FREE eBook

LEARNING Jsoup

Free unaffiliated eBook created from
Stack Overflow contributors.

#jsoup

Table of Contents

About ... 1

Chapter 1: Getting started with Jsoup ... 2

Remarks ... 2

JavaScript support ... 2

Official website & documentation ... 2

Download ... 2

Versions ... 3

Examples ... 3

Extract the URLs and titles of links ... 3

Extract full URL from partial HTML ... 4

Extract the data from HTML document file ... 4

Chapter 2: Formatting HTML Output ... 6

Parameters ... 6

Remarks ... 6

Examples ... 6

Display all elements as block ... 6

Chapter 3: Logging into websites with Jsoup ... 8

Examples ... 8

A simple authentication POST request with Jsoup ... 8

A more comprehensive authentication POST request with Jsoup ... 8

Logging with FormElement ... 9

Chapter 4: Parsing Javascript Generated Pages ... 11

Examples ... 11

Parsing JavaScript Generated Page with Jsoup and HtmUnit ... 11

Chapter 5: Selectors ... 13

Remarks ... 13

Examples ... 14

Selecting elements using CSS selectors ... 14

Extract Twitter Markup ... 15

Chapter 6: Web crawling with Jsoup ... 17

Examples 17
Extracting email adresses & links to other pages 17
Extracting JavaScript data with Jsoup 17
Extracting all the URLs from a website using JSoup (recursion) 18
Credits 20

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: jsoup

It is an unofficial and free Jsoup ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Jsoup.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Jsoup

Remarks

Jsoup is a HTML parsing and data extraction library for Java, focused on flexibility and ease of use. It can be used to extract sepecific data from HTML pages, which is commonly known as "web scraping", as well as modify the content of HTML pages, and "clean" untrusted HTML with a whitelist of allowed tags and attributes.

JavaScript support

Jsoup does not support JavaScript, and, because of this, any dynamically generated content or content which is added to the page after page load cannot be extracted from the page. If you need to extract content which is added to the page with JavaScript, there *are* a few alternative options:

- Use a library which does support JavaScript, such as Selenium, which uses an an actual web browser to load pages, or HtmlUnit.
- Reverse engineer how the page loads it's data. Typically, web pages which load data dynamically do so via AJAX, and thus, you can look at the network tab of your browser's developer tools to see where the data is being loaded from, and then use those URLs in your own code. See how to scrape AJAX pages for more details.

Official website & documentation

You can find various Jsoup related resources at **jsoup.org**, including the Javadoc, usage examples in the Jsoup cookbook and JAR downloads. See **the GitHub repository** for the source code, issues, and pull requests.

Download

Jsoup is available on Maven as `org.jsoup.jsoup:jsoup`, If you're using Gradle (eg. with Android Studio), you can add it to your project by adding the following to your `build.gradle` dependencies section:

```
compile 'org.jsoup:jsoup:1.8.3'
```

If you're using Ant (Eclipse), add the following to your POMs dependencies section:

```
<dependency>
  <!-- jsoup HTML parser library @ http://jsoup.org/ -->
  <groupId>org.jsoup</groupId>
  <artifactId>jsoup</artifactId>
  <version>1.8.3</version>
```

```
</dependency>
```

Jsoup is also available as **downloadable JAR** for other environments.

Versions

Version	Release Date
1.9.2	2016-05-17
1.8.3	2015-08-02

Examples

Extract the URLs and titles of links

Jsoup can be be used to easily extract all links from a webpage. In this case, we can use Jsoup to extract only specific links we want, here, ones in a `h3` header on a page. We can also get the text of the links.

```
Document doc = Jsoup.connect("http://stackoverflow.com").userAgent("Mozilla").get();
for (Element e: doc.select("a.question-hyperlink")) {
    System.out.println(e.attr("abs:href"));
    System.out.println(e.text());
    System.out.println();
}
```

This gives the following output:

```
http://stackoverflow.com/questions/12920296/past-5-week-calculation-in-webi-bo-4-0
Past 5 week calculation in WEBI (BO 4.0)?

http://stackoverflow.com/questions/36303701/how-to-get-information-about-the-visualized-
elements-in-listview
How to get information about the visualized elements in listview?

[...]
```

What's happening here:

- First, we get the HTML document from the specified URL. This code also sets the User Agent header of the request to "Mozilla", so that the website serves the page it would usually serve to browsers.
- Then, use `select(...)` and a for loop to get all the links to Stack Overflow questions, in this case links which have the class `question-hyperlink`.
- Print out the text of each link with `.text()` and the href of the link with `attr("abs:href")`. In this

case, we use `abs:` to get the *absolute* URL, ie. with the domain and protocol included.

Extract full URL from partial HTML

Selecting only the attribute value of a link:href will return the relative URL.

```
   String bodyFragment =
          "<div><a href=\"/documentation\">Stack Overflow Documentation</a></div>";


    Document doc = Jsoup.parseBodyFragment(bodyFragment);
    String link = doc
            .select("div > a")
            .first()
            .attr("href");

    System.out.println(link);
```

Output

```
/documentation
```

By passing the base URI into the `parse` method and using the `absUrl` method instead of `attr`, we can extract the full URL.

```
    Document doc = Jsoup.parseBodyFragment(bodyFragment, "http://stackoverflow.com");

    String link = doc
                .select("div > a")
                .first()
                .absUrl("href");

    System.out.println(link);
```

Output

```
http://stackoverflow.com/documentation
```

Extract the data from HTML document file

Jsoup can be used to manipulate or extract data from a **file** on local that contains HTML. `filePath` is path of a file on disk. `ENCODING` is desired Charset Name e.g. "Windows-31J". It is optional.

```
// load file
   File inputFile = new File(filePath);
   // parse file as HTML document
   Document doc = Jsoup.parse(filePath, ENCODING);
   // select element by <a>
   Elements elements = doc.select("a");
```

Read Getting started with Jsoup online: https://riptutorial.com/jsoup/topic/297/getting-started-with-

jsoup

Chapter 2: Formatting HTML Output

Parameters

Parameter	Detail
`boolean outline()`	Get if outline mode is enabled. Default is false. If enabled, the HTML output methods will consider all tags as block.
`Document.OutputSettings outline(boolean)`	Enable or disable HTML outline mode.

Remarks

Jsoup 1.9.2 API

Examples

Display all elements as block

By default, Jsoup will display only block-level elements with a trailing line break. Inline elements are displayed without a line break.

Given a body fragment, with inline elements:

```
<select name="menu">
    <option value="foo">foo</option>
    <option value="bar">bar</option>
</select>
```

Printing with Jsoup:

```
Document doc = Jsoup.parse(html);

System.out.println(doc.html());
```

Results in:

```
<html>
 <head></head>
 <body>
  <select name="menu"> <option value="foo">foo</option> <option value="bar">bar</option>
</select>
 </body>
</html>
```

To display the output with each element treated as a block element, the `outline` option has to be

enabled on the document's `OutputSettings`.

```
Document doc = Jsoup.parse(html);

doc.outputSettings().outline(true);

System.out.println(doc.html());
```

Output

```
<html>
 <head></head>
 <body>
  <select name="menu">
   <option value="foo">foo</option>
   <option value="bar">bar</option>
  </select>
 </body>
</html>
```

Source: JSoup - Formatting the <option> elements

Read Formatting HTML Output online: https://riptutorial.com/jsoup/topic/5954/formatting-html-output

Chapter 3: Logging into websites with Jsoup

Examples

A simple authentication POST request with Jsoup

A simple POST request with authentication data is demonstrated below, note that the `username` and `password` field will vary depending on the website:

```
final String USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/51.0.2704.103 Safari/537.36";
Connection.Response loginResponse = Jsoup.connect("yourWebsite.com/loginUrl")
                                         .userAgent(USER_AGENT)
                                         .data("username", "yourUsername")
                                         .data("password", "yourPassword")
                                         .method(Method.POST)
                                         .execute();
```

A more comprehensive authentication POST request with Jsoup

Most websites require a much more complicated process than the one demonstrated above.

Common steps for logging into a website are:

1. Get the unique `cookie` from the initial login form.
2. Inspect the login form to see what the destination url is for the authentication request
3. Parse the login form to check for any `security token` that needs to be sent along with username and password.
4. Send the request.

Below is an example request that will log you into the GitHub website

```
// # Constants used in this example
final String USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/51.0.2704.103 Safari/537.36";
final String LOGIN_FORM_URL = "https://github.com/login";
final String LOGIN_ACTION_URL = "https://github.com/session";
final String USERNAME = "yourUsername";
final String PASSWORD = "yourPassword";

// # Go to login page and grab cookies sent by server
Connection.Response loginForm = Jsoup.connect(LOGIN_FORM_URL)
                                     .method(Connection.Method.GET)
                                     .userAgent(USER_AGENT)
                                     .execute();
Document loginDoc = loginForm.parse(); // this is the document containing response html
HashMap<String, String> cookies = new HashMap<>(loginForm.cookies()); // save the cookies to
be passed on to next request

// # Prepare login credentials
String authToken = loginDoc.select("#login > form > div:nth-child(1) >
```

```
input[type=\"hidden\"]:nth-child(2)")
                            .first()
                            .attr("value");

HashMap<String, String> formData = new HashMap<>();
formData.put("commit", "Sign in");
formData.put("utf8", "e2 9c 93");
formData.put("login", USERNAME);
formData.put("password", PASSWORD);
formData.put("authenticity_token", authToken);

// # Now send the form for login
Connection.Response homePage = Jsoup.connect(LOGIN_ACTION_URL)
     .cookies(cookies)
     .data(formData)
     .method(Connection.Method.POST)
     .userAgent(USER_AGENT)
     .execute();

System.out.println(homePage.parse().html());
```

Logging with FormElement

In this example, we will log into the GitHub website by using the FormElement class.

```
// # Constants used in this example
final String USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/51.0.2704.103 Safari/537.36";
final String LOGIN_FORM_URL = "https://github.com/login";
final String USERNAME = "yourUsername";
final String PASSWORD = "yourPassword";

// # Go to login page
Connection.Response loginFormResponse = Jsoup.connect(LOGIN_FORM_URL)
                                             .method(Connection.Method.GET)
                                             .userAgent(USER_AGENT)
                                             .execute();

// # Fill the login form
// ## Find the form first...
FormElement loginForm = (FormElement)loginFormResponse.parse()
                                          .select("div#login > form").first();
checkElement("Login Form", loginForm);

// ## ... then "type" the username ...
Element loginField = loginForm.select("#login_field").first();
checkElement("Login Field", loginField);
loginField.val(USERNAME);

// ## ... and "type" the password
Element passwordField = loginForm.select("#password").first();
checkElement("Password Field", passwordField);
passwordField.val(PASSWORD);


// # Now send the form for login
Connection.Response loginActionResponse = loginForm.submit()
         .cookies(loginFormResponse.cookies())
         .userAgent(USER_AGENT)
```

```
        .execute();

System.out.println(loginActionResponse.parse().html());

public static void checkElement(String name, Element elem) {
    if (elem == null) {
        throw new RuntimeException("Unable to find " + name);
    }
}
```

All the form data is handled by the FormElement class for us (even the form method detection). A ready made Connection is built when invoking the FormElement#submit method. All we have to do is to complete this connection with addional headers (cookies, user-agent etc) and execute it.

Read Logging into websites with Jsoup online: https://riptutorial.com/jsoup/topic/4631/logging-into-websites-with-jsoup

Chapter 4: Parsing Javascript Generated Pages

Examples

Parsing JavaScript Generated Page with Jsoup and HtmUnit

page.html - source code

```
<html>
<head>
    <script src="loadData.js"></script>
</head>
<body onLoad="loadData()">
    <div class="container">
        <table id="data" border="1">
            <tr>
                <th>col1</th>
                <th>col2</th>
            </tr>
        </table>
    </div>
</body>
</html>
```

loadData.js

```
    // append rows and cols to table.data in page.html
    function loadData() {
        data = document.getElementById("data");
        for (var row = 0; row < 2; row++) {
            var tr = document.createElement("tr");
            for (var col = 0; col < 2; col++) {
                td = document.createElement("td");
                td.appendChild(document.createTextNode(row + "." + col));
                tr.appendChild(td);
            }
            data.appendChild(tr);
        }
    }
```

page.html when loaded to browser

Col1	Col2
0.0	0.1
1.0	1.1

Using jsoup to parse page.html for col data

```
    // load source from file
    Document doc = Jsoup.parse(new File("page.html"), "UTF-8");

    // iterate over row and col
    for (Element row : doc.select("table#data > tbody > tr"))

        for (Element col : row.select("td"))

            // print results
            System.out.println(col.ownText());
```

Output

(empty)

What happened?

Jsoup parses the source code as delivered from the server (or in this case loaded from file). It does not invoke client-side actions such as JavaScript or CSS DOM manipulation. In this example, the rows and cols are never appended to the data table.

How to parse my page as rendered in the browser?

```
    // load page using HTML Unit and fire scripts
    WebClient webClient = new WebClient();
    HtmlPage myPage = webClient.getPage(new File("page.html").toURI().toURL());

    // convert page to generated HTML and convert to document
    doc = Jsoup.parse(myPage.asXml());

    // iterate row and col
    for (Element row : doc.select("table#data > tbody > tr"))

        for (Element col : row.select("td"))

            // print results
            System.out.println(col.ownText());

    // clean up resources
    webClient.close();
```

Output

```
0.0
0.1
1.0
1.1
```

Read Parsing Javascript Generated Pages online: https://riptutorial.com/jsoup/topic/4632/parsing-javascript-generated-pages

Chapter 5: Selectors

Remarks

A selector is a chain of simple selectors, separated by combinators. Selectors are case insensitive (including against elements, attributes, and attribute values).

The universal selector (*) is implicit when no element selector is supplied (i.e. *.header and .header is equivalent).

Pattern	Matches	Example
`*`	any element	`*`
`tag`	elements with the given tag name	`div`
`ns|E`	elements of type E in the namespace ns	`fb|name finds <fb:name> elements`
`#id`	elements with attribute ID of "id"	`div#wrap, #logo`
`.class`	elements with a class name of "class"	`div.left, .result`
`[attr]`	elements with an attribute named "attr" (with any value)	`a[href], [title]`
`[^attrPrefix]`	elements with an attribute name starting with "attrPrefix". Use to find elements with HTML5 datasets	`[^data-], div[^data-]`
`[attr=val]`	elements with an attribute named "attr", and value equal to "val"	`img[width=500], a[rel=nofollow]`
`[attr="val"]`	elements with an attribute named "attr", and value equal to "val"	`span[hello="Cleveland"][goodbye="Columbus"], a[rel="nofollow"]`

Pattern	Matches	Example
`[attr^=valPrefix]`	elements with an attribute named "attr", and value starting with "valPrefix"	`a[href^=http:]`
`[attr$=valSuffix]`	elements with an attribute named "attr", and value ending with "valSuffix"	`img[src$=.png]`
`[attr*=valContaining]`	elements with an attribute named "attr", and value containing "valContaining"	`a[href*=/search/]`
`[attr~=regex]`	elements with an attribute named "attr", and value matching the regular expression	`img[src~=(?i)\.(png|jpe?g)]`
	The above may be combined in any order	`div.header[title]`

Selector full reference

Examples

Selecting elements using CSS selectors

```
String html = "<!DOCTYPE html>" +
              "<html>" +
                "<head>" +
                  "<title>Hello world!</title>" +
                "</head>" +
                "<body>" +
                  "<h1>Hello there!</h1>" +
                  "<p>First paragraph</p>" +
                  "<p class=\"not-first\">Second paragraph</p>" +
                  "<p class=\"not-first third\">Third <a href=\"page.html\">paragraph</a></p>"
+
                "</body>" +
              "</html>";
```

```
// Parse the document
Document doc = Jsoup.parse(html);

// Get document title
String title = doc.select("head > title").first().text();
System.out.println(title); // Hello world!

Element firstParagraph = doc.select("p").first();

// Get all paragraphs except from the first
Elements otherParagraphs = doc.select("p.not-first");
// Same as
otherParagraphs = doc.select("p");
otherParagraphs.remove(0);

// Get the third paragraph (second in the list otherParagraphs which
// excludes the first paragraph)
Element thirdParagraph = otherParagraphs.get(1);
// Alternative:
thirdParagraph = doc.select("p.third");

// You can also select within elements, e.g. anchors with a href attribute
// within the third paragraph.
Element link = thirdParagraph.select("a[href]");
// or the first <h1> element in the document body
Element headline = doc.select("body").first().select("h1").first();
```

You can find a detailed overview of supported selectors here.

Extract Twitter Markup

```
    // Twitter markup documentation:
    // https://dev.twitter.com/cards/markup
    String[] twitterTags = {
            "twitter:site",
            "twitter:site:id",
            "twitter:creator",
            "twitter:creator:id",
            "twitter:description",
            "twitter:title",
            "twitter:image",
            "twitter:image:alt",
            "twitter:player",
            "twitter:player:width",
            "twitter:player:height",
            "twitter:player:stream",
            "twitter:app:name:iphone",
            "twitter:app:id:iphone",
            "twitter:app:url:iphone",
            "twitter:app:name:ipad",
            "twitter:app:id:ipad",
            "twitter:app:url:ipadt",
            "twitter:app:name:googleplay",
            "twitter:app:id:googleplay",
            "twitter:app:url:googleplay"
    };

    // Connect to URL and extract source code
    Document doc = Jsoup.connect("http://stackoverflow.com/").get();
```

```
    for (String twitterTag : twitterTags) {

        // find a matching meta tag
        Element meta = doc.select("meta[name=" + twitterTag + "]").first();

        // if found, get the value of the content attribute
        String content = meta != null ? meta.attr("content") : "";

        // display results
        System.out.printf("%s = %s%n", twitterTag, content);
    }
```

Output

```
twitter:site =
twitter:site:id =
twitter:creator =
twitter:creator:id =
twitter:description = Q&A for professional and enthusiast programmers
twitter:title = Stack Overflow
twitter:image =
twitter:image:alt =
twitter:player =
twitter:player:width =
twitter:player:height =
twitter:player:stream =
twitter:app:name:iphone =
twitter:app:id:iphone =
twitter:app:url:iphone =
twitter:app:name:ipad =
twitter:app:id:ipad =
twitter:app:url:ipadt =
twitter:app:name:googleplay =
twitter:app:id:googleplay =
twitter:app:url:googleplay =
```

Read Selectors online: https://riptutorial.com/jsoup/topic/515/selectors

Chapter 6: Web crawling with Jsoup

Examples

Extracting email adresses & links to other pages

Jsoup can be used to extract links and email address from a webpage, thus "Web email address collector bot" First, this code uses a Regular expression to extract the email addresses, and then uses methods provided by Jsoup to extract the URLs of links on the page.

```
public class JSoupTest {

    public static void main(String[] args) throws IOException {
        Document doc =
Jsoup.connect("http://stackoverflow.com/questions/15893655/").userAgent("Mozilla").get();

        Pattern p = Pattern.compile("[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+");
        Matcher matcher = p.matcher(doc.text());
        Set<String> emails = new HashSet<String>();
        while (matcher.find()) {
            emails.add(matcher.group());
        }

        Set<String> links = new HashSet<String>();

        Elements elements = doc.select("a[href]");
        for (Element e : elements) {
            links.add(e.attr("href"));
        }

        System.out.println(emails);
        System.out.println(links);

    }

}
```

This code could also be easily extended to also recursively visit those URLs and extract data from linked pages. It could also easily be used with a different regex to extract other data.

(Please don't become a spammer!)

Extracting JavaScript data with Jsoup

In this example, we will try to find JavaScript data which containing `backgroundColor:'#FFF'`. Then, we will change value of backgroundColor `'#FFF' '#ddd'`. This code uses `getWholeData()` and `setWholeData()` methods to manipulate JavaScript data. Alternatively, `html()` method can be used to get data of JavaScript.

```
// create HTML with JavaScript data
    StringBuilder html = new StringBuilder();
```

```
    html.append("<!DOCTYPE html> <html> <head> <title>Hello Jsoup!</title>");
    html.append("<script>");
    html.append("StackExchange.docs.comments.init({");
    html.append("highlightColor: '#F4A83D',");
    html.append("backgroundColor:'#FFF',");
    html.append("});");
    html.append("</script>");
    html.append("<script>");
    html.append("document.write(<style type='text/css'>div,iframe { top: 0; position:absolute;
}</style>');");
    html.append("</script>\n");
    html.append("</head><body></body> </html>");

    // parse as HTML document
    Document doc = Jsoup.parse(html.toString());

    String defaultBackground = "backgroundColor:'#FFF'";
    // get <script>
    for (Element scripts : doc.getElementsByTag("script")) {
        // get data from <script>
        for (DataNode dataNode : scripts.dataNodes()) {
            // find data which contains backgroundColor:'#FFF'
            if (dataNode.getWholeData().contains(defaultBackground)) {
                // replace '#FFF' -> '#ddd'
                String newData = dataNode.getWholeData().replaceAll(defaultBackground,
"backgroundColor:'#ddd'");
                // set new data contents
                dataNode.setWholeData(newData);
            }
        }
    }
    System.out.println(doc.toString());
```

Output

```
<script>StackExchange.docs.comments.init({highlightColor:
'#F4A83D',backgroundColor:'#ddd',});</script>
```

Extracting all the URLs from a website using JSoup (recursion)

In this example we will extract all the web links from a website. I am using `http://stackoverflow.com/` for illustration. Here recursion is used, where each obtained link's page is parsed for presence of an `anchor tag` and that link is again submitted to the same function.

The condition `if(add && this_url.contains(my_site))` will limit results **to your domain only**.

```
    import java.io.IOException;
    import java.util.HashSet;
    import java.util.Set;
    import org.jsoup.Jsoup;
    import org.jsoup.nodes.Document;
    import org.jsoup.select.Elements;

    public class readAllLinks {

        public static Set<String> uniqueURL = new HashSet<String>();
        public static String my_site;
```

```
        public static void main(String[] args) {

            readAllLinks obj = new readAllLinks();
            my_site = "stackoverflow.com";
            obj.get_links("http://stackoverflow.com/");
        }

        private void get_links(String url) {
            try {
                Document doc = Jsoup.connect(url).userAgent("Mozilla").get();
                Elements links = doc.select("a");

                if (links.isEmpty()) {
                   return;
                }

                links.stream().map((link) -> link.attr("abs:href")).forEachOrdered((this_url)
-> {
                    boolean add = uniqueURL.add(this_url);
                    if (add && this_url.contains(my_site)) {
                        System.out.println(this_url);
                        get_links(this_url);
                    }
                });

            } catch (IOException ex) {

            }

        }
    }
```

The program will take much time to execute depending on your website. The above code can be extended to extract data (like titles of pages or text or images) from particular website. I would recommend you to go through company's terms of use before scarping it's website.

The example uses JSoup library to get the links, you can also get the links using `your_url/sitemap.xml`.

Read Web crawling with Jsoup online: https://riptutorial.com/jsoup/topic/319/web-crawling-with-jsoup

Credits

S. No	Chapters	Contributors
1	Getting started with Jsoup	Alice, Community, Jeffrey Bosboom, JonasCz, Zack Teater
2	Formatting HTML Output	Zack Teater
3	Logging into websites with Jsoup	Joel Min, JonasCz, Stephan
4	Parsing Javascript Generated Pages	Zack Teater
5	Selectors	JonasCz, Stephan, still_learning, Zack Teater
6	Web crawling with Jsoup	Alice, JonasCz, r_D, RamenChef