

[Python]網絡爬蟲（一）：抓取網頁的含義和 URL 基本構成

分類： 爬蟲 Python 2013-05-13 22:30 2158 人閱讀 評論(0) 收藏 舉報

一、網絡爬蟲的定義

網絡爬蟲，即 Web Spider，是一個很形象的名字。

把互聯網比喻成一個蜘蛛網，那麼 Spider 就是在網上爬來爬去的蜘蛛。

網絡蜘蛛是通過網頁的鏈接地址來尋找網頁的。

從網站某一個頁面（通常是首頁）開始，讀取網頁的內容，找到在網頁中的其它鏈接地址，

然後通過這些鏈接地址尋找下一個網頁，這樣一直循環下去，直到把這個網站所有的網頁都抓取完為止。

如果把整個互聯網當成一個網站，那麼網絡蜘蛛就可以用這個原理把互聯網上所有的網頁都抓取下來。


這樣看來，網絡爬蟲就是一個爬行程序，一個抓取網頁的程序。

網絡爬蟲的基本操作是抓取網頁。

那麼如何才能隨心所欲地獲得自己想要的頁面？

我們先從 URL 開始。

二、瀏覽網頁的過程

抓取網頁的過程其實和讀者平時使用 IE  瀏覽器瀏覽網頁的道理是一樣的。

比如說你在瀏覽器的地址欄中輸入 `www..baidu.com` 這個地址。

打開網頁的過程其實就是瀏覽器作為一個瀏覽的「客戶端」，向服務器端發送了一次請求，把服務器端的文件「抓」到本地，再進行解釋、展現。

HTML 是一種標記語言，用標籤標記內容並加以解析和區分。



瀏覽器的功能是將獲取到的 HTML 代碼進行解析，然後將原始的代碼轉變成我們直接看到的網站頁面。

三、URI 的概念和舉例

簡單的來講，URL 就是在瀏覽器端輸入的 `www.baidu.com` 這個字符串。

在理解 URL 之前，首先要理解 URI 的概念。

什麼是 URI？

Web 上每種可用的資源，如 HTML 文檔、 圖像、視頻片段、程序等都由一個通用資源標誌符(Universal Resource Identifier，URI)進行定位。

URI 通常由三部分組成：

①訪問資源的命名機制；



- ②存放資源的主機名；
- ③資源自身的名稱，由路徑表示。

如下面的 URI：

<http://www.why.com.cn/myhtml/html1223/>

我們可以這樣解釋它：

- ①這是一個可以通過 HTTP 協議訪問的資源，
- ②位於主機 www.webmonkey.com.cn 上，
- ③通過路徑「/html/html40」訪問。

四、URL 的理解和舉例

URL 是 URI 的一個子集。它是 Uniform Resource Locator 的縮寫，譯為「統一資源定位符」。

通俗地說，URL 是 Internet 上描述信息資源的字符串，主要用在各種 WWW 客戶程序和服務器程序上。

採用 URL 可以用一種統一的格式來描述各種信息資源，包括文件、服務器的地址和目錄等。

URL 的格式由三部分組成：

- ①第一部分是協議(或稱為服務方式)。
- ②第二部分是存有該資源的主機 IP 地址(有時也包括端口號)。
- ③第三部分是主機資源的具體地址，如目錄和文件名等。

第一部分和第二部分用「://」符號隔開，

第二部分和第三部分用「/」符號隔開。

第一部分和第二部分是不可缺少的，第三部分有時可以省略。

下面來看看兩個 URL 的小例子。

1.HTTP 協議的 URL 示例：

使用超級文本傳輸協議 HTTP，提供超級文本信息服務的資源。

例：<http://www.peopledaily.com.cn/channel/welcome.htm>

其計算機域名為 www.peopledaily.com.cn。

超級文本文件(文件類型為.html)是在目錄 /channel 下的 welcome.htm。

這是中國人民日報的一台計算機。

例：<http://www.rol.cn.net/talk/talk1.htm>

其計算機域名為 www.rol.cn.net。

超級文本文件(文件類型為.html)是在目錄/talk 下的 talk1.htm。

這是瑞得聊天室的地址，可由此進入瑞得聊天室的第 1 室。

2. 文件的 URL

用 URL 表示文件時，服務器方式用 file 表示，後面要有主機 IP 地址、文件的存取路徑(即目錄)和文件名等信息。

有時可以省略目錄和文件名，但「/」符號不能省略。

例：file://ftp.yoyodyne.com/pub/files/foobar.txt

上面這個 URL 代表存放在主機 ftp.yoyodyne.com 上的 pub/files/目錄下的一個文件，文件名是 foobar.txt。

例：file://ftp.yoyodyne.com/pub

代表主機 ftp.yoyodyne.com 上的目錄/pub。

例：file://ftp.yoyodyne.com/

代表主機 ftp.yoyodyne.com 的根目錄。

爬蟲最主要的處理對象就是 URL，它根據 URL 地址取得所需要的文件內容，然後對它進行進一步的處理。

因此，準確地理解 URL 對理解網絡爬蟲至關重要。

[Python]網絡爬蟲（二）：利用 urllib2 通過指定的 URL 抓取網頁內容

分類：爬蟲 Python 2013-05-13 23:45 2298 人閱讀 評論(0) 收藏 舉報

所謂網頁抓取，就是把 URL 地址中指定的網絡資源從網絡流中讀取出來，保存到本地。

類似於使用程序模擬 IE 瀏覽器的功能，把 URL 作為 HTTP 請求的內容發送到服務器端，然後讀取服務器端的響應資源。

在 Python 中，我們使用 urllib2 這個組件來抓取網頁。

urllib2 是 Python 的一個獲取 URLs(Uniform Resource Locators)的組件。

它以 urlopen 函數的形式提供了一個非常簡單的接口。

最簡單的 urllib2 的應用代碼只需要四行。

我們新建一個文件 urllib2_test01.py 來感受一下 urllib2 的作用：

```
1. import urllib2
2. response = urllib2.urlopen('http://www.baidu.com/')
3. html = response.read()
4. print html
```

按下 F5 可以看到運行的結果：



我們可以打開百度主頁，右擊，選擇查看源代碼（火狐 OR 谷歌瀏覽器均可），會發現也是完全一樣的內容。

也就是說，上面這四行代碼將我們訪問百度時瀏覽器收到的代碼們全部打印了出來。

這就是一個最簡單的 urllib2 的例子。

除了"http:"，URL 同樣可以使用"ftp:"，"file:"等等來替代。

HTTP 是基於請求和應答機制的：

客戶端提出請求，服務端提供應答。

urllib2 用一個 Request 對象來映射你提出的 HTTP 請求。

在它最簡單的使用形式中你將用你要請求的地址創建一個 Request 對象，通過調用 urlopen 並傳入 Request 對象，將返回一個相關請求 response 對象，這個應答對象如同一個文件對象，所以你可以在 Response 中調用.read()。

我們新建一個文件 urllib2_test02.py 來感受一下：

```
1. import urllib2
2. req = urllib2.Request('http://www.baidu.com')
3. response = urllib2.urlopen(req)
4. the_page = response.read()
5. print the_page
```

可以看到輸出的內容和 test01 是一樣的。

urllib2 使用相同的接口處理所有的 URL 頭。例如你可以像下面那樣創建一個 ftp 請求。

```
1. req = urllib2.Request('ftp://example.com/')
```

在 HTTP 請求時，允許你做額外的兩件事。

1.發送 data 表單數據

這個內容相信做過 Web 端的都不會陌生，

有時候你希望發送一些數據到 URL(通常 URL 與 CGI[通用網關接口]腳本，或其他 WEB 應用程序掛接)。

在 HTTP 中,這個經常使用熟知的 POST 請求發送。

這個通常在你提交一個 HTML 表單時由你的瀏覽器來做。

並不是所有的 POSTs 都來源於表單，你能夠使用 POST 提交任意的數據到你自己的程序。

一般的 HTML 表單，data 需要編碼成標準形式。然後做為 data 參數傳到 Request 對象。

編碼工作使用 urllib 的函數而非 urllib2。

我們新建一個文件 urllib2_test03.py 來感受一下：

```
1. import urllib
2. import urllib2
3.
4. url = 'http://www.someserver.com/register.cgi'
5.
6. values = {'name': 'WHY',
7.           'location': 'SDU',
8.           'language': 'Python' }
9.
10. data = urllib.urlencode(values) # 編碼工作
11. req = urllib2.Request(url, data) # 發送請求同時傳 data 表單
12. response = urllib2.urlopen(req) # 接受反饋的信息
13. the_page = response.read() # 讀取反饋的內容
```

如果沒有傳送 data 參數，urllib2 使用 GET 方式的請求。

GET 和 POST 請求的不同之處是 POST 請求通常有"副作用"，它們會由於某種途徑改變系統狀態(例如提交成堆垃圾到你的門口)。

Data 同樣可以通過在 Get 請求的 URL 本身上面編碼來傳送。

```
1. import urllib2
2. import urllib
3.
4. data = {}
5.
6. data['name'] = 'WHY'
7. data['location'] = 'SDU'
8. data['language'] = 'Python'
9.
10. url_values = urllib.urlencode(data)
11. print url_values
12.
13. name=Somebody+Here&language=Python&location=Northampton
```

```
14. url = 'http://www.example.com/example.cgi'
15. full_url = url + '?' + url_values
16.
17. data = urllib2.open(full_url)
```

這樣就實現了 Data 數據的 Get 傳送。

2.設置 Headers 到 http 請求

有一些站點不喜歡被程序（非人為訪問）訪問，或者發送不同版本的內容到不同的瀏覽器。

默認的 urllib2 把自己作為「Python-urllib/x.y」（x 和 y 是 Python 主版本和次版本號,例如 Python-urllib/2.7),

這個身份可能會讓站點迷惑，或者乾脆不工作。

瀏覽器確認自己身份是通過 User-Agent 頭，當你創建了一個請求對象，你可以給他一個包含頭數據的字典。

下面的例子發送跟上面一樣的內容，但把自身模擬成 Internet Explorer。

```
1. import urllib
2. import urllib2
3.
4. url = 'http://www.someserver.com/cgi-bin/register.cgi'
5.
6. user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
7. values = {'name': 'WHY',
8.           'location': 'SDU',
9.           'language': 'Python' }
10.
11. headers = { 'User-Agent': user_agent }
12. data = urllib.urlencode(values)
13. req = urllib2.Request(url, data, headers)
14. response = urllib2.urlopen(req)
15. the_page = response.read()
```

分享到：

[Python]網絡爬蟲（三）：異常的處理和 HTTP 狀態碼的分類

分類： [Python 爬蟲](#) 2013-05-14 09:51 1770 人閱讀 [評論\(5\)](#) [收藏](#) [舉報](#)

先來說一說 HTTP 的異常處理問題。

當 `urlopen` 不能夠處理一個 `response` 時，產生 `urlError`。
不過通常的 Python APIs 異常如 `ValueError`, `TypeError` 等也會同時產生。
`HTTPError` 是 `urlError` 的子類，通常在特定 HTTP URLs 中產生。

1. URLError

通常，`URLError` 在沒有網絡連接(沒有路由到特定服務器)，或者服務器不存在的情況下產生。

這種情況下，異常同樣會帶有 `"reason"` 屬性，它是一個 `tuple` (可以理解為不可變的數組)，

包含了一個錯誤號和一個錯誤信息。

我們建一個 `urllib2_test06.py` 來感受一下異常的處理：

```
1. import urllib2
2.
3. req = urllib2.Request('http://www.baibai.com')
4.
5. try: urllib2.urlopen(req)
6.
7. except urllib2.URLError, e:
8.     print e.reason
```

按下 F5，可以看到打印出來的內容是：

[Errno 11001] getaddrinfo failed

也就是說，錯誤號是 11001，內容是 `getaddrinfo failed`

2. HTTPError

服務器上每一個 HTTP 應答對象 `response` 包含一個數字"狀態碼"。

有時狀態碼指出服務器無法完成請求。默認的處理器會為你處理一部分這種應答。
例如:假如 `response` 是一個"重定向"，需要客戶端從別的地址獲取文檔，`urllib2` 將為你處理。

其他不能處理的，`urlopen` 會產生一個 `HTTPError`。

典型的錯誤包含 `"404"`(頁面無法找到)，`"403"`(請求禁止)，和 `"401"`(帶驗證請求)。

HTTP 狀態碼表示 HTTP 協議所返回的響應的狀態。

比如客戶端向服務器發送請求，如果成功地獲得請求的資源，則返回的狀態碼為 200，表示響應成功。

如果請求的資源不存在，則通常返回 404 錯誤。

HTTP 狀態碼通常分為 5 種類型，分別以 1 ~ 5 五個數字開頭，由 3 位整數組成：

200：請求成功 處理方式：獲得響應的內容，進行處理

201：請求完成，結果是創建了新資源。新創建資源的 URI 可在響應的實體中得到
處理方式：爬蟲中不會遇到

202：請求被接受，但處理尚未完成 處理方式：阻塞等待

204：服務器端已經實現了請求，但是沒有返回新的信息。如果客戶是用戶代理，則無須為此更新自身的文檔視圖。 處理方式：丟棄

300：該狀態碼不被 HTTP/1.0 的應用程序直接使用，只是作為 3XX 類型回應的默認解釋。存在多個可用的被請求資源。 處理方式：若程序中能夠處理，則進行進一步處理，如果程序中不能處理，則丟棄

301：請求到的資源都會分配一個永久的 URL，這樣就可以在將來通過該 URL 來訪問此資源 處理方式：重定向到分配的 URL

302：請求到的資源在一個不同的 URL 處臨時保存 處理方式：重定向到臨時的 URL

304 請求的資源未更新 處理方式：丟棄

400 非法請求 處理方式：丟棄

401 未授權 處理方式：丟棄

403 禁止 處理方式：丟棄

404 沒有找到 處理方式：丟棄

5XX 回應代碼以「5」開頭的狀態碼表示服務器端發現自己出現錯誤，不能繼續執行請求 處理方式：丟棄

HTTPError 實例產生後會有一個整型'code'屬性，是服務器發送的相關錯誤號。

Error Codes 錯誤碼

因為默認的處理器處理了重定向(300 以外號碼)，並且 100-299 範圍的號碼指示成功，所以你只能看到 400-599 的錯誤號碼。

BaseHTTPServer.BaseHTTPRequestHandler.response 是一個很有用的應答號碼字典，顯示了 HTTP 協議使用的所有的應答號。

當一個錯誤號產生後，服務器返回一個 HTTP 錯誤號，和一個錯誤頁面。

你可以使用 HTTPError 實例作為頁面返回的應答對象 response。

這表示和錯誤屬性一樣，它同樣包含了 read, geturl, 和 info 方法。

我們建一個 urllib2_test07.py 來感受一下：

```
1. import urllib2
```



```
2. req = urllib2.Request('http://bbs.csdn.net/callmewhy')
3.
4. try:
5.     urllib2.urlopen(req)
6.
7. except urllib2.URLError, e:
8.
9.     print e.code
10.    #print e.read()
```

按下 F5 可以看見輸出了 404 的錯誤碼，也就說沒有找到這個頁面。

3.Wrapping

所以如果你想為 HTTPError 或 URLError 做準備，將有兩個基本的辦法。推薦使用第二種。

我們建一個 urllib2_test08.py 來示範一下第一種異常處理的方案：

```
1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
3. req = Request('http://bbs.csdn.net/callmewhy')
4.
5. try:
6.
7.     response = urlopen(req)
8.
9. except HTTPError, e:
10.
11.     print 'The server couldn\'t fulfill the request.'
12.
13.     print 'Error code: ', e.code
14.
15. except URLError, e:
16.
17.     print 'We failed to reach a server.'
18.
19.     print 'Reason: ', e.reason
20.
21. else:
22.     print 'No exception was raised.'
23.     # everything is fine
```

和其他語言相似，try 之後捕獲異常並且將其內容打印出來。

這裡要注意的一點，except HTTPError 必須在第一個，否則 except URLError 將同樣接受到 HTTPError。

因為 HTTPError 是 URLError 的子類，如果 URLError 在前面它會捕捉到所有的 URLError（包括 HTTPError）。

我們建一個 urllib2_test09.py 來示範一下第二種異常處理的方案：

```
1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
3. req = Request('http://bbs.csdn.net/callmewhy')
4.
5. try:
6.
7.     response = urlopen(req)
8.
9. except URLError, e:
10.
11.     if hasattr(e, 'reason'):
12.
13.         print 'We failed to reach a server.'
14.
15.         print 'Reason: ', e.reason
16.
17.     elif hasattr(e, 'code'):
18.
19.         print 'The server couldn\'t fulfill the request.'
20.
21.         print 'Error code: ', e.code
22.
23. else:
24.     print 'No exception was raised.'
25.     # everything is fine
```

[Python]網絡爬蟲（四）：Opener 與 Handler 的介紹和實例應用

分類： Python 爬蟲 2013-05-14 15:09 1433 人閱讀 評論(0) 收藏 舉報

在開始後面的內容之前，先來解釋一下 urllib2 中的兩個個方法：info and geturl

urlopen 返回的應答對象 response(或者 HTTPError 實例)有兩個很有用的方法 info()和 geturl()

1.geturl():

這個返回獲取的真實的 URL，這個很有用，因為 urlopen(或者 opener 對象使用的)或許會有重定向。獲取的 URL 或許跟請求 URL 不同。

以人人中的一個超級鏈接為例，

我們建一個 urllib2_test10.py 來比較一下原始 URL 和重定向的鏈接：

```
1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
3.
4. old_url = 'http://rrurl.cn/b1UZuP'
5. req = Request(old_url)
6. response = urlopen(req)
7. print 'Old url : ' + old_url
8. print 'Real url : ' + response.geturl()
```

運行之後可以看到真正的鏈接指向的網址：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

2.info():

這個返回對象的字典對象，該字典描述了獲取的頁面情況。通常是服務器發送的特定頭 headers。目前是 httpplib.HTTPMessage 實例。

經典的 headers 包含"Content-length", "Content-type", 和其他內容。

我們建一個 urllib2_test11.py 來測試一下 info 的應用：

```
1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
3. old_url = 'http://www.baidu.com'
4. req = Request(old_url)
5. response = urlopen(req)
6. print 'Info():'
7. print response.info()
```

運行的結果如下，可以看到頁面的相關信息：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

下面來說一說 urllib2 中的兩個重要概念：Openers 和 Handlers。

1.Openers：

當你獲取一個 URL 你使用一個 opener(一個 urllib2.OpenerDirector 的實例)。

正常情況下，我們使用默認 opener：通過 urlopen。

但你能夠創建個性的 openers。

2.Handles：

Openers 使用處理器 handlers，所有的「繁重」工作由 handlers 處理。

每個 handlers 知道如何通過特定協議打開 URLs，或者如何處理 URL 打開時的各個方面。

例如 HTTP 重定向或者 HTTP cookies。

如果你希望用特定處理器獲取 URLs 你會想創建一個 openers，例如獲取一個能處理 cookie 的 opener，或者獲取一個不重定向的 opener。

要創建一個 opener，可以實例化一個 OpenerDirector，然後調用.add_handler(some_handler_instance)。

同樣，可以使用 build_opener，這是一個更加方便的函數，用來創建 opener 對象，他只需要一次函數調用。

build_opener 默認添加幾個處理器，但提供快捷的方法來添加或更新默認處理器。其他的處理器 handlers 你或許會希望處理代理，驗證，和其他常用但有點特殊的情況。

install_opener 用來創建（全局）默認 opener。這個表示調用 urlopen 將使用你安裝的 opener。

Opener 對象有一個 open 方法。

該方法可以像 urlopen 函數那樣直接用來獲取 urls：通常不必調用 install_opener，除了為了方便。

說完了上面兩個內容，下面我們來看一下基本認證的內容，這裡會用到上面提及的 Opener 和 Handler。

Basic Authentication 基本驗證

為了展示創建和安裝一個 handler，我們將使用 HTTPBasicAuthHandler。

當需要基礎驗證時，服務器發送一個 header(401 錯誤碼) 請求驗證。這個指定了 scheme 和一個 'realm'，看起來像這樣：Www-authenticate: SCHEME realm="REALM".

例如

Www-authenticate: Basic realm="cPanel Users"

客戶端必須使用新的請求，並在請求頭裡包含正確的姓名和密碼。

這是「基礎驗證」，為了簡化這個過程，我們可以創建一個 HTTPBasicAuthHandler 的實例，並讓 opener 使用這個 handler 就可以啦。

HTTPBasicAuthHandler 使用一個密碼管理的對象來處理 URLs 和 realms 來映射用戶名和密碼。

如果你知道 realm(從服務器發送來的頭裡)是什麼，你就能使用 HTTPPasswordMgr。

通常人們不關心 realm 是什麼。那樣的話，就能用方便的 HTTPPasswordMgrWithDefaultRealm。

這個將在你為 URL 指定一個默認的用戶名和密碼。

這將在你為特定 realm 提供一個其他組合時得到提供。

我們通過給 realm 參數指定 None 提供給 add_password 來指示這種情況。

最高層次的 URL 是第一個要求驗證的 URL。你傳給.add_password()更深層次的 URLs 將同樣合適。

說了這麼多廢話，下面來用一個例子演示一下上面說到的內容。

我們建一個 urllib2_test12.py 來測試一下 info 的應用：

```
1. # -*- coding: utf-8 -*-
2. import urllib2
3.
4. # 創建一個密碼管理者
5. password_mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()
6.
7. # 添加用戶名和密碼
8.
9. top_level_url = "http://example.com/foo/"
10.
```

```
11. # 如果知道 realm, 我們可以使用他代替 ``None``.
12. # password_mgr.add_password(None, top_level_url, username, password)
13. password_mgr.add_password(None, top_level_url, 'why', '1223')
14.
15. # 創建了一個新的 handler
16. handler = urllib2.HTTPBasicAuthHandler(password_mgr)
17.
18. # 創建 "opener" (OpenerDirector 實例)
19. opener = urllib2.build_opener(handler)
20.
21. a_url = 'http://www.baidu.com/'
22.
23. # 使用 opener 獲取一個 URL
24. opener.open(a_url)
25.
26. # 安裝 opener.
27. # 現在所有調用 urllib2.urlopen 將用我們的 opener.
28. urllib2.install_opener(opener)
29.
30.
```

注意：以上的例子我們僅僅提供我們的 HTTPBasicAuthHandler 給 build_opener。

默認的 openers 有正常狀況的 handlers：

ProxyHandler, UnknownHandler, HTTPHandler, HTTPDefaultErrorHandler, HTTPRedirectHandler, FTPHandler, FileHandler, HTTPErrorProcessor。

代碼中的 top_level_url 實際上可以是完整 URL(包含"http:"，以及主機名及可選的端口號)。

例如：http://example.com/。

也可以是一個「authority」(即主機名和可選的包含端口號)。

例如：「example.com」 or 「example.com:8080」。

後者包含了端口號。

[Python]網絡爬蟲（五）：urllib2 的使用細節與抓站技巧

分類： [爬蟲](#) [Python](#) 2013-05-14 16:21 1637 人閱讀 評論(0) [收藏](#) [舉報](#)

前面說到了 urllib2 的簡單入門，下面整理了一部分 urllib2 的使用細節。

1.Proxy 的設置

urllib2 默認會使用環境變量 http_proxy 來設置 HTTP Proxy。

如果想在程序中明確控制 Proxy 而不受環境變量的影響，可以使用代理。

新建 test14 來實現一個簡單的代理 Demo：

```
1. import urllib2
2. enable_proxy = True
3. proxy_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})
4. null_proxy_handler = urllib2.ProxyHandler({})
5. if enable_proxy:
6.     opener = urllib2.build_opener(proxy_handler)
7. else:
8.     opener = urllib2.build_opener(null_proxy_handler)
9. urllib2.install_opener(opener)
```

這裡要注意的一個細節，使用 urllib2.install_opener() 會設置 urllib2 的全局 opener。

這樣後面的使用會很方便，但不能做更細緻的控制，比如想在程序中使用兩個不同的 Proxy 設置等。

比較好的做法是不使用 install_opener 去更改全局的設置，而只是直接調用 opener 的 open 方法代替全局的 urlopen 方法。

2.Timeout 設置

在老版 Python 中（Python2.6 前），urllib2 的 API 並沒有暴露 Timeout 的設置，要設置 Timeout 值，只能更改 Socket 的全局 Timeout 值。

```
1. import urllib2
2. import socket
3. socket.setdefaulttimeout(10) # 10 秒鐘後超時
4. urllib2.socket.setdefaulttimeout(10) # 另一種方式
```

在 Python 2.6 以後，超時可以通過 urllib2.urlopen() 的 timeout 參數直接設置。

```
1. import urllib2
```

```
2. response = urllib2.urlopen('http://www.google.com', timeout=10)
```

3.在 HTTP Request 中加入特定的 Header

要加入 header，需要使用 Request 對象：

```
1. import urllib2
2. request = urllib2.Request('http://www.baidu.com/')
3. request.add_header('User-Agent', 'fake-client')
4. response = urllib2.urlopen(request)
5. print response.read()
```

對有些 header 要特別留意，服務器會針對這些 header 做檢查

User-Agent：有些服務器或 Proxy 會通過該值來判斷是否是瀏覽器發出的請求

Content-Type：在使用 REST 接口時，服務器會檢查該值，用來確定 HTTP Body 中的內容該怎樣解析。常見的取值有：

application/xml：在 XML RPC，如 RESTful/SOAP 調用時使用

application/json：在 JSON RPC 調用時使用

application/x-www-form-urlencoded：瀏覽器提交 Web 表單時使用

在使用服務器提供的 RESTful 或 SOAP 服務時，Content-Type 設置錯誤會導致服務器拒絕服務

4.Redirect

urllib2 默認情況下會針對 HTTP 3XX 返回碼自動進行 redirect 動作，無需人工配置。要檢測是否發生了 redirect 動作，只要檢查一下 Response 的 URL 和 Request 的 URL 是否一致就可以了。

```
1. import urllib2
2. my_url = 'http://www.google.cn'
3. response = urllib2.urlopen(my_url)
4. redirected = response.geturl() == my_url
5. print redirected
6.
7. my_url = 'http://rrurl.cn/b1UZuP'
8. response = urllib2.urlopen(my_url)
9. redirected = response.geturl() == my_url
```


10. `print` redirected

如果不想自動 redirect，除了使用更低層次的 `httplib` 庫之外，還可以自定義 `HTTPRedirectHandler` 類。

```
1. import urllib2
2. class RedirectHandler(urllib2.HTTPRedirectHandler):
3.     def http_error_301(self, req, fp, code, msg, headers):
4.         print "301"
5.         pass
6.     def http_error_302(self, req, fp, code, msg, headers):
7.         print "303"
8.         pass
9.
10. opener = urllib2.build_opener(RedirectHandler)
11. opener.open('http://rrurl.cn/b1UZuP')
```

5.Cookie

`urllib2` 對 Cookie 的處理也是自動的。如果需要得到某個 Cookie 項的值，可以這麼做：

```
1. import urllib2
2. import cookielib
3. cookie = cookielib.CookieJar()
4. opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
5. response = opener.open('http://www.baidu.com')
6. for item in cookie:
7.     print 'Name = '+item.name
8.     print 'Value = '+item.value
```

運行之後就會輸出訪問百度的 Cookie 值：

```
>>>
Name = 56237020
Value = 6733AF79244920861767F528318B998-FDP1
Name = 5_F9_282D3
Value = 2427_2432_2445_2783_2020
Name = 56237020
Value = 17
```

6.使用 HTTP 的 PUT 和 DELETE 方法

`urllib2` 只支持 HTTP 的 GET 和 POST 方法，如果要使用 HTTP PUT 和 DELETE，

只能使用比較低層的 `httplib` 庫。雖然如此，我們還是能通過下面的方式，使 `urllib2` 能夠發出 `PUT` 或 `DELETE` 的請求：

```
1. import urllib2
2. request = urllib2.Request(uri, data=data)
3. request.get_method = lambda: 'PUT' # or 'DELETE'
4. response = urllib2.urlopen(request)
```

7. 得到 HTTP 的返回碼

對於 `200 OK` 來說，只要使用 `urlopen` 返回的 `response` 對象的 `getcode()` 方法就可以得到 HTTP 的返回碼。但對其它返回碼來說，`urlopen` 會拋出異常。這時候，就要檢查異常對象的 `code` 屬性了：

```
1. import urllib2
2. try:
3.     response = urllib2.urlopen('http://bbs.csdn.net/why')
4. except urllib2.HTTPError, e:
5.     print e.code
```

8. Debug Log

使用 `urllib2` 時，可以通過下面的方法把 `debug Log` 打開，這樣收發包的內容就會在屏幕上打印出來，方便調試，有時可以省去抓包的工作

```
1. import urllib2
2. httpHandler = urllib2.HTTPHandler(debuglevel=1)
3. httpsHandler = urllib2.HTTPSHandler(debuglevel=1)
4. opener = urllib2.build_opener(httpHandler, httpsHandler)
5. urllib2.install_opener(opener)
6. response = urllib2.urlopen('http://www.google.com')
```

這樣就可以看到傳輸的數據包內容了：



9. 表單的處理

登錄必要填表，表單怎麼填？

首先利用工具截取所要填表的內容。

比如我一般用 firefox+httpfox 插件來看看自己到底發送了些什麼包。

以 verycd 為例，先找到自己發的 POST 請求，以及 POST 表單項。

可以看到 verycd 的話需要填 username,password,continueURI,fk,login_submit 這幾項，其中 fk 是隨機生成的（其實不太隨機，看上去像是把 epoch 時間經過簡單的編碼生成的），需要從網頁獲取，也就是說得先訪問一次網頁，用正則表達式等工具截取返回數據中的 fk 項。continueURI 顧名思義可以隨便寫，login_submit 是固定的，這從源碼可以看出。還有 username，password 那就很顯然了：

```
1. #-*- coding: utf-8 -*-
2. import urllib
3. import urllib2
4. postdata=urllib.urlencode({
5.     'username':'汪小光',
6.     'password':'why888',
7.     'continueURI':'http://www.verycd.com/',
8.     'fk':"",
9.     'login_submit':'登錄'
10.})
11. req = urllib2.Request(
12.     url = 'http://secure.verycd.com/signin',
13.     data = postdata
14.)
15. result = urllib2.urlopen(req)
16. print result.read()
```

10.偽裝成瀏覽器訪問

某些網站反感爬蟲的到訪，於是對爬蟲一律拒絕請求

這時候我們需要偽裝成瀏覽器，這可以通過修改 http 包中的 header 來實現

```
1. #...
2.
3. headers = {
4.     'User-Agent':'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6'
5. }
6. req = urllib2.Request(
7.     url = 'http://secure.verycd.com/signin/*http://www.verycd.com/',
8.     data = postdata,
9.     headers = headers
10.)
```

11. #...

11.對付"反盜鏈"

某些站點有所謂的反盜鏈設置，其實說穿了很簡單，就是檢查你發送請求的 header 裡面，referer 站點是不是他自己，所以我們只需要像把 headers 的 referer 改成該網站即可，以 cnbeta 為例：

#...

```
headers = {  
  
    'Referer':'http://www.cnbeta.com/articles'  
  
}
```

#...

headers 是一個 dict 數據結構，你可以放入任何想要的 header，來做一些偽裝。例如，有些網站喜歡讀取 header 中的 X-Forwarded-For 來看看人家的真實 IP，可以直接把 X-Forwarded-For 改了。

[Python]網絡爬蟲（六）：一個簡單的百度貼吧的小爬蟲

分類： [爬蟲](#) [Python](#) 2013-05-14 21:36 1617 人閱讀 [評論\(10\)](#) [收藏](#) [舉報](#)

```
1. #-*- coding: utf-8 -*-  
2. #-----  
3. # 程序：百度貼吧爬蟲  
4. # 版本：0.1  
5. # 作者：why  
6. # 日期：2013-05-14  
7. # 語言：Python 2.7  
8. # 操作：輸入帶分頁的地址，去掉最後面的數字，設置一下起始頁數和終點頁數。  
9. # 功能：下載對應頁碼內的所有頁面並存儲為 html 文件。  
10. #-----  
11.  
12. import string, urllib2  
13.  
14. #定義百度函數  
15. def baidu_tieba(url,begin_page,end_page):  
16.     for i in range(begin_page, end_page+1):  
17.         sName = string.zfill(i,5) + '.html'#自動填充成六位的文件名
```

```

18.     print '正在下載第' + str(i) + '個網頁，並將其存儲為' + sName + '.....'
19.     f = open(sName, 'w+')
20.     m = urllib2.urlopen(url + str(i)).read()
21.     f.write(m)
22.     f.close()
23.
24.
25. #----- 在這裡輸入參數 -----
26.
27. # 這個是山東大學的百度貼吧中某一個帖子的地址
28. #bdurl = 'http://tieba.baidu.com/p/2296017831?pn='
29. #iPostBegin = 1
30. #iPostEnd = 10
31.
32. bdurl = str(raw_input(u'請輸入貼吧的地址，去掉 pn=後面的數字：\n'))
33. begin_page = int(raw_input(u'請輸入開始的頁數：\n'))
34. end_page = int(raw_input(u'請輸入終點的頁數：\n'))
35. #----- 在這裡輸入參數 -----
36.
37.
38. #調用
39. baidu_tieba(bdurl, begin_page, end_page)

```

[Python]網絡爬蟲（七）：Python 中的正則表達式教程

分類： 爬蟲 Python 2013-05-15 13:29 1751 人閱讀 評論(0) 收藏 舉報

[目錄\(?\)](#)[\[+\]](#)

接下來準備用糗百做一個爬蟲的小例子。

但是在這之前，先詳細的整理一下 Python 中的正則表達式的相關內容。

正則表達式在 Python 爬蟲中的作用就像是老師點名時用的花名冊一樣，是必不可少的神兵利器。

一、正則表達式基礎

1.1.概念介紹

正則表達式是用於處理字符串的強大工具，它並不是 Python 的一部分。

其他編程語言中也有正則表達式的概念，區別只在於不同的編程語言實現支持的語法數量不同。

它擁有自己獨特的語法以及一個獨立的處理引擎，在提供了正則表達式的語言裡，正則表達式的語法都是一樣的。

下圖展示了使用正則表達式進行匹配的流程：



正則表達式的大致匹配過程是：

1. 依次拿出表達式和文本中的字符比較，
2. 如果每一個字符都能匹配，則匹配成功；一旦有匹配不成功的字符則匹配失敗。
3. 如果表達式中有量詞或邊界，這個過程會稍微有一些不同。

下圖列出了 Python 支持的正則表達式元字符和語法：



1.2. 數量詞的貪婪模式與非貪婪模式

正則表達式通常用於在文本中查找匹配的字符串。

貪婪模式，總是嘗試匹配儘可能多的字符；

非貪婪模式則相反，總是嘗試匹配儘可能少的字符。

Python 裡數量詞默認是貪婪的。

例如：正則表達式"ab*"如果用於查找"abbbbc"，將找到"abbb"。

而如果使用非貪婪的數量詞"ab*?"，將找到"a"。

1.3. 反斜槓的問題

與大多數編程語言相同，正則表達式裡使用"\\"作為轉義字符，這就可能造成反斜槓困擾。

假如你需要匹配文本中的字符"\\"，那麼使用編程語言表示的正則表達式裡將需要4個反斜槓"\\\\\\":

第一個和第三個用於在編程語言裡將第二個和第四個轉義成反斜槓，

轉換成兩個反斜槓\\"後再在正則表達式裡轉義成一個反斜槓用來匹配反斜槓\\。

這樣顯然是非常麻煩的。

Python 裡的原生字符串很好地解決了這個問題，這個例子中的正則表達式可以使用 r"\\\"表示。

同樣，匹配一個數字的"\\d"可以寫成 r"\\d"。

有了原生字符串，媽媽再也不用擔心我的反斜槓問題~

二、介紹 re 模塊

2.1. Compile

Python 通過 re 模塊提供對正則表達式的支持。

使用 re 的一般步驟是：

Step1：先將正則表達式的字符串形式編譯為 Pattern 實例。

Step2：然後使用 Pattern 實例處理文本並獲得匹配結果（一個 Match 實例）。

Step3：最後使用 Match 實例獲得信息，進行其他的操作。

我們新建一個 re01.py 來試驗一下 re 的應用：

```
1. #-*- coding: utf-8 -*-
2. #一個簡單的 re 實例，匹配字符串中的 hello 字符串
3.
4. #導入 re 模塊
5. import re
6.
7. # 將正則表達式編譯成 Pattern 對象，注意 hello 前面的 r 的意思是「原生字符串」
8. pattern = re.compile(r'hello')
9.
10. # 使用 Pattern 匹配文本，獲得匹配結果，無法匹配時將返回 None
11. match1 = pattern.match('hello world!')
12. match2 = pattern.match('helloo world!')
13. match3 = pattern.match('helllo world!')
14.
15. #如果 match1 匹配成功
16. if match1:
17.     # 使用 Match 獲得分組信息
18.     print match1.group()
19. else:
20.     print 'match1 匹配失敗！'
21.
22.
23. #如果 match2 匹配成功
24. if match2:
25.     # 使用 Match 獲得分組信息
26.     print match2.group()
27. else:
28.     print 'match2 匹配失敗！'
29.
30.
31. #如果 match3 匹配成功
32. if match3:
33.     # 使用 Match 獲得分組信息
```

```
34. print match3.group()
35. else:
36. print 'match3 匹配失敗！'
```

可以看到控制台輸出了匹配的三個結果：



下面來具體看看代碼中的關鍵方法。

★ `re.compile(strPattern[, flag])`:

這個方法是 `Pattern` 類的工廠方法，用於將字符串形式的正則表達式編譯為 `Pattern` 對象。

第二個參數 `flag` 是匹配模式，取值可以使用按位或運算符 `|` 表示同時生效，比如 `re.I | re.M`。

另外，你也可以在 `regex` 字符串中指定模式，

比如 `re.compile('pattern', re.I | re.M)` 與 `re.compile('(?im)pattern')` 是等價的。

可選值有：

- `re.I`(全拼： `IGNORECASE`): 忽略大小寫（括號內是完整寫法，下同）
- `re.M`(全拼： `MULTILINE`): 多行模式，改變 `^` 和 `$` 的行為（參見上圖）
- `re.S`(全拼： `DOTALL`): 點任意匹配模式，改變 `.` 的行為
- `re.L`(全拼： `LOCALE`): 使預定字符類 `\w \W \b \B \s \S` 取決於當前區域設定
- `re.U`(全拼： `UNICODE`): 使預定字符類 `\w \W \b \B \s \S \d \D` 取決於 `unicode` 定義的字符屬性
- `re.X`(全拼： `VERBOSE`): 詳細模式。這個模式下正則表達式可以是多行，忽略空白字符，並可以加入註釋。

以下兩個正則表達式是等價的：

```
1. #-*- coding: utf-8 -*-
2. #兩個等價的 re 匹配, 匹配一個小數
3. import re
4.
5. a = re.compile(r"""\d + # the integral part
6.               \. # the decimal point
```



```

7.         \d * # some fractional digits""", re.X)
8.
9. b = re.compile(r"\d+\.\d*")
10.
11. match11 = a.match('3.1415')
12. match12 = a.match('33')
13. match21 = b.match('3.1415')
14. match22 = b.match('33')
15.
16. if match11:
17.     # 使用 Match 獲得分組信息
18.     print match11.group()
19. else:
20.     print u'match11 不是小數'
21.
22. if match12:
23.     # 使用 Match 獲得分組信息
24.     print match12.group()
25. else:
26.     print u'match12 不是小數'
27.
28. if match21:
29.     # 使用 Match 獲得分組信息
30.     print match21.group()
31. else:
32.     print u'match21 不是小數'
33.
34. if match22:
35.     # 使用 Match 獲得分組信息
36.     print match22.group()
37. else:
38.     print u'match22 不是小數'

```

re 提供了眾多模塊方法用於完成正則表達式的功能。這些方法可以使用 Pattern 實例的相應方法替代，唯一的好處是少寫一行 `re.compile()` 代碼，但同時也無法復用編譯後的 Pattern 對象。這些方法將在 Pattern 類的實例方法部分一起介紹。如一開始的 hello 實例可以簡寫為：

```

1. # -*- coding: utf-8 -*-
2. #一個簡單的 re 實例，匹配字符串中的 hello 字符串
3. import re

```

```
4.
5. m = re.match(r'hello', 'hello world!')
6. print m.group()
```

re 模塊還提供了一個方法 `escape(string)`，用於將 `string` 中的正則表達式元字符如 `*/+/?` 等之前加上轉義符再返回

2.2. Match

Match 對象是一次匹配的結果，包含了很多關於此次匹配的信息，可以使用 Match 提供的可讀屬性或方法來獲取這些信息。

屬性：

1. `string`: 匹配時使用的文本。
2. `re`: 匹配時使用的 Pattern 對象。
3. `pos`: 文本中正則表達式開始搜索的索引。值與 `Pattern.match()` 和 `Pattern.seach()` 方法的同名參數相同。
4. `endpos`: 文本中正則表達式結束搜索的索引。值與 `Pattern.match()` 和 `Pattern.seach()` 方法的同名參數相同。
5. `lastindex`: 最後一個被捕獲的分組在文本中的索引。如果沒有被捕獲的分組，將為 `None`。
6. `lastgroup`: 最後一個被捕獲的分組的別名。如果這個分組沒有別名或者沒有被捕獲的分組，將為 `None`。

方法：

1. `group([group1, ...])`:
獲得一個或多個分組截獲的字符串；指定多個參數時將以元組形式返回。
`group1` 可以使用編號也可以使用別名；編號 0 代表整個匹配的子串；不填寫參數時，返回 `group(0)`；沒有截獲字符串的組返回 `None`；截獲了多次的組返回最後一次截獲的子串。
2. `groups([default])`:
以元組形式返回全部分組截獲的字符串。相當於調用 `group(1,2,...`

last)。default 表示沒有截獲字符串的組以這個值替代，默認為 None。

3. `groupdict([default])`:

返回以有別名的組的別名為鍵、以該組截獲的子串為值的字典，沒有別名的組不包含在內。default 含義同上。

4. `start([group])`:

返回指定的組截獲的子串在 string 中的起始索引（子串第一個字符的索引）。group 默認值為 0。

5. `end([group])`:

返回指定的組截獲的子串在 string 中的結束索引（子串最後一個字符的索引 +1）。group 默認值為 0。

6. `span([group])`:

返回(`start(group)`, `end(group)`)。

7. `expand(template)`:

將匹配到的分組代入 template 中然後返回。template 中可以使用 `\id` 或 `\g<id>`、`\g<name>` 引用分組，但不能使用編號 0。`\id` 與 `\g<id>` 是等價的；但 `\10` 將被認為是第 10 個分組，如果你想表達 `\1` 之後是字符 '0'，只能使用 `\g<1>0`。

下面來用一個 py 實例輸出所有的內容加深理解：

```
1. #-*- coding: utf-8 -*-
2. #一個簡單的 match 實例
3.
4. import re
5. # 匹配如下內容：單詞+空格+單詞+任意字符
6. m = re.match(r'(\w+) (\w+)?P<sign>.*', 'hello world!')
7.
8. print "m.string:", m.string
9. print "m.re:", m.re
10. print "m.pos:", m.pos
11. print "m.endpos:", m.endpos
12. print "m.lastindex:", m.lastindex
13. print "m.lastgroup:", m.lastgroup
14.
15. print "m.group():" , m.group()
16. print "m.group(1,2):" , m.group(1, 2)
```

```

17. print "m.groups():" , m.groups()
18. print "m.groupdict():" , m.groupdict()
19. print "m.start(2):" , m.start(2)
20. print "m.end(2):" , m.end(2)
21. print "m.span(2):" , m.span(2)
22. print r"m.expand(r'\g<2> \g<1>\g<3>'):" , m.expand(r'\2 \1\3')
23.
24. ### output ###
25. # m.string: hello world!
26. # m.re: <_sre.SRE_Pattern object at 0x016E1A38>
27. # m.pos: 0
28. # m.endpos: 12
29. # m.lastindex: 3
30. # m.lastgroup: sign
31. # m.group(1,2): ('hello', 'world')
32. # m.groups(): ('hello', 'world', '!')
33. # m.groupdict(): {'sign': '!'}
34. # m.start(2): 6
35. # m.end(2): 11
36. # m.span(2): (6, 11)
37. # m.expand(r'\2 \1\3'): world hello!

```

2.3. Pattern

Pattern 對象是一個編譯好的正則表達式，通過 Pattern 提供的一系列方法可以對文本進行匹配查找。

Pattern 不能直接實例化，必須使用 `re.compile()` 進行構造，也就是 `re.compile()` 返回的對象。

Pattern 提供了幾個可讀屬性用於獲取表達式的相關信息：

1. `pattern`: 編譯時用的表達式字符串。
2. `flags`: 編譯時用的匹配模式。數字形式。
3. `groups`: 表達式中分組的數量。
4. `groupindex`: 以表達式中有別名的組的別名為鍵、以該組對應的編號為值的字典，沒有別名的組不包含在內。

可以用下面這個例子查看 `pattern` 的屬性：

```

1. # -*- coding: utf-8 -*-
2. #一個簡單的 pattern 實例
3.

```

```

4. import re
5. p = re.compile(r'(\w+) (\w+)?P<sign>.*', re.DOTALL)
6.
7. print "p.pattern:", p.pattern
8. print "p.flags:", p.flags
9. print "p.groups:", p.groups
10. print "p.groupindex:", p.groupindex
11.
12. ### output ###
13. # p.pattern: (\w+) (\w+)?P<sign>.*
14. # p.flags: 16
15. # p.groups: 3
16. # p.groupindex: {'sign': 3}

```

下面重點介紹一下 pattern 的實例方法及其使用。

1.match

`match(string[, pos[, endpos]]) | re.match(pattern, string[, flags]):`

這個方法將從 string 的 pos 下標處起嘗試匹配 pattern;

如果 pattern 結束時仍可匹配，則返回一個 Match 對象;

如果匹配過程中 pattern 無法匹配，或者匹配未結束就已到達 endpos，則返回 None。

pos 和 endpos 的默認值分別為 0 和 len(string);

re.match()無法指定這兩個參數，參數 flags 用於編譯 pattern 時指定匹配模式。

注意：這個方法並不是完全匹配。

當 pattern 結束時若 string 還有剩餘字符，仍然視為成功。

想要完全匹配，可以在表達式末尾加上邊界匹配符'\$'。

下面來看一個 Match 的簡單案例：

```

1. # encoding: UTF-8
2. import re
3.
4. # 將正則表達式編譯成 Pattern 對象
5. pattern = re.compile(r'hello')
6.
7. # 使用 Pattern 匹配文本，獲得匹配結果，無法匹配時將返回 None
8. match = pattern.match('hello world!')
9.
10. if match:
11.     # 使用 Match 獲得分組信息
12.     print match.group()
13.
14. ### 輸出 ###

```

2.search

`search(string[, pos[, endpos]])` | `re.search(pattern, string[, flags])`:

這個方法用於查找字符串中可以匹配成功的子串。

從 `string` 的 `pos` 下標處起嘗試匹配 `pattern`,

如果 `pattern` 結束時仍可匹配, 則返回一個 `Match` 對象;

若無法匹配, 則將 `pos` 加 1 後重新嘗試匹配;

直到 `pos=endpos` 時仍無法匹配則返回 `None`。

`pos` 和 `endpos` 的默認值分別為 0 和 `len(string)`;

`re.search()`無法指定這兩個參數, 參數 `flags` 用於編譯 `pattern` 時指定匹配模式。

那麼它和 `match` 有什麼區別呢?

`match()`函數只檢測 `re` 是不是在 `string` 的開始位置匹配,

`search()`會掃描整個 `string` 查找匹配,

`match ()` 只有在 0 位置匹配成功的話才有返回, 如果不是開始位置匹配成功的話, `match()`就返回 `none`

例如:

```
print(re.match( 'super', 'superstition').span())
```

會返回(0, 5)

```
print(re.match( 'super', 'insuperable'))
```

則返回 `None`

`search()`會掃描整個字符串並返回第一個成功的匹配

例如:

```
print(re.search( 'super', 'superstition').span())
```

返回(0, 5)

```
print(re.search( 'super', 'insuperable').span())
```

返回(2, 7)

看一個 `search` 的實例:

```
1. #-*- coding: utf-8 -*-
2. #一個簡單的 search 實例
3.
4. import re
```

```

5.
6. # 將正則表達式編譯成 Pattern 對象
7. pattern = re.compile(r'world')
8.
9. # 使用 search()查找匹配的子串，不存在能匹配的子串時將返回 None
10. # 這個例子中使用 match()無法成功匹配
11. match = pattern.search('hello world!')
12.
13. if match:
14.     # 使用 Match 獲得分組信息
15.     print match.group()
16.
17. ### 輸出 ###
18. # world

```

3.split

split(string[, maxsplit]) | re.split(pattern, string[, maxsplit]):

按照能夠匹配的子串將 string 分割後返回列表。

maxsplit 用於指定最大分割次數，不指定將全部分割。

```

1. import re
2.
3. p = re.compile(r'\d+')
4. print p.split('one1two2three3four4')
5.
6. ### output ###
7. # ['one', 'two', 'three', 'four', '']

```

4.findall

findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags]):

搜索 string，以列表形式返回全部能匹配的子串。

```

1. import re
2.
3. p = re.compile(r'\d+')
4. print p.findall('one1two2three3four4')
5.
6. ### output ###
7. # ['1', '2', '3', '4']

```

5.finditer

`finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags]):`

搜索 string，返回一個順序訪問每一個匹配結果（Match 對象）的迭代器。

```
1. import re
2.
3. p = re.compile(r'\d+')
4. for m in p.finditer('one1two2three3four4'):
5.     print m.group(),
6.
7. ### output ###
8. # 1 2 3 4
```

6.sub

`sub(repl, string[, count]) | re.sub(pattern, repl, string[, count]):`

使用 repl 替換 string 中每一個匹配的子串後返回替換後的字符串。

當 repl 是一個字符串時，可以使用 `\id` 或 `\g<id>`、`\g<name>` 引用分組，但不能使用編號 0。

當 repl 是一個方法時，這個方法應當只接受一個參數（Match 對象），並返回一個字符串用於替換（返回的字符串中不能再引用分組）。

count 用於指定最多替換次數，不指定時全部替換。

```
1. import re
2.
3. p = re.compile(r'(\w+) (\w+)')
4. s = 'i say, hello world!'
5.
6. print p.sub(r'\2 \1', s)
7.
8. def func(m):
9.     return m.group(1).title() + ' ' + m.group(2).title()
10.
11. print p.sub(func, s)
12.
13. ### output ###
14. # say i, world hello!
15. # I Say, Hello World!
```

7.subn

`subn(repl, string[, count]) | re.subn(pattern, repl, string[, count]):`

返回 (sub(repl, string[, count]), 替換次數)。

```
1. import re
2.
3. p = re.compile(r'(\w+) (\w+)')
4. s = 'i say, hello world!'
5.
6. print p.subn(r'\2 \1', s)
7.
8. def func(m):
9.     return m.group(1).title() + ' ' + m.group(2).title()
10.
11. print p.subn(func, s)
12.
13. ### output ###
14. # ('say i, world hello!', 2)
15. # ('I Say, Hello World!', 2)
```

至此，Python 的正則表達式基本介紹就算是完成了^_^

[Python]網絡爬蟲（八）：糗事百科的網絡爬蟲（v0.2）源碼及解析

分類：Python 爬蟲 2013-05-15 20:59 1728 人閱讀 評論(16) 收藏 舉報

項目內容：

用 Python 寫的糗事百科的網絡爬蟲。

使用方法：

新建一個 Bug.py 文件，然後將代碼複製到裡面後，雙擊運行。

程序功能：

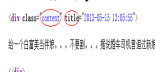
在命令提示行中瀏覽糗事百科。

原理解釋：

首先，先瀏覽一下糗事百科的主頁：<http://www.qiushibaike.com/hot/page/1>

可以看出來，鏈接中 page/後面的數字就是對應的頁碼，記住這一點為以後的編寫做準備。

然後，右擊查看頁面源碼：

A small screenshot showing a snippet of HTML source code. It highlights a `div class="content" title="...` structure. Below the code, there is a small note in Chinese: "從一個任意元素開始，...，不限制...，獲取該元素的所有子元素。"

觀察發現，每一個段子都用 div 標記，其中 class 必為 content，title 是發帖時間，我們只需要用正則表達式將其「扣」出來就可以了。

明白了原理之後，剩下的就是正則表達式的內容了，可以參照這篇博文：

<http://blog.csdn.net/wxg694175346/article/details/8929576>

運行效果：



```
C:\Python27\python.exe

-----
程序：糗百爬虫
版本：0.1
作者：why
日期：2013-05-15
语言：Python 2.7
操作：輸入quit退出閱讀糗事百科
功能：按下回車依次瀏覽今日的糗百熱點
-----

請按下回車瀏覽今日的糗百內容：

正在加載中請稍候.....
第1頁 2013-05-15 11:17:37
老師上課前的惆悵

第1頁 2013-05-15 11:42:40
高中真實故事、割了妹紙的咪咪……一天然呆妹紙，自習課睡得正香、班主任進教室發現她正睡得香、過去推她、沒反應、使勁搖、扭頭繼續睡、班主任又推、該妹紙伸出食指“一分鐘、就一分鐘、再睡一分鐘……”全班爆笑、妹紙惊醒“唉媽呀！我還以為我媽呢！”班主任慫到內傷……

第1頁 2013-05-15 16:00:37
我朋友是個遊戲迷，結婚了，小孩都五歲了。有次孩子問他，爸爸，我是從哪里來的？哥们淡淡的到：和你媽組隊刷boos一年多，才爆出你這個神獸！
```

1. # -*- coding: utf-8 -*-
2. #-----
3. # 程序：糗百爬虫
4. # 版本：0.2
5. # 作者：why
6. # 日期：2013-05-15
7. # 語言：Python 2.7
8. # 操作：輸入 quit 退出閱讀糗事百科
9. # 功能：按下回車依次瀏覽今日的糗百熱點
10. # 更新：解決了命令提示行下亂碼的問題
11. #-----
- 12.
13. import urllib2
14. import urllib
15. import re
16. import thread

```

17. import time
18.
19. #----- 處理頁面上的各種標籤 -----
20. class HTML_Tool:
21.     # 用非 貪婪模式 匹配 \t 或者 \n 或者 空格 或者 超鏈接 或者 圖片
22.     BgnCharToNoneRex = re.compile("(\\t|\\n | <a.*?>|<img.*?>)")
23.
24.     # 用非 貪婪模式 匹配 任意<>標籤
25.     EndCharToNoneRex = re.compile("<.*?>")
26.
27.     # 用非 貪婪模式 匹配 任意<p>標籤
28.     BgnPartRex = re.compile("<p.*?>")
29.     CharToNewLineRex = re.compile("<br/>|</p>|<tr>|<div>|</div>")
30.     CharToNextTabRex = re.compile("<td>")
31.
32.     # 將一些 html 的符號實體轉變為原始符號
33.     replaceTab = [("<" ,"<"),(">" ,">"),("&" ,"&"),("&" ,"\""),("&" ," ") ]
34.
35.     def Replace_Char(self,x):
36.         x = self.BgnCharToNoneRex.sub("",x)
37.         x = self.BgnPartRex.sub("\n ",x)
38.         x = self.CharToNewLineRex.sub("\n",x)
39.         x = self.CharToNextTabRex.sub("\t",x)
40.         x = self.EndCharToNoneRex.sub("",x)
41.
42.         for t in self.replaceTab:
43.             x = x.replace(t[0],t[1])
44.         return x
45. #----- 處理頁面上的各種標籤 -----
46.
47.
48. #----- 加載處理糗事百科 -----
49. class HTML_Model:
50.
51.     def __init__(self):
52.         self.page = 1
53.         self.pages = []
54.         self.myTool = HTML_Tool()
55.         self.enable = False
56.
57.     # 將所有的段子都扣出來，添加到列表中並且返回列表
58.     def GetPage(self,page):
59.         myUrl = "http://m.qiushibaike.com/hot/page/" + page

```

```
60. myResponse = urllib2.urlopen(myUrl)
61. myPage = myResponse.read()
62. #encode 的作用是將 unicode 編碼轉換成其他編碼的字符串
63. #decode 的作用是將其他編碼的字符串轉換成 unicode 編碼
64. unicodePage = myPage.decode("utf-8")
65.
66. # 找出所有 class="content"的 div 標記
67. #re.S 是任意匹配模式，也就是.可以匹配換行符
68. myItems = re.findall('<div.*?class="content".*?title="(.*?)">(.*?)</div>',unicodePage,re.S)
69. items = []
70. for item in myItems:
71.     # item 中第一個是 div 的標題，也就是時間
72.     # item 中第二個是 div 的內容，也就是內容
73.     items.append([item[0].replace("\n",""),item[1].replace("\n","")])
74. return items
75.
76. # 用於加載新的段子
77. def LoadPage(self):
78.     # 如果用戶未輸入 quit 則一直運行
79.     while self.enable:
80.         # 如果 pages 數組中的內容小於 2 個
81.         if len(self.pages) < 2:
82.             try:
83.                 # 獲取新的頁面中的段子們
84.                 myPage = self.GetPage(str(self.page))
85.                 self.page += 1
86.                 self.pages.append(myPage)
87.             except:
88.                 print '無法鏈接糗事百科！'
89.             else:
90.                 time.sleep(1)
91.
92. def ShowPage(self,q,page):
93.     for items in q:
94.         print u'第%d 頁' % page , items[0]
95.         print self.myTool.Replace_Char(items[1])
96.         myInput = raw_input()
97.         if myInput == "quit":
98.             self.enable = False
99.         break
100.
101. def Start(self):
```

```

102.     self.enable = True
103.     page = self.page
104.
105.     print u'正在加載中請稍候.....'
106.
107.     # 新建一個線程在後台加載段子並存儲
108.     thread.start_new_thread(self.LoadPage,())
109.
110.     #----- 加載處理糗事百科 -----
111.     while self.enable:
112.         # 如果 self 的 page 數組中存有元素
113.         if self.pages:
114.             nowPage = self.pages[0]
115.             del self.pages[0]
116.             self.ShowPage(nowPage,page)
117.             page += 1
118.
119.
120. #----- 程序的入口處 -----
121. print u"""
122. -----
123. 程序：糗百爬蟲
124. 版本：0.1
125. 作者：why
126. 日期：2013-05-15
127. 語言：Python 2.7
128. 操作：輸入 quit 退出閱讀糗事百科
129. 功能：按下回車依次瀏覽今日的糗百熱點
130. -----
131. """
132.
133.
134. print u'請按下回車瀏覽今日的糗百內容： '
135. raw_input(' ')
136. myModel = HTML_Model()
137. myModel.Start()

```

[Python]網絡爬蟲（九）： 百度貼吧的網絡爬蟲（v0.4）源碼及解析

分類： 爬蟲 Python 2013-05-16 13:48 1808 人閱讀 評論(10) 收藏 舉報

百度貼吧的爬蟲製作和糗百的爬蟲製作原理基本相同，都是通過查看源碼扣出關鍵數據，然後將其存儲到本地 txt 文件。

項目內容：

用 Python 寫的百度貼吧的網絡爬蟲。

使用方法：

新建一個 BugBaidu.py 文件，然後將代碼複製到裡面後，雙擊運行。

程序功能：

將貼吧中樓主發佈的內容打包 txt 存儲到本地。

原理解釋：

首先，先瀏覽一下某一條貼吧，點擊只看樓主並點擊第二頁之後 url 發生了一點變化，變成了：

http://tieba.baidu.com/p/2296712428?see_lz=1&pn=1

可以看出來，see_lz=1 是只看樓主，pn=1 是對應的頁碼，記住這一點為以後的編寫做準備。

這就是我們需要利用的 url。

接下來就是查看頁面源碼。

首先把題目摳出來存儲文件的時候會用到。

可以看到百度使用 gbk 編碼，標題使用 h1 標記：

1. `<h1 class="core_title_txt" title="【原創】時尚首席（關於時尚，名利，事業，愛情，勵志）">【原創】時尚首席（關於時尚，名利，事業，愛情，勵志）</h1>`

同樣，正文部分用 div 和 class 綜合標記，接下來要做的只是用正則表達式來匹配即可。

運行截圖：



生成的 txt 文件：



1. `# -*- coding: utf-8 -*-`
2. `#-----`
3. `# 程序：百度貼吧爬蟲`
4. `# 版本：0.5`
5. `# 作者：why`
6. `# 日期：2013-05-16`
7. `# 語言：Python 2.7`

```

8. # 操作：輸入網址後自動只看樓主並保存到本地文件
9. # 功能：將樓主發佈的內容打包 txt 存儲到本地。
10. #-----
11.
12. import string
13. import urllib2
14. import re
15.
16. #----- 處理頁面上的各種標籤 -----
17. class HTML_Tool:
18.     # 用非貪婪模式匹配 \t 或者 \n 或者 空格 或者 超鏈接 或者 圖片
19.     BgnCharToNoneRex = re.compile("(\\t|\\n| |<a.*?>|<img.*?>)")
20.
21.     # 用非貪婪模式匹配 任意<>標籤
22.     EndCharToNoneRex = re.compile("<.*?>")
23.
24.     # 用非貪婪模式匹配 任意<p>標籤
25.     BgnPartRex = re.compile("<p.*?>")
26.     CharToNewLineRex = re.compile("<br/>|</p>|<tr>|<div>|</div>")
27.     CharToNextTabRex = re.compile("<td>")
28.
29.     # 將一些 html 的符號實體轉變為原始符號
30.     replaceTab = [("<", "<"),(">", ">"),("&", "&"),("&", "&"),("&", "&"),("&", "&")]
31.
32.     def Replace_Char(self,x):
33.         x = self.BgnCharToNoneRex.sub("",x)
34.         x = self.BgnPartRex.sub("\n",x)
35.         x = self.CharToNewLineRex.sub("\n",x)
36.         x = self.CharToNextTabRex.sub("\t",x)
37.         x = self.EndCharToNoneRex.sub("",x)
38.
39.         for t in self.replaceTab:
40.             x = x.replace(t[0],t[1])
41.         return x
42.
43. class Baidu_Spider:
44.     # 申明相關的屬性
45.     def __init__(self,url):
46.         self.myUrl = url + '?see_lz=1'
47.         self.datas = []
48.         self.myTool = HTML_Tool()
49.         print u'已經啟動百度貼吧爬蟲，咔嚓咔嚓'
50.
51.     # 初始化加載頁面並將其轉碼儲存

```

```

52. def baidu_tieba(self):
53.     # 讀取頁面的原始信息並將其從 gbk 轉碼
54.     myPage = urllib2.urlopen(self.myUrl).read().decode("gbk")
55.     # 計算樓主發佈內容一共有多少頁
56.     endPage = self.page_counter(myPage)
57.     # 獲取該帖的標題
58.     title = self.find_title(myPage)
59.     print u'文章名稱: ' + title
60.     # 獲取最終的數據
61.     self.save_data(self.myUrl,title,endPage)
62.
63.     #用來計算一共有多少頁
64.     def page_counter(self,myPage):
65.         # 匹配 "共有<span class="red">12</span>頁" 來獲取一共有多少頁
66.         myMatch = re.search(r'class="red">(\d+?)</span>', myPage, re.S)
67.         if myMatch:
68.             endPage = int(myMatch.group(1))
69.             print u'爬蟲報告：發現樓主共有%d 頁的原創內容' % endPage
70.         else:
71.             endPage = 0
72.             print u'爬蟲報告：無法計算樓主發佈內容有多少頁！'
73.         return endPage
74.
75.     # 用來尋找該帖的標題
76.     def find_title(self,myPage):
77.         # 匹配 <h1 class="core_title_txt" title="">xxxxxxxxxx</h1> 找出標題
78.         myMatch = re.search(r'<h1.*?>(.*)</h1>', myPage, re.S)
79.         title = u'暫無標題'
80.         if myMatch:
81.             title = myMatch.group(1)
82.         else:
83.             print u'爬蟲報告：無法加載文章標題！'
84.         # 文件名不能包含以下字符： \ / : * ? " < > |
85.         title = title.replace("\\","").replace("/","").replace(":","").replace("*","").replace("?","").replace('"','').replace(">","").replace("<","").replace('|','')
86.         return title
87.
88.
89.     # 用來存儲樓主發佈的內容
90.     def save_data(self,url,title,endPage):
91.         # 加載頁面數據到數組中
92.         self.get_data(url,endPage)
93.         # 打開本地文件

```



```

94.     f = open(title+'.txt','w+')
95.     f.writelines(self.datas)
96.     f.close()
97.     print u'爬蟲報告：文件已下載到本地並打包成 txt 文件'
98.     print u'請按任意鍵退出...'
99.     raw_input();
100.
101.     # 獲取頁面源碼並將其存儲到數組中
102.     def get_data(self,url,endPage):
103.         url = url + '&pn='
104.         for i in range(1,endPage+1):
105.             print u'爬蟲報告：爬蟲%d 號正在加載中...' % i
106.             myPage = urllib2.urlopen(url + str(i)).read()
107.             # 將 myPage 中的 html 代碼處理並存儲到 datas 裡面
108.             self.deal_data(myPage.decode('gbk'))
109.
110.
111.     # 將內容從頁面代碼中摳出來
112.     def deal_data(self,myPage):
113.         myItems = re.findall('id="post_content.*?>(.*)</div>',myPage,re.S)
114.         for item in myItems:
115.             data = self.myTool.Replace_Char(item.replace("\n","").encode('gbk'))
116.             self.datas.append(data+"\n")
117.
118.
119.
120.     #----- 程序入口處 -----
121.     print u""""#-----
122.     # 程序：百度貼吧爬蟲
123.     # 版本：0.5
124.     # 作者：why
125.     # 日期：2013-05-16
126.     # 語言：Python 2.7
127.     # 操作：輸入網址後自動只看樓主並保存到本地文件
128.     # 功能：將樓主發佈的內容打包 txt 存儲到本地。
129.     #-----
130.     """
131.
132.     # 以某小說貼吧為例子
133.     bdurl = 'http://tieba.baidu.com/p/2296712428?see_lz=1&pn=1'
134.
135.     print u'請輸入貼吧的地址最後的數字串： '
136.     bdurl = 'http://tieba.baidu.com/p/' + str(raw_input(u'http://tieba.baidu.com/p/'))
137.

```

```
138. #調用
139.mySpider = Baidu_Spider(bdurl)
140.mySpider.baidu_tieba()
```

[Python]網絡爬蟲（十）：一個爬蟲的誕生全過程（以山東大學績點運算為例）

分類： 爬蟲 Python 2013-07-12 13:46 1425 人閱讀 評論(3) 收藏 舉報

先來說一下我們學校的網站：

http://jwxt.sdu.edu.cn:7777/zhxt_bks/zhxt_bks.html

查詢成績需要登錄，然後顯示各學科成績，但是只顯示成績而沒有績點，也就是加權平均分。



學分	學分	學分	學分	學分	學分
1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

顯然這樣手動計算績點是一件非常麻煩的事情。所以我們可以用 python 做一個爬蟲來解決這個問題。

1.決戰前夜

先來準備一下工具：HttpFox 插件。

這是一款 http 協議分析插件，分析頁面請求和響應的時間、內容、以及瀏覽器用到的 COOKIE 等。

以我為例，安裝在火狐上即可，效果如圖：



可以非常直觀的查看相應的信息。

點擊 start 是開始檢測，點擊 stop 暫停檢測，點擊 clear 清除內容。

一般在使用之前，點擊 stop 暫停，然後點擊 clear 清屏，確保看到的是訪問當前頁面獲得的數據。

2.深入敵後

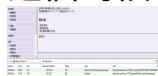
下面就去山東大學的成績查詢網站，看一看在登錄的時候，到底發送了那些信息。

先來到登錄頁面，把 httpfox 打開，clear 之後，點擊 start 開啟檢測：



輸入完了個人信息，確保 httpfox 處於開啟狀態，然後點擊確定提交信息，實現登錄。

這個時候可以看到，httpfox 檢測到了三條信息：



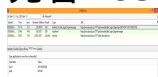
這時點擊 stop 鍵，確保捕獲到的是訪問該頁面之後反饋的數據，以便我們做爬蟲的時候模擬登陸使用。

3.庖丁解牛

乍一看我們拿到了三個數據，兩個是 GET 的一個是 POST 的，但是它們到底是什麼，應該怎麼用，我們還一無所知。

所以，我們需要挨個查看一下捕獲到的內容。

先看 POST 的信息：



既然是 POST 的信息，我們就直接看 PostData 即可。

可以看到一共 POST 兩個數據，stuid 和 pwd。

並且從 Type 的 Redirect to 可以看出，POST 完畢之後跳轉到了 bks_login2.loginmessage 頁面。

由此看出，這個數據是點擊確定之後提交的表單數據。

點擊 cookie 標籤，看看 cookie 信息：



沒錯，收到了一個 ACCOUNT 的 cookie，並且在 session 結束之後自動銷毀。

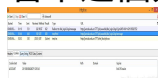
那麼提交之後收到了哪些信息呢？

我們來看看後面的兩個 GET 數據。

先看第一個，我們點擊 content 標籤可以查看收到的內容，是不是有一種生吞活剝的快感-。-HTML 源碼暴露無疑了：



看來這個只是顯示頁面的 html 源碼而已，點擊 cookie，查看 cookie 的相關信息：



啊哈，原來 html 頁面的內容是發送了 cookie 信息之後才接受到的。

再來看看最後一個接收到的信息：



大致看了一下應該只是一個叫做 style.css 的 css 文件，對我們沒有太大的作用。

4.冷靜應戰

既然已經知道了我們向服務器發送了什麼數據，也知道了我們接收到了什麼數據，基本的流程如下：

- 首先，我們 POST 學號和密碼--->然後返回 cookie 的值
- 然後發送 cookie 給服務器--->返回頁面信息。
- 獲取到成績頁面的數據，用正則表達式將成績和學分單獨取出並計算加權平均數。

OK，看上去好像很簡單的樣紙。那下面我們就來試試看吧。

但是在實驗之前，還有一個問題沒有解決，就是 POST 的數據到底發送到了哪裡？再來看一下當初的頁面：



很明顯是用一個 html 框架來實現的，也就是說，我們在地址欄看到的地址並不是右邊提交表單的地址。

那麼怎樣才能獲得真正的地址。-右擊查看頁面源代碼：

嗯沒錯，那個 name="w_right"的就是我們要的登錄頁面。

網站的原來的地址是：

http://jwxt.sdu.edu.cn:7777/zhxt_bks/zhxt_bks.html

所以，真正的表單提交的地址應該是：

http://jwxt.sdu.edu.cn:7777/zhxt_bks/xk_login.html

輸入一看，果不其然：



靠居然是清華大學的選課系統。。。目測是我校懶得做頁面了就直接借了。。結果連標題都不改一下。。。

但是這個頁面依舊不是我們需要的頁面，因為我們的 POST 數據提交到的頁面，應該是表單 form 的 ACTION 中提交到的頁面。

也就是說，我們需要查看源碼，來知道 POST 數據到底發送到了哪裡：



嗯，目測這個才是提交 POST 數據的地址。

整理到地址欄中，完整的地址應該如下：

http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login

（獲取的方式很簡單，在火狐瀏覽器中直接點擊那個鏈接就能看到這個鏈接的地址了）

5.小試牛刀

接下來的任務就是：用 python 模擬發送一個 POST 的數據並取到返回的 cookie 值。

關於 cookie 的操作可以看看這篇博文：

<http://blog.csdn.net/wxg694175346/article/details/8925978>

我們先準備一個 POST 的數據，再準備一個 cookie 的接收，然後寫出源碼如下：

```
1. #-*- coding: utf-8 -*-
2. #-----
3. # 程序：山東大學爬蟲
4. # 版本：0.1
5. # 作者：why
6. # 日期：2013-07-12
7. # 語言：Python 2.7
8. # 操作：輸入學號和密碼
9. # 功能：輸出成績的加權平均值也就是績點
10. #-----
11.
12. import urllib
13. import urllib2
14. import cookielib
15.
16. cookie = cookielib.CookieJar()
17. opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
18.
19. #需要 POST 的數據#
20. postdata=urllib.urlencode({
21.     'stuid':'201100300428',
22.     'pwd':'921030'
23. })
24.
25. #自定義一個請求#
26. req = urllib2.Request(
```

```

27. url = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login',
28. data = postdata
29.)
30.
31. #訪問該鏈接#
32. result = opener.open(req)
33.
34. #打印返回的內容#
35. print result.read()

```

如此這般之後，再看看運行的效果：



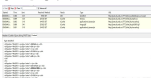
ok，如此這般，我們就算模擬登陸成功了。

6.偷天換日

接下來的任務就是用爬蟲獲取到學生的成績。

再來看看源網站。

開啟 HTTPFOX 之後，點擊查看成績，發現捕獲到了如下的數據：



點擊第一個 GET 的數據，查看內容可以發現 Content 就是獲取到的成績的內容。

而獲取到的頁面鏈接，從頁面源代碼中右擊查看元素，可以看到點擊鏈接之後跳轉的頁面（火狐瀏覽器只需要右擊，「查看此框架」，即可）：



從而可以得到查看成績的鏈接如下：

<http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre>

7.萬事俱備

現在萬事俱備啦，所以只需要把鏈接應用到爬蟲裡面，看看能否查看到成績的頁面。

從 httpfox 可以看到，我們發送了一個 cookie 才能返回成績的信息，所以我們就用 python 模擬一個 cookie 的發送，以此來請求成績的信息：

```

1. #-*- coding: utf-8 -*-
2. #-----

```

```
3. # 程序：山東大學爬蟲
4. # 版本：0.1
5. # 作者：why
6. # 日期：2013-07-12
7. # 語言：Python 2.7
8. # 操作：輸入學號和密碼
9. # 功能：輸出成績的加權平均值也就是績點
10. #-----
11.
12. import urllib
13. import urllib2
14. import cookielib
15.
16. #初始化一個 CookieJar 來處理 Cookie 的信息#
17. cookie = cookielib.CookieJar()
18.
19. #創建一個新的 opener 來使用我們的 CookieJar#
20. opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
21.
22. #需要 POST 的數據#
23. postdata=urllib.urlencode({
24.     'stuid':'201100300428',
25.     'pwd':'921030'
26. })
27.
28. #自定義一個請求#
29. req = urllib2.Request(
30.     url = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login',
31.     data = postdata
32. )
33.
34. #訪問該鏈接#
35. result = opener.open(req)
36.
37. #打印返回的內容#
38. print result.read()
39.
40. #打印 cookie 的值
41. for item in cookie:
42.     print 'Cookie: Name = '+item.name
43.     print 'Cookie: Value = '+item.value
44.
45.
46. #訪問該鏈接#
47. result = opener.open('http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscope')
```

```
48.  
49. #打印返回的內容#  
50. print result.read()
```

按下 F5 運行即可，看看捕獲到的數據吧：



既然這樣就沒有什麼問題了吧，用正則表達式將數據稍稍處理一下，取出學分和相應的分數就可以了。

8.手到擒來

這麼一大堆 html 源碼顯然是不利於我們處理的，下面要用正則表達式來摳出必須的數據。

關於正則表達式的教程可以看看這個博文：

<http://blog.csdn.net/wxg694175346/article/details/8929576>

我們來看看成績的源碼：



既然如此，用正則表達式就易如反掌了。

我們將代碼稍稍整理一下，然後用正則來取出數據：

```
1. #-*- coding: utf-8 -*-  
2. #-----  
3. # 程序：山東大學爬蟲  
4. # 版本：0.1  
5. # 作者：why  
6. # 日期：2013-07-12  
7. # 語言：Python 2.7  
8. # 操作：輸入學號和密碼  
9. # 功能：輸出成績的加權平均值也就是績點  
10. #-----  
11.  
12. import urllib  
13. import urllib2  
14. import cookielib  
15. import re  
16.
```



```

17. class SDU_Spider:
18.     # 申明相關的屬性
19.     def __init__(self):
20.         self.loginUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login' # 登錄的 url
21.         self.resultUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre' # 顯示成績的 url
22.         self.cookieJar = cookielib.CookieJar() # 初始化一個 CookieJar 來處理 Cookie 的信息
23.         self.postdata=urllib.urlencode({'stuid':'201100300428','pwd':'921030'}) # POST 的數據
24.         self.weights = [] #存儲權重，也就是學分
25.         self.points = [] #存儲分數，也就是成績
26.         self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookieJar))
27.
28.     def sdu_init(self):
29.         # 初始化鏈接並且獲取 cookie
30.         myRequest = urllib2.Request(url = self.loginUrl,data = self.postdata) # 自定義一個請求
31.         result = self.opener.open(myRequest) # 訪問登錄頁面，獲取到必須的 cookie 的值
32.         result = self.opener.open(self.resultUrl) # 訪問成績頁面，獲得成績的數據
33.         # 打印返回的內容
34.         # print result.read()
35.         self.deal_data(result.read().decode('gbk'))
36.         self.print_data(self.weights);
37.         self.print_data(self.points);
38.
39.         # 將內容從頁面代碼中摳出來
40.         def deal_data(self,myPage):
41.             myItems = re.findall('<TR>.*?<p.*?<p.*?<p.*?<p.*?<p.*?>(.*)</p>.*?<p.*?<p.*?>(.*)</p>.*?</TR>',myPage,re.S) #獲取到學分
42.             for item in myItems:
43.                 self.weights.append(item[0].encode('gbk'))
44.                 self.points.append(item[1].encode('gbk'))
45.
46.
47.         # 將內容從頁面代碼中摳出來
48.         def print_data(self,items):
49.             for item in items:
50.                 print item
51.
52. #調用
53. mySpider = SDU_Spider()
54. mySpider.sdu_init()

```

水平有限，，正則是有點醜，。運行的效果如圖：



ok, 接下來的只是數據的處理問題了。。

9.凱旋而歸

完整的代碼如下，至此一個完整的爬蟲項目便完工了。

```
1. #-*- coding: utf-8 -*-
2. #-----
3. # 程序：山東大學爬蟲
4. # 版本：0.1
5. # 作者：why
6. # 日期：2013-07-12
7. # 語言：Python 2.7
8. # 操作：輸入學號和密碼
9. # 功能：輸出成績的加權平均值也就是績點
10. #-----
11.
12. import urllib
13. import urllib2
14. import cookielib
15. import re
16. import string
17.
18.
19. class SDU_Spider:
20.     # 申明相關的屬性
21.     def __init__(self):
22.         self.loginUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login' # 登錄的 url
23.         self.resultUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre' # 顯示成績的 url
24.         self.cookiejar = cookielib.CookieJar() # 初始化一個 CookieJar 來處理 Cookie 的信息
25.         self.postdata=urllib.urlencode({'stuid':'201100300428','pwd':'921030'}) # POST 的數據
26.         self.weights = [] #存儲權重，也就是學分
27.         self.points = [] #存儲分數，也就是成績
28.         self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookiejar))
29.
30.     def sdu_init(self):
31.         # 初始化鏈接並且獲取 cookie
32.         myRequest = urllib2.Request(url = self.loginUrl,data = self.postdata) # 自定義一個請求
33.         result = self.opener.open(myRequest) # 訪問登錄頁面，獲取到必須的 cookie 的值
34.         result = self.opener.open(self.resultUrl) # 訪問成績頁面，獲得成績的數據
```

```
35.     # 打印返回的內容
36.     # print result.read()
37.     self.deal_data(result.read().decode('gbk'))
38.     self.calculate_date();
39.
40.     # 將內容從頁面代碼中摳出來
41.     def deal_data(self,myPage):
42.         myItems = re.findall('<TR>.*?<p.*?<p.*?<p.*?<p.*?>(.*?)</p>.*?<p.*?<p.*?>(.*?)</p>.*?</TR>',myPage,re.S)  #獲取到學分
43.         for item in myItems:
44.             self.weights.append(item[0].encode('gbk'))
45.             self.points.append(item[1].encode('gbk'))
46.
47.     #計算績點，如果成績還沒出來，或者成績是優秀良好，就不運算該成績
48.     def calculate_date(self):
49.         point = 0.0
50.         weight = 0.0
51.         for i in range(len(self.points)):
52.             if(self.points[i].isdigit()):
53.                 point += string.atof(self.points[i])*string.atof(self.weights[i])
54.                 weight += string.atof(self.weights[i])
55.             print point/weight
56.
57.
58. #調用
59. mySpider = SDU_Spider()
60. mySpider.sdu_init()
```