

爬下 12306

adream307@163.com

前言

本文在 Linux 平台上，以 Python 为开发工具，介绍 12306 抢票软件的基本原理，并引入示例，讲解如何自己编写一个 12306 抢票软件。对于 Windows 平台的读者，可以安装 Crywin 软件模拟 UNIX 的命令行界面。图形界面采用 Python 封装的 Qt 图形库。

在第一章中，我们举了个例子，讲如如何利用 Linux 系统中现存的 curl、grep 和 sed 抓取制定日期，指定车次的剩余票数。

在第二章中，我们讲述了一些和网络及 HTTP 相关的基础知识，第二章也是我耗时最长、篇幅最长的一个章节。

在第三章中，我们以 Python 为开发工具结合 pyqt，一步一步讲述如何实现软件订票。

本来我还想写得更多，但是我发现当讲完基本原理后，剩下的很多事是和 Coding 息息相关的。所以我也不知道如何下笔了。

另一层原因是，本文的出发点是介绍抢票软件的工作原理，在这一点上，我觉得我已经讲明白了。另外市面上专业的抢票软件已经够多了，我无意去挣，也挣不过。

当然，最主要的原因是自己变懒了。

所以抱歉挖了个坑。

本书相关的源代码见：<https://code.csdn.net/adream307/fetch12306.git>。

最后打个小广告，虽然这里挖了一个坑，我保证下次我绝不挖坑。

下次我想结合 Wireshark 讲讲 TCP 的故事。

1. 从 0 到 1

今天，天气晴朗，阳光明媚，你正开开心心地正坐在电脑前办(fa)公(dai)，这时 Boss 过来了，通知你周末要陪他出差，去北京，让你现在给查查看还有哪些票？

只见你不紧不慢的打开一个命令行终端，有条不紊的敲入“**fetch_sh-bj.sh 2015-07-18**”，Boss 还没明白过来你在干什么时，你悠然自得的将电脑屏幕转过去，气定神闲的说，“老大，周末从上海到北京的还有这些车次，每行最后 3 个数分别是二等座，一等座和商务座的剩余票数，我们订哪趟车”？

```
cyf@cyf$./fetch_sh-bj.sh 2015-07-18
% Total % Received % Xferd Average Speed   Time   Time   Time Current
          Dload Upload Total Spent Left Speed
100 41904 100 41904    0     0 144k      0 ---:---:---:---:---:---:---:--- 155k
G102 上海虹桥 北京南 06:43 12:17 630 64 9
G104 上海虹桥 北京南 07:00 12:23 5 140 11
G106 上海虹桥 北京南 07:10 12:42 148 86 24
G108 上海虹桥 北京南 07:20 13:11 无 67 8
G110 上海虹桥 北京南 07:30 13:32 294 102 23
G12 上海虹桥 北京南 08:00 13:16 421 108 26
G112 上海虹桥 北京南 08:05 13:53 401 60 11
G114 上海虹桥 北京南 08:18 14:12 36 123 25
G2 上海虹桥 北京南 09:00 13:48 无 71 6
G116 上海虹桥 北京南 09:34 15:23 270 85 13
G118 上海虹桥 北京南 09:54 15:49 311 73 7
G14 上海虹桥 北京南 10:00 14:55 24 114 14
G120 上海虹桥 北京南 10:05 15:59 97 131 25
G42 杭州东 北京南 10:28 16:06 无 73 11
G122 上海虹桥 北京南 10:46 16:33 497 97 11
G16 上海虹桥 北京南 11:00 15:55 127 108 15
G124 上海虹桥 北京南 11:05 16:55 481 91 8
G126 上海虹桥 北京南 11:10 17:00 599 89 10
G128 上海虹桥 北京南 11:15 17:13 586 40 7
```

图 1-1 动车订票信息

有没有觉得这个很酷，逼格是不是很高，是不是应该 get 这项新技能？

那么问题来了，如果 boss 不是出差去北京，而是去深圳或西安，我们如何依然保持这高格调呢？

为了解决后顾之忧，我们必须弄明白 fetch_sh-bj.sh 干了哪些事，然后山寨出个上海到深圳的查票工具 fetch_sh-sz.sh，以及上海到西安的查票工具 fetch_sh-xa.sh 之类的工具。不过，最终解决方案应该是弄个 fetch12306.sh，那么查上海到北京的动车票应该是这样的：**fetch12306.sh 上海 北京 2015-07-18**。如果你对这些感兴趣，那么我相信本书一定符合你的口味。

废话不多说，让我们先看看 fetch_sh-bj.sh 到底干了哪些事。

Unix 哲学是这样的：一个程序只做一件事，并做好。程序要能协作。程序要能处理文本流，因为这是最通用的接口。

fetch_sh-bj.sh 正是这种哲学细想下的产物，拼装现有的工具，协作完成我们期望的功能。该程序是个 bash 脚本文件，可以直接使用文本编辑器打，如图 1-2 所示。

可以清楚的看到，脚本以竖线(“|”)为单位划分成三部分：第一部分使用 **curl** 命令从 12306 官方网站获取车票信息，第二部分使用 **grep** 命令将车票信息按照车次分离，最后使用 **sed** 命令提取并打印需要的信息。通过 curl、grep 和 sed 三个工具的协作实现我们想要的功能。

```
1. #!/bin/bash
2. curl --insecure --user-agent "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0" "https://kyfw.12306.cn/otn/lcxxcx/query?purpose_codes=ADULT&queryDate=$1&from_station=SHH&to_station=BJP" | grep -oP "(?<=0[^{}]+(?=})" | sed -r 's/.*station_train_code": "([^\"]+).*start_station_name": "([^\"]+).*end_station_name": "([^\"]+).*start_time": "([^\"]+).*arrive_time": "([^\"]+).*ze_num": "([^\"]+).*zy_num": "([^\"]+).*swz_num": "([^\"]+).*\1 \2 \3 \4 \5 \6 \7 \8/'
```

图 1-2 fetch_sh-bj.sh

1.1. 爬取信息

对于图 1-2 中的脚本程序，也许你现在还看得一头雾水。但请不要着急，熬过了黑夜就可以见到黎明的曙光，先喝一口 24K 纯度的凉白开压压惊，下面听我为你娓娓道来关于 fetch_sh-bj.sh 前世今生。

话说 fetch_sh-bj.sh 一共可以分为三部分，这一小节我们先聊聊和爬取信息相关的那一部分--**curl**。

curl 命令可以分为三段：

第一段：--insecure 选项，告知 curl 不对当前网站的证书做校验。我相信很多童鞋在第一次使用 12306 网站定票时，都有过类似的体验，打开订票页面时，浏览器爆出个“当前网页不受信任，是否继续”之类的警告信息。

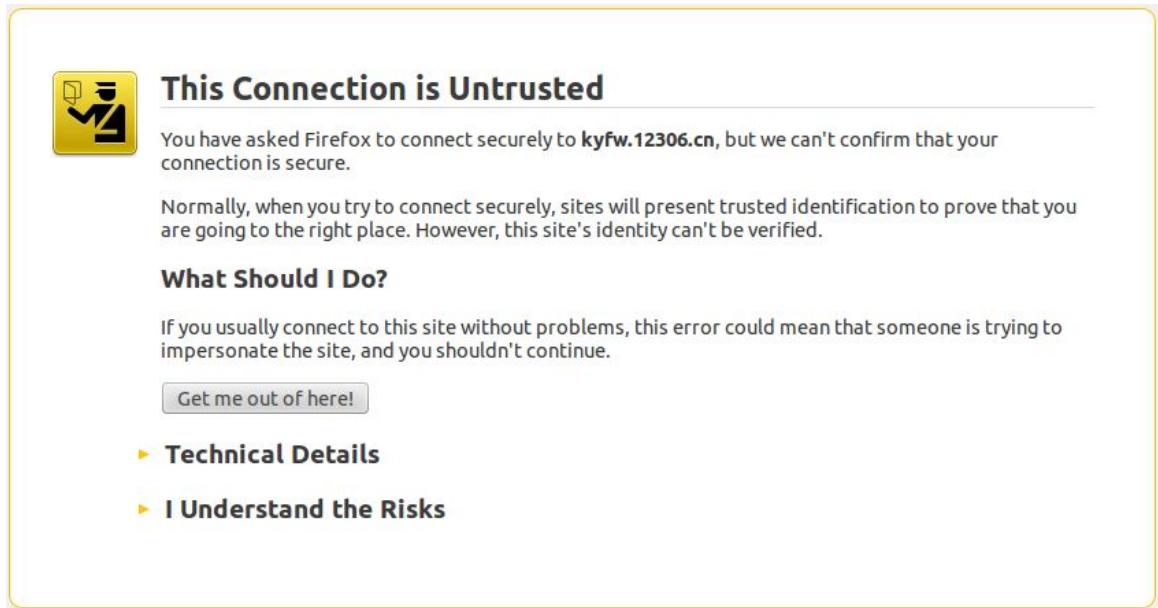


图 1-3 FireFox 当前网页不受信任警告

curl 在爬取订票信息时，干着和浏览器类似的事，如果不指明 insecure 选项，

则会显示当前网页认证失败，如图 1-4 所示。

读到这里，爱钻牛角尖的读者一定会问，“可不可以不使用 `insecure` 选项，并且 `curl` 依然可以成功认证”？

嗯，这是个好问题，关于这个问题，我先说一下结论，在后文会给出详细解释，结论就是“YES”。

```
cyf@cyf$curl --user-agent "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101
Firefox/38.0" "https://kyfw.12306.cn/otn/lcxxcx/query?purpose_codes=ADULT&query
Date=$1&from_station=SHH&to_station=BJP"
curl: (60) SSL certificate problem, verify that the CA cert is OK. Details:
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
More details here: http://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle"
of Certificate Authority (CA) public keys (CA certs). If the default
bundle file isn't adequate, you can specify an alternate file
using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
the bundle, the certificate verification probably failed due to a
problem with the certificate (it might be expired, or the name might
not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use
the -k (or --insecure) option.
```

图 1-4 curl 网页认证失败

第二段：`--user-agent`，该选项的值是“Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0”，网页抓取的基本原理就是模拟浏览器向服务器请求数据。在 FireFox 浏览器中打开网页时，浏览器向服务器发送类似“Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0”(不同浏览器版本下该值也许略有差异)的 `user-agent`。在这个例子中，`curl` 就是模拟浏览器从服务端获取数据，所以我们添加了这段用于欺骗服务器的 `user-agent` 声明。

细心的读者也许发现了，在这个例子中，不添加 `user-agent` 也是可以运行的。

确实如此，如果不添加 `user-agent`，则 `curl` 使用默认的 `user-agent`: `curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3`，不同版本下该值也许略有差异。

那是不是 `user-agent` 完全没用呢？

也不尽然，只是在本例中没有体现出来而已。客户端一般通过 `user-agent` 向服务端声明自己，服务器根据这个 `user-agent` 申明，判断客户端浏览器类型，针对不同的浏览器服务器给出不同的响应。最简单的例子就是智能机上浏览器和我们笔记本上的浏览器。不管屏幕尺寸多大的智能机，相比笔记本的屏幕还是很小的，所以同一个网页在手机上屏幕上呈现效果和在笔记本上绝对是不一样的；而且智能机不一定总是处在 `wifi` 环境，所以对相同的网页请求，服务器发给手机的数据量肯定比发给笔记本的数据量少。

举例说明，我笔记本上 FireFox 的 `user-agent` 是“Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0”；同时我手机上 UCWeb 的 `user-agent` 是“Moz

illa/5.0 (Linux; U; Android 4.2.2; en-US; HUAWEI P6-T00 Build/HuaweiP6-T00) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 UC Browser/10.6.0.586 U3/0.8.0 Mobile Safari/534.30”。

```
cyf@cyf$curl https://www.baidu.com --user-agent "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0" > ff_baidu.html
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total Spent   Left Speed
100 97320    0 97320    0      0  883k    0 --::-- --::-- --::--  989k
cyf@cyf$curl https://www.baidu.com --user-agent "Mozilla/5.0 (Linux; U; Android 4.2.2; en-US; HUAWEI P6-T00 Build/HuaweiP6-T00) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 UC Browser/10.6.0.586 U3/0.8.0 Mobile Safari/534.30" > uc_baidu.html
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total Spent   Left Speed
100 40153  100 40153    0      0  245k    0 --::-- --::-- --::--  264k
cyf@cyf$ls -l ff_baidu.html uc_baidu.html
-rw-rw-r-- 1 cyf cyf 97320 Jul 18 18:03 ff_baidu.html
-rw-rw-r-- 1 cyf cyf 40153 Jul 18 18:03 uc_baidu.html
```

图 1-5 模拟 FireFox 和 UCWeb 下载

我们分别假装自己是 FireFox 和 UCWeb 下载百度首页，并存文件，如图 1-5 所示。从文件量可以看出 FireFox 共下载了 97K 的数据，而 UCWeb 下载了 40K 的数据。直接使用浏览器打开 ff_baidu.html 和 uc_baidu.html 可以发现两者的展示效果亦不相同。

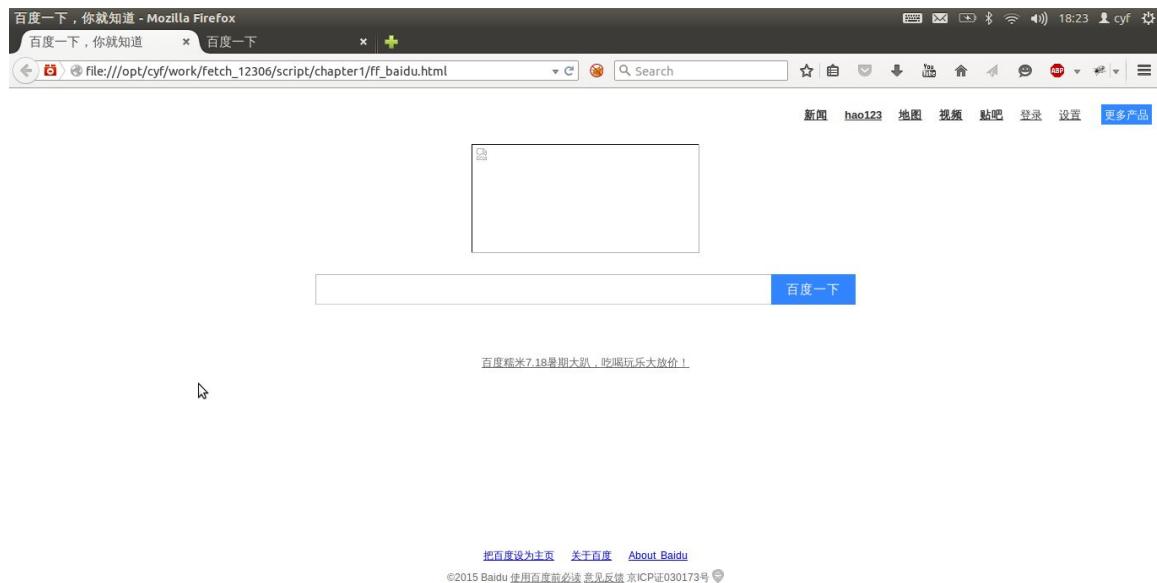


图 1-6 ff_baidu.html

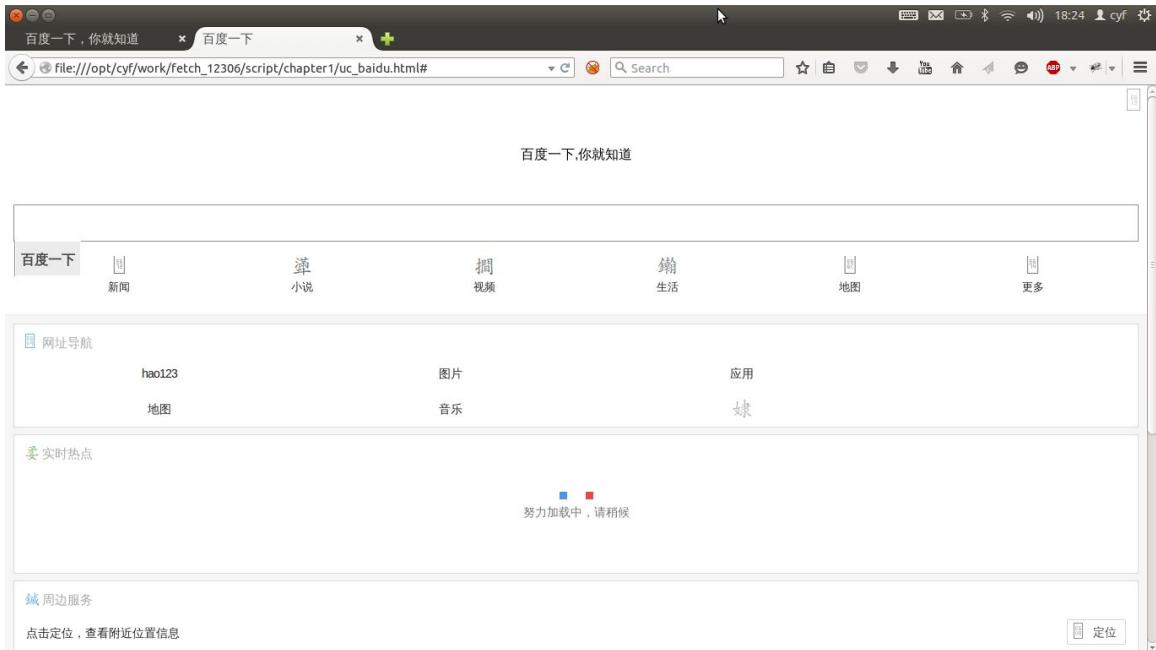


图 1-7 uc_baidu.html

读到这里，也许有些读者会问：“我承认服务器对不同的 user-agent 可能给出不同的响应，但是我怎么知道我自己浏览器的 user-agent 呢”？

嗯，这也是个好问题，这里先给出一种解决方案，后文继续给出其它解决方案。最简单的方法就是打开“<http://whatsmyuseragent.com>”网页，即可显示当前浏览器的 user-agent。该网站同时列出了常用设备的 user-agent。

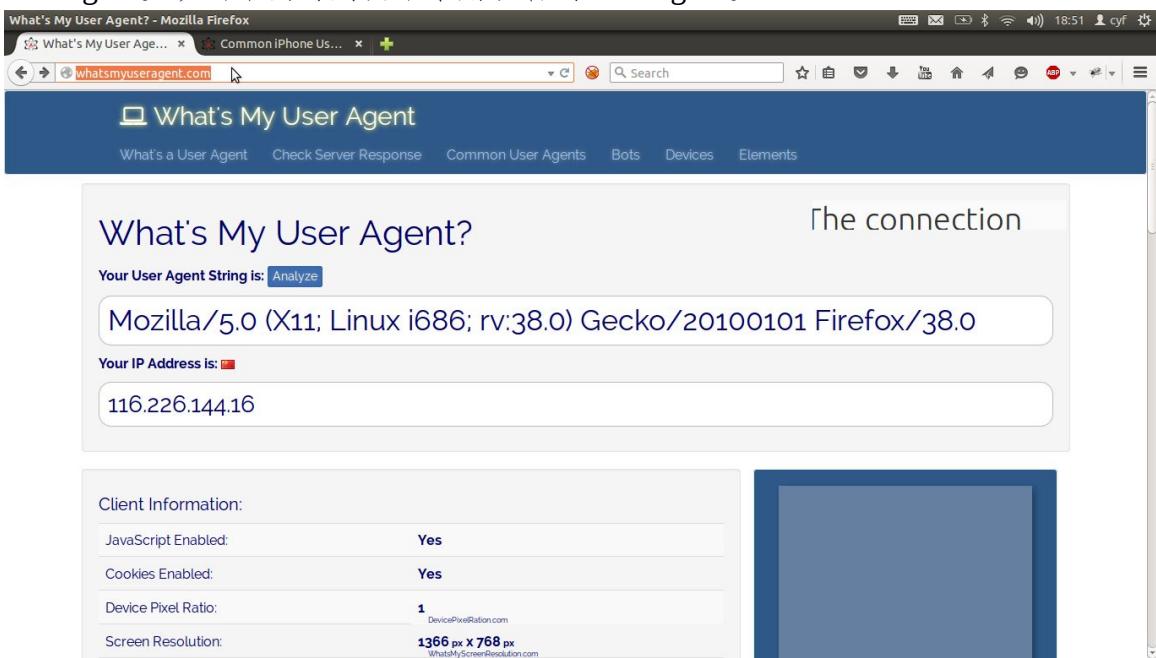


图 1-8 What's my user agent

聪明如你的读者一定能猜到第三段其实就是请求订票信息的 URL ([https://kyfw.12306.cn/otn/lcxxcx/query?purpose_codes=ADULT&queryDate=\\$1&from_station=SHH](https://kyfw.12306.cn/otn/lcxxcx/query?purpose_codes=ADULT&queryDate=$1&from_station=SHH))

&to_station=BJP), 而且你也一定猜到 queryDate=\$1 , 中的 \$1 就是我们手动敲入的日期。但是，你一定有这样几个小小的疑问：

- 1.这个请求订票信息的 URL 是如何找到的？
 - 2.from_station=SHH 中的 SHH 代表的是“上海火车站”、“上海南站”、“上海虹桥”还是“上海”？
 - 3.同理 BJP 代表的是北京还是北京的某个站点？
 - 4.SHH 和 BJP 是如何得到的，如果我想知道深圳、西安的代号，又该从哪找？
- 嗯，这些确实是问题，而且这些问题的解决手段是相同的，都是关于如何利用好现有工具的问题。

换位思考一下，当使用笔记本在浏览器查订票信息时，浏览器也一定向 12306 网站发送了相应的查票请求，只是浏览器把这些东西放在后台完成，没有对用户展现而已。如果我们有办法把浏览器查票时与服务器交互的所有操作均展示出来，那么我们是否可以解决以上几个疑问呢？

借助 FireFox 自带的开发者工具或者下载 Firebug 网页调试插件，可以把浏览器与服务器交互所有信息一览无余的展示给用户。如果读者使用 Chrome 或 Opera 或 IE 等其它浏览器，开发商也提供响应的网页调试工具。此处，我们使用 FireFox 自带的开发者工具来解决上述几个疑问。

在浏览器中打开 12306 查票页面，点击浏览器右上角的“Open menu”，选择“Developer”选项下的“Network”；或者点击“Tool”菜单，选择“Web Developer”下的“Network”选项，打开网页调试工具。

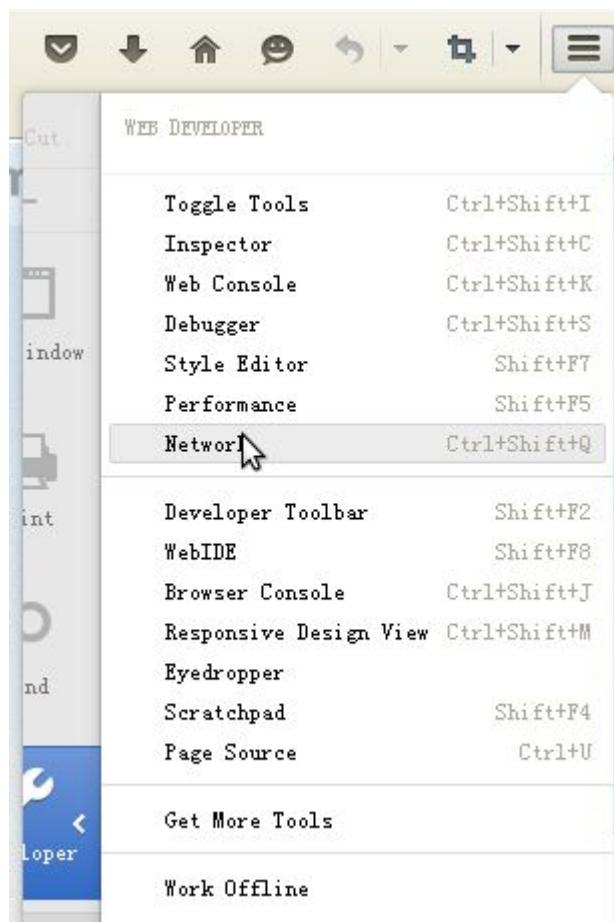


图 1-9 Openmenu->Developer->Network



图 1-10 Tools->Web Developer->Network

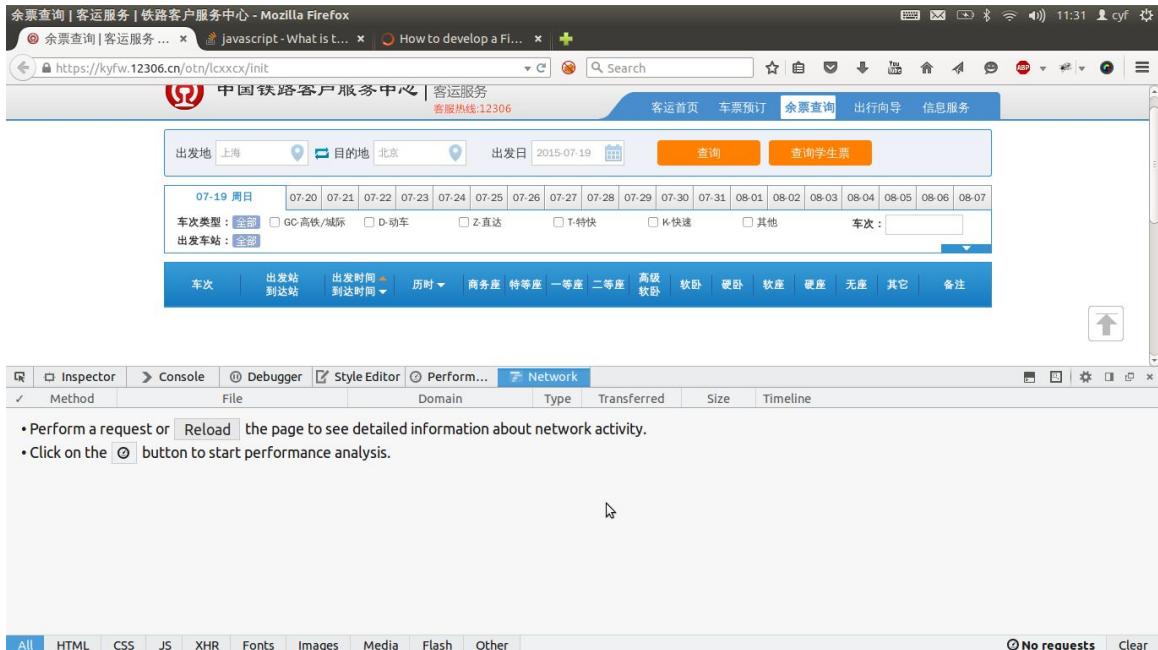


图 1-11 网页调试工具

随意选择“出发地”、“目的地”和“出发日”，点击查询。

凭借强大的调试工具，Network 将上述操作过程中，浏览器和 12306 服务器之间交互的数据均清晰的呈现出来。

聪明的读者一定注意到了，当我们在页面上点击“查询”时，Network 立刻多出一条信息：“**200 GET query?prurpose_codes=ADUILT&query...**”。

这条信息就是查询订票信息时，浏览器向服务器发送的数据，点击该条信息，可以在右边的小窗口中得到更为详细的描述，包括 Headers、Cookies、Params 等信息，图 1-12。

在 Headers 选项卡下，展开 Request Headers，可以查看浏览器向服务器发送查票请求时的 Header 信息，这里面就包含我们的 User-Agent。这是获得 User-Agent 的第二种方法。

Headers 选项卡中的 Request URL 即为第 1 个小疑问的答案。

选择 Params 选项卡，即可得到请求订票信息时，浏览器向服务器发送的参数，这些参数放在 URL 的“?”后面，并使用“&”区分不同的参数。

浏览器向服务器传递擦参数时，常用的方法有两种：GET 和 POST。GET 方法传递的参数直接加在 URL 后面，一般用于传递公开信息，而 POST 发则由浏览器在后台发送，一般用户传递用户名和密码等非公开信息。

Params 选项卡中内容即为第 2 个和第 3 个小疑问的答案，SHH 代表上海，BJP 代表北京。

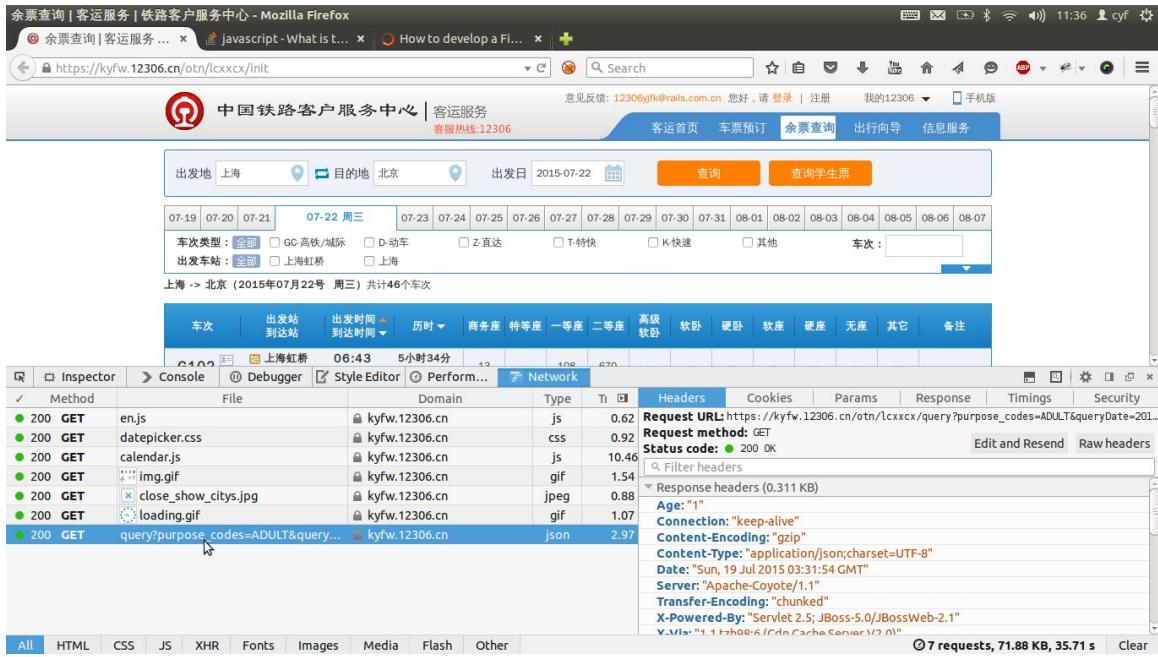


图 1-12 车票查询

那么如何解答第 4 个小疑问呢？

也许你会说，打开 Network，在页面上分别选择“深圳”和“西安”，点击“查询”，不是可以在 Network 中显示吗？

嗯，这确实是个临时性的方法，但是 12306 网站上一共有上千个动车站点（截至我写这本书时，一共有 2415 个站点），我们总不能每个站点都用这么土的方法获取吧，这样做即不准确，也浪费时间。所以一定有更优雅的，更高端的方法让我们一次性获得所有站点名称和代号。

让我们再次拿起 Network 这个调试利器，我们可以发现在选“出发地”和“目的地”时，只有第一次操作时，浏览器用 GET 方法向 12306 网站请求一个名为“close_show_cities.jpg”的图片（吐槽一下，city 的复数形式是 cities 不是 citys），之后无论如何修改“出发地”和“目的地”，浏览器均不与服务器发生任何数据交互。

据此可以断定，站点名称和站点代号对应关系的数据在用户打开查票网站 (<https://kyfw.12306.cn/otn/lcxxcx/init>) 时已经传入浏览器，所以之后选择站点时不需要从服务器再次获取。

因此需要看看页面打开时，服务器向浏览器发送了哪些数据。点击 Network 右下角的 clear，再点击刷新。

在 Network 的一堆数据中，我们注意到有这么一条信息：

200 GET station_name.js?station_version=1.8...

点击这条信息，在右边展开的小窗口中选择“Response”，这里就有我们想要的信息，所有的站点以及站点所对应的代号。

Method		File	Domain	Type	Ti
200	GET	init	kyfw.12306.cn	html	6.3
200	GET	WdatePicker.css	kyfw.12306.cn	css	0.1
200	GET	station_name.js?station_version=1.8...	kyfw.12306.cn	js	42.2
200	GET	WdatePicker.js	kyfw.12306.cn	js	3.9
200	GET	lcxxcx_css.css?cssVersion=1.8397	kyfw.12306.cn	css	15.4
200	GET	lcxxcx_js.js?scriptVersion=1.8399	kyfw.12306.cn	js	108.
200	GET	favorite_name.js	kyfw.12306.cn	js	0.6
200	GET	qss.js?station_version=1.8395	kyfw.12306.cn	js	11.2
200	GET	bg.png	kyfw.12306.cn	png	0.3
200	GET	logo.png	kyfw.12306.cn	png	29.6
200	GET	icon_phone.png	kyfw.12306.cn	png	0.2
200	GET	icon8.png	kyfw.12306.cn	png	0.6

图 1-13 获取站点代号

图 1-14 站点名词余代号的对应关系

现在，我们差不多解答了第 4 个小疑问，为什么是差不多呢？因为直接在 `Response` 里看不舒服，我们应该把这些信息提取出来，做成一张一一对应的表，一列是站点名词，一列是站点代号。此处我们使用 `Bash` 脚本完成此事，可以使用文本编辑器打开 `station_name.txt` 获得直观的站点名词和代号的对应关系。

- ```
1. curl --insecure https://kyfw.12306.cn/otn/resources/js/framework/station_name.js?
station_version=1.8395 | grep -oE "@[^@]+@" | gawk '{split($0,z,"|");print z[2],
z[3]}' > station_name.txt
```

图 1-15 fetch station name.sh

到此位置，我们完成了对 `fetch_sh-bj.sh` 脚本的 `curl` 命令的解读。我相信读者可以据此快速修改出 `fetch_sh-sz.sh`、`fetch_sh-xa.sh` 之类的脚本了。

在结束对 curl 的讲解前，我们先看看 curl 命令的最终输出结果，读者可以在 Network 的 Response 查看，或使用文本编辑器打开 sh-bj\_2015-07-18.txt 查看。

- ```
1. curl --insecure --user-agent "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0" "https://kyfw.12306.cn/otn/lcxxcx/query?purpose_codes=ADULT&queryDate=2015-07-18&from_station=SHH&to_station=BJP" > sh-bj_2015-07-18.txt
```

图 1-16 curl 命令从服务器返回的结果

```

{
  "datas": [
    {
      "train_no": "5l0000G10241",
      "station_train_code": "G102",
      "start_station_telecode": "AOH",
      "start_station_name": "上海虹桥",
      "end_station_telecode": "VNP",
      "end_station_name": "北京南",
      "from_station_telecode": "AOH",
      "from_station_name": "上海虹桥",
      "to_station_telecode": "VNP",
      "to_station_name": "北京南",
      "start_time": "06:43",
      "arrive_time": "12:17"
    }
  ]
}

```

图 1-17 curl 响应

1.2. 理清车次

聊完了 curl，现在我看开始聊聊 grep。正所谓温故而知新，在正式开始之前，我们先回顾一下 Unix 哲学：一个程序只做一件事，并做好。程序要能协作。程序要能处理文本流，因为这是最通用的接口。

curl 命令输出结果是一段文本，图 1-18，通过管道(|)将这段文本传入 grep 命令，作为它的输入。所以 grep 命令的核心是从 curl 命令的输出文本中提取需要的目标信息。

```
{
  "validateMessagesShowId": "_validatorMessage",
  "status": true,
  "httpstatus": 200,
  "data": {
    "datas": [
      {
        "train_no": "5l0000G10241",
        "station_train_code": "G102",
        "start_station_telecode": "AOH",
        "start_station_name": "上海虹桥",
        "end_station_telecode": "VNP",
        "end_station_name": "北京南",
        "from_station_telecode": "AOH",
        "from_station_name": "上海虹桥",
        "to_station_telecode": "VNP",
        "to_station_name": "北京南",
        "start_time": "06:43",
        "arrive_time": "12:17",
        "day_difference": "0",
        "train_class_name": "",
        "lishi": "05:34",
        "canWebBuy": "Y",
        "lishiValue": "334",
        "yp_info": "O055300630M0933000649174800008",
        "control_train_day": "20300303",
        "start_train_date": "20150718",
        "seat_feature": "O3M393",
        "yp_ex": "O0M090",
        "train_seat_feature": "3",
        "seat_types": "OM9",
        "location_code": "H1",
        "from_station_no": "01",
        "to_station_no": "09",
        "control_day": 59,
        "sale_time": "1330",
        "is_support_card": "1",
        "note": "",
        "gg_num": "--",
        "gr_num": "--",
        "qt_num": "--",
        "rw_num": "--",
        "rz_num": "--",
        "tz_num": "--",
        "wz_num": "--",
        "yb_num": "--",
        "yw_num": "--",
        "yz_num": "--",
        "ze_num": "630",
        "zy_num": "64",
        "swz_num": "8"
      },
      {
        "train_no": "5l0000G10441",
        "station_train_code": "...",
        ...
      }
    ]
  }
}
```

图 1-18 curl 输出的文本片段

这种格式文本有个专用名称：JSON。先不管 JSON 是什么东东，仔细阅读文本，可以发现“**datas**”后面中括号([...])间的内容就是我们想要的各个订票车次的信息，而且每个车次信息均包围在一对大括号({...})中，所以在本例中 grep 的任务就是提取这

对大括号中的内容，并打印输出。

这里我们需要用到关于正则表达式的知识，关于正则表达式更多内容推荐阅读 Jeffrey 的《精通正则表达式》。

此处我们需要使用 grep 帮我提取一段文本，这段文本具有以下特征：

特征一：文本必须以一个左大括号({})开始，以右大括号(})结束；

特征二：左大括号({})和右大括号(})间可以包含除“{}”和“}”外的任何内容；

特征三：只需要提取大括号里面的内容，大括号本身可以不需要；

特征二是必须说明的，因为所有的 datas 均包含在 data 的大括号中，而且 data 外层还包含一层大括号，所以我们必须明确的告诉计算机我们需要提取的这段内容包含在一对大括号内，而且这对大括号内不能嵌套大括号。结合这三个特征，我们编写在 grep 命令：

```
grep -oP "(?<=\{)[^}]*(?=})"
```

读者可以输入 **grep --help**，在帮助信息中可以得知-o 选项表示只输出匹配的文本，-P 选项表示采用 Perl 格式的正则表达式。在解释本例中正则表达式的含义前，先学习一下相关的正则表达式知识。

“(?<=>Express)”在正则表达式里称为环视(Lookaround)，它一共有 4 种分类：

1. 逆序肯定环视(positive look behind)“(?<=Express)”：寻找一个位置点，这个位置点的左侧与表达式 Express 相匹配；
2. 逆序否定环视(negative look behind)“(?<!Express)”：寻找一个位置点，这个位置点的左侧不能与表达式 Express 相匹配；
3. 顺序肯定环视(positive look ahead)“(?=Express)”：寻找一个位置点，这个位置点的右侧与表达式 Express 相匹配；
4. 顺序否定环视(negative look ahead)“(?!=Express)”：寻找一个位置点，这个位置点的右侧不能与表达式 Express 相匹配；

这个读起来是不是很拗口，有没有读“化肥会挥发”这种绕口令的感觉。关于环视，我们需要认识很关键的一点：环视的匹配结果是一个位置点，所以环视的匹配结果不包含任何字符，它仅仅是一个位置点，因此环视又被称为“零宽度断言(Zero-Length Assertion)”。在本例中，我们需要从“{}”的右侧开始匹配，但是我们并不需要“{}”，所以这里我们使用逆序肯定环视来确定这个位置点。

“[...]”：中扩号在正则里表示一个字符集。

什么叫字符集呢？

嗯，举例说明“[abcd]”表示匹配一个字符，这个字符可以是 a，可以是 b，可以是 c 也可以是 d，但是不能是 e 或 f 等其他内容。

需要明白很关键的一点：字符集只匹配一个包含在这个集合中的字符，只是一个，仅此而已。那我想匹配多个字符，那该怎么办呢？这是个好问题，后文回答。

在本例中，我们需要匹配的字符是除“{}”和“}”外的所有字符，我们总不能把所有其它字符全写上吧，这个也不现实啊。

如果字符集的第一字符是“^”，则表示除该集合意外的任何其他字符，所以“[^{}]"表示除“{"和"}”外的所有字符。这里也需要明确一点，必须在中括号里的第一个字符是“^”。例如“[^{}]", 表示的字符集为“{"或"}”或“^”中的一个字符，不能为其它字符；同理，“[^]”表示除“^”外的任意字符。

“+”：为量词，用来修饰前一个字符，表示至少一个字符，上不封顶。举例说明“a+”表示至少一个 a，因此可以是 a，或 aaaaaa，但是不能是 cccc。

现在让我们来理解一下本例 grep 正则表达式的含义：

(?<=)：表示寻找一个位置点，该位置点的左侧是一个左大括号；

[^{}]+：表示匹配至少一个字符，上不封顶，该字符为除“{"和"}”外的任意字符；

(?=)：表示寻找一个位置点，该位置点的右侧是一个右大括号；

正则表达式理解起来确实有点困难，但是一旦理解掌握了，它就是你手中的利器，我们将要讲解的第三个命令 sed 也使用了正则表达式，因此强烈建议读者花点时间阅读关于正则表达式的相关书籍。

1.3. 突出重点

一个简单的 fetch_sh-bj.sh 脚本，写到现在还没写完，我深表歉意。

天地良心，真心不是作者想拖沓，我力求希望读者不仅仅能“知其然”，更希望读者“知其所以然”。作者力求向读者讲明白每一步的含义，希望读者下次碰到类似的问题时，可以依稀记得曾经在某本书上读过类似的解决方案，作者就很开心了。

在讲解 sed 命令之前，我们先喝碗心灵鸡汤提提神。

Long long ago，有个很有钱很有钱的土豪，整天没事干就是到处买、买、买。一天他闲来无事，买了栋别墅。那时大约在冬季，别墅的小草坪上依稀长着几株杂草。

如此高大上的别墅怎么可以配个破草坪呢？于是土豪大手一挥，把草坪翻了个新，种上各种名贵花木，还经常邀请圈内的成功人事来别墅开 Party，玩 High 了，总是有意无意的到草坪上走走，乘机得瑟自己的文化素养，点评一下草坪上新种的各种名贵花木，顺便寒碜别墅的原主人，草坪上种杂草是多么的没品味啊。

久而久之，这话传到别墅原主人那，别墅的原主人只能对土豪的鉴赏能力和品味连连摇头。因为原别墅草坪上种的并不是所谓的“杂草”，而是他苦心栽培的名贵花木，只是当时在冬季，没开花而已，却被不识货的土豪当杂草给除了，真是可惜啊。

喝完了鸡汤有啥感觉呢，不要说自己呛着了。。。

现在我们来个语文的阅读理解，作为一名攻城师，这个故事给了我们什么启发呢？

1. 作为项目开发人员，一定要写好文档，程序中的关键变量及关键函数，一定要给出详细注释，不要让你的接手人员去猜这个是名贵花木，还是杂草。
2. 接收别人项目时，不要总是一边问候对方亲戚，一边重构项目。在完全理解

别人设计意图前，不要轻易修改他人设计，存在即合理。即使重构，也不要
把整个草坪翻新，从零开始。站在巨人的肩膀上可以看得更远。

站在巨人的肩膀上可以看得更远，合理的利用现有的现成的工具，我们可以做
到事半功倍。上学时遇到过这样的问题，要求删掉文件中的所有空行。

当时刚学 C++不久，于是我给出了这样的解决方案：

```
1. #include<iostream>
2. #include<fstream>
3. #include<string>
4.
5. using namespace std;
6.
7. int main(int argc,const char* argv[])
8. {
9.     ifstream infile(argv[1]);
10.    if(infile.bad()) return -1;
11.    string s;
12.    while(getline(infile,s))
13.    {
14.        if(s=="") continue;
15.        cout << s << endl;
16.    }
17.    return 0;
18. }
```

图 1-19 C++删除空行

细心的读者也许注意到了，上面这个程序只能删除空行，如果行包含 Tab 或空
格等空字符，则上面这个程序是无能为力的。那对于这种情况，我们该如何处理呢？

如果读者是从头到尾一口气看到这里的话，我相信你脑中一定灵光一闪，蹦出
解决方案，没错，答案就是正则表达式。

```
grep -v -E "^\s* $" old_file > new_file
```

敲入“grep --help”查看帮助文档，grep 命令的-v 选项表示只输出不匹配的行。

正则表达式中“^”表示一行的开始位置，它环视类似，只匹配一个位置，不匹配
任何字符；“\$”表示行的结束位置；“\s”代表任何空字符，可以是空格或 Tab 键。

综合起来，正则表达式“^\s* \$”代表不包含任何内容，或只包含空格或 Tab 的行。

除了 grep 能解决该问题外，我们将要介绍的 sed 也可以解决这个问题。sed 全
称 stream editor，是一款文本搜索和过滤工具(<http://www.gnu.org/software/sed>)。

回到本文主题，我们的目标是使用 sed 从目标文本中搜索列车车次、始发站、
终点站、剩余车票数目等关键信息。

```
"train_no":"5l0000G10241","station_train_code":"G102","start_station_telecode":"AOH","start_station_name":"上海虹桥","end_station_telecode":"VNP","end_station_name":"北京南","from_station_telecode":"AOH","from_station_name":"上海虹桥","to_station_telecode":"VNP","to_station_name":"北京南","start_time":"06:43","arrive_time":"12:17","day_difference":"0","train_class_name":"","lishi":"05:34","canWebBuy":"Y","lishiValue":334,"yp_info":"O055300630M0933000649174800008","control_train_day":"20300303","start_train_date":"20150718","seat_feature":"O3M393","yp_ex":"O0M090","train_seat_feature":3,"seat_types":OM9,"location_code":H1,"from_station_no":01,"to_station_no":09,"control_day":59,"sale_time":1330,"is_support_card":1,"note":"","gg_num":--,"gr_num":--,"qt_num":--,"rw_num":--,"rz_num":--,"tz_num":--,"wz_num":--,"yb_num":--,"yw_num":--,"yz_num":--,"ze_num":630,"zy_num":64,"swz_num":8"
```

图 1-20 grep 命令输出文本

如图 1-20 所示，grep 命令输出的文本规律很明显，“station_train_code”所对应的即为列车车次，“start_station_name”对应始发站，“end_station_name”对应终点站，“start_time”对应出发时间，“arrive_time”对应到达时间，“ze_num”对应二等座剩余票数，“zy_num”对应一等座剩余票数，“swz_num”对应商务座剩余票数。我们以“station_train_code”所对应的列车车次为例，说明如何提取目标信息。

列车车次信息具有如下特征：

特征一：目标文本的左侧是：**station_train_code": "**

特征二：目标文本的右侧是一个双引号(")

特征三：目标文本的不包含双引号("")

有没有觉得这几个特征描述很熟悉，在前面介绍 grep 命令时，我们提取的目标文本是一对大括号内的内容，此处我们的目标文本也可以简单的描述为一对双引号内的内容，如果我们的最终目标只需要提取车次的话，我相信聪明的读者已经能够用 grep 命令解决该问题了。

```
grep -oP '(?<=station_train_code":")[^"]+(?=")'
```

但是，我们现在的目标不是仅仅提取列车车次，还要提取始发站、终点站等信息。也许你会说依然使用 grep 命令，在提取列车车次的基础上再添加一个用于提取始发站的环视，车次和始发站之间的文字使用“.*”匹配。

```
grep -oP '(?<=station_train_code":")[^"]+(?=").*(?<=start_station_name":")[^"]+(?=")'
```

嗯，这个方法看上去很美好，但是它的输出确实这样子滴：

```
G102","start_station_telecode":"AOH","start_station_name":"上海虹桥  
G104","start_station_telecode":"AOH","start_station_name":"上海虹桥  
G106","start_station_telecode":"AOH","start_station_name":"上海虹桥  
G108","start_station_telecode":"AOH","start_station_name":"上海虹桥
```

图 1-21 grep 命令输出

它除了输出我们需要的列车车次和始发站信息外，还输出了我们不想要的中间

信息。因为 grep 命令只知道输出与目标正则表达式相匹配的信息，而表达式“.”与 start_station_telecode 是想匹配的。

所以，在本例中，我们不应该完完全全输出匹配文本，我们转换一下思路，要求只输出匹配文本中的一小部分内容，依然以列车车次为例，正则表达式如下：

```
*station_train_code": "([""]+).*
```

最前面的“.”用于匹配 train_no 等 station_train_code 前面的信息，最后面的“.”用于匹配列车车次后面的所有信息，而我们只希望输出圆括号中的信息。

换一种表达思路，只输出圆括号中的信息，可以理解成先圆括号中的信息替换整条信息，然后再输出整条信息。

Duang，替换出场了，sed 终于出场了。

```
sed -r 's/.*station_train_code": "([""]+).*/\1/'
```

输入 man sed，查看 sed 的帮助文档，可知 sed 替换命令的标准格式为：

```
s/regexp/replacement/
```

结合本例实际，即使用“\1”替换整行内容。

那么问题来了，那为什么“\1”可以表示圆括号中的内容呢？

这是正则表达式的另一个知识点：捕获和反向应用。

简单说，在正则表达式中，圆括号用于捕获匹配内容，之后可以使用反斜杠+数字的方式引用该圆括号中的内容。数字从 1 开始，从左往右，以左圆括号出现的顺序依次增加。例如下面这段语句的输出内容是：123456 34。

```
echo "123456" | sed -r 's/(12([34]+)56)\1 \2/'
```

至此，我相信聪明如你的读者一定知道本例的解决方案了。

```
sed -r 's/.*station_train_code": "([""]+).*start_station_name": "([""]+).*end_station_name": "([""]+).*start_time": "([""]+).*arrive_time": "([""]+).*ze_num": "([""]+).*zy_num": "([""]+).*swz_num": "([""]+).*\1 \2 \3 \4 \5 \6 \7 \8/'
```

图 1-22 sed 命令输出订票信息

爱动手的读者读到这里，也许已经迫不及待的想对 fetch_sh-bj.sh 脚本做修改了，同时觉得该脚本输出的内容不够丰富，还应该输出发车站点、到达站点、列车历时时间等信息。

```
sed -r 's/.*station_train_code": "([""]+).*start_station_name": "([""]+).*end_station_name": "([""]+).*from_station_name": "([""]+).*to_station_name": "([""]+).*start_time": "([""]+).*arrive_time": "([""]+).*lishi": "([""]+).*ze_num": "([""]+).*zy_num": "([""]+).*swz_num": "([""]+).*\1 \2 \3 \4 \5 \6 \7 \8 \9 \10 \11/'
```

图 1-23 sed 命令输出订票信息

等我们满怀期待的等待更丰富的订票信息时，我们悲剧的发现结果并不是我们想要的，为什么，到底哪里错了呢？因为 sed 命令的反向引用只能是\1 到\9，不能超过 10，\10 会被解释成\1 家数字 0。

那么问题又来了，怎样才能输出更多，更丰富的订票信息呢？

预知后事如何，请听下文分解。

2. 基础很重要

我们的先哲很早就认识到磨刀不误砍柴工；工欲善其事，必先利其器；马克思哲学教导我们，把人和动物区分开的一个重要标记就是人会使用工具。

本章重点介绍 Web 技术的几个基本概念，为了方便读者理解，更为了方便拥有 Geek 精神的读者动手验证这些基本概念，本章将以实验的方式展开。在实验的过程中，我们将引入几个非常有用的调试工具，这些工具将在后续章节中广泛使用。本章介绍的工具均为 Linux 下的开源工具，有些工具不乏 windows 版本，但是强烈建议 windows 下的读者安装虚拟机，比如 VirtualBox，试验本章的例子。

2.1. 防蹭网

上学时，不管你在哪所学校，校园里总有这样的名言：爱家爱国爱师妹，防火防盗防师兄。在“互联网+”时代，我们是不是应该再增加一个防范项目呢——防蹭网？不让隔壁的谁谁用我们家的 wifi，就算他用网上找的各种 wifi 破解工具黑了家里路由器的密码，也不能让他轻易使用。

首先让我们回顾一下，在打开网页的过程中，浏览器和服务器之间发生了什么？

1. 我们需要在浏览器的地址栏中敲入网页地址
2. 浏览器通过查询 DNS 服务器得到域名所对应的 IP 地址
3. 浏览器链接该 IP 地址，并发送数据请求
4. 服务器接受浏览器的数据请求，并发送数据响应

此处可以有两个问题：

1. 什么是 DNS 服务器？
2. 服务器向浏览器发送数据响应后，他们之间的 TCP 链接是否断开？

本小节先解决第一个问题，第二个问题留由后文解答。

DNS 全称 Domain Name System，域名系统，详细说明可参见维基百科：

中文版：<https://zh.wikipedia.org/zh/域名系统>

For English: https://en.wikipedia.org/wiki/Domain_Name_System

简单说域名就是网址和 IP 地址的对应，www.baidu.com 肯定比 115.239.210.27 这样一串数字更容易记忆，当然如果你想显摆，想 ZhuangB，故意去记 IP 地址，我也没办法。

使用域名而不是直接使用 IP 地址的另一个好处就是服务器的 IP 地址可能改变，但是域名不会变，而且一个域名可以对应多个 IP 地址哦。

在整个上网过程中，浏览器需要频繁的与 DNS 服务器进行交互，用于将网址转换成 IP 地址，可见 DNS 服务器的重要性。让我们脑补一下这样的场景：

第一步：山寨一个新微博的登录页面

第二步：搭建一个自己的 DNS 服务器，把微博网址指向我们的山寨版新浪

第三步：想办法让受害者使用我们的 DNS 服务器

第四步：受害者下次登录微博时，偷偷的盗取他们的用户名和秘密

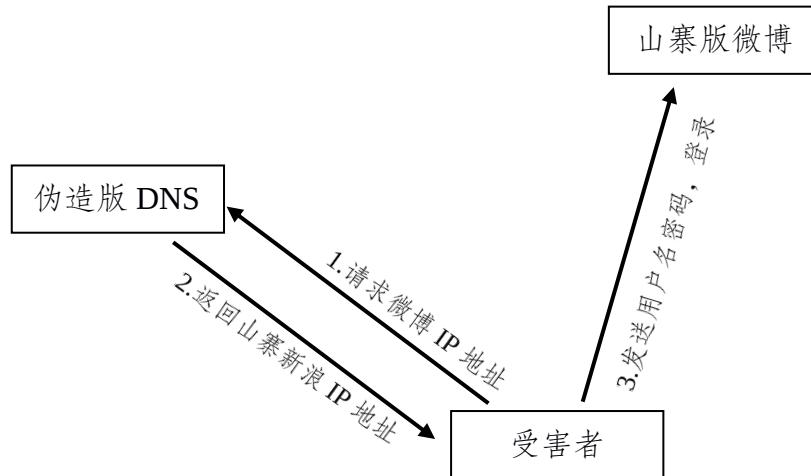


图 2-1 山寨新浪

这个传说中的 DNS 劫持是不是听上去很可怕，而且我相信很多读者都经历过类似的事件，不过不是新浪微博而已。

80 后的读者一定记得，早先年家的宽带上网，不管你输入什么网址，总是先弹出一个全是广告的页面。这就是宽带运营商劫持了你的网页请求，然后先向你推送了一堆的广告页信息。

所以 DNS 服务器是很重要的，万一我们用了坏银的 DNS，万一坏银山寨了淘宝的登录页面，万一他山寨了网银的登录有页面，万一……

好吧，有点迫害妄想症了，但是至少我们意识到 DNS 服务器确实是很重要的，那么作为用户你是否应该知道你正在用哪家的 DNS 服务器呢？

嗯，Linux 用户直接在根目录下全局搜索 resolv.conf 文件，该文件记录 DNS 服务器的 IP 地址。

```
sudo find / -name "resolv.conf"
```

这个是我本机上的 resolv.con 文件内容：

```
nameserver 202.96.209.133  
nameserver 192.168.0.1
```

你是否还记得在 XP 系统上，手动设置 IP 地址时，设置页面上的“首选 DNS 服务器”和“备用 DNS 服务器”，此处两个 nameserver 即为“首选 DNS 服务器”和“备用 DNS 服务器”。

Windows 用户在“开始”菜单点击“运行”，敲入“cmd”，按回车，在弹出的页面输入：**ipconfig /all**，可以查看自己正在用哪家的 DNS 服务器。

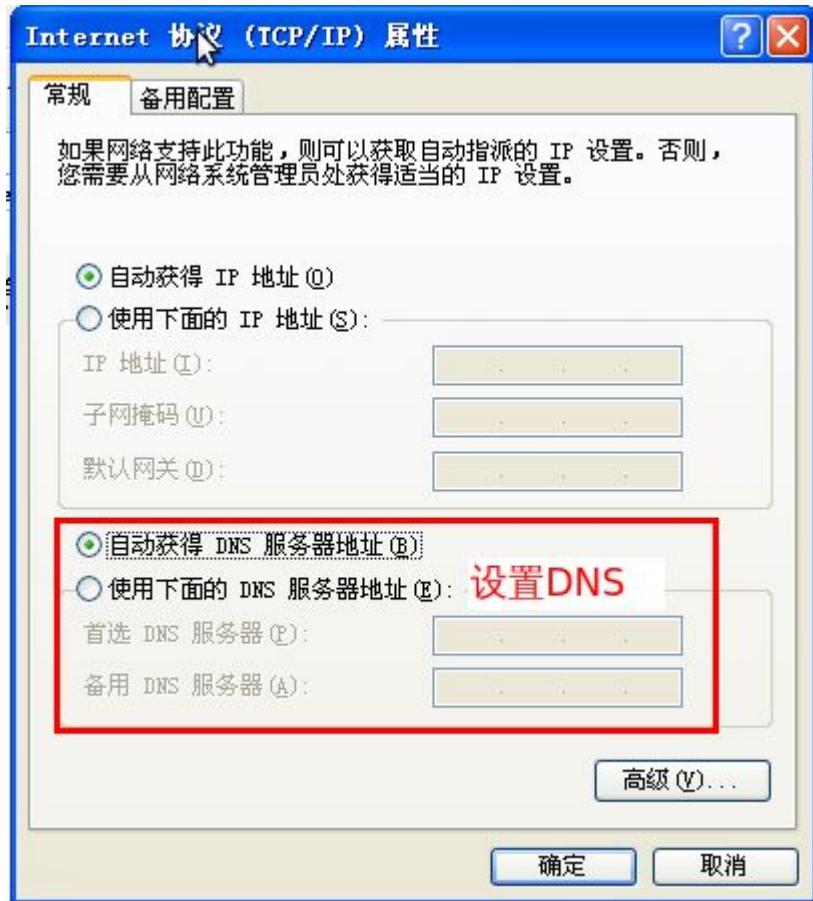


图 2-2 window 设置 DNS 服务器

```
Connection-specific DNS Suffix . . .
Description . . . . .
Physical Address . . . . .
Dhcp Enabled . . . . .
Autoconfiguration Enabled . . .
IP Address . . . . .
Subnet Mask . . . . .
Default Gateway . . . . .
DHCP Server . . . . .
DNS Servers . . . . .
Lease Obtained . . . . .
Lease Expires . . . . .
```

图 2-3 window 获取 DNS 服务器

爱思考的同学也许会问：我电脑是从路由器上自动获取 IP 地址的，从未设置过 DNS 服务器，这个又是从哪来的呢？

嗯，这又是个好问题，答案就是自动获取 IP 地址时，从路由器上获取 DNS 服务器地址的。

从浏览器登录路由器设置界面(路由器一般为 192.168.0.1 或 192.168.1.1), 在“网络参数”->“WAN 口设置”选项卡下，可以找到我们的 DNS 服务器地址。



图 2-4 路由器 DNS 设置

这里的 DNS 服务器是宽带运营商自动分配的，读者可以手动修改为其他 DNS 服务器。Google 提供免费的 DNS 服务，其 IP 地址为 8.8.8 和 8.8.4.4，国内比较常用的有 114 的 DNS 服务，其 IP 地址为 114.114.114.114 和 114.114.115.115。当然 DNS 服务商不仅仅是这两家，读者可以自行搜索。

结束本小节之前，让我们用一个恶作剧回答本小节最初抛出的那个问题吧：不能让隔壁的谁谁轻易使用我们家的 wifi。

第一步：在自己电脑上选择手动设置 DNS 服务器，而不是自动获取。

第二步：在路由器上将 DNS 服务器的 IP 地址指向不存在的 IP 地址，例如我们设置成 192.168.0.200 和 192.168.0.201。

第三步：在路由器上取消 wifi 密码，并把 SSID 名称改为“客官，选我呀!!!”。

好吧，如果你邻居不知道 DNS 服务器的话，我们可以脑补一下他连上你家 wifi 后，依然无法上网的崩溃表情吧。

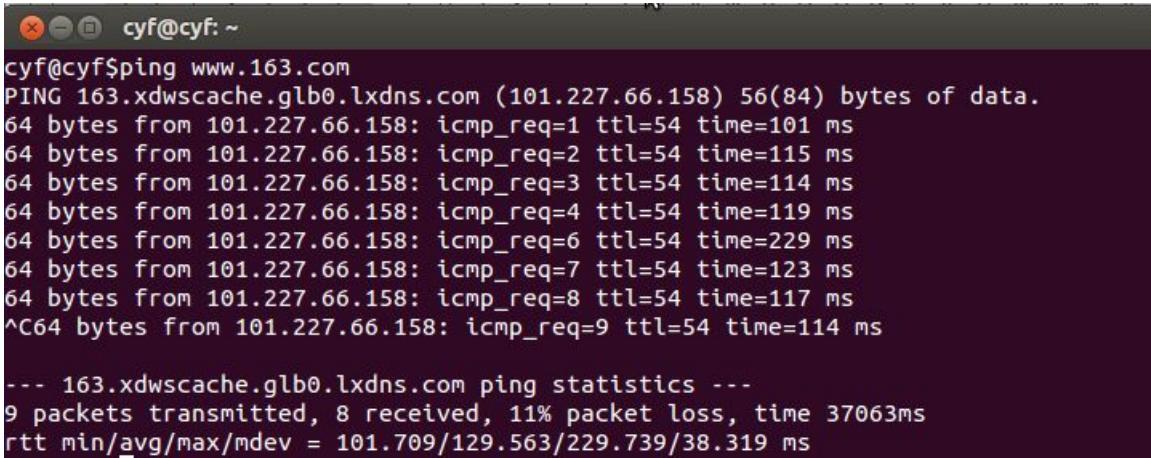
2.2. 花季护航

1987 年 9 月 14 日，北京计算机应用技术研究所发出了中国大陆的第一份电子邮件：“Across the Great Wall we can reach every corner in the world”。

在上一小节中，我们认识了 DNS 服务器，并且知道这货是用来把网址解析成 IP 地址的。但是这个转化一直都是浏览器在做的，对用户来说，并不知道这个 IP 地址确切是多少。

那么作为用户，如果我想知道 163 网站(www.163.com)的 IP 地址是多少，有什么办法显示出来吗？

嗯，办法有很多种。第一种方法，ping 命令。在命令行中直接 ping 目标网站，可以得到其 IP 地址，CTRL+C 结束 ping 命令。



```
cyf@cyf:~$ ping www.163.com
PING 163.xdwscache.glb0.lxdns.com (101.227.66.158) 56(84) bytes of data.
64 bytes from 101.227.66.158: icmp_req=1 ttl=54 time=101 ms
64 bytes from 101.227.66.158: icmp_req=2 ttl=54 time=115 ms
64 bytes from 101.227.66.158: icmp_req=3 ttl=54 time=114 ms
64 bytes from 101.227.66.158: icmp_req=4 ttl=54 time=119 ms
64 bytes from 101.227.66.158: icmp_req=6 ttl=54 time=229 ms
64 bytes from 101.227.66.158: icmp_req=7 ttl=54 time=123 ms
64 bytes from 101.227.66.158: icmp_req=8 ttl=54 time=117 ms
^C64 bytes from 101.227.66.158: icmp_req=9 ttl=54 time=114 ms

--- 163.xdwscache.glb0.lxdns.com ping statistics ---
9 packets transmitted, 8 received, 11% packet loss, time 37063ms
rtt min/avg/max/mdev = 101.709/129.563/229.739/38.319 ms
```

图 2-5 Ping 命令

借助 ping 命令，我们知道 www.163.com 的其中一个 IP 地址是 101.227.66.158。其中一个 IP 地址？？？

难道说 www.163.com 还有别的 IP 地址？？？

嗯，这里我们可以做个简单的实验，使用 Bash 脚本程序，向 www.163.com 发起 10 次 ping，每次 ping 命令只发送一个数据包。

```
1. #!/bin/bash
2. n=0
3. while [ $n -lt 10 ]
4. do
5.   ping www.163.com -c 1 &
6.   ((n=n+1))
7. done
```

图 2-6 ping_163.sh 脚本



```
cyf@cyf:~/Desktop$ ./ping_163.sh
cyf@cyf$ PING 163.xdwscache.glb0.lxdns.com (101.227.66.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (101.227.66.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (114.80.143.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (101.227.66.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (114.80.143.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (114.80.143.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (101.227.66.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (101.227.66.158) 56(84) bytes of data.
PING 163.xdwscache.glb0.lxdns.com (114.80.143.158) 56(84) bytes of data.
```

图 2-7 ping_163 脚本运行结果

使用 ping_163.sh 脚本，可以得到 www.163.com 的另一个 IP 地址 114.80.143.158。ping 命令一般用于测试网络是否联通，例如你在优酷上看视频，看到一半突然不动了，此时你可以使用 **ping www.youku.com** 测试一下网络，判断是不是宽带断了。因

此用 ping 命令获得 IP 地址，应该算是她的副产品，不是主业。

第二种获得 IP 地址的方法 **nslookup** 命令。很多同鞋不喜欢命令行的一个原因是命令太难记了。Linux 下很多命令都是引文单词的缩写，比如我们这个命令，**lookup** 本意即为查找，**ns** 就是单词 **name server** 的首字母。还记得上一小节介绍的 **resolv.conf** 文件吗，这里记录的 DNS 服务器不就是 **name server** 吗。

在命令行中直接敲 **nslook www.163.com** 即可获取 IP 地址。

```
cyf@cyf$nslookup www.163.com
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
www.163.com      canonical name = www.163.com.lxdns.com.
www.163.com.lxdns.com  canonical name = 163.xdwscache.glb0.lxdns.com.
Name: 163.xdwscache.glb0.lxdns.com
Address: 101.227.66.158
Name: 163.xdwscache.glb0.lxdns.com
Address: 114.80.143.158
```

图 2-8 nslookup

Address:127.0.0.1#53 表示从本地缓存中获取 **www.163.com** 的 IP 地址，在后面跟随 DNS 服务器地址，可以指定 nslookup 从该服务器中获取网站的 IP 地址。

```
cyf@cyf$nslookup www.163.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
www.163.com      canonical name = www.163.com.lxdns.com.
www.163.com.lxdns.com  canonical name = 163.xdwscache.glb0.lxdns.com.
Name: 163.xdwscache.glb0.lxdns.com
Address: 101.227.66.158
Name: 163.xdwscache.glb0.lxdns.com
Address: 114.80.143.158
```

图 2-9 nslookup 制定 DNS 服务器

此处，我们指定从 **google** 的 DNS 服务器获网站 **www.163.com** 的 IP 地址，结果和从本地缓存中获得的结果一样。

下面我们一起做一个有趣的实验，我们分别从本地缓存、**google** 的 DNS 和 114 的 DNS 服务器获取 **www.facebook.com** 的 IP 地址，结果如图 2-10 所示。

纳尼，这三个结果居然不一样？？？

同一个网址，从不同的 DNS 服务器获得的 IP 地址居然得到不一样、不一样、不一样（重要的事情要说三遍）IP 地址？也许读者在自己电脑上测试时，得到的 IP 地址与本文给出的 3 个 IP 地址也不一样。细心的读者也许还会发现同一个 DNS 服务器，每次返回 **www.facebook.com** 的 IP 地址都是不一样的，悲剧。

读到这里，相信读者心中一定有这个问题：到底那个 IP 地址是真的，爱问为什么的读者，一定还有另一个问题：为什么会有三个不一样的 IP 地址？

我们先来解答第一个问题：<http://whois.domaintools.com>。打开该网页，输入 IP 地址，即可查询关于该 IP 地址的详细信息。

```
cyf@cyf$nslookup www.facebook.com
Server:      127.0.0.1
Address:     127.0.0.1#53
```

```
Non-authoritative answer:
Name:   www.facebook.com
Address: 243.185.187.39
```

```
cyf@cyf$nslookup www.facebook.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53
```

```
Non-authoritative answer:
Name:   www.facebook.com
Address: 203.98.7.65
```

```
cyf@cyf$nslookup www.facebook.com 114.114.114.114
Server:      114.114.114.114
Address:     114.114.114.114#53
```

```
Non-authoritative answer:
Name:   www.facebook.com
Address: 159.106.121.75
```

图 2-10 nslookup 获取 facebook 的 IP 地址

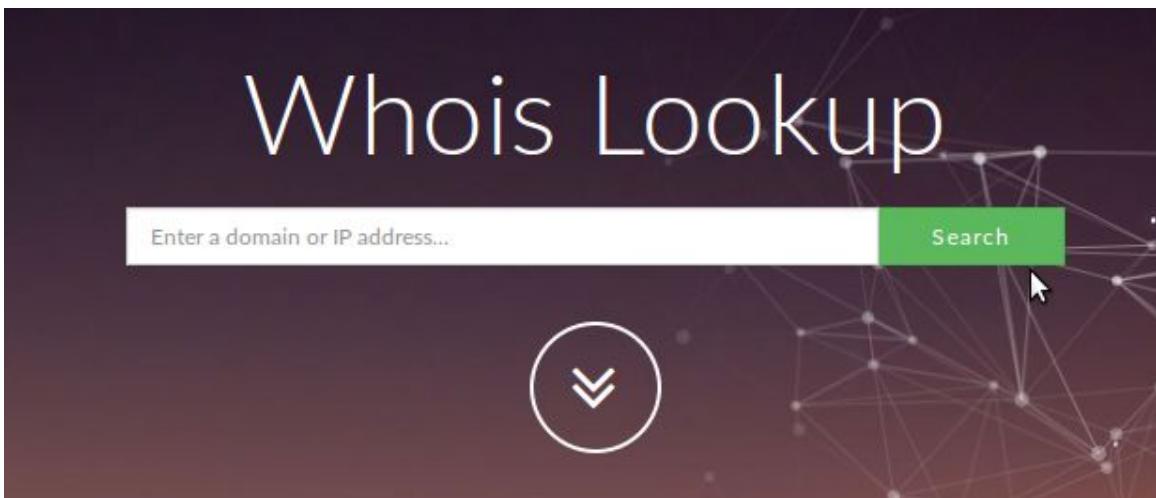


图 2-11 whois lookup

好了，那我们在该网站中依次输入这 3 个 IP 地址，鉴别一下哪个 IP 地址才是 www.facebook.com 的 IP 地址。

243.185.187.39: 240.0.0.0 - 255.255.255.255 是 IANA 机构保留的 IP 地址，不应该被使用，那么这个地址肯定不是 www.facebook.com 的 IP 地址。

203.98.7.65: 这个 IP 地址是归 TelstraClear 公司所有的，该公司是新西兰第三大通信公司，很显然这个地址也不是 www.facebook.com 的 IP 地址。

159.106.121.75: 这个 IP 地址是归 DoD Network Information Center(美国国防网络信息中心)所有的，那么这个地址也不是 www.facebook.com 的 IP 地址。

好了，那么新的问题来了，www.facebook.com 的 IP 地址到底是什么呢？

恩，这个问题有点小复杂，我们需要了解 nslookup 在查询 IP 地址时，DNS 服务器和 nslookup 之间发生了什么。

在上一章节中，我们认识了功能强大的 Firebug，该工具可以把浏览器与网页服务器之间交互的所有数据一览无余的呈现出来。那么是否有个工具可以把经过网口的所有数据呈现出来呢，包括本机程序从网口发出的数据和外界发到本机网口的数据，最好还能按照不同的协议帮我们把数据解析一下，就像 Firebug 帮我解析 Params 一样。

嗯，这种工具是存在的，她就是网络调试神器 Wireshark。(此处应该有掌声!!!)

Wireshark 是一款免费的开源软件，无论你是用 Windows 还是 Linux 或则是 Max OS，她都提供相应的安装包，很贴心吧 G ^_ ^G

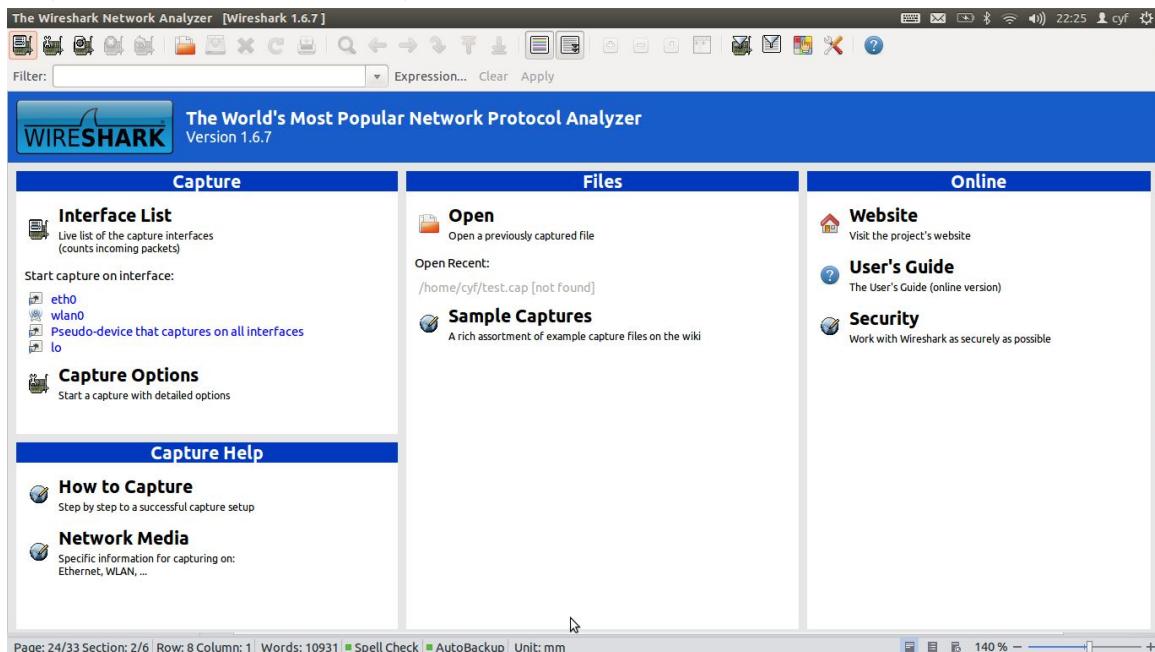


图 2-12 wireshark 启动界面

点击 Wireshark 菜单上的“Capture”，选择“Options”，

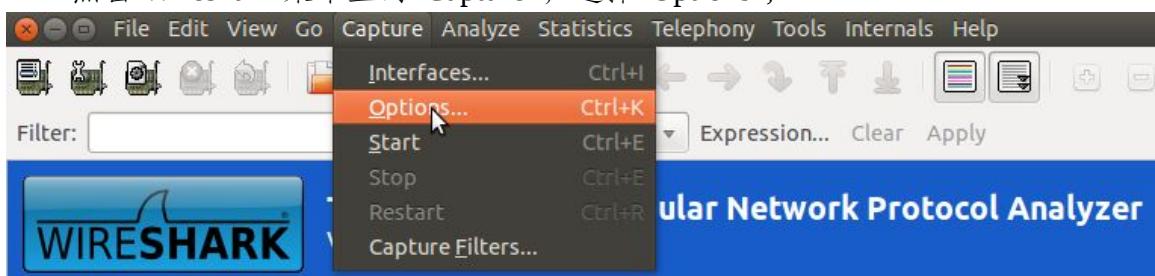


图 2-13 Capture->Options

在弹出的窗口中，Interface 指示你需要捕获哪个网口上的数据，此处我捕获本机无线网卡(wlan0)上的数据。Capture Filter 内输入：**host 8.8.8.8**，表示过滤出与主机 8.8.8.8 通信的数据。如果运行 Wireshark 捕获无线网卡上的数据时，产生无法上网的问题，可以勾掉选项卡“**Capature packets in promiscuous mode**”。点击“start”开

始捕获数据。在命令行中敲入“**nslookup www.facebook.com 8.8.8.8**”，从 google 的 DNS 服务器查询 www.facebook.com 的 IP 地址。

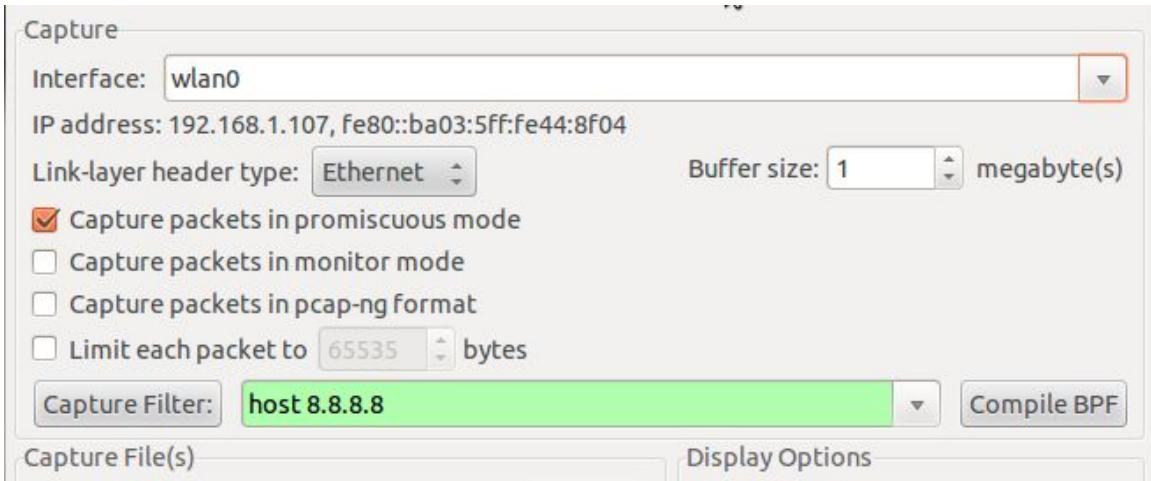


图 2-14 捕获 Roogle DNS 服务器的数据

```
cyf@cyf$nslookup www.facebook.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   www.facebook.com
Address: 78.16.49.15
```

图 2-15 Roogle DNS 查询 www.facebook.com

1 0.000000	192.168.1.107	8.8.8.8	DNS	76 Standard query A www.facebook.com
2 0.006099	8.8.8.8	192.168.1.107	DNS	92 Standard query response A 78.16.49.15
3 0.018205	8.8.8.8	192.168.1.107	DNS	108 Standard query response A 159.106.121.75
4 0.010262	192.168.1.107	8.8.8.8	ICMP	136 Destination unreachable (Port unreachable)
5 0.064021	8.8.8.8	192.168.1.107	DNS	116 Standard query response CNAME star.c10r.facebook.com 31.13.70.1

图 2-16 wireshark 捕获的数据

我们悲剧的发现这次 nslookup 返回的结果与上一次又不一样，囧rz.....。观察 wireshark 捕获的网卡数据中的数据，第 1 个数据包是我们向 8.8.8.8 发送查询请求，第 2 个数据包是 8.8.8.8 给我们返回的查询结果 78.16.49.15，就是 nslookup 显示的结果。但是故事并没有到此结束，还有第 3 个数据包和第 5 个数据包，也是 8.8.8.8 给我们返回的查询。这是什么情况，我们只查询了一次，她居然向我们返回了 3 次查询结果，买一送二啊!!!

在网站 <http://whois.domaintools.com> 上进一步验证可以发现第 3 数据的 IP 地址 159.106.121.73 也不是 www.facebook.com 的，但是我们可以惊喜的发现第 5 个数据包的 IP 地址 31.13.70.1 真的是 www.facebook.com 的 IP 地址。相信读者一定越读越糊涂的，这是怎么回事儿呢？

讲清楚这个需要先介绍一点背景知识。DNS 服务器可以采用 UDP 和 TCP 两种通信模式接受客户端的查询请求，默认模式下 nslookup 采用 UDP 与 DNS 服务器通信，在该模式下，nslookup 发出请求后，就等待 DNS 服务器的返回，而且只读取第一个返回结果。

那 nslookup 如何判断这个结果是 DNS 服务器返回的呢？

依据 IP 地址和端口号。

在 wireshark 中点击第 1 个数据包，在展开信息中可以得到，向 DNS 服务器查询 IP 地址时，本机的使用的端口号是 55628（每次查询时，该端口号随机分配），DNS 服务器的端口号为 53（该端口号固定）。简单总结一下，在向 DNS 服务器发送 IP 地址查询后，nslookup 就一直在等这么一个数据包：从服务器 8.8.8.8 的 53 号端口发送本机 192.168.1.107 的 55628 端口的 UDP 数据包。

因为 UDP 是未建立链接的不可靠传输，所以有时这个数据包不一定是真的 8.8.8.8 服务器 53 号端口发出的，如果有设备横在我们与 8.8.8.8 服务器之间，截获我们的查询请求，然后伪装出一个 8.8.8.8 服务器 53 端口的数据包发给我们，只要这个伪装的数据包比真实的数据包先到，那么就可以对 nslookup 产生干扰作用，让 nslookup 显示一个错误的 IP 地址。

热心的读者还回发现，不仅仅 facebook 返回错误的 IP 地址，twitter 也返回错误的 IP 地址，而且有时这两个网址返回的 IP 地址居然是一样的。

好吧，让我在风中凌乱一会儿。。。

1 0.000000	192.168.1.107	8.8.8.8	DNS	76 Standard query A www.facebook.com
2 0.006099	8.8.8.8	192.168.1.107	DNS	92 Standard query response A 78.16.49.15
3 0.012055	8.8.8.8	192.168.1.107	DNS	108 Standard query response A 159.106.121.75
4 0.012052	192.168.1.107	8.8.8.8	ICMP	136 Destination unreachable (Port unreachable)
5 0.064021	8.8.8.8	192.168.1.107	DNS	116 Standard query response CNAME star.c10r.facebook.com A 31.13.70.1
6 0.064070	192.168.1.107	8.8.8.8	ICMP	144 Destination unreachable (Port unreachable)

▶ Frame 1: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)
▶ Ethernet II, Src: IntelCor_44:8f:04 (08:03:05:44:8f:04), Dst: a8:57:4e:68:92:f0 (a8:57:4e:68:92:f0)
▶ Internet Protocol Version 4, Src: 192.168.1.107 (192.168.1.107), Dst: 8.8.8.8 (8.8.8.8)
▶ User Datagram Protocol, Src Port: 55628 (55628), Dst Port: domain (53)
▶ Domain Name System (query)

图 2-17 本机和 DNS 服务的 UDP 端口号

```
cyf@cyf$nslookup www.twitter.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   www.twitter.com
Address: 159.106.121.75

cyf@cyf$nslookup www.facebook.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   www.facebook.com
Address: 159.106.121.75
```

图 2-18 Facebook 和 Twitter 的 IP 地址

既然 UDP 被干扰，那么我们就用 TCP 查询。添加-vc 选项，强制 nslookup 使用 TCP 协议与 DNS 服务器通信，这时就可以得到正确结果了。

```

cyf@cyf$nslookup -vc www.facebook.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
www.facebook.com      canonical name = star.c10r.facebook.com.
Name:    star.c10r.facebook.com
Address: 31.13.70.1

```

图 2-19 nslookup TCP 协议查询 Facebook

最后来个黑色幽默，关于 Linux/Unix 命令组成的 dirty-joke，还“黑”了一把纽约时报。2010 年 5 月 12 日，纽约时报发表了一片关于 4 个 NYU(New York University) 学生的故事，<http://www.nytimes.com/2010/05/12/nyregion/12about.html>。故事本身未引起轰动，但是细心的读者发现新闻配图上藏了个 Linux 命令组成的 dirty-joke。



图 2-20 linux-dirty-joke

2.3. 断或不断，这是个问题

在“防蹭网”小节，我们抛出了两个问题，前文解决了其中一个问题，还剩一个问题没解决。当服务器向浏览器发送数据响应后，服务器和浏览器之间的 TCP 链接是否断开？

这里我们可以先做一个简单的试验，在浏览器内随意敲入一个网址，比如本文选择网易作为试验对象 (www.163.com)，之后不要对浏览器做任何操作（这点很重要，不能对浏览器做任何操作），在操作系统层面观察浏览器与服务器是否有 TCP

链接，实验步骤如下。

1. 使用 nslookup 获取 www.163.com 的 IP 地址；

```
cyf@cyf$nslookup -vc www.163.com 8.8.8.8
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
www.163.com      canonical name = www.163.com.lxdns.com.
www.163.com.lxdns.com  canonical name = 163.xdwscache.glb0.lxdns.com.
Name: 163.xdwscache.glb0.lxdns.com
Address: 101.227.66.158
Name: 163.xdwscache.glb0.lxdns.com
Address: 114.80.143.158
```

图 2-21 www.163.com

2. 在 bash 脚本程序，使用 netstat 命令循环查询本机的所有网络链接，并使用 grep 命令过滤出 www.163.com 之间的 TCP 链接；

```
1. #!/bin/bash
2. n=1;
3. while [ $n -gt 0 ];
4. do
5.   sudo netstat -anp | grep -E "(114.80.143.158)|(101.227.66.158)";
6.   echo $((n=n+1));
7.   sleep 1;
8. done
```

图 2-22 check_163.sh

3. 在浏览器中打开网页 www.163.com，观察第 2 步输出；

观察可以发现在打开网页的过程中，首先本机与 163 服务器建立一个 TCP 链接，之后在很短的时间内，本机与 163 服务器建立多个 TCP 链接；多个链接保持一段时间之后，这些 TCP 链接逐个断开，最后本机与 163 服务器之间的所有 TCP 链接均断开。

那我们是否可以简单的认为，当服务器向浏览器发送数据响应后，服务器立刻断开与浏览器之间的 TCP 链接呢？

嗯，这个东西可以说立刻断开，也可以说不是立刻断开，我们需要再做几个实验。第二个实验步骤如下。

1. 在浏览器中打开 www.163.com。
2. 待 www.163.com 完全加载后，打开 FireFox 的网页调试工具或 Firebug。
3. 点击 Network 选项卡下的“Reload”按钮，重新加载页面。

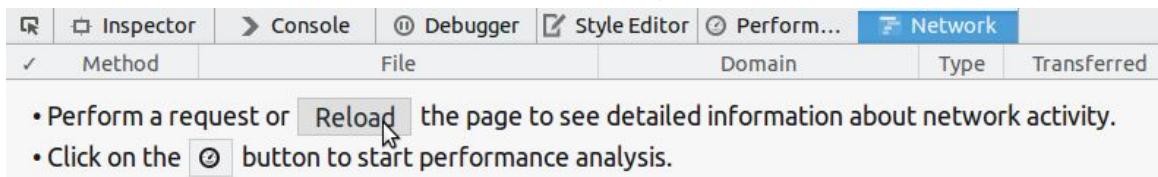


图 2-23 Eeload 163

4. 点击 Network 选项卡下的第一个数据项“**GET / www.163.com**”，观察 Headers 选项卡下的 Request headers，可看到有个 Connection 属性，其值为 “keep-alive”。浏览器正是通过这个 Connection 属性告诉服务器，“数据传完之后，建议先不要断开咱俩间的 TCP 链接，我还有用”。

为什么说是“建议先不要断开 TCP 链接呢”？因为是否保持这个 TCP 链接的决定权在服务器，即使你建议保持该 TCP 链接，服务器有可能也会因为资源紧张等原因断开该 TCP 链接。

Host: "www.163.com"
User-Agent: "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0"
Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
Accept-Language: "en-US,en;q=0.5"
Accept-Encoding: "gzip, deflate"
DNT: "1"
Cookie: "P_INFO= 1439466492 2 ... s_n_f_l_n3=c57701a6a5adc4c8143
Connection: "keep-alive"
Cache-Control: "max-age=0"

图 2-24 keep-alive

既然 Connection 可以设置成“keep-alive”，那么是否还可以设置成别的值？

嗯，是的，Connection 还可以设置成“close”。这个值相当于浏览器告诉服务器，“数据传完之后，可以直接把这个 TCP 链接断开，哥不用了”。此处，引入本小节的第三个实验。在该实验中，我们编写 python 脚本 connection-test.py，模拟浏览器向 www.163.com 请求数据，并且分别设置 Connection 为 close 和 keep-alive，观察脚本输出响应。

在 connection-test.py 脚本中，设置 socket 为非阻塞模式(**socket.setblocking(0)**)，在该模式下，如果程序与服务器间的 TCP 链接断开，则 **socket.recv** 函数返回 0；如果服务器数据已传输结束，但是与程序间的 TCP 链接未断开，则 **socket.recv** 将抛出 **errno.EAGAIN** 或 **errno.EWOULDBLOCK** 异常。

```

1. #!/usr/bin/env python
2. import socket
3. import errno
4. import sys
5. import time
6.
7. HOST='www.163.com'
8. PORT=80
9. s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
10. s.connect((HOST,PORT))
11. s.setblocking(0)
12. s.sendall('GET / HTTP/1.1\r\n')
13. s.sendall('Host: '+ HOST +'\r\n')
14. s.sendall('User-Agent: Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101
   Firefox/38.0\r\n')
15. s.sendall('Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.

```

```

8\r\n')
16. s.sendall('Accept-Language: en-US,en;q=0.5\r\n')
17. s.sendall('Accept-Encoding: gzip,deflate\r\n')
18. s.sendall('Connection: close\r\n')
19. #s.sendall('Connection: keep-alive\r\n')
20. s.sendall('\r\n')
21. length=0
22. previous=0
23. while True:
24.     try:
25.         msg=s.recv(1024)
26.         if msg.__len__()==0: break
27.         length = length+msg.__len__()
28.         print 'read %d bytes' % length
29.     except socket.error, e:
30.         err = e.args[0]
31.         if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
32.             if length==previous:
33.                 sys.stdout.write('#')
34.                 sys.stdout.flush()
35.             previous=length
36.             time.sleep(1)
37.             continue
38.         else: sys.exit(1)
39.     print 'press Enter to return'
40.     sys.stdin.readline()
41. s.close()

```

图 2-25 connection-test.py

当 Connection 的值为 close 时，connection-test.py 输出结果类似酱紫滴：“press Enter to return”与“read xxx bytes”之间没有别的数据。因为数据传输结束后，服务器立刻断开 TCP 链接，导致 socket.recv 的返回值变为 0，使得程序跳出 **while True** 循环，进而直接打印“press Enter to return”。

```

read 179324 bytes
read 180348 bytes
read 180748 bytes
press Enter to return

```

图 2-26 connection-test.py - close

但是当 Connection 的值为 keep-alive 时，输出结果很可能是酱紫：“press Enter to return”与“read xxx bytes”之间有若干个“#”（读者在自己电脑上做试验，可能得到与 close 相同的结果，因为 keep-alive 只是建议，服务器可能因为资源紧张等原因而直接断开此 TCP 链接，此时建议读者换个网站做实验，比如换新浪网）。因为数据传输结束后，“keep-alive”设置使得服务器并未立刻断开该 TCP 链接，而是保持若干时间。在这段时间内，服务器并未向客户端发送任何数据，因此 socket.recv 抛出异常，

程序进入的异常处理模块，打印“#”。之后服务器因为在该 TCP 链接上一直未收到任何数据，所以触发超时机制断开该 TCP 链接，随后 `socket.recv` 的返回值变为 0，程序打印“press Enter to return”。

```
read 178692 bytes
read 179716 bytes
read 180740 bytes
read 180854 bytes
#####
##### press Enter to return
```

图 2-27 connection-test.py - keep-alive

关于 HTTP 1.1 协议中的“断或不断”这个问题，我们简单总结一下：

1. 客户端向服务端发送数据请求时，可以在请求的 `headers` 区域发送设置 `Connection` 为 `close` 或 `keep-alive`，建议服务器在数据传输结束之后断开该 TCP 链接或继续保留该 TCP 链接一段时间，但是决定权在服务端。
2. 如果客户端建议的 `keep-alive` 生效，则客户端与服务器间的 TCP 链接将保持一段时间，如果规定的时间内，客户端一直未向服务器发送任何数据，则服务器触发超时机制，断开该 TCP 链接。
3. 不管 `keep-alive` 是否生效，最终浏览器与服务器之间的 TCP 链接是断开的，因此 HTTP 协议是无链接的（`connectionless`）。

现在让我们回顾一下本小节的标题，“断或不断，这是个问题”。我相信很多读者心中一定有这么个疑问，为什么 TCP 链接断开或不断开会成为一个问题呢？

在 HTTP 1.0 版中，是没有断或不断这种问题的，服务端在结束对浏览器的数据请求响应之后，直接断开与之建立的 TCP 链接。如果浏览器需要请求额外数据，则需要重新与服务器建立链接。因此如果浏览器需要发送 100 数据请求，那么她需要与服务器建立 100 次 TCP 链接。

有没有这觉这样子很烦人？

嗯，确实有点烦人，但当时的互联网绝大部分都还是静态网页，所以也很少出现一个网页需要发送 100 次请求的情况，而且这种设计不仅使得实现层面上简单化，可以保证服务端的资源不被浪费，服务器端不会出现空闲的 TCP 链接。

后来网页越做越复杂，打开单个网页需要发送的请求数目也越来越多（读者可以使用 Firefox 开发者工具自行测试），此时如果每个请求都需要重新建立 TCP 链接，显然是费时费力的浪费行为，于是 `keep-alive` 出现了。不过 HTTP 1.0 与 HTTP 1.1 之间的区别不仅仅在于 `keep-alive` 的问题，感兴趣的读者可详见 RFC 2616, Section 19.6.1：“Changes from HTTP/1.0”。

在结束本小节之前，我们来点关于技术层面的讨论。在 HTTP 1.0 中，所有 TCP 链接在完成数据响应之后立刻断开。那么浏览器在实现层面上可以根据当前 TCP 链接是否断开判断服务器是否完成数据传输。现在到了 HTTP 1.1，服务端可以选择暂时不断开 TCP 链接，那么浏览器是如何判断来自服务器的数据数据是否传输结束了呢？

关于这个问题，HTTP 1.1 协议在设计时已经考虑到了，在 RFC 2616 中一共给出了 5 种方法，详见 <http://tools.ietf.org/html/rfc2616#section-4.4>，本文介绍其中一种比较常用的方法，读者在工作如果碰到需要自行设计通信协议，可以此为参考，毕竟该方法已经在 HTTP 协议上使用了，具备简单、可靠的特点。本小节部分内容参考维基百科：https://en.wikipedia.org/wiki/Chunked_transfer_encoding。

随意打开某网页，并用 Firefox 的开发者工具显示网页数据，在数据响应的 headers 区域寻找“Transfer-Encoding: chunked”项，如果没找到，则换个网页继续寻找，本文依然以网易首页（www.163.com）为例。

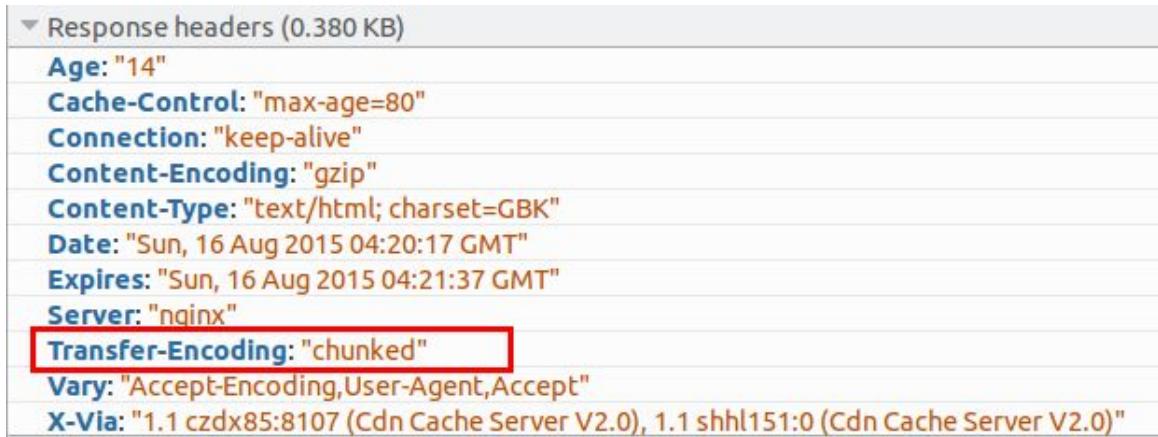


图 2-28 connection-test.py - keep-alive

“Transfer-Encoding: chunked”告诉浏览器数据区内容（message body）是分块的。在 HTTP 协议中，用一个空行（\r\n）分割数据头（message headers）和数据区内容，我们之前介绍的 User-Agent、Connection 以及此处介绍的 Transfer-Encoding 均属于数据头。

为了更好的说明问题，我们需要看看传说中的数据块到底长什么样子，用 curl 命令下载 www.163.com 页面，并以纯数据的方式存储。下载时，必须确保 curl 输出的响应头信息中包含“Transfer-Encoding: chunked”，否则更换下载网站。

```
1. #!/bin/bash
2. curl -iv --raw -H "Accept-Encoding: gzip deflate" \
3.       -H "User-Agent: Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko
   /20100101 Firefox/38.0" \
4.       -H "Accept: text/html,application/xhtml+xml,application/xml;q=0.
   9,*/*;q=0.8" \
5.       -H "Referer: http://www.163.com" \
6.       http://www.163.com -o 163.raw
```

图 2-29 curl-raw.sh

```
< HTTP/1.1 200 OK
< Expires: Sun, 16 Aug 2015 04:55:39 GMT
< Date: Sun, 16 Aug 2015 04:54:19 GMT
< Server: nginx
< Content-Type: text/html; charset=GBK
< Transfer-Encoding: chunked
< Vary: Accept-Encoding,User-Agent,Accept
< Cache-Control: max-age=80
< Content-Encoding: gzip
< Age: 32
< X-Via: 1.1 czdx85:8106 (Cdn Cache Server V2.0), 1.1 shhl151:0 (Cdn Cache Server V2.0)
< Connection: keep-alive
```

图 2-30 curl Transfer-encoding“chunked”

使用 16 进制编辑器打开下载的纯数据文件，定位到数据头与数据区内容的交界处。数据头中每一项内容都是以“`\r\n`”结尾的，而数据头与数据区内容的分割处是一个空行，那么数据头与数据区内容的分割点一定是第一次连续出现`\r\n\r\n`的地方，即十六进制“`0x0D 0x0A 0x0D 0x0A`”。

数据区内容又分成多个数据块，每个数据块由数据块长度和数据块内容组成，数据块长度和数据块内容均以`\r\n`结尾。

数据块长度以字符形式存储，以十六进制方式表示数据块内容的长度，例如本例中数据块 1 的长的为“`A`”（`0x61`），十六进制 `A` 代表的数据为 `10`，所以数据块 1 的长为 `10`，即数据快 1 的内容包含 `10` 个字符。紧挨着数据块 1 内容的是`\r\n`，然后是数据块 2 的长度，依次类推，直到遇见某个数据快的长度为 `0` 时，则代表所有数据块结束，因此 `chunked` 格式中，最后一个数据块一定是这样的：“`0\r\n\r\n`”。

数据块 1 长度(十六进制)	<code>\r\n</code>	数据块 1 内容	<code>\r\n</code>
数据块 2 长度(十六进制)	<code>\r\n</code>	数据块 2 内容	<code>\r\n</code>
数据块 n 长度(十六进制)	<code>\r\n</code>	数据块 n 内容	<code>\r\n</code>
数据块长度为 0 (十六进制)	<code>\r\n\r\n</code>		

图 2-31 chunked 数据块结构

000000150	30 20 28 43 64 6E 20 43 61 63 68 65 20 53	0 (Cdn Cache S
00000015e	65 [数据头与数据区内容分割点] 2E 3 [数据快1长度] 43	erver V2.0)..C
00000016c	6F 6E 6E 65 63 74 69 6F 6E 20 20 6B C0 65	onnection: kee
00000017a	70 2D 61 6C 69 76 65 0D 0A 0D 0A [61] 0D 0A	p-alive....a..
000000188	1F 8B 08 00 00 00 00 00 00 03 0D 0A [36 30] 60
000000196	[30 30] 0D 0A EC BD FB 9F 64 C3 71 27 FA B3	00.....d.q'..
0000001a4	F8 2E 8E BB A [数据快1内容] EA AE 67 3F 99 D1	.+...`....q?..
0000001b2	02 33 D8 78 23 EC 95 50 5D EF DA FE F4 A7	3.x%..X].....
0000001c0	5E DD 5D 33 D3 5D A5 AA EA E9 69 7A F9 63	^.]3.]....iz.c
0000001ce	10 D7 96 2C C9 B2 [数据快2长度] 08 09 D0 80	...,.....1....
0000001dc	06 90 80 91 C0 2B [D6] D6 FA CA BB+.....
0000001ea	6B 5F BF B4 F6 CA 9F FB 8D 47 E6 89 CC 73	k.....G...s
0000001f8	4E F5 63 06 AE AF 3F 9E 9E 99 AE 3A 27 33	N.c...?....:'3
000000206	32 32 32 22 32 32 32 32 32 B9 EF 17 2E FC	222"22222....
000000214	CA 83 8F FE DB 5F BD 98 6C 4F 77 06 E7 EF_..low...
000000222	BA 8F 7E 25 FD EE B9 B9 C9 E8 E3 FD D6 EE	...~%.....

图 2-32 curl Transfer-: ncoding“chunked”

最后我们一起来看看维基百科上所给出的例子，如果某次传输，数据区内容如下。

1. *4|r|n*
 2. *Wiki|r|n*
 3. *5|r|n*
 4. *pedia|r|n*
 5. *e|r|n*
 6. *in|r|n|r|nchunks.|r|n*
 7. *0|r|n*
 8. *|r|n*

图 2-33 chunked 的数据区内容

最后从 chunked 中解出来的数据区内容是这样的。

1. Wikipedia in
 - 2.
 3. chunks.

图 2-34 数据区内容

2.4. 水军

在上一小节中，我们介绍了 HTTP 协议的一个特点--无链接，即使浏览器发送 keep-alive 选项，服务器也会断开长时间无数据的空闲 TCP 链接。无链接导致的直接后果就是无状态（stateless）。

什么叫无状态呢？

就是对服务器来说，当前的响应只与当前的请求有关，与之前的任何操作无任何关系。再简单点，对同一个网址，你第 1 次打开和第 2 或者第 N 次的结果都是一样的。

爱思考的读者也许会说，“不对呀，一年前我在网易上看新闻输入的是 www.163.com，现在我在网易上看新闻还是输入这个地址，但是一年前的内容和现在的内容

肯定不一样，别说一年前的，就是昨天和今天的都不一样”。

嗯，这确实是个好论据，关于这个问题，我们可以这么理解，www.163.com 这个网址时刻向用户返回网易主页上最新的新闻，所以一年前得到的是一年前当时最新的新闻，现在输入得到的是当前最新的新闻。

聪明的读者也许还会问，“关于 163 页面，这么解释似乎还过得去，但是我在电脑打开微博页面 www.weibo.com，第一次登录时，需要输入用户名、密码，只要不是用户主动退出，下次打开 www.weibo.com 时，直接进入我的页面，不需要再次输入用户名、密码；但是如果用户主动退出了，下次登录时还是需要输入用户名和密码的。那么这该如何解释呢，相同的请求，不同的响应结果？”

嗯，这确实是一个很有力的论据，而且不能像 163 那么回答了。

在回答该问题之前，我们必须先明确一点，HTTP 协议是无链接、无状态的协议，在 HTTP 协议中，对于相同的请求，一定是返回相同响应的。

那为什么对于 www.weibo.com，会有不同的响应呢？

唯一合理的解释就是，浏览器向服务器发出的数据请求不一样。

第一次向 www.weibo.com 请求数据时，当用户输入用户名及密码完成认证之后，服务器程序在用户的浏览器端存储了一些数据，表明该用户已经完成认证了；第二次向 www.weibo.com 请求数据时，浏览器只需将这些数据发送给服务器，而无需重新输入用户名及密码。

那服务器程序在用户的浏览器端存储的是什么数据呢？

Duang，她就是传说中的 cookies！！！

爱思考的读者也许会问，既然是 cookies 的原因，那么如果我把 cookies 删了，然后重新请求 www.weibo.com，是不是又需要输入重新输入用户名及密码了呢？

嗯，答案是肯定的，我们再来看来看一个试验。

在 Firefox 地址栏内输入 `about:preferences`，选择 **Privacy**，点击 **Show Cookies**，在 **Search** 栏内输入 **weibo**，查找所有和 weibo 相关的 cookies，最后点击 **Remove All**，删除所有与 weibo 相关的 cookie。非 Firefox 用户可自行搜索删除 cookie 的方法。

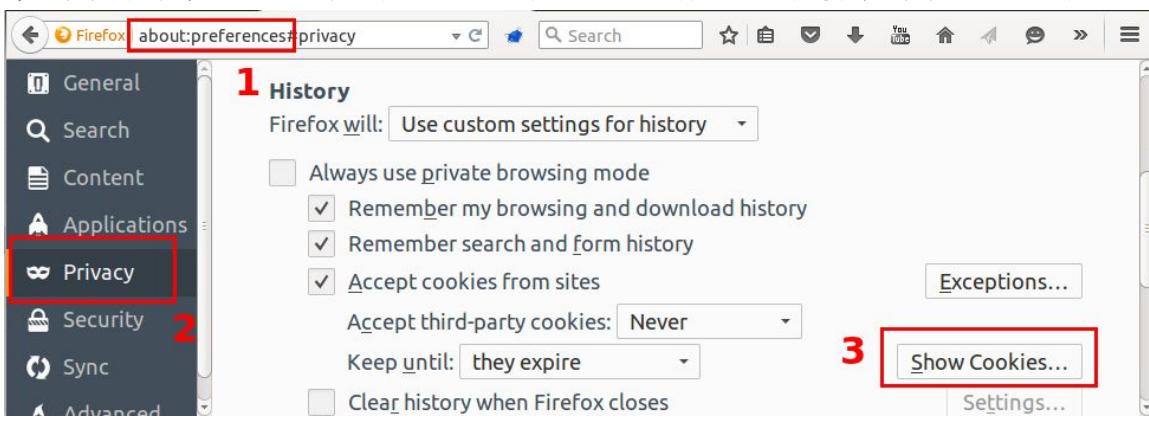


图 2-35 preference-cookies

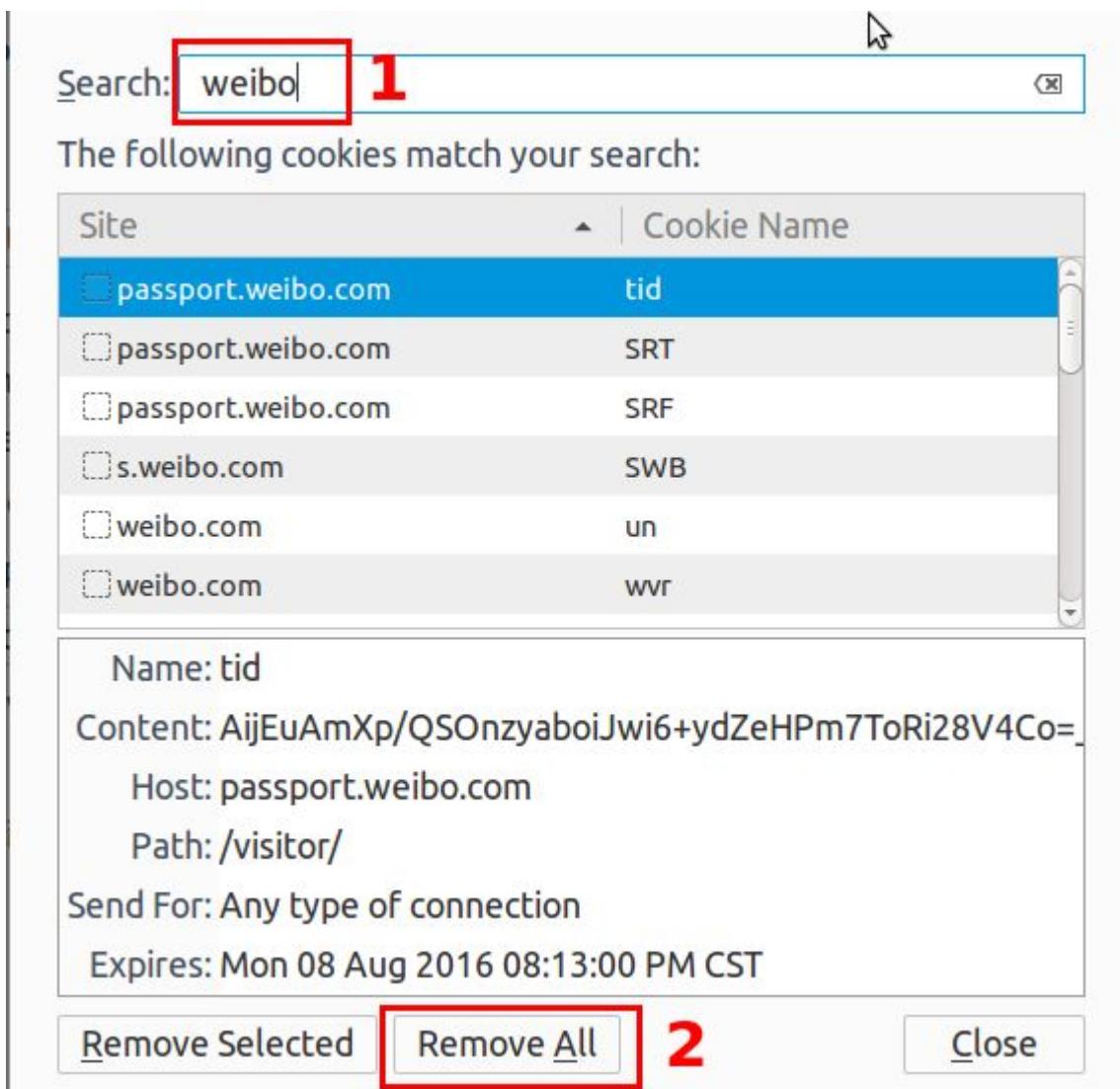


图 2-36 删除 cookies

删完 cookies 后，再重新打开 www.weibo.com，有没有感觉一切都变了，变得不那么熟悉了，你还是原先那个你，只是她已经不是刚才的那个她了。

好吧，文艺青年不好当，总之一句话，你需要再次输入用户名和密码了。

结合本小节标题“**水军**”，也许你已经猜到我下面要讲什么了。

真是 cookies 在手，天下我有，有了 cookies 就不需要用户名密码了，那我们能不能利用 cookies 实现微博的自动评论。

嗯，下面我们就这么干。

第一步：在浏览器里选择一条你想评论微博，内容随意填写，但是先不点击“评论”按钮。

第二步：打开 Firefox 的开发者工具，切换到 **Network** 选项卡下，并清除当前内容。

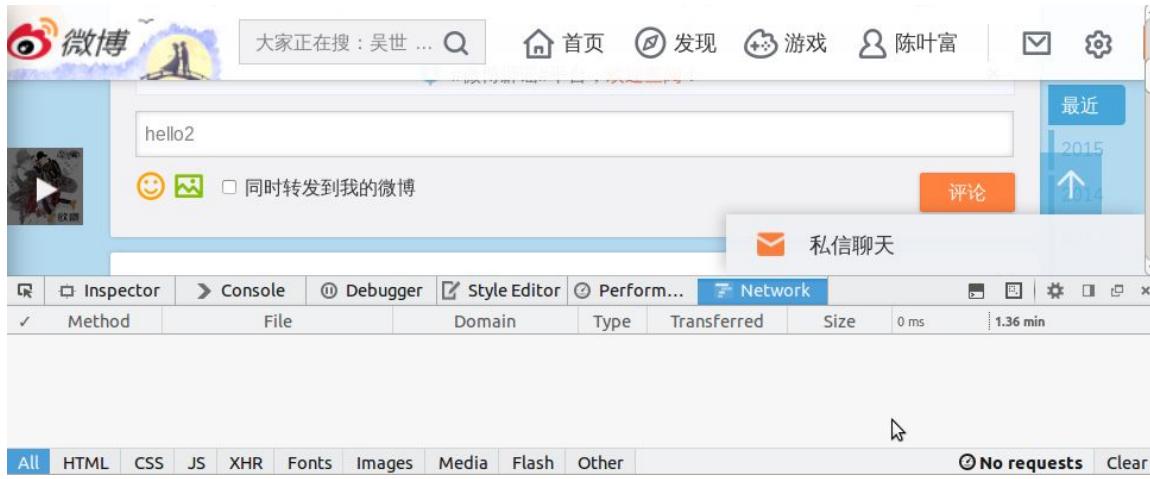


图 2-37 微博评论

第三步：点击“评论”，在开发者工具中捕获 **POST** 方法，查看“评论”的 URL 及参数。

Headers	Cookies	Params	Response	Timings
Request URL: http://weibo.com/aj/v6/comment/add?ajwvr=6&_rnd=1440078318260				
Request method: POST				
Status code: 200 OK				
<input type="button" value="Edit and Resend"/> <input type="button" value="Raw headers"/> <input type="text" value="Filter headers"/>				
▶ Response headers (0.348 KB) ▼ Request headers (1.744 KB)				
Host: "weibo.com" User-Agent: "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0" Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" Accept-Language: "en-US,en;q=0.5" Accept-Encoding: "gzip, deflate" DNT: "1" Content-Type: "application/x-www-form-urlencoded; charset=UTF-8" X-Requested-With: "XMLHttpRequest" Referer: "http://weibo.com/u/1903401211" Content-Length: "165" Cookie: "SUBP=0033WrSXqPxfM725Ws9jqgMF55529P9D9...ge-G0=c47452adc667e76a7435512bb2f774f3" Connection: "keep-alive" Pragma: "no-cache" Cache-Control: "no-cache"				

图 2-38 微博评论的 Headers 及 Request Headers

对同一条微博多次发送评论，测试可以发现每次“评论”时，Headers 选项卡中只有 URL 的 `_rnd` 的值在变，其余均保持不变。翻阅页面的 Javascript 代码，可以发现 `_rnd` 的值来自函数 `(new Date).valueOf()`，该值为时间量，表示 1970 年 1 月 1 日 0 时 0 分 0 秒到当前时刻的毫秒计数，所以每次都在变。

Params 选项卡下，Form data 中，`content` 即为评论内容，其余信息在同一条微博评论中均保持不变。

图 2-39 微博“评论”的参数

第四步：在 Firefox 中安装 Export Cookies 插件，导出浏览器的 cookies，并另存为文件“firefox-cookies.txt”。

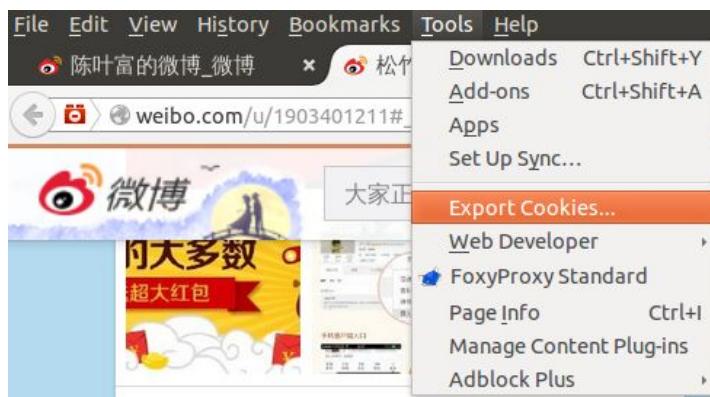


图 2-40 导出 cookies

第五步：编写脚本程序加载 firefox-cookies.txt，参照 Header 内容，模拟浏览器发送微博评论。

1. `#!/bin/bash`
2. `COOKIE_FILE=$PWD/firefox-cookies.txt`
3. `CONTENT=test`
4. `POST_DATA="act=post&mid=3868401644647971&uid=2946645580&forward=0&isroot=0&content=$CONTENT&location=page_100505_home&module=scommalist&group_source=&pdetail=1005051903401211&_t=0"`
5. `USER_AGENT="Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0"`
6. `RND=`date +%s%3N``
- 7.
8. `wget -S -d --user-agent="$USER_AGENT" --keep-session-cookies --header="Referer: http://weibo.com/u/1903401211" \`
- 9.

```
10. --header="DNT: 1" \
11. --header="Pragma: no-cache" \
12. --header="X-Requested-With: XMLHttpRequest" \
13. --load-cookies=$COOKIE_FILE \
14. --post-data="$POST_DATA" \
15. "http://weibo.com/aj/v6/comment/add?ajwvr=6&__rnd=$RND" \
16. -O -
```

图 2-41 weibo-autocomment.sh

最后一步：刷新页面，检查微博评论。



图 2-42 微博自动评论

2.5. 在网上裸奔

本小节，我们谈谈和隐私相关的东东，首先我们来看个小故事。

媳妇生日快要到了，N周前就说要礼物，又说了，送花什么的太俗气了，要有创意，要有新意，要有 Surprise。

眼看日期一天天的临近，你又想不出什么好的点子，心灰意冷之下，你决定上淘宝逛逛，也许会碰到让你眼前一亮的东东。于是你随意的在淘宝上浏览了几款女式包包。

逛淘宝逛的无聊了，你又决定打开了微博看段子。

这时你发现微博页面的广告居然是女式包包，你感觉微博似乎知道你在淘宝上浏览过女式包包。

细细一想，你会觉得这是件很恐怖的事情，似乎你在网上的一举一动都被人监视着，它知道你浏览过这个商品，于是它推测你应该对这个这一类商品感兴趣，于是它很“贴心”的向你推荐这这类商品。

说得更贴切一点，互联网上，我们一直都是赤裸得在一路狂奔。

现在，让我们从技术角度分析一下，微博是如何知道你在淘宝上浏览过女式包包的，进而看看有什么办法可以不让微博知道这些信息。

首先我们可以明确一点，微博之所以知道你在淘宝上浏览过女式包包，绝不是因为什么阿里巴巴集团把他的用户数据共享给了新浪集团，原因很简单，用户数据对企业来说是至关重要的，阿里巴巴绝不可能轻易把这种核心数据共享给新浪。

那么比较合理的解释应该是这样的，我们在逛淘宝的时候，浏览器上留下了一

点点痕迹，类似那种到此一游的痕迹；之后在逛微博时，正是这种痕迹让微博可以向我们展示针对性的广告。

那我们浏览网页时会在电脑上留下什么记录呢？

聪明的你一定猜到了，没错就是 **cookies**。

好了，现在让我们来捋一捋，猜猜微博是如何投放针对性广告的。

1. 首先我们得逛一逛淘宝。
2. 然后呢，淘宝会向你的浏览器发送 **cookies**。
3. 之后呢，我们上微博看段子。
4. 微博呢，就乘机向我们发送淘宝的广告链接。
5. 浏览器只知道这是一个淘宝的链接，于是呢就又向淘宝发送数据请求，顺道把上次逛淘宝留下来的 **cookies** 也发出去了。
6. 淘宝呢，根据你的发给她的 **cookies**，自然知道你之前在淘宝上看过哪些商品，也是呢，就向你返回了针对性的广告。
7. 于是呢，就看上去像似微博在向你投放针对性广告，人家也许是无辜的。

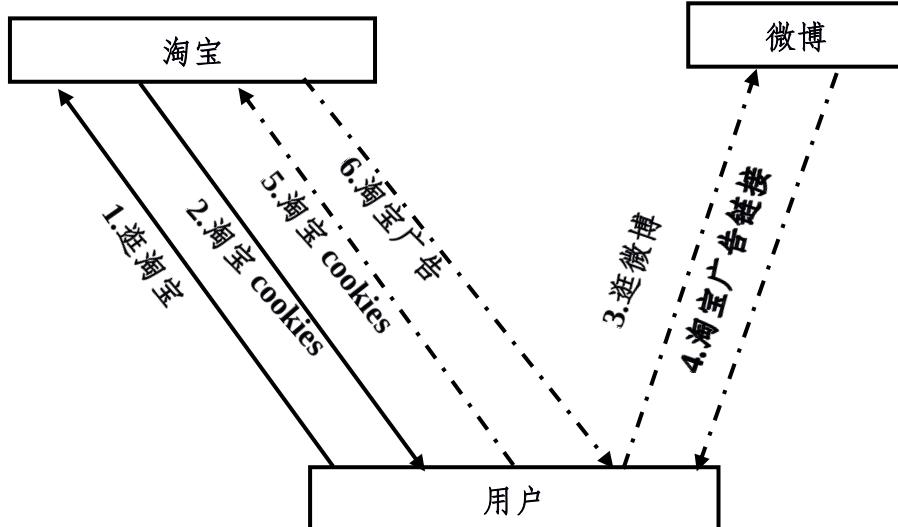


图 2-43 针对性广告

正所谓实践是检验真理的唯一标准，现在让我们一起来验证一下，看看我们前面的想法是否正确。

第一步：当然是登录淘宝，随意的点击某类商品。

第二步：让我登录微博，并打开 Firefox 的开发者工具，看看微博是不是利用了淘宝的 **cookies**。

虽然加载微博时，开发者工具罗列了一大堆信息，但是眼尖的你一定能够注意到这么一条发到 **login.taobao.com** 的请求信息。

没错，这条信息是发到淘宝登录页面，再看看这条微博的 **cookies**，在这里面你甚至可以找到自己的淘宝帐号用户名。还记得在“**水军**”那小节的内容吗？很多时候我们并不需要用户名和密码，只要有 **cookies** 就够了。当微博把你的 **cookies** 发给淘

宝时，淘宝自家的服务器肯定知道你之前在淘宝逛了哪些页面，看过哪些商品等一切你的淘宝上的操作信息，这时候从淘宝页面返回的广告信息就很有针对性了。



图 2-44 针对性广告

Request cookies	
cc	"UIHiLt3xSw=="
lg	"Ug=="
nk	"cyefu"
_tb_token_	"5bed31e371aee"
cna	"PaVgDmewz1UCAXTikBCQJ+eP"
cookie1	"VAmqMb2dlEbTewjJpjdeGeOGr6V/9hHhJbyvz+cArQY="
cookie17	"Uojcj33StcuO8g=="
cookie2	"1c62096e5d21c9ae2a0601ecacd551df"
existShop	"MTQ0MDc3MDE5NA=="
isg	"BF22EE73FBE3BF675D9492C01E4414D9"
l	"Ai8v-g/JnwWHV/eaMexuLv1Dn0k524Py"
lc	"Vy0Rpj1QCrW/Ug=="
lgc	"cyefu"
lid	"cyefu"
mt	"ci=0_1"
sg	"u59"
t	"d5042ddd93ad8d770539d643214d914e"
tg	"0"
thw	"cn"
tracknick	"cyefu"
uc1	"cookie14=UoWzWcOCEYB0sQ==&existSho...=1&cookie15=UIHiLt3xD8xYT...=&pas=0"
uc3	"nk2=AGNXVpQ=&id2=Uojcj33StcuO8g==&...XFFiBNkmxFk=&lg2=U+GCWk/75gdr5Q=="
unb	"1984071415"
v	"0"
whl	"-1&0&0&0"
x	"e=1&p=*&s=0&c=0&f=0&g=0&t=0"

图 2-45 淘宝的 cookies

所谓知己知彼，百战百胜。在我们理解了这种针对性广告的实现原理后，我们有什么办法可以禁掉这座针对性的广告呢？至少我不喜欢这种广告，有一种被人监视的感觉，让人很不舒服。

那么如何禁掉这种针对性的广告呢。

方法很简单，只要微博向 login.taobao.com 发送请求信息时，不附上 cookies 就 OK 了。那如何不附上 cookies 呢？

方法一，就是删除 cookies。当然我们肯定不能手动删除，否则太二了。Firefox 提供专干这类的插件，**Self-destructing Cookies** 就是这类插件中的一个代表，只要你把页面关了，和这个页面相关的 cookies 就全被干掉。当然这个方法的副作用就是，你每次登录微博都得重新输入用户名和密码。

嗯，每次都输入用户名和密码确实很烦人，作为用户的我们觉得烦，**Self-destr**

ucting Cookies 的作者也觉得挺烦的，所以这个插件可以设置白名单，在白名单内的网站不删除 cookies，其它的通通干掉。



图 2-46 淘宝的 cookies

方法二，我们不删除 cookies，我们仅仅只是让微博不能发送淘宝的 cookies。淘宝的 cookies 应该归淘宝所有，对微博来说，淘宝的 cookies 是一个专用的名词——第三方 cookies (third-party-cookies)。那么只需要设置浏览器不接受第三方 cookies 就可以实现我们想要的功能。在 Firefox 的 Preferences 中，选择 Privacy 选项卡，设置 **Accept third-party cookies** 的值为 **Never**。此时，再重新用开发者工具观察，可以发现虽然微博依旧向 **login.taobao.com** 发送请求信息，但是 **Request Cookies**，已经没了，这进一步验证了我们淘宝的 cookies 没有被发出。

相比方法一，方法二的好处就是，你的微博的 cookies 依然存在，这样你不需要每次登录都重新输入用户名和密码。

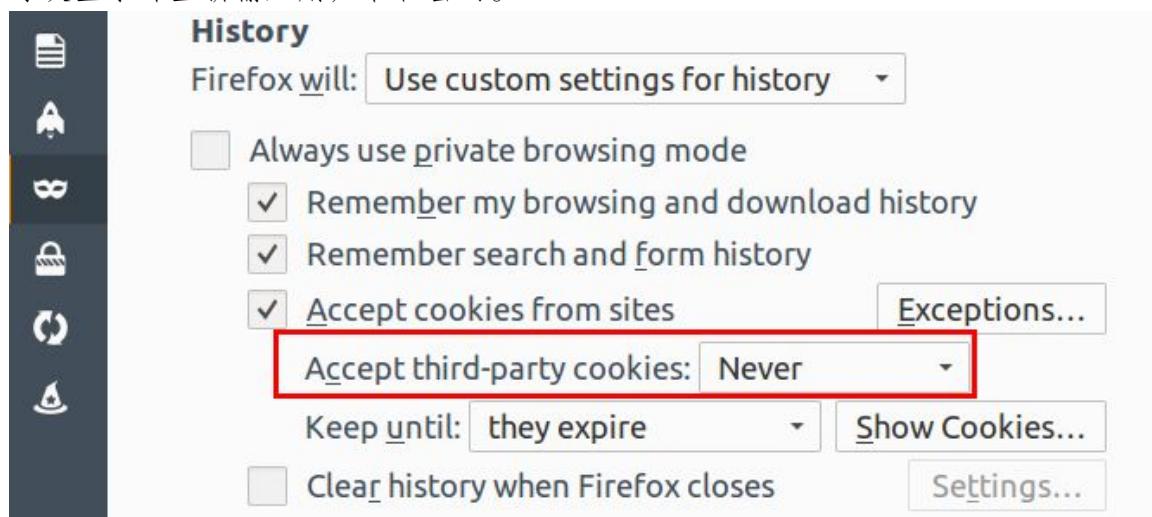


图 2-47 第三方 cookies

Domain	Type	Headers	Cookies	Params	Re
js2.t.sinajs.cn	js				
log.mmstat.com	gif				
log.mmstat.com	gif				
login.taobao.com	html				
mu1.sinaimg.cn	jpeg				
mu1.sinaimg.cn	jpeg				
mu1.sinaimg.cn	jpeg				

图 2-48 无第三方 cookies

在结束本小节之前，我们一起来看看一个和隐私有关的小故事，故事出处：<http://internet.solidot.org/article.pl?sid=12/03/29/0658236>。

2010 年 11 月 13 日，34 岁的佛罗里达居民 Christopher Chaney 想搜寻明星未公开的裸照。他从一个法庭文档上知道了好莱坞著名手包设计师西蒙娜·哈露什 (Simone Harouche) 的电邮帐号，但不知道密码。他利用从网上搜索到的邮箱安全问题答案，联系苹果的电邮服务，重置了密码。他将哈露什收件箱的邮件全部转发到专门为此次目的建立的雅虎邮箱。在哈露什恢复帐号控制后，她并没有立即注意到帐号设置被人动了手脚。哈露什在好莱坞交友甚广，通过她的联系人列表，他找到了克里斯蒂娜·阿奎莱拉(Christina Aguilera)的邮件地址，在尝试获得密码失败后，他模仿哈露什向阿奎莱拉索要穿衣很少的照片，结果阿奎莱拉真的给了他照片，随后立即被上传到网上。利用类似方法，他入侵了 50 多名明星的电邮，其中包括米娜·古妮丝和斯嘉丽·约翰逊，他登录斯嘉丽的帐号，要朋友发回私照的副本，他收到了多幅裸照，这些照片都在互联网上传播。当事情越闹越大，他才注意要隐藏 IP 地址，但为时已晚。当 FBI 拿着搜查令搜查时，他承认了所有罪行。他受到了多项罪行指控，面临最高 60 年徒刑。

2.6. 证明“你妈是你妈”

还记得第一章 curl 命令中的 `insecure` 选项吗？

`insecure` 选项告诉 curl 命令不对 12306 网站的证书进行校验。

那什么是证书呢，为什么要有证书呢，证书是干嘛的呢？

嗯，问题稍微有点多，简单的说，证书可以证明“12306 是 12306”。

为了更好的理解证书，我们需要先介绍一下 PKI(public key infrastructure) 和 Hash 的基础知识。

简单说，PKI 就是一组密钥对，私钥和公钥。如果我们使用公钥对信息进行加密，那么只能使用私钥对信息进行解密；反之，如果使用私钥加密，则需要使用公钥进行解密。RSA 是最常用的公钥加密算法。

Hash 则类比于信息的指纹，我们很难找到两个不同的人却拥有相同的指纹；那么同理，只要我们采用复杂的 Hash 算法，也很难找到两段不同的信息，却拥有相同的 Hash 值。常用的 Hash 算法有 MD5、SHA、SHA256 等。另外 Hash 还用一个很

重要的特点，无论输入信息量是有多长，Hash 输出的信息都是定长的，比如 MD5 算法输出结果是 16 个字节，SHA 则是 20 个字节。这样导致 Hash 有一个很重要的特性，不可逆性。比如我有一段信息的 SHA 值是“4531c606fa801a50749a6923ce1d43725f7e1c65”，你能知道我的原始信息吗？

好了，现在假设我们手里有这样一组密钥对，PrivateKey 和 PublicKey，其中 PrivateKey 自己保留（这点很重要），PublicKey 则发布出去。假设我们需要向外界发布的信息为 MessageValue，那么我们是如何向外界证明这个信息确实是由我发布的呢？

第一步：我们对 MessageValue 做 Hash 运算，得到 HashValue；

第二步：我们使用自己的 PrivateKey 对 HashValue 进行加密，得到 EncryptedHashValue；

第三步：我们把 MessageValue 和 EncryptedHashValue 一起发送给信息接收方；

那么作为信息接受方，在收到 MessageValue 和 EncryptedHashValue，他是如何判断这个信息是否来自正确的发送方呢？

第一步：他把收到的 MessageValue 也做 Hash 运算，得到 HashValue2；

第二步：他使用我们发布的 PublicKey 对收到 EncryptedHashValue 做解密运算，得到 DecryptedHashValue；

第三步：他对比一下 HashValue2 和 DecryptedHashValue 的值，如何两个一样，那么可以证明这条信息的 HashValue 确实是使用我们的 PrivateKey 加密的，因为 PrivateKey 是由我们自己保管的，那么这进一步证明了这个信息是由我们发出去的。

看完这个过程，细腻的你也许会有这样的疑问：因为我们的 PublicKey 是对外公开的，那么有没有可能破坏者根据我们的 PublicKey，破解我的 PrivateKey 呢？

嗯，这种想法理论是可行，而它的对抗方法也很简单，使用负责的加密算法。比如我们使用 4096 比特的 RSA 加密算法，那么可能需要几万年才能破解你的 PrivateKey，这时破解已经显得无意义了

有了这样的基本认识之后，我们来看看网站的证书到底长什么样。网站的证书至少包含两部分信息。**第一部分：**关于这张证书的基本信息，比如证书的使用者是谁，证书到期时间，证书是由谁签发的等；**第二部分：**则是由证书的签发机构，对上述基本信息做 Hash 运算，并使用该机构自己的 PrivateKey 对 Hash 运算结果加密得到，那么信息接收者在收到这张证书后，可以根据证书签发机构的 PublicKey 验证该证书是否由该机构签发，进而选择是否信任该证书。

好学的同学也许会有这样的疑问，谁来对证书的签发机构做认证呢？

嗯，证书签发机构是由更上一层的机构对其做认证的。

那么谁对这个“更上一层”的机构做认证呢？

嗯，这是由“更上一层”的机构完成的。那么如此递推下去，最后一级的机构只能是自己对自己做认证，这种“最后一级机构”的证书称为根证书。无论是操作系统还是浏览器，在发布时都会预装一些受信任的根证书。对于 Firefox 浏览器，在 P

reference->Advanced->View Certificates->Authorities，可以查看当前已安装的根证书。12306 的订票页面上使用的是的根证书是由中铁数字证书认证中心（Sinorail Certification Authority）签发的，而这个根证书并未被系统所预装，所以在 12306 网站首页上醒目的提醒用户需要安装根证书。

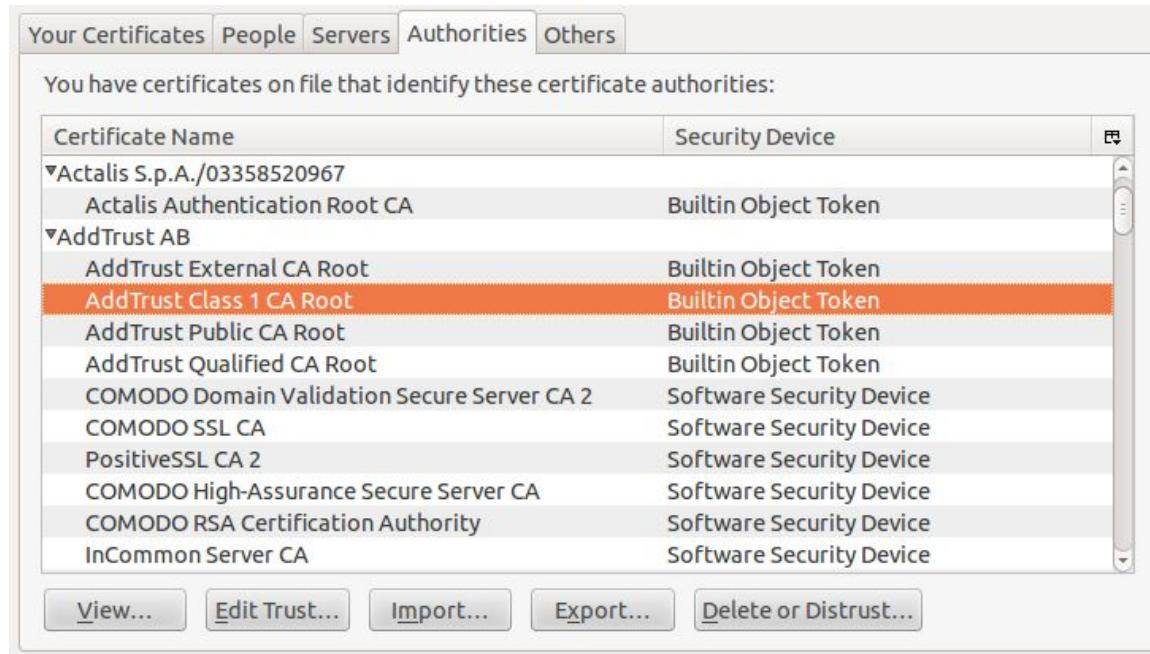


图 2-49 Firefox 根证书



图 2-50 12306 根证书

那么我们首先来解决第 1 章的遗留问题：在 curl 命令中如何不使用 insecure 选项？解决方法很简单，只要在 curl 命令中指明使用 12306 的根证书即可，具体步骤如下。

第一步：在 12306 官网下载根证书，并解压得到 srca.cer；

第二步：使用 openssl 将 srca.cer 由 DER 格式转换成 PEM 格式；

```
openssl x509 -in srca.cer -inform DER -out srca.crt -outform PEM
```

第三步：在 curl 命令中指定使用 srca.crt 证书

```
curl --cacert srca.crt "https://kyfw.12306.cn/otn/login/init"
```

解决第 1 章的遗留问题后，我们来看看 12306 网站的证书包含哪些内容，并手动使用根证书验证 12306 网站的证书。

第一步：首先让我们掌声有请网络调试神器 wireshark。

第二步：在 Capture->Option->Capture Filter 中输入“host kyfw.12306.cn and port 443”。



图 2-51 wireshark 设置

第三步：在浏览器中进入 12306 官网，并转到余票查询页面。

第四步：在 wireshark 的 Filter 中输入“**ssl.handshake.certificate**”，筛选出包含认证信息的网络包；

Filter: ssl.handshake.certificate						
No.	Time	Source	Destination	Protocol	Length	Info
124	4.423863	180.97.178.210	192.168.1.107	TLSv1.2	1506	Server Hello, Certificate, Server Hello Done
128	4.424924	180.97.178.210	192.168.1.107	TLSv1.2	1506	Server Hello, Certificate, Server Hello Done
213	4.435088	180.97.178.210	192.168.1.107	TLSv1.2	1506	Server Hello, Certificate, Server Hello Done

图 2-52 过滤网络包

第五步：选择其中一个网络包，展开 Secure Socket Layer 到 Certificates 选项，可以发现该选项下包含两个证书，证书 1 为 12306 网站的证书，该证书由证书 2 签发，证书 2 即为中铁数字证书认证中心的根证书。

The screenshot shows the SSL handshake details. Under the 'Secure Sockets Layer' section, it lists 'TLSv1.2 Record Layer: Handshake Protocol: Server Hello' and 'TLSv1.2 Record Layer: Handshake Protocol: Certificate'. The 'Certificate' section is expanded, showing 'Handshake Type: Certificate (11)', 'Length: 1372', and 'Certificates Length: 1369'. A red box highlights the 'Certificates (1369 bytes)' section. Below it, two certificates are listed: 'Certificate (id-at-commonName=kyfw.12306.cn, id-at-organizationalUnitName=\224\215[\b7g\rR\N_\, id-at-organizationName=Sinorail Certification Authority, id-at-countryName=CN)' and 'Certificate (id-at-commonName=SRCA, id-at-organizationName=Sinorail Certification Authority, id-at-countryName=CN)'. Both certificates are circled in red.

图 2-53 wireshark 中的证书

第六步：展开证书 1 即可观察 12306 网站证书的详细信息。signedCertificate 即为证书的基本信息，包括证书版本号、证书序列号、证书签发机构、证书有效期、证书公钥等；algorithmIdentifier 指明该证书的 Hash 算法为 SHA，加密算法为 RSA；最后 encrypted 的内容即为证书基本信息 Hash 之后再加密的结果。那么从理论上讲，encrypted 内容解密后的输出应该和 signedCertificate 内容 Hash 后的输出一致。

```

▼ Certificate (id-at-commonName=kyfw.12306.cn, id-at-organizationalUnitName=\224\215[\b7g\rR\N_\, id-at-organizationName=Sinorail Certification Authority, id-at-countryName=CN)
  ▼ signedCertificate
    version: v3 (2)
    serialNumber: -1237049511
  ▶ signature (shaWithRSAEncryption)
  ▶ issuer: rdnSequence (0)
  ▶ validity
  ▶ subject: rdnSequence (0)
  ▶ subjectPublicKeyInfo
  ▶ extensions: 5 items
  ▶ algorithmIdentifier (shaWithRSAEncryption)
    Padding: 0
    encrypted: 45deal392f51549e433031148bea7c2e63bc1b80a5cce89e...

```

图 2-54 12306 网站证书

第七步: 在证书 1 的 signedCertificate 上单击右键, 选择 Export Selected Packet, 导出证书基本信息, 另存为文件 12306signedCertificate.dat, 同理导出 encrypted, 另存为文件 12306encrypted.dat。

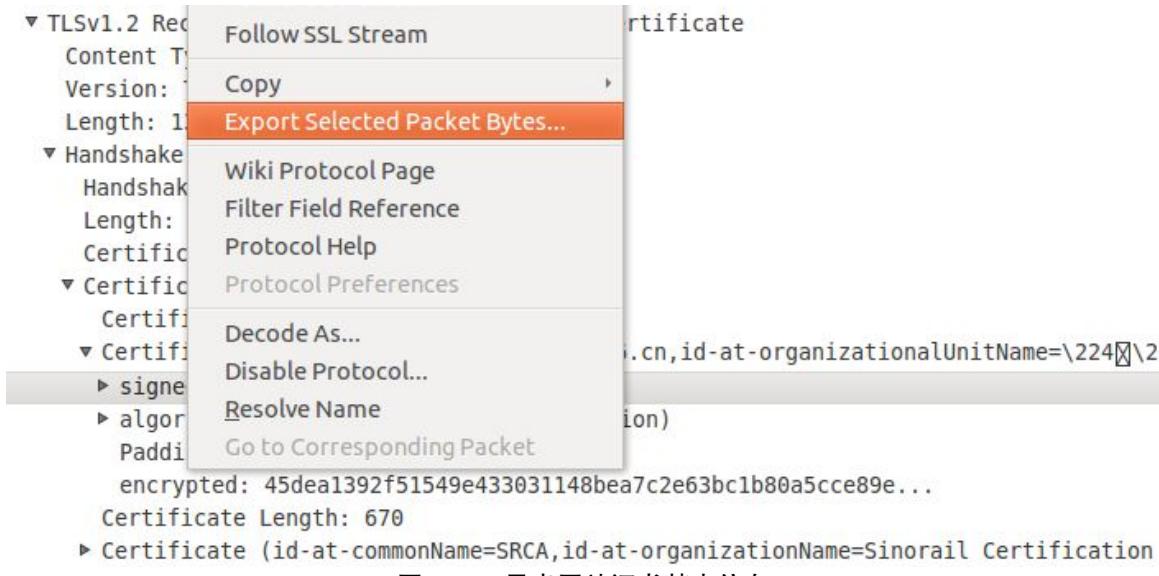


图 2-55 导出网站证书基本信息

第八步: 计算 12306signedCertificate.dat 文件的 SHA 值。

```
cyf@cyf$shasum 12306signedCertificate.dat  
44eb76e302bcd76dfd24446911890bb4c6b4b60c 12306signedCertificate.dat
```

第九步: 从 12306 根证书 srca.cer 中导出证书公钥, 另存为 srca.pub。

```
openssl x509 -in srca.cer -inform DER -pubkey -noout > srca.pub
```

第十步: 使用 srca.pub 公钥对加密数据 12306encrypted.dat 进行解密, 结果另存为 12306decrypted.dat。

```
openssl rsautl -in 12306encrypted.dat -inkey srca.pub -pubin > 12306decrypted.dat
```

第十一步: 12306decrypted.dat 是采用 BER 格式编码的数据文件, 可使用 16 进制编辑工具打开, 或直接使用 wireshark 打开。Wireshark 打开之后, 可以看到 OCT ESTRING 的内容即为 12306signedCertificate.dat 文件的 SHA 值。

```
► Frame 1: 35 bytes on wire (280 bits), 35 bytes captured (280 bits)  
▼ SEQUENCE  
► SEQUENCE  
OCTETSTRING: 44eb76e302bcd76dfd24446911890bb4c6b4b60c
```

图 2-56 解密后的数据文件

热心的读者可以自行导出证书 2 的内容, 对比可以发现其内容与 srca.cer 完全一致。二进制文件怎么对比呢? 对比两个文件的 Hash 值即可。

3. 爬下 12306

虽然磨刀不误砍柴工，但是我们不能光磨刀不砍柴。现在开始我们回归本书主题，重点讲讲如何用程序实现向 12306 网站提交订票信息。

读了前面的内容，我相信很多读者心中会有类似这样的解决方案雏形：使用 Firefox 的开发者工具抓取订票时的页面请求数据，然后使用程序模拟浏览器行为实现程序订票。

嗯，大体上说，这个想法是正确的，而且我们确实是这么干的。动手能力强的读者可以先合上书本，自己试着写写看。

也许部分读者会有这样的疑问：你们是如何破解 12306 的图形验证码的？

嗯，关于这个问题，我只能说让你失望了。

因为破解 12306 验证码并不是本书的重点，虽然这本书是介绍如何爬下 12306 网站订票信息，如何实现程序自动订票的，但是 12306 仅仅只是一个载体，我想要介绍的是一套方法。作者更希望读者在工作中碰到类似问题时知道如何解决，有哪些工具可以帮我们分析问题，解决问题。所以本书并没有解决 12306 的验证码问题，而是由程序向用户弹出一个消息框，请用户自行输入验证码。比如在登录页面中，我们是这样实现的，见下图。



图 3-1 12306 登录页面

如果读者确实对破解 12306 验证码感兴趣，可以在互联网搜索相关资料。这篇文章给出了相关的示例程序：<https://github.com/andelf/fuck12306>。

3.1. PyQt

本章的程序采用 PyQt4 编写图形界面，在正式扒 12306 之前，我们先简单介绍一下 PyQt4，以及 IDE 开发环境 Eric，如果读者曾经有过这方面的使用经验，可以

直接跳过本小节。本文使用的 Python 版本是 2.7.3，PyQt4 版本是 4.9.1。

本小节的目标是在 Eric 环境中，一步一步搭建如图 3-1 所示的图形界面，并运行。

本小节部分内容参考：<http://eric-ide.python-projects.org/tutorials/MiniBrowser/index.html>

第一步：启动 Eric。

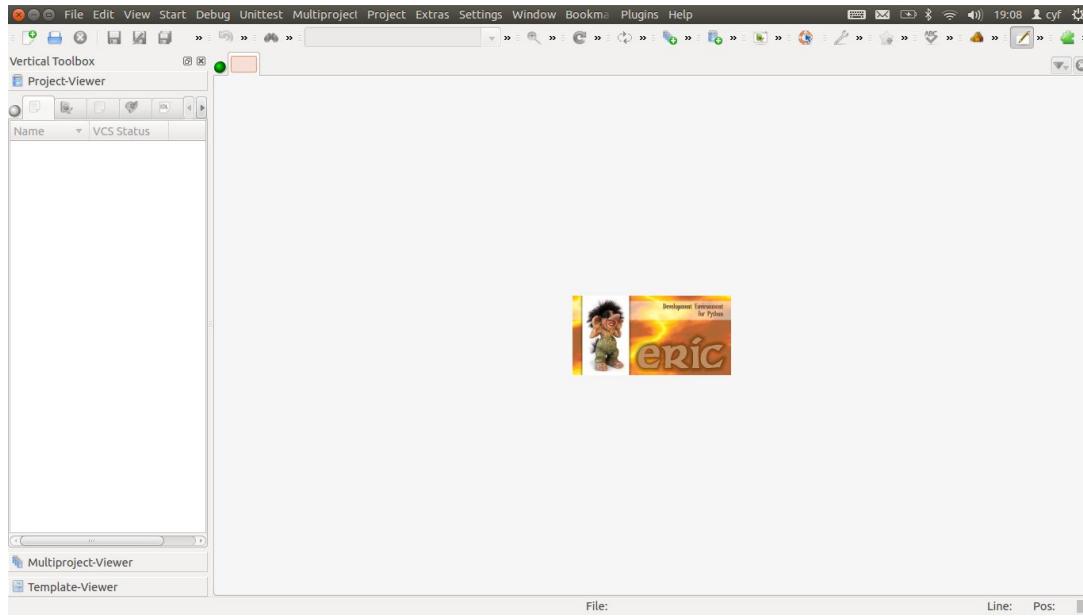


图 3-2 : ric 启动界面

第二步：点击 Project 菜单，选择 New。

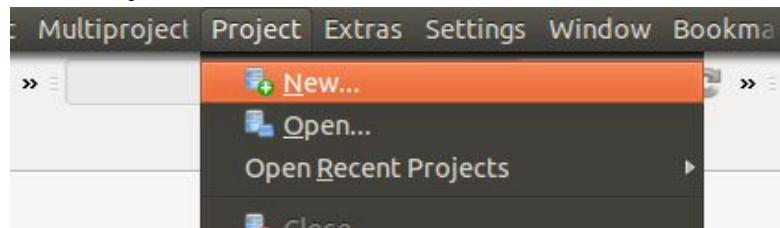


图 3-3 Project->New

第三步：设置工程名称为 12306、主脚本为 12306.py。

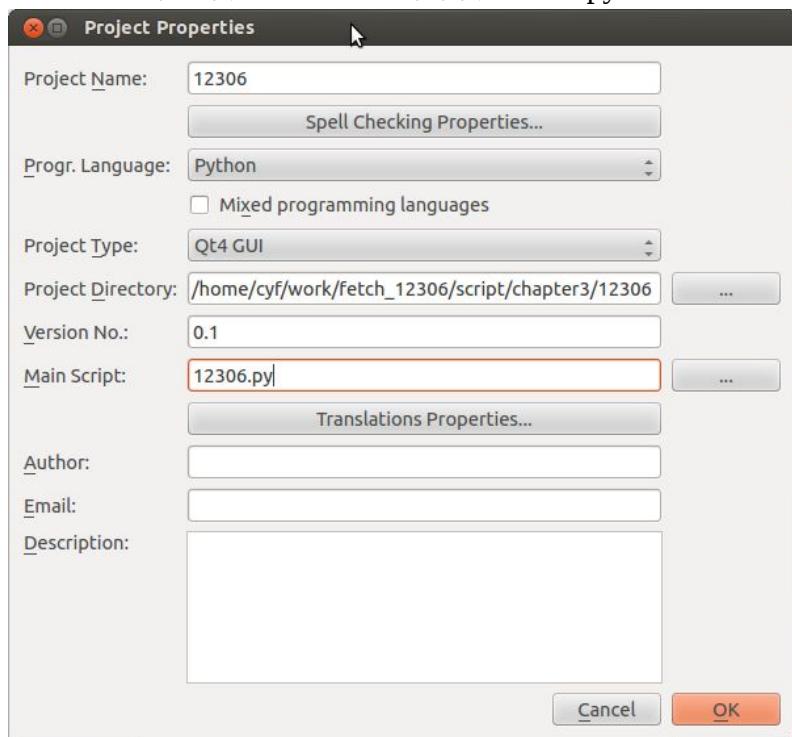


图 3-4 设置工程基本信息

第四步：选择 No，不向工程添加文件。

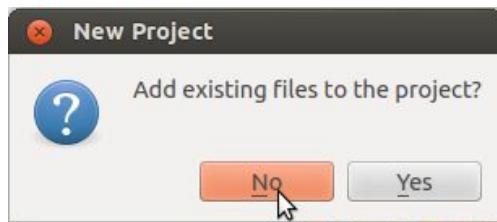


图 3-5 不添加文件

第五步：版本控制系统选择 None。

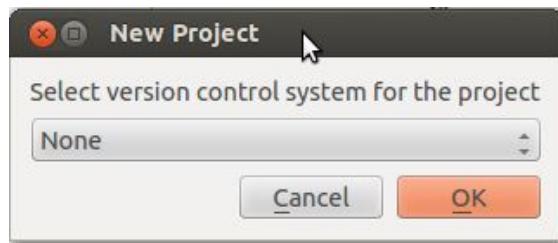


图 3-6 不使用版本控制

第六步：新建的工程应该包含两个文件 `__init__.py` 和 `12306.py`。

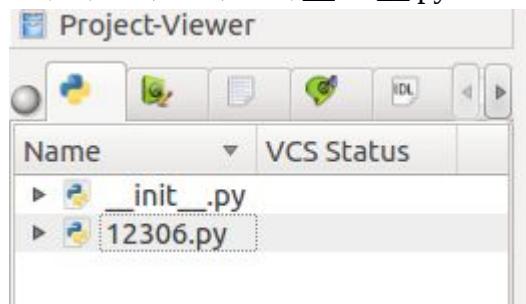


图 3-7 工程文件

第七步：右键，选择 New Package。

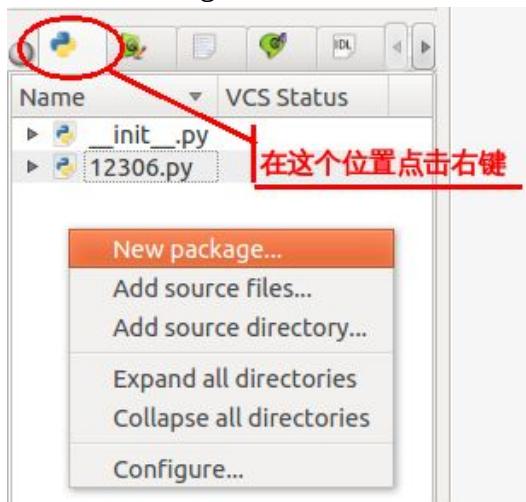


图 3-8 New package

第八步：新增 Package，ui，用于存储用户界面的 ui 文件，此时在工程目录下会多出一个名为 ui 的文件夹。



图 3-9 Package ui

第九步：右键，选择 New form。



图 3-10 New form

第十步：选择窗体类型为 Dialog。

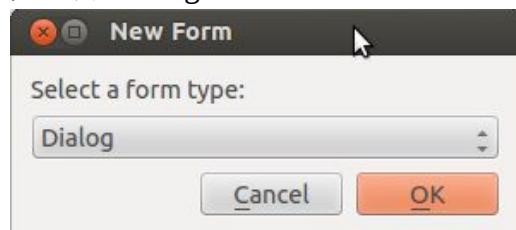


图 3-11 Dialog form

第十一步：存储文件名 login.ui，存储路径为之前建立的 ui 文件夹。

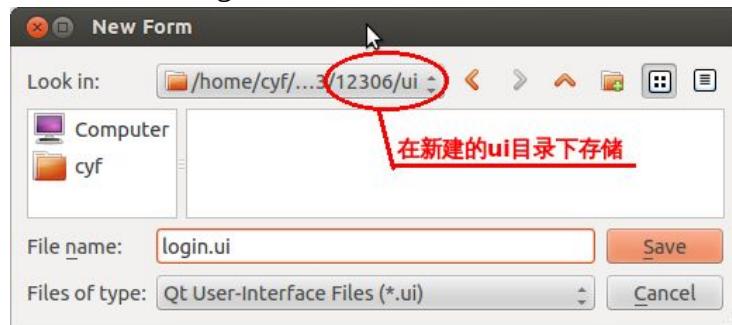


图 3-12 Dialog form

第十二步：在工程目录中可以看到新建的 login.ui 文件。

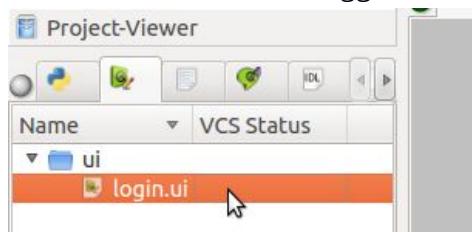


图 3-13 新建 login.ui 文件

第十三步：此时 Eric 自动调用 Qt Designer 打开 login.ui 文件，如果未自动打开，则双击 login.ui 文件打开。

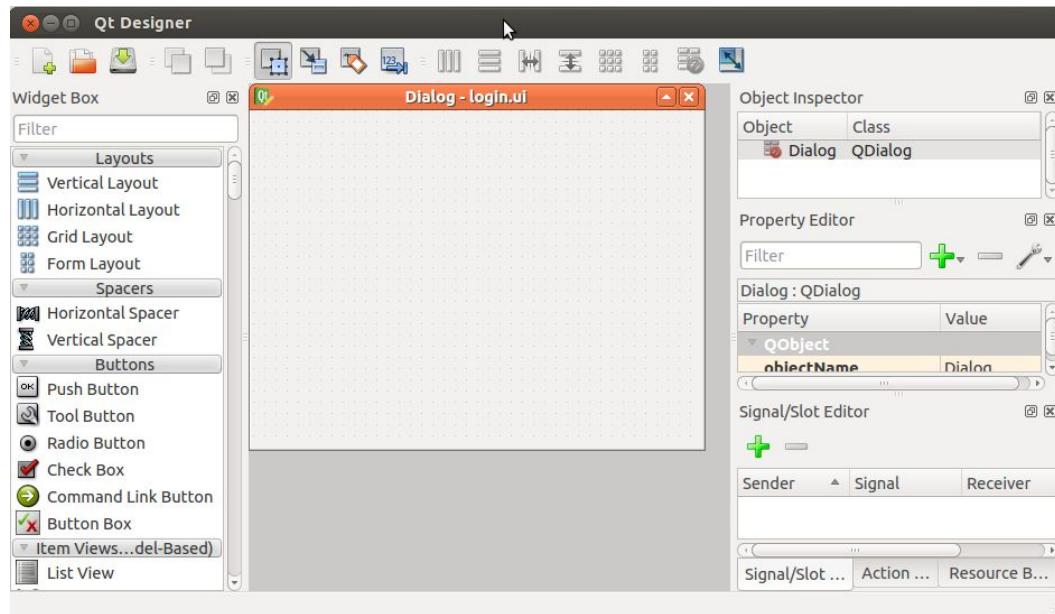


图 3-14 Qt Designer 打开 login.ui 文件

第十四步：修改 login.ui 窗体的 objectName 为 dlgLogin，windowTitle 为 12306 Login。

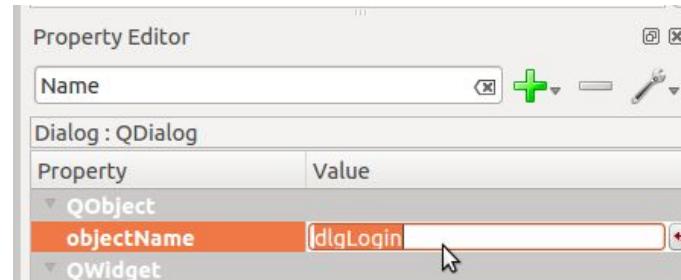


图 3-15 dlgHogin

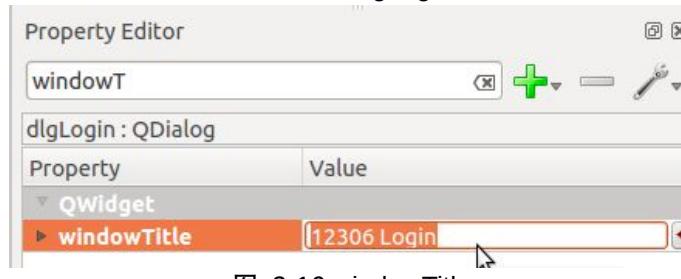


图 3-16 windowTitle

第十五步：按图 3-17 在窗体上放置控件。

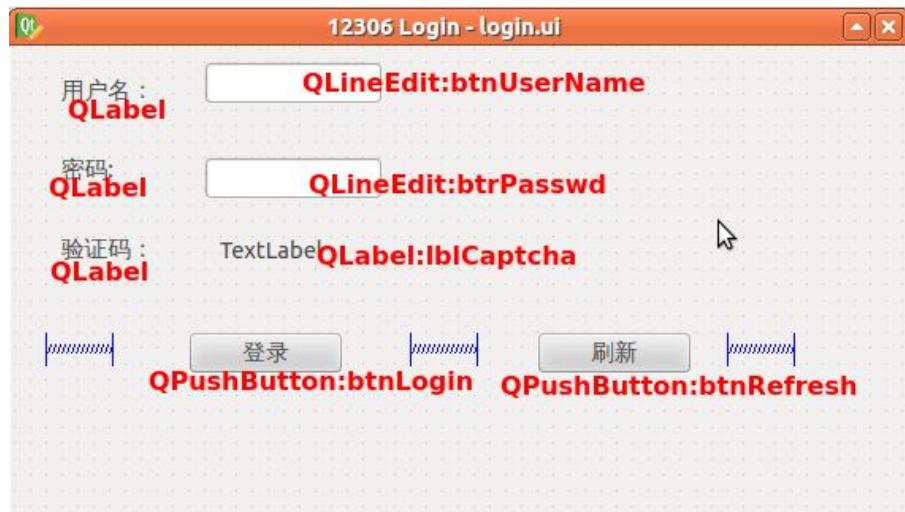


图 3-17 放置控件

第十六步：选中“用户名：”、“密码：”、“验证码：”、“TextLabel”以及两个文本输入控件，单击右键，选择“**Lay out**”->“**Lay Out in a Grid**”，效果如图 3-19 所示。

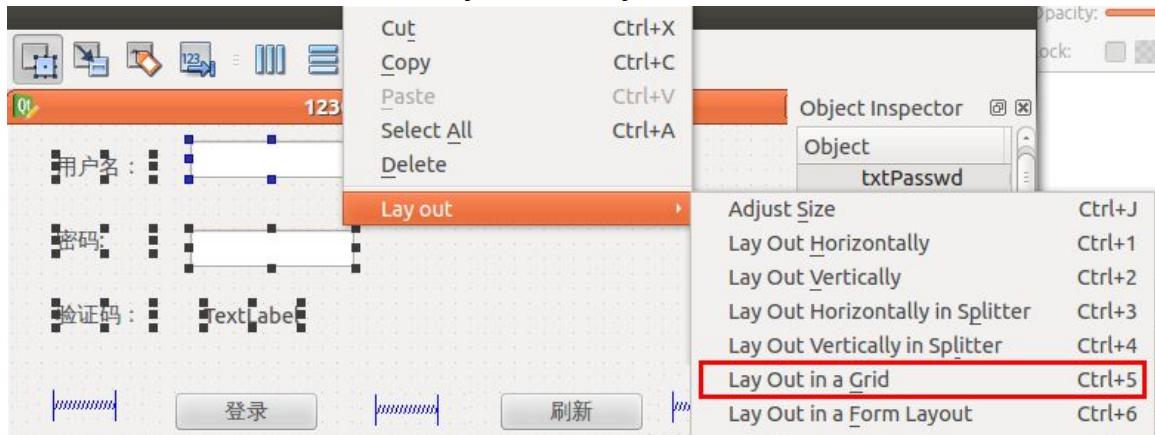


图 3-18 设置 Grid Layout



图 3-19 Grid Layout 效果

第十七步：选中剩下 5 个控件，右键选择“Lay out”->“Lay Out Horizontally”。

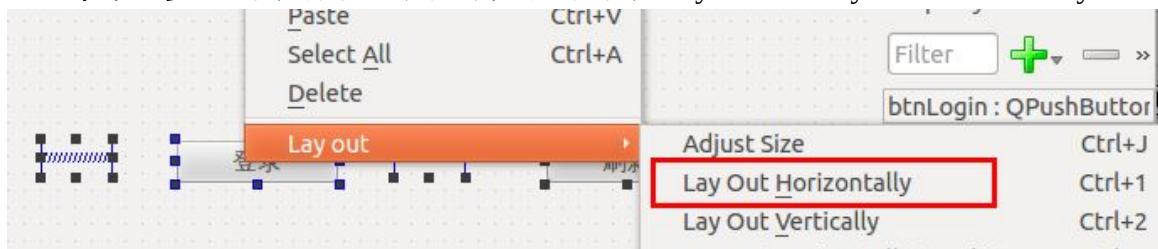


图 3-20 QoriJontally Hayout



图 3-21 QoriJontally Hayout 效果

第十八步：在窗体的空白区域点击右键，选择“Lay out”->“Lay Out in a Grid”，效果如图 3-23 所示。



图 3-22 Hay Out in a Rrid

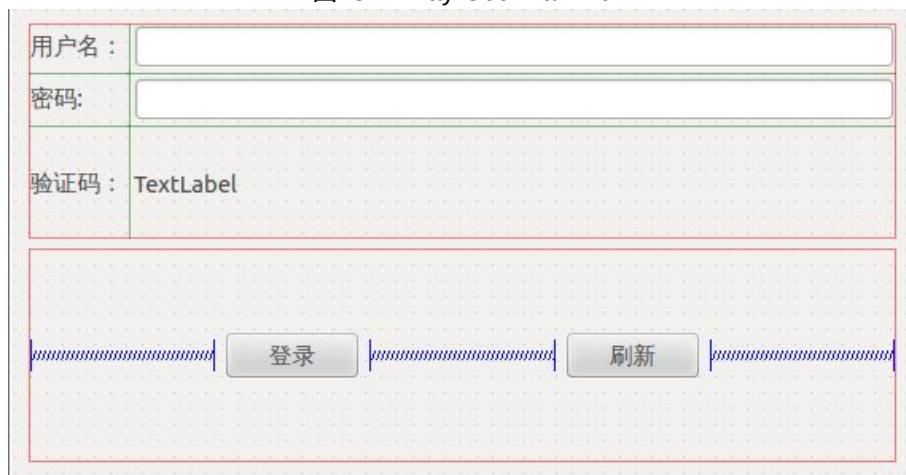


图 3-23 Hay Out in a Rrid 效果图

第十九步：在 12306 官网登录页面下载一张包含验证码的图片。

请点击下图中所有的 热水袋



图 3-24 12306 验证码

第二十步：设置 `TextLabel` 控件的 `pixmap` 属性指向为刚刚下载的 12306 验证码图片。

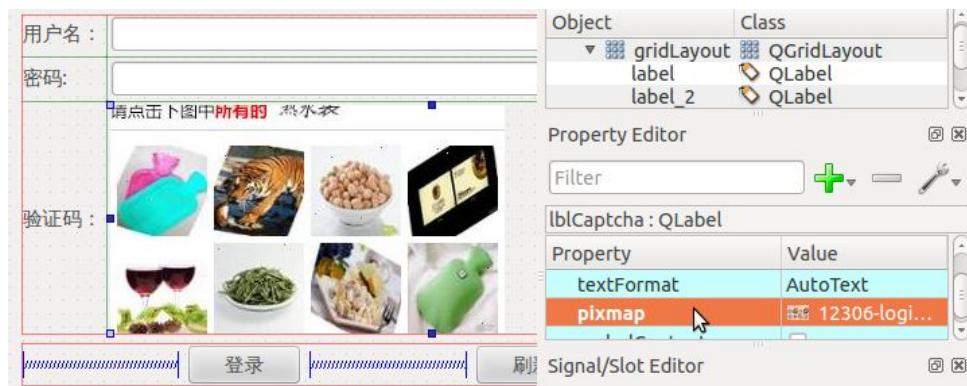


图 3-25 设置 `pixmap` 属性

第二十一步： 调解窗体大小至最合适尺寸，使得图片刚好可以完整显示，保存并退出 Qt Designer。



图 3-26 调解窗体大小

第二十二步：在 Eric4 环境中，右键 login.ui 文件，选择 Compile Form。

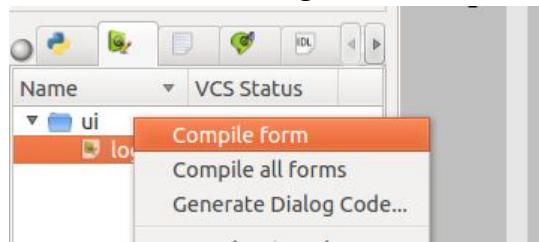


图 3-27 编译 login.ui

第二十三步：再次右键 login.ui，选择 Generate Dialog Code。

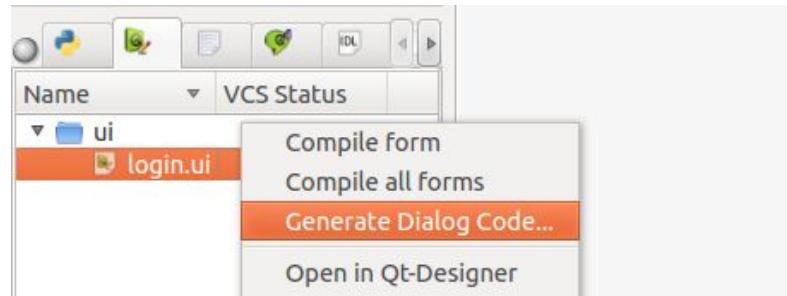


图 3-28 生存代码

第二十四步：设置 classname，存储路径选择 ui 文件夹，勾选 on_btnLogin_released() 和 on_btnRefresh_released()。

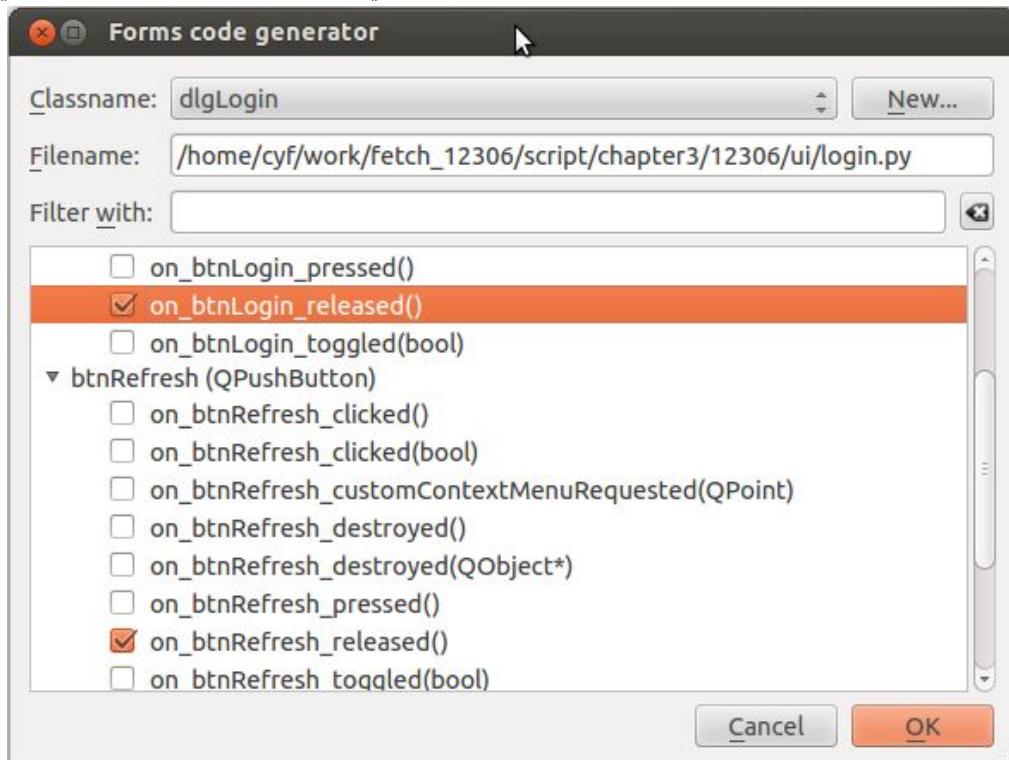


图 3-29 生存代码

第二十五步：在 Eric 工程中编辑 12306.py 文件，添加如下代码。

1. `# -*- coding: utf-8 -*-`
2. `from PyQt4 import QtCore, QtGui`

```

3. from ui.login import dlgLogin
4.
5. if __name__=="__main__":
6.     import sys
7.     app=QtGui.QApplication(sys.argv)
8.     ui_login=dlgLogin()
9.     ui_login.show()
10.    sys.exit(app.exec_())

```

图 3-30 12306.py 文件

第二十六步：打开 Ui_login.py，将 self.lblCaptcha.setPixmap，函数参数由相对路径改为绝对路径。

修改前：self.lblCaptcha.setPixmap(QtGui.QPixmap(_fromUtf8("../..../pic/chapter3/12306-login-check.jpeg")))

修改后：self.lblCaptcha.setPixmap(QtGui.QPixmap(_fromUtf8("/home/cyf/work/fetch_12306/pic/chapter3/12306-login-check.jpeg")))

第二十七步：进入 12306 文件夹，输入命令 **python 12306.py**，运行程序，效果如图 3-31 所示。



图 3-31 12306 登录界面

3.2. 验证码那些事儿

所谓买房的时候我们不能只挑楼层，还要懂装修；买车的时候不能只看款式，还要选内饰。人身最可悲的遇到哪些不但长得比你帅，而且学习成绩也甩你 N 条街的同龄人。

我们不但需要有好外框，更需要提升自己的内在修养，现在我们开始聊聊如何帅气的提升逼格。

凡在 12306 网站上买过票的童鞋都知道，登录时我们需要输入用户名、密码以及脑洞大开才能看清楚的验证码。

我们先来说说看，为什么需要验证码？

图 3-32 很明确的说明了验证码的作用--I'm not a robot。

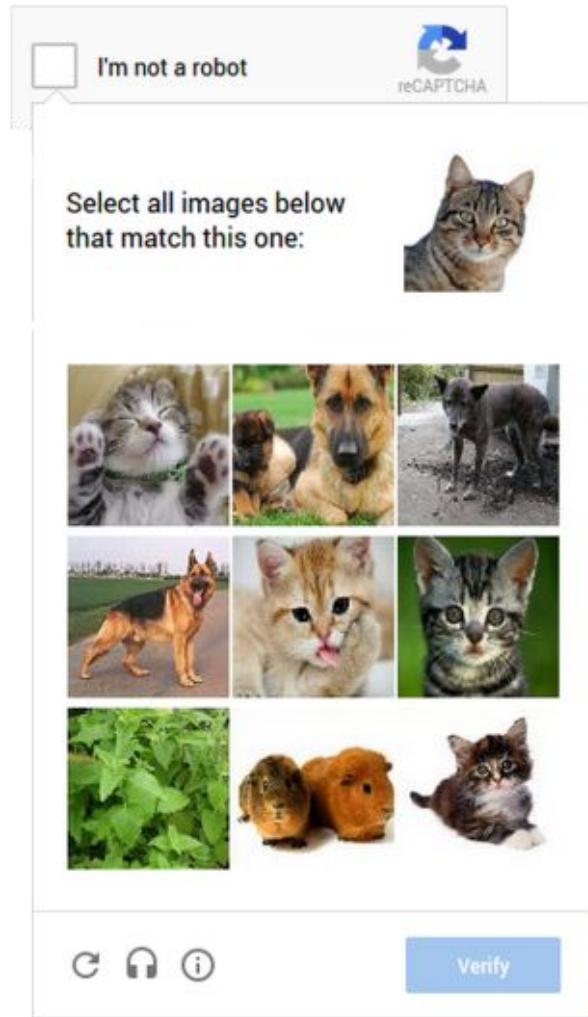


图 3-32 验证码的作用

在计算机领域有个很有名的测试，叫图灵测试(https://en.wikipedia.org/wiki/Turing_test)。

假设某一天你装小资，在星巴克喝下午茶，无聊之余微信帮你摇到靓妹一枚，于是你俩顺势聊上了，聊着聊着，你觉得和靓妹很投缘，简直就是相识恨晚啊。最后你提出希望和靓妹见面，请她喝下午茶。这时靓妹告诉你她其实是程序代码^_^!。

在上述的假象试验中，你其实就经历了一次图灵测试，而且最后是机器胜出的，因为程序代码严重欺骗了你的感情，让你误认为她是靓妹。

那么验证码可以看成是服务器程序对来访者做的一次简单的图灵测试，如果通过测试，则服务器认为你是个自然人，可以让你订票；否则服务器程序就认为你是个外挂机器人，将你拒之门外。

来个插曲，我们上点 ZhuangB 小知识，验证码的英文名为 Captcha，即 Completely Automated Public Turing Test to Tell Computers and Humans 的首字母缩写，意为全自

动区分计算机和人类的图灵测试。

现在让我们换位思考一下，让我们假象自己就是服务器程序，那么我们该如何处理验证码这档事呢？

现在假设我们就是服务器，此时用户 A 请求登录，我们给用户 A 发送验证码 cA，由于高峰时段比较忙，此时用户 B 也请求登录，我们给用户 B 也发送了验证码 cB。

在第 2 章“断或不断”小节中，我们提到了 HTTP 协议是无链接的、无状态的，那么现在问题来了：因为我们已经给用户 A 和用户 B 均发送验证码了，如果此时某个用户返回验证码的处理结果，那么我们如何分辨这是用户 A 的还是用户 B 的呢？

在我们的脑海中必须要有这样的概念，服务器程序必须是并发的，即服务器程序必须要在一个时间内同时处理多个用户的请求，它是并行的，不是串行的，不能在处理完用户 A 的所有请求后再去处理用户 B，必须能够同时处理用户 A 和用户 B 的请求。

现在让我们回到刚才那个问题，服务器是如何辨别当前返回数据是用户 A 的验证码还是用户 B 的验证码？

我想聪明的读者一定想到了，没错，答案就是 cookies。

按照第 2 章“水军”小节中提到的方法，查找与 12306 相关的 cookies，如图 3-33 所示。我们可以发现一个名为 **JSESSIONID** 的 Cookie。

Site	Cookie Name
kyfw.12306.cn	JSESSIONID
kyfw.12306.cn	BIGipServer0tn
kyfw.12306.cn	_NRF
kyfw.12306.cn	current_captcha_type

Name: JSESSIONID
Content: 0A02F015C42BC3A4CCD580695D67665A1D9F4A268E
Host: kyfw.12306.cn
Path: /otn
Send For: Any type of connection
Expires: At end of session

图 3-33 S: SSIONID

简单的说 JSESSIONID 类似与我们的省份证号码，由服务器统一分配。第一次链接 12306 时，服务器为客户端分配唯一的 JSESSIONID。如果我们清空 12306 的所有 cookies 后再链接服务器，则 12306 会为我们重新分别新的 JSESSIONID。

此处可以用一个简单的试验验证一下我们的想法：进入 12306 的登录页面后暂且不登录，先删除 JSESSIONID 这个 cookie，之后提交用户名、密码以及验证码请求登录，但此时你已经无法登录了，因为没了 JSESSIONID，服务器就无法证实你提交验证码是否正确，所以你就无法正确登录了。

聊完 JSESSIONID，我们来聊聊验证码本身，就是客户端是如何把验证信息提交给服务器的？

我们知道 12306 最初的验证码是类似图 3-34 样子的，对于这样的验证码，客户端只需要把图片信息转换成文字信息，提交到服务器既可。但是类似图 3-31 的验证码，在请求登录的过程中，客户端向服务器端提交了什么信息来表示当前的验证码呢？



图 3-34 12306 早期验证码

多说无益，我们直接看网络上的数据吧，使用 Firebug 捕获一次完整的登录过程，我们来看看，在验证码这档事上，客户端发了哪些数据到服务端。

进入 12306 登录页面，开启 Firebug 调试助手，点击 Net 选项卡下的 Persist 按钮，见图 3-35(这个是必须的)。正确输入用户名、密码及验证码。成功登录后 Firebug 中的内容应该类似图 3-36。其中 POST checkRandCodeAnsyn 即为与验证码相关的一次数据提交。



图 3-35 按下 Persist 按钮

xhr	Clear	Persist	All	HTML	CSS	JavaScript	XHR	Images	Plugins	Media	Fonts	Tin
URL Status Domain Size Remote IP												
登录 客运服务 铁路客户服务中心												
	[+]	POST	checkRandCodeAnsyn	200	OK		kyfw.12306.cn		148 B	180.97.178.210:443		
	[+]	GET	loading.gif	200	OK		kyfw.12306.cn		1.1 KB	180.97.178.210:443		
	[+]	POST	loginAysnSuggest	200	OK		kyfw.12306.cn		153 B	180.97.178.210:443		
我的12306 客运服务 铁路客户服务中心												
	[+]	POST	userLogin	302	Moved Temporarily		kyfw.12306.cn		0 B	180.97.178.210:443		
	[+]	GET	initMy12306	200	OK		kyfw.12306.cn		2.5 KB	180.97.178.210:443		
	[+]	GET	sidebar.css.css?cssVersio	200	OK		kyfw.12306.cn		11.2 KB	180.97.178.210:443		

图 3-36 Firebug 捕获 12306 登录数据

点开 POST checkRandCodeAnsyn，在 Post 选项卡下可以看到类似图 3-37 的内

容，randCode 即为与验证码相关的数据。



图 3-37 12306 验证码数据

那 randCode 是什么东西呢？

坐标。

什么坐标呢，简单解释一下。在浏览器眼中，验证码的图片是长成这样子滴，如图 3-38 所示，图片上附加了一个 X-Y 坐标系。当我们在图片上点击时，浏览器记录了点中心位置的坐标值，浏览器就是把这些坐标值发给服务器用于验证码校验的。



图 3-38 验证码坐标系

在了解验证码原理之后，我们来看看代码实现的事。

第一步：定位图 3-38 中，坐标原点在整幅图片中的像素偏移，记为 pos_zero。

`self.pos_zero=QtCore.QPoint(0, 30)`

第二步：为图片添鼠标释放时的事件处理函数 `on_lblCaptcha_released`。

`self.lblCaptcha.mouseReleaseEvent=self.on_lblCaptcha_released`

第三步：在 `on_lblCaptcha_released` 事件处理函数中，当鼠标点击验证码图片时，在点击位置放置一个 20*20 的正方形，并使用当前位置减 pos_zero 即得到验证码值。

```
1. def on_lblCaptcha_released(self, event):
2.     click_icon = QtGui.QLabel(self)
3.     click_icon.setGeometry(QtCore.QRect(self.lblCaptcha.x() + event.x() - 10, self.lblCaptcha.y() - 10 + event.y(), 20, 20))
4.     click_icon.setStyleSheet("QLabel { background-color : red; }")
5.     click_icon.show()
6.     self.click_icon.append(click_icon)
```

```
7.     self.click_pos.append(event.pos()-self.pos_zero)
8.     print event.pos()-self.pos_zero
```

图 3-39 获取验证码

在本小节结束之际，作者再次说明一下，本文不涉及关于如何破解 12306 验证码的任何内容，如果读者确实对破解 12306 验证码感兴趣，可以在参考网络神人给出的示例程序：<https://github.com/andelf/fuck12306>。

3.3. 秘密潜入

验证码就像横在宝藏前的第一道门槛，这道门槛可以帮服务器挡住简单的非法入侵行为。当我们跨过这道门槛后，就得想办法登堂入室了。

经过上一小节的洗礼，我相信大部分读者都有这样的感悟--cookies 很重要。

关于 cookies，概括起来大概需要做两件事：

1. 服务器向客户端发送信息时，如果包含 cookies 信息，那么客户端需要把它存起来。
2. 客户端向服务器发送数据时，需要把之前存的 cookies 连带请求信息一起发送出去。

那么这两件事我们是不是需要手动干呢？

不需要，因为用好工具，可以做到事半功倍。

这里我们使用 python 的 requests 库。

更多信息可以查阅官网：<http://docs.python-requests.org>。

细心的读者也许发现了，上一小节关于验证码讨论中，我们其实还遗留了一个问题，那就是验证码图片的是从哪个 URL 上下载得到的？这个 URL 的基本格式是这样的：<https://kyfw.12306.cn/otn/passcodeNew/getPassCodeNew?module=login&rand=sjrand&0.27189045538806567>。

每次请求时，只有最后那个数字在变。

那最后那个数字是什么鬼呢？

随机数，javascript 中 Math.random() 调用产生的随机数。

好学的读者也许会有这样的疑问，作者是如何知道这个是 Math.random() 调用产生的随机数呢？

嗯，这是个好问题，关于这个问题，我只能惭愧的说，我是猜的。。。

当然我也不是纯粹的瞎猜的，我们可以在 12306 网页代码中找到证据支撑我们的猜测。

到目前为止，对于 Firefox 的开发者工具，我们只用到了 Network 这一项功能，现在我来看看另一个项功能——Debugger。

在 Debugger 的搜索框内输入 **!getPassCodeNew**，可以帮助我们定位到在 new.js 文件中有这样一条语句：

```
randCodeImg.src= ctx + "passcodeNew/getPassCodeNew?module=" + module + "&rand=" + place + '&' + Math.random();
```

我们有理由相信这里的 randCodeImg.src 就是验证码图片的 URL。

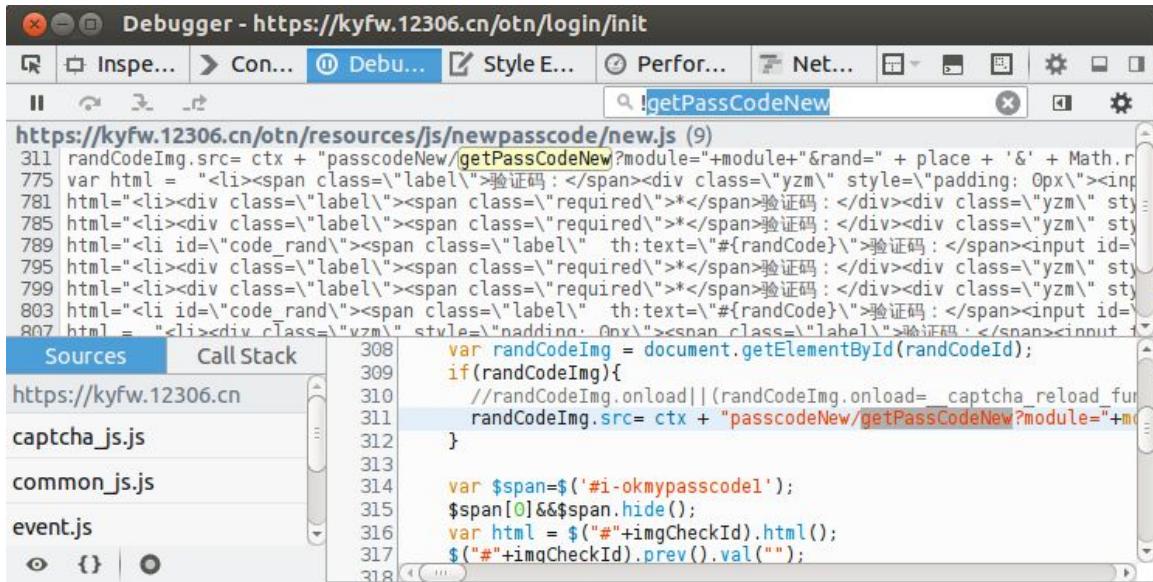


图 3-40 搜索 getPassCodeNew

也许有些读者已经看得不耐烦了，到目前为止，作者似乎一直在闲扯，还没有登录 12306。

好吧，作者也不为自己辩解了，我们下面就讲讲潜入 12306 的事。

如图 3-36 所示为用户登录 12306 时，浏览器与服务器之间交互的所有数据，登录过程描述如下：

第一步：浏览器向服务器发送验证码数据，图 3-37。

第二步：服务器向浏览器发送验证码的应答信息，图 3-41，如果验证码正确则继续，否则登录中止。



图 3-41 验证码正确应答

第三步：浏览器向服务器发送登录请求，包含用户名、密码和验证码数据，图 3-42。



图 3-42 登录请求

第四步：服务器向浏览器发送登录应答信息，图 3-43，如果用户密及密码正确则登录成成功。



```
{"validateMessagesShowId": "validatorMessage", "status": true, "httpstatus": 200, "data": {"loginAddress": "10.2.240.253", "otherMsg": "", "LoginCheck": "Y"}, "messages": [], "validateMessages": {}}
```

图 3-43 登录成功应答

第五步：没了，结束。

好吧，也许会有部分读者觉得作者什么都没讲似。其实作者挺为难的，因为方法我已经说明了，工具也已经介绍了，剩下的就是写代码的事了。虽然我也承认写代码的重要性并不亚于设计的重要性，但是写代码这东西真的没法说。

记得当初学 C 语言时，老师直接把算法那章跳过，我记得老师当时是这么说的，“算法是程序的灵魂，灵魂是没法讲的，也讲不透的，所以略过”。

所以作者建议读者先不要急着看后面的内容，让我们把书合上，打开电脑，在 3.2 小节的基础之上试着自己编写程序登录 12306 网站。如果读者自己编写的代码可以成功登录 12306，那么完全可以跳过本小节剩余内容。

-----我想静静的分割线!!!-----

-----不要问我静静是谁-----

现在我们假设读者已经完成尝试过了，那么我们就直接上代码吧。

在 12306.py 中，我们新建 requests.Session 的变量 conn，并且把该变量作为参数参数传入登录对话框。requests.Session 可以帮我们管理和 cookies 相关的东东，为此我们不需要显式的去存储发送 cookies，如图 3-44 所示。

```
1. if __name__=="__main__":
2.     import sys
3.     app=QtGui.QApplication(sys.argv)
4.     conn=requests.Session()
5.     ui_login=dlgLogin(conn)
6.     ui_login.show();
7.     sys.exit(app.exec_())
```

图 3-44 12306.py 程序

下面开始上正餐了，我们来看看登录对话框相关的函数，如图 3-45 所示。在构造函数中我主要定义了一些常量，如 self.rand_pic_url 表示验证码图片下载地址，self.headers 表示浏览器的 user-agent 声明。最后我们在构造函数中下载一张验证码图片并调整对话框尺寸。

```
1. def __init__(self,conn, parent = None):
2.     QDialog.__init__(self, parent)
3.     self.setupUi(self)
4.     winflag=self.windowFlags()
5.     winflag |= QtCore.Qt.CustomizeWindowHint
6.     winflag &= ~QtCore.Qt.WindowMaximizeButtonHint
7.     self.setWindowFlags(winflag)
8.     self.conn=conn
9.     self.rand_pic_url = 'https://kyfw.12306.cn/otn/passcodeNew/getPassCodeNew?module=login&rand=sjrand&'
10.    self.headers={
11.        "Host":"kyfw.12306.cn",
12.        "User-Agent":"Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0",
13.        "Connection":"keep-alive"
14.    r = self.conn.get(self.rand_pic_url+"%0.17f" % random.random(),headers
15.    =self.headers,verify=False)
16.    rand_pic= QtGui.QPixmap()
17.    rand_pic.loadFromData(r.content)
18.    self.lblCaptcha.setPixmap(rand_pic)
19.    self.lblCaptcha.mouseReleaseEvent=self.on_lblCaptcha_released
20.    self.click_pos=[]
21.    self.click_icon=[]
22.    self.pos_zero=QtCore.QPoint(0, 30)
23.    self.islogin=False
```

图 3-45 登录对话框 login.py 构造函数

下面出镜的应该是最重要的，登录按钮的响应函数，如图 3-46 所示。登录过

程已经在前文介绍过了，此处我们简单的提一提 JSON。

细心的读者也许注意到了，验证码及登录请求的返回格式均为 JSON。读者是否还记得在第一章时我们就提到了 JSON，当时我们没细说，现在我也不想细说。我们只提一点，requests 库已经帮我们隐藏了 JSON 解析的细节，我们只需要用好这个库就 OK 了。

例如在图 3-46 代码第 11 行，我们直接调用 `if r.json()["data"]["msg"]!="TRUE"` 检查验证码校验的返回值是否为 TRUE，而不是手动去解析如图 3-41 所示的字符串信息。

```
1. def on_btnLogin_released(self):
2.     self.rand_check_url='https://kyfw.12306.cn/otn/passcodeNew/checkRandCode
Ansyn'
3.     self.login_url='https://kyfw.12306.cn/otn/login/loginAysnSuggest'
4.     payload={"rand" : "sjrand", "randCode" : ""}
5.     for pos in self.click_pos:
6.         if payload["randCode"]=="":
7.             payload["randCode"] = "%s,%s" %(pos.x(), pos.y())
8.         else:
9.             payload["randCode"] += ",%s,%s" %(pos.x(), pos.y())
10.    r=self.conn.post(self.rand_check_url, data=payload, headers=self.headers, v
erify=False)
11.    if r.json()["data"]["msg"]!="TRUE":
12.        msg=QtGui.QMessageBox()
13.        msg.setText(_fromUtf8("验证码错误"))
14.        msg.exec_()
15.        self.on_btnRefresh_released()
16.        return
17.    print "rand code verify success"
18.    del payload["rand"]
19.    payload["loginUserDTO.user_name"]=str(self.txtUsername.text())
20.    payload["userDTO.password"]=str(self.txtPasswd.text())
21.    r=self.conn.post(self.login_url, data=payload, headers=self.headers, verify=
False)
22.    if r.json()["data"]=={}:
23.        print "username or password error"
24.        msg=QtGui.QMessageBox()
25.        msg.setText(_fromUtf8("用户名或密码错误"))
26.        msg.exec_()
27.        self.on_btnRefresh_released()
28.        return
29.    elif r.json()["data"]["loginCheck"]!="Y":
30.        print "username or password error"
31.        msg=QtGui.QMessageBox()
32.        msg.setText(_fromUtf8("用户名或密码错误"))
33.        msg.exec_()
```

```
34.         self.on_btnRefresh_released()
35.         return
36.     print "login success."
37.     self.islogin=True
38.     self.done(QDialog.Accepted)
```

图 3-46 login.py 登录按钮响应函数

最后呢，我们还剩一个函数还没介绍，那就是验证码刷新按钮的响应函数，如图 3-47 所示。

```
1. def on_btnRefresh_released(self):
2.     for icon in self.click_icon:
3.         icon.hide()
4.     self.click_icon=[]
5.     self.click_pos=[]
6.     r = self.conn.get(self.rand_pic_url+"%0.17f" % random.random(),headers
   =self.headers,verify=False)
7.     rand_pic= QtGui.QPixmap()
8.     rand_pic.loadFromData(r.content)
9.     self.lblCaptcha.setPixmap(rand_pic)
```

图 3-47 login.py 验证码刷新按钮响应函数

运行效果如图 3-1 所示，如果登录成功则打印“**login success.**”信息

3.4. 添砖加瓦盖大楼

在上一小节，我们谈了如何“登录”12306 网站，那么这一小节我们干点什么呢？也许有些童鞋已经磨拳擦掌，准备抢票了。

咳咳。。。。。

我们要做文明人，说抢多俗呀，我们不说抢，我们要正大光明、堂堂正正、大大方方的**订票**。

怎么个订票法呢？我们用软件模拟网上订票的所有操作。

Linux 的作者 Linus Torvalds 有过一句“名言”，学编程最好的方式是，“Read the f*** source code”。

在本小节中我们不再对每步操作进行详细解构，具体实现可以查阅随书附赠的源代码，因此本小节的主要内容是向各位读者演示一下如何使用我们的 12306 订票软件。

本书采用 PyQt 编写图形界面，关于 Qt 的更多知识，推荐阅读《C++ GUI Qt4 编程》。

程序启动时，首先进入图 3-1 的登录界面，如果用户登录成功，则进入图 3-48 的 12306 订票软件主界面；如果用户不登录，而是选择直接关闭登录界面，程序同样进入订票软件主界面，此时用户可以点击软件左上角的“**用户登录**”菜单，打开登录对话框。

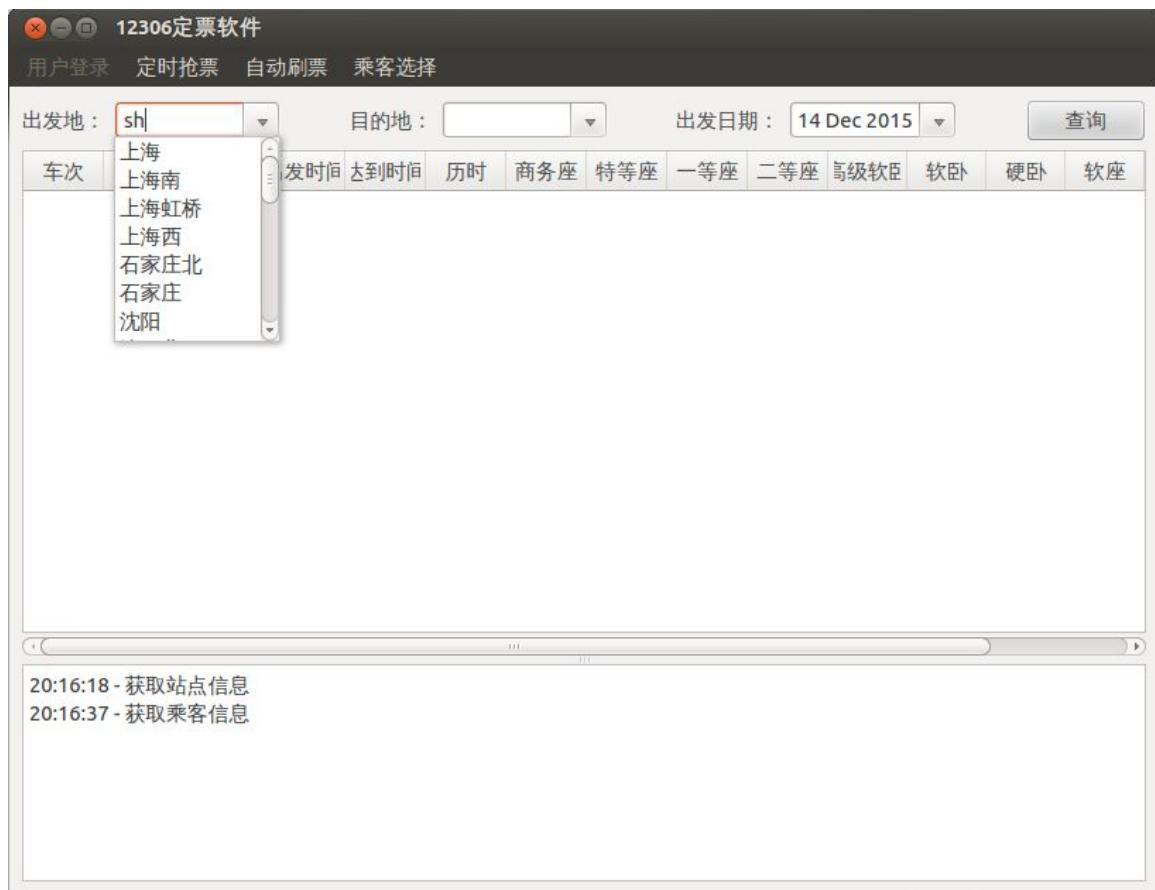


图 3-48 定票软件主页面

“定时抢票”和“自动刷票”为后续待开发的功能，目前只预留接口，无具体实现。如果用户登录成功，“乘客选择”菜单下罗列当前用户可以添加的乘客，如图 3-49 所示。



图 3-49 乘客选择

在图 3-50 的车票查询页面中，如果该趟车次有票，那么双击“剩余票数”，即可弹出如图 3-51 所示的订票页面。

在订票页面下勾选乘车人，并正确填写验证码即可完成订票操作。

如果一切顺利，最后会弹出如图 3-52 的消息框提示用户订票成功。

12306定票软件

用户登录 定时抢票 自动刷票 乘客选择

出发地 : 上海 目的地 : 北京 出发日期 : 8 Jan 2016 查询

	车次	出发站	到达站	出发时间	到达时间	历时	商务座	特等座	一等座	二等座	高级软卧	软卧	硬卧	软
1	G102	上海...	北京南	06:43	12:17	05:34	24	--	56	487	--	--	--	--
2	G104	上海...	北京南	06:59	12:23	05:24	23	--	134	413	--	--	--	--
3	G106	上海...	北京南	07:10	12:42	05:32	13	--	43	354	-- 双击	--	--	--
4	G108	上海...	北京南	07:20	13:11	05:51	25	--	75	488	--	--	--	--
5	G110	上海...	北京南	07:30	13:33	06:03	18	--	66	545	--	--	--	--
6	G12	上海...	北京南	08:00	13:16	05:16	10	--	93	179	--	--	--	--
7	G112	上海...	北京南	08:05	13:53	05:48	23	--	58	423	--	--	--	--
8	G114	上海...	北京南	08:18	14:12	05:54	24	--	99	315	--	--	--	--
9	G2	上海...	北京南	09:00	13:48	04:48	1	--	24	无	--	--	--	--
10	G116	上海...	北京南	09:34	15:23	05:49	20	--	69	488	--	--	--	--

图 3-50 车票查询

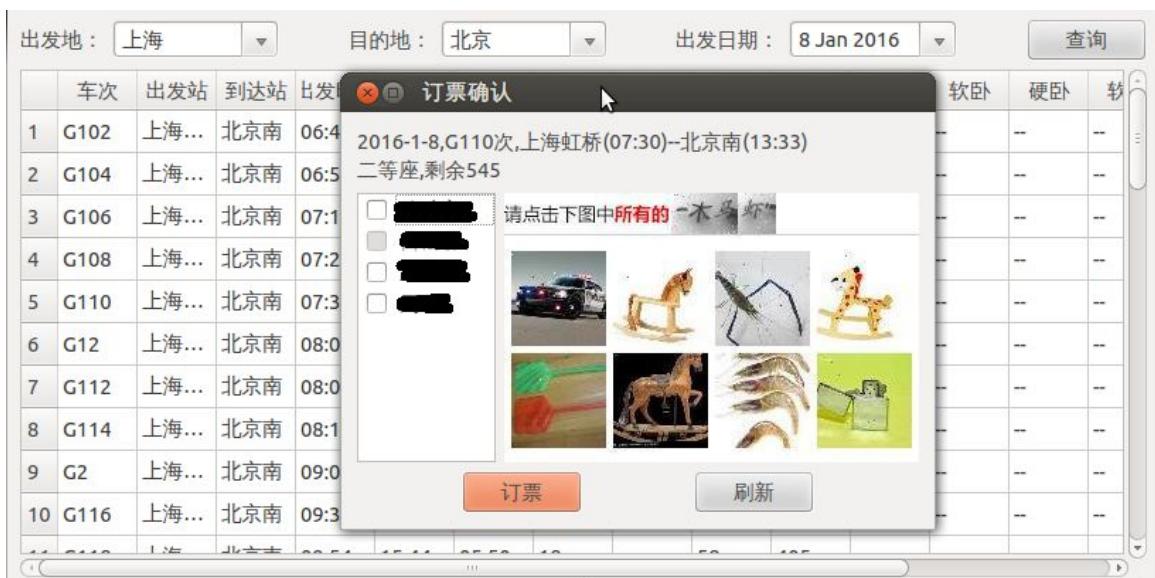


图 3-51 订票页面



图 3-52 订票成功

然后呢？

然后当然是网页或手机登录 12306 付款啊。

So easy，妈妈再也不用担心我订票了!!!

好吧，本小节就到此为止了，更多操作，试用一下软件就知道了 ^_^。

在读者试着自己写代码或阅读本文提供的代码之前，请允许作者再罗嗦一下，我们软件的核心思想是模拟浏览器订票时的各种操作，所以读者可以使用浏览器的网页调试工具捕捉订票所产生的所有数据，以这些数据为基础可以更好的编写代码或理解本文提供的代码。

-----我在静静地看源代码分割线-----

3.5. 后记

笑话一则：

话说某某公司因为服务器喜欢宕机，所以专门高薪招了名程序员来维护服务器软件。

程序员看了一下遗留代码，心中千万头草泥马奔腾而过。

于是程序在维护服务器的同时，利用业余时间把服务器重写了。

之后服务器再也没宕机了，程序员也开始了天天喝茶看报的幸福生活。

故事是不是到这里就结束了呢，程序员是否从此过上了幸福的生活呢，然而并不是这样的。

老板发现程序员已经连续一个月都在那喝茶看报了，于是就把他解雇了。

于是程序员失业了。

故事到此结束。

很久很久以前，我还是个懵懂的小孩时，那时我是想写一本书的。

当我写完前两章后，我发现剩下的很多事是和 Coding 息息相关的，而抢票的核心思想就是模拟浏览器向 12306 网站发数据请求。

所以高级的抢票工具几乎就是完全模拟浏览器行为的，或是直接在浏览器上开发插件的。

这本书，好吧，我承认暂时还没到书的级别，但是至少应该是一个小册子，我们姑且称之为小册子吧。

在 2016 年初的时候，我基本完成了这本小册子的前 3.4 小节的编写。

原先打算在个人博客上逐个章节发表的，后来因为工作原因出了趟长差，之后就把这事给耽误了。

原本我画了张很大饼，我想说说自动刷票，我也想讲讲定时抢票，我还想讲讲搭建一个SAE服务器，让服务器去帮我们去刷票。

现在也只能算是一个坑了，抱歉。

好吧，我承认主要原因是我懒。

精力有限，实在抱歉。

因为我觉得我已经把最核心的东东讲完了，如果读者感兴趣的话，剩下的东东可以自己去 [google](#)，然后自己动手实践的。

最后呢，再来个小插曲，前文强调过，强票的核心思想是模拟浏览器的行为向12306网站发送数据请求。

但是，请注意这个但是

但是到目前为止，我们其实一直是利用 Firebug 查看订票时我们与 12306 的哪些 URL 发生了哪些数据交互。换句话说，我们并没有模拟浏览器的行为去解析 12306 网页的任何信息。那么这种行为导致的直接后果就是如果 12306 把某些 URL 的地址

稍作修改，那么我们目前的软件就无法工作了，只能使用 Firebug 重新去获取新的 URL 地址，然后修改软件才能工作。

以上是问题一，下面还有个问题：

很多网站在设计的时候是考虑过阻止我们这种爬虫的，至少是希望能够阻止我们这种简单的爬虫行为的。

如何阻止呢？JS 脚本。

简单的说，如果你需要请求网页 A 的内容，那么你必须先请求 A.js 文件，之后才能请求网页 A。如果你跳过 A.js 文件直接请求网页 A 的内容，那么抱歉不能让你过去。

曾经有一段时间，我发现这个程序无论如何都无法正常登录，后来我在发送登录数据之前加了段对这个 URL 的请求，然后就 OK 了。

https://kyfw.12306.cn/otn/resources/merged/login_js.js

所以牛 B 的强票软件，至少都是在部分功能上是模拟浏览器行为的。

我为什么要说这个呢？

我请求你不要向我扔鸡蛋或西红柿，因为我想告诉你的是，前几天我刚刚试了一下，3.4 章节对应的那个程序其实已经**不能订票了**。

===== 没有西红柿鸡蛋的分界线 =====

那如何让他又能重新订票呢？

留给读者自己吧。

也许你从头看下来，已经自己实现了一个订票软件了。

恭喜恭喜。

===== 再听我罗嗦一下的分界线 =====

我写这个小册子的目的不是想实现一个强票软件，目前市面上的抢票软件很多，你真的想抢票的话，可以用他们的工具，而且很多工具连验证码都帮你做了。

我写这个小册子的目的是想告诉你一些关于网页或者说是 HTTP 的基础知识，然后以这个为基础，我们如何一步一步的自己构建一个刷票软件。

至于抢票呢，我更喜欢用这样一个小脚本去捡漏。

我相信聪明的读者一定知道这个脚本是如何工作的，也一定能改出一个适合自己的专用捡漏小脚本的。

本文相关的所有程序及脚本在这里：

<https://code.csdn.net/adream307/fetch12306.git>

```
1. #!/bin/bash
2. CNT=0
3. while [ 1 -gt 0 ]; do
4.     NUM=`curl --insecure --user-agent "Mozilla/5.0 (X11; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0" "https://kyfw.12306.cn/otn/lcxxcx/query?purpose_codes=ADULT&queryDate=2016-12-21&from_station=SHH&to_station=VHH" | grep -oP "(?=<{})[^{}]+(?=})" | grep "G7521" | sed -r 's/.*ze_num":' "[^"]+").*/1/` 
5.     echo "fetch G7521"
6.     if [ $NUM = "无" ]; then
7.         CNT=$((CNT+1))
8.     else
9.         paplay /usr/share/sounds/ubuntu/stereo/phone-incoming-call.ogg
10.    fi
11.    echo $CNT
12.    sleep 10
13. done
```

图 3-53 捡漏专用小脚本