java 爬虫抓取网页数据教程

数据是科研活动重要的基础，而爬虫是获取数据一个比较常见的方法，爬虫的基本原理很简单，就是利用程序访问互联网，然后将数据保存到本地中。我们都知道，互联网提供的服务大多数是以网站的形式提供的。

我们需要的数据一般都是从网站中获取的，如电商网站商品信息、商品的评论、微博的信息等。爬虫和我们手动将看到的数据复制粘贴下来是类似的，只是获取大量的数据靠人工显然不太可能。因此，需要我们使用工具来帮助获取知识。

使用程序编写爬虫就是使用程序编写一些网络访问的规则，将我们的目标数据保存下来。Java 作为爬虫语言的一种，下面为大家介绍 java 爬虫抓取网页数据教程。

1、使用 HttpClient 简单抓取网页

首先，假设我们需要爬取数据学习网站上第一页的博客（http://www.datalearner.com/blog ）。首先，我们需要使用导入 HttpClient 4.5.3 这个包（这是目前最新的包，你可以根据需要使用其他的版本）。

Java 本身提供了关于网络访问的包，在 java.net 中，然后它不够强大。于是 Apache 基金会发布了开源的 http 请求的包，即 HttpClient，这个包提供了非常多的网络访问的功能。在这里，我们也是使用这个包来编写爬虫。好了，使用 pom.xml 下载完这个包之后我们就可以开始编写我们的第一个爬虫例子了。其代码如下（注意，我们的程序是建立在 test 包下面的，因此，需要在这个包下才能运行）：

```
package test;

import org.apache.http.HttpEntity;import

org.apache.http.client.methods.CloseableHttpResponse;impor

t org.apache.http.client.methods.HttpGet;import
```

```
org.apache.http.impl.client.CloseableHttpClient;import

org.apache.http.impl.client.HttpClients;import

org.apache.http.util.EntityUtils;

import java.io.IOException;

/**
 * 第一个爬虫测试
 * Created by DuFei on 2017/7/27.
 */public class FirstTest {


  public static void main(String[] args) {


    //建立一个新的请求客户端
    CloseableHttpClient httpClient =
HttpClients.createDefault();

    //使用 HttpGet 方式请求网址

    HttpGet httpGet = new

HttpGet("http://www.datalearner.com/blog");


    //获取网址的返回结果

    CloseableHttpResponse response = null;


    try {

      response = httpClient.execute(httpGet);

    } catch (IOException e) {

      e.printStackTrace();
    }

    //获取返回结果中的实体
```

```
    HttpEntity entity = response.getEntity();

    //将返回的实体输出
    try {
      System.out.println(EntityUtils.toString(entity));
      EntityUtils.consume(entity);
    } catch (IOException e) {
      e.printStackTrace();
    }

  }
}
```

如上面的代码所示，爬虫的第一步需要构建一个客户端，即请求端，我们这里使用 CloseableHttpClient 作为我们的请求端，然后确定使用哪种方式请求什么网址，再然后使用 HttpResponse 获取请求的地址对应的结果即可。最后取出 HttpEntity 转换一下就可以得到我们请求的网址对应的内容了。上述程序对应的输出如下图所示：

```
<!DOCTYPE html>
<html lang="zh">
<head>
  <base href="/">
  <title>统计、机器学习与编程知识的原创博客简介 | 学习数据(Datalearner)</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0,user-scalable=no">
  <meta name="apple-touch-fullscreen" content="yes">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="pragma" content="no-cache" />
  <meta http-equiv="cache-control" content="max-age=3600" />
  <meta http-equiv="expires" content="0" />
  <meta http-equiv="keywords" content="数据分析,文献综述,自然语言处理,数据挖掘,统计,文本处理,文献推荐">
  <meta http-equiv="description" content="统计、机器学习与编程知识的原创博客,数据学习网站博客将提供优秀的数据挖掘，数据分析，贝叶斯

  <link href="/resources/css/bootstrap.min.css" rel="stylesheet">
  <link href="/resources/css/font-awesome.min.css" rel="stylesheet">
  <link href="/resources/css/main.css" rel="stylesheet">

  <style type="text/css">
    #sort_bar div{
      line-height:30px;
    }
    span{
      font: normal 14px/24px \5FAE\8F6F\96C5\9ED1;
```

显然，这就是我们需要的网址对应的页面的源代码。于是我们的第一个爬虫就成功的将网门需要的页面的内容下载下来了。

2、HttpClient 的详细使用

在上篇博客里面，我们讲述了如何使用 HttpClient 请求一个简单的网页。但是，在实际中，有很多网页的请求需要附带许多参数设置。主要包括请求的 Header 设置以及路径参数。在 HttpClient 4.3 及以上的版本中，这个过程主要包含如下步骤：

使用 List<NameValuePair>添加路径参数（请求参数）

使用 URI 对请求路径及其参数进行设置

使用 List<Header>设置请求的头部

初始化自定义的 HttpClient 客户端，并设置头部

使用 HttpUriRequest 设置请求

使用 HttpClient 请求上述步骤中的 HttpUriRequest 对象

我们看一个代码示例

```
import com.google.common.collect.Lists;import
org.apache.http.Header;import
org.apache.http.HttpHeaders;import
org.apache.http.HttpResponse;import
org.apache.http.NameValuePair;import
org.apache.http.client.HttpClient;import
org.apache.http.client.methods.HttpUriRequest;import
org.apache.http.client.methods.RequestBuilder;import
org.apache.http.client.utils.URIBuilder;import
org.apache.http.impl.client.HttpClients;import
org.apache.http.message.BasicHeader;import
org.apache.http.message.BasicNameValuePair;
import java.io.IOException;import java.net.URI;import
java.net.URISyntaxException;import java.util.List;
/**********
 * HttpClient 使用示例
 * **********/
```

```
public class HttpClientTest {

  public static void main(String[] args) throws
URISyntaxException, IOException {

    String url = "";    //请求路径

    //构造路径参数
    List<NameValuePair> nameValuePairList =
Lists.newArrayList();
    nameValuePairList.add(new

BasicNameValuePair("username","test"));

    nameValuePairList.add(new

BasicNameValuePair("password","password"));

    //构造请求路径，并添加参数
    URI uri = new

URIBuilder(url).addParameters(nameValuePairList).build();

    //构造 Headers
    List<Header> headerList = Lists.newArrayList();
    headerList.add(new

BasicHeader(HttpHeaders.ACCEPT_ENCODING,"gzip, deflate"));

    headerList.add(new BasicHeader(HttpHeaders.CONNECTION,

"keep-alive"));
```

```
    //构造 HttpClient
    HttpClient httpClient =
HttpClients.custom().setDefaultHeaders(headerList).build()
;

    //构造 HttpGet 请求
    HttpUriRequest httpUriRequest =
RequestBuilder.get().setUri(uri).build();

    //获取结果
    HttpResponse httpResponse =
httpClient.execute(httpUriRequest);

    //获取返回结果中的实体
    HttpEntity entity = httpResponse.getEntity();

    //查看页面内容结果
    String rawHTMLContent = EntityUtils.toString(entity);

    System.out.println(rawHTMLContent);

    //关闭 HttpEntity 流
    EntityUtils.consume(entity);

  }

}
```

这种方式可以使我们一次性构造一个统一头部的 HttpClient，后面所有的请求都可以使用带有这个 Headers 的 HttpClient。非常简单方便。

3、将下载的网页解析，转换成结构化数据

上一个步骤介绍获取了 http://www.datalearner.com/blog_list 页面的 HTML 源码，但是这些源码是提供给浏览器解析用的，我们需要的数据其实是页面上博客的标题、作者、简介、发布日期等。我们需要通过一种方式来从 HTML 源码中解

析出这类信息并提取，然后存到文本或者数据库之中。在这篇博客中，我们将介绍使用 Jsoup 包帮助我们解析页面，提取数据。

Jsoup 是一款 Java 的 HTML 解析器，可以直接解析某个 URL 地址，也可以解析 HTML 内容。其主要的功能包括解析 HTML 页面，通过 DOM 或者 CSS 选择器来查找、提取数据，可以更改 HTML 内容。Jsoup 的使用方式也很简单，使用 Jsoup.parse(String str)方法将之前我们获取到的 HTML 内容进行解析得到一个 Documend 类，剩下的工作就是从 Document 中选择我们需要的数据了。举个例子，假设我们有个 HTML 页面的内容如下：

```
<html>
    <div id="blog_list">
        <div class="blog_title">
            <a href="url1">第一篇博客</a>
        </div>
        <div class="blog_title">
            <a href="url2">第二篇博客</a>
        </div>
        <div class="blog_title">
            <a href="url3">第三篇博客</a>
        </div>
    </div></html>
```

通过 Jsoup 我们可以把上面的三篇博客的标题提取到一个 List 中。使用方法如下：

首先，我们通过 maven 把 Jsoup 引入进来

```
<!-- https://mvnrepository.com/artifact/org.jsoup/jsoup
--><dependency>

    <groupId>org.jsoup</groupId>

    <artifactId>jsoup</artifactId>

    <version>1.10.3</version></dependency>
```

然后编写 Java 进行解析。

```
package org.hfutec.example;

import org.jsoup.Jsoup;import org.jsoup.nodes.Document;import

org.jsoup.nodes.Element;import org.jsoup.select.Elements;

import java.util.ArrayList;import java.util.List;

/*******
 * created by DuFei at 2017.08.25 21:00
 * web crawler example
 * ******/
public class DataLearnerCrawler {


  public static void main(String[] args) {


    List<String> titles = new ArrayList<String>();

    List<String> urls = new ArrayList<String>();

    //假设我们获取的HTML的字符内容如下

    String html = "<html><div id=\"blog_list\"><div

class=\"blog_title\"><a href=\"url1\">第一篇博客</a></div><div
class=\"blog_title\"><a href=\"url2\">第二篇博客</a></div><div
class=\"blog_title\"><a href=\"url3\">第三篇博客

</a></div></div></html>";


    //第一步，将字符内容解析成一个Document类
    Document doc = Jsoup.parse(html);

    //第二步，根据我们需要得到的标签，选择提取相应标签的内容
    Elements elements =

doc.select("div[id=blog_list]").select("div[class=blog_title]");
```

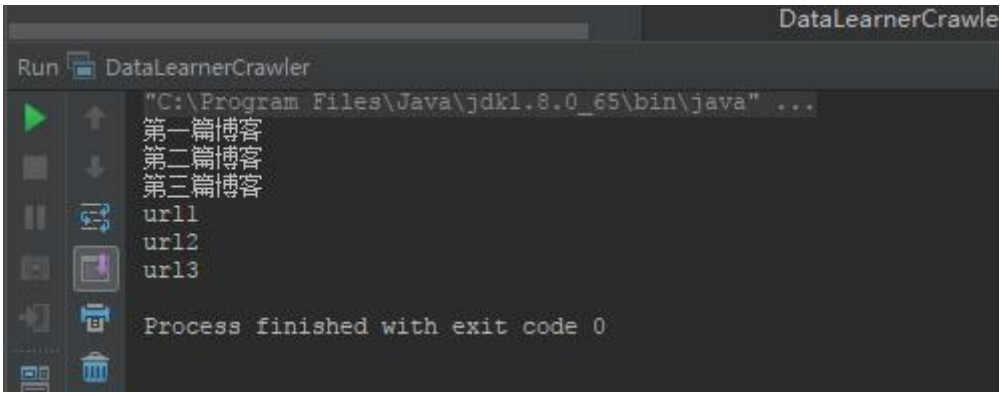
```
    for( Element element : elements ){
      String title = element.text();
      titles.add(title);
      urls.add(element.select("a").attr("href"));
    }

    //输出测试
    for( String title : titles ){
      System.out.println(title);
    }

    for( String url : urls ){
      System.out.println(url);
    }

  }
}
```

我们简单说明一下 Jsoup 的解析过程。首先第一步都是调用 parse()方法将字符对象变成一个 Document 对象，然后我们对这个对象进行操作。一般提取数据就是根据标签选择数据，使用 select()方法语法格式和 javascript/css 选择器都是一样的。一般都是提取某个标签，其属性值为指定内容。得到的结果是一个 Element 的集合，为 Elements（因为符合条件的标签可能很多，所以结果是一个集合）。select()方法可以一直进行下去，直到选择到我们想要的标签集合为止（注意，我们并不一定要按照标签层级一级一级往下选，可以直接写 select()方法到我们需要的标签的上一级，比如这里的示例代码可以直接写成 Elements elements = doc.select("div[class=blog_title]"); 其效果是一样的）。对于选择到的 Elements 的集合，我们可以通过循环的方式提取每一个需要的数据，比如，我们需要拿到标签的文本信息，就可以使用 text()方法，如果我们需要拿到对应的 HTML 属性信息，我们可以使用 attr()方法。我们可以看到上述方法的输出结果如下：



更多的 Jsoup 解析的操作可以参考如下：

1、https://www.ibm.com/developerworks/cn/java/j-lo-jsouphtml/index.html

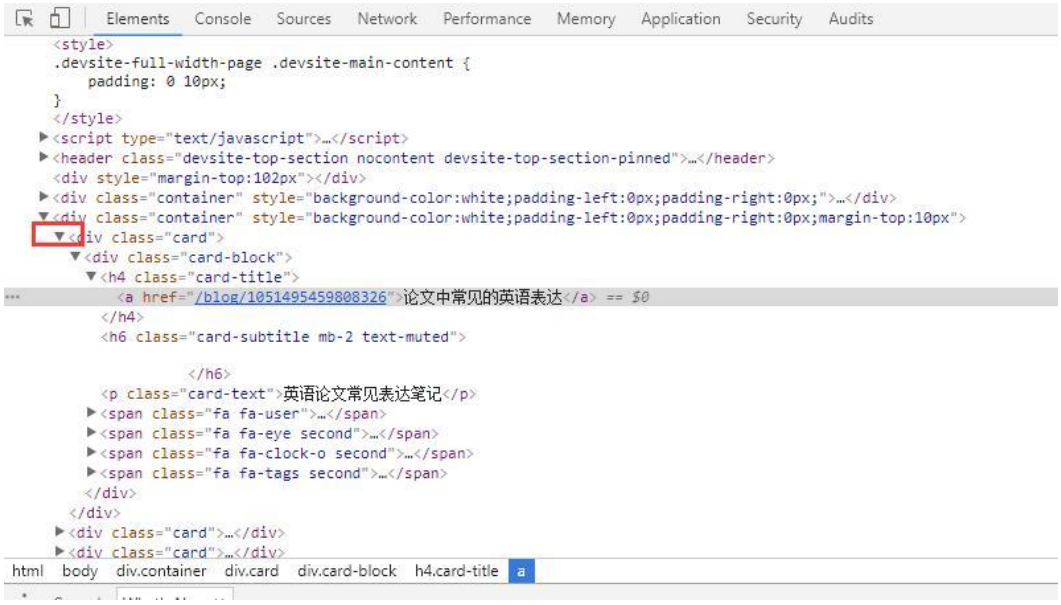
2、https://jsoup.org/

一个实例

我们接着上一个爬取数据学习官方网站博客列表的例子讲解一个实例。我们已经知道可以使用 Jsoup 来解析爬取到的 HTML 页面内容。那么如何查看我们需要的内容对应的标签呢？以 Chrome 浏览器为例，我们需要爬取 http://www.datalearner.com/blog_list 这个页面的的博客，首先用 Chrome 浏览器打开这个网址，然后鼠标右键单击博客的标题，点击“检查”就可以得到 HTML 页面了。如下图所示。


论文中常见的英语表达
英语论文常见
小木
英语# #论文#
在新标签页中打开链接(T)
在新窗口中打开链接(W)
在隐身窗口中打开链接(G)
链接另存为(K)...
复制链接地址(E)
检查(N)
Ctrl+Shift+I
R语言技巧总结笔记
蛋疼的R语言
小木
103
2017/08/22 15:49:43
#R语言# #编程#
常见的正则表达式写法总结

图 2 右键单击标题


Elements Console Sources Network Performance Memory Application Security Audits
<style>
.devsite-full-width-page .devsite-main-content {
padding: 0 10px;
}
</style>
<script type="text/javascript">…</script>
<header class="devsite-top-section nocontent devsite-top-section-pinned">…</header>

…

<h4 class="card-title">
论文中常见的英语表达 == $0
</h4>
<h6 class="card-subtitle mb-2 text-muted">
</h6>
英语论文常见表达笔记
…
…
…
…

…
…
html body div.container div.card div.card-block h4.card-title a

图 3 点击所在元素的父级元素边上的小三角，收起代码查看

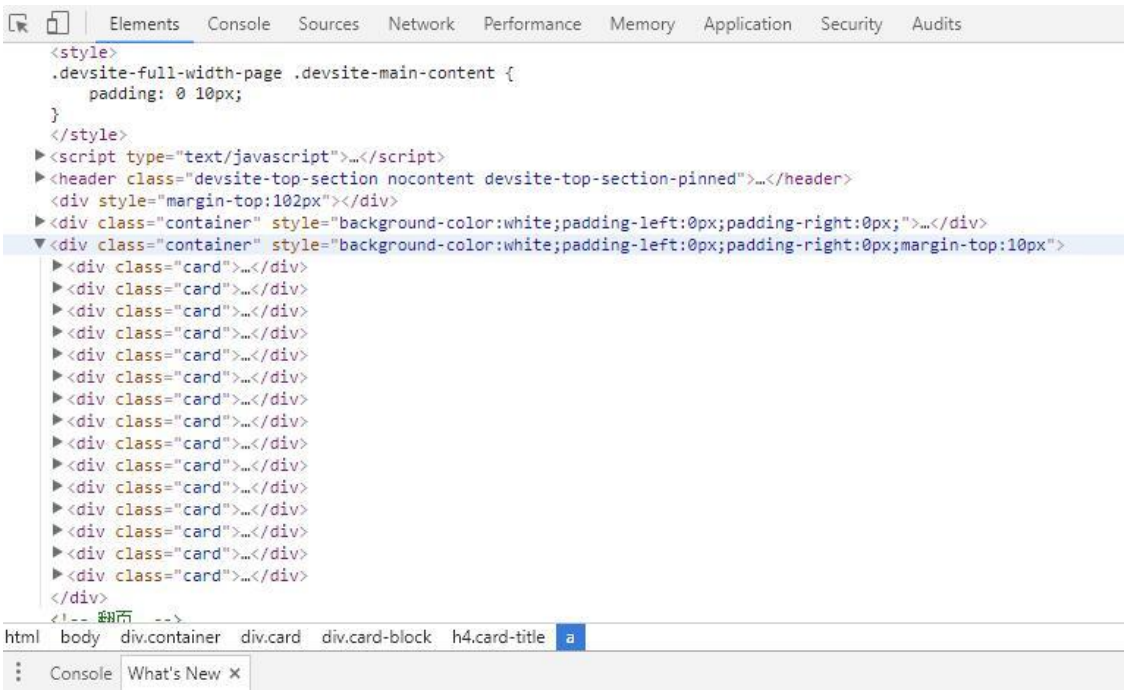
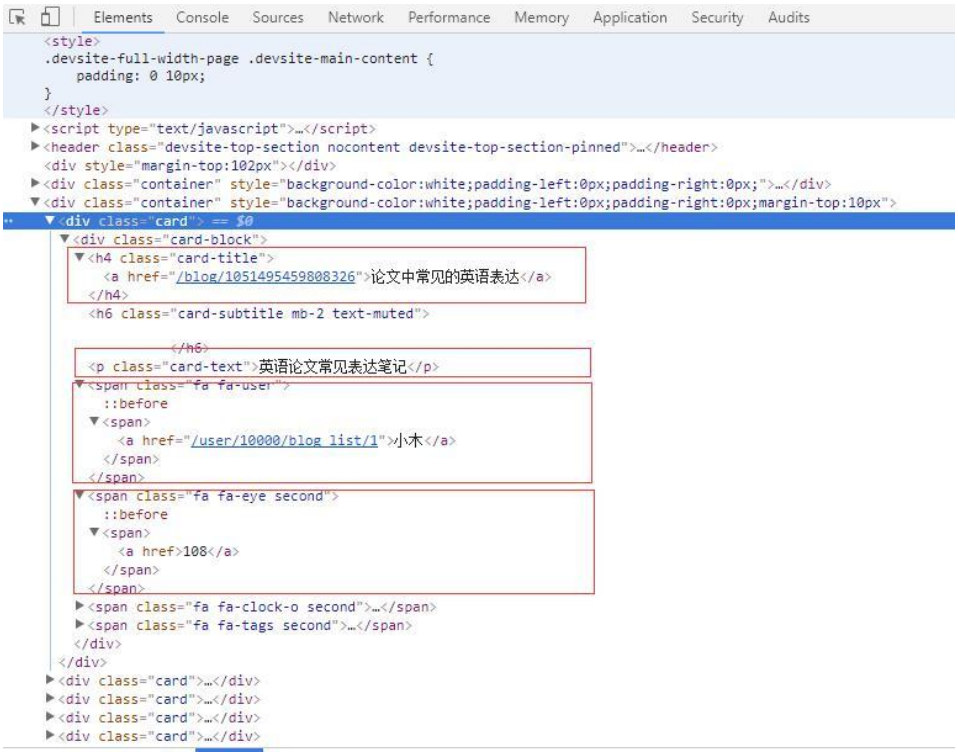


图 4 确认当前博客的 HTML 代码的一致性

通过上述操作之后，我们已经可以看到，所有的博客的标题等信息都存在 class=card 的 div 里面了。于是，我们只要关注这个标签里面的内容是如何组织的，就可以了。如下图所示，我们需要的信息所属的标签，通过点击小三角展开就能得到了。



因此，解析博客列表的代码可以写成如下形式了。

```
package org.hfutec.example;

import org.apache.http.HttpEntity;import

org.apache.http.client.methods.CloseableHttpResponse;impor

t org.apache.http.client.methods.HttpGet;import

org.apache.http.impl.client.CloseableHttpClient;import

org.apache.http.impl.client.HttpClients;import

org.apache.http.util.EntityUtils;import
```

```
org.jsoup.Jsoup;import org.jsoup.nodes.Document;import

org.jsoup.nodes.Element;import org.jsoup.select.Elements;

import java.io.IOException;

/*******
 * created by DuFei at 2017.08.25 21:00
 * web crawler example
 * ******/
public class DataLearnerCrawler {

  public static void main(String[] args) {


    String url = "http://www.datalearner.com/blog_list";

    String rawHTML = null;

    try {
      rawHTML = getHTMLContent(url);
    } catch (IOException e) {
      e.printStackTrace();
    }

    //将当前页面转换成 Jsoup 的 Document 对象
    Document doc = Jsoup.parse(rawHTML);

    //获取所有的博客列表集合
    Elements blogList = doc.select("div[class=card]");

    //针对每个博客内容进行解析，并输出
    for( Element element : blogList ){
```

```
        String title =
element.select("h4[class=card-title]").text();
        String introduction =
element.select("p[class=card-text]").text();
        String author = element.select("span[class=fa
fa-user]").text();

        System.out.println("Title:\t"+title);
        System.out.println("introduction:\t"+introduction);
        System.out.println("Author:\t"+author);
        System.out.println("------------------------");

    }
  }
```

//根据 url 地址获取对应页面的 HTML 内容，我们将上一节中的内容打包成了一个方法，方便调用

```
  private static String getHTMLContent( String url ) throws
IOException {

    //建立一个新的请求客户端
    CloseableHttpClient httpClient =
HttpClients.createDefault();

    //使用 HttpGet 方式请求网址
    HttpGet httpGet = new HttpGet(url);
```

```
    //获取网址的返回结果
    CloseableHttpResponse response =
httpClient.execute(httpGet);

    //获取返回结果中的实体
    HttpEntity entity = response.getEntity();

    String content = EntityUtils.toString(entity);

    //关闭 HttpEntity 流
    EntityUtils.consume(entity);


    return content;

  }

}
```

最终的输出结果如下图所示：