Selenium+Chrome 爬取动态网页数据

网页上的信息不一定都是静态的 HTML 数据，实际上很多信息是通过 JavaScript 处理后得到的，那么怎么样去爬取这些数据呢？我们先来设计一个包含 JavaScript 的网页文档，看看怎么样爬取它的数据。

1、网页文件

网页文件 index.html 包含三个，第一个的 ID 是"hMsg"，它的信息是确定的"Html Message"；第二个 ID 是"jMag"，它的信息是在网页转载时由 JavaScript 的程序赋予值"JavaScript Message"；第三个的 ID 是"sMsg"，它的信息是网页在转载时通过 Ajax 的方法向服务器提出 GET 请求获取的，服务器返回字符串值"Server Message"。

```
<script>
    function init()
    {
        http=new XMLHttpRequest();
        http.open("get","/show",false);
        http.send(null);
        msg=http.responseText;
        document.getElementById("sMsg").innerHTML=msg;
        document.getElementById("jMsg").innerHTML="JavaScript Message";
    }
</script>
<body onload="init()">
Testing<br>
<span id="hMsg">Html Message</span><br>
<span id="jMsg"></span><br>
<span id="sMsg"></span>
</body>
```

2、服务器程序

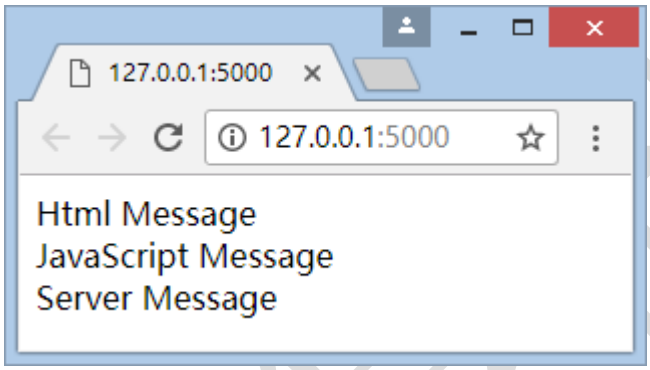
服务器程序 server.py 显示出 index.html 文件，其中 index 函数读取该文件并发送出去，show 函数是在接受地址"/show"请求后发送"Server Message".

```
import flask
app=flask.Flask(__name__)

@app.route("/")
def index():
    f=open("index.html","r")
    data=f.read()
    f.close()
    return data
```

```
@app.route("/show")
def show():
    return "Server Message"

if __name__=="__main__":
    app.run()
```

3、浏览器浏览

启动服务器 server.py 程序，浏览 web 地址 http://127.0.0.1:5000，结果如图。



4、普通的客户端程序

设计一个客户端程序 normalClient.py，它通过 urllib.request 直接访问 http://127.0.0.1:5000,程序如下：

```
import urllib.request
resp=urllib.request.urlopen("http://127.0.0.1:5000");
data=resp.read()
data=data.decode()
print(data)
```

执行该程序，我们看到输出的结果如下：

```
<script>
    function init()
    {
        http=new XMLHttpRequest();
        http.open("get","/show",false);
        http.send(null);
        msg=http.responseText;
        document.getElementById("sMsg").innerHTML=msg;
        document.getElementById("jMsg").innerHTML="JavaScript Message";
    }
</script>
<body onload="init()">
<span id="hMsg">Html Message</span><br>
<span id="jMsg"></span><br>
<span id="sMsg"></span>
```

```
</body>
```

这个结果就是 index.html 文件。

5、普通的爬虫程序

普通的爬虫程序如下：

```
from bs4 import BeautifulSoup
import urllib.request
resp=urllib.request.urlopen("http://127.0.0.1:5000");
html=resp.read()
html=html.decode()
soup=BeautifulSoup(html,"lxml")
hMsg=soup.find("span",attrs={"id":"hMsg"}).text
print(hMsg)
jMsg=soup.find("span",attrs={"id":"jMsg"}).text
print(jMsg)
sMsg=soup.find("span",attrs={"id":"sMsg"}).text
print(sMsg)
```

显然如果我们通过该方法获取网页 HTML 文档然后进行数据爬起，那么只能爬取 hMsg 的信息"HTML Message"，但是爬取不到 jMsg 与 sMsg 的信息的！因为 jMsg 与 sMsg 的信息不是静态地嵌入在网页中的，而是通过 JavaScript 与 Ajax 动态产生的，通过 urllib.request.urlopen 得到的网页中没有这样的动态信息，如果要得到这些信息就必须在获取网页后客户端能按要求执行对应的 JsvaScript 程序。

6、 模拟浏览器的客户端程序

前面已经分析了要获取 jMsg 与 sMsg 的这些信息就必须在获取网页后客户端能按要求执行对应的 JsvaScript 程序。显然一般的客户端程序没有这个能力去执行 JavaScript 程序，我们必须寻找一个能像浏览器那样工作的插件来完成这个工作，它就是 selenium。selenium 就是一个没有显示界面的浏览器，它能与通用的浏览器（例如 chrome,firefox 等）配合工作。我们要先安装 selenium 与 chrome 的驱动程序。

(1) 安装 selenium

```
pip install selenium
```

执行该命令即可完成安装。

(2) 安装 chrome 驱动程序

要 selenium 与浏览器配合工作就必须安装浏览器对应的驱动程序，例如要与 chrome 配合就要先到 http://chromedriver.chromium.org/网站下载 chromedrive.exe 的驱动程序，然后把它复制到 python 的 scripts 目录下。

根据 selenium 的说明，编写客户端程序 chromeClient.py 如下：

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
```

```
chrome= webdriver.Chrome(chrome_options=chrome_options)
chrome.get("http://127.0.0.1:5000")
html=chrome.page_source
print(html)
```

该程序先从 selenium 引入 webdriver，引入 chrome 程序的选择项目 Options：

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
```

设置启动 chrome 时不可见：

```
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
```

让后创建 chome 浏览器，并访问网站，获取网站的 HTML 文档，执行文档中的 JavaScript 程序，通过 page_source 获取最后的网页文档：

```
chrome= webdriver.Chrome(chrome_options=chrome_options)
chrome.get("http://127.0.0.1:5000")
html=chrome.page_source
print(html)
```

执行这个程序，结果如下：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><script>
    function init()
    {
        http=new XMLHttpRequest();
        http.open("get","/show",false);
        http.send(null);
        msg=http.responseText;
        document.getElementById("sMsg").innerHTML=msg;
        document.getElementById("jMsg").innerHTML="JavaScript Message";
    }
</script></head>
<body onload="init()">
<span id="hMsg">Html Message</span><br />
<span id="jMsg">JavaScript Message</span><br />
<span id="sMsg">Server Message</span>
</body>
</html>
```

由此可见我们得到的 HTML 文档时执行完 JavaScript 程序后的文档，其中包含了 jMsg 与 sMsg 的信息，显然如果用这个文档进行数据爬取就可以爬取到 jMsg 与 sMsg 的信息了。

现在很多网站额网页都不是静态的 HTML 文档，大部分都包含 JsvaScript 程序，很多信息都是通过 JavaScript 程序处理后才显示出来的，因此我们要使用模拟浏览器访问网站的方法来获取网页文档。

7、 改进的爬虫程序

模拟浏览器访问网站的方法来获取网页文档，然后从中爬取要的数据，这样的爬虫程序功能就比较强大了，为此设计爬虫程序 spider.py 如下：

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from bs4 import BeautifulSoup
chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
chrome = webdriver.Chrome(chrome_options=chrome_options)
chrome.get("http://127.0.0.1:5000")
html=chrome.page_source
soup=BeautifulSoup(html,"lxml")
hMsg=soup.find("span",attrs={"id":"hMsg"}).text
print(hMsg)
jMsg=soup.find("span",attrs={"id":"jMsg"}).text
print(jMsg)
sMsg=soup.find("span",attrs={"id":"sMsg"}).text
print(sMsg)
```

执行该程序，结果我们爬取到了所有数据：

```
Html Message
JavaScript Message
Server Message
```

严禁复制