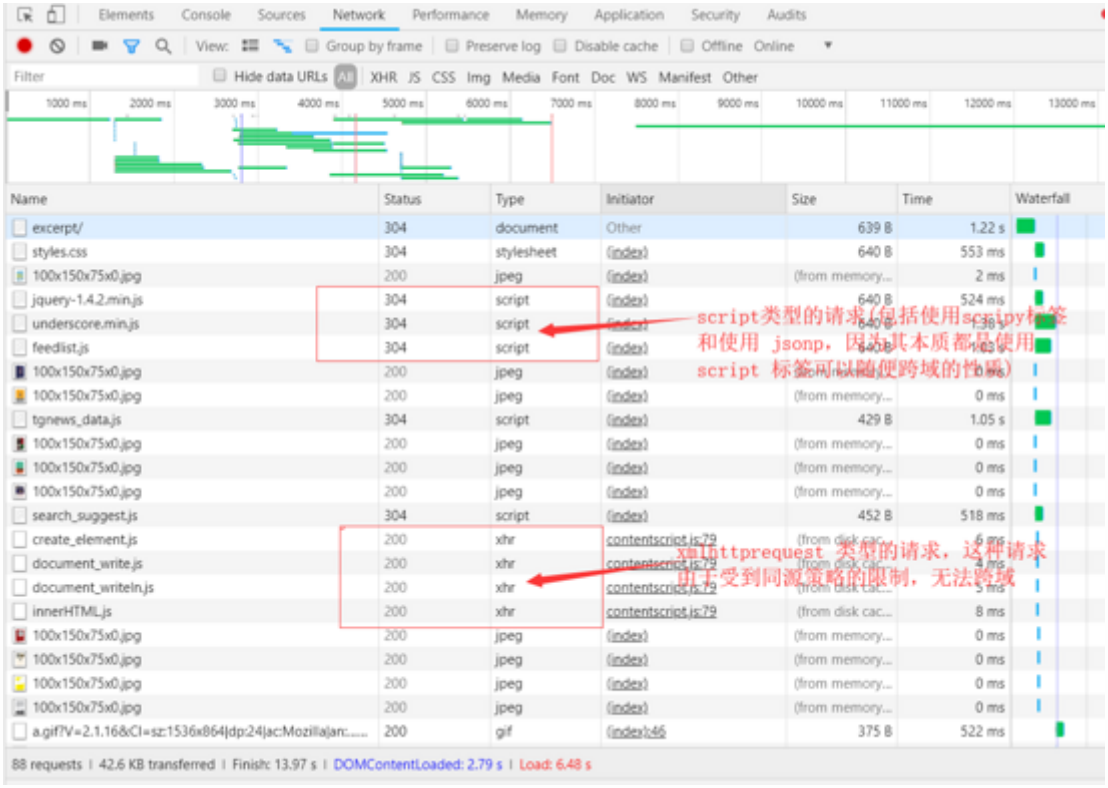
如圖所示：



並且點選更多書摘或下一頁時，網頁URL並沒有發生變化。

這與我們最前面所說的兩種情況相同，說明這個網頁就是使用 JS 動態載入資料的。

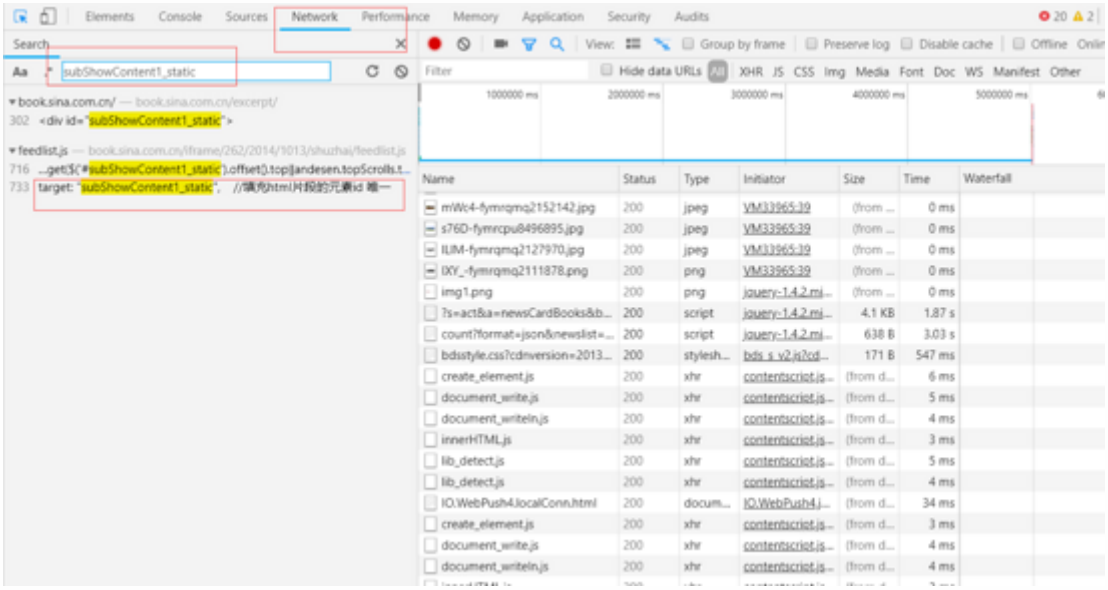
F12開啟除錯工具，開啟NetWork視窗，F5重新整理，可以看到瀏覽器傳送以及接收到的資料記錄(我們可以點選上面的 XHR 或者 JS 對這些請求進行過濾)：



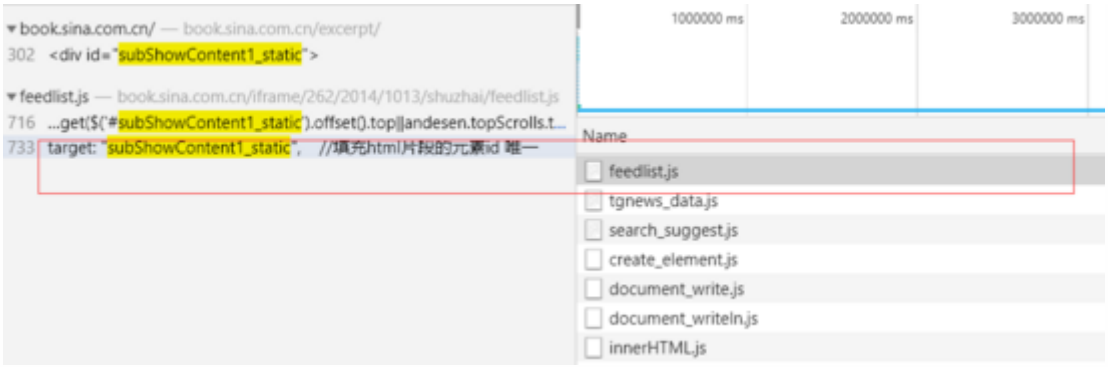
可以發現目前兩種型別的請求都是存在的，暫時還不能判斷我們 **div 中內容** 的動態載入使用的是哪一種方式，不過沒關係，我們可以進一步進行測試。

1.根據 id 進行查詢

我們知道,js 操作頁面的資料一定要進行定位，最常用的方法就是使用 id 定位，因為 id 在整個頁面中是唯一的，那麼我們第一步就是在所有的 js 檔案中找和 subShowContent1_static 這個 id 相關的檔案，於是我在 network 頁面使用 ctrl+f 進行全域性搜尋



最終定位到了可能性最大的檔案 feedlist.js



進入這個檔案以後我就定位到了一個匿名函式 $(),這個函式將引數傳入 Listmore() 函式

```
$(function () {
    var listurl = "http://feed.mix.sina.com.cn/api/roll/get";
    var moreListConfig = [
        //最新
        {
            moduleid: "moduleid1",//模块id 唯一
            url: listurl,          //请求数据的url
            target: "subShowContent1_static",     //填充html片段的元素id 唯一
            data: moreListConfigData,
            morebtnId: "subShowContent1_loadMore",//“加载更多”按钮id 唯一
            pageTurnId: "subShowContent1_page"
        }
    ];

    andesen.objMoreList = new andesen.ListMore(moreListConfig);
    //moduleid0 是对应的模块id,firstLoad方法只用于实现第一次加载
    //eg:objMoreList.firstLoad(["moduleid0","moduleid1","moduleid2"]);
    andesen.objMoreList.firstLoad(["moduleid1"]);
//    $("#andsen_ad1").smartFloat("tab-nav-fixed", '.part').css({"background-color": "#fff",
    $("#gul").smartFloat("tab-nav-fixed", '.wrap').css({"background-color":"#fff","width":"30

});
```

listmore() 函式呼叫了 Getmorelist() 函式

```
var ListMore = function (options) {
    if (!$.isArray(options)) {
        return;
    }
    var len,
        i,
        obj = {},
        unique = {};
    for (i = 0, len = options.length; i < len; i++) {

        //如果unique对象中有这个moduleid属性，说明当前moduleid和前面有重复的，则跳出本次循环，进行下一次循环
        //如果传了重复的moduleid,相同的moduleid这只生成一个对象
        //如果传递的moduleid错误或不存在，只生成对应的空对象，该空对象能够调用其原型对象的方法，
        //空对象调用getMore会直接return,retoreList会return false,sucess回报模版错
        if (unique[options[i].moduleid]) {
            continue;
        } else {
            unique[options[i].moduleid] = 1;
        }
        obj[options[i].moduleid] = new Getmorelist(options[i]);
    }
    this.obj = obj;
}
```

Getmorelist() 函式 呼叫了 getMore() 函式

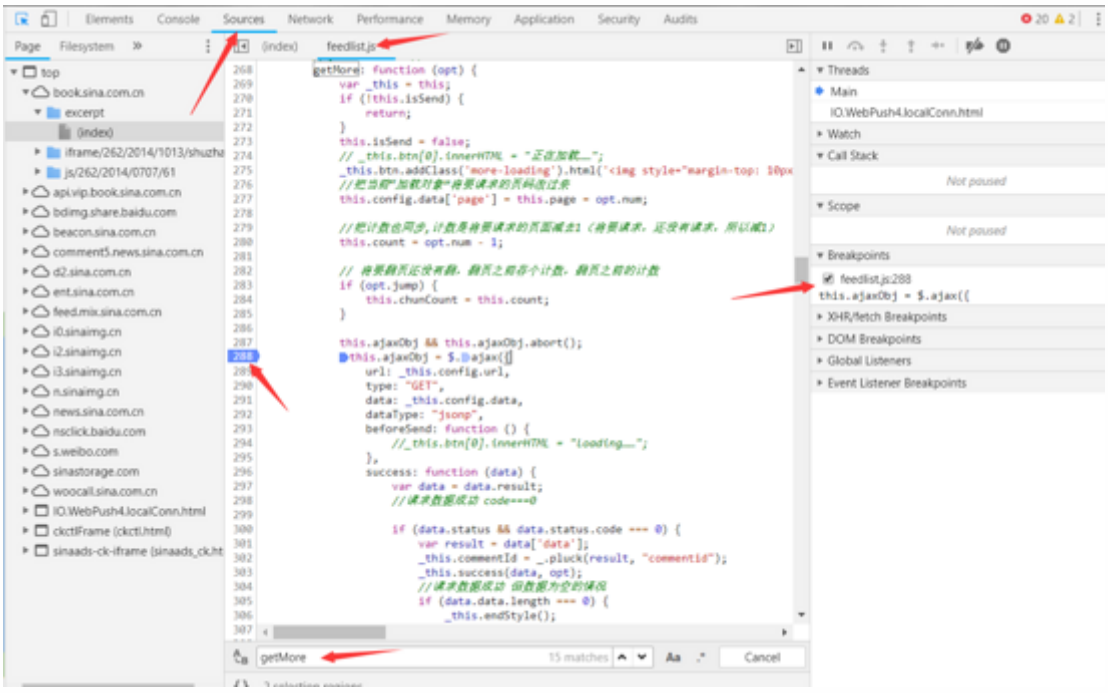
```
//查看更多按钮
this.btn = $("#" + config.morebtnId);
this.btn.bind('click', function (event) {
    // ie6 hack
    if(navigator.userAgent.toLowerCase().indexOf("msie 6")>-1){
        setTimeout(function(){
            _this.getMore({num: _this.page, jump: false});
        },16);
    }else{
        _this.getMore({num: _this.page, jump: false});
    }
});
```

getmore() 函式定義了我們的請求

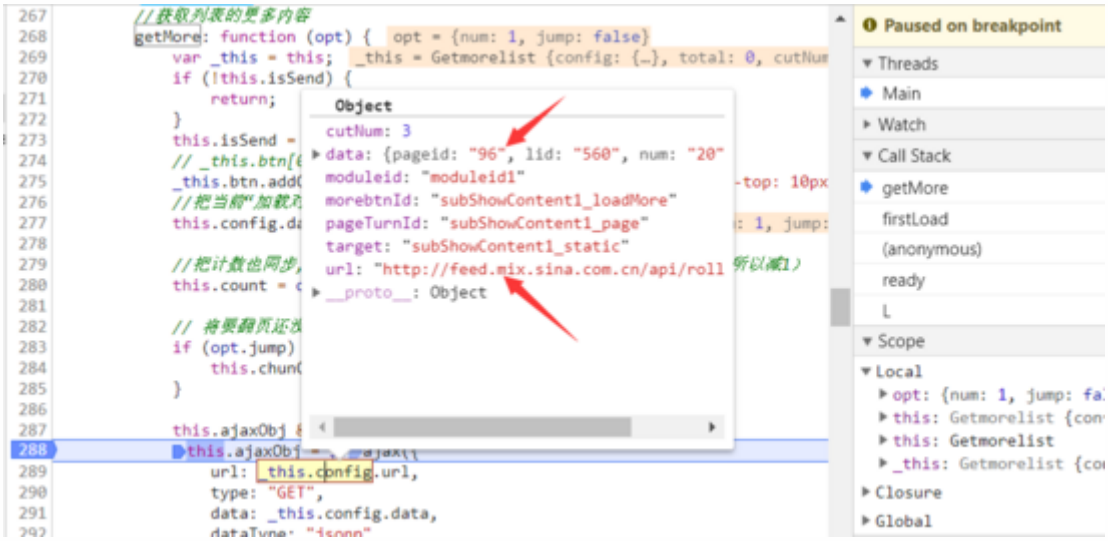
```
//获取列表的更多内容
getMore: function (opt) {
    var _this = this;
    if (!this.isSend) {
        return;
    }
    this.isSend = false;
    // _this.btn[0].innerHTML = "正在加载……";
    _this.btn.addClass('more-loading').html('<img style="margin-top: 10px" src="http://i2.sinaimg.
    //把当前“加载对象”将要请求的页码改过来
    this.config.data['page'] = this.page = opt.num;

    //把计数也同步,计数是将要请求的页面减去1（将要请求，还没有请求，所以减1）
    this.count = opt.num - 1;

    // 将要翻页还没有翻，翻页之前存个计数，翻页之前的计数
    if (opt.jump) {
        this.chunCount = this.count;
    }

    this.ajaxObj && this.ajaxObj.abort();
    this.ajaxObj = $.ajax({
        url: _this.config.url,
        type: "GET",
        data: _this.config.data,
        dataType: "jsonp",
        beforeSend: function () {
            //_this.btn[0].innerHTML = "loading……";
        },
```

2.設定斷點進行動態捕獲

可以看到這裡使用的是 jsonp 的形式跨域傳遞資料的，然後 URL 是一個物件，是執行中生成的，我們可以在執行中對這個函式新增一個斷點



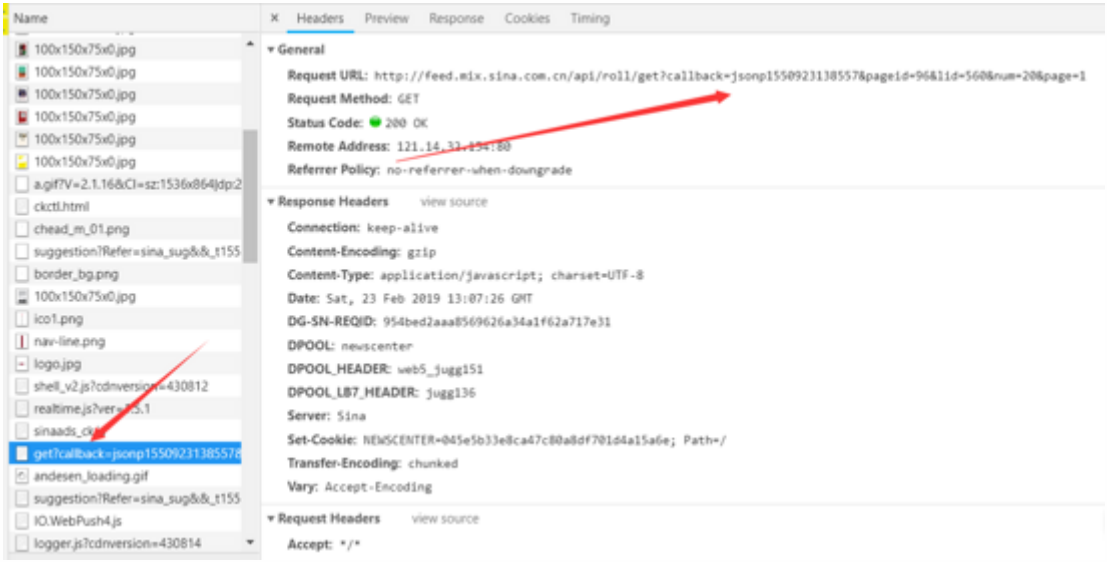
然後 f5 重新整理



斷下來以後就能看到我們想要看到的 URL 以及後面跟著的引數了，這樣就可以根據jQuery的ajax()用法構造正確的Ajax 請求連結：

```
http://feed.mix.sina.com.cn/api/roll/get?callback=xxxxxxxx&pageid=96&lid=560#=20&page=1
```

那麼這個 callback 是多少呢，我們現在還看不出來，但是，既然這個是一個請求，那麼肯定會在 network 中有記錄，我們找找看



我們現在就鎖定了我們想要找的連結，得到Ajax請求連結之後，可以直接得到請求的資料，一般為json格式，處理後即可使用。

注：

其實當你有了經驗之後，對一些不是很複雜的網頁，根本就不用進行這麼複雜的逆向工程，憑URL形式可以很快的在NetWork視窗 選擇-驗證 出所需的Ajax請求。

(2)渲染動態網頁法

1.瀏覽器渲染引擎：

(1)簡介：

在介紹這種方式之前，我們需要首先了解一些瀏覽器渲染引擎的基本知識。

渲染引擎的職責就是渲染，即在瀏覽器視窗中顯示所請求的內容。瀏覽器向伺服器傳送請求，得到伺服器返回的資原始檔後，需要經過渲染引擎的處理，將資原始檔顯示在瀏覽器視窗中。

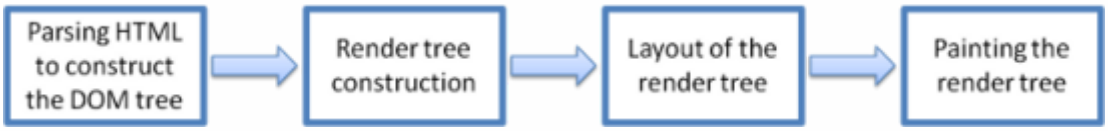
目前使用較為廣泛的渲染引擎有兩種：

```
webkit—使用者有Chrome, Safari
Geoko—使用者有Firefox
```

(2)渲染主流程：

渲染引擎首先通過網路獲得所請求文件的內容，通常以8K分塊的方式完成。

下面是渲染引擎在取得內容之後的基本流程：



解析html來構建dom樹 -> 構建render樹 -> 佈局render樹 -> 繪製render樹

渲染引擎開始解析html，並將標籤轉化為內容樹中的dom節點 。如果遇到JS，那麼此時會啟用另外的連線進行下載(下載過程中 dom 樹的構建不會停止)，並且在下載完成後立即執行(執行過程中會阻塞 瀏覽器的其他行為，因為 js 的執行可能會改變 dom 樹的結構，為了不讓剛剛構建好的 dom 樹又被 js 改變，聰明的瀏覽器停止了 dom 樹的構建)。

接著，它解析外部CSS檔案及style標籤中的樣式資訊。這些樣式資訊以及html中的可見性指令將被用來構建另一棵樹——render樹(其實這一步是和上一步同時進行的，為了頁面顯示更迅速，css 不會等到 dom 樹構建完畢才開始構建 render樹)。

Render樹由一些包含有顏色和大小等屬性的矩形組成，它們將被按照正確的順序顯示到螢幕上。

Render樹構建好了之後，將會執行佈局過程，它將確定每個節點在螢幕上的確切座標。

再下一步就是繪製，即遍歷render樹，並使用UI後端層繪製每個節點。

補充知識：

1.瀏覽器會解析三個東西：

（1） HTML/SVG/XHTML，解析這三種檔案會產生一個 DOM Tree。

（2） CSS，解析 CSS 會產生 CSS 規則樹(CSSOM)。

（3） Javascript指令碼，主要是通過 DOM API 和 CSSOM API 來操作 DOM Tree 和 CSS Rule Tree.

2.形象的HTML頁面載入和解析流程:

1. 使用者輸入網址（假設是個html頁面，並且是第一次訪問），瀏覽器向伺服器發出請求，伺服器返回html檔案
2. 瀏覽器開始載入html程式碼，發現 < head > 標籤內有一個 < link > 標籤引用外部CSS檔案；
3. 瀏覽器又發出CSS檔案的請求，伺服器返回這個CSS檔案；

4. 瀏覽器繼續載入html中 < body > 部分的程式碼，並且CSS檔案已經拿到手了，可以開始渲染頁面了；
5. 瀏覽器在程式碼中發現一個 < img > 標籤引用了一張圖片，向伺服器發出請求。此時瀏覽器不會等到圖片下載完，而是繼續渲染後面的程式碼；
6. 伺服器返回圖片檔案，由於圖片佔用了一定面積，影響了後面段落的排布，因此瀏覽器需要回過頭來重新渲染這部分程式碼；
7. 瀏覽器發現了一個包含一行Javascript程式碼的 < script > 標籤，趕快執行它；
8. Javascript指令碼執行了這條語句，它命令瀏覽器隱藏掉程式碼中的某個 < div > （style.display=”none”）。突然少了這麼一個元素，瀏覽器不得不重新渲染這部分程式碼；
9. 終於等到了 < /html > 的到來，瀏覽器淚流滿面……
10. 等等，還沒完，使用者點了一下介面中的“換膚”按鈕，Javascript讓瀏覽器換了一下 < link > 標籤的CSS路徑
11. 瀏覽器召集了在座的各位 < div > < span > < ul > < li > 們，“大夥兒收拾收拾行李，咱得重新來過……”，瀏覽器向伺服器請求了新的CSS檔案，重新渲染頁面。

3.Javascript的載入和執行的特點：

(1)載入後馬上執行；

(2)執行時會阻塞頁面後續的內容（包括頁面的渲染、其它資源的下載）。原因：因為瀏覽器需要一個穩定的DOM樹結構，而JS中很有可能有程式碼直接改變了DOM樹結構，比如使用 document.write 或appendChild,甚至是直接使用的location.href進行跳轉，瀏覽器為了防止出現JS修改DOM樹，需要重新構建DOM樹的情況，所以就會阻塞其他的下載和呈現。

(3)思考：

瞭解了瀏覽器渲染引擎的基本原理，我們可以發現：

當瀏覽器渲染引擎完成了dom樹以及render樹的構建之後，樹中就已經包含了我們在瀏覽器視窗中可以看到的所有資料。

那麼我們就有了一種爬取動態網頁的 **新思路：**

在瀏覽器渲染引擎執行layout以及printing之前，得到dom樹或者render樹，從樹中獲取動態載入的資料。

2.渲染動態網頁：

(1)有兩種選擇：

1.自己從頭實現一個瀏覽器渲染引擎，在合適的時機返回構建的dom樹或render樹：這需要進行大量的工作，需要考慮html、js、css等不同格式檔案的解析方式以及解析順序等。

2.接下來將使用WebKit 渲染引擎，通過 PySide (/jump/aHR0cDovL3B5c2lkZS5naXRodWIuaW8vZG9jcy9weXNpZGUv) 這個python庫可以獲得該引擎的一個便捷介面。

由於相當於第一種方法來說，第二種方法稍微簡單一些，於是這裡以第二種為例

(2)示例：

還是以 新浪讀書——書摘 (/jump/aHR0cDovL2Jvb2suc2luYS5jb20uY24vZXhjZXJwdC8=) 為例，可以發現：頁面中文章列表的部分是動態載入的。

使用PySide庫進行處理的示例程式碼如下：

```
#coding=utf-8

from PySide.QtGui import *
from PySide.QtCore import *
from PySide.QtWebKit import *


if __name__ == '__main__':

    url = "http://book.sina.com.cn/excerpt/rwws/"

    app = QApplication([])  # 完成其他Qt物件之前，必須先建立該物件
    webview = QWebView()  # 該物件是Web 物件的容器

    # 呼叫show方法顯示視窗
    # webview.show()

    # 設定迴圈事件， 並等待網頁載入完成
    loop = QEventLoop()
    webview.loadFinished.connect(loop.quit)
    webview.load(QUrl(url))
    loop.exec_()

    frame = webview.page().mainFrame()  # QWebFrame類有很多與網頁互動的有用方法

    # 得到頁面渲染後的html程式碼
    html = frame.toHtml()

    print html
```

通過print語句，我們可以發現：頁面的原始碼html中已經包含了動態載入的內容。

與網站互動：

得到動態載入的內容後，需要解決的另一個問題是翻頁問題。還好PySide庫的QWebKit模組還有一個名為QWebFrame的類，支援很多與網頁的互動操作。

如“點選”：

```
#根據CSS Selector 找到所需“進行翻頁”的元素

elem = frame.findFirstElement('#subShowContent1_loadMore')

# 點選：通過evaluateJavaScript()函式可以執行Js程式碼

elem.evaluateJavaScript('this.click()')
```

除了點選事件，還可以進行填充表單，滾動視窗等操作

需要注意的是，在進行了翻頁、或者獲取更多內容時，一個最大的難點在於如何確定頁面是否完成了載入，因為我們難以估計Ajax事件或者Js準備資料的時間。

對於這個問題有兩種解決思路：

(1)等待固定的一段時間，比如time.sleep(3)：這種方法容易實現，但效率較低。

(2)輪詢網頁，等待特定內容出現：這種方法雖然會在檢查是否載入完成時浪費CPU週期，但更加可靠。

以下是一個簡單的實現：

```
elem = None
while not elem:
 app.processEvents()
 elem = frame.findAllElemnets('#pattern')
```

程式碼迴圈，直到出現特定元素。每次迴圈，呼叫app.processEvents()方法，用於給Qt事件迴圈執行任務的時間，比如響應點選事件。

但是PySide畢竟是一個為了Python的GUI 程式設計而開發的， 其功能對於爬蟲來說實在是太過於龐大，所以我們可以把爬蟲經常使用的功能進行封裝，來提升編寫爬蟲的效率。

(3)對PySide 常用功能的封裝 —— ghost.py

ghost.py (/jump/aHR0cHM6Ly9naG9zdC1weS5yZWFkdGhlZG9jcy5pby9lbi9sYXRlc3QvIw==) 是目前一個針對爬蟲且功能比較完善的PySide的封裝模組，使用它可以很方便的進行資料採集。

還是以獲取列表頁中每篇文章詳情頁地址為目標，

1.示例程式碼：

```
# coding=utf-8

import re
import time

from ghost import Ghost, Session


class SinaBookSpider(object):

    # 初始化相關引數
    gh = Ghost()
    ss = Session(gh, display=True)  # 設定display為true, 方便除錯

    total = 1526  # 預先計算的總資料量
    count = 0  # 已爬取的資料量

    # 記錄解析以及翻頁位置
    location = 0
    click_times = 0

    def run(self):
        """
        開始爬蟲
        :return:
        """
        # 開啟網頁
        self.ss.open("http://book.sina.com.cn/excerpt/rwws/")
        # 等待資料載入完成
        self.ss.wait_for_selector('#subShowContent1_static > div:nth-child(20)')

        self.parselist()

        while self.count < self.total:
            if self.click_times is 0:
                # 點選載入更多
                self.ss.click('#subShowContent1_loadMore')
                # 每次翻頁，或載入更多，要等待至載入完成
                self.ss.wait_for_selector('#subShowContent1_static > div:nth-child(21)')

                self.click_times += 1
                self.parselist()
            elif self.click_times is 1:
                self.ss.click('#subShowContent1_loadMore')
                self.ss.wait_for_selector('#subShowContent1_static > div:nth-child(41)')

                self.click_times += 1
                self.parselist()
            elif self.click_times is 2:
                self.ss.click('#subShowContent1_page .pagebox_next a')
                self.ss.sleep(2)

                self.click_times = 0
                self.location = 0
                self.parselist()
```

```
    def parselist(self):
        """
        解析列表頁
        :return:
        """
        html = self.ss.content.encode('utf8')
        # print html

        pattern = re.compile(r'<div class="item"><h4><a href="(.*?)" target="_blank">', re
        links = pattern.findall(html)

        for i in range(self.location, len(links)):
            print links[i]
            self.count += 1
            self.location += 1
        print self.count


if __name__ == '__main__':
    spider = SinaBookSpider()
    spider.run()
```

2.程式碼地址：

https://github.com/linbo-lin/dynamic-web-process (/jump/aHR0cHM6Ly9naXRodWIuY29tL2xpbmJvLWxpbi9keW5hbWljLXdlYi1wcm9jZXNz)

3.補充：

ghost.py對直接獲取元素支援的不是很好，但可以藉助BeautifulSoup或正則表示式來解決。

ghost.py支援與網頁的簡單互動，如點選，填充表單等

set_field_value(*args, * kwargs)

fill(*args, * kwargs)

click(*args, * kwargs)

ghost.py很好的解決了確定元素載入完成的問題，通過以下方法可以讓爬蟲等待，直到滿足設定的條件。

wait_for(condition, timeout_message, timeout=None)

wait_for_page_loaded(timeout=None)

wait_for_selector(selector, timeout=None)

wait_for_text(text, timeout=None)

wait_while_selector(selector, timeout=None)

(3)模擬瀏覽器行為法

前面的例子中，我們使用WebKit庫，可以自定義瀏覽器渲染引擎，這樣就可以完全控制想要執行的行為。如果不需要那麼高的靈活性，那麼還有一個不錯的替代品 Selenium (/jump/aHR0cHM6Ly9kb2NzLnNlbGVuaXVtaHEub3JnLw==) 可以選擇，它提供了使瀏覽器自動化的API 介面。

1.Selenium 簡介：

Selenium 是一個用於Web應用程式測試的工具。Selenium測試直接執行在瀏覽器中，就像真正的使用者在操作一樣。支援市面上幾乎所有的主流瀏覽器。

本來打算使用的是selenium + PhantomJS(由於內部 webkit 元件無人維護並且會出現各種各樣的問題，所以作者也已經不再維護)的組合，但發現Chrome以及FireFox也相繼推出無頭 (headless) 瀏覽器模式，個人比較傾向Chrome。本文采用的是Selenium+Chrome的組合。

2.示例：

運用到爬蟲中的思路是：

使用Selenium 渲染網頁，解析渲染後的網頁原始碼，或者直接通過Selenium 介面獲取頁面中的元素。

還是以 新浪讀書——書摘 (/jump/aHR0cDovL2Jvb2suc2luYS5jb20uY24vZXhjZXJwdC8=) 這個網站為例，目標是獲取列表中每篇文章詳情頁的地址

示例程式碼：

```
# coding=utf-8

import time

from selenium import webdriver


class SinaBookSpider(object):

    # 建立可見的Chrome瀏覽器， 方便除錯
    driver = webdriver.Chrome()

    # 建立Chrome的無頭瀏覽器
    # opt = webdriver.ChromeOptions()
    # opt.set_headless()
    # driver = webdriver.Chrome(options=opt)

    driver.implicitly_wait(10)

    total = 1526  # 預先計算的總資料量
    count = 0  # 已爬取的資料量

    # 記錄解析以及翻頁位置
    location = 0
    click_times = 0

    def run(self):
        """
        開始爬蟲
        :return:
        """
        # get方式開啟網頁
        self.driver.get("http://book.sina.com.cn/excerpt/rwws/")

        self.parselist()

        while self.count < self.total:
            if self.click_times is 2:

                self.driver.find_element_by_css_selector('#subShowContent1_page > span:nth

                # 等待頁面載入完成
                time.sleep(5)
                self.click_times = 0
                self.location = 0
            else:
                self.driver.find_element_by_css_selector('#subShowContent1_loadMore').clic

                # 等待頁面載入完成
                time.sleep(3)
                self.click_times += 1

            # 分析載入的新內容，從location開始
            self.parselist()
```

```
        self.driver.quit()

    def parselist(self):
        """
        解析列表
        :return:
        """
        divs = self.driver.find_elements_by_class_name("item")

        for i in range(self.location, len(divs)):
            link = divs[i].find_element_by_tag_name('a').get_attribute("href")
            print link

            self.location += 1
            self.count += 1
        print self.count


if __name__ == '__main__':
    spider = SinaBookSpider()
    spider.run()
```

程式碼地址： https://github.com/linbo-lin/dynamic-web-process (/jump/aHR0cHM6Ly9naXRodWIuY29tL2xpbmJvLWxpbi9keW5hbWljLXdlYi1wcm9jZXNz)

如果你想實際執行上述程式碼，請在執行之前確定：安裝了與瀏覽器版本對應的驅動，並正確的新增到了環境變數中。

3.使用selenium時同樣要特別注意的是如何確定 網頁是否載入完成

有三種方式：

(1)強制等待

(2)隱形等待

(3)顯性等待

有關這三種方式的講解可以看這裡： Python selenium —— 一定要會用selenium的等待，三種等待方式解讀 —— 灰藍的部落格 (/jump/aHR0cHM6Ly9odWlsYW5zYW1lLmdpdGh1Yi5pby9odWlsYW5zYW1lLmdpdGh1Yi5pby9hcmNoaXZlcnMvc2xlZXAtaW1wbGljaXRseXdhaXQtd2FpdA==)

(4)總結：

到此，我們介紹了動態頁面處理的一些思路：

1.逆向回溯 :該方法屬於手工方法，不適合自動檢測

2.渲染動態頁面 :使用PySide或ghost.py，但是由於太過久遠已經被時代淘汰了，所以這種方法並不優雅

3.selenium 模擬瀏覽器:這種方法是現代大型爬蟲最常使用的模式

「JavaScript (/plt/JavaScript)」

58個面向 Web 開發人員的JavaScript技巧彙總 (/pl/aECK/zh-tw)

JavaScript 陣列新增 4 個非破壞性方法！ (/pl/aEQg/zh-tw)

JavaScript 的幾種迴圈方式 (/pl/aEJJ/zh-tw)

淺談JS記憶體機制 (/pl/aEcb/zh-tw)

最近兩週出去面試遇到的面試題（前端初級、長更） (/pl/aEfL/zh-tw)

基於 js 實現一個小型編譯器 (/pl/aEac/zh-tw)

JavaScript 基礎系列之陣列（四） (/pl/aEpu/zh-tw)

JavaScript 實現圖資料結構 (/pl/aEpr/zh-tw)

支援中文！秒建 wiki 知識庫的開源專案，構建私人知識網路 (/pl/aE2p/zh-tw)

javaScript 記憶體管理機制 (/pl/aEmq/zh-tw)