# Jsoup

我们抓取到页面之后，还需要对页面进行解析。可以使用字符串处理工具解析页面，也可以使用正则表达式，但是这些方法都会带来很大的开发成本，所以我们需要使用一款专门解析html页面的技术。

## jsoup介绍

jsoup 是一款Java 的HTML解析器，可直接解析某个URL地址、HTML文本内容。它提供了一套非常省力的API，可通过DOM，CSS以及类似于jQuery的操作方法来取出和操作数据。

jsoup的主要功能如下：

从一个URL，文件或字符串中解析HTML；

使用DOM或CSS选择器来查找、取出数据；

可操作HTML元素、属性、文本；

先加入Jsoup依赖：

<!--Jsoup-->  
<dependency>  
 <groupId>org.jsoup</groupId>  
 <artifactId>jsoup</artifactId>  
 <version>1.10.3</version>  
</dependency>  
<!--测试-->  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
</dependency>

<!--工具-->  
<dependency>  
 <groupId>org.apache.commons</groupId>  
 <artifactId>commons-lang3</artifactId>  
 <version>3.7</version>  
</dependency>  
<dependency>  
 <groupId>commons-io</groupId>  
 <artifactId>commons-io</artifactId>  
 <version>2.6</version>  
</dependency>

## jsoup解析

jsoup.JsoupFirstTest

### 解析url

Jsoup可以直接输入url，它会发起请求并获取数据，封装为Document对象

@Test  
public void testJsoupUrl() throws Exception {  
 // 解析url地址  
 Document document = Jsoup.*parse*(new URL("http://www.itcast.cn/"), 1000);  
  
 //获取title的内容  
 Element title = document.getElementsByTag("title").first();  
 System.*out*.println(title.text());  
}

PS：虽然使用Jsoup可以替代HttpClient直接发起请求解析数据，但是往往不会这样用，因为实际的开发过程中，需要使用到多线程，连接池，代理等等方式，而jsoup对这些的支持并不是很好，所以我们一般把jsoup仅仅作为Html解析工具使用

### 解析字符串

先准备以下html文件

|  |
| --- |
| <html>  <head>  <title>传智播客官网-一样的教育,不一样的品质</title>  </head>  <body>  <div class="city">  <h3 id="city\_bj">北京中心</h3>  <fb:img src="/2018czgw/images/slogan.jpg" class="slogan"/>  <div class="city\_in">  <div class="city\_con" style="display: none;">  <ul>  <li id="test" class="class\_a class\_b">  <a href="http://www.itcast.cn" target="\_blank">  <span class="s\_name">北京</span>  </a>  </li>  <li>  <a href="http://sh.itcast.cn" target="\_blank">  <span class="s\_name">上海</span>  </a>  </li>  <li>  <a href="http://gz.itcast.cn" target="\_blank">  <span abc="123" class="s\_name">广州</span>  </a>  </li>  <ul>  <li>天津</li>  </ul>  </ul>  </div>  </div>  </div>  </body>  </html> |

Jsoup可以直接输入字符串，并封装为Document对象

@Test  
public void testJsoupString() throws Exception {  
 //读取文件获取  
 String html = FileUtils.*readFileToString*(new File("D:\\jsoup.html"), "UTF-8");  
  
 // 解析字符串  
 Document document = Jsoup.*parse*(html);  
  
 //获取title的内容  
 Element title = document.getElementsByTag("title").first();  
 System.*out*.println(title.text());  
  
}

### 解析文件

Jsoup可以直接解析文件，并封装为Document对象

@Test  
public void testJsoupHtml() throws Exception {  
 // 解析文件  
 Document document = Jsoup.*parse*(new File("D:\\jsoup.html"),"UTF-8");  
  
 //获取title的内容  
 Element title = document.getElementsByTag("title").first();  
 System.*out*.println(title.text());  
}

### 使用dom方式遍历文档

元素获取

1. 根据id查询元素getElementById
2. 根据标签获取元素getElementsByTag
3. 根据class获取元素getElementsByClass
4. 根据属性获取元素getElementsByAttribute

//1. 根据id查询元素getElementById  
Element element = document.getElementById("city\_bj");  
  
//2. 根据标签获取元素getElementsByTag  
element = document.getElementsByTag("title").first();  
  
//3. 根据class获取元素getElementsByClass  
element = document.getElementsByClass("s\_name").last();  
  
//4. 根据属性获取元素getElementsByAttribute  
element = document.getElementsByAttribute("abc").first();  
element = document.getElementsByAttributeValue("class", "city\_con").first();

元素中获取数据

1. 从元素中获取id
2. 从元素中获取className
3. 从元素中获取属性的值attr
4. 从元素中获取所有属性attributes
5. 从元素中获取文本内容text

//获取元素  
Element element = document.getElementById("test");  
  
//1. 从元素中获取id  
String str = element.id();  
  
//2. 从元素中获取className  
str = element.className();  
  
//3. 从元素中获取属性的值attr  
str = element.attr("id");  
  
//4. 从元素中获取所有属性attributes  
str = element.attributes().toString();  
  
//5. 从元素中获取文本内容text  
str = element.text();

### 使用选择器语法查找元素

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。

### Selector选择器概述

**tagname**: 通过标签查找元素，比如：span

**#id**: 通过ID查找元素，比如：# city\_bj

**.class**: 通过class名称查找元素，比如：.class\_a

**[attribute]**: 利用属性查找元素，比如：[abc]

**[attr=value]**: 利用属性值来查找元素，比如：[class=s\_name]

//tagname: 通过标签查找元素，比如：span  
Elements span = document.select("span");  
for (Element element : span) {  
 System.*out*.println(element.text());  
}  
  
//#id: 通过ID查找元素，比如：#city\_bjj  
String str = document.select("#city\_bj").text();  
  
//.class: 通过class名称查找元素，比如：.class\_a  
str = document.select(".class\_a").text();  
  
//[attribute]: 利用属性查找元素，比如：[abc]  
str = document.select("[abc]").text();  
  
//[attr=value]: 利用属性值来查找元素，比如：[class=s\_name]  
str = document.select("[class=s\_name]").text();

### Selector选择器组合使用

**el#id**: 元素+ID，比如： h3#city\_bj

**el.class**: 元素+class，比如： li.class\_a

**el[attr]**: 元素+属性名，比如： span[abc]

**任意组合**: 比如：span[abc].s\_name

**ancestor child**: 查找某个元素下子元素，比如：.city\_con li 查找"city\_con"下的所有li

**parent > child**: 查找某个父元素下的直接子元素，比如：

.city\_con > ul > li 查找city\_con第一级（直接子元素）的ul，再找所有ul下的第一级li

**parent > \***: 查找某个父元素下所有直接子元素

//el#id: 元素+ID，比如： h3#city\_bj  
String str = document.select("h3#city\_bj").text();  
  
//el.class: 元素+class，比如： li.class\_a  
str = document.select("li.class\_a").text();  
  
//el[attr]: 元素+属性名，比如： span[abc]  
str = document.select("span[abc]").text();  
  
//任意组合，比如：span[abc].s\_name  
str = document.select("span[abc].s\_name").text();  
  
//ancestor child: 查找某个元素下子元素，比如：.city\_con li 查找"city\_con"下的所有li  
str = document.select(".city\_con li").text();  
  
//parent > child: 查找某个父元素下的直接子元素，  
//比如：.city\_con > ul > li 查找city\_con第一级（直接子元素）的ul，再找所有ul下的第一级li  
str = document.select(".city\_con > ul > li").text();  
  
//parent > \* 查找某个父元素下所有直接子元素.city\_con > \*  
str = document.select(".city\_con > \*").text();