# day34-基础加强【xml&jsoup】

# 学习目标

01\_能够说出XML的作用

02\_了解XML的组成元素

03\_能够说出有哪些XML约束技术

04\_能够说出解析XML文档DOM方式原理

05\_能够使用Jsoup技术3种方式获取Document对象

06\_熟悉Document对象常用方法

07\_能够列举Jsoup选择器查找元素相关方法

08\_能够使用Xpath解析HTML文档

# 项目清单

|  |
| --- |
|  |

# 回顾与提问

1\_有哪三种方式获取Class对象？

|  |
| --- |
| 1\_类名.class  2\_对象.getClass()  3\_Class.forName(“全限定类名”) |

2\_BeanUtils.populate()是什么意思？

|  |
| --- |
| 将Map<String,Object>类型的值一一封装到JavaBean中去。 |

# 问题引入

|  |
| --- |
|  |

# 第01章 XML

## 1.1 XML的概述

### 1.1.1什么是XML

XML指可扩展标记语言（eXtensible Markup Language）

XML是一种标记语言，很类似 HTML，HTML文件也是XML文档

XML标签没有被预定义，你需要按要求自行定义标签

XML的设计宗旨是传输数据，而非显示数据

XML被设计为具有自我描述性，就是易于阅读

XML是W3C的推荐标准

W3C在1988年2月发布1.0版本，2004年2月又发布1.1版本，单因为1.1版本不能向下兼容1.0版本，所以1.1没有人用。同时，在2004年2月W3C又发布了1.0版本的第三版。我们要学习的还是1.0版本。

### 1.1.2 XML与HTML的主要差异

XML不是HTML的替代

XML和HTML为不同的目的而设计

XML被设计为传输和配置数据，其焦点是数据的内容

HTML被设计用来显示数据，其焦点是数据的外观

HTML旨在显示信息，而XML旨在传输和配置数据

### 1.1.3 XML的应用场景

1. XML可以作用一个小型简单数据库。

2. XML可以作为数据交换的载体，也就是说使用XML格式进行数据的传输。

XML 数据是以纯文本格式的文件数据（以.xml扩展名的文件），因此提供了一种独立于软件和硬件的数据，可用于不同软件或系统之间传输数据。

1. XML可以作为配置文件，例如后面框架阶段我们学习的Spring框架的配置都是

通过XML进行配置的。

例如：spring框架的 ApplicationContext.xml配置文件。

@注解

### 1.1.4编写和运行XML

1. 用开发工具编写XML

|  |
| --- |
| 01\_book.xml  *<?***xml version="1.0" encoding="UTF-8"** *?>* <**book id="javaee01"**>  <**name**>JavaWeb入门</**name**>  <**price**>100</**price**>  <**author**>张三</**author**> </**book**> |

1. 用浏览器运行XML

|  |
| --- |
|  |

|  |
| --- |
|  |

## 1.2 XML文件的语法

XML文件中的数据一共由7个组成元素构成，分别为文档声明、 元素、 属性、注释、转义字符、字符区、处理指令，接下来我们分别讨论。

### 1.2.1文档声明

<?xml version="1.0" encoding="utf-8" ?>

1. 文档声明必须为<?xml开头，以?>结束。

2. 文档声明必须从文档的0行0列位置开始。

3. 文档声明只有三个属性：

version：指定XML文档版本。这里一般选择1.0，必须属性。

enconding：指定当前文档的编码，默认值是utf-8，可选属性。

standalone：指定文档独立性。可选属性，

默认值为yes，表示当前文档时独立文档。

如果为no表示当前文档不是独立的文档，会依赖外约束文件。

### 1.2.2元素/<book>

<book></book>

1. 元素是XML文档中最重要的组成部分。

2. 普通元素的结构由开始标签、元素体、结束标签组成。

例如： <name>张三</name>

1. 元素体：元素体可以是元素，也可以是文本。

例如： <person><name>张三</name></person>

1. 空元素：空元素只有标签，而没有结束标签，但元素必须自己闭合。

例如： <sex></sex>或<sex/>

1. 元素命名

区分大小写，

不能使用空格，不能使用冒号，

不建议以XML、xml、Xml开头。

6. 格式化良好的XML文档，必须只有一个根元素。

### 1.2.3属性

<book id="javaee01">

1. 属性是元素的一部分，它必须出现在元素的开始标签中。

2. 属性的定义格式：属性名=“属性值”，其中属性值必须使用单引或双引号括起来。

3. 一个元素可以有0~N个属性，但一个元素中不能出现同名属性。

4. 属性名不能使用空格、冒号等特殊字符，且必须以字母开头。

### 1.2.4注释

<!‐‐注释内容‐‐>

HTML <!-- -->

CSS /\* \*/

JS /\* \*/ //

XML <!-- -->

XML的注释与HTML相同，以 <!-- 开始， 以--> 结束。

### 1.2.5转义字符

<message>if salary &lt; 1000 then</message>

XML中的转义字符与HTML一样。因为很多符号已经被文档结构所使用，所以在元素体或属性值中想使用这些符号

就必须使用转义字符（也叫实体字符），例如：">"、"<"、"'"、"""、"&。

|  |
| --- |
|  |

### 1.2.6字符区

<![CDATA[

if (a < b && a < 0) then

]]>

1. CDATA 指的是不应由 XML 解析器进行解析的文本数据。
2. 字符区格式： <![CDATA[文本数据]]> ，CDATA 部分由 "<![CDATA[" 开始，由 "]]>" 结束。
3. 当大量的转义字符出现在XML文档中时，会使XML文档的可读性大幅度降低。

这时如果使用CDATA段就会好一些。

注意：

CDATA 段不能包含字符串 "]]>"。也不允许嵌套的 CDATA 段。

标记 CDATA 段结尾的 "]]>" 不能包含空格或折行。

### 1.2.7处理指令

<?xml-stylesheet type="text/css" href="a.css"?>

在XML文档中可以使用xml-stylesheet指令，通知XML解析引擎，引入CSS文件显示XML文档内容。

1. 处理指令必须以“<?”作为开头，以“?>”作为结尾，XML声明语句就是最常见的一种处理指令。

2. 处理指令，简称PI （processing instruction），处理指令用来指挥解析引擎如何解析XML文档内容。

3. xml-stylesheet指令应用样式需要使用浏览器浏览才可以看到效果。

步骤：

1. 创建一个book.xml文件

|  |
| --- |
| 01\_book.xml  *<?***xml version="1.0" encoding="UTF-8"** *?>* <**book id="javaee01"**>  <**name**>JavaWeb入门</**name**>  <**price**>100</**price**>  <**author**>张三</**author**> </**book**> |

1. 创建一个book.css文件

|  |
| --- |
| 01\_book.css  **name**{  **font-size**: 10**px**;  **color**:**red**; } **price**{  **font-size**: 12**px**;  **color**:**green**; } **author**{  **font-size**: 14**px**;  **color**:**blue**; } |

1. 让book.xml引用book.css文件

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?> <?***xml-stylesheet type="text/css" href="01\_book.css"***?>* <**book id="javaee01"**>  <**name**>JavaWeb入门</**name**>  <**price**>100</**price**>  <**author**>张三</**author**> </**book**> |

1. 用浏览器打开book.xml文件

|  |
| --- |
|  |

## 1.3 XML文件的约束

### 1.3.1 DTD约束

#### 1.3.1.1学习要求

在企业开发中，我们很少自己编写DTD约束文档，通常情况下通过框架提供的DTD约束文档编写对应的XML文档。所以只要求大家是可以根据DTD约束文档内容编写XML文档。

#### 1.3.1.2什么是DTD约束

DTD是文档类型定义（Document Type Definition）。

DTD 可以定义在 XML文档中出现的哪些元素、元素出现的顺序、次数，它们如何相互嵌套以及XML文档结构的详细信息。

#### 1.3.1.3 DTD约束体验

体验步骤：

1. 创建book.dtd文件。

|  |
| --- |
| 01\_book.dtd  **<!ELEMENT book** (**name**,**price**,**author**)> **<!ELEMENT name** (#PCDATA)> **<!ELEMENT price** (#PCDATA)> **<!ELEMENT author** (#PCDATA)> |

1. 创建book.xml文件。

|  |
| --- |
| 01\_book.xml  *<?***xml version="1.0" encoding="UTF-8"** *?>* <**book id="javaee01"**>  <**name**>JavaWeb入门</**name**>  <**price**>100</**price**>  <**author**>张三</**author**> </**book**> |

1. 在book.xml文件中引入book.dtd。

|  |
| --- |
| 01\_book.xml  *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE book SYSTEM "01\_book.dtd"*>*** <**book id="javaee01"**>  <**name**>JavaWeb入门</**name**>  <**price**>100</**price**>  <**author**>张三</**author**> </**book**> |

1. 开发工具会根据book.dtd对你的book.xml进行检查。

|  |
| --- |
|  |

|  |
| --- |
|  |

#### 1.3.1.4 DTD约束语法

##### 1.3.1.4.1文档声明

1. 内部DTD，在XML文档内部嵌入DTD，只对当前XML有效。

#PCDATA 只是一个字符串，它不区分类型

|  |
| --- |
| <?xml version="1.0" encoding="UTF‐8"?>  <!DOCTYPE 根元素 [...//具体语法]>  <根元素>  </根元素> |

1. 外部DTD—本地DTD，DTD文档在本地系统上，企业内部自己项目使用。

|  |
| --- |
| <?xml version="1.0" encoding="UTF‐8"?>  <!DOCTYPE 根元素 SYSTEM "book.dtd">  <根元素>  </根元素> |

1. 外部DTD—公共DTD，DTD文档在网络上，全世界可以使用，一般都有框架提供。

|  |
| --- |
| <?xml version="1.0" encoding="UTF‐8"?>  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd">  <web-app>  </web-app>  解释：  <!DOCTYPE 根元素 PUBLIC DTD名字 DTD文件的网网址 |

##### 1.3.1.4.2元素声明

1. 约束元素的嵌套层级

|  |
| --- |
| <!ELEMENT 父标签 （子标签1，子标签2，…）>  父标签里面，依次是子标签1，子标签2，。。。这样的顺序  例如：<!ELEMENT book (name,price,author)> |

1. 约束元素体里面的数据

|  |
| --- |
| <!ELEMENT 标签 标签类型>    例如：<!ELEMENT name (#PCDATA)>  <!ELEMENT marry ANY>  <!ELEMENT addr EMPTY> |

1. 数量词

|  |
| --- |
| 例如：<!ELEMENT book (name\*,price+,author?)>  <!ELEMENT price (a | b)>  <!ELEMENT a (#PCDATA)>  <!ELEMENT b (#PCDATA)> |

##### 1.3.1.4.3属性声明

|  |
| --- |
| <!ATTLIST 标签  属性名称1 属性类型1 属性说明1  属性名称2 属性类型2 属性说明2  …  >      例如：  <!ATTLIST 书  id ID #REQUIRED  编号 CDATA #IMPLIED  出版社 (清华|北大|传智播客) "传智播客"  类别 CDATA #FIXED "IT"  >  解释：  书标签有：  id------------------唯一类型-----------------必须的属性  编号--------------字符串类型--------------可选的属性  出版社----------枚举类型------------------三者选一，默认是传智播客，可选的属性  类别-------------字符串类型---------------可选的属性，如果写，必须是IT |

### 1.3.2 Schema约束

#### 1.3.2.1学习要求

虽然Schema功能比DTD强大，但是编写要比DTD复杂，同样以后我们在企业开发中也很少会自己编写Schema文件。关于Schema的学习，大家可以参考《W3CSchool.chm》。

这里我们只是体验一下。

#### 1.3.2.2什么是Schema约束

Schema语言也称为 XSD（XML Schema Definition）。

Schema比DTD强大，是DTD代替者。

Schema本身也是XML文档，Schema文档扩展名为xsd，而不是xml。

Schema功能更强大，数据类型约束更完善。

#### 1.3.2.3 Schema约束体验

1. 创建note.xml文件

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**note**>  <**name**>赵君</**name**>  <**age**>12</**age**>  <**sal**>1111.11</**sal**> </**note**> |

2. 创建note.xsd文件

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  targetNamespace="http://www.w3school.com.cn"  xmlns="http://www.w3school.com.cn"  elementFormDefault="qualified"**>  <**xs:element name="note"**>  <**xs:complexType**>  <**xs:sequence**>  <**xs:element name="name" type="xs:string"**/>  <**xs:element name="age" type="xs:int"**/>  <**xs:element name="sal" type="xs:double"**/>  </**xs:sequence**>  </**xs:complexType**>  </**xs:element**>  </**xs:schema**> |

3. 在note.xml文件中引入note.xsd

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**note  xmlns="**http://www.w3school.com.cn**"  xmlns:xsi="http://www**.w3.org/2001/XMLSchema-inst**ance"  xsi:schemaLocation="http://www.w3school.com.cn 02\_note.xsd"**>   <**name**>赵君</**name**>  <**age**>12</**age**>  <**sal**>1111.11</**sal**>  </**note**>  解释：  **xmlns="**http://www.w3school.com.cn**"**  规定了默认命名空间的声明。此声明会告知 schema 验证器，  在此 XML 文档中使用的所有元素都被声明于 "http://www.w3school.com.cn" 这个命名空间。  命名空间可以理解为装标签的包。  **xmlns:xsi="http://www**.w3.org/2001/XMLSchema-inst**ance"**  可用的 XML Schema 实例命名空间  **xsi:schemaLocation="http://www.w3school.com.cn 02\_note.xsd"**  您就可以使用 schemaLocation 属性了。此属性有两个值。  第一个值是需要使用的命名空间。  第二个值是供命名空间使用的 XML schema 的位置 |

|  |
| --- |
|  |

|  |
| --- |
|  |

# 第02章 Jsoup【重点】

XML编写和约束内容已经完成了，根据XML的作用我们了解到，无论是XML作为配置文件还是数据传输，我们的程序都要获取XML文档中的数据以便我们进行具体的业务操作，接下来我们就要学习XML解析技术Jsoup。

## 2.1 XML解析

### 2.1.1什么是XML解析

数据存储在XML后，我们就希望通过程序获取XML的内容。

如果我们使用Java基础所学的IO知识是可以完成的，不过非常繁琐的操作。

人们为不同问题提供不同的解析方式，使用不同的解析器进行解析，方便开发人员操作XML。

### 2.1.2解析方式和解析器

解析方式：市场上的解析方式有三种：

1. DOM：要求解析器把整个XML文档装载到内存，

Document Object Model

并解析成一个Document对象

优点：元素与元素之间保留结构关系，可以进行增删改查操作。

缺点：XML文档过大，可能出现内存溢出。

1. SAX：是一种速度更快，更有效的方法。它逐行扫描文档，

Simple Api for Xml

一边扫描一边解析。并以事件驱动的方式进行解析。

优点：处理速度快，可以处理过大XML文档。

缺点：只能读，逐行后将释放资源，解析操作繁琐。

1. PULL：Android内置的XML解析方式，类似SAX。

解析器：根据不同的解析方式提供具体实现。有的解析器操作过于繁琐，为了方便开发人员，有提供易于操作的解析开发包。

|  |
| --- |
|  |

解析开发包：

JAXP：sun公司提供支持DOM和SAX开发包

Dom4j：比较简单的的解析开发包

JDom：与Dom4j类似

JSoup：功能强大DOM方式的XML解析开发包，尤其对HTML解析更加方便。

## 2.2 Jsoup的概述

### 2.2.1 什么是Jsoup

Jsoup是一款基于Java 的HTML（HTML也是XML文档的一种）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。

### 2.2.2 Jsoup的应用场景

1. 解析XML文档并操作和管理XML元素、属性、元素体等数据。

2. 解析HTML页面并进行数据收集，例如：爬虫

## 2.3 Jsoup的基本使用

### 2.3.1 DOM解析原理及结构模型

1. 解析原理：

XML DOM 和 HTML DOM一样，XML DOM 将整个XML文档加载到内存，生成一个DOM树，并获得一个Document对象，通过Document对象就可以对DOM进行操作。

以下面books.xml文档为例：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**books**>  <**book id="it0001"**>  <**name**>JavaWeb开发教程</**name**>  <**author**>张孝祥</**author**>  <**sale**>100.00元</**sale**>  </**book**>  <**book id="it0002"**>  <**name**>三国演义</**name**>  <**author**>罗贯中</**author**>  <**sale**>100.00元</**sale**>  </**book**> </**books**> |

1. 结构模型：

|  |
| --- |
|  |

DOM中的核心概念就是节点，在XML文档中的元素、属性、文本，在DOM中都是节点！所有的节点都封装到了Document对象中。

1. 引入Jsoup的JAR包：

|  |
| --- |
|  |

由于DOM方式解析XML文档所有都是节点，所有节点又都被封装到Document对象中，所以解析的重点就是获取Document对象，之后在根据Document对象的相关方法就可以获取和操作XML文档数据，那么现在如何使用Jsoup获取Document对象呢？

### 2.3.2获取Document文档对象

#### 2.3.2.1根据XML文档字符串获取Document

API解释：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| */\*\*  \* 根据XML文档字符串获取Document  \*/* @Test **public void** test1(){  String str = **"<html><a href='#'>点点</a></html>"**;  Document document = Jsoup.*parse*(str);  System.***out***.println(document); } |

#### 2.3.2.2根据文件获取Document

API解释：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| */\*\*  \* 根据文件获取Document【重点】  \* 加载src/01\_book.xml  \*/* @Test **public void** test2() **throws** Exception{  String path = **this**.getClass().getResource(**"/01\_book.xml"**).getPath();  File file = **new** File(path);  Document document = Jsoup.*parse*(file, **"UTF-8"**);  System.***out***.println(document); } |

#### 2.3.2.3根据url获取Document

API解释：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| */\*\*  \* 根据url获取Document【重点】  \*/* @Test **public void** test3() **throws** Exception{  String url = **"https://www.tmall.com/"**;  Document document = Jsoup.*connect*(url).get();  System.***out***.println(document); } |

### 2.3.3 Document对象常用方法解析XML

API解释：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| **package** com.itheima.jsoup;  **import** org.jsoup.Jsoup; **import** org.jsoup.nodes.Document; **import** org.jsoup.nodes.Element; **import** org.jsoup.select.Elements; **import** org.junit.Before; **import** org.junit.Test; **import** java.io.File;  */\*\*  \* Document对象常用方法解析XML  \* 读取src/02\_index.html  \*/* **public class** Demo02 {   **private** Document **document**;   @Before  **public void** init() **throws** Exception{  String path = **this**.getClass().getResource(**"/02\_index.html"**).getPath();  File file = **new** File(path);  **document** = Jsoup.*parse*(file, **"UTF-8"**);  }   */\*\*  \* 需求1：获取index.html中所有h3标签名称的元素列表并打印输出  \* API：Elements elements = document.getElementsByTag("h3");  \* API：String nodeName = element.nodeName();  \* API：String html = element.html();  \* API：String text = element.text();  \*/* @Test  **public void** test1() **throws** Exception{  Elements elements = **document**.getElementsByTag(**"h3"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  System.***out***.println(e.html());  System.***out***.println(e.text());  }  }  }   */\*\*  \*需求2：获取index.html中所有元素含有class属性值为item并打印输出  \* API：Elements elements = document.getElementsByClass("item");  \*/* @Test  **public void** test2() **throws** Exception{  Elements elements = **document**.getElementsByClass(**"item"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求3：获取index.html中所有元素含有属性data‐toggle所有元素列表并打印输出  \* API：Elements elements = document.getElementsByAttribute("data-toggle");  \*/* @Test  **public void** test3() **throws** Exception{  Elements elements = **document**.getElementsByAttribute(**"data-toggle"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求4：获取index.html中元素属性id="footer"的一个元素并打印输出  \* API：Element e = document.getElementById("footer");  \*/* @Test  **public void** test4() **throws** Exception{  Element e = **document**.getElementById(**"footer"**);  System.***out***.println(e.nodeName());  }   */\*\*  \*需求5：获取index.html中元素属性id="footer"的一个元素的父元素标签名称并打印输出  \* API：Element ep = e.parent();  \*/* @Test  **public void** test5() **throws** Exception{  Element e = **document**.getElementById(**"footer"**);  Element ep = e.parent();  System.***out***.println(ep.nodeName());  }   */\*\*  \*需求6：获取index.html中元素属性id="footer"的元素的父元素的子元素列表的长度个数  \*/* @Test  **public void** test6() **throws** Exception{  Element e = **document**.getElementById(**"footer"**);  Element ep = e.parent();  Elements elements = ep.children();  **if**(elements!=**null** && elements.size()>0){  **for** (Element element : elements) {  System.***out***.println(element.nodeName());  }  }  }  } |

## 2.4 Jsoup选择器解析（针对HTML/XML）

### 2.4.1选择器解析介绍

选择器大家并不陌生，Jsoup提供的API方法使得我们可以类似jQuery、CSS选择器一样快速查找出想要的元素内容，接下来就要我们一起学习Jsoup选择器解析XML文档技术。

### 2.4.2选择器解析语法

#### 2.4.2.1 Document选择器方法API介绍

|  |
| --- |
|  |

选择器分为：基本选择器、组合选择器、伪类选择器。

Elements对象API方法介绍。

|  |
| --- |
|  |

#### 2.4.2.2基本选择器

语法：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| **package** com.itheima.jsoup;  **import** org.jsoup.Jsoup; **import** org.jsoup.nodes.Document; **import** org.jsoup.nodes.Element; **import** org.jsoup.select.Elements; **import** org.junit.Before; **import** org.junit.Test; **import** java.io.File;  */\*\*  \* Jsoup中的三种选择器解析-基本选择器  \* 读取src/02\_index.html  \*/* **public class** Demo03 {   **private** Document **document**;   @Before  **public void** init() **throws** Exception{  String path = **this**.getClass().getResource(**"/02\_index.html"**).getPath();  File file = **new** File(path);  **document** = Jsoup.*parse*(file, **"UTF-8"**);  }   */\*\*  \*需求1：获取id="footer"元素并输出元素名称  \*API：Elements elements = document.select("#footer");  \*/* @Test  **public void** test1() **throws** Exception{  Elements elements = **document**.select(**"#footer"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求2：获取index.html中所有元素含有class属性值为item并打印输出元素体内容  \* API：Elements elements = document.select(".item");  \*/* @Test  **public void** test2() **throws** Exception{  Elements elements = **document**.select(**".item"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求3：获取index.html中所有h3标签名称的元素列表并打印输出元素名称  \* API：Elements elements = document.select("h3");  \*/* @Test  **public void** test3() **throws** Exception{  Elements elements = **document**.select(**"h3"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求4：获取index.html中含有属性data‐toggle所有元素并打印输出元素名称和元素体数据  \* API：Elements elements = document.select("[data‐toggle]");  \*/* @Test  **public void** test4() **throws** Exception{  Elements elements = **document**.select(**"[data‐toggle]"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求5：获取index.html中含有属性以data属性名开头的所有元素列表并打印输出元素名称  \* API：Elements elements = document.select("[^data]");  \*/* @Test  **public void** test5() **throws** Exception{  Elements elements = **document**.select(**"[^data]"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求6：获取index.html中属性data‐toggle值为"abc"的所有元素列表并打印输出元素名称  \* API：Elements elements = document.select("[data‐toggle='abc']");  \*/* @Test  **public void** test6() **throws** Exception{  Elements elements = **document**.select(**"[data‐toggle='abc']"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求7：获取index.html中属性data‐toggle值以"a"开头前缀的所有元素列表并打印输出元素名称  \* API：  \*/* @Test  **public void** test7() **throws** Exception{  Elements elements = **document**.select(**"[data‐toggle^='a']"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }  } |

#### 2.4.2.3组合选择器

语法：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| **package** com.itheima.jsoup;  **import** org.jsoup.Jsoup; **import** org.jsoup.nodes.Document; **import** org.jsoup.nodes.Element; **import** org.jsoup.select.Elements; **import** org.junit.Before; **import** org.junit.Test; **import** java.io.File;  */\*\*  \* Jsoup中的三种选择器解析-组合选择器  \* 读取src/02\_index.html  \*/* **public class** Demo04 {   **private** Document **document**;   @Before  **public void** init() **throws** Exception{  String path = **this**.getClass().getResource(**"/02\_index.html"**).getPath();  File file = **new** File(path);  **document** = Jsoup.*parse*(file, **"UTF-8"**);  }   */\*\*  \*需求1：获取index.html中div元素含有class属性值为item并打印元素数据  \*API：Elements elements = document.select("div.item");  \*/* @Test  **public void** test1() **throws** Exception{  Elements elements = **document**.select(**"div.item"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求2：获取index.html中a元素含有属性href的所有元素列表并打印元素数据  \*API：Elements elements = document.select("a[href]");  \*/* @Test  **public void** test2() **throws** Exception{  Elements elements = **document**.select(**"a[href]"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求3：获取index.html中属性id值为"main"的所有div子孙元素并打印元素数据  \*API：Elements elements = document.select("#main div");  \*/* @Test  **public void** test3() **throws** Exception{  Elements elements = **document**.select(**"#main div"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求4：获取index.html中属性id值为"main"的直接div子元素并打印元素数据  \*API：Elements elements = document.select("#main>div");  \*/* @Test  **public void** test4() **throws** Exception{  Elements elements = **document**.select(**"#main>div"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求5：获取index.html中class="item"元素后面同级第一个div元素并打印元素数据  \*API：Elements elements = document.select(".item + div");  \*/* @Test  **public void** test5() **throws** Exception{  Elements elements = **document**.select(**".item + div"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求6：获取index.html中class="item"元素后面同级所有div元素并打印元素数据  \*API：Elements elements = document.select(".item ~ div");  \*/* @Test  **public void** test6() **throws** Exception{  Elements elements = **document**.select(**".item ~ div"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName()+**"#"**+e.text());  }  }  }  } |

#### 2.4.2.4伪类选择器

语法：

|  |
| --- |
|  |

示例代码：

|  |
| --- |
| **package** com.itheima.jsoup;  **import** org.jsoup.Jsoup; **import** org.jsoup.nodes.Document; **import** org.jsoup.nodes.Element; **import** org.jsoup.select.Elements; **import** org.junit.Before; **import** org.junit.Test; **import** java.io.File;  */\*\*  \* Jsoup中的三种选择器解析-伪类选择器  \* 读取src/02\_index.html  \*/* **public class** Demo05 {   **private** Document **document**;   @Before  **public void** init() **throws** Exception{  String path = **this**.getClass().getResource(**"/02\_index.html"**).getPath();  File file = **new** File(path);  **document** = Jsoup.*parse*(file, **"UTF-8"**);  }   */\*\*  \*需求1：属性class值为"item"的元素,并且该元素含有font元素并打印元素数据  \*API：Elements elements = document.select(".item(font)");  \*/* @Test  **public void** test1() **throws** Exception{  Elements elements = **document**.select(**".item:has(font)"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求2：获取index.html中ul中的li元素，并且排除当前元素class="gz"的元素并打印元素数据  \*API：Elements elements = document.select("ul li:not(.gz)");  \*/* @Test  **public void** test2() **throws** Exception{  Elements elements = **document**.select(**"ul li:not(.gz)"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求3：获取index.html中ul中的li元素，并且排除当前元素class="gz"的元素，并且索引号大于0，并打印元素数据  \*API：Elements elements = document.select("ul li:not(.gz):gt(0)");  \*/* @Test  **public void** test3() **throws** Exception{  Elements elements = **document**.select(**"ul li:not(.gz):gt(0)"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }   */\*\*  \*需求4：获取index.html中ul中的li元素，并且排除当前元素class="gz"的元素，并且索引号等于0，并打印元素数据  \*API：Elements elements = document.select("ul li:not(.gz):gt(0)");  \*/* @Test  **public void** test4() **throws** Exception{  Elements elements = **document**.select(**"ul li:not(.gz):eq(0)"**);  **if**(elements!=**null** && elements.size()>0){  **for** (Element e : elements) {  System.***out***.println(e.nodeName());  }  }  }  } |

## 2.5 JsoupXPath（针对HTML）

### 2.5.1介绍

JsoupXpath 是一款纯Java开发的使用XPath 解析HTML的解析器，XPath 语法分析与执行完全独立，html的DOM树生成借助Jsoup，故命名为JsoupXpath，所以JsoupXpath不是Jsoup的一部分，是在Jsoup基础上进行的扩展。

XPath 使用路径表达式来选取HTML文档中的元素节点或属性节点。节点是通过沿着路径来选取的。XPath在解析HTML文档方面提供了一独树一帜的路径思想。

### 2.5.2 XPath使用步骤

1. 导入JsoupXpath-0.3.2.jar，源码下载：<https://github.com/zhegexiaohuozi/JsoupXpath>。

2. 利用XPath语法完成选取HTML文档元素节点进行解析操作。

### 2.5.3 XPath语法

XPath表达式，就是用于选取HTML文档中节点的表达式字符串。

获取HTML文档节点元素一共有如下4种XPath语法方式：

1. 绝对路径表达式方式

2. 相对路径表达式方式

3. 全文搜索路径表达式方式

4. 谓语（条件筛选）方式

获取不同节点语法：

|  |
| --- |
|  |

JsoupXPath核心类JXDocument和执行Xpath表达式字符串的API方法介绍：

|  |
| --- |
|  |

JXNode类的API方法介绍：

|  |
| --- |
|  |

#### 2.5.3.1绝对路径表达式方式

语法：

|  |
| --- |
| String xpth = “/元素/子元素”  绝对路径是以“/”开头，一级一级描述标签的层级路径就是绝对路径，这里注意不可以跨层级。  绝对路径是从根元素开始写路径的，这里开头的“/”代表HTML文档根元素，所以在绝对路径中不可以写根元素路径。 |

#### 2.5.3.2相对路径表达式方式

语法：

|  |
| --- |
| String xpath1="子元素/子子元素...";//获取相对当前路径元素里面的子元素的选取  String xpath2="./子元素/子子元素";//"./"代表当前元素路径位置  String xpath3="/子元素/子子元素";//功能与xpath1格式功能一样  相对路径就是相对当前节点元素位置继续查找节点，需要使用JXNode.sel（xpath）进行执行相对路径表达式。  注意：  这里不可以使用“../”，JsoupXPath没有实现获取上一级节点功。 |

#### 2.5.3.3全文搜索路径表达式方式

语法：

|  |
| --- |
| String xpth = “//元素/子元素”  一个“/”符号，代表逐级写路径  2个“//”符号，不用逐级写路径，可以直接选取到对应的节点，是全文搜索匹配的不需要按照逐层级。 |

#### 2.5.3.4谓语（条件筛选）方式

语法：

|  |
| --- |
| 谓语，又称为条件筛选方式，就是根据条件过滤判断进行选取节点。  String xpath1="//元素[@attr1=value]";  //获取元素属性attr1=value的元素  //定位元素，且元素中有attr1属性，attr1属性值为value。 |

#### 2.5.3.5 XPath语法示例代码

|  |
| --- |
| **package** com.itheima.jsoupxpath;  **import** cn.wanghaomiao.xpath.model.JXDocument; **import** cn.wanghaomiao.xpath.model.JXNode; **import** org.jsoup.Jsoup; **import** org.jsoup.nodes.Document; **import** org.junit.Before; **import** org.junit.Test; **import** java.io.File; **import** java.util.List;  */\*\*  \* JsoupXpath的四种Xpath路径写法  \* 读取src/02\_index.html  \*/* **public class** Demo01 {   **private** JXDocument **jxDocument**;   @Before  **public void** init() **throws** Exception{  String path = **this**.getClass().getResource(**"/02\_index.html"**).getPath();  File file = **new** File(path);  Document document = Jsoup.*parse*(file, **"UTF-8"**);  **jxDocument** = **new** JXDocument(document);  }   */\*\*  \*绝对路径表达式方式  \*需求1：获取index.html文件中所有li标签之间的内容  \*API：List<JXNode> jxNodeList = jxDocument.selN("/body/ul/li");  \*/* @Test  **public void** test1() **throws** Exception{  List<JXNode> jxNodeList = **jxDocument**.selN(**"/body/ul/li"**);  **if**(jxNodeList!=**null** && jxNodeList.size()>0){  **for** (JXNode jxNode : jxNodeList) {  System.***out***.println(jxNode.getElement().nodeName());  }  }  }   */\*\*  \*相对路径表达式方式  \*需求2：获取index.html文件中所有li标签之间的内容  \*API：List<JXNode> jxNodeList = jxDocument.selN("body/ul/li");  \*/* @Test  **public void** test2() **throws** Exception{  List<JXNode> jxNodeList = **jxDocument**.selN(**"body/ul/li"**);  **if**(jxNodeList!=**null** && jxNodeList.size()>0){  **for** (JXNode jxNode : jxNodeList) {  System.***out***.println(jxNode.getElement().nodeName());  }  }  }   */\*\*  \*全文搜索路径表达式方式  \*需求3：获取index.html文件中所有li标签之间的内容  \*API：List<JXNode> jxNodeList = jxDocument.selN("//li");  \*/* @Test  **public void** test3() **throws** Exception{  List<JXNode> jxNodeList = **jxDocument**.selN(**"//li"**);  **if**(jxNodeList!=**null** && jxNodeList.size()>0){  **for** (JXNode jxNode : jxNodeList) {  System.***out***.println(jxNode.getElement().nodeName());  }  }  }   */\*\*  \*谓语（条件筛选）方式  \*需求4：获取index.html文件中li标签中class属性为gz的标签之间的内容  \*API：JXNode jxNode = jxDocument.selNOne("//li[@class='gz']");  \*/* @Test  **public void** test4() **throws** Exception{  JXNode jxNode = **jxDocument**.selNOne(**"//li[@class='gz']"**);  **if**(jxNode!=**null**){  System.***out***.println(jxNode.getElement().text());  }  }  } |

# 随堂测试