

# 表单交互和验证码处理

## 提交表单

Python爬虫入门-表单提交与模拟登录

利用Request库进行POST请求表单交互

cookie实现模拟登录网站

Requests实现POST请求

今requests可以以多种形式进行post请求，比如form表单形式、json形式等。今天主要以表单形式举例：

Regeusts支持以form表单形式发送post请求，只需要将请求的参数构造成一个字典，然后传给requests.post()的data参数即可。

示例网站：豆瓣网：<https://www.douban.com>

有两种方式可以查询提交表单的字段：

通过查询源代码的form标签,input标签

通过浏览器的Network项查询

第一种：

首先我们找到登录的元素，在输入账号处选中->右键->检查

然后直接查询网页源代码去找到上面的部分，根据标签来观察提交的表单参数，这里强调一下：

form标签和form标签下的input标签非常重要，form标签中的action属性代表请求的URL，input标签下的name属性代表提交参数的KEY。

代码参考如下：

```
import requests
url="https://www.douban.com/accounts/login"      #action属性
params={
    "source":"index_nav",                        #input标签下的name
    "form_email":"xxxxxx",                       #input标签下的name
    "form_password":"xxxxxx"                     #input标签下的name
}
html=requests.post(url,data=params)
print(html.text)
```

运行后发现已登录账号

第二种：

通过浏览器Network项查询表单参数：

点击右键->检查->选择Network

然后手动输入账号和密码登录，此时显示加载了文件，选择加载的第一个文件：

选中后，查看Headers字段下的数据，会发现请求的URL

往下拉，会发现字段参数：

然后再按照上面的代码写一下就可以了。

### Cookie模拟登录

Cookie，有时也用其复数形式 Cookies，指某些网站为了辨别用户身份、进行 session 跟踪而储存在用户本地终端上的数据（通常经过加密）。

我们可以通过手动登录后，查看浏览器的Network选项找到cookie值，记住cookie值不要透露出去。

操作步骤：

右键->检查->选择Network->手动登录->在加载文件中找到本网址的Name

得到cookie和URL之后，把cookie添加到headers中，运行代码如下：

```
import requests
url="https://www.douban.com/"
header={"Cookie":'XXXXXXXXXXXXXXXXX'} #cookie值不要泄露
html=requests.get(url,headers=header)
print(html.text)
```

## 手动提交（略，用鼠标键盘敲代码输入）

## 自动提交

### [【Python3爬虫】当爬虫碰到表单提交，有点意思](#)

#### 一、写在前面

我写爬虫已经写了一段时间了，对于那些使用GET请求或者POST请求的网页，爬取的时候都还算得心应手。不过最近遇到了一个有趣的网站，虽然爬取的难度不大，不过因为表单提交的存在，所以一开始还是有点摸不着头脑。至于最后怎么解决的，请慢慢往下看。

#### 二、页面分析

这次爬取的网站是：[https://www.ctic.org/crm?tdsourcetag=s\\_pctim\\_aiomsg](https://www.ctic.org/crm?tdsourcetag=s_pctim_aiomsg)，该网站提供了美国的一些农田管理的数据。要查看具体的数据，需要选择年份、单位、地区、作物种类等，如下图：

\* Year :  ⓘ

\* Units : ☒ Acres ☐ Hectares ☐ Percentage ⓘ


\* Area : ☐ National ☐ Regional ☐ State ☒ County ⓘ

Region :  ▼

State :  ▼

County :  ▼

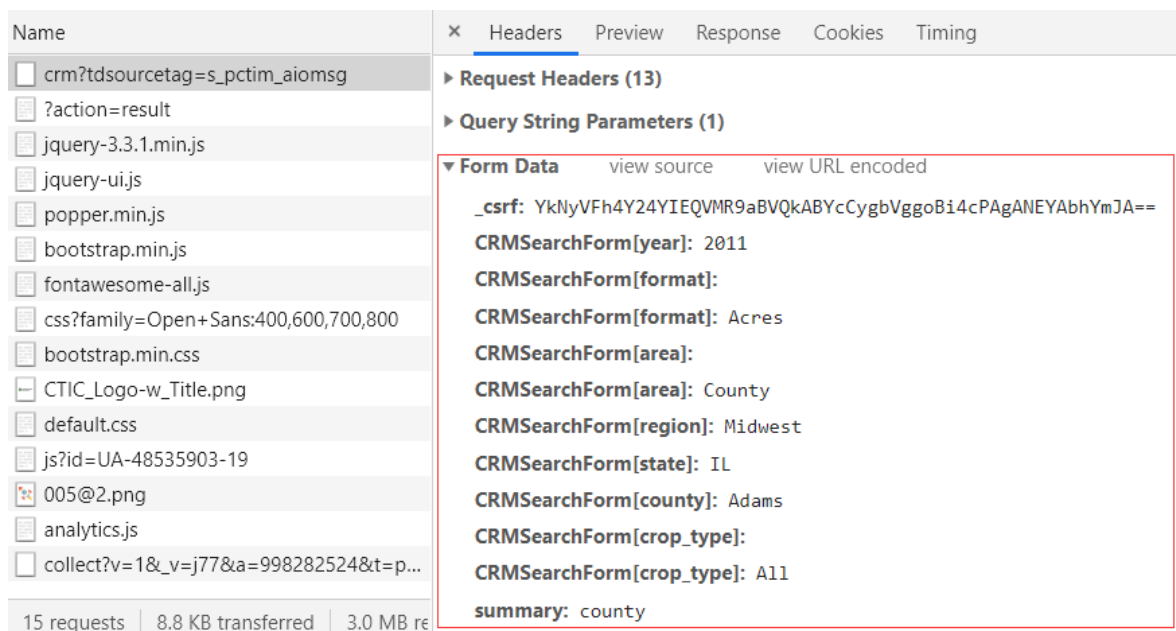
\* Crop type : ☒ All crops ☐ Custom

 View Summary

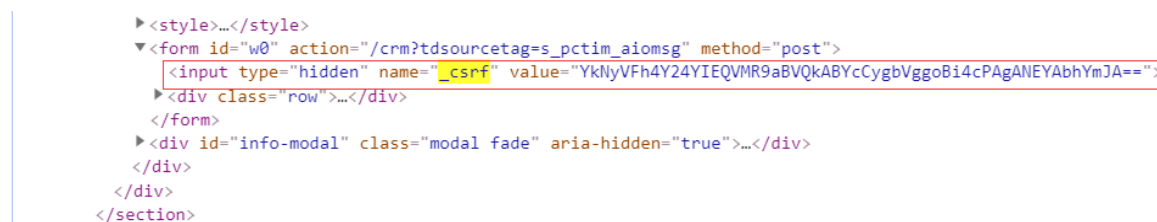
根据以往的经验，这种表单提交都是通过ajax来完成的，所以熟练地按F12打开开发者工具，选择XHR选项，然后点击“View Summary”，结果却什么都没有.....

Filter	<input type="checkbox"/> Hide data URLs	All	XHR	JS	CSS	Img	Media	Font	Doc	WS	Manifest	Other
5000 ms												
10000 ms												
15000 ms												
20000 ms												
25000 ms												
Name	Status		Type									

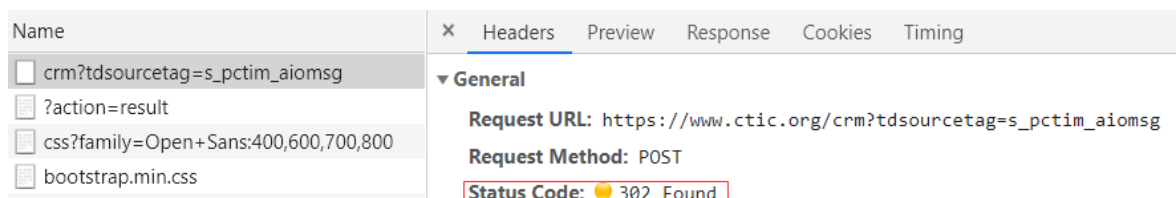
这是怎么回事？不急，切换到All看一下有没有什么可疑的东西。果然就找到了下面这个，可以看到在Form Data中包含了很多参数，而且可以很明显看出来是一些年份、地区等信息，这就是表单提交的内容：



可以注意到在这些参数中有一个csrf，很明显是一个加密参数，那么要怎么得到这个参数呢？返回填写表单的网页，在开发者工具中切换到Elements，然后搜索csrf看看，很快就找到了如下信息：



其余参数都是表单中所选择的内容，只要对应填写就行了。不过这个请求返回的状态码是302，为什么会是302呢？302状态码的使用场景是请求的资源暂时驻留在不同的URI下，因此还需要继续寻找。



通过进一步查找可知，最终的URL是：<https://www.ctic.org/crm/?action=result>。

### 三、主要步骤

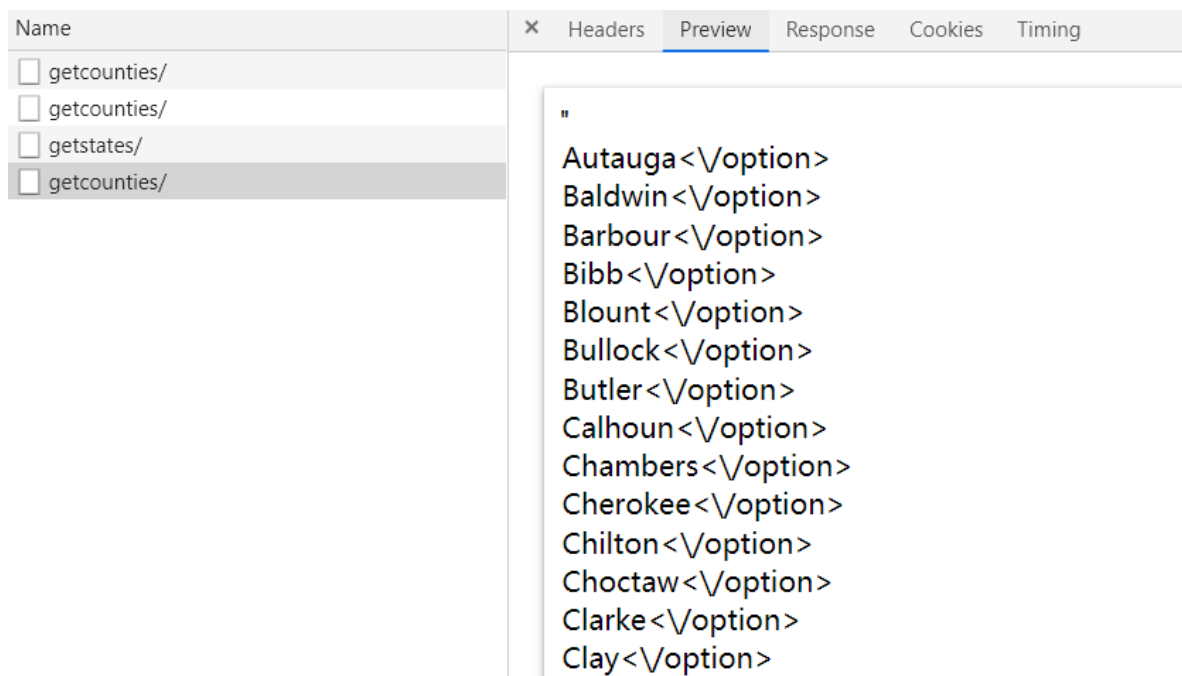
#### 1.爬取郡县信息

可以看到表单中包含了地区、州、郡县选项，在填写表单的时候，这一部分都是通过JS来实现的。打开开发者工具，然后在页面上点选County，选择Region和State，就能在开发者工具中找到相应的请求。主要有两个请求，如下：

<https://www.ctic.org/admin/custom/crm/getstates/>

<https://www.ctic.org/admin/custom/crm/getcounties/>

这两个请求返回的结果格式如下图：



这里可以使用正则匹配，也可以使用lxml来解析，我选择使用后者。示例代码如下：

```

1 from lxml import etree
2
3
4 html = '<option value="Autauga">Autauga</option><option
value="Baldwin">Baldwin</option><option value="Barbour">Barbour</option>
<option value="Bibb">Bibb</option><option value="Blount">Blount</option>
<option value="Bullock">Bullock</option><option
value="Butler">Butler</option><option value="Calhoun">Calhoun</option>
<option value="Chambers">Chambers</option><option
value="Cherokee">Cherokee</option><option
value="Chilton">Chilton</option><option value="Choctaw">Choctaw</option>
<option value="Clarke">Clarke</option><option value="Clay">Clay</option>
<option value="Cleburne">Cleburne</option><option
value="Coffee">Coffee</option><option value="Colbert">Colbert</option>
<option value="Conecuh">Conecuh</option><option
value="Coosa">Coosa</option><option value="Covington">Covington</option>
<option value="Crenshaw">Crenshaw</option><option
value="Cullman">Cullman</option><option value="Dale">Dale</option><option
value="Dallas">Dallas</option><option value="DeKalb">DeKalb</option>
<option value="Elmore">Elmore</option><option
value="Escambia">Escambia</option><option value="Etowah">Etowah</option>
<option value="Fayette">Fayette</option><option
value="Franklin">Franklin</option><option value="Geneva">Geneva</option>
<option value="Greene">Greene</option><option value="Hale">Hale</option>
<option value="Henry">Henry</option><option
value="Houston">Houston</option><option value="Jackson">Jackson</option>
<option value="Jefferson">Jefferson</option><option
value="Lamar">Lamar</option><option
value="Lauderdale">Lauderdale</option><option
value="Lawrence">Lawrence</option><option value="Lee">Lee</option><option
value="Limestone">Limestone</option><option
value="Lowndes">Lowndes</option><option value="Macon">Macon</option>
<option value="Madison">Madison</option><option
value="Marengo">Marengo</option><option value="Marion">Marion</option>
<option value="Marshall">Marshall</option><option
value="Mobile">Mobile</option><option value="Monroe">Monroe</option>
<option value="Montgomery">Montgomery</option><option
value="Morgan">Morgan</option><option value="Perry">Perry</option><option
value="Pickens">Pickens</option><option value="Pike">Pike</option><option
value="Randolph">Randolph</option><option
value="Russell">Russell</option><option value="Shelby">Shelby</option>
<option value="St Clair">St Clair</option><option
value="Sumter">Sumter</option><option
value="Talladega">Talladega</option><option
value="Tallapoosa">Tallapoosa</option><option
value="Tuscaloosa">Tuscaloosa</option><option
value="Walker">Walker</option><option
value="Washington">Washington</option><option
value="Wilcox">Wilcox</option><option value="Winston">Winston</option>'
5 et = etree.HTML(html)
6 result = et.xpath('//option/text()')
7 result = [i.rstrip('') for i in result]
8 print(result)

```

上面代码输出的结果为：

```
['Autauga', 'Baldwin', 'Barbour', 'Bibb', 'Blount', 'Bullock', 'Butler', 'Calhoun', 'Chambers',
'Cherokee', 'Chilton', 'Choctaw', 'Clarke', 'Clay', 'Clebune', 'Coffee', 'Colbert', 'Conecuh',
'Coosa', 'Covington', 'Crenshaw', 'Cullman', 'Dale', 'Dallas', 'DeKalb', 'Elmore', 'Escambia',
'Etowah', 'Fayette', 'Franklin', 'Geneva', 'Greene', 'Hale', 'Henry', 'Houston', 'Jackson',
'Jefferson', 'Lamar', 'Lauderdale', 'Lawrence', 'Lee', 'Limestone', 'Lowndes', 'Macon',
'Madison', 'Marengo', 'Marion', 'Marshall', 'Mobile', 'Monroe', 'Montgomery', 'Morgan',
'Perry', 'Pickens', 'Pike', 'Randolph', 'Russell', 'Shelby', 'St Clair', 'Sumter', 'Talladega',
'Tallapoosa', 'Tuscaloosa', 'Walker', 'Washington', 'Wilcox', 'Winston']
```

获取所有郡县信息的思路为分别选择四个地区，然后遍历每个地区下面的州，再遍历每个州所包含的郡县，最终得到所有郡县信息。

## 2.爬取农田数据

在得到郡县信息之后，就可以构造获取农田数据的请求所需要的参数了。在获取农田数据之前，需要向服务器发送一个提交表单的请求，不然是得不到数据的。在我测试的时候，发送提交表单的请求的时候，返回的状态码并不是302，不过这并不影响之后的操作，所以可以忽略掉。

需要注意的是，参数中是有一个年份信息的，前面我一直是默认用的2011，不过要爬取更多信息的话，还需要改变这个年份信息。而通过选择页面元素可以知道，这个网站提供了16个年份的农田数据信息，这16个年份为：

```
[1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 2002, 2004, 2006, 2007, 2008, 2011]
```

得到这些年份信息之后，就可以和前面的郡县信息进行排列组合得到所有提交表单的请求所需要的参数。说道排列组合，一般会用for循环来实现，不过这里推荐一种方法，就是使用itertools.product，使用示例如下：

```
1 from itertools import product
2
3 a = [1, 2, 3]
4 b = [2, 4]
5 result = product(a, b)
6 for i in result:
7     print(i, end=" ")
8
9
10 # (1, 2) (1, 4) (2, 2) (2, 4) (3, 2) (3, 4)
```

下面是农田数据的部分截图，其中包含了很多种类的作物，还有对应的耕地面积信息，不过在这个表中有些我们不需要的信息，比如耕地面积总量信息，还有空白行，这都是干扰数据，在解析的时候要清洗掉。

Crop	Total Planted (Acres)	Conservation Tillage			Conservation Tillage Total	Other Tillage Practices	
		No-Till	Ridge-Till	Mulch-Till		Reduced-Till (15-30% Residue)	Conventional-Till (0-15% Residue)
Corn	136,900	12200	0	54200	66,400	42000	28500
Soybeans (FS)	123,000	66400	0	45500	111,900	7400	3700
Soybeans (DC)	1,500	1500	0	0	1,500	0	0
Cotton	0	0	0	0	0	0	0
Spring Wheat	0	0	0	0	0	0	0
Winter Wheat	15,300	13500	0	1600	15,100	200	0
Oats	500	250	0	250	500	0	0

解析农田数据部分的代码如下：

```
1 et = etree.HTML(html)
2 crop_list = et.xpath('//*[@id="crm_results_eight"]/tbody/tr/td[1]/text()') # 作物名称
3 area_list = et.xpath('//*[@id="crm_results_eight"]/tbody/tr/td[2]/text()') # 耕地面积
4 conservation_list = et.xpath('//*[@id="crm_results_eight"]/tbody/tr/td[6]/text()') # 受保护耕地面积
5 crop_list = crop_list[:-3]
6 area_list = area_list[:-3]
7 conservation_list = conservation_list[:-3]
```

完整代码已上传到[GitHub](#)！

## 验证码处理

### 1. 输入式验证码

这种验证码主要是通过用户输入图片中的字母、数字、汉字等进行验证。如下图



图1

图2

解决思路：这种是最简单的一种，只要识别出里面的内容，然后填入到输入框中即可。这种识别技术叫OCR，这里我们推荐使用Python的第三方库，tesseract。对于没有什么背景影响的验证码如图2，直接通过这个库来识别就可以。但是对于有嘈杂的背景的验证码这种，直接识别识别率会很低，遇到这种我们就得需要先处理一下图片，先对图片进行灰度化，然后再进行二值化，再去识别，这样识别率会大大提高。

### 2. 滑动式验证码

这种是将备选碎片直线滑动到正确的位置，如下图





解决思路：对于这种验证码就比较复杂一点，但也是有相应的办法。我们直接想到的就是模拟人去拖动验证码的行为，点击按钮，然后看到了缺口 的位置，最后把拼图拖到缺口位置处完成验证。

第一步：点击按钮。然后我们发现，在你没有点击按钮的时候那个缺口和拼图是没有出现的，点击后才出现，这为我们找到缺口的位置提供了灵感。

第二步：拖到缺口位置。我们知道拼图应该拖到缺口处，但是这个距离如果用数值来表示？通过我们第一步观察到的现象，我们可以找到缺口的位置。这里我们可以比较两张图的像素，设置一个基准值，如果某个位置的差值超过了基准值，那我们就找到了这两张图片不一样的位置，当然我们是从那块拼图的右侧开始并且从左到右，找到第一个不一样的位置时就结束，这是的位置应该是缺口的left，所以我们使用selenium拖到这个位置即可。这里还有个疑问就是如何能自动的保存这两张图？这里我们可以先找到这个标签，然后获取它的location和size，然后 top, bottom, left, right = location['y'], location['y']+size['height'], location['x'] + size['width'], 然后截图，最后抠图填入这四个位置就行。具体的使用可以查看selenium文档，点击按钮前抠张图，点击后再抠张图。最后拖动的时候要需要模拟人的行为，先加速然后减速。因为这种验证码有行为特征检测，人是不可能做到一直匀速的，否则它就判定为是机器在拖动，这样就无法通过验证了。

### 3.点击式的图文验证 和 图标选择

图文验证：通过文字提醒用户点击图中相同字的位置进行验证。

图标选择：给出一组图片，按要求点击其中一张或者多张。借用万物识别的难度阻挡机器。

这两种原理相似，只不过一个是给出文字，点击图片中的文字，一个是给出图片，点出内容相同的图片。

这两种没有特别好的方法，只能借助第三方识别接口来识别出相同的内容，推荐一个超级鹰，把验证码发过去，会返回相应的点击坐标。

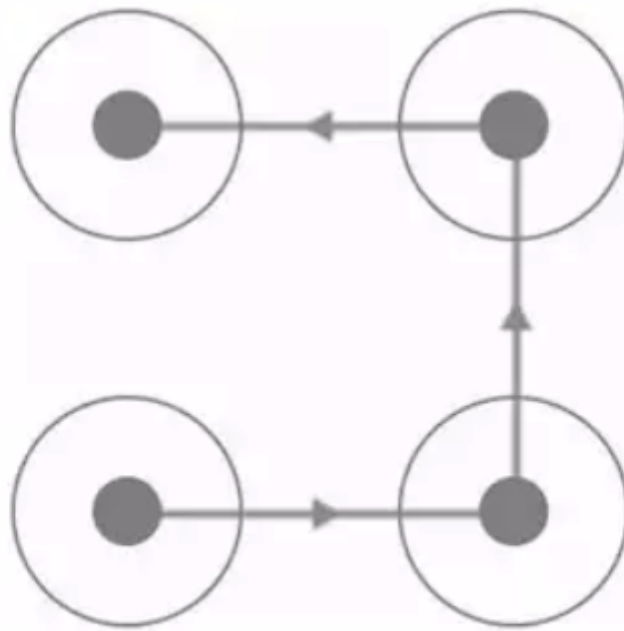
然后再使用selenium模拟点击即可。具体怎么获取图片和上面方法一样。

### 4.宫格验证码



## 安全验证

请按照箭头路线滑动手指



[https://blog.csdn.net/qq\\_2811974](https://blog.csdn.net/qq_2811974)

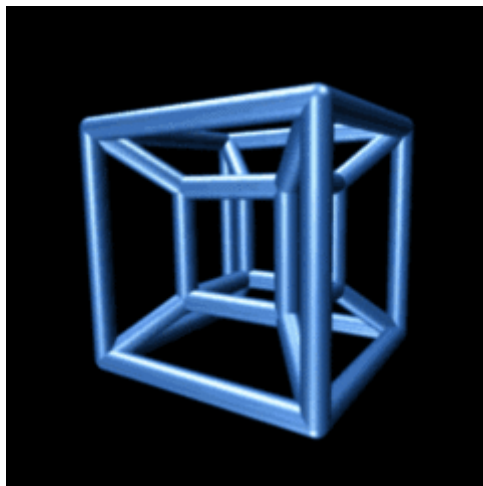
这种就很棘手，每一次出现的都不一样，但是也会出现一样的。而且拖动顺序都不一样。

但是我们发现不一样的验证码个数是有限的，这里采用模版匹配的方法。我觉得就好像暴力枚举，把所有出现的验证码保存下来，然后挑出不一样的验证码，按照拖动顺序命名，我们从左到右上下到下，设为1,2,3,4。上图的滑动顺序为4,3,2,1所以我们命名4\_3\_2\_1.png，这里得手动搞。当验证码出现的时候，用我们保存的图片——枚举，与出现这种比较像素，方法见上面。如果匹配上了，拖动顺序就为4,3,2,1。然后使用selenium模拟即可。

**加载验证码(网站自己加载略)**

### 光学字符识别

光学字符识别（OCR）是从图像中抽取文本的工具，可以应用于公安、电信、物流、金融等诸多行业，例如识别车牌，身份证扫描识别、名片信息提取等。在爬虫开发中，如果遭遇了有文字验证码的表单，就可以利用OCR来进行验证码处理。Tesseract-OCR引擎最初是由惠普公司开发的光学字符识别系统，目前发布在Github上，由Google赞助开发。



## 处理更复杂的验证码

很多网站为了分别出提供验证码的是人还是机器使用了更为复杂的验证码，例如拼图验证码、点触验证码、九宫格验证码等。关于这方面的知识，在崔庆才同学的[《Python 3网络爬虫开发实战》](#)有较为详细的讲解，有兴趣的可以购买阅读。

## 验证码处理服务（略）