5.5 Selenium 爬取 Ajax 网页数据

任务：

现在的网页中大量使用了 Ajax 技术，通过 JavaScript 在客户端向服务器发出请求，服务器返回数据给客户端，客户端再把数据展现出来，这样做可以减少网页的闪动，让用户有更好的体验。我们先设计一个这样的网页，然后使用 Selenium 编写爬虫程序爬取网页的数据。

5.3.1 创建 Ajax 的网站

1、创建网页文件

在 templates 文件夹中创建网页文件 phone.html，它的内容如下：

```
<script>
    function init()
    {
        var marks=new Array("华为","苹果","三星");
        var selm=document.getElementById("marks");
        for(var i=0;i<marks.length;i++)
        {
            selm.options.add(new Option(marks[i],marks[i]));
        }
        selm.selectedIndex=0;
        display();
    }

    function display()
    {
        try
        {
            var http=new XMLHttpRequest();
            var selm=document.getElementById("marks");
            var m=selm.options[selm.selectedIndex].text;
            http.open("get","/phones?mark="+m,false);
            http.send(null);
            msg=http.responseText;
            obj=eval("("+msg+")");
            s="<table width='200' border='1'><tr><td>型号</td><td>价格</td></tr>"
            for(var i=0;i<obj.phones.length;i++)
            {

s=s+"<tr><td>"+obj.phones[i].model+"</td><td>"+obj.phones[i].price+"</td></tr>";
            }
            s=s+"</table>";
            document.getElementById("phones").innerHTML=s;
        }
```

```
        catch(e) { alert(e); }
    }

</script>
<body onload="init()">
<div>选择品牌<select id="marks" onchange="display()"></select></div>
<div id="phones"></div>
</body>
```

说明：

(1) 网页框架

这个网页的主体框架很简单，只有两个 元素，一个包含了<select>选择列表，另外一个是显示信息使用的，在没有执行 JavaScript 之前内容都是空的。

(2) init 函数

在网页被加载时执行 init 函数，为<select >增加了三个手机品牌，即"华为"、"苹果"、"三星"。

(3) diaplay 函数

当用户选择其中一个品牌时就触发<select>的 onchange 事件，从而执行 display 函数，扎个函数通过 Ajax 技术把选择的手机品牌通过 http.open("get","/phones?mark="+m,false)语句发送给服务器，服务器收到该品牌手机后返回这个品牌的手机信息 http.responseText 给这个网页。返回的数据是 JSON 格式的字符串，经 eval 转换为 JavaScript 对象 phones 后生成一张表格的 HTML 代码放到 中显示。

2、创建服务器程序

服务器程序首先提交一个 phone.html 的网页，然后相应"/phones?mark=..."的请求，根据 mark 的值确定品牌，返回该品牌下的手机记录，返回的记录采用 JSON 数据格式，服务器程序如下：

```
import flask
import json
app=flask.Flask(__name__)
@app.route("/")
def index():
    return flask.render_template("phone.html")

@app.route("/phones")
def getPhones():
    mark=flask.request.values.get("mark")
    phones=[]
    if mark=="华为":
        phones.append({"model":"P9","mark":"华为","price":3800})
        phones.append({"model": "P10", "mark": "华为", "price": 4000})
    elif mark=="苹果":
        phones.append({"model":"iPhone5","mark":"苹果","price":5800})
        phones.append({"model":"iPhone6","mark":"苹果","price":6800})
    elif mark=="三星":
```

```
        phones.append({"model":"Galaxy A9","price":2800})
    s=json.dumps({"phones":phones})
    return s
if __name__=="__main__":
    app.run()
```

3、浏览器浏览

启动服务器程序浏览 web 地址"http://127.0.0.1:5000"，结果如图 5-3-1 所示，用户选择另外一个品牌后就触发 phone.html 中<select>的 onchange 事件，再次使用 Ajax 获取该品牌的手机记录进行显示。

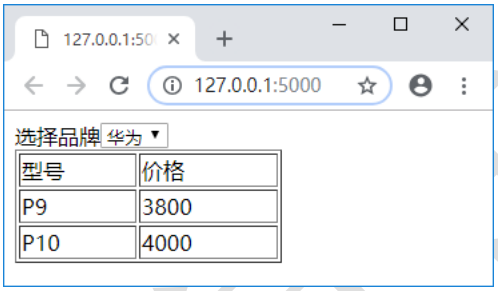

图 5-3-1 手机网站

5.3.2 理解 Selenium 爬虫程序

1、普通爬虫程序

编写下列程序获取网页的 HTML 代码：

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

执行该程序结果如下：

```
<script>
    function init()
    {
        var marks=new Array("华为","苹果","三星");
        var selm=document.getElementById("marks");
        for(var i=0;i<marks.length;i++)
        {
            selm.options.add(new Option(marks[i],marks[i]));
        }
        selm.selectedIndex=0;
        display();
    }

    function display()
    {
        try
        {
```

```
                var http=new XMLHttpRequest();
                var selm=document.getElementById("marks");
                var m=selm.options[selm.selectedIndex].text;
                http.open("get","/phones?mark="+m,false);
                http.send(null);
                msg=http.responseText;
                obj=eval("("+msg+")");
                s="<table width='200' border='1'><tr><td>型号</td><td>价格</td></tr>"
                for(var i=0;i<obj.phones.length;i++)
                {

s=s+"<tr><td>"+obj.phones[i].model+"</td><td>"+obj.phones[i].price+"</td></tr>";
                }
                s=s+"</table>";
                document.getElementById("phones").innerHTML=s;
            }
            catch(e) { alert(e); }
        }
    </script>
    <body onload="init()">
    <div>选择品牌<select id="marks" onchange="display()"></select></div>
    <div id="phones"></div>
    </body>
```

显然如果要爬取这个 Web 网站的所有手机记录，采用简单的爬虫程序方法是得不到的，因为在网页的 HTML 代码中根本就看不出任何手机的记录信息，网页的 HTML 就与 phone.html 的内容完全一样。

实际在用 BeautifulSoup 解析时 BeautifulSoup 根据网页的 HTML 代码建立了这棵树。在使用 scrapy 的 Selector 解析时 Selector 也按网页的 HTML 代码建立了这棵树。只是 BeautifulSoup 与 Selector 都只能根据静态的 HTML 代码构建这棵树，它们都不能使用 JavaScript 构建树。

2、Selenium 爬虫程序

但是 Selenium 就不一样，它是一个浏览器，在浏览一个网页后会在它内部构建一棵 HTML 元素结构树，编写下列程序查看 HTML 代码：

```
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')
driver = webdriver.Chrome(chrome_options=chrome_options)
driver.get("http://127.0.0.1:5000")
print(driver.page_source)
driver.close()
```

执行该程序结果如下：

```
<html xmlns="http://www.w3.org/1999/xhtml"><head><script>
    function init()
    {
        var marks=new Array("华为","苹果","三星");
        var selm=document.getElementById("marks");
        for(var i=0;i&lt;marks.length;i++)
        {
            selm.options.add(new Option(marks[i],marks[i]));
        }
        selm.selectedIndex=0;
        display();
    }

    function display()
    {
        try
        {
            var http=new XMLHttpRequest();
            var selm=document.getElementById("marks");
            var m=selm.options[selm.selectedIndex].text;
            http.open("get","/phones?mark="+m,false);
            http.send(null);
            msg=http.responseText;
            obj=eval("("+msg+")");
            s="&lt;table width='200' border='1'&gt;&lt;tr&gt;&lt;td&gt; 型 号
&lt;/td&gt;&lt;td&gt;价格&lt;/td&gt;&lt;/tr&gt;"
            for(var i=0;i&lt;obj.phones.length;i++)
            {

s=s+"&lt;tr&gt;&lt;td&gt;"+obj.phones[i].model+"&lt;/td&gt;&lt;td&gt;"+obj.phones[i].price+"&lt
;/td&gt;&lt;/tr&gt;";
            }
            s=s+"&lt;/table&gt;";
            document.getElementById("phones").innerHTML=s;
        }
        catch(e) { alert(e); }
    }
</script>
</head><body onload="init()">
<div>选择品牌<select id="marks" onchange="display()"><option value="华为">华为
</option><option value="苹果">苹果</option><option value="三星">三星
</option></select></div>
<div id="phones"><table width="200" border="1"><tbody><tr><td>型号</td><td>价格
</td></tr><tr><td>P9</td><td>3800</td></tr><tr><td>P10</td><td>4000</td></tr></tbody></
```

table>
</body></html>

由此可见这棵树的元素包括静态 HTML 的元素，也包括从 JavaScript 执行后产生的元素，Selenium 会按这棵树去查找各个元素。

5.5.3 编写爬虫程序

使用 Selenium 模拟浏览器去浏览网页，然后再模拟用户选择<select>中各个手机品牌的过程实现换页，就可以逐页爬取所有的手机数据了，爬虫程序过程如下：

(1) 创建一个浏览器对象 driver，使用这个 driver 对象模拟浏览器。

(2) 访问 http://127.0.0.1:5000 网站，爬取第一个页面的手机数据。

(3) 从第一个页面中获取<select>中所有的选择项目 options。

(4) 循环 options 中的每个 option,并模拟这个 option 的 click 点击动作，触发 onchange 事件调用 display()函数，爬取每个页面的手机数据。

根据这个规则，编写爬虫程序如下：

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

def spider(index):
    trs=driver.find_elements_by_tag_name("tr")
    for i in range(1,len(trs)):
        tds=trs[i].find_elements_by_tag_name("td")
        model=tds[0].text
        price=tds[1].text
        print("%-16s%-16s" %(model,price))
    select=driver.find_element_by_id("marks")
    options=select.find_elements_by_tag_name("option")
    if index<len(options)-1:
        index+=1
        options[index].click()
        time.sleep(0.5)
        spider(index)

chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
driver = webdriver.Chrome(chrome_options=chrome_options)
driver.get("http://127.0.0.1:5000")
spider(0)
driver.close()
```

说明：

(1) 创建 chrome 对象：

```
driver = webdriver.Chrome(chrome_options=chrome_options)
```

然后浏览网站的第一页：

driver.get("http://127.0.0.1:5000")

(2) spider 函数负责爬取当前页面的所有手机记录，其中：

trs=driver.find_elements_by_tag_name("tr")

是获取所用的<tr>元素，在网页中有很多个<tr>元素，然后通过循环去获取每个<tr>元素，程序跳过第一个<tr>的表格头，从第二个<tr>开设是手机的记录。

再通过：

tds=trs[i].find_elements_by_tag_name("td")

获取<tr>下面的所有<td>元素，每个<tr>下面有两个<td>元素，第一个是手机的型号，第二个是价格：

model=tds[0].text

price=tds[1].text

这样就爬取到了各个手机记录。

(3) 程序通过：

select=driver.find_element_by_id("marks")

获取网页中的<select>元素，再次通过：

options=select.find_elements_by_tag_name("option")

获取该元素下所有的<option>元素。

(4) 程序使用 index 变量记录是第几个<option>被选择，在一个页面被爬取后 index 增加 1，重新获取 options 列表，调用 options[index].click()选择新页面继续爬取数据，一直到最后一个<option>被选择为止，因此 spider 被递归调用：

```
if index<len(options)-1:
    index+=1
    options[index].click()
    spider(index)
```

其中 options[i].click()是一个模拟用户点击该<option>的动作，它会触发<select>的 onchange 事件，从而执行 display()函数，用 Ajax 从服务器获取该手机品牌的手机记录，再次调用 spider()就可以爬取。

5.5.4 执行爬虫程序

执行该程序爬取到了所有的手机记录如下：

```
P9              3800
P10             4000
iPhone5         5800
iPhone6         6800
Galaxy A9       2800
```

这些手机记录也就是各个页面的手机记录的总和，说明爬虫程序爬取成功。