

1. 解析和遍历一个 html 文档

输入

- 2. 解析一个 html 字符串
- 3. 解析一个 body 片断
- 4. 根据一个 url 加载 Document 对象
- 5. 根据一个文件加载 Document 对象

数据抽取

- 6. 使用 dom 方法来遍历一个 Document 对象
- 7. 使用选择器语法来查找元素
- 8. 从元素集合抽取属性、文本和 html 内容
- 9. URL 处理
- 10. 程序示例：获取所有链接

数据修改

- 11. 设置属性值
- 12. 设置元素的 html 内容
- 13. 设置元素的文本内容

html 清理

- 14. 消除不受信任的 html (来防止 xss 攻击)

1.解析和遍历一个 HTML 文档

如何解析一个 HTML 文档：

```
String html = "<html><head><title>First  
parse</title></head>"  
+ "<body><p>Parsed HTML into a doc.</p></body></html>";  
Document doc = Jsoup.parse(html);
```

(更详细内容可查看 [解析一个 HTML 字符串](#).)

其解析器能够尽最大可能从你提供的 HTML 文档来创建一个干净的解析结果，无论 HTML 的格式是否完整。比如它可以处理：

- 没有关闭的标签 (比如: `<p>Lorem <p>Ipsum parses to <p>Lorem</p><p>Ipsum</p>`)
- 隐式标签 (比如. 它可以自动将 `<td>Table data</td>` 包装成 `<table><tr><td>?`)
- 创建可靠的文档结构 (html 标签包含 head 和 body, 在 head 只出现恰当的元素)

一个文档的对象模型

- 文档由多个 Elements 和 TextNodes 组成 (以及其它辅助 nodes: 详细可查看: [nodes package tree](#)).
- 其继承结构如下: Document 继承 Element 继承 Node. TextNode 继承 Node.
- 一个 Element 包含一个子节点集合, 并拥有一个父 Element. 他们还提供了一个唯一的子元素过滤列表。

参见

- 数据抽取: [DOM 遍历](#)
- 数据抽取: [Selector syntax](#)

2.解析一个 HTML 字符串

存在问题

来自用户输入, 一个文件或一个网站的 HTML 字符串, 你可能需要对它进行解析并取其内容, 或校验其格式是否完整, 或想修改它。怎么办? jsoup 能够帮你轻松解决这些问题

解决方法

使用静态 Jsoup.parse(String html) 方法或 Jsoup.parse(String html, String baseUrl) 示例代码:

```
String html = "<html><head><title>First  
parse</title></head>"  
+ "<body><p>Parsed HTML into a doc.</p></body></html>";  
Document doc = Jsoup.parse(html);
```

描述

parse(String html, String baseUrl) 这方法能够将输入的 HTML 解析为一个新的文档 (Document), 参数 baseUrl 是用来将相对 URL 转成绝对 URL, 并指定从哪个网站获取文档。如这个方法不适用, 你可以使用 parse(String html) 方法来解析成 HTML 字符串如上面的示例。

只要解析的不是空字符串, 就能返回一个结构合理的文档, 其中包含(至少)一个 head 和一个 body 元素。

一旦拥有了一个 Document, 你就可以使用 Document 中适当的方法或它父类 Element 和 Node 中的方法来取得相关数据。

3.解析一个 body 片断

问题

假如你有一个 HTML 片断 (比如. 一个 div 包含一对 p 标签; 一个不完整的 HTML 文档) 想对它进行解析。这个 HTML 片断可以是用户提交的一条评论或在一个 CMS 页面中编辑 body 部分。

办法

使用 Jsoup.parseBodyFragment(String html) 方法。

```
String html = "<div><p>Lorem ipsum.</p>";  
Document doc = Jsoup.parseBodyFragment(html);  
Element body = doc.body();
```

说明

parseBodyFragment 方法创建一个空壳的文档, 并插入解析过的 HTML 到 body 元素中。假如你使用正常的 Jsoup.parse(String html) 方法, 通常你也可以得到相同的结果, 但是明确将用户输入作为 body 片段处理, 以确保用户所提供的任何糟糕的 HTML 都被解析成 body 元素。

Document.body() 方法能够取得文档 body 元素的所有子元素, 与 doc.getElementsByTag("body") 相同。

保证安全 Stay safe

假如你可以让用户输入 HTML 内容, 那么要小心避免跨站脚本攻击。利用基于 Whitelist 的清除器和 clean(String bodyHtml, Whitelist whitelist)方法来清除用户输入的恶意内容。



4. 从一个 URL 加载一个 Document

存在问题

你需要从一个网站获取和解析一个 HTML 文档，并查找其中的相关数据。你可以使用下面解决方法：

解决方法

使用 `Jsoup.connect(String url)` 方法：

```
Document doc = Jsoup.connect("http://example.com/").get();
String title = doc.title();
```

说明

`connect(String url)` 方法创建一个新的 `Connection`，和 `get()` 取得和解析一个 HTML 文件。如果从该 URL 获取 HTML 时发生错误，便会抛出 `IOException`，应适当处理。

`Connection` 接口还提供一个方法链来解决特殊请求，具体如下：

```
Document doc = Jsoup.connect("http://example.com")
    .data("query", "Java")
    .userAgent("Mozilla")
    .cookie("auth", "token")
    .timeout(3000)
    .post();
```

这个方法只支持 Web URLs (http 和 https 协议)；假如你需要从一个文件加载，可以使用 `parse(File in, String charsetName)` 代替。

~ 4 ~

6. 使用 DOM 方法来遍历一个文档

问题

你有一个 HTML 文档要从中提取数据，并了解这个 HTML 文档的结构。

方法

将 HTML 解析成一个 `Document` 之后，就可以使用类似于 DOM 的方法进行操作。示例代码：

```
File input = new File("/tmp/input.html");
Document doc = Jsoup.parse(input, "UTF-8",
    "http://example.com/");

Element content = doc.getElementById("content");
Elements links = content.getElementsByTag("a");
for (Element link : links) {
    String linkHref = link.attr("href");
    String linkText = link.text();
}
```

说明

`Elements` 这个对象提供了一系列类似于 DOM 的方法来查找元素，抽取并处理其中的数据。具体如下：

查找元素

- `getElementById(String id)`
- `getElementsByTag(String tag)`
- `getElementsByClass(String className)`
- `getElementsByAttribute(String key)` (and related methods)
- `Element siblings: siblingElements(), firstElementSibling(), lastElementSibling(), nextElementSibling(), previousElementSibling()`
- `Graph: parent(), children(), child(int index)`

元素数据

~ 6 ~



5. 从一个文件加载一个文档

问题

在本机硬盘上有一个 HTML 文件，需要对它进行解析从中抽取数据或进行修改。

办法

可以使用静态 `Jsoup.parse(File in, String charsetName, String baseUri)` 方法：

```
File input = new File("/tmp/input.html");
Document doc = Jsoup.parse(input, "UTF-8",
    "http://example.com/");
```

说明

`parse(File in, String charsetName, String baseUri)` 这个方法用来加载和解析一个 HTML 文件。如在加载文件的时候发生错误，将抛出 `IOException`，应作适当处理。

`baseUri` 参数用于解决文件中 URLs 是相对路径的问题。如果不需要可以传入一个空的字符串。

另外还有一个方法 `parse(File in, String charsetName)`，它使用文件的路径做为 `baseUri`。这个方法适用于如果被解析文件位于网站的本地文件系统，且相关链接也指向该文件系统。

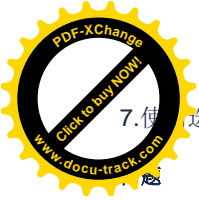
~ 5 ~

- `attr(String key)` 获取属性 `attr(String key, String value)` 设置属性
- `attributes()` 获取所有属性
- `id(), className()` and `classNames()`
- `text()` 获取文本内容 `text(String value)` 设置文本内容
- `html()` 获取元素内 HTML `html(String value)` 设置元素内的 HTML 内容
- `outerHtml()` 获取元素外 HTML 内容
- `data()` 获取数据内容 (例如: `script` 和 `style` 标签)
- `tag()` and `tagName()`

操作 HTML 和文本

- `append(String html), prepend(String html)`
- `appendText(String text), prependText(String text)`
- `appendElement(String tagName), prependElement(String tagName)`
- `html(String value)`

~ 7 ~



7.使用选择器语法来查找元素

你想使用类似于 CSS 或 jQuery 的语法来查找和操作元素。

方法

可以使用 `Element.select(String selector)` 和 `Elements.select(String selector)` 方法实现：

```
File input = new File("/tmp/input.html");
Document doc = Jsoup.parse(input, "UTF-8",
"http://example.com/");

Elements links = doc.select("a[href]"); //带有 href 属性的 a
元素
Elements pngs = doc.select("img[src$=.png]");
//扩展名为 .png 的图片

Element masthead = doc.select("div.masthead").first();
//class 等于 masthead 的 div 标签

Elements resultLinks = doc.select("h3.r > a"); //在 h3 元素之
后的 a 元素
```

说明

jsoup elements 对象支持类似于 CSS (或 jquery) 的选择器语法，来实现非常强大和灵活的查找功能。

这个 select 方法在 Document, Element, 或 Elements 对象中都可以使用。且是上下文相关的，因此可实现指定元素的过滤，或者链式选择访问。

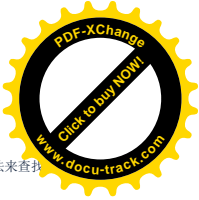
Select 方法将返回一个 Elements 集合，并提供一组方法来抽取和处理结果。

~ 8 ~

- `:not(selector)`: 查找与选择器不匹配的元素，比如: `div:not(.logo)` 表示不包含 `class=logo` 元素的所有 `div` 列表
- `:contains(text)`: 查找包含给定文本的元素，搜索不区分大小写，比如: `p:contains(jsoup)`
- `:containsOwn(text)`: 查找直接包含给定文本的元素
- `:matches(regex)`: 查找哪些元素的文本匹配指定的正则表达式，比如: `div:matches(?:)login`
- `:matchesOwn(regex)`: 查找自身包含文本匹配指定正则表达式的元素
- 注意: 上述伪选择器索引是从 0 开始的，也就是说第一个元素索引值为 0，第二个元素 index 为 1 等

可以查看 [Selector API](#) 参考来了解更详细的内容

~ 1 0 ~



Selector 选择器概述

- `tagname`: 通过标签查找元素，比如: `a`
- `ns|tag`: 通过标签在命名空间查找元素，比如: 可以用 `fb:name` 语法来查找 `<fb:name>` 元素
- `#id`: 通过 ID 查找元素，比如: `#logo`
- `.class`: 通过 class 名称查找元素，比如: `.masthead`
- `[attribute]`: 利用属性查找元素，比如: `[href]`
- `[^attr]`: 利用属性名前缀来查找元素，比如: 可以用 `[^data-]` 来查找带有 HTML5 Dataset 属性的元素
- `[attr=value]`: 利用属性值来查找元素，比如: `[width=500]`
- `[attr^=value], [attr$=value], [attr*=value]`: 利用匹配属性值开头、结尾或包含属性值来查找元素，比如: `[href*=path/]`
- `[attr~regex]`: 利用属性值匹配正则表达式来查找元素，比如: `img[src~=(?i)\.(png|jpe?g)]`
- `*`: 这个符号将匹配所有元素

Selector 选择器组合使用

- `el#id`: 元素+ID，比如: `div#logo`
- `el.class`: 元素+class，比如: `div.masthead`
- `el[attr]`: 元素+class，比如: `a[href]`
- 任意组合，比如: `a[href].highlight`
- `ancestor child`: 查找某个元素下子元素，比如: 可以用 `.body p` 查找在 "body" 元素下的所有 `p` 元素
- `parent > child`: 查找某个父元素下的直接子元素，比如: 可以用 `div.content > p` 查找 `p` 元素，也可以用 `body > *` 查找 `body` 标签下所有直接子元素
- `siblingA + siblingB`: 查找在 A 元素之前第一个同级元素 B，比如: `div.head + div`
- `siblingA ~ siblingX`: 查找 A 元素之前的同级 X 元素，比如: `h1 ~ p`
- `el, el, el`: 多个选择器组合，查找匹配任一选择器的唯一元素，例如: `div.masthead, div.logo`

伪选择器 selectors

- `:lt(n)`: 查找哪些元素的同级索引值 (它的位置在 DOM 树中是相对于它的父节点) 小于 n，比如: `td:lt(3)` 表示小于三列的元素
- `:gt(n)`: 查找哪些元素的同级索引值大于 n，比如: `div p:gt(2)` 表示哪些 `div` 中有包含 2 个以上的 `p` 元素
- `:eq(n)`: 查找哪些元素的同级索引值与 n 相等，比如: `form input:eq(1)` 表示包含一个 `input` 标签的 `Form` 元素
- `:has(selector)`: 查找匹配选择器包含元素的元素，比如: `div:has(p)` 表示哪些 `div` 包含了 `p` 元素

~ 9 ~

8.从元素抽取属性，文本和 HTML

问题

在解析获得一个 Document 实例对象，并查找到一些元素之后，你希望取得在这些元素中的数据。

方法

- 要取得一个属性的值，可以使用 `Node.attr(String key)` 方法
- 对于一个元素中的文本，可以使用 `Element.text()` 方法
- 对于要取得元素或属性中的 HTML 内容，可以使用 `Element.html()`，或 `Node.outerHtml()` 方法

示例:

```
String html = "<p>An <a href='http://example.com/'><b>example</b></a> link.</p>";
Document doc = Jsoup.parse(html); //解析 HTML 字符串返回一个
```

Document 实现

```
Element link = doc.select("a").first(); //查找第一个 a 元素
```

```
String text = doc.body().text(); // "An example link" //取得
字符串中的文本
String linkHref = link.attr("href"); //
"http://example.com/" //取得链接地址
```

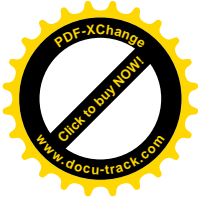
```
String linkText = link.text(); // "example" //取得链接地址中
的文本
```

```
String linkOuterH = link.outerHtml();
// "<a href='http://example.com'><b>example</b></a>"
```

```
String linkInnerH = link.html(); // "<b>example</b>" //取得
```

链接内的 html 内容

~ 1 1 ~



法是元素数据访问的核心办法。此外还其它一些方法可以使用：

- `Element.id()`
- `Element.tagName()`
- `Element.className()` and `Element.hasClass(String className)`

这些访问器方法都有相应的 `setter` 方法来更改数据。

参见

- `Element` 和 `Elements` 集合类的参考文档
- [URLs 处理](#)
- [使用 CSS 选择器语法来查找元素](#)

~ 1 2 ~

10. 示例程序：获取所有链接

这个示例程序将展示如何从一个 URL 获得一个页面。然后提取页面中的所有链接、图片和其它辅助内容。并检查 URLs 和文本信息。

运行下面程序需要指定一个 URLs 作为参数

```
package org.jsoup.examples;

import org.jsoup.Jsoup;
import org.jsoup.helper.Validate;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.io.IOException;

/**
 * Example program to list links from a URL.
 */
public class ListLinks {
    public static void main(String[] args) throws IOException {
        Validate.isTrue(args.length == 1, "usage: supply url to fetch");
        String url = args[0];
        print("Fetching %s...", url);

        Document doc = Jsoup.connect(url).get();
        Elements links = doc.select("a[href]");
        Elements media = doc.select("[src]");
        Elements imports = doc.select("link[href]");

        print("\nMedia: (%d)", media.size());
        for (Element src : media) {
            if (src.tagName().equals("img"))
                print(" * %s: <%s> %sx%s (%s)",
                    src.tagName(), src.attr("abs:src"),
                    src.attr("width"), src.attr("height"),
                    trim(src.attr("alt"), 20));
            else
                print(" * %s: <%s>", src.tagName(),
                    src.attr("abs:src"));
        }
    }
}
```

~ 1 4 ~

9. 处理 URLs

问题

你有一个包含相对 URLs 路径的 HTML 文档，需要将这些相对路径转换成绝对路径的 URLs。

方法

1. 在你解析文档时确保有指定 base URI，然后
2. 使用 `abs`：属性前缀来取得包含 base URI 的绝对路径。代码如下：

```
Document doc =
Jsoup.connect("http://www.open-open.com").get();

Element link = doc.select("a").first();
String relHref = link.attr("href"); // == "/"
String absHref = link.attr("abs:href"); //
"http://www.open-open.com/"
```

说明

在 HTML 元素中，URLs 经常写成相对于文档位置的相对路径：
`...`。当你使用 `Node.attr(String key)` 方法来取得 `a` 元素的 `href` 属性时，它将直接返回在 HTML 源码中指定的值。

假如你需要取得一个绝对路径，需要在属性名前加 `abs`：前缀。这样就可以返回包含根路径的 URL 地址 `attr("abs:href")`

因此，在解析 HTML 文档时，定义 base URI 非常重要。

如果你不想使用 `abs`：前缀，还有一个方法能够实现同样的功能 `Node.absUrl(String key)`。

~ 1 3 ~

```
    }

    print("\nImports: (%d)", imports.size());
    for (Element link : imports) {
        print(" * %s <%s> (%s)",
            link.tagName(), link.attr("abs:href"), link.attr("rel"));
    }

    print("\nLinks: (%d)", links.size());
    for (Element link : links) {
        print(" * a: <%s> (%s)", link.attr("abs:href"),
            trim(link.text(), 35));
    }
}

private static void print(String msg, Object... args) {
    System.out.println(String.format(msg, args));
}

private static String trim(String s, int width) {
    if (s.length() > width)
        return s.substring(0, width-1) + "...";
    else
        return s;
}
}
```

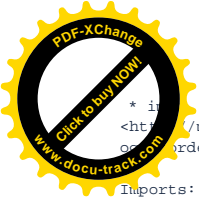
`org/jsoup/examples/ListLinks.java`

示例输入结果

Fetching http://news.ycombinator.com/...

```
Media: (38)
 * img: <http://ycombinator.com/images/y18.gif> 18x18 ()
 * img: <http://ycombinator.com/images/s.gif> 10x1 ()
 * img: <http://ycombinator.com/images/grayarrow.gif> x ()
 * img: <http://ycombinator.com/images/s.gif> 0x10 ()
 * script: <http://www.co2stats.com/propres.php?s=1138>
 * img: <http://ycombinator.com/images/s.gif> 15x1 ()
 * img: <http://ycombinator.com/images/hnsearch.png> x ()
 * img: <http://ycombinator.com/images/s.gif> 25x1 ()
```

~ 1 5 ~

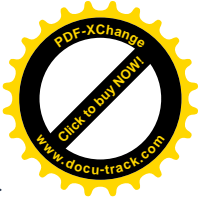


```
* a: <http://news.ycombinator.com/lists> (Lists)
* a: <http://news.ycombinator.com/rss> (RSS)
* a: <http://ycombinator.com/bookmarklet.html>
(Bookmarklet)
* a: <http://ycombinator.com/newsguidelines.html>
(Guidelines)
* a: <http://ycombinator.com/newsfaq.html> (FAQ)
* a: <http://ycombinator.com/newsnews.html> (News News)
* a: <http://news.ycombinator.com/item?id=363> (Feature
Requests)
* a: <http://ycombinator.com> (Y Combinator)
* a: <http://ycombinator.com/w2010.html> (Apply)
* a: <http://ycombinator.com/lib.html> (Library)
* a: <http://www.webmynd.com/html/hackernews.html> ()
* a: <http://mixpanel.com/?from=yc> ()

Imports: (2)
* link <http://ycombinator.com/news.css> (stylesheet)
* link <http://ycombinator.com/favicon.ico> (shortcut icon)

Links: (141)
* a: <http://ycombinator.com> ()
* a: <http://news.ycombinator.com/news> (Hacker News)
* a: <http://news.ycombinator.com/newest> (new)
* a: <http://news.ycombinator.com/newcomments> (comments)
* a: <http://news.ycombinator.com/leaders> (leaders)
* a: <http://news.ycombinator.com/jobs> (jobs)
* a: <http://news.ycombinator.com/submit> (submit)
* a: <http://news.ycombinator.com/x?fnid=JKhQjFU7gW>
(login)
* a:
<http://news.ycombinator.com/vote?for=1094578&dir=up&when
ce=%6e%65%77%73> ()
* a:
<http://www.readwriteweb.com/archives/facebook_gets_faste
r_debuts_homegrown_php_compiler.php?utm_source=feedburner
&utm_medium=feed&utm_campaign=Feed%3A+readwriteweb+%28Rea
dWriteWeb%29&utm_content=Twitter> (Facebook speeds up PHP)
* a: <http://news.ycombinator.com/user?id=mcxx> (mcxx)
* a: <http://news.ycombinator.com/item?id=1094578> (9
comments)
* a:
<http://news.ycombinator.com/vote?for=1094649&dir=up&when
ce=%6e%65%77%73> ()
* a:
<http://groups.google.com/group/django-developers/msg/a65
fbbc8effcd914> ("Tough. Django produces XHTML.")
* a: <http://news.ycombinator.com/user?id=andybak>
(andybak)
* a: <http://news.ycombinator.com/item?id=1094649> (3
comments)
* a:
<http://news.ycombinator.com/vote?for=1093927&dir=up&when
ce=%6e%65%77%73> ()
* a: <http://news.ycombinator.com/x?fnid=p2sdPLE7Ce>
(More)
```

~ 1 6 ~



```
* a: <http://news.ycombinator.com/lists> (Lists)
* a: <http://news.ycombinator.com/rss> (RSS)
* a: <http://ycombinator.com/bookmarklet.html>
(Bookmarklet)
* a: <http://ycombinator.com/newsguidelines.html>
(Guidelines)
* a: <http://ycombinator.com/newsfaq.html> (FAQ)
* a: <http://ycombinator.com/newsnews.html> (News News)
* a: <http://news.ycombinator.com/item?id=363> (Feature
Requests)
* a: <http://ycombinator.com> (Y Combinator)
* a: <http://ycombinator.com/w2010.html> (Apply)
* a: <http://ycombinator.com/lib.html> (Library)
* a: <http://www.webmynd.com/html/hackernews.html> ()
* a: <http://mixpanel.com/?from=yc> ()
```

~ 1 7 ~

11. 设置属性的值

问题

在你解析一个 `Document` 之后可能想修改其中的某些属性值，然后再保存到磁盘或都输出到前台页面。

方法

可以使用属性设置方法 `Element.attr(String key, String value)`，和 `Elements.attr(String key, String value)`。

假如你需要修改一个元素的 `class` 属性，可以使用 `Element.addClass(String className)` 和 `Element.removeClass(String className)` 方法。

`Elements` 提供了批量操作元素属性和 `class` 的方法，比如：要为 `div` 中的每一个 `a` 元素都添加一个 `rel="nofollow"` 可以使用如下方法：

```
doc.select("div.comments a").attr("rel", "nofollow");
```

说明

与 `Element` 中的其它方法一样，`attr` 方法也是返回当 `Element` (或使用选择器是返回 `Elements` 集合)。这样能够很方便使用方法连用的书写方式。比如：

```
doc.select("div.masthead").attr("title",
"jsoup").addClass("round-box");
```

~ 1 8 ~

12. 设置一个元素的 HTML 内容

问题

你需要一个元素中的 HTML 内容

方法

可以使用 `Element` 中的 HTML 设置方法具体如下：

```
Element div = doc.select("div").first(); // <div></div>
div.html("<p>lorem ipsum</p>"); // <div><p>lorem
ipsum</p></div>
```

```
div.prepend("<p>First</p>");//在div前添加html内容
```

```
div.append("<p>Last</p>");//在div之后添加html内容
```

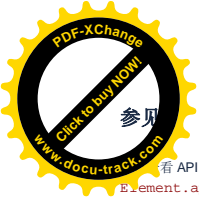
```
// 添完后的结果: <div><p>First</p><p>lorem
ipsum</p><p>Last</p></div>
```

```
Element span = doc.select("span").first(); //
<span>One</span>
span.wrap("<li><a href='http://example.com/'><a></li>");
// 添完后的结果: <li><a
href="http://example.com"><span>One</span></a></li>
```

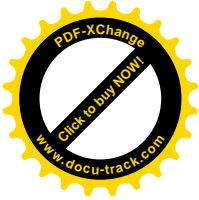
说明

- `Element.html(String html)` 这个方法将先清除元素中的 HTML 内容，然后用传入的 HTML 代替。
- `Element.prepend(String first)` 和 `Element.append(String last)` 方法用于在分别在元素内部 HTML 的前面和后面添加 HTML 内容
- `Element.wrap(String around)` 对元素包裹一个外部 HTML 内容。

~ 1 9 ~



看 API 参考文档中 `Element.prependElement(String tag)` 和 `Element.appendElement(String tag)` 方法来创建新的元素并作为文档的子元素插入其中。



13. 设置元素的文本内容

问题

你需要修改一个 HTML 文档中的文本内容

方法

可以使用 `Element` 的设置方法：

```
Element div = doc.select("div").first(); // <div></div>
div.text("five > four"); // <div>five &gt; four</div>
div.prepend("First ");
div.append(" Last");
// now: <div>First five &gt; four Last</div>
```

说明

文本设置方法与 `HTML setter` 方法一样：

- `Element.text(String text)` 将清除一个元素中的内部 HTML 内容，然后提供
的文本进行代替
- `Element.prepend(String first)` 和 `Element.append(String last)`
将分别在元素的内部 html 前后添加文本节点。

对于传入的文本如果含有像 `<`, `>` 等这样的字符，将以文本处理，而非 HTML。

~ 2 0 ~

~ 2 1 ~

14. 消除不受信任的 HTML (来防止 XSS 攻击)

问题

在做网站的时候，经常会提供用户评论的功能。有些心怀不测的用户，会搞一些脚本到评论内容中，而这些脚本可能会破坏整个页面的行为，更严重的是获取一些机密信息，此时需要清理该 HTML，以避免跨站脚本 `cross-site scripting` 攻击 (XSS)。

方法

使用 jsoup HTML `Cleaner` 方法进行清除，但需要指定一个可配置的 `Whitelist`。

```
String unsafe =
    "<p><a href='http://example.com/'>Link</a></p>";
onClick='stealCookies()'>Link</a></p>";
String safe = Jsoup.clean(unsafe, Whitelist.basic());
// now: <p><a href="http://example.com/"
rel="nofollow">Link</a></p>
```

说明

XSS 又叫 CSS (Cross Site Script)，跨站脚本攻击。它指的是恶意攻击者往 Web 页面里插入恶意 html 代码，当用户浏览该页之时，嵌入其中 Web 里面的 html 代码会被执行，从而达到恶意攻击用户的特殊目的。XSS 属于被动的攻击，因为其被动且不好利用，所以许多人常忽略其危害性。所以我们经常只让用户输入纯文本的内容，但这样用户体验就比较差了。

一个更好的解决方法就是使用一个富文本编辑器 WYSIWYG 如 `CKEditor` 和 `TinyMCE`。这些可以输出 HTML 并能够让用户可视化编辑。虽然他们可以在客户端进行校验，但是这样还不够安全，需要在服务器端进行校验并清除有害的 HTML 代码，这样才能确保输入到你网站的 HTML 是安全的。否则，攻击者能够绕过客户端的 Javascript 验证，并注入不安全的 HTML 直接进入您的网站。

jsoup 的 `whitelist` 清理器能够在服务器端对用户输入的 HTML 进行过滤，只输出一些安全的标签和属性。

jsoup 提供了一系列的 `Whitelist` 基本配置，能够满足大多数要求；但如有必要，也可以进行修改，不过要小心。

这个 `cleaner` 非常好用不仅可以避免 XSS 攻击，还可以限制用户可以输入的标签范围。

参见

- 参阅 `XSS cheat sheet`，有一个例子可以了解为什么不能使用正则表达式，而采用安全的 `whitelist parser-based` 清理器才是正确的选择。
- 参阅 `Cleaner`，了解如何返回一个 `Document` 对象，而不是字符串
- 参阅 `Whitelist`，了解如何创建一个自定义的 `whitelist`
- `nofollow` 链接属性了解

~ 2 2 ~

~ 2 3 ~