Python实现网站模拟登陆

一、实验简介

1.1 基本介绍

本实验中我们将通过分析登陆流程并使用 Python 实现模拟登陆到一个实验提供的网站，在实验过程中将学习并实践 Python 的网络编程，Python 实现模拟登陆的方法，使用 Firefox 抓包分析插件分析网络数据包等知识。

模拟登录可以帮助用户自动化完成很多操作，在不同场合下有不同的用处，无论是自动化一些日常的繁琐操作还是用于爬虫都是一项很实用的技能。本课程通过 Firefox 和 Python 来实现，环境要求如下：

- Python 库：urllib, urllib2, cookielib, Django
- Firefox 要求：装有 live http header插件 (已提供)

1.2 知识点

本项目中我们将学习并实践以下知识点：

1. 网站登录流程分析
2. Python 网络编程基础
3. Firefox 抓包分析插件 Live http header
4. Python 模拟登陆实现流程

1.3 实验材料

为了节省时间，实验用到的材料已经提前制作完成，可以按照材料清单中给出的链接下载。

实验网站源码：
`http://labfile.oss.aliyuncs.com/courses/640/mysite.zip`

FireFox抓包插件：
`http://labfile.oss.aliyuncs.com/courses/640/live_http_headers.xpi`

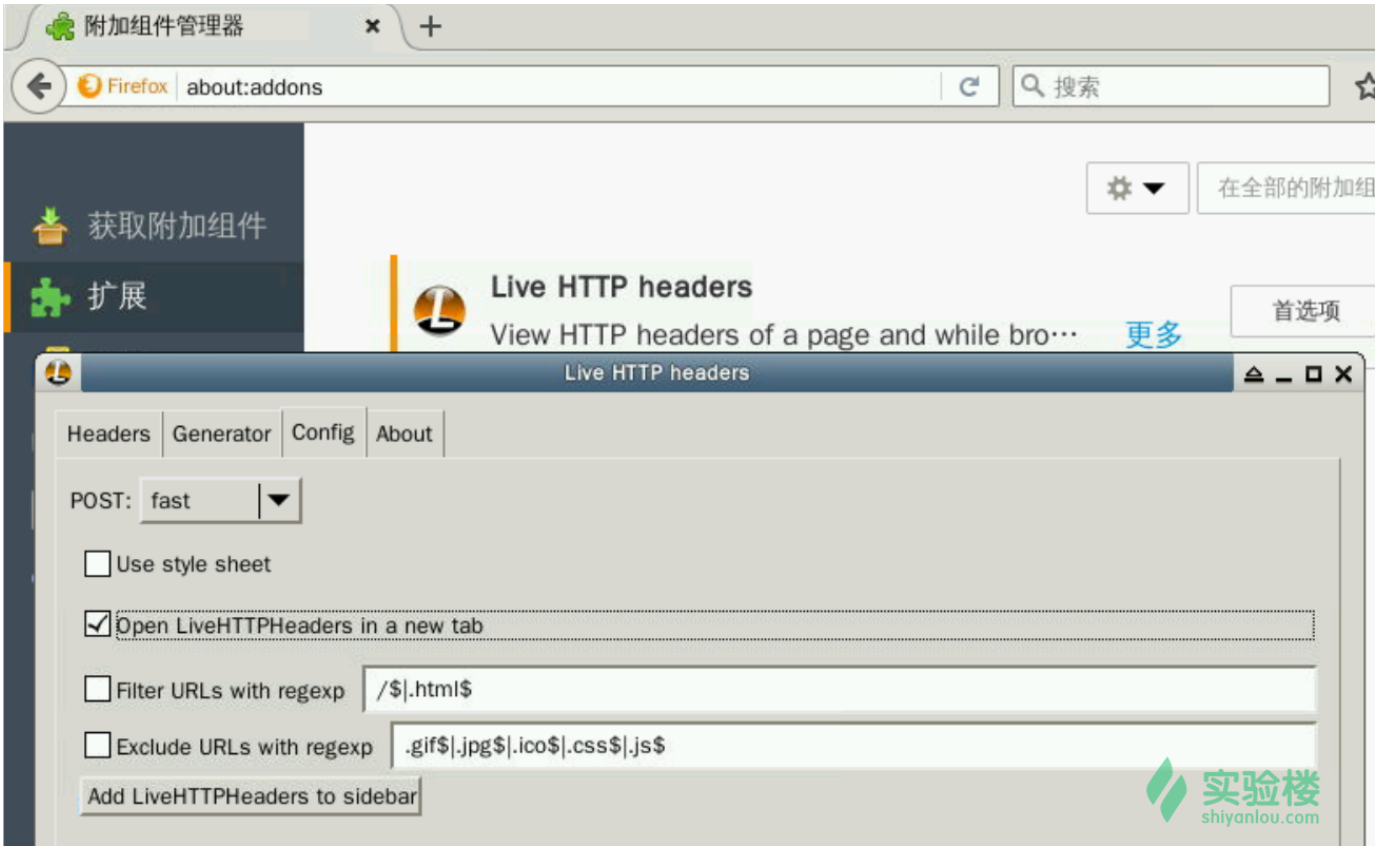
1.4 实验准备

1) 安装抓包插件Live Http Headers

打开 Linux Xfce 终端并通过 `wget` 命令下载插件文件。下载成功后首选右键单击xpi文件-->使用FireFox打开，之后按照界面提示安装Live Http Header插件：

按照提示重启之后，通过 打开菜单-->附加组件-->扩展 找到安装好的插件，点击 首选项 ，勾选 Config 选项卡中的 Open LiveHTTPHeaders in a new tab 选项以方便使用。

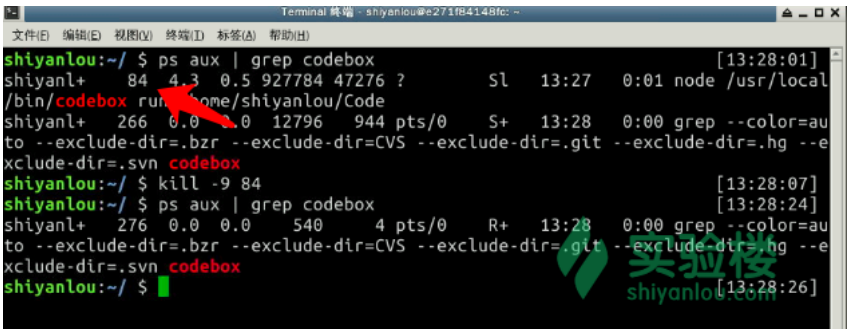
动手实践是学习 IT 技术最有效的方式！ 开始实验



2)启动web应用

由于实验楼会员环境中启动的 WebIDE 会占用 8000 端口，所以如果是实验楼会员，请先停止 codebox 进程后再部署下面的 Web 应用。

使用 `ps -aux | grep codebox` 查询获得 codebox 的进程号，然后使用 `kill -9 进程号` 停止 codebox 进程。执行过程见下图：



首先安装demo依赖的web框架django，并测试是否安装成功：

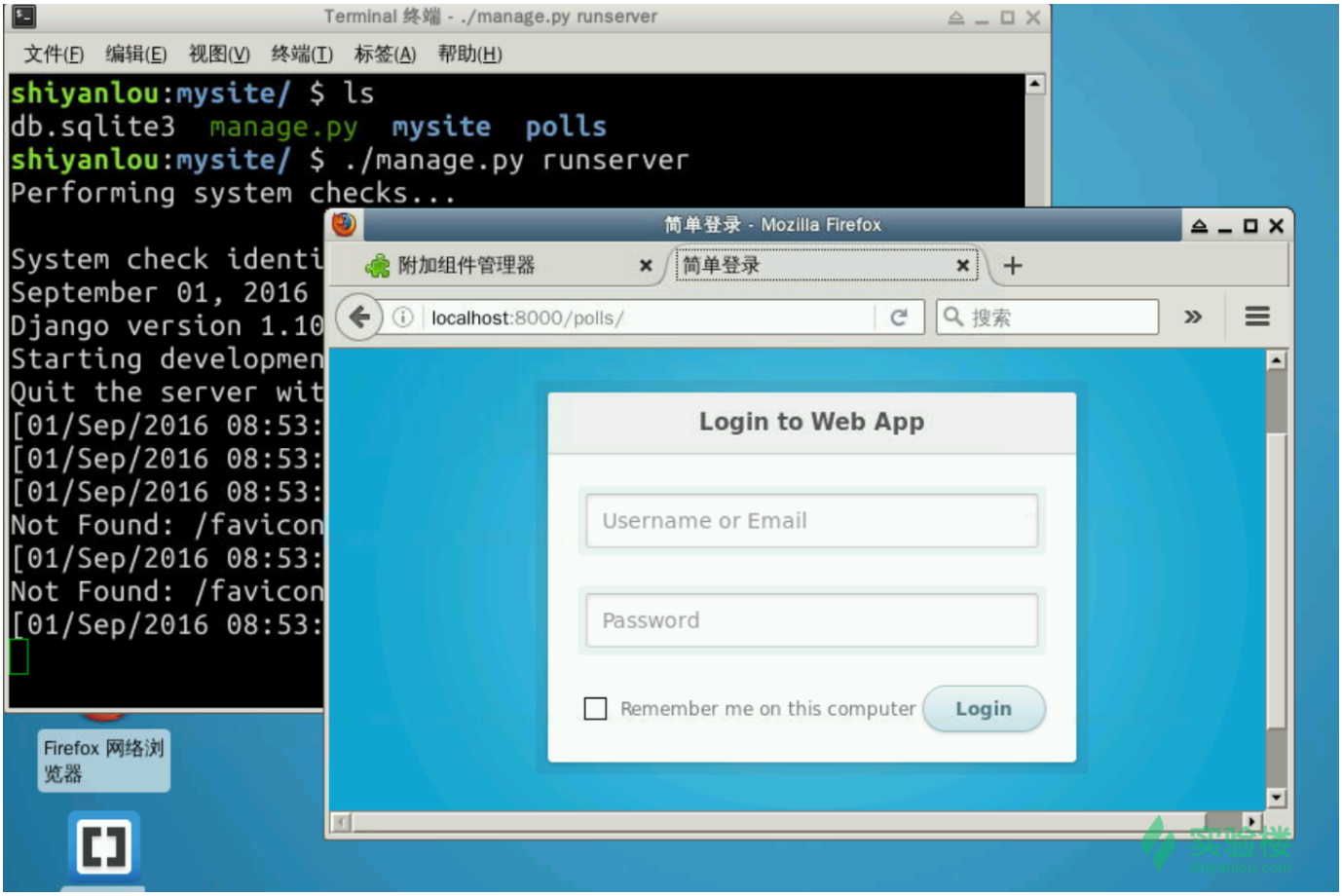
```
$ sudo pip install django
$ python
>>> import django
>>> django.VERSION
(1, 10, 0, u'final', 1)
```

再通过unzip命令解压网站文件，并启动网站服务。

```
$ wget http://labfile.oss.aliyuncs.com/courses/640/mysite.zip
$ unzip mysite.zip
$ cd mysite
$ ./manage.py runserver
```

启动成功后在浏览器中输入 `http://localhost:8000/polls` 看到登录页面表示启动成功。

动手实践是学习 IT 技术最有效的方式！ 开始实验



二、分析登录过程

要通过编程实现登录，首先需要理解一般Web应用的登陆过程。

不同的网站和应用登录的安全性和复杂性都不同，因此它们的登录实现过程自然会存在差异，尽管如此，最基础，核心的过程依然是相同的。对于复杂的登录过程(例如淘宝)，对请求包和响应包的分析能力是很重要的。

浏览器有自带的分析工具，但是界面比较窄，考虑到实验环境界面较小，本课程选择了 Live Http Header，同学们可以在本地选用任何自己喜欢的工具进行分析。我们先通过示例页面分析一下简单的登陆过程熟悉下分析过程。

2.1 抓取请求

1. 输入 `http://localhost:8000/polls` 打开登录页。
2. 打开live http header(`F10->工具->Live Http Header`)。
3. 输入用户名和密码(都是 `shiyanlou`)并提交表单，登入系统。
4. 切换到Live Http Header页面查看http请求和响应信息

2.2原理分析

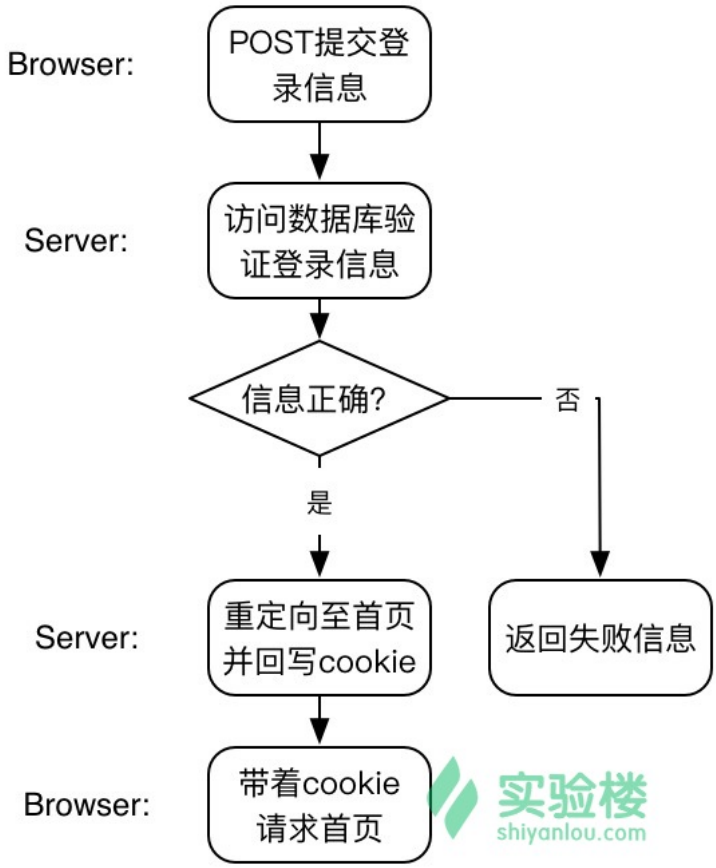
Live http Headers插件的 `Headers` 选项卡中会列出抓取到的所有的http请求和响应头，一次请求的url、请求和响应之间通过空行隔开，不同的请求之间通过虚线隔开。在按照2.1中的步骤登入系统后，我们在列表中看到了2个请求。第一个请求核心内容如下(还记得HTTP协议的内容吗？)：

```
POST /polls/login HTTP/1.1  //请求行
********************        //此处省略其他请求头
Content-Type: application/x-www-form-urlencoded //请求实体类型
Content-Length: 41      //实体信息长度
    name=shiyanlou&pwd=shiyanlou&commit=Login //实体内容

HTTP/1.0 302 Found      //响应行，302重定向
Location: .             //重定向的路径
********************    //此处省略其他响应头
Set-Cookie: sessionid=aped4pzerxjgd3db0ixw0dowkgrdpsxb;expires=Thu, 15-Sep-2016 15:04:28 GMT; httponly; Max-Age=1209600; Path=/     //响应头，服务器回写cookie
```

为什么是2个请求而不是1个呢?通过分析登录请求发现，登陆成功之后服务器发送了302重定向响应，服务器要求浏览器重新请求首页，这就产生了第二个请求。再来分析第二个请求，可以看到它相比登录请求少了一些请求头

```
Cookie: sessionid=aped4pzerxjgd3db0ixw0dowkgrdpsxb
```

这个 Cookie 中的 sessionid 是从上一个响应头中的 Set-Cookie 行中获取，所以值相同。 session 的意思是会话，服务器通过session存储和维护与单个用户之间的会话信息。不同的用户在服务器端有不同的session，并且为了确保正常通信，每个用户的session都是唯一的。登录过程其实就是在验证用户登录信息后在session中存储用户登录标识的过程。在开发web应用时，“登陆成功”与“在session中存储用户登录标识”是等价的。整个登录流程如下图所示：



那么问题来了，服务器到底是如何区别不同用户的session的？为什么登陆成功会后要回写cookie呢？答案就是sessionid！每个用户的session都有独立的、唯一的编号sessionid用来标识用户身份。用户登录后，服务器通过从用户的session中读取登录标识来进行身份认证，因此必须要知道sessionid来访问用户的session，而使用cookie正是满足这个需求的方法。服务器将需要浏览器存储的信息通过 Set-Cookie 响应头发送给浏览器，浏览器会将cookie存储在本地，并在每次访问该网站时附带发送指定的cookie以满足服务器的需求，而通过cookie存储sessionid就是其中的一种应用。

2.3小结

对于服务器来说，登录=验证+写session。对于浏览器来说，登录=发送登录信息+获取带sessionid的cookie。可以说，只要获得了sessionid，就算实现了模拟登录。有了它我们便可以游离于系统之中。

三、使用Python实现登录(简单实例)

理解了登录过程的原理和细节之后，开始用Python来编写模拟登陆程序吧。在任意目录下新建 login_base.py 文件,使用你喜欢的编辑器打开。实现流程如下：

3.1导入模块

不要忘记编写文件头、导入必要的依赖模块哦

```
#!/usr/bin/python
#-*- coding:utf-8 -*-
import urllib
import urllib2
import cookielib
```

3.2构造登录请求

一个http请求由3部分组成：请求url、请求头以及请求实体(附带数据)。在Python的urllib2库中，由urllib2.Request对象描述一个request。其中url是一个字符串、请求头是一个dict，key是请求头名称，value是请求头的内容。至于请求数据就要分具体情况了，在表单提交这种场景下，请求数据类型为 application/x-www-form-urlencoded ,意思就是经过url编码的表单数据，数据的组织形式是 key=value ， 多组键值对之间用 & 分隔。登录请求的实体部分如下：

```
Content-Type: application/x-www-form-urlencoded //请求实体类型
Content-Length: 41      //实体信息长度
    name=shiyanlou&pwd=shiyanlou&commit=Login //实体内容
```

此时，我们只需要使用dict来存储键值对，再用 urllib.urlencode() 方法进行编码就可以了。最后创建 urllib2.Request 对象，全部代码如下：

```
url = 'http://localhost:8000/polls/login'
values = {
    'name':'shiyanlou',
    'pwd':'shiyanlou',
    'commit':'Login'
}
headers = {'Referer':'http://localhost:8000/polls/show_login'}
request = urllib2.Request(url,data=urllib.urlencode(values),headers=headers)
```

附加参数

需要注意的是，在浏览器页面操作时，用户只需要输入用户名和密码，看起来好像只需要提交2个参数到服务器。但实际上前端页面一般都会提交其他元参数到服务器，大多数都是与服务端的访问api设计有关，还有一些控制参数。一个登录请求包含5个以上的参数是再正常不过的事情，但并不是每一个参数都是登录所必要的，有些参数即使没有也不会影响正常登录。在实验demo中，commit这个参数就是必须提交的，否则会登录失败。

防盗链

Web 应用的资源都是有url的，只要获得了url就能够在任何地方引用。听起来很方便，但这可能会导致你的资源被别人盗用。为了访问量，把自己辛辛苦苦PS的一张美照放到个人站点上，却被别人的站点给引用走了，岂不是很气人？而HTTP的请求头 Referer 就是为了解决这类问题而生，Referer描述了当前请求的发出者，也就是引用者，服务器可以通过Referer来判断当前的引用者是否是合法的引用者从而决定是否返回请求的资源。考虑到一定的安全性，一般的网站登录都会限制Referer域来防止非法登录。因此，为了成功登录，我们需要在request头中填写Referer头，内容从抓取的请求头中照搬来即可。

3.3发送请求并保存cookie

一般情况下，打开url默认使用urllib2的 urlopen() 函数，但是它不能处理cookie。这里我们需要自己创建能够存储cookie的opener。python内置有cookielib库来处理cookie，其中的MozillaCookieJar可以将cookie存储到文件中，并可以从文件中读取cookie。先创建MozillaCookieJar对象，再使用 urllib2.HTTPCookieProcessor 创建cookie处理器，最后使用 urllib2.build_opener 创建opener。接下来就可以用opener发送请求并存储cookie了。代码如下：

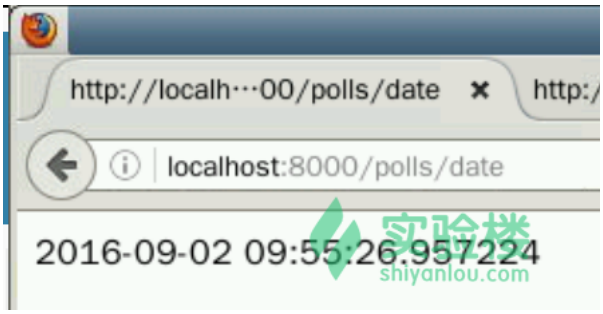
```
# 创建opener
cookies = cookielib.MozillaCookieJar('my_cookies.txt') # 指定cookie的存储文件
cookie_handler = urllib2.HTTPCookieProcessor(cookies)
opener = urllib2.build_opener(cookie_handler)

# 发送请求 & 保存cookie
response = opener.open(request)
cookies.save()
print response.code
print response.read()
```

如果登陆成功，就可以在指定的文件my_cookies.txt中看到sessionid了。

```
localhost.local    FALSE    /    FALSE    1474019663    sessionid vejomjbliggkwblzfagobv3u8foik6am
```

3.4 使用cookie访问系统服务

登录系统只不过是一个开始，使用sessionid访问系统服务才是最终目的。举一个简单的例子，我们的网站demo的首页中由一个查询系统当前时间的接口(在现实生活中可能是更有用的，比如信息查询)，我们在登录后点击它，可以看到它的url是 http://localhost:8000/polls/date ，这是只为登录用户提供的服务。如果我们现在返回首页，退出登录之后直接在地址栏中输入这个url，页面将会302重定向至登录页，无法看到当前时间。



在我们模拟登录成功后，就可以直接通过opener打开这个url来使用这项系统服务。代码实现如下：

```
url2 = 'http://localhost:8000/polls/date'
response2 = opener.open(url2)
print response2.code
print response2.read()
```

如果有是在另外一个py文件中使用这个cookie的话，再打开url之前需要先载入cookie：

```
# 载入cookie
cookie = cookielib.MozillaCookieJar()
cookie.load('my_cookies.txt')
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
```

当然，以上只是一个简单的示例，只是为了说明原理。实际应用中的请求中会有各种各样的请求参数、元数据参数甚至是多样的验证cookie，只有理解了原理才能够以不变应万变。

四、高级登陆练习

是不是原理听懂了，但是只实现demo感觉不够过瘾？我们来一起做一做下面这个稍微难一点的练习吧。

4.1题目

已知信息：

- 登录页面url : http://localhost:8000/admin/login/?next=/admin/ (http://localhost:8000/admin/login/?next=/admin/)
- 登录提交url : http://localhost:8000/admin/login/?next=/admin/ (http://localhost:8000/admin/login/?next=/admin/)
- 用户名： admin
- 密码 : djangoadmin

要求：

- 模拟登陆成功并存储cookie

4.2 过程分析提示

管理员的登录请求的请求如下：

```
POST /admin/login/?next=/admin/ HTTP/1.1
Referer: http://localhost:8000/admin/login/?next=/admin/
Cookie: csrftoken=adxAiQKnRYgQoS4RmoooCesA7mBgJtwVnsd98nttE1arcBAnGwRbILLvWeS5xLfM
Content-Type: application/x-www-form-urlencoded
Content-Length: 137
csrfmiddlewaretoken=tsxs3KcOUt4ZdMg1gMBCKkNV8JTxZigaGHd1ThVUHwYA1vMxAU4pQR6QXBamNAZ1&username=admin&password=djangoadmin&next
=%2Fadmin%2F

HTTP/1.0 302 Found
Vary: Cookie
Last-Modified: Fri, 02 Sep 2016 09:24:46 GMT
Location: /admin/
Content-Type: text/html; charset=utf-8
Set-Cookie: csrftoken=seowCkz5vprAPZbkqF7M4QuzYZPKgoUrsO5nE3WWxtYnc5NRKB34rmmrTAIDdCaK; expires=Fri, 01-Sep-2017 09:24:46 GM
T; Max-Age=31449600; Path=/
Set-Cookie: sessionid=cb6z7xlu31uxt8kdupfh0td43ewhgvpx; expires=Fri, 16-Sep-2016 09:24:46 GMT; httponly; Max-Age=1209600; P
ath=/
```

不难观察到登录请求中附带了 csrftoken 这个**cookie**，表单数据中必须提交 csrfmiddlewaretoken , username , password 这三个参数，最后的 next 参数是**url**中附带的，不需要额外添加。必须得到前两个参数才能够成功登陆，这两个参数从名字上来看都用来作认证令牌，应该是由服务器生成的。其中的**cookie**肯定是来自上一个**response**的回写，而从 csrfmiddlewaretoken 的位置来看，应该是在登陆表单当中。注销登陆，重新进入登录页面，点击右键查看源代码发现了这个输入域(每次请求的值都是不同的)：

```
<input type='hidden' name='csrfmiddlewaretoken' value='PSM5KbmRGdHraaVXK9UEzsWmLYlHxYjUPstWMUJIIheexgxu45QWWYOeG
zeAuczd' />
```

这样一来我们的思路就很清晰了：

解析页面可以使用**python**的正则表达式 re模块 或者是比较火的 beautifulsoup库 。希望同学们先自己动手挑战一下，欢迎再课程回复中和我讨论，届时我将提供登录代码。

五、总结&扩展

模拟登陆的关键是弄清楚登录请求需要提交的参数，重点是要获取带有**sessionid**的**cookie**，最终目的是不受限制的访问系统提供的有价值的服务。

只有分析清楚登陆过程才能达到最终目的，这就需要具备强的**HTTP**包分析能力，理解**HTTP**协议，并配合包分析工具(**FireBug**,**Live Http Header**,**Burp Suite**等)解析出关键参数，有时需要编辑和重发包来删减掉不必要的多余参数。下面给出一些提高的参考资料，学有余力的同学可以深入研究：

- 神器**Burp Suite**官网 (https://portswigger.net/burp/)
- 复杂的淘宝模拟登陆 (http://python.jobbole.com/81361/)
- 正则表达式30分钟入门 (http://deerchao.net/tutorials/regex/regex.htm)
- **beautifulsoup**官方中文文档 (http://beautifulsoup.readthedocs.io/zh_CN/latest/)